Р. С. ГУТЕР
Б. В. ОВЧИНСКИЙ
П. Т. РЕЗНИКОВСКИЙ

ПРОГРАММИРОВАНИЕ И ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Р. С. ГУТЕР, Б. В. ОВЧИНСКИЙ,
П. Т. РЕЗНИКОВСКИЙ

ПРОГРАММИРОВАНИЕ И ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА

Допущено
Министерством просвещения РСФСР
в качестве учебного пособия для школ
программистов-вычислителей

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

518
Г 97
УДК 519.95 (075.2)

АННОТАЦИЯ

Книга рассчитана в основном на учащихся школ, готовящих программистов-вычислителей. Она может быть использована также для самостоятельного овладения навыками программирования. В ней даны наиболее часто используемые простейшие численные методы (доведенные до составления соответствующих программ). Основное внимание уделено изложению приемов программирования. Книга соответствует программе 9—10 классов школ математического профиля.

ОГЛАВЛЕНИЕ

Предисловие 6

ЧАСТЬ I

ПРИЕМЫ РУЧНОГО СЧЕТА

Глава I. Вычисления по готовой формуле 9
§ 1. Элементарные действия. Способы записи чисел 9
§ 2. Таблицы. Линейная интерполяция. Таблицы пропорциональных частей 13
§ 3. Расписка формулы 17
§ 4. Контроль вычислений по готовым формулам 22
§ 5. Погрешности арифметических действий 24

Глава II. Средства вычислений 30
§ 6. Функциональные шкалы 30
§ 7. Логарифмическая линейка 33
§ 8. Арифмометры и клавишные машины 35
§ 9. Другие средства вычислений 36

ЧАСТЬ II

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Глава III. Арифметические основы программирования 38
§ 10. Системы счисления 38
§ 11. Двоичная арифметика 43
§ 12. Восьмеричная система счисления. Переход от одной системы к другой 48
§ 13. Смешанные системы счисления 51
§ 14. Формы представления чисел 56

Глава IV. Элементы программирования 59
§ 15. Основные устройства электронной счетной машины 59
§ 16. Команда в трехадресной машине 61
§ 17. Арифметические операции. Расписка формулы по командам 62
§ 18. Разветвляющиеся вычислительные процессы. Команды передачи управления 67
§ 19. Разветвляющиеся программы 71
§ 20. Арифметические циклы 80

§ 21. Итерационные циклы 91
§ 22. Цикл в цикле 96

Глава V. **Перевод программы на язык машины** 100
§ 23. Ячейка памяти. Представление команды в машине 100
§ 24. Кодирование программы 103
§ 25. Представление двоичных чисел в машине 110
§ 26. Представление десятичных чисел в машине 117
§ 27. Перфорация и ввод 119

Глава VI. **Переадресация** 125
§ 28. Действия над числами с фиксированной запятой 125
§ 29. Циклы с переадресацией. Восстановление переменных команд 134
§ 30. Двойные циклы с переадресацией 145
§ 31. Индексный регистр (регистр адреса) 152
§ 32. Операции с регистром адреса 155
§ 33. Применение регистра адреса в сложных циклах 169

Глава VII. **Операции над машинными словами** 177
§ 34. Машинное слово 177
§ 35. Сдвиги 177
§ 36. Первоначальные сведения из математической логики 184
§ 37. Логические операции машины 186
§ 38. Логические шкалы 191
§ 39. Программа перевода числа из десятичной системы в двоичную 195
§ 40. Программа перевода числа из двоичной системы в десятичную. Операция печати 200

Глава VIII. **Подпрограммы** 205
§ 41. Операция безусловной передачи управления с возвратом. Блоки и подпрограммы 205
§ 42. Стандартные подпрограммы с входными и выходными ячейками 210
§ 43. Стандартные подпрограммы с информацией. Формирование команд 214
§ 44. Применение регистра адреса в стандартных подпрограммах 219
§ 45. Библиотека стандартных подпрограмм 227

Глава IX. **Организация программы** 231
§ 46. Блочное программирование 231
§ 47. Блок-программы 237
§ 48. Работа с внешней памятью 243

Глава X. **Отладка программы** 251
§ 49. Подготовка программы к отладке 251
§ 50. Пульт машины 253
§ 51. Проверка работы программы на машине 256

Глава XI. **Общие сведения об электронных вычислительных машинах** 261
§ 52. Вычислительные машины непрерывного и дискретного действия 261
§ 53. Арифметические действия. Машины с плавающей и фиксированной запятой 264
§ 54. Одноадресные и двухадресные машины 264

ЧАСТЬ III

МЕТОДЫ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Глава XII. **Численное решение алгебраических и трансцендентных уравнений** 267

§ 55. Подбор корней 267
§ 56. Способ хорд и проведение параболы 270
§ 57. Способ касательных. Комбинированный способ 273
§ 58. Способ итераций 279
§ 59. Алгебраические уравнения. Схема Горнера 285
§ 60. Программирование подбора корней 287
§ 61. Программа для способа хорд и касательных 291
§ 62. Программирование итерационного процесса 294

Глава XIII. **Системы уравнений** 296

§ 63. Некоторые сведения о векторах и матрицах 296
§ 64. Определители 301
§ 65. Решение системы линейных уравнений по способу Гаусса . . 323
§ 66. Применение схемы Гаусса для вычисления определителя и нахождения обратной матрицы 330
§ 67. Итерации для линейных систем 337
§ 68. Способ Зейделя 344
§ 69. Способ итераций для нелинейных систем уравнений 348
§ 70. Программа решения системы линейных уравнений по способу Гаусса 350
§ 71. Программирование итерационного процесса для систем линейных уравнений 361

Глава XIV. **Интерполяция и экстраполяция** 366

§ 72. Общая постановка задачи интерполяции 366
§ 73. Табличные разности и их свойства 368
§ 74. Точность линейной интерполяции. Квадратичная интерполяция по схеме Эйткина 383
§ 75. Интерполяционные формулы Лагранжа и Ньютона 386
§ 76. Экстраполяция. Обратная интерполяция 395
§ 77. Программирование прямой и обратной интерполяции 398
§ 78. Программа работы с табличной функцией 402

Глава XV. **Численное интегрирование** 407

§ 79. Формулы прямоугольников и трапеций 407
§ 80. Формула Симпсона 411
§ 81. Проверка точности результатов численного интегрирования 415
§ 82. Программирование формулы Симпсона 419

Глава XVI. **Вычисление элементарных функций** 423

§ 83. Общие замечания. Вычисление многочленов 423
§ 84. Применение степенных рядов 425
§ 85. Цепные дроби 429
§ 86. Программы вычисления элементарных функций с помощью рядов 435
§ 87. Программы вычисления элементарных функций с помощью цепных дробей 443

ПРЕДИСЛОВИЕ

Свою книгу мы писали как учебник для старших классов средних школ с математической специализацией. Она написана по инициативе лаборатории при секторе обучения математике Института общего и политехнического образования Академии педагогических наук РСФСР и соответствует программе курса «Программирование и вычислительная математика» в 9—10 классах таких школ.

Первые две части книги (Приемы ручного счета и Основы программирования) не требует от читателя никакой дополнительной математической подготовки, выходящей за пределы программы курса математики восьмилетней школы.

Исключение составляет § 5 гл. I, посвященный погрешностям арифметических действий. Его изучение предполагает знание основных понятий дифференциального исчисления. Поэтому можно рекомендовать ограничиться на первых порах лишь утверждениями, перенеся рассмотрение их доказательств на конец курса.

Вторая часть (Основы программирования), по-видимому, наиболее простая для изучения, представляла для авторов наибольшие трудности. Среди многочисленных причин этих трудностей наибольшее значение имеют две: различие машин и многообразие точек зрения и методов программирования.

Мы сочли целесообразным не касаться в книге вопросов использования алгоритмических языков программирования, полагая, что «ручное программирование» при всех обстоятельствах должно быть известно каждому программисту. На базе усвоенного материала нашей книги овладение алгоритмическим языком, в случае необходимости, уже не будет представлять никаких затруднений.

Из различных методов программирования мы без всяких колебаний выбрали метод программирования в содержательных обозначениях, автором которого является А. Л. Брудно. Если при фактическом программировании для некоторой конкретной машины можно еще писать программы в кодах и действительных адресах, то для преподавания преимущества программирования в содержательных обозначениях настолько очевидны и неоспоримы, что нет никакой нужды на этом останавливаться. Их подтвердили и Академия педагогических

наук и Министерство просвещения РСФСР, рекомендовавшие программирование в содержательных обозначениях в качестве основного метода программирования для изучения в школах программистов-вычислителей.

Чтобы учебник могли использовать ученики, работающие на различных машинах, мы рассматриваем программирование для условной трехадресной машины. Система команд этой условной машины выбиралась таким образом, чтобы она была возможно ближе к имеющимся у нас различным трехадресным машинам. В ряде случаев указывается отличие тех или иных операций различных конкретных машин от рассматривающейся в книге условной машины. Некоторые специальные операции, облегчающие программирование, но различные у разных конкретных машин, мы не рассматривали совсем.

На некоторых электронных вычислительных машинах (например, на одноадресных машинах типа «Урал») применяется индексный регистр, который называют также счетчиком циклов или регистром адреса. Поэтому мы включили в свою книгу описание работы с индексным регистром.

Читателям, желающим глубже ознакомиться с блочным программированием, использованием библиотеки стандартных программ и более сложными вопросами программирования, можно рекомендовать выходящую вскоре в издательстве «Наука» книгу автора программирования в содержательных обозначениях А. Л. Брудно «Введение в программирование», а также вышедший недавно справочник В. Л. Арлазарова, Р. С. Гутера и А. В. Ускова «Практика программирования».

Третья часть книги посвящена основным методам вычислительной математики. Она требует знакомства с основными понятиями дифференциального исчисления, а глава XV — и с понятием определенного интеграла. Программирование в содержательных обозначениях позволило нам не только изложить математическую сторону вычислительных методов и привести расчетные схемы для ручных расчетов, чем до сих пор обычно ограничивались, но и довести изложение этих методов до написания соответствующих программ. Подобное изложение методов вычислительной математики, насколько нам известно, делается чуть ли не впервые в нашей литературе. Поэтому мы полагаем, что соответствующие главы и параграфы книги будут интересны не только для школьников.

Часть материала книги набрана петитом. Это — вопросы, которые показались нам более трудными или не безусловно необходимыми для всех учащихся, но представляющие интерес для более глубокого изучения предмета. Поэтому при первом чтении места, набранные петитом, могут быть пропущены.

Весь материал книги использовался авторами в практике преподавания средней школы (школы № 7 и № 444 г. Москвы) и втузов. Ряд примеров программ взят из уже упоминавшегося справочника по

программированию В. Л. Арлазарова, Р. С. Гутера и А. В. Ускова. Некоторые числовые примеры третьей части заимствованы из ранее вышедшей книги Р. С. Гутера и Б. В. Овчинского «Элементы численного анализа и математической обработки результатов опыта» (Физматгиз, 1962 г.). При рассмотрении определителей мы воспользовались определением, приведенным Г. Е. Шиловым в его книге «Введение в теорию линейных пространств» (Гостехиздат, 1952).

Отметим еще, что П. Т. Резниковский не принимал участия в написании первой части книги, а Б. В. Овчинский — в написании второй части.

Для удобства пользования книгой мы провели сквозную нумерацию глав и параграфов. Сквозная нумерация формул и примеров показалась нам менее удачной. По этой причине формулы и примеры нумеруются в каждом параграфе заново. Однако для облегчения различных ссылок *после* номера каждой формулы или примера указывается параграф, в котором они находятся, так что, например, номер (3.75) означает формулу (3) из § 75.

План нашей книги сложился в значительной степени под влиянием советов А. С. Кронрода и бесед с ним. Л. А. Люстерник и С. И. Шварцбурд ознакомились с рукописью, а В. Л. Арлазаров и Ф. М. Филлер — с отдельными ее частями и высказали нам свои замечания.

Большую помощь в работе над книгой нам оказала Т. А. Муратова. Ею выполнена вся кодировка, почти все вычисления и большая техническая работа по оформлению рукописи. Кроме этого, она высказала очень полезные замечания по первой части книги. Ряд редакционных замечаний и поправок был сделан редактором книги Ю. П. Оревковым и Р. А. Махмуд-заде. Всем названным лицам авторы выражают свою глубокую признательность.

ЧАСТЬ ПЕРВАЯ

ПРИЕМЫ РУЧНОГО СЧЕТА

ГЛАВА I

ВЫЧИСЛЕНИЯ ПО ГОТОВОЙ ФОРМУЛЕ

§ 1. Элементарные действия. Способы записи чисел

Почти все числа, с которыми приходится иметь дело в процессе вычислений, являются *приближенными*. Действительно, исходные числовые данные обычно выражают значения тех или иных физических величин, полученные в результате измерений, которые, по самому существу своему, носят приближенный характер. Далее, действительные числа, хотя бы и рациональные, чаще всего не могут быть точно записаны в десятичной системе счисления. Их приходится записывать в виде конечной десятичной дроби, обрывая ее в том или ином месте. Наконец, арифметические действия в большинстве случаев тоже не могут быть выполнены точно.

В этих условиях серьезную роль играет вопрос о точности получаемых результатов и размерах возможных погрешностей. Рассмотрение его следует начать с знакомства с терминологией.

Пусть A — некоторая величина, истинное значение которой известно или неизвестно. Число a, которое можно принять за значение величины A, мы будем называть *приближенным значением величины* A или просто *приближенным числом*. Число a называют *приближением по недостатку*, если оно меньше истинного значения, и *по избытку*, если оно больше. Так, 3,14 является приближенным значением числа π по недостатку, а 2,72 — приближением числа e по избытку. Чтобы охарактеризовать степень точности данного приближения, пользуются понятием *погрешности* или *ошибки*.

Абсолютная погрешность или *абсолютная ошибка* приближенного числа есть абсолютная величина разности между истинным значением величины и данным ее приближенным значением. Чаще всего, когда речь идет об арифметических действиях с приближенными числами, употребляется термин *погрешность*, а когда говорят об измерениях, то употребляют термин *ошибка*. Нужно еще иметь в виду, что в обыденной речи этот термин употребляется совсем в ином смысле. Именно, в обыденной речи под о ш и б к о й пони-

мается **неверный** результат. В вычислительной математике в этом смысле употребляется слово *просчет*.

Поскольку истинное значение величины обычно остается неизвестным, неизвестной остается также и абсолютная погрешность. Вместо нее приходится рассматривать *предельную абсолютную погрешность*, которая означает число, не меньшее абсолютной погрешности *).

Приближенные числа принято записывать таким образом, чтобы вид числа показывал его абсолютную погрешность. Исходные данные, получающиеся в результате измерений, а также числа в математических таблицах записывают так, чтобы абсолютная погрешность не превосходила половины единицы последнего разряда, сохраняемого при записи. Например, запись 3,1416 означает, что абсолютная погрешность этого приближенного числа не превосходит 0,00005. Для числа 2,150 абсолютная погрешность не превосходит 0,0005, а для числа 280 — величины 0,5. Если это число имеет бо́льшую точность, например, если абсолютная погрешность меньше 0,05, то следует писать не 280, а 280,0. Таким образом, при указанной форме записи числа все записанные цифры оказываются **верными**.

Если в результате действий получается большее, чем нужно, количество цифр, то число следует округлять, отбрасывая излишние цифры. При этом руководствуются известным правилом округления: если первая из отброшенных цифр 4 или меньше, то последняя цифра сохраняется без изменения; если первая из отброшенных цифр 5 или больше, то последняя оставшаяся цифра увеличивается на единицу.

Исключением из этого правила является случай, когда отбрасывается только пятерка или же пятерка с нулями. Здесь можно выбрать любое правило округления, например, округлять всегда вверх или, наоборот, всегда вниз. Мы будем пользоваться правилом, предложенным К. Гауссом: последняя оставшаяся цифра сохраняется без изменения, если она четная, и увеличивается на единицу до четной, если она была нечетная. По этому правилу при округлении до целого числа 2,5 округляется до 2, а 3,5 — до 4.

Точность данного приближенного числа не характеризуется его абсолютной погрешностью. Действительно, погрешность 0,5 *м* слишком велика при измерении длины комнаты, допустима при измерении участка, отводимого для постройки дома, и не может быть замечена при измерении расстояния между городами. Настоящим показателем точности результата измерения или вычисления является его относительная погрешность.

Относительной погрешностью приближенного значения величины называют абсолютную величину отношения его абсолютной погреш-

*) Иногда, ради краткости речи, слово «предельная» опускают, понимая под абсолютной погрешностью именно предельную.

ности к истинному значению этой величины. Часто эту относительную погрешность выражают в процентах.

Ввиду того, что фактически вместо абсолютной погрешности приходится рассматривать предельную, относительную погрешность также заменяют предельной относительной погрешностью, которая означает число, не меньшее относительной погрешности. Более того, при нахождении предельной относительной погрешности приходится заменять неизвестное истинное значение величины приближенным. Последняя замена обычно не отражается на величине относительной погрешности ввиду близости этих значений и малости абсолютной погрешности.

Например, для приближенного значения $\pi = 3{,}14$ предельная абсолютная погрешность составляет 0,0016, а относительная 0,00051, или $0{,}05\%$.

Для больших чисел даже и при малых относительных погрешностях абсолютные погрешности могут иметь порядок единиц, десятков, сотен и т. п. Например, для числа 234000 может не быть данных, указывающих на то, являются ли здесь нули действительно цифрами данного числа или, как принято говорить в таких случаях, замещают неизвестные цифры. Подчиняясь сформулированному ранее правилу, мы имеем право записать это число в виде 234000 только тогда, когда его абсолютная погрешность не превосходит 0,5. Если же абсолютная погрешность не превосходит пятидесяти, то последние два нуля замещают неизвестные цифры. Поэтому запись 234000 в этом случае является незаконной; число следует записывать в виде $2340 \cdot 10^2$ или $0{,}2340 \cdot 10^6$.

Абсолютная погрешность приближенного числа определяется числом десятичных знаков в его записи. Относительная погрешность определяется количеством *значащих цифр* в числе. При этом значащими цифрами являются все цифры числа, кроме тех нулей, которые употребляются для определения места других цифр в десятичной дроби или же замещают неизвестные цифры. Например, в числе 0,20003 значащими являются все цифры, кроме нуля целых. В числе 0,040 значащей является цифра 4, а также и нуль после нее, если число записано с соблюдением приведенного ранее правила и его абсолютная погрешность не превышает 0,0005. Два нуля впереди цифры 4 не являются значащими цифрами, так как они только определяют положение цифры 4. Можно не писать этих нулей, а записывать число в виде $40 \cdot 10^{-3}$, $4{,}0 \cdot 10^{-2}$ или же $0{,}40 \cdot 10^{-1}$.

Пользуясь термином «значащая цифра», мы можем упомянутое выше правило о записи приближенных чисел формулировать таким образом: *приближенные числа следует записывать так, чтобы все цифры числа, кроме нулей впереди, если они есть, были значащими и верными цифрами.*

О числах $0{,}40 \cdot 10^{-1}$ и $0{,}2340 \cdot 10^6$ говорят, что они записаны в *нормальной* или *плавающей форме* (*с плавающей запятой*), в про-

тивоположность обычной записи, которую называют *естественной* или *фиксированной (с фиксированной запятой)*. Как видно из приведенных примеров, запись числа в плавающей форме не определяется однозначно. Чтобы устранить неоднозначность, принято первый множитель брать меньшим единицы и состоящим только из значащих цифр (кроме нуля целых), так что его первая цифра отлична от нуля. В этом случае число называют *нормализованным*.

Таким образом, запись числа N в плавающей форме означает его представление в виде

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

где $|N_0| < 1$. Число N_0 называют *мантиссой* числа N, а число p — его *порядком*. Если число N нормализовано, то первая цифра мантиссы отлична от нуля, т. е.

$$1 > |N_0| \geqslant 0{,}1$$

и порядок p определен однозначно. Легко сообразить, что порядок числа, большего единицы, равен количеству целых разрядов числа до запятой, а порядок числа, меньшего единицы, отрицателен (либо равен нулю), и его абсолютная величина показывает количество нулей после запятой перед первой значащей цифрой числа.

При записи отрицательного числа в плавающей форме отрицательной считают его мантиссу.

Обычно в процессе вычислений стремятся иметь все числа с одинаковой точностью. Записывая все числа в фиксированной форме с одним и тем же количеством десятичных знаков, мы можем для всех чисел обеспечить одинаковую абсолютную погрешность. Запись всех чисел в плавающей форме с одинаковым числом знаков в мантиссе позволяет обеспечить для всех одинаковую относительную погрешность. В зависимости от потребностей вычислений и употребляется та или иная форма записи.

Фиксированная форма более удобна для сложения и вычитания, чем плавающая, но переводить числа для сложения или вычитания в фиксированную форму не обязательно. Если слагаемые имеют одинаковый порядок, то сложение, как и вычитание, производится по обычным правилам и порядок суммы или разности остается равным порядку слагаемых. При этом надо только иметь в виду, что результат может не оказаться нормализованным и для его нормализации нужно сдвинуть мантиссу в ту или иную сторону, изменив соответствующим образом порядок.

Легко понять, как это делается, рассмотрев следующие примеры:

$$0{,}347541 \cdot 10^3 + 0{,}532612 \cdot 10^3 = 0{,}880153 \cdot 10^3,$$
$$0{,}791564 \cdot 10^{-1} - 0{,}272316 \cdot 10^{-1} = 0{,}519248 \cdot 10^{-1},$$
$$0{,}631825 \cdot 10^{-1} + 0{,}723007 \cdot 10^{-1} = 1{,}354832 \cdot 10^{-1} = 0{,}135483\,(\cdot 10^0),$$
$$0{,}568188 \cdot 10^3 - 0{,}489313 \cdot 10^3 = 0{,}078875 \cdot 10^3 = 0{,}78875 \cdot 10^2,$$
$$0{,}713954 \cdot 10^4 - 0{,}711259 \cdot 10^4 = 0{,}002695 \cdot 10^4 = 0{,}2695 \cdot 10^3.$$

В первых двух случаях результат получился нормализованным. В трех следующих потребовалась последующая нормализация с изменением порядка.

Если слагаемые имеют различные порядки, то, прежде чем производить сложение или вычитание, необходимо выровнять порядки, сдвинув соответствующим образом мантиссу. Порядок выравнивается по большему числу, так что у меньшего числа, которое требуется «разнормализовать», мантисса сдвигается вправо. Обычно ее при этом округляют, как это видно из приведенных ниже примеров:

$$0{,}564137 \cdot 10^4 + 0{,}253912 \cdot 10^3 = 0{,}564137 \cdot 10^4 + 0{,}025391 \cdot 10^4 = 0{,}589528 \cdot 10^4,$$

$$0{,}876839 \cdot 10^{-4} + 0{,}272731 \cdot 10^{-1} = 0{,}000877 \cdot 10^{-1} + 0{,}272731 \cdot 10^{-1} = 0{,}273608 \cdot 10^{-1},$$

$$0{,}643995 \cdot 10^{-1} - 0{,}987414 \cdot 10^{-3} = 0{,}643995 \cdot 10^{-1} - 0{,}009874 \cdot 10^{-1} = 0{,}634121 \cdot 10^{-1}.$$

При умножении или делении чисел в плавающей форме порядки могут быть какими угодно. При умножении мантиссы сомножителей перемножаются по обычным правилам, а порядки складываются. Точно так же при делении мантиссы делятся, а порядки вычитаются. Не надо только забывать, что в результате действий могут получиться ненормализованные числа и может понадобиться сдвиг мантиссы для нормализации результата.

Примеры:

$$0{,}235642 \cdot 10^2 \times 0{,}853165 \cdot 10^{-1} = 0{,}201042 \cdot 10^1,$$

$$0{,}312317 \cdot 10^{-1} \times 0{,}292877 \cdot 10^{-2} = 0{,}0914705 \cdot 10^{-3} = 0{,}914705 \cdot 10^{-4},$$

$$0{,}156341 \cdot 10^3 : 0{,}592658 \cdot 10^2 = 0{,}263796 \cdot 10^1,$$

$$0{,}828145 \cdot 10^{-2} : 0{,}327140 \cdot 10^{-1} = 2{,}53147 \cdot 10^{-1} = 0{,}253147 \,(\cdot 10^0).$$

§ 2. Таблицы. Линейная интерполяция. Таблицы пропорциональных частей

Всевозможные таблицы функций являются одним из самых распространенных вспомогательных средств вычислений. Наиболее употребительными являются таблицы квадратов и кубов, квадратных и кубических корней, обратных величин, тригонометрических функций, логарифмов, показательной и других элементарных функций. Часто встречается необходимость и в разного рода таблицах специальных функций *).

*) К элементарным функциям относятся степенные, различного рода многочлены и алгебраические функции, логарифмическая, показательная, тригонометрические функции и обратные им, а также различные комбинации перечисленных функций. Остальные табулированные функции принято называть специальными.

Почти во всех случаях значения аргумента, которые приводятся в таблице, образуют арифметическую прогрессию; ее разность называют *шагом* таблицы. В некоторых таблицах для различных участков изменения аргумента выбирается различный шаг.

Значения функции обычно приводятся с одной и той же предельной абсолютной погрешностью, т. е. с фиксированным числом десятичных знаков. Реже встречаются таблицы, в которых применяется принцип постоянства относительной погрешности, т. е. дается определенное число значащих цифр.

Для практического ручного счета чаще всего бывает достаточно четырехзначных или пятизначных таблиц. Наиболее удобными и полными, а также и наиболее распространенными являются «Пятизначные математические таблицы» Б. И. Сегала и К. А. Семендяева, «Четырехзначные математические таблицы» Милн-Томсона и Комри, а также четырехзначные таблицы из «Справочника по математике» И. Н. Бронштейна и К. А. Семендяева. К ним следует присоединить также носящие более узкий характер и содержащие различного рода специальные функции, но также весьма употребительные четырехзначные «Таблицы функций с формулами и кривыми» Е. Янке и Ф. Эмде. Можно упомянуть также о четырехзначных таблицах В. М. Брадиса, носящих больше учебный характер и мало приспособленных для практической вычислительной работы.

Контрольные ручные расчеты для отладки программ для электронных счетных машин ведутся уже с значительно большей точностью. Как правило, они бывают восьмизначными, а иногда и с большим числом знаков. Среди таких многозначных таблиц нужно прежде всего указать «Таблицы Барлоу», содержащие квадраты, кубы, квадратные и кубические корни, а также обратные величины. Там, где нельзя дать точных значений, эти таблицы являются восьмизначными. Существует также большое количество многозначных таблиц элементарных и специальных функций.

Основной задачей, которая возникает при пользовании таблицами, является нахождение значений функции для тех значений аргумента, которые находятся между имеющимися в таблице. Эту задачу называют задачей *интерполяции*. Более подробно интерполяция функций будет рассматриваться в гл. XIV. Здесь мы рассмотрим лишь наиболее простой и наиболее широко распространенный способ — *линейную интерполяцию*.

Идея линейной интерполяции состоит в том, что функцию предполагают изменяющейся между двумя имеющимися в таблице значениями аргумента *линейно*. Иначе говоря, приращение функции на участке между двумя табличными значениями аргумента принимается пропорциональным приращению аргумента.

Нетрудно вывести общую формулу для линейной интерполяции. Пусть требуется найти значение функции в точке x, а x_0, x_1 — два

табличных значения аргумента, в которых функция равна соответственно y_0 и y_1. Тогда на участке длины $x_1 - x_0$ функция изменяется на $y_1 - y_0$. На единицу длины приходится приращение $\frac{y_1 - y_0}{x_1 - x_0}$, а на участок длины $x - x_0$ — от точки x_0 до точки x — приращение $\frac{y_1 - y_0}{x_1 - x_0}(x - x_0)$. Таким образом, в точке x функцию следует принять равной

$$y = y_0 + \frac{y_1 - y_0}{x_1 - x_0}(x - x_0). \qquad (1.2)$$

По большей части приращение функции $y_1 - y_0$ невелико и его считают в целых единицах последнего знака таблицы функции. Интерполяция обычно производится на один, следующий за имеющимся в таблице, десятичный знак, и поэтому разности $x_1 - x_0$ и $x - x_0$ также считают в целых единицах этого знака *), причем разность $x_1 - x_0$ в этом случае равна 10.

Пример 1.2. По табличным данным для функции $y = e^x$

$$x_0 = 0{,}83, \quad y_0 = 2{,}2933,$$

$$x_1 = 0{,}84, \quad y_1 = 2{,}3164.$$

Найдем значение функции, соответствующее $x = 0{,}833$.

Приращение функции равно здесь 231 единице четвертого десятичного знака функции на 10 единиц третьего знака аргумента. Отсюда сразу получается, что на единицу третьего знака приходится 23, а на три единицы — 69 единиц четвертого знака функции. Таким образом, значение функции в точке $x = 0{,}833$ следует принять равным $2{,}2933 + 0{,}0069 = 2{,}3002$. Сравнение с более подробными таблицами показывает, что все полученные знаки верны.

Возможные погрешности интерполяции уменьшаются, если считать приращение на единицу следующего знака аргумента с одним запасным знаком, т. е. в нашем примере брать приращение на единицу третьего знака аргумента равным 23,1. Например, для значения $x = 0{,}837$ приращение следует считать равным $23{,}1 \cdot 7 = 162$ единицы четвертого знака функции (результат умножения округлен!), так что функцию следует принимать равной 2,3095.

Для облегчения интерполяции в таблицах часто печатаются разности между соседними значениями функции (*табличные разности*). Поскольку деление на 10 (или на 100, если аргумент имеет два лишних десятичных знака) достигается помещением запятой в соответствующем месте разности, остается только умножить полученное число на нужный множитель.

*) Мы принимаем для простоты, что шаг таблицы, как это чаще всего бывает, равен 10^{-p}, т. е. единице какого-либо десятичного знака.

Впрочем, результаты умножения также можно заготовить заранее. Это сделано в специальной таблице, которую называют *таблицей пропорциональных частей*. Эта таблица печатается обычно на отдельном листе, который вкладывается в книгу. Она может быть использована для линейной интерполяции в любой таблице функций.

Пример 2.2. Таблица 1.2 содержит часть такой таблицы для значений разностей от 75 до 82.

Таблица 1.2

	75	76	77	78	79	80	81	82
1	7.5	7.6	7.7	7.8	7.9	8.0	8.1	8.2
2	15.0	15.2	15.4	15.6	15.8	16.0	16.2	16.4
3	22.5	22.8	23.1	23.4	23.7	24.0	24.3	24.6
4	30.0	30.4	30.8	31.2	31.6	32.0	32.4	32.8
5	37.5	38.0	38.5	39.0	39.5	40.0	40.5	41.0
6	45.0	45.6	46.2	46.8	47.4	48.0	48.6	49.2
7	52.5	53.2	53.9	54.6	55.3	56.0	56.7	57.4
8	60.0	60.8	61.6	62.4	63.2	64.0	64.8	65.6
9	67.5	68.4	69.3	70.2	71.1	72.0	72.9	73.8

Если, например, табличная разность равна 78 и аргумент имеет один лишний десятичный знак 6, то значение функции требуется увеличить на 47 единиц последнего знака. Если мы будем иметь два лишних десятичных знака, то, например, для аргумента $x=0{,}3527$ значение функции в точке 0,35 нужно увеличить на $15{,}6+5{,}5=21$ (округлено!) единиц последнего знака (при той же разности 78).

Часто бывает выгодно вычислить непосредственно искомое значение функции, а не приращение, которое нужно прибавлять к табличному значению функции, как это делалось выше. Соответствующую формулу легко получить, исходя из уравнения прямой, проходящей через две данные точки, или путем алгебраического преобразования приведенной выше общей формулы (1.2).

Наиболее удобной формулой для получения значения функции является формула, полученная по так называемой *схеме Эйткина*. Пусть даны два значения аргумента x_0 и x_1, в которых функция принимает соответственно значения y_0 и y_1. Значение функции в точке x может быть получено по формуле *)

$$P_{0,1}(x)=\frac{1}{x_1-x_0}\begin{vmatrix} y_0 & x_0-x \\ y_1 & x_1-x \end{vmatrix}. \qquad (2.2)$$

*) Здесь мы пользуемся выражением, которое называют *определителем второго порядка*; оно представляет сокращенную запись для разности

Действительно, выражение $P_{0,1}(x)$ представляет собой многочлен первой степени относительно x, т. е. изменение функции на рассматриваемом участке предполагается линейным. Остается убедиться в том, что многочлен $P_{0,1}(x)$ принимает в заданных точках нужные значения. Чтобы это сделать, вычислим значения $P_{0,1}$ в точках x_0 и x_1:

$$P_{0,1}(x_0)=\frac{1}{x_1-x_0}\begin{vmatrix} y_0 & 0 \\ y_1 & x_1-x_0 \end{vmatrix}=y_0,$$

$$P_{0,1}(x_1)=\frac{1}{x_1-x_0}\begin{vmatrix} y_0 & x_0-x_1 \\ y_1 & 0 \end{vmatrix}=y_1.$$

Итак, функция $P_{0,1}(x)$ действительно является линейной функцией, совпадающей с заданной функцией в точках x_0 и x_1.

Схема Эйткина очень удобна для вычислений и дает возможность сразу получить требуемое значение функции.

Пример 3.2. По значениям функции $y=\sin x$ с аргументом, заданным в радианной мере,

$$x_0=0{,}40, \qquad y_0=0{,}3894,$$
$$x_1=0{,}42, \qquad y_1=0{,}4078,$$

найдем с помощью схемы Эйткина значение синуса для $x=0{,}411$.

Пользуясь формулой (2.2), находим

$$P_{0,1}=\frac{1}{0{,}42-0{,}40}\cdot\begin{vmatrix} 0{,}3894 & -0{,}011 \\ 0{,}4078 & 0{,}009 \end{vmatrix}=\frac{0{,}3894\cdot 0{,}009+0{,}4078\cdot 0{,}011}{0{,}02}=$$
$$=0{,}3995.$$

§ 3. Расписка формулы

Основным, если не единственным, видом ручных расчетных работ является вычисление по готовой формуле. Вычисления по одной и той же формуле могут производиться в сравнительно большом числе точек либо в одной-двух точках, если речь идет о контрольном расчете при проверке программы, составленной для электронной счетной машины.

Прежде чем приступать к непосредственным вычислениям, нужно продумать схему их выполнения и составить расчетную таблицу, в

произведений. Именно,

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}=a_1b_2-a_2b_1,$$

так что приведенный нами определитель второго порядка равен

$$\begin{vmatrix} y_0 & x_0-x \\ y_1 & x_1-x \end{vmatrix}=y_0(x_1-x)-y_1(x_0-x).$$

Подробнее об определителях речь будет идти в гл. XIII (см. § 64).

которой и должны выполняться все вычисления. Основное правило, о котором никогда нельзя забывать, состоит в следующем:

вычислять и записывать вычисления следует так, чтобы любой результат можно было всегда проверить.

Форма расчетной таблицы определяется видом заданной функции. В каждом ее столбце должны выполняться одни и те же действия. Столбцы нужно стараться располагать в таком порядке, чтобы результаты, полученные в одном из них, использовались при вычислении значений следующего. В заголовке каждого столбца, соответствующего промежуточным результатам, указывается, каким образом числа из этого столбца получаются через предыдущие. Если какой-либо промежуточный результат имеет свое обозначение, то это обозначение тоже нужно указать в заголовке.

Кроме наименования результатов, находящихся в данном столбце, в заголовке столбца следует стараться избегать употребления буквенных обозначений. Если речь идет о величинах, фигурировавших в предыдущих столбцах данной таблицы, то необходимо указывать не букву, а номер столбца. Если же речь идет о постоянной величине, ранее в таблице не встречавшейся и являющейся, например, слагаемым или множителем, то лучше в заголовке столбца помещать соответствующее числовое значение.

Форма расчетной таблицы зависит и от того, какими вычислительными средствами мы раснолагаем. Например, если пользоваться автоматической клавишной машиной, то некоторые промежуточные результаты можно не записывать и объединять несколько столбцов в один. Иногда бывает удобно воспользоваться какими-либо имеющимися таблицами, например, выписывать значения квадратов или квадратных корней из таблицы, вместо того чтобы производить соответствующие вычисления на машине.

Составленную расчетную таблицу обычно называют *распиской формулы*. При вычислениях по расписке рекомендуется производить вычисления не по строке, а заполнять последовательно столбец за столбцом. Это, во-первых, ускоряет вычисления, так как при заполнении каждого столбца требуется выполнение однотипных операций, во-вторых, это облегчает контроль вычислений: благодаря монотонности изменения результатов, которая обычно имеет место, отдельные просчеты могут быть легче обнаружены.

Пример 1.3. Составим расписку для вычисления величины y по формуле

$$y = \frac{4x}{3 - x^2} - \sqrt{2x}.$$

Требуемая расписка приведена в табл. 1.3.

Так как в заголовках столбцов наряду с номерами предыдущих столбцов встречаются и числа, то они стоят в кавычках. Иногда их записывают также с индексом N (число).

Таблица 1.3

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
x	«2» · (1)	$\sqrt{(2)}$	(1)^2	«3»—(4)	«2» · (2)	(6) : (5)	(7)—(3)

Пример 2.3. Зависимость от времени скорости тела, падающего без начальной скорости в воздухе, с учетом сопротивления воздуха выражается формулой

$$v = v_{кр} \frac{e^{\frac{gt}{v_{кр}}} - e^{-\frac{gt}{v_{кр}}}}{e^{\frac{gt}{v_{кр}}} + e^{-\frac{gt}{v_{кр}}}},$$

где t — время, g — ускорение силы тяжести и $v_{кр}$ — критическая скорость, равная

$$v_{кр} = \sqrt{\frac{g}{k_1}}.$$

В свою очередь, коэффициент k_1 определяется формулой

$$k_1 = \alpha \cdot \frac{S \cdot d}{P},$$

в которой d — вес кубометра воздуха в $кГ$, P — вес падающего тела в $кГ$, S — площадь проекции тела на плоскость, перпендикулярную к направлению движения, и α — коэффициент сопротивления, зависящий от формы тела.

Из опытов Эйфеля известно, что для парашюта можно принять $\alpha = 0{,}664$. В обычных условиях можно считать, что $d = 1{,}225$ и $g = 9{,}81$. Принимая проекцию парашюта на горизонтальную плоскость за круг радиуса 4 $м$ и вес парашютиста с парашютом равным 90 $кГ$, рассчитаем его скорость за первую секунду с начала падения с шагом 0,2 $сек$.

Для площади S находим значение $S = \pi R^2 = 16\pi = 50{,}265$, а затем для коэффициента k_1 значение $k_1 = 4{,}5429$. Все остальные вычисления приведены в табл. 2.3.

Для составления расписок удобно пользоваться бланками, напечатанными типографским путем. В зависимости от числа знаков, с которыми нужно вести вычисления, в бланках делают столбцы различной ширины.

Расположение расписки, приведенное в предыдущих примерах, удобно для самостоятельных ручных расчетов. Для расчетов, являющихся контрольными для отладки программ на машине, вычисления производятся лишь в одной-двух точках. Поэтому расписку удобнее располагать не по столбцам, а по строкам, для чего печатаются бланки другой формы.

Таблица 2.3

(1)	(2)	(3)	(4)	(5)	(6)	(7)
k_1	«9,81» : (1)	$v_{кр} = \sqrt{(2)}$	$\frac{«9,81»}{(3)}$	t	(4) · (5)	$e^{(6)}$
4,5429	2,1594	1,4695	6,6757	0	0	1
				0,2	1,3351	3,80038
				0,4	2,6703	$1,44443 \cdot 10^1$
				0,6	4,0054	$5,48938 \cdot 10^1$
				0,8	5,3406	$2,08637 \cdot 10^2$
				1	6,6757	$7,92902 \cdot 10^2$

(8)	(9)	(10)	(11)	(12)
$e^{-(6)}$	(7)—(8)	(7)+(8)	(9):(10)	$v = (11) \cdot (3)$
1	0	2,00000	0	0
0,26312	3,53726	4,06350	0,87050	1,2792
$0,69221 \cdot 10^{-1}$	$1,43751 \cdot 10^1$	$1,45135 \cdot 10^1$	0,99046	1,4555
$0,18127 \cdot 10^{-1}$	$5,48757 \cdot 10^1$	$5,49120 \cdot 10^1$	0,99934	1,4685
$0,47931 \cdot 10^{-2}$	$2,08632 \cdot 10^2$	$2,08642 \cdot 10^2$	0,99995	1,4694
$0,12612 \cdot 10^{-2}$	$7,92901 \cdot 10^2$	$7,92903 \cdot 10^2$	0,99999	1,4695

Пример 3.3. Два вектора, $\boldsymbol{p}_1$ и $\boldsymbol{p}_2$, заданы своими модулями и направляющими косинусами относительно некоторой системы координатных осей. Требуется определить модуль вектора $\boldsymbol{p} = \boldsymbol{p}_1 + \boldsymbol{p}_2$, а также величины

$$E_i = \sqrt{|\boldsymbol{p}_i|^2 + \mu^2}, \quad i = 1, 2,$$

$$E = E_1 + E_2,$$

$$\varkappa = \sqrt{E^2 - |\boldsymbol{p}|^2},$$

где μ — заданная постоянная величина.

Если все случаи рассчитываются вручную, то естественно пользоваться обычной распиской и обычной формой. Если случаи будут обрабатываться на электронной счетной машине, то необходимо провести контрольный ручной расчет для одного варианта. В этом случае удобнее пользоваться формой с другим расположением вычислений. Вычисления приведены в табл. 3.3. Первые строки таблицы содержат исходные числовые данные, вычисления ведутся, как обычно для ручных расчетов при отладке программ, с восемью значащими цифрами.

Таблица 3.3

p_1	(1)		0,78
$\cos\alpha_1$	(2)		0,32929
$\cos\beta_1$	(3)		0,54882
$\cos\gamma_1$	(4)		0,76835
p_2	(5)		3,12
$\cos\alpha_2$	(6)		0,42426
$\cos\beta_2$	(7)		0,70711
$\cos\gamma_2$	(8)		0,56569
x_1	(9)	(1) · (2)	0,2568462
y_1	(10)	(1) · (3)	0,4280796
z_1	(11)	(1) · (4)	0,5993130
x_2	(12)	(5) · (6)	1,3236912
y_2	(13)	(5) · (7)	2,2061832
z_2	(14)	(5) · (8)	1,7649528
x_p	(15)	(9) + (12)	1,5805374
y_p	(16)	(10) + (13)	2,6342628
z_p	(17)	(11) + (14)	2,3642658
	(18)	$(15)^2$	2,4980985
	(19)	$(16)^2$	6,9393405
	(20)	$(17)^2$	5,5897528
$x_p^2+y_p^2+z_p^2=p^2$	(21)	(18) + (19) + (20)	15,027192
p	(22)	$\sqrt{(21)}$	3,8764922
	(23)	$(1)^2$	0,6084
	(24)	(23) + «$0{,}19485368\cdot10^{-1}$»	0,62788537
$E_1=\sqrt{p_1^2+\mu^2}$	(25)	$\sqrt{(24)}$	0,79239218
	(26)	$(5)^2$	9,7344
	(27)	(26) + «$0{,}19485368\cdot10^{-1}$»	9,7538854
$E_2=\sqrt{p_2^2+\mu^2}$	(28)	$\sqrt{(27)}$	3,1231211
$E=E_1+E_2$	(29)	(25) + (28)	3,9155133
E^2	(30)	$(29)^2$	15,331244
	(31)	(30) — (21)	0,304052
$\varkappa=\sqrt{E^2-\lvert\bar{p}\rvert^2}$	(32)	$\sqrt{(31)}$	0,55140910

§ 4. Контроль вычислений по готовым формулам

Все результаты вычислений должны контролироваться, так как в процессе вычислений могут возникать просчеты. Причиной неверного результата могут служить неисправности в работе вычислительной машины. Однако и при правильно работающей машине возможны просчеты. Вычислитель может, особенно когда он начинает утомляться и его внимание рассеивается, записать или прочесть не ту цифру, взять число не из нужного места или записать его не в нужную колонку или строку, либо нажать не ту клавишу, которую надо, и не заметить этого. Неверное число войдет затем в другие вычисления, и в конечном счете многие из полученных результатов могут оказаться неверными.

Различают контроль *заключительный* и *текущий*. Заключительный контроль предполагает проверку окончательных результатов. Например, если целью вычислений было отыскание корня какого-либо уравнения, в качестве заключительного контроля необходимо подставить полученное значение корня в уравнение и произвести проверку того, что уравнение удовлетворяется с нужной степенью точности. Такая проверка корней о б я з а т е л ь н а в с е г д а, хотя бы уравнение было просто квадратным.

При построении таблиц функций хорошим заключительным контролем является составление табличных разностей. При отсутствии у функции особенностей, которые можно обнаружить, исследовав ее аналитическое выражение, разности между последовательными значениями функции должны изменяться плавно и (на небольших участках) быть практически монотонными. Нарушение правильности хода разностей может быть вызвано либо аналитическими особенностями, либо просчетами в вычислении ее значений.

Более детальные исследования разностей и, в особенности, привлечение разностей более высоких порядков позволяют довольно точно обнаружить неверный результат и даже исправить его. Этот вопрос будет рассмотрен ниже, в § 73.

Аналогичную роль играет построение графика функции по вычисленным значениям. При отсутствии аналитических особенностей построенные по результатам вычислений точки должны ложиться на плавную гладкую кривую.

К сожалению, заключительный контроль возможен не для всех типов вычислений. Кроме того, даже в тех случаях, когда он возможен, им нельзя ограничиваться: при сложных и громоздких расчетах слишком много работы будет потрачено зря, если пропустить просчет, допущенный на ранней стадии вычислений. Поэтому безусловно необходим также текущий контроль.

Наиболее совершенной формой текущего контроля является использование *контрольных соотношений*. Некоторые величины, получаю-

щиеся в результате независимых вычислений, могут быть связаны между собою какими-либо соотношениями, и в качестве контроля необходимо проверить выполнение этих соотношений.

Например, при решении треугольников по двум сторонам a и b и углу C нужно прежде всего найти третью сторону c по теореме косинусов

$$c^2 = a^2 + b^2 - 2ab\cos C.$$

После этого нужно находить углы A и B по теореме синусов

$$\sin A = \frac{c\sin C}{a}, \qquad \sin B = \frac{c\sin C}{b}.$$

Контрольным соотношением будет выполнение равенства

$$A + B + C = \pi.$$

Нарушение этого равенства показывает наличие просчета в вычислениях. Некоторая излишняя вычислительная работа, которую нужно провести, окупается уверенностью в правильности полученных результатов.

Аналогичные соотношения используются при работе с матрицами и решении систем линейных уравнений. Эти вопросы будут подробно разобраны в гл. XIII.

Контрольные соотношения можно найти далеко не для всех вычислений. Иногда такие соотношения удается построить искусственно. Во всех тех случаях, когда можно воспользоваться контрольными соотношениями, это непременно следует делать.

Самым распространенным способом контроля вычислений, хотя и не дающим полной гарантии их безошибочности, являются *вычисления в две руки*. Практически все вычисления должны вестись параллельно по одинаковой расписке двумя вычислителями. Результаты вычислений должны сверяться, и нужно, чтобы эти результаты совпадали. Если вычисления производились на клавишных машинах одинакового типа и велись с одинаковым числом значащих цифр, то допустимым можно считать расхождение на одну-две единицы последнего (запасного) знака. При больших расхождениях следует сверить все промежуточные результаты и обнаружить место и причину полученного расхождения.

Вычисления в две руки позволяют с очень большой надежностью исключить влияние просмотра, описки или очитки вычислителя и неправильностей в работе машины, так как вероятность одинаковых просчетов ничтожно мала. Тем не менее и при вычислениях в две руки можно получить совпадающие ошибочные результаты, например, в том случае, когда оба вычислителя пользовались одной и той же таблицей, в которой имеется опечатка.

При больших и сложных или громоздких расчетах необходимо сверять не только окончательные, но и промежуточные результаты,

т. е. вести не только заключительный, но и текущий контроль. Но сверять каждое число или даже каждую колонку сразу после ее получения нельзя, так как при этом резко возрастает вероятность одинаковых просчетов. Сверку результатов лучше всего производить через каждые восемь — десять столбцов расписки.

Подчеркнем, что вычисления в две руки никак не отменяют и не заменяют других способов текущего и заключительного контроля: использования контрольных соотношений, если их можно получить, и заключительного контроля по разностям или по графику, если речь идет о таблице функций, или вычисления значения функции, если речь идет об отыскании корня. Все эти вычисления обязательно должны производиться, и также в две руки.

§ 5. Погрешности арифметических действий

В § 1 шла речь о погрешностях в исходных данных. Кроме них, на погрешности результата влияют также и погрешности выполняемых нами арифметических действий. Поэтому в процессе вычислений сразу возникает вопрос о том, с какой точностью, т. е. с каким числом знаков, должны вестись вычисления. Часто такой вопрос решается однозначно: если исходные данные получены из опыта, нет необходимости вести промежуточные вычисления с большей точностью, чем точность исходных данных. Но в таких случаях нужно оценить, с какой точностью мы будем знать окончательный результат.

Примером такой задачи может являться следующая. Предположим, что радиус круга r измерен с некоторой ошибкой. Как оценить ошибку величины площади круга, вычисленной по формуле $S=\pi r^2$? Иногда необходимо решать и обратную задачу: задается величина допустимой ошибки для площади круга; требуется определить, с какой точностью следует измерять ее радиус. Ясно, что если величину площади круга нужно иметь с небольшой точностью, то нет смысла добиваться слишком большой точности в измерении его радиуса.

В более общей постановке эти задачи можно формулировать следующим образом. Независимое переменное x известно с некоторой точностью; с какой точностью можно найти значение функции $y=f(x)$? Аналогично формулируется и обратная задача: необходимо получить значение функции y с заданной точностью; какова должна быть точность значений независимого переменного x?

Решение этих задач дается следующей основной теоремой.

Теорема. *Предельная абсолютная погрешность функции равна произведению абсолютной величины ее производной на предельную абсолютную погрешность аргумента.*

Доказательство. Пусть число x является приближенным значением величины X с абсолютной погрешностью $|\Delta x|$,

$$X=x+\Delta x,$$

причем Δx может быть как положительным, так и отрицательным, потому что x может быть приближением как по недостатку, так и по избытку. Обозначим абсолютную погрешность функции через $|\Delta y|$.

Тогда

$$|\Delta y| = |f(X) - f(x)| = |f(x+\Delta x) - f(x)|.$$

Ввиду малости $|\Delta x|$ мы можем заменить приращение функции ее дифференциалом. Тогда получим

$$f(x+\Delta x) \approx f(x) + f'(x)\cdot\Delta x,$$

откуда

$$|\Delta y| \approx |f'(x)|\cdot|\Delta x|.$$

Обозначив предельную погрешность аргумента через α, т. е. считая, что $|\Delta x| \leqslant \alpha$, найдем

$$|\Delta y| \leqslant \alpha |f'(x)|.$$

Таким образом, предельную абсолютную погрешность β функции $f(x)$ можно принять равной

$$\beta = \alpha\cdot|f'(x)|. \tag{2.5}$$

Теорема доказана.

Обозначим предельную относительную погрешность аргумента через δ_x и функции через δ_y. Тогда, учитывая, что $\delta_x = \frac{\alpha}{|x|}$, т. е. $\alpha = |x|\cdot\delta_x$, получим для δ_y выражение

$$\delta_y = \frac{\beta}{|f(x)|} = \frac{\alpha|f'(x)|}{|f(x)|} = \left|x\cdot\frac{f'(x)}{f(x)}\right|\cdot\delta_x. \tag{3.5}$$

Вспомнив выражение для логарифмической производной

$$\frac{d\ln f(x)}{dx} = \frac{f'(x)}{f(x)},$$

перепишем равенство (3.5) в виде

$$\delta_y = \left|x\frac{d\ln f(x)}{dx}\right|\cdot\delta_x. \tag{4.5}$$

Формулой (4.5) можно воспользоваться для нахождения относительных погрешностей ряда конкретных функций. Пусть, например, $f(x) = x^n$. Тогда

$$\delta_y = \left|x\cdot\frac{nx^{n-1}}{x^n}\right|\cdot\delta_x = |n|\cdot\delta_x, \tag{5.5}$$

т. е. *предельная относительная погрешность степени равна предельной относительной погрешности основания, умноженной на абсолютную величину показателя степени.*

Найдем еще предельную абсолютную погрешность β для логарифмической функции. Положив $f(x) = \ln x$, по формуле (3.5) находим

$$\beta = \alpha\left|\frac{1}{x}\right| = \left|\frac{\alpha}{x}\right| = \delta_x,$$

так что *предельная абсолютная погрешность натурального логарифма равна предельной относительной погрешности аргумента.*

Пример 1.5. Определим предельную относительную погрешность площади круга, вычисляемой по формуле $S = \pi r^2$.

Из (3.5) следует

$$\delta_S = \left| r\frac{2\pi r}{\pi r^2} \right| \cdot \delta_r = 2\delta_r.$$

Таким образом, относительная погрешность вычисленной площади вдвое больше относительной погрешности измеренного радиуса.

Пример 2.5. Определим, с какой относительной погрешностью может быть найдена сторона квадрата, если известно, что его площадь равна $S = 12{,}34$ *см*^{2}.

Из выражения площади видно, что она дана с предельной абсолютной погрешностью 0,005 *см*^{2}, откуда следует, что ее относительная погрешность

$$\delta_S = \frac{0{,}005}{12{,}34} = 0{,}0004 \ (= 0{,}04\%).$$

Так как сторона квадрата $a = \sqrt{S}$, то из (3.5) находим

$$\delta_a = \left| S \cdot \frac{1}{2\sqrt{S}\cdot\sqrt{S}} \right| \delta_S = \frac{1}{2}\cdot \delta_S = 0{,}0002 \ (= 0{,}02\%).$$

Пример 3.5. Вычислим относительную погрешность объема шара, определяемого по формуле $v = \frac{4}{3}\pi R^3$. Пользуясь формулой (3.5), находим

$$\delta_v = \left| R \cdot \frac{4\pi R^2}{\frac{4}{3}\pi R^3} \right| \cdot \delta_R = 3\delta_R.$$

Подобно результату, полученному в примере 1.5, мы получили, что относительная погрешность объема шара равна утроенной относительной погрешности радиуса.

Рассмотрим теперь вопрос об оценке предельной абсолютной погрешности функции $u = f(x, y)$ двух независимых переменных.

Пусть x и y являются соответственно приближенными значениями величин X и Y с абсолютными погрешностями $|\Delta x|$, $|\Delta y|$. Следовательно,

$$X = x + \Delta x, \ Y = y + \Delta y.$$

Тогда

$$|\Delta u| = |f(X, Y) - f(x, y)|$$

или

$$|\Delta u| = |f(x + \Delta x, y + \Delta y) - f(x, y)|.$$

Считая величины Δx и Δy малыми, заменим приращение полным дифференциалом du. Это приводит к приближенному равенству

$$\Delta u \approx \frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y,$$

откуда

$$|\Delta u| \leqslant \left|\frac{\partial u}{\partial x}\right| \cdot |\Delta x| + \left|\frac{\partial u}{\partial y}\right| \cdot |\Delta y|.$$

Если обозначить предельные абсолютные погрешности аргументов x и y соответственно через α_x и α_y:

$$|\Delta x| \leqslant \alpha_x, \ |\Delta y| \leqslant \alpha_y,$$

то

$$|\Delta u| \leqslant \left|\frac{\partial u}{\partial x}\right| \cdot \alpha_x + \left|\frac{\partial u}{\partial y}\right| \cdot \alpha_y$$

и за предельную абсолютную погрешность β функции $f(x, y)$ можно принять величину

$$\beta = \left|\frac{\partial u}{\partial x}\right| \alpha_x + \left|\frac{\partial u}{\partial y}\right| \alpha_y. \tag{6.5}$$

Ясно, что аналогичное равенство имеет место и для дифференцируемой функции любого большего числа аргументов.

Рассмотрим несколько примеров функций частного вида.

Пусть $u = x + y + z + \ldots + t$. Тогда

$$\left|\frac{\partial u}{\partial x}\right| = \left|\frac{\partial u}{\partial y}\right| = \ldots = \left|\frac{\partial u}{\partial t}\right| = 1$$

и

$$|\Delta u| \leqslant \alpha_x + \alpha_y + \ldots + \alpha_t.$$

Абсолютная погрешность суммы не превосходит суммы предельных абсолютных погрешностей слагаемых. При этом слагаемые могут иметь различные знаки. Следовательно, *предельная абсолютная погрешность суммы равна сумме предельных абсолютных погрешностей слагаемых.*

При установлении предельной относительной погрешности суммы надо различать два случая:

а) все слагаемые имеют одинаковые знаки;

б) слагаемые имеют разные знаки.

В первом случае, считая для простоты все слагаемые положительными, имеем

$$\delta = \frac{\alpha_x + \alpha_y + \ldots + \alpha_t}{x + y + \ldots + t}.$$

Обозначим через δ_x, δ_y, ..., δ_t относительные погрешности слагаемых x, y, ..., t. Тогда $\alpha_x = \delta_x \cdot x$, $\alpha_y = \delta_y \cdot y$, ..., $\alpha_t = \delta_t \cdot t$. Обозначим еще через $\delta_{\max}$ и $\delta_{\min}$ наибольшее и наименьшее из чисел δ_x, δ_y, ..., δ_t. Будем иметь следующие оценки:

$$\delta = \frac{\delta_x \cdot x + \delta_y \cdot y + \ldots + \delta_t \cdot t}{x + y + \ldots + t} < \frac{\delta_{\max}(x + y + \ldots + t)}{x + y + \ldots + t} = \delta_{\max}$$

и, аналогично, $\delta > \delta_{\min}$.

Таким образом,

$$\delta_{\min} < \delta < \delta_{\max},$$

т. е. *относительная погрешность суммы слагаемых одного знака заключена между наименьшей и наибольшей относительными погрешностями слагаемых.*

Во втором случае дело обстоит сложнее. Пусть $x > 0$, $y > 0$ и $u = x - y$. Тогда (сохраняя прежние обозначения) будем иметь

$$\delta = \frac{\alpha_x + \alpha_y}{|x - y|}.$$

Следовательно, если числа x и y близки друг к другу, то даже при очень малых погрешностях x и y величина предельной относительной ошибки может быть весьма значительной.

Поясним сказанное числовым примером. Пусть $x = 1{,}245$, $y = 1{,}235$. Здесь $\alpha_x = \alpha_y = 0{,}0005$; $\delta_x \approx \delta_y \approx 0{,}04\%$. Вычисляя же предельную относительную погрешность разности, получим $\delta = \dfrac{0{,}0005 + 0{,}0005}{0{,}01} \cdot 100\% = 10\%$.

Таким образом, в результате вычитания двух близких чисел может произойти большая потеря точности.

Положим $u = xyz$ $(x > 0,\ y > 0,\ z > 0)$.

Тогда

$$\left|\frac{\partial u}{\partial x}\right| = yz; \quad \left|\frac{\partial u}{\partial y}\right| = xz; \quad \left|\frac{\partial u}{\partial z}\right| = xy.$$

Формула (6.5) дает

$$\beta = yz\alpha_x + xz\alpha_y + xy\alpha_z.$$

Предельная относительная погрешность функции u будет равна

$$\delta_u = \frac{\beta}{xyz} = \frac{yz\alpha_x + xz\alpha_y + xy\alpha_z}{xyz} = \delta_x + \delta_y + \delta_z,$$

так что *предельная относительная погрешность произведения равна сумме предельных относительных погрешностей сомножителей.*

Положим, наконец, $u = \frac{x}{y}$ $(x > 0,\ y > 0)$. Тогда

$$\left|\frac{\partial u}{\partial x}\right| = \frac{1}{y}, \quad \left|\frac{\partial u}{\partial y}\right| = \left|-\frac{x}{y^2}\right| = \frac{x}{y^2}.$$

Из формулы (6.5) находим

$$\beta = \frac{1}{y}\alpha_x + \frac{x}{y^2}\alpha_y,$$

откуда

$$\delta_u = \frac{\beta}{u} = \frac{\alpha_x}{x} + \frac{\alpha_y}{y} = \delta_x + \delta_y,$$

следовательно, *предельная относительная погрешность частного равна сумме предельных относительных погрешностей делимого и делителя.*

Пример 4.5. Сила взаимодействия между двумя зарядами, величины e_1 и e_2, находящимися на расстоянии r друг от друга, выражается законом Кулона

$$F = \frac{e_1 e_2}{\varepsilon r^2},$$

где ε — диэлектрическая постоянная среды. Зная, что величина ε определена с относительной погрешностью 5%, величины зарядов e_1 и e_2 — с относительной погрешностью 0,5%, а расстояние между ними r — с относительной погрешностью 1%, определим относительную ошибку величины F.

Согласно выражению для предельной относительной погрешности частного имеем

$$\delta_F = \delta_{e_1} + \delta_{e_2} + \delta_\varepsilon + 2\delta_r,$$

откуда $\delta_F = 8\%$.

Итак, для оценки погрешности мы получаем следующие правила:

1) *При сложении и вычитании абсолютные погрешности складываются.*

2) *При умножении и делении относительные погрешности складываются; при возведении в степень относительные погрешности умножаются на абсолютную величину показателя степени.*

3) *При нахождении значения функции абсолютная погрешность функции равна произведению абсолютной погрешности аргумента на абсолютную величину производной.*

Заметим еще, что при вычислении значений функции абсолютная погрешность может существенным образом зависеть от того, каким образом запи-

сана расчетная формула и каково расположение операций в этой формуле. Для пояснения рассмотрим следующий пример.

Пример 5.5. Вычислим площадь кругового кольца с внутренним радиусом $r = 1{,}750$ и толщиной $h = 0{,}005$.

Здесь вычисления можно производить по формулам

$$S = \pi [(r + h)^2 - r^2]$$

и

$$S = \pi (2r + h) h.$$

Хотя алгебраически эти формулы тождественны, но для вычислений вторая во много раз лучше первой, так как при вычитании близких величин в первой формуле относительная погрешность сильно возрастает.

Действительно, подсчитаем предельную абсолютную погрешность величины $\frac{S}{\pi}$ в том и другом случаях. В соответствии с правилом записи приближенных чисел следует считать, что предельная абсолютная погрешность величин r и h составляет 0,0005; при этом предельная абсолютная погрешность их суммы $r + h$ равна 0,001. Тогда предельные относительные погрешности величин r и $r + h$ равны соответственно 0,03 и 0,06%, а их квадратов — 0,06 и 0,12% (по второму правилу). Следовательно, предельная абсолютная погрешность величины $(r + h)^2$ равна 0,0037, а r^2 равна 0,0018. По первому правилу абсолютная погрешность их разности составляет $0{,}0037 + 0{,}0018 = 0{,}0055$, т. е. при расчете по первой формуле можно гарантировать лишь, что абсолютная погрешность величины $\frac{S}{\pi}$ не превзойдет 0,0055.

Совсем иначе обстоит дело в случае второй формулы. Предельная абсолютная погрешность множителя $2r + h$ равна 0,0015, предельная относительная погрешность — 0,04%, а предельная относительная погрешность h — 10%. Поэтому предельная относительная погрешность отношения $\frac{S}{\pi}$ составляет 10,04%, так что предельная абсолютная погрешность этого отношения при расчете по второй формуле равна 0,0018, что в три раза меньше, чем в предыдущем случае.

Пример 5.5 показывает, насколько важно правильно выбрать вид расчетной формулы. Поэтому к указанным выше правилам подсчета погрешностей следует присоединить важный практический совет:

Формулы для вычислений надо стараться преобразовывать к такому виду, чтобы в них не было вычитания близких величин; последнее может привести к большой потере точности и большим относительным ошибкам.

Обращаем внимание читателя, что во всем предыдущем изложении был рассмотрен только один вопрос: *как, зная погрешности исходных данных, оценить погрешность результата расчета.* При этом мы совсем не касались вопроса о погрешностях округления. Точная оценка влияния погрешностей округления на окончательный результат пока еще не может быть сделана. Поэтому мы ограничимся следующим практическим советом: *для уменьшения погрешностей округления промежуточные действия рекомендуется производить, сохраняя один-два лишних знака, после чего окончательный результат следует округлять.*

В связи с ошибками округления мы можем снова сравнить две расчетные формулы, приведенные в примере 5.5 с другой точки зрения. Можно считать, что значения r и h являются точными, и интересоваться вопросом о том, с каким числом знаков нужно вести вычисления для достижения нужной точности. И с этой точки зрения вторая формула, не содержащая разностей близких величин, оказывается гораздо лучше, нежели первая.

ГЛАВА II

СРЕДСТВА ВЫЧИСЛЕНИЙ

§ 6. Функциональные шкалы

Кроме хорошо известного способа геометрического представления функции в виде графика, в вычислительной математике применяется также и другой способ изображения — *функциональные шкалы.*

Рассмотрим функцию $y=f(x)$, непрерывную и монотонную на некотором замкнутом интервале $[a, b]$. Возьмем ось OM, на которой будет строиться шкала, выберем на ней точку отсчета O и установим определенный масштаб μ, т. е. величину отрезка, принимаемого за единицу.

Функциональная шкала для функции $f(x)$ строится теперь следующим образом.

Разбив интервал $[a, b]$ на равные части, вычислим значение функции $f(x)$ в каждой из точек деления, включая начало и конец отрезка, и отложим на оси OM отрезок $\mu f(x)$. Полученная при этом точка снабжается отметкой x, т. е. откладывается в выбранном масштабе значение функции, а надписывается значение аргумента (рис. 1).

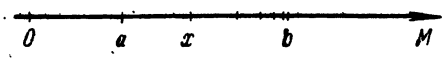


Рис. 1.

Иногда начало шкалы помещают в точку отсчета, т. е. точку с надписью a совмещают с точкой O. Тогда точка x будет находиться в конце отрезка $\mu[f(x)-f(a)]$.

Полученная шкала позволяет судить о поведении функции на рассматриваемом участке. Так, если функция возрастает, то надписи на шкале будут возрастать в положительном направлении на оси, а если убывает — в противоположном; большие промежутки между отметками укажут, что функция возрастает быстрее, чем там, где эти промежутки малы, и т. п.

Если наряду с функциональной шкалой построить также равномерную шкалу с отметками, содержащими истинные длины отложенных отрезков, то такую шкалу можно использовать для нахождений значений функции или значений обратной функции. Точность этих

значений, естественно, определяется выбором масштаба и числа точек деления.

Построим для примера функциональную шкалу для функций $y = x^2$ на участке [1, 2]. Выбрав масштаб μ, мы определим тем самым длину получающейся шкалы. Чаще поступают наоборот: задавшись заранее длиной шкалы l, определяют отсюда масштаб, что оказывается более удобным. Длина шкалы l и масштаб μ связаны соотношением $\mu[f(b) - f(a)] = l$. В нашем случае $f(a) = f(1) = 1$, $f(b) = f(2) = 4$, поэтому, выбрав $l = 6$ *см*, найдем $\mu = 2$ *см*.

Для построения шкалы разобьем отрезок [1, 2] на десять частей и вычислим значения функции во всех точках деления, а затем подсчитаем длины отрезков, которые надо откладывать в выбранном

Таблица 1.6

(1)	x	1,00	1,10	1,20	1,30	1,40	1,50	1,60	1,70	1,80	1,90	2,00
(2)	$(1)^2$	1,00	1,21	1,44	1,69	1,96	2,25	2,56	2,89	3,24	3,61	4,00
(3)	(2)—«1»	0,00	0,21	0,44	0,69	0,96	1,25	1,56	1,89	2,24	2,61	3,00
(4)	«2» · (3)	0,00	0,42	0,88	1,38	1,92	2,50	3,12	3,78	4,48	5,22	6,00

масштабе. Все вычисления приведены в табл. 1.6. Перенеся полученные результаты на чертеж, получим функциональную шкалу для функции $y = x^2$ (рис. 2). Здесь начало шкалы как раз совпадает с точкой отсчета.

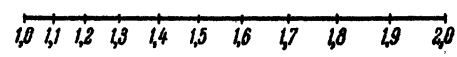


Рис. 2.

На рис. 3 изображена та же функциональная шкала, дополненная равномерной шкалой (сверху). По ней можно находить значение x^2 для $1 \leqslant x \leqslant 2$. Для этого следует найти x на нижней шкале и прочесть значение на верхней шкале, в случае необходимости интерполируя на глаз. Эта же шкала может быть использована для нахождения значений $\sqrt{x}$ при $1 \leqslant x \leqslant 4$, для чего следует найти x на верхней шкале и прочесть значение на нижней.

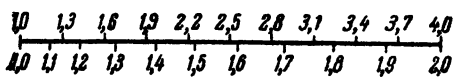


Рис. 3.

Примером функциональной шкалы является основная шкала логарифмической линейки. Она представляет шкалу, построенную для функции $y = \lg x$ на участке $1 \leqslant x \leqslant 10$. Масштаб совпадает здесь с длиной линейки, так как $\lg 10 = 1$. Чаще всего берется $\mu = 25$ *см*. Иногда встречаются также маленькие линейки с $\mu = 15$ *см* или, наоборот, большие, у которых $\mu = 50$ *см*. Действия на логарифмической линейке будут рассмотрены в § 7.

Функциональные шкалы находят широкое применение при обработке экспериментальных данных благодаря тому, что графики многих функций могут быть специальным подбором функциональных шкал преобразованы к прямолинейному виду. Познакомимся с некоторыми примерами.

Пример 1. 6. Рассмотрим уравнения парабол:

$$\text{а) } y = x^2,$$
$$\text{б) } y = 3x^2 + 25,$$
$$\text{в) } y = -\frac{1}{2}x^2 + 50.$$

Построим теперь на оси Oy обычную равномерную шкалу, а на оси Ox — шкалу квадратов. Тогда получится сетка, изображенная на рис. 4. Построение такой шкалы эквивалентно замене переменных $x^2 = \xi$. Поэтому в новых координатах уравнения парабол будут иметь вид

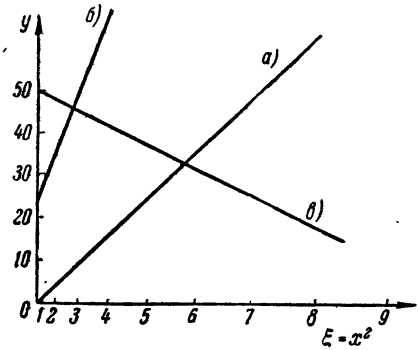


Рис. 4.

$$\text{а) } y = \xi,$$
$$\text{б) } y = 3\xi + 25,$$
$$\text{в) } y = -\frac{1}{2}\xi + 50,$$

т. е. будут уравнениями первой степени, а значит, будут изображаться прямыми. Эти прямые и изображены на рис. 4.

Координатные сетки, построенные с помощью функциональных шкал, называют *функциональными сетками*.

Особенно часто используются логарифмические сетки, позволяющие «выпрямлять» графики показательных и степенных функций.

Действительно, рассмотрим показательную функцию $y = ae^{bx}$. Логарифмирование (по основанию 10) дает $\lg y = \lg a + b \lg e \cdot x$. Обозначим $\lg a = A$ и $b \cdot \lg e = B$. Если построить на оси Ox равномерную шкалу, а на оси Oy — логарифмическую, т. е. положить $\lg y = Y$, то получится линейная зависимость $Y = A + Bx$, которая выражается прямой линией.

Аналогично обстоит дело и для степенной функции $y = ax^b$. Логарифмирование дает $\lg y = \lg a + b \lg x$. Построив логарифмические сетки на обеих осях, т. е. полагая $\lg x = X$ и $\lg y = Y$, и обозначив $\lg a = A$, получим линейную зависимость $Y = A + bX$.

Такие сетки печатаются типографским способом и продаются наравне с миллиметровой бумагой под названием *полулогарифмической* и *логарифмической бумаги*.

§ 7. Логарифмическая линейка

Счетная логарифмическая линейка является наиболее простым и удобным, а потому и наиболее распространенным средством вычислений. Она позволяет вести вычисления с точностью трех значащих цифр, т. е. с относительной погрешностью порядка 0,1%.

Принцип действия логарифмической линейки основан на сложении отрезков. Если взять две одинаковые равномерные шкалы, то, сдвигая

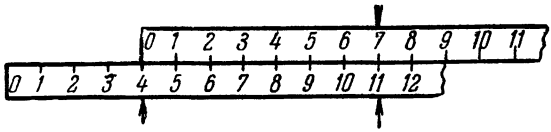


Рис. 5.

начальную точку одной из них на величину, равную одному слагаемому и откладывая на сдвинутой шкале второе слагаемое, можно на неподвижной шкале найти соответствующую сумму (рис. 5).

Как было сказано в предыдущем параграфе, основной шкалой логарифмической линейки является не равномерная шкала, а логарифмическая. Поэтому сложение отрезков будет соответствовать уже не сложению чисел, а сложению их логарифмов, т. е. умножению. Вычитанию отрезков соответствует деление. Таким образом, умножение и деление чисел производится с помощью основных шкал на линейке и на движке так, как это показано на рис. 6 и 7.

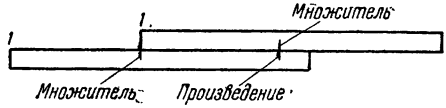


Рис. 6.

Для вычислений на логарифмической линейке числа удобнее всего задавать в плавающей форме, нормализованными. Порядок произведения в этом случае легко определяется; нужно только учитывать возможную необходимость нормализации результата. Впрочем, необходимость сдвига при нормализации легко определить по положению движка.

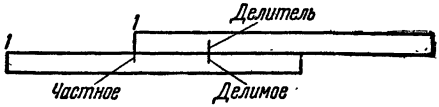


Рис. 7.

Действительно, так как мантиссы нормализованных чисел заключены между 0,1 и 1 (по абсолютной величине), то их произведения содержатся между 0,01 и 1. Если произведение мантисс меньше, чем 0,1, то движок будет сдвинут влево, а если больше 0,1, то вправо. Во втором случае произведение получается нормализованным и его порядок равен сумме порядков сомножителей. В первом случае для нормализации мантиссу следует сдвинуть на один разряд влево, для чего

необходимо уменьшить порядок на единицу, т. е. после нормализации порядок произведения на единицу меньше суммы порядков слагаемых.

Итак, мы приходим к следующему правилу: *порядок произведения равен сумме порядков сомножителей, если движок линейки сдвинут влево, и на единицу меньше этой суммы при сдвиге движка вправо.* Аналогичное правило справедливо также и для деления: *порядок частного равен разности порядков делимого и делителя, если движок линейки сдвинут влево, и на единицу больше этой разности при сдвиге движка вправо.*

Кроме основных шкал, на логарифмической линейке имеется также еще ряд шкал, которые могут быть использованы для различных целей. В двух экземплярах — вверху движка и на корпусе линейки — имеется логарифмическая шкала, построенная в половинном масштабе и повторенная дважды. Она является шкалой квадратов, так как при переходе от основной шкалы к данной логарифм числа удваивается, а это соответствует возведению числа в квадрат. С помощью этой шкалы можно также и извлекать квадратные корни, переходя со шкалы квадратов на основную.

Легко сообразить, что *порядок квадрата равен удвоенному порядку числа, если квадрат находится в правой половине шкалы, и на единицу меньше удвоенного порядка числа, если квадрат находится в левой половине.* При извлечении квадратного корня *мантисса подкоренного выражения помещается на левой половине шкалы квадратов, если порядок числа нечетный, и на правой половине, если этот порядок четный.* Порядок корня определяется по следующему правилу: *порядок корня равен половине порядка подкоренного числа, если этот порядок четный, и на* $\frac{1}{2}$ *больше ее, если порядок подкоренного числа нечетный.*

Благодаря тому, что шкала квадратов имеется и на движке, и на корпусе линейки, с помощью этих шкал можно производить умножение и деление, как и на основных. Это может быть полезным, например, в тех случаях, когда из произведения или частного нужно еще извлечь квадратный корень.

Самой верхней шкалой линейки обычно является утроенная логарифмическая шкала, которая служит шкалой кубов. С ее помощью можно возводить число в куб или извлекать кубический корень так же, как это описывалось выше для квадратов. Внизу на линейке помещается равномерная шкала. С ее помощью можно находить мантиссы десятичных логарифмов чисел или, наоборот, числа по их десятичным логарифмам, т. е. значения показательной функции $y = 10^x$. Это делается так же, как и по шкале квадратов, изображенной на рис. 3 в предыдущем параграфе. В середине движка часто помещается шкала обратных величин, которая является той же логарифмической шкалой, только построенной справа налево.

На обороте движка нанесены шкалы для отыскания значений тригонометрических функций. Значение синуса для углов от 5°45′ до 90° или тангенса для углов от 5°45′ до 45° находится на основной шкале против отметки, соответствующей требуемому углу на шкале синусов или тангенсов. Так как для указанных углов синусы и тангенсы заключены в пределах от 0,1 до 1, то мы получаем нормализованную мантиссу и нулевой порядок. Для углов, меньших чем 5°45′, синус и тангенс в пределах точности линейки совпадают. Соответствующее значение угла ищется на средней шкале движка. Порядок найденного числа равен — 1, мантисса находится на основной шкале.

Описанный способ предполагает, что движок перевернут тригонометрическими шкалами вверх. Благодаря рискам, нанесенным в прорезях с обратной стороны линейки, можно находить тригонометрические функции не переворачивая движка. Для этого нужно установить требуемый угол на соответствующей шкале против риски; значение тригонометрической функции можно тогда прочесть на основной шкале движка, против единицы основной шкалы линейки.

Для свободного владения техникой вычислений на логарифмической линейке нужно уметь:

1) Быстро и безошибочно находить на любой шкале нужное число (его мантиссу) и устанавливать против него визирную линию или требуемую точку движка, а также легко считывать мантиссу полученного результата.

2) Легко подсчитывать порядок полученного результата.

3) Комбинировать действия таким образом, чтобы получать требуемый результат после наименьшего числа перебрасываний движка.

Соответствующие навыки приобретаются лишь при достаточной тренировке. Мы не станем описывать специальных приемов выполнения или комбинирования ряда операций, отсылая читателя к книгам, специально посвященным логарифмической линейке. Из них можно рекомендовать книгу Л. З. Румшиского «Счетная линейка» («Наука», 1964), а также книги К. А. Семендяева или Д. Ю. Панова с тем же названием, неоднократно издававшиеся.

§ 8. Арифмометры и клавишные машины

Основными средствами для ручных вычислений, позволяющими вести вычисления с точностью от четырех до восьми — десяти значащих цифр, являются в настоящее время арифмометры: ручные, полуавтоматические или автоматические. До недавнего времени наиболее распространенными ручными арифмометрами были рычажные арифмометры «Феликс» и десятиклавишные ВК-1. Сейчас они уже практически не используются; в основном вычисления ведутся на полуавтоматических или автоматических арифмометрах, которые иначе называют *клавишными машинами*. Самыми распространенными клавиш-

ными машинами являются полноклавишные автоматы типа «Мерседес» и «Рейнметалл». По типу «Рейнметалла» (модель САР) сделаны автоматы ВММ-2 и полуавтоматы ВМП-2. Используются также десятиклавишные автоматы и полуавтоматы ВК-2.

Приемы работы на различных клавишных машинах различны и описывать их здесь бесполезно. Для свободного владения техникой вычислений на клавишных машинах нужно уметь:

1) Быстро и безошибочно набирать на нужных клавиатурах требуемые числа и считывать с соответствующих счетчиков полученные результаты. Удобно набирать числа на клавиатуре аккордом, нажимая одновременно несколько клавиш.

Некоторые вычислители рекомендуют набирать числа на клавиатуре левой рукой, чтобы правая была свободна для записей.

2) Твердо знать расположение клавиш соответствующих действий и находить их не глядя на машину.

3) Шире пользоваться различными дополнительными возможностями, предоставляемыми машиной.

Клавишные машины следует считать, вообще говоря, машинами с плавающей запятой, поскольку положение запятой на них заранее не фиксировано. Однако для каждого отдельного действия число представляется в фиксированной форме, так что фактически в каждом *отдельном действии* клавишная машина работает как машина *с фиксированной запятой*. Положение запятой может быть установлено с помощью специальных указателей, которые передвигаются вычислителем. Указатель запятой на счетчике результатов следует устанавливать до того, как нажата клавиша выполнения действия.

Клавишные машины позволяют выполнять все четыре арифметических действия. Кроме того, они дают возможность выполнения некоторых комбинированных действий. Например, благодаря возможности накопления на регистре (счетчике) результатов, можно получить сумму произведений, не выписывая промежуточных результатов. На некоторых машинах можно переносить числа с регистра результата на регистр установки множителя, благодаря чему можно выполнять несколько умножений подряд, также не выписывая промежуточных результатов.

§ 9. Другие средства вычислений

Самыми важными из вспомогательных средств вычислений следует считать различного рода математические таблицы. Их применение позволяет заменять выполнение многих операций выписыванием готовых результатов.

Вычисления с помощью таблиц логарифмов имели очень большое значение для сложных многозначных расчетов без использования машин. Как известно, умножению или делению чисел соответствует сло-

жение или вычитание их логарифмов, а эти действия легко выполнять вручную. В связи с этим был разработан ряд приемов преобразования выражений, особенно содержащих тригонометрические функции, к виду, удобному для логарифмирования.

Вследствие широкого распространения клавишных вычислительных машин вычисления с логарифмами потеряли в своей важности очень многое и играют уже далеко не ту первостепенную роль, как раньше. По сути дела, логарифмы в практике вычислений используются сейчас лишь для возведения в степень, если она дробная или большая, и для извлечения корня (кроме квадратного, который извлекается непосредственно). Ввиду того, что вычисления с логарифмами потеряли массовый характер, преимущества десятичных логарифмов перед другими системами уже не играют никакой роли. Поэтому в настоящее время при вычислениях предпочтительнее пользоваться натуральными логарифмами, которые играют значительно бо́льшую теоретическую роль, нежели десятичные.

В качестве одного из средств вычислений с небольшой точностью можно указать применение функциональных шкал, описанных в § 6. Более широкое их использование дает возможность производить вычисления по достаточно сложным формулам. Это составляет предмет специального раздела математики, называемого *номографией*. Рассмотрение методов построения номограмм не входит в наши намерения.

О различных типах математических машин, являющихся мощным современным средством вычислений, речь будет идти в гл. XI.

ЧАСТЬ ВТОРАЯ
ОСНОВЫ ПРОГРАММИРОВАНИЯ

ГЛАВА III
АРИФМЕТИЧЕСКИЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ

§ 10. Системы счисления

Конструкция вычислительных машин и программирование на них тесно связаны с *системой счисления.* Так называют в математике способы наименования и записи чисел.

Все системы счисления делятся на две группы: *позиционные* и *непозиционные системы.* В позиционной системе одна и та же цифра может означать различные числа, в зависимости от места (позиции), где она стоит. В непозиционной системе, наоборот, каждый знак всегда изображает одно и то же число.

Хорошо известным примером непозиционной системы счисления является пришедшая к нам из Древнего Рима *римская система,* в которой для записи чисел используются буквы латинского алфавита. При этом буква I всегда означает единицу, буква V — пять, X — десять, L — пятьдесят, C — сто, D — пятьсот, M — тысячу и т. д. Число 368 запишется в римской системе в виде

CCCLXVIII

Таким образом, величина числа получается здесь сложением величин, изображаемых отдельными буквами. По этой причине непозиционные системы часто называют также *аддитивными* (от латинского слова additio, что значит сложение).

Правда, при развитии римской системы было внесено некоторое изменение: чтобы уменьшить число знаков, требуемых для записи числа, установили, что если поместить букву, означающую меньшее число, слева от большего, то это меньшее число следует вычитать, а не прибавлять. Например, три записывается как III, а четыре как IV. Аналогично IX означает девять, XL — сорок, XC — девяносто и т. д. Но это изменение никак не отражается на основном принципе — каждая используемая буква всегда означает одно и то же число. Поэтому для записи больших чисел введенных знаков будет не хватать и нужны будут новые, и сколько бы мы их не ввели, всегда можно при-

думать число, которое уже введенными знаками изобразить трудно; так, например, трудно изобразить тысячу, имея лишь знаки I, V, X.

Совсем иначе обстоит дело в *позиционных* системах. Здесь имеется определенное количество знаков (цифр), каждая из которых может означать различные числа, в зависимости от места (*позиции*), которое этот знак занимает.

Общепринятой системой счисления является *десятичная позиционная система*, берущая свое начало от счета на пальцах. Она была изобретена в Индии, затем заимствована там арабами и уже через арабские страны пришла в Европу. В этой системе для записи любого числа используется лишь десять цифр — 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Значение каждой цифры в позиционной системе счисления определяется как ею самой, так и местом, которое она занимает.

Так, единица изображается цифрой 1, десять — той же цифрой, стоящей на втором месте справа, т. е. 10, сто — той же цифрой 1, но уже стоящей на третьем месте и т. д. Таким образом, каждое число разбивается на разряды, которые считаются справа налево и единица каждого следующего разряда в определенное число раз превосходит единицу предыдущего. Это отношение соседних разрядов называют *основанием системы счисления*. Принятая система счисления потому и называется *десятичной*, что ее основанием является число *десять*: каждый следующий разряд в десять раз больше предыдущего.

Возможны системы счисления с любым основанием. В Древнем Вавилоне, например, применялась шестидесятеричная система счисления. Остатки ее мы находим в сохранившемся до наших дней обыкновении делить час или градус на 60 минут, минуту на 60 секунд, а круг на 360, т. е. шесть раз по 60 градусов.

Возникновение шестидесятеричной системы имело вполне материальные основы. Она возникла около двух тысячелетий до нашей эры при слиянии в одно государство двух древних народов — сумерийцев и аккадян. Во вновь образованном государстве остались в ходу единицы веса, используемые ранее тем и другим народами, причем одна из этих единиц была приблизительно в шестьдесят раз больше другой.

Кроме шестидесятеричной системы употреблялась также и двенадцатеричная, следами которой является сохранившийся обычай считать некоторые предметы дюжинами. Встречалась также (например, в древнем Китае) пятеричная система.

Для изображения числа в позиционной системе счисления требуется столько различных цифр, каково основание системы. Так, в десятичной системе для записи числа употребляется десять цифр 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, в пятеричной системе достаточно пяти цифр 0, 1, 2, 3, 4. Основание системы — число *пять* — изобразится здесь как 10, поскольку оно является единицей следующего (второго) разряда.

Легко понять, что так будет в любой системе счисления: *основание системы счисления в любой системе записывается как 10.*

В шестнадцатеричной системе имеющихся десяти цифр для изображения числа не хватит. Требуется ввести еще шесть цифр для изображения чисел, равных десяти, одиннадцати, двенадцати, тринадцати, четырнадцати, пятнадцати, которые в шестнадцатеричной системе являются однозначными. Введя для этих чисел обозначения $\overline{0}$, $\overline{1}$, $\overline{2}$, $\overline{3}$, $\overline{4}$, $\overline{5}$, получим 16 цифр, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, $\overline{0}$, $\overline{1}$, $\overline{2}$, $\overline{3}$, $\overline{4}$, $\overline{5}$. Число «шестнадцать» как основание системы будет иметь вид 10.

Чтобы различать, в какой системе счисления записано число, мы будем в дальнейшем указывать основание системы счисления в виде индекса (в десятичной системе!). Например, число 257_{10} записано в обычной десятичной системе, а число 257_8 — в восьмеричной.

Запись целого числа в какой-либо системе счисления означает представление этого числа в виде суммы степеней основания с различными коэффициентами, меньшими основания. Эти коэффициенты и являются цифрами в записи числа. Например, запись числа 1309 в десятичной системе означает

$$1309 = 1 \cdot 10^3 + 3 \cdot 10^2 + 0 \cdot 10^1 + 9 \cdot 10^0.$$

Аналогично шестнадцатеричное число $\overline{1}\overline{0}3\overline{5}$ в десятичной системе равно

$$11 \cdot 16^3 + 0 \cdot 16^2 + 13 \cdot 16^1 + 5 \cdot 16^0.$$

Этим замечанием легко воспользоваться для перевода целого числа из любой системы в десятичную. Оно же может быть использовано для доказательства того, что *любое целое число может быть записано в любой системе счисления.* Рассмотрим сначала доказательство этого утверждения на конкретном примере.

Пример 1.10. Докажем, что десятичное число 1386 можно записать в пятеричной системе счисления. Прежде всего выпишем различные степени числа пять

$$5^0 = 1, \quad 5^1 = 5, \quad 5^2 = 25, \quad 5^3 = 125, \quad 5^4 = 625, \quad 5^5 = 3125.$$

Так как 5^5 уже больше, чем рассматриваемое нами число, то в нем может содержаться лишь четвертая степень пятерки, равная 625. Разделив данное число на 625, найдем в частном 2 и в остатке 136. Таким образом,

$$1386 = 2 \cdot 5^4 + 136.$$

Отметим, что частное должно быть заведомо меньше пяти (иначе число превосходило бы 3125 и 5^4 не было бы самой высокой степенью пятерки, содержащейся в этом числе), а остаток меньше делителя, равного 5^4. Теперь мы можем выделить следующую степень пятерки из остатка.

Разделив 136 на 125, получим в частном 1 и в остатке 11. Поэтому

$$136 = 1 \cdot 5^3 + 11,$$
$$1386 = 2 \cdot 5^4 + 1 \cdot 5^3 + 11.$$

Мы снова могли заранее утверждать, что полученное частное должно быть меньше пяти, а остаток — меньше делителя. Но в нашем случае остаток получился даже меньше 5^2, поэтому

$$1386 = 2 \cdot 5^4 + 1 \cdot 5^3 + 0 \cdot 5^2 + 11,$$

а так как $11 = 2 \cdot 5^1 + 1$, то

$$1386 = 2 \cdot 5^4 + 1 \cdot 5^3 + 0 \cdot 5^2 + 2 \cdot 5^1 + 1 \cdot 5^0.$$

Итак, десятичное число 1386 может быть представлено в виде суммы степеней пяти, т. е. в виде пятеричного числа 21021, так что имеет место равенство

$$1386_{10} = 21021_5.$$

Рассмотренный пример не может служить доказательством приведенного выше общего утверждения. Однако внимательный читатель может заметить, что наш пример содержит все основные элементы этого доказательства, а читатель, имеющий некоторый опыт в математических рассуждениях, сумеет после разбора этого примера провести общее доказательство самостоятельно.

Докажем, что любое целое число N может быть записано в системе счисления с основанием n. В ряду последовательных степеней основания $n^0 = 1, n, n^2, n^3, \ldots, n^k, \ldots$ найдем наибольшую степень, содержащуюся в N, т. е. такую, чтобы $n^k \leqslant N$, но уже $n^{k+1} > N$, что всегда можно сделать. Разделим N на n^k. Если частное от деления равно a_k, а остаток N_k, то $N = a_k n^k + N_k$, причем $a_k < n$ и $N_k < n^k$. Теперь разделим N_k на n^{k-1}, что дает $N_k = a_{k-1} n^{k-1} + N_{k-1}$, откуда $N = a_k n^k + a_{k-1} n^{k-1} + N_{k-1}$, причем снова $a_{k-1} < n$ и $N_{k-1} < n^{k-1}$.

Продолжая этот процесс, дойдем до равенства

$$N_2 = a_1 n + N_1,$$

так что, обозначив $N_1 = a_0$, получим

$$N = a_k n^k + a_{k+1} n^{k-1} + \ldots + a_1 n + a_0.$$

Здесь число N представлено в виде суммы степеней числа n с коэффициентами, меньшими n, что и означает возможность записи числа N в системе с основанием n в виде

$$N = a_k a_{k-1} \ldots a_1 a_0.$$

До сих пор речь шла только о целых числах. Аналогично можно обращаться и с дробями. Основным способом записи дробей при вычислениях являются *десятичные дроби*; десятичная дробь есть

представление дробного числа в виде суммы степеней десяти, но не только положительных, а и отрицательных.

Запись 1309,26 означает число

$$1\cdot 10^3+3\cdot 10^2+0\cdot 10^1+9\cdot 10^0+2\cdot 10^{-1}+6\cdot 10^{-2}.$$

Аналогично, в восьмеричной системе счисления запись 1254,7632 означает число, которое в десятичной системе равно

$$1\cdot 8^3+2\cdot 8^2+5\cdot 8^1+4\cdot 8^0+7\cdot 8^{-1}+6\cdot 8^{-2}+3\cdot 8^{-3}+2\cdot 8^{-4}.$$

Дробь $0{,}7632_8$ естественно называть *восьмеричной дробью*. Известно, что не всякое рациональное число может быть выражено конечной десятичной дробью. Такое же явление наблюдается и в других системах счисления. При этом может случиться, что рациональное число выражается конечной дробью в одной системе и бесконечной (периодической) в другой. Например, число $\frac{1}{3}$ выражается в десятичной системе бесконечной дробью 0,3333..., а в шестеричной — конечной дробью $0{,}2_6$. Наоборот, $\left(\frac{1}{10}\right)_{10}$ в десятичной системе выражается конечной дробью 0,1, а в шестеричной — бесконечной дробью $0{,}03333\ldots_6$.

В системе счисления с основанием n произвольное число M записывается в виде

$$M_n=(a_ka_{k-1}\ldots a_0,\ a_{-1}a_{-2}\ldots a_{-s})_n.$$

Здесь $a_ka_{k-1}\ldots a_1a_0$ — целая часть числа, $a_{-1}a_{-2}\ldots a_{-s}$ — его дробная часть, а a_k, a_{k-1}, ..., a_{-s} — цифры, принимающие значения от 0 до $k-1$.

Развернутая запись этого числа имеет вид

$$M_n=a_kn^k+a_{k-1}n^{k-1}+\ldots+a_1n+a_0n^0+a_{-1}n^{-1}+\ldots+a_{-s}n^{-s}.$$

Производить арифметические действия в непозиционной системе очень неудобно и сложно. В этом легко убедиться, если попробовать, например, сложить числа CCCLIX и CLXXIV или, еще того лучше, перемножить числа CDXVIII и CCLXIV, пользуясь лишь римской системой счисления.

Совсем иное дело арифметические операции в позиционной системе счисления. Здесь дело сводится к соответствующим действиям над однозначными числами в различных разрядах и переносу из младших разрядов в старшие (или наоборот) по определенным правилам. Поэтому как при ручных, так и при машинных расчетах всегда применяются позиционные системы счисления.

Запись чисел в непозиционной системе (римскими цифрами) употребляется теперь очень редко, в основном для нумерации веков в хронологии, глав в книгах и т. п.

В зависимости от основания принятой системы счисления, для изображения числа в машине в каждом разряде требуется различное число элементов или устойчивых состояний элемента. Большинство механических вычислительных устройств работает в десятичной системе счисления. Каждому разряду в них соответствует элемент, имеющий десять различных устойчивых состояний. Например, в русских счетах для изображения одного разряда числа употребляется спица с десятью косточками, в арифмометре — шестерня с десятью зубьями, расположенными по окружности через каждые 36°.

Впрочем, для механических элементов выбор системы счисления не очень важен: для пятеричной системы можно надеть на спицу пять косточек (так и сделано на китайских счетах «суан-пан»), а для двенадцатеричной — сделать шестерню с двенадцатью зубьями — через 30° вместо 36°.

Если же перейти к другим физическим принципам построения машин, то наличие определенного количества устойчивых состояний элемента, используемого для изображения числа, становится очень важным. Для электромеханических и электронных элементов характерно наличие двух устойчивых состояний. Так, например, электромеханическое реле может быть замкнуто или разомкнуто, конденсатор заряжен или разряжен, электронная лампа может проводить или не проводить ток.

Использование элементов, имеющих два различных устойчивых состояния, требует применения системы счисления, использующей только две цифры. Такой является *двоичная система счисления*, т. е. *позиционная система счисления с основанием два.*

Кроме удобства изображения чисел с помощью электронных элементов, двоичная система счисления обладает рядом преимуществ при выполнении арифметических операций. Поэтому конструкция современных электронных вычислительных машин предусматривает в основном представление чисел в двоичной системе счисления.

§ 11. Двоичная арифметика

Двоичная система счисления есть позиционная система с основанием *два.*

Для изображения чисел в этой системе требуется лишь две цифры: 0 и 1. Благодаря этому, числа можно изображать с помощью элементов, имеющих два различных устойчивых состояния: одно из них принимается за изображение нуля, другое — за изображение единицы.

Основание двоичной системы «два» изображается в этой системе как «10». Прибавляя к этому числу единицу, мы получим 11, т. е. двоичную запись числа 3, прибавляя еще единицу, мы должны сделать перенос в двоичные «десятки», а затем в «сотни», что даст 100 — двоичное изображение числа 4.

Первые числа натурального ряда выражаются в двоичной системе следующим образом:

Десятичные числа	Двоичные числа	Десятичные числа	Двоичные числа	Десятичные числа	Двоичные числа	Десятичные числа	Двоичные числа
1	1	6	110	11	1011	16	10 000
2	10	7	111	12	1100	17	10 001
3	11	8	1000	13	1101	18	10 010
4	100	9	1001	14	1110	19	10 011
5	101	10	1010	15	1111	20	10 100

Отсюда видно, что запись числа в двоичной системе значительно длиннее десятичной записи; так, двухразрядное десятичное число 19 записывается пятью двоичными разрядами, одноразрядное число 8 — четырьмя двоичными разрядами. Легко убедиться в том, что для записи произвольного целого числа в двоичной системе требуется в среднем в три раза больше двоичных разрядов, чем в десятичной системе — десятичных. Поэтому при обычном ручном счете десятичная система счисления удобнее двоичной. Однако в электронных машинах существенно не число разрядов (элементов), а общее число устойчивых состояний всех элементов, необходимых для изображения чисел. Можно показать, что с этой точки зрения двоичная система выгоднее десятичной. Так, к примеру, для машины, работающей в десятичной системе счисления, для изображения любого целого числа от 1 до 999 требуется всего 30 устойчивых состояний трех элементов, а в двоичной — лишь 20 устойчивых состояний десяти элементов ($2^{10}=1024$).

Будем считать, что система счисления с основанием α наиболее выгодна для изображения числа в машине, если она требует наименьшего количества N состояний всех элементов, служащих для изображения целых чисел в заданном диапазоне. Найдем основание такой системы.

Если для изображения чисел используются n элементов (n разрядов), то $N=\alpha n$, а наибольшее целое число, которое можно изобразить, $M=\alpha^n$. Поэтому

$$N=\frac{\alpha}{\ln \alpha}\ln M.$$

При заданном диапазоне чисел (заданном M) N будет наименьшим при таком целом α, при котором величина $\frac{\alpha}{\ln \alpha}$ минимальна.

Для нахождения минимума функции $y=\frac{x}{\ln x}$ воспользуемся обычным приемом дифференциального исчисления. Производная этой функции равна

$$y'=\frac{\ln x - x\cdot\frac{1}{x}}{(\ln x)^2}=\frac{\ln x-1}{(\ln x)^2}$$

и обращается в нуль при $\ln x = 1$, т. е. $x = e$. С помощью любого достаточного условия легко проверить, что при этом значении аргумента функция имеет минимум.

Учитывая, что основание системы должно быть целым числом, мы приходим к выводу, что наиболее выгодной для изображения числа в машине является троичная система счисления, почти столь же выгодна двоичная и значительно менее выгодна десятичная.

Так, для изображения всех чисел от 1 до 10^9 в двоичной системе счисления требуется 60 устойчивых состояний, в троичной — 57, в десятичной — 90. Однако из-за трудности осуществления надежных физических элементов с тремя устойчивыми состояниями троичная система не получила широкого распространения *).

Арифметические действия производятся в двоичной системе по обычным правилам.

Например, при сложении складываются соответствующие разряды, начиная с младших. Если в данном разряде образуется сумма, уже не умещающаяся в нем, то соответствующее превышение переносится в следующий старший разряд. Таким образом, фактически приходится использовать таблицу сложения для однозначных чисел, которая в двоичной системе имеет следующий вид:

$$0 + 0 = 0,$$

$$0 + 1 = 1,$$

$$1 + 0 = 1,$$

$$1 + 1 = 10.$$

Сложение двух многозначных двоичных чисел выглядит так:

$$\begin{array}{r} 1011100101 \\ +\quad 10111100 \\ \hline 1110100001 \end{array}$$

Аналогично производится и сложение дробных чисел, например,

$$\begin{array}{r} 10011{,}0101 \\ +\quad 1101{,}0111 \\ \hline 100000{,}1100 \end{array}$$

Если слагаемых много, то в некоторых разрядах может оказаться несколько единиц переноса. В таких случаях их удобнее не держать «в уме», а выписывать сверху над соответствующими разрядами.

*) На троичной системе основана электронная вычислительная машина «Сетунь», сконструированная в Московском университете.

Например, сложение четырех слагаемых будет тогда выглядеть так:

```
  111 1111
1111111111  1
-------------
 101100111, 01
  10111011, 10
   1100001, 01
  11001111, 11
-------------
1101010011, 11
```

Таблица умножения в двоичной системе предельно проста. Поскольку умножение на нуль всегда дает нуль, можно считать, что здесь таблица умножения состоит лишь из одной строки

$$1 \cdot 1 = 1.$$

Но умножение на единицу не меняет числа. Поэтому умножение многозначных чисел сводится к двоичной системе лишь к сдвигу и сложению.

Если в множителе несколько единиц, то между множителями и частичными произведениями полезно оставлять свободное место для записи единиц переноса.

Приведем два примера на умножение:

```
     11011
   ×   100,1
-------------
     11011
  +11011
-------------
   1111001,1
```

```
        111001101
      ×   1110001
-----------------
  11    11
 111111111
-----------------
        111001101
    111001101
+   111001101
  111001101
-----------------
1100101101111101
```

В первом примере без записей единиц переноса легко было обойтись.

Так же просто выполняются и обратные действия — вычитание и деление. Например,

$$\begin{array}{r} 1101001101 \\ -\;\;11100110 \\ \hline 1001100111 \end{array}$$

```
_101011100011111 | 11011
  11011          |----------
 ------          | 110011101
 _100001
   11011
  ------
   _110000
     11011
    ------
    _101011
      11011
     ------
     _100001
       11011
      ------
        _11011
         11011
        ------
```

С помощью специального приема вычитание можно свести к сложению. Для этого нужно воспользоваться понятием двоичного дополнения.

Двоичным дополнением данного положительного двоичного числа называют такое положительное двоичное число, которое, будучи сложено с данным, дает в сумме число, равное ближайшей к данному числу степени двойки (т. е. единице следующего разряда). Например, для числа 1011101 двоичным дополнением будет 100011 потому, что

$$\begin{array}{r} 1011101 \\ +\;\;100011 \\ \hline 10000000 \end{array}$$

а для числа 111001010 — число 110110. Действительно,

$$\begin{array}{r} 111001010 \\ +\;\;\;\;110110 \\ \hline 1000000000 \end{array}$$

Чтобы найти двоичное дополнение данного числа, достаточно вычесть его из единицы старшего разряда. Но легко заметить, что двоичное дополнение можно получить проще: заменить в данном числе все нули единицами, а единицы нулями, после чего прибавить к полученному числу единицу младшего разряда (нули впереди, естественно, можно зачеркнуть). Рекомендуем читателю самостоятельно убедиться на примерах в справедливости этого правила. Пользуясь двоичным дополнением, можно заменить вычитание сложением.

Пример 1.11. Пусть требуется произвести вычитание 1110101100—1000111010. По указанному правилу найдем двоичное дополнение вычитаемого. Оно равно

$$\begin{array}{r} 0111000101 \\ +\quad\quad 1 \\ \hline 111000110 \end{array}$$

Результат вычитания найдется теперь как сумма уменьшаемого и двоичного дополнения вычитаемого, из которой нужно вычесть единицу того разряда, до которого дополнение производилось. В данном случае

$$\begin{array}{r} 1110101100 \\ +\ 111000110 \\ \hline 10101110010 \end{array}$$

Это есть единица одиннадцатого разряда, которую следует просто опустить и мы получаем разность 101110010. Читатель может убедиться в правильности полученного результата самостоятельно.

Приведенные примеры хорошо выясняют преимущества и недостатки двоичной системы счисления. С одной стороны, арифметические действия в двоичной системе очень просты и выполняются автоматически; для представления чисел в машине двоичная система требует меньшего числа элементов, чем любая другая, кроме троичной. С другой стороны, запись чисел в двоичной системе очень громоздка и однообразна: для записи чисел на бумаге в двоичной системе требуется в три-четыре раза больше разрядов, чем в десятичной, и число представляется в виде набора из одних только нулей и единиц.

Преимущества двоичной системы находят применение в конструкции электронных счетных машин. О том, как преодолеть ее недостатки, мы сообщим в следующих параграфах этой главы.

§ 12. Восьмеричная система счисления. Переход от одной системы к другой

Кроме двоичной системы счисления, в программировании используется также восьмеричная система. Причины ее применения будут выяснены в следующем параграфе.

В восьмеричной системе для записи чисел используются цифры 0, 1, 2, 3, 4, 5, 6, 7. Арифметические действия производятся в ней по правилам, общим для позиционных систем. Для этой цели необходимо знать таблицы сложения и умножения одноразрядных чисел. Мы не будем приводить их полностью, полагая, что читатель может составить их самостоятельно. Приведем лишь по одному столбцу из

каждой:

$$\begin{array}{llll} 3+1=4 & 3+5=10 & 5\cdot 1=5 & 5\cdot 5=31 \\ 3+2=5 & 3+6=11 & 5\cdot 2=12 & 5\cdot 6=36 \\ 3+3=6 & 3+7=12 & 5\cdot 3=17 & 5\cdot 7=43 \\ 3+4=7 & & 5\cdot 4=24 & \end{array}$$

Так как в программировании используются различные системы счисления, то нужно уметь быстро переводить числа из одной системы в другую. Основные принципы этого перевода были уже рассмотрены в § 10 при доказательстве возможности записать любое число в любой системе счисления (см. стр. 41).

На практике удобнее пользоваться несколько иным приемом, чем рассмотренный в § 10. Познакомимся с ним сначала на примере.

Пример 1.12. Переведем в восьмеричную систему число 2739_{10}. Разделив данное число на 8, найдем в частном 342 и в остатке 3. Это означает, что наше число, кроме некоторого количества восьмерок, содержит еще три единицы, т. е. последняя цифра восьмеричной записи данного числа есть 3.

Для определения следующей (справа налево) цифры разделим на 8 полученное частное 342. Найдем, что частное равно 42, а остаток 6. Остаток и дает нам вторую справа цифру восьмеричного представления. Все вычисления удобно объединять в следующей схеме:

$$\begin{array}{llll} 2739 & \vert 8 & & \\ \underline{33} & \underline{342} & \vert 8 & \\ 19 & 22 & 42 & \vert 8 \\ \underline{\vert 3 \vert} & \underline{\vert 6 \vert} & \underline{\vert 2 \vert} & \underline{\vert 5 \vert} \end{array}$$

Результат деления можно представить в виде

$$2739=342\cdot 8+3=(42\cdot 8+6)\cdot 8+3=[(5\cdot 8+2)\cdot 8+6]\cdot 8+ \\ +3=5\cdot 8^3+2\cdot 8^2+6\cdot 8^1+3\cdot 8^0.$$

Последнее и означает, что число 2739_{10} имеет в восьмеричной системе вид 5263_8.

Легко догадаться, что описанный прием является общим. Действительно, пусть число N записано в некоторой системе счисления с основанием a и его нужно перевести в новую систему счисления с основанием n. Разделив N на n, получим $N=b_0n+a_0$, причем деление производится в системе счисления с основанием a. Разделив, далее, b_0 на n, найдем, что $b_0=b_1n+a_1$, откуда

$$N=(b_1n+a_1)\,n+a_0=b_1n^2+a_1n+a_0.$$

Этот процесс будем продолжать до тех пор, пока последнее частное, которое мы обозначим через a_k, не станет меньше n. Тогда мы

получим

$$N = a_k n^k + \ldots + a_2 n^2 + a_1 n + a_0,$$

т. е. *цифрами, представляющими число N в системе с основанием n, будут остатки, получающиеся при последовательном делении N на n, записанные в обратном порядке.*

Деление производится в первоначальной системе счисления с основанием a. Если $n < a$, то делитель, а значит, и все остатки однозначны и мы сразу получаем нужные цифры. Если же $n > a$, то в системе с основанием a делитель, и, быть может, некоторые остатки, будут содержать больше одной цифры. В новой системе счисления их следует заменить новыми цифрами, которые в системе с основанием a отсутствовали.

Чтобы пояснить последнее замечание, рассмотрим пример.

Пример 2.12. Переведем число 113451_8 из восьмеричной системы в десятичную. Новое основание в восьмеричной системе равно 12. Деление следует производить в восьмеричной системе:

113451	|12			
54	7435	|12		
45	35	602	|12	
71	11	102	46	|12
7		6	10	3

Теперь мы заметим, что 10_8 в десятичной системе есть 8, а $11_8 = 9$. Поэтому

$$113451_8 = 38697_{10}.$$

Проверим правильность перевода, преобразуя десятичное число 38697 в восьмеричную систему:

38697	|8				
66	4837	|8			
29	37	604	|8		
57	5	44	75	|8	
1		4	3	9	|8
				1	1

Таким образом, $38697_{10} = 113451_8$.

При переводе в другую систему дробных чисел, меньших единицы, действуют иначе.

Пример 3.12. Переведем в двоичную систему десятичную дробь 0,4140625.

В двоичной системе счисления первая цифра после запятой означает половину. Поэтому первая цифра будет единицей или нулем, смотря по тому, будет ли данное число больше или меньше половины. Это легко установить, проверив, будет ли единицей или нулем целая часть числа после его удвоения. Таким же образом можно получить следующие двоичные цифры путем дальнейшего удвоения.

Производимые вычисления удобно записать подряд в виде следующей схемы:

0,	$\times\dfrac{4140625}{2}$
0,	$\times\dfrac{8281250}{2}$
1,	$\times\dfrac{6562600}{2}$
1,	$\times\dfrac{3125000}{2}$
0,	$\times\dfrac{6250000}{2}$
1,	$\times\dfrac{2500000}{2}$
0,	$\times\dfrac{5000000}{2}$
1,	0000000,

причем следует помнить, что удваивается всякий раз только дробная часть числа. Из приведенных вычислений видно, что $0{,}4140625_{10} = 0{,}0110101_2$. Очевидно, что этот прием может быть применен при переводе любого дробного числа, меньшего единицы, из любой системы счисления в любую другую.

Если число является смешанным, т. е. его целая и дробная части отличны от нуля, их следует переводить в другую систему счисления отдельно, каждое в соответствии с рассмотренными выше правилами. Впрочем, в этом случае лучше пользоваться другим, практически наиболее удобным приемом, который будет рассмотрен в § 14.

§ 13. Смешанные системы счисления

Если представить число в какой-либо системе счисления, а затем каждую цифру этого числа записать в другой системе, то мы получим запись числа в *смешанной системе счисления*. Практически используются двоично-десятичная и двоично-восьмеричная системы.

В *двоично-десятичной системе* число представляется в десятичной форме, а затем каждая десятичная цифра записывается в двоичной системе. При этом различные десятичные цифры требуют для

своего двоичного написания различного числа двоичных разрядов, от одного для нуля и единицы до четырех для восьми и девяти. Чтобы не применять никаких разделительных знаков, для двоичного изображения десятичной цифры всегда выделяется четыре двоичных разряда. Такая группа из четырех двоичных разрядов, предназначенная для изображения одной десятичной цифры, называется *тетрадой*.

Десятичное число требует для своего изображения в двоично-десятичной форме стольких тетрад, каково количество десятичных разрядов числа. Например, число 3842 в двоично-десятичной системе будет иметь вид 0011 1000 0100 0010. Для удобства чтения мы записали здесь тетрады с промежутками между ними. На самом деле все цифры могут быть поставлены рядом и надо только помнить, что каждая группа состоит из четырех разрядов. Например, двоично-десятичная запись 01011001011000000010 означает десятичное число 59602.

Из возможных шестнадцати различных тетрад 0000, 0001, 0010, ..., 1110, 1111 в двоично-десятичной системе используются только первые десять; остальные тетрады не означают никакой десятичной цифры и поэтому не имеют смысла в двоично-десятичной системе. По этой причине арифметические операции в двоично-десятичной системе затруднительны.

Например, при сложении $56+23$ в двоично-десятичной системе можно действовать так же, как в двоичной. Мы получаем

$$\begin{array}{r} 01010110 \\ +\ 00100011 \\ \hline 01111001 \end{array}$$

т. е. 79 в десятичной системе. Если же нужно складывать $56+25$, то сложение в двоичной системе дает

$$\begin{array}{r} 01010110 \\ +\ 00100101 \\ \hline 01111011 \end{array}$$

где последняя тетрада не имеет десятичного смысла. Сумма, равная 81, должна иметь в двоично-десятичной системе вид 10000001.

Как мы уже говорили, в электронной машине числа записываются обычно в двоичной системе счисления. Однако исходные данные для счета на машине, естественно, всегда представляются в десятичной системе. Двоично-десятичная система записи и служит для записи десятичных чисел в машине. Перед вводом в машину десятичные числа записывают в двоично-десятичной системе, а затем уже в самой машине двоично-десятичные числа переводятся в двоичные. Результаты счета на машине получаются в двоичном виде, полученные двоичные числа в машине переводятся в двоично-десятичные, а уже затем в десятичные.

Аналогично двоично-десятичной системе, *двоично-восьмеричная* форма записи числа состоит в том, что число переводится в восьмеричную систему, а затем каждая восьмеричная цифра записывается в двоичном виде. Так как наибольшая цифра в восьмеричной системе есть 7, то для изображения любой восьмеричной цифры достаточно трех разрядов. Такая группа из трех разрядов, отведенная для изображения восьмеричной цифры, называется *триадой*.

Так, например, восьмеричное число 7102563 запишется в двоично-восьмеричной системе семью триадами 111 001 000 010 101 110 011.

В отличие от тетрад, используемых в двоично-десятичной системе, в двоично-восьмеричной все возможные триады используются. Это обстоятельство составляет преимущество двоично-восьмеричной системы. Другим, и самым важным, преимуществом является то, что *двоично-восьмеричная запись числа совпадает с его двоичной записью.*

Убедимся сначала в справедливости этого утверждения на примере.

Пример. 1.13. Пусть дано число 2736_{10}. Переведя его в двоичную систему

```
2736 | 2
----  ----
   0  1368 | 2
      ----  ---
         0  684 | 2
            ---  ---
              0  342 | 2
                 ---  ---
                   0  171 | 2
                      ---  ---
                        1   85 | 2
                            --  ---
                             1  42 | 2
                                --  ---
                                 0  21 | 2
                                    --  ---
                                     1  10 | 2
                                        --  ---
                                         0   5 | 2
                                            --  ---
                                             1   2 | 2
                                                --  ---
                                                 0   1
```

найдем двоичную форму 101010110000_2. При переводе этого числа в восьмеричную систему получим

```
2736 | 8
----  ---
   0  342 | 8
      ---  ---
        6   42 | 8
            --  ---
             2   5
```

т. е. 5260_8. Записав это восьмеричное число в двоично-восьмеричной форме, будем иметь 101010110000, что совпадает с полученным выше двоичным представлением.

Не представляет труда доказать высказанное утверждение в общем виде. Для определенности ограничимся рассмотрением двоичного числа с шестью разрядами до и тремя после запятой. Пусть оно имеет вид

$$N = b_5 b_4 b_3 b_2 b_1 b_0, b_{-1} b_{-2} b_{-3}.$$

В десятичной системе это число равно

$$N = b_5 2^5 + b_4 2^4 + b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0 2^0 + \\ + b_{-1} 2^{-1} + b_{-2} 2^{-2} + b_{-3} 2^{-3}.$$

Представим его в виде

$$N = (b_5 2^2 + b_4 2^1 + b_3 2^0) \cdot 2^3 + (b_2 2^2 + b_1 2^1 + b_0 2^0) 2^0 + . \\ + (b_{-1} 2^2 + b_{-2} 2^1 + b_{-3} 2^0) 2^{-3} = a_1 8^1 + a_2 8^0 + a_{-1} 8^{-1},$$

где

$$a_1 = b_5 2^2 + b_4 2^1 + b_3 2^0,$$

$$a_2 = b_2 2^2 + b_1 2^1 + b_0 2^0,$$

$$a_3 = b_{-1} 2^2 + b_{-2} b^1 + b_{-3} 2^0.$$

Из представления

$$N = a_1 8^1 + a_0 8^0 + a_{-1} 8^{-1}$$

следует, что восьмеричными цифрами числа N являются a_1, a_0, a_{-1}. Вместе с тем равенство $a_1 = b_5 2^2 + b_4 2^1 + b_3 2^0$ показывает, что если записать восьмеричную цифру a_1 в виде триады двоичных цифр, то этими двоичными цифрами будут b_5, b_4, b_3. Поскольку аналогичные равенства имеют место также и для a_0, a_{-1}, тем самым доказано, что двоичная запись числа совпадает с двоично-восьмеричной записью.

Доказанное утверждение объясняет применение восьмеричной системы при программировании.

Элементы машины имеют два устойчивых состояния, и число в машине изображается в двоичной системе. Однако количество разрядов в двоичном числе обычно бывает довольно велико, а человеку иметь дело с длинным набором одних только нулей и единиц затруднительно.

Поэтому для сокращения записи двоичных чисел в программировании используют следующий способ. Двоичное число разбивают на тройки двоичных разрядов — триады, а затем каждую триаду заменяют соответствующей восьмеричной цифрой. От этого число становится втрое короче, его запись значительно более разнообраз-

ной, в результате чего облегчается переписывание и уменьшается вероятность описок и просчетов. В силу доказанного утверждения формально преобразованное таким образом число является восьмеричным представлением исходного двоичного числа.

Восьмеричную систему удобно использовать при переводе чисел из десятичной системы в двоичную вручную *). Для этого сначала переводят десятичное число в восьмеричную систему, а затем каждую восьмеричную цифру заменяют соответствующей триадой. Аналогично можно производить и обратный перевод из двоичной системы в десятичную. В этом случае следует, разбив двоичное число на триады, получить его восьмеричное изображение, а затем восьмеричное число перевести в десятичное.

Пример 2.13. Переведем в двоичную систему число $940{,}6875_{10}$. Производим перевод в восьмеричную систему отдельно целой и дробной частью числа:

$$\begin{array}{r|l} 940 & 8 \\ \hline 4 & 117 \quad 8 \\ \hline & 5 \quad 14 \quad 8 \\ \hline & \quad 6 \quad 1 \end{array} \qquad \begin{array}{r|l} 0,6875 & \\ \hline & 8 \\ \hline 5, & 5000 \\ & 8 \\ \hline 4, & 0000 \end{array}$$

Таким образом, $940{,}6875_{10} = 1654{,}54_8$. Отсюда $940{,}6875_{10} = 1110\ 101\ 100{,}1011_2$.

Проведем обратный перевод:

$$1110\ 101\ 100{,}1011_2 = 1654{,}54_8,$$

$$\begin{array}{r|l} 1654 & 12 \\ \hline 45 & 136 \quad 12 \\ \hline 74 & 16 \quad 11 \\ \hline 0 & 4 \end{array} \qquad \begin{array}{r|l} 0, & 54 \\ & 12 \\ \hline 6, & 70 \\ & 12 \\ \hline 10, & 60 \\ & 12 \\ \hline 7, & 40 \\ & 12 \\ \hline 5, & 00 \end{array}$$

что дает $940{,}6875_{10}$.

Очевидно, этот способ значительно сокращает записи по сравнению с непосредственным переводом из десятичной системы в двоичную и обратно.

*) Со способом перевода десятичных чисел в двоичную систему на машине мы познакомимся в гл. VII. Там же будет рассмотрен и способ обратного перевода.

Рассмотренные особенности восьмеричной системы объясняются тем, что восемь есть степень двух. Точно так же можно было бы пользоваться шестнадцатеричной и двоично-шестнадцатеричной системами, в которой вместо триад были бы тетрады.

В отличие от двоично-десятичной системы, в двоично-шестнадцатеричной все возможные тетрады используются.

§ 14. Формы представления чисел

Как и при ручном счете, числа в машине могут записываться различными способами — и с фиксированной, и с плавающей запятой. В первой части мы уже говорили, что запись числа в форме с плавающей запятой означает его представление в виде двух множителей

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

Такая запись имеет смысл в любой системе счисления. При этом 10 означает основание системы счисления.

При нормализованной записи числа на мантиссу накладывается ограничение

$$0{,}1 \leqslant |N_0| < 1.$$

При этом 0,1 не обязательно означает десятую, а есть 10^{-1}, где 10 — основание системы счисления, в которой число записано.

Следовательно, в двоичной системе мантисса должна быть не меньше половины, а в восьмеричной — не меньше одной восьмой. В любой системе это неравенство означает, что мантисса числа меньше единицы, но ее первая цифра после запятой отлична от нуля. При этом порядок числа может быть как положительным, так и отрицательным.

Например, десятичные числа 0,00379; 0,0379; 0,379; 3,79; 37,9 в нормализованной форме записываются так: $0{,}379 \cdot 10^{-2}$, $0{,}379 \cdot 10^{-1}$; $0{,}379 \cdot 10^{0}$; $0{,}379 \cdot 10^{1}$; $0{,}379 \cdot 10^{2}$. Аналогично записывается число в плавающей нормализованной форме и в двоичной системе счисления. Например, число $26{,}875_{10} = 11010{,}111_2$ имеет нормальную форму $11010{,}111 = 0{,}11010111 \cdot 10^{101}$.

Как уже говорилось, вместо двоичной записи чисел обычно применяют восьмеричную. Заменив каждую триаду нашего числа восьмеричной цифрой (для этого понадобится еще приписать справа один нуль), получим

$$0{,}656 \cdot 10^5.$$

При этом надо помнить, что в такой записи *порядок числа является двоичным*, хотя и записан в восьмеричной форме. В восьмеричной системе число $26{,}875_{10} = 32{,}7_8$ имеет порядок 2, но не 5.

Приведенный пример показывает, что в нормальной форме двоичная и двоично-восьмеричная записи числа могут не совпадать.

Действительно, для нашего примера запись числа $26,875_{10}$ в нормальной восьмеричной форме имеет вид $0,327 \cdot 10^2$.

Легко понять в чем здесь дело, если перейти от этой восьмеричной записи к двоично-восьмеричной, заменив восьмеричные цифры триадами, а основание 10_8 основанием 10_2 (порядок можно по-прежнему иметь в восьмеричной форме). Тогда мы получим $0,011\ 010\ 111 \cdot 10^6$.

Итак, получилась мантисса, состоящая из тех же значащих цифр, но первая ее цифра есть нуль, т. е. число получилось ненормализованным. Чтобы его нормализовать, нужно сдвинуть запятую на один разряд вправо, соответственно уменьшив порядок на единицу. Это равносильно умножению числа на два.

Так как мантисса числа всегда меньше единицы, то при переводе десятичного числа в восьмеричную систему нужно последовательно умножать его на 8. Однако первый раз множитель приходится изменять таким образом, чтобы получить сразу нормализованное число. В рассмотренном примере для числа $26,875_{10}$ этот первый множитель равен 0,25.

Первые множители для различных чисел различны и зависят от двоичного порядка. Оба они определяются величиной переводимого десятичного числа. Для достаточно широкого диапазона чисел их можно найти в приводимой нами таблице (см. табл. 1.14). В случае необходимости таблицу можно самостоятельно продолжить в обе стороны.

Таблица 1.14

Интервал десятичных чисел	Двоичный порядок в восьмеричной форме	Первый множитель
$0,0078125 \leqslant N < 0,015625$	—6	512
$0,015625 \leqslant N < 0,03125$	—5	256
$0,03125 \leqslant N < 0,0625$	—4	128
$0,0625 \leqslant N < 0,125$	—3	64
$0,125 \leqslant N < 0,25$	—2	32
$0,25 \leqslant N < 0,5$	—1	16
$0,5 \leqslant N < 1$	0	8
$1 \leqslant N < 2$	1	4
$2 \leqslant N < 4$	2	2
$4 \leqslant N < 8$	3	1
$8 \leqslant N < 16$	4	0,5
$16 \leqslant N < 32$	5	0,25
$32 \leqslant N < 64$	6	0,125
$64 \leqslant N < 128$	7	0,0625
$128 \leqslant N < 256$	10	0,03125
$256 \leqslant N < 512$	11	0,015625
$512 \leqslant N < 1024$	12	0,0078125
$1024 \leqslant N < 2048$	13	0,00390625
$2048 \leqslant N < 4096$	14	0,001953125
$4096 \leqslant N < 8192$	15	0,0009765625

Таким образом, для перевода десятичного числа в нормализованную двоично-восьмеричную форму нужно умножить это число сначала на множитель, взятый из таблицы, а затем последовательно дробные части получающихся произведений умножать на 8. Целые части получающихся произведений образуют последовательные цифры мантиссы. Порядок числа определяется той же таблицей.

Пример 1.14. Переведем в нормализованную двоично-восьмеричную форму десятичное число $N=940{,}6875$. Из таблицы находим, что для строки $512 \leqslant N < 1024$ двоичный порядок в восьмеричной форме $p=12$, а первый множитель равен 0,0078125. Умножая N на этот множитель, получим 7,34912109375. Последовательным умножением на 8 и выделением целых частей находим

7,	34912109375
	8
2,	79296875000
	8
6,	34375000
	8
2,	75000
	8
6,	00

Таким образом, в нормализованном двоично-восьмеричном виде получаем $N=0{,}72626 \cdot 10^{12}$.

ГЛАВА IV

ЭЛЕМЕНТЫ ПРОГРАММИРОВАНИЯ

§ 15. Основные устройства электронной счетной машины

Процессу вычислений, который производится человеком, свойственны следующие основные элементы:

1. Хранение информации. Здесь под информацией подразумеваются исходные данные, промежуточные и окончательные результаты счета, а также *алгоритм вычислений*, т. е. способ счета и формулы, различного рода условия и т. п. Эта информация человеком частично запоминается, частично записывается на бумаге. Часть информации берется из различных справочников и таблиц. Память человека, бумага, справочники и таблицы являются различными видами запоминающих устройств.

2. Обработка информации. Для процесса вычислений это означает просто выполнение арифметических действий над числами. Обработка информации производится либо вручную (в уме), либо при помощи специальных счетных приборов, например, логарифмической линейки или арифмометра. При этом производится обмен информацией между устройством, предназначенным для выполнения арифметических действий, и запоминающим устройством: исходные данные берутся с листа бумаги, переносятся на арифмометр (или линейку), а затем результат вычислений снова записывается на бумаге или запоминается человеком.

3. Управление вычислительным процессом. Оно производится человеком. В соответствии с алгоритмом вычислитель производит определенные операции в определенной последовательности, каждый раз решая, какие числа и в каком порядке переносить с бумаги на арифмометр или обратно.

Автоматическая программно-управляемая вычислительная машина должна быть устроена так, чтобы все перечисленные выше элементы процесса вычислений осуществлялись в ней без участия человека во время ее работы. В соответствии с этим требованием вычислительная машина должна содержать различные устройства, осуществляющие эти элементы процесса.

Основными частями электронной счетной машины являются:
а) запоминающее устройство (*память*);
б) арифметическое устройство (*арифметика*);
в) устройство управления (*управление*);
г) устройства ввода и вывода (*ввод* и *вывод*).

Память машины предназначена для хранения всей требуемой информации, т. е. не только исходных, промежуточных и окончательных числовых значений, но также и алгоритма счета.

Памятью обладают все вычислительные устройства. Простые устройства, например арифмометр, обладают памятью в виде трех-четырех регистров (счетчиков) для исходных данных и результатов счета. В памяти современных электронных машин можно поместить тысячи и даже десятки тысяч чисел. Таким образом, память электронной машины отличается от памяти арифмометра неизмеримо большим объемом. Кроме этого, в памяти электронной машины помещается, помимо числового материала, полный алгоритм счета, записанный в специальной форме. Его называют *программой* решения задачи.

Арифметика машины предназначена для переработки информации, т. е. для выполнения операций над числами, которые поступают туда из памяти. Это могут быть не только четыре основные арифметические операции, но и ряд других, о которых речь будет идти ниже.

Главным отличием арифметики электронной счетной машины от арифметических устройств других счетных приборов является громадное быстродействие. Современные машины способны выполнять десятки тысяч арифметических операций в секунду. Конструируются машины с быстродействием в миллионы операций в секунду.

Управление вызывает из памяти сведения о выполнении очередной операции, расшифровывает их, вызывает из памяти нужные числа и засылает их в арифметическое устройство, а затем отсылает в память полученный результат, т. е. управляет ходом вычислительного процесса в соответствии с программой.

Ввод и *вывод* предназначены для общения человека с машиной и позволяют вводить в машину исходный материал и выводить из нее результаты счета.

Общее представление о связи перечисленных устройств дает рис. 8, содержащий схему соединения основных частей (*блок-схему*) электронной счетной машины. Проследим с помощью рис. 8 порядок решения задачи на машине.

Исходные числовые данные и программа задачи наносятся на перфоленту или перфокарты (бумажная или зачерненная кинолента или картонные карты, на которых цифры изображаются пробитыми в нужных местах отверстиями) и вводятся через устройство ввода в память. Через устройство управления в машину подается

сигнал на начало счета. *Управление* вызывает из *памяти* сведения о первой элементарной операции программы и пересылает в *арифметику* исходные числа для данной операции и сведения, какую операцию над ними следует выполнить. После выполнения в *арифметике* этой операции *управление* пересылает результат снова в *память*, а затем извлекает оттуда сведения о следующей операции. Окончательные результаты счета из *памяти* по сигналу *управления* поступают в устройство вывода, которое печатает их на бумажной ленте. Полученный таким образом числовой материал принято называть *табулограммой*.

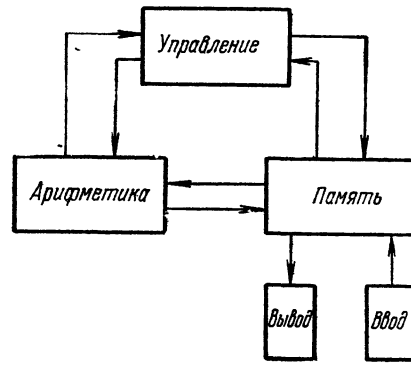


Рис. 8.

§ 16. Команда в трехадресной машине

Память электронной счетной машины состоит из большого числа N одинаковых ячеек, пронумерованных подряд от 0 до $N-1$. Номер ячейки называется ее *адресом*. Каждая ячейка содержит определенное число n двоичных разрядов. Поэтому в ячейке памяти может быть помещен набор из n нулей или единиц или n-разрядное целое двоичное число, которое мы в дальнейшем будем называть *словом*. *Слово* может быть использовано как число, как команда, либо какими-нибудь другими способами.

Ячейка памяти машины обладает той особенностью, что записанное в ней слово может храниться в ней и прочитываться любое число раз до тех пор, пока в эту ячейку не придется записать новое содержимое. При записи в ячейку предварительно стирается прежнее содержимое, при чтении слова из ячейки оно сохраняется там без изменения.

Рассмотрим более подробно использование слова как команды.

Арифметика каждой машины может выполнять определенное число *элементарных операций*. Для работы машины алгоритм решения задачи следует расписать в виде последовательности таких элементарных операций. Такую последовательность и называют *программой*. Программа состоит из *команд*, каждая из которых содержит сведения об одной элементарной операции.

Команда занимает одну ячейку памяти и состоит из двух основных частей — *операционной* и *адресной*. Операционная часть команды содержит сведения о характере операции. Для этой цели все элементарные операции машины нумеруются и в операционной части

команды записывается номер операции, которую следует выполнить. Этот номер называют *кодом операции.*

Адресная часть команды содержит адреса (номера) ячеек, содержимое которых участвует в операции. У различных машин число адресов в адресной части команды может быть различным. Мы ограничимся рассмотрением *трехадресных машин,* в адресной части которых указаны *адреса трех ячеек памяти.* Их называют первым, вторым и третьим адресами. В первом и втором адресах записываются номера ячеек памяти, где находятся исходные числа, над которыми нужно совершить данную операцию. Третий адрес указывает номер ячейки, куда следует записать результат *).

В процессе работы машины команда, находящаяся в некоторой ячейке памяти, извлекается устройством управления и расшифровывается. В соответствии с кодом операции, арифметическое устройство выполняет нужную операцию. Затем управление вызывает из памяти и направляет в арифметику требуемые числа, а после выполнения операции отсылает в память полученный результат, после чего извлекает из памяти следующую команду.

Для арифметических операций *следующая команда* всегда означает *команду, лежащую в следующей ячейке,* т. е. в ячейке, адрес которой на единицу больше. Поэтому для правильного выполнения программы последовательные команды должны размещаться в ячейках памяти подряд.

§ 17. Арифметические операции. Расписка формулы по командам

Основными элементарными операциями любой электронной машины являются арифметические: сложение, вычитание, умножение и деление. Эти операции производятся в машине над нормализованными числами с плавающей запятой.

Арифметические команды в трехадресной машине естественно записывать следующим образом:

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

Здесь через a, b, c обозначены адреса ячеек памяти, содержимое которых также обозначено через a, b, c.

К примеру, запись $a:b=c$ означает команду: число из ячейки, адрес которой обозначен через a, разделить на число из ячейки b, а частное поместить в ячейку c.

*) Мы описываем структуру команды, соответствующей арифметической операции. В других случаях использование адресов может быть иным.

Простейшая задача, которую может решать электронная машина, есть вычисление алгебраического выражения, образующегося при помощи четырех арифметических действий. Составить программу расчета означает здесь расписать последовательность соответствующих элементарных операций.

Пример 1.17. В качестве первого примера рассмотрим вычисление значения квадратного трехчлена:

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

Сделаем расписку этой формулы по элементарным операциям:

$$\begin{aligned} &1) \quad b \cdot x = R_1 \\ &2) \quad R_1 + c = R_2 \\ &3) \quad x \cdot x = R_3 \\ &4) \quad a \cdot R_3 = R_4 \\ &5) \quad R_2 + R_4 = y. \end{aligned}$$

По такой схеме, в которой через R_1, R_2, R_3, R_4 обозначены результаты промежуточных действий, можно производить вычисления на арифмометре или на клавишной счетной машине. Эта же схема может служить и программой вычисления квадратного трехчлена на трехадресной машине.

Предположим, что числовые величины x, a, b помещены в некоторые ячейки памяти машины; обозначим адреса этих ячеек теми же буквами, что и соответствующие величины. Кроме того, обозначим через R_1, R_2, R_3, R_4 адреса ячеек, в которых в машине будут записываться результаты промежуточных действий, а через y — адрес ячейки для значения квадратного трехчлена. Тогда формулу

$$1) \quad b \cdot x = R_1$$

можно рассматривать как трехадресную команду: перемножить числа из ячеек b и x и произведение поместить в ячейку R_1. Аналогично, формулы 2)—5) можно рассматривать как арифметические операции над числами, находящимися в соответствующих ячейках.

Поместим команды 1), 2), 3), 4), 5) в пять ячеек памяти подряд и заставим машину последовательно выполнить эти команды. Тогда в ячейке y окажется значение квадратного трехчлена. После команды 5) следует поставить команду остановки машины, которая записывается так:

$$6) \quad стоп$$

(все три адреса у этой команды нулевые).

В рассмотренной простейшей программе разные ячейки памяти используются по-разному: шесть ячеек, в которых записаны команды 1)—6), называются *командными*; ячейки для исходных данных x, a,

b, c — *аргументами* программы; ячейки для промежуточных величин R_1, R_2, R_3, R_4 — *рабочими ячейками*, y — ячейкой *результата*. Всего программа занимает 15 ячеек:

1), 2), 3), 4), 5), 6), a, b, c, x, y, R_1, R_2, R_3, R_4,

которые могут быть расположены в любых местах памяти. Только командные ячейки 1)—6) должны быть расположены подряд.

При составлении программы счета квадратного трехчлена можно сократить количество рабочих ячеек от четырех до одной, если записывать промежуточные результаты в одну рабочую ячейку R и ячейку результата y:

1) $b \cdot x = y$
2) $y + c = y$
3) $x \cdot x = R$
4) $a \cdot R = R$
5) $y + R = R$
6) *стоп.*

Обратим внимание на команды 2), 4), 5), в которых одна и та же ячейка встречается и слева, и справа. Например, в команде 2) написано $y + c = y$. Если бы здесь буква y означала определенную величину, то такое равенство означало бы, что $c = 0$. На самом деле это не так, потому что в нашей записи команды 2) буква y означает адрес ячейки, которая отведена для величины y, а равенство $y + c = y$ не есть арифметическое равенство, а есть записанная в привычных арифметических обозначениях команда для машины: числа из ячеек, адреса которых обозначены буквами c и y, сложить и результат записать в ячейку y.

Можно вообще обойтись без рабочих ячеек, составляя программу по формуле $y = x(ax + b) + c$:

1) $a \cdot x = y$
2) $y + b = y$
3) $x \cdot y = y$
4) $y + c = y$
5) *стоп.*

Таким образом, по сравнению с первоначальным вариантом мы сократили объем памяти, занимаемой программой, на пять ячеек: избавились от четырех рабочих ячеек и уменьшили количество команд с шести до пяти.

Пример 2.17. Составить программу вычисления алгебраического выражения:

$$z = \frac{5x^2y - 3}{x^3 - 2xy + 4y^2}.$$

Обозначим через «3», «4», «5» адреса ячеек, в которых расположены целые числа 3, 4, 5; через x^2, x^3, y^2, xy — адреса ячеек для величин x^2, x^3, y^2, xy; через R_1 и R_2 — ячейки, в которых образуются числитель и знаменатель (x^2, x^3, y^2, xy, R_1, R_2 — рабочие ячейки).

Программа вычисления z имеет вид:

1)	x	$\cdot$	$x=x^2$	
2)	x^2	$\cdot$	$y=R_1$	
3)	«5»	$\cdot$	$R_1=R_1$	
4)	R_1	$-$	«3» $=R_1$	(числитель)
5)	x^2	$\cdot$	$x=x^3$	
6)	y	$\cdot$	$y=y^2$	
7)	«4»	$\cdot$	$y^2=R_2$	
8)	x^3	$+$	$R_2=R_2$	
9)	x	$\cdot$	$y=xy$	
10)	R_2	$-$	$xy=R_2$	
11)	R_2	$-$	$xy=R_2$	(знаменатель)
12)	R_1	:	$R_2=z$	
13)	*стоп.*			

Можно сократить число рабочих ячеек с шести до трех, число констант с трех до двух и число команд с тринадцати до одиннадцати, если вычисления производить по формуле

$$z=\frac{5x^2y-3}{x^3+2y(2y-x)},$$

помещая числитель в ячейку результата z, знаменатель в рабочую ячейку u, а промежуточные величины в рабочие ячейки R_1 и R_2:

1)	x	$\cdot$	$x=R_1$	
2)	R_1	$\cdot$	$y=z$	
3)	«5»	$\cdot$	$z=z$	
4)	z	$-$	«3» $=z$	(числитель)
5)	R_1	$\cdot$	$x=u$	
6)	y	$+$	$y=R_1$	
7)	R_1	$-$	$x=R_2$	
8)	R_1	$\cdot$	$R_2=R_1$	
9)	u	$+$	$R_1=u$	(знаменатель)
10)	z	:	$u=z$	
11)	*стоп.*			

При помощи четырех арифметических операций могут быть составлены программы решения простейших алгебраических уравнений.

Пример 3.17. Составить программу решения системы двух линейных алгебраических уравнений:

$$\left.\begin{aligned} a_1x+b_1y&=c_1,\\ a_2x+b_2y&=c_2. \end{aligned}\right\}$$

Решение этой системы имеет вид

$$x=\frac{D_1}{D},\quad y=\frac{D_2}{D},$$

где

$$\begin{aligned} D&=a_1b_2-a_2b_1,\\ D_1&=b_2c_1-b_1c_2,\\ D_2&=a_1c_2-a_2c_1. \end{aligned}$$

Программа нахождения x, y с пятью рабочими ячейками R_1, R_2, D, D_1, D_2 имеет следующий вид:

$$\begin{aligned} a_1 &\cdot b_2=R_1\\ a_2 &\cdot b_1=R_2\\ R_1 &-R_2=D\\ b_2 &\cdot c_1=R_1\\ b_1 &\cdot c_2=R_2\\ R_1 &-R_2=D_1\\ a_1 &\cdot c_2=R_1\\ a_2 &\cdot c_1=R_2\\ R_1 &-R_2=D_2\\ D_1 &: D=x\\ D_2 &: D=y\\ &\textit{стоп.} \end{aligned}$$

Читателю предоставляется возможность составить программу, использующую лишь две рабочие ячейки.

В некоторых трехадресных машинах элементарными являются еще две арифметические операции: вычитание абсолютных величин и извлечение корня. Эти операции записываются так:

$$\begin{aligned} |a|-|b|&=c,\\ \sqrt{a}\quad\ &=c. \end{aligned}$$

При операции извлечения квадратного корня второй адрес не используется (это двухадресная операция).

Пример 4.17. Вычислить величину:

$$z=\left(|x|-\left|\frac{x}{y}\right|\right)\cdot\sqrt{|x+2|}$$

и поместить ее в две ячейки, z_1 и z_2.

Аргументами программы являются ячейки x, y, «2», результатами — ячейки z_1, z_2. Составим программу, воспользовавшись двумя рабочими ячейками R, z и ячейкой «0», в которую поместим число нуль:

1) x : $y=R$
2) $|x|$ — $|R|=z$
3) x + «2» $=R$
4) $|R|-|$«0»$|=R$
5) $\sqrt{R}$ $=R$
6) R · $z=z$
7) «0» + $z=z_1$
8) «0» + $z=z_2$
9) *стоп.*

В этом примере легко можно было бы уменьшить программу на одну команду и обойтись без рабочих ячеек R и z, помещая промежуточные величины в результирующие ячейки z_1 и z_2. Отметим, однако, что экономию ячеек памяти производят только тогда, когда это необходимо. Часто важнее не сэкономить несколько ячеек, а сохранить в программе обозначения, более близкие к формульной записи программируемого выражения.

§ 18. Разветвляющиеся вычислительные процессы. Команды передачи управления

В большинстве вычислительных процессов мы сталкиваемся с тем, что выбор хода дальнейших вычислений определяется результатами предыдущих. Более точно можно сказать, что вычислительный процесс разбивается на ряд этапов, и переход от одного этапа к другому зависит от выполнения некоторых условий. Проверка выполнения этих условий тоже может рассматриваться как некоторый этап вычислительного процесса, хотя он и не требует выполнения арифметических операций.

Например, процесс нахождения корней квадратного уравнения $ax^2+bx+c=0$ можно разбить на следующие этапы:

1) вычисление дискриминанта уравнения $D=b^2-4ac$,
2) проверка знака дискриминанта,
3) вычисление действительных корней уравнения (производится при $D\geqslant 0$),

4) вычисление мнимых корней уравнения (производится при $D<0$).

Первый этап состоит из арифметических операций. Второй является проверкой логического условия и арифметических операций не требует. Это логическое условие можно формулировать в виде: «верно ли, что дискриминант уравнения неотрицателен?», или «выполняется ли условие $D\geqslant 0$?». В зависимости от результатов вычислений, которые были произведены в первом этапе, мы получаем один из двух возможных ответов: «да» или «нет». В случае ответа «да» дальнейшие вычисления производятся по формулам, соответствующим третьему этапу — вычислению действительных корней. При ответе «нет» следует обратиться к формулам четвертого этапа.

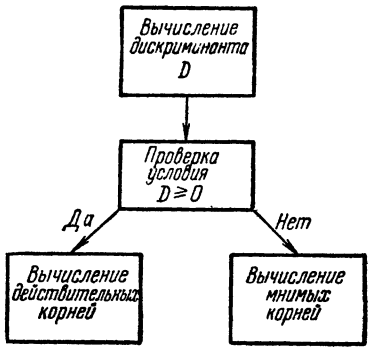


Рис. 9.

Разбиение этого процесса на этапы можно наглядно представить схемой, изображенной на рис. 9. Такие схемы называются *блок-схемами вычислительного процесса.*

Здесь мы имеем дело с вычислительным процессом, разветвляющимся в двух направлениях. Если допустить, что коэффициент a квадратного уравнения может обращаться в нуль, то вычислительный процесс будет разветвляться уже в трех направлениях. Его блок-схема имеет вид, показанный на рис. 10.

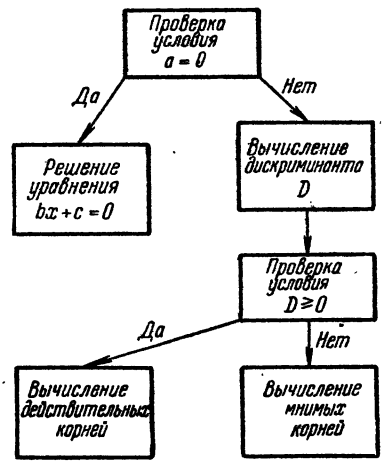


Рис. 10.

Другим примером разветвляющегося процесса является задача нахождения наибольшего из нескольких чисел. В процессе ее решения вообще не приходится выполнять вычислительных операций. Нужно только, перебирая по очереди числа, сравнивать их попарно между собой и запоминать большее число из пары.

К разветвляющимся процессам приводит и вычисление ряда физических величин. Например, при движении тела в воздухе величина силы сопротивления считается по-разному, в зависимости от скорости движения; при малых скоростях сила сопротивления пропорциональна первой степени скорости, а при больших — второй.

При ручном счете проверка выполнения логических условий, определяющая дальнейшее направление вычислительного процесса, производится вычислителем. Электронная счетная машина работает автоматически, т. е. без участия человека, и с очень большой скоростью. Поэтому в ней должна быть заложена возможность с помощью программы производить проверку выполнения логических условий и осуществлять, в зависимости от результатов этой проверки, разветвление вычислительного процесса.

Прежде всего в машине должен вырабатываться некоторый признак, показывающий, выполняется или не выполняется проверяемое условие. Для этой цели в машине имеется специальный регистр, в котором может быть записан 0 или 1. Содержание этого регистра называют *управляющим сигналом* или *сигналом* ω.

Проверяемому логическому условию можно придать форму утверждения о знаке или абсолютной величине некоторого получающегося результата арифметического действия *), например, «верно ли, что $D \geqslant 0$?», как при решении квадратного уравнения. Так как для этой цели может быть использован любой результат арифметического действия, то управляющий сигнал ω вырабатывается в машине при каждой арифметической операции.

Операции сложения, вычитания и вычитания абсолютных величин вырабатывают сигнал $\omega = 1$, если результат операции отрицателен, и $\omega = 0$, если он неотрицателен. После выполнения операций умножения, деления и извлечения квадратного корня вырабатывается сигнал $\omega = 0$, если результат не превосходит единицы по абсолютной величине, и $\omega = 1$ в противоположном случае.

Можно считать, что проверка выполнения логического условия (имеющего указанную выше форму) происходит одновременно с выполнением операции. К моменту окончания операции условие уже проверено и в машине выработался соответствующий признак. Однако мало получить этот признак, нужно еще иметь возможность им воспользоваться. Для этой цели в числе элементарных операций машины предусмотрены *операции условной передачи управления*.

Поясним сначала терминологию. Если машина выполняет команду, взятую из ячейки памяти с номером r, то говорят, что *управление находится в ячейке* r. Как было сказано в § 16, после выполнения арифметической операции машина переходит к выполнению команды, находящейся в следующей ячейке. Это означает, что арифметическая команда *всегда передает управление следующей ячейке*, так что естественный порядок выполнения команд сохраняется. *Операциями передачи управления называют команды, изменяющие естественную последовательность выполнения команд программы.*

*) Другие формы логических условий, не относящиеся к результатам арифметических операций, будут рассмотрены в гл. VII.

В машине имеется две операции условной передачи управления по сигналу ω, которые мы будет обозначать через $У_0$ и $У_1$. Здесь буква $У$ обозначает условную передачу управления, а индекс 0 или 1 — значение сигнала ω, при котором передача управления происходит. В адресной части команды нужно указать адрес ячейки, которой, в зависимости от выполнения условия, следует передать управление. Мы будем предполагать, что этот адрес записывается во втором (среднем) адресе команды передачи управления; первый и третий адреса будем пока считать нулевыми.

Команды условной передачи управления выполняются следующим образом. Если команда

$$У_0 \quad n$$

лежит в ячейке с адресом r и в результате предыдущей операции выработался управляющий сигнал $\omega = 0$, то управление будет передано в ячейку n, т. е. следующей будет выполняться команда, записанная в ячейке n. Если же в результате предыдущей операции выработался сигнал $\omega = 1$, то управление передается ячейке с адресом $r + 1$.

Наоборот, команда

$$У_1 \quad n,$$

лежащая в ячейке с адресом r, передает управление ячейке n, если в результате предыдущей операции выработался сигнал $\omega = 1$, и ячейке $r + 1$ при сигнале $\omega = 0$ *).

Кроме описанных команд условной передачи управления, в машине есть также команда *безусловной передачи управления*, которая всегда передает управление ячейке, номер которой записан во втором адресе команды. Команду безусловной передачи управления мы будем обозначать так:

$$Б \quad n.$$

Первый и третий адреса этой команды также будем пока считать нулевыми.

---

*) Существуют и другие разновидности команд условной передачи управления. Например, в команде могут быть указаны два адреса и управление может передаваться по одному или по другому, в зависимости от сигнала ω. В других машинах нет сигнала ω, а команда условной передачи управления пишется так: $a < b$; c. Числа из ячеек a и b сравниваются, и если указанное неравенство справедливо, то управление передается ячейке c, а если нет, то следующей ячейке. Ту же команду можно записывать и в виде $a \geqslant b$; c и говорить, что при выполнении неравенства управление передается следующей ячейке, а в противном случае — ячейке c.

§ 19. Разветвляющиеся программы

Рассмотрим несколько программ разветвляющихся вычислительных процессов.

Пример 1.19. Составить программу решения квадратного уравнения

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

Будем считать, что всегда $a\neq 0$. Тогда, если $D=b^2-4ac\geqslant 0$, то $x_1=-\frac{b}{2a}+\frac{\sqrt{D}}{2a}$, $x_2=-\frac{b}{2a}-\frac{\sqrt{D}}{2a}$, если же $D<0$, то $x_1=-\frac{b}{2a}+i\frac{\sqrt{|D|}}{2a}$, $x_2=-\frac{b}{2a}-i\frac{\sqrt{|D|}}{2a}$.

После работы программы в ячейки x_1, x_1', x_2, x_2' должны быть помещены действительные и мнимые части корней. Блок-схему (схему счета) программы можно представить в виде рис. 11.

Программы для отдельных блоков (участков) задачи можно написать так:

Первый блок

1) $a+a=2a$
2) $b:2a=R_1$
3) «0» $-R_1=\alpha$
4) $b\cdot b=R_1$
5) $2a\cdot c=R_2$
6) $R_2+R_2=R_2$
7) $R_1-R_2=D$

Третий блок

1) $\sqrt{D}=R_1$
2) $R_1:2a=\beta$
3) $\alpha+\beta=x_1$
4) $\alpha-\beta=x_2$
5) «0» + «0» $=x_1'$
6) «0» + «0» $=x_2'$

Четвертый блок

1) «0» $-D=R_1$
2) $\sqrt{R_1}=R_1$
3) $R_1:2a=\beta$
4) «0» + $\alpha=x_1$
5) «0» + $\alpha=x_2$
6) «0» + $\beta=x_1'$
7) «0» $-\beta=x_2'$.

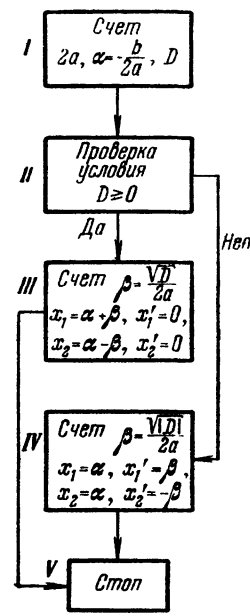


Рис. 11.

Для того чтобы эти три счетных блока, записанные последовательно, объединить в программу решения квадратного уравнения,

воспользуемся тем, что последняя команда первого блока вырабатывает сигнал $\omega = 1$, если $D < 0$, и сигнал $\omega = 0$, если $D \geqslant 0$.

Поэтому, расположив блоки в порядке, указанном на схеме рис. 11, следует после блока I поставить команду

$$У_1 \quad \text{блок IV},$$

которая заставит обойти счет действительных корней (блок III) и сосчитать мнимые корни, если $\omega = 1$, т. е. $D < 0$.

При выполнении условия $D \geqslant 0$ после блока I сигнал ω окажется равным нулю, и команда «$У_1$ блок IV» передаст управление на первую команду следующего за ней блока III вычисления действительных корней. Для того чтобы после счета действительных корней не считались мнимые, после блока III следует поставить команду безусловной передачи управления на конец счета:

$$Б \quad \textit{стоп}.$$

Таким образом, программа вычисления корней квадратного уравнения имеет следующий вид:

1)	$a + a = 2a$	Блок I
2)	$b : 2a = R_1$	
3)	«0» $- R_1 = \alpha$	
4)	$b \cdot b = R_1$	
5)	$2a \cdot c = R_2$	
6)	$R_2 + R_2 = R_2$	
7)	$R_1 - R_2 = D$	
8)	$У_1$ 16)	Блок II
9)	$\sqrt{D} = R_1$	Блок III
10)	$R_1 : 2a = \beta$	
11)	$\alpha + \beta = x_1$	
12)	$\alpha - \beta = x_2$	
13)	«0» + «0» $= x_1'$	
14)	«0» + «0» $= x_2'$	
15)	Б 23)	
16)	«0» $- D = R_1$	Блок IV
17)	$\sqrt{R_1} = R_1$	
18)	$R_1 : 2a = \beta$	
19)	«0» $+ \alpha = x_1$	
20)	«0» $+ \alpha = x_2$	
21)	«0» $+ \beta = x_1'$	
22)	«0» $- \beta = x_2'$	
23)	*стоп*.	Блок V

Проследим, как работает составленная программа. Сначала блок I (команды 1) — 7)) вычисляет величины $2a$, $\alpha = -\frac{b}{2a}$, $D = b^2 - 4ac$. Если дискриминант D оказывается отрицательным, то команда 7) вырабатывает сигнал $\omega = 1$, и по команде 8) управление передается на начальную команду 16) блока IV вычисления мнимых корней; вычислив мнимые корни (команды 16) — 22)), машина останавливается (команда 23)). Если же $D \geqslant 0$, то команда 7) вырабатывает сигнал $\omega = 0$, и команда 8) передает управление следующей за ней начальной команде 9) блока III счета действительных корней. Сосчитав эти корни (команды 10) — 14)), машина выполняет команду безусловной передачи управления 15), которая заставляет машину, обойдя блок IV счета мнимых корней, выйти на *стоп* (команда 23)).

До сих пор мы предполагали 1-й и 3-й адреса команд передачи управления нулевыми. Эти команды работали как одноадресные, в них использовался лишь 2-й адрес.

В машине предусмотрена возможность работы команд передачи управления как трехадресных. В этом более общем случае они записываются так:

$$У_0 \quad \overrightarrow{a \quad b \quad c},$$

$$У_1 \quad \overrightarrow{a \quad b \quad c},$$

$$Б \quad \overrightarrow{a \quad b \quad c}.$$

Как и ранее, по этим командам осуществляется условная или безусловная передача управления команде с номером b или команде с номером $r + 1$; символы $У_0$, $У_1$, $Б$ здесь имеют прежнее значение. Однако, кроме передачи управления, по этим командам происходит пересылка содержимого ячейки с номером a (первый адрес) в ячейку с номером c (третий адрес). Такое совмещение двух операций в одной команде не замедляет ее выполнения, ибо пересылка производится одновременно с передачей управления. При этом надо иметь в виду, что *пересылка выполняется всегда*, независимо от значения сигнала ω.

Покажем на примере программы решения квадратного уравнения, как использовать команды передачи управления в указанном более общем виде.

В этой программе имеются пять команд, являющихся, по существу, командами пересылки: 13), 14), 19), 20), 21). Пересылку 14) можно совместить с командой безусловной передачи управления 15), которая после этого будет выглядеть так:

$$Б \quad \overrightarrow{«0» \quad стоп \quad x_2'}.$$

Команду пересылки 19) можно совместить с командой условной передачи управления 8), что дает

$$У_1 \quad \overrightarrow{\alpha \quad \text{блок IV} \quad x_1}$$

Заметим, кроме того, что в блоке IV можно еще избавиться от команды 21), помещая результат выполнения команды 18) в ячейку x_1'. После этих преобразований программа решения квадратного уравнения примет такой вид:

1)		$a + a$	$= \rho$
2)		$b : \rho$	$= R_1$
3)		«0» $- R_1$	$= \alpha$
4)		$b \cdot b$	$= R_1$
5)		$\rho \cdot c$	$= R_2$
6)		$R_2 + R_2$	$= R_2$
7)		$R_1 - R_2$	$= D$
8)	$У_1$	α 15)	x_1
9)		$\sqrt{D}$	$= R_1$
10)		$R_1 : \rho$	$= \beta$
11)		$\alpha + \beta$	$= x_1$
12)		$\alpha - \beta$	$= x_2$
13)		«0» + «0»	$= x_1'$
14)	$Б$	«0» 20)	x_2'
15)		«0» $- D$	$= R_1$
16)		$\sqrt{R_1}$	$= R_1$
17)		$R_1 : \beta$	$= x_1'$
18)		«0» $- x_1'$	$= x_2'$
19)		«0» $+ x_1$	$= x_2$
20)		*стоп.*	

Таким образом, программа сократилась на три команды, причем выполняться она будет быстрее, чем прежняя.

Отметим еще следующую особенность этой программы: по команде 8) одновременно с передачей управления в ячейку x_1 засылается величина α, вне зависимости от того, вычисляются ли дальше мнимые или действительные корни. Однако $x_1 = \alpha$ лишь в случае мнимых корней. Не получаются ли по нашей программе неправильные значения действительных корней? На самом деле это не так, ибо при работе блока вычисления действительных корней в ячейку x_1 посылается взамен α правильное значение $\alpha + \beta$ (команда 11)).

В написанной программе используются шесть рабочих ячеек: ρ, α, D, β, R_1, R_2. Можно вообще избавиться от рабочих ячеек, помещая промежуточные результаты в результирующие ячейки x_1, x_2, x_1', x_2'.

При записи программ в буквенном виде широко применяют стрелки вместо вторых адресов команд передачи управления. Эти стрелки направляются к той команде, к которой условно или безусловно передается управление. Нумерация команд при этом оказывается излишней.

К разветвлению программ мы приходим в задачах, в которых какая-либо величина находится по-разному, в зависимости от выпол-

нения каких-либо условий. Простейшим примером такого рода является нахождение модуля числа a:

$$|a| = \begin{cases} a, & \text{если } a \geqslant 0, \\ -a, & \text{если } a < 0. \end{cases}$$

Пример 2.19. В трехадресных машинах, в которых отсутствует элементарная операция нахождения разности модулей двух чисел,

$$|a| - |b| = c,$$

модуль числа можно определить при помощи следующей простой программы:

1) $a \quad + «0» = |a|$
2) $У_0$ ——————
3) $«0» \quad - \quad a \quad = |a|$
4) *стоп*

Команда 1) этой программы помещает в ячейку $|a|$ величину a и вырабатывает сигнал $\omega = 0$, если $a \geqslant 0$, и сигнал $\omega = 1$, если $a < 0$. В первом из этих случаев команда 2) передает управление на *стоп*, и в ячейке $|a|$ остается величина a. Во втором случае, т. е. при $a < 0$, команда 2) передает управление команде 3), и в ячейке $|a|$ оказывается $-a$, а ранее засланное стирается.

Рассмотрим пример из механики.

Пример 3.19. Шар массы M с радиусом ρ и постоянной плотностью δ притягивает материальную точку массы m, находящуюся на расстоянии r от центра шара. Сила F притяжения точки шаром вычисляется так: если точка находится вне шара, или на его поверхности, т. е. если $r \geqslant \rho$, то

$$F = f \cdot \frac{Mm}{r^2},$$

где f — коэффициент, называемый постоянной тяготения; если же точка находится внутри шара, т. е. если $r < \rho$, то

$$F = f \cdot \frac{mM(r)}{r^2},$$

где $M(r)$ — масса части данного шара с радиусом r. Очевидно,

$$M = \frac{4}{3}\pi\rho^3\delta, \quad M(r) = \frac{4}{3}\pi r^3\delta.$$

Поэтому формулу для вычисления силы F можно представить в виде

$$F = \begin{cases} \gamma\frac{\rho^3}{r^2}, & \text{если } r \geqslant \rho, \\ \gamma r, & \text{если } r < \rho, \end{cases}$$

где $\gamma = \frac{4}{3}\pi f \delta$ — постоянный коэффициент (эта величина имеет определенный физический смысл: она равна силе, с которой шар радиусом 1 и плотностью δ притягивает точку, находящуюся на его поверхности).

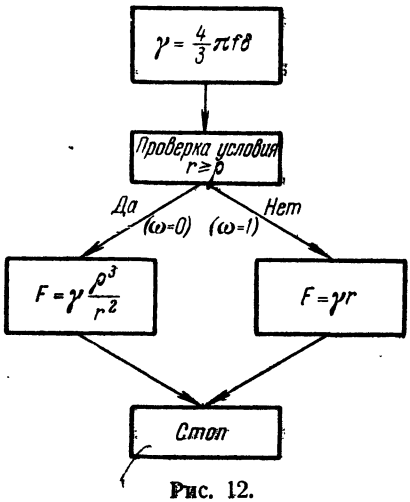


Рис. 12.

Условие $r \geqslant \rho$ можно записать и так: $r - \rho \geqslant 0$. Чтобы его проверить, следует выполнить команду $r - \rho = k$. Если величина $k \geqslant 0$, т. е. $\omega = 0$, следует считать F по первой формуле, если $k < 0$ ($\omega = 1$), то по второй.

Разность $k = r - \rho$ для расчета F не нужна, поэтому в качестве ячейки k можно использовать любую свободную ячейку. Обычно в качестве такой ячейки берут ячейку с нулевым номером. Команду, вырабатывающую сигнал ω, пишут тогда так:

$$r - \rho = 0.$$

Составим блок-схему расчета F. Она показана на рис. 12.

Программа из 13 команд, составленная в соответствии с этой блок-схемой, имеет следующий вид:

1)		«$\frac{4}{3}$»	• «π»	= γ
2)		γ	• f	= γ
3)		γ	• δ	= γ
4)		r	— ρ	= 0
5)	$У_0$			
6)		γ	• r	= F
7)	$Б$			
8)		γ	: r	= F
9)		F	: r	= F
10)		F	• ρ	= F
11)		F	• ρ	= F
12)		F	• ρ	= F
13)		*стоп.*		

Запишем формулы для определения F несколько иначе:

$$F=\begin{cases}\gamma r\left(\frac{\rho}{r}\right)^3, & \text{если } \frac{\rho}{r}<1,\\ \gamma r, & \text{если } \frac{\rho}{r}\geqslant 1.\end{cases}$$

При такой записи естественно вести вычисления в следующем порядке: сначала вычислить значение F по второй из формул, затем проверить условие $\frac{\rho}{r}\geqslant 1$, и если оно не выполняется, домножить прежнее значение F на $\left(\frac{\rho}{r}\right)^3$.

Программа вычисления F указанным способом состоит из 10 команд и имеет такой вид:

1) $\ll\frac{4}{3}\gg\cdot\ll\pi\gg=\gamma$
2) $\gamma\cdot f=\gamma$
3) $\gamma\cdot\delta=\gamma$
4) $\gamma\cdot r=F$
5) $\rho:r=k$
6) Y_1
7) $F\cdot k=F$
8) $F\cdot k=F$
9) $F\cdot k=F$
10) *стоп.*

Команда деления 5) в этой программе используется и для получения частного $\rho:r=k$, и для выработки сигнала ω. Если величина $k\geqslant 1$, то команда 5) вырабатывает сигнал $\omega=1$ и команда 6) выводит нас на *стоп*, сохраняя вычисленное при помощи команд 1) — 4) значение $F=\gamma r$. Если же $k<1$, то после команды 5) сигнал $\omega=0$, и команда 6) передает управление на команду 7). После этого, в полном соответствии с последней формулой, старое значение $F=\gamma r$ три раза умножается на $k=\frac{\rho}{r}$ и машина выходит на *стоп*.

Сравнивая эту программу с предыдущей, мы убеждаемся в том, что более рациональная запись формулы и использование команды деления для выработки сигнала ω дали возможность сэкономить три команды.

При помощи команд условной передачи управления можно решать и некоторые логические задачи. Рассмотрим одну из простейших задач такого рода.

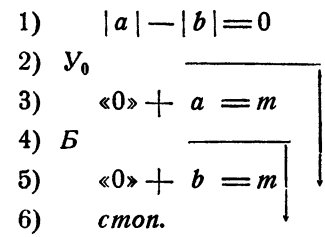
Пример 4.19. Из двух чисел a и b найти меньшее по модулю и поместить в ячейку m.

Чтобы сравнить величины модулей чисел a и b, выполним команду

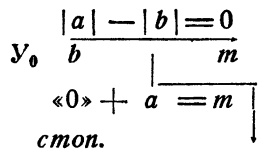
$$|a| - |b| = 0.$$

Если $|a| \geqslant |b|$, эта команда выработает сигнал $\omega = 0$. Если $|a| < |b|$, сигнал ω будет равен 1. Поэтому при $\omega = 0$ в m следует послать b, а при $\omega = 1$ посылать a.

Программа нахождения m имеет вид



Можно сократить эту программу на две команды, совместив пересылку «0» $+ a = m$ с условной передачей управления и опустив безусловную передачу управления:



Проследим, как работает эта программа. Если $|a| \geqslant |b|$, команда 1) вырабатывает сигнал $\omega = 0$ и команда 2) передает управление на *стоп*, одновременно засылая в m меньшее по модулю число b. Если же $|a| < |b|$, то команда 1) вырабатывает сигнал $\omega = 1$. После этого команда 2) передает управление команде 3), а в ячейке m, вопреки требуемому, оказывается большее по модулю число b; однако следующая команда 3) посылает в ячейку m меньшее по модулю число a.

В некоторых задачах для получения результата следует произвести проверку многих логических условий. Такая проверка при программировании осуществляется при помощи нескольких условных передач управления.

Пример 5.19. Найдем расстояние от точки M до полукруглого контура с радиусом a *).

Введем в плоскости прямоугольную декартову систему координат и расположим контур так, как показано на рис. 13.

Расстояние ρ точки M от контура будет вычисляться по-разному, в зависимости от ее расположения по отношению к контуру. На

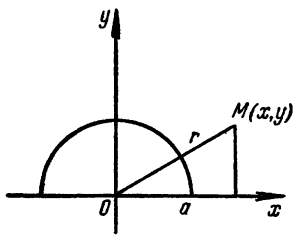


Рис. 13.

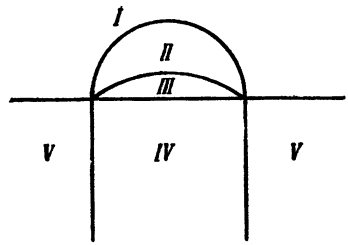


Рис. 14.

рис. 14 вся плоскость разбита на пять областей I, II, III, IV, V, в которых может находиться точка M.

Если точка M находится в области I, в которой $y>0$, $r\geqslant a$, то $\rho=r-a$,

в области II $(y\geqslant a-r>0)$	$\rho=a-r$,
в области III $(y>0,\ y\leqslant a-r)$	$\rho=y$,
в области IV $(y\leqslant 0,\ \lvert x\rvert<a)$	$\rho=-y$,

и, наконец, в области $V(y\leqslant 0,\ \lvert x\rvert\geqslant a)$

$$\rho=\sqrt{(\lvert x\rvert-a)^2+y^2}.$$

Таким образом,

$$\rho=\begin{cases} r-a, & \text{если } y>0,\ r-a\geqslant 0,\\ a-r, & \text{если } y>0,\ r-a<0,\ y+r-a\geqslant 0,\\ y, & \text{если } y>0,\ r-a<0,\ y+r-a<0,\\ -y, & \text{если } y\leqslant 0,\ \lvert x\rvert-a<0,\\ \sqrt{(\lvert x\rvert-a)^2+y^2}, & \text{если } y\leqslant 0,\ \lvert x\rvert-a\geqslant 0. \end{cases}$$

*) Расстоянием точки от кривой называется наименьшее из расстояний этой точки до всевозможных точек кривой.

Программа определения ρ имеет следующий вид:

1)		$x \cdot x = R_1$
2)		$y \cdot y = R_2$
3)		$R_1 + R_2 = R_1$
4)		$\sqrt{R_1} \quad = r$
5)		«0» $- y = \rho$
6)	$У_1$	13)
7)		$\lvert x \rvert - \lvert a \rvert = R_1$
8)	$У_1$	18)
9)		$R_1 \cdot R_1 = R_1$
10)		$R_2 + R_1 = R_1$
11)		$\sqrt{R_1} \quad = \rho$
12)	*Б*	18)
13)		$r - a = \rho$
14)	$У_0$	18)
15)		$y + \rho = 0$
16)	$У_1$	y 18) ρ
17)		$a - r = \rho$
18)		*стоп.*

Команды 1) — 4) вычисляют r. При помощи команд 5), 6) отделяются области *I*, *II*, *III* верхней полуплоскости от областей *IV*, *V* нижней полуплоскости, причем команда 5) дает значение ρ для области *IV*.

По командам 7), 8) мы выходим на *стоп*, если точка лежит в области *IV*, или переходим к выполнению операций 9) — 11), дающих ρ для области *V*. Команда 13) дает ρ для области *I*, команда 16) — для области *III* и команда 17) — для области *II*. Соответствующие разветвления (условные передачи управления) рекомендуется проследить самому читателю.

§ 20. Арифметические циклы

Как видно из примеров предыдущего параграфа, характерной особенностью разветвляющихся программ является то, что отдельные команды или группы команд выполняются или не выполняются (обходятся) в зависимости от тех или иных условий. Однако во всех случаях каждая выполняемая команда выполняется только один раз.

Ясно, что задачи, программы для которых имеют такой характер, крайне невыгодны для решения на электронных счетных машинах, поскольку время написания одной команды неизмеримо превышает

время ее выполнения. Наиболее выгодными для машины являются такие задачи, в которых одна и та же команда или группа команд выполняется многократно. Составление таких программ тем более возможно, что в команде указываются не числа, над которыми производятся операции, а адреса ячеек; содержимое же этих ячеек при повторном выполнении тех же команд может быть иным.

Группу команд, которая выполняется в процессе решения задачи несколько раз, принято называть *циклом* или *циклической программой*. Циклы являются основными частями всяких практически используемых программ, а потому организация цикла есть наиболее часто встречающаяся задача программирования. Как мы увидим, циклы состоят всегда из одних и тех же частей, которые располагаются определенным образом.

Рассмотрим сначала несколько примеров программирования циклических вычислительных процессов.

Пример 1.20. Составить программу для вычисления n-й степени числа x (n — целое положительное число):

$$z = x^n.$$

Расписывая формулу для z по командам, получим программу:

«0» $+ \; x = z$
$z \cdot x = z$
$z \cdot x = z$
. . . . .
$z \cdot x = z$
} $n - 1$ раз

стоп.

Таким образом, при программировании этой задачи потребовалась однообразная работа по выписыванию $n + 1$ команд. Между тем очевидно, что процесс вычисления z является циклическим: он состоит из многократного повторения команды умножения $z \cdot x = z$. Приведенная программа является бесцикловой в том смысле, что каждая из написанных команд выполняется только один раз.

Следует обратить внимание на то, что написанные в программе команды одинаковы. Все они имеют вид $z \cdot x = z$, т. е. содержат тот же код операции умножения и те же адреса ячеек z и x. Однако выполняются они всякий раз иначе, так как содержимое ячейки z изменяется. Благодаря этому в ячейке z и появляются следующие степени числа x. Естественно попытаться написать программу так, чтобы эта команда была написана один раз, а выполнялась столько раз, сколько нужно.

Постараемся составить цикловую программу, т. е. такую, в которой команда цикла $z \cdot x = z$ будет написана только один раз,

а выполняться будет многократно. Перепишем сначала предыдущую программу в несколько ином виде, заменив первую команду двумя:

$$\left.\begin{array}{l} «0» + «1» = z \\ z \cdot x = z \\ z \cdot x = z \\ \cdots\cdots \\ z \cdot x = z \end{array}\right\} n \text{ раз}$$

стоп.

Команда «0» + «1» = z служит для подготовки цикла, она заносит в ячейку результата z его начальное значение 1.

Чтобы повторить выполнение одной и той же команды несколько раз, нужно после нее поставить команду передачи управления, которая возвращала бы машину снова к выполнению той же команды. Однако безусловная передача управления здесь не годится, так как тогда управление передавалось бы всегда и нельзя было бы обеспечить выполнение команды нужное число раз.

Введем в рассмотрение специальную ячейку a (*«счетчик»*), в которой будем считать, сколько раз выполняется нужная нам команда. При каждом выполнении команды будем прибавлять к содержимому ячейки a единицу. С помощью этой ячейки можно обеспечить повторение команды нужное число раз. Для этого достаточно каждый раз проверять содержимое ячейки a и передавать управление на повторение команды, пока a меньше n, либо на окончание, когда счетчик покажет выполнение команды нужное число раз.

Таким образом, мы приходим к следующей программе:

1) «0» + «1» = z
2) «0» + «0» = a
3) $z \cdot x = z$
4) a + «1» = a
5) $a - n = 0$
6) $У_1$ 3)
7) *стоп.*

Проследим, как работает эта программа. Сначала выполняются команды 1), 2), составляющие подготовительную часть цикловой программы. Затем первый раз выполняется команда 3), образующая рабочую часть цикла; в ячейке z оказывается x.

После выполнения команды 4) в счетчике a появляется 1; разность $a - n$ будет отрицательной, и команда 5) вырабатывает сигнал

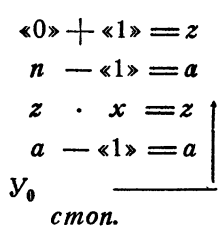
$\omega = 1$; команда 6) передает управление на команду 3), после выполнения которой в z оказывается x^2.

Такое повторение цикла произойдет n раз. Действительно, после того как цикл повторится $n-1$ раз, в z окажется величина x^{n-1}, а в счетчике a — число $n-1$. Тогда разность величин $a-n$ будет отрицательной, сигнал $\omega = 1$, и команда 6) заставит машину выполнить команды 3), 4), 5) еще раз, в результате чего в ячейке z появится x^n, в a окажется n, разность $a-n$ станет равной нулю, сигнал ω первый раз станет нулем, и команда 6) передаст управление на *стоп*.

Сравним теперь цикловую и бесцикловую программы. Бесцикловая программа состоит из $n+1$ команд, цикловая при любом n — из семи команд. Следовательно, при $n > 5$ цикловая программа короче бесцикловой. Кроме того, бесцикловую программу для произвольного n вообще невозможно написать. Однако следует учитывать, что цикловая программа работает медленнее бесцикловой.

Действительно, при работе бесцикловой программы фактически выполняется столько операций, сколько команд в программе, т. е. $n+1$. При работе же цикловой программы один раз выполняются лишь подготовительные команды 1), 2) и заключительная команда 7), основные же команды программы 3) — 6) выполняются n раз. Таким образом, при работе цикловой программы выполняется $4n+3$ элементарные операции, т. е. в четыре раза больше, чем в цикловой программе. Это обстоятельство в тех случаях, когда важнее выгадать в быстродействии, чем в памяти, заставляет для циклических процессов составлять вместо коротких цикловых длинные бесцикловые программы.

В рассматриваемой задаче можно сократить число команд в цикле, если изменять счетчик в обратном направлении не от 0 до n, а от $n-1$ до -1:



Этот вариант программы короче предыдущего на одну команду за счет того, что образование счетчика и выработка сигнала ω производятся одной командой $a - «1» = a$ (в предыдущем варианте программы на это требовалось две команды — 4) и 5)). Вместе с тем эта программа будет работать быстрее предыдущей, ибо теперь на получение z требуется $3n+3$ элементарные операции.

Пример 2.20. Запрограммируем вычисление произведения последовательных целых чисел от 1 до 15:

$$x = 15! = 1 \cdot 2 \cdot \ldots \cdot 15.$$

Процесс вычисления в этой задаче целесообразно организовать по плану:

а) в некоторую ячейку n заносим 0, а в ячейку x — единицу,
б) к a прибавляем 1, результат заносим в a,
в) x умножаем на a, результат заносится в x,
г) пункты б), в) повторяем 15 раз.

Бесцикловая программа имеет такой вид:

$$\left.\begin{array}{l} \text{«0»} + \text{«0»} = a \\ \text{«0»} + \text{«1»} = x \\ a + \text{«1»} = a \\ x \cdot a = x \\ \ldots\ldots\ldots \\ \ldots\ldots\ldots \\ a + \text{«1»} = a \\ x \cdot a = x \end{array}\right\} \text{15 пар команд}$$

стоп

и состоит из 33 команд. Процесс счета в рассматриваемой программе состоит из выполнения двух подготовительных команд:

1) «0» + «0» = a
2) «0» + «1» = x

и 15 пар одинаковых команд:

3) a + «1» = a
4) $x \cdot a = x$.

Чтобы получить цикловую программу, следует, написав команды 3), 4) только один раз, заставить их выполняться 15 раз. Такое повторение можно осуществить при помощи двух команд:

5) a — «15» = 0
6) $У_1$ 3).

Действительно, команда 5) будет вырабатывать сигнал $\omega = 1$ до тех пор, пока a не станет равным 15, поэтому команда 6) будет передавать управление на рабочую часть цикла 3), 4) 15 раз. Когда a окажется равным 15, по команде 4) в x образуется 15!. Команда 5) выработает сигнал $\omega = 0$, а команда 6) передаст управление на команду *стоп*.

Таким образом, цикловая программа для нахождения x имеет вид

$$
\begin{array}{|l}
\text{«0»} + \text{«0»} = a \\
\text{«0»} + \text{«1»} = x \\
a + \text{«1»} = a \\
x \cdot a = x \\
a - \text{«15»} = 0 \\
\hline
\end{array}
\quad
\begin{array}{l}
\uparrow \\ \\ \\ \\ \\ \\
\end{array}
$$

$У_1$

стоп.

Аналогичный вид имеет программа для вычисления величины $x = n!$ при произвольном натуральном n:

$$
\begin{array}{|l}
\text{«0»} + \text{«0»} = a \\
\text{«0»} + \text{«1»} = x \\
a + \text{«1»} = a \\
x \cdot a = x \\
a - n = 0 \\
\hline
\end{array}
\quad
\begin{array}{l}
\uparrow \\ \\ \\ \\ \\ \\
\end{array}
$$

$У_1$

стоп.

Ячейка a служит в этой и предыдущей программах *счетчиком цикла*; когда рабочая часть цикла пройдена один раз, $a = 1$, два раза — $a = 2$ и т. д. Отметим, что в примере 2.20 ячейка a участвует в вычислениях, тогда как в предыдущем примере 1.20 счетчик в вычислениях не участвовал, а был образован специально для проверки числа повторений цикла. По этой причине счетчик в последнем примере может быть назван *естественным*, тогда как в примере 1.20 его можно назвать *искусственным счетчиком*.

В обоих примерах ячейка n содержала число повторений цикла. Это число называют *эталоном цикла*. Проверка окончания цикла в наших примерах происходила путем сравнения счетчика с эталоном. Для обоих рассмотренных примеров характерно многократное повторение группы одинаковых команд, причем число повторений n известно заранее. Программа такого вычислительного процесса называется *арифметическим циклом*.

Рассмотрим еще одну физическую задачу, решение которой также приводит к программированию арифметического цикла.

Пример 3.20. Вдоль луча на равных расстояниях друг от друга находится 101 электрически заряженная частица. Заряды всех частиц одинаковы и положительны. Найти силу, с которой частица, находящаяся в начале луча, отталкивается остальными.

Из физики известно, что сила отталкивания двух положительных зарядов q_1 и q_2, находящихся на расстоянии $r_{1,2}$ друг от друга, определяется формулой

$$F_{1,2}=\gamma\frac{q_1q_2}{r_{1,2}^2}.$$

Поэтому силы отталкивания точки q_0 остальными точками $q_1, q_2, \ldots, q_{100}$ равны:

$$F_{0,1}=\gamma\frac{q_0q_1}{r_{0,1}^2},\quad F_{0,2}=\gamma\frac{q_0q_2}{r_{0,2}^2},\quad \ldots,\quad F_{0,100}=\gamma\frac{q_0q_{100}}{r_{0,100}^2}.$$

Результирующая сила:

$$F_0=F_{0,1}+F_{0,2}+\ldots+F_{0,100}=\gamma q_0\left(\frac{q_1}{r_{0,1}^2}+\frac{q_2}{r_{0,2}^2}+\ldots+\frac{q_{100}}{r_{0,100}^2}\right).$$

По условию задачи одинаковы все заряды и расстояния между парами соседних зарядов. Поэтому $q_0=q_1=\ldots=q_{100}=q$, $r_{0,1}=\Delta$, $r_{0,2}=2\Delta$, $\ldots$, $r_{0,100}=100\Delta$, и, следовательно,

$$F=\gamma\frac{q^2}{\Delta^2}\left(\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{100^2}\right)$$

или

$$F=\gamma\left(\frac{q}{\Delta}\right)^2\Sigma,$$

где

$$\Sigma=\sum_{p=1}^{100}\frac{1}{p^2}.$$

Составим программу для вычисления F. Предположим, что мы уже вычислили сумму k слагаемых и в ячейке Σ лежит сумма

$$\frac{1}{1^2}+\frac{1}{2^2}+\ldots+\frac{1}{k^2}.$$

Для получения следующего значения суммы нужно выполнить команду

$$\Sigma+u=\Sigma,$$

причем в ячейку u следует предварительно положить значение $\frac{1}{(k+1)^2}$. По этой формуле нужно произвести вычисления 100 раз, полагая, что в начале цикла мы приняли $\Sigma=0$. Каждый такой шаг можно сделать с помощью четырех команд:

1) $k+$«1»$=k$
2) «1» : $k=u$
3) $u\cdot u=u$
4) $\Sigma+u=\Sigma$.

Приведенная группа команд состоит из двух частей. Команда 1) служит для изменения содержимого ячейки k, которая является здесь и счетчиком, и основной рабочей ячейкой программы. Команды 2)—4) служат *рабочей частью* цикла. Здесь вычисляется очередное слагаемое и оно прибавляется в ячейку Σ. Эту группу команд нужно повторить сто раз, что можно сделать при помощи следующих команд:

$$k - \text{«100»} = 0,$$
$$y_1 \qquad 1).$$

Для подготовки цикла следует заслать нули в ячейки суммы Σ и счетчика k:

$$\text{«0»} + \text{«0»} = \Sigma,$$
$$\text{«0»} + \text{«0»} = k.$$

Собирая все эти команды вместе и добавляя команды для получения F из Σ, получим следующую программу:

1)		«0»	+	«0»	$= \Sigma$
2)		«0»	+	«0»	$= k$
3)		k	+	«1»	$= k$
4)		«1»	:	k	$= u$
5)		u	·	u	$= u$
6)		Σ	+	u	$= \Sigma$
7)		k	−	«100»	$= 0$
8)	y_1				
9)		q	:	Δ	$= u$
10)		Σ	·	u	$= F$
11)		F	·	u	$= F$
12)		F	·	γ	$= F$
13)		*стоп.*			

Как видно из рассмотренных примеров, программы арифметических циклов всегда состоят из одних и тех же частей, которые соединяются между собой в определенной последовательности. Рассмотрим эти части на примере только что составленной программы.

Команды 1) и 2) засылают в рабочие ячейки программы k и Σ нули, т. е. их первоначальное содержимое. Эту группу команд называют командами *подготовки цикла.*

Благодаря наличию команд подготовки цикла выполнение программы можно повторять любое число раз, независимо от предыдущего состояния рабочих ячеек, без нового ввода программы в память. Программа

работала бы так же, если опустить эти две команды, но одновременно с вводом программы ввести в ячейки k и Σ нули. Однако в этом случае мы не смогли бы повторить программу еще раз, выполняя ее сначала, так как в процессе ее работы содержимое ячеек k и Σ изменилось бы и, чтобы пустить программу еще раз, пришлось бы снова вводить ее в память.

Программу с командами 1) и 2) можно пускать любое число раз, выполнив ее до этого полностью или частично и прервав ее выполнение в любом месте. Команды 1) и 2), помещенные в начале программы, восстановят первоначальное содержание рабочих ячеек k и Σ. Такие программы, которые можно пускать любое число раз без нового ввода, называют *самовосстанавливающимися программами* (иногда также *самовозобновляющимися*), а группу команд подготовки цикла, которые обеспечивают программе свойство самовосстановления, называют также *командами восстановления*. Из сказанного ясно, что команды восстановления или подготовки цикла *должны обязательно быть первой частью цикловой программы*.

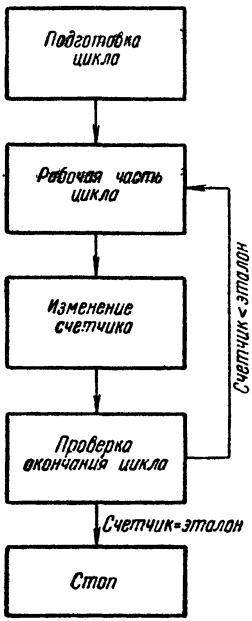


Рис. 15.

Команда 3) предназначена для изменения содержимого ячейки k, играющей роль счетчика и вместе с тем участвующей в вычислениях. С помощью этой команды мы переходим к вычислению следующего слагаемого, т. е. к выполнению следующего шага цикла. Ее называют поэтому командой *изменения*.

Дальше идет группа команд 4) — 6), которые составляют *рабочую часть цикла*, т. е. основные команды, вычисляющие очередное слагаемое и прибавляющее его к сумме. Наконец, команды 7) и 8) образуют группу команд *проверки окончания цикла*. Эти команды сверяют содержимое счетчика с эталоном и, в зависимости от результатов, передают управление либо на изменение и рабочую часть для выполнения следующего шага цикла, либо на выход из цикла и дальнейшие вычисления, которые можно производить после того, как цикл закончен. В нашей программе эти дальнейшие вычисления, не входящие в цикл, осуществляются командами 9) — 13).

Перечисленные части цикловой программы расположены в программе так, как это показано на рис. 15. Однако эта схема не является наиболее удобной.

Обратим внимание на следующее обстоятельство. Эталон «100», используемый при проверке окончания цикла, совпадает с числом пов-

торений цикла, что очень удобно. Но при подготовке цикла мы засылаем в счетчик не то значение, с которого фактически начинаются вычисления ($k = 1$), а предыдущее ($k = 0$). Это уже составляет неудобство, так как такое предшествующее значение счетчика надо еще вычислять, а это не всегда так легко, как в данном примере. Между тем при данной блок-схеме цикла иначе поступить нельзя, потому что изменение k происходит до рабочей части цикла.

Можно было бы в подготовке цикла полагать $k = 1$ и переходить сразу к рабочей части, переставив команду изменения ниже, между рабочей частью и проверкой окончания, т. е. поместив нынешнюю команду 3) между командами 6) и 7). Такая перестановка вполне естественна, но, как легко заметить, она потребовала бы изменения эталона окончания цикла.

Действительно, если *изменение* стоит после *рабочей части*, то проверка окончания происходит до того, как вычислен член с соответствующим значением k. Тогда после вычисления 99-го члена в ячейке k образуется уже число 100, и если оставить эталон окончания без изменения, то разность $k - 100$ будет уже равна нулю и команда 8) передаст управление на выход из цикла, так что последний член останется невычисленным. Для верной работы программы нужно в качестве эталона брать уже не 100, а 101, т. е. не число повторений цикла, а следующее за ним, которое тоже надо еще вычислять.

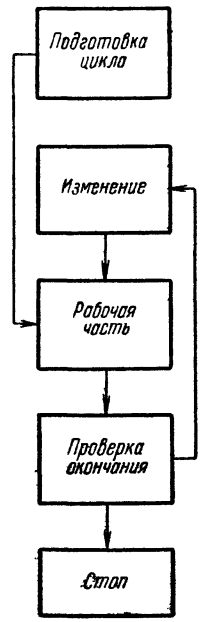


Рис. 16.

Оказывается, все эти затруднения легко преодолеть, если оставить команду изменения в начале программы, но обходить ее в начале цикла таким образом, чтобы после подготовки цикла переходить сразу к выполнению рабочей части. В этом случае можно засылать в счетчик в подготовке цикла фактически первое его значение и в качестве эталона для проверки окончания использовать число повторений цикла. Для того чтобы этого достигнуть, достаточно заменить команду 2) нашей программы такой:

$$2)\ Б\ \overline{\text{«1»}}\ \overrightarrow{4)}\ k.$$

Блок-схема цикла при таком расположении его частей имеет вид, показанный на рис. 16. Это и есть наиболее простая и естественная схема, которая может быть рекомендована к использованию практически во всех случаях.

Впрочем, иногда можно, ценой некоторого отступления от этой схемы, сократить цикл на одну команду. Для этого нужно изменять

счетчик в обратном направлении, как это делалось в последней программе примера 1.20. Если n — число повторений цикла, то команда подготовки цикла, приводящая в начальное состояние счетчик, должна тогда выглядеть так:

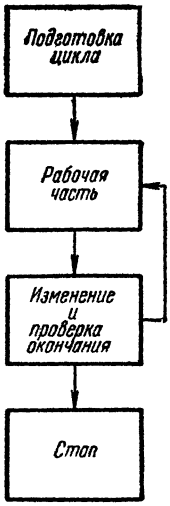


Рис. 17.

$$n - «1» = счетчик,$$

а команды изменения и проверки окончания цикла могут быть совмещены и представлены в виде

$$счетчик - «1» = счетчик,$$
$$У_0 \qquad\qquad РЧ,$$

где *РЧ* означает первую команду рабочей части. Такое сокращение возможно всегда с искусственным счетчиком, а с естественным лишь в том случае, когда вычисление соответствующих членов определяется только состоянием счетчика и не требует знания предыдущих. Блок-схема такого видоизмененного цикла показана на рис. 17.

Не следует думать, что арифметические циклы должны быть организованы исключительно по счетчику. Следующий пример представляет собой арифметический цикл, организованный по иному способу.

Пример 4.20. Вычислим сумму

$$S = f(a) + f(a+h) + f(a+2h) + \ldots,$$

где $f(x) = \frac{x^2-1}{x^2+1}$, а сумма распространена на значения x, удовлетворяющие условию $a \leqslant x \leqslant b$.

Здесь мы имеем обычный арифметический цикл, хотя число повторений и неизвестно. Его легко вычислить, но в этом нет никакой нужды, так как проверку окончания цикла можно осуществить, сравнивая значения x и b. Вычисления можно производить по следующей программе:

1) «0» + «0» = S
2) Б a 4) x
3) $x + h = x$
4) $x \cdot x = u$
5) $u - «1» = v$
6) $u + «1» = u$
7) $v : u = u$
8) $S + u = S$
9) $x - b = 0$
10) $У_1$ 3)
11) *стоп.*

§ 21. Итерационные циклы

Во многих циклических вычислительных задачах число шагов, нужных для решения задачи, заранее определить либо невозможно, либо очень трудно.

Рассмотрим, например, как вычисляется число π. Из геометрии известно, что число π есть предел полупериметра вписанного в единичную окружность правильного многоугольника при неограниченном увеличениии числа его сторон.

Для нахождения π с заданным числом верных десятичных знаков следует, последовательно увеличивая число сторон правильного многоугольника, вычислять его полупериметр. Здесь налицо циклический процесс. Когда же кончать цикл?

Заранее определить, сколько сторон нужно взять у многоугольника, чтобы получить π с нужной точностью, очень трудно. Однако можно поступить следующим образом: сравнивать два последовательных значения полупериметра и оканчивать цикл тогда, когда эти значения с заданной точностью совпадут.

Такие циклы, в которых заранее неизвестно число повторений и проверка окончания происходит не по счетчику, а по достижении нужной точности, называют *итерационными*. Читателю должно быть ясно, что никаких изменений в структуре цикла при переходе от арифметических циклов к итерационным не происходит. Итерационный цикл состоит из тех же частей и строится по той же блок-схеме, что и арифметический (см. рис. 16).

Рассмотрим несколько примеров.

Пример 1.21. Составим программу для вычисления числа π.

Впишем в единичную окружность правильный шестиугольник. Его полупериметр, равный трем, будет служить нам первым приближенным значением π. Далее, будем удваивать число сторон вписанного многоугольника. Обозначим половину стороны шестиугольника через b_0, двенадцатиугольника — через b_1, двадцатичетырехугольника — через b_2 и т. д., а их полупериметры — соответственно через S_0, S_1, S_2, ...

Очевидно, $S_0 = 6b_0$, $S_1 = 12b_1 = 6 \cdot 2b_1$, $S_2 = 24b_2 = 6 \cdot 2^2 b_2$, ... Значения S_0, S_1, S_2, ... будут служить последовательными приближенными значениями π, причем

$$\lim_{n \to \infty} S_n = \pi.$$

Для определения π с нужной точностью следует дойти до такого n, чтобы

$$|S_n - S_{n-1}| < \varepsilon,$$

где величина ε определяется требуемой точностью. Тогда π можно будет принять равным S_n.

Для того чтобы иметь возможность последовательно вычислять величины S_0, S_1, S_2, ..., выведем формулу, связывающую половины сторон b_{n-1} и b_n двух вписанных многоугольников.

На рис. 18 изображена сторона AB вписанного многоугольника с числом сторон $6 \cdot 2^{n-1}$ и две стороны AC и CB вписанного многоугольника с удвоенным числом сторон $6 \cdot 2^n$.

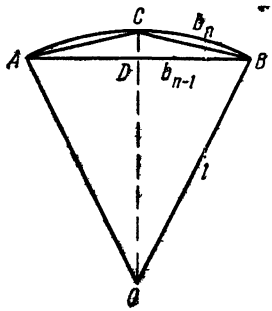


Рис. 18.

Применяя теорему Пифагора последовательно к треугольникам OBD и CBD, придем к соотношению

$$b_n = \sqrt{\frac{1-\sqrt{1-b_{n-1}^2}}{2}}.$$

При помощи этого соотношения, зная $b_0 = \frac{1}{2}$, можно последовательно вычислять b_1, b_2, b_3, ... Полупериметр многоугольника, дающий приближенное значение π, определится формулой $S_n = q_n b_n$, где $q_n = 6 \cdot 2^n$.

Приведенные формулы дают возможность находить последовательные приближенные значения для π. Условия окончания счета можно записать в виде

$$|S_n - S_{n-1}| - \varepsilon < 0.$$

Приступая к составлению программы, напишем прежде всего подготовительную часть цикла, в которой заготовим величины b_{n-1}, S_{n-1}, q_{n-1} для шестиугольника, т. е. для $n = 1$:

1) «0» + «6» = q
2) «0» + «3» = S_{n-1}
3) «0» + «$\frac{1}{2}$» = b_{n-1}.

В рабочей части цикла для получения S_n нам нужно, удваивая число сторон многоугольника, провести вычисление b_n через b_{n-1} по приведенной выше формуле и затем вычислить S_n:

4) q + q = q
5) b_{n-1} · b_{n-1} = R
6) «1» − R = R
7) $\sqrt{R}$ = R
8) «1» − R = R
9) R · «$\frac{1}{2}$» = R
10) $\sqrt{R}$ = b_n
11) q · b_n = S_n.

Проверку окончания цикла следует организовать так:

12) $S_n - S_{n-1} = R$
13) $|R| - |\varepsilon| = 0$
14) $У_0$ 4).

Для перехода к следующему шагу цикла необходимо переслать значение b_n в ячейку b_{n-1} и S_n в S_{n-1}. Одну из этих пересылок можно совместить с условной передачей управления 14). Поэтому команду 14) нужно заменить командой

$$У_0 \ \overrightarrow{S_n \ 4) \ S_{n-1}}.$$

Кроме того, между командами 11) и 12) вставим команду «0» $+ b_n = b_{n-1}$. После окончания цикла пересылаем S_n в ячейку π и останавливаемся:

15) «0» $+ S_n = \pi$
16) *стоп.*

Таким образом, программа вычисления π имеет следующий вид:

1) «0» $+$ «6» $= q$
2) «0» $+$ «3» $= S_{n-1}$
3) «0» $+$ «$\frac{1}{2}$» $= b_{n-1}$
4) $q + q = q$
5) $b_{n-1} \cdot b_{n-1} = R$
6) «1» $- R = R$
7) $\sqrt{R} = R$
8) «1» $- R = R$
9) $R \cdot$ «$\frac{1}{2}$» $= R$
10) $\sqrt{R} = b_n$
11) $q \cdot b_n = S_n$
12) $0 + b_n = b_{n-1}$
13) $S_n - S_{n-1} = R$
14) $|R| - |\varepsilon| = 0$
15) $У_0 \ S_n \ | \ S_{n-1}$
16) $0 + S_n = \pi$
17) *стоп.*

Заметим, что эту программу можно несколько сократить. Прежде всего, можно совместить ячейки b_n и b_{n-1} в одной ячейке b_n, тогда команда 12) окажется излишней.

Полупериметр S_n можно сразу помещать в ячейку π, тогда окажется ненужной пересылка 16). Кроме того, можно команду 1) заменить командой

$$«3» + «3» = q,$$

что позволяет сократить число необходимых констант.

Предоставляем читателю возможность написать сокращенную программу.

Пример 2.21. Вычислим кубический корень из числа x, зная приближенное значение x_0 этого корня.

Для извлечения кубического корня воспользуемся последовательными приближениями. Если известно некоторое приближенное значение $\sqrt[3]{x}$, равное y_n, то следующее приближение можно получить по формуле

$$y_{n+1} = \frac{1}{3}\frac{x}{y_n^2} + \frac{2}{3}y_n.$$

Легко убедиться*), что если предел $\lim\limits_{n\to\infty} y_n = y$ существует, то $y = \sqrt[3]{x}$. В самом деле, перейдя к пределу в равенстве, определяющем y_{n+1}, и заметив, что y_n и y_{n+1} имеют одинаковые пределы y, мы придем к равенству

$$y = \frac{1}{3}\frac{x}{y^2} + \frac{2}{3}y,$$

откуда после простых алгебраических преобразований находим $y^3 = x$, т. е. $y = \sqrt[3]{x}$.

Теперь легко написать программу цикла следующим образом:

1) $0 + y_0 = y_n$
2) $y_n \cdot y_n = R_1$
3) $x : R_1 = R_1$
4) $R_1 \cdot «\frac{1}{3}» = R_1$
5) $y_n \cdot «\frac{2}{3}» = R_2$
6) $R_1 + R_2 = y$
7) $y - y_n = R_1$
8) $|R_1| - |\varepsilon| = 0$ $\xrightarrow{\qquad}$
9) $y_0\ y \qquad 2) \qquad y_n$
10) *стоп.*

*) Вывод этой формулы и доказательство сходимости см. в § 58.

Команда 1) образует здесь подготовку цикла, команды 2)—6)— рабочую часть цикла, а команды 7) — 9) — проверку окончания. Команды изменения здесь, как и в предыдущем примере, отсутствуют, поэтому после подготовки цикла мы сразу переходим к рабочей части.

Заметим еще, что, кроме ячеек x и y_0, аргументами программы являются также ячейки, содержащие константы $\frac{1}{3}$ и $\frac{2}{3}$.

Пример 3.21. Найти сумму бесконечного ряда

$$S=1-\frac{1}{1\cdot 3}+\frac{1}{1\cdot 3\cdot 5}-\ldots+(-1)^k\frac{1}{1\cdot 3\cdot\ldots\cdot(2k+1)}+\ldots$$

с точностью до заданного ε.

Так как члены ряда монотонно убывают по абсолютной величине и изменяют знаки, то ряд сходится и ошибка замены суммы ряда его частичной суммой не превосходит абсолютной величины первого из отброшенных членов. Это дает возможность организовать проверку окончания цикла. Если член ряда u_k с последним множителем $n=2k-1$ в знаменателе уже вычислен, то следующий член можно получить по формуле

$$u_{k+1}=-u_k:(n+2).$$

После сделанных замечаний напишем программу вычисления S:

```
1)    «0» + «1» = u
2)    «0» + «1» = n
3)  Б «0»       | S
4)      n + «2» = n
5)    «0» -  u  = u
6)      u :  n  = u
7)      S +  u  = S
8)     |u| - |ε| = 0
9)  У0
10) стоп.
```

Здесь команды 1) — 3) являются командами подготовки цикла, команды 4) — 5) — изменение, команды 6) — 7) — рабочая часть и, наконец, команды 8) — 9) — проверка окончания.

Поскольку число повторений в итерационном цикле заранее определить невозможно, для таких задач нельзя написать бесцикловую программу даже теоретически, что в принципе возможно для арифметических циклов.

§ 22. Цикл в цикле

Простейшие арифметические или итерационные циклы, рассмотренные нами в предыдущих параграфах, обычно являются составными частями более сложных программ. В частности, такие циклы могут находиться внутри других циклов, являясь их рабочими частями.

Таким образом, обычный цикл (его называют в таких случаях *внутренним*) оказывается «вложенным» в другой (*внешний*) цикл.

Рассмотрим два примера задач такого рода.

Пример 1.22. Вычислить число e (основание натуральных логарифмов) с точностью до $\varepsilon = 10^{-4}$.

Число e определяется, как известно, при помощи следующего предельного перехода:

$$e = \lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n,$$

поэтому величина

$$e_k = \left(1 + \frac{1}{k}\right)^k$$

дает при достаточно больших k приближенное значение e, сколь угодно близкое к его истинному значению.

При заданном k величину e_k можно получить при помощи простого арифметического цикла, который можно построить по схеме рис. 17,

$$\begin{array}{l} \text{«0»} + \text{«1»} = e_k \\ \text{«1»} : k = R_1 \\ \text{«1»} + R_1 = R_1 \\ k - \text{«1»} = S \\ e_k \cdot R_1 = e_k \\ S - \text{«1»} = S \\ \underline{\qquad\qquad} \\ y_0 \quad стоп. \end{array}$$

Значение k, при котором $e_k \approx e$, с заданной точностью определить заранее трудно. Поэтому мы поступим следующим образом: будем последовательно вычислять $e_1, e_2, e_3, \ldots$, каждый раз проверяя выполнение неравенства

$$|e_{k+1} - e_k| < \varepsilon.$$

Как только это неравенство окажется выполненным, мы положим e равным последнему значению e_{k+1}.

Для последовательного вычисления $e_1, e_2, e_3, \ldots$ нам придется включить составленный ранее простой (внутренний) арифметический цикл во внешний итерационный. Во внешнем цикле мы должны последовательно придавать k значения «1», «2», «3» и осуществлять проверку окончания по достижении требуемой точности.

Для вычисления значения e получается следующая программа:

1) $0 + «1» = e_k$
2) $Б\ «1»$ $\quad k$
3) $k + «1» = k$
4) $0 + «1» = e_{k+1}$
5) $«1» : k = R_1$
6) $1 + R_1 = R_1$
7) $k - «1» = S$
8) $e_{k+1} \cdot R_1 = e_{k+1}$
9) $S - «1» = S$
10) $У_0$
11) $e_{k+1} - e_k = R_1$
12) $|R_1| - |\varepsilon| = 0$
13) $У_0\ e_{k+1}$ $\quad 3)$ $\quad e_k$
14) *стоп.*

Здесь команды 1) — 2) составляют подготовку внешнего цикла, команда 3) — изменение внешнего цикла, которое при первом выполнении программы обходится. Команды 4) — 10) составляют рабочую часть внешнего цикла и одновременно образуют внутренний цикл, команды 11) — 13) — проверка окончания внешнего цикла. Во внутреннем цикле команды 4) — 7) составляют подготовку цикла, команда 8) является рабочей частью, команда 9) — изменение и, вместе с командой 10), также проверка окончания. Значение e получается в ячейке e_{k+1}.

Пример 2.22. Вычислим сумму

$$S = \frac{1}{\sqrt[3]{1}} + \frac{1}{\sqrt[3]{2}} + \frac{1}{\sqrt[3]{3}} + \ldots + \frac{1}{\sqrt[3]{n}}.$$

Обозначим

$$S_k = \frac{1}{\sqrt[3]{1}} + \frac{1}{\sqrt[3]{2}} + \ldots + \frac{1}{\sqrt[3]{k}}.$$

Тогда, очевидно, $S_{k+1} = S_k + \frac{1}{\sqrt[3]{k+1}}$ и $S = S_n$.

Значение S, очевидно, можно получить при помощи простого арифметического цикла. Однако, поскольку в машине нет операции извлечения кубического корня, для получения слагаемых (т. е. внутрь арифметического цикла) придется вставить итерационный цикл для нахождения $\sqrt[3]{k}$. При этом за начальное приближение можно принимать

вычисленное ранее (на предыдущем шаге внешнего цикла) значение $\sqrt[3]{k-1}$.

Начнем с команд восстановления для внешнего цикла. Если отвести для кубического корня ячейку q, то команды восстановления будут иметь вид

$$\begin{aligned} &«0» + «1» = q \\ &«0» + «0» = S \\ &«0» + «1» = k. \end{aligned}$$

Впрочем, последнюю команду надо заменить пересылкой с передачей управления, чтобы обойти команду изменения

$$k + «1» = k.$$

После команды изменения идет рабочая часть, которая содержит внутренний итерационный цикл для вычисления кубического корня; его можно полностью заимствовать из примера 2.21. После него необходимы еще команды

$$\begin{aligned} «1» \;:\; q &= u \\ S + u &= S, \end{aligned}$$

а затем две обычные команды проверки окончания внешнего цикла. Окончательно требуемую программу можно записать так:

$$\begin{aligned}
&«0» + «1» = q \\
&«0» + «0» = S \\
&\overrightarrow{\qquad\qquad} \\
&Б\ «1» \qquad\quad | \quad k \\
&\overline{\qquad\qquad} \\
&\left|\begin{array}{rl} k + «1» &= k \\ «0» + q &= q_1 \\ q_1 \cdot q_1 &= R_1 \\ k \;:\; R_1 &= R_1 \\ R_1 \cdot «\tfrac{1}{3}» &= R_1 \\ q_1 \cdot «\tfrac{2}{3}» &= R_2 \\ R_1 + R_2 &= q \\ q - q_1 &= R_1 \\ |R_1| - |\varepsilon| &= 0 \end{array}\right. \\
&\overrightarrow{\qquad\qquad} \\
&У_0 \quad q \qquad\quad | \quad q_1 \\
&\overline{\qquad\qquad} \\
&\left|\begin{array}{rl} «1» \;:\; q &= u \\ S + u &= S \\ k - n &= 0 \end{array}\right. \\
&\overline{\qquad\qquad} \\
&У_1 \\
&\textit{стоп.}
\end{aligned}$$

В рассмотренных нами примерах мы имели арифметический цикл, вставленный внутрь итерационного, или, наоборот, итерационный, помещенный внутрь арифметического. Наряду с такими задачами нередки и такие, в которых оба цикла одинаковы по своему характеру.

Такие программы, имеющие вид цикла в цикле, называют *двойными циклами*. Иногда приходится иметь дело и с *тройными циклами*, т. е. с двойным циклом, заключенным внутри обычного внешнего. Встречаются циклы и большей кратности.

ГЛАВА V

ПЕРЕВОД ПРОГРАММЫ НА ЯЗЫК МАШИНЫ

§ 23. Ячейка памяти. Представление команды в машине

В предыдущей главе мы составляли программы решения различных вычислительных задач в буквенной форме. Процесс решения задачи расчленялся на элементарные операции — команды, которые может выполнять машина. Каждая команда представлялась в буквенном виде, например:

$$a + b = c,$$

где a, b, c — обозначения I, II и III адресов, а «$+$» — обозначение кода операции сложения. Однако машина не воспринимает букв и символов, а «понимает» лишь язык двоичных чисел. Поэтому, чтобы машина могла выполнить заданную последовательность элементарных операций, записанных нами в буквенном виде, необходимо эту программу перевести на двоичный язык машины. В этой главе мы и займемся таким переводом.

Как мы уже говорили, память машины состоит из ячеек, в каждой из которых находится двоичное число. Мы будем далее заниматься трехадресной машиной, память которой содержит $4096_{(10)} = 2^{12}$ ячеек. Ячейки памяти занумерованы подряд от 0 до 4095. Фактически в устройствах машины (*арифметике* и *управлении*) ячейки нумеруются двенадцатиразрядными двоичными числами. Самый младший адрес равен $000\,000\,000\,000_{(2)} = 0_{(10)}$, самый старший изображается двенадцатью единицами $111\,111\,111\,111_{(2)} = 2^{12} - 1 = 4095_{(10)}$.

Такая нумерация ячеек удобна для машины, ибо машина работает в двоичной системе счисления. Однако двоичные адреса очень громоздки и поэтому неудобны для нумерации ячеек человеком. Для записи адресов ячеек на бумаге используется восьмеричная система счисления.

Как было показано в § 13, для перевода целого двоичного числа в восьмеричное следует разбить двоичное число справа налево

на триады (тройки двоичных цифр) и каждую триаду заменить соответствующей восьмеричной цифрой согласно табличке:

Триады	Восьмеричные цифры	Триады	Восьмеричные цифры	Триады	Восьмеричные цифры
000	0	011	3	110	6
001	1	100	4	111	7
010	2	101	5		

Двенадцатиразрядный двоичный адрес содержит, очевидно, четыре триады. Поэтому ячейки памяти получают восьмеричные адреса от $0000_{(8)}$ до $7777_{(8)} = 4095_{(10)}$. Так, первая ячейка памяти имеет адрес 0001, 58_{10}-я — адрес 72_8, $3095_{(10)}$-я — адрес 6025_8. В дальнейшем мы будем пользоваться только восьмеричной записью адресов ячеек памяти.

Рассмотрим теперь структуру ячейки памяти. Каждая ячейка состоит из сорока пяти двоичных разрядов. Эти разряды нумеруются

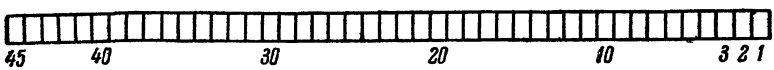


Рис. 19.

(для наглядности) десятичными числами от 1 до 45 справа налево, как показано на рис. 19.

В ячейку может быть записан набор из сорока пяти двоичных цифр (нулей или единиц), который называют *словом*. Особую роль играет ячейка с нулевым адресом — 0000. В этой ячейке всегда содержится *нулевое слово*, т. е. слово, состоящее из сорока пяти нулевых двоичных разрядов, и запись другого содержимого в эту ячейку невозможна. Таким образом, содержимое этой ячейки, в отличие от всех других ячеек памяти, не меняется.

Слово, содержащееся в ячейке памяти, может быть использовано в программе либо как команда, либо как двоичное число с плавающей запятой, либо как десятичное число с плавающей запятой, либо каким-нибудь иным способом.

Рассмотрим сначала, как представляется в трехадресной машине команда. Команда в машине занимает одну ячейку памяти и делится на две основные части — *операционную* и *адресную*. Адресная часть ячейки в трехадресной машине делится в свою очередь на три части: I, II и III адреса по схеме, показанной на рис. 20.

В трех адресах команды указываются обычно номера (адреса) ячеек памяти, участвующих в операции, код которой указан в

операционной части. Как мы видели, для изображения номера (адреса) любой ячейки памяти достаточно 12 двоичных разрядов. Поэтому на адресную часть в командной ячейке отводится 36 двоичных разрядов (по 12 разрядов на каждый адрес). Разряды эти распределяются по следующей схеме (рис. 21): левый адрес (36—25 разряды) является первым, средний (24—13 разряды) — вторым, правый (12—1 разряды) — третьим.

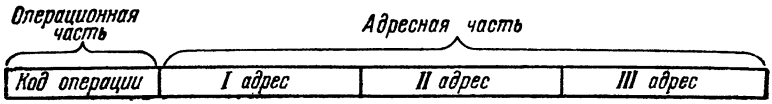


Рис. 20.

Рис. 21.

Разряды с 37 по 42 отводятся под код операции. Таким образом, код операции представляется шестиразрядным двоичным числом. Поскольку каждому такому числу может соответствовать определенная элементарная операция, то в машине может быть осуществлено не более $2^6 = 64_{10}$ элементарных операций (в предыдущей главе мы познакомились с десятью элементарными операциями). Разряды 43—45-й пока использоваться не будут, и мы будем полагать, что в них записаны нули.

Для того чтобы получить какую-либо команду, например:

$$a + b = c,$$

в том виде, в котором ее может выполнить машина, следует прежде всего выделить в памяти ячейки для величин a, b, c, т. е. приписать этим величинам определенные адреса. Пусть эти адреса суть

a	5712
b	0654
c	4361

Кроме того, нужно знать код операции сложения. Как и адреса ячеек, код тоже удобно записывать в восьмеричной форме. Шесть

разрядов, отведенных для записи кода, разбивают на две триады, каждая из которых записывается в виде восьмеричной цифры. Таким образом, код операции есть двузначное восьмеричное число. Код операции сложения изображается восьмеричным числом 01.

Команда $a+b=c$ в ячейке машины представляется следующим двоичным словом:

000 000 001	101 111 001 010	000 110 101 100	100 011 110 001
код операции	I адрес	II адрес	III адрес

При записи команды на бумаге команда представится в виде 001 5712 0654 4361 *).

В такой восьмеричной записи эта команда расшифровывается так: взять числа из ячеек с адресами 5712 и 0654, произвести над ними операцию с кодом 01 (т. е. сложение), результат (сумму) записать в ячейку с адресом 4361.

Подобная замена буквенной записи каждой команды программы восьмеричной записью называется *кодированием программы*. Буквенную запись программы называют записью *в содержательных обозначениях*.

§ 24. Кодирование программы

Прежде чем произвести кодирование программы, надо выделить определенные ячейки памяти для исходных данных, рабочих ячеек, результатов счета и команд программы. Эта процедура называется *распределением памяти*.

Распределим, например, память для первого варианта программы в примере 1.17. Поставим в соответствие буквам a, b, c, x, y, R_1, R_2, R_3, R_4 следующие ячейки памяти:

a	2011	R_1	2001
b	2012	R_2	2002
c	2013	R_3	2003
x	2014	R_4	2004
y	2015		

Шесть команд программы 1) — 6) поместим в последовательных ячейках памяти от 1000 до 1005.

Для того чтобы закодировать программу, надо еще знать коды используемых в ней элементарных операций.

---

*) Напомним, что самую левую восьмеричную цифру, соответствующую триаде из 45—43 разрядов, мы пока полагаем равной нулю.

Арифметическим операциям и операции остановки машины присвоены следующие коды:

Название операции	Обозначение	Код
Сложение	$a + b = c$	01
Вычитание	$a - b = c$	02
Умножение	$a \cdot b = c$	05
Деление	$a : b = c$	04
Вычитание модулей	$\lvert a\rvert - \lvert b\rvert = c$	03
Извлечение квадратного корня	$\sqrt{a} = c$	44
Остановка	*стоп*	17

Разделим лист для программы на две половины — левую и правую. В левой части напишем буквенную программу, а в правой будем писать закодированную. Правую половину листа разграфим на пять столбиков: адреса команд, код операции, IA, IIA, IIIA.

Приступая к кодированию, заполним прежде всего первый столбец правой части, занося в него, в соответствии с принятым распределением памяти, адреса команд 1000, 1001, 1002, 1003, 1004, 1005 (табл. 1.24).

Затем кодируем поочередно каждую команду, начиная с 1). Команда 1), записанная в левой части как $b \cdot x = R_1$, в кодированном виде должна выглядеть так:

$$05, \text{ адрес } b, \text{ адрес } x, \text{ адрес } R_1.$$

Согласно табличке распределения памяти

$$\text{адрес } b = 2012,$$
$$\text{адрес } x = 2014,$$
$$\text{адрес } R_1 = 2001.$$

Поэтому в правой части команду 1) следует писать в виде

$$005\ 2012\ 2014\ 2001$$

Так же кодируются и команды 2), 3), 4), 5). У команды 6) в правой части проставляется код операции 17 и нулевые адреса (табл. 1.24).

Программирование, распределение памяти и кодирование производят на специальных бланках, отпечатанных типографским способом. В таблицах 2.24 и 3.24 приведены образцы программного бланка и части бланка для распределения памяти.

Бланк распределения памяти представляет собой таблицу с написанными восьмеричными адресами ячеек. Бланки бывают различного

Таблица 1.24

	Вычисление значения квадратного трехчлена	Адреса команд	Код операции	IA	IIA	IIIA
1)	$b \cdot x = R_1$	1000	0 | 05	2012	2014	2001
2)	$R_1 + c = R_2$	1001	0 | 01	2001	2013	2002
3)	$x \cdot x = R_3$	1002	0 | 05	2014	2014	2003
4)	$a \cdot R_3 = R_4$	1003	0 | 05	2011	2003	2004
5)	$R_2 + R_4 = y$	1004	0 | 01	2002	2004	2015
6)	*стоп*	1005	0 | 17	0000	0000	0000

формата и на различное количество адресов. Обычно один бланк содержит $1000_8 = 512_{10}$ или $2000_8 = 1024_{10}$ клеток и для всей памяти требуется соответственно восемь или четыре бланка. В некоторых случаях в каждой клетке проставляются цифры от 000 до 777, а первая цифра адреса, т. е. номер восьмеричной тысячи, проставляется вручную. Такой бланк с записанными в нем сведениями о данной задаче мы будем называть *памяткой*. Его называют также *словарем*. Впрочем, чаще всего такой бланк называют *шпаргалкой*.

Для облегчения кодирования буквенную программу нужно писать четко и аккуратно, стараясь, чтобы буквы, соответствующие первому, второму и третьему адресам, располагались одна под другой, а буквы $У_0$, $У_1$, $Б$, заменяющие коды операции, находились ближе к левому краю бланка. Иногда левую часть бланка делят тонкими линиями на колонки, соответствующие разбиению правой части.

Таблица 2.24

составил	задача	блок	стр.	лист

				A	шифр № карты

				A	шифр № карты

кодировала:
проверила кодировку:

перфорировала:
проверила карты:

Таблица 3.24

5000	5040	5100	5140	5200
5001	5041	5101	5141	5201
5002	5042	5102	5142	5202
5003	5043	5103	5143	5203
5004	5044	5104	5144	5204
5005	5045	5105	5145	5205
5006	5046	5106	5146	5206
5007	5047	5107	5147	5207
5010	5050	5110	5150	5210
5011	5051	5111	5151	5211
5012	5052	5112	5152	5212
5013	5053	5113	5153	5213
5014	5054	5114	5154	5214
5015	5055	5115	5155	5215
5016	5056	5116	5156	5216
5017	5057	5117	5157	5217
5020	5060	5120	5160	5220
5021	5061	5121	5161	5221
5022	5062	5122	5162	5222

Коды операций передачи управления

Название операции	Обозначение	Код
Безусловная передача управления	$Б\ \overrightarrow{a\ b\ c}$	56
Условная передача управления при $\omega = 0$	$У_0\ \overrightarrow{a\ b\ c}$	76
Условная передача управления при $\omega = 1$	$У_1\ \overrightarrow{a\ b\ c}$	36

Таблица 4.24

3000	a	3040	1)	3100
3001	b	3041	2)	3101
3002	c	3042	3)	3102
3003	x_1	3043	4)	3103
3004	x_1'	3044	5)	3104
3005	x_2	3045	6)	3105
3006	x_2'	3046	7)	3106
3007	D	3047	8)	3107
3010	ρ	3050	9)	3110
3011	α	3051	10)	3111
3012	β	3052	11)	3112
3013	R_1	3053	12)	3113
3014	R_2	3054	13)	3114
3015		3055	14)	3115
3016		3056	15)	3116
3017		3057	16)	3117
3020		3060	17)	3120
3021		3061	18)	3121
3022		3062	19)	3122
3023		3063	20)	3123
3024		3064		3124
3025		3065		3125
3026		3066		3126

Программа

Таблица 5.24

					3040	A	
1)	$a + a = \rho$	3040	0	01	3000	3000	3010
2)	$b : \rho = R_1$	1	0	04	3001	3010	3013
3)	«0» $- R_1 = \alpha$	2	0	02	0000	3013	3011
4)	$b \cdot b = R_1$	3	0	05	3001	3001	3013
5)	$\rho \cdot c = R_2$	4	0	05	3010	3002	3014
6)	$R_2 + R_2 = R_2$	5	0	01	3014	3014	3014
7)	$R_1 - R_2 = D$	6	0	02	3013	3014	3007
8)	y_1 $\overrightarrow{\alpha \quad 15)}$ $\overrightarrow{x_1}$	7	0	36	3011	3056	3003
9)	$\sqrt{D} = R_1$	3050	0	44	3007	0000	3013
10)	$R_1 : \rho = \beta$	1	0	04	3013	3010	3012
11)	$\alpha + \beta = x_1$	2	0	01	3011	3012	3003

					3053	A	
12)	$\alpha - \beta = x_2$	3053	0	02	3011	3012	3005
13)	«0» + «0» $= x_1'$	4	0	01	0000	0000	3004
14)	$Б$ $\overrightarrow{\text{«0»} \quad 20)}$ $\overrightarrow{x_2'}$	5	0	56	0000	3063	3006
15)	«0» $- D = R_1$	6	0	02	0000	3007	3013
16)	$\sqrt{R_1} = R_1$	7	0	44	3013	0000	3013
17)	$R_1 : \rho = x_1'$	3060	0	04	3013	3010	3004
18)	«0» $- x_1' = x_2'$	1	0	02	0000	3004	3006
19)	«0» $+ x_1 = x_2$	2	0	01	0000	3003	3005
20)	*стоп*	3	0	17	0000	0000	0000

Программный бланк разделен на две группы по 12_{10} строк. В строках со второй по двенадцатую записываются команды программы. Место в первой строке слева от буквы А предназначено для *адресного слова*. Сюда записывается адрес ячейки памяти, в которую помещается команда, написанная в следующей (второй) строке. Записанные далее команды помещаются подряд в ячейки со следующими номерами.

Для кодирования программ необходимо знать коды всех элементарных операций машины. Коды арифметических операций были уже приведены на стр. 104. На стр. 107 приведены коды уже рассмотренных нами операций управления; коды остальных операций будут приводиться по мере их введения.

В табл. 4.24 приведена памятка для программы решения квадратного уравнения. Аргументы, рабочие и ответные ячейки программы размещены в ячейках от 3000 до 3014. Для программы отведены ячейки 3040—3063. Программа в содержательных обозначениях и в закодированном виде приведена в табл. 5.24.

В предыдущем параграфе было отмечено, что в ячейке с адресом 0000 (нулевой ячейке) всегда находится нулевое слово. Далее мы увидим, что это слово, использованное как двоичное число, означает нуль. Поэтому в командах 3), 13), 14), 15), 18) и 19) число «0» закодировано нулевым адресом.

Заметим также, что в рассматриваемой программе приходится кодировать, наряду с обозначениями рабочих ячеек, исходных данных и результатов счета, еще и номера 15), 20) команд, которым передается управление.

Нумерация всех команд программы загромождает ее запись и является излишней. Вполне достаточно нумеровать лишь те команды, которым передается управление, ибо номера только этих команд участвуют в кодировании.

Более удобно применять не нумерацию, а различные условные обозначения команд. Эти номера или условные обозначения мы будем в дальнейшем называть *метками* команд. Метки команд пишут на бланке на месте номеров и наряду с буквенными обозначениями ячеек вносят в памятку.

§ 25. Представление двоичных чисел в машине

Как уже говорилось в гл. III, машина может производить операции лишь над двоичными числами. Как же изображаются двоичные числа в машине?

Арифметические действия в трехадресной машине производятся над двоичными числами с плавающей запятой, т. е. над числами, представленными в виде $M \cdot 2^p$, где p — двоичный порядок числа, а M — его мантисса.

Сорокапятиразрядная ячейка машины, содержащая двоичное число, делится на части по схеме, приведенной на рис. 22.

Мантисса числа занимает разряды с 1-го по 36-й, порядок — с 37-го по 43-й. 44-й разряд характеризует знак числа: если число положительное, в этом разряде ставится нуль, если число отрицательное — единица. 45-й разряд при записи числа не используется; мы будем всегда считать его нулевым.

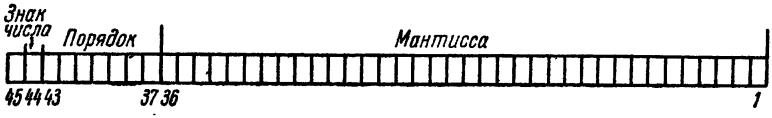


Рис. 22.

Выясним теперь, с какой точностью при указанном распределении разрядов представляются в машине двоичные числа и каков возможный их диапазон.

Точность представления числа с плавающей запятой определяется, очевидно, числом разрядов в мантиссе. Наибольшее значение целого числа, которое может поместиться в разрядах мантиссы, равно $7777\ 7777\ 7777_8 = (8^{12} - 1)_{10} \approx 10^{11}$. Следовательно, машина работает с числами, имеющими (в переводе в десятичную систему) примерно одиннадцать значащих цифр.

Обычно машина работает с нормализованными числами, для которых, как мы знаем,

$$\frac{1}{2} \leqslant M < 1.$$

Это означает, что самый старший (36-й) разряд мантиссы для нормализованного двоичного числа должен быть равен 1. Наибольшее значение мантиссы равно $1 - 2^{-36}$, и, следовательно,

$$\frac{1}{2} \leqslant M \leqslant 1 - 2^{-36}.$$

Диапазон представления чисел в машине определяется величиной порядка. *Машинный порядок* *) p' двоичного числа занимает в ячейке семь двоичных разрядов (с 37-го по 43-й) и, следовательно, может принимать целые положительные значения от $0\,000\,100_2 = 000_8$ до $1\,111\,111_2 = 177_8$:

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

Машинный порядок p' связан с действительным порядком p соотношением

$$p' = p + 100_8.$$

*) Иногда говорят также *условный порядок*.

Отсюда следует, что диапазон изменения действительного порядка p определяется неравенством

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

или

$$-64_{10} \leqslant p \leqslant 63_{10}.$$

Поэтому все машинные двоичные числа x заключены в диапазоне

$$2^{-64}M \leqslant |x| \leqslant 2^{63}M.$$

Таким образом, для нормализованных двоичных чисел

$$2^{-65} \leqslant |x| \leqslant 2^{63}(1-2^{-36}).$$

Вычисляя 2^{+63} и 2^{-65} при помощи таблицы логарифмов, получим приближенное неравенство

$$10^{-19,6} \leqslant |x| \leqslant 10^{19}.$$

Предыдущие рассуждения приводят нас к следующему важному выводу: *трехадресная машина работает с нормализованными двоичными числами, имеющими (в переводе в десятичную систему) примерно одиннадцать значащих цифр и заключенными в диапазоне от $10^{-19,6}$ до 10^{+19}.*

В процессе вычислений по заданной программе результат некоторой арифметической операции может выйти из указанного диапазона. Посмотрим, что же тогда произойдет.

Пусть, например, перемножаются два таких больших (по модулю) числа, что произведение оказывается больше 10^{19}. Тогда машинный порядок результата оказывается большим 177_8 и, следовательно, не может поместиться в разрядной сетке машины (происходит *переполнение* разрядной сетки). Конструкцией машины предусмотрено, что получение результата, не помещающегося в машине, вызывает ее остановку, так называемый *«стоп по переполнению»* или *«аварийный останов»*.

Во избежание аварийного останова следует так организовать алгоритм счета, чтобы ни один из промежуточных результатов вычислений не превосходил 10^{19}. Приведем простейший пример. Для трех десятичных чисел $a=0{,}21\cdot 10^{10}$, $b=0{,}3\cdot 10^{11}$, $c=0{,}5\cdot 10^{7}$ при вычислении величины $x=\dfrac{ab}{c}$ мы выйдем на аварийный останов, если будем считать так: $x=(a\cdot b):c$, ибо $a\cdot b=0{,}63\cdot 10^{20}$. Изменив порядок действий:

$$x=(a:c)\cdot b,$$

мы получим число $x=[(0{,}21\cdot 10^{10}):(0{,}5\cdot 10^{7})]\cdot 0{,}3\cdot 10^{11}=0{,}42\cdot 10^{3}\times$ $\times 0{,}3\cdot 10^{11}=0{,}144\cdot 10^{14}$, укладывающееся в разрядной сетке машины.

Заметим, что аварийный останов происходит не только при переполнении разрядной сетки, но также при попытке деления на нуль или извлечении квадратного корня из отрицательного числа. Нормализованное число, меньшее (по модулю) $10^{-19,6}$, т. е. двоичное число, порядок которого меньше -64_{10}, воспринимается машиной как нуль (так называемый *машинный нуль*). В некоторых случаях машинным нулем может оказаться величина, которая при правильном порядке действий окажется большей $10^{-19,6}$.

Например, при вычислении величины $z=\frac{ab}{c}$, где $a=0{,}21\cdot10^{-10}$, $b=0{,}7\cdot10^{-11}$, $c=0{,}5\cdot10^{-9}$, появится машинный нуль, если производить действия так: $z=(a\cdot b):c$. Действительно, $a\cdot b=(0{,}21\cdot10^{-10})\times\times(0{,}7\cdot10^{-11})=0{,}147\cdot10^{-21}$, и это число воспримется как нуль, а следующее деление даст тоже нулевой результат. Если же производить вычисления в другом порядке: $z=(a:c)\cdot b=[(0{,}21\cdot10^{-10}):(0{,}5\cdot10^{-9})]\times\times(0{,}7\cdot10^{-11})=(0{,}42\cdot10^{-1})(0{,}7\cdot10^{-11})$, то получим число $0{,}294\cdot10^{-14}$, отличное от машинного нуля.

Рассмотрим теперь подробнее вопрос о том, как представить двоичное число в машине.

Из формулы $p'=p+100_8$, связывающей действительный и машинный порядки, видно, что при $p\geqslant0$ имеем $p'\geqslant100_8$; в этом случае в 43-м разряде ячейки, изображающей двоичное число, стоит 1. Если же $-77_8\leqslant p<0$, то $0\leqslant p'\leqslant77_8$, и в 43-м разряде стоит 0.

Поэтому можно считать, что 43-й разряд характеризует знак действительного порядка; однако нуль в этом разряде будет обозначать знак минус, а единица—знак плюс (ср. с обозначениями знака числа в 44-м разряде).

Величина машинного порядка занимает при этом условии разряды от 37-го по 42-й (см. рис. 22), причем, если действительный порядок неотрицателен, величины действительного и машинного порядка совпадают; если же действительный порядок отрицателен, величина машинного порядка является двоичным дополнением величины действительного порядка до $1000000_2=100_8$ *).

Так, числа первого и второго положительного порядков будут иметь машинный порядок 101_8 и 102_8, при отрицательных первом и втором порядках машинные порядки оказываются равными 77_8 и 76_8; нулевому действительному порядку соответствует машинный порядок 100_8.

Исходные числовые данные записывают на бланке следующим образом. В левой части бланка пишется нужное число в десятичной системе счисления. Это число вручную, по способу, указанному в § 14, переводится в нормализованную двоично-восьмеричную форму. В правой части бланка в восьмеричной форме пишется мантисса: 12 цифр

*) В этом случае говорят, что порядок изображается в машине *в дополнительном коде*. О двоичном дополнении см. стр. 47.

мантиссы располагаются по графам правой части бланка тремя группами по четыре восьмеричные цифры, т. е. так же, как адреса команды. Затем действительный порядок двоичного числа, записанный в восьмеричном виде, переводится в машинный порядок, «величина» машинного порядка в виде двух восьмеричных цифр записывается в графе «код операции» *). Знак же машинного порядка вместе со знаком числа (и нулем в 45-м разряде) кодируются одной восьмеричной цифрой, которая ставится перед величиной порядка.

Рассмотрим несколько примеров.

Нулевое слово, рассматриваемое как двоичное число, имеет мантиссу, равную нулю, и действительный двоичный порядок, равный

$$-1000\,000_2 = -64_{10}.$$

Поэтому любая «пустая» ячейка памяти машины содержит двоичное число, равное нулю. Ячейка с номером 0000, содержание которой всегда нулевое, постоянно содержит двоичный нуль. Нуль кодируется так:

0 00 0000 0000 0000.

Закодируем теперь числа $-\left(\frac{3}{16}\right)_{10}$, $\frac{1}{2}$, 1, 2, —5, 9. Очевидно,

$$-\frac{3}{16} = -0{,}11 \cdot 10^{-10}$$
$$\frac{1}{2} = 0{,}1 \cdot 10^{0}$$
$$1 = 0{,}1 \cdot 10^{1}$$
$$2 = 0{,}1 \cdot 10^{10} \quad \text{(в двоичной системе)}$$
$$-5 = -0{,}101 \cdot 10^{11}$$
$$9 = 0{,}1001 \cdot 10^{100}.$$

Машинные порядки этих чисел соответственно равны 076, 100, 101, 102, 103, 104. Учитывая знаки чисел и записывая мантиссы в восьмеричной форме, получим машинное представление:

276	6000	0000	0000
100	4000	0000	0000
101	4000	0000	0000
102	4000	0000	0000
303	5000	0000	0000
104	4400	0000	0000

*) Эту часть работы можно еще упростить, заменив в таблице, приведенной на стр. 57, действительный порядок машинным.

В качестве примера кодирования программы вместе с числовыми исходными данными рассмотрим программу решения системы двух линейных алгебраических уравнений.

Пусть коэффициенты уравнений имеют следующие числовые значения:

$$a_1 = 0{,}01875, \quad b_1 = 1{,}8, \quad c_1 = -0{,}1,$$
$$a_2 = -20{,}5, \quad b_2 = 92{,}4, \quad c_2 = -0{,}06125.$$

Память распределим следующим образом (табл. 1.25). Заполним левую часть программного бланка (табл. 2.25) и закодируем программу обычным способом.

Таблица 1.25

1000		1040	a_1
1001		1041	b_1
1002		1042	c_1
1003		1043	a_2
1004		1044	b_2
1005		1045	c_2
1006	Программа	1046	x
1007		1047	y
1010		1050	D
1011		1051	D_1
1012		1052	D_2
1013		1053	R_1
1014		1054	R_2
1015		1055	

Переведем коэффициенты, выписанные в левой части в десятичной системе, в нормализованную двоично-десятичную и запишем в правой части бланка.

Таблица 2.25

					1000	A	
	$a_1 \cdot b_2 = R_1$	1000	0	05	1040	1044	1053
	$a_2 \cdot b_1 = R_2$	1	0	05	1043	1041	1054
	$R_1 - R_2 = D$	2	0	02	1053	1054	1050
	$b_2 \cdot c_1 = R_1$	3	0	05	1044	1042	1053
	$b_1 \cdot c_2 = R_2$	4	0	05	1041	1045	1054
	$R_1 - R_2 = D_1$	5	0	02	1053	1054	1051
	$a_1 \cdot c_2 = R_1$	6	0	05	1040	1045	1053
	$a_2 \cdot c_1 = R_2$	7	0	05	1043	1042	1054
	$R_1 - R_2 = D_2$	1010	0	02	1053	1054	1052
	$D_1 : D = x$	1	0	04	1051	1050	1046
	$D_2 : D = y$	2	0	04	1052	1050	1047

					1013	A	
	стоп	1013	0	17	0000	0000	0000
					1040	A	
	$a_1 = 0{,}01875$	1040	0	73	4631	4631	4632
	$b_1 = 1{,}8$	1	1	01	7146	3146	3146
	$c_1 = -0{,}1$	2	2	75	6314	6314	6315
	$a_2 = -20{,}5$	3	3	05	5100	0000	0000
	$b_2 = 92{,}4$	4	1	07	5614	6314	6315
	$c_2 = -0{,}06125$	5	2	74	7656	0507	5341

§ 26. Представление десятичных чисел в машине

Перевод вручную десятичных чисел в двоичные является довольно трудоемкой работой. Поэтому обычно только некоторое небольшое количество чисел подготавливается к вводу в машину в двоичном виде. Большие же массивы чисел всегда вводятся в машину в десятичном виде.

Как же изображаются в машине десятичные числа?

Десятичное число с плавающей запятой записывается в виде

$$M \cdot 10^p,$$

где p — величина порядка чисел, а M — мантисса. Ячейка, используемая для записи десятичного числа, делится на части по схеме,

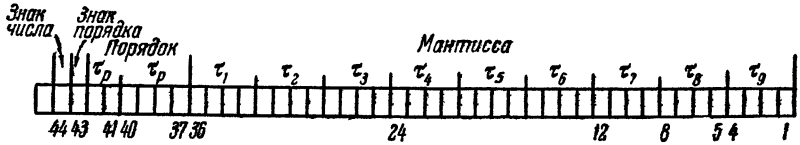


Рис. 23.

приведенной на рис. 23. Эта схема аналогична схеме распределения разрядов ячейки для двоичного числа (см. рис. 22). Действительно, мантисса также занимает разряды от 1-го по 36-й, порядок (вместе со знаком) — разряды с 37-го по 43-й, знак числа — 44-й разряд.

Для положительного десятичного числа в 44-м разряде ставится 0, для отрицательного — 1. Так же кодируется и знак порядка: знаку плюс соответствует 0 в 43-м разряде, знаку минус — единица.

Для изображения цифр мантиссы и порядка в ячейке машины используется *двоично-десятичная система записи чисел.* В 36 разрядах мантиссы умещается девять тетрад, т. е. четверок двоичных цифр, изображающих отдельные десятичные цифры. Поэтому мантисса десятичного числа, вводимого в машину, должна содержать не более девяти десятичных цифр.

Каждая десятичная цифра мантиссы изображается в ячейке соответствующей тетрадой. Эти цифры заносятся в мантиссу, начиная с 36-го разряда, слева направо. Так, например, мантисса 0,957 000 000 изображается в ячейке в соответствии со следующей схемой (табл. 1.26).

Таблица 1.26

Номера разрядов	36	35	34	33	32	31	30	29	28	27	26	25	...	4	3	2	1
Двоичная мантисса	1	0	0	1	0	1	0	1	0	1	1	1	...	0	0	0	0
Десятичная мантисса	9			5			7			...	0						

В шести разрядах, отведенных для величины десятичного порядка, умещается одна тетрада (разряды с 37-го по 40-й) и одна *диада* (пара разрядов, 41 и 42-й разряды). Поэтому максимально допустимая разрядной сеткой машины величина десятичного порядка равна 39_{10}. Однако, как было выяснено в предыдущем параграфе, машина может производить арифметические операции лишь над числами, не превосходящими (по модулю) 10^{19}. Поэтому максимальная величина десятичного порядка равна 19, и, следовательно, в 42-м разряде ячейки, изображающей десятичное число, должен стоять нуль. Таким образом, величина десятичного порядка фактически изображается пятью разрядами — одной *монадой* (41-й разряд) и одной тетрадой (разряды с 37-го по 40-й).

На программном бланке десятичные числа кодируются в следующем порядке. В левой части бланка числа записываются в обычном виде (с запятой, фиксированной на любом месте) и нормализуются. Затем в графу правой части бланка, отведенную под код операции команды, заносятся последовательно знак числа (±), знак порядка (±) и две цифры десятичного порядка; после этого в графы, отведенные для I, II и III адресов команды, записываются цифры мантиссы по три в каждом адресе. На табл. 2.26 закодированы таким образом коэффициенты системы двух линейных уравнений из примера, рассмотренного в предыдущем параграфе.

Таблица 2.26

	Числа			1040	A		
$a_1 = 0{,}01875$	1040	+	− 01	187	500	000	
$b_1 = 1{,}8$	1	+	+ 01	180	000	000	
$c_1 = -0{,}1$	2	−	+ 00	100	000	000	
$a_2 = -20{,}5$	3	−	+ 02	205	000	000	
$b_2 = 92{,}4$	4	+	+ 02	924	000	000	
$c_2 = -0{,}06125$	5	−	− 01	612	500	000	

Заметим, что нулевое слово, рассматриваемое как десятичное число, также означает нуль.

Так как машина может производить счет лишь в двоичной системе счисления, то десятичные числа, введенные в указанном виде в машину, должны быть переведены в двоичную систему. Этот

перевод производится специальной программой, которую мы рассмотрим в гл. VII.

В предыдущем параграфе было показано, что действия над двоичными числами в машине производятся фактически примерно с одиннадцатью знаками (в переводе на десятичную систему). В то же время десятичные числа вводятся в машину (и выводятся из нее) с девятью знаками. Следовательно, *в трехадресной машине расчеты производятся с двумя запасными значащими цифрами.* Это дает возможность в подавляющем большинстве задач получить результаты расчетов с достаточной точностью.

§ 27. Перфорация и ввод

Большинство современных электронных машин не может «читать» цифры, записанные на бумаге. Поэтому, прежде чем вводить числа и команды в машину, их сначала переносят на перфокарты (картонные карточки с пробиваемыми на них отверстиями).

Стандартная перфокарта (рис. 24) состоит из 80 колонок (столбцов) и 12 позиций (строк). В каждую позицию может заноситься одно сорокапятиразрядное слово. Для изображения слова выделено 45 колонок. Пробивка в колонке соответствует двоичной цифре 1, отсутствие пробивки — цифре 0.

Кроме того, на перфокарте выделены две специальные маркерные колонки — основная с номером 18 и вспомогательная с номером 80. На рис. 24 в верхней позиции указаны пробивками 45 основных и две маркерные колонки.

В позицию (строку) перфокарты могут заноситься обычные и специальные слова.

Обычные слова — это числа или команды, вводимые с перфокарт в память машины. Признаком того, что в позицию занесено обычное слово, является пробивка в основной маркерной дорожке.

На рис. 24 изображено, как пробивается на карте команда или двоичное число (для наглядности код операции и три адреса разнесены на четыре позиции: пятую, шестую, седьмую и восьмую).

Кроме обычных слов, на перфокарте могут быть пробиты и специальные слова двух видов: адресные и контрольные.

Признаком *адресного слова* является пробивка на вспомогательной дорожке. Адресное слово занимает только колонки, соответствующие первому адресу в обычном командном слове.

Как мы уже упоминали в § 24, обычные слова, идущие после адресного слова, записываются при вводе в машину в последовательные ячейки памяти, начиная с ячейки, указанной в адресном слове. Каждая перфокарта обычно начинается с адресного слова. На программном бланке размещается материал для пробивки двух перфокарт.

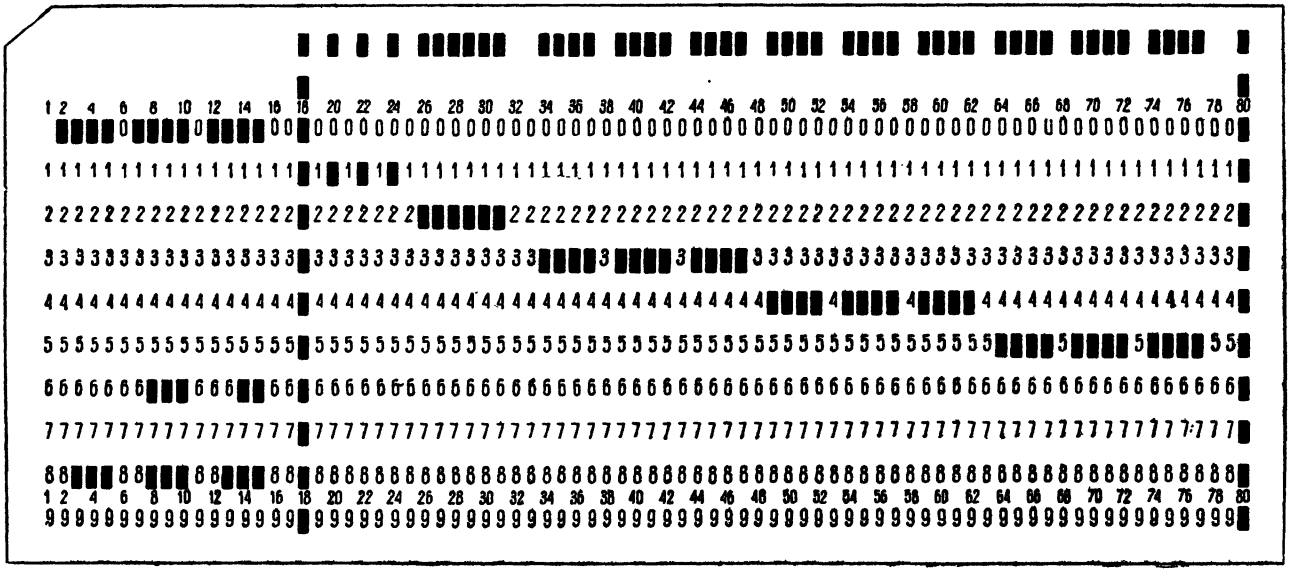


Рис. 24.

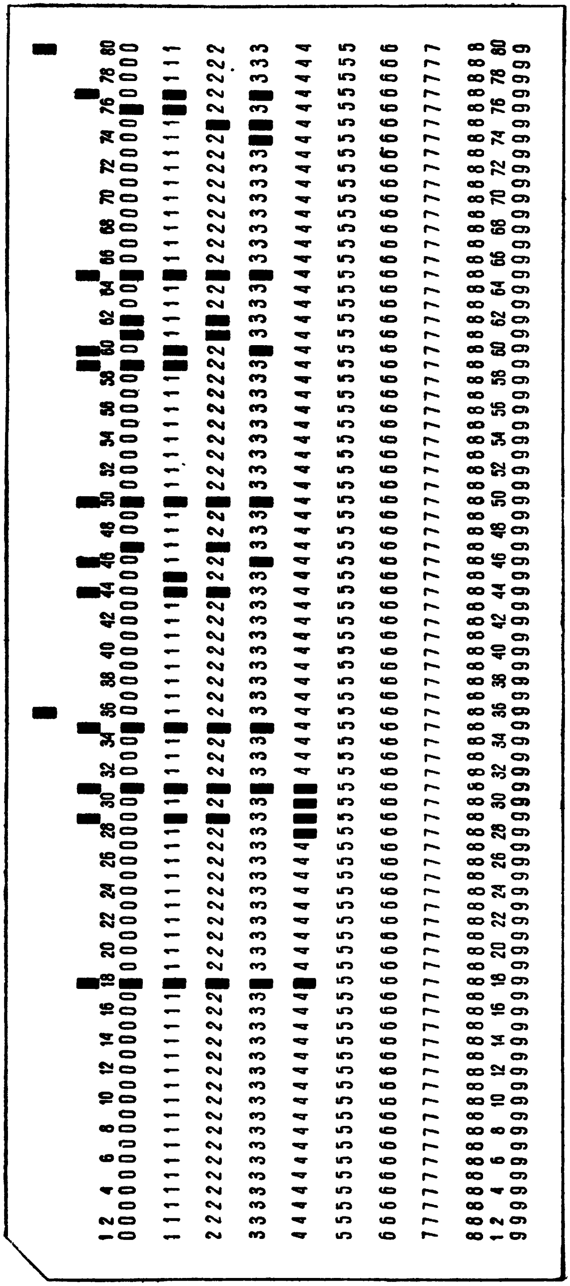


Рис. 25.

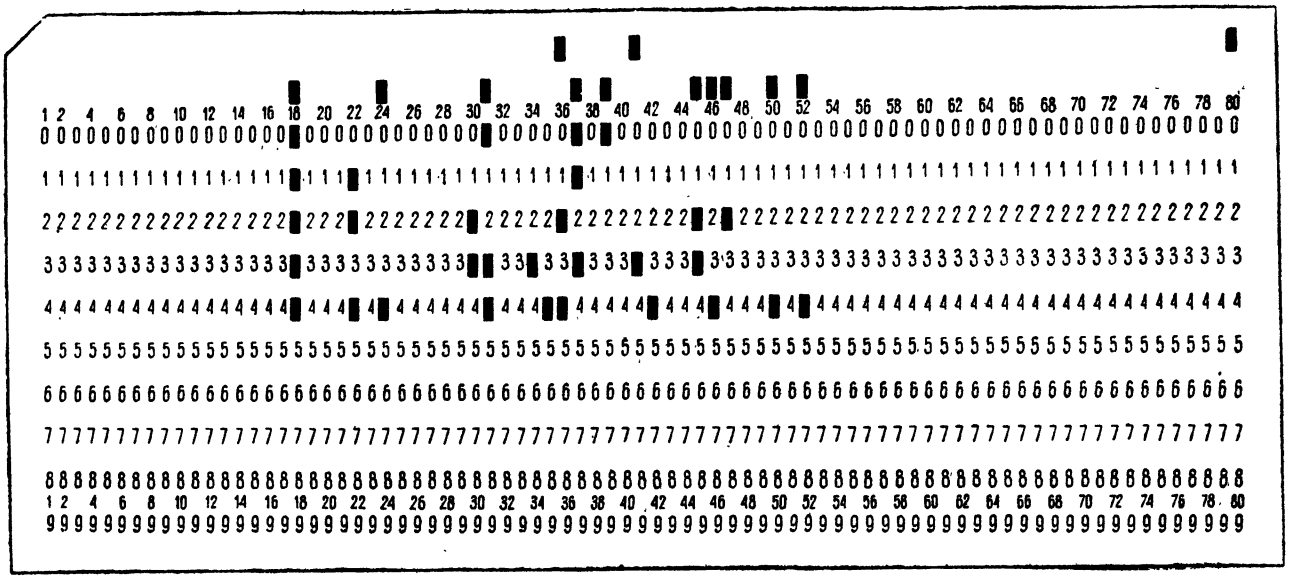


Рис. 26.

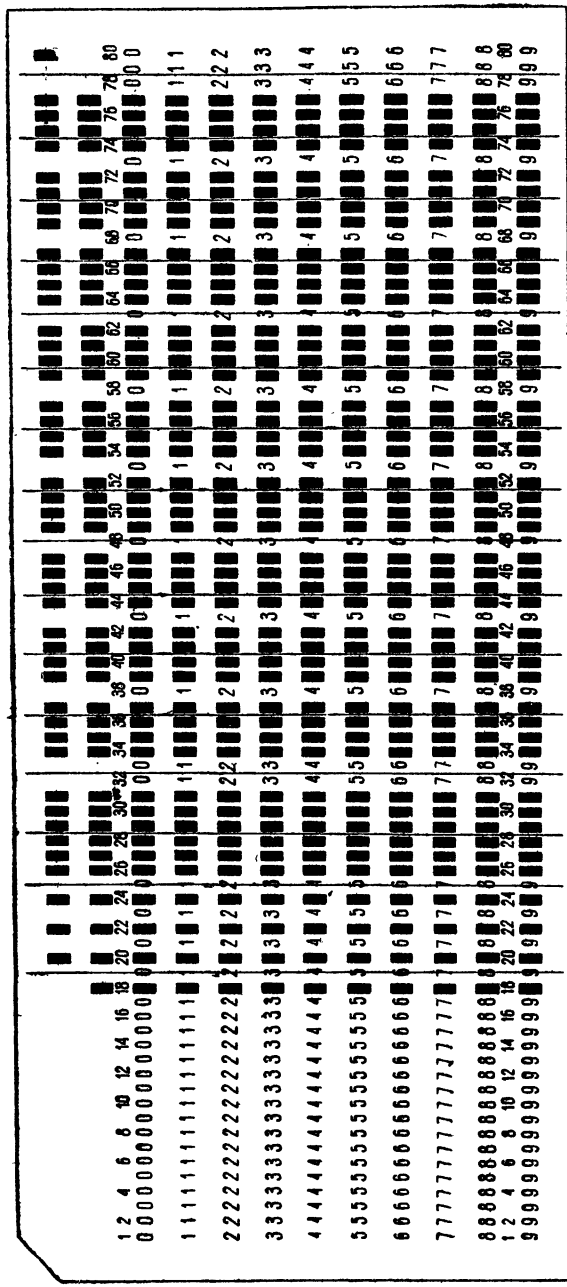


Рис. 27.

Контрольное слово отличается от других слов пробивками в обоих маркерных дорожках. Наличие этого слова дает сигнал машине о прекращении ввода материала с перфокарт в память *).

В качестве примера приведем вид перфокарты (рис. 25) с программой нахождения значения квадратного трехчлена (табл. 1.24). В первой позиции перфокарты пробит дополнительный маркер (для адресного слова), в остальных — основной маркер (для обычных слов). На рис. 26 изображена перфокарта с пробитыми на ней десятичными числами (ср. с табл. 2.26).

Пробивка программы на перфокартах производится при помощи специального перфоратора. Перфоратор может работать в двух режимах — восьмеричном и десятичном. Переход с одного режима на другой производится при помощи специального тумблера.

В восьмеричном режиме пробиваются команды и двоичные числа, причем при нажатии восьмеричной цифры на клавиатуре происходит автоматический перевод этой цифры в соответствующую триаду. В десятичном режиме (при пробивке десятичных чисел) происходит перевод нажатой на клавиатуре десятичной цифры в соответствующую тетраду (для пробивки знаков числа и порядка имеются специальные клавиши $+$ и $-$).

Во избежание путаницы при пробивке, десятичные числа следует писать на отдельных бланках с надписью «числа».

Пробитые перфокарты контролируются одним из двух способов:

а) чтением при помощи специального шаблона (рис. 27);

б) выпечатыванием на бумажную ленту и считыванием с программой.

Колода перфокарт с набитыми на них командами и числами через специальное вводное устройство (*ввод*) поступает в память машины.

Кодирование, перфорация и ввод в машину команд и числового материала является чисто технической работой, не вызывающей, при небольшом навыке, трудностей. Поэтому в следующих главах книги мы ограничимся в основном написанием содержательных частей программ, не производя их кодировки.

*) Кроме того, это слово имеет и другое специальное назначение, о котором мы скажем ниже.

ГЛАВА VI

ПЕРЕАДРЕСАЦИЯ

§ 28. Действия над числами с фиксированной запятой

В главе IV мы познакомились с десятью элементарными операциями трехадресной машины. В этой главе мы изучим еще некоторые из них.

В электронных вычислительных машинах применяются операции над числами как с плавающей, так и с фиксированной запятой. Арифметические операции производятся над словами, изображающими числа с плавающей запятой. Кроме них, существуют еще операции, в которых слово, записанное в ячейке, или некоторая его часть, рассматривается как целое число с фиксированной запятой. Такие операции называются *фиксированными действиями.*

Мы рассмотрим фиксированные действия, которые совершаются не над полным содержанием ячейки, а над ее частями. Как было указано в предыдущей главе, при записи числа или команды ячейка делится на две неравные части, состоящие соответственно из 9 и 36 разрядов.

Правая часть ячейки, состоящая из разрядов 1—36, при записи команды отводится для ее адресной части, а при записи числа — для мантиссы. Операции сложения и вычитания этих частей слова, рассматриваемых как целые числа с фиксированной запятой, можно назвать соответственно *сложением* и *вычитанием мантисс* или же *сложением* и *вычитанием адресных частей* *). Мы будем обозначать эти операции следующим образом:

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

В операции $a+,\ b=c$ мантисса слова c образуется сложением мантисс a и b как целых чисел с фиксированной запятой, а порядок

*) Более распространенным является название *сложение мантисс*, хотя название *сложение адресных частей* следует считать более удачным, так как эта операция используется главным образом при преобразованиях команд, как это будет видно в следующих параграфах.

(кодовая часть) слова c полагается равным порядку a. Таким образом, результат операций $a+$, b и $b+$, a будет, вообще говоря, различным.

При сложении может возникнуть необходимость переноса единицы из 36-го разряда в старший разряд. Поскольку 37-й разряд ячейки не относится уже к мантиссе, то этот перенос не происходит и единица переноса теряется. Признаком того, что такой перенос должен был произойти, является выработка управляющего сигнала $\omega = 1$. Если же сумма мантисс a и b умещается в пределах 36 разрядов мантиссы c, то вырабатывается сигнал $\omega = 0$.

Операция $a-$, $b = c$ выполняется так: порядок слова c полагается равным порядку уменьшаемого (a). Мантисса c находится вычитанием мантисс a и b, рассматриваемых как целые числа с фиксированной запятой. Если мантисса b не превосходит мантиссы a, то вычитание происходит обычным образом и вырабатывается сигнал $\omega = 0$. Если же мантисса a меньше мантиссы b, то вычитание происходит так, как будто к уменьшаемому слева была приписана единица 37-го разряда. В этом случае при вычитании мантисс вырабатывается управляющий сигнал $\omega = 1$.

Аналогичные операции можно определить и для левых частей ячейки, состоящих из 37—45 разрядов. Эта часть ячейки при записи команды отводится для кода операции, а при записи числа — для его порядка. Поэтому соответствующие операции, которые мы будем обозначать

$$a, + b = c,$$

$$a, - b = c,$$

носят названия *сложения* и *вычитания порядков* или же *сложения* и *вычитания кодовых частей* *).

В операции $a, + b = c$ порядок слова c равен сумме порядков слов a и b, рассматриваемых как целые числа с фиксированной запятой. Мантисса слова c переносится целиком из мантиссы a. Если возникает необходимость переноса единицы из 45-го разряда, то единица переноса теряется, но вырабатывается управляющий сигнал $\omega = 1$, если же такой необходимости не возникало, то вырабатывается сигнал $\omega = 0$.

Точно так же, при вычитании порядков $a, - b = c$, мантисса слова c равна мантиссе a, а порядок c получается вычитанием из порядка a порядка b. Если порядок b не превосходит порядка a, то происходит обычное вычитание и вырабатывается сигнал $\omega = 0$. В противном случае вырабатывается сигнал $\omega = 1$, а вычитание

*) Как и в предыдущем случае, название *сложение порядков* является более распространенным, а *сложение кодовых частей* — более соответствующим существу операции.

происходит так, как будто перед порядком a была приписана единица 46-го разряда.

Рассмотрим на примерах, как выполняются описанные выше фиксированные операции.

Пример 1.28. Пусть

$$a+, b=c,$$

причем $a=001\ 1005\ 2000\ 1500$ и $b=105\ 7776\ 0000\ 0002$. Тогда

$$c=001\ 1003\ 2000\ 1502,$$

и выработается сигнал $\omega=1$. При том же содержимом ячеек a и b по команде $b+, a=c$ получим $c=105\ 1003\ 2000\ 1502$. Для операции $a-, b=c$, если $a=000\ 2735\ 0001\ 5007$ и $b=012\ 2510\ 7776\ 3541$, то $c=000\ 0224\ 0003\ 1246$ и вырабатывается сигнал $\omega=0$. Если же $a=075\ 1344\ 2752\ 0001$ и $b=031\ 5121\ 3174\ 0002$, то $c=075\ 4222\ 7555\ 7777$ и $\omega=1$.

Для операции $a, +b=c$ при $a=154\ 2577\ 3143\ 2614$ и $b=342\ 3217\ 2500\ 7441$ получаем $c=516\ 2577\ 3143\ 2614$ и $\omega=0$, а при том же a и $b=767\ 1243\ 1544\ 6704$ будем иметь $c=143\ 2577\ 3143\ 2614$ и $\omega=1$.

Операции сложения и вычитания мантисс и порядков очень широко используются при программировании в самых разнообразных случаях. Покажем прежде всего их применение при составлении арифметических циклов.

Во всех рассмотренных в § 20 примерах в качестве счетчиков цикла использовались целые двоичные числа с плавающей запятой. Операции вычитания мантисс и порядков дают возможность образовывать *фиксированные* целые счетчики, т. е. использовать в качестве счетчиков мантиссу или порядок слова. Эта возможность тем более полезна, что фиксированные действия происходят заметно быстрее плавающих.

В большинстве случаев максимальное значение счетчика не превышает $4095_{10}=7777_8$. Поэтому обычно в качестве счетчика можно использовать не всю мантиссу слова, а какой-либо один адрес, и помещать в одной ячейке несколько различных счетчиков.

Покажем на примере, как это делается.

Пример 2.28. Вычислить

$$y=\underbrace{x+\sqrt{x+\sqrt{x+\ldots+\sqrt{x}}}}_{n\text{ членов}}.$$

Легко видеть, что величину y можно получить при помощи следующего процесса:

$$y_0=0,$$

$$y_k=x+\sqrt{y_{k-1}};\ k=1, 2, \ldots, n;\ y=y_n.$$

Этот процесс можно запрограммировать как арифметический цикл с рабочей частью, состоящей из двух команд:

$$\sqrt{y} \quad = R,$$
$$x + R = y.$$

В начале цикла следует положить $y = 0$, а затем выполнить написанные две команды n раз.

Цикл удобно организовать с фиксированным счетчиком, с обратным изменением счетчика, в соответствии со схемой, изображенной на рис. 17.

Предположим, что в некоторой ячейке машины содержится слово, в котором в виде фиксированного числа единиц третьего адреса записано число n, а все остальные разряды содержат нуль. Такую ячейку мы будем обозначать через $(0, 0, n)$. Пусть, кроме того, в некоторой ячейке записано слово 000 0000 0000 0001, которое мы будем коротко обозначать через $(0, 0, 1)$. Тогда в качестве начального значения счетчика мы можем выбрать

$$(0, 0, n) —, (0, 0, 1) = \nu,$$

а изменение счетчика вместе с проверкой окончания производить по команде

$$\nu —, (0, 0, 1) = \nu.$$

Тогда арифметический цикл запишется так:

1) $0 + 0 = y$
2) $(0, 0, n) —, (0, 0, 1) = \nu$
3) $\sqrt{y} \quad = R$
4) $x + R = y$
5) $\nu —, (0, 0, 1) = \nu$
6) $У_0$
7) *стоп.*

После того как цикл будет выполнен $n - 1$ раз, содержимое ячейки ν сделается равным нулю. После передачи управления на рабочую часть и вычисления окончательного значения y придется выполнять команду вычитания мантисс, причем мантисса вычитаемого будет превосходить мантиссу уменьшаемого. При этом выработается сигнал $\omega = 1$ и команда $У_0$ передаст управление следующей ячейке, в которой находится команда *стоп*.

С тем же успехом в качестве счетчика можно было взять первый или второй адрес ячейки ν. Например, для использования в качестве счетчика второго адреса нужно иметь ячейки $(0, n, 0)$ и $(0, 1, 0)$ и

заменить команды 2) и 5) следующими командами:

$$(0, n, 0) —, \ (0, 1, 0) = \nu$$

и

$$\nu —, \ (0, 1, 0) = \nu.$$

Нужно только помнить, что число n записано в ячейке в восьмеричной (точнее, в двоично-восьмеричной) системе.

Практика программирования показывает, что счетчики-адреса (*фиксированные счетчики*) удобнее счетчиков-чисел с плавающей запятой (*плавающих счетчиков*), так как вручную легче перевести начальное значение счетчика в целое восьмеричное число, чем в плавающее двоично-восьмеричное.

Если максимальное значение счетчика не превышает $511_{10} = 777_8$, то в качестве счетчика можно использовать вместо мантиссы порядковую часть слова; в этом случае проверку окончания арифметического цикла следует осуществлять при помощи операции вычитания порядков.

Так, в рассмотренном примере цикл можно организовать так:

1) $0 \quad + \quad 0 \quad = y$
2) $(n, 0, 0, 0), — (1, 0, 0, 0) = \nu$
3) $\sqrt{y} \quad = R$
4) $x \quad + \quad R \quad = y$
5) $\nu \quad , — (1, 0, 0, 0) = \nu$
6) $У_0$
7) *стоп.*

В рассмотренном примере операции фиксированного вычитания применялись для проверки окончания арифметического цикла.

Операция вычитания мантисс иногда оказывается полезной и в итерационных циклах. Предположим, что некоторая величина должна быть найдена методом последовательных приближений с машинной точностью. Это значит, что после окончания цикла все цифры мантиссы приближенного значения этой величины должны быть правильными. Проверку выполнения этого условия можно осуществить, как видно из нижеследующего примера, при помощи операции вычитания мантисс.

Пример 3.28. Вычислим число π с машинной точностью.

В отличие от способа, рассмотренного в § 21, будем находить число π, последовательно приближая его полупериметрами как вписанных, так и описанных правильных многоугольников (радиус окружности $= 1$).

Обозначим через $2c_n$ и $2a_n$ длины сторон правильных вписанных и описанных многоугольников с 2^n углами.

Из рис. 28 легко вывести выражения c_n и a_{n+1} через a_n:

$$\left.\begin{aligned} c_n &= \frac{a_n}{\sqrt{1+a_n^2}}, \\ a_{n+1} &= \frac{\sqrt{1+a_n^2}-1}{a_n} = \frac{a_n}{1+\sqrt{1+a_n^2}}. \end{aligned}\right\} \qquad (1.28)$$

Для описанного квадрата имеем

$$a_2 = 1.$$

Формулы (1.28) дают возможность последовательно вычислять длины сторон $2a_n$ и $2c_n$ многоугольников. Их полупериметры равны соответственно $2^n a_n$ и $2^n c_n$. Из определения числа π следует, что $2^n c_n < \pi < 2^n a_n$.

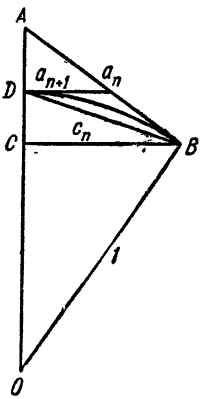


Рис. 28.

Легко убедиться, что при $n > 2$ полупериметры вписанных и описанных многоугольников, а значит и a_n и c_n имеют одинаковые порядки. Поэтому из последнего соотношения вытекает, что значение мантиссы числа π заключено между мантиссами чисел c_n и a_n:

$$M(c_n) \leqslant M(\pi) \leqslant M(a_n). \qquad (2.28)$$

С другой стороны, истинный двоичный порядок π равен 2, а машинный $p'(\pi) = 102_8$. Отсюда следует, что для получения значения π нет необходимости вычислять полупериметры многоугольников; достаточно определять лишь мантиссы величин c_n и a_n.

В то же время из неравенства (2.28) видно, что для получения π с машинной точностью достаточно вести вычисления до тех пор, пока мантиссы a_n и c_n не совпадут. Проверку окончания итерационного цикла можно осуществить при помощи команд

$$c_n —, \ a_n = 0$$

$$У_1 \text{ (начало цикла)}.$$

Действительно, пока $M(c_n) < M(a_n)$, будет вырабатываться сигнал $\omega = 1$. Если же $M(c_n) = M(a_n)$, то $\omega = 0$, и цикл заканчивается. В рабочей части цикла должны вычисляться по формулам (1.28) величины c_n и a_n.

Программа нахождения π с машинной точностью имеет следующий вид:

1)	$0 + \text{«1»} = a_n$	подготовка цикла
2)	$a_n \cdot a_n = R$	рабочая часть цикла
3)	$\text{«1»} + R = R$	
4)	$\sqrt{R} = R$	
5)	$a_n : R = c_n$	
6)	$R + \text{«1»} = R$	
7)	$a_n : R = a_n$	
8)	$c_n -, a_n = 0$	проверка окончания
9)	y_1 2)	
10)	$(102,0,0,0), + a_n = \pi$	
11)	*стоп.*	

По команде 10) этой программы мантисса π получается сложением мантиссы a_n с нулевой мантиссой константы (102, 0, 0, 0), а порядок π образуется из порядка этой константы (при сложении мантисс порядок числа, указанного по второму адресу, опускается).

В рассмотренных ниже примерах операция сложения мантисс используется для выработки сигнала ω, а операция сложения порядков входит в рабочую часть цикла.

Пример 4.28. Подсчитаем число отличных от нуля двоичных разрядов (число единиц) в мантиссе слова a, поместив это число в порядок ячейки v. Очистим сначала счетчик:

$$1)\ 0, + 0 = v.$$

Теперь произведем над словом a операцию

$$2)\ a +, a = a.$$

Тогда мантисса слова удвоится, т. е. сдвинется на один разряд влево, так что место 36-го разряда займет 35-й, место 35-го — 34-й и т. д. Если в 36-м разряде стояла единица, то при выполнении команды 2) появится единица переноса, т. е. сигнал ω будет иметь значение $\omega = 1$. Если же в 36-м разряде стоял нуль, то $\omega = 0$. В первом случае в счетчик числа единиц нужно прибавлять единицу, а во втором случае счетчик надо оставить без изменения.

Для этой цели после команды 2) поставим следующие две команды:

$$3)\ y_0 \quad 5)$$
$$4)\ v, + (1, 0, 0, 0) = v,$$

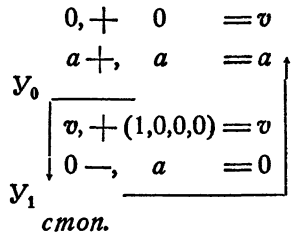
обеспечивающие при повторении рабочей части цикла 1), 2), 3) счет числа единиц в порядке v. Цикл следует кончить тогда, когда ман-

тисса a окажется нулевой, что можно осуществить при помощи двух команд:

5) $0-,\ a=0,$

6) $У_1 \quad 2).$

Таким образом, программа счета единиц в мантиссе слова имеет вид



Пример 5.28. Напишем программу для нормализации двоичного числа x.

В главе IV мы предполагали, что арифметические операции производятся над нормализованными двоичными числами. Эти операции можно производить и над ненормализованными числами, но точность результатов при этом может оказаться худшей *). Поэтому если в результате каких-либо операций в ячейке x оказалось ненормализованное число, перед выполнением арифметических действий его следует нормализовать.

Ненормализованное число x можно представить в виде

$$x=M\cdot 2^p, \tag{3.28}$$

причем

$$|M|<\frac{1}{2}.$$

Пусть

$$M=2^s\cdot N,$$

где

$$\frac{1}{2}\leqslant|N|<1. \tag{4.28}$$

Тогда

$$x=2^{p+s}N;$$

при этом новое значение x будет нормализованным.

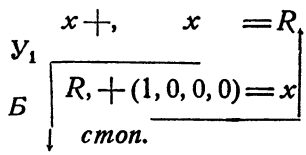
Посмотрим, как осуществить переход от формулы (3.28) к формуле (4.28) в программе.

---

*) Кроме того, операция деления для ненормализованных чисел в машине не всегда осуществима.

Ненормализованная мантисса M отличается от нормализованной наличием нуля в старшем 36-м разряде. Поэтому переход от (3.28) к (4.28) можно осуществлять так: удваивать мантиссу x, прибавляя каждый раз к порядку по единице, пока старший 36-й разряд мантиссы не станет равным 1.

Программа нормализации x имеет вид:



Предоставляем разобраться в этой программе читателю.

Отметим еще один частный прием, в котором используется операция сложения порядков. Предположим, что в слове c мы хотим получить порядок слова b и нулевую мантиссу. Этого можно достичь при помощи команды

$$0\ ,+b=c.$$

Действительно, в нулевой ячейке находится слово с нулевым порядком и нулевой мантиссой. Поэтому мантисса слова c оказывается равной нулю, а порядок — сумме порядков нулевого слова и b, т. е. порядку слова b.

Аналогичным приемом, но уже применяя операцию сложения мантисс, можно выделить в слове b мантиссу, обращая в нуль порядок:

$$0+,\ b=c.$$

В следующих параграфах мы подробно рассмотрим главную область применения операций сложения и вычитания мантисс — переадресацию переменных команд в цикле.

К фиксированным действиям можно также отнести и так называемое *циклическое сложение*, которое мы будем обозначать

$$a \oplus b = c.$$

По этой команде происходит одновременное фиксированное сложение мантисс и фиксированное сложение порядков. Кроме того, если при сложении мантисс образуется единица переноса в старший разряд, то она не пропадает, как при обычном сложении мантисс, а добавляется к младшему разряду мантиссы, т. е. к первому. Аналогично, единица переноса из 45-го разряда при сложении порядков, если она образуется, также не пропадает, а прибавляется к младшему разряду порядка — 37-му. Циклическое сложение используется в программах контрольного суммирования, которые применяются для проверки правильности обмена между различными видами памяти: при вводе с

перфокарт в оперативную память, при записи из оперативной памяти во внешнюю и наоборот и т. п.

Для кодирования нужно знать коды фиксированных операций. Приводим соответствующую таблицу.

Коды фиксированных операций

Название операции	Обозначение	Код
Сложение мантисс	$a+, b=c$	13
Вычитание мантисс	$a-, b=c$	33
Сложение порядков	$a, + \ b=c$	53
Вычитание порядков	$a, - \ b=c$	73
Циклическое сложение	$a \oplus b=c$	07

§ 29. Циклы с переадресацией. Восстановление переменных команд

Рассмотрим задачу, на первый взгляд аналогичную задаче возведения числа x в степень n, рассмотренной в § 20.

Найдем произведение

$$z = x_1 \cdot x_2 \cdot \ \dots \ \cdot x_{100}$$

ста чисел, расположенных в ста последовательных ячейках памяти

$$x_1, \ x_2, \ \dots \ , \ x_{100}.$$

Распишем формулу для z по командам

$$\begin{gathered} 0 + \text{«1»} = z \\ z \ \cdot \ x_1 = z \\ z \ \cdot \ x_2 = z \\ \cdot \ \cdot \ \cdot \ \cdot \ \cdot \ \cdot \\ z \ \cdot \ x_{100} = z \\ \textit{стоп.} \end{gathered}$$

Очевидно, что процесс вычислений здесь имеет циклический характер. Однако, в отличие от задачи вычисления x^{100}, здесь нет много раз повторяющихся одинаковых команд, а есть 100 выполняющихся подряд различных команд, хотя и схожих между собой.

Если бы мы, точно копируя способ образования цикла при вычислении x^{100}, сто раз повторили выполнение команды

$$z \cdot x_1 = z,$$

то вместо произведения $x_1 \cdot x_2 \cdot \ \dots \ \cdot x_{100}$ получили бы степень x_1^{100}.

Для того чтобы правильно образовать цикл в рассматриваемой задаче, мы должны иметь возможность при каждом его повторении изменять команду

$$z \cdot x_1 = z,$$

заменяя в ней последовательно адрес x_1 на адрес x_2, x_2 на x_3, ..., x_{99} на x_{100}, т. е. увеличивая на каждом шаге цикла второй адрес команды на единицу.

В результате такого изменения будут последовательно выполняться операции $z \cdot x_1 = z$, $z \cdot x_2 = z$, ..., $z \cdot x_{100} = z$ и, в конце концов, в ячейке z окажется искомое произведение

$$x_1 \cdot x_2 \cdot \ldots \cdot x_{100}.$$

Важной особенностью электронных счетных машин является возможность изменения команд самой машиной. Такое изменение оказывается необходимым как в рассмотренном простом случае, так и в задачах гораздо более сложных.

Обозначим через (0, 1, 0) слово, у которого в коде, первом и третьем адресах находятся нули, а во втором адресе единица, и через K команду:

$$K: \quad z \cdot x_1 = z.$$

Выполним команду

$$K +, (0, 1, 0) = K.$$

По этой команде к адресам z, x_1, z слова, находящегося в ячейке (K), прибавляются соответственно адреса 0, 1, 0 слова ячейки (0, 1, 0), и результат с кодом ячейки K записывается в ту же ячейку. В результате в ячейке K оказывается уже преобразованная команда

$$z \cdot x_2 = z.$$

Таким образом, происходит увеличение на единицу второго адреса команды K, т. е. ее *переадресация*; слово (0, 1, 0) называют *константой переадресации*.

Возвратимся теперь к задаче, сформулированной в начале параграфа.

Пример 1.29. Составим цикловую программу для вычисления величины

$$z = x_1 \cdot x_2 \cdot \ldots \cdot x_{100}.$$

Проверку окончания цикла будем производить при помощи операции вычитания мантисс.

Так как $100_{10} = 144_8$, то за начальное значение счетчика можно взять слово (0, 0, 144).

Подготовительную и рабочую части цикла запишем в виде

1) (0, 0, 144)—, (0, 0, 1)$=n$
2) 0 $+$ «1» $=z$
3) z $\cdot$ x_1 $=z$.

Для того чтобы при следующем повторении цикла в ячейке последовательно получать произведения $x_1 \cdot x_2$; $x_1 \cdot x_2 \cdot x_3$ и т. д., напишем команду увеличения на единицу второго адреса команды 3):

4) 3)$+$, (0, 1, 0)$=$3).

Теперь нам нужно (условно) передать управление на рабочую часть цикла 3), повторив его 100 раз. Это делается тем же способом, что и ранее:

5) n—, (0, 0, 1)$=n$
6) $У_0$ 3).

Таким образом, цикловая программа для вычисления z имеет следующий вид:

1) (0, 0, 144)—, (0, 0, 1)$=n$
2) 0 $+$ «1» $=z$
3) z $\cdot$ x_1 $=z$
4) 3) $+$, (0, 1, 0)$=$3)
5) n —, (0, 0, 1)$=n$
6) $У_0$
7) *стоп*.

Проследим, как работает составленная программа.

Команда 1) подготавливает счетчик цикла, а 2) заносит в z его первоначальное содержание — единицу. Команда 3) есть рабочая команда цикла. Сначала в ячейке 3) находится команда

3) $z \cdot x_1 = z$,

после выполнения которой в z оказывается величина x_1. Затем по команде переадресации 4) второй адрес команды 3) увеличивается на единицу, так что в ячейке 3) получается команда

3) $z \cdot x_2 = z$.

Затем происходит обычная проверка окончания арифметического цикла (команды 5), 6)) и управление опять переходит в 3). После вторичного выполнения команды 3) в ячейке z оказывается произведение $x_1 \cdot x_2$, затем команда 3) при помощи 4) опять переадресовывается, принимая вид 3) $z \cdot x_3 = z$, после проверки окончания цикла 4), 5) и третьего выполнения команды 3) в z получаем величину $x_1 \cdot x_2 \cdot x_3$.

После того как цикл прокрутится 99 раз, в ячейке z окажется произведение $x_1 \cdot x_2 \cdot \ldots \cdot x_{99}$, команда 3) примет вид

3) $z \cdot x_{100} = z$,

а счетчик n станет равным нулю.

По команде 5) выработается сигнал $\omega = 0$ и команда 6) еще раз передаст управление на команду 3), после выполнения которой в z образуется искомое произведение. Затем команда 5) выработает сигнал $\omega = 1$ и по команде 6) машина выйдет на *стоп*.

Рассмотренная программа является простейшим примером арифметического цикла с переменными командами или *арифметического цикла с переадресацией*.

Команда $z \cdot x_1 = z$ в этой программе меняется с каждым повторением цикла, принимая в конце концов вид

$$z \cdot x_{100} = z.$$

Заметим, что по написанной программе можно произвести расчет величины

$$z = x_1 \cdot x_2 \cdot \ldots \cdot x_{100}$$

только один-единственный раз. Действительно, если мы после однократного вычисления z второй раз обратимся к этой программе, то в ячейке 3) будет находиться команда

$$z \cdot x_{100} = z$$

и программа не сможет правильно работать.

Однако в большинстве задач, решаемых на машинах, один и тот же участок программы должен работать многократно, выполняя одни и те же функции (вычисляя, например, произведение чисел, находящихся в ячейках $x_1, x_2, \ldots, x_{100}$). Поэтому программы должны писаться так, чтобы они были *самовосстанавливающимися* (см. определение на стр. 88). Это особенно важно для циклов, содержащих переменные команды, которые изменяются в процессе работы машины. *Переменные команды всегда должны восстанавливаться.*

В нашем примере мы могли бы достичь своей цели, записав где-либо, например, после команды *стоп*, начальное состояние команды 3) и поместив перед командой 7) команду засылки этого начального состояния в ячейку, отведенную для команды 3). Программа приняла бы тогда вид

$$\begin{array}{lllll}
 & (0,\ 0,\ 144) & - , & (0,\ 0,\ 1) & = n \\
 & 0 & + & \text{«1»} & = z \\
T & z & \cdot & x_1 & = z \\
 & T & + , & (0,\ 1,\ 0) & = T \\
 & n & - , & (0,\ 0,\ 1) & = n \\
y_0 & & & & \\
 & T_0 & + , & 0 & = T \\
 & \multicolumn{4}{c}{стоп} \\
T_0 & z & \cdot & x_1 & = z.
\end{array}$$

Рассмотренный способ называют *конечным восстановлением:* программа восстанавливается к первоначальному виду после ее работы. Однако такое восстановление не обеспечивает возможностей нормальной работы программы во всех случаях.

Дело в том, что при конечном восстановлении переменная команда восстановится только в том случае, если программа дойдет до конца. Если же выполнение программы будет прервано до окончания цикла по каким бы то ни было причинам, например, в результате случайного сбоя машины, то команда не восстановится и пустить ее сначала без нового ввода в память будет невозможно.

Чтобы обеспечить восстановление переменной команды во всех случаях, следует поместить команду $T_0 +, 0 = T$ в начале цикла, в группе команд подготовки цикла. Это есть частный случай общего правила, которым должен руководствоваться программист, имея дело с переменными командами: *восстановление переменной команды должно происходить до ее первого исполнения.*

Общая структура цикла с переадресацией ничем, собственно, не отличается от структуры обычного арифметического цикла. Как и там, чтобы начинать цикл с первого фактически используемого значения счетчика и в качестве эталона проверки брать число повторений цикла, удобно ставить переадресацию и изменение до рабочей части цикла и обходить их при первом выполнении. Программа, написанная по такой схеме, будет иметь такой вид:

```
          0  +    «1»      = z
          0  +, (0, 0, 1)   = n
Б  T0                       T
   |  T  +, (0, 1, 0)   = T
   |  n  +, (0, 0, 1)   = n
T  |  z  ·   x1         = z
   |  n  —, (0, 0, 144) = 0
У1
        стоп
T0    z  ·   x1         = z.
```

Заметим, что команда $z \cdot x_1 = z$ в ячейке T может и не быть написана, потому что при выполнении третьей команды программы туда будет заслана команда T_0. Поэтому на месте команды T можно ничего не писать или писать все, что угодно. На месте такой команды пишут значок н. п., что означает *не перфорировать.* Впрочем, в левой части лучше написать первоначальное состояние команды (в скобках, чтобы показать, что ее не нужно перфорировать. В правой части ставится н. п.).

В приведенном примере можно сократить программу на одну ячейку, если применить изменение счетчика в обратном направлении:

$$
\begin{array}{lll}
 & 0 + \text{«1»} = z & \\
 & (0,\ 0,\ 144) -,\ (0,\ 0,\ 1) = n & \\
 & \longrightarrow & \\
Б & T_0 \quad\quad\quad\quad\quad\quad T & \\
 & \big|\ T + ,\ (0,\ 1,\ 0) = T & \\
\text{Т} & \downarrow (z \cdot x_1 = z) & \text{н. п.} \\
 & n -,\ (0,\ 0,\ 1) = n & \\
 & \text{стоп} & \\
У_0 & & \\
\text{Т}_0 & z \cdot x_1 = z. &
\end{array}
$$

Команду T_0 называют *восстановителем*. Восстановитель лучше помещать в конце программы, после стопа. Однако иногда его помещают перед командой переадресации таким образом, чтобы управление ему не передавалось; в нашем примере его можно было бы поместить между третьей и четвертой командами.

Пример 2.29. В последовательных ячейках памяти находятся числа $x_1, x_2, \ldots, x_{1000}$. Вычислить величины $y_1, y_2, \ldots, y_{1000}$ по формулам

$$
\begin{aligned}
y_1 &= ax_1^2 + bx_1 + c,\\
y_2 &= ax_2^2 + bx_2 + c,\\
&\ldots\ldots\ldots\\
y_{1000} &= ax_{1000}^2 + bx_{1000} + c
\end{aligned}
$$

и поместить в последовательные ячейки памяти. Расписывая эти формулы по командам, получим бесцикловую программу:

$$
\begin{array}{r}
a \cdot x_1 = y_1\\
y_1 + b = y_1\\
x_1 \cdot y_1 = y_1\\
y_1 + c = y_1\\
\hline
a \cdot x_2 = y_2\\
y_2 + b = y_2\\
x_2 \cdot y_2 = y_2\\
y_2 + c = y_2\\
\hline
\ldots\ldots\\
\hline
a \cdot x_{1000} = y_{1000}\\
y_{1000} + b = y_{1000}\\
x_{1000} \cdot y_{1000} = y_{1000}\\
y_{1000} + c = y_{1000}\\
\hline
\text{стоп.}
\end{array}
$$

Такая программа вместе с исходными данными и результатами заняла бы — 6004_{10} ячейки, т. е. не поместилась бы в памяти машины. Ненамного труднее произвести все вычисления вручную, чем выполнить работу по выписыванию всех команд такой бесцикловой программы.

Составим теперь цикловую программу.

Рабочая часть цикла в первоначальном виде состоит из четырех команд:

$$a \cdot x_1 = y_1$$
$$y_1 + b = y_1$$
$$x_1 \cdot y_1 = y_1$$
$$y_1 + c = y_1.$$

Все эти команды, как видно из бесцикловой программы, в дальнейшем должны переадресовываться; в первой из этих команд должны на единицу увеличиваться второй и третий адреса, во второй — первый и третий, в третьей — все три адреса, в четвертой — первый и третий.

Для того чтобы организовать цикл, нужно, кроме команд изменения счетчика, иметь еще четыре восстановителя и четыре команды переадресации. Программу можно сократить, если ввести одну рабочую ячейку и записать рабочую часть в виде

$$a \cdot x_1 = R$$
$$R + b = R$$
$$x_1 \cdot R = y_1$$
$$y_1 + c = y_1.$$

В этом случае вторую команду нет нужды переадресовывать.

Окончательно программу можно написать так:

	(0, 0, 1750)	— ,	(0, 0, 1)	$= n$	
	T_1^0	+ ,	0	$= T_1$	
	T_2^0	+ ,	0	$= T_2$	
				→	
Б	T_3^0		T_1	T_3	
	T_1	+ ,	(0, 1, 0)	$= T_1$	↑
	T_2	+ ,	(1, 0, 1)	$= T_2$	|
	T_3	+ ,	(1, 0, 1)	$= T_3$	|
T_1	$(a$	·	x_1	$= R)$	| н. п.
	R	+	b	$= R$	|
T_2	$(x_1$	·	R	$= y_1)$	| н. п.
T_3	$(y_1$	+	c	$= y_1)$	| н. п.
	n	— ,	(0, 0, 1)	$= n$	|
$У_0$					
	стоп				
T_1^0	a	·	x_1	$= R$	
T_2^0	x_1	·	R	$= y_1$	
T_3^0	y_1	+	c	$= y_1$	

Пример 3.29. Вычислим сто значений $y_1, y_2, \ldots, y_{100}$ функции

$$y=\frac{ax^2+bx+c}{dx^2+ex+f}$$

при $x=x_1, x_2, \ldots, x_{100}$.

Рабочая часть цикла программы для вычисления величин $y_1, y_2, \ldots, y_{100}$ имеет такой вид:

$$\begin{aligned}
a\cdot x_1&=R\\
R+b&=R\\
x_1\cdot R&=R\\
R+c&=R \qquad \text{числитель}\\
d\cdot x_1&=S\\
S+e&=S\\
x_1\cdot S&=S\\
S+f&=S \qquad \text{знаменатель}\\
R:S&=y_1
\end{aligned}$$

и содержит, как легко видеть, пять переменных команд. Поэтому на восстановление, переадресацию и команды-восстановители циклической программы с написанной рабочей частью потребуется 15 ячеек памяти. Однако для этой и подобных программ можно упростить циклы следующим способом.

Введем стандартную ячейку x, в которую будем засылать поочередно значения аргумента $x_1, x_2, \ldots, x_{100}$, вычисляя значение функции по формуле

$$y=\frac{x(ax+b)+c}{x(dx+e)+f}.$$

Полученные в стандартной ячейке y значения функции будем затем последовательно пересылать в ячейки $y_1, y_2, \ldots, y_{100}$.

Тогда цикл будет содержать лишь две переменные команды: засылка в стандартную ячейку

$$0+x_1=x$$

и пересылка из стандартной ячейки

$$0+y=y_1.$$

Программа запишется в виде

	(0, 0, 144)	—	(0, 0, 1)	$=n$	
	X_0	+	0	$=X$	
Б	Y_0			Y	
	X	+	(0, 1, 0)	$=X$	
	Y	+	(0, 0, 1)	$=Y$	
X	(0	+	x_1	$=x)$	н. п.
	a	·	x	$=R$	
	R	+	b	$=R$	
	x	·	R	$=R$	
	R	+	c	$=R$	
	d	·	x	$=S$	
	S	+	e	$=S$	
	S	·	x	$=S$	
	S	+	f	$=S$	
	R	:	S	$=y$	
Y	(0	+	y	$=y_1)$	н. п.
	n	—	(0, 0, 1)	$=n$	
Y_0					
		стоп			
X_0	0	+	x_1	$=x$	
Y_0	0	+	y	$=y_1$	

Заметим, что эту программу можно сократить на одну ячейку, совместив пересылку y в y_1 с условной передачей управления. Разумеется, при этом придется несколько изменить восстановитель. Предоставляем читателю сделать это самостоятельно.

Рассмотренный в этом примере прием засылки в стандартные ячейки и пересылки из стандартных ячеек широко применяется при программировании.

В приведенных примерах переадресации переменных команд цикла команды сложения мантисс работали так, что сложение производилось поадресно. Заметим, что так будет только тогда, когда при поадресном сложении третьих адресов не произойдет перенос во второй адрес, или вторых адресов — в первый адрес (мантиссы складываются не по-

адресно, а как целые числа). Аналогичное замечание относится и к операции вычитания мантисс.

Команды сложения и вычитания мантисс применяются главным образом для переадресации переменных команд цикла. В большинстве случаев массивы для исходных данных и результатов счета цикла расположены в идущих подряд ячейках.

В этих случаях изменение каждого из адресов переменной команды происходит на единицу. Поэтому наиболее часто применяют следующие семь констант переадресации (0, 0, 1); (0, 1, 0); (0, 1, 1); (1, 0, 0); (1, 0, 1); (1, 1, 0); (1, 1, 1).

Эти константы, называемые *стандартными константами переадресации*, целесообразно всегда вводить в память машины, а не присоединять к каждой отдельной программе. Отметим, однако, что даже в случае изменения адресов на единицу не всегда целесообразно применять стандартные константы переадресации. Приведем пример такого рода.

Пример 4.29. В идущих подряд ячейках памяти находятся числа последовательности $a_1, a_2, a_3, \ldots, a_{50}$. Переставить эти числа в обратном порядке, поместив в ячейки $b_1, b_2, b_3, \ldots, b_{50}$. Единственной рабочей переменной командой цикла в этом примере является, очевидно, команда пересылки, которая сначала имеет вид

$$0 + a_1 = b_{50}.$$

Затем эта команда должна последовательно принимать вид

$$\begin{aligned} 0 + a_2 &= b_{49}, \\ 0 + a_3 &= b_{48}, \\ &\cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ 0 + a_{50} &= b_1, \end{aligned}$$

т. е. ко второму адресу должна последовательно прибавляться, а из третьего вычитаться единица. Построим цикл с применением стандартных констант переадресации:

$$
\begin{array}{llllll}
 & (0, & 0, \ 62) & -, & (0,\ 0,\ 1) & = n \\
Б & T_0 & & & & \to T \\
 & T & & +, & (0,\ 1,\ 0) & = T \\
 & T & & -, & (0,\ 0,\ 1) & = T \\
\text{Т} \downarrow & (0 & & + & a_1 & = b_{50}) \quad \text{н. п.} \\
 & n & & -, & (0,\ 0,\ 1) & = n \\
У_0 & & & & & \\
 & & \textit{стоп} & & & \\
\text{Т}_0 & 0 & & + a_1 & & = b_{50}.
\end{array}
$$

Здесь цикл содержит пять команд и вся программа для своего выполнения требует 253 операции. Постараемся теперь подобрать

такую (нестандартную) константу переадресации, чтобы переадресовывать переменную команду цикла в одну операцию, а не в две.

Третий адрес команды T нам нужно уменьшать на 1. Покажем, что это можно сделать, прибавляя к третьему адресу T восьмеричное число 7777_8.

Действительно, пусть, например, $b_{50}=3462$; тогда

$$3462+7777=13461.$$

Таким образом, третий адрес уменьшился на 1 и, кроме того, произошел перенос единицы из третьего адреса во второй. Легко показать, что то же будет иметь место при любом адресе слова b_{50}. Таким образом, если мы к команде $T:(0+a_1=b_{50})$ прибавим константу переадресации (0, 0, 7777), то тем самым мы одновременно уменьшим третий адрес на единицу и увеличим второй адрес на единицу, а как раз это нам и требуется.

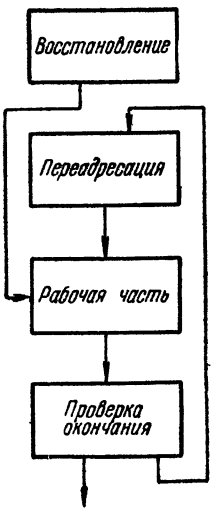


Рис. 29.

Цикл с одной командой переадресации имеет вид:

```
                (0, 0, 62)—, (0, 0, 1)=n
        Б       T0                T      T
                T        +, (0, 0, 7777)=T
        T       (0       +      a1     =b50)  } н. п.
                n        —, (0, 0, 1)  =n
        У0
                стоп
        T0      0        +      a1     =b50.
```

Теперь для решения задачи требуется выполнить 203 операции вместо 253 в первом варианте программы.

Рассмотренный пример показывает, что при помощи операции сложения мантисс можно не только увеличивать, но и уменьшать адреса переменных команд. Делается это специальным подбором констант переадресации.

Часто встречающуюся константу 7777_8 мы будем обозначать буквой F и вместо (0, 0, 7777) писать (0, 0, F).

Можно составить предыдущий цикл, применяя для переадресации одну операцию вычитания мантисс. Для этого следует в качестве константы переадресации использовать слово. (7777, 7777, 1), или, в других обозначениях, (F, F, 1). Рекомендуем читателю самостоятельно убедиться в этом.

Все рассмотренные в этом параграфе программы циклов содержали переменные команды. Присутствие таких команд потребовало

введения в цикл команд переадресации, начального восстановления, специальных констант переадресации и команд-восстановителей.

Как уже было сказано, общая структура цикла с переадресацией принципиально не отличается от структуры обычного арифметического цикла. Блок-схема цикла с переадресацией показана на рис. 29.

§ 30. Двойные циклы с переадресацией

В практике решения задач на электронной машине приходится встречаться с цикловыми программами довольно разнообразного характера. В частности, в некоторых случаях, как мы видели в § 22, цикловая программа оказывается «вложенной» в другую цикловую программу. При этом как внутренний, так и внешний циклы могут быть циклами с переадресацией.

Рассмотрим примеры такого рода.

Пример 1.30. Вычислим девятые степени чисел $x_1, x_2, \ldots, x_{30}$ и поместим в ячейки $y_1, y_2, \ldots, y_{30}$. Мы имеем

$$y_1 = x_1^9,\ y_2 = x_2^9,\ \ldots,\ y_{30} = x_{30}^9.$$

Арифметический цикл для вычисления $y_1 = x_1^9$ «перепишем» из примера 1.20, проверяя, однако, окончание цикла при помощи фиксированного счетчика:

1)	0	+	«1»	$=y_1$
2)	(0, 0, 11)	—,	(0, 0, 1)	$=n$
3)	x_1	·	y_1	$=y_1$
4)	n	—,	(0, 0, 1)	$=n$
5) $У_0$				

Команды 1) — 5) служат для вычисления отдельного значения $y = y_1$ и образуют внутренний цикл. Для того чтобы получить все значения y от y_1 до y_{30}, следует этот внутренний цикл повторить 30 раз, включив его во внешний цикл.

При этом, чтобы при втором прохождении внутреннего цикла считалась величина $y_2 = x_2^9$, нужно переадресовать команды 1), 3), придав им вид

1) $0 + «1» = y_2$

3) $x_2 \cdot y_2 = y_2$.

Переадресацию следует производить во внешнем цикле каждый раз после вычисления очередного значения y. Обозначим фиксированный счетчик внешнего цикла через m и будем изменять его также в обратном направлении.

Программа для вычисления $y_1, y_2, \ldots, y_{30}$ имеет такой вид:

```
                (0, 0, 36)—, (0, 0, 1)=m   }
                  T1^0   +,     0   =T1    } Восстановление внешнего цикла,
Б                 T2^0          T1   T2    }

                  T1     +, (0, 0, 1)=T1   } Переадресация внешнего цикла
                  T2     +, (1, 1, 1)=T2   }
T1              (0       +     «1»   =y1)  } Восстановление внутреннего цикла
                (0, 0, 11)—, (0, 0, 1)=n   }
T2              (x1      ·     y1    =y1)  } Рабочая часть внутреннего цикла
                  n      —, (0, 0, 1)=n    } Изменение и проверка окончания внутреннего цикла
          У0      ------------------------
                  m      —, (0, 0, 1)=m    } Изменение и проверка окончания внешнего цикла
          У0      ------------------------
                  стоп

T1^0            0      +     «1»   =y1   }
T2^0            x1     ·     y1    =y1   } Восстановители
```

Рабочая часть внешнего цикла — внутренний цикл

Пример 2.30. Вычислим значения функции

$$z=\sqrt{ax+by+c}$$

для всех возможных пар значений x и y из следующих последовательностей

$$x=x_1, x_2, \ldots, x_{10},$$

$$y=y_1, y_2, \ldots, y_6.$$

Легко видеть, что нам нужно вычислить всего 60 значений функции z, которые мы будем помещать в 60 последовательных ячейках $z_1, z_2, \ldots, z_{60}$.

Положим сначала $y=y_1$. Тогда нам нужно будет получить десять значений z:

$$z_1=\sqrt{ax_1+by_1+c},$$

$$z_2=\sqrt{ax_2+by_1+c},$$

$$\ldots\ldots\ldots$$

$$z_{10}=\sqrt{ax_{10}+by_1+c}.$$

Эти значения вычисляются при помощи простого цикла с переадресацией:

	(0, 0, 12) — ,		(0, 0, 1) = n			
	U_0	+ ,	0	= U		
Б	W_0		U	W		
	U	+ ,	(0, 1, 0) = U			
	W	+ ,	(0, 0, 1) = W			
U	(a	·	x_1	= R_1)		н. п.
V	b	·	y_1	= R_2		
	R_1	+	R_2	= R_1		
	R_1	+	c	= R_1		
W	($\sqrt{R_1}$			= z_1)		н. п.
	n	— ,	(0, 0, 1) = n			
$У_0$.						

Здесь восстановители U_0 и W_0 должны иметь вид

W_0	$\sqrt{R_1}$	$= z_1$
U_0	$a \cdot x_1$	$= R_1$.

Для того чтобы получить следующую серию из 10 значений z:

$$z_{11} = \sqrt{ax_1 + by_2 + c},$$
$$z_{12} = \sqrt{ax_2 + by_2 + c},$$
$$\cdots\cdots\cdots$$
$$z_{20} = \sqrt{ax_{10} + by_2 + c},$$

следует, очевидно, повторить вычисление по написанному внутреннему циклу, заменив, однако, в нем y_1 на y_2, т. е. увеличивая второй адрес команды V на 1. Аналогично можно получить и все следующие серии по 10 значений z.

Внешний цикл должен состоять из получения шести серий значений z (по числу значений y). В этом цикле должна переадресовываться, а следовательно, и восстанавливаться команда V, поэтому цикл должен содержать команду восстановления V_0 = V и команду переадресации

V + , (0, 1, 0) = V.

Вводя обычную проверку окончания внешнего цикла и объединяя все приведенные команды, получим следующую программу:

	(0, 0, 6)	—,	(0, 0, 1)	$=m$	
	V_0	+,	0	$=V$	
Б	W_0			W	
□	V	+,	(0, 1, 0)	$=V$	
	(0, 0, 12)	—,	(0, 0, 1)	$=n$	
Б	U_0			U	
⊗	U	+,	(0, 0, 1)	$=U$	
U	$(a$	$\cdot$	x_1	$=R_1)$	н. п.
V	$(b$	$\cdot$	y_1	$=R_2)$	н. п.
	R_1	+	R_2	$=R_1$	
	R_1	+	c	$=R_1$	
W	$(\sqrt{R_1}$			$=z_1)$	н. п.
	W	+,	(0, 0, 1)	$=W$	
	n	—,	(0, 0, 1)	$=n$	
y_0			⊗		
	m	—,	(0, 0, 1)	$=m$	
y_0			□		
			стоп		
U_0	a	$\cdot$	x_1	$=R_1$	
V_0	b	$\cdot$	y_1	$=R_2$	
W_0	$\sqrt{R_1}$			$=z_1.$	

Следует обратить внимание на не совсем обычное расположение команд переадресации. Команда V переадресуется только во внешнем цикле, а во внутреннем постоянна. Поэтому она восстанавливается во внешнем цикле и переадресуется во внешнем же цикле, как обычно. Команда U переадресуется во внутреннем цикле и при следующем обращении к внутреннему циклу восстанавливается, так как и для $y=y_2$ мы должны будем вернуться к значению $x=x_1$. Таким образом, команда U восстанавливается во внутреннем цикле и переадресуется там же, тоже как обычно. С командой W дело обстоит иначе. Она переадресуется во внутреннем цикле, но при новом обращении к внутреннему циклу не восстанавливается, а снова должна переадресоваться, так как при переходе к следующему значению y мы снова должны переходить и к следующему значению z. Поэтому восстановление команды W стоит во внешнем цикле, а переадресацию ее

мы вынуждены поставить после исполнения, так как эта переадресация не может обходиться при восстановлении команд внутреннего цикла.

Работу этой программы можно заметно ускорить, если заметить, что второе и третье слагаемые подкоренного выражения не зависят от x, т. е. не меняются во внутреннем цикле. Поэтому две из четырех команд рабочей части можно вынести во внешний цикл, в результате чего работа внутреннего цикла ускорится. Эта программа в закодированном виде приведена в табл. 1.30, а ее памятка — в табл. 2.30.

Команды программы расположены, начиная с ячейки 4000. В памятку внесены обозначения рабочих ячеек R_1, R_2, счетчиков m, n, исходных данных a, b, c, x_1, x_2, ..., x_{10}, y_1, y_2, ..., y_6 и результатов счета z_1, z_2, ..., z_{60}. Внесем в этот бланк также метки U, V, W переадресуемых команд, команд-констант U_0, V_0, W_0 и обозначения начальных значений восьмеричных счетчиков (0, 0, 6), (0, 0, 12).

Команды, которым передается управление, в этой программе нет необходимости снабжать метками, ибо передачи управления обозначены стрелками. Второй адрес команды передачи управления в этом случае кодируется адресом команды, который стоит в правой части в одной строке с концами стрелки. Константа (0, 0, 6) в правой части записывается так:

00 0000 0000 0006

Команды в этой программе кодируются тем же способом, что и в ранее рассмотренных случаях.

Как уже говорилось выше, стандартные константы переадресации вводятся в память машины вместе с любой программой. Обычно их размещают так, чтобы последняя цифра адреса ячейки совпадала с восьмеричной цифрой, которая совпадает с константой переадресации, прочитанной как триада. Например, константа (1, 0, 1), прочитанная как триада, означает цифру 5, поэтому ее помещают в ячейку, адрес которой заканчивается пятеркой. Это облегчает запоминание расположения констант переадресации. Мы будем предполагать, что они размещены в следующих ячейках памяти:

(0, 0, 1)	0121
(0, 1, 0)	0122
(0, 1, 1)	0123
(1, 0, 0)	0124
(1, 0, 1)	0125
(1, 1, 0)	0126
(1, 1, 1)	0127

Поэтому в нашей программе константы (0, 0, 1) и (0, 1, 0) отдельно не выписаны и закодированы соответственно адресами 0121 и 0122.

Таблица 1.30

					4000	A	
	(0, 0, 6) —, (0, 0, 1) = m	4000	0	33	4040	0121	4065
	V_0 +, 0 = V	1	0	13	4023	0000	4004
	Б $\overline{W_0}$ → $\overline{W}$	2	0	56	4024	4004	4013
⊕	V +, (0, 1, 0) = V	3	0	13	4004	0122	4004
V	(b · y_0 = R_0	4				н. п.	
	R_0 + c = R_0	5	0	01	4100	4044	4100
	(0, 0, 12) —, (0, 0, 1) = n	6	0	33	4041	0121	4066
	Б $\overline{U_0}$ → $\overline{U}$	7	0	56	4022	4011	4011
□	U +, (0, 0, 1) = U	4010	0	13	4011	0121	4011
U	(a · x_1 = R_1)	1				н. п.	
	R_0 + R_1 = R_1	2	0	01	4100	4101	4101

					4013	A	
W	($\sqrt{R_1}$ = z_1)	4013				н. п.	
	W +, (0, 0, 1) = W	4	0	13	4013	0121	4013
	n —, (0, 0, 1) = n	5	0	33	4066	0121	4066
	y_0 □	6	0	76	0000	4010	0000
	m —, (0, 0, 1) = m	7	0	33	4065	0121	4065
	y_0 ⊕	4020	0	76	0000	4003	0000
	стоп	1	0	17	0000	0000	0000
U_0	a · x_1 = R_1	2	0	05	4042	4045	4101
V_0	b · y_1 = R_0	3	0	05	4043	4057	4100
W_0	$\sqrt{R_1}$ = z_1	4	0	44	4101	0000	4102

кодировала: перфорировала:

проверила кодировку: проверила карты:

Таблица 2.30

4000			4040 (0, 0, 6)		4100	R_0	4140	z_{31}
4001			4041 (0, 0, 12)		4101	R_1	4141	
4002			4042	a	4102	z_1	4142	
4003			4043	b	4103	z_2	4143	
4004	V		4044	c	4104		4144	
4005			4045	x_1	4105		4145	
4006			4046	x_2	4106		4146	
4007			4047	x_3	4107		4147	
4010			4050	x_4	4110		4150	
4011	U		4051	x_5	4111		4151	z_{40}
4012			4052	x_6	4112		4152	z_{41}
4013	W	Программа	4053	x_7	4113	z_{10}	4153	
4014			4054	x_8	4114	z_{11}	4154	
4015			4055	x_9	4115		4155	
4016			4056	x_{10}	4116		4156	
4017			4057	y_1	4117		4157	
4020			4060	y_2	4120		4160	
4021			4061	y_3	4121		4161	
4022	U_0		4062	y_4	4122		4162	
4023	V_0		4063	y_5	4123		4163	z_{50}
4024	W_0		4064	y_6	4124		4164	z_{51}
4025			4065	m	4125	z_{20}	4165	
4026			4066	n	4126	z_{21}	4166	
4027			4067		4127		4167	

Сделаем еще одно дополнительное замечание. Как уже было указано, восстанавливаемые команды можно не перфорировать либо вводить туда любое содержимое. Очень удобно вводить на место восстанавливаемой команды команду *стоп*. Тогда, если вследствие ошибки программиста переменная команда не восстановится, машина остановится, указав тем самым на необходимость исправить ошибку.

Проще всего условиться, что знак н. п. в правой части означает, что нужно перфорировать

$$017\ 0000\ 0000\ 0000,$$

т. е. команду *стоп*. Обычно так и поступают. Если же это невозможно, то при кодировке в правой части кодируют *стоп*, а знак н. п. оставляют в левой части программы, однако не на месте первоначального состояния команды, а рядом с ним.

§ 31. Индексный регистр (регистр адреса)

Характерной особенностью циклических вычислительных процессов является работа с числовыми последовательностями.

Так, в примере 1.20 вычислялась последовательность целых степеней числа x

$$x^1,\ x^2,\ \ldots,\ x^n;$$

в примере 1.21 находилась последовательность приближенных значений числа π;

в примере 1.29 подсчитывалось произведение членов последовательности

$$x_1,\quad x_2,\ \ldots,\quad x_{100};$$

в примере 1.30 из членов последовательности

$$x_1,\quad x_2,\ \ldots,\quad x_{30}$$

получалась последовательность их девятых степеней:

$$y_1 = x_1^9,$$
$$y_2 = x_2^9,$$
$$\ldots,$$
$$y_{30} = x_{30}^9.$$

Обычно члены последовательности нумеруются целыми числами от 0 до n или от 1 до n, причем номер члена обозначается индексом. Выясним связь между значениями индекса и адресами членов последовательности в памяти.

Пусть нам дана числовая последовательность $x_0, x_1, x_2, \ldots, x_n$. Поместим члены этой последовательности в идущие подряд ячейки

памяти и обозначим адреса этих ячеек через

$$«x_0», \quad «x_1», \quad «x_2», \quad \ldots, \quad «x_n» \text{ *).}$$

Эти адреса связаны со значениями индекса очевидными соотношениями:

$$\begin{aligned} «x_1» &= «x_0» + 1, \\ «x_2» &= «x_0» + 2, \\ &\ldots\ldots\ldots \\ «x_j» &= «x_0» + j, \\ &\ldots\ldots\ldots \\ «x_n» &= «x_0» + n. \end{aligned}$$

Таким образом, адрес произвольного члена последовательности x_j определяется адресом начального члена x_0 и значением индекса j. Из того, что вся последовательность должна умещаться в памяти машины, следует, что j есть двоичное число, меньшее 2^{12}. Таким образом, для *записи любого индекса достаточно одного адреса машинного слова.*

Для облегчения работы с числовыми последовательностями, члены которых занумерованы при помощи индексов, в некоторых машинах имеются *индексные регистры.*

Индексный регистр, называемый также *регистром адреса,* — это устройство, в которое можно поместить любое двоичное число длиной в один адрес (например, индекс).

Для рассматриваемой трехадресной машины регистр адреса состоит из 12 разрядов и является частью устройства управления. Двенадцатиразрядное двоичное число, находящееся в этом регистре, так же как и сам этот регистр, мы будем обозначать буквами PA.

Регистр адреса применяется при программировании в основном для образования адресов членов числовой последовательности. Предположим, что нам нужно образовать адрес $«x_j»$ члена последовательности x_j.

Так как

$$«x_j» = «x_0» + j,$$

то для этого следует в регистр адреса занести значение индекса j, а затем к адресу начального числа последовательности прибавить PA. Для этой операции — добавления PA к адресу машинного слова x_0 — мы введем специальное обозначение:

$$«x_0»^* = «x_0» + PA.$$

---

*) Ранее мы обозначали адрес ячейки, содержащей некоторую величину, той же буквой, что и эту величину. Исключение делалось для ячеек, содержащих известные и никакой буквой в программе не обозначенные числа. Адреса таких ячеек мы обозначали теми же числами, взятыми в кавычках. Например, «1» означало «адрес ячейки, в которой лежит число 1». Так как в данном случае важны не числа $x_0, x_1, \ldots$, а адреса ячеек, в которых они лежат, то мы и здесь пользуемся таким обозначением. Как и для чисел, $«x_0»$ означает адрес ячейки, в которой лежит число x_0.

Очевидно, если $PA=0$, то $«x_0»^*=«x_0»$; если $PA=0001$, то $«x_0»^*=«x_0»+1=«x_1»$; если $PA=j$, то $«x_0»^*=«x_0»+j=«x_j»$.

Таким образом, звездочный адрес $«x_0»^*$ при соответствующем значении PA можно превратить в адрес любого члена последовательности.

Посмотрим, как выполняется команда со звездочным адресом на примере команды

$$K: \quad «1»+x_0^*=a.$$

Пусть перед выполнением этой команды $PA=0003$. Тогда $«x_0»^*=«x_0»+0003=«x_3»$, и команда будет выполняться в виде

$$K^{(1)}: \quad «1»+x_3=a.$$

Как же происходит переадресация (переход от (K) к $(K^{(1)})$) и выполнение команды $(K^{(1)})$?

В памяти машины команда находится в виде (K). В тот момент, когда управление передается на (K), эта команда «переносится» из памяти в устройство управления машины, где и происходит увеличение помеченного звездочкой адреса на значение PA. Таким образом, оставляя команду (K) в памяти неизменной, машина производит переадресацию (изменение адреса) непосредственно в *управлении*.

Прибавление PA к адресам команды мы будем называть *переадресацией по PA*.

Переадресовать по PA можно один, два или все три адреса команды. Так, например, команда

$$x_0^* \cdot x_1^* = c$$

при $PA=0004$ исполняется в устройстве управления в виде

$$x_4 \cdot x_5 = c.$$

Из изложенного видно, что при переадресации по PA нет необходимости в начальном восстановлении переадресуемой команды, ибо эта команда в памяти машины не меняется. Как мы увидим далее, это обстоятельство дает возможность значительно упростить построение циклов с переадресацией.

Рассмотрим теперь вопрос о том, как изображаются в ячейке памяти машины команды, использующие регистр адреса, т. е. содержащие звездочные адреса.

Для указания звездочных адресов в машинном слове, изображающем команду, используются 43, 44 и 45-й разряды. Если звездочным является первый (левый) адрес, то в 45-м (левом) разряде ставится единица, если второй (средний) адрес — звездочный, то единицу ставят в 44-м (среднем) разряде, третьему (правому) звездочному адресу соответствует единица в 43-м (правом) разряде.

Разряды 45, 44 и 43-й обозначаются соответственно через π_1, π_2, π_3 и называются *признаками изменения адресов команды по PA*

(или просто *признаками*). Таким образом, машинное слово, используемое как команда, разбивается на части по схеме, указанной на рис. 30.

При записи команды в ячейке эти три признака объединяются в одну восьмеричную цифру, которая и ставится впереди кода операции. Это и есть та самая цифра, которую мы до сих пор полагали равной нулю.

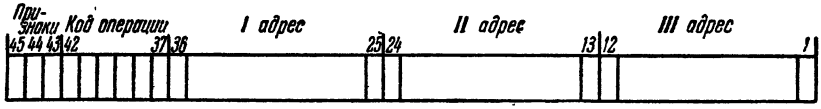


Рис. 30.

Обозначим через A_1, A_2, A_3 адреса команды в памяти машины (*действительные адреса*), а через A_1', A_2', A_3' — адреса той же команды при ее исполнении в устройстве управления (*исполнительные адреса*). Из определения звездочных адресов следует, что

$$A_1' = A_1 + \pi_1 PA,$$
$$A_2' = A_2 + \pi_2 PA,$$
$$A_3' = A_3 + \pi_3 PA.$$

Отсюда видно, что если $\pi_1 = \pi_2 = \pi_3 = 0$ или если $PA = 0$, то исполнительные адреса совпадают с действительными и команда исполняется в том виде, в каком она записана в ячейке памяти. Если же хотя бы один из признаков π_1, π_2, π_3 отличен от нуля и $PA \neq 0$, то команда при исполнении в устройстве управления переадресуется по PA.

При образовании исполнительных адресов по предыдущим формулам может произойти переполнение двенадцатиразрядной сетки соответствующего адреса; в этом случае образовавшийся тринадцатый разряд не переносится в следующий слева адрес, а теряется. Например, если звездочный адрес команды равен 4574, а $PA = 5312$, то соответствующий исполнительный адрес будет равен 2106.

§ 32. Операции с регистром адреса

Переадресация по PA применяется главным образом при образовании циклов с переменными командами. В качестве примера такого цикла рассмотрим следующую простую задачу.

Пусть имеются две последовательности

$$u_0, \quad u_1, \quad u_2, \quad \ldots, \quad u_n$$

и

$$v_0, \quad v_1, \quad v_2, \quad \ldots, \quad v_n$$

и требуется образовать новую последовательность

$$w_0, \quad w_1, \quad w_2, \quad \ldots, \quad w_n,$$

попарно перемножая члены двух исходных,

$$\begin{aligned} w_0 &= u_0 \cdot v_0, \\ w_1 &= u_1 \cdot v_1, \\ w_2 &= u_2 \cdot v_2, \\ &\cdots\cdots \\ w_n &= u_n \cdot v_n. \end{aligned}$$

Произвольный j-й член последовательности определяется формулой

$$w_j = u_j \cdot v_j.$$

Для его вычисления в машине следует занести в регистр адреса значение индекса j, а затем выполнить команду

$$u_0^* \cdot v_0^* = w_0^*.$$

Для того чтобы получить все члены последовательности w_0, w_1, w_2, ..., w_n, следует эту команду включить в цикл, во время работы которого PA должен принимать последовательно значения 0000, 0001, 0002, ... n. Цикл следует кончать тогда, когда PA примет значение n.

Из рассмотренного примера видно, что для образования цикла с переадресацией по PA необходимо иметь возможность:

1) заносить в регистр адреса любое двенадцатиразрядное двоичное число,

2) изменять значение PA,

3) сравнивать PA с заданным 12-разрядным эталоном и, в зависимости от результата сравнения, передавать управление в ту или иную ячейку памяти.

Для работы с регистром адреса в машине предусмотрены специальные команды. Две из них предназначены для первоначального занесения в регистр нужного содержимого; две другие — для сравнения содержимого регистра с эталоном и передачи управления.

Первую из этих команд мы будем обозначать так:

$$PA \quad \bar{b}.$$

При выполнении этой команды машина заносит в регистр адреса фиксированное двенадцатиразрядное двоичное число b, которое записывается во втором адресе команды. Черточку над буквой b мы пишем, чтобы показать, что при кодировке в среднем адресе нужно записать то самое число, которое стоит в содержательной части программы (разумеется, в восьмеричной форме). Первый и третий адреса этой команды будем пока полагать нулевыми. Об их использовании речь будет идти лишь в следующем параграфе.

Например, команда

$$PA \quad 0$$

очищает регистр адреса, а команда

$$PA \quad \overline{3}$$

заносит в регистр адреса число 0003. После выполнения двух команд

$$PA \quad \overline{3}$$
$$u_0^* \cdot v_0^* = w_0^*$$

в регистр адреса будет занесено число 0003, а в ячейку w_3 — произведение $u_3 \cdot v_3$.

Рассматриваемая команда может применяться не только для занесения в PA нужного содержимого, но и для его изменения. Например, команда

$$PA \quad \overline{1}^*$$

будет выполняться так: звездочный второй адрес при поступлении в устройство управления преобразуется в исполнительный адрес, равный

$$0001^* = 0001 + PA,$$

поэтому в регистр адреса занесется старое значение PA, увеличенное на единицу.

Часто бывает необходимо занести в регистр адреса число, неизвестное программисту в момент написания программы, а записанное в некоторой ячейке. Для этой цели используется команда

$$[PA] \quad b.$$

При выполнении этой команды число b, записанное в ее втором адресе, рассматривается как адрес некоторой ячейки и в регистр заносится двенадцатиразрядное двоичное число, содержащееся во втором адресе этой ячейки b.

Например, команда

$$[PA] \quad 0$$

заносит в регистр нуль, так как в нулевой ячейке всегда записано нулевое слово. Команда

$$[PA] \quad b,$$

если в ячейке b записано число (0, 1, 0), занесет в регистр число 0001. Левый и правый адреса этой команды, как и предыдущей, мы будем пока считать нулевыми.

Рассмотренные команды используются главным образом для подготовки цикла, занося в начале его в регистр нужное содержимое. Две другие команды, о которых сейчас будет идти речь, предназначены для сравнения содержимого регистра адреса с эталоном и пере-

дачи управления. Кроме того, они дают возможность одновременно менять содержимое регистра адреса. Эти команды обычно применяются в качестве команд проверки окончания цикла.

Первую из них мы будем записывать в виде

$$PA < a \quad b \quad c,$$

где a и c — целые двенадцатиразрядные двоичные числа, а b — адрес некоторой ячейки памяти. Команда выполняется так: проверяется справедливость написанного слева неравенства, т. е. содержимое PA сравнивается с числом a. Если неравенство $PA < a$ справедливо, то управление передается ячейке b. Если, наоборот, $PA \geqslant a$, то управление передается следующей ячейке. Одновременно с передачей управления в регистр адреса заносится число c, стоящее в третьем адресе выполняемой команды.

Как правило, эта команда используется несколько иначе, а именно, в виде

$$PA < a \quad b \quad c^*.$$

При ее выполнении третий звездочный адрес переадресуется в управлении машины на прежнее содержимое PA, поэтому после выполнения команды в регистр адреса запишется число $c^* = c + PA$, т. е. *содержимое регистра адреса увеличится на c двоичных единиц.*

Вторая команда для работы с регистром адреса имеет вид

$$PA \geqslant a \quad b \quad c$$

и выполняется точно так же, как и предыдущая, с той только разницей, что проверяется справедливость другого неравенства. Управление передается ячейке b, если $PA \geqslant a$, и следующей ячейке, если $PA < a$. Как и предыдущая команда, она чаще всего используется в виде

$$PA \geqslant a \quad b \quad c^*,$$

так что после ее выполнения к содержимому регистра адреса прибавляется число c.

Учитывая, что при переполнении разрядной сетки регистра адреса образовавшийся тринадцатый разряд теряется, можно подобрать число c так, чтобы содержимое регистра не увеличивалось, а у м е н ь ш а л о с ь. Например, чтобы при проверке окончания содержимое регистра адреса уменьшилось на две единицы, достаточно выполнить команду

$$PA \geqslant a \quad b \quad \overline{7776}^*.$$

Читатель легко убедится в справедливости этого утверждения.

Коды операций с регистром адреса, которые необходимо знать для кодировки программ, приводятся в следующей таблице. Заметим,

что команды занесения в регистр мы пишем здесь со всеми тремя адресами, хотя левый и правый адреса будем пока считать нулевыми.

Коды операций с регистром адреса

Обозначение команды	Код
$PA\ \bar{a}\ \bar{b}\ c$	52
$[PA]\ \bar{a}\ b\ c$	72
$PA < \bar{a}\ b\ \bar{c}$	12
$PA \geqslant \bar{a}\ b\ \bar{c}$	32

Рассмотрим теперь примеры применения регистра адреса при программировании циклов. Самым простым является применение регистра адреса для арифметических циклов с искусственными счетчиками.

Пример 1.32. Рассмотрим цикловую программу для нахождения $z = x^{100}$, которую мы уже разбирали (см. пример 1.20). Используя регистр адреса в качестве счетчика, мы можем написать такую программу:

1) PA 0
2) 0 $+$ «1» $= z$
3) z $\cdot$ x $= z$
4) $PA < \overline{143}$ $\overline{1^*}$
5) *стоп.*

Этот цикл не содержит переменных команд, и все адреса команд, кроме правого адреса команды 4), не являются звездочными, т. е. выполняются точно так, как написаны. Стрелка, проведенная из среднего адреса команды 4) вверх, показывает, что при выполнении условия $PA < 143$ управление передается команде 3).

Программа работает так: команды 1) и 2) являются командами подготовки цикла. Команда 1) очищает регистр адреса, а команда 2) заносит в ячейку z первоначальное содержимое. Команда 3) составляет рабочую часть цикла. После ее первого выполнения в ячейке z будет находиться x, а $PA = 0$. Неравенство, написанное слева в команде 4), будет выполнено, и команда 4) передаст управление команде 3). Одновременно содержимое регистра адреса увеличится на 1, т. е. станет равно $PA = 0001$.

Каждый раз при выполнении команды 4) к содержимому PA будет прибавляться единица. Таким образом, команда 3) будет выполняться при $PA = 0, 1, 2, \ldots, 143$, т. е. $144_8 = 100_{10}$ раз. При $PA = 143$ неравенство, написанное в команде 4), не выполнится, и команда 4)

передаст управление уже не команде 3), а команде 5), которая и остановит машину. Заметим, между прочим, что прибавление единицы в регистр все равно произойдет, так что после окончания цикла содержимое регистра адреса будет $PA = 0144$.

Использование регистра адреса оказалось здесь очень удобным по ряду причин. Во-первых, команда 4) одновременно выполняет три обязанности, для которых раньше приходилось расходовать две или три команды: сравнение счетчика с эталоном, изменение счетчика и передачу управления рабочей части. Во-вторых, отпала необходимость иметь специальные ячейки для хранения эталона и константы изменения счетчика, так как эти числа пишутся в самой команде.

В следующих примерах регистр адреса используется уже не только как счетчик, но и для переадресации команд.

Пример 2.32. Перемножить две последовательности

$$u_0, \quad u_1, \quad u_2, \ \ldots, \quad u_n$$

и

$$v_0, \quad v_1, \quad v_2, \ \ldots, \quad v_n,$$

члены которых расположены в идущих подряд ячейках памяти. Полученную последовательность поместить в идущие подряд ячейки

$$w_0, \quad w_1, \quad \ldots, \quad w_n.$$

Для решения задачи требуется выполнить последовательно $n+1$ команд умножения:

$$u_0 \cdot v_0 = w_0,$$
$$u_1 \cdot v_1 = w_1,$$
$$\ldots\ldots$$
$$u_n \cdot v_n = w_n.$$

Каждая следующая из этих команд может быть получена из предыдущей увеличением всех трех индексов на единицу. Поэтому для построения цикла в этом примере следует сначала выполнить две команды:

$$PA \quad 0,$$
$$u_0^* \cdot v_0^* = w_0^*,$$

а затем n раз*) повторить выполнение второй из этих команд, увеличивая каждый раз PA на единицу.

Осуществляется это при помощи следующей программы:

1) $PA \quad 0$
2) $\quad u_0^* \cdot v_0^* = w_0^*$
3) $PA < \bar{n} \quad$ | 2 |
4) *стоп.*

*) Здесь n предполагается целым восьмеричным числом.

Проследим, как работает эта программа. Сначала по команде 1) очищается регистр адреса. Так как $PA = 0$, то

$$«u_0»^* = «u_0», \quad «v_0»^* = «v_0», \quad «w_0»^* = «w_0»$$

и команда 2) исполняется в виде $u_0 \cdot v_0 = w_0$. Сравнение нулевого значения PA с первым адресом команды 3) приводит к передаче управления на 2), при этом одновременно в регистр адреса заносится третий исполнительный адрес команды 3):

$$\bar{1}^* = 0000 + 0001 = 0001.$$

Таким образом, на следующем шаге цикла $PA = 0001$ и в команде 2):

$$«u_0»^* = «u_0» + 1, \quad «v_0»^* = «v_0» + 1, \quad «w_0»^* = «w_0» + 1.$$

Поэтому команда 2), не изменяясь в памяти, фактически исполняется в виде

$$u_1 \cdot v_1 = w_1.$$

Проследив далее выполнение программы, легко убедиться в том, что на каждом следующем шаге цикла PA будет увеличиваться на единицу, а команда 2) будет последовательно исполняться в виде

$$u_2 \cdot v_2 = w_2,$$
$$u_3 \cdot v_3 = w_3,$$
$$\cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot$$

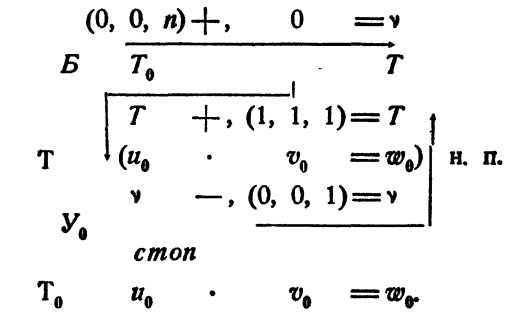
При этом цикл будет работать до тех пор, пока выполняется условие: $PA < n$. Рассмотрим тот этап цикла, когда перед выполнением команды 3) имеем $PA = n - 1$. Тогда команда 3) передает управление на 2), а к PA прибавляется единица. При $PA = n$ команда 2) исполняется в виде

$$u_n \cdot v_n = w_n,$$

т. е. образуется последний член нужной последовательности. Так как $PA = n$, то выполняемая после 2) команда 3) передает управление на *стоп*, т. е. цикл закончится. Как и в предыдущем примере, к моменту окончания цикла PA оказывается равным $n + 1$, ибо команда 3) прибавит при последнем исполнении единицу к прежнему значению PA, равному n.

Рассматриваемую задачу можно запрограммировать, используя вместо регистра адреса переадресацию переменной команды: $u_0 \cdot v_0 = w_0$ при помощи константы (1, 1, 1). Самовосстанавливающаяся программа,

составленная таким способом, имеет такой вид:



Эта программа в два раза длиннее предыдущей и, кроме того, работает в два раза медленнее. Из рассмотренного примера видно, что применение регистра адреса для переадресации переменных команд в цикле позволяет значительно сократить программу и ускорить ее работу.

Это достигается благодаря следующим обстоятельствам:

а) переменная команда цикла в памяти остается неизменной, а переадресуется в устройстве управления, поэтому оказывается излишним начальное восстановление;

б) переадресация производится при помощи *РА* и, следовательно, устраняется необходимость держать в памяти константы переадресации и вводить в программу команды переадресации;

в) проверка окончания цикла происходит по значению *РА*, играющего роль счетчика цикла, поэтому нет нужды вводить специальный счетчик.

Пример 3.32. Найти силу F, с которой положительный заряд отталкивается девятью положительными зарядами $q_1, q_2, \ldots, q_9$, находящимися на луче, выходящем из заряда q, на расстояниях $r_1, r_2, \ldots, r_9$ от него.

По закону Кулона заряд q_j отталкивает заряд q с силой

$$F_j = k\frac{qq_j}{r_j^2}.$$

Поэтому равнодействующая всех сил отталкивания определяется формулой

$$F = kq\left(\frac{q_1}{r_1^2} + \frac{q_2}{r_2^2} + \ldots + \frac{q_9}{r_9^2}\right),$$

или

$$F = kq\,\Sigma,$$

где

$$\Sigma = \frac{q_1}{r_1^2} + \frac{q_2}{r_2^2} + \ldots + \frac{q_9}{r_9^2}.$$

Очевидно, Σ можно получить при помощи следующих формул:

$$\Sigma_0 = 0,$$

$$\Sigma_j = \Sigma_{j-1} + \frac{q_j}{r_j^2} \qquad (j = 1, 2, \ldots, 9),$$

$$\Sigma = \Sigma_9.$$

Приступая к составлению программы, очистим сначала ячейку Σ:

$$0 + 0 = \Sigma.$$

Для того чтобы из величины Σ_{j-1}, находящейся в ячейке Σ, получить Σ_j, достаточно выполнить три команды:

$$q_j : r_j = f$$
$$f : r_j = f$$
$$\Sigma + f = \Sigma.$$

Для нахождения Σ следует эти команды включить в цикл, в котором j должно меняться от 1 до 9. Заменим в этих командах адрес q_j на q_1^* и адрес r_j на r_1^*:

$$q_1^* : r_1^* = f$$
$$f : r_1^* = f$$
$$\Sigma + f = \Sigma.$$

Изменяя PA от 0001 до 0010, мы получим цикл нахождения Σ:

$$\begin{array}{lll} 0 & +0 & =\Sigma \\ PA & 0 & \\ q_1^* & : r_1^* & = f \\ f & : r_1^* & = f \\ \Sigma & + f & = \Sigma \\ PA < \overline{10} & & \overline{1^*} \end{array}$$

Присоединим к этим командам две команды получения F из Σ и *стоп*:

$$q \cdot \Sigma = F$$
$$k \cdot F = F$$
стоп.

Заметим еще, что команду очистки ячейки Σ можно поместить после очистки регистра адреса.

Перепишем эту программу в левую часть программного бланка и закодируем (табл. 1.32).

Таблица 1.32

					4200	A	
PA 0 0 0	4200	0	·52	0000	0000	0000	
0 + 0 = Σ	1	0	01	0000	0000	4300	
q_1^* : $r_1^*=f$	2	6	04	4240	4251	4301	
f : $r_1^*=f$	3	2	04	4301	4251	4301	
Σ + f = Σ	4	0	01	4300	4301	4300	
$PA<\overline{10}$ | $\overline{1}^*$	5	1	12	0010	4202	0001	
q · Σ = F	6	0	05	4220	4300	4302	
k · $F=F$	7	0	05	4221	4302	4302	
стоп	4210	0	17	0000	0000	0000	

Программу, исходные данные, результат F и рабочие ячейки распределим в памяти согласно памятке (табл. 2.32). В переадресуемой по *PA* команде 4202: $\pi_1=1$, $\pi_2=1$, $\pi_3=0$. Объединяя три признака π_1, π_2, π_3 в триаду π и переводя ее затем в одну восьмеричную цифру, получим $\pi=110_2=6_8$. Это значение π записываем в специально отведенную для признаков графу правой части бланка. Аналогично кодируются признаки в команде 4203 ($\pi_1=0$, $\pi_2=1$, $\pi_3=0$; $\pi=010_2=2_8$) и в команде 4205 ($\pi_1=0$, $\pi_2=0$, $\pi_3=1$; $\pi=001_2=1_8$).

Пример 4.32. Вычислить сумму пятнадцати комплексных чисел

$$z_1, z_2, \ldots, z_{15}.$$

Поместим действительную часть числа z_1 в ячейку u_1, мнимую в следующую ячейку v_1, действительную и мнимую части числа z_2 в следующие две ячейки u_2, v_2 и т. д. Тогда числа последовательности u_1, u_2, ..., u_{15} будут расположены в ячейках памяти через одну, так что

$$«u_j» = «u_1» + 2(j-1) \qquad (j=1, 2, \ldots, 15). \qquad (1.32)$$

Аналогично для последовательности v_1, v_2, ..., v_{15}

$$«v_j» = «v_1» + 2(j-1). \qquad (2.32)$$

Действительную и мнимую части суммы

$$z = z_1 + z_2 + \ldots + z_{15}$$

Таблица 2.32

4200	Программа	4240	q_1	4300	Σ
4201		4241	•	4301	f
4202		4242	•	4302	F
4203		4243		4303	
4204		4244	•	4304	
4205		4245	•	4305	
4206		4246	•	4306	
4207		4247	•	4307	
4210		4250	q_9	4310	
4211		4251	r_1	4311	
4212		4252	•	4312	
4213		4253	•	4313	
4214		4254	•	4314	
4215		4255	•	4315	
4216		4256	•	4316	
4217		4257	•	4317	
4220	q	4260	•	4320	
4221	k	4261	r_9	4321	
4222		4262		4322	

мы расположим в ячейках u и v. Тогда

$$\left.\begin{aligned} u &= u_1 + u_2 + \ldots + u_{15}, \\ v &= v_1 + v_2 + \ldots + v_{15}. \end{aligned}\right\} \qquad (3.32)$$

Приступая к составлению программы, очистим ячейки u и v:

1) $0 + 0 = u$,

2) $0 + 0 = v$.

Для получения u и v по формулам (3.32) следует в цикле для $j=1, 2, 3, \ldots, 15$ выполнить команды

$$u+u_j=u,$$
$$v+v_j=v.$$

На первом шаге цикла имеем

$$u+u_1=u,$$
$$v+v_1=v.$$

Будем переадресовывать эти команды при помощи регистра адреса:

3) $PA \quad 0$
4) $u+u_1^*=u$
5) $v+v_1^*=v.$

Согласно (1.32) и (2.32) в рассматриваемом примере, в отличие от двух предыдущих, PA на каждом шаге цикла следует увеличивать не на 0001, а на 0002. Кончать цикл нужно тогда, когда выполнится равенство

$$«u_1»^* = «u_{15}»,$$

а это будет при $PA=2\cdot 14_{10}=2\cdot 16_8=34_8$. Поэтому окончание цикла и изменение PA следует осуществлять так:

6) $PA<\overline{34} \quad 4) \quad \bar{2}^*.$

Таким образом, программа получения z имеет вид

$PA \quad 0$
$0+0=u$
$0+0=v$
$u+u_1^*=u$
$v+v_1^*=v$
$PA<\overline{34} \quad \overline{\bar{2}^*}$
стоп.

Рассмотренный пример характерен тем, что члены обрабатываемых последовательностей расположены в памяти не подряд (с шагом 1), а через одну ячейку (с шагом 2).

Пусть в цикле следует «переработать» все члены некоторой последовательности $a_1, a_2, \ldots, a_n$, причем в памяти эта последовательность расположена с шагом Δ. Цикл для такой последовательности следует организовывать по схеме

$PA \quad 0$
Рабочая часть
$PA<\Delta\overline{(n-1)} \quad \overline{\Delta^*}.$

В трех рассмотренных примерах элементы последовательностей обрабатывались в порядке возрастания индексов, начиная с самого первого члена до последнего.

В некоторых задачах оказывается удобнее перебирать члены последовательности, начиная с конца. Если применять в этом случае для переадресации регистр адреса, то придется уменьшать адреса членов последовательности, т. е. в цикле уменьшать *РА*.

Пример 5.32. В последовательности комплексных чисел

$$z_1,\ z_2,\ \ldots,\ z_{25}$$

найти последнее отличное от нуля действительное число w и разделить все члены последовательности на это число. Задача здесь распадается на два этапа:

I) нахождение числа w,

II) образование последовательности:

$$\zeta_1=\frac{z_1}{w},\quad \zeta_2=\frac{z_2}{w},\quad \ldots,\quad \zeta_{25}=\frac{z_{25}}{w}.$$

Будем находить w, перебирая члены последовательности с конца. Пусть

$$z_1=u_1+iv_1,\quad z_2=u_2+iv_2,\quad \ldots,\quad z_{25}=u_{25}+iv_{25},$$

и ячейки памяти заняты в следующем порядке:

$$u_1,\ v_1,\ u_2,\ v_2,\ \ldots,\ u_{25},\ v_{25}.$$

Проверку условия: является ли число z_j действительным и отличным от нуля, можно осуществить при помощи команд:

1)′	$\lvert 0\rvert-\lvert v_j\rvert=0$	
2)′	$У_1$	7)′
3)′	$\lvert 0\rvert-\lvert u_j\rvert=0$	
4)′	$У_0$	7)′
5)′	$0+u_j=w$	
6)′	$Б$	II этап.

Эти команды следует включить в цикл, выполняя их для $j=25$, 24, 23, ... до тех пор, пока мы не выйдем из цикла по команде безусловной передачи управления. Для того чтобы производить переадресацию по *РА*, перепишем эти команды в виде:

1)″	$\lvert 0\rvert-\lvert v_1^*\rvert=0$	
2)″	$У_1$	7)″
3)″	$\lvert 0\rvert-\lvert u_1^*\rvert=0$	
4)″	$У_0$	7)″
5)″	$0+u_1^*=w$	
6)″	$Б$	II этап.

На первом шаге цикла «v_1»* должно равняться «v_{25}»:

$$«v_1»^* = «v_{25}» = «v_1» + 2 \cdot 24_{10},$$

т. е. $PA = 2 \cdot 24_{10} = 48_{10} = 60_8$. Таким образом, перед началом цикла следует поставить команду

$$PA \quad \overline{60}.$$

На втором шаге цикла должно быть

$$«v_1»^* = «v_{24}» = «v_1» + 2 \cdot 23_{10},$$

т. е. PA должно уменьшиться на 2. Такое уменьшение PA и передачу управления на начало цикла можно осуществить при помощи команды

$$7)'' \quad PA \geqslant 0 \quad 1)'' \quad \overline{7776}.$$

Объединяя все выписанные команды и совмещая пересылку 5)' с передачей управления 6)', получим программу вычисления числа w:

0)	PA	$\overline{60}$
1)		$\lvert 0 \rvert - \lvert v_1 \rvert^* = u$
2)	$У_1$	6)
3)		$\lvert 0 \rvert - \lvert u_1 \rvert^* = 0$
4)	$У_0$	6)
5)	$Б$	$\overline{u_1^* \quad \text{II}) \quad \overrightarrow{w}}$
6)	$PA \geqslant 0$	1) $\overline{7776}^*$.

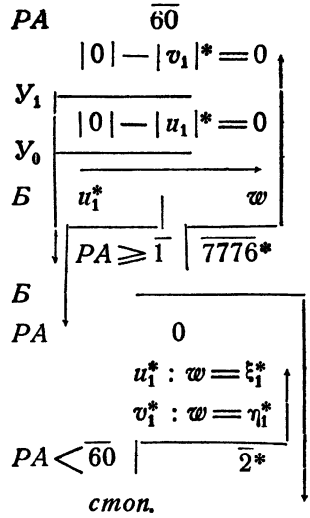
Написанная программа может зациклиться, если среди чисел последовательности нет действительных чисел, отличных от нуля. Действительно, команда 6) дает безусловную передачу управления, а выход из цикла происходит только по команде 5), когда будет обнаружено отличное от нуля действительное число. Чтобы избежать зацикливания, изменим команду 6) таким образом, чтобы выход из цикла был обеспечен после окончания перебора всех членов последовательности, после чего можно поместить безусловную передачу управления на *стоп*. Для этой цели можно воспользоваться командами

6)	$PA \geqslant \overline{2}$	1) $\overline{7776}^*$
7)	$Б$	*стоп*.

В самом деле, после того как цикл пройдет 24_{10} раза, после выполнения команды 6) мы получим $PA = 0$, так что при следующем переходе к той же команде управление перейдет команде 7).

Второй этап решения задачи, нахождение последовательности $(\zeta_1, \zeta_2, \ldots, \zeta_{25})$, программируется тем же способом, что и в предыдущем примере.

Программа решения всей задачи имеет следующий вид:



Здесь через ξ_1, η_1, ξ_2, η_2, ..., ξ_{25}, η_{25} обозначены адреса последовательных ячеек памяти, содержащих действительные и мнимые части искомых комплексных чисел ζ_1, ζ_2, ..., ζ_{25}.

§ 33. Применение регистра адреса в сложных циклах

В двойных циклах, которые рассматривались в § 22, приходилось устраивать два счетчика, отдельно для внутреннего и внешнего циклов. Если использовать в качестве счетчика цикла регистр адреса, как это делалось в предыдущем параграфе, то необходимо иметь возможность при переходе к внутреннему циклу запоминать соответствующее значение регистра и восстанавливать его при обращении к командам внешнего цикла, т. е. после выхода из внутреннего. Этой цели и служит использование крайних адресов команд занесения в регистр.

Как уже было сказано, команды занесения в регистр являются трехадресными. Команда

$$РА \quad \overline{a} \quad \overline{b} \quad c$$

выполняется так: в регистр адреса заносится число b. Одновременно в ячейку с адресом c записывается команда занесения в регистр числа a, т. е. команда

$$РА \quad \tilde{a}.$$

Обычно команда занесения в регистр используется в виде

$$РА \quad 0^* \quad 0 \quad c.$$

Первый адрес команды является звездочным. Исполнительный адрес формируется еще при старом значении регистра. Поэтому в ячейку c запишется команда

$$PA \quad 0 \quad PA_{\text{старое}} \quad 0,$$

а регистр очистится (если в среднем адресе написать любое число b, то оно занесется в регистр). Когда управление придет в ячейку c, то находящаяся там команда восстановит прежнее содержимое регистра адреса.

Точно так же используются первый и третий адреса и в команде

$$[PA] \quad \bar{a} \quad b \quad c:$$

в регистр адреса заносится второй адрес ячейки b, а в ячейку c — команда

$$PA \quad \bar{a}.$$

Как и предыдущую, эту команду чаще всего используют в виде

$$[PA] \quad 0^* \quad b \quad c.$$

Рассмотрим теперь некоторые примеры.

Пример 1.33. На прямой расположено 20 частиц с зарядами

$$q_1, q_2, \ldots, q_{20}.$$

Найдем силы, действующие на каждую из них.

Введем на прямой систему координат и обозначим через $x_1, x_2, \ldots, x_{20}$ координаты зарядов $q_1, q_2, \ldots, q_{20}$. Тогда расстояние между двумя частицами q_i и q_j будет равно

$$r_{ij} = |x_i - x_j|.$$

Абсолютная величина силы взаимодействия между ними определяется формулой

$$|F_{ij}| = \frac{kq_i q_j}{r_{ij}^2} = \frac{kq_i q_j}{(x_i - x_j)^2}.$$

Будем считать силу положительной, если ее направление совпадает с положительным направлением оси Ox. Сила, с которой заряд q_j действует на заряд q_i, равна

$$F_{ij} = \frac{kq_i q_j}{(x_i - x_j)\,|x_i - x_j|}.$$

Результирующая сила действия всех зарядов $q_1, q_2, \ldots, q_{i-1}, q_{i+1}, \ldots, q_{20}$, на заряд q_i имеет вид суммы

$$F_i = kq_i \sum_{j=1}^{20}{}' \frac{q_j}{(x_i - x_j)\,|x_j - x_i|}, \tag{1.33}$$

где штрих означает, что при суммировании следует опустить член, соответствующий $j = i$, т. е. $x_j = x_i$.

Для решения задачи нужно найти величины всех результирующих сил $F_1, F_2, \ldots, F_{20}$. Составим сначала программу нахождения суммы $\sum$ по формуле (1.33), предполагая, что координата x_i заслана предварительно в определенную ячейку x.

На j-м шаге цикла нахождения Σ нам нужно:
1) проверить выполнение условия $x = x_j$,
2) если оно не выполнено, прибавить к Σ величину

$$\frac{q_j}{(x_i - x_j)\,|x_j - x_i|},$$

3) если оно выполнено, оставить Σ без изменения.
Эти действия реализуются при помощи следующих команд:

$$\begin{array}{rl} & x_j - x = \rho \\ & |0| - |\rho| = R \\ У_0 & \\ & \rho \cdot R = R \\ & q_j : R = R \\ & \Sigma + R = R \end{array}.$$

Заменяя здесь переменные адреса x_j, q_j звездочными x_1^*, q_1^* и включая написанные команды в цикл с переадресацией по PA, получим программу нахождения Σ:

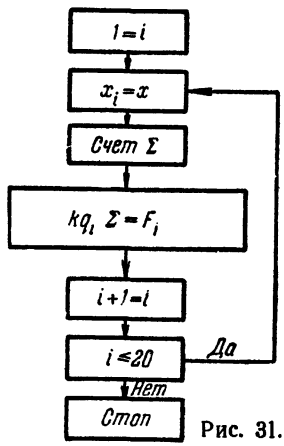


Рис. 31.

1)	PA	0	
2)		$0 + 0 = \Sigma$	
3)		$x_1^* - x = \rho$	
4)		$\|0\| - \|\rho\| = R$	
5)	$У_0$	9)	
6)		$\rho \cdot R = R$	
7)		$q_1^* : R = R$	
8)		$\Sigma + R = \Sigma$	
9)	$PA < \overline{23}$	3) $\overline{1}^*$.	

Теперь этот внутренний цикл нужно включить во внешний цикл получения F_1, F_2, ..., F_{20}. Его можно организовать по схеме, показанной на рис. 31. Внешний цикл по индексу i также организуем при помощи регистра адреса

$$\begin{array}{rl} PA & 0 \\ & 0 + x_1^* = x \\ & \boxed{\text{счет } \Sigma} \\ & k \cdot \Sigma = R \\ & q_1^* \cdot R = F_1^* \\ PA < \overline{23} & \overline{1}^*. \end{array}$$

Однако в таком виде программа будет работать неверно. Действительно, во внешнем цикле PA должен меняться от 0000 до 0023, увеличиваясь на каждом шаге цикла на 0001. Однако каждый раз при входе во внутренний цикл PA очищается (см. команду 1)), а при выходе из внутреннего цикла PA становится равным 0024 (см. команду 9)).

Поэтому для правильной работы внешнего цикла следует при входе во внутренний цикл запоминать значение PA для внешнего цикла и восстанавливать его после выхода из внутреннего. Для этого заменим команду 1) командой

$$PA\ 0^*\ 0\ BP,$$

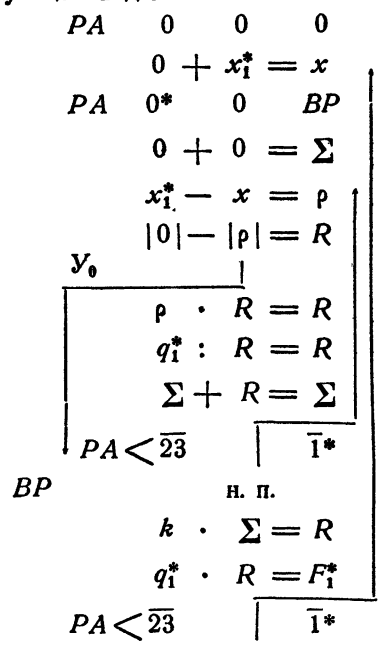
где BP — адрес ячейки, которую следует оставить пустой после команды 9) внутреннего цикла.

Написанная команда, очищая PA для начала работы внутреннего цикла получения $\sum$, одновременно заготавливает в ячейке BP команду

$$PA \quad PA_{\text{старое}},$$

при помощи которой после выхода из внутреннего цикла восстанавливается значение PA для внешнего цикла.

Таким образом, программа нахождения сил взаимодействия зарядов записывается в следующем виде:



Пример 2.33. Дана таблица из семи строк и пяти столбцов:

$$\begin{array}{llll} a_{11} & a_{12} & \ldots & a_{15}, \\ a_{21} & a_{22} & \ldots & a_{25}, \\ \ldots & \ldots & \ldots & \ldots \\ a_{71} & a_{72} & \ldots & a_{75}, \end{array}$$

расположенная в памяти по строкам, т. е. $a_{11}, a_{12}, \ldots, a_{15}, a_{21}, \ldots, a_{25}, \ldots, a_{71}, \ldots, a_{75}$. Вычислим суммы элементов столбцов этой таблицы:

$$\begin{array}{l} S_1 = a_{11} + a_{21} + \ldots + a_{71}, \\ S_2 = a_{12} + a_{22} + \ldots + a_{72}, \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ S_5 = a_{15} + a_{25} + \ldots + a_{75}. \end{array}$$

Составим прежде всего цикл нахождения S_1:

1) PA 0
2) $0 + 0 = S$
3) $S + a_{11}^* = S$
4) $PA < \overline{36}$ $\overline{5}^*$
5) $0 + S = S_1$.

Здесь на каждом шагу цикла регистр увеличивается на 5, ибо элементы $a_{11}, a_{21}, \ldots, a_{71}$ расположены в памяти с шагом, равным 5. Цикл оканчивается, когда PA достигает значения

$$\Delta(n-1) = (5 \cdot 6)_{10} = 36_8.$$

Для того чтобы найти суммы S_2, S_3, S_4, S_5, выписанные команды 1)—5) внутреннего цикла следует включить во внешний цикл. Для получения S_2 следует, очевидно, а) заменить в команде 3) второй адрес a_{11}^* на a_{12}^*, т. е. увеличить этот адрес на 1; б) увеличить третий адрес S_1 команды 5) на 1.

Переадресацию команды 5) можно организовать при помощи PA во внешнем цикле, но для этого следует по команде 1) не только очищать PA, но и восстанавливать регистр адреса внешнего цикла

$$1)' \quad PA \quad 0^* \quad 0 \quad BP,$$

причем ячейку BP следует поставить между командами 4) и 5), а команду 5) переписать в виде

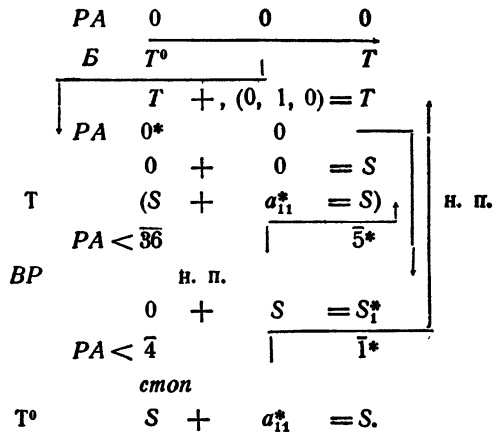
$$5)' \quad 0 + S = S_1^*.$$

Команду 3) не удастся переадресовать при помощи PA, ибо в ней PA уже использовано. Для замены a_{11}^* на a_{12}^*, затем a_{12}^* на a_{13}^* и т. д. приходится применять в данном случае команду сложения мантисс:

$$8) \quad +, \ (0, 1, 0) = 3).$$

В силу этого команда 3) меняется в памяти и, следовательно, требуется ее начальное восстановление.

Учитывая все эти замечания, составим программу для нахождения сумм $S_1, \ldots, S_5$:



Пример 2.33 характерен тем, что для переадресации цикла в цикле оказалось недостаточным применения регистра адреса; пришлось переадресовывать команду T в памяти при помощи операции сложения мантисс. Этот пример показывает, что регистр адреса не является универсальным средством переадресации.

Заметим, что регистр адреса удается использовать лишь тогда, когда все переменные адреса переадресуемых команд цикла должны изменяться (увеличиваться или уменьшаться) с одним и тем же шагом. Большинство встречающихся при программировании циклов имеет именно такой характер. Однако встречаются задачи и другого рода. Пусть, например, нам нужно переставить члены последовательности

$$a_1, a_2, \ldots, a_{50}$$

в обратном порядке, размещая новую последовательность в ячейках

$$b_1, b_2, \ldots, b_{50}$$

(см. пример 4.29).

В этом случае следует в цикле при $i = 1, 2, \ldots, 50$ выполнить команду

$$a_i = b_{51-i}.$$

Очевидно, переадресовывать такую команду при помощи PA не удастся.

В заключение главы рассмотрим пример, в котором команда занесения в регистр используется иначе, чем в предыдущих примерах.

Пример 3.33. Найти максимальное по модулю из восьми чисел $a_0, a_1, a_2, \ldots, a_7$ и поместить его в ячейку M. Во второй адрес ячейки у заслать индекс числа, равного M.

Будем находить число M следующим способом:

Сначала в ячейку M зашлем нуль, затем будем последовательно сравнивать (по модулю) числа $a_0, a_1, \ldots, a_7$ с M; в случае, если, например, $|a_j| \geqslant M$, в ячейку M зашлем a_j, если же $|a_j| < M$, оставим в M старое содержание, затем перейдем к сравнению с M следующего числа a_{j+1} и т. д.

Рабочая часть цикла на j-м шаге должна исполняться в виде

$$\begin{array}{lll} 1) & |a_j| - |M| = 0 & \\ 2) & У_1 & 4) \\ 3) & 0 + a_j & = M. \end{array}$$

Действительно, если $|a_j| \geqslant M$, то команда 1) вырабатывает сигнал $\omega = 0$, команда 2) передает управление на 3) и в M засылается большее по модулю число a_j. Если же $|a_j| < M$, то после 1) имеем $\omega = 1$ команда 2) передает управление на продолжение счета, оставив в M старое содержимое, поскольку команда 3) обходится. Команды 1) и 3) рабочей части цикла содержат переменный адрес a_j и поэтому должны переадресовываться.

Добиться исполнения рабочей части цикла в виде 1) — 3) можно, заслав нуль в регистр адреса и переписав команды 1) — 3) в виде

$$\begin{array}{lll} 1)' & |a_0|^* - |M| = 0 & \\ 2)' & У_1 & 4) \\ 3)' & 0 + a_0^* & = M. \end{array}$$

Выполняя эти три команды при значениях PA, равных 0000, 0001, ..., 0007, мы переберем все члены последовательности и получим в M максимальное из них по модулю:

$$\begin{array}{lll} 1) & PA & 0 \quad 0 \quad 0 \\ 2) & & 0 + 0 = M \\ 3) & & |a_0|^* - |M| = 0 \\ 4) & У_1 & \\ 5) & & 0 + a_0^* = M \\ 6) & PA < \overline{7} & \overline{1}^* \end{array}$$

Так решена первая часть задачи: нахождение M. Теперь по условию задачи нам требуется получить индекс члена последовательности, равного M. Для этого достаточно между командами 5) и 6) написанной

программы вставить команду $PA\ \ 0^*\ \ 0^*\ \ \nu$. Получаем программу:

1) $PA \qquad 0 \qquad 0 \qquad 0$
2) $\qquad\quad 0 \quad +0 \quad =M$
3) $\qquad\quad |a_0|^* - |M| = 0$
4) $У_1$
5) $\qquad\quad 0 \quad +a_0^* \quad =M$
6) $\qquad PA \quad 0^* \qquad 0^* \qquad \nu$
7) $\qquad PA < \overline{7} \qquad\quad \overline{1}^*$
8) $\qquad стоп.$

Проследим, как образуется ν. Пусть на j-м шаге цикла оказывается, что $|a_j| > M$. Тогда в результате выполнения команд 3), 4), 5) в M зашлется наибольший (по модулю) элемент a_j последовательности $a_0, a_1, \ldots, a_j$. Содержимое регистра адреса при этом будет равно j. Команда 6) после этого выполнится в виде

$$PA \quad j \quad j \quad \nu,$$

т. е. старое значение PA сохранится, а в ν зашлется команда $PA\ j$, во втором адресе которой запомнился индекс числа последовательности $a_0, a_1, \ldots, a_j$, равного M. После прохождения всего цикла во втором адресе ν окажется искомый индекс.

ГЛАВА VII

ОПЕРАЦИИ НАД МАШИННЫМИ СЛОВАМИ

§ 34. Машинное слово

Как уже было сказано в § 16, ячейка памяти может хранить набор из сорока пяти двоичных цифр (нулей или единиц), который мы и называем *машинным словом*. До сих пор мы рассматривали машинные слова, которые употреблялись как число (плавающее или фиксированное) или как команда. В обоих случаях отдельные разряды слова определенным образом связаны между собой.

Операции, которые рассматривались в предыдущих главах, были операциями над числами или над командами. В настоящей главе будут разобраны операции над словами, характер которых безразличен, т. е. которые рассматриваются именно как набор нулей и единиц, записанных в определенном порядке и никак не связанных между собою.

Мы рассмотрим два типа таких операций: операции *сдвига* и *логические операции*. В операциях сдвига действия происходят над некоторым машинным словом, рассматриваемым как единый набор независимых друг от друга двоичных цифр. В логических операциях действия происходят фактически даже не со словами, а с отдельными разрядами этих слов.

§ 35. Сдвиги

Сдвиг заключается в том, что машинное слово, как целое, сдвигается внутри разрядной сетки ячейки влево или вправо на определенное число разрядов. Двоичные разряды, выходящие за разрядную сетку, теряются. На место освобождающихся разрядов ставятся нули.

Так, например, слово

011 011 111 101 100 011 000 100 110 001 111 010 110 111 001

при сдвиге на 15 разрядов вправо превращается в слово

000 000 000 000 000 011 011 111 101 100 011 000 100 110 001,

а при сдвиге на 15 разрядов влево — в слово

011 000 100 110 001 111 010 110 111 001 000 000 000 000 000.

Если рассматривать слово как целое двоичное число, то сдвиг вправо на один разряд означает уменьшение числа вдвое. Поэтому

сдвиг вправо называют *отрицательным*, а сдвиг влево, увеличивающий слово, называют *положительным сдвигом*.

Будем считать, что адрес сдвигаемого слова указывается во втором адресе команды сдвига, а результат сдвига записывается по третьему адресу. Первый адрес команды сдвига используется для указания направления сдвига и числа разрядов, на которое нужно сдвинуть слово. Это можно сделать различными способами.

Проще всего написать число разрядов, на которое нужно произвести сдвиг, прямо в виде фиксированного числа единиц первого адреса. Здесь мы еще раз встречаемся с командой, адрес которой не означает адреса ячейки памяти, а представляет собой некоторое условное число (такую же роль играли, например, крайние адреса в командах проверки содержимого регистра).

Такую операцию называют *сдвигом по адресу*. Для записи этой команды в левой части воспользуемся следующим обозначением. Сдвиг будем обозначать стрелкой, направленной в ту же сторону, в которую надо сдвинуть слово, а перед нею указывать число разрядов, на которое сдвиг происходит.

Так, команда

$$\overline{24} \rightarrow a = b$$

означает, что слово, содержащееся в ячейке a, надо сдвинуть на $24_8 = 20_{10}$ разрядов вправо и результат записать в ячейку b. Команда

$$\overline{37} \leftarrow a = b$$

означает, что слово a сдвигается на $37_8 = 31_{10}$ разрядов влево.

При кодировке этой команды в первом адресе записывается *условное число*, равное $100_8 + \nu$, где ν — число разрядов сдвига со знаком, в зависимости от того, положительным или отрицательным является сдвиг. Так, в первом из приведенных выше примеров в первом адресе надо написать число $100 - 24 = 054$, а во втором $100 + 37 = 137$.

Кроме сдвига всего слова, используется также операция *сдвига мантиссы* или, что то же самое, *сдвига адресной части* слова. Она обозначается так же, как и предыдущая, только по аналогии с фиксированными действиями, для указания сдвига мантиссы мы будем ставить запятую у стрелки, означающей сдвиг. Например, команда

$$\overline{25} \rightarrow, \quad a = b$$

означает сдвиг мантиссы ячейки a на $25_8 = 21_{10}$ разрядов вправо. При кодировке первый адрес команды должен содержать условное число 053. При выполнении команды сдвига мантиссы порядок остается на месте, а мантисса сдвигается на указанное число разрядов.

Сдвиг по адресу можно применять тогда, когда величина сдвига известна программисту заранее. Возможны, однако, случаи, когда эта

величина заранее неизвестна, а определяется в процессе работы программы. Тогда применяются операции *сдвига по порядку*.

Команду сдвига по порядку мы будем обозначать так:

$$a\,[\rightarrow]\,b = c.$$

При выполнении этой команды слово из ячейки b сдвигается и результат сдвига кладется в ячейку c. Величина сдвига принимается равной истинному порядку числа, записанного в ячейке a. Направление сдвига определяется знаком порядка этого числа. Таким образом, семь младших разрядов порядка ячейки a используются здесь так же, как и первый адрес команды сдвига по адресу.

Если, например, машинный порядок числа в ячейке a равен 114, то это означает положительный сдвиг (влево) на $14_8 = 12_{10}$ разрядов (при этом истинный порядок числа равен, как известно, 14). Если машинный порядок есть 070, то истинный порядок будет равен -10 и сдвиг происходит на $10_8 = 8_{10}$ разрядов вправо.

Кроме описанной, существует также операция *сдвига мантиссы по порядку*. Она обозначается так:

$$a\,[\rightarrow,]\,b = c,$$

и выполняется так же, как и сдвиг по порядку, но только сдвигается не все слово, а лишь его адресная часть (мантисса).

При операциях сдвига управляющий сигнал ω вырабатывается по следующим правилам. Он принимает значение $\omega = 1$ в тех случаях, когда в результате сдвига получилось нулевое слово, если сдвигалось все слово, либо слово с нулевой мантиссой, если сдвигалась только мантисса. Во всех остальных случаях вырабатывается сигнал $\omega = 0$.

Приведем теперь таблицу кодов описанных операций.

Коды операций сдвига

Название операции	Обозначение	Код
Сдвиг мантиссы по адресу	$a \rightleftarrows,\ b = c$	14
Сдвиг мантиссы по порядку . . .	$a\,[\rightarrow,]\ b = c$	34
Сдвиг слова по адресу	$a \rightleftarrows\ b = c$	54
Сдвиг слова по порядку	$a\ [\rightarrow]\ b = c$	74

Следующие примеры показывают использование операций сдвига.

Пример 1.35. Найдем модуль двоичного числа x.

Эта задача решалась в гл. IV (см. пример 2.19) при помощи операции условной передачи управления. С помощью сдвига ее можно решить проще. Знак числа x определяется 44-м разрядом слова: если в этом разряде стоит 0, число x положительно, если в нем 1 — от-

рицательно. Поэтому, чтобы получить модуль x, следует превратить 44-й разряд в нуль. Достигнуть этого можно при помощи двух последовательных сдвигов слова x влево и вправо на два разряда *):

$$\overline{2} \leftarrow x = R,$$
$$\overline{2} \rightarrow R = |x|.$$

Пример 2.35. Рассмотрим задачу *сжатия таблицы*. Пусть в машину введена таблица 1000 двоичных чисел $x_0, x_1, x_2, \ldots, x_{999}$, каждое из которых известно с точностью, не превышающей 14 двоичных знаков (4—5 десятичных знаков).

Сожмем эту таблицу вдвое, помещая в каждую из 500 ячеек $a_0, a_1, \ldots, a_{499}$ по два числа из исходной таблицы.

45	44 ← Разряды → 23	22 ← Разряды → 1
0	x_0	x_1

Рис. 32.

Посмотрим сначала, каким образом можно два двоичных числа x_0 и x_1, заданных с указанной точностью, поместить в одну ячейку. Двоичное число x_0 состоит из восьми разрядов порядковой части (знак числа и порядок) и 36 разрядов мантиссы. Из мантиссы, согласно условию, достаточно оставить 14 левых разрядов (с 23-го по 36-й), а правые 22 разряда отбросить. Таким образом, в числе x_0 существенными являются 22 левых разряда. То же можно сказать и о числе x_1.

Если выделить в числах x_0 и x_1 по 22 разряда, то их можно разместить в одной ячейке a_0 по схеме, указанной на рис. 32.

Сделать это можно при помощи следующих операций:

1) $\overline{26} \rightarrow x_0 = R$
2) $\overline{26} \leftarrow R = R$
3) $\overline{26} \rightarrow x_1 = a_0$
4) $R +, a_0 = a_0$.

Команды 1) и 2) превращают в нуль младшие 22 разряда числа x_0 и помещают затем старшие 22 разряда в левую часть ячейки R. При помощи команды 3) старшие 22 разряда числа x_1 помещаются в правую половину ячейки a_0. Наконец, команда 4) ставит в левую часть ячейки a_0 число x_0.

*) Напомним, что значение 45-го разряда слова, изображающего двоичное число, не влияет на значение числа.

Чтобы сжать всю таблицу, следует эти четыре команды включить в цикл, который должен повториться 500 раз.

Программа цикла с использованием регистра адреса будет иметь вид

$$
\begin{array}{lll}
PA & 0 & 0 \qquad\qquad 0 \longrightarrow \\
Б & T^0 & \qquad\qquad\quad T \\
 & \boxed{T+,\ (0,\ 0,\ 1)=T} & \\
 & \overline{26} \rightarrow & x_0^*=R_1 \\
 & \overline{26} \leftarrow & R_1=R_1 \\
 & \overline{26} \rightarrow & x_1^*=R_2 \\
\mathrm{T} & (R_1+, & R_2=a_0) \quad \text{н. п.} \\
PA & <\overline{1746} & \qquad\quad \overline{2}^*
\end{array}
$$

стоп

$$
\mathrm{T}^0 \quad R_1+, \quad R_2=a_0.
$$

Пример 3.35. Разберем задачу, обратную рассмотренной в предыдущем примере. Пусть дана сжатая таблица $a_0, a_1, \ldots, a_{499}$, причем в каждой из ячеек a_i лежит пара чисел x_{2i}, x_{2i+1}, занимающих по 22_{10} разряда каждое. Пусть, кроме того, задана ячейка s, в третьем адресе которой указан некоторый восьмеричный номер n нужного числа

$$s=(0,\ 0,\ n).$$

Требуется выбрать это число из таблицы и записать его в ячейку x.

Выберем сначала слово a_k, содержащее число x_n. Так как числа x_0, x_1 лежат в ячейке a_0, числа x_2, x_3 — в ячейке a_1 и т. д., то число x_n лежит в ячейке a_k с номером k, равным целой части $\frac{n}{2}$. Целую часть $\frac{n}{2}$ можно получить, сдвинув ячейку s на один разряд вправо,

$$\overline{1} \rightarrow s=q.$$

Если теперь T° означает команду a_0+, $0=x$, то можно переслать слово a_k в ячейку x с помощью следующей программы:

$$
\begin{array}{l}
\overline{1} \rightarrow s=q \\
\overline{30} \leftarrow q=q \\
T^0+,\ q=T \\
\mathrm{T} \text{ н. п.}
\end{array}
$$

Остается узнать, в какой половине ячейки x находится нужное число. Для этого надо определить четность числа n: при четном n число x_n находится в левой половине ячейки, а при нечетном — в правой. Четность n определяется значением первого разряда ячейки s. Поэтому можно выяснить четность n, сдвинув ячейку на $44_{10}=54_8$ разрядов влево. Тогда в ячейке останется лишь последний разряд. Если n число четное, и этот разряд равен нулю, то результатом сдвига будет нулевое слово и выработается сигнал $\omega=1$. Если же n нечетно, то в сдвинутом слове останется единица последнего разряда, так что управляющий сигнал ω будет равен нулю.

Программу выборки можно теперь закончить командами

$$
\begin{array}{ll}
1) & \overline{54} \leftarrow s = 0 \\
2) & У_0 \\
3) & \overline{26} \rightarrow x = x \\
4) & \overline{26} \leftarrow x = x \\
5) & стоп.
\end{array}
$$

Действительно, при четном n команда 2) передает управление команде 3), которая сдвинет ячейку на 22 разряда вправо, в результате чего сотрется правое число в ячейке. После этого команда 4) вернет оставшееся нужное число на место. При нечетном n управление перейдет сразу команде 4), которая сдвинет нужное число налево, стерев ненужное предыдущее.

Окончательный вид программы приведен в табл. 1.35 в кодированном виде. Здесь предположено, что таблица находится в ячейках, начиная с 2000, ячейкам s, x и q приданы соответственно адреса 1012, 1020 и 1021, а $n=17_8$.

С помощью регистра адреса ту же программу можно написать короче:

$$
\begin{array}{lll}
1) & & \overline{13} \leftarrow \ s = q \\
2) & [PA] & \quad q \\
3) & & a_0^* +, \ 0 = x \\
4) & & \overline{54} \leftarrow \ s = 0 \\
5) & У_0 & \\
6) & & \overline{26} \rightarrow \ x = x \\
7) & & \overline{26} \leftarrow \ x = x. \\
8) & стоп & \\
s\ 9) & & (0,\ 0,\ n).
\end{array}
$$

Действительно, при сдвиге ячейки s на $13_8=11_{10}$ разрядов во втором адресе ячейки q окажется целая часть $\frac{n}{2}$. Она берется в регистр

и затем команда 3) переадресуется по $РА$, перенося в ячейку x нужное слово.

Таблица 1.35

					1000	A	
	$\overline{1} \rightarrow s=q$	1000	0	54	0077	1012	1021
	$\overline{30} \leftarrow q=q$	1	0	54	0130	1021	1021
	$T^0+, q=T$	2	0	13	1011	1021	1003
T	$(a_0+, 0=x)$	3				н. п.	
	$\overline{54} \leftarrow s=0$	4	0	54	0154	1012	0000
	$У_0$	5	0	76	0000	1007	0000
	$\overline{26} \rightarrow x=x$	6	0	54	0052	1020	1020
	$\overline{26} \leftarrow x=x$	7	0	54	0126	1020	1020
	стоп	1010	0	17	0000	0000	0000
T^0	$a_0+, 0=x$	1	0	13	2000	0000	1020
s	(0, 0,17)	2	0	00	0000	0000	0017

Пример 4.35. Найдем целую и дробную части положительного двоичного числа x.

Пусть истинный порядок числа x равен p, причем $p > 0$. Тогда слово, изображающее число x, можно разбить на части по схеме, изображенной на рис. 33. Из нее видно, что мантиссу дробной части

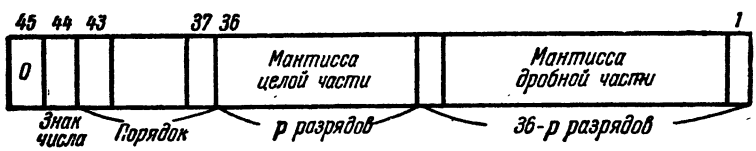


Рис. 33.

можно получить, сдвинув мантиссу x на p разрядов влево. Ясно, что величина сдвига определяется порядком самого числа x.

Итак, в результате сдвига $x[\rightarrow,]x=b$ в мантиссе ячейки b окажется мантисса дробной части x, а порядок останется равным

порядку x. Дробная часть числа должна иметь нулевой порядок, т. е. машинный порядок, равный 100_8. Его можно приписать ячейке b, прибавив ее фиксированным образом к константе (100, 0, 0, 0):

$$(100,\ 0,\ 0,\ 0)+,\ b=b.$$

Для нахождения целой части достаточно из числа x вычесть дробную часть. Таким образом, наша задача решается программой

$$\begin{aligned} x\,[\to,]\ & x=b\\ (100,0,0,0)+,\ & b=b\\ x\ -\ & b=a\\ & \textit{стоп.} \end{aligned}$$

Заметим, что дробная часть может получиться в ячейке b ненормализованной.

§ 36. Первоначальные сведения из математической логики

Логика есть наука о формах и законах мышления. В математической логике рассматриваются те стороны логики, которым может быть придана математическая форма. Для дальнейшего нам необходимо ознакомиться с первоначальными понятиями той части математической логики, которая называется *алгеброй логики* или *исчислением высказываний*.

Под *высказыванием* понимается любое утверждение, о котором имеет смысл говорить, что оно *истинно* или *ложно*. Так, «три — простое число» является истинным высказыванием, а «восемь делится на пять» — ложным. В исчислении высказываний мы интересуемся, собственно, не самими высказываниями, т. е. не их содержанием, а значением *функции истинности* высказывания.

Функция истинности ставит в соответствие каждому высказыванию определенное число, которое называют *значением функции истинности*. Мы будем полагать, что для истинных высказываний значение функции истинности есть единица, а для ложных — нуль. Таким образом, значение функции истинности есть двоичная переменная.

Если значения функции истинности двух высказываний совпадают, то такие высказывания называют *эквивалентными* или *равнозначными*.

Уже известным примером двоичной переменной, которая служит значением функции истинности, является управляющий сигнал ω. При арифметических действиях сложения, вычитания или вычитания модулей значение сигнала ω есть значение функции истинности высказывания «результат действия отрицателен». Читатель легко самостоятельно сформулирует высказывания, соответствующие другим операциям машины. Еще одним примером двоичной переменной является сигнал аварийного останова. Он служит функцией истинности для высказывания «результат действия переполняет разрядную сетку».

Задачей исчисления высказываний является рассмотрение сложных высказываний, составленных из простых с помощью различных логических связей, и вычисление функций истинности сложных высказываний. Рассмотрим основные примеры таких *логических связей* и *логических операций*.

I. Отрицание. *Отрицанием высказывания A называется такое другое высказывание, которое истинно, если высказывание A ложно, и ложно, если высказывание A истинно.* Отрицание высказывания A коротко назы-

вают «*не A*» и обозначают $\overline{A}$. Значение истинности высказывания $\overline{A}$ получается по таблице

$$\left.\begin{aligned}\overline{1}&=0,\\ \overline{0}&=1.\end{aligned}\right\} \tag{1.36}$$

II. Логическое произведение. *Логическим произведением высказываний A и B называется такое высказывание C* ($C=A\wedge B$), *которое истинно тогда и только тогда, когда истинны оба первоначальных высказывания.*

Логическое умножение называют также конъюнкцией или «*операцией и*». Значение функции истинности логического произведения дается таблицей

$$\left.\begin{aligned}0\wedge 0&=0,\\ 0\wedge 1&=0,\\ 1\wedge 0&=0,\\ 1\wedge 1&=1,\end{aligned}\right\} \tag{2.36}$$

которая, собственно, совпадает с обычной таблицей умножения для двоичной системы.

III. Логическая сумма. *Логической суммой высказываний A и B называется такое сложное высказывание C* ($C=A\vee B$), *которое ложно тогда и только тогда, когда оба первоначальных высказывания ложны.* Иначе говоря, логическая сумма высказываний истинна, когда истинно A или B (или оба). Логическое сложение называют также *дизъюнкцией* и «*операцией или*».

Значение функции истинности логической суммы определяется таблицей

$$\left.\begin{aligned}0\vee 0&=0,\\ 0\vee 1&=1,\\ 1\vee 0&=1,\\ 1\vee 1&=1,\end{aligned}\right\} \tag{3.36}$$

которая, в отличие от (2.36), из-за последней строки уже не совпадает с двоичной таблицей сложения.

IV. Равнозначность. *Равнозначностью двух высказываний называют такое высказывание, которое истинно тогда и только тогда, когда значения функций истинности первоначальных высказываний совпадают,* т. е. когда оба данных высказывания одновременно истинны, либо одновременно ложны. Для обозначения равнозначности употребляется знак $\sim$: $A\sim B=C$. Значение функции истинности получается из таблицы

$$\left.\begin{aligned}0\sim 0&=1,\\ 0\sim 1&=0,\\ 1\sim 0&=0,\\ 1\sim 1&=1.\end{aligned}\right\} \tag{4.36}$$

V. Отрицание равнозначности. Это сложное высказывание обозначается $A\not\sim B=C$ и определяется равенством

$$A\not\sim B=(\overline{A\sim B}).$$

Значение функции истинности можно получить, пользуясь таблицами (1.36) и (4.36). Это дает таблицу:

$$\left.\begin{aligned} 0 \not\sim 0 &= 0, \\ 0 \not\sim 1 &= 1, \\ 1 \not\sim 0 &= 1, \\ 1 \not\sim 1 &= 0. \end{aligned}\right\} \tag{5.36}$$

Из таблиц (1.36) — (5.36) видно, что все логические операции коммутативны, т. е. подчиняются переместительному закону.

§ 37. Логические операции машины

Некоторым из рассмотренных в предыдущем параграфе операций исчисления высказываний соответствуют машинные операции над словами, которые и называются *логическими операциями* машины. В логических операциях каждое слово рассматривается как набор двоичных цифр, не связанных между собою. Значения отдельных разрядов слова a будем обозначать через a_l ($l = 1, 2, \ldots, 45$). Они являются двоичными переменными, поскольку могут принимать значения 0 или 1.

Логические операции выполняются, собственно, не над словами, а над каждым разрядом слова в отдельности, т. е. *по-разрядно*. У машины есть три логические операции, которые определяются следующим образом.

Логическим сложением слов a и b называется операция $a \vee b = c$, определяемая формулой

$$a_l \vee b_l = c_l \qquad (l = 1, 2, \ldots, 45).$$

Иначе говоря, слово c, являющееся логической суммой слов a и b, образуется в результате логического сложения соответствующих разрядов слов — слагаемых в соответствии с (3.36).

Логическое сложение выполняется значительно быстрее, чем плавающее и фиксированное, потому что не требует времени для переноса единиц из разряда в разряд. По этой причине его удобно применять для пересылки слов из одной ячейки в другую, необходимость в которой встречается довольно часто. В предыдущих главах для пересылки слов мы пользовались плавающим или фиксированным сложением с нулем. Более удобно делать это путем логического сложения с нулем.

Действительно, при выполнении команды $0 \vee b = c$, в соответствии с табл. (3.36), находим, что $b_l = c_l$ ($l = 1, 2, \ldots, 45$), т. е. слово из ячейки b пересылается в ячейку c без изменения. В дальнейшем мы будем широко использовать эту возможность и введем для команды $0 \vee b = c$ сокращенное обозначение $b = c$.

Операция логического сложения применяется также для формирования некоторого слова (например, команды) из отдельных частей, находящихся в различных ячейках. Например, если слово a имеет вид $(\alpha, 0, 0)$, а слово b — вид $(0, \beta, 0)$, то путем логического сложения $a \vee b = c$ мы получим слово c, имеющее вид $(\alpha, \beta, 0)$.

Логическим умножением слов a и b называется операция $a \wedge b = c$, выполняемая в соответствии с формулой

$$a_i \wedge b_i = c_i \qquad (i = 1, 2, \ldots, 45).$$

Операцию логического умножения часто называют *пересечением* или *выделением*. Чаще всего ее применяют для того, чтобы высечь из слова некоторую группу разрядов, заменив остальные разряды нулями. Для этого нужно логически умножить данное слово на такое, в котором на месте выделяемых разрядов стоят единицы, а в остальных разрядах нули.

Сверкой двух слов a и b называют операцию $a \not\sim b = c$, определяемую формулой

$$a_i \not\sim b_i = c_i \qquad (i = 1, 2, \ldots, 45).$$

Операция сверки применяется главным образом для проверки логических условий, которым можно придать вид совпадения двух слов. Действительно, если слова a и b совпадают, то результат операции $a \not\sim b$ будет, как видно из (5.36), нулевым словом и выработается сигнал $\omega = 1$. Если же a и b отличаются хотя бы одним разрядом, то $\omega = 0$.

При выполнении любой из трех перечисленных логических операций вырабатывается управляющий сигнал $\omega = 0$ во всех случаях, кроме того, когда результатом операции является нулевое слово. В последнем случае получаем $\omega = 1$.

Коды логических операций приведены в следующей таблице:

Коды логических операций

Название операции	Обозначение	Код
Логическое сложение	$a \vee b = c$	75
Логическое умножение	$a \wedge b = c$	55
Сверка	$a \not\sim b = c$	15

Перейдем теперь к рассмотрению некоторых примеров применения логических операций.

Пример 1.37. Числу a присвоить знак числа b, результат записать в ячейку c *).

*) У некоторых трехадресных машин эта операция является элементарной и записывается в виде $|a|\operatorname{sgn} b = c$.

Знак двоичного числа характеризуется 44-м разрядом слова, изображающего это число. Поэтому для решения сформулированной задачи следует в числе b выделить знаковый 44-й разряд, в числе a все остальные разряды (т. е. взять модуль a), и результаты этих двух операций логически сложить.

Знак числа b можно выделить, логически умножая это число на константу e_{44}, имеющую единицу в 44-м разряде и нули в остальных:

$$b \bigwedge e_{44} = c.$$

Для нахождения $|a|$ можно логически умножить слово a на константу, имеющую единицы во всех разрядах, кроме знакового 44-го; это слово записывается в восьмеричном виде так: 577 7777 7777 7777. Обозначая это слово через $e_{\overline{44}}$, напишем команду нахождения модуля a:

$$a \bigwedge e_{\overline{44}} = |a|.$$

Присвоение числу $|a|$ знака числа b осуществляется при помощи логического сложения:

$$c \bigvee |a| = c.$$

Таким образом, задача решается при помощи трех команд:

$$b \bigwedge e_{44} = c,$$
$$a \bigwedge e_{\overline{44}} = |a|,$$
$$c \bigvee |a| = c,$$

и двух констант:

$$e_{44}: (200,\ 0000,\ 0000,\ 0000),$$
$$e_{\overline{44}}: (577,\ 7777,\ 7777,\ 7777).$$

Пример 2.37. Подсчитать число нулевых двоичных разрядов в слове a; полученное число поместить в порядковую часть слова u.

Очистим сначала счетчик числа нулей:

$$1) \qquad 0 = u.$$

Для того чтобы узнать, есть ли нуль в первом разряде слова a, выполним операцию логического умножения:

$$2)\ a \bigwedge (0,\ 0,\ 1) = 0.$$

Если в первом разряде a есть нуль, то эта операция выработает сигнал $\omega = 1$; если в этом разряде единица, то $\omega = 0$. Поэтому после 2) следует поместить две команды:

$$3)\ V_0 \qquad 5)$$
$$4) \qquad u+,\ (1,\ 0,\ 0,\ 0) = u.$$

Сдвинем теперь слово a на один разряд вправо:

$$5)\ \overline{1} \to a = a.$$

Тогда второй разряд займет место первого и после повторения команд 2) — 4) в u появится число нулей в двух разрядах слова.

Рабочую часть цикла следует повторить $45_{10} = 55_8$ раз. Объединяя команды 1) — 5), организуем проверку окончания цикла по счетчику, записанному в порядковой части ячейки ν. Кроме того, чтобы программа была самовосстанавливающейся, слово a в памяти не должно «портиться», т. е. изменяться. Поэтому его следует переслать в ячейку α и дальнейшие операции производить над словом α.

Программу можно записать в таком виде:

1)		(55, 0, 0, 0)	,—	(1, 0, 0, 0)	$=\nu$
2)				0	$=u$
3)				a	$=\alpha$
4)		α	$\wedge$	(0, 0, 1)	$=0$
5)	$У_0$				
6)		u	$+$,	(1, 0, 0, 0)	$=u$
7)		$\overline{1}$	$\to$	α	$=\alpha$
8)		ν	,—	(1, 0, 0, 0)	$=\nu$
9)	$У_0$				
10)		*стоп.*			

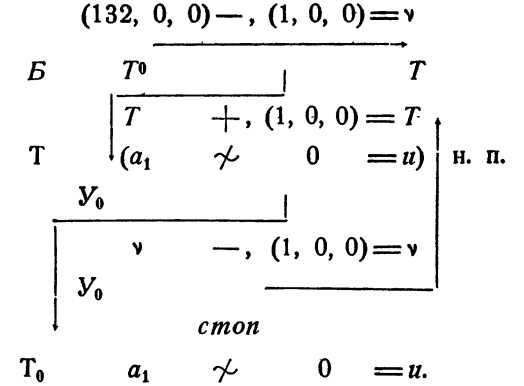
Более короткую и быструю программу можно получить, пользуясь другим алгоритмом. Сложим все разряды слова a. Полученная таким образом сумма будет равна количеству единиц слова. Число нулей можно получить, вычитая эту сумму из числа 55_8, разрядов машинного слова. Это можно осуществить с помощью такой программы:

1)			a		$=\alpha$
2)			(0, 0, 55)		$=u$
3)		α	$\wedge$	(0, 0, 1)	$=q$
4)		u	—,	q	$=u$
5)		$\overline{1}$	$\to$	α	$=\alpha$
6)	$У_0$				
7)		$\overline{44}$	$\leftarrow$	u	$=u$
8)		*стоп.*			

Пример 3.37. Из последовательности чисел a_1, a_2, ..., a_{90}, расположенных в идущих подряд ячейках памяти, найти первое ненуле-

вое и заслать в ячейку u. Если все числа последовательности нулевые, в ячейку u заслать нуль.

Составим программу, образуя цикл при помощи восьмеричного счетчика, записанного в первом адресе ячейки ν и используя операцию сверки с нулем для нахождения первого ненулевого элемента последовательности:

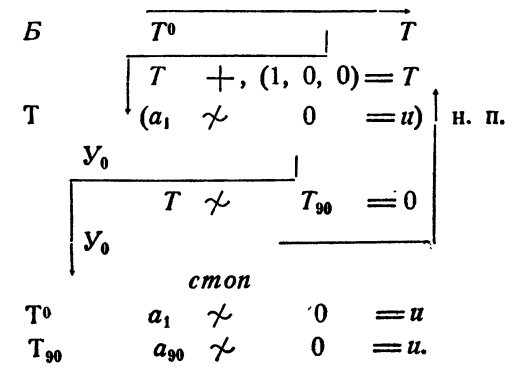


Для пояснения программы заметим, что при сверке с нулем слово не изменяется; поэтому команда T всякий раз записывает в ячейку u очередное слово a. Если все слова были нулевыми, то после окончания цикла в ячейке u будет записано нулевое слово.

В этой программе можно еще раз воспользоваться операцией сверки для проверки окончания цикла. Цикл следует продолжать до тех пор, пока не будет сверено с нулем слово a_{90}. Поэтому последнее состояние команды T будет

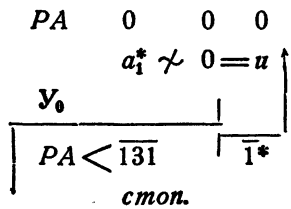
$$T_{90}: a_{90} \neq 0 = u,$$

и проверку окончания по счетчику можно заменить проверкой по состоянию переменной команды. Тогда счетчик окажется вообще ненужным. Программа будет иметь вид



Этот прием позволяет сократить, хотя и незначительно, программу и число рабочих ячеек, а в некоторых случаях и уменьшить число команд в цикле (на одну). Однако необходимо иметь в виду, что подавляющее большинство ошибок при организации цикла состоит в неверной проверке окончания, а ошибки при определении последнего состояния переменной команды много вероятнее, чем при определении последнего состояния счетчика. Поэтому рассмотренным приемом рекомендуется не пользоваться или же пользоваться с большой осторожностью.

Для машины с регистром адреса программа может быть записана совсем коротко. Она состоит лишь из пяти команд:



§ 38. Логические шкалы

При помощи операций логического умножения и сдвига можно организовать работу с *логическими шкалами.*

Логической шкалой называется двоичное слово, в котором каждый разряд имеет значение истинности некоторого высказывания. При помощи одной логической шкалы, содержащейся в ячейке трехадресной машины, можно записать значения истинности не более 45 высказываний. Если число высказываний больше 45, то под логическую шкалу можно занять несколько ячеек.

Логическими шкалами при программировании пользуются тогда, когда в задаче требуется проверка многих однотипных логических условий.

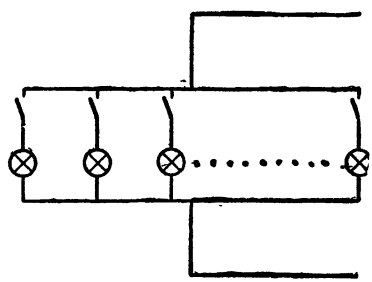


Рис. 34.

Пример 1.38. В электрическую цепь параллельно включены 20 лампочек с сопротивлениями R_1, R_2, ..., R_{20} (рис. 34). Каждая лампочка снабжена своим выключателем. Составить программу расчета мощности, потребляемой этой сетью, при произвольном положении выключателей.

Если все выключатели находятся в положении *включено,* то величина потребляемой мощности W находится по формулам

(сопротивлением проводов пренебрегаем)

$$W=\frac{U^2}{R},\quad \frac{1}{R}=\frac{1}{R_1}+\frac{1}{R_2}+\ldots+\frac{1}{R_{20}},$$

где U — напряжение в цепи.

Величина проводимости $\frac{1}{R}$ подсчитывается при помощи простого арифметического цикла:

$$\begin{array}{lcccc} PA & 0 & 0 & 0 & \\ & & 0 & =\frac{1}{R} & \\ & & \text{«1»} : R_1^* & =k & \\ & & \frac{1}{R}+k & =\frac{1}{R} & \Big\uparrow \\ PA<\overline{23} & & & \overline{1}^*, & \end{array}$$

после чего W определяется двумя арифметическими операциями:

$$U\cdot U=k,$$

$$k\cdot\frac{1}{R}=W.$$

Рассмотрим теперь более сложный случай, когда часть лампочек отключена от сети, а часть включена. Положение выключателей охарактеризуем 20 двоичными переменными $\lambda_1, \lambda_2, \ldots, \lambda_{20}$. Пусть значение $\lambda_i=1$ соответствует включенной i-й лампочке, а $\lambda_i=0$ — выключенной. Тогда проводимость цепи определится формулой

$$\frac{1}{R}=\frac{\lambda_1}{R_1}+\frac{\lambda_2}{R_2}+\ldots+\frac{\lambda_{20}}{R_{20}}.$$

Состояние цепи, т. е. положение выключателей, естественно задать при помощи логической шкалы, для которой мы используем 20 младших разрядов (от 1-го до 20-го) ячейки λ. В первый разряд мы занесем значение λ_1, во второй λ_2, ..., в двадцатый — λ_{20}. Счет проводимости по последней формуле организуем так: очистим ячейку $\frac{1}{R}$, в которой подсчитывается проводимость:

$$1)\qquad 0=\frac{1}{R}.$$

Проверим при помощи операции логического умножения значение λ_1 первого разряда ячейки λ; если там единица, прибавим к $\frac{1}{R}$ величину

проводимости первой лампочки:

2) $\lambda \wedge (0,\ 0,\ 1) = 0$

3) $У_1$ 6)

4) «1»: $R_1 = k$

5) $\frac{1}{R} + k = \frac{1}{R}$.

Сдвинем теперь шкалу λ на один разряд вправо

6) $\overline{1} \to \lambda = \lambda$,

увеличим второй адрес команды 4) на 1 и передадим управление на 2). Тогда к величине $\frac{1}{R}$ прибавится второе слагаемое $\frac{\lambda_2}{R_2}$.

Продолжая таким образом далее, мы получим величину проводимости, а затем и мощность W.

Используя регистр адреса, получим следующую программу расчета мощности:

1)	PA	0	0	0
2)			0	$= \frac{1}{R}$
3)			λ	$= \lambda_{раб}$
4)		$\lambda_{раб} \wedge (0,$	$0,\ 1) = 0$	
5)	$У_1$			
6)		«1» :	R_1^*	$= k$
7)		$\frac{1}{R} +$	k	$= \frac{1}{R}$
8)		$\overline{1} \to$	$\lambda_{раб}$	$= \lambda_{раб}$
9)	$PA < \overline{23}$			$\overline{1}^*$
10)		$U \cdot$	U	$= k$
11)		$k \cdot$	$\frac{1}{R}$	$= W$
12)	*стоп.*			

В задачах с более сложными логическими условиями, чем предыдущая, шкалу делят на группы из нескольких разрядов. Каждая такая группа содержит информацию о тех или иных условиях решения задачи.

Пример 2.38. Водопроводная сеть состоит из 15 участков. На каждом участке имеется кран, который может занимать четыре положения: 0, 1, 2, 3. При положении 0 (кран закрыт) вода на участок

не поступает (расход $q_0 = 0$), при положении 1 — поступает q_1 литров воды, при положении 2 — q_2 литров, при положении 3 — q_3 литров. Определить полный расход воды в сети при произвольном положении всех 15 кранов?

Положение первого крана охарактеризуем двухразрядным двоичным числом *(диадой)* μ_1, положение второго — числом μ_2, ..., положение пятнадцатого — числом μ_{15}.

Из этих чисел составим шкалу μ, поместив μ_1 — в 1 и 2-й разряды, μ_2 — в 3 и 4-й разряды, ..., μ_{15} — в 29 — 30-й разряды. Разряды с 31-го по 45-й оставим свободными.

Полный расход воды

$$Q = q_{\mu_1} + q_{\mu_2} + \ldots + q_{\mu_{15}}.$$

Для подсчета Q применим следующий способ: будем извлекать из шкалы μ пару разрядов μ_1, засылать это число в PA (для чего его нужно сдвинуть из третьего адреса во второй), и сформируем при помощи PA команду сложения Q с q_{μ_1}:

1)		0	$= Q$
2)	μ	$\bigwedge (0,\ 0,\ 3)$	$= k$
3)	$\overline{14}\leftarrow,$	k	$= k$
4)	$[PA]$	k	
5)	$Q +$	q_0^*	$= Q.$

Если теперь сдвинуть шкалу на два разряда вправо и повторить те же операции 2), 3), 4), 5), то получим в Q сумму $q_{\mu_1} + q_{\mu_2}$.

Цикл получения Q записывается в следующем виде:

		0	$= Q$
		μ	$= \mu_{раб}$
	$\mu_{раб}$	$\bigwedge (0,\ 0,\ 3)$	$= k$
	$\overline{14}\leftarrow,$	k	$= k$
$[PA]$		k	
	$Q +$	q_0^*	$= Q$
	$\overline{2}\ \rightarrow$	$\mu_{раб}$	$= \mu_{раб}$
	y_0		

стоп.

§ 39. Программа перевода числа из десятичной системы в двоичную

Как уже говорилось в гл. V, десятичные числа, введенные в машину в двоично-десятичном виде, перед счетом должны быть переведены в двоичную систему счисления. Здесь мы рассмотрим программу, осуществляющую такой перевод.

Десятичное число α представляется перед вводом в машину в виде

$$\alpha_{10} = M \cdot 10^p, \tag{1.39}$$

где p — десятичный порядок, а M — десятичная мантисса, причем

$$-19 \leqslant p \leqslant +19.$$

Модуль десятичной мантиссы в ячейке памяти записывается в двоично-десятичном виде, т. е. последовательностью девяти тетрад:

$$M = \tau_8\tau_7\tau_6\tau_5\tau_4\tau_3\tau_2\tau_1\tau_0,$$

а модуль порядка в виде монады и тетрады

$$p = m_p\tau_p.$$

Мантисса M является правильной девятизначной десятичной дробью. Для того чтобы облегчить перевод $\alpha_{(10)}$ в $\alpha_{(2)}$, умножим M на 10^9. Тогда мы получим целое десятичное число

$$N_{10} = 10^9 M. \tag{2.39}$$

Заменяя в формуле (1.39) M на N, получим

$$\alpha_{10} = 10^{p-9} N_{10}. \tag{3.39}$$

Алгоритм перевода $\alpha_{(10)}$ в двоичную форму $\alpha_{(2)}$ заключается в следующем: сначала переводят из двоично-десятичной формы в двоичную число $|N_{(10)}|$, т. е. мантиссу M, рассматриваемую как целое девятиразрядное десятичное число с плавающей запятой. Этот перевод осуществляется по формуле:

$$N_{(2)} = \pm (10^8\tau_8 + 10^7\tau_7 + \ \ldots\ + 10^1\tau_1 + 10^0\tau_0), \tag{4.39}$$

причем все действия должны выполняться в двоичной системе счисления.

При известном $N_{(2)}$ искомое $\alpha_{(2)}$, согласно (3.39), определяется по формуле

$$\alpha_{(2)} = 10^{p-9} N_{(2)}. \tag{5.39}$$

Если $p > 9$, то для получения $\alpha_{(2)}$ следует $N_{(2)}$ помножить $p - 9$ раз на число «10», записанное в двоичной системе. Если же $p \leqslant 9$, то $N_{(2)}$ следует $9 - p$ раз помножить на «0,1». Поэтому формулу (5.39) для удобства счета следует представить в виде

$$\alpha_{(2)} = k^\rho N_{(2)}, \tag{6.39}$$

где

$$k = \begin{cases} 10, & \text{если } p > 9, \\ 0{,}1, & \text{если } p \leqslant 9, \end{cases} \tag{7.39}$$

и

$$\rho = |p - 9|.$$

В машине нет возможности складывать и вычитать порядки с учетом их знака. Поэтому ρ приходится подсчитывать следующим образом:

$$\rho = \begin{cases} p-9, & \text{если } p>9, \\ 9-p, & \text{если } 0\leqslant p\leqslant 9, \\ |p|+9, & \text{если } p<0. \end{cases} \tag{8.39}$$

Программа перевода числа α из двоично-десятичной системы в двоичную состоит из следующих трех частей (блоков):

А) Перевод мантиссы M как целого числа из двоично-десятичной системы в двоичную по формуле (4.39).

Б) Перевод порядка p из двоично-десятичной системы в двоичную, получение ρ и k по формулам (7.39), (8.39).

В) Вычисление $\alpha_{(2)}$ по формуле (6.39).

Будем считать, что исходное число α_{10} находится в ячейке α; $\alpha_{(2)}$ будем получать в ячейке γ.

Блок А

Вычисление целого числа $N_{(2)}$ мы будем вести по формуле (4.39), преобразованной к виду

$$N_{(2)} = \pm [\tau_0 + 10(\tau_1 + 10(\tau_2 + 10(\tau_3 + 10(\tau_4 + 10(\tau_5 + 10(\tau_6 + 10(\tau_7 + 10\tau_8)\ \ldots)\ldots)]. \tag{9.39}$$

Для счета по этой формуле нужно превратить последовательные тетрады $\tau_8, \tau_7, \ldots, \tau_0$ в целые плавающие двоичные числа. Заметим, что если все эти числа будут иметь знак числа α, то предыдущая формула даст $N_{(2)}$ с нужным знаком. Выделим прежде всего из числа α мантиссу вместе со знаком *):

$$1)'\ \alpha \wedge (200, F, F, F) = \alpha_{\text{зап}}.$$

Старшая тетрада, как целое число, должна иметь машинный порядок 104. Присоединим этот порядок к $\alpha_{\text{зап}}$ и отделим затем τ_8 от остальных тетрад:

$$2)'\ \alpha_{\text{зап}} \vee (104, 0, 0, 0) = \alpha_{\text{зап}},$$

$$3)'\ \alpha_{\text{зап}} \wedge (377, 7400, 0, 0) = R.$$

В ячейке R мы получили целое число τ_8 со знаком α. Для получения целого числа τ_7 сдвинем мантиссу $\alpha_{\text{зап}}$ на четыре разряда влево:

$$4)'\ \overline{4} \leftarrow, \alpha_{\text{зап}} = \alpha_{\text{зап}}.$$

Теперь тетрада τ_7 заняла место τ_8 и, выполняя вновь команды 2)' и 3)', мы получим в ячейке R целое число τ_7 со знаком α. Тем же способом можно получить и целые числа $\tau_6, \tau_5, \ldots, \tau_0$.

Вычисления по формуле (9.39) будем выполнять в следующем порядке:

а) зашлем в ячейку результата γ нуль и прибавим к нему целое число τ_8, полученное указанным способом;

б) величину γ умножим на «10» и прибавим к результату целое число τ_7;

в) пункт б) повторим еще семь раз, последовательно заменяя τ_7 на τ_6, τ_6 на τ_5, ..., τ_1 на τ_0.

---

*) Напомним еще раз, что буква F в адресе означает число 7777_8.

Очевидно, этот процесс носит циклический характер, причем число шагов цикла равно девяти. Программа расчета $N_{(2)}$ имеет такой вид:

1)	α	$\wedge$ (200, F, F, F)	$=\alpha_{зап}$
2)	$\alpha_{зап}$	$\vee$ (104, 0, 0, 0)	$=\alpha_{зап}$
3)		0	$=\gamma$
4)	$Б$	e_{43}	Сч
5)		«10» $\cdot$	γ $=\gamma$
6)		$\overline{4}$ $\leftarrow$,	$\alpha_{зап}$ $=\alpha_{зап}$
7)		$\alpha_{зап}$ $\wedge$ (377, 7400, 0, 0)	$=R$
8)		γ $+$	R $=\gamma$
9)		$\overline{5}$ $\rightarrow$ Сч	$=$ Сч
10)	$У_0$.		

Рассмотрим, как работает эта программа. Счетчик цикла здесь изменяется при помощи операции сдвига 9). Сначала он имеет единицу в 43-м разряде (команда 4), после первого шага цикла он превращается в e_{38}, после второго — в e_{33}, ..., после восьмого в e_3; при этом каждый раз команда 10) вырабатывает сигнал $\omega = 0$, и шаги цикла повторяются. После девятого шага цикла команда 9) превращает счетчик в нуль, вырабатывается сигнал $\omega = 1$, и мы выходим из цикла.

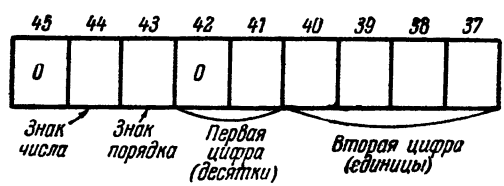


Рис. 35.

Таким образом, рабочая часть цикла повторяется 9 раз.

Вычисления в цикле производятся так. Сначала в γ засылается нуль (команда 3)). Затем на первом шаге цикла при помощи команд 7), 8) в γ засылается целое число τ_8. На втором шаге цикла γ, равное τ_8, помножается на «10» (команда 5)), затем по команде 6) все тетрады мантиссы сдвигаются на четыре разряда влево, таким образом, место тетрады τ_8 занимает τ_7, место $\tau_7 - \tau_6$, ..., место τ_1 занимает τ_0. Затем при помощи 7) образуется целое число τ_7, которое по команде 8) прибавляется к γ. Итак, в ячейке γ оказывается число

$$\pm(\tau_7 + 10\tau_8).$$

Легко убедиться в том, что после девяти шагов цикла в ячейке γ окажется число $N_{(2)}$, определенное формулой (9.39).

Блок Б

Десятичный порядок p изображается в ячейке машины так, как показано на рис. 35.

Чтобы образовать величину p, следует произвести ряд операций над десятичным порядком p. В машине предусмотрены лишь операции над порядками, представленными в двоичном виде. Поэтому следует привести двоично-десятичный порядок

$$p = m_p\tau_p$$

к двоичному виду $p_{(2)}$. Очевидно,

$$p_{(2)} = 10m_p + \tau_p.$$

Для вычисления $p_{(2)}$ следует извлечь из α тетраду τ_p порядка

11) $\alpha \wedge (17, 0, 0, 0) = p,$

затем посмотреть, находится ли в 41-м разряде α (монада m_p) нуль или единица, и если $m_p = 1$, к p прибавить $10_{10} = 12_8$:

12) $\alpha \wedge e_{41}$ $= 0$
13) $У_1$ 15)
14) $p, + (12, 0, 0, 0) = p.$

Теперь десятичный порядок приведен к двоичному виду.

Приступим к нахождению ρ и k. Величину ρ будем образовывать в разрядах порядка. Нужное нам число 9, помещенное в восьмеричном виде в порядок, имеет вид (11, 0, 0, 0).

При вычислении ρ и k по формулам (8.39) и (7.39) мы будем проверять в нижеследующей программе два условия: $p < 0$ и $p > 9$:

15) $\alpha \wedge e_{43}$ $= 0$
16) $У_0$ «0,1» 21) k
17) (11, 0, 0, 0), $-p$ $= \rho$
18) $У_0$ 22)
19) $p,$ $-(11, 0, 0, 0) = \rho$
20) $Б$ «10» 22) k
21) $p,$ $+(11, 0, 0, 0) = \rho.$

Рассмотрим, как работает эта программа в трех возможных случаях:

I) $p < 0$.

В этом случае в 43-м разряде ячейки p стоит единица и команда 15) вырабатывает сигнал $\omega = 0$. По команде 16) в k засылается 0,1 и передается управление на команду 21), в результате выполнения которой в порядке ячейки ρ получаем величину $|p| + 9$.

II) $0 \leqslant p \leqslant 9$.

В этом случае после команды 15) будем иметь $\omega = 1$. По команде 16) в k засылается 0,1 и происходит переход к команде 17), в порядке ячейки ρ образуется величина $9 - p$, ω оказывается равным нулю и по команде 18) мы выходим из программы.

III) $p > 9$.

Команды 15) — 17) выполняются как и во втором случае. В результате в k оказывается 0,1, а в порядке ρ отрицательная величина $9 - p$. Поэтому после команды 17) имеем $\omega = 1$ и команда 18) передает управление на 19). Затем в порядке ρ образуется $p - 9$ и в k засылается число 10.

Таким образом, величины ρ и k во всех трех случаях образуются в полном соответствии с формулами (8.39) и (7.39).

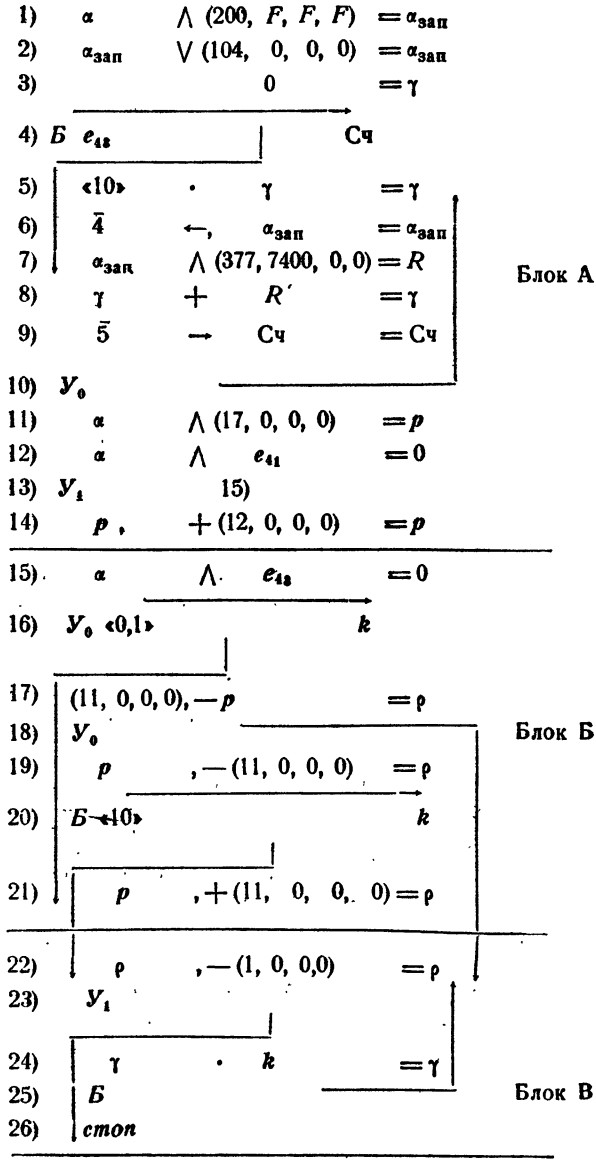
Блок В

Теперь нам остается лишь получить двоичное число $\alpha_{(2)}$ по формуле (6.39).

Этот, самый простой, блок программы имеет вид

22) ρ , $-(1, 0, 0, 0) = \rho$
23) $У_1$
24) $\gamma \cdot k$ $= \gamma$
25) $Б$
26) *стоп.*

Объединяя три составленных блока и приписывая к ним нужные константы, получим программу перевода двоично-десятичного числа α в двоичную форму γ:



27) (200, F, F, F)
28) (104, 0, 0, 0)
29) (377, 7400, 0, 0)
30) (17, 0, 0, 0)
31) (12, 0, 0, 0) Константы
32) (11, 0, 0, 0)
33) $e_{37} = (1, 0, 0, 0)$
34) $e_{41} = (20, 0, 0, 0)$
35) $e_{43} = (100, 0, 0, 0)$
36) «10»
37) «0,1».

§ 40. Программа перевода числа из двоичной системы в десятичную. Операция печати

Для выдачи на печать результатов вычислений необходимо перевести эти результаты из двоичной системы счисления в десятичную. Этот перевод осуществляется в два этапа:

1) при помощи специальной программы, которую мы рассмотрим в этом параграфе, двоичные числа переводятся в двоично-десятичные;

2) печатающее устройство машины превращает двоично-десятичные слова в обычные десятичные числа.

Двоичное число α записано в машине в виде

$$\alpha_{(2)} = M \cdot 2^p = 2^{-64} M \cdot 2^{p'}, \tag{1.40}$$

где $p' = p + 64_{10}$ — машинный порядок, p — действительный порядок, а M — двоичная мантисса числа.

Число α, переведенное в двоично-десятичную форму, запишем в виде

$$\gamma = \alpha_{10} = \pm 10^q \cdot N, \tag{2.40}$$

где q — десятичный порядок, а N — модуль десятичной мантиссы, причем

$$0{,}1 \leqslant N < 1.$$

Алгоритм программы превращения $\alpha_{(2)}$ в γ состоит из двух основных блоков:

А) получение порядковой части числа γ, т. е. знака числа, знака и величины десятичного порядка q;

Б) нахождение десятичной мантиссы.

Познакомимся с программами этих блоков.

Блок А

При получении q следует различать два случая: $p > 0$ и $p \leqslant 0$.

Если $p > 0$, то $|\alpha| \geqslant 1$. Для того чтобы найти десятичный порядок q, следует делить α на число «10», представленное в двоичном виде, до тех пор, пока модуль частного не станет меньше единицы. Подсчитав число таких делений, получим q, а последнее частное окажется равным N, причем числа q и N будут представлены в двоичном виде. Если же $p \leqslant 0$, то $|\alpha| < 1$. В этом случае порядок q можно найти, подсчитав число умножений на «10», которое нужно произвести с α, чтобы получить произведение по модулю, большее 0,1. Последнее произведение окажется при этом равным N.

Приступая к составлению программы, очистим прежде всего ячейку q, в которой будет образовываться десятичный порядок.

1) $0 = q.$

В соответствии с принятым алгоритмом, составляем программу получения q с разветвлением на два случая: $p > 0$ и $p \leqslant 0$;

2) $\alpha \cdot$ «1» $= N$

3) $У_0$

4) $q, +(1, 0, 0, 0) = q$

5) $N : $ «10» $= N$

6) $У_1$

7) $Б$ 0 13) $ЗП$

8) $q, +(1, 0, 0, 0) = q$

9) $N \cdot$ «10» $= N$

10) $|N| -$ $|$«0,1»$| = 0$

11) $У_1$

12) $e_{48} = ЗП.$

В случае $p > 0$ команда 1) засылает α в N и вырабатывает сигнал $\omega = 1$, вследствие чего команда 3) передает управление на 4). После этого выполнение в цикле команд 4) — 6) обеспечивает формирование в ячейках q и N десятичного порядка и десятичной мантиссы в двоичном виде. Команда 7), передавая управление на продолжение счета, пересылает в ячейку $ЗП$ (*знак порядка*) нулевой знак порядка.

Пусть теперь $p \leqslant 0$, т. е. $|\alpha| < 1$. Тогда команда 3) передаст управление на 10). Здесь прежде всего проверяется условие

$$|N| \geqslant \frac{1}{10}.$$

Если оно выполнено, т. е. если

$$\frac{1}{10} \leqslant |N| < 1, \tag{3.40}$$

то управление передается на 12); в этом случае q должно остаться равным нулю.

Если же условие (3.40) не выполнено, то после команды 10) вырабатывается сигнал $\omega = 1$ и команда 11) передает управление на 8). Затем команды 8) — 11) выполняются в цикле до тех пор, пока не выполнится неравенство (3.40).

В результате в ячейках q и N, так же как и в первом случае, образуются десятичные порядок и мантисса, представленные пока еще в двоичном виде.

Для перехода от двоичного порядка q к двоично-десятичному следует узнать, больше ли q, чем 9, и если больше, то образовать монаду и тетраду:

13) $q, -(12, 0, 0, 0) = R$

14) $У_1$ 16)

15) $R, +(20, 0, 0, 0) = q.$

Если $q > 9$, то команда 13) вычитает из q число $12_8 = 10_{10}$ и вырабатывает сигнал $\omega = 0$. В ячейке q оказывается нужная тетрада, а монаду к этой тетраде присоединяет команда 15). Если же $q \leqslant 9$, то в ячейке q уже лежит нужная тетрада.

Для получения порядковой части десятичного числа γ нам остается присоединить к q знак порядка и присвоить знак числа α. Это осуществляется следующими тремя командами:

16) $q \vee ЗП = \gamma$
17) $\alpha \wedge e_{44} = R$
18) $\gamma \vee R = \gamma$,

после которых работа блока A заканчивается.

Блок Б

Мы получили в ячейке γ порядковую часть десятичного числа. Остается найти десятичную мантиссу.

Полученное выше двоичное число N, удовлетворяющее неравенству (3.40), равно мантиссе десятичного числа γ. Для получения из этого числа десятичных цифр-тетрад следует поступать так (см. § 12): умножим N на число «десять», записанное в двоичном виде. Целая часть полученного числа оказывается равной старшей цифре мантиссы.

Умножая дробную часть на десять и выделяя из полученного произведения целую часть, найдем следующую цифру мантиссы. Повторяя эту операцию девять раз, мы получим все нужные цифры десятичной мантиссы. Если бы нам нужны были только эти цифры, а не вся мантисса, можно было бы образовать цикл их получения таким образом:

19)′ $e_{43} = Сч$
20)′ $N \cdot$ «10» $= N$
21)′ $0+, N = R$
22)′ $N[\rightarrow] R = N$
23)′ $\overline{44} \rightarrow N = \tau$
24)′ $e_{43}+, N = N$
25)′ $\overline{5} \rightarrow Сч = Сч$
26)′ $У_0$.

Отметим прежде всего, что здесь команды 19)′, 25)′, 26)′ обеспечивают работу цикла девять раз тем же способом, что и в блоке А программы предыдущего параграфа. Сделаем теперь пояснения к рабочей части цикла. После умножения N на «10» (команда 20)′), в N выделяется мантисса (команда 21)′), а затем при помощи 22)′ и 23)′ целая часть N передвигается в порядок и сдвигается в хвост ячейки τ. Таким образом, ячейка τ содержит в младших разрядах тетраду искомой десятичной мантиссы. Затем при помощи 24)′ от N отделяется мантисса дробной части, к которой приписывается машинный порядок 100 (т. е. действительный нулевой порядок).

На втором шаге цикла дробная часть N умножается на «10» и тем же способом выделяется вторая слева цифра (тетрада) десятичной мантиссы. Выделение остальных семи цифр мантиссы происходит аналогичным способом.

Однако рассмотренный цикл еще не решает поставленную задачу, ибо нам нужны не отдельные тетрады, а сама мантисса, составленная из этих тетрад. Для того чтобы сформировать мантиссу, вставим в предыдущий цикл команды, обеспечивающие сдвиг тетрад, получаемых в ячейке τ, на

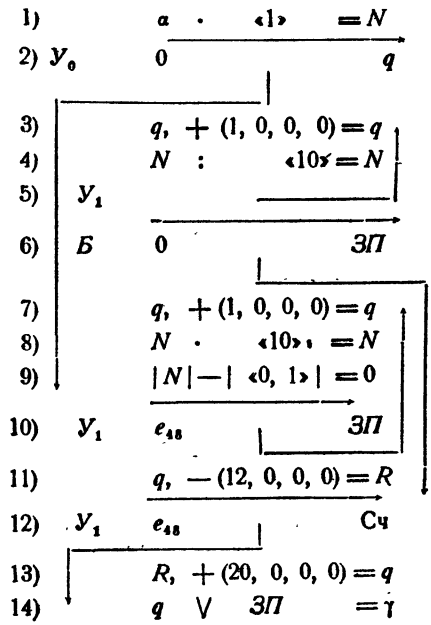
нужные места и объединяющие тетрады в одно целое:

19) $e_{48} = \text{Сч}$
20) $\overline{4} \leftarrow , \quad \gamma = \gamma$
21) $N \cdot \quad \text{«10»} = N$
22) $0 + , \quad N = R$
23) $N[\rightarrow] \quad R = N$
24) $\overline{44} \rightarrow \quad N = \tau$
25) $\gamma \vee \quad \tau = \gamma$
26) $e_{43} + , \quad N = N$
27) $\overline{5} \rightarrow \quad \text{Сч} = \text{Сч}$
28) $У_0 \quad 20).$

Как мы уже говорили, команды 21) — 24) обеспечивают формирование в младших разрядах ячейки τ последовательных тетрад десятичной мантиссы. На первом шаге цикла тетрада τ_8 при помощи команды 25) приформировывается к γ, а затем в цикле при помощи команды 20) сдвигается каждый раз на четыре разряда (при восьми сдвигах — на 32_{10} разряда), т. е. ставится на нужное место. На втором шаге цикла в младшие разряды γ по команде 25) ставится следующая тетрада τ_7, которая затем в цикле при помощи команды 20) сдвигается на $4 \cdot 7 = 28$ разрядов. Таким же образом приформировываются к γ остальные тетрады.

Объединяя блоки А и Б, мы совместим пересылки 1), 12), 19) с передачами управления 3), 11), 14).

Выписав затем необходимые константы, мы получим программу перевода двоичного числа α в двоично-десятичную форму γ:



Блок А

```
15)        α  ∧  e44      = R
16)        γ  ∨  R        = γ
-------------------------------------------
17)        4̄ ← , γ        = γ
18)        N ·   «10»     = N
19)        0 + , N        = R
20)        N[→]  R        = N
21)        44̄ →  N        = τ
22)        γ  ∨  τ        = γ
23)        e48 + , N      = N
24)        5̄ →   Сч       = Сч
25) Y0
26)    стоп
-------------------------------------------
27)        e48 = (100, 0, 0, 0)
28)        e44 = (200, 0, 0, 0)
29)              (20, 0, 0, 0)
30)              (12, 0, 0, 0)
31)              (1, 0, 0, 0)
32)              «10»
33)              «0,1»
34)              «1».
```

Блок Б

Константы

Полученное на этой программе число γ может быть напечатано в десятичном виде. Чтобы это сделать, следует заменить команду *стоп* программы перевода $2 \rightarrow 10$ командой печати, имеющей вид *)

Печать, γ, 0, 0,

где *Печать* — буквенное обозначение кода операции печати (восьмеричный код 30).

По этой команде число из ячейки памяти γ передается на печатающее устройство и печатается на бумажной ленте в десятичном виде, т. е. каждая тетрада воспроизводится как соответствующая десятичная цифра. В том случае, когда тетрада не имеет десятичного смысла, печать дает пропуск.

*) В действительности на конкретной трехадресной машине операция печати имеет несколько иной вид, но мы для простоты будем пользоваться такой операцией.

ГЛАВА VIII

ПОДПРОГРАММЫ

§ 41. Операция безусловной передачи управления с возвратом. Блоки и подпрограммы

Во многих задачах, решаемых на машине, приходится неоднократно вычислять одни и те же функции. Приведем простейший пример такого рода.

Пример 1.41. При заданных значениях x, y, z вычислить величину

$$w=\frac{2x^3-5x^2+6x-4}{2y^3-5y^2+6y-4}-9(2z^3-5z^2+6z-4).$$

Кубический многочлен в числителе запишем так $[(2x-5)x+6]x-4$. Аналогично переписываются и два других многочлена.

Программа вычисления w, написанная «в лоб», имеет такой вид:

1) «2» · x = R_1
2) R_1 — «5» = R_1
3) R_1 · x = R_1
4) R_1 + «6» = R_1
5) R_1 · x = R_1
6) R_1 — «4» = R_1
7) «2» · y = R_2
8) R_2 — «5» = R_2
9) R_2 · y = R_2
10) R_2 + «6» = R_2
11) R_2 · y = R_2
12) R_2 — «4» = R_2
13) «2» · z = R_3
14) R_3 — «5» = R_3
15) R_3 · z = R_3
16) R_3 + «6» = R_3

17) $R_3 \cdot z = R_3$
18) $R_3 - \text{«4»} = R_3$
19) $R_1 : R_2 = R_1$
20) $\text{«9»} \cdot R_3 = R_3$
21) $R_1 - R_3 = w$
22) *стоп.*

Легко понять, что вычисление w запрограммировано здесь очень нецелесообразно. При программировании мы не воспользовались тем, что в формулу для w входят три значения одного и того же кубического многочлена:

$$Q(\alpha) = 2\alpha^3 - 5\alpha^2 + 6\alpha - 4.$$

Формулу для w можно записать так:

$$w = \frac{Q(x)}{Q(y)} - 9Q(z).$$

Из этой записи видно, что программирование счета w естественно вести следующим образом: отдельно выписать программу расчета $Q(\alpha)$; затем, обращаясь к этой программе, при $\alpha = x$ получить числитель, при $\alpha = y$ — знаменатель; после этого подсчитать дробь $\frac{Q(x)}{Q(y)}$, и, обращаясь к программе $Q(\alpha)$ при $\alpha = z$, найти $Q(z)$; наконец, получить по последней формуле w.

Напишем прежде всего программу вычисления $Q(\alpha)$:

Счет Q

Q $\text{«2»} \cdot \alpha = R_1$
$R_1 - \text{«5»} = R_1$
$R_1 \cdot \alpha = R_1$
$R_1 + \text{«6»} = R_1$
$R_1 \cdot \alpha = R_1$
$R_1 - \text{«4»} = \gamma$
к. Q н. п.

Здесь через γ обозначена ячейка результата. Первая команда программы помечена буквой Q, последняя, конечная — буквами к. Q (как мы увидим дальше, это свободная ячейка нужна для организации счета w).

Для того чтобы получить $Q(x)$, зашлем x во входную ячейку α программы *Счет Q* и передадим управление на начало этой программы:

Счет w

1) $x = \alpha$
2) Б Q.

Тогда после выполнения шести команд программы *Счет Q* в ячейке γ окажется величиной $Q(x)$.

Для продолжения вычислений необходимо, чтобы после окончания работы программы счета Q управление передавалось следующей команде 3) нашей основной программы. Это можно обеспечить, заслав в ячейку к. Q команду безусловной передачи управления, которую нужно еще заготовить. Однако этим дело не ограничится, потому что к программе счета Q нам придется обращаться еще дважды и всякий раз при таком обращении необходимо обеспечивать возврат из программы счета Q в различные места основной программы.

Возможность возврата в нужное место основной программы можно обеспечить с помощью специальной команды *безусловной передачи управления с возвратом*. Эту команду мы будем записывать в виде

$$БВ \quad a \; b \; c.$$

Выполняется она так: управление, как и при выполнении команды $Б$, передается ячейке b. Одновременно в ячейку c заносится не содержимое ячейки a, как при выполнении команды $Б$, а *команда передачи управления ячейке a*

$$БВ \quad 0 \; a \; 0.$$

Таким образом, если при обращении к программе счета Q воспользоваться вместо команды 2) этой командой в виде

$$БВ \quad 3) \quad Q \quad к.\,Q,$$

то одновременно с передачей управления ячейке Q в ячейку к. Q запишется команда передачи управления команде 3) основной программы, т. е. возврат в основную программу будет обеспечен.

Нужную нам программу можно теперь записать таким образом:

Счет w

1) $x = a$
2) *БВ* 3) Q к. Q
3) $\gamma = R_0$
4) $y = a$
5) *БВ* 6) Q к. Q
6) $R_0 : \gamma = R_0$
7) $z = a$
8) *БВ* 9) Q к. Q
9) «9» $\cdot\ \gamma = R_2$
10) $R_0 - R_2 = w$
11) *стоп.*

Программа вычисления w состоит из написанных здесь 11 команд и 7 команд, служащих для счета Q. Программа счета Q называется *блоком*, а команды 2), 5), 8) — *командами обращения к блоку*.

Блоки, к которым приходится обращаться из нескольких мест программы, мы будем называть *подпрограммами*.

Из рассмотренного примера видно, что выделение счета $Q(\alpha)$ в отдельную подпрограмму позволило сократить программу. Экономия памяти при выделении в подпрограммы счета функций, которые приходится вычислять в десятках мест программы, окажется, конечно, еще более существенной. Не менее существенным является еще то обстоятельство, что выделение часто встречающихся функций в подпрограммы не только экономит память, но и дает возможность записывать программу более естественно и наглядно.

При обращении к блоку управление передается началу блока, а команда возврата засылается в его конец, т. е. обращение происходит по команде

$БВ$ a Начало блока Конец блока

Эта команда дает возможность возвратиться к любой команде a. Однако в большинстве случаев возврат осуществляется на следующую команду основной программы.

Для упрощения записи таких команд введем следующее условие. Обозначим через $Я$ адрес команды обращения к блоку, в которой находится управление в данный момент. Тогда $Я+1$ будет адресом следующей команды и обращение к блоку запишется так:

$БВ$ $Я+1$ Начало блока Конец блока

Команды 2), 5), 8) предыдущей программы запишутся при таком условии одинаково:

$БВ$ $Я+1$ Q к.Q.

При этом для команды 2) $Я+1$ будет иметь значение 3), для команды 5) — значение 6), а для команды 8) — значение 9).

В табл. 1.41 приведена программа счета w в содержательных обозначениях и закодированном виде. В левой части снята ненужная здесь нумерация команд. Операция $БВ$ имеет код 16. При кодировке за величинами, входящими в программу, закреплены следующие ячейки:

x	1231	α	0140
y	1232	γ	0160
z	1233	«2»	0102
w	1234	«4»	0104
R_0	0010	«5»	0105
R_1	0011	«6»	0106
R_2	0012	«9»	0111

Таблица 1.41

	Счет w				1200	A	
	$x=\alpha$	1200	0	75	0000	1231	0140
	БВ *Я*+1 *Q* к. *Q*	1	0	16	1202	1220	1226
	$\gamma=R_0$	2	0	75	0000	0160	0010
	$y=\alpha$	3	0	75	0000	1232	0140
	БВ *Я*+1 *Q* к. *Q*	4	0	16	1205	1220	1226
	$R_0 : \gamma=R_0$	5	0	04	0010	0160	0010
	$z=\alpha$	6	0	75	0000	1233	0140
	БВ *Я*+1 *Q* к. *Q*	7	0	16	1210	1220	1226
	«9» $\cdot \gamma=R_2$	1210	0	05	0111	0160	0012
	$R_0-R_2=w$	1	0	02	0010	0012	1234
	стоп	2	0	17	0000	0000	0000

	Счет Q				1220	A	
Q	«2» $\cdot \alpha = R_1$	1220	0	05	0102	0140	0011
	R_1-«5»$=R_1$	1	0	02	0011	0105	0011
	$R_1 \cdot \alpha = R_1$	2	0	05	0011	0140	0011
	R_1+«6»$=R_1$	3	0	01	0011	0106	0011
	$R_1 \cdot \alpha = R_1$	4	0	05	0011	0140	0011
	R_1-«4»$=\gamma$	5	0	02	0011	0104	0160
к. Q	н. п.	6				н. п.	

§ 42. Стандартные подпрограммы с входными и выходными ячейками

В процессе решения каждой конкретной задачи математик выделяет часто встречающиеся ему функции в «свои» подпрограммы. Однако есть функции, которые встречаются при решении самых разнообразных задач. Так, например, во многих задачах приходится иметь дело с тригонометрическими функциями. Естественно для таких функций раз и навсегда написать *стандартные подпрограммы*, которыми сможет воспользоваться любой программист. Правила обращения к стандартной подпрограмме должны быть естественными и простыми, а следовательно, легко запоминаемыми.

Все стандартные подпрограммы по способу обращения к ним делятся на две группы: *подпрограммы с входными и выходными ячейками* и *подпрограммы с информацией*.

Подпрограммы первой группы служат для вычисления функций одного или двух независимых переменных, например тригонометрических, показательной, показательно-степенной и др. Покажем, как составляются такие подпрограммы на примере.

Пример 1.42. Составить стандартную подпрограмму вычисления кубического корня $y=\sqrt[3]{x}$.

Программа должна удовлетворять следующему требованию. Если в некоторую *стандартную входную ячейку* α заслать аргумент

$$x=\alpha$$

и обратиться к подпрограмме

$$БВ \quad Я+1 \quad \sqrt[3]{\ } \quad к.\sqrt[3]{},$$

то после того как работа этой программы закончится и управление будет передано ячейке $Я+1$, в *стандартной выходной ячейке* γ окажется значение кубического корня из α. Это значение должно быть вычислено с машинной точностью. Таким образом, для вычисления $y=\sqrt[3]{x}$ нужно написать три команды:

$$\begin{array}{llll} & x & =\alpha & \\ БВ & Я+1 & \sqrt[3]{\ } & к.\sqrt[3]{} \\ & \gamma & =y & \end{array}$$

Здесь $\sqrt[3]{\ }$ — метка начальной ячейки подпрограммы, а к. $\sqrt[3]{}$ — метка конечной ее ячейки (ячейки выхода). Ранее нами была составлена программа вычисления кубического корня (см. пример 2.21). Чтобы превратить эту программу в стандартную, поставим на место команды *стоп* свободную ячейку к. $\sqrt[3]{\ }$, заменим обозначения x на α, y на γ и организуем проверку окончания счета на машинную точность. Тогда

получится следующая программа:

$$
\begin{array}{lll}
\sqrt[3]{} & \alpha \cdot «\tfrac{1}{3}» = R_0 & \\
 & R_0 \cdot R_0 = R_1 & \\
 & \alpha : R_1 = R_1 & \\
 & R_1 \cdot «\tfrac{1}{3}» = R_1 & \\
 & R_0 \cdot «\tfrac{2}{3}» = R_2 & \\
 & R_1 + R_2 = \gamma & \\
 & \gamma \nsim R_0 = 0 & \\
 & \xrightarrow{\qquad\qquad} & \\
У_0 & \gamma \quad | \quad R_0 & \\
\text{к. } \sqrt[3]{} & \text{н. п.} &
\end{array}
$$

При обращении к программе

$$БВ \quad Я+1 \quad \sqrt[3]{} \quad \text{к. } \sqrt[3]{}$$

в ячейке γ вычисляется кубический корень из числа, находящегося в ячейке α.

Это обращение имеет то неудобство, что для кодировки команды обращения надо знать два адреса: адрес начала $\sqrt[3]{}$ и адрес конца к. $\sqrt[3]{}$ стандартной программы. Можно устранить это неудобство, вводя для всех стандартных программ одну и ту же ячейку конца, обозначаемую буквой Ω.

Тогда обращение к подпрограмме примет вид

$$БВ \quad Я+1 \quad \sqrt[3]{} \quad Ω$$

и для того, чтобы обеспечить возврат к основной программе (на $Я+1$), следует в последнюю (ранее свободную) ячейку стандартной подпрограммы поставить команду

$$Б \quad Ω.$$

По тому же образцу, что и для подпрограммы $\sqrt[3]{}$, строятся обращения к другим подпрограммам, вычисляющим функции одного независимого переменного. Так, для вычисления $y = e^x$ следует обратиться к подпрограмме вычисления показательной функции следующим образом:

$$
\begin{array}{lll}
 & & x = \alpha \\
БВ & Я+1 \quad \exp & Ω \\
 & & \gamma = y.
\end{array}
$$

Аналогично находится $y = \ln x$

```
          x = α
БВ  Я+1   ln   Ω
          γ = y.
```

При использовании программ для нахождения кубического корня, показательной и логарифмической функций обычно не засылают аргумент в α, а при вычислении этого аргумента (если его нужно вычислять и не требуется запомнить) прямо образуют его в ячейке α, результат же не пересылают из γ, а используют в последующих вычислениях.

Пример 2.42. Вычислить $y = 5e^{2\sqrt[3]{x-1}}$. Здесь y легко находится по программе

```
   x     — «1» = α
БВ Я+1     ∛   Ω
  «2»     ·  γ = α
БВ Я+1    exp  Ω
  «5»     ·  γ = y
  стоп    ·
```

Мы рассмотрели обращения к трем стандартным подпрограммам: $\sqrt[3]{}$, exp, ln. Все эти подпрограммы, с точки зрения их использования в рабочих программах, построены одинаково. Каждая из них вычисляет функцию одного независимого переменного

$$\gamma = f(\alpha),$$

причем аргумент берется из стандартной входной ячейки α, а результат помещается в стандартную выходную ячейку γ. Обращение к подпрограмме имеет вид

$$БВ \quad Я+1 \quad f \quad Ω,$$

где $Ω$ — стандартная ячейка возврата, а f — метка начальной команды подпрограммы.

Заметим, впрочем, что, обращаясь к подпрограмме, можно возвратиться и не на следующую $Я+1$, а на любую команду S:

$$БВ \quad S \quad f \quad Ω.$$

Помимо указанных функций, в практике вычислений часто встречаются еще тригонометрические и обратные тригонометрические функции. Тригонометрические функции $\cos x$ и $\sin x$ обычно в формулах встречаются парами. Далее мы увидим, что и алгоритмы вычисления этих функций таковы, что удобно получать сразу пару функций $\cos\alpha$ и $\sin\alpha$ одного независимого переменного α. Поэтому для тригономе-

трических функций обычно используется одна подпрограмма вычисления $\cos\alpha$ и $\sin\alpha$, которая по аргументу, находящемуся в стандартной входной ячейке α, вычисляет в двух стандартных выходных ячейках γ_0 и γ_1 косинус и синус α, так что

$$\gamma_0 = \cos\alpha,$$
$$\gamma_1 = \sin\alpha$$

(в ячейке с четным индексом 0 получается четная функция $\cos\alpha$, в ячейке с нечетным индексом 1 — нечетная функция $\sin\alpha$).

Обращение к этой стандартной подпрограмме имеет вид

$$БВ \quad Я+1 \quad \cos,\ \sin \quad Ω,$$

где cos, sin — метка начальной ячейки подпрограммы. Точно так же пишутся обращения к стандартным подпрограммам, вычисляющим пары обратных тригонометрических функций:

$$БВ \quad Я+1 \quad \arccos,\ \arcsin \quad Ω$$

и

$$БВ \quad Я+1 \quad \operatorname{arcctg},\ \operatorname{arctg} \quad Ω,$$

причем в результате работы первой из них получаем

$$\gamma_0 = \arccos\alpha,$$
$$\gamma_1 = \arcsin\alpha,$$

а после работы второй

$$\gamma_0 = \operatorname{arcctg}\alpha,$$
$$\gamma_1 = \operatorname{arctg}\alpha.$$

Таким образом, все эти подпрограммы вычисляют пары функций одного аргумента

$$\gamma_0 = f_0(\alpha),$$
$$\gamma_1 = f_1(\alpha).$$

Обращение же к ним пишется в виде

$$БВ \quad Я+1 \quad f_0,\ f_1 \quad Ω.$$

Составление стандартных программ для вычисления элементарных функций будет рассмотрено в гл. XVI.

Пример 3.42. Вычислить величину

$$y = \sin x - 4\cos x + 3\operatorname{tg} 5x.$$

По определению, $\operatorname{tg} x = \frac{\sin x}{\cos x}$. Программа вычисления y имеет вид

		x	$=\alpha$
БВ *Я*+1		cos,sin	Ω
«4»	$\cdot$	γ_0	$=R$
γ_1	$-$	R	$=y$
«5»	$\cdot$	x	$=\alpha$
БВ *Я*+1		cos,sin	Ω
γ_1	:	γ_0	$=R$
«3»	$\cdot$	R	$=R$
y	$+$	R	$=y$
стоп.			

В некоторых задачах приходится вычислять функции двух переменных, например показательно-степенную функцию

$$z = x^y,$$

где x и y — действительные числа, причем $x > 0$.

Обращение к стандартной подпрограмме, вычисляющей эту функцию двух независимых переменных, строится следующим образом:

$$x = \alpha_0$$
$$y = \alpha_1$$
$$БВ \quad Я+1 \quad | \quad \Omega$$
$$\gamma = z.$$

Здесь | — метка начальной команды подпрограммы, вычисляющей в стандартной ячейке γ по величинам, заданным в стандартных ячейках α_0, α_1, величину

$$\gamma = \alpha_0^{\alpha_1}.$$

§ 43. Стандартные подпрограммы с информацией. Формирование команд

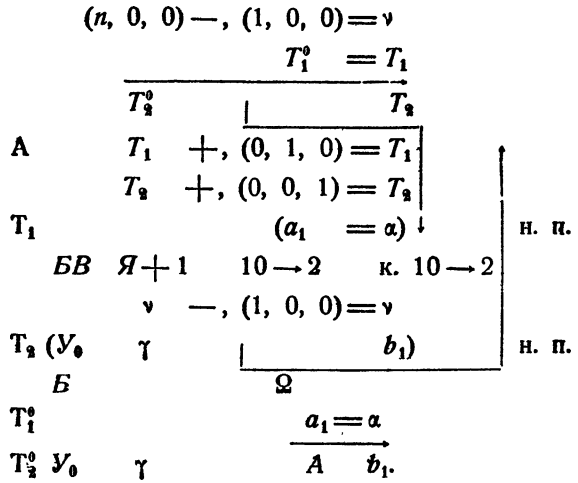
Программы с входными и выходными ячейками, рассмотренные в предыдущем параграфе, вполне удобны для использования, однако не всегда удается ими ограничиться. Чтобы в этом убедиться, рассмотрим сначала следующий пример.

Пример 1.43. Программа должна работать с массивом исходных числовых данных, введенных в память в двоично-десятичном виде в ячейки от a_1 до a_n подряд. Для работы программы необходимо перевести эти числа в двоичную систему и разместить их в ячейках памяти подряд, начиная с ячейки b_1.

В § 39 была рассмотрена программа «10 → 2» с входной ячейкой α и выходной γ, которая переводит десятичное число, лежащее в ячейке α, в двоичное число и помещает его в ячейку γ. Для превращения этой программы в стандартную достаточно заменить команду *стоп*.

Для удобства дальнейшего изложения оставим этой программе собственный конец. Тогда взамен команды *стоп* нужно оставить неперфорируемую ячейку, которую мы будем обозначать к. 10 → 2.

Напишем теперь блок, который обеспечит перевод всех чисел заданного массива в двоичную систему и размещение их в нужных местах. Это будет обычный цикл с переадресацией, содержащий обращение к программе 10 → 2.



Таким образом, нам пришлось написать блок, занимающий 13 команд. Ясно, что этот блок является необходимым для любой задачи, работающей с массивом входных данных. Но можно ли сделать так, чтобы *этот блок был стандартным* и чтобы каждому программисту не пришлось писать его для себя заново?

В таком виде, как этот блок написан, его можно считать стандартным только в том случае, если ячейки a_1, b_1 и $(n, 0, 0)$ известны заранее и зафиксированы, т. е. тоже являются стандартными. Но тогда программисту все равно придется пересылать числа из своих ячеек в ячейки $a_1, \ldots, a_n$, а затем из ячеек $b_1, \ldots, b_n$ в нужные для своей программы ячейки, так что *никакой экономии не получается*.

Гораздо удобнее иметь стандартную программу, которая будет брать исходные числа из тех ячеек, где они лежат, и помещать их сразу на нужное место. В такой программе команды T_1^0 и T_2^0 не могут

быть написаны заранее. *Их необходимо сформировать по заданной информации.*

Информация к этой программе составляет три не известных нам заранее адреса: адреса a_1 и a_n начальной и конечной ячеек массива десятичных чисел и адрес b_1 первой ячейки для размещения двоичных чисел. Предположим, что эти адреса записаны в некоторой ячейке памяти

$$i\colon\ (0,\ a_1,\ b_1,\ a_n).$$

Пример 2.43. Пусть задана ячейка i, содержащая указанную выше информацию. Составим стандартную программу, переводящую десятичные числа из ячеек $a_1, \ldots, a_n$ в двоичную систему и размещающую двоичные числа в ячейки, начиная с b_1.

Для этой цели можно воспользоваться составленной в предыдущем примере программой. Нужно только сформировать команды T_1^0 и T_2^0. Впрочем, можно сразу формировать начальное состояние команд T_1 и T_2. Надобность в восстановлении этих переменных команд в данном случае отпадает, поскольку они в начале работы программы все равно формируются. Кроме того, нужно еще позаботиться о восстановлении счетчика ν.

Заметим, что $a_n - a_1 = n - 1$. Поэтому начальное состояние счетчика можно получить из ячейки i при помощи следующих команд:

$$\begin{aligned}
&1)\ \overline{30}\leftarrow,\quad i\quad =R\\
&2)\ R\ -,\quad i\quad =\nu\\
&3)\ \nu\ \wedge\ (F,\ 0,\ 0)\ =\nu.
\end{aligned}$$

Действительно, команда 1) сдвигает a_n в левый адрес, команда 2) образует в левом адресе ячейки ν величину $n-1$. Наконец, команда 3) высекает этот адрес.

Команды T_1 и T_2, которые нужно сформировать, имеют вид

$$\begin{array}{llllll}
T_1 & & & \overrightarrow{a_1 = \alpha} & & \\
T_2 & Y_0 & \gamma & A & b_1. &
\end{array}$$

Запишем в двух ячейках памяти команды-«заготовки»

$$\begin{array}{llllll}
C_1 & & & \overrightarrow{0 = \alpha} & & \\
C_2 & Y_0 & \gamma & A & 0. &
\end{array}$$

Команду T_1 можно получить, прибавив ко второму адресу C_1 адрес a_1. Аналогично для получения команды T_2 надо к правому адресу C_2 прибавить b_1. Это можно сделать следующим образом:

$$\begin{aligned}
&4)\ \overline{14}\rightarrow,\quad i\quad =R\\
&5)\ R\ \wedge\ (0,\ F,\ 0)\ =T_1\\
&6)\ C_1\ \vee\quad T_1\quad =T_1\\
&7)\ R\ \wedge\ (0,\ 0,\ F)\ =R\\
&8)\ C_2\ \vee\quad R\quad =T_2.
\end{aligned}$$

В самом деле, после сдвига командой 4) в ячейке R в среднем адресе будет a_1, а в правом b_1, и их нужно только высечь и прибавить к нужным командам-заготовкам.

Команды 1) — 8) и константы C_1, C_2 образуют *формирующую часть стандартной подпрограммы.* Объединяя формирующую часть и счетный цикл, написанный в предыдущем примере, получим программу группового перевода «10 → 2».

При составлении этой программы мы считали, что информация к ней находится в некоторой известной ячейке l, которую можно считать стандартной. Тогда при обращении к программе необходимо предварительно засылать в ячейку l соответствующую информацию. Более удобно помещать информацию рядом с командой обращения к стандартной программе, так, чтобы стандартная программа сама переносила ее в нужное место.

Обращение к такой стандартной программе принято писать так:

$$\begin{array}{lllll} Я: & БВ & Я+2 & 10\to 2 & Ω \\ Я+1: & 0 & a_1 & b_1 & a_n \\ Я+2: & \text{что делать дальше.} & & & \end{array}$$

При этом обращении в ячейку Ω засылается команда $БВ\ \ Я+2$, т. е. возврат происходит в ячейку, содержащую следующую команду программы. Ячейка $Я+1$ содержит информацию для стандартной программы, к которой мы обращаемся. Ее называют *строкой информации.* Здесь мы (впервые) встречаемся со словом, которое *прочитывается не как число и не как команда, а иным способом.*

Для окончательного завершения программы остается произвести перенос информации из ячейки $Я+1$ в ячейку l. Это можно сделать командой

$$\text{U:}\ \ Я+1=l,$$

которую также нужно сформировать. К моменту начала работы программы в ячейке Ω записана команда $БВ\ \ Я+2$. Поэтому, если взять заготовку U^0 в виде

$$\text{U}^0\text{:}\ \ F\vee F=l,$$

то прибавление к ней $Я+2$ в среднем адресе даст нам нужную команду. Поэтому перенос информации достигается двумя командами:

$$\begin{array}{ll} U^0+, & Ω\ =U \\ \text{U (} & Я+1=l) \text{ н. п.} \end{array}$$

Присоединяя эти две команды к формирующей части программы и заготовку U^0 к константам, получим *стандартную программу*

группового перевода «10 → 2» в окончательном виде:

```
        1)        U⁰ +,   Ω     = U
  U     2)   (                Я + 1  = i)  н. п.
        3)       30  ←,   i     = R
        4)        R  —,   i     = ν
        5)        ν  ∧  (F, 0, 0) = ν
        6)       14  →,   i     = R
        7)        R  ∧  (0, F, 0) = T1
        8)       C1  ∨     T1    = T1
        9)        R  ∧  (0, 0, F) = R
       10)       C2  ∨     R     = T2
  T1   11)   (                a1  = α)  н. п.
       12)  БВ Я + 3        10 → 2  к. 10 → 2
       13)       T1  +, (0, 1, 0) = T1
       14)       T2  +, (0, 0, 1) = T2
       15)        ν  —, (1, 0, 0) = ν
            ──────────────────────────→
  T2   16)  (У0   γ        T1      b1)  н. п.
       17)  Б                Ω
  C1   18)                   0     = α
            ──────────────────────────→
  C2   19)  У0   γ       T1 — 1    F
  U⁰   20)       F  ∨      F      = i
```

Составленная нами *стандартная программа* состоит из следующих основных частей:

1) *перенос информации*, находящейся в ячейке $Я + 1$ в стандартную ячейку i;

2) *формирование* команд рабочей части программы, зависящих от адресов ячейки i;

3) *рабочая часть* программы (в нашем примере в рабочую часть входит еще стандартный блок перевода одного числа);

4) *константы формирования*.

Наличие этих четырех частей характерно для любой стандартной программы с информацией. При этом первая часть (перенос информации) организуется во всех программах совершенно одинаково.

Отметим, что команда обращения к подпрограмме играет для программ с информацией двоякую роль. Во-первых, как и при обращении к любому блоку, она обеспечивает возврат в основную программу. Во-вторых, записывая в ячейке Ω адрес места, куда следует возвратиться, она тем самым сообщает стандартной программе местонахождение строки информации.

Такие стандартные программы называют *программами с принудительным концом*, в отличие от *программ со свободным концом*, которыми

являются программы без информации; последние допускают обращение

$$БВ \quad S \quad СП \quad Ω,$$

где $СП$ — начало стандартной программы, а S — произвольная ячейка. К программе с информацией такое обращение возможно лишь тогда, когда информация находится в ячейке $S-1$.

Некоторые стандартные программы требуют не одной строки информации, а двух или трех. В таком случае в ячейку Ω засылается возврат к $Я+3$ или $Я+4$.

Можно привести большое число различных примеров стандартных программ с информацией. Одной из наиболее важных таких программ является *программа печати*. Как мы знаем, результаты вычислений получаются в машине в двоичной системе счисления. Программа печати обеспечивает вывод полученных результатов. Она переводит окончательные данные в десятичную систему, обращаясь к программе «2 → 10», и печатает их. Обычное обращение к этой программе имеет вид

$$\begin{array}{llll} Я: & БВ\ Я+2 & Печать & Ω \\ Я+1: & a_1 & q & a_n, \end{array}$$

где a_1, a_n — адрес первой и последней ячейки печатаемого массива. Число q, стоящее в среднем адресе ячейки информации, показывает, что полученные числа нужно печатать группами по q штук в каждой. Это делается для удобства ориентировки в табулограмме.

Мы познакомились, таким образом, с *формированием команд* машиной. Так принято называть *одно или несколько действий, целью которых является получение команды*. Для формирования команд применяются, как мы видели, фиксированные действия, сдвиги и логические операции. Частным случаем формирования команд является *переадресация*, рассматривавшаяся нами в гл. VI.

§ 44. Применение регистра адреса в стандартных подпрограммах

Алгоритмы многих стандартных подпрограмм приводят к циклам с переадресацией. Как мы видели, в большинстве случаев такие циклы наиболее экономно образуются при помощи регистра адреса. Регистр адреса широко применяется в стандартных подпрограммах как для образования циклов, так и для формирования.

Пример 1.44. Составим стандартную подпрограмму для вычисления скалярного произведения двух векторов $a\{a_1, a_2, \ldots, a_n\}$ и $b\{b_1, b_2, \ldots, b_n\}$. Скалярным произведением векторов a и b называется величина *)

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

*) См. ниже § 63.

Обращение к стандартной подпрограмме, вычисляющей c, запишем в виде

$$\begin{array}{cccc} БВ & Я+2 & СП & Ω \\ n & a_1 & b_1 & c, \end{array}$$

где n — записанная в коде размерность каждого из векторов, a_1 — адрес первой координаты вектора a, b_1 — адрес первой координаты b, c — адрес результата. Координаты векторов $a(a_1, a_2, \ldots, a_n)$ и $b(b_1, b_2, \ldots, b_n)$ предполагаются записанными в последовательных ячейках памяти (подряд).

Приступая к составлению подпрограммы, напишем прежде всего ее рабочую часть. Если известна размерность n и адреса a_1, b_1, c, то скалярное произведение находится при помощи следующего простого цикла:

$$\begin{array}{lcccc} PA & 0 & 0 & 0 & \\ & & 0 & = c & \\ & a_1^* & \cdot\ b_1^* & = R_1 & \Big| \\ & c & + R_1 & = c & \Big| \\ PA < n-1 & & & \overline{1}^*. & \end{array}$$

Из пяти написанных команд четыре зависят от n, a_1, b_1, c и, следовательно, должны формироваться. Поэтому при такой рабочей части подпрограммы формирующая часть будет очень большой (она займет 13 команд и четыре константы). Для того чтобы уменьшить количество ячеек, занимаемых всей подпрограммой, следует постараться уменьшить число формируемых команд. Этому условию удовлетворяет следующий цикл получения c:

$$\begin{array}{lcccc} & [PA] & & N & \\ & & & 0 = s & \\ T_1 & & a_0^* & \cdot\ b_0^* = R_1 & \Big| \\ & & s & + R_1 = s & \Big| \\ & PA \geqslant 2 & & \overline{F}^* & \\ T_2 & & & s = c. & \end{array}$$

Здесь через N обозначена ячейка, второй адрес которой равен n, а a_0 и b_0 — адреса ячеек, находящихся соответственно перед a_1 и b_1. Формироваться в этой программе должны команды T_1, T_2 и константа N. Прежде чем составлять формирующую часть, сделаем одно важное замечание.

К стандартным подпрограммам, использующим регистр адреса, предъявляется следующее обязательное требование: *после выхода из*

подпрограммы PA должен иметь то же значение, какое было у него при входе. Поэтому в написанном цикле должен восстанавливаться регистр адреса. Организуя восстановление PA и выход в стандартную ячейку Ω, получим рабочую часть подпрограммы в виде

$$
\begin{array}{llllll}
 & [PA] & 0^* & N & BP \\
 & & & 0 = s & \\
T_1 & & a_0^* \cdot b_0^* = R_1 & & \uparrow \\
 & & s + R_1 = s & & | \\
 & PA \geqslant \overline{2} & & \overline{F^*} & | \\
BP & & \text{н. п.} & & \\
 & & \longrightarrow & & \\
T_2 & Б & s & Ω & c.
\end{array}
$$

Засылку информации и формирование, предшествующие рабочей части, напишем в таком виде:

$$
\begin{array}{lllll}
 & U^0 +, & Ω & = U \\
U & & \text{н. п.} & & \\
 & \overline{30} \rightarrow & i & = N \\
 & i \bigwedge (F, F, 0) & & = R_1 \\
 & T_1^0 +, & R_1 & = T_1 \\
 & i \bigwedge (0, 0, F) & & = R_1 \\
 & T_2^0 \bigvee & R_1 & = T_2.
\end{array}
$$

Здесь U^0, T_1^0 и T_2^0 — константы, имеющие вид

$$
\begin{array}{llll}
U^0 & F \bigvee F & = i \\
T_1^0 & (F - 1)^* \cdot F^* & = R_1 \\
 & \longrightarrow & & \\
T_2^0 \; Б & s & Ω & 0.
\end{array}
$$

Вся подпрограмма *Скалярное произведение* состоит из 14 команд и трех констант. Эту подпрограмму можно сократить еще на три ячейки следующим способом.

Для извлечения строки информации можно воспользоваться командой с кодом $[PA]$ так:

$$
\begin{array}{lll}
1) & [PA] & Ω \\
2) & & F^* = i.
\end{array}
$$

Действительно, по первой из этих команд в PA попадет $Я + 2$, т. е. увеличенный на единицу адрес строки информации; вторая команда будет исполняться в виде

$$Я + 1 = i,$$

т. е. выполнится нужная пересылка строки информации. Заметим, что можно не пересылать строку информации в l, а непосредственно производить формирование, воспользовавшись тем, что в PA находится $Я+2$. Так, команду формирования N можно написать в виде

$$\overline{30} \to F^* = N.$$

Команда 1) изменяет значение PA, поступившее на вход в подпрограмму. Перепишем поэтому ее с восстановлением PA:

$$1)' \quad [PA] \quad 0^* \quad Ω \quad Ω-1.$$

Здесь мы подготавливаем восстановление PA не в самой стандартной подпрограмме, а в ячейке $Ω-1$, предшествующей стандартному концу $Ω$. Это дает возможность сэкономить еще одну ячейку BP. При этом следует первую и последнюю команду рабочей части заменить следующими:

$$[PA] \; 0 \quad N \quad 0$$

и

$$Б \quad s \quad \overrightarrow{Ω-1} \quad c.$$

Со всеми указанными изменениями подпрограмма *Скалярное произведение* приведена в табл. 1.44 в содержательных обозначениях и закодированном виде. При кодировке для рабочих ячеек были взяты следующие адреса:

R_1	0011
N	0017
s	0020
$Ω-1$	0006
$Ω$	0007

Заметим, что использованный в этом примере прием извлечения строки информации и восстановления применяется во всех стандартных подпрограммах, использующих PA. Каждая такая подпрограмма, как правило, начинается с команд:

$$[PA] \quad 0^* \quad Ω \quad Ω-1$$
$$F^* = l$$

и кончается безусловной передачей управления ячейке $Ω-1$.

Рассмотрим теперь пример, в котором все формирование в стандартной подпрограмме производится при помощи регистра адреса.

Пример 2.44. Составить стандартную подпрограмму *Комплексная арифметика* (*КА*) для четырех арифметических действий над комплексными числами.

Будем предполагать, что каждое комплексное число занимает две последовательные ячейки памяти, в первой из которых находится его действительная часть, во второй — мнимая. Обращение к стандартной подпрограмме

Таблица 1.44

					6000	A		
	$[PA]$ 0^* Ω Ω − 1	6000	4	72	0000	0007	0006	
	$\overline{30} \rightarrow F^* = N$	1	2	54	0050	7777	0017	
	$F^* \wedge (F, F, 0) = R_1$	2	4	55	7777	0136	0011	
	$T_1^0 +$, $R_1 = T_1$	3	0	13	6014	0011	6010	
	$F^* \wedge (0, 0, F) = R_1$	4	4	55	7777	0131	0011	
	$T_2^0 \vee R_1 = T_2$	5	0	75	6015	0011	6013	
	$[PA]$ N	6	0	72	0000	0017	0000	
	$0 = s$	7	0	75	0000	0000	0020	
T_1	$(a_0^* \cdot b_0^* = R_1)\uparrow$	6010				н. п.		
	$s + R_1 = s$	11	0	01	0020	0011	0020	
	$PA \geqslant \bar{2}$ F^*	12	1	32	0002	6010	7777	

					6013	A		
T_2	$(Б$ $\overrightarrow{s \quad Ω-1 \quad c})$	6013				н. п.		
T_1^0	$(F-1)^* \cdot F^* = R_1$	4	6	05	7776	7777	0011	
T_2^0	$Б$ $\overrightarrow{s \quad Ω-1 \quad 0}$	5	0	56	0020	0006	0000	

построим следующим образом. Если нужно сложить два комплексных числа $u = u_0 + iu_1$ и $v = v_0 + iv_1$ и получить сумму в ячейках z_0, z_1 $(z = z_0 + iz_1)$, то напишем обращение:

$$БВ \quad Я+2 \quad КА \quad Ω$$
$$u_0 + v_0 = z_0.$$

Если нужно эти числа вычесть, то обращение запишем в виде

$$БВ \quad Я+2 \quad КА \quad Ω$$
$$u_0 - v_0 = z_0.$$

Для умножения и деления будем писать соответственно

БВ Я+2 КА Ω

$$u_0 \cdot v_0 = z_0$$

и

БВ Я+2 КА Ω

$$u_0 : v_0 = z_0.$$

Таким образом, чтобы произвести какое-либо арифметическое действие над двумя комплексными числами u и v, следует обратиться к стандартной подпрограмме, начинающейся с команды *КА*, с одной строкой информации, которая имеет вид соответствующей арифметической операции над действительными частями u_0, v_0 комплексных чисел u, v. При помощи стандартной программы *Комплексная арифметика* можно, очевидно, программировать вычисление выражений, содержащих четыре арифметических действия над комплексными числами почти так же просто, как и выражений, содержащих действительные числа.

Приступая к составлению подпрограммы *КА*, выпишем прежде всего известные формулы для четырех арифметических действий над парами комплексных чисел:

сложение и вычитание

$$z = u \pm v; \qquad \begin{aligned} z_0 &= u_0 \pm v_0, \\ z_1 &= u_1 \pm v_1; \end{aligned}$$

умножение

$$z = uv, \qquad \begin{aligned} z_0 &= u_0v_0 - u_1v_1, \\ z_1 &= u_0v_1 + u_1v_0; \end{aligned}$$

деление

$$z = \frac{u}{v}, \qquad \begin{aligned} z_0 &= \frac{u_0v_0 + u_1v_1}{v_0^2 + v_1^2}, \\ z_1 &= \frac{-u_0v_1 + u_1v_0}{v_0^2 + v_1^2}. \end{aligned}$$

Заметим, что формулы для произведения и частного можно записать в одном и том же виде:

$$\begin{aligned} z_0 &= u_0w_0 - u_1w_1, \\ z_1 &= u_0w_1 + u_1w_0, \end{aligned}$$

где для произведения uv

$$w_0 = v_0, \quad w_1 = v_1,$$

а для частного $\frac{u}{v}$ величины w_0 и w_1 суть действительная и мнимая части обратной величины $\frac{1}{v}$, определяемые формулами

$$w_0 = \frac{v_0}{v_0^2 + v_1^2}, \qquad w_1 = -\frac{v_1}{v_0^2 + v_1^2}.$$

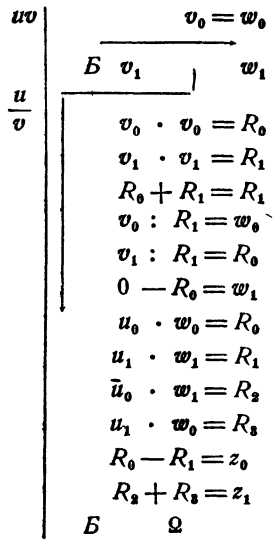
Если известны адреса u_0, u_1, v_0, v_1, z_0, z_1, то программы нахождения $u+v$, $u-v$, uv и $\frac{u}{v}$ записываются в следующем виде:

$u+v$ | *Б*

$$\begin{aligned} u_0 + v_0 &= w_0 \\ u_1 + v_1 &= w_1 \\ &\Omega \end{aligned}$$

$u-v$ | *Б*

$$\begin{aligned} u_0 - v_0 &= w_0 \\ u_1 - v_1 &= w_1 \\ &\Omega \end{aligned}$$



Заметим, что, воспользовавшись приведенными выше формулами для обратной величины $\frac{1}{v}$, мы написали единую программу для умножения и деления. Если передать управление на команду с меткой *uv*, то вычислится произведение, при передаче управления на команду с меткой $\frac{u}{v}$ — частное.

Программа всех четырех арифметических действий займет 21 ячейку памяти. Отметим, что 17 команд программы зависят от адресов u_0, u_1, v_0, v_1, w_0, w_1, т. е. требуют формирования. Ниже мы покажем, что формирование значительно облегчится при использовании регистра адреса.

Начало подпрограммы запишем в следующем виде:

	1)	[*PA*]	0* Ω	Ω − 1
	2)	(775, 0, 0, 0)	, + *F**	= 0
	3)	$У_1$	*F** умнож. делен.	*j*
	4)		*j* +, (1, 1, 1)	= *j* + 1
j	5)		н. п.	
j + 1	6)		н. п.	
	7)	*Б*	Ω − 1	

Команда 1) заносит в *PA* адрес строки информации, увеличенный на 1, и подготавливает восстановление *PA* основной программы в ячейке Ω − 1. Затем команда 2) анализирует код операции строки информации. Если этот код равен 04 или 05 (деление или умножение), то при увеличении порядка строки информации на 775_8 выработается сигнал $\omega = 1$ и команда 3) передает управление на начало программы умножения-деления. При кодах же сложения или вычитания (01 или 02) после команды 2) ω будет равно 0 и команда 3) передаст управление на 4). Одновременно с передачей управления по команде 4) строка информации перешлется в ячейку *j*, Таким образом,

в ячейке j уже подготовлена команда сложения (или вычитания) действительных частей комплексных чисел u и v.

Команда 4), увеличивая на 1 все адреса строки информации, подготавливает в ячейке $j+1$ команду сложения (или вычитания) мнимых частей. Затем в ячейках j и $j+1$ выполняется сложение (или вычитание) комплексных чисел и по команде 7) передается управление на стандартную ячейку $\Omega-1$, где восстанавливается PA, а затем через ячейку Ω происходит возврат в основную программу.

Часть стандартной подпрограммы KA, организующая умножение и деление, напишем в таком виде:

умножение-деление	8)	$[PA]$			j	
	9)	(773, 0, 0, 0)			$,+j$	$=0$
умножение	10)	$У_0$	0^*	деление		w_0
	11)	$Б$	$\overline{1}^*$	общая		w_1
деление	12)		0^*	$\cdot$	0^*	$=R_0$
	13)		$\overline{1}^*$	$\cdot$	$\overline{1}^*$	$=R_1$
	14)		R_0	$+$	R_1	$=R_1$
	15)		0^*	$:$	R_1	$=w_0$
	16)		$\overline{1}^*$	$:$	R_1	$=R_1$
	17)		0	$-$	R_1	$=w_1$
общая	18)		$\overline{14}$	$\rightarrow$	j	$=u$
	19)	$[PA]$			u	
	20)		0^*	$\cdot$	w_0	$=R_0$
	21)		$\overline{1}^*$	$\cdot$	w_1	$=R_1$
	22)		0^*	$\cdot$	w_1	$=R_2$
	23)		$\overline{1}^*$	$\cdot$	w_0	$=R_3$
	24)		$\overline{14}$	$\leftarrow$	j	$=z$
	25)	$[PA]$			z	
	26)		R_0	$-$	R_1	$=0^*$
	27)		R_2	$+$	R_3	$=\overline{1}^*$
	28)	$Б$			$\Omega-1$	
	29)	(773, 0, 0, 0)				
	30)	(775, 0, 0, 0)				

Разберем работу этой части подпрограммы. Команда 8) заносит в PA адрес действительной части v_0 комплексного числа v ($v=v_0+iv_1$). Команда 9) проверяет код строки информации; если этот код равен 04 (деление), то вырабатывается сигнал $\omega=0$, если он равен 05 (умножение), то $\omega=1$. В случае умножения по командам 10), 11) в ячейки w_0, w_1 засылается число v. В случае же деления при помощи команд 12)—17) в w_0 и w_1 образуются действительная и мнимая части обратной величины $\frac{1}{v}\left(w_0=\frac{v_0}{v_0^2+v_1^2}, w_1=-\frac{v_1}{v_0^2+v_1^2}\right)$. Затем и в случае умножения, и в случае деления выполняются команды 18)—28). Команды 18), 19) заносят в PA адрес действитель-

ной части u_0 числа u ($u = u_0 + iu_1$). Затем при помощи команд 20) — 23) вычисляются произведения u_0w_0, u_1w_1, u_0w_1, u_1w_0. После этого команды 24), 25) заносят в PA адрес действительной части результата и, наконец, при помощи команд 26), 27) вычисляются действительная и мнимая части произведения или частного.

§ 45. Библиотека стандартных подпрограмм

Совокупность всех стандартных подпрограмм образует *библиотеку стандартных подпрограмм*. Библиотечные подпрограммы располагают обычно в конце памяти.

В библиотеку, кроме подпрограмм, включают еще часто встречающиеся константы: плавающие числа «1», «2», «3», «4», «5», «6»,

Таблица 1.45

0117		0157
0120		0160
0121	(0, 0, 1)	0161
0122	(0, 1, 0)	0162
0123	(0, 1, 1)	0163
0124	(1, 0, 0)	0164
0125	(1, 0, 1)	0165
0126	(1, 1, 0)	0166
0127	(1, 1, 1)	0167
0130	(77, 0, 0, 0)	0170
0131	(0, 0, *F*)	0171
0132	(0, *F*, 0)	0172
0133	(0, *F*, *F*)	0173
0134	(*F*, 0, 0)	0174
0135	(*F*, 0, *F*)	0175
0136	(*F*, *F*, 0)	0176
0137	(*F*, *F*, *F*)	0177

«7», «8», «9», «10», «1/2», «1/3», «1/4», «1/8», «0,1», $\frac{\pi}{2}$, π, 2π, e, $\sqrt{2}$, константы переадресации (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1), константы выделения (77, 0, 0, 0), (0, 0, F), (0, F, 0), (0, F, F), (F, 0, 0), (F, 0, F), (F, F, 0), (F, F, F).

Расположение некоторых констант в памяти указано на памятке (табл. 1.45). Кроме того, на памятке помечают буквами R_1, R_2, ..., R_{17}, Ω, Ω — 1, α_0, α_1, γ_0, γ_1 — рабочие ячейки библиотеки. Библиотеку снабжают *каталогом*, т. е. перечнем всех стандартных подпрограмм с краткими пояснениями к ним.

Каталог нашей библиотеки. Подпрограммы с входными α_0, α_1 и выходными γ_0, γ_1 ячейками. Обращение *БВ S f* Ω, где *f* — обозначение вычисляемой функции.

f	Результат	
exp	$\gamma_0 = \exp \alpha_0$	$(= e^{\alpha_0})$
ln	$\gamma_0 = \ln \alpha_0$	
$\sqrt[3]{\ }$	$\gamma_0 = \sqrt[3]{\alpha_0}$	
cos, sin	$\gamma_0 = \cos \alpha_0$,	$\gamma_1 = \sin \alpha_0$
arccos, arcsin	$\gamma_0 = \arccos \alpha_0$,	$\gamma_1 = \arcsin \alpha_0$
arcctg, arctg	$\gamma_0 = \operatorname{arcctg} \alpha_0$,	$\gamma_1 = \operatorname{arctg} \alpha_0$
ch, sh	$\gamma_0 = \operatorname{ch} \alpha_0$,	$\gamma_1 = \operatorname{sh} \alpha_0$ *)
↑	$\gamma_0 = \alpha_0^{\alpha_1}$	

Подпрограммы с информацией. Обращение

БВ *Я*+2 *U* Ω
k *a* *b* *c*;

где *U* — название (обозначение) подпрограммы, а (*k*, *a*, *b*, *c*) — строка информации к ней.

Название подпрограммы *U*	Строка информации	Пояснение
Перевод	0, a_1, b_1, a_n	Массив чисел от a_1 до a_n переводится из двоично-десятичной системы в двоичную и ставится, начиная с ячейки b_1
Печать	0, a_1, q, a_n	Массив чисел от a_1 до a_n переводится из двоичной системы в двоично-десятичную и печатается группами по q штук
Перенос	0, a_1, b_1, a_n	Массив чисел от a_1 до a_n переносится на другое место оперативной памяти, начиная с b_1

*) О функциях ch x, sh x см. § 86.

Кроме этих наиболее употребительных подпрограмм, в библиотеку включают еще подпрограммы действий над комплексными числами, решения систем линейных уравнений, интегрирования и другие *).

Составление библиотечных подпрограмм является высококвалифицированной работой, так как к ним предъявляются очень жесткие требования. Они должны быть по возможности быстрыми и короткими. Эти два требования обычно противоречат друг другу; так, если алгоритм программы имеет циклический характер, то самой короткой будет программа с циклом, а самой быстрой — программа без цикла. Это обстоятельство заставляет составителей стандартных подпрограмм идти при удовлетворении указанных требований на разумный компромисс. В настоящее время машины обладают довольно большим быстродействием и ограниченным объемом оперативной памяти. Поэтому обычно подпрограммы стараются делать короткими, жертвуя, в разумных пределах, быстротой.

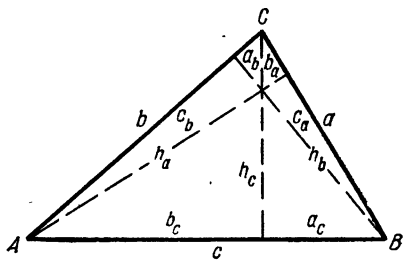


Рис. 36.

В заключение главы рассмотрим один пример, в котором используются несколько библиотечных подпрограмм.

Пример 1.45. В треугольнике ABC даны две стороны a, b и угол C между ними. Вычислить высоты треугольника и проекции сторон друг на друга. При помощи рис. 36 получим формулы, определяющие искомые отрезки:

$$h_a = b \sin C, \quad b_a = b \cos C, \quad c_a = a - b_a,$$

$$h_b = a \sin C, \quad a_b = a \cos C, \quad c_b = b - a_b,$$

$$A = \arccos \frac{c_b}{\sqrt{c_b^2 + h_b^2}},$$

$$h_c = b \sin A, \quad b_c = b \cos A, \quad a_c = c - b_c.$$

Мы будем предполагать, что величины a, b, C находятся в последовательных ячейках $a_{(10)}$, $b_{(10)}$, $C_{(10)}$ в двоично-десятичной форме, причем угол C_{10} задан в градусах и долях градуса.

Эти три величины перед расчетом по формулам следует перевести в двоичную систему счисления, а C, кроме того, перевести в радианы.

*) Некоторые из этих подпрограмм мы рассмотрим в третьей части.

Результаты расчета поместим в памяти в следующем порядке:

$$h_a,\ h_b,\ h_c;\quad a_b,\ a_c;\quad b_a,\ b_c;\quad c_a,\ c_b,$$

переведем в десятичную систему и напечатаем.

Программа решения задачи имеет следующий вид:

БВ Я+2		*Перевод*	Ω
$a_{(10)}$		a	$C_{(10)}$
C	·	«$\frac{\pi}{180}$»	$=a$
БВ Я+1		cos, sin	Ω
b	·	γ_1	$=h_a$
b	·	γ_0	$=b_a$
a	—	b_a	$=c_a$
a	·	γ_1	$=h_b$
a	·	γ_0	$=a_b$
b	—	a_b	$=c_b$
c_b	·	c_b	$=R_1$
h_b	·	h_b	$=R_2$
R_1	+	R_2	$=R_1$
$\sqrt{R_1}$			$=R_1$
c_b	:	R_1	$=a$
БВ Я+1		arccos, arcsin	Ω
		γ_0	$=a$
БВ Я+1		cos, sin	Ω
b	·	γ_1	$=h_c$
b	·	γ_0	$=b_c$
c	—	b_c	$=a_c$
БВ Я+2		*Печать*	Ω
h_a		$\overline{3}$	h_c
БВ Я+2		*Печать*	Ω
a_b		$\overline{2}$	c_b
стоп.			

ГЛАВА IX

ОРГАНИЗАЦИЯ ПРОГРАММЫ

§ 46. Блочное программирование

На машине обычно решают задачи, программы которых содержат многие сотни, а иногда и тысячи команд. В таких задачах всегда следует выделять части, имеющие самостоятельное значение, и в программе эти части писать как отдельные блоки.

Выделение отдельных частей программы в блоки следует производить не только в сложных, но и в простых программах. Так, например, если рабочая часть цикла содержит более двух-трех команд, ее следует выделять в отдельный блок. Тем более целесообразно выделять в отдельный блок внутренние циклы в двойных и многократных циклах.

Пример 1.46. Рассчитать таблицу значений функции

$$y=\left(x-\frac{1}{x}\right)^2+x^3$$

для $x=1, 2, 3, \ldots, 20$.

Напишем прежде всего блок вычисления y при заданном значении x. При этом, как было указано в § 42, обращаясь к блоку, мы будем засылать команду возврата в стандартную ячейку Ω и кончать блок командой передачи управления этой ячейке:

$y(x)$ | «1» : $x = R_1$
$x - R_1 = R_1$
$R_1 \cdot R_1 = R_1$
$x \cdot x = R_2$
$R_2 \cdot x = R_2$
$R_1 + R_2 = y$
$Б$ Ω

Теперь при помощи регистра адреса образуем цикл счета y для различных значений x:

Таблица

$РА$	0 →	0	0
$Б$	«1»		x
	x	$+$ «1»	$=x$
$БВ$ $Я+1$	$y(x)$		Ω
		y	$=y_1^*$
$РА<\overline{23}$			$\overline{1}^*$
стоп.			

В этом примере программа, в которой счет y не был бы выделен в отдельный блок, была бы короче на две ячейки. Однако потеря двух ячеек в написанной программе с лихвой компенсируется тем обстоятельством, что программа выигрывает в простоте и наглядности. Отметим, в частности, что *команда обращения к блоку*

$$БВ \quad Я+1 \quad y(x) \quad Ω$$

выглядит в программе как «элементарная» операция: счет $y(x)$. Тип этой элементарной операции зависит от того, что делает сам блок *Счет* $y(x)$.

В случае необходимости можно в рассмотренном цикле протабулировать при $x=1, 2, 3, \ldots, 20$ другую функцию $z(x)$. Для этого достаточно заменить в программе обращение к блоку

$$БВ \quad Я+1 \quad y(x) \quad Ω$$

на

$$БВ \quad Я+1 \quad z(x) \quad Ω$$

и написать блок счета $z(x)$.

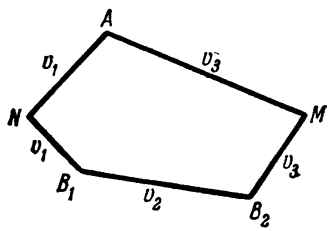


Рис. 37.

Пример 2.46. Самолет должен вылететь из пункта N, взять груз в некоторых пунктах и доставить его в пункт M. Пунктами погрузки может быть либо пункт A, либо два пункта B_1 и B_2 (рис. 37). Расстояния NA, AM, NB_1, B_1B_2, B_2M заданы. Самолет без груза летит со скоростью v_1, с неполным грузом — со скоростью v_2, с полным грузом — со скоростью v_3.

В пункте B_1 на погрузку тратится τ_1 минут, в пункте B_2 — τ_2 минут, в пункте A — τ_3 минут. По какому из путей, NAM или NB_1B_2M, должен лететь самолет, чтобы быстрее доставить груз?

Здесь естественно выделить в программе три блока: счет времени полета через пункт A (блок *Через A*), счет времени полета

через пункты B_1 и B_2 (блок *Через* B_1, B_2) и, наконец, сравнение этих двух времен (блок *Сравнение*).

Блок *Сравнение* должен в результате своей работы выработать некоторый признак, по которому можно судить о том, по какому пути должен лететь самолет. Пусть, например, в ячейку ν засылается нуль, если выгоден путь через A, и (0, 0, 1), если через B_1B_2.

Блоки программы имеют следующий вид:

Через A	$NA : v_1 = R_1$
	$R_1 + \tau_3 = R_1$
	$AM : v_3 = t_A$
	$t_A + R_1 = t_A$
	Б Ω
Через B_1B_2	$NB_1 : v_1 = R_1$
	$R_1 + \tau_1 = R_1$
	$B_1B_2 : v_2 = R_2$
	$R_2 + \tau_2 = R_2$
	$R_1 + R_2 = R_1$
	$B_2M : v_3 = R_2$
	$R_1 + R_2 = t_B$
	Б Ω
Сравнение	$t_A - t_B = 0$
	$У_1$ 0 $Я + 2$ ν
	(0, 0, 1) = ν
	Б Ω

Чтобы объединить эти три блока в единую программу, напишем *собирающую программу*, состоящую из обращений к этим трем блокам и стопа:

Собирающая	*БВ* $Я + 1$ *Через A* Ω
	БВ $Я + 1$ *Через* B_1B_2 Ω
	БВ $Я + 1$ *Сравнение* Ω
	стоп.

Заметим, что при принятом способе обращения к блокам команды обращения (в содержательных обозначениях) отличаются лишь вторыми адресами. Поэтому для сокращения записи и для наглядности можно писать в собирающей только обозначения вторых адресов.

Тогда собирающая рассмотренного примера примет вид

Собирающая	*Через A*
	Через B_1B_2
	Сравнение
	стоп.

Программа решения рассмотренной задачи состоит из *собирающей* и трех блоков: *Через A*, *Через B_1*, *B_2*, *Сравнение*. Для решения задачи все эти четыре программы вместе с исходными данными нужно закодировать, ввести в память и передать управление первой команде собирающей.

Блочная структура программы дает возможность в ряде случаев переходить от решения простых задач к решению более сложных, не меняя написанных блоков, а внося в них лишь некоторые добавления и исправления и дописывая новые блоки.

Пусть, например, в рассматриваемой задаче известны не расстояния, а прямоугольные координаты пунктов. Тогда для решения задачи следует, ничего не меняя в написанных трех блоках программы, написать еще четвертый блок, определяющий расстояния, а собирающую дополнить обращением к этому блоку:

Новая собирающая	*Расстояния*
	Через A
	Через B_1B_2
	Сравнение
	стоп.

В рассматриваемой задаче обращения к блокам содержались лишь в собирающей программе. В ряде случаев такие обращения могут содержаться и в самих блоках программы, в частности, блоки программы могут содержать обращения к стандартным подпрограммам.

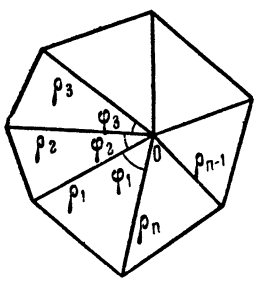


Рис. 38.

Пример 3.46. Внутри многоугольника с n сторонами выбрана точка. Заданы длины отрезков, соединяющих эту точку с вершинами многоугольников, и углы $\varphi_1, \varphi_2, \ldots, \varphi_n$ между этими отрезками (рис. 38). Найти площадь многоугольника S.

Площадь многоугольника будем подсчитывать как сумму площадей треугольников OA_nA_1, $OA_1A_2, \ldots, OA_{n-1}A_n$. Поэтому в рассматриваемой задаче естественно выделить блок *Треугольник*, в котором по двум сторонам и углу между этими сторонами находится площадь

треугольника. Аргументами этого блока пусть будут ячейки a, b (стороны), φ (угол), а ячейкой результата q (площадь).

Основная программа должна последовательно засылать в ячейки a, b, φ стороны и углы треугольников, составляющих многоугольник, обращаться к блоку *Треугольник* и складывать полученные площади треугольников.

Эта программа пишется как обычный цикл с переадресацией по PA и имеет такой вид:

Площадь многоугольника

1)	PA	0^*	0	10)
2)			$0 = S$	
3)	$Б$	ρ_n		a
4)			$b = a$	
5)			$\varphi_1^* = \varphi$	
6)			$\rho_1^* = b$	
7)		*Треугольник*		
8)		$S+$	$q = S$	
9)	$PA <$	$\overline{n-1}$	4)	$\overline{1}^*$
10)			н. п.	
11)	*стоп.*			

Здесь команды 1), 2), 3) служат для подготовки цикла. Команда 1), очищая PA, одновременно подготавливает восстановление PA в ячейке 10). Это делается для того, чтобы в случае необходимости оформить программу *Многоугольник* как блок и включить ее в цикл с переадресацией по PA. Команда 2) приводит в начальное состояние ячейку S для площади многоугольника. Команда 3) засылает в ячейку a одну из сторон и передает управление команде 5), которая вместе с 6) засылает нужные данные в две другие ячейки аргумента. После них команда 7) обращается к блоку *Треугольник*, который и вычисляет площадь первого треугольника. В дальнейшем засылка данных о следующих треугольниках происходит уже в цикле одинаково, с помощью команд 4), 5), 6). Все эти площади суммируются командой 8). Наконец, команда 9) проверяет, кончился ли цикл.

Площадь треугольника по двум сторонам a, b и углу φ между ними определяется формулой

$$q = ab \sin \varphi.$$

Напишем этот блок в обычном виде *):

Треугольник	1)′					φ	$=\alpha$
	2)′	*БВ*	*Я*	$+1$	cos, sin		Ω
	3)′		a	$\cdot$		b	$=R_1$
	4)′		R_1	$\cdot$		γ_1	$=q$
	5)′	*Б*				Ω	

и посмотрим, как он будет работать при обращении к нему из основной программы. Команда 7) передает управление на 1)′ и засылает в Ω команду возврата

Ω: *БВ* 8).

Затем, после пересылки φ в α (команда 1)′, команда 2)′ передает управление на подпрограмму cos, sin и засылает в Ω команду

БВ 3)′.

Таким образом, команда возврата в основную программу, ранее засланная в ячейку Ω, оказывается потерянной, и после вычисления при помощи команд 2)′, 3)′, 4)′ площади треугольника и выполнения команды 5)′ мы возвратимся к команде 3)′, а не к команде 8), что необходимо для правильной работы программы.

Отсюда видно, что блок, содержащий обращение к другим блокам, нельзя заканчивать командой

Б Ω.

Чтобы обеспечить правильную работу такого блока, следует оставить в его конце свободную ячейку *конец*, а в начале выполнить команду пересылки

Ω = *конец*.

Блок *Треугольник* примет после таких изменений вид:

Треугольник	1)″					Ω	= конец
	2)″					φ	$=\alpha$
	3)″				cos, sin		
	4)″		a	$\cdot$		b	$=R_1$
	5)″		R_1	$\cdot$		γ_1	$=q$
	6)″	*конец*.					

*) Нужно только иметь в виду, что аргумент стандартной программы cos, sin должен быть задан в радианной мере. Если величина φ была задана в градусах, нужно иметь ячейку, содержащую константу $\frac{\pi}{180}$ и команду 1)′ писать в виде

$$\varphi \cdot \left\langle \frac{\pi}{180} \right\rangle = \alpha.$$

Теперь в начале работы блока *Треугольник* команда возврата в основную программу, записанная в ячейке Ω, сразу же пересылается *в конец этого блока.* Поэтому любые дальнейшие обращения к другим блокам (в частности, к стандартным подпрограммам) изнутри данного не испортят команду возвращения в основную программу.

Отметим, что рассмотренный прием сохранения команды возврата при оформлении блока является общим. К любому блоку можно обращаться при помощи команды

БВ S Начало блока Ω.

При этом первой командой блока должна быть пересылка

Ω = *конец*,

где *конец* — последняя ячейка данного блока, которую необходимо оставлять свободной. В этом случае внутри данного блока можно в любом количестве ставить подобные же обращения к другим блокам, которые в свою очередь следует оформлять таким же способом.

Самые внутренние блоки, которые не обращаются ни к каким другим, можно было бы не начинать с команды

Ω = *конец*,

а заканчивать командой

Б Ω.

Однако следует оформлять все блоки, в том числе и самые внутренние, указанным *стандартным образом.*

Такая стандартизация нужна для того, чтобы гарантировать себя от возможных ошибок. Кроме того, начиная писать блок, никогда нельзя быть уверенным в том, что некоторую его часть не нужно будет выделить в качестве отдельного блока.

§ 47. Блок-программы

Из рассмотренных в предыдущем параграфе примеров видно, что блоки могут быть весьма различными. Одни из них состоят просто из расписки формул или цикла, другие содержат обращение к стандартным подпрограммам, третьи — проверку каких-либо логических условий. Вообще, блок может быть сколь угодно сложным образованием. Единственным свойством блока, которое можно считать его определением, является следующее:

при обращении к блоку при помощи команды

БВ S Начало блока Ω

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

Обращения к блокам программы, расположенные в нужной последовательности, образуют *блок-программу*, которую мы уже ввели в предыдущем параграфе под названием *собирающей*.

Если обращения к некоторым блокам являются условными, то блок-программа должна содержать проверку соответствующих условий. При этом, если передача управления некоторому блоку зависит от результатов работы предыдущего, предыдущий блок должен быть устроен так, чтобы он вырабатывал признаки, *проверка которых должна производиться в собирающей, но не в блоке*. Это требование непосредственно вытекает из данного выше определения блока.

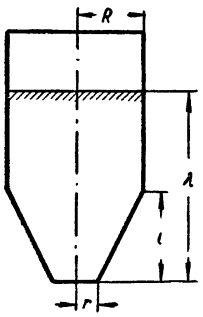


Рис. 39.

Рассмотрим пример с условным обращением к блокам из собирающей.

Пример 1.47. Резервуар с горючим имеет форму усеченного конуса с высотой l и радиусами оснований R и r, завершенного цилиндром с радиусом основания R (рис. 39). В резервуар налито v литров горючего. Найти высоту λ уровня жидкости, отсчитываемую от дна резервуара.

Очевидно, высота λ должна вычисляться по разным формулам в зависимости от того, больше или меньше величина v объема усеченного конуса $v_{УК}$. Поэтому, прежде всего, при помощи блока, который мы назовем *Усеченный конус*, следует вычислить объем $v_{УК}$ усеченного конуса. Затем, если $v < v_{УК}$, то λ следует подсчитывать как высоту наполненной жидкостью части усеченного конуса (блок *По конусу*), если же $v \geqslant v_{К}$, то λ равно высоте наполненной жидкостью части цилиндра, сложенной с l (блок *По цилиндру*). В соответствии с этим планом напишем блок-программу задачи:

Резервуар	*Усеченный конус*
$v-$	$v_{УК} \quad = \Delta v$
$У_0$	$Я+2$
БВ $Я+2$	*По конусу* Ω
	По цилиндру
	стоп.

Здесь мы воспользовались уже указанным ранее способом сокращенной записи: если при обращении к блоку возвращение происходит на следующую команду ($Я+1$), то в обращении записывается лишь второй адрес, т. е. название блока.

Объем усеченного конуса будем находить как разность объемов двух полных конусов. Для того чтобы это сделать, следует предва-

рительно определить высоту большего конуса H. Поэтому блок *Усеченный конус* мы построим как собирающую, состоящую из обращения к блоку *Высота*, вычисляющему H, двух обращений к блоку *Конус*, со входными ячейками ρ (радиус основания) и h (высота) и выходной v_K, и команды вычитания объемов большего и меньшего конусов:

Усеченный конус

Ω = *конец*

Высота

$H = h$

$R = \rho$

Конус

$v_K = v_{зап}$

$H - l = h$

$r = \rho$

Конус

$v_{зап} - v_K = v_{УК}$

конец.

Напишем теперь блоки *Высота* и *Конус*. Из подобия треугольников, получаемых в сечении малого и большого конусов, имеем

$$H = \frac{lR}{R-r}.$$

Поэтому блок *Высота* записывается в виде

Высота

Ω = *конец*

$l \cdot R = k_1$

$R - r = k_2$

$k_1 : k_2 = H$

конец.

Объем конуса определяется как одна треть площади основания, умноженная на высоту:

Конус

Ω = *конец*

$\pi \cdot \rho = q$

$q \cdot \rho = q$

$q \cdot h = q$

$q \cdot «\frac{1}{3}» = v_K$

конец.

Таким образом, программа нахождения объема усеченного конуса состоит из собирающей (*Усеченный конус*) и двух блоков — *Высота* и *Конус*.

Составим теперь блок *По цилиндру*, работающий лишь при $v \geqslant v_{УК}$. В этом блоке следует вычислить площадь основания цилиндра S, разделить вычисленный в собирающей *Резервуар* излишек объема Δv на S и сложить эту величину с l:

По цилиндру

Ω = *конец*
$\pi \cdot R = S$
$S \cdot R = S$
$\Delta v : S = \lambda$
$\lambda + l = \lambda$
конец.

При $v < v_{УК}$ должен работать блок *По конусу*. При составлении этого блока мы воспользуемся тем обстоятельством, что в блоке *Усеченный конус* уже вычислены высота h и объем v_K малого конуса.

Между искомой величиной λ и известными величинами v, v_K, h имеет место соотношение

$$v + v_K = \frac{1}{3}\pi\rho_1^2(h + \lambda),$$

где ρ_1 — радиус поверхности жидкости. Из подобия треугольников имеем

$$\frac{\rho_1}{r} = \frac{h + \lambda}{h}.$$

Поэтому

$$v + v_K = \frac{1}{3}\pi r^2 \frac{(h + \lambda)^3}{h^2}.$$

Отсюда находим

$$\lambda = \sqrt[3]{\frac{3(v + v_K)h^2}{\pi r^2}} - h.$$

Блок *По конусу*, вычисляющий по этой формуле λ, содержит обращение к библиотечной подпрограмме *Извлечение кубического*

корня:

По конусу

$$\Omega = \textit{конец}$$
$$v + v_K = t_1$$
$$\text{«3»} \cdot t_1 = t_1$$
$$t_1 : \pi = t_1$$
$$h : r = t_2$$
$$t_1 \cdot t_2 = t_3$$
$$t_3 \cdot t_2 = \alpha$$
$$\sqrt[3]{\ }$$
$$\gamma - h = \lambda$$

конец.

Таким образом, программа нахождения уровня жидкости в резервуаре состоит из собирающей (*Резервуар*) и пяти блоков (*Усеченный конус*, *Высота*, *Конус*, *По цилиндру*, *По конусу*), из которых первый фактически также является собирающей.

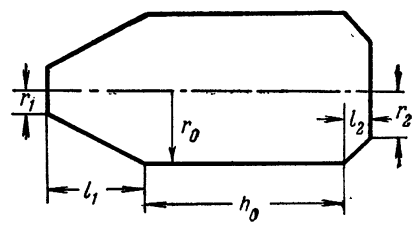


Рис. 40.

Пример 2.47. Тело, имеющее форму цилиндра с примыкающими к нему двумя усеченными конусами, погружено в жидкость плотности Δ. Размеры тела (в поперечном сечении) указаны на рис. 40. Плотность левого усеченного конуса d_1, правого — d_2, цилиндра — d_0. Найти силу F, действующую на это тело.

По известному закону Архимеда

$$F = P - \Phi,$$

где P — вес тела, а Φ — вес вытесненной жидкости.

Очевидно,

$$P = Mg, \qquad \Phi = mg,$$

где M — масса тела, m — масса жидкости.

Обозначим через v объем тела, а через v_0, v_1, v_2 — объемы цилиндра, левого и правого усеченных конусов. Тогда имеем

$$m = v\Delta,$$
$$M = v_0 d_0 + v_1 d_1 + v_2 d_2.$$

Для вычисления по этим формулам нужно знать величины v, v_0, v_1, v_2. Поэтому собирающая нашей программы, которую мы назовем *Архимед*,

записывается в виде

Архимед | *Объем*
Сила
стоп.

Здесь словом *Объем* обозначен блок, в котором находятся объемы v_0, v_1, v_2, v, а словом *Сила* — блок, который вычисляет силу F по написанным формулам.

Напишем сначала этот блок:

Сила | $\Omega = конец$
$v_0 \cdot d_0 = R_0$
$v_1 \cdot d_1 = R_1$
$v_2 \cdot d_2 = R_2$
$R_0 + R_1 = M$
$M + R_2 = M$
$v \cdot \Delta = m$
$m \cdot g = \Phi$
$M \cdot g = P$
$P - \Phi = F$
конец.

В блоке *Объем* должны вычисляться объемы двух усеченных конусов и цилиндра. Поэтому этот блок мы оформим как собирающую, содержащую обращение к двум блокам: *Цилиндр* и *Усеченный конус*. При этом блок *Усеченный конус* можно не писать, а взять его готовым из предыдущего примера.

Объем | $\Omega = конец$
Цилиндр
$l_1 = l$
$r_1 = r$
$r_0 = R$
Усеченный конус
$v_{УК} = v_1$
$l_2 = l$
$r_2 = r$
Усеченный конус
$v_{УК} = v_2$

$$v_0 + v_1 = v$$
$$v + v_2 = v$$
конец.

Блок *Цилиндр*, вычисляющий объем цилиндра v_0, состоит всего из нескольких команд:

Цилиндр | $\Omega = $ *конец*
$\pi \cdot r_0 = S_1$
$S_1 \cdot r_0 = S_1$
$S_1 \cdot h_0 = v_0$
конец.

В этом примере мы встретились со случаем, когда блок, написанный для программы решения одной задачи, без всякого изменения переносился в программу другой задачи.

В предыдущих главах при составлении программ мы под промежуточные результаты счета занимали рабочие ячейки R_0, R_1, R_2, ..., причем в одни и те же ячейки помещали совершенно различные величины. При блочном программировании во всех рассмотренных примерах мы для каждого блока выделяли свои рабочие ячейки, а для промежуточных величин также брали свои ячейки, обозначая их наиболее естественным образом. Такой способ выбора рабочих ячеек делает программу более наглядной и упрощает ее проверку. Правда, он требует дополнительных рабочих ячеек, но к нему следует прибегать всегда, когда программа свободно размещается в памяти.

§ 48. Работа с внешней памятью

Трехадресная машина, которую мы рассматриваем, имеет довольно большой объем памяти — 4096 сорокапятиразрядных ячеек. Однако во многих задачах этого количества ячеек не хватает для того, чтобы разместить всю необходимую информацию.

Так, в некоторых задачах приходится обрабатывать таблицы, содержащие много тысяч чисел, в процессе решения других задач появляется большое количество промежуточных результатов, третьи имеют такой громоздкий алгоритм решения, что в памяти не помещается программа.

Память машины, о которой мы говорили до сих пор, является оперативной, так как она непосредственно обменивается информацией с арифметическим устройством. Назовем оперативное запоминающее устройство *внутренней памятью*. У машины, наряду с внутренней памятью, существует еще и *внешняя память*. Внешняя память связана с арифметикой только через внутреннюю память.

Внешняя память машины может в свою очередь состоять из нескольких различных устройств. Обычно она осуществляется на маг-

нитных барабанах или лентах типа магнитофонных. Емкость барабана, как правило, бывает равна объему оперативной (внутренней) памяти, емкость магнитной ленты во много раз больше.

Для простоты описания предположим, что машина имеет один барабан, т. е. внешняя память имеет, как и внутренняя, 4096 ячеек, которые также нумеруются от 0000 до 7777_8.

Перенос информации из внутренней памяти во внешнюю называется *записью*, а из внешней памяти во внутреннюю — *чтением*. На рис. 41 представлена схема взаимодействия запоминающих устройств и арифметики машины.

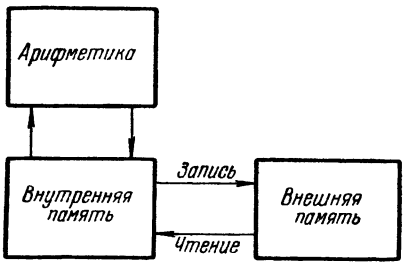


Рис. 41.

Элементарные операции, осуществляющие перенос информации из одних запоминающих устройств машины в другие, называются *операциями обмена*.

Рассмотрим две операции обмена: *запись* и *чтение*. Первая из этих операций имеет вид

Запись a_1 b_1 a_n.

По этой операции содержание массива ячеек внутренней памяти от a_1 до a_n записывается в ячейки внешней памяти, начиная с b_1.

При помощи второй операции

Чтение a_1 b_1 a_n

содержание массива n ячеек внешней памяти, начинающейся с ячейки b_1, переносится в ячейки внутренней памяти от a_1 до a_n.

Коды этих операций следующие:

Запись 50

Чтение 70

Правильность выполнения операций обмена контролируется в машине следующим образом. Все слова обмениваемого массива в процессе выполнения соответствующей операции (записи или чтения) суммируются в арифметическом устройстве при помощи операции циклического сложения, описанной нами в § 28. При записи, вслед за последней ячейкой записанного массива b_n, записывается циклическая сумма массива от a_1 до a_n.

Следовательно, во внешней памяти оказываются занятыми не n, а $n+1$ ячеек — b_1, b_2, ..., b_n, b_{n+1}. При чтении записанного таким образом массива, сумма, накопленная в процессе чтения в арифметическом устройстве, по-разрядно сравнивается с содержанием ячейки b_{n+1} внешней памяти. Если эти слова совпадают, то выполняется следующая

за командой *чтение* команда. Если же они не совпадают хотя бы в одном разряде, машина останавливается. Отметим, что этот *стоп* может свидетельствовать либо о неисправности машины (неправильное выполнение операций обмена), либо об ошибке в программе. Так, например, может быть случай, когда в операции *чтение* неправильно указывается начало массива во внешней памяти. Тогда, естественно, контрольная сумма и содержание ячейки, следующей за считываемым массивом, не совпадут, и машина остановится.

К операциям обмена прибегают лишь тогда, когда вся нужная для решения задачи информация не умещается во внутренней памяти машины.

Пример 1.48. Задано число x. В последовательности

$$z_1 = \operatorname{tg} x, \quad z_2 = \operatorname{tg} 2x, \quad \ldots, \quad z_{4000} = \operatorname{tg} 4000x$$

найти количество ν чисел, бо́льших их среднего арифметического:

$$z = \frac{z_1 + z_2 + \ldots + z_{4000}}{4000}.$$

Предположим, что этот пример является заключительной частью некоторой более сложной задачи, в которой сначала находится x, а затем ν, причем программа вычисления числа x, вместе с библиотекой стандартных подпрограмм, занимает бо́льшую часть внутренней памяти, так что для нахождения ν в нашем распоряжении остается лишь несколько сотен ячеек. Число ν можно получить одним из двух способов:

1) Подсчитать z, не сохраняя членов последовательности $z_1, z_2, \ldots, z_{4000}$. После этого заново находить числа последовательности, сравнивая их с z.

2) Подсчитывая z, сохранять все члены последовательности.

Первый из этих способов дает возможность разместить всю программу во внутренней памяти, но требует двукратного расчета последовательности. Во втором способе последовательность рассчитывается один раз, однако придется ее (по частям) переносить во внешнюю память.

Рассмотрим второй способ *).

Разобьем всю последовательность $z_1, z_2, \ldots, z_{4000}$ на восемь частей равной длины (по 500 элементов), предполагая, что такая часть поместится в оперативной памяти и останется место для размещения программы. Будем поочередно находить эти подпоследовательности и переносить их во внешнюю память. Для этого нам потребуется во внутренней памяти рабочее поле в 500 ячеек, которые мы обозначим буквами $\rho_1, \rho_2, \ldots, \rho_{500}$.

При распределении внешней памяти следует принимать во внимание, что для каждой подпоследовательности потребуется 501 ячейка:

*) Рекомендуем читателю постараться реализовать первый способ самостоятельно.

500 для ее членов и одна — для контрольной суммы. Условимся помещать первую подпоследовательность в ячейки внешней памяти с адресами от 0000 до $500_{(10)} = 0764_{(8)}$, вторую подпоследовательность в ячейки от $501_{(10)} = 0765_{(8)}$ до $1001_{(10)} = 1755_{(8)}$ и т. д.

Собирающую нашей задачи запишем в виде:

Собирающая | *Среднее*
Счет ν
стоп.

В блоке *Среднее* рассчитываются и переносятся во внешнюю память восемь подпоследовательностей, а также находится среднее арифметическое. В блоке *Счет* ν подпоследовательности переносятся из внешней памяти во внутреннюю и определяется число ν.

При принятом разбиении последовательности $z_1, z_2, \ldots, z_{4000}$ на части подсчет среднего удобнее производить по формуле

$$z = \frac{u_1 + u_2 + \ldots + u_8}{4\,000},$$

где $u_1, u_2, \ldots, u_8$ — суммы членов подпоследовательностей:

$$u_1 = z_1 + z_2 + \ldots + z_{500}$$
$$u_2 = z_{501} + z_{502} + \ldots + z_{1000}$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$u_8 = z_{3501} + z_{3502} + \ldots + z_{4000}.$$

Блок *Среднее* напишем в виде цикла, работающего восемь раз:

```
Среднее |  Ω ≐ конец
        |  0 = z
        |  0 = n
        |  0 = S
        |  РА 0* 0 Предконец
        |  Подпоследовательность
        |        Туда
        |        u + z = z
        |  РА < 7̄  | 1̄*
        |  z : «4000» = z
        |        Предконец
        |        конец.
```

Этот блок сам является собирающей, содержащей обращение к другим блокам. Блок *Подпоследовательность* вычисляет 500 членов очередной подпоследовательности и находит их сумму *u*. Блок *Туда*

переносит эту подпоследовательность из рабочего поля во внешнюю память. В начале собирающей *Среднее* очищаются ячейки n и S, являющиеся счетчиками блоков *Подпоследовательность* и *Туда*.

Блок *Подпоследовательность* является по сути дела единственным счетным блоком нашей программы и имеет вид

```
Подпоследовательность |  Ω = конец
                      |  PA  0*    0   Предконец
                      |            0 = u
                      |          n + «1» = n
                      |          n · x   = a
                      |          cos, sin
                      |          γ1 : γ0  = ρ1*
                      |          u +  ρ1* = u
          PA < 763    |                   1̄*
                Предконец
                конец.
```

Этот блок работает в программе восемь раз. Перед его первым прохождением счетчик $n = 0$, и в цикле на рабочем поле $\rho_1, \rho_2, \ldots, \rho_{500}$ подсчитываются члены первой подпоследовательности $z_1, z_2, \ldots, z_{500}$. После выхода из блока в ячейке u оказывается сумма этих величин, а в n число «500». Это число является входной величиной для расчета следующей подпоследовательности при втором обращении к рассматриваемому блоку.

Перейдем теперь к составлению блока *Туда*. В этом блоке при каждом прохождении цикла собирающей *Среднее* мы должны переносить содержимое рабочего поля $\rho_1, \rho_2, \ldots, \rho_{500}$ во внешнюю память. Поэтому основная рабочая команда блока имеет вид

Запись ρ_1 b_1 ρ_{500}.

Согласно принятому распределению внешней памяти при записи во внешнюю память первой подпоследовательности b_1 должно равняться нулю, при записи второй подпоследовательности $b_1 = 501_{(10)} = 0765_{(8)}$, для третьей подпоследовательности $b_1 = 1002_{(10)} = 1752_{(8)}$ и т. д. Таким образом, значение b_1 должно увеличиваться на $501_{(10)} = 765_{(8)}$ при каждом новом прохождении блока *Туда*. Такую переадресацию мы организуем при помощи PA следующим образом:

```
Туда  |  Ω = конец
      |  [PA]   0*    S    BP
      |  Запись ρ1    0*   ρ500
      |  [PA]   765*  BP   S
              конец.
```

При этом ячейка, которую мы обозначим *BP*, служит для запоминания предыдущего состояния *PA* и его восстановления в конце работы блока.

Перейдем теперь к составлению блока *Счет* ν. Этот блок, так же как и блок *Среднее*, состоит из обычного цикла:

```
Счет ν |        Ω = конец
       |        0 = ν
       |        0 = σ
       | PA   0*  0   Предконец
       |      Обратно
       |      Сравнение
       | PA < 7̄          1̄*
       |      Предконец
       |      конец.
```

В блоке *Обратно* восемь подпоследовательностей по очереди переносятся из внешней памяти во внутреннюю, ячейка σ служит счетчиком этого блока. В блоке *Сравнение* накапливается число ν при помощи сравнения с членами очередной подпоследовательности. Блок *Обратно* пишется аналогично блоку *Туда*:

```
Обратно |        Ω = конец
        | [PA]   0*  σ   BP
        | Чтение ρ1  0*  ρ500
        | [PA]   765̄* BP  σ
        |        конец
```

Блок *Сравнение* представляет собой простой цикл:

```
Сравнение |        Ω = конец
          | PA 0*    0   Предконец
          |   z  —  ρ1* = 0
          | У0   Я + 2
          |   ν + «1» = ν
          | PA < 763̄ |  1̄*
          |          Предконец
          |          конец.
```

Как мы видели, рассматриваемую задачу можно было решить двумя способами: при помощи только внутренней памяти и с исполь-

зованием внешней памяти. Эти два способа не равноценны по быстродействию. В нашем случае члены последовательности подсчитываются довольно просто. Поэтому повторный расчет последовательности, требуемый в первом способе, не приведет к серьезной потере времени по сравнению со вторым. Однако если бы последовательность рассчитывалась более сложно, фактор времени заставил бы нас безусловно предпочесть разобранный нами второй способ первому.

Отметим, что во многих громоздких задачах не существует двух способов их решения на машине: с внешней памятью и без нее, а существует лишь один — с использованием внешней памяти. Так, если во внутренней памяти не помещается программа решения задачи или исходные данные — таблицы (а не промежуточные величины), то приходится часть этой информации помещать во внешнюю память. Такого рода информация не образуется в процессе решения задачи, а готовится заранее на программных бланках и наносится на перфокарты. Слова, нанесенные на перфокарты, посредством устройства ввода переносятся в оперативную память машины. Непосредственной связи между устройством ввода и внешней памятью не существует. Поэтому, если информация, нанесенная на перфокарты, не помещается во внутренней памяти, часть информации вводят во внутреннюю память и переписывают (при помощи операции *Запись*) во внешнюю. Такого рода последовательность операций повторяют нужное число раз. Связь между перфокартами, внутренним и внешним запоминающими устройствами характеризуется блок-схемой, изображенной на рис. 42.

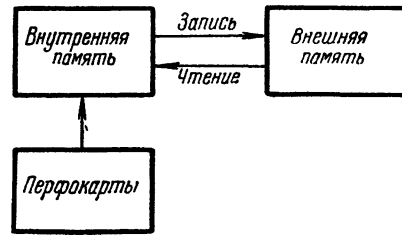


Рис. 42.

Ввод информации с перфокарт во внутреннюю память может производиться также и программным путем с помощью специальной элементарной операции, записываемой так:

Ввод *a*, 0, 0.

При выполнении этой команды обычные слова, набитые на перфокартах, находящихся в устройстве ввода, вводятся в ячейку памяти подряд, начиная с *a*. Если же перфокарта начинается с адресного слова, то адрес *a* игнорируется и ввод происходит в соответствии с набитым на карте адресным словом.

В процессе ввода обычные и адресные слова, набитые на перфокартах, автоматически суммируются операцией циклического сложения. Ввод перфокарт прекращается в тот момент, когда в устройство

ввода поступает контрольное слово, которое сравнивается с суммой, накопленной ранее. При несовпадении машина останавливается, а при совпадении — передает управление следующей за командой ввода ячейке программы.

Необходимо иметь в виду, что описанная проверка контролирует лишь правильность прочтения вводимого материала, но никак не проверяет правильность его записи в память. Правильность ввода программы в память необходимо проверять другими способами. Речь об этом будет идти в следующей главе.

ГЛАВА X

ОТЛАДКА ПРОГРАММЫ

§ 49. Подготовка программы к отладке

Процесс решения задачи при помощи машины можно разбить на ряд этапов:

I) разработка алгоритма и составление программы в содержательных обозначениях;

II) распределение памяти и кодирование;

III) перфорация;

IV) отладка программы;

V) счет.

Математик, приступая к задаче, должен иметь в виду, что программа является одним из наиболее естественных способов записи алгоритма решения задачи. Поэтому мы и объединили в одном пункте создание алгоритма и написание программы. Математик, разрабатывая отдельные части алгоритма, записывает их в виде блоков программы. Эти отдельные части алгоритма, блоки, он объединяет вместе при помощи связующей части алгоритма — блок-программы.

На втором этапе прежде всего при помощи памятки происходит распределение памяти. В тех случаях, когда задача не помещается во внутренней памяти, распределение материала между внутренней и внешней памятью становится существенной частью первого этапа решения задачи — разработки машинного алгоритма-программы.

В громоздких задачах, наряду с памятками на восьмеричную тысячу ячеек, употреблявшимися нами ранее, используют еще памятки на всю внутреннюю память. На таких памятках удобно помечать расположение в памяти библиотечных подпрограмм, крупных блоков, таблиц и т. п. При помощи памятки и таблички кодов операций программу кодируют.

На третьем этапе закодированную программу наносят на перфокарты. Полученную колоду перфокарт дублируют и отпечатывают на бумажную ленту. Эту ленту для контроля считывают с правой частью программы. После третьего этапа работы алгоритм решения задачи оказывается превращенным в колоду перфокарт.

Казалось бы, теперь математику для решения задачи достаточно ввести программу в память машины, передать управление на ее начальную команду и подождать пока на бумажной ленте напечатаются результаты счета. Однако на самом деле такую идеальную работу почти никогда не приходится видеть.

При первом выходе на машину никогда нельзя быть уверенным в том, что в программе нет ошибок. Поэтому счету, т. е. непосредственному решению задачи на машине, всегда предшествует отладка программы. *Отладка программы есть выявление и исправление ошибок в программе.*

Как мы уже говорили, при выходе на машину программа является колодой перфокарт. Ошибки в этой колоде могут появиться на первых трех этапах решения задачи. Отметим прежде всего, что кодирование и перфорация являются чисто техническим делом и при надлежащей организации работы ошибки на этих этапах могут быть почти полностью устранены. В тех случаях, когда программисту «не тесно» во внутренней памяти, в распределении памяти также обычно не бывает ошибок. Поэтому *отладка программы в основном состоит из выявления ошибок в алгоритме и в программе задачи.*

Перед выходом на машину каждый блок программы следует тщательно проверить путем детального просмотра. Однако такая проверка «на глаз» недостаточна для выявления всех ошибок.

К отладке следует детально подготовиться. *Основной составной частью такой подготовки является ручной расчет.*

Ручной расчет следует производить *по-блочно.* Для этого берут сначала самые внутренние блоки программы, задают для блока определенные исходные данные и рассчитывают результаты.

Для проверяемого блока пишут отладочную программу, содержащую засылку во входные ячейки блока исходных данных ручного расчета, обращение к блоку и команды отладочной печати. Эти команды содержат обращения к библиотечной подпрограмме печати чисел, при помощи которых выводятся на печать исходные данные, промежуточные величины и результаты работы блока. Если, как мы рекомендовали выше, не совмещать ячейки для более или менее существенных промежуточных величин, то по такой печати можно во всех деталях разобрать работу блока. Если в блоке имеются переменные команды, то полезно при помощи библиотечной программы *Печать программы* вывести на печать команды блока.

Заметим, что полный ручной расчет в силу громоздкости возможен не для всех блоков. Сплошь и рядом, например, встречаются блоки, содержащие циклы, работающие сотни, а то и тысячи раз. В этом случае проделывают ручной расчет для двух-трех шагов цикла, и на время отладки вносят изменение в проверяемый блок, заставляя его работать нужное число раз. В арифметическом блоке это можно сделать, меняя на время отладки соответствующим обра-

зом начальное (или конечное) значение счетчика или регистра адреса. В итерационном цикле для этого следует значительно уменьшить требуемую точность, тогда число итераций станет небольшим.

После ручного расчета для внутренних блоков проделывают расчеты и для блоков, использующих внутренние. Для этих блоков также пишут отладочные программы. При этом в отладочной программе, наряду с ячейками, характеризующими работу внешнего блока, полезно выдать на печать и содержимое ячеек, используемых в работе внутренних блоков, входящих во внешний. Если эти ячейки для разных внутренних блоков различны, мы по отладочной печати внешнего блока можем составить себе представление о совместной работе внешнего и внутренних блоков.

Для проверки программы всей задачи полезно сделать полный ручной расчет для одного из вариантов исходных данных *). Ручной расчет как для отдельных блоков, так и для задачи в целом следует делать в две руки и с большим числом знаков.

Для отладки программы необходимо хорошо знать работу за пультом машины. Описанию пульта будет посвящен следующий параграф.

§ 50. Пульт машины

Пульт машины состоит из двух панелей: горизонтальной и вертикальной, на которых расположено большое количество кнопок, тумблеров, клавиш, переключателей и лампочек. Мы опишем здесь лишь те части пульта, которые нужны программисту **).

На рис. 43 изображена схема вертикальной панели. В средней части вертикальной панели находятся несколько горизонтальных рядов из 45 неоновых лампочек. Горящая лампочка имеет значение двоичной единицы, негорящая — значение нуля. Нижний ряд из 45 лампочек называется *Регистр команд*; в нем горит команда, исполняющаяся в данный момент. Выше этого регистра расположены еще три регистра — *РI*, *РII* и *РIII*, в которых горит содержимое ячеек по I, II и III адресу исполняющейся команды.

В правой части вертикальной панели расположены 12-разрядные ряды лампочек: *Счетчик команд* и *Регистр адреса* (*РА*). В счетчике команд находится адрес команды, исполняющейся в данный момент, а в *РА* значение регистра адреса в тот же момент. Кроме того, на вертикальной панели находятся лампочки — управляющий сигнал ω и сигнал аварийной остановки — *авост*.

---

*) Организация такого расчета является довольно сложным делом, которое трудно стандартизовать.

**) Ряд частей пульта предназначен специально для инженеров, обслуживающих машину.

Состояние описанных регистров полностью характеризует выполнение отдельной команды в машине. Поэтому их можно использовать для детальной, покомандной проверки программы.

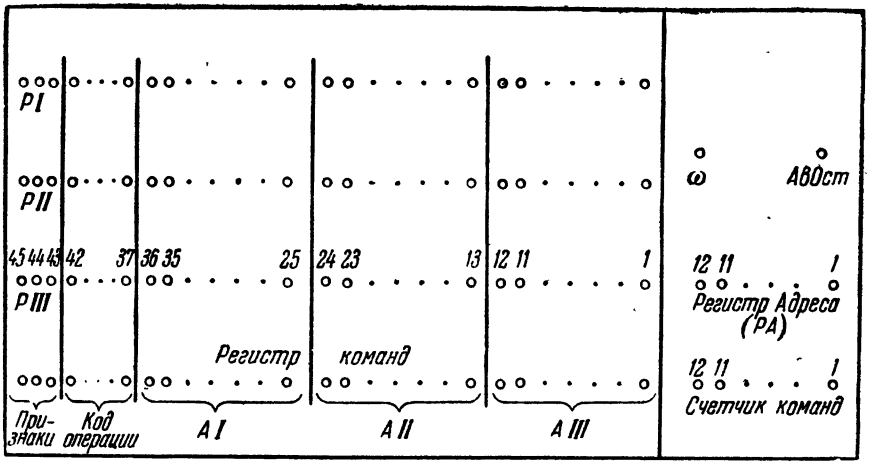


Рис. 43.

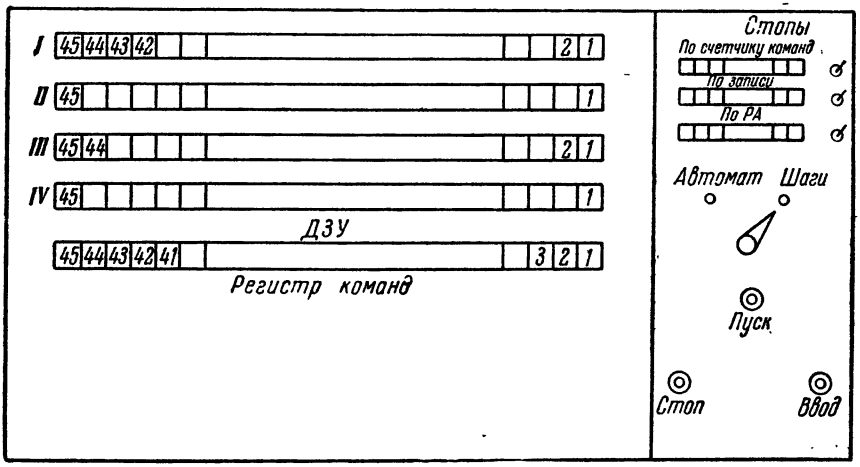


Рис. 44.

На горизонтальной панели (рис. 44) находится ряд тумблеров, кнопок и переключателей, нужных для ручного управления работой машины.

1. Кнопка *Ввод*. При ее нажатии начинает работать устройство ввода, и материал с перфокарт, стоящих на этом устройстве ввода, поступает во внутреннюю память машины.

2. Кнопка *Пуск*, нажатие которой включает работу машины.

3. *Регистр команд пульта* — ряд из 45 клавишей, на которых можно набрать команду, которую нужно с пульта исполнить.

4. *Переключатель режимов работы машины*. Он может занимать два положения — *шаговый режим* и *автоматический режим*. При работе в шаговом режиме (*на шагах*) при нажатии кнопки *Пуск* машина выполняет очередную команду, после чего останавливается; чтобы выполнить следующую команду, нужно еще раз нажать кнопку *Пуск*. При положении переключателя *автоматический режим* машина после нажатия кнопки *Пуск* выполняет команды программы автоматически (*на автомате*), останавливаясь либо по команде *стоп*, либо в результате переполнения разрядной сетки машины (аварийная остановка).

5. Кнопка — *Стоп*; служит для остановки работы машины или устройства ввода.

Указанных устройств пульта достаточно для того, чтобы ввести программу в машину и заставить ее работать. Для этого ставят в устройство ввода колоду перфокарт с программой, затем нажимают кнопку *Ввод*. Программа вводится во внутреннюю память машины. Устройство ввода прекращает работу, когда под его щетками окажется контрольное слово. Если в конце вводимого материала нет контрольного слова, то после того как все карты прошли через ввод, его работу нужно остановить кнопкой *Стоп*.

После этого на регистре команд пульта набирают команду передачи управления на начало программы:

$$16 \quad N_{\text{нач}},$$

где $N_{\text{нач}}$ — адрес начальной команды программы. Переключатель режимов ставят в положение *на автомате* и нажимают кнопку *Пуск*. На регистр команд переносится набранная команда

$$16 \quad N_{\text{нач}},$$

которая затем исполняется.

В счетчик команд заносится адрес $N_{\text{нач}}$ и машина начинает автоматически выполнять команды программы. При остановке машины по состоянию регистров вертикальной панели пульта можно определить характеристики последней исполнявшейся команды: на счетчике команд — ее адрес, на регистре команд — саму команду, на регистрах *P*I, *P*II, *P*III — содержимое ячеек по I, II и III адресам, на *PA* — регистр адреса, на ω — значение управляющего сигнала.

Помимо указанных устройств на горизонтальной панели имеются еще устройства, используемые специально для отладки программы и оперативной работы за пультом.

6. Три *тумблера отладочных стопов*, включение которых дает возможность остановить работу машины в нужном месте. Возле каждого тумблера имеется ряд из двенадцати клавиш, в котором можно набрать двенадцатиразрядное двоичное слово.

Стоп по счетчику команд. При наборе на этом ряде клавиш адреса некоторой ячейки внутренней памяти, машина останавливается перед исполнением команды, лежащей в этой ячейке. На счетчике команд в момент стопа горит указанный адрес.

Стоп по записи. При наборе здесь адреса ячейки памяти машина останавливается перед исполнением команды, заносящей в эту ячейку какое-либо машинное слово.

Стоп по РА. Машина останавливается в тот момент, когда в регистр адреса заносится двенадцатиразрядное слово, равное набранному на этом ряду клавиш.

7. Дополнительные запоминающие устройства (ДЗУ).

Каждое из дополнительных запоминающих устройств представляет собой ряд из 45 клавишей, в котором может быть набрано 45-разрядное двоичное слово. Предполагается, что на пульте имеется четыре таких устройства и что им присвоены адреса 7771, 7772, 7773 и 7774 ячеек памяти машины. Их обозначают ДЗУ-1, ДЗУ-2, ДЗУ-3, ДЗУ-4. Таким образом, у машины имеется еще одно запоминающее устройство из четырех ячеек, содержание которых можно оперативно (вручную) менять на пульте машины. Машина в процессе выполнения команд программы может брать с ДЗУ информацию, как из обычных ячеек памяти. Запись в эти ячейки памяти из машины невозможна.

§ 51. Проверка работы программы на машине

Проверку работы программы на машине следует начинать с отладки самых внутренних блоков. Для этого, кроме основной программы, вводят в машину отладочные программы для отдельных блоков и передают управление на отладочную программу первого из проверяемых блоков. Результаты работы блока, выданные на печать, сверяют с ручным расчетом. Если результаты совпадают, то блок считается отлаженным. В случае несовпадения каких-либо результатов сверяют напечатанные исходные данные и промежуточные величины с теми, которые были получены при ручном расчете. По несовпадающим промежуточным величинам обычно нетрудно найти участок блока, в котором имеется ошибка.

Если ошибку не удается обнаружить путем проверки блока «на глаз», можно поступить следующим образом. Пусть, например, участок блока от начальной команды K_1 до некоторой команды K_2 работает правильно (все промежуточные величины, вычисляемые на этом участке, выданы на печать такими же, какими они были при

ручном расчете), а на участке от K_2 до K_3, по-видимому, имеется ошибка. Тогда составляют такую отладочную программу:

1)	*БВ Я+1*	*Подготовка*		Ω
2)		K_3	$=$	*Запас*
3)	*БВ Я+1*	K_2		K_3
4)	*БВ Я+2*	*Печать чисел*		Ω
5)	a_1	0		a_n
6)		*Запас*	$=$	K_3
7)	*БВ Я+1*	K_3	*Конец блока*	
8)	*БВ Я+1*	*Печать чисел*		Ω
9)	a_1	0		a_n
10)	*стоп.*			

В этой программе команда 1) обращается к блоку, готовящему исходные данные для работы части отлаживаемого блока от команды K_2 до его конца. При помощи команд 2) и 6) выводится в запас и восстанавливается команда K_3, которая портится командой 3), обращающейся к участку $K_2 - K_3$ отлаживаемого блока*). Вывод на печать при помощи команд 4), 5), 8), 9) массива чисел $a_1 - a_n$ дает возможность проанализировать работу блока на двух участках $K_2 - K_3$ и K_3 — конец блока.

Ошибку в промежуточных или окончательных результатах работы блока иногда можно обнаружить еще следующим способом. Пусть нам нужно проверить образование ячейки a. Наберем ее адрес на клавишах отладочного *Стопа по записи* и включим тумблер этого стопа. Тогда при работе блока машина будет останавливаться перед выполнением каждой команды, заносящей некоторое содержимое в ячейку a. По регистрам вертикальной панели пульта можно проанализировать изменение содержимого этой ячейки. При помощи *Стопа по записи* можно проследить, в частности, как изменяются переадресуемые команды.

Такими способами можно детально проверять работу всех внутренних блоков. При этом следует иметь в виду, что если в блоке имеются разветвления вычислительного процесса, то нужно проверить блок для нескольких вариантов исходных данных. Эти варианты должны составляться так, чтобы в программе проходились все возможные пути.

Найденные в блоках ошибки необходимо исправить в программных бланках и сразу же перебить и заменить соответствующие перфокарты.

---

*) Заметим, что команда K_3 при указанном способе отладки не должна быть переменной.

После того как отлажены внутренние блоки, составляют отладочные программы для проверки внешних блоков. При несовпадении результатов работы внешнего блока (собирающей) на машине с ручным расчетом часто бывает полезно проверить правильность следования обращений к внутренним блокам. Для этого на клавишах отладочного *Стопа по счетчику команд* набирают адрес команды обращения к внутреннему блоку, включают тумблер этого стопа и передают управление на начало собирающей. Машина должна остановиться перед выполнением соответствующей команды обращения. Подобным же образом проверяется последовательность обращений и к другим внутренним блокам.

До сих пор мы предполагали, что машина проходит отлаживаемый блок до конца. Однако далеко не всегда так бывает. Здесь могут встретиться следующие случаи:

1. Машина продолжает вычисления без конца, не останавливаясь и совершая вычисления по некоторому замкнутому циклу *(зацикливание).*

2. Машина останавливается при переполнении разрядной сетки машины *(аварийный стоп) внутри блока.*

3. Машина останавливается по аварийному или обычному стопу *вне блока.*

К зацикливанию могут привести довольно разнообразные ошибки. Это может случиться при неверном изменении счетчика цикла. В этом случае можно обнаружить ошибку, набрав на клавишах *Стопа по записи* адрес этого счетчика и прослеживая его изменение при работе цикла. Если цикл меняется по регистру адреса, может помочь пультовый *Стоп по РА.* При помощи него можно проследить изменение РА в цикле. Часто бывает полезно использовать и *Стоп по счетчику команд,* набирая на его клавиатуре адрес команды, содержащей проверку условия окончания цикла.

Заметим, что эти рекомендации помогают только тогда, когда мы знаем, в каком блоке есть ошибки. Обнаружить это можно, проходя весь отлаживаемый блок или его части на шагах. Если блок является собирающей, то следует проходить собирающую на автомате, набирая на *Стопе по счетчику команд* адреса обращений к последовательным блокам. При зацикливании в одном из этих блоков машина не выйдет на следующий стоп. Если же все стопы «сработали», нужно искать зацикливание в собирающей.

Случай выхода на аварийный стоп, по-существу, эквивалентен получению неверного результата. Неверное значение имеет та величина, адрес которой равен третьему адресу команды, обусловившей аварийный стоп. Для нахождения причины ошибки следует пройти на шагах отлаживаемый блок, начиная от команды, в которой начинается вычисление соответствующей величины, и до команды, приводящей к аварийному стопу. При этом следует прослеживать, как

выполняется каждая команда по регистрам вертикальной панели пульта. С третьим случаем (стоп вне проверяемого блока) приходится сталкиваться тогда, когда имеются ошибки, приводящие к неверным передачам управления. При неверной передаче управления наружу блока машина либо начинает выполнять «не те» команды, либо начинает воспринимать как команды двоичные числа или константы. Ошибки такого рода обнаруживают при помощи пультового *Стопа по счетчику команд*, останавливая машину по очереди на последовательных командах передачи управления.

Встречаются ошибки, когда передача управления происходит на ячейки, не занятые программой. Для обнаружения таких ошибок применяют следующий прием: перед вводом программы и началом работы производят не очистку внутренней памяти нулями, а заполнение всех ее ячеек машинным словом:

$$Щ = (777,\ F,\ F,\ F) = 777\ 7777\ 7777\ 7777.$$

Это делается при помощи специальной программы *«Роспись Щ»*. После этого вводят свою программу.

Теперь, если при работе программы произойдет передача управления на ячейку, не занятую программой, в регистр команд поступит *Щ*. Это машинное слово воспринимается как команда *Стоп*. Поэтому машина остановится, сигнализируя об ошибке в программе. Заметим, что указанный прием имеет еще одно применение. *Щ*, как двоичное число, есть наибольшее по модулю число, какое можно записать в машине. Поэтому если арифметическая команда возьмет слово *Щ* в качестве числа, то в большинстве случаев произойдет аварийный стоп в результате переполнения разрядной сетки машины. Это обстоятельство дает возможность во многих случаях обнаружить ошибки в адресах арифметических команд.

При отладке программы за пультом машины сплошь и рядом программист оказывается в совершенно непредвиденной ситуации. Поэтому важно иметь удобные средства для оперативной работы за пультом. Так, например, очень полезно иметь возможность напечатать с пульта массив чисел или участок программы. Важно также уметь быстро и надежно исправлять с пульта ошибки в программе. При работе за пультом удобно пользоваться следующими тремя обслуживающими программами.

1. *Печать чисел с пульта.* Информацией для программы служат границы массива ячеек памяти, набираемые в первом и третьем адресах первого дополнительного запоминающего устройства пульта (ДЗУ-1).

После передачи управления на начало этой программы числа из указанного на ДЗУ-1 массива переводятся из двоичной системы в десятичную и печатаются. Напечатав последнее число, машина останавливается.

2. *Печать программы с пульта.* После передачи управления с пульта на начало этой программы содержание ячеек массива, адреса начала и конца которого указаны в ДЗУ-1, выдаются на печать в командном виде. По окончании печати машина останавливается.

3. *Поправка команды с пульта.* Во втором адресе ДЗУ-2 набирается адрес исправляемой команды, а в ДЗУ-1 сама команда. После передачи управления на начало программы следует стоп и содержимое ДЗУ-1 и ДЗУ-2 загорается в регистрах PI и PII вертикальной панели пульта. Проверив правильность набранной информации, следует нажать кнопку *Пуск*. Тогда произойдет исправление команды, исправленная команда напечатается, и машина остановится.

Этими тремя программами широко пользуются при отладке программ. Так, останавливаясь в «подозрительных» местах программы по пультовым отладочным стопам, выпечатывают при помощи указанных программ массивы рабочих ячеек или команд, после чего можно продолжать работу программы с того места, где она была остановлена.

Обнаружив ошибку, исправляют ее при помощи третьей из указанных программ. После отладки, по печати, выдаваемой этой программой, исправляют ошибки на перфокартах.

Отметим в заключение, что отладка программы является сложной работой, требующей высокой квалификации и не укладывающейся ни в какие твердые каноны. Все изложенное выше нужно рассматривать поэтому лишь как некоторые начальные рекомендации.

ГЛАВА XI

ОБЩИЕ СВЕДЕНИЯ ОБ ЭЛЕКТРОННЫХ ВЫЧИСЛИТЕЛЬНЫХ МАШИНАХ

§ 52. Вычислительные машины непрерывного и дискретного действия

По принципу своей работы все математические машины делятся на два больших класса: *машины непрерывного действия* (*моделирующие машины*) и *машины дискретного действия* (*цифровые машины*). В настоящем параграфе мы познакомимся с основными принципами работы математических машин различных классов.

В машинах и устройствах непрерывного действия числа представляются физическими величинами различного рода, которые могут меняться непрерывным образом. Это может быть длина отрезка или угол поворота вала, ток или напряжение в электрической цепи, температура, давление и т. п.

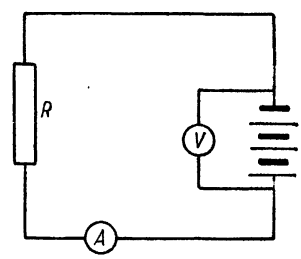


Рис. 45.

Простейшим примером вычислительного устройства непрерывного действия является *логарифмическая линейка*, на которой число изображается длиной отрезка. Работа линейки уже рассматривалась в § 7. Легко привести пример схемы, которую можно использовать для умножения и деления и в которой числа изображаются электрическими величинами.

Возьмем электрическую цепь, состоящую из батареи и омического сопротивления. Известно, что ток в такой цепи подчиняется закону Ома.

$$I=\frac{E}{R},$$

где I — ток и E — напряжение. Включив в цепь амперметр и вольтметр (рис. 45) и зная величину сопротивления R, мы можем по показаниям амперметра находить частное от деления E на R, а по показаниям вольтметра — произведение IR, придавая остальным величи-

нам нужные значения. Таким образом, эта простейшая электрическая схема может быть использована для умножения и деления, причем числа изображаются здесь электрическими величинами: напряжением, величиной тока и сопротивлением.

Из этого примера ясно, почему математические машины непрерывного действия называют *моделирующими машинами*. Машины непрерывного действия фактически не выполняют никаких вычислений. Они лишь воспроизводят (*моделируют*) процесс, который описывается данной функцией, уравнением или системой уравнений. Такое моделирование называют *математическим моделированием*, в отличие от физического моделирования, в котором изменяются лишь масштабы, но не физическая природа моделируемого процесса или явления.

Математические модели могут быть построены с использованием величин самой различной физической природы: механических, тепловых, гидродинамических, электрических, электромагнитных, электронных и т. п. Естественно, что чаще всего в математических машинах непрерывного действия используются электрические величины различного рода. Это объясняется компактностью соответствующих элементов, простотой их измерения и удобством осуществления нужных схем.

Особенно часто применяются машины непрерывного действия для решения различных *дифференциальных уравнений*, в которых ищется неизвестная функция по известной связи между нею и ее производными. Дело в том, что очень часто весьма различные физически процессы описываются математически одними и теми же уравнениями, что и позволяет моделировать одни процессы другими. Такая возможность является одним из проявлений единства окружающего нас мира. Это отмечал еще В. И. Ленин в своем замечательном произведении «Материализм и эмпириокритицизм». «Единство природы, — писал он, — обнаруживается в „поразительной аналогичности" дифференциальных уравнений, относящихся к различным областям явлений»*).

Существенным недостатком машин непрерывного действия является *сравнительно небольшая точность результатов*. Точность машины непрерывного действия определяется точностью измерительных приборов, измеряющих соответствующие физические величины, например, ток и напряжение в электрической цепи. Как правило, относительная ошибка измерения имеет порядок нескольких процентов или, в лучшем случае, нескольких десятых процента. Это означает, что в показаниях прибора, а значит, и в результатах, выданных машиной, верными можно считать две или три значащие цифры. Дальнейшее повышение точности вызывает уже серьезные затруднения.

Другим недостатком моделирующих машин является *их специализированность*. Строго говоря, для каждой задачи необходимо иметь свою схему, т. е. свою модель, а значит, и другую машину.

*) В. И. Ленин, Соч., изд. IV, т. 14, стр. 276.

Для изменения схемы используются переключения, но эти возможности ограничены типом основных схем, имеющихся в данной машине. По этой причине машина непрерывного действия не может иметь универсального характера и область ее применения всегда ограничена задачами определенного класса.

Совсем иначе изображаются числа в машинах дискретного действия. Простейшим примером дискретного вычислительного устройства являются хорошо известные *русские счеты*. Здесь число изображается в обычной цифровой форме в десятичной системе счисления. Для изображения каждого разряда числа отводится отдельная спица с надетыми на нее десятью косточками; чтобы изобразить какую-либо цифру, следует опустить соответствующее число косточек вниз. Вследствие того, что число в дискретных машинах изображается в цифровой форме, их называют также *цифровыми машинами*.

К цифровым машинам относятся и арифмометры, ручные и автоматические. В них числа изображаются также в десятичной системе счисления. Каждая цифра изображается углом поворота шестерни, которая может находиться в одном из десяти различных устойчивых состояний.

Часто используются также *счетно-аналитические* и *релейные вычислительные машины*. В счетно-аналитических машинах для изображения чисел используются пробивки на перфокартах. Эти машины также работают в десятичной системе счисления.

Релейные вычислительные машины являются промежуточным этапом между ручными и счетно-аналитическими машинами и электронными счетными машинами. В них для изображения чисел служат специальные электромеханические реле. Релейные машины работают уже в двоичной системе счисления и являются машинами с *программным управлением*.

Скорости работы и возможности цифровых вычислительных машин весьма различны. С помощью автоматической клавишной вычислительной машины опытный вычислитель может выполнить до двух с половиной тысяч действий за семичасовую смену. Скорость работы счетно-аналитических и релейных машин составляет уже несколько операций в секунду. Советская релейная вычислительная машина РВМ-1 конструкции инженера Н. И. Бессонова, являющаяся наиболее совершенной релейной машиной, обладает скоростью 20 арифметических операций в секунду.

Электронные счетные машины в настоящее время обладают скоростями порядка десятков и сотен тысяч арифметических операций в секунду. Такое быстродействие связано с тем, что электронные устройства, применяемые в современных электронных счетных машинах, не обладают инерцией механических или релейных элементов и время их срабатывания исчисляется миллионными долями секунды (микросекундами).

§ 53. Арифметические действия. Машины с плавающей и фиксированной запятой

Как мы уже говорили в § 1, числа могут быть записаны в естественной или в нормальной формах или, иначе, с фиксированной или с плавающей запятой. Для сложения и вычитания более удобна фиксированная форма, чем плавающая. Выше мы всюду предполагали, что число записывается в ячейке памяти в плавающей форме. Машины с таким представлением числа называют машинами *с плавающей запятой.*

Сложение чисел с плавающей запятой состоит из нескольких элементов. Прежде всего нужно сравнить порядки слагаемых. Если эти порядки различны, то одно из слагаемых надо «разнормализовать», сдвинув мантиссу таким образом, чтобы уравнять порядки. После этого можно производить сложение, а затем, быть может, понадобится еще раз сдвигать мантиссу для нормализации результата. Примеры таких действий приводились нами в § 1.

В ряде случаев бывает выгодно или удобно производить действия с ненормализованными числами. В машине предусматриваются тогда специальные модификации арифметических операций с блокировкой (выключением) нормализации или округления результатов действия.

Используются и машины *с фиксированной запятой,* в которых положение запятой фиксировано, обычно перед первым разрядом числа, так что в ячейке машины с фиксированной запятой предполагаются записанными только числа, меньшие единицы. Такие машины конструктивно проще, так как в них все разряды, кроме знакового, равноправны и работают совершенно одинаково. Арифметические операции в машине с фиксированной запятой выполняются проще. Например, сложение чисел можно производить сразу так, как они написаны, ничего не сдвигая и не проверяя.

Недостатком машин с фиксированной запятой является ограниченность диапазона записываемых в них чисел. Для работы на машине с фиксированной запятой необходимо, чтобы все исходные данные, результаты промежуточных вычислений, а также окончательные результаты изображались числами, меньшими единицы. Это достигается выбором единиц измерения и подбором соответствующих масштабных коэффициентов, но существенно затрудняет программирование. Поэтому для сколько-нибудь сложных задач применение машин с фиксированной запятой невыгодно.

§ 54. Одноадресные и двухадресные машины

Для целей программирования наиболее удобными являются трехадресные машины, команда в которых содержит три адреса. Это позволяет записывать команду в содержательных обозначениях наиболее удобным и естественным способом. Однако дальнейшее развитие трех-

адресных машин, по-видимому, будет сдерживаться следующим обстоятельством.

Рассматриваемая нами машина имеет $4096 = 2^{12}$ ячеек памяти, поэтому для записи трех адресов нужно 36 разрядов. Кроме них, на запись кода отводится шесть разрядов и три — на запись признаков для работы с индексным регистром. Мы получаем 45 разрядов, что вполне достаточно для записи числа. Если же увеличить оперативную память, то ячейка для записи трехадресной команды может оказаться слишком длинной. Например, если оперативная память будет состоять из $32768 = 2^{15}$ ячеек, то уже для записи адресной части команды понадобятся 45 разрядов. Если еще добавить, что в машине удобно иметь не один, а несколько индексных регистров, то ясно, что потребуется весьма длинная ячейка памяти, которую слишком громоздко осуществлять.

Выход из этого положения — в изменении структуры команды и уменьшении числа упоминаемых в ней адресов. Такие машины с меньшим числом адресов в команде (одноадресные и двухадресные) применяются в настоящее время. Программирование для одноадресных и двухадресных машин имеет свои особенности, и мы не будем на них останавливаться. Сделаем только несколько общих замечаний.

Наиболее распространенными у нас являются одноадресные машины типа *Урал* конструкции инженера Б. И. Рамеева, в особенности машина *Урал*-2. Машина *Урал*-2 имеет 4096 двадцатиразрядных ячеек памяти и может работать в режимах с плавающей и с фиксированной запятой. Плавающее число записывается в двух соседних ячейках памяти, т. е. является сорокаразрядным.

Команда машины *Урал* состоит из кода операции и одного адреса. Вследствие этого программа для решения задачи требует большего числа команд. Например, одна трехадресная команда

$$a + b = c$$

потребует для своей записи трех одноадресных, которые можно записать, например, таким образом (Σ означает сумматор машины):

$$1)\ \Sigma \Leftarrow a$$
$$2)\ \Sigma + b$$
$$3)\ \Sigma \Rightarrow c$$

Эти команды означают:

1) послать число из ячейки a в сумматор;
2) содержимое сумматора сложить с числом из ячейки b;
3) содержимое сумматора переслать в ячейку c памяти.

Не следует думать, что такое удлинение имеет место всегда. Во-первых, ряд команд является, по существу, одноадресными (например, команды передачи управления). Во-вторых, ряд промежуточных результатов не требуется отправлять на хранение в ячейку памяти. Они

могут быть использованы тут же, и за счет этого программу можно значительно сократить.

Рассмотрим, например, как напишется программа вычисления кубического многочлена по схеме Горнера:

$$y = a_0x^3 + a_1x^2 + a_2x + a_3 = ((a_0x + a_1)x + a_2)x + a_3.$$

Для трехадресной машины эта программа требует семи команд:

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

Для одноадресной машины здесь будет достаточно девяти команд:

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

Увеличение, как мы видим, совсем не такое большое.

Более близкими по методам программирования к трехадресным являются двухадресные машины *Минск*. Команда в такой машине содержит, кроме кода операции, два адреса. В зависимости от кода, результат действия либо остается в сумматоре, как у одноадресной машины, либо записывается в ячейку по второму адресу, как в трехадресной.

ЧАСТЬ ТРЕТЬЯ

МЕТОДЫ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

ГЛАВА XII

ЧИСЛЕННОЕ РЕШЕНИЕ АЛГЕБРАИЧЕСКИХ И ТРАНСЦЕНДЕНТНЫХ УРАВНЕНИЙ

§ 55. Подбор корней

Как известно, далеко не всякое уравнение может быть решено точно. В первую очередь это относится к большинству трансцендентных уравнений, т. е. уравнений, в которых неизвестная x находится под знаком трансцендентной функции. Доказано также, что нельзя построить формулу, по которой можно было бы решать произвольное алгебраическое уравнение степени выше четвертой *).

Однако точное решение уравнения не всегда является необходимым. Задачу отыскания корней уравнения можно считать практически решенной, если мы сумеем определить корни с нужной степенью точности и указать пределы возможной погрешности. Говоря о приближенном решении уравнений, мы всюду будем иметь в виду отыскание лишь действительных корней.

Большинство употребляющихся приближенных способов решения уравнений являются, по существу, способами *уточнения корней*, т. е. для их применения необходимо знание примерных значений корня. Для этой последней цели могут служить графические способы, о которых и будет сейчас идти речь.

Пусть рассматриваемое уравнение имеет вид

$$f(x)=0. \tag{1.55}$$

Построим в декартовой системе координат схематический график функции $y=f(x)$. Абсциссы точек пересечения построенной кривой с осью Ox дадут нам значения действительных корней уравнения (1.55).

После того как схематический график построен и примерно выделены участки оси абсцисс, в которых будут лежать корни функции,

*) Доказательство этого факта связано с именами замечательных математиков Абеля (1802—1829) и Галуа (1811—1832).

мы приступим к уточнению значений корней. Для этого можно построить на выделенных участках график функции в более крупном масштабе, производя при этом более точное вычисление значений функции.

Разумеется, при этом мы значительно точнее найдем точки пересечения графика с осью абсцисс. Для построения такого графика полезно воспользоваться миллиметровой бумагой.

Графическое отыскание корня можно производить иначе. Допустим, что уравнение (1.55) можно представить в виде

$$f_1(x)=f_2(x). \tag{2.55}$$

В этом случае строим графики функций $y=f_1(x)$ и $y=f_2(x)$; абсциссы точек пересечения кривых будут действительными корнями уравнения.

Пример 1.55. Найдем приближенно корни уравнения

$$x\cdot 2^x=1.$$

Построение графика функции $y=x\cdot 2^x$ может представлять затруднения. Поэтому для графического подбора корней выгоднее записать уравнение в виде (2.55). Это можно сделать двумя различными способами:

$$2^x=\frac{1}{x}$$

или

$$x=2^{-x}.$$

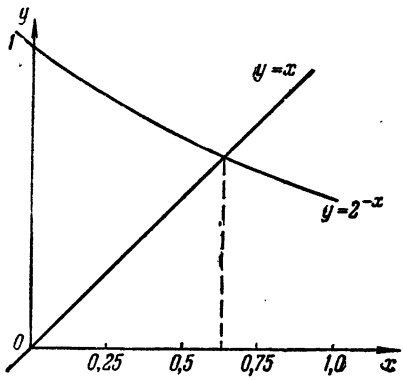


Рис. 46.

Ясно, что второй из них предпочтительнее, потому что в первом случае нужно строить две кривые, а во втором — кривую и прямую.

Построив графики кривой $y=2^{-x}$ и прямой $y=x$ (рис. 46), найдем, что точка пересечения этих линий имеет абсциссу $x\approx 0{,}64$, что можно считать грубым приближением корня.

Перейдем теперь к аналитическим способам уточнения значений корней. Сразу подчеркнем: все эти способы предполагают, что нам известен некоторый интервал $[a, b]$, в котором лежит уточняемый корень уравнения. Выбор этого интервала производится на основании известного свойства непрерывных функций: если функция $f(x)$ непрерывна на замкнутом интервале $[a, b]$ и на его концах имеет различные знаки, $f(a)\cdot f(b)<0$, то между точками a и b имеется хотя бы один корень уравнения $f(x)=0$.

Мы будем считать интервал $[a, b]$ настолько малым, что на нем лежит в точности один корень нашего уравнения. Тогда говорят, что интервал $[a, b]$ является *интервалом изоляции корня* *).

Сужéние интервала изоляции можно производить самым простым образом. Выбираем какую-либо точку c, лежащую внутри интервала (обычно за точку c принимают середину рассматриваемого интервала или близкую к ней точку, в которой удобнее вычислять значение функции), и вычисляем значение $f(c)$.

В качестве нового интервала изоляции мы примем ту из двух половинок интервала $[a, b]$, на концах которой функция имеет разные знаки.

Таким путем можно как угодно сузить участок, на котором находится корень, т. е. получить приближенное значение корня с любой степенью точности. Вместе с тем мы получаем здесь и оценку точности приближенного решения (корень заключен между концами очередного участка). Однако, несмотря на принципиальную простоту, такое последовательное сужéние участка на практике не всегда проводится, ибо часто требует слишком большого количества вычислений, особенно если решение производится вручную или с помощью клавишных вычислительных машин, а не на электронной счетной машине.

При применении других способов уточнения корня будем требовать, чтобы на рассматриваемом отрезке $[a, b]$ функция $f(x)$ удовлетворяла следующим условиям:

1. Функция $f(x)$ непрерывна на отрезке $[a, b]$ вместе со своими производными первого и второго порядков.

2. Значения $f(x)$ на концах участка $[a, b]$ имеют разные знаки:

$$f(a) \cdot f(b) < 0.$$

3. Первая и вторая производные $f'(x)$ и $f''(x)$ сохраняют определенный знак на всем участке.

Эти условия гарантируют, что корень уравнения (1.55) содержится в интервале $[a, b]$ и других корней на этом участке не имеется. В самом деле, условия 1 и 2 гарантируют, что между точками a и b находится хотя бы один корень уравнения; из условия 3 следует, что функция $f(x)$ на данном интервале монотонна и поэтому корень будет единственным **).

В дальнейшем во всей этой главе мы будем предполагать, что функция $f(x)$ и интервал $[a, b]$ удовлетворяют перечисленным выше условиям 1 —3.

Заметим, кстати, что мы можем ограничиваться только тем случаем, когда корни уравнения $f(x) = 0$ положительны. Случай отрицательных корней может быть сведен к рассмотрению положительных, для чего в уравнении достаточно заменить x на $-x$.

---

*) Условия, при соблюдении которых интервал $[a, b]$ будет интервалом изоляции, сформулированы ниже.

**) Значение постоянства знака $f''(x)$ выяснится несколько позже.

§ 56. Способ хорд и проведение параболы

Идея способа хорд состоит в том, что можно, с известным приближением, допустить, что функция на достаточно малом участке $[a, b]$ изменяется линейно. Тогда кривую $y=f(x)$ на участке $[a, b]$ можно заменить хордой и в качестве приближенного значения корня принять точку пересечения хорды с осью абсцисс.

Для лучшего выяснения сути способа обратимся к чертежу (рис. 47). Построим график функции $y=f(x)$ на участке $[a, b]$. Истинный корень уравнения $f(x)=0$ есть абсцисса точки A, являющейся точкой пересечения кривой MM' с осью абсцисс. Заменив кривую MM' хордой MM', мы примем в качестве приближенного значения корня абсциссу точки B, в которой хорда пересекается с осью.

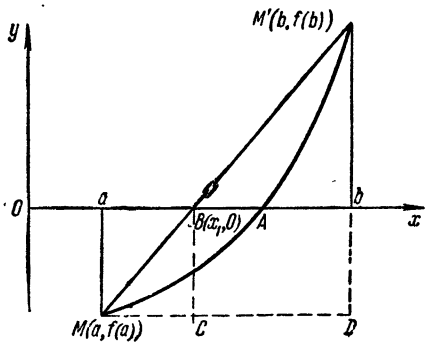


Рис. 47.

Найдем аналитическое выражение для приближенного значения корня. Как видно из рис. 46, угловой коэффициент хорды MM' равен

$$k=\frac{f(b)-f(a)}{b-a}.$$

Легко заметить, что это выражение не изменится и в том случае, если $f(a)>0$, а $f(b)<0$. Таким образом, уравнение хорды MM' можно записать в виде

$$y=\frac{f(b)-f(a)}{b-a}x+m. \tag{1.56}$$

Коэффициент m можно определить, например, из условия, что при $x=a$ хорда должна проходить через точку M кривой, т. е. мы должны иметь $y=f(a)$. Тогда

$$f(a)=\frac{f(b)-f(a)}{b-a}a+m,$$

откуда

$$m=f(a)-\frac{f(b)-f(a)}{b-a}a,$$

и уравнение хорды получает вид

$$y=\frac{f(b)-f(a)}{b-a}(x-a)+f(a).$$

Чтобы получить абсциссу точки B пересечения хорды с осью Ox, положим $y=0$ и решим полученное уравнение относительно x.

Будем иметь

$$\frac{f(b)-f(a)}{b-a}(x-a)=-f(a),$$

откуда

$$x_1=a-\frac{(b-a)f(a)}{f(b)-f(a)}. \tag{2.56}$$

Это и есть формула для приближенного значения корня, полученного по способу хорд.

В ряде случаев бывает удобнее отправляться от точки b вместо точки a, как мы сделали это выше. Тогда коэффициент m в уравнении (1.56) нужно искать, исходя из того, что при $x=b$ должно быть $y=f(b)$, т. е.

$$f(b)=\frac{f(b)-f(a)}{b-a}b+m,$$

откуда

$$m=f(b)-\frac{f(b)-f(a)}{b-a}b,$$

и уравнение хорды получает вид

$$y=\frac{f(b)-f(a)}{b-a}(x-b)+f(b).$$

Отсюда для абсциссы точки B, т. е. для приближенного значения корня уравнения, находим, как и выше,

$$x_1=b-\frac{(b-a)f(b)}{f(b)-f(a)}. \tag{3.56}$$

Очевидно, что формулы (2.56) и (3.56) тождественны. В этом легко убедиться и непосредственными преобразованиями. Мы будем пользоваться той из них, которая окажется более удобной.

Полученное значение x_1 можно снова использовать для дальнейшего уточнения корня по способу хорд, рассматривая интервал $[a, x_1]$ или же $[x_1, b]$, смотря по тому, в каком из них лежит истинный корень. Чтобы определить это, находят знак $f(x_1)$.

Пример 1.56. Найдем по способу хорд положительный корень уравнения $x^3-2x^2+x-3=0$.

Прежде всего определяем знаки функции в различных точках. Результаты вычислений приведены в таблице (табл. 1.56), причем значения аргумента помещены в том порядке, в каком вычисления проводились.

Таблица 1.56

x	0	1	2	3	2,5	2,2	2,1
$f(x)$	−	−	−	+	+	+	−

Из таблицы видно, что функция меняет знак на отрезке [2, 3]. Однако этот участок слишком велик; дальнейшее сужение дает отрезок [2,1; 2,2], к которому мы и применим способ хорд. Вычисление значений функции дает

$$f(2{,}1) = -0{,}459,$$
$$f(2{,}2) = +0{,}168.$$

По формуле (2.56)

$$x_1 = 2{,}1 - \frac{(2{,}2 - 2{,}1)(-0{,}459)}{0{,}168 + 0{,}459} = 2{,}173.$$

Вычислив значение функции при $x_1 = 2{,}173$, находим что $f(2{,}173) = -0{,}01011$. Отсюда видно, что истинный корень расположен в интервале [2,173; 2,2]. Снова применив к этому участку способ хорд, получим

$$x_1^{(2)} = 2{,}173 - \frac{(2{,}2 - 2{,}173)(-0{,}01011)}{0{,}168 + 0{,}01011} = 2{,}1745.$$

Вычисления значений функции дают

$$f(2{,}1745) = -0{,}38544 \cdot 10^{-3}, \quad f(2{,}1746) = +0{,}26335 \cdot 10^{-3}.$$

Полагая значение корня равным $x = 2{,}17455$, видим, что погрешность полученного приближения меньше 0,00005.

Более точно корень уравнения $f(x) = 0$ можно получить, если вычислить значения функции $f(x)$ в трех точках и провести через них параболу, которая лучше приближает функцию, чем хорда. За приближенное значение корня можно взять тогда точку пересечения этой параболы с осью Ox. Рассмотрим соответствующий пример.

Пример 2.56. Найдем корень уравнения, рассмотренного в примере 1.55:

$$x \cdot 2^x = 1.$$

Так как здесь не требуется строить графика функции, то можно записать уравнение в виде

$$x \cdot 2^x - 1 = 0,$$

т. е. положить

$$f(x) = x \cdot 2^x - 1.$$

Как уже было замечено, корень этого уравнения лежит на участке (0,1). Выбрав еще точку $x = \frac{1}{2}$ и вычислив значения функции для этих трех значений аргумента, найдем точки

$$x = 0, \quad y = -1,$$
$$x = \frac{1}{2} \quad y = \frac{\sqrt{2}}{2} - 1 \approx -0{,}293,$$
$$x = 1 \quad y = 1,$$

через которые и надо проводить параболу. Запишем уравнение параболы в виде

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

Для того чтобы эта парабола проходила через полученные нами точки, коэффициенты a, b, c должны удовлетворять уравнениям

$$\begin{aligned} a \cdot 0^2 + b \cdot 0 + c &= -1, \\ a \cdot \frac{1}{4} + b \cdot \frac{1}{2} + c &= -0{,}293, \\ a + b + c &= 1, \end{aligned}$$

откуда сразу находим $c = -1$; для остальных двух коэффициентов получаем систему

$$\begin{aligned} a + b &= 2, \\ a + 2b &= 2{,}828, \end{aligned}$$

так что $a = 1{,}172$ и $b = 0{,}828$. Таким образом, уравнение искомой параболы имеет вид

$$y = 1{,}172x^2 + 0{,}828\,x - 1.$$

Чтобы получить приближенно корень интересующего нас уравнения, надо найти точку пересечения этой параболы с осью Ox, т. е. решить квадратное уравнение

$$1{,}172x^2 + 0{,}828x - 1 = 0,$$

что дает корень на участке (0, 1)

$$x = 0{,}636.$$

Это и есть приближенное значение корня нашего уравнения с значительно большей точностью, чем оно было получено графически.

Внимательный читатель сможет заметить связь, существующую между способом хорд и линейной интерполяцией, которая рассматривалась в § 2. Аналогичная связь существует также между способом проведения параболы и квадратичной интерполяцией, которая будет рассматриваться нами ниже, в четырнадцатой главе. Без этого вывод общей формулы для приближенного значения корня способом проведения параболы оказывается затруднительным. Поэтому здесь придется ограничиться рассмотренным примером. Мы еще раз вернемся к этому способу в § 77, где будет рассмотрено программирование приближенного нахождения корня с помощью проведения параболы.

§ 57. Способ касательных. Комбинированный способ

Наряду со способами хорд и проведения параболы часто используется способ, также основанный на замене графика функции участком прямой линии.

Обратимся снова к уравнению $f(x) = 0$.

Возьмем некоторую точку c участка $[a, b]$ и проведем в точке $[c, f(c)]$ графика функции касательную к этому графику. Уравнение касательной имеет вид

$$y - f(c) = f'(c) \cdot (x - c).$$

В качестве приближенного корня уравнения $f(x) = 0$ примем абсциссу точки пересечения касательной с осью Ox. Полагая в уравнении касательной $y = 0$, находим для абсциссы точки пересечения

$$x_2 = c - \frac{f(c)}{f'(c)} \qquad (1.57)$$

(см. рис. 48, где принято $c = b$).

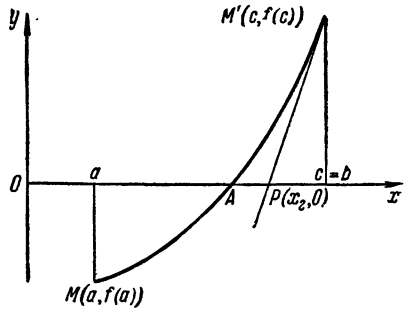


Рис. 48.

Остается решить вопрос о выборе точки c. На рис. 48 мы приняли $c = b$. Нетрудно видеть, что в этом случае $f(c) > 0$ и $f''(c) > 0$, ибо кривая вогнута. Обычно принимают $c = a$ или $c = b$, смотря по тому, в какой из этих точек знак функции совпадает со знаком второй производной, т. е. c выбирают так, чтобы произведение $f(c) \cdot f''(c)$ было положительно. Как будет показано ниже, в этом случае можно гарантировать, что приближенное значение корня, полученное по способу касательных, лежит на участке $[a, b]$, т. е. что $a < x_2 < b$.

Как и в способе хорд, значение x_2 можно использовать для дальнейшего уточнения корня, беря участок $[a, x_2]$ или $[x_2, b]$.

Пример 1.57. Рассмотрим то же уравнение, что и в примере 1.56:

$$x^3 - 2x^2 + x - 3 = 0.$$

Здесь $f'(x) = 3x^2 - 4x + 1 > 0$ и $f''(x) = 6x - 4 > 0$, ибо мы выбрали интервал [2,1; 2,2]. Если принять $c = a$, то $f(c) \cdot f''(c) < 0$, ибо $f(2,1) < 0$. Наоборот, при $c = b = 2,2$ имеем $f(c) \cdot f''(c) > 0$, так что касательную следует проводить в точке $c = b$. Формула (1.57) дает

$$x_2 = 2,2 - \frac{0,168}{6,72} = 2,175.$$

Так как $f(2,175) > 0$, то на участке [2,1; 2,175] можно вновь применить способ касательных, полагая $c = 2,175$. Воспользовавшись снова формулой (1.57), получим

$$x_2^{(2)} = 2,175 - \frac{0,28594 \cdot 10^{-2}}{6,491875} = 2,17456.$$

Рассмотренный пример показывает, что способ хорд и способ касательных дают приближение корня с разных сторон (больше и меньше истинного корня). Поэтому обычно бывает выгодно применять оба эти способа одновременно, благодаря чему уточнение корня может быть получено быстрее.

Ограничения, наложенные в § 55 на функцию и отрезок $[a, b]$, показывают, что возможны четыре различных случая поведения функции на этом отрезке, в зависимости от знаков производных. На рис. 49—52 изображены все возможные случаи поведения функции. По рисункам видно, что во всех четырех случаях точка A, изображающая истинный корень, лежит между точкой B, т. е. приближенным значением корня по способу хорд, и точкой P, изображающей приближенное значение корня, полученное по способу касательных.

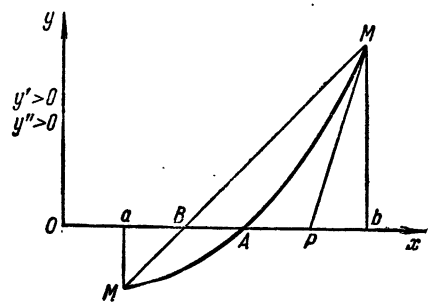


Рис. 49.

Обозначим абсциссу точки A через $\bar{x}$, точки B — через x_1 и точки P — через x_2. Из рис. 49 видно, что для случая, когда $y' > 0$ и $y'' > 0$, имеют место неравенства

$$a < x_1 < \bar{x} < x_2 < b.$$

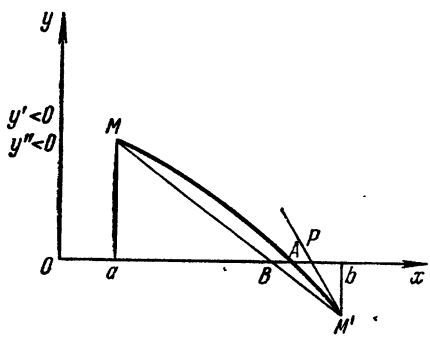


Рис. 50.

Это же неравенство имеет место для случая $y' < 0$, $y'' < 0$ (рис. 50). Если $y' > 0$, но $y'' < 0$ (рис. 51) или, наоборот, $y' < 0, y'' > 0$ (рис. 52),

$$a < x_2 < \bar{x} < x_1 < b.$$

При этом во всех случаях касательная проводится в соответствии со сделанными указаниями.

Таким образом, во всех четырех случаях истинный корень уравнения заключен между приближенными корнями, получающимися по способу хорд и по способу касательных. Расположение корней можно охарактеризовать таблицей 1.57.

Сделаем теперь несколько замечаний, касающихся практического применения комбинации способов хорд и касательных. Прежде всего необходимо, как было указано выше, выделить интервал $[a, b]$, т. е. изолировать корень. Далее, определив знаки производных, мы можем судить о расположении корней по приведенной здесь таблице.

Пусть, например, $y'y'' > 0$. Тогда способ хорд дает значение корня с недостатком. Для нахождения этого корня мы воспользуемся

формулой (2.56), записав ее в виде

$$a_1 = a + \Delta a, \quad \text{где} \quad \Delta a = -\frac{(b-a)f(a)}{f(b)-f(a)}. \tag{2.57}$$

Для способа касательных следует выбрать $c = b$, так что формулу (1.57) можно записать в виде

$$b_1 = b + \Delta b, \quad \Delta b = -\frac{f(b)}{f'(b)}. \tag{3.57}$$

Так как способ касательных дает значение с избытком, то истинный корень находится на интервале $[a_1, b_1]$. Поэтому можно

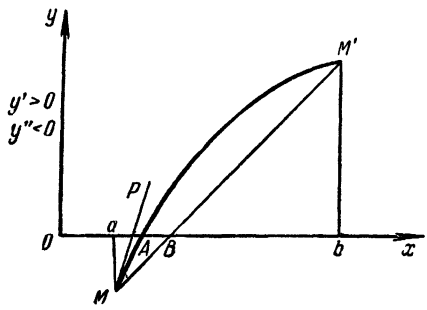


Рис. 51.

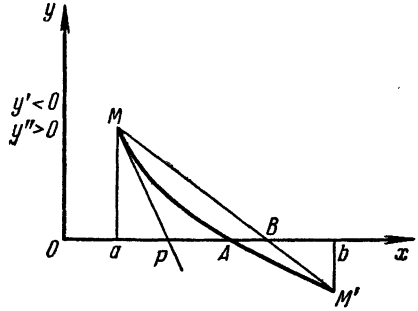


Рис. 52.

продолжать дальнейшее уточнение корня, получая новый участок по формулам

$$a_2 = a_1 + \Delta a_1, \quad \Delta a_1 = -\frac{(b_1-a_1)f(a_1)}{f(b_1)-f(a_1)}, \tag{4.57}$$

$$b_2 = b_1 + \Delta b_1, \quad \Delta b_1 = -\frac{f(b_1)}{f'(b_1)}. \tag{5.57}$$

Таблица 1.57

	Способ хорд	Способ касательных
$y'y'' > 0$	с недостатком	с избытком
$y'y'' < 0$	с избытком	с недостатком

Подобное уточнение следует продолжать до тех пор, пока не получится интервал, длина которого не превосходит допустимой погрешности.

Отметим, что обычно среднее арифметическое приближенных корней, полученных по способу хорд и способу касательных, дает лучшее приближение, нежели каждый из этих корней в отдельности.

Если $y'y''<0$, то ход решения остается тем же, но формулы меняются местами. Способ хорд дает в этом случае корень с избытком, а способ касательных — с недостатком.

Для способа касательных следует выбирать $c=a$, поэтому формулу (1.57) мы запишем в виде

$$a_1=a+\Delta a, \quad \Delta a=-\frac{f(a)}{f'(a)}. \tag{6.57}$$

В способе хорд мы воспользуемся формулой (3.56), записав ее в виде

$$b_1=b+\Delta b, \quad \Delta b=-\frac{f(b)(b-a)}{f(b)-f(a)}. \tag{7.57}$$

Истинный корень находится в интервале $[a_1, b_1]$. Дальнейшее уточнение производится по формулам

$$a_2=a_1+\Delta a_1, \quad \Delta a_1=-\frac{f(a_1)}{f'(a_1)}, \tag{8.57}$$

$$b_2=b_1+\Delta b_1, \quad \Delta b_1=-\frac{f(b_1)(b_1-a_1)}{f(b_1)-f(a_1)} \tag{9.57}$$

и т. д.

Пример 2.57. Решим способом хорд и касательных уравнение

$$(4+x^2)(e^x-e^{-x})=18.$$

Представив это уравнение в виде

$$\frac{e^x-e^{-x}}{2}=\frac{9}{4+x^2},$$

построим графики функций $y=\frac{9}{4+x^2}$ и $y=\frac{e^x-e^{-x}}{2}$ $(=\operatorname{sh}x)$. Как видно на рис. 53, приближенное значение корня можно считать равным $x=1{,}25$. Для уточнения способом хорд и касательных можно записать уравнение в виде $f(x)=(4+x^2)(e^x-e^{-x})-18=0$.

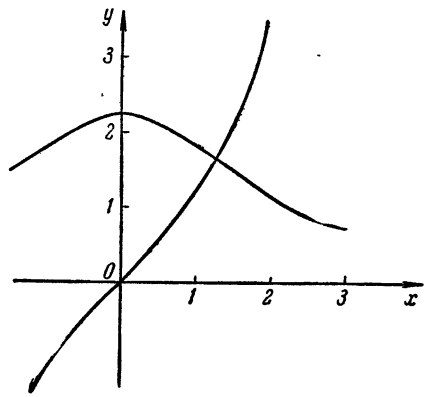


Рис. 53.

В качестве начального интервала изоляции корня, как видно из того же рисунка, можно взять интервал (1,2; 1,3). Производные на этом участке равны

$$f'(x)=2x(e^x-e^{-x})+(4+x^2)(e^x+e^{-x})>0,$$

$$f''(x)=2(e^x-e^{-x})+4x(e^x+e^{-x})+(4+x^2)(e^x-e^{-x})>0.$$

Из таблицы 1.57 следует, что способ хорд дает в этом случае приближение с недостатком, а способ касательных — с избытком. Дальнейшие вычисления приведены в табл. 2.57.

Таблица 2.57

(1)	(2)	(3)	(4)	(5)	(6)
x	(1)^2 + «4»	$e^{(1)}$	$e^{-(1)}$	(3) — (4)	(2) · (5) — «18»
1,2	5,44	3,320107	0,301194	3,018913	— 1,577114
1,3	5,69	3,669297	0,272529	3,396768	1,327610
1,254	5,572516	3,504325	0,285361	3,218964	— 0,062275
1,257	5,580049	3,514872	0,284507	3,230365	0,025595
1,256126					
1,256127					

(7)	(8)	(9)	(10)	(11)	(12)
(1) · (5)	(3) + (4)	(2) · (8)	$f'(x)$ «2» · (7) + (9)	Δa $-\frac{(b-a)f(a)}{f(b)-f(a)}$	Δb — (6) : (11)
				0,054	
4,415798	3,941826	22,428990	31,260587		— 0,043
				0,002126	
4,060569	3,799379	21,200721	29,321859		—0,000873

Остается только показать, что, пользуясь этими двумя способами в отдельности (или их комбинацией) несколько раз, можно получить корень с любой степенью точности, т. е. доказать сходимость рассматриваемых процессов.

Пусть $x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(n)}, \ldots$ — последовательность приближенных корней уравнения, полученных по способу хорд, а $\bar{x}$ — точный корень того же уравнения. Как было показано выше, в зависимости от знака производных, имеют место неравенства

$$a < x_1^{(1)} < x_1^{(2)} < \ldots < x_1^{(n)} < \ldots < \bar{x}$$

или

$$b > x_1^{(1)} > x_1^{(2)} > \ldots > x_1^{(n)} > \ldots > \bar{x}.$$

При этом два соседних значения связаны соотношением

$$x_1^{(n+1)} = x_1^{(n)} - \frac{(b - x_1^{(n)}) f(x_1^{(n)})}{f(b) - f(x_1^{(n)})}. \qquad (10.57)$$

Так как последовательность $\{x_1^{(n)}\}$ монотонна и ограничена, то она имеет предел. Обозначив его через α,

$$\lim_{n \to \infty} x_1^{(n)} = \alpha,$$

перейдем к пределу в равенстве (10.57), что дает

$$\alpha = \alpha - \frac{(b - \alpha) f(\alpha)}{f(b) - f(\alpha)},$$

откуда вытекает $f(\alpha) = 0$, т. е. $\alpha = \bar{x}$ *), ибо других корней в интервале $[a, b]$, по предположению, нет.

Аналогично доказывается сходимость для способа касательных. Обозначим через $\{x_2^{(n)}\}$ последовательность корней, получающихся по способу касательных. Как и выше, эта последовательность монотонна и ограничена, а потому имеет предел. Кроме того,

$$x_2^{(n+1)} = x_2^{(n)} - \frac{f(x_2^{(n)})}{f'(x_2^{(n)})}. \qquad (11.57)$$

Пусть $\lim\limits_{n \to \infty} x_2^{(n)} = \beta$. Переходя к пределу в равенстве (11.57), получим

$$\beta = \beta - \frac{f(\beta)}{f'(\beta)},$$

что дает $f(\beta) = 0$, т. е. $\beta = \bar{x}$.

§ 58. Способ итераций

В ряде случаев весьма удобным приемом решения уравнений является *метод итераций* (повторений). Для применения этого метода исходное уравнение нужно записывать в форме

$$x = \varphi(x). \qquad (1.58)$$

Пусть каким-либо способом выделен интервал $[a, b]$ изоляции корня этого уравнения, и x_0 — любая точка этого интервала *(нулевое приближение)*. Для получения следующего приближения x_1 в правую часть уравнения (1.58) вместо x подставляем x_0, так что

$$x_1 = \varphi(x_0).$$

Следующие приближения получаются по схеме

$$x_2 = \varphi(x_1),$$
$$x_3 = \varphi(x_2),$$
$$\ldots\ldots$$
$$x_n = \varphi(x_{n-1}),$$
$$\ldots\ldots$$

*) Здесь $\alpha \neq b$ и $f(\alpha) \neq f(b)$.

Если последовательность $x_1, x_2, \ldots, x_n, \ldots$ имеет предел $\lim\limits_{n\to\infty} x_n = \bar{x}$, то $\bar{x}$ является корнем уравнения (1.58). В самом деле, предполагая, что $\varphi(x)$ — непрерывная функция, получим

$$\bar{x} = \lim_{n\to\infty} x_n = \lim_{n\to\infty} \varphi(x_{n-1}) = \varphi\left(\lim_{n\to\infty} x_n\right) = \varphi(\bar{x}),$$

т. е.

$$\bar{x} = \varphi(\bar{x}).$$

Следовательно, $\bar{x}$ является корнем нашего уравнения. Поэтому одно из значений x_n с достаточно большим номером можно принять за приближенное значение корня. Однако может случиться, что последовательность $x_1, x_2, \ldots, x_n, \ldots$ не имеет предела, и тогда метод итераций не приводит к цели.

Представляет большой интерес выяснить условия, при которых итерационный процесс сходится. Имеет место следующая теорема.

Теорема. *Пусть интервал $[a, b]$ является интервалом изоляции корня уравнения $x = \varphi(x)$ и во всех точках этого интервала производная $\varphi'(x)$ удовлетворяет неравенству*

$$|\varphi'(x)| \leqslant M < 1. \tag{2.58}$$

Если при этом выполняется условие $a \leqslant \varphi(x) \leqslant b$, то итерационный процесс сходится, причем за нулевое приближение x_0 можно брать любую точку интервала $[a, b]$ *).

Доказательство. Пусть дано уравнение $x = \varphi(x)$ и интервал $[a, b]$ изоляции корня этого уравнения. Будем считать, что функция $\varphi(x)$ дифференцируема и ее производная удовлетворяет условию (2.58).

Пусть x_0 — любая точка интервала $[a, b]$ и $x_1 = \varphi(x_0)$ — первое приближение. Если $\bar{x}$ — точный корень уравнения, то, применяя теорему Лагранжа, получим

$$\bar{x} - x_1 = \varphi(\bar{x}) - \varphi(x_0) = (\bar{x} - x_0)\varphi'(\xi_0),$$

где точка ξ_0 лежит между точками $\bar{x}$ и x_0, т. е. заведомо на интервале $[a, b]$. Согласно неравенству (2.58) будем иметь

$$|\bar{x} - x_1| = |\bar{x} - x_0| \cdot |\varphi'(\xi_0)| \leqslant M|\bar{x} - x_0|.$$

---

*) Из неравенства (2.58) уже следует, что уравнение (1.58) не может иметь в интервале $[a, b]$ двух корней. В самом деле, допустив противное: $x' = \varphi(x')$ и $x'' = \varphi(x'')$, придем к равенству $x' - x'' = \varphi(x') - \varphi(x'') = (x' - x'')\cdot\varphi'(\xi)$, которое невозможно, так как $|\varphi'(\xi)| < 1$. При соблюдении условия $a \leqslant x \leqslant b$ все последующие приближения будут лежать в интервале $[a, b]$. Если $0 \leqslant \varphi'(x) < 1$, то это условие следует само собой. Если же $-1 < \varphi'(x) \leqslant 0$, то нужно только проверить, что первое приближение x_1 не выходит за границы интервала $[a, b]$; тогда остальные приближения будут обязательно лежать в $[a, b]$.

Для второго приближения $x_2 = \varphi(x_1)$ аналогично получим (по условию теоремы точка x_1 принадлежит интервалу $[a, b]$)

$$\bar{x} - x_2 = \varphi(\bar{x}) - \varphi(x_1) = (\bar{x} - x_1)\varphi'(\xi_1),$$

где ξ_1 заключено опять-таки между a и b. Применяя предыдущее неравенство, установим оценку:

$$|\bar{x} - x_2| \leqslant |\bar{x} - x_0| M^2.$$

Повторяя указанный процесс, найдем, что

$$|\bar{x} - x_n| \leqslant |\bar{x} - x_0| \cdot M^n. \tag{3.58}$$

Так как $M < 1$, то $M^n \to 0$ и, следовательно,

$$\lim_{n\to\infty} (\bar{x} - x_n) = 0, \quad \text{т. е.} \quad \lim_{n\to\infty} x_n = x.$$

Таким образом, для сходимости итерационного процесса достаточно, чтобы неравенство $|\varphi'(x)| < 1$ соблюдалось на рассматриваемом интервале. При этом неравенство (3.58) позволяет дать оценку точности приближения. Так как $|\bar{x} - x_0| < b - a$, то

$$|\bar{x} - x_n| < (b - a) M^n. \tag{4.58}$$

Пример 1.58. Решим методом итераций уравнение

$$5x - 8 \ln x = 8.$$

Представив это уравнение в виде

$$\frac{5}{8} x - 1 = \ln x,$$

графически найдем, что оно имеет два корня, приближенно равные $x_0 \approx 0{,}45$ и $x_0 \approx 3{,}7$. (Рекомендуем читателю проделать построение самостоятельно.) Эти значения можно принять за нулевые приближения.

Для более точного нахождения правого корня запишем уравнение в виде

$$x = 1{,}6\,(1 + \ln x).$$

Здесь $\varphi(x) = 1{,}6\,(1 + \ln x)$. Итерационный процесс сходится, так как $\varphi'(x) = \frac{1{,}6}{x}$, что в окрестности правого корня положительно и меньше единицы. Вычисления приведены в табл. 1.58.

При отыскании левого корня итерационный процесс окажется расходящимся, потому что $\varphi'(x) = \frac{1{,}6}{x}$ в окрестности точки $x = 0{,}45$ имеет значение около 3,5. Поэтому первоначальное уравнение следует переписать иначе. Записав его в виде

$$x = e^{0{,}625x - 1},$$

Таблица 1.58

(1)	(2)	(3)
x	«1» + ln (1)	$\overset{\varphi(x)}{\text{«1,6»}}$ · (2)
3,7	2,3	3,69
3,69	2,306	3,689
3,689	2,30536	3,6886
3,6886	2,30525	3,68840
3,68840	2,305193	3,688309
3,688309	2,3051681	3,6882690
3,6882690	2,30515251	3,68824402
3,68824402	2,30515046	3,68824074
3,68824074	2,30514957	3,68823931
3,68823931	2,30514919	3,68823870
3,68823870	2,30514902	3,68823843
3,68823843		

найдем, что $\varphi(x) = e^{0,625x} - 1$ и $\varphi'(x) = 0,625e^{0,625x} - 1$. Здесь производная тоже положительна и меньше единицы (около 0,3). Вычисления, приведенные в табл. 2.58, показывают, что для вычисления корня с точностью до шести знаков требуется семь шагов.

Таблица 2.58

(1)	(2)	(3)
x	«0,625» · (1) — «1»	$\overset{\varphi(x)}{e^{(2)}}$
0,45	—0,719	0,487
0,487	—0,6956	0,4988
0,4988	—0,68825	0,50245
0,50245	—0,68597	0,503601
0,503601	—0,685249	0,5039647
0,5039647	—0,6850221	0,50407914
0,50407914	—0,68495054	0,504115162
0,504115162	—0,684928024	0,5041265149
0,5041265149	—0,6549209282	0,50413008816
0,50413008816		

Для приведения уравнения $f(x) = 0$ к виду $x = \varphi(x)$ таким образом, чтобы получить сходящийся итерационный процесс, часто поступают так: уравнение $f(x) = 0$ равносильно уравнению $\lambda f(x) = 0$; прибавив x к левой и правой частям этого равенства, приходим к уравнению

$$\lambda f(x) + x = x, \tag{5.58}$$

в котором уже можно положить $\lambda f(x)+x=\varphi(x)$, так что уравнение (5.58) приведется к виду (1.58):

$$x=\varphi(x).$$

Параметр λ остался свободным, и его можно подобрать таким образом, чтобы в окрестности корня $\varphi'(x)=\lambda f'(x)+1$ было меньше единицы по абсолютной величине, что будет гарантировать сходимость итерационного процесса.

Иногда прибегают и к другим искусственным приемам для преобразования уравнения к виду, обеспечивающему сходимость итерационного процесса.

Пример 2.58. Выведем формулу для приближений к кубическому корню. Речь идет, следовательно, о решении уравнения $x^3=a$.

Так как a, а значит, и x, могут быть и больше и меньше единицы, то нужно иметь выражение для $\varphi(x)$, которое содержало бы x и в числителе, и в знаменателе. Для этого преобразуем наше уравнение так. Сначала прибавим к обеим его частям $2x^3$. Тогда

$$3x^3=a+2x^3.$$

Разделив обе части равенства на $3x^3$ (очевидно, $x\neq 0$), найдем

$$x=\frac{a}{3x^3}+\frac{2x}{3},$$

т. е. $\varphi(x)=\frac{a}{3x^3}+\frac{2x}{3}$. Производная этой функции равна

$$\varphi'(x)=\frac{2}{3}\frac{x^3-a}{x^3}.$$

Отсюда видно, что если выбрать начальное приближение достаточно близким к истинному значению корня, то выполнение неравенства $|\varphi'(x)|<1$, гарантирующего сходимость итерационного процесса, будет обеспечено.

Если x_{n-1} означает некоторое приближение для кубического корня, то следующее приближение получается по формуле

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

Этой формулой мы уже пользовались во второй части книги (см. § 21).

Интересно рассмотреть геометрический смысл итерационного процесса. Построим графики функций $y=\varphi(x)$ и $y=x$ (рис. 54). Корнем уравнения является абсцисса точки пересечения кривой $y=\varphi(x)$ с биссектрисой координатного угла. Если x_0 — абсцисса нулевого приближения, то $x_1=\varphi(x_0)$ равно ординате соответствующей точки M кривой или же абсциссе точки M_1. Аналогично находятся следующие приближения (рис. 54). Здесь можно также установить роль условия $|\varphi'(x)|<1$.

Так, рис. 54 изображает случай, когда $0<\varphi'(x)<1$, так что кривая пересекает биссектрису слева направо и справа лежит под биссектрисой. Итерационный процесс в этом случае сходится, причем

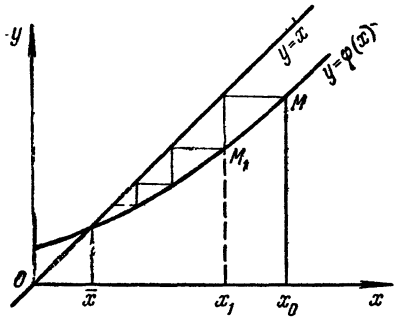


Рис. 54.

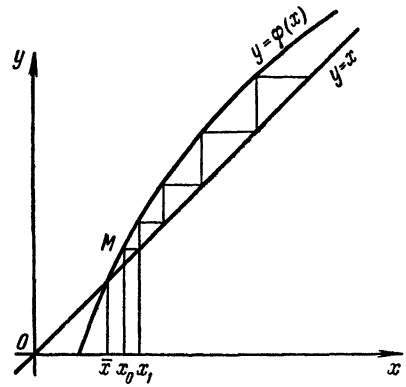


Рис. 55.

приближения монотонно убывают, если $x_0>\bar{x}$, или монотонно возрастают при $x_0<\bar{x}$. На рис. 55 приведен случай, когда $\varphi'(x)>1$. Здесь кривая пересекает биссектрису снизу вверх, процесс оказывается расходящимся. На рис. 56 и 57 соответственно изображены

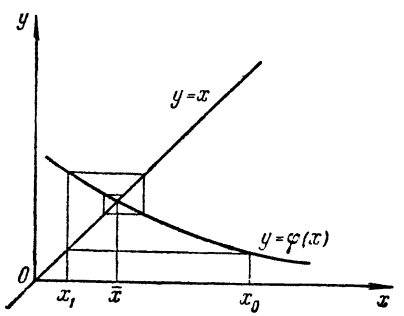


Рис. 56.

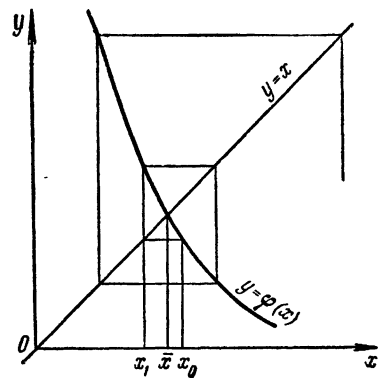


Рис. 57.

случаи, когда производная $\varphi'(x)$ отрицательна. Если при этом $|\varphi'(x)|<1$ (рис. 56), то итерационный процесс сходится, но приближения колеблются около истинного значения корня. При $|\varphi'(x)|>1$ (рис. 57) процесс расходится.

§ 59. Алгебраические уравнения. Схема Горнера

Действительные корни алгебраического уравнения находятся теми же приемами, которые были рассмотрены в предыдущих параграфах. Мы сделаем здесь только несколько дополнительных замечаний, касающихся нахождения всех действительных корней данного алгебраического уравнения.

Воспользуемся известной *теоремой Безу*, состоящей в том, что *остаток от деления многочлена $f(x)$ на двучлен $x-\alpha$ равен значению $f(\alpha)$*. Из нее вытекает, что *если α является корнем многочлена $f(x)$, то этот многочлен делится на $x-\alpha$*.

Сделанное замечание позволяет для отыскания всех действительных корней алгебраического уравнения поступать следующим образом. Вычислив один из корней α с достаточной степенью точности, разделим многочлен $f(x)$ на $x-\alpha$. В частном получится многочлен на единицу меньшей степени, корни которого затем ищутся теми же способами. Так можно поступать до тех пор, пока не дойдем до многочлена, не имеющего действительных корней, либо пока не найдем всех корней первоначального уравнения.

Для быстрого деления многочлена $f(x)$ на двучлен $x-\alpha$ можно воспользоваться так называемой *схемой Горнера*. Обозначив частное от деления $f(x)$ на $x-\alpha$ через $\varphi(x)$, а остаток *) через b_n, можем записать

$$f(x)=(x-\alpha)\varphi(x)+b_n, \tag{1.59}$$

где, как мы уже знаем, $b_n=f(\alpha)$.

Если

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

и

$$\varphi(x)=b_0x^{n-1}+\ldots+b_{n-2}x+b_{n-1},$$

то мы имеем тождество

$$a_0x^n+a_1x^{n-1}+\ldots+a_{n-1}x+a_n=(x-\alpha)(b_0x^{n-1}+\ldots+b_{n-2}x+b_{n-1})+b_n. \tag{2.59}$$

Раскрывая скобки в правой части тождества (2.59) и сравнивая коэффициенты в левой и правой частях тождества, находим

$$\left.\begin{aligned} a_0&=b_0,\\ a_1&=b_1-\alpha b_0,\\ a_2&=b_2-\alpha b_1,\\ &\cdots\cdots\cdots\\ a_{n-1}&=b_{n-1}-\alpha b_{n-2},\\ a_n&=b_n-\alpha b_{n-1}. \end{aligned}\right\} \tag{3.59}$$

*) Мы рассматриваем здесь схему деления многочлена $f(x)$ на двучлен $x-\alpha$, не предполагая, что α есть корень многочлена $f(x)$.

Равенства (3.59) можно переписать в виде

$$\left.\begin{aligned} b_0 &= a_0, \\ b_1 &= a_1 + \alpha b_0, \\ b_2 &= a_2 + \alpha b_1, \\ &\cdots\cdots\cdots \\ b_{n-1} &= a_{n-1} + \alpha b_{n-2}, \\ b_n &= a_n + \alpha b_{n-1}, \end{aligned}\right\} \tag{4.59}$$

что дает возможность вычислять последовательно все коэффициенты частного и остаток b_n.

При ручном счете (если речь идет о нахождении одного значения многочлена) вычисления записывают с помощью схемы

$$\begin{array}{c|cccccc} & a_0 & a_1 & a_2 & \dots & a_{n-1} & a_n \\ \hline \alpha_0 & b_0 & b_1 & b_2 & \dots & b_{n-1} & b_n \end{array}.$$

Числа в нижней строке вычисляются последовательно слева направо, причем $b_0 = a_0$, а каждое следующее число b_k равно сумме коэффициента a_k, стоящего над ним, и произведения предыдущего коэффициента b_{k-1} на α.

Пример 1.58. Вычислим частное и остаток от деления многочлена $x^4 - 3x^3 + 2x^2 + x + 3$ на двучлен $x - 1{,}3$. Пользуясь приведенной схемой, находим

	1	-3	-2	1	3
1,3	1	$-3+1\cdot 1{,}3 = -1{,}7$	$2+(-1{,}7)\,1{,}3 = -0{,}21$	$1+(-0{,}21)\cdot 1{,}3 = 0{,}727$	$3+0{,}727\cdot 1{,}3 = 0{,}9451$

Итак, частное есть многочлен $x^3 - 1{,}7x^2 - 0{,}21x + 0{,}727$, а остаток, равный значению многочлена в точке $x = 1{,}3$, есть 0,9451.

Если в формулах (4.59) подставить каждую предыдущую формулу в последующую, то получится такая цепочка равенств:

$$\begin{aligned} b_1 &= a_1 + \alpha a_0, \\ b_2 &= a_2 + \alpha(a_1 + \alpha a_0), \\ b_3 &= a_3 + \alpha(a_2 + \alpha(a_1 + \alpha a_0)), \\ &\cdots\cdots\cdots\cdots \end{aligned}$$

Дойдя до последней формулы и переписав правую часть в обратном порядке, получим такое равенство:

$$b_n = f(\alpha) = (\dots((a_0\alpha + a_1)\alpha + a_2)\alpha + \dots a_{n-1})\alpha + a_n. \tag{5.59}$$

Справедливость формулы (5.59) легко проверить непосредственно, раскрыв скобки в правой части равенства. Такая форма записи схемы Горнера применяется при вычислении значений многочлена (см. § 83).

Пример 2.59. Найдем все действительные корни уравнения

$$x^3 - 3x^2 - 3x + 4 = 0.$$

Сразу видно, что $f(0) = 4$, $f(1) = -1$ и поэтому один из корней уравнения лежит на участке (0, 1). С помощью схемы Горнера для $x = 0{,}8$ и $x = 0{,}9$ находим

	1	−3	−3	4
0,8	1	−2,2	−4,76	0,192

	1	−3	−3	4
0,9	1	−2,1	−4,89	−0,401

т. е. $f(0{,}8) = 0{,}102$, и $f(0{,}9) = -0{,}401$. Применяя способ хорд и касательных на участке $[0{,}8;\ 0{,}9]$, находим $x = 0{,}8326$ (с точностью до четырех знаков). Разделив $f(x)$ на $x - 0{,}8326$ по той же схеме Горнера, находим

	1	−3	−3	4
0,8326	1	−2,1674	−4,8046	−0,0003

так что остается решить квадратное уравнение

$$x^2 - 2{,}1674x - 4{,}8046 = 0,$$

корни которого

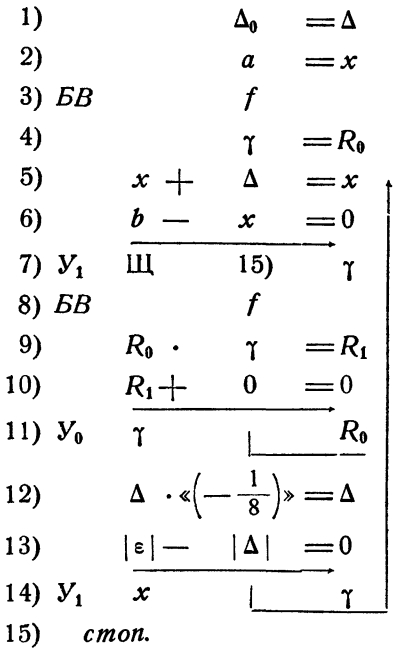
$$x_2 = -1{,}3615, \quad x_3 = 3{,}5289.$$

§ 60. Программирование подбора корней

Пусть ищется корень уравнения $f(x) = 0$ на участке $[a, b]$. Предположим, что вычисление функции $f(x)$ осуществляется с помощью отдельной подпрограммы *Счет* $f(x)$, аргументом которой служит ячейка x, а ответом — ячейка γ.

Подбор корней можно осуществить следующим образом. Выберем некоторый начальный шаг Δ и будем вычислять значения $f(x)$, начиная с точки a с шагом Δ, до тех пор, пока функция не изменит знак или пока мы не дойдем до точки b. Во втором случае уравнение не имеет корней и в ответную ячейку мы будем засылать Щ, как признак отсутствия корня. В первом случае уменьшим шаг Δ (например, в восемь раз) и сменим у него знак, т. е. пойдем от полученной точки в обратную сторону. Такой процесс можно продолжать до тех пор, пока мы не получим корня с достаточной степенью точности, т. е. пока шаг Δ не сделается достаточно малым.

Описанный процесс можно осуществить с помощью следующей программы, в которой ответной ячейкой также служит ячейка γ:



Здесь команда 1) засылает начальное значение шага в рабочую ячейку, команды 2) — 4) вычисляют $f(a)$ и засылают это значение в ячейку R_0. Далее идет увеличение аргумента на Δ (5)), проверка достижения правого конца (6) — 7)) и вычисление функции в новой точке (8)). Команды 9) — 11) проверяют совпадение знаков у двух значений функции. Если произведение этих значений R_1 положительно, то мы возвращаемся к команде 5) и делаем еще один шаг вправо. Если же $R_1 < 0$, то функция изменила знак. Тогда мы переходим к команде 12), которая уменьшит шаг и изменит его знак. Далее, команды 13) — 14) проверят, не достигнута ли нужная точность и либо возвратят на команду 5), либо остановят машину.

Написанная нами программа не является стандартной, так как здесь предполагается известными ячейки a, b, Δ_0, x, ε, f. Для стандартной программы все эти сведения должны быть заданы в информации к программе, а команды, содержащие адреса этих ячеек, должны быть сформированы. В нашем случае необходимо формировать команды 1), 2), 3), 5), 6), 8), 13), 14), т. е. восемь команд.

Как мы видели в § 43, формирование каждой команды занимает довольно много места. Поэтому выгоднее несколько изменить программу так, чтобы уменьшить в ней число формируемых команд. Кроме того, если мы хотим

сделать программу блоком, а не самостоятельной программой, то следует заменить команду *Стоп* неперфорируемой ячейкой *конец* и начать блок командой Ω = конец.

Приведенную программу можно изменить таким образом:

1)			Ω	=	*конец*
2)			Δ_0	=	Δ
3)			*a*	=	*z*
4)	*БВ*	5)	17)		Ω
5)			γ	=	R_0
6)		*z* +	Δ	=	*z*
7)		*b* —	*z*	=	0
8)	$У_1$	*Щ*	16)		γ
9)	*БВ*	10)	17)		Ω
10)		R_0 ·	γ	=	R_1
11)		R_1 +	0	=	0
12)	$У_0$	γ	6)		R_0
13)		Δ	$\cdot\left(-\frac{1}{8}\right)$	=	Δ
14)		|ε| —	|Δ|	=	0
15)	$У_1$	*z*	6)		γ
16)		*конец*			
17)	*Б*	*z*	*f*		*x*

Вместо ячейки *x* «для внутреннего употребления» здесь использована рабочая ячейка *z*. Пересылка значения *z* в аргумент *x* производится одновременно с обращением к функции в команде 17). Обращение к программе счета функции происходит только из одного места, причем команда возврата засылается в ячейку Ω раньше — командой 9) (в цикле) или 4) (в начале работы программы). В этом варианте программы нужно сформировать только пять команд: 2), 3), 7), 14) и 17), которые мы обозначим соответственно *A*, *B*, *C*, *D*, *E*.

Пусть обращение к нашей программе *Корень* имеет вид

Я:	*БВ*	*Я*+3	*Корень*	Ω
Я+1		*a*	*b*	*x*
Я+2		*f*	Δ_0	ε

Напишем формирование команд *A* — *E* для стандартной программы по заданной информации.

Прежде всего необходимо перенести информацию из ячеек *Я*+1 и *Я*+2 в рабочие ячейки. Этими рабочими ячейками могут быть, например, ячейки R_0 и R_1. Для переноса ячейки *Я*+2 в ячейку R_1 нужно сформировать команду

$$Q: \quad 0 \vee Я+2 = R_1.$$

Так как в среднем адресе ячейки Ω после выполнения команды *Я* обращения к корню записан адрес *Я*+3, то команду *Q* можно получить из константы $F \vee F = R_1$, где *F* означает число 7777, путем фиксированного сложения

$$(F \vee F = R_1) +, \quad Ω = Q.$$

Действительно, при этом средний адрес команды сделается равным $Я+2$, а левый — нулю. Для переноса ячейки $Я+1$ в R_0 надо иметь команду P

$$P: \quad 0 \vee Я+1 = R_0,$$

которую легко получить переадресацией команды Q:

$$Q —, \quad (0, 1, 1) = P.$$

Сформировав и выполнив команды P и Q, мы перенесем информацию в ячейки R_0 и R_1, так что их содержимое будет

$$R_0: \quad a \quad b \quad x,$$
$$R_1: \quad f \quad \Delta_0 \quad \varepsilon.$$

Команду

$$A: \quad \Delta_0 = \Delta$$

можно теперь сформировать из заготовки $0 \vee 0 = \Delta$, прибавив к среднему адресу этой заготовки средний адрес ячейки R_1. Это можно выполнить двумя командами:

$$R_1 \quad \wedge (0, F, 0) = R_2,$$
$$(0 \vee 0 = \Delta) \vee \quad R_2 \quad = A.$$

Аналогично формируется и команда B, которую можно записывать в виде $a \vee 0 = z$.

Для формирования команды

$$C: \quad b — z = 0$$

надо взять заготовку $0 — z = 0$ и прибавить адрес b к ее левому адресу. Адрес b записан в среднем адресе ячейки информации. Поэтому его надо после высечения сдвинуть. На формирование команды C придется потратить уже три команды

$$R_0 \quad \wedge \quad (0, F, 0) = R_2$$
$$\overline{14} \quad \leftarrow, \quad R_2 \quad = R_2$$
$$(0 — z = 0) \vee \quad R_2 \quad = C.$$

Точно так же формируются и все остальные команды.

Приведем теперь стандартную программу *Корень* в окончательном виде, со всем формированием и всеми константами:

	1)		Ω	= конец
	2)	$(F \vee F = R_1)$	$+,$ Ω	$= Q$
Q	3)	(0	$\vee$ $Я+2$	$= R_1)$ н. п.
	4)	Q	$—,$ $(0, 1, 1)$	$= P$
P	5)	(0	$\vee$ $Я+1$	$= R_0)$ н. п.
	6)	R_1	$\wedge$ $(0, F, 0)$	$= R_2$
	7)	$(0 \vee 0 = \Delta)$	$\vee$ R_2	$= A$
	8)	R_0	$\wedge$ $(F, 0, 0)$	$= R_2$
	9)	$(0 \vee 0 = z)$	$\vee$ R_2	$= B$
	10)	R_0	$\wedge$ $(0, F, 0)$	$= R_2$
	11)	$\overline{14}$	$\leftarrow,$ R_2	$= R_2$
	12)	$(0 — z = 0)$	$\vee$ R_2	$= C$
	13)	$\overline{30}$	$\leftarrow,$ R_1	$= R_2$

```
   14)    (|0|—|Δ|=0)  ∨    R₂      =D
   15)      R₀         ∧  (0, 0, F)  =R₂
   16)      R₁         ∧  (F, 0, 0)  =R₃
   17)     14          →,   R₃      =R₃
   18)      R₂         ∨    R₃      =R₂
   19)   (Б z 0 0)     ∨    R₂      =E
A  20)   (                  Δ₀      =Δ) н. п.
B  21)   ( a            ∨   0       =z) н. п.
   22) БВ  Я+1              E       Ω
   23)                      γ       =R₀
   24)      z          +    Δ       =z
C  25)   ( b          —     z       =0) н. п.
   26) У₁   Щ       конец            γ
   27) БВ  Я+1              E       Ω
   28)      R₀        ·     γ       =R₁
   29)      R₀        +     0       =0
   30) У₀   γ                 24)    R₀
   31)      Δ         ·  «(−1/8)»    =Δ
D  32)   (|ε|      —        |Δ|     =0) н. п.
   33) У₁   z                 24)    γ
   34)          конец
E  35) (Б   z                 f      x) н. п.
   36)      F         ∨      F      =R₁
   37)      0         ∨      0      =Δ
   38)      0         ∨      0      =z
   39)      0         —      z      =0
   40)     |0|        —     |Δ|     =0
   41) Б    z                 0      0
```

§ 61. Программа для способа хорд и касательных

Пусть известен интервал $[a, b]$ изоляции корня уравнения $f(x)=0$. Рассмотрим программу для одного шага применения комбинированного способа хорд и касательных, предполагая, что написаны три блока, вычисляющие функцию и две ее первые производные. Эти блоки мы будем обозначать $f(\alpha)$, $f'(\alpha)$ и $f''(\alpha)$. Они предполагаются оформленными как стандартные подпрограммы с входной ячейкой (аргументом) α и выходной (ответной) γ.

Основными вопросами, которые следует решить, является выбор точки, в которой нужно проводить касательную, а затем выбор между формулами (2.57) — (3.57) либо (6.57) — (7.57).

Для решения первого из этих вопросов нужно сравнить знаки функции и ее второй производной в какой-либо точке, например, в точке a. Если они совпадают, то касательную следует проводить в точке a, а если различны, то в точке b. Этого можно достичь следующей программой:

0) $\Omega = конец$
1) $a = \alpha$
2) $БВ\ Я + 1\ \ f(\alpha)\ \ \Omega$
3) $\gamma = f(a)$
4) $БВ, Я + 1\ f''(\alpha)\ \ \Omega$
5) $\gamma = R_0$
6) $\gamma \cdot f(a) = R_1$
7) $R_1 + 0 = 0$
8) $У_0\ \ a\ \ c$
9) $b = c$
10) $c = \alpha$
11) $БВ\ Я + 1\ f'(\alpha)\ \ \Omega$
12) $\gamma = f'(c)$

После выполнения команды 12) в ячейках c и $f'(c)$ лежат нужные значения. Остается только выяснить, по какой из формул следует вычислять Δa и Δb. Прежде всего нужно сравнить знак производной со знаком функции, вычисленной в той же точке. Кроме того, нужно еще вычислить $f(b)$, что мы и сделаем в первую очередь.

13) $b = \alpha$
14) $БВ\ Я + 1\ f(\alpha)\ \ \Omega$
15) $\gamma = f(b)$

Теперь нужно сравнить знаки первой и второй производной. Каждая из них вычислена только в одной точке, но в силу условий, наложенных на функцию в § 55, они сохраняют знак на всем участке. Если обе производные имеют одинаковые знаки, то Δa и Δb считаются по формулам (2.57) — (3.57), т. е. способ хорд дает значение Δa, а способ касательных — значение Δb. Если производные имеют противоположные знаки, то воспользуемся формулами (6.57) — (7.57).

Поступим дальше так. Пусть последние команды, вычисляющие поправки соответственно по способу хорд и способу касательных, обозначены буквами T и U. Напишем команды-заготовки T^0 и U^0,

отличающиеся от T и U тем, что они имеют нулевой третий адрес и заготовим константы (0, 0, Δa) и (0, 0, Δb), где Δa и Δb — адреса ячеек для соответствующих поправок. Тогда команды T и U можно сформировать, прибавляя к T^0 и U^0 те или иные третьи адреса в зависимости от обстоятельств:

16) $f'(c) \cdot R_0 = R_0$
17) $R_0 + 0 = 0$
18) $У_1$ $f(b)$ 22) R_2
19) $T^0 \vee (0, 0, \Delta a) = T$
20) $U^0 \vee (0, 0, \Delta b) = U$
21) $Б$ $f(a)$ 24) R_2
22) $T^0 \vee (0, 0, \Delta b) = T$
23) $U^0 \vee (0, 0, \Delta a) = U$

После этого можно уже вычислять сами поправки. Для способа хорд имеем

24) $b - a = R_0$
25) $f(b) - f(a) = R_1$
26) $R_0 \cdot R_2 = R_0$
T 27) $(R_0 : R_1 = \Delta a)$ н. п.

Для способа касательных нужно еще определить, в какой точке следует брать значение функции, для чего мы сравним значение c с одним из концов интервала

28) $a \not\sim c = 0$
29) $У_0$
30) $f(a) : f'(c) = R_0$
31) $Б$
32) $f(b) : f'(c) = R_0$
U 33) $(0 - R_0 = \Delta b)$ н. п.
34) $a - \Delta a = a_1$
35) $b + \Delta b = b_1$
36) *конец*
T^0 37) $R_0 : R_1 = 0$
U^0 38) $0 - R_0 = 0$
39) (0, 0, Δa)
40) (0, 0, Δb)

Выполнив вычисления по приведенной программе, мы получим новый, меньший участок (a_1, b_1), на котором находится корень. Включив этот блок в цикл, мы можем получить значения корня с любой точностью. Такую программу можно написать, например, следующим образом (*Шаг* означает написанный выше блок):

$$
\begin{array}{rl}
 & a_{\text{нач}} = a \\
 & b_{\text{нач}} = b \\
\textit{БВ}\ \textit{Я}+1\ \textit{Шаг}\ \ \Omega & \\
 & b_1 - a_1 = \Delta \\
 & a_1 = a \\
 & \Delta - \varepsilon = 0 \\
 & \hline \\
\textit{У}_0 & b_1 \mid\ b \\
 & a_1 + b_1 = \gamma \\
 & \gamma \cdot \langle\!\langle \tfrac{1}{2} \rangle\!\rangle = \gamma \\
 & \textit{стоп}.
\end{array}
$$

Приведенная программа не является стандартной. Для того чтобы написать стандартную программу, необходимо, как мы это делали в предыдущем параграфе, сформировать команды, зависящие от информации. Легко видеть, что в нашей программе таких команд довольно много.

§ 62. Программирование итерационного процесса

Программирование итерационного процесса, собственно, сводится к написанию обычного итерационного цикла, который уже рассматривался в § 21. Там же было приведено и некоторое число примеров. Мы можем теперь написать в самом общем виде программу для решения уравнения $x = \varphi(x)$ методом итераций, в предположении, что итерационный процесс сходится. Если блок счета $\varphi(x)$ написан как стандартная подпрограмма, считающая $\gamma = \varphi(\alpha)$, то программу нахождения корня можно записать в таком виде:

$$
\begin{array}{lll}
1) & & x_0 = \alpha \\
2) & \textit{БВ}\ \textit{Я}+1\ \varphi(\alpha)\ \ \Omega & \\
3) & \gamma - \alpha = \delta & \\
4) & |\delta| - |\varepsilon| = 0 & \\
 & \hline & \\
5) & \textit{У}_0\ \ \gamma \mid\ \alpha & \\
6) & \textit{стоп} &
\end{array}
$$

Здесь x_0 — нулевое приближение, которое считается известным, а ε — требуемая точность.

Не составляет никакого труда превратить эту программу в стандартную. Пусть, например, обращение к программе *Итерация* имеет вид

$$\begin{array}{llll} Я) & БВ\ Я+2 & Итерация & Ω \\ Я+1) & x_0 & \varphi(\alpha) & \varepsilon, \end{array}$$

где x_0 — адрес ячейки, содержащей начальное приближение, $\varphi(\alpha)$ — адрес начала блока счета функции $\varphi(x)$ и ε — ячейка, содержащая требуемую точность. Как видно из приведенной программы, в ней требуется формировать команды 1), 2) и 4), которые мы назовем соответственно A, B, C.

Перенос информации из ячейки $Я+1$ в рабочую ячейку R_0 достигается с помощью команд

$$\begin{array}{l} (F \vee F = R_0) +, \quad Ω \quad = Q \\ Q:(\quad 0 \quad \vee Я+1 = R_0)\ \text{н. п.} \end{array}$$

Команду 1) удобно формировать в виде $x_0 \vee 0 = \alpha$, для чего достаточно написать

$$\begin{array}{l} R_0 \quad \wedge (F, 0, 0) = R_1 \\ (0 \vee 0 = \alpha) \vee \quad R_1 \quad = A. \end{array}$$

Так же легко формируются и остальные команды. Только для команды C высеченный адрес ε необходимо будет сдвинуть из правого адреса в средний.

Стандартная программа для метода итераций будет иметь следующий вид:

	1)			Ω	= *конец*	
	2)	$(F \vee F = R_0)$	+ ,	Ω	$= Q$	
Q	3)	(0	$\vee$	$Я+1$	$= R_0$)	н. п.
	4)	R_0	$\wedge$	$(F, 0, 0)$	$= R_1$	
	5)	$(0 \vee 0 = \alpha)$	$\vee$	R_1	$= A$	
	6)	R_0	$\wedge$	$(0, F, 0)$	$= R_1$	
	7)	(*БВ* 13) 0 Ω)	$\vee$	R_1	$= B$	
	8)	R_0	$\wedge$	$(0, 0, F)$	$= R_1$	
	9)	$\overline{14}$	$\leftarrow$,	R_1	$= R_1$	
	10)	$(\lvert\delta\rvert - \lvert 0\rvert = 0)$	$\vee$	R_1	$= C$	
A	11)	(x_0	$\vee$	0	$= \alpha$)	н. п.
B	12)	(*БВ* 13)		$\varphi(\alpha)$	Ω)	н. п.
	13)	γ	—	α	$= \delta$	
C	14)	($\lvert\delta\rvert$	—	$\lvert\varepsilon\rvert$	$= 0$)	н. п.
	15)	$У_0$ γ		B	α	
	16)		*конец*			
	17)	F	$\vee$	F	$= R_0$	
	18)	0	$\vee$	0	$= \alpha$	
	19)	*БВ* 13)		0	Ω	
	20)	$\lvert\delta\rvert$	—	$\lvert 0\rvert$	$= 0$	

ГЛАВА XIII

СИСТЕМЫ УРАВНЕНИЙ

§ 63. Некоторые сведения о векторах и матрицах

В этом параграфе будут приведены основные сведения о векторах и матрицах, которые потребуются при рассмотрении систем линейных алгебраических уравнений.

Вектор задается двумя точками пространства — его началом и концом. Представим себе, что все векторы откладываются из одной и той же точки пространства — начала координат. Тогда для задания вектора достаточно указать одну точку — конец вектора или, что то же самое, три числа — координаты этой точки.

Таким образом, каждый вектор пространства определяется упорядоченной тройкой чисел — координат конца вектора, если считать его отложенным из начала координат. Наоборот, каждая упорядоченная тройка чисел определяет единственный вектор, соединяющий начало координат с точкой, для которой заданная тройка чисел служит координатами.

Мы будем *отождествлять вектор x и упорядоченную тройку чисел x_1, x_2, x_3*, которые называются *координатами вектора x* или его *составляющими*. Действия с векторами легко представить как действия с тройками чисел.

В самом деле, при умножении вектора на число все его координаты умножаются на то же число, так что

$$\lambda \cdot (x_1, x_2, x_3) = (\lambda x_1, \lambda x_2, \lambda x_3). \tag{1.63}$$

Аналогично,

$$(x_1, x_2, x_3) \pm (y_1, y_2, y_3) = (x_1 \pm y_1, x_2 \pm y_2, x_3 \pm y_3). \tag{2.63}$$

Модуль (длина) вектора $x = (x_1, x_2, x_3)$ находится по формуле

$$|x| = \sqrt{x_1^2 + x_2^2 + x_3^2}. \tag{3.63}$$

Большую роль играет понятие скалярного произведения векторов. *Скалярным произведением* векторов x и y называется число, равное произведению модулей векторов на косинус угла между ними:

$$x \cdot y = |x| \cdot |y| \cos(\widehat{x, y}).$$

Если векторы x и y имеют координаты соответственно (x_1, x_2, x_3) и (y_1, y_2, y_3), то скалярное произведение векторов равно сумме произведений

$$x \cdot y = x_1 y_1 + x_2 y_2 + x_3 y_3. \tag{4.63}$$

По аналогии с произведенным выше отождествлением, мы можем теперь определить понятие n-мерных векторов и действия над ними. Прежде всего, *n-мерным вектором будем называть упорядоченную систему n действительных чисел* $(x_1, x_2, \ldots, x_n)$. Умножение вектора $(x_1, x_2, \ldots, x_n)$ на число λ *определяется* формулой

$$\lambda \cdot (x_1, x_2, \ldots, x_n) = (\lambda x_1, \lambda x_2, \ldots, \lambda x_n), \tag{5.63}$$

а сумма и разность векторов $(x_1, x_2, \ldots, x_n)$ и $(y_1, y_2, \ldots, y_n)$ — формулой

$$(x_1, x_2, \ldots, x_n) \pm (y_1, y_2, \ldots, y_n) = (x_1 \pm y_1, x_2 \pm y_2, \ldots, x_n \pm y_n). \tag{6.63}$$

Модулем n-мерного вектора $x = (x_1, x_2, \ldots, x_n)$ *называется* действительное число

$$|x| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}. \tag{7.63}$$

Точно так же скалярным произведением двух n-мерных векторов $(x_1, x_2, \ldots, x_n)$ и $(y_1, y_2, \ldots, y_n)$ называется действительное число

$$x \cdot y = x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n. \tag{8.63}$$

Формулы (5.63) — (8.63) представляют собой полную аналогию формулам (1.63) — (4.63). Однако между ними имеется существенное различие. В то время как формулы (1.63) — (4.63) являются алгебраическим выражением геометрических определений и свойств векторов в трехмерном пространстве, формулы (5.63) — (8.63) служат для n-мерных векторов *определениями* соответствующих операций.

В ряде случаев приходится рассматривать не один вектор, а *систему векторов*. Запись координат векторов такой системы представляет собой прямоугольную таблицу, которую называют *матрицей*. Элементы матрицы обычно обозначают одной буквой, например a, с двумя номерами (*индексами*). Первый из них означает номер строки, в которой стоит данный элемент, второй — номер столбца. Слева и справа ставят круглые скобки или двойные вертикальные черточки.

Например, матрица из трех строк и четырех столбцов (*порядка* 3×4) выглядит так:

$$\begin{Vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{Vmatrix}.$$

Эту матрицу можно рассматривать как систему из трех четырехмерных векторов-с т р о к или же как систему четырех трехмерных векторов-с т о л б ц о в.

Чаще всего приходится иметь дело с матрицами, в которых число строк и число столбцов одинаковы. Матрицу, имеющую n строк и n столбцов, называют *квадратной матрицей n-го порядка.*

Естественно и просто определяются операции умножения произвольной матрицы на число и сложения и вычитания матриц одинакового порядка. Мы не будем останавливаться на этих определениях, так как указанные операции нам не потребуются. Для дальнейшего нам нужно ознакомиться с операциями умножения матрицы на вектор и перемножения матриц.

Чтобы определить произведение матрицы на вектор, запишем вектор-множитель в виде матрицы, состоящей из одного столбца, который будет содержать координаты вектора. Тогда *произведением матрицы на вектор мы будем называть такой вектор-столбец, координаты которого равны скалярным произведениям векторов-строк матрицы на данный вектор-множитель,*

$$\begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{Vmatrix} \cdot \begin{Vmatrix} b_1 \\ b_2 \\ \dots \\ b_n \end{Vmatrix} = \begin{Vmatrix} c_1 \\ c_2 \\ \dots \\ c_m \end{Vmatrix}, \tag{9.63}$$

где

$$\begin{aligned} c_1 &= a_{11} b_1 + a_{12} b_2 + \dots + a_{1n} b_n, \\ c_2 &= a_{21} b_1 + a_{22} b_2 + \dots + a_{2n} b_n, \\ &\dots\dots\dots\dots\dots\dots \\ c_m &= a_{m1} b_1 + a_{m2} b_2 + \dots + a_{mn} b_n. \end{aligned} \tag{10.63}$$

Из приведенного определения следует, что размерность (т. е. число координат) вектора-множителя должна быть равна размерности вектора-строки матрицы, т. е. числу ее столбцов. В качестве произведения матрицы на вектор получаем вектор-столбец, размерность которого равна числу строк матрицы.

При определении перемножения матриц ограничимся, для простоты, квадратными матрицами. *Произведением двух квадратных матриц одинакового порядка называется такая квадратная матрица*

того же порядка, элементы которой равны скалярным произведениям строк первой матрицы на столбцы второй,

$$\begin{Vmatrix} a_{11} & a_{12} \dots a_{1n} \\ a_{21} & a_{22} \dots a_{2n} \\ \dots & \dots \\ a_{n1} & a_{n2} \dots a_{nn} \end{Vmatrix} \cdot \begin{Vmatrix} b_{11} & b_{12} \dots b_{1n} \\ b_{21} & b_{22} \dots b_{2n} \\ \dots & \dots \\ b_{n1} & b_{n2} \dots b_{nn} \end{Vmatrix} = \begin{Vmatrix} c_{11} & c_{12} \dots c_{1n} \\ c_{21} & c_{22} \dots c_{2n} \\ \dots & \dots \\ c_{n1} & c_{n2} \dots c_{nn} \end{Vmatrix}, \tag{11.63}$$

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} + \dots + a_{1n}b_{n1}, \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} + \dots + a_{1n}b_{n2}, \\ &\dots \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} + \dots + a_{2n}b_{n1}, \\ &\dots \end{aligned}$$

и, вообще,

$$c_{ik} = a_{i1}b_{1k} + a_{i2}b_{2k} + \dots + a_{in}b_{nk}, \tag{12.63}$$

т. е. *элемент из i-й строки и k-го столбца матрицы-произведения равен скалярному произведению i-й строки первой матрицы на k-й столбец второй.*

Легко убедиться в том, что *произведение квадратных матриц не обладает коммутативностью*, т. е. от перемены мест сомножителей произведение меняется. Действительно, уже для матриц второго порядка имеем

$$\begin{Vmatrix} 1 & -1 \\ 3 & 2 \end{Vmatrix} \cdot \begin{Vmatrix} -2 & 4 \\ -4 & 7 \end{Vmatrix} = \begin{Vmatrix} 1\cdot(-2)+(-1)\cdot(-4) & 1\cdot 4+(-1)\cdot 7 \\ 3\cdot(-2)+2\cdot(-4) & 3\cdot 4+2\cdot 7 \end{Vmatrix} = \begin{Vmatrix} 2 & -3 \\ -14 & 26 \end{Vmatrix}$$

$$\begin{Vmatrix} -2 & 4 \\ -4 & 7 \end{Vmatrix} \cdot \begin{Vmatrix} 1 & -1 \\ 3 & 2 \end{Vmatrix} = \begin{Vmatrix} (-2)\cdot 1+4\cdot 3 & (-2)\cdot(-1)+4\cdot 2 \\ (-4)\cdot 1+7\cdot 3 & (-4)(-1)+7\cdot 2 \end{Vmatrix} = \begin{Vmatrix} 10 & 10 \\ 17 & 18 \end{Vmatrix}.$$

Особое значение имеет квадратная матрица, имеющая единицы на главной диагонали, а все остальные элементы — равными нулю. Такую матрицу называют *единичной* и обозначают буквой E:

$$E = \begin{Vmatrix} 1 & 0 \dots 0 \\ 0 & 1 \dots 0 \\ \dots & \dots \\ 0 & 0 \dots 1 \end{Vmatrix}.$$

Название «*единичная матрица*» оправдывается тем, что для любой квадратной матрицы A справедливо равенство

$$A \cdot E = E \cdot A = A. \tag{13.63}$$

Иначе говоря, единичная матрица играет такую же роль в умножении матриц, как и число 1 в умножении чисел. Равенство (13.63) легко устанавливается непосредственным перемножением, исходя из определений (11.63) и (12.63). Рекомендуем читателю проделать соответствующие выкладки самостоятельно для двух случаев: сначала для какой-либо конкретной матрицы A, а затем и в общем виде.

Во многих случаях, наряду с матрицей A, приходится рассматривать также и так называемую *обратную* матрицу.

Пусть дана квадратная матрица A. *Обратной матрицей для нее называется квадратная матрица того же порядка, обозначаемая* A^{-1} *и обладающая свойством*

$$A \cdot A^{-1} = A^{-1} \cdot A = E, \tag{14.63}$$

где E *— единичная матрица того же порядка.* Из равенства (14.63) следует, что матрица A является обратной для A^{-1}, т. е. матрицы A и A^{-1} будут *взаимно обратными.*

О существовании такой матрицы речь будет идти в § 64. Ниже (в § 66) будет рассмотрен способ нахождения элементов обратной матрицы.

В качестве примера использования приведенных определений рассмотрим матричную форму записи системы линейных уравнений. Пусть дана система линейных уравнений

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned}\right\} \tag{15.63}$$

Обозначим через A матрицу, составленную из коэффициентов системы (15.63)

$$A = \begin{Vmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{Vmatrix},$$

через B — вектор-столбец, координаты которого суть правые части уравнений системы (15.63)

$$B = \begin{Vmatrix} b_1 \\ b_2 \\ \ldots \\ b_n \end{Vmatrix},$$

и через X — вектор-столбец, координатами которого служат значения неизвестных

$$X = \begin{Vmatrix} x_1 \\ x_2 \\ \ldots \\ x \end{Vmatrix}.$$

Тогда систему (15.63) можно записать в виде одного равенства

$$A \cdot X = B. \tag{16.63}$$

Чтобы в этом убедиться, достаточно для произведения матрицы A на вектор X воспользоваться определением (10.63) и сравнить полученную из (16.63) систему равенств с системой (15.63).

Зная матрицу A^{-1}, обратную матрице A, мы можем найти неизвестный вектор X, умножая обе части равенства (16.63) слева на A^{-1}. Мы получим

$$X = A^{-1}B.$$

Таким образом, задача решения системы линейных уравнений сводится к нахождению для матрицы коэффициентов обратной матрицы и умножению ее на вектор-столбец свободных членов.

§ 64. Определители

В § 2 мы уже встречались с понятием определителя второго порядка. Пусть дана квадратная матрица второго порядка

$$A = \begin{Vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{Vmatrix}.$$

Ее определителем *) называют число, определенное равенством

$$D = \det A = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}. \tag{1.64}$$

Таким образом, *определитель есть функция матрицы:* каждой квадратной матрице второго порядка ставится в соответствие число, равное разности произведений диагональных элементов. Как видно из формулы (1.64), для обозначения определителя слева и справа от элементов матрицы ставится по одной вертикальной черте вместо двух.

С помощью определителей второго порядка удобно записывать решение системы линейных уравнений с двумя неизвестными. Рассмотрим систему

$$\left.\begin{aligned} a_{11}x + a_{12}y &= b_1, \\ a_{21}x + a_{22}y &= b_2. \end{aligned}\right\} \tag{2.64}$$

Умножив первое из уравнений системы на a_{22}, а второе на $-a_{12}$ и сложив, получим

$$(a_{11}a_{22} - a_{12}a_{21})\,x = b_1a_{22} - b_2a_{12},$$

откуда

$$x = \frac{b_1a_{22} - b_2a_{12}}{a_{11}a_{22} - a_{12}a_{21}}.$$

Аналогично находим

$$y = \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{12}a_{21}}.$$

*) Иногда вместо определителя говорят также «детерминант».

Воспользовавшись определителями, мы можем записать решение системы (2.64) в виде

$$x=\frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}, \qquad y=\frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}. \tag{3.64}$$

Определитель $D=\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}$, стоящий в знаменателе, называют *определителем системы* (2.64). Формулы (3.64), очевидно, имеют смысл при $D\neq 0$.

Аналогичные функции — *определители* — можно построить и для матриц более высокого порядка. При этом естественно определять их так, чтобы для систем линейных уравнений соответствующего порядка имели место формулы, подобные формулам (3.64).

Начнем с рассмотрения системы трех линейных уравнений с тремя неизвестными. Запишем эту систему в виде

$$\left.\begin{aligned} a_{11}x+a_{12}y+a_{13}z&=b_1,\\ a_{21}x+a_{22}y+a_{23}z&=b_2,\\ a_{31}x+a_{32}y+a_{33}z&=b_3. \end{aligned}\right\} \tag{4.64}$$

Из первых двух уравнений, перенеся z вправо, найдем

$$x=\frac{\begin{vmatrix} b_1-a_{13}z & a_{12} \\ b_2-a_{23}z & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}, \qquad y=\frac{\begin{vmatrix} a_{11} & b_1-a_{13}z \\ a_{21} & b_2-a_{23}z \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}.$$

Подставив найденные выражения для x и y в третье уравнение (4.64) и приведя его затем к общему знаменателю, получим

$$a_{31}\begin{vmatrix} b_1-a_{13}z & a_{12} \\ b_2-a_{23}z & a_{22} \end{vmatrix}+a_{32}\begin{vmatrix} a_{11} & b_1-a_{13}z \\ a_{21} & b_2-a_{23}z \end{vmatrix}+a_{33}\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}z=$$

$$=b_3\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}.$$

Вычислив все определители второго порядка, определим коэффициент при z в полученном уравнении. Он оказывается равным

$$-a_{13}a_{22}a_{31}+a_{12}a_{23}a_{31}-a_{11}a_{23}a_{32}+a_{13}a_{21}a_{32}+a_{11}a_{22}a_{33}-a_{12}a_{21}a_{33},$$

а общее выражение для z будет иметь вид

$$z=\frac{b_1a_{22}a_{33}+b_3a_{12}a_{23}+b_2a_{13}a_{32}-b_3a_{13}a_{22}-b_2a_{12}a_{33}-b_1a_{23}a_{32}}{a_{11}a_{22}a_{33}+a_{12}a_{23}a_{31}+a_{13}a_{21}a_{32}-a_{13}a_{22}a_{31}-a_{12}a_{21}a_{33}-a_{11}a_{23}a_{32}}.$$

Знаменатель полученной дроби мы и примем за выражение определителя третьего порядка. Таким образом, *определителем третьего порядка называется выражение*

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - \\ - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}. \quad (5.64)$$

Вместо того, чтобы выводить теперь для системы уравнений с тремя неизвестными формулы, аналогичные (3.64), мы пойдем другим путем. Прежде всего внимательно проанализируем выражение (5.64), с тем, чтобы, отправляясь от него, дать общее определение для определителя, как функции произвольной квадратной матрицы n-го порядка. Далее, установим в общем случае ряд свойств определителей n-го порядка. После этого формулы для решения системы линейных уравнений, аналогичные формулам (3.64), могут быть выведены в самом общем виде.

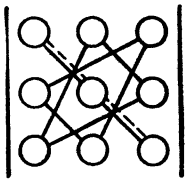
Рассмотрим выражение (5.64) и сравним его с выражением (1.64) для определителя второго порядка. В обоих случаях мы замечаем, что определитель равен алгебраической сумме произведений, причем каждое из произведений содержит для определителя второго порядка два сомножителя, а для определителя третьего порядка — три сомножителя. При этом каждое произведение содержит сомножители, принадлежащие непременно различным строкам и различным столбцам матрицы. Кроме того, легко подсчитать, что для матрицы второго порядка можно составить два, а для матрицы третьего порядка — шесть различных таких произведений, т. е. именно столько, сколько и есть в выражениях (1.64) и (5.64). Таким образом, можно сказать, что *определитель второго или третьего порядка есть сумма всевозможных произведений элементов матрицы соответственно по два или по три сомножителя, выбранных так, чтобы никакие два множителя ни в каком из произведений не принадлежали одной и той же строке или одному и тому же столбцу.*

Сформулированное выше определение переносится на случай матрицы любого порядка. Не хватает лишь правила знаков, позволяющего присвоить каждому из составленных произведений определенный знак. Для формулировки такого правила *) рассмотрим более подробно, как расставлены знаки у произведений в выражении (5.64).

---

*) Подчеркнем еще раз, что определитель третьего порядка о п р е д е л я е т с я равенством (5.64). Мы хотим сформулировать правило таким образом, чтобы оно давало именно такие знаки, какими они являются в выражении (5.64) и, кроме того, чтобы его можно было использовать для определителей более высокого порядка.

Изобразим элементы матрицы третьего порядка кружками и соединим отрезками кружки, соответствующие элементам, входящим в одно произведение. Собрав вместе произведения, входящие в выражение (5.64) с одинаковым знаком, получим схему, изображенную в табл. 1.64. Если назвать диагональ, идущую слева направо сверху



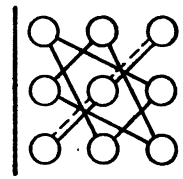


Таблица 1.64.

вниз, — главной диагональю матрицы, а другую диагональ — побочной, то правило знаков для определителя третьего порядка можно формулировать так: *произведения, члены которых расположены по главной диагонали или в вершинах треугольника, одна из сторон которого параллельна главной диагонали, берутся со знаком плюс; наоборот, произведения, члены которых расположены по побочной диагонали или в вершинах треугольника, одна из сторон которого параллельна побочной диагонали, берутся со знаком минус.* Это правило называют *правилом треугольника.*

Пример 1.64. Вычислим определители третьего порядка:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \text{ и } \begin{vmatrix} 1 & -3 & 2 \\ 5 & 4 & -1 \\ 0 & -2 & 3 \end{vmatrix}.$$

Пользуясь правилом треугольника, находим

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1\cdot5\cdot9+2\cdot6\cdot7+4\cdot8\cdot3-7\cdot5\cdot3-4\cdot2\cdot9-1\cdot8\cdot6=0,$$

$$\begin{vmatrix} 1 & -3 & 2 \\ 5 & 4 & -1 \\ 0 & -2 & 3 \end{vmatrix} = 1\cdot4\cdot3+(-3)(-1)\cdot0+$$

$$+5\cdot(-2)\cdot2-0\cdot4\cdot2-5\cdot(-3)\cdot3-1\cdot(-1)\cdot(-2)=35.$$

Недостатком правила треугольника является то, что оно относится к определителю третьего порядка и не может быть перенесено на более общий случай. Для более удобной формулировки правила знаков введем следующие термины. Назовем отрезки, соединяющие элементы, входящие в данное произведение, *отрезками*

положительного наклона, если они идут слева направо сверху вниз. Отрезки, идущие, наоборот, слева направо снизу вверх, назовем *отрезками отрицательного наклона.*

Например, отрезок, соединяющий элементы a_{11} и a_{23}, — положительного наклона (см. табл. 1.64), а отрезок, соединяющий элементы a_{21} и a_{12}, имеет отрицательный наклон.

Обратимся снова к схеме, изображенной в табл. 1.64. Мы заметим, что если соединить элементы, входящие в произведение, отрезками попарно, то произведения, входящие в определитель со знаком плюс, либо не содержат отрезков отрицательного наклона, либо содержат два таких отрезка. Наоборот, произведения, имеющие знак минус, содержат один или три таких отрезка. Воспользовавшись этим замечанием, мы можем теперь сформулировать правило знаков так: *соединим все элементы матрицы, входящие в данное произведение, отрезками попарно; если число отрезков отрицательного наклона будет при этом четным, то данное произведение берется со знаком плюс; произведение, которому соответствует нечетное число отрезков отрицательного наклона, берется со знаком минус.*

Легко проверить, что и правило треугольника и правило (1.64) для определителя второго порядка являются частными случаями этого правила. Кроме того, это правило без всякого изменения может быть применено для вычисления определителя любого порядка.

Например, для определителя четвертого порядка произведение $a_{13}a_{22}a_{34}a_{41}$ входит со знаком плюс, потому что из шести отрезков, соединяющих попарно эти элементы в матрице, четыре (четное число) являются отрезками отрицательного наклона (см. табл. 2.64; отрезки

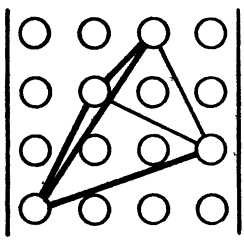


Таблица 2.64.

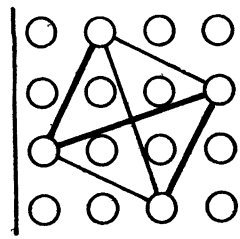


Таблица 3.64.

отрицательного наклона набраны жирно). Произведение $a_{12}a_{24}a_{31}a_{43}$ входит со знаком минус (табл. 3.64; три отрезка отрицательного наклона). Произведение $a_{14}a_{22}a_{31}a_{44}$ в определитель вообще входить не может, так как содержит два элемента из одного и того же столбца (a_{14} и a_{44} принадлежат четвертому столбцу матрицы).

Мы можем теперь дать общее определение для определителя любого порядка. *Каждой квадратной матрице n-го порядка A*

ставится в соответствие определенное число D, называемое определителем этой матрицы,

$$D = \det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix},$$

которое равно сумме всех возможных произведений по n сомножителей в каждом, составленных из элементов матрицы так, чтобы никакие два сомножителя из одного произведения не принадлежали одной строке или одному столбцу; произведение берется со знаком плюс или минус смотря по тому, четным или нечетным оказывается число отрезков отрицательного наклона, соединяющих попарно элементы матрицы, входящие в данное произведение.

Перейдем теперь к рассмотрению свойств определителей n-го порядка, которые можно вывести, исходя из приведенного определения. При этом в ряде случаев для сокращения речи мы будем говорить о строках и столбцах определителя вместо строк и столбцов матрицы, как следует говорить на самом деле.

1. *При замене строк матрицы столбцами, а столбцов — строками*) величина определителя не меняется.*

Например, для определителя четвертого порядка это свойство утверждает, что

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{21} & a_{31} & a_{41} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{vmatrix}.$$

Доказательство. Прежде всего заметим, что в определитель, полученный после транспозиции матрицы, входят все те же произведения, что и в первоначальный. Действительно, каждое произведение состоит из n множителей, принадлежащих разным строкам и разным столбцам. Но после транспозиции все эти множители будут по-прежнему принадлежать разным столбцам и разным строкам, так что оно будет входить и в новый определитель.

Остается проверить, что все произведения сохранят свои знаки. В этом, однако, легко убедиться, если заметить, что транспозиция матрицы может быть осуществлена путем ее поворота вокруг главной диагонали, а направление наклона отрезков, соединяющих элементы, входящие в каждое данное произведение, при таком повороте

*) Такое преобразование матрицы называют *транспозицией.*

сохраняется. Следовательно, новый определитель будет состоять из тех же произведений, что и старый, и каждое произведение войдет в определитель с тем же самым знаком, что и в первоначальный, т. е. величина определителя сохранится.

Доказанное свойство играет большую роль, так как констатирует *равноправие строк и столбцов определителя.* Все дальнейшие свойства мы будем формулировать для строк, но, в силу доказанного, они имеют место также и для столбцов.

2. *Если одна из строк (или столбцов) определителя состоит из нулей, то определитель равен нулю.*

Для доказательства заметим, что каждое произведение, входящее в определитель, содержит по одному множителю из каждой строки. Поэтому все произведения содержат нулевой множитель, а значит, равны нулю. Отсюда следует, что равен нулю и определитель.

3. *При перестановке двух строк матрицы определитель меняет знак на противоположный.* Как было указано, это свойство справедливо также и для столбцов, т. е. перестановка двух столбцов определителя также приводит к изменению его знака.

Рассмотрим сначала случай, когда переставляются две соседние строки. Как и в первом свойстве, каждое произведение, входившее в определитель до перестановки строк, будет входить в него и после перестановки. Решим вопрос о знаке каждого произведения.

Отрезки, соединяющие элементы произведения, не входящие в переставляемые строки, останутся без изменения. Отрезки, один из концов которых принадлежит переставляемой строке, сдвинутся, но направление их наклона останется прежним. Кроме того, имеется еще (один!) отрезок, соединяющий элементы, стоящие в переставляемых строках. Этот отрезок обязательно изменит направление наклона на противоположное, т. е. отрезок отрицательного наклона превратится в отрезок положительного наклона, и наоборот *).

Таким образом, число отрезков отрицательного наклона, соединяющих элементы, входящие в данное произведение, изменится точно на единицу. Это означает, что каждое произведение будет входить в определитель с переставленными строками с противоположным знаком по сравнению с первоначальным. Следовательно, мы доказали, что при перестановке двух соседних строк определитель меняет знак.

Пусть теперь переставляются строки с номерами l и m. Примем для определенности, что $l < m$. Будем переставлять l-ю строку с $l+1$-й, затем с $l+2$-й и т. д. до тех пор, пока она не станет на место m-й. При этом каждый раз мы переставляем соседние строки, в результате чего определитель каждый раз меняет свой знак. Для

*) Советуем для большей наглядности проиллюстрировать эти утверждения рисунком.

того чтобы l-я строка стала на место m-й, потребуется $m-l$ перестановок. Теперь l-я строка стоит на нужном месте, а m-я находится на месте $m-1$-й.

Чтобы переставить m-ю строку на место l-й, меняя местами соседние, потребуется теперь $(m-1)-l$ перестановок. Итак, для того чтобы поменять местами строки с номерами l и m, переставляя между собою только соседние, требуется всего

$$(m-l)+(m-1)-l=2(m-l)-1$$

перестановок, т. е. нечетное число. Поскольку при каждой перестановке соседних строк определитель меняет знак, то он изменит его на противоположный и при перестановке любых двух строк.

4. *Определитель с двумя равными строками (или равными столбцами) равен нулю.*

В самом деле, пусть такой определитель равен D. Переставим в нем равные строки. С одной стороны, все произведения сохранят свое значение, так как переставлялись одинаковые элементы, т. е. величина определителя не изменится. С другой стороны, в силу свойства 3 определитель должен изменить знак, т. е. будет равен $-D$. Поэтому $D=-D$, откуда следует $D=0$.

5. *Если все элементы одной из строк (или столбцов) определителя содержат общий множитель, то его можно вынести за знак определителя:*

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ \lambda a_{i1} & \lambda a_{i2} & \dots & \lambda a_{in} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \lambda \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \tag{6.64}$$

Доказательство. Все произведения, сумма которых составляет определитель, содержат один и только один множитель, принадлежащий i-й строке. Поэтому общий множитель λ, входящий во все элементы i-й строки, можно вынести за скобку, внутри которой останется аналогичный определитель.

6. *Определитель, две строки (или столбца) которого пропорциональны, равен нулю.*

Это свойство является непосредственным следствием двух предыдущих: вынося из одной строки за знак определителя коэффициент пропорциональности, получаем определитель с двумя равными строками.

7. *Если каждый элемент одной из строк (или столбцов) определителя представляет собой сумму двух слагаемых, то данный определитель равен сумме двух других определителей того же порядка. Все элементы обоих определителей, кроме выделенной*

строки, совпадают с соответствующими элементами исходного определителя, а на месте выделенной строки стоят соответственно только первые или только вторые слагаемые:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a'_{k1}+a''_{k1} & a'_{k2}+a''_{k2} & \dots & a'_{kn}+a''_{kn} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a'_{k1} & a'_{k2} & \dots & a'_{kn} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} + \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a''_{k1} & a''_{k2} & \dots & a''_{kn} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \quad (7.64)$$

Доказательство этого свойства вполне аналогично доказательству свойства **5**. В самом деле, все произведения, входящие в определитель, содержат один и только один множитель, принадлежащий выделенной k-й строке. Любое произведение выглядит так:

$$a_{1i}a_{2j}\dots(a'_{kl}+a''_{kl})\dots a_{np},$$

поэтому его можно представить в виде

$$a_{1i}a_{2j}\dots a'_{kl}\dots a_{np}+a_{1i}a_{2j}\dots a''_{kl}\dots a_{np}.$$

Собрав отдельно произведения, содержащие a'_{kl} и a''_{kl}, легко заметим, что они представляют собою два определителя, записанные в правой части равенства (7.64).

8. *Если к одной из строк определителя прибавить любую другую строку, умноженную на любой постоянный множитель, то величина определителя не изменится.* Аналогичное утверждение справедливо и для столбцов:

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \dots & \dots & \dots & \dots \\ a_{k1} & a_{k2} & \dots & a_{kn} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \dots & \dots & \dots & \dots \\ a_{k1}+\lambda a_{i1} & a_{k2}+\lambda a_{i2} & \dots & a_{kn}+\lambda a_{in} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \quad (8.64)$$

Действительно, если определитель, стоящий в (8.64) справа, представить как сумму двух определителей по свойству **7**, то легко заметить, что первый из них совпадает с определителем, стоящим слева, а

второй содержит две пропорциональные строки и, следовательно, равен нулю в силу свойства **6**. Тем самым равенство (8.64) доказано.

9. *Если одна из строк определителя равна сумме других строк с некоторыми коэффициентами*), то такой определитель равен нулю.* Аналогичное свойство имеет место и для столбцов.

В самом деле, вычитая из этой строки остальные строки, умноженные на соответствующие коэффициенты, придем к определителю, одна из строк которого состоит из нулей, т. е. он равен нулю вследствие свойства **2**. Так как в силу свойства **8** величина определителя при таких вычитаниях не изменяется, то и первоначальный определитель равен нулю.

Пример 2.64. Вычислить определитель

$$\begin{vmatrix} 1 & 2 & 2 & 5 \\ 0 & -1 & 1 & 2 \\ 3 & 1 & -1 & 1 \\ 2 & 3 & 0 & 2 \end{vmatrix}.$$

Заметим, что четвертый столбец определителя равен сумме первого столбца и удвоенного третьего. Действительно,

$$\begin{aligned} 1+2\cdot 2 &= 5, \\ 0+2\cdot 1 &= 2, \\ 3+2\cdot(-1) &= 1, \\ 2+2\cdot 0 &= 2. \end{aligned}$$

Поэтому вследствие свойства **9** этот определитель равен нулю.

Пример 3.64. Вычислить определитель

$$\begin{vmatrix} -2 & 1 & 2 & 0 \\ 1 & 3 & 3 & -7 \\ 2 & 2 & -1 & -3 \\ -5 & -1 & 4 & 2 \end{vmatrix}.$$

Прибавив к первому столбцу определителя второй, третий и четвертый столбцы, приведем его к виду

$$\begin{vmatrix} -2 & 1 & 2 & 0 \\ 1 & 3 & 3 & -7 \\ 2 & 2 & -1 & -3 \\ -5 & -1 & 4 & 2 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 2 & 0 \\ 0 & 3 & 3 & -7 \\ 0 & 2 & -1 & -3 \\ 0 & -1 & 4 & 2 \end{vmatrix}.$$

---

*) В таком случае говорят, что одна из строк является *линейной комбинацией* остальных.

Так как в каждое произведение должен входить элемент из первого столбца, то отличными от нуля останутся лишь те произведения, в которые входит элемент a_{11}. Что касается таких произведений, то ясно, что это могут быть все возможные произведения, составленные из элементов 2, 3, 4 строк и 2, 3, 4 столбцов. Поскольку у нас $a_{11}=1$, а отрезки, соединяющие элемент a_{11} с другими элементами матрицы, обязательно имеют положительный наклон, то наш определитель четвертого порядка оказывается равным определителю третьего порядка, стоящему в правом нижнем углу, т. е. получающемуся вычеркиванием первой строки и первого столбца матрицы:

$$\begin{vmatrix} 1 & 1 & 2 & 0 \\ 0 & 3 & 3 & -7 \\ 0 & 2 & -1 & -3 \\ 0 & -1 & 4 & 2 \end{vmatrix} = \begin{vmatrix} 3 & 3 & -7 \\ 2 & -1 & -3 \\ -1 & 4 & 2 \end{vmatrix}.$$

Полученный определитель третьего порядка можно вычислить по правилу треугольника. Окончательно получаем

$$\begin{vmatrix} 3 & 3 & -7 \\ 2 & -1 & -3 \\ -1 & 4 & 2 \end{vmatrix} = 3\cdot(-1)\cdot 2 + 3\cdot(-3)\cdot(-1) + 2\cdot 4\cdot(-7) -$$
$$-(-1)\cdot(-1)(-7) - 4\cdot 3\cdot(-3) - 2\cdot 3\cdot 2 = -22.$$

Рассуждение, подобное только что приведенному в примере 3.64, если его удастся провести в общем виде, было бы очень важным, так как оно позволило нам свести задачу вычисления определителя четвертого порядка к вычислению определителя меньшего, третьего порядка. В рассмотренном примере мы пользовались тем, что все элементы первого столбца, кроме одного элемента, были нулями. Однако оказывается, что и без этого можно сводить вычисление определителя к вычислению определителей меньшего порядка, хотя и нескольких. Рассмотрим это преобразование в общем виде.

Пусть дан определитель n-го порядка

$$\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}.$$

Выделим в нем любую строку, например вторую. Так как каждое произведение содержит один элемент из выделенной второй строки, то все произведения, входящие в определитель, можно разбить на группы в зависимости от того, какой элемент второй строки они содержат.

Соберем, например, произведения, содержащие элемент a_{21}, и вынесем этот элемент за скобку. Внутри скобки получится группа произведений, каждое из которых содержит $n-1$ множитель. Все эти множители не могут принадлежать второй строке, а также первому столбцу.

Назовем выражение, стоящее в скобках, *алгебраическим дополнением элемента* a_{21} и обозначим его через A_{21}. Тогда произведение $a_{21}A_{21}$ равно сумме всех тех произведений, составляющих определитель, которые содержат a_{21}. Аналогично поступим со всеми элементами выделенной второй строки. Мы придем к равенству

$$D = a_{21}A_{21} + a_{22}A_{22} + \ldots + a_{2n}A_{2n},$$

поскольку каждое произведение, входящее в определитель, содержит один множитель, принадлежащий второй строке, а значит, входит в одно из написанных справа слагаемых.

Для примера обратимся к определителю третьего порядка. Из правой части равенства (5.64) видим, что есть два члена, содержащих элемент a_{21}:

$$a_{13}a_{21}a_{32} - a_{12}a_{21}a_{33} = a_{21}(a_{13}a_{32} - a_{12}a_{33}).$$

Отсюда следует, что $A_{21} = a_{13}a_{32} - a_{12}a_{33}$. Аналогично, для элементов a_{22} и a_{23} находим $A_{22} = a_{11}a_{33} - a_{13}a_{31}$ и $A_{23} = a_{12}a_{31} - a_{11}a_{32}$, а для определителя D

$$D = a_{21}(a_{13}a_{32} - a_{12}a_{33}) + a_{22}(a_{11}a_{33} - a_{13}a_{31}) + a_{23}(a_{12}a_{31} - a_{11}a_{32}) = \\ = a_{21}A_{21} + a_{22}A_{22} + a_{23}A_{23}.$$

Вторая строка была выбрана нами совершенно произвольно и вместо нее можно было бы взять любую другую строку или любой столбец. Если считать, что выделенной является строка с номером i, то получается равенство

$$D = a_{i1}A_{i1} + a_{i2}A_{i2} + \ldots + a_{in}A_{in}. \qquad (9.64)$$

Оно выражает важную теорему: *определитель равен сумме произведений элементов любой своей строки (или столбца) на алгебраические дополнения этих элементов.*

Заметим, что при том определении, которое дано выше алгебраическому дополнению, равенство (9.64) остается практически бесполезным. Им можно воспользоваться лишь в том случае, если мы сумеем находить алгебраические дополнения элементов, не выписывая всех произведений, входящих в определитель, как это описывалось выше. Оказывается, что это нетрудно сделать, если воспользоваться понятием минора элемента.

Пусть в определителе n-го порядка выделен некоторый элемент a_{ik}. *Минором* этого элемента называется определитель $(n-1)$-го порядка, который получается из предыдущего вычеркиванием из него

i-й строки и k-го столбца, т. е. строки и столбца, содержащих выделенный элемент. Минор элемента a_{ik} мы будем обозначать через M_{ik}. Очевидно, что любой минор может быть вычислен отдельно, независимо от всего определителя.

Сравним теперь алгебраическое дополнение и минор элемента a_{11}. Соберем среди всех произведений, образующих в сумме определитель, те из них, которые содержат элемент a_{11}, и вынесем в этой группе произведений a_{11} за скобку. Выражение, стоящее в скобках, и есть по определению, алгебраическое дополнение A_{11}. Рассмотрим его структуру.

Определитель n-го порядка состоит из произведений, содержащих по n множителей. Поскольку один из множителей был вынесен за скобку, то в A_{11} входят произведения, состоящие из $n-1$ множителей. Далее, множители, входящие в одно из произведений, принадлежат разным строкам и разным столбцам (иначе они не могли бы входить в состав одного произведения, образующего определитель) и не могут, кроме того, принадлежать первой строке или первому столбцу.

Таким образом, алгебраическое дополнение A_{11} состоит из произведений по $n-1$ сомножителю в каждом, составленных из элементов строк и столбцов определителя, не принадлежащих первой строке и первому столбцу, т. е. тех именно произведений, которые образуют минор M_{11}. С другой стороны, каждое произведение, входящее в M_{11}, входит также и в A_{11}, потому что после умножения такого произведения на a_{11} мы получим произведение из n элементов, которое должно содержаться в первоначальном определителе.

Итак, минор M_{11} и алгебраическое дополнение A_{11} состоят из одних и тех же произведений. Убедимся теперь в том, что все эти произведения входят как в M_{11}, так и в A_{11} с одинаковыми знаками. Действительно, знак произведения, входящего в минор M_{11}, определяется числом отрезков отрицательного наклона среди всех отрезков, соединяющих сомножители данного произведения. Знак того же произведения в A_{11} определяется его знаком в определителе, т. е. числом отрезков отрицательного наклона в том же произведении, но с присоединением еще элемента a_{11}. Однако из элемента a_{11} не могут выходить отрезки отрицательного наклона, потому что он стоит в левом верхнем углу матрицы. Отсюда следует, что знаки всех произведений в M_{11} и A_{11} совпадают, т. е. имеет место равенство

$$A_{11} = M_{11}. \tag{10.64}$$

Заметим еще раз, что равенство (10.64) устанавливает равенство минора и алгебраического дополнения лишь для элемента a_{11}. Однако, пользуясь им, можно связать между собою алгебраическое дополнение и минор любого элемента. Для этого надо только установить, что происходит с минором и алгебраическим дополнением при переста-

новке двух соседних строк или столбцов. Нетрудно заметить, что при перестановке выделенной строки с любой другой минор не изменяется, ибо порядок строк и столбцов в миноре сохраняется прежним. Алгебраическое дополнение же меняет знак на противоположный, так как при перестановке меняется на противоположный знак определителя.

Пусть теперь нам нужно найти алгебраическое дополнение A_{ik} элемента a_{ik}. Переставим строки и столбцы матрицы так, чтобы элемент a_{ik} попал в левый верхний угол на место a_{11}. Для этого нужно переставлять $i-1$ раз соседние строки и $k-1$ раз — соседние столбцы. Величина минора при этом сохранится, а алгебраическое дополнение изменит знак $i+k-2$ раза, т. е. умножится на $(-1)^{i+k-2}$ или, что то же самое, на $(-1)^{i+k}$. Но для элемента a_{11} справедливо равенство (10.64), из чего мы заключаем, что

$$A_{ik}=(-1)^{i+k}M_{ik}. \qquad (11.64)$$

Итак, *алгебраическое дополнение элемента равно минору этого элемента, если сумма номеров строки и столбца, в которых стоит этот элемент, есть число четное, и равна минору со знаком минус, если эта сумма нечетна.*

Равенство (11.64) позволяет вычислить алгебраические дополнения элементов через их миноры. Благодаря этому равенство (9.64) сводит определитель n-го порядка к сумме определителей $n-1$-го порядка. Если подставить полученные выражения для алгебраических дополнений через миноры, то равенство (9.64) можно записать так:

$$D=(-1)^{i+1}a_{i1}M_{i1}+(-1)^{i+2}a_{i2}M_{i2}+\ldots+(-1)^{i+n}a_{in}M_{in}. \qquad (12.64)$$

Его называют *разложением определителя по минорам i-й строки.* Аналогичное равенство можно написать и для столбца.

Пример 4.64. Вычислить определитель

$$\begin{vmatrix} 1 & 0 & 3 & 0 \\ 2 & -1 & 4 & 1 \\ 0 & 2 & -2 & 3 \\ 3 & -2 & 0 & -1 \end{vmatrix}.$$

Разлагая определитель по минорам первой строки, приведем его к сумме определителей третьего порядка, причем таких определителей будет только два, поскольку два элемента первой строки являются нулями и их алгебраические дополнения вычислять не нужно. Получим

$$\begin{vmatrix} 1 & 0 & 3 & 0 \\ 2 & -1 & 4 & 1 \\ 0 & 2 & -2 & 3 \\ 3 & -2 & 0 & -1 \end{vmatrix} = 1\cdot\begin{vmatrix} -1 & 4 & 1 \\ 2 & -2 & 3 \\ -2 & 0 & -1 \end{vmatrix} + 3\begin{vmatrix} 2 & -1 & 1 \\ 0 & 2 & 3 \\ 3 & -2 & -1 \end{vmatrix} =$$

$$=(-2-24-4+8)+3(-4-9-6+12)=-43.$$

Приведенный определитель выгоднее раскрывать по первой строке, содержащей два нуля, чем по третьей или четвертой, в которых стоит только по одному нулю, так что пришлось бы вычислять три определителя третьего порядка. Еще более выгодно было бы раскрывать определитель по такой строке, в которой лишь один элемент отличен от нуля, так как в этом случае определитель четвертого порядка привелся бы к одному определителю третьего порядка, как в примере 3.64. Здесь такой строки нет, однако этого легко добиться преобразованиями, подобными рассмотренным в примере 3.64. Действительно, прибавляя к третьему столбцу определителя первый столбец, умноженный на -3, получаем

$$\begin{vmatrix} 1 & 0 & 3 & 0 \\ 2 & -1 & 4 & 1 \\ 0 & 2 & -2 & 3 \\ 3 & -2 & 0 & -1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 3+1\cdot(-3) & 0 \\ 2 & -1 & 4+2\cdot(-3) & 1 \\ 0 & 2 & -2+0\cdot(-3) & 3 \\ 3 & -2 & 0+3\cdot(-3) & -1 \end{vmatrix} =$$

$$= \begin{vmatrix} 1 & 0 & 0 & 0 \\ 2 & -1 & -2 & 1 \\ 0 & 2 & -2 & 3 \\ 3 & -2 & -9 & -1 \end{vmatrix} = 1 \begin{vmatrix} -1 & -2 & 1 \\ 2 & -2 & 3 \\ -2 & -9 & -1 \end{vmatrix}.$$

Вместо того, чтобы сразу вычислять полученный определитель третьего порядка по правилу треугольника, удобно его предварительно преобразовать теми же приемами. Например, прибавляя ко второй строке удвоенную первую, а из третьей вычитая удвоенную первую, находим

$$\begin{vmatrix} -1 & -2 & 1 \\ 2 & -2 & 3 \\ -2 & -9 & -1 \end{vmatrix} = \begin{vmatrix} -1 & -2 & 1 \\ 0 & -6 & 5 \\ 0 & -5 & -3 \end{vmatrix} = (-1)\cdot \begin{vmatrix} -6 & 5 \\ -5 & -3 \end{vmatrix} =$$

$$= -(18+25) = -43.$$

Пример 5.64. Вычислить определитель пятого порядка:

$$\begin{vmatrix} 3 & 6 & 5 & 6 & 4 \\ 5 & 9 & 7 & 8 & 6 \\ 6 & 12 & 13 & 9 & 7 \\ 4 & 6 & 6 & 5 & 4 \\ 2 & 5 & 4 & 5 & 3 \end{vmatrix}.$$

Матрица этого определителя совсем не содержит нулей, но их легко получить уже применявшимися преобразованиями. Вычтем, напри-

мер, из первой строки пятую, из второй и четвертой — удвоенную пятую, а из третьей — утроенную пятую. Мы получим

$$\begin{vmatrix} 3 & 6 & 5 & 6 & 4 \\ 5 & 9 & 7 & 8 & 6 \\ 6 & 12 & 13 & 9 & 7 \\ 4 & 6 & 6 & 5 & 4 \\ 2 & 5 & 4 & 5 & 3 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & -2 & 0 \\ 0 & -3 & 1 & -6 & -2 \\ 0 & -4 & -2 & -5 & -2 \\ 2 & 5 & 4 & 5 & 3 \end{vmatrix} =$$

Теперь можно вычесть из первой строки вторую, а из пятой — удвоенную вторую, после чего можно раскрывать определитель по первому столбцу

$$= \begin{vmatrix} 0 & 2 & 2 & 3 & 1 \\ 1 & -1 & -1 & -2 & 0 \\ 0 & -3 & 1 & -6 & -2 \\ 0 & -4 & -2 & -5 & -2 \\ 0 & 7 & 6 & 9 & 3 \end{vmatrix} = (-1)^{1+2} \cdot 1 \cdot \begin{vmatrix} 2 & 2 & 3 & 1 \\ -3 & 1 & -6 & -2 \\ -4 & -2 & -5 & -2 \\ 7 & 6 & 9 & 3 \end{vmatrix} =$$

Вычитаем теперь удвоенный четвертый столбец из первого и второго и утроенный четвертый столбец из третьего, после чего определитель можно снова разлагать по первой строке

$$= - \begin{vmatrix} 0 & 0 & 0 & 1 \\ 1 & 5 & 0 & -2 \\ 0 & 2 & 1 & -2 \\ 1 & 0 & 0 & 3 \end{vmatrix} = -(-1)^{1+4} \cdot 1 \cdot \begin{vmatrix} 1 & 5 & 0 \\ 0 & 2 & 1 \\ 1 & 0 & 0 \end{vmatrix} =$$

Последний определитель можно сразу разлагать по третьей строке

$$= (-1)^{3+1} \cdot \begin{vmatrix} 5 & 0 \\ 2 & 1 \end{vmatrix} = 5.$$

Рассмотренные приемы вычисления определителей хороши и удобны для определителей не очень высокого порядка, элементы которых являются целыми числами. В других случаях эти приемы могут оказаться уже не такими удобными. Ниже, в § 66, будет рассмотрен способ вычисления определителей, более приспособленный для широкого использования.

Для дальнейшего нам потребуются еще два замечания, связанные с формулой (9.64). Пусть дан определитель n-го порядка

$$D=\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix},$$

в котором мы выделили строку с номером i. Как было показано, справедливо равенство $D=a_{i1}A_{i1}+a_{i2}A_{i2}+\dots+a_{in}A_{in}$, где A_{ik} означают алгебраические дополнения соответствующих элементов.

Если $b_1, b_2, \dots, b_n$ — какие-либо n чисел, то замена чисел a_{ik} на b_k в последнем равенстве приводит, очевидно, к определителю, в котором i-я строка заменена числами b_k, а все остальные элементы остаются без изменения:

$$b_1A_{i1}+b_2A_{i2}+\dots+b_nA_{in}=\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ b_1 & b_2 & \dots & b_n \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix}. \qquad (13.64)$$

Посмотрим, в частности, что произойдет, если в качестве чисел b_k в равенстве (13.64) выбрать некоторую другую j-ю строку того же определителя D. Тогда справа мы получим определитель с двумя одинаковыми строками, так как j-я строка будет стоять в нем и на своем месте, и на месте i-й строки, взамен чисел b_k. Такой определитель вследствие свойства 4 равен нулю.

Таким образом, мы приходим к важному равенству

$$a_{j1}A_{i1}+a_{j2}A_{i2}+\dots+a_{jn}A_{in}=0, \qquad (14.64)$$

выражающему следующую теорему: *сумма произведений элементов некоторой строки (или столбца) определителя на алгебраические дополнения элементов другой строки (или столбца) равна нулю.*

Свойства (9.64) и (14.64) алгебраических дополнений позволяют доказать теорему о существовании обратной матрицы, которая упоминалась в § 63.

Теорема. *Если определитель матрицы A отличен от нуля* *), *то обратная матрица A^{-1} существует и единственна.*

*) Такие матрицы называют *невырожденными* или *неособенными*.

Для доказательства этой теоремы достаточно указать вид обратной матрицы и убедиться в том, что она действительно обладает требуемыми свойствами.

Оказывается, что обратной для квадратной матрицы A является матрица из алгебраических дополнений, которая транспонирована и все ее элементы разделены на определитель D матрицы A:

$$A^{-1}=\begin{Vmatrix} \frac{A_{11}}{D} & \frac{A_{21}}{D} & \dots & \frac{A_{n1}}{D} \\ \frac{A_{12}}{D} & \frac{A_{22}}{D} & \dots & \frac{A_{n1}}{D} \\ \dots & \dots & \dots & \dots \\ \frac{A_{1n}}{D} & \frac{A_{2n}}{D} & \dots & \frac{A_{nn}}{D} \end{Vmatrix}. \tag{15.64}$$

Вычислим произведение матриц $A\cdot A^{-1}$, используя правило умножения матриц (12.63). Умножая i-ю строку матрицы A на k-й столбец матрицы A^{-1}, получим

$$c_{ik}=a_{i1}\frac{A_{k1}}{D}+a_{i2}\frac{A_{k2}}{D}+\dots+a_{in}\frac{A_{kn}}{D}=$$

$$=\frac{1}{D}(a_{i1}A_{k1}+a_{i2}A_{k2}+\dots+a_{in}A_{kn}).$$

При $i\neq k$ последнее выражение равно нулю вследствие равенства (14.64), а при $i=k$ оно равно 1 по (9.64). Таким образом,

$$A\cdot A^{-1}=E,$$

т. е. матрица (15.64) действительно является матрицей, обратной A.

Нахождение обратной матрицы по формуле (15.64) может потребовать большого числа действий. Более рациональный для практического применения способ вычисления элементов обратной матрицы будет рассмотрен в § 66.

Перейдем теперь к выводу общего правила для решения системы линейных уравнений. Запишем систему в виде

$$\left.\begin{aligned} a_{11}x_1+a_{12}x_2+\dots+a_{1n}x_n&=b_1,\\ a_{21}x_1+a_{22}x_2+\dots+a_{2n}x_n&=b_2,\\ \dots\dots\dots\dots\dots\dots&\\ a_{n1}x_1+a_{n2}x_2+\dots+a_{nn}x_n&=b_n, \end{aligned}\right\} \tag{16.64}$$

предположив, что определитель системы $D\neq 0$.

Будем решать систему (16.64) путем исключения неизвестных. Для нахождения неизвестного x_k подберем множители таким образом, чтобы исключить сразу все неизвестные, кроме искомого. В качестве таких множителей можно взять алгебраические дополнения элементов k-го столбца. Итак, умножим обе части первого уравнения системы на A_{1k}, второго — на A_{2k} и т. д. и сложим все уравнения. Мы

получим равенство

$$(a_{11}A_{1k}+a_{21}A_{2k}+\ldots+a_{n1}A_{n1})x_1+(a_{12}A_{1k}+a_{22}A_{2k}+\ldots$$
$$\ldots+a_{n2}A_{nk})x_2+\ldots+(a_{1k}A_{1k}+a_{2k}A_{2k}+\ldots+a_{nk}A_{nk})x_k+\ldots$$
$$\ldots+(a_{1n}A_{1k}+a_{2n}A_{2k}+\ldots+a_{nn}A_{nk})x_n=$$
$$=b_1A_{1k}+b_2A_{2k}+\ldots+b_nA_{nk}. \qquad (17.64)$$

Легко заметить, что коэффициенты при каждом неизвестном x_l в равенстве (17.64) равны сумме произведений элементов l-го столбца на алгебраические дополнения элементов k-го столбца. Вследствие (14.64), при $l\neq k$ эти коэффициенты равны нулю, т. е. все неизвестные действительно исключаются. Что касается коэффициента при x_k, то он равен D, как это видно из равенства (9.64).

Таким образом, левая часть равенства (17.64) имеет вид Dx_k. Обозначив правую часть через D_k, перепишем равенство (17.64) в виде

$$D\cdot x_k=D_k,$$

откуда в силу предположения $D\neq 0$ находим

$$x_k=\frac{D_k}{D} \qquad (k=1,\ 2,\ \ldots,\ ). \qquad (18.64)$$

Формула (18.64) дает возможность вычислить значения всех неизвестных из системы (16.64). Остается только отметить, что числитель дроби равен

$$D_k=b_1A_{1k}+b_2A_{2k}+\ldots+b_nA_{nk}.$$

Воспользовавшись равенством (13.64), видим, что D_k есть определитель n-го порядка, полученный из определителя системы, в котором k-й столбец заменен столбцом свободных членов

$$D_k=\begin{vmatrix} a_{11} & a_{12} & \ldots & b_1 & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & b_2 & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & b_n & \ldots & a_{nn} \end{vmatrix}. \qquad (19.64)$$

Правило, выраженное формулой (18.64), можно сформулировать следующим образом: *если определитель системы линейных уравнений отличен от нуля, то каждое неизвестное этой системы равно дроби, знаменателем которой является определитель системы, а числитель получается из определителя системы заменой столбца коэффициентов при определяемом неизвестном столбцом свободных членов.* Это правило называют *правилом Крамера.* Его частным случаем при $n=2$ и является правило, приведенное в начале настоящего параграфа.

Пример 6.64. Решить систему уравнений

$$\begin{aligned} 2x_1 + 3x_2 + 11x_3 + 5x_4 &= 2, \\ x_1 + x_2 + 5x_3 + 2x_4 &= 1, \\ 2x_1 + x_2 + 3x_3 + 2x_4 &= -3, \\ x_1 + x_2 + 3x_3 + 4x_4 &= -3. \end{aligned}$$

Вычислим прежде всего определитель системы:

$$D = \begin{vmatrix} 2 & 3 & 11 & 5 \\ 1 & 1 & 5 & 2 \\ 2 & 1 & 3 & 2 \\ 1 & 1 & 3 & 4 \end{vmatrix}.$$

Вычитая четвертую строку из второй и удвоенную четвертую из первой и третьей, находим

$$D = \begin{vmatrix} 0 & 1 & 5 & -3 \\ 0 & 0 & 2 & -2 \\ 0 & -1 & -3 & -6 \\ 1 & 1 & 3 & 4 \end{vmatrix} = -\begin{vmatrix} 1 & 5 & -3 \\ 0 & 2 & -2 \\ -1 & -3 & -6 \end{vmatrix} =$$

$$= -(-12 + 10 - 6 - 6) = 14.$$

Поскольку определитель системы отличен от нуля, то к системе применимо правило Крамера. Чтобы им воспользоваться, нужно вычислить еще четыре определителя четвертого порядка, получающиеся заменой столбцов коэффициентов при соответствующих неизвестных столбцом свободных членов

$$D_1 = \begin{vmatrix} 2 & 3 & 11 & 5 \\ 1 & 1 & 5 & 2 \\ -3 & 1 & 3 & 2 \\ -3 & 1 & 3 & 4 \end{vmatrix} = \begin{vmatrix} 11 & 0 & 2 & -1 \\ 4 & 0 & 2 & 0 \\ -3 & 1 & 3 & 2 \\ 0 & 0 & 0 & 2 \end{vmatrix} = -\begin{vmatrix} 11 & 2 & -1 \\ 4 & 2 & 0 \\ 0 & 0 & 2 \end{vmatrix} =$$

$$= -2 \cdot \begin{vmatrix} 11 & 2 \\ 4 & 2 \end{vmatrix} = -28.$$

Здесь мы вначале вычитали третью строку из второй и четвертой и утроенную третью строку из первой, затем разлагали определитель по элементам второго столбца и, наконец, полученный определитель третьего порядка — по элементам его третьей строки. Для

следующего определителя имеем

$$D_2=\begin{vmatrix} 2 & 2 & 11 & 5 \\ 1 & 1 & 5 & 2 \\ 2 & -3 & 3 & 2 \\ 1 & -3 & 3 & 4 \end{vmatrix}=\begin{vmatrix} 2 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 2 & -5 & -7 & -2 \\ 1 & -4 & -2 & 2 \end{vmatrix}=$$

$$=-\begin{vmatrix} 0 & 1 & 1 \\ -5 & -7 & -2 \\ -4 & -2 & 2 \end{vmatrix}=-(8+10-28+10)=0.$$

В этом определителе мы вычитали первый столбец из второго, удвоенный первый столбец из четвертого и умноженный на пять первый столбец — из третьего. После этого мы разлагаем определитель по второй строке, а полученный определитель третьего порядка вычисляем по правилу треугольника

$$D_3=\begin{vmatrix} 2 & 3 & 2 & 5 \\ 1 & 1 & 1 & 2 \\ 2 & 1 & -3 & 2 \\ 1 & 1 & -3 & 4 \end{vmatrix}=\begin{vmatrix} -1 & 0 & 11 & -7 \\ 0 & 0 & 4 & -2 \\ 1 & 0 & 0 & -2 \\ 1 & 1 & -3 & 4 \end{vmatrix}=$$

$$=\begin{vmatrix} -1 & 11 & -7 \\ 0 & 4 & -2 \\ 1 & 0 & -2 \end{vmatrix}=8-22+28=14.$$

Здесь мы вычитали четвертую строку из второй и третьей, а утроенную четвертую — из первой, после чего разлагали определитель по второму столбцу и затем воспользовались правилом треугольника. Наконец, для последнего определителя получаем, как и для D_3,

$$D_4=\begin{vmatrix} 2 & 3 & 11 & 2 \\ 1 & 1 & 5 & 1 \\ 2 & 1 & 3 & -3 \\ 1 & 1 & 3 & -3 \end{vmatrix}=\begin{vmatrix} -1 & 0 & 2 & 11 \\ 0 & 0 & 2 & 4 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 3 & -3 \end{vmatrix}=$$

$$=\begin{vmatrix} -1 & 2 & 11 \\ 0 & 2 & 4 \\ 1 & 0 & 0 \end{vmatrix}=\begin{vmatrix} 2 & 11 \\ 2 & 4 \end{vmatrix}=-14.$$

По правилу Крамера (18.64) находим

$$x_1=-2,\quad x_2=0,\quad x_3=1,\quad x_4=-1.$$

Правило Крамера требует для нахождения значений неизвестных вычисления большого числа определителей и поэтому невыгодно для практического использования. В §§ 65, 67 и 68 будут рассмотрены более удобные практически способы решения систем линейных уравнений.

До сих пор при рассмотрении систем линейных уравнений мы предполагали, что определитель системы отличен от нуля, $D \neq 0$. В этом случае система всегда совместна и имеет единственное решение, которое можно найти либо по правилу Крамера, либо другими приемами, которые описываются в следующих параграфах. Если же $D = 0$, то дело обстоит значительно сложнее.

При $n = 2$ из $D = 0$ вытекает пропорциональность коэффициентов при неизвестных. Действительно, если

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = 0,$$

т. е. $a_{11}a_{22} - a_{12}a_{21} = 0$, то $a_{11}a_{22} = a_{12}a_{21}$, откуда

$$\frac{a_{11}}{a_{21}} = \frac{a_{12}}{a_{22}}.$$

Теперь могут представиться две различные возможности.

1. Свободные члены пропорциональны коэффициентам при неизвестных, т. е.

$$\frac{a_{11}}{a_{21}} = \frac{a_{12}}{a_{22}} = \frac{b_1}{b_2}.$$

Это означает, что одно из уравнений есть следствие другого, так что любая пара значений x, y, удовлетворяющая первому из уравнений, удовлетворяет и всей системе. Так как одно линейное уравнение с двумя неизвестными имеет бесчисленное множество решений, то такая система совместна, но неопределенна.

2. Свободные члены не пропорциональны коэффициентам при неизвестных, т. е.

$$\frac{a_{11}}{a_{21}} = \frac{a_{12}}{a_{22}} \neq \frac{b_1}{b_2}.$$

В этом случае система *несовместна*; решений системы не существует.

Как известно, система двух линейных уравнений с двумя неизвестными допускает простую геометрическую иллюстрацию. Каждое из уравнений первой степени с двумя переменными геометрически изображается прямой линией на плоскости. Задача решения системы есть задача отыскания координат точки пересечения двух прямых.

Предусмотренный правилом Крамера случай $D \neq 0$ соответствует тому, что прямые имеют различный наклон относительно оси абсцисс. Тогда они пересекаются в одной точке. При $D = 0$ прямые параллельны. Если при этом свободные члены пропорциональны коэффи-

циентам, то прямые отсекают равные отрезки на осях, т. е. совпадают, так что все точки прямой являются точками пересечения, а значит, и решениями системы. В противном случае прямые различны и вследствие параллельности общих точек не имеют, так что система несовместна.

При $n > 2$ исследование линейной системы становится уже значительно более громоздким. В этом легко убедиться снова с помощью геометрической иллюстрации, рассмотрев для $n = 3$ всевозможные случаи взаимного расположения трех плоскостей в пространстве. Мы не будем поэтому останавливаться на таком исследовании. Отметим лишь, что, как и для $n = 2$, требование правила Крамера $D \neq 0$ соответствует геометрическому требованию, чтобы никакая пара плоскостей не оказалась параллельной.

§ 65. Решение системы линейных уравнений по способу Гаусса

Способ Гаусса является одним из наиболее распространенных способов решения систем линейных уравнений. Он является *точным*, т. е. если точно выполнить все требуемые в нем действия, то мы получим точное решение системы. Практически, впрочем, точного решения получить не удается, поскольку арифметические действия далеко не всегда могут быть выполнены с полной точностью.

Способ Гаусса может быть реализован в виде различных вычислительных схем, в основе которых лежит одна и та же идея последовательного исключения неизвестных. Мы рассмотрим схему, которая называется *схемой единственного деления*.

Вычислительную схему удобно проиллюстрировать на примере конкретной системы, поэтому мы ограничимся рассмотрением системы четвертого порядка. Ясно, что те же приемы могут быть применены и в любом другом случае.

Рассмотрим систему линейных уравнений четвертого порядка:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15} &= 0,\\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 + a_{25} &= 0,\\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 + a_{35} &= 0,\\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 + a_{45} &= 0, \end{aligned}\right\} \tag{1.65}$$

в которой все члены для удобства дальнейших рассуждений перенесены в одну часть равенства. Примем, что $a_{11} \neq 0$, либо, в противном случае, переставим уравнения так, чтобы это условие было выполнено. Находя x_1 из первого уравнения, получим

$$x_1 = a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}, \tag{2.65}$$

где

$$\alpha_{1i} = -\frac{a_{1i}}{a_{11}} \qquad (i = 2,\ 3,\ 4,\ 5).$$

С помощью уравнения (2.65) можно исключить из оставшихся уравнений x_1, для чего достаточно подставить значение (2.65) для x_1 во второе, третье и четвертое уравнения системы. Тогда мы придем к системе трех уравнений, не содержащих x_1:

$$\left.\begin{aligned} a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + a_{24}^{(1)}x_4 + a_{25}^{(1)} &= 0,\\ a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + a_{34}^{(1)}x_4 + a_{35}^{(1)} &= 0,\\ a_{42}^{(1)}x_2 + a_{43}^{(1)}x_3 + a_{44}^{(1)}x_4 + a_{45}^{(1)} &= 0. \end{aligned}\right\} \qquad (3.65)$$

Из способа получения этой системы видно, что

$$a_{22}^{(1)} = a_{21}\cdot\alpha_{12} + a_{22}$$

и вообще

$$a_{ik}^{(1)} = a_{i1}\alpha_{1k} + a_{ik} \qquad (i = 2,\ 3,\ 4;\quad k = 2,\ 3,\ 4,\ 5).$$

Полученную систему (3.65) можно подвергнуть тому же преобразованию, что и первоначальную. Как и ранее, можно предположить, что $a_{22}^{(1)} \neq 0$, либо переставить уравнения (или неизвестные). Находя теперь x_2 из первого уравнения, получим

$$x_2 = \alpha_{23}x_3 + \alpha_{24}x_4 + \alpha_{25}, \qquad (4.65)$$

где положено

$$\alpha_{2i} = -\frac{a_{2i}^{(1)}}{a_{22}^{(1)}} \qquad (i = 3,\ 4,\ 5).$$

Подставляя выражение (4.65) для x_2 во второе и третье уравнения системы (3.65), получим систему

$$\left.\begin{aligned} a_{33}^{(2)}x_3 + a_{34}^{(2)}x_4 + a_{35}^{(2)} &= 0,\\ a_{43}^{(2)}x_3 + a_{44}^{(2)}x_4 + a_{45}^{(2)} &= 0, \end{aligned}\right\} \qquad (5.65)$$

коэффициенты которой находятся по формулам

$$a_{ik}^{(2)} = a_{i2}^{(1)}\cdot\alpha_{2k} + a_{ik}^{(1)} \qquad (i = 3,\ 4;\ k = 3,\ 4,\ 5).$$

Наконец, из системы (5.65) легко тем же путем перейти к уравнению с одним неизвестным. Сначала мы приходим к уравнению

$$x_3 = \alpha_{34}x_4 + \alpha_{35}, \qquad (6.65)$$

с коэффициентами

$$\alpha_{3i} = -\frac{a_{3i}^{(2)}}{a_{33}^{(2)}} \qquad (i = 4,\ 5).$$

Затем, подставляя найденное значение x_3 во второе уравнение (5.65), получаем

$$a_{44}^{(3)}x_4 + a_{45}^{(3)} = 0,$$

где

$$a_{4k}^{(3)} = a_{43}^{(2)} \cdot a_{3k} + a_{4k}^{(2)} \qquad (k = 4, 5).$$

Последнее уравнение можно переписать в виде

$$x_4 = \alpha_{45}, \tag{7.65}$$

положив

$$\alpha_{45} = -\frac{a_{45}^{(3)}}{a_{44}^{(3)}}.$$

Итак, мы получили четыре уравнения (2.65), (4.65), (6.65) и (7.65), которые можно объединить в систему

$$\left.\begin{aligned} x_1 &= \alpha_{12}x_2 + \alpha_{13}x_3 + \alpha_{14}x_4 + \alpha_{15}, \\ x_2 &= \alpha_{23}x_3 + \alpha_{24}x_4 + \alpha_{25}, \\ x_3 &= \alpha_{34}x_4 + \alpha_{35}, \\ x_4 &= \alpha_{45}. \end{aligned}\right\} \tag{8.65}$$

Из этих уравнений последовательно находятся значения всех четырех неизвестных, x_4, x_3, x_2, x_1.

Процесс нахождения значений неизвестных по способу Гаусса распадается, таким образом, на два этапа. Первый состоит в приведении системы к треугольному виду (8.65). Его принято называть *прямым ходом*. Определение неизвестных по полученным формулам составляет второй этап вычислительного процесса, который называют *обратным ходом*.

Чтобы помочь лучше уяснить суть способа, прежде чем перейти к рассмотрению вычислительной схемы, разберем числовой пример. При этом умышленно берется уже рассматривавшаяся система с целыми коэффициентами, и все вычисления производятся в таком же порядке, как и при теоретических выкладках.

Пример 1.65. Решим систему уравнений

$$\begin{aligned} 2x_1 + 3x_2 + 11x_3 + 5x_4 - 2 &= 0, \\ x_1 + x_2 + 5x_3 + 2x_4 - 1 &= 0, \\ 2x_1 + x_2 + 3x_3 + 2x_4 + 3 &= 0, \\ x_1 + x_2 + 3x_3 + 4x_4 + 3 &= 0. \end{aligned}$$

Эта система имеет вид (1.65). Чтобы привести первое уравнение к виду (2.65), найдем из него x_1, разделив его на $-a_{11} = -2$. Уравнение примет вид

$$x_1 = -1{,}5x_2 - 5{,}5x_3 - 2{,}5x_4 + 1.$$

С помощью этого уравнения надо исключить x_1 из оставшихся трех уравнений. Подставив значение x_1 во второе уравнение, получим

$$(-1{,}5\cdot 1+1)x_2+(-5{,}5\cdot 1+5)x_3+ \\ +(-2{,}5\cdot 1+2)x_4+(1\cdot 1-1)=0,$$

т. е.

$$-0{,}5x_2-0{,}5x_3-0{,}5x_4=0.$$

Подставляя x_1 в третье и четвертое уравнения, находим

$$(-1{,}5\cdot 2+1)x_2+(-5{,}5\cdot 2+3)x_3+ \\ +(-2{,}5\cdot 2+2)x_4+(1\cdot 2+3)=0,$$

$$(-1{,}5\cdot 1+1)x_2+(-5{,}5\cdot 1+3)x_3+ \\ +(-2{,}5\cdot 1+4)x_4+(1\cdot 1+3)=0,$$

т. е. мы приходим к системе трех уравнений вида (3.65)

$$\begin{aligned} -0{,}5x_2-0{,}5x_3-0{,}5x_4 &=0,\\ -2x_2-8x_3-3x_4+5&=0,\\ -0{,}5x_2-2{,}5x_3+1{,}5x_4+4&=0. \end{aligned}$$

С этой системой нужно проделать ту же операцию. Разделим первое уравнение на $-a^{(1)}_{(22)}=0{,}5$. Тогда

$$x_2=-x_3-x_4.$$

Это и есть уравнение (4.65). Умножаем полученное значение x_2 на $a^{(1)}_{32}=-2$ и $a^{(1)}_{42}=-0{,}5$ и подставляем во второе и третье уравнения:

$$((-1)\cdot(-2)+(-8))x_3+((-1)(-2)+ \\ +(-3))x_4+(0\cdot(-2)+5)=0,$$

$$((-1)\cdot(-0{,}5)+(-2{,}5))x_3+((-1)\cdot(-0{,}5)+1{,}5)x_4+ \\ +(0\cdot(-0{,}5)+4)=0$$

или

$$\begin{aligned} -6x_3-x_4+5&=0,\\ -2x_3+2x_4+4&=0. \end{aligned}$$

Мы получили систему (5.65). Разделив первое уравнение на $-a^{(2)}_{33}=6$, получаем уравнение (6.65):

$$x_3=-\frac{1}{6}x_4+\frac{5}{6}.$$

Умножая его на $a^{(2)}_{43}=-2$ и подставляя во второе, приходим к уравнению

$$\left(\left(-\frac{1}{6}\right)\cdot(-2)+2\right)x_4+\left(\frac{5}{6}\cdot(-2)+4\right)=0,$$

$$\frac{7}{3}x_4+\frac{7}{3}=0,$$

откуда делением на $-a_{43}^{(3)} = -\frac{7}{3}$ приходим к уравнению (7.65)

$$x_4 = -1.$$

Итак, в результате вычислений, составляющих *прямой ход*, мы пришли к системе (8.65), имеющей в данном случае вид

$$\begin{aligned} x_1 &= -1{,}5x_2 - 5{,}5x_3 - 2{,}5x_4 + 1, \\ x_2 &= \quad -x_3 - \quad x_4 \quad , \\ x_3 &= \quad -\frac{1}{6}x_4 + \frac{5}{6}, \\ x_4 &= \quad -1, \end{aligned}$$

откуда легко находим (вычисления, составляющие обратный ход, мы опускаем ввиду их простоты) $x_4 = -1$, $x_3 = 1$, $x_2 = 0$, $x_1 = -2$. Подстановка полученных значений в первоначальную систему показывает, что неизвестные найдены правильно.

В рассмотренном примере вычисления проводились так, чтобы проиллюстрировать теоретические выкладки. При практическом решении систем их следует, как всегда, располагать в виде определенной вычислительной схемы. Прямой ход делится на несколько этапов, которые мы условно обозначим через A_1, A_2, A_3, A_4; обратный ход обозначим через B.

В первом этапе, A_1, в таблицу записываются коэффициенты при неизвестных и свободные члены данной системы уравнений. Кроме того, под коэффициентами последнего уравнения записывается строка коэффициентов уравнения (2.65), т. е. величины α_{1k}. Числа в этой строке получаются делением соответствующих элементов первой строки на ее крайний левый элемент, взятый с противоположным знаком. На этом вычисления первого этапа A_1 заканчиваются.

Во втором этапе производятся вычисления коэффициентов $a^{(1)}$ системы (3.65). При этом *к каждому элементу матрицы этапа* A_1 (кроме первой и последней строк) *прибавляется произведение крайнего левого элемента той же строки на крайний нижний элемент того же столбца* (см. табл. 1.65). Вычисления этапа A_2 завершаются получением строки α_{2k} путем деления элементов первой полученной строки на ее крайний левый элемент, взятый с противоположным знаком. Аналогично выполняются и вычисления этапов A_3, A_4.

Этап A_4 завершает прямой ход вычислений, давая в то же время и значение неизвестного x_4. После этого вычисляются значения остальных неизвестных, что составляет этап B вычислений. Значения неизвестных заносятся в первую строку таблицы, отведенную для этапа B и отмеченную буквой x. В столбец свободных членов в этой строке ставится единица.

Неизвестное x_4 получается умножением этой единицы на число α_{45}, стоящее над ним, и записывается в строку x и столбец x_4. После

этого x_3 можно получить, составляя сумму попарных произведений уже найденных чисел строки x и соответствующей части последней строки схемы A_3

$$x_3 = x_4 a_{34} + 1 \cdot a_{35}.$$

Также вычисляются и x_2 и x_1. Вообще, можно воспользоваться следующим правилом: *каждое неизвестное x_k равно скалярному произведению уже вычисленной строки неизвестных на соответствующую часть нижней строки схемы A_k.*

Таблица 1.65

	x_1	x_2	x_3	x_4	Своб. члены	Σ
A_1	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	c_1
	$\boxed{a_{21}}$	$\boxed{a_{22}}$	a_{23}	a_{24}	a_{25}	c_2
	a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	c_3
	a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	c_4
	-1	$\boxed{\alpha_{12}}$	α_{13}	α_{14}	α_{15}	β_1
A_2		$a_{22}^{(1)}$	$a_{23}^{(1)}$	$a_{24}^{(1)}$	$a_{25}^{(1)}$	$c_2^{(1)}$
		$a_{32}^{(1)}$	$a_{33}^{(1)}$	$a_{34}^{(1)}$	$a_{35}^{(1)}$	$c_3^{(1)}$
		$a_{42}^{(1)}$	$a_{43}^{(1)}$	$a_{44}^{(1)}$	$a_{45}^{(1)}$	$c_4^{(1)}$
		-1	α_{23}	α_{24}	α_{25}	β_2
A_3			$a_{33}^{(2)}$	$a_{34}^{(2)}$	$a_{35}^{(2)}$	$c_3^{(2)}$
			$a_{43}^{(2)}$	$a_{44}^{(2)}$	$a_{45}^{(2)}$	$c_4^{(2)}$
			-1	α_{34}	α_{35}	β_3
A_4				$a_{44}^{(3)}$	$a_{45}^{(3)}$	$c_4^{(3)}$
				-1	α_{45}	β_4
B	x_1	x_2	x_3	x_4	1	
	$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	1	

Для способа Гаусса можно получить простые *контрольные соотношения*, которые служат хорошим контролем вычислений. Рассмотрим новую линейную систему с той же матрицей коэффициентов и со свободными членами, равными суммам всех элементов строки

$$c_i = a_{i1} + a_{i2} + a_{i3} + a_{i4} + a_{i5} \quad (i = 1, 2, 3, 4).$$

Пусть числа x_1, x_2, x_3, x_4 образуют решение системы (1.65). Легко проверить, что *системе со свободными членами* c_i *удовлетворяют числа* $x_1 - 1$, $x_2 - 1$, $x_3 - 1$, $x_4 - 1$.

Действительно, подставим эти значения в левую часть уравнения

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15} = 0.$$

Тогда получим

$$\begin{aligned} a_{11}(x_1 - 1) + a_{12}(x_2 - 1) + a_{13}(x_3 - 1) + a_{14}(x_4 - 1) + c_1 = \\ = (a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4) - (a_{11} + a_{12} + a_{13} + a_{14}) + c_1 = \\ = (a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 + a_{15}) - \\ - (a_{11} + a_{12} + a_{13} + a_{14} + a_{15}) + c_1. \end{aligned}$$

Здесь мы прибавили a_{15} в первой и второй скобках. Но теперь ясно, что первая скобка равна нулю, а вторая равна c_1, так что вся сумма обращается в нуль. Произведя аналогичную проверку для каждого из уравнений, убеждаемся в справедливости высказанного утверждения.

Присоединим теперь к вычислениям по способу Гаусса еще один столбец, элементы которого равны суммам элементов соответствующих строк. Выполняя с ним те же вычисления, что и с предыдущим, мы решаем одновременно две системы уравнений со свободными членами a_{i5} и c_i. Тем самым мы получаем хороший заключительный контроль, так как решения второй системы должны быть на единицу меньше решений первой. Но, кроме этого, мы получаем возможность текущего контроля: *в процессе всех вычислений сумма элементов строки* (кроме последнего элемента) *должна быть всегда равна последнему элементу*.

Общая вычислительная схема способа Гаусса изображена в табл. 1.65.

Пример 2.65. Решим по способу Гаусса систему уравнений

$$\begin{aligned} 4{,}11x_1 - 1{,}26x_2 - 5{,}99x_3 + 1{,}29x_4 + 0{,}75 = 0, \\ -1{,}26x_1 + 2{,}00x_2 + 4{,}00x_3 - 1{,}08 = 0, \\ 3{,}18x_1 - 1{,}97x_2 + 0{,}49x_3 - 1{,}00x_4 - 3{,}38 = 0, \\ 1{,}29x_1 + 3{,}81x_2 - 1{,}56x_3 - 0{,}87 = 0. \end{aligned}$$

Вычисления будем выполнять с двумя запасными знаками, т. е. с пятью значащими цифрами, по схеме, описанной выше. Все вычисления произведены в табл. 2.65.

Таблица 2.65

x_1	x_2	x_3	x_4	Своб. члены	Σ
4,11	—1,26	—5,99	1,29	0,75	—1,10
—1,26	2,00	4,00	0	—1,08	3,66
3,18	—1,97	0,49	—1,00	—3,38	—2,68
1,29	3,81	—1,56	0	—0,87	2,67
—1	0,3066	1,4574	—0,3139	—0,1825	0,2676
	1,6137	2,1637	0,3955	—0,8500	3,3229
	—0,9950	5,1245	—1,9982	—3,9604	—1,8291
	4,2055	0,3200	—0,4049	—1,1054	3,0152
	—1	—1,3408	—0,2451	0,5267	—2,0592
		6,4586	—1,7543	—4,4845	0,2198
		—5,3187	—1,4357	1,1096	—5,6448
		—1	0,2716	0,6943	—0,0340
			—2,8803	—2,5832	—5,4635
			—1	—0,8969	—1,8969
0,7995	0,1422	0,4507	—0,8969	1	
—0,2005	—0,8578	—0,5493	—1,8969	1	

§ 66. Применение схемы Гаусса для вычисления определителя и нахождения обратной матрицы

Рассмотренный в предыдущем параграфе способ Гаусса для решения системы линейных уравнений основан, в сущности, на том, что матрица коэффициентов приводится к диагональному виду. Благодаря этому его можно использовать и для вычисления определителей иным путем, отличным от рассмотренного в § 64.

Пусть требуется вычислить определитель

$$D=\begin{vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{vmatrix},$$

причем $a_{11} \neq 0$. Вынесем из первой строки за знак определителя элемент a_{11}, разделив на него все элементы этой строки. Вводя обозначение

$$\alpha_{1j}=\frac{a_{1j}}{a_{11}} \qquad (j=2,\ 3,\ \dots,\ n),$$

запишем определитель в виде

$$D=a_{11}\begin{vmatrix} 1 & \alpha_{12} & \alpha_{13} & \dots & \alpha_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{vmatrix}.$$

Вычитая первую строку, умноженную на надлежащие множители, из следующих строк, можно добиться того, чтобы в первом столбце все элементы, кроме первого, были равны нулю. Величина определителя при этом остается неизменной. Для этого надо вычитать из второй строки первую, умноженную на a_{21}, из третьей — первую, умноженную на a_{31}, ..., из n-й — первую, умноженную на a_{n1}. Получившиеся элементы мы обозначим через $a_{ij}^{(1)}$. Они будут равны:

$$\begin{aligned} a_{22}^{(1)} &= a_{22} - a_{21}\cdot\alpha_{12}, \\ a_{23}^{(1)} &= a_{23} - a_{21}\cdot\alpha_{13}, \\ &\dots\dots\dots\dots \\ a_{2n}^{(1)} &= a_{2n} - a_{21}\alpha_{1n}, \\ a_{32}^{(1)} &= a_{32} - a_{31}\alpha_{12}, \\ &\dots\dots\dots\dots \end{aligned}$$

и вообще

$$a_{ij}^{(1)} = a_{ij} - a_{i1}\alpha_{1j}, \tag{1.66}$$

а определитель примет вид

$$D=a_{11}\begin{vmatrix} 1 & \alpha_{12} & \dots & \alpha_{1n} \\ 0 & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \dots & \dots & \dots & \dots \\ 0 & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{vmatrix}.$$

Разлагая этот определитель по элементам первого столбца, получим

$$D = a_{11}\begin{vmatrix} a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \cdot & \cdot & \cdot \\ a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{vmatrix},$$

причем определитель будет уже $(n-1)$-го порядка. С этим определителем проделаем ту же операцию, что и с исходным, вынося множитель $a_{22}^{(1)}$, который также следует предполагать отличным от нуля. Тогда

$$D = a_{11}a_{22}^{(1)}\begin{vmatrix} 1 & a_{23} & \dots & a_{2n} \\ a_{32}^{(1)} & a_{33}^{(1)} & \dots & a_{3n}^{(1)} \\ \cdot & \cdot & \cdot & \cdot \\ a_{n2}^{(1)} & a_{n3}^{(1)} & \dots & a_{nn}^{(1)} \end{vmatrix},$$

где

$$a_{2j} = \frac{a_{2j}^{(1)}}{a_{22}^{(1)}} \qquad (j = 3,\ 4,\ \dots,\ n).$$

Умножая первую строку последовательно на $a_{32}^{(1)}, \dots, a_{n2}^{(1)}$ и вычитая получающиеся строки соответственно из второй, третьей, ..., строк, получим

$$D = a_{11}a_{22}^{(1)}\begin{vmatrix} 1 & a_{23} & \dots & a_{2n} \\ 0 & a_{33}^{(2)} & \dots & a_{3n}^{(2)} \\ \cdot & \cdot & \cdot & \cdot \\ 0 & a_{n3}^{(2)} & \dots & a_{nn}^{(2)} \end{vmatrix} = a_{11}\cdot a_{22}^{(1)}\begin{vmatrix} a_{33}^{(2)} & \dots & a_{3n}^{(2)} \\ \cdot & \cdot & \cdot \\ a_{n3}^{(2)} & \dots & a_{nn}^{(2)} \end{vmatrix},$$

куда входит уже определитель $(n-2)$-го порядка.

Продолжая такие преобразования, придем к формуле

$$D = a_{11}a_{22}^{(1)}a_{33}^{(2)} \dots a_{n-2,\,n-2}^{(n-3)}\begin{vmatrix} a_{n-1,\,n-1}^{(n-2)} & a_{n-1,\,n}^{(n-2)} \\ a_{n,\,n-1}^{(n-2)} & a_{n,\,n}^{(n-2)} \end{vmatrix}.$$

Вынесем из первой строки элемент $a_{n-1,\,n-1}^{(n-2)}$. Вычитая из второй строки первую, умноженную на $a_{n,\,n-1}^{(n-2)}$, получим окончательно

$$D = a_{11}a_{22}^{(1)} \dots a_{nn}^{(n-1)}. \tag{2.66}$$

Таким образом, *определитель равен произведению ведущих элементов схемы Гаусса.*

При этом все ведущие элементы должны быть отличны от нуля, так как в противном случае деление на них будет невозможно. Этого всегда можно добиться путем перестановки строк или столбцов в матрице, что можно сделать на любом шаге, соблюдая только правило знаков.

Вычислительная схема для нахождения определителя имеет тот же вид, что и для решения системы линейных уравнений, с той только разницей, что в ней отсутствует столбец свободных членов. Контрольные соотношения также остаются прежними. Поэтому мы не будем приводить схему в общем виде, ограничившись рассмотрением примера.

Пример 1.66. Вычислим по схеме Гаусса определитель

$$\begin{vmatrix} 8{,}2 & 1{,}4 & -2{,}3 & 0{,}2 \\ -1{,}6 & 5{,}4 & -7{,}7 & 3{,}1 \\ 0{,}7 & 1{,}9 & -8{,}5 & 4{,}8 \\ 5{,}3 & -5{,}9 & 2{,}7 & -7{,}9 \end{vmatrix}.$$

Вычисления приведены в табл. 1.66. Как и в предыдущем параграфе, вычисления разбиты на этапы. В каждом этапе приводится соответствующая матрица, а затем строка, получающаяся делением первой строки матрицы на ее ведущий элемент. Переход к матрице

Таблица 1.66

8,2	1,4	—2,3	0,2	7,5
—1,6	5,4	—7,7	3,1	—0,8
0,7	1,9	—8,5	4,8	—1,1
5,3	—5,9	2,7	—7,9	—5,8
1	0,1707	—0,2805	0,0244	0,9146
	5,6731	—8,1488	3,1390	0,6633
	1,7805	—8,3036	4,7829	—1,7402
	—6,8047	4,1866	—8,0293	—10,6474
	1	—1,4364	0,5533	0,1169
		—5,7461	3,7977	—1,9484
		—5,5877	—4,2643	—9,8520
		1	—0,6609	0,3391
			—7,9572	—7,9572

следующего этапа происходит по формулам (1.66). Определитель равен произведению ведущих элементов, которые в таблице обведены рамкой. Как показывают вычисления, $D=0{,}212\cdot 10^4$.

Рассмотрим теперь применение этой же схемы Гаусса к вычислению обратной матрицы. Пусть дана невырожденная матрица

$$A=\begin{Vmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{Vmatrix}. \tag{3.66}$$

Вследствие приведенной в § 64 теоремы существует обратная матрица A^{-1}, элементы которой мы обозначим буквами x с соответствующими индексами, т. е.

$$A^{-1}=\begin{Vmatrix} x_{11} & x_{12} & \dots & x_{1k} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2k} & \dots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ x_{n1} & x_{n2} & \dots & x_{nk} & \dots & x_{nn} \end{Vmatrix}. \tag{4.66}$$

Равенство (14.63), служащее определением обратной матрицы, позволяет написать систему линейных уравнений для неизвестных элементов обратной матрицы. Так как матрица имеет n^2 элементов, то у нас есть n^2 неизвестных, и мы получим систему из n^2 уравнений, которую можно решить по способу Гаусса.

Для получения первого столбца матрицы $E=A\cdot A^{-1}$ нужно взять скалярное произведение первой, второй, ... строк матрицы A на первый столбец матрицы A^{-1}, причем в качестве первого элемента столбца должна получиться единица, а все остальные элементы равны нулю. Таким образом, приходим к системе уравнений

$$\left.\begin{aligned} a_{11}x_{11}+a_{12}x_{21}+a_{13}x_{31}+\dots+a_{1n}x_{n1}&=1,\\ a_{21}x_{11}+a_{22}x_{21}+a_{23}x_{31}+\dots+a_{2n}x_{n1}&=0,\\ \dots\dots\dots\dots\dots\dots\dots\dots\dots&\\ a_{n1}x_{11}+a_{n2}x_{21}+a_{n3}x_{31}+\dots+a_{nn}x_{n1}&=0. \end{aligned}\right\} \tag{5.66}$$

Матрицей коэффициентов в системе (5.66) служит заданная матрица A, неизвестными являются элементы первого столбца искомой матрицы A^{-1}. Единицу в первом уравнении можно перенести влево, и тогда система будет иметь вид системы (1.65).

Находя элементы второго столбца матрицы $E=AA^{-1}$, мы придем к новой системе уравнений, аналогичной системе (5.66), в которой неизвестными будут являться элементы в т о р о г о столбца матрицы A^{-1}.

Правые части (или свободные члены после переноса единицы влево) этой системы будут иными (единица переместится во второе уравнение), но матрица коэффициентов останется прежней; ею будет служить матрица A.

Итак, система из n^2 уравнений с n^2 неизвестными для нахождения элементов обратной матрицы распадается на n групп уравнений с n неизвестными в каждой. *Все они имеют ту же матрицу коэффициентов, отличаясь лишь свободными членами.* Поскольку при решении системы по способу Гаусса основные вычисления приходится проводить над матрицей коэффициентов, то решение этих n систем

Таблица 2.66

a_{11}	a_{12}	a_{13}	a_{14}	-1	0	0	0
a_{21}	a_{22}	a_{23}	a_{24}	0	-1	0	0
a_{31}	a_{32}	a_{33}	a_{34}	0	0	-1	0
a_{41}	a_{42}	a_{43}	a_{14}	0	0	0	-1
-1	α_{12}	α_{13}	α_{14}	α_{15}	0	0	0
	$a_{22}^{(1)}$	$a_{23}^{(1)}$	$a_{24}^{(1)}$	$a_{25,\,\mathrm{I}}^{(1)}$	$a_{25,\,\mathrm{II}}^{(1)}$	0	0
	$a_{32}^{(1)}$	$a_{33}^{(1)}$	$a_{34}^{(1)}$	$a_{35,\,\mathrm{I}}^{(1)}$	0	$a_{35,\,\mathrm{III}}^{(1)}$	0
	$a_{42}^{(1)}$	$a_{43}^{(1)}$	$a_{44}^{(1)}$	$a_{45,\,\mathrm{I}}^{(1)}$	0	0	$a_{45,\,\mathrm{IV}}^{(1)}$
	-1	α_{23}	α_{24}	$\alpha_{25,\,\mathrm{I}}$	$\alpha_{25,\,\mathrm{II}}$	0	0
		$a_{33}^{(2)}$	$a_{34}^{(2)}$	$a_{35,\,\mathrm{I}}^{(2)}$	$a_{35,\,\mathrm{II}}^{(2)}$	$a_{35,\,\mathrm{III}}^{(2)}$	0
		$a_{43}^{(2)}$	$a_{44}^{(2)}$	$a_{45,\,\mathrm{I}}^{(2)}$	$a_{45,\,\mathrm{II}}^{(2)}$	0	$a_{45,\,\mathrm{IV}}^{(2)}$
		-1	α_{34}	$\alpha_{35,\,\mathrm{I}}$	$\alpha_{35,\,\mathrm{II}}$	$\alpha_{35,\,\mathrm{III}}$	0
			$a_{44}^{(3)}$	$a_{45,\,\mathrm{I}}^{(3)}$	$a_{45,\,\mathrm{II}}^{(3)}$	$a_{45,\,\mathrm{III}}^{(3)}$	$a_{45,\,\mathrm{IV}}^{(3)}$
			-1	$\alpha_{45,\,\mathrm{I}}$	$\alpha_{45,\,\mathrm{II}}$	$\alpha_{45,\,\mathrm{III}}$	$\alpha_{45,\,\mathrm{IV}}$
x_{11}	x_{21}	x_{31}	x_{41}	1			
x_{12}	x_{22}	x_{32}	x_{42}	1			
x_{13}	x_{23}	x_{33}	x_{43}	1			
x_{14}	x_{24}	x_{34}	x_{44}	1			

можно *объединить в одной схеме, рассматривая одновременно n столбцов свободных членов.* Решив все системы, мы найдем элементы x_{ik} обратной матрицы A^{-1}. Схема расположения вычислений для случая матрицы четвертого порядка приведена в табл. 2.66.

Пример 2.66. Найдем матрицу, обратную для матрицы четвертого порядка:

$$\begin{Vmatrix} 7{,}13 & 8{,}21 & 4{,}47 & -2{,}11 \\ 3{,}25 & 15{,}4 & 2{,}91 & 5{,}43 \\ -6{,}34 & -8{,}17 & -10{,}2 & 3{,}93 \\ 4{,}52 & 6{,}73 & 1{,}37 & -9{,}89 \end{Vmatrix}.$$

Таблица 3.66

7,13	8,21	4,47	—2,11	—1	0	0	0
3,25	15,4	2,91	5,43	0	—1	0	0
—6,34	—8,17	—10,2	3,93	0	0	—1	0
4,52	6,73	1,37	—9,89	0	0	0	—1
—1	—1,15	—0,646	0,296	0,140	0	0	0
	11,66	0,810	6,393	0,455	—1	0	0
	0,89	—6,11	2,060	—0,887	0	—1	0
	1,53	—1,55	—8,55	0,633	0	0	—1
	—1	—0,0697	—0,598	—0,0391	0,0858	0	0
		—6,172	1,528	—0,922	0,0765	—1	0
		—1,657	—9,465	0,573	0,136	0	—1
		—1	0,249	—0,149	0,0125	—0,162	0
			—9,878	0,820	0,115	0,268	—1
			—1	0,083	0,012	0,027	—0,115
0,340	—0,080	—0,128	0,083	1			
0,281	—0,036	—0,147	0,012	1			
0,292	—0,045	—0,142	0,027	1			
0,053	0,114	—0,120	—0,115	1			

Вычисления приведены в табл. 3.66 и никаких пояснений не требуют. Как видно из таблицы, обратная матрица имеет вид

$$A^{-1}=\begin{Vmatrix} 0{,}340 & 0{,}281 & 0{,}292 & 0{,}053 \\ -0{,}080 & -0{,}036 & -0{,}045 & 0{,}114 \\ -0{,}128 & -0{,}147 & -0{,}142 & -0{,}120 \\ 0{,}083 & 0{,}012 & 0{,}027 & -0{,}115 \end{Vmatrix}.$$

§ 67. Итерации для линейных систем

В противоположность способу Гаусса, являющемуся, как уже было сказано, точным, способ итераций дает возможность получить последовательность приближенных значений, сходящуюся к точному решению системы. С вычислительной стороны применение способа итераций часто бывает более выгодным, так как здесь требуется гораздо меньшая точность промежуточных вычислений. Кроме того, способ итераций устойчив относительно просчетов в промежуточных вычислениях: отдельные просчеты на предыдущих шагах итерации не влекут за собой необходимость браковки последующих расчетов.

Рассмотрим систему n линейных уравнений с n неизвестными:

$$\left.\begin{aligned} a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n+a_{1,\,n+1}&=0,\\ a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n+a_{2,\,n+1}&=0,\\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots&\\ a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n+a_{n+1,\,n+1}&=0. \end{aligned}\right\} \tag{1.67}$$

Разрешим первое уравнение относительно x_1, второе — относительно x_2, ..., последнее — относительно x_n. Тогда систему (1.67) можно будет переписать в виде

$$\left.\begin{aligned} x_1&=\quad\alpha_{12}x_2+\alpha_{13}x_3+\ldots+\alpha_{1n}x_n\quad+\alpha_{1,\,n+1},\\ x_2&=\alpha_{21}x_1\quad+\alpha_{23}x_3+\ldots+\alpha_{2n}x_n\quad+\alpha_{2,\,n+1},\\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ x_n&=\alpha_{n1}x_1+\alpha_{n2}x_2+\alpha_{n3}x_3+\ldots+\alpha_{n,\,n-1}x_{n-1}+\alpha_{n,\,n+1}, \end{aligned}\right\} \tag{2.67}$$

где

$$\alpha_{ik}=-\frac{a_{ik}}{a_{ii}}\qquad(i=1,\,2,\,\ldots,\,n;\ k=1,\,2,\,\ldots,\,n,\,n+1). \tag{3.67}$$

Правые части системы (2.67) являются функциями переменных x_1, x_2, ..., x_n. Обозначим правую часть первого из уравнений через $L_1(x_1, x_2, \ldots, x_n)$. При этом мы не обращаем внимания на то обстоятельство, что фактически правая часть первого уравнения не зависит от x_1. Ничего не изменилось бы, если бы правая часть содержала еще и слагаемое вида $\alpha_{11}x_1$. Аналогично, правую часть второго уравнения обозначим через $L_2(x_1, x_2, \ldots, x_n)$ и т. д. Систему (2.67) можно тогда переписать в виде

$$\left.\begin{aligned} x_1&=L_1(x_1,\,x_2,\,\ldots,\,x_n),\\ x_2&=L_2(x_1,\,x_2,\,\ldots,\,x_n),\\ &\ldots\ldots\ldots\ldots\\ x_n&=L_n(x_1,\,x_2,\,\ldots,\,x_n). \end{aligned}\right\} \tag{4.67}$$

Зададим какие-либо начальные значения неизвестных $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$. Подставляя эти значения в правые части системы (4.67), получим первые приближения:

$$x_1^{(1)} = L_1(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}),$$
$$x_2^{(1)} = L_2(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}),$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x_n^{(1)} = L_n(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}).$$

Полученные первые приближения могут быть таким же образом использованы для получения вторых, третьих и т. д., так что для любого ν можно получить ν-е приближение $x_1^{(\nu)}, x_2^{(\nu)}, \ldots, x_n^{(\nu)}$.

Естественно возникает вопрос об условиях, выполнение которых обеспечит сходимость получающихся приближений к истинному решению системы $x_1, x_2, \ldots, x_n$. Установим эти условия.

Обозначим через $\varepsilon_k^{(\nu)}$ ошибку ν-го приближения для неизвестного x_k

$$\varepsilon_k^{(\nu)} = x_k^{(\nu)} - x_k,$$

где x_k означает точное значение неизвестного в решении системы. Так как x_k удовлетворяет системе (2.67), то из соответствующего уравнения этой системы имеем

$$x_k = \alpha_{k1}x_1 + \alpha_{k2}x_2 + \ldots + \alpha_{kn}x_n + \alpha_{k,\,n+1}.$$

С другой стороны, по определению $(\nu+1)$-го приближения,

$$x_k^{(\nu+1)} = \alpha_{k1}x_1^{(\nu)} + \alpha_{k2}x_2^{(\nu)} + \ldots + \alpha_{kn}x_n^{(\nu)} + \alpha_{k,\,n+1}.$$

Вычитая первое уравнение из второго, находим

$$\varepsilon_k^{(\nu+1)} = \alpha_{k1}\varepsilon_1^{(\nu)} + \alpha_{k2}\varepsilon_2^{(\nu)} + \ldots + \alpha_{kn}\varepsilon_n^{(\nu)}. \qquad (5.67)$$

Оценим абсолютную величину ошибки $\varepsilon_k^{(\nu+1)}$. Учитывая, что абсолютная величина произведения равна произведению абсолютных величин сомножителей, а абсолютная величина суммы не превосходит суммы абсолютных величин слагаемых, получим

$$|\varepsilon_k^{(\nu+1)}| \leqslant |\alpha_{k1}| \cdot |\varepsilon_1^{(\nu)}| + |\alpha_{k2}| \cdot |\varepsilon_2^{(\nu)}| + \ldots + |\alpha_{kn}| \cdot |\varepsilon_n^{(\nu)}|. \qquad (6.67)$$

Неравенства типа (6.67) можно, очевидно, написать для всех $k = 1, 2, \ldots, n$. Представим себе, что все эти неравенства выписаны, сложим их все и сгруппируем члены справа по столбцам. Мы получим

$$|\varepsilon_1^{(\nu+1)}| + |\varepsilon_2^{(\nu+1)}| + \ldots + |\varepsilon_n^{(\nu+1)}| \leqslant (|\alpha_{11}| + |\alpha_{21}| + \ldots + |\alpha_{n1}|) \cdot |\varepsilon_1^{(\nu)}| +$$
$$+ (|\alpha_{12}| + |\alpha_{22}| + \ldots + |\alpha_{n2}|)\,|\varepsilon_2^{(\nu)}| + \ldots + (|\alpha_{1n}| + |\alpha_{2n}| + \ldots +$$
$$+ |\alpha_{nn}|) \cdot |\varepsilon_n^{(\nu)}|. \qquad (7.67)$$

Так как число всех коэффициентов системы конечно и число различных сумм модулей равно n, то заведомо найдется такое C, что при любом k будет выполнено неравенство

$$|\alpha_{1k}| + |\alpha_{2k}| + \ldots + |\alpha_{nk}| < C.$$

После этого неравенство (7.67) можно переписать в виде

$$|\varepsilon_1^{(\nu+1)}| + |\varepsilon_2^{(\nu+1)}| + \ldots + |\varepsilon_n^{(\nu+1)}| \leqslant C(|\varepsilon_1^{(\nu)}| + |\varepsilon_2^{(\nu)}| + \ldots + |\varepsilon_n^{(\nu)}|).$$

Если положить $|\varepsilon_1^{(\nu)}|+|\varepsilon_2^{(\nu)}|+\ldots+|\varepsilon_n^{(\nu)}|=\sigma_\nu$, то последнее неравенство примет вид

$$\sigma_{\nu+1}\leqslant C\sigma_\nu. \tag{8.67}$$

Неравенство (8.67) справедливо при любых $\nu=0, 1, 2, \ldots$ Поэтому можно написать

$$\begin{aligned}
&\sigma_1\leqslant C\sigma_0,\\
&\sigma_2\leqslant C\sigma_1\leqslant C^2\sigma_0,\\
&\sigma_3\leqslant C\sigma_2\leqslant C^3\sigma_0,\\
&\ldots\ldots\ldots\\
&\sigma_\nu\leqslant C^\nu\sigma_0,\\
&\ldots\ldots\ldots
\end{aligned}$$

Если $C<1$, то, как известно, $\lim\limits_{\nu\to\infty} C^\nu=0$, откуда следует, что и

$$\lim_{\nu\to\infty}\sigma_\nu=0.$$

Но в σ_ν все слагаемые положительны. Поэтому для любого k справедливо неравенство $|\varepsilon_k^{(\nu)}|<\sigma_\nu$. Отсюда вытекает, что при любом k

$$\lim_{\nu\to\infty}\varepsilon_k^{(\nu)}=0$$

или

$$\lim_{\nu\to\infty}x_k^{(\nu)}=x_k.$$

Таким образом, выполнение условия $C<1$ достаточно для того, чтобы наш итерационный процесс сходился к точному решению системы. Чтобы выяснить, что означает условие $C<1$ для коэффициентов уравнения, заметим, что, по определению, C есть верхняя грань сумм коэффициентов системы (2.67) по столбцам, которые в свою очередь являются отношениями элементов матрицы коэффициентов системы (1.67) к диагональным элементам. Таким образом, *все коэффициенты системы должны быть малы по сравнению с диагональными.*

Это условие можно сформулировать и более точно: *для того чтобы итерационный процесс сходился, достаточно, чтобы в любом столбце сумма отношений коэффициентов системы к диагональным коэффициентам, взятым из той же строки, была строго меньше единицы.*

То же рассуждение, которое использовалось для установления достаточных условий сходимости итерационного процесса, можно применить для получения оценки погрешности очередного приближения. Обозначим разность между двумя последовательными значениями k-го неизвестного через

$$\delta_k^{(\nu)}=x_k^{(\nu+1)}-x_k^{(\nu)}.$$

Прибавляя и вычитая в правой части точное значение неизвестного x_k, найдем

$$\delta_k^{(\nu)}=(x_k^{(\nu+1)}-x_k)-(x_k^{(\nu)}-x_k)=\varepsilon_k^{(\nu+1)}-\varepsilon_k^{(\nu)}.$$

Так как абсолютная величина разности не меньше разности абсолютных величин уменьшаемого и вычитаемого, то

$$|\delta_k^{(\nu)}|\geqslant|\varepsilon_k^{(\nu+1)}|-|\varepsilon_k^{(\nu)}|,$$

а также

$$|\delta_k^{(\nu)}| \geqslant |\varepsilon_k^{(\nu)}| - |\varepsilon_k^{(\nu+1)}|. \tag{9.67}$$

Полагая в неравенстве (9.67) $k=1, 2, \ldots, n$ и складывая затем все полученные неравенства, находим

$$|\delta_1^{(\nu)}| + |\delta_2^{(\nu)}| + \ldots + |\delta_n^{(\nu)}| \geqslant (|\varepsilon_1^{(\nu)}| + |\varepsilon_2^{(\nu)}| + \ldots + |\varepsilon_n^{(\nu)}|) - (|\varepsilon_1^{(\nu+1)}| + |\varepsilon_2^{(\nu+1)}| + \ldots + |\varepsilon_n^{(\nu+1)}|).$$

Обозначив $|\delta_1^{(\nu)}| + |\delta_2^{(\nu)}| + \ldots + |\delta_n^{(\nu)}| = \Delta_\nu$ и используя введенное выше обозначение для σ_ν, перепишем последнее неравенство в виде

$$\Delta_\nu \geqslant \sigma_\nu - \sigma_{\nu+1}.$$

Заменим в неравенстве (10.67) вычитаемое $\sigma_{\nu+1}$ на $C\sigma_\nu$. От этого в силу (8.67) наше неравенство только усилится. Но тогда

$$\Delta_\nu \geqslant \sigma_\nu (1-C), \tag{10.67}$$

откуда

$$\sigma_\nu \leqslant \frac{\Delta_\nu}{1-C}.$$

Как мы уже замечали, из определения σ_ν следует, что $|\varepsilon_k^{(\nu)}| < \sigma_\nu$ для всех $k=1, 2, \ldots, n$. Поэтому

$$|\varepsilon_k^{(\nu)}| \leqslant \frac{\Delta_\nu}{1-C}. \tag{11.67}$$

Неравенство (11.67) и дает оценку погрешности ν-й итерации через разности двух последовательных итераций.

Таким образом, оценка погрешностей $\varepsilon_k^{(\nu)}$ значений неизвестных на ν-м шаге итерации может быть получена так: сначала следует найти суммы модулей коэффициентов столбцов матрицы системы (2.67). Наибольшее из этих чисел принимаем за величину C. После этого находим величины $\delta_k^{(\nu)}$, являющиеся разностями значений неизвестных на ν-м и $(\nu+1)$-м шагах:

$$\delta_k^{(\nu)} = x_k^{(\nu+1)} - x_k^{(\nu)} \qquad (k=1, 2, \ldots, n),$$

и сумму модулей этих величин

$$\Delta_\nu = |\delta_1^{(\nu)}| + |\delta_2^{(\nu)}| + \ldots + |\delta_n^{(\nu)}|.$$

После этого можно получить оценку погрешностей неизвестных, воспользовавшись неравенством (11.67).

Перейдем теперь к рассмотрению вычислительной схемы для решения линейной системы методом итераций. Для простоты и определенности ограничимся системой трех уравнений с тремя неизвестными, записав ее сразу в виде

$$\begin{aligned} x_1 &= \alpha_{12}x_2 + \alpha_{13}x_3 + \alpha_{14}, \\ x_2 &= \alpha_{21}x_1 + \alpha_{23}x_3 + \alpha_{24}, \\ x_3 &= \alpha_{31}x_1 + \alpha_{32}x_2 + \alpha_{34}. \end{aligned} \tag{12.67}$$

За нулевые приближения значений неизвестных можно принять свободные члены соответствующих уравнений

$$x_1^{(0)} = \alpha_{14}, \quad x_2^{(0)} = \alpha_{24}, \quad x_3^{(0)} = \alpha_{34}. \tag{13.67}$$

В процессе вычислений удобно находить не непосредственно значения неизвестных, а поправки к предыдущим приближениям. Начнем с вычисления первых поправок. Пусть

$$\delta_1^{(0)} = x_1^{(1)} - x_1^{(0)}, \quad \delta_2^{(0)} = x_2^{(1)} - x_2^{(0)}, \quad \delta_3^{(0)} = x_3^{(1)} - x_3^{(0)}. \tag{14.67}$$

При этом значения $x_1^{(0)}$, $x_2^{(0)}$, $x_3^{(0)}$ можно считать заданными равенствами (13.67), а первые приближения $x_1^{(1)}$, $x_2^{(1)}$, $x_3^{(1)}$ находятся по формулам

$$\begin{aligned} x_1^{(1)} &= \quad\quad\quad a_{12}x_2^{(0)} + a_{13}x_3^{(0)} + a_{14}, \\ x_2^{(1)} &= a_{21}x_1^{(0)} \quad\quad\quad + a_{23}x_3^{(0)} + a_{24}, \\ x_3^{(1)} &= a_{31}x_1^{(0)} + a_{32}x_2^{(0)} \quad\quad\quad + a_{34}. \end{aligned}$$

Вычитая из этих равенств равенства (13.67), находим

$$\left.\begin{aligned} \delta_1^{(0)} &= a_{12}x_2^{(0)} + a_{13}x_3^{(0)}, \\ \delta_2^{(0)} &= a_{21}x_1^{(0)} + a_{23}x_3^{(0)}, \\ \delta_3^{(0)} &= a_{31}x_1^{(0)} + a_{32}x_2^{(0)}. \end{aligned}\right\} \tag{15.67}$$

Равенства (14.67) можно переписать в виде

$$x_1^{(1)} = x_1^{(0)} + \delta_1^{(0)}, \; x_2^{(1)} = x_2^{(0)} + \delta_2^{(0)}, \; x_3^{(1)} = x_3^{(0)} + \delta_3^{(0)},$$

где $\delta_i^{(0)}$ $(i = 1, 2, 3)$ уже определены равенствами (15.67). Точно таким же путем можно получить и поправки для следующих шагов. Действительно, по определению,

$$\delta_1^{(\nu)} = x_1^{(\nu+1)} - x^{(\nu)},$$

откуда

$$x_1^{(\nu+1)} = x_1^{(\nu)} + \delta_1^{(\nu)}.$$

Вместе с тем, по построению,

$$\begin{aligned} x_1^{(\nu+1)} &= a_{12}x_2^{(\nu)} \quad\quad + a_{13}x_3^{(\nu)} + a_{14}, \\ x_1^{(\nu)} &= a_{12}x_2^{(\nu-1)} + a_{13}x_3^{(\nu-1)} + a_{14}. \end{aligned}$$

Вычитая из первого уравнения второе, находим

$$x_1^{(\nu+1)} - x_1^{(\nu)} = a_{12}(x_2^{(\nu)} - x_2^{(\nu-1)}) + a_{13}(x_3^{(\nu)} - x_3^{(\nu-1)}),$$

т. е.

$$\delta_1^{(\nu)} = a_{12}\delta_2^{(\nu-1)} + a_{13}\delta_3^{(\nu-1)}, \tag{16.67}$$

и аналогично для других неизвестных

$$\left.\begin{aligned} \delta_2^{(\nu)} &= a_{21}\delta_1^{(\nu)} \quad\quad + a_{23}\delta_3^{(\nu-1)}, \\ \delta_3^{(\nu)} &= a_{31}\delta_1^{(\nu-1)} + a_{32}\delta_2^{(\nu-1)}. \end{aligned}\right\} \tag{17.67}$$

Итак, мы установили формулы, позволяющие вычислять последующие поправки через предыдущие. Для определения искомых значений неизвестных можно воспользоваться тождеством

$$x_1 = x_1 + x_1^{(0)} + (x_1^{(1)} - x_1^{(0)}) + (x_1^{(2)} - x_1^{(1)}) + \ldots + (x_1^{(\nu)} - x^{(\nu-1)}) - x_1^{(\nu)},$$

которое можно записать в виде

$$x_1 = x_1^{(0)} + \delta_1^{(0)} + \delta_1^{(1)} + \ldots + \delta_1^{(\nu)} + \ldots + (x_1 - x_1^{(\nu)}).$$

Как было доказано выше, при $C < 1$ последовательность $\{x_1^{(\nu)}\}$ сходится к x_1, поэтому разность $x_1 - x_1^{(\nu)}$ стремится к нулю. Следовательно, можно

написать

$$x_1 = x_1^{(0)} + \delta_1^{(0)} + \delta_1^{(1)} + \delta_1^{(2)} + \ldots + \delta_1^{(\nu)} + \ldots \qquad (18.67)$$

Для двух других неизвестных точно так же получаем

$$\left.\begin{aligned} x_2 &= x_2^{(0)} + \delta_2^{(0)} + \delta_2^{(1)} + \delta_2^{(2)} + \ldots + \delta_2^{(\nu)} + \ldots, \\ x_3 &= x_3^{(0)} + \delta_3^{(0)} + \delta_3^{(1)} + \delta_3^{(2)} + \ldots + \delta_3^{(\nu)} + \ldots \end{aligned}\right\} \qquad (19.67)$$

Формулы (13.67) — (19.67) являются исходными для построения вычислительной схемы, приведенной в табл. 1.67 для случая системы третьего порядка. Часть A схемы содержит коэффициенты и свободные члены исходной системы (1.67), часть B — коэффициенты системы (2.67), приготовленной к итерации. Для получения строк части B таблицы строки части A делятся каждая на свой диагональный элемент, взятый с противоположным знаком. Поскольку получающиеся по диагонали — 1 не участвуют в дальнейших вычислениях, то их в таблице и не пишут, а на их место ставят прочерк. Часть X таблицы содержит последовательные значения поправок и неизвестных.

Таблица 1.67

A		a_{11}	a_{12}	a_{13}	a_{14}
		a_{21}	a_{22}	a_{23}	a_{24}
		a_{31}	a_{32}	a_{33}	a_{34}
B	I	—	α_{12}	α_{13}	α_{14}
	II	α_{21}	—	α_{23}	α_{24}
	III	α_{31}	α_{32}	—	α_{34}
$x^{(0)}$		$x_1^{(0)}$	$x_2^{(0)}$	$x_3^{(0)}$	X
$\delta^{(0)}$		$\delta_1^{(0)} = x^{(0)} \cdot (\mathrm{I})$	$\delta_2^{(0)} = x^{(0)} \cdot (\mathrm{II})$	$\delta_3^{(0)} = x^{(0)} \cdot (\mathrm{III})$	
$x^{(1)}$		$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	
$\delta^{(1)}$		$\delta_1^{(1)} = \delta^{(0)} \cdot (\mathrm{I})$	$\delta_2^{(1)} = \delta^{(0)} \cdot (\mathrm{II})$	$\delta_3^{(1)} = \delta^{(0)} \cdot (\mathrm{III})$	
$x^{(2)}$		$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	
$\delta^{(2)}$		$\delta_1^{(1)} \cdot (\mathrm{I})$	$\delta_2^{(1)} \cdot (\mathrm{II})$	$\delta_3^{(1)} \cdot (\mathrm{III})$	
...		...	...	...	

Строка нулевого приближения $x^{(0)}$ образуется переписыванием столбца свободных членов приведенной системы. Элементы строки $\delta^{(0)}$ получаются как скалярные произведения строки $x^{(0)}$ и строк коэффициентов

приведенной системы. Запись $x^{(0)} \cdot (\mathrm{I})$ означает

$$x^{(0)} \cdot (\mathrm{I}) = x_1^{(0)} \cdot 0 + x_2^{(0)} \cdot \alpha_{12} + x_3^{(0)} \cdot \alpha_{13}.$$

Аналогично,

$$x^{(0)} \cdot (\mathrm{II}) = x_1^{(0)} \cdot \alpha_{21} + x_2^{(0)} \cdot 0 + x_3^{(0)} \cdot \alpha_{23},$$
$$x^{(0)} \cdot (\mathrm{III}) = x_1^{(0)} \cdot \alpha_{31} + x_2^{(0)} \cdot \alpha_{32} + x_3^{(0)} \cdot 0.$$

Строка $x^{(1)}$ образуется сложением двух предыдущих строк, а строка $\delta^{(1)}$ из строк (I), (II), (III) и $x^{(1)}$ — тем же способом, что и строка $\delta^{(0)}$ из строк (I), (II), (III) и $x^{(0)}$.

Вычисления продолжают обычно до тех пор, пока поправки не станут меньше требуемых погрешностей. Последнюю строку x принимают после этого за окончательные значения неизвестных. Иногда используют также и оценки погрешностей с помощью формулы (11.67).

Пример 1.67. Решим способом итераций систему уравнений

$$\begin{aligned} 4x_1 + 0{,}24x_2 - 0{,}08x_3 - 8 &= 0, \\ 0{,}09x_1 + 3x_2 - 0{,}15x_3 - 9 &= 0, \\ 0{,}04x_1 - 0{,}08x_2 + 4x_3 - 20 &= 0. \end{aligned}$$

Таблица 2.67

	4	0,24	—0,08	—8
	0,09	3	—0,15	—9
	0,04	—0,08	4	—20
I	—	—0,06	0,02	2
II	—0,03	—	0,05	3
III	—0,01	0,02	—	5
$x^{(0)}$	2	3	5	
$\delta^{(0)}$	—0,08	0,19	0,04	
$x^{(1)}$	1,92	3,19	5,04	
$\delta^{(1)}$	—0,0106	0,0044	0,0046	
$x^{(2)}$	1,9094	3,1944	5,0446	
$\delta^{(2)}$	—0,000172	0,000548	0,000194	
$x^{(3)}$	1,909228	3,194948	5,044794	
$\delta^{(3)}$	. . . . .	. . . . .	. . . . .	
$x^{(4)}$	. . . . .	. . . . .	. . . . .	

Вычисления приведены в табл. 2.67. Как видно из таблицы, поправки после трех шагов уже достаточно малы и вычисления на этом можно закончить.

Оценим погрешность полученных решений. Суммы модулей коэффициентов рассматриваемой системы по столбцам равны 0,04, 0,08 и 0,07. Поэтому можно принять $C = 0{,}08$. Далее, для Δ_ν находим при $\nu = 3$ значение $\Delta_\nu = 0{,}00055$. Поэтому формула (11.67) дает

$$\varepsilon^{(3)} = 0{,}0006.$$

Заметим, что в способе итераций не рассматриваются контрольные соотношения. Дело в том, что этот способ, как принято говорить, устойчив относительно промежуточных просчетов, т. е. просчет в промежуточных вычислениях не отражается на точности получающегося в конечном счете решения, поскольку такой просчет означает лишь изменение исходных данных для итерации.

§ 68. Способ Зейделя

В предыдущем параграфе при нахождении очередных приближений для значений неизвестных мы пользовались формулами

$$\left.\begin{aligned} x_1^{(\nu)} &= L_1(x_1^{(\nu-1)}, x_2^{(\nu-1)}, \ldots, x_n^{(\nu-1)}),\\ x_2^{(\nu)} &= L_2(x_1^{(\nu-1)}, x_2^{(\nu-1)}, \ldots, x_n^{(\nu-1)}),\\ &\ldots\ldots\ldots\ldots\ldots\\ x_n^{(\nu)} &= L_n(x_1^{(\nu-1)}, x_2^{(\nu-1)}, \ldots, x_n^{(\nu-1)}), \end{aligned}\right\} \qquad (1.68)$$

т. е. следующее приближение целиком определялось по значениям, полученным на предыдущем шаге. Между тем уже при построении первого приближения можно, например, вычисляя значение $x_2^{(1)}$, воспользоваться для x_1 не нулевым приближением $x_1^{(0)}$, а уже полученным первым $x_1^{(1)}$. При вычислении x_3 можно пользоваться уже полученными новыми значениями $x_1^{(1)}$ и $x_2^{(1)}$ и т. д.

Иначе говоря, формулы (1.68), используемые для нахождения следующих значений неизвестных при итерации, предлагается изменить следующим образом:

$$\left.\begin{aligned} x_1^{(\nu)} &= L_1(x_1^{(\nu-1)}, x_2^{(\nu-1)}, \ldots, x_n^{(\nu-1)}),\\ x_2^{(\nu)} &= L_2(x_1^{(\nu)}, x_2^{(\nu-1)}, \ldots, x_n^{(\nu-1)}),\\ x_3^{(\nu)} &= L_3(x_1^{(\nu)}, x_2^{(\nu)}, \ldots, x_n^{(\nu-1)}),\\ &\ldots\ldots\ldots\ldots\ldots\\ x_n^{(\nu)} &= L_n(x_1^{(\nu)}, x_2^{(\nu)}, \ldots, x_{n-1}^{(\nu)}, x_n^{(\nu-1)}). \end{aligned}\right\} \qquad (2.68)$$

Это изменение и составляет идею *итераций по способу Зейделя.*

Следует ожидать, что итерации по способу Зейделя дадут при том же числе шагов более точные результаты, чем обычные итерации, или, что то же самое, такая же точность будет достигнута за меньшее число шагов, поскольку очередные значения неизвестных опре-

деляются здесь более точно. Эти ожидания, как правило, оправдываются, т. е. итерации по способу Зейделя, вообще говоря, действительно дают более точные результаты. Нужно, однако, иметь в виду, что условия сходимости обычного итерационного процесса и процесса Зейделя различны и один из этих процессов для данной системы может оказаться расходящимся тогда, когда другой сходится.

Мы не будем останавливаться на условиях сходимости процесса Зейделя и перейдем сразу к рассмотрению вычислительной схемы. Как и в предыдущем параграфе, рассмотрим эту схему на примере трех линейных уравнений с тремя неизвестными. Формулы для нахождения следующих приближений через предыдущие будут для процесса Зейделя иметь вид

$$\left.\begin{aligned} x_1^{(\nu)} &= \alpha_{12}x_2^{(\nu-1)} + \alpha_{13}x_3^{(\nu-1)} + \alpha_{14},\\ x_2^{(\nu)} &= \alpha_{21}x_1^{(\nu)} + \alpha_{23}x_3^{(\nu-1)} + \alpha_{24},\\ x_3^{(\nu)} &= \alpha_{31}x_1^{(\nu)} + \alpha_{32}x_2^{(\nu)} + \alpha_{34}. \end{aligned}\right\} \tag{3.68}$$

Формулы для вычисления поправок легко получить, исходя непосредственно из определения и используя (3.68). Действительно,

$$\delta_1^{(\nu)} = x_1^{(\nu+1)} - x_1^{(\nu)} = (\alpha_{12}x_2^{(\nu)} + \alpha_{13}x_3^{(\nu)} + \alpha_{14}) - (\alpha_{12}x_2^{(\nu-1)} + \alpha_{13}x_3^{(\nu-1)} + \alpha_{14}) = \alpha_{12}(x_2^{(\nu)} - x_2^{(\nu-1)}) + \alpha_{13}(x_3^{(\nu)} - x_3^{(\nu-1)}),$$

так что

$$\delta_1^{(\nu)} = \alpha_{12}\delta_2^{(\nu-1)} + \alpha_{13}\delta_3^{(\nu-1)}. \tag{4.68}$$

Для $\delta_2^{(\nu)}$ формула выглядит несколько иначе. Так как

$$\delta_2^{(\nu)} = x_2^{(\nu+1)} - x_2^{(\nu)} = (\alpha_{21}x_1^{(\nu+1)} + \alpha_{23}x_3^{(\nu)} + \alpha_{24}) - (\alpha_{21}x_1^{(\nu)} + \alpha_{23}x_3^{(\nu-1)} + \alpha_{24}),$$

то

$$\delta_2^{(\nu)} = \alpha_{21}\delta_1^{(\nu)} + \alpha_{23}\delta_3^{(\nu-1)}. \tag{5.68}$$

Аналогично,

$$\delta_3^{(\nu)} = \alpha_{31}\delta_1^{(\nu)} + \alpha_{32}\delta_2^{(\nu)}. \tag{6.68}$$

Формулы (4.68) — (6.68) справедливы для всех $\nu = 1, 2, \ldots$, но не для $\nu = 0$, как и в обычном итерационном процессе. Если, как мы это делали в предыдущем параграфе, принять за нулевые приближения неизвестных значения соответствующих свободных членов

$$x_1^{(0)} = \alpha_{14},\quad x_2^{(0)} = \alpha_{24},\quad x_3^{(0)} = \alpha_{34},$$

то для первых поправок получим выражения, аналогичные формулам (15.67):

$$\left.\begin{aligned} \delta_1^{(0)} &= \alpha_{12}x_2^{(0)} + \alpha_{13}x_3^{(0)},\\ \delta_2^{(0)} &= \alpha_{21}x_1^{(1)} + \alpha_{23}x_3^{(0)},\\ \delta_3^{(0)} &= \alpha_{31}x_1^{(1)} + \alpha_{32}x_2^{(1)}, \end{aligned}\right\} \tag{7.68}$$

где $x_1^{(1)} = x_1^{(0)} + \delta_1^{(0)}$ и $x_2^{(1)} = x_2^{(0)} + \delta_2^{(0)}$.

Опишем теперь вычислительную схему. Исходные данные и величины α_i располагаются так же, как и в обычном итерационном процессе. Поэтому первая часть вычислений может быть расположена так, как это показано в табл. 1.67, и мы ее здесь не приводим. Таблица для вычисления поправок имеет здесь вид, несколько отличный от табл. 1.67.

Таблица 1.68 показывает расположение вычислений поправок в способе Зейделя. Таблица может содержать любое число групп по четыре строки в каждой. Каждая такая группа предназначена для вычисления очередной поправки. Мы приводим две из них, вычисляя поправки $\delta^{(0)}$ и $\delta^{(1)}$. Следующие поправки вычисляются по той же схеме, что и $\delta^{(1)}$.

Таблица 1.68

	δ_1	δ_2	δ_3
(4)	—	$\underline{\alpha_{21}(\alpha_{14}+\delta_1^{(0)})}$	$\underline{\alpha_{31}(\alpha_{14}+\delta_1^{(0)})}$
(5)	$\alpha_{12}\alpha_{24}$	—	$\underline{\alpha_{32}(\alpha_{24}+\delta_2^{(0)})}$
(6)	$\alpha_{13}\alpha_{34}$	$\alpha_{23}\alpha_{34}$	—
(7) = (4) + (5) + (6)	$\delta_1^{(0)}$	$\delta_2^{(0)}$	$\delta_3^{(0)}$
(8)	—	$\underline{\alpha_{21}\delta_1^{(1)}}$	$\underline{\alpha_{31}\delta_1^{(1)}}$
(9)	$\alpha_{12}\delta_2^{(0)}$	—	$\underline{\alpha_{32}\delta_2^{(1)}}$
(10)	$\alpha_{13}\delta_3^{(0)}$	$\alpha_{23}\delta_3^{(0)}$	—
(11) = (8) + (9) + (10)	$\delta_1^{(1)}$	$\delta_2^{(1)}$	$\delta_3^{(1)}$
12	...	...	...

Изменением по сравнению со схемой обычного итерационного процесса является то, что в вычислениях очередных поправок в ряде случаев используются уже полученные поправки того же этапа вычислений. Соответствующие места в табл. 1.68 подчеркнуты.

Как и при обычном итерационном процессе, вычисления продолжают до получения достаточно малых поправок. Окончательные значения неизвестных получаются по формулам

$$x_1 = \alpha_{14} + \delta_1^{(0)} + \delta_1^{(1)} + \ldots,$$
$$x_2 = \alpha_{24} + \delta_2^{(0)} + \delta_2^{(1)} + \ldots,$$
$$x_3 = \alpha_{34} + \delta_3^{(0)} + \delta_3^{(1)} + \ldots$$

Таблица 2.68

10	1	1	12	
2	10	1	13	
2	2	10	14	
—	—0,1	—0,1	1,2	(1)
—0,2	—	—0,1	1,3	(2)
—0,2	—0,2	—	1,4	(3)
1,2	1,3	1,4		$(x)^{(0)}$

Таблица 3.68

	δ_1	δ_2	δ_3
(4)	—	—0,186	—0,186
(5)	—0,130	—	—0,196
(6)	—0,140	—0,140	—
(7)	—0,270	—0,326	—0,382
(8)	—	—0,0014	—0,0014
(9)	0,0326	—	—0,0074
(10)	0,0382	0,0382	—
(11)	0,0708	0,0368	—0,0088
(12)	—	—0,0006	—0,0006
(13)	—0,0037	—	0,0003
(14)	0,0009	0,0009	
(15)	—0,0028	—0,0015	0,0006
x	0,9980	1,0183	1,0095

Пример 1.68. Решим методом Зейделя систему уравнений

$$\begin{aligned} 10x_1 + x_2 + x_3 &= 12, \\ 2x_1 + 10x_2 + x_3 &= 13, \\ 2x_1 + 2x_2 + 10x_3 &= 14. \end{aligned}$$

Вычисления приведены в табл. 2.68 и 3.68.

§ 69. Способ итераций для нелинейных систем уравнений

Способ итераций, рассмотренный в § 58 для одного уравнения и в § 67 для систем линейных уравнений, может быть с успехом применен для решений систем более общего вида. Рассмотрим применение способа итерации для системы двух уравнений с двумя неизвестными. Такую систему можно записать в виде

$$\begin{aligned} f(x, y) &= 0, \\ \varphi(x, y) &= 0. \end{aligned} \tag{1.69}$$

Предположим, что даны начальные приближения корней x_0, y_0. Перепишем систему в виде

$$\left.\begin{aligned} x &= F(x, y), \\ y &= \Phi(x, y). \end{aligned}\right\} \tag{2.69}$$

Подставив в правые части системы (2.69) вместо x и y значения нулевых приближений x_0, y_0, получим первые приближения корней:

$$\begin{aligned} x_1 &= F(x_0, y_0), \\ y_1 &= \Phi(x_0, y_0). \end{aligned}$$

Аналогично определяются вторые приближения:

$$\begin{aligned} x_2 &= F(x_1, y_1), \\ y_2 &= \Phi(x_1, y_1) \end{aligned}$$

и, вообще,

$$\begin{aligned} x_n &= F(x_{n-1}, y_{n-1}), \\ y_n &= \Phi(x_{n-1}, y_{n-1}). \end{aligned}$$

Легко установить, что если функции $F(x, y)$ и $\Phi(x, y)$ непрерывны и последовательности $x_1, x_2, \ldots, x_n, \ldots$ и $y_1, y_2, \ldots, y_n, \ldots$ сходятся, то пределы их являются корнями системы (2.69).

Сформулируем теперь условия, при которых описанный итерационный процесс является сходящимся.

Теорема. *Пусть $\bar{x}$ и $\bar{y}$ — истинные значения корней системы* (2.69) *и известно, что $a < \bar{x} < b$, $c < \bar{y} < d$; предположим также, что в прямоугольнике, ограниченном прямыми $x = a$, $x = b$, $y = c$ и $y = d$, других кор-*

ней нет. Тогда, если в указанном прямоугольнике выполняются неравенства

$$\left|\frac{\partial F}{\partial x}\right| \leqslant p_1, \quad \left|\frac{\partial F}{\partial y}\right| \leqslant q_1, \quad \left|\frac{\partial \Phi}{\partial x}\right| \leqslant p_2, \quad \left|\frac{\partial \Phi}{\partial y}\right| \leqslant q_2,$$

где $p_1 + p_2 \leqslant M < 1$ *и* $q_1 + q_2 \leqslant M < 1$, *то итерационный процесс сходится, причем в качестве нулевого приближения* (x_0, y_0) *можно брать любую точку прямоугольника* *).

Доказательство этой теоремы проводится так же, как и теоремы о сходимости итерационного процесса для системы линейных уравнений в § 67, но требует применения теоремы Лагранжа для функции нескольких переменных. Поэтому мы его не приводим.

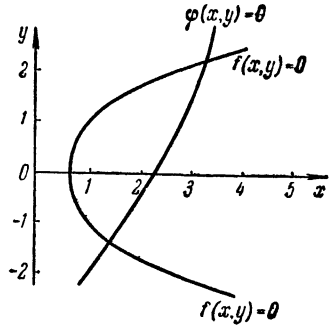


Рис. 58.

Пример 1.69. Решим способом итераций систему

$$f(x, y) = x + 3 \lg x - y^2 = 0,$$
$$\varphi(x, y) = 2x^2 - xy - 5x + 1 = 0.$$

Построим кривые $f(x, y) = x + 3 \lg x - y^2 = 0$ и $\varphi(x, y) = 2x^2 - xy - 5x + 1 = 0$ и определим графически точки их пересечения (рис. 58). Это будут точки (1,4; —1,4) и (3,4; 2,2).

Рассмотрим вторую из этих точек и уточним соответствующие корни по способу итераций.

Прежде всего систему необходимо привести к виду (2.69), что можно сделать различными путями. Если приведем систему к виду

$$x = y^2 - 3 \lg x$$
$$y = 2x - \frac{1}{x} - 5,$$

то $F(x, y) = y^2 - 3 \lg x$, $\Phi(x, y) = 2x - \frac{1}{x} - 5$. Здесь производные

$$\frac{\partial F}{\partial x} = -\frac{3 \lg e}{x}; \quad \frac{\partial F}{\partial y} = 2y; \quad \frac{\partial \Phi}{\partial x} = 2 + \frac{1}{x^2}; \quad \frac{\partial \Phi}{\partial y} = 0.$$

Отсюда видно, что в окрестности точки $x_0 = 3{,}4$; $y_0 = 2{,}2$ будут иметь место неравенства

$$\left|\frac{\partial \Phi}{\partial x}\right| + \left|\frac{\partial F}{\partial x}\right| > 1, \quad \left|\frac{\partial F}{\partial y}\right| > 4.$$

Это показывает, что при таком виде системы итерационный процесс расходится.

*) В условии теоремы следовало бы еще добавить, что ни одно из последующих приближений (x_n, y_n) не выходит за границы рассматриваемого прямоугольника; впрочем, обычно это условие выполняется (см. сноску на стр. 280).

Определим теперь x из второго уравнения, а y из первого:

$$x=\sqrt{\frac{x(5+y)-1}{2}},\quad y=\sqrt{x+3\lg x}.$$

Здесь

$$\frac{\partial F}{\partial x}=\frac{1}{\sqrt{2}}\frac{5+y}{2\sqrt{x(5+y)-1}};\quad \frac{\partial F}{\partial y}=\frac{1}{\sqrt{2}}\frac{x}{2\sqrt{x(5+y)-1}};$$

$$\frac{\partial \Phi}{\partial x}=\frac{1+\frac{3\lg e}{x}}{2\sqrt{x+3\lg x}};\quad \frac{\partial \Phi}{\partial y}=0.$$

За область изоляции корня можно принять прямоугольник $3<x<4$, $2<y<2{,}5$. Легко установить, что в этом прямоугольнике

$$\left|\frac{\partial F}{\partial x}\right|<0{,}60,\quad \left|\frac{\partial F}{\partial y}\right|<0{,}32,\quad \left|\frac{\partial \varphi}{\partial x}\right|<0{,}34.$$

Следовательно, процесс сходится, но так как сумма производных по x сравнительно велика, то скорость сходимости оказывается небольшой.

Вычисления с нулевым приближением $x_0=3{,}4$; $y_0=2{,}2$ дают

$$x_1=\sqrt{\frac{3{,}4\,(2{,}2+5)-1}{2}}=3{,}426,$$

$$y_1=\sqrt{3{,}426+\lg\ 3{,}426}=2{,}243,$$

$$x_2=3{,}451,\qquad y_2=2{,}2505,$$

$$x_3=3{,}466,\qquad y_3=2{,}255,$$

$$x_4=3{,}475,\qquad y_4=2{,}258,$$

$$x_5=3{,}480,\qquad y_5=2{,}259,$$

$$x_6=3{,}483\qquad y_6=2{,}260.$$

§ 70. Программа решения системы линейных уравнений по способу Гаусса

Как уже говорилось в § 65, способ Гаусса может быть представлен в виде различных вычислительных схем. Характерной чертой рассмотренной в § 65 *схемы единственного деления* является разбиение процесса решения задачи на два этапа: прямой и обратный ход. Такое разбиение представляет некоторые неудобства для программирования. Более удобной для программирования на электронных счетных машинах является другая вычислительная схема способа Гаусса, которую называют *схемой Жордана*. В ней значения неизвестных получаются в результате прямого хода, так что обратный ход оказывается ненужным.

Рассмотрим схему Жордана на примере системы четырех уравнений с четырьмя неизвестными, которую мы будем предполагать записанной в виде

$$\left.\begin{aligned}
a_{11}^{(0)}x_1+a_{12}^{(0)}x_2+a_{13}^{(0)}x_3+a_{14}^{(0)}x_4&=b_1^{(0)},\\
a_{21}^{(0)}x_1+a_{22}^{(0)}x_2+a_{23}^{(0)}x_3+a_{24}^{(0)}x_4&=b_2^{(0)},\\
a_{31}^{(0)}x_1+a_{32}^{(0)}x_2+a_{33}^{(0)}x_3+a_{34}^{(0)}x_4&=b_3^{(0)},\\
a_{41}^{(0)}x_1+a_{42}^{(0)}x_2+a_{43}^{(0)}x_3+a_{41}^{(0)}x_4&=b_4^{(0)}.
\end{aligned}\right\}\qquad (1.70)$$

Разделив первое уравнение системы на $a_{11}^{(0)}$ и подставляя затем найденное отсюда значение x_1 в остальные уравнения, получим

$$\left.\begin{aligned} x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + a_{14}^{(1)}x_4 &= b_1^{(1)},\\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + a_{24}^{(1)}x_4 &= b_2^{(1)},\\ a_{32}^{(1)}x_2 + a_{33}^{(1)}x_3 + a_{34}^{(1)}x_4 &= b_3^{(1)},\\ a_{42}^{(1)}x_2 + a_{43}^{(1)}x_3 + a_{44}^{(1)}x_4 &= b_4^{(1)}. \end{aligned}\right\} \tag{2.70}$$

Коэффициенты системы (2.70) получаются по формулам

$$a_{1j}^{(1)} = e_{1j}, \quad b_1^{(1)} = u_1 \qquad (j=2,\ 3,\ 4) \tag{3.70}$$

и

$$a_{ij}^{(1)} = a_{ij}^{(0)} - a_{i1}^{(0)}e_{1j}, \quad b_i^{(1)} = b_i^{(0)} - a_{i1}^{(0)}u_1 \qquad (i,\ j=2,\ 3,\ 4), \tag{4.70}$$

причем

$$e_{1j} = \frac{a_{1j}^{(0)}}{a_{11}^{(0)}}, \qquad u_1 = \frac{b_1^{(0)}}{a_{11}^{(0)}} \qquad (j=2,\ 3,\ 4). \tag{5.70}$$

На следующем шаге разделим второе уравнение системы (2.70) на $a_{22}^{(1)}$, определим x_2 из второго уравнения и подставим его значение во все остальные уравнения системы (в том числе и в первое. В этом и состоит отличие схемы Жордана от рассмотренной ранее схемы, где значение x_2 подставлялось только в два следующие уравнения). Тогда получим

$$\left.\begin{aligned} x_1 \qquad + a_{13}^{(2)}x_3 + a_{14}^{(2)}x_4 &= b_1^{(2)},\\ x_2 + a_{23}^{(2)}x_3 + a_{24}^{(2)}x_4 &= b_2^{(2)},\\ a_{33}^{(2)}x_3 + a_{34}^{(2)}x_4 &= b_3^{(2)},\\ a_{43}^{(2)}x_3 + a_{44}^{(2)}x_4 &= b_4^{(2)}. \end{aligned}\right\} \tag{6.70}$$

Подобно предыдущему,

$$a_{2j}^{(2)} = e_{2j}, \quad b_2^{(2)} = u_2 \qquad (j=3,\ 4), \tag{7.70}$$

$$a_{ij}^{(2)} = a_{ij}^{(1)} - a_{i2}^{(1)}e_{2j}, \quad b_i^{(2)} = b_i^{(1)} - a_{i2}^{(1)}u_2 \qquad (i=1,\ 3,\ 4;\ j=3,\ 4), \tag{8.70}$$

$$e_{2j} = \frac{a_{2j}^{(1)}}{a_{22}^{(1)}}, \qquad u_2 = \frac{b_2^{(1)}}{a_{22}^{(1)}} \qquad (j=3,\ 4). \tag{9.70}$$

Разделив третье уравнение системы (6.70) на $a_{33}^{(2)}$ и исключив затем x_3 снова из всех уравнений системы, найдем

$$\left.\begin{aligned} x_1 \qquad\qquad + a_{14}^{(3)}x_4 &= b_1^{(3)},\\ x_2 \qquad + a_{24}^{(3)}x_4 &= b_2^{(3)},\\ x_3 + a_{34}^{(3)}x_4 &= b_3^{(3)},\\ a_{44}^{(3)}x_4 &= b_4^{(3)}. \end{aligned}\right\} \tag{10.70}$$

Коэффициенты этой системы получаются из коэффициентов предыдущей по формулам, подобным (7.70) — (9.70). Исключая затем x_4 из системы (10.70),

получим, наконец,

$$\left.\begin{aligned} x_1 &= b_1^{(4)},\\ x_2 &= b_2^{(4)},\\ x_3 &= b_3^{(4)},\\ x_4 &= b_4^{(4)}. \end{aligned}\right\} \tag{11.70}$$

Таким образом, исключение неизвестных в процессе прямого хода уже привело к получению их значений. Благодаря отсутствию обратного хода схема Жордана программируется проще, хотя и требует примерно в полтора раза больше арифметических действий, чем схема единственного деления.

Рассмотрим программу схемы Жордана для системы n уравнений с n неизвестными:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ \dots\ + a_{1n}x_n &= b_1,\\ a_{21}x_1 + a_{22}x_2 + \ \dots\ + a_{2n}x_n &= b_2,\\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\\ a_{n1}x_1 + a_{n2}x_2 + \ \dots\ + a_{nn}x_n &= b_n. \end{aligned}\right\} \tag{12.70}$$

Предположим, что матрица коэффициентов системы расположена в идущих подряд ячейках памяти $a_{11}, a_{12}, \dots, a_{1n}, a_{21}, \dots, a_{nn}$, а столбец правых частей — в последовательных ячейках $b_1, b_2, \dots, b_n$.

Коэффициенты преобразованной системы можно помещать на место коэффициентов преобразуемой. Тогда в конце преобразований на месте свободных членов получатся значения неизвестных. Если исходные коэффициенты требуется сохранить, то их следует предварительно перенести на специально отведенное рабочее поле. Удобно воспользоваться для этого библиотечной программой *Перенос* (см. § 45).

Программу для вычисления коэффициентов первого уравнения системы (2.70) по формулам (3.70), (5.70) можно написать так:

T_1	«1» : $a_{11} = R_0$	
	$[PA]$	k
T_2	$a_{11}^* \cdot R_0 = a_{11}^*$	
	$PA \geqslant \overline{2}$	T_2 F^*
U_2	$b_1 \cdot R_0 = b_1$	

Здесь k — слово $(0, n-1, 0)$. Нужные коэффициенты получаются в цикле последовательно, начиная с последнего. Коэффициенты второго уравнения можно вычислить (по формулам (4.70) с $i=2$) с помощью программы

	$[PA]$	k
T_3	$a_{11}^* \cdot a_{21} = R_0$	↑
T_4	$a_{21}^* - R_0 = a_{21}^*$	⌊
	$PA \geqslant \overline{2}$	⌊ F^*
U_3	$b_1 \cdot a_{21} = R_0$	
U_4	$b_2 - R_0 = b_2$.	

Ту же программу можно использовать для получения коэффициентов третьего уравнения, если переадресовать команды T_3, T_4, U_3, U_4, превратив адреса a_{21} в a_{31} и b_2 в b_3:

$$T_3+,\ (0,\ n,\ 0) = T_3,$$
$$T_4+,\ (n,\ 0,\ n) = T_4,$$
$$U_3+,\ (0,\ n,\ 0) = U_3,$$
$$U_4+,\ (1,\ 0,\ 1) = U_4.$$

Чтобы получить коэффициенты всех уравнений системы, нужно приведенный участок программы включить в цикл с переадресацией, в котором наш внутренний цикл будет играть роль рабочей части. Двойной цикл удобно написать следующим образом:

T_1	«1»	:	$a_{11} = R_0$
	$[PA]$		k
T_2	a^*_{11}	·	$R_0 = a^*_{11}$
	$PA \geqslant \bar{2}$		T_2 F^*
U_2	b_1	·	$R_0 = b_1$
	Б $(0,\ n,\ 0)$ Пров. s		s
Нач. s	k	$\nsim$	$s = 0$
	$У_1$	Переадр.	
	$[PA]$		k
T_3	a^*_{11}	·	$a_{11} = R_0$
T_4	a^*_{11}	—	$R_0 = a^*_{11}$
	$PA \geqslant \bar{2}$		F^*
U_3	b_1	·	$a_{11} = R_0$
U_4	b_1	—	$R_0 = b_1$
Переадр.	$T_3+,\ (0,\ n,\ 0) = T_3$		
	$T_4+,\ (n,\ 0,\ n) = T_4$		
	$U_3+,\ (0,\ n,\ 0) = U_3$		
	$U_4+,\ (1,\ 0,\ 1) = U_4$		
Пров. s	s —, $(0,\ 1,\ 0) = s$		
	$У_0$	Нач. s.	

Команды первой группы вычисляют коэффициенты первого преобразованного уравнения. Далее следует двойной цикл. В его начале в счетчике s внешнего цикла образуется значение $(0,\ n-1,\ 0)$. Поэтому для первого уравнения системы, которое не должно уже преобразовываться, рабочая часть цикла обходится, но переадресация работает. Благодаря этому при втором прохождении цикла команды T_3, T_4, U_3, U_4 приобретают уже вид, приведенный в первоначальном варианте программы для преобразования коэффициентов второго уравнения.

Таким образом, приведенная выше программа дает возможность получить из первоначальной системы (1.70) систему (2.70), т. е. провести

первое преобразование. Рассмотрим теперь, чем отличается процесс второго преобразования от уже проведенного.

Прежде всего, исключается второе неизвестное из второго уравнения. Коэффициенты второго уравнения можно получить с помощью первой группы команд написанной программы, если заменить в них адрес a_{11} на a_{22}. Этого можно достигнуть переадресацией

$$T_1+,\quad (0,\ n+1,\ 0)\quad = T_1,$$
$$T_2+,\ (n+1,\ 0,\ n+1) = T_2,$$

после чего эти команды примут вид

$$(T_1)\quad \text{«1»}\colon\ a_{22} = R_0,$$
$$(T_2)\quad a^*_{22}\cdot R_0 = a^*_{22}.$$

Кроме того, для получения свободного члена второго уравнения переадресуем команду U_2

$$U_2+,\ (1,\ 0,\ 1) = U_2,$$

так что она будет выполняться в виде

$$(U_2)\qquad b_2\cdot R_0 = b_2.$$

Далее, в рассмотренных группах команд следует изменять начальное значение PA в цикле от $n-2$ до 1, для чего достаточно предварительно выполнить команду

$$k-,\ (0,\ 1,\ 0) = k.$$

Для правильного преобразования первого уравнения, команды T_3, T_4, U_3, U_4 должны исполняться в виде

$$\begin{array}{l|l}(T_3) & a_{23}\cdot a_{12} = R_0\\ (T_4) & a_{13} - R_0 = a_{13}\\ (U_3) & b_2\cdot a_{12} = R_0\\ (U_4) & b_1 - R_0 = b_1.\end{array}$$

Поэтому их следует дополнительно переадресовать так:

$$T_3+,\ (n+1,\ 0,\ 0) = T_3$$
$$T_4+,\quad (1,\ 0,\ 1)\quad = T_4$$
$$U_3+,\quad (1,\ 1,\ 0)\quad = U_3.$$

Читатель без особого труда сумеет проверить, что после произведенных переадресаций вторая и третья группы команд написанной нами выше программы, составляющие двойной цикл, обеспечат правильное вычисление коэффициентов остальных уравнений. В частности, вследствие того, что $k = (0,\ n-2,\ 0)$, внутренний цикл, составляющий рабочую часть внешнего, обходится на втором шагу цикла, т. е. второе уравнение не преобразуется, что и требуется по алгоритму.

Остается позаботиться о восстановлении переменных команд и о работе наружного цикла $n-1$ раз. Тогда программа решения системы линейных

уравнений по схеме Жордана примет вид

				T_1^0	$=T_1$
				T_2^0	$=T_2$
				T_3^0	$=T_3^{зап}$
				T_4^0	$=T_4^{зап}$
				U_2^0	$=U_2$
				U_3^0	$=U_3^{зап}$
				U_4^0	$=U_4^{зап}$
	Б	(0, *n*, 0)	Пров. *k*		*k*
Нач. *k*				$T_3^{зап}$	$=T_3$
				$T_4^{зап}$	$=T_4$
				$U_3^{зап}$	$=U_3$
				$U_4^{зап}$	$=U_4$
T_1		‹1›	:	a_{11}	$=R_0$) н. п.
	[*PA*]			*k*	
T_2		(a_{11}^*	·	R_0	$=a_{11}^*$) н. п.
	PA	$\geqslant$ 2		T_2	F^*
U_2		(b_1	·	R_0	$=b_1$) н. п.
	Б	(0, *n*, 0)	Пров. *s*		*s*
Нач. *s*		*k*	$\not\sim$	*s*	$=0$
	$У_1$		Переадр.		
	[*PA*]			*k*	
T_3		(a_{11}^*	·	a_{11}	$=R_0$) н. п.
T_4		(a_{11}^*	—	R_0	$=a_{11}^*$) н. п.
	PA	$\geqslant$ 2		T_3	F^*
U_3		(b_1	·	a_{11}	$=R_0$) н. п.
U_4		(b_1	—	R_0	$=b_1$) н. п.
Переадр.		T_3+,	(0, *n*, 0)		$=T_3$
		T_4+,	(*n*, 0, *n*)		$=T_4$
		U_3+,	(0, *n*, 0)		$=U_3$
		U_4+,	(1, 0, 1)		$=U_4$
Пров. *s*		*s*—,	(0, 1, 0)		$=s$
	$У_0$		Нач. *s*		
		T_1 +,	(0, $n+1$, 0)		$=T_1$
Измен.		T_2 +,	($n+1$, 0, $n+1$)		$=T_2$
		U_2 +,	(1, 0, 1)		$=U_2$
		$T_3^{зап}$ +,	($n+1$, 1, 0)		$=T_3^{зап}$

Пров. k		$T_4^{зап}+$,	(1, 0, 1) $= T_4^{зап}$
		$U_3^{зап}+$,	(1, 1, 0) $= U_3^{зап}$
		k —,	(0, 1, 0) $= k$
	$У_0$	Нач. k	

Для работы с этой программой необходимо присоединить к ней машинные слова, представляющие начальное состояние изменяемых команд, а также все требуемые константы.

T_1^0	«1» : $a_{11} = R_0$
T_2^0	a_{11}^* · $R_0 = a_{11}^*$
T_3^0	a_{11}^* · $a_{11} = R_0$
T_4^0	a_{11}^* — $R_0 = a_{11}^*$
U_2^0	b_1 · $R_0 = b_1$
U_3^0	b_1 · $a_{11} = R_0$
U_4^0	b_1 — $R_0 = b_1$
	(0, n, 0)
	(n, 0, n)
	(0, $n+1$, 0)
	($n+1$, 0, $n+1$)
	($n+1$, 1, 0)

Остальные требуемые константы переадресации являются библиотечными.

Можно избежать выписывания начальных состояний команд и констант, оформив нашу программу как стандартную подпрограмму библиотеки, в которой все начальные состояния команд и все константы будут формироваться.

Запишем обращение к подпрограмме решения системы линейных уравнений по способу Гаусса в виде

БВ Я+2 ЛУГ Ω

a_{11} n b_1,

где в строке информации заданы: a_{11} — адрес начала матрицы коэффициентов системы (12.70), b_1 — адрес начала столбца свободных членов, а n — порядок системы, записанный в виде числа единиц второго адреса строки информации. Будем считать, что значения неизвестных получаются в ячейках $b_1, \ldots, b_n$.

Формирование производится с помощью приемов, рассмотренных в § 43. В табл. 1.70 — 3.70 приведена полностью программа *ЛУГ* с формирующей и рабочей частью в содержательных обозначениях и в кодированном виде. В табл. 4.70 дается памятка программы, по которой произведено кодирование.

Таблица 1.70

					1000	A	
ЛУГ	$[PA]$ 0* Ω Ω — 1	1000	4	72	0000	0007	0006
	$F^* \wedge (0, F, 0) = (0, n, 0)$	1	4	55	7777	0132	1105
форм. T	$\overline{30} \rightarrow F^* = R_3$	2	2	54	0050	7777	0013
	$\overline{14} \leftarrow R_3 = R_2$	3	0	54	0114	0013	0012
	$\overline{14} \leftarrow R_2 = R_1$	4	0	54	0114	0012	0011
	$T_1^0 \vee R_2 = T_1$	5	0	75	1072	0012	1035
	$R_1 \vee R_3 = R_0$	6	0	75	0011	0013	0010
	$T_2^0 \vee R_0 = T_2$	7	0	75	1073	0010	1037
	$T_4^0 \vee R_0 = T_4$, зап	1010	0	75	1075	0010	1104
	$R_1 \vee R_2 = R_4$	1	0	75	0011	0012	0014
	$T_3^0 \vee R_4 = T_3$, зап	2	0	75	1074	0014	1101

					1013	A	
форм. U	$F^* \wedge (0, 0, F) = R_3$	1013	4	55	7777	0131	0013
	$\overline{30} \leftarrow R_3 = R_1$	4	0	54	0130	0013	0011
	$R_1 \vee R_3 = R_0$	5	0	75	0011	0013	0010
	$U_2^0 \vee R_0 = U_2$	6	0	75	1076	0010	1041
	$U_4^0 \vee R_0 = U_4$, зап	7	0	75	1100	0010	1104
	$R_1 \vee R_2 = R_0$	1020	0	75	0011	0012	0010
	$U_3^0 \vee R_0 = U_3$, зап	1	0	75	1077	0010	1103
форм. N	$(0, n, 0) +, (0, 1, 0) = (0, n+1, 0)$	2	0	13	1105	0122	1106
	$\overline{14} \leftarrow (0, n+1, 0) = (n+1, 1, 0)$	3	0	54	0114	1106	1107
	$(n+1, 1, 0) \vee (0, 1, 0) = (n+1, 1, 0)$	4	0	75	1107	0122	1107
	$\overline{14} \rightarrow (0, n+1, 0) = R_3$	5	0	54	0064	1106	0013

Таблица 2.70

					1026	A	
	$R_3 \vee (n+1, 0, 0) = (n+1, 0, n+1)$	1026	0	75	0013	1107	1110
	$(n+1, 0, n+1) —, (1, 0, 1) = (n, 0, n)$	7	0	33	1110	0125	1111
	Б (0, n, 0) Пров. $\overrightarrow{k}$ $\overrightarrow{k}$	1030	0	56	1105	1067	1112
Нач. k	$T_{3,\text{зап}} = T_3$	1	0	75	0	1101	1046
	$T_{4,\text{зап}} = T_4$	2	0	75	0	1102	1047
	$U_{3,\text{зап}} = U_3$	3	0	75	0	1103	1051
	$U_{4,\text{зап}} = U_4$	4	0	75	0	1104	1052
T_1	(«1»: $a_{11} = R_0$)	5			н.	п.	
	[PA] k	6	0	72	0	1112	0
T_2	(a^*_{11} · R_0 = a^*_{11})	7			н.	п.	
	$PA \geqslant \overline{2}$ T_2 F^*	1040	1	32	0002	1037	7777

					1041	A	
U_2	(b_1 · R_0 = b_1)	1041			н.	п.	
	Б (0, n, 0) Пров. $\overrightarrow{s}$ $\overrightarrow{s}$	2	0	56	1105	1057	1113
Нач. s	$k \nsim s = 0$	3	0	15	1112	1113	0
	$У_1$ Переадр.	4	0	36	0	1053	0
	[PA] k	5	0	72	0	1112	0
T_3	(a^*_{11} · $a_{11} = R_0$)	6			н.	п.	
T_4	(a^*_{11} — $R_0 = a^*_{11}$)	7			н.	п.	
	$PA \geqslant \overline{2}$ T_3 F^*	1050	1	32	0002	1046	7777
U_3	(b_1 · a_{11} = R_0)	1			н.	п.	
U_4	(b_1 — R_0 = b_1)	2			н.	п.	
Переадр.	$T_3 +, (0, n, 0) = T_3$	3	0	13	1046	1105	1046

Таблица 3.70

					1054	A	
	T_4+, $(n, 0, n)$ $=T_4$	1054	0	13	1047	1111	1047
	U_3+, $(0, n, 0)$ $=U_3$	5	0	13	1051	1105	1051
	U_4+, $(1, 0, 1)$ $=U_4$	6	0	13	1052	0125	1052
Пров. s	$s-$, $(0, 1, 0)$ $=s$	7	0	33	1113	0122	1113
	$У_0$ Нач. s	1060	0	76	0	1043	0
Измен.	T_1+, $(0, n+1, 0)$ $=T_1$	1	0	13	1035	1106	1035
	T_2+, $(n+1, 0, n+1)=T_2$	2	0	13	1037	1110	1037
	U_2+, $(1, 0, 1)$ $=U_2$	3	0	13	1041	0125	1041
	$T_{3,\text{зап}}+$, $(n+1, 1, 0)=T_{3,\text{зап}}$	4	0	13	1101	1107	1101
	$T_{4,\text{зап}}+$, $(1, 0, 1)$ $=T_{4,\text{зап}}$	5	0	13	1102	0125	1102
	$U_{3,\text{зап}}+$, $(1, 1, 0)$ $=U_{3,\text{зап}}$	6	0	13	1101	0126	1101

					1067	A	
Пров. k	$k-$, $(0, 1, 0)=k$	1067	0	33	1112	0122	1112
	$У_0$, Нач. k	1070	0	76	0	1031	0
	$Б$ $Ω-1$	1	0	56	0	0006	0
const T_1^0	«1» : 0 $=R_0$	2	0	05	0101	0	0010
T_2^0	$0^*\cdot R_0$ $=0^*$	3	5	05	0	0010	0
T_3^0	$0^*\cdot 0$ $=R_0$	4	4	05	0	0	0010
T_4^0	0^*-R_0 $=0^*$	5	5	02	0	0010	0
U_2^0	$0\ R_0$ $=0$	6	0	05	0	0010	0
U_3^0	$0\cdot 0$ $=R_0$	7	0	05	0	0	0010
U_4^0	$0-R_0$ $=0$	1100	0	02	0	0010	0

Таблица 4.70

1000 *ЛУГ*	1040	1100 U_4^2
1001	1041 U_2	1101 $T_{3,\text{зап}}$
1002 Форм. T	1042	1102 $T_{4,\text{зап}}$
1003	1043 Нач. s	1103 $U_{3,\text{зап}}$
1004	1044	1104 $U_{4,\text{зап}}$
1005	1045	1105 $(0, n, 0)$
1006	1046 T_3	1106 $(0, n+1, 0)$
1007	1047 T_4	1107 $(n+1, 1, 0)$
1010	1050	1110 $(n+1, 0, n+1)$
1011	1051 U_3	1111 $(n, 0, n)$
1012	1052 U_4	1112 k
1013 Форм. U	1053 Переадр.	1113 s
1014	1054	1114 i
1015	1055	1115
1016	1056	1116
1017	1057 Пров. s	1117
1020	1060	1120

Продолжение

1021	1061 Измен.	1121
1022 Форм. N	1062	1122
1023	1063	1123
1024	1064	1124
1025	1065	1125
1026	1066	1126
1027	1067 Пров. k	1127
1030	1070	1130
1031 Нач. k	1071	1131
1032	1072 T_1^0	1132
1033	1073 T_2^0	1133
1034	1074 T_3^0	1134
1035 T_1	1075 T_4^0	1135
1036	1076 U_2^0	1136
1037 T_2	1077 U_6^0	1137

§ 71. Программирование итерационного процесса для систем линейных уравнений

Как и для одного уравнения (см. § 62), программирование итерационного процесса для системы сводится, в сущности, к итерационному циклу. Однако здесь имеются и некоторые особенности. В настоящем параграфе будет рассмотрена программа для решения системы линейных уравнений по способу Зейделя.

Систему линейных уравнений n-го порядка удобно записывать в виде

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2, \\ \cdots\cdots\cdots\cdots\cdots\cdots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n. \end{aligned}\right\} \qquad (1.71)$$

При подготовке к итерации мы приводим ее к виду (2.67)

$$\begin{aligned} x_1 &= \quad\quad\quad\quad \alpha_{12}x_2 + \ldots + \alpha_{1n}x_n + \beta_1, \\ x_2 &= \alpha_{21}x_1 \quad\quad\quad + \ldots + \alpha_{2n}x_n + \beta_2, \\ \cdots\cdots & \cdots\cdots\cdots\cdots\cdots\cdots \\ x_n &= \alpha_{n1}x_1 + \alpha_{n2}x_2 + \ldots \quad\quad\quad + \beta_n, \end{aligned}$$

разрешая первое уравнение относительно x_1, второе — относительно x_2 и т. д. Коэффициенты в этой системе находятся по формулам

$$\left.\begin{aligned} \alpha_{11} &= 0, \quad \alpha_{12} = -\frac{a_{12}}{a_{11}}, \quad \ldots, \quad \alpha_{1n} = -\frac{a_{1n}}{a_{11}}, \\ \alpha_{21} &= -\frac{a_{21}}{a_{22}}, \quad \alpha_{22} = 0, \quad \ldots, \quad \alpha_{2n} = -\frac{a_{2n}}{a_{22}} \\ \cdots\cdots & \cdots\cdots\cdots\cdots\cdots\cdots \\ \alpha_{n1} &= -\frac{a_{n1}}{a_{nn}}, \quad \alpha_{n2} = -\frac{a_{n2}}{a_{nn}}, \ldots, \quad \alpha_{nn} = 0. \end{aligned}\right\} \qquad (2.71)$$

Начальные приближения неизвестных берутся равными

$$x_1^{(0)} = \beta_1 = \frac{b_1}{a_{11}}, \quad x_2^{(0)} = \beta_2 = \frac{b_2}{a_{22}}, \quad \ldots, \quad x_n^{(0)} = \frac{b_n}{a_{nn}}. \qquad (3.71)$$

Поправки и очередные значения неизвестных вычисляются по формулам (4.68) — (6.68) и (8.68). Вычисления производятся до тех пор, пока все поправки не окажутся по абсолютной величине меньшими заданной величины ε, т. е. до выполнения условий

$$|\delta_1^{(\nu)}| < \varepsilon, \quad |\delta_2^{(\nu)}| < \varepsilon, \quad \ldots, \quad |\delta_n^{(\nu)}| < \varepsilon. \qquad (4.71)$$

Пусть матрица коэффициентов системы записана подряд в ячейках $a_{11}, a_{12}, \ldots, a_{nn}$, а столбец свободных членов — в ячейках b_1, $b_2, \ldots, b_n$. Пусть, кроме того, в памяти отведено рабочее поле $c_{11} — c_{nn}$, размер которого равен размеру матрицы коэффициентов.

Программа решения системы по способу Зейделя состоит из двух блоков:

А) Нахождение коэффициентов α по формулам (2.71) и начальных приближений по формулам (3.71).

Б) Определение поправок и последовательных значений неизвестных и проверка окончания работы программы.

Прежде всего, чтобы не портить матрицу коэффициентов (иначе программа не будет самовосстанавливающейся), перешлем коэффициенты на рабочее поле:

Пересылка:

PA	0	
	a_{11}^*	$=c_{11}^*$
$PA<\overrightarrow{n^2-1}$	$Я-1$	$\bar{1}^*$

Начальное значение x_1^0 и первая строка коэффициентов α могут быть получены на рабочем поле с помощью программы:

Рабочая часть:

T_1	«1» :	c_{11}		$=R_0$
	0 —	R_0		$=R_1$
T_2	b_1 ·	R_0		$=x_1$
T_3		x_1		$=\delta_1$
	$[PA]$	$(0,\ n-1,\ 0)$		
T_4	c_{11}^* ·	R_1		$=c_{11}^*$
	PA $\geqslant\bar{1}$	T_4		F^*.
T_5		0		$=c_{11}$

Для того чтобы, применяя те же команды, получить $x_2^{(0)}$, $\delta_2^{(0)}$ и вторую строку коэффициентов α, команды $T_1 - T_5$ нужно переадресовать следующим образом:

Переадресация:

$T_1+,$	$(0,\ n+1,\ 0)=T_1$
$T_2+,$	$(1,\ 0,\ 1)\ =T_2$
$T_3+,$	$(0,\ 1,\ 1)\ =T_3$
$T_4+,$	$(n,\ 0,\ n)\ =T_4$
$T_5+,$	$(0,\ 0,\ n+1)=T_5$

Теперь мы можем записать программу блока A) в виде блок-программы:

Пересылка

$(0,\ n,\ 0)-,\quad (0,\ 1,\ 0)=k$

Восстановление переменных команд с обходом переадресации и передачей управления на T_1

Переадресация

Рабочая часть

$k\quad -,\quad (0,\ 1,\ 0)=k$

$У_0$ Переадресация

При этом лучше не передавать управление отдельным блокам, а выписать команды этих блоков подряд в виде одной общей программы, чего мы не делаем лишь в целях экономии места.

Программирование блока Б) начнем с определения величины $\delta_1^{(1)}$, по формуле (4.68) в ячейке S:

$$\begin{array}{l|lll} & [PA] & (0,\ n-1,\ 0) & \\ & & 0 & =S \\ \mathrm{T}_6 & \delta_1^* \cdot & c_{i1}^* & =R_0 \\ & S+ & R_0 & =S \\ & PA \geqslant \bar{1} & T_6 & F^* \end{array}$$

Теперь нужно переслать вычисленное в ячейке S значение в ячейку δ_1, использовать его при получении нового приближения и проверить выполнение неравенства (4.71).

Введем специальный счетчик — ячейку q, в которой будем считать, для какого числа поправок это неравенство уже выполняется. (Предварительно в ячейку q нужно записать нуль.) Перечисленные задачи выполняются программой

$$\begin{array}{l|ll} \mathrm{T}_7 & x_1 + S & =x_1 \\ & |S| - |\varepsilon| & =0 \\ \mathrm{T}_8 \quad y_0 & S \quad \text{Я}+2 & \delta_1 \\ & q\ +,(0,\ 1,\ 0) & =q \end{array}$$

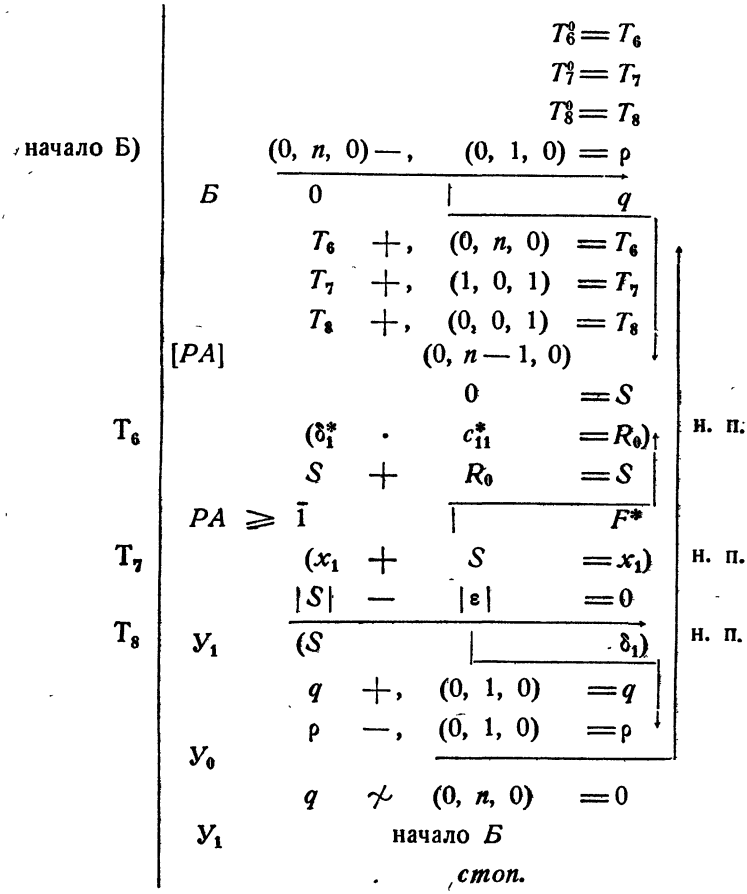
Для нахождения величин $x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)}$ написанные две группы команд следует включить в цикл, причем команды $T_6 - T_8$ переадресуются следующим образом:

$$T_6+,\ (0,\ n,\ 0)=T_6$$
$$T_7+,\ (1,\ 0,\ 1)=T_7$$
$$T_8+,\ (0,\ 0,\ 1)=T_8.$$

Окончание цикла итераций производится с помощью команд

$$q \not< (0,\ n,\ 0)=0$$
$$y_0 \qquad \text{начало } Б$$
$$\text{стоп},$$

так что программу блока Б) можно записать так:



Если программу решения системы оформлять как стандартную, то большое число команд придется формировать. При этом, однако, отпадает необходимость восстановления переменных команд $T_1 — T_8$, так как их можно формировать в начале программы в требуемом виде. Отметим еще, что можно организовать переадресацию команд T_7 и T_8 с помощью регистра адреса.

ГЛАВА XIV

ИНТЕРПОЛЯЦИЯ И ЭКСТРАПОЛЯЦИЯ

§ 72. Общая постановка задачи интерполяции

В § 2 мы уже встречались с задачей интерполяции и простейшим приемом ее решения — линейной интерполяцией. Под интерполяцией там понималось отыскание значений функции, соответствующих промежуточным значениям аргумента, отсутствующим в таблице. Первоначальной задачей интерполяции действительно являлось «искусство чтения между строками». В настоящее время задача интерполяции понимается шире.

В известном смысле можно сказать, что задача интерполяции обратна задаче табулирования функций. Именно, при табулировании по аналитическому выражению функции находится таблица ее значений, а при интерполяции, наоборот, по таблице значений функции строится ее аналитическое выражение. Поясним, что следует понимать под этими словами.

Пусть $y=f(x)$ — некоторая функция, для которой известна лишь таблица ее значений, т. е. известно, что при значениях аргумента $x=x_0, x_1, \ldots, x_n$ функция принимает значения $y_0, y_1, \ldots, y_n$:

$$\left.\begin{aligned} f(x_0)&=y_0,\\ f(x_1)&=y_1,\\ &\cdots\cdots\\ f(x_n)&=y_n. \end{aligned}\right\} \qquad (1.72)$$

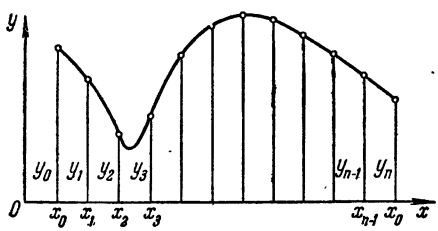


Рис. 59.

Геометрически задача отыскания функции $f(x)$ по заданным ее частным значениям означает, что мы должны построить кривую, проходящую через точки плоскости с координатами (x_0, y_0), (x_1, y_1), ..., (x_n, y_n) (рис. 59). Читателю должно быть ясно, что через данные точки (пусть даже в большом числе) можно провести бесчисленное множество различных кривых. Таким образом, задача отыскания функции $f(x)$ по конечному числу

заданных ее значений слишком неопределенна; таких функций можно построить бесчисленное множество.

Мы будем в дальнейшем обозначать через $F(x)$ л ю б у ю функцию, которая в заданных точках принимает заданные значения. Из сказанного выше следует, что функций $F(x)$ может быть сколько угодно.

Предположим теперь, что функция $F(x)$ не произвольная, а удовлетворяет некоторым дополнительным требованиям; тогда задача приобретает значительно более определенный характер. Чаще всего требуют, чтобы функция $F(x)$ была многочленом степени на единицу меньшим, чем число известных табличных значений.

Таким образом, мы приходим к следующей формулировке задачи.

Для данных значений $x=x_0, x_1, \ldots, x_n$ *и* $y=y_0, y_1, \ldots, y_n$ *найти многочлен* $y=F(x)$ *степени* n, *удовлетворяющий условиям*

$$\left.\begin{aligned} F(x_0)&=y_0,\\ F(x_1)&=y_1,\\ &\cdots\cdots\\ F(x_n)&=y_n. \end{aligned}\right\} \tag{2.72}$$

Иначе говоря, речь идет об отыскании аналитического выражения для м н о г о ч л е н а, принимающего в заданных точках заданные значения. Такую задачу называют задачей *параболической интерполяции*. Точки $x_0, x_1, \ldots, x_n$ называют *узлами интерполяции*.

Многочлен $F(x)$, удовлетворяющий условиям (2.72), называется *интерполяционным многочленом*, а формулы для его построения — *интерполяционными формулами*. Различным интерполяционным формулам будут посвящены следующие параграфы настоящей главы.

Основная идея применения интерполяционных формул состоит в том, что функция $y=f(x)$, для которой известна лишь таблица значений (1.72), заменяется интерполяционным многочленом, который рассматривается как приближенное аналитическое выражение для функции $f(x)$.

Замена функции $f(x)$ ее интерполяционным многочленом используется прежде всего для нахождения промежуточных значений функций, как о том говорилось в самом начале. С этим применением интерполяционных формул связано и их название. Но этим случаем их применение не ограничивается. Такая замена может потребоваться и тогда, когда аналитическое выражение для $f(x)$ известно, но является слишком сложным, а функция $f(x)$ должна подвергаться различным математическим операциям (например, интегрированию). Возможно также, что значения функции $f(x)$ получаются из экспериментальных данных, так что получение других промежуточных значений может оказаться затруднительным или невозможным, а аналитическое выражение функции — неизвестным.

Несмотря на указанную выше «противоположность» задач табулирования и интерполирования, интерполяционные формулы широко используются при составлении таблиц функций. Именно, отдельные значения функции вычисляются (чаще всего с помощью степенных рядов) непосредственно с большой точностью, а затем полученные таблицы многократно уплотняются с помощью интерполяционных формул, благодаря чему получают более или менее подробные таблицы.

Этим обстоятельством мы будем широко пользоваться в настоящей главе. Для примеров будут браться таблицы функций, аналитическое выражение которых известно, со сравнительно большим шагом. Значения, полученные с помощью интерполяционных формул, будут сравниваться со значениями, взятыми из более подробных таблиц для этих же функций.

§ 73. Табличные разности и их свойства

Предположим, что *рассматриваемые значения аргумента являются равноотстоящими*, т. е. образуют арифметическую прогрессию. Такое предположение обычно имеет место при интерполяции функций, заданных в виде таблиц с постоянным шагом, где $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, ..., $x_m = x_0 + mh$. Разность арифметической прогрессии h и называется *шагом таблицы*.

Прежде чем перейти к рассмотрению интерполяционных формул, познакомимся с понятием *конечных разностей*.

Пусть значения функции $f(x)$ заданы в точках x_0, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, ..., $x_n = x_0 + nh$ (*узлы интерполяции*). Составим разности значений функции:

$$y_1 - y_0 = f(x_0 + h) - f(x_0) = \Delta y_0 = \Delta f(x_0),$$
$$y_2 - y_1 = f(x_0 + 2h) - f(x_0 + h) = \Delta y_1 = \Delta f(x_1),$$
$$\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots$$
$$y_n - y_{n-1} = f(x_0 + nh) - f(x_0 + (n-1)h) = \Delta y_{n-1} = \Delta f(x_0 + (n-1)h).$$

Эти значения называют *первыми разностями* функции, или *разностями первого порядка*. По ним мы можем составить разности второго порядка, или *вторые разности*,

$$\Delta^2 y_0 = \Delta y_1 - \Delta y_0, \quad \Delta^2 y_1 = \Delta y_2 - \Delta y_1, \quad \ldots, \quad \Delta^2 y_m = \Delta y_{m+1} - \Delta y_m,$$

и, вообще, разности любого порядка k, или *k-е разности*

$$\Delta^k y_m = \Delta^{k-1} y_{m+1} - \Delta^{k-1} y_m. \tag{1.73}$$

Последовательные разности располагают в форме таблицы. Употребляются две формы таблиц последовательных разностей. В табл. 1.73

Таблица 1.73

x	y	Разности			
		первые	вторые	третьи	...
x_0	y_0				
		Δy_0			
$x_1 = x_0 + h$	y_1		$\Delta^2 y_0$		
		Δy_1		$\Delta^3 y_0$	
$x_2 = x_0 + 2h$	y_2		$\Delta^2 y_1$		
		Δy_2		$\Delta^3 y_1$	
$x_3 = x_0 + 3h$	y_3		$\Delta^2 y_2$		
		Δy_3		$\Delta^3 y_2$	
$x_4 = x_0 + 4h$	y_4		$\Delta^2 y_3$		
		Δy_4		...	
$x_5 = x_0 + 5h$	y_5		...		
		...			
...	...				

разности в каждом столбце вписываются между соответствующими значениями уменьшаемого и вычитаемого; такая таблица называется *диагональной*. В таблице же 2.73 разности ставятся в одной строке с вычитаемым; в этом случае таблицу разностей называют *горизонтальной*. Так как разности функций обычно бывают невелики, то их принято записывать в единицах последнего знака, без нулей впереди.

Равенства (1.73) определяют разности различных порядков последовательно. Установим соотношения, дающие выражения для конечных разностей непосредственно через значения функции. Действительно, так как $\Delta y_0 = y_1 - y_0$ и $\Delta y_1 = y_2 - y_1$, то $\Delta^2 y_0 = (y_2 - y_1) - (y_1 - y_0)$, так что

$$\Delta^2 y_0 = y_2 - 2y_1 + y_0.$$

Точно так же

$$\Delta^3 y_0 = y_3 - 3y_2 + 3y_1 - y_0.$$

Нетрудно доказать, что для любого k

$$\Delta^k y_0 = y_k - k y_{k-1} + \frac{k(k-1)}{2!} y_{k-2} - \ldots + (-1)^{k-1} k y_1 + (-1)^k y_0. \quad (2.73)$$

Таблица 2.73

x	y	Разности			
		первые	вторые	третьи	...
x_0	y_0	Δy_0	$\Delta^2 y_0$	$\Delta^3 y_0$	
$x_1 = x_0 + h$	y_1	Δy_1	$\Delta^2 y_1$	$\Delta^3 y_1$	
$x_2 = x_0 + 2h$	y_2	Δy_2	$\Delta^2 y_2$	$\Delta^3 y_2$	
$x_3 = x_0 + 3h$	y_3	Δy_3	$\Delta^2 y_3$	...	
$x_4 = x_0 + 4h$	y_4	Δy_4	...		
$x_5 = x_0 + 5h$	y_5	...			
...	...				

Эта формула имеет место и для разности $\Delta^k y_m$; достаточно ко всем номерам значений функции прибавить m. Формула (2.73) легко запоминается, если заметить, что ее выражение напоминает разложение бинома $(y-1)^k$ по формуле бинома Ньютона. Надо только вместо степеней y писать y с тем же индексом. Например, вместо y^k следует писать y_k, вместо y^{k-1} писать y_{k-1} и т. д.; в последнем слагаемом вместо $1 = y^0$ пишем y_0.

Нередко бывают полезны и другие формулы, дающие выражение значений функции через конечные разности. Именно, из $\Delta y_0 = y_1 - y_0$ вытекает, что

$$y_1 = y_0 + \Delta y_0.$$

Аналогично, $y_2 = y_1 + \Delta y_1 = (y_0 + \Delta y_0) + \Delta y_1$, но так как $\Delta^2 y_0 = \Delta y_1 - \Delta y_0$, то

$$y_2 = y_0 + 2\Delta y_0 + \Delta^2 y_0.$$

Как и выше, эти выкладки можно провести для любого k, что дает

$$y_k = y_0 + k\Delta y_0 + \frac{k(k-1)}{2!} \Delta^2 y_0 + \ldots + \Delta^k y_0. \quad (3.73)$$

Полученная формула дает возможность определить значения y_k при любом k через значение y_0 и конечные разности до k-го порядка включительно. Число k необходимо является целым. Формулу (3.73) удобно записать символически следующим образом:

$$y_k = (1 + \Delta)^k y_0. \tag{4.73}$$

Здесь скобка $(1 + \Delta)^k$ раскрывается по формуле бинома Ньютона, а полученные «произведения» $\Delta^r y_0$ означают разности соответствующих порядков.

Отметим некоторые простейшие свойства конечных разностей:

1°. Если C постоянно, то $\Delta C = 0$.

2°. $\Delta [Cf(x)] = C\Delta f(x)$.

3°. $\Delta [f_1(x) + f_2(x)] = \Delta f_1(x) + \Delta f_2(x)$.

Перечисленные свойства достаточно очевидны. Кроме них, нам потребуется выражение разности функции $f(x) = x^n$ для целого n:

$$4°.\ \Delta(x^n) = nhx^{n-1} + \frac{n(n-1)}{2!} h^2 x^{n-2} + \ldots + nh^{n-1}x + h^n.$$

Для доказательства этого равенства достаточно написать

$$\Delta(x^n) = (x + h)^n - x^n$$

и воспользоваться формулой бинома Ньютона.

С помощью этих свойств легко получить выражения для последовательных разностей многочлена. Действительно, если

$$y = f(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1}x + a_n, \tag{5.73}$$

то в силу свойств 3° и 2°

$$\Delta y = a_0 \Delta(x^n) + a_1 \Delta(x^{n-1}) + \ldots + a_{n-1}\Delta(x),$$

так что вследствие 4°

$$\Delta y = a_0 \left[nhx^{n-1} + \frac{n(n-1)}{2!} h^2 x^{n-2} + \ldots + nh^{n-1}x + h^n \right] +$$
$$+ a_1 \left[(n-1)hx^{n-2} + \frac{(n-1)(n-2)}{2!} h^2 x^{n-3} + \ldots \right] + \ldots + a_{n-1}h,$$

или, окончательно,

$$\Delta y = a_0 nhx^{n-1} + \left[a_0 \frac{n(n-1)}{2!} h^2 + a_1 (n-1)h \right] x^{n-2} + \ldots + a_{n-1}h. \tag{6.73}$$

Таким образом, *первая разность многочлена степени n* со старшим членом $a_0 x^n$ *есть многочлен степени* $n-1$ со старшим членом $a_0 nhx^{n-1}$. Вычисляя аналогичным путем последовательные разности многочлена любого порядка, убедимся в справедливости следующего утверждения.

Если $f(x)$ — многочлен степени n относительно x со старшим членом a_0x^n, то для любого $m<n$ разность $\Delta^m f(x)$ есть многочлен от x степени $n-m$ со старшим членом

$$n(n-1)\ldots(n-m+1)a_0h^mx^{n-m}.$$

Для $m=n$ разность $\Delta^n f(x)=n!\,a_0h^n$ и при $m>n$ разность $\Delta^m f(x)$ тождественно равна нулю.

Подчеркнем, что этот вывод верен только тогда, когда h постоянно, т. е. значения x образуют арифметическую прогрессию.

Справедлива и обратная теорема, которую мы приведем без доказательства: *если n-ые разности функции, образованные для равноотстоящих значений аргумента при любом шаге h, постоянны, то функция представляет собой многочлен степени n.*

Последнее утверждение, позволяет при отыскании промежуточных значений функции (если шаг таблицы постоянен) ограничиваться вычислениями многочленов, степень которых равна порядку практически постоянных разностей. Способы построения таких многочленов будут указаны в следующем параграфе.

Пример 1.73. Рассмотрим таблицу значений многочлена $f(x)=x^2-3x+2$ (табл. 3.73).

Согласно принятому условию записываем разности в единицах последнего знака без нулей впереди. В соответствии с доказанной выше теоремой, вторые разности оказываются постоянными, а третьи разности равны нулю.

Пример 2.73. Рассмотрим функцию, значения которой заданы табл. 4.73, и составим разности функции.

Мы видим, что третьи разности можно считать практически постоянными, и, следовательно, четвертые равными нулю. Поэтому на рассматриваемом участке функция ведет себя приблизительно как многочлен третьей степени.

Необходимо, однако, иметь в виду, что теорема о конечных разностях для многочленов относится к точным разностям функции. Если же допускать округления, то, как видно из приводимых примеров, порядок практически постоянных разностей зависит как от точности вычисления значений функции, так и от шага таблицы.

Пример 3.73. Рассмотрим многочлен четвертой степени.

$$y=0{,}02x^4+0{,}05x^3+0{,}04x^2-0{,}01x-8{,}85.$$

Точные значения этого многочлена даны в табл. 5.73. Как видно из таблицы, разности четвертого порядка постоянны, что вполне соответствует утверждениям теоремы.

Если же рассматривать значения этого многочлена с точностью до четвертого десятичного знака, то, как показывает табл. 6.73, при сохранении того же шага $h=0{,}1$ практически постоянным можно считать уже третьи разности. Более того, в табл. 7.73 приведены

Таблица 3.73

(1)	(2)	(3)	(4)	(5)
x	y	Разности		
		Δy	$\Delta^2 y$	$\Delta^3 y$
1,0	0,00			
		—16		
1,2	—0,16		8	
		—8		0
1,4	—0,24		8	
		0		0
1,6	—0,24		8	
		8		0
1,8	—0,16		8	
		16		0
2,0	0,00		8	
		24		0
2,2	0,24		8	
		32		0
2,4	0,56		8	
		40		0
2,6	0,96		8	
		48		
2,8	1,44			

Таблица 4.73

(1)	(2)	(3)	(4)	(5)
x	y	Разности		
		Δy	$\Delta^2 y$	$\Delta^3 y$
0,30	1,0313			
		—330		
0,35	0,9983		—38	
		—368		6
0,40	0,9615		—32	
		—400		5
0,45	0,9215		—27	
		—427		4
0,50	0,8788		—23	
		—450		7
0,55	0,8338		—16	
		—466		5
0,60	0,7872		—11	
		—477		6
0,65	0,7395		—5	
		—482		
0,70	0,6913			

Таблица 5.73

(1)	(2)	(3)	(4)	(5)	(6)
x	y	Δy	$\Delta^2 y$	$\Delta^3 y$	$\Delta^4 y$
4,0	0,150000				
		810 972			
4,1	0,960972		53 448		
		864 420		2292	
4,2	1,825392		55 740		48
		920 160		2340	
4,3	2,745552		58 080		48
		978 240		2388	
4,4	3,723792		60 468		48
		1 038 708		2436	
4,5	4,762500		62 904		48
		1 101 612		2484	
4,6	5,864112		65 388		48
		1 167 000		2532	
4,7	7,031112		67 920		48
		1 234 920		2580	
4,8	8,266032		70 500		48
		1 305 420		2628	
4,9	9,571452		73 128		
		1 378 548			
5,0	10,950000				

Таблица 6.73

(1)	(2)	(3)	(4)	(5)
x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
4,0	0,1500			
		8 110		
4,1	0,9610		534	
		8 644		24
4,2	1,8254		558	
		9 202		22
4,3	2,7456		580	
		9 782		25
4,4	3,7238		605	
		10 387		24
4,5	4,7625		629	
		11 016		25
4,6	5,8641		654	
		11 670		25
4,7	7,0311		679	
		12 349		27
4,8	8,2660		706	
		13 055		26
4,9	9,5715		730	
		13 785		
5,0	10,9500			

Таблица 7.73

(1)	(2)	(3)	(4)
x	y	Δy	$\Delta^2 y$
4,0	0,15		
		81	
4,1	0,96		6
		87	
4,2	1,83		5
		92	
4,3	2,75		5
		97	
4,4	3,72		9
		106	
4,5	4,76		4
		110	
4,6	5,86		7
		117	
4,7	7,03		7
		124	
4,8	8,27		6
		130	
4,9	9,57		8
		138	
5,0	10,95		

значения того же многочлена с точностью до одной сотой; из нее мы видим, что в этом случае практически постоянными оказываются уже вторые разности.

Опираясь на замечание, сделанное выше, можно сказать, что если бы мы заранее не знали, что таблицы 6.73 и 7.73 дают значение многочлена четвертой степени, то мы искали бы промежуточные значения с помощью многочленов третьей степени в первом случае и второй степени во втором случае. Это означает, что *чем с меньшей точностью даны значения функции, тем более простые интерполяционные формулы надо выбирать.*

Необходимо еще иметь в виду, что на разностях высших порядков заметно сказываются ошибки округления. Сравнение разностей в табл. 6.73 и 7.73 с разностями в табл. 5.73 наглядно это поясняют.

Чтобы выяснить влияние шага таблицы на разности, рассмотрим таблицу значений того же многочлена с шагом $h = 0{,}01$. Как пока-

Таблица 8.73

(1)	(2)	(3)	(4)
x	y	Δy	$\Delta^2 y$
4,00	0,15000		
		7876	
4,01	0,22876		51
		7927	
4,02	0,30803		51
		7978	
4,03	0,38781		53
		8031	
4,04	0,46812		52
		8083	
4,05	0,54895		52
		8135	
4,06	0,63030		

зывает табл. 8.73, при таком шаге вторые разности оказываются практически постоянными даже для таблицы с пятью знаками.

Если функция на рассматриваемом участке не имеет аналитических особенностей, то обычно ее разности изменяются плавно. Поэтому нарушение плавного изменения разностей позволяет в ряде случаев обнаружить ошибки в отдельных значениях функции, заданной с помощью таблицы.

Пусть одно из значений функции содержит ошибку, равную ε. Влияние этой ошибки на разности функции можно проследить по табл. 9.73. Из нее видно, что ошибка тем больше, чем выше порядок разности.

Таблица 9.73

(1)	(2)	(3)	(4)	(5)
x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
...	...	...	...	...
x_{n-2}	y_{n-2}			
		Δy_{n-2}		$\Delta^3 y_{n-3}+\varepsilon$
x_{n-1}	y_{n-1}		$\Delta^2 y_{n-2}+\varepsilon$	
		$\Delta y_{n-1}+\varepsilon$		$\Delta^3 y_{n-2}-3\varepsilon$
x_n	$y_n+\varepsilon$		$\Delta^2 y_{n-1}-2\varepsilon$	
		$\Delta y_n-\varepsilon$		$\Delta^3 y_{n-1}+3\varepsilon$
x_{n+1}	y_{n+1}		$\Delta^2 y_n+\varepsilon$	
		Δy_{n+1}		$\Delta^3 y_n-\varepsilon$
x_{n+2}	y_{n+2}		...	
...	...	...		

Если предположить, что вторые разности в рассматриваемой таблице практически постоянны всюду, кроме выделенного участка, то в качестве исправленного значения второй разности $\Delta^2 y_{n-1}$ можно принять среднее арифметическое трех выписанных в табл. 9.73 вторых разностей, потому что сумма этих трёх разностей уже не

содержит ошибки. Таким образом, полагаем

$$\overline{\Delta^2 y_{n-1}} = \frac{1}{3}[(\Delta^2 y_{n-2} + \varepsilon) + (\Delta^2 y_{n-1} - 2\varepsilon) + (\Delta^2 y_n + \varepsilon)] = \\ = \frac{\Delta^2 y_{n-2} + \Delta^2 y_{n-1} + \Delta^2 y_n}{3}, \qquad (7.73)$$

где $\overline{\Delta^2 y_{n-1}}$ означает исправленное значение второй разности. Напомним, что вторые разности предполагаются постоянными.

Зная исправленное значение второй разности, легко найдем ошибку значения y_n, находящегося в той же горизонтальной строке, что и исправляемая вторая разность

$$\varepsilon = \frac{1}{2}(\Delta^2 y_{n-1} - \overline{\Delta^2 y_{n-1}}). \qquad (8.73)$$

Ошибка выражается в целых единицах последнего разряда, как и значения функции и все разности.

Исправив ошибочное значение функции, следует снова пересчитать все разности. Нужно только иметь в виду, что таким путем могут быть обнаружены лишь отдельные ошибки, находящиеся в таблице далеко друг от друга. Кроме того, функция может иметь на рассматриваемом участке такие особенности (например, резко выраженный максимум), что ее разности вообще не будут иметь плавного течения.

Пример 4.73. Функция задана таблицей (табл. 10.73), в которой вторые разности можно считать практически постоянными, кроме участка, соответствующего значениям аргумента 0,25, 0,30, 0,35. Естественно считать, что значение y, соответствующее $x = 0{,}30$, содержит ошибку.

Исправленное значение второй разности найдем по формуле (7.73)

$$\overline{\Delta^2 y} = \frac{48 + 9 + 48}{3} = 35$$

в единицах седьмого знака. Ошибка найдется по формуле (8.73)

$$\varepsilon = \frac{1}{2}(9 - 35) = -13$$

единиц седьмого знака. Таким образом, исправленное значение функции при $x = 0{,}30$ должно быть $y = 0{,}302\,2637$ вместо указанного в таблице 0,302 2650. Пересчитывая разности (табл. 11.73), убеждаемся, что у вновь полученной таблицы вторые разности практически постоянны всюду.

Рассмотренный прием позволяет исправлять отдельные ошибки функции также и в том случае, когда постоянны третьи разности.

Таблица 10.73

(1)	(2)	(3)	(4)
x	y	Δy	$\Delta^2 y$
0,00	0,292 8341		
		15 632	
0,05	0,294 3973		33
		15 665	
0,10	0,295 9638		34
		15 699	
0,15	0,297 5337		34
		15 733	
0,20	0,299 1070		33
		15 766	
0,25	0,300 6836		48
		15 814	
0,30	0,302 2650		9
		15 823	
0,35	0,303 8473		48
		15 871	
0,40	0,305 4344		35
		15 906	
0,45	0,307 0250		35
		15 941	
0,50	0,308 6191		

Таблица 11.73

(1)	(2)	(3)	(4)
x	y	Δy	$\Delta^2 y$
0,20	0,299 1070		
		15 766	
0,25	0,300 6836		35
		15 801	
0,30	0,302 2637		35
		15 836	
0,35	0,303 8473		35
		15 871	
0,40	0,305 4344		

В заключение отметим связь между конечными разностями и производными функциями. Из определения производной следует, что

$$f'(x)=\lim_{h\to 0}\frac{f(x+h)-f(x)}{h}=\lim_{h\to 0}\frac{\Delta f(x)}{h}.$$

Отсюда вытекает, что при малых h имеем $f'(x)\approx\frac{\Delta f(x)}{h}$, или, в более короткой записи,

$$y'\approx\frac{\Delta y}{h}. \tag{9.73}$$

Найдем теперь

$$\lim_{h\to 0}\frac{\Delta^2 y_0}{h^2}=\lim_{h\to 0}\frac{f(x+2h)-2f(x+h)+f(x)}{h^2}.$$

Считая функцию $f(x)$ дважды непрерывно дифференцируемой, применим два раза правило Лопиталя (при постоянном x и переменном h). Тогда

$$\lim_{h\to 0}\frac{f(x+2h)-2f(x+h)+f(x)}{h^2}=\lim_{h\to 0}\frac{f'(x+2h)\cdot 2-2f'(x+h)}{2h}=$$

$$=\lim_{h\to 0}\frac{f''(x+2h)\cdot 2-f''(x+h)}{1}=f''(x),$$

т. е.

$$\lim_{h\to 0}\frac{\Delta^2 y_0}{h^2}=f''(x).$$

Отсюда при малых h будем иметь

$$y'' \approx \frac{\Delta^2 y_0}{h^2}.$$

Рассуждая аналогичным образом и дальше, получим приближенную формулу для любого n (предполагая, что функция $y=f(x)$ имеет n-ю непрерывную производную)

$$y^{(n)} \approx \frac{\Delta^n y}{h^n}. \tag{10.73}$$

Формулы (9.73) и (10.73) могут быть использованы для приближенного нахождения производных. Однако их точность весьма невелика (особенно при $n > 1$), поэтому на практике применяются более совершенные приемы.

§ 74. Точность линейной интерполяции. Квадратичная интерполяция по схеме Эйткина

В § 2 мы рассматривали линейную интерполяцию, суть которой как там было сказано, состоит в том, что на участке между двумя табличными значениями аргумента мы считаем функцию изменяющейся линейно. Пользуясь рассмотренными свойствами разностей функции, мы можем теперь сказать, что линейная интерполяция предполагает на рассматриваемом участке постоянство первых разностей. Во всех тех случаях, когда первые разности изменяются, линейная интерполяция будет давать некоторую погрешность.

Ясно, что речь идет не о точном постоянстве первых разностей, что возможно, как об этом было сказано в предыдущем параграфе, только в случае линейной функции, а о том, чтобы первые разности были практически постоянными. Во всех случаях, когда функция отлична от линейной и первые разности не являются точно постоянными, линейная интерполяция приводит к погрешностям в определении промежуточных значений функции. Эта погрешность будет тем более велика, чем больше различаются между собой разности функции, т. е. чем больше ее вторые разности.

Не входя в подробное рассмотрение вопросов оценки погрешностей интерполяции, приведем без доказательства важное правило, позволяющее оценить точность линейной интерполяции. Это правило гласит:

абсолютная ошибка линейной интерполяции по таблице с постоянным шагом и практически постоянными вторыми разностями не превосходит восьмой части абсолютной величины второй разности.

Из этого правила мы заключаем, что линейная интерполяция дает приблизительно на один знак больше, чем величина второй разности. Это означает, что если, например, вторая разность есть величина порядка десяти единиц последнего знака таблицы, то линейная интерполяция дает точность порядка единицы последнего знака. Если вторая

разность существенно больше, то линейная интерполяция может оказаться уже непригодной. Она будет тем более непригодной, если вторая разность на рассматриваемом участке таблицы заметно изменяется. В таких случаях требуется интерполяция более высокой степени.

Наиболее часто используемым способом параболической интерполяции более высокой степени является *квадратичная интерполяция*, для которой требуются значения функции уже не в двух, а в трех точках. Геометрический смысл ее состоит в том, что через выбранные три точки проводится парабола вида $y = Ax^2 + Bx + C$ и принимается, что значения функции на рассматриваемом участке могут быть вычислены как значения построенного квадратного трехчлена. Коэффициенты A, B, C трехчлена находятся из условия прохождения параболы через заданные точки.

При практическом применении квадратичной интерполяции, даже если не пользоваться специальными формулами, речь о которых будет идти в следующем параграфе, нет никакой нужды в действительном вычислении коэффициентов квадратного трехчлена.

Для вычисления значений функции, соответствующих промежуточным значениям аргумента, при квадратичной интерполяции удобно пользоваться схемой Эйткина, аналогичной схеме линейной интерполяции, рассмотренной в § 2. Пусть даны три значения аргумента x_0, x_1, x_2, в которых функция принимает соответственно значения y_0, y_1, y_2. Линейная интерполяция на каждом из этих участков осуществляется выражениями вида

$$P_{0,1}(x) = \frac{1}{x_1 - x_0}\begin{vmatrix} y_0 & x_0 - x \\ y_1 & x_1 - x \end{vmatrix}, \tag{1.74}$$

$$P_{1,2}(x) = \frac{1}{x_2 - x_1}\begin{vmatrix} y_1 & x_1 - x \\ y_2 & x_2 - x \end{vmatrix}. \tag{2.74}$$

Значение функции в точке x, соответствующее квадратичной интерполяции по указанным трем точкам, получается путем линейной интерполяции между значениями этих линейных выражений, т. е. по формуле

$$P_{0,1,2}(x) = \frac{1}{x_2 - x_0}\begin{vmatrix} P_{0,1}(x) & x_0 - x \\ P_{1,2}(x) & x_2 - x \end{vmatrix}. \tag{3.74}$$

Действительно, так как $P_{0,1}$ и $P_{1,2}$ являются, как это видно из их выражений, многочленами первой степени, то $P_{0,1,2}$ — второй степени относительно x. Далее, по построению $P_{0,1}(x)$ и $P_{1,2}(x)$ имеем

$$P_{0,1}(x_0) = y_0; \quad P_{0,1}(x_1) = y_1; \quad P_{1,2}(x_1) = y_1; \quad P_{1,2}(x_2) = y_2.$$

Поэтому

$$P_{0,1,2}(x_0) = \frac{1}{x_2 - x_0}\begin{vmatrix} P_{0,1}(x_0) & 0 \\ P_{1,2}(x_0) & x_2 - x_0 \end{vmatrix} = P_{0,1}(x_0) = y_0.$$

Аналогично,

$$P_{0,1,2}(x_2)=\frac{1}{x_2-x_0}\begin{vmatrix} P_{0,1}(x_2) & x_0-x_2 \\ P_{1,2}(x_2) & 0 \end{vmatrix}=P_{1,2}(x_2)=y_2.$$

Кроме того, при $x=x_1$

$$P_{0,1,2}(x_1)=\frac{1}{x_2-x_0}\begin{vmatrix} P_{0,1}(x_1) & x_0-x_1 \\ P_{1,2}(x_1) & x_2-x_1 \end{vmatrix}=\frac{1}{x_2-x_0}\begin{vmatrix} y_1 & x_0-x_1 \\ y_1 & x_2-x_1 \end{vmatrix}=$$

$$=\frac{y_1}{x_2-x_0}[(x_2-x_1)-(x_0-x_1)]=y_1.$$

Итак, мы получили, что многочлен $P_{0,1,2}(x)$ принимает в точках x_0, x_1, x_2 соответственно значения y_0, y_1, y_2, так что он и дает требуемую нам параболу.

Пример 1.74. Функция $y=f(x)$ задана в таблице 1.74. С помощью квадратичной интерполяции по схеме Эйткина найдем значение функции $f(13{,}13)$. Шаг таблицы равен здесь $h=0{,}05$. За x_0 примем значение $x_0=13{,}10$.

Таблица 1.74

(1)	(2)
x	$f(x)$
13,00	1,49936 17229
13,05	1,50106 09921
13,10	1,50292 30057
13,15	1,50494 18872
13,20	1,50711 14191
13,25	1,50942 50600

На участке (13,10; 13,15) находим

$$P_{0,1}=\frac{1}{0{,}05}\begin{vmatrix} 1{,}50292\,30057 & -0{,}03 \\ 1{,}50494\,18872 & 0{,}02 \end{vmatrix}=1{,}50413\,43346$$

Это значение является искомым значением функции, полученным линейной интерполяцией. Для следующего участка

$$P_{1,2}=\frac{1}{0{,}05}\begin{vmatrix} 1{,}50494\,18872 & 0{,}02 \\ 1{,}50711\,14191 & 0{,}07 \end{vmatrix}=1{,}50407\,40744.$$

Теперь для квадратичной интерполяции по формуле (3.74) получаем

$$P_{0,1,2}=\frac{1}{0{,}1}\begin{vmatrix} 1{,}50413\,43346 & -0{,}03 \\ 1{,}50407\,40744 & 0{,}07 \end{vmatrix}=1{,}50411\,62565.$$

Это и есть искомое значение функции. Точное значение функции, взятое из более подробной таблицы, равно $f(13{,}13)=1{,}50411\,59009$.

Сравнение результатов, полученных линейной и квадратичной интерполяцией с точным значением, показывает, что квадратичная интерполяция сохраняет верными семь значащих цифр, тогда как линейная интерполяция только пять.

Рассмотренная здесь схема Эйткина может применяться не только для квадратичной интерполяции, но и для параболической интерполяции более высоких степеней. Например, если функция задана в четырех точках, то для интерполяции третьей степени можно воспользоваться формулой

$$P_{0,1,2,3}(x)=\frac{1}{x_3-x_0}\begin{vmatrix} P_{0,1,2}(x) & x_0-x \\ P_{1,2,3}(x) & x_3-x \end{vmatrix}, \tag{4.74}$$

где $P_{0,1,2}$ и $P_{1,2,3}$ получены по формулам типа (3.74).

§ 75. Интерполяционные формулы Лагранжа и Ньютона

Задача параболической интерполяции в том общем виде, как она была поставлена в § 72, решается с помощью различных *интерполяционных формул*. Как уже было сказано, эту задачу можно формулировать следующим образом: *рассматривается функция* $f(x)$, *для которой заданы значения* $y_i=f(x_i)$ $(i=0, 1, 2, \ldots, n)$, *причем все* x_i *и* y_i *известны. Требуется определить многочлен* $y=F(x)$ *степени* n, *для которого* $F(x_i)=f(x_i)$ $(i=0, 1, 2, \ldots, n)$.

Мы можем написать

$$F(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n. \tag{1.75}$$

Используя написанные выше условия, получим для нахождения неизвестных коэффициентов $a_0, a_1, a_2, \ldots, a_n$ систему $n+1$ уравнений с $n+1$ неизвестными:

$$\left.\begin{aligned} a_0+a_1x_0+a_2x_0^2+\ldots+a_nx_0^n&=y_0,\\ a_0+a_1x_1+a_2x_1^2+\ldots+a_nx_1^n&=y_1,\\ \cdots\cdots\cdots\cdots\cdots\cdots&\\ a_0+a_1x_n+a_2x_n^2+\ldots+a_nx_n^n&=y_n. \end{aligned}\right\} \tag{2.75}$$

Дальше будет показано, что эта система имеет единственное решение, если значения $x_0, x_1, \ldots, x_n$ отличны друг от друга; тогда, определив значения коэффициентов a_i из системы (2.75), мы и получим интерполяционный многочлен $F(x)$, дающий решение поставленной задачи.

Перейдем теперь к решению общей задачи отыскания интерполяционного многочлена. Вместо того, чтобы решать систему уравнений (2.75), мы поступим иначе, непосредственно построив многочлен $F(x)$, удовлетворяющий всем поставленным условиям.

Найдем прежде всего выражение для многочлена, принимающего в точке $x=x_0$ значение $y_0=1$, а в точках $x=x_1$, $x=x_2$, ...,

$x=x_n$ — значения $y_1=y_2=\ldots=y_n=0$. Легко проверить, что такой многочлен будет иметь вид

$$\frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)}.$$

В самом деле, значения $x=x_1$, $x=x_2, \ldots, x=x_n$ являются корнями нашего многочлена, а при $x=x_0$ числитель равен знаменателю.

Теперь построим многочлен $y=F_0(x)$, принимающий в точке $x=x_0$ значение y_0 и обращающийся в нуль для значений $x=x_i$ $(i=1, 2, \ldots, n)$. Учитывая предыдущее построение, легко заметить, что многочлен $F_0(x)$ должен иметь вид

$$F_0(x)=\frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)}y_0.$$

После этих предварительных построений можно перейти к нахождению многочлена $F(x)$, принимающего в точках $x=x_i$ $(i=0, 1, 2, \ldots, n)$ заданные значения $F(x_i)=y_i$ $(i=0, 1, 2, \ldots, n)$. Для этого определим при фиксированном j $(0\leqslant j\leqslant n)$ многочлен $F_j(x)$, принимающий в точке $x=x_j$ значение $F_j(x_j)=y_j=F(x_j)$, а во всех остальных точках $x=x_i$ $(i=0, 1, 2, \ldots, n, i\neq j)$ — значения $F_j(x_i)=0$. Это будет многочлен

$$F_j(x)=\frac{(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)(x_j-x_1)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}y_j.$$

Искомый многочлен будет равен сумме

$$F(x)=\sum_{j=0}^{n}F_j(x),$$

так как в каждой точке x_j одно из слагаемых принимает нужное значение y_j, а все остальные слагаемые обращаются в нуль. Итак, многочлен, удовлетворяющий условиям поставленной задачи, найден. Подставляя вместо $F_j(x)$ их выражения, получим

$$F(x)=\sum_{j=0}^{n}\frac{(x-x_0)(x-x_1)\ldots(x-x_{j-1})(x-x_{j+1})\ldots(x-x_n)}{(x_j-x_0)(x_j-x_1)\ldots(x_j-x_{j-1})(x_j-x_{j+1})\ldots(x_j-x_n)}y_j, \tag{3.75}$$

или, в развернутом виде,

$$F(x)=\frac{(x-x_1)(x-x_2)\ldots(x-x_n)}{(x_0-x_1)(x_0-x_2)\ldots(x_0-x_n)}y_0+$$
$$+\frac{(x-x_0)(x-x_2)\ldots(x-x_n)}{(x_1-x_0)(x_1-x_2)\ldots(x_1-x_n)}y_1+\ldots+$$
$$+\frac{(x-x_0)(x-x_1)\ldots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\ldots(x_n-x_{n-1})}y_n. \tag{4.75}$$

Полученная формула называется *интерполяционной формулой Лагранжа.*

Покажем теперь, что интерполяционный многочлен Лагранжа является единственным решением поставленной задачи. Действительно, пусть существует еще один многочлен $R(x)$ степени n, принимающий в заданных точках заданные значения. Тогда разность $F(x)-R(x)$ представляет собой многочлен степени не выше n, который обращается в нуль в точках $x=x_i$ $(i=0, 1, 2, \ldots, n)$, т. е. имеет $n+1$ корень. Отсюда следует, что эта разность равна нулю тождественно, так как многочлен степени не выше n не может иметь $n+1$ корень.

Мы заключаем, таким образом, что, *каковы бы ни были значения* $x_0, x_1, \ldots, x_n$, *среди которых нет совпадающих, и совершенно произвольные значения* $y_0, y_1, \ldots, y_n$, *существует единственный многочлен* $F(x)$ *степени* n, *принимающий в заданных точках заданные значения, т. е. удовлетворяющий условиям* $F(x_i)=y_i$ $(i=0, 1, 2, \ldots, n)$.

Пример 1.75. Положим $n=1$. Ясно, что мы имеем в этом случае две точки и интерполяционная формула Лагранжа дает уравнение прямой, проходящей через две заданные точки. Обозначив абсциссы этих точек через x_0 и x_1, получим интерполяционный многочлен в виде

$$F(x)=\frac{x-x_1}{x_0-x_1}y_0+\frac{x-x_0}{x_1-x_0}y_1. \tag{5.75}$$

Очевидно, что он совпадает с многочленом $P_{0,1}(x)$, построенным по схеме Эйткина.

Пример 2.75. Примем $n=2$. Тогда мы получим уравнение параболы, проходящей через три точки. Оно будет иметь вид

$$F(x)=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1+$$
$$+\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2. \tag{6.75}$$

Читатель легко может убедиться, что формула (3.74) для многочлена $P_{0,1,2}(x)$, осуществляющего квадратичную интерполяцию, совпадает с формулой (6.75), так же как формула (1.74) для линейной интерполяции совпадает с формулой (5.75). Следовательно, приведенные в предыдущем параграфе формулы являются частными случаями интерполяционной формулы Лагранжа.

Интерполяционные формулы Ньютона, к рассмотрению которых мы переходим, предназначены для решения той же общей интерполяционной задачи, что и формула Лагранжа. При их выводе сделаем дополнительное предположение, что рассматриваются *равноотстоящие значения аргумента.*

Пусть значения функции $y=f(x)$ заданы для равноотстоящих значений аргумента x_0, $x_1=x_0+h$, $x_2=x_0+2h$, $\ldots$, $x_n=x_0+nh$.

Обозначим эти значения соответственно $y_0 = f(x_0)$, $y_1 = f(x_1)$, ..., $y_n = f(x_n)$.

Как было показано выше, существует единственный многочлен $F(x)$ степени n, такой, что $F(x_k) = y_k$ $(k = 0, 1, \ldots, n)$. Сейчас речь пойдет о другом способе записи и отыскания этого многочлена, который в конечном счете совпадает с многочленом, полученным по формуле Лагранжа. Запишем искомый многочлен в виде

$$F(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \\ + a_3(x - x_0)(x - x_1)(x - x_2) + \ldots \\ \ldots + a_n(x - x_0)(x - x_1)\ldots(x - x_{n-1}). \qquad (7.75)$$

Для определения коэффициентов $a_0, a_1, \ldots, a_n$ положим в (7.75) $x = x_0$. Тогда

$$y_0 = F(x_0) = a_0.$$

Далее, полагая $x = x_1$, получаем

$$y_1 = F(x_1) = a_0 + a_1(x_1 - x_0),$$

или, так как $x_1 - x_0 = h$,

$$y_1 = y_0 + a_1 h,$$

откуда

$$a_1 = \frac{y_1 - y_0}{h} = \frac{\Delta y_0}{h}.$$

Продолжая вычисление коэффициентов, положим $x = x_2$. Тогда

$$y_2 = F(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1).$$

Заменим найденные коэффициенты a_0, a_1 их значениями

$$y_2 = y_0 + \frac{\Delta y_0}{h} \cdot 2h + a_2 \cdot 2h \cdot h;$$

тогда

$$y_2 - 2\Delta y_0 - y_0 = y_2 - 2y_1 + y_0 = 2a_2 h^2.$$

Воспользовавшись формулой (2.73), выражающей разности через значения функции, получим

$$a_2 = \frac{\Delta^2 y_0}{2!\, h^2}.$$

Точно так же получим

$$a_3 = \frac{\Delta^3 y_0}{3!\, h^3}.$$

Аналогичные дальнейшие вычисления позволяют записать общую формулу для нахождения коэффициентов

$$a_k = \frac{\Delta^k y_0}{k!\, h^k}.$$

Подставив найденные выражения коэффициентов в формулу (7.75), находим

$$F(x)=y_0+\frac{\Delta y_0}{1!\,h}(x-x_0)+\frac{\Delta^2 y_0}{2!\,h^2}(x-x_0)(x-x_1)+$$
$$+\frac{\Delta^3 y_0}{3!\,h^3}(x-x_0)(x-x_1)(x-x_2)+\ldots+$$
$$+\frac{\Delta^n y_0}{n!\,h^n}(x-x_0)(x-x_1)\ldots(x-x_{n-1}). \qquad (8.75)$$

Полученную формулу и называют *первой интерполяционной формулой Ньютона.*

Теперь ясно видно различие между формулами Ньютона и Лагранжа. В формуле Лагранжа (4.75) каждое из слагаемых представляет многочлен n-й степени и все эти слагаемые равноправны. Поэтому мы не можем заранее (т. е. до производства вычислений) пренебрегать какими-либо из них. В формулу же Ньютона входят в качестве слагаемых многочлены повышающихся степеней, причем коэффициентами при них служат последовательные конечные разности, деленные на факториалы. Как мы уже видели в § 73, последовательные разности обычно довольно быстро уменьшаются. Мы получаем поэтому возможность не учитывать в формуле Ньютона тех слагаемых, коэффициенты при которых становятся пренебрежимо малыми. Благодаря этому можно вычислять промежуточные значения функции достаточно точно, пользуясь простыми интерполяционными формулами.

Для практического использования формулу Ньютона (8.75) обычно записывают в несколько преобразованном виде. Чтобы получить его, введем обозначение

$$\frac{x-x_0}{h}=t \text{ или } x=x_0+th.$$

Множители, входящие в формулу (8.75), выразятся через t следующим образом:

$$\frac{x-x_1}{h}=\frac{x-x_0-h}{h}=t-1,$$
$$\frac{x-x_2}{h}=\frac{x-x_0-2h}{h}=t-2.$$
$$\ldots\ldots\ldots\ldots\ldots$$
$$\frac{x-x_{n-1}}{h}=\frac{x-x_0-(n-1)h}{h}=t-n+1.$$

Подставив эти выражения в формулу (8.75), приведем ее к виду

$$F(x_0+th)=y_0+\frac{t}{1!}\Delta y_0+\frac{t(t-1)}{2!}\Delta^2 y_0+\frac{t(t-1)(t-2)}{3!}\Delta^3 y_0+$$
$$+\ldots+\frac{t(t-1)\ldots(t-n+1)}{n!}\Delta^n y_0. \qquad (9.75)$$

Это и есть окончательный вид первой интерполяционной формулы Ньютона. По причинам, которые будут указаны ниже, ее называют также *интерполяционной формулой Ньютона для интерполирования вперед.*

Если в формуле (9.75) положить t равным целому числу k ($k \leqslant n$), то правая часть формулы будет содержать $k+1$ слагаемое, причем последнее из них будет

$$\frac{k(k-1)\ldots(k-k+1)}{k!}\Delta^k y_0 = \Delta^k y_0.$$

Коэффициенты при всех последующих слагаемых будут содержать множитель $(k-k)$ и, следовательно, обратятся в нуль. Формула для $F(x_0+kh)$ примет тогда в точности тот же вид, что и формула (3.73), выражающая y_k через y_0 и последовательные разности. Следовательно,

$$F(x_0+kh)=y_k.$$

Впрочем, последнее равенство ясно и из хода вывода формулы (3.73).

Если же k не равно целому числу, а этот случай и является наиболее интересным, то формула (9.75) дает значения функции $F(x)$ для значений аргумента, отсутствующих в исходной таблице, т. е. для значений x, не совпадающих ни с одним из x_k ($k=0, 1, \ldots, n$).

Формулу (9.75), как и формулу (3.73), можно записать в символической форме

$$F(x_0+th)=(1+\Delta)^t y_0, \qquad (10.75)$$

где выражение $(1+\Delta)^t$ нужно разложить в биномиальный ряд и оборвать его на члене, содержащем Δ^n, и каждое произведение Δ^r на y_0 заменить разностью $\Delta^r y_0$.

Формулой (9.75) следует пользоваться при вычислении значений функции $f(x)$ для значений аргумента, лежащих между x_0 и x_1, т. е. для $t<1$. Переходя к интервалу $x_1<x<x_2$, нецелесообразно брать ту же формулу, так как t будет больше единицы. В этом случае удобнее принимать за x_0 следующий узел интерполяции.

Приведем теперь пример применения интерполяционной формулы Ньютона.

Пример 3.75. По данной таблице значений семизначных логарифмов для чисел от 1000 до 1050 с шагом 10 (табл. 1.75) вычислить значения логарифмов всех целых чисел от 1000 до 1010.

Составленная таблица разностей, записанная как диагональная, показывает, что третьи разности функции можно считать практически постоянными. Полагаем $x_0=1000$, $y_0=3{,}0000000$, $\Delta y_0=0{,}0043214$, $\Delta^2 y_0=-0{,}0000426$, $\Delta^3 y_0=0{,}0000008$. Остается определить значение t. Так как здесь $h=10$, то для $x=1001$ имеем $t_1=0{,}1$, для $x=1002$ будет $t_2=0{,}2$ и т. д.

Таблица 1.75

(1)	(2)	(3)	(4)	(5)
x	y	Δy	$\Delta^2 y$	$\Delta^3 y$
1000	3,000 0000			
		43 214		
1010	3,004 3214		—426	
		42 788		8
1020	3,008 6002		—418	
		42 370		9
1030	3,012 8372		—409	
		41 961		8
1040	3,017 0333		—401	
		41 560		
1050	3,021 1893			

Для определения искомых значений y при выбранных t удобно расположить требуемые вычисления в таблицу (табл. 2.75). При этом промежуточные вычисления проводятся с одним запасным знаком, а окончательные результаты округлены.

Проверив полученные значения по семизначной таблице логарифмов, убедимся, что полученные значения верны до последнего знака.

Для составления таблицы значений в интервале $1010 < x < 1020$ полагаем $x_0 = 1010$. Тогда $y_0 = 3{,}0043214$, $\Delta y_0 = 0{,}0042788$, $\Delta^2 y_0 = -0{,}0000418$, $\Delta^3 y_0 = 0{,}0000009$; далее поступаем так же, как и ранее.

Как видно из рассмотренного примера, в интерполяционной формуле (9.75) используются разности, идущие по диагонали вниз. Поэтому применение этой формулы удобно в начале таблицы, где имеется достаточное число разностей.

Наоборот, она оказывается непригодной в конце таблицы, где этих разностей слишком мало.

Так, в том же примере для $x_0 = 1030$ имеется только первая и вторая разности, а для $x_0 = 1040$ — только первая.

Таблица 2.75

(1)	(2)	(3)	(4)	(5)	(6)
t	x	$t\,\Delta y_0$	$\frac{t(t-1)}{2}\Delta^2 y_0$	$\frac{t(-1)(1-2)}{6}\Delta^3 y_0$	$y = y_0 + (3) + (4) + (5)$
0	1000	0	0	0	3,000 0000
0,1	1001	43 214	192	2	3,000 4341
0,2	1002	86 428	341	4	3,000 8677
0,3	1003	129 642	447	5	3,001 3009
0,4	1004	172 856	511	5	3,001 7337
0,5	1005	216 070	532	5	3,002 1661
0,6	1006	259 284	511	5	3,002 5930
0,7	1007	302 498	447	4	3,003 0295
0,8	1008	345 712	343	2	3,003 1606
0,9	1009	388 926	192	1	3,003 8912
1,0	1010	432 140	0	0	3,004 3214

Таким образом, первая интерполяционная формула Ньютона наиболее удобна для отыскания значений функции, соответствующих бо́льшим, нежели начальные, значениям аргумента, чем и объясняется приведенное выше ее другое название — интерполяционная формула для интерполирования вперед.

Для интерполирования в конце таблицы применяется иная формула, которую мы сейчас и рассмотрим. Напишем искомый интерполяционный многочлен в форме

$$F(x) = a_0 + a_1(x - x_n) + a_2(x - x_n)(x - x_{n-1}) + \ldots + a_n(x - x_n)(x - x_{n-1})\ldots(x - x_1). \quad (11.75)$$

Как и выше, коэффициенты $a_0, a_1, \ldots, a_n$ определяются из условия $y_i = f(x_i) = F(x_i)$. Положим в (11.75) $x = x_n$. Тогда

$$a_0 = y_n.$$

Точно так же, полагая $x = x_{n-1}$, имеем

$$y_{n-1} = y_n + a_1(x_{n-1} - x_n),$$

а так как $x_{n-1} - x_n = -h$, то

$$a_1 = \frac{y_n - y_{n-1}}{h} = \frac{\Delta y_{n-1}}{h}.$$

Далее, полагая в (11.75) $x = x_{n-2}$ и заменяя найденные коэффициенты a_0, a_1 их значениями, получаем

$$a_2 = \frac{y_n - 2y_{n-1} + y_{n-2}}{2h^2} = \frac{\Delta^2 y_{n-2}}{2!\,h^2}.$$

Продолжая аналогичные вычисления, получим общую формулу для коэффициентов

$$a_k = \frac{\Delta^k y_{n-k}}{k!\,h^k} \qquad (k = 1, 2, \ldots, n).$$

После подстановки в (11.75) полученных значений коэффициентов

формула примет вид

$$F(x)=y_n+\frac{\Delta y_{n-1}}{1!\,h}(x-x_n)+\frac{\Delta^2 y_{n-2}}{2!\,h^2}(x-x_n)(x-x_{n-1})+\ldots+$$
$$+\frac{\Delta^n y_0}{n!\,h^n}(x-x_n)(x-x_{n-1})\ldots(x-x_1). \qquad (12.75)$$

Это и есть *вторая интерполяционная формула Ньютона.* Для применения, однако, ее предварительно преобразуют, как и первую. Введем обозначение

$$\frac{x-x_n}{h}=t \text{ или } x=x_n+th.$$

Выразим теперь через t множители в формуле (12.75):

$$\frac{x-x_{n-1}}{h}=\frac{x-(x_n-h)}{h}=t+1,$$
$$\frac{x-x_{n-2}}{h}=\frac{x-(x_n-2h)}{h}=t+2,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\frac{x-x_1}{h}=\frac{x-(x_n-(n-1)h)}{h}=t+n-1.$$

Произведя такую замену, получим окончательно

$$F(x)=F(x_n+th)=y_n+t\Delta y_{n-1}+\frac{t(t+1)}{2!}\Delta^2 y_{n-2}+\ldots+$$
$$+\frac{t(t+1)\ldots(t+n-1)}{n!}\Delta^n y_0. \qquad (13.75)$$

Формулу (13.75) называют *второй интерполяционной формулой Ньютона* или *интерполяционной формулой Ньютона для интерполирования назад.* Приведем пример использования формулы (13.75) для интерполяции.

Пример 4.75. По таблице, данной в примере 3.75 (табл. 1.75), найти значения логарифма для $x=1044$. Примем $x_n=1050$, $y_n=3{,}0211893$, $\Delta y_{n-1}=0{,}0041560$, $\Delta^2 y_{n-1}=-0{,}0000401$, $\Delta^3 y_{n-3}=0{,}0000008$. Для $x=1044$ получаем отрицательное значение $t=-0{,}6$. По формуле (13.75)

$$F(1044)=3{,}0211893+(-0{,}6)\cdot 0{,}004\,15160+$$
$$+\frac{(-0{,}6)(-0{,}6+1)}{2}(-0{,}0000401)+$$
$$+\frac{(-0{,}6)(-0{,}6+1)(-0{,}6+2)}{2\cdot 3}\,0{,}0000008.$$

Произведя вычисления, получим

$$F(1044)=3{,}018\,7005.$$

Проверка по семизначным таблицам логарифмов показывает, что все семь знаков верные.

§ 76. Экстраполяция. Обратная интерполяция

В предыдущем параграфе были приведены примеры применения интерполяционных формул для отыскания значений функции, соответствующих промежуточным значениям аргумента, отсутствующим в таблице.

Эти же формулы могут быть использованы и для нахождения значений функции, соответствующих значениям аргумента, находящимся вне пределов таблицы, т. е. для *экстраполяции*.

Применение интерполяционных формул для экстраполяции ничем не отличается от рассмотренного в предыдущих примерах. Единственным различием является то, что при интерполяции по первой формуле Ньютона значение t оказывается положительным, а при экстраполяции — отрицательным. Для второй формулы Ньютона, наоборот, при интерполяции значение t отрицательно, а при экстраполяции — положительно.

Таблица 1.76

(1)	(2)	(3)	(4)	(5)
x	$\cos x$	Δ	Δ^2	Δ^3
65° 00′	0,4226 1826			
		—52 7981		
65° 20′	0,4173 3845		—1412	
		—52 9393		17
65° 40′	0,4120 4452		—1395	
		—53 0788		19
66° 00′	0,4067 3664		—1376	
		—53 2164		17
66° 20′	0,4014 1500		—1359	
		—53 3523		18
66° 40′	0,3960 7977		—1341	
		—53 4864		
67° 00′	0,3907 3113			

Таким образом, *первая интерполяционная формула Ньютона применяется для интерполирования вперед и экстраполирования назад, а вторая — для интерполирования назад и экстраполирования вперед.*

Пример 1.76. Дана таблица значений функции

$$y = \cos x$$

с шагом 20′ с точностью до восьми знаков в пределах от 65° до 67° (табл. 1.76). Вычислить значения $\cos x$ для углов 64°40′ до 65° с шагом 5′.

Разности мы записали в диагональную таблицу в той же табл. 1.76. Как показывают вычисления, уже третьи разности можно считать практически постоянными.

Вычисления экстраполированных значений приведены в табл. 2.76. В той же таблице в последней колонке справа приведены точные значения косинуса для соответствующих углов, взятые из более подробной таблицы. Сравнение этих значений со значениями, полученными экстраполяцией, показывает, что уже при экстраполяции больше чем на половину шага таблицы ошибка составляет две единицы последнего (восьмого) знака функции.

Таблица 2.76

(1)	(2)	(3)	(4)
t	$\frac{t(t-1)}{2}$	$\frac{t(t-1)(t-2)}{6}$	(1) Δ
—0,25	0,15625	—0,117 1875	13 1995 2
—0,50	0,37500	—0,312 5000	26 3990 4
—0,75	0,65625	—0,546 8750	39 5985 6
—1	1	—1	52 7981 0
(5)	(6)	(7)	(8)
(2) Δ^2	(3) Δ^3	$y=y_0+(4)+(5)+(6)$	$y=\cos x$
—220 6	—2 0	0,4239 3599	0,4239 3599
—529 5	—5 3	0,4252 5282	0,4252 5281
—926 6	—9 3	0,4265 6876	0,4265 6874
—1412 0	—17 0	0,4278 8378	0,4278 8376

Экстраполяция по второй интерполяционной формуле Ньютона производится точно так же. Что касается формулы Лагранжа, то для вычисления значения функции при любом x (т. е. и для интерполирования, и для экстраполирования) требуется только подставить в формулу (4.75) соответствующее значение аргумента.

Отметим, что экстраполяция, вообще говоря, дает бо́льшие ошибки, нежели интерполяция, и пределы ее применения ограничены.

До сих пор рассматривались лишь задачи нахождения значений функции, соответствующих данным значениям аргумента, отсутствующим в таблице. Между тем нередко приходится сталкиваться и с задачами иного, обратного характера: *по таблице функции отыскать значение аргумента x, которому соответствует данное значение функции, отсутствующее в таблице.* Так поставленную задачу называют *задачей обратной интерполяции.*

Задачу обратной интерполяции можно легко обратить, считая значения функции, наоборот, значениями аргумента. Однако так как разности функции не постоянны, то обратная интерполяция приводит к необходимости интерполировать в таблице, значения аргумента в которой не являются равноотстоящими. По этой причине для обратной интерполяции применяется обычно интерполяционная формула Лагранжа.

Таблица 3.76

x	$f(x)$
3,7	0,2160494
3,8	0,2129001
3,9	0,2098875
4,0	0,2070019
4,1	0,2042345

Таблица 4.76

x	$\varphi(x)$
0,212 9001	3,8
0,209 8875	3,9
0,207 0019	4,0

Пример 2.76. Функция $y=f(x)$ задана таблицей (табл. 3.76).

Определим, какому аргументу соответствует значение функции 0,2116793.

Как видно из табл. 3.76, значение аргумента заключено между 3,8 и 3,9. Поэтому задачу нахождения аргумента можно считать задачей интерполяции в таблице функции с переменным шагом, приведенной в табл. 4.76. Здесь мы поменяли местами функцию и аргумент.

Пользуясь формулой Лагранжа для $x=0{,}2116793$, находим

$$y=\frac{(0{,}2116793-0{,}2098875)(0{,}2116793-0{,}2070019)}{(0{,}2129001-0{,}2098875)(0{,}2129001-0{,}2070019)}\cdot 3{,}8+$$

$$+\frac{(0{,}2116793-0{,}2129001)(0{,}2116793-0{,}2070019)}{(0{,}2098875-0{,}2129001)(0{,}2098875-0{,}2070019)}\cdot 3{,}9+$$

$$+\frac{(0{,}2116793-0{,}2129001)(0{,}2116793-0{,}2098875)}{(0{,}2070019-0{,}2129001)(0{,}2070019-0{,}2098875)}\cdot 4{,}0=3{,}8400.$$

Таким образом, значение $f(x)=0{,}2116793$ следует считать соответствующим значению аргумента $x=3{,}8400$.

Более просто получить тот же результат, пользуясь вычислениями по схеме Эйткина. Рекомендуем читателю проделать соответствующие вычисления самостоятельно. Для их контроля приводим значения многочленов первой степени

$$P_{0,1}=3{,}8405 \text{ и } P_{1,2}=3{,}8379.$$

§ 77. Программирование прямой и обратной интерполяции

Из рассмотренных в предыдущих параграфах интерполяционных формул и схем наиболее удобной для программирования является схема Эйткина. Как показывают формулы (1.74), (3.74) и (4.74), линейная, квадратичная и кубическая интерполяции производятся по схеме Эйткина единообразно, причем для интерполяции следующего порядка используются многочлены, полученные на предыдущем шаге. Это позволяет строить интерполяцию любого порядка в виде двойного цикла, в котором внутренний цикл подсчитывает все нужные интерполяционные многочлены данного порядка, а внешний ведется по порядку интерполяции.

Пусть заданы две группы по $n+1$ ячеек, в которых записаны подряд значения $x_0, x_1, \ldots, x_n$ и $y_0, y_1, \ldots, y_n$. Степень интерполяции n записана в ячейках $(0, n, 0)$ и $(n, 0, 0)$ в виде числа единиц адресов. Предположим, что в ячейке α лежит требуемое значение аргумента; составим программу, которая вычислит интерполированное значение функции и положит его в ячейку γ.

Из формул (1.74) и (2.74) видно, что роль многочленов нулевого порядка играют значения функции $y_0, y_1, \ldots, y_n$. Поэтому работу программы нужно начать с пересылки этих значений на рабочее поле, отведенное для счета многочленов. Это будут ячейки $P_0, P_1, \ldots, P_n$. Пересылку можно осуществить с помощью трех команд, пользуясь регистром адреса

1) $[PA]$	$(0, n, 0)$	
2)	y_0^*	$=P_0^*$
3) $PA\geqslant\overline{1}$	2)	F^*.

Напишем теперь цикл для вычисления многочленов $P_{0,1}(\alpha)$ $P_{1,2}(\alpha), \ldots, P_{n-1,n}(\alpha)$ по формулам типа (1.74). Многочленов первой степени будет уже не $n+1$, а только n. При переходе ко второй степени число многочленов снова уменьшится на единицу. Поэтому можно организовать внешний цикл по счетчику ν, который будет при каждом шаге уменьшаться на единицу. Вычисление многочленов первой

степени можно организовать так:

4) $(n,\ 0,\ 0)-,\ (1,\ 0,\ 0)=\nu$
5) PA 0
6) x_0^* — α $=R_0$
7) x_1^* — α $=R_1$
8) P_0^* · R_1 $=R_2$
9) P_1^* · R_0 $=R_3$
10) R_2 — R_3 $=R_4$
11) x_1^* — x_0^* $=R_5$
12) R_4 : R_5 $=P_0^*$
13) $PA<\overline{n-1}$ 6) $\overline{1}^*$.

Команды 5) — 13) образуют внутренний цикл нашей программы. При переходе к следующему шагу внешнего цикла, при новом обращении к внутреннему нужно уменьшить значение ν и переадресовать команды 7) и 11), увеличив их левые адреса на единицу:

14) ν —, $(1,\ 0,\ 0)=\nu$
15) 7) +, $(1,\ 0,\ 0)=$ 7)
16) 11) +, $(1,\ 0,\ 0)=11)$.

Команду 13) проверки окончания цикла можно сформировать из заготовки $13)_0$ $PA<0$ 6) $\overline{1}^*$ таким образом:

17) $13)_0+,\ \nu=13)$.

Обозначив переадресуемые команды 7) и 11) через T_1, T_2, а формируемую команду 13) через U, организуем внешний цикл следующим образом:

а) $(n,\ 0,\ 0)-,\ (1,\ 0,\ 0)=\nu$
б) T_1^0 $=T_1$
в) T_2^0 T_2
г) T_1 +, $(1,\ 0,\ 0)=T_1$
д) T_2 +, $(1,\ 0,\ 0)=T_2$
е) U^0 +, $\nu=U$
ж) *Внутренний цикл*
з) ν —, $(1,\ 0,\ 0)=\nu$
и) $У_0$ P_0 г) γ
к) *стоп*

Здесь команда и), одновременно с передачей управления, пересылает ответ в ячейку γ. Программа будет работать верно для любого n, кроме случая $n=0$, который необходимо исключить.

Окончательно программу интерполяции можно записать в виде

$[PA]$			, $(0,\ n,\ 0)$		
			y_0^*	$=P_0^*$	
$PA\geqslant$	$\overline{1}$		$Я-1$	F^*	
	$\overline{14}$	$\leftarrow$,	$(0,\ n,\ 0)$	$=(n,\ 0,\ 0)$	
	$(n,\ 0,\ 0)$	—,	$(1,\ 0,\ 0)$	$=\nu$	
			T_1^0	$=T_1$	
$Б$	T_2^0		Δ	T_2	
∇	T_1	+,	$(1,\ 0,\ 0)$	$=T_1$	
	T_2	+,	$(1,\ 0,\ 0)$	$=T_2$	
	U^0	+,	ν	$=U$	
PA			0		
Δ	x_0^*	—	α	$=R_0$	
T_1	$(x_1^*$	—	α	$=R_1)$	н. п.
	P_0^*	·	R_1	$=R_2$	
	P_1^*	·	R_0	$=R_3$	
	R_2	—	R_3	$=R_4$	
T_2	$(x_1^*$	—	x_0^*	$=R_5)$	н. п.
	R_4	:	R_5	$=P_0^*$	
$U(PA<\overline{n-1}$				$\overline{1}^*)$	н. п.
	ν	—,	$(0,\ 1,\ 0)$	$=\nu$	
$У_0$	P_0		∇	γ	
	стоп				
T_1^0	x_1^*	—	α	$=R_1$	
T_2^0	x_1^*	—	x_0^*	$=R_5$	
U_0	$PA<0$		∇	$\overline{1}^*$.	

Та же программа пригодна и для экстраполяции, поскольку никаких ограничений на значение аргумента в ней не содержится. Программа вычисляет значение многочлена n-й степени для любого x.

Наоборот, в некоторых случаях нужно, чтобы программа не считала значений функции для аргументов, выходящих за пределы данной таблицы. Такая надобность возникает, например, при работе с табличной функцией (см. следующий параграф,). Тогда в программе надо предусмотреть специальную проверку.

Обратная интерполяция, т. е. нахождение промежуточного значения аргумента по значению функции, может быть выполнена с помомощью той же программы, в которой, однако, требуется поменять местами значения x и y и заслать в ячейку α заданное значение функции. Программа выдаст в ячейке γ соответствующее значение аргумента.

В качестве примера обратной интерполяции рассмотрим программирование нахождения корня способом проведения параболы, который разбирался в § 56. Как мы теперь видим, это есть обычная задача обратной интерполяции: даны значения y_0, y_1, y_2 функции $f(x)$ в точках x_0, x_1, x_2; требуется найти такую точку x, в которой значение функции равно нулю. Поскольку теперь степень интерполяции мала и значение функции равно нулю, т. е. его не требуется вычислять, то можно обойтись без двойного цикла. Нужно только предположить, как и раньше, что значения x_0, x_1, x_2 и y_0, y_1, y_2 расположены в ячейках памяти подряд.

Приближенное значение корня в ячейке γ можно получить с помощью такой программы:

$$
\begin{array}{ll}
PA & 0 \\
 & x_0^* \cdot y_1^* = R_0 \\
 & x_1^* \cdot y_0^* = R_1 \\
 & R_0 - R_1 = R_2 \\
 & y_1^* - y_0^* = R_3 \\
 & R_2 : R_3 = P_0^* \\
PA < & \overline{1} \qquad \overline{1}^* \\
 & P_0 \cdot y_2 = R_0 \\
 & P_1 \cdot y_0 = R_1 \\
 & R_0 - R_1 = R_2 \\
 & y_2 - y_0 = R_3 \\
 & R_2 : R_3 = \gamma \\
 & \textit{стоп}
\end{array}
$$

При фактическом нахождении корня эту программу следует включить

в качестве блока в цикл, который будет находить аргументы x_0, x_1, x_2 и соответствующие значения y_0, y_1, y_2 в зависимости от заданной функции. Внешний цикл строится подобно тому, как это делалось в программах § 61—62.

§ 78. Программа работы с табличной функцией

В практике вычислений приходится иметь дело с функциями, которые на различных участках определяются различными формулами. В этом случае часто бывает выгодно задавать функцию не совокупностью формул, а таблицей значений в отдельных точках. Еще более важно задание функции таблицей значений, когда эти значения получены в результате эксперимента, так что аналитическое выражение функции может оказаться вообще неизвестным.

Если функция задана в виде таблицы значений, записанной в оперативной памяти машины, то для работы с такой функцией требуется иметь программу, работающую по образцу стандартных программ функций одного переменного. Задав в ячейке α аргумент и обратившись к этой программе, мы должны в ячейке γ получить соответствующее значение функции, либо взятое прямо из таблицы, либо, если это требуется, полученное путем интерполяции табличных значений. Такую программу называют *программой работы с табличной функцией;* ее составлению и посвящен настоящий параграф.

Пусть функция $y=f(x)$ определена на отрезке $[a, b]$ и задана в $n+1$ равноотстоящих точках отрезка $x_0=a$, $x_1=x_0+h$, $x_2=x_0+2h, \ldots, x_n=x_0+nh=b$. Предположим, что эти значения аргумента расположены в идущих подряд ячейках памяти $x_0, x_1, \ldots, x_n$, а соответствующие им значения функции — в идущих подряд ячейках памяти $y_0, y_1, \ldots, y_n$. Шаг таблицы будем считать настолько малым, чтобы для нахождения значений функции, соответствующих промежуточным значениям аргумента, было достаточно квадратичной интерполяции.

Нахождение значения функции по значению аргумента, находящемуся в ячейке α, состоит из двух этапов:

I) определение номера первого из трех табличных значений функции, нужных для интерполяции;

II) квадратичная интерполяция по найденным трем значениям.

Рассмотрим сначала программу, осуществляющую второй этап вычислений. Предположим, что требуемый в первом этапе вычислений номер i уже определен и записан в ячейке ν в виде числа единиц второго адреса, так что $\nu \equiv (0, i, 0)$. Интерполяцию будем осуществлять по первой интерполяционной формуле Ньютона (9.75), которая для случая квадратичной интерполяции имеет вид

$$y=F(x)=F(x_i+th)=y_i+t\Delta y_i+\frac{t(t-1)}{2}\Delta^2 y_i.$$

Нужные вычисления осуществляются программой:

Интерполяция

[PA]		ν.
α	$-x_0^*$	$=R_0$
R_0	$: h$	$=t$
y_1^*	$-y_0^*$	$=\Delta y_i$
y_2^*	$-y_1^*$	$=\Delta y_{i+1}$
Δy_{i+1}	$-\Delta y_i$	$=\Delta^2 y_i$
t	$\cdot \Delta y_i$	$=R_0$
y_0^*	$+R_0$	$=R_0$
t	$-$«1»	$=R_1$
t	$\cdot R_1$	$=R_1$
R_1	$\cdot$ «$\frac{1}{2}$»	$=R_1$
R_1	$\cdot \Delta^2 y_i$	$=R_1$
R_0	$+R_1$	$=\gamma$
	стоп.	

В этой программе первая команда заносит в регистр адреса номер l, который затем прибавляется к адресам встречающихся в программе ячеек x_0, y_0, y_1, y_2. Дальнейших пояснений к программе не требуется.

Заметим кстати, что при $\alpha = x_i$ мы получим $t = 0$, вследствие чего $\gamma = y_i$. Аналогично при $\alpha = x_{i+1}$ будем иметь $t = 1$ и $\gamma = y_{i+1}$, а при $\alpha = x_{i+2}$ получим $t = 2$ и $\gamma = y_{i+2}$. Таким образом, если заданное значение аргумента совпадает с одним из узлов интерполяции, то в качестве значения функции программа выдаст соответствующее табличное значение.

Перейдем теперь к рассмотрению первого этапа решения задачи — отысканию номера i первого из трех значений y, нужных для интерполяции. Для того чтобы приведенная формула при заданном значении α осуществляла интерполяцию, а не экстраполяцию, должно быть выполнено условие

$$x_i \leqslant \alpha < x_{i+1}.$$

Так как $x_i = x_0 + ih$, $x_{i+1} = x_0 + (i+1)h$, то отсюда вытекает, что

$$\frac{\alpha - x_0}{h} - 1 < i \leqslant \frac{\alpha - x_0}{h}.$$

Из последнего неравенства видно, что i есть целая часть числа $\frac{\alpha - x_0}{h}$:

$$i = \left[\frac{\alpha - x_0}{h}\right].$$

Выделение целой и дробной частей числа уже рассматривалось нами во второй части книги в примере 4.35. Там было замечено, что если произвести сдвиг мантиссы ячейки v по порядку этой же ячейки, т. е. выполнить команду

$$v\,[\rightarrow,]\,v = R,$$

то в ячейке R выделится мантисса дробной части числа, лежащая в адресной части. Целая часть при этом теряется, но если производить полный сдвиг, то она попала бы в кодовую часть и была бы записана непосредственно перед первым адресом. Чтобы сдвинуть целую часть в средний адрес, надо произвести еще один сдвиг на два адреса, т. е. на $24_{10} = 30_8$ разрядов вправо.

Поскольку сдвиг вправо является отрицательным, то можно сразу сдвинуть целую часть ячейки v во второй адрес, производя сдвиг мантиссы v не на порядок v, а на порядок, уменьшенный на 30_8. Таким образом, мы можем получить величину i во втором адресе ячейки ν с помощью команд

$$\begin{array}{llll} \alpha & — & x_0 & = u \\ u & : & h & = v \\ v & , — & (30,\ 0,\ 0,\ 0) & = R_0 \\ R_0\,[\rightarrow,] & & v & = \nu. \end{array}$$

При этом в кодовой части ячейки ν останется порядок числа $v = \dfrac{x - x_0}{h}$, а в третьем адресе — начало мантиссы дробной части. Однако для операции взятия в регистр содержимое кодовой части не играет роли, а ячейка ν ни для чего другого не используется.

Мы не можем пока считать задачу окончательно решенной, так как необходимо рассмотреть еще два исключительных случая: 1) когда заданное значение аргумента выходит за пределы отрезка $[a, b]$, являющегося областью определения функции или 2) когда заданный аргумент попадает между последними узлами интерполяции. В первом случае ($\alpha < x_0$ либо $\alpha > x_n$) условимся засылать в ячейку γ вместо ответа число *Щ* (777, *F*, *F*, *F*) и выходить на *стоп*. Во втором случае, когда аргумент α удовлетворяет неравенству

$$x_{n-1} \leqslant \alpha < x_n,$$

в качестве i в среднем адресе ячейки ν появится число $n - 1$. Вычисления с таким значением i невозможны, так как справа от точки x_i есть тогда лишь одно, а не два значения функции. Поэтому значение i нужно в этом случае уменьшить на единицу.

Таким образом, с учетом всех указанных возможностей программу для осуществления первого этапа можно записать в виде

1)	α —	x_0	$=u$
2)	$У_1$ *Щ*	*стоп*	γ
3)	u :	h	$=v$
4)	v , —	(30, 0, 0, 0)	$=R_0$
5)	$R_0[\rightarrow,]$	v	$=\nu$
6)	ν —,	(0, n, 0)	$=R_0$
7)	$У_0$	*стоп*	
8)	R_0+,	(0, 1, 0)	$=0$
9)	$У_0$	*Интерполяция*	
10)	ν —,	(0, 1, 0)	$=\nu$.

Если $\alpha < x_0$, то в результате команды 1) выработается сигнал $\omega = 1$ и команда 2), заслав *Щ* в γ, передаст управление на *стоп*. В противном случае команды 3) — 5) вычислят величину i во втором адресе ячейки ν. При $i \geqslant n$, что означает $\alpha \geqslant x_n$, команда 6) выработает сигнал $\omega = 1$ и команда 7) передаст управление на *стоп*. При $i < n$ в левом адресе ячейки R_0 получится F, а в среднем F, если $i = n - 1$, и число, меньшее F, если $i < n - 1$.

Команда 8), прибавляя к R_0 единицу среднего адреса, проверяет это обстоятельство. Если $i < n - 1$, то в среднем адресе R_0 мы имеем число, меньшее F, прибавление единицы не переполняет мантиссу R_0 и команда 8) вырабатывает сигнал $\omega = 0$. Тогда в ячейке ν лежит нужное значение i и команда 9) передает уравнение на интерполяцию. Если же $i = n - 1$, то в среднем адресе R_0 находится F и прибавление единицы переполняет мантиссу R_0, так что вырабатывается сигнал $\omega = 1$. Поэтому команда 9) передает управление команде 10), которая уменьшает i на единицу. Так как команда 10) является командой с принудительным управлением и передает управление следующей ячейке, то в ней должна начинаться программа интерполяции либо находиться команда безусловной передачи управления программе интерполяции.

В составленную программу можно внести еще одно уточнение. При нахождении тройки табличных значений y_i, y_{i+1}, y_{i+2}, которые брались для интерполяции, мы выбирали i таким образом, чтобы

$$x_i \leqslant \alpha < x_{i+1}.$$

Однако очевидно, что если α ближе к x_i, чем к x_{i+1}, то квадратичная интерполяция будет точнее, если вместо значений, указанных выше, брать значения y_{i-1}, y_i, y_{i+1}, т. е. если уменьшить i на единицу, как и для последнего участка.

Если α ближе к x_i, чем к x_{i+1}, то дробная часть числа $v = \frac{\alpha - x_0}{h}$ будет меньше половины, в противном случае — больше. Поскольку начало дробной части числа v размещается в третьем адресе ячейки ν, то об этом можно судить по первой цифре этого адреса, т. е. по двенадцатому разряду ячейки ν. Если в двенадцатом разряде стоит нуль, то дробная часть v меньше половины и α ближе к x_i, а если единица, то дробная часть v больше половины и α ближе к x_{i+1}. В первом случае i надо уменьшить на единицу (если $i > 0$; при $i = 0$, т. е. $v < 1$, значение i не изменяется), во втором — оставить без изменения. Проверку можно осуществить высечением двенадцатого разряда ячейки ν, что достигается командой

$$\nu \wedge (0,\ 0,\ 4000) = 0$$

и последующей условной передачей управления. При этом удобно также несколько изменить команды 8) — 10).

С внесенными изменениями программа работы с табличной функцией будет выглядеть так:

	α —		x_0	$= u$
$У_1$	*Щ*		*стоп*	γ
	u :		h	$= v$
$У_0$	0 *Интерполяция*			ν
	v	—,	(30, 0, 0, 0,)	$= R_0$
	R_0	[→,]	v	$= \nu$
	ν	—,	(0, n, 0)	$= R_0$
$У_0$		*стоп*		
	R_0	+,	(0, 1, 0)	$= 0$
$У_1$				
	ν	$\wedge$	(0, 0, 4000)	$= 0$
$У_0$				
	ν	—,	(0, 1, 0)	$= \nu$
Интерполяция [*PA*]			ν	
	α	—	x_0^*	$= R_0$
	R_0	:	h	$= t$
	y_1^*	—	y_0^*	$= \Delta y_i$
	y_2^*	—	y_1^*	$= \Delta y_{i+1}$
	Δy_{i+1}	—	Δy_i	$= \Delta^2 y_i$
	t	·	Δy_i	$= R_0$
	y_0^*	+	R_0	$= R_0$
	t	—	«1»	$= R_1$
	t	·	R_1	$= R_1$
	R_1	·	«$\frac{1}{2}$»	$= R_1$
	R_1	·	$\Delta^2 y_i$	$= R_1$
	R_0	+	R_1	$= \gamma$
	стоп.			

ГЛАВА XV

ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

§ 79. Формулы прямоугольников и трапеций

Формулы для приближенного вычисления определенных интегралов применяются очень часто. Дело в том, что для большого числа элементарных функций первообразные уже не выражаются через элементарные функции, в результате чего нельзя вычислить определенный интеграл с помощью формулы Ньютона — Лейбница. Встречаются также и случаи, когда приходится прибегать к формулам приближенного интегрирования даже для таких интегралов, которые могут быть найдены в конечном виде, но такое выражение оказывается слишком сложным. Особенно важны формулы приближенного интегрирования при решении задач, содержащих функции, заданные таблично.

Наиболее простыми формулами для численного интегрирования являются формулы *прямоугольников* и *трапеций*. Вывод их основан на использовании геометрического смысла определенного интеграла, выражающего, как известно, площадь криволинейной трапеции.

Формула прямоугольников, собственно, есть не что иное, как интегральная сумма, составленная с учетом некоторых дополнительных предположений, впрочем, совершенно естественных.

Пусть требуется вычислить интеграл $\int_a^b f(x)\,dx$. Разобьем участок интегрирования $[a, b]$ на n равных частей и поместим точки, значения функции в которых входят в интегральную сумму, в левых концах полученных участков. Если считать, что n достаточно велико, т. е. длина участков разбиения $h = \frac{b-a}{n}$ достаточно мала, то интегральная сумма должна уже мало отличаться от величины интеграла. Таким образом, мы получаем приближенное равенство

$$\int_a^b f(x)\,dx \approx h(y_0 + y_1 + \ldots + y_{n-1}) = h\sum_{k=0}^{n-1} y_k, \qquad (1.79)$$

которое и является *формулой прямоугольников*. Здесь, как и всюду в дальнейшем, через $y_0, y_1, \ldots, y_n$ обозначены значения функции $y=f(x)$ в точках деления $x_0, x_1, \ldots, x_n$.

Аналогичная формула прямоугольников получится и в том случае, если брать для интегральной суммы значения функции не в левых, а в правых концах участков разбиения. Тогда формула примет вид

$$\int_a^b f(x)\,dx \approx h(y_1+y_2+\ldots+y_n)=h\sum_{k=1}^{n} y_k. \qquad (2.79)$$

Для функции, монотонной на отрезке интегрирования, всякая интегральная сумма, а значит, и определенный интеграл заключены между приближенными значениями, указанными в правых частях формул (1.79) и (2.79). Геометрическую иллюстрацию этого факта можно видеть на рис. 60, где взята возрастающая функция и слагаемые, входящие в различные суммы, показаны пунктиром и штриховкой. Благодаря этому представление о погрешностях формулы прямоугольников можно получить, рассматривая разность результатов, полученных по формулам (1.79) и (2.79).

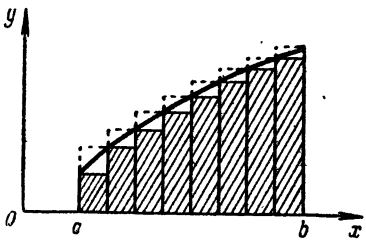


Рис. 60.

Если функция имеет на отрезке интегрирования конечное число экстремумов, то отрезки интегрирования можно разбить на участки монотонности и получить, таким образом, оценку погрешности формулы прямоугольников.

Пример 1.79. Вычислить по формулам прямоугольников интеграл $\int_0^1 \frac{dx}{1+x}$, разбив участок интегрирования на 10 частей.

Значения функции приведены в табл. 1.79. Применяя формулы (1.79) и (2.79), находим в первом случае

$$\int_0^1 \frac{dx}{1+x}=0{,}1\cdot(1{,}0000+0{,}9091+\ldots+0{,}5263)=0{,}7188,$$

и во втором

$$\int_0^1 \frac{dx}{1+x}=0{,}1\cdot(0{,}9091+0{,}8333+\ldots+0{,}5000)=0{,}6688.$$

Как известно, истинным значением этого интеграла является число $\ln 2=0{,}6931\ldots$, которое действительно заключено между двумя

полученными приближениями. Зная это истинное значение, мы можем определить относительные погрешности приближенных значений интеграла; они равны соответственно $\delta_1 = 3{,}7\%$ и $\delta_2 = 3{,}5\%$. Но и имея только два вычисленных приближения, можно утверждать, что абсолютная погрешность каждого из них не превосходит $\Delta = 0{,}05$.

Таблица 1.79

(1)	(2)	(3)
x	«1»+(1)	y=«1» : (2)
0	1,0	1,0000
0,1	1,1	0,9091
0,2	1,2	0,8333
0,3	1,3	0,7692
0,4	1,4	0,7143
0,5	1,5	0,6667
0,6	1,6	0,6250
0,7	1,7	0,5882
0,8	1,8	0,5556
0,9	1,9	0,5263
1,0	2,0	0,5000

Легко заметить, что среднее арифметическое полученных по формулам прямоугольников приближений равно 0,6938, т. е. довольно точно совпадает уже с истинным значением интеграла: относительная погрешность равна здесь $0{,}1\%$.

Это наводит на мысль принимать в качестве приближенного значения интеграла среднее арифметическое приближений, полученных по формулам (1.79) и (2.79).

Заметим, что нет никакой нужды вычислять предварительно эти значения, так как можно сразу воспользоваться готовой формулой. Действительно, взяв среднее арифметическое правых частей формул (1.79) и (2.79), мы получим

$$\int_a^b f(x)\,dx \approx h\left(\frac{1}{2}y_0 + y_1 + \ldots + y_{n-1} + \frac{1}{2}y_n\right) = h\left(\frac{y_0}{2} + \sum_{k=1}^{n-1} y_k + \frac{y_n}{2}\right). \quad (3.79)$$

Это и есть *формула трапеций.*

Формулу трапеций (3.79) можно легко получить и непосредственно, исходя из ее геометрического смысла. Разобьем отрезок интегрирования на n равных частей точками $a = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = b$, проведем ординаты во всех точках деления и заменим каждую из полученных криволинейных трапеций прямолинейной, как это показано на рис. 61. Сторонами каждой из этих трапеций являются две соседние ординаты, участок оси Ox, длина которого $h = \frac{b-a}{n}$, и хорда кривой.

Площадь самой левой трапеции равна $\Delta s_1 = \frac{y_0 + y_1}{2} h$. Аналогично для трапеции, расположенной над участком (x_{i-1}, x_i), находим

$$\Delta s_i = \frac{y_{i-1} + y_i}{2} h. \quad (4.79)$$

Суммирование выражений (4.79) по всем i от 1 до n и приводит к формуле (3.79), так как все ординаты, кроме крайних, используются в выражениях вида (4.79) дважды.

Пример 2.79. Вычислим по формуле трапеций длину дуги параболы $y = x(1-x)$ между точками пересечения ее с осью Ox (рис. 62).

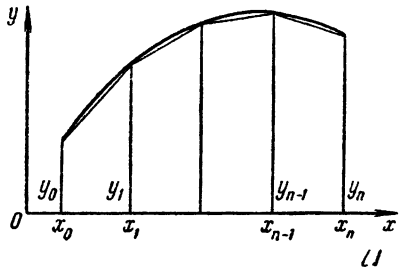


Рис. 61.

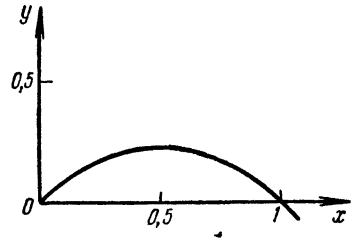


Рис. 62.

В интегральном исчислении доказывается, что длина дуги кривой $y = f(x)$ на участке $[a, b]$ выражается интегралом

$$l = \int_a^b \sqrt{1 + (y')^2}\, dx, \tag{5.79}$$

где $y' = f'(x)$ — производная функции $f(x)$. Парабола $y = x(1-x)$ пересекается с осью Ox в точках $x = 0$ и $x = 1$. Так как $y' = 1 - 2x$, то длина дуги выразится интегралом

$$l = \int_0^1 \sqrt{1 + (1-2x)^2}\, dx. \tag{6.79}$$

Разобьем отрезок интегрирования на четыре части и вычислим соответствующие значения функции (табл. 2.79). Пользуясь формулой трапеций (3.79), находим

$$l \approx 0{,}25\left(\frac{1{,}414}{2} + 1{,}118 + 1{,}000 + 1{,}118 + \frac{1{,}414}{2}\right) = 1{,}162.$$

Таблица 2.79

(1)	(2)	(3)	(4)	(5)	(6)
x	«2»·(1)	«1»—(2)	(3)^2	«1»+(4)	$f(x) = \sqrt{(5)}$
0	0	1	1	2,000	1,414
0,25	0,500	0,500	0,250	1,250	1,118
0,50	1,000	0	0	1,000	1,000
0,75	1,500	—0,500	0,250	1,250	1,118
1	2,000	—1	1	2,000	1,414

Интеграл (6.79) можно выразить через элементарные функции. Опуская все промежуточные выкладки, можем написать

$$l=\int_0^1 \sqrt{1+(1-2x^2)}\,dx=\left[\frac{2x-1}{4}\sqrt{4x^2-4x+2}+\right.$$

$$\left.+\frac{1}{4}\ln\left(4\sqrt{4x^2-4x+2}+8x-4\right)\right]_0^1=\frac{\sqrt{2}}{2}+\frac{1}{2}\ln(\sqrt{2}+1)=1{,}148.$$

Отсюда видно, что полученное по формуле трапеций приближенное значение длины дуги l дает абсолютную погрешность $\Delta l=0{,}014$ или относительную погрешность $\delta=1{,}2\%$. Вместе с тем вычисления, требуемые формулой трапеций, во много раз проще и короче, нежели вычисления по точной формуле.

Точность приближенного значения интеграла, полученного по формуле трапеций (3.79), можно увеличить, увеличивая число n участков разбиения отрезка интегрирования, хотя при этом, естественно, возрастает объем требуемых вычислений. Например, уже при $n=5$ и $h=0{,}2$ получаем в качестве приближенного значения по формуле трапеций $l=1{,}157$ с абсолютной погрешностью $\Delta=0{,}009$ и относительной — $0{,}8\%$, а при $n=10$ и $h=0{,}1$ уже $l=1{,}150$, так что абсолютная погрешность составляет только 0,002, а относительная $0{,}017\%$. Вычисления для этого последнего случая приведены в табл. 3.79. Даже и в этом последнем случае вычисления осуществляются проще и быстрее, нежели по точной формуле (включая нахождение неопределенного интеграла).

Таблица 3.79

(1)	(2)	(3)	(4)	(5)	(6)
x	«2»·(1)	«1»—(2)	(3)^2	«1»+(4)	$f(x)=\sqrt{(5)}$
0	0	1	1	2,00	1,414
0,1	0,2	0,8	0,64	1,64	1,281
0,2	0,4	0,6	0,36	1,36	1,166
0,3	0,6	0,4	0,16	1,16	1,077
0,4	0,8	0,2	0,04	1,04	1,020
0,5	1,0	0	0	1,00	1,000
0,6	1,2	—0,2	0,04	1,04	1,020
0,7	1,4	—0,4	0,16	1,16	1,077
0,8	1,6	—0,6	0,36	1,36	1,166
0,9	1,8	—0,8	0,64	1,64	1,281
1	2	—1	1	2,00	1,414

§ 80. Формула Симпсона

Формула Симпсона является более точной, нежели рассмотренная в предыдущем параграфе формула трапеций. Это означает, что для достижения той же точности в ней можно брать меньшее число n

участков разбиения и соответственно больший шаг h, а при одном и том же шаге h, т. е. при том же объеме вычислений *) она дает меньшие абсолютную и относительную погрешности.

Формулу Симпсона можно получить с помощью того же приема, который уже дважды применялся в § 79. Разобьем участок $[a, b]$ на четное число $n = 2m$ частей точками $a = x_0 < x_1 < \ldots < x_{n-1} < x_n = b$, обозначим ординаты в точках деления через $y_0, y_1, \ldots, y_n$ и рассмотрим пару соседних участков, например, с левым концом в точке $a = x_0$ (рис. 63). Проведем через три точки кривой с координатами (x_0, y_0), (x_1, y_1), (x_2, y_2) параболу с осью, параллельной оси Oy. Ее уравнение будет

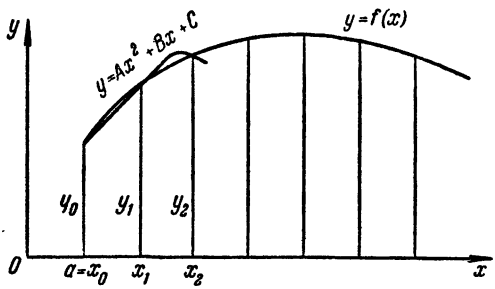


Рис. 63.

$$y = Ax^2 + Bx + C, \quad (1.80)$$

причем коэффициенты A, B, C остаются пока неизвестными **).

Заменив площадь заданной криволинейной трапеции на участке $[x_0, x_2]$ площадью криволинейной трапеции, ограниченной параболой (1.80), придем к приближенному равенству

$$\int_{x_0}^{x_2} f(x)\,dx \approx \int_{x_0}^{x_2} (Ax^2 + Bx + C)\,dx = \left[A\frac{x^3}{3} + B\frac{x^2}{2} + Cx\right]_{x_0}^{x_2} =$$

$$= A\frac{x_2^3 - x_0^3}{3} + B\frac{x_2^2 - x_0^2}{2} + C(x_2 - x_0).$$

Вынеся за скобку общий множитель $x_2 - x_0$ и приведя к общему знаменателю, получим

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{x_2 - x_0}{6}[2A(x_0^2 + x_0x_2 + x_2^2) + 3B(x_0 + x_2) + 6C]. \quad (2.80)$$

Неизвестные коэффициенты A, B, C в уравнении (1.80) и формуле (2.80) находятся из условия, что при значениях x, равных x_0, x_1, x_2, функция $f(x)$ принимает соответственно значения y_0, y_1, y_2.

*) Как видно из приведенных в § 79 примеров, почти вся вычислительная работа идет на нахождение значений функции. Поэтому объем вычислений определяется числом требуемых ординат.

**) Для получения этого уравнения можно воспользоваться любой из интерполяционных формул, рассмотренных в § 75. Тогда не возникнет задача нахождения неизвестных коэффициентов A, B, C. Тем не менее, приведенный ниже способ вывода проще, чем при использовании интерполяционных формул.

Заметив, что $x_1 = \frac{x_0 + x_2}{2}$, запишем эти условия в виде

$$\left.\begin{aligned} y_0 &= Ax_0^2 + Bx_0 + C, \\ y_1 &= A\left(\frac{x_0 + x_2}{2}\right)^2 + B\frac{x_0 + x_2}{2} + C, \\ y_2 &= Ax_2^2 + Bx_2 + C. \end{aligned}\right\} \tag{3.80}$$

Умножая второе равенство (3.80) на четыре и складывая после этого все три равенства (3.80), находим

$$y_0 + 4y_1 + y_2 = A[x_0^2 + (x_0 + x_2)^2 + x_2^2] + B[x_0 + 2(x_0 + x_2) + x_2] + 6C = 2A(x_0^2 + x_0x_2 + x_2^2) + 3B(x_0 + x_2) + 6C, \tag{4.80}$$

что совпадает с квадратной скобкой в правой части равенства (2.80).

Подставив (4.80) в правую часть равенства (2.80) и заметив, что $x_2 - x_0 = 2h$, где $h = \frac{b-a}{n}$, придем к приближенному равенству

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}(y_0 + 4y_1 + y_2). \tag{5.80}$$

Ясно, что для каждой следующей пары участков получается такая же формула

$$\int_{x_2}^{x_4} f(x)\,dx \approx \frac{h}{3}(y_2 + 4y_3 + y_4) \tag{6.80}$$

. . . . . .

Суммируя равенства вида (5.80) и (6.80) по всем участкам, получим формулу

$$\int_a^b f(x)\,dx \approx \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \ldots + 2y_{2m-2} + 4y_{2m-1} + y_{2m}). \tag{7.80}$$

Формула (7.80) и есть нужная нам *формула Симпсона*. Учитывая геометрический смысл формулы, ее называют также *формулой парабол*. В ней все ординаты с нечетными номерами умножаются на четыре, а все ординаты с четными номерами (кроме крайних) — на два. Крайние ординаты y_0 и y_{2m} входят в формулу с коэффициентами единица.

Пример 1.80. Вычислим $\int_0^1 \frac{dx}{1+x^2}$ способом трапеций и парабол, разбив участок (0,1) на 10 частей, т. е. приняв $h = 0{,}1$. Вычисление значений функции приведено в табл. 1.80.

Таблица 1.80

(1)	(2)	(3)	(4)
x	$(1)^2$	«1»+(2)	$f(x) =$ «1» : (3)
0,0	0,00	1,00	1,000 0000
0,1	0,01	1,01	0,990 0990
0,2	0,04	1,04	0,961 5385
0,3	0,09	1,09	0,917 4312
0,4	0,16	1,16	0,862 0690
0,5	0,25	1,25	0,800 0000
0,6	0,36	1,36	0,735 2941
0,7	0,49	1,49	0,671 1409
0,8	0,64	1,64	0,609 7561
0,9	0,81	1,81	0,552 4862
1,0	1,00	2,00	0,500 0000

Просуммировав полученные значения функции с соответствующими коэффициентами, получим:

по формуле трапеций

$$\int_0^1 \frac{dx}{1+x^2} \approx 0{,}1\left(\frac{1{,}000\,000}{2} + 0{,}9900990 + \ldots + 0{,}5524862 + \right.$$
$$\left. + \frac{0{,}500\,0000}{2}\right) = 0{,}7849815;$$

по формуле Симпсона

$$\int_0^1 \frac{dx}{1+x^2} \approx \frac{0{,}1}{3}(1{,}000\,0000 + 4 \cdot 0{,}990\,0990 + 2 \cdot 0{,}961\,5385 + \ldots +$$
$$+ 4 \cdot 0{,}5524862 + 0{,}500\,0000) = 0{,}785\,4648.$$

Рассматриваемый интеграл равен

$$\int_0^1 \frac{dx}{1+x^2} = \operatorname{arctg} x \Big|_0^1 = \frac{\pi}{4};$$

поэтому можно считать, что, вычисляя этот интеграл, мы находим приближенное значение числа $\frac{\pi}{4}$. Так как истинное значение $\frac{\pi}{4} = 0{,}78539816$, то относительная погрешность при пользовании методом трапеций составляет $0{,}04\%$, а при пользовании методом парабол — $0{,}009\%$.

Пример 2.80. Вычислим с помощью формулы Симпсона длину дуги параболы из примера 2.79, разбивая отрезок интегрирования на четыре части.

Беря значения функции из табл. 2.79 и пользуясь формулой (7.80), находим

$$l=\frac{0{,}25}{3}(1{,}414+4\cdot 1{,}118+2\cdot 1{,}000+4\cdot 1{,}118+1{,}414)=1{,}148.$$

Таким образом, формула Симпсона уже при $n=4$ дает для длины дуги четыре значащие цифры точными.

Пример 3.80. Вычислим с помощью формулы Симпсона интеграл

$$\int_0^2 \frac{dx}{\sqrt{1+x^2+x^4}},$$

разбивая отрезок интегрирования на восемь частей.

Значения функции вычислены в табл. 2.80. Значение интеграла по формуле Симпсона равно

$$\int_0^2 \frac{dx}{\sqrt{1+x^2+x^4}}=\frac{0{,}25}{3}(1+4\cdot 0{,}9683+2\cdot 0{,}9053+\ldots+0{,}2182)=$$

$$=1{,}2069.$$

Таблица 2.80

(1)	(2)	(3)	(4)	(5)	(6)
x	$(1)^2$	$(2)^2$	«1»+(2)+(3)	$\sqrt{(4)}$	$f(x)=$ «1» : (5)
0	0	0	1	1	1
0,25	0,0625	0,0039	1,0664	1,0327	0,9683
0,50	0,2500	0,0625	1,3125	1,1456	0,8729
0,75	0,5625	0,3164	1,8789	1,3707	0,7296
1,00	1,0000	1,0000	3,0000	1,7321	0,5773
1,25	1,5625	2,4414	5,0039	2,2369	0,4470
1,50	2,2500	5,0625	8,3125	2,8831	0,3468
1,75	3,0625	9,3789	13,4414	3,6662	0,2728
2,00	4,0000	16,0000	21,0000	4,5826	0,2182

Поскольку точное значение интеграла нам в этом случае неизвестно, то указать возможную погрешность полученного результата мы сейчас не можем.

§ 81. Проверка точности результатов численного интегрирования

Возможность оценить точность результатов, полученных по формулам численного интегрирования, имеет очень большое значение. Действительно, в примерах, рассматривавшихся в предыдущих параграфах (кроме примера 3.80), мы имели возможность сравнить полученные

по приближенным формулам значения интегралов с точными значениями. Однако практический интерес представляют, конечно, случаи, когда точное значение интеграла остается неизвестным, как это имело место в примере 3.80.

В рассмотренных примерах мы заранее задавались числом частей n, на которые разбивался участок интегрирования, что определяло уже выбор шага. Естественно возникает вопрос — какова точность полученного при этом приближения? Другой, тесно связанный с ним и наиболее практически важный вопрос — каким нужно выбрать шаг и число частей n, чтобы получить значение интеграла с погрешностью, не превосходящей заданного предела?

Вопросы эти весьма сложны и дать ответ на них в общем виде не представляется возможным. Если речь идет о функциях, заданных аналитическими выражениями, для которых легко получить и оценить старшие производные, то для оценки точности формул трапеций и парабол можно воспользоваться выражением соответствующих остаточных членов, равных разности между интегралом и его приближенным значением.

Остаточный член формулы трапеций имеет вид *)

$$R = -\frac{(b-a)^3}{12n^2} M_2. \tag{1.81}$$

Здесь $(b-a)$ — длина участка интегрирования, n — число частей, на которые разбит участок, и M_2 — наибольшее по абсолютной величине значение второй производной $f''(x)$ на рассматриваемом участке (взятое, тем не менее, со своим знаком).

Мы не будем останавливаться на выводе этой формулы, а ограничимся тем, что проанализируем влияние всех входящих в нее членов. Длина участка интегрирования входит в числитель формулы (1.81). Из этого следует, что чем больше участок, тем больше погрешность формулы трапеций и, наоборот, при уменьшении участка интегрирования погрешность падает. Это обстоятельство вполне естественно. Столь же естественно и то, что число n частей разбиения входит в знаменатель: при увеличении n точность формулы трапеций возрастает, так что погрешность быстро падает.

Легко уяснить себе также зависимость погрешности формулы трапеций от величины второй производной. Прежде всего, если $f''(x) \equiv 0$, то формула трапеций дает точный результат, поскольку погрешность обращается в нуль. Но вторая производная обращается тождественно в нуль лишь для линейной функции. Для нее формула трапеций и должна дать точный результат, поскольку она основана на замене функции линейной. Вторая производная характеризует

*) Это выражение представляет собой предельную абсолютную погрешность (см. § 1).

кривизну графика функции, которая является мерой отклонения кривой от прямой линии. Поэтому ясно, что чем меньше вторая производная, тем меньше график функции $f(x)$ отличается от прямой линии и тем меньшую погрешность мы будем получать при интегрировании по формуле трапеций, заменяя функцию линейной.

Так же просто объяснить и знак минус в формуле (1.81). В самом деле, если график функции на рассматриваемом участке является выпуклым вверх, то хорды кривой, как это видно на рис. 61, лежат ниже кривой, и формула трапеций дает значение, меньшее истинного, и, значит, погрешность этой формулы положительна. Вместе с тем вторая производная в этом случае отрицательна. Наоборот, для функции с графиком, выпуклым вниз, вторая производная положительна, а хорды проходят выше кривой, и формула трапеций имеет отрицательную погрешность. Таким образом, знак погрешности в обоих случаях противоположен знаку второй производной.

Практическое значение самой формулы (1.81) невелико, так как оценить величину M_2 удается далеко не всегда, особенно в тех случаях, когда функция задана таблицей. Однако в некоторых простых случаях удается получить довольно хорошую оценку погрешности, используя следующий прием.

Увеличим шаг формулы трапеций вдвое, т. е. уменьшим вдвое n (при этом первоначальное значение n должно быть четным), и найдем значение интеграла в этом случае, что очень просто, поскольку вычисления новых значений функции не требуется. Так как n входит в знаменатель в квадрате, то уменьшение n вдвое увеличит остаточный член вчетверо, и если разность между приближенным и истинным (неизвестным) значением интеграла в первом случае составляет R, то во втором это будет $4R$. Если считать, что оба приближенных значения отклоняются от истинного в одну и ту же сторону, то разность между двумя приближенными значениями, которую можно найти, не зная точного значения, равна утроенной ошибке приближения, полученного с первоначальным n. Это дает возможность оценить порядок погрешности полученного приближения.

Возвратимся к примеру 2.79 и оценим таким способом погрешность приближения, полученного при $n=10$. По табл. 3.79 мы получили значение $I=1{,}150$, а при $n=5$ соответствующее значение интеграла $I=1{,}157$. Разность между этими значениями 0,007, как было сказано выше, равна утроенной ошибке, поэтому можно считать, что абсолютная погрешность значения $I=1{,}150$ не превышает 0,003, что, как мы знаем, дает вполне точное представление о действительной величине погрешности.

Аналогичная формула имеет место и для формулы Симпсона. Именно, остаточный член формулы парабол имеет вид

$$R=-\frac{(b-a)^5}{180n^4}M_4, \tag{2.81}$$

где, как и выше, $(b-a)$ — длина участка интегрирования, $n=2m$ — общее число частей, на которые разбит участок, и M_4 — наибольшее по абсолютной величине значение четвертой производной функции, которую мы интегрируем.

Вывод формулы парабол основан, как показывает само название, на замене интегрируемой функции участками парабол. Отсюда можно заключить, что формула парабол точна для всех многочленов второй степени. Из формулы (2.81) вытекает, что формула парабол точна даже и для многочленов третьей степени, так как их четвертая производная тождественно равна нулю.

Сама по себе формула (2.81) практически бесполезна, так как найти и оценить четвертую производную весьма затруднительно. Однако сделанное выше замечание позволяет в ряде случаев так или иначе оценить точность полученных по формуле Симпсона приближений.

Прежде всего о применимости формулы Симпсона с данным шагом h можно судить по разностям функции, составленным с тем же шагом. Если вторые или третьи разности функции практически постоянны, то функция достаточно хорошо изображается многочленом соответственно второй или третьей степени, для которых формула Симпсона дает точный результат; в этом случае можно считать, что погрешность результата выражается единицами тех же разрядов, что и третьи разности функции.

Если практически постоянны четвертые разности функции, то можно преобразовать формулу (2.81), заменяя четвертую производную разностями, как об этом было сказано в § 73. Именно, формула (10.73) с $n=4$ дает

$$f^{\mathrm{IV}}(x)\approx\frac{\Delta^4 y}{h^4}.$$

Так как $h=\dfrac{b-a}{n}$, то вместо (2.81) получаем формулу

$$R\approx-\frac{b-a}{180}\Delta^4 y, \tag{3.81}$$

которой уже значительно проще воспользоваться, чем формулой (2.81).

Недостатки формулы (3.81) не только в том, что она применима лишь при практически постоянных четвертых разностях, но еще и в том, что вычисление всех разностей функции до четвертого порядка является довольно трудоемкой работой. Гораздо более простым и надежным является прием, который уже использовался нами выше для формулы трапеций — удвоение шага. При удвоении шага (для возможности этого следует брать n не только четным, но кратным четырем) погрешность формулы Симпсона возрастает в 16 раз. Поэтому погрешность результата, вычисленного с шагом h, примерно в 15 раз меньше, чем разность между этим результатом и результатом, вычисленным с шагом $2h$.

Последнее заключение обычно выражают в форме следующего практического правила: *в интеграле I_{2n} верных знаков на один больше, чем совпадающих знаков в I_n и I_{2n}.*

Обратившись в качестве примера к интегралу, рассмотренному в примере 3.80, заметим, что при интегрировании с двойным шагом ($h=0{,}5$) получается значение интеграла 1,2086. Таким образом, разность между этими значениями составляет 0,0017. Это означает, что погрешность полученного там значения около 0,0001, так что во всяком случае все знаки в числе 1,207 можно считать верными. Более точные вычисления показывают, что так оно и есть.

§ 82. Программирование формулы Симпсона

Рассмотрим программу для вычисления интеграла $\int_a^b f(x)\,dx$ по формуле Симпсона, предполагая, что известны пределы интегрирования a, b и шаг интегрирования h, а блок вычисления подынтегральной функции $f(x)$ написан как стандартный блок с входной ячейкой α и выходной γ. Значение интеграла будем помещать также в ячейку γ. Программу можно написать в виде арифметического цикла с проверкой окончания по достижении абсциссой правого конца b отрезка интегрирования:

$\Omega = $ *конец*
$0 = \Sigma$
$h \cdot \langle\frac{1}{3}\rangle = \Delta$
$a = \alpha$
$f(x)$
$\Sigma + \gamma = \Sigma$
$\alpha + h = \alpha$
$f(x)$
$\gamma \cdot$ «4» $= R$
$\Sigma + R = \Sigma$
$\alpha + h = \alpha$
$f(x)$
$\Sigma + \gamma = \Sigma$
$b - \alpha = 0$
$У_0$
$\Sigma \cdot \Delta = \gamma$
конец

Как видно из написанной программы, мы пользуемся формулой (5.80) для участка (x_0, x_0+2h), начиная с $x_0=a$. Если $\alpha<b$, то вычисленное для этой точки значение функции прибавляется к Σ снова, как значение на левом конце нового участка.

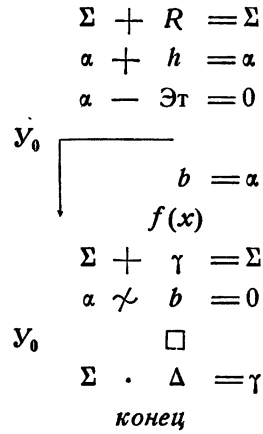
При целых или двоично-рациональных a, b и двоично-рациональном h эта программа будет работать без ошибок. В противном случае эта программа может дать неверные результаты, так как возможны ошибки при проверке окончания цикла. Действительно, если a, b и h выражаются бесконечными двоичными дробями, то при записи в машине их придется округлить. При этом сумма $a+nh$ может не совпасть с b с машинной точностью, а оказаться хотя бы на единицу последнего знака меньше или больше, чем b.

Если случится, что $a+nh<b$, то проверка окончания снова передает управление на рабочую часть цикла, так что интеграл будет вычислен не по тому участку, по которому следует. Кроме того, функция $f(x)$ может оказаться не определенной за пределами заданного участка, поэтому программа может дать совсем нелепый результат или выйти на аварийный останов в блоке счета $f(x)$. То же самое может случиться, если $a+nh>b$, так как придется вычислять функцию в точке за пределами участка интегрирования, т. е., быть может, за пределами области определения функции.

Чтобы избежать этих ошибок, необходимо изменить проверку окончания. Для этого нужно сравнивать очередное значение α не с b, а с некоторым эталоном, отличающимся от b меньше, чем на часть шага, достаточно даже на величину $\Delta=\frac{1}{3}h$, которая все равно вычисляется в программе. Полезно, кроме того, заменить такую точку на b, чтобы не вычислять функцию вне участка интегрирования, если получится $a+nh>b$.

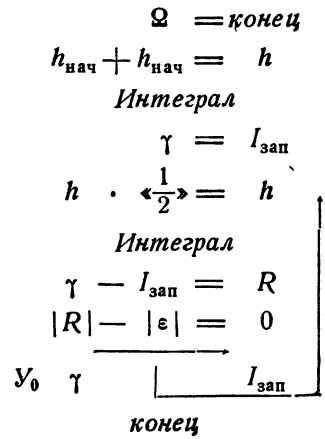
Программу с измененной проверкой окончания можно теперь написать так:

$$\begin{aligned}
&\Omega = \textit{конец}\\
&0 = \Sigma\\
h \cdot &\text{«}\tfrac{1}{3}\text{»} = \Delta\\
&b-\Delta = \text{Эт}\\
&a = \alpha\\
&f(x)\\
\square\quad &\Sigma+\ \gamma = \Sigma\\
&\alpha+\ h = \alpha\\
&f(x)\\
&\gamma \cdot \text{«4»} = R
\end{aligned}$$

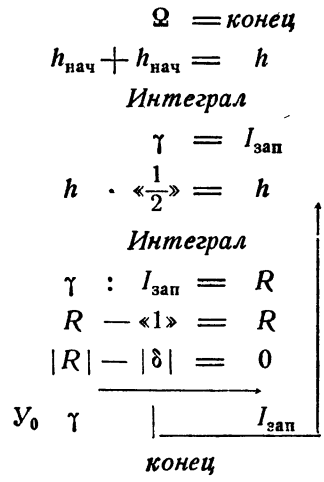


Написанная программа предполагает, что шаг интегрирования h уже выбран. Легко написать программу с автоматическим выбором шага для достижения нужной точности. Проверка точности результатов интегрирования производится путем сравнения результатов, полученных при интегрировании с различными шагами.

Удобно написать такой цикл сравнения с обращением к уже написанной программе, как к блоку под названием *Интеграл*. Будем вычислять интеграл с некоторым шагом $h_{\text{нач}}$ и $2h_{\text{нач}}$. Если полученные значения совпадают с заданной точностью, то интеграл с шагом $h_{\text{нач}}$ дает ответ. В противном случае мы будем уменьшать шаг вдвое до тех пор, пока совпадение с нужной точностью не будет достигнуто. Программу можно написать следующим образом:



Как видно из программы, проверка окончания производится по достижении заданной а б с о л ю т н о й точности ε совпадения интегралов, вычисленных с различным шагом. Не составляет труда написать такую же программу с проверкой окончания по достижении заданной о т н о с и т е л ь н о й точности δ:



Приведенные нами программы трудно превратить в стандартные, так как слишком большое число команд приходится формировать. В блоке *Интеграл*, например, таких команд ровно половина: десять команд из двадцати. Ясно, что формирование такого большого числа команд слишком удлинит программу и для превращения в стандартную ее следует писать иначе.

Мы не будем останавливаться на возможностях составления стандартной программы интегрирования по Симпсону, тем более, что практически более выгодно и удобно строить стандартные программы интегрирования на других приближенных формулах.

ГЛАВА XVI

ВЫЧИСЛЕНИЕ ЭЛЕМЕНТАРНЫХ ФУНКЦИЙ

§ 83. Общие замечания. Вычисление многочленов

Почти во всех случаях при выполнении вычислений требуются значения тех или иных элементарных функций *). При ручных расчетах всегда бывает достаточно воспользоваться для этой цели готовыми таблицами, которые имеются в больших количествах, и, быть может, еще интерполяцией в них. Исключение составляют здесь лишь самые простые функции — многочлены, которые приходится вычислять, так как все многочлены затабулировать невозможно.

Иначе обстоит дело при вычислениях на электронных счетных машинах. Для интерполяции, как мы видели в предыдущей главе, все равно придется писать специальную программу, а запись в память машины больших таблиц элементарных функций оказывается совершенно невозможной из-за ограниченного объема памяти. Поэтому для работы на машине необходимо иметь стандартные программы вычисления элементарных функций. Принципам, на которых основаны такие программы, и посвящена настоящая глава.

Начнем с вычисления многочленов, необходимого и при ручном счете. При однократном вычислении значения многочлена невысокой степени последовательность выполнения операций не имеет особого значения. Однако для вычисления многочленов достаточно высокой степени или же для вычисления большого числа значений многочлена в различных точках последовательность выполнения операций уже существенна.

Предварительное вычисление всех нужных степеней аргумента x^2, x^3, ... обычно является невыгодным, так как требует довольно большого числа операций. При вычислении значений многочлена n-й степени для получения степеней до x^n включительно требуется $n-1$ умножений. Кроме этого, нужно еще n умножений на коэффициенты, т. е. всего $2n-1$ умножений, и n сложений.

---

*) См. примечание на стр. 13.

Обычно для вычисления многочлена пользуются описанной в § 59 *схемой Горнера*, которая требует всего n умножений, т. е. почти вдвое меньше, и те же n сложений. Многочлен

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

представляется здесь в виде

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

(см. формулу (5.59)).

Пример 1.83. Составим таблицу значений многочлена

$$x^4-3x^3+2x^2+x+3$$

для значений x от 1 до 2 с шагом 0,1.

Расписка для выполнения вычислений вручную и все вычисления приведены в табл. 1.83. В ней выписаны результаты всех промежуточных вычислений. При работе на клавишных вычислительных машинах ряд промежуточных колонок можно опускать не выписывая. На

Таблица 1.83

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
x	(1)—«3»	(2)·(1)	(3)+«2»	(4)·(1)	(5)+«1»	(6)·(1)	(7)+«3»
1	—2,0	—2,00	0	0	1,000	1,0000	4,0000
1,1	—1,9	—2,09	—0,09	—0,099	0,901	0,9911	3,9911
1,2	—1,8	—2,16	—0,16	—0,192	0,808	0,9696	3,9696
1,3	—1,7	—2,21	—0,21	—0,273	0,727	0,9451	3,9451
1,4	—1,6	—2,24	—0,24	—0,336	0,664	0,9296	3,9296
1,5	—1,5	—2,25	—0,25	—0,375	0,625	0,9375	3,9375
1,6	—1,4	—2,24	—0,24	—0,384	0,616	0,9856	3,9856
1,7	—1,3	—2,21	—0,21	—0,357	0,643	1,0931	4,0931
1,8	—1,2	—2,16	—0,16	—0,288	0,712	1,2816	4,2816
1,9	—1,1	—2,09	—0,09	—0,171	0,829	1,5751	4,5751
2	—1,0	—2,00	0	0	1,000	2,0000	5,0000

клавишной машине и даже на ручном арифмометре к произведению легко прибавить любое слагаемое, поэтому можно пользоваться распиской, приведенной в табл. 2.83. После получения суммы ее прихо-

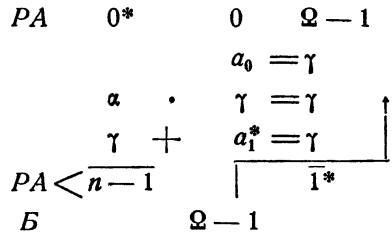
Таблица 2.83

(1)	(2)	(3)	(4)	(5)
x	(1)—«3»	(2)·(1)+«2»	(3)·(1)+«1»	(4)·(1)+«3»

дится сбрасывать с регистра результата и набирать вновь на регистре набора множителя. Поэтому сумму следует записать, а значит, в этом месте можно перейти к вычислению значений в следующей строке.

Если на машине можно переносить число с регистра результата на регистр установки множителя, то и эта расписка не требуется, так как в этом случае все вычисления для нахождения значения многочлена в одной точке можно выполнять, не сбрасывая результата, а значит, нет никакой надобности в записи промежуточных результатов.

Программирование схемы Горнера для вычисления значений многочлена в одной точке не представляет никакого труда. Для многочлена невысокой степени проще всего писать обычную бесцикловую расписку формулы. Для многочленов достаточно высокой степени (например, $n \geqslant 6$) удобнее уже предполагать, что коэффициенты многочлена расположены в ячейках памяти подряд, и программировать цикл с переадресацией. Если использовать регистр адреса, то программу можно представить, например, так:



§ 84. Применение степенных рядов

Степенные ряды являются очень важным аппаратом для табулирования элементарных функций, так как их применение позволяет свести задачу вычисления значений функции к задаче вычисления многочлена, т. е. к выполнению арифметических операций.

Разложение основных элементарных функций в степенные ряды известно из курса математического анализа. Для других функций это разложение может быть получено путем комбинации известных разложений или с помощью общей формулы Тейлора.

Формула Тейлора имеет вид

$$f(x)=f(a)+\frac{f'(a)}{1}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\ldots+$$
$$+\frac{f^{(n)}(a)}{n!}(x-a)^n+\ldots \quad (1.84)$$

Обычно в ней полагают $a=0$ *), благодаря чему получается обычный

*) Распространенным, хотя и исторически необоснованным названием для этого частного случая является название *формула Маклорена*.

степенной ряд по степеням x, обрывая который в нужном месте мы получаем многочлен, приближающий данную функцию.

Наиболее употребительными являются степенные ряды для функций e^x, $\cos x$ и $\sin x$, которые сходятся при любом значении x:

$$\left.\begin{aligned} e^x &= 1 + \frac{x}{1} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots, \\ \cos x &= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots, \\ \sin x &= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \ldots \end{aligned}\right\} \quad (2.84)$$

Для логарифмической функции непосредственно из формулы Тейлора (1.84) можно получить ряд

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n-1} \frac{x^n}{n} + \ldots,$$

который, однако, мало пригоден для вычисления логарифмов, так как сходится только для значений x, удовлетворяющих условию $-1 < x \leqslant 1$. Практически для вычисления значений натурального логарифма используют получающийся из приведенной формулы ряд

$$\ln \frac{1+x}{1-x} = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \ldots + \frac{x^{2n+1}}{2n+1} + \ldots\right). \quad (3.84)$$

Этот ряд тоже сходится лишь при $|x| < 1$, но дробь $\frac{1+x}{1-x}$ может при этом принимать любые положительные значения. Например, для нахождения величины $\ln 5$ достаточно положить $\frac{1+x}{1-x} = 5$, откуда $x = \frac{2}{3}$. Таким образом,

$$\ln 5 = 2\left[\frac{2}{3} + \frac{1}{3}\left(\frac{2}{3}\right)^3 + \frac{1}{5}\left(\frac{2}{3}\right)^5 + \ldots\right].$$

Часто используется также и *биномиальный ряд*, частным случаем которого при натуральном m является известная формула бинома Ньютона:

$$(1+x)^m = 1 + mx + \frac{m(m-1)}{2!}x^2 + \frac{m(m-1)(m-2)}{3!}x^3 + \ldots + \frac{m(m-1)\ldots(m-n+1)}{n!}x^n + \ldots \quad (4.84)$$

Действительно, при натуральном m коэффициенты ряда, начиная с некоторого места, обратятся в нуль, т. е. ряд оборвется. Биномиальный ряд удобен при возведении в дробную степень и при извлечении корней.

Просто выглядит и легко получается также ряд для функции $\operatorname{arctg} x$:

$$\operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \ldots + (-1)^n \frac{x^{2n+1}}{2n+1} + \ldots, \quad (5.84)$$

который, как и ряд для логарифмической функции, сходится лишь в области $-1 < x \leqslant 1$. Тем не менее, для вычисления $\operatorname{arctg} x$ ряда (5.84) вполне достаточно, потому что для $|x| > 1$ можно воспользоваться тождеством

$$\operatorname{arctg} x = \frac{\pi}{2} - \operatorname{arctg} \frac{1}{x}.$$

При вычислении элементарных функций с помощью степенных рядов часто бывает удобно пользоваться *рекуррентными соотношениями* *), которые позволяют вычислять очередной член ряда не непосредственно, а через уже вычисленные предыдущие члены. Для приведенных выше рядов эти соотношения могут быть легко выведены непосредственно. Проще всего взять отношение двух соседних членов.

Рассмотрим, например, степенной ряд (2.84) для функции e^x. Его общий член имеет вид $a_n = \frac{x^n}{n!}$. Взяв отношение последующего члена к предыдущему, получим

$$a_{n+1} : a_n = \frac{x^{n+1}}{(n+1)!} : \frac{x^n}{n!} = \frac{x}{n+1}.$$

Таким образом, для двух соседних членов ряда получаем соотношение

$$a_n = a_{n-1} \cdot \frac{x}{n}. \quad (6.84)$$

Последнее очень удобно для вычисления членов ряда последовательно.

При вычислениях с рядами мы заменяем сумму ряда его частичной суммой, т. е. ограничиваемся конечным числом членов. Естественно возникает вопрос об оценке остатка, т. е. погрешности сделанного приближения, в том случае, когда мы сохраняем данное число членов ряда. Решение вопроса в такой форме дает возможность решить и основной практический вопрос, который ставится наоборот: сколько членов ряда нужно сохранить, чтобы получающаяся погрешность не превышала заданной?

Если члены ряда убывают достаточно быстро и притом с самого начала, то выгодно иметь дело со знакопеременным рядом **), погрешность которого легко оценивать. Действительно, из свойств рядов

*) Так называют соотношения, связывающие между собой два или несколько соседних членов ряда и позволяющие найти следующий член через известные предыдущие. Их называют также *возвратными*.

**) Ряд называют *знакопеременным* (или *знакочередующимся*), если его члены попеременно меняют знаки. Примерами знакопеременных рядов являются степенные ряды для синуса и косинуса.

известно, что *сумма знакопеременного ряда меньше его первого члена* (по абсолютной величине). Отсюда следует, что, заменяя сумму ряда его частичной суммой, мы допускаем погрешность, не превосходящую по модулю *первого из отброшенных членов*. Однако нужно помнить, что эта оценка полезна лишь при указанных выше условиях. Если же члены ряда убывают медленно или убывают хотя и быстро, но не сначала, а первые члены ряда достаточно велики, то хотя общая теорема о сумме ряда остается в силе, но фактическая погрешность будет заметно большей из-за погрешностей вычитания первых больших членов. В таких случаях гораздо выгоднее иметь дело с рядами, все члены которых положительны.

Например, члены рядов (2.84) убывают достаточно быстро и ряды сходятся при любом x. Тем не менее при больших x (скажем, $x>10$) первые члены этих рядов довольно быстро возрастают, и поэтому вычисление $\cos x$ и $\sin x$ с помощью этих рядов чрезвычайно затруднительно, потому что при вычитании больших первых членов происходит такая потеря точности, которую нельзя возместить вычислением большого числа слагаемых. Для рядов с положительными членами оценка погрешности более сложна, и никаких общих методов, пригодных для всех рядов, предложить нельзя. Рассмотрим такую оценку на конкретном примере.

Пример 1.84. Вычислим значение $\ln 5$ с помощью ряда (3.84), взяв пять членов ряда и оценим полученную погрешность.

Как уже указывалось выше, в этом случае надо положить $x=\frac{2}{3}$. Тогда

$$\ln 5 \approx 2\left[\frac{2}{3}+\frac{1}{3}\left(\frac{2}{3}\right)^3+\frac{1}{5}\left(\frac{2}{3}\right)^5+\frac{1}{7}\left(\frac{2}{3}\right)^7+\frac{1}{9}\left(\frac{2}{3}\right)^9+\ldots\right].$$

Абсолютная погрешность этого равенства равна сумме ряда

$$R=2\left[\frac{1}{11}\left(\frac{2}{3}\right)^{11}+\frac{1}{13}\left(\frac{2}{3}\right)^{13}+\ldots\right].$$

Оценку величины R можно получить следующим образом. Если в квадратной скобке все множители перед степенями $\frac{2}{3}$ заменить на $\frac{1}{11}$, то величины суммы могут только возрасти. Тогда

$$R<\frac{2}{11}\left[\left(\frac{2}{3}\right)^{11}+\left(\frac{2}{3}\right)^{13}+\ldots\right].$$

Но сумма справа представляет собой теперь сумму геометрической прогрессии с первым членом $\left(\frac{2}{3}\right)^{11}$ и знаменателем $\left(\frac{2}{3}\right)^2=\frac{4}{9}$. Поэтому

$$R<\frac{2}{11}\,\frac{\left(\frac{2}{3}\right)^{11}}{1-\frac{4}{9}}=\frac{18}{55}\left(\frac{2}{3}\right)^{11}<0{,}004.$$

Приведенный метод позволяет решить и обратную задачу — определить число членов ряда, которые нужно сохранить, чтобы получить значение $\ln 5$ с наперед заданной точностью.

Скорость сходимости приведенных выше рядов, т. е., в конечном счете, число членов, которое нужно взять, чтобы получить сумму с нужной точностью, весьма различна. Более того, даже для одного и того же ряда требуемое число членов меняется в зависимости от x. Вследствие этого быстродействие программ, вычисляющих значения элементарных функций с помощью рядов, оказывается различным. Подробнее речь об этом будет идти в § 86.

§ 85. Цепные дроби

Кроме степенных рядов, при вычислении элементарных функций на электронных счетных машинах используется также аппарат цепных дробей. Так как цепные дроби не изучаются в настоящее время в курсе математики, то мы приведем несколько определений и свойств цепных дробей.

Цепной или *непрерывной дробью* называют выражение вида

$$\cfrac{a_0}{b_0+\cfrac{a_1}{b_1+\cfrac{a_2}{b_2+\dots}}} \tag{1.85}$$

Эта дробь может быть как конечной, так и бесконечной. Для конечной цепной дроби нет нужды ставить вопрос о ее числовом значении, поскольку оно может быть получено преобразованием такой дроби в обычную и притом однозначно, так как речь идет о конечном числе арифметических операций. Для постановки такого вопроса относительно бесконечной цепной дроби нужно познакомиться с некоторыми терминами.

Числа a_0, a_1, a_2, ... в выражении (1.85) называют *частными числителями* цепной дроби, b_0, b_1, b_2, ... — *частными знаменателями*; простую дробь $\frac{a_n}{b_n}$ называют n-м *звеном* цепной дроби, а a_n и b_n — *членами* этого звена. Мы будем полагать, что все частные знаменатели отличны от нуля.

Запись (1.85) слишком громоздка и неудобна. Вместо нее обычно используют более короткую, записывая звенья цепной дроби, соединенные знаками плюс, поставленными на уровне знаменателей:

$$\frac{a_0}{b_0}+\frac{a_1}{b_1}+\frac{a_2}{b_2}+\dots \tag{2.85}$$

Обрывая цепную дробь в различных местах, получаем конечные цепные дроби вида

$$\frac{a_0}{b_0},\quad \frac{a_0}{b_0}+\frac{a_1}{b_1},\quad \frac{a_0}{b_0}+\frac{a_1}{b_1}+\frac{a_2}{b_2},\quad \ldots\ldots \tag{3.85}$$

Они образуют последовательность *подходящих дробей* первоначальной цепной дроби. Дробь $\frac{a_0}{b_0}+\frac{a_1}{b_1}+\ldots+\frac{a_n}{b_n}\equiv\frac{P_n}{Q_n}$ называют n-й подходящей дробью.

Для числителей и знаменателей подходящих дробей легко получить *рекуррентные соотношения* *), позволяющие вычислить P_n и Q_n через n-е звено цепной дроби и предыдущие подходящие дроби. Непосредственным вычислением получаем

$$\frac{P_0}{Q_0}=\frac{a_0}{b_0},$$

$$\frac{P_1}{Q_1}=\frac{a_0}{b_0+\frac{a_1}{b_1}}=\frac{a_0b_1}{b_0b_1+a_1},$$

$$\frac{P_2}{Q_2}=\frac{a_0}{b_0+\frac{a_1}{b_1+\frac{a_2}{b_2}}}=\frac{a_0}{b_0+\frac{a_1b_2}{b_1b_2+a_2}}=$$

$$=\frac{b_2a_0b_1+a_2a_0}{b_2(b_0b_1+a_1)+a_2b_0}=\frac{b_2P_1+a_2P_0}{b_2Q_1+a_2Q_0}.$$

Если заметить, что для перехода от $\frac{P_n}{Q_n}$ к $\frac{P_{n+1}}{Q_{n+1}}$ надо заменить b_n суммой $b_n+\frac{a_{n+1}}{b_{n+1}}$ и воспользоваться методом полной математической индукции, то можно получить требуемые рекуррентные соотношения. Они имеют вид **)

$$\left.\begin{aligned}P_n&=b_nP_{n-1}+a_nP_{n-2},\\ Q_n&=b_nQ_{n-1}+a_nQ_{n-2}.\end{aligned}\right\} \tag{4.85}$$

Выражение для n-й подходящей дроби можно, следовательно, записать так:

$$\frac{P_n}{Q_n}=\frac{b_nP_{n-1}+a_nP_{n-2}}{b_nQ_{n-1}+a_nQ_{n-2}}. \tag{5.85}$$

---

*) Определение термина «*рекуррентное соотношение*» см. на стр. 427.

**) Для того чтобы эти формулы были справедливы и для $n=1$, нужно дополнительно положить $P_{-1}=0$, $Q_{-1}=1$.

Доказательство этого утверждения достаточно простое, и мы предоставляем читателю провести его самостоятельно.

Формулы (4.85) позволяют получить дополнительные соотношения между очередными подходящими дробями. Так, положив, как и раньше, $P_{-1}=0$, $Q_{-1}=1$, получим

$$P_0 Q_{-1} - Q_0 P_{-1} = a_0. \tag{6.85}$$

Отсюда, используя формулы (4.85), методом математической индукции получаем выражение для разности соседних подходящих дробей

$$\frac{P_n}{Q_n} - \frac{P_{n-1}}{Q_{n-1}} = \frac{(-1)^n a_0 a_1 \dots a_n}{Q_{n-1} Q_n}. \tag{7.85}$$

Действительно, предположим, что для некоторого n справедливо равенство

$$P_n Q_{n-1} - Q_n P_{n-1} = (-1)^n a_0 a_1 \dots a_n. \tag{8.85}$$

При $n=0$ это равенство совпадает с формулой (6.85), справедливость которой очевидна. Далее, с помощью формул (4.85) находим

$$P_{n+1}Q_n - Q_{n+1} P_n = (b_{n+1} P_n + a_{n+1} P_{n-1}) Q_n - (b_{n+1} Q_n + a_{n+1} Q_{n-1}) P_n = \\ = -a_{n+1}(P_n Q_{n-1} - Q_n P_{n-1}),$$

так что, вследствие индуктивного предположения (8.85),

$$P_{n+1} Q_n - Q_{n+1} P_n = (-1)^{n+1} a_0 a_1 \dots a_n a_{n+1}.$$

Тем самым доказано, что формула (8.85) справедлива при любом n. Но тогда

$$\frac{P_n}{Q_n} - \frac{P_{n-1}}{Q_{n-1}} = \frac{P_n Q_{n-1} - Q_n P_{n-1}}{Q_n Q_{n-1}} = \frac{(-1)^n a_0 a_1 \dots a_n}{Q_n Q_{n-1}},$$

а это и есть формула (7.85).

Напишем аналогичное равенство для следующей пары соседних подходящих дробей:

$$\frac{P_{n+1}}{Q_{n+1}} - \frac{P_n}{Q_n} = \frac{(-1)^{n+1} a_0 a_1 \dots a_{n+1}}{Q_n Q_{n+1}},$$

и сложим его с равенством (7.85):

$$\frac{P_{n+1}}{Q_{n+1}} - \frac{P_{n-1}}{Q_{n-1}} = (-1)^n \frac{a_0 a_1 \dots a_n}{Q_n} \left(\frac{1}{Q_{n-1}} - \frac{a_{n+1}}{Q_{n+1}} \right) = \\ = (-1)^{n+1} \frac{a_0 a_1 \dots a_n}{Q_{n-1} Q_n Q_{n+1}} (a_{n+1} Q_{n-1} - Q_{n+1}).$$

Заменив в последней скобке Q_{n+1} по формуле (4.85), получим

$$\frac{P_{n+1}}{Q_{n+1}} - \frac{P_{n-1}}{Q_{n-1}} = (-1)^n \frac{a_0 a_1 \dots a_n b_{n+1}}{Q_{n-1} Q_{n+1}}. \tag{9.85}$$

Формула (9.85) играет чрезвычайно важную роль, позволяя сравнивать между собой подходящие дроби четного или, наоборот, нечетного порядка. Если сделать дополнительное предположение, что все элементы цепной дроби положительны, то из формулы (9.85) вытекает два следующих утверждения:

Подходящие дроби четного порядка монотонно убывают. Подходящие дроби нечетного порядка монотонно возрастают.

В самом деле, из формулы (9.85) следует, что разность между следующей и предыдущей дробями четного порядка всегда есть отрицательное число, так как если и $n+1$ и $n-1$ числа четные, то само n нечетно. Аналогичная разность для дробей нечетного порядка всегда положительна.

Последовательность подходящих дробей конечной цепной дроби конечна. Значения подходящих дробей служат приближениями истинного значения конечной цепной дроби попеременно по избытку и по недостатку.

Для бесконечной цепной дроби последовательность подходящих дробей бесконечна. Из выражения (1.85) для цепной дроби легко вывести, что подходящие дроби нечетного порядка не превосходят $\frac{a_0}{b_0}$, а подходящие дроби четного порядка положительны. Таким образом, *последовательность подходящих дробей нечетного порядка монотонно возрастает и ограничена сверху, а последовательность подходящих дробей четного порядка монотонно убывает и ограничена снизу.* Следовательно, *обе эти последовательности сходятся.*

Из сказанного не вытекает, что обе последовательности подходящих дробей имеют общий предел. Вообще говоря, этого может и не быть. Но *если обе последовательности подходящих дробей сходятся к общему пределу, то говорят, что бесконечная цепная дробь сходится.* В этом случае предел последовательности подходящих дробей $\lim\limits_{n\to\infty}\frac{P_n}{Q_n}$ существует и конечен. Этот предел и называют *числовым значением бесконечной цепной дроби.* Как и в случае ряда, для практического нахождения числового значения бесконечной цепной дроби ее заменяют подходящей дробью достаточно большого порядка.

Если отказаться от предположения положительности всех элементов цепной дроби, то монотонность последовательностей подходящих дробей четного и нечетного порядков нельзя гарантировать. Это не отражается, конечно, на определении сходимости бесконечной цепной дроби и ее численного значения.

Из выражения (1.85) видно, что если умножить элементы a_0, b_0 и a_1 цепной дроби на какой-либо множитель p_0, то значение дроби при этом не изменится. Точно так же можно умножать на один и тот же

множитель p_m элементы a_m, b_m, a_{m+1}. Поэтому справедливо тождество

$$\frac{a_0}{b_0}+\frac{a_1}{b_1}+\frac{a_2}{b_2}+\ldots+\frac{a_n}{b_n}+\ldots=$$
$$=\frac{p_0a_0}{p_0b_0}+\frac{p_0p_1a_1}{p_1b_1}+\frac{p_1p_2a_2}{p_2b_2}+\ldots+\frac{p_{n-1}p_na_n}{p_nb_n}+\ldots \qquad (10.85)$$

С помощью тождества (10.85) можно подвергнуть цепную дробь различным преобразованиям. Например, можно подобрать множители p_0, p_1, ..., p_m, ... таким образом, чтобы все частные числители цепной дроби были равны единице. Для этого достаточно, очевидно, положить

$$p_0=\frac{1}{a_0},\quad p_1=\frac{1}{a_1p_0},\quad p_2=\frac{1}{a_2p_1},\ldots$$

Тогда цепная дробь приведется к виду

$$\cfrac{1}{a_0+\cfrac{1}{a_1+\cfrac{1}{a_2+\ldots}}}$$

или, в сокращенной записи,

$$\frac{1}{a_0}+\frac{1}{a_1}+\frac{1}{a_2}+\ldots,$$

где

$$a_0=p_0b_0,\ a_1=p_1b_1,\ \ldots,\ a_n=p_nb_n,\ \ldots$$

Такие цепные дроби называют *обыкновенными*, а ее частные знаменатели a_i — *неполными частными* обыкновенной цепной дроби. Для обыкновенной цепной дроби используется также сокращенное обозначение

$$[a_0,\ a_1,\ \ldots,\ a_n,\ \ldots]. \qquad (11.85)$$

Ограничиваясь рассмотрением только обыкновенных цепных дробей, можно сформулировать простую и удобную для использования теорему об условиях сходимости.

Теорема. *Если все неполные частные обыкновенной цепной дроби*

$$\frac{1}{a_0}+\frac{1}{a_1}+\ldots+\frac{1}{a_n}+\ldots$$

положительны, то для сходимости этой цепной дроби необходимо

и достаточно, чтобы ряд из ее неполных частных

$$\sum_{n=1}^{\infty} a_n = a_1 + a_2 + \ldots + a_n + \ldots$$

расходился.

На доказательстве этой теоремы мы останавливаться не будем.

Л. Эйлер еще в 1739 г. предложил общую формулу для преобразования степенно́го ряда в цепную дробь. Этой формулой можно воспользоваться для разложения элементарных функций в цепные дроби. Подходящие дроби будут давать при этом *рациональные приближения* соответствующих функций. Цепную дробь можно строить таким образом, чтобы ее n-я подходящая дробь тождественно равнялась соответствующей частичной сумме степенно́го ряда (такие дроби называют *равноценными*).

Формула Эйлера для преобразования степенно́го ряда в цепную дробь имеет вид

$$\sum_{n=0}^{\infty} c_n x^n = \frac{c_0}{1} - \frac{c_1 x}{c_0 + c_1 x} - \frac{c_0 c_2 x}{c_1 + c_2 x} - \frac{c_1 c_3 x}{c_2 + c_3 x} - \ldots - \frac{c_{n-2}\, c_n x}{c_{n-1} + c_n x} - \ldots \qquad (12.85)$$

Нужно только иметь в виду, что вопрос о сходимости цепной дроби, полученной по формуле (12.85), приходится решать отдельно, независимо от сходимости или расходимости исходного степенно́го ряда. Оказывается, что сходимость или расходимость преобразуемого степенно́го ряда и получающейся цепной дроби никак не связаны между собою и фактически могут встретиться все четыре возможные комбинации.

Пользуясь формулой Эйлера (12.85), можно получать различные разложения элементарных функций в цепные дроби. Приведем некоторые из формул, наиболее употребительные при вычислении элементарных функций.

Для показательной функции e^x справедливы следующие разложения:

$$e^x = \frac{1}{1} - \frac{x}{1} + \frac{x}{2} - \frac{x}{3} + \frac{x}{2} - \frac{x}{5} + \ldots + \frac{x}{2} - \frac{x}{2n+1} + \ldots \qquad (13.85)$$

Приведенная цепная дробь сходится для всех значений x, хотя скорость ее сходимости невелика. Чаще используется другое разложение:

$$e^x = 1 + \frac{2x}{2-x} + \frac{x^2}{6} + \frac{x^2}{10} + \frac{x^2}{14} + \ldots + \frac{x^2}{2(2n+1)} + \ldots \qquad (14.85)$$

Логарифмическую функцию можно представить в виде цепной дроби

$$\ln(1+x) = \frac{x}{1} + \frac{x}{2} + \frac{x}{3} + \frac{2x}{2} + \frac{2x}{5} + \ldots + \frac{nx}{2} + \frac{nx}{2n+1} + \ldots \qquad (15.85)$$

Тригонометрические функции также могут быть представлены в виде цепных дробей, сходящихся для всех значений x. Наиболее употребительны следующие разложения:

$$\left.\begin{aligned}\sin x &= x - \cfrac{x^3}{6} + \cfrac{3x^2}{10} - \cfrac{11x^2}{42} + \cfrac{25x^2}{66} - \ldots \\ \cos x &= \cfrac{1}{1} + \cfrac{x^2}{2} - \cfrac{5x^2}{6} + \cfrac{3x^2}{50} - \cfrac{313x^2}{126} + \ldots\end{aligned}\right\} \qquad (16.85)$$

Более удобно следующее представление для $\operatorname{tg} x$, в котором, в отличие от приведенных разложений (16.85), легко выписать общий член разложения

$$\operatorname{tg} x = \cfrac{x}{1} - \cfrac{x^2}{3} - \cfrac{x^2}{5} - \cfrac{x^2}{7} - \ldots - \cfrac{x^2}{2n+1} - \ldots \qquad (17.85)$$

Можно привести также и представление в виде цепной дроби обратной тригонометрической функции $y = \operatorname{arctg} x$

$$\operatorname{arctg} x = \cfrac{x}{1} + \cfrac{x^2}{3} + \cfrac{4x^2}{5} + \cfrac{9x^2}{7} + \ldots + \cfrac{n^2x^2}{2n+1} + \ldots \qquad (18.85)$$

При вычислении значений $\arcsin x$ пользуются формулой

$$\arcsin x = \operatorname{arctg} \frac{x}{\sqrt{1-x^2}}.$$

Использование приведенных формул для программирования вычисления элементарных функций будет рассмотрено в § 87.

§ 86. Программы вычисления элементарных функций с помощью рядов

Степенные ряды, приведенные в § 84, являются удобным средством для программирования вычисления элементарных функций. Каждый из рядов легко программируется как обычный цикл, который можно писать и как арифметический, и (чаще) как итерационный.

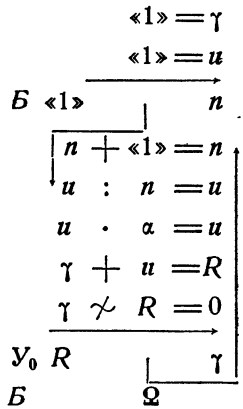
Начнем с рассмотрения показательной функции $y = e^x$. Заметим, что каждый следующий член ряда

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$$

может быть получен из предыдущего с помощью соотношения $u_n = u_{n-1} \cdot \frac{x}{n}$. Если не ограничивать область изменения аргумента, то цикл следует писать как итерационный. Для вычислений с машинной точностью можно проверять общий член ряда на совпадение с нулем, т. е. считать до тех пор, пока члены ряда не станут машинным нулем. Впрочем, в этом нет никакой необходимости. Гораздо проще и быстрее сравнивать две соседние суммы и прекращать вычисления тогда, когда они совпадут, т. е. прибавление очередного слагаемого

не будет изменять суммы. Поскольку вычисления в машине ведутся с определенной относительной точностью, этот момент наступит раньше, чем члены ряда обратятся в машинный нуль.

Программу будем писать как стандартную, с входной ячейкой α и выходной γ.



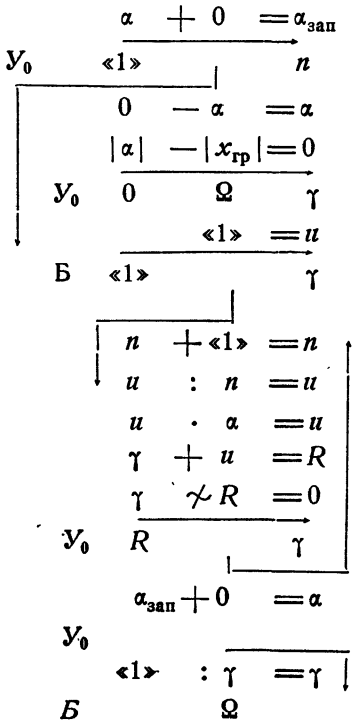
Написанная программа неудобна для отрицательных аргументов, больших по абсолютной величине, так как в этом случае ряд получается знакопеременным и его первые члены велики по абсолютной величине, тогда как значение функции мало. В этом случае может получиться большая потеря точности. Выгоднее иметь дело с другой программой, которая для отрицательного аргумента пользуется тождеством

$$e^x = \frac{1}{e^{-x}},$$

вычисляя значения функции все-таки для положительного показателя степени, т. е. работая с знакопостоянным рядом.

У такой программы был бы другой недостаток. Для очень больших положительных показателей степени (в рассматриваемой машине — для $x > 46$) значение функции не помещается в разрядной сетке машины и при попытке вычисления этого значения программа остановится по переполнению разрядной сетки. Это явление вполне естественно. Но при вычислении показательной функции при очень большом отрицательном показателе степени машина тоже будет выходить на аварийный останов, а это уже неестественно. Программа должна в этом случае не останавливаться и выдавать нуль в качестве значения функции. Для этого мы вставим в нашу программу дополнительную проверку: отрицательное значение аргумента сравнивается с некоторым граничным значением $x_{гр}$; если $|\alpha| < |x_{гр}|$, то значение функции вычисляется так, как сказано ранее; если же $|\alpha| \geqslant |x_{гр}|$, то в ячейку γ засылается нуль.

Программу, удовлетворяющую этим условиям, можно написать, например, так:



Сначала аргумент α пересылается в запасную ячейку $\alpha_{зап}$. Одновременно мы анализируем знак аргумента. При $\alpha > 0$ первая команда выработает сигнал $\omega = 0$ и управление будет передано на начало рабочей части, вычисляющей значение показательной функции, как и в предыдущей программе. При $\alpha < 0$ первая команда вырабатывает сигнал $\omega = 1$ и управление будет передано следующей команде, которая изменит знак у аргумента. Затем идет сравнение α с $x_{гр}$. Если $|\alpha| > |x_{гр}|$, то следующая команда засылает 0 в γ и передает управление ячейке Ω, т. е. выходит из программы. В противном случае программа вычисляет значение $e^{-\alpha}$.

После того как заканчивается работа цикла по вычислению функции, из ячейки $\alpha_{зап}$ в ячейку α засылается первоначальное значение аргумента и снова проверяется его знак. При $\alpha > 0$ программа уходит на Ω, а при $\alpha < 0$ сначала находится обратная величина, т. е. значение e^{α}.

Кроме показательной функции e^x, часто приходится иметь дело с комбинациями показательных функций, которые называются *гипер-*

болическими функциями. Наиболее важными из них являются *гиперболический косинус* $\operatorname{ch} x$ и *гиперболический синус* $\operatorname{sh} x$, которые определяются формулами

$$\operatorname{ch} x = \frac{e^x + e^{-x}}{2}, \quad \operatorname{sh} x = \frac{e^x - e^{-x}}{2}. \tag{1.86}$$

Разложение этих функций в степенные ряды имеет вид

$$\begin{aligned} \operatorname{ch} x &= 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + \frac{x^{2n}}{(2n)!} + \ldots, \\ \operatorname{sh} x &= x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2n+1}}{(2n+1)!} + \ldots \end{aligned} \tag{2.86}$$

Ряды (2.86) легко получить, взяв полусумму и полуразность рядов для e^x и e^{-x}. Последний сразу получается из ряда для e^x, если в нем заменить x на $-x$. Читатель без труда самостоятельно справится с этими выкладками.

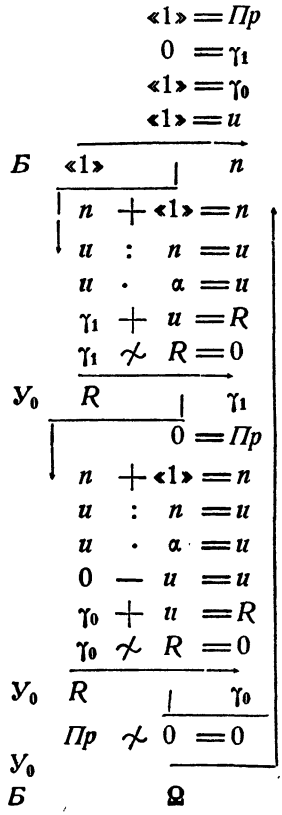
Члены этих двух рядов удобно вычислять поочередно, как это и делалось в программе для вычисления e^x. Поэтому выгодно иметь одну программу, одновременно вычисляющую функции $\gamma_0 = \operatorname{ch} \alpha$ и $\gamma_1 = \operatorname{sh} \alpha$. Так как всегда справедливо неравенство $\operatorname{ch} x > \operatorname{sh} x$ и члены обоих рядов, начиная с некоторого места, убывают, достижение машинной относительной точности можно проверять только для $\operatorname{ch} x$. Тогда программу можно будет написать следующим образом:

$$\begin{array}{llcl}
 & & 0 & = \gamma_1 \\
 & & «1» & = \gamma_0 \\
 & & «1» & = u \\
Б & «1» & \overrightarrow{\quad\quad} & n \\
A & n & + «1» & = n \\
 & u & : \; n & = u \\
 & u & \cdot \; \alpha & = u \\
 & \gamma_1 & + \; u & = \gamma_1 \\
 & n & + «1» & = n \\
 & u & : \; n & = u \\
 & u & \cdot \; \alpha & = u \\
 & \gamma_0 & + \; u & = R \\
 & \gamma_0 & \not\sim \; R & = 0 \\
У_0 & R & A & \gamma_0 \\
Б & & \Omega &
\end{array}$$

Сравнив ряды (2.86) с рядами (2.84) для тригонометрических функций $\cos x$ и $\sin x$, заметим, что они состоят из тех же членов. Поэтому написанная программа может быть использована и для вычисления

$\gamma_0 = \cos\alpha$ и $\gamma_1 = \sin\alpha$, если вставить еще одну команду перемены знака. Впрочем, кроме этого, нужно еще позаботиться о том, чтобы проверять точность вычислений обеих функций, так как неизвестно, для какой из них требуется большое число членов.

Последнего можно добиться, например, таким путем. Возьмем специальную ячейку — признак (*Пр*), куда вначале запишем единицу. После того как будет вычислен синус с машинной точностью, в эту ячейку будет записан нуль. Проверка окончания цикла после вычисления очередного члена для косинуса будет передавать управление на начало цикла до тех пор, пока не будет вычислен с машинной точностью косинус, независимо от признака. После того как косинус будет сосчитан, мы будем проверять содержимое ячейки *Пр* и если в ней записана пока еще единица, то управление снова будет передаваться началу цикла. Вычисления будут закончены лишь тогда, когда обе функции будут сосчитаны с машинной точностью. В написанной ниже программе легко теперь разобраться самостоятельно



Как уже было отмечено, программы для вычисления тригонометрических и гиперболических функций с помощью рядов отличаются одной командой. Поэтому нет никакого смысла записывать их в состав библиотеки стандартных программ отдельно. Выгоднее и экономнее объединить две эти программы в одну программу с двумя разными входами, которые либо зашлют на нужное место команду перемены знака, если нужно считать тригонометрические функции, либо на место этой команды зашлют какую-нибудь «пустую» команду, вроде $0 \vee 0 = 0$, если требуется вычисление гиперболических. По сравнению с написанной программой нужно будет потратить еще только четыре ячейки — две для заголовков и две для засылаемых команд, вместо того чтобы размещать в памяти еще одну такую же программу. Единая программа будет выглядеть так:

cos, sin	*Б*	Триг			*T*
ch, sh		Гип			$= T$
		«1»			$=$ *Пр*
		0			$= \gamma_1$
		«1»			$= \gamma_0$
		«1»			$= u$
	Б	«1»			n
A		n	$+$	«1»	$= n$
		u	:	n	$= u$
		u	$\cdot$	α	$= u$
		γ_1	$+$	u	$= R$
		γ_1	$\not\sim$	R	$= 0$
	У$_0$	R			γ_1
		0			$=$ *Пр*
		n	$+$	«1»	$= n$
		u	:	n	$= u$
		u	$\cdot$	α	$= u$
T		н. п.			
		γ_0	$+$	u	$= R$
		γ_0	$\not\sim$	R	$= 0$
	У$_0$	R		A	γ_0
		Пр	$\not\sim$	0	$= 0$
	У$_0$			A	
	Б			Ω	
Триг		0	$-$	u	$= u$
Гип		0	$\vee$	0	$= 0$

Для вычисления натуральных логарифмов с помощью ряда можно воспользоваться рядом (3.84) для $\ln \alpha = \ln \frac{1+x}{1-x}$. Отсюда следует, что $x = \frac{\alpha - 1}{\alpha + 1}$. Работу нужно начать с вычисления значения x, которое мы запишем в ячейку R_1. Кроме того, в ячейке R_2 вычислим x^2; в ячейке n будем вычислять последовательные нечетные числа. Общий член ряда вычисляется делением нечетной степени x, образуемой в ячейке R_1, на соответствующее нечетное целое число. Последняя команда программы удваивает полученную сумму ряда.

При $\alpha < 0$ мы получим $|x| > 1$, и ряд окажется расходящимся. В этом случае частичные суммы ряда неограниченно возрастают и машина выйдет на *стоп* по переполнению разрядной сетки.

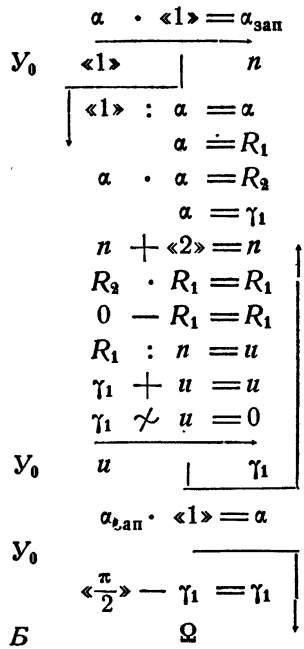
Программу можно написать так:

$$
\begin{array}{lccccl}
 & \alpha & - & \text{«1»} & = & R_1 \\
 & \alpha & + & \text{«1»} & = & R_2 \\
 & R_1 & : & R_2 & = & R_1 \\
 & R_1 & \cdot & R_1 & = & R_2 \\
 & & & R_1 & = & \gamma_0 \\
 & & & \text{«1»} & = & n \\
 & n & + & \text{«2»} & = & n \\
 & R_1 & \cdot & R_2 & = & R_1 \\
 & R_1 & : & n & = & u \\
 & \gamma_0 & + & u & = & u \\
 & \gamma_0 & \not\sim & u & = & 0 \\
У_0 & u & & | & & \gamma_0 \\
 & \gamma_0 & + & \gamma_0 & = & \gamma_0 \\
Б & & & \Omega & &
\end{array}
$$

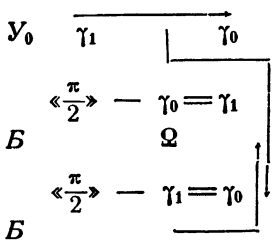
Нетрудно написать и программу для вычисления $\operatorname{arctg} x$ с помощью ряда (5.84). В самом деле, этот ряд содержит те же члены, что и ряд для логарифма, использованный нами в предыдущей программе, но является знакопеременным. Так как этот ряд сходится лишь при $|x| < 1$, а для $|x| > 1$ нужно пользоваться тождеством $\operatorname{arctg} x = \frac{\pi}{2} - \operatorname{arctg} \frac{1}{x}$, то программу можно написать так, как мы это делали для показательной функции.

Прежде всего перепишем значение α в $\alpha_{\text{зап}}$, выяснив при этом, верно ли неравенство $|\alpha| < 1$. Если оно верно, перейдем к вычислению суммы ряда, а если нет, заменим α на $\frac{1}{\alpha}$. После вычисления суммы ряда тем же способом проверим, восстанавливая значение α,

получен ли уже окончательный результат или полученное значение надо вычесть из $\frac{\pi}{2}$. Программу можно написать так:



Для получения, кроме значения $\gamma_1 = \operatorname{arctg} \alpha$, еще и $\gamma_0 = \operatorname{arcctg} \alpha$, вместо последних трех команд написанной только что программы можно написать следующие пять:



Все написанные нами программы рассчитаны на вычисление элементарных функций с машинной точностью. Не составляет труда внести в них изменения таким образом, чтобы значения функции вычислялись с заданной абсолютной или относительной точностью. Для этого нужно только изменить команды проверки окончания цикла. Быстродействие программы при этом, естественно, может повыситься.

Однако скорость сходимости рядов резко изменяется в зависимости от значения аргумента. Поэтому стандартные программы, написанные с помощью рядов, выгодно писать всегда как итерационные циклы, а не как арифметические. Заранее оценивать требуемое число членов ряда имеет смысл только для таких программ, в которых ряды используются для точно фиксированных областей изменения аргумента.

§ 87. Программы вычисления элементарных функций с помощью цепных дробей

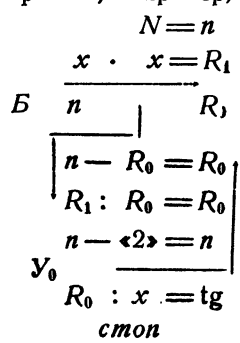
Представление элементарных функций в виде цепных дробей, рассмотренное в § 85, дает еще один важный аппарат для вычисления элементарных функций. Его преимуществом является возможность оценки погрешности и нужного порядка подходящей дроби, позволяющего писать стандартные программы в виде арифметических, а не итерационных циклов.

В качестве первого примера рассмотрим вычисление тангенса по формуле (17.85). Чтобы лучше представлять себе требуемые действия, запишем эту дробь в развернутом виде

$$\operatorname{tg} x = \cfrac{x}{1-\cfrac{x^2}{3-\cfrac{x^2}{5-\cfrac{x^2}{7-\dots}}}} \tag{1.87}$$

Из написанного выражения видно, что вычисления удобно программировать в виде цикла, идущего снизу вверх. Вследствие этого цикл должен быть необходимо арифметическим, так как мы должны начать с определенного места.

Пусть N — некоторое заранее выбранное нечетное число. Напишем цикл для вычисления по приведенной формуле, полагая, что аргумент находится в ячейке x, а ответ нужно поместить в ячейку tg. Цикл можно запрограммировать, например, так:



Читатель легко разберется в приведенной программе. Заметим только, что после окончания работы цикла мы получим значение дроби, в числителе которой стоит x^2. Поэтому для нахождения тангенса найденное значение нужно еще разделить на x.

Для нахождения значений тригонометрических функций $\sin x$ и $\cos x$ удобно воспользоваться формулами, выражающими эти функции через тангенс половинного угла:

$$\cos x = \frac{1 - \operatorname{tg}^2 \frac{x}{2}}{1 + \operatorname{tg}^2 \frac{x}{2}}, \qquad \sin x = \frac{2 \operatorname{tg} \frac{x}{2}}{1 + \operatorname{tg}^2 \frac{x}{2}}. \tag{2.87}$$

Вычислив при данном значении аргумента x величину $\operatorname{tg} \frac{x}{2}$ по формуле (1.87) с помощью написанного выше арифметического цикла, можно после этого получить значения $\cos x$ и $\sin x$ по формулам (2.87).

Напишем соответствующую программу, оформив ее как стандартную программу без информации. Аргумент будем считать записанным в ячейке α, а ответы — в ячейках $\gamma_0 = \cos x$ и $\gamma_1 = \sin x$. Оценка величины цепной дроби показывает, что достаточная точность получится уже при $N = 7$. Поэтому программу вычисления тригонометрических функций с помощью цепных дробей можно написать так:

$«7» = n$

$\alpha : «2» = R_2$

$R_2 \cdot R_2 = R_1$

$Б \; n \qquad R_0$

$n - R_0 = R_0$

$R_1 : R_0 = R_0$

$n - «2» = n$

$У_0$

$R_0 : R_2 = R_2$

$R_2 + R_2 = R_1$

$R_2 \cdot R_2 = R_0$

$«1» + R_0 = R_2$

$R_1 : R_2 = \gamma_1$

$«1» - R_0 = R_1$

$R_1 : R_2 = \gamma_0$

$Б \qquad \Omega$

Аналогичную программу можно написать также и для гиперболических функций. Для них верны и формулы (2.87) и разложение (1.87), в котором только следует заменить все знаки минус на плюсы. Поэтому, как и при вычислении с помощью рядов (см. § 86), программы для вычисления гиперболических функций будут отличаться от написанной лишь одной командой. Не составит труда объединить эти две программы в одну, как это было сделано нами в § 86.

Рассмотрим теперь вычисление показательной функции e^x. Прежде всего воспользуемся тождеством

$$e^x = 2^{\frac{x}{\ln 2}}.$$

Далее, можно написать

$$e^x = 2^A \cdot 2^B,$$

где $A = \left[\frac{x}{\ln 2}\right]$ — целая часть числа $\frac{x}{\ln 2}$, а $B = \left\{\frac{x}{\ln 2}\right\} = \frac{x}{\ln 2} - \left[\frac{x}{\ln 2}\right]$ — его дробная часть *). Выделение целой и дробной части числа уже рассматривалось нами в § 35. Окончательно напишем

$$e^x = 2^A \cdot e^{B \ln 2}. \qquad (3.87)$$

Два множителя, входящие в правую часть равенства (3.87), вычисляются отдельно. Заметим, что мантисса числа 2^A, где A — целое, состоит из одной единицы в 36-м разряде, т. е. имеет вид (4000,0,0). Истинная величина порядка числа 2^A равна самому числу A, записанному в виде количества единиц порядковой (кодовой) части ячейки с соответствующим знаком. Поэтому для получения условного порядка 2^A надо число A прибавить к 100_8 с помощью операции сложения порядков, или вычитать A из 100_8, если A отрицательно.

Таким образом, можно взять ячейку, имеющую вид (100,4000,0,0), которая представляет собой число $\frac{1}{2}$, записанное в плавающей форме, и прибавлять к ее порядку (или вычитать из него) число A, записанное в виде числа единиц порядковой части ячейки. Это можно осуществить с помощью программы

1)			T^0	$= T$
2)	(200, 0, 0, 0)	$\wedge$	A	$= 0$
3)	y_1		5)	
4)	T	, +	(20, 0, 0, 0)	$= T$
5)	A	$\wedge$	(F, F, F)	$= R_1$
6)	A	[→]	R_1	$= R_1$
T 7)	(«$\frac{1}{2}$»	, +	R_1	$= \gamma_0$) н. п.

*) Фактически удобнее брать в качестве A ближайшее к $\frac{x}{\ln 2}$ целое число, но это не меняет существа дела.

Команда 1) служит здесь для восстановления переменной команды T, первоначальное состояние которой T^0 указано в скобках. Далее, команда 2) проверяет знак числа A, записанного в плавающей форме. Если в знаковом разряде ячейки A записан нуль, т. е. $A \geqslant 0$, то в результате команды 2) в пересечении получится нулевое слово и выработается сигнал $\omega = 1$. Тогда команда 3) передаст управление команде 5), которая является началом рабочей части.

При $A < 0$ в знаковом разряде ячейки A будет записана единица, которая останется и в пересечении. Поэтому после команды 2) выработается сигнал $\omega = 0$ и команда 3) передаст управление команде 4), которая к кодовой части ячейки прибавит 20. При этом команда T из команды сложения кодовых частей (с кодом 53) превратится в команду вычитания (с кодом 73). После этого управление перейдет команде 5).

Рабочая часть программы состоит из трех команд. Команда 5) высекает мантиссу числа A, стоящую на своем месте. Затем команда 6) сдвигает эту мантиссу в младшие разряды кодовой части, так что после нее число A оказывается записанным в фиксированной форме, в виде числа единиц кодовой части ячейки R_1. После этого команда 7) прибавит полученное число к заготовке или вычтет его из нее, если команда T переформировывалась командой 4) и в ячейке γ_0 будет записано 2^A.

Остается вычислить второй множитель формулы (3.87), имеющий вид e^t, где $t = B \ln 2$. Для этого мы воспользуемся разложением показательной функции в цепную дробь по формуле (14.85). Поскольку при любых x мы будем получать $t < 1$, то достаточно взять небольшое число членов. Оценка показывает, что можно ограничиться тремя членами дроби, т. е. разложением вида

$$e^t = 1 + \cfrac{2t}{(2-t) + \cfrac{t^2}{6 + \cfrac{t^2}{10}}}. \tag{4.87}$$

Для последней дроби выгодно писать бесцикловую программу. Поэтому дробь (4.87) удобно переписать в виде подходящей дроби

$$e^t = \frac{120 + 60t + 12t^2 + t^3}{120 - 60t + 12t^2 - t^3},$$

для вычисления которой ее следует записать так:

$$e^t = \frac{12(10 + t^2) + t(60 + t^2)}{12(10 + t^2) - t(60 + t^2)}. \tag{5.87}$$

Программирование формулы (5.87) не вызывает никаких затруднений.

Требуемую программу можно, например, написать так:

$$
\begin{aligned}
t \cdot t &= t^2 \\
t^2 + \text{«10»} &= R_0 \\
R_0 \cdot \text{«12»} &= R_0 \\
t^2 + \text{«60»} &= R_1 \\
R_1 \cdot t &= R_1 \\
R_0 + R_1 &= R_2 \\
R_0 - R_1 &= R_1 \\
R_2 : R_1 &= R_0,
\end{aligned}
$$

после чего команда

$$\gamma_0 \cdot R_0 = \gamma_0.$$

дает нам в ячейке γ_0 нужное значение e^{α}.

Для получения окончательной программы вычисления $\gamma_0 = e^{\alpha}$ достаточно объединить написанные нами программы, присоединив к ним команды получения величин $\frac{\alpha}{\ln 2}$, $A = \left[\frac{\alpha}{\ln 2}\right]$, и $t = \alpha - A \ln 2$, а также все требуемые константы. С этой работой читатель легко справится самостоятельно.

Необходимо иметь в виду, что фактические библиотечные программы для вычисления элементарных функций как с помощью рядов, так и с помощью цепных дробей, хотя и построены на описанных нами принципах, но не совпадают в точности с теми, которые приводятся в нашей книге.

Это объясняется прежде всего тем, что на конкретных вычислительных машинах имеется ряд специальных операций, не рассматривавшихся нами, которые позволяют отдельные места программировать более коротко и просто. Кроме того, так как при составлении библиотечных программ экономия ячеек имеет очень большое значение, в ряде случаев удается сократить программу при помощи различных искусственных приемов.

Рафаил Самойлович Гутер,
Борис Владимирович Овчинский,
Павел Тувьевич Резниковский

Программирование
и вычислительная математика

М., 1965 г., 448 стр. с илл.

Редактор *Ю. П. Оревков.*
Техн. редактор *С. Я. Шкляр.*
Корректор *М. Л. Липелис.*

Сдано в набор 10/IV 1965 г. Подписано к печати 23/VIII 1965 г. Бумага 60×90¹/₁₆. Физ. печ. л. 28. Условн. печ. л. 28. Уч.-изд. л. 25,68. Тираж 65 000 экз. Т-10735. Цена книги 87 коп. Заказ № 1700.

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

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

Цена 87 коп.