

Ю.В. ПУХНАЧЕВ И.Д. ДАНИЛОВ

МИКРО КАЛЬКУЛЯТОРЫ для ВСЕЙ

Искусство ■
программирования ■
Сфера применения ■
калькуляторов ■



народный университет · 1986
естественнонаучный факультет

ЗНАНИЕ НАРОДНЫЙ УНИВЕРСИТЕТ
естественнонаучный факультет
Издаётся с 1961 г.

Ю. В. ПУХНАЧЕВ,
кандидат физико-математических наук

И. Д. ДАНИЛОВ,
кандидат технических наук

**МИКРО-
КАЛЬКУЛЯТОРЫ
для ВСЕХ**



ПУХНАЧЕВ Юрий Васильевич — кандидат физико-математических наук, доцент Московского физико-технического института. Автор ряда научно-популярных книг по математике. Ведущий разделов «Человек с микроКалькулятором», «Человек и компьютер» в журнале «Наука и жизнь».

ДАНИЛОВ Игорь Данилович — кандидат технических наук, старший программист объединения «Росспецгеология». Выступает в научно-популярных журналах со статьями, пропагандирующими вычислительную технику индивидуального пользования, активно сотрудничает в разделах «Человек с микрокалькулятором», «Человек и компьютер» журнала «Наука и жизнь».

Рецензент: Савин А. П.— кандидат физико-математических наук

Пухначев Ю. В., Данилов И. Д.

Д 18 Микрокалькуляторы для всех.—М.: Знание, 1986.—192 с.—(Нар. ун-т. Естественнонаучный фак.).
55 к. 80 000 экз.

В век научно-технического прогресса вычислительная техника проникает буквально во все сферы человеческой деятельности. Сегодня микрокалькуляторы стали настущной необходимостью для всех — для рабочего и инженера, научного работника и экономиста.

Как научиться программировать? Каковы перспективы развития малой вычислительной техники? На эти вопросы в полулярной форме, иллюстрируя многочисленными примерами из практики, отвечают авторы книги, рассчитанной на слушателей и преподавателей народных университетов естественнонаучных знаний.

Д 2405000000-053 35-86
073 (02)-86

ББК 32.973

© Издательство «Знание», 1986 г.

В «Основных направлениях экономического и социального развития СССР на 1986—1990 годы и на период до 2000 года» говорится: «Высокими темпами наращивать масштабы применения современных высокопроизводительных электронно-вычислительных машин всех классов». Один из этих классов образуют программируемые микрокалькуляторы. Ни по быстродействию, ни по объему памяти им не сравниться с самыми мощными из современных ЭВМ. Однако у них есть свои достоинства. Они портативны, просты в обращении. Ими можно воспользоваться в любой момент для проведения относительно несложного расчета. А необходимость в этом то и дело возникает у работников самых различных специальностей — технологов и топографов, врачей и конструкторов... Микрокалькулятор способен по введенной в него программе решить алгебраическое или дифференциальное уравнение, вычислить интеграл или построить гистограмму, словом, выполнить довольно сложную вычислительную работу. Вот почему эти «карманные ЭВМ» за последнее время получили широкое распространение в нашей стране. Сегодня они приносят немалый экономический эффект.

Разумеется, для того, чтобы пользоваться программируемым микрокалькулятором, необходимо иметь определенные знания и навыки. Нужно знать возможности микрокалькулятора, приемы решения прикладных задач на вычислительных устройствах, способы разработки вычислительных алгоритмов и написания программ для микрокалькулятора. Успеху способствует опыт работы на этих машинах. Простые на первый взгляд, они имеют немало «секретов», которые раскрываются и осваиваются лишь по ходу практической работы на них.

Книга Ю. В. Пухначева и И. Д. Данилова «Микрокаль-

куляторы для всех» помогает читателю, желающему использовать в своей работе программируемый микрокалькулятор, приобрести необходимые для этого знания и навыки.

Написанная живо, понятно и содержательно, она способствует все более широкому внедрению электронно-вычислительной техники в разнообразные сферы производственной и учебной деятельности, а тем самым все более высокой эффективности этой деятельности.

Академик А. А. САМАРСКИЙ

Границы применения современных электронных вычислительных машин обширны и продолжают расширяться. Сегодня уже не вызовет сенсации сообщение о том, что компьютер управляет заводом или переводит книги с одного языка на другой. Но усложнение задач, решаемых с помощью ЭВМ,— это лишь одна сторона дела. Другая связана с тем, что электронные помощники облегчают сегодня труд работников самых разнообразных специальностей при решении многочисленных повседневных проблем. Понятно, что при этом нужны компьютеры с отнюдь не рекордными возможностями. Здесь ценятся устройства дешевые, несложные в управлении, портативные, а вместе с тем способные производить основные математические операции и вести вычисления по введенной в них программе. Именно таковы программируемые микрокалькуляторы.

Микрокалькуляторы занимают самую низшую ступеньку в иерархии современных компьютеров. Их отличительные особенности — компактность и невысокое быстродействие. Некоторые из них умеют выполнять лишь немногие единичные действия по командам, отдаваемым им нажатием на клавиши. Это простейшие микрокалькуляторы. Инженерные микрокалькуляторы отличаются от них более широким набором выполняемых команд, а программируемые — еще и способностью запоминать программы вычислений, необходимые для этого исходные данные и вести расчеты в автоматическом режиме.

Тот, кто умеет обращаться с программируемым микрокалькулятором, за считанные минуты решит систему уравнений, составит таблицу значений функций, подвергнет статистической обработке результаты эксперимента, не говоря уже о таких бытовых вычислительных задачах, как, скажем, расчет квартирной платы.

Помочь в освоении программируемого микрокалькулятора и самостоятельном составлении программ для него — цель данной книги. Она знакомит читателя с одним из самых распространенных в нашей стране программируемых микрокалькуляторов «Электроника Б3-34». Все, что рассказано в книге, относится также к микрокалькуляторам «Электроника МК-54» и «Электроника МК-56», поскольку по своему устройству они тождественны «Электронике Б3-34» и отличаются от нее лишь некоторыми обозначениями на клавишиах и формой корпуса.

Книга делится на две части. В первой говорится о том, какие операции выполнимы на «Электронике Б3-34», как составляются программы для нее, описываются методика решения задач на программируемом микрокалькуляторе, способы оценки точности получаемых результатов и т. д. Во второй части книги приводятся конкретные задачи, решаемые с помощью микрокалькулятора. Для каждой из них прослеживается весь путь от ее постановки до составления программы расчетов и получения результатов. Почти каждая программа снабжена комментарием, в котором разбираются примененные при ее составлении специфические приемы программирования. Знание этих приемов поможет в совершенстве овладеть микрокалькулятором.

Авторы книги в течение ряда лет участвовали в работе раздела «Человек с микрокалькулятором» в журнале «Наука и жизнь». Материалы раздела послужили отправной точкой при написании книги. Ее авторы выражают сердечную признательность читателям журнала, чьи выступления на его страницах обусловили тематику и направленность книги. Думается, что в силу такой обусловленности книга ответит на запросы многочисленных владельцев «карманных ЭВМ».

Как бы ни были ограничены возможности «карманных ЭВМ», решение задач на них проходит по тем же этапам, которые характерны для работы на любой вычислительной машине. Поэтому тот, кто освоит программируемый микрокалькулятор, легче овладеет искусством программирования на больших ЭВМ. Сегодня это искусство столь же необходимо любому специалисту, как умение читать и писать, ведь вычислительная техника в ближайшее время найдет применение во всех без исключения специальностях.



ШКОЛА ПРОГРАММИРОВАНИЯ

1. Знакомство с микрокалькулятором

Искусством программирования невозможно овладеть за один присест. Чтобы сохранить калькулятор для учебных упражнений и дальнейшей работы, надо строго соблюдать правила обращения с ним, оговоренные в инструкции, и прежде всего порядок его включения и выключения.

Необходимое ему для работы питание калькулятор может получать либо от размещенных внутри него аккумуляторов, либо от сети 220 вольт. В первом случае особой осторожности не требуется: чтобы включить калькулятор, переведите в крайнее правое положение тумблер в левом верхнем углу его панели, под индикатором. Тотчас на индикаторе загорится нуль с точкой: калькулятор готов к работе. (Если загорятся запятые, это значит, что заряд аккумуляторов иссякает.)

Если же вы намереваетесь пользоваться электроэнергией от сети, то тщательно следите за последовательностью своих действий: вначале к калькулятору подключается блок питания, затем блок питания включается в сеть и уже потом включается сам калькулятор.

Выключение калькулятора производится в обратной последовательности: сначала выключается он сам, потом от розетки отсоединяется блок питания, потом, если требуется, он отсоединяется от калькулятора. Промежуток времени между выключением и повторным включением калькулятора должен составлять не менее 10 секунд.

На панели калькулятора тридцать клавиш. На каждой простирано свое обозначение. Кроме того, обозначения есть и над клавишами, а в нижнем ряду клавиатуры — и под ними. Такое обилие связано с тем, что наш калькулятор может выполнять довольно много разных операций (около двухсот). Поэтому каждая клавиша предназначена для выполнения двух, а то и трех действий. Однако это не приводит к путанице. Опыт различия «что есть что» приведет очень быстро.

Если нажать на клавишу, то будет выполняться дейст-

вие, обозначенное на ней. Если же нажать сначала клавишу F и затем какую-то другую клавишу *, то будет выполнена операция, обозначенная над клавишой.

Например, если надо вычислить $\sqrt{5}$, то нажимаем клавиши

5 F $\sqrt{}$

На индикаторе загорается значение корня: 2,2360679.
(Обратите внимание: числа, выводимые на индикатор калькулятора, могут содержать до восьми разрядов.)

Отметим, что последняя из нажатых вами клавиш несет на себе символ «минус». Но вы, вероятно, не обратили на это внимание. Символ квадратного корня, написанный над нею, однозначно указал, что нажимать нужно было именно ее. Стало быть, вы уже приобретаете верную ориентацию в обозначениях на клавищах калькулятора. В дальнейшем мы всегда будем называть требуемую по смыслу операцию независимо от того, обозначена она на самой клавише, сверху или снизу от нее.

Отметим и то, что для вычисления корня калькулятору сначала было сообщено подкоренное выражение, пять, а уже потом указано требуемое действие (хотя, казалось бы, традиционная запись корня $\sqrt{5}$ диктует иначе: сначала $\sqrt{}$, потом 5). Такой обратный порядок характерен для команд, отдаваемых микрокалькулятору. К этому надо привыкнуть.

И еще примечание: действие клавиши F распространяется лишь на одну операцию, клавиша которой нажата сразу вслед. Для повторного выполнения «надклавишной» операции (только что выполненной или любой другой) необходимо снова нажать клавишу F, а уже затем требуемую, скажем, Ig или sin.

Бывает, что клавиша F нажата по ошибке. Ее действие отменяется клавишей CF. Нажмите ее, и если вслед за этим вы нажмете какую-то клавишу с обозначенной на ней операцией, то будет выполнена именно эта операция, а не та, что написана над клавишей.

При вычислении тригонометрических функций надо четко представлять себе, в каких единицах выражен угол — в радианах или в градусах. Соответственно надо устанавливать и переключатель Р — Г. При градусном

* Работающие на вычислительных машинах обычно говорят «нажать клавишу», а не «нажать на клавишу».

измерении минуты и секунды передаются на индикаторе десятичной дробью. Например, $17^{\circ}45'$ будет выражено так: 17,75 — ведь 45 минут составляют 0,75 от одного градуса.

С помощью переключателя Р — Г можно переходить от градусной меры к радианной, и наоборот. Например, переведем в радианы угол $37^{\circ}30'$. Чтобы ввести эти величины в калькулятор, превратим минуты в десятичную дробь: 37,5. Установив переключатель в положение Г, вводим это число последовательным нажатием клавиш

3 7 , 5

Заметьте: если число содержит не более восьми знаков, то ввести его проще всего именно так, как мы только что поступили,— набирая на клавиатуре его последовательные цифры и не забывая своевременно нажать на клавишу «запятая», если это нужно.

Возьмем от введенного числа функцию «синус», нажав клавиши

F Sin

На индикаторе читаем значение синуса: 0,60876144.

Переставим переключатель в положение Р и от результата предыдущей выкладки возьмем арксинус

F arcsin

На индикаторе читаем радианную меру нашего угла: 0,65449853.

Впрочем, так прочтет показания индикатора лишь тот, кто обладает известной сноровкой в обращении с выводимыми на него числами. На самом деле индикатор показывал сначала

6.0876144-0!

а потом

6.5449853-0!

Это связано с тем, что числа, по абсолютной величине меньшие 1 и большие 99999999, в микрокалькуляторе представляются в так называемой экспоненциальной форме (как

еще говорят, с плавающей запятой). Такой вид чисел применяется с помощью степеней десятки:

$$10^1=10 \quad 10^3=1000 \quad 10^7=10000000; \\ 10^{-1}=0,1 \quad 10^{-3}=0,001 \quad 10^{-7}=0,0000001$$

и т. д.

Любое число, как бы мало или велико оно ни было, можно представить в виде произведения двух сомножителей. В одном из них те же цифры, что и в исходном числе, но запятая стоит сразу после первой ненулевой (значащей) его цифры. Этот сомножитель называется мантиссой. Другой сомножитель — это 10 в некоторой степени (она называется порядком). Например:

$$23790000=2,379 \cdot 10000000=2,379 \cdot 10^7; \\ 0.00002379=2,379 \cdot 1/100000=2,379 \cdot 10^{-5}.$$

Использование такого представления, кроме чисто технических соображений, удобно во многих отношениях: благодаря этому калькулятор может работать воистину с гигантским диапазоном чисел — от 10^{-99} до почти 10^{100} . (Чтобы оценить по достоинству широту этого диапазона, примите во внимание такие цифры: масса протона около 10^{-24} г, а масса Вселенной оценивается в 10^{55} г).

На индикаторе порядок числа представлен двумя крайними правыми цифрами со знаком перед ними («+» не показывается). Все, что левее, — мантисса. Полученное в предыдущем примере число радиан перепишем с индикатора так:

$$6.5449853 \cdot 10^7 = 6.5449853 \cdot 10^7 = 0.65449853$$

К такому представлению числа привыкнуть совсем не трудно. Для начала можно пользоваться таким советом: если знак порядка отрицательный, то перед первой цифрой мантиссы напишите столько нулей, сколько указано в порядке, и поставьте запятую после первого нуля; если же порядок задан положительным числом, то на такое число цифр переносите запятую вправо — по исчерпании цифр мантиссы добавляйте нули:

Знак порядка	Порядок
2,375	-03
7,5732	05
Знак мантисса	числа

$0,002375$
 -5732000

Определить мантиссу и порядок числа для его ввода можно, используя наш совет «наоборот».

Если число начинается с нулей, то его порядок отрицательный и равен количеству нулей. Мантиссу получаем, смешав запятую вправо, чтобы она оказалась после первой ненулевой цифры, и отбрасывая нули. В остальных случаях получаем мантиссу, сдвигая запятую влево, чтобы она стояла после первой значащей цифры. Количество десятичных разрядов, пройденное запятой при таком сдвиге, даст порядок числа — разумеется, неотрицательный. Он равен нулю, если запятая с самого начала стоит после первой значащей цифры.

При вводе числа в калькулятор действуют так: вводят мантиссу, нажимают клавишу ВП (Ввод Порядка, при этом справа появляются два нуля) и вводят порядок. Если число отрицательное, то после набора мантиссы нажимают клавишу /—/. Если порядок отрицательный, то клавишу /—/ нажимают после ввода порядка.

Ради тренировки введем в калькулятор числа 0,00002375, — 237500 и —0,00002375. Нажатые клавиши указываем, как и прежде, символами в квадратных рамках, показания индикатора — записью, окруженной рамкой в форме параллелограмма.

Вводим 2,375	ВП	5	/—/	читаем 2,375 -05
Вводим 2,375	/—/	ВП	5	читаем -2,375 05
Вводим 2,375	/—/	ВП	5	читаем -2,375 -05

Отметим, что вводить числа в нормализованном виде приходится только тогда, когда без этого не обойтись. Например, если требуется ввести число 0,000034655789, то обычным образом можно ввести только 0,0000346: цифры, вводимые после восьми начальных, восприняты не будут. Представленное же в виде $3,465789 \cdot 10^{-5}$, оно вводится полностью.

Потренируйтесь сами в вводе чисел в экспоненциальной форме и умении их легко читать. Для этого используйте клавишу ↑; она автоматически нормализует набранное на клавиатуре число, меньшее единицы:

Вводим 0,0000857	↑	читаем 8,57 -05
------------------	---	-----------------

Вернемся к примеру с переводом угла из градусной меры в радианную. Мы уже выяснили, что $37,5^\circ = 0,6544449853$ ра-

диана. Теперь посмотрим, как совершается перевод из радианной меры в градусную. Установив переключатель в положение Р, наберем 0,65449853 и нажмем клавиши F sin. На индикаторе читаем 0,60876151. Установим затем переключатель в положение Г и нажмем клавиши F arcsin. На индикаторе читаем 37,500009.

Тот факт, что значения синуса, вычисленные нами прежде и теперь, совпадают не полностью и что исходный угол получен обратно с маленькой «добавкой», обычен для вычислительной техники и не должен смущать.

Любой ЭВМ свойственно делать ничтожные ошибки. Это происходит из-за целого ряда причин; например, в нашем случае из-за того, что функции $\sin x$ и $\arcsin x$ подсчитываются по довольно сложным программам, заложенным в машину. Каждая из этих программ допускает ошибку в вычислениях, нередко достигающую шестого знака после запятой. Но заметьте: мы четыре раза использовали эти программы да еще два раза делали перевод из одной меры в другую, что влечет за собой умножение и деление. Остается только удивляться, что после таких «приключений» исходное число возвратилось назад с завидной точностью.

Процесс работы микрокалькулятора, вычислявшего синус и арксинус, был хорошо заметен: в это время на индикаторе около секунды ничего не было. Если бы мы взялись вручную повторить все то, что делала наша миниатюрная вычислительная машина, по таким же формулам и с такой же точностью, нам потребовалось бы не менее рабочего дня.

Наш калькулятор, как и любая другая вычислительная машина, оперирует с числами. Важно понять, что при выполнении любого действия используемые числа не могут находиться в калькуляторе иным образом, как только записанными в память.

Числа запоминаются машиной в отведенных для этого ячейках, регистрах памяти.

Каждый регистр памяти в калькуляторе имеет свое обозначение в виде цифры или буквы. Десять из них обозначают начальными натуральными числами, от 0 до 9 включительно, еще четыре — буквами А, В, С, Д и еще пять — буквами Х, Y, Z, Т, XI. Группа последних пяти регистров существенно отличается от остальных, и об этом мы поговорим подробнее на следующем занятии.

В дальнейшем для краткости мы будем обозначать регистры сокращенно: РХ, Р4 или РА.

При вводе в калькулятор число заносится в регистр Х. От всех прочих он отличается тем, что его содержимое (то есть записанное в него число) видно на индикаторе. И наоборот: если на индикаторе видно какое-то число, то оно находится в РХ. Для прочного запоминания, скажем, допуская некоторое округление: индикатор и регистр Х — это одно и то же.

Все операции, при помощи которых вычисляются функции от некоторого числа, выполняются в нашем калькуляторе таким образом, что в качестве аргумента берется число из РХ и туда же помещается результат. Так было и с $\sqrt{5}$: после нажатия клавиш 5 и $\sqrt{}$ значение корня было занесено в РХ, то есть появилось на индикаторе.

Каждая такая операция (извлечение квадратного корня, возведение числа в квадрат, получение обратной величины от числа, вычисление синуса, косинуса, тангенса и обратных к ним функций, возведение в степень чисел 10 и e , вычисление логарифмов, десятичного и натурального) выполняется над одним числом, и потому все они называются одноместными. Сложнее структура у арифметических операций — сложения и вычитания, умножения и деления. Все они двуместные, каждая выполняется над двумя числами. Порядок построения команд и в этом случае обратный: сначала калькулятору сообщают оба числа, а потом символ операции, которую требуется над ними совершить.

Каким бы странным ни казался такой порядок, у него есть определенные преимущества по сравнению с привычной для нас записью арифметических операций. Записывая их, мы не можем обойтись без скобок. Уберите их, например, в выражении $3,5 \times (2,5 - 1)$, и оно превратится в $3,5 \times 2,5 - 1$, что приведет к другому результату. А команды для нашего калькулятора, как мы увидим чуть позже, в подобных случаях можно отдавать, не прибегая к скобкам.

Используемая в калькуляторных выкладках бесскобочная запись называется также польской, потому что впервые ее предложил польский ученый А. Лукасевич.

Числа, над которыми нужно совершить ту или иную арифметическую операцию, должны находиться в двух регистрах — РХ и РY (оттого их и называют операционными). Ход в первый из них нам уже знаком: вводимое в него число набирается на клавиатуре. Находящееся в РХ число можно переслать в РY нажатием клавиши \uparrow . При этом копия

переданного числа остается в РХ. Затем туда записывается второе число; бывшее там прежде автоматически стирается.

Порядок расположения обоих чисел в регистрах РХ и РY неважен, если их предстоит сложить или перемножить. В случае вычитания уменьшаемое должно находиться в РY, вычитаемое — в РХ. В случае деления в РY должно располагаться делимое, в РХ — делитель.

Введя числа в оба регистра, можно нажать клавишу выбранной операции. Результат ее будет помещен в РХ. То, что было прежде в РY, не сохранится.

Ради примера вычислим арифметические выражения, встретившиеся нам тремя абзацами выше. Итак, $3,5 \times (2,5 - 1)$. Начнем с выражения в скобках. Набираем на клавиатуре уменьшаемое 2,5 и нажимаем клавишу \uparrow . Число на индикаторе «мигнуло»: теперь оно переслано в РY, а его копия осталась в РХ. Вводим вычитаемое, единицу, и нажимаем клавишу «минус». Результат вычитания читаем на индикаторе: 1,5. Казалось бы, для выполнения умножения его нужно переслать в РY при помощи все той же клавиши \uparrow . Эту операцию, однако, можно сэкономить благодаря интересному свойству нашего калькулятора. Оказывается, если число на индикаторе является результатом некоторой операции, то оно передвигается в РY, когда в РХ вводится новое число.

Набираем на клавиатуре 3,5. Теперь все готово для умножения: в РY находится 1,5 и в РХ записано 3,5. Нажимаем клавишу со знаком умножения и читаем на индикаторе окончательный результат: 5,25.

Выполним вторую выкладку: $3,5 \times 2,5 - 1$. Продумайте последовательность нажатия клавиш и сравните ее с приведенной здесь:

3 1 5 \uparrow 2 + 5 \times 1 -

Совершите все эти действия и сверьте результат с истинным: 7,75.

Теперь, когда мы познакомились со всеми вычислительными действиями, на которые способен наш микрокалькулятор, нужно отметить, что любое из них выполнимо лишь при соблюдении определенных условий.

Некоторые из них вытекают из самого определения математических операций, теряющих смысл при нарушении

этих условий. Нельзя делить на нуль, извлекать квадратный корень из отрицательного числа, вычислять арксинус и арккосинус от числа, большего единицы, брать логарифм от неположительного числа, вычислять тангенс от числа вида $\pi/2 \pm \pi n$, возводить отрицательное число в произвольную степень. И если отдать микрокалькулятору команду о выполнении подобной некорректной операции, он остановится, и на индикаторе появится надпись ЕГГОГ (от английского *error* — ошибка).

Если результат какой-либо вычислительной операции превосходит $9,999999 \cdot 10^{39}$, он не может быть показан на индикаторе, где умещаются лишь восемь знаков мантиссы и два знака порядка. Калькулятор и в этом случае выдает сообщение ЕГГОГ.

Если же результат операции меньше 10^{-39} , то и тогда показать его нельзя, ведь его порядок выражается трехзначным числом. На индикаторе в таком случае появляется нуль (машинный нуль, как говорят тогда, желая подчеркнуть, что в действительности это число не нулевое, но не выводимое на индикатор).

Последними двумя обстоятельствами, в частности, ограничены диапазоны аргументов, над которыми совершаются те или иные одноместные операции, выполнимые нашим калькулятором. Эти диапазоны указаны в инструкции, прилагаемой к нему. Там же сказано, с какой погрешностью выполняется каждая из этих операций.

Вдумаемся в только что отмечавшуюся особенность нашего калькулятора: если в РХ ввести очередное число, то результат предыдущей операции передаст в РY. Вслед за этим можно выполнить действие с введенным в РХ числом и предыдущим результатом, находящимся в РY. Получившийся результат опять передаст в РY при наборе нового числа и так далее. Такие действия называются цепочечными.

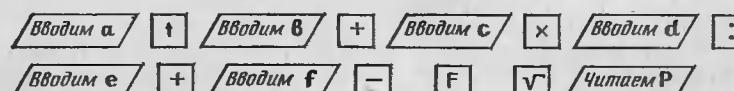
В качестве иллюстрации рассмотрим суммирование нескольких членов: 11, 12, 13... Набираем число 11 и нажатием клавиши \uparrow переводим его в РY, вводим 12, нажимаем клавишу «плюс» и читаем на индикаторе результат: 23. Набираем число 13. При этом предыдущий результат 23 сразу переходит в РY, и при нажатии клавиши «плюс» читаем сумму: 36. Теперь вводим 14... и так далее.

Вводим 11 \uparrow Вводим 12 + Читаем 23 // Вводим 13 + Читаем 36

Возьмем формулу посложнее:

$$P = \sqrt{\frac{(a+b) \times c}{d}} + e - f.$$

Советуем и на сей раз самостоятельно предложить порядок действий и уже затем проверить себя по приведенному здесь:



Вычислите значение P при таких исходных данных: $a=10, b=11, c=12, d=-156, e=15,2$ и $f=-348,3$. Проверьте, получилось ли $P=19,023265$.

На результаты промежуточных вычислений можно не обращать особого внимания, ограничившись лишь быстрым контролем. Например, если выполнялось деление положительного и отрицательного чисел и частное получилось положительным, значит, при вводе не был установлен знак числа. Напомним: он устанавливается после того, как введены все цифры числа.

Неверно введенное число можно ввести заново: стираем его нажатием клавиши Сх, набираем его вновь и продолжаем работу.

В отличие от результата арифметического действия результат одноместной операции не изменяет содержимого РУ. Если над этим результатом, находящимся в РХ, опять произвести одноместную операцию, то содержимое РУ опять не испортится. И так будет продолжаться до задания арифметической, двуместной операции.

Этот факт существенно облегчает проведение расчета по сложным формулам. Пусть, например, нам требуется вычислить гипotenузу прямоугольного треугольника c , у которого известны катеты a и b . Воспользуемся хорошо известной из геометрии теоремой Пифагора:

$$c = \sqrt{a^2 + b^2}.$$

Вычисления на калькуляторе по этой формуле сводятся к короткой цепочке действий

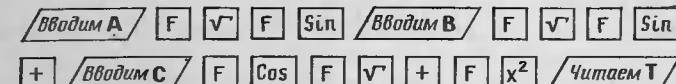


Проверьте, что при $a=3$ и $b=4$ получается $c=5$.

Теперь немного тригонометрии. Вот формула:

$$T = (\sin \sqrt{A} + \sin \sqrt{B} + \sqrt{\cos C})^2.$$

Порядок вычисления здесь может быть таким:



Для заключительной иллюстрации обратимся к пропорции золотого сечения, сыгравшей большую роль в истории математики и архитектуры. Выражающее ее число φ равно $(1+\sqrt{5})/2$. Вычислите его на калькуляторе. Должно получиться 1,6180339.

Известно несколько формул, позволяющих вычислить эту величину приближенно, но со сколь угодно высокой точностью. Например, такая:

$$\varphi_k \approx 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{2}}}}}}$$

Вычислите значение φ_k при небольшом k — четыре, пять, шесть — и сравните его с величиной φ . Небольшая подсказка: начинайте вычисления с конца формулы.

Повторите эту процедуру еще и еще раз, удлиняя дробь. Как видите, результаты все менее отличаются от φ .

Метод последовательных приближений широко и разнообразно применяется в вычислительной математике. На будущих занятиях мы еще встретимся с соответствующими примерами.

2. Команды микрокалькулятора

Команд, о которых говорилось на прошлом занятии, достаточно, чтобы приняться за составление какой-нибудь несложной программы. Например, такой, по которой калькулятор вычислит площадь круга с диаметром D .

Формула для площади круга хорошо известна: $S = \pi D^2/4$. Необходимые для расчета числа π и 4 имеются в калькуляторе, и их можно будет вызывать по ходу вычислений. Величину D надо ввести в регистр Х заранее. Пусть, например, она равна 3.

Нажата клавиша 3 и то же число на индикаторе, а значит, и в регистре Х.

Если ввести расчет по формуле вручную, то для этого, очевидно, далее надо нажать такие клавиши:

F x² F π × 4 :

На индикаторе читаем результат: 7,0685832.

Те же клавиши и в той же последовательности нужно будет нажать, когда мы станем вводить в калькулятор программу для вычисления площади круга. В таком естественном порядке, воспроизводящем ход расчетов вручную, в калькулятор вводится любая программа для прямых вычислений по формулам, включающим выполнимые на нем математические операции. В этом заключается значительное удобство.

Программа не может располагаться в калькуляторе иначе, как в виде отдельных команд, каждая из которых занимает свою ячейку программной памяти (а некоторые команды — даже две ячейки). Всего таких ячеек 98. Им присвоены номера, называемые адресами, — от 00 до 97.

Чтобы ввести программу в калькулятор, надо перевести его в состояние, называемое режимом программирования. Делается это нажатием двух клавиш:

F ПРГ

Тотчас же в правом углу индикатора загорится 00. Это значит, что счетчик адресов установлен на нулевую отметку и команда, которую мы сейчас введем, займет адрес 00.

Ввод каждой новой команды станет увеличивать на единицу содержимое счетчика адресов, отображаемое в правом углу индикатора.

Нажимаем клавишу F. На индикаторе ничего не изменилось. Нажимаем клавишу x². В левом углу загорается 22. Это код операции возведения в квадрат. Его появление на индикаторе означает, что команда занесена в программную память. Одновременно сменилось число в правом углу: сейчас там горит 01. По такому адресу разместится следующая введенная нами команда:

22

01

Нажимаем еще раз клавишу F, и вновь это не вызывает никаких перемен на индикаторе: клавиша F самостоятельного значения не имеет и образует команду лишь тогда, когда вслед за ней нажата еще одна клавиша.

Нажимаем клавишу π. Код 22 сдвинулся вправо, а на его месте появилось число 20 — код засылки числа π в регистр X. В правом углу — 02. По этому адресу разместится следующая команда: «умножить». Нажав соответствующую клавишу, замечаем: в левом углу оба кода разом сместились вправо, а на освободившемся месте загорелся код операции умножения: 12.

20 22 02
12 20 22 03 ~

Три кода выстроились на индикаторе от левого его края направо. Ввод каждой новой команды теперь будет приводить к тому, что левый и средний из этой тройки кодов смеются на одну позицию вправо, стирая правый, а на освобожденном ими месте появится код только что введенной операции. В правом же углу при этом загорится адрес, который будет занят командой, введенной вслед.

Вот как это происходит, когда мы вводим команды, по которым заканчивается вычисление площади круга:

04 12 20 04
13 04 12 05 ~

В ячейки памяти калькулятора программа записывается как последовательность кодов операций.

Когда калькулятор выполнит первую команду введенной в него программы, содержимое счетчика адресов автоматически увеличится на единицу и калькулятор приступит к выполнению второй команды; выполнив ее, перейдет к следующей... По исполнении команды, дающей окончательный результат вычислений, машину нужно остановить. Для этого в программе ставится команда С/П.

Нажав на клавишу с таким обозначением, завершаем ввод нашей программы. Текст ее запишем, указывая перед каждой командой ее адрес с точкой между ними:

00.Fx² 01.Fπ 02.×03.4 04.:05.C/P.

У клавиши С/П два назначения. Одно из них мы уже выяснили: если нажать ее в режиме программирования, то в программе появится команда останова. Чтобы познакомить-

ся с другим назначением клавиши, вернем калькулятор в состояние, в котором он был сразу после включения.

Это состояние мы будем называть режимом вычислений. Находясь в нем, калькулятор либо выполняет отдельные команды, отдаваемые ему нажатием клавиш (режим ручных вычислений), либо автоматически вычисляет по имеющейся в его памяти программе (автоматический режим). Запуск калькулятора на автоматический счет и производится клавишей С/П. Отсюда ее обозначение, расшифровываемое как «Стоп/Пуск».

Чтобы вернуть калькулятор в режим вычислений из режима программирования, надо нажать клавиши



Калькулятор готов к работе по программе. Но прежде чем запускать его, надо еще сообщить ему, с какой команды он должен начать вычисления.

Начальная команда введенной нами программы располагается по адресу 00. Калькулятор снабжен клавишей В/0, по сигналу которой счетчик адресов в режиме ручных вычислений возвращается на нулевую отметку — как говорят программисты, управление передается на адрес 00. Отсюда и обозначение В/0 («Возврат на 0»).

(Это не единственное назначение клавиши В/0 — в свое время мы познакомимся и с другими.)

Итак, установим счетчик адресов на отметку 00. Введем в регистр X диаметр круга (пусть он по-прежнему равен трем). Запускаем калькулятор клавишей С/П — и он за несколько секунд выполнит одну за другой все команды нашей крохотной программы. На индикаторе — площадь круга с диаметром 3, уже известная нам: 7,0685832.

Подсчитав на калькуляторе площадь круга вручную и по программе, иной читатель может разочарованно подумать, что никаких выгод по сравнению с ручным счетом программирование нам не принесло. Чтобы ввести программу, потребовалось нажать не только те же восемь клавиш, что и при работе вручную, но еще и семь «лишних»: F и ПРГ, чтобы перейти в режим программирования; С/П, чтобы калькулятор знал, где остановиться; F и АВТ, чтобы вернуться в режим вычислений; В/0 и С/П, чтобы запустить программу на счет.

15 нажатий против 8. Конечно, такое сопоставление не в пользу программирования, если речь идет о единичном

расчете по заданной формуле. Но представьте, что вам нужно определить площади нескольких кругов, скажем, с целочисленными диаметрами от 1 до 20 включительно. Сколько раз придется для этого нажать на клавиши?

При ручном счете — 171 раз (можете проверить). А при счете по программе — всего 83 раза. В самом деле, 15 нажатий уйдет на то, чтобы получить первый результат. Но для получения каждого следующего нужно лишь после очередного останова вводить новое значение диаметра (одно нажатие, если число однозначное, два — если двузначное) и запускать программу на новый счет клавишами В/0 и С/П. Программа, записанная в память калькулятора, сохраняется неизменной, покуда он не выключен. Для нового использования ее не нужно вводить заново.

83 против 171 — в таком сопоставлении выигрыш от программных расчетов налицо.

Он заметен тем отчетливее, чем больше программа и чем чаще приходится проводить по ней повторяющиеся вычисления. Так оно бывает, например, при составлении таблиц. А методы последовательных приближений, столь характерные для вычислительной математики? В них каждое приближение выполняется одной и той же последовательностью команд.

Большие программы отличаются еще и тем, что в вычислениях по ним, как правило, используется много исходных данных. Хранят их в так называемых адресуемых регистрах памяти, обозначаемых цифрами (от 0 до 9) и буквами (А, В, С, Д). Нетрудно подсчитать, что всего их 14.

Числа направляются на хранение в эти регистры командами засылки. Перед выполнением такой команды засыпаемое в память число сначала должно оказаться в регистре X (в результате набора на клавиатуре, вызова или выполнения какой-либо операции), и уже оттуда оно направляется в адресуемый регистр. Извлечь число из адресуемого регистра можно командой вызова; при этом оно направляется опять-таки в регистр X.

Команда засылки выполняется с помощью клавиши П (от слова «Память»), команда вызова — с помощью клавиши ИП («Из Памяти»). На других калькуляторах эти команды обозначаются иначе — см. таблицу 1.

Освоим обе команды сначала в ручном исполнении. Наберем на клавиатуре какое-нибудь число, например 3. Оно

Таблица I

	МК-54 МК-52	МК-56 МК-61	В книге
ИП	$\Pi \rightarrow x$	ИП	
П	$x \rightarrow \Pi$	П	
\leftrightarrow	\leftrightarrow	\leftrightarrow	
\uparrow	$B\uparrow$	\uparrow	
\overleftarrow{W}	\overleftarrow{W}	ШГ вправо	
\overleftarrow{W}	\overleftarrow{W}	ШГ влево	
\div	\div	:	
○	○	F0	
\arcsin	\sin^{-1}	Farcsin	
\arccos	\cos^{-1}	Farcos	
\arctg	tg^{-1}	Farctg	

тотчас появляется на индикаторе и, стало быть, находится сейчас в регистре X.

Допустим, мы хотим поместить его в регистр 4. Нажимаем клавишу П и затем клавишу 4. Число на индикаторе мигнуло: теперь оно и в регистре X, и в регистре 4. То, что было раньше в регистре 4, пропадает.

Очистим регистр X, нажав клавишу Сх. На индикаторе—нуль. Затем нажимаем клавиши ИП и 4. На индикаторе снова 3: число переписано из регистра 4. Но и там оно не пропало; в этом можно убедиться, вторично очистив индикатор и нажав клавиши ИП 4.

Опробуем теперь те же операции в программном варианте. Предположим, что в некоторой большой программе по ходу расчета потребовалось вычислить площадь круга. Его диаметр берется из регистра 4 (напомним, что мы уже поместили туда число три), а найденная площадь засыпается в регистр 5. Допустим, что цепочка команд, выполняющих этот расчет, должна размещаться в программе, начиная с адреса 10.

Устанавливаем счетчик адресов на нужный адрес в режиме ручных вычислений с помощью клавиши БП («Безсловенный Переход»). Нажимаем сначала ее, а потом набираем на клавиатуре 10. Это число будет гореть в правом углу индикатора, когда нажатием клавиш F ПРГ мы переведем калькулятор в режим программирования. Именно по 10-му адресу запишется команда, которую мы введем первой.

Ею, очевидно, диаметр круга должен вызываться из

регистра 4 в регистр X. Речь, стало быть, идет о команде ИП4.

Нажимаем клавишу ИП. На индикаторе никаких перемен. Нажимаем клавишу 4. В левом углу загорается 64. Это код операции вызова в регистр X содержимого регистра 4. То, что он загорелся лишь после нажатия второй клавиши, свидетельствует: клавиша ИП самостоятельного значения не имеет и образует команду вызова лишь в сочетании с другой клавишей, обозначающей номер регистра, содержимое которого вызывается в регистр X.

Вводим далее команды, по которым определяется площадь круга. Затем нажимаем клавиши П и 5: ими в программу вводится команда засылки результата в регистр 5. Наблюдая в это время за индикатором, можно убедиться, что клавиша П также не имеет самостоятельного значения и образует команду засылки в сочетании с нажимаемой далее клавишей, цифровой или буквенной.

Завершим ввод нажатием клавиши С/П. Вот текст введенной нами программы:

10.ИП4 11.Fx² 12.Fп 13.× 14.4 15.: 16.П5 17.С/П.

Клавишами F АВТ возвращаем калькулятор в режим вычислений. Клавишами БП 10 устанавливаем счетчик адресов на 10-й адрес. Клавишей С/П запускаем программу. Через несколько мгновений вычисления заканчиваются и на индикаторе загорается знакомое нам число 7,0685832. Нажав клавиши ИП 5, можно убедиться, что оно размещается в регистре 5. А заодно и в том, что в момент останова по команде С/П в регистре X находится результат выполнения предыдущей команды.

Выгоды от программирования мы могли оценить уже на примере, где требовалось провести многократные однотипные расчеты при меняющихся исходных данных. Такие задачи на практике встречаются нередко. Но при этом часто требуется не только получить ряд результатов, но и подвергнуть их статистической обработке, например, вычислить их среднее арифметическое.

Предположим, что подобным требованием дополнена наша задача о вычислении площади кругов с целочисленными диаметрами от 1 до 10: в довершение расчетов нужно определить среднюю площадь. Программу придется расширить: вычислив площадь очередного круга, следует прибавить ее к сумме площадей, попутно надо подсчитывать

количество кругов, а когда все они будут перебраны, полученную сумму нужно будет поделить на это количество.

Обе эти величины можно шаг за шагом накапливать в адресуемых регистрах: скажем, количество кругов — в нулевом, сумму их площадей — в первом. Получив по составленной нами программе площадь очередного круга в регистре X , можно вызвать туда из регистра 1 накопленную на этот момент сумму площадей. Содержимое регистра X при этом сдвинется в регистр Y . Затем остается выполнить сложение — так к сумме площадей прибавится новое слагаемое. Результат сложения, как мы знаем, получится в регистре X . Оттуда его нужно отослать в регистр 1.

Подобным же образом, вызвав содержимое регистра 0, затем единицу и сложив оба числа, получим пополнившееся количество кругов и отошлем его обратно в регистр 0 — регистр-счетчик.

К этой цепочке команд надо приписать еще команду останова и команду возврата на нулевой адрес. Когда счет остановится по первой из них, надо ввести в калькулятор диаметр нового круга и нажать клавишу С/П. Программа передаст к следующей команде, совершив по ней возврат на адрес 00 и проведет расчет для нового круга.

Чтобы рассчитать среднее арифметическое, надо приписать к нашей программе еще четыре команды: вызов содержимого регистра 1 (сумма площадей), вызов содержимого регистра-счетчика 0 (количество кругов), деление (оно даст интересующую нас среднюю площадь) и останов (на индикаторе будет светиться результат предыдущей операции, то есть ответ на поставленную задачу). На первую из этих команд мы передадим управление с помощью клавиши БП, когда переберем все круги, и после останова считаем искомый ответ с индикатора.

Это описание нетрудно перевести на язык команд. Но прежде надо решить, как организовать возврат на нулевой адрес. Дело в том, что команда В/0 передает управление на нулевой адрес только в режиме ручных вычислений. Стало быть, вместо нее в программе придется использовать команду безусловного перехода — ту самую, которую мы уже исполняли недавно с помощью клавиши БП.

У этой команды своя тонкость. Когда мы станем вводить ее в программу (забегая вперед, скажем, что счетчик адресов к этому моменту будет показывать 16) и нажмем клавишу БП, содержимое счетчика адресов тотчас же увеличится на единицу: 17. Когда же мы наберем адрес перехода,

содержимое счетчика адресов увеличится еще раз: 18. Это означает, что команда безусловного перехода «двойная», то есть занимает в программе два адреса. В первом из них записывается операция безусловного перехода БП, во втором — адрес перехода.

Отказавшись от команды В/0, мы свободны в выборе этого адреса. Учтем, что перед запуском программы следует очистить регистры 0 и 1, предназначенные для накопления сумм. Очистку можно произвести вручную, нажав клавиши Сх П 0 П 1. Но лучше, чтобы это сделал калькулятор, и при этом с самого начала: 00. Сх 01. П0 02. П1 03. С/П.

Команда останова здесь не лишняя да и в дальнейшем пригодится: на нее-то мы и станем передавать управление, вычислив площадь очередного круга, и покуда калькулятор стоит, будем вводить диаметр следующего круга.

С учетом высказанных соображений программа примет такой вид: 00.Сх 01.П0 02.П1 03.С/П 04.Fx² 05.Fп 06.× 07.4 08.: 09.ИП1 10.+ 11.П1 12.ИП0 13.1 14.+ 15.П0 16.БП 17.03 18.ИП1 19.ИП0 20.: 21.С/П.

Программа получилась объемистая — будем внимательны при ее вводе. Введя, вернемся в режим вычислений нажатием клавиш F АВТ, затем нажмем клавиши В/0, С/П. Очистив за три первые команды регистры 0 и 1, калькулятор остановится. Введем диаметр первого круга, единицу, запустим программу, после останова введем диаметр второго круга, двойку, запустим программу вновь... и так будем продолжать, пока не будет вычислена площадь последнего круга, чей диаметр равен десяти. Передадим управление на адрес 18, нажав для этого клавиши БП 1 8, и после нового пуска получим на индикаторе искомую среднюю площадь кругов: 30,237829.

Тот, кто знакомится с программированием на предлагаемых нами примерах, вряд ли представит себе, что ответ на поставленную нами задачу можно получить по более короткой программе. Между тем программисты не зря говорят: всякую программу можно сократить по крайней мере на одну команду. Сейчас мы докажем это на примере нашей задачи. Правда, для этого нам придется освоить несколько новых команд.

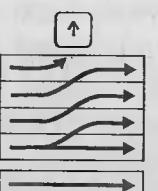
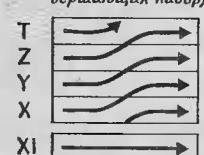
Когда на прошлом занятии, поговорив об одноместных операциях, мы перешли к рассмотрению двуместных, мы сразу отметили, что для их выполнения необходимы два операционных регистра — X и Y . Мы узнали, как с по-

мощью клавиши \uparrow в них вводятся участники арифметических действий и где размещается результат.

Ради простоты мы умолчали тогда, что перемещение чисел охватывает при этом не только регистры X и Y, но также Z и T. Эти четыре регистра находятся в тесной взаимосвязи и объединяются названием «стек» (английское слово, означающее «охапка, стопка»). К ним примыкает регистр X1, куда после выполнения многих операций направляется прежнее содержимое регистра X.

То, что мы узнали прежде, дополним здесь до завершенной картины всевозможных перемещений чисел по стеку.

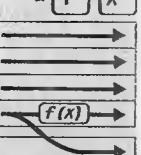
Вызов из памяти, набор на клавиатуре после выполнения операции (кроме \uparrow и Cx, завершающих набор)



Продолжение набора на клавиатуре после



Выполнение одноместной операции и F x^y



Посмотрим сначала, что происходит в нем, когда в регистре X засыпается новое число. Пусть оно вызывается туда из какого-то адресуемого регистра (командами ИП1, ИПА и т. п.) или из ячейки, где хранится число «пи» (командой Fл). Тогда прежнее содержимое RX смещается в PY, содержимое PY — в PZ, содержимое PZ — в PT, прежнее содержимое регистра T пропадает. И если представить регистры стека один над другим (X внизу, T вверху — см. рисунок), то описанное перемещение удобно назвать движением снизу вверх. Содержимое регистра X1 при этом не меняется.

Когда новое число вводится в регистр X с клавиатуры, числа в стеке тоже перемещаются снизу вверх, за исключением случаев, когда перед этим с клавиатуры вводилось какое-то число и вслед за его набором были выполнены команды \uparrow или Cx. Тогда при вводе нового числа в RX с остальными регистрами стека и с регистром RX1 ничего не происходит.

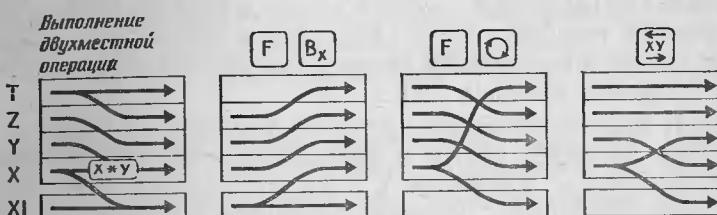
Если после набора числа на клавиатуре не нажать клавишу \uparrow , то при дальнейшем нажатии цифровых клавиш все будет происходить так, как если бы мы продолжали вводить цифры предыдущего числа.

Кстати сказать, аналогичным образом можно ввести число в регистр X в режиме автоматических вычислений по программе. Например, цепочка команд 00.3 01., 02.1 03.4 запишет в регистр X число 3,14.

По команде \uparrow числа, находившиеся в стеке, сдвигаются по его регистрам снизу вверх, причем в RX остается копия числа, находившегося там прежде.

По команде Cx содержимое регистра X стирается (точнее, заменяется нулем), а в прочих регистрах стека и в регистре X1 все остается по-прежнему.

Когда над числом, находящимся в регистре X, совершается какая-либо одноместная операция, в регистрах Y, Z, T также ничего не происходит, а в регистр X1 отправляется прежнее содержимое регистра X. Таким же образом выполняется и операция x^y .



Каждая арифметическая операция выполняется так: ее участники берутся из регистров X и Y, результат направляется в регистр X. Прежнее содержимое регистра X отсылается в регистр X1, прежнее содержимое регистра Y утрачивается — туда спускается содержимое PZ. Содержимое PT, в свою очередь, смещается в PZ, оставляя на прежнем месте свою копию.

При засылке числа из регистра X в любой адресуемый регистр сохраняется прежнее содержимое и регистра X, и всех других регистров стека, и регистра X1.

Вот еще три команды, по которым движутся числа по регистрам стека.

Команда FBx вызывает содержимое регистра X1 в регистр X. Числа в регистрах стека при этом сдвигаются снизу вверх. В регистре X1 остается копия находившегося там прежде числа.

По команде FQ содержимое PY переходит в RX, содержимое RX — в PY и RX1. С ее помощью удобно выводить на индикатор содержимое регистра Y.

Команда FO осуществляет круговое движение чисел по

стеку: из PY — в PX, из PZ — в PY, из PT — в PY, из PX — в PT и PX1. С ее помощью можно просмотреть на индикаторе содержимое всех регистров стека. Эта команда оказывается полезной и тогда, когда число, введенное в регистр X для какой-то промежуточной операции, сдвигнет вверх прежнее содержимое PX, PY и PZ. Восстановить прежнее размещение чисел по этим регистрам можно, выполнив команду F0.

Научившись управлять движением чисел по стеку, мы сможем сократить последнюю из составленных нами программ на целые четыре команды:

00.Cx 01.↑ 02.C/П 03.Fx² 04.Fπ 05.× 06.4 07.: 08.+ 09.→ 10.1 11.+ 12.→ 13.БП 14.02 15.→ 16.: 17.C/П.

Здесь начальные команды Cx и ↑ очищают регистры X и Y перед последующим накоплением количества кругов и суммы их площадей. После останова в регистр X вводится диаметр первого круга, калькулятор запускается вновь, после останова вводится диаметр следующего круга и снова нажимается клавиша C/П...

К моменту очередного останова в регистре X находится сумма площадей кругов с первого по некоторый k -й (обозначим ее Σ_k), в регистре Y — их количество k . В этом нетрудно убедиться с помощью таблицы, в последовательных столбцах которой указаны адреса, команды, коды операций, а также содержимое регистров стека после выполнения каждой команды.

Первые два столбца заполним по тексту программы. Ручные операции ввода (когда в PX заносится диаметр нового круга) и последующего пуска программы клавишей C/П запишем в таблицу без адресов.

Проходя по строчкам таблицы сверху вниз, заполняйте остальные столбцы в соответствии с результатами выполнения команд, и при вычислении площади ($k+1$ -го круга она примет вид, подтверждающий, что программа работает так, как мы предполагали (см. стр. 29; прочерк означает, что содержимое регистра сохранилось неизменным.)

Попробуйте продолжить эту таблицу до завершающего адреса 17, проверьте, что к моменту останова по этому адресу в регистре X (и на индикаторе) находится искомое среднее арифметическое.

Ведите программу в калькулятор, проведите расчет с теми же исходными данными, что прежде, и убедитесь, что результат получается тот же: 30,237829.

Адрес	Команда	Код операции	Содержимое регистров				
			X	Y	Z	T	X1
02	C/П	50	Σ_k	k			k
	Вход D _{k+1}		D _{k+1}	Σ_k	k		—
	Пуск		—	—	—	—	—
03	Fx ²	22	D _{k+1} ²	—	—		D _{k+1}
04	Fπ	20	π	D _{k+1} ²	Σ_k	k	—
05	X	12	πD _{k+1} ²	Σ_k	k	—	π
06	4	04	4	πD _{k+1} ²	Σ_k	k	—
07	:	13	πD _{k+1} ² /4	Σ_k	k	—	4
08	+	10	Σ_{k+1}	k	—	—	πD _{k+1} ² /4
09	→	14	k	Σ_{k+1}	—	—	Σ_{k+1}
10	1	01	1	k	Σ_{k+1}	—	Σ_{k+1}
11	+	10	$k+1$	Σ_{k+1}	k	—	1
12	↔	14	Σ_{k+1}	$k+1$	—	—	$k+1$
13	БП	51	—	—	—	—	—
14	02	02	—	—	—	—	—

Заметим, что подобные таблицы очень облегчают разбор программ — и своих, и особенно составленных другими.

Попытаемся теперь переделать нашу последнюю программу так, чтобы уменьшить количество ручных операций. Ведь чем их больше, тем выше риск ошибки.

Программа распадается на две части. В первой (адреса 00—14) вычисляется площадь очередного круга, подсчитывается сумма площадей всех перебранных кругов и их количество. Во второй части (адреса 15—17) вычисляется среднее арифметическое площадей. По какой цепочки команд пойти, определяем мы сами, в первом случае — просим запускать программу после очередного останова, во втором — совершая безусловный переход на адрес 15.

Нельзя ли поручить выбор пути самому калькулятору?

Вопросы такого рода могут возникнуть при составлении многих программ, даже весьма несложных. Например, при решении квадратного уравнения: если его дискриминант положительный или равен нулю, то корни отыскиваются по одним формулам, если отрицательный — то по другим. Или возьмем пример из тригонометрии: при положительном аргументе арккосинус связан с арксинусом одним соотношением, при отрицательном — другим. И если по ходу работы программы может возникнуть любая из двух каких-то альтернативных возможностей, программа должна

содержать фрагменты для расчета каждого варианта и работать по тому или другому в зависимости от выполнения или невыполнения некоторого условия. Фрагменты такого характера называют ветвями.

На эти случаи в нашем калькуляторе предусмотрены команды условного перехода. Они вводятся в программу с помощью клавиш, над которыми написаны соотношения: $x < 0$, $x = 0$, $x \geq 0$, $x \neq 0$. Судя по этим обозначениям, проверка любого альтернативного условия должна быть сведена к сравнению с нулем содержимого регистра X.

В силу «надклавищного» расположения надписей ввода команд условного перехода в программу предваряется нажатием клавиши F. Нажав вслед за ней клавишу, помеченную выбранным соотношением (скажем, $x \neq 0$), замечаем, что счетчик адресов увеличился на единицу. Он вновь увеличится на единицу, когда затем мы введем адрес перехода. Как видим, каждая команда условного перехода двойная, занимает в программе два адреса. Например: 03.Fx \neq 0 04.17.

На указанный в такой команде адрес перехода управление передается, если условие не выполняется. Так в нашем примере, если содержимое регистра X равно нулю и неравенство $x \neq 0$ не соблюдается, совершается переход на адрес 17. Если же условие выполняется, то есть $x \neq 0$, управление передается на адрес, следующий за командой условного перехода, в нашем примере — на адрес 05.

Для иллюстрации подобных переходов мы не случайно привели фрагмент 03.Fx \neq 0 04.17. Оказывается, если вставить его в нашу программу, она будет гораздо удобнее:

00.Cx 01.↑ 02.С/П 03.Fx \neq 0 04.17 05.Fx² 06.Fп 06.× 08.409.: 10.+ 11.⇄ 12.1 13.+ 14.⇄ 15.БП 16.02 17.F○ 18.⇄ 19.: 20.С/П.

Покуда мы будем вводить в калькулятор один за другим диаметры кругов (выраженные, разумеется, ненулевыми числами), команда условного перехода будет передавать управление на адрес 05, начальный адрес первой части программы, где вычисляются и суммируются площади кругов, подсчитывается их количество. А когда расчет этих величин закончится, введем нуль. Поскольку условие $x \neq 0$ теперь не выполнено, программа перейдет на адрес 17, начальный адрес второй части, где будет вычислена средняя площадь кругов.

По адресу 17 в программу, как видно, вставлена новая команда: F○. Необходимость в ней возникает потому, что

нуль, введенный в регистр X для перехода на заключительную часть программы, нарушил нужное нам расположение величин в стековых регистрах: суммарная площадь кругов сместилась из РХ в РY, количество кругов — из РY в РZ. Команда F○ возвращает эти величины на свои места.

Теперь для перехода к заключительной стадии расчета надо нажимать всего две клавиши, 0 и С/П, вместо четырех — БП, две цифры адреса перехода, С/П. Не нужно вспоминать или сверяться по инструкции к программе, каков этот адрес. Заботы об этом калькулятор берет на себя.

Наш учебный пример с определением средней площади круга, конечно, слишком прост, чтобы иметь самостоятельное значение. Но вполне может статься, что подобный расчет придется выполнить по ходу решения какой-то крупной задачи. И тогда в программу для ее решения цепочка команд, вычисляющих среднюю площадь, войдет в качестве фрагмента.

Допустим, он должен разместиться в большой программе, начиная с 10-го адреса. Предположим, что к началу его работы диаметры кругов уже находятся в адресуемых регистрах: D₁ — в РД, D₂ — в РС, D₃ — в РВ, D₄ — в РА, D₅ — в Р9, D₆ — в Р8 и так далее. (Такой «обратный» порядок обнаружит свой смысл позже.) Предположим еще, что длина массива заполненных регистров известна и не превышает 12, так что регистр I остается свободным, а выражающее эту длину число кругов находится в регистре 0.

(Эти облегчающие предположения не поблажка читателю, они естественны для программ, составленных опытным программистом, который всегда думает об удобствах своей работы.)

Подсчет средней площади кругов теперь представляется совсем простым: вызывать один за другим диаметры кругов из последовательных регистров, вычислять по ним и суммировать площади кругов, а затем поделить сумму на число слагаемых. Но для такой процедуры нужны соответствующие команды. Во-первых, такая команда условного перехода, по которой он совершается заданное число раз. Во-вторых, такая команда вызова, которая при многократном ее исполнении вызывает в регистр X числа из последовательных адресуемых регистров.

Нужные нам команды условного перехода вводятся в программу с помощью клавиш F L0, F L1, F L2, F L3. Будем объединять эти пары клавиш обозначением FLM. Как и при

вводе всякой команды условного перехода, нажав такую пару, мы должны далее ввести адрес перехода. В программе такая команда займет два адреса: в первом — операция FLM, во втором — две цифры, означающие адрес перехода.

Допустим, он меньше того, под которым записана операция FLM, а в регистре M находится целое число N, большее единицы. Выполнив цепочку команд, предшествующих операции FLM, программа приступает к выполнению команды условного перехода. Операцией FLM из содержимого регистра M вычитается единица и полученная разность сравнивается с нулем. Если она не равна нулю, то засыпается в регистр M, а управление передается по адресу перехода. Цепочка тех же команд выполняется еще раз, снова из содержимого регистра M вычитается единица... Так продолжается до тех пор, пока результат вычитания не окажется равным нулю. В таком случае он не засыпается в регистр M, и там остается единица; управление же передается на команду, следующую за командой условного перехода, то есть через адрес от операции FLM.

Это происходит, как нетрудно установить, после N-кратного прохождения цепочки. Засыпая в регистр M нужное число, мы можем задавать количество таких прохождений.

Фрагмент программы, выполняемый много раз подряд, называется циклом. Поэтому операция FLM, образующая только что описанную команду условного перехода, называется операцией организации цикла.

Есть среди команд нашего калькулятора и нужная нам команда вызова чисел из последовательных регистров. В программу она вводится нажатием трех клавиш: сначала K, затем ИП, затем той, что указывает номер одного из адресуемых регистров. Условно обозначим этот номер M и предположим, что в нем находится целое число.

Несмотря на обилие клавиш, требуемых для ее ввода, эта команда занимает в программе один адрес и записывается слитно: КИПМ. Работает она по-разному в зависимости от значения M.

Пусть M равно одному из чисел от 0 до 3 включительно. Тогда по команде КИПМ содержимое регистра M уменьшается на единицу и в RX вызывается содержимое того регистра, номер которого равен получившейся разности.

Пусть M равно одному из чисел от 4 до 6 включительно. Тогда по команде КИПМ содержимое регистра M увеличивается на единицу и в RX вызывается содержимое того регистра, номер которого равен получившейся сумме.

Пусть M представляет собой число от 7 до 9 или букву от А до Д. Тогда по команде КИПМ в RX вызывается содержимое того регистра, номер которого равен числу, находящемуся в регистре M.

Это число может превышать 9. Скажем, если оно равно 10, то калькулятор поймет его как обозначение регистра A, 11 — как B, 12 — как C, 13 — как D. Именно 13 получится, например, в регистре 1 при первом выполнении команды КИП1, если до того там находилось 14.

Команды вида КИПМ называются командами косвенного вызова. Есть у нашего калькулятора и сходные с ними по структуре команды косвенной засылки. Набираются они нажатием клавиши K, П и еще одной, указывающей номер некоторого регистра M. Каждая занимает в программе один адрес.

Работают они аналогично командам косвенного вызова и тоже в предположении, что в регистре M находится целое число. Вот, скажем, что происходит по команде КП1: из содержимого регистра 1 вычитается единица, и в регистр, номер которого равен получившейся разности, направляется содержимое регистра X.

Кстати, приведем соответствующий термин: уменьшение или увеличение содержимого каких-либо регистров при использовании команд косвенного вызова и косвенной засылки называется модификацией адреса.

Располагая командами косвенного вызова и организации циклов, нетрудно составить задуманный нами фрагмент:

10.ИП0 11.1 12.4 13.П1 14.Сх 15.КИП1 16.Fx² 17.Fп
18.Х 19.4 20.: 21.+ 22.FL0 23.15 24.⇄ 25.:

Начертите таблицу, с помощью которой можно следить за движением чисел по стеку, и проанализируйте по ней, как «работает» только что составленная нами цепочка команд.

Команда ИП0 вызывает из регистра 0 в регистр X количество кругов: оно понадобится в самом конце для нахождения среднего арифметического. Три дальнейшие команды засыпают в регистр 1 число 14, используемое в работе следующей команды КИП1, и очищают регистр X; в нем далее будет накапливаться сумма площадей. При первом выполнении команды КИП1 из начального содержимого регистра 1, то есть из числа 14, вычитается единица и получится 13 — номер регистра D. Из него-то и будет вызвано значение первого диаметра. Цикл команд, записанных под адресами 15—21, в первый раз вычислит площадь первого круга

га, во второй раз — уже площадь второго круга, ведь при втором исполнении команды КИП1 из содержимого ячейки 1 вновь вычитается единица и получится 12 — номер регистра С. Там находится значение второго диаметра — его-то и вызовет на сей раз команда КИП1 в регистр Х. Команда FL0 15 (адреса 22—23) будет вновь и вновь передавать управление на начало цикла, и каждый раз из содержимого регистра 0, куда вначале было записано количество кругов, будет вычитаться по единице. Сколько кругов, столько раз и будет пройден цикл. По выходе из него в регистре Х — сумма площадей кругов, в регистре Y — их количество. Их надо вначале переставить (адрес 24), чтобы затем поделить (адрес 25) и получить тем самым искомую среднюю площадь.

В разобранном нами фрагменте программы нет команды останова. Можно предположить, что полученный результат будет незамедлительно использован в следующей части программы. Но возможно и другое толкование: расчет, производимый этим фрагментом, требуется проводить в различных местах «большой» программы. Писать его в тексте программы несколько раз неэкономно. Лучше каждый раз передавать на него управление и возвращаться обратно.

Для организации таких переходов с возвратом служит клавиша ПП («Переход к Подпрограмме»). Вслед за ее нажатием нужно ввести в программу начальный адрес фрагмента, на который нужно передать управление. Такой фрагмент, реализующий некоторый самостоятельный алгоритм, и называется подпрограммой. Команда перехода к нему занимает два адреса. На следующий за ними адрес произойдет возврат, если в конце подпрограммы поставить команду В/0.

Судя по сказанному до сих пор, система команд у «Электроники Б3-34» весьма богатая: мы сумели выполнить все свои замыслы при составлении и совершенствовании задуманных программ. Некоторые возможности нашего микрокалькулятора при этом даже остались неиспользованными и неописанными.

Мы ничего не говорили, например, про клавиши ШГ со стрелками вправо и влево, которые с каждым своим нажатием соответственно увеличивают или уменьшают на единицу содержимое счетчика адресов, про клавишу НОП, позволяющую стирать в программе лишние команды (если нажать клавиши К и НОП, то в программе появится команда, не совершающая никаких действий, но занимающая

один адрес), про работу клавиши ПП в режиме ручных вычислений (нажимая ее раз за разом, мы заставляем калькулятор выполнять одну за другой команды введенной в него программы, демонстрируя результат выполнения каждой команды на индикаторе). Обо всем этом у нас будет разговор на следующем занятии, когда речь пойдет об отладке сложных программ.

Есть у «Электроники Б3-34» своеобразные варианты и аналоги описанных выше команд. Например, нажатием соответствующих клавиш можно набрать команды КП↑ и КИП↑. Работают они совершенно аналогично командам КП0 и КИП0, отличаясь лишь тем, что не изменяют содержимого регистра 0.

Упомянем здесь же команды условного перехода Kx<OM, Kx=OM, Kx≥OM, Kx≠OM, где число в регистре М указывает адрес перехода, а также аналогичные им команды безусловного перехода КБПМ и перехода к подпрограмме КППМ. Каждая из них занимает всего один адрес, а не два, как уже знакомые нам команды перехода, однако требует для своей работы один адресуемый регистр.

(Кстати, введем новый термин: всякий раз, когда адрес засылки, вызова или перехода указывается не прямо, а со ссылкой на ячейку памяти, где он хранится, говорят о косвенной адресации.)

Разумеется, на нескольких страницах невозможно раскрыть все секреты нашего калькулятора. Их знание придет с опытом работы по составлению и совершенствованию программ. Сделать программу достаточно короткую, чтобы она вмещалась в память калькулятора, максимально удобную в работе, добиться, чтобы она давала результаты с заданной точностью за возможно более короткое время, — вот цели, преследуемые при этом программистом.

Здесь уместен пример, имеющий прямое отношение к разбирающейся нами задаче. Определяя среднюю площадь кругов, мы подсчитывали площадь каждого круга, умножая квадрат его диаметра на π и деля на 4. Между тем можно определять среднее от квадратов диаметров, а результат единственный раз умножить на π и разделить на 4. Меньше умножений — быстрее получается результат, удобнее и работа с программой, хотя на первый взгляд в ней все прежнее по сравнению с последним ее вариантом — и команды, и длина:

10.ИП0 11.1 12.4 13.П1 14.Сх 15.КИП1 16.Fx² 17.+-
18.FL0 19.15 20.- 21.: 22.Фл 23.Х 24.4 25.:

Как мы увидим на дальнейших занятиях, «Электроника Б3-34» позволяет своему владельцу весьма эффективно решать многие сложные и важные задачи. Можно даже сказать: мало найдется задач, при решении которых не пригодилась бы эта миниатюрная, но умелая ЭВМ.

3. Этапы решения задачи

Микрокалькулятор — машина миниатюрная, и ее возможности ограничены: мала емкость запоминающих устройств, невысока скорость вычислений, отсутствуют внешние устройства для автоматического ввода и вывода информации. Но тем не менее это современная электронная вычислительная машина, и потому решение любой задачи на ней можно подразделить на те же этапы, которые характерны для работы на всякой ЭВМ.

Постановка задачи. Описывается исходная информация об исследуемом объекте, явлении, ситуации, и ставится вопрос, ответ на который требуется получить путем надлежащей переработки этой информации.

Чтобы поначалу не осложнять дело чисто математическими трудностями, возьмем задачу попроще, например из сборника загадок на смекалку: «У отца с матерью сыновей на два больше, чем дочерей, а всего мужчин в семье вдвое больше, чем женщин. Сколько у отца с матерью сыновей и сколько дочерей?»

Математическая формулировка задачи. Взаимосвязи между данными и искомыми величинами выражаются в виде равенств, неравенств, уравнений. Все вместе они образуют математическую модель исследуемого объекта, явления, ситуации.

Обозначим в нашей шутливой задаче число сыновей через y , число дочерей — через z . Разность этих величин по условию равна двум: $y-z=2$.

Выразим далее соотношение между числом мужчин ($y+1$) и числом женщин ($z+1$) в семье: $y+1=2(z+1)$ или иначе: $y-2z=1$.

Теперь уже нетрудно заключить: в семье три сына и одна дочь.

Наша задача, таким образом, свелась к системе двух линейных уравнений. При математической формулировке многих проблем возникают подобные системы, решить которые не так просто, как нашу шутливую задачу. Поэтому

стоит рассмотреть общие подходы к их решению на микрокалькуляторе, заменив конкретные числа буквами:

$$\begin{cases} B_1y + C_1z = D_1, \\ B_2y + C_2z = D_2. \end{cases}$$

Выбор метода решения. На этом этапе с самого начала следует изыскивать возможности для упрощения работы. Например, может оказаться, что искомые величины явным образом выражаются через данные. Так дело обстоит с нашей системой — по методу Крамера каждое из неизвестных выражается отношением определителей второго порядка:

$$z = \frac{\begin{vmatrix} B_1D_1 \\ B_2D_2 \end{vmatrix}}{\begin{vmatrix} B_1C_1 \\ B_2C_2 \end{vmatrix}} = \frac{D_2B_1 - D_1B_2}{B_1C_2 - B_2C_1},$$

$$y = \frac{\begin{vmatrix} D_1C_1 \\ D_2C_2 \end{vmatrix}}{\begin{vmatrix} B_1C_1 \\ B_2C_2 \end{vmatrix}} = \frac{D_1C_2 - D_2C_1}{B_1C_2 - B_2C_1}.$$

Если к тому же оказывается, что расчет несложен и его требуется провести один раз, то разумнее всего выполнить его вручную, предварительно продумав наиболее экономный порядок действий. Так, решая нашу систему из двух линейных уравнений и вычислив z по второй из приведенных формул, y лучше всего вычислять не по первой формуле, а иначе: $y = (D_1 - C_1z)/B_1$. В таком случае расчет потребует меньшего числа операций и результат будет получен быстрее. (Забегая вперед, скажем, что реализующая такой подход программа для «Электроники Б3-34» насчитывает 26 команд. Без предпринятого нами упрощения она потребовала бы более тридцати команд.)

Если явных зависимостей искомых величин от данных вывести не удалось, следует обратиться к математической литературе и там поискать метод аналитического или приближенного решения задачи. В первую очередь стоит заглянуть в книги, посвященные вычислениям на программируемых микрокалькуляторах: А. Н. Цветков и В. А. Епаничников. Прикладные программы для микро-ЭВМ «Электроника Б3-34», «МК-56», «МК-54» (М., Финансы и статистика, 1984); Я. К. Трохименко и Ф. Д. Любич. Инженерные расчеты на микрокалькуляторах (Киев, Техника, 1980); Радиотехнические расчеты на микрокаль-

куляторах (М., Радио и связь, 1983); Инженерные расчеты на программируемых микрокалькуляторах (Киев, Техника, 1985). Если там отыщется подходящая программа, проведите расчет по ней.

Правда, готовая программа, взятая из книг или полученная от коллег, требует известной осторожности: порой, относясь в принципе к интересующему вас классу задач, она может охватывать не все возможные варианты. Поэтому, прежде чем пользоваться ею, надо тщательно выяснить рамки ее применения.

Но, предположим, готовой программы не нашлось, а составить ее необходимо (так бывает, когда задачу приходится решать многократно при меняющихся исходных данных или когда ее решение требует сложной вычислительной работы). В таком случае все дальнейшие этапы решения, начиная с выбора метода, надо пройти самостоятельно.

В математической литературе может найтись даже не один метод, пригодный для решения стоящей перед вами задачи. Из них надо взять наиболее подходящий. Следует выбирать его так, чтобы он обеспечивал решение с заданной точностью при минимальных затратах времени на программирование и счет, допускал создание программы в границах числовой и программной памяти имеющегося у вас микрокалькулятора. Узость этих границ порой вынуждает разбить чрезмерно длинную программу на несколько небольших или вообще отказаться от программирования для микрокалькулятора и обратиться к более мощной ЭВМ.

Проиллюстрируем перипетии выбора на примере нашей задачи, несколько усложнив ее. Предположим, что нам предстоит решить систему не из двух, а из трех линейных уравнений:

$$\begin{cases} a_1x + b_1y + c_1z = d_1, \\ a_2x + b_2y + c_2z = d_2, \\ a_3x + b_3y + c_3z = d_3. \end{cases}$$

Если решать систему по методу Крамера, надо будет вычислить четыре определителя третьего порядка. Каждый выражается алгебраической суммой шести произведений из трех сомножителей. Нетрудно подсчитать, что вычисление каждого такого произведения и его прибавление к общей сумме требуют шести операций (вызов первого сомножителя, вызов второго, умножение, вызов третьего сомножителя, умножение, наконец, сложение или вычитание). Прямое вычисление неизвестных по методу Крамера требует,

таким образом, более 140 операций. Включающая их программа не уместится в памяти «Электроники Б3-34».

Обратимся к методу Гаусса, согласно которому система решается после преобразований, исключающих неизвестные из уравнений и в итоге оставляющих в каждом уравнении по одному неизвестному. Если в нашей системе $a_3 \neq 0$, то из первых двух уравнений можно исключить x :

$$\begin{cases} \left(b_1 - a_1 \frac{b_3}{a_3} \right) y + \left(c_1 - a_1 \frac{c_3}{a_3} \right) z = \left(d_1 - a_1 \frac{d_3}{a_3} \right), \\ \left(b_2 - a_2 \frac{b_3}{a_3} \right) y + \left(c_2 - a_2 \frac{c_3}{a_3} \right) z = \left(d_2 - a_2 \frac{d_3}{a_3} \right). \end{cases}$$

Пара этих уравнений приобрела вид системы, выписанной нами в самом начале занятия, и это позволяет сокращенно обозначить выражения в скобках:

$$\begin{aligned} B_1 &= b_1 - a_1 \frac{b_3}{a_3}; & C_1 &= c_1 - a_1 \frac{c_3}{a_3}; & D_1 &= d_1 - a_1 \frac{d_3}{a_3}, \\ B_2 &= b_2 - a_2 \frac{b_3}{a_3}; & C_2 &= c_2 - a_2 \frac{c_3}{a_3}; & D_2 &= d_2 - a_2 \frac{d_3}{a_3}. \end{aligned}$$

Расчет каждого из шести новых коэффициентов требует семи операций (например, если речь идет о коэффициенте B_1 : вызов b_1 , вызов b_3 , вызов a_3 , деление, вызов a_1 , умножение, вычитание). Итого — 42 операции. Значения y и z можно определить тем же методом, который мы уже описывали (26 операций). Зная y и z , можно определить x из третьего уравнения (11 операций). Таким образом, программа для нахождения всех трех неизвестных будет насчитывать около 80 команд. (Возможно, нам удастся ее сократить, когда дело дойдет до конкретной разработки. Короче говоря, есть надежда вписать ее в программную память нашего калькулятора.)

Числовой памяти тоже должно хватить, ведь в ней надо разместить шесть коэффициентов системы для определения y и z , четыре — для определения x . На это потребуется 10 из 14 адресуемых регистров. Остальные четыре можно будет использовать для хранения промежуточных результатов.

Разумеется, эти прикидки уточняются при составлении программы. Сейчас можно лишь сказать, что возможностей калькулятора для ее составления, по-видимому, достаточно.

Построение алгоритма, то есть последовательности конкретных действий, дающих в итоге ответ на поставленную задачу. Этот этап начинается с анализа расчетных формул.

Их следует максимально упростить, а точнее, придать им такой вид, чтобы вычисления по ним были наиболее просты с точки зрения возможностей имеющейся машины. Желательно выделить в формулах повторяющиеся комбинации переменных и принять эти комбинации за промежуточные переменные. Их целесообразно вычислить однажды и потом использовать в готовом виде. Иногда стоит преобразовать формулы так, чтобы в них появились подобные повторяющиеся комбинации. Затем следует установить порядок выполнения операций, а попутно продумать, как разместить по адресуемым регистрам исходные данные, значения промежуточных переменных и окончательные результаты.

Как правило, для вычислений по одним и тем же формулам можно предложить несколько алгоритмов, притом далеко не равноценных. Соответствующий пример нетрудно усмотреть в стоящей перед нами задаче о решении системы из трех линейных уравнений. Исключая неизвестную x из первых двух уравнений, можно получить вариант системы, уже приведенный выше (назовем его первым вариантом), а можно прийти и к другому варианту, который условимся назвать вторым:

$$(a_3b_1 - a_1b_3)y + (a_3c_1 - a_1c_3)z = (d_1a_3 - a_1d_3), \\ (a_3b_2 - a_2b_3)y + (a_3c_2 - a_2c_3)z = (d_2a_3 - a_2d_3).$$

Нетрудно подметить, что в обоих вариантах константы в левой и правой частях можно вычислить по однотипной схеме, в циклическом порядке, занося в стек сначала набор величин b_1, b_2, b_3 , затем c_1, c_2, c_3 , затем d_1, d_2, d_3 . Предположим, что величины a_1, a_2, a_3 уже размещены перед этим в адресуемых регистрах так: a_1 — в РВ, a_2 — в РС, a_3 — в РД.

Если внести коэффициенты b_1, b_2, b_3 в естественном порядке, с b_1 по b_3 , то в регистрах стека они окажутся на таких местах: b_1 — в РZ, b_2 — в РY, b_3 — в РХ. Если принять второй вариант преобразованной системы, то для вычисления разности $(a_3b_1 - a_1b_3)$ следует, очевидно, начать с таких операций:

Команда	X	Y	Z	T
ИПВ	a_1	b_3	b_2	b_1
X	a_1b_3	b_2	b_1	b_1

А дальше? Надо перегнать величину b_1 в регистр X. Это можно сделать командой FО, выполнив ее дважды. Но тогда величина b_2 окажется в регистре T. Когда она по-

надобится, опять придется прибегнуть к команде FО, причем использовать ее троекратно. Это уже очевидное излишество. Не легче подсчитать и разность $(a_3b_2 - a_2b_3)$.

Если же принять первый вариант, то тут сначала удобно вычислить дробь b_3/a_3 , потом образовать выражение a_2b_3/a_3 и вычесть его из коэффициента b_2 , благо он к этому моменту (проследите за движением чисел по стеку!) окажется рядом, в регистре Y. Получив в регистре X величину $B_2 = b_2 - a_2b_3/a_3$ и переслав ее на нужное место, можно вычислять далее выражение $B_1 = b_1 - a_1b_3/a_3$. Для этого b_1 надо предварительно переместить в регистр X, а это можно сделать, использовав команду FО один-единственный раз. В цикле, где вычисляются величины b_3/a_3 , $B_2 = b_2 - a_2b_3/a_3$ и $B_1 = b_1 - a_1b_3/a_3$, удобно использовать для их отсылки в адресуемые регистры команду вида КПМ ($M=0, 1, 2$ или 3). Тогда при следующих повторениях цикла с исходными данными c_1, c_2, c_3 , затем d_1, d_2, d_3 образующиеся величины $c_3/a_3, C_2 = c_2 - a_2c_3/a_3, C_1 = c_1 - a_1c_3/a_3, d_3/a_3, D_2 = d_2 - a_2d_3/a_3, D_1 = d_1 - a_1d_3/a_3$ встанут в адресуемые регистры одна за другой. Говоря точнее, если номер регистра-счетчика M будет иметь одно из указанных значений, названные выражения расположатся по адресуемым регистрам в порядке убывания их номеров (вспомните, как модифицируется адрес при выполнении команд вида КПМ).

Решение системы из двух линейных уравнений с неизвестными y и z можно отыскать точно так же, как мы это делали в начале занятия:

$$z = \frac{D_2B_1 - D_1B_2}{B_1C_2 - B_2C_1}; \quad y = \frac{D_1 - C_1z}{B_1}.$$

Получив y и z , останется вычислить x , например, по формуле

$$x = \frac{d_3}{a_3} - \frac{c_3}{a_3}z - \frac{b_3}{a_3}y.$$

Здесь опять пригодятся дроби $b_3/a_3, c_3/a_3, d_3/a_3$, вычислившиеся при образовании системы для y и z . На этом закончится решение исходной системы из трех линейных уравнений.

Чтобы завершить это схематическое описание алгоритма, следует сказать о том, что в начальной его части величины a_1, a_2, a_3 , вводимые в стек, соответствующими командами отсылаются на свои места (например, в регистры B, C, D, как мы оговаривали походя). Следует также упомя-

нуть, что некоторые из регистров будут играть роль счетчиков. Таких счетчиков нам, очевидно, понадобится два. Один — для организации цикла, где вычисляются коэффициенты системы с неизвестными y и z . Пусть это будет, скажем, Р0. Поскольку цикл проходится троекратно, туда перед его прохождением надо заслать тройку. Другой регистр-счетчик (пусть это будет Р1) потребуется для расстановки вычисленных коэффициентов системы по адресуемым регистрам. Туда перед прохождением того же цикла следует заслать 11. Тогда первая засылка произойдет в регистр 10, то есть в регистр А, и в процессе дальнейших вычислений адресуемые регистры заполняются так:

0	1	2	3	4	5	6	7	8	9	A	B	C	D
3	11	D_1	D_2	d_3/a_3	C_1	C_2	C_3/a_3	B_1	B_2	B_3/a_3	a_1	a_2	a_3

Как видим, их вполне должно хватить для реализации разработанного нами алгоритма. Некоторые регистры можно даже использовать дважды. Например, судя по формулам для неизвестных x , y и z , в их вычислении уже не участвуют непосредственно коэффициенты a_1 , a_2 , a_3 . В регистры В, С, Д, где хранились эти коэффициенты, зашлем вычисленные значения неизвестных. Если по завершении счета возникнет подозрение, что они неточно считаны с индикатора, их можно будет вызвать командами ИПВ, ИПС, ИПД и списать повторно.

Теперь все готово для того, чтобы приняться за написание программы. Ведь она, говоря по существу, представляет собой перевод разработанного алгоритма на язык команд микрокалькулятора. Алгоритм же изложен достаточно подробно и полно.

Правда, в подобном изложении сложный алгоритм плохо обозрим. Поэтому, прежде чем приниматься за составление программы, его записывают на специальном алгоритмическом языке, а для большей наглядности изображают в графической форме. В этом случае каждая стадия решения описывается словами и формулами, заключенными в замкнутую рамку. Такие фигуры называются блоками. Они соединяются линиями, указывающими последовательность стадий. Вся конструкция именуется блок-схемой алгоритма. Составить ее следует так, чтобы по каждому блоку можно было написать достаточно независимые и обозримые куски программы.

Составление блок-схемы алгоритма. Всякая такая схема состоит из блоков четырех различных очертаний: овалов, параллелограммов, прямоугольников и ромбов. Форма блока соответствует характеру действий, в нем записываемых. В параллелограммах обозначаются ввод и вывод информации, в прямоугольниках — конкретные вычисления (если они выполняются в подпрограмме, то вертикальные стороны прямоугольника чертятся двойными), в ромбах — проверки условий (условные переходы). В овалах пишутся слова «начало» и «конец»; такие фигуры располагаются по концам блок-схемы.

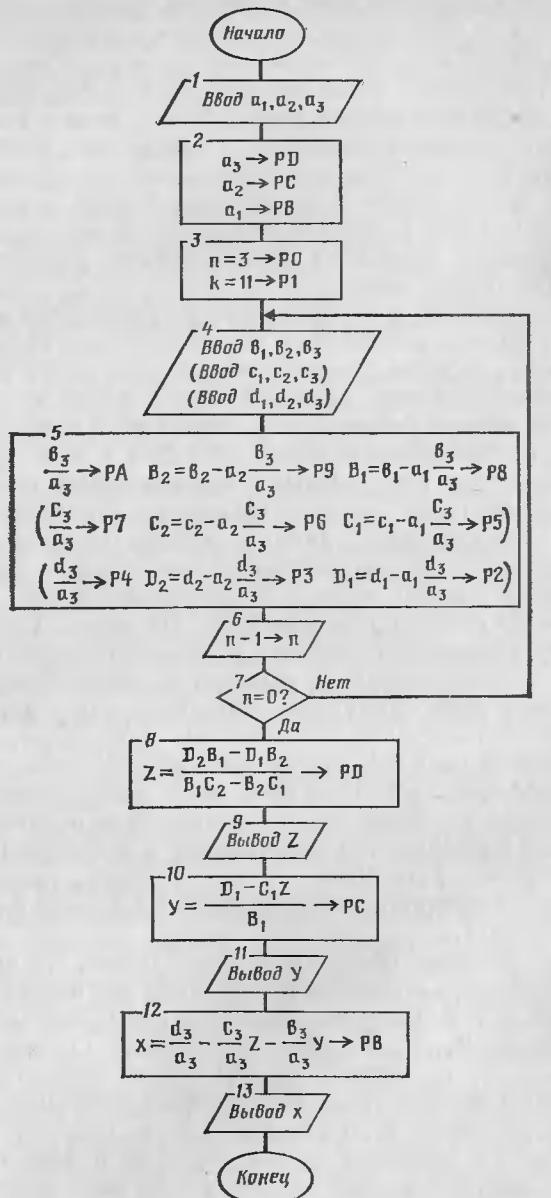
Стоит заметить: если форма блоков закреплена нормами ГОСТа и должна соблюдаться неукоснительно, то содержание блоков, способы описания алгоритмов этими нормами не регламентируются — лишь бы было понятнее.

Блоки обычно размещаются сверху вниз и часто нумеруются. Соединительные линии рисуются в виде прямолинейных отрезков или ломаных с вертикальными и горизонтальными звеньями, иногда снабжаются стрелками. Если же на соединительных линиях стрелки отсутствуют, то предполагается, что по вертикальным прямым движение совершается сверху вниз, а по горизонтальным — слева направо. Из каждого ромба выходят две линии. Одна соответствует выполнению условия и помечается словом «да», другая — невыполнению и помечается словом «нет».

Составим блок-схему разработанного нами алгоритма для решения системы из трех линейных уравнений.

Нарисовав овал с надписью «начало», поставим под ним параллелограмм, обозначающий ввод коэффициентов a_1 , a_2 , a_3 . Ниже поставим прямоугольник, где показано, что введенные коэффициенты рассылаются в регистры В, С, Д соответственно. Еще ниже — прямоугольник, где описана организация счетчиков: засылка числа 3 в регистр 0 и числа 11 в регистр 1.

Затем поставим образующие цикл блоки, где указано, как вычисляются коэффициенты системы для отыскания неизвестных y и z . Сначала параллелограмм, описывающий ввод величин b_1 , b_2 , b_3 (при дальнейших прохождениях цикла — c_1 , c_2 , c_3 , затем d_1 , d_2 , d_3). Ниже прямоугольник, где собраны все формулы, выражющие величины B_1 , B_2 , C_1 , C_2 , D_1 , D_2 через a_1 , a_2 , a_3 , b_1 , b_2 , b_3 , c_1 , c_2 , c_3 , d_1 , d_2 , d_3 . Детализация в подобных случаях вряд ли нужна: каждая формула довольно жестко определяет порядок вычислений по ней, а общая последовательность вычислительных дей-



ствий в блоке задается порядком записи формул. В следующем прямоугольнике отражено уменьшение числа n , содержащимого регистра 0, на единицу, а в замыкающем цикл ромбе — проверка: $n=0$? Невыполнение условия диктует возврат к началу цикла и его повторение, выполнение — выход из цикла.

Следующие прямоугольник и параллелограмм — это вычисление z , его засылка в регистр РД и вывод на индикатор. Следующая пара того же вида — вычисление, засылка в РС и вывод y . Еще одна такая же пара — вычисление, засылка в РВ и вывод x . Замыкает блок-схему овал с надписью «конец».

Составив блок-схему, внимательно проверим ее. Она должна быть разработана так тщательно, чтобы превратить программирование в почти автоматическое расписывание команд программы по отдельным блокам.

Программирование. Начнем его с «начала», хотя этот блок ни в каких командах не воплощается. Не находят отражения в программе и действия, записанные в параллелограммах: ввод информации производится вручную с клавиатуры, вывод заключается в считывании чисел с индикатора. Короче говоря, первые команды нашей программы отразят содержимое первого из прямоугольников блок-схемы, по командам которого введенные в калькулятор коэффициенты a_1, a_2, a_3 расставляются по своим местам в числовой памяти. Вводить их проще всего все вместе, перемежая их набор на клавиатуре нажатием клавиши \uparrow , что в нашей записи будет выглядеть так: $a_1 \uparrow a_2 \uparrow a_3$.

Читателю, ознакомившемуся на предыдущем занятии с системой команд «Электроники Б3-34», по-видимому, не составит труда написать этот начальный фрагмент. Завершим его командой С/П, чтобы была возможность ввести следующую тройку: $b_1 \uparrow b_2 \uparrow b_3$.

00.ПД 01.Ф 02.ПС 03.Ф 04.ПВ 05.1 06.1 07.П1 08.3 09.ПО 10.С/П.

Переводить на язык команд следующий блок, пожалуй, будет уже посложнее. Здесь наверняка придется в полной мере использовать стек. Поэтому, составляя программу, будем указывать перемещения чисел по его регистрам с помощью уже привычной нам таблицы. Заполнять ее станем карандашом: если допустим ошибку, то сотрем неверные записи и впишем на их место исправленные.

Предположим, фрагмент получился таким, каким он записан в первых трех столбцах таблицы. В следующих столб-

Адрес	Команда	Код	Регистры						Алгоритм	Блок
			X	Y	Z	T	X1	Оперативные		
10	С/П	50	B ₃	B ₂	B ₁			PB: a ₁	первое прохождение цикла	
11	ИПД	6Г	a ₃	B ₃	B ₂	B ₁		PC: a ₂		
12	:	13	B ₃ /a ₃	B ₂	B ₁	B ₁	a ₃	PD: a ₃		
13	КП1	L1	—	—	—	—	—	P1: 10		
14	ИПС	60	a ₂	B ₃ /a ₃	B ₂	—	—	PA: B ₃ / a ₃		
15	X	12	a ₂ B ₃ / a ₃	B ₂	B ₁	—	a ₂			
16	—	11	B ₂ -a ₂ B ₃ / a ₃	B ₁	—	—	a ₂ B ₃ / a ₃	P1: 9		
17	КП1	L1	—	—	—	—	—	P9: B ₂ -a ₂ B ₃ / a ₃		
18	F	25	B ₁	—	—	B ₂	B ₂ -a ₂ B ₃ / a ₃		третье прохождение цикла	
19	ИПА	6-	B ₃ /a ₃	B ₁	—	B ₁	—	d ₃ / a ₃	P4	
20	ИПВ	6L	a ₁	B ₃ / a ₃	B ₁	—	—		D ₂ =d ₂ -a ₂ d ₃ / a ₃	P3
21	X	12	a ₁ B ₃ / a ₃	B ₁	—	—	a ₁		D ₁ =d ₁ -a ₁ d ₃ / a ₃	P2
22	—	11	B ₁ -a ₁ B ₃ / a ₃	—	—	—	a ₁ B ₃ / a ₃	P1: 8 P8: B ₁ -a ₁ B ₃ / a ₃		
23	КП1	L1	—	—	—	—	—	PO: 2	n-1 → n	6
24	FLO	5Г	—	—	—	—	—		если n ≠ 0, то к adr. 10 если n = 0, то к adr. 26	7
25	10	10	—	—	—	—	—			

цах показано, как будут заполняться по ходу выполнения цикла регистры стека, если вначале в него были занесены величины b_1 , b_2 , b_3 .

Как видим, нынешняя таблица шире той, что составлялась нами на прежнем занятии. В последних двух колонках отмечен номер блока, которому соответствует данная цепочка команд, и приведен фрагмент алгоритма, переводом которого на командный язык она служит. Добавлена также колонка, где указано содержимое адресуемых регистров, упоминаемых в этом отрывке текста программы.

Последив цикл от начала до конца, вернемся к его началу и мысленно введем следующую тройку чисел: c_1 , c_2 , c_3 . Сотрем содержимое колонок, соответствующих регистрам стека, и будем заполнять их заново, прослеживая новое прохождение цикла.

Добравшись до адреса 19, выполняем записанную там команду ИПА. Она вызовет в регистр X дробь b_3/a_3 . Нам же нужна сейчас совсем другая дробь: c_3/a_3 . Ошибка? Не-

омненно. Как же исправить ее? Очевидно, дроби b_3/a_3 , $/a_3$, d_3/a_3 после их вычисления надо засыпать не только регистры А, 7, 4, но и запасать еще где-то, откуда извлекать при необходимости. Но где? Все адресуемые регистры уже отведены под величины, участвующие или определяемые в процессе вычислений. Однако заполняются они не разу. И тот из них, который заполняется последним (точка Р2), можно привлечь для временного хранения нужных нам дробей.

Исправляем ошибочный фрагмент программы с учетом
этого замечания: 10.С/П 11.ИПД 12.: 13.КП1 14.П2 15.ИПС
6.Х 17.— 18.КП1 19.F○ 20.ИП2 21.ИПВ 22.Х 23.—
4.КП1 25.FL0 26.10.

Составьте заново таблицу и убедитесь, что теперь вычисление коэффициентов $B_1, B_2, C_1, C_2, D_1, D_2$ протекает верно.

(Здесь следует заметить, что решению задач на ЭВМ вообще присущ подобный «циклический» характер, когда ошибка, обнаруженная на некотором этапе работы, возвращает программиста назад, к одному из уже пройденных этапов. Так, например, не найдя хорошего метода решения, приходится заново формулировать задачу; не сумев разработать алгоритм, выполнимый на имеющейся машине,— сковать новый метод решения. При составлении блок-схемы, затем и при программировании нередко вскрываются ошибки, допущенные при разработке алгоритма... И даже тогда, когда программа составлена окончательно и предстает безупречной, во время счета в ней может вскрыться не замеченная ранее ошибка, исправление которой ставит начать работу вновь с того или иного из предшествующих этапов вплоть до того, где задача еще только дала математическая формулировка. Программист, особенно начинающий, должен быть готов к подобным неприятностям.)

Попробуйте самостоятельно составить завершающий фрагмент программы, где вычисляются z , y и x . Не забудьте ставить команду останова там, где получается каждая неизвестных; это нужно, чтобы считывать их значения индикатора. Сравните полный текст программы с приведенным на стр. 53.

Отладка программы. Так называется этап, начинающийся с ввода программы в калькулятор и заключающийся проверке, действительно ли она выполняет алгоритм, который должна выполнять.

Итак, включаем калькулятор, устанавливаем его в режим программирования и вводим в него нашу программу. Как мы уже знаем, она хранится в калькуляторе в виде последовательности кодов операций, записанных в ячейки программной памяти. Если при вводе допущена ошибка, эта последовательность не совпадает с той, что находится в третьей колонке только что составленного нами текста программы.

Чтобы выяснить, так это или не так, возвращаемся в режим вычислений, передаем управление на начальный адрес программы с помощью клавиши В/0 и вновь переводим калькулятор в режим программирования. Затем раз за разом нажимаем клавишу «ШГ вправо». С каждым ее нажатием содержимое счетчика адресов, отображаемое в правой стороне индикатора, увеличивается на единицу, так что у левого края индикатора последовательно появляются коды, записанные в программную память. Адрес крайнего левого кода всегда на единицу меньше того, что в данный момент виден у правого края.

Так можно просмотреть всю программную память. Если после очередного нажима клавиши «ШГ вправо» слева на индикаторе появился не тот код, что записан в тексте программы, следует вернуться на адрес назад, нажав клавишу «ШГ влево», а потом заново набрать команду, которая должна стоять по адресу, высвечиваемому справа. Прежнее содержимое этого адреса сотрется и заменится кодом набранной операции.

Команды перехода, каждая из которых занимает два адреса, в ходе подобного исправления должны вводиться заново целиком. Пусть, например, просматривая составленную нами программу и добравшись до адреса 26, мы обнаружили, что в команде организации цикла неверно указан адрес перехода: 25.FL0 26.20. Чтобы исправить ошибку, надо дважды нажать клавишу «ШГ влево», а затем заново ввести и операцию FL0, и верный адрес перехода 10. Если бы мы попытались поступить проще, ввести вновь одно лишь это число 10, калькулятор «не понял» бы, что речь идет об адресе перехода, и записал бы в две последовательные ячейки программной памяти 1 и 0, а это совсем не то, что требуется. Так, исправляя ошибку за ошибкой, можно добиться полного совпадения текстов, записанных в память калькулятора и на бумаге.

Окончательно убедившись, что программа правильно введена в калькулятор, проверим, правильно ли она ра-

ботает, и исправим ошибки, если они были допущены при ее составлении.

Если программа невелика, проще всего это сделать так. В режиме ручных вычислений передаем управление на начальный адрес, а затем раз за разом нажимаем клавишу ПП. («Потактовый Проход» — так раскрывается это обозначение, если вычисления выполняются вручную.) С каждым ее нажатием последовательно выполняется по одной команде программы и результат выводится на индикатор. Особенно легко протекает такая проверка, если составить контрольный пример, для которого можно сказать, какие результаты должны получиться после выполнения каждой команды.

Обратимся к нашей программе для решения системы трех линейных уравнений. Пусть система такова:

$$\begin{cases} x + 2y + 3z = 10, \\ x + 3y + 4z = 13, \\ 2x + y + 2z = 10. \end{cases}$$

Циклический фрагмент программы, стоящий по адресам с 10 по 26, должен преобразовать первые два уравнения системы к виду

$$\begin{cases} 1,5y + 2z = 5, \\ 2,5y + 3z = 8. \end{cases}$$

Таким образом, $B_1=1,5$; $B_2=2,5$; $C_1=2$; $C_2=3$; $D_1=5$; $D_2=8$.

Предположим, что этот фрагмент содержит ошибку, допущенную нами при первой попытке составить его: 10.C/P 11.ИПД 12.: 13.КП1 14.ИПС 15.× 16.— 17.КП1 18.ФО 19.ИПА 20.ИПВ 21.× 22.— 23.КП1 24.ФЛО 25.10.

Передаем управление на нулевой адрес с помощью клавиши В/0 и вводим в калькулятор числа $a_1=1$, $a_2=1$, $a_3=2$. С помощью клавиши ПП проходим первые девять команд и останавливаемся по адресу 10. Вводим $b_1=2$, $b_2=3$, $b_3=1$ и, нажимая клавишу ПП, выполняем интересующий нас цикл в режиме потактового прохода. Дойдя до команды 17.КП1, мы видим, что она отсылает в Р9 верную величину $B_2=2,5$. Продолжая проход, отмечаем, что команда 23.КП1 направляет в Р8 опять-таки верную величину $B_1=1,5$. Наконец, по команде 24.ФЛО 25.10 программа вернется на адрес 10 и остановится на нем (мы заметим при этом, что всякая команда перехода выполняется за одно нажатие клавиши ПП). Введем $c_1=3$, $c_2=4$, $c_3=2$ и начнем второй

потактовый проход цикла. И тут мы обнаружим, что команда 23.КП1 пошлет в Р6 неверно вычисленный коэффициент $C_1=2,5$ (должно быть 2).

Как исправить ошибку, мы уже знаем: между командами КП1 и ИЛС, стоящими по адресам 13 и 14, надо вставить команду П2. Однако вакантного адреса для нее нет. Исправленную программу, начиная с 14-го адреса, придется вводить заново.

Бывает, надо сказать, и так, что верный фрагмент оказывается короче неверного. Тогда при замене старого фрагмента исправленным можно не трогать остальной текст программы, а в освободившиеся адреса записать «пустую» операцию. Она заносится нажатием двух клавиш К и НОП («Нет Операции») и занимает один адрес.

Начинающим программистам рекомендуется при составлении первого варианта программы записывать эту команду в сомнительные фрагменты, обеспечивая резервы программной памяти на тот случай, если исправление ошибочного фрагмента обернется его удлинением, как это только что произошло у нас.

Описанный способ потактового прохода очень удобен при отладке простых программ. Для более сложных он несколько трудоемок. В этом случае прибегают к побочному проходу. После конечного адреса каждого блока или в другие ключевые места программы вставляют при ее составлении команду С/П. После запуска программа останавливается в этих местах, и в моменты останова на индикаторе появляются промежуточные результаты. Анализируя их, определяют ошибочный блок или фрагмент и проверяют его в режиме потактового прохода. Для этого программу вновь запускают с самого начала и, остановившись перед подозрительным фрагментом, проходят его с помощью клавиши ПП команда за командой. При появлении явно неверного результата можно, не нажимая в очередной раз клавишу ПП и перейдя в режим программирования (F ПРГ), увидеть слева код последней выполненной операции. В лучшем случае причина ошибки — неверно введенная команда (тогда надо лишь занести ее заново, предварительно нажав клавишу «ШГ влево»). В худшем — неправильно составленная программа (в таком случае ее порой приходится переделывать с самого начала).

Иногда по самому характеру неверного промежуточного результата удается определить ошибочные команды. Если их адреса близки к адресу, высвечиваемому справа на ин-

дикаторе после нажатия клавиши F ПРГ, до них можно добраться с помощью клавиш «ШГ вправо» или «ШГ влево». При большой разнице в адресах ошибочной и текущей команд надо перейти в режим вычислений (F АВТ) и далее передать управление на нужный адрес.

Случается, что при первом пуске программы или при побочном ее проходе калькулятор останавливается и на индикаторе появляется сообщение ЕГГОГ (ошибка) — сигнал так называемого аварийного останова. Он может означать, что неверно составленная программа поставила калькулятор перед необходимостью выполнить невыполнимую операцию (разделить число на нуль, извлечь квадратный корень из отрицательного числа, вычислить арксинус от числа, большего единицы, взять логарифм от x или выполнить команду x^y при неположительном x , вычислить тангенс от числа вида $\pi/2 \pm \text{пл}$). Если сразу после этого перейти в режим программирования, то вторым слева на индикаторе будет код операции, послужившей причиной аварийного останова. Адрес, по которому он записан, на два меньше того, что высвечивается на индикаторе справа. Если теперь вернуться в режим вычислений, на индикаторе появится и число — виновник останова. После этого надо выявить причину создавшейся ситуации.

Сообщение ЕГГОГ появляется и в том случае, если содержимое регистра Х превышает по абсолютной величине $9,9999999 \cdot 10^{99}$ и потому не может быть выведено на индикатор. Надо заметить, что останов в подобных случаях происходит не всегда. Это зависит от того, какая последовательность команд привела к появлению чрезмерно большого числа.

Когда ошибка в программе устранена, можно продолжить решение контрольного примера, предварительно исправив содержимое регистров, искаженное вследствие ошибки. Но если сделать это трудно, отладку надо начать сначала.

Но вот отладка окончена. Чтобы окончательно убедиться в том, что программа не содержит ошибок, решение контрольного примера проводят вновь с самого начала, попутно отмечая, сколько времени требуется на получение каждого из искомых результатов (эти данные понадобятся при оформлении программы).

Как видим, при составлении качественной программы порой приходится тратить много времени на постановку задачи, выбор метода, разработку алгоритма, составление

программы и ее отладку. Однако все эти усилия, как правило, окупаются при использовании программы. Использовать же ее можно лишь тогда, когда она отлажена окончательно, то есть верность выдаваемых ею результатов гарантируется.

Если пользоваться ею в будущем предполагается неоднократно, ее следует тщательно оформить.

Оформление программы должно быть очень аккуратным. Ведь достаточно одной неразборчиво написанной команды или кода — и использовать программу будет невозможно.

Для ее оформления лучше всего взять лист ватмана или плотной бумаги. В заголовке помещают название программы — несколько слов, кратко и точно поясняющих ее суть, потом — достаточно подробное описание того, что делает программа, со всеми необходимыми формулами, с указанием области допустимых значений исходных данных.

В нашем примере условия применения составленной программы, как нетрудно усмотреть из анализа используемых формул, таковы:

$$a_3 \neq 0; \quad \begin{vmatrix} b_1 a_1 \\ b_3 a_3 \end{vmatrix} \neq 0; \quad \begin{vmatrix} b_1 c_1 \\ b_2 c_2 \end{vmatrix} \neq 0.$$

Желательно указать, как используются по ходу счета адресуемые регистры, сколько времени уходит на получение каждого результата, сообщить о существенных, но не очевидных особенностях программы. Без подобных комментариев даже ее составителю бывает трудно работать с ней некоторое время спустя, не говоря уже о тех, кто захочет воспользоваться этой программой.

Затем следует собственно текст программы: адреса, команды, коды по столбцам.

Сразу после текста пишется инструкция по использованию данной программой. По пунктам, коротко и понятно описывается работа человека от ввода текста программы, начальных данных и констант до получения результата. Можно оформить инструкцию в виде таблицы (см. стр. 53); символы вводимых величин в графе «нажимаемые клавиши» означают, что указанные числа вводятся с клавиатуры.

Далее обязательно приводится контрольный пример с конкретными значениями начальных данных и получающихся результатов. Как бы тщательно ни вводилась программа, лишь точное решение контрольного примера дает уверенность, что она введена без ошибок.

Решение системы трех линейных уравнений

$$\begin{cases} a_1x + b_1y + c_1z = d_1 \\ a_2x + b_2y + c_2z = d_2 \\ a_3x + b_3y + c_3z = d_3 \end{cases}$$

Алгоритм: из первых двух уравнений исключается x

$$\begin{cases} B_1 & C_1 & D_1 \\ \left(B_1 - a_1 \frac{B_3}{a_3} \right) y + \left(C_1 - a_1 \frac{C_3}{a_3} \right) z = \left(D_1 - a_1 \frac{D_3}{a_3} \right) \\ B_2 & C_2 & D_2 \\ \left(B_2 - a_2 \frac{B_3}{a_3} \right) y + \left(C_2 - a_2 \frac{C_3}{a_3} \right) z = \left(D_2 - a_2 \frac{D_3}{a_3} \right) \\ \end{cases}$$

$$z = \frac{D_2 B_1 - D_1 B_2}{B_1 C_2 - B_2 C_1}; \quad y = \frac{D_1 - C_1 z}{B_1}; \quad x = \frac{d_3}{a_3} - \frac{c_3}{a_3} z - \frac{b_3}{a_3} y$$

Регистры: 01 — счетчики; 2-9.П — оперативные; В-x; С-y; D-z

Коэффициенты и свободные члены системы уравнений вводятся по столбцам.
Время занесения каждого столбца 3-5с, время вычисления каждого неизвестного 4с.

Программа

адрес	команда	код									
0 0	ПЛ	4 Г	1 2	3 2	X	1 2	4 8	—	1 1		
0 1	FO	2 5	1 7	—		1 1	4 9	ИП8	6 8		
0 2	ПС	4 С	1 8	КП1	L1	3 4	ИП8	6 8	5 0	:	1 3
0 3	FO	2 5	1 9	FO	2 5	3 5	ИП8	6 6	5 1	ПС	4 С
0 4	ПВ	4 L	2 0	ИП2	6 2	3 6	X	1 2	5 2	С/П	5 0
0 5	1	0 1	2 1	ИП8	6 L	3 7	ИП9	6 9	5 3	ИП4	6 4
0 6	1	0 1	2 2	X	1 2	3 8	ИП5	6 5	5 4	ИП7	6 7
0 7	П1	4 1	2 3	—	1 1	3 9	X	1 2	5 5	ИП0	6 Г
0 8	3	0 3	2 4	КП1	L1	4 0	—	1 1	5 6	X	1 2
0 9	П0	4 0	2 5	FL0	5 Г	4 1	:	1 3	5 7	—	1 1
1 0	С/П	5 0	2 6	10	1 0	4 2	ПД	4 Г	5 8	ИП8	6 —
1 1	ИП0	6 Г	2 7	ИП8	6 8	4 3	С/П	5 0	5 9	ИПС	6 С
1 2	:	1 3	2 8	ИП2	6 3	4 4	ИП2	6 2	6 0	X	1 2
1 3	КП1	L1	2 9	X	1 2	4 5	ИП5	6 5	6 1	—	1 1
1 4	П2	4 2	3 0	ИП9	6 9	4 6	ИП0	6 Г	6 2	П8	4 Л
1 5	ИП8	6 С	3 1	ИП2	6 2	4 7	X	1 2	6 3	С/П	5 0

Инструкция

Пункт	Действие	Нажимаемые клавиши	На индикаторе	Пункт	Действие	Нажимаемые клавиши	На индикаторе
1	Переход в режим программирования	F ПРГ	8	Вычисление у	С/П	у	
2	Ввод программы	по тексту программы	9	Вычисление x	С/П	x	
3	Переход в режим вычислений	F АВТ	10	Повторное считывание z	ИП0	z	
4	Ввод первого столбца коэффициентов	B/0a ₁ f ₁ a ₃ С/П	11	Повторное считывание у	ИПС	у	
5	Ввод второго столбца коэффициентов	B ₁ f ₂ B ₃ С/П	12	Повторное считывание x	ИПВ	x	
6	Ввод третьего столбца коэффициентов	c ₁ f ₂ c ₃ С/П	13	Решение новой системы: к п. 4			
7	Ввод четвертого столбца коэффициентов, вычисление z	d ₁ f ₂ d ₃ С/П	z				

Центральный пример:

$$\begin{cases} x + 2y + 3z = 10 \\ x + 3y + 4z = 13 \\ 2x + y + 2z = 10 \end{cases}$$

$$x = 3, y = 2, z = 1$$

Для начинающего программиста нелишне прикладывать к оформленной таким образом программе блок-схему алгоритма и таблицы, показывающие движение чисел по стеку для тех участков программы, разработка которых вызвала особые затруднения. Подробный комментарий к таблице, записанный в графе «алгоритм», поможет избежать трудностей в дальнейшем, при работе с программой.

Различные программы лучше писать на листах одинакового формата. Со временем, когда их скопится много, их нужно будет расклассифицировать по тематике, по характеру решаемых задач и хранить в добротной коробке.

Лист с только что составленной нами программой для решения системы трех линейных уравнений может выглядеть так, как показано на стр. 53 (ср.: А. Н. Цветков и В. А. Епаничников. Прикладные программы для микро-ЭВМ «Электроника Б3-34», «МК-56», «МК-54». М., Финансы и статистика, 1984, с. 14—15).

Разумеется, оформлять программы можно и по-иному — так, как удобнее их предполагаемым и возможным пользователям.

Счет по программе требует тщательной подготовки. Ведь при более или менее серьезных вычислениях приходится иметь дело с большим количеством чисел (исходные данные, промежуточные и окончательные результаты), многократно нажимать клавиши, считывать числа с индикатора и записывать на бумагу... Далеко ли тут до ошибки?

Поэтому перед началом вычислений нужно должным образом подготовить рабочее место: по возможности исключить внешние помехи, приготовить перечень исходных данных и бумагу для записи результатов, авторучку, карандаши.

4. Ввод и вывод информации

Известно несколько основных способов ввода чисел в калькулятор. Выбор подходящего определяется при составлении конкретной программы ее особенностями, а те — ее назначением. Различные способы можно комбинировать в одной и той же программе. Однако не следует забывать, что считающему на калькуляторе удобнее единообразие. Несчитающему на калькуляторе удобнее чередование клавиш, нажимаемых при вводе, оправданное чередование клавиш, нажимаемых при вводе, может значительно затруднить работу, послужить причиной ошибок.

Идеалом начинающего программиста должна стать простота. А наиболее просто работать с калькулятором, когда все управление им сводится к нажатию в нужные моменты на одну-единственную клавишу.

Чтобы запустить программу на счет, то, какова бы она ни была, надо нажать клавишу С/П. Вот ею-то в лучшем случае и следует ограничиться, управляя калькулятором.

...Калькулятор включен, управление передано на начальный адрес программы, на клавиатуре набрано первое из вводимых чисел, нажата клавиша С/П, запущенная программа переслала введенное число в какой-либо из адресуемых регистров или вычислила от него какое-то выражение, выполнила последующие действия, для которых хватает введенного числа, и остановилась по команде останова, представленной в программе. Пока калькулятор бездействует, на клавиатуре набрано следующее из вводимых чисел, снова нажата клавиша С/П, введенное число, в свою очередь, подвергнуто обработке, и вновь программа остановилась, готовая к приему и обработке другого, очередного числа...

Работать в таком порядке очень легко, а это немаловажно, если за калькулятором новичок. Однако описанный порядок осуществим не всегда. Нередко приходится вводить исходные данные более или менее обширными группами. Способам группового ввода в основном и посвящено это занятие.

Руководство по эксплуатации, прилагаемое к «Электронике Б3-34», рекомендует вводить числа с помощью клавиши П, вызывать на индикатор — с помощью клавиши ИП. При вводе нужное число набирается на клавиатуре и отображается на индикаторе, а одновременно заносится в регистр Х. Затем нажимается клавиша П и вслед за нею — клавиша с номером нужного адресуемого регистра. В нем тотчас копируется содержимое регистра Х. При выводе результатов сначала нажимают клавишу ИП, затем клавишу нужного адресуемого регистра, после чего его содержимое копируется в регистре Х и высвечивается на индикаторе.

Пусть нам требуется, например, заслать в регистр 5 число 3, а из регистра 6 вызвать хранящееся там число 4. Выразим это, прибегая к таким обозначениям:

3 (П5) (ИП6) 4

Здесь над чертой пишутся числа, набираемые на цифровых клавищах и выводимые на индикатор, или их

символы, а в кружках — команды, выполняемые для засылки и вызова.

Рассмотренный способ ввода и вывода информации будем для краткости именовать ПР-вводом и ИПР-выводом. С точки зрения человека, работающего с микрокалькулятором (назовем его оператором), это самый неудобный и медленный способ. Напрягая внимание, чтобы не ошибиться при нажатии нужных клавиш, оператор быстро утомляется, скорость его работы падает, он начинает допускать ошибки.

Вспомним, как действует клавиша потактового прохождения программы — ПП. Одно ее нажатие приводит к тому, что калькулятор выполняет лишь одну, очередную команду программы. Между двумя нажатиями клавиши ПП можно ввести новое число в регистр X.

Чтобы лучше понять, как протекает работа оператора в этом случае, рассмотрим ради примера небольшую программу: 00.ПС 01.ПВ 02.ПА 03.— 04.Fx² 05.→ 06.: 07 С/П 08.ИПС 09.ИПВ 10.ИПА.

Набрав программу, и вернувшись в режим вычислений (F АВТ), выполните затем такие действия:

B/0 16 (ПП) 12 (ПП) 20 (ПП)

По команде В/0 счетчик команд настраивается на адрес 00. Затем на клавиатуре набирается число 16. Нажата клавиша ПП. Калькулятор выполняет команду, записанную по адресу 00, то есть команду ПС, пересылая число 16 из регистра X в регистр С, и останавливается. Пока он стоит, сменим содержимое регистра X, набрав число 12. Вновь нажмем клавишу ПП. Число 12 окажется в регистре В в соответствии с командой по адресу 01. Далее аналогичным образом наберем и поместим в регистр А число 20.

Группа команд, записанная в нашем примере по адресам 00—02, служит для ввода исходной информации и называется программным блоком ввода данных или проще — блоком ввода.

Обратим внимание: после того как введенное в регистр X число послано в какой-то адресуемый регистр, оно передвигается в регистр Y при вводе следующего числа. Так по ходу записи чисел в адресуемые регистры происходит и их размещение по регистрам стека. В нашем примере в регистре Z оказывается в итоге число 16, а в регистре Y — число 12, в регистре X — число 20. Это обстоятельство

удобно использовать для организации последующих вычислений. Так и в нашем примере. После того как ввод закончен и калькулятор стоит перед выполнением команды по адресу 03, его можно запустить на счет клавишей С/П. Выполнив команды по адресам 03—06, калькулятор остановится по адресу 07, где записана команда останова. Выполненные команды, как нетрудно проследить, вычислят дробь $(20-12)^2/16=4$. Четверка и высветится на индикаторе после останова.

Команды, записанные в нашей программе по адресам 08—10, образуют блок вывода. Они обеспечивают последовательный вызов на индикатор чисел, записанных в регистрах С, В, А. Для этого надо лишь три раза подряд нажать все ту же клавишу ПП.

Этот способ будем в дальнейшем называть ПП-вводом и соответственно ПП-выводом. Для оператора он удобен тем, что не нужно думать, в какие адресуемые регистры засыпать и из каких регистров вызывать нужные числа. Просто нажимай клавишу ПП, и очередное число обрабатывается должным образом. Последовательность ввода чисел устанавливается инструкцией к программе.

Однако блоки ввода и вывода требуют места в программной памяти. Можно ли устраниТЬ этот недостаток? И в каких случаях?

Иногда вводимые числа не обязательно записывать в адресуемые регистры. Нужно лишь вычислить какое-то зависящее от них выражение, которое будет использоваться в дальнейших вычислениях по программе. Здесь опять можно обратиться к только что рассмотренной нами цепочке команд: из введенных в калькулятор чисел $a=20$, $b=12$, $c=16$ они сформировали величину $(a-b)^2/c$.

Если в подобном выражении участвует не более четырех чисел, их можно ввести в регистры стека с помощью клавиши ↑, а затем клавишей С/П запустить программу, в самом начале которой должны, разумеется, стоять команды, вычисляющие нужное выражение.

Рассмотренная нами программа при таком способе ввода (назовем его СТ-вводом) выглядела бы короче: 00.— 01.Fx² 02.→ 03.: 04.С/П.

Набираем на клавиатуре $16 \uparrow 12 \uparrow 20 \text{B/0}$, нажимаем клавишу С/П — и через несколько секунд на индикаторе появится знакомое число 4.

Применяя СТ-ввод, программирование задачи лучше начинать с выяснения, в какой последовательности вводить

исходные данные. В нашем примере она была такой: c, b, a . Но, допустим, с точки зрения оператора удобнее вводить числа в алфавитном порядке их символов: a, b, c . Если вы хотите, чтобы с вашими программами было легко работать, то возьмите за правило отдавать предпочтение в выборе последовательности вводимых чисел оператору. Раз ему удобнее последовательность a, b, c — так тому и быть. Набираем на клавиатуре $20 \uparrow 12 \uparrow 16$, и числа помещаются в стеке в следующем порядке: $a=20$ в РZ, $b=12$ в РY, $c=16$ в РX. В таком случае нашу программу нужно дополнить командами, переставляющими числа по регистрам стека так, как это требуется для подсчета дроби $(a-b)^2/c$. Например: $00. \uparrow 01.FO \quad 02.FO \quad 03.- \quad 04.Fx^2 \quad 05.\leftarrow \quad 06.: \quad 07.C/P$.

Наряду с СТ-вводом можно говорить и про СТ-вывод результатов. Назовем так способ, при котором числа из различных регистров стека перемещаются в регистр X и отображаются на индикаторе. Для такого вывода часто используют команду \rightarrow . С ее помощью на индикаторе появляется содержимое регистра Y. А командой FO можно последовательно вывести на индикатор содержимое всех регистров стека.

При СТ-выводе программа составляется так, чтобы по окончании ее работы результаты располагались по стековым регистрам нужным образом.

В ряде случаев, когда вводится много чисел и потому СТ-ввод применять нельзя, а для ГПП-ввода не хватает программной памяти, можно использовать команды косвенной засылки вида КГМ. Имея дело с ними, будем говорить о КП-вводе. В качестве регистра M выбирается один из тех, чье содержимое при косвенной засылке модифицируется, то есть регистры 0, 1, ..., 6. Выбор подходящего регистра зависит от особенностей решаемой задачи. В подобных случаях находит удачное применение команда КП↑, не описанная инструкцией к «Электронике Б3-34». Вариант, где используется она и операция FL0, мы сейчас и рассмотрим.

Команда КП↑ «работает» почти так же, как команда КПО: засыпает содержимое регистра X в адресуемый регистр, номер которого есть содержимое регистра 0 (положим для простоты, что в регистре 0 находится целое неотрицательное число от 0 до 13 включительно). От команды КПО команда КП↑ отличается тем, что перед своим выполнением не уменьшает на единицу содержимое Р0 — как говорят, не модифицирует адреса. Аналогичным образом коман-

да КП↑ производит вызов так же, как команда КИПО, но без модификации адреса.



Программу, поясняющую КП-ввод, для большей наглядности изобразим рисунком, где команды располагаются в столбик и без указания адресов (между ними могут располагаться команды промежуточной обработки вводимых данных).

Такой блок позволяет вводить числа в адресуемые регистры в порядке убывания адресов — с некоторого N -ного по 1-й. Число N может равняться даже 13 (калькулятор «поймет» его как обозначение регистра Д).

Разумеется, если число ($N-1$) двузначное, то его знаки записываются в программу по двум последовательным адресам. Программа «наберет» это число точно так же, как мы набирали бы его на цифровых клавишиах вручную.

Положим, что перед началом ввода по этим командам калькулятор стоит, готовый к выполнению первой из них, ПН. Набираем на клавиатуре первое из вводимых чисел x_1 и нажимаем клавишу С/П. Командой ПН оно отсылается в N -й адресуемый регистр. Следующими двумя командами в регистр 0 направляется число ($N-1$). Калькулятор останавливается вновь. Набираем на клавиатуре второе из вводимых чисел x_2 и запускаем машину клавишей С/П. Командой КП↑ число x_2 отсылается в ($N-1$)-й адресуемый регистр — тот самый, номер которого указан содержимым нулевого регистра. Операция FL0 уменьшает это содержимое до ($N-2$).

После остановки наберем на клавиатуре число x_3 и команда КП↑ направит его в регистр с номером $N-2$. При дальнейших повторениях цикла, охваченного на схеме стрелкой, вводимые далее числа будут размещаться в регистрах с убывающими номерами — до первого.

Сколько бы сложным ни казалось описание работы программы, действия оператора будут при этом предельно простыми: набрано x_1 , нажата клавиша С/П, набрано x_2 , вновь нажата клавиша С/П..., и так далее.

Команды, заключенные на рисунке в скобки, могут и отсутствовать. Поставить их нужно, если желательно ввести какое-то число еще и в регистр 0. Набрать это число следует, пока калькулятор стоит. Затем надо нажать клавишу С/П — и калькулятор приступит к выполнению команд, находящихся в программе далее: набранное число зашлется в Р0.

В нашем разговоре давно назревала тема, связанная с исправлением ошибок ввода. Если оператор заметил ошибку по ходу набора, нужно нажать клавишу Сх и набрать число вновь. Сложнее исправить ошибку тогда, когда число уже заслано в память. Тут многое зависит от способа, с помощью которого вводятся исходные данные.

Проще всего исправлять ошибки при ПР-вводе: набранное заново число нужно отослать на свое место повторно. Чуть сложнее исправить ошибку при ПП-вводе: здесь следует вернуть программу на один шаг назад и лишь затем повторить ввод:

шг x_i ПП

Однако если при ПП-вводе числа должны не только рассыпаться по адресуемым регистрам, но и определенным образом размещаться по регистрам стека, то при исправлении ошибки надо восстановить и предыдущее расположение чисел в стеке:

FO шг x_i ПП

При СТ-вводе отдельное неверно введенное число вводится заново так:

FO x_i ↑

При КП-вводе исправление ошибок зависит от того, каков блок ввода. Для блока, приведенного выше в качестве примера (см. рис. на стр. 53), ошибка исправляется следующими действиями на клавиатуре:

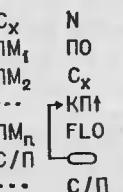
ИПО 1+ПО x_i С/П

К борьбе с ошибками ввода и вывода можно отнести целый набор специальных приемов, повышающих надежность работы оператора с программой. Хорошо сделать так, чтобы программа сама подсказывала оператору порядковый номер числа, которое должно быть введено. Вот, например, цепочка команд КИП4 ИП4 С/П. Команда косвенного вызова КИП4 перед своим выполнением увеличит на единицу содержимое регистра 4. Что она вызовет на индикатор, не очень-то важно. Следующей командой на индикатор будет вызвано содержимое регистра 4, после чего калькулятор остановится. С каждым повторением этой цепочки на индикаторе будут появляться последовательно нарастающие целые числа, под которыми можно понимать порядковые номера вводимых величин и в соответствии с такими подсказками эти величины и набирать на клавиатуре.

Может, правда, оказаться, что вводимые числа похожи на числа-подсказки. В таких случаях во избежание путаницы числам-подсказкам можно придавать особенный вид, скажем, 100, 200...

В заключение стоит рассмотреть несколько специальных случаев ввода чисел. Константы, содержащие немного значащих цифр, проще занести в регистр X, не набирая на клавиатуре, а командами программы. Например, цепочка команд 10.1 11., 12.2 13.5 запишет в регистр X число 1,25; его затем можно направить в любой адресуемый регистр, скажем, командой ПА в регистр А.

Довольно часто приходится засыпать нули в некоторые регистры памяти (нулевыми должны быть, например, начальные значения всевозможных сумм в программах математической статистики, аппроксимации и т. п.). Такого рода операция производится либо вручную способом ПР-ввода, либо автоматически, с помощью представленных здесь блоков:



При выборе подходящего блока следует учитывать, что варианты типа ПП-ввода (рис. слева) позволяют направлять нули в регистры с произвольными, а варианты типа

КП-ввод (рис. справа) — с упорядоченными номерами. Передав управление на первую из команд блока, оператор нажимает клавишу С/П, после чего во все нужные регистры нули засыпаются автоматически, и затем программа останавливается, готовая к продолжению работы.

Навыки безошибочного ввода и вывода информации придут к начинающему программисту постепенно, по мере накопления опыта практической работы. Сделав первые программы и поработав с ними некоторое время, подумайте, что вам в них не нравится, каким образом можно улучшить процесс ввода-вывода данных, если для этого имеются возможности.

Однозначные рекомендации здесь высказать сложно — слишком велико разнообразие решаемых задач. Но все же попытаемся дать краткие сравнительные характеристики рассмотренных выше способов ввода и вывода.

Для коротких и быстрых программ, когда вводится не более трех чисел и возможностей стековой памяти достаточно для вычислений по программе, можно уверенно рекомендовать СТ-ввод. СТ-вывод хорош для двух выводимых показателей, для большего числа выводимых данных он используется значительно реже. СТ-ввод с успехом может быть применен и в сложных программах для циклически повторяющегося ввода небольших порций информации, которые сразу же перерабатываются и отсылаются в адресуемые регистры в переработанном виде.

В случаях, когда СТ-ввод по каким-либо причинам не может быть применен, а желательно сохранить легкую и удобную форму ввода, можно успешно использовать ПП-ввод. При ПП-вводе одновременно происходит размещение выводимых чисел по стековым регистрам, что дает ему преимущество по сравнению с СТ-вводом в тех случаях, когда вводимую информацию нужно не только переслать в адресуемые регистры, но и определенным образом разместить в регистрах стека для последующей обработки. Если выводимых данных много, то их при ПП-способе лучше записывать в виде таблицы — как при вводе, так и при выводе. К недостаткам ПП-способа следует отнести повышенный расход программной памяти, а также то обстоятельство, что при его применении оператору всегда приходится отчетливо помнить, на каком адресе остановилась программа при вводе очередного числа.

КП-ввод исходных данных используется значительно чаще, чем КИП-вывод результатов. Этот способ становится

целесообразным в том случае, если вводится много чисел и их можно разместить в упорядоченной последовательности по адресуемым регистрам. С учетом последнего обстоятельства применение КП-ввода становится эффективным по сравнению с ПП-вводом тогда, когда число вводимых показателей превышает число команд, затрачиваемых на организацию программного блока КП-ввода (его длина составляет обычно шесть-семь команд). КП-ввод лучше использовать тогда, когда в программе обрабатываются однотипные данные и для этой обработки не требуется большого количества адресуемых регистров памяти (например, суммирование какой-либо числовой последовательности).

Применение ПР-ввода не накладывает никаких ограничений на программирование задач, не удлиняет программу. Занести исходные данные этим способом можно еще до набора программы. Безразлична и последовательность записи чисел в регистры. Этот способ остается единственным возможным в случаях, когда программа занимает всю программную память. На него применение лучше ориентироваться также в сложных программах, когда вычисления происходят достаточно долго, когда любая ошибка при вводе показателей может перечеркнуть большую подготовительную работу. В этих случаях лучше сознательно идти на некоторые сложности при вводе, предусматривать проверку записи показателей, чтобы, потеряв несколько дополнительных минут, не терять часы из-за возможных ошибок.

5. Ветви, циклы, подпрограммы

Самая первая программа, составленная нами для вычисления площади круга, была короткой и простой: 00.Fx² 01.Fп 02.× 03.4 04.: 05.C/П.

Простота ее заключается не только в краткости. Обратите внимание: все ее команды выполняются в том порядке, в котором они записаны. Программы, отличающиеся таким свойством, называются линейными. На практике они встречаются чрезвычайно редко. Для большинства программ характерно то, что порядок выполнения команд не совпадает с порядком их записи. Для того чтобы изменить естественный порядок выполнения команд, применяются управляющие конструкции, которые подразделяются на три типа:

— **ветвление**, когда в зависимости от соблюдения или несоблюдения некоторого условия надо выполнить либо одну, либо другую последовательность действий;

— цикл, многократное выполнение каких-либо действий (записанная один раз в программе последовательность команд выполняется несколько раз подряд);

— подпрограмма, то есть реализующая определенный алгоритм, специальным образом оформленная последовательность команд, к которой можно обратиться: потребовать прервать выполнение команд основной программы, выполнить всю последовательность команд подпрограммы, а затем продолжить выполнение команд основной программы.

О трех перечисленных конструкциях и пойдет сейчас речь. Как и ранее, будем иллюстрировать алгоритмы блок-схемами; записывать же их станем на алгоритмическом языке, с которым знакомятся школьники девятых классов.

Для изменения естественного порядка выполнения команд у калькулятора «Электроника Б3-34» имеется операция безусловного перехода БП, четыре операции условных переходов $Fx=0$, $Fx \neq 0$, $Fx < 0$, $Fx \geq 0$, четыре операции организаций циклов FL0, FL1, FL2, FL3, операция перехода к подпрограмме ИПВ и команда возвращения в основную программу В/0. (Есть еще команды косвенных безусловных и условных переходов, но они употребляются крайне редко и потому пока рассматриваться не будут.)

Отметим характерную особенность операций условных переходов: все они основаны на сравнении содержимого регистра X с нулем. Правда, на нашем калькуляторе возможны не все такие проверки: нет операций $x > 0$, $x \leq 0$. Когда необходимо выполнить какую-либо из них, то у величин, подлежащих подобной проверке, меняют знак и проводят сравнения $-x < 0$, $-x \geq 0$ соответственно.

Бетвление. Рассмотрим несложную задачу: взять максимальное из значений переменных a и b , хранящихся в РА и РВ соответственно, и присвоить взятое значение переменной c , хранящейся в РС. Нетрудно составить алгоритм требуемого действия: если $a \geq b$, то положить $c = a$, в противном случае положить $c = b$. На алгоритмическом языке это можно записать так:

если $a \geq b$ то $c := a$ иначе $c := b$ все.

Составим соответствующую программу для нашего калькулятора. Будем рассматривать ее как фрагмент какой-то большой программы и потому станем записывать ее не с нулевого адреса, а с какого-то другого, скажем, с 10-го. (Так будем поступать и при записи дальнейших программ, играющих роль примера.)

Сначала программируем условие $a \geq b$. Видоизменяем

его так, чтобы получилось выполнимое на калькуляторе сравнение с нулем: $a - b \geq 0$.

10. ИПА 11. ИПВ 13. — 13. $Fx \geq 0$ 14. □.

Если условие не выполняется, то произойдет переход по адресу, указанному вслед за операцией сравнения. Начиная с этого адреса, мы будем программировать ветвь **иначе**. Но этот адрес мы пока не знаем, а потому место для него оставляем незаполненным и обводим рамкой, чтобы не забывать о необходимости заполнить его в свое время, когда станет ясно, где расположится ветвь **иначе**.

Если же условие выполняется ($a - b \geq 0$, то есть $a \geq b$), управление будет передано на следующий за операцией сравнения 15-й адрес. Начиная с него и следует запрограммировать ветвь **то**, записать команды пересылки в регистр С числа a , которое хранится в регистре А:

15. ИПА 16. ПС

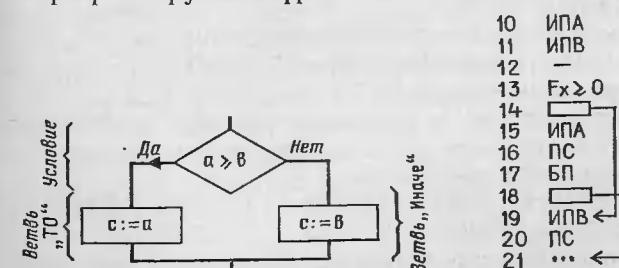
Далее можно разместить ветвь **иначе**. Но писать ее команды сразу вслед за командами ветви **то** нельзя: в таком случае калькулятор выполнит подряд и те, и другие действия, смешает их. Закончив ветвь **то**, надо обойти ветвь **иначе**. Ставим команду безусловного перехода и опять оставляем пустым место для адреса перехода, так как пока не знаем, куда надо перейти:

17. БП 18. □.

А затем программируем ветвь **иначе**, пересылку в регистр С числа b , которое хранится в регистре В:

19. ИПВ 20. ПС.

Для лучшего понимания всю эту цепочку команд, записанных в строчку или в столбик, можно снабдить стрелками, как на рисунках вверху. Там же приведена и блок-схема программируемого фрагмента.



Теперь можно заполнить пропущенные ячейки. Чтобы обойти ветвь **то** (ветвь да на блок-схеме), по адресу 14, после

операции сравнения, записываем 19 — начальный адрес ветви **иначе** (ветви нет на блок-схеме). Чтобы обойти ветвь **иначе**, по адресу 18, после операции безусловного перехода, записываем 21 — начальный адрес последующего фрагмента программы. Заметим: на этот адрес мы попадаем, пройдя и по той, и по другой ветви.

Как видим, записанное на алгоритмическом языке или в виде блок-схемы ветвление переводится в программу для калькулятора по чисто формальным правилам:

1. Запрограммировать левую часть условия, предварительно преобразованного к сравнению с нулем.
2. Записать соответствующую команду условного перехода и оставить незаполненным адрес перехода.
3. Запрограммировать ветвь **то**.
4. Записать команду БП и оставить незаполненным адрес перехода.
5. Запрограммировать ветвь **иначе**.

6. В пропущенную на шаге 2 ячейку записать адрес начала ветви **иначе**; в пропущенную на шаге 4 ячейку записать адрес очередной еще не занятой ячейки.

Тщательная запись алгоритма на алгоритмическом языке превращает дальнейшее составление программы в почти автоматический процесс. Однако наша программа получилась не оптимальной из-за того, что школьный алгоритмический язык не отражает всех возможностей нашего калькулятора. На ветви **то** (как и на ветви **иначе**) можно запрограммировать лишь вызов величины *a* (соответственно *b*) в регистр X, а закончив ветвление, содержимое регистра X надо переслать в регистр C. Так мы получим более короткую программу:

10. ИПА 11.ИПВ 12.— 13.F_x≥0. 14.18 15.ИПА 16.БП 17.19 18.ИПВ 19.ПС.

Пролеживая перемещения чисел по регистрам X, A, B, C, убедитесь, что этот вариант программы дает те же результаты, что и предыдущий.

Если же отказаться от «подстрочного» перевода нашей записи алгоритма на язык команд микрокалькулятора, то возможен еще более короткий вариант:

10. ИПА 11.ИПВ 12.ПС 13.— 14.F_x≥0 15.18 16.ИПА 17.ПС.

Этот пример ставит нас перед вопросом: насколько полезны при составлении программы такие вспомогательные средства, как блок-схема алгоритма и его запись на алгоритмическом языке? И какое из этих средств предпочтительнее?

Конечно, запись на алгоритмическом языке уступает блок-схеме в наглядности, но при своей построчечной структуре, подкрепленная строгими правилами ее перевода на язык команд, эта запись, пожалуй, надежнее позволяет получить работающую программу даже в сложных случаях, при наличии многих циклов и ветвлений. Это тем более верно, если в алгоритмическом языке имеются конструкции, отражающие существенные для программирования особенности микрокалькулятора. К сожалению, используемый нами язык не таков. Блок-схема и в этом отношении превосходит его, способна оказать большую помощь при поиске оптимального варианта программы. Достигнет же оптимума тот, кто владеет всеми возможностями микрокалькулятора (прежде всего возможностями стека) и умело использует их, претворяя алгоритм в программу.

Перевод разветвляющегося алгоритма в программу упрощается, когда при невыполнении условия надо просто ничего не делать, то есть ветвь **иначе** оказывается пустой:

1. Запрограммировать проверку условия.
2. Записать команду условного перехода и оставить незаполненным адрес перехода.
3. Запрограммировать ветвь **то**.
4. В пропущенную на шаге 2 ячейку записать адрес очередной свободной ячейки.

Например, пусть надо получить абсолютную величину числа *a*, хранящегося в РА (забудем на время о том, что проще всего это делается командами F_x² FV). Иными словами, надо выполнить действие:

если *a*<0, то *a*:=—*a* все

Первой командой программы, реализующей этот алгоритм, вызовем в регистр X число *a*, которое предстоит сравнить с нулем: 10.ИПА. Пишем команду сравнения, оставляя пока незаполненным адрес перехода: 11. F_x<0 12... Программируем ветвь **то** — изменяем знак у содержимого РХ и засыпаем результат в РА: 13./—/ 14.ПА. Начальный адрес последующего фрагмента — 15. Его и вписываем в пустующее место по адресу 12. Окончательно:

10. ИПА 11. F_x<0 12.15 13./—/ 14. ПА.

Цикл. Напомним, что так называется предписание о многократном повторении подряд некоторого фрагмента программы. Сам этот фрагмент называется телом цикла. Очередное его повторение определяется проверкой некоторого условия.

Обычно выделяют два типа циклов в зависимости от того, когда производится проверка условия:

— цикл «пока» (сначала проверяется условие, и если оно истинно, то выполняется тело цикла; опять проверяется условие и так далее — таким образом, тело цикла выполняется, пока соблюдается условие);

— цикл «до» (сначала выполняется тело цикла, затем проверяется условие, и если оно не истинно, тело цикла выполняется вновь и так далее — таким образом, тело цикла выполняется до тех пор, как становится истинным условие).

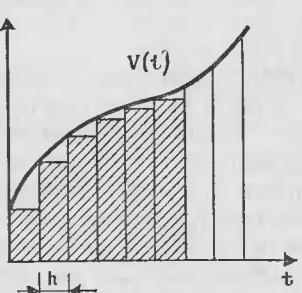
На алгоритмическом языке цикл «пока» записывается так: **пока** (указать условие) **цикл** (описать действия) **кц.**

Цикл «до» записывается так:

цикл (описать действия) **до** (указать условие) **кц.** (Здесь **кц** — служебное слово, означающее «конец цикла».)

Поясним сказанное примером. При взлете самолета его вертикальная скорость v (скорость набора высоты) изменяется в зависимости от времени t по некоторому закону $v(t)$. Надо определить, за какое время самолет достигнет заданной высоты H . Высота S , набранная к моменту времени T , выражается интегралом $S = \int_0^T v(t) dt$. Оставив в

стороне самолет и сосредоточившись на математической стороне дела, получаем задачу: найти T такое, что $\int_0^T v(t) dt = H$.



Для вычисления интеграла возьмем простейшую формулу прямоугольников (см. рисунок). Разобъем площадь под кривой $v(t)$ на столбки ширины h и каждый из них заменим прямоугольником той же ширины с высотой, равной значению функции в левой точке основания. Суммар-

ная площадь таких прямоугольников есть приближенное значение интеграла. Будем набирать прямоугольники, пока их суммарная площадь не превосходит величину H .

Отметим, что при $H=0$ нам не надо брать ни одного прямоугольника. Таким образом, проверку, брать ли очередной прямоугольник, надо производить перед тем, как его взять. Вспоминая, что циклы бывают двух типов, мы можем сказать, что программируемый нами относится к типу «пока».

Запишем лежащий в его основе алгоритм — назовем его ВЗЛЕТ. В заголовке алгоритма перечислим его аргументы (H, h) и результаты (T). Промежуточную переменную S укажем позже, сразу вслед за служебным словом **нач** (начало). Примем, что скорость нарастает как квадрат времени: $v(t)=t^2$.

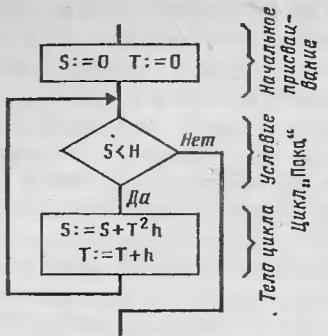
В перечне промежуточных переменных, как видим, нет v и t , которые встречаются в программируемых нами формулах. Первое из этих обозначений излишне, поскольку мы установили явную зависимость $v(t)$. Обозначение t можно объединить с T . Так можно сократить запись алгоритма — избежать присваивания $T:=t$, когда переменная S достигнет значения H : при этом значение T и будет давать ответ поставленной задачи.

Придадим величинам S и T исходные нулевые значения (стартуем в нулевой момент с нулевой высоты). Начинаем записывать цикл.

Напишем условие, при выполнении которого тело цикла должно быть пройдено в очередной раз: **пока** $S < H$. Пока это неравенство соблюдается, величине S при каждом очередном прохождении цикла надо присваивать значение $S+T^2h$, а величине T — значение $T+h$. Перечень этих операций, составляющих тело цикла, предварим служебным словом **цикл**, а завершим служебным словом **кц** (конец цикла).

Если неравенство $S < H$ не соблюдается, должен произойти выход из цикла. Текущее значение величины T при этом и является результатом.

алг ВЗЛЕТ (вещ H, h , T);
арг H, h ; рез T ;
нач вещ S ;
 $S := 0; T := 0;$
пока $S < H$ **цикл**
 $S := S + T^2; T := T + h;$
кц
кон



Запрограммируем разработанный алгоритм. Аргументы и результаты, указанные в первых строках его записи на алгоритмическом языке, а также вспомогательные переменные, указанные далее, напоминают нам: перед составлением программы необходимо распределить адресуемые регистры под хранение величин, фигурирующих в алгоритме. Пусть величина S хранится в Р0, T — в Р1, h — в Р2, H — в Р3. Поскольку начальные значения S и T должны равняться нулю, запишем нули в Р0 и Р1.

10. Сх 11. П0 12. П1.

Преобразуем условие $S < H$ к сравнению с нулем $S - H < 0$ и запрограммируем это сравнение. Адрес перехода пока оставим незаполненным.

13.ИП0 14. ИП3. 15.— 16. $Fx < 0$ 17. □.

При соблюдении условия $Fx < 0$ калькулятор передаст управление на команду, следующую за командой условного перехода. С этого места мы и запишем тело цикла. Завершим его командой безусловного перехода: она должна передавать управление на команду, с которой начинается проверка условия, определяющего очередное прохождение цикла.

18.ИП0 19.ИП1 20. Fx^2 21. ИП2 22. x 23. + 24. П0 25.ИП1
26.ИП2 27. + 28. П1 29.БП 30.13.

Внимательно разберите эту цепочку команд. Проверьте, что команды 18—23 вычисляют выражение $S + T^2 \cdot h$. Команда безусловного перехода (адреса 29—30), как и требуется от нее, передает управление на адрес 13, то есть на команду, с которой начинается проверка условия $S - H < 0$. Если оно выполняется вновь, тело цикла будет пройдено еще раз. Если не выполняется, надо передать управление на начало фрагмента, который должен выполняться по выходе из

цикла, то есть на адрес 31. Его-то и нужно записать под адресом 17, пустующим до сих пор адресом команды условного перехода.

Проверьте, правильно ли составлена и верно ли работает программа. Введите ее в калькулятор, завершив набор командами 31.ИП1 32.С/П. Клавишами F АВТ верните калькулятор в режим вычислений; занесите в регистр 3 число 3, в регистр 2 — число 0, 1; клавишами БП 1 0 передайте управление на адрес 10; клавишей С/П запустите калькулятор. Примерно через две минуты на индикаторе появится результат 2,2. Он довольно сильно отличается от точного (2,08), так как ширина прямоугольников h была выбрана нами довольно большой.

Если разбор приведенной программы покажется вам трудным, укажите переходы стрелками и сравните свой рисунок с помещенным на этой странице.

10	Сх
11	П0
12	П1
13	ИП0 <
14	ИП3
15	—
16	$Fx < 0$
17	□
18	ИП0
19	ИП1
20	Fx^2
21	ИП2
22	Х
23	+
24	П0
25	ИП1
26	ИП2
27	+
28	П1
29	БП
30	□
31	... <

Рассмотренный нами пример позволяет сформулировать правила программирования цикла «пока»:

1. Программируем условие, преобразованное к сравнению с нулем.

2. Записываем команду сравнения и оставляем незаполненным адрес перехода.

3. Программируем тело цикла.

4. Записываем операцию БП, за нею — адрес начала проверки условия.

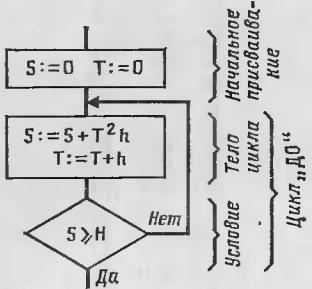
5. В пропущенную на шаге 2 ячейку записываем первый еще не занятый адрес.

Строя только что разобранную программу по типу цикла «пока», мы поступали так лишь потому, что из чисто формальных соображений не захотели исключать из рассмотрения случай $H=0$. Если же заведомо ясно, что величина H не нулевая, то программу для решения той же задачи можно составить по типу цикла «до». В этом случае тот же интеграл, что и ранее, будет подсчитываться до наступления условия $S \geq H$ (то есть условия, противоположного тому, на проверке которого строилась предыдущая программа с помощью цикла «пока»).

На сей раз тело цикла запишется так: сначала, после служебного слова цикл, повторяемые действия, то есть тело цикла, затем, после слова до, условие, затем служебное слово кц.

цикл $S:=S+T^2h; T:=T+h;$

до $S \geq H$ кц



Предлагаем читателю самостоятельно составить программу, реализующую этот алгоритм, и сверить ее с помещенной ниже. Напомним лишь, что условие повторения цикла надо преобразовать к сравнению с нулем: $S-H \geq 0$. Проверять его будем с помощью операции $Fx \geq 0$, которую проставим за телом цикла. А вслед за нею проставим начальный адрес тела цикла. Если неравенство не выполняется, тело цикла будет проидено еще раз. Если выполняется, произойдет выход из цикла.

10.Сх 11.П0 12.П1 13.ИП0 14.ИП1 15.Ф x^2 16.ИП2 17.×
18.+ 19.П0 20. ИП1 21.ИП2 22.+ 23.П1 24.ИП0 25.ИП3
26.— 27. Ф $x \geq 0$ 28.13 29.ИП1 30.С/П.

Как нетрудно усмотреть уже из этого примера, правила программирования цикла «до» несколько проще, чем цикла «пока»:

1. Запрограммировать тело цикла.

2. Запрограммировать левую часть условия, преобразованного к сравнению с нулем.

3. Записать операцию сравнения и за нею — адрес начала тела цикла.

В обоих только что рассмотренных примерах к моменту начала выполнения цикла (к моменту входа в цикл) мы не могли сказать, сколько раз он будет проиден — это определяется по мере всех новых его прохождений. Такие циклы называются итерационными. Но часто встречаются случаи, когда количество повторений цикла можно определить заранее. Например, если бы надо было вычислить интеграл $\int_a^b f(x)dx$ описанным выше методом, цикл нужно было бы пройти столько раз, на сколько отрезков разбит промежуток интегрирования.

Поскольку подобные случаи весьма часто встречаются в практике программирования, для них имеется специальная форма записи на алгоритмическом языке:

цикл (указать число повторений) раз (описать действия)
кц

Именно для организации таких циклов предназначены имеющиеся у «Электроники Б3-34» операции вида FLM (M — номер одного из первых адресуемых регистров, от нулевого до третьего включительно; его в подобных случаях называют счетчиком повторений цикла).

На предыдущих занятиях мы уже не раз использовали эти операции, знаем их свойства. Каждая из них образует команду в соединении с записанным сразу вслед за ней адресом перехода. Такая команда ставится в конце тела цикла и совершает два действия: уменьшает на единицу содержимое регистра M и, если это содержимое не равно единице к моменту выполнения команды, совершает переход на начальный адрес тела цикла. Если же к моменту выполнения команды содержимое регистра M равно единице, то, не уменьшая его, команда передает управление на следующий за нею адрес.

Зная все это, мы сможем сформулировать правила перевода конструкций, описанных выше на алгоритмическом языке, на язык команд «Электроники Б3-34»:

1. Выбрать один из регистров: Р0, Р1, Р2, Р3. Занести в него число повторений цикла.

2. Запрограммировать тело цикла.

3. Записать одну из операций: FL0, FL1, FL2, FL3 (в соответствии с тем, какой регистр был выбран на шаге 1) и за неё — начальный адрес тела цикла.

Чтобы не отступать от принятого порядка изложения, следовало бы разобрать здесь пример на применение операции вида FLM. Попробуйте самостоятельно описать на алгоритмическом языке и затем запрограммировать для

нашего калькулятора вычисление интеграла $\int_a^b f(x)dx$, по-

ложив по-прежнему $f(x)=x^2$. Обозначьте число отрезков, на которые разбит промежуток интегрирования, через n . Воспользуйтесь тем, что переменные величины в описаниях алгоритмов можно обозначать не только буквами, но и словами. Например, вычисляемый интеграл можно так и называть: «интеграл».

Сравните свой вариант с приведенным здесь:

алг ИНТЕГРАЛ (вещ a , b , интеграл, цел n);

арг a , b , n ; рез интеграл;

нач вещ h , x ;

интеграл:=0; $x:=a$;

$h:=(b-a)/n$;

цикл n раз

интеграл:=интеграл+ x^2h ; $x:=x+h$;

кц

кон

10. Сх 11.П5 12.ИП0 13.П4 14.ИП1 15.ИП0 16.—
17.ИП3 18.: 19.П2 20. ИП5 21.ИП4 22.F x^2 23.ИП2 24.×
25.+ 26.П5 27.ИП4 28.ИП2 29.+ 30.П4 31.FL3 32.20
33.ИП5 34.С/П.

Распределение регистров тут таково: a —Р0, b —Р1, h —Р2, n —Р3, x —Р4, интеграл — Р5. Адреса 10—19: очищаем Р5, заносим число a в Р4, вычисляем $h:=(b-a)/n$ и заносим в Р2. Адреса 20—30: тело цикла. Адреса 31—32: записанная здесь команда FL3 20 совершает переход на начальный адрес тела цикла до тех пор, пока содержимое регистра-счетчика Р3 станет равным единице. Тогда команда 31.FL3 32.20 совершит выход из цикла — передаст управление на адрес 33, команду вызова результата на индикатор. Для контроля: при начальных данных $a=0$, $b=2,2$, $n=22$ (введем их, набирая на клавиатуре 0 П0 2,2 П1 22. П3) результат составит 3,311.

Если посмотреть на три последние из составленных нами программ, то можно заметить, что в каждой из них имеется

переменная (скажем, x в самой последней программе, T в остальных), которая на каждом шаге цикла изменяется регулярным образом, то есть на одно и то же значение. Это столь частая в программировании ситуация, что для подобных циклов имеется специальное название «циклы с параметром» (под параметром понимается регулярно меняющаяся переменная). Сюда относится и программа, по которой на втором нашем занятии мы вычисляли среднюю площадь кругов. Напомним один из вариантов этой программы:

10. ИП0 11. 1 12. 4 13.П1 14.Сх 15.КИП1 16.F x^2 17.Фл
18.× 19.4 20.: 21.+ 22.FL0 23.15 24.⇄ 25.:

Здесь намеренно приведен неоптимальный вариант: он нагляднее и легче поддается разбору.

Пусть диаметры двенадцати кругов занесены в следующие регистры: D_1 — в РД, D_2 — в РС, D_3 — в РВ, D_4 — в РА, D_5 — в Р9, D_6 — в Р8 и так далее до Р2. С каждым прохождением цикла (адреса 15—23) вычисляется площадь очередного круга. Иными словами, параметром тут служит индекс k в обозначении величины D_k : он изменяется с шагом 1. В регистр 1 перед входом в цикл занесено число 14. Команда КИП1, как мы уже знаем, при первом же ее выполнении уменьшит содержимое Р1 на единицу (14—1=13) и вызовет в регистр X число, хранящееся в регистре с получившимся в результате вычитания номером. Мы знаем, что номер 13 истолковывается в подобных случаях как обозначение регистра Д. Стало быть, команда КИП1 при первом ее выполнении вызовет в регистр X величину D_1 , при втором — D_2 и так далее. И каждый раз будет подсчитываться площадь круга с вызванным в РХ диаметром и результат будет прибавляться к сумме площадей Σ .

Напомним, что автоматическое уменьшение или увеличение адреса, по которому происходит вызов или засылка, называется его модификацией. И еще один термин: совокупность переменных, имеющих одинаковое имя и различающихся номерами, называется массивом. Номер, по которому конкретные переменные отличаются друг от друга, называется индексом массива. В нашем примере массив образуют диаметры кругов D_k . Здесь D — имя переменной, k — индекс массива, принимающий значения от 1 до 12.

Для программирования циклов, в которых обрабатываются элементы массива, удобны команды косвенного вызова (вида КИПN) и косвенной засылки (вида КПN), при выполнении которых происходит модификация адреса.

Если N равно 0, 1, 2 или 3, то перед каждым выполнением такой команды содержимое регистра уменьшается на единицу. Если же N равно 4, 5 или 6, то увеличивается на единицу.

Поскольку длина массива, обрабатываемого в таких циклах, всегда строго определена, организуются подобные циклы с помощью операции FLM. Так оно было и в нашей последней программе.

Просмотрите ее вновь, вспомните, как она составлялась. Правила программирования циклов в этих случаях таковы:

1. Занести в регистр с номером M (равным 0, 1, 2 или 3) число повторений цикла.

2. Если для вызова и засылки элементов массива предполагается использовать в теле цикла команды КИПН и КПН, где $N=0, 1, 2$ или 3, то занести в регистр N число, на единицу большее, нежели номер регистра, с которого начнется вызов или засылка. Если $N=4, 5$ или 6, то занести в регистр N число, на единицу меньшее, нежели номер регистра, с которого начнется вызов или засылка.

3. Запрограммировать тело цикла.

4. Записать операцию FLM и за нею начальный адрес тела цикла.

(Номера M и N , как правило, не совпадают. Нетрудно сообразить, что при их совпадении косвенный вызов и засылка затрагивали бы регистры, следующие друг за другом не подряд. Впрочем, порой именно это и требуется. Тогда M и N следует полагать совпадающими.)

Условимся о форме записи описанных конструкций на алгоритмическом языке. Запись « $k :=$ адрес a » будем понимать «присвоить переменной k значение адреса переменной a »; запись « $x := [k]$ » будем понимать «присвоить переменной x значение той переменной, адрес которой есть значение переменной k ». (Заметим: мы употребили здесь ту же букву, которой обозначали индекс массива, — и не случайно). Для того, чтобы отразить модификацию адреса с его уменьшением или увеличением, будем применять записи $[-k]$ и $[+k]$ соответственно. То, что минус и плюс стоят вверху слева от символа k , означает, что значение k изменяется перед его использованием.

С применением этих средств алгоритм вычисления средней площади кругов можно записать так:

алг СРЕДНЕЕ (вещ таб $D [1 : n]$, цел n , вещ Σ);

арг D, n ; рез Σ ;

нач цел k ;

$k :=$ адрес $D [n] + 1$; $\Sigma := 0$;
цикл n раз $\Sigma := \Sigma + [-k]^2 \cdot \pi / 4$ кц
 $\Sigma := \Sigma / n$;

кон

(Мы употребили одну и ту же переменную, обозначая сумму площадей и среднюю площадь. Это упростит программирование — позволит использовать один и тот же адресуемый регистр для размещения обеих величин.)

Точный перевод этого алгоритма на язык команд калькулятора даст нам иную программу, нежели приведенная выше. Нам понадобятся адресуемые регистры для величин n, Σ, k , да еще под счетчик повторений цикла, так что под массив останется только 10 адресуемых регистров. Вызов и засылка величины Σ будет требовать дополнительных команд. Этих неудобств можно избежать, если некоторые величины — скажем, n и Σ — хранить не в адресуемых регистрах, а в стеке. Величину Σ лучше хранить в регистре X — тогда ее не надо будет вызывать и засыпать; величину n , используемую реже, разместим выше, в регистре Y .

Когда в регистр X будет вызван диаметр очередного круга для вычисления его площади, величины Σ и n сдвинутся вверх по стековым регистрам. Вычислив площадь очередного круга, надо прибавить ее к сумме площадей Σ . Площадь получается в регистре X , а сумма к этому моменту находится в регистре Y , как и требуется для сложения. После сложения она вернется в регистр X , а количество кругов n окажется также на прежнем месте, в регистре Y . Так оно будет и после последнего сложения, перед вычислением средней площади. Разумеется, перед делением Σ на n обе величины надо будет поменять местами командой \leftrightarrow .

Как видим, размещение этих величин в стеке весьма целисообразно. Под счетчик повторений цикла отведем адресуемый регистр 0, под переменную k — регистр 1. Тогда у нас останется 12 регистров под диаметры кругов.

Приступаем к программированию. Займемся сначала присваиваниями $k :=$ адрес $D [n] + 1$; $\Sigma := 0$, вызовем из Р0 в стек величину n . После выполнения этих действий начальное нулевое значение суммы Σ должно оказаться в регистре X , число n — в регистре Y , а в регистре 1 должно получиться число 14, которое можно «набрать» командами программы (так же, как числа набираются на клавиатуре). Экономнее всего это достигается именно теми командами, с которых начинается знакомый нам вариант программы:

10.ИПО 11 1 12.4 13.П1 14.Сх.

Программируем тело цикла. Прежде всего вызовем в регистр X диаметр круга, с которого начнем вычисления,— величину, обозначенную в записи алгоритма символом $[-k]$. Под переменную k у нас отведен регистр 1, стало быть, нужная нам команда — это 15.КИП1. Дальнейшее программирование формулы $\Sigma := \Sigma + [-k]^2 \cdot \pi / 4$ очевидно:

16.Fx² 17.Fп 18.Х 19.4 20.: 21.+

Тело цикла запрограммировано. Пишем команду возврата к началу цикла: 22. FL0 23.15.

Наконец программируем вычисление средней площади кругов: 24. \Rightarrow 25.: В итоге программа приобретает знакомый нам вид (см. стр. 75).

При буквальном переводе алгоритма на язык команд микрокалькулятора, быть может, программа получилась бы более понятной. Однако при этом она обрабатывала бы меньший массив, была бы длиннее, работала бы дольше. Подобный буквальный перевод естествен для начинающего программиста. Но по мере того как растет опыт программирования, следует стремиться к составлению как можно более эффективных программ, умело используя все возможности калькулятора.

Отметим в этой связи, что для вызова очередного диаметра мы употребили команду КИП1. Вместо нее можно было написать команду КИП0, КИП2 или КИП3: как и КИП1, они тоже вызывают числа из регистров с убывающими номерами. Но регистр 0, с которым «работает» команда КИП0, уже занят под счетчик повторений цикла. Если же употребить команду КИП2 или КИП3, то тем самым пришлось бы занять регистры 2 или 3 соответственно, а тогда массив, где хранятся диаметры кругов, опять-таки сузился бы. Выбирая для организации цикла регистры 0 (операция FL0) и 1 (команда КИП1), то есть самые первые из адресуемых регистров, мы достигаем предельно возможной длины массива чисел, обрабатываемых в цикле. Такой выбор часто встречается на практике.

Если бы нам требовалось вызывать значения величин, хранящихся в регистрах с нарастающими номерами, то в записи алгоритма мы употребили бы символ $[+k]$, а при программировании — команду вида КИПN, где N=4,5 или 6. Поскольку регистр N в этом случае находится уже не в начале, а ближе к середине ряда адресуемых регистров, длина обрабатываемого массива будет сильно ограничена. Вот почему в программах для «Электроники Б3-34» в подоб-

ных случаях сравнительно редко встречается такая модификация адреса, при которой номера регистров увеличиваются.

Подпрограммы. Обратимся вновь к задаче о вычислении определенного интеграла. Программу для этой цели мы составили, полагая подынтегральную функцию $f(x)$ равной x^2 . Ну а если потребуется вычислить интеграл от другой функции? Нужно заменить команды, по которым вычисляется значение функции, и внести их вместо прежних в программную память калькулятора. Как это делается, мы уже знаем. Но как быть, если вычисление новой функции требует большего числа команд, чем прежней? Тогда программу придется переписывать и вводить в калькулятор заново.

Чтобы не возникала неприятная необходимость в переписывании всей программы, можно поступить так: вынести за ее пределы фрагмент, где вычисляется подынтегральная функция. Когда потребуется получить ее значение от заданного аргумента, будем передавать управление на начальный адрес такого фрагмента, а после его выполнения — возвращаться к основной программе (на тот адрес, который значится за командой передачи управления).

Смена подынтегральной функции теперь повлечет за собой лишь перезапись выносного фрагмента. Будь он короче или длиннее прежнего, перемена никак не отразится на основной программе.

Такие фрагменты и называются подпрограммами. Они оказываются полезными не только в случаях, подобных описанному, но и тогда, когда по ходу выполнения программы приходится в разных ее местах выполнять одну и ту же последовательность команд. Эту последовательность такжеывает целесообразным вынести за пределы основной программы в виде подпрограммы и обращаться к ней в нужные моменты так, как описано выше. С подобным примером мы вскоре встретимся.

Обращение к подпрограмме следует оговаривать еще на той стадии решения задачи, когда на алгоритмическом языке записывается алгоритм ее решения. В его тексте в нужном месте приводится обращение к вспомогательному алгоритму, который будет оформлен в виде подпрограммы. Его записывают отдельно.

Попробуем написать по-новому, со сменным вспомогательным фрагментом алгоритм для вычисления определен-

ного интеграла. Там, где в прежнем варианте стояла строчка с формулой подынтегральной функции, напишем: **ФУНКЦИЯ** — озаглавим так нужный нам здесь вспомогательный алгоритм.

```
алг ИНТЕГРАЛ (вещ a, b, интеграл, цел n);
    арг a, b, n; рез интеграл;
нач вещ h, x;
    h:=(b-a)/n; интеграл:=0, x:=a;
    цикл n раз
        интеграл:=интеграл + функция (x)·h; x:=x+h;
    кц
кон
```

Далее запишем вспомогательный алгоритм для вычисления подынтегральной функции. Пусть на сей раз она будет такой: $f(x)=x+x^2$.

```
алг вещ ФУНКЦИЯ (вещ x);
нач знач:=x+x2;
кон
```

Во вспомогательном алгоритме может использоваться несколько аргументов и получаться несколько результатов. Мы разбираем здесь простейший пример, когда по одному аргументу вычисляется один результат, значение функции.

Для программирования подпрограмм у нашего калькулятора имеется операция ПП («Переход к Подпрограмме») и команда В/0 (здесь мы ее расшифруем как «Возврат к Основной программе»). Операция ПП выполняет переход на указанный вслед за нею адрес, начальный адрес подпрограммы; при этом калькулятор запоминает место, откуда произошел переход. Команда В/0 ставится в конце подпрограммы и, когда очередь дойдет до нее, совершает возврат на запомненное место, то есть к продолжению основной программы.

Составление подпрограммы требует известной осмотрительности. Например, необходимо следить за тем, чтобы участвующие в ее работе адресуемые регистры не совпадали с теми, где хранятся числа, необходимые для дальнейшей работы основной программы. Аргументы, используемые в подпрограмме, лучше всего к моменту обращения к ней располагать в регистрах стека. Там же рекомендуется размещать результаты работы подпрограммы к моменту возвращения к основной программе.

Как всегда в подобных случаях, сформулируем правила программирования подпрограмм:

1. В том месте основной программы, где надо выполнить подпрограмму, записать команды засылки в стек ее аргументов, операцию ПП, а следующий за ней адрес оставить незаполненным; далее записать команды вызова из стека ее результатов.

2. Продолжить запись основной программы.

3. Начиная с адреса, следующего за последним адресом основной программы, записать подпрограмму.

4. В конце подпрограммы поставить команду В/0.

5. В пропущенную на шаге 1 ячейку вписать начальный адрес подпрограммы.

К одной основной программе может быть «приписано» одновременно несколько подпрограмм. Переходы на них в основной программе осуществляются операцией ПП по указываемым вслед за нею соответствующим адресам.

Составим заново программу для вычисления определенного интеграла, полагая подынтегральную функцию $f(x)=x+x^2$ и вынося ее вычисления в подпрограмму.

Начало программы остается прежним вплоть до адреса 21, где в регистр X вызывается величина x. Далее пишем обращение в подпрограмму, вычисляющей значение $f(x)$: ставим операцию ПП, а адрес обращения (начальный адрес подпрограммы) пока оставляем пустым. Значение подынтегральной функции, очевидно, получится в регистре X. Так оно было и в прежнем варианте программы. Поэтому новую программу мы можем закончить командами прежней. Они разместятся по адресам 24—35.

А дальше можно писать подпрограмму — начиная прямо с адреса 36. Его и поставим в пустовавшую до сих пор ячейку, вслед за операцией ПП. Вся подпрограмма уложится в три команды: 36.↑ 37. Fx² 38. + (проследите, что в регистре X получается величина $x+x^2$). Не забудем поставить в конце подпрограммы команду В/0.

10.Cx 11.П5 12.ИП0 13.П4 14.ИП1 15.ИП0 16.—17. ИП3 18.: 19.П2 20.ИП5 21.ИП4 22.ИП1 23.36 24.ИП2 25.× 26.+ 27.П5 28.ИП4 29.ИП2 30.+ 31.П4 32.FL3 33.20 34.ИП5 35.С/П 36.↑ 37.Fx² 38.+ 39.В/0.

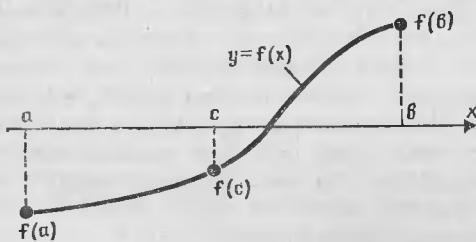
При тех же начальных данных, что прежде (0 П0 2,2 П1 22 П3), результат равен 5,621.

Комбинирование ветвлений и циклов. Если посмотреть на блок-схемы, где изображены ветвление и цикл, то можно увидеть, что каждая из этих конструкций имеет один вход и один выход. Во фрагментах программ, реализующих обе конструкции, вход в каждую из них происходит по одному

адресу; по выходе из нее мы попадаем опять-таки на один адрес — начальный адрес следующего фрагмента. Это позволяет строить комбинации ветвлений и циклов.

Имеется всего два способа подобного комбинирования: следование и вложение. При следовании выход одной конструкции сцепляется со входом другой. Соответствующие части программы размещаются друг за другом, и при программировании никаких проблем не возникает. Сложнее дело обстоит при вложении, когда в качестве тела цикла, ветви то или ветви иначе, в свою очередь, используется цикл или ветвление. Чтобы не ошибаться при программировании столь сложных комбинаций, надо придерживаться простого принципа: каждая конструкция программируется согласно приводившимся выше правилам независимо от того, является она вложенной или нет.

Запрограммируем ради примера алгоритм нахождения корня функции методом половинного деления. Пусть в концах отрезка $[a, b]$ непрерывная монотонная функция $f(x)$ принимает значения разных знаков. Тогда на отрезке $[a, b]$ у нее есть только один корень. Возьмем середину отрезка x и из двух образованных ею отрезков $[a, x]$ и $[x, b]$ выберем тот, на концах которого функция опять-таки принимает значения разных знаков. С выбранным отрезком повторим тот же процесс (см. рисунок). На каждом шаге длина



отрезка уменьшается вдвое. Когда она станет меньше некоторого заданного ε , то любую точку отрезка (скажем, его середину) можно с абсолютной погрешностью ε считать корнем функции.

Обозначим левый конец очередного отрезка *лев*, правый — *прав*. Их начальные значения: *лев*=*a*, *прав*=*b*. Выбор нужной половины после деления отрезка пополам будем производить, перемножая значения $f(a)$ и $f(x)$. Допустим, произведение отрицательно. Тогда в силу монотонности функции отрицательным будет и произведение ее значений в концах отрезка $[лев, x]$. Стало быть, функция

меняет знак на этом отрезке, его и нужно выбрать — присваиваем переменной *прав* значение *x*. Если же произведение $f(a) \cdot f(x)$ положительно, функция не меняет знак на отрезке $[лев, x]$ и меняет его на отрезке $[x, прав]$, который и следует выбирать, — присваиваем значение *x* переменной *лев*. Так же поступаем, если $f(a) \cdot f(x)$ равно нулю.

Алгоритм решения задачи можно записать так:

```
алг КОРЕНЬ (вещ лев, прав, x, ε);  
    арг лев, прав, ε; rez x;  
    нач вещ fa;  
        fa:=f (лев);  
        пока прав — лев ≥ε цикл  
            x:=(лев+прав)/2;  
            если f(x) · fa ≥0, то лев:=x иначе прав:=x все  
        кон кц
```

Приступим к программированию. Начнем с распределения адресуемых регистров. Перебрав знакомые нам методы ввода исходной информации, легко сообразить, что здесь удобно применить СТ-ввод. Величины *a*, *b*, *ε* наберем на клавиатуре, перенежая их набор нажатием клавиши \uparrow . Тогда *a* окажется в регистре *Z*, *b* — в регистре *Y*, *ε* — в регистре *X*. Отсюда их нужно будет тотчас разослать по адресуемым регистрам, распределение которых может быть таким: *ε* — *P0*, *fa* — *P1*, *лев* — *P2*, *прав* — *P3*, *x* — *P4*.

Поскольку наша задача имеет вполне самостоятельный характер, программу для ее решения будем писать, начиная с адреса 00. Первыми же командами вычислим величину *fa*, разместим по своим местам величины *ε* и *fa*, а также начальные значения переменных *лев* и *прав*, равные соответственно *a* и *b*:

00.П0 01.ФО 02.П3 03.ФО 04.П2 05.ПП 06.□ 07.П1.

Предлагаем читателю самостоятельно разобрать этот фрагмент и убедиться, что после выполнения команд с адресами 00—04 величина *ε* оказывается в *P0*, *b* — в *P3*, *a* — в *P2* и в *PX*. Далее выполняется подпрограмма, вычисляющая значение функции. Начальный адрес подпрограммы мы пока не знаем и потому оставляем незаполненным. Когда мы будем ее писать, учтем, что аргумент функции берется из регистра *X*. Напомним попутно, что исходную информацию для выполнения подпрограмм бывает удобно размещать именно так — в стеке. Там же удобно оставлять и результаты их выполнения — в данном случае значение

f(a). Командой 07.П1 мы отсылаем эту величину в Р1, чтобы потом ее не надо было многократно вычислять в цикле.

Программируем следующую строку алгоритма — проверку условия цикла:

08.ИП3 09.ИП2 10.— 11.ИП0 12.— 13.F $x \geq 0$ 14.□.

Оставив незаполненным адрес 14, переходим к программированию тела цикла. Сначала находим величину x — середину отрезка [лев, прав] — и отсылаем ее в Р4:

15. ИП2 16.ИП3 17.+ 18.2 19.: 20.П4 21.С/П.

После вычисления середины отрезка мы поставили команду С/П. Эта команда поможет нам при отладке программы: определив величину x , калькулятор будет останавливаться, и мы сможем проследить, правильно ли работает программа. Удостоверившись в ее правильной работе, мы заменим эту команду на «пустую» — КНОП.

Теперь программируем ветвление. Работаем с ним по стандартным правилам, не обращая внимания на то, что оно вложено в цикл. Величину x можно в РХ не вызывать, так как она уже находится там, а сразу обращаться к подпрограмме:

22.ПП 23.□ 24.ИП1 25.× 26.F $x \geq 0$ 27.□ 28.ИП4 29.П2 30.БП 31.□ 32.ИП4 33.П3.

В тексте алгоритма мы дошли до служебного слова **все**, завершающего ветвление. Значит, теперь мы можем заполнить адреса, указывающие порядок переходов в зависимости от выполнения или невыполнения условия 26. F $x \geq 0$. В соответствии с правилами программирования ветвлений по адресу 27 записываем 32, адрес начала ветви **иначе**, а по адресу 31 записываем 34, очередной свободный адрес.

Пройдя в тексте алгоритма служебное слово **все**, мы оказываемся на служебном слове **кц**. Оно отмечает конец цикла. Иными словами, оно означает, что тело цикла за-программировано полностью. Судя по тексту алгоритма, он принадлежит к типу «пока». В соответствии с правилами программирования подобных циклов записываем операцию БП и за нею — адрес начала проверки условия цикла, то есть 08:

34.БП 35.08.

Затем в соответствии с теми же правилами по адресу 14, который был оставлен незаполненным в конце проверки условия цикла, записываем 36 — очередной свободный адрес.

На этот адрес 36 мы попадаем, когда длина сужающегося с каждым делением отрезка [лев, прав] становится меньше допустимой абсолютной погрешности ε . Середину этого отрезка примем за искомое значение корня функции:

36.ИП4 37.С/П.

Программирование закончено.

Итак, составленная нами программа заняла адреса до 37-го включительно. Стало быть, подпрограмму можно размещать с адреса 38. Его и запишем по пустующим до сих пор адресам 06 и 23.

Вот окончательный текст программы:

00.П0 01.Ф○ 02.П3 03.Ф○ 04.П2 05.ПП 06.38 07.П1
08.ИП3 09.ИП2 10.— 11.ИП0 12.— 13.F $x \geq 0$ 14.36. 15.ИП2
16.ИП3 17.+ 18.2 19.: 20.П4 21.С/П 22.ПП 23.38 24.ИП1
25.× 26.F $x \geq 0$ 27.32 28.ИП4 29.П2 30.БП 31.34 32.ИП4
33.П3 34.БП 35.08 36.ИП4 37.С/П.

Возьмем монотонную функцию $f(x) = x^3 - 1$ и попробуем отыскать ее корень на отрезке [0, 3] с точностью 0.01 по составленной нами программе.

Дополним текст программы подпрограммой вычисления исследуемой функции. В конце подпрограммы не забудем поставить команду В/0:

38. F x^2 39.1 40.— 41.В/0.

Вводим в калькулятор программу и подпрограмму, возвращающуюся в режим вычислений (F АВТ), устанавливаем счетчик адресов на нулевой адрес (В/0), вводим в стек числа, выраждающие левый конец отрезка, правый его конец, точность вычислений ($0 \uparrow 3 \uparrow 0, 01$), и пускаем программу на счет (С/П).

Покуда желаемая точность не достигнута, калькулятор будет останавливаться по команде 21.С/П и на индикаторе будут последовательно загораться числа 1.5, 0.75, 1.125, 0.9375, 1.03125, 0.984375...

Поиск корня идет верно. Заменяем команду 21.С/П командой 21.КНОП (БП 21 F ПРГ К НОП), возвращаемся в режим вычислений (F АВТ) и даем команду на продолжение счета (С/П).

Вскоре, достигнув установленной точности, калькулятор останавливается. На индикаторе — приближенное значение корня 1.0019531. От точного значения, единицы, оно действительно отличается менее чем на 0.01.

6. Погрешности вычислений

Мы получили результат вычислений на микрокалькуляторе. Мы применяем его в нашей практической работе. Исходя из него, мы делаем какие-то выводы, предположения.

Чтобы правильно использовать или истолковать результат, необходимо знать достоверность проведенных расчетов, выявить ошибки, которые могли повлиять на точность наших вычислений, оценить погрешность полученного результата.

Источниками погрешностей и ошибок при вычислениях могут быть недостаточно точное отображение реальных явлений их математической моделью (погрешности модели), приближенное значение величин, входящих в условие задачи (погрешности исходных данных), применение приближенных расчетных формул вместо точных (методические погрешности), особенности работы операционного устройства микрокалькулятора (операционные погрешности).

Все это приводит к тому, что результат вычислений a всегда является приближенным, отличается от точного значения a^* искомой величины. Абсолютная величина их разности называется истинной абсолютной погрешностью.

Разумеется, не зная искомую величину в точности (иначе зачем ее вычислять?), мы не можем назвать истинную погрешность наших вычислений. Остается лишь оценивать ее заведомо превосходящим ее числом. Это число обычно обозначают прописной греческой буквой Δ и называют абсолютной погрешностью приближенной величины a ; его отношение к абсолютной величине $|a|$ именуют относительной погрешностью и обозначают строчной греческой буквой δ .

Оценка истинной погрешности становится достовернее, если принять число Δ достаточно большим. Но практическая ценность оценки при этом, конечно, снизится. Так возникает проблема: анализируя процесс получения приближенного результата, оценить его отклонение от истинного по возможности меньшим числом Δ , называемым в таком случае предельной абсолютной погрешностью.

Абсолютная погрешность
 $\Delta > |a - a^*|$
Приближенное Точное
значение значение

$$\delta = \frac{\Delta}{|a|}$$

Относительная
погрешность

Если такая оценка согласуется с допустимой точностью расчета, то приближенный результат можно использовать как точный. Если не согласуется — встает новая проблема: повысить точность вычислений.

Обеим этим проблемам и будет посвящен наш рассказ. Но прежде чем приступить к нему, сделаем несколько замечаний.

На практике точность вычислений чаще оценивают относительной погрешностью, но нередко удобной бывает и абсолютная. Выбор определяется существом задачи. Отметим, что абсолютная погрешность — величина размерная, и размерность у нее та же, что у определяемой величины. Относительная же погрешность — величина безразмерная.

Числа, с которыми оперирует любая вычислительная машина (и микрокалькулятор в том числе), вводят в нее и выводят в десятичном представлении. Если погрешность Δ не превышает половины единицы некоторого n -го разряда ($\Delta \leq 0.5 \cdot 10^{-n}$), то все цифры в целой части числа и n первых слева цифр в дробной его части называются верными.

Поскольку величина Δ служит лишь оценкой погрешности, не имеет смысла приводить ее с большой точностью. Обычно ее округляют до одной-двух значащих цифр: $x = -6,72 \pm 0,01$, $y = 420,54 \pm 0,15$ и т. д. Две значащие цифры в погрешности целесообразно оставлять лишь тогда, когда первая значащая цифра равна 1 или 2 (например, если $\Delta x = 0,15$, то, округлив погрешность до $\Delta x = 0,2$, мы весьма сильно изменим ее). Последняя значащая цифра результата вычислений должна быть величиной того же порядка, что и погрешность. Скажем, если абсолютная погрешность составляет 0,05, а на индикаторе микрокалькулятора мы получили число 84,359265, то результат следует записать $x = 84,36 \pm 0,05$.

Погрешности модели. Всю совокупность факторов реального физического явления невозможно отразить в его математической модели. Чем больше факторов мы учтем, тем сложнее анализ явления и расчет по его математической модели. Поэтому принимают различные условия и допущения, упрощающие решение задачи.

В небольшой статье трудно дать какие-то общие правила для составления математических моделей ввиду большого разнообразия явлений природы. Поэтому здесь мы ограничимся лишь словами академика А. Н. Крылова, полезными для начинающего программиста: «Сколько бы ни было точно математическое решение, оно не может быть точнее тех

приближенных предпосылок, на коих оно основано. Об этом часто забывают, делают вначале какое-нибудь грубое приближенное предположение или допущение, часто даже не оговорив таковое, а затем придают полученной формуле гораздо большее доверие, нежели она заслуживает».

Погрешности исходных данных. Вычислителю, как правило, приходится работать с приближенными числами, полученными при проведении экспериментов, при обработке статистических данных, при округлении результатов предыдущих расчетов и т. п. При выполнении любой математической операции над приближенными числами присущие им погрешности обуславливают погрешности результата, причем для каждой операции по-своему (см. табл. 2).

Таблица 2

Результат операции	Чувствительность
$f = x_1 + x_2$	$S_1 = x_1/(x_1 + x_2), S_2 = x_2/(x_1 + x_2)$
$f = x_1 - x_2$	$S_1 = x_1/(x_1 - x_2), S_2 = -x_2/(x_1 - x_2)$
$f = x_1 x_2$	$S_1 = S_2 = 1$
$f = x_1/x_2$	$S_1 = 1, S_2 = -1$
$f = x_1/x_2$	$S_1 = x_2, S_2 = x_2 \ln x_1$
$f = 1/x$	$S = -1$
$f = x^2$	$S = 2$
$f = \sqrt{x}$	$S = 1/2$
$f = e^x$	$S = x$
$f = 10^x$	$S = x \ln 10$
$f = \ln x$	$S = 1/\ln x$
$f = \lg x$	$S = 1/\ln x \lg x$
$f = \sin x$	$S = x/\tan x$
$f = \cos x$	$S = -x \tan x$
$f = \tg x$	$S = 2x/\sin 2x$
$f = \arcsin x$	$S = x/(\sqrt{1-x^2} \arcsin x)$
$f = \arccos x$	$S = -x/(\sqrt{1-x^2} \arccos x)$
$f = \operatorname{arctg} x$	$S = x/((1+x^2) \operatorname{arctg} x)$

Внимательное рассмотрение таблицы подсказывает, что один из самых коварных источников погрешностей — вычитание близких величин. Для примера вычтем из числа $x_1 = 110.5 \pm 0.1$ число $x_2 = 110.4 \pm 0.1$. Результат: 0.1 ± 0.2 . Как видим, относительная погрешность, равная $0.2/0.1$, составляет 200 процентов, что неприемлемо ни при каком разумном приближении.

Нужно также заметить, что при вычислении элементарных функций на микрокалькуляторе аргумент может по-

падать в «опасные зоны», где его незначительная относительная погрешность вызывает гораздо более высокую относительную ошибку в определении функции. Так оно происходит, например, при вычислении $\arcsin x$ при x , близком к единице.

В подобных случаях говорят о высокой чувствительности результатов операции к погрешностям ее участников (операндов). Кстати, само слово «чувствительность» — строгий математический термин. Так называют коэффициенты в представлении относительной погрешности результата операции через относительные погрешности операндов.

$$\begin{array}{l} \text{Результат} \\ \text{вычислений} \\ \delta f = \sum_i s_i \delta x_i \\ \text{Относительная} \\ \text{погрешность} \\ \text{результата} \end{array} \quad \begin{array}{l} \text{Чувствительность} \\ i - \text{го} \\ \text{исходного} \\ \text{значения} \end{array}$$

При вычислениях на микрокалькуляторе погрешность конечного результата можно определить шаг за шагом, учитывая погрешности каждой выполненной операции. Так можно выяснить, насколько точными должны быть исходные данные, чтобы обеспечить приемлемую точность результата. Излишняя точность почти всегда оказывается ненужной. Каждая лишняя цифра в исходных данных увеличивает время ее ввода в калькулятор, а следовательно, и общее время счета, не делая результат более точным. Длинный ряд недостоверных цифр в полученном числе может только вводить в заблуждение. «Помните, что каждая неверная цифра — это ошибка, всякая лишняя цифра — это пол ошибки», — уместно снова процитировать слова академика А. Н. Крылова.

Обычно округление чисел при их сложении и вычитании производят таким образом, чтобы в них оставалось на один десятичный разряд больше, чем имеется верных знаков в наименее точно заданном числе (если более уточнить его нельзя) или чем предписано необходимой точностью результата вычислений.

Если при вычислениях не требуется строгий расчет погрешности, то можно рекомендовать такие правила:

— при сложении и вычитании чисел в результате следует сохранять столько десятичных знаков, сколько их имеет число с наименьшим количеством верных знаков;

— при умножении и делении чисел в результате следует сохранять столько значащих цифр, сколько их имеет число с наименьшим количеством верных значащих цифр;

— при возведении числа в квадрат или куб в результате необходимо сохранить столько значащих цифр, сколько верных значащих цифр в основании степени;

— при извлечении квадратного корня в результате необходимо сохранить столько же значащих цифр, сколько верных значащих цифр в подкоренном числе;

— при вычислении промежуточных результатов в них следует оставлять на одну цифру больше, чем рекомендуют предшествующие правила, а в окончательном результате эту цифру отбросить.

Использование этих правил при расчетах на микрокалькуляторе, конечно, не означает, что надо округлять каждый промежуточный результат. Однако их целесообразно учитывать при ручных расчетах, при вводе исходных данных для решения задачи по программе и при оценке точности окончательного результата.

Операционные погрешности обычно возникают из-за ограниченного количества разрядов мантиссы вводимых чисел при их сложении и вычитании, из-за недостаточного числа разрядов в операционных регистрах при умножении и делении чисел, из-за недостаточной точности, с которой вычисляются элементарные функции, и т. п.

Округление результата той или иной математической операции, когда его приходится укладывать в восемьизрядную сетку калькулятора, производится «Электроникой Б3-34» по-разному, в частности простым отбрасыванием не укладывающихся разрядов. Другие способы округления, которым также «обучен» наш калькулятор, более точны. Они применяются при сложении и вычитании чисел. Но если приходится оценивать погрешность округления, то для достоверности оценки лучше полагать ее соответствующей отбрасыванию, то есть равной единице самого младшего из разрядов, отображаемых на индикаторе.

Погрешности значений элементарных функций, вычисляемых на микрокалькуляторе, приведены в руководстве по его эксплуатации. Особенно большие относительные погрешности, как правило, возникают при вычислении \sqrt{x} и x^y , поэтому многократное применение таких команд в программе может привести к существенной ошибке.

Методические погрешности. Применение приближенных формул вместо точных позволяет проводить на микрокаль-

куляторе расчет достаточно сложных математических выражений. Этот прием основан на замене функций их разложением в ряды, замене производной — разностью, интеграла — суммой и т. д.

Разумеется, используя приближенные формулы, надо всегда четко знать диапазон аргумента, для которого они применимы, и точность вычисления по ним. Обычно такие сведения даются в справочниках. В общем случае при вычислениях на микрокалькуляторах погрешность метода целеусобразно выбирать такой, чтобы она была в несколько раз меньше погрешности исходных данных.

Порой расчет по сложной точной формуле удается заменить расчетом по более простой, но применяемой много-кратно, в процессе последовательных приближений. Здесь следует предостеречь от излишнего доверия распространенному приему, когда точность вычислений оценивают разностью между двумя последовательными результатами. Хрестоматийный пример на этот счет представляет собой так называемый гармонический ряд:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots$$

С нарастанием числа слагаемых его сумма стремится к бесконечности, между тем разность между последовательными значениями суммы — к нулю. Наивный подход к делу диктует сравнивать эти значения и прекращать суммирование, когда они достаточно близки, скажем, взять сто первых слагаемых ряда, если желательна абсолютная погрешность, не превосходящая 0.01. Однако ни об этой, да и ни о какой-либо иной точности здесь говорить не приходится, поскольку бессмыслен сам поиск суммы гармонического ряда, растущей до бесконечности с неограниченным ростом числа его слагаемых.

Когда вычисления на калькуляторе ведутся с помощью метода последовательных приближений, случается, что при достижении определенного количества верных знаков дальнейшие приближения в силу операционных погрешностей начинают последовательно принимать несколько повторяющихся неодинаковых значений. Порой эти повторяющиеся результаты различаются между собой на довольно значительную величину. Точность, меньшая этой разности, уже недостижима для избранного метода.

Подобная неприятность нередко затрудняет вычисление рядов и интегралов. Теоретически точность результата долж-

на неограниченно растя, когда увеличивается число слагаемых ряда или число отрезков, на которые разбит промежуток интегрирования. Но при этом увеличивается и число выполняемых операций, а вместе с ними растет и общая операционная погрешность, так что и тут сколь угодно высокая точность результата оказывается недостижимой в рамках избранного метода и алгоритма.

Совокупная оценка результата вычислений, как видим, требует учитывать не только погрешности всех видов (методические и операционные, погрешности модели и исходных данных), но и их взаимосвязи. Даже в простых на первый взгляд случаях это может оказаться делом нелегким, достоверная оценка — неочевидной.

Для примера найдем относительную погрешность величины K , вычисленной по формуле $K = x_1^2 \cdot x_4 / (x_2 \cdot x_3^2)$. Пусть $x_1 = 1.8 \pm 0.2$; $x_2 = 2.8 \pm 0.3$; $x_3 = 1.1 \pm 0.1$; $x_4 = 0.45 \pm 0.04$. Тогда $\delta K = 2 \cdot 0.2/1.8 + 0.3/2.8 + 2 \cdot 0.1/1.1 + 0.04/0.45 = 0.6$. Относительная погрешность результата составила почти 60 процентов, хотя относительная ошибка каждого из чисел, участвующих в вычислении, находится в пределах между 9 и 11 процентами.

В то же время бывает и так, что погрешности чисел, участвующих в том или ином вычислении, взаимно компенсируются. Учет этого явления вполне возможен, но он требует довольно сложных математических средств. Для ознакомления с ними и с различными другими вопросами приближенных вычислений можно порекомендовать литературу, список которой приведен в конце книги.

Повышение точности результатов вычислений. Здесь имеет смысл перечислить лишь несколько простых рецептов, не вдаваясь в их математическое обоснование (читатель найдет его в рекомендуемой литературе).

Погрешность суммы нескольких чисел при расчете на микрокалькуляторе уменьшается, если начинать суммирование с меньших по величине слагаемых.

Определим ради примера сумму трех чисел, складывая их в том порядке, как они записаны: $S = 200013,11 + 200,05399 + 106,51401 = 200\ 319,67$. Проводя суммирование в обратном порядке, от третьего слагаемого до первого, получим более точное значение $S = 200\ 319,68$.

Как видим, переместительный закон сложения не выполняется при вычислениях на микрокалькуляторе: сумма меняется от перемены мест слагаемых. Дальнейшие примеры покажут нам, что при калькуляторных расчетах не

выполняются и другие законы арифметики: сочетательные законы сложения и умножения, переместительный закон умножения, распределительный закон умножения относительно сложения и вычитания. Причины подобных парадоксов — в округлении промежуточных результатов, когда многозначные числа не умещаются в разрядную сетку индикатора. Благодаря этой причине и оказывается небезразличным порядок, в котором следует выполнять арифметические операции, чтобы результат был возможно точнее.

Если складывается достаточно много чисел, то их лучше разбить на группы из слагаемых, близких по порядку величины, провести суммирование в группах по предыдущей рекомендации, после чего полученные суммы сложить, начиная с меньшей.

Вычисляя разность двух чисел (особенно близких по величине), целесообразно вынести за скобки их множитель. Для примера вычислим на нашем микрокалькуляторе величину $P = 6,250001 \cdot 16 - 25,000003 \cdot 4 = 1 \cdot 10^{-5}$. Вынесем число 4 за скобки и вновь проведем вычисления. $P = 4(6,250001 \cdot 4 - 25,000003) = 4 \cdot 10^{-6}$. Взяв в выкладках достаточное число разрядов, можно убедиться, что второй результат точнее.

Уменьшить погрешность разности позволяют преобразования:

$$\begin{aligned} (a + \varepsilon)^2 - a^2 &= \varepsilon(2a + \varepsilon); \\ \sqrt{a + \varepsilon} - \sqrt{a} &= \varepsilon / (\sqrt{a + \varepsilon} + \sqrt{a}); \\ a - \sqrt{a^2 + \varepsilon} &= -\varepsilon / (a + \sqrt{a^2 + \varepsilon}); \\ 1 - a / (a + \varepsilon) &= \varepsilon / (a + \varepsilon). \end{aligned}$$

Здесь ε — малое по сравнению с a число.

Иногда разность двух близких чисел может встретиться неожиданно, даже при решении хорошо известной задачи. Допустим, нам попалось не совсем обычное квадратное уравнение $x^2 + 1000x + 1 = 0$. Попытаемся найти наименьший по величине корень этого уравнения. Воспользовавшись известной формулой и считая на микрокалькуляторе, получаем: $x_1 = -500 + \sqrt{250000 - 1} = -500 + 499,99899 = -1,01 \times 10^{-3}$. Подставляя значение корня в уравнение, убеждаемся, что решение выполнено с большой погрешностью. Это произошло из-за того, что в примененной нами формуле для x_1 присутствует разность двух близких чисел — 500 и $\sqrt{250000 - 1}$. Значительно более точное значение корня

можно получить, если по только что данной рекомендации решение представить в виде

$$x_1 = -1/(\sqrt{250000-1} + 500) = -1,000001 \cdot 10^{-3}.$$

При перемножении нескольких чисел сначала надо умножить наибольший множитель на наименьший; из оставшихся сомножителей следует взять либо наибольший, либо наименьший, судя по тому, меньше или больше единицы абсолютная величина промежуточного результата. Если делятся друг на друга два произведения, то рекомендуется чередовать операции умножения и деления, выбирая ту или иную в зависимости от абсолютной величины очередного промежуточного результата, которая может оказаться либо больше, либо меньше единицы.

Пренебрежение порядком счета в подобных случаях может привести к тому, что какой-нибудь промежуточный результат не сможет быть представленным на индикаторе с его двумя разрядами для чисел порядка. Если порядок превышает 99, то говорят о переполнении или машинной бесконечности; при этом калькулятор может остановиться, а может и продолжать счет — все зависит от многих конкретных обстоятельств. Если же порядок меньше минус 99, то говорят о машинном нуле. В этом случае калькулятор и впрямь в качестве результата операции записывает в регистр X нуль.

Например, найдем произведение четырех чисел $\Pi=2 \cdot 10^{-96} \times 0,0001 \times 3 \cdot 10^{95} \times 1,4 \cdot 10^{10}$, перемножая их на микрокалькуляторе «Электроника Б3-34» в различном порядке:

$$2 \cdot 10^{-96} \cdot 0,0001 \cdot 3 \cdot 10^{95} \cdot 1,4 \cdot 10^{10} = (0),$$

$$3 \cdot 10^{95} \cdot 1,4 \cdot 10^{10} (\text{ЕГГОГ}) \cdot 2 \cdot 10^{-96} \cdot 0,0001 = (840000),$$

$$3 \cdot 10^{95} \cdot 2 \cdot 10^{-96} \cdot 1,4 \cdot 10^{10} \cdot 0,0001 = (840000).$$

В первом случае перемножение чисел $2 \cdot 10^{-96} \times 0,0001$ переводит микрокалькулятор в область машинного нуля и окончательный результат оказывается равным нулю. Во втором случае перемножение чисел $3 \cdot 10^{95} \times 1,4 \cdot 10^{10}$ приводит к машинной бесконечности и при ручном счете на индикаторе появляется сигнал ошибки ЕГГОГ, однако при продолжении счета результат оказывается правильным. При вычислении по программе в этом случае остановки счета не происходит, и сигнал ошибки на индикаторе не высвечивается. В третьем случае сомножители расположены таким образом, что не возникает ни машинного нуля, ни машинной бесконечности.

Если в расчетах встречаются функции, вычисляемые

калькулятором, то следует помнить, что особенно большие относительные погрешности возникают при вычислении \sqrt{x} и x^y . Этих операций надо по возможности избегать, изменяя алгоритм вычислений. Надо также следить за тем, чтобы аргумент функции не попадал в «опасные зоны», где ее чувствительность наиболее высока. Замена алгоритма при этом может увеличить число выполняемых операций, но, несмотря на это, погрешность окончательного результата может снизиться.

Разумеется, расчетные формулы надо постараться преобразовать так, чтобы общее число операций было наименьшим. Общеизвестный пример такого рода связан с вычислением полинома, где наименьшее число операций обеспечивает схему Горнера.

Иногда при различных значениях аргумента для увеличения точности целесообразно применять различные алгоритмы. Например, для достаточно быстрого вычисления $n!$ часто используют формулу Стирлинга:

$$n! = \sqrt{2\pi n} e^{-n} n^n [1 + 1/(12n) + 1/(288n^2) - \dots].$$

Однако для значений $n \leq 10$ формула дает большую ошибку. Исправить положение можно, введя в программу счета округление результата до единиц, например, с помощью фрагмента 1 БП 7+FBx—. Теперь программа при использовании трех членов в формуле Стирлинга дает точное значение $n!$ для чисел от 1 до 9, а для больших значений n относительная ошибка не превосходит $5 \cdot 10^{-7}$. Время счета — всего около 20 секунд.

Так, подбирая удачный алгоритм, грамотно составляя программу вычислений, тщательно готовя исходные данные, контролируя накопление погрешностей, можно достичь желаемую точность результата.

7. Культура и искусство программирования

Какая программа лучше — короткая или быстрая, точная или наглядная?

Конечно, неверна сама постановка вопроса. И тем не менее нередко приходится видеть, как автор той или иной конкретной программы отстаивает достоинства своего детища именно с таких позиций.

В то же время ответ станет очевидным, если взять в расчет назначение программы. Программа-справочник для

астрономических расчетов требует прежде всего повышенной точности, учебная программа — прозрачности и сосредоточения на том или ином классе операций, программа генерации псевдослучайных чисел — скорости. Главная черта культуры программирования и состоит в умении подчинить все компоненты программы определенному целевому назначению.

Несложную программу для текущего расчета надо постараться написать попроще, не пользуясь изощренными приемами экономии памяти, не добиваясь особого сокращения времени вычислений. Если такая программа считает верно и написана без особых затрат времени и сил, значит, она отвечает целевому назначению, она функциональна. Можно даже считать, что в этой функциональности — ее красота.

Программа может получиться лучше или хуже, короче или длиннее в зависимости от того, насколько ее автору стали привычными известные приемы программирования на микрокалькуляторе данного типа. Это характеризует еще одну черту культуры программирования.

Культура и искусство программирования (эти понятия ведут начало от известной книги Д. Кнута, тема которой такое искусство) — не единственные факторы, которые надо принять во внимание для составления хорошей программы. В частности, надо подобрать алгоритм, соответствующий условиям задачи и свойствам используемого микрокалькулятора. Так что если говорить точнее, речь идет о культуре и искусстве решения вычислительных задач с помощью ЭВМ. Тем не менее, чтобы не нарушать традиций, мы не будем менять терминологию, понимая при этом «культуру» и «искусство» программирования в широком смысле.

Начинается культура программирования с выбора типа ЭВМ. Никому не придет в голову отправиться в командировку из Москвы в Томск на велосипеде, но и в Малаховку из Москвы самолетом не летают. И напрасно иногда считающих на микрокалькуляторах рассматривают как доминошников или, в лучшем случае, извозчиков в эпоху «Жигулей». Прошло время, когда на микрокалькуляторах можно было решать лишь самые простые задачи. Вряд ли прости интегрирование дифференциальных уравнений пятого порядка, проверка статистических гипотез или вычисление значений всех специальных функций из классического справочника Янке и Эмде. А ведь программы для таких расчетов давно со-



ставлены, опубликованы и уже никого не удивляют. Но это лишь одна сторона вопроса.

Привлекает внимание любопытный факт: программируемые микрокалькуляторы все шире применяются в организациях, хорошо обеспеченных большими ЭВМ. Одна из главных причин этого — предоставляемая микрокалькуляторами возможность решить поставленную задачу за кратчайшее время. Ситуацию можно проиллюстрировать диаграммой, по вертикальной оси которой отложено некоторое условное количество результатов. На горизонтальной оси — календарное время от момента, когда вы решили просчитать задачу по имеющейся программе.

До получения результатов на любой ЭВМ проходит некоторый промежуток времени. В микрокалькулятор надо ввести программу, проверить ее, ввести числовые параметры и выполнить вычисления. Как правило, на это уходит от десятка минут до нескольких часов. По-другому обстоит дело с большой ЭВМ. Здесь вы находитесь в определенной зависимости от намерений и занятости других людей. На ожидание очереди к терминалу или пробивку перфокарт, а также коррекцию исходных параметров по результатам расчета иногда уходит несколько дней. Начав считать, большая ЭВМ, конечно, сразу обгонит микрокалькулятор, но бывает, что задержка, связанная с подготовкой к счету, обесценивает результат. Это случается, например, при анализе вариантов новой конструкции.

Сказанное вовсе не означает, что во всех случаях надо стремиться использовать микрокалькулятор. Сложность задач, решаемых на любой ЭВМ и, конечно, на микрокалькуляторе, ограничивается вычислительными ресурсами машины. Если, например, не хватает памяти, то резко нарастает время программирования, приходится проявлять

чудеса изобретательности, чтобы сэкономить одну-две команды. Можно, конечно, комбинируя методы ручного и машинного счета, решать на «Электронике Б3-34» системы из семи-восьми линейных уравнений, но все же следует признать, что такая задача неадекватна нашей ЭВМ, и надо переходить к машине следующего класса, скажем, «Электронике Д3-28».

Рассмотрим теперь, как составляются программы, отвечающие сформулированным выше требованиям. О программах для текущих расчетов мы уже говорили. Со стандартными программами общематематического профиля можно познакомиться по опубликованным сборникам прикладных программ. Речь пойдет о программах третьего вида, специализированных, предназначенных для многократного применения.

У каждого считающего научного работника и инженера есть программы, к которым приходится обращаться достаточно часто, всякий раз, когда варьируются параметры того или иного исследуемого явления. Нередко в подобных случаях результаты вычислений используются исследователем как промежуточные данные при поиске путей решения основной научной задачи. Поэтому программа должна быть построена так, чтобы не тормозить творческого процесса и чтобы в роли оператора мог выступать как сам исследователь или его коллега, так и их помощник менее высокой квалификации.

Поставим задачу так, как ставятся задачи оптимизации в математике: найти минимум по одной переменной, наложив ограничения на остальные.

Считая заданными точность и вычислительные ресурсы микрокалькулятора, будем искать программу, обеспечивающую минимум времени получения результата. Это время — критерий качества составляемой нами программы. Оно определяется не только скоростью работы машины. Сюда входят затраты времени на ввод, вывод, контроль и пересчет в случае ошибки.

Рассмотрим внешние требования к программе — те, что существенны для пользователя.

Прежде всего программа должна быть оформлена так, чтобы не заставлять вас искать дополнительные материалы или размышлять над техническими, не важными в данном случае сторонами вопроса. Взяв программу после некоторого перерыва, когда вы успели кое-что подзабыть, вы должны иметь возможность сразу убедиться, что она вычисляет

то, что надо, и с достаточной точностью. Для этого в шапке программы должны быть указаны алгоритм, погрешность расчета и область допустимых значений аргументов.

Если для решения задачи надо составлять подпрограмму, должны быть сведения, откуда брать аргументы, куда засыпать результаты, какими регистрами можно пользоваться.

Какая запись программы лучше? Для публикации или «карманной» библиотеки важна компактность записи. Команды, бывает, пишут в строчку, без кодов операций, а если в каждой строке располагается ровно по десятку команд, не пишут и адреса. Однако для ввода такая система — не самая лучшая. В поле зрения попадают соседние команды, и ввод идет с определенным напряжением. Для контроля правильности ввода нужна еще таблица кодов операций: не все помнят коды, и здесь нетрудно ошибиться. Запись команд столбиком, с адресами и кодами операций, явно менее компактна, но ввод и контроль менее утомительны. Все это в конечном счете ведет к сохранению высокого темпа вычислений.

Большую роль играет однозначная бессылочная инструкция. К примеру, ссылка вида «далее — как в программе ТТ» заставит вас искать эту программу и разбираться еще и в ней.

Необходим контрольный пример, который не только позволит судить о правильности ввода программы, но и покажет, что вы верно поняли инструкцию. Пример должен быть простым: если вычисляется коэффициент корреляции, надо ограничиться двумя парами чисел. Конечно, на практике столь малую выборку не анализируют, но для проверки вполне достаточно двух пар.

Надо указать время счета, пусть и ориентировочное. Это поможет правильно спланировать работу.

Теперь рассмотрим внутренние требования к программе: то есть такие, которые существенны с точки зрения программиста.

Засылка констант в адресуемые регистры может выполняться в режимах как ручного, так и автоматического счета. Ручная засылка, когда на клавиатуре набирается число и выполняется команда ПМ (где М — обозначение адресуемого регистра), не удлиняет программы. Однако, если имеются резервы командной памяти, лучше организовать автоматический ввод констант с помощью безликих команд С/П, ПП или циклом. Ввод убыстрится, снизится вероят-

ность ошибки, меньше будет утомляться оператор: ему придется «пропускать через себя» только числа, а номера регистров его не касаются. Это же относится к засылке нулей в сумматоры перед каждым вариантом расчета. Скажем так: «все, что оператор может не делать, пусть он и не делает».

Особое внимание надо уделить программированию ввода больших массивов данных, например, при статистических расчетах. Эти данные после ввода сразу обрабатывают и в адресуемых регистрах накапливают соответствующие суммы или произведения. В зависимости от того, как составлена программа первичной обработки, затраты времени на ввод могут изменяться в полтора-два раза. Но дело даже не только во времени. Если очередные значения надо вводить через 5—10 секунд, оператор успевает чуть-чуть передохнуть, а если через 15—20 секунд, оператор невольно начинает отвлекаться. Вероятность ошибки растет.

Программа может помочь оператору, если его отвлекли и он спутал, которое число ему предстоит вводить. Для этого перед командой останова в программу ставят команду вызова номера числа, которое надо ввести. Возможны и другие варианты подсказки — это зависит от вашей фантазии. Важно только соблюсти чувство меры, так как для подсказки или, скажем, коррекции неправильного ввода требуются дополнительные команды, и это затягивает обработку.

Вывод результатов расчета в подобных случаях желательно организовать так же, как и ввод исходного материала: не заставлять оператора рыскать по регистрам, а поочередно выводить результаты на индикатор, нажимая клавиши С/П или ПП.

Теперь о компоновке программы. Ошибок и потерь времени будет меньше, если постараться разместить изменяющиеся части программы, в частности подпрограммы, вне границ основного поля адресов. Такое требование удлинит программу на две-три команды, но коллегам, работающим по вашей программе, не придется путаться при изменении адресов перехода. Темп работы не снизится. Коротко скажем так: «не допускай никого в тело программы».

Не нужно доказывать, что трудно составить хорошую, долго живущую программу, не владея описанными в руководстве по эксплуатации микрокалькулятора способами работы со стеком, способами организации циклов, косвенной адресацией. Обратим внимание на менее очевидные приемы. Это прежде всего тождественные преобразования

алгоритма, согласующие его с особенностями микрокалькулятора, затем группирование операций алгоритма и, наконец, выбор наилучшего порядка вычислений.

Пусть, например, требуется вычислить значение двойной суммы

$$S = \sum_{i=1}^{20} \sum_{j=1}^{10} i^2 e^{-2i} j \cos [0.5 + (i+j)\pi/6].$$

Программа, буквально соответствующая этой формуле, содержит 37 команд. Расчет выполняется примерно за 37 минут.

00.2 01.0 02.П0 03.Сх 04.1 05.0 06.П1 07.→ 08.ИП0 09.Fx²
10.ИП0 11.2 12.× 13./—/ 14.Fex 15.× 16.ИП1 17.× 18.ИП0.
19.ИП1 20.+ 21.Fx 22.× 23.6 24.: 25.0 26., 27.5 28.+ 29.F cos
30.× 31.+ 32.FL1 33.08 34. FL0 35.04 36. С/П.

Тождественное преобразование $i^2 e^{-2i} = (i/e^i)^2$ подсказывает, как обойтись пятью командами ИП0 ↑ Fex : Fx² вместо восьми команд с адресами 08—15. Для убывания счета организуем засылку констант 0,5 и π/6 в регистры Р3 и Р4 вне цикла, а внутри цикла по мере надобности будем их извлекать. При образовании этих констант для экономии командной памяти используем тождества $0,5 = 2^{-1}$ и $\pi/6 = \arcsin 0,5$.

Заметим, что вычислять целиком произведение под знаком двойной суммы 200 раз не обязательно. Можно вынести за скобки j , во внутреннем цикле вычислять сумму по i , а во внешнем — по j . Можно поступить и наоборот: за скобки вынести $(i/e^i)^2$, во внутреннем цикле суммировать по j , а во внешнем — по i . Нетрудно показать, что при достаточноном числе слагаемых полное время счета меньше там, где быстрее считается каждое слагаемое внутреннего цикла. Короче: «во внутренний цикл ставь то, что считается быстрее». В нашем случае суммирование во внутреннем цикле надо осуществлять по переменной j . Программа потребует также 37 команд, но время счета — около 19 минут, то есть почти вдвое меньше.

00.2 01.F1/x 02.П3 03.F arcsin 04.П4 05.2 06.0 07.П1 08.Сх
09.1 10.0 11.П0 12.→ 13.0 14.ИП0 15.ИП1 16.+ 17.ИП4 18.×
19.ИП3 20.+ 21.Fcos 22.ИП0 23.× 24.+ 25.FL0 26.14 27.
ИП1 28.↑ 29.Fex 30.: 31.Fx² 32.× 33.+ 34.FL1 35.09 36.С/П.

Перечислим некоторые тождественные преобразования, позволяющие приспособить исходный алгоритм к совокупности операций микрокалькулятора «Электроника Б3-34»

целью ускорения вычислений и экономии программной памяти.

1. $e^{2x} = (e^x)^2$. Пусть аргумент x находится в регистре M , Левая часть тождества требует четыре команды: ИПМ 2 $\times F e^x$, правая — три: ИПМ $F e^x F x^2$. Вычисления ускоряются на 0,4—0,5 секунды. Далее приводятся только тождества; эффект от их использования предлагается оценить читателю.

$$2. e^{x/2} = \sqrt{e^x}; \quad 3. 2 \ln x = \ln x^2; \quad 4. 0,5 \ln x = \ln \sqrt{x};$$

$$5. (1/x) e^{-x^2/2} = 1/\sqrt{x^2 e^{x^2}} \quad (x > 0);$$

$$6. \frac{2x-a}{2x+a} = \frac{x-a/2}{x+a/2};$$

$$7. \pi/4 = \operatorname{arctg} 1; \quad 8. \pi/2 = \operatorname{arccos} 0; \quad 9. \pi/2 = \operatorname{arcsin} 1.$$

Надо знать чисто конструктивные особенности микрокалькулятора «Электроника Б3-34», которые могут оказаться полезными при программировании. Вот некоторые из них:

1. Если стек возврата пуст, то есть выполнены все переходы с возвратом ПП N... В/0 (N — адрес перехода), то в режиме автоматического счета команда В/0 эквивалентна составной команде БП 01.

2. В режиме автоматического счета команда ВП, если ей предшествуют команды ИПМ или КИПМ, а также пара команд ↑ ВП, если ей предшествуют другие команды, переводит нуль в единицу, но не изменяет других чисел.

3. Если число $0,1 \leq x \leq 1$ находится в регистре M , команды КПМ и КИПМ при $M=0, 1, 2, 3$ уменьшают его, а при $M=4, 5, 6$ увеличивают на $1 \cdot 10^{-8}$. Если $x < 0,1$, эти команды изменяют его на другую величину. Читатель может проследить это самостоятельно.

4. Содержимое регистра M при неоднократном выполнении команды FLM уменьшается до единицы, но не далее. В то же время при выполнении команд КПМ и КИПМ ($M=0, 1, 2, 3$) содержимое регистра M , уменьшаясь, может стать равным нулю.

5. При выполнении команд КПМ и КИПМ ($M=0,1,2,3$) числа засыпаются (извлекаются) в регистры Д, С, В, А, ..., 4, 3, 2, 1, 0, 1, 0, Д, С, В, А, затем выделенная жирным шрифтом последовательность повторяется. Если же $M=4, 5, 6$, числа засыпаются (извлекаются) в регистры 0, 1, 2, ..., 9, А, В, С, Д, 0, 1, 0, 1, 2, 3, затем выделенная жирным шрифтом последовательность повторяется.

6. Расчет по ряду программ завершают после того, как некоторая переменная x уменьшится по абсолютной величине до заданного предела ε . Пусть ε находится в регистре M . Если $1 \cdot 10^{-99} \leq \varepsilon < 1 \cdot 10^{-50}$, для проверки на окончание можно использовать фрагмент программы A1.Fx<0 A2.A4 A3./—/ A4.ИПМ A5.— A6.Fx<0 A7.A9 A8.С/П.

Если $\varepsilon > 1 \cdot 10^{-50}$, используют фрагмент Fx^2 ИПМ—Fx<0 N С/П. В регистре M при этом хранят величину ε^2 , N — адрес перехода.

Заметим, что область машинного нуля на «Электронике Б3-34» при выполнении различных операций неодинакова. Например, пара операций $\operatorname{arccos}(\cos x)$ дает в результате нуль уже при значениях $x < 2 \cdot 10^{-4}$, тогда как пара операций $\ln(e^x)$ дает нуль только при $x < 4 \cdot 10^{-8}$. Пользуясь этим обстоятельством, фрагмент проверки на окончание счета при некоторых частных значениях ε можно сократить. Не понадобится и специальный регистр для хранения величины ε :

при $\varepsilon = 2 \cdot 10^{-4}$ Fcos Farccos Fx=0 N С/П;
» $\varepsilon = 4 \cdot 10^{-8}$ Fe^x Fln Fx=0 N С/П;
» $\varepsilon = 1 \cdot 10^{-8}$ Fl0^x Flg Fx=0 N С/П;
» $\varepsilon = 1 \cdot 10^{-25}$ Fx² Fx² Fx=0 N С/П;
» $\varepsilon = 1 \cdot 10^{-50}$ Fx² Fx=0 N С/П.

Здесь невозможно перечислить приемы, используемые на практике. Для совершенствования культуры программирования полезно с карандашом в руках разбирать опубликованные программы.

Мы смело можем считать, что овладели культурой программирования, если каждый этап составления и оформления программы мы умеем согласовать с ее целевым назначением и в работе используем богатый арсенал известных приемов программирования.

Очень хорошо, если для решения задачи хватает командной памяти. А если нет? В этом случае имеются три основных пути. Самый простой, но далеко не лучший — отказ от некоторых «достижений культуры». Очищать счетчики, засыпать константы и вводить исходные данные можно вручную прямо в регистры. Выводить результаты можно из регистров командами ИПМ. Останавливать расчет по достижении переменной величиной нуля можно аварийным остановом при помощи команды F1/x. Эти меры позволяют высвободить не так уж много, обычно до десятка команд. Но работать с такой программой более утомительно. К тому

же введение дополнительных ручных операций увеличивает вероятность ошибки.

Второй путь — выполнять одну или несколько неразветвленных частей программы в режиме ручного счета. Здесь экономия памяти может быть значительной, но этим способом имеет смысл пользоваться только при единичных расчетах.

Третий, самый плодотворный, но и трудный путь — замена алгоритма. Характерен призыв автора книги по прикладным программам В. Епанечникова: не ищите экономии в одну команду, экономьте сразу пятьдесят! Первый вариант его программы для решения системы из четырех линейных уравнений требовал 190 команд. Маленькими хитростями тут заведомо не обойдешься. А изменив алгоритм, ему удалось сократить программу до 91 команды. Здесь уже можно говорить об искусстве программирования. Под ним мы будем понимать разработку новых способов решения вычислительных задач на микрокалькуляторе не известными ранее методами.

Искусство эффективно там, где оно опирается на культуру программирования. Надо знать особенности машины, знать приемы программирования, разработанные другими авторами. Точность вычислений и диапазон допустимых значений параметров должны быть не больше, чем требуется в данной задаче. Тогда появляются резервы, позволяющие создавать простые приближенные формулы, такие, как формула для гамма-функции:

$$\Gamma(x) \approx \frac{1}{x} \sqrt{e^x (x-1)}, \quad 0 < x \leq 1, \quad \text{точность — один процент}$$

(В. Епанечников) или формула обратного интеграла вероятности:

$$\Phi(Q) \approx [-\ln(1-Q)]^{0.6} \quad (\text{А. Цветков}),$$

погрешность которой во всей области доверительных вероятностей $0.8 \leq Q \leq 0.999$ не превышает 4 процентов.

В одной из книг, посвященных программируемым калькуляторам, исходя из заданного объема числовой памяти «Электроники Б3-34», автор утверждает, что интерполировать по формуле Лагранжа на этом микрокалькуляторе можно максимум по пяти узлам. Однако Я. Трохименко и Ф. Любич, применив приближенную формулу и изменив способ ввода, разработали программу интерполяции по 10 узлам и при этом уложились в 42 команды вместо 97 у

категоричного автора. В который раз убеждаешься: там, где начинается искусство, формально верным, но незыбленным методам приходится отступать.

8. Сфера применения программируемых микрокалькуляторов

Каждому из нас приходится считать. Простейшие вычисления мы выполняем мысленно, но уже при арифметических действиях над многозначными числами обращаемся к авторучке. Если такие операции, особенно умножение и деление, приходится выполнять часто, то целесообразно купить простейший микрокалькулятор.

Приобретение программируемого микрокалькулятора обеспечит возможность успешно решать практически все домашние вычислительные задачи. Но самое главное, ради чего и ведется речь, — программируемый микрокалькулятор способен стать умелым и надежным помощником для работников самых различных специальностей.

Правда, у специалиста, которому доступны различные вычислительные средства, неизбежно возникает вопрос: какие задачи стоит решать на программируемом калькуляторе? Ответ связан со словом «стоит», прямо указывающим на стоимость применения калькулятора. Именно экономические критерии, хотя об этом мы часто забываем, определяют целесообразность создания и применения любых технических средств.

Все вычислительные задачи сводятся к последовательности арифметических операций и могут быть решены (по крайней мере в принципе) с помощью карандаша и бумаги. Однако по мере усложнения задач быстро возрастает количество необходимых операций. Соответственно увеличивается стоимость рабочего времени, затрачиваемого специалистом на получение результата. В этом случае использование даже простейших вычислительных средств, сокращающее затраты времени, дает значительный экономический эффект.

В то же время обращение к ЭВМ окупится лишь тогда, когда с ее помощью данную задачу можно решить дешевле, чем другими способами, например на вычислительной машине другого типа. По этой причине в каждом поколении ЭВМ имеются типы с различной производительностью, и еще в первом (ламповом) поколении были созданы простейшие

машины, названные калькуляторами и явившиеся дальними «предками» современных программируемых микрокалькуляторов.

Стоимость использования ЭВМ оценивают по времени ее работы, необходимому для решения задачи. Стоимость часа машинного времени несложно определить, поделив на расчетный срок службы машины все затраты на ее приобретение и эксплуатацию. Час машинного времени наиболее распространенных стационарных ЭВМ третьего (на интегральных компонентах) поколения стоит от нескольких десятков до нескольких сотен и даже тысяч рублей в зависимости от вычислительных возможностей машины. Стоимость часа машинного времени массовых программируемых микрокалькуляторов без внешней памяти, определяемая стоимостью покупки, ремонта и питания, например, за семилетний срок амортизации, составляет около 10 копеек.

Стационарные ЭВМ, машинное время которых недешево, окупаются лишь при практически непрерывном решении последовательности достаточно сложных задач. Для этого необходимо иметь запас (пакет) соответствующих программ, выполняемых по очереди. При таком пакетном режиме результат решения конкретной задачи, вычисляемый машиной за несколько минут или даже секунд, заказчик получит через несколько часов или даже дней. Следует также упомянуть, сколь труден прямой доступ заказчика к таким машинам, сколь значительны затраты времени обслуживания персонала на составление нестандартных программ, отсутствующих в машинной библиотеке, и затраты машинного времени на их отладку.

Время, в течение которого приходится ожидать результата решения, существенно уменьшается при использовании ЭВМ в режиме разделения времени между окончательными устройствами ввода и вывода информации (терминалами), к которым имеют прямой доступ пользователи машины. Стоимость таких машин большая, и они окупаются лишь при решении с терминалов достаточно сложных задач по оптимальным программам.

Таким образом, стационарные ЭВМ при любом режиме работы являются коллективными вычислительными средствами, предназначенными для решения достаточно сложных задач. Между тем каждый член коллектива специалистов встречается с множеством повседневных неотложных задач, решение которых в связи с их относительной простотой на стационарных и даже современных персональных

ЭВМ экономически невыгодно. Именно для решения таких вычислительных задач и предназначены программируемые микрокалькуляторы.

Их применение практически всегда выгодно, если для получения результата с их помощью требуется не более нескольких десятков минут. Наиболее экономичны здесь вычисления по заданным формулам с помощью одной или даже нескольких программ. Относительно быстро на калькуляторах типа «Электроника Б3-34» и аналогичных ему вычисляются многочлены от вещественного и комплексного аргумента с максимальной степенью соответственно 13 и 10. Достаточно удобно применение программируемого микрокалькулятора для решения большинства типовых математических задач, к последовательности которых сводятся практические расчеты: имеется в виду численное интегрирование, решение дифференциальных уравнений первого порядка (например, методом Рунге — Кutta, используемым и «большими» машинами) и т. п. Удобно и экономически выгодно применение программируемых микрокалькуляторов для аппроксимации с требуемой точностью различных специальных функций без обращений к таблицам и номограммам.

Решение нелинейных уравнений часто связано со значительными затратами машинного времени, но обычно и в этом случае рентабельно использовать программируемый микрокалькулятор. В качестве примера рассмотрим решение трансцендентного уравнения

$$e^{0,274x} - 2,831x = 0$$

с начальным приближением $x=0,4$ и требуемой точностью $\epsilon=10^{-3}$.

На машине МИР-2 корень этого уравнения $x=0,511577$ с тремя верными цифрами вычисляется по стандартной программе методом Ньютона с параметром за 60 секунд. На программируемом микрокалькуляторе типа «Электроника Б3-34» это же уравнение решается с той же точностью ($x=0,511$) методом простых итераций за 40 секунд. Следовательно, даже без учета затрат времени на ввод программы стоимость решения рассмотренной задачи на калькуляторе оказывается в несколько сотен раз (примерно в 450) меньше, чем на машине МИР-2 с ее относительно низкой стоимостью машинного времени.

В практических приложениях часто приходится решать системы уравнений. С помощью одной программы в автоматическом режиме на калькуляторе типа «Электроника Б3-34» можно решить систему, содержащую от двух до пяти линейных уравнений. При большем числе уравнений целесообразно использовать программу для преобразования системы по методу Жордана — Гаусса, не содержащему обратного хода. В этом случае для системы из n уравнений требуется вычислить и записать $n^2(n+1)/2$ чисел, из которых n последних являются искомыми корнями. При этом на получение очередного числа затрачивается менее 20 секунд. Для решения, например, системы из шести линейных уравнений в этом случае при отсутствии ошибок в считывании и записи потребуется около 50 минут.

Для получения тех же результатов на машине МИР-2 по стандартной программе требуется только 21 минута, но с учетом стоимости машинного времени использование программируемого микрокалькулятора и тут оказывается экономически значительно более выгодным даже при учете рабочего времени пользователя.

В подобных сравнениях могли бы выступать и большие ЭВМ других типов, но и без дальнейших пояснений очевидно: на программируемом микрокалькуляторе стоит решать все задачи, которые нельзя решить дешевле другими способами. Опыт ряда инженерных коллективов свидетельствует о том, что в среднем свыше 90 процентов задач инженерного расчета целесообразно решать на программируемых микрокалькуляторах без обращения к более дорогим ЭВМ. Опыт свидетельствует также и о том, что близорукое «престижное» увлечение высокопроизводительными ЭВМ, аналогичное «стрельбе из пушек по воробьям», без использования такого «личного оружия» специалиста, каким являются калькуляторы, приводит к значительным явным или, еще чаще, скрытым экономическим потерям.

Естественно, эффективное использование программируемых микрокалькуляторов возможно лишь в том случае, когда их пользователи успешно освоили программирование, методы вычислительной математики и располагают достаточно полной библиотекой оптимальных программ для решения типовых задач. Многие пользователи добиваются этого самостоятельно, но гораздо больший эффект дает обучение. Затраты на него, как показывает опыт ряда технических вузов, полностью окупаются. Студенты-первокурсники, прошедшие такое обучение в объеме 30—40 лекционных ча-

сов и 20—30 часов практических занятий под руководством опытных преподавателей, в дальнейшем успешно используют программируемые микрокалькуляторы для решения достаточно сложных задач не только в стенах вуза, но и после его окончания. При этом существенно упрощается последующее изучение алгоритмических языков ЭВМ, а привычка эффективно использовать ограниченные ресурсы памяти калькулятора оказывается весьма полезной при программировании сложных задач на ЭВМ высокой производительности.

Следует подчеркнуть, что по экономическим показателям и портативности программируемые микрокалькуляторы (несмотря на расширяющееся внедрение персональных ЭВМ) еще долго будут вне конкуренции со стороны ЭВМ других классов при решении относительно простых задач, не требующих вывода информации в графической форме. Это обстоятельство особенно важно учитывать при постановке изучения основ информатики в средней школе, требующей в масштабе всей страны колоссальных затрат на вычислительную технику и ее эксплуатацию.

* * *

В проведении занятий «Школы программирования» участвовали Г. В. Славин (1), И. К. Вязовский (2, 3), М. Ф. Поснова, Н. Н. Поснов, Г. Н. Бакунин, В. Л. Леонтьев (3), С. В. Комисаров (4), Л. Ф. Штернберг (5), М. И. Петров (6), А. Н. Цветков (7), Я. К. Трохименко (8; в скобках указаны номера занятий).

В первой части книги были описаны язык и основы программирования на микрокалькуляторах. Настало время посмотреть, как применять изложенные там сведения на практике.

В этой части разбираются прикладные программы из разных областей науки и техники, предложенные читателями журнала «Наука и жизнь». Здесь они помещены в чисто учебных целях. Несмотря на их узкую профессиональную направленность, подробный их разбор может оказаться полезным каждому начинающему программисту. Почти все программы снабжены подробными комментариями, где освещаются их наиболее интересные и тонкие места.

Не все программы даны в том виде, как их прислали авторы. Часть из них составители посчитали необходимым отредактировать, чтобы сделать удобнее и эффективнее. Отбор программ диктовался желанием дать наиболее широкое представление о спектре возможных применений микрокалькуляторов. Вы найдете здесь и простейшие программы (такие, как расчет калькуляции), и довольно сложные (например, распределение добавок при плавке стали). Приведены программы для решения нечисловых задач и даже для использования калькуляторов в качестве часов, таймера и секундомера.

Два типа задач было решено не затрагивать в этой книге. Это, во-первых, программы для решения часто встречающихся математических проблем (алгебраические и дифференциальные уравнения, численное интегрирование, интерполяция и т. д.). Такого рода программы опубликованы в очень хороших сборниках А. Н. Цветкова и В. А. Епанечникова «Прикладные программы для микро-ЭВМ «Электроника Б3-34», «МК-56», «МК-54» (М., Финансы и статистика, 1984) и Я. К. Трохименко и Ф. Д. Любича «Инженерные расчеты на программируемых микрокалькуляторах» (Киев, Техника, 1985). Во-вторых, читатель не найдет в книге игровых программ. Причина та же: им посвящена специальная книга «Микрокалькуляторы в играх и задачах» (М., Наука, 1986).

На примерах программ для поиска простых чисел и для статистических расчетов подробно описаны методика состав-

ления алгоритмов и оптимизация программ. Большая же часть программ призвана проиллюстрировать типичный круг микрокалькуляторных проблем: расчеты по формулам, решение несложных уравнений и т. п.

Как уже было упомянуто, круг затронутых в книге вопросов определялся исключительно читательской почтой журнала «Наука и жизнь». Вы найдете здесь программы, написанные металлургами и сварщиками, топографами и медиками, технологами и плановиками. Конечно, нельзя объять необъятное. Но отсутствие программ химических и биологических, геологических или кулинарных (а почему бы и нет?) объясняется не недостатком места. Просто специалисты из этих областей, очевидно, не нашли применения микрокалькулятору для решения своих профессиональных проблем. Можно надеяться, однако, что после знакомства с этой книгой представители перечисленных (да и других) профессий без труда найдут в своей практике «калькуляторные» задачи.

Калькуляция на калькуляторе

Иногда приходится сталкиваться с мнением, что программируемый калькулятор стоит использовать лишь для решения сравнительно сложных задач. Что же касается вычислений, где всего несколько сложений да умножений, то здесь и арифмометра хватит. В крайнем случае калькулятора простейшего, непрограммируемого. А если эти сложения да умножения придется делать десятки раз на дню? Беспрерывно нажимать на клавиши калькулятора — занятие утомительное. Хоть расчетчик в этом случае избавится от стука костяшек на счетах и лязга арифмометра, усталость возьмет свое. Постоянное напряжение, необходимость запоминать, какие действия и в какой последовательности производить, неминуемо приведет к нарастанию ошибок.

Не лучше ли поручить эту работу программируемому калькулятору и запускать каждый очередной расчет одним нажатием на клавишу?

Именно так и поступил инженер-технолог завода по ремонту вычислительной техники г. Дружковка Донецкой области С. Чучанов. Он разработал простую программу для расчетов калькуляции. Она приспособлена к нуждам конкретного завода. Однако по такому же принципу можно

составить программу для любого другого предприятия, изменив должным образом процентные содержания одних статей по отношению к другим. Несмотря на простоту программы, польза от ее внедрения достаточно велика, ведь она автоматизирует расчеты, чаще всего производящиеся вручную.

Исходными данными для расчета калькуляции является стоимость материалов (M) и величина основной заработной платы (Z_o), выраженные в рублях. Остальные величины определяются из простых формул:

дополнительная заработка плата:

$$Z_d = Z_o \cdot 10,8\%;$$

отчисление на соцстрахование:

$$O_c = (Z_o + Z_d) \cdot 7\%;$$

общезаводские расходы:

$$P_o = Z_o \cdot 57,4\%;$$

производственная себестоимость:

$$C = M + Z_o + O_c + P_o + Z_d;$$

прибыль:

$$\Pi = C \cdot 20\%$$

и оптовая цена:

$$O_u = C + \Pi.$$

Программа. 00.0 01., 02.0 03.7 04.ИП2 05.0 06., 07.1 08.0
09.8 10.× 11. С/П 12.ИП2 13.+ 14× 15.С/П 16.FBx 17.+
18.0 19., 20.5 21.7 22.4 23. ИП2 24.× 25.С/П 26.+ 27.ИП1
28.+ 29.↑ 30.С/П 31.5 32.: 33. С/П 34.+ 35.С/П.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F ABT).
3. Ввести M П1 Z_o П2.
4. Очистить счетчик команд (B/0).
5. Запустить программу (С/П).
6. Вывод: Z_d С/П O_c С/П P_o С/П C С/П Π С/П O_u .
7. Продолжение работы:
 - 7.1. Повторение с теми же данными — перейти к п. 4.
 - 7.2. Расчет с новыми исходными данными — перейти к п. 3.

8. Контрольный пример: $M=10$; $Z_o=20$.

Результаты: $Z_d=2.16$; $O_c=1.55$; $P_o=11.48$; $C=45.19$;
 $\Pi=9.04$; $O_u=54.23$.

Комментарий.

1. Так как программа занимает мало места в памяти, автор решил не записывать многозначные константы в память, а формировать их по ходу выполнения программы

(фрагменты 00—03, 05—09, 18—22). Благодаря такой записи уменьшаются затраты ручного труда при работе с программой и суммарное количество нажимаемых клавиш для подготовки к работе (не надо нажимать клавиши ПН, где N — имя регистра), да и вероятность ошибок также уменьшается.

2. Обратите внимание, что для экономии длины программы умножение на 20%, то есть на 0,2, заменено делением на 5 (адреса 31—32). Такие моменты полезно иметь в виду при записи констант.

У токарного станка

Эта программа почти такая же простая, как и предыдущая. Разве что формул побольше. Но если первая поступила из планового отдела, то эта пришла «от станка».

При обучении токарным премудростям учащимся часто приходится рассчитывать режим резания. Для этого используется несколько формул. Вот они:

$$v_d = k_o \cdot k_1 \cdot k_2 \cdot k_3 \cdot k_4 \cdot k_5 \cdot v_{раб};$$

$$n_{шп} = v_d \cdot 10^3 / \pi D;$$

$$v_\phi = n_{шп} \pi D / 10^3;$$

$$P_z = KTS_{cr};$$

$$M_{рез} = P_z D / 2 \cdot 10^3;$$

$$T_o = L_i' n_{cr} S_{cr}.$$

Величины, стоящие в левых частях равенств, — расчетные параметры: v_d — действительная скорость резания, м/мин; $n_{шп}$ — частота вращения шпинделя, об/мин; v_ϕ — фактическая скорость резания, м/мин; P_z — сила резания, кгс; $M_{рез}$ — момент резания, кгс·м, и T_o — основное (машинное) время, мин.

В правых частях формул стоят параметры частично постоянные, определяемые типом станка, частично изменяемые. Варьируя их, токарь подбирает оптимальный режим обработки детали.

Обычно подобные расчеты производились вручную. Расчет одного варианта при этом отнимал не менее десятка минут, да и вероятность ошибки была довольно велика.

Преподаватель СПТУ М. Бажин и инженер А. Скрипов из Перми написали соответствующую программу, расчеты по которой занимают всего пару десятков секунд, и вероятность ошибок сведена к минимуму, поскольку они могут произойти только из-за неправильности вводимых данных.

Данных этих довольно много — четырнадцать. Причем при подборе режима обработки некоторые из них меняются, остальные, те, что задаются типом станка, остаются неизменными. Поэтому наиболее целесообразно ввести все исходные параметры напрямую в адресуемые регистры и затем при новых расчетах часть их занести повторно. Это накладывает на программу определенные требования — не использовать адресуемые регистры ни для хранения промежуточных величин, ни для размещения результатов.

Распределяются регистры следующим образом:

P0— k_0 , P1— k_1 , P2— k_2 , P3— k_3 , P4— k_4 , P5— k_5 (поправочные коэффициенты на скорость резания); P6— K — коэффициент резания, кгс/мм; P7— D — диаметр обрабатываемой детали, мм; P8 — t — глубина резания, мм; P9— S_{ct} — подача (паспортная), мм/об; PA — v_{tab} — скорость резания табличная, м/мин; PB — L — расчетная длина обработки с учетом врезания и перебега, мм; PC — i — число рабочих ходов; PD — n_{ct} — частота вращения (паспортная), об/мин.

Программа: 00.ИП0 01.ИП1 02.× 03.ИП2 04.× 05.ИП3 06.× 07.ИП4 08.× 09.ИП5 10.× 11.ИПА 12.× 13.↑ 14.ИП7 15. Fπ 16.× 17.3 18.F10× 19.: 20.: 21. FBx 22. ИПД 23.× 24.С/П 25.ИП6 26.ИП8 27.× 28.ИП9 29.× 30.↑ 31.ИП7 32.× 33.2 34. ВП 35.3 36.: 37.ИПВ 38.ИПС 39.× 40.ИПД 41.: 42.ИП9 43.: 44.С/П.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F ABT).
3. Очистить командный счетчик (B/0).
4. Ввести исходные данные в регистры (распределение регистров дано выше).
5. Запустить программу (С/П).
6. Вывод: $v_{\phi} F \bigcirc n_{\text{шп}} F \bigcirc v_d \text{ С/П } T_o F \bigcirc M_{\text{рез}} F \bigcirc P_z$.
7. Продолжение работы: перейти к п. 3.
8. Контрольный пример.

$k_0=1.2$, $k_1=1$, $k_2=1.15$, $k_3=1$, $k_4=0.65$, $k_5=1$, $K=178$, $D=85$, $t=3$, $S_{ct}=1$, $v_{tab}=127$, $L=46.5$, $i=1$, $n_{ct}=315$.
Результаты: $v_{\phi}=84.12$, $n_{\text{шп}}=426.6$, $v_d=113.92$, $T_o=-0.15$, $M_{\text{рез}}=22.7$, $P_z=534$.

Комментарий.

1. Выходные данные накапливаются в стеке и затем выводятся порциями (по 3 числа). При такой организации вывода экономится как длина программы, так и время ее работы.

2. По адресам 13 и 30 стоит команда ↑. Она введена для того, чтобы создать в стеке копии некоторой величины (в первом случае v_d , во втором — P_z). Копии нужны, так как упомянутые величины, с одной стороны, участвуют в дальнейших вычислениях, с другой — должны сохраняться в стеке для вывода. Командой ↑ в программах следует пользоваться с осторожностью. Копия сохраняется только в том случае, если после команды ↑ стоит тоже команда, скажем, ИП1, П1, Fπ и т. д. Если же в следующей ячейке записано число, то копия затирается. Например, пусть в Р1 записано число 1 и во всех регистрах стека нули. Тогда после выполнения фрагмента ИП1↑Fπ содержимое стека будет таково: 3.14 1 1 0. Если же выполнить фрагмент ИП1↑2, то в стеке будут находиться числа 2 1 0 0, то есть во втором случае копия величины 1 была затерта следующим числом, записанным в программе.

Эту особенность команды ↑ следует постоянно помнить при программировании и правильно записывать последовательность команд (в программировании от перемены мест слагаемых сумма меняется).

Проверка с микрокалькулятором

Тему этой программы предложил учитель из г. Красноярска Г. Давыдович. Он преподает в учебно-производственном комбинате курс «Контрольно-измерительные приборы и автоматика». Тема одного из практических занятий по этому курсу — поверка измерительных приборов. По ходу занятий школьники должны контролировать правильность показаний термометра сопротивления. Прибор этот представляет собой катушку из тонкой платиновой или медной проволоки. При изменении температуры сопротивление проволоки меняется. Изменения фиксируются так называемым вторичным прибором, который преобразует величину сопротивления в значение температуры.

Если сам термометр работает весьма надежно, то вторичные приборы, более сложные по конструкции, могут иногда давать ошибки. Поэтому их периодически поверяют. Подключают на вход вместо термометра магазин сопротивлений и смотрят, на сколько отклоняются его показания от тех, которые соответствуют эталонным сопротивлениям.

Непосредственные измерения занимают при этом не более получаса. Вычисления, связанные с их обработкой,

отнимают в несколько раз большие времени. По ходу их приходится неоднократно обращаться к специальным таблицам. Потому-то и возникла мысль использовать программируемый микрокалькулятор, поручив ему функции как хранилища таблиц, так и вычислителя.

Алгоритм вычисления табличных величин — сопротивлений при разных температурах — известен. Для платины зависимость эта определяется формулой

$$R = \begin{cases} R_0(1 + At + Bt^2) & \text{при } 0^\circ\text{C} \leq t \leq 650^\circ\text{C}; \\ R_0[1 + At + Bt^2 + Ct^3(t - 100)] & \text{при } -200^\circ\text{C} \leq t < 0^\circ\text{C}. \end{cases}$$

Величина R_0 определяет градуировку прибора. Она может быть равна 10,46, или 100 Ом. Значения величин A , B и C также известны: $A = 3.96847 \cdot 10^{-3} \text{ } 1/\text{C}$, $B = -5.847 \cdot 10^{-7} \text{ } 1/\text{C}^2$, $C = -4.22 \cdot 10^{-12} \text{ } 1/\text{C}^4$. Для меди аналогичная формула значительно проще и программирование ее, естественно, легче.

Так как в лабораторных экспериментах температура выше 650°C и ниже минус 200°C не встречается, то можно считать, что верхняя из приведенных формул «работает» для всех положительных температур, а нижняя — для всех отрицательных.

Прежде чем писать программу, давайте внимательно посмотрим на обе формулы. Сразу бросается в глаза их сходство. Ясно, что дважды писать одинаковые последовательности команд для их вычисления нецелесообразно.

Далее, если ввести во второй формуле вспомогательный коэффициент $B' = B + Ct$ ($t - 100$), то она приобретет вид, совершенно аналогичный первой формуле.

Так как при решении этой задачи в отличие от ранее рассмотренных вычисления могут протекать по различным путям, то есть вычислительный алгоритм имеет ветвистую структуру, целесообразно перед началом программирования составить блок-схему алгоритма. Она приведена на рисунке. В блок-схеме, кроме уже употребленных обозначений, использованы еще следующие: $t_{\text{ нач}}$ — начальная температура, для которой проверяется прибор, ΔT — шаг шкалы прибора, ΔR — разность сопротивлений между начальной и конечной точками шкалы (табличная величина), ϵ_{abc} — абсолютная погрешность, $\epsilon_{\text{отн}}$ — относительная погрешность в процентах.

Вы, очевидно, заметили, что в блок-схеме отсутствуют блоки ввода величин A , B , C , R_0 , ΔR , $t_{\text{ нач}}$ и ΔT . Величины эти постоянны на всем протяжении работы, поэтому



их целесообразно «загрузить» сразу, перед началом расчетов, в соответствующие адресуемые регистры и больше о них не вспоминать. А кстати, в какие? Одной стороны, вроде бы это безразлично. Нужно только знать, в каком регистре находится какая величина, чтобы при необходимости ее считать. Но с другой... При программировании на языках высокого уровня, где используются не команды, а операторы, принято вводить в программах обозначения, соответствующие обозначениям в формулах. Так и проверить и отладить программу будет легче. В микрокалькуляторах такой возможности нет. И все-таки если есть у нас регистры A , B и C , то почему бы именно в них не загрузить величины, обозначаемые теми же символами? Это также облегчит и ввод, и проверку программы.

Поэтому прежде чем приступить к программированию, заранее распределим величины по регистрам:

- A — РА;
- B — РВ;
- C — РС;
- R_0 — Р0.

На этом, к сожалению, соответствия заканчиваются, поэтому остальные величины распределяем «волевым» порядком:

- $t_{\text{ нач}}$ — РД;
- 100 — Р1;
- ΔT — Р2;
- ΔR — Р3.

Засылка величины 100 в регистр Р1, вообще говоря, — ведь не обязательная. Эту константу легко записать в самой программной памяти, используя последовательность команд 1 ВП 2, или еще короче 2 F10^x. И не будь других загружаемых величин, мы бы так и поступили. Но так как, с одной стороны, величина эта должна использоваться в нашей программе несколько раз, минимум дважды (посмотрите блок-схему), а с другой — мы все равно идем на «адресный ввод», то предварительная засылка этой величины дает очень небольшой относительный приварок в общее время ввода, а считывание ее из регистра и быстрее, и команда потребует меньше, чем формирование ее в программе.

Программа: 00.ИПД 01.Fx \geqslant 0 02.07 03 \leftarrow 04.ИПВ 05.БП 06.15 07. \uparrow 08.ИП1 09.— 10. \times 11. ИПС 12. \times 13. ИПВ 14.+ 15. ИПД 16. \times 17. ИПА 18.+ 19.ИПД 20. \times 21.1 22.+ 23.ИПО 24. \times 25.П4 26.— 27.Fx² 28. FV \downarrow 29. \uparrow 30.ИПЗ 31.: 32.ИП1 33. \times 34.ИПД 35.ИП2 36.+ 37.ПД 38.FO 39. ИП4 40.С/П 41.БП 42.00.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Ввести постоянные величины: А ПА В ПВ С ПС $t_{\text{нач}}$.
- ПД R_0 ПО 100 П1 ΔT П2 ΔR П3.
4. Очистить счетчик команд (В/0).
5. Ввод: $R_{\text{изм}}$ С/П.
6. Вывод: $R_{\text{таб}}$ FO $e_{\text{отн}}$ FO $e_{\text{абс}}$.
7. Продолжение работы. Перейти к п. 5.

Комментарий.

1. Обратим внимание на вычисление абсолютной величины (команды 27—28). Так как в языке микрокалькулятора «Электроника Б3-34» специальной команды для этой цели нет, то абсолютная величина вычисляется с помощью двух операций: сначала величина эта возводится в квадрат, а затем из результата извлекается квадратный корень.

2. Мы не используем специального регистра для хранения измеренного сопротивления. Будучи введенной в стек перед началом работы программы, эта величина кочует буквально по всем четырем регистрам и в нужный момент, перед выполнением команды 26 оказывается как раз в регистре Y стека, откуда и берется для проведения операции вычитания.

На сварке труб

Покинем теперь лабораторию контрольно-измерительных приборов и вновь перейдем на производственный участок. На сей раз мы будем присутствовать при сварке труб.

Иногда приходится врезать трубы в трубопроводы под заданным углом. При этом нужно предварительно размечать трубы по специально приготовленным шаблонам. Вычерчивание шаблонов — дело трудоемкое, сопряженное с вычислением по громоздким формулам.

А. Таланкин из г. Абакана решил переложить этот труд на программируемый калькулятор.

Расчет шаблонов состоит в вычислении координат точек огибающих их кривых по формулам:

$$A = \frac{1 - \cos \frac{l}{r}}{\sin \alpha} (r - b) - \frac{1 - \cos \left[\arcsin \left(\frac{r-b}{R} \sin \frac{l}{r} \right) \right]}{\tan \alpha} R;$$

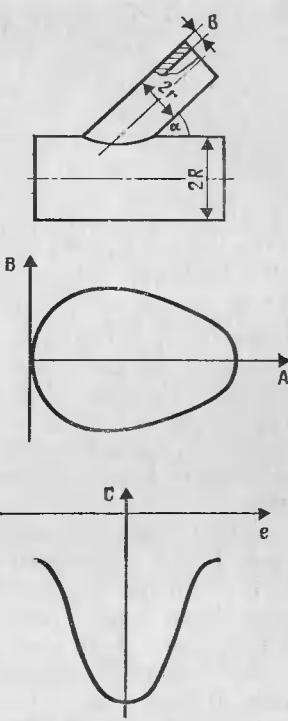
$$B = R \arcsin \left(\frac{r-b}{R} \sin \frac{l}{r} \right);$$

$$C = \frac{1 - \cos \frac{l}{r}}{\tan \alpha} (r - b) - \frac{1 - \cos \left[\arcsin \left(\frac{r-b}{R} \sin \frac{l}{r} \right) \right]}{\sin \alpha} R.$$

Обозначения, используемые в формулах, понятны из рисунка, где изображены эскизы шаблонов. Входящая в формулы величина l означает текущую длину и изменяется от 0 до πr .

Алгоритм решения задачи состоит в последовательных вычислениях по приведенным формулам, поэтому в блок-схеме не нуждается.

Основная проблема, возникшая при программировании, связана с измерением углов. Дело в том, что переменный



угол α/r , используемый в программе, должен измеряться в радианах. Угол же между свариваемыми трубами α , естественно, привычнее задавать в градусах. Так как режим работы с тригонометрическими функциями (градусы или радианы) обычно задается перед началом работы, то сочетание обоих мер в одной программе крайне нежелательно.

Разрешить противоречие можно несколькими способами.

Первый, самый простой вариант: заставить пользователя пересчитывать градусы в радианы вручную и задавать калькулятору значения α в радианах. Второй более остружен: составить программу так, чтобы вычисления с градусами были в одной ее части, а с радианами — в другой. Тогда, разделив эти части командой останова, можно первую из них выполнять при положении переключателя Р—Г «градусы», а вторую — «радианы». Наконец, третий способ: поручить калькулятору переводить градусы в радианы программным путем и вести все расчеты в режиме «радианы».

Первый способ, пожалуй, наихудший. Хоть программа получается и простая, зато возрастает доля ручного труда при счете по программе, что нежелательно. Ведь автоматизация счета основной своей целью имеет как раз уменьшение этой доли. Рекомендовать этот подход можно лишь в крайних случаях, когда никакие другие варианты невозможны.

Второй способ никаких предварительных ручных операций не требует. Программа, составленная на его основе, как правило, не длиннее сделанной по первому способу. Но... Необходимость постоянно следить за переключением Р—Г хотя и не требует машинной памяти, зато засоряет память оператора. Ведь эту информацию надо держать в голове. Забыли переключить — вся работа наスマрку. Этот способ советуем использовать в тех случаях, когда длина программы велика и в памяти нет места для записи нескольких лишних команд.

Наконец, третий способ — программный перевод градусов в радианы (или наоборот). При этом варианте программа удлиняется на пяток команд, но память пользователя освобождается и работать с программой становится легче и удобнее. Именно этот способ и выбран в предлагаемой программе.

Также из соображений удобства константа 180 записана непосредственно в программной памяти, а не в адресуемом регистре, куда ее нужно было бы вводить вручную.

Еще один момент, на который хочется обратить внимание. Для построения графиков нужно вычислять довольно много точек. Удобнее всего задавать определенный шаг изменения координаты Δl и организовать цикл для получения выходных данных. Однако почти наверняка выбранный шаг не уложится целое число раз на отрезке πr . Поэтому в программе предусмотрено сравнение текущей координаты с величиной πr . Как только координата превысит эту величину, значение l принимается равным πr , и именно при этой величине проводится последний расчет.

Программа.

00.П1 01.F	02.П2 03.≥ 04.— 05.П3 06.ИП1
07.: 08.П4 09.≥ 10.1 11.8 12.0 13.: 14.Фл 15.× 16.Фsin 17.П5	18.ФВх 19.Фtg 20.П6 21. ИП2 22.Фл 23.× 24.П0 25.С/П
26.П8 27.ПД 28.ИПД 29.ИП2 30.: 31.Фcos 32.ФВх 33.Фsin	34.ИП4 35.× 36.Фarcsin 37.ПВ 38.Фcos 39.1 40.— 41.ИП1
42.× 43.ПС 44.ИП6 45.: 46.≥ 47.1 48.— 49. ИП3 50.×	51.П7 52.ИП5 53.: 54.— 55.ПА 56. ИПВ 57.ИП1 58.×
59.ИПС 60. ИП5 61.: 62.ИП7 63.ИП6 64.: 65.— 66.≥	67.ИПА 68.ИПД 69.С/П 70.ИП0 71.ИП8 72.ИПД 73.+
74.ПД 75.— 76. Fx<0 77.28 78.ФV— 79.КНОП 80.ИП0	81.БП 82. 27.

Инструкция.

1. Ввести программу. Установить переключатель Р—Г в положение Р.

2. Перейти в режим вычислений (F АВТ).

3. Очистить командный счетчик (В/0).

4. Ввод: $\alpha \uparrow b \uparrow r \uparrow R$ С/П (после останова на индикаторе — величина πr) Δl С/П.

5. Вывод: l F \downarrow A F \downarrow B F \downarrow C.

6. Продолжение работы: С/П. При появлении на индикаторе ЕГТОГ: С/П, вывод последней точки ($l=\pi r$).

7. Проведение новых расчетов: перейти к п. 3.

Примечание. Шаг изменения координаты Δl можно менять вручную при любом останове программы. Для этого нужно заслать новое значение Δl в Р8, то есть выполнить команды: Δl П8.

8. Контрольный пример: $\alpha=30^\circ$, $b=3,5$, $r=38$, $R=57$. $\Delta l=20$. Результаты — в таблице 3.

Комментарий.

1. Программа логически делится на три блока:

а) команды 00—25. Обработка величин, не зависящих от текущей координаты, то есть постоянных для всего процесса счета;

Таблица 3

<i>l</i>	20	40	60	80	100	119,4
<i>A</i>	4,7	20,1	49,4	89,7	124,8	138,0
<i>B</i>	17,6	31,6	37,1	31,2	17,1	0
<i>C</i>	2,7	13,1	37,0	73,5	106,8	119,5

б) команды 26—69. Проведение расчетов по формулам и накопление итоговых данных в стеке.

в) команды 70—83. Вычисление новой величины текущей координаты в соответствии с заданным шагом, сравнение ее с максимально допустимой величиной πr и передача управления на начало расчета либо с координатой *l*, если она допустима, либо с координатой πr в противном случае. При этом появление предупреждающего сообщения ЕГГОГ означает: «Внимание, последняя точка расчета».

2. Команда 78 использована для формирования сигнала ЕГГОГ. Как только разность $\pi r - l$ становится отрицательной, попытка извлечь квадратный корень из этой величины приводит к аварийному останову и появлению на индикаторе сообщения ЕГГОГ. Если потом нажать клавишу С/П, то программа продолжит работу, пропустив, правда, следующую за «ошибочной» команду. Для этого и поставлена по адресу 79 пустая команда КНОП.

Если нужно построить график

В том что графики приходится строить часто, убеждать никого не надо. Наглядный пример — материал предыдущей главы. Ведь шаблон — тоже график определенной функции. Хорошо, когда для этого есть специальные графопостроители. А если их нет, то строят графики вручную на миллиметровке.

Нередко при заданных размерах графика масштабы по осям оказываются неудобными. Требуется, скажем, нанести на координатную ось некоторую величину. Сколько для этого надо отложить миллиметров? Расчеты, поиск нужной точки занимают значительное время и к тому же

ведут к накоплению ошибок. Это особенно проявляется при необходимости строить график в том же темпе, в котором снимаются данные, как говорят, в реальном масштабе времени. Сложность еще более возрастает при использовании нелинейных масштабов.

П. Полянский из Москвы разработал программы, автоматизирующие этот процесс и позволяющие для любого масштаба определить целое число миллиметров, которые нужно откладывать по той или иной оси. Первая из этих программ предназначена для получения графиков в линейно-линейном, вторая — в линейно-логарифмическом масштабе.

Программа 1. 00.ИП2 01.× 02.БП 03.06 04.ИП1 05.× 06.7 07.Ф10^x 08.+ 09.ФВх 10.— 11.С/П 12.БП 13.04.

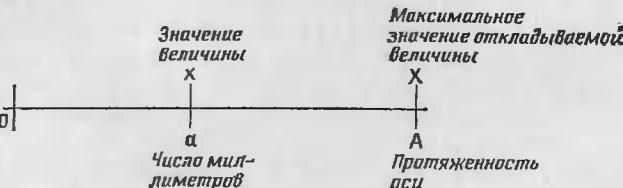
Программа 2. 00.ИП2 01.× 02.БП 03.09 04.Ф^{lg} 05.ИП4 06.— 07.ИП3 08.× 09.7 10. Ф10^x 11.+ 12.ФВх 13.— 14.С/П 15. БП 16.04.

Так как программы однотипны, то ограничимся одной общей инструкцией для работы с ними.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Ввод констант:
 - 3.1. Для первой программы: A/X^{**} П1 B/Y П2.
 - 3.2. Для второй программы: B/Y П2 lgX_0 П4.
4. Набрать БП 04.
5. Ввод координат.
 - 5.1. Абсциссы: x С/П.
 - 5.2. Ординаты: y В/0 С/П.
6. Результат — количество миллиметров — на индикаторе.
7. Продолжение работы: перейти к п. 5.
8. Переход на построение нового графика: перейти к п. 3.

** Смысл обозначений понятен из рисунка.



Комментарий.

1. В этих программах число ручных операций, необходимых для запуска программы, пожалуй, больше, чем во всех остальных, вместе взятых. В чем же дело? Ведь программы-то короткие, места в памяти хоть отбавляй. Тогда почему же не предусмотрен более удобный ввод начальных параметров и избавление пользователя от лишней ручной работы?

Конечно, это можно было бы сделать, но при этом либо программа увеличилась бы в несколько раз, либо пользователю пришлось бы производить дополнительные манипуляции при вводе координат — вводить признак оси или команду передачи управления. Поэтому автор программы решил, что лучше поручить пользователю предварительную работу по заполнению адресуемых регистров и минимизировать число нажимаемых клавиш при работе в «режиме реального времени».

Как видите, единые рецепты на все случаи жизни дать нельзя. Каждый раз приходится решать «оптимизационную задачу» в применении к конкретной программе.

2. Команды 06—10 первой программы и 09—13 второй выделяют целую часть числа. С числом, большим единицы, желаемого результата можно было бы достичь с помощью более короткой цепочки команд ПА КИПА ИПА. Команда вида КИПМ в таком случае отсекает дробную часть у содержимого регистра M . (Если $M=0,1,2,3$, то из «усеченного» числа еще вычитается единица; если $M=4,5,6$, то единица прибавляется.) Однако, повторяем, этот способ применим лишь тогда, когда число в используемом регистре превышает единицу. При построении же графиков могут встречаться и числа, меньшие единицы. Поэтому здесь выбран иной способ: сначала к числу добавляется величина 10^7 , тем самым дробные цифры сдвигаются за восьмой разряд и пропадают, затем та же величина вычитается. Остаток как раз равен целой части исходного числа.

Раз уж зашла речь о графической обработке информации, то упомянем еще гистограмму — один из видов графического представления совокупности некоторых наблюдаемых величин X_i , распределенных в диапазоне между m и M .

Для построения гистограммы диапазон (m, M) делится на n равных интервалов с длиной $(M-m)/n$. Потом подсчитывается число наблюдений M_i , находящееся на i -й интервал. Затем можно определить частоту $H_i=M_i/N$, где N — общее число наблюдений, и приступить к рисованию.

На горизонтальной оси откладываются равные промежутки, пропорциональные выбранным интервалам, и на них строятся прямоугольники, высоты которых пропорциональны M_i .

Москвич А. Бойко написал программу, с помощью которой калькулятор сортирует значения X_i по адресуемым регистрам. Три из 14 регистров приходится выделить для служебных нужд. Таким образом, максимальное число интервалов, на которое можно дробить диапазон изменения анализируемой величины, равно 11.

Программа составлена так, что высоты прямоугольников хранятся в регистрах от 0 до A. После ввода всех значений X_i остается узнать содержимое этих регистров, вооружиться карандашом и бумагой и рисовать гистограмму.

Программа. 00.ПД 01.— 02. \geq 03.П0 04.0 05.КП↑ 06.FL0 07.05 08.П0 09.FC 10.: 11.ПС 12.С/П 13. ИПД 14.— 15. ИПС 16.: 17.ПВ 18.КИПВ 19.1 20.+ 21.КПВ 22.БП 23.12.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Очистить счетчик команд (B/0).
4. Начальный ввод: $n \uparrow M \uparrow m$ С/П. На индикаторе $(M-m)/n$.

5. Ввод: X_i С/П.
6. Продолжение работы: перейти к п. 5.
7. Результаты: в n регистрах, начиная с нулевого.
8. Работа с новой гистограммой: перейти к п. 4.

Примечания.

1. Величина n не должна превышать 11. В противном случае программа будет работать неправильно.

2. Верхняя граница диапазона (M) должна обязательно быть больше всех вводимых чисел. Поэтому если максимальное значение переменной, получаемое в эксперименте, равно X_M , то в качестве M необходимо выбрать число хотя бы на единицу в последнем разряде больше X_M , иначе переменная будет помещена не «по адресу».

Комментарий.

1. Команды 05—07 производят первоначальную очистку n регистров, отведенных для хранения M_i . В этом фрагменте используется команда КП↑, которая вызывает содержимое регистра, номер которого записан в Р0, но не модифицирует его. Модификацией занимается команда FL0, организующая цикл по переборке n регистров.

2. Если бы командам программы можно было бы ставить оценку, как ходам шахматной партии, команды 17—20,

безусловно, были бы достойны восклицательного знака. Автор программы необычайно остроумно решил вопрос о накоплении результатов в регистрах. Свойство косвенной адресации позволяет это сделать без всякого сравнения величин и помещать их в нужный диапазон, то есть в нужный регистр. Кстати, обратите внимание: если содержимое регистра, ответственного за косвенную адресацию, меньше единицы, обращение происходит к нулевому регистру, хотя целая часть из этого содержимого после обращения не выделяется.

Спорт + медицина = калькулятор

Мы уже упоминали, что сфера использования программируемых калькуляторов поистине необозрима. Программа, которая будет сейчас рассмотрена, родилась в кабинете спортивного врача. Идею ее предложил студент-медик из г. Иванова С. Назаров.

В практике спортивного врача важное место занимают профилактические осмотры спортсменов, когда определяется уровень их физического развития (рост, вес, объем легких) и физическая работоспособность. Данные осмотра сравниваются с величинами, средними для спортсменов того же пола и возраста. Оценка выводится так. Из полученной при обследовании величины показателя a вычитают среднее значение \bar{a} , и разность $\Delta a = a - \bar{a}$ делится на величину стандартного отклонения σ . Результат δ показывает, на сколько «сигм» фактический параметр отличается от стандартного.

Для удобства оценок применяется десятибалльная шкала, позволяющая сделать результат более наглядным.

Если он меньше — 2, то ставится 1 балл, если он больше — 2, но меньше — 1,5, то 2 балла и так далее, прибавляя каждый раз по 0,5 к значению $\Delta a / \sigma$ и по 1 к оценке. Наконец, результат, больший 2, получает наивысшую оценку, 10 баллов.

Запись дробных чисел в адресуемые регистры потребовала бы сравнительно большого количества команд или ручных операций. Кстати, подобные проблемы часто встречаются в задачах по классификации, типичных для медицины и биологии.

Чтобы обойти эту трудность, целесообразно заменить отклонение $\Delta a / \sigma$ на величину $2(\Delta a / \sigma + 2)$. Для этой вели-

чины границы классов станут целыми числами (0,1,2 и т. д.), что даст возможность воспользоваться свойствами косвенной адресации при определении принадлежности к классам.

Программа: 00.КНОП 01.С/П 02.ПС 03.F_O 04.— 05.ИПС 06.: 07.С/П 08.2 09.+ 10.FBx 11.× 12. F_x≥0 13.28 14.1 15.+ 16.П6 17.КИП6 18.ИП6 19.1 20.0 21.— 22. F_x<0 23.26 24.ИП6 25.В/0 26.FBx 27.В/0 28.1 29.В/0.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Очистить счетчик команд (В/0) и провести начальный запуск (С/П).

4. Ввод: $a\bar{a}\bar{\sigma}$ С/П.

5. Вывод: δ С/П. На индикаторе — оценка в баллах.

6. Продолжение работы: перейти к п. 4.

Контрольный пример: $a=163$, $\bar{a}=161.5$, $\sigma=3.8$.

Результаты: $\delta=0.39$, оценка=6.

Как видите, работа с программой проста и удобна.

Например, у лыжницы первого разряда при обследовании получены данные: рост — 163 см, вес — 62 кг, мышечная сила правой кисти — 36 кг. Из таблицы, приведенной в книге «Спортивная медицина», под ред. В. Л. Каримана, берем эталонные значения: рост 161.5 ± 3.8 , вес 60.3 ± 4.2 , сила 41.2 ± 6.0 . В результате обработки данных получаем для этих параметров значения: 0.39, или 6 баллов, 0.4, или 6 баллов, и — 0.87, или 4 балла. Весь расчет трех величин занимает не более двух минут. Если учесть, что для комплексной оценки состояния спортсмена иной раз требуется анализ нескольких десятков показателей, ясно, что без калькулятора врачу приходится очень трудно.

При наличии соответствующих таблиц программа может быть использована для оценки здоровья людей в ходе диспансеризации, в педиатрии и других отраслях медицины.

Комментарий.

Программа очень короткая, всего 30 команд, но зато специфических приемов при ее написании использовано много.

1. Обратите внимание на необычное начало программы. Первой командой стоит КНОП, то есть отсутствие операции. К чему бы это? Теперь обратите внимание на команды по

адресам 25, 27 и 29. Там стоит В/0. Обычно команда эта используется для того, чтобы возвратить управление команде, следующей за обращением к подпрограмме. Однако если подпрограмм в программе нет (правильнее сказать не «подпрограмм», а «обращений к подпрограммам», то есть команд ГП), то команда В/0 передает управление по адресу 01. Иначе говоря, в этом случае она эквивалентна команде БП 01. Поэтому программа и построена так, чтобы первая ее команда была несущественна для работы.

Можно было бы, конечно, обойтись и без этого. Удалить команду КНОП и вместо В/0 подставить БП 00. При этом длина программы увеличилась бы на две ячейки. Но зачем удлинять программу, если из-за этого она не становится ни быстрее, ни удобнее?

2. По адресу 01 стоит команда С/П. Это тоже необычно. Естественнее выглядит команда останова, поставленная в конце. Однако специфика нашей программы в том, что остановов в ней много, фактически перед каждым В/0. Целесообразно объединить их, вынеся общие части ветвей в ствол программы. Еще две ячейки сэкономлены без ущерба для качества программы.

3. Рассмотрим внимательно фрагмент 14—17. Последняя команда фрагмента, КИП6 — это косвенный вызов содержимого регистра, номер которого записан в Р6, и увеличение этого содержимого на единицу. Если перед вызовом оно представляло собой не целое число, то при вызове дробная часть числа отбрасывается. В нашей программе эта команда используется не для вызова, а для того, чтобы отбросить дробную часть содержимого Р6 и заодно увеличить его на единицу. После такого примечания естествен вопрос: почему нельзя было аналогичным образом организовать команды 14 и 15? Это ведь еще более сократило бы программу! Оказывается, нельзя. Дело в том, что дробная часть при косвенных обращениях отбрасывается лишь в тех случаях, когда содержимое модифицируемого регистра больше единицы. В нашем же случае может оказаться, что величина $2(2+(\Delta a/\sigma))$ меньше единицы и применение команды КИП6 приведет к совершенно неправильному результату.

Об этой особенности команд косвенной адресации нужно постоянно помнить, иначе ошибки неминуемы.

Вот, оказывается, как много интересного может быть в очень короткой программе!

По алгоритму Жюля Верна

Не показалось ли вам, что работа с микрокалькулятором настраивает на серьезный лад? Такое впечатление было бы обманчиво, ведь калькулятор может и шутить. Не верите? Тогда давайте разберем предлагаемую ниже программу.

«Когда после достаточно долгого периода хорошей погоды барометр начинает быстро и непрерывно падать — это первый признак дождя», — пишет Жюль Верн в своей книге «Пятнадцатилетний капитан».

Точный прогноз погоды — дело исключительно сложное. Для его определения мало собирать огромное количество метеорологических данных. Приходится еще решать систему из нескольких сот уравнений, что не всегда под силу даже очень большой ЭВМ. Поэтому даже Гидрометцентр дает точный прогноз, мягко выражаясь, не всегда. А вот приблизительный прогноз особого искусства не требует. Если ломят кости, то это к дождю, село солнце в облака — жди ветра, а если звезды видны хорошо, то день будет ясным.

То, что предложил научный сотрудник Тартуского государственного университета Г. Славин, лежит, наверное, посередине между прогнозом Гидрометцентра и прогнозом по «эломоте в костях».

Обученный по его программе калькулятор, пользуясь вводимой в него информацией, выдает свой «прогноз». От владельца калькулятора при этом требуется немного — уметь следить за барометром, знать, какое время года на дворе, и... немного юмора. Все остальное калькулятор берет на себя. Он же подсказывает пользователю, в каком порядке вводить информацию.

Программа: 00.Сх 01.4 02.6 03.ПД 04.5 05.П0 06.1 07.П6 08.ИП6 09.ВП 10.5 11.С/П 12.КП6 13.FL0 14.08 15.ИП4 16.2 17.— 18.Fx=0 19.24 20.1 21.ПВ 22.БП 23.43 24.ИП3 25.Fx≠0 26.28 27.1 28.ИП5 29.+ 30.ПВ 31.ИП4 32.Fx=0 33.64 34.ИП5 35.Fx=0 36.43 37.ИП6 38.Fx=0 39.55 40.ИП2 41.Fx≠0 42.53 43.1 44.ИП2 45.+ 46.ПА 47.2 48.F10^x 49.Х 50.ИПВ 51.+ 52.С/П 53.3 54.КБПД 55.ИП6 56.2 57.— 58.Fx≥0 59.62 60.4 61.КБПД 62.5 63.КБПД 64.ИП6 65.3 66.— 67.Fx=0 68.72 69.ИП2 70.Fx=0 71.76 72.2 73.ИП2 74.— 75. КБПД 76.6 77.КБПД.

Отойдем на сей раз от сухого написания инструкции. Во-первых, получается она довольно большой, а во-вторых, сама проблема к сухости не располагает.

Опишем работу с программой «нестрого».

После ее ввода следует нажать клавиши В/0 и С/П и сообщить калькулятору исходную информацию, на основе которой будет формироваться прогноз. Для этого требуется ввести пять чисел:

1. Состояние погоды: 0 — хорошая погода, 1 — плохая.
2. Давно ли стоит такая погода: 0 — недавно, 1 — давно.
3. Показания барометра: 0 — падают, 1 — растут, 2 — не изменяются.
4. Меняются эти показания: 0 — быстро, 1 — медленно, 2 — вообще не меняются.
5. Времена года: 0 — лето, 1 — зима, 2 — весна, 3 — осень.

Каждое из этих чисел вводится в ответ на очередной запрос микрокалькулятора, который, чтобы не спутать его с ответом, выводится в виде пятизначного числа. Например, 20 000 — запрос № 2.

Каждый ввод завершается нажатием клавиши С/П. После последнего, пятого, ввода калькулятор высвечивает «прогноз» в виде трехзначного числа АOB.

Цифра A показывает характер погоды, B — ее продолжительность. Расшифровываются показания нашего электронного предсказателя так. A: 1 — плохая погода, 2 — хорошая, 3 — ливень, гроза, 4 — сильный ветер, 5 — оттепель, снегопад, 6 — северный ветер, мороз. B: 0 — продержится такая погода совсем недолго (меньше суток), 1 — несколько дней, 2 — будет стоять долго.

Например, в калькулятор введена такая информация: хорошая погода (0) стояла долго (1); барометр падает (0) и причем быстро (0); за окном весна (2). Через несколько секунд прогноз готов: 401, то есть сильный ветер в течение нескольких дней.

После получения прогноза советуем включить радио или телевизор и послушать, какую погоду обещают на завтра. Если эта информация не совпадает с выданной калькулятором, значит, кто-то из двух предсказателей не прав.

Комментарий.

1. Основное отличие этой программы от рассмотренных ранее в том, что в ней практически отсутствуют расчеты. Это наглядный пример логической программы, где основной упор сделан именно на логику — сравнение, выбор пути решения и т. д. Если вы попытаетесь нарисовать блок-схему программы (а мы намеренно ее не приводим, чтобы читателю возможность поуражняться в таком рисо-

вании), то убедитесь, что она обилием ветвей напоминает пущистую новогоднюю елку.

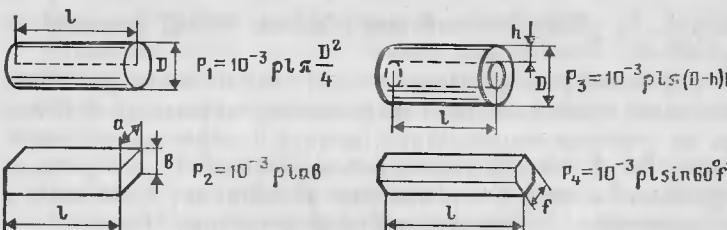
2. Рассмотрим фрагмент 08—14. Он отвечает за вывод запросов, прием ответов и их рассылку по регистрам. В качестве счетчика используется регистр 0, а рассылкой занимается Р6. Его содержимое, равное вначале 1, при каждом обращении к нему увеличивается на единицу, и все ответы записываются последовательно в регистры, начиная со второго. Интересно, что ответ на последний, пятый, запрос попадет как раз в регистр 6, тем самым блокируя изменение его содержимого. Это обстоятельство следует иметь в виду, используя косвенную запись в регистр. Если с самого начала не обратить на это внимание, то может оказаться, что регистр-передатчик сам попадает в череду пересылок и его содержимое изменится совсем не так, как предполагал автор программы. В нашем случае это происходит лишь при последнем обращении, поэтому ничего не портится.

3. Команды 47—51 формируют сообщение в виде АOB. Делается это просто. Величина A умножается на 100 и к результату прибавляется B. Этот способ стоит взять на вооружение, когда желательно одним выводом дать как можно больше информации. Ведь так можно собрать вместе до 8 однозначных чисел и одновременно вывести их на индикатор.

4. По адресам 54, 61, 63, 75 и 77 стоит команда косвенной передачи управления КБПД. В данной программе это сделано для сокращения длины программы. Правда, это потребовало трех лишних команд 01—03. Но все равно замена КБПД на БП 46 удлинила бы программу на две ячейки. Иногда сокращение длины большой программы просто необходимо. Адрес перехода в крайнем случае можно внести в соответствующий регистр и вручную. Об этом стоит помнить, когда программа не влезает в память, а свободные регистры имеются.

Сколько весит заготовка

И снова в цех, на этот раз в механосборочный. Часто технологам, работающим на таких производствах, для расчета норм расхода материалов приходится определять вес заготовок различных форм. Формулы расчетов вроде бы несложные, но очень уж утомительно производить на линейке или инженерном калькуляторе по нескольку умножений да и к тому же держать в уме расчетные формулы.

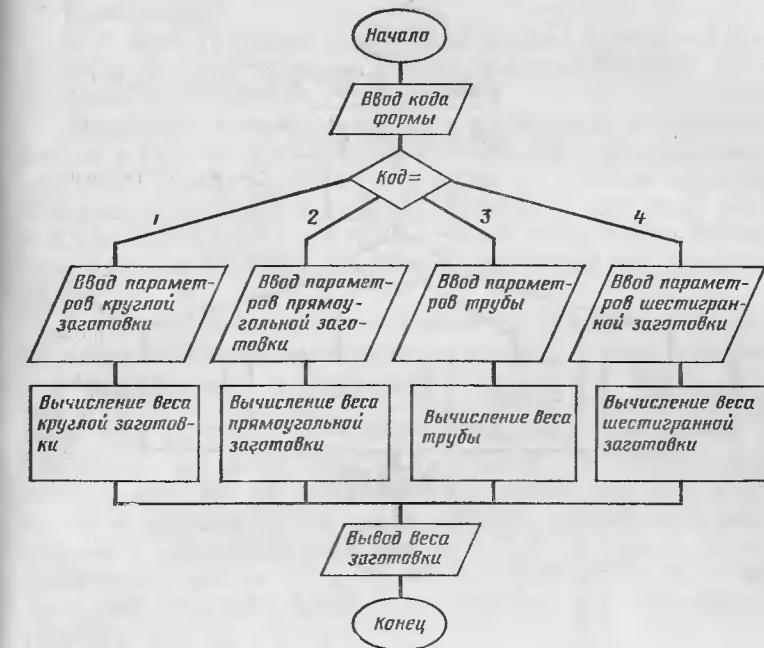


Поэтому и возникла у киевского инженера В. Дякина мысль сделать универсальную программу, позволяющую считать вес заготовок наиболее часто встречающихся форм: круглого, прямоугольного и шестигранного прутков и трубы. Расчетные формулы для этих типов заготовок приведены на рисунке. Буквой ρ обозначен удельный вес материала, $\text{г}/\text{см}^3$. Чтобы получить результаты в килограммах, дополнительно вводится множитель 10^{-3} . Остальные обозначения на том же рисунке.

По какому же пути пойти, составляя программу? Самый простой путь — написать четыре программы, каждая из которых будет вычислять вес заготовки только одного типа, вводить каждый раз требующуюся программу в память и вычислять. При всей своей простоте этот путь, однако, не самый лучший. А если требуется вычислить вес заготовок разных типов? Постоянно менять программы? Тогда проще уж работать в режиме ручных вычислений, нажимать соответствующие клавиши, и программа не нужна.

А что если объединить все эти программы в одной? Этот путь кажется эффективнее. Ведь достаточно включить калькулятор, ввести один раз программу и затем... Что затем? Хранить в своей памяти двузначные адреса и передавать управление той или иной части программы? Это тоже нудно. А может быть «закодировать» типы заготовок и поручить программе самой передавать управление нужной подпрограмме? Наверное, это самый удобный путь, благо язык микрокалькулятора предоставляет нам такую возможность, если использовать команды косвенной передачи управления на подпрограмму. Таким образом, вырисовывается первая версия блок-схемы решения задачи.

Но обратите внимание: для такой записи программы мы должны будем 4 раза повторить одинаковые операции — вычисление произведения первых трех сомножителей. Целесообразнее вынести эту общую часть вычислений в ствол дерева нашего вычислительного алгоритма. Еще один мо-



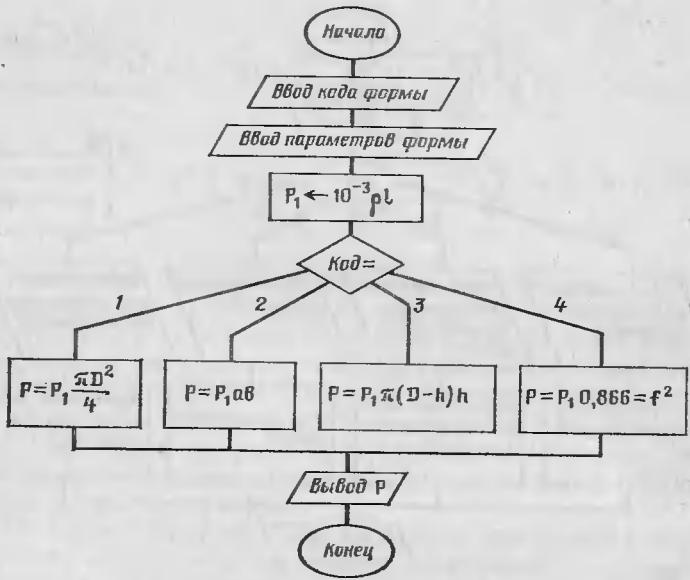
мент: когда и как вводить исходные данные? Здесь мы оказываемся перед альтернативой: либо вводить каждый параметр отдельно, либо все сразу в стек. В первом случае удобно сначала вводить «общую часть»: удельный вес и длину, потом «уникальную» — размеры сечения. При этом надо предусмотреть минимум два останова для вводов. Во втором случае вводим все сразу в стек и заставляем саму программу интерпретировать данные. Пожалуй, такой способ предпочтительнее. Для пользователя удобнее да и в программе лишнего останова ставить не надо.

Уточненная блок-схема будет иметь вид, показанный на втором рисунке.

Программа: 00.2 01.6 02.П1 03.3 04.1 05.П2 06.3 07.2 08.П3 09.3 10.9 11.П4 12.С/П 13.ПД 14.КИПД 15.ПС 16.С/П 17.× 18.3 19.F10^x 20.: 21.КППС 22.× 23.× 24.БП 25.12 26.≥ 27.Fx² 28.4 29.: 30.Fл 31. В/0 32. Fл 33. FО 34. F○ 35. — 36. FBх 37.× 38. В/0 39. ≥ 40. Fx² 41. 6 42. 0 43. Fsin 44. В/0.

Инструкция.

1. Ввести программу.



2. Перейти в режим вычислений (F ABT) и установить переключатель «Р—Г» в положение «Г»

3. Очистить счетчик команд (B/0) и произвести начальный пуск (С/П.)

4. Ввод кода формы: К С/П (круглый пруток 1 С/П, прямоугольный пруток 2 С/П, трубчатый пруток 3 С/П, шестигранный пруток 4 С/П).

5. Ввод параметров формы (в см):

- 5.1. Для круглого: $D \uparrow l \uparrow \rho$ С/П.
- 5.2. Для прямоугольного: $a \uparrow b \uparrow l \uparrow \rho$ С/П.
- 5.3. Для трубы: $D \uparrow h \uparrow l \uparrow \rho$ С/П.
- 5.4. Для шестигранника: $f \uparrow l \uparrow \rho$ С/П.

6. Результат — вес (в кг) на индикаторе.

7. Для продолжения работы перейти к п. 4.

Контрольный пример:

- 1) $K=1, D=2, l=50, \rho=7,9$. Вес: 1,24;
- 2) $K=2, a=5, b=4, l=40, \rho=7,9$. Вес: 6,32;
- 3) $K=3, D=4,5; h=0,25; l=70, \rho=7,9$. Вес: 1,85;
- 4) $K=4, f=3, l=60, \rho=7,9$. Вес: 3,69.

Комментарий.

1. В этой программе с помощью команд косвенного обращения к подпрограммам удалось реализовать блок многократного ветвления.

Введенный пользователем код по команде 13 записывается в РД, затем следующая команда (КИПД) «косвенно» считывает номер регистра, где хранится нужный адрес и пересыпает этот адрес в регистр С (если бы регистров не хватало, можно было бы и в тот же самый, то есть РД). Наконец, команда 21 КППС передает управление по нужному адресу.

2. При создании этой программы причудливым образом сплелись проблемы максимизации удобств и минимизации длины программы. С одной стороны, вынесение общих операций (двух умножений) в ствол программы (адреса 22 и 23) позволило сократить ее длину на 6 команд. А с другой... Программа могла бы стать на 15 команд (в полтора раза!) короче, если бы мы отказались от записи констант в ее тексте и вводили бы их перед началом работы непосредственно в адресуемые регистры (речь идет об адресах перехода, коэффициенте 10^{-3} и $\sin 60^\circ$, который можно было бы вычислить заранее и записать его величину в какой-нибудь регистр).

Нет ли здесь противоречия? Сокращая программу на 6 команд, мы одновременно удлиняем ее на 15. Противоречия нет. Еще раз напомним: сокращения целесообразны только в тех случаях, когда они не отражаются на удобствах. А что толку сократить программу на полтора десятка адресов и заставить пользователя перед началом работы нажимать столько же, а может быть, и больше клавиш? Кстати, и вероятность ошибки при большой доле ручного труда повышается.

Сокращение программы — не самоцель. Именно поэтому и стоит сократить 6 бесполезных команд, чтобы была возможность вставить 15 полезных и в конечном счете создать большие удобства пользователю.

3. Еще одна небольшая деталь. Обратите внимание, что при коде заготовки, равном 2, управление передается по адресу 31, где стоит команда В/0. Иначе говоря, реализуется переход на «пустую» подпрограмму. Это сделано, чтобы не нарушать общей структуры программы, не включать в нее анализ значения кода, загружая ее лишними в данном случае командами сравнения и передачи управления. Советуем и этот прием взять на вооружение.

В предыдущей главе мы коснулись вопроса об оптимизации программ. Теперь рассмотрим эту проблему подробнее.

Какую программу следует считать хорошей? Вопрос не так прост, как кажется. Прежде всего он молчаливо подразумевает, что есть несколько программ, решающих одну и ту же задачу. Ведь не будем же мы выбирать лучшую из программ, одна из которых вычисляет веса добавок для плавки, а другая решает дифференциальные уравнения. Итак, задача одна, программы разные. Первая, скажем, короче других, вторая — быстрее, третья — точнее, четвертая — удобнее и требует меньшие всего ручных операций. Какую из них предпочесть?

Здесь все определяется конкретной ситуацией. Если цель наша — просто решить какую-нибудь «одноразовую» задачу, то, наверное, самое главное — как можно быстрее получить ответ, то есть насколько возможно сократить путь от постановки задачи до получения результата. В этом случае, наверное, самая лучшая программа та, которая написана быстро. Иное дело, если решается задача «массовая», тем более если предназначеннная для этой цели программа будет многократно использоваться разными людьми. Здесь на первый план выходит время работы программы. Ведь даже небольшое его сокращение, помноженное на число пользователей и частоту обращений к программе, может дать ощутимый экономический эффект.

Если алгоритм настолько сложен и объемен, что его вообще трудно втиснуть в прокрустово ложе калькуляторной памяти, то главным критерием становится длина. Подчас такую программу и сравнивать не с чем, более длинный ее вариант просто не влезет в калькулятор. Немаловажна длина и при создании подпрограмм, рассчитанных на работу с большим классом программ, в том числе, естественно, и довольно объемных. Наконец, удобства (хотя это понятие и довольно расплывчатое) важны для любых программ. Чем программа удобнее в работе, чем меньшее число клавиш надо нажимать при вводе и выводе результатов, чем нагляднее интерпретация результатов, тем программа, естественно, лучше. Удобная программа экономит время пользователя при многократных расчетах да и, кроме того, работа с ней просто приятнее. А это, как известно, немаловажный фактор в любом труде.

Итак, как видите, однозначного решения для оценки качества программы не существует. В зависимости от поставленной цели на первый план выходит то одно, то другое ее свойство. Главное, что, составляя программу, всегда нужно помнить, что программа — это не самоцель, а средство для решения поставленной задачи и именно из этого исходить в первую очередь.

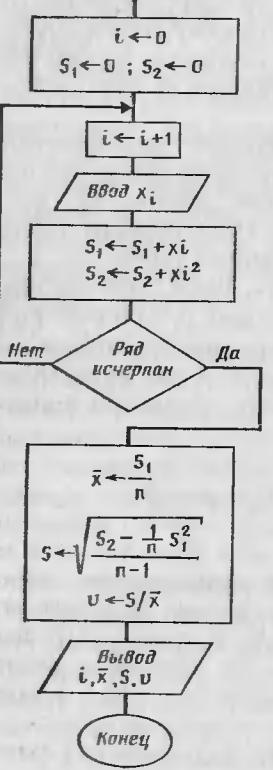
Однако это ни в коей мере не освобождает пользователя от знания приемов, позволяющих сокращать время работы программы и ее длину, естественно, без ущерба для удобства ее использования и ее точности. Некоторые из таких приемов иллюстрирует пример, описанный ниже.

В журнале «Наука и жизнь» (№ 12, 1983) была опубликована программа вычисления некоторых статистических характеристик: средней величины \bar{X} , среднеквадратического отклонения S и коэффициента вариации V для произвольного набора n исходных величин X_i . Эти параметры вычисляются по формулам:

$$\bar{X} = \frac{\sum X_i}{n}; \quad S = \sqrt{\frac{\sum (X_i - \bar{X})^2}{n-1}}; \quad V = \frac{S}{\bar{X}}.$$

Среднеквадратическое отклонение S характеризует абсолютный разброс величины вокруг среднего значения \bar{X} , а коэффициент вариации V — величина безразмерная, она дает относительное отклонение, степень изменчивости величины. По этому коэффициенту можно судить об устойчивости и однообразии протекающих процессов, о состоянии наладки оборудования и т. п. Количественные характеристики изменчивости имеют большое экономическое значение. Известно, например, что при производстве бетона уменьшение коэффициента вариации до 5 процентов вместо исходных 13 снижает расход цемента на 10—15 процентов: чем однороднее смесь, тем меньше цемента нужно для получения требуемой прочности. Аналогичные примеры можно привести и из других областей техники.

Прежде чем приступить к составлению программы, необходимо преобразовать формулу для вычисления среднеквадратического отклонения S . Использование ее в «перевозданном» виде привело бы к необходимости организовать два цикла: один — для подсчета величины \bar{X} , а второй, с использованием результатов первого, — для вычисления величины S .



Несложные алгебраические преобразования позволяют избавиться от такого дублирования. Записав формулу в виде

$$S = \sqrt{\frac{\sum x_i^2 - (\sum x_i)^2/n}{n-1}}$$

мы сможем провести все вычисления в одном цикле.

Блок-схема алгоритма дана на рисунке.

Было предложено реализовать алгоритм в виде такой программы (автор Н. Богин, г. Москва):

Программа 00.Cx 01.П3 02.П4 03.П5 04.П6 05.П7 06.С/П 07.П8 08.ИП3 09.+ 10.П3 11.ИП8 12.Fx² 13.ИП4 14.+ 15.П4 16.ИП7 17.1 18.+ 19.П7 20.БП 21.06 22.ИП3 23.Fx² 24.ИП7 25.: 26./—/ 27.ИП4 28.+ 29.П8 30.ИП7 31.1 32.— 33.F1/x 34.ИП8 35.× 36.FV 37.П5 38.С/П 39.ИП3 40.ИП7 41.: 42.П6 43.С/П 44.ИП5 45.ИП6 46.: 47.П8 48.С/П 49.ИП7 50.С/П.

Инструкция.

1. Включить калькулятор и ввести программу.
2. Перейти в режим вычислений (F АВТ).

3. Очистить счетчик команд (В/О) и запустить программу на счет (С/П).

4. Ввод X_i С/П. На индикаторе *i*.
5. После ввода всех чисел набрать БП 22 С/П.
6. Вывод: *S* С/П \bar{X} С/П *V*.
7. Для продолжения работы с новым числовым рядом: перейти к п. 4.

Контрольный пример: $X_1=2$, $X_2=3$, $X_3=4$.

Результаты: $S=1$, $\bar{X}=3$, $V=3,33333 \cdot 10^{-1}$.

Основную свою функцию — решение поставленной задачи — программа выполняет. Длина ее не так уж велика, чуть больше половины программной памяти микрокалькулятора. Посему можно считать работу над программой законченной. Однако после появления программы в журнале

редакция получила множество писем от читателей с предложениемми сократить длину программы и увеличить ее быстродействие. В данном примере оказалось возможным улучшить программу сразу по двум этим параметрам. Надо сказать, что для программируемого микрокалькулятора это вообще не редкость. Если программа не имеет циклов и в ней не используются подпрограммы, то почти всегда сокращение ее длины за счет избавления от «лишних» счетных команд ведет за собой и уменьшение времени ее работы.

Итак, окинем приведенную программу критическим взглядом. Первые пять команд введены, чтобы очистить адресуемые регистры Р3—Р7. Однако очистка регистров Р5 и Р6 явно лишняя, ведь их начальное содержимое не влияет на работу программы. Фактически команды 37 и 42, заносящие в эти регистры нужную для дальнейшей обработки информацию, и очищают их, стирая предыдущее содержимое. Далее можно исключить из числа рабочих регистров 8 (если уж экономить, то не только ячейки программы, но и адресуемые регистры — их еще меньше, чем ячеек программной памяти). Для этого достаточно заменить команды по адресам 07 и 11 соответственно командами ↑ и ⇢. Смысл фрагмента 16—19 в увеличении содержимого счетчика (Р7) на единицу. И здесь экономия числа команд лежит на поверхности. Почему бы для увеличения счетчика не использовать команду косвенной адресации? Правда, для этого считать количество введенных чисел стоит не в регистре 7, а в регистрах с номерами от 4 до 6. В этом случае для достижения того же эффекта достаточно пары команд: команды косвенного считывания и команды прямого считывания, скажем, КИП4 ИП4.

Явно лишней выглядит команда 26 (смена знака), за которой следует команда 28 — сложение. От нее можно избавиться, изменив расположение результатов предыдущих операций в регистрах стека, то есть заменить фрагмент из семи команд по адресам 22—28 последовательностью шести команд: ИП4 ИП3 Fx² ИП7 :—. Не оправдана и перегруженность фрагмента 29—35. Зачем засыпать промежуточный результат в Р8 для того только, чтобы один раз извлечь его содержимое командой 34? Требуемая величина преспокойно пребывает в стеке, в регистре У. Излишне и вычисление обратной величины с последующим умножением, проще разделить. В общем, программа совершенно не пострадает при замене 7 команд (адреса 29—35) четырьмя ИП7 1 :—.

Наконец, один принципиальный момент. Слишком уж неудобным выглядит переход от ввода информации к получению результатов. Требуется вручную нажать четыре клавиши: БП, 2, 2, С/П. Особенно неприятно, что действия пользователя никак видимым образом не отражаются на состоянии индикатора. Подобные ручные операции — дополнительный и весьма существенный источник ошибок. И если об уменьшении длины программы мы говорим в сослагательном наклонении («хорошо бы уменьшить»), то о сокращении числа ручных операций, на наш взгляд, следует говорить в наклонении повелительном («обязательно сокращайтесь!»). В рассматриваемой программе это легко сделать, используя специфическую особенность команды «ШГ вправо». Выполненная в режиме вычислений, она увеличивает содержимое командного счетчика на единицу, тем самым передавая управление команде, находящейся через одну от команды останова. Таким образом, если соответствующим образом составить программу, то достаточно вместо четырех клавиш (в первом варианте) нажать всего две — ШГ вправо и С/П. Важно отметить, что при этом вероятность ошибки сокращается не только благодаря использованию двух клавиш вместо четырех, но и за счет избавления именно от цифровых клавиш, таящих в себе наибольшие возможности для ошибок.

С учетом этих замечаний, высказанных в читательских письмах, А. Бойко, взяв за основу программу, предложенную ленинградским врачом В. Козловым, написал свой вариант:

Программа. 00.↑ 01.Cx 02.П4 03.П5 04.П6 05.F○ 06.↑ 07.ИП4 08.± 09.П4 10.↔ 11. F_x² 12.ИП5 13.+ 14.П5 15.КИП6 16.ИП6 17.С/П 18.БП 19.06 20.ИП5 21.ИП4 22. F_x² 23.ИП6 24.: 25.— 26.ИП6 27.1 28.— 29.: 30.FV— 31.↑ 32.ИП4 33.ИП6 34.: 35.: 36.FBx 37.С/П.

Инструкция.

1. Включить микрокалькулятор и ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Очистить счетчик команд (В/0).
4. Ввод: X_i , С/П; на индикаторе i .
5. После ввода всех чисел набрать ШГ вправо С/П.
6. Вывод: \bar{X} F○ V F○ S.
7. Продолжение работы: перейти к п. 3.

Комментарий:

1. Команда 00 вроде бы не нужна. Можно было бы после

очистки регистров поставить команду останова (С/П) и именно с этого момента начать диалог с калькулятором. При этом пользователь должен один лишний раз нажать клавишу С/П. Но почему бы не уменьшить число ручных операций хотя бы на одну? Это как раз та экономия, к которой всегда следует стремиться.

2. Использование команд косвенной адресации (15 и 16) для увеличения содержимого регистра б на единицу подробно описано как в тексте этой главы, так и в комментарии к предыдущей программе.

3. Клавиша «ШГ вправо» нажимается в тот момент, когда на счетчике команд стоит адрес 18. Именно он, точнее, его содержимое, и пропускается. Считывание следующей ячейки интерпретируется как код 06, то есть код цифры 6, которая и попадает в стековый регистр X. Но программа построена так, что число 6 впоследствии «заталкивается» на задворки стека и никак не отражается на работе программы.

4. Действие команды ↑ по адресу 31 описано в комментарии к главе «У токарного станка», п. 2.

Как видите, модернизированная программа получилась в полтора раза короче, чуть быстрее и намного удобнее в работе.

Но возвратимся к вопросу, стоявшему в начале этой главы. Можно ли, перефразируя слова Остапа Бендера в его бессмертной шахматной лекции, сказать, что программа Н. Богина плохая, а программа А. Бойко — В. Козлова хорошая? Нет, нет и нет!

Н. М. Богин писал свою программу, так сказать, на зарекалькуляторной грамотности. Программа задачу свою решала, широко эксплуатировалась на многих предприятиях и, безусловно, принесла большую пользу и экономический эффект. Поэтому назвать ее «плохой» просто несправедливо. Вообще, хорошая программа — это работающая программа. Другое дело, что ее можно улучшить. Как это делается, и было показано в этой главе.

Две скружности

Если сокращение длины программы, рассмотренное в предыдущей главе, носило, так сказать, показательный характер и преследовало цель улучшить программу, то при составлении программы в этой главе оно стало сурою необходимостью.

«Недавно я столкнулся с задачей, как определить координаты точек пересечения двух окружностей при известных координатах их центров и радиусах. Задача эта возникла при составлении управляющей программы для фрезерного станка с ЧПУ», — пишет М. Абдухин из Костромы.

Внешне задача выглядит довольно простой. Имеются два уравнения окружностей с центрами в точках x_1, y_1 и x_2, y_2 и радиусами r_1 и r_2 :

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = r_1^2, \\ (x - x_2)^2 + (y - y_2)^2 = r_2^2. \end{cases}$$

Рассматривая эти соотношения как систему двух уравнений с двумя неизвестными x и y , нужно найти все ее решения, то есть определить значения x и y , при которых удовлетворяются оба уравнения.

После замены переменных по формулам $\xi = x - x_2$, $\eta = y - y_2$ и несложных алгебраических преобразований система может быть записана в виде

$$\begin{cases} X\xi + Y\eta = \sigma^2, \\ \xi^2 + \eta^2 = r^2, \end{cases}$$

где $X = x_1 - x_2$, $Y = y_1 - y_2$, $2\sigma^2 = r_2^2 - r_1^2 + X^2 + Y^2$.

Формулы, определяющие решения этой системы, для удобства обозрения приведены в виде блок-схемы (см. рисунок). Заметим, что в зависимости от параметров окружности могут встретиться четыре ситуации:

1. Окружности вообще не касаются, то есть у системы нет действительных решений.

2. Окружности касаются в одной точке (одно решение системы).

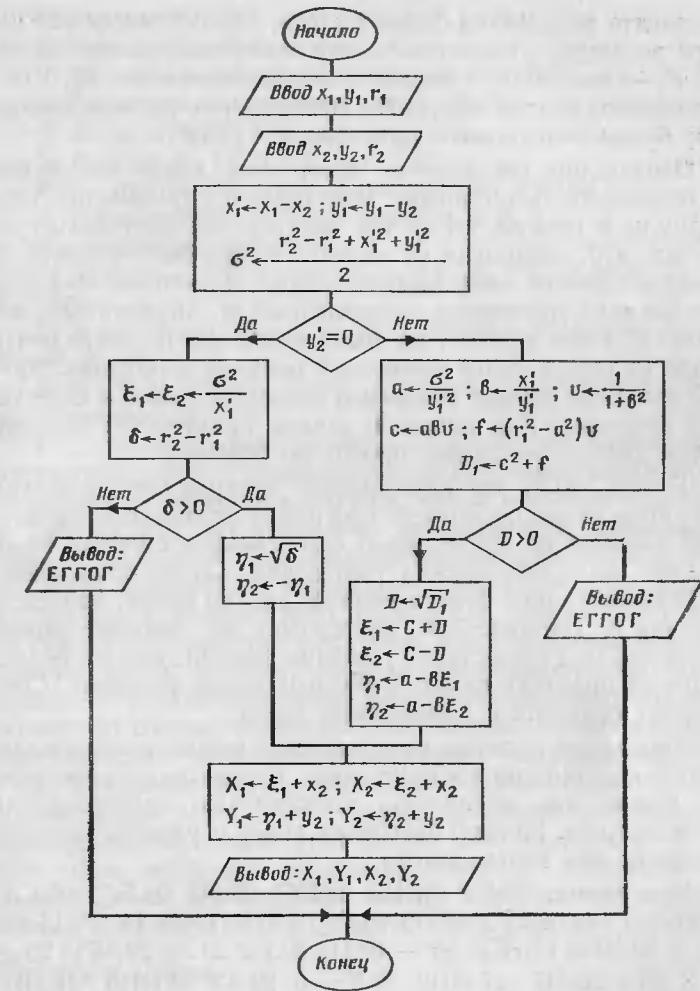
3. Окружности пересекаются в двух точках (два решения).

4. Окружности совпадают, система имеет бесконечное множество решений.

Отбросив четвертый случай, как не представляющий практического интереса, объединим второй и третий варианты, считая, что при касании точки пересечения сливаются, то есть система имеет два совпадающих решения.

Остаются, таким образом, два варианта: либо окружности вообще не касаются, либо пересекаются. Эти ситуации и отображены на блок-схеме.

Таким образом, задача математически сформулирована, алгоритм ее разработан, и можно приступить к составле-



нию программы. Первые этапы решения задачи трудностей не вызвали. Казалось бы, никаких сложностей не подстерегает нас и на очередном этапе — составлении программы. Ведь формулы просты и очевидны, никаких циклов в программе нет, ветвлений очень мало.

Сформулируем технические требования к программе. Пусть ввод будет организован самым очевидным образом: сначала параметры одной окружности, потом второй. На-

капливать результаты будем в стеке, благо там как раз четыре регистра, по максимальному числу получаемых чисел: X_1, Y_1 — координаты первой точки пересечения; X_2, Y_2 — координаты второй точки. На ситуацию, когда пересечения нет, будем реагировать сообщением ЕГГОГ.

Однако при составлении программы, столь несложной по предварительной оценке, и начинаются трудности. Хоть формулы и просты, но их так много и они настолько громоздки, что, записывая их «в лоб», можно быстро выйти за пределы памяти микрокалькулятора. Достаточно сказать, что вариант программы, составленный М. Абдухиным, занимал 95 ячеек и решал частный случай задачи, когда центр одной из окружностей совпадал с началом координат. Причем исходные данные вводились непосредственно в адресуемые регистры. Как видим, в данном примере без минимизации длины программы просто не обойтись.

Кстати, если вы начинающий программист, советуем вам, прежде чем посмотреть программу, попытаться сделать свой вариант по приведенной блок-схеме и с учетом изложенных выше технических требований. Скорее всего, работа над программой будет нелегкой, однако время, проведенное над ее составлением, безусловно, не пропадет даром: ничто так не способствует усвоению знаний, как их применение на практике да еще в экстремальных условиях. Свой вариант сравните с помещенным здесь.

Программа впритык располагается в памяти микрокалькулятора, занимает все ее 98 ячеек. Рассматривать программу можно как наглядную иллюстрацию использования свойств стека. Именно благодаря этому и удалось записать алгоритм для калькулятора.

Программа. 00.Fx² 01.П3 02.F \bigcirc 03.П2 04.F \bigcirc 05.С/П 06.Fx² 07.П6 08.F \bigcirc 09.П5 10.F \bigcirc 11.П4 12.— 13.П1 14.Fx² 15.+ 16.ИП2 17.ИП5 18.— 19.П2 20.Fx² 21.+ 22.ИП3 23.— 24.2 25.: 26.П7 27.ИП2 28.Fx=0 29.43 30.ИП6 31.ИП7 32.ИП1 33.: 34.ПА 35.ПС 36.Fx² 37.— 38.FV $\sqrt{-}$ 39.↑ 40./—/ 41. БП 42.83 43.: 44.П9 45.ИП1 46.ИП2 47.: 48.П1 49.× 50.FBx 51.Fx² 52.1 53.+ 54.F1/x 55.× 56.П8 57.FBx 58.ИП3 59.ИП9 60.Fx² 61.— 62.× 63.≥ 64.Fx² 65.+ 66.FV $\sqrt{-}$ 67.— 68.ПА 69.FBx 70.ИП8 71.+ 72.ПС 73.ИП1 74.× 75.ИП9 76.≥ 77.— 78.ИП9 79.ИПА 80.ИП1 81.× 82.— 83.ИП5 84.+ 85.ПВ 86.≥ 87.ИП5 88.+ 89.ИПС 90.ИП4 91.+ 92.ИПA 93.ИП4 94.+ 95.ИПВ 96.≥ 97.С/П.

Инструкция.

1. Ввести программу.
 2. Перейти в режим вычислений (F АВТ).
 3. Очистить командный счетчик (В/0).
 4. Ввод: $x_1 \uparrow y_1 \uparrow r_1$ С/П $x_2 \uparrow y_2 \uparrow r_2$ С/П.
 5. Вывод: $X_1 F \bigcirc Y_1 F \bigcirc X_2 F \bigcirc Y_2$.
- При отсутствии точек пересечения: ЕГГОГ.
6. Продолжение работы: перейти к п. 3.
- Контрольный пример:
- 1) $x_1 = y_1 = r_1 = 1, x_2 = y_2 = 2, r_2 = 1. X_1 = 1, Y_1 = 2, X_2 = 2, Y_2 = 1;$
 - 2) $x_1 = 1, y_1 = 0, r_1 = 1, x_2 = 3, y_2 = 0, r_2 = 1. X_1 = 2, Y_1 = 0, X_2 = 2, Y_2 = 0;$
 - 3) $x_1 = 0, y_1 = r_1 = 1, x_2 = 3, y_2 = r_2 = 1.$ Результат: ЕГГОГ.

Комментарий.

1. Сравнение с блок-схемой показывает, что в программе отсутствуют ветви, анализирующие ситуацию, когда система корней не имеет. Оказывается, в этом нет необходимости. Случай, когда нет решения, автоматически выявляются командами 38 и 66. Там записана операция извлечения корня. Если подкоренное выражение отрицательно, то попытка извлечь корень из него приводит к выводу сообщения ЕГГОГ, а этого вполне достаточно для характеристики ситуации. Таким образом, команды эти убивают сразу двух зайцев: и утилитарные свои функции выполняют, то есть реализуют соответствующие блоки схемы, и сигнал ЕГГОГ при отсутствии действительных корней вырабатывают.

2. Как ни странно, комментариев к самой большой программе очень мало. Ведь никаких специфических приемов при программировании этой задачи использовано не было. Вся игра построена исключительно на использовании свойств стека, оптимальном выборе порядка вычислений. Только и всего. Но, как видите, и этого не так уж мало.

Простые числа

Каждая программа — своего рода художественное произведение, и трудно ожидать, что два программиста написали бы одинаковую программу. Это все равно, как если бы два поэта написали бы одинаковые стихи на заданную тему.

И если устраиваются конкурсы музыкантов на лучшее исполнение какого-либо произведения, конкурсы парикмахеров на создание лучшей прически, почему бы не устроить конкурс программистов на написание лучшей программы для решения какой-либо задачи?

Такой конкурс был проведен журналом «Наука и жизнь». В момент его объявления не обошлось без сомнений: по какому критерию оценивать конкурсные программы? Каждую из них считать лучшей? Ту, которая считает на одну минуту быстрее, или ту, которая на десяток команд короче, а может быть, ту, работать с которой удобнее?

К счастью, сомнения оказались напрасными. Оказалось, что самые быстрые программы, как правило, и самые короткие и самые удобные. Впрочем, это не удивительно. Ведь лучшие программы создают лучшие программисты, а они потому и являются таковыми, что владеют искусством писать быстродействующие, компактные и удобные программы.

Однако значение конкурса вышло за рамки определения победителей. Он превратился для всех участников в своеобразный семинар по программированию и разработке алгоритмов. Просматривая присланные решения, удалось выявить типичные ошибки, систематизировать интересные методы.

А раз и книга наша преследует учебные цели, то мы решили познакомить читателей с результатами конкурса.

Темой конкурса была программа по поиску простых чисел в их естественной последовательности. Простым числом, как известно, называется целое положительное число, большее единицы и не имеющее никаких делителей, кроме единицы и самого себя.

Читатель может задаться вопросом: а кому нужны такие числа? При сварке труб их не используешь, для определения веса металлических болванок они тоже не нужны. Но с другой стороны, существует множество научных дисциплин, связанных с анализом случайных процессов и других явлений, где эти числа просто необходимы. Есть даже специальные таблицы простых чисел.

Числа эти занимали умы математиков с глубокой древности. Один из первых алгоритмов для поиска простых чисел принадлежит древнегреческому математику Эратосфену, жившему в третьем веке до нашей эры. Алгоритм его, получивший название решета Эратосфена, состоит в том, что выписываются подряд все натуральные числа и вычеркивается каждое второе число, большее двух, потом каждое третье, большее трех, потом каждое пятое, большее пяти (каждое четвертое уже было вычеркнуто, когда вычеркивалось каждое второе), и так далее. «Просеиваются» сквозь это решето и остаются невычеркнутыми лишь простые числа, притом все без исключения: 2, 3, 5, 7, 11...

Было сделано множество попыток найти формулы, позволяющие получить все простые числа. Однако до сих пор таких формул не найдено ***.

Существуют лишь алгоритмы (об одном мы уже упомянули), с помощью которых можно строить последовательности простых чисел.

Именно отсутствие формул и подогрело интерес к конкурсу. Ведь для того чтобы проверить число «на простоту», необходимо убедиться в отсутствии у него делителей. А сделать это можно по-разному. С выбора способа проверки и начинается построение алгоритма.

Самое первое, что приходит в голову, — брать все числа подряд и делители к ним примерять тоже подряд. Программы, составленные по этому алгоритму, могут быть довольно компактными. Но работать они будут слишком долго. (Одна из них, по оценке самого автора, число порядка 10^9 должна проверять на простоту около 10 лет.)

Многие участники конкурса правильно заметили, что четные числа явно не простые, потому и проверять их не нужно. Программы, составленные с учетом этого соображения, не сложнее первых. Какая разница, увеличивать число в цикле на единицу или на двойку? Работают же они существенно быстрее. Кстати, а на каком наибольшем делителе нужно прерывать проверку? Увы, очень многие участники конкурса испытывали делители вплоть до $n/2$ или $n/3$, хотя довольно очевидно, что достаточно ограничиться величиной \sqrt{n} . И здесь многие проявили излишнюю аккуратность. Они сравнивали делители с целой частью этой величины, что требовало проведения на калькуляторе целого ряда лишних операций. Достаточно же при анализе на «больше — меньше» сравнивать делители с самой величиной \sqrt{n} , пусть и не целой. Отметим ещё один «риф» на этом пути. Это касается уже самого программирования. Так как сравнение делителя с величиной \sqrt{n} происходит

*** Интересно, что до наших дней находятся энтузиасты, продолжающие ломать голову над такими формулами. Были такие и среди участников нашего конкурса. Однако авторы, приславшие формулы, генерирующие все простые числа, ограничивались при этом лишь экспериментальной демонстрацией своих генераторов. Никто из них не доказал, что, во-первых, приведенные формулы позволяют «выудить» из натурального ряда все простые числа, а во-вторых, что все числа, полученные по этим формулам, простые. Пока этого не будет сделано, остается примириться с фактом, что аналитического решения проблемы не существует, и пользоваться алгоритмами, подобными решету Эратосфена.

в цикле, то многократное выполнение операции FV значительно увеличивает время работы программы. Лучше вычислять эту величину до начала цикла и запоминать ее в одном из адресуемых регистров. Тогда в теле цикла достаточно будет лишь вызывать ее значение. Это хоть и увеличит длину программы на пару команд, зато сократит время ее работы.

Очень многие присланные программы страдали еще одним недостатком, на который стоит обратить внимание. Выделение целой части числа в этих программах было оформлено в виде подпрограммы. Конечно, подпрограммы — одно из мощных средств программирования. Об этом свидетельствуют хотя бы примеры, собранные в этой книге. Но когда к подпрограмме обращаются всего один раз и причем прямым, не косвенным образом, то она превращается в своего рода архитектурное излишество. Она и удлиняет программу, и замедляет ее за счет выполнения команд перехода.

Мы потому столь подробно проводим здесь анализ допущенных огехов, что они типичны не только для решения данной задачи. Ими, как правило, грешат начинающие программисты при разработке разнообразных программ.

Вернемся к алгоритму перебора. Некоторые участники конкурса заранее исключили из рассмотрения не только четные числа, но и числа, кратные трем. Их программы еще более эффективны. Наконец, лучший (по крайней мере для реализации на микрокалькуляторе) алгоритм нашло несколько человек. Это А. Сельский (г. Сумы), В. Илюхин (г. Москва) и Г. Натансон (г. Ленинград). Они исключили из рассмотрения числа, кратные 2, 3 и 5.

Оказалось, что сделать это можно довольно остроумно. Впрочем, предоставим слово одному из победителей конкурса профессору Г. Натансону, который подробно описал этот алгоритм и к тому же снабдил свою программу весьма содержательными комментариями.

Можно заметить, пишет он, что числа, не кратные двум, трем и пяти, образуют последовательность, состоящую из групп по восемь чисел, начиная с семерки, причем разности между соседями в этих группах одинаковы: 6, 4, 2, 4, 2, 4, 6, 2. Длина каждой группы тоже одинакова: 30. Отсюда и проистекает идея алгоритма, по которому можно находить простые числа одно за другим, начиная с любого назначенного.

Делается это так. Делим назначенное число на 30 и запоминаем остаток. Затем составляем сумму из единицы и последовательности вышеупомянутых чисел до тех пор, пока сумма эта не превзойдет остатка. Потом вычитаем из испытуемого числа остаток и прибавляем сумму, иначе говоря, заменяем исходное число числом специального вида $n=30K+\delta$, где δ не делится на 2, 3 и 5. Таким образом, мы получаем число, не превосходящее исходного и заведомо не кратное перечисленным делителям. С этого числа начинается анализ на простоту. По ходу дела запоминаем также «номер добавки», то есть количество разностей, использованных при формировании δ , и начинаем перебор. Делители выбираются на основе той же последовательности, то есть чисел $7=1+6$, $11=7+4$, $13=11+2$ и так далее до 31; при этом смотрим, не «перескочил» ли восьмой делитель через \sqrt{n} . Если нет, строим новую восьмерку: $37=31+6$, $41=37+4\dots$ и снова делаем ту же проверку. Когда делитель становится больше \sqrt{n} , процесс оканчивается. Если ни на одном шаге испытуемое число не разделилось нацело ни на один делитель, заключаем: n — простое число. Высвечиваем его на экране, выбираем следующее число из ряда и продолжаем проверку.

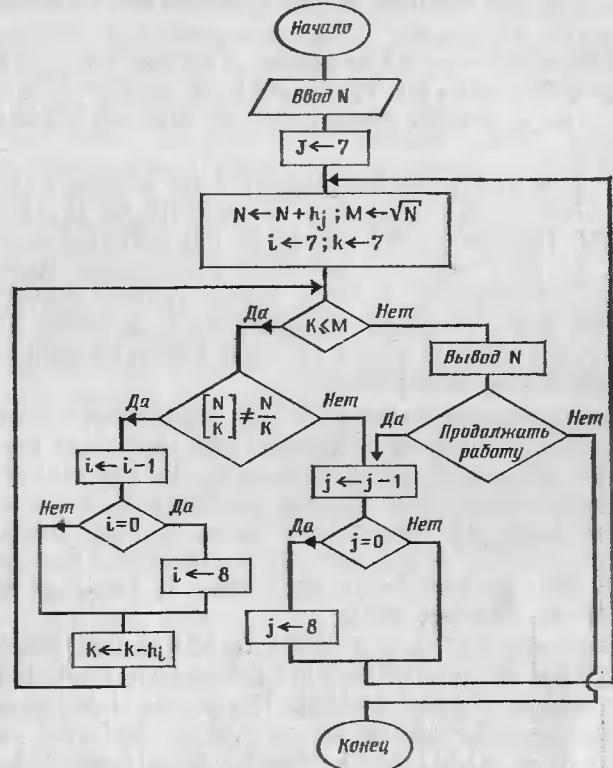
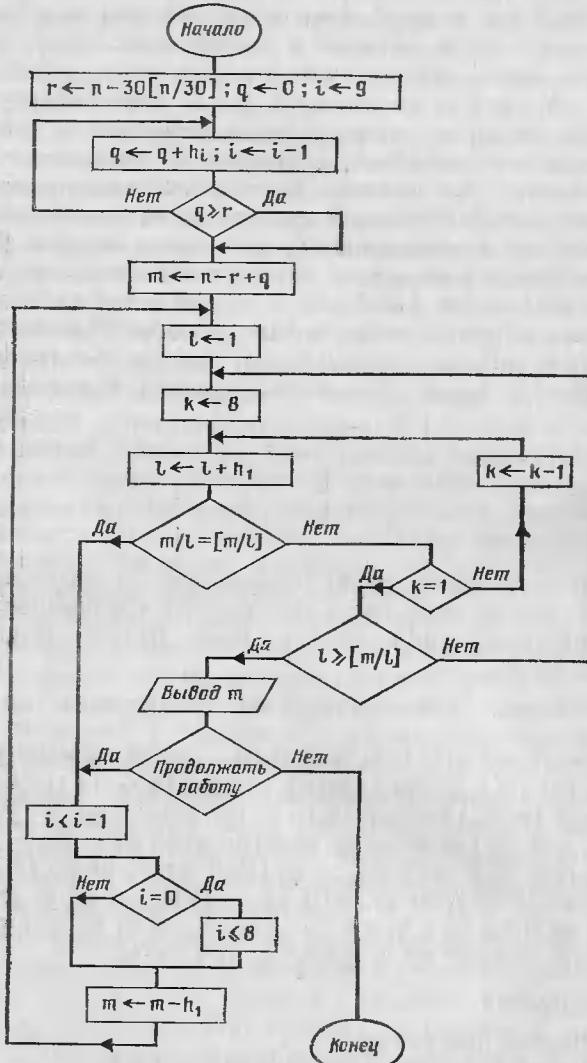
Если испытуемое число разделилось на какой-нибудь делитель без остатка, сразу переходим к следующему числу и повторяем цикл, пока не будет найдено очередное простое число.

Блок-схема этого алгоритма изображена на стр. 150.

Программа. 00.ПА 01.3 02.0 03.: 04.ПД 05.КИПД 06.1 07.0 08.ПО 09.1 10.П9 11.ИПД 12.ФВх 13.× 14.ИПА 15.— 16.КИП0 17.+ 18.Fx \geqslant 0 19.16 20.ИПА 21.+ 22.ПС 23.ИП0 24.ПВ 25.8 26.П0 27.ИПС 28.ИП9 29.КИП↑ 30.+ 31.П9 32.: 33.ПД 34.КИПД 35.≥ 36.ИПД 37.— 38.Fx \neq 0 39.49 40.ФЛ0 41.27 42.ИП9 43.ИПД 44.— 45.Fx \geqslant 0 46.25 47.ИПС 48.С/П 49.ИПВ 50.1 51.П9 52.— 53.Fx=0 54.56 55.8 56.ПВ 57.КИПВ 58.ИПС 59.+ 60.ПС 61.БП 62.25.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Начальный ввод: 2 П1 6 П2 4 П3 2 П4 4 П5 2 П6 4 П7 6 П8 (на блок-схеме эти числа обозначены h_i).
4. Очистить счетчик команд (B/0).



5. Ввод: n С/П.

6. Вывод: простое число на индикаторе.

7. Продолжение работы.

7.1. Для получения следующих простых чисел: С/П.

7.2. Для получения ряда с новым числом: перейти к п. 4.

Примечание: n должно быть больше 30.

Комментарий.

1. Команда 05. КИПД используется для выделения целой части содержимого регистра D ($n/30$). Мы уже отмечали, что правильно такая команда будет работать только при содержании указанного регистра, большем единицы. Отсюда и ограничение на минимальную величину вводимого числа.

2. Команда 12. F Вх восстанавливает предыдущий результат. Последним действием перед ней было деление на

30 (команда 03). Промежуточные команды 04—11 (запись и считывание из памяти, задание констант в программе) содержимое регистра X1 не меняют. Поэтому команда F Вх вызовет в РХ число 30. Таким образом, можно не запоминать это число в адресуемом регистре, экономя и команду, и регистр.

3. В этой программе используются две модификации команды косвенного считывания из регистра 0: 16.КИП0 и 29.КИП†. Напомним, что первая из них работает «классически», то есть сначала уменьшает содержимое нулевого регистра на единицу, а затем, рассматривая полученное число как номер регистра, вызывает его содержимое в РХ. КИП† — это «черный ход» в Р0. При использовании ее содержимое Р0 не модифицируется.

Программа победителя конкурса московского студента В. Илюхина еще короче и быстрее. Она настолько оптимизирована, что даже блок-схема (см. стр. 151) не дает полного представления о ней. Правда, за быстроту и краткость программы надо платить. В качестве платы здесь выступает специальная форма начального числа, с которого начинается анализ. Оно должно иметь вид: $30K+1$, где K — любое целое число, большее нуля.

Программа. 00.ПД 01.7 02.ПА 03.КИПА 04.ИПД 05.+ 06.ПД 07.ФУ 08.ПС 09.7 10.ПО 11.ПВ 12.ИПС 13.ИПВ 14.— 15.Fx \geq 0 16.34 17.ИПД 18.ИПВ 19.: 20.П9 21.КИП9 22. \Rightarrow 23.ИП9 24.— 25.Fx \neq 0 26.36 27.КИП0 28.Fx \neq 0 29.44 30.ИПВ 31.+ 32.БП 33.11 34.ИПД 35.С/П 36.ИПА 37.1 38.— 39.Fx=0 40.02 41.8 42.БП 43.02 44.9 45.ПО 46.БП 47.27.

Инструкция.

Она полностью совпадает с инструкцией для предыдущей программы, за исключением п. 3. Он должен быть записан так:

3. Начальный ввод: 6 П7 4 П6 2 П5 4 П4 2 П3 4 П2 6 П1 2 П8.

Комментарий.

При описании этой программы мы решили отойти от традиционной формы подачи материала. Комментария не будет. И не потому, что пояснить нечего. Просто мы считали, что так как алгоритм решения задачи вам известен и похожая программа прокомментирована, будет очень полезно самостоятельно разобрать эту программу и взять на вооружение то, что вам покажется интересным.

Диагноз ставит микрокалькулятор

Мы уже приводили пример использования микрокалькулятора в медицине. Там речь шла об обследовании здоровых людей и оценке их физических данных.

Программа, которая будет описана сейчас, выполняет более ответственную работу: она оценивает степень тяжести болезни. Речь идет о довольно неприятном заболевании — вирусном гепатите (болезни Боткина). Это острое заболевание человека с поражением печени. Для лечения его необходимы подробные обследования больного. Результаты анализов выражаются множеством цифр. На их основе врач назначает оптимальный режим лечения. Но как свести это множество цифр к одной характеристике? Как объективно оценить тяжесть заболевания?

Алгоритм для выработки такой численной характеристики был предложен академиком Г. И. Марчуком, математическая модель разработана Вычислительным центром СО АН СССР и кафедрой детских болезней II Московского медицинского института. Результаты применения модели оказались блестящими. Они подробно описаны в книге Н. И. Нисевич и В. Ф. Учайкина «Тяжелые и злокачественные формы вирусного гепатита у детей» (М., Медицина, 1982). Авторы этой книги сетуют только на то, что не всякая больница имеет ЭВМ для расчетов по этой модели. А между тем с помощью программы, разработанной Г. Славиным из г. Тарту, все расчеты по этой модели может выполнить программируемый микрокалькулятор. Низкая цена делает его доступным любой без исключения больнице. А для овладения им вполне достаточно знакомства с нашей книгой.

Расчеты проводятся следующим образом. Сначала вычисляется биохимический индекс φ_6 , характеризующий степень функционального нарушения. Расчет его идет по формулам: $p=b+3B$; $K_1=2,2-0,12p$; $K_2=2-K_1$; $J_1=0,01[10(p-1,5)+2(f-1,2)+0,5(\beta-40)]$; $J_2=\frac{20}{\beta}+\frac{b+\beta}{5f}$; $\varphi_6=K_1J_1+K_2J_2$.

Здесь: b — количество свободного билирубина в миллиграмм-процентах;

B — количество связанного билирубина в миллиграмм-процентах (если обе величины выражены в микромолях на литр, их надо разделить на 17,1);

- β — содержание β -липопротеидов в условных единицах (по методу Бурштейна);
 f — активность Ф1—ФА (фруктоза-1-фосфатальделаза) в единицах экстинкции (метод Товарицкого в модификации Брагинского).

При $p \geq 10$ принимается $K_1 = K_2 = 1$; кроме того, если величины, стоящие в круглых скобках в выражении для J_1 , отрицательные, они заменяются нулями.

После этого считается другой параметр: φ_k — клинический индекс, характеризующий клиническое состояние больного. Для его расчета используется семь клинических симптомов: S_1 — вялость, S_2 — отсутствие аппетита, S_3 — частота рвоты, S_4 — желтушность кожных покровов, S_5 — размеры печени, S_6 — подкожные кровоизлияния, S_7 — беспокойство (капризность, нарушение сна).

Каждый из этих симптомов оценивается в баллах по степени его выраженности: 0 — симптом отсутствует (то есть показатель в норме), 1 — выражен слабо, 2 — умеренно и 3 — выражен резко.

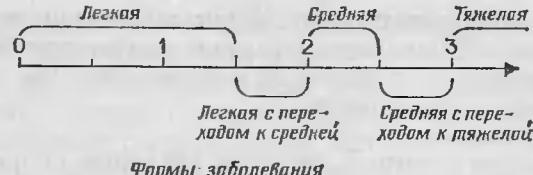
(Отвлечемся на минуту. Вспомните шуточную программу «Прогноз погоды». Ведь там проигрывалась сходная математическая ситуация. В этом и состоит величие математики. Подчас совершенно разные объекты и явления она описывает похожими алгоритмами. Вот и оказывается, что опыт при составлении программы-шутки используется при разработке серьезнейшей медицинской проблемы.)

Величина φ_k определяется как четверть суммы величин $S_1 - S_7$.

Наконец, совокупный индекс тяжести заболевания φ_t определяется так: $\varphi_t = 0,25\varphi_k + 0,75\varphi_b$ (коэффициенты 0,25 и 0,75 определены методом наименьших квадратов из условий наилучшего согласования φ_t с оценкой тяжести заболевания общепринятыми клиническими методами). Интерпретируем совокупный индекс согласно рисунку.

Анализ полученных индексов помогает правильно и объективно оценить тяжесть заболевания, своевременно диагностировать тяжелую форму и принять решение о проведении интенсивной терапии. Весьма плодотворен анализ индексов в динамике, с помощью их графического изображения в зависимости от времени.

Программа. 00.ИПА 01.ИПВ 02.3 03.х 04.+ 05.П9
 06.ИПЗ 07.ПП 08.90 09.ИП8 10.х 11.ИПД 12.ИП2 13.ПП
 14.90 15.2 16.х 17.+ 18.ИП5 19.ИП5 20.ПП 21.90 22.2



Формы заболевания

23.: 24.+ 25.2 26.ИП10 \times 27.: 28.П6 29.ИПА 30.ИПВ 31.+ 32.ИПД 33.5 34.х 35.: 36.2 37.0 38.ИПС 39.: 40.+ 41.ПА 42.ИП9 43.ИП8 44.— 45.Ф $x \geq 0$ 46.51 47.1 48.П7 49.БП 50.59 51.2 52.ИП4 53.ИП9 54.ИП1 55.х 56.— 57.П7 58.— 59.ИПА 60.х 61.ИП7 62.ИП6 63.х 64.+ 65.ПА 66.Сх 67.П6 68.П9 69.7 70.П0 71.КИП6 72.ИП6 73./—/ 74.С/П 75.ИП9 76.+ 77.П9 78.ФЛ0 79.71 80.4 81.: 82.ПВ 83.ИПА 84.3 85.х 86.+ 87.4 88.: 89.С/П 90.— 91.Ф $x < 0$ 92.95 93.ФО 94.0 95.В/0.

Так как программа занимает практически всю память калькулятора, места для сервиса осталось очень мало. Пришлось использовать комбинированный ввод: часть параметров засыпается непосредственно в адресуемые регистры, часть вводится «безадресно» в процессе работы программы в режиме диалога, поэтому инструкция, то есть программа для пользователя, получилась довольно насыщенной.

Инструкция.

1. Ввести программу.
 2. Перейти в режим вычислений (F АВТ).
 3. Ввод постоянных величин: 0,12 П1 1,2 П2 1,5 П3 2,2 П4 40 П5 10 П8.
 4. Ввод результатов анализов: b ПА B ПВ β ПС f ПД.
 5. Очистить счетчик команд (В/0) и запустить программу (С/П).
 6. Ввод семи клинических симптомов:
На индикаторе — число i со знаком минус; S_i С/П.
 7. Вывод: φ_t — на индикаторе;
ИПА — φ_b ;
ИПВ — φ_{β} .
 8. Продолжение работы с новыми данными: перейти к п. 4.
- Контрольный пример.
Данные биохимических исследований: $b=2$, $B=4$, $\beta=-78$, $f=44$. Клинические признаки получили оценки: $S_1=3$, $S_2=3$, $S_3=3$, $S_4=3$, $S_5=2$, $S_6=2$, $S_7=2$. Результат: $\varphi_t=3,06$ ИПА $\varphi_b=2,58$ ИПВ $\varphi_{\beta}=4,5$. По рисунку находим, что функциональное состояние больного

соответствует среднетяжелой форме заболевания с переходом к тяжелой, клиническая картина соответствует тяжелой форме, обобщенный индекс Φ_t расценивается как тяжелая форма вирусного гепатита.

Комментарий.

1. Мы уже отметили, что места для сервиса в программе не хватило. Конечно, лучше было бы задать все константы непосредственно в программе, а четыре изменяемых параметра вводить «безадресно» в стек. Но, как говорится, не до жири...

2. При вычислении каждого выражения в круглых скобках нужно анализировать результат. Если он больше нуля, то сохранить его, если нет, считать, что он равен нулю. Подобные однотипные действия встречаются трижды. Стоит их выделить в подпрограмму (адреса 93—97). Таким образом экономится (в нашей программе) 3 ячейки ($2 \times 3 + 5 + 1 = 12$ вместо $3 \times 5 = 15$). Опять-таки, напомним, не будь этого — просто не было бы программы.

3. Ввод оценок клинических симптомов больного организован по тому же принципу, что и ввод оценок погоды («По алгоритму Жюля Верна»). Только там для того, чтобы не спутать номер запроса с вводимым числом, запрос выводился в виде пятизначного числа; здесь же, опять-таки для экономии программной памяти, попроще, просто знак минус перед номером запроса. Но и этого достаточно, чтобы не спутать вопрос с ответом. И еще одно различие. В программе «Прогноз погоды» вводимые оценки рассыпаются по регистрам для их последующего использования, здесь же они сразу идут «в дело».

4. Не будем перечислять все ситуации, когда по ходу работы программы используются свойства стека, — нужно было бы перечислять почти все ее команды. Обратите внимание хотя бы на фрагмент 51—58. Здесь используются все четыре стековых регистра. Важно, что при этом величины вводятся в стек именно в той последовательности, какая нужна для проведения арифметических операций. Только таким образом можно сокращать программу за счет использования возможностей стека.

Калькулятор считает время

Поистине неисчерпаема фантазия владельцев микрокалькуляторов. Каким только образом они не используют своего электронного друга! Калькулятор — счетчик, калькуля-

тор — предсказатель погоды, калькулятор — диагности, и вот еще один вариант: калькулятор — часы.

В основе работы практически всех современных часов лежит один и тот же принцип — счет. Какой-либо генератор вырабатывает с определенной частотой сигналы, затем сигналы эти пересчитываются и преобразуются либо в движение стрелок, либо в смену цифр на индикаторе. В качестве генератора может быть использован либо маятник (кругиальный или продольный), либо пластинка кварца, порождающая колебания электрического тока. В качестве считающего устройства в механических часах используется система шестеренок, а в электронных — микросхемы, такие же, как в обычных ЭВМ.

Так как в микрокалькуляторе время выполнения одной и той же операции примерно одинаково, то почему бы этот факт не использовать как идею задающего генератора? Тогда пересчитывающее устройство можно реализовать программным путем.

На этом принципе и построены различные программы — измерители времени. Правда, необходимо сделать оговорку. При разработке схем для электронных часов особое внимание уделяется их стабильности, независимости работы их от температуры и других внешних условий. Калькуляторные схемы такой стабильностью не обладают. Поэтому если запустить «калькуляторные» часы, скажем, на неделю, то, скорее всего, их показания будут значительно отличаться от реального времени. Но ведь и сам калькулятор не расчитан на непрерывную недельную работу. Для измерения же сравнительно небольших временных интервалов вполне приемлема точность, которую можно достичь с его помощью.

Мы познакомим вас с программами — измерителями времени трех типов. Первый — секундомер. Это программа должна отмерять различные промежутки времени и при остановке высвечивать их. Основное требование: остановленная в любой момент, она должна показывать истекший промежуток времени на индикаторе, а запущенная после этого — продолжать счет с того момента, где он был прерван.

Программа-таймер должна работать заданное число минут или секунд и остановиться после их истечения, выдав на индикаторе условный сигнал. С. Конин и А. Шарапов из Ленинграда рекомендуют для этого какое-либо «яркое» число, например $8.888888 \cdot 10^{-88}$.

Наконец, программа-часы должна полностью высвечивать на индикаторе текущее время.

А. Бойко предложил для использования калькулятора в качестве секундомера предельно простую программу:
00. + 01. + 02. + 03. + 04. + 05. + 06. + 07. + 08. +
09. + 10. + 11. В/0 12. В/0.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Очистить счетчик команд (В/0).
4. Начальный ввод 1 ↑ ↑ ↑.
5. Настройка: Сх С/П. Остановить программу через 100 секунд, считать число a с индикатора.
6. Рабочий запуск: $100 \uparrow a : \uparrow \uparrow \uparrow$ (заслать результат деления 100 на число a во все регистры стека) Сх С/П.
7. Вывод: С/П — на индикаторе время в секундах.
8. Продолжение работы.
 - 8.1. С суммированием времени: С/П.
 - 8.2. С очисткой «секундомера»: перейти к п. 6.

Комментарий.

1. Необычная программа, не правда ли? Зачем нужен десяток плюсов? Ведь тот же результат можно получить, используя всего один плюс. Результат тот же, да время получения его будет другим. Ведь команда В/0 выполняется за иное время по сравнению с командой сложения. А нам ведь нужно добиться, чтобы время выполнения операции было одним и тем же. Поэтому, вообще говоря, чем больше плюсов, тем точнее будет работать «секундомер». Автор программы решил, что десяток плюсов обеспечит вполне приемлемую точность. Он оценил ее величиной около одного процента.

2. Зачем в программе две команды В/0? Специфика программы в том, что она может быть остановлена в любой момент. Если при этом следующей командой будет +, то все в порядке — счет пойдет дальше. Если же случайно мы попадем на В/0, то после нее будет выполняться следующая команда. Вот и служит вторая В/0 своего рода контргайкой, фиксирующей программу и не дающей калькулятору выйти за ее пределы.

Кстати, если заполнить плюсами всю программную память вплоть до 97-й ячейки, то ни «гайка» (первое В/0), ни «контргайка» (второе В/0) не понадобятся. Программное колесо, прокрутившись до конца, начнет снова перебирать

команды с начала, таким образом программа окажется замкнутой сама на себя.

Требования к программе-таймеру не такие жесткие. Ведь таймер не надо останавливать, он должен остановиться сам, подобно струе песка в песочных часах, и тем самым дать сигнала об истечении заданного временного интервала. Вот программа-таймер ленинградцев С. Конина и А. Шарапова:

00.ИП9 01.ПО 02.Фп 03.Ф1/х 04.Ф1/х 05.ФЛО 06.02
07.ИП8 08.С/П.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Ввести «сигнал»: 8.888888 ВП 88 /—/ П8.
4. Очистить командный счетчик (В/0).
5. Ввести промежуток времени в секундах t П9.
6. Запустить программу (С/П).
7. Вывод: примерно через t секунд на индикаторе сигнал 8.888888·10⁻⁸⁸.
8. Продолжение работы: перейти к п. 5.

Комментарий.

Здесь все просто: используется цикл из команд 02—04, который прокручивается заданное число раз. Недостатком таймера является отсутствие настройки, ведь на разных экземплярах калькуляторов время выполнения однотипных операций, вообще говоря, не совпадает. От этого недостатка свободна программа Г. Ионова из г. Балашиха Саратовской области.

Эта программа в отличие от ранее рассмотренной после остановки выводит на индикатор заданное время (в минутах). Кроме того, допускается настройка варьированием числа, засыпаемого в Р7 (у автора это — 1/120).

Программа. 00.П8 01.ИП7 02.: 03.П1 04.ФЛ1 05.04
06.ИП8 07.С/П 08.БП 09.00.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Очистить командный счетчик (В/0).
4. Ввести: 1/120 П7.
5. Ввод: число минут t С/П.
6. Вывод: на индикаторе t
7. Продолжение работы: перейти к п. 5.

Комментарий.

В этой программе цикл образован оператором цикла

(адреса 04—05), передающим управление на самого себя. Это можно представить себе в виде мотора, вращающегося вхолостую.

Еще одну программу, связанную с измерением времени, придумал В. И. Лесняк из Днепропетровска. Здесь измерение времени — не самоцель, а средство для определения частоты пульса.

В качестве задающего генератора в этой программе, как и в предыдущей, используется цикл, замкнутый сам на себя. Зная количество циклов *C* за определенное время, скажем, за 10 минут, легко найти «константу цикла» — $10/C$, то есть время работы одного цикла. Отмерив теперь число циклов *K*, соответствующее 10 ударам пульса, мы легко сможем получить его частоту, число ударов в минуту. Частота будет равна K/C . Величина *K*, в свою очередь, определяется как разность между начальным содержимым регистра, по которому организован цикл, допустим РО, и его содержимым в момент останова.

Определение частоты пульса с помощью данной программы сводится, таким образом, лишь к двум нажатиям кнопок: сначала для отмера времени и потом для вычисления. Простота использования, довольно высокая точность и небольшое время работы (около 15 секунд на одного пациента) делают программу очень полезной и удобной при массовых обследованиях. Да и в домашних условиях ее использование способствует сочетанию приятного с полезным: и за здоровьем следить, и калькулятор осваивать.

Программа. 00.Cx 01.FLO 02.01 03.ИПС 04.↑ 05.ИПО
06.— 07.: 08.ПД 09.КИПД 10.ИПС 11.ПО 12.ИПД 13.С/П
14.БП 15.00.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Очистить счетчик команд (В/0).
4. Предварительная настройка:
 - 4.1. Ввести 3000 ПО.
 - 4.2. Запустить программу на счет (С/П) и остановить через 10 минут.

Занести содержимое РО в РС (ИПО ПС), С/П.

5. Работа с программой.

- 5.1. Нажав пульс у обследуемого, запустить программу с первым ударом пульса (С/П) и остановить сразу после десятого удара (С/П, на индикаторе 0).

5.2. Нажать С/П.

6. Вывод: частота пульса на индикаторе.

7. Продолжение работы: перейти к п. 5.
Комментарий.

Читателю, знакомому с предыдущими программами, разбор этой не составит труда. Обратим внимание лишь на один момент.

Программа состоит из двух частей. Первая (адреса 01—02) считает число циклов, соответствующих 10 ударам пульса. Вторая часть — команды 03—15. Они вычисляют частоту пульса и готовят калькулятор к обслуживанию следующего пациента. (Команда 00.Cx не в счет. Она просто очищает индикатор.)

Так вот для перехода от одной части программы к другой не используется никаких специальных команд, ни ШГ, ни БП 03. Нажим клавиши С/П во второй раз вызывает на счет именно вторую часть. В чем же дело? Здесь остроумно использован тот факт, что после останова начинает отрабатываться код в ячейке, следующей за той, где произошел останов. В нашей задаче нажатие клавиши С/П может прервать работу программы либо на «команде» **** FLO, либо на «команде» 01. Во втором случае после повторного нажатия клавиши С/П мы сразу попадаем на нужный адрес — 03. А в первом случае калькулятор после повторного пуска отрабатывает «хвост» команды, код 01, интерпретируя его как вызов в регистр X числа 1. Работа цикла, как видите, в обоих случаях прерывается. Что же касается единицы, то следующие команды выталкивают ее на периферию стека и она никак не отражается на вычислениях.

Кстати, это хорошая иллюстрация того факта, что ручной останов программы с последующим пуском может нарушить ее правильное выполнение за счет пропуска одной из команд.

Наконец, калькулятор — настольные часы. Авторы этой программы, уже упомянутые С. Конин и А. Шарапов, использовали свойство калькулятора — показывать во время работы на индикаторе содержимое регистра X. Правда, обычно цифры беспорядочно мелькают. Если же они в течение какого-то времени не меняются, то их легко разглядеть. Мельтешения при работе «часов» полностью избежать не удалось — оно будет в момент «передвижения» стрелок.

**** Термин «команда» взят в кавычки неспроста. Ведь команда (без кавычек) — это FLO 01. Речь же у нас идет о частях команды, точнее, о содержимом отдельных адресов, где хранятся отдельные коды.

Но как только стрелки передвинутся, в течение целой минуты на индикаторе легко наблюдать показания — часы и минуты. Часы можно подстраивать. Делается это варьированием содержимого РА (у авторов там находится число 10).

Программа. 00.ИП4 01.ИП8 02.— 03.П3 04.ИП6 05.ИП9 06.— 07.ПО 08.ИП8 09.ИП7 10.× 11.ИП9 12.+ 13.П5 14.ИП2 15.П1 16.ИП5 17.↑ 18.↑ 19.↑ 20.↑ 21.↑ 22.↑ 23.↑ 24.↑ 25.↑ 26.↑ 27.↑ 28.↑ 29.FL1 30.17 31.1 32.ИПА 33.Fx^y 34.Fx^z 35.КИП5 36.FL0 37.14 38.ИП6 39.П0 40.ИП5 41.ИП7 42.: 43.П5 44.1 45.— 46.Fx<0 47.50 48.Cx 49.П5 50.КИП5 51.ИП5 52.ИП7 53.× 54.П5 55.FL3 56.14 57.ИП4 58.П3 59.Cx 60.П5 61.БП 62.14.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F АВТ).
3. Ввод констант: 10 ПА 11 П2 24 П4 60 П6 100 П7.
4. Очистить командный счетчик (В/0).
5. Ввести текущее время: часы П8 минуты П9.
6. Запустить программу (С/П).
7. Выход: на индикаторе постоянно «точное» время.

Комментарий.

Принцип, который использован в этой программе, собственно говоря, уже описан — циклическое повторение одних и тех же действий. Правда, в отличие от простых программ здесь предусмотрено еще и изменение содержимого ряда регистров в зависимости от истекшего времени. Иначе говоря, это — настоящие часы, где 60 минут складываются в час, а в 24 часа идет обнуление счетчика часов. Десяток команд по адресам 17—28 играет роль «маятника» или, как сказали бы радиотехники, линии задержки, отмеряя положенный временной интервал.

Можно было бы подобрать другие команды, выполняемые за то же время, скажем, «медленный» синус — тогда писать бы пришлось много меньше команд. Но, во-первых, выполнение команды ↑ не зависит от содержимого операционного регистра (в отличие от вычислительных команд), а во-вторых, не меняет содержимого регистра X, что нам особенно важно для индикации времени.

В помощь сталевару

Структура этой главы отличается от всех предыдущих. Если обычно мы давали готовую программу и затем объясняли, как она построена и как работает, то эту главу мож-

но было бы назвать «Как это делается». Здесь будет подробно описан сам процесс написания программы. Разница между первым и вторым подходом примерно такая же, как между обучением молодого токаря путем показа ему готовой детали с разъяснением, как она получилась, и демонстрацией изготовления детали на его глазах.

Для демонстрации процесса построения программы мы выбрали задачу, присланную нам Ю. Воловичем, заместителем директора завода «Днепропресссталь» (г. Запорожье). Алгоритм решения задачи написал ленинградский профессор Г. И. Натансон.

...Чтобы сталь легче перерабатывалась, а изделия из нее дольше служили, ее легируют, то есть добавляют в нее некоторые элементы: никель или молибден, хром или ванадий. Общее число добавок может достигать доброго десятка. Каждый легирующий элемент в стали той или иной марки должен присутствовать в строго определенном процентном количестве. Заданное соотношение компонент достигается не сразу: по ходу плавки в металл приходится добавлять ферросплавы, в каждый из которых наряду с железом входит соответствующий легирующий элемент. Как же рассчитать количество добавок? Точный ответ на этот вопрос должен дать плавильный мастер.

Опытные мастера славятся тем, что умеют решать такие задачи, не прибегая к расчетам, одними им ведомыми умозрительными приемами. А если плавку ведет не очень опытный мастер? Ему без расчетов не обойтись. Да и опытному мастеру не помешало бы поверить интуитивную гармонию алгеброй. Но кого просить о помощи в расчетах? Обратиться в заводской вычислительный центр и, терпеливо дождавшись своей очереди, получить точнейший ответ? Это слишком долго. Не лучше ли призвать на помощь микрокалькулятор? Хоть он считает и медленнее ЭВМ из вычислительного центра, зато всегда под рукой.

Задача поставлена — теперь дело за ее математической формулировкой.

Как только загруженный в плавильную печь металл расплавился, в заводской лаборатории методами химического и спектрального анализа определяют его состав. Обозначим символами A₁, A₂, A₃... содержание каждого из присутствующих в исходном сплаве легирующих элементов, выраженное в весовых процентах. Символами B₁, B₂, B₃... обозначим содержание легирующих элементов в сплаве, который должен быть получен. Процентное содержание каждого

легирующего элемента в соответствующем ферросплаве обозначим $C_1, C_2, C_3\dots$. Вес добавки каждого ферросплава пусть выражается символами $P_1, P_2, P_3\dots$ (эти количества и подлежат определению); общий вес металла при отборе пробы — P_0 , а после добавления ферросплавов — P_k (его мы подсчитаем простым суммированием: $P_k = P_0 + \sum P_i$). Попробуем составить систему уравнений для отыскания весовых количеств P_1, P_2 и следующих. Общее их число обозначим N .

Есть хорошая пословица: «Умный начинает с конца, глупый кончает вначале». Начнем и мы по-умному — с конца. Преположим, что все весовые добавки определены нами верно. С ними металл, находящийся в плавильной печи, приобрел вес $P_k = P_0 + \sum P_i$. Поскольку состав сплава теперь соответствует норме, то процентное содержание в нем каждого из легирующих элементов выражается числами $B_1, B_2\dots$ Общий вес первого элемента в полученном сплаве равен: $(P_0 + \sum P_i) B_1$. С другой стороны, первый легирующий элемент содержался в исходном сплаве в количестве $P_0 A_1$, а в добавке первого ферросплава — в количестве $P_1 C_1$. Сведем все эти величины в равенство:

$$P_0 A_1 + P_1 C_1 = (P_0 + \sum P_i) B_1.$$

Аналогично составим равенства, относящиеся ко второму, третьему и дальнейшим легирующим элементам:

$$P_0 A_2 + P_2 C_2 = (P_0 + \sum P_i) B_2;$$

$$P_0 A_3 + P_3 C_3 = (P_0 + \sum P_i) B_3;$$

...

$$P_0 A_N + P_N C_N = (P_0 + \sum P_i) B_N.$$

Так, равенство за равенством, сложилась система из N линейных алгебраических уравнений с N неизвестными: P_1, P_2, \dots, P_N . Методы решения таких систем хорошо отработаны, их нетрудно запрограммировать для нашего калькулятора. Для скольких же элементов мы сумеем произвести расчет таким путем?

Прежде чем решать систему, надо разместить в регистрах калькулятора коэффициенты при неизвестных величинах и значения известных, стоящих в правых частях уравнений. Систему из двух уравнений с двумя неизвестными образуют, как нетрудно подсчитать, шесть таких чисел, систему из трех — двенадцать, систему из четырех — двадцать... Но у нашего калькулятора всего четырнадцать регистров!

Значит, придется ограничиться всего тремя неизвестными? И составлять программу, которая позволит плавильному мастеру уточнять соотношение компонент в сплавах лишь весьма простого состава, где число легирующих элементов не превышает трех? Право, ради столь мизерного эффекта не стоит приниматься за программирование.

Но приглядимся к нашей системе внимательнее. Ее удивительно стройный вид позволяет надеяться, что у нее существует аналитическое решение. Любители математических выкладок без большого труда отыщут его:

$$P_i = \left(\frac{B_i}{C_i} K - \frac{A_i}{C_i} \right) P_0, \text{ где}$$

$$K = \frac{1 - \sum A_i / C_i}{1 - \sum B_i / C_i}.$$

Теперь возможности программы, которую мы собираемся составить, определяются уже не размерами выписанной выше системы уравнений, а тем, сколько чисел A_i, B_i, C_i удастся разместить в регистрах памяти нашего калькулятора. Если расчет придется вести для сплава с четырьмя легирующими добавками, всего таких чисел будет у нас двенадцать, если добавок будет пять — пятнадцать. Следовательно, для четырех добавок расчет, пожалуй, доступен. При этом два регистра еще остаются «в запасе». Только хватит ли их для хранения промежуточных данных да и величины P_0 , которую тоже нужно где-то хранить? Правда, можно использовать в качестве «хранилищ» и регистры стека. Навыки программирования, приобретенные нами по мере чтения предыдущих глав, должны помочь нам в поисках выхода из затруднительного положения.

Ну, а программная память? Хватит ли нам ее? Давайте прикинем. Чтобы вычислить каждую из дробей вида A_i/C_i или B_i/C_i и вычесть из единицы, как этого требует формула для коэффициента K , понадобится четыре операции: вызов числителя (A_i или B_i), вызов знаменателя C_i , деление, вычитание. Всего таких дробей восемь, так что все вместе они потребуют 32 команды. Плюс предварительный ввод единицы. Плюс последующее деление полученных разностей друг на друга и засыпка полученного коэффициента K в принятый ему регистр памяти. Да еще останется для засыпки величины P_0 . (Ее лучше засыпать не в начале работы, а после получения промежуточных сумм, когда часть регистров уже освободится). Итого 45 команд. А сколько команд уйдет на вычисление величины $P_i = (B_i K / C_i - A_i / C_i) P_0$? Поста-

раемся придать выражющим их формулам наиболее экономный с вычислительной точки зрения вид: $P_0 = (B_i K - A_i) P_0 / C_i$. Расчет по такой формуле, как нетрудно прикинуть, разворачивается в десять команд. И если всего неизвестных величин P_i четыре, то на их получение по приведенным формулам требуется 40 команд. Прибавляя их к ранее полученным, 45, получаем 85. Это как минимум. А удобства работы с программой? Они тоже требуют программной памяти. А в нашем распоряжении — всегда 98 адресов. Так что, если мы желаем создать программу, удобную в работе, мы должны работать на «пределе», максимально ужать ее вычислительную часть и для того прибегнуть ко всем подходящим здесь ухищрениям.

Такой целью следует задаться уже сейчас, когда мы только начинаем анализировать стоящую перед нами задачу.

Приглядимся еще раз к формулам, по которым определяются величины P_i . Всюду здесь участвуют не сами по себе числа A_i, B_i, C_i , а их отношения A_i/C_i и B_i/C_i . Быть может, стоит для каждого легирующего элемента заносить в регистры памяти не три числа A_i, B_i и C_i , а только два — A_i/C_i и B_i/C_i ? Тогда четырнадцать регистров, имеющихся в нашем калькуляторе, смогут вместить информацию уже не о четырех, а о семи элементах.

Далее. Расчеты выражений $(1 - \sum A_i/C_i)$ и $(1 - \sum B_i/C_i)$ требуют N -кратного выполнения совершенно однотипных операций вычитания; вычисления величин P_i протекают столь же единообразно. Быть может, следует придать всем этим вычислениям циклический характер?

Правда, в таком случае один из регистров памяти нужно будет превратить в счетчик циклов. Более того, к организации циклов придется, по всей вероятности, привлечь еще один регистр — и вот почему. Циклов у нас будет по меньшей мере два: один — для получения коэффициента K , другой — для вычисления величин P_i . По окончании одного понадобится обновить содержимое счетчика циклов, а для этого надо где-то запастися число N , указывающее, сколько раз надо прокрутить цикл.

Для хранения дробей A_i/C_i и B_i/C_i из четырнадцати регистров остается лишь двенадцать. Приходится ограничиться расчетами марок стали не более чем с шестью легирующими элементами. Это не так уж плохо. Ведь начинали мы с трех! И кроме того, на практике таких марок — изрядное количество. Стало быть, составленная нами программа будет иметь реальный практический смысл.

Но коль скоро она предназначается для практического использования, мы с самого начала должны учесть реальные условия, в которых она будет применяться.

Не нужно думать, что, получив из химической лаборатории данные анализа, плавильный мастер сможет уединиться в укромном уголке, где от него мигом отлетят все заботы, связанные с его хлопотливой должностью. Отнюдь нет! Окружающая его напряженная производственная обстановка будет отвлекать его, мешая верно и безошибочно ввести в калькулятор исходные данные. Лучший способ предотвратить ошибки — придать исходной информации предельно четкий и стройный вид. Например, свести ее в таблицу (см. стр. 177).

Здесь в первом столбце проставлены сверху вниз номера легирующих элементов, во втором — их названия, в третьем — их реальное процентное содержание в сплаве (A_i), в четвертом — их требуемые доли (B_i), в пятом — их процентное содержание в соответствующих ферросплавах (C_i), в шестом — искомые веса добавок. Все столбцы, кроме третьего и шестого, можно заполнить перед плавкой, сразу по получении данных анализа занести их в третий столбец и, проведя расчет, заполнить его результатами последний, шестой, столбец.

В каждой строке таблицы перед началом расчета — три числа, относящиеся к определенному элементу. Чтобы свести к минимуму вероятность ошибки, очевидно, следует вводить эту информацию в калькулятор следующим образом: ввести сначала в стек три числа A_1, B_1, C_1 из первой строки таблицы одно за другим и запустить фрагмент программы, который вычислит и отошлет на нужные места две дроби: A_1/C_1 и B_1/C_1 . Этот фрагмент должен быть оформлен в виде цикла с остановом (в начале или в конце). После останова надо ввести в стек числа A_2, B_2, C_2 из второй строчки таблицы и снова прокрутить тот же цикл, разумеется, изменив адреса рассылки вычисленных на сей раз дробей A_2/C_2 и B_2/C_2 . И так далее. Конечно, перед этим должен быть пройден фрагмент программы, устанавливающий в счетчике циклов число их повторений. Останов, которым будет окончен этот фрагмент, целесообразно совместить с остановом, прерывающим каждое прохождение «рассыльного» цикла.

Теперь подумаем о порядке расстановки дробей A_i/C_i и B_i/C_i . Здесь мыслимы два варианта. Первый: разместить в одном месте рядом все дроби A_i/C_i , в другом — все дроби

B_i/C_i . Второй: разместить дроби обоих сортов в чередующейся последовательности. Какой из вариантов лучше? И еще вопрос: в какой последовательности расставлять дроби: по регистрам с убывающими или с возрастающими номерами?

Поскольку с каждым прохождением цикла нам придется изменять адреса рассылки, то тут не обойтись без команд косвенной адресации вида КПМ, где M — номер регистра-счетчика циклов. По находящемуся там в каждый очередной момент времени и будет определяться адрес засылки каждой очередной дроби A_i/C_i и B_i/C_i . Но если дроби разных сортов станут рассыпаться в разные участки массива регистров, то нам потребуется уже не один, а два счетчика. На такое расточительство мы пойти уже не можем. Итак, из соображений экономии расставляем дроби в чередующемся порядке, парами: рядом A_1/C_1 и B_1/C_1 , потом A_2/C_2 и B_2/C_2 ...

Те же соображения экономии вынуждают нас возвращаться к началу цикла по команде вида FLM. Но в этих командах в роли счетчика циклов могут фигурировать лишь регистры с номерами M от нулевого до третьего. С каждым выполнением команды FLM число, находящееся в регистре M , будет уменьшаться на единицу. При указанных номерах M команда КПМ также будет уменьшать на единицу содержимое регистра-счетчика перед каждым ее выполнением. Значит, дроби надо расставлять по регистрам с убывающими номерами.

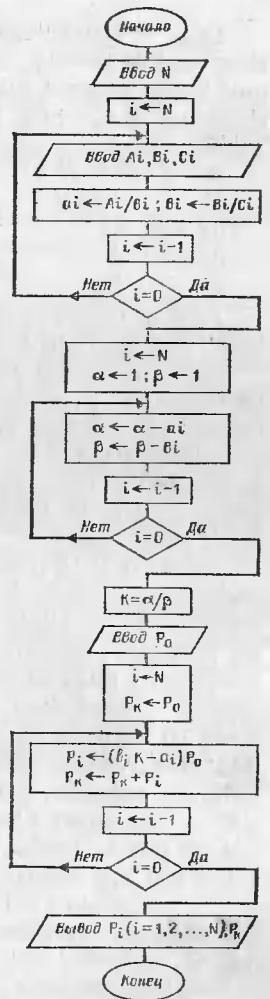
Но вот вопрос: не слишком ли много будет в цикле команд, уменьшающих содержимое регистра M ? Поскольку дроби мы собираемся расставлять парами, то с каждым прохождением цикла оно должно уменьшиться ровно на два. А у нас уже сейчас набирается по меньшей мере три команды, каждая из которых вычитет из счетчика по единице: засылка дроби A_i/C_i (команда КПМ), засылка дроби B_i/C_i (такая же команда), возврат к началу цикла (команда FLM)...

К счастью, среди команд косвенной засылки есть КП†. (О ней мы уже неоднократно упоминали.) Она посылает содержимое регистра X в регистр, номер которого равен числу, находящемуся в нулевом регистре, так же, как и команда КПО, но в отличие от нее не меняет содержимого нулевого регистра. Кстати говоря, это однозначно диктует нам номер регистра-счетчика: нулевой, то есть $M=0$. Ну, а номер регистра, куда будет заслана первая из наших дробей? Как его назначить? Это совсем просто. Номер будет ра-

вен удвоенному числу легирующих элементов. А само число $2N$ (или $N+N$) должно быть заслано в Р0. Тогда если первая из рассылочных команд будет иметь вид КП†, то при $N=6$ первая из дробей попадет аккурат в 12-й регистр, то есть в РС. А уменьшать содержимое счетчика будем в самом конце цикла.

Свободным при таком распределении памяти останется лишь регистр Д. В нем, очевидно, и следует хранить число повторений циклов N , а еще лучше — число, вдвое большее, то есть $2N$, чтобы обновление содержимого регистра-счетчика заключалось бы просто в пересылке туда числа $2N$.

Сказанным до сих пор алгоритм решения задачи был обрисован достаточно подробно для того, чтобы приступить к составлению эскиза его блок-схемы. Во многом она уже ясна из проведенных рассуждений. Сначала надо ввести в калькулятор N — число легирующих элементов, удвоить его и отослать результат в нулевой регистр и в регистр Д. Следующая часть блок-схемы будет носить циклический характер: ввод очередной тройки чисел A_i, B_i, C_i , вычисление двух дробей A_i/C_i и B_i/C_i , наконец, их рассылка по регистрам памяти. Циклической должна быть и следующая часть блок-схемы: вычитание дробей A_i/C_i и B_i/C_i из единицы. Следующий блок: вычисление коэффициента K и величин $B_iK/C_i - A_i/C_i$. Затем ввод величины P_0 . И еще одна циклическая часть: вычисление искомых величин P_i и их суммирование с P_0 , что дает итоговый вес плавки P_k . Вывод полученного значения P_k на индикатор. Наконец, заключительная часть, также циклическая: вывод на индикатор величин P_i .



Теперь приступим к программе. Чтобы лучше представить себе ее работу, используем способ ее записи, при котором после каждой команды станем приводить и содержимое стековых регистров. Начнем с того, что введем число N и, удвоив, запишем результат в РО.

00	↑	OE	N	N
01	+	10	2	N
02	ПД	4Г	—	
03	ПО	40	—	
04	С/П	50	—	

(После символа каждой команды записан ее код; прочерк означает, что содержимое регистра не изменилось.)

После останова вводим в калькулятор число A_1 , нажимаем клавишу \uparrow , вводим B_1 , нажимаем еще раз клавишу \uparrow и вводим C_1 . В стеке эти три числа расположатся в обратной последовательности. Все промежуточные результаты мы постараемся тоже размещать в регистрах стека. Впрочем, иначе и быть не может при нашей острой нехватке адресуемых регистров.

Числа B_1 и C_1 располагаются в стеке так, как нужно для вычисления дроби B_1/C_1 . Получим ее, выполнив операцию деления, и отправим по адресу, который «известен» нулевому регистру, выполнив команду КП \uparrow . Величина C_1 выпала из нашего поля зрения не насовсем. Она находится в регистре предыдущего результата РХ1. Подняв ее в РХ и немного погоняв числа по стеку, мы получим в свое распоряжение нужную пару: A_1 — в РУ и C_1 — в РХ. Снова выполняем деление, результат которого засыпаем на хранение уже с помощью команды КП0. Здесь уменьшение адреса нам на руку. Дробь A_1/C_1 попадает как раз в регистр, соседний с тем, где находится B_1/C_1 . Команда 12.FLO 13.04 уменьшает содержимое Р0 еще на единицу и передает управление на команду останова, при котором будут вводиться A_2 , B_2 , C_2 .

Структура цикла ясна полностью. Напишем его от начала до конца, повторив для наглядности команду по адресу 04.

04	С/П	50	C_i	B_i	A_i
05	:	13	B_i/C_i	A_i	
06	КП \uparrow	LE	—	—	
07	FBx	0	C_i	B_i/C_i	A_i
08	→	14	B_i/C_i	C_i	A_i
09	F0	25	C_i	A_i	
10	:	13	C_i/A_i		

11	КП0	L0	—
12	FLO	5Г	—
13	04	04	—

Одно прохождение цикла за другим — и вот уже последняя дробь A_N/C_N занесена в регистр 1, содержимое регистра 0 стало равным единице. Команда 12.FLO 13.04 совершают выход из цикла.

Пора приступить к вычислению K , а там — величин P_i и P_k . Команды косвенного вызова и засылки должны и тут способствовать краткости программы.

Впрочем, краткость — лишь один из критериев, по которым оценивается программа. Есть еще критерии быстроты и точности. Но они не доставят нам забот. Алгоритм, по которому мы решаем задачу, предполагает прямые расчеты по однозначным формулам. Арифметические операции, с помощью которых образованы наши формулы, дают ошибку от силы в последнем из восьми знаков, появляющихся на индикаторе нашей вычислительной машины. А данные лабораторного анализа содержат лишь по три значащие цифры. Результат расчетов, таким образом, будет не менее точен, чем исходная информация.

Убедившись в этом, продолжим работу над составлением программы. Теперь нам предстоит образовать разности $(1 - \sum A_i/C_i)$ и $(1 - \sum B_i/C_i)$. Будем вычислять их последовательным вычитанием дробей A_i/C_i и B_i/C_i из единицы. Для этого единицу надо дважды ввести в стек командами 1 и \uparrow . А еще раньше надо обновить содержимое счетчика циклов.

14	ИПД	6Г	2N
15	П0	40	2N
16	1	01	1 2N
17	↑	OE	1 1 2N

Опыт составления предыдущего цикла подсказывает, как произвести предстоящие вычисления. Советуем читателю самостоятельно написать этот цикл, а потом сравнить с приведенным здесь.

18	КИП \uparrow	ГЕ	B_i/C_i	1	1	2N
19	—	11	$I - B_i/C_i$	1	2N	:
20	→	14	I	$1 - B_i/C_i$	2N	:
21	КИП0	Г0	A_i/C_i	1	$1 - B_i/C_i$	2N
22	—	11	$I - A_i/C_i$	$1 - B_i/C_i$	2N	:

Перед возвратом к началу цикла расположим полуфабрикаты наших разностей в стеке так, как они расположены

гались там перед первым вычитанием, — поменяем их местами. Завершим цикл командой возврата к его началу.

23	→	14	$1-B_i/C_i$	$1-A_i/C_i$	2N
24	FLO	5Г	—	—	—
25	18	18	—	—	—

К окончанию цикла выражения $(1-\sum A_i/C_i)$ и $(1-\sum B_i/C_i)$ стоят в стековых регистрах X и Y именно так, как это требуется для вычисления коэффициента K. Получим его делением первого выражения на второе.

26 : 13 K

Настало время вычислять выражения: $(B_iK/C_i - A_i/C_i)$.

Подсчеты этих разностей будут носить опять-таки циклический характер. Обновим содержимое счетчика циклов:

27	IПД	6Г	2N	K
28	П0	40	2N	K

Казалось бы, в стековых регистрах и адресуемых регистрах памяти запасено все, что потребуется для проведения предстоящих вычислений. Но будем бдительны, ведь уже при подсчете первой же разности $(B_iK/C_i - A_i/C_i)$ коэффициент K исчезнет после его умножения на B_i/C_i . Чтобы не допустить этого, поместим копию K еще в одном регистре стека. Сделаем так:

29	F0	25	K	
30	↑	0E	K	K

Далее все пойдет привычным для нас путем использования команд косвенного вызова и засылки:

31	КИП↑	ГЕ	B_i/C_i	K	K
32	X	12	B_iK/C_i	K	
33	КИП0	Г0	A_i/C_i	B_iK/C_i	
34	—	11	$B_iK/C_i - A_i/C_i$	K	
35	КП↑	ЛЕ	—	—	
36	FLO	5Г	—	—	
37	29	29	—	—	

Итак, после первого прохождения цикла величина $B_iK/C_i - A_i/C_i$ очутилась в 11-ом регистре, то бишь в регистре В. По мере дальнейших повторений цикла аналогичные разности окажутся в регистрах с нечетными номерами: девятом, седьмом и так далее, до первого. Регистры с четными номерами, со второго по двенадцатый, можно использовать далее как рабочие: их содержимое нам уже не нужно.

После возврата по команде 36.FLO 37.29 к началу цикла, то есть на 29-й адрес, стоящие по этому и следующему адресу команды вновь продублируют коэффициент K, чтобы все было готово для следующего прохождения цикла. Вый-

дя из него, восстановим содержимое счетчика циклов: он еще потребуется нам для подсчета искомых добавок P_i и общего веса плавки P_k :

38	ИПД	6Г	2N
39	П0	40	2N

Чтобы провести такой подсчет, плавильный мастер должен ввести в калькулятор величину P_0 . Для этого выполнение программы нужно прервать. И хорошо, чтобы на экране остановившегося калькулятора горело какое-то условное число, которое напоминало бы мастеру, что он должен сделать. Например, число нуль. Зашилим его в регистр X, а потом скомандуем «стоп»:

40	0	00	0	2N
41	С/П	50	0	2N

Введем теперь значение величины P_0 с клавиатуры и, запустив программу вновь (С/П), перешлем величину P_0 из регистра X в один из освободившихся адресуемых регистров, например в двенадцатый, то есть в регистр С.

42 ПС 4C P_0

Осталось сделать немногое, но близость к завершению работы не должна притупить нашей осмотрительности. Обратим внимание: в счетчике циклов находится число $2N$, а нам для подсчета первой из искомых величин P_i надо вызвать разность $(B_iK/C_i - A_i/C_i)$ из 11-го регистра. Ясно — тут надо употребить команду КИП0, уменьшающую содержимое счетчика. Затем вызванную разность надо умножить на величину P_0 . Оба сомножителя стоят так, как это требуется для умножения — в регистрах X и Y. Но если подсчитать это произведение сейчас, величина P_0 исчезнет из стека, а нам она еще понадобится для подсчета итогового веса плавки $P_k = P_0 + \sum P_i$. Лучше поступим так: вызовем P_0 из регистра С, и нужные нам сомножители снова окажутся в регистрах X и Y, хотя и в ином порядке. Выполнив умножение P_0 на $(B_iK/C_i - A_i/C_i)$, перешлем результат в тот же 11-й регистр. Тут уж потребуется команда КП↑. После ее выполнения в регистре Y (проследите по выписанному ниже фрагменту программы движение информации по стеку) окажется величина P_0 , а в регистре X будет находиться P_1 . Такое расположение весьма кстати: сложив обе величины, мы начнем подсчет итогового веса плавки P_k , увеличившегося за счет добавок. После сложения можно возвращаться к началу цикла:

43	КИП0	Г0	$B_iK/C_i - A_i/C_i$	P_0
44	ИПС	6C	$P_0 B_iK/C_i - A_i/C_i$	P_0

45	\times	12	$(B_1 K/C_1 - A_1/C_1) P_0$	P_0
46	KП↑	LE	—	—
47	+	10	$P_0 + P_1$	
48	FL0	5Г	$P_0 + P_1$	
49	43	43	$P_0 + P_1$	

По выходе из цикла мы имеем в регистре X величину P_k — итоговый вес плавки. Можно было бы тут и остановить программу, чтобы эта величина вы wyświetлилась на индикаторе. Но ведь нам еще предстоит вывести и веса добавок P_i . Для этого придется организовать еще один цикл. Поэтому отсылаем величину P_k в один из освободившихся четных регистров (скажем, в 10-й, регистр А), восстанавливаем содержимое счетчика циклов, а уж затем командой FО возвращаем величину P_k в регистр X. Вот теперь, остановив программу командой С/П, считаем значение P_k с индикатора. Далее надо выводить значения P_1 , P_2 и последующие. Дело это несложное:

50	ПА	4—	P_k	
51	ИПД	6Г	$2N$	P_k
52	П0	40	$2N$	P_k
53	FО	25	P_k	
54	С/П	50	P_k	
55	КИП0	ГО	P_t	
56	FL0	5Г	P_t	
57	54	54	P_t	
58	С/П	50	P_t	

Остается раз за разом нажимать клавишу С/П и после каждого очередного останова считывать с индикатора P_1 , P_2 и следующие искомые величины. Когда будет считана последняя, произойдет выход из цикла — и тут будет можно поставить последнюю точку, команду С/П.

И все-таки повременим с окончанием работы над программой. Вспомним о тех непростых условиях, в которых ей суждено работать. Представьте себе, что плавильный мастер сбился, считывая с индикатора величины P_i , и, терпеливо дождавшись завершения цикла, хочет заново вывести на индикатор какую-то из них, скажем, P_3 . Где она размещается? Нетрудно составить таблицу соответствий и выяснить по ней, что P_3 находится в регистре 7. Но сколь бы ни был прост такой поиск, он все равно требует какого-то времени и напряжения внимания. Не лучше ли организовать расстановку полученных результатов так, чтобы индексы величин P_i совпадали с номерами регистров, их содержащих: P_1 — в первом регистре, P_2 — во втором и так

далее? А значение итогового общего веса плавки можно поместить в нулевой регистр (его номер 0 можно толковать как первую букву слова «общий»).

Следуя этим соображениям, перепишем заново заключительный фрагмент программы, начиная с 50-го адреса. При переходе к нему в регистре X находится величина P_k . Перешлем ее в нулевой регистр. Теперь организуем пересылку величин P_i на новые места: P_1 направим из 11-го регистра в 1-й, P_2 — из 9-го во 2-й ... P_6 — из 1-го в 6-й. Впрочем, вести пересылку в том порядке, в каком она здесь записана, нельзя: скажем, если сначала переслать содержимое 11-го регистра в 1-й, где до этого хранилась величина P_6 , то она пропадет, и ее уже неоткуда будет взять для засылки в 6-й регистр. Очевидно, пересылку лучше вести в обратном порядке и тщательно следить за заполнением первого, третьего и пятого регистров: новая информация в них должна попасть не ранее чем их содержимое перебазируется на новое место. Вот как можно поступить:

50	П0	40	P_k
51	ИП1	61	P_6
52	П6	46	P_6
53	ИП5	65	P_4
54	П4	44	P_4
55	ИП3	63	P_5
56	П5	45	P_5
57	ИП7	67	P_3
58	П3	43	P_3
59	ИП9	69	P_2
60	П2	42	P_2
61	ИПВ	6L	P_1
62	П1	41	P_1
63	ИП0	60	P_k
64	С/П	50	P_k

На индикаторе остановившегося калькулятора — итоговый общий вес плавки. Его надо списать. Теперь надо вывести на индикатор и считать значения P_i . Проще всего приписать для этого к программе последовательность команд вызова и пройти их одну за другой, нажимая клавишу ПП. На индикаторе по порядку станут возникать нужные числа:

65	ИП1	61	P_1
66	ИП2	62	P_2
67	ИП3	63	P_3
68	ИП4	64	P_4

Марка стали 06Х23Н28М3ДЗТ

69 · ИПБ 65 P_6
 70 · ИП6 66 P_6

Если плавильному мастеру потребуется вновь осведомиться об одной из полученных величин (например, P_3), ему достаточно будет нажать две клавиши (в нашем примере ИП 3) и считать запрошенное число с индикатора.

Сравним теперь новую и старую версии заключительного фрагмента программы. Он удлинился на 12 шагов. Тем самым мы, казалось бы, изменили своему стремлению составить как можно более короткую программу, о котором заявляли, еще только приступая к ее разработке. Но вспомним, что говорилось по этому поводу в главе «Сколько весит заготовка»: для того мы и сокращали «бесполезные» вычислительные команды, чтобы вставить полезные, сервисные.

Вот сейчас можно считать нашу работу по составлению программы законченной. Осталось привести инструкцию.

Инструкция к программе.

1. Ввести программу.
2. Перейти в режим вычислений (F ABT).
3. Очистить счетчик команд (B/0).
4. Ввести число легирующих элементов N С/П.
- 5: Вход:
 - 5.1. $A_i \uparrow B_i \uparrow C_i$ С/П.
 - 5.2. Повторять п. 5.1. до появления нуля на индикаторе.
 - 5.3. P_6 С/П.
6. Вывод: на индикаторе — P_k ПП P_1 ПП P_2 ... ПП P_N .
7. Повторный вывод — нажатием клавиш ИП*i*. На индикаторе P_i .
8. Продолжение работы с новыми данными: перейти к п. 3.

Контрольный пример представлен таблицей 4.

Итак, стоявшая перед нами задача решена, хотя поначалу в этом были сомнения. Казалось, что программу удастся составить для расчета лишь трех легирующих добавок. А в итоге вышло так, что их количество возросло до шести.

Комментарий.

Собственно говоря, комментировать программу нет смысла — это уже сделано в тексте. Поэтому ограничимся общими замечаниями.

Видно, что краткость программы, точнее, ее вычислительной части, достигнута в первую очередь благодаря организации циклов, причем с использованием команд кос-

№ п/п	Элемент	$A_i(\%)$	$B_i(\%)$	$C_i(\%)$	$P_i(\text{т})$
1	Хром	14,5	23	70	4,812
2	Никель	32,3	27	99	0,445
3	Марганец	0,4	0,6	96	0,086
4	Молибден	3,1	2,75	60	0,148
5	Медь	3,0	2,9	100	0,155
6	Титан	0	1,2	70	0,5

Начальный вес — 23 тонны.

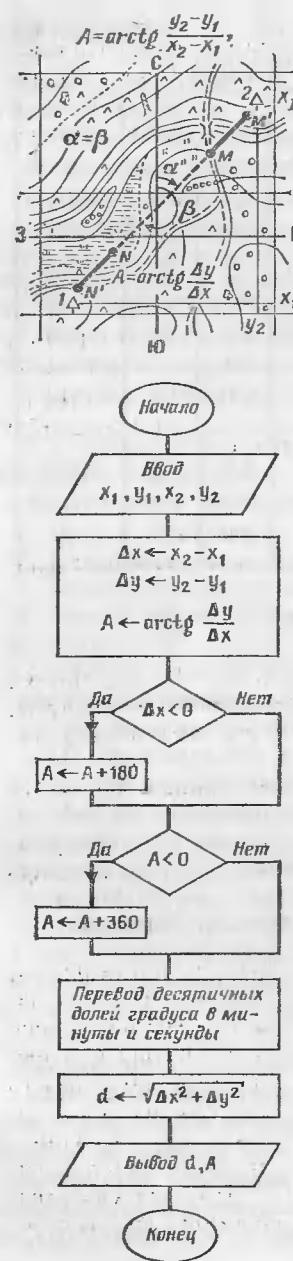
Конечный вес — 29,146 тонны.

венной адресации. В других случаях подобный эффект достигается за счет использования подпрограмм или других приемов, допускаемых возможностями калькулятора. Важно каждый раз искать. Кто ищет, тот всегда найдет!

Думаем, что некоторые выводы, вытекающие из работы над программой, могут оказаться полезными при создании программ, предназначенных для решения практических задач из различных сфер человеческой деятельности — механики и электроники, химии и сельского хозяйства...

Электронный топограф

Программа, которая будет здесь описана, — плод коллективного творчества. Первый ее вариант прислал В. Молчанов из Воронежа. Алгоритм для перевода долей градусов в минуты и секунды, используемый в этой программе, стал предметом конкурса, объявленного редакцией «Науки и жизни». Победителем конкурса был назван Э. Балуянц из Баку. Предложенная им реализация этого алгоритма,



модернизированная А. Бойко, и включена в нашу программу. Но, впрочем, начнем по порядку.

В топографии часто приходится решать так называемую обратную геодезическую задачу, когда по прямоугольным координатам двух точек на плоскости нужно определить расстояние между ними и дирекционный угол, то есть направление прямой линии, соединяющей эти точки. Дирекционный угол отсчитывается по часовой стрелке от направления на север и измеряется в градусах. Он может принимать значения от 0° до 360° .

Если вычисление расстояния никаких трудностей не представляет, оно равно $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, то с вычислением угла дело обстоит сложнее.

Формула $A = \text{arctg}(y_2 - y_1)/(x_2 - x_1)$ дает нужные результаты только для углов в северо-восточном направлении. Для углов в северо-западном направлении угол будет формально получен правильно, но будет он отрицательным, что недопустимо в топографии. Что же касается юго-западного и юго-восточного направлений, то углы, полученные по вышеприведенной формуле, будут неотличимы от северо-восточного и северо-западного направлений.

Поэтому, чтобы не перенуть север с югом, приходится усложнять алгоритм, учитывая знаки разностей $\Delta x =$

$= x_2 - x_1$ и $\Delta y = y_2 - y_1$. Кроме того, микрокалькулятор вычисляет значение угла в градусах и его десятичных долях, в то время как топографы привыкли иметь дело с минутами и секундами. Впрочем, о проблеме перевода мы уже упоминали.

Блок-схема алгоритма приведена на рисунке. С ее помощью легко проследить логику программы. Вот сама программа:

```

09. → 01.F ○ 02. → 03. — 04.П2 05.F ○ 06. — 07.П1 08.ИП2
09. → 10.: 11.Farc tg 12.ИП1 13.Fx<0 14.22. 15. → 16.1
17.8 18.0 19.— 20.БП 21.23 22. → 23.Fx<0 24.29 25.3
26.6 27.0 28.+ 29.3 30.F10x 31.+ 32.ПП 33.45 34.×
35.ПП 36.45 37.: 38.ИП1 39.Fx2 40. ИП2 41.Fx2 42.+ 43.FV
44.С/П 45.ПА 46.КИПА 47. → 48.ИПА 49.— 50.0 51., 52.6
53.× 54.ИПА 55.+ 56.2 57.F10x 58.В/0
  
```

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F ABT).
3. Установить переключатель Р—Г в положение Г-градусы.
4. Очистить счетчик команд (В/0).
5. Ввод: x_1, y_1, x_2, y_2 С/П.
6. Вывод: $d \rightarrow A$ (угол A выводится в виде 1 ГГГ, ММСС, то есть после цифры 1 три цифры означают градусы, две после запятой — минуты и две последних — секунды).
7. Продолжение работы с новыми данными: перейти к п. 4.
8. Контрольный пример: $x_1=2, y_1=1, x_2=4, y_2=2$. Результат: $r=2.24; A=26^\circ 33' 56''$ (на индикаторе 1026,3355).

Комментарий.

1. Вот ведь как бывает! Не удивительно, если программа, вычисляющая результат по десятку сложных формул, занимает около сотни ячеек, но здесь всего-то две небольшие формулы, памяти же занято всего раза в полтора меньше. Но, во-первых, программу, если есть в том необходимость (например, надо дополнить её еще какими-то вычислениями), легко сократить почти на десять команд, запрятив используемые в ней константы в адресуемые регистры, а во-вторых, длина программы — плата за удобства. Все ручные операции при работе с программой сведены к минимуму.

2. Все исходные данные засыпаются в стек. Как видно из алгоритма, нужны они лишь для получения разностей $x_2 - x_1$ и $y_2 - y_1$. Исходные данные вводятся в последовательности, удобной пользователю, а не ЭВМ. (Кстати, это очень важно: хотя программа написана для ЭВМ, работать с ЭВМ приходится человеку. Его запросы — главное.) Поэтому в первых адресах программы записаны команды, тасующие эти числа, как кубики в игрушке Рубика. При этом оказывается возможным получить нужные результаты без записи исходных данных в адресуемые регистры.

3. Команды 09—27 использованы для получения «правильного» значения дирекционного угла.

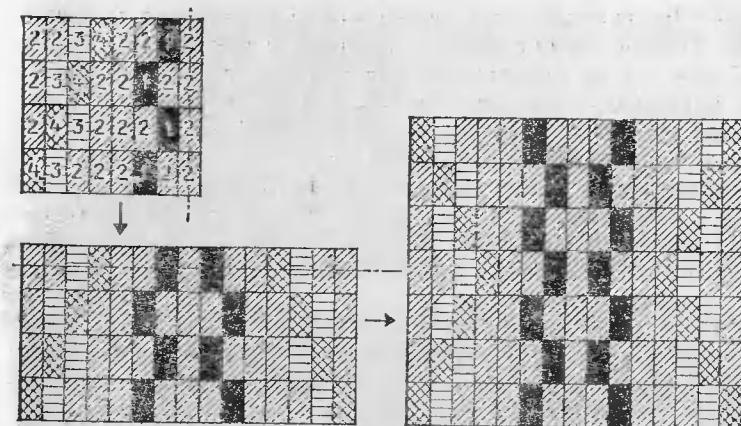
4. В ячейках 29—36 и 43—56 записан алгоритм перевода десятичных долей градуса в минуты и секунды. Остановимся на нем подробнее и начнем с вывода результата. В форме ГГГ, ММССД (Д — десятичные доли секунд) он довольно нагляден, за исключением случаев, когда число градусов меньше единицы. Число в диапазоне от 0 до 1 выводится с плавающей запятой, поэтому не так просто сообразить, что 7.24445—02 означает $0^{\circ}7'24,445''$, а 4.4444—03 значит $0^{\circ}0'44,444''$. Для борьбы с неочевидностью такого представления Э. Балуянц предложил увеличивать число градусов на 1000, то есть добавить в старший градусный разряд единицу. При этом независимо от числа градусов результат всегда будет выводиться в форме 1ГГГ, ММСС. Для этой цели и используются команды 28—30.

5. Сам алгоритм перевода десятичных долей градуса в минуты и секунды очень прост. Он записан в виде подпрограммы (адреса 43—56). Выделяются целая и дробная части угла. Дробная умножается на 0,6 и суммируется с целой частью, результат умножается на 100, и аналогичная процедура повторяется с выделением секунд. Затем полученное число, естественно, делится на 100 для восстановления порядка — и все готово.

6. Так как анализируемое число всегда больше единицы (не зря же мы 1000 добавляли), для выделения целой и дробной части смело можно использовать команды косвенной адресации (адреса 43—47).

Для тех, кто вяжет

Как часто любительницы вышивания стоят перед проблемой выбора узоров. Наверное, многие из них с завистью вспоминают про детский калейдоскоп: богатство появляю-



щихся в нем орнаментов поистине неисчерпаемо. И в то же время они строго симметричны, потому что получаются многократным отражением в зеркалах той картинки, которая складывается из цветных стеклышек в центральном поле. Но эти узоры так трудно зарисовать! Малейшее движение — и только что появившаяся картинка заменилась новой.

Дело решило исправить научный сотрудник Тартуского государственного университета Г. Славин. Микрокалькулятор по программе, созданной им, предлагает узоры в виде, наиболее удобном для вышивания.

Правда, микрокалькулятор дает лишь рисунок центрального поля, где вместо цветных пятен — цифры. Замена цифр цветами — дело нашего творчества. Заметим, что нули также обозначают некоторый цвет и используются для наглядности всей картины. В роли зеркал мы тоже будем выступать сами. Научиться этой интересной работе нетрудно. Берем основной прямоугольник размером 8 на 4, который предлагает нам «Электроника Б3-34», и ставим на него сбоку воображаемое зеркало. Картина удваивается. Затем ставим зеркало снизу картинки и удваиваем еще раз. Ясно, что такую операцию можно делать сколько угодно раз.

Количество цифр (то есть цветов) мы выбираем сами, помня, что истинная красота проста. Лучшие результаты получаются при выборе от двух до четырех цветов. Количество получаемых программой узоров хоть и не бесконечно, но достаточно велико. Для двух цветов — 256, для четы-

рех — более 65000, для шести — более полутора миллионов. Правда, если заказать орнамент с шестью или более цветами, то не обязательно все они будут присутствовать на картинке — это дело случая. Не понравилось — запустили программу еще раз, благо работает она не долго, не более двух с половиной минут.

Программа довольно велика — 94 команды. Но ведь и дел у этой программы многовато. Нужно случайным образом, как в калейдоскопе, выбрать цифры-цвета, сформировать из них картину, записать ее определенным образом в адресуемые регистры.

Программу нельзя назвать чисто вычислительной, хотя здесь есть вычисления, не является она и чисто логической, хотя, как вы убедитесь из комментариев, логика ее довольно сложна. Большая часть команд программы занята форматированием информации. Именно с этой точки зрения она представляет наибольший интерес.

Прежде чем приводить инструкцию, сделаем одно замечание. Датчик псевдослучайных чисел, работающий в программе, нуждается во вводе первого «случайного» числа. Автор предложил это делать так. Вводим текущую дату (число) и время (часы и минуты). А уже из комбинации этих трех чисел программа сама формирует первый член псевдослучайного ряда. Надеемся, что вы не будете эксплуатировать программу ровно в полночь, потому что комбинация 0 часов 00 минут является единственной недопустимой для ее работы.

Программа. 00. + 01.: 02.П9 03.0 04.П1 05.П2 06.П3 07.П4 08.ПП 09.24 10.4 11.П0 12.КИП↑ 13.ВП 14.4 15.КП↑ 16.FL0 17.12 18.ПП 19.24 20.1 21.С/П 22.БП 23.03 24.4 25.П0 26.9 27.П5 28.1 29.0 30.П8 31.ИП9 32.1 33.1 34.× 35.Фп 36. + 37.П9 38.КИП9 39.≥ 40.ИП9 41.— 42.П9 43.ИП6 44.× 45.1 46. + 47.КП5 48.FL0 49.31 50.КИПА 51.КИПВ 52.КИПС 53.КИПД 54.5 55.П0 56.ИПВ 57.ИПС 58.ИПД 59.ИПА 60.ПП 61.78 62.ИПС 63.ИПД 64.ИПА 65.ИПВ 66.ПП 67.78 68.ИПД 69.ИПС 70.ИПВ 71.ИПА 72.ПП 73.78 74.ИПС 75.ИПВ 76.ИПА 77.ИПД 78.П7 79.ФО 80.ИП8 81.× 82.+ 83.ИП8 84.× 85.+ 86.ИП8 87.× 88.ИП7 89.+ 90.КИП0 91.+ 92.КП↑ 93.В/0.

Инструкция.

1. Ввести программу.
2. Перейти в режим вычислений (F ABT).
3. Очистить командный счетчик (B/0).
4. Начальная засылка: число цветов П6.

5. Начальный ввод: число месяца ↑ часы ↑ минуты С/П.
6. Вывод: на индикаторе — 1.
ИП 1 — 1-я строка узора.
ИП 2 — 2-я строка узора.
ИП 3 — 3-я строка узора.
ИП 4 — 4-я строка узора.
7. Продолжение работы.
 - 7.1. С тем же количеством цветов: С/П.
 - 7.2. С новым количеством цветов: перейти к п. 4.

Комментарий.

1. По логической структуре эта «легкая» программа, пожалуй, самая сложная из помещенных в сборнике, поэтому, думаем, разобраться в ней будет трудновато. В программе многократно используется косвенная адресация почти всеми возможными способами, множество вложенных циклов, «цепочечные» обращения к подпрограммам, то есть из одной подпрограммы происходит обращение ко второй, а из нее к третьей... используется датчик псевдослучайных чисел, компоновка результатов для вывода.

Впрочем, начнем по порядку.

2. В пяти местах программы используется обращение к подпрограммам (адреса 08, 18, 60, 66, 72). Заметим, что употребление термина «подпрограмма» в рассматриваемой программе не совсем правомерно. Подпрограмм как отдельных частей программы, решавших самостоятельные задачи, здесь, собственно, нет. Правильнее было бы сказать, что используется команда «переход с возвратом». Это упрощает понимание работы программы. Проследим, как отрабатываются эти переходы. После первого обращения (команда 08) на адрес 24 в стеке возврата фиксируется адрес (10), на который нужно передать управление по команде В/0. Однако до В/0 мы добираемся не сразу. По дороге встречается новый переход (команда 60) на адрес 78. При этом адрес 10 заталкивается во 2-й регистр стека возврата, а в первом появляется новый — 62. На сей раз мы отрабатываем все команды до В/0.

Так как в стеке возврата записан адрес 62, то управление и адресуется на него. Он исчезает из стека, уступая свое место «верхнему» адресу — 10. Однако между командой 62 и командой В/0 опять встречается ПП (адрес 66). Вновь происходит замена адреса возврата в стеке и снова отрабатываются команды от 78 до 93. Еще раз происходит то же самое после команды по адресу 72. Фактически ситуация повторяется и после команды 78, но так как «подпрограмма»

начинается сразу за ней, то и ставить новый переход нет резона. Наконец, в этом случае мы доходим до команды В/0, когда в стеке записан адрес 10. Только теперь управление и передается на этот адрес. Аналогичная картина наблюдается и при следующем переходе (адрес 18). Здесь цикл обращений полностью повторится.

3. Команды косвенной адресации по адресам 12, 15, 47, 90 и 92 работают по своему прямому назначению, то есть считывают либо записывают содержимое регистров, номера которых хранятся в соответствующих командах. Команды же 38, 50—53 используются исключительно для выделения целой части чисел.

4. В командах 31—42 записан алгоритм получения псевдослучайного числа. Каждое новое число выражается через предыдущее по формуле: $\xi_k = \{11\xi_{k-1} + \pi\}$. Фигурными скобками обозначена дробная часть заключенного в них числа.

5. Команды 80—91 вместе с 12—14 формируют восьмизначное число из восьми различных цифр. Делается это так. Сначала четыре однозначных числа A , B , C и D компонуются в четырехзначное число $ABCD$ по формуле: $D + 10C + 100B + 1000A$. Затем это число умножается на 1000 (получаем $ABCD0000$), и к нему добавляется новое четырехзначное число $A'B'C'D'$, полученное описанным выше способом. Таким образом, на выходе появляется восьмизначное число: $ABCD A'B'C'D'$.

Рекомендуемая литература

Демидович Б. П., Марон И. А. Основы вычислительной математики. М., Наука, 1970.

Трохименко Я. К., Любич Ф. Д. Инженерные расчеты на микрокалькуляторах. Киев, Техника, 1980.

Трохименко Я. К., Любич Ф. Д. Инженерные расчеты на программируемых микрокалькуляторах. Киев, Техника, 1985.

Цветков А. Н., Епанечников В. А. Прикладные программы для микро-ЭВМ «Электроника Б3-34», «Электроника МК-56», «Электроника МК-54». М., Финансы и статистика, 1984.

Итак, книга закончена. И если наш читатель потратил время не только на чтение, но и с калькулятором в руках разобрал приведенные программы, можно надеяться, что он освоил основные принципы программирования на микрокалькуляторах.

Но можно ли сказать, что тем самым наш читатель приобщился к миру ЭВМ, миру, жить в котором нам предстоит в недалеком будущем? Ответить на этот вопрос непросто. И прежде, чем сделать это, попробуем ответить на другой вопрос: «Что такое программируемый калькулятор?»

Как только его не называют: карманный ЭВМ, микро-ЭВМ, персональный компьютер... С одной стороны, не в названии дело. Ведь говорят же в народе: «Хоть горшком назови, только в печку не ставь». Но с другой стороны, есть и такая поговорка: «Назвался груздем, полезай в кузов». А может ли называться наш друг калькулятор Электронной Вычислительной Машиной? Может ли он расцениваться как полноправный представитель большой семьи современных ЭВМ? Обладает ли он самыми существенными фамильными чертами своих старших родственников?

С одной стороны — да. Ведь для каждой ЭВМ характерно наличие двух компонент: процессора — так называют устройство, которое выполняет команды по преобразованию информации, и памяти — устройства, где эта информация хранится. И то и другое в программируемом калькуляторе есть.

С другой стороны, память современной ЭВМ приспособлена для хранения информации в произвольной форме, каждый элемент памяти может хранить как команду, так и ее операнды (числа в различной форме, коды символов, над которыми проводятся операции). В нашем же калькуляторе области памяти строго разделены: есть адресуемые регистры, где хранятся числа, и есть программируемая память, где находятся команды программы*. Интересно, что такой же принцип был реализован и на самых первых вычислительных машинах, что в значительной степени было тормозом их успешного развития. И только идея, предложенная Дж. фон Нейманом: рассматривать единую область памяти, трактуя каждый ее элемент либо как число (или символ), либо как команду, — привела к созданию современных ЭВМ. С этой точки зрения калькулятор никакой современной ЭВМ, будь она хоть «мини-микро», не родня.

Это вопросы структуры. Теперь о программировании.

Кардинальным моментом, превратившим программирование из элитарной профессии единиц в занятие миллионов, был переход от языков команд, которыми были оснащены ЭВМ первого поколения, к языкам

* Пусть читателя не вводит в заблуждение возможность записывать числа в программной памяти. Ведь там они записываются как команды последовательного вызова знаков числа в регистр X , и обращаться с ними, как с содержимым адресуемых регистров, мы не можем. Нельзя, к примеру, считать содержимое ячеек программной памяти с номером 05 или записать туда новое число, как это мы делаем с адресуемым регистром, скажем, Р5.

программирования высокого уровня, понятным для всех современных ЭВМ, — языкам, максимально приближенным к общепринятым языкам математических выкладок.

Калькулятор же понимает только язык символов, нанесенных на его клавиши, язык элементарных команд. Поэтому если он и родственник «настоящим» ЭВМ, то не в большей степени, чем домашняя кошка усурийскому тигру.

Так что лучше, во всяком случае во избежание неизвестной терминологической путаницы, называть программируемый калькулятор ... программируемым калькулятором.

Значит, калькулятор — не ЭВМ? Ну и что с того? Известный физик Р. Фейнман в свое время дал такое определение науки, что под него не подпадала математика. Когда ему указали на это, учёный ответил: «...не все, что наука, уже обязательно плохо. Любовь, например, тоже не наука. Словом, когда какую-то вещь называют не наукой, это не значит, что с нею что-то неладно: просто не наука она, и все». Очевидно, следуя за Фейнманом, можно сказать: «Не все, что не ЭВМ, плохо». И роль и значение программируемого калькулятора отнюдь не уменьшается, если не называть его ЭВМ. Просто калькулятор — одно из устройств, призванных облегчить рутинную вычислительную работу. Устройство, продолжающее ряд, начатый счетными досками-абаками, конторскими счетами, механическими арифмометрами и логарифмической линейкой, но несравненно более быстродействующее, удобное и значительно более простое в обращении.

Поэтому если вы овладеете микрокалькулятором не с целью приобщения к миру ЭВМ, а как самостоятельный вычислительный устройством, то и в этом случае игра стоит свеч. В лице калькулятора человек получает в свое распоряжение средство, которое значительно облегчает его труд в тех случаях, когда под рукой более мощных вычислительных средств нет.

Что же касается приобщения к миру ЭВМ, то здесь опыт программирования на микрокалькуляторе, если понимать под ним процесс перевода алгоритма на язык калькуляторных команд, пригодится не в такой уж значительной мере. Ведь то, на что обращают внимание «калькуляторные» программисты (оптимальная запись формул с использованием свойств стека, всевозможные ухищрения по выделению целой и дробной частей числа, запутанные передачи управления с помощью косвенной адресации), — все это при программировании на языках высокого уровня не нужно. Там те же результаты достигаются совершенно простыми способами.

Однако само ощущение, что можно нажимать на клавиши, не опасаясь, что при этом машина поломается, привычка строго детерминировать свои действия, пользоваться ограниченным набором средств для достижения цели, записывать алгоритм решения задачи в виде блок-схемы, владение принципами организации ветвлений и циклов, использование аппарата подпрограмм — этот багаж, безусловно, можно брать с собой, пересаживаясь за пульт любой ЭВМ. Он и тут не утратит своей ценности и станет отличным подспорьем.

И наконец, соображения сугубо практические. Спрос на вычислительные устройства постоянно растет. Конечно, растет и выпуск персональных ЭВМ, призванных в идеале занять место на каждом письменном столе на работе и дома, на каждой школьной парте. Но парт и столов у нас столь много, что вряд ли в течение ближайших лет все они будут оснащены персональными компьютерами. Да и цены этих компьютеров в десятки раз выше, чем у программируемых калькуляторов. И чем

ждущих времен, готовясь к плаванию в открытом море, не покидая обжитой комнаты, лучше уж отрепетировать хотя бы некоторые приемы плавания пусть в небольшом бассейне, но наполненном водой.

Итак, попробуем подвести итоги.

Прежде всего скажем, что для начального обучения программированию компьютеры, конечно, предпочтительнее программируемых калькуляторов. Тексты и движущиеся картинки, выводимые на экран дисплея, гораздо сильнее привлекут внимание ребенка, да и взрослого, чем мельчание сухих цифр. Но там, где компьютеров нет, начать приобщение к вычислительной технике лучше с калькулятора, чем, скажем, со счетов.

Что же касается проведения оперативных расчетов, то калькулятор для этого значительно более удобен, чем арифмометр, логарифмическая линейка и том справочных таблиц, вместе взятые.

Поэтому осваивайте программируемые калькуляторы! Тем самым вы получите возможность автоматизировать часть своего умственного труда уже сегодня, а не в будущем, хотя бы и близком.

ПРИЛОЖЕНИЕ

ЯЗЫК ДЛЯ ЗАПИСИ АЛГОРИТМОВ

В этой книге читатель столкнулся с двумя способами записи алгоритмов: в виде блок-схем и на специальном языке. Что касается блок-схем, то целесообразность их использования очевидна: наглядно представлены последовательность действий, ветвления, циклы. Блок-схему легко проанализировать, проследить за замыканием ветвей, избежать дублирования вычислений и т. д. Возникает вопрос: а зачем же тогда нужен еще и специальный язык?

Дело в том, что при всей своей наглядности блок-схема не позволяет оценить некоторые аспекты, важные именно для машинной реализации алгоритмов,— такие, например, как длина программы, объем памяти, потребной для хранения переменных, типы этих переменных. Кроме того, слишком большое пространство, занимаемое блок-схемами, является серьезным препятствием для их публикации в печатных изданиях.

Есть и еще один фактор — слишком велика дистанция от блок-схемы до программы. Перевод блок-схемы в программу чем-то напоминает «рассказ по картинке» и мало способствует созданию привычки выражать свои мысли с помощью ограниченного набора средств — программистских конструкций, которые перевелись бы в программу для ЭВМ почти автоматически.

Казалось бы, с учетом таких соображений блок-схемам следовало бы предпочесть какое-то другое средство для записи программ, более близкое к системе вычислительных действий, выполняемых электронной вычислительной машиной. Но этот путь сулит свои сложности и противоречия. Чем «понятнее» машине язык, на котором пишутся программы, тем он сложнее для человека. И наоборот: чем более гибок язык программирования, чем более он приближен к привычному для человека языку математических выкладок, тем сложнее транслятор — специальная программа, переводящая вводимые в машину тексты программы

на языки машинных команд. Разрешить это противоречие вряд ли возможно. Отсюда следует важный вывод: универсальная методология программирования не может ориентироваться на конкретный язык.

Для этих целей необходим специальный язык, или лексикон, как его называет академик А. П. Ершов. Такой язык создан. Он включил в себя лучшие достижения существующих языков программирования, в первую очередь таких, как Алгол-60 и Паскаль. Но в отличие от конкретных языков программирования алгоритмический язык для записи алгоритмов работы с величинами не замкнут. Он открыт. Это означает, что нет никаких ограничений для использования всевозможных символов, не существует жестких ограничений на запись имен (или идентификаторов) переменных. Вместе с тем алгоритмический язык, как это свойственно всем языкам программирования, обладает жесткими требованиями относительно записи и интерпретации программистских конструкций.

Подобно тому как при составлении блок-схемы отдельные действия записываются в фигурах определенных очертаний, так при записи алгоритма на алгоритмическом языке роль своеобразной оправы, обрамляющей описание употребляемых величин и совершаемых над ними действий, играют определенные слова, называемые служебными. Их немного, они сокращаются до двух-трехбуквенных сочетаний и выделяются жирным шрифтом.

Служебное слово алг (алгоритм) открывает запись всякого алгоритма. Вслед за ним пишется название алгоритма. Далее, в круглых скобках, перечисляются имена (обозначения) тех величин, которые берутся в качестве исходных данных (аргументы), и тех, которые получаются в итоге (результаты). Для каждой из этих величин указывается ее характер: если это целое число, то перед соответствующим обозначением пишется цел., если вещественное —вещ. и т. п. Если упомянутся несколько однородных величин, обозначаемых одним именем с разными индексами, то перед ним еще добавляется слово таб (таблица), а за ним, в квадратных скобках, — начальное и конечное значение индекса, разделяемые дветочием. При записи алгоритма могут использоваться также литерные переменные, предваряемые в перечне словом лит. Их значения представляют собой слова и даже тексты. Например, результат решения квадратного уравнения можно вывести в виде корней, если они существуют, либо в виде литерной переменной КОРНЕЙ НЕТ, если вещественные корни отсутствуют.

Величины, используемые при записи алгоритма, можно обозначать не только буквами, но и словами. Например, площади прямоугольника можно придать имя площадь, а формулу для ее вычисления записать в виде основание×высота.

Приведя название алгоритма и перечень аргументов и результатов строчкой ниже, с некоторым отступом, после слова арг указывают все аргументы, а далее, после слова рез, — результаты. Эти две строки образуют заголовок алгоритма.

Заголовок алгоритма, включающий его название, перечень аргументов и результатов, обозначения типов переменных, используемых в алгоритме, имеет большое значение для понимания его сути и исправления возможных ошибок, допущенных при его записи.

Дело в том, что переменные различных типов интерпретируются в ЭВМ по-разному. Например, переменная, определенная как целая (цел.), занимает меньше памяти, чем вещественная, то есть нецелая (вещ.). Да и обрабатываются целые переменные быстрее. Однако область их допустимых значений значительно меньше, чем у вещественных пере-

менных. (В записи переменных целого типа участвуют только цифры, а при записи вещественных переменных можно использовать запятые для отделения целой части числа от дробной и экспоненциальные представления с порядками, варьируемыми в широких пределах.)

После заголовка алгоритма без отступа пишут слово нач (начало) и вслед за ним — перечень промежуточных переменных с указанием их типа.

Эта информация преследует те же цели, что и описание переменных в заголовке. При этом мы получаем возможность оценить объем памяти, нужной для хранения переменных. Это особенно важно для таких структур данных, как массивы или таблицы. Чаще всего именно такие структуры определяют львиную долю памяти ЭВМ при реализации алгоритма.

Еще ниже, с отступом, строчка за строчкой пишутся команды, то есть предписания выполнить отдельные законченные действия. Простейшим примером команды является присваивание. Оно записывается в виде $x := E$ (реже $E \rightarrow x$). Здесь x — переменная величина, E — любое алгебраическое выражение, в которое входят константы, переменные, знаки операций и скобки. Величина x называется получателем, E — источником. В выражение E должны входить только те переменные, которые используются в алгоритме, и только такие операции, которые могут быть совершены исполнителем алгоритма. Чтобы выполнить команду присваивания, нужно взять текущие значения переменных величин, входящих в источник, и подставить их на места своих имен. Вычисленное значение присваивается переменной x .

Команды можно соединять друг с другом в серии, разделяя точкой с запятой. Выполняются они в порядке написания. Существуют также составные команды, состоящие из некоторого условия и более простых команд, та или иная из которых выполняется в зависимости от условия. Наиболее часто встречающиеся составные команды — это ветвление и цикл. О них подробно говорится в главе 5 первой части книги. Составную команду можно размещать в одной строке, но для большей наглядности лучше написать в отдельной строке условие, а ниже, с отступом, также в отдельных строчеках — простые команды, входящие в составную.

Запись алгоритма завершается служебным словом кон (конец), помещаемым без отступа.

Более подробно об алгоритмическом языке можно прочитать в статье академика А. П. Ершова «Алгоритмический язык», опубликованной в журнале «Наука и жизнь» (№ 11 за 1985 г. и № 1 за 1986 г.).

Отметим в заключение, что существование алгоритмического языка ни в коей мере не отменяет использования блок-схем. Они особенно полезны при разработке крупных программных комплексов. Каждый блок такой схемы — это большой самостоятельный алгоритм, и для понимания их совместной работы блок-схема является самым наглядным средством.

СОДЕРЖАНИЕ

Предисловие	3
Введение	5
Часть I. Школа программирования	7
1. Знакомство с микрокалькулятором	7
2. Команды микрокалькулятора	17
3. Этапы решения задачи	36
4. Ввод и вывод информации	54
5. Ветви, циклы, подпрограммы	63
6. Погрешности вычислений	86
7. Культура и искусство программирования	105
8. Сфера применения программируемых микрокалькуляторов	110
Часть II. Практика программирования	111
Калькуляция на калькуляторе	111
У токарного станка	113
Проверка с микрокалькулятором	115
На сварке труб	119
Если нужно построить график	122
Спорт + медицина == калькулятор	126
По алгоритму Жюля Верна	129
Сколько весит заготовка	131
Экономия на статистике	136
Две окружности	141
Простые числа	145
Диагноз ставит микрокалькулятор	153
Калькулятор считает время	156
В помощь сталевару	162
Электронный топограф	177
Для тех, кто вяжет	180
Рекомендуемая литература	185
Послесловие	186
Приложение. Язык для записи алгоритмов	188

Юрий Васильевич ПУХНАЧЕВ

Игорь Данилович ДАНИЛОВ

МИКРОКАЛЬКУЛЯТОРЫ ДЛЯ ВСЕХ

Главный отраслевой редактор А. Нелюбов

Редактор Н. Феоктистова

Младший редактор Н. Калякина

Художественный редактор М. Бабичева

Технический редактор А. Красавина

Корректор В. Каючкина

ИБ № 7811

Сдано в набор 08.01.86. Подписано к печати 11.06.86. А01131.
Формат бумаги 84×108¹/₃₂. Бумага тип. № 1. Гарнитура литература-
тическая. Печать высокая. Усл. печ. л. 10,08. Усл. кр.-отт. 10,40.
Уч.-изд. л. 10,13. Тираж 80 000 экз. Заказ 6-689. Цена 55 коп.
Издательство «Знание». 101835, ГСП, Москва, Центр, проезд
Серова, д. 4. Индекс заказа 866712.

Отпечатано с матриц МПО «1-ой Образцовой типографии»
им. А. Л. Жданова на Головном предприятии республиканского
производственного объединения «Полиграфкнига».

252057, Киев — 57, ул. Довженко, 3.

**Сфераы применения
налькуляторов :**

- рассчи тываем
плавину стали
- если нужно построить
график
- электронный топограф
- советчик врача
- налькулятор в спорте
- прогноз погоды
по налькулятору
- для тех, кто вяжет

ЗНАНИЕ