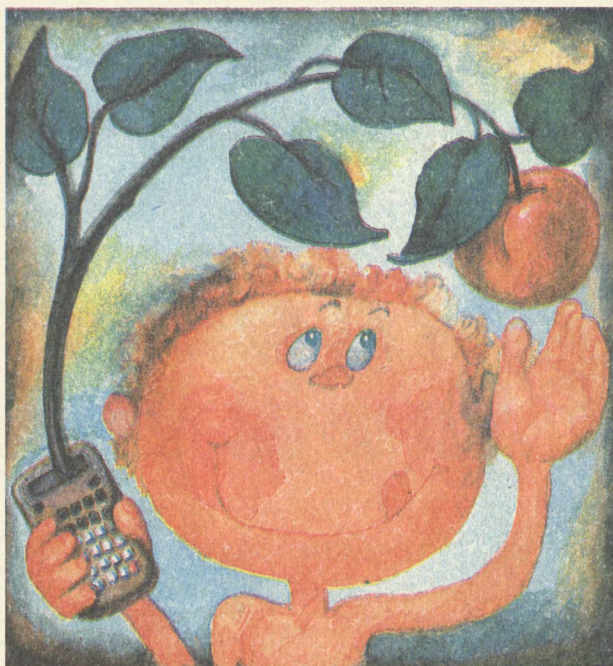




БИБЛИОТЕЧКА • КВАНТ •
выпуск 55

И. Д. ДАНИЛОВ

СЕКРЕТЫ ПРОГРАММИРУЕМОГО МИКРОКАЛЬКУЛЯТОРА



12345678-10



Электроника
БЗ-34



F	$x < 0$ ↔	$x = 0$ ↔	$x > 0$ ↔	$x \neq 0$ ↔
К	L0	L1	L2	L3
sin	cos	tg	x^2	π
arc sin	arccos	arc tg	$1/x$	$\sqrt{}$
e^x	lg	ln	x^y	Bx
10^x	↺	ABT	ПРГ	CF
НОП	A	B	C	D



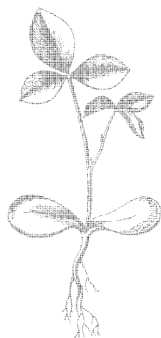
БИБЛИОТЕЧКА • КВАНТ •
выпуск 55

И.Д. ДАНИЛОВ

СЕКРЕТЫ ПРОГРАММИРУЕМОГО МИКРОКАЛЬКУЛЯТОРА



МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1986



Scan AAW

ББК 22.18
Д18
УДК 519.85 (023)

РЕДАКЦИОННАЯ КОЛЛЕГИЯ:

Академик **Ю. А. Осипьян** (председатель), академик **А. Н. Колмогоров** (заместитель председателя), профессор **Л. Г. Асламазов** (заместитель председателя), кандидат физ.-мат. наук **А. И. Буздин** (ученый секретарь), член-корреспондент АН СССР **А. А. Абрикосов**, академик **А. С. Боровик-Романов**, академик **Б. К. Вайнштейн**, заслуженный учитель РСФСР **Б. В. Воздвиженский**, академик **В. Л. Гинзбург**, академик **Ю. В. Гуляев**, академик **А. П. Ерипов**, профессор **С. П. Капица**, академик **А. Б. Мигдал**, академик **С. П. Новиков**, академик АПН СССР **В. Г. Разумовский**, академик **Р. З. Сагдеев**, профессор **Я. А. Смородинский**, академик **С. Л. Соболев**, член-корреспондент АН СССР **Д. К. Фаддеев**

Рецензент

член-корреспондент АН СССР *Л. Н. Королев*

Данилов И. Д.

Д18 Секреты программируемого микрокалькулятора. — М.: Наука. Гл. ред. физ.-мат. лит., 1986. — 160 с. — (Б-чка «Квант». Вып. 55.) — 30 к.

30 к. 300 000 экз.

Описываются особенности и принципы работы на самых миниатюрных ЭВМ — программируемых микрокалькуляторах (ПМК). Хотя возможностей у ПМК меньше, чем у больших ЭВМ, основное достоинство вычислительных машин — возможность работать по введенной программе — есть и у них. Какие задачи и как их решать на микрокалькуляторе, как быстрее и лучше написать программу — ответы на эти и целый ряд других вопросов найдут читатели в этой книге.

Для школьников, учащихся ПТУ и техникумов, студентов, преподавателей.

Д $\frac{1702070000 - 166}{053(02)-86}$ 157-86

ББК 22.18

© Издательство «Наука».
Главная редакция
физико-математической
литературы, 1986

Нужно ли уметь программировать? Сегодня, возможно, еще найдутся люди, думающие, что это не обязательно. Но очень скоро человек, не умеющий программировать, станет такой же «белой вороной», как неграмотный. Не случайно академик А. П. Ершов называет умение программировать второй грамотностью.

Владение навыками работы на ЭВМ не только дает возможность использовать их для решения различных задач, но и вырабатывает определенный, «программистский» стиль мышления: умение четко и лаконично формулировать свои мысли, правильно ставить задачу и находить оптимальные пути ее решения, быстро ориентироваться в потоке информации и, наконец, привычно обращаться к ЭВМ для решения задач из любой области. Эти навыки нужны всем без исключения.

Ясно, что лучше осваивать программирование, постоянно общаясь с машиной. И если раньше это вызывало затруднения (машин было маловато), то теперь положение меняется. Мы на пороге нового исторического процесса. Машины начинают вторгаться в сферу производства и быт. Они уже прочно заняли место в лабораториях и на предприятиях, на письменных столах, в портфелях, ранцах. Могут уместиться и в кармане. В последнем случае речь идет о самом распространенном сегодня типе вычислительных устройств — программируемых микрокалькуляторах (ПМК).

Эти изящные маленькие коробочки с кнопками и миниатюрным экраном могут быть полезны каждому. Инженеру и технику — для решения профессиональных задач, студенту — при проведении курсовых и лабораторных работ, школьнику и учащемуся ПТУ — при решении задач по математике, физике, химии. Они помогут рассчитать семейный бюджет и обучить ребенка устному счету. Микрокалькулятор даст возможность скоротать досуг, заменив партнера в увлекательной игре. Да и само ощущение комфорта при работе на своей собствен-

ной машине, сидеть за которой можно когда угодно и сколько угодно, с которой можно постоянно экспериментировать, открывая все новые ее тайны, даст немало приятных минут.

Программируемый микрокалькулятор — самая малая из всех существующих ЭВМ, но, тем не менее, он заслуживает вполне серьезного к себе отношения. Хотя возможности его меньше, чем у больших ЭВМ и персональных компьютеров, основное достоинство вычислительных машин — возможность работать по введенной программе — есть и у него. А так как микрокалькулятор — машина портативная, а главное — недорогая, то доступность его является даже преимуществом перед стационарными ЭВМ.

В нашей стране выпускаются несколько типов программируемых микрокалькуляторов. Общее количество их превышает несколько сотен тысяч и постоянно растет. Это ПМК «Электроника БЗ-34», «Электроника МК-54», «Электроника МК-56», «Электроника МК-52» и «Электроника МК-61».

Первые три типа микрокалькуляторов различаются лишь оформлением, системы команд у них, а отсюда и их возможности, совершенно одинаковы. Почти та же система команд и у последних двух микрокалькуляторов. Почти, потому что она несколько шире. Язык «Электроники МК-52» и «Электроники МК-61» включает в себя все команды калькуляторов первой группы и еще ряд дополнительных, увеличивающих возможности этих ПМК. Самый распространенный на сегодня тип ПМК — «Электроника БЗ-34». Потому-то она и выбрана в качестве эталона для рассказа о программировании на этих машинах. Естественно, все сказанное в книге относится и к другим упомянутым выше калькуляторам.

Чтобы читать эту книгу, не требуется никаких предварительных знаний ни о программировании, ни об ЭВМ. Что касается математики, то вполне достаточно знаний, приобретенных в средней школе.

Материал расположен в книге так, чтобы читатель постепенно не только расширял, но и углублял свои знания. Сначала описываются приемы работы с микрокалькулятором в непрограммном, ручном режиме, затем даются краткие сведения о его структуре и логике работы.

Отдельные разделы из этой книги публиковались в журналах «Техника — молодежи» и «Наука и жизнь».

Автор пользуется случаем, чтобы выразить благодарность А. Б. Бойко, предложившему несколько интересных программ, и особую признательность — Ю. В. Пухначеву, взявшему на себя труд прочесть рукопись и сделавшему ряд ценных замечаний, учтенных при создании настоящей книги.

1. ПЕРВОЕ ЗНАКОМСТВО

Микрокалькулятор — перед вами. Тридцать разноцветных клавиш выстроились стройными рядами по пять в линию. На каждой клавише — обозначение. Над клавишами — тоже, а в нижнем ряду — и под клавишами.

Некоторые обозначения понятны сразу. Это цифры, символ «» для отделения целой части числа от дробной, знаки арифметических операций, названия элементарных функций. Другие обозначения менее очевидны, о них мы еще будем говорить.

Над рядами клавиш — два переключателя. Левый включает калькулятор, а правый задает режим работы с тригонометрическими функциями: «Р» означает радианы, «Г» — градусы. На вычислении остальных функций положение этого переключателя не отражается.

Передвигаем левый переключатель вправо. На индикаторе загорается цифра 0. Микрокалькулятор готов работать, повинаясь командам, которые мы станем задавать ему, нажимая на клавиши.

Надо сказать, что сигналы от нажимаемых клавиш по-разному воспринимаются микрокалькулятором в зависимости от того, в каком состоянии он находится. Таких состояний два. В первом выполняются команды, либо отдаваемые непосредственно нажатием на клавиши, либо записанные в память микрокалькулятора в виде программы. Программы вводятся в нашу миниатюрную ЭВМ нажатием на те же клавиши, когда она находится во втором состоянии.

Включая микрокалькулятор, мы автоматически устанавливаем его в первое состояние. Память его при этом совершенно чиста. Так что сразу после включения мы можем вести вычисления на нем лишь в непрограммном режиме. Ими мы и займемся: это поможет понять принципы работы машины, назначение клавиш и значительно облегчит переход к программированию.

Для начала рекомендуем освоить клавиатуру. Нажимаем на цифровые клавиши. На экране загораются те же цифры, что обозначены на клавишах: 1, 2, ..., 7, 8. Нажимаем дальше. Девятая цифра не появляется. Калькулятор рассчитан на работу не более чем с восьмиразрядными числами, и если восемь цифр числа уже введены, дальнейший ввод блокируется, нажатие на клавиши уже не воспринимается машиной.

Чтобы занести дробное число, нажимаем после цифр целой части клавишу «.» и затем вводим дробную часть. Интересно, что сам символ «.» располагается в той же позиции, что и последняя введенная перед ним цифра. Так что при употреблении десятичной запятой общее число позиций для записи значащих цифр не меняется. Ради тренировки введем в наш калькулятор числа 3,1415926 (отношение длины окружности к диаметру), затем 2,7182818 (основание натуральных логарифмов).

Нетрудно сообразить, что возможности такой записи чисел ограничены. Нельзя, к примеру, таким образом ввести в микрокалькулятор заряд электрона, измеренный в кулонах; он выражается десятичной дробью с девятнадцатью нулями после запятой. И расстояние от Земли до Солнца таким способом в калькулятор не ввести: выраженное в километрах, оно содержит девять знаков, а в метрах — еще больше.

Подобные затруднения преодолимы, если записывать числа в так называемом экспоненциальном виде $a \cdot 10^b$, где число a должно начинаться с ненулевой цифры, после которой стоит запятая. Число a называется *мантиссой*, b — *порядком*.

Вот как, например, записывается в экспоненциальной форме заряд электрона: $1,6021892 \cdot 10^{-19}$ Кл. А вот как — среднее расстояние от Земли до Солнца: $1,497 \cdot 10^8$ км.

Чтобы ввести в микрокалькулятор число, записанное в экспоненциальном виде, надо сначала набрать мантиссу. Если число отрицательное, вслед за этим следует нажать клавишу «/ — /». Затем, чтобы указать порядок числа, надо нажать клавишу «ВП» (Ввод Порядка). В правой части индикатора тотчас появятся два нуля. Теперь нажатие на любую цифровую клавишу приводит к ее отображению в правом углу индикатора. Набираем порядок, и если он отрицательный, то нажимаем затем клавишу «/ — /».

Введите в микрокалькулятор значение радиуса Вселенной, равное по современным оценкам астрономов приблизительно $4 \cdot 10^{20}$ м. Для этого следует нажать клавиши «4», «ВП», «2», «0». На индикаторе читаем: 4 20. Введем теперь радиус электрона: $2,817938 \cdot 10^{-18}$ м. Нажимаем на клавиши «2», «.», «8», «1», «7», «9», «3», «8», «ВП», «1», «8», «/ — /» и читаем на индикаторе: 2,817938 —18.

Судя по последнему примеру, диапазон чисел, доступных нашему микрокалькулятору, огромен. Максимальное из них, как легко сообразить, $9,9999999 \cdot 10^{99}$, минимальное по абсолютной величине — 10^{-99} .

Если при задании порядка числа попытаться ввести более двух цифр, то каждая новая из вводимых будет занимать правую позицию, бывшая там ранее цифра сместится влево, а стоявшая слева исчезнет.

Если вы ошиблись при вводе числа — не беда. Клавиша «Сх» очищает индикатор. Но дело тут не только в том, что гаснут горевшие на нем цифры. Любое число, введенное в калькулятор с клавиатуры, помещается в ячейку памяти, называемую *регистром* X. Содержимое этого регистра отображается на индикаторе. Действие клавиши «Сх» как раз в том и заключается, что она очищает регистр X (отсюда и ее название: Стереть X). На индикаторе отражается лишь результат стирания, то есть цифра 0. После этого ввод можно повторить или ввести другое число.

Теперь перейдем к вычислениям. Почти наверняка каждый владелец калькулятора захочет убедиться, что 2×2 она вычисляет правильно. Тем более, что результат ему известен. Вводим число 2, нажимаем на клавишу « \times ». На экране появляется 0. Это что, уже ошибка? Нет. Просто мы еще не сказали, что порядок вычислений на нашем калькуляторе отличается от общепринятого.

Записывая любую арифметическую операцию, мы ставим знак ее между участниками операции (их называют *операндами*). Читая любое арифметическое выражение, мы всегда помним о приоритете операций. Сначала действия в скобках, потом умножение и деление, потом сложение и вычитание. Таким образом удастся однозначно толковать любую запись. К примеру, вычисляя $a \times b + c \times d$, мы сначала выполняем два умножения, а затем складываем полученные произведения. Если нужно изменить порядок вычислений, то ставят скобки, например так: $a \times (b + c \times d)$. Все просто и ясно.

Однако способ этот хоть и удобен для нас, но не экономичен для микрокалькулятора. Ведь надо где-то хранить промежуточные результаты, запоминать знаки предыдущих операций, проверять их приоритеты. От этих недостатков свободен другой метод записи арифметических выражений, предложенный польским логиком Я. Лукасевичем и получивший название *обратной бесскобочной* или *польской* записи. Все операции при такой записи равноценны, а знаки их ставятся не между операндами, а после них, то есть не $a \times b$, а $ab \times$. Это избавляет от необходимости использовать скобки и проверять приори-

теты. При польской записи первое из приведенных выражений запишется так: $ab \times cd \times +$, а второе — так: $cd \times b + a \times$.

Приступим снова к умножению 2×2 . Опять вводим 2, затем еще раз 2. На индикаторе — число 22. Мы же хотели занести в калькулятор вовсе не его, а две двойки, два отдельных числа.

Такова любая арифметическая операция: не только умножение, но и сложение, и вычитание, и деление. Каждая требует для своего выполнения двух исходных чисел и потому называется *двуместной*.

Для размещения участников двуместной операции требуются два различных регистра. Микрокалькулятор предоставляет такую возможность. Регистр X, куда вводятся числа с клавиатуры, — это лишь один из четырех взаимосвязанных регистров, обозначаемых буквами X, Y, Z, T и образующих вместе так называемый *стек* (английское слово, означающее «стог, скирда, поленища»). В регистрах X и Y должны располагаться числа, участвующие в любой двуместной операции.

Делается это так. Сначала одно из этих чисел набирается на клавиатуре и тем самым записывается в регистр X. Потом нажимается клавиша «↑». Содержимое регистра X при этом копируется в регистр Y. Если теперь набрать на клавиатуре новое число, оно запишется опять-таки в регистр X, стирая прежнее записанное в нем число. После этого можно нажимать клавишу требуемой двуместной операции.

Узнав об этом, повторим вычисление 2×2 . Нажимаем клавишу «2», затем «↑», затем «2», затем « \times » (или в более короткой записи: $2 \uparrow \times$). На индикаторе 4. Все верно: дважды два — четыре.

Но не лишним ли был второй нажим клавиши «2» в нашем примере? Ведь клавиша «↑» не стирает содержимое регистра X, копируя его в регистр Y. Повторим умножение: $2 \uparrow \times$. На индикаторе вновь 4.

Результат всякой операции записывается в регистр X — потому он и виден на индикаторе. (В дальнейшем для краткости мы иногда будем писать не «регистр X, регистр Y», а RX, RY.)

Порядок расположения чисел в регистрах X и Y безразличен, если числа складываются или перемножаются. Порядок их расположения при вычислении числа в некоторой степени, например в виде x^y , задается самой записью операции: в RX — основание, в RY — показатель степени. А вот на случай вычитания и деления порядок расположения операндов следует запомнить.

При вычитании: в регистре X — вычитаемое, в регистре Y — уменьшаемое. Для лучшего запоминания советуем представлять результат операции вычитания в виде $Y - X$.

При делении: в регистре X — делитель, в регистре Y — делимое. Советуем представлять результат деления в виде Y/X .

Попробуйте выполнить на микрокалькуляторе вычитание $42 - 17$ и деление $54 : 18$ и получить соответственно 25 и 3.

Рассказывая об арифметических операциях, мы упоминали только о регистрах X и Y. Между тем и при вводе чисел для выполнения любой операции на микрокалькуляторе, и при ее совершении происходит движение чисел по всему стеку; это движение затрагивает еще один регистр, примыкающий к стеку и обозначаемый X1.

После ввода в регистр X некоторого числа мы нажимаем клавишу «↑»; при этом содержимое регистра X копируется в регистре Y, прежнее содержимое RY смещается в RZ, содержимое RZ — в RT, а старое содержимое регистра T при этом пропадает.

Когда над числами, находящимися в регистрах X и Y, совершается какая-либо арифметическая операция и ее результат записывается в регистр X, то прежнее содержимое регистра X пересылается в регистр X1, содержимое регистра Z смещается в RY, содержимое RT — в RZ, оставляя после себя в регистре T свою копию. То, что было в регистре Y, при этом пропадает.

Когда содержимое регистра X используется для вычисления какой-либо функции, результат записывается в тот же регистр X. Прежнее его содержимое пересылается в регистр X1. Содержимое других регистров не меняется.

Заметим, что названия всех функций, вычисляемых на микрокалькуляторе, написаны над клавишами. Чтобы совершить любую «надклавишную» операцию, нужно предварительно нажать кнопку «F», а уже потом соответствующую клавишу. Заметим также, что вычисление любой функции называется *одноместной операцией*, поскольку требует для своего выполнения одного исходного числа.

Если число, находящееся в регистре X, не введено туда с клавиатуры, а является результатом какой-либо операции (не только арифметической, но любой, выполненной по какой угодно команде, кроме команды «Сх» *)), то ввод нового числа в регистр X смещает его содержимое в регистр Y, содержимое RY — в RZ, содержимое RZ — в RT, а содержимое RT стирается.

Это еще не все возможные перемещения чисел по стеку. Допустим, вам захотелось использовать содержимое регистра X1. Нажмите клавиши «F» и «Vх» (Вверх). Число из регистра

*) Это относится к работе клавиши «Сх» только в режиме ручных вычислений.

X1 переместится в RX, а числа, находившиеся до этого в регистрах стека, сместятся точно так же, как при вводе нового числа: из RX — в RY, из RY — в RZ, из RZ — в RT, из RT — пропадет.

Можно заставить числа двигаться по стеку и в обратном направлении, нажав клавиши «F» и «O» (на клавиатуре микрокалькулятора этот символ образован двумя полукруглыми стрелками, замкнутыми друг на друга). Тогда число из RT перейдет в RZ, из RZ — в RY, из RY — в RX, из RX — в RT и RX1.

Можно переставить числа, находящиеся в RX и RY, нажав клавишу «XY». Содержимое регистра X при этом перейдет не только в RY, но и в RX1.

Диаграммы, иллюстрирующие движение чисел по стеку при выполнении различных операций, помещены на четвертой странице обложки. Мы вернемся к ним еще раз в разделе 4. Здесь же ради тренировки предлагаем читателю проверить, что вычисления по уже встречавшейся нам формуле $a \times (b + c \times d)$ можно провести на нашем микрокалькуляторе иначе, нежели было предложено ранее — не только в последовательности $c \uparrow d \times b + a \times$, но и в последовательности: $a \uparrow b \uparrow c \uparrow d \times + \times$. Чтобы убедиться в этом, можно построить таблицу, в которой показывается заполнение регистров стека после выполнения каждой операции. Числа a, b, c, d после их ввода располагаются в стеке так: a — в RT, b — в RZ, c — в RY, d — в RX. Что происходит далее, видно из таблицы (табл. 1).

Таблица 1

Регистры	Операции			
		\times	$+$	\times
T	a	a	a	a
Z	b	a	a	a
Y	c	b	a	a
X	d	cd	$b + cd$	$a(b + cd)$
XI	—	d	cd	$b + cd$

Попробуем теперь применить полученные в этом разделе знания при решении небольших задач.

Задача 1. Расчет платы за электроэнергию и газ. Такую задачу каждый квартиросъемщик решает ежемесячно. Плата начисляется по формуле $D = (P_n - P_{ct}) \times 0,04 + K \times 0,42$, где

$P_{\text{ст}}$ и $P_{\text{н}}$ — старые и новые показания счетчика в киловатт-часах, K — число людей, проживающих в квартире; D — плата в рублях. Пусть $P_{\text{ст}} = 16478$, $P_{\text{н}} = 16592$ и $K = 4$.

Включаем микрокалькулятор, вводим величину $P_{\text{н}}$. Нажимаем клавишу « \uparrow », и затем вводим величину $P_{\text{ст}}$. Обе величины размещаются в стеке так, как это требуется для вычитания. Нажимаем клавишу « $-$ ». Разность $P_{\text{н}} - P_{\text{ст}} = 114$ — в регистре X и на индикаторе. Ее надо умножить на $0,04$, а для этого второй сомножитель надо ввести в регистр X . Нажимаем соответствующие цифровые клавиши. Число 114 , бывшее прежде в регистре X , сместится, как результат предыдущей операции, в регистр Y . Нажав клавишу « \times », вычисляем произведение $(P_{\text{н}} - P_{\text{ст}}) \times 0,04 = 114 \times 0,04 = 4,56$. Вводим в стек сомножители следующего произведения. Для этого нажимаем клавиши « 4 », « \uparrow », « 0 », « $,$ », « 4 », « 2 ». Находившееся в регистре X число $4,56$ при первом вводе сместится в регистр Y , при втором — в регистр Z . Когда, нажав клавишу « \times », мы получим в регистре X произведение $4 \cdot 0,42 = 1,68$, содержимое регистра Z перейдет в регистр Y . Оба произведения $(P_{\text{н}} - P_{\text{ст}}) \times 0,04$ и $K \times 0,42$, таким образом, размещаются в стеке так, как это требуется для их сложения по формуле расчета. Нажимаем на клавишу « $+$ » и читаем окончательный результат: $6,24$.

Задача 2. Перед нами — та же расчетная книжка. Подсчитаем теперь среднюю плату за прошлый год.

Вероятно, теперь вы сами сможете предложить порядок действий. Набираем плату за январь, нажимаем клавишу « \uparrow », затем 11 раз повторяем две операции: ввод платы за следующий месяц, « $+$ ». Набираем потом 12 и нажимаем клавишу « \div » (знак деления). На экране виден результат. Предоставляем вам возможность провести расчет самостоятельно, используя свою расчетную книжку. Заодно получите «информацию к размышлению»: когда расход энергии был больше среднего и насколько. Возможно, проанализировав эти цифры, вы найдете способ сокращения затрат. Это будет первый экономический эффект от использования калькулятора.

Задача 3. Вычислить длину стороны c произвольного треугольника, зная длины двух других его сторон, a и b , и угол φ между ними.

Из школьного курса тригонометрии известно, что эта зависимость описывается формулой

$$c = \sqrt{a^2 + b^2 - 2ab \cos \varphi}.$$

Можно решить эту задачу, что называется, в лоб. Но придется дважды вводить числа a и b , что нерационально, особенно если числа эти многозначны.

Сделаем снова небольшой экскурс в структуру ПМК. Кроме стека, в машинке есть еще одна область для хранения чисел. Это — *адресуемые регистры*. Их четырнадцать. Обозначаются они так: R0, R1, ..., R9, RA, ..., RD. В эти регистры можно гереносить и сохранять там содержимое регистра X. Делается это с помощью двух клавиш: «П» (Память) и «N» (где N — цифра или буква, «имя» регистра, в который пересылается число из регистра X). Извлекается информация из нужного регистра нажатием опять-таки двух клавиш, «ИП» (Из Памяти) и «N». Причем при переписи информации из регистра X в адресуемый регистр содержимое стека не меняется, а при записи из адресуемого регистра в регистр X движется так, как при вводе числа с клавиатуры, то есть прежнее содержимое RX переходит в RY, число из RY перемещается в RZ, из RZ — в RT, из RT — пропадает.

Вернемся к задаче 3. Пусть $a = 13,24$, $b = 18,46$, угол $\varphi = 50^\circ$.

Устанавливаем правый переключатель в положение «Г» (ведь значение угла дано в градусах) и нажимаем клавиши:

«1», «3», «,», «2», «4» — значение a вводится в регистр X;

«П», «1» — заносим a в регистр R1;

«F», « x^2 » — вычисляем a^2 ;

«1», «8», «,», «4», «6» — вводим b ;

«П», «2» — заносим b в R2;

«F», « x^2 » — вычисляем b^2 ;

«+» — получаем сумму $a^2 + b^2$;

«5», «0» — вводим угол φ (аргумент косинуса);

«F», «cos» — вычисляем $\cos \varphi$;

«ИП», «1» — вызываем величину a из R1 в RX;

«x» — вычисляем $a \cos \varphi$;

«ИП», «2» — вызываем содержимое R2 (величину b) в регистр X;

«x» — домножаем содержимое регистра Y на b ; то есть вычисляем произведение $ba \cos \varphi$;

«2» — вводим коэффициент 2;

«x» — все умножения закончены, в RX — величина $2ba \cos \varphi$;

«-» — совершив путешествие почти по всем регистрам стека, величина $a^2 + b^2$ перекочевала в RY, и теперь из нее вычитается полученное ранее произведение; в RX сейчас находится $a^2 + b^2 - 2ab \cos \varphi$;

«F», « $\sqrt{}$ » — извлекаем квадратный корень из содержимого RX.

Величина c — на экране. Если все операции выполнены правильно, $c = 14,207788$.

Советуем вам построить диаграмму движения информации в стеке, как при решении первой задачи.

Подытожим сведения, изложенные в этом разделе.

1. Числа вводятся в микрокалькулятор с клавиатуры и попадают в регистр X.

2. Порядок ввода: цифры целой части, символ «.», цифры дробной части и, если число отрицательное, символ «/ - /». Если число задано в экспоненциальной форме, то описанным образом вводится мантисса (число, которое нужно будет умножить на 10 в заданной степени), затем нажимается клавиша «ВП», набираются цифры порядка и, если порядок отрицательный, символ «/ - /».

3. Содержимое регистра X, если это требуется, стирается целиком клавишей «Сх».

4. Одноместные операции, то есть те, которые заключаются в вычислении функций, оперируют содержимым регистра X. Результат помещается в этот же регистр. Перед набором символа функции не забудьте нажать клавишу «F».

5. Двуместные операции — арифметические и возведение в степень — проводятся с содержимым регистров X и Y, причем при вычитании и делении в регистре X должно размещаться соответственно вычитаемое или делитель. Результат всегда находится в регистре X. Знак операции набирается после ввода пары чисел, а не между ними.

6. Сохранить содержимое регистра X в одном из адресуемых регистров можно, нажав клавиши «П», «N», где N — «имя» регистра.

7. Извлекается информация из адресуемого регистра клавишами «ИП», «N». Попадает она при этом в регистр X.

2. КАКИЕ ЗАДАЧИ И КАК РЕШАТЬ НА ПРОГРАММИРУЕМОМ МИКРОКАЛЬКУЛЯТОРЕ?

«Никакую серьезную задачу на микрокалькуляторе решить нельзя».

«Любую задачу можно решить на микрокалькуляторе».

Таковы крайние точки спектра ответов, которые можно получить, если обратиться к специалистам с вопросом, поставленным в заголовке.

Кто же прав? Попробуем разобраться. Специалисты аргументируют свои ответы достаточно убедительно. В век вычислительных машин, утверждают первые, когда быстроедействие в сотни тысяч операций в секунду совсем не редкость, говорить о «малютке», которая выполняет в секунду лишь несколько

арифметических операций, а элементарный синус считает несколько секунд — это все равно, что всерьез обсуждать проблему использования велосипедного транспорта для разгрузки авиалиний.

Вторые убеждены, что коль скоро нет под рукой большой, «серьезной» ЭВМ, а решать задачу надо, то уж лучше делать это с помощью микрокалькулятора, чем «вручную». Другими словами, если у вас есть велосипед, то выгоднее ехать на нем, чем ходить пешком. Все это безусловно правильно, но, по нашему мнению, есть задачи, которые не только можно решать на микрокалькуляторе, но лучше всего решать именно на нем.

Попробуем определить этот класс задач. Прежде всего, не будем забывать, что в отличие от стационарной ЭВМ, место которой в вычислительном центре, микрокалькулятор — машина, всегда находящаяся в вашем распоряжении. Ее можно взять с собой на работу и даже в дорогу, она всегда доступна, обратиться к ней можно в любое удобное время. О большой ЭВМ этого не скажешь. Она, как правило, перегружена. Поэтому приходится часами и днями ждать своей очереди. Кроме того, при решении новой задачи изрядная доля рабочего времени уходит на предварительные операции, в частности перевод программы на язык машины. Причем это время сравнительно велико и мало зависит от объема программы. Вот и получается, что решать маленькую задачу на большой ЭВМ — все равно, что ездить в метро на один перегон. Спуск и подъем на эскалаторе занимают больше времени, чем сама езда. Быстрее уж на трамвае, а иногда и пешком.

Вывод ясен. Если задача небольшая, то ее не только можно, но нужно решать на микрокалькуляторе. Общее время от ее постановки до получения результата будет при этом меньше, да и дорогостоящее время на больших машинах высвободится. Таким образом, увеличится эффективность использования парка ЭВМ в масштабах страны.

Какие же задачи следует считать небольшими и тем самым относить к числу «калькуляторных»? Это расчеты, в которых используется лишь небольшое число формул, решение алгебраических и некоторых дифференциальных уравнений, систем из двух или трех уравнений, статистическая обработка результатов несложных экспериментов и некоторые другие. Потребность в решении таких задач возникает в повседневной деятельности ученого и инженера, студента и школьника, людей самых разнообразных профессий.

Ясно также, какие задачи на микрокалькуляторе не решить. Не годится он, скажем, для определения прогноза погоды, расчета

траектории космического корабля, анализа экономического положения в отрасли, то есть в тех случаях, когда необходимо обрабатывать большие массивы информации и производить огромное количество вычислений.

Конечно, это только рекомендации. Ведь если микрокалькулятор — единственная ЭВМ, находящаяся в вашем распоряжении, то поневоле приходится решать на нем и большие задачи, сочетая при этом машинную работу с ручной.

В общем, в мире ЭВМ, безусловно, есть своя «экологическая ниша» и для программируемого микрокалькулятора.

Какие задачи решать, мы обсудили. Второй вопрос: как их решать? Поясним основные этапы этого процесса на конкретном примере задачи из школьного курса физики.

Задача 4. Брусok массы $m = 350$ г скользит по горизонтальной плоскости под действием силы, приложенной к нему под углом $\alpha = 40^\circ$. Ускорение бруска $a = 0,3$ м/с², коэффициент трения $k = 0,11$. Ускорение свободного падения $g = 9,8$ м/с². Найти силу натяжения нити T и силу давления N бруска на поверхность (рис. 1).

В общем виде решение задачи запишется так:

$$T = m \frac{a + kg}{\cos \alpha + k \sin \alpha}, \quad N = m \frac{g \cos \alpha - a \sin \alpha}{\cos \alpha + k \sin \alpha}.$$

Итак, что же мы видим? Две простые формулы. Задача типично «микрокалькуляторная». Можно включать машину и начинать работу. Вычислить числитель первой формулы, потом знаменатель, разделить одно на другое и проделать затем то же самое со второй формулой, снова вводя значение угла α и вычисляя его косинус, вводя значение g и умножая его на $\cos \alpha$,...

Но ведь не будь калькулятора, мы бы так не поступили. Зачем несколько раз заглядывать в таблицы в поисках значения косинуса одного и того же угла? Мы выписали бы его один раз на бумаге, а потом брали бы готовое значение. Да и знаменатель не стали бы считать дважды. Мы выделили бы общие части обоих выражений, чтобы избежать дублирования вычислений.

Так же нужно поступить и при подготовке формул для вычислений на машине. Будем называть этот процесс *приведением формул к машинному виду*. В нашем примере видно, что несколько раз употребляются значения синуса и косинуса

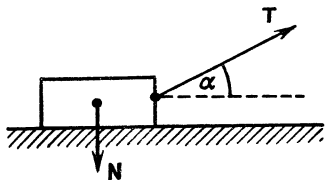


Рис. 1

одного и того же аргумента и у обеих формул есть общий множитель

$$\frac{m}{\cos \alpha + k \sin \alpha}.$$

Что ж, придадим формулам «машинный» вид:

$$x = \cos \alpha, \quad y = \sin \alpha, \quad z = \frac{m}{x + ky},$$

$$T = z(a + kg), \quad N = z(gx - ay).$$

Вместо двух формул получилось пять. Прибавка не такая уж большая, зато ни одну величину не надо вычислять дважды. И счет короче, да и клавиш при работе надо будет нажимать меньше, а тем самым уменьшится риск ошибки.

Вот теперь пора браться за работу. Включаем микрокалькулятор, передвинув левый переключатель вправо, устанавливаем правый переключатель в положение «Г» — градусы (ведь нам придется вычислять значения тригонометрических функций, аргументы которых записаны в градусах) и начинаем нажимать клавиши:

«4», «0» — вводим значение угла α , выраженное в градусах, в регистр X;

«F», «cos» — примерно через 3,5 с на экране появляется значение косинуса угла α ;

«П», «1» — вычисленное значение $\cos \alpha$ отправлено на хранение в R1 (это все равно, что записать его на бумаге «для памяти»);

«F», «Vx» — для вычисления синуса нам нужно задать значение его аргумента. Но ведь мы уже вводили его! Где же он сейчас? Постройте, как в предыдущей главе, диаграмму движения информации по стеку, и вы убедитесь, что аргумент находится в регистре X1. Команда «F», «Vx» вызывает его в регистр X;

«F», «sin», «П», «2» — теперь в регистрах X и R2 находится значение $\sin \alpha$;

«0», «,», «1», «1» — вводим значение k . При этом величина $\sin \alpha$ поднялась в регистр Y, и все готово к умножению,

«x» — произведение $k \sin \alpha$ получено. Эта величина на экране и, соответственно, в регистре X. Нужно сложить ее с $\cos \alpha$. Если вы проследите с помощью диаграмм движение информации по стеку, то убедитесь, что вычисленное ранее значение $\cos \alpha$ «опустилось» в регистр Y, и получить сумму теперь можно, нажав клавишу, задающую сложение;

«+» — знаменатель обеих формул вычислен;

«0», «,», «3», «5» — вводим массу бруска m . Ее надо разделить на полученный знаменатель, который находится сейчас в RY. При делении же числитель должен находиться в RY, а знаменатель — в RX. А у нас пока все наоборот;

«XY» — команда, заданная нажатием этой клавиши, помещая местами содержимое регистров X и Y, не затрагивая Z и T. Теперь все встало на свои места и можно провести деление;

«÷» — на экране видим величину, обозначенную в наших формулах буквой z ;

«П», «3» — записали результат деления в R3 (все промежуточные вычисления закончены. Можно приступать к вычислениям величин T и N по последним двум формулам);

«0», «,», «1», «1» — вновь вводим значение k ;

«↑» — пересылаем k в RY;

«9», «,», «8» — в RX введена величина g , все готово для получения произведения kg ;

«×» — произведение в RX, его можно визуальным способом проконтролировать;

«0», «,», «3» — записали ускорение a в RX и одновременно «подняли» прежний содержимый RX в RY;

«+» — величина $a + kg$, на которую надо умножить коэффициент z для получения значения T , готова. Можно умножать. А где находится z ? В регистре 3. Можно извлечь его оттуда. Но нужно ли? Ведь величина z находится также и в RY. (Проследите ее путь сами.) Вот оно, удобство стека!

«×» — одна из двух требуемых величин — сила натяжения нити T — вычислена. Перепишите ее значение в тетрадь. Не забудьте, что она выражена в ньютонах. Приступим к вычислению по последней формуле;

«9», «,», «8» — вновь вводим ускорение свободного падения в RX;

«ИП», «1», «×» — извлекаем содержимый R1 (там хранится $\cos \alpha$) в RX и умножаем его на g ;

«0», «,», «3», «ИП», «2», «×» — делаем то же самое с величинами a и $\sin \alpha$;

«-» — из первого произведения (оно как раз сейчас в RY) вычитаем второе. Получена величина $g \cos \alpha - a \sin \alpha$, которую осталось умножить на z ;

«ИП», «3», «×» — перед умножением мы переводим величину z из R3 в RX; на экране теперь последний результат — сила давления N в ньютонах. Все. Расчет закончен.

Если все действия выполнены правильно, то ответ таков: $T = 5,7639597 \cdot 10^{-1}$, $N = 3,0594998$. Восемь значащих цифр результата приведены только для того, чтобы вы смогли про-

верить правильность своих вычислений. Вообще же приводить ответ с такой точностью бессмысленно. Ведь значение величины g дано всего с двумя значащими цифрами, поэтому и результаты верны с точностью до двух знаков.

(Вопрос о точности вычислений очень серьезен и заслуживает более подробного рассмотрения. Мы вернемся к нему в разделе 9.)

Не показались ли вам манипуляции с микрокалькулятором сложными и утомительными? Скорее всего, показались, если вы впервые сели за «пульт» своей ЭВМ. Но не отчаивайтесь. Заучивать таблицу умножения было не легче. Решив на микрокалькуляторе десяток — другой задач, вы приобретете нужные навыки и будете проводить вычисления с неменьшим автоматизмом, чем умножение на бумаге в столбик, и, естественно, намного быстрее.

Есть у вас, вероятно, и еще одна причина для недоумения. Уже второй раздел в книге о программируемом микрокалькуляторе подходит к концу, рассмотрено несколько примеров решения задач на нем, а о программировании как будто ни слова. И тем не менее вы уже программируете. Ведь что такое программирование, как не составление набора инструкций, показывающего, какие операции, в какой последовательности и над какими данными должна проводить машина для получения результата? Другое дело, что программа не записывалась в память машин, а хранилась в вашей собственной памяти. Нужно было самим запоминать последовательность инструкций и давать машине указания об их выполнении. С таким же успехом можно было записать программу целиком в память машины и поручить ей исполнение программы. Но нужно ли было это делать? Давайте обсудим этот весьма немаловажный вопрос.

Программа вводится в микрокалькулятор с помощью тех же клавиш, которые мы нажимаем при ручном счете. Поэтому, если требуется провести один расчет, как в нашем примере, то ввод программы в микрокалькулятор никакой экономии времени не дает. Поэтому он и не нужен. Проще, удобнее и быстрее получить результат, просто последовательно вводя команды «из головы» нажатием на соответствующие клавиши, то есть работая вручную.

Иное дело, когда по одним и тем же формулам ведется расчет для разных значений исходных параметров. К примеру, решая ту же физическую задачу, вы должны определить не просто величины T и N , а их зависимости от угла α . Здесь при ручной работе пришлось бы многократно вводить одну и ту же последовательность команд. А программу можно

ввести один раз. Потом достаточно набирать значения угла и автоматически получать результаты нажатием всего одной клавиши «С/П».

Не обойтись без программы, решая на микрокалькуляторе сложные уравнения. Там приходится выполнять однотипные операции при постоянно меняющихся входных данных. Это вообще характерная черта большинства численных методов, позволяющих свести приближенное решение задач к выполнению конечного числа арифметических действий над числами.

Итак, если перед вами задача, при решении которой приходится многократно выполнять однотипные действия, составляйте программу. Однако для этого необходимо пройти ряд предварительных этапов, готовя задачу к ее решению на машине. Они всегда одни и те же, идет ли речь о большой ЭВМ или о миниатюрном калькуляторе. Правда, если задача предельно проста, отдельные этапы удастся пройти быстрее или вовсе миновать.

Первый этап — постановка задачи. На этом этапе четко формулируются цели ее решения, описываются ее исходные данные и допущения, при которых она будет решаться. Так, приступая к только что решенной нами задаче из механики, мы описали движение бруска по горизонтальной плоскости, а затем поставили цель: найти силу натяжения нити и давления бруска на поверхность, по которой он скользит.

Второй этап — математическая формулировка задачи. На этом этапе необходимо выписать все формулы, которые будут использованы при ее решении. Тем самым формулировка задачи, составленная на первом этапе, переводится на математический язык.

Третий этап — выбор метода решения. Нередко, как и в нашем примере с бруском, бывает, что все дело сводится к

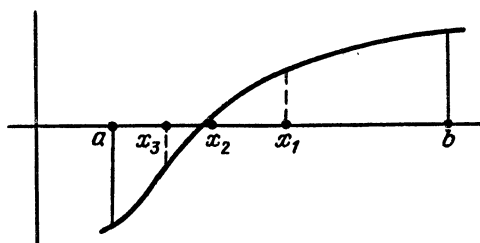


Рис. 2. Метод деления отрезка пополам

Отрезок, содержащий корень, делится пополам; часть отрезка, не содержащая корень, отбрасывается, а оставшаяся вновь делится пополам. Процесс продолжается до тех пор, пока длина отрезка, содержащего корень, не станет меньше некоторого числа — точности определения корня.

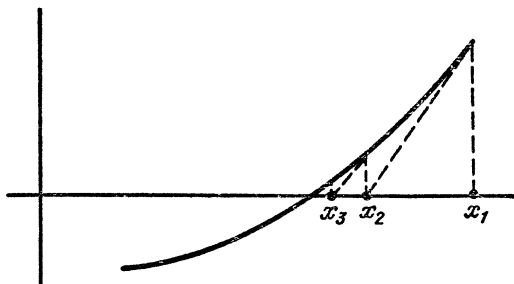


Рис. 3. Метод касательных

Из некоторой точки проводится касательная к линии графика. Вычисляется значение функции в точке пересечения касательной с осью x и из этой точки проводится новая касательная. Процесс повторяется, пока координаты двух подряд вычисленных точек пересечения касательной с осью x не будут отличаться друг от друга на величину, меньшую заданной точности вычислений.

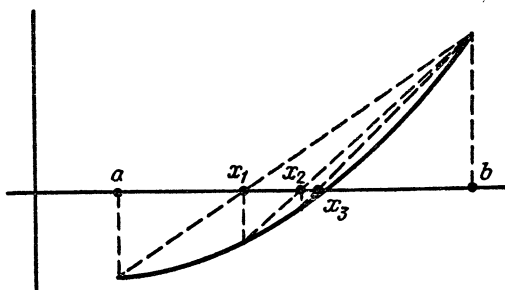


Рис. 4. Метод хорд

Через крайние точки графика функции проводится хорда. Вычисляется значение функции в точке пересечения хорды с осью x . Через новую точку графика вновь проводится хорда. Критерий окончания работы, как и в предыдущем методе.

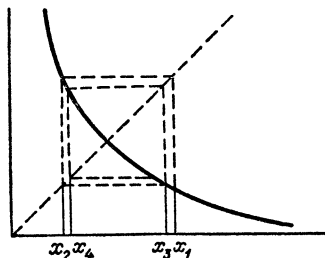


Рис. 5. Итерационный метод

Этим методом решаются уравнения, записанные в виде $x = f(x)$. В произвольной точке вычисляют значение функции. Принимают его за новый аргумент и вновь вычисляют функцию. Критерий окончания аналогичен двум предыдущим.

тому, чтобы выразить в виде формул зависимость искомых величин от данных. Тогда метод решения прост: вычисления по полученным формулам. Бывает однако и так, что выбор метода сам по себе становится сложной проблемой. Скажем, корень функции можно находить разными способами (рис. 2–5). Как правило, в подобных случаях каждый метод при его сравнении с другими обнаруживает свои достоинства и недостатки. Выбрать нужно такой, который, во-первых, вообще применим для решения данной задачи, во-вторых, может быть реализован на имеющейся машине, в-третьих, обеспечивает необходимую точность, в-четвертых, позволяет получить результат за возможно более короткое время.

Сделав выбор, следует записать алгоритм, то есть последовательность операций, выполнимых на имеющейся машине и обеспечивающих в итоге получение результата по выбранному методу. Это — четвертый этап. Он ставит свои проблемы. Ведь один и тот же метод может быть реализован различными последовательностями операций, разными алгоритмами. Мы уже отмечали это, когда обсуждали решение задачи о бруске, скользящем по горизонтальной плоскости: можно было вести расчеты по каждой из двух получившихся у нас формул независимо друг от друга, а можно было выделить в обеих формулах повторяющиеся выражения, вычислить их заранее и использовать в готовом виде при дальнейших вычислениях. Да и по ходу счета мы не раз стояли перед выбором — например, где хранить общий множитель обеих формул? В одном из адресуемых регистров, откуда его можно вызывать при необходимости? Или в стеке, чтобы он в нужный момент был на том месте, где должен находиться перед умножением?

При составлении сложных программ подобные вопросы возникают то и дело. И чтобы увереннее их решать, рисуют графический портрет алгоритма, так называемую *блок-схему*. Составление блок-схемы — это пятый этап решения задачи. Готовя к расчетам задачу о бруске на плоскости, мы миновали этот этап, поскольку расчеты оказались весьма несложными.

Пройдя эти пять этапов, можно приступить к шестому — составлению программы. Впрочем, может оказаться, что кто-то уже составил ее до вас. Стоит ли в таком случае тратить время на изобретение давно изобретенного? Вооружитесь сборниками программ для микрокалькуляторов*) и поищите, нет ли

*) Цветков А., Епанечников В. Прикладные программы для микроЭВМ. — М.: Финансы и статистика, 1984.

Трохименко Я., Любич Ф. Инженерные расчеты на программируемых микрокалькуляторах. — Киев: Техника, 1985.

там готовой программы, решающей вашу задачу. Если есть, то зачем выдумывать ее заново? Можно воспользоваться готовой. И только если такой программы не найдется, садитесь ее составлять.

Кстати, сборники программ помогут вам и в том случае, когда вы тренировки ради соберетесь написать программу, аналогичную имеющейся. Сравнив свою программу с напечатанной, вы легко сможете выявить плюсы и минусы своего произведения.

После того, как программа составлена, нужно ее отладить, то есть убедиться, что она работает правильно; если же она содержит ошибки — исправить их, короче говоря, сделать программу работоспособной. Это седьмой этап. Только на нем начинается непосредственный контакт пользователя с ЭВМ.

Убедившись в том, что программа работает правильно, можно наконец переходить к последнему, восьмому этапу — эксплуатации программы. Мы намеренно употребили несколько расплывчатый термин «эксплуатация» вместо более конкретного «проведение расчетов». Ведь кроме вычислительных программ, существуют и другие, например игровые, демонстрационные и т. д. По этим программам никаких расчетов как таковых проводить не надо. Однако все этапы разработки и использования каких бы то ни было программ совпадают с только что описанными.

В заключение подведем итоги.

1. На микрокалькуляторе целесообразно решать небольшие задачи, не требующие одновременного хранения в памяти многих величин и позволяющие получить результат за приемлемое время. Только в таких случаях использование ПМК даст выигрыш по сравнению с большими ЭВМ.

2. Если есть необходимость решить большую задачу, а из всех ЭВМ доступен только микрокалькулятор, то задачу следует все же решать с его помощью, разбивая ее на ряд небольших «микрокалькуляторных» задач. Хотя этот процесс довольно сложен и трудоемок, он дает определенный выигрыш по сравнению с расчетами вручную.

3. Решение задач на ПМК возможно в двух режимах: ручном и программном. В первом случае команды реализуются сразу после ввода их с клавиатуры. Во втором — вся программа сначала вводится в память машины и лишь затем выполняется. Рекомендуется использовать ручной режим для проведения таких расчетов по формулам, в которых не приходится многократно использовать однотипные операции. В других случаях предпочтительнее программный режим.

4. Прежде чем писать программу, убедитесь, нет ли аналогичной уже готовой. Если есть, лучше использовать ее. Таким образом экономится драгоценное время.

5. Этапы решения задачи с помощью ЭВМ:

- 1) постановка задачи,
- 2) математическая формулировка задачи,
- 3) выбор метода решения,
- 4) выбор алгоритма,
- 5) составление блок-схемы алгоритма,
- 6) написание программы,
- 7) отладка написанной программы,
- 8) эксплуатация программы.

3. ЛОГИКА МИКРОКАЛЬКУЛЯТОРА

В предыдущем разделе речь шла о том, как решать задачи на микрокалькуляторе. А как решает задачу сам микрокалькулятор? Что происходит в нем, когда мы нажимаем те или иные клавиши или когда калькулятор работает по введенной в него программе? Какие «колеса» приходят в невидимое и бесшумное движение при вычислении произведений, сумм или квадратных корней? Ответы на эти вопросы можно получить, познакомившись со структурой программируемого микрокалькулятора, с логикой его работы.

Начнем это знакомство с описания тех действий, которые совершаются при вводе программ в микрокалькулятор. Для этого его следует перевести в состояние, называемое *режимом программирования*. Перевод совершается при помощи клавиш «F», «ПРГ»^{*)}.

Подобно тому как использование переключателя «Р—Г» не подразумевает совершения конкретных операций, а только задает режим интерпретации аргументов тригонометрических функций, клавиша «ПРГ» тоже лишь задает режим интерпретации вводимой информации. Если теперь нажать одну из клавиш, которыми пользовались в ходе расчетов, микрокалькулятор никаких вычислений не совершит, а запишет вырабатываемую этой клавишей команду в свою программную память.

Чтобы после ввода программы вернуться в *режим вычислений*, надо нажать клавиши FАВТ. Стоит сказать, что в

^{*)} Так как клавиша «F» самостоятельной роли не играет, а позволяет лишь выбирать функции, написанные над другими клавишами, то в дальнейшем наборы, подобные «F», «ПРГ», мы будем обозначать единой записью FПРГ. То же относится к наборам, начинающимся и с других «служебных» клавиш — «К», «П», «ИП».

«Руководстве по эксплуатации», прикладываемом заводом-изготовителем к микрокалькулятору, этот режим именуется иначе: «Автоматическая работа». Название не очень удачное, оно сбивает с толку. Ведь в этом режиме можно заставить микрокалькулятор как исполнять отдельные команды, работая вручную, нажимая на клавиши, так и автоматически выполнять расчеты по программе; только во втором случае режим можно назвать автоматическим. Мы будем придерживаться названия «Режим вычислений».

Итак, включаем микрокалькулятор, нажимаем клавиши **ГПРГ**. В правом углу экрана появляются цифры 00. Программируемый микрокалькулятор готов к приему программы.

Программа для ПМК представляет собой набор команд, следуя которым машина обрабатывает информацию, то есть решает запрограммированную задачу.

Полная совокупность команд вместе с правилами их употребления и толкования образует язык микрокалькулятора. Естественно, чтобы общаться с ПМК, язык этот надо изучить. Лучший способ учить иностранный язык — разбирать несложные тексты, написанные на этом языке. Мы тоже начнем с несложных программ-текстов на языке ПМК.

В предыдущем разделе была рассмотрена небольшая физическая задача и описано ее решение в режиме вычислений. Теперь для тех же целей мы напишем программу.

Напомним условие задачи. Брусок массой $m = 350$ г скользит по горизонтальной плоскости под действием силы, приложенной к нему под углом α . Ускорение бруска $a = 0,3$ м/с², коэффициент трения $k = 0,11$. Ускорение свободного падения $g = 9,8$ м/с². Найти зависимость силы натяжения T нити и силы давления N бруска на поверхность от угла α .

Как видите, мы слегка изменили условие задачи. Вместо определения значений T и N при некотором значении α предлагается найти зависимость T и N от угла α , то есть получить набор значений этих величин. В этом случае создание программы оправдано.

Первый этап решения задачи, заключающийся в ее постановке, нами уже пройден. Второй, третий и четвертый этапы (математическая формулировка задачи, выбор метода ее решения и алгоритма расчетов) описаны в предыдущем разделе. Там же говорилось, что ввиду простоты задачи можно миновать пятый этап ее подготовки к машинному решению — составление блок-схемы алгоритма. Так что приступим сразу к шестому этапу — составлению программы.

Команды, из которых состоит любая программа для микрокалькулятора, должны быть записаны в последовательных

ячейках его программной памяти. Выполнив одну команду, микрокалькулятор автоматически приступит к выполнению следующей, выполнив ее — к выполнению следующей за нею, и так до конца программы.

Каждая ячейка программной памяти носит свой номер, называемый *адресом*. В «Электронике БЗ-34» таких ячеек 98. Нумеруются они двузначными числами от 00 до 97. Количество ячеек определяет максимальную длину программы, которая может быть введена в память микрокалькулятора.

Программа для решения только что поставленной задачи о движении бруска по горизонтальной плоскости перед вами (табл. 2). Написана она в трех колонках. В первой колонке

Таблица 2

Адрес:	Команда	Код:	Адрес:	Команда	Код
00	Fcos	1Г	16	+	10
01	П1	41	17	×	12
02	FBx	0	18	С/П	50
03	Fsin	1Г	19	ИПД	6Г
04	П2	42	20	ИП1	61
05	ИПВ	6L	21	×	12
06	×	12	22	ИПА	6—
07	+	10	23	ИП2	62
08	ИПС	6Г	24	×	12
09	XY	14	25	—	11
10	÷	13	26	ИП3	63
11	П3	43	27	×	12
12	ИПВ	6L	28	С/П	50
13	ИПД	6Г	29	БП	51
14	×	12	30	00	00
15	ИПА	6—			

пишется адрес команды, во второй — сама команда (клавиши, нажимаемые при ее вводе), в третьей — код команды. Стоящие рядом обозначения во второй и третьей колонках по существу выражают собою одно и то же с той лишь разницей, что во второй колонке стоит символ, привычный для работающего на микрокалькуляторе, в третьей же — своеобразный перевод того же символа на язык цифр, понятный машине.

Если сравнить программу с последовательностью клавиш, которые мы нажимали, решая аналогичную задачу из предыдущего раздела, то легко убедиться, что программа почти полностью повторяет тот же набор. Фигурируют те же символы знаков операций (сложение — команды по адресам 07 и 16, вычитание — по адресу 25), умножение (06, 14, 17, 21 и 24), обращение к функциям (sin — адрес 03, cos — адрес 00), команда

перемены местами содержимого регистров X и Y (адрес 09) и вызов содержимого регистра X1 в регистр X (адрес 02). Однако две команды нам еще не встречались: С/П (18 и 28) и БП 00. Вторая из них, в отличие от всех предыдущих, размещается в двух смежных ячейках (по адресам 29 и 30).

Команда С/П (Стоп/Пуск) используется в программе для остановки процесса вычислений (для «останова», как говорят программисты). В нашем случае эти «остановы» записаны после вычисления величин T и N , чтобы считать их значения с индикатора. В режиме вычислений эта команда либо останавливает программу, если она в работе, либо запускает, если она «стоит».

Команда БП nm (в нашем случае, БП 00), где nm — двузначное число от 00 до 97, читается: Безусловный Переход на адрес nm . Она прерывает последовательное выполнение команд, записанных в программе. Следующей после нее выполняется команда, записанная по адресу nm . В нашем случае команда БП 00 введена для того, чтобы по окончании расчета величин T и N для заданного угла α передать управление к началу программы, обеспечив тем самым возможность расчета для нового значения угла.

Вернемся к микрокалькулятору, на индикаторе которого в правом углу горят цифры 00. Они означают не только готовность к вводу программы, но также и то, что первая из введенных нами команд будет записана по адресу 00. Нажимаем клавиши, названия которых стоят во втором столбце представленного выше текста программы. Цифры 00 в правом углу индикатора последовательно сменяются цифрами 01, 02 и так далее. В левом же углу будут появляться следующие символы: 1Г, 41, ... И каждая новая комбинация из двух символов смещает вправо появившиеся ранее комбинации. Далее на индикаторе всегда будут гореть четыре двузначных выражения. Самое левое — код последней введенной команды, за ним — коды двух предыдущих и, наконец, последняя пара цифр — адрес команды, которую предстоит вводить. Коды позволяют контролировать, правильно ли вводится программа.

Т а б л и ц а 3

Команда	Индикатор			
Fcos	1Г			01
П1	41	1Г		02
FBx	0	41	1Г	03
Fsin	1Г	0	41	04
П2	42	1Г	0	05

Фрагмент программы в том виде, в каком он отображается на индикаторе, показан на табл. 3.

Среди символов, образующих коды, нередко встречаются непривычные знаки: L, Г, С. Читатель, интересующийся ими, может прочесть о них подробнее в приложении. Полная таблица кодов дана на третьей странице обложки.

Наконец, программа введена и записана в память микрокалькулятора. Но правильно ли она там записана? Ведь вводит ее человек, а ему, увы, свойственно ошибаться.

Исправление ошибок ввода — часть седьмого из тех этапов, на которые разбивается решение задачи на машине. Подробному описанию этого этапа будет посвящен раздел 8. Здесь же мы дадим один совет: если вы заметили, что код только что введенной вами команды не соответствует записанному в третьем столбце программы, то нажмите клавишу «ШГ» (Шаг назад) и повторите ее ввод.

Например, при вводе программы на индикаторе появилось:
1Г 0 41 04.

Это означает (загляните в таблицу кодов), что по адресу 01 в программе стоит команда засылки в регистр 1 содержимого регистра X, по адресу 02 — команда вызова в регистр X содержимого регистра X1, по адресу 03 — команда вычисления косинуса. Но, согласно тексту программы, по адресу 03 должна стоять команда вычисления синуса. Стало быть, произошла ошибка. Нажимаем клавишу «ШГ». Картина на индикаторе меняется:

0 41 1Г 03.

Повторяем ввод команды $F \sin$. В левом углу индикатора появляется нужный нам код

1Г 0 41 04.

Можно продолжать ввод.

После ряда «проб и ошибок» программа введена. Что теперь? Нужно перевести микрокалькулятор в режим вычислений. Нажимаем клавиши FAVT. Машина готова считать по программе. Но мы еще не все подготовили для этого.

Обратите внимание, что в программе нет конкретных значений величин a , k , g , m и α . А без них какой же счет? Там, где по смыслу должны вводиться эти величины, записаны команды ИПА, ИПВ и т. д. Подразумевается, что значения постоянных величин записаны в адресуемые регистры. В нашей задаче величина a должна быть в RA, b — в RB, m — в RC и g — в RD. Естественно, сами они туда не попадут. Их надо

вести, набирая на клавиатуре нужное число, а затем нажимая клавишу «П» и еще одну, задающую номер регистра, куда это число должно быть занесено. Проведем эту работу:

0,3 ПА 0,11 ПВ 0,35 ПС 9,8 ПД

Величины a , k , t и g записаны теперь в регистры RA, RB, RC, RD. Число α засылать в адресуемый регистр не стоит: ведь с него начинаются вычисления, так что его лучше всего занести в регистр X непосредственно перед началом очередного расчета.

Микрокалькулятор должен начать этот расчет с того адреса, где записана первая команда программы, то есть в нашем случае — с нулевого. Нажимаем клавишу «В/О» (Возврат/Очистка). Введенная в режиме вычислений, эта команда возвращает программу к нулевому адресу. Вот теперь-то мы и введем в регистр X значение угла в градусах (проверьте, установлен ли переключатель «Р—Г» в положение «Г»). Набираем на клавиатуре нужную величину, для начала 40. (Заметим, что в более сложных программах каждый вариант расчета может определяться не одним, а двумя, тремя и даже четырьмя переменными параметрами. Их можно вводить в стек. О подобной возможности у нас еще будет случай поговорить в последующих разделах.)

Теперь все готово к вычислениям по программе. Нажав клавишу «С/П», запускаем программу на счет. Примерно через 10 секунд читаем на индикаторе: 5,7639597 — 01. Первые 8 цифр — мантисса числа, две последние со знаком «—» — порядок. Перепишем это число на бумагу в привычном виде: 0,57639597. Снова нажимаем «С/П». Секунд через 5 можно прочесть второе число: 3,0594998.

(Если числа на экране не совпадают с написанными, значит, при вводе программы допущена ошибка. Поскольку процесс исправления и отладки программ будет рассмотрен в одном из следующих разделов, то здесь нам остается только порекомендовать самый простой выход из положения. Выключите калькулятор, секунд через 10 включите вновь и повторите ввод программы, строго контролируя каждый шаг.)

Если результаты расчета совпали с написанными выше, программа введена правильно, и можно продолжать решение задачи. Теперь достаточно набирать на клавиатуре новые значения угла в градусах и нажимать на клавишу «С/П». После первого останова на экране — значение силы натяжения нити T , после второго — силы давления N . Можно, вводя последовательно нарастающие значения угла α , построить графики зависимостей T и N от α . Один из этих графиков позволит вам сделать открытие: оказывается, при некотором значении

α сила N обращается в нуль и затем меняет знак. Можно задаться вопросом: почему это происходит? Может ли давление тела на плоскость стать отрицательным? Если да, то почему тело не улетает? А если нет, то почему формула дает такой результат? Здесь без знания физики не обойтись. Вот вам и пример использования микрокалькулятора при изучении этой науки.

Но вернемся к изучению собственно микрокалькулятора. Мы выяснили, что программа для него состоит из набора команд, которые вводятся с пульта в режиме программирования. Когда программа запущена, на экране мелькают цифры. А что происходит в это время внутри ПМК? Как считает микрокалькулятор?

С одной стороны, знать это не обязательно, подобно тому, как владельцу магнитофона не нужны знания о принципах записи и воспроизведения звука. Но, с другой стороны, всегда лучше знать, хотя бы в общих чертах, как работает устройство, с которым имеешь дело. Вот мы и решили познакомить вас с принципиальной схемой ПМК, логикой его работы и кратко охарактеризовать функциональные назначения его основных элементов (рис. 6).

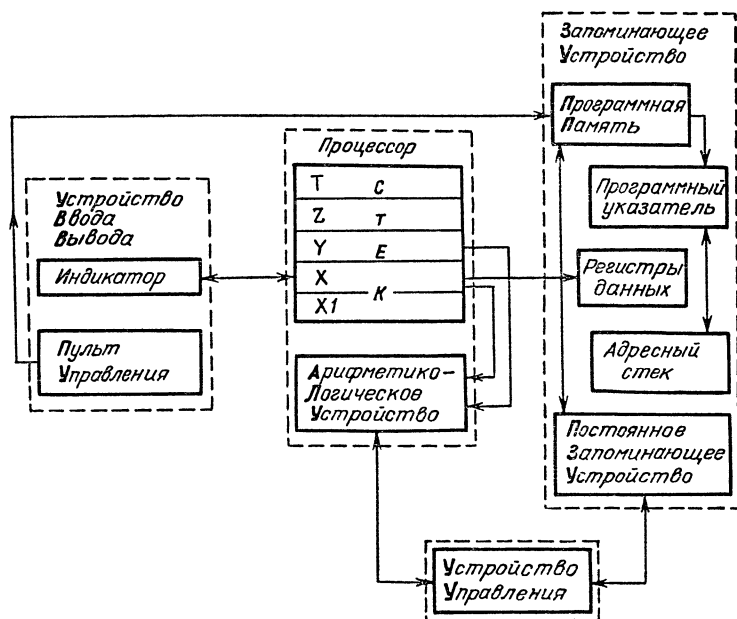


Рис. 6.

Основными элементами ПМК являются устройство ввода и вывода информации (УВВ), устройство преобразования информации (процессор), запоминающее устройство (ЗУ) и устройство управления (УУ).

Устройство ввода и вывода — единственное, которое мы непосредственно видим. Состоит оно из клавиатуры, сочетающей функции устройства ввода и пульта управления, и индикатора. Клавиатура используется как для ввода программ и чисел, так и для задания команд управления: установления режима работы, пуска и останова программ и т. д. Программа и числа, вводимые с клавиатуры, отображаются на индикаторе. Туда же выводятся результаты вычислений. Вообще, индикатор — единственное «окно» в память машины, с помощью которого можно получить сведения о том, что в памяти записано.

Команда, введенная с клавиатуры, попадает в запоминающее устройство ПМК. Состоит ЗУ из нескольких различных секций: программной памяти, регистров данных, постоянного запоминающего устройства (ПЗУ), а также программного указателя и адресного стека.

Программная память (ПП) представляет собой набор ячеек, занумерованных адресами от 00 до 97, в каждом из которых может быть записан один код. Организована ПП наподобие колеса обозрения. Адрес текущей ячейки записывается в программном указателе. При вводе команды адрес этот автоматически увеличивается на единицу и «колесо» поворачивается, подготавливая следующую «кабинку»-ячейку для приема очередного «пассажира»-команды. Содержимое программного указателя может изменяться по командам, задаваемым с пульта управления или записанным в тексте программы. При этом и «колесо» может поворачиваться в любую сторону на заданное число позиций. Когда все 98 ячеек программной памяти заполнены, то попытка ввести новые команды приводит к повороту «колеса» в начальное положение, и команды эти попадают в первые адреса памяти, естественно, стирая при этом то, что там было записано ранее.

Адресный стек состоит из пяти ячеек и используется для запоминания адреса команды, на которую нужно передать управление после окончания работы какой-либо подпрограммы (об использовании подпрограмм будет написано в разделе 6).

Регистры данных служат для записи и хранения числовой информации. Как уже говорилось в первом разделе, их 14. Таково максимальное количество чисел, которые можно одновременно хранить в памяти ПМК.

Постоянное запоминающее устройство содержит программы, которые, собственно, и организуют процесс вычислений. Все

программы, записанные в ПЗУ, не могут быть изменены, они реализованы совокупностью электронных схем. Их нельзя даже прочесть. К ним можно только обращаться и получать результаты их работы. Именно эти программы обеспечивают выполнение арифметических операций, считают значения функций, названия которых записаны на клавиатуре.

Выполняет же все операции по программам, хранящимся в ПЗУ, процессор, точнее, арифметико-логическое устройство (АЛУ). Работает оно совместно с операционным стеком, состоящим из пяти регистров X, Y, Z, T, X1. Числа двигаются по этим регистрам либо автоматически при выполнении некоторых операций, либо с помощью специальных команд (см. четвертую страницу обложки). Два регистра стека особо важны: это X и Y. Из них АЛУ черпает информацию для выполнения двуместных операций: сложения, вычитания, умножения, деления и возведения в степень. Одноместные операции, такие, как извлечение квадратного корня, возведение в квадрат, вычисление тригонометрических функций, выполняются с содержимым регистра X.

В соответствии с кодом команды арифметико-логическое устройство вырабатывает результат операции и помещает его в регистр X. Напомним, что содержимое лишь этого регистра отображается на экране. Теперь понятно, что за мелькание видно на индикаторе во время его работы: это мелькают промежуточные результаты вычислений, появляющиеся в регистре X.

И, наконец, есть в микрокалькуляторе устройство, обеспечивающее совместную работу всех блоков ПМК, — устройство управления (УУ).

Зная функции отдельных элементов микрокалькулятора, проследим теперь полный цикл его работы при выполнении программы. Предположим, что она уже введена в память, установлен режим вычислений FAVT и все необходимые постоянные уже введены в адресуемые регистры. Нажатием клавиши «В/О» мы очищаем программный указатель. Клавиша «С/П» запускает программу. Устройство управления считывает команду, адрес которой записан в программном указателе. После ее анализа и определения типа операции команда пересылается в АЛУ. По сигналам, поступившим из УУ, процессор вырабатывает результат операции. Затем устройство управления опрашивает программный указатель и выясняет, какая команда должна выполняться следующей. После чего весь цикл повторяется. Время выполнения цикла зависит от типа команды и колеблется от десятых долей секунды для команд типа записи-считывания и операций типа сложения до нескольких секунд для вычисления тригонометрических функций.

Несколько иначе выполняются команды, отдаваемые микрокалькулятору в непрограммном (ручном) режиме работы. Такая команда не записывается в программную память, а поступает непосредственно в устройство управления, и при ее выполнении программный счетчик своего состояния не меняет. (Исключение представляют команды типа В/О, БП, ПП и им подобные, основным действием которых является именно изменение счетчика команд.) Благодаря этому появляется возможность решать на микрокалькуляторе задачи, сочетая автоматическую обработку информации по программе, хранящейся в его памяти, с непрограммными вычислениями, производимыми нажатием на клавиши. Такое сочетание оказывается полезным при необходимости решать на микрокалькуляторе сложные задачи, для которых не удается составить единую программу, уменьшающуюся в «микропамяти» калькулятора.

Вернемся к нашей задаче о движении бруска. На ее примере, как на простой модели сложного процесса, видны все особенности работы программируемого микрокалькулятора. Введя в него программу, мы перевели его в режим вычислений. В этом режиме мы выполнили команды пересылки содержимого регистра X в адресуемые регистры памяти. Затем мы нажали клавиши «В/О» и «С/П». В режиме ручных вычислений первая из этих команд устанавливает программный указатель на нулевую отметку, а вторая, если выполняется в том же режиме, запускает микрокалькулятор по введенной в него программе, начиная с адреса, на котором установлен программный указатель.

Подробнее о командах микрокалькулятора речь пойдет в следующем разделе, а этот мы закончим подведением итогов.

1. Микрокалькулятор может работать в двух режимах. Первый предназначен для ввода и редактирования программ, второй — для вычислений. Первый устанавливается клавишей FПРГ, второй — FАВТ. При включении микрокалькулятора автоматически устанавливается режим вычислений.

2. Программа для микрокалькулятора состоит из последовательности команд, вводится с клавиатуры и записывается в программную память. Помните, что очередная команда, которую вы собираетесь ввести, будет записана по адресу, который высвечивается в правом углу индикатора. После ее ввода это число автоматически увеличивается на единицу.

3. Порядок работы с программой:

- 1) установить режим программирования FПРГ,
- 2) ввести программу,
- 3) перейти в режим вычислений FАВТ,
- 4) ввести постоянные параметры в адресуемые регистры,
- 5) установить начальный адрес считывания программы,

6) набрать на клавиатуре значения переменных параметров, определяющих собой очередной расчет,

7) запустить программу,

8) если нужно, повторить расчет для нового набора переменных параметров, перейти к п. 6).

4. Максимальная длина программы — 98 шагов, максимальное количество чисел, которые могут одновременно храниться в адресуемых регистрах, — 14.

Приложение. $10 + 10 = 100$

Это не опечатка и не ошибка: именно такой результат получается, если числа записаны в двоичной системе счисления.

Системой счисления называется способ выражения и записи чисел. Числа записываются в виде последовательности специальных символов. Смысл каждого символа зависит от позиции, или разряда, в котором он записан. Количество единиц младшего разряда, объединяемых в одну единицу смежного с ним старшего разряда, называется *основанием* системы, а символы, используемые для обозначения единиц каждого разряда — *цифрами*.

Наиболее употребительна десятичная система. Десять единиц младшего разряда, объединяясь, дают одну единицу старшего, а знаки, изображающие цифры, — это 0, 1, ..., 9. Мы настолько привыкли к этой системе, что, не задумываясь, «раскрываем» любое число. Например, $512 = 2 + 1 \cdot 10 + 5 \cdot 10^2$. Эта система представляется нам столь же естественной, как родной язык ребенку. Но так же, как естественен любой язык, естественна и любая система счисления. В вычислительной технике используются двоичная, восьмеричная и шестнадцатеричная системы. Двоичная — самая простая и наиболее удобная для технической реализации. Цифр в ней всего две: 0 и 1. Когда в разряде *) накапливаются две единицы, то они, подобно десятке в привычной нам системе, заменяются единицей старшего разряда. Поэтому число 2_{10} (цифрой внизу обозначается основание системы) в двоичной системе записывается так: 10_2 .

Вообще, любое число, записанное в n -ричной системе, переводится в десятичную по простому правилу. К последней n -ричной цифре прибавляется предпоследняя, умноженная на

*) Называется двоичный разряд *битом*. Несколько двоичных разрядов, чаще всего 8, объединяются в *байт* — это величина, с которой ЭВМ работает, как с одним целым.

n , затем — стоящая перед ней, умноженная на n^2 , и т. д. Скажем, двоичное число $101_2 = 1 + 0 \cdot 2 + 1 \cdot 2^2 = 5_{10}$. Привлекательность двоичной системы, как уже говорилось, в простоте технической реализации. Каждый разряд — это некоторое устройство, которое может находиться всего в двух состояниях: открыто — закрыто, или есть ток — нет тока.

В микрокалькуляторе для размещения одного символа кода отводится *тетрада* — четыре двоичных разряда. Легко подсчитать максимальное число, которое можно записать в ней:

$$1111_2 = 1 + 1 \cdot 2 + 1 \cdot 2^2 + 1 \cdot 2^3 = 15_{10}.$$

Добавив к этому числу еще один символ «0», получаем 16 различных «цифр». В такой шестнадцатеричной системе и изображаются коды в «Электронике БЗ-34». Так как десятичных знаков для изображения шестнадцатеричных цифр не хватает, то приходится «выдумывать» специальные символы. В ПМК символом «-» изображается число 10, «L» — 11, «G» — 12, «Г» — 13 и «Е» — 14. Цифра «15» в обозначениях кодов не используется.

4. ЯЗЫК МИКРОКАЛЬКУЛЯТОРА

Существование языка, на котором записываются программы, является принципиальной особенностью любой ЭВМ, будь то самая большая и мощная или самая маленькая, такая, как наш программируемый микрокалькулятор.

Языки программирования развивались по мере развития ЭВМ. Первые машины знали один язык — язык команд. На нем приходилось изъясняться и тем, кто с этими машинами общался. Каждая операция на таком языке представлялась цифровым кодом: сложение, скажем, обозначалось 01, умножение — 05. Размещалась каждая команда в отдельной ячейке машинной памяти. Кроме кода операции, туда же записывались адреса операндов — тех чисел, над которыми по этой команде выполнялась предписанная операция.

Программа, составленная на таком языке, представляла собой набор определенным образом записанных цифр. Такие программы были максимально понятны машине и минимально — человеку. Их составление и отладка требовали значительных навыков и труда.

Желание записывать программы в виде, более понятном человеку, привело к появлению новых языков, называемых *автокодами* или *языками ассемблера*. Отличались они тем, что в них допускалось употребление символов вместо числовых кодов: + вместо 01, × — вместо 05. Операнды можно было

также обозначать символами, а не адресами, в которых они хранятся. Это сильно облегчило запись программ.

Поясним сказанное примером. Допустим, требуется сложить числа X и Y , хранящиеся в ячейках памяти с адресами 0101 и 0102, и поместить сумму Z в ячейку с адресом 0150. Раньше такая команда записывалась так:

01 0101 0102 0150

Теперь же запись команды выглядела понятнее:

+ XYZ

При таком способе записи программ появилась необходимость в *трансляции* — переводе программ на язык машины. Ведь ЭВМ понимает только цифровой язык. А при подобной записи появились символы, причем не только для обозначения кодов операций, но и адресов. Адреса, выраженные символами, стали называть *относительными*. За адресами, выраженными числами, укрепилось название *абсолютных*. Все это привело к необходимости перевода: символов операций — в коды, а относительных адресов — в абсолютные.

Тем не менее, оказалось, что игра стоит свеч: затраты времени на трансляцию программ не шли ни в какое сравнение с экономией времени на их составление и отладку. Это обстоятельство подстегнуло разработчиков, и был сделан еще один шаг: на смену автокодам пришли универсальные алгоритмические языки. В них, например, приведенная выше команда суммирования записывается в виде привычного для любого школьника выражения

$$Z = X + Y.$$

Трансляция программ, конечно, при этом намного усложнилась, но это окупилось сторицей: программирование из редкой профессии превратилось в занятие миллионов.

В языках высокого уровня (так стали называть универсальные алгоритмические языки в отличие от автокодов — языков низкого уровня) место команд заняли *операторы*. Если команда — это элементарное действие машины, то оператор — это уже совокупность таких действий. Например, оператор

$$c = a \uparrow 2 + b \uparrow 2 + 2 \times a \times b \times \cos(c)$$

(символом « \uparrow » здесь обозначено возведение в степень) требует для своей реализации около десятка команд.

Достоинства языков высокого уровня несомненны. Недостаток же у них всех один и тот же — необходимость перевода, трансляции программ с этого языка на язык команд, необхо-

димось выделять в машинной памяти резервы для хранения программы-транслятора.

Для программируемого микрокалькулятора с его микропамятью в сотню ячеек это оказалось не под силу. Пришлось изобретать для записи его программ специальный язык. По своему положению он занимает место посередине между самыми примитивными языками команд и автокодами. От первых взята жесткая привязка команд к абсолютным адресам программной памяти, а используемых в программе чисел — к адресуемым регистрам, от вторых — условная символическая запись операций, обозначения которых мы видим на клавиатуре. Зато при этом не существует проблемы перевода: микрокалькулятор и его пользователь говорят на одном языке.

Как и всякий другой язык программирования, язык микрокалькулятора представляет собой набор символов и правил, определяющих, как с помощью этих символов писать тексты и как понимать написанное.

Правда, в отличие, скажем, от русского языка, где слова расчленяются на буквы и могут изменяться при склонении и спряжении, язык микрокалькулятора напоминает скорее китайский или японский. Он состоит из несклоняемых слов-«иероглифов» и только порядком их следования определяется смысл текстов — программ для ПМК. Каждый «иероглиф» — это надпись на клавише или над ней (а в нижнем ряду клавиш — и под ней). Две клавиши — «К» и «F» — самостоятельной роли не играют. По своему действию они подобны переключателю пишущей машинки, от которого зависит, будут печататься прописные или строчные буквы. Если требуется выбрать «иероглиф», написанный на клавише, то нажать нужно только ее, если же выбирается действие, написанное над клавишей, то предварительно нужно нажать клавишу «F», а в некоторых случаях (о них будет сказано особо) клавишу «К».

Не играют самостоятельной роли также клавиши «П» и «ИП». Они образуют команду лишь в сочетании с какой-либо цифровой или буквенной клавишей.

Как видим, некоторые команды требуют набора двух (и даже трех, о таких — ~~позже~~) клавиш, но все равно на индикаторе и в памяти микрокалькулятора они отображаются одним кодом. Исключение составляют команды переходов. После них нужно указывать адрес перехода. Поэтому состоят такие команды из двух кодов: код собственно команды и код адреса.

Многие из команд «Электроники БЗ-34» уже хорошо знакомы нам по предыдущим разделам. Полученные ранее сведе-

ния в этом разделе, носящем в значительной мере справочный характер, будут лишь завершены и систематизированы. В этом поможет таблица кодов, приведенная на третьей странице обложки, которая будет особенно полезна при отладке программ (о чем разговор пойдет в одном из следующих разделов). Обратите внимание, что в таблице много «белых пятен». Это значит, что не все возможные пары шестнадцатеричных цифр задействованы. Оставленные резервы разработчики ПМК постепенно используют, расширяя наборы команд для следующих модификаций микрокалькуляторов, например «Электроники МК-52», «Электроники МК-61». Кстати, и в нашей «Электронике БЗ-34» некоторые комбинации клавиш имеют вполне определенные коды и образуют новые команды, не описанные в инструкции. Об этих командах мы поговорим в следующих разделах и дадим их перечень в разделе 13. Тогда вы сможете нанести их на нашу «карту микрокалькулятора», уничтожив на ней часть «белых пятен».

Этой же таблицей могут пользоваться и владельцы микрокалькуляторов «Электроника МК-54» и «Электроника МК-56». Система команд у них та же самая. Различия имеются лишь в некоторых обозначениях (« $X \rightarrow P$ » вместо « P », « $P \rightarrow X$ » вместо « IP », « $X \leftrightarrow Y$ » вместо « XY » и « $B \uparrow$ » вместо « \uparrow ») и в названиях обратных тригонометрических функций \sin^{-1} , \cos^{-1} , tg^{-1} вместо \arcsin , \arccos , arctg соответственно. Смысл этих операций и их коды полностью идентичны приведенным в таблице.

Условно все команды микрокалькулятора можно разделить на два класса: команды, используемые в программе, и команды, предписывающие порядок работы микрокалькулятора.

Вообще говоря, команды второго класса не принято относить к языку. Ведь в программу они не входят. В больших ЭВМ управление является функцией *операционной системы*. Так называется набор программ, управляющих всеми процессами обработки информации. В микрокалькуляторе функции управления, доступные работающему, сводятся к установлению того или иного режима работы, вводу и редактированию программ, изменению значения программного указателя, пошаговой прогонке программы.

Команды, реализующие эти функции, вводятся в режиме вычислений и будут рассмотрены, когда мы будем описывать процесс отладки программ и проведения вычислений по ним.

Первый класс команд можно разделить на такие группы: 1) вычислительные команды; 2) команды обмена информацией; 3) команды управления ходом вычислений; 4) команды, использующие режим косвенной адресации. Последние по своим

функциям не отличаются от команд второй и третьей групп, но используют способ адресации, называемый *косвенным*. Поэтому они и будут рассмотрены отдельно. Особняком стоит команда КНОП, которая будет пояснена в конце раздела.

Итак, начнем с вычислительных команд. Прежде всего, это команды арифметических операций: сложение + (код 10), вычитание — (11), умножение \times (12) и деление \div (13). Все эти команды двуместные и работают с содержимым двух регистров стека, X и Y, причем при вычитании в регистр X записывается вычитаемое, а при делении — делитель. Результат каждой арифметической операции заносится в регистр X, прежнее содержимое этого регистра перемещается при этом в регистр X1. То, что было в Y, пропадает, замещается числом из регистра Z, а в RZ попадает содержимое регистра T. Прежнее содержимое регистра T остается при этом на своем месте.

Напомним, что область чисел, с которыми может работать микрокалькулятор, ограничена. Во-первых, максимальное число не должно превосходить 10^{100} , точнее, $9,9999999 \cdot 10^{99}$, минимальное должно быть по модулю не меньше, чем 10^{-99} . Во-вторых, все вычисления проводятся с восемью разрядами. Но если второе ограничение влияет только на точность расчетов, то первое может привести к появлению «аварийного останова» (*авоста*) в случае, когда операция не может быть выполнена. Наибольшую осторожность надо соблюдать при умножении и делении. Часто бывает, что конечный результат цепочки операций лежит в пределах возможностей ПМК, а на промежуточном этапе возникает авост. Этого можно избежать, правильно организовав процесс вычислений.

Рассмотрим простой пример. Нужно вычислить дробь ab/c , где $a = 2 \cdot 10^{51}$, $b = 3 \cdot 10^{49}$, $c = 4 \cdot 10^{50}$. Устная прикидка показывает, что результат лежит в допустимых пределах. Допустим, что исходные величины хранятся в одноименных регистрах, и запишем фрагмент программы, вычисляющий эту дробь. Сделать это можно по-разному. Например, так:

ИПА ИПВ \times ИПС \div

Но в этом случае при выполнении умножения получается число $6 \cdot 10^{100}$. Возникает авост, калькулятор выводит на экран сообщение ЕГГОГ, и вычисления прекращаются. Если же записать фрагмент так:

ИПА ИПС \div ИПВ \times

то никаких неприятностей не произойдет.

К арифметическим можно отнести еще одну команду, / - / (0L). Не следует путать ее с вычитанием, код которого 11. Она одноместная, использует только регистр X. Работа команды состоит в изменении знака числа, находящегося в этом регистре (плюса на минус или минуса на плюс). Если же она используется после команды ВП и ввода порядка числа, задаваемого в экспоненциальной форме, то меняет она знак порядка, а не числа.

Остальные вычислительные команды используются для расчета значений различных функций. Названия их написаны над соответствующими клавишами. Поэтому, чтобы получить значение нужной функции, требуется предварительно нажать клавишу «F». При описании этих команд клавишу «F» мы для краткости упоминать не будем.

Какие же функции позволяет вычислить наш ПМК? Вот они:

- извлечение квадратного корня, $\sqrt{}$ (21);
- возведение в квадрат, x^2 (22);
- получение обратной величины, $1/x$ (23);
- возведение числа 10 в любую степень, 10^x (15);
- возведение в степень числа e (основания натуральных логарифмов), e^x (16);
- вычисление десятичного и натурального логарифмов, \lg (17) и \ln (18);
- вычисление тригонометрических функций, аргументы которых могут быть заданы как в градусах, так и в радианах,
 \sin (1D), \cos (1F), tg (1E),

а также обратных тригонометрических функций,

$$\arcsin$$
 (19), \arccos (1—), arctg (1L).

Аргумент каждой из этих функций всякий раз берется из регистра X. Туда же записывается результат. Прежнее содержимое этого регистра после выполнения операции записывается в X1, а числа в других регистрах не меняются.

Особняком среди команд подобного рода стоит x^y (24) — возведение числа в степень. Возводимое число берется из регистра X, а степень — из регистра Y. После выполнения команды результат, как обычно, заносится в RX; то, что было там прежде, переходит в RX1, а вот содержимое RY не стирается, оно остается на месте, так же, как и содержимое двух других регистров. Надо сказать, что эту команду «Электроника БЗ-34» выполняет хуже прочих: результата приходится ждать долго, да и точность его ниже, чем при выполнении других команд. Так, 2^2 , вычисленное по этой

команде, равно 3,9999996. Для сравнения: та же величина, вычисленная по команде x^2 , равна в точности четырем. Калькуляторы первых выпусков при использовании этой команды иногда дают вообще неправильные результаты. Поэтому рекомендуем по возможности ее избегать.

Для каждой из команд первой группы существует область допустимых чисел, с которыми они могут работать. Иногда эти ограничения диктуются математическим определением функции (скажем, нельзя вычислять арксинус и арккосинус от чисел, больших по модулю единицы, нельзя извлекать квадратный корень или брать логарифмы отрицательного аргумента), иногда — возможностями микрокалькулятора. Мы остановимся только на последних. Поскольку не все числа доступны микрокалькулятору, то аргументы функций 10^x и e^x не могут превосходить 99,999999 и 230,25 соответственно. Не допускаются и отрицательные аргументы, по модулю превосходящие эти числа. Функция x^y при любых значениях показателя не определена для отрицательных оснований. При вычислении тригонометрических функций запрещается выбирать в качестве аргументов числа, большие 10^{10} , независимо от того, измеряются ли они в градусах или в радианах.

Примеры использования вычислительных команд неоднократно приводились в предыдущих главах, будут они даны и дальше. Сейчас напомним только то, что в микрокалькуляторах типа «Электроника БЗ-34» используется обратная бесскобочная (или польская) запись арифметических выражений, при которой сначала записываются аргументы операции, а потом ее символ, то есть не $a \times b$, а $ab \times$, и не \sqrt{x} , а $x\sqrt{}$.

При составлении программ нужно следить, чтобы перед совершением каждой арифметической операции стек был заполнен так, как требуется для ее выполнения.

Перемещением чисел по регистрам стека и регистрам данных занимается вторая группа команд, названных в нашем списке *командами обмена*. Четыре из них работают только с регистрами стека. Это \uparrow (0E), FBx (0) (кстати, это единственная команда, имеющая однозначный код), XY (14) и FO (25).

Первая из перечисленной четверки команд — \uparrow — используется чаще всего для разделения вводимых чисел. Она сдвигает числа в стеке «снизу вверх» (см. диаграмму на четвертой странице обложки), сохраняя содержимое регистра X и выбрасывая за пределы стека число, хранившееся в регистре T.

FBx используется для вызова содержимого регистра предыдущего результата X1 в RX. В остальном ее действие

подобно действию предыдущей команды: сдвиг чисел в стеке «снизу вверх» и вытеснение содержимого регистра Т.

Команда ХУ меняет местами содержимое регистров Х и У. При этом число, находившееся в RX, дополнительно копируется еще и в регистре X1, вытесняя его прежнее содержимое. То, что было в остальных регистрах, при этом не меняется.

Команда FO совершает круговой обмен: число из RX перемещается в RT, содержимое RY передвигается в RX, RZ — в RY, а RT — в RZ. Пропадает при этом только содержимое регистра X1: там копируется содержимое RX.

Движение информации при выполнении всех этих команд наглядно изображено на диаграммах, помещенных на четвертой странице обложки.

Ряд команд позволяет пересылать содержимое RX в адресуемые регистры, называемые также *регистрами данных*. Всего их 14. Первые десять обозначаются числами от 0 до 9, последние — буквами А, В, С, Д. Для записи в них информации служат команды П0 (40), П1 (41), ..., П9 (49), ПА (4—), ПВ (4L), ПС (4Г) и ПД (4Г). Обратите внимание, что буквы А, В, С и Д написаны под клавишами, но после нажатия клавиши «П» воспринимаются именно они.

Считывается информация из адресуемых регистров в регистр RX с помощью команд ИП0 (60), ИП1 (61), ... ИПА (6—), ..., ИПД (6Г).

Условно к этим же командам можно отнести и команду Пπ (20). Все действие ее состоит в вызове из постоянного запоминающего устройства записанного там числа π в регистр X.

В языке микрокалькулятора нет специальных команд ввода. Числа просто набираются на клавиатуре и автоматически заносятся в регистр X. Однако набор числа можно поручить и программе. Например, если нужно умножить содержимое R0 на 2, мы пишем

ИП0 2 ×

Так же можно поступать и с многозначными числами. К примеру, если второй сомножитель будет не 2, а 4,58, то соответствующий фрагмент программы выглядит так:

ИП0 4 , 5 8 ×

При этом для записи числа будут использованы четыре ячейки памяти, по одной на каждый символ. Чаще всего такая запись нецелесообразна, но если все адресуемые регистры заняты, а в программной памяти место есть, то такой способ записи чисел оказывается полезным. Перечислим для всех цифр их коды:

«0» (00), «1» (01), ..., «9» (09); укажем также коды специальных символов, используемых для записи чисел: «,» (0—), «ВП» (0С) — Ввод Порядка. К примеру, для записи в программу числа $1,6734 \cdot 10^{-27}$ (масса атома водорода в килограммах) нужно нажать клавиши «1», «,», «6», «7», «3», «4», «ВП», «2», «7», «/—/». (Нетрудно подсчитать, что в программе одно это число займет 10 ячеек.)

И, наконец, последняя команда этой группы — Сх (0Г). Она «стирает» содержимое регистра Х, или, иначе говоря, засылает в него число 0.

Стоит заметить, что ее действие не идентично засылке нуля в регистр Х с помощью цифровой клавиши «0». Команда Сх просто изменяет содержимое регистра Х, не затрагивая остальных регистров, при нажатии же клавиши «0» ввод нуля в регистр сдвигает все числа в стеке «снизу вверх», как при вводе любого нового числа.

Рассмотренные выше команды составляют уже достаточный минимум для написания несложных программ, выполняющих расчеты по последовательно записанным формулам. Однако очень часто встречаются задачи, для решения которых этого набора недостаточно. Ведь даже в таком элементарном случае, как решение квадратного уравнения, сначала требуется проверить знак дискриминанта квадратного трехчлена и в зависимости от него выбрать тот или иной путь решения, то есть вычислять либо действительные корни уравнения, либо действительную и мнимую части корней комплексных.

К счастью, в языке микрокалькулятора есть набор средств для проведения подобных операций. Они реализуются с помощью команд управления программой.

Сначала рассмотрим наиболее часто встречающуюся «команду останова» С/П (50). Она используется для остановки процесса вычислений, чтобы дать пользователю возможность либо прочесть полученный результат, либо ввести какие-нибудь числа или предписывающие команды с клавиатуры. В каждой программе обязательно должна быть хотя бы одна команда С/П — не может же программа работать бесконечно!

Команда безусловного перехода БП (51) передает управление команде, адрес которой записан сразу после нее. Фактически она занимает две смежные ячейки памяти. В первой — собственно БП, во второй — адрес перехода: две цифры. Кстати, обратите внимание, что две цифры подряд, нажатые после «БП», записываются одним кодом, совпадающим с этими цифрами.

Команды условного перехода также передают управление, но только при выполнении определенных условий. Действие

их можно описать так: если условие, записанное в команде, выполнено, то следует обрабатывать команду, написанную после команды перехода, в противном случае — передать управление по указанному в команде адресу. В качестве условия в этих командах используется сравнение R_X с нулем. Например, команда $Fx < 0 \ 23$ работает так. Проверяется содержимое регистра X . Если оно меньше нуля, то выполняется команда, записанная сразу после команды перехода, в противном случае — команда по адресу 23 *). Команд условного перехода четыре:

$$x < 0 (5Г), \quad x = 0 (5Е), \quad x \neq 0 (57), \quad x \geq 0 (59).$$

Так как названия их написаны над клавишами, то перед ними требуется нажимать клавишу «F».

Есть среди команд перехода четыре команды, специально приспособленные для организации циклов, то есть для многократного выполнения заданной последовательности команд. Это команды

$$L0 (5Г), \quad L1 (5Л), \quad L2 (58), \quad L3 (5—).$$

Перед ними также надо набирать клавишу «F». После любой из этих команд, как и после предыдущих, записывается адрес перехода. Работают они так. При каждом обращении к команде организации цикла из содержимого соответствующего регистра R_0, R_1, R_2 или R_3 вычитается единица. Если при этом получается не нуль, то управление передается команде по адресу перехода, если же результат вычитания равен нулю, то команде, записанной после адреса перехода. Таким образом, можно, записав в один из регистров $R_0—R_3$ некоторое число n , добиться выполнения определенного куска программы n раз.

Последней из команд перехода рассмотрим ПП (53) — Переход на Подпрограмму. Структура ее та же, что и остальных команд этой группы: сначала сама команда, потом — адрес перехода. По первому впечатлению команда ПП работает так же, как и БП: и та, и другая прерывают последовательность выполнения команд программы и «безусловно» передают управление по адресу, указанному в следующей после нее ячейке. Но если действие команды БП на этом закан-

*) По правде говоря, не очень понятно, почему команды условного перехода интерпретируются именно так. Было бы естественнее, если бы команда, скажем, $Fx = 0 \ 25$ читалась так: если $x = 0$, то перейти к адресу 25, а не как в микрокалькуляторе: если $x = 0$, то не переходить к адресу 25. Но уж как есть, так есть. Тем из читателей, кто программировал на других языках, остается только смириться с этим и зазубрить новое правило.

чивается, то команда ПП наряду с передачей управления записывает в программный стек адрес команды, следующей после нее. Поэтому когда подпрограмма выполнена, то по последней ее команде — а это обязательно должна быть команда В/О (52) — в программном стеке восстанавливается «старый» адрес, и программа продолжает работать с него.

Примеры использования различных команд переходов будут даны в следующих разделах.

Наконец, команда КНОП (54). Набирается она двумя клавишами «К» и «НОП» и является «пустой» командой. Она не совершает никаких действий. Употребляется чаще всего при отладке программ. Если выясняется, что какая-нибудь команда лишняя, то, чтобы не переписывать остальные, на ее место просто записывают «пустую» команду КНОП. При этом адресация всех остальных команд не меняется.

Команды косвенной адресации будут рассмотрены в разделе 7.

О командах, с которыми мы познакомились в этом разделе, советуем запомнить следующее.

1. Язык микрокалькулятора состоит из набора команд, названия которых написаны на клавиатуре. Чтобы использовать команды, названия которых даны над клавишами, нужно предварительно нажать клавишу «F», а перед командой НОП клавишу «К».

2. Каждая вычислительная команда размещается в одной ячейке памяти.

3. Результаты работы вычислительных команд всегда помещаются в регистр X. Исходные данные для одностепенных операций черпаются из этого же регистра, а для двуместных — из регистров X и Y.

4. Во всех операциях, записывающих информацию в регистры данных и извлекающих ее из них, обязательно участвует регистр X. Только его содержимое можно засылать в регистры данных и только в него можно помещать числа, извлекаемые из этих регистров.

5. Все команды перехода записываются в двух ячейках памяти: в одной — сама команда, в другой — адрес перехода. Адрес перехода обязательно набирается в виде двузначного числа, от 00 до 97.

6. При использовании вычислительных команд следите за тем, верно ли расположены в регистрах стека аргументы соответствующих операций, и постоянно помните об ограничениях, накладываемых на модуль аргумента. Нарушение этих ограничений приводит к останову работы программы и выводу на индикатор сообщения ЕГГОГ.

7. Старайтесь по возможности избегать употребления команды x^y . Работает она медленно, а ошибки при вычислениях дает большие.

5. БЛОК-СХЕМА – ПОРТРЕТ ПРОГРАММЫ

Язык микрокалькулятора, описанный в предыдущем разделе, позволяет создавать разнообразные и эффективные программы. Однако само по себе знание этого языка не гарантирует совершенства написанных на нем программ.

Здесь уместно сравнение с литературой. Знания какого-либо языка, скажем русского или английского, самого по себе недостаточно для того, чтобы писать на нем стихи. Требуется еще изучить законы стихосложения и, главное, иметь образное, поэтическое мышление, без которого стихи получаться не будут. Так и в программировании: для составления совершенных программ нужно знать некоторые законы и приемы, нужно освоить определенный программистский стиль мышления.

Мастерство программиста проявляется уже на том этапе анализа стоящей перед ним задачи, когда ищется метод ее решения. К сожалению, здесь почти невозможно дать общие рекомендации, поскольку разнообразие задач, решаемых в наши дни с помощью ЭВМ, поистине необозримо.

Следующий этап – выбор алгоритма. Напомним, что под этим термином в математике понимается точное предписание, определяющее процесс переработки исходных данных в искомый результат. Здесь мы поясним смысл и содержание этого термина, обратившись к решению квадратного уравнения вида $ax^2 + bx + c = 0$. Известны формулы, позволяющие находить его корни:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Разберемся, что играет в нашем примере роль исходных данных? Набор коэффициентов a , b , c . Чем определяется искомый результат? Двумя приведенными формулами. В чем заключается процесс переработки исходных данных? В вычислениях по этим формулам.

Даже на этом простом примере можно сформулировать некоторые соображения по поводу построения наилучших алгоритмов. Прежде всего следует по возможности упростить расчетные формулы, выделить повторяющиеся выражения и обозначить их промежуточными переменными. Такую работу мы называли приведением формул к «машинному» виду. Чи-

татель, ознакомившийся с предыдущими разделами книги и научившийся проделывать подобные преобразования, сможет провести их и на сей раз:

$$B = \frac{b}{2}; \quad d = B^2 - ac, \quad x_1 = \frac{-B + \sqrt{d}}{a}; \quad x_2 = \frac{-B - \sqrt{d}}{a}.$$

Как мы вскоре увидим, расчет по этим формулам не так прост, как может показаться на первый взгляд. При составлении программы, реализующей сложный вычислительный алгоритм, отличным вспомогательным средством служит так называемая *блок-схема* — своеобразный графический портрет алгоритма. Пользу от нее мы подчеркнули еще тогда, когда перечисляли этапы, из которых складывается решение той или иной задачи на ЭВМ: составление блок-схемы в этом перечне было выделено в особый этап. Даже опытные программисты — и те, как правило, начинают работу над программой с наброска блок-схемы выбранного алгоритма. При дальнейшей детализации блок-схема уточняется настолько, что перевод ее на язык команд требует минимального напряжения мысли.

Обо всем этом мы сейчас и поговорим.

Для того чтобы нарисовать блок-схему, не требуется особого дарования художника. Нужно уметь рисовать лишь четыре фигуры: овал, прямоугольник, параллелограмм и ромб. Таковы обозначения блоков, составных элементов любой блок-схемы. Каждый из них включает в себе один из этапов переработки данных.

В самом верху блок-схемы рисуется овал с надписью «Начало», в самом низу — другой овал. В нем написано «Конец». В каждой блок-схеме должен присутствовать один и только один блок «Начало» и один и только один блок «Конец». Все остальные блоки должны располагаться между ними.

Параллелограммы со словами «Ввод» и «Вывод» используются для того, чтобы указать, в каких местах программы нужно вводить исходные данные и в каких — выводить результаты. В прямоугольниках описываются вычисления. В них можно писать формулы, а можно и текст. Например, «Вычисление корней квадратного уравнения». Впрочем, формулы для его корней довольно просты, их можно написать сразу. Столь же просты и указания для ввода и вывода данных, необходимых для поиска корней. Всего этого достаточно, чтобы сделать набросок блок-схемы алгоритма, по которому мы станем решать квадратное уравнение (рис. 7).

Последовательно нарисованные прямоугольники можно объединять. К примеру, в нашей блок-схеме вычисления по

всем формулам можно вынести в отдельный блок (отмечено штриховкой).

Линии, соединяющие блоки, показывают, в какой последовательности выполняются различные этапы обработки данных. Направление движения по линии считается «положительным», если линия направлена вниз или вправо. В этом случае стрелок на линии ставить не обязательно. В иных случаях направление обязательно нужно указывать стрелками.

Предлагая вашему вниманию эту блок-схему, мы не накладывали никаких ограничений на величины a , b и c . Однако эти ограничения неявно предполагались. В частности, a не может равняться нулю. В этом случае записанные формулы «не работают». Человек-вычислитель учитывает подобные условия автоматически. В его памяти накоплено много информации, и пользуется он ею подчас безотчетно, как чем-то само собой разумеющимся. А в памяти машины имеется лишь то, что туда заложит человек-разработчик. Разработчики микрокалькулятора вложили в него предостережение: *делить на нуль нельзя*. И если машина будет поставлена перед необходимостью совершить эту недозволённую операцию, произойдет авост и на индикаторе загорится ЕГГОГ.

А между тем мы не можем гарантировать, что решаемое нами уравнение не введет нас в подобное положение. Ведь коэффициенты его могут возникнуть в результате работы какой-нибудь другой программы. И величина a при этом может оказаться равной нулю. Нужно обязательно научить нашего электронного помощника, как поступать в столь каверзных ситуациях! Иначе говоря, в алгоритме решения задачи должны быть предусмотрены реакции машины на все (абсолютно все!) возможные ситуации, которые при этом могут сложиться.

Давайте же попытаемся проанализировать все варианты исходных данных нашей задачи. Мы уже упомянули о случае, когда a равно нулю. Тогда приведенными формулами вообще нельзя пользоваться: ведь в этом случае уравнение уже не квадратное, а линейное (вырожденное, как говорят математики). Решаться оно должно по другим формулам, программу расчета по которым тоже надо ввести в машину. А значит,

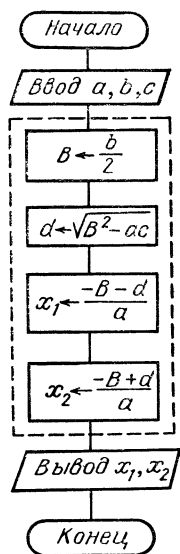


Рис. 7

в общей программе надо предусмотреть блок, в соответствии с которым машина бы проверяла коэффициент a на равенство нулю и в зависимости от этого вела расчет по тем или иным формулам.

Далее. Может статься, что и $a = 0$, и $b = 0$. Тогда из уравнения выпадает неизвестная величина x , и решать его уже не имеет смысла. Нужно научить машину реагировать и на такое сочетание коэффициентов.

Теперь займемся случаем, когда $a \neq 0$. Заметим сразу: этого неравенства еще не достаточно, чтобы без опаски вести расчеты по формулам, выписанным в самом начале раздела. Дискриминант квадратного трехчлена d может оказаться отрицательным. Тогда квадратное уравнение будет иметь два комплексных корня, вычисляемых по своим формулам: отдельно действительная часть (она у обоих корней одинакова) и мнимая (в одном случае — со знаком плюс, а в другом — со знаком минус).

Итак, сравнение коэффициента a с нулем разветвляет нашу блок-схему надвое, и каждая из ветвей также разветвляется на два направления. В точках ветвления, подобно стрелкам на железнодорожных путях, ставятся блоки сравнения. Они изображаются ромбом, внутри которого записана операция сравнения. Выходят из ромба две линии. Это — два возможных пути. Один помечен словом «Да» (сюда надо свернуть, если сравнение выполняется), другое — словом «Нет» (повернуть туда надо, если сравнение не выполняется).

Введем в нашу блок-схему все оговоренные выше операции сравнения, и тогда она примет вид, показанный на рис. 8. Чтобы не перегружать ее, мы не стали анализировать бессмысленную и весьма маловероятную ситуацию, когда все три коэффициента равны нулю.

(Можно было бы ввести еще одно дополнительное сравнение $d = 0$. Хотя решение в этом случае не отличается от ситуации $d > 0$, результат обладает примечательной особенностью: корни уравнения совпадают. Математики в таком случае говорят, что уравнение имеет кратный корень. В некоторых технических приложениях этот случай очень важен. Например, когда исследуется механическая колебательная система, то частоты колебаний отыскиваются как решение некоторого уравнения. И если какой-то из его корней оказывается кратным, это означает не только то, что совпадают частоты некоторых колебаний; может случиться, что амплитуды этих колебаний станут неограниченно нарастать, грозя разрушить систему. Именно такова природа коварного врага авиаторов — флаттера. Если совпадают частоты крутильных и изгибных колебаний

самолетного крыла, оно раскачивается все сильнее и в конце концов ломается.)

Как видим, исчерпывающий анализ квадратного уравнения, такого простого и привычного, — дело довольно сложное, в значительной степени диктуемое целями решения задачи.

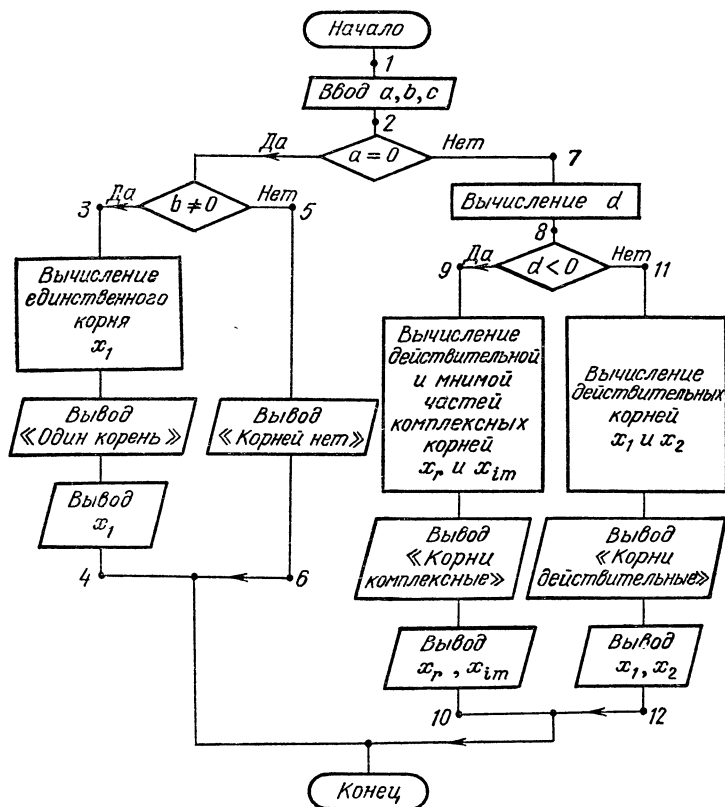


Рис. 8

Наша цель — учебная, поэтому мы ограничимся приведенной блок-схемой. Не правда ли, она напоминает дерево? Только лучше считать, что поставлено оно «с ног на голову». Корень его наверху — это «Начало». Потом ствол разделяется на ряд ветвей, и внизу, то есть на верхушке, подобно звезде на новогодней елке — «Конеч».

Достоинства такого представления алгоритма налицо. Легко проконтролировать его правильность: все ветви должны быть

замкнуты, то есть каждая линия обязательно должна заканчиваться на каком-нибудь блоке; из каждого прямоугольника — только один выход, из ромба — два; в кружок «Начало» нет ни одного входа, из кружка «Конец» — ни одного выхода. Все указания, написанные в элементах блок-схемы, понятны, просты, удобны и избавляют составителя программы от необходимости хранить в своей памяти информацию о том, что делать, если ... Кстати, попробуйте описать этот алгоритм столь же подробно словами. Вы сразу убедитесь, насколько это сложно по сравнению с графическим способом описания алгоритма.

Прежде чем приступить к написанию программы по блок-схеме, ее нужно детализировать, заменив словесные описания последовательностью формул. Чтобы различать отдельные части блок-схемы, мы пометили некоторые ее узлы цифрами. (Вообще принято пометать не узлы, а блоки, однако мы допустим эту вольность для удобства дальнейшего изложения.)

Детализация одной из ветвей, той, что лежит между узлами 11–12, практически проведена: сюда надо просто вставить формулы из предыдущего эскизного варианта. Для ветви 5–6 никаких формул не надо — вся работа на этом этапе заключается в выводе сообщения «Корней нет». Осталось заполнить формулами две ветви.

Для первой из них, ветви 3–4, требуется всего одна формула

$$x_1 = -\frac{c}{b}.$$

А вот формулы для последней ветви, 9–10:

$$B = \frac{b}{2}; \quad d = B^2 - ac; \quad x_r = -\frac{B}{a}; \quad x_{im} = \frac{\sqrt{-d}}{a}.$$

Здесь x_r и x_{im} — действительная и мнимая части комплексных корней, которые с помощью так называемой мнимой единицы, величины $j = \sqrt{-1}$, выражаются формулами

$$x_1 = x_r + jx_{im}, \quad x_2 = x_r - jx_{im}.$$

Если сравнить теперь формулы для ветвей 9–10 и 11–12, то видно, что между ними много общего. Для блок-схемы это безразлично, а вот для программы означает, что одни и те же последовательности команд будут написаны дважды, следовательно, общая длина программы увеличится. Целесообразно выполнять общие для каких-то ветвей вычисления еще до разделения этих ветвей, в стволе дерева. С учетом этих требований блок-схема примет вид, показанный на рис. 9.

Вот теперь с помощью блок-схемы уже можно писать программу. Важно отметить, что писать ее можно для любой ЭВМ и на любом языке программирования. Это своего рода запись мелодии, которую можно аранжировать для любого инструмента с учетом его специфики.

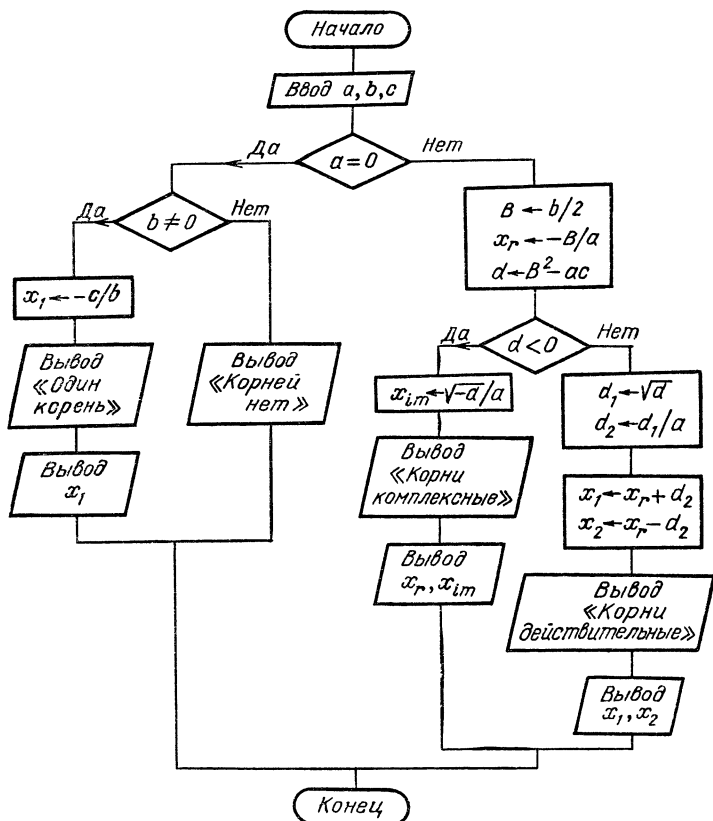


Рис. 9

Когда программа пишется для микрокалькулятора, то его специфика проявляется, в частности, в двух аспектах. Во-первых, у него разделены области памяти для хранения программ и данных. Во-вторых, ПМК оперирует только цифрами — буквенных символов в его языке нет. В силу первой причины приходится вручную распределять информацию по регистрам (при программировании на языках высокого уровня это делается автоматически), а вторая заставляет шифровать цифрами

сообщения об особенностях решения (в нашем случае — о количестве и природе корней).

С первой загвоздкой мы справляемся довольно быстро. Предварительно намечаем, что переменные будут распределяться в регистрах так:

$a \rightarrow RA$ x_1 или $x_r \rightarrow R1$ и RX

$b \rightarrow RB$ x_2 или $x_{im} \rightarrow R2$ и RY

$c \rightarrow RC$

$d \rightarrow RD$

Предварительной такая наметка является потому, что в процессе составления программы могут понадобиться какие-нибудь дополнительные регистры или, наоборот, какие-либо из запланированных регистров могут оказаться лишними.

Придумать систему «шифров» для необходимых нам сообщений тоже вроде бы легко. Скажем, появление на индикаторе нуля означает «Корней нет», появление единицы — «Имеется один корень» и т. д. Часто так и поступают. Однако у этого метода есть существенный недостаток: можно спутать зашифрованное сообщение с результатом вычислений. Мы предлагаем другой путь. Чтобы пойти по нему, надо приоткрыть дверцу в «секретную область» микрокалькулятора, то есть сказать несколько слов о возможностях микрокалькулятора, не описанных в инструкции к нему.

Вам уже известно, что в микрокалькуляторе используются символы шестнадцатеричных цифр, которые не спутаешь ни с одной десятичной. Оказывается, есть возможность, формально выполняя некоторые ошибочные операции, получать на индикаторе и запоминать в адресуемых регистрах комбинации этих символов с обычными десятичными цифрами. Если выбирать их в качестве сообщений, то ни с одним десятичным числом их уже не спутать. Как получать подобные комбинации, мы скажем позже, а пока договоримся использовать такие шифры: E00 — «Корней нет», E01 — «Один корень», E02 — «Два действительных корня» и Г. — «Корни комплексные».

Для хранения шифров тоже нужны регистры. Поэтому в дополнение к предварительному распределению памяти запишем:

$E00 \rightarrow R0$ $E02 \rightarrow R4$

$E01 \rightarrow R3$ Г. $\rightarrow R5$

Затем нужно продумать организацию ввода и вывода. Можно, конечно, вводить в режиме вычислений значения коэф-

фициентов сразу в соответствующие адресуемые регистры и читать результаты, вызывая на индикатор содержимое других регистров после останова. Однако большое число операций, требующихся для этого, и необходимость постоянно помнить, что куда вводить и что откуда выводить, резко увеличит общее время получения результата, да и возможность ошибок при вводе будет велика. Лучше минимизировать долю подобных операций, организовав ввод и вывод таким образом, чтобы введенные числа автоматически рассылались по нужным регистрам и чтобы для прочтения результатов приходилось бы нажимать как можно меньше клавиш. Мы еще остановимся на таком порядке ввода и вывода. Коэффициенты уравнения вводятся в их естественной последовательности: a , b , c . После ввода тройки этих чисел нажимаем клавишу «С/П». После останова калькулятора видим на индикаторе шифрованное сообщение о характере результата; после нажима клавиши «С/П» и следующего останова — на индикаторе один корень и после нажима клавиши «ХУ» — второй, если он есть.

Теперь все технические требования к программе изложены, и можно приступить к ее созданию.

Рекомендуем первый вариант программы писать так, как показано на табл. 4, то есть кроме команд, их адресов и кодов указывать еще содержимое регистров стека — хотя бы не всех, а только тех, которые могут понадобиться в дальнейшем. Правее желательно оставить еще одну колонку для кратких примечаний. Они помогут ориентироваться в программе, поскольку иной раз легче написать новую программу, чем разобраться в старой, особенно если не вы сами ее составляли. В этом примере мы используем подробные примечания, поэтому они даны после текста программы. Как писать краткие примечания, будет показано в одном из следующих примеров.

Ввод a . Эта операция выполняется перед пуском программы. Величина a набирается на клавиатуре. Набор заканчивается нажатием клавиши «С/П».

00. Запись a в RA.

01. «Останов» для ввода b . Набираем это значение на клавиатуре и снова нажимаем «С/П».

02. Ввод третьего параметра уравнения, коэффициента c . Теперь клавиша «С/П» запускает программу на счет.

03. Значение a хотя и находится в стеке, но слишком далеко. Чем несколько раз «гонять» числа по стеку, проще вызвать a из его «собственного» регистра.

04—05. В стеке ничего не меняется. Мы лишь проверили, равно ли нулю содержимое регистра X. Если «да», то есть

Таблица 4

Адрес	Команда	Код	Регистр				
			X	Y	Z	T	X1
ВВОД a			a				
00	ПА С/П	4—	a				
01		50	a				
ВВОД b			b	a			
02	С/П	50	b	a			
ВВОД c			c	b	a		
03	ИПА $Fx = 0$	6—	a	c	b	a	
04		5C					
05	22	22					
06	ФО	25	c	b	a	a	
07	ХУ	14	b	c	a	a	
08	$Fx \neq 0$	57					
09		18	18				
10	\div	13	c/b	a	a	a	
11	/—/	0L	x_1				
12	ИПЗ	63	E01	x_1			
13	С/П	50					
14	ХУ	14	x_1	E01			
15	С/П	50					
16	БП	51					
17	59	59					
18	ИПО	60	E00				
19	С/П	50					
20	БП	51					
21	59	59					
22	\times	12	ac	b	a		
23	/—/	0L	$-ac$	b	a		
24	ХУ	14	b	$-ac$	a		
25	2	02	2	b	$-ac$	a	
26	\div	13	$b/2$	$-ac$	a		
27	ПВ	4L	B	$-ac$	a		
28	Fx^2	22	B^2	$-ac$	a		
29	+	10	d	a			
30	ИПВ	6L	B	d	a		
31	ИПА	6—	a	B	d	a	
32	\div	13	B/a	d	a		
33	/—/	0L	$-B/a$	d	a		
34	П1	41	x_r	d	a		
35	ХУ	14	d	x_r	a		
36	$Fx < 0$	5C					
37	48	48					
38	/—/	0L	$-d$	x_r	a		
39	$F\sqrt{}$	21	$\sqrt{-d}$	x_i	a		

Адрес	Команда	Код	Регистр				
			X	Y	Z	T	X1
40	ИПА	6—	a	$\sqrt{-d}$	x_r	a	
41	÷	13	$\sqrt{-d/a}$	x_r			
42	ИП5	65	Γ	x_{im}	x_r		
43	С/П	50					
44	FO	25	x_{im}	x_r			
45	С/П	50					
46	БП	51					
47	59	59					
48	$F\sqrt{\quad}$	21	\sqrt{d}	x_r	a		
49	ИПА	6—	a	d_1	x_r		
50	÷	13	d_1/a	x_r			
51	+	10	x_1				d_1/a
52	ИП1	61	x_r	x_1			
53	FBx	0	d_1/a	x_r	x_1		d_1/a
54	—	11	x_2	x_1			
55	ИП4	64	E02	x_2	x_1		
56	С/П	50					
57	FO	25	x_2	x_1			
58	С/П	50					
59	БП	51					
60	00	00					

уравнение вырожденное, будем выполнять команду по адресу 06. Если «нет», перейдем к команде, записанной по адресу 22.

06—07. Две команды использованы только для того, чтобы поместить в RX значение b . Можно было бы обойтись и одной ИПВ. Но надо же «помнить о будущем». Ведь скоро придется делить c на b . А при таком распределении величин по стеку, как у нас теперь, все для этого уже готово.

08—09. Если $b = 0$, то перейдем к команде по адресу 18 (на ветвь 5—6), иначе — по адресу, записанному сразу после команды перехода, то есть 10.

10—15. Команды по этим адресам реализуют вычисления, записанные в ветви 3—4.

10. Вот где пригодилась вроде бы лишняя команда (06 и 07 вместо одной ИПВ). Теперь мы сэкономим две команды: вызов величины c и перемену местами содержимого регистров RX и RY. Точнее, даже три. Ведь чтобы вызвать величину c из адресуемого регистра, ее надо было бы предварительно

туда записать (ПС). Мы же пока обходимся без записи величины s . Она к нашим услугам прямо в стеке.

11. Изменяем знак у частного, и в регистре X получаем корень x_1 .

12–13. Вычисления по ветви 3–4 закончены. Вызываем в регистр X сообщение $E01$ из $R3$ и останавливаем программу, чтобы его можно было прочесть. Иначе говоря, реализуем блок «Вывод. Один корень».

14–15. После нажатия клавиши «С/П» реализуем блок «Вывод x_1 ». Значение корня видим на индикаторе.

16–17. Записанная под этими адресами команда безусловного перехода замыкает ветвь 3–4, управление передается на последнюю команду программы – ту, что соответствует блоку «Конец». Все остальные ветви обходятся, и работа программы заканчивается.

18–21. Ветвь 5–6. Сюда мы попадаем, если $a = 0$ и $b = 0$. Вычислений никаких проводить не надо, просто выводим на экран сообщение $E00$, что означает «корней нет», и замыкаем ветвь подобно предыдущей.

22–24. Ветвь 7–8. Здесь мы вычисляем величины, нужные для получения пары корней квадратного уравнения.

22. Если уж мы вводим эту команду, значит, уравнение невырожденное. Надо вычислять дискриминант, а потом корни по одной из двух ветвей. Кстати, вас не смущает, что после команды по адресу 10 мы расстались с величиной s ? Ведь она у нас находилась лишь в стеке и ни в одном адресуемом регистре не была записана. Но не волнуйтесь, все в порядке. Если мы вводим команду по адресу 22, то делаем это сразу после команды по адресу 03, а все промежуточные команды между ними не выполняются. Потому и содержимое стека такое же, как и до команды перехода. Все готово для умножения a на s . Вот после этой команды величина s потеряна для нас навсегда. Но она больше и не нужна.

24. Выдвигаем величину b на первый план – переводим из регистра Y в регистр X .

25–27. Вводим в RX число 2, делим на него b и записываем результат в RB .

28–29. Величина B возведена в квадрат, дискриминант вычислен. Но прежде чем перейти к его анализу, нужно получить планируемую по схеме величину x_2 , так как она понадобится впоследствии в двух ветвях.

30. Извлекаем величину B из хранилища RB .

31. Для деления нужна величина a . Она уже в стеке. Но, как говорится, «близок локоть, да не укусишь». Ведь мало просто перевести ее в RX , нужно, чтобы при этом величина B

находилась в RY. Оперируя только стеком для вызова величины a , нужно было вместо команд 30—31 употребить команды FO ИПВ ХУ. Проще поступить, как у нас: вызвать a в стек заново.

32—34. Теперь все вычисления по ветви 7—8 закончены. Величина x_r отправлена на хранение в R1, можно переходить к анализу величины d , благо она рядом, в регистре Y.

35—37. Делаем последнее сравнение в программе: если $d \geq 0$, то корни действительные, и надо перейти на ветвь 9—10 (команда 48). Если же $d < 0$, то корни комплексные, надо вычислять их по ветви 11—12.

38—41. Ветвь 9—10, вычисление мнимой и действительной частей комплексных корней.

38—39. Так как величина d меньше нуля, то, чтобы вычислить корни, надо сначала изменить ее знак.

40. Для вычисления x_{im} нужна величина a . Проще всего вызвать ее из RA. Теперь все готово для деления $\sqrt{-d}$ на a .

41. Мнимая часть комплексных корней вычислена.

42—43. Все готово для вывода результата. Значения x_{im} и x_r доступны. Можно останавливать микрокалькулятор и читать их с индикатора. Но нельзя забывать об удобстве работы с программой. Ведь мы еще не вывели сообщение о том, что за величины получены. Поэтому приходится отодвигать готовые результаты и считывать с RX шифрованное сообщение: Г.— «Корни комплексные».

44—45. Сообщение выведено. Снова передвигаем результаты в стеке на старое место и останавливаем программу, чтобы считать их.

46—47. Ветвь 9—10 замкнута. Переходим к описанию вычислений по последней ветви.

48—60. Вот мы и добрались до нее: ветвь 11—12, вычисление действительных корней.

48. Так как d находится в RX (вернитесь к команде 35 и просмотрите содержимое стека), то сразу же извлекаем из него квадратный корень: $d_1 = \sqrt{d}$.

49—50. Вычисление вспомогательной величины $d_2 = \sqrt{d/a}$.

51. Получаем первый корень x_1 . Жаль, что после сложения исчезает величина x_r , зато d_2 — в зоне досягаемости, в регистре предыдущего результата X1.

52—54. Вычисляем второй корень x_2 . Вычислительная работа программы закончена.

55—58. Вывод результатов этой ветви полностью аналогичен выводу результатов предыдущей.

59—60. Вот и последние команды программы. Это своеобразная реализация блока «Конец». Фактический же смысл этих команд — в подготовке программы для приема новой информации, то есть в передаче управления ее началу. Теперь можно вводить новые данные и повторять расчет.

Займемся вновь распределением памяти. При составлении программы предварительные наброски несколько изменились. Окончательная картина распределения информации по регистрам такова:

$$\begin{array}{ll} a \rightarrow RA & x_r, x_1 \rightarrow RY \\ B \rightarrow RB & x_{im}, x_2 \rightarrow RX \\ x_r \rightarrow R1 \end{array}$$

Три регистра удалось сэкономить. Смысл этой экономии в данном примере не очевиден. Однако в оставшуюся часть программной памяти можно записать еще какую-нибудь программу. Тогда «лишние» регистры будут весьма кстати.

Теперь, как и было обещано, о получении «шифrogramм». Сообщение E00 получается, если в режиме вычислений выполнить следующие действия: набрать 100 ВП 99. При этом на индикаторе появится надпись ЕГГОГ. Не смущаясь, продолжаем: ВП ↑. На индикаторе то, что надо: E00. Достаточно нажать П0, и шифrogramма записана в R0.

Сообщения E01 и E02 получаются аналогично. Только вместо числа 100 надо набрать 101 или 102 соответственно. Алгоритм для получения сообщения Г. другой: $Sx \uparrow \div$ (здесь опять появляется ЕГГОГ, ведь делится нуль на нуль), ВП ВП ↑. На индикаторе Г.; эту величину мы записываем в R5.

Оглядим еще раз внимательно творение наших рук. Не слишком ли большой получилась программа для решения элементарного уравнения? А если уравнение будет посложней? Что делать тогда?

Эти вопросы не должны вызывать смущение. Во-первых, не такую уж элементарную задачу мы решили. Создана программа, которая не просто решает квадратное уравнение по известным формулам, но и анализирует возможность их использования, выбирает тот или иной путь решения, тем самым избавляя нас от предварительного анализа. И, если уж на то пошло, фактически написано четыре разных программы, каждая из которых рассматривает отдельный случай решения, плюс еще программа, которая выбирает одну из этой четверки. Это не так уж мало.

А во-вторых, программа действительно получилась слишком длинная. Ее можно сократить, и довольно значительно, не

поступившись при этом ни возможностями ее, ни удобством использования. Вопрос о том, нужно ли сокращать программы, а если нужно, то как, будет рассмотрен в одном из следующих разделов.

Теперь зададимся вопросом: как пользоваться созданной программой? Если вы писали ее сами, то вам это известно. Хотя осмелимся утверждать, что через некоторое время и вы, скорее всего, это забудете. Поэтому правила пользования программой должны быть изложены в специальной инструкции к ней. В наш век, когда обработка информации становится отраслью промышленности, а программа — изделием, необходимость инструкций для них так же очевидна, как и для любого другого изделия, будь то телевизор или утюг.

Согласно современным взглядам на программирование, инструкцию необходимо хранить в самой программе. Так чаще всего и поступают при работе на больших ЭВМ или персональных компьютерах. Хорошая программа сама дает подсказки, что нужно предпринимать для успешной ее работы: какие данные и в каком порядке вводить, как интерпретировать выходную информацию и т. д.

Из-за ограниченности возможностей микрокалькулятора решение любой задачи на нем не полностью автоматизировано. Оно реализуется совместными усилиями человека и микрокалькулятора. Человек при этом становится полноправной частью вычислительной системы, а значит, для него тоже нужно писать программу. Этой программой и является инструкция.

Вот как может выглядеть инструкция для работы с составленной нами программой решения квадратного уравнения.

1. Ввести программу.
2. Установить режим вычислений (FABT).
3. Ввести: 100 ВП 99 ВП ↑ П0
101 ВП 99 ВП ↑ П3
102 ВП 99 ВП ↑ П4
Сх ↑ ÷ ВП ВП ↑ П5
4. Очистить программный указатель (В/О).
5. Ввод исходных данных:
а С/П б С/П с С/П.
6. Вывод; после первого «останова» на индикаторе:
Е00 — корней нет,
Е01 — уравнение линейное, корень только один,
Е02 — два действительных корня,
Г. — корни комплексные.

7. Если корней нет, то для продолжения расчетов перейти к п. 5. Если корни есть, то нажать «С/П». После останова на индикатор выводится значение первого корня (если они действительные) или мнимой части комплексных корней (если они таковы). Для чтения другого корня или действительной части комплексных корней нажать «ХУ».

8. Для продолжения расчетов перейти к п. 5.

Контрольный пример.

Ввод: $a = 2, b = 5, c = 3;$	Вывод: E02, $-1,5, -1;$
$a = 1, b = -4, c = 5;$	Г., $1, 2;$
$a = 0, b = 8, c = 3;$	E01, $-3,75 \cdot 10^{-1};$
$a = 0, b = 0, c$ любое.	E00.

Подчеркнем еще раз, что инструкция — это тоже программа, только команды ее должны выполняться человеком. Он должен в определенном порядке нажимать клавиши, считывать и запоминать (или записывать на бумаге) результаты.

Разберем теперь эту инструкцию подробнее. Программа для человека заслуживает того не менее, чем программа для калькулятора.

1. Этот пункт предполагает начальное знакомство пользователя со своей ЭВМ. Ведь что такое ввести программу? Нужно предварительно включить калькулятор, затем установить режим ГПРГ и уж затем начать ввод, последовательно нажимая клавиши, как указано в тексте программы. Если же калькулятор уже был включен и работал по какой-нибудь другой программе, то надо еще и очистить программный указатель, введя в режиме вычислений команду В/О.

2. Команда FАВТ переводит калькулятор в режим вычислений, и теперь он может принимать команды, отдаваемые ему с пульта управления. На этом этапе в регистры данных вводится информация, нужная для решения задачи.

3. Здесь конкретизируются операции по вводу служебной информации. Подробно процесс получения и записи шифрованных сообщений описан выше.

5. Ввод исходных данных. Это очень ответственный этап. Данные должны быть введены в строгом соответствии с инструкцией. Ясно, что нарушение этого порядка приведет к неверным результатам счета.

6—7. Здесь описано, как интерпретировать полученную информацию. Об этом достаточно подробно говорилось в тексте.

Контрольный пример. Трудно сказать, какой из пунктов инструкции более важен. Наверное, нельзя обойтись ни без одного. У каждого из них свой смысл. И все-таки последний пункт выделим особо. Смысл его — в проверке работоспособ-

ности программы. Контрольный пример не имеет отдельного номера в инструкции: вводить его надо в соответствии с описанными выше пунктами. Роль его велика для любой программы на любой ЭВМ. Но если при работе на большой ЭВМ программа записывается на какой-либо постоянный носитель — магнитную ленту, магнитный диск, перфокарты и т. д., а в некоторых видах калькуляторов — на магнитные карты или постоянно хранится в неразрушаемой при выключении памяти и потому контрольный пример вводится один раз, то в нашем микрокалькуляторе «Электроника БЗ-34» дело обстоит не так.

Программу после каждого выключения микрокалькулятора приходится вводить заново вручную. Ошибки при этом весьма вероятны. Чтобы убедиться в их отсутствии, необходимо каждый раз решать контрольный пример. Казалось бы, при этом можно обойтись одним комплектом данных. Чаше всего так и поступают. Однако если программа «ветвистая», то далеко не всегда один пример позволяет проверить все ветви. Таков и наш случай. Поэтому и приходится использовать четыре примера — по числу ветвей. И только когда все примеры дадут правильные результаты, можно приступать к практической работе с программой.

Теперь, как всегда, подведем итоги.

1. Написанию программы всегда должно предшествовать составление блок-схемы ее алгоритма.

2. Целесообразно составлять две блок-схемы: принципиальную и детализированную. Первая выражает общую структуру программы, ее логику, деление программы на ветви и т. д., вторая раскрывает подробное содержание всех ее ветвей, и поэтому вполне пригодна для перевода ее на язык программирования.

3. При записи программы для микрокалькулятора рядом с каждой командой обязательно указывайте заполнение регистров стека. Это позволит создать более быстродействующую и компактную программу.

4. Не экономьте время на запись комментариев. С их помощью значительно легче отлаживать программу и оптимизировать ее.

5. Инструкция по использованию программы должна рассматриваться как необходимая ее часть. Без нее пользование программой невозможно.

6. К каждой программе должен быть составлен *тест* — совокупность контрольных примеров, с помощью которых можно убедиться в работоспособности всех частей программы.

6. ВЕТВИ, ЦИКЛЫ, ПОДПРОГРАММЫ

За тремя словами названия этого раздела стоят очень важные приемы программирования, без которых не обходится практически ни одна программа, разве что самые элементарные, которые сводятся к последовательным вычислениям по цепочке команд — от первой до последней.

Мы уже имели возможность убедиться в этом на собственном опыте. Когда в предыдущей главе анализировался процесс решения квадратного уравнения $ax^2 + bx + c = 0$, то выяснилось, что решать его приходится по-разному: если коэффициент a не равен нулю, то по одной формуле, известной формуле корней квадратного уравнения; если равен — по другой, поскольку уравнение тогда превращается из квадратного в линейное. Каждый из случаев, в свою очередь, делился на два подслучая. Формула корней квадратного уравнения в зависимости от знака дискриминанта давала либо действительные, либо комплексные результаты; формула для корня линейного уравнения имела смысл при ненулевом коэффициенте b и теряла его при нулевом.

В блок-схеме алгоритма, использованного нами для решения квадратного уравнения, возможность выбрать тот или иной способ решения каждый раз выражалась разветвлением схемы. Ветви неминуемо появляются, когда в зависимости от сочетания параметров задачи требуется пойти по тому или иному пути расчета, провести вычисления по одним или другим формулам.

Ветви всегда растут попарно, выходя из одной «почки», роль которой играет блок сравнения. На блок-схеме решения квадратного уравнения (рис. 8) первое ветвление произошло в ромбе, содержащем сравнение $a = 0$. При неравном нулю коэффициенте a следующее ветвление было обусловлено сравнением $b^2 - 4ac > 0$. При $a = 0$ дальнейшее ветвление диктовалось сравнением $b \neq 0$.

Рисуя блок-схему на листе бумаги, мы вольны разбегаться во всю его ширь. Но когда мы начинаем писать программу, то все команды необходимо выстроить в одну линию, последовательно их нумеруя. Ветви приходится разрывать и укладывать одну за другой.

Впрочем, слово «разрывать» здесь не совсем точное. Линии, расходящиеся в стороны от ромбовидного блока сравнения, лучше представить себе не жесткими, а гибкими, вроде растягивающихся пружин или резинок. Такими же «резинками» будем воображать и линии, сходящиеся в точке смыкания ветвей. Растягивая резинки, ветви можно расположить последо-

вательно, одну за другой, не разрывая связей. Будем иметь это в виду, составляя ветвистую программу, переводя на язык команд операции, записанные в блоках.

Вернемся к программе из предыдущего раздела и поясним этот процесс перевода на ее примере.

Первый блок сравнения, встречающийся на нашем пути — это $a = 0$. Записываем команду, соответствующую поставленной в нем операции (04. $Fx = 0$), а следующую ячейку (05) оставляем пока пустой. Чем ее заполнить, мы узнаем несколько позже. Потом пишем команды той ветви, по которой нужно следовать при выполнении условия, определяющего разветвление. Переход на первую из этих команд — растяннутая «резинка», исходящая из блока сравнения. Так уж устроена каждая команда сравнения, выполняемая нашим микрокалькулятором: при соблюдении закодированного ею условия управление передается на следующую команду, расположенную через ячейку после нее. Если выбранная нами ветвь не последняя, то, закончив ее перевод на командный язык, ставим команду безусловного перехода (16. БП). Следующую ячейку (17), где должен содержаться адрес перехода, тоже оставляем пустой: этот адрес зависит от длины цепочки команд, образующих другие ветви. Далее записываем следующую ветвь. Адрес, с которого она начинается (22), заносим в оставленную пустой ячейку (05) после команды сравнения. Переход на этот адрес — растяннутая «резинка», выходящая из блока сравнения.

Точно таким же образом переведем на язык команд все ветви, расположенные одна за другой. После этого можно заполнить зарезервированные ячейки после команд БП. Адрес, записанный в этих ячейках (59), — это растяннутая «резинка» из тех, что сходятся в точке смыкания.

Как видно из нашего описания, общая длина программы складывается из суммы длин ее ветвей. Поэтому чтобы сократить программу, целесообразно общие для разных ветвей последовательности команд по возможности выносить в «ствол», предшествующий точке ветвления либо находящийся за точкой смыкания ветвей. Время выполнения ветвистых программ зависит от пути, по которому надо пройти для получения результата. Максимальное время определяется, естественно, самым длинным путем.

Программы, состоящие только из ветвей и в этом подобные решению квадратного уравнения, — вещь довольно редкая. Чаще ветви соседствуют с циклами.

Циклом в программировании называется неоднократное выполнение одной и той же последовательности команд программы. В виде цикла реализуется всякий метод после-

довательного приближения. А такие методы в вычислительной математике встречаются сплошь и рядом, идет ли речь о нахождении корней функции или решении системы уравнений, о вычислении интегралов или о статистических расчетах.

Поясним понятие цикла на примере уже однажды затронутой проблемы нахождения корней функции; частным случаем ее было решение квадратного уравнения — нахождение корней квадратного трехчлена. Сейчас мы познакомимся с методом, пригодным для функций весьма разнообразного вида — алгебраических, тригонометрических и многих других. Называется он *методом половинного деления*. Чтобы подойти с этим универсальным методом к какой-либо функции, от нее требуется немного: во-первых, чтобы она меняла знак на концах некоторого отрезка и, во-вторых, чтобы на этом отрезке она была непрерывна*).

Метод половинного деления — не самый быстрый среди тех, при помощи которых можно решать уравнения. Но он, пожалуй, самый надежный, так как довольно «равнодушен» к накоплению ошибок, да и серьезных требований к функциям не предъявляет.

Идея метода половинного деления, или, как его называют, *дихотомии*, очень проста (рис. 10). Исходный отрезок делится пополам. В средней точке вычисляется значение функции. Затем отбрасывается та половина отрезка, на концах которого функция не меняет знака. Остаток вновь делится

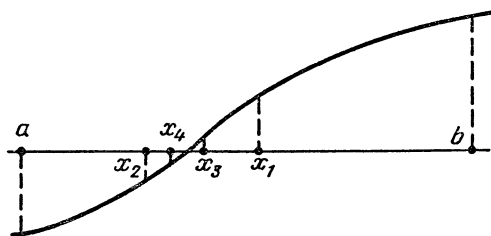


Рис. 10

пополам, и процесс этот повторяется до тех пор, пока не получится отрезок меньше некоторого наперед заданного числа ε . Тогда, выбрав в качестве корня любую точку этого отрезка, говорят, что корень функции определен с точностью ε . Может

*) Если на отрезке находится несколько корней, то заранее утверждать, какой из них будет найден, невозможно. Правда, какой-нибудь корень будет найден всегда. Чтобы не обсуждать этот вопрос, предположим, что на искомом отрезке находится только один корень функции.

статься, что при одном из делений мы точно попадем в корень, но это почти невероятно. Во-первых, как правило, численными методами решаются сложные уравнения, корни которых не могут быть выражены конечными десятичными дробями, во-вторых, вычисления всегда происходят с некоторыми погрешностями, поэтому «точный» корень не может быть их результатом.

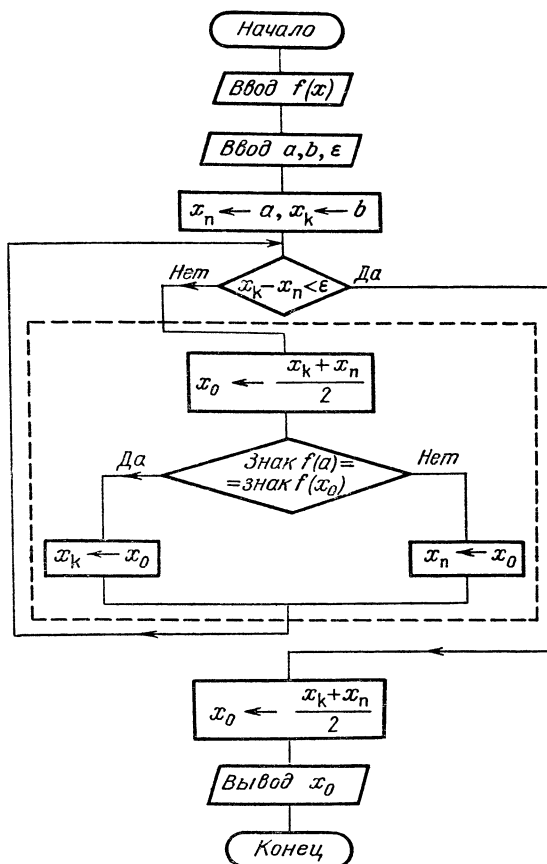


Рис. 11

Составим теперь блок-схему изложенного алгоритма (рис. 11). Оставим пока без пояснений первый блок «Ввод $f(x)$ » (об этом после). Следующий этап — ввод в память микрокалькулятора величин: ε — погрешности вычисления корня, a и b —

концов отрезка, который будет делиться в поисках корня. Концы отрезка, выбранного после очередного деления, обозначим через x_k и x_n . Присвоим им вначале значения a и b соответственно.

Может, вообще говоря, оказаться, что a и b отстоят друг от друга менее, чем на ε . Тогда вся дальнейшая работа по поиску корня окажется ненужной. В качестве его значения в этом случае можно выбрать любую точку отрезка. Вполне ясно, что она удалена от корня не более, чем на ε , и потому обеспечивает заданную точность. Впрочем, мы будем полагать, что $b - a > \varepsilon$.

Проверяем условием $x_k - x_n > \varepsilon$. Если оно не выполняется, задаем значение корня серединой отрезка, $x_0 = (x_k + x_n)/2$ (пополам, так пополам!) и выводим найденный результат. Если выполняется, будем вести поиск. Метод поиска был описан весьма подробно, так что составление блока-схемы не должно составить труда для читателя, не забывшего содержание предыдущего раздела. Поэтому изобразим нашу блок-схему сразу в готовом виде (рис. 11).

Фрагмент, заключенный в штриховую рамку, носит особое название — *тело цикла*. Так называется последовательность команд, которая выполняется многократно в зависимости от выполнения (или невыполнения) некоторого условия (в нашем случае неравенства $x_k - x_n > \varepsilon$).

Нелишне сравнить нарисованную блок-схему метода половинного деления с блок-схемой алгоритма решения квадратного уравнения из предыдущего раздела. Не правда ли, новая схема значительно проще? Забегая вперед, скажем, что и программа на сей раз будет значительно короче, чем для решения квадратного уравнения. А ведь задача-то намного сложнее: мы намереваемся определять корни любой функции, а не только квадратного трехчлена, для которого, кстати, их тоже можно отыскать методом половинного деления, если они действительно и известны отрезки, на которых они находятся.

Впрочем, и краткость программы, и простота алгоритма — вещи довольно обманчивые. И то и другое достигается благодаря использованию циклов. А так как тело цикла выполняется многократно, то ясно, что общее время работы программы может быть при ее краткости довольно большим. Во всяком случае, большим, нежели время счета по чисто ветвистой программе.

Понятно поэтому, что при программировании цикла нужно проявлять особую тщательность. Прежде всего, цикл надо *чистить*. Так называется вынесение за его пределы вычислений, не использующих величин, меняющихся в цикле, — у нас та-

ковыми являются x_n , x_k и x_0 . А в теле нашего цикла есть такая операция: сравнение знаков функций $f(x_0)$ и $f(a)$. Первая величина в цикле изменяется, а вот вторая остается постоянной на всем его протяжении. Поэтому целесообразно вычислить ее значение до начала цикла, запомнить в каком-либо адресуемом регистре, а в цикле это (уже вычисленное) значение при необходимости вызывать.

Кстати, обратите внимание, что цель, преследуемая чисткой цикла, — уменьшить время работы программы. Если при этом длина ее не изменится или даже сократится, то это очень хорошо, но если программа и увеличится на несколько команд, все равно стоит идти на это. Все-таки экономия времени превыше всего.

Но длина программы тоже играет немаловажную роль! Каждое вычисление значения функции может потребовать немало команд. Двукратная запись одинаковых последовательностей команд для вычисления $f(a)$ и $f(x_0)$ займет при этом изрядное место, и общая длина программы может превысить объем памяти микрокалькулятора. Как этого избежать?

Вот тут-то и настало время обстоятельно разобрать понятие, стоящее за третьим словом названия раздела: *подпрограммы*. Так называются части программы, обычно реализующие определенные, самостоятельные алгоритмы, и к которым можно обращаться из любого ее места. Именно таким образом целесообразно записать цепочку команд, по которым вычисляется функция $f(x)$, и обратиться к подобной подпрограмме в одном случае с аргументом a , а в другом — с аргументом x_0 . При таком подходе к делу не произойдет дублирования, которого мы опасались. И еще одна выгода: программу удобно будет модифицировать, переходя от поиска корня одной функции к поиску корня другой; всю ее переписывать не придется, надо будет лишь заменить подпрограмму вычисления функции.

На блок-схеме подпрограмма для наглядности выделена в блок «Ввод $f(x)$ ».

Подпрограммы широко используются для сокращения текстов программ в тех случаях, когда требуется провести вычисления по одному и тому же алгоритму в разных частях программы. Кроме того, существование подпрограмм позволяет собирать из них большие программы наподобие того, как строят дома из функциональных блоков.

В «калькуляторных» программах всякая подпрограмма представляет собой последовательность команд, обязательно заканчивающуюся командой В/О. Обращение к подпрограмме осуществляется командой ПП, после которой указывается адрес начала подпрограммы. Напомним, что отличие этой

команды от команды безусловного перехода БП в том, что как только встречается команда В/О, управление передается на адрес, следующий за командой обращения к подпрограмме. В некоторых языках программирования команда, подобная ПП, называется *Переход с возвратом*, что более точно отражает ее сущность.

Но продолжим анализ блок-схемы. В одном из блоков у нас записано сравнение знаков функций. В языке нашего микрокалькулятора «Электроника БЗ-34» нет специальной функции для выделения знака числа. Можно, конечно, сделать это с помощью нескольких команд, но лучше поступить проще, воспользовавшись известным свойством умножения: если сомножители имеют одинаковый знак, то произведение их положительно, в противном случае — отрицательно. Вычислите произведение, сравните его с нулем на «больше — меньше» и быстрее для машины (не надо делать лишних операций), и программа при этом получается короче.

Наконец, рассмотрим саму структуру цикла. Вычисления проводятся до тех пор, пока величина $x_k - x_n$ не станет меньше ε . На каждом шаге цикла требуется вычислить эту разность и сравнить ее с ε . А сколько раз будет прокручиваться цикл? Это в нашем случае легко подсчитать. Ведь если при каждом прохождении цикла отрезок уменьшается вдвое, то после n циклов его длина будет равна $(b - a)/2^n$. Решив неравенство $(b - a)/2^n < \varepsilon$, мы определим минимальное число повторений, необходимых для того, чтобы оно выполнялось:

$$n = \left[\frac{\ln(b - a)/\varepsilon}{\ln 2} \right] + 1.$$

Символ $[]$ используется для обозначения целой части, а единица добавлена, чтобы неравенство заведомо выполнялось.

Теперь, когда мы знаем, как определять число повторений цикла, у нас появилась возможность изменить в лучшую сторону его структуру. Цикл можно организовать так, чтобы он был повторен заданное число раз.

Есть в системе команд нашего микрокалькулятора такая команда, которая создана специально для этих целей. Называется она *командой организации цикла* FLN. Вводя ее в программу, вместо N следует подставить номер одного из регистров памяти (с нулевого по третий включительно), а в этот регистр занести перед началом работы цикла требуемое число его повторений. Эта команда пишется в конце цикла. Вслед за ней проставляется двузначное число mn — адрес начала цикла.

Напомним, как работает эта команда. Из содержимого регистра N после каждого обращения к команде, то есть

выполнения тела цикла, вычитается единица и, если результат не равен нулю, управление передается по адресу *mn*. В противном случае выполняется команда, следующая после пары FLN *mn*, то есть происходит выход из цикла.

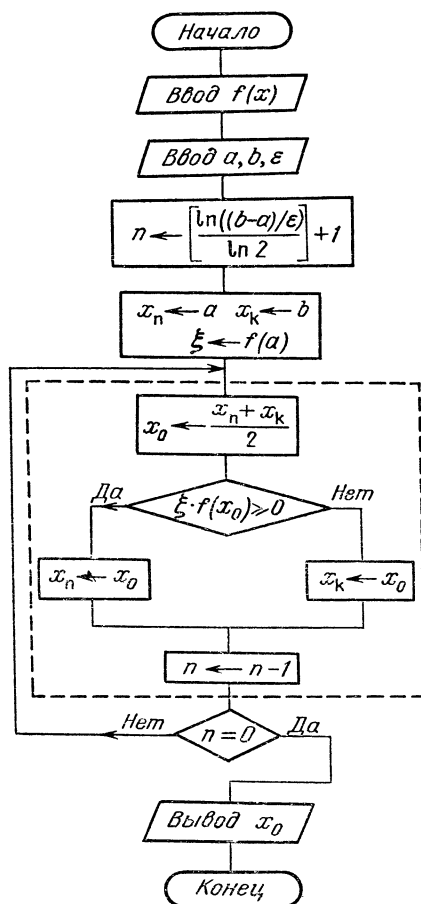


Рис. 12

С учетом высказанных соображений блок-схема алгоритма для определения корня методом половинного деления примет вид, показанный на рис. 12.

Сравните теперь рис. 11 и рис. 12. Обратите внимание: в первом случае блок сравнения расположен перед телом цикла, во втором — за ним. В теории программирования известны

термины: ЦИКЛ-ПОКА и ЦИКЛ-ДО. Их схематические структуры показаны на рис. 13. Первый выполняется, ПОКА истинно условие, записанное в ромбовидном блоке сравнения, то есть выход из цикла происходит по ветви «Нет». Второй, наоборот, работает ДО выполнения условия, указанного в ромбе, выход из него происходит по ветви «Да». В первой модификации тело цикла может не выполниться ни разу (там условие проверяется до начала его работы), во второй же

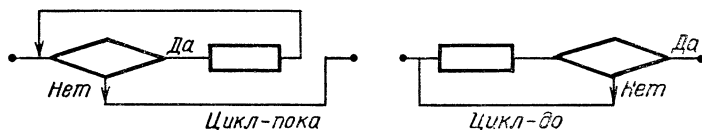


Рис. 13

записанные в теле цикла команды выполняются хотя бы один раз (проверка записана в конце цикла). Нетрудно сообразить, что на рис. 11 перед нами ЦИКЛ-ПОКА, на рис. 12 — ЦИКЛ-ДО.

Последняя схема, можно сказать, — идеальное поле деятельности для команды FLN: она одновременно производит вычитание единицы из счетчика цикла ($n \leftarrow n - 1$), сравнивает результат с нулем и осуществляет передачу управления по заданному адресу.

Теперь можно приступить к созданию программы. Как всегда, начнем с распределения адресуемых регистров:

$$\begin{aligned} a, x_n &\rightarrow RA & x_0 &\rightarrow RC \\ b, x_k &\rightarrow RB & n &\rightarrow R0 \text{ (счетчик повторений цикла)} \\ f(a) &\rightarrow R1 \end{aligned}$$

Программа показана на табл. 5. Ее запись сопровождается краткими примечаниями. Смысл их в описании действия некоторых команд и пояснении соответствия между программой и блок-схемой. Обозначения, используемые в комментариях, достаточно просты и наглядны. К примеру, $a \rightarrow RA$ означает, что величина, обозначенная на блок-схеме через a , заносится в регистр A. Так как использование стека в этой программе достаточно очевидно, то для сокращения места колонки с содержимым стековых регистров опущены.

Обратите внимание, что в программе отсутствует выделение целой части арифметического выражения при вычислении величины n . При этом использован очередной «секрет» микрокалькулятора. Дело в том, что если в регистре 0, 1, 2 или 3 записано не целое число, то при первом же обращении к

Таблица 5

Адрес	Команда	Код	Примечание
ВВОД a			
00	ПА	4—	} $a \rightarrow RA$ вычисление $f(a)$ $f(a) \rightarrow R1$
01	ПП	53	
02	39	39	
03	П1	41	
04	С/П	50	
ВВОД b			
05	ПВ	4L	$b \rightarrow RB$
06	ИПА	6—	$RX \leftarrow a$
07	—	11	$b - a$
08	С, П	50	
ВВОД ε			
09	\div	13	} $n = \frac{\ln(b-a)/\varepsilon}{\ln 2} + 1$ $n \rightarrow R0$ начало цикла
10	Fln	18	
11	2	02	
12	Fln	18	
13	\div	13	
14	1	01	
15	+	10	
16	П0	40	
17	ИПА	6—	
18	ИПВ	6L	
19	+	10	$x_n + x_k$
20	2	02	$x_0 = \frac{x_n + x_k}{2}$
21	\div	13	$x_0 \rightarrow RC$
22	ПС	4C	} вычисление $f(x_0)$
23	ПП	53	
24	39	39	
25	ИП1	61	
26	\times	12	
27	$Fx \geq 0$	59	$f(a) \cdot f(x_0)$
28	33	33	$x_n \leftarrow x_0$
29	ИПС	6C	
30	ПА	4—	
31	БП	51	
32	35	35	
33	ИПС	6L	} $x_k \leftarrow x_0$ конец цикла
34	ПВ	4L	
35	FL0	5Г	
36	17	17	
37	ИПС	6C	
38	С/П	50	

команде FLN, использующей этот регистр (FL0 в нашей программе), число это «усекается», то есть от него берется целая часть. Таким образом, отпадает необходимость делать это с помощью других команд. Запомните, однако, что свойство это довольно коварное. Если число, записанное в регистре, меньше единицы, то округления не происходит. Цикл с таким числом в регистре будет длиться «вечно» (пока не сломается машинка). К счастью, в нашей программе такого быть не может. Прибавляя единицу к величине $\ln((b-a)/\varepsilon)/\ln 2$, мы не только достигаем требуемой точности, но и формируем число, всегда большее единицы. Не будь этого, при $(b-a)/2 < \varepsilon$ наша программа неминуемо бы «зациклилась».

Как всегда, к программе должна быть приложена инструкция. Вот она.

1. Ввести программу.

2. Ввести подпрограмму вычисления функции, начиная с адреса 39. (Вот где отрабатывается блок нашей схемы «Ввод $f(x)$ ».) Аргумент функции брать из регистра X, результат помещать туда же. При описании подпрограммы не использовать регистры RA, RB, RC, R0, R1.

3. Перейти в режим вычислений.

4. Ввод: В/О a С/П b С/П ε С/П.

5. Вывод. После «останова» — результат на индикаторе.

6. Продолжение работы.

6.1. Для вычисления корней той же функции с другой точностью или на другом отрезке перейти к п. 4.

6.2. Для вычисления корня новой функции набрать на клавиатуре: БП 39, ФПРГ и перейти к п. 2.

Контрольный пример. $f(x) = x^2 - 3$ ($a = 1$, $b = 2$, $\varepsilon = 10^{-5}$). Подпрограмма:

39. Fx²; 40. 3; 41. — ; 42. В/О.

Результат: 1,7320478 (на самом деле первые 8 цифр корня в пределах точности ПМК: $\sqrt{3} = 1,7320508$).

Большинство пунктов инструкции аналогично рассмотренным в предыдущей главе. Здесь хотелось бы обратить внимание только на п. 2 (ввод подпрограммы). Повторим, что именно благодаря подпрограмме удастся достигнуть столь большого уровня универсальности, когда одна программа может решать большой класс различных задач (в нашем примере — практически любое уравнение, лишь бы оно удовлетворяло двум требованиям, приведенным в начале раздела, а функцию $f(x)$ можно было бы вычислить на микрокалькуляторе и подпрограмма для этого вменялась бы в его память).

В этом же пункте указано, где должен находиться аргумент функции и куда следует помещать результат. Это очень важная

информация. Ведь программа может быть устроена так, что входные данные для подпрограммы удобнее черпать не из регистра X, а из какого-нибудь другого регистра. То же относится и к результатам работы подпрограммы. Поэтому в инструкции обязательно нужно указывать источник входной информации и хранилище выходной информации.

Далее, в этом же пункте перечислены свободные регистры (перечисление сделано по методу исключения: все, кроме...). Так как анализируемая функция может быть довольно сложной, то при записи подпрограммы для вычисления ее значений могут понадобиться и адресуемые регистры. Ясно, что при этом нежелательно затрагивать регистры, где хранятся переменные основной программы — можно все испортить.

Обращение к подпрограмме из основной программы в нашем примере происходит в цикле. Поэтому к программированию ее необходимо подойти очень ответственно. Подпрограмма должна поставлять значения функции в основную программу с максимально возможной быстротой.

Приведенный пример — это, если можно так сказать, классическое использование подпрограммы. Вообще же область применения подобных структур значительно шире. В частности, для сокращения длины основной программы в подпрограммы можно выносить повторяющиеся в нескольких местах одинаковые последовательности команд. При этом необязательно, чтобы они представляли самостоятельный алгоритм. Более подробно вопросы оптимизации программ, в том числе и с использованием подпрограмм, будут рассмотрены в следующих разделах. Там же мы рассмотрим использование управляющих команд в режиме вычислений. Пока же отметим, что упоминаемая в п. 6.2 команда БП 39, набранная в этом режиме, никак не отражается на состоянии индикатора. Однако когда после ее ввода ПМК переводится в режим программирования, в качестве адреса вводимой команды на индикаторе будет стоять именно 39.

Теперь попытаемся сформулировать основные рекомендации по использованию ветвей, циклов и подпрограмм.

1. Использование ветвей позволяет реализовать альтернативные алгоритмы, то есть такие, где путь решения зависит от сочетания определенных параметров. Начальной точкой ветви является команда условного перехода. При необходимости обхода той или иной ветви используются команды безусловного перехода.

2. Для сокращения длины ветвистых программ последовательности команд, одинаковые для нескольких ветвей, лучше размещать до точки ветвления или после точки смыкания.

3. Циклы используются для неоднократного повторения команд одного и того же участка программы. При программировании на микрокалькуляторах типа «Электроника БЗ-34» желательно все циклы по возможности сводить к структурам типа ЦИКЛ-ДО. Эти структуры реализуются более эффективно благодаря использованию специальной команды организации цикла.

4. Циклические программы работают, как правило, довольно долго. Поэтому чистке цикла нужно уделять особое внимание. Помните, что при этом ценой небольшого удлинения программы можно значительно сократить время ее выполнения.

5. Подпрограммы используются в тех случаях, когда в разных местах программы нужно выполнить одну и ту же последовательность команд. В программе может быть использовано любое число подпрограмм. Размещаются они обычно после основной программы. Заканчивается подпрограмма всегда командой В/О.

6. С помощью подпрограмм удается решать широкий класс задач. Программа при этом состоит из двух частей: неизменяемой, основной программы и изменяемой — подпрограммы. В этих случаях использование подпрограмм оправдывается даже при однократных обращениях к ним.

7. ПО КОСВЕННЫМ АДРЕСАМ

Вряд ли найдется человек, который никогда не писал или не получал писем. На конверте письма всегда написан адрес, скажем, такой: Сочи, ул. Зеленая, 16. Иванову К. В. Адрес этот однозначно определяет, куда и кому должно быть доставлено письмо.

Но встречаются и другие способы адресации. Например, Ялта, ул. Солнечная, 25. Сидорову А. В. (Для Петренко Г. С.)

Видите отличие? Автор письма указывает не непосредственно адрес Петренко, а адрес Сидорова, который знает адрес Петренко.

Программист назвал бы адрес, указанный на первом конверте, прямым, а на втором — косвенным. Для программиста такие термины привычны. Действительно, в программировании на языках типа автокодов *косвенная адресация* используется часто. В отличие от *прямой адресации*, где указывается адрес команды или числа, при косвенной адресации указывается «адрес адреса». Такой прием значительно расширяет возможности программирования.

Набор команд, с которым мы познакомились в разделе 4, позволяет составлять сравнительно широкий спектр программ,

но оказывается бессильным, когда по сути алгоритмов приходится перебирать информацию, лежащую в разных адресуемых регистрах, варьировать адреса переходов и т. д.

Для этих целей, впрочем, как и для многих других, оказываются незаменимыми команды косвенной адресации. Описывая команды микрокалькулятора, мы уже упоминали о них. Напомним, что служат они для обмена информацией между операционным регистром X и регистрами данных, а также для передачи управления в программах. Для набора их используются те же клавиши, что и для обычных команд, но начинается набор каждый раз с клавиши «К».

Команды косвенной записи в регистры:

КП0 (код L0), КП1 (L1), ..., КП9 (L9),
КПА (L-), ..., КПД (LG);

они записывают содержимое регистра X не непосредственно в регистр, указанный в команде, а в другой, в тот, номер которого записан в указанном регистре. Аналогично действуют и команды косвенного считывания:

КИП0 (Г0), КИП1 (Г1), ..., КИП9 (Г9),
КИПА (Г-), ..., КИПД (ГГ).

Например, если в R7 записано число 5, а в R5 — число 10, то команда КИП7 вызовет в регистр X число 10, то есть содержимое регистра, номер которого записан в R7, — пятого. Иначе говоря, при косвенной адресации регистры, упоминаемые в командах, используются не как хранилища данных, а как хранилища адресов данных. Естественно, возникает вопрос: а как обратиться к регистрам с буквенными «номераами»? Не можем же мы заслать в регистр число «А» или «В».

Разработчики ПМК такую ситуацию предусмотрели. Все адресуемые регистры микрокалькулятора имеют сквозную нумерацию. Регистр А числится под номером 10, регистр В — 11, RС — 12 и RД — 13. Поэтому использование команды КП9 при содержимом регистра 9, равном 11, передает число из RХ в регистр В.

Однако все выше описанное относится лишь к косвенным обращениям к регистрам с именами 7, 8, 9, А, ..., Д. При использовании других регистров происходят вещи еще более интересные. Числа, хранящиеся в них, модифицируются, то есть изменяются во время косвенной команды, и обмен информацией происходит по этому модифицированному адресу. Модификация состоит в том, что содержимое регистров 0, 1, 2 и 3 уменьшается на единицу, а содержимое регистров 4, 5 и 6 на единицу же увеличивается.

Проследим этот процесс подробнее. Если в регистр 1 записать число 8, то по команде КИП1 будет сделано следующее. Во-первых, содержимое R1 будет уменьшено на единицу и станет равным 7 и, во-вторых, в регистр X будет передано число, хранящееся в 7-м регистре. Еще пример. Если мы запишем в R4 число 11, то команда КП4 передаст число из регистра X в 12-й регистр, то есть в регистр C.

Представляем себе недоумение человека, с программированием не знакомого. Зачем эти сложности? По имени одного регистра искать имя другого, да и учитывать еще, что имя это может быть не тем, что записано в первом... Все равно, что

ехать из Москвы в Ленинград через Владивосток и Игарку. Но раз так делают, «значит, это кому-нибудь нужно»?

Допустим, что в результате решения некоторой задачи в адресуемых регистрах накоплено несколько чисел и нужно найти среди них наибольшее.

Задача несложная. Алгоритм поиска максимума давно и хорошо отработан. Нужно взять любое число из набора и предположить, что оно максимальное. Затем перебирать все остальные числа подряд и сравнивать их с «условным» максимумом. Если какое-либо из этих чисел окажется больше его, то положить «исполняющим обязанности» максимума это число. Так, перебрав все числа, мы на месте «условного» максимума будем иметь макси-

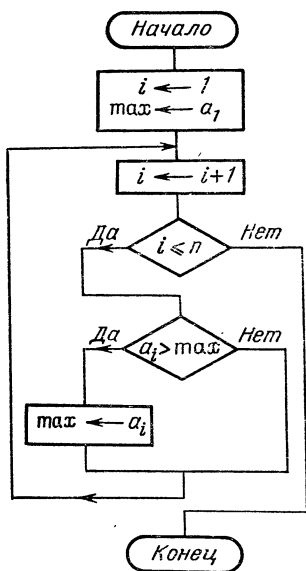


Рис. 14

мум абсолютный. Блок-схема алгоритма для поиска максимума среди n чисел a_1, a_2, \dots, a_n приведена на рис. 14.

В этой блок-схеме отсутствуют блоки ввода и вывода. Ведь мы предполагаем, что этот алгоритм будет включен в какую-нибудь другую программу как составная часть и эта большая программа будет готовить входную информацию и использовать выходную.

Блок-схема представляет собой цикл. Подобные структуры подробно рассмотрены в предыдущем разделе.

Так вот, без использования команд косвенной адресации составить программу решения поставленной задачи для ПМК

просто невозможно. Никаким иным способом обращаться к разным регистрам с помощью одной и той же команды нельзя.

Чтобы более полно использовать возможности косвенной адресации при записи программы для поиска максимума, блок-схему алгоритма целесообразно несколько изменить. Так как данные удобно записывать в адресуемые регистры последовательно, то использовать свойство модификации адреса в регистрах 4, 5 или 6 мы не сможем — ведь сами эти регистры могут оказаться в центре событий, в них тоже могут попасть анализируемые числа. Лучше в качестве счетчика-указателя регистров использовать какой-либо регистр, лежащий с «краю», то есть 0 или Д. При обращении к РД содержимое его не модифицируется, остается использовать R0. Но в нем при каждом выполнении косвенной команды содержимое не увеличивается на единицу, как записано в нашей блок-схеме, а наоборот, уменьшается. Потому-то блок-схему и стоит переделать, заменив последовательность опроса регистров: не с 1 до n , а с n до 1.

Так как эти изменения незначительны, то новую блок-схему мы приводить не будем. Заметим только, что перемены коснутся трех верхних блоков: в первом вместо 1 нужно записать n , во втором — поменять знак «+» на «-», и в третьем сравнивать текущее значение i не с n , а с единицей. Выбрав в качестве счетчика регистр 0, для хранения результата выделим регистр Д. Он тоже «крайний». Тогда максимальное количество чисел, которое сможет «переваривать» наша программа, будет равно 12 (все регистры от R1 до RC). Вот один из вариантов такой программы:

00	КИПО	Г0	07	ИПД	6Г	14	БП	51
01	ПД	4Г	08	КИПО	Г0	15	02	02
02	ИПО	60	09	—	11	16	С/П	50
03	1	01	10	$Fx < 0$	5Г			
04	—	11	11	14	14			
05	$Fx \neq 0$	57	12	FBx	0			
06	16	16	13	ПД	4Г			

Перед началом работы программы в нулевой регистр должна быть занесена величина, на единицу большая количества анализируемых чисел. Сами числа должны располагаться последовательно в адресуемых регистрах, начиная с первого. После окончания работы программы максимальное число из набора будет в регистре Д. Надеемся, что читатель самостоятельно разберет приведенную программу.

В этой программе свойства модификации адреса используются, так сказать, по «прямому назначению», то есть с

помощью команды косвенного считывания КИПО мы перебираем содержимое других регистров памяти.

Однако иногда полезно использовать эти свойства и в других целях. Например, если мы хотим просто увеличить на единицу содержимое регистра 4, 5 или 6, то вместо команд, скажем, ИП4 1 + П4, достаточно записать КИП4. При этом нас может не интересовать, что будет вызываться в операционный регистр X, — нам вполне достаточно побочного эффекта от этого процесса, то есть увеличения содержимого самого регистра 4 на единицу. Таким же образом при необходимости можно уменьшать содержимое регистров 0, 1, 2 или 3.

Теперь о *косвенно-адресных командах передачи управления*. От обычных, ранее рассмотренных команд, выполняющих те же функции, эти команды отличаются тем, что адрес, на который нужно передавать управление, должен содержаться в адресуемом регистре, фигурирующем в команде. Например, команда КБПА, если в регистре А записано число 20, эквивалентна команде БП 20. Занимают косвенно-адресные команды такого типа одну ячейку в памяти вместо двух, положенных для хранения их прямых аналогов.

К командам этой группы относятся команды безусловного перехода:

КБПО (код 80), КБП1 (81), ..., КБП9 (89),

КБПА (8—), ..., КБПД (8Г);

условного перехода:

К $x < 0$ 0 (Г0), ..., К $x < 0$ Д (ГГ),

К $x = 0$ 0 (Е0), ..., К $x = 0$ Д (ЕГ),

К $x \neq 0$ 0 (70), ..., К $x \neq 0$ Д (7Г),

К $x \geq 0$ 0 (90), ..., К $x \geq 0$ Д (9Г);

перехода на подпрограмму:

КППО (—0), ..., КППД (—Г).

Все, что было сказано о модификации адреса, относится и к этим командам. Используются косвенно-адресные команды передачи управления сравнительно редко, во всяком случае, значительно реже команд косвенно-адресной записи и считывания. Основное поле их действия — некоторое улучшение сервиса и экономия ячеек программной памяти, правда, ценой использования регистров данных.

Вы уже, наверное, обратили внимание, что все команды косвенной адресации используют целые числа. А что будет,

если в регистре, с которым оперирует косвенная команда, записать число дробное? В этом случае наш ПМК поступает самым естественным образом. Он просто игнорирует дробную часть числа, отбрасывая ее при выполнении команды. Например, если в регистре 8 было записано число 3,14, то после выполнения команды КИП8 в операционный регистр будет вызвано содержимое 3-го регистра, а в R8 окажется число 3. Это дает, кстати, возможность довольно простым способом выделять целую часть числа, что очень полезно, так как специальной функции для этих целей у ПМК нет. Правда, надо отметить, что для чисел, меньших единиц и, тем более, для отрицательных этот способ не пригоден. При использовании регистров от нулевого до шестого дробная часть числа также отбрасывается, а остаток модифицируется, как обычно.

Примеры использования косвенно-адресных команд в микрокалькуляторных программах будут даны в следующих разделах, а пока подведем итоги.

1. Все косвенно-адресные команды начинаются с набора клавиши «К» и занимают одну ячейку памяти.

2. Команды считывания и записи, использующие косвенную адресацию, осуществляют обмен информацией между операционным регистром X и тем регистром, номер которого записан в регистре, указанном в команде.

3. Команды передачи управления, использующие косвенную адресацию, передают управление на адрес, который записан в соответствующем регистре.

4. При использовании в этих командах регистров 0, 1, 2 или 3 содержимое регистра сначала уменьшается на единицу, а уже потом происходит обмен информацией или передача управления.

5. При использовании регистров 4, 5 или 6 перед выполнением команды содержимое регистра, указанного в команде, увеличивается на единицу.

6. Если в регистре, используемом в команде косвенной адресации, записано нецелое число, то при выполнении команды дробная часть его отбрасывается.

8. КАК ОТЛАЖИВАТЬ И УЛУЧШАТЬ ПРОГРАММЫ

Мы уже говорили о том, что программа — это изделие. Как и всякое техническое изделие, она проходит стадии проектирования, изготовления рабочих чертежей, макетирования и наладки. Только называются эти стадии у программистов несколько иначе. Проектирование программы — это

постановка задачи, выбор метода и алгоритма. Рабочим чертежом здесь служит блок-схема. Макетирование программы — это запись ее первой версии. А последняя стадия производства, наладка, в применении к программе называется *отладкой*.

Пока длится разработка изделия, на разных стадиях этого процесса участие в нем принимают разные специалисты: проектировщики, чертежники, сборщики, наладчики. При создании программы для микрокалькулятора все эти функции объединяются в одном лице — владельце ПМК.

Чтобы наладить изделие, сделать его работоспособным, нужны инструменты и знания приемов наладки. При отладке программ инструментами служат специальные команды, управляющие работой калькулятора. Часть из них используется в режиме программирования, остальные — в режиме вычислений.

...Первая версия программы введена в программную память микрокалькулятора. Нелишне проверить, верно ли она там записана, и если нужно, исправить ошибки ввода. Но как прочесть текст или хотя бы его фрагмент, скрытый в памяти миниатюрной вычислительной машины?

Для этого нужно нажать клавиши «F», «ABT», «B/O», «F», «ПРГ». В правом углу индикатора загорится 00. А далее нужно раз за разом нажимать клавишу «ШГ». При каждом ее нажатии программный указатель увеличивается на единицу, так что на месте цифр 00 станут последовательно появляться цифры 01, 02, 03 и так далее. А в левом углу индикатора появятся коды операций: адрес самого левого кода на единицу меньше стоящего справа, за ним стоят коды двух предыдущих операций. Скажем, вся строчка на индикаторе выглядит так:

12 6L 11 17

Это означает, что по четырнадцатому адресу находится команда с кодом 11 (вычитание), по пятнадцатому — с кодом 6L (вызов в регистр X содержимого регистра B), по шестнадцатому — с кодом 12 (умножение).

Если вы хотите просмотреть программу не с самого начала, а только ее фрагмент, начиная с адреса *mn*, то в таком случае просмотр надо предварить командами

FABT, БП *mn*, FПРГ.

И далее станем опять нажимать клавишу «ШГ»: она поведет вас по программе в порядке нарастания адресов. Но можно пойти и вспять, по убывающим адресам. Для этого следует нажимать клавишу «ШГ». Каждое выполнение этой команды уменьшает на единицу программный указатель.

Используя эти две клавиши, нетрудно «подогнать» к индикатору, этому своеобразному «окну» в программу, любой адрес. И когда код команды загорится в левом углу индикатора, вы сможете исправить ее, если она неверна.

Заметим сразу: сделать это так, как исправляют неверно введенные числа, то есть с помощью клавиши «Сх», нельзя. С ее помощью нельзя даже стереть только что введенную команду, если вы сразу убедились, что она неправильна. В режиме программирования команда Сх (Стереть х) — это обычная команда, которая будет просто записана в память. Стереть неправильную команду можно только одним способом — записать на ее место новую. Удалить же лишнюю команду можно, записав на ее место пустую команду КНОП (Нет Операции).

Допустим, в только что приводившемся фрагменте код операции, стоящий с левого края и записанный в программе по адресу 16, ошибочен: по этому адресу должно быть выполнено сложение. Нажимаем на клавишу «ШГ», то есть делаем шаг назад по тексту программы. В правом углу появляется цифра 16. Это значит, что именно в шестнадцатую ячейку будет занесен код операции, клавишу которой вы сейчас нажмете, а находившийся там прежде код будет при этом стерт. Нажимаем клавишу «+», и ошибка исправлена. Таким способом можно изменить содержимое любой ячейки памяти, за исключением тех, где записаны адреса переходов. Причина исключения в том, что адрес перехода состоит из двух цифр и воспринимается как один код только тогда, когда перед ним набрана команда перехода (БП, ПП, FL0, Fx = 0). В противном случае каждая цифра записывается в отдельную ячейку. Поэтому для изменения адреса перехода нужно вернуться на предыдущую команду, то есть команду перехода, записать ее вновь и уже затем вписать новый адрес. Например, чтобы изменить во фрагменте

17. БП 18. 60

число 60 на 59, недостаточно установить указатель адреса на 18 и вводить 59. В этом случае цифры 5 и 9 будут записаны в две смежные ячейки. При этом адрес перехода станет равен 05, да и команда по адресу 19 окажется испорченной. Если же установить указатель адреса на 17 и повторить ввод всей команды перехода БП 59, то все будет правильно.

С помощью команд ШГ и ШГ можно добраться до любой ячейки памяти. Заметим, что сами эти команды в память не записываются. Они лишь меняют показание счетчика команд. Также не фиксируется в памяти команда FCF. Она используется для отказа от «полунабранной» команды. Речь идет о командах,

требующих нажатия нескольких клавиш, то есть начинающихся с нажатия клавиш «F», «K», «P», «IP». Случайно нажатая клавиша (из перечисленных) еще не отображается на индикаторе — калькулятор ждет, что будет нажато затем, чтобы правильно интерпретировать всю последовательность использованных клавиш, например Flп или КИП5. Так вот, чтобы отменить этот режим ожидания, и служит команда FCF. После нее интерпретация нажимаемых клавиш начнется «с нуля».

Используя описанные приемы отладки, можно добиться, чтобы текст программы, введенный в память калькулятора, полностью совпал с тем, что написан вами на бумаге.

Но можно ли после этого считать, что программа создана и работать над ней больше уже не нужно? Можно ли быть уверенным, что наше изделие заработает при его включении, то есть при запуске программы?

Любой программист знает, что между первым вариантом программы и ее работающей версией — «дистанция огромного размера». Работа по преодолению этой дистанции и есть отладка программы. Процесс этот трудоемкий и достаточно длительный. Порой отладка сложной программы отнимает в несколько раз больше времени, чем ее написание. Ведь вероятность появления ошибок существует на любом этапе проектирования — и при написании текста программы, и при создании блок-схемы и алгоритма, и, наконец, даже при постановке задачи. Последнее «сито», просеивающее ошибки, — это отладка.

Как ее проводить?

После того, как программа введена в память и все нужные для ее работы числа занесены в адресуемые регистры, можно, конечно, запустить программу на счет. Напомним еще раз, как это делается. Калькулятор переводится в режим вычислений (FABT), программный указатель устанавливается на нулевой адрес (B/O) и программа запускается (C/P).

Но не торопитесь нажимать пусковую клавишу. Для первого раза рекомендуем прогнать программу в специальном режиме с остановом после выполнения каждой команды. Для этой цели существует специальная отладочная команда ПП. Мы уже писали, что в режиме программирования ПП — это Переход на Подпрограмму. В режиме вычислений эта команда интерпретируется как Потактовый Проход. Каждое нажатие клавиши «ПП» вызывает выполнение очередной команды и останов. При этом на индикаторе видно содержимое регистра X, то есть результат выполнения команды.

Рассмотрим более подробно процесс отладки программы на решении конкретной задачи из области электротехники.

Имеется электрическая цепь, составленная из нескольких сопротивлений (рис. 15). Нужно определить полное сопротивление цепи между узлами *A* и *B*.

Когда эта задача была предложена учащимся техникума, то один студент предложил следующее решение.

С использованием известных соотношений электротехники формулы для программирования были записаны им так:

$$R_{23} = \frac{R_2 R_3}{R_2 + R_3};$$

$$R_{123} = R_1 + R_{23};$$

$$R_{AB} = \frac{R_4 R_{123}}{R_4 + R_{123}}.$$

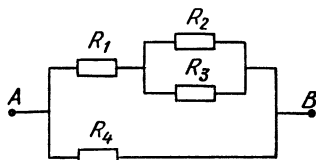


Рис. 15

Исходные данные было предложено вводить в одноименные регистры, а результат читать на индикаторе после останова программы.

Сама программа, реализующая вычисления по этим формулам, была написана в виде, изображенном на табл. 6. Не

Таблица 6

Адрес	Команда	Код	Адрес	Команда	Код
00	ИП2	62	09	ПВ	4L
01	ИП3	63	10	ИП4	64
02	+	10	11	×	12
03	ИП2	62	12	ИП8	68
04	ИП3	63	13	ИП4	64
05	×	12	14	+	10
06	÷	13	15	÷	13
07	ИП1	61	16	С/П	50
08	+	10			

сомневаемся, что те из вас, у кого уже накоплен опыт работы с программируемым микрокалькулятором, даже не запуская программу на счет, быстро найдут в ней ошибки. Однако в более сложных программах это удастся далеко не всегда. Поэтому на нашем простом примере покажем, как это делается.

Введем текст программы в память микрокалькулятора, контролируя правильность ввода по таблице кодов (она помещена на третьей странице обложки).

После ввода программы переводим калькулятор в режим вычислений и записываем в адресуемые регистры значения сопротивлений. Для начала выберем такие значения, при которых результат легко получить даже устно. Например, $R_1 = 0,5$, $R_2 = R_3 = R_4 = 1,0$. Результат должен быть равен 0,5. Запускаем программу на счет, и через несколько секунд

читаем на индикаторе ответ: 2,5. Это неверно. Придется искать ошибку. Лучше всего это сделать, как мы уже говорили, при потактовой прогонке. Очищаем счетчик команд: «В/О». Исходные значения сопротивлений уже занесены в адресуемые регистры. Можно начать прогон. Нажимаем последовательно на клавишу «ПП» и проверяем содержимое регистра X.

Итак, начали. На индикаторе последовательно появляются числа 1, 1, 2, 1, 1, 1. Пока все вроде бы правильно: первыми тремя командами подсчитан знаменатель дроби $R_2 R_3 / (R_2 + R_3)$, следующими тремя — ее числитель. Теперь поглядим, как они будут поделены друг на друга — нажмем на клавишу «ПП». На индикаторе: 2. Это ошибка! Значение дроби, как нетрудно подсчитать в уме, должно быть равно 0,5. Причина ошибки выясняется быстро: студент неправильно приготовил стек для деления, числитель и знаменатель перепутаны местами. Значит, нужно ввести дополнительную команду: либо $F1/x$ после деления, либо XY до него. Второе явно предпочтительнее. Ведь $F1/x$ — команда вычислительная, а каждое вычисление хоть чуть-чуть, но ухудшает точность результата. Однако и тот и другой путь означают ввод дополнительной команды. Так как «раздвинуть» программу на микрокалькуляторе нельзя, то приходится при вводе дополнительной команды перебирать все остальные, записанные после нее. В нашем случае можно, к счастью, избежать этого. Кто мешает сначала вычислить числитель первой формулы, а потом ее знаменатель? Стек при этом будет заполнен правильно, а количество команд не изменится. Для этого нужно заменить команды по адресу 02 («x» вместо «+») и по адресу 05 («+» вместо «x»). Набираем команду БП 02 и переводим ПМК в режим программирования. Ячейка 02 подготовлена для приема команды. Вводим ее, нажимая «x», затем дважды нажимаем «ШГ» и заменяем команду по адресу 05: «+».

Можно начать прогонку снова: «F», «ABT», «В/О» и далее раз за разом «ПП». Следим за экраном: 1, 1, 1 (правильно!), 1, 1, 2, $5 \cdot 10^{-1}$ (прошли ошибочный участок), $5 \cdot 10^{-1}$, 1 (вычисление R_{123}), 1, 1, 1 (получено произведение $R_4 R_{123}$), 0. А это еще откуда? Командой 09 мы записали промежуточный результат (R_{123}) в регистр В. А теперь считываем содержимое регистра 8. Ясно, что это ошибка. Возможно, студент просто перепутал похожие по написанию букву В и цифру 8.

Вообще, путаница с символами для микрокалькулятора типа «Электроника БЗ-34» — явление довольно редкое. А вот на ЭВМ, где используются и буквенные, и цифровые символы для записи программ, — увы, заурядное. Особенно часто путают цифру «0» и букву «О», цифру «1» и латинскую букву «I».

Но вернемся к микрокалькулятору. Надо исправить ошибку по адресу 12. Так как только что была выполнена именно эта команда, на счетчике команд должно быть 13. Переводим ПМК в режим программирования (ФПРГ), нажимаем клавишу «ШГ». На индикаторе — нужный адрес 12. Вводим команду ИПВ и вновь прогоняем программу. Не будем утомлять читателя еще одним подробным описанием этого процесса — на этот раз все пройдет правильно.

После первой прогонки желательно пропустить еще один тест посложнее, убедиться, что результат совпадает с полученным на бумаге, и можно приступать к расчетам по программе.

Конечно, на практике процесс отладки намного более трудоемок (его трудоемкость определяется сложностью программы). Ведь выловить менее очевидные ошибки, например в самом алгоритме, намного сложнее. Тем не менее с помощью описанных способов всегда можно выявить, в каком месте программы или алгоритма происходит сбой. А дальше все зависит от опыта и искусства программиста.

Наконец, программа отлажена и работает. С одной стороны, цель достигнута — можно получать результаты решения поставленной задачи. Но, с другой стороны, остается сомнение: хрсошая ли получилась программа? Можно ли ее улучшить? И нужно ли ее улучшать?

Здесь мы подходим к очень важному вопросу, ответ на который вовсе не так очевиден, как кажется с первого взгляда. Речь идет о проблеме оптимизации программ.

Сделаем небольшой экскурс в историю. Появились вычислительные машины не так уж давно — несколько десятилетий тому назад. Первые представители этого семейства были громоздкими, обладали сравнительно невысоким быстродействием, малой емкостью памяти. К тому же они отличались неимоверно высокими ценами. Причем дорого стоила не только сама машина, но и время работы на ней. Это рождало специфический подход к программированию. К выходу на ЭВМ программист готовился, как к ответственному свиданию, заранее продумывал, как использовать каждую минуту драгоценного времени. Естественно, что и программы стремились делать так, чтобы они экономно расходовали машинную память и время ее работы. Время работы самого программиста стоило настолько дешевле машинного, что оно просто не принималось в расчет. Писались программы на языке, единственно понятном ЭВМ первого поколения, — языке машинных команд. Время создания программ исчислялось месяцами и даже годами.

По мере развития вычислительной техники возможности ЭВМ увеличивались, появились удобные алгоритмические языки. Программирование из элитарной профессии стало превращаться в средство для массовой автоматизации вычислительных работ. При этом и цена машинного времени стала падать. Постепенно выяснилось, что самым трудоемким процессом стало не само создание машины, а «начинка» ее *математическим обеспечением* — совокупностью программных средств, с помощью которых ЭВМ может решать конкретные задачи. Оказалось, что цена времени, затрачиваемого на создание программ, стала сопоставима и иногда сильно превышала цену машинного времени. Поэтому и отношение к программам изменилось. На первый план стало выступать время их создания, четкость структур программ, возможность их модернизации и совместной работы с другими программами.

Но машин все же оставалось намного меньше, чем пользователей. Сейчас и эта ситуация начинает меняться. Причиной тому, в частности, появление программируемых микрокалькуляторов — вычислительных машин индивидуального пользования. Для владельца микрокалькулятора проблемы доступности ЭВМ не существует — она всегда под рукой. Но микроскопичность ее, проявляющаяся не только в размерах, но и в памяти, быстродействии, языковом обеспечении, причудливым образом роднит ПМК с самыми старыми машинами.

Думается, это накладывает определенный отпечаток на психологию пользователей ПМК и объясняет стремление некоторых из них к безудержной «переоптимизации» программ.

Откройте почти любое пособие по программированию на ПМК, просмотрите рубрику «Человек с микрокалькулятором» в журнале «Наука и жизнь» и вы везде прочтете, что программу всегда нужно оптимизировать.

И тем не менее, рискуя вызвать яростную критику со стороны программистского племени, на вопрос: «Всегда ли нужно оптимизировать программы?» — ответим: «Нет, не всегда».

Ведь как ни упоителен сам по себе процесс составления программ, не будем забывать, что программа — это не самоцель, а средство для решения задачи. Поэтому на первый план стоит поставить не оптимизацию программы, а оптимизацию всего процесса решения задачи, то есть получение результата за возможно более короткое время и с наименьшими затратами сил.

Очень часто в повседневной практике встречаются задачи, так сказать, разового использования. Например, нужно построить график зависимости одной величины от другой, а

задана эта зависимость набором довольно сложных формул. Допустим, что по этим формулам нужно посчитать сотню выходных значений. Решать задачу поэтому целесообразно не в режиме вычислений, а составив программу. Допустим далее, что, потратив пару часов, вы составили программу, считающую каждую точку графика за две минуты. Время счета показалось вам слишком большим, и проработав еще часа три, вы написали новую программу. Она считает каждую точку вдвое быстрее. Оправдан ли был ваш труд? Конечно, вторая программа лучше, но давайте проведем небольшой расчет, для которого микрокалькулятор нам даже не понадобится. Время получения результата в первом случае равно: $2 \text{ ч} + 100 \times 2 \text{ мин} = 5 \text{ ч } 20 \text{ мин}$, а во втором: $2 \text{ ч} + 3 \text{ ч} + 100 \times 1 \text{ мин} = 6 \text{ ч } 40 \text{ мин}$. Сократив вычисления по программе более чем на полтора часа, мы почти на столько же проиграли в общем времени получения результата. Так кому нужна такая оптимизация? Это все равно, что потратить полчаса на выбор оптимального сочетания средств транспорта, чтобы попасть с одной улицы на другую, когда пешком этот путь можно проделать за двадцать минут!

Иное дело, если задача, которую вы решаете, — «массовая», то есть используется часто либо вами, либо другими. Здесь уж время, затраченное на оптимизацию, окупится сторицей. И уж, конечно, необходимо оптимизировать программы, претендующие на роль стандартных, то есть рекомендуемые широкому кругу пользователей. Кстати, прекрасные образцы оптимизации демонстрирует сборник «Прикладные программы для микроЭВМ» А. Н. Цветкова и В. А. Епанечникова. Программы в сборнике настолько «вычищены», что если вам удастся сократить любую из них хотя бы на пару команд или уменьшить время счета на несколько процентов (естественно, без ущерба для удобства и точности), считайте, что вы — программист высшей квалификации.

Последней фразой мы вплотную подошли к вопросу: по какому критерию оптимизировать программы (если в том есть необходимость)? У нас фигурировали четыре параметра: длина программы, ее быстродействие, удобство использования и точность выходных результатов. Этот перечень, пожалуй, определяет полный набор свойств, характеризующих программу для микрокалькулятора. Ясно, что из двух программ, решающих одну и ту же задачу, лучше та, которая короче, быстрее, точнее и удобнее. Как было бы хорошо, если бы все это достигалось одновременно! Увы, нет в мире совершенства. Поэтому приходится выбирать, что важнее, иначе говоря, не вообще улучшать, а именно оптимизировать.

В математике процесс оптимизации состоит в поиске минимума (или максимума) некоторой функции и значений переменных, обеспечивающих его. При этом обычно на некоторые переменные накладываются определенные ограничения. Например, при составлении оптимального заводского плана по выпуску продукции, обеспечивающего наибольшую прибыль, ограничениями являются имеющееся количество сырья, технические ресурсы и т. д.

Попробуем выявить ограничения для процесса оптимизации программы.

В первую очередь, очевидно, нужно сказать о точности результатов. Она, как правило, задается практическими соображениями, и посему попытка укоротить программу или уменьшить время ее счета за счет снижения точности вообще не может рассматриваться. Далее — длина программы. Она ограничена возможностями ЭВМ: 98 ячеек программной памяти и 14 адресуемых регистров. Наконец, время и удобства. Тут явных ограничений нет. Но понятно, что программа, получающая результат, скажем, после недели непрерывной работы, или программа, дающая результат, в котором нельзя разобраться, никому не нужны.

Мы уже упоминали, что удовлетворить всем требованиям в отношении качества программы по «максимуму» невозможно. Поэтому стоит некоторые требования выделить и считать их заданными заранее. Отнесем сюда точность (она закладывается уже на этапе выбора алгоритма) и, пожалуй, удобство программы (или, как говорят программисты, ее сервис). Подробнее о нем мы будем говорить в одном из следующих разделов. Пока же ограничимся замечанием, что при работе с программой желательно свести к минимуму «ручные» операции и сделать вывод результатов наглядным и легко интерпретируемым. Будем считать, что сервис программы тоже задан заранее в «технических условиях» для ее составления.

Остается длина программы и ее быстродействие. Эти параметры альтернативны, как нетрудно показать на простом примере. Допустим, что в программе есть цикл и в его теле используется величина $\sin a$. Параметр a в цикле не меняется и значение его лежит в регистре А. Вычисления можно записать просто: ИПА F \sin — всего две команды. Но известно, что вычисление синуса — операция довольно длительная, поэтому, чтобы увеличить быстродействие, лучше данный фрагмент преобразовать. Вне цикла запишем: ИПА F \sin ПВ, а в цикле — ИПВ. Команд стало на две больше, да и лишний адресуемый регистр задействован, но зато время работы будет меньше. Поэтому, чтобы уменьшить время счета, пред-

почтительнее второй вариант, для сокращения же длины программы — первый. А что важнее?

Давайте взглянем на программу глазами пользователя. Если она поместилась в память микрокалькулятора, то какая разница, длиннее она на пару команд или короче? А вот время ее работы — это время получения результата, и уменьшение его дает ощутимый эффект. Поэтому, по нашему мнению, быстроедействие программы — это самый важный ее параметр, так что оптимизацию проводить надо в первую очередь ради того, чтобы уменьшить время работы программы.

Существует довольно много методов, позволяющих увеличивать быстроедействие программ. Овладение большинством из них приходит вместе с опытом. Мы же опишем некоторые часто применяемые (стандартные) приемы, убыстряющие работу программ.

Прежде всего, следует отдавать себе отчет, что большое время счета характерно для циклических программ. Поэтому уже на этапе выбора алгоритма стоит отдавать предпочтение *аналитическим методам* — так в математике называются методы решения задач, когда конечный результат выражается набором формул (нециклических!). Хотя иной раз программы, написанные для реализации этих методов, значительно длиннее, но время счета по ним намного меньше. Например, корни квадратного уравнения можно вычислить по специальной, но длинной программе (она приведена в разделе 5) и по универсальной, намного более короткой программе дихотомии (см. раздел 6). Так вот, время счета по первой программе намного меньше.

Что касается записи самих программ, то в первую очередь, конечно, нужно «чистить» циклы, удаляя оттуда операции, не зависящие от параметров цикла. Желательно также заменять «медленные» операции более «быстрыми». Если нужно, например, число из регистра А возвести в куб, то можно это сделать так:

3 ИПА Fx³,

а можно и так:

ИПА $\uparrow Fx^2 \times$.

Хотя второй способ на одну команду длиннее и вместо одной выполняются две вычислительные команды, время работы его меньше.

Однако не следует думать, что длина программы никогда не имеет значения. Во-первых, встречаются задачи, программы решения которых без минимизации просто не помещаются

в программную память микрокалькулятора. Таковы, например, приведенные в уже упомянутом сборнике А. Н. Цветкова и В. А. Епанечникова программы решения системы четырех линейных уравнений с четырьмя неизвестными (91 ячейка) или программа, вычисляющая 7 различных функций комплексного переменного (98 ячеек). Во-вторых, чем короче программа, тем меньше вероятность ошибки при ее наборе. В-третьих, само время ввода при этом меньше, а оно тоже входит в общее время получения результата. Поэтому знание приемов, позволяющих сокращать программу без ущерба для остальных ее параметров, весьма полезно.

Рецепты по минимизации длины программы в основном просты: следует избегать повторения одинаковых последовательностей команд, то есть если программа ветвистая — выносить их из ветвей в «общий ствол», а в других случаях оформлять эти последовательности в виде подпрограмм. Второй путь, правда, не всегда приводит к ожидаемому результату. Действительно, если в нашей программе есть k

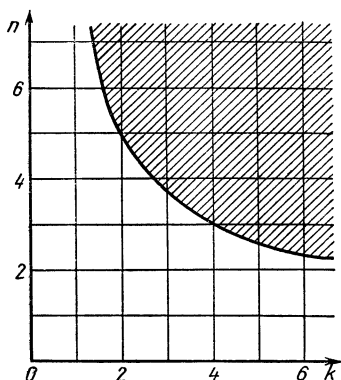


Рис. 16

повторяющихся фрагментов, длина каждого из которых — n команд, то их общая длина — nk команд. Если же их вынести в подпрограмму, то они займут $2k + n + 1$ ячеек ($2k$ — количество команд обращения к подпрограмме, n — ее длина и еще единица — команда В/О, замыкающая подпрограмму). Ясно, что применять этот прием есть смысл только в случаях, когда $nk > 2k + n + 1$.

Заменим в этом соотношении знак неравенства на равенство и после элементарных преобразований построим зависимость $n(k)$: $n = (2k + 1)/(k - 1)$ (рис. 16). На графике штриховкой выделена область значений n и k , для которых выгодна замена одинаковых последовательностей команд подпрограммой. Видно, что достичь экономии можно только при длине подпрограммы, не меньшей трех команд. В этом случае подпрограмму стоит применять, если $k > 4$. При $n > 5$ экономия достигается уже при двукратном обращении к подпрограмме. Однако следует помнить, что выделение фрагмента в подпрограмму всегда приводит к увеличению времени счета, так как время обращения к подпрограмме довольно велико,

примерно вдвое больше времени выполнения вычислительной операции. Поэтому если к подпрограмме обращаются внутри цикла, это может привести к значительному увеличению общего времени счета.

С другой стороны, не надо забывать, что основная цель подпрограммы — не экономить команды, а делать программу универсальной. Поэтому иногда бывает оправданным выделение в подпрограмму даже однократно используемого фрагмента.

Вообще же искусство сокращать программы можно сравнить с умением быстро бегать. Применять его всегда вроде бы не обязательно, однако иногда только с его помощью можно достигнуть цели — скажем, успеть на уходящий поезд. Так и в программировании: постоянная привычка писать как можно более короткие программы иногда позволяет укладывать весьма сложный алгоритм на прокрустово ложе калькуляторной памяти.

А теперь рассмотрим некоторые из описанных приемов и попытаемся с их помощью сократить программу решения квадратного уравнения, приведенную в разделе 5.

В программе есть дважды повторяющаяся последовательность команд по адресам 43—47 и 56—60. Заменяв первую последовательность командами БП 56, то есть вынеся их за пределы ветвей в «общий ствол», мы уже сэкономим три команды.

Далее. Даже после этой замены в двух местах программы останутся одинаковые команды перехода: БП 59. Они используются для передачи управления последней команде программы, а та, в свою очередь, возвращает управление к началу программы. Такая запись была сделана, чтобы показать более наглядно связь программы с блок-схемой. Ничего не изменится, если вместо БП 59 записать БП 00. Тогда таких команд в нашей программе будет три. Но можно передать управление к началу программы более коротким путем, используя команду В/О. Команда эта, если она выполняется в ходе автоматических вычислений по программе, обнаруживает интересное свойство. Если перед ней не встречается команда ПП (Переход на Подпрограмму), то она очищает программный указатель, устанавливая его, правда, не на нуль, а на единицу. Иначе говоря, команда В/О в таких случаях эквивалентна команде БП 01. Значит, нужно сделать так, чтобы программа начиналась с адреса 01. Проще всего для этого записать по адресу 00 команду КНОП (Нет Операций). Тогда, трижды заменив команду БП 00 командой В/О, мы укоротим программу еще на пару команд.

Наконец, обратите внимание на вывод информации. После «останова» все результаты счета находятся в стеке: в регистре

X — шифрованное сообщение о природе корней уравнения, а в регистрах Y и Z (или только Y) — сами корни. Прочитав сообщение, мы нажимаем клавишу «С/П», а затем «XY». Не проще ли после появления сообщения один или два раза нажать клавиши FO? Программа при этом еще более сократится.

Новая программа, полученная с учетом описанных изменений, приведена на табл. 7. Она почти на 10 команд короче первого варианта. При этом мы не пожертвовали ни удобствами, ни скоростью ее выполнения, да и времени на ее модернизацию потратили немного. Конечно, такое сокращение можно только приветствовать.

Таблица 7

Адрес	Команда	Код	Адрес	Команда	Код
00	КНОП	54	26	+	10
01	ПА	4—	27	ИПВ	6L
02	С/П	50	28	ИПА	6—
03	С/П	50	29	÷	13
04	ИПА	6—	30	/—/	0L
05	$Fx = 0$	5E	31	П1	41
06	19	19	32	XY	14
07	FO	25	33	$Fx < 0$	5C
08	XY	14	34	42	42
09	$Fx \neq 0$	57	35	/—/	0L
10	16	16	36	$F\sqrt{\quad}$	21
11	÷	13	37	ИПА	6—
12	/—/	0L	38	÷	13
13	ИПЗ	63	39	ИП5	65
14	С/П	50	40	С/П	50
15	В/О	52	41	В/О	52
16	ИП0	60	42	$F\sqrt{\quad}$	21
17	С/П	50	43	ИПА	6—
18	В/О	52	44	÷	13
19	×	12	45	+	10
20	/—/	0L	46	ИП1	61
21	XY	14	47	FBx	0
22	2	02	48	—	11
23	÷	13	49	ИП4	64
24	ПВ	4L	50	С/П	50
25	Fx^2	22	51	В/О	52

Иногда сокращения программы добиваются использованием вместо команд прямой адресации команд косвенной адресации.

Правда, хотим предостеречь вас от бездумного увлечения этим способом сокращения. Хотя команды косвенного перехода занимают в памяти всего одну ячейку вместо двух, занимаемых командами прямого перехода, зато при этом

используются адресуемые регистры. А их в нашем распоряжении существенно меньше, чем ячеек программной памяти.

Однако, пожалуй, самый радикальный путь к сокращению программы — выбор более компактного алгоритма. Вернемся к задаче расчета полного сопротивления участка цепи (рис. 15). Запишем последовательность расчетных формул по-другому:

$$R_{23} = \frac{1}{1/R_2 + 1/R_3}; \quad R_{123} = R_1 + R_{23}; \quad R_{AB} = \frac{1}{1/R_{123} + 1/R_4}.$$

Составление программы по этим формулам, да и отладка ее большого труда не составит. Вот эта программа (табл. 8).

Таблица 8

Адрес	Команда	Код	Адрес	Команда	Код
00	ИП2	62	07	+	10
01	F1/x	23	08	F1/x	23
02	ИП3	63	09	ИП4	64
03	F1/x	23	10	F1/x	23
04	+	10	11	+	10
05	F1/x	23	12	F1/x	23
06	ИП1	61	13	С/П	50

Она без применения каких-либо специфических приемов сразу оказалась на три команды короче.

Какие же выводы можно сделать из всего вышесказанного?

1. Прежде чем начать отладку программы, убедитесь, что она соответствует написанному тексту.

2. Для редактирования программы используйте команды **ШГ** и **ШГ** в режиме программирования и команды **В/О** и **БП** (*mn* — адрес) в режиме вычислений.

3. Если неправильно записан адрес в команде перехода, то для исправления его вернитесь на шаг назад и снова введите саму команду, а уж затем исправленный адрес.

4. При отладке программ желательно использовать команду **ПП** (Потактовый Проход). С ее помощью быстрее всего «вылавливается» большинство ошибок. Однако помните, что полную уверенность в работоспособности программы дает только контрольный пример, пропущенный в автоматическом режиме.

5. Не забывайте, что программа — это не самоцель, а средство решения определенной задачи. Поэтому оптимизировать следует в первую очередь не программу, а весь процесс получения результатов по поставленной задаче.

6. Если вы решили оптимизировать программу, то прежде всего старайтесь уменьшить время ее работы.

7. Привыкайте применять алгоритмы, требующие для выполнения возможно меньшего числа операций. Это избавит вас от необходимости сокращать составленную программу, когда ее размер превышает программную память микрокалькулятора.

9. ПОГРЕШНОСТИ ВЫЧИСЛЕНИЙ

На уроке истории учитель спросил: «Когда была Куликовская битва?». «В 1242 году», — ответил первый ученик. «В XIV веке», — ответил второй. «В 1380 году», — был ответ третьего. «Первый ученик допустил ошибку, его ответ неверен, — сказал учитель, — ответ второго правильный, но не достаточно точный, третий назвал дату абсолютно верно».

Математик бы, наверное, оценил ответы иначе. Скорее всего, он сказал бы так: «Результаты всех учеников даны с погрешностями, причем погрешность первого ответа наибольшая, а последнего наименьшая. Отклонение его результата от истинной величины не превышает одного года». (Для справки. Куликовская битва состоялась 8 сентября 1380 года.)

Вообще говоря, в быту мы привыкли называть ошибкой результат оплошности, незнания или неумения. Слово «ашипка», сказали бы мы, написано с ошибками, а в слове «ошибка» ошибок нет. В этом смысле ответ второго ученика нельзя признать ошибочным. В подобных случаях на помощь приходит слово «точность» и появляется не очень гладкое словосочетание «правильно, но не точно». Наконец, мы очень часто употребляем слова «абсолютно верно», «без единой ошибки». Границы в употреблении терминов «ошибка» и «погрешность» расплывчаты. Часто эти слова употребляются как синонимы. Следуя традиции, мы для характеристики точности будем использовать термин «погрешность».

Далее. Абсолютно точные результаты встречаются в вычислительной математике не намного чаще, чем эдельвейсы в городских парках. Поэтому слово «погрешность» сопутствует описанию решения любой практической задачи, любого численного метода.

Что же это такое?

Из железнодорожного справочника можно узнать, что расстояние между Москвой и Ленинградом равно 650 км. Если вы вооружитесь линейкой и измерите высоту страницы книги, которую вы читаете, то убедитесь, что она равна 198 мм. Какой из этих результатов точнее?

Расстояния, приведенные в справочнике, выражены в километрах и округлены до целого числа. То есть фактическое расстояние может отличаться от приведенного на полкилометра в любую сторону. При измерении длины страницы мы пользовались линейкой с миллиметровыми делениями. Таким образом, здесь мы могли ошибиться не более, чем на полмиллиметра.

Итак, в одном случае — 0,5 км, а в другом — 0,5 мм. Вроде бы все ясно. Страница измерена точнее. Но не будем торопиться. Погрешность, с которой измерено расстояние между городами, составляет $0,5/650 = 1/1230$ этого расстояния. Соответствующая характеристика второго измерения равна $0,5/198 = 1/396$, то есть явно больше. Выходит, первая величина точнее? Нет ли здесь противоречия? Нет. Просто в первом случае мы оценивали *абсолютную погрешность*, а во втором — *относительную*.

Первая из них, абсолютная погрешность, определяется как модуль разности между истинной величиной и ее приближенным представлением. Вторая, относительная погрешность, равна отношению абсолютной погрешности к модулю самой величины. В различных ситуациях на первый план выступает то одна, то другая.

Абсолютно точных измерений не бывает по целому ряду причин. Отметим некоторые из них. Во-первых, если в нашем «железнодорожном» примере речь идет о расстоянии, понимаемом как общая длина рельсов, то оно не остается постоянным как в течение года, так и даже в течение суток. Ведь рельсы стальные, и длина их зависит от температуры. Во-вторых, — здесь мы обратимся к примеру с измерением книжной страницы — даже если пренебречь этим эффектом, то сам измерительный прибор не в состоянии обеспечить идеальной точности. К примеру, используя линейку, мы не можем добиться точности большей, чем цена ее деления, минимальное расстояние между нанесенными на нее рисками. И наконец, в-третьих, самый идеальный измерительный прибор не может дать точности, большей, чем длина волны падающего света (около 10^{-8} м). Таким образом, если исходная величина является результатом измерения, не важно чего — расстояния, времени или электрического тока — в ней всегда присутствует некоторая погрешность. Абсолютно точно задаются лишь некоторые коэффициенты в формулах, и то, если они выражаются целыми числами или конечными десятичными дробями.

Это в первую очередь нужно знать тем, кто проводит вычисления, в частности, владельцам микрокалькуляторов.

Борясь за увеличение точности вычислений, всегда нужно помнить, что она с самого начала ограничена погрешностями исходных данных и увеличивать ее без меры нецелесообразно.

Погрешности возникают и в процессе машинных вычислений. В этом можно убедиться на простом примере. Попытайтесь выполнить с помощью своего электронного друга вычисления по такой цепочке: $(1/3) \times 7 \times 3/7$. Нетрудно убедиться, что точный результат должен быть равен единице. ПМК же выдаст: $9,9999998 \cdot 10^{-1}$. Две единицы в последнем разряде исчезли. Если тот же цикл повторить еще раз, то получим: $9,9999994 \cdot 10^{-1}$. Погрешность достигла уже 6 единиц последнего разряда. Проведя еще несколько экспериментов такого же рода, вы убедитесь, что последнему знаку результата верить нельзя.

Причина подобных погрешностей проста. Микрокалькулятор, да и вообще любая ЭВМ, работает с ограниченным числом разрядов. Иначе говоря, сохраняет лишь определенное количество цифр результата (наш ПМК — 8). Остальные цифры отбрасываются. При этом, если первая из отброшенных цифр равна или больше пяти, к последней оставшейся цифре прибавляется единица. Такой процесс называется *округлением*. Так происходит при сложении и вычитании. При проведении других операций наш калькулятор просто отбрасывает «лишние» цифры. Погрешность, возникающая в результате этих процессов, называется *погрешностью округления* или *отбрасывания*.

Существование этих ошибок приводит к забавным парадоксам. Так, например, при вычислениях на ЭВМ не выполняется (по крайней мере не всегда выполняется) переместительный закон сложения, тот самый, что вошел в поговорку: «от перемены мест слагаемых сумма не меняется». Так вот, при вычислениях на ЭВМ она меняется!

Сложим на нашем ПМК пять чисел: 100013,26; 10004,624; 1000,5432; 100,04365 и 100,03416. (Советуем, набирая числа на клавиатуре, одновременно записывать их в адресуемые регистры. Числа эти еще понадобятся.) Результат: 111218,49. А теперь сложим те же числа в обратном порядке, от пятого к первому. Результат равен: 111218,51. Сумма изменилась!

Не выполняется и распределительный закон, выражаемый формулой $a(b - c) = ab - ac$. Проверим это. Пусть $a = 5$, $b = 2,0000008$, $c = 2,0000007$. Вычисление по первой формуле дает $5 \cdot 10^{-7}$, а по второй — нуль.

Наконец, что довольно не очевидно, сочетательный закон при расчетах на ЭВМ справедлив тоже не всегда. Иначе говоря, $a + b + c + d \neq (a + b) + (c + d)$. Не будем отказывать вам в удовольствии подобрать пример самостоятельно. Скажем

лишь, что более рельефно невыполнение это заметно, если складываемые числа близки по величине.

Как же объяснить эти парадоксы? Ведь на законах, которые мы «опровергли», зиждется вся математика.

Парадокс на то и парадокс, что законов не опровергает. Просто он заставляет взглянуть на них по-новому. Так же, как разное время падения камня и перышка, брошенных с одинаковой высоты, не отменяет закона всемирного тяготения, так и наши эксперименты ни в коей мере не бросают тень на фундаментальные законы математики. В обоих случаях кажущееся нарушение законов — результат действия среды. В первом случае роль «нарушителя» выполняет сопротивление воздуха, а во втором — специфика вычислений на ЭВМ. Зная, откуда берутся и как проявляют себя эти «нарушители законов», мы сможем в обоих случаях правильно оценить полученный результат.

Прежде чем приступить к краткому изложению некоторых правил, описывающих распространение вычислительных погрешностей, рассмотрим еще один тип погрешностей, характерный для численных методов.

Если причина погрешностей первого типа — погрешностей округления — кроется в самом вычислительном устройстве, то погрешности другого типа, о которых пойдет речь, *погрешности ограничения*, связаны со спецификой численных методов.

Основная идея любого численного метода — замена бесконечного конечным. Например, вычисляя значение довольно распространенной в математике функции e^x , мы пользуемся рядом:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \dots + \frac{x^n}{n!} + \dots$$

($n!$ читается «эн факториал». Эта величина равна произведению всех положительных целых чисел от 1 до n).

Знак равенства, стоящий в этой формуле, строго говоря, относится к случаю бесконечного числа членов ряда, записанного справа от него. Фактически же мы можем обработать хоть и любое, но конечное число членов. Отбрасываемые члены и дают погрешность ограничения. Та же картина наблюдается, когда численным способом находятся определенные интегралы. Интервал интегрирования разбивается на ряд участков, где интегрируемая функция заменяется более простыми функциями, например горизонтальными отрезками прямых (рис. 17). Это так называемый метод прямоугольников. Существуют также методы трапеций и парабол. При этом площадь, ограниченная кривой, только тогда будет

равна сумме площадей прямоугольников, трапеций или парабол, когда величины отрезков разбиения будут стремиться

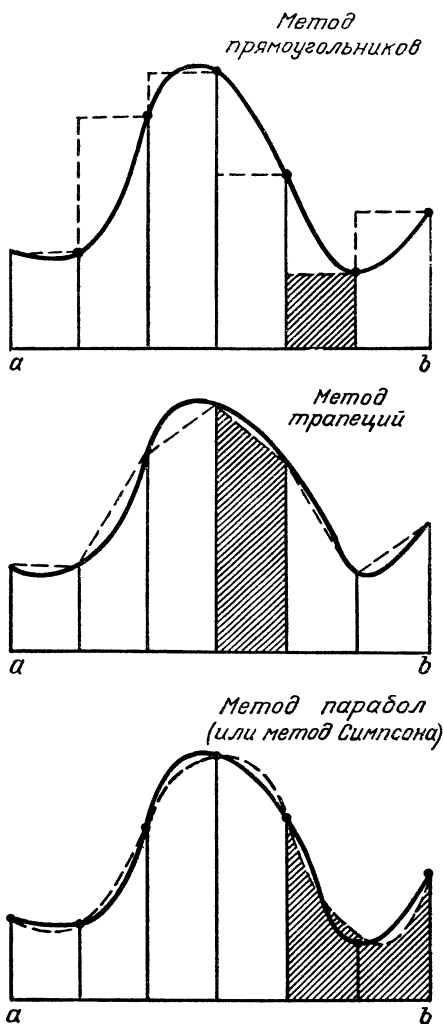


Рис. 17

к нулю. При конечных же величинах мы опять имеем дело с погрешностями ограничения. Наглядный пример накопления погрешностей ограничения дает подпрограмма вычисления функции x^y , реализованная на нашем ПМК. Вычисляя с ее

помощью, скажем 2^2 , мы получаем погрешность в восьмом знаке результата: 3,9999996.

Интересно, что разные причины возникновения погрешностей при вычислениях на ЭВМ делают невозможным их одновременное уменьшение. Действительно, при отыскании величины e^x уменьшить погрешности ограничения можно, увеличив число членов разложения, при вычислении определенных интегралов — уменьшив длины отрезков разбиения. Но при этом возрастает количество вычислительных операций. Каждая из них вносит погрешность отбрасывания, следовательно, суммарная погрешность тоже будет возрастать. Это наглядно иллюстрирует рис. 18, где изображена суммарная

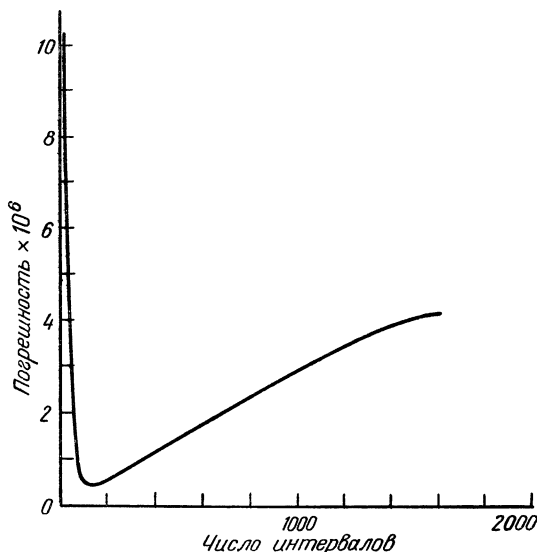


Рис. 18

погрешность при численном вычислении интеграла по методу парабол.

Итак, мы выяснили, что при численных расчетах на ЭВМ приходится считаться с погрешностями трех типов: погрешностью задания исходных данных (их причина — неточности измерений), погрешностью ограничения (причина — специфика численных методов) и погрешностью отбрасывания (причина — ограниченная разрядность ЭВМ).

Бороться с погрешностями первого типа при работе на машине нам не под силу — ведь они допущены уже в исходных данных. Однако знать их необходимо. Это помогает правиль-

но выбрать алгоритм обработки, чтобы, с одной стороны, не мучиться понапрасну, пытаясь вычислять, скажем с точностью до пятого знака, результаты, когда в исходных данных верны лишь три знака, а с другой — зная, как распространяются погрешности, правильно оценить точность полученного результата.

Кстати, вопрос оценки точности при численных расчетах довольно непрост. Ведь по определению погрешность — это разность между истинной величиной и ее приближенным представлением. Но истинной величины мы не знаем (иначе к чему расчеты?). Так что из чего вычитать?

Можно, конечно, зная погрешности исходных данных, погрешности метода и погрешности вычислительных операций, вычислить максимальную погрешность результата. Как это делается, мы скажем дальше. Однако в большинстве случаев сделать это затруднительно. Поэтому поступают проще. Вычисляют, скажем, интеграл при некоторой длине отрезка разбиения, затем каждый из этих отрезков делят пополам и повторяют процесс. Разность полученных результатов и принимают за абсолютную погрешность. В большинстве практических применений этот метод оказывается удовлетворительным, но, к сожалению, не всегда. Один из примеров, где такой критерий не работает, мы покажем ниже.

Как же бороться с погрешностями?

Что касается погрешностей ограничения, то это ясно из самой их структуры. Увеличивая число членов ряда или уменьшая длины отрезков разбиения, можно в принципе сделать эти погрешности сколь угодно малыми.

Для того чтобы уменьшить погрешности округления, надо знать, как они ведут себя при различных вычислительных процессах.

Этими вопросами занимается специальный раздел вычислительной математики. Существуют формулы, выражающие связь между погрешностями исходных данных и погрешностями результатов операций. Мы приведем лишь некоторые из них.

Закон распространения погрешностей при суммировании двух чисел, x и y , описывается такой формулой:

$$\frac{e_s}{|x+y|} \leq \left| \frac{x}{x+y} \right| e_x + \left| \frac{y}{x+y} \right| e_y + t,$$

здесь e_s — абсолютная погрешность результата, e_x и e_y — относительные погрешности исходных величин x и y , t — погрешность округления.

Для вычитания формула имеет почти такой же вид. Только в знаменателях вместо сумм стоят разности. Абсолютная погрешность при этих операциях не превосходит суммы абсолютных погрешностей участников операции плюс еще некоторая добавка $t|x+y|$ (или $t|x-y|$), которая, как видно, при сложении даже больше. Однако относительная погрешность ведет себя иначе. При сложении она не превосходит наибольшей из относительных погрешностей слагаемых плюс погрешность округления. При вычитании же она может быть сколь угодно большой, тем большей, чем ближе друг к другу уменьшаемое и вычитаемое. Она может быть настолько большой, что ни одна из цифр результата не будет верной.

Обратите внимание, что погрешность появляется и тогда, когда исходные данные заданы абсолютно точно. Ведь погрешность округления t , входящая в формулы, никак не зависит от степени точности исходных величин. Здесь как раз кроется разгадка парадоксов, которые мы приводили выше. Проиллюстрируем один из них графом вычислительного процесса. *Графом* называется произвольный контур, состоящий

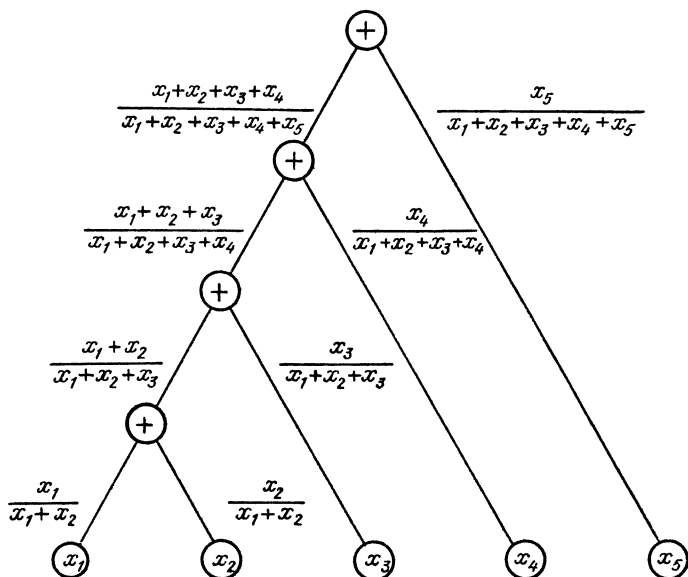


Рис. 19

из некоторого числа вершин, соединенных «ребрами». Граф вычислительного процесса является весьма удобным методом, позволяющим проследить, как изменяется погрешность при

вычислениях. В крайних вершинах нашего графа поместим исходные данные, в промежуточных — знаки операций, а около ребер напомним коэффициенты, с которыми исходные погрешности входят в следующий результат. Граф для последовательного сложения пяти чисел изображен на рис. 19. Если пренебречь погрешностями исходных данных и учитывать только относительные погрешности округления, то после несложных алгебраических преобразований мы получим формулу для абсолютной погрешности результата:

$$e_s = (4x_1 + 4x_2 + 3x_3 + 2x_4 + x_5) t.$$

Видно, что вклад слагаемых в общую погрешность неодинаков. Допустим, что одно из них (например x_1) много больше остальных. Поэтому если поставить его на первое место, общая погрешность будет примерно равна $4x_1 t$, если же на последнее, то $x_1 t$, то есть в 4 раза меньше! Вот вам и объяснение парадокса с переменной мест слагаемых. При первом сложении самая большая величина стояла в начале суммы, а при втором — в конце. Разные величины погрешностей и привели к разным результатам.

Так же можно показать, что разные погрешности получаются при сложении четверки чисел подряд и попарно. Погрешность при сложении парами меньше. В отличие от первого примера здесь, как уже говорилось, эффект сильнее ощутим при сложении примерно одинаковых чисел.

В разрешении этих парадоксов кроются простые рецепты по увеличению точности вычислений.

Однако погрешности, появляющиеся при сложении, не идут ни в какое сравнение с погрешностями, возникающими при вычитании. Поэтому если уж бороться за увеличение точности, то начать следует именно с вычитания. Это — самый страшный источник погрешностей. Мы уже говорили, что погрешность при этом процессе может возрасти настолько, что полностью «замаскирует» полученный результат.

Наглядный пример такого эффекта дает классический пример вычисления синуса с помощью разложения его в ряд по формуле

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + \frac{(-1)^{n-1} x^{2n-1}}{(2n-1)!} + \dots$$

С точки зрения чистой математики эта формула верна при любых значениях x . Однако на практике использовать ее для машинных расчетов при больших значениях аргумента невозможно. В этом вы можете убедиться сами, используя программу, данную в приложении к разделу.

Введите эту программу в ПМК, проверьте правильность ввода, пропустив контрольный пример, и приступайте к эксперименту. Сосчитав по программе несколько значений синуса для аргументов, не больших единицы, убедимся, что полученные числа отличаются от даваемых микрокалькулятором по команде $F \sin$ не больше, чем на заданную погрешность. Однако по мере роста аргумента неточность будет расти. Погрешность становится явно больше заданного значения. Наконец, при x , измеряемых десятками, мы начнем получать совсем уж несусветные величины. Так $\sin 20 = 1,297$, а $\sin 30 = -1986,1271$ (!). Микрокалькулятор с завидной педантичностью дает 8 значащих цифр, из которых не только ни одна не верна, но и в самой величине кроется погрешность на 4 порядка. Причина — в погрешностях, возникающих при использовании элементарных арифметических операций сложения и вычитания. Так как вначале получаются довольно большие числа, то хотя абсолютная погрешность каждого вычитания невелика, относительная нарастает настолько сильно, что полностью затушевывает результат.

Попытка увеличить точность за счет добавки лишних членов ни к чему не приводит. Снизить погрешность, возникающую за счет округления, не удастся. Это как раз тот случай, когда оценить погрешность, сравнивая результаты расчетов для различного числа членов ряда, невозможно. С такими погрешностями бороться вообще трудно. В принципе, если каждый член ряда запомнить, а затем расположить их по «ранжиру», от меньшего к большему, и только после этого сложить, а еще лучше отдельно сложить ранжированные положительные и отрицательные члены, а затем получить их разность, результат будет несколько точнее. Но такой способ явно неудобен. Лучше в подобных случаях действовать кардинальным образом — менять алгоритм вычислений. В нашем примере целесообразно, используя периодичность синуса, приводить большие аргументы к значениям, меньшим, скажем, $\pi/2$. Правда, при этом нужно следить, как меняется знак функции. Зато, как мы знаем, при малых x точность представления синуса рядом вполне удовлетворительная. Кстати, примерно так и работает встроенная в микрокалькулятор подпрограмма вычисления синуса.

Подобные ситуации с большими погрешностями, особенно при вычитании, вообще не редкость при численных расчетах. Даже при решении квадратного уравнения, о котором мы уже много писали, наблюдается потеря точности, когда корни сильно различаются. Так, решая по приведенным в предыдущих разделах программам уравнение $x^2 - 1,0000001x + 10^{-7} = 0$,

вы получите $x_1 = 1,1 \cdot 10^{-7}$, $x_2 = 9,9999999 \cdot 10^{-1}$. Точное решение этого уравнения получить нетрудно: $1 \cdot 10^{-7}$ и 1. Если относительная погрешность второго результата мала (10^{-7}), то для первого она в миллион раз больше (10^{-1}). Причина, как и ранее, — вычитание двух близких чисел. И здесь бороться с погрешностью нужно, модернизируя алгоритм, а именно больший корень, x_1 , вычислять по обычным формулам, а меньший уточнять с помощью соотношения $x_1 x_2 = c/a$. Кстати, подумайте, как для этого изменить программу, приведенную в разделе 8.

Как видите, хотя неточность неминуемо сопутствует численным расчетам, борьба за ее уменьшение небесперспективна. Зная причины погрешностей, почти всегда можно их уменьшить.

Перечислим некоторые приемы увеличения точности счета.

1. Складывая разные числа, начинайте с малых. При сложении одинаковых по порядку чисел желательно разбивать их на группы, получать частные суммы групп и затем их складывать. Точность при большом количестве слагаемых при этом значительно возрастет.

2. Старайтесь записывать цепочки вычислений так, чтобы по возможности избегать вычитания близких чисел. Это вносит самые большие погрешности в результат.

3. Тщательный выбор алгоритма позволяет, как правило, существенно повысить точность вычислений.

4. При решении практических задач необходимо знать погрешность исходных данных. Борьба за достижение точности, значительно превышающей точность входных величин, приводит лишь к бессмысленной трате времени и сил.

5. Точность при численных расчетах безусловно необходима, но «недостатки математического образования с наибольшей отчетливостью проявляются в чрезмерной точности численных расчетов». Советуем запомнить эти слова, сказанные почти полтора столетия тому назад выдающимся немецким математиком К. Ф. Гауссом.

Приложение. Считаем синус

Прежде чем приступить к составлению программы для расчета по формуле, приведенной в этой главе, надо, как и всегда, составить алгоритм. Самый простой на первый взгляд путь счета — задать значение x и затем последовательно вычислять каждый член ряда, возводя x в нужную степень и деля на соответствующий факториал. Но это нерацionalmente.

Проанализировав выражение для общего члена ряда, легко убедиться, что каждый член отличается от предыдущего на однотипный сомножитель

$$x_i = x_{i-1} \frac{-x^2}{(2i-2)(2i-1)}.$$

Это уже проще. Не надо возводить x в большие степени и считать факториалы. Если же привести формулы к «машинному виду», то есть записать в виде

$$k = 2(i-1), x_i = x_{i-1} \frac{-x^2}{k(k+1)},$$

то вычисления станут еще быстрее, да и программа короче.

Блок-схема приведена на рис. 20. Символом y обозначен x , s — это сумма членов ряда, ε — планируемая абсолютная погрешность.

Обратите внимание, что сравнение $|x_i| < \varepsilon$ заменено на эквивалентное $x_i^2 < \varepsilon^2$. Это сделано, чтобы избежать вычисления модуля $|x_i|$, так как специальной команды для этого в ПМК не существует; запись же вычисления модуля величины в виде нескольких команд и время работы увеличит, и программу удлинит. Далее. Вместо номера члена i фигурирует величина $i_1 = i - 1$. Так как в других сочетаниях величина i в алгоритме не встречается, то можно с самого начала задавать вместо номера члена величину, на единицу меньшую. Это избавит от одного вычитания. Все остальное понятно без комментариев. Программа, составленная по этой схеме, приведена на табл. 9. Увеличение i_1 на единицу сделано с помощью одной команды — КИП5. Формально вызывая содержимое регистра, номер которого записан в R5, мы тем самым увеличиваем на единицу величину, находящуюся в нем самом.

В программе задействованы регистры $\varepsilon^2 \rightarrow RД$, $y \rightarrow R0$, $s \rightarrow R1$, $i_1 \rightarrow R5$ (счетчик).

Инструкция для пользования программой.

1. Ввести программу.
2. Перейти в режим вычислений (FABT).
3. Очистить программный указатель (B/O).

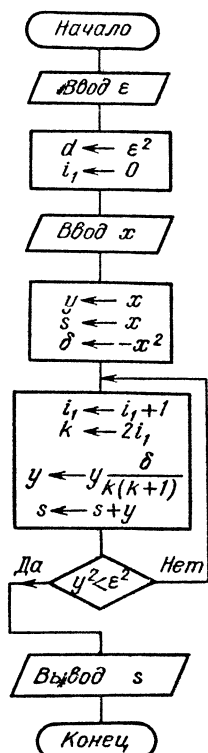


Рис. 20

Таблица 9

Адрес	Команда	Код	Регистры			
			X	Y	Z	T
ВВОД ε			ε			
00	Fx^2	22	ε^2			
01	ПД	4Г	ε^2			
02	Cx	0Г	0			
03	П5	45	0			
04	С/П	50	0			
ВВОД x			x			
05	П0	40	x			
06	П1	41	x			
07	Fx^2	22	x^2			
08	/—/	0L	$-x^2$			
09	ПА	4—	$-x^2$			
10	КИП5	Г5		$-x^2$		
11	ИПА	6—	$-x^2$			
12	1	01	1	$-x^2$		
13	ИП5	65	k	1	$-x^2$	
14	2	02	2	k	1	$-x^2$
15	\times	12	$2k$	1	$-x^2$	
16	+	10	$2k+1$	$-x^2$		
17	FVx	0	$2k$	$2k+1$	$-x^2$	
18	\times	12	$2k(2k+1)$	$-x^2$		
19	\div	13	$-x^2$			
			$2k(2k+1)$			
20	ИП0	60	x	$-x^2$		
				$2k(2k+1)$		
21	\times	12	x_i			
22	П0	40	x_i			
23	ИП1	61	s_{i-1}			
24	+	10	s_i			
25	П1	41	s_i			
26	ИП0	60	x_i	s_i		
27	Fx^2	22	x_i^2	s_i		
28	ИПД	6Г	ε^2	x_i^2	s_i	
29	—	11	$x_i^2 - \varepsilon^2$	s_i		
30	$Fx < 0$	5Г	$x_i^2 - \varepsilon^2$	s_i		
31	10	10	$x_i^2 - \varepsilon^2$	s_i		
32	ИП1	61	s_i			
33	С/П	50	s_i			
34	БП	51				
35	00	00				

4. Ввод: ε С/П x С/П.
5. Результат счета — на индикаторе.
6. Продолжение работы.
 - 6.1. Для нового значения x перейти к п. 3.
 - 6.2. Для уточнения результата (при меньшем значении ε):
В/О ε С/П БП 10 С/П.

Контрольный пример. Пусть $\varepsilon = 1 \cdot 10^{-5}$, $x = 0,1$. Результат: $9,9833413 \cdot 10^{-2}$ ($\sin 0,1$, вычисленный по программе $F \sin$, заложенной в ПМК, равен $9,9833417 \cdot 10^{-2}$).

10. СЕРВИС СЛУЖБЫ ПРОГРАММИРОВАНИЯ

В предыдущих разделах мы говорили о том, как создавать изделия-программы и как делать их более экономичными, быстрыми и точными.

Сейчас речь пойдет об удобствах, программном сервисе. Это тоже немаловажный аспект, ведь работа с более удобным инструментом не только приятнее, но и эффективнее.

Выемка на ручке топора делается не для того, чтобы он лучше выглядел (хотя и это немаловажно), а чтобы удобнее было обхватить топорище рукой, и разноцветные кнопки на диспетчерском пульте в аэропорту не для того только, чтобы радовать глаз, но прежде всего, чтобы облегчить работу диспетчера, предотвратить его ошибки.

То же самое относится и к программе. Более удобная программа экономит время пользователя, уменьшает вероятность его ошибок, рождает положительные эмоции — все это, в конце концов, способствует повышению производительности труда. А это — одна из основных задач нашего общества.

Говоря о сервисе программирования, очевидно, стоит выделить два его аспекта: удобства для пользователя и удобства для программиста.

Сервис для программиста — это, если продолжить технические аналогии, набор удобных, высококачественных инструментов, с помощью которых специалист может сделать свое изделие быстрее и лучше. Готовят этот сервис в первую очередь разработчики микрокалькуляторов. Сюда относятся средства редактирования и отладки программ. О них мы подробно писали в разделе 8. Дополняют набор удобств и некоторые приемы, облегчающие создание программ. В частности, это составление блок-схемы алгоритма, сначала укрупненной, потом детализированной, запись содержимого регистров стека после каждой команды, снабжение программы подробными комментариями, использование клавиши «ПП» для потактного

прохода программы. Об этом мы тоже писали. Здесь можно добавить еще несколько советов.

Так как все команды, вводимые в память, располагаются одна за другой, то вставить между ними новую команду (а потребность в этом довольно часто возникает в процессе отладки) невозможно: приходится вводить весь «хвост» программы за дополнительной командой заново. Если же в первом варианте программы после, скажем, каждой десятой команды, вставлять пустые команды КНОП (конечно, если объем памяти позволяет это сделать), то процесс вставок будет проходить менее болезненно: перебирать заново нужно будет лишь несколько команд.

Еще одна загвоздка, особенно часто встречающаяся при отладке программ: *ошибки переполнения*. Так называются ситуации, когда машина не в состоянии выполнить какую-либо операцию, например разделить число на нуль или извлечь корень из отрицательного числа. На индикаторе при этом появляется сообщение ЕГГОГ. Причину его выяснить можно сразу. Для этого надо перейти в режим программирования (ФПРГ). На индикаторе будет высвечиваться код операции, на которой произошла ошибка. Из высвечиваемых кодов в зависимости от ситуации это может быть либо самый левый код, либо тот, что находится рядом с ним, но правее его. Если теперь вновь перейти в режим вычисления (FABT), то на индикаторе появится и число — «виновник» ошибки.

Немаловажен с точки зрения сервиса и сам способ записи программ. Запись типа «адрес — команда — код — стек — комментарий» дает наиболее полную информацию о программе, но она слишком громоздка. Это создает большие неудобства, когда программу приходится публиковать в книге или журнале. Существует множество предложений по более компактной записи программ: это и соображения по использованию более наглядной символики, и предложения объединить группы команд в операторы, и просто технические приемы, например запись программы в виде таблицы по десять команд в строке (таким образом экономится место для написания адресов), и отказ от записи кодов, и т. д.

У каждого из этих предложений есть очевидные достоинства, и мы не будем их обсуждать. Основной же недостаток всех этих предложений — отсутствие стандарта, общего для всех пользователей. Когда вы используете эти приемы для себя, то это ваше личное дело. Вы можете выбрать любой наиболее удобный для вас способ. Если же вы хотите обмениваться программами с другими пользователями, то приходится каждую из них снабжать подобием словаря, с помощью

которого другой программист может перевести ее с «вашего» языка на язык микрокалькулятора. Поэтому мы воздержимся от рекомендаций относительно каких-нибудь оригинальных способов записи программ. Лучше, по крайней мере пока, пользоваться языком и способом записи программ, который приведен в заводской инструкции к микрокалькулятору и который мы с небольшим дополнением (колонка, где показано заполнение регистров стека, и колонка комментариев) используем в этой книге. Единый способ записи облегчает обмен программами и предотвращает ошибки.

Теперь перейдем к вопросу о сервисе для пользователя.

Для пользователя программа — черный ящик, начинка которого ему в общем-то неинтересна. Ему важно знать лишь возможности программы и методику работы с ней. Судит пользователь о программе по ее внешним признакам: времени работы, появлению тех или иных символов на индикаторе. Знакомство с программой он начинает с инструкции к ней. Инструкция должна давать ему информацию и о возможностях программы, и о действиях, которые надо выполнять для обеспечения ее нормальной работы. Иначе говоря, инструкция играет ту же роль, что, скажем, инструкция для владельца телевизора, которая описывает, как включать и выключать аппарат, как переключать программы, регулировать громкость звука и т. д.

Однако одной инструкцией сервис, естественно, не ограничивается. Представьте себе, что в инструкции к телевизору было бы указано, что для включения его нужно нажать три кнопки в определенной последовательности, а чтобы переключить программу — повернуть пару ручек, но при этом номер программы нигде бы не был указан. Вряд ли кто стал бы покупать такой телевизор.

То же относится и к программам для микрокалькулятора. А ведь не редкость, когда программист считает свою работу законченной, если программа в принципе решает задачу, хотя и требует с десятков предварительных операций, которые к тому же никак не отражаются на индикаторе.

Опытные программисты, работающие на больших ЭВМ, знают, что нигде так не проявляется искусство программирования, как в сервисе. Максимально облегчить работу пользователя, застраховать программу от возможных ошибок — задача, которую просто необходимо решать при создании программ массового пользования.

Микровозможности микрокалькулятора, к сожалению, не позволяют в полной мере воспользоваться опытом, накопленным при программировании на больших ЭВМ, — текстовыми

подсказками и указаниями, которые дает сама программа пользователю, диагностикой ошибок и вообще полным описанием того, что должен делать пользователь для правильной эксплуатации программы.

Специфика микрокалькулятора рождает и специфические приемы. При этом, повторим еще раз, особая нагрузка падает на инструкцию. Но и сама программа должна быть написана так, чтобы, во-первых, уменьшить время ввода и вывода информации, во-вторых, снизить вероятность ошибок при вводе и интерпретации вывода, в-третьих, свести к минимуму число ручных операций при запуске программы.

Начнем с ввода.

Как известно, число, набранное на клавиатуре, автоматически заносится в стековый регистр X. Дальнейший путь числа определяется командами программы. В зависимости от алгоритма оно либо сразу направляется на обработку, либо записывается в один из адресуемых регистров и, когда возникает в том надобность, вызывается из него.

Иногда перед пуском программы исходные данные в режиме вычислений прямо записывают в адресуемые регистры, то есть вручную выполняют команды типа *a PN* (*a* — число, *N* — имя регистра). Такой процесс ввода (назовем его *адресным*) должен быть подробно описан в инструкции. Достоинством его является простая возможность проверки. Перед запуском программы можно с помощью команд *IPN* проконтролировать содержимое нужных регистров и при обнаружении ошибок исправить их. Еще один плюс адресного ввода заключается в том, что вводить числа в регистры можно в любом порядке, важно лишь проследить за тем, чтобы каждое число оказалось в своем регистре.

Однако эти достоинства меркнут по сравнению с основным недостатком адресного ввода. Он не технологичен, то есть доля ручного труда при таком вводе излишне велика, нажимать приходится массу клавиш. Отсюда слишком большое время ввода. Время это еще более увеличивается за счет большого количества проверок, так как вероятность спутать номер регистра при таком вводе весьма велика.

Другой способ — *безадресный ввод*. При этом данные по регистрам разбрасывает сама программа. Число просто набирается на клавиатуре, затем нажимается клавиша «С/П», а в программе стоят фрагменты типа *P1 С/П P2 С/П* и т. п. Доля ручного труда здесь существенно меньше. После набора числа нажимается каждый раз лишь одна клавиша «С/П». При этом надо знать лишь последовательность ввода данных, не заботясь о том, куда, в какие регистры они попадут. Однако

при таком способе ввода программа сильно распухает из-за бесчисленных «С/П». Впрочем, объем программы можно уменьшить, не уменьшая удобства пользователя. Если заканчивать каждый ввод не клавишей «С/П», а клавишей «ПП», то промежуточные остановки можно вычеркнуть из программы, и фрагмент ввода будет выглядеть так: П1 П2 и т. д. Действительно, каждое нажатие клавиши «ПП» вызывает выполнение очередной команды и останов, а это нам и надо.

Еще одна разновидность безадресного ввода — использование клавиши «↑» и накопление вводимых данных в стеке. Таким способом, правда, можно ввести не более четырех чисел подряд. Числа из стека можно постепенно перегнать в регистр Х и разослать потом по адресуемым регистрам, например так: П1 FO П2 FO П3. Такой ввод наиболее эффективен, если по алгоритму исходные данные не записываются (по крайней мере, не все записываются) в адресуемые регистры, а непосредственно поступают на обработку. При этом экономится и время ввода, и время обработки. Да и адресуемые регистры высвобождаются, а потому могут быть использованы более полно.

Какой же из описанных способов ввода предпочтительнее? Однозначного ответа на этот вопрос дать нельзя. Часто в одной программе разные порции исходных данных вводятся по-разному. Поэтому предложим вам лишь некоторые рекомендации.

Прямой, адресный ввод, очевидно, наиболее удобен для ввода данных, постоянных при различных значениях переменных, то есть коэффициентов в формулах, физических и других констант. К примеру, в электротехнических расчетах часто нужен заряд электрона ($1,602 \cdot 10^{-19}$ Кл), в расчетах по механике — ускорение свободного падения ($9,81 \text{ м/с}^2$). Эти величины не будут меняться при всех значениях переменных. Поэтому их целесообразно занести в режиме адресного ввода перед началом работы программы. Часто бывает необходимо в качестве коэффициентов использовать комбинации постоянных (к примеру, в задачах по электронике — величину kT , где k — постоянная Больцмана, T — температура по шкале Кельвина). Если при этом программная память и регистры заполнены, то здесь придется идти на некоторое уменьшение удобства пользователя и за этот счет высвободить ячейки памяти и регистры. Здесь адресный ввод может сочетаться с расчетами в режиме вычислений, то есть можно вычислять значения нужных коэффициентов в этом режиме и заносить их в соответствующие регистры перед началом работы программы.

Однако подчеркнем, что использование такого метода оправдано лишь в крайних случаях. Если есть возможность, то и вычисления коэффициентов лучше записывать в программу, чтобы избавлять пользователя от этой скучной работы.

Если количество вводимых чисел не превышает четырех, то, наверное, предпочтительнее ввод в регистры стека: число «↑», число «↑»,

При вводе примерно десятка данных, тем более если по программе их нужно записывать в адресуемые регистры, можно рекомендовать ввод такого типа: число «ПП», число «ПП». Более интересный вариант подобного ввода — использование для записи в регистры команд косвенной адресации. Это не только уменьшает длину программы, но и делает ее более универсальной — можно вводить в одну и ту же программу произвольное, меняющееся от одного расчета к другому количество чисел, конечно, не превышающее количества адресуемых регистров. Соответствующий пример будет рассмотрен ниже.

Встречаются такие программы, в которых по самой сути алгоритма приходится вводить столь много данных, что их число превышает количество регистров. К таким программам сводятся задачи статистической обработки данных, аппроксимации их различными функциями (так называемые *программы сглаживания*) и ряд других. Все такого рода программы распадутся на две части: первая — собственно ввод и накопление промежуточных результатов, вторая — получение итоговых данных.

Если при этом надо вводить числа по одному, как в задачах статистической обработки, то ввод организуют по следующей схеме: число «С/П»; если же данные вводятся парами, как в задачах сглаживания, то чаще всего по такой схеме: число «↑», число «С/П».

Однако основная проблема при программировании подобных задач иная. Как переходить от одной части к другой? Рассмотрим несколько способов такого перехода на примере задачи простейшей статистической обработки — поиска среднего арифметического произвольного набора чисел.

Алгоритм решения задачи простой: вводим числа, накапливаем их сумму и заодно запоминаем их количество. Когда все числа введены, делим полученную сумму на количество введенных чисел. Такая структура вообще типична для различных статистических задач.

Один из вариантов программы приведен в табл. 10.

Легко просматриваются две части программы: первая — команды 00—07, вторая — 08—11. Видно также, что в програм-

Т а б л и ц а 10

Адрес	Команда	Код	Адрес	Команда	Код
00	ИП1	61	06	БП	51
01	+	10	07	00	00
02	П1	41	08	ИП1	61
03	КИП4	Г4	09	ИП4	64
04	Сх	0Г	10	÷	13
05	С/П	50	11	С/П	50

ме нет перехода на вторую часть. Как же работать с этой программой?

Набираем число на клавиатуре, нажимаем «С/П». На индикаторе — нуль. Вводим новое число, снова «С/П», на индикаторе опять нуль и т. д. до тех пор, пока все числа не исчерпываются. Теперь переходим на вторую часть. Самое простое — это в режиме вычислений ввести БП 08, изменив тем самым содержимое программного указателя, и запустить программу с этого адреса клавишей «С/П». Самое простое, однако, не всегда самое лучшее. И вообще, хотя программа получилась довольно краткая, число ее недостатков, наверное, обратно пропорционально длине.

Во-первых, если считать по этой программе несколько раз, то придется каждый раз очищать не только программный указатель («В/О»), но и регистры R1 и R4, «засоренные» во время предыдущего расчета.

Во-вторых, слишком уж безлика реакция программы на очередной ввод. Каждый раз высвечивается нуль. При небольшой модернизации можно заставить программу выводить, скажем, номер последнего введенного числа, чтобы пользователю было удобнее контролировать ввод.

И, в-третьих, очень неудобно «вручную» передавать управление на вторую часть программы. Нажимая три клавиши «БП», «0», «8», легко можно ошибиться, тем более, что действия наши никак не отражаются на индикаторе.

К счастью, недостатки эти легко исправимы. Начнем с последнего. Количество нажимаемых клавиш можно уменьшить, не трогая программу. Вместо «БП», «0», «8» достаточно нажать «ШГ». Эта команда увеличит на единицу содержимое адресного указателя и таким образом передаст управление второй части программы. Такой способ достаточно эффективен. Но, к сожалению, нажатие клавиши «ШГ» никак не отражается на индикаторе, и если вы на минуту отвлеклись, то заметить, нажата клавиша или нет, невозможно. То есть

узнать это вообще-то можно, только надо дополнительно набрать ФПРГ, перейти в режим программирования, проверить номер команды на индикаторе и затем перейти снова в режим вычислений (FABT). Но слишком уж это долго.

Лучше всего было бы заставить саму программу управлять своей работой. Сделать это можно, например, так. В качестве последнего числа вводить какой-нибудь признак, по которому программа «поймет», что ввод окончен и нужно приступить к расчетам по второй части. Предположим, нам известно, что все числа из анализируемого набора положительные. Тогда признаком окончания ввода будет любое отрицательное число. В других случаях в качестве признака можно выбрать число какого-нибудь «специального» типа, например 0 или $1 \cdot 10^{98}$. Остановимся пока на первом варианте.

Итак, вводим в программу сравнение вводимого числа с нулем. Если число больше нуля, продолжаем ввод, если нет, передаем управление на вторую часть. Запрограммируем также вывод на экран количества введенных чисел и заставим нашу программу перед началом каждого сеанса ввода обнулять программный указатель и регистры R1 и R4. Новая программа будет выглядеть так, как показано в табл. 11.

Она длиннее прежней на 9 команд, но зато насколько удобнее! Для работы с ней требуется лишь при первом сеансе обнулить программный указатель («В/О»), а затем вводить числа, заканчивая их последовательность вводом любого отрицательного числа. Все остальное она делает сама. Команды по адресам 00 и 04 носят тоже чисто сервисный характер. Вместо них можно было бы ограничиться одной «С/П» по адресу 04. Однако пришлось бы сначала запускать программу клавишей «С/П» и уже после останова вводить числа. Всего одна лишняя клавиша. Но почему бы, если есть место в памяти, не сократить ручные операции хотя бы на одну?

Таблица 11

Адрес	Команда	Код	Адрес	Команда	Код
00	↑	0E	11	ИП4	64
01	Cx	0F	12	С/П	50
02	П1	41	13	БП	51
03	П4	44	14	05	05
04	XY	14	15	ИП1	61
05	$Fx \geq 0$	59	16	ИП4	64
06	15	15	17	÷	13
07	ИП1	61	18	С/П	50
08	+	10	19	БП	51
09	П1	41	20	00	00
10	КИП4	Г4			

Проблема сервиса при вводе не исчерпывается минимизацией нажимаемых клавиш и избавлением пользователя от необходимости запомнить, в какие регистры что засылать. Кроме этого, хорошо бы иметь надежные средства для того, чтобы контролировать и при необходимости исправлять ошибочно введенную информацию. Однако эти вопросы целесообразно рассматривать в приложении к конкретным программам и каждый раз разрешать специальным образом.

Теперь перейдем к проблеме вывода.

Вывод должен быть организован столь же удобно, как и ввод, и в дополнение к этому нагляден. Первая проблема решается средствами, подобными используемым для ввода, только, если можно так сказать, со знаком минус. Иначе говоря, если результаты накоплены в адресуемых регистрах, то извлекать их можно с помощью команд ИПН, обратных по действию командам ПН, и делать это надо, естественно, после окончания работы программы.

Здесь применимы приемы, рассмотренные выше. К примеру, после команды останова, завершающей работу программы, можно поместить фрагмент типа ИПА С/П ИПВ С/П и т. д. и использовать для вызова результатов команду С/П. Можно считывать данные и с помощью команды ПП. Это сокращает программу, так как фрагмент, аналогичный предыдущему, записывается так: ИПА ИПВ ... Если результатов расчета не более четырех, то для их хранения целесообразно использовать стек. Извлечение результатов оттуда легко реализуется клавишей «FO»: содержимое регистров стека постепенно подводится к «окошку» (регистру X) и может быть прочитано.

Однако эти приемы оказываются неудовлетворительными, если при работе ветвистой программы нужно не только прочесть, но и правильно интерпретировать результаты. Классический пример подобной ситуации — решение квадратного уравнения, когда нужно не только прочесть полученные числа, но и узнать, не было ли уравнение вырожденным, а если не было, то какого типа получились корни — действительные или комплексные. Подобные же ситуации встречаются при решении многих технических задач. Да и в игровых задачах вывод сообщений играет не последнюю роль.

С некоторыми приемами формирования диагностических сообщений мы познакомились в разделе 5. Напомним, что самый простой путь — занумеровать сообщения и выводить их номера — 1, 2 и т. д. При этом, однако, можно спутать номер или шифр сообщения с самим результатом счета. О том, как сделать шифр непохожим на число, также говорилось в упо-

мянутом разделе. Были описаны способы получения сообщений типа E00, E01 и, вообще, E mn , где m и n — любые десятичные цифры. Там же было описано получение «шифра» Г.

Подобный перечень можно расширить. Например, выдавать сообщения типа Г0, Г00. Для этого надо нажать клавиши «Сх», «↑», «÷» и после появления на индикаторе ЕГГОГ продолжить: «ВП», «ВП», затем либо сразу нажать клавишу «↑», тогда получим Г, либо ввести «порядок» и нажать «↑». Если этот порядок меньше 8, то получим на экране после символов Г нули в количестве, равном порядку. Если порядок больше 8, то на экране будет в левом углу Г, а в правом — сам порядок.

Хотя вывод сообщений с помощью указанного приема довольно нагляден, есть у него два недостатка, и довольно существенных: во-первых, он требует большого числа ручных операций, а во-вторых, постоянно занимает адресуемые регистры. Можно рекомендовать другой метод, не столь наглядный, но зато очень простой: выводить числа в какой-либо специальной форме, например не 1, а 10000 или 100001. Этот прием используется довольно часто, так как подобные сообщения можно и сохранять в регистрах, и формировать в программе. Вообще проблема выдачи сообщений наиболее остра для задач, работающих в режиме диалога. Из числа «микрокалькуляторных» к таким задачам относятся прежде всего игровые программы. Пример подобной программы будет дан в одном из следующих разделов.

Познакомившись с некоторыми приемами сервиса, посмотрим теперь пример их практической реализации.

Задача, решение которой мы вам предложим, может возникнуть в самых разнообразных ситуациях. Суть ее в том, что некоторые данные последовательно вводятся в микрокалькулятор и затем обрабатываются, причем процесс обработки для нас не имеет решающего значения. Это может быть статистическая обработка, упорядочивание введенных чисел, определение процентного вклада каждой величины в общую сумму и т. д.

Можно представить себе, например, такие случаи. Коллектив авторов пишет книгу. Каждый из них написал разное число страниц, а для бухгалтерии требуется указать долю участия в работе всех авторов, выраженную в процентах. Или в правление колхоза поступают сведения о площадях, засеянных разными бригадами. Надо выяснить вклад каждой бригады в посевную кампанию.

Если перед вами лежит микрокалькулятор, то для решения подобной задачи бумага вам не понадобится. Данные можно

вводить непосредственно в ПМК и после окончания ввода получать результаты. Поможет в этом программа, разработанная московским инженером А. Б. Бойко.

Так как программа предназначена для человека, занятого еще и другой работой, то ввод должен быть максимально упрощен. Организуем его так, чтобы достаточно было набрать число на клавиатуре и нажать после этого одну клавишу «С/П». Для удобства пользователя перед каждым вводом будем высвечивать номер очередного числа (начнем нумерацию с нуля). Этот же номер будет определять регистр, в который записывается число. Признаком окончания ввода будем считать набор числа π , благо на нашем микрокалькуляторе есть специальная клавиша для этой цели. Вероятность того, что в результате измерения мы получим число, равное π , да еще с точностью до восьмого знака, пренебрежимо мала.

Возможности нашего калькулятора ограничены числом адресуемых регистров. Напомним их всего четырнадцать. Два из них в программе используются для служебных целей, следовательно, ввести в нашу программу можно не более двенадцати чисел. А что будет, если пользователь по ошибке введет тринадцатое? Ничего страшного. В программе предусмотрена и такая ситуация. В ответ на попытку ввести лишнее число программа выдаст на индикатор сообщение ЕГГОГ, после чего достаточно нажать клавишу «С/П» для получения выходных данных.

Признаком завершения работы будет останов микрокалькулятора и вывод на индикатор числа π .

К сожалению, так как адресуемые регистры заняты, мы не можем воспользоваться методами получения шифрованных сообщений, описанными в разделе 5. Однако и используемые нами приемы (вывод сообщения ЕГГОГ или восьми знаков числа π) довольно наглядны и понимаются однозначно.

Программа приведена на табл. 12. По функциональному признаку ее можно разделить на такие части.

00—02. Очистка служебных регистров: счетчика (регистр Д) и сумматора (регистр С; в этом регистре формируется сумма введенных чисел).

03—04. Вывод номера вводимого числа (начиная с 0) и подготовка ввода самого числа.

05—09. Сравнение введенного числа с π и выбор продолжения работы: если число не равно π , то произойдет запись его в очередной регистр и получение текущего значения суммы (адреса 10—14), а если равно — то переход на вторую часть программы (адрес 25) и обработка, вычисление процентных долей.

Таблица 12

Адрес	Команда	Код	Адрес	Команда	Код
00	Cx	0Г	22	F√	21
01	ПД	4Г	23	БП	51
02	ПС	4С	24	03	03
03	ИПД	6Г	25	ИПС	6С
04	С/П	50	26	2	02
05	↑	0Е	27	F10 ^x	15
06	Fπ	20	28	÷	13
07	—	11	29	ПС	4С
08	Fx ≠ 0	57	30	ИПД	6Г
09	25	25	31	1	01
10	XУ	14	32	—	11
11	КПД	LG	33	ПД	4Г
12	ИПС	6С	34	КИПД	ГГ
13	+	10	35	ИПС	6С
14	ПС	4С	36	÷	13
15	1	01	37	КПД	LG
16	1	01	38	ИПД	6Г
17	ИПД	6Г	39	Fx = 0	5Е
18	1	01	40	30	30
19	+	10	41	Fπ	20
20	ПД	4Г	42	С/П	50
21	—	11			

15–16. Ввод числа 11 в стек. Это число нужно для дополнительной проверки: не превышает ли номер очередного регистра 11, то есть не сделана ли попытка ввести число в регистр С, который уже занят для хранения суммы. Если превышает, то по команде 21 будет получено отрицательное число и попытка извлечь из него квадратный корень (команда 22) даст сообщение ЕГГОГ – сигнал того, что больше чисел вводить нельзя. Клавиша «С/П», нажатая в такой ситуации, вызовет пропуск одной команды и таким образом передаст управление по адресу 25, на вторую часть программы.

Советуем вам обратить внимание на этот прием. Полезное использование ошибочной ситуации при решении задач на микрокалькуляторе – явление довольно частое. Благодаря этому можно получать информацию и продолжать работу программы.

25–40. Эти команды образуют цикл для вычисления процентных долей введенных чисел. Как только содержимое счетчика регистра Д станет равным нулю, то есть все числа будут исчерпаны, цикл заканчивается, и на индикатор выводится признак конца работы – число π (команда 41).

Инструкция для работы с программой.

1. Ввести программу.

2. Перейти в режим вычислений (FABT).
3. Очистить программный указатель (B/O).
4. Запустить программу (C/П).
5. Ввод: если на индикаторе 0, то a_0 С/П, если на индикаторе 1, то a_1 С/П, ... Окончание ввода: Fл С/П.

Если на индикаторе 11, то ввод любого числа приводит к появлению сообщения ЕГГОГ. В этом случае нажать «C/П».

6. Вывод: результаты (вычисленные в процентах) находятся в последовательно расположенных регистрах данных. Для чтения результатов последовательно нажимать: ИП0, ИП1 и т. д.

7. Продолжение работы с новой порцией данных: перейти к п. 3.

Контрольный пример.

Ввод: 5 С/П 20 С/П 25 С/П Fл С/П. В регистрах: R0 = 10, R1 = 40, R2 = 50.

Как видите, программа получилась довольно большой. Причем большинство команд носит ярко выраженный «сервисный» характер. Но ведь и цель наша заключалась в том, чтобы создать программу, облегчающую рутинную работу пользователя.

Вообще, не следует забывать, что программа должна быть прежде всего удобной пользователю, поэтому всякие сокращения за счет удобств крайне нежелательны, прибегать к этому стоит лишь в исключительных случаях, когда иначе разместить программу в памяти просто невозможно. Во всех остальных случаях лучше, если программа будет длиннее, зато удобнее для работы.

Это утверждение, пожалуй, и будет основным выводом данного раздела.

11. КАЛЬКУЛЯТОР – ПОМОЩНИК В РАБОТЕ

В одном из первых разделов книги говорилось, что существуют задачи, для решения которых среди всего многообразия ЭВМ наилучшим образом подходят программируемые микрокалькуляторы.

Речь шла о задачах, для решения которых достаточно «микровозможностей» микрокалькулятора и на первый план выходят его достоинства: постоянная доступность и возможность использования его на любом рабочем месте – в лаборатории, на стройке, в цехе.

Один из примеров, когда калькулятор используется для решения практической, производственной задачи, мы сейчас

и разберем. И хотя речь будет идти о конкретной узко-специальной задаче, думаем, что подробное описание процесса составления программы окажется полезным специалистам самых разных областей. Надеемся, что читатели после этого сами увидят в сфере своей деятельности задачи, которые можно свести к «микрокалькуляторным» и для решения которых можно с успехом применить знания, полученные при прочтении книги.

.. Чтобы сталь было легче обрабатывать, а изделия из стали 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 соотношении с другими. Заданные пропорции между компонентами достигаются 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 добавки? Точный ответ на этот вопрос должен дать горновой мастер.

Опытные мастера славятся тем, что умеют решать такие задачи, не прибегая к расчетам, одним им ведомыми умозрительными приемами. А если плавку ведет не очень опытный мастер? Ему без расчетов не обойтись. Да и опытному мастеру не помешало бы «алгеброй гармонию поверить». Но кого просить о помощи в расчетах? Обратиться в заводской вычислительный центр и, терпеливо дождавшись своей очереди, получить точнейший ответ? Но это слишком долго. А не призвать ли на помощь микрокалькулятор? Хоть он считает и медленнее ЭВМ из вычислительного центра, зато всегда под рукой.

Итак, попробуем. 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 A_1, A_2, A_3, \dots содержание каждого из присутствующих в исходном сплаве легирующих элементов, выраженное в процентах. Символами B_1, B_2, B_3, \dots 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 . Масса добавки каждого ферросплава, которая и подлежит определению, пусть выражается символами M_1, M_2, M_3, \dots ; общая масса металла при отборе пробы — M_0 , а после

добавления ферросплавов — M_k ,

$$M_k = M_0 + \sum M_i$$

(его мы подсчитаем простым суммированием).

Попробуем составить систему уравнений для отыскания масс M_1 , M_2 и последующих. Общее число уравнений обозначим буквой n .

Начнем решать задачу «с конца». Предположим, что масса всех добавок в полученном сплаве определена нами верно. Масса металла, находящегося в плавильной печи, $M_k = M_0 + \sum M_i$. Поскольку состав сплава соответствует норме, то процентное содержание в нем каждого из легирующих элементов выражается числами B_1, B_2 и т. д. Общая масса первого элемента в полученном сплаве равна $(M_0 + \sum M_i) B_1$. С другой стороны, первый легирующий элемент содержался как в исходном сплаве (здесь масса его была равна $M_0 A_1$), так и в добавке первого ферросплава (здесь масса легирующего элемента $M_1 C_1$).

Сведем все эти величины в равенство

$$M_0 A_1 + M_1 C_1 = (M_0 + \sum M_i) B_1.$$

Аналогично составим равенства, относящиеся ко второму, третьему и дальнейшим легирующим элементам:

$$M_0 A_2 + M_2 C_2 = (M_0 + \sum M_i) B_2,$$

$$M_0 A_3 + M_3 C_3 = (M_0 + \sum M_i) B_3,$$

• • • • •

$$M_0 A_n + M_n C_n = (M_0 + \sum M_i) B_n.$$

Так, равенство за равенством, сложилась система из n линейных алгебраических уравнений с n неизвестными M_1, M_2, \dots, M_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 уравнений — двенадцать, систему из четырех — двадцать и т. д.

Но у нашего калькулятора всего четырнадцать регистров!

Значит, придется ограничиться всего тремя неизвестными? И составлять программу, которая позволит горновому мастеру уточнять соотношение компонентов 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:

$$M_i = \left(\frac{B_i}{C_i} K - \frac{A_i}{C_i} \right) M_0, \quad K = \frac{1 - \sum (A_i/C_i)}{1 - \sum (B_i/C_i)}.$$

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 M_0 , которую тоже нужно где-то хранить? Правда, можно использовать в качестве «хранилищ» и регистры стека. Навыки программирования, приобретенные нами по ходу предыдущих занятий, 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 , понадобятся 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 единиц, плюс последующее деление полученных разностей друг на друга и засылка полученного коэффициента K в приданный ему регистр памяти. Да еще останов для засылки величины M_0 . (Ее лучше засылать не в начале работы, а после получения промежуточных сумм, когда часть регистров уже освободится.)

Итого 37 команд. А сколько команд уйдет на вычисление величин $M_i = (B_i K / C_i - A_i / C_i) M_0$? Постараемся придать выражающим их формулам наиболее экономный с вычислительной точки зрения вид $M_i = (B_i K - A_i) M_0 / C_i$. Расчет по такой формуле, как нетрудно прикинуть, разворачивается в десять команд. И если всего неизвестных величин четыре, то на их получение по приведенным формулам требуется 40 команд. Прибавляя их к ранее полученным 37, получаем 77. Это как

минимум. А удобства работы с программой? Она тоже требует программной памяти. А в нашем распоряжении всего 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, освоенным нами в предыдущих разделах.

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 , C_i , а только два — A_i/C_i и B_i/C_i ? Тогда четырнадцать регистров, имеющих 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 M_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 M_i . По окончании одного понадобится обновить содержимое счетчика циклов, а для этого надо где-то записать 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 предназначена 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, горновой 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 (табл. 13).

Таблица 13

№	Элемент	A_i (%)	B_i (%)	C_i (%)	M_i (кг)
1	Хром	14,5	23	70	4812
2	Никель	32,3	27	99	445
3	Марганец	0,4	0,6	96	86
4	Молибден	3,1	2,75	60	148
5	Медь	3,0	2,9	100	155
6	Титан	0	1,2	70	500

Марка стали — 06Х23Н28М3Д3Т; начальная масса — 23000 кг, конечная масса — 29146 кг.

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 требуемое содержание (B_i), в пятом — их процентное содержание в соответствующих ферросплавах (C_i), в шестом — искомые массы 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, относящихся к определенному элементу. Чтобы свести к минимуму вероятность ошибки, очевидно, следует вводить эту информацию в калькулятор следующим образом: ввести сначала в стек одно за другим три числа — A_1 , B_1 , C_1 — из первой строки таблицы 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 «рассылочного» цикла.

Теперь подумаем о порядке расстановки дробей A_i/C_i и B_i/C_i . Здесь возможны 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 видов 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 KPN , где N — номер регистра-счетчика циклов. По находящемуся там в каждый очередной момент числу и будет определяться адрес засылки каждой очередной дроби A_i/C_i или B_i/C_i . Но если дроби разных видов станут рассылаться в разные участки массива регистров, то нам потребуются уже не один, а два счетчика. На такое расточительство мы пойти уже не можем. Итак, из соображений экономии расставляем дроби в чередующемся порядке, парами: A_1/C_1 и рядом 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 FLN . Но в этих командах в роли счетчика циклов могут фигурировать лишь регистры с номерами N от нулевого до третьего. С каждым выполнением команды FLN число, находящееся в регистре N , будет уменьшаться на единицу. При указанных номерах N команда KPN 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 N ? 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 KPN), засылка дроби B_i/C_i (такая же команда), возврат к началу цикла (команда FLN).

К счастью, среди команд косвенной засылки есть такая: $KP\uparrow$ (это еще один секрет микрокалькулятора, в инструкции команда $KP\uparrow$ не описана). Она посылает содержимое регистра X в регистр, номер которого равен числу, находящемуся в нулевом регистре, — так же, как и команда $KP0$, но, в отличие от нее, не меняет содержимого нулевого регистра. Аналогично действует и команда $KIP\uparrow$. Это счастливое обстоятельство наверняка поможет нам.

Кстати говоря, оно однозначно диктует нам номер регистра-счетчика: нулевой. Ну, а номер регистра, куда будет заслана

первая из наших дробей? Как его назначить? Это совсем просто. Номер будет равен удвоенному числу легирующих элементов. А само число $2n$ (или $n + n$) должно быть заслано в R0. Тогда если первая из рассылочных команд будет иметь вид КП↑, первая из дробей попадет точно в 12-й регистр, то есть в RC. 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 программы. 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 $B_i K/C_i - A_i/C_i$. Затем ввод величины M_0 . И еще одна циклическая часть: вычисление искомых величин M_i и их суммирование с M_0 , что даст итоговую массу сплава M_k ; вывод полученного значения M_k на индикатор. Наконец, заключительная часть, также циклическая: вывод на индикатор величин M_i .

Теперь приступим к программе.

Чтобы лучше представлять себе ее работу, используем предложенный в разделе 5 способ ее записи, при котором после команды станем приводить и содержимое стековых регистров.

Начнем с того, что введем число n и, удвоив его, запишем результат в R0:

ввод n				
00	↑	0E	n	n
01	+	10	$2n$	
02	ПД	4Г	$2n$	
03	ПО	40	$2n$	
04	С/П	50	$2n$	

После останова вводим в калькулятор число A_1 , нажимаем клавишу «↑», вводим B_1 , нажимаем еще раз клавишу «↑»

и вводим C_1 . В стеке эти три числа расположатся в обратной последовательности. Все промежуточные результаты мы постараемся тоже размещать в регистрах стека, полнее использовать его возможности. Впрочем, иначе и быть не может при нашей острой нехватке адресуемых регистров.

Числа B_i и C_i располагаются в стеке так, как нужно для вычисления дроби B_i/C_i . Получим ее, выполнив операцию деления, и отправим по адресу, который «известен» нулевому регистру, выполнив команду $KP\uparrow$. Величина C_i выпала из нашего поля зрения не насовсем. Она находится в регистре предыдущего результата $RX1$. Подняв ее в RX и немного «погоняв» числа по стеку, получим в свое распоряжение нужную пару: A_i в RY и C_i в RX . Снова выполняем деление, результат которого засылаем на хранение уже с помощью команды $KP0$. Здесь уменьшение адреса нам на руку. Дробь A_i/C_i попадает как раз в регистр, соседний с тем, где находится B_i/C_i . Команда цикла $FL0$ уменьшает содержимое счетчика еще на единицу и передает управление к началу ввода.

Структура цикла ясна полностью. Напишем его от начала до конца, повторив для наглядности команду по адресу 04 (прочерк означает, что содержимое регистра не изменилось):

04	C/П	50	C_i	B_i	A_i
05	÷	13	B_i/C_i	A_i	
06	$KP\uparrow$	LE	—	—	
07	FBx	0	C_i	B_i/C_i	A_i
08	XY	14	B_i/C_i	C_i	A_i
09	FO	25	C_i	A_i	
10	÷	13	A_i/C_i		
11	$KP0$	L0	—		
12	FL0	5Г	—		
13	04	04	—		

Одно прохождение цикла за другим — и вот уже последняя дробь A_n/C_n занесена в свой регистр, содержимое регистра 0 стало равным единице. Команда $FL0$ совершает выход из цикла.

Пора приступить к вычислению K , а затем — величин M_i и M_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. Из предыдущих разделов мы уже знаем, что такие алгоритмы — одни из самых «быстрых». Арифметические

операции, с помощью которых образованы наши формулы, дают погрешность от силы в последнем из восьми знаков, появляющихся на индикаторе вычислительной машины. А данные лабораторного анализа содержат лишь по три значащих 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 клавишами «1» и «↑». А еще раньше надо обновить содержимое счетчика циклов:

14	ИПД	6Г	2n
15	ПО	40	2n
16	1	01	1 2n
17	↑	0E	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↑	ГЕ	B_i/C_i	1	1	2n
19	—	11	$1 - B_i/C_i$	1	2n	
20	ХУ	14	1	$1 - B_i/C_i$	2n	
21	КИПО	ГО	A_i/C_i	1	$1 - B_i/C_i$	2n
22	—	11	$1 - A_i/C_i$	$1 - B_i/C_i$	2n	

Перед возвратом к началу цикла расположим «полуфабрикаты» наших разностей в стеке так, как они располагались там перед первым вычитанием, — поменяем их местами. Завершив цикл командой возврата к его началу:

23	ХУ	14	$1 - B_i/C_i$	$1 - A_i/C_i$	2n
24	FL0	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:

$$26 \quad \div \quad 13 \quad \frac{1 - \sum (A_i/C_i)}{1 - \sum (B_i/C_i)}$$

Настало время вычислять выражения $(B_i K/C_i - A_i/C_i)$. Подсчеты этих разностей будут носить опять-таки циклический

характер. Обновим содержимое счетчика циклов:

```
27 ИПД 6Г 2n K
28 ПО 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 $B_1K/C_1 - A_1/C_1$ коэффициент K исчезнет после его умножения на B_1/C_1 . Чтобы не допустить этого, поместим копию K еще в одном регистре стека. Сделаем так:

```
29 FO 25 K
30 ↑ 0E K K
```

Далее все пойдет привычным для нас путем использования команд косвенного вызова и засылки:

```
31 КИП↑ GE  $B_i/C_i$  K K
32 × 12  $B_iK/C_i$  K
33 КИПО Г0  $A_i/C_i$   $B_iK/C_i$  K
34 — 11  $B_iK/C_i - A_i/C_i$  K
35 КП↑ LE — —
36 FLO 5Г — —
37 29 29 — —
```

Как видим, после первого прохождения цикла величина $B_iK/C_i - A_i/C_i$ очутилась в 11-ом регистре, то есть в регистре В. По мере дальнейших повторений цикла аналогичные разности окажутся в регистрах с нечетными номерами: девятым, седьмым 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 FLO 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. Выйдя из него, восстановим содержимое счетчика циклов: он еще потребуется нам для подсчета масс искомых добавок M_i и общей массы плавки M_K :

```
38 ИПД 6Г 2n
39 ПО 40 2n
```

Чтобы такой подсчет был выполнен, горновой мастер должен ввести в калькулятор величину M_0 . Для этого выполнение программы нужно прервать. И хорошо бы, чтобы на экране остановившегося калькулятора горело какое-то услов-

ное число, которое напоминало бы мастеру, что он должен сделать. Например, нуль. Зашлем его в регистр X, а потом скомандуем «стоп» и введем M_0 :

40	0	00	0	$2n$
41	C/П	50	0	$2n$
	ввод	M_0	0	$2n$

Перешлем величину M_0 из регистра X в один из освобожденных адресуемых регистров, например в 12-й, то есть в регистр C:

42 ПС 4Г M_0

Осталось сделать немного, 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 M_1 надо вызвать разность $(B_1K/C_1 - A_1/C_1)$ из 11-го регистра. Ясно, что тут надо употребить команду КИП0, уменьшающую содержимое счетчика. Затем вызванную разность надо умножить на величину M_0 . Оба сомножителя стоят так, как это требуется для умножения — в регистрах X и Y. Но если подсчитать это произведение сейчас, величина M_0 исчезнет, а нам она еще понадобится для подсчета итоговой массы плавки $M_k = M_0 + \sum M_i$. Лучше поступим так: вызовем M_0 из регистра C. Нужные нам сомножители снова окажутся в регистрах X и Y, хотя и в ином порядке. Выполнив умножение $(B_1K/C_1 - A_1/C_1) \cdot M_0$, перешлем результат в тот же 11-й регистр. Тут уже потребуется команда КП↑. После ее выполнения в регистре Y (проследите по выписанному ниже фрагменту программы движение информации по стеку) окажется величина M_0 , а в регистре X будет находиться M_1 . Такое расположение весьма кстати: сложив обе величины, мы начнем подсчет итоговой массы плавки M_k , увеличившейся за счет добавок. После сложения можно возвращаться к началу цикла:

43	КИП0	Г0	$B_iK/C_i - A_i/C_i$	M_0
44	ИПС	6Г	M_0	$B_iK/C_i - A_i/C_i \quad M_0$
45	×	12	$(B_iK/C_i - A_i/C_i) M_0$	M_0
46	КП↑	LE	—	—
47	+	10	$M_0 + M_i$	
48	FL0	5Г	—	
49	43	43	—	

По выходе из цикла мы имеем в регистре X величину M_k — итоговую массу плавки. Можно было бы тут и остановить программу, чтобы эта величина высветилась на индика-

торе. Но ведь нам еще предстоит вывести на индикатор и массы добавок M_i . Для этого придется организовать еще один цикл. Поэтому отсылаем величину M_k в один из освободившихся четных регистров (скажем, в 10-й, регистр А), восстанавливаем содержимое счетчика циклов, а уже затем командой ФО возвращаем величину M_k в регистр X. Вот теперь, остановив программу командой С/П, считываем значение M_k с индикатора. Далее надо выводить значения M_1 , M_2 и последующие. Дело это несложное, выполнимое всего тремя командами (55–57):

50	ПА	4	—	M_k
51	ИПД	6Г	$2n$	M_k
52	ПО	40	$2n$	M_k
53	ФО	25	M_k	
54	С/П	50	M_k	
55	КИПО	Г0	M_i	
56	FL0	5Г	M_i	
57		54	M_i	
58	С/П	50	M_i	

Остается раз за разом нажимать клавишу «С/П» и после каждого очередного останова считывать с индикатора M_1 , M_2 и следующие искомые величины. Когда будет считана последняя, произойдет выход из цикла, и тут уже можно будет поставить последнюю точку: команду С/П (по адресу 58).

И все-таки повременим с окончанием работы над программой. Вспомним о тех непростых условиях, в которых ей суждено работать. Представьте себе, что горновой мастер сбился, считывая с индикатора величины M_i , и, терпеливо дождавшись завершения цикла, хочет заново вывести на индикатор какую-то из них, скажем, M_3 . Где она размещается? Нетрудно составить таблицу соответствий и выяснить по ней, что M_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 M_i совпадали с номерами регистров, их содержащих, то есть M_1 — в первом регистре, M_2 — во втором и так далее? А значение итоговой, общей массы 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 M_k . Перешлем ее в нулевой регистр. Теперь организуем пересылку величин M_i на новые места: M_1 направим из 11-го регистра

в 1-й, M_2 — из 9-го во 2-й, ..., M_6 — из 1-го в 6-й. Впрочем, вести пересылку в том порядке, в каком она здесь описана, нельзя: скажем, если сначала переслать содержимое 11-го регистра в 1-й, где до этого хранилась величина M_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	ПО	40	M_k
51	ИП1	61	M_6
52	П6	46	M_6
53	ИП5	65	M_4
54	П4	44	M_4
55	ИП3	63	M_5
56	П5	45	M_5
57	ИП7	67	M_3
58	П3	43	M_3
59	ИП9	69	M_2
60	П2	42	M_2
61	ИПВ	6L	M_1
62	П1	41	M_1
63	ИПО	60	M_k
64	С/П	50	

На индикаторе остановившегося калькулятора — итоговая, общая масса плавки. Ее надо списать. Теперь надо вывести на индикатор и считать значения M_i . Проще всего приписать для этого к программе последовательность команд ИПН и пройти их одну за другой, нажимая клавишу «ПП». На индикаторе по порядку станут возникать нужные числа:

65	ИП1	61	M_1
66	ИП2	62	M_2
67	ИП3	63	M_3
68	ИП4	64	M_4
69	ИП5	65	M_5
70	ИП6	66	M_6

Если горновому мастеру потребуется вновь осведомиться об одной из полученных величин, например M_3 , ему достаточно будет нажать две клавиши (в нашем примере «ИП» и «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 которой заявляли, еще только приступая к ее разработке. Но вспомним: это говорилось по поводу вычислительной ее части — иначе она просто не уместилась бы в памяти микрокалькулятора. Мы добились своего: вычисления заканчиваются на 49-м адресе. Еще 48 ячеек свободны. Почему бы не истратить их на то, чтобы сделать программу как можно более удобной для работы? Лозунг краткости теряет смысл после того, как искомые результаты получены. Какой прок был бы теперь от сэкономленных адресов, если такая экономия создавала бы затруднения пользователю программы?

Конечно, говоря об этом, не следует оставлять в стороне соображения быстроты и точности. Что ж, примем во внимание и их. Нескільки лишних операций, выполненных однократно, — это от силы несколько секунд, о которых не стоит и говорить. А о точности не приходится заботиться, когда речь идет о пересылках информации, ничуть ее не искажающих.

Вот сейчас можно считать нашу работу по составлению программы законченной. Осталось привести инструкцию. Она может выглядеть так.

Инструкция к программе.

1. Ввести программу.
2. Перейти в режим вычислений (FABT).
3. Очистить программный указатель (B/O).
4. Ввести число легирующих элементов: n С/П.
5. Ввести одно за другим, перемежая нажатием клавиши «↑», процентное содержание первого легирующего элемента в исходном сплаве A_1 , содержание первого легирующего элемента в сплаве, который должен быть получен, то есть B_1 , содержание первого легирующего элемента в первом ферросплаве C_1 . Запустить программу на счет (С/П).

6. Продолжить ввод чисел A_i , B_i , C_i , как описано в п. 5 ($i = 2, \dots, n$); по окончании ввода перейти к п. 7.

7. После останова, во время которого на индикаторе высвечивается число 0, ввести исходную массу плавки в килограммах: M_0 С/П.

8. После останова считать с индикатора итоговый вес плавки M_k . Последовательно нажать n раз клавишу «ПП», и каждый раз считывать с индикатора вес добавки очередного ферросплава M_i ($i = 1, \dots, n$), выраженный в килограммах.

9. Если требуется вновь вывести на индикатор массу добавки i -того ферросплава M_i , нажать клавиши «ИП», «i» и считать с индикатора появившееся на нем число.

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.

Проследим окончательный вариант программы от первого адреса до последнего.

Видно, что краткость программы, точнее ее вычислительной части, достигнута в первую очередь благодаря организации циклов, причем с использованием команд косвенной 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!»

Подчеркнем, что задача, которую мы рассматривали, взята прямо из жизни. Ее сформулировал зам. директора запорожского завода «Днепроспецсталь» Ю. Г. Волович, алгоритм решения предложил ленинградский профессор Г. И. Натансон.

Думаем, что некоторые выводы, вытекающие из работы над этой программой, могут оказаться полезными при создании программ, предназначенных для решения практических задач из различных сфер человеческой деятельности: механики и электротехники, химии и сельского хозяйства...

В число этих выводов должны, на наш взгляд, войти такие:

1. Прежде чем приступать к составлению программы для решения технической задачи, убедитесь, нельзя ли построить аналитическое решение, то есть свести алгоритм к расчету по последовательно записанным формулам. Помните, что существование ЭВМ не освобождает от знания математики.

2. Старайтесь использовать такой мощный аппарат программирования, как косвенная адресация. С ее помощью удастся во многих случаях сильно уменьшить размеры программ.

3. Желательно при написании программ для решения технических задач предусматривать возможность повторять вывод результатов, особенно при эксплуатации программы в производственных условиях.

4. Старайтесь создавать такие программы, использование которых дает максимум удобств пользователю. Сокращая программы за счет бесполезных вычислительных команд, не экономьте на сервисе. Нажатие одной лишней клавиши при работе с программой не окупается сокращением ее длины даже на десяток команд.

12. МИКРОКАЛЬКУЛЯТОР – ПАРТНЕР В ИГРЕ

«Делу — время, потехе — час». Посвятив уже немало разделов работе с микрокалькулятором, мы можем уделить один игре с ним.

Вообще игра с ЭВМ — это не такое уж легкомысленное занятие. Достаточно сказать, что персональные ЭВМ, начавшие свой триумфальный путь в последнее десятилетие, первоначально задумывались именно как игровые. Не пренебрегал машинными играми и один из основоположников кибернетики К. Шеннон.

Игры с ЭВМ заслуживают внимания с двух точек зрения. Во-первых, это прекрасный способ приобщения новичка к вычислительной технике, ненавязчивое обучение пользованию клавиатурой, пониманию сообщений машины, да и, кроме того, неплохая тренировка логического мышления. Во-вторых, с точки зрения программиста, это хорошая возможность моделирования самых разнообразных ситуаций, использования «тонких» возможностей машины и проверки знаний основ программирования.

Конечно, игра становится более интересной, если используется дисплей, на котором игрок видит движущуюся картину: ракету, летящую к Луне, зайца, убегающего от волка, или людоеда, прячущегося за деревьями.

К сожалению, микрокалькулятор таких возможностей не имеет. На экран он может выводить только цифры, все остальное должно являться плодом воображения играющего. И тем не менее, игр с микрокалькулятором придумано немало.

В некоторых играх калькулятор выступает соперником своего владельца. Введенная в него программа выдает ответный ход на каждый очередной ход партнера. Разработка игровых программ — увлекательное занятие. Выбрав какую-то известную игру (скажем, «крестики — нолики» или «ним»), можно поставить вопрос: а существует ли для нее выигрышная стратегия? Иными словами, можно ли придумать алгоритм, позволяющий делать ходы так, чтобы их последовательность в конечном итоге вела к выигрышу? Если такая стратегия существует, если вами построен такой алгоритм, остается составить по нему программу для микрокалькулятора, и тогда ваша карманная ЭВМ, обученная и наставленная вами, обыграет любого, кто попытается сразиться с нею. Только вам самому эта игра никакого удовольствия уже не доставит. Да и всякий другой игрок после нескольких партий наверняка потеряет вкус к ней. И в самом деле: что за интерес играть, если ты заранее обречен на проигрыш?

Есть и такие игры с микрокалькулятором, где он не соперник своему пользователю, где роль его заключается в том, чтобы создавать игровую ситуацию. В этой ситуации пользователь сам ищет путь к цели — выигрышному итогу. Здесь в качестве аналогии можно привести кубик Рубика. Игровая ситуация создается случайным сочетанием цветных полей на его гранях. Цель — такое сочетание цветов, когда на каждой грани все поля одного цвета. Игра заключается в поиске такой последовательности выполнимых перестроек кубика, которая приводит к цели.

Особенно увлекательны такие игры, когда на пути к желаемому результату игрока преследуют случайности, неожиданности, требующие от него незамедлительных ответных решений. Еще большую остроту игра приобретает, когда игроков несколько: кто первым придет к цели? Таковы всевозможные гонки — от ребячьего бега наперегонки до многодневных авторалли.

Гонки мы и примем за образец игры, которую сейчас попытаемся запрограммировать для «Электроники БЗ-34». Конечно, условия гонок должны быть достаточно просты, чтобы игровая программа вместились в короткую память нашего калькулятора. Путь от старта до финиша пусть будет прямолинейным. Случайности пусть заключаются в каких-то боковых воздействиях, которые то и дело сбивают гонщика с дороги и требуют принятия мер, возвращающих его на прежний путь. Хорошо бы устроить так, чтобы эти случайности по-разному сказывались на гонщиках разного стиля. Пусть, скажем, вероятность неудачи повышается для тех, кто особенно рьяно устремляется вперед — тогда продвижение к финишу усложнится подбором оптимальной скорости. Для большей остроты состязаний можно запрограммировать и такие неудачи, которые полностью гасили бы скорость продвижения к финишу, как это происходит во время автогонок с автомобилем, сорвавшимся в кювет или врезавшимся в придорожный сугроб. Такой риск вылета на обочину должен подстергать гонщика (опять-таки для большей остроты борьбы!) буквально в каждый момент гонок — подобно тому, как в неспокойную погоду порывы ветра заставляют яхтсмена всегда быть настороже.

Попытаемся синтезировать все свои пожелания в едином образе. Здесь возможен, например, такой вариант. Замерзшее озеро занесено снегом. В снегу расчищено несколько параллельных ледовых дорожек одинаковой ширины. По ним скользят аэросани. Поперек их движения — допустим, справа — дует порывистый ветер, сносящий каждую машину с осевой линии

дорожки к обочине. Компенсировать снос можно поворотом руля. В определенные моменты, отстоящие друг от друга на условную единицу времени, гонщик, зная свое смещение от осевой линии и скорость аэросаней, может задавать приращение скорости и угол поворота руля: в этом и заключается очередной ход игры. Дорожку не будем делать ни слишком широкой, ни слишком узкой. Пусть ее ширина равна шести единицам, и сани стартуют, находясь на осевой линии, так что расстояние от левой и правой обочины равно трем единицам. Порывы ветра пусть будут не очень сильными, т. е. сани под воздействием этих порывов сносятся с осевой линии на расстояние, не превышающее половины единицы.

Договоримся также о правилах гонок. Во-первых, каждое очередное приращение скорости не должно превышать единицы. Иными словами, примем, что с каждым ходом игрока скорость саней либо уменьшается на единицу, либо на столько же увеличивается, либо остается прежней (в таком случае приращение скорости равно нулю). Во-вторых, скорость саней не должна падать до нуля. В-третьих, сани не должны выезжать на обочину. В каждом из этих трех случаев условимся начислять гонщику штрафное очко, а в последнем случае — гасить скорость его саней до нуля.

Составление игровых программ во многом аналогично составлению вычислительных и также может быть разбито на восемь условных этапов, перечисленных в разделе 2. Подобно тому как при разработке вычислительной программы все начинается с постановки задачи и ее математической формулировки, мы и здесь начали с описания игры, в ходе которого по возможности прибегали к цифрам. Судя по описанию, игра не представит затруднений при составлении программы для нее и в этом будет подобна тем математическим задачам, методы решения которых сводятся к последовательным вычислениям по цепочке формул. В программе для нашей игры, по всей вероятности, придется делать примерно то же самое: вычислять продвижение саней вдоль дистанции, их положение относительно краев дорожки, начислять штрафные очки за нарушение правил гонок. Алгоритмы подобных расчетов несложны, так что мы не будем останавливаться на них особо, а обсудим их при составлении блок-схемы.

Первый блок — ввод исходных данных, постоянных на всем протяжении гонок. Здесь, собственно, нужно ввести единственное число — длину дистанции L . Пусть она будет не очень большой, единиц 20—50. В этом же блоке предусмотрим очистку рабочих регистров, где будут находиться текущие значения скорости v , пройденного пути S , расстояния d от саней

до одного из краев дорожки (например левого), времени t , прошедшего с момента старта, силы порывов ветра ξ , угла поворота руля φ и приращения скорости саней Δv , задаваемые гонщиком, число заработанных им штрафных очков n .

Цепочка дальнейших блоков должна образовывать цикл, моделирующий очередной ход игрока, перемещение саней за единицу времени. Нелишнее в первом же из этих блоков сразу прибавить единицу к счетчику времени. И тотчас же хорошо бы вывести на индикатор показатели сиюминутной ситуации, чтобы гонщик знал расстояние от своих саней до левого края дорожки и их скорость.

Следующий блок — ввод данных на очередной ход: гонщик должен задать приращение скорости саней и угол отклонения руля.

Далее — блок проверки: не нарушил ли гонщик правила соревнований, не превысил ли он предел приращения скорости? Если превысил, надо перейти к блоку, где на индикатор будет выведен сигнал аварийного останова ЕГГОГ, а к счетчику штрафных очков прибавлена единица. Если правила не нарушены, работа программы продолжается. Вычисляется изменившаяся скорость аэросаней. И вновь проверка: может быть, гонщик по собственной оплошности допустил, что скорость его саней упала до нуля? Тогда все произойдет точно так же, как в ходе предыдущей проверки. Если же сани по-прежнему в движении, расчет переходит к новому этапу. Вычисляется путь, пройденный санями вдоль дистанции. Поскольку отдельный ход соответствует единице времени, то полное перемещение саней за один ход численно равно скорости, а продвижение вдоль дистанции — скорости, умноженной на косинус угла, который образуют направление дорожки и направление движения саней, то есть угла их отклонения от курса. И тут же — еще одна проверка: не достигнут ли финиш? Если да, то на индикатор надо вывести условный сигнал об этом и перейти к заключительному блоку «Конец». Если нет, программа работает далее.

В следующем блоке надо рассчитать поперечное смещение саней. Образуется оно двумя составляющими. Первая вызвана несовпадением курса саней с направлением дорожки, и ее значение за единицу времени равно произведению скорости саней на синус угла отклонения от курса. Вторая — это снос саней из-за ветра, дующего справа. Силу ветра будем представлять сложной также из двух компонент — постоянной и переменной, обусловленной случайными его порывами.

Структуру этого блока с самого начала стоит продумать повнимательнее. Как сделать так, чтобы чересчур лихой гон-

щик был бы более подвержен каверзам случайных порывов ветра? Чем больше скорость саней, тем большее расстояние пройдут они за очередной ход. А что, если рассчитывать это расстояние не просто перемножением скорости на синус угла отклонения от курса, а циклически? Прокрутить цикл столько раз, сколько единиц насчитывает значение скорости, и при каждом прохождении цикла выдавать очередное значение случайной составляющей скорости ветра? А постоянную составляющую поперечного смещения саней увеличивать также при каждом прохождении цикла, сделать ее пропорциональной скорости — только коэффициент пропорциональности взять поменьше, чтобы для устремившегося вперед гонщика риск врезаться в придорожный сугроб не был катастрофически велик.

Рассчитывая таким образом боковое поперечное смещение саней, будем по завершении каждого цикла проверять, не оказался ли гонщик на обочине — левой или правой. Если он пересек ограничительную линию, сбросим скорость его саней до нуля и начислим ему штрафное очко. Если нет, вернемся к началу цикла, чтобы продолжить вычисление поперечного смещения.

Кстати, не рассчитывать ли нам в том же цикле также и продвижение саней вдоль дистанции? Тогда удастся сэкономить две команды — вызов значения скорости из соответствующего регистра и его умножение на косинус угла отклонения от курса. При каждом прохождении цикла нужно будет лишь вычислять этот косинус и прибавлять его к значению пройденного пути. По выходе из этого «малого» цикла вернемся к началу «большого» — к той команде, которая прибавляет единицу к счетчику времени.

Осталось обдумать структуру блока начисления штрафных очков. Здесь должна быть команда прибавления единицы к счетчику этих очков, команда, вырабатывающая сигнал авоста, команда, выводящая на индикатор некое условное сообщение, по которому гонщик узнает причину штрафа. Возвращаться из этого блока, очевидно, следует на команду, прибавляющую единицу к счетчику времени. Это вполне справедливо. В самом деле, представьте себе, что было бы в аварийных случаях, если бы подобные гонки происходили в действительности. Если гонщик нарушил правила, судья останавливает его; если мотор аэросаней заглох, его надо запустить; если сани оказались в снегу, надо выбираться на лед — и на все это требуется время.

Блок-схема составлена. Можно приступать к написанию программы. Но прежде стоит уточнить некоторые детали.

Как, например, вырабатывать сигнал авоста? Известно, что он появляется на индикаторе, если не соблюдаются условия выполнения некоторых операций, например если при вычислении дроби знаменатель оказывается равным нулю или при нахождении арксинуса его аргумент превышает по модулю единицу. Но если идти к сигналу ошибки по такому пути, потребуется засылать некоторое число в регистр X, а между тем часто бывает желательно, чтобы его содержимое при авосте не портилось. Поэтому многие владельцы «Электроники БЗ-34» ищут иные пути получения сигнала ЕГГОГ, не занимающие много места в программе, быстродействующие и не портящие содержимого регистра X. Один из таких путей — введение в программу команды, задаваемой последовательным нажатием клавиш «K» и «—». Дойдя до нее, программа останавливается на следующем адресе, не выполняя стоящей по нему команды (поэтому она может быть какой угодно); на индикаторе при этом будет гореть сигнал авоста. Если же вслед за этим поставить команду C/P, то запущенный после авоста калькулятор остановится на ней, а на индикатор будет выведено содержимое регистра X. Целесообразно заранее заслать туда такое число, по виду которого игрок узнает причину штрафной остановки. Поскольку команда, стоящая между командами K— и C/P, при этом может быть любой, то в ее качестве можно поставить команду, засылающую в регистр X сигнал о достижении финиша, и именно на нее передавать управление, когда финиш достигнут.

Еще вопрос: как получать случайные числа? В книге А. Н. Цветкова и В. А. Епанечникова предлагается такой способ: взять некоторое случайное число ξ и определить его дробную часть; результат умножить на 11, прибавить к произведению π и от суммы отделить дробную часть; те же операции умножения на 11, сложения с π и отделения дробной части проделывать с каждым вновь получаемым числом. Так, одно за другим, будут возникать числа из диапазона между нулем и единицей. Как мы уже условились, для моделирования случайных порывов ветра станем делить каждое из этих чисел пополам.

Приведенный алгоритм обеспечивает длину неповторяющейся последовательности из примерно 8000 чисел, что для нашей игры более чем достаточно.

Тут, конечно, сразу же возникает такой вопрос: а как отделять от числа его дробную часть? По определению она равна разности между самим числом и его целой частью. А о получении целой части числа мы уже писали в разделе 7: если выполнить команду вида КИПN (где N — это число от

7 до 9 или буква от А до Д включительно), то в регистре N, где прежде находилось произвольное число, большее единицы, останется целая часть этого числа.

Все эти манипуляции по получению случайных чисел, выделению дробной части промежуточных результатов и т. п. требуют, конечно, немалого числа команд. И возникает опасение: уместится ли разрабатываемая нами программа в памяти «Электроники БЗ-34»? Постараемся быть лаконичными при ее составлении. Например, счетчики времени и штрафных очков оформим в виде команд КИПН, где номер регистра N равен одному из чисел от 4 до 6 включительно. Известно, что каждое выполнение такой команды увеличивает содержимое регистра N на единицу. А когда станем вычислять постоянную составляющую поперечного смещения саней, пропорциональную их скорости, то поступим просто: поделим скорость на 9. Нам ведь желательно, чтобы скорость делилась на возможно большее число. Девятка же — самое большое из однозначных чисел, то есть таких, вызов которых выполняется всего одной командой.

Займемся теперь распределением памяти. По опыту составления предыдущей программы мы уже знаем, что нулевой регистр удобно использовать как счетчик циклов. Так и поступим, когда в циклическом порядке будем рассчитывать продвижение саней вдоль дистанции и их поперечное смещение: продублируем в нулевом регистре значение их скорости и организуем цикл командой FL0. По соображениям, оговоренным в предыдущих абзацах, в регистрах 5 и 6 будем накапливать соответственно число штрафных очков n и единиц времени t , прошедших с момента старта; в регистре 7 пусть в каждый момент времени значится сила очередного случайного порыва ветра ξ . Регистры 8 и 9 отведем под показатели сиюминутной ситуации — расстояние d от саней до левого края дороги и скорость саней v соответственно.

Длину дистанции L будем хранить в регистре 1, а пройденный путь S — в регистре 2. Регистры 3 и 4 отдадим в распоряжение игрока: пусть он заносит туда соответственно угол поворота руля φ , выбирая его из диапазона между -90° и 90° , и очередное приращение скорости Δv , полагая его равным -1 , 0 или 1 . Поскольку угол поворота руля задан в градусах, заблаговременно установим переключатель «Р—Г» в положение «Г». Наконец, в регистре Д будем размещать число, по которому игрок узнает причину штрафного останова. Пусть это будут следующие числа: π , если слишком резко возросла скорость саней, нуль, если их скорость упала до нуля, или d , меньшее нуля или большее шести, если сани заехали за

обочину. Будем засылать эти числа в регистр Д в ходе проверки выполнения каждого условия, а вызывать их из регистра Д в регистр Х станем перед выводом на индикатор сигнала ЕГГОГ.

Приступим к составлению программы. Примем, что перед ее запуском в регистр Х введена длина дистанции L (напомним, что ей лучше придать значение из интервала между 20 и 50).

Программу начнем, как обычно, с адреса 00. Поместим под ним команду, пересылающую только что введенное число L на предписанное ему место — регистр 1. Затем установим сани на середину дороги — зашлем число 3 в регистр 8:

00.П1 01.3 02.П8

Очистим все рабочие регистры:

03.Сх 04.П2 05.П5 06.П6 07.П7 08.П9

Теперь приступим к программированию очередного хода гонщика. Прежде всего командой КИП6 прибавим единицу к счетчику времени:

09.КИП6

Затем дадим гонщику возможность получить представление о положении дел на данный момент: какова скорость его саней и не слишком ли близок он к обочине. Вызываем соответствующие данные одно за другим в стек и останавливаем программу:

10.ИП9 11.ИП8 12.С/П

Вызванные числа расположатся в регистрах Х и Y. Первое отображается на индикаторе, а чтобы прочесть второе, надо нажать клавишу «ХУ».

Оценив ситуацию, гонщик должен определить скорость и направление своего продвижения в предстоящую единицу времени. Для этого он должен последовательно ввести в калькулятор допустимое приращение скорости (одно из трех чисел: -1 , 0 , 1) и новый угол поворота руля (число градусов из диапазона от -90° до 90°). Сделав это, нажатием клавиши «С/П» он запускает калькулятор на расчет перемещения саней. Это, например, может выглядеть так: $1 \uparrow 45 \text{ С/П}$.

Как же будет протекать этот расчет?

Сначала отсылаем на свои места введенные значения угла поворота руля и приращения скорости:

13.П3 14.ХУ 15.П4

Предоставим калькулятору возможность проверить, выполнены ли условия игры, не превышен ли предел приращения скорости. Математически это условие выражается так: $|\Delta v| \leq 1$. Для его проверки можно было бы вычислить модуль Δv , но мы используем прием, описанный в разделе 9. Заменим неравенство равносильным, $(\Delta v)^2 \leq 1$, или еще более удобным с точки зрения программирования для микрокалькулятора: $1 - (\Delta v)^2 \leq 0$. После этого преобразования составление обсуждаемого фрагмента программы становится довольно очевидным. Начнем его засылкой числа π в регистр Д, а завершим командами условного перехода, который будет совершен при невыполнении только что сформулированного условия. Конкретное значение адреса перехода назвать еще невозможно, мы назначим его позже, когда станет ясно, где разместить «штрафной» фрагмент. Пока обозначим его A_1 :

16.F π 17.ПД 18.1 19.ИП4 20.Fx² 21.—
22.Fx ≥ 0 23.A₁

Если же условие приращения скорости соблюдено, вычислим новое ее значение и тут же проверим, не упала ли она до нуля. Если упала, то перейдем к «штрафному» фрагменту, занеся в регистр Д нулевое значение скорости:

24.ИП4 25.ИП9 26.+ 27.ПД 28.Fx $\neq 0$ 29.A₁

Если же скорость саней не нулевая, запишем ее значение в регистр 9 и продублируем это число в регистре 0:

30.П9 31.П0

Вычислим перемещение саней вдоль дистанции за единицу времени. Мы уже отмечали, что это перемещение равно $v \cos \varphi$, и условились, что и его, и поперечное смещение саней за то же время будем подсчитывать в циклическом порядке, повторив цикл столько раз, сколько единиц насчитывает величина v . При каждом прохождении цикла станем вычислять $\cos \varphi$, прибавлять результат к пройденному пути S и сравнивать полученное значение с длиной дистанции L . Сохраняя верность принципу экономии программной памяти, не будем расходовать на выполнение этой операции новые адресуемые регистры — необходимую информацию разместим в стеке: сначала — последнее из используемых чисел, далее — остальные в обратном порядке.

Сравнение S и L произведем, вычисляя их разность $L - S$. Если она окажется отрицательной — дистанция пройдена, надо останавливать калькулятор и высвечивать на индикаторе какое-то характерное число. Удобно использовать для этого

величину L . Поэтому в «штрафном» фрагменте между командами К — и С/П разместим команду ИП1 и именно на нее передадим управление при достижении финиша. Ее адрес определится позже, а сейчас проставим вместо него символ A_2 :

32.ИП1 33.ИП2 34.ИП3 35.F cos 36.+
37.П2 38.— 39.Fx ≥ 0 40.A₂

Если разность $L - S$ окажется положительной, надо продолжать счет — вычислять поперечное смещение саней.

В согласии с идеей его циклического вычисления станем определять составляющую смещения, обусловленную отклонением от курса, раз за разом прибавляя $\sin \varphi$ к d :

41.ИП3 42.F sin 43.ИП8 44.+

Рассмотрим теперь снос саней под воздействием ветра. Постоянную составляющую смещения мы условились вычислять на каждом шагу цикла как величину, пропорциональную скорости саней с коэффициентом пропорциональности, равным одной девятой. Учитываем эту величину со знаком минус, так как ветер дует справа:

45.ИП9 46.9 47.÷ 48.—

И вот теперь — случайные порывы ветра. Их моделирование было объяснено настолько подробно, что написание соответствующего фрагмента программы вряд ли представит затруднения. Вычтя эту случайную компоненту из общего смещения, занесем результат в регистры 8 и Д:

49.ИП7 50.1 51.1 52.× 53.Fπ 54.+
55.П7 56.КИП7 57.FO 58.ИП7 59.— 60.П7
61.2 62.÷ 63.— 64.П8 65.ПД

Сразу же проверим, не оказались ли мы на обочине. Пересечение левой ограничительной линии равносильно неравенству $d \leq 0$, пересечение правой — неравенству $6 - d \leq 0$. В обоих случаях отошлем программу к «штрафному» блоку. Попросим ее только оказать небольшую услугу гонщику, попавшему в неприятную оказию, — вытащить сани из снега на середину дорожки, то есть придать величине d значение 3. А скорости придадим значение нуль. Эти действия потребуют дополнительных операций, которые следует добавить перед «штрафным» фрагментом, начиная с адреса A_3 , значение которого выяснится позже. К моменту авоста в регистр X из регистра Д будет вызвана величина d , меньшая нуля или большая шести, поскольку сани оказались на обочине. По этим при-

метам гонщик и узнает причину штрафа:

66.Fx \geq 0 67.A₃ 68.6 69.ИП8 70.—
71.Fx \geq 0 72.A₃

Закончено программирование «малого» цикла, который должен быть пройден столько раз, сколько единиц в значении скорости. Надо организовать возврат на его начало, на ту команду, с которой начинается расчет перемещения саней вдоль дистанции:

73.FL0 74.32

Когда «малый» цикл пройден должное число раз, то тем самым закончено вычисление как продвижения саней вдоль дистанции за истекшую единицу времени, так и их смещения поперек дорожки за тот же период. Иными словами, выходом из «малого» цикла заканчивается и «большой» цикл, завершается расчет очередного хода. После этого надо вернуться к началу «большого» цикла, к команде, прибавляющей единицу к счетчику времени:

75.БП 76.09

Осталось написать «штрафной» фрагмент программы. Теперь ясно, что его начальный адрес A₃ — это адрес 77. И первые его команды — это сброс скорости саней до нуля, затем — возврат врезавшихся в снег саней на середину дорожки, то есть засылка тройки в регистр 8:

77.Cx 78.П9 79.3 80.П8

Далее — та часть, начало которой мы когда-то отметили адресом A₁. Это, как выясняется теперь, адрес 81. Здесь прибавляется единица к счетчику штрафных очков, в регистр X вызывается содержимое регистра Д, вырабатывается сигнал авоста, причем эти команды должны не идти друг за другом, а разделяться, как мы уже условились, командой вызова величины L — значения длины дистанции — из регистра 1. Ее адрес, ранее обозначенный нами A₂, теперь уже ясен: 84. После трех этих команд должны стоять команды возврата к началу «большого» цикла, к тому месту, где к счетчику времени прибавляется единица:

81.КИП5 82.ИПД 83.К— 84.ИП1 85.С/П 86.БП 87.09

Программа составлена. Просмотрим ее еще раз единым взглядом. Обдумаем порядок работы с ней.

Введя программу, набираем на клавиатуре длину дистанции. Запуск программы клавишами «В/О», «С/П» вскоре при-

водит к останову. На индикаторе светится число 3; оно показывает начальное положение саней посередине дорожки. Это число находится в регистре X, а в регистре Y содержится начальное значение скорости саней. Нажав клавишу «XY», можно убедиться, что оно равно нулю.

Вводим в регистр X приращение скорости, нажимаем клавишу «↑» и вводим угол поворота руля. (Напомним, что ветер дует справа, так что отрицательные значения угла вряд ли понадобятся сейчас и в дальнейшем.) Клавишей «С/П» пускаем программу далее. Через некоторое время она остановится вновь. На индикаторе — расстояние от саней до левой обочины; нажав клавишу «XY», считываем с индикатора значение их скорости. Только эти два числа известны гонщику перед совершением очередного хода. Сколько осталось до финиша, он не знает — вся его забота о том, чтобы не оказаться на обочине, то есть чтобы число, выводимое на индикатор после очередного останова, не было бы опасно близким ни к нулю, ни к шести. Этому гонщик добивается, вводя в калькулятор после каждого останова приращение скорости и новый угол поворота руля; тем самым определяется его перемещение за предстоящую единицу времени. Определяется оно не в полной мере: ведь сила порывов бокового ветра, сбивающего сани с пути, неизвестна гонщику, и чем быстрее устремляется он к финишу, тем выше такая неопределенность. Это вносит в игру некоторый оттенок риска, делает ее более увлекательной.

На каждом ходу программа контролирует действия гонщика и останавливается, во-первых, если превышен предел приращения скорости, во-вторых, если скорость упала до нуля, в-третьих, если сани оказались на обочине. В каждом таком случае гонщик получает штрафное очко и должен повторить свой ход, заново задать приращение скорости и угол поворота руля. В последнем случае штраф усугубляется еще тем, что скорость саней снижается до нуля и они вновь начинают свой разбег с точки, находящейся посередине дорожки на пройденном расстоянии от старта.

При каждой такой остановке на индикаторе горит слово ЕГГОГ — извещение о штрафе. Причину штрафа гонщик может узнать, нажав клавишу «С/П». Тогда на индикаторе загорается некоторое число. Если это число π — значит, слишком резко возросла скорость саней, если нуль — скорость упала до нуля, если число, меньшее нуля или большее шести, — сани заехали на обочину.

Если же во время остановки на индикаторе стоит длина дистанции, это означает, что сани достигли финиша. Тогда

надо посмотреть, сколько времени потрачено на прохождение заданного расстояния (вызвать на индикатор содержимое регистра 6) и сколько при этом заработано штрафных очков (вызвать содержимое регистра 5).

Выразим сказанное в виде инструкции.

Инструкция к программе.

1. Ввести программу.
2. Перейти в режим вычислений (FABT).
3. Очистить программный указатель (B/O).
4. Ввести длину дистанции (рекомендуется $20 \leq L \leq 50$). Запустить программу.
5. После остановки считать с индикатора расстояние саней от левого края дорожки d . Нажать клавишу «XY» и считать значение скорости саней v .
6. Ввести приращение скорости саней Δv , соблюдая условие $|\Delta v| \leq 1$, и новый угол поворота руля φ . Нажать клавишу «C/П». Если на индикаторе остановившегося калькулятора высвечивается длина дистанции, то перейти к п. 7, иначе — к п. 5.
7. Гонка окончена. Вводом команд ИП6 и ИП5 вызвать и считать время t , потраченное на прохождение дистанции, и число n полученных при этом штрафных очков.
8. Если на индикаторе остановившегося калькулятора высвечивается сигнал о штрафном останове ЕГГОГ, нажать клавишу «C/П». Вслед за этим на индикаторе появится сообщение о причине штрафа, выраженное условным числом. Если это число π — приращение скорости саней превысило единицу, если 0 — скорость саней упала до нуля, если число, меньшее нуля или большее шести, — сани заехали за обочину. После считывания перейти к п. 6.

Время, за которое калькулятор рассчитывает очередной ход, вероятно, покажется вам слишком долгим. Неудивительно: составляя программу, мы не позаботились о чистке единственного содержащегося в ней цикла (адреса 32—74). Попробуйте оптимизировать программу и сравните свой вариант с приведенным здесь (табл. 14). Он отличается тем, что на сей раз в цикле (адреса 54—80) подсчитывается лишь боковой снос из-за случайных порывов ветра; расчеты по формулам $s + v \cos \varphi$ и $d + v \sin \varphi - v^2/9$ ведутся вне цикла (адреса 32—53).

Потренировавшись в этой игре наедине с калькулятором, можно устроить состязание для нескольких участников. При этом надо очень внимательно следить за безошибочным вводом программы всеми участниками. Легче всего, конечно, пойти по пути механической сверки текста с содержимым программной памяти. Если программа введена в несколько калькуляторов сразу, то можно просто взаимно проконтроли-

Таблица 14

Адрес	Команда	Код	Адрес	Команда	Код
00	П1	41	50	Fx^2	22
01	3	03	51	9	09
02	П8	48	52	\div	13
03	Сх	0Г	53	—	11
04	П2	42	54	ИП7	67
05	П5	45	55	1	01
06	П6	46	56	1	01
07	П7	47	57	\times	12
08	П9	49	58	$F\pi$	20
09	КИП6	Г6	59	+	10
10	ИП9	69	60	П7	47
11	ИП8	68	61	КИП7	Г7
12	С/П	50	62	FO	25
13	П3	43	63	ИП7	67
14	ХУ	14	64	—	11
15	П4	44	65	П7	47
16	$F\pi$	20	66	2	02
17	ПД	4Г	67	\div	13
18	1	01	68	—	11
19	ИП4	64	69	П8	48
20	Fx^2	22	70	ПД	4Г
21	—	11	71	$Fx \geq 0$	59
22	$Fx \geq 0$	59	72	83	83
23	87	87	73	6	06
24	ИП4	64	74	ИП8	68
25	ИП9	69	75	—	11
26	+	10	76	$Fx \geq 0$	59
27	ПД	4Г	77	83	83
28	$Fx \neq 0$	57	78	ИП8	68
29	87	87	79	FL0	5Г
30	П9	49	80	54	54
31	ПО	40	81	БП	51
32	ИП1	61	82	09	09
33	ИП2	62	83	Сх	0Г
34	ИП3	63	84	П9	49
35	$F \cos$	1Г	85	3	03
36	ИП9	69	86	П8	48
37	\times	12	87	КИП5	Г5
38	+	10	88	ИПД	6Г
39	П2	42	89	К—	27
40	—	11	90	ИП1	61
41	$Fx \geq 0$	59	91	С/П	50
42	90	90	92	БП	51
43	ИП8	68	93	09	09
44	ИП3	63			
45	$F \sin$	1Г			
46	ИП9	69			
47	\times	12			
48	+	10			
49	ИП9	69			

ровать совпадение кодов. Почти невероятно, чтобы разные люди при вводе допустили одни и те же ошибки. Однако самым объективным критерием проверки работоспособности программы является ее тестирование (табл. 15).

Т а б л и ц а 15

d	v	Δv	Φ
3,0	0	1	0
2,8	1	1	3
1,7	2	1	45
2,0	3	1	20
0,5	4	1	70
1,1	5	1	80
1,0	6	1	85
0,8	7	1	87
ЕГГОГ на индикаторе: $-0,01$ ($d < 0$)			
			С/П С/П
3,0	0	1	0
2,8	1	1	5
1,9	2	1	10
0,5	3	1	15
на индикаторе: 20 (финиш) ИП6: $t = 12$ ИП5: $n = 1$			

Когда все участники гонок убедились, что игровая программа правильно введена во все калькуляторы, судья соревнований даст стартовый сигнал. Когда все гонщики придут к финишу, судья определяет победителя по времени прохождения дистанции и числу штрафных очков. Новую партию можно сыграть, вернув программу на адрес 00 командой В/О; перед этим, если требуется, можно ввести новую длину дистанции.

Развлечение, которое доставят вам такие гонки, — не единственная польза от этой игры. Обратим внимание, как она протекает: мы вводим некоторую информацию в калькулятор; переработав ее, он выдает нам ответную информацию; обдумав ее, мы вводим новую порцию информации, и так ход за ходом.

Такой режим работы с вычислительным устройством называется *диалоговым*. Признано, что такое использование вычислительной техники наиболее эффективно. Недаром способность к работе в диалоговом режиме стала неотъемлемым

свойством ЭВМ последних поколений. Своему пользователю они предоставляют разнообразные возможности общения: тут и печатающие устройства, и дисплеи со световым пером, и графопостроители ... Ничего этого нет у нашей карманной вычислительной машинки. Тем не менее в ходе увлекательной игры, описанной на этих страницах, она сумеет познакомить своего владельца с некоторыми существенными особенностями диалога «человек — ЭВМ».

«Делу — время, потехе — час». Час, который мы уделили потехе, игре с микрокалькулятором, был не таким уж праздным. Он позволил нам прийти к некоторым важным выводам.

1. Составление игровых программ — хороший способ повышения квалификации программиста, с одной стороны, и средство привлечения к работе с ЭВМ начинающих — с другой.

2. Игра с калькулятором будет интереснее при использовании элемента случайности, неповторяющихся ситуаций.

3. К каждой игровой программе должен быть составлен тест, так же как к каждой вычислительной программе — контрольный пример. Без этого очень трудно убедиться, правильно ли введена программа.

4. При программировании игр особое внимание нужно уделять сервису: простому вводу информации и легкой расшифровке сообщений машины.

5. Программирование игр — это шаг к использованию ЭВМ для моделирования жизненных ситуаций, а для владельцев ПМК — и к работе со своей ЭВМ в диалоговом режиме.

Приложение. Случайные и псевдослучайные числа

Случайным в математике называют такое число, которое возникает в результате некоторого случайного процесса, то есть процесса, исход которого непредсказуем. Определение довольно расплывчатое, поэтому проиллюстрируем его простым примером.

Если мы будем бросать игральную кость, то число очков, выпадающее при каждом бросании, будет разным, причем с одинаковой вероятностью может выпасть и 1, и 2, и 6, и любое другое из чисел, написанных на гранях кубика. Таким образом, мы можем получить идеальный датчик случайных чисел — нормированный, как говорят математики, от 1 до 6. Если выписывать результаты бросаний в ряд, то обнаружится интересная закономерность: сколь бы длинный отрезок этого ряда мы ни взяли, он не будет повторяться периодически на дальнейшем протяжении ряда. Поэтому предугадать

наверняка следующее число в этом ряду нам не дано. Непредсказуемость — это одна из основных особенностей случайных чисел.

Используются такие числа в вычислительной математике достаточно широко. Существует большой класс задач, для решения которых они просто необходимы. Некоторые из методов решения, использующие такие числа, получили игривое название «Метода Монте-Карло», по имени известной рулеточной столицы. Кстати, это название принадлежит вполне серьезным ученым Дж. фон Нейману и С. Уламу, использовавшим случайные числа при изучении поведения нейтронов. Необходимы эти числа и при решении множества не математических проблем, таких, как моделирование вызова абонентов на телефонной станции, проведение выборочной переписи населения и т. д.

ЭВМ, увы, не умеет подбрасывать игральный кубик и считать очки на его гранях. Можно, однако, создать программу, моделирующую этот процесс. Цель ее состоит в выработке последовательности чисел, которые можно интерпретировать, как случайные. Заметьте: интерпретировать, как случайные. Потому что на самом деле они не таковы. Их ряду свойственна периодичность, тогда как ряд случайных чисел обладать ею никак не может. Составить программу, которая давала бы непериодический ряд чисел, очень трудно. Другое дело, что можно добиться, чтобы повторяющийся фрагмент этого набора был достаточно большим, насчитывающим, скажем, тысячу или даже сотню тысяч чисел. Тогда на протяжении этого периода «машинная» последовательность не будет отличаться по своим свойствам от ряда подлинно случайных чисел. Называются числа, полученные таким путем, *псевдослучайными*, и именно они-то и используются в вычислительной математике.

В состав математического обеспечения большинства ЭВМ входят специальные программы, называемые *генераторами* или *датчиками псевдослучайных чисел*. Тем самым пользователь таких ЭВМ избавляется от необходимости изобретать подобные датчики самостоятельно.

В принципе, с помощью ЭВМ можно создать датчик псевдослучайных чисел, последовательность которых непериодична. Для этой цели можно использовать, например, цифры какого-нибудь иррационального числа, скажем, π . Построив алгоритм, вычисляющий эту величину с любой степенью точности, мы можем брать цифры этого числа подряд, будучи твердо уверены, что последовательность их никогда не повторится. На то ведь число и иррациональное, что выражающая

его десятичная дробь не периодическая, а значит, и появление в его представлении любой цифры на любом месте равновероятно*). Однако такой метод слишком «дорог». Даже на большой ЭВМ по мере увеличения числа цифр ждать каждую новую пришлось бы довольно долго. Поэтому обычно используются алгоритмы, которые генерируют пусть не случайные, а псевдослучайные числа, но зато такие, что получаются они быстро и период повторяемости у них достаточно велик.

Кстати, у псевдослучайных чисел есть даже преимущества перед «настоящими» случайными числами. Если последовательность «настоящих» повторить невозможно, то последовательность псевдослучайных чисел можно повторить в точности. Это имеет большое значение для отладки и тестирования программ.

На нашем микрокалькуляторе «Электроника БЗ-34» встроенного датчика случайных чисел нет. Поэтому вызов случайного числа с помощью одной команды — обращения к такому датчику — мы не можем сделать. Алгоритм получения случайных чисел и программу для его реализации приходится делать самостоятельно. Предпочтение при этом, как правило, отдается методам, которые дают последовательности хотя и не очень длинные, но зато такие, на выработку каждого числа которых тратится по возможности минимальное время.

13. СЕКРЕТЫ МИКРОКАЛЬКУЛЯТОРА

В названии книги стоит слово «секреты». Но какие уж тут секреты, когда почти все, о чем в ней говорится, отображено в той или иной форме либо в инструкции к микрокалькулятору, либо в специальной литературе о нем.

*) Кстати, анализ цифр числа π , как случайной последовательности, привел к занятным последствиям. Выяснилось, что живший в прошлом веке английский математик У. Шенкс, почти всю жизнь посвятивший вычислению знаков числа π и насчитавший их ни много ни мало 702, ошибся в 529-м знаке. Таким образом, весь его дальнейший труд пошел насмарку. А ведь Шенкс даже попросил выбить эти цифры на своем надгробном памятнике в качестве эпитафии, что и было сделано. Лишь через добрую сотню лет выяснилось, что эта надпись «неверна». Математики заметили, что семерка среди цифр Шенкса появляется реже, чем ей полагается по теории вероятностей. Ведь появление каждой цифры в случайном ряду равновероятно, а семерка в числе Шенкса появлялась реже любой из прочих цифр. Это породило сомнение в правильности его вычислений. Найдена была ошибка в наше время, когда с помощью ЭВМ число π было рассчитано с точностью до 10 000 знаков.

Однако в тексте некоторых разделов это слово употреблялось в смысле буквальном. Под «секретами» в этих случаях понимались свойства ПМК либо совсем не отраженные в инструкции, либо отраженные неполностью, либо, увы, отраженные неправильно.

Вообще, трудно переоценить роль инструкции. Представьте себе, что вы приобрели электроутюг, на котором две кнопки. В инструкции написано, что нажатие одной кнопки позволяет поддерживать температуру утюга на уровне 40°C , а другой — 60°C . И только. А если нужна температура 100°C ? Покупать новый утюг? Но, оказывается, можно получить такую температуру и на старом: надо нажать две кнопки сразу. Значит, инструкция сказала нам не все, что *можно* было сказать. Еще хуже, когда она не говорит всего, что *нужно* сказать. К примеру, через каждый час работы утюг нужно выключать, а то он перегреется и выйдет из строя, а в инструкции об этом не сказано ни слова.

К сожалению, в подобной ситуации подчас оказываются владельцы микрокалькулятора. С одной стороны, в инструкции, прилагаемой к нему, описаны не все его возможности, с другой — ничего не говорится о ситуациях, когда описанные возможности не реализуются.

О таких «секретах» и пойдет речь в этом разделе. Некоторые из них уже упоминались, но повторим их здесь для общности.

Начнем с «новых» команд, то есть таких, которые микрокалькулятор воспринимает, однозначно интерпретирует, но которые в инструкции почему-то не фигурируют.

Во всех командах, где используется имя регистра (0, 1, ..., ..., 9, A, ..., D), допустимо, кроме этих «явных» имен, ставить стрелку (\uparrow). Таких команд набирается 10. Это $\text{P}\uparrow$ (код 4E), $\text{IP}\uparrow$ (6E), $\text{KP}\uparrow$ (LE), $\text{КИП}\uparrow$ (GE), $\text{КБП}\uparrow$ (8E), $\text{КПП}\uparrow$ (—E) и команды косвенных условных переходов типа Kx?0 (здесь символом ? обозначены операции сравнения: $=$, \neq , \geq , $<$). Все эти команды используют нулевой регистр. Причем если действие двух первых, $\text{P}\uparrow$ и $\text{IP}\uparrow$, практически ничем не отличается от работы команд P0 и IP0 , то остальные работают по-другому. Содержимое регистра 0 при их использовании не меняется, в отличие от случаев явного употребления имени регистра в команде. Естественно, это значительно расширяет возможности программирования. Примеры использования подобных команд даны в двух предыдущих разделах.

Другая серия команд: $\text{K} +$ (26), $\text{K} -$ (27), $\text{K} \times$ (28), $\text{K} \div$ (29), KXY (2—), K3 (30), ..., K9 (36), K , (37), $\text{K}/-$ (38), КВП (39), КСx (3—) и $\text{K}\uparrow$ (3L). Эти команды вызывают авост и вывод

на индикатор сообщения ЕГГОГ. Их использование для получения этого сообщения эффективнее, чем запись некорректных операций (деление на нуль, извлечение корня из отрицательного числа и т. д.). Это свойство позволяет вводить в программы дополнительный сервис. Один из примеров такого использования приведен в разделе 12.

Правда, хотим предостеречь пользователей микрокалькулятора «Электроника БЗ-34» или подобных ПМК, появившихся позднее. Команды первой серии на ваших калькуляторах действуют не так, как мы их описали, а вторая серия может быть представлена у вас не полностью. Часть кодов, приведенных выше, задействована в новых типах микрокалькуляторов в других целях. Поэтому перед употреблением описанных в этом разделе команд убедитесь сначала, как они работают.

Еще одна «секретная» команда, точнее режим, который тоже может оказаться полезным. Многие, наверное, знают, что кроме измерения углов в градусах и радианах применяется еще одна измерительная шкала. Вся окружность делится при этом не на 360, а на 400 частей. Единицей угла в этом случае служит одна четырехсотая часть окружности, называемая *градом* или *гоном*. Прямой угол в этой шкале равен 100 градам. Такое деление в ходу, например, у моряков. Так вот, наш микрокалькулятор может работать и с гонами. Для этого переключатель «Р—Г» достаточно поставить в среднее положение.

Расскажем еще об одном случае, когда инструкция сказала не все, что *можно* было сказать.

Все команды типа ИРН и КИРН, где N — имя регистра, кроме своих основных функций записи и считывания информации, обладают свойством отбрасывать дробную часть числа, которое в них записано. Этим свойством мы также пользовались в ранее рассмотренных программах. Так же «расправляются» с дробной частью чисел и команды организации циклов FLN, где N = 0, 1, 2 и 3. Но интересно отметить, что происходит это только тогда, когда содержимое соответствующего регистра больше единицы. В противном случае действие команды меняется. Так, если в нулевой регистр записано положительное число, меньшее единицы, то по команде КИР0 каждый раз будет вычитаться единица из его последнего разряда. Эту особенность остроумно использовали А. Н. Цветков и В. А. Епанечников в программах по генерации случайных чисел, приведенных в их сборнике программ для микроЭВМ.

Часто при составлении программ для микрокалькулятора не по инструкции используется команда В/О. Основное ее

назначение — завершать подпрограмму, то есть передавать управление на команду, стоящую после обращения к подпрограмме ПП. Однако если подпрограмм в программе нет, то команда В/О передает управление на команду по адресу 01, иначе говоря, эквивалентна команде БП 01, занимая при этом на одну ячейку меньше места. Заметьте, что в режиме ручных вычислений клавиша «В/О» передает управление по адресу 00.

Несколько интересных хитростей связано с использованием команды ВП. Так, например, если в программе есть фрагмент \uparrow ПН ВП, то после него отбрасывается старший разряд результата. Для чисел в диапазоне от 0 до 10 это можно использовать для выделения дробной части числа. К примеру, введем программу

\uparrow П1 ВП С/П,

наберем на клавиатуре 1,56 и запустим программу на счет. Через пару секунд можно прочесть результат: $5,6 \cdot 10^{-1}$, то есть дробную часть результата.

О получении шифрованных сообщений с использованием той же команды подробно писалось в разделе 5. Этот прием особенно привлекателен при создании учебных и игровых программ, если, конечно, в распоряжении программиста есть свободные регистры.

Теперь о командах $\overline{\text{ШГ}}$ и $\overline{\text{ШГ}}$. Кодов они не имеют, так как по инструкции используются только в режиме программирования для изменения счетчика команд при просмотре текстов программ. Однако и в режиме вычислений они выполняют те же функции. Поэтому клавиша « $\overline{\text{ШГ}}$ » часто используется для передачи управления следующей команде в программе, состоящей из нескольких частей. Это значительно удобнее и быстрее, чем использование команд типа БП *mn* (*mn* — двузначный адрес).

Все вышесказанное — это, если хотите, подобно нажатию сразу двух кнопок на воображаемом утюге, о котором говорилось в начале раздела. Это — дополнительные возможности, которые могут помочь в составлении более высококачественных программ. Но без них можно и обойтись.

Мы же сейчас расскажем о ситуациях менее приятных, когда инструкция не говорит того, что *нужно было бы* сказать.

Это относится, в частности, к использованию команд \uparrow и Сх. Первая из них, как известно, перемещает числа в стеке на шаг вверх, а вторая очищает регистр X, записывая в него ноль. Введенное после этих команд число замещает содержи-

мое регистра X, не вызывая перемещений информации в стеке. Однако так происходит не всегда, а только в тех случаях, когда новое число вводится с клавиатуры или считывается из программной памяти сразу после выполнения команд \uparrow и Сх. Например, если стек был предварительно очищен, то после команд $5 \uparrow 8$ (неважно, выполняется этот фрагмент вручную или записан в памяти) регистры стека X, Y, Z и T будут заполнены так: 8, 5, 0, 0. Если же предварительно заполнить стек единицами, то выполнение команд Сх 8 заполнит стек числами 8, 1, 1, 1 соответственно.

Теперь запишем число 8 в один из адресуемых регистров, допустим в R1, проведем те же, что и упомянутые раньше, манипуляции по очистке и выполним команды $5 \uparrow$ ИП1. В стеке будут числа 8, 5, 5, 0. Команды же Сх ИП1 заполнят стек так: 8, 0, 1, 1.

Как видите, эти результаты отличаются от предыдущих. Здесь ввод очередного числа приводит к перемещению содержимого стековых регистров. Тот же эффект будет и при выполнении групп команд:

$5 \uparrow$ ПА 8 и Сх ПА 8

И в этих случаях ввод нового числа сдвигает информацию в стеке.

С первого взгляда это — несущественная мелочь. Новое число все равно попадает в регистр X. Однако, так как программирование на ПМК — это в первую очередь работа со стеком, то игнорирование описанных особенностей может привести к ошибкам в программах, где используются команды \uparrow и Сх.

К сожалению, есть в нашем микрокалькуляторе особенности, не только не описанные в инструкции, но и подчас противоречащие ей.

Введите в «Электронику БЗ-34» простенькую программу:

ПА /-/ С/П.

Наберите на пульте число, скажем 1, и запустите программу. Результат не заставит себя ждать. На индикаторе: -1, что совершенно верно. А теперь введем новое число:

$1 \cdot 10^{-3}$ (1 ВП 3 /-/).

Запускаем программу и читаем на индикаторе: 1000. Получается, что команда изменения знака в этом случае действует не на число, а на порядок, что находится в явном противоречии с описанием действия команды /-/. Но и это еще не все. Та же программа поможет выявить еще одну ошибку микрокалькулятора. Снова наберите на пульте $1 \cdot 10^{-3}$, но на

этот раз давайте запустим ее в режиме потактового прохода, нажимая клавишу «ПП». На индикаторе ответ: $-1 \cdot 10^{-3}$. На этот раз правильно. Но стоит ли радоваться этому? Ведь получается, что программа может дать разные результаты, будучи пропущенной в автоматическом режиме и в потактовом.

Таким образом, наша микропрограмма позволила сразу выявить две ошибки калькулятора, вовсе не очевидные.

Какие же выводы можно сделать, ознакомившись с описанными «секретами»?

1. Отладка программы с помощью режима потактовой прогонки — вещь безусловно нужная и полезная, однако не всегда позволяющая выявить все ошибки. Окончательным критерием правильности программы следует считать только расчет по контрольному примеру.

2. Составляя контрольный пример, старайтесь представлять исходные данные в разной форме, в том числе в экспоненциальном виде.

3. Шире используйте команды типа $KP\uparrow$ и $KIP\uparrow$. Они способствуют оптимизации программ.

14. ЧТО ДАЛЬШЕ?

В настоящее время во всех средних учебных заведениях нашей страны введен курс «Основы информатики и вычислительной техники». Приобщение молодежи к компьютерной грамотности становится реальностью.

Программируемый микрокалькулятор «Электроника БЗ-34» — один из первых шагов на этом пути. Конечно, хорошо, если бы слово «микро» в его названии относилось бы только к размерам. К сожалению, в нем все «микро»: и быстрое действие, и емкость памяти, и язык, а стало быть, и возможности.

Но так же, как, изучая каплю воды, можно составить представление о мировом океане, так и осваивая микрокалькулятор, можно получить представление о необъятном мире ЭВМ. Ведь как и всякая ЭВМ, наша миниатюрная машинка призвана освободить своего владельца от рутинной работы, сделать его труд более творческим и производительным. Главное же, она помогает выработать особый алгоритмический стиль мышления, без которого трудно представить человека завтрашнего дня независимо от профессии.

Потому-то наш микропредставитель семейства ЭВМ заслуживает самого серьезного отношения. Научиться программировать — легко. Научиться программировать хорошо — во сто крат труднее.

Говорится это не для того, чтобы каждый прочитавший эту книгу посвятил всю свою жизнь углублению знаний в программировании на микрокалькуляторах. В этом особой нужды нет. Во-первых, не надо забывать, что калькулятор (как, впрочем, и любая ЭВМ) — не самоцель, а средство для решения конкретных задач. Во-вторых, на смену ПМК идут гораздо более мощные представители вычислительного семейства: персональные компьютеры, тоже находящиеся в личном распоряжении, но обладающие гораздо большими возможностями, допускающие использовать в качестве дисплея обычный серийный телевизор, а в качестве внешнего запоминающего устройства — бытовой кассетный магнитофон. Программирование на персональных компьютерах отличается от программирования на микрокалькуляторах. Персональные компьютеры оснащены алгоритмическими языками, запись программ на которых намного нагляднее, чем на командном языке микрокалькулятора, и набором столь удобных программ, что работать с ними сможет человек, вообще с программированием не знакомый.

А что же будет с нашим другом микрокалькулятором? Не будет ли он вытеснен персональной ЭВМ, как конка была вытеснена трамваем?

Скорее всего, нет. Ведь, что ни говори, а персональный компьютер — машина все-таки стационарная, место ее на рабочем столе. А программируемый микрокалькулятор — машина портативная. Места она занимает мало, питаться может от автономного источника питания — аккумулятора или батарейки, ее можно носить в портфеле и даже в кармане. Потому она в буквальном смысле слова всегда под рукой. Но дело не только в этом. Рискнем утверждать, что для решения небольших числовых задач она, пожалуй, и удобнее. Минимальная клавиатура, команды, названия которых написаны прямо на клавишах, простота языка — все это можно отнести к явным преимуществам ПМК перед другими ЭВМ при решении небольших задач.

И вообще, нужно ли противопоставлять друг другу разные средства автоматизации вычислений? Перефразируя известную поговорку, можно сказать: «Всякой задаче — свою ЭВМ!» Только в этом случае программист не станет уподобляться чудаку из другой поговорки, который стреляет из пушек по воробьям, а из рогатки — по слону.

Мирное сосуществование различных средств обработки информации давно уже стало реальностью. Давайте вспомним, сколько лет «трудились» бок о бок логарифмические линейки и различные типы арифмометров — предшественники нынеш-

них микрокалькуляторов. Линейки, кстати, выпускаются до сих пор. Причем не только они сами, но и литература о них. К примеру, книга Д. Ю. Панова «Счетная линейка» вышла совсем недавно 25-м изданием.

Вот линейкам, по нашему мнению, микрокалькулятор будет серьезным конкурентом. Ведь он так же доступен, как и линейка, может сделать все, что делает она, причем гораздо быстрее и точнее, а в обращении он, пожалуй, даже попроще.

Но для нашего программируемого микрокалькулятора подобная работа — это, если можно так выразиться, езда на велосипеде с моторчиком без использования моторчика. «Моторчик» ПМК — это способность работать по введенной в него программе. Именно она делает его вполне конкурентоспособным с «мотоциклами» — персональными ЭВМ, и даже с более серьезными вычислительными «автомобилями» и «самолетами». При езде на небольшие расстояния, или, переходя с языка образов на обычный, при решении небольших численных задач, он даже удобнее.

Кроме того, по мере развития технологии возможности микрокалькуляторов будут возрастать. Шире станет набор команд, появится возможность работать не только с числами, но и с символами, программировать как в командах, так и на алгоритмических языках высокого уровня. И все это, естественно, без существенного подорожания калькуляторов и усложнения работы с ними, а может быть, даже наоборот.

И еще: мы считаем, что опыт работы с микрокалькулятором пригодится и тем, кто решит перейти к более мощным типам вычислительных устройств. Ведь этапы решения научно-технических задач с помощью ЭВМ не зависят от типов машин. Умение правильно ставить задачу, давать ей четкую математическую формулировку, выбирать оптимальный алгоритм ее решения и составлять его блок-схему необходимо всем, кто имеет дело с вычислительной техникой.

Что же касается борьбы за лаконичность и быстродействие программ, то хотя эти вопросы для больших ЭВМ не стоят столь остро, как для их микрособратьев, опыт такой борьбы не бесполезен. Чем больше возможности машины, тем более сложные задачи захочется решать на ней, и всегда найдутся проблемы, для решения которых даже самая большая ЭВМ окажется малой. И здесь-то очень пригодятся навыки работы с микрокалькулятором, умение укладывать программы на прокрустово ложе его памяти. Во всяком случае, заметим в заключение, человек, освоивший микрокалькулятор, станет увереннее осваивать любую другую ЭВМ.

СОДЕРЖАНИЕ

Предисловие	3
1. Первое знакомство	5
2. Какие задачи и как решать на программируемом микро- калькуляторе?	13
3. Логика микрокалькулятора	23
4. Язык микрокалькулятора	34
5. Блок-схема — портрет программы	45
6. Ветви, циклы, подпрограммы	62
7. По косвенным адресам	74
8. Как отлаживать и улучшать программы	79
9. Погрешности вычислений	94
10. Сервис службы программирования	107
11. Калькулятор — помощник в работе	119
12. Микрокалькулятор — партнер в игре	135
13. Секреты микрокалькулятора	152
14. Что дальше?	157

Игорь Данилович Данилов

СЕКРЕТЫ ПРОГРАММИРУЕМОГО МИКРОКАЛЬКУЛЯТОРА

Серия «Библиотечка «Квант», вып. 55

Редакторы *Н. И. Воронина, М. Н. Андреева*

Художественный редактор *Т. Н. Кольченко*

Технический редактор *В. Н. Кондакова*

Корректоры *Т. С. Родионова, Н. Д. Храпко*

ИБ № 12987

Сдано в набор 31.03.86. Подписано к печати 24.09.86. Т-19588. Формат 84 × 108/32. Бумага тип. № 2. Гарнитура таймс. Печать высокая. Усл. печ. л. 8,4. Усл. кр.-отг. 9,24. Уч.-изд. л. 10,05. Тираж 300 000 экз. Заказ № 340. Цена 30 коп.

Ордена Трудового Красного Знамени издательство «Наука»

Главная редакция физико-математической литературы

117071 Москва В-71, Ленинский проспект, 15

Ордена Октябрьской Революции, ордена Трудового Красного Знамени Ленинградское производственно-техническое объединение «Печатный Двор» имени А. М. Горького Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 197136 Ленинград П-136, Чкаловский пр., 15

2-й символ

0	1	2	3	4	5	6	7	8	9	-	L	С	Г	Е
0	0	1	2	3	4	5	6	7	8	9	/-/	ВП	Сх	↑
1	+	-	X	÷	$\vec{x}\vec{y}$	$F10^x$	Fe^x	Fzg	$F\sin$	$F\arccos$	$F\arctg$	$F\sin$	$F\cos$	FBx
2	Fπ	Fγ	Fx^2	$F^{1/x}$	$F\vec{x}$	$F\vec{y}$								Ftg
3														
4	по	п1	п2	п3	п4	п5	п6	п7	п8	п9	пA	пB	пC	п↑
5	с/п	Бп	В/0	пп	кноп			$Fx \neq 0$	FL2	$Fx \geq 0$	FL3	FL1	$Fx < 0$	$Fx = 0$
6	ипо	ип1	ип2	ип3	ип4	ип5	ип6	ип7	ип8	ип9	ипA	ипB	ипC	ип↑
7	$Kx \neq 0$	$Kx \neq 01$	$Kx \neq 02$	$Kx \neq 03$	$Kx \neq 04$	$Kx \neq 05$	$Kx \neq 06$	$Kx \neq 07$	$Kx \neq 08$	$Kx \neq 09$	$Kx \neq 0A$	$Kx \neq 0B$	$Kx \neq 0C$	$Kx \neq 0\Delta$
8	КБпо	КБп1	КБп2	КБп3	КБп4	КБп5	КБп6	КБп7	КБп8	КБп9	КБпA	КБпB	КБпC	КБп↑
9	$Kx \geq 0$	$Kx \geq 01$	$Kx \geq 02$	$Kx \geq 03$	$Kx \geq 04$	$Kx \geq 05$	$Kx \geq 06$	$Kx \geq 07$	$Kx \geq 08$	$Kx \geq 09$	$Kx \geq 0A$	$Kx \geq 0B$	$Kx \geq 0C$	$Kx \geq 0\Delta$
-	кппо	кпп1	кпп2	кпп3	кпп4	кпп5	кпп6	кпп7	кпп8	кпп9	кппA	кппB	кппC	кпп↑
L	кпо	кп1	кп2	кп3	кп4	кп5	кп6	кп7	кп8	кп9	кпA	кпB	кпC	кп↑
С	$Kx < 0$	$Kx < 01$	$Kx < 02$	$Kx < 03$	$Kx < 04$	$Kx < 05$	$Kx < 06$	$Kx < 07$	$Kx < 08$	$Kx < 09$	$Kx < 0A$	$Kx < 0B$	$Kx < 0C$	$Kx < 0\Delta$
Г	кипо	кип1	кип2	кип3	кип4	кип5	кип6	кип7	кип8	кип9	кипA	кипB	кипC	кип↑
Е	$Kx = 0$	$Kx = 01$	$Kx = 02$	$Kx = 03$	$Kx = 04$	$Kx = 05$	$Kx = 06$	$Kx = 07$	$Kx = 08$	$Kx = 09$	$Kx = 0A$	$Kx = 0B$	$Kx = 0C$	$Kx = 0\Delta$

1-й символ

