

Глава 3. Учебная машина

*Никто не заставит тебя учиться.
Учиться ты будешь тогда, когда захочешь этого.*

*Ричард Бах.
«Карманный справочник Мессии»*

*Учение без размышления бесполезно, но и
размышление без учения опасно.*

Конфуций, V век до н.э.

В этой главе, используя принципы Фон Неймана в качестве базовых, будет сконструирована собственная учебная ЭВМ. Для понимания особенностей архитектуры ЭВМ нам понадобятся несколько различных учебных машин, но одна из них, однако, будет изучаться более подробно, чтобы иметь возможность писать для неё простые полные программы на языке машины. Будем называть эту машину УМ-3, смысл этого названия – учебная машина трёхадресная. УМ-3 будет удовлетворять *всем* принципам Фон Неймана.

Для построения конкретной ЭВМ необходимо строго определить характеристики памяти этой машины, задать набор машинных команд, описать устройство процессора, определить правила взаимодействия с "внешним миром" (операции ввода/вывода) и т.д.

Память УМ-3 будет состоять из 512 (это 2^9) ячеек, имеющих адреса от 0 до 511. Каждая ячейка состоит из 32 двоичных разрядов. По современным понятиям это очень маленькая память (всего 2 Kb).

Будем предполагать, что целые числа имеют тот же формат, что и тип Longint, а вещественные – тип Single в языке Free Pascal, внутреннее представление этих форматов будет приведено в следующей главе. Записанное в ячейке машинное слово может трактоваться (рассматриваться) как одно целое или вещественное *число*¹ или как одна *команда*. Машинное слово, трактуемое как команда, будет разбиваться на четыре поля, и представляться в следующей форме:

КОП	A1	A2	A3
5 разрядов	9 разрядов	9 разрядов	9 разрядов

Номера ячеек, кодов операций и адресов операндов будем для удобства записывать в десятичном виде, хотя первые программисты использовали для этого шестнадцатеричную или восьмеричную системы счисления. Первое поле с именем КОП содержит число от 0 до 31. Это число задает номер (код) операции машинной команды, а поля с именами A1, A2 и A3 задают адреса операндов (это целые числа от 0 до 511). Таким образом, в каждой команде могут задаваться адреса двух аргументов (это A2 и A3) и адрес результата операции A1.

Определим далее, какие регистры находятся в устройстве управления.

- RA – регистр, называемый *счётчиком адреса*, он имеет 9 разрядов и во время выполнения текущей команды будет содержать адрес команды, которая должна будет выполняться следующей;
- RK – регистр команд имеет 32 разряда и содержит текущую выполняемую команду, которая, как уже говорилось, содержит код операции КОП и адреса операндов A1, A2 и A3;
- ω – регистр длиной два бита, его название произносится как "омега" (оно ещё называется регистром-признаком результата). В этот регистр после выполнения некоторых команд (в УМ-3 это будут арифметические команды сложения, вычитания, умножения и деления) записывается число от 0 до 2 по следующему правилу (S – числовой результат арифметической операции, полученной на регистре-сумматоре АЛУ):²

¹ Будем предполагать, что целые числа имеют тот же формат, что и тип Longint, а вещественные – тип Single в языке Free Pascal, внутреннее представление этих форматов будет приведено в следующей главе.

² Аналогом регистра ω в процессорах фирмы Intel является комбинация бит C3 и C0 в так называемом регистре состояния сопроцессора, они принимают значения: 00 – результат > 0, 01 – результат < 0, 10 – результат = 0.

$$w := \begin{cases} 0, & S = 0 \\ 1, & S < 0 \\ 2, & S > 0 \end{cases}$$

- Err – регистр ошибки, содержащий нуль в случае успешного выполнения очередной команды и единицу в противном случае.

В таблице 3.1 приведены все различные команды учебной машины УМ-3. Множество всех допустимых форматов команд называется, как уже говорилось, *языком машины*.

Таблица 3.1. Команды учебной машины УМ-3

КОП	Смысл операции и её мнемоническое обозначение
01	СЛВ – сложение вещественных чисел
11	СЛЦ – сложение целых чисел
02	ВЧВ – вычитание вещественных чисел
12	ВЧЦ – вычитание целых чисел
03	УМВ – умножение вещественных чисел
13	УМЦ – умножение целых чисел
04	ДЕВ – деление вещественных чисел
14	ДЕЦ – деление целых чисел (то же, что и div в Паскале)
15	МОД – остаток от деления (то же, что и mod в Паскале)
00	ПЕР – пересылка: <A1>:=<A3>
10	ЦЕЛ – вещественное в целое: <A1>:=Round (<A3>)
30	ВЕЩ – целое в вещественное: <A1>:=Real (<A3>)
09	БЕЗ – безусловный переход: goto A2, т.е. RA:=A2
19	УСЛ – условный переход: case ω of 0: RA:=A1; 1: RA:=A2; 2: RA:=A3 end
20	ПР – переход по S = 0: if ω=0 then RA:=A2
21	ПНР – переход по S <> 0: if ω<>0 then RA:=A2
22	ПВ – переход по S > 0: if ω=2 then RA:=A2
23	ПМ – переход по S < 0: if ω=1 then RA:=A2
24	ПБР – переход по S >= 0: if ω<>1 then RA:=A2
25	ПМП – переход по S <= 0: if ω<2 then RA:=A2
31	СТОП – остановка выполнения программы
06	ВВВ – ввод A2 вещественных чисел в память, начиная с адреса A1 for i:=1 to A2 do Readln (<A1+i-1>)
07	ВВВ – вывод A2 вещественных чисел, начиная с адреса A1 for i:=1 to A2 do Writeln (<A1+i-1>)
16	ВВЦ – ввод массива целых чисел, аналогично ВВВ
17	ВВЦ – вывод массива целых чисел, аналогично ВВВ

Как вскоре станет ясно, даже этого маленького набора команд окажется достаточно для программирования не очень больших задач. На самом деле этот набор команд полный: его достаточно для программирования всех задач, для решения которых хватает ресурсов этой учебной машины. И пусть Вас не смущает отсутствие многих типов данных, привычных в языках высокого уровня, так как, например, символы можно хранить в виде целых чисел – номеров символов в алфавите, логические значения – в виде целых чисел 0 и 1 и т.д. Более того, внимательное рассмотрение языка нашей машины выявит даже его *избыточность*. Так, например, вместо команды безусловного перехода можно использовать команду условного перехода, в которой все три адреса совпадают. Все команды условных переходов, кроме команды с мнемоникой УСЛ, также являются избыточными.

Можно сказать, что при разработке архитектуры этой учебной машины отдавалось некоторое предпочтение удобству программирования в ущерб экономии электронных схем процессора. Хорошо, однако, вовремя остановиться и не пойти по этому пути дальше, например, не вводить в язык машины команду вычисления абсолютной величины или квадратного корня (такие команды часто реализовывались в ЭВМ первого поколения).

Также предполагается, что внешнее представление целых и вещественных чисел на устройствах ввода, как, естественно, и на устройстве вывода, различаются.

3.1. Схема выполнения команд

Большинство людей находят концепцию программирования очевидной, но само программирование невозможным.

*Алан Перлис,
первый лауреат премии Тьюринга*

Рассмотрим теперь схему работы процессора. В соответствии с принципом Фон Неймана последовательного выполнения команд, процессор выполняет одну команду за другой, пока не выполнит команду СТОП или же пока очередная команда не зафиксирует при своём выполнении аварийную ситуацию, установив в единицу регистр Err в устройстве управления. Выполнение всей программы для наглядности будет записано в виде цикла языка Паскаль, причём тело этого цикла соответствует выполнению одной команды:

repeat

1. $RK := \langle RA \rangle$; чтение очередной команды из ячейки памяти с адресом RA на регистр команд RK в устройстве управления.
2. $RA := RA + 1$; увеличение счётчика адреса на единицу, чтобы следующая команда (если текущая команда не является командой перехода) выбиралась из следующей ячейки памяти.
3. Выполнение операции, заданной в коде операции (КОП) в АЛУ, при необходимости знак результата записывается на регистр ω . При несуществующем коде операции или других ошибках при выполнении команды (например, делении на ноль) выполнение команды прекращается и производится присваивание $Err := 1$.

until ($Err=1$) **or** ($КОП=СТОП$)



Уточним теперь работу арифметико-логического устройства в нашей УМ-3. Все *бинарные* операции (т.е. те, которые имеют два аргумента и один результат) выполняются в АЛУ нашей учебной машины по схеме: $\langle A1 \rangle := \langle A2 \rangle \otimes \langle A3 \rangle$ (\otimes – любая бинарная операция). Для реализации таких операций в арифметико-логическом устройстве предусмотрены регистры первого (R1) и второго (R2) операндов, а также регистр сумматора (S), на котором размещается результат операции. Таким образом, как и в машине Фон Неймана, АЛУ, подчиняясь управляющим сигналам со стороны УУ, выполняет бинарную операцию по схеме

$R1 := \langle A2 \rangle$; $R2 := \langle A3 \rangle$; $S := R1 \otimes R2$; $\langle A1 \rangle := S$;
Формирование регистра ω для арифметических операций.

Теперь осталось определить условие начала работы программы. Прежде всего, программа каким-то образом должна оказаться в памяти нашей ЭВМ. Вообще говоря, исходя из принципа неразличимости команд и данных, нашу программу можно, например, представить в виде массива целых чисел и ввести в память, используя команду ввода (например, с кодом операции 16, см. таблицу 3.1). Заметим, однако, что для *начального* ввода программы в память нельзя использовать *команды ввода* из языка машины, так как для этого в памяти уже должна находиться такая команда. Чтобы разорвать этот замкнутый круг, в современных ЭВМ обычно часть памяти отводят для постоянного хранения специальной небольшой программы, которая называется программой *начальной загрузки*. Эта программа начинает автоматически выполняться при включении ЭВМ.¹ Память, занимаемая программой начальной загрузки, недоступна для записи, такую память, как уже говорилось, обычно обозначают английской аббревиатурой ROM – Read Only Memory. Ясно, что наличие программы начальной загрузки нарушает принцип Фон Неймана однородности памяти.

В учебной ЭВМ пойти по этому пути загрузки программы нельзя по двум причинам. Во-первых, память и так очень маленькая, и отводить часть её для постоянного хранения программы начальной загрузки нерационально, а, во-вторых, в первой учебной ЭВМ мы договорились не отступать от принципа Фон Неймана однородности памяти. В УМ-3 первичной загрузкой программы в память и формированием начальных значений регистров в устройстве управления занимается *устройство*

¹ При включении многоядерного процессора сначала активно только одно ядро. Перед собственно загрузчиком, который вводит в память первую после включения компьютера программу, работает служебная процедура POST (Power On Self Test), она проверяет правильность функционирования основных узлов ЭВМ и выполняет ещё некоторые другие вспомогательные действия.

ввода. Для этого на устройстве ввода имеется специальная кнопка ПУСК ( Пуск ). При нажатии этой кнопки устройство ввода самостоятельно (без сигналов со стороны устройства управления, которое ещё не функционирует) выполняет следующую последовательность действий:

1. Производит ввод расположенного на устройстве ввода массива машинных слов в память, начиная с *первой ячейки*; этот массив машинных слов заканчивается специальным *признаком конца ввода*.¹
2. $RA := 1$; первой будет выполняться команда из ячейки с номером один.
3. $\omega := 0$; начальное значение признака результата нулевое.
4. $Err := 0$; признак ошибки сбрасывается (устанавливается равным **false**).

Как видно, при нажатии кнопки ПУСК устройство ввода выполняет основные функции упомянутой выше программы начальной загрузки современных ЭВМ. Вот теперь всё готово для автоматической работы процессора по загруженной в память программе, и за дело принимается устройство управления.

Таким образом, полностью определены условия начала и конца работы этой учебной машины как алгоритмической системы. Здесь по аналогии хорошо вспомнить, как условия начала работы определялись, скажем, для машины Тьюринга: на ленте расположено входное слово, головка установлена у первого слева символа этого слова и машина находится в первом состоянии.




ЭВМ Стрела, 1953 г.

Отметим, что по своей архитектуре УМ-3 похожа на первые ЭВМ, построенные в соответствии с принципами Фон Неймана, например, на отечественную серийную ЭВМ Стрела [3], выпускавшуюся в середине прошлого века. Эта трехадресная машина имела оперативную память 4096 43-разрядных слов, имела 64 команды, потребляла 150 кВт и занимала площадь 300 кв.м. ЭВМ Стрела содержала 6200 электронных ламп и 60000 полупроводниковых диодов, она работала со скоростью примерно 2000 операций в секунду. Именно на Стреле рассчитывались траектории полёта первых спутников и первого космонавта Юрия Гагарина. Эти ЭВМ стояли в Академии наук СССР и в центрах Советского Союза по разработке ядерного оружия "Арзамас-16" и "Челябинск-70". Заметим, что Стрела с серийным №4 была установлена в Научно-исследовательском вычислительном центре МГУ на Ленинских Горах.

3.2. Примеры программ для УМ-3

При изучении наук примеры полезнее правил.

Исаак Ньютон

Далее рассматриваются несколько простых полных программ для данной учебной машины. Эти программы призваны проиллюстрировать основные приёмы программирования на языке машины для первых ЭВМ, удовлетворяющих всем принципам Фон Неймана. Такие примеры позволят Вам лучше понять способ работы учебной ЭВМ и, следовательно, её архитектуру. Итак, представим себя первыми программистами .

3.2.1. Пример 1. Оператор присваивания

Каждый должен учиться программированию, потому что оно учит думать.

Стив Джобс

Составим программу, которая вводит целое число x и реализует арифметический оператор присваивания языка Free Pascal:

¹ В качестве внешних носителей информации в первых ЭВМ часто использовались специальные картонные *перфокарты*, на которых отверстия кодировались хранимые данные. Например, для нашей УМ-3 каждое машинное слово будет занимать на перфокарте 32 позиции, причём в тех позициях, где машинное слово имеет разряд единицу, пробивается отверстие. Признак конца ввода массива является специальным словом, которое имеет дополнительные (сверх 32-х) позиции (обычно признак конца ввода располагался на отдельной перфокарте, в память он не вводился). Заметим, что такой массив с признаком конца ввода аналогичен современному понятию последовательного файла машинных слов.

$$y := (x+1)^2 \bmod (2 \cdot x)$$

На языке Free Pascal эта программа может выглядеть, например, так:

```
program First;
  var x,y: Longint;
begin Read(x); y:=sqr(x+1) mod (2*x); Writeln(y) end.
```

Текст первой программы с комментариями приведён на рис. 3.1 (напомним, что все числа – десятичные). Подробно прокомментируем процесс написания этой машинной программы. Как Вы уже знаете, команды программы после начального ввода при нажатии кнопки ПУСК будут располагаться в памяти, начиная с первой ячейки. Теперь необходимо решить, в каких ячейках памяти будут располагаться переменные x и y . Эта работа называется *распределением памяти* под хранение переменных. При программировании на Паскале эту работу выполняла за нас Паскаль-машина, когда видела описания переменных:

```
var x,y: Longint;
```

Теперь нам самим придётся распределять память под хранение переменных. Сделаем сначала естественное предположение, что наша программа будет занимать не более 100 ячеек памяти. Поэтому, начиная со 101 ячейки, память будет свободна. Пусть тогда для хранения значения переменной x будет выделена 101 ячейка, а переменной y – 102 ячейка памяти. Остальные переменные при необходимости будут размещаться в последующих ячейках памяти. Для реализации этой программы также понадобятся дополнительные (как говорят, *рабочие*) переменные $r1$ и $r2$, которые будут размещаться в ячейках 103 и 104 соответственно.

№	Команда					Комментарий
001	16	101	001	000		Read(x)=Read(<101>)
2	11	103	101	008		$r1 := (x+1)$
3	13	103	103	103		$r1 := (x+1)^2$
4	11	104	101	101		$r2 := (x+x)=2 \cdot x$
5	15	102	103	104		$y := r1 \bmod r2$
6	17	102	001	000		Write(y)
7	31	000	000	000		Стоп
8	00	000	000	001		Целая константа 1

Рис 3.1. Программа первого примера.

Теперь надо понять, что при программировании на этом машинном языке, в отличие от Паскаля, не будут существовать константы. Действительно, в какой бы ячейке не хранилось значение константы, по принципу Фон Неймана однородности памяти ничто не мешает записать в эту ячейку новое значение. Поэтому будем называть *константами* такие переменные, которые имеют начальные значения, и которые не планируется изменять в ходе выполнения программы.

Важно понять, что в начале работы программы имеют значения только те ячейки памяти, в которые произведён ввод данных по кнопке ПУСК.¹ Отсюда следует, что константы, как и переменные с начальным значением, следует располагать в *тексте программы* ("маскируя" их под команды) и загружать в память вместе с программой при нажатии кнопки ПУСК. Разумеется, такие константы, и переменные с начальными значениями следует располагать в таком месте программы, чтобы устройство управления не начало бы выполнять их как команды. Чтобы избежать этой неприятности, эти ячейки рекомендуется размещать в конце программы, после команды с кодом операции 31 (СТОП).

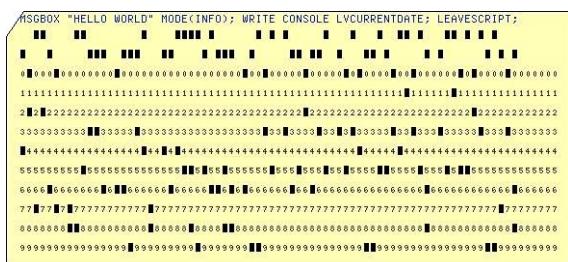
Далее, используя принцип неразличимости команд и чисел, целочисленная константа 1 представлена в виде команды 00 000 000 001 (скоро при изучении внутреннего машинного представления целых чисел Вы поймете, что как машинные слова они совпадают). Этот приём позволил ввести эту константу по кнопке ПУСК, "замаскировав" её в виде команды (обычно так и поступали программисты первых ЭВМ).

Следует обратить внимание и на тот факт, что изначально программист не знает, сколько ячеек в памяти будут занимать команды его программы. Поэтому адреса программы, ссылающиеся на переменные с начальным значением, до завершения программы командой СТОП остаются неизвестными

¹ В первых ЭВМ все ячейки памяти после включения машины обычно имели полностью нулевые значения (очищались), но в машине УМ-3 это не предполагается, так же обстоит дело и в стандарте языка Паскаль, где после порождения (размещения в памяти) переменные имеют неопределённые значения.

(незаполненными), и уже потом, разместив эти переменные в памяти (обычно сразу же вслед за командами программы), следует указать их адреса в тексте программы. В примерах те адреса программы учебной машины, которые заполняются программистом в последнюю очередь, будут выделяться подчёркиванием.

Как видно из приведенного примера, для программиста запись программы состоит из строк, каждая строка снабжается номером (адресом) той ячейки, куда будет помещаться это машинное слово (команда, константа или переменная с начальным значением) при загрузке программы. Вслед за номером задаются все поля команды, затем программист может указать комментарий (разумеется, номера машинных слов, и комментарии *не вводятся* в память ЭВМ, для них нет места в машинном слове). Кроме того, так как числа в памяти неотличимы от команд, то, как уже говорилось, они записываются в программе тоже в виде команд.



Перфокарта для первых ЭВМ

Для ввода программы в ЭВМ первые программисты кодировали все машинные слова в двоичном виде и переносили их на какой-нибудь *носитель данных* для устройства ввода, например, на картонные перфокарты (Punched Card). После этого оставалось снабдить такую, как говорили, *колоду перфокарт*, перфокартой-признаком конца ввода, поместить на устройство ввода и нажать кнопку ПУСК. В нашем случае надо поместить на устройство ввода два массива

машинных слов – саму программу (8 машинных слов + признак конца ввода) и число x (одно машинное слово). Как уже говорилось, первый массив заканчивался специальным признаком конца ввода, так что устройство ввода само знало, сколько машинных слов надо ввести в память по кнопке ПУСК.

3.2.2. Пример 2. Условный оператор

Первое условие, которое надлежит выполнять в математике, – это быть точным, второе – быть ясным и, насколько можно, простым.

Лазар Карно

Составим теперь программу, реализующую условный оператор присваивания. Пусть целочисленная переменная y принимает значение в зависимости от значения вводимой целочисленной переменной x :

$$y := \begin{cases} x+2, & \text{при } x < 2, \\ 2, & \text{при } x = 2, \\ 2 * (x+2), & \text{при } x > 2; \end{cases}$$

В данном и следующих примерах при записи программы на месте кода операции будем для удобства вместо числового номера указывать его мнемоническое обозначение, приведённое в таблице команд 3.1. Разумеется, потом, перед вводом программы в ЭВМ, необходимо будет заменить эти мнемонические обозначения соответствующими им двоичными числами.

Для определения того, является ли значение некоторого числа x больше, меньше или равным константе 2, будет выполняться команда вычитания $x-2$, получая как побочный эффект этой операции в регистре признаке результата ω значение 0 при $x=2$, значение 1 при $x<2$ и значение 2 при $x>2$. При этом сам результат операции вычитания $x-2$ в этой программе не нужен, но по принятому в УМ-3 формату команд указание адреса ячейки для записи результата является обязательным. Для записи таких ненужных значений будем чаще всего использовать ячейку с номером 0. Заметим, что в нашей архитектуре программа специально вводится в память по кнопке ПУСК, начиная с *первой* ячейки, оставляя программисту для подобных целей свободной ячейку с номером ноль. В соответ-

вии с принципом однородности памяти, эта ячейка ничем не отличается от других, то есть, доступна как для записи, так и для чтения данных.¹

Для хранения переменных x и y выделим, например, ячейки 100 и 101 соответственно. Заметим, что программист сам определяет порядок размещения в программе трёх ветвей условного оператора присваивания. В нашем примере будем сначала располагать вторую ветвь (для $x=2$), затем первую ($x<2$), а потом третью ($x>2$). Заметим, что значение $x+2$ понадобится при программировании, как в первой, так и в третьей ветви условного оператора, поэтому более рационально вычисление выражения $x+2$ расположить *перед* вычислением условного оператора, что и сделано в этом примере. На рис. 3.2 приведён текст этой программы.

№	Команда				Комментарий
001	ВВЦ	100	001	000	Read(x)
2	СЛЦ	101	100	011	$y := x+2$
3	ВЧЦ	000	100	011	$\langle 000 \rangle := x-2$; формирование ω
4	УСЛ	005	007	009	Case ω of 0: goto 005; 1: goto 007; 2: goto 009 end
5	ВЫЦ	011	001	000	Write(2)
6	СТОП	000	000	000	Конец работы
7	ВЫЦ	101	001	000	Write(y)
8	СТОП	000	000	000	Конец работы
9	СЛЦ	101	011	101	$y := 2*y = y+y$; Команда СЛЦ быстрее УМЦ
010	БЕЗ	000	007	000	Goto 007
1	00	000	000	002	Целая константа 2

Рис 3.2. Программа второго примера.

Для перехода на три ветви нашего алгоритма использовалась команда условного перехода УСЛ,² предварительно предыдущей командой вычитания ВЧЦ 000 100 011 было определено значение регистра признака результата ω . Второй и третий адрес в команде условного перехода тоже подчеркнуты, так как эти значения становятся известны программисту только после написания первой (для $\omega=0$) и второй (для $\omega=1$) ветвей нашего алгоритма соответственно.

Обратите внимание, что целочисленная константа 2 располагается в таком месте программы (за командой безусловного перехода), где она никогда не будет выполняться как команда. Заметим также, что эта константа неотличима от команды ПЕР 000 000 002 для пересылки содержимого второй ячейки памяти в нулевую ячейку, что и позволило нам записать эту константу в виде такой команды. Именно эта команда и выполнялась бы, если бы такая константа была (случайно) выбрана на регистр команд устройства управления.

Разберёмся теперь, на каком же языке написан второй пример программы для УМ-3. Ясно, что это не язык машины, так как он по определению состоит только из 32-разрядных двоичных чисел, а у нас вместо кодов операции стоят мнемонические обозначения, адреса написаны в десятичной форме записи, да ещё и комментарии присутствуют. Такой язык программирования для первых ЭВМ обычно называли *псевдокодом*, что хорошо отражало его сущность.³ Для того чтобы выполнить программу на псевдокоде, её надо предварительно перевести (или, как теперь принято говорить, *оттранслировать*) на язык машины. Очевидно, что для этого необходимо заменить все мнемонические обозначения соответствующими им числовыми кодами операций, затем перевести все десятичные числа в двоичную систему счисления, а также отбросить номера ячеек из первой колонки и комментарии. Для первых ЭВМ эту работу выполняли сами программисты. В дальнейшем псевдокоды постепенно развивались и превратились в языки низкого уровня – *ассемблеры* (в русскоязычной литературе их сначала называли *автокодами*, подразумевая, что соответствующая программа *авто-*

¹ В некоторых первых реальных ЭВМ этот принцип нарушался: при считывании из этой ячейки всегда возвращался ноль, а запись в ячейку с адресом ноль физически не осуществлялась, на практике такой принцип работы с этой ячейкой часто был удобнее. В частности, в такой машине всегда была константа ноль.

² В языках высокого уровня такой условный переход на три ветви существует, например, в Фортране:

if (<арифметическое выражение>) MetkaNeg, MetkaZero, MetkaPos

³ Считается, что первый псевдокод был создан в 1949 г. Джоном Преспером Эккертом (John Presper Eckert) и Джоном Мок(ч)ли (John William Mauchly) для разрабатываемой ими коммерческой ЭВМ BINAC, которая, впрочем, так и не начала работать.

матически переводила (кодировала) с псевдокода на язык машины). Следующие программы для УМ-3 также будут для удобства писаться не на "чистом" языке машины, а на таком псевдокоде.

3.2.3. Пример 3. Реализация цикла

Если вы не можете объяснить что-то простым языком, то вы не понимаете этого.

Ричард Фейнман

В качестве следующего примера напомним программу для вычисления суммы элементов начального отрезка гармонического ряда:

$$y = \sum_{i=1}^n 1/i$$

Программа должна вводить значение переменной n и выводить в качестве результата работы y . Для хранения переменных n , y и параметра цикла i выделены ячейки 100, 101 и 102 соответственно. На рис. 3.3 приведена возможная программа для решения этой задачи.

№	Команда	Комментарий
001	ВВЦ 100 001 000	Read (n)
2	ВЧВ 101 101 101	$y := y - y = 0.0$
3	ПЕР 102 000 013	$i := 1$
4	ВЧЦ 000 102 100	$\langle 0 \rangle := i - n$; формирование w
5	ПБ 000 011 000	if $i > n$ { $w=2$ } then goto 011
6	ВЕЩ 000 000 102	$\langle 0 \rangle := \text{Real}(i)$
7	ДЕВ 000 014 000	$\langle 0 \rangle := 1.0 / \text{Real}(i)$
8	СЛВ 101 101 000	$y := y + \langle 0 \rangle$
9	СЛЦ 102 102 013	$i := i + 1$
010	БЕЗ 000 004 000	Следующая итерация цикла
1	ВЫВ 101 001 000	Write (y)
2	СТОП 000 000 000	Стоп
3	00 000 000 001	Целая константа 1
4	<1.0>	Вещественная константа 1.0

Рис 3.3. Программа третьего примера.

Сделаем некоторые замечания к этой программе. В этом алгоритме реализуется *цикл с предусловием*, его аналог на Паскале можно записать в виде:

```
y:=0.0; i:=1;
while i<=n do begin y:=y+1.0/i; i:=i+1 end;
```

Таким образом, при вводе значения $n < 1$ тело цикла не будет выполняться ни одного раза, и программа будет выдавать нулевой результат, что не противоречит математическому смыслу поставленной задачи.

Заметим далее, что каждый член ряда является по смыслу задачи вещественным числом, в то время как параметр цикла i – целая величина. В языке машины нет команды деления вещественного числа на целое, поэтому при вычислении слагаемого $1.0/i$ нам пришлось воспользоваться командой ВЕЩ 000 000 102. Эта команда преобразует значение целой переменной i в вещественное значение (заметим, что Паскаль-машина будет делать это автоматически!). Обратите также внимание, что для нашей учебной машины ещё не определен формат представления вещественных чисел, (это будет сделано позже), поэтому в ячейке с адресом 14 стоит пока просто условное обозначение константы 1.0, а не её машинное представление.

3.2.4. Пример 4. Работа с массивами

... время идёт, и за компьютеры садятся те, кто разбирается в них всё меньше и меньше ...

Э. Таненбаум.

«Архитектура компьютера»

Полезно время от времени ставить знак вопроса на вещах, которые тебе давно представляются несомненными.

Бертран Рассел

Пусть требуется написать программу для ввода массива x из 100 вещественных чисел и вычисления суммы всех элементов этого массива:

$$S = \sum_{i=1}^{100} x[i]$$

Сделаем естественное предположение, что длина нашей программы не будет превышать 199 ячеек, и поместим массив x , начиная с 200-ой ячейки памяти. Вещественную переменную S с начальным значением 0.0 и целую переменную n с начальным значением 100 поместим в конце текста программы (после команды СТОП), и будем вводить в память вместе с командами при нажатии кнопки ПУСК. Заметим, что так в программе не нужна ячейка, в которой должна была бы храниться команда обнуления переменной S . Таким образом, в качестве счётчика цикла будет удобнее использовать не целую переменную i , изменяющуюся от 1 до 100, как в приведённой выше формуле, а переменную n с начальным значением 100, которая будет изменяться от 100 до 1. Другими словами, реализуется цикл, который на языке Free Pascal можно было бы записать в виде

```
var S: Single=0.0; n: Longint=100; i: Longint=1;
. . .
repeat S:=S+x[i]; i:=i+1; n:=n-1 until n=0;
```

На рис. 3.4 приведён текст этой программы. Рассматриваемая программа выделяется новым приёмом программирования, она является *сагомодифицирующей программой*, о них говорилось при изучении машины Фон Неймана. Обратим внимание на третью строку программы. Содержащаяся в ней команда изменяет исходный код программы (меняет команду в ячейке с адресом 002) для организации цикла перебора элементов массива.¹ При первом выполнении этой команды она адресуется к первому элементу массива, затем ко второму и т.д. Для перехода от одного элемента массива к следующему модифицируемая команда рассматривается как *целое число*, к которому прибавляется специально подобранная *константа переадресации*. Согласно одному из принципов Фон Неймана, числа и команды в учебной машине неотличимы друг от друга, а, значит, изменяя числовое представление команды, можно изменять и выполняемые ей действия.

№	Команда				Комментарий
001	ВВВ	200	100	000	Read(x); массив x в ячейках 200÷299
2	СЛВ	008	200	008	$S := S+x[1]$
3	СЛЦ	002	002	011	Модификация команды в ячейке 2
4	ВЧЦ	010	010	009	$n := n-1$
5	ПВ	000	002	000	if $n>0$ then goto 002
6	ВЫВ	008	001	000	Write(S)
7	СТОП	000	000	000	Стоп
8	00	000	000	000	Вещественная переменная $S = 0.0$
9	00	000	000	001	Целая константа 1
010	00	000	000	100	Переменная n с начальным значением 100
1	00	000	001	000	Константа переадресации

Рис 3.4. Программа четвёртого примера.

У такого метода программирования есть один существенный недостаток: модификация кода программы внутри её самой повышает сложность программирования, может привести к путанице и вызывать появление ошибок. Заметим также, что такую программу нельзя повторно выполнить, просто

¹ Как уже говорилось, в теории алгоритмов некоторым аналогом такого исполнителя, например, является такая модификация машины Тьюринга, которая в процессе выполнения алгоритма может изменять клетки своей таблицы. Доказана теорема о том, что все задачи, которые может решать такая модифицированная машина Тьюринга, может решить и обычная машина Тьюринга, т.е. такие алгоритмические системы эквивалентны.

передав управление на её первую команду,¹ так как нужно будет предварительно восстановить исходный вид всех модифицированных команд. Кроме того, саомодифицирующуюся программу трудно понимать и вносить в неё изменения. Только представьте себе, что Вам необходимо составить алгоритм для машины Тьюринга, в которой можно изменять команды в клетках таблицы (например, заменить команду движения головки по ленте влево на движение вправо). Настоящий кошмар для современного программиста!

В нашей учебной машине, однако, саомодифицирующаяся программа – это *единственный* способ обработки достаточно больших массивов. В других архитектурах ЭВМ, с которыми Вы познакомитесь несколько позже, есть и иные, более эффективные способы работы с массивами, поэтому метод программирования с модификацией команд в современных компьютерах, как уже говорилось, обычно не используется.

3.3. Формальное описание учебной машины

Весь математический формализм является как бы забором, следуя вдоль которого, слепой может уверенно двигаться в намеченном направлении.

Станіслав Лем. «Сумма технологий»

При описании архитектуры учебной ЭВМ УМ-3 на естественном языке многие вопросы остались нераскрытыми. Что, например, будет происходить после выполнения команды из ячейки с адресом 511? Какое значение после нажатия кнопки ПУСК имеют ячейки, расположенные вне введённого массива машинных слов? Как представляются целые и вещественные числа? Как будет, например, выполняться такая машинная команда ввода массива: BBB 500 100 000?

Для ответа на почти все такие вопросы будет приведено *формальное описание* нашей учебной машины. При этом в качестве метаязыка будет использоваться язык Free Pascal.² Другими словами, будет представлена программа на языке Free Pascal, выполнение которой *моделирует* работу УМ-3, т.е. наша машина, по определению, работает "почти так же", как и эта написанная программа-модель.

Ниже приведена реализация учебной машины на языке Free Pascal:

```
program YM_3;
const
  N = 511;
type
  Address = 0..N;
  Tag = (kom, int, fl);
{В машинном слове может храниться команда,
 целое или вещественное число}
  Komanda = bitpacked record
    KOP: 0..31;
    A1, A2, A3: Address
  end;
  Slovo = packed record
    case Tag of
      kom: (k: Komanda);
      int: (i: LongInt);
      fl: (f: Single)
    end
  end
  Memory = array[0..N] of Slovo;
var
  Mem: Memory;
  S, R1, R2: Slovo; {Регистры ALU}
```

¹ Повторное выполнение находящейся в памяти программы может оказаться весьма полезным, например, при счёте нескольких вариантов задачи с разными входными данными. Так, многие диалоговые программы идут на своё повторное выполнение, задав вопрос "Повторить? (Y/N)".

² Так как учебная ЭВМ является не языком, а исполнителем алгоритмов, то в качестве метаязыка необходимо выбрать не просто формальный язык (например, язык синтаксических диаграмм), а формальный алгоритмический язык, т.е. язык, правильные слова в котором являются записями алгоритмов для некоторого исполнителя.

```

RK: Komanda; {Регистр команд}
RA: Address; {Счётчик адреса}
Om: 0..2; {Регистр w}
Err: Boolean;
begin
{Процедура Input_Program должна вводить текст программы
с устройства ввода в память по кнопке ПУСК}
Input_Program;
Om := 0; Err := False; RA := 1; {Начальная установка регистров}
with RK do
repeat {Основной цикл выполнения команд}
RK := Mem[RA].k;
RA := (RA+1) mod (N+1);
case KOP of {Анализ кода операции}
00: { ПЕР }
begin R1 := Mem[A3]; Mem[A1] := R1 end;
01: { СЛБ }
begin
R1 := Mem[A2]; R2 := Mem[A3]; S.f := R1.f + R2.f;
if S.f = 0.0 then OM := 0 else
if S.f < 0.0 then OM := 1 else OM := 2;
Mem[A1] := S; { Err := ? }
end;
09: { БЕЗ }
RA := A2;
15: { МОД }
begin
R1 := Mem[A2]; R2 := Mem[A3];
if R2.i = 0 then Err := true else begin
S.i := R1.i mod R2.i; Mem[A1] := S;
if S.i = 0 then OM := 0 else
if S.i < 0 then OM := 1 else OM := 2;
end
end;
31: { СТОП } ;
{ Реализация остальных кодов операций }
else
Err := true;
end; { case }
until Err or (KOP = 31)
end.

```

Прокомментируем эту программу, описывающую работу учебной машины. Отметим сначала, что некоторая трудность возникает при моделировании начального ввода программы в память учебной машины при нажатии кнопки ПУСК. В нашей модели для задания такого начального ввода использован вызов процедуры с именем `Input_Program`, описание этой процедуры не приводится.¹

Для хранения машинных слов учебной машины описан тип данных `Slovo`, который является записью с вариантами языка `Free Pascal`. В такой записи на одном и том же месте памяти могут располагаться команды, длинные (32-битные) целые числа или же 32-битные вещественные числа стандартного типа `Single` (более привычный для программистов тип `Real` языка `Free Pascal` здесь не подходит, потому что имеет длину 64 бита, а не 32 бита, как нам нужно). Таким образом, этот тип данных позволяет нам реализовать в программе на Паскале неразличимость представления команд, целых и вещественных чисел учебной машины.

Эта программа-модель ведёт себя почти так же, как учебная машина. Одно из немногих мест, где это поведение расходится, показано в тексте программы комментариями с вопросительным знаком, например, при реализации команды сложения вещественных чисел. Программа на Паскале при пере-

¹ В архитектуре ЭВМ принято выделять центральную часть, куда входит основная (оперативная) память и центральный процессор. Вся остальная аппаратура ЭВМ относится к так называемой периферии (периферийным устройствам). Таким образом, модель на Паскале формально описывает только центральную часть учебной машины, но не её периферию.

полнении (когда результат сложения не помещается в переменную S) просто производит аварийное завершение программы, а учебная машина сначала присваивает регистру Err значение 1, а затем останавливает выполнение программы.

Заметим, что наше формальное описание отвечает и на вопрос о том, как в учебной машине представляются целые и вещественные числа: точно так же, как в переменных соответствующих типов программы на языке Free Pascal. Это представление будет подробно изучено в этой книге несколько позже.

В приведенной модели реализовано выполнение только некоторых типичных команд из языка учебной машины. Реализацию остальных команд Вы легко можете выполнить сами.

Обратите также внимание, как в нашей модели вычисляется адрес следующей выполняемой команды: $RA := (RA + 1) \bmod (N + 1)$. Такое правило перехода к следующей по порядку команде программы позволяет считать, что память учебной машины как бы замкнута в кольцо: после выполнения команды из ячейки с адресом 511 (если это не команда перехода) следующая команда будет выполняться из ячейки с адресом ноль. Такая организация памяти типична для многих ЭВМ.

Вопросы и упражнения

1. Что такое код операции ?
2. Для чего необходим регистр признака результата ?
3. Объясните, как в нашей учебной машине должна выполняться такая команда ввода массива вещественных чисел `BVB 100 500 000`. Напишите соответствующую ветвь в формальном описании УМ-3 для реализации команды ввода вещественных чисел.
4. Реализуйте в модели на Паскале выполнение команд ввода/вывода учебной машины.
5. Почему при программировании на языке машины в памяти не существуют константы, как, например, в языке Паскаль ?
6. Что такое переменная с начальным значением и как такую переменную разместить в памяти учебной ЭВМ ?
7. Что такое псевдокод ?
8. Объясните, почему для машины УМ-3 при решении некоторой задачи нельзя сделать такое распределение памяти: "Пусть массив X располагается в ячейках с адресами от 100 до 199, а константа $n=100$ – в ячейке с адресом 200" ?
9. Что такое самомодифицирующаяся программа ?
10. Почему в учебной машине УМ-3 обработка больших массивов возможна только при помощи самомодифицирующейся программы ?
11. Напишите программу для УМ-3, она должна вводить два вещественных массива X и Y длины $N=200$ и вычислять сумму S

$$S := \sum_{i=1}^N X[i] * Y[N - i + 1]$$