## 1.3. MIX

В этой книге ОЧЕНЬ ЧАСТО встречаются упоминания о внутреннем машинном языке компьютера. Причем использовать мы будем гипотетический компьютер под названием "MIX". MIX практически ничем не отличается от любого другого компьютера 60–70-х годов, только он, вероятно, более изящен. При разработке языка компьютера MIX преследовалась цель сделать его достаточно мощным, позволяющим для большинства алгоритмов писать короткие программы, и в то же время достаточно простым, чтобы его операции можно было легко запомнить

Настоятельно рекомендую читателю внимательно изучить этот раздел, так как язык MIX используется в очень многих разделах книги. Отбросьте все сомнения по поводу того, стоит ли изучать машинный язык. Автор однажды понял, что нет ничего необычного в том, чтобы в течение одной недели заниматься написанием программ на нескольких различных машинных языках! Каждый, кто серьезно интересуется компьютерами, должен рано или поздно изучить по крайней мере один машинный язык. При разработке MIX были специально сохранены основные черты реальных компьютеров, чтобы его характеристики можно было легко понять и усвоить.

*Тем не менее нужно признать, что в настоящее время MIX полностью устарел. Поэтому в последующих изданиях данной книги он будет заменен новым компьютером под названием MMIX (номер 2009). MMIX будет представлять собой так называемый компьютер с усеченным набором команд (RISC), который выполняет арифметические операции над 64-битовыми словами. Будучи более изящным, чем MIX, он станет аналогом тех компьютеров, которые в 90-х годах завоевали ключевые позиции на рынке компьютерной техники.*

*Полный и повсеместный переход в данной книге от MIX к MMIX отнимет много времени, поэтому огромная просьба к добровольцам — окажите посильную помощь в этом деле. Между тем, автор надеется, что читатели согласятся подождать еще несколько лет и пока удовлетворятся устаревшей архитектурой MIX, которая все еще заслуживает внимания, поскольку обеспечивает среду для дальнейших разработок.*

### 1.3.1. Описание MIX

MIX — это первый в мире полиненасыщенный компьютер*. Как и у большинства компьютеров, у него есть идентификационный номер — 1009. Этот номер получен следующим образом: взяли 16 очень похожих на MIX реальных компьютеров, на которых можно легко имитировать MIX, а затем нашли среднее значение их номеров, взятых с равными весовыми коэффициентами:

$$\lfloor(360 + 650 + 709 + 7070 + \mathrm{U3} + \mathrm{SS80} + 1107 + 1604 + \mathrm{G20} + \mathrm{B220} + \mathrm{S2000} + 920 + 601 + \mathrm{H800} + \mathrm{PDP\text{-}4} + \mathrm{II})/16\rfloor = 1009. \quad (1)$$

Это же число можно получить значительно проще — прочитать слово MIX как римское число.

Характерная особенность компьютера MIX состоит в том, что он является двоичным и десятичным одновременно. *Программисты MIX на самом деле даже не знают,*

* Аналогия с полиненасыщенными жирами, широко рекламируемыми сегодня по всему миру. — *Прим. перев.*

*компьютер с какой арифметикой они программируют — с двоичной или десятичной* Поэтому алгоритмы, написанные для MIX, с небольшими изменениями можно использовать на любом из этих типов компьютеров и MIX можно легко имитировать на этих компьютерах. Те программисты, которые привыкли к двоичному компьютеру, могут считать MIX двоичным, а те, которые привыкли к десятичному, могут считать MIX десятичным. Программисты же с другой планеты могут считать MIX троичным компьютером.

**Слова.** Основной единицей информации является *байт*. Каждый байт должен принимать по меньшей мере 64 различных значения, но реальный объем содержащейся в байте информации может быть *разным*. Таким образом, в одном байте может содержаться любое число от 0 до 63 включительно. Более того, в каждом байте может содержаться *максимум* 100 различных значений. Следовательно, в двоичном компьютере байт должен состоять из шести разрядов, а в десятичном — из двух*.

Программы на языке MIX должны быть написаны так, чтобы в байте содержалось *не более* 64 значений. Так, для представления числа 80 мы всегда будем выделять два байта, хотя в десятичном компьютере для этого достаточно одного байта. *Алгоритм на языке MIX должен работать правильно независимо от размера байта.* Конечно, вполне возможно написать программы, зависящие от размера байта, но в данной книге такие действия осуждаются и допустимыми считаются только те программы, которые дают правильный результат независимо от размера байта Обычно придерживаться этого основного правила совсем нетрудно, и, таким образом, мы обнаружим, что программирование на десятичном компьютере не особенно отличается от программирования на двоичном.

С помощью двух соседних байтов можно выразить числа от 0 до 4 095.

С помощью трех соседних байтов можно выразить числа от 0 до 262 143.

С помощью четырех соседних байтов можно выразить числа от 0 до 16 777 215.

С помощью пяти соседних байтов можно выразить числа от 0 до 1 073 741 823.

*Машинное слово состоит из пяти байтов и знака.* Знак может принимать только два значения: "+" и "−".

**Регистры.** В компьютере MIX всего девять регистров (рис. 13).

Регистр A (аккумулятор) содержит 5 байт и знак.

Регистр X (расширение аккумулятора) тоже содержит 5 байт и знак

В регистрах I (индексных регистрах) I1, I2, I3, I4, I5 и I6 содержится по два байта и знак.

Регистр J (адрес перехода) содержит два байта; его знак — всегда "+".

Для обозначения регистра компьютера MIX будем использовать в качестве приставки к имени регистра строчную букву "r". Таким образом, "rA" обозначает "регистр A".

Регистр A имеет много применений, особенно часто он используется при выполнении арифметических действий и операций над данными. Регистр X используется

* Приблизительно с 1975 года слово "байт" стало обозначать последовательность, состоящую ровно из восьми двоичных цифр, что позволяет представлять числа от 0 до 255 Поэтому размеры байтов реальных компьютеров больше, чем размеры байтов гипотетической машины MIX И в самом деле, старомодные байты компьютера MIX только чуть-чуть больше, чем половина байта реального компьютера Говоря о байтах применительно к MIX, мы будем придерживаться прежнего значения этого слова, вновь возвращаясь к тем дням, когда понятие байта еще не было так стандартизовано

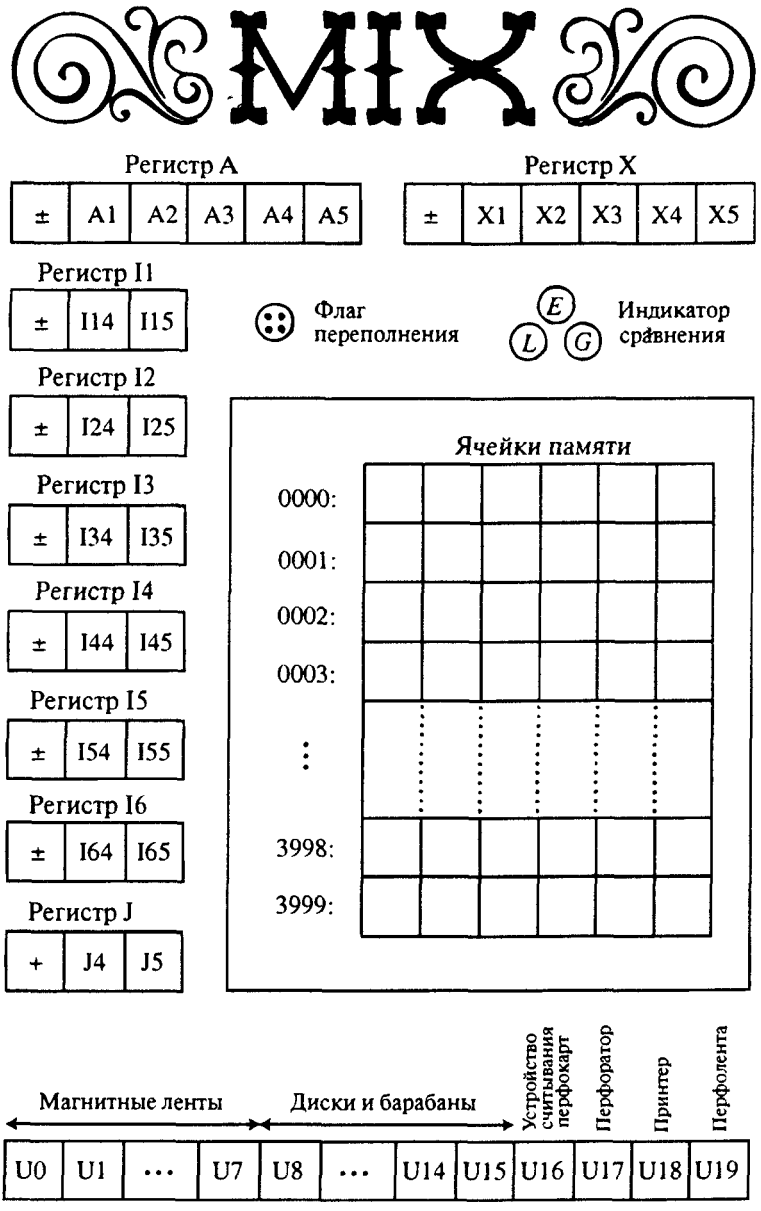



**Рис. 13.** Компьютер `MIX`.

для расширения вправо регистра A и вместе с rA — для хранения 10 байт произведения или делимого. Он может применяться и для хранения информации, сдвинутой вправо из rA. Индексные регистры rI1, rI2, rI3, rI4, rI5 и rI6 используются, главным образом, как счетчики и для ссылок на различные адреса памяти. В регистре J всегда хранится адрес команды, которая следует за последней операцией "перехода"; этот регистр используется, главным образом, для вызова подпрограмм.

Помимо регистров, компьютер MIX содержит следующие элементы:

*триггер переполнения* (один бит, который может принимать значение "нуль" или "единица");

*флаг сравнения* (принимающий одно из трех значений. LESS (меньше), EQUAL (равно) и GREATER (больше));

*память* (4 000 слов, каждое из которых состоит из 5 байт и знака);

*устройства ввода-вывода* (перфокарты, ленты, диски и т. д.).

**Структура машинного слова.** 5 байт и знак, из которых состоит машинное слово, нумеруются следующим образом:

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ± | Байт | Байт | Байт | Байт | Байт |

(2)

Большинство команд таковы, что программист может при желании использовать только часть слова. В подобных случаях можно задать нестандартную "спецификацию поля". При этом допустимо использовать поля, которые являются соседними в машинном слове; они обозначаются в виде (L·R), где L — номер левой, а R — номер правой части поля. Приведем примеры спецификации полей:

(0:0): только знак;
(0:2): знак и первые два байта;
(0:5): целое слово; это самая распространенная спецификация поля;
(1:5): все слово, кроме знака;
(4:4): только четвертый байт;
(4:5): два младших значащих байта.

Использование спецификации поля несколько меняется от команды к команде; при рассмотрении каждой команды мы поговорим об этом более подробно. На самом деле каждая спецификация поля (L:R) представляется внутри компьютера одним числом — 8L + R; заметим, что это число легко помещается в одном байте

**Формат команды.** Машинные слова, используемые как команды, имеют следующий формат:

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| ± | A | A | I | F | C |

(3)

Крайний байт справа, C, — это *код операции*, который указывает, какая операция должна быть выполнена. Например, C = 8 определяет операцию LDA*, "загрузить регистр A".

Байт F определяет *модификацию* кода операции. Обычно это спецификация поля (L:R) = 8L+R. Например, если C = 8 и F = 11, то операцией будет "загрузить в регистр A поле (1:3)" Иногда F используется для других целей Например, для команд ввода-вывода F — это номер соответствующего входного или выходного устройства.

* LDA — сокращение от "load the A register" — *Прим. перев.*

Левая часть команды, ±AA, определяет *адрес*. (Обратите внимание, что знак является частью адреса.) Поле I, которое следует за адресом, — это *спецификация индекса*, которую можно применять для модификации фактического адреса. Если I = 0, то адрес ±AA используется без изменений. В противном случае в поле I должно содержаться число $i$ от 1 до 6, и тогда содержимое индексного регистра I$i$ алгебраически добавляется к ±AA перед выполнением команды. Полученный результат используется как адрес. Процесс индексирования выполняется для *каждой* команды. Обозначим буквой "M" адрес, получаемый после каждой операции индексирования. (Если после добавления содержимого индексного регистра к адресу ±AA получается результат, который не помещается в двух байтах, то значение M будет неопределенным.)

Для большинства команд M указывает на ячейку памяти. Термины "ячейка памяти" и "адрес ячейки памяти" в этой книге почти всегда являются эквивалентными. Предполагается, что имеется 4 000 ячеек памяти с номерами от 0 до 3 999 Поэтому адрес каждой ячейки можно представить с помощью двух байтов. Для каждой команды, в которой M обозначает ячейку памяти, должно выполняться неравенство $0 \le M \le 3999$. В этом случае запись `CONTENTS(M)` будет обозначать величину, которая хранится в ячейке с адресом M.

Для некоторых команд "адрес" M имеет несколько иной смысл; он может даже быть отрицательным. Так, например, одна команда добавляет M к индексному регистру и при этом принимается во внимание знак M.

**Форма записи.** Чтобы сделать команды более читабельными, для обозначения команды типа (3) будет использоваться следующая форма записи:

```
OP    ADDRESS,I(F) .                    (4)
```

Здесь `OP` — символическое имя кода операции (часть C) команды, `ADDRESS` — часть ±AA, `I` и `F` — поля I и F соответственно.

Если I равно нулю, то запись "`,I`" опускается. Если F — *стандартная* F-спецификация для данной команды, то запись "`(F)`" использовать не нужно. Практически для всех команд стандартной F-спецификацией является (0:5), что соответствует целому слову. Если же стандартной является другая спецификация F, то она будет упомянута особо при описании конкретной команды

Например, команда загрузки числа в аккумулятор называется `LDA` и ее код операции — 8. Имеем следующее.

| Условное представление | Реальное представление команды в цифровом виде | | | | |
|---|---|---|---|---|---|
| `LDA 2000,2(0:3)` | + | 2000 | 2 | 3 | 8 |
| `LDA 2000,2(1:3)` | + | 2000 | 2 | 11 | 8 |
| `LDA 2000(1:3)` | + | 2000 | 0 | 11 | 8 |
| `LDA 2000` | + | 2000 | 0 | 5 | 8 |
| `LDA -2000,4` | - | 2000 | 4 | 5 | 8 |

(5)

Команда "`LDA 2000,2(0:3)`" читается следующим образом: "Загрузить в регистр A содержимое ячейки 2000 с индексом 2, поле "нуль–три".

Для представлений цифрового содержимого слова MIX будем использовать клеточную запись, как было сделано выше. Обратите внимание, что в слове

| + | 2000 | 2 | 3 | 8 |
|---|---|---|---|---|

число +2000 занимает два соседних байта и знак. Заметим, что реальное содержимое байта (1:1) и байта (2:2) будет меняться при переходе от одного компьютера MIX к другому, так как меняется размер байта. В следующем примере подобной записи

| - | 10000 | 3000 |
|---|---|---|

представлено слово, состоящее из двух полей. В первом поле, содержащем три байта и знак, находится число −10000, а во втором поле, размером два байта, — число 3000. Когда слово разбито на несколько полей (т. е. содержит более одного поля), говорят, что оно "упаковано".

**Описания команд.** В замечаниях, следующих за записью (3) (см. выше), были определены величины M, F и C для любого слова, используемого в качестве команды. А теперь определим действия, соответствующие каждой команде.

### Команды загрузки

• LDA (load A — загрузить A). C = 8; F = поле.
Заданное поле CONTENTS(M) заменяет предыдущее содержимое регистра A.

Во всех операциях, в которых в качестве входного значения используется частичное поле, знак учитывается, если он является частью этого поля; в противном случае берется знак "+". По мере загрузки поле сдвигается в правую часть регистра.

*Примеры.* Если F — стандартная спецификация поля (0:5), то все содержимое ячейки M копируется в rA. Если F — спецификация (1:5), то абсолютное значение CONTENTS(M) загружается со знаком "+". Если в M содержится *команда* и F — это спецификация (0:2), поле "±AA" загружается как

| ± | 0 | 0 | 0 | A | A |
|---|---|---|---|---|---|

.

Предположим, в ячейке с адресом 2000 содержится слово

| - | 80 | 3 | 5 | 4 |
|---|---|---|---|---|

; (6)

тогда, загружая различные частичные поля, получим следующие результаты.

| Команда | | Последующее содержимое rA | | | | |
|---|---|---|---|---|---|---|
| LDA | 2000 | - | 80 | | 3 | 5 | 4 |
| LDA | 2000(1:5) | + | 80 | | 3 | 5 | 4 |
| LDA | 2000(3:5) | + | 0 | 0 | 3 | 5 | 4 |
| LDA | 2000(0:3) | - | 0 | 0 | 80 | | 3 |
| LDA | 2000(4:4) | + | 0 | 0 | 0 | 0 | 5 |
| LDA | 2000(0:0) | - | 0 | 0 | 0 | 0 | 0 |
| LDA | 2000(1:1) | + | 0 | 0 | 0 | 0 | ? |

(В последнем примере значение частично не определено, так как размер байта переменный.)

• LDX (load X — загрузить X). C = 15; F = поле.
Эта команда идентична LDA, за исключением того, что вместо rA загружается rX.

• LD$i$ (load $i$ — загрузить $i$). C = 8 + $i$; F = поле.
Эта команда идентична LDA, только вместо rA загружается r$I i$. Индексный регистр содержит только два байта (а не пять) и знак; байты 1, 2, 3 всегда считаются нулевыми. Поэтому, если установить для байтов 1, 2 или 3 любые значения. не равные нулю, то команда LD$i$ станет неопределенной.

В описаниях всех команд "$i$" обозначает целое число, $1 \le i \le 6$. Таким образом. LD$i$ обозначает шесть различных команд: LD1, LD2, ..., LD6.

• LDAN (load A negative — загрузить в A с обратным знаком). C = 16; F = поле.
• LDXN (load X negative — загрузить в X с обратным знаком). C = 23; F = поле.
• LD$i$N (load $i$ negative — загрузить в $i$ с обратным знаком). C = 16 + $i$; F = поле.
Эти восемь команд идентичны командам LDA, LDX и LD$i$ соответственно. но только величины загружаются с *обратным* знаком.

### Команды записи в память

• STA (store A — записать A). C = 24; F = поле.
Часть содержимого rA заменяет поле CONTENTS(M), которое указано в F Другие части CONTENTS(M) остаются неизменными.

Для операции *записи в память* поле F имеет противоположный смысл по сравнению с операцией *загрузки*. Нужное количество байтов в поле, взятом из правой части регистра, в случае необходимости сдвигается *влево*, а затем помещается в соответствующее поле CONTENTS(M). Знак меняется только тогда, когда он является частью поля. Содержимое регистра также остается без изменений.

*Примеры.* Предположим, в ячейке 2000 содержится

| - | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|

а в регистре A -

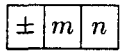
| + | 6 | 7 | 8 | 9 | 0 |
|---|---|---|---|---|---|

Тогда получаем следующее.

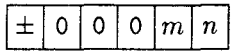
| Команда | Содержимое ячейки 2000 после выполнения команды | | | | | |
|---|---|---|---|---|---|---|
| STA 2000 | + | 6 | 7 | 8 | 9 | 0 |
| STA 2000(1:5) | - | 6 | 7 | 8 | 9 | 0 |
| STA 2000(5:5) | - | 1 | 2 | 3 | 4 | 0 |
| STA 2000(2:2) | - | 1 | 0 | 3 | 4 | 5 |
| STA 2000(2:3) | - | 1 | 9 | 0 | 4 | 5 |
| STA 2000(0:1) | + | 0 | 2 | 3 | 4 | 5 |

• STX (store X — записать X) C = 31: F = поле.
Идентична команде STA, но только сохраняется содержимое rX, а не rA.

• STi (store $i$ — записать $i$). C = 24 + $i$; F = поле.
Идентична команде STA, за исключением того, что сохраняется содержимое rI$i$, а не rA. Байты 1, 2 и 3 индексного регистра являются нулевыми, поэтому, если в rI1 содержится



значит, в нем находится



• STJ (store J — сохранить J). C = 32; F = поле.
Идентична команде STi, но только сохраняется rJ и знаком всегда является "+".

*Для команды STJ стандартной спецификацией поля для F является* (0:2), *а не* (0:5). И это естественно, так как STJ почти всегда выполняет запись в адресное поле команды.

• STZ (store zero — записать нуль). C = 33; F = поле.
Идентична команде STA, но только сохраняется нуль со знаком "+". Другими словами, заданное поле CONTENTS(M) обнуляется.

**Арифметические команды.** При выполнении операций сложения, вычитания, умножения и деления допускается спецификация поля. Спецификацию поля "(0:6)" можно использовать для указания операции с плавающей точкой (см. раздел 4.2). Но лишь в некоторых программах для MIX мы будем пользоваться этой возможностью, так как, в первую очередь, нас будут интересовать целочисленные алгоритмы.

Стандартной спецификацией поля, как обычно, является (0:5). Остальные поля имеют такой же смысл, как при выполнении команды LDA. Буквой "V" будем обозначать заданное поле CONTENTS(M). Таким образом, V — это значение, которое должно быть загружено в регистр A, если кодом операции является LDA.

• ADD (сложение). C = 1; F = поле.
V добавляется к rA. Если абсолютное значение результата слишком велико для того, чтобы поместиться в регистре A, то для триггера переполнения устанавливается значение 1, а в rA остается значение, которое выглядит так, как будто "1" перенесено в другой регистр слева от rA. (В противном случае значение для триггера переполнения не меняется.) Если результат равен нулю, то знак регистра rA остается неизменным.

*Пример.* Последовательность команд, приведенных ниже, служит для вычисления суммы значений, находящихся в пяти байтах регистра A.

```
STA  2000
LDA  2000(5:5)
ADD  2000(4:4)
ADD  2000(3:3)
ADD  2000(2:2)
ADD  2000(1:1)
```

Иногда эту операцию называют сложением наискосок.

В одних компьютерах MIX переполнение происходит, а в других — нет, что связано с разницей в размерах байта. Мы не говорили, что переполнение обязательно произойдет, если результат превысит число 1 073 741 823; переполнение случается,

когда абсолютное значение результата не помещается в пяти байтах (в зависимости от размера байта). Тем не менее можно написать программы, которые правильно работают и дают одинаковые окончательные результаты независимо от размера байта.

• SUB (вычитание). C = 2; F = поле.
V вычитается из rA. (Эквивалентна команде ADD, только вместо V берется −V.)

• MUL (умножение). C = 3; F = поле.
Занимающее 10 байт произведение V и значения из rA заменяет содержимое регистров A и X. В качестве знаков rA и rX устанавливается алгебраический знак произведения (а именно — "+", если знаки V и rA были одинаковы, и "−", если они были различны).

• DIV (деление). C = 4; F = поле.
Значение из rA и rX, рассматриваемое как 10-байтовое число rAX со знаком rA, делится на значение V. Если V = 0 или если абсолютная величина частного не помещается в пяти байтах (это эквивалентно условию $|\mathrm{rA}| \geq |\mathrm{V}|$), то регистры A и X заполняются неопределенной информацией и для триггера переполнения устанавливается значение 1. В противном случае частное $\pm\lfloor |\mathrm{rAX}/\mathrm{V}| \rfloor$ помещается в rA, а остаток $\pm(|\mathrm{rAX}| \bmod |\mathrm{V}|)$ — в rX. После выполнения операции знаком rA становится алгебраический знак частного (а именно — "+", если знаки V и rA были одинаковы, и "−", если они были различны). Знаком rX после выполнения операции будет тот знак, который был у rA до ее выполнения.

*Примеры арифметических команд.* В большинстве случаев арифметические действия выполняются только со словами MIX, которые являются одинарными пятибайтовыми числами, не упакованными в нескольких полях. Тем не менее можно выполнять арифметические операции и над упакованными словами MIX, если принять некоторые меры предосторожности. Для этого следует внимательно изучить приведенные ниже примеры. (Как и раньше, ? обозначает неопределенное значение.)

| | | | | | | |
|---|---|---|---|---|---|---|
| | + | 1234 | 1 | 150 | | rA до операции |
| | + | 100 | 5 | 50 | | Ячейка 1000 |
| ADD 1000 | + | 1334 | 6 | 200 | | rA после операции |

| | | | | | | |
|---|---|---|---|---|---|---|
| | − | 1234 | 0 | 0 | 9 | rA до операции |
| | − | 2000 | 150 | | 0 | Ячейка 1000 |
| SUB 1000 | + | 766 | 149 | | ? | rA после операции |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | + | 1 | 1 | 1 | 1 | 1 | rA до операции |
| | + | 1 | 1 | 1 | 1 | 1 | Ячейка 1000 |
| MUL 1000 | + | 0 | 1 | 2 | 3 | 4 | rA после операции |
| | + | 5 | 4 | 3 | 2 | 1 | rX после операции |

| | | | | | | |
|---|---|---|---|---|---|---|
| | - | | | | 112 | rA до операции |
| | ? | 2 | ? | ? | ? | ? | Ячейка 1000 |
| MUL 1000(1:1) | - | | | | 0 | rA после операции |
| | - | | | | 224 | rX после операции |

| | | | | | | |
|---|---|---|---|---|---|---|
| | - | 50 | 0 | 112 | 4 | rA до операции |
| | - | 2 | 0 | 0 | 0 | 0 | Ячейка 1000 |
| MUL 1000 | + | 100 | 0 | 224 | rA после операции |
| | + | 8 | 0 | 0 | 0 | 0 | rX после операции |

| | | | | | | |
|---|---|---|---|---|---|---|
| | + | | | | 0 | rA до операции |
| | ? | | | | 17 | rX до операции |
| | + | | | | 3 | Ячейка 1000 |
| DIV 1000 | + | | | | 5 | rA после операции |
| | + | | | | 2 | rX после операции |

| | | | | | | |
|---|---|---|---|---|---|---|
| | - | | | | 0 | rA до операции |
| | + | 1235 | 0 | 3 | 1 | rX до операции |
| | - | 0 | 0 | 0 | 2 | 0 | Ячейка 1000 |
| DIV 1000 | + | 0 | 617 | ? | ? | rA после операции |
| | - | 0 | 0 | 0 | ? | 1 | rX после операции |

(Эти примеры были подготовлены с учетом той точки зрения, что лучше дать трудное, но полное описание, чем простое, но неполное.)

**Команды операций с адресами.** В следующих операциях "адрес" M (возможно, индексированный) используется как число со знаком, а не как адрес ячейки памяти.

• `ENTA` (enter A). C = 48; F = 2.
Величина M загружается в rA. Это действие эквивалентно команде "`LDA`" для загрузки из памяти слова, содержащего число M со знаком. Если M = 0, то загружается знак команды.

*Примеры.* "`ENTA 0`" обнуляет rA и устанавливает для него знак "+". "`ENTA 0,1`" заносит в rA текущее содержимое индексного регистра 1, только −0 меняется на +0. Действие команды "`ENTA -0,1`" аналогично, только +0 меняется на −0.

• `ENTX` (enter X — ввести X). C = 55; F = 2.
• `ENT`$i$ (enter $i$ — ввести $i$). C = 48 + $i$; F = 2.
Эти команды аналогичны `ENTA`, но только загружается соответствующий регистр.

• `ENNA` (enter negative A — ввести A с обратным знаком). C = 48; F = 3.
• `ENNX` (enter negative X — ввести X с обратным знаком). C = 55; F = 3.

• ENN$i$ (enter negative $i$ — ввести $i$ с обратным знаком). C = 48 + $i$; F = 3.
Эти команды идентичны ENTA, ENTX и ENT$i$, но в данном случае при загрузке знак меняется на противоположный.

*Пример.* Команда "ENN3 0,3" меняет на противоположный знак содержимого регистра rI3, хотя −0 так и остается −0.

• INCA (increase A — увеличить A). C = 48; F = 0.
К содержимому регистра rA добавляется величина M; это эквивалент команды ADD для добавления из памяти слова, содержащего величину M. Здесь также возможно переполнение, которое обрабатывается, как и в случае команды ADD.

*Пример.* Команда "INCA 1" увеличивает содержимое rA на единицу.

• INCX (increase X — увеличить X). C = 55; F = 0.
Величина M добавляется к содержимому rX. Если происходит переполнение, то оно обрабатывается так же, как в случае команды ADD, только вместо rA используется rX. Действие этой команды никогда не затрагивает регистр A.

• INC$i$ (increase $i$ — увеличить $i$). $C = 48 + i$; F = 0.
К содержимому rI$i$ добавляется величина M. Переполнения быть не должно; если M + rI$i$ не помещается в двух байтах, то результат команды считается неопределенным.

• DECA (decrease A — уменьшить A). C = 48; F = 1.
• DECX (decrease X — уменьшить X). C = 55; F = 1.
• DEC$i$ (decrease $i$ — уменьшить $i$). C = 48 + $i$; F = 1.
Эти восемь команд аналогичны INCA, INCX и INC$i$ соответственно, но только M не добавляется к содержимому регистра, а вычитается из него.

Обратите внимание, что для команд ENTA, ENNA, INCA и DECA используется один и тот же код операции C; поле F используется для того, чтобы можно было отличить одну операцию от другой.

**Команды сравнения.** При выполнении всех команд сравнения MIX сравнивается величина, содержащаяся в регистре, с величиной, содержащейся в памяти. Затем для флага сравнения устанавливается значение LESS (меньше), EQUAL (равно) или GREATER (больше) в зависимости от того, будет ли содержащееся в регистре значение *меньше, равно или больше величины, содержащейся в ячейке памяти.* При этом нуль со знаком "−" считается *равным* нулю со знаком "+".

• CMPA (compare A — сравнить A). C = 56; F = поле.
Заданное поле rA сравнивается с *тем же* полем CONTENTS(M). Если F не содержит бит знака, то оба поля считаются неотрицательными; в противном случае сравнение выполняется с учетом знака. (Равенство всегда будет результатом сравнения, если F — это (0:0), так как нуль со знаком "−" равен нулю со знаком "+".)

• CMPX (compare X — сравнить X). C = 63; F = поле.
Эта команда аналогична CMPA.

• CMP$i$ (compare $i$ — сравнить $i$). C = 56 + $i$; F = *поле*.
Аналог CMPA. Байты 1, 2 и 3 индексного регистра при сравнении считаются нулевыми. (Поэтому, если F = (1:2), результатом сравнения не может быть знак GREATER (больше).)

**Команды перехода.** Команды обычно выполняются последовательно. Другими словами, команда, которая выполняется после команды из ячейки P, обычно находится в ячейке P + 1. Но команды "перехода" позволяют нарушить этот последовательный ход выполнения. При выполнении типичной команды перехода в регистр J заносится адрес следующей команды (т. е. команды, которая оказалась бы следующей, не будь перехода). Затем в случае необходимости программист сможет использовать "адрес J" для определения адресного поля другой команды, чтобы вернуться к первоначальному месту программы. Содержимое регистра J меняется при каждом переходе в программе, за исключением случая, когда используется команда перехода JSJ. Если перехода нет, содержимое этого регистра измениться никак не может.

• JMP (jump — перейти). C = 39; F = 0.
Команда безусловного перехода: следующая команда выбирается из ячейки M.

• JSJ (jump, save J — перейти, сохранить J). C = 39; F = 1.
Эта команда идентична JMP, но только содержимое rJ не меняется.

• JOV (jump on overflow — перейти при переполнении). C = 39; F = 2.
Если для флага переполнения установлено значение 1, то он переключается в положение 0 и выполняется команда JMP; в противном случае ничего не происходит.

• JNOV (jump on no overflow — перейти, если нет переполнения). C = 39; F = 3.
Если для флага переполнения установлено значение 0, то выполняется команда JMP; в противном случае ничего не происходит.

• JL, JE, JG, JGE, JNE, JLE (jump on less, equal, greater, greater-or-equal, unequal, less-or-equal — перейти, если меньше, равно, больше, больше или равно, не равно, меньше или равно). C = 39; F = 4, 5, 6, 7, 8, 9 соответственно.
Переход осуществляется в случае, если для флага сравнения установлено указанное значение. Например, по команде JNE переход будет выполнен, если значением флага сравнения является LESS (меньше) или GREATER (больше). Заметим, что эти команды не изменяют значение самого флага сравнения.

• JAN, JAZ, JAP, JANN, JANZ, JANP (jump A negative, zero, positive, nonnegative, nonzero, nonpositive — перейти, если в регистре A отрицательное значение, нулевое, положительное, ненулевое, неположительное). C = 40; F = 0, 1, 2, 3, 4, 5 соответственно.
Если содержимое rA удовлетворяет заданному условию, то выполняется команда JMP, в противном случае ничего не происходит. "Положительным" является значение, которое *больше* нуля (но не нуль), а "неположительным" — наоборот, нуль или отрицательное значение.

• JXN, JXZ, JXP, JXNN, JXNZ, JXNP (jump X negative, zero, positive, nonnegative, nonzero, nonpositive — перейти, если в регистре X отрицательное значение, нуль, положительное, неотрицательное, ненулевое, неположительное). C = 47; F = 0, 1, 2, 3, 4, 5 соответственно.

• J*i*N, J*i*Z, J*i*P, J*i*NN, J*i*NZ, J*i*NP (jump $i$ negative, zero, positive, nonnegative, nonzero, nonpositive — перейти, если в индексном регистре $i$ — отрицательное значение, нуль, положительное, неотрицательное, ненулевое, неположительное). C = 40 + $i$; F = 0, 1, 2, 3, 4, 5 соответственно. Это аналоги соответствующих команд для rA.

### Другие команды

• SLA, SRA, SLAX, SRAX, SLC, SRC (shift left A (сдвинуть A влево), shift right A (сдвинуть A вправо), shift left AX (сдвинуть AX влево), shift right AX (сдвинуть AX вправо), shift left AX circularly (циклический сдвиг AX влево), shift right AX circularly (циклический сдвиг AX вправо)). C = 6; F = 0, 1, 2, 3, 4, 5 соответственно.

Это команды "сдвига", где M обозначает число байтов компьютера MIX, которые нужно сдвинуть вправо или влево; M должно быть неотрицательным. Команды SLA и SRA не оказывают влияния на содержимое rX; остальные команды сдвига оказывают такое действие на оба регистра A и X, как будто это один 10-байтовый регистр. При выполнении команд SLA, SRA, SLAX и SRAX в регистр с одной стороны входят нули, а с другой стороны исчезает информация из сдвинутых байтов. Команды SLC и SRC вызывают "циклический" сдвиг, при котором байты, "исчезнувшие" с одной стороны, снова входят в регистр с другой стороны. В циклическом сдвиге участвуют оба регистра — rA и rX. Заметим, что ни одна из команд сдвига никак не влияет на знаки регистров A и X.

*Примеры*

| | Регистр A | | | | | | Регистр X | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Первоначальное содержимое | + | 1 | 2 | 3 | 4 | 5 | - | 6 | 7 | 8 | 9 | 10 |
| SRAX 1 | + | 0 | 1 | 2 | 3 | 4 | - | 5 | 6 | 7 | 8 | 9 |
| SLA 2 | + | 2 | 3 | 4 | 0 | 0 | - | 5 | 6 | 7 | 8 | 9 |
| SRC 4 | + | 6 | 7 | 8 | 9 | 2 | - | 3 | 4 | 0 | 0 | 5 |
| SRA 2 | + | 0 | 0 | 6 | 7 | 8 | - | 3 | 4 | 0 | 0 | 5 |
| SLC 1 | + | 0 | 6 | 7 | 8 | 3 | - | 4 | 0 | 0 | 5 | 0 |

• MOVE (переместить). C = 7; F = число.

Количество слов, определенное значением F, перемещается, начиная от ячейки M, в другие ячейки, адрес первой из которых задается содержимым индексного регистра 1. Перемещение осуществляется по одному слову за раз, и к концу выполнения операции значение в регистре rI1 увеличивается на F. Если F = 0, то ничего не происходит.

Необходимо следить за тем, чтобы группы ячеек, участвующих в перемещении, не перекрывались. Предположим, что F = 3 и M = 1000. Тогда, если rI1 = 999, перемещаем CONTENTS(1000) в CONTENTS(999), CONTENTS(1001) в CONTENTS(1000) и CONTENTS(1002) в CONTENTS(1001); в данном случае все нормально. Но если бы в регистре rI1 содержалось число 1001, то в результате были бы выполнены перемещения CONTENTS(1000) в CONTENTS(1001), CONTENTS(1001) в CONTENTS(1002), CONTENTS(1002) в CONTENTS(1003), т. е. мы бы переместили *одно и то же* слово CONTENTS(1000) в три различных места.

• NOP (no operation — нет операции). C = 0.

Никакие действия не выполняются, и эта команда просто пропускается. F и M игнорируются.

• HLT (halt — остановить). C = 5; F = 2.

Остановка работы компьютера. Пока оператор будет его перезапускать, все будет выглядеть так, как будто работает команда NOP (т. е. никакие действия не выполняются).

**Команды ввода-вывода.** MIX можно оснастить достаточно большим количеством устройств ввода-вывода (причем все они поставляются за дополнительную плату). Каждому устройству соответствует определенный номер.

| Номер устройства | Периферийное устройство | Размер блока, слов |
|---|---|---|
| $t$ | Накопитель на магнитной ленте номер $t$ $(0 \le t \le 7)$ | 100 |
| $d$ | Диск или барабан номер $d$ $(8 \le d \le 15)$ | 100 |
| 16 | Устройство чтения перфокарт | 16 |
| 17 | Перфоратор | 16 |
| 18 | Принтер | 24 |
| 19 | Терминал ввода данных | 14 |
| 20 | Перфолента | 14 |

Не каждый компьютер MIX будет оснащен всеми этими устройствами, поэтому время от времени мы будем специально упоминать о наличии тех или иных устройств. Некоторые из них нельзя использовать и для ввода, и для вывода. В приведенной выше таблице указаны фиксированные размеры блоков (в словах) для каждого устройства.

При вводе или выводе с помощью таких устройств, как накопитель на магнитной ленте, диск или барабан, происходит чтение или запись полных слов (состоящих из пяти байтов и знака). Но устройства ввода-вывода с номерами от 16 до 20 всегда работают в *символьном коде*, в котором каждый байт представляет один буквенно-цифровой символ. Поэтому с помощью каждого слова MIX передается сразу пять символов. Символьный код приведен в верхней части табл. 1, которая находится в конце данного раздела, и на форзацах в конце книги. Код 00 соответствует символу "␣", который обозначает *пробел*. Коды 01–29 представляют буквы от A до Z, среди которых есть несколько греческих букв; коды 30–39 представляют цифры 0, 1, ..., 9, а следующие коды 40, 41, ... — знаки пунктуации и другие специальные символы. (Набор символов MIX отражает положение дел на то время, когда компьютеры еще не могли справиться со строчными буквами.) С помощью символьного кода нельзя прочитать или записать все возможные величины, которые могут содержаться в байте, так как некоторые комбинации не определены. Более того, некоторые устройства ввода-вывода могут оказаться не предназначенными для обработки всех элементов из набора символов, например символы Σ и Π, встречающиеся в тексте, скорее всего, не будут восприняты устройством чтения перфокарт. Когда данные вводятся в символьном коде, всем словам присваиваются знаки "+", а при выводе знаки игнорируются. Если данные вводятся с терминала, то набор в конце каждой строки символа возврата каретки приводит к тому, что остаток строки заполняется пробелами.

Дисковые и барабанные устройства — это внешние устройства памяти, каждое из которых содержит блоки из 100 слов. При выполнении каждой команды IN, OUT или IOC (см. ниже) конкретный блок из 100 слов, на который ссылается команда, определяется текущим содержимым rX и не должен превосходить емкость используемого диска или барабана.

• IN (input — ввод). C = 36; F = номер устройства.
Эта команда реализует передачу информации из заданного устройства ввода в последовательно расположенные ячейки, начиная с ячейки M. Число ячеек, из которых передается информация, соответствует размеру блока для данного устройства (см. приведенную выше таблицу). Если предыдущая операция на этом же устройстве еще не закончена, то компьютер будет ожидать ее завершения. Время, в течение которого будет длиться передача информации, начатая по этой команде, зависит от скорости работы устройства ввода. Поэтому до момента завершения передачи информации программа не должна обращаться к этой информации в памяти. Не следует пытаться прочитать с магнитной ленты любой блок, следующий за блоком, который был записан последним.

• OUT (output — вывод). C = 37; F = номер устройства.
Эта команда реализует передачу информации из ячеек памяти, начиная с ячейки M, на заданное устройство вывода. Если сначала устройство не было готово, то компьютер будет ждать его готовности. Время, в течение которого будет длиться передача информации, начатая по этой команде, зависит от скорости устройства вывода. Поэтому до момента завершения передачи информации программа не должна производить изменения в соответствующих ячейках памяти.

• IOC (input-output control — управление вводом-выводом). C = 35; F = номер устройства.
В случае необходимости компьютер ожидает, пока освободится заданное устройство. Затем выполняется управляющая команда, которая зависит от типа применяемого устройства. В этой книге будут использоваться следующие примеры.

*Магнитная лента.* Если M = 0, то лента перематывается в начало. Если M < 0, то лента перематывается на −M блоков (т. е. на M блоков назад) или в начало, в зависимости от того, что произойдет раньше. Если M > 0, то лента перематывается вперед; перематывая ленту вперед, нельзя заходить дальше блока, который был записан последним.

Например, последовательность команд "OUT 1000(3): IOC -1(3); IN 2000(3)" записывает сто слов на ленту 3, а затем снова считывает их. Если надежность ленты не ставится под сомнение, то использование последних двух команд этой последовательности представляет собой медленный способ перемещения слов 1000–1099 в ячейки 2000–2099. Последовательность команд "OUT 1000(3); IOC +1(3)" некорректна.

*Диск или барабан.* M должно быть равно нулю. В результате устройство позиционируется в соответствии с содержимым rX, чтобы следующая операция IN или OUT на этом устройстве выполнялась быстрее в случае, если используется то же значение rX.

*Принтер.* M должно быть равно нулю. Команда "IOC 0(18)" заставит принтер перейти к началу (т. е. к верху) следующей страницы.

*Перфолента.* M должно быть равно нулю. Команда "IOC 0(20)" перематывает ленту в начало.

• JRED (jump ready — переход при готовности). C = 38; F = номер устройства.
Переход происходит в случае, если заданное устройство готово, т. е. завершена предыдущая операция, инициированная командой IN, OUT или IOC.

• JBUS (jump busy — переход при занятости). C = 34; F = номер устройства.
Антипод команды JRED: переход происходит, если заданное устройство *не* готово.

*Пример.* Команда "JBUS 1000(16)" из ячейки 1000 будет повторно выполняться до тех пор, пока устройство 16 *не* будет готово.

Перечисленные выше простые команды исчерпывают набор команд ввода-вывода компьютера MIX. В этом компьютере нет индикаторов контроля ленты и т. д., служащих для предотвращения аварийных ситуаций на периферийных устройствах. Любая подобная ситуация (бумага застревает, устройство отключается, лента отсутствует и т. д.) приводит к тому, что устройство остается занятым, звенит звонок и опытный оператор устраняет проблемы вручную, выполняя обычные процедуры обслуживания. О более сложных периферийных устройствах, которые являются более дорогими и более типичными представителями современного оборудования по сравнению с описанными здесь лентами, барабанами и дисками с фиксированным размером блоков, пойдет речь в разделах 5.4.6 и 5.4.9.

**Команды преобразования**

• NUM (convert to numeric — преобразовать в число). C = 5; F = 0.
Эта команда используется для преобразования символьного кода в число. M игнорируется. Предполагается, что регистры A и X содержат 10-байтовое число в символьном коде и команда NUM заносит в rA полученное числовое значение (которое рассматривается как десятичное число). Содержимое rX и знак rA не меняются. Байты 00, 10, 20, 30, 40, ... преобразуются в нулевые; в байты 01, 11, 21, ... заносится цифра 1 и т. д. Если происходит переполнение (что вполне возможно), сохраняется остаток по модулю $b^5$, где $b$ — размер байта.

• CHAR (convert to characters — преобразовать в символы). C = 5; F = 1.
Эта операция используется для преобразования машинного кода в символьный код, который воспринимается устройством вывода на перфокарты, ленту или на принтер. Значение из rA преобразуется в 10-байтовое десятичное число, которое заносится в регистры A и X в символьном коде. Знаки rA и rX не меняются. M игнорируется.

*Примеры*

| | Регистр A | | | | | | Регистр X | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Первоначальное содержимое | - | 00 | 00 | 31 | 32 | 39 | + | 37 | 57 | 47 | 30 | 30 |
| NUM 0 | - | | | | 12977700 | | + | 37 | 57 | 47 | 30 | 30 |
| INCA 1 | - | | | | 12977699 | | + | 37 | 57 | 47 | 30 | 30 |
| CHAR 0 | - | 30 | 30 | 31 | 32 | 39 | + | 37 | 37 | 36 | 39 | 39 |

**Время выполнения.** Чтобы определить количественные характеристики эффективности программ для MIX, для каждой его команды задано *время выполнения*, которое типично для компьютеров "урожая 1970 года".

ADD, SUB, все команды LOAD, все команды STORE (включая STZ), все команды сдвига и все команды сравнения выполняются в течение *двух тактов*. Для выполнения команды MOVE требуется один такт плюс еще два на каждое перемещаемое слово. Для каждой команды MUL, NUM, CHAR требуется 10 тактов, а для DIV — 12. Время выполнения операций с плавающей точкой определяется в разделе 4.2.1. Для всех остальных операций необходим один такт плюс время, в течение которого компьютер может ожидать завершения выполнения команд IN, OUT, IOC и HLT.

## Таблица 1

Код символа:

| 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ␣ | A | B | C | D | E | F | G | H | I | Δ | J | K | L | M | N | O | P | Q | R | Σ | Π | S | T | U |

| **00** | *1* | **01** | *2* | **02** | *2* | **03** | *10* |
|---|---|---|---|---|---|---|---|
| Нет операции<br>`NOP(0)` | | rA ← rA + V<br>`ADD(0:5)`<br>`FADD(6)` | | rA ← rA − V<br>`SUB(0:5)`<br>`FSUB(6)` | | rAX ← rA × V<br>`MUL(0:5)`<br>`FMUL(6)` | |
| **08** | *2* | **09** | *2* | **10** | *2* | **11** | *2* |
| rA ← V<br>`LDA(0:5)` | | rI1 ← V<br>`LD1(0:5)` | | rI2 ← V<br>`LD2(0:5)` | | rI3 ← V<br>`LD3(0:5)` | |
| **16** | *2* | **17** | *2* | **18** | *2* | **19** | *2* |
| rA ← −V<br>`LDAN(0:5)` | | rI1 ← −V<br>`LD1N(0:5)` | | rI2 ← −V<br>`LD2N(0:5)` | | rI3 ← −V<br>`LD3N(0:5)` | |
| **24** | *2* | **25** | *2* | **26** | *2* | **27** | *2* |
| M(F) ← rA<br>`STA(0:5)` | | M(F) ← rI1<br>`ST1(0:5)` | | M(F) ← rI2<br>`ST2(0:5)` | | M(F) ← rI3<br>`ST3(0:5)` | |
| **32** | *2* | **33** | *2* | **34** | *1* | **35** | *1+T* |
| M(F) ← rJ<br>`STJ(0:2)` | | M(F) ← 0<br>`STZ(0:5)` | | Устр. F занято?<br>`JBUS(0)` | | Упр. устр. F<br>`IOC(0)` | |
| **40** | *1* | **41** | *1* | **42** | *1* | **43** | *1* |
| rA : 0, переход<br>`JA[+]` | | rI1 : 0, переход<br>`J1[+]` | | rI2 : 0, переход<br>`J2[+]` | | rI3 : 0, переход<br>`J3[+]` | |
| **48** | *1* | **49** | *1* | **50** | *1* | **51** | *1* |
| rA ← [rA]? ± M<br>`INCA(0) DECA(1)`<br>`ENTA(2) ENNA(3)` | | rI1 ← [rI1]? ± M<br>`INC1(0) DEC1(1)`<br>`ENT1(2) ENN1(3)` | | rI2 ← [rI2]? ± M<br>`INC2(0) DEC2(1)`<br>`ENT2(2) ENN2(3)` | | rI3 ← [rI3]? ± M<br>`INC3(0) DEC3(1)`<br>`ENT3(2) ENN3(3)` | |
| **56** | *2* | **57** | *2* | **58** | *2* | **59** | *2* |
| CI ← rA(F) : V<br>`CMPA(0:5)`<br>`FCMP(6)` | | CI ← rI1(F) : V<br>`CMP1(0:5)` | | CI ← rI2(F) : V<br>`CMP2(0:5)` | | CI ← rI3(F) : V<br>`CMP3(0:5)` | |

Общая форма записи:

| **C** | *t* |
|---|---|
| Описание<br>`OP(F)` | |

C — код операции, поле команды (5:5)
F — уточн. кода опер., поле команды (4:4)
M — адрес команды после индексации
V = M(F) — содержимое поля F ячейки M
OP — символический код операции
(F) — стандартное значение F
*t* — время выполнения; *T* — время блокировки

| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| V | W | X | Y | Z | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | . | , | ( | ) | + | - | * | / | = | $ | < | > | @ | ; | : | ' |

| **04** | *12* | **05** | *10* | **06** | *2* | **07** | *1 + 2F* |
|---|---|---|---|---|---|---|---|
| rA ← rAX/V<br>rX ← остаток<br>DIV(0:5)<br>FDIV(6) | | Специальные<br>NUM(0)<br>CHAR(1)<br>HLT(2) | | Сдвиг на M байт<br>SLA(0) SRA(1)<br>SLAX(2) SRAX(3)<br>SLC(4) SRC(5) | | Переместить F слов из M в rI1<br>MOVE(1) | |
| **12** | *2* | **13** | *2* | **14** | *2* | **15** | *2* |
| rI4 ← V<br>LD4(0:5) | | rI5 ← V<br>LD5(0:5) | | rI6 ← V<br>LD6(0:5) | | rX ← V<br>LDX(0:5) | |
| **20** | *2* | **21** | *2* | **22** | *2* | **23** | *2* |
| rI4 ← −V<br>LD4N(0:5) | | rI5 ← −V<br>LD5N(0:5) | | rI6 ← −V<br>LD6N(0:5) | | rX ← −V<br>LDXN(0:5) | |
| **28** | *2* | **29** | *2* | **30** | *2* | **31** | *2* |
| M(F) ← rI4<br>ST4(0:5) | | M(F) ← rI5<br>ST5(0:5) | | M(F) ← rI6<br>ST6(0:5) | | M(F) ← rX<br>STX(0:5) | |
| **36** | *1 + T* | **37** | *1 + T* | **38** | *1* | **39** | *1* |
| Ввод с устр. F<br>IN(0) | | Вывод на устр. F<br>OUT(0) | | Устр. F готово?<br>JRED(0) | | Переходы<br>JMP(0) JSJ(1),<br>JOV(2) JNOV(3),<br>а также [*] ниже | |
| **44** | *1* | **45** | *1* | **46** | *1* | **47** | *1* |
| rI4 : 0, переход<br>J4[+] | | rI5 : 0, переход<br>J5[+] | | rI6 : 0, переход<br>J6[+] | | rX : 0, переход<br>JX[+] | |
| **52** | *1* | **53** | *1* | **54** | *1* | **55** | *1* |
| rI4 ← [rI4]? ± M<br>INC4(0) DEC4(1)<br>ENT4(2) ENN4(3) | | rI5 ← [rI5]? ± M<br>INC5(0) DEC5(1)<br>ENT5(2) ENN5(3) | | rI6 ← [rI6]? ± M<br>INC6(0) DEC6(1)<br>ENT6(2) ENN6(3) | | rX ← [rX]? ± M<br>INCX(0) DECX(1)<br>ENTX(2) ENNX(3) | |
| **60** | *2* | **61** | *2* | **62** | *2* | **63** | *2* |
| CI ← rI4(F) : V<br>CMP4(0:5) | | CI ← rI5(F) : V<br>CMP5(0:5) | | CI ← rI6(F) : V<br>CMP6(0:5) | | CI ← rX(F) : V<br>CMPX(0:5) | |

rA — регистр A
rX — регистр X
rAX — регистры A и X вместе
rI$i$ — индексный регистр $i$, $1 \le i \le 6$
rJ — регистр J
CI — индикатор сравнения

| [*]: | | [+]: |
|---|---|---|
| JL(4) | < | N(0) |
| JE(5) | = | Z(1) |
| JG(6) | > | P(2) |
| JGE(7) | ≥ | NN(3) |
| JNE(8) | ≠ | NZ(4) |
| JLE(9) | ≤ | NP(5) |

В частности, заметим, что ENTA выполняется в течение одной единицы времени, а LDA — в течение двух. Время выполнения команд легко запомнить, потому что, за исключением команд сдвига, преобразования, MUL и DIV, число единиц времени равно числу обращений к оперативной памяти (включая обращение к самой команде).

Основная единица измерения времени MIX — это относительная мера, которую мы просто обозначим через $u$. Ее можно считать равной, скажем, 10 мкс (для сравнительно недорогого компьютера) или 10 нс (для достаточно дорогого).

*Пример.* Последовательность команд LDA 1000; INCA 1; STA 1000 выполняется ровно $5u$.

*Мой взор проник в ее глубины —*
*Туда, где бьется пульс машины.*

— ВИЛЬЯМ ВОРДСВОРС (WILLIAM WORDSWORTH),
*She Was a Phantom of Delight* (1804)

**Резюме.** Итак, мы обсудили все характеристики MIX, за исключением его "кнопки GO" (пуск), о которой пойдет речь в упр. 26. Хотя в компьютере MIX приблизительно 150 различных операций, они вписываются в несколько простых схем и поэтому легко запоминаются. В табл. 1 приведены операции для параметра C. После имени каждой команды в круглых скобках указано стандартное значение поля F.

Следующие упражнения помогут лучше усвоить материал данного раздела. В основном, эти упражнения очень просты, так что постарайтесь их выполнить.

## УПРАЖНЕНИЯ

**1.** [*00*] Если бы MIX был троичным компьютером, то сколько троичных чисел поместилось бы в байте?

**2.** [*02*] Если бы величина, которую нужно представить в машине MIX, могла достигать такого большого значения, как 99 999 999, то сколько соседних байтов было бы занято этой величиной?

**3.** [*02*] Укажите частичные спецификации поля (L:R) для (a) адресного поля, (b) индексного поля, (c) поля спецификации поля и (d) поля кода операций команды MIX.

**4.** [*00*] Последним примером в (5) является команда "LDA -2000,4". Насколько она законна ввиду того, что адреса памяти не должны быть отрицательными?

**5.** [*10*] Какая символическая запись, аналогичная (4), соответствует (6), если (6) рассматривать как команду MIX?

▸ **6.** [*10*] Предположим, что в ячейке 3000 содержится

| + | 5 | 1 | 200 | 15 |
|---|---|---|---|---|

.

Каким будет результат выполнения следующих команд? (Установите, являются ли какие-либо из них неопределенными или только частично определенными.) (a) LDAN 3000; (b) LD2N 3000(3:4); (c) LDX 3000(1:3); (d) LD6 3000; (e) LDXN 3000(0:0).

**7.** [*M15*] С помощью алгебраических операций $X \bmod Y$ и $\lfloor X/Y \rfloor$ дайте точное определение результатов выполнения команды DIV для всех случаев, когда не происходит переполнение.

**8.** [*15*] В *последнем примере команды* DIV на с. 165 *содержимым* rX *до операции является* | + | 1235 | 0 | 3 | 1 |. Если бы содержимым регистра было | - | 1234 | 0 | 3 | 1 |, в то время как все остальные условия примера остались бы прежними, то что содержалось бы в регистрах A и X после выполнения команды DIV?

▸ **9.** [*15*] Перечислите все команды MIX, которые могут изменить значение флага переполнения. (Исключите из рассмотрения операции с плавающей точкой.)

**10.** [*15*] Перечислите все команды MIX, которые могут изменить значение флага сравнения

▸ **11.** [*15*] Перечислите все команды MIX, которые могут изменить содержимое rI1.

**12.** [*10*] Найдите единственную команду, действие которой эквивалентно умножению текущего содержимого rI3 на два и сохранению результата в rI3.

▸ **13.** [*10*] Предположим, в ячейке 1000 содержится команда "JOV 1001" Эта команда переключает флаг переполнения в положение 0, если он находится в положении 1 (и в любом случае следующая команда выбирается из ячейки 1001). Изменится ли что-нибудь, если заменить эту команду командой "JNOV 1001"? Что будет, если заменить ее командой "JOV 1000" или "JNOV 1000"?

**14.** [*20*] Для каждой команды MIX выясните, существует ли способ определения части ±AA, I и F таким образом, чтобы она была эквивалентна NOP (за исключением того, что время выполнения может увеличиться). Предположим, о содержимом регистров либо ячеек памяти ничего не известно В тех случаях, когда возможно осуществить NOP, объясните, как это сделать. *Примеры.* INCA эквивалентна NOP, когда адресное и индексное поля равны нулю. JMP никогда не будет эквивалентна NOP, так как она изменяет содержимое rJ.

**15.** [*10*] Сколько *буквенно-цифровых символов* содержится в блоке данных терминала или устройства вывода на перфоленту; устройства чтения перфокарт или перфоратора; АЦПУ?

**16.** [*20*] Напишите программу, которая обнуляет все ячейки памяти 0000–0099 и является (a) настолько короткой, насколько это возможно; (b) настолько быстрой, насколько это возможно. [*Указание.* Воспользуйтесь командой MOVE.]

**17.** [*26*] Пусть выполнены условия предыдущего упражнения, только нужно обнулить ячейки с 0000 по $N$ включительно, где $N$ — текущее содержимое rI2. Ваши программы должны удовлетворять условиям (a) и (b) из предыдущего упражнения, работать для любого значения $0 \le N \le 2999$ и начинаться с ячейки 3000.

▸ **18.** [*22*] После того как будет выполнена следующая программа "номер один", какие изменения произойдут с регистрами, флагом переполнения, флагом сравнения и оперативной памятью? (Например, каким будет окончательное значение в регистре rI1 или rX? Каким будет значение флага переполнения и флага сравнения?)

```
STZ  1
ENNX 1
STX  1(0:1)
SLAX 1
ENNA 1
INCX 1
ENT1 1
SRC  1
ADD  1
DEC1 -1
STZ  1
CMPA 1
```

```
MOVE -1,1(1)
NUM  1
CHAR 1
HLT  1
```

▸ **19.** [*14*] Каким будет время выполнения программы из предыдущего примера без учета команды HLT?

**20.** [*20*] Напишите программу, которая заносит во *все* 4 000 ячеек памяти команду HLT, а затем останавливается.

▸ **21.** [*24*] (a) Может ли нуль вообще содержаться в регистре J? (b) Напишите программу, которая помещает в регистр J значение $N$, $0 < N \leq 3000$, при условии, что $N$ содержится в rI4. Программа должна начинаться с адреса 3000. После выполнения программы содержимое всех ячеек памяти должно остаться неизменным.

▸ **22.** [*28*] В ячейке 2000 содержится целое число $X$. Напишите две программы, которые вычисляют $X^{13}$ и останавливаются после занесения результата в регистр A. Одна программа должна занимать минимальное количество ячеек памяти MIX, а другая — выполняться за минимальное время. При этом предполагается, что $X^{13}$ умещается в одном слове.

**23.** [*27*] В ячейке 0200 содержится слово

| + | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|

.

Напишите две программы, которые получают "отраженное" слово

| + | $e$ | $d$ | $c$ | $b$ | $a$ |
|---|---|---|---|---|---|

и останавливаются после занесения этого результата в регистр A. В одной из программ не должна использоваться способность MIX загружать и сохранять частичные поля слов. Обе программы должны занимать минимальное количество ячеек памяти, которое возможно при данных условиях (включая ячейки, занимаемые самой программой, а также ячейки, используемые для временного хранения промежуточных результатов).

**24.** [*21*] Пусть в регистрах A и X содержатся

| + | 0 | $a$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|---|

и

| + | $e$ | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|

соответственно. Напишите две программы, которые меняют содержимое этих регистров на

| + | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|

и

| + | 0 | $f$ | $g$ | $h$ | $i$ |
|---|---|---|---|---|---|

соответственно и удовлетворяют двум условиям: (a) занимают минимальный объем памяти; (b) выполняются за минимальное время.

▸ **25.** [*30*] Предположим, фирма — производитель компьютера MIX планирует выпустить более мощный компьютер ("Mixmaster"?) и ей нужно убедить как можно больше владельцев компьютера MIX потратиться на более дорогой компьютер Причем фирма собирается разработать аппаратное обеспечение нового компьютера таким образом, чтобы он был *расширенным вариантом* MIX. Это означает, что все правильно написанные программы для MIX будут работать на новом компьютере и в них не нужно будет вносить каких-либо изменений. Внесите свои предложения по поводу того, что можно было бы воплотить в новом компьютере. (Например, *можете ли вы извлечь бóльшую пользу, применив поле* I команды?)

▸ **26.** [*32*] Эта задача состоит в написании программы загрузки перфокарт. На каждом компьютере процесс начальной загрузки, т. е. получения первоначальной информации и

корректного начала работы, выполняется по-разному. В случае использования компьютера MIX содержимое перфокарты можно считать только в символьном коде, и перфокарты, содержащие саму загружающую программу, тоже должны удовлетворять этому ограничению Поэтому с перфокарты можно считать не все возможные значения в байтах; кроме того, каждое слово, считываемое с перфокарт, является положительным.

У компьютера MIX есть одна функция, о которой не говорилось в тексте раздела. Речь идет о "кнопке GO", которая используется для начального запуска компьютера, когда в его памяти содержится произвольная информация. Когда оператор нажимает эту кнопку, выполняются следующие действия.

1) Одна карта считывается в ячейки 0000–0015; по сути, это эквивалентно команде "IN 0(16)".
2) Когда карта полностью прочитана и устройство чтения перфокарт больше не занято, осуществляется переход (JMP) в ячейку 0000. Кроме того, обнуляется содержимое регистра J.
3) Компьютер начинает выполнять программу, которую он считал с перфокарты.

*Замечание.* У компьютеров MIX, не имеющих устройств чтения перфокарт, кнопка GO находится на другом устройстве ввода Но в данной задаче предполагается наличие устройства чтения перфокарт под номером 16.

Программу загрузки нужно написать так, чтобы она удовлетворяла следующим условиям.

i) Вводимая колода должна начинаться с программы загрузки, за которой следуют перфокарты входных данных, содержащие загружаемые числа. Затем идет "переходная карта", которая закрывает программу загрузки и переходит к началу программы. Программа загрузки должна поместиться на двух перфокартах

ii) Перфокарты входных данных должны иметь следующий формат.

Колонки 1–5: программой загрузки игнорируются
Колонка 6: число последовательных слов, загружаемых с этой перфокарты (*лежит в промежутке от 1 до 7 включительно*).
Колонки 7–10: адрес ячейки, в которую загружается слово 1. Этот адрес всегда больше 100, чтобы не было перекрытия с программой загрузки.
Колонки 11–20: слово 1.
Колонки 21–30: слово 2 (если содержимое колонки 6 $\geq 2$).
...
Колонки 71–80: слово 7 (если в колонке 6 содержится число 7).

Содержимое слов 1, 2, ... перфорируется и представляется в числовом виде по аналогии с десятичными числами. Если слово отрицательно, то знак "–" ("позиция 11") *перфорируется поверх* наименьшей значащей цифры, т. е в колонке 20. Предполагается, что по этой причине символьный код вводится в колонки 10, 11, 12, ..., 19, а не в 30, 31, 32, .. , 39. *Например, если на перфокарте в колонках 1–40 выбито*

$$\texttt{ABCDE3100001234567890000000001000000010}\overline{\texttt{0}},$$

то будут загружены следующие данные:

1000. +0123456789, 1001. +0000000001; 1002: −0000000100.

iii) На переходной карте в колонках 1–10 должна содержаться информация в формате TRANSOnnnn, где nnnn — адрес ячейки, с которой начинается выполнение программы.

iv) Программа загрузки должна работать независимо от размера байта, чтобы в содержащие ее перфокарты не нужно было вносить никаких изменений. На картах не должны содержаться символы, соответствующие байтам 20, 21, 48, 49, 50, ... (т. е. символы Σ,

П, =, $, <, ...), так как их не сможет прочитать ни одно устройство чтения перфокарт. В частности, нельзя использовать команды ENT, INC и CMP, поскольку их невозможно перфорировать на карте.

### 1.3.2. Язык ассемблера компьютера MIX

Символический язык компьютера MIX используется для того, чтобы облегчить чтение и написание программ, а также избавить программиста от беспокойства по поводу утомительных мелких деталей, которые часто становятся причиной дополнительных ошибок. Этот язык под названием MIXAL (MIX Assembly Language*) является расширенным вариантом системы обозначений, которая использовалась для команд в предыдущем разделе. Главной его особенностью является то, что для обозначения чисел можно использовать буквенные имена, а с помощью поля метки связывать имена с ячейками памяти.

Чтобы легче было разобраться в языке MIXAL, рассмотрим простой пример. Приведенный ниже код является частью большой программы; это подпрограмма нахождения максимума $n$ элементов $X[1], \ldots, X[n]$ с помощью алгоритма 1.2.10M.

**Программа M** (*Нахождение максимума*). Занесение значений в регистры: rA $\equiv m$, rI1 $\equiv n$, rI2 $\equiv j$, rI3 $\equiv k$, $X[i] \equiv$ CONTENTS(X + $i$).

| Машинный код | | | | | | Номер строки | МЕТКА | ОП | АДРЕС | Повтор | Примечания |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | 01 | X | EQU | 1000 | | |
| | | | | | | 02 | | ORIG | 3000 | | |
| 3000: | + | 3009 | 0 | 2 | 32 | 03 | MAXIMUM | STJ | EXIT | 1 | Связь с подпрограммой. |
| 3001: | + | 0 | 1 | 2 | 51 | 04 | INIT | ENT3 | 0,1 | 1 | *M1. Инициализация.* $k \leftarrow n$. |
| 3002: | + | 3005 | 0 | 0 | 39 | 05 | | JMP | CHANGEM | 1 | $j \leftarrow n$, $m \leftarrow X[n]$, $k \leftarrow n-1$. |
| 3003: | + | 1000 | 3 | 5 | 56 | 06 | LOOP | CMPA | X,3 | $n-1$ | *M3. Сравнение.* |
| 3004: | + | 3007 | 0 | 7 | 39 | 07 | | JGE | *+3 | $n-1$ | Перейти к шагу M5, если $m \geq X[k]$. |
| 3005: | + | 0 | 3 | 2 | 50 | 08 | CHANGEM | ENT2 | 0,3 | $A+1$ | *M4. Изменение m.* $j \leftarrow k$. |
| 3006: | + | 1000 | 3 | 5 | 08 | 09 | | LDA | X,3 | $A+1$ | $m \leftarrow X[k]$. |
| 3007: | + | 1 | 0 | 1 | 51 | 10 | | DEC3 | 1 | $n$ | *M5. Уменьшение k.* |
| 3008: | + | 3003 | 0 | 2 | 43 | 11 | | J3P | LOOP | $n$ | *M2. Все проверено?* Перейти к M3, если $k>0$. |
| 3009: | + | 3009 | 0 | 0 | 39 | 12 | EXIT | JMP | * | 1 | Возврат к главной программе. ∎ |

Эта программа позволяет проиллюстрировать сразу несколько моментов.

a) Столбцы "МЕТКА", "ОП" и "АДРЕС" представляют особый интерес; в них содержится программа на символическом языке MIXAL, и ниже мы подробно рассмотрим эту программу.

b) В столбце "Машинный код" содержатся реальные цифровые команды машинного языка, соответствующие символическим командам языка MIXAL. MIXAL был разработан так, чтобы любую программу на этом языке можно было легко транслировать на цифровой машинный язык. Трансляция обычно выполняется другой компьютерной программой, которая называется *ассемблерной программой* или *ассемблером*. Таким образом, для программирования на машинном языке программист может использовать MIXAL, чтобы не определять эквивалентные цифровые коды команд вручную. В этой книге практически все программы для MIX написаны на языке MIXAL.

* Язык ассемблера компьютера MIX. — *Прим. перев.*

c) Столбец "Номер строки" является необязательным компонентом программы на языке MIXAL, в этой книге он включен в примеры программ просто для облегчения ссылок на различные части программы.

d) В столбце "Примечания" содержатся пояснения к программе и ссылки на шаги алгоритма 1 2.10М. Читателю следует сравнить этот алгоритм (с. 127) с приведенной выше программой. Обратите внимание, что при переводе алгоритма в код для MIX допущена некоторая "программистская вольность", например шаг М2 оказался последним. При "занесении значений в регистры" (см. начало программы М) показано, какие компоненты MIX соответствуют переменным, используемым в алгоритме.

e) Столбец "Повтор" будет полезен при изучении многих программ для MIX, которые рассматриваются в данной книге. Он представляет *профиль программы*, т. е. показывает, сколько раз команда из данной строки будет выполняться в ходе работы программы. Так, например, команда из строки 06 будет выполнена $n-1$ раз и т. д. На основании этой информации можно определить, сколько времени требуется на выполнение подпрограммы; оно равно $(5+5n+3A)u$, где $A$ — величина, которая была тщательно проанализирована в разделе 1.2.10.

А теперь давайте обсудим запись программы М на языке MIXAL. Строка 01,

```
X EQU 1000,
```

говорит о том, что символ X будет *эквивалентом* числа 1 000. Эффект этого действия проявляется в строке 06, в которой цифровой эквивалент команды "CMPA X,3" имеет вид

| + | 1000 | 3 | 5 | 56 |
|---|---|---|---|---|

т. е. "CMPA 1000,3".

Строка 02 указывает, что следующие строки расположены, начиная с адреса 3000. Поэтому символ MAXIMUM, находящийся в поле METKA строки 03, становится эквивалентным числу 3000, символ INIT — числу 3001, символ LOOP — числу 3003 и т. д.

В строках с 03 по 12 поля ОП содержатся символические имена команд MIX: STJ, ENT3 и т. д. Но символические имена EQU и ORIG, которые находятся в столбце ОП строк 01 и 02, несколько отличаются от них; EQU и ORIG называются *псевдооперациями*, так как они являются операторами языка MIXAL, но не порождают машинных команд MIX. Псевдооперации предоставляют специальную информацию о символической программе и в то же время не являются командами самой машины MIX. Так, например, строка

```
X EQU 1000
```

только сообщает некоторые сведения *о* программе М; это не означает, что во время выполнения программы какой-либо переменной присваивается значение 1000. Обратите внимание, что для строк 01 и 02 машинные команды не порождаются.

Строка 03 — это команда "сохранить J", которая сохраняет содержимое регистра J в поле (0:2) ячейки EXIT. Другими словами, она сохраняет rJ в поле адреса команды, расположенной в строке 12.

Как уже упоминалось, программа М является частью большой программы. Если, например, в каком-нибудь месте этой программы встретится последовательность

команд

```
ENT1 100
JMP  MAXIMUM
STA  MAX
```

это приведет к вызову программы M со значением $n$, равным 100. В этом случае программа M найдет максимальный элемент среди $X[1], \ldots, X[100]$ и вернет управление команде "STA MAX", записав максимальное значение в rA, а его номер $j$ — в rI2 (см. упр. 3).

Строка 05 передает управление строке 08. Строки 04, 05 и 06 в дополнительных объяснениях не нуждаются. В строке 07 вводится новое обозначение: звездочка (читается "текущий"), которая указывает на начальную ячейку текущей команды; поэтому запись "*+3" ("текущий плюс три") означает "три ячейки после начала текущей команды". Поскольку в строке 07 находится команда, начинающаяся с ячейки 3004, то "*+3" означает ссылку на ячейку 3007.

Оставшаяся часть символического кода не требует каких-либо разъяснений, так как говорит сама за себя. Обратите внимание, что в строке 12 снова появляется звездочка (см. упр. 2).

В следующем примере демонстрируются другие функции языка ассемблера. Задача заключается в том, чтобы вычислить и напечатать таблицу первых 500 простых чисел, расположив их в 10 столбцах по 50 чисел. На АЦПУ эта таблица будет напечатана следующим образом.

```
ПЕРВЫЕ ПЯТЬСОТ ПРОСТЫХ ЧИСЕЛ
      0002 0233 0547 0877 1229 1597 1993 2371 2749 3187
      0003 0239 0557 0881 1231 1601 1997 2377 2753 3191
      0005 0241 0563 0883 1237 1607 1999 2381 2767 3203
      0007 0251 0569 0887 1249 1609 2003 2383 2777 3209
      0011 0257 0571 0907 1259 1613 2011 2389 2789 3217
        :                                             :
      0229 0541 0863 1223 1583 1987 2357 2741 3181 3571
```

Воспользуемся следующим методом.

**Алгоритм P** (*Печать таблицы 500 простых чисел*). Этот алгоритм (рис. 14) состоит из двух частей: при выполнении шагов P1–P8 готовится внутренняя таблица 500 простых чисел, а при выполнении шагов P9–P11 результаты печатаются в таком виде, как показано выше. В последней части программы используются два "буфера", в которых создаются образы строк: пока печатается содержимое одного буфера, другой заполняется информацией.

**P1.** [Начать таблицу.] Присвоить PRIME[1] $\leftarrow 2$, N $\leftarrow 3$, J $\leftarrow 1$. (В этой программе N пробегает нечетные числа в поисках кандидатов в простые числа; J подсчитывает, сколько простых чисел уже найдено.)

**P2.** [N — простое число.] Присвоить J $\leftarrow$ J $+ 1$, PRIME[J] $\leftarrow$ N.

**P3.** [500 чисел найдены?] Если J $= 500$, перейти к шагу P9.

**P4.** [Увеличить N.] Присвоить N $\leftarrow$ N $+ 2$.

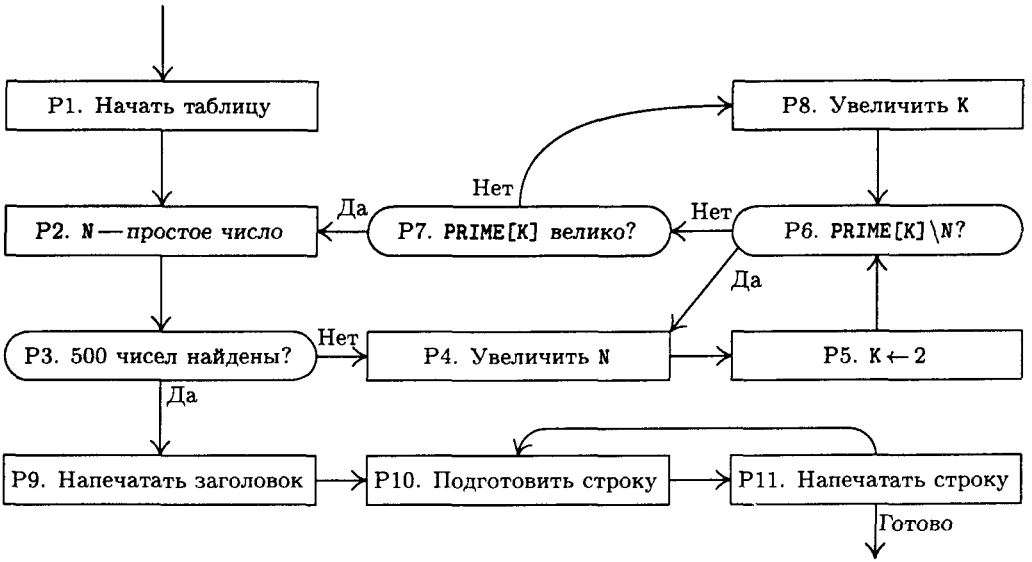



**Рис. 14.** Алгоритм P.

**P5.** [K ← 2.] Присвоить K ← 2. (PRIME[K] *пробегает возможные простые делители* N.)

**P6.** [PRIME[K]\N?] Разделить N на PRIME[K]; пусть Q — это частное от деления, а R — остаток. Если R = 0 (т. е. N не является простым), перейти к шагу P4.

**P7.** [PRIME[K] велико?] Если Q ≤ PRIME[K], перейти к шагу P2. (В таком случае N должно быть простым; доказательство этого факта интересное и немного необычное; см. упр. 6.)

**P8.** [Увеличить K.] Увеличить K на 1 и перейти к шагу P6.

**P9.** [Напечатать заголовок.] Теперь мы готовы к тому, чтобы напечатать таблицу. Переведем АЦПУ на следующую страницу. Занесем в BUFFER[0] строку заголовка и напечатаем эту строку. Присвоим B ← 1, M ← 1.

**P10.** [Подготовить строку.] Поместить PRIME[M], PRIME[50 + M], ..., PRIME[450 + M] в BUFFER[B] в соответствующем формате.

**P11.** [Напечатать строку.] Напечатать BUFFER[B]; присвоить B ← 1 − B (тем самым переключаясь на другой буфер) и увеличить M на 1. Если M ≤ 50, вернуться к шагу P10; в противном случае выполнение алгоритма заканчивается. ∎

**Программа P** (*Печать таблицы 500 простых чисел*). Данная программа написана несколько "топорным" способом, и это неспроста. Причина в том, что преследовалась цель — проиллюстрировать в одной программе большинство возможностей MIXAL. rI1 ≡ J − 500; rI2 ≡ N; rI3 ≡ K; rI4 указывает на B; rI5 равно M плюс число, кратное 50.

```
01 * ПРИМЕР ПРОГРАММЫ  ...  ТАБЛИЦА ПРОСТЫХ ЧИСЕЛ
02 *
03 L        EQU  500          Искомое количество простых чисел.
04 PRINTER  EQU  18           Номер АЦПУ.
05 PRIME    EQU  -1           Память для таблицы простых чисел.
06 BUF0     EQU  2000         Память для BUFFER[0].
07 BUF1     EQU  BUF0+25      Память для BUFFER[1].
08          ORIG 3000
09 START    IOC  0(PRINTER)   Перейти к новой странице.
10          LD1  =1-L=        P1. Начать таблицу. J ← 1.
11          LD2  =3=              N ← 3
12 2H       INC1 1            P2. N — простое число. J ← J + 1.
13          ST2  PRIME+L,1        PRIME[J] ← N.
14          J1Z  2F           P3. 500 чисел найдены?
15 4H       INC2 2            P4. Увеличить N.
16          ENT3 2            P5. K ← 2.
17 6H       ENTA 0            P6. PRIME[K]\N?
18          ENTX 0,2              rAX ← N.
19          DIV  PRIME,3          rA ← Q,  rX ← R.
20          JXZ  4B               Перейти к P4, если R = 0.
21          CMPA PRIME,3      P7. PRIME[K] велико?
22          INC3 1            P8. Увеличить K.
23          JG   6B               Перейти к P6, если Q > PRIME[K].
24          JMP  2B               В противном случае N — простое.
25 2H       OUT  TITLE(PRINTER) P9. Напечатать заголовок.
26          ENT4 BUF1+10          Присвоить B ← 1.
27          ENT5 -50              Присвоить M ← 0.
28 2H       INC5 L+1              Увеличить M
29 4H       LDA  PRIME,5      P10. Подготовить строку
30          CHAR                  Преобразовать PRIME[M] в
31          STX  0,4(1:4)             десятичный формат.
32          DEC4 1
33          DEC5 50               (rI5 уменьшается на 50, пока
34          J5P  4B                   не станет неположительным.)
35          OUT  Q,4(PRINTER) P11. Напечатать строку.
36          LD4  24,4             Переключить буфера.
37          J5N  2B               Если rI5 = 0, то работа алгоритма
38          HLT                       заканчивается.
39 * ПЕРВОНАЧАЛЬНОЕ СОДЕРЖИМОЕ ТАБЛИЦ И БУФЕРОВ
40          ORIG PRIME+1
41          CON  2            Первое простое число — 2.
42          ORIG BUF0-5
43 TITLE    ALF  FIRST        Символы для
44          ALF   FIVE            строки заголовка.
45          ALF   HUND
46          ALF  RED P
47          ALF  RIMES
48          ORIG BUF0+24
49          CON  BUF1+10      Буфера ссылаются один на другой.
50          ORIG BUF1+24
```

```
51          CON  BUF0+10
52          END  START          Конец программы.
```

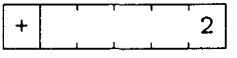
В отношении этой программы необходимо отметить следующие интересные моменты.

1. Строки 01, 02 и 39 начинаются со звездочки. Таким образом обозначается строка комментария, которая содержит только пояснения и не оказывает реального воздействия на транслируемую программу.

2. Как и в программе M, псевдооперация EQU из строки 03 определяет эквивалент символа. В данном случае эквивалентом L назначено число 500. (В строках 10–24 этой программы L представляет количество простых чисел, которые нужно найти.) Обратите внимание, что в строке 05 символу PRIME присваивается *отрицательный* эквивалент; вообще говоря, эквивалентом символа может быть любое число, состоящее из пяти байтов и знака. В строке 07 эквивалент BUF1 вычисляется по формуле BUF0+25, что в результате дает 2025. В MIXAL арифметические операции над числами можно выполнять в ограниченном объеме. Еще один пример арифметической операции появляется в строке 13, в которой ассемблер вычисляет значение PRIME+L (в данном случае это 499).

3. Символ PRINTER в строках 25 и 35 используется в F-части. F-часть, которая всегда заключается в круглые скобки, может состоять из чисел либо символов точно так, как другие части поля ADDRESS. В строке 31 иллюстрируется спецификация частичного поля "(1:4)", в котором используется двоеточие.

4. В MIXAL предусмотрено несколько способов определения слов, которые не являются командами. В строке 41 псевдооперация CON используется для определения обычной константы, "2". В результате трансляции строки 41 получится слово



.

В строке 49 присутствует немного более сложная константа, "BUF1+10", в результате трансляции которой получится слово

.

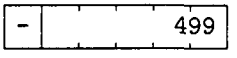
Константа может быть заключена между знаками равенства (см. строки 10 и 11); в этом случае она называется *литералом (или буквенной константой)*. Ассемблер автоматически создает для буквенных констант внутренние имена и вставляет строки "CON". Например, в результате трансляции строк 10 и 11 программы P получится

```
10          LD1  con1
11          LD2  con2
```

а в конце программы, между строками 51 и 52, в результате трансляции будут вставлены строки

```
51a  con1  CON  1-L
51b  con2  CON  3
```

Трансляция строки 51а дает слово




.

Использовать буквенные константы (литералы), несомненно, удобно, так как программистам не нужно изобретать для тривиальных констант символические имена, а также помнить о том, что в конце каждой программы необходимо вставлять константы. Таким образом, программист может сосредоточиться на главной задаче и не волноваться по поводу подобных деталей. (Однако нужно заметить, что примеры литералов в программе P не слишком удачны, поскольку, заменив строки 10 и 11 более эффективными командами "ENT1 1-L" и "ENT2 3", мы несколько улучшили бы программу.)

5. Хороший язык ассемблера должен имитировать ход *мыслей* программиста при написании машинных программ. Одним из примеров этой философии является использование литералов, о которых только что шла речь. В качестве другого примера можно привести применение символа "*", которое обсуждалось при описании программы M. А третьим примером является идея использования *локальных символов*, таких как символ 2H, который появляется в поле метки строк 12, 25 и 28.

Локальные символы — это специальные символы, которые можно *переопределять* столько раз, сколько нужно. Глобальный символ, например PRIME, имеет только одно значение на протяжении всей программы, и если бы он появился в поле метки более чем одной строки, то ассемблер зафиксировал бы ошибку. Но локальные символы имеют различную природу; например, мы пишем 2H ("2 here" — "2 здесь") в поле метки и 2F ("2 forward" — "2 вперед") или 2B ("2 backward" — "2 назад") в адресном поле строки MIXAL:

2B означает ближайшую *предыдущую* метку 2H;
2F означает ближайшую *следующую* метку 2H.

Таким образом, символ "2F" в строке 14 означает ссылку на строку 25, символ "2B" в строке 24 — ссылку назад, на строку 12, а символ "2B" в строке 37 — ссылку на строку 28. Адрес 2F или 2B никогда не относится к *собственной* строке. Например, три строки кода MIXAL

```
2H    EQU   10
2H    MOVE  2F(2B),
2H    EQU   2B-3
```

в сущности, эквивалентны одной строке

```
MOVE *-3(10).
```

Символы 2F и 2B никогда не следует использовать в поле метки, а символ 2H — в поле адреса. Существует десять локальных символов, которые можно получить, заменив в этих примерах "2" любой другой цифрой от 0 до 9.

Идею локальных символов выдвинул М. Э. Конвей (M. E. Conway) в 1958 году в связи с разработкой ассемблера для машины UNIVAC I. Локальные символы освобождают программиста от необходимости выбора символических имен для каждого адреса, когда нужно всего лишь сослаться на команду, находящуюся на расстоянии нескольких строк. Соседним командам не всегда можно придумать подходящие имена, поэтому программисты склонны вводить такие лишенные содержания символы,

как X1, X2, X3 и т. д., что создает потенциальную опасность их повторения. Поэтому использовать локальные символы в языке ассемблера очень полезно и совершенно естественно.

6. Адресная часть строк 30 и 38 пуста. Это означает, что в результате трансляции получится нулевой адрес. В строке 17 адресную часть тоже можно было бы оставить пустой, но без этого лишнего 0 программа стала бы менее наглядной.

7. В строках 43–47 используется операция ALF, которая создает пятибайтовую константу в буквенно-цифровом символьном коде MIX. Например, в результате трансляции строки 45 получится слово

| + | 00 | 08 | 24 | 15 | 04 |
|---|---|---|---|---|---|

т. е. "␣HUND" (часть слова "пятьсот") — часть строки заголовка в выходных данных программы P.

*Все ячейки, содержимое которых не определено в программе MIXAL, обычно обнуляются* (за исключением ячеек, которые используются загружающей программой; обычно это ячейки 3700–3999). Поэтому после строки 47 нет необходимости определять пробелы в других словах заголовка.

8. Вместе с операцией ORIG можно выполнять арифметические действия (см. строки 40, 42 и 48).

9. В последней строке законченной программы на языке MIXAL всегда присутствует код операции END. Адрес в этой строке указывает на ячейку, с которой начинает выполняться программа после загрузки в память.

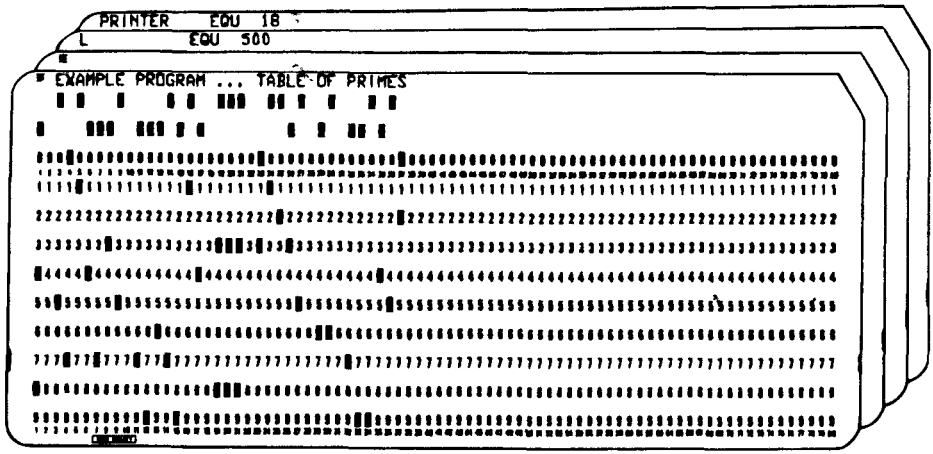
10. И в завершение анализа программы P следует отметить, что ее команды построены так, чтобы значения в индексных регистрах можно было сравнивать с нулем, когда это только возможно. Например, в регистре rI1 сохраняется величина J-500, а не J. Особого внимания в этом смысле заслуживают строки 26–34, хотя разобраться в них, наверное, непросто.

Интересно отметить статистические характеристики, которые наблюдаются во время реальной работы программы P. Команда деления из строки 19 была выполнена 9 538 раз, а время выполнения строк 10–24 составило $182144u$.

Программы на языке MIXAL можно перфорировать на картах или набрать на терминале компьютера, как показано на рис. 15. При использовании перфокарт выбирается следующий формат.

| | |
|---|---|
| Колонки 1–10 | Поле МЕТКА |
| Колонки 12–15 | Поле ОП |
| Колонки 17–80 | Поле АДРЕС и необязательные примечания |
| Колонки 11, 16 | Пустые |

Но если в колонке 1 содержится звездочка, то все содержимое перфокарты рассматривается как комментарий. Поле АДРЕС заканчивается первой же пустой колонкой (пробелом), следующей за колонкой 16. Справа от этой первой колонки можно




```
* ПРИМЕР ПРОГРАММЫ...  ТАБЛИЦА ПРОСТЫХ ЧИСЕЛ
*
L EQU 500
PRINTER EQU 18
PRIME EQU -1
BUF0 EQU 2000
BUF1 EQU BUF0+25
 ORIG 3000
START IOC 0(PRINTER)
 LD1 =1-L=
```

**Рис. 15.** Первые строки программы P, перфорированные на картах либо набранные на терминале.

перфорировать любые комментарии, которые не влияют на транслируемую программу. (*Исключение.* Если в поле ОП содержится команда ALF, то примечания всегда начинаются в колонке 22.)

Когда входные данные вводятся с терминала, используется менее ограничительный формат: поле МЕТКА заканчивается первым же пробелом, а поля ОП и АДРЕС (если они есть) начинаются непустым символом и продолжаются до следующего пробела. В то же время за особым кодом операции ALF следуют либо два пробела и пять буквенно-цифровых символов, либо один пробел и пять буквенно-цифровых символов, первый из которых не является пробелом. В оставшейся части каждой строки могут содержаться примечания.

Ассемблер MIX берет подготовленные таким образом файлы и преобразует их в загрузочные модули программ на машинном языке. При благоприятном стечении обстоятельств читатель сможет получить доступ к ассемблеру MIX и имитатору MIX и проработать различные упражнения из этой книги.

Теперь вы знаете, что можно сделать с помощью языка MIXAL. В заключение этого раздела мы дадим более подробное описание правил и, в частности, обратим внимание на то, чего *не* разрешается делать на языке MIXAL. Фактически язык определяется сравнительно небольшим количеством правил, приведенных ниже.

1. *Символ* — это строка, содержащая от одной до десяти букв и/или цифр, среди которых должна быть по крайней мере одна буква. *Примеры* PRIME, TEMP, 20BY20. Специальные символы $d$H, $d$F и $d$B, где $d$ — это одна цифра, в целях данного определения будут заменяться другими уникальными символами в соответствии с соглашением о "локальных символах", о котором говорилось выше.

2. *Число* — это строка, содержащая от одной до десяти цифр. *Пример:* 00052.

3. В каждом случае появления в программе MIXAL символ называется либо определенным символом, либо ссылкой вперед. *Определенный символ* — это символ, который появляется в поле METKA одной из предыдущих строк программы MIXAL. *Ссылка вперед* — это символ, который пока еще не был определен подобным образом.

4. *Элементарное выражение* — это либо

a) число, либо

b) определенный символ (обозначающий числовой эквивалент этого символа, см. правило 13), либо

c) звездочка (обозначающая значение ⊛; см. правила 10 и 11).

5. *Выражение* — это либо

a) элементарное выражение, либо

b) элементарное выражение, перед которым стоит знак "+" или "−", либо

c) выражение, за которым следует бинарная операция, а за ней — элементарное выражение.

Допустимыми являются шесть следующих бинарных операций: +, -, *, /, // и :. Они определяются для числовых слов MIX следующим образом.

```
C = A+B     LDA AA; ADD BB; STA CC.
C = A-B     LDA AA; SUB BB; STA CC.
C = A*B     LDA AA; MUL BB; STX CC.
C = A/B     LDA AA; SRAX 5; DIV BB; STA CC.
C = A//B    LDA AA; ENTX 0; DIV BB; STA CC.
C = A:B     LDA AA; MUL =8=; SLAX 5; ADD BB; STA CC.
```

Здесь AA, BB и CC — ячейки, содержащие соответствующие значения символов A, B и C. Операции внутри выражения выполняются слева направо. *Примеры:*

| | |
|---|---|
| -1+5 | равно 4. |
| -1+5*20/6 | равно 4*20/6 равно 80/6 равно 13 (операции выполняются слева направо). |
| 1//3 | равно слову MIX, размер которого приблизительно равен $b^5/3$, где $b$ — размер байта; т. е. слово, представляющее дробь $\frac{1}{3}$ с десятичной точкой слева. |
| 1:3 | равно 11 (обычно используется в частичной спецификации поля). |
| *-3 | равно ⊛ минус три. |
| *** | равно ⊛, умноженному на ⊛. |

6. *A-часть* (которая используется для описания адресного поля команды MIX) либо

a) пуста (и обозначает нулевое значение), либо

b) является выражением, либо

c) является ссылкой вперед (и обозначает окончательный эквивалент символа; *см. правило* 13), *либо*

d) является литералом (и обозначает ссылку на внутренний символ; см. правило 12).

7. *Индексная часть* (которая используется для описания индексного поля команды MIX), либо

a) пуста (и обозначает нулевое значение), либо

b) состоит из запятой и следующего за ней выражения (и обозначает значение этого выражения).

8. *F-часть* (которая используется для описания F-поля команды MIX) либо

a) пуста (что обозначает стандартное F-значение в зависимости от содержимого поля ОП (см. табл. 1.3.1–1)), либо

b) состоит из выражения, заключенного в круглые скобки (и обозначает значение этого выражения).

9. *W-значение* (которое используется для описания константы MIX, занимающей *полное слово*) — это *либо*

a) выражение, за которым следует F-часть (в этом случае пустая F-часть обозначается через (0:5)), либо

b) W-значение, за которым после запятой следует W-значение вида (a).

W-значение указывает числовое значение слова MIX, которое определяется следующим образом. Пусть W-значение имеет вид "$E_1(F_1),E_2(F_2),\ldots,E_n(F_n)$", где $n \geq 1$, $E_i$ — выражения, а $F_i$ — поля. Желаемый результат — окончательное значение, которое появилось бы в ячейке памяти WVAL после выполнения следующей гипотетической программы:

$$\texttt{STZ WVAL; LDA } C_1\texttt{; STA WVAL(}F_1\texttt{); ...; LDA } C_n\texttt{; STA WVAL(}F_n\texttt{).}$$

Здесь $C_1, \ldots, C_n$ обозначают ячейки, содержащие значения выражений $E_1, \ldots, E_n$. Каждое $F_i$ должно иметь вид $8L_i + R_i$, где $0 \leq L_i \leq R_i \leq 5$. *Примеры:*

| | | |
|---|---|---|
| 1 | слово | +\|\|1 |
| 1,-1000(0:2) | слово | -\|1000\|1 |
| -1000(0:2),1 | слово | +\|\|1 |

10. В процессе трансляции используется величина, которая обозначается через ✱ (и называется *счетчиком адреса*). Первоначальное значение счетчика адреса равно нулю. Значение ✱ всегда должно быть неотрицательным числом, которое помещается в двух байтах. Если в строке поле метки не пусто, то оно должно содержать символ, который не был определен ранее. Эквивалент этого символа затем определяется как текущее значение ✱.

11. После обработки поля METKA, как описано в правиле 10, процесс трансляции будет зависеть от значения содержимого поля ОП. Существует шесть возможностей для ОП.

a) В поле ОП содержится символический оператор MIX. В табл. 1 из предыдущего раздела определены стандартные значения C и F для каждого оператора MIX. В этом случае в поле АДРЕС должна находиться A-часть (правило 6), за которой следует индексная часть (правило 7), а затем — F-часть (правило 8). Таким образом, получаем четыре значения: C, F, A и I. В результате транслируется слово, которое определяется последовательностью "LDA C; STA WORD; LDA F; STA WORD(4:4); LDA I; STA WORD(3:3); LDA A; STA WORD(0:2)" и помещается в ячейку, заданную ⊛, а затем увеличивается на 1 значение счетчика ⊛.

b) В поле ОП содержится операция "EQU". В поле АДРЕС должно содержаться W-значение (см. правило 9). Если поле METKA не пусто, то значение содержащегося здесь символа устанавливается равным значению, заданному в поле АДРЕС. Это правило имеет более высокий приоритет, чем правило 10. Значение ⊛ не меняется. (В качестве нетривиального примера рассмотрим строку

```
BYTESIZE  EQU  1(4:4),
```

позволяющую программисту получить символ, значение которого зависит от размера байта. Эта ситуация допустима до тех пор, пока программа имеет смысл для всех возможных размеров байта.)

c) В поле ОП находится "ORIG". В поле АДРЕС должно содержаться W-значение (см. правило 9); значение счетчика адреса ⊛ устанавливается равным этому значению. (Заметьте, что согласно правилу 10 символ, находящийся в поле METKA строки с операцией ORIG, принимает значение ⊛ *до* его изменения. Например,

```
TABLE    ORIG  *+100
```

делает символ TABLE эквивалентным текущему адресу плюс 100.)

d) В поле ОП находится "CON". В поле АДРЕС должно содержаться W-значение. В результате происходит трансляция слова, имеющего это значение, помещение его в ячейку, заданную ⊛, и увеличение значения счетчика ⊛ на 1.

e) В поле ОП находится "ALF". В результате выполняется трансляция слова из символьных кодов, образуемого первыми пятью символами адресного поля; в остальном операция аналогична CON.

f) В поле ОП находится "END". В поле ADDRESS должно содержаться W-значение, определяющее в своем поле (4:5) адрес команды, с которой начинается программа. Строка END обозначает окончание программы на языке MIXAL. В завершение ассемблер вставляет в произвольном порядке непосредственно перед строкой END дополнительные строки, соответствующие всем неопределенным символам и литеральным константам (см. правила 12 и 13). Таким образом, символ в поле METKA строки END будет обозначать первую ячейку, следующую за вставленными словами.

12. Литеральные константы. W-значение, длина которого — менее 10 символов, можно заключить между знаками "=" и использовать в качестве ссылки вперед.

В результате будет создан новый внутренний символ и сразу перед строкой END будет вставлена строка CON, определяющая этот символ (см. примечание 4 после программы P).

13. Каждому символу соответствует одно и только одно значение. Это число, занимающее полное слово MIX, обычно определяется символом из поля METKA в соответствии с правилом 10 или 11, (b). Если этого символа не было в поле METKA, то перед строкой END вставляется новая строка, у которой ОП = "CON", АДРЕС = "0" и в поле LOC которой содержится имя символа.

*Замечание.* Самым важным следствием из приведенных выше правил является ограничение на ссылки вперед. Для этого нельзя использовать символ, который еще не был определен в поле LOC одной из предыдущих строк; его можно применять только в качестве A-части команды. В частности, этот символ нельзя использовать (a) в связи с арифметическими операциями или (b) в поле АДРЕС операций EQU, ORIG и CON. Например, операции

```
LDA  2F+1
```

и

```
CON  3F
```

недопустимы. Это ограничение было наложено для того, чтобы обеспечить более эффективную трансляцию программ. Кроме того, опыт, полученный в процессе написания данной серии книг, показал, что это очень мягкое ограничение, которое редко имеет сколько-нибудь существенное значение.

На самом деле у MIX есть два символических языка программирования низкого уровня: MIXAL*, машинно-ориентированный язык, предназначенный для облегчения трансляции за один проход с помощью очень простого ассемблера, и PL/MIX, который более адекватно отражает информационные и управляющие структуры и выглядит, как поле примечаний программ на языке MIXAL. PL/MIX будет описан в главе 10.

## УПРАЖНЕНИЯ (часть 1)

**1.** [*00*] В тексте раздела отмечалось, что запись "X EQU 1000" не генерирует машинной команды, которая присваивает значение переменной. Предположим, вы пишете программу для MIX, в которой хотите присвоить значение, равное 1 000, некоторой ячейке памяти (с символическим именем X). Как это сделать на языке MIXAL?

▸ **2.** [*10*] Строка 12 программы M выглядит так: "JMP *", где * — метка этой же строки. Почему программа не зацикливается, вновь и вновь повторяя эту команду?

▸ **3.** [*23*] Каким будет результат выполнения следующей программы, если она используется вместе с программой M?

```
START IN   X+1(0)
      JBUS *(0)
      ENT1 100
1H    JMP  MAXIMUM
      LDX  X,1
```

* Автор был крайне удивлен, когда в 1971 году узнал о том, что MIXAL — это еще и название югославского стирального порошка для *автоматических* стиральных машин.

```
        STA  X,1
        STX  X,2
        DEC1 1
        J1P  1B
        OUT  X+1(1)
        HLT
        END  START  ▌
```

▶ **4.** [*25*] Выполните трансляцию программы P вручную. (Это займет не так много времени, как вы думаете.) Какие числа будут фактически содержаться в оперативной памяти в соответствии с этой символической программой?

**5.** [*11*] Почему в программе P не нужна команда JBUS для определения времени готовности АЦПУ?

**6.** [*HM20*] (a) Покажите, что если $n$ не является простым числом, то $n$ имеет делитель $d$, такой, что $1 < d \leq \sqrt{n}$. (b) Учитывая этот факт, докажите, что проверка на шаге P7 алгоритма P показывает, что N — простое число.

**7.** [*10*] (a) Что означает "4B" в строке 34 программы P? (b) Каким будет эффект (если он вообще будет), если метку строки 15 заменить меткой "2H", а адрес строки 20 — адресом "2B"?

▶ **8.** [*24*] Что делает следующая программа? (Не запускайте ее на компьютере, найдите ответ "вручную"!)

```
        * ЗАГАДОЧНАЯ ПРОГРАММА
BUF     ORIG *+3000
1H      ENT1 1
        ENT2 0
        LDX  4F
2H      ENT3 0,1
3H      STZ  BUF,2
        INC2 1
        DEC3 1
        J3P  3B
        STX  BUF,2
        INC2 1
        INC1 1
        CMP1 =75=
        JL   2B
        ENN2 2400
        OUT  BUF+2400,2(18)
        INC2 24
        J2N  *-2
        HLT
4H      ALF  AAAAA
        END  1B  ▌
```

## УПРАЖНЕНИЯ (часть 2)

Эти упражнения представляют собой небольшие задачи по программированию, иллюстрирующие типичные примеры применения компьютеров и охватывающие широкий диапазон различных методов. Читателю настоятельно рекомендуется решить хотя бы несколько из этих задач, чтобы получить некоторый опыт использования MIX, а также

достаточно полное представление об основах искусства программирования. Если хотите, можете проработать эти упражнения одновременно с чтением оставшейся части главы 1.

Ниже перечислены используемые в задачах методы программирования.

Использование таблиц-переключателей для многовариантных решений: упр. 9, 13 и 23.

Использование индексных регистров и двумерных массивов: упр. 10, 21 и 23.

Распаковка символов: упр. 13 и 23.

Целочисленная и десятичная арифметика и масштабирование: упр. 14, 16 и 18.

Использование подпрограмм: упр. 14 и 20.

Буферизация ввода: упр. 13.

Буферизация вывода: упр. 21 и 23.

Обработка списков: упр. 22.

Управление в реальном времени: упр. 20.

Графическое отображение: упр. 23.

Каждый раз, когда в упражнении из этой книги говорится "напишите программу для MIX" или "напишите подпрограмму для MIX", следует написать только символический код на языке MIXAL для решения требуемой задачи. Этот код будет представлять собой не законченную программу, а только фрагмент полной (гипотетической) программы. В фрагменте кода не нужно выполнять ввод или вывод данных, если они должны быть взяты из внешних источников. Необходимо использовать только следующие поля строк программы на языке MIXAL: МЕТКА, ОП и АДРЕС, а также снабдить их соответствующими комментариями. Числовое представление транслированных машинных команд, номера строк и колонки с итоговым количеством выполнений различных команд (см. программу M) указывать не нужно, за исключением случаев, когда это оговаривается особо. Строка END тоже не нужна.

С другой стороны, если в упражнении говорится "напишите *полную* программу для MIX", значит, нужно написать выполняемую программу на языке MIXAL, которая, в частности, должна включать финальную строку END. Ассемблеры и имитаторы MIX, на которых можно протестировать полные программы, широко распространены, поэтому проблем возникнуть не должно.

▸ **9.** [*25*] В ячейке INST содержится слово MIX, которое предположительно является командой MIX. Напишите программу для MIX, в которой совершается переход к ячейке GOOD, если это слово имеет допустимое C-поле, допустимое ±AA-поле, допустимое I-поле и допустимое F-поле в соответствии с табл. 1.3.1–1. В противном случае в программе должен быть выполнен переход к ячейке BAD. Помните, что допустимость F-поля зависит от C-поля — например, если C = 7 (MOVE), то допустимо любое F-поле, но если C = 8 (LDA), то F-поле должно иметь вид 8L + R, где $0 \le L \le R \le 5$. ±AA-поле считается допустимым, за *исключением* случая, когда C определяет команду, запрашивающую адрес памяти, I = 0 и ±AA не является допустимым адресом памяти.

*Замечание.* Неопытные программисты склонны решать подобную задачу, программируя длинные серии тестов C-поля, такие как "LDA C; JAZ 1F; DECA 5; JAN 2F; JAZ 3F; DECA 2; JAN 4F; ...". Этот подход никуда *не* годится! Лучший способ выбрать один из многих вариантов — подготовить вспомогательную *таблицу*, содержащую информацию, в которой сосредоточена необходимая логическая схема. Если бы эта таблица содержала, например, 64 элемента, мы бы написали "LD1 C; LD1 TABLE,1; JMP 0,1" и в результате очень быстро перешли бы к нужной программе. В подобной таблице можно хранить и другую полезную информацию. Табличный подход к решению данной задачи лишь ненамного увеличивает длину программы (из-за включения таблицы), но зато существенно повышает скорость ее выполнения и делает более универсальной.

▸ **10.** [*31*] Пусть имеется матрица размера $9 \times 8$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{18} \\ a_{21} & a_{22} & a_{23} & \dots & a_{28} \\ \vdots & & & & \vdots \\ a_{91} & a_{92} & a_{93} & \dots & a_{98} \end{pmatrix},$$

которая хранится в памяти так, что $a_{ij}$ находится в ячейке $1000 + 8i + j$. Поэтому ячейки памяти, в которых хранится эта матрица, выглядят следующим образом:

$$\begin{pmatrix} (1009) & (1010) & (1011) & \dots & (1016) \\ (1017) & (1018) & (1019) & \dots & (1024) \\ \vdots & & & & \vdots \\ (1073) & (1074) & (1075) & \dots & (1080) \end{pmatrix}$$

Говорят, что матрица имеет "седловую точку", если некоторый элемент является минимальным значением в строке и максимальным — в столбце. Математически это можно выразить так: элемент $a_{ij}$ является седловой точкой данной матрицы, если

$$a_{ij} = \min_{1 \le k \le 8} a_{ik} = \max_{1 \le k \le 9} a_{kj}.$$

Напишите программу для MIX, которая определяет адрес седловой точки (если в матрице есть по крайней мере одна такая точка) или выдает нулевое значение (если седловой точки нет) и заканчивает работу, поместив найденный результат в rI1.

**11.** [*M29*] Чему равна *вероятность* того, что матрица из предыдущего упражнения имеет седловую точку, если ее 72 элемента различны и все $72!$ варианта одинаково возможны? Чему будет равна соответствующая вероятность, если матрица состоит только из нулей и единиц и все $2^{72}$ вариантов таких матриц являются одинаково возможными?

**12.** [*HM42*] К упр. 10 даны два решения (см. с. 570) и предложено найти еще одно, но не ясно какое из них лучше. Проанализируйте эти алгоритмы на основании предположений из упр 11 и решите, какой метод лучше.

**13.** [*28*] Дешифровщику необходимо подсчитать частоту появления букв в некотором закодированном тексте. Этот текст перфорирован на бумажной ленте; о его окончании сигнализирует символ "звездочка" Напишите полную программу для MIX, которая считывает данные с перфоленты, подсчитывает частоту появления каждого символа вплоть до первой звездочки, а затем печатает результаты в виде

```
A    0010257
B    0000179
D    0794301
```

и т д . по одному символу в строке Не нужно подсчитывать количество пробелов, а также печатать результаты для символов, частота появления которых равна нулю (например, для таких, как символ C в нашем примере) В целях эффективности используйте "буферизацию" при вводе во время считывания блока в одну область памяти можно выполнять подсчет символов в другой области Будем предполагать, что на ленте, содержащей входные данные, есть еще один дополнительный блок (следующий за тем, в котором находится заключительная звездочка).

▸ **14.** [*31*] Следующий алгоритм, который был получен неаполитанским астрономом Алоизием Лилиусом (Aloysius Lilius) и немецким математиком иезуитом Кристофером Клавиусом (Christopher Clavius) в конце 16 века, используется большинством западных христианских церквей для определения даты пасхального воскресенья для любого года после 1582.

**Алгоритм E** (*Определение даты пасхи*). Пусть $Y$ — это год, для которого нужно получить дату пасхи.

**E1.** [Золотое число.] Присвоить $G \leftarrow (Y \bmod 19)+1$. ($G$ — это так называемое "золотое число" года в 19-летнем лунно-солнечном цикле.)

**E2.** [Столетие.] Присвоить $C \leftarrow \lfloor Y/100 \rfloor + 1$. (Если $Y$ не кратно 100, то $C$ — номер столетия, например 1984 год принадлежит двадцатому столетию.)

**E3.** [Поправки.] Присвоить $X \leftarrow \lfloor 3C/4 \rfloor - 12$, $Z \leftarrow \lfloor (8C+5)/25 \rfloor - 5$. (Здесь $X$ — число годов, таких как 1900, когда дополнительный день високосного года не добавляется, чтобы "идти в ногу" с Солнцем; $Z$ — специальная поправка, предназначенная для синхронизации даты пасхи с орбитой Луны.)

**E4.** [Поиск воскресенья.] Присвоить $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$. [$((-D) \bmod 7)$ марта действительно будет воскресеньем.]

**E5.** [Эпакта.] Присвоить $E \leftarrow (11G + 20 + Z - X) \bmod 30$. Если $E = 25$ и золотое число $G$ больше 11 или если $E = 24$, то увеличить $E$ на 1. (Это число $E$ — *эпакта*, которая определяет дату полнолуния.)

**E6.** [Поиск полнолуния.] Присвоить $N \leftarrow 44 - E$. Если $N < 21$, то присвоить $N \leftarrow N + 30$. (Считается, что пасха — это первое воскресенье, следующее за первым полнолунием, которое произошло не ранее 21 марта. На самом деле это определение не является абсолютно точным из-за возмущений лунной орбиты, но в данном случае нас интересует не реальная, а "календарная" луна. $N$-е марта — это календарное полнолуние.)

**E7.** [Перейти к воскресенью.] Присвоить $N \leftarrow N + 7 - ((D + N) \bmod 7)$.

**E8.** [Определить месяц.] Если $N > 31$, то искомой датой будет $(N - 31)$ `АПРЕЛЯ`; в противном случае — $N$ `МАРТА`. ▮

Напишите подпрограмму вычисления и печати даты пасхи для заданного года, предполагая, что номер этого года не превышает 100000. Выходные данные должны иметь вид "*dd* `МЕСЯЦ`, *yyyyy*", где *dd* — день, а *yyyyy* — год. Напишите полную программу для `MIX`, в которой указанная подпрограмма используется для построения таблицы дат пасхи с 1950 по 2000 год.

**15.** [*M30*] Достаточно распространенная ошибка при решении предыдущего упражнения состоит в непонимании того, что величина $(11G + 20 + Z - X)$ на шаге E5 может быть отрицательной; в результате положительный остаток mod 30 может быть вычислен неправильно. (См. *CACM* **5** (1962), 556.) Например, для года 14250 мы нашли бы $G = 1$, $X = 95$, $Z = 40$, поэтому, если бы у нас было $E = -24$ вместо $E = +6$, мы получили бы нелепый результат: "`42 АПРЕЛЯ`". Напишите полную программу для `MIX`, определяющую *самый ранний* год, для которого эта ошибка стала бы причиной неправильного вычисления даты пасхи.

**16.** [*31*] В разделе 1.2.7 было показано, что ряд $1 + \frac{1}{2} + \frac{1}{3} + \cdots$ расходится. Но если эту сумму вычислить на компьютере с ограниченной точностью, то в некотором смысле можно получить сходящуюся сумму, так как в конце концов ее члены окажутся настолько малыми, что уже ничего не будут добавлять к сумме. Для примера вычислим сумму с точностью до одного десятичного знака; получим $1 + 0.5 + 0.3 + 0.3 + 0.2 + 0.2 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 = 3.9$.

А если говорить более строго, пусть $r_n(x)$ — это число $x$, округленное с точностью до $n$ десятичных знаков; тогда $r_n(x) = \lfloor 10^n x + \frac{1}{2} \rfloor / 10^n$ Теперь нужно найти

$$S_n = r_n(1) + r_n(\tfrac{1}{2}) + r_n(\tfrac{1}{3}) + \cdots.$$

Известно, что $S_1 = 3.9$ и задача состоит в том, чтобы написать полную программу для MIX, которая вычисляет и печатает значения $S_n$ для $n = 2$, 3, 4 и 5

*Замечание.* Для этого существует гораздо более быстрый способ, чем простая процедура добавления членов $r_n(1/m)$ по одному, пока $r_n(1/m)$ не обратится в нуль Например, для всех значений $m$ от 66667 до 200000 имеем $r_5(1/m) = 0.00001$, и значит, все 133334 раза можно избежать вычисления $1/m$! Необходимо использовать алгоритм, содержащий следующие строки.

A Начать с $m_h = 1$, $S = 1$.

B Присвоить $m_e = m_h + 1$ и вычислить $r_n(1/m_e) = r$.

C. Найти $m_h$ — наибольшее $m$, для которого $r_n(1/m) = r$

D. Добавить $(m_h - m_e + 1)r$ к $S$ и вернуться к шагу B.

**17.** [*HM30*] Используя обозначения из предыдущего упражнения, докажите или опровергните формулу

$$\lim_{n\to\infty}(S_{n+1} - S_n) = \ln 10$$

**18.** [*25*] Возрастающая последовательность всех несократимых дробей между 0 и 1, знаменатели которых не превосходят $n$, называется рядом Фарея порядка $n$ Например, рядом Фарея порядка 7 является последовательность

$$\frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1}.$$

Если обозначить члены этого ряда через $x_0/y_0$, $x_1/y_1$, $x_2/y_2$, ..., то из упр. 19 следует, что

$$x_0 = 0, \quad y_0 = 1, \qquad x_1 = 1, \quad y_1 = n;$$
$$x_{k+2} = \lfloor (y_k + n)/y_{k+1} \rfloor x_{k+1} - x_k,$$
$$y_{k+2} = \lfloor (y_k + n)/y_{k+1} \rfloor y_{k+1} - y_k.$$

Напишите подпрограмму для MIX, которая вычисляет ряд Фарея порядка $n$, сохраняя значения $x_k$ и $y_k$ в ячейках $\mathtt{X} + k$, $\mathtt{Y} + k$ соответственно (Общее количество членов этого ряда приблизительно равно $3n^2/\pi^2$, поэтому можно предполагать, что $n$ достаточно мало́ )

**19.** [*M30*] (a) Покажите, что числа $x_k$ и $y_k$, которые определяются рекуррентным соотношением из предыдущего упражнения, удовлетворяют равенству $x_{k+1}y_k - x_k y_{k+1} = 1$. (b) На основании факта, доказанного в п. (a), покажите, что дроби $x_k/y_k$ действительно являются членами ряда Фарея порядка $n$

▸ **20.** [*33*] Предположим, что флаг переполнения и регистр X машины MIX подключены к светофору, расположенному на углу проспекта Дель-Мар (Del Mar Boulevard) и Беркли-авеню (Berkeley Avenue), следующим образом:

rX(2:2) = светофор для транспорта на Дель-Мар
rX(3:3) = светофор для транспорта на Беркли
} 0 — выключен, 1 — зеленый, 2 — желтый, 3 — красный;

rX(4.4) = сигнал для пешеходов на Дель-Мар
rX(5·5) = сигнал для пешеходов на Беркли
} 0 — выключен, 1 — "ИДИТЕ", 2 — "СТОЙТЕ".

Машины или пешеходы, которые хотят, двигаясь по Беркли, пересечь проспект, должны включить переключатель, который установит флаг переполнения MIX в положение 1 Если такая ситуация не возникнет, то сигнал светофора для Дель-Мар всегда будет оставаться зеленым.

При этом установлены следующие временные интервалы.

Зеленый свет для транспорта на Дель-Мар $\geq 30$ с, желтый — 8 с.
Зеленый свет для транспорта на Беркли — 20 с, желтый — 5 с.

Когда в одном направлении для транспорта горит зеленый или желтый сигнал светофора, в другом направлении горит красный свет. Когда транспорту дан зеленый свет, то включен соответствующий сигнал СТОЙТЕ, только перед переключением зеленого света на желтый сигнал СТОЙТЕ мигает в течение 12 с по следующей схеме циклов:

$$\left.\begin{array}{ll}\texttt{СТОЙТЕ} & \frac{1}{2}\text{ с} \\ \text{Отключен} & \frac{1}{2}\text{ с}\end{array}\right\}\ \text{повторяется 8 раз;}$$

СТОЙТЕ 4 с (и остается во время циклов желтого и красного света).

Если флаг переполнения включился, когда на Беркли горит зеленый сигнал, то машина или пешеход пройдет в этом же цикле, но если это случилось во время цикла желтого или красного света, то придется ждать следующего цикла, который наступит после того, как проедет поток машин по Дель-Мар.

Пусть единица времени для MIX составляет 10 $\mu$sec. Напишите полную программу для MIX, которая управляет сигналами светофора с помощью rX, в зависимости от того, что подано на вход флага переполнения. Необходимо неукоснительно придерживаться установленных промежутков времени; исключение может быть сделано только для тех случаев, когда это невозможно. *Замечание.* Значение в rX меняется точно в момент *завершения* выполнения команды LDX или INCX.

**21.** [*28*] *Магический квадрат порядка* $n$ — это такое расположение в квадрате чисел от 1 до $n^2$, при котором сумма элементов и по вертикали, и по горизонтали равна $n(n^2+1)/2$; такой же результат дает сумма элементов двух главных диагоналей. На рис. 16 показан магический квадрат порядка 7. Для его составления используется следующее правило. Начните с 1, вставив ее в клетку, расположенную прямо под "центральной" клеткой (см. рис. 16), затем двигайтесь вниз и вправо по диагонали (при выходе за границу квадрата представляйте себе, что вся плоскость выложена подобными квадратами), пока не достигнете заполненной клетки; затем опуститесь вниз на две клетки от последней заполненной клетки и продолжайте движение по диагонали вниз и вправо. Этот метод подходит для любого нечетного $n$.

Используя тот же принцип распределения памяти, что и в упр. 10, напишите полную программу для MIX, которая генерирует магический квадрат размера $23 \times 23$ описанным выше методом и печатает результат. [Этот алгоритм принадлежит Мануэлю Мосхопулосу (Manuel Moschopoulos), который жил в Константинополе приблизительно в 1300 году. Другие многочисленные и не менее интересные методы построения магических квадратов, которые представляют собой хорошие упражнения по программированию, можно найти в книге W. W. Rouse Ball, *Mathematical Recreations and Essays*, revised by H. S. M. Coxeter (New York: Macmillan, 1939), Chapter 7.]

**22.** [*31*] (*Задача Иосифа.*) Пусть $n$ человек стоят по кругу. Начиная с некоторой позиции, они ведут отсчет по кругу и предают жестокой казни каждого $m$-го человека; после каждой казни круг смыкается. Например (рис. 17), при $n = 8$ и $m = 4$ казнить будут в следующем порядке: 54613872. Первого человека казнят пятым, второго — четвертым и т. д. Напишите полную программу для MIX, которая распечатывает порядок выполнения казней для $n = 24$, $m = 11$. Постарайтесь придумать эффективный алгоритм, который быстро работает при больших $n$ и $m$ (однажды это может спасти вам жизнь). [*Ссылки.* W. Ahrens, *Mathematische Unterhaltungen und Spiele* **2** (Leipzig: Teubner, 1918), Chapter 15.]

| | | | | | | |
|---|---|---|---|---|---|---|
| 22 | 47 | 16 | 41 | 10 | 35 | 04 |
| 05 | 23 | 48 | 17 | 42 | 11 | 29 |
| 30 | 06 | 24 | 49 | 18 | 36 | 12 |
| 13 | 31 | 07 | 25 | 43 | 19 | 37 |
| 38 | 14 | 32 | 01 | 26 | 44 | 20 |
| 21 | 39 | 08 | 33 | 02 | 27 | 45 |
| 46 | 15 | 40 | 09 | 34 | 03 | 28 |

**Рис. 16.** Магический квадрат.

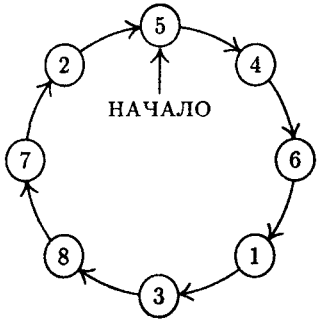



**Рис. 17.** Задача Иосифа, $n = 8$, $m = 4$.

**23.** [*37*] Цель этого упражнения — помочь читателю получить опыт применения компьютеров в сфере, где выходные данные должны быть отображены графически, а не в традиционном табличном виде. В данном случае необходимо "нарисовать" схему кроссворда.

Входными данными является матрица, состоящая из нулей и единиц. Нуль соответствует белому квадратику, а единица — черному. В качестве выходных данных нужно получить схему кроссворда, в котором соответствующие квадратики с номерами обозначают слова, расположенные по вертикали и по горизонтали.

Например, для матрицы

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

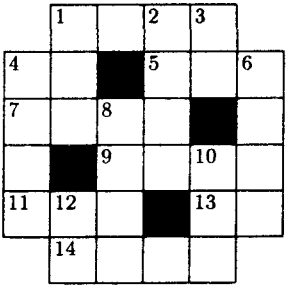



**Рис. 18.** Схема кроссворда, соответствующая матрице из упр. 23.

соответствующая схема кроссворда должна выглядеть, как показано на рис. 18. Квадратик нумеруется, если он белый и либо (a) под ним расположен белый квадратик и прямо над ним нет белого квадратика, либо (b) справа находится белый квадрат и непосредственно слева нет белого квадрата. Если черный квадратик оказался у края, то его нужно удалить из схемы. Это проиллюстрировано на рис. 18, где черные квадратики по углам были удалены. Существует простой способ достижения этой цели — "искусственно" вставить строки и столбцы, содержащие $-1$ сверху, снизу и по боковым сторонам заданной входной матрицы, а затем поменять каждую $+1$, которая соседствует с $-1$, на $-1$, пока рядом с любой $-1$ не останется ни одной $+1$.

Для печати окончательной схемы на АЦПУ следует использовать следующий метод. Каждый квадратик кроссворда должен отображаться на печатной странице с помощью 5 столбцов и 3 строк, причем эти 15 позиций заполняются следующим образом.

| | | | | | |
|---|---|---|---|---|---|
| Непронумерованные | ␣␣␣␣+ | Номер nn | nn␣␣+ | Черные | +++++ |
| белые квадраты: | ␣␣␣␣+ | белого квадрата: | ␣␣␣␣+ | квадраты: | +++++ |
| | +++++ | | +++++ | | +++++ |

Квадраты с $-1$ в зависимости от того, справа или снизу находятся $-1$:

| | | | | |
|---|---|---|---|---|
| ␣␣␣␣+ | ␣␣␣␣+ | ␣␣␣␣␣ | ␣␣␣␣␣ | ␣␣␣␣␣ |
| ␣␣␣␣+ | ␣␣␣␣+ | ␣␣␣␣␣ | ␣␣␣␣␣ | ␣␣␣␣␣ |
| +++++ | ␣␣␣␣+ | +++++ | ␣␣␣␣+ | ␣␣␣␣␣ |

Схема, изображенная на рис. 18, будет напечатана, как показано на рис. 19.

Ширины строки принтера — 120 символов — достаточно для того, чтобы печатать кроссворды, содержащие до 23 столбцов. В качестве входных данных должна быть предоставлена матрица размера 23 × 23, состоящая из нулей и единиц, все строки которой перфорируются в столбцах 1–23 входной перфокарты. Например, карта, соответствующая первой строке приведенной выше матрицы, будет перфорирована следующим образом: "10000111111111111111111". Схема кроссворда необязательно будет симметричной, и у нее могут оказаться длинные ленты черных квадратиков, присоединенные к наружным сторонам кроссворда самым экзотическим образом.

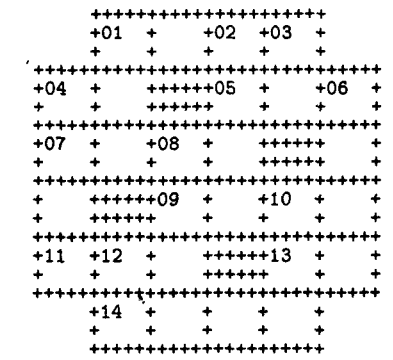



**Рис. 19.** Распечатка на АЦПУ кроссворда, показанного на рис. 18.

### 1.3.3. Применение к перестановкам

В данном разделе мы приведем еще несколько примеров программ для MIX и наряду с этим познакомимся с некоторыми важными свойствами перестановок. Эти исследования позволят также выявить интересные аспекты компьютерного программирования в целом.

Перестановки уже обсуждались в разделе 1.2.5; в нем перестановка $c\,d\,f\,b\,e\,a$ рассматривалась как некоторое *расположение* шести объектов $a$, $b$, $c$, $d$, $e$, $f$ в ряд. Но возможна и другая точка зрения. Перестановку можно рассматривать как *переупорядочение или переименование объектов*. При такой интерпретации* обычно используют двухстрочную запись, например

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}. \tag{1}$$

Это означает, что "$a$ переходит в $c$, $b$ переходит в $d$, $c$ переходит в $f$, $d$ переходит в $b$, $e$ переходит в $e$, $f$ переходит в $a$". С точки зрения переупорядочения это означает, что объект $c$ переходит на место, которое ранее занимал объект $a$. А если рассматривать это как переименование, то можно считать, что объект $a$ переименован в $c$. При использовании двухстрочной записи изменение порядка столбцов в перестановке никакой роли не играет. Например, перестановку (1) можно записать в виде

$$\begin{pmatrix} c & d & f & b & a & e \\ f & b & a & d & c & e \end{pmatrix},$$

а также еще 718 способами.

В связи с описанной интерпретацией часто используют *циклическую запись*. Перестановку (1) можно записать и в виде

$$(a\,c\,f)\,(b\,d), \tag{2}$$

---

* В русскоязычной математической литературе в этом случае часто используют термин "подстановка". — *Прим. перев.*

что, опять же, означает "$a$ переходит в $c$, $c$ переходит в $f$, $f$ переходит в $a$, $b$ переходит в $d$, $d$ переходит в $b$". Цикл $(x_1\, x_2 \ldots x_n)$ означает "$x_1$ переходит в $x_2$, $\ldots$, $x_{n-1}$ переходит в $x_n$, $x_n$ переходит в $x_1$". Так как элемент $e$ является в перестановке фиксированным (т. е. переходит не в какой-либо другой элемент, а в самого себя), он не появляется в циклической записи. Таким образом, единичные циклы типа "$(e)$" записывать не принято. Если в перестановке фиксированными *являются все элементы, так что присутствуют только единичные циклы, то она* называется *тождественной перестановкой* и обозначается "()".

Запись перестановки в виде цикла не является единственной. Например,

$$(b\,d)\,(a\,c\,f), \qquad (c\,f\,a)\,(b\,d), \qquad (d\,b)\,(f\,a\,c) \tag{3}$$

и т. д. эквивалентны (2). Но запись "$(a\,f\,c)\,(b\,d)$" уже не будет им эквивалентна, так как она означает, что $a$ переходит в $f$.

Легко видеть, почему перестановку всегда можно представить в виде цикла. Начиная с элемента $x_1$, перестановка переводит, скажем, $x_1$ в $x_2$, $x_2$ в $x_3$ и т. д., пока наконец (поскольку количество элементов ограничено) мы не придем к некоему элементу $x_{n+1}$, который уже встречался среди $x_1, \ldots, x_n$. Этот элемент $x_{n+1}$ должен быть равен $x_1$. Предположим, он равен $x_3$. Но это невозможно, так как известно, что $x_2$ переходит в $x_3$, а по предположению в $x_{n+1}$ переходит $x_n \neq x_2$. Поэтому $x_{n+1} = x_1$ и получаем цикл $(x_1\, x_2 \ldots x_n)$, являющийся частью перестановки для некоторого $n \geq 1$. Если этим циклом вся перестановка не исчерпывается, то существует элемент $y_1$, такой, что $y_1 \neq x_i$ для всех $i$, для которого аналогичным образом получим еще один цикл $(y_1\, y_2 \ldots y_m)$. Ни один $y_i$ не может равняться любому $x_i$, так как из $x_i = y_j$ следует, что $x_{i+1} = y_{j+1}$ и т. д. В конце концов для некоторого $k$ получим $x_k = y_1$, что противоречит предположению о выборе $y_1$. Действуя подобным образом, можно найти все циклы, содержащиеся в перестановке.

В программировании эти понятия применяются каждый раз, когда некоторый набор из $n$ объектов нужно расположить в другом порядке. Чтобы переупорядочить объекты, не перемещая их в какое-либо другое место, необходимо, в сущности, придерживаться циклической структуры. Например, чтобы переупорядочить (1), т. е. присвоить

$$(a,\, b,\, c,\, d,\, e,\, f) \leftarrow (c,\, d,\, f,\, b,\, e,\, a),$$

будем следовать циклической структуре (2) и последовательно присваивать

$$t \leftarrow a, \quad a \leftarrow c, \quad c \leftarrow f, \quad f \leftarrow t; \quad t \leftarrow b, \quad b \leftarrow d, \quad d \leftarrow t.$$

Но нужно отдавать себе отчет в том, что любое подобное преобразование можно осуществлять в непересекающихся циклах.

**Произведения перестановок.** Две перестановки можно перемножить в том смысле, что их произведение означает применение одной перестановки вслед за другой. Например, если за перестановкой (1) следует перестановка

$$\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix}$$

то получим, что $a$ переходит в $c$, которое затем переходит в $c$; $b$ переходит в $d$, которое затем переходит в $a$ и т. д.:

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix}$$
$$= \begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \times \begin{pmatrix} c & d & f & b & e & a \\ c & a & e & d & f & b \end{pmatrix}$$
$$= \begin{pmatrix} a & b & c & d & e & f \\ c & a & e & d & f & b \end{pmatrix}. \tag{4}$$

Вполне очевидно, что произведение перестановок не обладает свойством коммутативности; другими словами, $\pi_1 \times \pi_2$ необязательно равно $\pi_2 \times \pi_1$, где $\pi_1$ и $\pi_2$ — перестановки. Читатель может убедиться в том, что, если поменять местами сомножители, произведение в (4) даст другой результат (см. упр. 3).

Кое-кто перемножает перестановки справа налево, вместо того чтобы делать это более естественным образом слева направо, как показано в (4). Фактически математики разделились на два лагеря: одни считают, что результат последовательного применения преобразований $T_1$ и $T_2$ следует обозначать через $T_1T_2$, а другие — через $T_2T_1$. Здесь мы будем использовать обозначение $T_1T_2$.

Пользуясь циклической записью, запишем равенство (4) следующим образом:

$$(a\,c\,f)\,(b\,d)\,(a\,b\,d)\,(e\,f) = (a\,c\,e\,f\,b). \tag{5}$$

Обратите внимание на то, что знак умножения "$\times$" принято опускать; это не противоречит циклической записи, так как легко видеть, что перестановка $(a\,c\,f)(b\,d)$ на самом деле является произведением перестановок $(a\,c\,f)$ и $(b\,d)$.

Произведение перестановок можно выполнять непосредственно с помощью циклической записи. Например, вычислив произведение перестановок

$$(a\,c\,f\,g)\,(b\,c\,d)\,(a\,e\,d)\,(f\,a\,d\,e)(b\,g\,f\,a\,e), \tag{6}$$

находим (двигаясь слева направо), что "$a$ переходит в $c$, $c$ переходит в $d$, $d$ переходит в $a$, $a$ переходит в $d$ и $d$ остается без изменений". Таким образом, конечным результатом (6) является то, что $a$ переходит в $d$, и мы запишем "$(a\,d$" в качестве частичного ответа. Теперь выясним, что происходит с $d$: "$d$ переходит в $b$, которое затем переходит в $g$"; следовательно, имеем частичный результат "$(a\,d\,g$". Рассмотрев, что происходит с $g$, находим, что "$g$ переходит в $a$, затем в $e$, в $f$ и в $a$"; таким образом, первый цикл замыкается: "$(a\,d\,g)$". Теперь выберем новый элемент, который еще не встречался, например $c$. Находим, что $c$ переходит в $e$, и читатель может проверить, что окончательным ответом для (6) будет "$(a\,d\,g)(c\,e\,b)$".

А теперь попробуем выполнить эту процедуру с помощью компьютера. Следующий алгоритм формализует описанный выше метод таким образом, чтобы его можно было выполнить с помощью компьютера.

**Алгоритм А** (*Перемножение перестановок, представленных в виде циклов*). Этот алгоритм (рис. 20) берет произведение циклов, такое как (6), и вычисляет получающуюся в результате перестановку в виде произведения непересекающихся циклов. Для простоты здесь не приводится описание процедуры удаления единичных циклов; это было бы лишь незначительным обобщением алгоритма. По мере выпол-

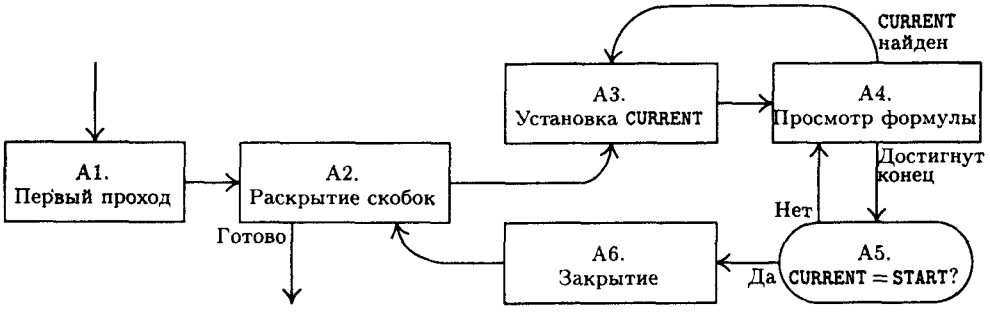



**Рис. 20.** Алгоритм A (перемножение перестановок).

нения алгоритма последовательно "помечаем" элементы исходной формулы, т. е. те символы, которые уже "обработаны".

**A1.** [Первый проход.] Отметить все левые скобки и заменить все правые скобки помеченной копией входного символа, который следует за соответствующей левой скобкой (см. пример в табл. 1).

**A2.** [Раскрытие скобок.] Выполнив поиск слева направо, найти первый непомеченный элемент входной формулы. (Если все элементы помечены, то работа алгоритма заканчивается.) Установить START равным этому элементу; вывести левую скобку; вывести элемент и пометить его.

**A3.** [Установка CURRENT.] Установить CURRENT равным следующему элементу формулы.

**A4.** [Просмотр формулы.] Продвигаться вправо, пока не будет либо достигнут конец формулы, либо найден элемент, равный CURRENT; в последнем случае пометить его и вернуться к шагу A3.

**A5.** [CURRENT = START?] Если CURRENT $\neq$ START, вывести CURRENT и вернуться к шагу A4, снова начав с левого края формулы (продолжая тем самым вывод искомого цикла).

**A6.** [Закрытие.] (Полный искомый цикл выведен.) Вывести правую скобку и вернуться к шагу A2. ∎

Например, рассмотрим формулу (6). В табл. 1 показаны последовательные этапы ее обработки. В первой строке таблицы приведена формула после замены правых скобок начальными элементами соответствующих циклов. В последующих строках показан процесс обработки формулы по мере пометки все большего числа элементов. Курсор указывает на текущий элемент, подлежащий обработке. В результате будет выведено следующее выражение: "$(a\,d\,g)(c\,e\,b)(f)$". Обратите внимание, что в выводе присутствует единичный цикл.

**Программа для MIX.** Чтобы реализовать этот алгоритм для MIX, "пометки" можно делать с помощью знака слова. Предположим, что входная формула перфорирована на картах в следующем формате: состоящая из 80 колонок карта разделена на 16 полей по 5 символов. Каждое поле представляет собой один из следующих вариантов: (a) "␣␣␣␣(", что обозначает левую скобку начала цикла; (b) ")␣␣␣␣",

**Таблица 1**

ПРИМЕНЕНИЕ АЛГОРИТМА А К ВЫРАЖЕНИЮ (6)

| После шага | START | CURRENT | ( a c f g ) | ( b c d ) | ( a e d ) | ( f a d e ) | ( b g f a e ) | Вывод |
|---|---|---|---|---|---|---|---|---|
| A1 | | | a c f g | b c d | a e d | f a d e | b g f a e | |
| A2 | a | | ⌶c f g | b c d | a e d | f a d e | b g f a e | ( a |
| A3 | a | c | c⌶f g | b c d | a e d | f a d e | b g f a e | |
| A4 | a | c | c f g | b ⌶d | a e d | f a d e | b g f a e | |
| A4 | a | d | c f g | b d | a e | f a d e | b g f a e | |
| A4 | a | a | c f g | b d | a e | f ⌶d e | b g f a e | |
| A5 | a | d | c f g | b d | a e | f d e | b g f a e ⌶ | d |
| A5 | a | g | c f g | b | a e | f d e | g f a e ⌶ | g |
| A5 | a | a | c f | b | e | f d | g a e ⌶ | |
| A6 | a | a | c f | b | e | f d | g a e ⌶ | ) |
| A2 | c | a | ⌶f | b | e | f d | g a e | ( c |
| A5 | c | e | f | b | e | d | g e ⌶ | e |
| A5 | c | b | f | b | | | g ⌶ | b |
| A6 | c | c | f | | | | g ⌶ | ) |
| A6 | f | f | | | | | ⌶ | ( f ) |

Здесь ⌶ обозначает положение курсора сразу за только что проверенным элементом помеченные элементы выделены светло серым цветом

что обозначает правую скобку конца цикла, (c) "␣␣␣␣␣", т е все пробелы, которые можно вставить в любом месте, чтобы заполнить пространство, (d) любой символ, обозначающий элемент, который нужно переставить Последняя карта входных данных определяется по тому, что в ее колонках 76–80 содержится "␣␣␣␣=" Например, (6) можно перфорировать на двух картах следующим образом

```
   ( A   C   F   G )     ( B   C   D )       ( A   E   D )
   ( F   A   D   E )     ( B   G   F   A   E )                 =
```

В выводе нашей программы будет содержаться точная копия ввода с последующим ответом в таком же, в сущности, формате

**Программа А** (*Перемножение перестановок, представленных в виде циклов*) В этой программе реализован алгоритм А, а также предусмотрены процедуры ввода, вывода и удаления единичных циклов

```
01 MAXWDS  EQU  1200          Максимальная длина ввода
02 PERM    ORIG *+MAXWDS      Вводная перестановка
03 ANS     ORIG *+MAXWDS      Место для ответа
04 OUTBUF  ORIG *+24          Место для печати
05 CARDS   EQU  16            Номер устройства чтения перфокарт
06 PRINTER EQU  18            Номер АЦПУ
07 BEGIN   IN   PERM(CARDS)   Читать первую карту
```

```
08          ENT2 0
09          LDA  EQUALS
10  1H      JBUS *(CARDS)          Ожидать окончания цикла
11          CMPA PERM+15,2
12          JE   *+2               Это последняя перфокарта?
13          IN   PERM+16,2(CARDS)  Нет, читать следующую.
14          ENT1 OUTBUF
15          JBUS *(PRINTER)        Напечатать копию введенной
16          MOVE PERM,2(16)          перфокарты.
17          OUT  OUTBUF(PRINTER)
18          JE   1F
19          INC2 16
20          CMP2 =MAXWDS-16=
21          JLE  1B                Повторять до окончания ввода.
22          HLT  666               Слишком много входной информации!
23  1H      INC2 15                1  В данный момент rI2 введенных слов
24          ST2  SIZE              1    находятся в ячейках PERM, PERM + 1, ....
25          ENT3 0                 1  A1. Первый проход.
26  2H      LDAN PERM,3            A  Взять следующий элемент ввода.
27          CMPA LPREN(1:5)        A  Это "("?
28          JNE  1F                A
29          STA  PERM,3            B  Если да, пометить ее.
30          INC3 1                 B  Поместить следующий непустой символ ввода
31          LDXN PERM,3            B    в rX
32          JXZ  *-2               B
33  1H      CMPA RPREN(1:5)        C
34          JNE  *+2               C
35          STX  PERM,3            D  Заменить ")" помеченным элементом из rX.
36          INC3 1                 C
37          CMP3 SIZE              C  Все элементы обработаны?
38          JL   2B                C
39          LDA  LPREN             1  Подготовиться к главной программе.
40          ENT1 ANS               1  rI1 — место сохранения следующего ответа.
41  OPEN    ENT3 0                 E  A2. Открытие.
42  1H      LDXN PERM,3            F  Искать непомеченный элемент
43          JXN  GO                F
44          INC3 1                 G
45          CMP3 SIZE              G
46          JL   1B                G
47  *                                 Все элементы помечены. Выполняется вывод
48  DONE    CMP1 =ANS=
49          JNE  *+2                  Ответ является тождественной перестановкой?
50          MOVE LPREN(2)             Если да, заменить на "()"
51          MOVE =0=                  Поместить 23 слова пробелов после ответа.
52          MOVE -1,1(22)
53          ENT3 0
54          OUT  ANS,3(PRINTER)
55          INC3 24
56          LDX  ANS,3                Напечатать столько строк, сколько необходимо.
57          JXNZ *-3
```

```
58          HLT
59  *
60  LPREN   ALF      (                Константы, используемые в программе.
61  RPREN   ALF   )
62  EQUALS  ALF      =
63  *
64  GO      MOVE LPREN          H    Открыть цикл в выводе.
65          MOVE PERM,3         H
66          STX  START          H
67  SUCC    STX  PERM,3         J    Пометить элемент.
68          INC3 1              J    Сделать один шаг вправо.
69          LDXN PERM,3(1:5)    J    A3. Установить CURRENT (а именно — rX).
70          JXN  1F             J    Пропустить пробелы.
71          JMP  *-3            0
72  5H      STX  0,1            Q    Вывести CURRENT.
73          INC1 1              Q
74          ENT3 0              Q    Снова просмотреть формулу.
75  4H      CMPX PERM,3(1:5)    K    A4. Просмотр формулы.
76          JE   SUCC           K    Элемент = CURRENT?
77  1H      INC3 1              L    Продвинуться вправо.
78          CMP3 SIZE           L    Конец формулы?
79          JL   4B             L
80          CMPX START(1:5)     P    A5. CURRENT = START?
81          JNE  5B             P
82  CLOSE   MOVE RPREN          R    A6. Закрытие.
83          CMPA -3,1           R    Заметить: rA = "(".
84          JNE  OPEN           R
85          INC1 -3             S    Удалить единичные циклы.
86          JMP  OPEN           S
87          END  BEGIN
```

Эта программа, содержащая примерно 75 команд, значительно длиннее программ из предыдущего раздела, да и большинства программ, с которыми мы встретимся в этой книге. Но ее длина не должна внушать вам страх, так как она делится на несколько маленьких частей, которые достаточно независимы одна от другой. В строках 07–22 происходит считывание входных перфокарт и печать копии каждой карты; в строках 23–38 выполняется шаг A1 алгоритма, т. е. предварительная обработка входных данных; в строках 39–46 и 64–86 выполняется основная часть алгоритма A, а в строках 48–57 выводится ответ. Читателю полезно изучить как можно больше программ для MIX, приведенных в данной книге, поскольку чрезвычайно важно приобрести опыт чтения программ, написанных другими. Но, к сожалению, подобной формой обучения пренебрегали на многих компьютерных курсах, что привело к крайне неэффективному использованию компьютерной техники.

**Время выполнения.** Те части программы A, которые не связаны с операциями ввода-вывода, снабжены счетчиками частоты, указывающими, сколько раз они выполняются (как в программе 1.3.2M); таким образом, предполагается, что строка 30 выполняется $B$ раз. Для удобства считается, что во входной информации пустые слова могут находиться только у правого края; при этом условии никогда не выполняется строка 71 и никогда не происходит переход в строке 32.

Выполняя простые операции сложения, получаем, что общее время выполнения программы равно

$$(7 + 5A + 6B + 7C + 2D + E + 3F + 4G + 8H + 6J + 3K + 4L + 3P + 4Q + 6R + 2S)u \quad (7)$$

плюс время, затрачиваемое на ввод и вывод. Чтобы понять смысл формулы (7), нужно рассмотреть пятнадцать неизвестных $A$, $B$, $C$, $D$, $E$, $F$, $G$, $H$, $J$, $K$, $L$, $P$, $Q$, $R$, $S$ и связать их с соответствующими характеристиками входных данных. Проиллюстрируем общие принципы решения проблем подобного рода.

Сначала применим первый закон Кирхгофа теории электрических цепей: команда должна выполняться столько раз, сколько раз осуществляется переход к ней. Это, казалось бы, очевидное правило часто приводит к неочевидным соотношениям между несколькими величинами. Анализируя ход выполнения программы A, получаем следующие уравнения.

| *Из строк* | *Выведено* |
|---|---|
| 26, 38 | $A = 1 + (C - 1)$ |
| 33, 28 | $C = B + (A - B)$ |
| 41, 84, 86 | $E = 1 + R$ |
| 42, 46 | $F = E + (G - 1)$ |
| 64, 43 | $H = F - G$ |
| 67, 70, 76 | $J = H + (K - (L - J))$ |
| 75, 79 | $K = Q + (L - P)$ |
| 82, 72 | $R = P - Q$ |

Не все уравнения, полученные с помощью закона Кирхгофа, будут линейно независимыми, например в данном случае мы видим, что первое и второе уравнения явно эквивалентны. Более того, последнее уравнение можно вывести из остальных, так как из третьего, четвертого и пятого следует, что $H = R$. Таким образом, шестое означает, что $K = L - R$. Во всяком случае мы уже исключили шесть из пятнадцати неизвестных:

$$A = C, \quad E = R + 1, \quad F = R + G, \quad H = R, \quad K = L - R, \quad Q = P - R. \quad (8)$$

Первый закон Кирхгофа — это очень эффективное средство; более подробно мы рассмотрим его в разделе 2.3.4.1.

Следующий шаг состоит в том, чтобы попытаться сопоставить переменные с важными характеристиками данных. Из строк 24, 25, 30 и 36 находим, что

$$B + C = \text{количество слов ввода} = 16X - 1, \quad (9)$$

где $X$ — число перфокарт с входной информацией. Из строки 28 получаем, что

$$B = \text{количество “(” во вводе} = \text{количество циклов во вводе.} \quad (10)$$

Аналогично из строки 34 получаем

$$D = \text{количество “)” во вводе} = \text{количество циклов во вводе.} \quad (11)$$

А теперь из (10) и (11) получаем то, что нельзя вывести из закона Кирхгофа:

$$B = D. \tag{12}$$

Из строки 64 получаем

$$H = \text{количество циклов в выводе (включая единичные циклы).} \tag{13}$$

Из строки 82 следует, что $R$ равно этой же величине; в данном случае соотношение $H = R$ *можно было* вывести из закона Кирхгофа, так как оно уже содержится в (8).

Учитывая тот факт, что каждое непустое слово в конце концов помечается, из строк 29, 35 и 67 находим, что

$$J = Y - 2B, \tag{14}$$

где $Y$ — число непустых слов, содержащихся в перестановках ввода. Учитывая то, что каждый *отличный от других* элемент, содержащийся во входной перестановке, записывается в вывод только один раз, либо в строке 65, либо в строке 72, получаем

$$P = H + Q = \text{количество различных элементов во вводе.} \tag{15}$$

(См. соотношения (8).) Если немного поразмыслить, это очевидным образом следует также из строки 80. И, наконец, из строки 85 видно, что

$$S = \text{количество единичных циклов в выводе.} \tag{16}$$

Очевидно. что величины $B$, $C$, $H$, $J$, $P$ и $S$, которые мы сейчас интерпретировали как независимые параметры, влияют на время выполнения программы A.

Теперь остается проанализировать только неизвестные $G$ и $L$. Для этого придется проявить немного больше изобретательности. Просмотры входной информации, которые начинаются в строках 41 и 74, всегда заканчиваются либо в строке 47 (прошлый раз), либо в строке 80. В течение каждого из этих $P + 1$ циклов команда "`INC3 1`" выполняется $B + C$ раз. Это имеет место только в строках 44, 68 и 77, поэтому получаем нетривиальное соотношение

$$G + J + L = (B + C)(P + 1), \tag{17}$$

связывающее неизвестные $G$ и $L$. К счастью, время выполнения (7) — это функция от $G + L$ (так как оно включает слагаемые $\cdots + 3F + 4G \cdots + 3K + 4L + \cdots = \cdots + 7G + \cdots + 7L + \cdots$), поэтому нам не нужно больше пытаться анализировать величины $G$ и $L$ по отдельности.

Просуммировав все эти результаты, находим, что общее время выполнения программы без учета времени, затрачиваемого на операции ввода-вывода, равно

$$(112NX + 304X - 2M - Y + 11U + 2V - 11)u. \tag{18}$$

В этой формуле были использованы новые обозначения для характеристик данных:

$$\begin{aligned}
&X \text{ — количество перфокарт ввода;}\\
&Y \text{ — количество непустых полей во вводе (исключая конечное "=");}\\
&M \text{ — количество циклов во вводе;}\\
&N \text{ — количество имен различных элементов во вводе;}\\
&U \text{ — количество циклов в выводе (включая единичные циклы);}\\
&V \text{ — количество единичных циклов в выводе.}
\end{aligned} \tag{19}$$

**Таблица 2**

ПЕРЕМНОЖЕНИЕ ПЕРЕСТАНОВОК ЗА ОДИН ПРОХОД

| | $(a\ f\ g)\ (b\ c\ d)\ (a\ e\ d)\ (f\ a\ d\ e)\ (b\ g\ f\ a\ e)$ |
|---|---|
| $a \to$ | $d\,d\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,a\,d\,d\,d\,d\,d\,d\,e\,e\,e\,e\,e\,e\,e\,e\,e\,a\,a$ |
| $b \to$ | $c\,c\,c\,c\,c\,c\,c\,c\,c\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,g\,b\,b\,b\,b\,b$ |
| $c \to$ | $e\,e\,e\,d\,d\,d\,d\,d\,d\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c\,c$ |
| $d \to$ | $g\,g\,g\,g\,g\,g\,g\,)\,)\,)\,d\,d\,)\,)\,)\,b\,b\,b\,b\,b\,d\,d\,d\,d\,d\,d\,d\,d\,d$ |
| $e \to$ | $b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,b\,a\,a\,a\,)\,)\,)\,)\,b\,b\,)\,)\,)\,)\,)\,e$ |
| $f \to$ | $f\,f\,f\,f\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,e\,a\,a\,a\,a\,a\,a\,a\,a\,f\,f\,f$ |
| $g \to$ | $a\,)\,)\,)\,)\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,g\,g\,g\,g$ |

Следовательно, можно сделать вывод, что анализ такой программы, как A, во многих отношениях подобен решению занимательной головоломки.

Ниже мы покажем,что если на выходе предполагается получить случайную перестановку, то средние величин $U$ и $V$ будут равны $H_N$ и 1 соответственно.

**Другой подход.** Алгоритм A перемножает перестановки почти так же, как это обычно делают люди. Часто оказывается, что задачи, решаемые с помощью компьютера, очень похожи на те, с которыми люди постоянно сталкивались в течение долгих лет. Поэтому освященные веками методы решения, разработанные для простых смертных, таких, как мы с вами, подходят и для компьютерных алгоритмов.

Но не менее часто приходится иметь дело с новыми методами, которые идеально подходят для компьютера, но совершенно непригодны для человека. Главная причина этого состоит в том, что компьютер "думает" по-другому; тип его памяти, с помощью которой он запоминает информацию, отличается от типа памяти человека. Это различие можно проследить на примере нашей задачи перемножения перестановок. Используя приведенный ниже алгоритм, компьютер может выполнить умножение перестановок "одним махом", запоминая текущее состояние перестановки в целом по мере перемножения циклов. Предназначенный для человека алгоритм A просматривает формулу много раз, по одному для каждого элемента вывода, в то время как новый алгоритм делает все за один проход. *Homo sapiens* на такое вряд ли способен.

Что представляет собой предназначенный для компьютера метод перемножения перестановок? Главная идея проиллюстрирована в табл. 2. В столбце, расположенном в этой таблице под каждым символом выражения, которое представлено в виде циклов, показано, какая перестановка выражена частичными циклами *справа*. Например, фрагмент формулы "... $d\,e)(b\ g\ f\ a\ e)$" представляет перестановку

$$\begin{pmatrix} a & b & c & d & e & f & g \\ e & g & c & b & ? & a & f \end{pmatrix},$$

которая появляется в таблице под крайним справа символом $d$.

Внимательно изучив табл. 2, можно понять принцип ее построения, если начать с тождественной перестановки справа и двигаться назад справа налево Столбец, расположенный под символом $x$, отличается от столбца справа (в котором записано

предыдущее состояние) только строкой $x$; и новое значение в строке $x$ — это значение, которое исчезло в результате предыдущего изменения. Короче говоря, имеем следующий алгоритм.

**Алгоритм B** (*Перемножение перестановок, представленных в виде циклов*). Этот алгоритм (рис. 21) дает, в сущности, тот же результат, что и алгоритм A. Предположим, что переставляемыми элементами являются $x_1, x_2, \ldots, x_n$. Будем использовать вспомогательную таблицу $T[1], T[2], \ldots, T[n]$; по окончании работы алгоритма $x_i$ переходит в $x_j$ в перестановке ввода тогда и только тогда, когда $T[i] = j$.

**B1.** [Инициализация.] Присвоить $T[k] \leftarrow k$, где $1 \le k \le n$. Подготовиться также к просмотру справа налево.

**B2.** [Следующий элемент.] Рассмотреть следующий элемент ввода (справа налево). Если ввод исчерпан, то работа алгоритма заканчивается. Если рассматривается элемент ")", то присвоить $Z \leftarrow 0$ и повторить шаг B2; если это элемент "(", то перейти к шагу B4. В противном случае рассматривается элемент $x_i$ для некоторого $i$; тогда перейти к шагу B3.

**B3.** [Замена $T[i]$.] Выполнить взаимный обмен $Z \leftrightarrow T[i]$. Если в результате получится $T[i] = 0$, то присвоить $j \leftarrow i$. Вернуться к шагу B2.

**B4.** [Замена $T[j]$.] Присвоить $T[j] \leftarrow Z$. (Здесь $j$ — это строка, определяющая элемент ")" в обозначениях табл. 2, т. е. правую скобку, которая соответствует только что просмотренной левой скобке.) Вернуться к шагу B2. ▮

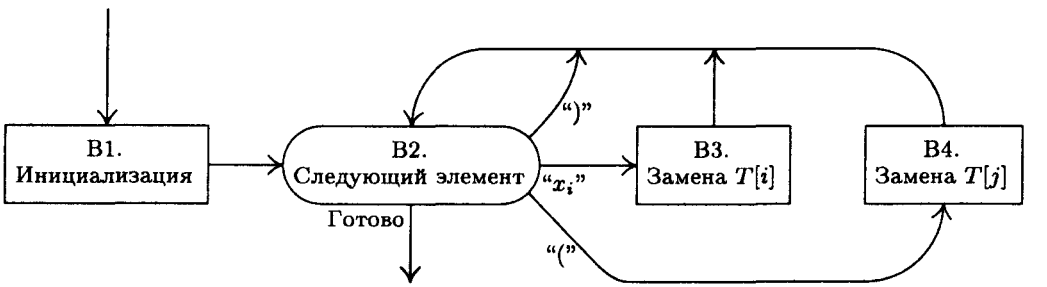



**Рис. 21.** Алгоритм B: перемножение перестановок.

Разумеется, после выполнения этого алгоритма необходимо еще вывести содержимое таблицы $T$ в циклическом виде; как будет показано ниже, это легко сделать методом "пометок".

Теперь на основе нового алгоритма давайте напишем программу для MIX. Будем придерживаться тех же основных правил, что и в программе A, используя такой же формат ввода и вывода, как и раньше. Но тут возникает небольшая проблема: как можно реализовать алгоритм B, не зная наперед, чему равны элементы $x_1, x_2, \ldots, x_n$? Неизвестно, чему равно $n$, а также будет ли элемент $b$ равен $x_1$ или $x_2$ и т. д. Существует простой способ решения данной проблемы — создать таблицу, внести в нее имена элементов, которые уже встречались, и каждый раз искать имя текущего элемента (см. строки 35–44 в программе ниже).

**Программа B** (*Дает тот же результат, что и программа A*). rX $\equiv Z$; rI4 $\equiv i$; rI1 $\equiv j$; rI3 $= n$, количество различных просмотренных имен.

```
01  MAXWDS EQU  1200          Максимальная длина ввода.
02  X      ORIG *+MAXWDS      Таблица имен.
03  T      ORIG *+MAXWDS      Вспомогательная таблица состояния.
04  PERM   ORIG *+MAXWDS      Входная перестановка.
05  ANS    EQU  PERM          Место для ответа.
06  OUTBUF ORIG *+24          Место для печати.
07  CARDS  EQU  16      }
    ...                 }     Совпадают со строками 05–22 программы A.
24         HLT  666     }     В этот момент rI2 слов
25  1H     INC2 15        1      ввода находятся в PERM, PERM + 1, ...
26         ENT3 1         1      и мы пока еще не видели ни одного имени.
27  RIGHT  ENTX 0         A   Присвоить Z ← 0.
28  SCAN   DEC2 1         B   B2. Следующий элемент.
29         LDA  PERM,2    B
30         JAZ  CYCLE     B   Пропустить пробелы.
31         CMPA RPREN     C
32         JE   RIGHT     C   Следующим элементом является ")"?
33         CMPA LPREN     D
34         JE   LEFT      D   Это "("?
35         ENT4 1,3       E   Приготовиться к поиску.
36         STA  X         E   Сохранить в начале таблицы.
37  2H     DEC4 1         F   Искать в таблице имен.
38         CMPA X,4       F
39         JNE  2B        F   Повторять до нахождения соответствия.
40         J4P  FOUND     G   Это имя появлялось ранее?
41         INC3 1         H   Нет; увеличить размер таблицы.
42         STA  X,3       H   Вставить новое имя x_n.
43         ST3  T,3       H   Присвоить T[n] ← n,
44         ENT4 0,3       H     i ← n.
45  FOUND  LDA  T,4       J   B3. Замена T[i].
46         STX  T,4       J   Сохранить Z.
47         SRC  5         J   Установить Z.
48         JANZ SCAN      J
49         ENT1 0,4       K   Если Z было нулем, присвоить j ← i.
50         JMP  SCAN      K
51  LEFT   STX  T,1       L   B4. Замена T[j].
52  CYCLE  J2P  SCAN      P   Вернуться к B2, если работа еще не закончена.
53  *
54  OUTPUT ENT1 ANS       1   Весь ввод просмотрен.
55         J3Z  DONE      1   Таблицы x и T содержат ответ.
56  1H     LDAN X,3       Q   Теперь построим циклическую запись.
57         JAP  SKIP      Q   Имя было помечено?
58         CMP3 T,3       R   Это единичный цикл?
59         JE   SKIP      R
60         MOVE LPREN     S   Открыть цикл.
61  2H     MOVE X,3       T
62         STA  X,3       T   Пометить имя.
```

```
63          LD3  T,3        T  Найти элемент, в который переходит этот элемент.
64          LDAN X,3        T
65          JAN  2B         T  Он уже помечен?
66          MOVE RPREN      W  Да, цикл закрывается.
67    SKIP  DEC3 1          Z  Перейти к следующему имени.
68          J3P  1B         Z
69    *
70    DONE  CMP1 =ANS=   }
      ...                } Совпадают со строками 48–62 программы A.
84    EQUALS ALF     =   }
85          END  BEGIN      ▮
```

Строки 54–68, в которых строится циклическая запись на основании таблицы $T$ и таблицы имен, представляют собой небольшой алгоритм, который заслуживает внимания. Величины $A$, $B$, ..., $R$, $S$, $T$, $W$, $Z$, влияющие на время выполнения программы, конечно, отличаются от одноименных величин, которые использовались при анализе программы A. Анализ этих величин будет интересным упражнением для читателя (см. упр. 10).

Опыт показывает, что основная часть времени выполнения программы B будет потрачена на поиск в таблице имен; это время определяется параметром $F$. Существуют более эффективные алгоритмы поиска и построения словарей имен; они называются *алгоритмами таблиц символов* и играют важную роль в прикладной математике. В главе 6 мы займемся всесторонним исследованием эффективных алгоритмов таблиц символов.

**Обратные перестановки.** Обратная перестановка $\pi^-$ для перестановки $\pi$ — это такая перестановка, которая отменяет действие $\pi$; если в $\pi$ элемент $i$ переходит в $j$ $\pi$, то в $\pi^-$ элемент $j$ переходит в $i$. Таким образом, произведение $\pi\pi^-$ равняется тождественной перестановке; то же самое справедливо и для произведения $\pi^-\pi$. Обратную перестановку часто обозначают через $\pi^{-1}$, а не $\pi^-$, но верхний индекс 1 явно лишний (по той же причине, по которой $x^1 = x$).

Каждая перестановка имеет обратную. Например, обратной перестановкой для

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \quad \text{является} \quad \begin{pmatrix} c & d & f & b & e & a \\ a & b & c & d & e & f \end{pmatrix} = \begin{pmatrix} a & b & c & d & e & f \\ f & d & a & b & e & c \end{pmatrix}.$$

Рассмотрим некоторые простые алгоритмы вычисления обратной перестановки.

Предположим, что в оставшейся части раздела мы будем иметь дело с перестановками чисел $\{1, 2, \ldots, n\}$. Если $X[1]X[2]\ldots X[n]$ — такая перестановка, то существует простой метод вычисления перестановки, обратной к ней. Присвоим $Y[X[k]] \leftarrow k$ для $1 \le k \le n$. Тогда $Y[1]Y[2]\ldots Y[n]$ — искомая обратная перестановка. В этом методе используется $2n$ ячеек памяти, а именно — $n$ для $X$ и $n$ для $Y$.

А теперь ради интереса предположим, что $n$ очень велико, в то время как нужно вычислить обратную перестановку к $X[1]X[2]\ldots X[n]$, не используя слишком много дополнительного пространства памяти. Необходимо вычислить обратную перестановку "на месте", чтобы после окончания работы алгоритма массив $X[1]X[2]\ldots X[n]$ представлял собой перестановку, обратную к первоначальной. Простое присвоение $X[X[k]] \leftarrow k$ для $1 \le k \le n$ в этом случае, разумеется, не пройдет, но, рассматривая циклическую структуру, можно получить следующий простой алгоритм.

**Таблица 3**

ОБРАЩЕНИЕ ПЕРЕСТАНОВКИ 621543 С ПОМОЩЬЮ АЛГОРИТМА I

| После шага: | I2 | I3 | I3 | I3 | I5* | I2 | I3 | I3 | I5 | I2 | I5 | I5 | I3 | I5 | I5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X[1]$ | 6 | 6 | 6 | −3 | −3 | −3 | −3 | −3 | −3 | −3 | −3 | −3 | −3 | −3 | 3 |
| $X[2]$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | −4 | 2 | 2 |
| $X[3]$ | 1 | 1 | −6 | −6 | −6 | −6 | −6 | −6 | −6 | −6 | −6 | 6 | 6 | 6 | 6 |
| $X[4]$ | 5 | 5 | 5 | 5 | 5 | 5 | 5 | −5 | −5 | −5 | 5 | 5 | 5 | 5 | 5 |
| $X[5]$ | 4 | 4 | 4 | 4 | 4 | 4 | −1 | −1 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $X[6]$ | 3 | −1 | −1 | −1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $m$ | 6 | 3 | 1 | 6 | 6 | 5 | 4 | 5 | 5 | 4 | 4 | 3 | 2 | 2 | 1 |
| $j$ | −1 | −6 | −3 | −1 | −1 | −1 | −5 | −4 | −4 | −4 | −4 | −4 | −2 | −2 | −2 |
| $i$ | 3 | 1 | 6 | −1 | −1 | 4 | 5 | −1 | −4 | −5 | −5 | −6 | −4 | −2 | −3 |

Читать столбцы слева направо. В момент * цикл (1 6 3) уже был обращен.

**Алгоритм I** (*Обратная перестановка на месте*). Заменить $X[1]X[2]\ldots X[n]$, которая является перестановкой чисел $\{1,2,\ldots,n\}$, обратной перестановкой. Этот алгоритм был предложен Бин-Чао Хуаном (Bing-Chao Huang) [*Inf. Proc. Letters* **12** (1981), 237–238].

**I1.** [Инициализация.] Присвоить $m \leftarrow n$, $j \leftarrow -1$.

**I2.** [Следующий элемент.] Присвоить $i \leftarrow X[m]$. Если $i < 0$, перейти к шагу I5 (этот элемент уже был обработан).

**I3.** [Обратить один элемент.] (В этот момент $j < 0$ и $i = X[m]$. Если $m$ не является наибольшим элементом своего цикла, то первоначальная перестановка давала $X[-j] = m$.) Присвоить $X[m] \leftarrow j$, $j \leftarrow -m$, $m \leftarrow i$, $i \leftarrow X[m]$.

**I4.** [Конец цикла?] Если $i > 0$, перейти к шагу I3 (этот цикл не закончен); иначе — присвоить $i \leftarrow j$. (В последнем случае первоначальная перестановка давала $X[-j] = m$, где $m$ — наибольший элемент в его цикле.)

**I5.** [Сохранить окончательное значение.] Присвоить $X[m] \leftarrow -i$. (Первоначально $X[-i]$ было равно $m$.)

**I6.** [Цикл по $m$.] Уменьшить $m$ на 1. Если $m > 0$, перейти к I2; в противном случае работа алгоритма заканчивается. ∎

Пример данного алгоритма приведен в табл. 3. Этот метод основан на обращении последовательных циклов перестановки; для пометки обращенных элементов их делают отрицательными, а впоследствии восстанавливают первоначальный знак.

Алгоритм I частично напоминает алгоритм A и очень сильно напоминает алгоритм нахождения циклов, реализованный в программе B (строки 54–68). Таким образом, это типичный представитель алгоритмов, имеющих отношение к перестановкам. Во время подготовки его реализации для MIX было обнаружено, что удобнее всего сохранять в регистре величину $-i$, а не саму $i$.

**Программа I** (*Обращение на месте*). rI1 $\equiv m$; rI2 $\equiv -i$; rI3 $\equiv j$ и $n =$ N (этот символ определяется, когда программа транслируется как часть большей программы).

```
01  INVERT ENT1 N      1  I1. Инициализация. m ← n.
02          ENT3 -1    1  j ← −1.
```

```
03  2H      LD2N X,1    N   I2. Следующий элемент. i ← X[m].
04          J2P  5F     N   Переход к шагу I5, если i < 0.
05  3H      ST3  X,1    N   I3. Обратить элемент. X[m] ← j.
06          ENN3 0,1    N   j ← −m.
07          ENN1 0,2    N   m ← i.
08          LD2N X,1    N   i ← X[m].
09  4H      J2N  3B     N   I4. Конец цикла? Переход к шагу I3, если i > 0.
10          ENN2 0,3    C   В противном случае присвоить i ← j.
11  5H      ST2  X,1    N   I5. Сохранить окончательное значение. X[m] ← −i.
12  6H      DEC1 1      N   I6. Цикл по m.
13          J1P  2B     N   Переход к шагу I2, если m > 0. ▮
```

Время выполнения этой программы легко подсчитать способом, о котором говорилось выше. Каждому элементу $X[m]$ сначала присваивается отрицательное значение на шаге I3, а затем — положительное на шаге I5. Общее время выполнения составляет $(14N+C+2)u$, где $N$ — размерность массива, а $C$ — общее число циклов. Ниже будет проанализировано поведение $C$ в случайной перестановке.

Почти всегда существует несколько алгоритмов решения любой поставленной задачи, поэтому можно ожидать, что есть еще один способ обращения перестановки. Следующий остроумный алгоритм был предложен Дж. Бутройдом (J. Boothroyd).

**Алгоритм J** (*Обращение на месте*). Этот алгоритм дает такой же результат, как и алгоритм I, но на основании другого метода.

**J1.** [Сделать все величины отрицательными.] Присвоить $X[k] \leftarrow -X[k]$ для $1 \le k \le n$. Присвоить также $m \leftarrow n$.

**J2.** [Инициализация $j$.] Присвоить $j \leftarrow m$.

**J3.** [Нахождение отрицательного элемента.] Присвоить $i \leftarrow X[j]$. Если $i > 0$, присвоить $j \leftarrow i$ и повторить этот шаг.

**J4.** [Обращение.] Присвоить $X[j] \leftarrow X[-i]$, $X[-i] \leftarrow m$.

**J5.** [Цикл по $m$.] Уменьшить $m$ на 1; если $m > 0$, вернуться к шагу J2. В противном случае работа алгоритма заканчивается. ▮

Пример выполнения алгоритма Бутройда приведен в табл. 4. В сущности, в основе метода опять лежит циклическая структура, но в данном случае то, что алгоритм действительно решает поставленную задачу, гораздо менее очевидно. Мы предоставляем читателю проверить это самостоятельно (см. упр. 13).

**Программа J** (*Аналог программы I*). rI1 ≡ $m$; rI2 ≡ $j$; rI3 ≡ $-i$.

```
01  INVERT ENN1 N               1   J1. Сделать все величины отрицательными.
02         ST1  X+N+1,1(0:0)    N   Установить отрицательный знак.
03         INC1 1               N
04         J1N  *-2             N   Еще?
05         ENT1 N               1   m ← n.
06  2H     ENN3 0,1             N   J2. Инициализация j. i ← m.
07         ENN2 0,3             A   j ← i.
08         LD3N X,2             A   J3. Нахождение отрицательного элемента.
```

**Таблица 4**

ОБРАЩЕНИЕ ПЕРЕСТАНОВКИ 621543 С ПОМОЩЬЮ АЛГОРИТМА J

| После шага: | J2 | J3 | J5 | J3 | J5 | J3 | J5 | J3 | J5 | J3 | J5 | J3 | J5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X[1]$ | −6 | −6 | −6 | −6 | −6 | −6 | −6 | −6 | 3 | 3 | 3 | 3 | 3 |
| $X[2]$ | −2 | −2 | −2 | −2 | −2 | −2 | −2 | −2 | −2 | −2 | 2 | 2 | 2 |
| $X[3]$ | −1 | −1 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $X[4]$ | −5 | −5 | −5 | −5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| $X[5]$ | −4 | −4 | −4 | −4 | −5 | −5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| $X[6]$ | −3 | −3 | −1 | −1 | −1 | −1 | −1 | −1 | −6 | −6 | −6 | −6 | 1 |
| $m$ | 6 | 6 | 5 | 5 | 4 | 4 | 3 | 3 | 2 | 2 | 1 | 1 | 0 |
| $i$ | | −3 | −3 | −4 | −4 | −5 | −5 | −1 | −1 | −2 | −2 | −6 | −6 |
| $j$ | 6 | 6 | 6 | 5 | 5 | 5 | 5 | 6 | 6 | 2 | 2 | 6 | 6 |

```
09      J3N   *-2        A   i > 0?
10      LDA   X,3        N   J4. Обращение.
11      STA   X,2        N   X[j] ← X[−i].
12      ST1   X,3        N   X[−i] ← m.
13      DEC1  1          N   J5. Цикл по m.
14      J1P   2B         N   Переход к шагу J2, если m > 0   ▮
```

Чтобы выяснить, насколько быстро работает данная программа, нужно знать величину $A$; она настолько интересна и поучительна, что мы оставили ее для упражнения (см. упр. 14).

Хотя алгоритм J невероятно изящен, анализ показывает, что алгоритм I намного его превосходит. На самом деле оказывается, что среднее время выполнения алгоритма J, в сущности, пропорционально $n \ln n$, а среднее время выполнения алгоритма I пропорционально $n$. Возможно, кто-нибудь когда-нибудь найдет применение алгоритму J (или некоторой его модификации); это слишком изящный алгоритм, чтобы его можно было совсем забыть.

**Замечательное соответствие.** Как уже отмечалось, запись перестановки в циклическом виде не единственна; состоящую из шести элементов перестановку (1 6 3)(4 5) можно записать как (5 4)(3 1 6) и т. д. Поэтому полезно рассмотреть *каноническую форму* циклического представления, которая *является* единственной. Для получения канонической формы выполните следующие действия.

a) Выпишите явно все единичные циклы.
b) Внутри каждого цикла поместите на первое место наименьшее число.
c) Расположите циклы в порядке *убывания* их первых элементов.

Например, для перестановки (3 1 6)(5 4) получим

$$\text{(a): (3 1 6) (5 4) (2);} \quad \text{(b): (1 6 3) (4 5) (2);} \quad \text{(c): (4 5) (2) (1 6 3).} \tag{20}$$

Важным свойством этой канонической формы является то, что скобки можно удалить, а затем восстановить единственным образом. Следовательно, расставить скобки в "4 5 2 1 6 3" для получения канонической циклической формы можно только единственным способом. Необходимо вставить левую скобку непосредственно перед каждым *минимумом слева направо* (т. е. перед тем элементом, перед которым нет меньшего элемента).

Это свойство канонической формы позволяет получить замечательное однозначное соответствие между множеством всех перестановок, представленных в циклическом виде, и множеством всех перестановок, представленных в линейной форме. Например, канонической формой для перестановки 6 2 1 5 4 3 является (4 5) (2) (1 6 3); удалив скобки, получим перестановку 4 5 2 1 6 3, циклической формой которой является (2 5 6 3) (1 4); удалив скобки, получим 2 5 6 3 1 4, циклической формой которой является (3 6 4) (1 2 5) и т. д.

Это соответствие имеет многочисленные применения в изучении перестановок различных типов. Например, зададимся вопросом: "Сколько циклов имеет перестановка из $n$ элементов в среднем?". Чтобы ответить на него, рассмотрим множество всех $n!$ перестановок, представленных в канонической форме, и уберем скобки. Останется множество всех $n!$ перестановок, расположенных в некотором порядке. Поэтому первоначальный вопрос будет эквивалентен следующему: "Сколько в среднем минимумов слева направо имеет перестановка из $n$ элементов?". На последний вопрос мы уже ответили в разделе 1.2.10; это была величина $(A + 1)$ из анализа алгоритма 1.2.10M, для которой найдены статистические характеристики

$$\min 1, \quad \text{ave } H_n, \quad \max n, \quad \text{dev } \sqrt{H_n - H_n^{(2)}}. \tag{21}$$

(На самом деле мы искали среднее число максимумов справа налево, но очевидно, что это то же самое, что число минимумов слева направо.) Более того, мы, в сущности, доказали, что перестановка из $n$ объектов имеет $k$ минимумов слева направо с вероятностью $\left[{n \atop k}\right]/n!$; следовательно, *перестановка из $n$ объектов имеет $k$ циклов с вероятностью* $\left[{n \atop k}\right]/n!$.

Можно также задать вопрос о среднем расстоянии *между* минимумами слева направо, которое эквивалентно средней длине цикла. Согласно (21) *общее* число циклов во всех $n!$ перестановках равно $n!\,H_n$, так как это $n!$, умноженное на *среднее* число циклов. Если выбрать один из этих циклов наугад, то чему будет равна его средняя длина?

Представьте себе все $n!$ перестановок $\{1, 2, \ldots, n\}$, записанных в циклической форме. Сколько среди них будет трехэлементных циклов? Чтобы ответить на этот вопрос, выясним, сколько раз появляется конкретный трехэлементный цикл $(x\,y\,z)$. Очевидно, что он появляется ровно в $(n - 3)!$ перестановках, так как это число способов, которыми можно переставить оставшиеся $n - 3$ элемента. Количество различных возможных трехэлементных циклов $(x\,y\,z)$ равно $n(n-1)(n-2)/3$, поскольку $x$ можно выбрать $n$ способами, $y$ — $(n - 1)$ способами, а $z$ — $(n - 2)$ способами, и среди этих $n(n - 1)(n - 2)$ вариантов каждый отличный от других трехэлементный цикл появляется в трех формах: $(x\,y\,z)$, $(y\,z\,x)$, $(z\,x\,y)$. Поэтому общее число трехэлементных циклов среди всех $n!$ перестановок равно $n(n-1)\times(n-2)/3$, умноженному на $(n-3)!$, т. е. $n!/3$. Аналогично общее число $m$-элементных циклов равно $n!/m$, где $1 \le m \le n$. (Это дает еще одно простое доказательство того факта, что общее число циклов равно $n!\,H_n$. Следовательно, как мы уже знаем, среднее число циклов в случайной перестановке равно $H_n$.) В упр. 17 показано, что средняя длина случайно выбранного цикла равна $n/H_n$, если считать, что все $n!\,H_n$ *циклов* являются равновероятными. Но если *элемент* в случайной перестановке выбран случайно, то средняя длина содержащего его цикла будет несколько больше, чем $n/H_n$.

Чтобы закончить анализ алгоритмов A и B, следует узнать среднее число *единичных циклов* в случайной перестановке. Это очень интересный вопрос. Предположим, мы записали $n!$ перестановок, перечислив сначала те, в которых нет единичных циклов, затем те, в которых есть только один единичный цикл и т. д. Например, для $n = 4$ это будет выглядеть так.

| | |
|---|---|
| Нет фиксированных элементов: | 2143 2341 2413 3142 3412 3421 4123 4312 4321 |
| Один фиксированный элемент: | $\overline{1}342$ $\overline{1}423$ $3\overline{2}41$ $4\overline{2}13$ $24\overline{3}1$ $41\overline{3}2$ $231\overline{4}$ $312\overline{4}$ |
| Два фиксированных элемента: | $\overline{12}43$ $\overline{14}\overline{3}2$ $\overline{1}32\overline{4}$ $4\overline{23}1$ $3\overline{2}1\overline{4}$ $21\overline{34}$ |
| Три фиксированных элемента: | |
| Четыре фиксированных элемента: | $\overline{1234}$ |

(В этом списке отмечены единичные циклы, т. е. элементы, которые в результате перестановки остаются на месте (фиксированные элементы).) Перестановки, не имеющие фиксированных элементов, называются *беспорядочными*; число таких перестановок — это количество способов так разложить $n$ писем по $n$ конвертам, чтобы ни одно из них не попало в свой конверт.

Пусть $P_{nk}$ — число перестановок из $n$ объектов, имеющих ровно $k$ фиксированных элементов. Тогда, например,

$$P_{40} = 9, \quad P_{41} = 8, \quad P_{42} = 6, \quad P_{43} = 0, \quad P_{44} = 1.$$

При изучении приведенного выше списка обнаруживаются главные соотношения, связывающие эти числа. Можно получить все перестановки с $k$ фиксированными элементами, сначала выбирая те $k$ элементов, которые нужно зафиксировать (это можно сделать $\binom{n}{k}$ способами), а затем переставляя оставшиеся $n - k$ элементов всеми $P_{(n-k)0}$ способами, которые больше не оставляют фиксированных элементов. Отсюда

$$P_{nk} = \binom{n}{k} P_{(n-k)0}. \tag{22}$$

Существует также правило, которое говорит о том, что "целое — это сумма составляющих его частей":

$$n! = P_{nn} + P_{n(n-1)} + P_{n(n-2)} + P_{n(n-3)} + \cdots. \tag{23}$$

Подставив (22) в (23) и выполнив простые преобразования, получаем

$$n! = \frac{P_{00}}{0!} + n\frac{P_{10}}{1!} + n(n-1)\frac{P_{20}}{2!} + n(n-1)(n-2)\frac{P_{30}}{3!} + \cdots. \tag{24}$$

Эта формула должна быть справедлива для всех целых положительных $n$. Она уже встречалась ранее, в разделе 1.2.5 (в связи с попытками Стирлинга обобщить факториальную функцию), а простой вывод ее коэффициентов был приведен в разделе 1.2.6 (пример 5). Мы приходим к выводу, что

$$\frac{P_{m0}}{m!} = 1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^m \frac{1}{m!}. \tag{25}$$

Теперь пусть $p_{nk}$ — вероятность того, что перестановка из $n$ объектов имеет ровно $k$ единичных циклов. Так как $p_{nk} = P_{nk}/n!$, из (22) и (25) получаем

$$p_{nk} = \frac{1}{k!}\left(1 - \frac{1}{1!} + \frac{1}{2!} - \cdots + (-1)^{n-k}\frac{1}{(n-k)!}\right). \tag{26}$$

Следовательно, производящую функцию $G_n(z) = p_{n0} + p_{n1}z + p_{n2}z^2 + \cdots$ можно представить в виде

$$G_n(z) = 1 + \frac{1}{1!}(z-1) + \cdots + \frac{1}{n!}(z-1)^n = \sum_{0\le j\le n}\frac{1}{j!}(z-1)^j. \tag{27}$$

Из этой формулы следует, что $G_n'(z) = G_{n-1}(z)$. Воспользовавшись методами, которые применялись в разделе 1.2.10, получим следующие статистические характеристики для числа единичных циклов:

$$(\min\ 0, \quad \text{ave}\ 1, \quad \max\ n, \quad \text{dev}\ 1), \qquad \text{если} \quad n \ge 2. \tag{28}$$

Несколько более прямой способ подсчета числа перестановок, не имеющих единичных циклов, следует из *принципа включения и исключения*, который имеет большое значение для многих задач перечисления. Общий принцип включения и исключения можно сформулировать следующим образом. Дано $N$ элементов и $M$ подмножеств этих элементов $S_1, S_2, \ldots, S_M$. Необходимо подсчитать, сколько элементов не попало ни в одно из этих подмножеств. Пусть $|S|$ — число элементов множества $S$. Тогда искомое число объектов, не принадлежащих ни одному из множеств $S_j$, равно

$$N - \sum_{1\le j\le M}|S_j| + \sum_{1\le j<k\le M}|S_j\cap S_k| - \sum_{1\le i<j<k\le M}|S_i\cap S_j\cap S_k| + \cdots + (-1)^M|S_1\cap\cdots\cap S_M|. \tag{29}$$

(Таким образом, сначала из общего числа $N$ вычитается количество элементов множеств $S_1, \ldots, S_M$; но это меньше искомой величины, так как мы вычли больше, чем нужно. Поэтому снова добавляем число элементов, которые являются общими для пар множеств, $S_j \cap S_k$, для каждой пары $S_j$ и $S_k$; но это уже больше искомой величины. Поэтому вычитаем элементы, общие для каждой тройки множеств, и т. д.) Существует несколько способов доказательства этой формулы, и читателю предлагается найти один из них самостоятельно (см. упр. 25).

Чтобы подсчитать число перестановок из $n$ элементов, не имеющих единичных циклов, рассмотрим $N = n!$ перестановок и обозначим через $S_j$ множество перестановок, в которых элемент $j$ образует единичный цикл. Если $1 \le j_1 < j_2 < \cdots < j_k \le n$, то количество элементов в $S_{j_1}\cap S_{j_2}\cap\cdots\cap S_{j_k}$ — это число перестановок, в которых $j_1, \ldots, j_k$ образуют единичные циклы. Очевидно, что оно равно $(n-k)!$. Таким образом, формула (29) принимает вид

$$n! - \binom{n}{1}(n-1)! + \binom{n}{2}(n-2)! - \binom{n}{3}(n-3)! + \cdots + (-1)^n\binom{n}{n}0!,$$

что согласуется с (25).

Принцип включения и исключения был предложен А. де Муавром (A. de Moivre) [см. его работу *Doctrine of Chances* (London, 1718), 61–63; 3rd ed. (1756, переиздано

Chelsea, 1957), 110–112]. Но значение этого принципа не было по достоинству оценено до тех пор, пока В. А. Витворт (W. A. Whitworth) не популяризировал и не развил его в своей знаменитой книге *Choice and Chance* (Cambridge, 1867).

В разделе 5.1 будет продолжено изучение комбинаторных свойств перестановок.

## УПРАЖНЕНИЯ

**1.** [*02*] Рассмотрите преобразование множества $\{0, 1, 2, 3, 4, 5, 6\}$, которое меняет $x$ на $2x \bmod 7$. Покажите, что это преобразование является перестановкой, и представьте ее в циклической форме.

**2.** [*10*] В тексте раздела показано, как можно присвоить $(a, b, c, d, e, f) \leftarrow (c, d, f, b, e, a)$, выполнив ряд операций замены $(x \leftarrow y)$ и введя одну вспомогательную переменную $t$. Покажите, как сделать то же самое с помощью ряда операций *взаимного обмена* $(x \leftrightarrow y)$ и без вспомогательных переменных.

**3.** [*03*] Вычислите произведение $\left(\begin{smallmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{smallmatrix}\right) \times \left(\begin{smallmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{smallmatrix}\right)$ и запишите результат в виде двухстрочной записи. (Ср. с соотношением (4).)

**4.** [*10*] Выразите $(a\,b\,d)(e\,f)(a\,c\,f)(b\,d)$ в виде произведения непересекающихся циклов.

▶ **5.** [*M10*] В (3) приведено несколько эквивалентных способов представления одной и той же перестановки в циклической форме. Сколько существует таких представлений перестановки, в которых вообще нет единичных циклов?

**6.** [*M23*] Как изменится время выполнения программы A, если отказаться от предположения, что все пустые слова появляются у правого края ввода?

**7.** [*10*] Если на вход программы A подается произведение перестановок (6), то чему равны величины $X$, $Y$, $M$, $N$, $U$ и $V$ из (19)? Каким будет время выполнения программы A без учета ввода-вывода?

▶ **8.** [*23*] Можно ли модифицировать алгоритм B так, чтобы просматривать входные данные слева направо, а не справа налево?

**9.** [*10*] Программы A и B воспринимают одинаковые входные данные и ответ дают, в сущности, в одинаковой форме. Дают ли обе программы *в точности* один и тот же вывод?

▶ **10.** [*M28*] Рассмотрите характеристики времени выполнения программы B, а именно — величины $A$, $B$, ..., $Z$, о которых шла речь в тексте раздела. Выразите общее время выполнения через величины $X$, $Y$, $M$, $N$, $U$, $V$, определенные в (19), и через $F$. Сравните общее время выполнения программ A и B для ввода (6), проведя вычисления, как в упр. 7.

**11.** [*15*] Найдите простое правило, по которому можно записать $\pi^-$ в циклической форме, если перестановка $\pi$ задана в циклической форме.

**12.** [*M27*] (*Транспонирование прямоугольной матрицы.*) Пусть матрица $(a_{ij})$ размера $m \times n$, $m \neq n$, хранится в памяти, как описано в упр. 1.3.2–10, так что значение $a_{ij}$ находится в ячейке $L + n(i-1) + (j-1)$, где $L$ — это ячейка, в которой содержится $a_{11}$. Необходимо найти способ *транспонирования* этой матрицы, чтобы получить матрицу $(b_{ij})$ размера $n \times m$, где $b_{ij} = a_{ji}$ хранится в ячейке $L + m(i-1) + (j-1)$. Таким образом, наша матрица должна быть транспонирована "на себя". (a) Покажите, что преобразование транспонирования переводит значение из ячейки $L + x$ в ячейку $L + (mx \bmod N)$ для всех $x$ из промежутка $0 \leq x < N = mn - 1$. (b) Обдумайте методы выполнения такого транспонирования с помощью компьютера.

▶ **13.** [*M24*] Докажите справедливость алгоритма J.

▶ **14.** [*M34*] Найдите среднее значение величины $A$, которая является одной из составляющих времени выполнения алгоритма J.

**15.** [*M12*] Существует ли перестановка, для которой каноническая циклическая форма без скобок и линейная форма совпадают?

**16.** [*M15*] Пусть задана перестановка 1324 в линейной форме. Преобразуйте ее в каноническую циклическую форму, а затем удалите скобки. Повторяйте эту процедуру до тех пор, пока не придете к исходной перестановке. Какие перестановки получатся в ходе выполнения этого процесса?

**17.** [*M24*] (a) В тексте раздела показано, что среди всех перестановок из $n$ элементов существует всего $n!\,H_n$ циклов. Если эти циклы (включая единичные) записать по отдельности на $n!\,H_n$ листочках бумаги, а затем выбрать наугад один из этих листочков, то чему будет равна средняя длина выбранного таким образом цикла? (b) Запишем $n!$ перестановок на $n!$ листочках бумаги, наугад выберем число $k$ и таким же случайным образом вытянем один из листочков. Чему равна вероятность того, что цикл, содержащий элемент $k$, является $m$-циклом? Чему равна средняя длина цикла, содержащего элемент $k$?

▸ **18.** [*M27*] Чему равна $p_{nkm}$, вероятность того, что перестановка $n$ объектов имеет ровно $k$ циклов длины $m$? Какой будет соответствующая производящая функция $G_{nm}(z)$? Чему равно среднее число $m$-циклов и каким будет среднее квадратичное отклонение? (В тексте раздела рассматривается только случай, когда $m = 1$.)

**19.** [*HM21*] Пользуясь обозначениями из (25), покажите, что число перестановок $P_{n0}$ в точности равно значению $n!/e$, *округленному до ближайшего целого*, для всех $n \geq 1$.

**20.** [*M20*] Пусть все единичные циклы выписаны явно. Сколько существует различных способов представления в виде циклов перестановки, имеющей $\alpha_1$ циклов длины 1, $\alpha_2$ циклов длины 2, ... (см. упр. 5)?

**21.** [*M22*] Чему равна вероятность $P(n; \alpha_1, \alpha_2, \dots)$ того, что перестановка $n$ объектов имеет ровно $\alpha_1$ циклов длины 1, $\alpha_2$ циклов длины 2 и т. д.?

▸ **22.** [*HM34*] (Следующий подход, предложенный Л. Шеппом (L. Shepp) и С. Ф. Ллойдом (S. P. Lloyd), дает удобный и мощный метод решения задач, имеющих отношение к циклическим представлениям случайных перестановок. Вместо того чтобы считать число объектов $n$ фиксированным, а число перестановок — переменным, будем предполагать, что мы независимо выбираем величины $\alpha_1, \alpha_2, \alpha_3 \dots$, рассмотренные в упр. 20 и 21, в соответствии с некоторым распределением вероятностей. Пусть $w$ — любое действительное число между 0 и 1.

a) Предположим, что выбраны случайные переменные $\alpha_1, \alpha_2, \alpha_3, \dots$ согласно правилу, которое гласит: "Вероятность того, что $\alpha_m = k$, равна $f(w, m, k)$", где $f(w, m, k)$ — некоторая функция. Определите функцию $f(w, m, k)$ так, чтобы выполнялись следующие два условия: (i) $\sum_{k \geq 0} f(w, m, k) = 1$, где $0 < w < 1$ и $m \geq 1$; (ii) вероятность того, что $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots = n$ *и* что $\alpha_1 = k_1$, $\alpha_2 = k_2$, $\alpha_3 = k_3$, ..., равна $(1 - w)w^n P(n; k_1, k_2, k_3, \dots)$, где $P(n; k_1, k_2, k_3, \dots)$ определяется в упр. 21.

b) Очевидно, что перестановка, циклическая структура которой имеет вид $\alpha_1, \alpha_2, \alpha_3, \dots$, переставляет ровно $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots$ объектов. Покажите, что если $\alpha_i$, $i \geq 1$, выбираются случайно в соответствии с распределением вероятностей из п. (a), то вероятность того, что $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots = n$, равна $(1 - w)w^n$, а вероятность того, что сумма $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots$ *бесконечна*, равна нулю.

c) Пусть $\phi(\alpha_1, \alpha_2, \dots)$ — произвольная функция от бесконечного числа аргументов $\alpha_1$, $\alpha_2, \dots$. Покажите, что если $\alpha_i$, $i \geq 1$, выбираются в соответствии с распределением вероятностей из п. (a), то среднее значение $\phi$ равно $(1 - w) \sum_{n \geq 0} w^n \phi_n$. Здесь $\phi_n$ обозначает среднее значение $\phi$ для всех перестановок $n$ объектов, а переменная $\alpha_j$ представляет число $j$-циклов в перестановке. [Например, если $\phi(\alpha_1, \alpha_2, \dots) = \alpha_1$, то значением $\phi_n$ будет среднее число единичных циклов в случайной перестановке $n$ объектов. Из (28) следует, что $\phi_n = 1$ для всех $n$.]

d) Используйте этот метод для нахождения среднего числа циклов *четной* длины в случайной перестановке $n$ объектов.

e) Используйте этот метод для решения упр. 18.

**23.** [*HM42*] (Голомб (Golomb), Шепп (Shepp) и Ллойд (Lloyd).) Обозначим через $l_n$ среднюю длину *самого длинного* цикла в перестановке $n$ объектов. Покажите, что $l_n \approx \lambda n + \frac{1}{2}\lambda$, где $\lambda \approx 0.62433$ — константа. Докажите, что $\lim_{n\to\infty}(l_n - \lambda n - \frac{1}{2}\lambda) = 0$.

**24.** [*M41*] Найдите дисперсию величины $A$, которая является одной из характеристик времени выполнения алгоритма J (см. упр. 14).

**25.** [*M22*] Докажите соотношение (29).

▸ **26.** [*M24*] Обобщите принцип включения и исключения, чтобы получить формулу для числа элементов, которые имеются ровно в $r$ из подмножеств $S_1, S_2, \ldots, S_M$. (В тексте раздела рассматривается только случай $r = 0$.)

**27.** [*M20*] Используйте принцип включения и исключения для подсчета количества таких целых чисел $n$ в интервале $0 \le n < am_1m_2\ldots m_t$, которые не делятся ни на одно из $m_1, m_2, \ldots, m_t$. Здесь $m_1$, $m_2$, ..., $m_t$ и $a$ — положительные целые числа, такие, что $m_j \perp m_k$, если $j \ne k$.

**28.** [*M21*] (И. Капланси (I. Kaplansky).) Если "перестановку Иосифа", определенную в упр. 1.3.2–22, выразить в циклической форме, то получим (1 5 3 6 8 2 4)(7), где $n = 8$ и $m = 4$. Покажите, что в общем случае эта перестановка представляет собой произведение $(n\ n{-}1\ \ldots\ 2\ 1)^{m-1}(n\ n{-}1\ \ldots\ 2)^{m-1}\ldots(n\ n{-}1)^{m-1}$.

**29.** [*M25*] Докажите, что циклическую форму перестановки Иосифа при $m = 2$ можно получить, сначала выразив "двойную" перестановку $\{1, 2, \ldots, 2n\}$, которая переводит $j$ в $(2j) \bmod (2n+1)$, в циклической форме, а затем рассмотрев циклы в обратном порядке и убрав все числа, которые больше $n$. Например, при $n = 11$ двойная перестановка будет иметь вид (1 2 4 8 16 9 18 13 3 6 12)(5 10 20 17 11 22 21 19 15 7 14), а перестановка Иосифа — (7 11 10 5)(6 3 9 8 4 2 1).

**30.** [*M24*] Используя упр. 29, покажите, что фиксированными элементами перестановки Иосифа при $m = 2$ будут в точности числа $(2^{d-1} - 1)(2n+1)/(2^d - 1)$ для всех таких положительных $d$, при которых это выражение дает целые числа.

**31.** [*HM33*] Обобщая упр. 29 и 30, докажите, что $k$-м казненным для произвольных $m$ и $n$ будет тот, кто находится в позиции $x$, где $x$ можно вычислить следующим образом: положить $x \leftarrow km$, а затем присваивать $x \leftarrow \lfloor (m(x-n)-1)/(m-1) \rfloor$ до тех пор, пока $x \le n$. В результате среднее число фиксированных элементов для $1 \le n \le N$ и фиксированного $m$ при $N \to \infty$ будет стремиться к $\sum_{k\ge 1}(m-1)^k/(m^{k+1} - (m-1))^k$. [Так как это значение лежит между $(m-1)/m$ и 1, перестановка Иосифа имеет немного меньше фиксированных элементов, чем случайные перестановки.]

**32.** [*M25*] (a) Докажите, что для любой перестановки $\pi = \pi_1\pi_2\ldots\pi_{2m+1}$ вида

$$\pi = (2\ 3)^{e_2}(4\ 5)^{e_4}\ldots(2m\ 2m{+}1)^{e_{2m}}(1\ 2)^{e_1}(3\ 4)^{e_3}\ldots(2m{-}1\ 2m)^{e_{2m-1}},$$

где каждое $e_k$ — либо 0, либо 1, имеет место $|\pi_k - k| \le 2$ для $1 \le k \le 2m+1$.

(b) Для любой заданной перестановки $\rho$ элементов $\{1, 2, \ldots, n\}$ постройте перестановку $\pi$ указанного вида, такую, чтобы произведение $\rho\pi$ давало единственный цикл. Таким образом, каждая перестановка "близка" к циклу.

**33.** [*M33*] Пусть $m = 2^{2^l}$, а $n = 2^{2l+1}$. Покажите, как построить последовательность перестановок $(\alpha_{j1}, \alpha_{j2}, \ldots, \alpha_{jn}; \beta_{j1}, \beta_{j2}, \ldots, \beta_{jn})$ для $0 \le j < m$, имеющих следующее свойство "ортогональности":

$$\alpha_{i1}\beta_{j1}\alpha_{i2}\beta_{j2}\ldots\alpha_{in}\beta_{jn} = \begin{cases} (1\,2\,3\,4\,5), & \text{если } i = j; \\ (), & \text{если } i \ne j. \end{cases}$$

Каждое $\alpha_{jk}$ и $\beta_{jk}$ должно быть перестановкой чисел $\{1, 2, 3, 4, 5\}$.

▸ **34.** [*M25*] (*Транспонирующие блоки данных.*) Одной из самых распространенных перестановок, использующихся на практике, является переход от $\alpha\beta$ к $\beta\alpha$, где $\alpha$ и $\beta$ — это подстроки массива. Другими словами, если $x_0x_1\ldots x_{m-1} = \alpha$ и $x_mx_{m+1}\ldots x_{m+n-1} = \beta$, то нужно заменить массив $x_0x_1\ldots x_{m+n-1} = \alpha\beta$ массивом $x_mx_{m+1}\ldots x_{m+n-1}x_0x_1\ldots x_{m-1} = \beta\alpha$. Это перестановка на множестве $\{0, 1, \ldots, m+n-1\}$, которая переводит $k$ в $(k+m) \bmod (m+n)$. Покажите, что каждая такая перестановка "с циклическим сдвигом" имеет простую циклическую структуру, и используйте эту структуру для разработки простого алгоритма получения нужной перестановки.

**35.** [*M30*] В продолжение предыдущего упражнения положим $x_0x_1\ldots x_{l+m+n-1} = \alpha\beta\gamma$, где $\alpha$, $\beta$ и $\gamma$ — строки длины $l$, $m$ и $n$ соответственно, и предположим, что нужно заменить $\alpha\beta\gamma$ на $\gamma\beta\alpha$. Покажите, что соответствующая перестановка имеет подходящую циклическую структуру, которая позволяет получить эффективный алгоритм. [Упр. 34 рассматривалось как частный случай при $m = 0$.] *Указание.* Рассмотрите замену $(\alpha\beta)(\gamma\beta)$ на $(\gamma\beta)(\alpha\beta)$.

**36.** [*27*] Напишите для `MIX` подпрограмму, реализующую алгоритм, который приведен в ответе к упр. 35, и проанализируйте время его выполнения. Сравните этот алгоритм с более простым методом, в котором осуществляется переход от $\alpha\beta\gamma$ к $(\alpha\beta\gamma)^R = \gamma^R\beta^R\alpha^R$ и к $\gamma\beta\alpha$, где $\sigma^R$ обозначает полное обращение строки $\sigma$, т. е. строка читается в обратном порядке.