Машина с произвольным доступом к памяти

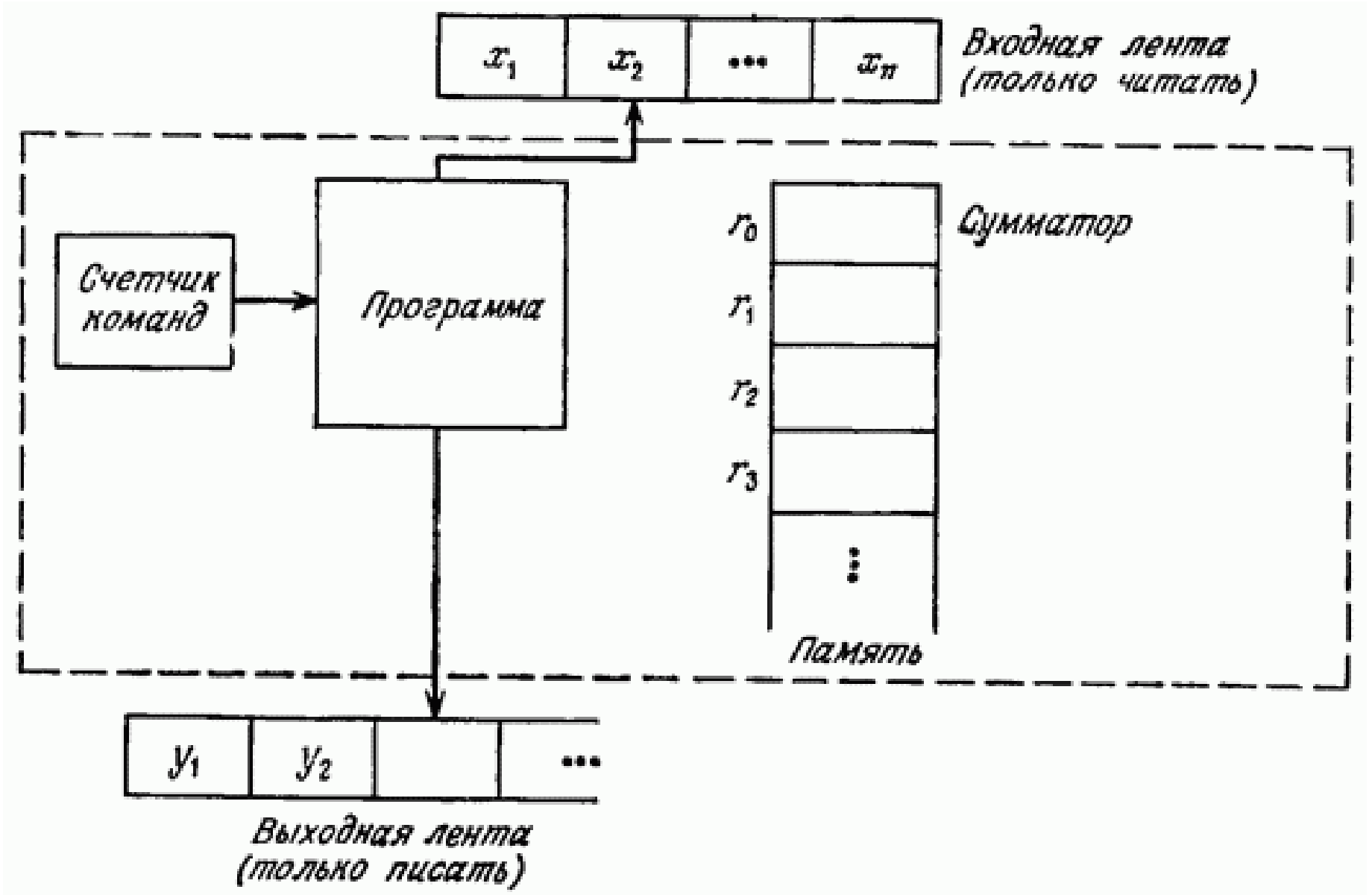
Random access machine (RAM)

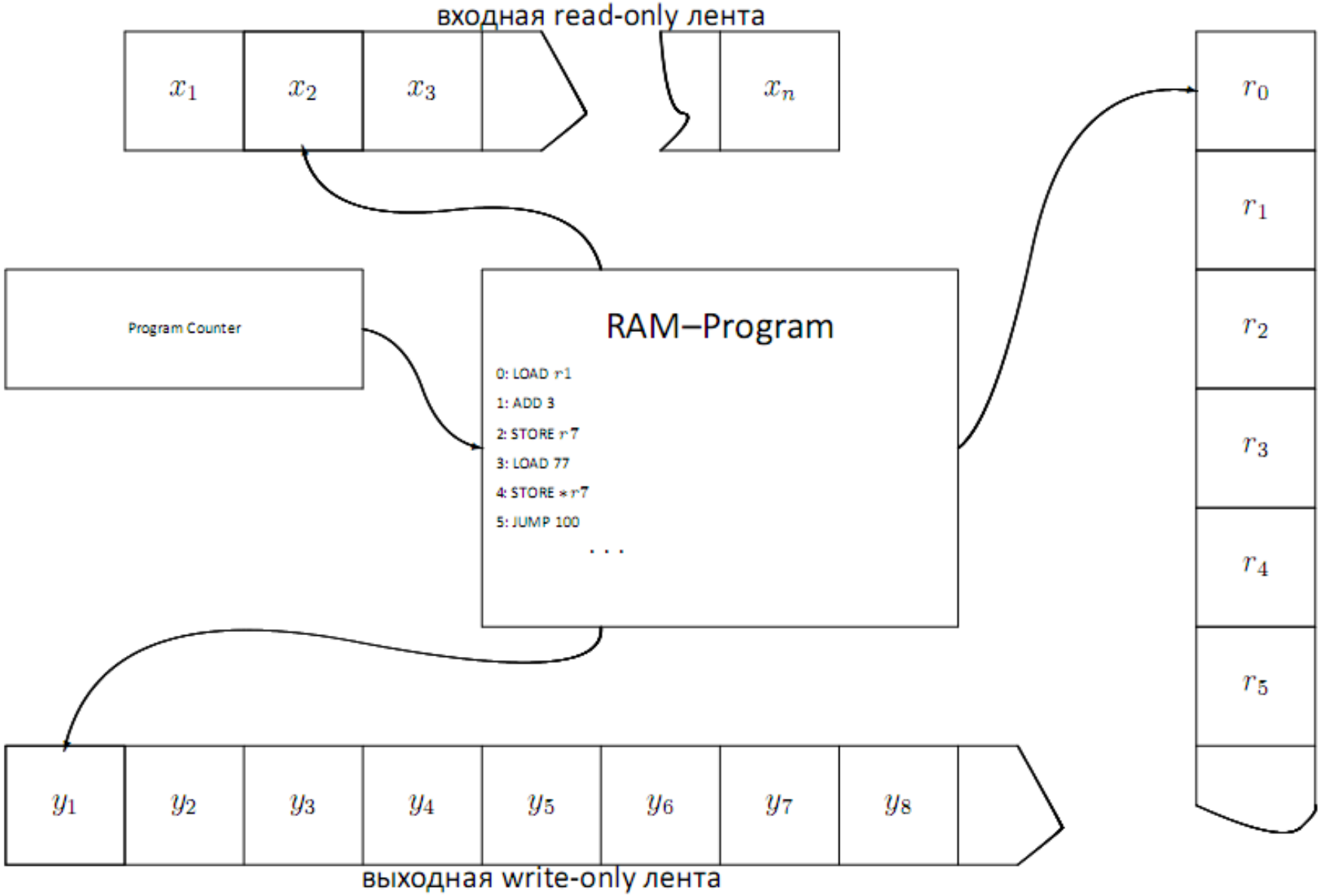
- Машина с произвольным доступом к памяти
- Равнодоступная адресная машина
- RAM-машина

Идеализированная модель ЭВМ предложенная в 70-х годах XX века с целью моделирования реальных вычислительных машин и анализа сложности вычислений.

Модель машины с одним сумматором, команды программы не могут изменять сами себя.

- РАМ состоит из входной ленты, с которой она может только считывать, выходной ленты, на которую она может только записывать, и памяти.
- Входная лента представляет собой последовательность клеток, каждая из которых может содержать целое число (возможно, отрицательное). Всякий раз, когда символ считывается с входной ленты, ее читающая головка сдвигается на одну клетку вправо.
- Выход представляет собой ленту, на которую машина может только записывать; она разбита на клетки, которые вначале все пусты. При выполнении команды записи в той клетке выходной ленты, которую в текущий момент обозревает ее головка, печатается целое число и головка затем сдвигается на одну клетку вправо. Как только выходной символ записан, его уже нельзя изменить.


x_1
x_2
•••
x_n
Входная лента
(только читать)
Счетчик команд
Программа
r_0
r_1
r_2
r_3
Сумматор
Память
y_1
y_2
•••
Выходная лента
(только писать)


входная read-only лента
x_1
x_2
x_3
x_n
Program Counter
RAM–Program
0: LOAD $r1$
1: ADD 3
2: STORE $r7$
3: LOAD 77
4: STORE $*r7$
5: JUMP 100
. . .
r_0
r_1
r_2
r_3
r_4
r_5
y_1
y_2
y_3
y_4
y_5
y_6
y_7
y_8
выходная write-only лента

Память состоит из последовательности регистров r_0, r_1, …, r_n каждый из которых способен хранить произвольное целое число. На число регистров, которые можно использовать, верхней границы нет.

Такая идеализация допустима в случаях, когда

1. размер задачи достаточно мал, чтобы она поместилась в основную память вычислительной машины,
2. целые числа, участвующие в вычислении, достаточно малы, чтобы их можно было помещать в одну ячейку.

Состоянием машины или *конфигурацией* называется последовательность чисел, записанных в регистрах r_0, r_1, …, r_n .

Функционирование машины заключается в изменении конфигураций путем выполнения *команд* в порядке их написания.

Если вычисление остановилось, то последовательность содержимого регистров r_0, r_1, …, r_n называется *заключительной конфигурацией*.

- Программа для РАМ (или РАМ-программа) не записывается в память. Поэтому предполагается, что программа не изменяет сама себя.
- Программа является последовательностью (возможно) помеченных команд. Точный тип команд, допустимых в программе, не слишком важен, они напоминают те, которые обычно встречаются в реальных вычислительных машинах.
- Имеются арифметические команды, команды ввода-вывода, косвенная адресация (например, для индексации массивов) и команды разветвления.

- Все вычисления производятся в первом регистре r_0 называемом *сумматором*, который, как и всякий другой регистр памяти, может хранить произвольное целое число.
- Пример набора команд для РАМ приведен далее. Список команд зависит от постановки задачи, но похож на типичный язык *ассемблера*.
- Каждая команда состоит из двух частей — *кода операции* и *адреса*.

Таблица команд РАМ

Код операции	Адрес
1. LOAD	операнд
2. STORE	операнд
3. ADD	операнд
4. SUB	операнд
5. MULT	операнд
6. DIV	операнд
7. READ	операнд
8. WRITE	операнд
9. JUMP	метка
10. JGTZ	метка
11. JZERO	метка
12. HALT	

Каждая команда состоит из двух частей — кода операции и адреса.

Операнд может быть одного из следующих типов:

- целое число
- Регистр, содержащий целое число
- Регистр, содержащий адрес (косвенная адресация)

1. LOAD a	$c(0) \leftarrow v(a)$
2. STORE i	$c(i) \leftarrow c(0)$
STORE $*i$	$c(c(i)) \leftarrow c(0)$
3. ADD a	$c(0) \leftarrow c(0) + v(a)$
4. SUB a	$c(0) \leftarrow c(0) - v(a)$
5. MULT a	$c(0) \leftarrow c(0) \times v(a)$
6. DIV a	$c(0) \leftarrow \lfloor c(0)/v(a) \rfloor$ [1])
7. READ i	$c(i) \leftarrow$ очередной входной символ.
READ $*i$	$c(c(i)) \leftarrow$ очередной входной символ. В обоих случаях головка входной ленты сдвигается на одну клетку вправо.
8. WRITE a	$v(a)$ печатается в той клетке выходной ленты, которую в данный момент обозревает ее головка. Затем эта головка сдвигается на одну клетку вправо.
9. JUMP b	Счетчик команд устанавливается на команду с меткой b.
10. JGTZ b	Если $c(0) > 0$, то счетчик команд устанавливается на команду с меткой b, в противном случае на следующую команду.
11. JZERO b	Если $c(0) = 0$, то счетчик команд устанавливается на команду с меткой b, в противном случае на следующую команду.
12. HALT	Работа прекращается.

При выполнении любой из первых восьми команд *счетчик команд* увеличивается на единицу. Поэтому команды в данной программе выполняются последовательно, до тех пор пока не встретится команда JUMP или HALT, либо JGTZ при содержимом сумматора, большем нуля, либо JZERO при содержимом сумматора, равном нулю.

Вначале счетчик команд установлен на первую команду, а выходная лента пуста. После выполнения *k*-й команды счетчик команд автоматически переходит на *k*+1 (т.е. на следующую команду), если *k*-я команда не была командой вида JUMP, HALT, JGTZ или JZERO.

Моделирование циклов для PAM

```
   ...
for i in range( 1..78 ):
    тело цикла
   ...
```

$\Rightarrow$

```
          ...
0777:   LOAD     1
0778:   STORE    r17
0779:   начало тела цикла
          ...
1035:   LOAD     1
1036:   ADD      r17
1037:   STORE    r17
1038:   конец тела цикла
1039:   LOAD     78
1040:   SUB      r17
1041:   JGTZ     0779
          ...
```

Простое вычисление R_1*R_2 на RAM

Вход: Натуральные R_1, R_2
Выход: $R_1 \times R_2$

$R \leftarrow 0$
for all $i \in 1..R_1$ **do**
  $R \leftarrow R + R_2$
end for
return R

- Предположим, что программа *P* всегда считывает с входной ленты *n* целых чисел и записывает на выходную ленту не более одного целого числа. Пусть $x_1, x_2, \ldots, x_n$ - целые числа, находящиеся в *n* первых клетках входной ленты, и пусть программа *P* записывает *y* в первую клетку выходной ленты, а затем через некоторое время останавливается. Тогда говорят, что *P* вычисляет функцию $f(x_1, x_2, \ldots, x_n) = y$.

Другой способ интерпретировать программу для РАМ - это посмотреть на нее с точки зрения допускаемого ею языка.

Алфавит — это конечное множество символов, *язык* — множество цепочек (слов) алфавита. Символы алфавита можно представить целыми числами 1,2,…, k при некотором k.

Данная РАМ допускает (воспринимает) язык в следующем смысле. Пусть на входной ленте находится цепочка $s = a_1 a_2 \ldots a_n$, причем символ a_1 расположен в первой клетке, a_2 во второй и т. д., а в $(n+1)$ клетке расположен 0-й символ, который будет использоваться в качестве концевого маркера, т. е. маркера конца входной цепочки.

- Входная цепочка *s* *допускается* РАМ-программой *P*, если *P* прочитывает все её символы и концевой маркер, пишет 1 в первой клетке выходной ленты и останавливается.
- Для входных цепочек, не принадлежащих языку, допускаемому программой *P*, она может напечатать на выходной ленте символ, отличный от 1, и остановиться, а может даже и не остановиться вообще.
- Языком, *допускаемым программой P*, называется множество всех цепочек (слов), допускаемых ею.

Несмотря на «примитивный ассемблер» RAM-машины потенциально мощнее любых существующих компьютеров и физически нереализуемы, т.к. оперируют бесконечной памятью, где доступ к любой ячейке-регистру осуществляется мгновенно при выполнении соответствующей инструкции, и каждая ячейка этой памяти может содержать произвольное целое число (т.е. не ограничена по размеру). Но эта модель уже дает возможность вводить более или менее формальные определения времени выполнения программы (как число тактов до остановки машины) и, соответственно, сложности алгоритма.

Язык РАМ-программ содержит основные процедуры языков программирования и позволяет устраивать композицию (соединение) программ и использовать программы в качестве подпрограмм других программ. Это является основанием для предположения о том, что введенный класс вычислимых функций в точности отвечает классу алгоритмически вычислимых функций.

Данное предположение называется *Тезисом Черча* (для РАМ). Так же как и тезис Тьюринга, данный тезис доказать нельзя, однако принятие его позволяет истолковывать утверждения о несуществовании РАМ для решения конкретных задач как утверждения о несуществовании алгоритмов вообще.