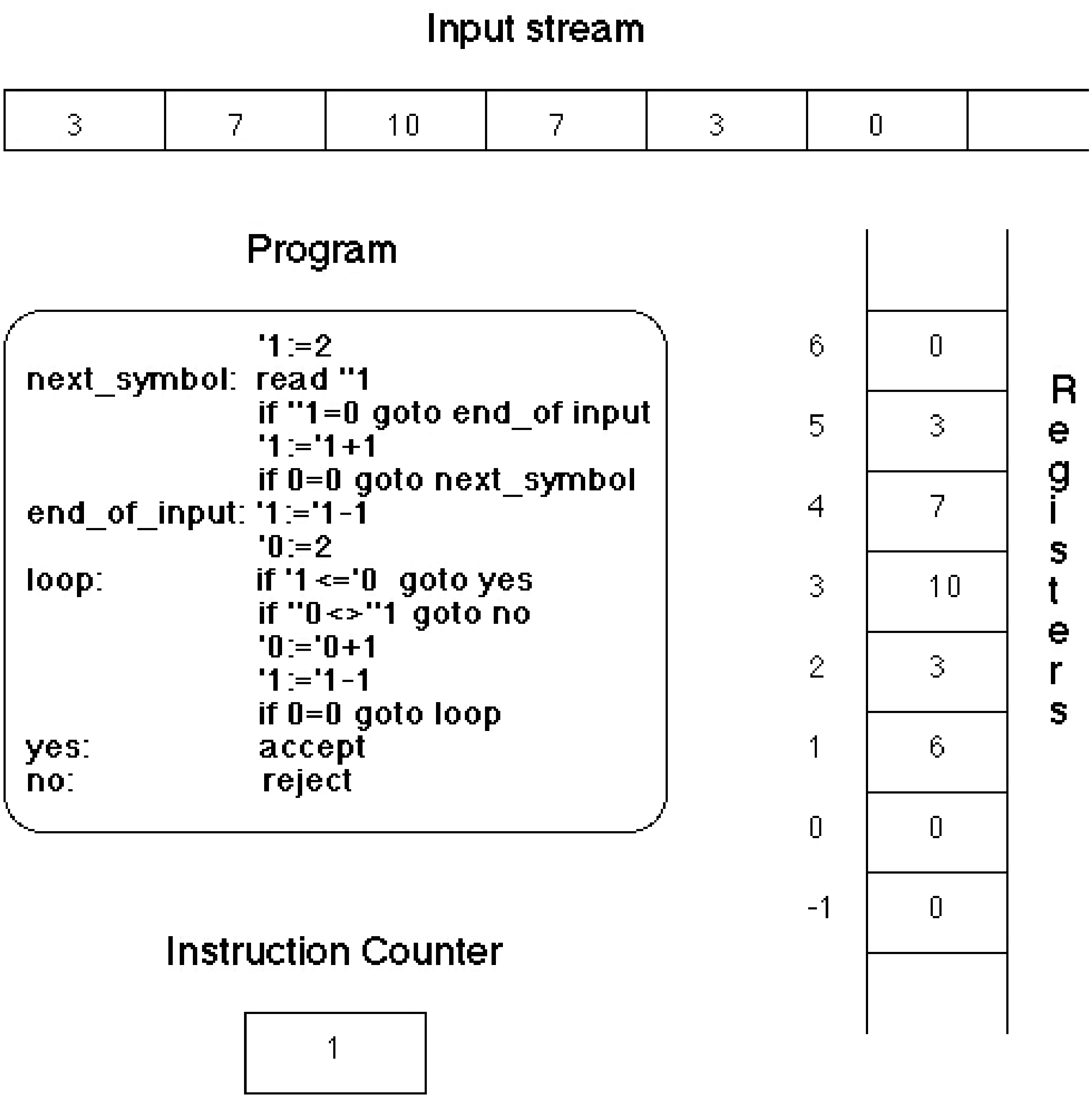
Random Access Machines (RAMs)



Syntax of Ram programs:

$$
\begin{aligned}
\mathit{program} &= \mathit{instruction}\ \mathit{program} \mid \mathit{instruction} \\
\mathit{instruction} &= [\,\mathit{label}\ \texttt{:}\,]\ (\texttt{accept} \\
&\qquad \mid \texttt{reject} \\
&\qquad \mid \texttt{read}\ \mathit{l\_value} \\
&\qquad \mid \mathit{l\_value}\ \texttt{:=}\ \mathit{r\_value}\ \mathit{arithmetic\_op}\ \mathit{r\_value} \\
&\qquad \mid \texttt{if}\ \mathit{r\_value}\ \mathit{relational\_op}\ \mathit{r\_value}\ \texttt{goto}\ \mathit{label}\,) \\
\mathit{l\_value} &= \texttt{'}\mathit{integer} \mid \texttt{"}\,\mathit{integer} \\
\mathit{r\_value} &= \mathit{integer} \mid \texttt{'}\mathit{integer} \mid \texttt{"}\,\mathit{integer} \\
\mathit{arithmetic\_op} &= \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{div} \\
\mathit{relational\_op} &= \texttt{=} \mid \texttt{<>} \mid \texttt{<=} \mid \texttt{<} \\
\mathit{label} &= \mathit{alphanumeric\_sequence}.
\end{aligned}
$$

Semantics of RAM programs:

- Infinitely many *registers*.
- Each register indexed by integer *address*.
- Initially all registers contain 0.
- Input is a stream of integers.

Instruction counter says which instruction to execute next (number them from 0 onwards).

State of the RAM: formally a function

$$s : \mathbb{Z} \to \mathbb{Z}$$

For every integer i,

$$s(i) = \text{contents of register } i.$$

Note: $s(i) = 0$ for all but finitely many i.

At each step, state of machine and instruction counter are updated.

$\langle \text{state}, \text{instruction counter} \rangle$ play role of configuration in Turing machine model.

l-**values and** r-**values:** L, R respectively,

- L evaluates to an *address* a.
- R evaluates to a *value* v.

In context s (the *state*):

$$a = \begin{cases} k & \text{if } L = \texttt{'}k, \\ s(k) & \text{if } L = \texttt{"}k. \end{cases}$$

$$v = \begin{cases} k & \text{if } R = k, \\ s(k) & \text{if } R = \texttt{'}k, \\ s(s(k)) & \text{if } R = \texttt{"}k. \end{cases}$$

···	10	···	12	2	0	···	6	···	−5	···
	−5			0	1		10		24	

If $k = 24$:

- as an l-value: $\texttt{'}k = 24$, $\texttt{"}k = -5$,
- as an r-value: $\texttt{'}k = -5$, $\texttt{"}k = 10$.

Recognizing languages:

Given alphabet Σ, encode as $1, 2, 3, \ldots, |\Sigma|$.

Encode $\not{b}$ (end of input) by 0.

Example:

$$\begin{array}{lcccccccc} \Sigma & = & \{ & a, & b, & 0, & 1, & \$ & \} \\ & & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ \text{Code} & & & 1, & 2, & 3, & 4, & 5 & \end{array}$$

As given, RAMs don't compute functions, but could just add a `write` instruction.

Example: A RAM that accepts $\{a^n b^n \mid n \in \mathbb{N}\}$.

Slight nuisance: RAMs only read integers.

- Just replace a by 1 and b by 2, language is now $\{1^n 2^n \mid n \in \mathbb{N}\}$.

High level description of algorithm:

```
count := 0
while input is a 1
   count := count + 1
while not at end of input
   if input is a 2
      count := count - 1
   else
      reject
if count = 0
   accept
else
   reject
```

RAM program: hold *count* in register 0, read current input into register 1.

```
start:  read '1
        if '1 = 1 goto ones
        if '1 = 2 goto twos
        if '1 = 0 goto check
        if 0 = 0 goto no
 ones:  '0 = '0 + 1
        if 0 = 0 goto start
 twos:  '0 = '0 − 1
        read '1
        if '1 = 2 goto twos
        if '1 = 0 goto check
        if 0 = 0 goto no
check:  if '0 = 0 goto yes
   no:  reject
  yes:  accept
```

Palindromes:

```
               '1 := 2
next_symbol:   read "1
               if "1 = 0 goto end_of_input
               '1 := '1 + 1
               if 0 = 0 goto next_symbol
end_of_input:  '1 := '1 - 1
               '0 := 2
loop:          if '1 <= '0 goto yes
               if "0 <> "1 goto no
               '0 := '0 + 1
               '1 := '1 - 1
               if 0 = 0 goto loop
yes:           accept
no:            reject
```

Bounded RAMs: Only difference is that registers restricted to contain integers in some bounded range $\{-N, -N+1, \ldots, N-1, N\}$.

Obvious fact: Bounded RAMs are no more powerful than standard ones.

Question: What about the converse?

Fact: Bounded RAMs cannot recognize palindromes, so they are less powerful than RAMs.

Equivalence of Turing machines and RAMs:

THEOREM Let L be a language over some alphabet Σ. If there is a RAM that accepts L, then there is a Turing machine that also accepts L.

Three-register RAMs: Just three registers,

v_{-1}	v_0	v_1
-1	0	1

State is a function from $\{-1, 0, 1\} \rightarrow \mathbb{Z}$.

- only *l_values* allowed are `'-1`, `'0`, and `'1`;
- only *r_values* allowed are `'-1`, `'0`, `'1`, and signed decimal constants;
- 'indirect addressing' forbidden.

THEOREM Let L be a language over some alphabet Σ. If there is a Turing machine that accepts L, then there is a three-register RAM that also accepts L.

Recursively enumerable languages:

• $\mathcal{C}_{\mathsf{TM}}$ denotes class of languages that are accepted by some Turing machine, i.e.,

$$\mathcal{C}_{\mathsf{TM}} = \{L(M) \mid M \text{ is a Turing machine}\}.$$

• $\mathcal{C}_{\mathsf{RAM}}$ denotes similar class for RAMs.

• $\mathcal{C}_{\mathsf{3RAM}}$ denotes similar class for three-register RAMs.

Theorems above show

$$\mathcal{C}_{\mathsf{TM}} = \mathcal{C}_{\mathsf{RAM}} = \mathcal{C}_{\mathsf{3RAM}}.$$

(Tacit assumption: we have fixed a common alphabet.)

Languages in $\mathcal{C}_{\mathsf{TM}}$ called *recursively enumerable* (usually abbreviated to r.e.).