Random Access Machines

Random Access Machine

A **Random Access Machine** (**RAM**) is an idealized model of a computer.

It consists of the following parts:

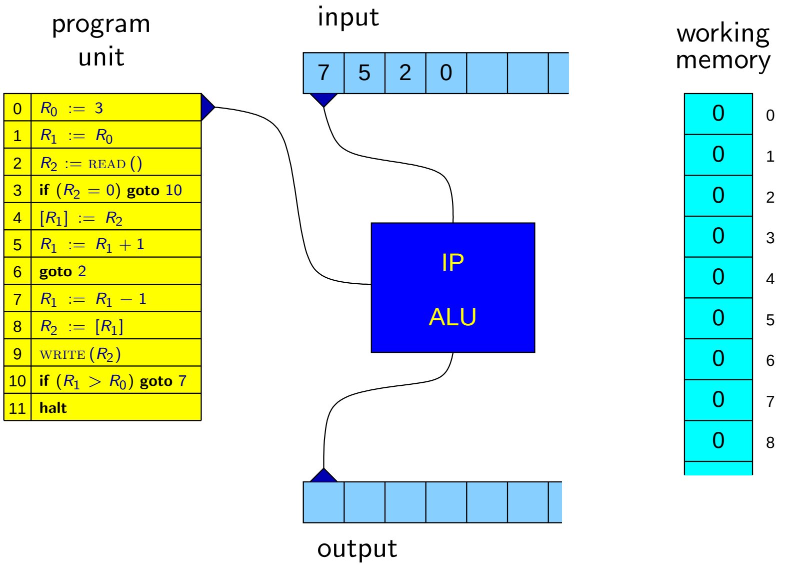
- **Program unit** – contains a program for the RAM and a pointer to the currently executed instruction
- **Working memory** consists of cells numbered $0, 1, 2, \ldots$

  These cells will be denoted $R_0, R_1, R_2, \ldots$

  The content of the cells can be read and written to.
- **Input tape** – read-only
- **Output tape** – write-only

The cells of memory, as well as the cells of input and output tapes contain integers (i.e., elements of set $\mathbb{Z}$) as their values.

Random Access Machine



0	$R_0 := 3$
1	$R_1 := R_0$
2	$R_2 :=$ READ ()
3	**if** $(R_2 = 0)$ **goto** 10
4	$[R_1] := R_2$
5	$R_1 := R_1 + 1$
6	**goto** 2
7	$R_1 := R_1 - 1$
8	$R_2 := [R_1]$
9	WRITE (R_2)
10	**if** $(R_1 > R_0)$ **goto** 7
11	**halt**

Random Access Machine

Overview of instructions:

$R_i := c$	– assignment of a constant
$R_i := R_j$	– assignment
$R_i := [R_j]$	– load (reading from memory)
$[R_i] := R_j$	– store (writing to memory)
$R_i := R_j \mathit{op} R_k$ or $R_i := R_j \mathit{op}\ c$	– arithmetic instructions, $\mathit{op} \in \{+, -, *, /\}$
if $(R_i \mathit{rel} R_j)$ **goto** ℓ or **if** $(R_i \mathit{rel}\ c)$ **goto** ℓ	– conditional jump, $\mathit{rel} \in \{=, \neq, \leq, \geq, <, >\}$
goto ℓ	– unconditional jump
$R_i := \textsc{read}()$	– reading from input
$\textsc{write}(R_i)$	– writing to output
halt	– program termination

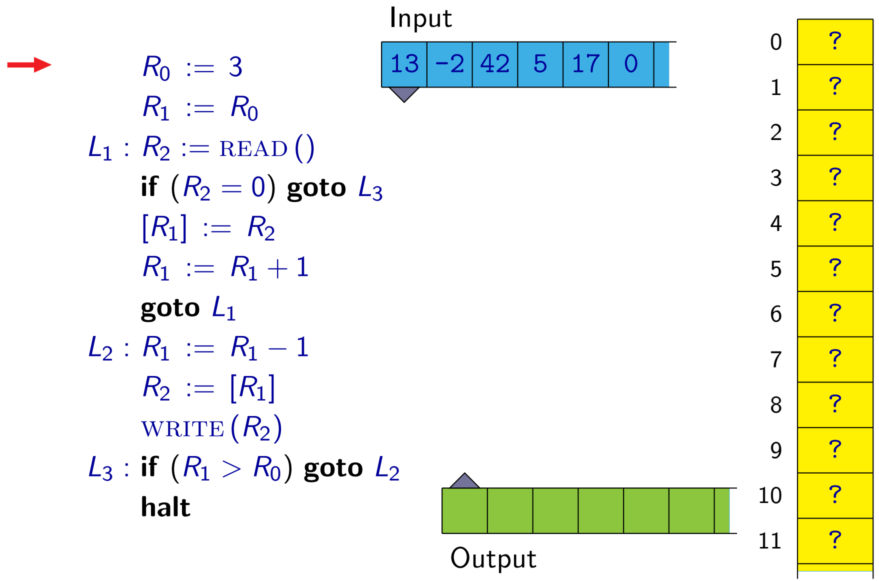
Random Access Machine

Examples of instructions:

$R_5 := 42$	– assignment of a constant
$R_{12} := R_3$	– assignment
$R_8 := [R_2]$	– load (reading from memory)
$[R_{15}] := R_9$	– store (writing to memory)
$R_7 := R_3 + R_6$	– arithmetic instruction
$R_{18} := R_{18} - 1$	– arithmetic instruction
if $(R_4 \geq R_1)$ **goto** 2801	– conditional jump
if $(R_2 \neq 0)$ **goto** 3581	– conditional jump
goto 537	– unconditional jump
$R_{23} := \textsc{read}()$	– reading from input
$\textsc{write}(R_{17})$	– writing to output
halt	– program termination

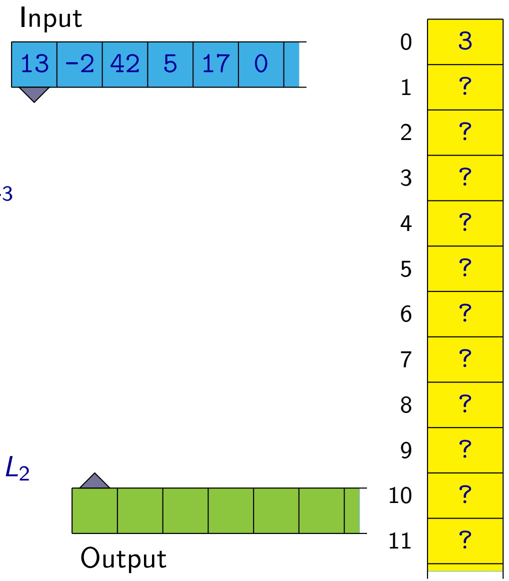
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 := \textsc{read}()$
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        $\textsc{write}(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



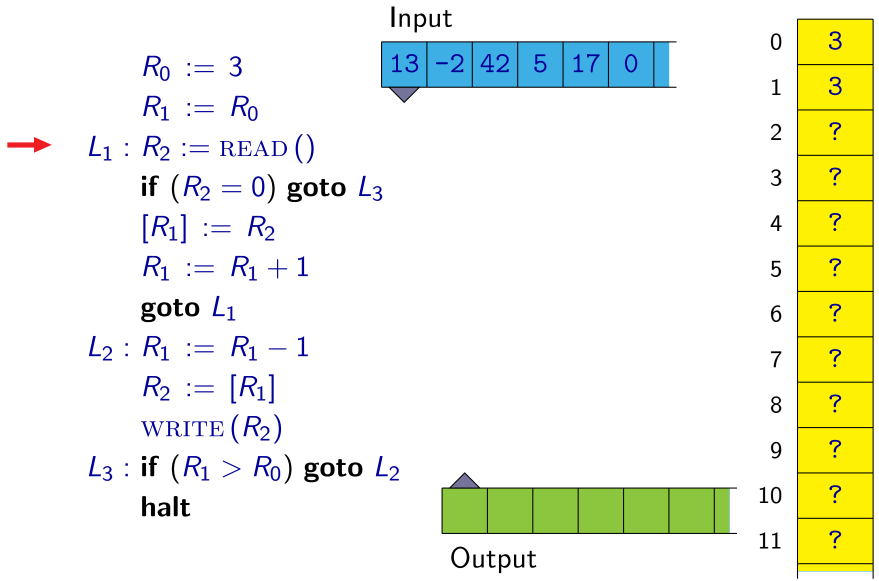
Random Access Machine

```
       $R_0 := 3$
→      $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       if ($R_2 = 0$) goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE ($R_2$)
$L_3$: if ($R_1 > R_0$) goto $L_2$
       halt
```



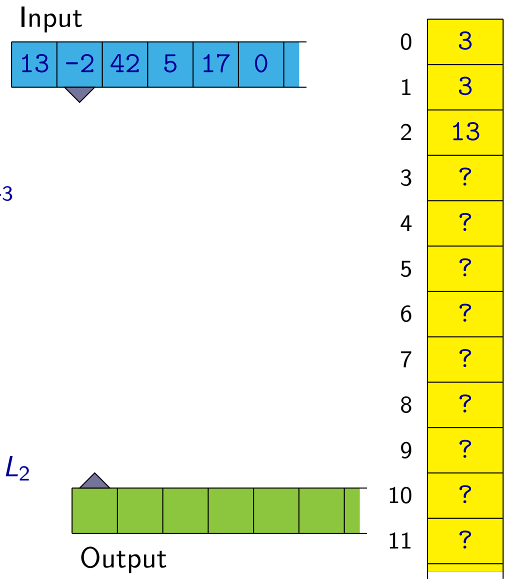
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



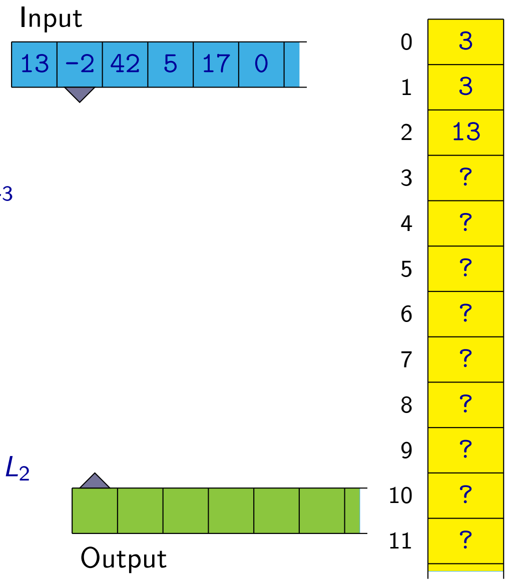
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE ($R_2$)
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



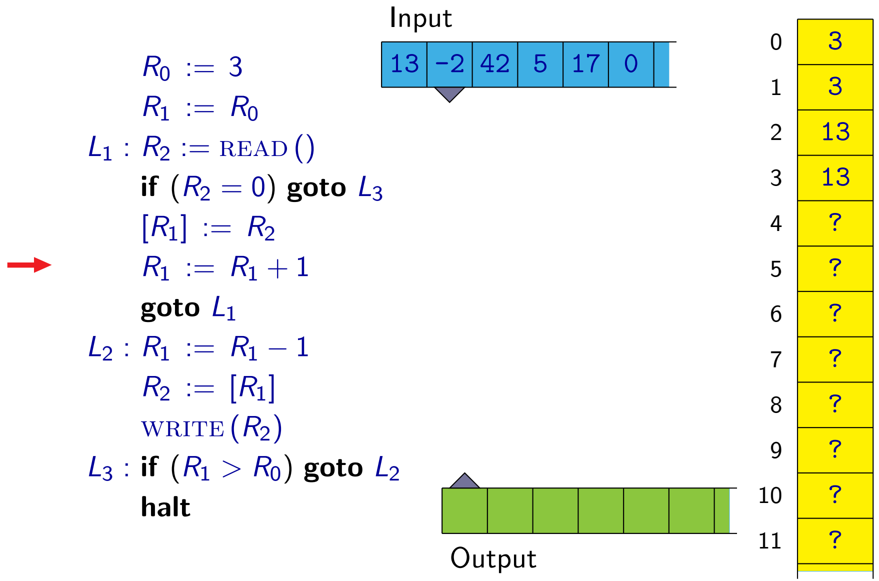
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$ : $R_2 := \text{READ}()$
       if $(R_2 = 0)$ goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       $\text{WRITE}(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
       halt
```



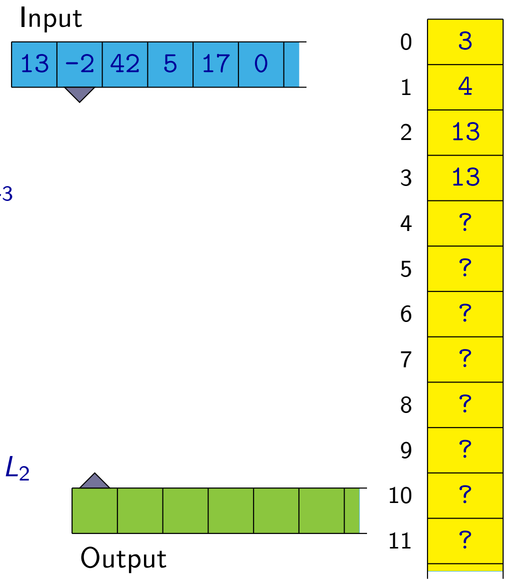
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 := \text{READ}()$
       if $(R_2 = 0)$ goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       $\text{WRITE}(R_2)$
$L_3$: if $(R_1 > R_0)$ goto $L_2$
       halt
```



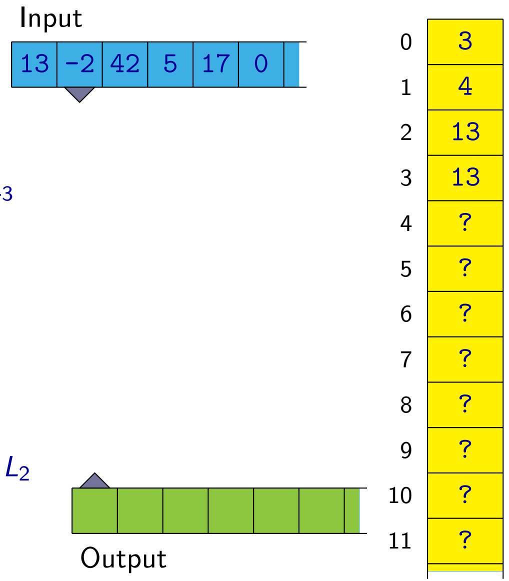
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ()
       if ($R_2 = 0$) goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
→      goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE($R_2$)
$L_3$: if ($R_1 > R_0$) goto $L_2$
       halt
```



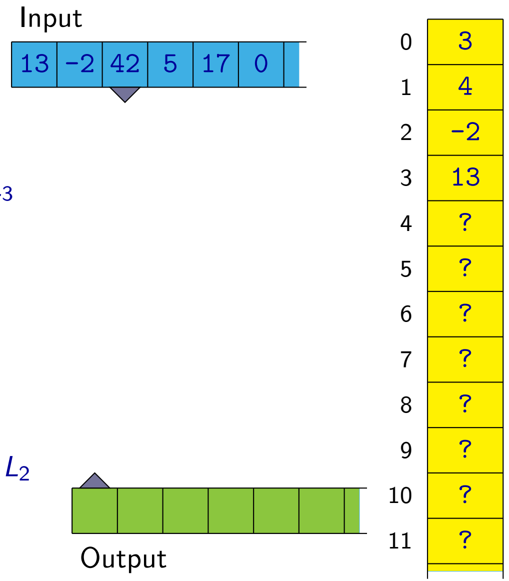
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
→ $L_1$ : $R_2 := \text{READ}()$
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        $\text{WRITE}(R_2)$
  $L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



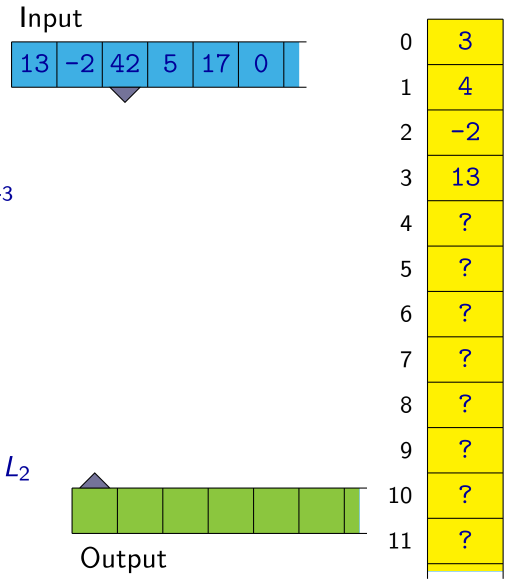
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
$L_1$ :  $R_2 := \text{READ}()$
         if $(R_2 = 0)$ goto $L_3$
         $[R_1] := R_2$
         $R_1 := R_1 + 1$
         goto $L_1$
$L_2$ :  $R_1 := R_1 - 1$
         $R_2 := [R_1]$
         $\text{WRITE}(R_2)$
$L_3$ :  if $(R_1 > R_0)$ goto $L_2$
         halt
```



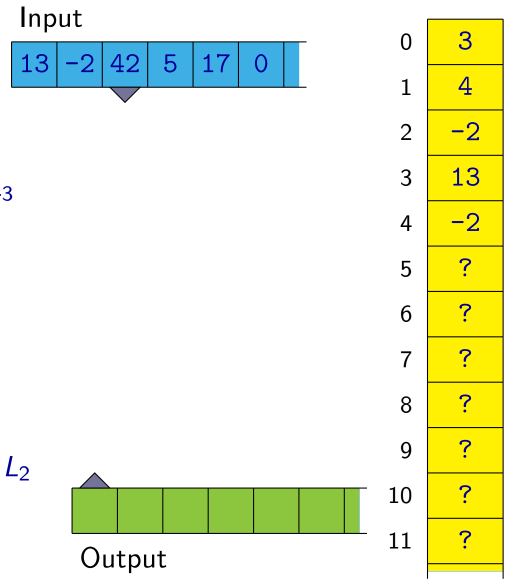
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



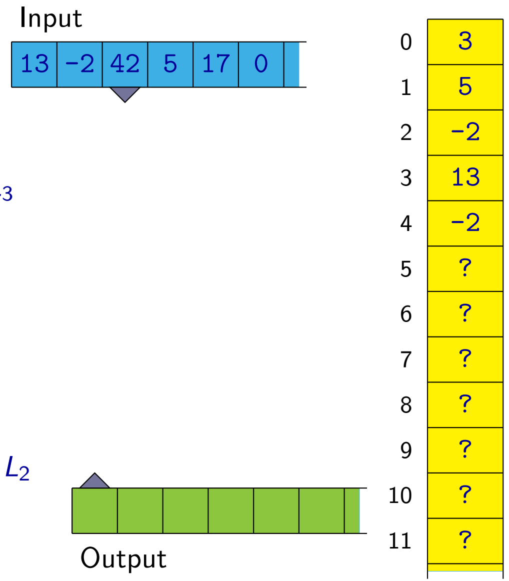
Random Access Machine

```
           $R_0 := 3$
           $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
           if ($R_2 = 0$) goto $L_3$
           $[R_1] := R_2$
→          $R_1 := R_1 + 1$
           goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
           $R_2 := [R_1]$
           WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
           halt
```



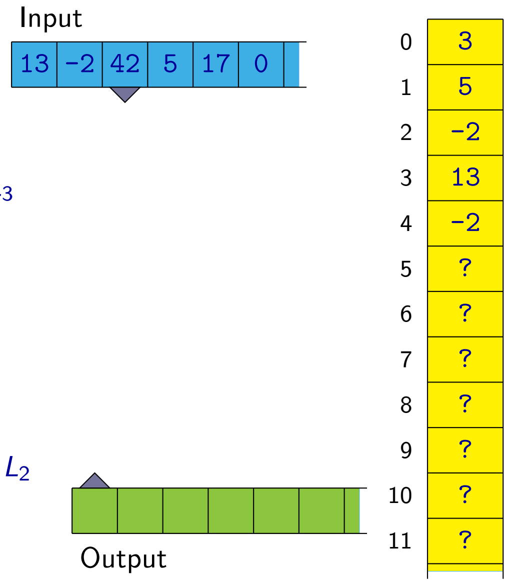
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 := \textsc{read}()$
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
→       **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        $\textsc{write}(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



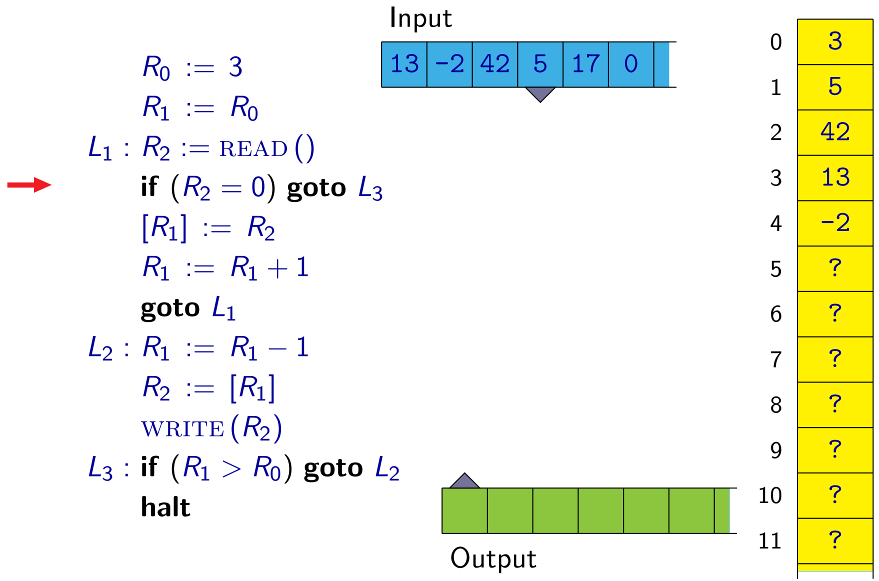
Random Access Machine

```
        R_0 := 3
        R_1 := R_0
→ L_1 : R_2 := READ()
        if (R_2 = 0) goto L_3
        [R_1] := R_2
        R_1 := R_1 + 1
        goto L_1
  L_2 : R_1 := R_1 − 1
        R_2 := [R_1]
        WRITE(R_2)
  L_3 : if (R_1 > R_0) goto L_2
        halt
```



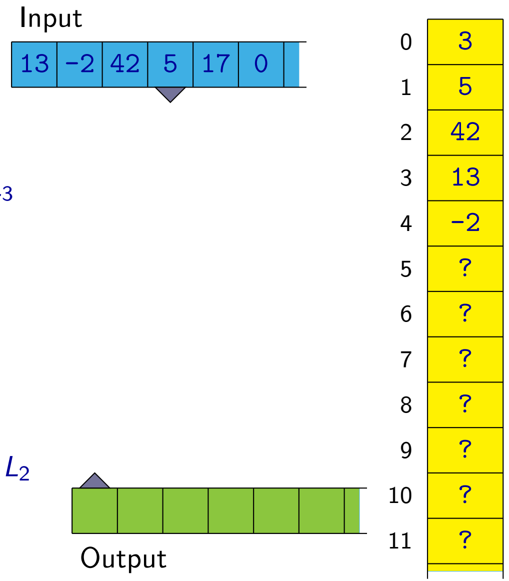
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 := \text{READ}()$
       if $(R_2 = 0)$ goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       $\text{WRITE}(R_2)$
$L_3$: if $(R_1 > R_0)$ goto $L_2$
       halt
```



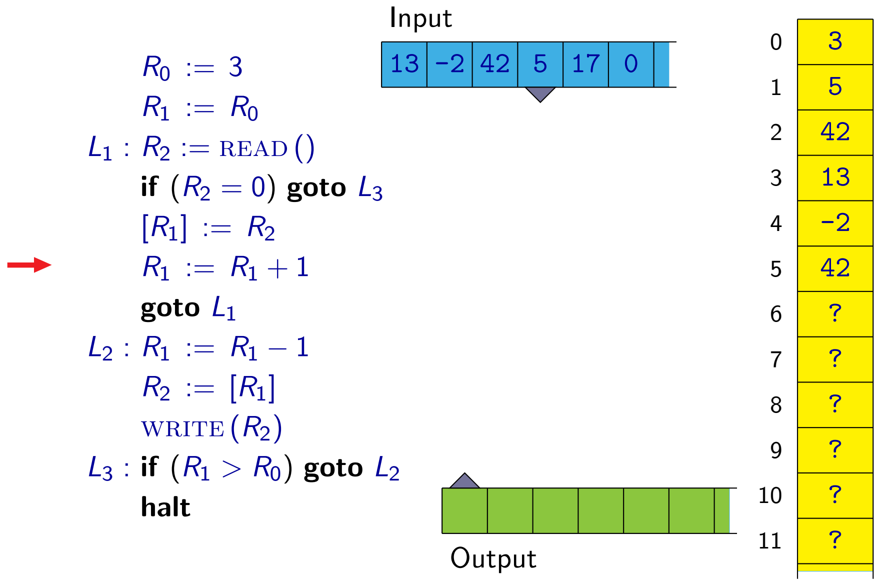
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
→     $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$: $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$: **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



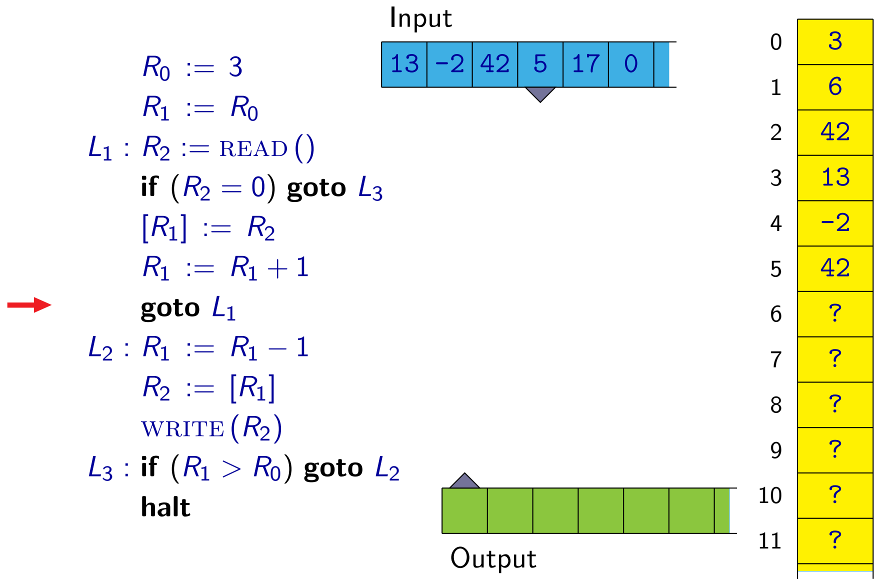
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



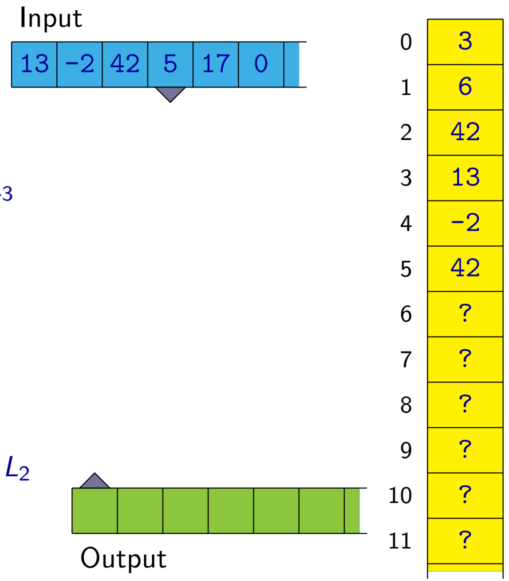
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



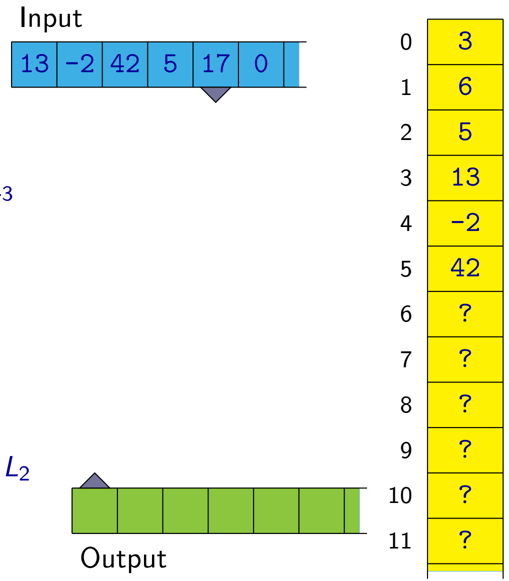
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
→ $L_1$ : $R_2 :=$ READ ()
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
  $L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



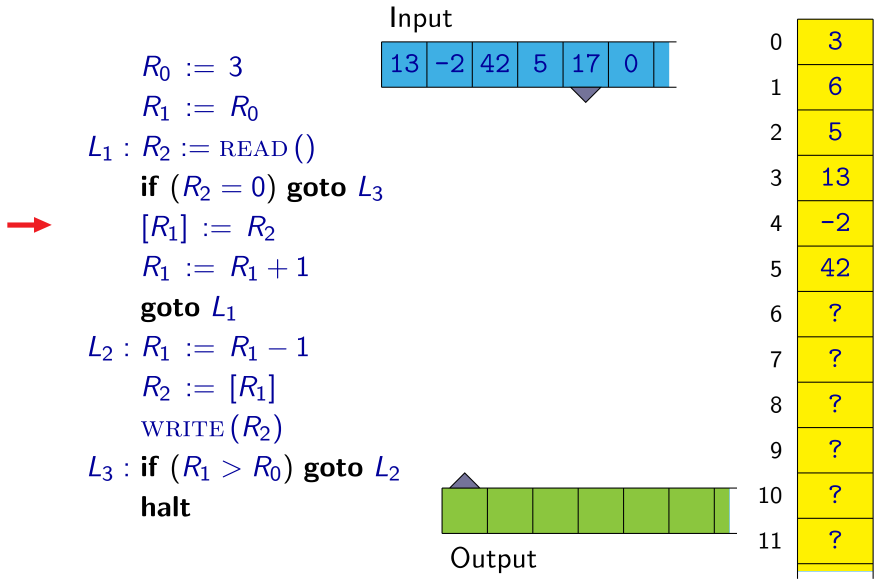
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
$L_1$ :   $R_2 :=$ READ ()
→         **if** $(R_2 = 0)$ **goto** $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          **goto** $L_1$
$L_2$ :   $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE $(R_2)$
$L_3$ :   **if** $(R_1 > R_0)$ **goto** $L_2$
          **halt**
```



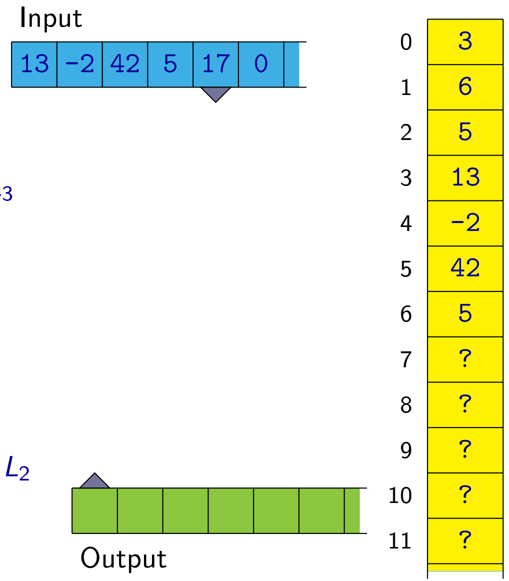
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       **if** $(R_2 = 0)$ **goto** $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       **goto** $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE $(R_2)$
$L_3$: **if** $(R_1 > R_0)$ **goto** $L_2$
       **halt**
```



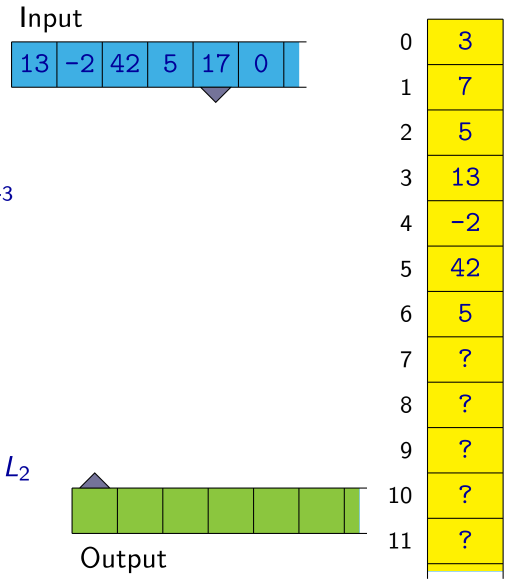
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
$L_1$ :  $R_2 := \text{READ}()$
         **if** $(R_2 = 0)$ **goto** $L_3$
         $[R_1] := R_2$
→        $R_1 := R_1 + 1$
         **goto** $L_1$
$L_2$ :  $R_1 := R_1 - 1$
         $R_2 := [R_1]$
         $\text{WRITE}(R_2)$
$L_3$ :  **if** $(R_1 > R_0)$ **goto** $L_2$
         **halt**
```



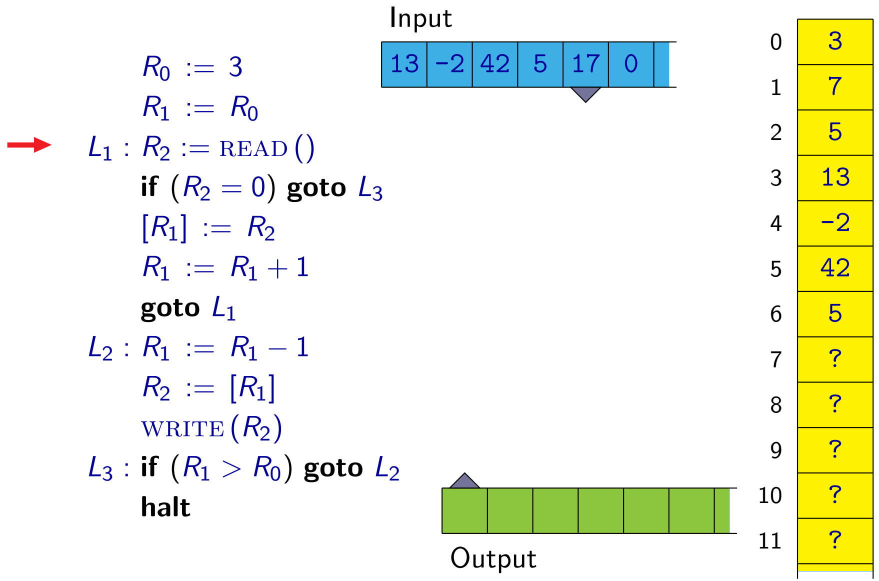
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
→       **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



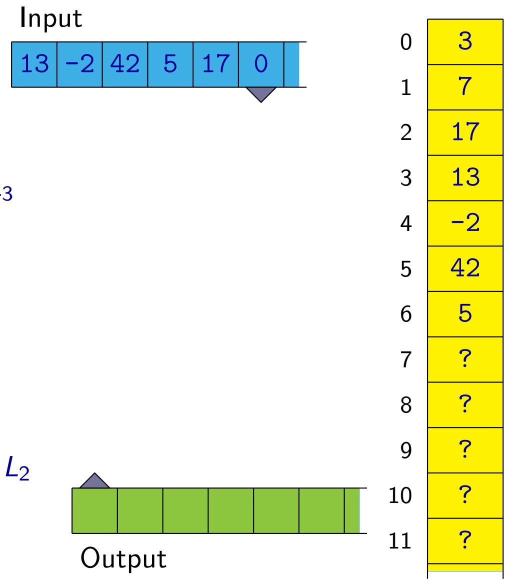
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



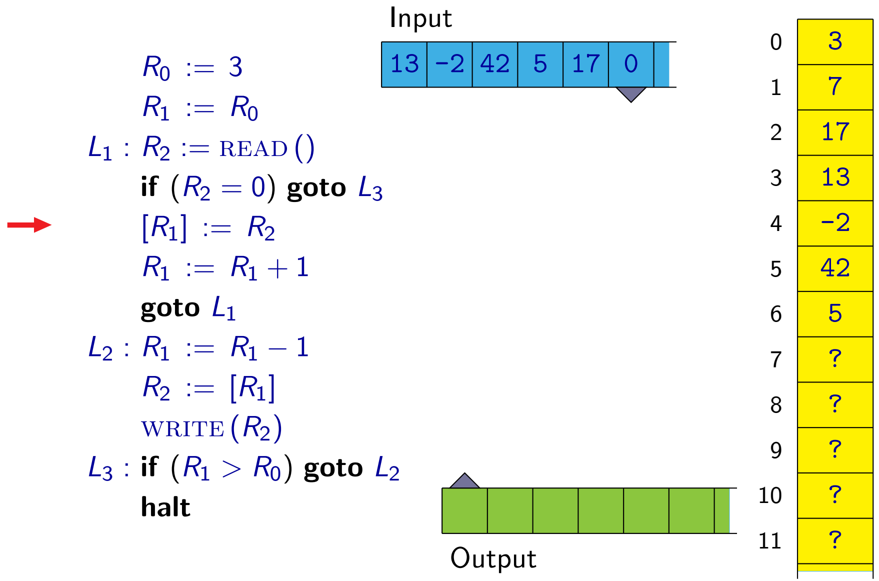
Random Access Machine

```
           $R_0 := 3$
           $R_1 := R_0$
    $L_1$ : $R_2 := \text{READ}()$
→          if $(R_2 = 0)$ goto $L_3$
           $[R_1] := R_2$
           $R_1 := R_1 + 1$
           goto $L_1$
    $L_2$ : $R_1 := R_1 - 1$
           $R_2 := [R_1]$
           $\text{WRITE}(R_2)$
    $L_3$ : if $(R_1 > R_0)$ goto $L_2$
           halt
```



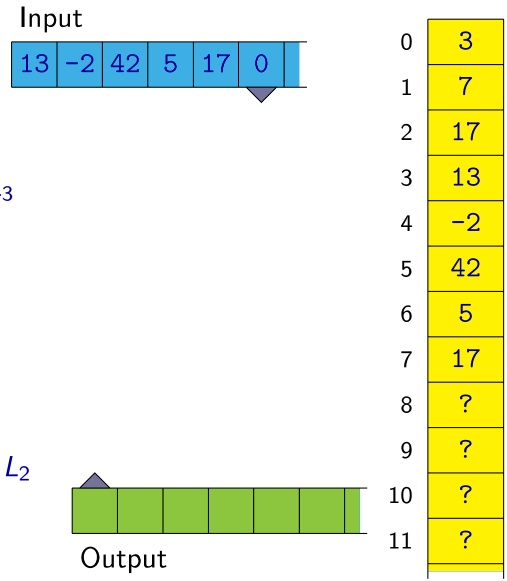
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 := \text{READ}()$
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      $\text{WRITE}(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



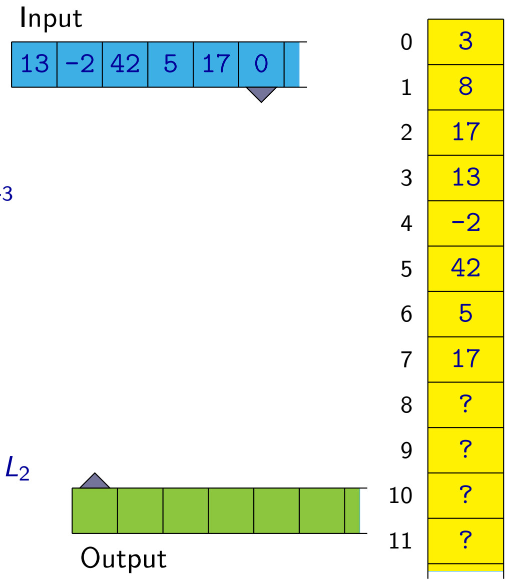
Random Access Machine

```
        R_0 := 3
        R_1 := R_0
L_1 :   R_2 := READ()
        if (R_2 = 0) goto L_3
        [R_1] := R_2
→       R_1 := R_1 + 1
        goto L_1
L_2 :   R_1 := R_1 - 1
        R_2 := [R_1]
        WRITE(R_2)
L_3 :   if (R_1 > R_0) goto L_2
        halt
```



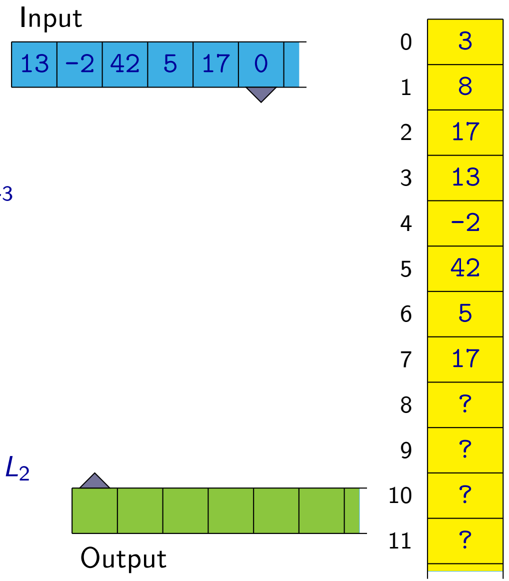
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



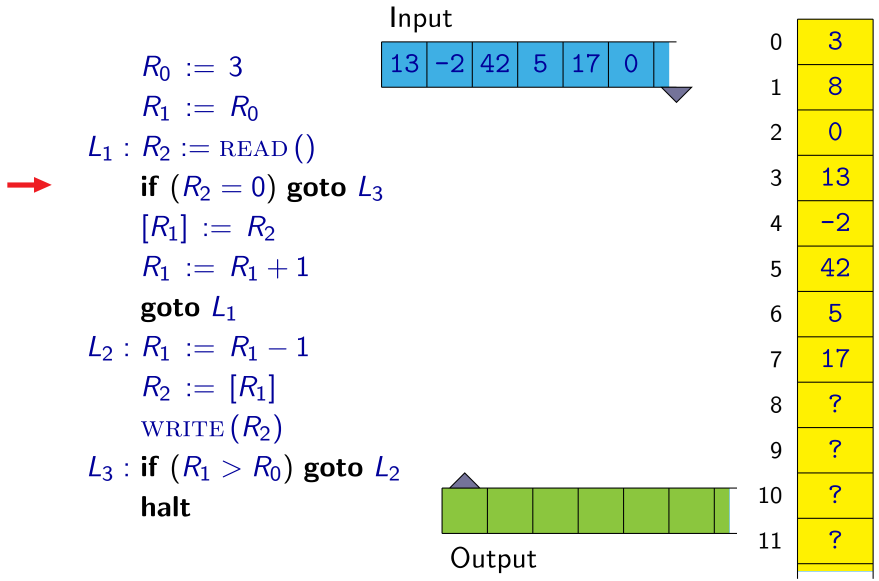
Random Access Machine

```
     $R_0 := 3$
     $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
     **if** $(R_2 = 0)$ **goto** $L_3$
     $[R_1] := R_2$
     $R_1 := R_1 + 1$
     **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
     $R_2 := [R_1]$
     WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
     **halt**
```



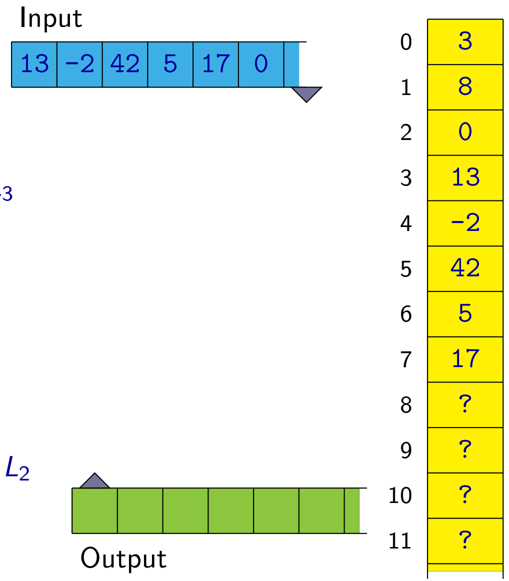
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



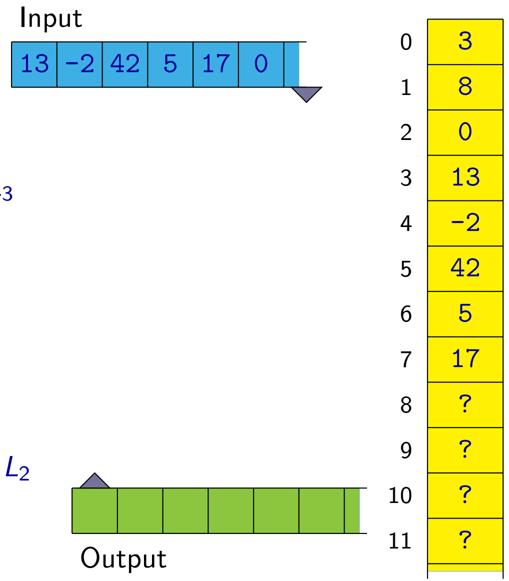
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
→ $L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



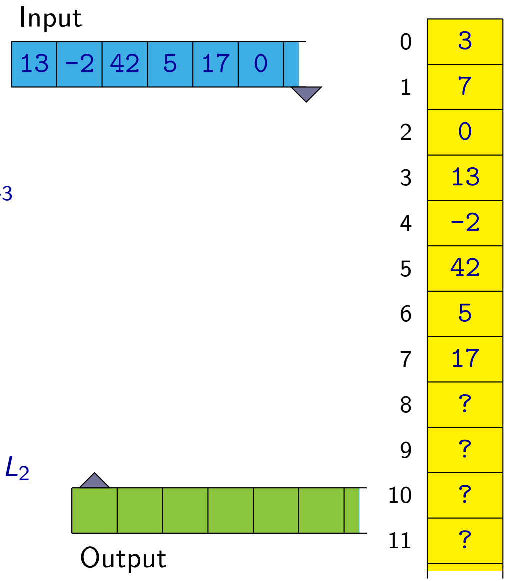
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
    $L_1$ : $R_2 :=$ READ()
          if $(R_2 = 0)$ goto $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          goto $L_1$
→   $L_2$ : $R_1 := R_1 - 1$
          $R_2 := [R_1]$
          WRITE$(R_2)$
    $L_3$ : if $(R_1 > R_0)$ goto $L_2$
          halt
```



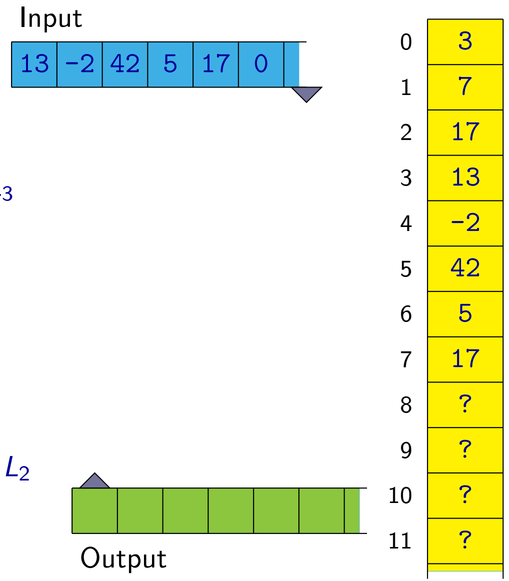
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
→       $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



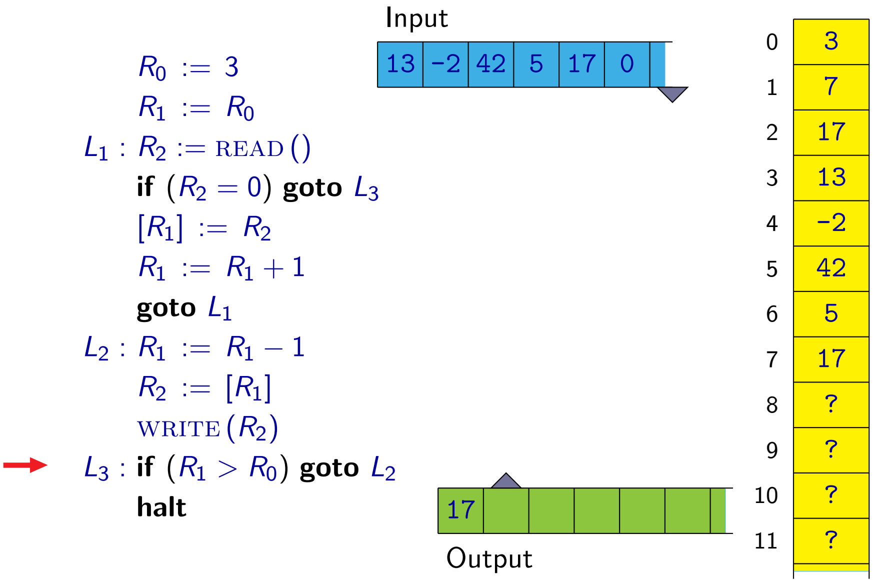
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



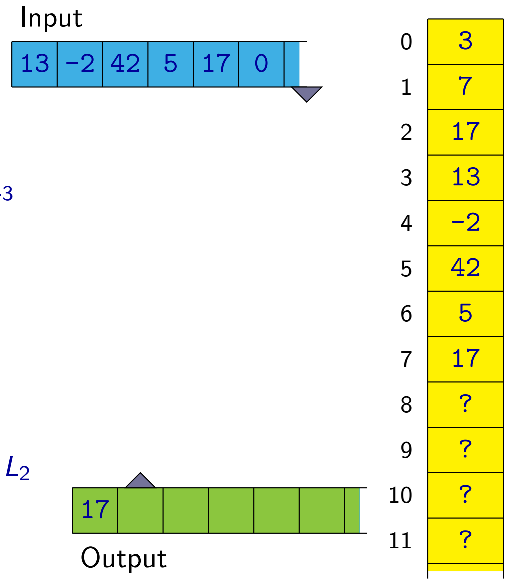
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 := \text{READ}()$
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        $\text{WRITE}(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



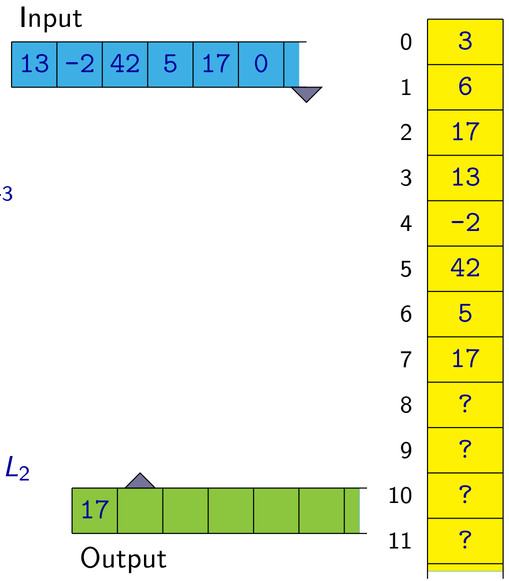
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
       **if** $(R_2 = 0)$ **goto** $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       **goto** $L_1$
→ $L_2$ : $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
       **halt**
```



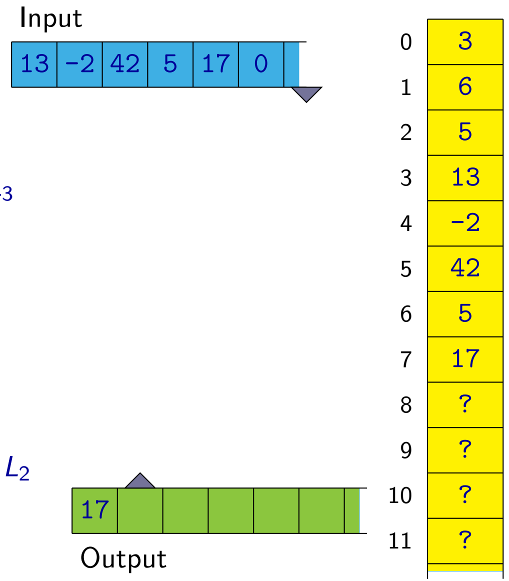
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 := \text{READ}()$
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        $\text{WRITE}(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



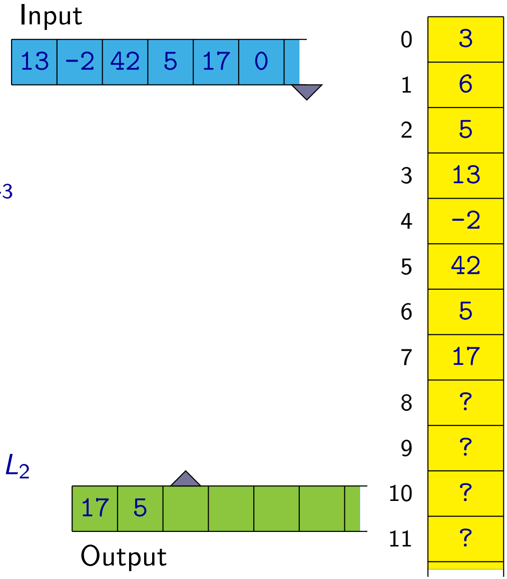
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      if ($R_2 = 0$) goto $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
      halt
```



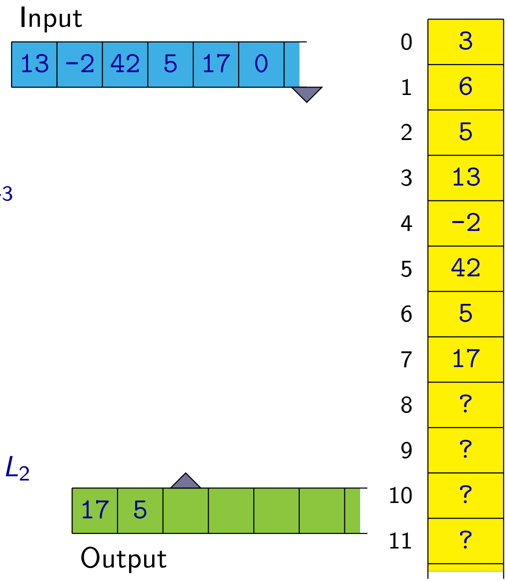
Random Access Machine

```
         $R_0 := 3$
         $R_1 := R_0$
$L_1$ :  $R_2 :=$ READ ()
         **if** $(R_2 = 0)$ **goto** $L_3$
         $[R_1] := R_2$
         $R_1 := R_1 + 1$
         **goto** $L_1$
$L_2$ :  $R_1 := R_1 - 1$
         $R_2 := [R_1]$
         WRITE $(R_2)$
→ $L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
         **halt**
```



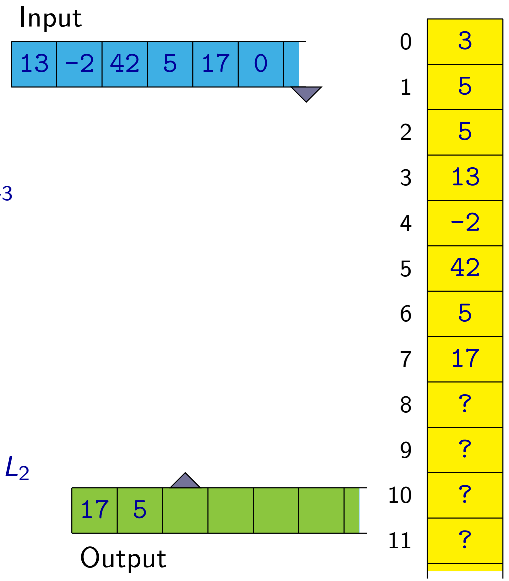
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



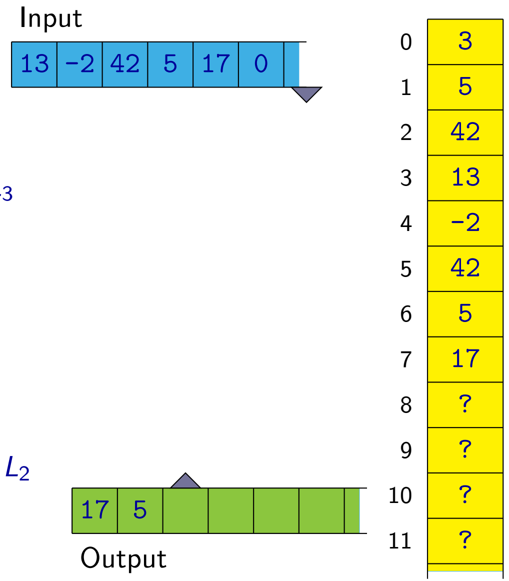
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



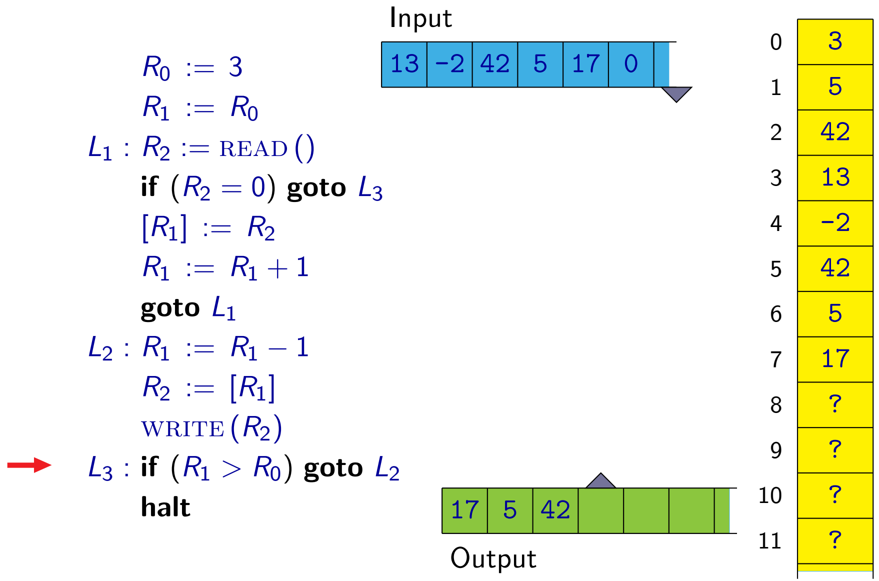
Random Access Machine

```
          $R_0 := 3$
          $R_1 := R_0$
$L_1$ :   $R_2 := \text{READ}()$
          if $(R_2 = 0)$ goto $L_3$
          $[R_1] := R_2$
          $R_1 := R_1 + 1$
          goto $L_1$
$L_2$ :   $R_1 := R_1 - 1$
          $R_2 := [R_1]$
→         $\text{WRITE}(R_2)$
$L_3$ :   if $(R_1 > R_0)$ goto $L_2$
          halt
```



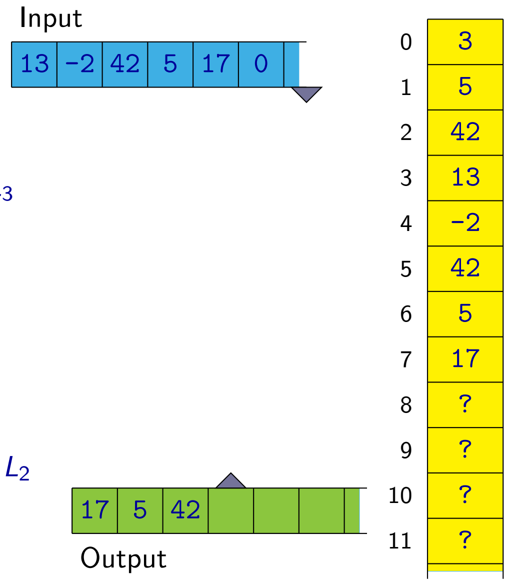
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 := \textsc{read}()$
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        $\textsc{write}(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



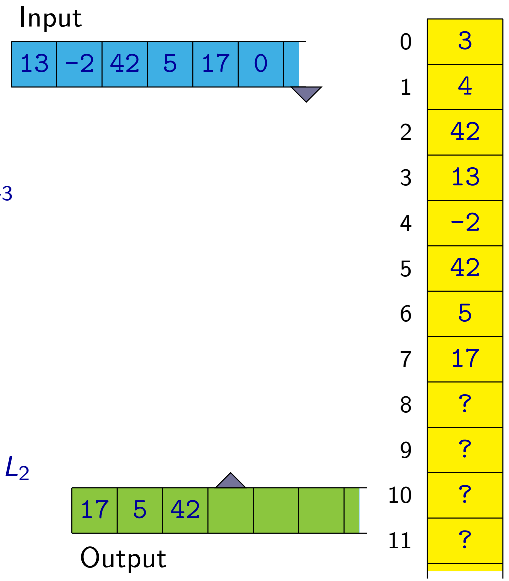
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



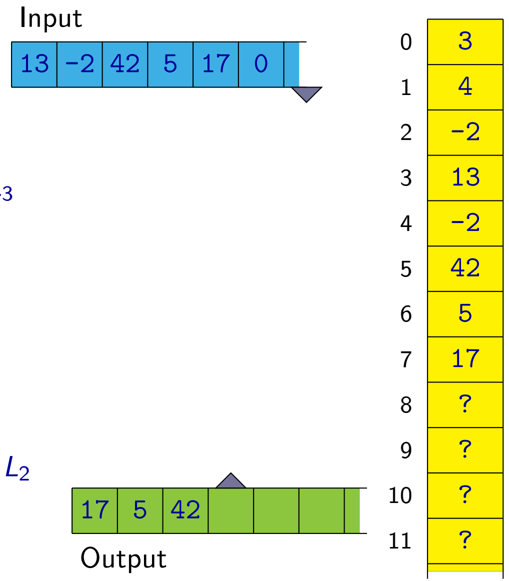
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
  $L_1$ : $R_2 :=$ READ ()
        if ($R_2 = 0$) goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
  $L_2$ : $R_1 := R_1 - 1$
→       $R_2 := [R_1]$
        WRITE ($R_2$)
  $L_3$ : if ($R_1 > R_0$) goto $L_2$
        halt
```



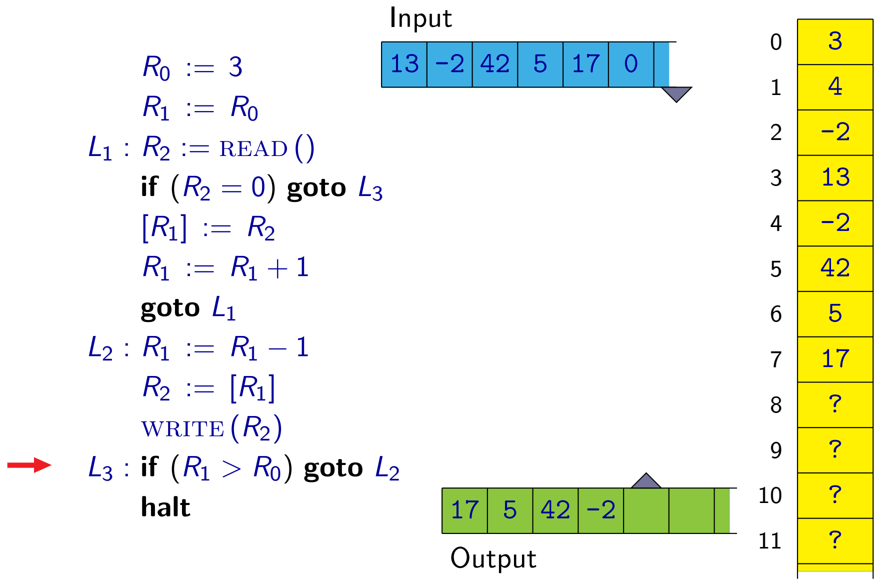
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$: $R_2 :=$ READ ()
       **if** $(R_2 = 0)$ **goto** $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       **goto** $L_1$
$L_2$: $R_1 := R_1 - 1$
       $R_2 := [R_1]$
  →    WRITE $(R_2)$
$L_3$: **if** $(R_1 > R_0)$ **goto** $L_2$
       **halt**
```



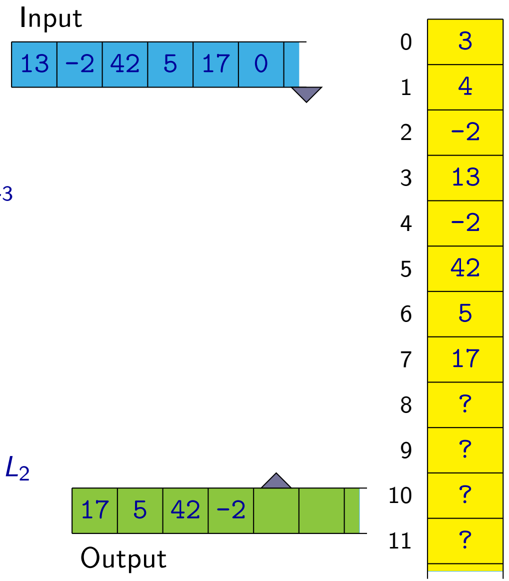
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$: $R_2 :=$ READ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$: $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE($R_2$)
$L_3$: **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



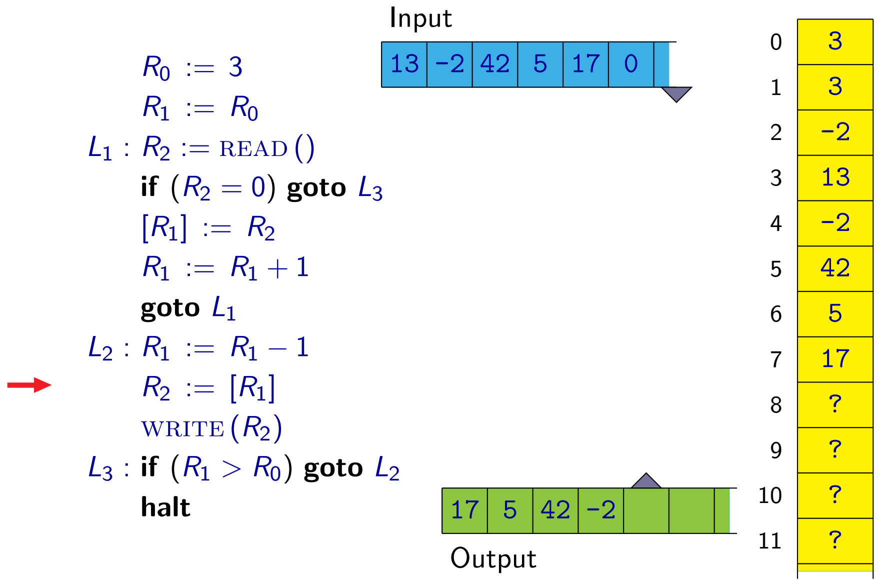
Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      if ($R_2 = 0$) goto $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE ($R_2$)
$L_3$ : if ($R_1 > R_0$) goto $L_2$
      halt
```



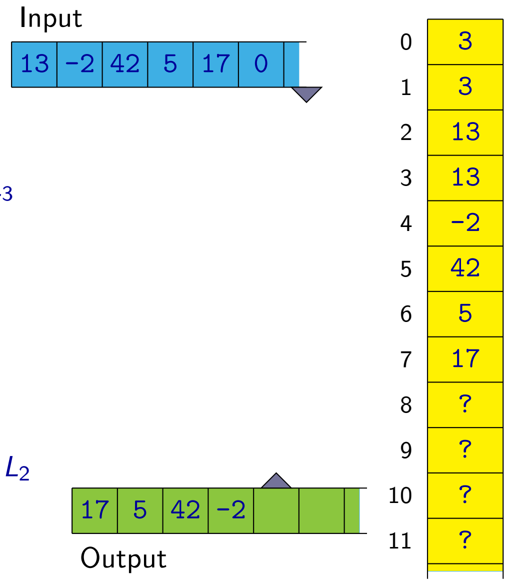
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
        **if** $(R_2 = 0)$ **goto** $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
        WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
        **halt**
```



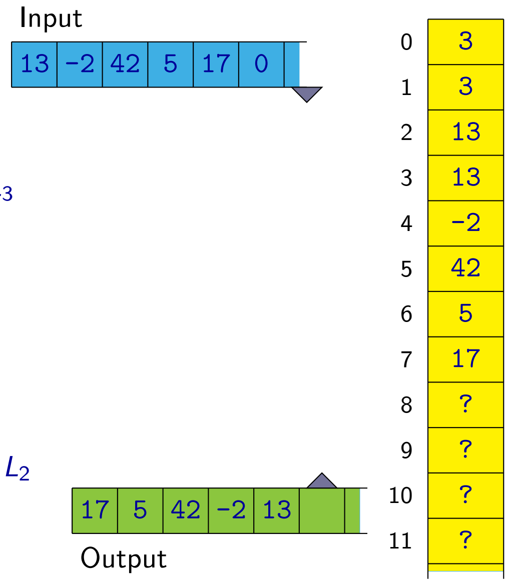
Random Access Machine

```
        $R_0 := 3$
        $R_1 := R_0$
$L_1$ : $R_2 := \text{READ}()$
        if $(R_2 = 0)$ goto $L_3$
        $[R_1] := R_2$
        $R_1 := R_1 + 1$
        goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
        $R_2 := [R_1]$
→       $\text{WRITE}(R_2)$
$L_3$ : if $(R_1 > R_0)$ goto $L_2$
        halt
```



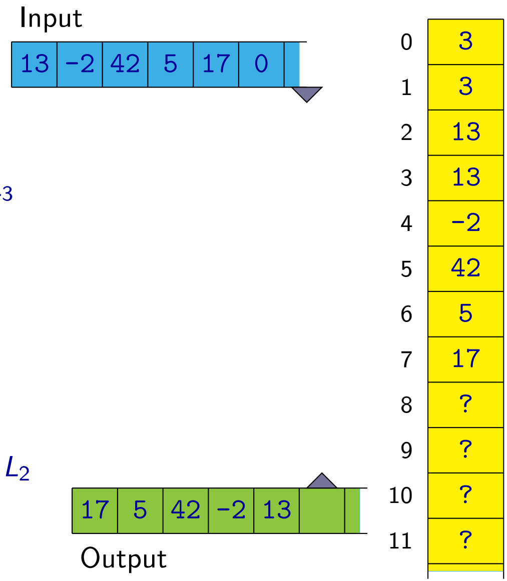
Random Access Machine

```
       $R_0 := 3$
       $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
       if $(R_2 = 0)$ goto $L_3$
       $[R_1] := R_2$
       $R_1 := R_1 + 1$
       goto $L_1$
$L_2$ : $R_1 := R_1 - 1$
       $R_2 := [R_1]$
       WRITE $(R_2)$
→ $L_3$ : if $(R_1 > R_0)$ goto $L_2$
       halt
```



Random Access Machine

```
      $R_0 := 3$
      $R_1 := R_0$
$L_1$ : $R_2 :=$ READ ()
      **if** $(R_2 = 0)$ **goto** $L_3$
      $[R_1] := R_2$
      $R_1 := R_1 + 1$
      **goto** $L_1$
$L_2$ : $R_1 := R_1 - 1$
      $R_2 := [R_1]$
      WRITE $(R_2)$
$L_3$ : **if** $(R_1 > R_0)$ **goto** $L_2$
      **halt**
```



Random Access Machine

Main differences with respect to real computers:

- The size of memory is not limited (an address can be an arbitrary natural number).
- The size of a content of individual memory cells is not limited (a cell can contain an arbitrary integer).
- It reads data sequantially from an input that consists of a sequence of integers. The input is read-only.
- It writes data sequantially on the output that consists of a sequence of integers. The output is write-only.

Random Access Machine

- Operations like an access to a memory cell with an address less than zero or division by zero result in an error — the computation is stuck.
- For an initial content of memory there are basically two possibilities how to define it:
  - All cells are initialized with value 0.
  - Reading a cell, to which nothing has been written, results in an error.

    Cells at the beginning contain a special value (denoted here by symbol '?') that represents that the given cell has not been initialized yet.
- We could consider also variants of RAMs where memory cells (and cells of input and output) do not contain integers (i.e., the elements of set $\mathbb{Z}$) but they can contain only natural numbers (i.e., elements of set $\mathbb{N}$).

  For example, operation of subtraction ($R_i := R_j - R_k$) then behaves in such a way that whenever the result should be a negative number, then value 0 is assigned as the result.

Random Access Machine

- Different variants of RAMs can differ in what particular operations can be used in arithmetic instructions.

  For example:

  - a support of bitwise operations (and, or, not, xor, ...), bit shifts, ...
  - a variant of RAM that does not have operations for multiplication and division

- We could also consider a variant of RAM where instead of instructions of the form

  $$\textbf{if } (R_i \ \mathit{rel}\ R_j) \textbf{ goto } \ell \qquad \text{nebo} \qquad \textbf{if } (R_i \ \mathit{rel}\ c) \textbf{ goto } \ell$$

  all conditional jumps are of the form

  $$\textbf{if } (R_i \ \mathit{rel}\ 0) \textbf{ goto } \ell$$

  Instead of all relations $\{=, \neq, \leq, \geq, <, >\}$, only a subset of them can be supported, e.g., $\{=, >\}$.

Random Access Machine

- In some variants of RAM, the input and output are not in a form of sequence of numbers.

  Instead, such machine could work with input and output tapes containg sequences of symbols from some alphabet, e.g., $\{0, 1\}$.

  This machine then could have for example some instructions that allow the branch the computation according to a symbol read from the input.

  However, the internal memory even in this variant works with numbers.

- When a machine should produce an answer of the form Yes/No (i.e., to accept or reject the given input), it does not need to have an output tape.

  Instruction **halt** is then replaced with instructions **accept** and **reject**.

Random Access Machine

- In the standard definition of RAM, jump instructions jumping to an adress stored in some memory cell are usualy not considered:

  $$\textbf{goto } R_i$$

  RAM could be extended with these instructions.

- For RAMs, a code of a program is usually stored in a separate read-only memory, not in a working memory.

  So the code can not be modified during a computation.

Random Access Machine

- A type of a machine, similar to RAM, but where its program is stored in its working memory (instructions are encoded by numbers) and so it can be modified during a computations, is called **RASP** (**random-access stored program**).

  RASP can simulate behaviour of self-modifying programs.

Random Access Machine

A running time of a RAM can be computed in two different ways:

- **uniform cost** — the number of executed instructions
- **logarithmic cost** — the sum of cost of individual instructions; the cost of one instruction depends on the number of bits of values used in the given instruction.

  For example:

  - The cost of execution of instructions for addition and subtruction is the sum of the number of bits of their operands.
  - The cost of execution of instructions for multiplication and division is the product of the number of bits of their operands.
  - The cost of instructions accessing memory (load, store) is the sum of the number of bits of an address and the number of bits of a number that is read or written.

  Remark: When counting the number of bits of a given number, it is assumed that value 0 has 1 bit.

Random Access Machine

Also the amount of memory used during a computation by a RAM can be computed in two different ways:

- **uniform cost** — the number of memory cells used, i.e., the number of cells, which were read or written to during the computation.
- **logarithmic cost** — the maximal number of bits of memory that were used during the computation.

  The number of bits includes both the number of bits of used cells and the number of bits of their addresses.

Random Access Machine

The uniform cost realistically represents the amount of a work done during the execution and the amount of used memory only in those cases where the values stored in memory cells are “small”, i.e., if in a real implemantation for “reasonably” big inputs it would be possible to represents them for example as 32-bit or 64-bit numbers.

Random Access Machine

- For those machines that do not have an instruction for multiplication, it can be easily shown that each instruction can produce a number that has at most one bit more than the bigger (in absolute value) of its operands.

  For such machines, after t steps of computation, each cell contains a number that has at most $t + m + n$ bits where m is the number of bits of the biggist constant occurring in the program and n is the biggest number of bits of a number in the input.

- If the machine has an instruction for the multiplication then after t steps, some memory cell can contain a number that has approximately 2^t bits.

Random Access Machine and Turing Machine

Any program in any programming language can be implemented as a program for a RAM.

It is not difficult (although a little bit tedious) to realize that every algorithm performed by a RAM can also be implemented by a Turing machine.

A Turing machine can implement any algorithm that can be written as a program in an arbitrary programming language.

Remark: And of course, it is possible to simulate a behaviour of a Turing machine by a RAM.

Turing Machine Simulating RAM

In the description of how a Turing machine can simulate a RAM, it is simpler to proceed by smaller steps:

- We will show how to simulate a varint of RAM described before by a variant of RAM with somewhat simpler instructions.
- We will show how to simulate the behaviour of this simpler variant of RAM by a multitape Turing machine.
- We have already seen before how a multitape Turing machine can be simulated by one-tape Turing machine.

A simpler variant of RAM

This simpler variant of RAM has, in addition to its working memory, also three **registers**:

- **register A** — almost all instructions work with this register, results of all operations are stored into this register

  Remark: This kind of register is often called an **accumulator**.

- **register B** — this register is used to store the second operand of arithmetic instructions (the first operand is always in the accumulator)

- **register C** — this register is used to store an address of a memory cell, to which a value is written by a store operation

A simpler variant of RAM

Overview of instructions:

$A := c$	– assinment of a constant
$B := A$	– assinment to register B
$C := A$	– assinment to register C
$A := [A]$	– load (reading from memory)
$[C] := A$	– store (writing to memory)
$A := A\ op\ B$	– arithmetic instructions, $op \in \{+, -, *, /\}$
if $(A\ rel\ 0)$ **goto** ℓ	– conditional jump, $rel \in \{=, \neq, \leq, \geq, <, >\}$
goto ℓ	– unconditional jump
$A := \text{READ}()$	– reading from input
$\text{WRITE}(A)$	– writing to output
halt	– program termination

A simpler variant of RAM

For example, instruction

$$R_5 := 42$$

can be replaced with a sequence of instructions:

$$\begin{aligned} A &:= 5 \\ C &:= A \\ A &:= 42 \\ [C] &:= A \end{aligned}$$

A simpler variant of RAM

For example, instruction

$$R_{12} := R_3$$

can be replaced with a sequence of instructions:

$$\begin{aligned} A &:= 12 \\ C &:= A \\ A &:= 3 \\ A &:= [A] \\ [C] &:= A \end{aligned}$$

A simpler variant of RAM

For example, instruction

$$R_8 := [R_2]$$

can be replaced with a sequence of instructions:

$$\begin{aligned} A &:= 8 \\ C &:= A \\ A &:= 2 \\ A &:= [A] \\ A &:= [A] \\ [C] &:= A \end{aligned}$$

A simpler variant of RAM

For example, instruction

$$[R_{15}] := R_9$$

can be replaced with a sequence of instructions:

$$
\begin{aligned}
A &:= 15 \\
A &:= [A] \\
C &:= A \\
A &:= 9 \\
A &:= [A] \\
[C] &:= A
\end{aligned}
$$

A simpler variant of RAM

For example, instruction

$$R_7 := R_3 + R_6$$

can be replaced with a sequence of instructions:

$$\begin{aligned}
A &:= 7\\
C &:= A\\
A &:= 6\\
A &:= [A]\\
B &:= A\\
A &:= 3\\
A &:= [A]\\
A &:= A + B\\
[C] &:= A
\end{aligned}$$

A simpler variant of RAM

For example, instruction

$$\textbf{if } (R_4 \geq R_{11}) \textbf{ goto } \ell$$

can be replaced with a sequence of instructions:

$$\begin{aligned} A &:= 11 \\ A &:= [A] \\ B &:= A \\ A &:= 4 \\ A &:= [A] \\ A &:= A - B \\ &\textbf{if } (A \geq 0) \textbf{ goto } \ell \end{aligned}$$

A simpler variant of RAM

For example, instruction

$$R_{23} := \textsc{read}()$$

can be replaced with a sequence of instructions:

$$\begin{aligned} A &:= 23 \\ C &:= A \\ A &:= \textsc{read}() \\ [C] &:= A \end{aligned}$$

A simpler variant of RAM

For example, instruction

$$\text{WRITE}\,(R_{17})$$

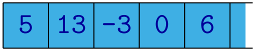
can be replaced with a sequence of instructions:

$$\begin{aligned} &A := 17 \\ &A := [A] \\ &\text{WRITE}\,(A) \end{aligned}$$

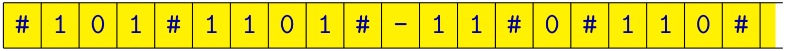
Turing Machine Simulating RAM

A Turing machine works with words over some alphabet, while a RAM works with numbers. But numbers can be written as sequences of symbols and conversely symbols of an alphabet can be written as numbers.

For example the following input of a RAM



can be represented for a Turing machine as



Turing Machine Simulating RAM

A Turing machine simulating a computation of a RAM has several tapes:

- A tape containing a content of the working memory of the RAM.
- Three tapes containing values of registers A, B, and C.

  (Values of registers A, B, and C will be written on these tapes in binary and delimited from the left and from the right by symbols #.)
- A tape representing the input tape of the RAM.
- A tape representing the output tape of the RAM.
- One auxiliary tape used for an implementation of the simulation of some instructions.

Turing Machine Simulating RAM

The Turing machine stores the information about the instruction of the RAM that is currently executed in its control unit.

Execution of most of instructions is not difficult:

- $A := c$

  it writes bits of the constant c to the tape of register A

- $B := A$ or $C := A$

  it will copy a content of the tape of register A to the tape of register B or C

- **goto** ℓ

  just changes the state of the control unit of the Turing machine

- **if** $(A\ rel\ 0)$ **goto** ℓ, kde $rel \in \{=, \neq, \leq, \geq, <, >\}$

  the content of the working register is tested and the state of the control unit is changed accordingly

Turing Machine Simulating RAM

- $A := \textsc{read}()$

  copy the value (marked at the ends by symbols "#") from the input tape to the tape of register A

- $\textsc{write}(A)$

  copy the value of register A to the output tape.

- **halt**

  the computation halts

Turing Machine Simulating RAM

Also arithmetic instructions are rather easy to implement, although the a little bit more complicated than the previous instructions:

- $A := A \ op \ B$, where $op \in \{+, -, *, /\}$
  The Turing machine performs the given operation (such as addition or subtraction) bit by bit, the result is stored to register A.

Remark: Multiplication and division can be done as a sequence of additions and bit shifts.

In the implementation of addition and division, it may be necessary to use an auxiliary tape to store intermediate results.

Turing Machine Simulating RAM

Probably the most complex is the implementation of the RAM memory.

One possibility is to store only values of those cells that were actually used so far in the computation of the RAM (we know that all other cells contain value 0).

Example: The RAM worked so far only with cells 2, 3 and 6:

- Cell 2 contains value 11.
- Cell 3 contains value -1.
- Cell 6 contains value 2.

The content of the tape of the Turing machine representing the content of the memory of the RAM will be as follows:

$	#	1	0	:	1	0	1	1	#	1	1	:	-	1	#	1	1	0	:	1	0	#	$	

Turing Machine Simulating RAM

Load instruction, i.e., $A := [A]$:

- The Turing machine will search the given address, stored in register A, on the tape containg the content of the memory of the RAM. (If it does not find it, it will appened it at the end with value 0.)
- The given value in the cell is copied to the tape of register A.

Turing Machine Simulating RAM

Store instruction, i.e., $[C] := A$:

- Similarly as before, the Turing machine will find the position of the tape representing a content of the memory, where the value in the given address, stored in register C, occurs.
- The rest of the memory tape is copied to an auxiliary tape.
- The content of the tape of register A is copied to the corresponding place.
- The rest of the tape, copied on the auxiliary tape, is copied back to the memory tape (after the newly written value).

Turing Machine Simulating RAM

It is not hard to see that in the simulation of a RAM by a Turing machine described above, the number of steps performed by the Turing machine is polynomial (quadratic) with respect to the running time of the RAM computed using logarithmic cost.

Remark: For those RAMs that do not have an instruction for multiplication, the number of steps performed by the simulating Turing machine is polynomial also with respect to the running time of the RAM computed using uniform cost.

It the RAM executes t instruction, the Turing machine that simulates it executes approximately $O(t^3)$ instructions.