

Výpočetní modely

Algoritmy jsou vykonávány stroji — může to být například:

- skutečný počítač — vykonává instrukce strojového kódu
- virtuální stroj — vykonává instrukce bytekódu
- nějaký idealizovaný matematický model počítače
- ...

Stroj může být:

- jednoúčelový — vykonává jen jeden algoritmus
- obecnější — algoritmus dostává ve formě **programu**

Stroj pracuje po **krocích**.

Algoritmus během výpočtu zpracovává konkrétní **vstup** a produkuje příslušný **výstup**.

Výpočetní model — nějaký idealizovaný matematický model počítače

- abstrahujeme od různých nepodstatných implementačních detailů
- chceme analyzovat ty vlastnosti algoritmů, které pokud možno co nejméně závisí na detailech stroje, který bude daný algoritmus vykonávat

Příklady některých výpočetních modelů:

- konečné automaty
- zásobníkové automaty
- Turingovy stroje
- stroje RAM
- ...

Během výpočtu si stroj typicky musí pamatovat:

- která instrukce se právě provádí
- obsah své pracovní paměti

Podle typu stroje je určeno:

- s jakým typem dat stroj pracuje
- jak jsou tato data v paměti organizována
- jaké operace s těmito daty může stroj vykonávat

Podle typu algoritmu a typu analýzy, kterou chceme provádět, se můžeme rozhodnout, zda má smysl mezi obsah paměti zahrnout i místa

- odkud se čtou vstupní data
- kam se zapisují výstupní data

Jedním z využití výpočetních modelů je to, že mohou sloužit pro přesné definování pojmů, důležitých pro stanovení **výpočetní složitosti** daného algoritmu:

- **doby výpočtu** daného algoritmu \mathcal{A} pro daný vstup w
(pozn.: většinou je to počet kroků vykonaných strojem během výpočtu)
- **množství použité paměti** během tohoto výpočtu

Obecně je pro různé výpočetní modely také důležité

- zda je daný typ stroje schopen **simulovat** výpočty nějakého jiného typu stroje
- jak se při této simulaci liší doba výpočtu či množství použité paměti oproti původnímu stroji

Vysvětlení toho, co to znamená, že stroj \mathcal{M} je **simulován** strojem \mathcal{M}' :

- Výpočet stroje \mathcal{M} pro vstup w je (konečná nebo nekonečná) posloupnost konfigurací stroje \mathcal{M}

$$\alpha_0 \longrightarrow \alpha_1 \longrightarrow \alpha_2 \longrightarrow \dots$$

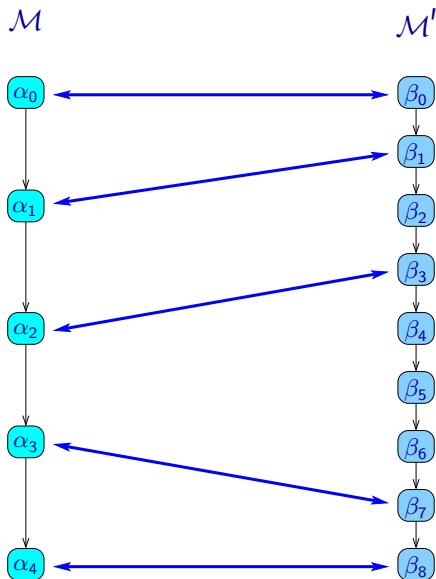
- Tomuto výpočtu odpovídá výpočet stroje \mathcal{M}' tvořený konfiguracemi

$$\beta_0 \longrightarrow \beta_1 \longrightarrow \beta_2 \longrightarrow \dots$$

kde každé konfiguraci α_i odpovídá nějaká konfigurace $\beta_{f(i)}$, kde $f : \mathbb{N} \rightarrow \mathbb{N}$ je funkce, pro kterou platí $f(i) \leq f(j)$ pro každé i a j , kde $i < j$.

- Existuje relace mezi vzájemně si odpovídajícími konfiguracemi stroje \mathcal{M} a jim odpovídajícími konfiguracemi stroje \mathcal{M}' .
- Existují funkce mapující vstup w na odpovídající počáteční konfigurace α_0 a β_0 a analogicky funkce mapující koncové konfigurace na výsledek výpočtu.

Simulace výpočtu



Některé výpočetní modely jsou slabší (konečné automaty, zásobníkové automaty, ...) a není pomocí nich možné implementovat libovolný algoritmus.

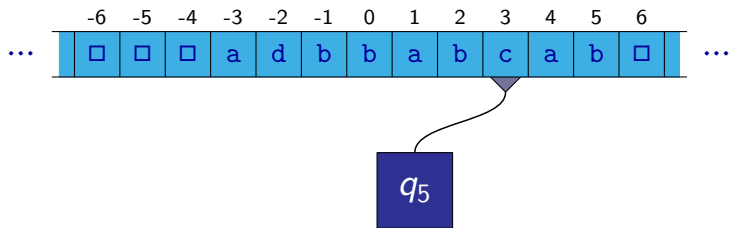
My se teď zaměříme na výpočetní modely, které jsou dostatečně silné na to, aby byly schopny vykonávat libovolný algoritmus (např. takový, jaký je možné zapsat jako program v nějakém programovacím jazyce).

Takovým výpočetním modelům se říká **Turingovsky úplné**:

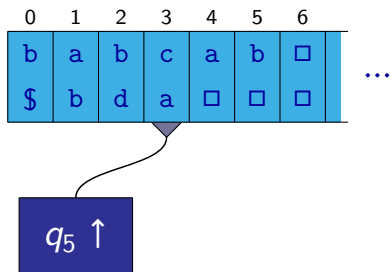
- samy jsou schopny simulovat činnost libovolného Turingova stroje
- jejich činnost může být simulována Turingovým strojem

Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

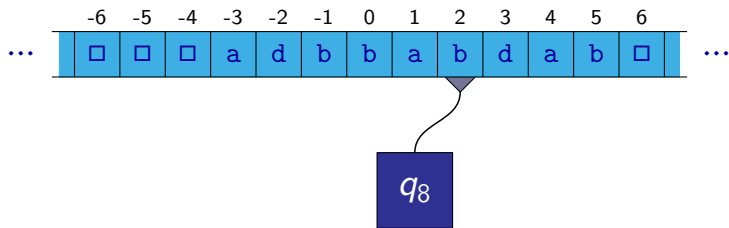


Jednostranně nekonečná páska:

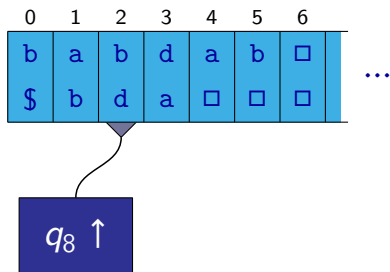


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

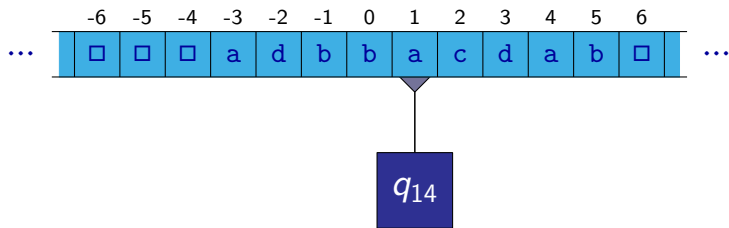


Jednostranně nekonečná páska:

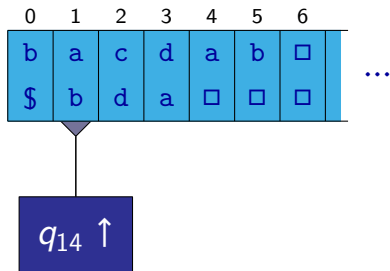


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

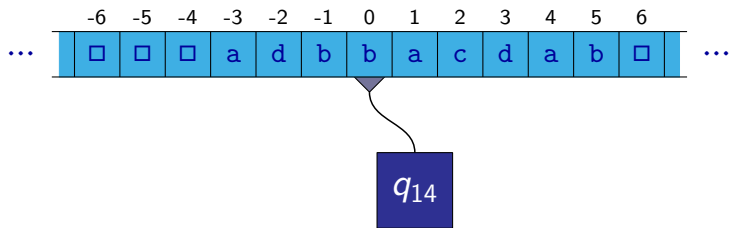


Jednostranně nekonečná páska:

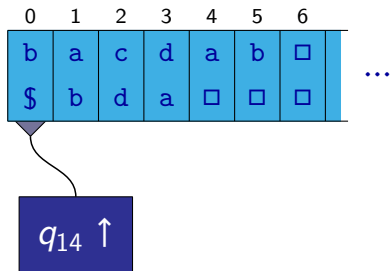


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

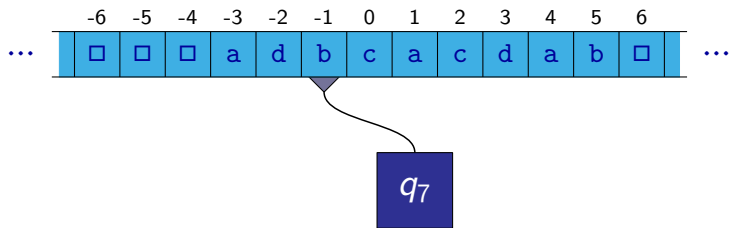


Jednostranně nekonečná páska:

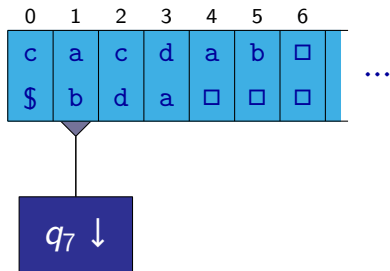


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

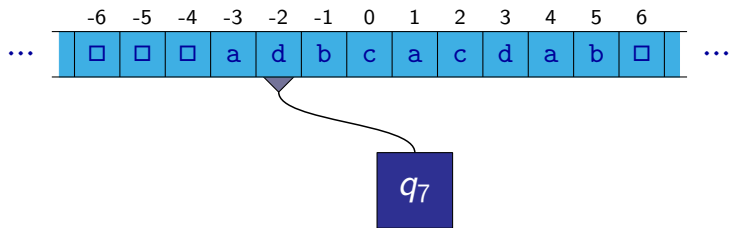


Jednostranně nekonečná páska:

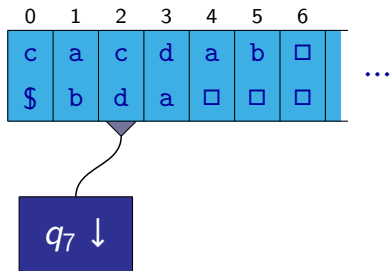


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

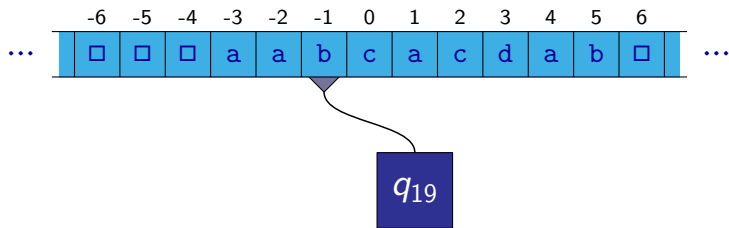


Jednostranně nekonečná páska:

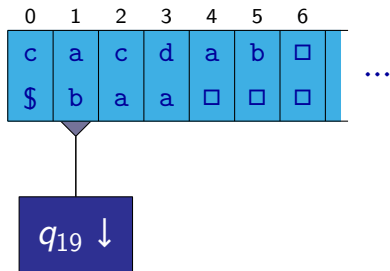


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

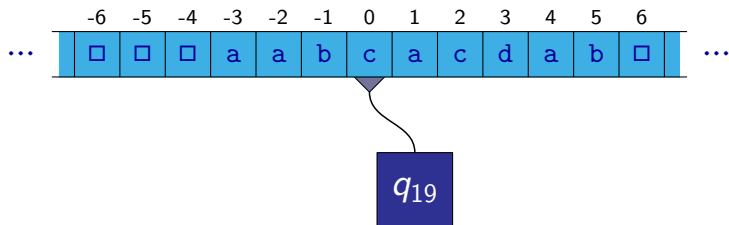


Jednostranně nekonečná páska:

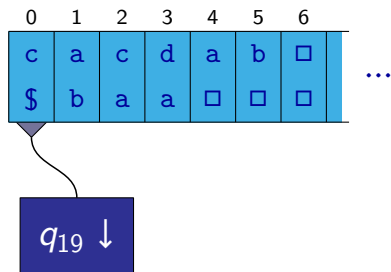


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

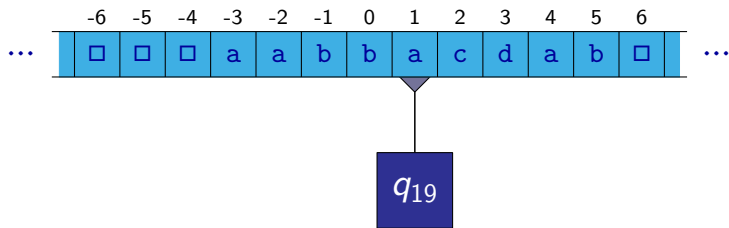


Jednostranně nekonečná páska:

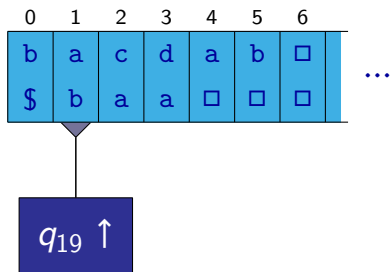


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:

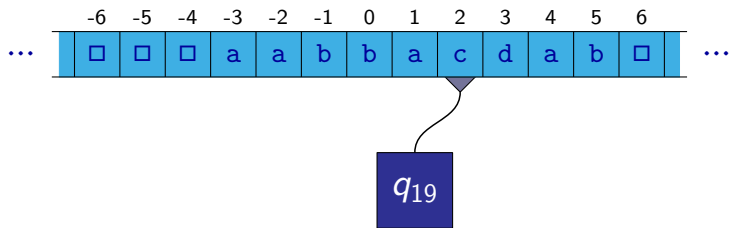


Jednostranně nekonečná páska:

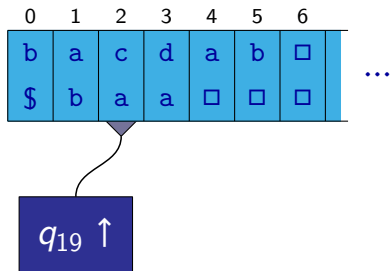


Oboustranně nekonečná páska pomocí jednostranné

Oboustranně nekonečná páska:



Jednostranně nekonečná páska:



Abeceda $\{0, 1\}$

Činnost stroje s libovolnou páskovou abecedou Γ může být simulována strojem s páskovou abecedou $\{0, 1\}$.

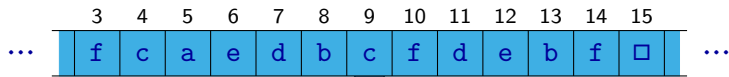
Stačí zvolit nějaké vhodné kódování symbolů abecedy Γ pomocí k -bitových sekvencí.

Příklad: Pásková abeceda $\Gamma = \{\square, a, b, c, d, e, f, g\}$

\square	\leftrightarrow	000
a	\leftrightarrow	001
b	\leftrightarrow	010
c	\leftrightarrow	011
d	\leftrightarrow	100
e	\leftrightarrow	101
f	\leftrightarrow	110
g	\leftrightarrow	111

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

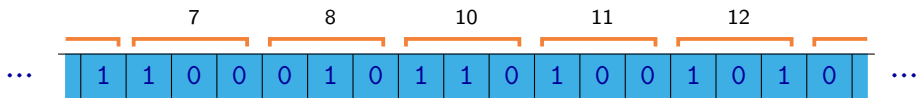


q_7

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

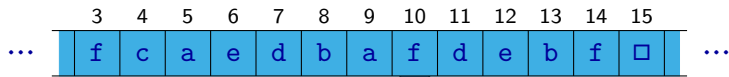
Stroj s abecedou $\{0, 1\}$:



q_7 011

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

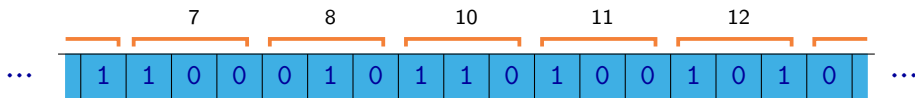


q_{12}

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

Stroj s abecedou $\{0, 1\}$:

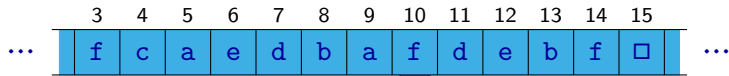


q_{12}

001; ϵ
right

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

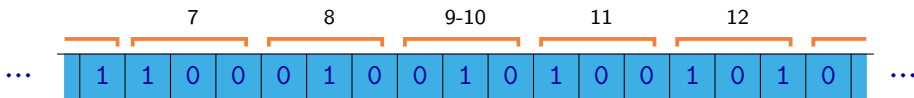


q_{12}

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

Stroj s abecedou $\{0, 1\}$:

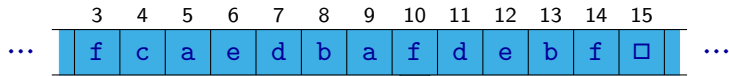


q_{12}

01; 1
right

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

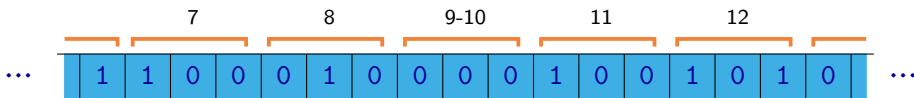


q_{12}

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

Stroj s abecedou $\{0, 1\}$:

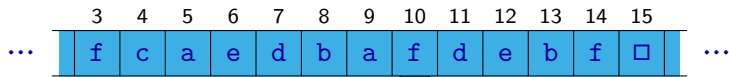


q_{12}

1; 11
right

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

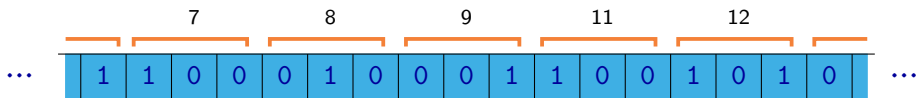


q_{12}

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

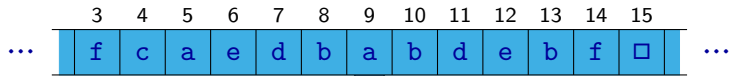
Stroj s abecedou $\{0, 1\}$:



q_{12} 110

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

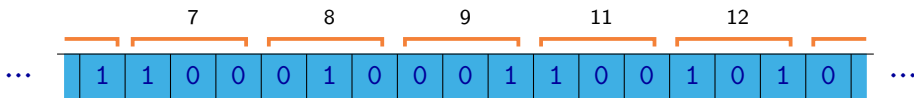


q_5

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

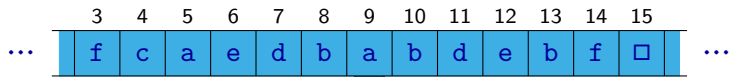
Stroj s abecedou $\{0, 1\}$:



q_5 $\epsilon; 010$
left

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

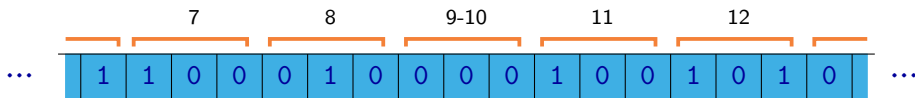


q_5

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

Stroj s abecedou $\{0, 1\}$:

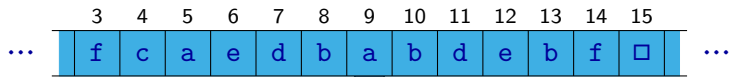


q_5

1; 01
left

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

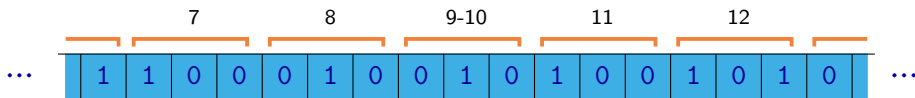


q_5

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

Stroj s abecedou $\{0, 1\}$:

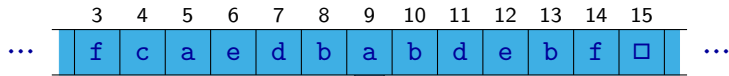


q_5

01; 0
left

Abeceda $\{0, 1\}$

Stroj s páskovou abecedou Γ :

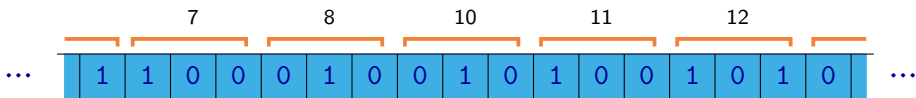


q_5

$$\delta(q_7, c) = (q_{12}, a, +1)$$

$$\delta(q_{12}, f) = (q_5, b, -1)$$

Stroj s abecedou $\{0, 1\}$:



q_5 001

Při výše uvedené simulaci je jeden krok původního stroje simulován $k + 1$ kroky, kde k je počet bitů kódující jeden symbol abecedy Γ .

Pokud tedy původní stroj provede během výpočtu t kroků, simulující stroj provede $O(t)$ kroků.

Poznámka: Tak, jako je možné zmenšit páskovou abecedu na pouhé dva symboly za cenu nárůstu velikosti počtu stavů řídicí jednotky, je rovněž možné snížit počet stavů řídicí jednotky:

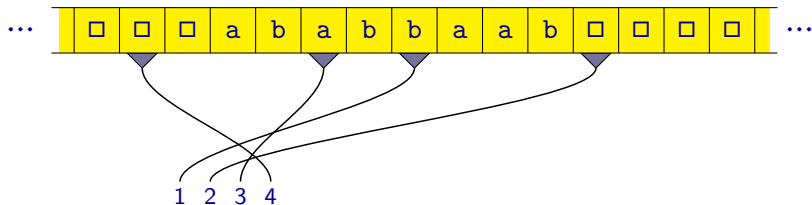
- Činnost libovolného Turingova stroje je možné simulovat Turingovým strojem, který má pouze dva nekonečné stavy řídicí jednotky (a případně nějaké konečné stavy), ovšem za cenu nárůstu velikosti páskové abecedy.

Podobně jako v předchozím případě je jeden krok původního stroje simulován s kroky, kde s je konstanta závisící pouze na počtu stavů řídicí jednotky původního stroje (tj. na velikosti množiny Q).

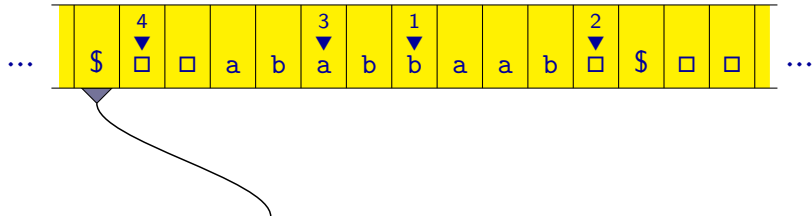
Opět zde tedy platí, že pokud původní stroj provede během výpočtu t kroků, simulující stroj provede $O(t)$ kroků.

Simulace více hlav na pásce pomocí jedné

Více hlav na pásce:

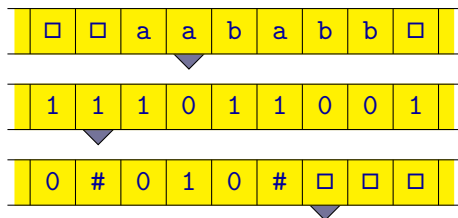


Páska s jednou hlavou:

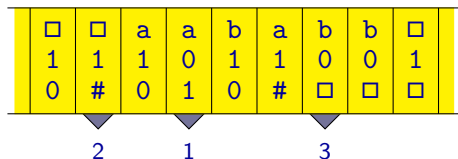


Simulace více pásek pomocí jedné

Více pásek:

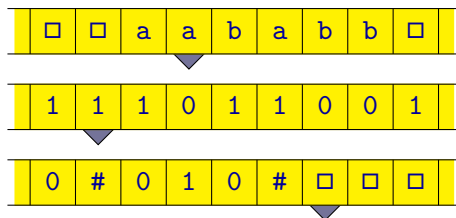


Jedna páska s více hlavami:

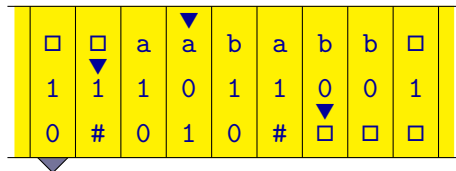


Simulace více pásek pomocí jedné

Více pásek:

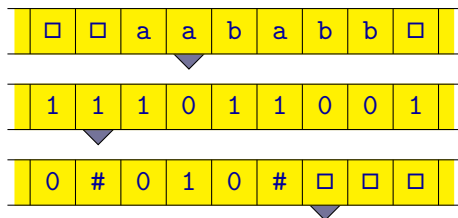


Jedna páska s jednou hlavou: varianta, kde se posunují značky hlav

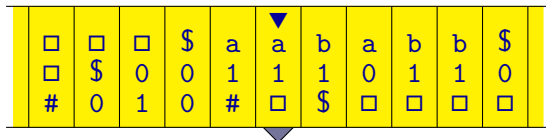


Simulace více pásek pomocí jedné

Více pásek:



Jedna páska s jednou hlavou: varianta, kde se posunují obsahy pásek



Můžeme uvažovat různé stroje, které mají konečnou řídicí jednotku doplněnou o nějaký druh neomezeně velké paměti.

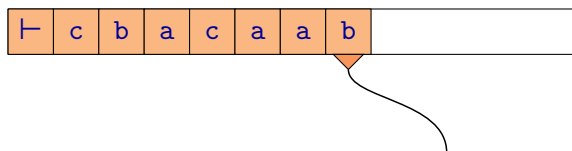
Tato paměť může být tvořena jednou nebo více strukturami, jako jsou třeba:

- **Páska** — čtení a zápis symbolu na aktuální pozici, posun hlavy doleva a doprava
Poznámka: Páska může být jednostranně nebo oboustranně nekonečná.
- **Zásobník** — push, pop, test prázdnosti zásobníku
- **Čítač** — hodnotou je přirozené číslo, operace přičtení nebo odečtení hodnoty jedna, test rovnosti nule

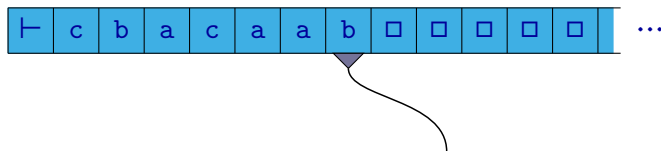
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



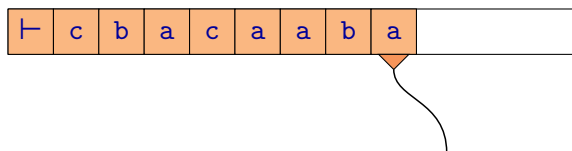
Páska:



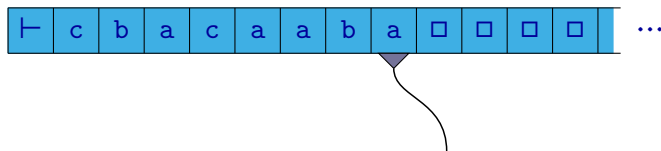
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



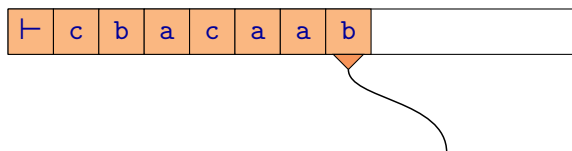
Páska:



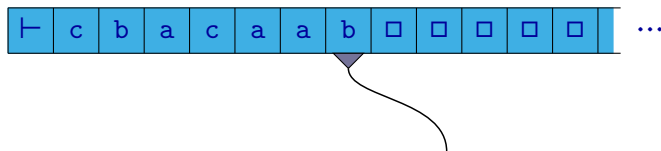
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



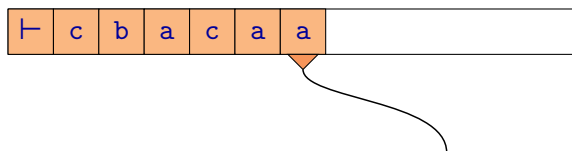
Páska:



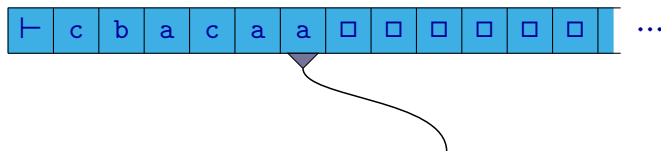
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:



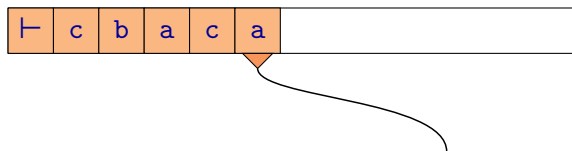
Páska:



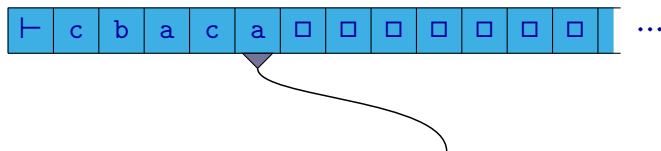
Zásobník

Na zásobník je možné se dívat jako na speciální případ jednostranně nekonečné pásky.

Zásobník:

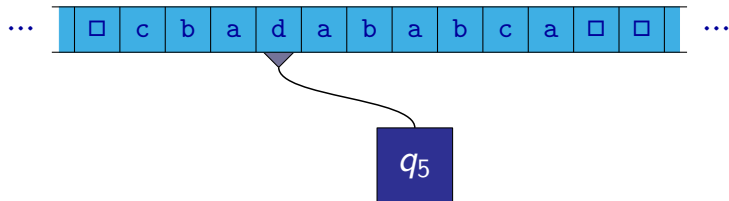


Páska:

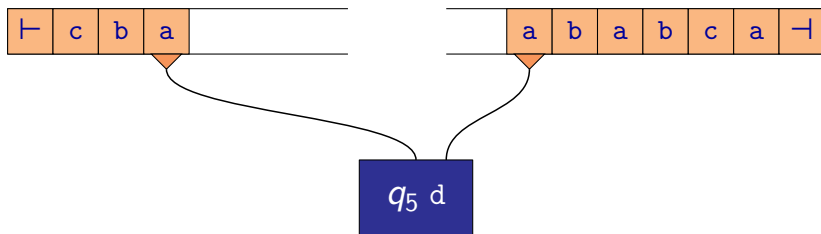


Zásobník

Oboustranně nekonečnou pásku je možné simulovat pomocí dvou zásobníků:

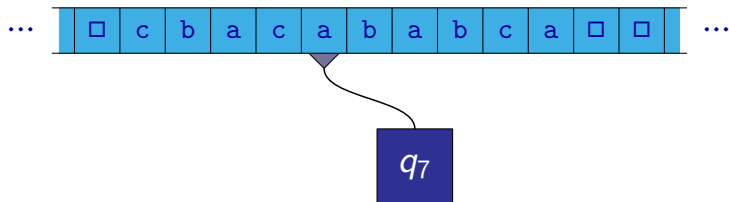


Stroj se dvěma zásobníky:

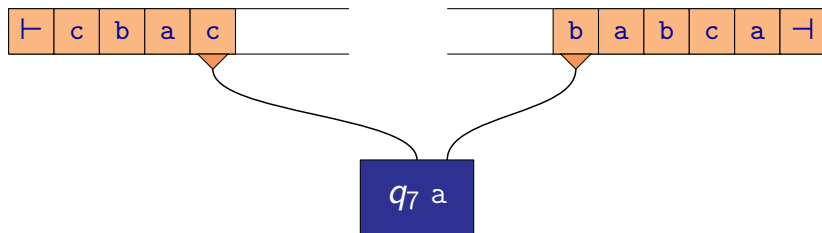


Zásobník

Oboustranně nekonečnou pásku je možné simulovat pomocí dvou zásobníků:



Stroj se dvěma zásobníky:



Čítač — hodnotou čítače může být libovolně velké přirozené číslo, tj. prvek množiny $\mathbb{N} = \{0, 1, 2, 3, \dots\}$.

Základní operace:

- zvýšení hodnoty o jedna:

$$x := x + 1$$

- snížení hodnoty o jedna:

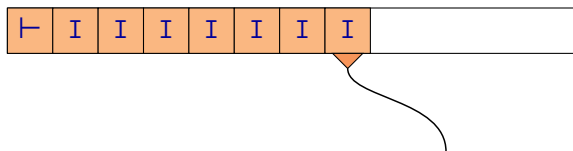
$$x := x - 1$$

- test, jestli je hodnota čítače nula:

if ($x = 0$) **goto** ℓ

Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

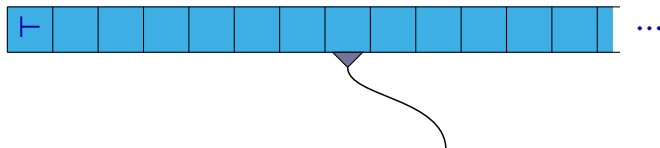
Zásobník:



Čítač:

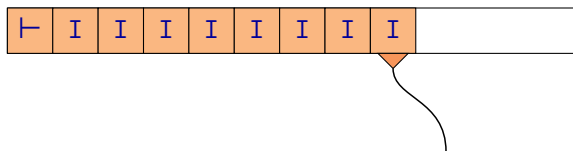


Páska:



Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

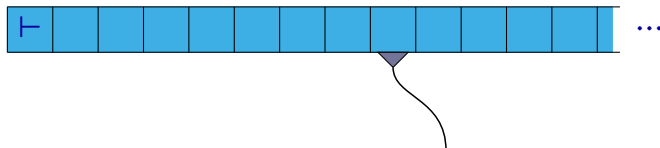
Zásobník:



Čítač:

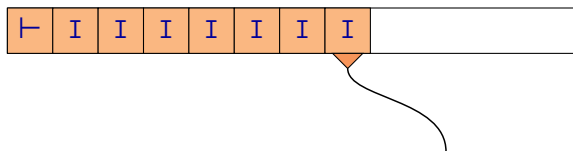


Páska:



Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

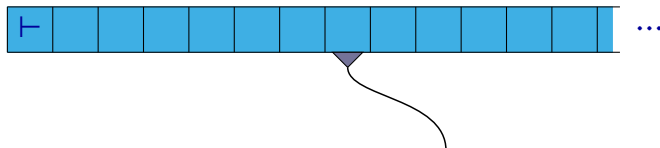
Zásobník:



Čítač:

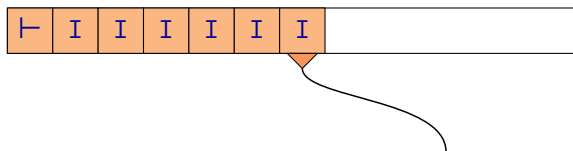


Páska:



Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

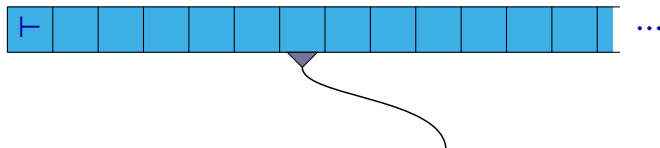
Zásobník:



Čítač:

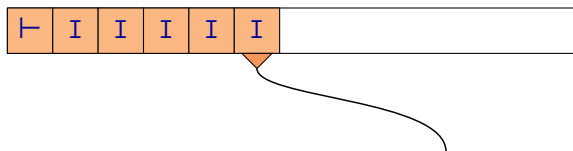


Páska:



Na čítač je možné se dívat jako na speciální případ zásobníku či pásky.

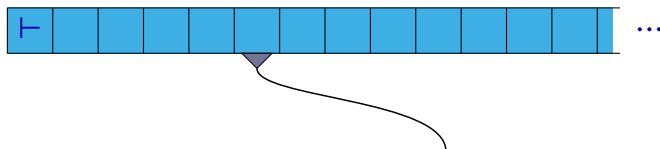
Zásobník:



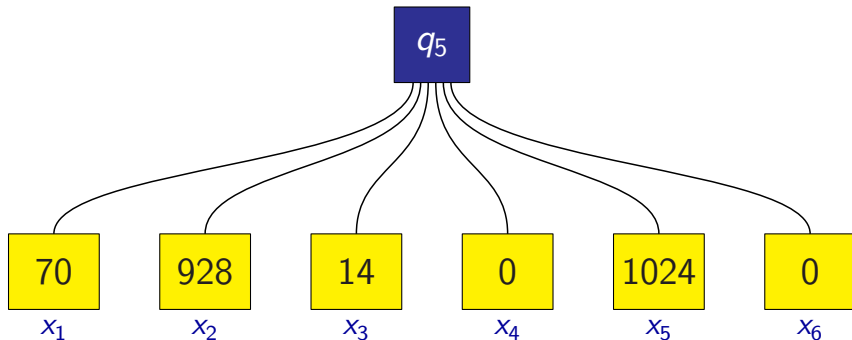
Čítač:



Páska:



Minského stroj — stroj, který má konečnou řídicí jednotku a konečný počet čítačů x_1, x_2, \dots, x_k :



Poznámka: Pro označení čítačů budeme kromě symbolů x_1, x_2, \dots používat také symboly jako x, y, z, \dots

Na Minského stroj se můžeme dívat jako na program tvořený posloupností instrukcí následujících pěti typů:

- zvýšení hodnoty daného čítače o jedna:

$$x_i := x_i + 1$$

- snížení hodnoty daného čítače o jedna:

$$x_i := x_i - 1$$

- test, jestli je hodnota daného čítače nula:

if ($x_i = 0$) **goto** ℓ

- nepodmíněný skok:

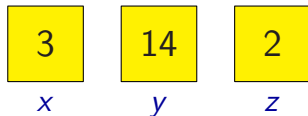
goto ℓ

- zastavení programu:

halt

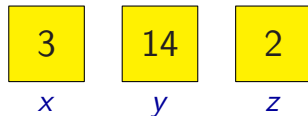
Vynulování čítače x :

→ L_1 : **if** ($x = 0$) **goto** L_2
 $x := x - 1$
 goto L_1
 L_2 : ...



Vynulování čítače x :

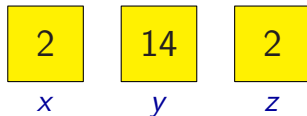
→ L_1 : **if** ($x = 0$) **goto** L_2
 $x := x - 1$
 goto L_1
 L_2 : ...



Vynulování čítače x :

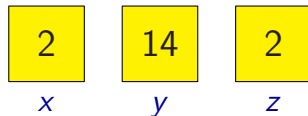
```
 $L_1$  : if ( $x = 0$ ) goto  $L_2$   
       $x := x - 1$   
      goto  $L_1$   
 $L_2$  : ...
```

→



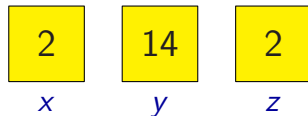
Vynulování čítače x :

→ L_1 : **if** ($x = 0$) **goto** L_2
 $x := x - 1$
 goto L_1
 L_2 : ...



Vynulování čítače x :

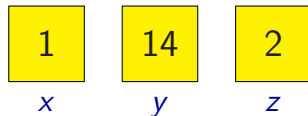
→ L_1 : **if** ($x = 0$) **goto** L_2
 $x := x - 1$
 goto L_1
 L_2 : ...



Vynulování čítače x :

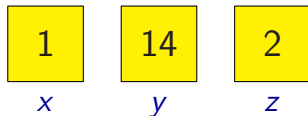
```
 $L_1$  : if ( $x = 0$ ) goto  $L_2$   
       $x := x - 1$   
      goto  $L_1$   
 $L_2$  : ...
```

→



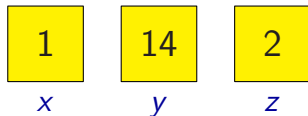
Vynulování čítače x :

→ L_1 : **if** ($x = 0$) **goto** L_2
 $x := x - 1$
 goto L_1
 L_2 : ...



Vynulování čítače x :

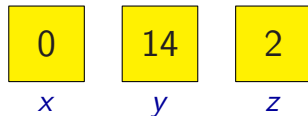
→ L_1 : **if** ($x = 0$) **goto** L_2
 $x := x - 1$
 goto L_1
 L_2 : ...



Vynulování čítače x :

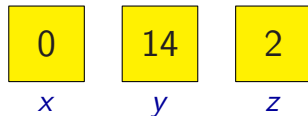
```
 $L_1$  : if ( $x = 0$ ) goto  $L_2$   
       $x := x - 1$   
      goto  $L_1$   
 $L_2$  : ...
```

→



Vynulování čítače x :

```
→  $L_1$  : if ( $x = 0$ ) goto  $L_2$   
       $x := x - 1$   
      goto  $L_1$   
 $L_2$  : ...
```



Vynulování čítače x :

L_1 : **if** ($x = 0$) **goto** L_2

$x := x - 1$

goto L_1

→ L_2 : ...

0

x

14

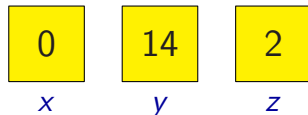
y

2

z

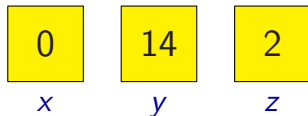
Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

→ L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



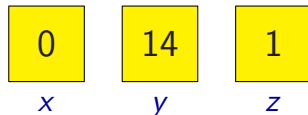
Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

→ L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

L_2 : **if** ($z = 0$) **goto** L_3

$z := z - 1$

$y := y + 1$

goto L_1

L_3 : ...

0

x

15

y

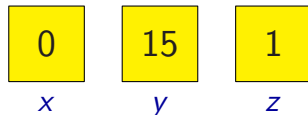
1

z



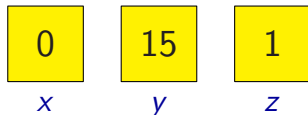
Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

→ L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



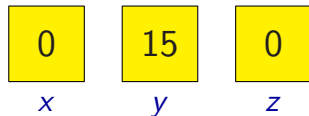
Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

→ L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



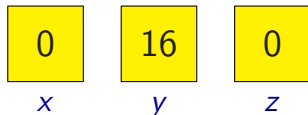
Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



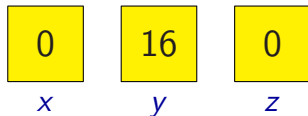
Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

```
 $L_2$  : if ( $z = 0$ ) goto  $L_3$   
       $z := z - 1$   
       $y := y + 1$   
      goto  $L_1$   
 $L_3$  : ...
```



Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

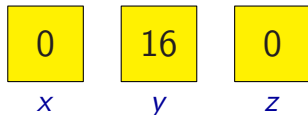
→ L_2 : **if** ($z = 0$) **goto** L_3
 $z := z - 1$
 $y := y + 1$
 goto L_1
 L_3 : ...



Přičtení obsahu čítače z k čítači y (a vynulování čítače z):

```
 $L_2$  : if ( $z = 0$ ) goto  $L_3$   
       $z := z - 1$   
       $y := y + 1$   
      goto  $L_1$ 
```

→ L_3 : ...



Vynásobení hodnoty čítače x číslem 5:

L_1 : **if** ($x = 0$) **goto** L_2

$x := x - 1$

$y := y + 1$

$y := y + 1$

$y := y + 1$

$y := y + 1$

$y := y + 1$

goto L_1

L_2 : **if** ($y = 0$) **goto** L_3

$y := y - 1$

$x := x + 1$

goto L_2

L_3 : ...

Vydělení hodnoty čítače x číslem 5 a zjištění zbytku po dělení:

```
 $L_1$  : if ( $x = 0$ ) goto  $M_0$   
       $x := x - 1$   
      if ( $x = 0$ ) goto  $M_1$   
       $x := x - 1$   
      if ( $x = 0$ ) goto  $M_2$   
       $x := x - 1$   
      if ( $x = 0$ ) goto  $M_3$   
       $x := x - 1$   
      if ( $x = 0$ ) goto  $M_4$   
       $x := x - 1$   
       $y := y + 1$   
      goto  $L_1$ 
```

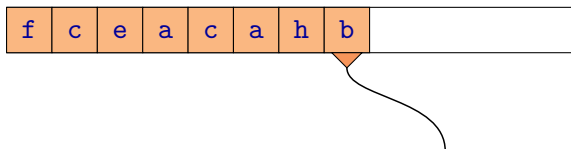
Zásobník je možné simulovat pomocí dvou čítačů — hodnota jednoho čítače reprezentuje obsah zásobníku jako číslo, jehož zápis v číselné soustavě o základu $k = |\Gamma| + 1$ (kde Γ je zásobníková abeceda) odpovídá obsahu zásobníku.

- Symbol na vrcholu zásobníku — zbytek po dělení číslem k
- Pop — vydělit číslem k
- Push — vynásobit číslem k a přičíst kód příslušného symbolu

Druhý čítač slouží jako pomocný při provádění výše uvedených operací.

Příklad:

a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9



63513182

Příklad:

a ↔ 1

b ↔ 2

c ↔ 3

d ↔ 4

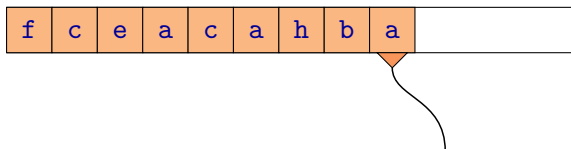
e ↔ 5

f ↔ 6

g ↔ 7

h ↔ 8

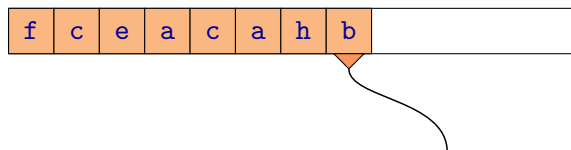
i ↔ 9



635131821

Příklad:

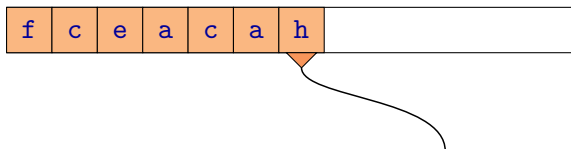
a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9



63513182

Příklad:

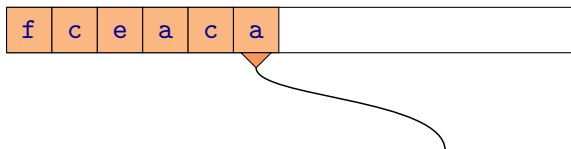
a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9



6351318

Příklad:

a ↔ 1
b ↔ 2
c ↔ 3
d ↔ 4
e ↔ 5
f ↔ 6
g ↔ 7
h ↔ 8
i ↔ 9



635131

Připomeňme, že oboustranně nekonečnou pásku je možné simulovat pomocí dvou zásobníků.

V Minského stroji může být obsah každého z těchto zásobníků reprezentován jemu odpovídajícím čítačem.

Navíc potřebujeme ještě jeden pomocný čítač pro implementaci operací násobení a dělení na těchto čítačích reprezentujících obsahy zásobníků.

Vidíme, že Turingův stroj s k páskami je možné simulovat Minského strojem s $2k + 1$ čítači.

Libovolný konečný počet čítačů je možné simulovat pomocí dvou čítačů.

- Jeden čítač (označme jej C) reprezentuje hodnoty všech čítačů — např. hodnoty tří čítačů x , y , z mohou být v čítači C reprezentovány jako číslo $2^x 3^y 5^z$.
- Druhý čítač je používán jako pomocný při provádění operací násobení a dělení na čítači C .
- Přičtení jedničky k čítači x je simulováno jako vynásobení čítače C hodnotou 2 , přičtení jedničky k čítači y jako vynásobení hodnotou 3 , atd.
- Analogicky je odečtení jedničky od čítače x simulováno pomocí vydělení čítače C hodnotou 2 , odečtení jedničky od čítače y vydělením hodnotou 3 , atd.
- Test podmínky $x = 0$ odpovídá testu, zda je hodnota C dělitelná dvěma, atd.

Vidíme, že činnost libovolného Turingova stroje je možné simulovat Minského strojem s dvěma čítači.

Tato simulace je však mimořádně neefektivní:

- Už simulace pásky Turingova stroje pomocí tří čítačů vyžaduje exponenciálně větší počet kroků, než kolik by jich vykonal tento Turingův stroj.
- Simulace činnosti těchto tří čítačů pomocí dvou čítačů tento počet kroků dále exponenciálně zvyšuje.

Stroje RAM

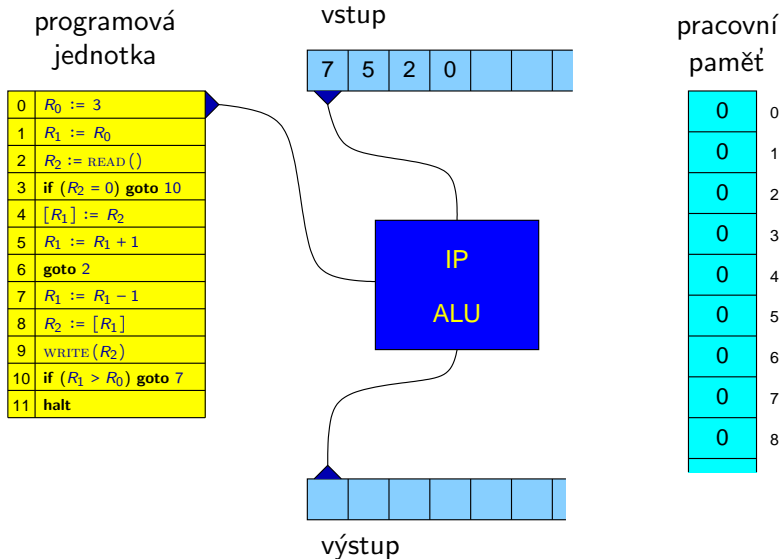
Stroj RAM (Random Access Machine) je idealizovaný model počítače.

Skládá se z těchto částí:

- **Programová jednotka** – obsahuje program stroje RAM a ukazatel na právě prováděnou instrukci
- **Pracovní paměť** tvořená buňkami očíslovanými $0, 1, 2, \dots$
Tyto buňky budeme označovat R_0, R_1, R_2, \dots
Obsah buněk je možno číst i do nich zapisovat.
- **Vstupní páska** – je z ní možné pouze číst
- **Výstupní páska** – je na ni možno pouze zapisovat

Buňky paměti i vstupní a výstupní páska obsahují jako hodnoty celá čísla (tj. prvky množiny \mathbb{Z}).

Stroj RAM



Přehled instrukcí:

- $R_i := c$ – přiřazení konstanty
- $R_i := R_j$ – přiřazení
- $R_i := [R_j]$ – load (čtení z paměti)
- $[R_i] := R_j$ – store (zápis do paměti)
- $R_i := R_j \text{ op } R_k$
nebo $R_i := R_j \text{ op } c$ – aritmetické instrukce, $op \in \{+, -, *, /\}$
- if** ($R_i \text{ rel } R_j$) **goto** ℓ
nebo **if** ($R_i \text{ rel } c$) **goto** ℓ – podmíněný skok, $rel \in \{=, \neq, \leq, \geq, <, >\}$
- goto** ℓ – nepodmíněný skok
- $R_i := \text{READ}()$ – čtení ze vstupu
- WRITE** (R_i) – zápis na výstup
- halt** – zastavení programu

Příklady instrukcí:

$R_5 := 42$

$R_{12} := R_3$

$R_8 := [R_2]$

$[R_{15}] := R_9$

$R_7 := R_3 + R_6$

$R_{18} := R_{18} - 1$

if ($R_4 \geq R_1$) **goto** 2801

if ($R_2 \neq 0$) **goto** 3581

goto 537

$R_{23} := \text{READ}()$

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

halt

- přiřazení konstanty
- přiřazení
- load (čtení z paměti)
- store (zápis do paměti)
- aritmetická instrukce
- aritmetická instrukce
- podmíněný skok
- podmíněný skok
- nepodmíněný skok
- čtení ze vstupu
- zápis na výstup
- zastavení programu

Stroj RAM



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

Input



Output

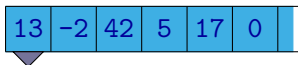
0	?
1	?
2	?
3	?
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM



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

Input



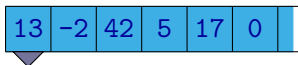
Output

0	3
1	?
2	?
3	?
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



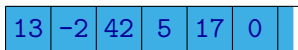
Output

0	3
1	3
2	?
3	?
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

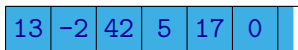
0	3
1	3
2	13
3	?
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

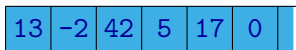
0	3
1	3
2	13
3	?
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

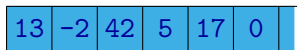
0	3
1	3
2	13
3	13
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



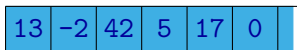
Output

0	3
1	4
2	13
3	13
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



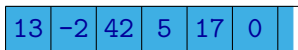
Output

0	3
1	4
2	13
3	13
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



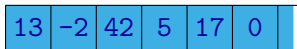
Output

0	3
1	4
2	-2
3	13
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

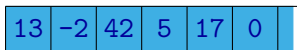
0	3
1	4
2	-2
3	13
4	?
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

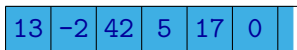
0	3
1	4
2	-2
3	13
4	-2
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



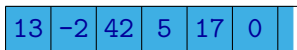
Output

0	3
1	5
2	-2
3	13
4	-2
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



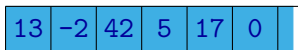
Output

0	3
1	5
2	-2
3	13
4	-2
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



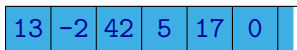
Output

0	3
1	5
2	42
3	13
4	-2
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

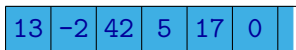
0	3
1	5
2	42
3	13
4	-2
5	?
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

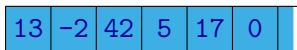
0	3
1	5
2	42
3	13
4	-2
5	42
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



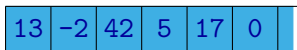
Output

0	3
1	6
2	42
3	13
4	-2
5	42
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



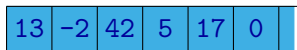
Output

0	3
1	6
2	42
3	13
4	-2
5	42
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



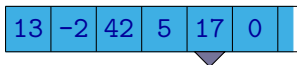
Output

0	3
1	6
2	5
3	13
4	-2
5	42
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

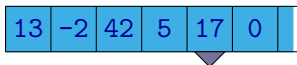
0	3
1	6
2	5
3	13
4	-2
5	42
6	?
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

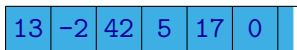
0	3
1	6
2	5
3	13
4	-2
5	42
6	5
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



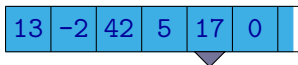
Output

0	3
1	7
2	5
3	13
4	-2
5	42
6	5
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



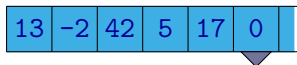
Output

0	3
1	7
2	5
3	13
4	-2
5	42
6	5
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

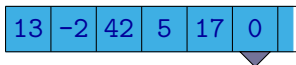
0	3
1	7
2	17
3	13
4	-2
5	42
6	5
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

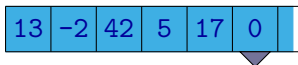
0	3
1	7
2	17
3	13
4	-2
5	42
6	5
7	?
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

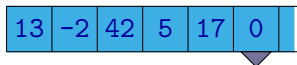
0	3
1	7
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



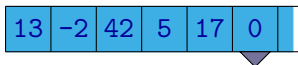
Output

0	3
1	8
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



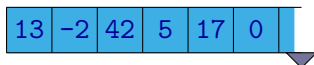
Output

0	3
1	8
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



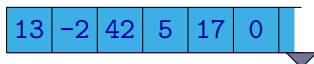
Output

0	3
1	8
2	0
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



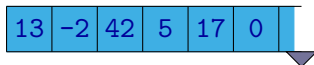
Output

0	3
1	8
2	0
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

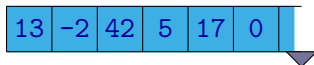
0	3
1	8
2	0
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



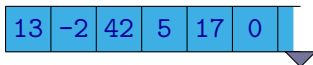
Output

0	3
1	7
2	0
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



0	3
1	7
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

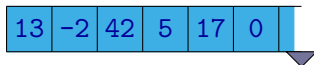


Output

Stroj RAM

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

Input



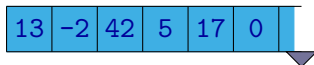
Output

0	3
1	7
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

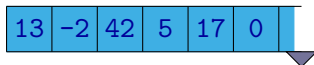
0	3
1	7
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

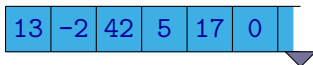
0	3
1	6
2	17
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



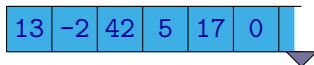
Output

0	3
1	6
2	5
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



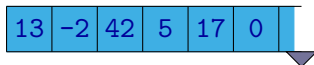
Output

0	3
1	6
2	5
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

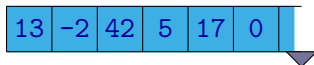
0	3
1	6
2	5
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

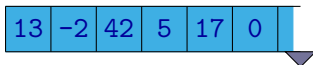
0	3
1	5
2	5
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



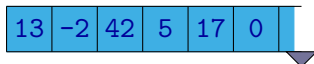
Output

0	3
1	5
2	42
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



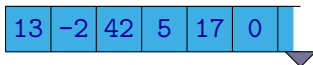
Output

0	3
1	5
2	42
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



0	3
1	5
2	42
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?



Output

Stroj RAM

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



Input



Output

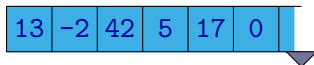
0	3
1	4
2	42
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



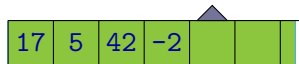
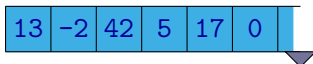
Output

0	3
1	4
2	-2
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



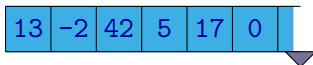
Output

0	3
1	4
2	-2
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

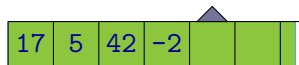
Stroj RAM

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

Input



0	3
1	4
2	-2
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?



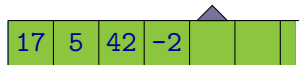
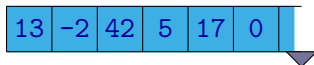
Output

Stroj RAM

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



Input



Output

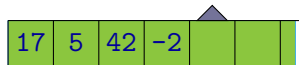
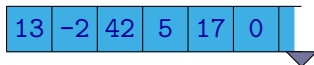
0	3
1	3
2	-2
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



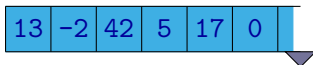
Output

0	3
1	3
2	13
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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

Input



Output

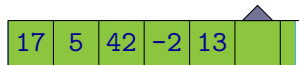
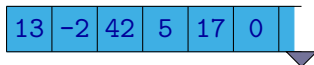
0	3
1	3
2	13
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Stroj RAM

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



Input



Output

0	3
1	3
2	13
3	13
4	-2
5	42
6	5
7	17
8	?
9	?
10	?
11	?

Rozdíly oproti skutečnému počítači:

- Velikost paměti není omezena (adresa může být libovolné přirozené číslo).
- Velikost obsahu jednotlivých buněk není omezena (buňka může obsahovat libovolné celé číslo).
- Čte data sekvenčně ze vstupu, který je tvořen sekvencí celých čísel. Ze vstupu lze pouze číst.
- Zapisuje data sekvenčně na výstup, který je tvořen sekvencí celých čísel. Na výstup je možné pouze zapisovat.

- Operace jako přístup k buňce paměti na adrese menší než nula nebo dělení nulou vedou k chybě — výpočet se (neúspěšně) zastaví.
- Co se týká počátečního obsahu paměti, jsou dvě možnosti, jak ho definovat:
 - Všechny buňky jsou inicializovány hodnotou 0.
 - Čtení obsahu buňky, do které nebylo dosud nic zapsáno, způsobí chybu. Buňky na začátku obsahují speciální hodnotu (označenou zde symbolem '?'), která reprezentuje to, že buňka nebyla dosud inicializována.
- Uvažují se i varianty strojů RAM, kde buňky paměti (a vstupu a výstupu) neobsahují celá čísla (tj. prvky množiny \mathbb{Z}), ale mohou obsahovat jen přirozená čísla (tj. prvky množiny \mathbb{N}).

Například operace odčítání ($R_i := R_j - R_k$) se pak chová tak, že pokud by výsledkem mělo být záporné číslo, je jako výsledek operace přiřazena hodnota 0.

- Různé varianty strojů RAM se mohou lišit tím, jaké konkrétní operace v aritmetických instrukcích podporují nebo naopak nepodporují.

Například:

- podpora bitových operací (and, or, not, xor, ...), bitový posunů, ...
 - varianta stroje RAM, která nemá operace násobení a dělení
- Mohli bychom také uvažovat variantu stroje RAM, kde místo instrukcí tvaru

if ($R_i \text{ rel } R_j$) **goto** ℓ nebo **if** ($R_i \text{ rel } c$) **goto** ℓ

jsou všechny podmíněné skoky jen tvaru

if ($R_i \text{ rel } 0$) **goto** ℓ

Místo všech relací $\{=, \neq, \leq, \geq, <, >\}$ může být podporována jen nějaká podmnožina z nich, např. $\{=, >\}$.

- V některých variantách stroje RAM nemají vstup a výstup podobu sekvence čísel.

Místo toho pracuje stroj z hlediska vstupu a výstupu s páskami obsahujícími sekvence symbolů z nějaké dané abecedy, např. $\{0, 1\}$.

Stroj má pak například instrukce, které mu umožňují větvit výpočet podle symbolu přečteného ze vstupu.

Vnitřní paměť ovšem i v této variantě pracuje s čísly.

- Pokud má stroj jako výsledek dávat jen odpověď Ano/Ne (tj. přijmout nebo nepřijmout daný vstup), nemusí mít výstupní pásku.

Instrukce **halt** je pak nahrazena instrukcemi **accept** a **reject**.

- Ve standardní definici stroje RAM se většinou neuvažují instrukce skoku na adresu instrukce uloženou v buňce paměti, tj. instrukce typu

goto R_i

Stroj RAM bychom mohli rozšířit o tento druh instrukcí.

- Jako standardní se u stroje RAM bere to, že kód programu není uložen v pracovní paměti, ale má zvláštní samostatnou paměť, která je jen pro čtení.

V průběhu výpočtu se tedy kód programu nemůže měnit.

- Druh stroje podobný stroji RAM, kde je ovšem program uložen v pracovní paměti (instrukce jsou kódovány čísly) a je možné ho průběhu výpočtu měnit, se označuje jako stroj **RASP** (**random-access stored program**).

Stroj RASP tak umožňuje provádět činnost sebemodifikujících se programů.

Turingův stroj simulující činnost stroje RAM

Není těžké si rozmyslet, že činnost libovolného Turingova stroje je možné simulovat pomocí stroje RAM.

Promyslet si, že i naopak činnost každého stroje RAM je možné simulovat Turingovým strojem, je o něco komplikovanější.

Při popisu toho, jak simulovat činnost stroje RAM pomocí Turingova stroje, budeme postupovat po menších krocích:

- Ukážeme, jak činnost stroje RAM ve variantě, kterou jsme si popsali, simulovat variantou stroje RAM s poněkud jednoduššími instrukcemi.
- Ukážeme, jak činnost této jednodušší varianty stroje RAM simulovat vícepáskovým Turingovým strojem.
- Už dříve jsme viděli, jak činnosti vícepáskového Turingova stroje simulovat pomocí jednopáskového Turingova stroje.

Tato jednodušší varianta stroje RAM bude mít kromě pracovní paměti tři **registry**:

- **registr A** — téměř všechny instrukce pracují s tímto registrem, výsledky všech operací se ukládají do tohoto registru

Poznámka: Tento druh registru se často označuje jako **akumulátor**.

- **registr B** — tento registr slouží k uložení druhého operandu pro aritmetické instrukce (první operand je vždy v akumulátoru)
- **registr C** — tento registr slouží k uložení adresy, na kterou bude zapisovat instrukce store

Jednodušší varianta stroje RAM

Přehled instrukcí:

$A := c$	– přiřazení konstanty
$B := A$	– přiřazení do registru B
$C := A$	– přiřazení do registru C
$A := [A]$	– load (čtení z paměti)
$[C] := A$	– store (zápis do paměti)
$A := A \text{ op } B$	– aritmetické instrukce, $op \in \{+, -, *, /\}$
if ($A \text{ rel } 0$) goto ℓ	– podmíněný skok, $rel \in \{=, \neq, \leq, \geq, <, >\}$
goto ℓ	– nepodmíněný skok
$A := \text{READ}()$	– čtení ze vstupu
$\text{WRITE}(A)$	– zápis na výstup
halt	– zastavení programu

Jednodušší varianta stroje RAM

Například instrukce

$$R_7 := R_3 + R_6$$

může být nahrazena posloupností instrukcí:

$$A := 7$$

$$C := A$$

$$A := 6$$

$$A := [A]$$

$$B := A$$

$$A := 3$$

$$A := [A]$$

$$A := A + B$$

$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

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

může být nahrazena posloupností instrukcí:

$$A := 15$$

$$A := [A]$$

$$C := A$$

$$A := 9$$

$$A := [A]$$

$$[C] := A$$

Jednodušší varianta stroje RAM

Například instrukce

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

může být nahrazena posloupností instrukcí:

```
A := 11
```

```
A := [A]
```

```
B := A
```

```
A := 4
```

```
A := [A]
```

```
A := A - B
```

```
if ( $A \geq 0$ ) goto  $\ell$ 
```

Turingův stroj simulující činnost stroje RAM

Turingův stroj pracuje se slovy nad nějakou abecedou, zatímco stroj RAM s čísly. Čísla ale můžeme zapisovat jako sekvence symbolů a naopak symboly nějaké abecedy můžeme zapisovat jako čísla.

Například následující vstup stroje RAM

5	13	-3	0	6	
---	----	----	---	---	--

může být v případě Turingova stroje reprezentován jako

#	1	0	1	#	1	1	0	1	#	-	1	1	#	0	#	1	1	0	#
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Turingův stroj simulující činnost stroje RAM bude mít několik pásek:

- Pásku, na které bude uložen obsah pracovní paměti stroje RAM.
- Tři pásy, na kterých budou uloženy hodnoty registrů A , B a C .
(Hodnoty registrů A , B a C budou na těchto páskách zapsány binárně bez vedoucích nul a zleva a zprava budou ohraničeny symboly #.)
- Pásku reprezentující vstupní pásku stroje RAM.
- Pásku reprezentující výstupní pásku stroje RAM.
- Jednu pomocnou pásku používanou při implementaci simulace jednotlivých instrukcí.

Turingův stroj simulující činnost stroje RAM

Turingův stroj si bude v řídicí jednotce pamatovat, která instrukce stroje RAM se právě provádí.

Provedení většiny instrukcí není složité:

- $A := c$
zapiše jednotlivé bity konstanty c na pásku registru A
- $B := A$ nebo $C := A$
zkopíruje obsah pásky registru A na pásku registru B nebo C
- **goto** l
změní se jen stav řídicí jednotky Turingova stroje
- **if** ($A \text{ rel } 0$) **goto** l , kde $rel \in \{=, \neq, \leq, \geq, <, >\}$
snadno se otestuje obsah registru A a podle výsledku se změní stav řídicí jednotky Turingova stroje

Turingův stroj simulující činnost stroje RAM

- $A := \text{READ}()$

zkopírování hodnoty (ohraňené znaky “#”) ze vstupní pásky na pásku registru A

- $\text{WRITE}(A)$

zkopírování hodnoty registru A na výstupní pásku.

- **halt**

výpočet se zastaví

Také aritmetické instrukce jsou poměrně jednoduché, i když o něco složitější než předchozí instrukce:

- $A := A \text{ op } B$, kde $\text{op} \in \{+, -, *, /\}$

Příslušnou operaci (např. sčítání nebo odčítání) provede Turingův stroj bit po bitu, výsledek je ukládán do registru A .

Poznámka: Násobení a dělení je možné realizovat pomocí série sčítání, odčítání a bitových posunů.

Při implementaci násobení a dělení může být potřeba použít pomocnou pásku k ukládání mezivýsledků.

Turingův stroj simulující činnost stroje RAM

Asi nejsložitější je realizace pracovní paměti stroje RAM.

Jednou z možností je pamatovat si jen obsah těch buněk, se kterými stroj RAM v průběhu své činnosti někdy pracoval.

Příklad: Stroj RAM zatím pracoval jen s buňkami 2, 3 a 6:

- Buňka 2 obsahuje hodnotu 11.
- Buňka 3 obsahuje hodnotu -1.
- Buňka 6 obsahuje hodnotu 2.

Obsah pásky Turingova stroje reprezentující buňky paměti stroje RAM bude následující:

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

Instrukce load, tj. $A := [A]$:

- Turingův stroj bude hledat příslušnou adresu uloženou v registru A na pásce reprezentující obsah paměti stroje RAM.
(Pokud ji nenajdeme, přidá ji na konec, s tím, že obsahuje hodnotu 0.)
- Příslušnou hodnotu zkopíruje na pásku registru A .

Instrukce store, tj. $[C] := A$:

- Podobně jako u instrukce load se najde příslušné místo na pásce reprezentující pracovní paměť, kde se nachází obsah buňky, jejíž adresa je v registru C .
- Zbytek pásky s obsahem paměti stroje RAM se zkopíruje na pomocnou pásku.
- Na příslušné místo se zkopíruje obsah pásky registru A .
- Zbytek pásky, který byl zkopírován na pomocnou pásku, se zkopíruje zpět (za nově zapsanou hodnotu).