Základy složitosti a vyčíslitelnosti

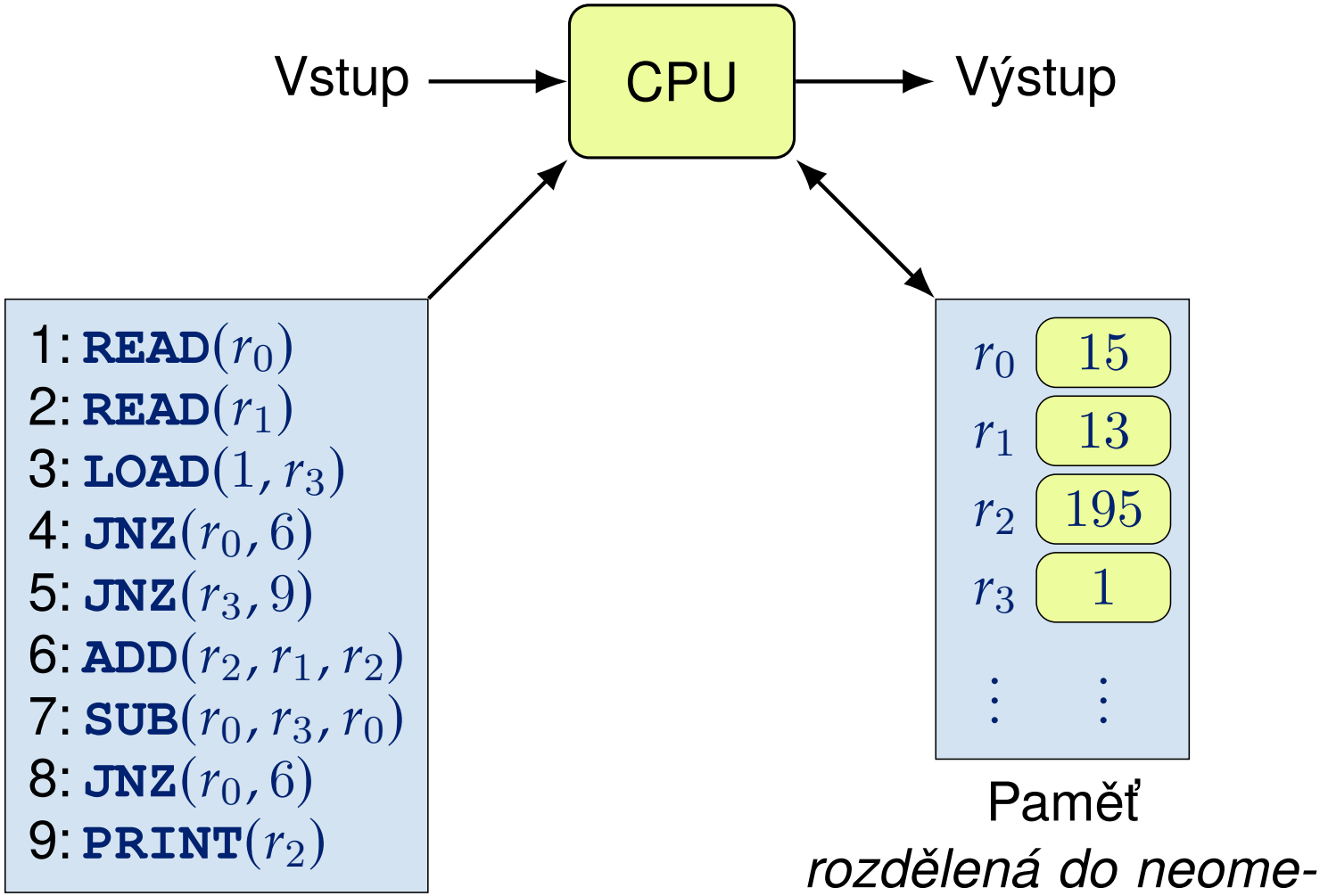
NTIN090

Petr Kučera

2020/21 (2. přednáška)

Random Access Machine

Random Access Machine (RAM)



Random Access Machine (definice)

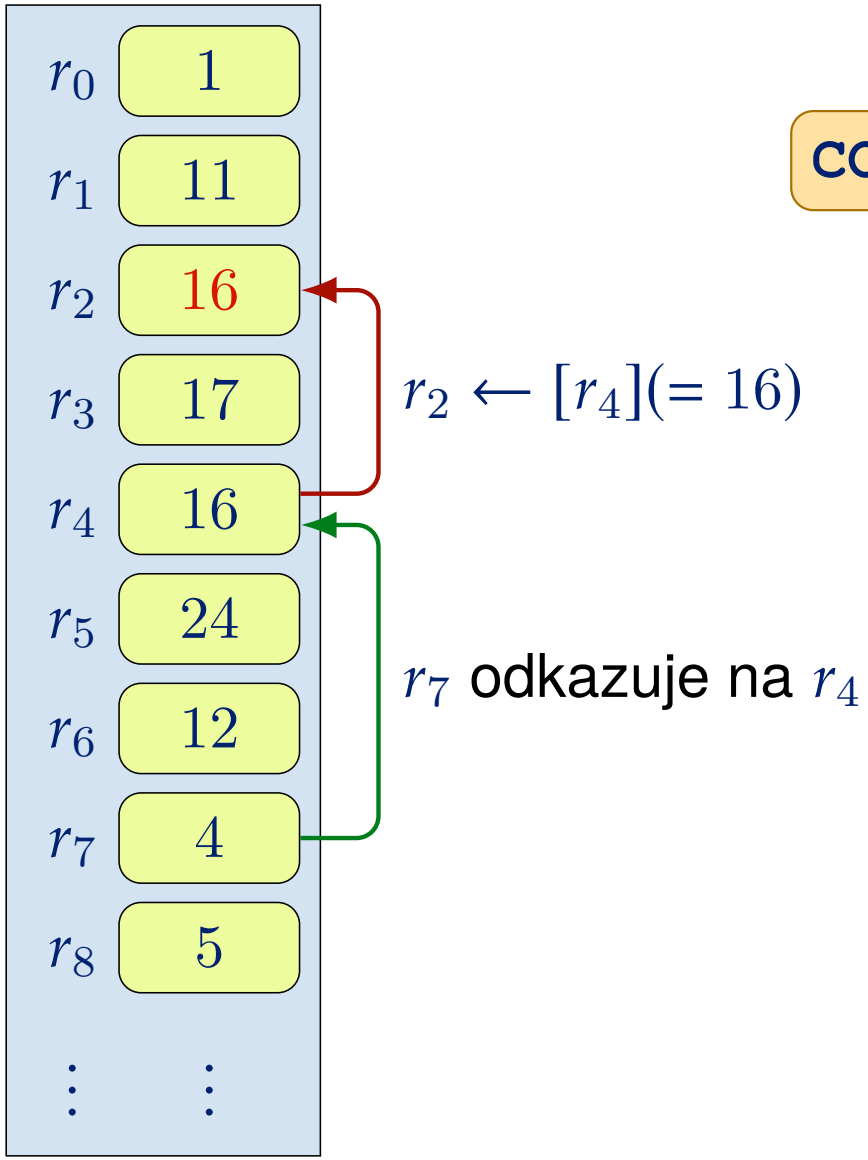
- Random Access Machine (RAM) se skládá z
  - řídící jednotky (procesoru, CPU) a
  - neomezené paměti
- Paměť RAMu je rozdělená do registrů r_i, $i \in \mathbb{N}$.
- V každém registru může být libovolné přirozené číslo
  - na začátku obsahují 0
- $[r_i]$ označuje obsah registru r_i
- Nepřímá adresace: $[\![r_i]\!] = [r_{[r_i]}]$
- Programem pro RAM je konečná posloupnost instrukcí $P = I_0, I_1, \ldots, I_\ell$
- Instrukce jsou vykonávané v pořadí daném programem

Možné instrukce RAM

Instrukce	Efekt
LOAD(C, r_i)	$r_i \leftarrow C$
ADD(r_i, r_j, r_k)	$r_k \leftarrow [r_i] + [r_j]$
SUB(r_i, r_j, r_k)	$r_k \leftarrow [r_i] \dot{-} [r_j]$
COPY$([r_p], r_d)$	$r_d \leftarrow [\![r_p]\!]$
COPY$(r_s, [r_d])$	$r_{[r_d]} \leftarrow [r_s]$
JNZ(r_i, I_z)	**if** $[r_i] > 0$ **then goto** z
READ(r_i)	$r_i \leftarrow$ input
PRINT(r_i)	output $\leftarrow [r_i]$

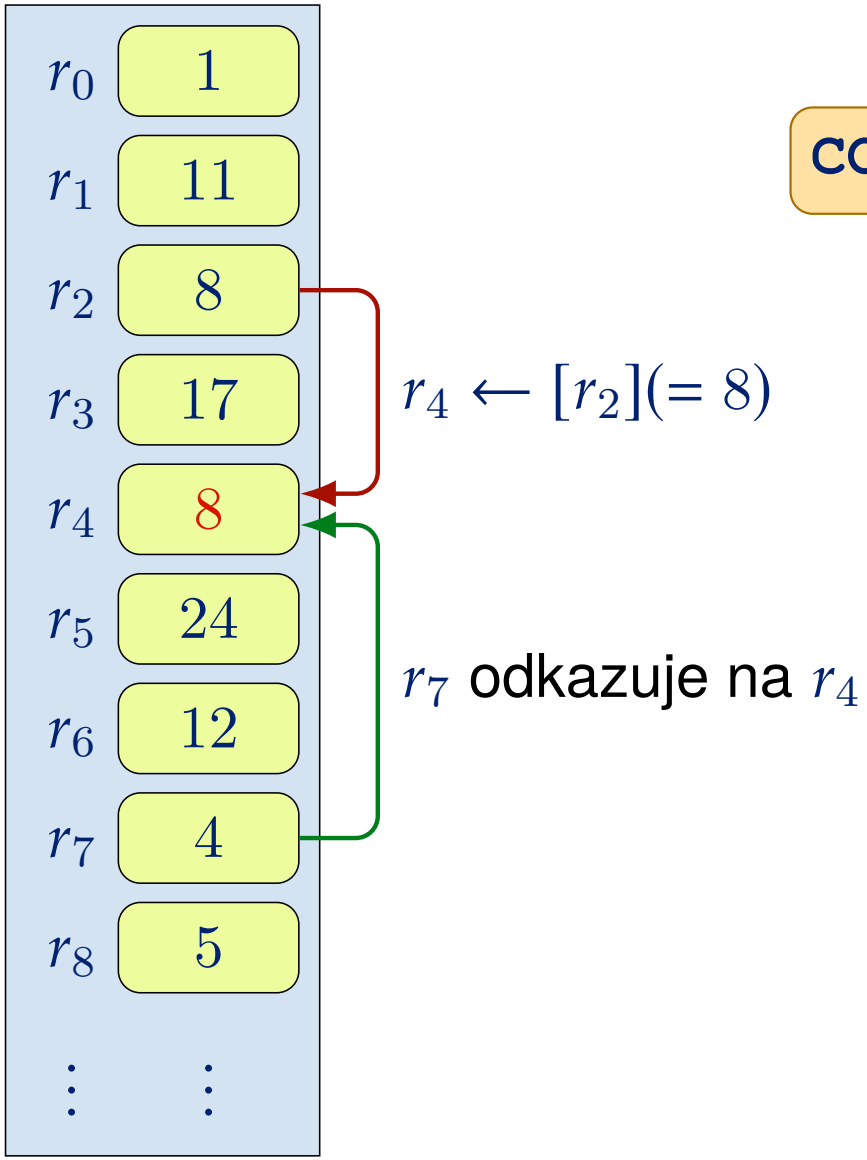
$$x \dot{-} y = \begin{cases} x - y & x > y \\ 0 & \text{jinak} \end{cases}$$

Nepřímá adresace zdroje



COPY$([r_7], r_2)$ provede $r_2 \leftarrow [\![r_7]\!]$

Nepřímá adresace cíle



COPY$(r_2, [r_7])$ provede $r_{[r_7]} \leftarrow [r_2]$

RAM pro součin čísel

- Naším cílem je konstrukce RAM, který
  - Načte dvě čísla x a y ze vstupu a
  - vypíše na výstup jejich součin $x \cdot y$

Algoritmus 1: Algoritmus RAM pro výpočet součinu

```
1 READ(x)
2 READ(y)
3 z ← 0
4 while x > 0 do
5 |   z ← z + y
6 |   x ← x ∸ 1
7 PRINT(z)
```

RAM pro součin čísel

Proměnným přiřadíme registry $r_0 \leftarrow x$, $r_1 \leftarrow y$, $r_2 \leftarrow z$.

Algoritmus 2: Algoritmus RAM pro výpočet součinu

```
1 READ(r_0)  // x
2 READ(r_1)  // y
3 r_2 ← 0  // z ← 0
4 while [r_0] > 0 do
5 |   [r_2] ← [r_2] + [r_1]
6 |   [r_0] ← [r_0] ∸ 1
7 PRINT(r_2)
```

RAM pro součin čísel

Nahradíme **while** cyklus podmínkami a skoky

Algoritmus 3: Algoritmus RAM pro výpočet součinu

```
1 READ(r_0)
2 READ(r_1)
3 r_2 ← 0
4 if [r_0] > 0 then
5 |   [r_2] ← [r_2] + [r_1]
6 |   [r_0] ← [r_0] ∸ 1
7 |   goto 4
8 PRINT(r_2)
```

RAM pro součin čísel

Přepíšeme jen s pomocí podmíněných a nepodmíněných skoků

Algoritmus 4: Algoritmus RAM pro výpočet součinu

1 **READ**(r_0)
2 **READ**(r_1)
3 $r_2 \leftarrow 0$
4 **if** $[r_0] > 0$ **then goto** 6
5 **goto** 9
6 $[r_2] \leftarrow [r_2] + [r_1]$
7 $[r_0] \leftarrow [r_0] \dot{-} 1$
8 **if** $[r_0] > 0$ **then goto** 6
9 **PRINT**(r_2)

RAM pro součin čísel

Přidáme pomocný registr s konstantou 1

Algoritmus 5: Algoritmus RAM pro výpočet součinu

```
1  READ(r_0)
2  READ(r_1)
3  r_2 ← 0
4  r_3 ← 1
5  if [r_0] > 0 then goto 7
6  goto 10
7  [r_2] ← [r_2] + [r_1]
8  [r_0] ← [r_0] ∸ [r_3]          // [r_0] ← [r_0] ∸ 1
9  if [r_0] > 0 then goto 7
10 PRINT(r_2)
```

RAM pro součin čísel

Nahradíme nepodmíněný skok podmíněnými
(*se zaručeně splněnou podmínkou*)

Algoritmus 6: Algoritmus RAM pro výpočet součinu

1 **READ**(r_0)
2 **READ**(r_1)
3 $r_2 \leftarrow 0$
4 $r_3 \leftarrow 1$
5 **if** $[r_0] > 0$ **then goto** 7
6 **if** $[r_3] > 0$ **then goto** 10 // **goto** 10
7 $[r_2] \leftarrow [r_2] + [r_1]$
8 $[r_0] \leftarrow [r_0] \dot{-} [r_3]$ // $[r_0] \leftarrow [r_0] \dot{-} 1$
9 **if** $[r_0] > 0$ **then goto** 7
10 **PRINT**(r_2)

RAM pro součin čísel

Přepíšeme pomocí instrukcí RAM

Algoritmus 7: Program RAM pro výpočet součinu

```
1 READ(r_0)
2 READ(r_1)
  // r_2 ← 0 není třeba
3 LOAD(1, r_3)                                  // r_3 ← 1
4 JNZ(r_0, 6)                          // if [r_0] > 0 then goto 6
5 JNZ(r_3, 9)                                   // goto 9
6 ADD(r_2, r_1, r_2)                    // [r_2] ← [r_2] + [r_1]
7 SUB(r_0, r_3, r_0)                    // [r_0] ← [r_0] ∸ 1
8 JNZ(r_0, 6)                          // if [r_0] > 0 then goto 6
9 PRINT(r_2)
```

Programování na RAMu

Programy pro RAM odpovídají procedurálnímu jazyku:

proměnné	skalární i neomezená pole
cykly	`for` i `while` s pomocí podmíněného skoku, případně čítače v proměnné
nepodmíněný skok	`goto` s použitím pomocného registru, kam uložíme `1` a použijeme podmíněný skok
podmíněný příkaz	s pomocí podmíněného skoku
funkce a procedury	inline, do místa použití funkce rovnou v programu napíšeme tělo funkce
nepřímá adresace	(**`COPY`**) umožňuje přístup k libovolně velké části paměti v závislosti na vstupu

Chybí rekurzivní volání funkcí, která lze implementovat pomocí cyklu `while` a zásobníku

Proměnné v programu pro RAM

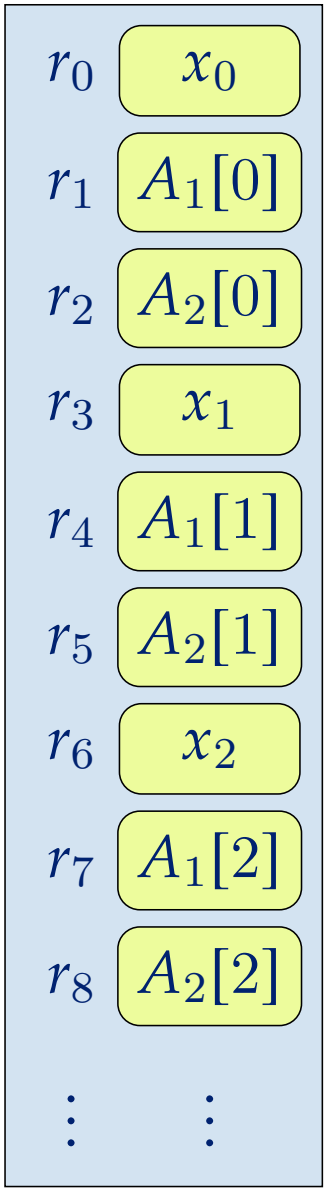
Předpokládejme, že v programu používáme pole $A_1, \ldots, A_p$ a skalární proměnné $x_0, \ldots, x_s$.

- Pole indexujeme přirozenými čísly (od 0)
- Prvek $A_i[j]$ umístíme do registru $r_{i+j*(p+1)}$
- Proměnnou x_i umístíme do registru $r_{i*(p+1)}$
- Prvky pole A_i jsou v registrech

$$r_i, r_{i+p+1}, r_{i+2(p+1)}, \ldots$$

- Skalární proměnné jsou v registrech

$$r_0, r_{p+1}, r_{2(p+1)}, \ldots$$



Jazyky rozhodnutelné RAMem

- Uvažme abecedu $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$.
- RAM R čte slovo $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n}$ jako posloupnost čísel $i_1, \ldots, i_n$ zakončenou 0
- RAM R přijme slovo w, pokud $R(w)\downarrow$ a první číslo, které R zapíše na výstup je 1
- RAM R odmítne slovo w, pokud $R(w)\downarrow$ a R buď na výstup nezapíše nic, nebo první zapsané číslo je jiné než 1
- Jazyk slov přijímaných RAMem R označíme pomocí $L(R)$
- (RAMem) částečně rozhodnutelný jazyk = přijímán nějakým RAMem
- (RAMem) rozhodnutelný jazyk = přijímán nějakým RAMem, který se zastaví pro každý vstup
  - každé slovo buď přijme, nebo odmítne

Funkce vyčíslitelné na RAMu

O RAMu R řekneme, že počítá částečnou aritmetickou funkci $f : \mathbb{N}^n \mapsto \mathbb{N}$, $n \geq 0$, pokud se vstupem $x_1, \ldots, x_n$ platí:

- Je-li $f(x_1, \ldots, x_n)\downarrow$, pak $R(x_1, \ldots, x_n)\downarrow$ a R vypíše na výstup hodnotu $f(x_1, \ldots, x_n)$.
- Je-li $f(x_1, \ldots, x_n)\uparrow$, pak $R(x_1, \ldots, x_n)\uparrow$.

RAM vyčíslitelná funkce počítaná nějakým RAMem

Řetězcové funkce vyčíslitelné na RAMu

RAM R počítá částečnou funkci $f : \Sigma^* \mapsto \Sigma^*$, kde $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, pokud platí:

- Vstupní řetězec $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_n}$ je předaný jako posloupnost čísel $i_1, \ldots, i_n$ ukončený 0
- Pokud je $f(w)\downarrow = \sigma_{j_1}\sigma_{j_2}\ldots\sigma_{j_m}$, pak $R(w)\downarrow$ a na výstup je zapsaná posloupnost čísel $j_1, j_2, \ldots, j_m, 0$
- Pokud $f(w)\uparrow$, pak $R(w)\uparrow$

RAM vyčíslitelná funkce počítaná nějakým RAMem

Turingův stroj $\longrightarrow$ RAM

Věta

Ke každému Turingovu stroji M existuje ekvivalentní RAM R.

- R simuluje práci M instrukci po instrukci
- R počítá touž funkci jako M
- R přijímá týž jazyk

Předpokládáme, že M má pásku neomezenou pouze doprava

- M nikdy nepohne hlavou nalevo od nejlevějšího symbolu vstupu
- Lze předpokládat bez újmy na obecnosti — každý TS lze převést do této podoby (viz cvičení)

Displej a konfigurace Turingova stroje

Displej dvojice (q, a), kde $q \in Q$ je aktuální stav a $a \in \Sigma$ je symbol pod hlavou

- Na základě displeje TS rozhoduje, jaký další krok má vykonat.

Konfigurace zachycuje stav výpočtu Turingova stroje a obsahuje

- stav řídící jednotky
- slovo na pásce
  - od nejlevějšího do nejpravějšího neprázdného políčka
- pozici hlavy na pásce
  - v rámci slova na pásce

Ekvivalentní RAM R musí ve své paměti reprezentovat konfiguraci a tím i displej M.

Technické předpoklady

$$M = (Q, \Sigma, \delta, q_0, F)$$

- $Q = \{q_0, q_1, \ldots, q_r\}$ pro nějaké $r \geq 0$, kde q_0 je počáteční stav
- $\Sigma = \{\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_s\}$ pro nějaké $s \geq 1$, kde $\sigma_0 = \lambda$ označuje znak prázdného políčka
- Nula ukončující vstup RAMu tedy odpovídá prázdnému políčku

Reprezentace konfigurace

obsah pásky je uložen v poli T

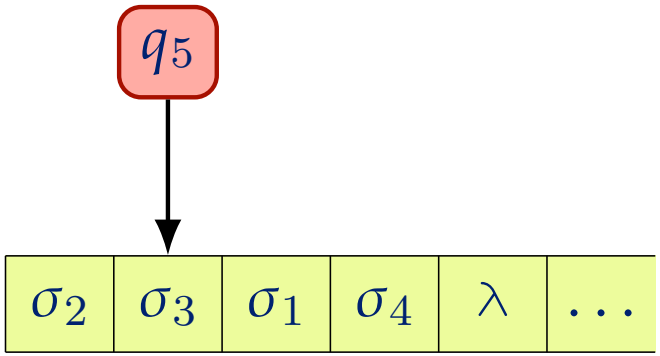
- Dle předpokladu je páska neomezená jen doprava
- $T[0]$ obsahuje první znak vstupu (po jeho načtení)

poloha hlavy v proměnné h

- $T[h]$ obsahuje symbol pod hlavou

stav v proměnné q

Konfigurace M



Reprezentace v R

$$q = 5$$

$$h = 1$$

$$T = \{2, 3, 1, 4, 0, \ldots\}$$

Algoritmus R

1. Načti vstup do pole T
   - Vstup je ukončený 0, která reprezentuje prázdné políčko λ
2. Polož $q = 0$, $h = 0$
3. Dokud $\delta(q, T[h])$ je definovaná, odsimuluj krok určený přechodovou funkcí
   - Přechodová funkce je uložená v programu R
   - Určení přechodu je provedeno posloupností podmíněných příkazů
   - Simulace kroku spočívá v aktualizaci $T[h]$, h a q dle instrukce
4. *Pokud nás zajímá přijetí slova*
   - je-li v q číslo přijímajícího stavu, zapiš 1 na výstup, jinak zapiš 0
5. *Pokud nás zajímá obsah pásky*
   - opiš obsah pásky na výstup

Přepis přechodové funkce do programu

Přechodová funkce M

q, c	$\rightarrow$	q', c', Z
q_0, σ_2	$\rightarrow$	q_3, σ_1, R
q_3, σ_1	$\rightarrow$	q_2, σ_0, L
q_2, σ_0	$\rightarrow$	q_0, σ_2, N

Odpovídající část programu R

if $q = 0$ **and** $T[h] = 2$ **then**
    $q \leftarrow 3$
    $T[h] \leftarrow 1$
    $h \leftarrow h + 1$
else if $q = 3$ **and** $T[h] = 1$ **then**
    $q \leftarrow 2$
    $T[h] \leftarrow 0$
    $h \leftarrow h - 1$
else if $q = 2$ **and** $T[h] = 0$ **then**
    $q \leftarrow 0$
    $T[h] \leftarrow 2$
else
    Konec simulace

RAM $\longrightarrow$ Turingův stroj

Věta

Ke každému RAMu R existuje ekvivalentní Turingův stroj M.

Obsah paměti R reprezentujeme na pásce M takto:

Jsou-li aktuálně využité registry $r_{i_1}, r_{i_2}, \ldots, r_{i_m}$, kde $i_1 < i_2 < \cdots < i_m$, pak je na pásce reprezentující paměť RAM R řetězec:

$$(i_1)_B|([r_{i_1}])_B\#(i_2)_B|([r_{i_2}])_B\#\ldots\#(i_m)_B|([r_{i_m}])_B$$

RAM $\longrightarrow$ Turingův stroj (struktura TS)

K RAMu R sestrojíme TS M jako 4-páskový.

Vstupní páska posloupnost čísel, která má dostat R na vstup

- Čísla jsou zakódovaná binárně a oddělená znakem $\#$
- Z této pásky M jen čte

Výstupní páska sem zapisuje M čísla, která R zapisuje na výstup

- Čísla jsou zakódovaná binárně a oddělená znakem $\#$
- Na tuto pásku M jen zapisuje

Paměť RAM obsah paměti stroje R

Pomocná páska pro výpočty součtu, rozdílu, nepřímých adres, posunu části paměťové pásky a podobně

RAM $\longrightarrow$ Turingův stroj (přechodová funkce)

- Číslo prováděné instrukce (pořadí v programu) je uloženo ve stavu
- Každá instrukce R je provedena řetězcem instrukcí M
- Většinou jde o
  - Nalezení registrů s operandy instrukce
  - Provedení aritmetických operací s operandy
  - Výpočet adres nepřímé adresace
  - Úprava paměti podle výsledku instrukce
    - Přidání nového registru do seznamu
    - Přepis obsahu některého registru
    - Může být nutné posunout obsah pásky s pamětí
- Následuje přechod do stavu, jímž začíná provádění další instrukce
- Pokud už další instrukce nenásleduje, simulace končí
  - Přijetí je dáno tím, jestli na výstupní pásku bylo zapsáno jen číslo 1
  - Toto je možné pamatovat si ve stavu