ELMÉLETI INFORMATIKA

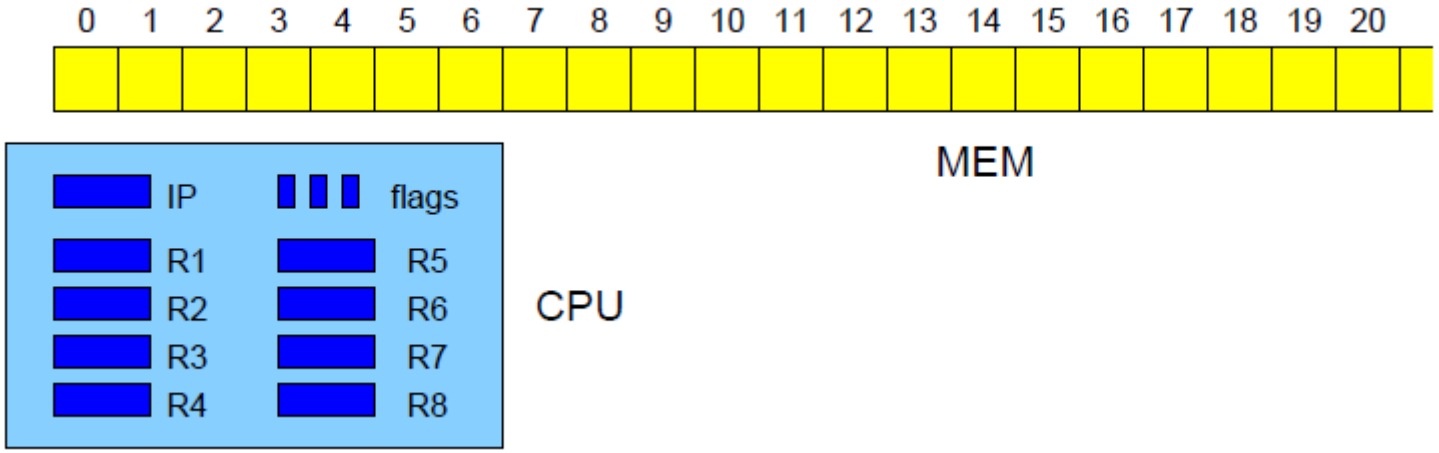
II. rész

Algoritmus- és kiszámíthatóságelmélet

Számítási modellek 2
RAM-gép

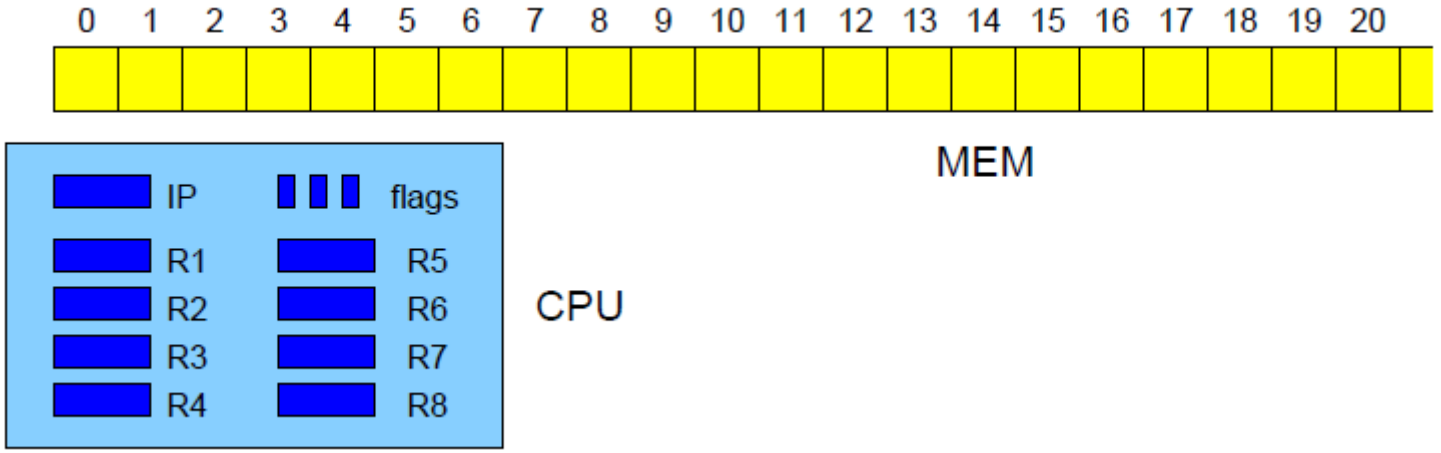
11. előadás

Számítógép architektúra



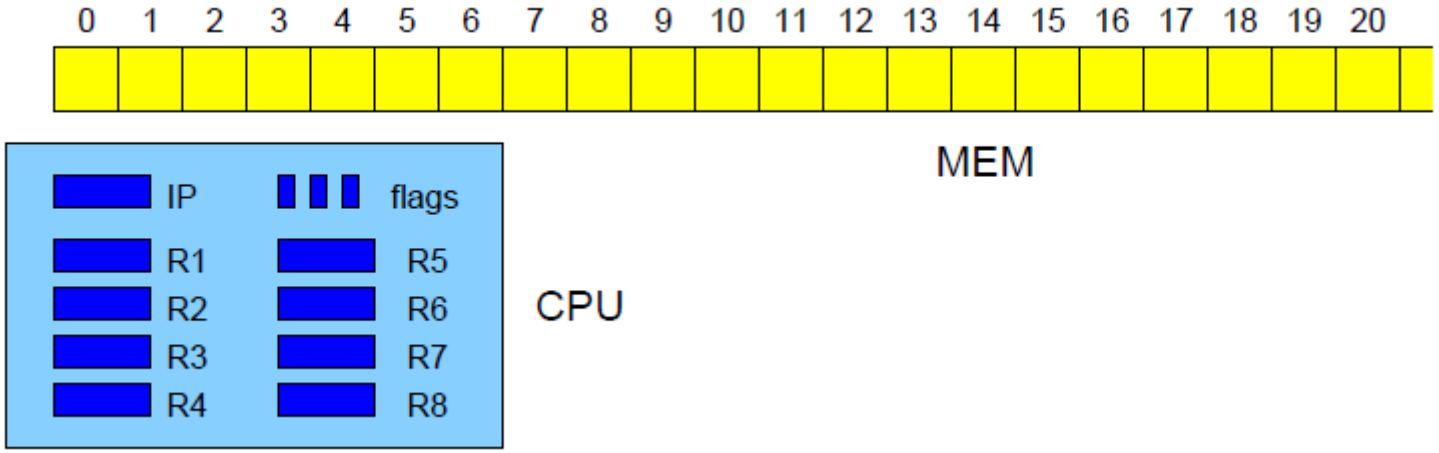
- A számítógép nagyszámú rekeszt tartalmazó **memóriá**val rendelkezik.
- Minden rekesz (regiszter) adott nagyságú számot tartalmazhat, pl. **1 byte**, azaz `0..255` közötti szám.
- Minden rekesznek saját száma van (**memóriacím**).

Számítógép architektúra



- A program utasításai a memóriában tárolódnak, (minden utasításnak saját **kód**ja van) és a **processzor** szekvenciálisan dolgozza fel azokat.
- A processzor utasításmutatót (**IP**) tartalmaz, amelyben az éppen végrehajtás alatt lévő utasítás címe található.
- A processzor az **IP** által megadott címről beolvassa az utasítást, megnöveli az **IP** értékét az épp beolvasott utasítás hosszával, majd végrehajtja az utasítást.

Számítógép architektúra



- A processzor tartalmaz néhány rögzített nagyságú regisztert (pl. 32 vagy 64 bit).
- A legtöbb művelet végrehajtása a regiszterekben történik.
- A processzor tartalmaz még néhány 1-bites speciális regisztert (**flag**), amelyek a processzor működését befolyásolják, illetve állapotát mutatják (pl. **OF** – túlcsordulást jelez, **ZF** – jelzi, hogy az utoljára végrehajtott művelet eredménye 0 volt).

Egy processzor utasításkészletének utasításai:

- adott memóriarekesz (vagy több egymás után következő memóriarekesz) tartalmának valamelyik regiszterbe való beolvasása (**LOAD**),
- adott regiszter tartalmának valamelyik memóriarekeszbe (vagy több egymás után következő memóriarekeszbe) való beírása (**STORE**),
- adott regiszter tartalmának másik regiszterbe való átírása (**MOV**),
- aritmetikai utasítások (**ADD**, **SUB**, **MUL**, **DIV**, **INC**, **DEC**, …),
- logikai utasítások (**AND**, **OR**, **XOR**, **NOT**, …),
- biteltolások (**SHL**, **SHR**, …),
- feltétel nélküli ugrás (**JUMP**),
- feltételes ugrások (**JZERO**, **JGTZ**),
- alprogramok (szubrutinok) hívása (**CALL**, **RET**),
- egyéb utasítások (I/O utasítások, memóriahozzáférés vezérlése).

11.1 példa: A $z := y + 2$ magasabb programozási nyelven leírt utasítás egy CPU utasításkészlete segítségével a következő módon adható meg:

```
LOAD  R2, [0x001b7c41]
ADD   R2,2
STORE [0x001b7c37], R2
```

miközben feltételezzük, hogy az y változó a `[0x001b7c41]` címen, a z változó pedig a `[0x001b7c37]` memóriacímen helyezkedik el.

A **RAM** (**Random Access Machine** = Közvetlen Elérésű Gép) a Turing-gépnél bonyolultabb, és a valódi számítógépekhez közelebb álló matematikai modell. Mint a neve is mutatja, a Turing-gépnél a memóriakezelés terén hatékonyabb, ugyanis itt a memóriarekeszeket nem szekvenciálisan lehet elérni, hanem közvetlenül.

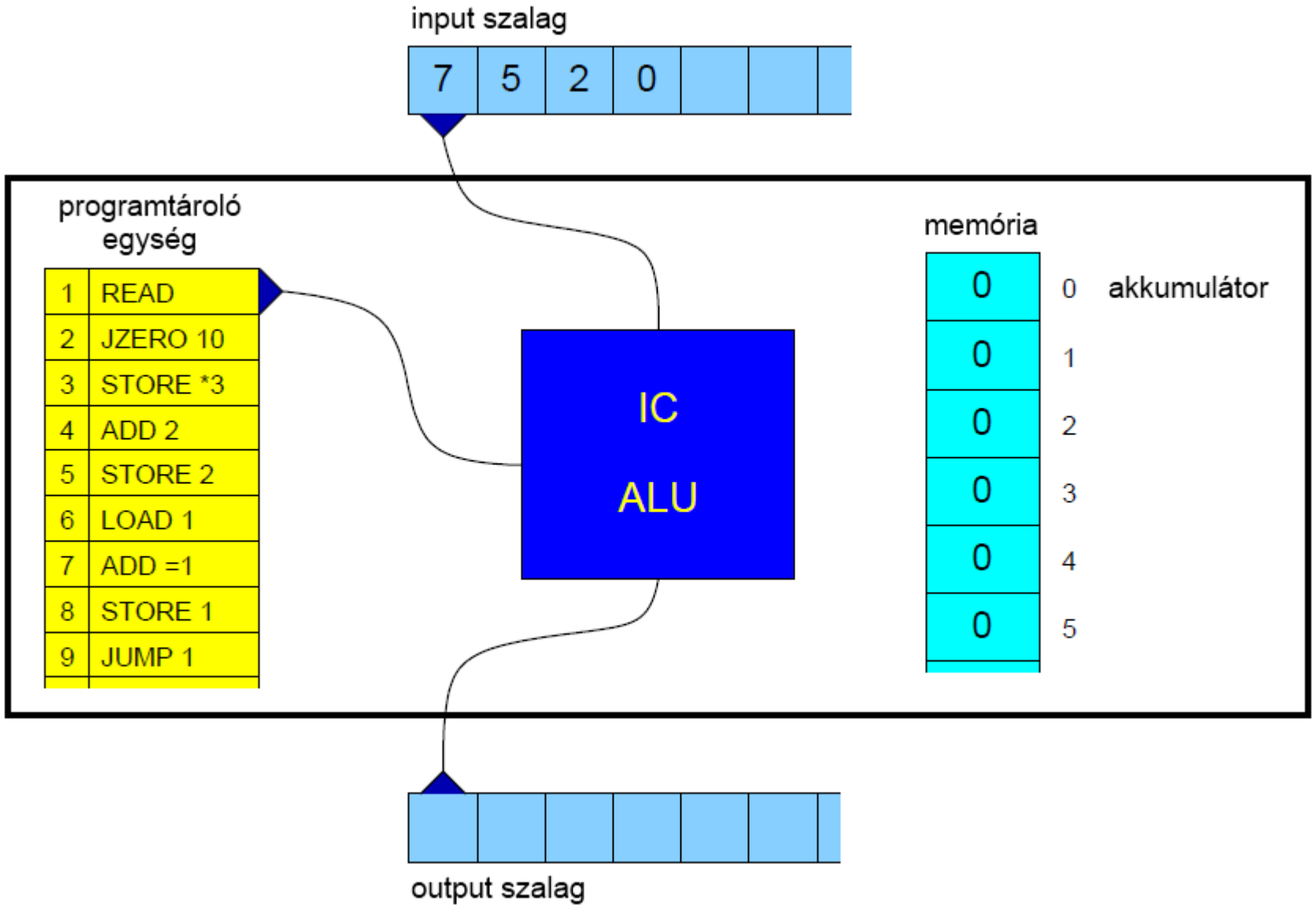
A RAM-gép architektúrája:

- **Programtároló egység** – tartalmazza a programot és az éppen végrehajtás alatt lévő utasítás mutatóját,
- **Munkamemória** – amelynek rekeszei (regiszterek) természetes számokkal vannak indexelve, a memóriarekeszek írhatók és olvashatók,
- **Input szalag** – csak olvasható,
- **Output szalag** – csak írható.

A **RAM**-gép egy idealizált gép, amely a valódi számítógépektől az alábbiakban tér el:

- a memória végtelen sok regisztert tartalmaz,
- egy regiszter mérete végtelen, azaz benne tetszőlegesen nagy egész számot tárolhatunk,
- a programtároló egység végtelen számú sort tartalmaz, ide véges hosszúságú programot írhatunk (minden sorba egy utasítást),
- az input szalag egy irányban végtelen hosszúságú,
- az output szalag egy irányban végtelen hosszúságú.

A RAM-gép architektúrája:



Aritmetikai utasítások	Adatmozgatásra való utasítások	Vezérlési utasítások
ADD *operandus*	LOAD *operandus*	JUMP *címke*
SUB *operandus*	STORE *operandus*	JGTZ *címke*
MULT *operandus*	READ *operandus*	JZERO *címke*
DIV *operandus*	WRITE *operandus*	HALT

Az **operandus** alakja és értéke:

$= i$ az i szám,

i az i-edik regiszter tartalma,

$* i$ az i-edik regiszterben lévő érték által meghatározott regiszter tartalma.

11.2 példa: Legyen $r[0] = -4$, $r[1] = 5$, $r[2] = 3$, $r[3] = 0$. Ekkor

az $=1$ operandus értéke 1,

az 1 operandus értéke 5,

a $*3$ operandus értéke -4.

Aritmetikai utasítások	Adatmozgatásra való utasítások	Vezérlési utasítások
ADD *operandus*	LOAD *operandus*	JUMP *címke*
SUB *operandus*	STORE *operandus*	JGTZ *címke*
MULT *operandus*	READ *operandus*	JZERO *címke*
DIV *operandus*	WRITE *operandus*	HALT

Az aritmetikai műveleteknél az első argumentum az akkumulátor ($r[0]$ regiszter) tartalma, a második argumentum pedig az *operandus* értéke. Az eredménye mindig az akkumulátorba kerül:

ADD *operandus* $\quad r[0] \leftarrow r[0] + operandus$

SUB *operandus* $\quad r[0] \leftarrow r[0] - operandus$

MULT *operandus* $\quad r[0] \leftarrow r[0] * operandus$

DIV *operandus* $\quad r[0] \leftarrow r[0]$ div $operandus$

Aritmetikai utasítások	Adatmozgatásra való utasítások	Vezérlési utasítások
ADD *operandus*	LOAD *operandus*	JUMP *címke*
SUB *operandus*	STORE *operandus*	JGTZ *címke*
MULT *operandus*	READ *operandus*	JZERO *címke*
DIV *operandus*	WRITE *operandus*	HALT

LOAD *operandus* $r[0] \leftarrow operandus$

STORE *operandus* $operandus \leftarrow r[0]$

READ x $r[x] \leftarrow$ input, és az olvasófej egy cellával jobbra mozog

WRITE x output $\leftarrow r[x]$, és az írófej egy cellával jobbra mozog

Aritmetikai utasítások	Adatmozgatásra való utasítások	Vezérlési utasítások
ADD *operandus*	LOAD *operandus*	JUMP *címke*
SUB *operandus*	STORE *operandus*	JGTZ *címke*
MULT *operandus*	READ *operandus*	JZERO *címke*
DIV *operandus*	WRITE *operandus*	HALT

JUMP *címke* — ugrás a megadott címkéjű utasításra

JGTZ *címke* — ha $r[0] > 0$, akkor ugrás a megadott címkéjű utasításra, egyébként a soron következő utasítás kerül végrehajtásra

JZERO *címke* — ha $r[0] = 0$, akkor ugrás a megadott címkéjű utasításra, egyébként a soron következő utasítás kerül végrehajtásra

HALT — megáll

11.3 példa:

LOAD $= 5$ az akkumulátorba beírja az 5 értéket,

LOAD 5 az akkumulátorba beírja az 5. regiszter tartalmát,

LOAD $* 5$ az akkumulátorba beírja az j-edik regiszter tartalmát, ahol $j = r[5]$.

11.4 példa:

INPUT két egész szám
OUTPUT a két egész szám összege

1	**READ** 0	$r[0] \leftarrow input$
2	**READ** 1	$r[1] \leftarrow input$
3	**ADD** 1	$r[0] \leftarrow r[0] + r[1]$
4	**WRITE** 0	$output \leftarrow r[0]$

11.5 példa:

INPUT egy n természetes szám
OUTPUT $n!$ a természetes szám faktoriálisa

1	`READ 1`	$r[1] \leftarrow input$
2	`LOAD 1`	$r[0] \leftarrow r[1]$
3	`JZERO 16`	ha $r[0] = 0$, akkor ugrás a **16**-re
4	`STORE 2`	$r[2] \leftarrow r[0]$
5	`STORE 3`	$r[3] \leftarrow r[0]$
6	`LOAD 2`	$r[0] \leftarrow r[2]$
7	`SUB =1`	$r[0] \leftarrow r[0] - 1$
8	`STORE 2`	$r[2] \leftarrow r[0]$
9	`JZERO 14`	ha $r[0] = 0$, akkor ugrás a **14**-re
10	`LOAD 3`	$r[0] \leftarrow r[3]$
11	`MULT 2`	$r[0] \leftarrow r[0] * r[2]$
12	`STORE 3`	$r[3] \leftarrow r[0]$
13	`JGTZ 6`	ha $r[0] \geq 0$, akkor ugrás a **6**-re
14	`WRITE 3`	$output \leftarrow r[3]$
15	`JZERO 1`	ha $r[0] = 0$, akkor ugrás a **1**-re
16	`HALT`	megáll

11.6 példa:

INPUT $w \in \{0,1\}^*$, a w szó után egy 2 szerepel a szalagon

OUTPUT a w szó tükörképe

```
1  LOAD =2        r[0] ← 2
2  STORE 1        r[1] ← r[0]
3  READ           r[0] ← input
4  SUB =2         r[0] ← r[0] − 2
5  JZERO 11       ha r[0] = 0, akkor ugrás a 11-re
6  ADD =2         r[0] ← r[0] + 2
7  STORE *1       r[r[1]] ← r[0]
8  LOAD 1         r[0] ← r[1]
9  ADD =1         r[0] ← r[0] + 1
10 JUMP 2         ugrás a 2-re
11 LOAD 1         r[0] ← r[1]
12 SUB =1         r[0] ← r[0] − 1
13 STORE 1        r[1] ← r[0]
14 LOAD 1         r[0] ← r[1]
15 SUB =1         r[0] ← r[0] − 1
16 JZERO 23       ha r[0] = 0, akkor ugrás a 23-ra
17 LOAD *1        r[0] ← r[r[1]]
18 WRITE          output ← r[0]
19 LOAD 1         r[0] ← r[1]
20 SUB =1         r[0] ← r[0] − 1
21 STORE 1        r[1] ← r[0]
22 JUMP 14        ugrás a 14-re
23 HALT           megáll
```

A RAM-gép idő és tárigénye

A RAM-gép mint számítási modell nagy előnye, hogy segítségével egyszerűen vizsgálható a számítások **idő-** és **tárigénye**. A RAM-gép idő- és tárigényét kétféleképpen is meghatározhatjuk.

Egyszerűbb esetben a program utasításainak végrehajtásához szükséges időt és tárat *egységesen azonos*nak tekintjük, ekkor **uniform bonyolultság**ról beszélünk. Egy program (algoritmus) uniform bonyolultságát az **időigény szempontjából** a végrehajtott utasítások száma, **tárigény szempontjából** pedig a felhasznált memóriaregiszterek száma adja meg.

A másik esetben a program utasításainak végrehajtásához szükséges időt és tárat *nem tekintjük azonosnak*, ehelyett egy utasítás költségét a benne szereplő adatok (regisztercímek, tartalmak) tízes számrendszerbeli számjegyeinek számát vesszük. Mivel ez az adott szám tízes alapú logaritmusa, ezért ekkor **logaritmikus bonyolultság**ról beszélünk.

Definiáljuk az $lcost(x)$ függvényt a következőképpen:

$$lcost(x) = \begin{cases} \lfloor \log|x| \rfloor + 1 & \text{ha } x \neq 0 \\ 1 & \text{ha } x = 0 \\ 0 & \text{ha } x \text{ végtelen} \end{cases}$$

11.7 példa: Határozzuk meg az **ADD** $*1$ utasítás időigényének uniform és logaritmikus bonyolultságát!

Uniform bonyolultság: 1

Logaritmikus bonyolultság: az **ADD** $*1$ utasítás hatására a következő fog történni: $r[0] \leftarrow r[0] + r[r[1]]$

Ezért az **ADD** $*1$ utasítás időigényének logaritmikus bonyolultsága:

$$lcost(r[0]) + lcost(1) + lcost(r[1]) + lcost(r[r[1]])$$

Segédtáblázat a logaritmikus bonyolultság meghatározásához:

Operandus `i`	Bonyolultság $t(i)$
`=i`	$lcost(i)$
`i`	$lcost(i) + lcost(r[i])$
`*i`	$lcost(i) + lcost(r[i]) + lcost(r[r[i]])$

Segédtáblázat a logaritmikus bonyolultság meghatározásához:

Utasítás	Bonyolultság
`LOAD x`	$t(x)$
`STORE x`	$lcost(r[0]) + lcost(x)$
`STORE *x`	$lcost(r[0]) + lcost(x) + lcost(r[x])$
`ADD x`	$lcost(r[0]) + t(x)$
`SUB x`	$lcost(r[0]) + t(x)$
`MULT x`	$lcost(r[0]) + t(x)$
`DIV x`	$lcost(r[0]) + t(x)$
`READ x`	$lcost(\text{input}) + lcost(x)$
`READ *x`	$lcost(\text{input}) + lcost(x) + lcost(r[x])$
`WRITE x`	$t(x)$
`JUMP e`	1
`JGTZ e`	$lcost(r[0])$
`JZERO e`	$lcost(r[0])$
`HALT`	1

11.8 példa: Határozzuk meg az alábbi RAM-gép program uniform és logaritmikus bonyolultságát, ha az input `2` és `10`.

1	`READ 0`	$r[0] \leftarrow input$
2	`READ 1`	$r[1] \leftarrow input$
3	`ADD 1`	$r[0] \leftarrow r[0] + r[1]$
4	`WRITE 0`	$output \leftarrow r[0]$

Uniform bonyolultság: 4

Logaritmikus bonyolultság: 12

1	`READ 0`	$lcost(2) + lcost(0) = 1 + 1 = \mathbf{2}$
2	`READ 1`	$lcost(10) + lcost(1) = 2 + 1 = \mathbf{3}$
3	`ADD 1`	$lcost(r[0]) + t(1) = lcost(2) + lcost(1) + lcost(r[1]) =$ $= lcost(2) + lcost(1) + lcost(10) = 1 + 1 + 2 = \mathbf{4}$
4	`WRITE 0`	$t(0) = lcost(0) + lcost(r[0]) = lcost(0) + lcost(12) = 1 + 2 = \mathbf{3}$

Operandus	Bonyolultság $t(i)$
`i`	$lcost(i) + lcost(r[i])$

Utasítás	Bonyolultság
`ADD x`	$lcost(r[0]) + t(x)$
`READ x`	$lcost(\text{input}) + lcost(x)$
`WRITE x`	$t(x)$

A RAM-gép és a Turing-gép ekvivalenciája

11.1 tétel: Tetszőleges M Turing-gép szimulálható olyan RAM-gép programmal, amelynek uniform bonyolultsága $O\left(T_M(n)\right)$, logaritmikus bonyolultsága pedig $O(T_M(n)\log T_M(n))$.

11.2 tétel: Egy $t(n)$ logaritmikus bonyolultságú, **`MULT`** és **`DIV`** utasításokat nem tartalmazó RAM-gép program szimulálható olyan M Turing-géppel, amelynek maximális időigénye $T_M(n) = O(t(n)^2)$.

11.3 tétel: Tetszőleges $t(n)$ logaritmikus bonyolultságú RAM-gép program szimulálható olyan M Turing-géppel, amelynek maximális időigénye $T_M(n) = O(t(n)^3)$.

A **RAM** szimulátor letölthető az alábbi címről:

http://www.szkup.com/?pid=msthesis&lang=en