

HARMADIK GYAKORLAT

HYMN SZIMULÁTOR

Ebben a feladatban a következőket fogjuk áttekinteni:

- Neumann rendszerű számítógép felépítése, működése.
- Egyszerű gépi kódú és assembler programok készítése számítógép szimulátoron

A feladat megoldása hozzávetőlegesen 60 percet vesz igénybe.

PROGRAMOZÁSI FELADAT

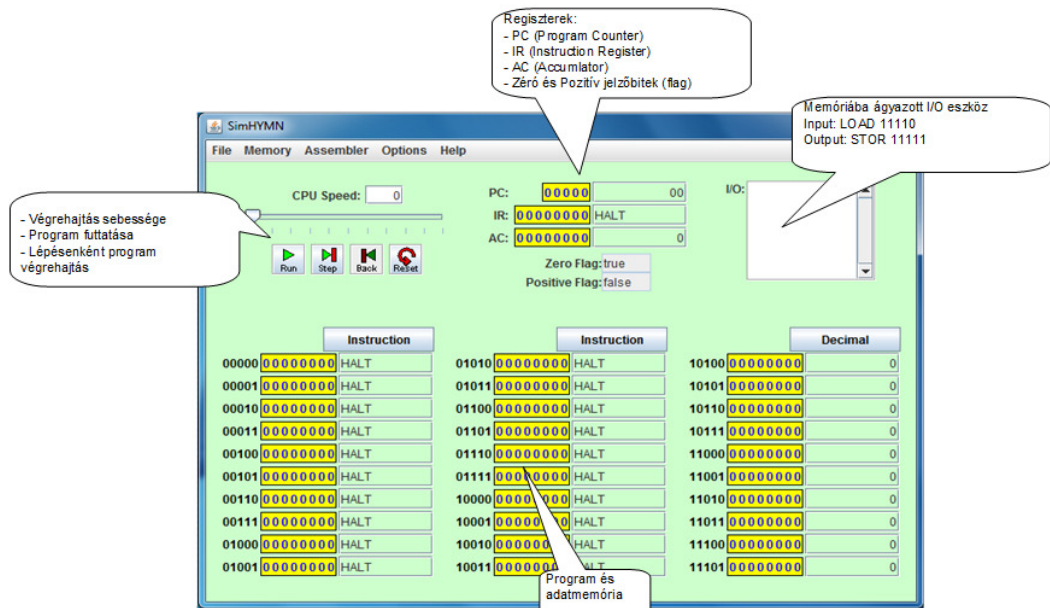
A feladat célja, hogy pár, egyszerű program elkészítésén keresztül jobban megismerjük a Neumann architektúrájú számítógép működésének alapvető mechanizmusait. Programjainkat egy leegyszerűsített számítógép modellen (szimulátoron) fogjuk elkészíteni.

Az interneten több ilyen oktatási célú számítógép szimulátor is elérhető, többek között:

- **Hymn CPU szimulátor:** <http://ozark.hendrix.edu/~burch/socs/hymn/>
- **Little Man Computer (LMC)** <http://mathtt.me/projects/lmc/>

A feladatsorban szereplő programok a Hymn szimulátorra készültek.

A HYMN SZIMULÁTOR FELÉPÍTÉSE



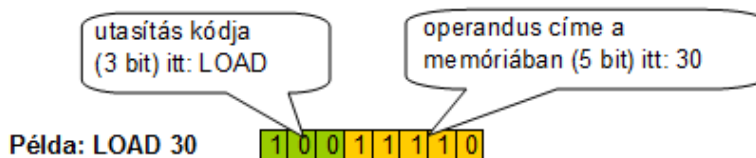
A HYMN cpu szimulátor kezelőfelülete

A szimulátor „műszaki paraméterei” a következők:

- 8 utasítás
- 30 byte operatív tár (program és adat memória)
- Két darab speciális célú regiszter (PC és IR)
- Egy darab általános célú regiszter (AC)
- Memóriába ágyazott I/O egység (30-as címen az input, 31-es címen az output)

UTASÍTÁSKÉSZLET

A Hymn cpu szimulátor gépi utasításai egységes felépítésűek:



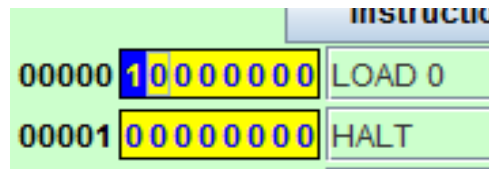
Példa egy utasítás felépítésére (LOAD 30)

Kód	Mnemonik	Leírás
000	HALT	nem történik semmi
001	JUMP	$PC := \text{operandus}$ (ugrás az operandus címre)
010	JZER	ha $AC = 0$ akkor $PC := \text{data}$ (ugrás az op. címre) egyébként pedig $PC := PC + 1$ (következő utasítás végrehajtása)
011	JPOS	ha $AC > 0$ akkor $PC := \text{data}$ (ugrás az op. címre) egyébként pedig $PC := PC + 1$ (következő utasítás végrehajtása)
100	LOAD	$AC := M[\text{data}]$; $PC := PC + 1$ (az operandus által megcímezett memória tartalmának betöltése AC-be)
101	STOR	$M[\text{data}] := AC$; $PC := PC + 1$ (AC tartalmának kiírása az operandussal által megcímezett memória cellába)
110	ADD	$AC := AC + M[\text{data}]$; $PC := PC + 1$ (AC regiszter tartalmához hozzáadja az operandussal megcímezett memória tartalmát)
111	SUB	$AC := AC - M[\text{data}]$; $PC := PC + 1$ (AC regiszter tartalmából kivonja az operandussal megcímezett memória tartalmát)

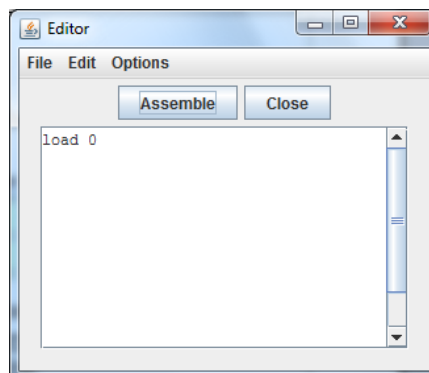
A HYMN utasításkészlete

UTASÍTÁSOK BEVITELE

- Egérkattintással – kattintsunk a memóriacellákon belül a megfelelő bitekre, ha módosítani szeretnénk azok értékét.



- Assembler editorból – kattintsunk az „Assembler” menü „Show editor” pontjára, majd gépeljük be a program utasításait. Az így elkészített programot az „Assemble” nyomógombbal tölthetjük be a memóriába.



Az assembler editor használatának további előnye, hogy a programkódban megjegyzéseket és címkéket (memória címre mutató hivatkozásokat) is elhelyezhetünk.



Megjegyzések és címkék használata

ELSŐ PROGRAM

Első programunk olyan egyszerű, hogy semmilyen számítási műveletet nem fog végezni, egyszerűen csak két cím között fog „ugrálni” a végrehajtás.

Nyissuk meg az assembler editor ablakot (Assembler -> Show Editor), majd gépeljük be a következő két sort:

```

JUMP 1      # ugrás az 1-es (00001) címre
JUMP 0      # ugrás az 0-ás (00000) címre

```

A kód bevitele után nyomjuk meg az Assemble nyomógombot. Ha visszatérünk a főablakba, akkor a következő memória tartalmat kell látnunk:

Instruction				
00000	00100001	JUMP 1		01
00001	00100000	JUMP 0		01
00010	00000000	HALT		01
00011	00000000	HALT		01

A „step” nyomógombbal futtassuk lépésenként a programunkat és figyeljük meg a PC és IR regiszterek tartalmának változását.

A PC regiszter mindig a végrehajtandó utasítás címét tartalmazza, míg az IR magát az utasítás kódját.

MÁSODIK PROGRAM

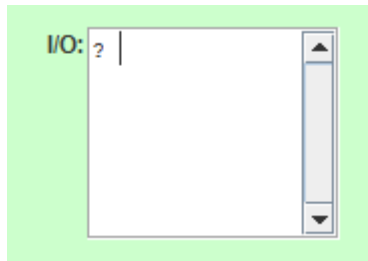
Második lépésben ismerkedjünk meg a be és kiviteli (I/O) lehetőségekkel. Mivel ez a számítógép szimulátor memóriába ágyazott I/O egységet tartalmaz, ezért az I/O műveletek nem különböznek a memóriából olvasás (LOAD) és memóriába írás (STOR) műveletektől.

Nyissuk meg az assembler editor ablakot (Assembler -> Show Editor), majd gépeljük be a következő programot:

```
LOAD 30    # Az input-ról érkező adatot betöltjük
            # az AC regiszterbe

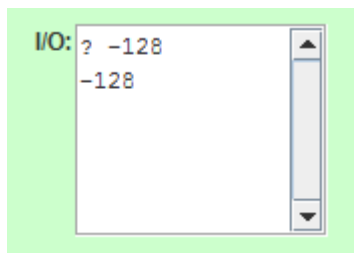
STOR 31    # Az AC regiszterben lévő adatot kiírjuk az
            # output-ra
```

A kód bevitele után nyomjuk meg az Assemble nyomógombot és futtassuk a programunkat.



Az I/O ablakban megjelenő kérdőjel jelzi, hogy a programunk bemenő értékre vár.

Írjunk be egy számot [-128..127] tartományban és üssünk [ENTER] billentyűt. Figyeljük meg, hogy a beírt szám megjelenik az AC regiszterben, majd a következő utasítás végrehajtásakor ugyanez az érték kiíródik a kimenetre



HARMADIK PROGRAM

Miután megismerkedtünk egy alapvető vezérlő utasítással (JUMP) és a I/O kezelésére (valójában memória olvasásra és írásra) szolgáló utasításokkal (LOAD, STOR), ismerkedjünk meg egy aritmetikai utasítással – az összeadással.

Nyissuk meg az assembler editor ablakot (Assembler -> Show Editor), majd gépeljük be a következő programot:

```
LOAD 30      # Az input-ról érkező adatot betöltjük
              # az AC regiszterbe

ADD 30        # Az AC regiszter tartalmához hozzáadjuk a
              # 30-as címről (input-ról) érkező második
              # értéket,

STOR 31       # majd az AC regiszterben lévő adatot kiírjuk
              # az output-ra
```

A kód bevitele után nyomjuk meg az Assemble nyomógombot és futtassuk a programunkat. A fenti program egy egyszerű „számológép”. Bekér két számot, majd kiírja az összegüket. Figyeljünk arra, hogy itt is 8 bites előjeles számokat kezel a program [-128..127].

A LOAD 30 utasítást a READ, a STOR 31-et pedig a WRITE paranccsal is lehet helyettesíteni.

```
LOAD a
WRITE
READ
ADD a
HALT
a:12
```

Készítsünk az előzőhöz hasonló, csak a kivonás műveletet végző programot.

NEGYEDIK PROGRAM

Az ADD (összeadás) és a SUB (kivonás) művelet segítségével készítsünk programot a $12 - 23 + 31$ műveletsor elvégzésére.

```
LOAD a      # Az "a" memóriacímen lévő adat betöltése az
            # AC regiszterbe.

SUB b        # Az AC regiszter tartalmából kivonjuk a
            # "b" memóriacímen lévő adat értékét.

ADD c        # Az AC regiszter tartalmához hozzáadjuk a "c"
            # címen lévő adat értékét.

WRITE       # Az AC regiszter értékét kiírjuk az outputra.
```

```

HALT
a: 12
b: 23
c: 31

```

Írja át úgy az előző programot, hogy az a, b és c érték az inputról érkezzen!

ÖTÖDIK PROGRAM

Feladat: Írjunk olyan programot, ami bekér az inputról egy számot, majd megvizsgálja annak előjelét: ha pozitív szám érkezik, akkor +1-et, ha negatív, akkor -1-et, nulla esetén pedig 0-t ír ki.

Értelmezze, és futtassa le soronként a következő programot! Nézze meg, hogy hogyan változnak a flagek a futás során!

```

READ                # Adat beolvasása az inputról az AC regiszterbe.
JZER kiir           # A kiir címre ugrás, ha az AC regiszter értéke 0.
JPOS pluszegybe     # A pluszegybe címre ugrás, ha az AC regiszterben po-
                    # zítív szám van.
LOAD minegy         # A -1 betöltése az AC regiszterbe.
JUMP kiir           # A +1 betöltése az AC regiszterbe.
pluszegybe:
LOAD pluszegy
kiir:
WRITE              # Az AC regiszter tartalmának kiírása az outputra.
HALT
minegy: -1
pluszegy: 1

```

Zárásként újból hangsúlyozzuk, hogy a fejezet célja nem a programozás megtanítása volt, hanem a számítógép belső működésének a bemutatása néhány egyszerű program segítségével. A programozással a következő félévben a Számítási módszerek tárgy keretein belül fogunk részletesebben foglalkozni.

PLUSZ FELADATOK

- Készítsünk programot, ami kiírja a születési dátumunkat, tipp - használjunk címkéket a dátum karaktereinek címezéséhez. (JPOS, JZER)
- Készítsük el az $A + \text{abs}(B)$ műveletet megvalósító programot (használjuk a JPOS utasítást)
- Írjunk programot $A * B$ kiszámítására (mivel nincs szorzás művelet, vezessük vissza a feladatot összeadásra)

