ARM processzor (Advanced RISC Machines)

Ebben a fejezetben megmutatjuk, hogyan kell közvetlenül egy mikroprocesszorra programozni.

Az ARM processzort választottuk, hiszen nagy valószínűséggel olyan mobiltelefonunk van ami ezt a típust használja. Nem törekszünk teljességre, csak az adatmozgatás, összehasonlítás, indirekt memória elérés és a verem alaphasználatát mutatjuk be.

Ezen a linken találunk egy teljes értékű ARM CPU szimulátort: https://cpulator.01xz.net/?sys=arm , a továbbiakban ezt használjuk a próbáink során.

Összeadás

Rögtön készítsük el az első programunkat!

Másoljuk be a következő program szöveget a weblapon középen elhelyezkedő szövegmezőbe:

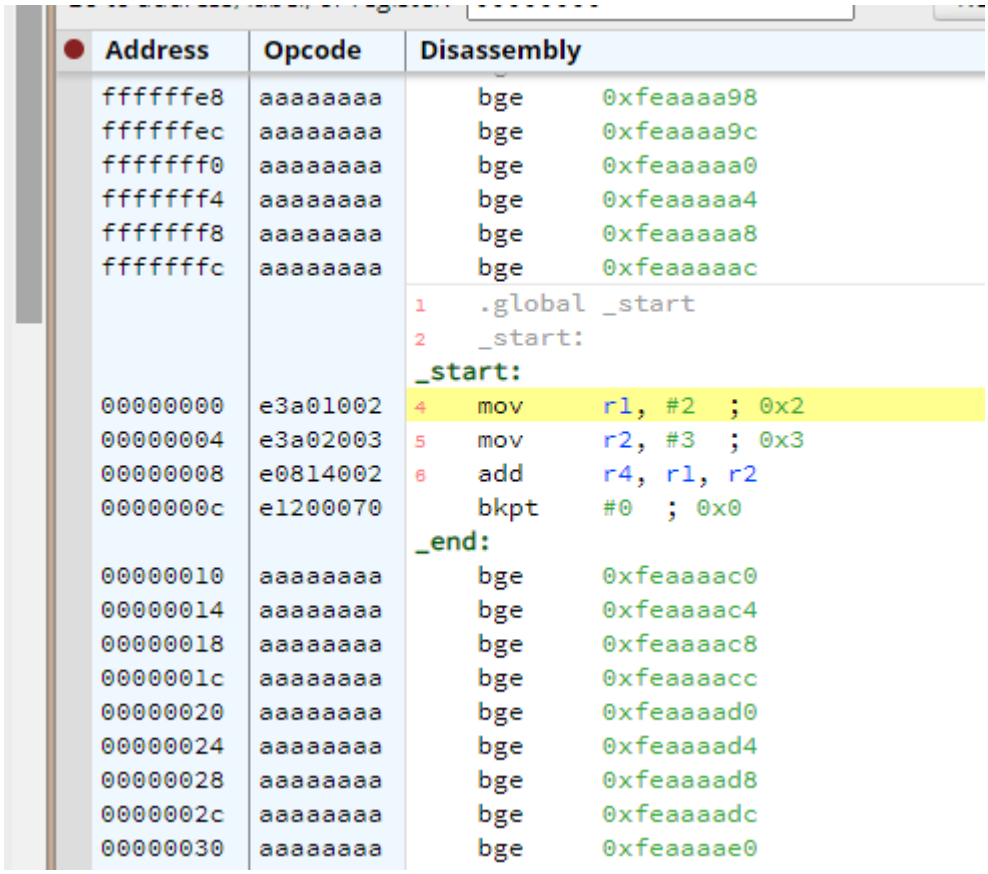
```
.global _start
_start:

 mov r1, #2
 mov r2, #3
 add r4, r1, r2

 bkpt
```

A fenti program kiszámítja a 2 + 3 összeadás értékét. Először az r1 regiszterbe másolja a 2-t (mov r1, #1), majd r2-be a 3-mat (mov r2, #3), majd r4-be helyezi az r1 és r2 összegét.

A “Compile and Load (F5)” feliratú gombot megnyomva a szöveges assembly program lefordul és a bináris változata betöltődik a memória 00000000 címére:



Address	Opcode	Disassembly
fffffffe8	aaaaaaaa	bge 0xfeaaaa98
fffffffec	aaaaaaaa	bge 0xfeaaaa9c
fffffff0	aaaaaaaa	bge 0xfeaaaaa0
fffffff4	aaaaaaaa	bge 0xfeaaaaa4
fffffff8	aaaaaaaa	bge 0xfeaaaaa8
fffffffc	aaaaaaaa	bge 0xfeaaaaac
		1 .global _start
		2 _start:
		_start:
00000000	e3a01002	4 mov r1, #2 ; 0x2
00000004	e3a02003	5 mov r2, #3 ; 0x3
00000008	e0814002	6 add r4, r1, r2
0000000c	e1200070	bkpt #0 ; 0x0
		_end:
00000010	aaaaaaaa	bge 0xfeaaaac0
00000014	aaaaaaaa	bge 0xfeaaaac4
00000018	aaaaaaaa	bge 0xfeaaaac8
0000001c	aaaaaaaa	bge 0xfeaaaacc
00000020	aaaaaaaa	bge 0xfeaaaad0
00000024	aaaaaaaa	bge 0xfeaaaad4
00000028	aaaaaaaa	bge 0xfeaaaad8
0000002c	aaaaaaaa	bge 0xfeaaaadc
00000030	aaaaaaaa	bge 0xfeaaaae0

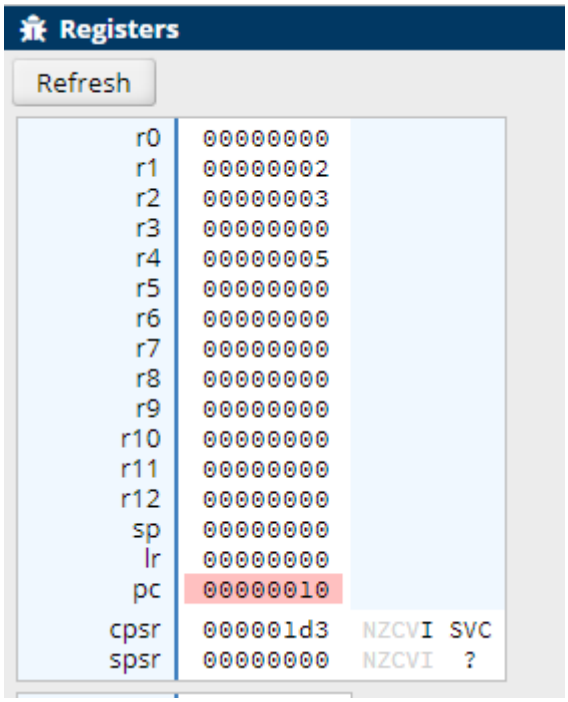
Az **Address** oszlop a memória címet mutatja, az **opcode** a gépi kódu utasítást. A fenti programkód gépi kódú alakja a következő:

e3a01002e3a02003e0814002e1200070

Ez a 16 byte jelenik meg a memóriában a nullás címtől kezdve a szimulátorban. Az eredeti forráskódot a **Disassembly** oszlopban piros kicsi számokkal jelölve látjuk. Rögtön észrevehető, hogy vannak olyan jelölések pl: _start, aminek nincs konkrét gépi kódú megfelelője.

A **Disassembly** visszafordítást jelent, a rendszer megpróbálja visszaalakítani a gépi kódú utasításokat assembly nyelvű szöveggé.

A sárgával kijelölt sor mutatja, hogy az utasításszámláló melyik memóriacímet fogja végrehajtani. Nyomjuk meg többször a F2 billenytyűt, ami a soronkénti végrehajtást jelenti, közben figyeljük meg a regiszterek változását a bal oldalon (pirossal kiemeli a szimulátor a változásokat). A **bkpt** parancsig elég elmenni.



Jól látható, hogy a megfelelő regiszterekben megjelentek a számok, és az eredmény is a r4-esben.

Összehasonlítás

Az előző ábrán látható a **Current Program Status Register (CPSR)** regiszter legalul (ez az állapot regiszter). Ennek egyes bitjei jelölik azt, hgy egy művelet eredménye nulla, negatív, átvitelt eredményez, stb. Nem kell tudni, hogy a 32 bitből ezek melyek pontosan, a mellette megjelenő **NZCVI** szöveg egyes betűi jelölik a főbb állapotbiteket.

Vátsunk vissza a forráskód szerkesztő fűlre: ctrl+e, vagy alul rákattintva, majd másoljuk be az alábbi kódot a szerkesztőbe, majd fordísuk és kövessük nyomon:

```
.global main

_start:

mov     r0, #2     /* r0 = 2 */
cmp     r0, #3     /* r0 - 3. Negatív bit egyre fog váltani */
addlt   r0, r0, #5 /* lt => less than. Ha (r0 < 3) akkor adj hozzá 5-öt */
cmp     r0, #3     /* r0 - 3. Zero bit 1 lesz. Negative bit 0 lesz */
addlt   r0, r0, #5 /* Ha (r0 < 3) akkor adj hozzá 1-et az r0-hoz */

bkpt
```

A 2. sorban a **cmp** (**com**pare) összehasonlítja az argumentumait. Utána az **addlt** csak akkor ad hozzá 5-öt az r1 hez ha (less than) feltételnek megfelelően az alábbi táblázatban N!=V azaz a N és V kapcsolók értéke nem egyezik meg.

Feltétel kód	Jelentés	Státusz bitek
EQ	Equal	Z==1
NE	Not Equal	Z==0

GT	Signed Greater Than	(Z==0) és (N==V)
LT	Signed Less Than	N!=V
GE	Signed Greater Than or Equal	N==V
LE	Signed Less Than or Equal	(Z==1) vagy (N!=V)
MI	Negative (or Minus)	N==1
PL	Positive (or Plus)	N==0

Elágazások

```
.global main

_start:

  mov     r4, #10
loop_label:
  sub     r4, r4, #1
  cmp     r4, #0
  bne     loop_label
```

A fenti kódrészlet 10-től 0-ig számol visszafelé, az r1 regiszterben tárolja a 10-et kezdetben, majd lépésenként csökkenti 1-el. **sub r4, r4, #1** jelentése r4 = r4 - 1, azaz mentsd el az r4 aktuális értékét mínusz 1-et az r4 ben. * A cmp összehasonlítja 0-val a r4-et, a **bne** (branch if not equals) ágazz el ha nem egyenlő.

Memória kezelés

```
.data           /* .data szekció dinamikusan jön létre, előre nehéz
megmondani, hogy hol fog elhelyezkedni */
var1: .word 3  /* var1 változó */
var2: .word 4  /* var2 változó */

.text           /* kód kezdete */
.global _start

_start:
    ldr r0, adr_var1  @ töltsük be a var1 memória címét az r0-be
    ldr r1, adr_var2  @ töltsük be a var2 memória címét az r1-be
    ldr r2, [r0]      @ r2 be töltsük be az r0-regiszter által mutatott
címen lévő értéket
    str r2, [r1]      @ r2 értékét írjuk ki az r1 által mutatott memória
címre
    bkpt

adr_var1: .word var1  /* a var1 címe lesz letárolva itt */
adr_var2: .word var2  /* a var2 címe lesz letárolva itt */
```

A példában látható hogyan lehet változókat tárolni és a memóriacímeket ahol el vannak tárolva,

hogyan lehet módosítani, illetve betölteni a regiszterekbe tartalmukat. A legutolsó utasítás felülírja adr_var2 helyen tárolt értéket, ezt a módosulást a harmadik memory fülön tudjuk ellenőrizni.

Verem

A verem adattárolásra szolgál, memóriacímét az **sp** regiszter mutatja és új érték verembe helyezésekor az sp regiszter csökken 4 byteot.

```
.global _start

_start:
     mov   r0, #2  /* r0 = 2 */
     push  {r0}    /* r0 értékének tárolása a veremben */
     mov   r0, #3  /* r0 = 3 */
     pop   {r0}    /* r0 korábbi értékének visszatöltése a veremből. */

bkpt
```

A push utasítás után a “memory” fülön megnézhetjük hogy a 000000 memóriacím elé 4 byte-al kezdődően beíródott a 2 egy 32 bites egészként.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:szamitastechnika:arm_assembly_alapok?rev=1662411997

Last update: **2022/09/05 21:06**