

Mikrokontroller II.

Kékesi Kristóf
NEPTUN kód: ZI6I4M
Mérőpár: Bor Gergő

Mérés ideje: 2024.05.15. 15:15-18:00

Mérés helye: Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
1083 Budapest, Práter utca 50/A 421-es labor
kekesi.kristof.mihaly@hallgato.ppke.hu

Kivonat—

A jegyzőkönyv részletesen leírja a május 15-én megoldandó mérési feladatokat, valamint az ezek megoldásához szükséges információkat. A dokumentum célja, hogy átfogó útmutatást nyújtson a feladatok megoldásának folyamatáról és a reprodukálhatósághoz szükséges lépésekről.

A jegyzőkönyv részletesen ismerteti az egyes feladatok megoldásához szükséges lépéseket, beleértve a szükséges eszközök és eljárások használatát is. Ezáltal segíti az azt olvasókat a feladatok hatékony és pontos megoldásában, valamint elősegíti a feladatok reprodukálhatóságát és értelmezhetőségét.

Keywords-Mikrokontroller; Assembly; Regiszterek; Műveletek; Számrendszerek; Számábrázolás;

MÉRÉSEL KAPCSOLATOS FOGALMAK

- **Számrendszer:** Olyan jelölési rendszer, amelyet a számok írására és az aritmetikai műveletek elvégzésére használnak. Alapvetően egy adott alapszám köré épül, ami a számrendszer alapja, amely meghatározza a rendszerben felhasználható szimbólumok (számjegyek) számát és a helyiérték szerinti szorzót. A legismertebb számrendszer a decimális, vagy tízes számrendszer, amely 10 különböző számjegyet használ (0-tól 9-ig).

Különböző kultúrák és számítógépes alkalmazások különböző számrendszereket használhatnak. Például a számítástechnikában gyakran alkalmazzák a bináris (2-es alapú), az oktális (8-as alapú) és a hexadecimális (16-os alapú) számrendszereket, mivel ezek hatékonyan modellezhetik a digitális áramkörök működését.

$$101,01_{[10]} = 1 \cdot 10^2 + 0 \cdot 10^1 + 1 \cdot 10^0 + 0 \cdot 10^{-1} + 1 \cdot 10^{-2} \quad (1)$$

- **Kettes komplementes számábrázolási módszer:** A kettes komplementes módszer a negatív számok bináris ábrázolására szolgál. Ez a módszer lehetővé teszi a bináris összeadás használatát mind pozitív, mind negatív számok esetében anélkül, hogy külön figyelemmel kellene kísérni a szám előjelét. Egy adott bites szélességű szám kettes komplementer alakjának meghatározása a következő lépésekből áll, ha a szám negatív:

- 1) Hozzáadunk a számhoz egyet;
- 2) Az így kapott szám abszolútértékét vesszük;
- 3) Felírjuk binárisan a kapott számot, előre definiált biten;
- 4) Minden bitet negálunk.

[1]

- **Fix pontos ábrázolás:** Előre definiált pontosvessző hely alapján tudjuk, hogy a . képletben a hatványok kitevőit mennyivel toljuk el.

- **Assembly programozási nyelv:** Egy alacsony szintű, gépi kódhoz közel álló nyelv, melyet az adott processzor architektúrájának utasításkészletével írnak. Az assembly programozási nyelv lehetővé teszi a programozók számára, hogy közvetlenül befolyásolják a processzor működését, így nagyfokú kontrollt biztosítanak az alkalmazások felett. Általában a gépi kódhoz legközelebb álló emberi érthető formában íródik, és közvetlenül fordítható gépi kóddá. Mivel az assembly nyelv közvetlenül kommunikál a hardverrel, ezáltal nagy teljesítményt és precizitást biztosít, azonban általában bonyolultabb és kevésbé átlátható kódot eredményez, mint a magasabb szintű programozási nyelvek. [2]

- **Regiszterek:** Olyan kis méretű adatokat tároló hardveres komponensek, amelyek közvetlenül kapcsolódnak a processzorhoz. Ezek a tárolók rendkívül gyors hozzáférést tesznek lehetővé a processzor számára az adatokhoz és utasításokhoz. A regisztereknek különböző típusai vannak, beleértve az általános célú regisztereket, az adatregisztereket, az indexregisztereket és a vezérlőregisztereket. Ezek a regiszterek játszanak kulcsfontosságú szerepet az assembly nyelvben írt programokban, mivel közvetlenül manipulálhatók az alacsony szintű utasításokon keresztül, lehetővé téve a programok számára a hatékony adatmanipulációt és vezérlést.

Az egyes regiszterek általában a processzor architektúrájától függően vannak elnevezve, és ezek elnevezése a processzor tervezésétől és az adott architektúra konvencióitól függ. A regiszterek elnevezése gyakran követi egy adott architektúra belső működését és funkcióit. Az x86 architektúrában a regiszterek elnevezése a következő típusok szerint csoportosítható:

- Általános célú regiszterek: Például az EAX, EBX, ECX, EDX regiszterek.
- Index regiszterek: Például az ESI, EDI regiszterek.
- Adatregiszterek: Például az AL, AH, BL, BH regiszterek (byte regiszterek), valamint az AX, BX, CX, DX regiszterek (word regiszterek).
- Pontosító regiszterek: Például az EFLAGS regiszter.

Más architektúrák esetében más elnevezési konvenciókat használnak, például az ARM architektúra regiszterei különböző típusokra oszlanak. Az elnevezési konvenciók változhatnak az architektúrától és a processzorgyártótól függően. [3]

- **Műveletek Assembly-ben:**

- MOV: Egy megadott regiszter értékét másoljuk át

egy másik megadott regiszterbe.

$$\text{src} \rightarrow \text{dst}$$

- ADD: Egy megadott regiszter értékét hozzáadjuk egy másik megadott regiszter értékéhez. Másnéven az összeadás művelet. A C++ nyelvben az ehhez leghasonlóbb a += operátor.

$$\text{src} + \text{dst} \rightarrow \text{dst}$$

- ADDC: Az "összeadás cipeléssel" (add with carry) műveletet végzi el. Ez az utasítás hasonló az egyszerű ADD utasításhoz, azonban a CARRY (cipelés) állapotot is figyelembe veszi. A CARRY egy speciális jelzőbit a processzorban, amely jelzi, hogy egy előző aritmetikai művelet során az eredmény túlsordult-e (overflow), vagyis több bitet igényel, mint amennyi a célregiszterben elfér. Az ADDC utasítás két operandust ad össze, valamint figyelembe veszi a CARRY jelzőbitet is. Ha a CARRY be van állítva (1), akkor az ADDC az operandusokat összeadja, valamint az egyesek helyiértékén levő cipelést is figyelembe veszi. Ha a CARRY nem aktív (0), akkor az ADDC ugyanúgy működik, mint az ADD utasítás.

$$\text{src} + \text{dst} + C \rightarrow \text{dst}$$

- SUB: Assembly programozási nyelvben a "kivonás" (subtract) műveletet valósítja meg. Ez az utasítás lehetővé teszi két operandus különbségének kiszámítását.

$$\text{dst} + \neg \text{src} + 1 \rightarrow \text{dst}$$

- SUBC: Assembly programozási nyelvben a "kivonás cipeléssel" (subtract with carry) műveletet valósítja meg. Ez az utasítás hasonló az egyszerű SUB utasításhoz, viszont a CARRY (cipelés) állapotot is figyelembe veszi. A CARRY egy speciális jelzőbit a processzorban, amely jelzi, hogy egy előző aritmetikai művelet során az eredmény túlsordult-e (overflow), vagyis több bitet igényel, mint amennyi a célregiszterben elfér. A SUBC utasítás két operandust von ki egymásból, valamint figyelembe veszi a CARRY jelzőbitet is. Ha a CARRY be van állítva (1), akkor az SUBC az operandusokat kivonja, valamint az egyesek helyiértékén levő cipelést is figyelembe veszi. Ha a CARRY nem aktív (0), akkor az SUBC ugyanúgy működik, mint az SUB utasítás.

$$\text{dst} + \neg \text{src} + C \rightarrow \text{dst}$$

- CMP: Ez a művelet az assembly nyelv egyik alapvető utasítása, amely két operandust hasonlít össze. A CMP utasítás lényegében az alapvető kivonás műveletét végzi el, de az eredményt nem tárolja el. Az CMP utasítás csak a jelzőbiteket állítja be annak megfelelően, hogy az első operandus nagyobb, kisebb vagy egyenlő-e a másodikkal megadott regiszterrel.

$$\text{dst} - \text{src}$$

- DADD: Az összeadás (Addition) az alapvető összeadás műveletét valósítja meg, de specifikus jelentéssel nem rendelkezik a legtöbb architektúrában.

Az "D" prefix (például az DADD) gyakran a Double, azaz double számokhoz kapcsolódik, és azt jelzi, hogy a művelet double számokkal történik. Ez gyakran az FP (Floating Point), azaz lebegőpontos számokkal való műveletek esetén fordul elő, ahol a dupla precizitású adatokhoz szükség lehet 64 bites (vagy ennél nagyobb) adatokra.

$$\text{src} + \text{dst} + C \rightarrow \text{dst} \text{ (decimally)}$$

- BIT: A logikai és (\wedge) műveletet valósítja meg a megadott src és dst regiszterek között, majd a közöttük lévő és kapcsolat értékét a dst regiszterbe tárolja el.

$$\text{src} \wedge \text{dst}$$

- BIC: A BIT-hez hasonlóan logikai és kapcsolatot vizsgál a két megadott regiszter között, viszont az src regiszter értékének a negáltjával.

$$\neg \text{src} \wedge \text{dst} \rightarrow \text{dst}$$

- BIS: A logikai vagy (\vee) műveletet valósítja meg a megadott src és dst regiszterek között, majd a közöttük lévő és kapcsolat értékét a dst regiszterbe tárolja el.

$$\text{src} \vee \text{dst} \rightarrow \text{dst}$$

- XOR: A logikai kizáró vagy (\oplus) műveletet valósítja meg a megadott src és dst regiszterek között, majd a közöttük lévő kizáró vagy kapcsolat értékét a dst regiszterbe tárolja el.

$$\text{src} \oplus \text{dst} \rightarrow \text{dst}$$

- AND: A logikai és (\wedge) műveletet valósítja meg a megadott src és dst regiszterek között, majd a közöttük lévő és kapcsolat értékét a dst regiszterbe tárolja el.

$$\text{src} \wedge \text{dst} \rightarrow \text{dst}$$

- RRC:

$$C \rightarrow \text{MSB} \rightarrow \dots \text{LSC} \rightarrow C$$

- RRA:

$$\text{MSB} \rightarrow \text{MSB} \rightarrow \dots \text{LSB} \rightarrow C$$

- PUSH:

$$\text{SP} - 2 \rightarrow \text{SP}, \quad \text{src} \rightarrow @\text{SP}$$

- SWPB:

$$\text{src} \rightarrow \text{dst}, \quad \text{dst} \rightarrow \text{src}$$

- CALL:

$$\text{SP} - 2 \rightarrow \text{SP}, \quad \text{PC} + 2 \rightarrow @\text{SP}, \quad \text{dst} \rightarrow \text{PC}$$

- RETI:

$$\text{TOS} \rightarrow \text{SR}, \quad \text{SP} + 2 \rightarrow \text{SP}$$

$$\text{TOS} \rightarrow \text{PC}, \quad \text{SP} + 2 \rightarrow \text{SP}$$

- SXT:

$$\text{Bit7} \rightarrow \text{Bit8} \rightarrow \text{Bit9} \rightarrow \text{Bit10} \rightarrow \text{Bit11} \rightarrow \text{Bit12}$$

$$\text{Bit12} \rightarrow \text{Bit13} \rightarrow \text{Bit14} \rightarrow \text{Bit15}$$

- JMP: Az ugrás (Jump) egy alapvető utasítás az assembly programozási nyelvben, amelyet elágazások

végrehajtására használnak. Az ugrás utasítás arra szolgál, hogy átugorja a program kódsorának egy adott részét, és folytassa a végrehajtást egy másik címről. Az utasítás paraméterként egy cél-címet vár, ahova a program vezérlése átkerül. Ez a cél-cím lehet egy cím, egy regiszterben vagy változóban tárolt érték.

Az ugrás utasítás a program futását a cél-címen lévő utasításokkal folytatja, anélkül hogy bármilyen feltételt ellenőrizne. Ez azt jelenti, hogy az JMP utasítás általában egy abszolút elágazást valósít meg, vagyis mindig végrehajtható, függetlenül a körülményektől. [4] [5]

I. MÉRÉSI FELADAT

Végezzen el a kettővel való szorzást egy 8 bites előjel nélküli számon.

Ebben a feladatban egy 8 bites előjel nélküli számot szorzunk meg 2-vel. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
1 mov.b #6,R04 ; szorzando
2 rla.b R04
```

A program lefuttatása után a szimulált mikrokontroller regiszterein az I. táblázatban összegyűjtött értékeket láttuk.

I. táblázat. A I. mérési feladatban a feladat lefuttatása után a regiszterekben maradt értékek.

Regiszter neve	Regiszter értéke
R4	0x000C (12)

II. MÉRÉSI FELADAT

Végezzen el a tízzel való szorzást egy 8 bites előjel nélküli számon.

Ebben a feladatban egy 8 bites előjel nélküli számot szorzunk meg 10-zel. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
1 mov.b #6,R4 ; szorzando
2 rla.b R4
3 rla.b R4
4 rla.b R4
5 mov.b #6,R5
6 add.b R4,R5
7 add.b R4,R5
```

A program lefuttatása után a szimulált mikrokontroller regiszterein a II. táblázatban összegyűjtött értékeket láttuk.

II. táblázat. A II. mérési feladatban a feladat lefuttatása után a regiszterekben maradt értékek.

Regiszter neve	Regiszter értéke
R4	0x0030 (48)
R5	0x003C (60)

III. MÉRÉSI FELADAT

Végezzen el a két 8 bites előjel nélküli szám szorzását.

Ebben a feladatban két 8 bites előjel nélküli számot szorzunk össze egymással. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
1 mov.b #12,R4 ; szorzando
2 mov.b #2,R5 ; szorzo
3 clr R6
4
5 AA:
6 add R5,R6
7 dec R4
8 jne AA
```

A program lefuttatása után a szimulált mikrokontroller regiszterein a III. táblázatban összegyűjtött értékeket láttuk.

III. táblázat. A III. mérési feladatban a feladat lefuttatása után a regiszterekben maradt értékek.

Regiszter neve	Regiszter értéke
R4	0x0000 (0)
R5	0x0002 (2)
R6	0x0018 (24)

IV. MÉRÉSI FELADAT

Végezzen el szorzást két 16 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét.

Ebben a feladatban két 16 bites előjel nélküli számot szorzunk össze egymással. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
1 mov.w #12,R4 ; szorzando
2 mov.w #2,R5 ; szorzo
3 clr R6
4
5 AA:
6 add R5,R6
7 dec R4
8 jne AA
```

A program lefuttatása után a szimulált mikrokontroller regiszterein a IV. táblázatban összegyűjtött értékeket láttuk.

A flageket megvizsgálva láthatjuk, hogy a kód természetéből adódóan, amikor összeadásnál a két szám összege meghaladja a 16 bitbe elférő szám értékét, az túlszordul, ezt angolul overflow-nak hívják. Ilyenkor a C, mint carry flag értéke igaz lesz.

IV. táblázat. A IV. mérési feladatban a feladat lefuttatása után a regiszterekben maradt értékek.

Regiszter neve	Regiszter értéke
R4	0x0000 (0)
R5	0x0002 (2)
R6	0x0018 (24)

V. MÉRÉSI FELADAT

Végezzen el szorzást két 32 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét.

Ebben a feladatban két 32 bites előjel nélküli számot szorzunk össze egymással. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```

1 mov.w #20,R5          ; szorzando
2 mov.w #10,R6
3 mov.w #0,R7
4 mov.w #0,R8
5
6 mov.w #5,R9           ; szorzo
7 mov.w R9,R10
8
9 mov.w #0,R11
10 mov.w #0,R12
11 mov.w #0,R13
12 mov.w #0,R14
13
14 mov.w #0,R4
15
16
17 a:  jz break
18     rra R10
19     rrc R9
20     jnc b
21
22     add R5,R11
23     addc R6,R12
24     addc R7,R13
25     addc R8,R14
26
27     rla R5
28     rlc R6
29     rlc R8
30
31     dec.w R4
32     jmp a
33
34
35 b:  rla R5
36     rlc R6
37     rlc R7
38     rlc R8
39
40     dec.w R4
41     jmp a
42
43
44 break: nop

```

A program lefuttatása után a szimulált mikrokontroller regiszterein az **V.** táblázatban összegyűjtött értékeket láttuk.

A flageket megvizsgálva láthatjuk, hogy a kód természetéből adódóan, amikor összeadásnál a két szám összege meghaladja a 16 bitbe elférő szám értékét, az túlsordul, ezt angolul overflow-nak hívják. Ilyenkor a C, mint carry flag értéke igaz lesz.

V. táblázat. A **V.** mérési feladatban a feladat lefuttatása után a regiszterekben maradt értékek.

Regiszter neve	Regiszter értéke
R4	0x0000 (0)
R5	0x0000 (0)
R6	0x0000 (0)
R7	0x0000 (0)
R8	0x0000 (0)
R9	0x0000 (0)
R10	0x0000 (0)
R11	0x0064 (100)
R12	0x0096 (—)
R13	0x001E (—)
R14	0x0000 (0)

VI. MÉRÉSI FELADAT

A tanultakat ellenőrizze az 1-5; feladat megoldásával előjeles környezetben is.

Előjeles környezetben a programok létrehozása és a megírt kódok nem változnak, csak arra kell figyelni, hogy a számok tárolása során az előjel bit miatt 8 bit helyett csak 7 biten tárolhatunk számokat. Előjeles összeadásnál érdemes a kettes komplement számábrázolást használni. Ez azt jelenti, hogy a legnagyobb helyiértékű bitben tároljuk a szám előjelét. Ha a szám nem negatív, akkor a bit értéke 0, ha viszont negatív, akkor a bit értéke az 1 értéket vesz fel. Annak érdekében, hogy megállapíthassuk, hogy az eredmény pozitív vagy negatív, az úgynevezett "negatív zászlót" (N flag) kell figyelni: ha értéke 0, akkor nem negatív, ha 1, akkor negatív számot kaptunk eredményül. Ha a végeredmény a bit számának megfelelő tartományon kívül esik, az úgynevezett "túlsordulás zászló" (O flag; Overflow) értéke 0-ról 1-re vált. A "carry bit" akkor lesz 1, ha a túlsordulás a tartomány pozitív felén történik.

VII. MÉRÉSI FELADAT

Végezzen el osztást egy 16 bites osztandó és egy 8 bites osztó között előjel nélküli számábrázolás esetén.

Ebben a feladatban egy 16 bites előjel nélküli osztandó és 8 bites előjel nélküli osztót osztunk el. Ehhez a csatolt '16and8bitdivide.asm' assembly kódfájl részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

VIII. MÉRÉSI FELADAT

Végezzen el az osztást 16 bites előjel nélküli számábrázolás mellett.

Ebben a feladatban két 16 bites előjel nélküli számot osztunk el egymással. Ehhez a csatolt '16bitdivide.asm' assembly kódfájl részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

IX. MÉRÉSI FELADAT

Végezzen el az osztást 32 bites előjel nélküli számábrázolás mellett.

Ebben a feladatban két 32 bites előjel nélküli számot osztunk el egymással. Ehhez a csatolt '32bitdivid.asm' assembly kódfájl részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

X. MÉRÉSI FELADAT

A tanultakat ellenőrizze az 7-8 feladat megoldásával előjeles környezetben is.

A szorzáshoz hasonlóan, ebben az esetben is azonosak a programok a előjel nélküli változatokhoz képest, de fontos, hogy megjelöljük a számok előjelét. Az osztás definíció alapján hasonló a kivonásra, kivonás esetén pedig ugyanúgy érvényes a számábrázolási tartomány, mint az összeadás esetén. Ha a végeredmény negatív, akkor az N flag 1 értéket vesz fel, és kettes komplementként kell kezelni. Emellett az overflow flag is jelzi a túlsordulást a művelet során, így a kapott érték nem fér bele az ábrázolási tartományba. A carry bit ebben az esetben ellentétesen működik, hiszen negatív irányból történik túlsordulás esetén vált csak 1-es értékre.

HIVATKOZÁSOK

- [1] T. Finley, *Two's Complement*. 2000. cím: <https://www.cs.cornell.edu/~tomf/notes/cps104/twoscomp.html> (elérés dátuma 2024. 05. 15.).
- [2] *x86 Assembly Language Reference Manual*. Oracle. cím: <https://docs.oracle.com/cd/E19641-01/802-1948/802-1948.pdf> (elérés dátuma 2024. 05. 15.).
- [3] *Description of the MIPS R2000*. Imperial College London. cím: <https://www.doc.ic.ac.uk/lab/secondyear/spim/node9.html> (elérés dátuma 2024. 05. 16.).
- [4] Wikipedia, *Mikrovezérlő*. cím: <https://hu.wikipedia.org/wiki/Mikrovez%C3%A9rl%C5%91> (elérés dátuma 2024. 05. 15.).
- [5] *MSP430x1xx Family User's Guide*. Texas Instruments. cím: https://www.ti.com/lit/ug/slau049f/slau049f.pdf?ts=1649510678917&ref_url=https%253A%252F%252Fwww.ti.com%252Fsite%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253Dslau049%2526nr%253D160 (elérés dátuma 2024. 05. 16.).