Mikrokontroller I.

Kékesi Kristóf
NEPTUN kód: ZI6I4M
Mérőpár: Bor Gergő

Mérés ideje: 2024.05.08. 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 az április 23-án 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;**

MÉRÉSSEL 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) számrrendszereket, 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 komplemens számábrázolási módszer**: A kettes komplemens 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 túlcsordult-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 aritmetikaiművelet során az eredmény túlcsordult-e (overflow), vagyis több bitet igényel, mint amennyi a célregiszterben elfér. A SUBBC 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.

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

– 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.

$$\text{src} \wedge \text{dts}$$

– 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:

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

– RRA:

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

– 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 végrehajtódik, függetlenül a körülményektől. [4] [5]

I. Mérési feladat

Végezzen el összeadást két 8 bites előjel nélküli szám között.

```
mov.b #5,R4
mov.b #6,R5

add.b R4,R5
```

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

Regiszter neve	Regiszter értéke
R4	$0x000B$ (11)
R5	$0x0006$ (6)

II. Mérési feladat

Végezzen el összeadást 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 adtunk össze. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
mov.w #65535,R4
mov.w #6,    R5

mov.w R5,    R4
```

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

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

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

Regiszter neve	Regiszter értéke
R4	$0x0005$ (5)
R5	$0x0006$ (6)

III. Mérési feladat

Végezzen el összeadást 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 adtunk össze. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
mov.w #3000,R4
mov.w #3200,R5
mov.w #3400,R6
mov.w #3600,R7

add.w R4,R5
add.w R6,R7
addc.w R5,R7
```

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

A flageket megvizsgálva láthatjuk, hogy amikor a két szám összege meghaladja a 32 bitbe elférő szám értékét, az túlcsordul, ezt angolul overflow-nak hívják. Ilyenkor a C, mint carry flag értéke igaz lesz.

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

Regiszter neve	Regiszter értéke
R4	$0x0BB8$ (3000)
R5	$0x1838$ (6200)
R6	$0x0D48$ (3400)
R7	$0x3390$ (13200)

IV. Mérési feladat

Végezzen el összeadást két 64 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét.

Ebben a feladatban két 64 bites előjel nélküli számot adtunk össze. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
mov.w #11,R4
mov.w #22,R5
mov.w #33,R6
mov.w #44,R7
mov.w #55,R8
mov.w #66,R9

add.w R4,R5
add.w R6,R7
add.w R8,R9
```

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

A flageket megvizsgálva láthatjuk, hogy amikor a két szám összege meghaladja a 64 bitbe elférő szám értékét, az túlcsordul, ezt angolul overflow-nak hívják. Ilyenkor a C, mint carry flag értéke igaz lesz.

V. Mérési feladat

A tanultakat ellenőrizze az 1;2;3; feladat megoldásával előjeles környezetben is.

A programok létrehozása és a kódok megírása ugyanúgy működik előjelezett környezetben, de fontos figyelembe venni,

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

Regiszter neve	Regiszter értéke
R4	$0x000B$ (11)
R5	$0x0021$ (33)
R6	$0x0021$ (33)
R7	$0x0047$ (77)
R8	$0x0037$ (55)
R9	$0x0079$ (121)

hogy az előjelbit miatt csak 7 biten tudunk számokat tárolni, nem 8 biten. Előjeles összeadás esetén érdemes a kettes komplemens számábrázolást alkalmazni, ahol a legnagyobb helyiértékű biten tároljuk a szám előjelét. Ha a szám pozitív, akkor ennek a bitnek értéke 0, ha viszont negatív, akkor 1. Az eredmény pozitív vagy negatív jellegét az úgynevezett negatív (N) flag jelzi. Amennyiben ez az érték 0, akkor pozitív, ha 1, akkor negatív számot kaptunk eredményül. Ha az eredmény a bitek számának megfelelő tartományon kívül esik, a túlcsordulás (O; Overflow) flag értéke 0-ról 1-re vált. A carry (C) flag akkor 1, ha a túlcsordulás a tartomány pozitív felé történik.

VI. Mérési feladat

Végezzen el kivonást két 8 bites előjel nélküli szám között.

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

```
mov.b #5,R4
mov.b #6,R5

sub.b R4,R5
```

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

A flageket megvizsgálva láthatjuk, hogy amikor a két szám különbsége kisebb a 8 bitbe elférő szám értékénél, az alulcsordul, ezt angolul underflow-nak hívják.

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

Regiszter neve	Regiszter értéke
R4	$0x0005$ (5)
R5	$0x0001$ (1)

VII. Mérési feladat

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

```
mov.w #65535,R4
mov.w #6,    R5

sub.w R5,    R4
```

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

Regiszter neve	Regiszter értéke
R4	$0xFFFF$ (65535)
R5	$0xFFF9$ (65529)

VIII. Mérési feladat

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

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

```
mov.w #3200,R4
mov.w #3000,R5
mov.w #3600,R6
mov.w #3400,R7

sub.w R4,R5
sub.w R6,R7
subc.w R5,R7
```

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

A flageket megvizsgálva láthatjuk, hogy amikor a két szám különbsége kisebb a 32 bitbe elférő szám értékénél, az alulcsordul, ezt angolul underflow-nak hívják.

A kivonás műveletnél a "kölcsönvevés" (borrow) flag jelzi, hogy a két szám amivel a kivonást végeztük megeggyeznek e. Amennyiben a két szám értéke megegyezik, a művelet elvégzésével a borrow flag igaz értéket vesz fel. Kivonáson kívül az összehasonlítás művelet használja még a borrow flaget. Összehasonlításkor (CMP) a borrow flag logikai igaz értéket vesz fel, ha a két megadott érték, vagy regiszter tartalma megyegyezik.

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

Regiszter neve	Regiszter értéke
R4	$0x0C80$ (3200)
R5	$0x00C8$ (200)
R6	$0x0E10$ (3600)
R7	$0x00C8$ (200)

IX. Mérési feladat

Végezzen el kivonást két 64 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a borrow bit értékét.

Ebben a feladatban két 64 bites előjel nélküli számot vonunk ki. Ehhez az alábbi assembly kód részletet illesztettük bele az IAR szimulátor által elkészített assembly sablonba.

```
mov.w #66,R4
mov.w #55,R5
mov.w #44,R6
mov.w #33,R7
mov.w #22,R8
mov.w #11,R9

sub.w R4,R5
sub.w R6,R7
sub.w R8,R9
```

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

A flageket megvizsgálva láthatjuk, hogy amikor a két szám különbsége kisebb a 64 bitbe elférő szám értékénél, az alulcsordul, ezt angolul underflow-nak hívják.

A kivonás műveletnél a "kölcsönvevés" (borrow) flag jelzi, hogy a két szám amivel a kivonást végeztük megeggyeznek e. Amennyiben a két szám értéke megegyezik, a művelet elvégzésével a borrow flag igaz értéket vesz fel. Kivonáson kívül az összehasonlítás művelet használja még a borrow flaget. Összehasonlításkor (CMP) a borrow flag logikai igaz értéket vesz fel, ha a két megadott érték, vagy regiszter tartalma megyegyezik.

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

Regiszter neve	Regiszter értéke
R4	$0x0042$ (66)
R5	$0x000B$ (11)
R6	$0x002C$ (44)
R7	$0x000B$ (11)
R8	$0x0016$ (22)
R9	$0x000B$ (11)

X. Mérési feladat

A tanultakat ellenőrizze az 6;9; feladat megoldásával előjeles környezetben is.

Az összeadáshoz hasonlóan, ebben az esetben is azonosak a programok a előjel nélküli változatokhoz képest, de fontos, hogy megjelöljük a számok előjelét. Kivonás esetén ugyanúgy érvényes a számábrázolási tartomány, mint az összeadás esetén. Ha a végeredmény negatív, akkor az N flag 1 értéket vesz fel, és kettes komplemensként kell kezelni. Emellett az overflow flag is jelzi a túlcsordulást 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 túlcsordulás 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. 10.).

[2] *x86 Assembly Language Reference Manual*. Oracle. cím: https://docs.oracle.com/cd/E19641-01/802-1948/802-1948.pdf (elérés dátuma 2024. 05. 10.).

[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. 10.).

[4] Wikipedia, *Mikrovezérlő*. cím: https://hu.wikipedia.org/wiki/Mikrovez%C3%A9rl%C5%91 (elérés dátuma 2024. 05. 10.).

[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%252Fsitesearch%252Fen-us%252Fdocs%252Funiversalsearch.tsp%253FlangPref%253Den-US%2526searchTerm%253Dslau049%2526nr%253D160 (elérés dátuma 2024. 05. 10.).