Mikrokontroller I.

Radványi Zita
NEPTUN kód: F346YE
Mérőpár: Zahoray Anna
NEPTUN kód: EF2JUM
Mérés ideje: 2023. 03. 23. 8:00-11:00
Mérés helye: Pázmány Péter Katolikus Egyetem Információs Technológiai és Bionikai Kar
Magyarország, 1083, Budapest, Práter utca 50/a
radvanyi.zita@hallgato.ppke.hu

***Abstract*—Komplemens számábrázolás használata, összeadás megvalósítása, kivonás megvalósítása**

***Keywords*-Mikrokontroller, regiszter, műveletek, számábrázolás**

I. Fogalmak, használt programok

- Gépi számábrázolás: a számok (számító)gépek memóriájában vagy egyéb egységében történő tárolását vagy valamely adathálózaton történő továbbítás formátumát adja meg
- Előjelbites ábrázolás: az előjel nélküli egészek ábrázolásához egy előjelet jelentő bitet adunk (0 ha pozitív és 1 ha negatív az előjel). A többi biten pedig ábrázoljuk a szám értékét.
- Komplemens számábrázolás: A kettes komplementerképzés módszere Ugyanis a szorzás összeadások sorozatára, az osztás pedig kivonások sorozatára vezethető vissza; ha tehát a kivonást sikerül összeadásra visszavezetni, akkor a gépnek tulajdonképpen csak az összeadás műveletét kell ismernie. A fixpontos ábrázolási módoknál a törtpont (tizedesvessző, tizedespont, kettedespont stb.) helye rögzített. Többségében egész számok tárolására használják, így a törtpont az ábrázolt szám végén van. Az egy byte-on tárolt bináris számírásnak könnyen belátható korlátjai vannak. A számítógépen a fixpontos számokat általában két byte-on vagy négy byte-on ábrázolják, azaz egy szám hossza 16 vagy 32 bit. De a negatív számok ábrázolásáról is gondoskodnunk kell.
- Számrendszerek: A számábrázolási rendszer, röviden: számrendszer meghatározza, hogyan ábrázolható egy adott szám. A számjegy egy szimbólum (vagy azok csoportja), ami egy számot ír le. A számjegyek éppen úgy különböznek az általuk leírt számtól, mint egy szó attól a dologtól, amit valójában jelent.
- Regiszter: A regiszterek a számítógépek központi feldolgozó egységeinek (CPU-inak), illetve mikroprocesszorainak gyorsan írható-olvasható, ideiglenes tartalmú, és általában egyszerre csak 1 gépi szó feldolgozására alkalmas tárolóegységei. A regiszterek felépülhetnek statikus memóriaelemekből vagy egy RAM memória részeként. Néhány géptípusnál egyetlen chipben mind a két megoldást alkalmazzák. Egy-egy regiszter hozzáférési ideje általában néhányszor 10 ns.
- Számábrázolás: A számábrázolás az a mód, ahogyan a számokat szimbólumokkal jelöljük. Ez történhet akár írásban, akár szóban, akár máshogy (pl. tárgyak vagy valamilyen gép által). Szűkebb értelemben véve a számábrázolás az a mód, ahogyan a számítógépek a számszerű adatokat tárolják (gépi számábrázolás).
- IAR Embedded Workspace: Az IAR Embedded Workbench számos mikroprocesszorhoz és mikrokontrollerek a 8-, 16- és 32-bites szegmensben, lehetővé téve, hogy egy jól ismert fejlesztői környezet a következő projektjéhez is. Biztosít egy könnyen megtanulható és rendkívül hatékony fejlesztői környezet maximális kóddal öröklési képességek, átfogó és konkrét céltámogatás. IAR Embedded A Workbench elősegíti a hasznos munkamódszert, és ezáltal a munkavégzés jelentős csökkentéséta fejlesztési idő az IAR Systems eszközeivel érhető el.

II. Első feladat

Végezzen el összeadást két 8 bites előjel nélküli szám között. Helyezze az összeadandókat mint konstansokat egy-egy regiszterbe, majd végezze el az adatok összeadását. Az eredményt ellenőrizze a Registers ablakban.A program működését lépésenkénti futtatással lehet ellenőrizni. Ismételje meg a feladatot más konstansokkal is. Ellenőrizze, hogy mi történik akkor ha az eredmény túllép a számábrázolási határon. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
Az első feladat elkezdéséhez első sorban a mikrokontroller programozáshoz szükséges IAR Embedded Workbench programot használtunk, melynek segírségével létrehozhettuk a szükséges szimulációkat, ezzel tesztelve a különböző értékeken a feladatokat. A program középső ablaka szolgál arra, hogy létrehozzuk a megfelelő programkódokat. Bal oldalon található a Registers 1 ablak, melynek segítségével megfigyelhetjük az egyes regiszterekben szereplő értékeket, azoknak a változását a program során. Ennek az ablaknak a megfigyelése kulcsfontosságú volt a feladatok megoldása során, hiszen itt tudtuk nyomonkövetni az egyes regiszterekben tárolt értékek változását, valamint nagy segítséget jelentett a Debugger ablak is az esetleges hibák kiszűrésében. Ez különböző hibakódokkal jelez vissza felénk, ha hibá kódot hoztunk létre, emiatt elakaszva a szabályos futtatást. A feladat során a mintaként megadott programrészletet alakítottuk át, valamint futtattuk le több értékkel, így megvizsgálva az eredményeket. Fontos észrevétel volt számunkra, hogy mivel 16 bit-es rendszert használ a program, így előjel nélkül a legnagyobb megadható érték a 255 volt, míg elsőjeles számábrázolásként -128-tól 127ig tudtuk megjelenítei a számokat.

A mintaként megadott kód az alábbi módon működik:

```
mov.b   #5,R04
mov.b   #6,R05
add.b   R05,R04
```

A leírt kód az alábbi utasításokat hajtotta végre: Elsőször a négyes számú regiszterbe helyezte el az első konstanst a megadottak alapján, majd a másokódik lépésben az ötös számú regiszterbe helyezte el a másik megadott konstanst. Ezt követően összeadta ezeket és eltárolta az eredményt a négyes számú regiszterbe. A fent leírt kód a következő utasításokat végezte el. Először a négyes számú regiszterbe (R4) behelyezte az első konstanst az összeadandók közül (én esetemben ez a szám a 3 volt). Második lépésben az ötös számú regiszterbe (R5) beillesztette a második konstanst (az én esetemben ez az 5 volt). Végül összeadta a program a két konstanst és eltárolta az eredményt a négyes számú regszterbe.

```
R4          = 0x000B
R5          = 0x0006
```

III. Második feladat

Végezzen el összeadást két 16 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.

Ebben az esetben a létrehozott szimuláció nagyon hasonlatosan működik az előzőekben létrehozottra. A különbség a két program között, hogy ez esetben 16 biten ábrázotuk a számokat, így a legnagyobb ábrázolható szám a 65535 volt. A létrehozott programkód:

```
mov.b #255,R6
mov.b #150,R7
add.b R6,R7
```

Ez a kód nagyon hasonlatosan űködik az előzőhöz, annyi különbségg vélhető felfedezni, hogy itt már 16 bit-en dolgoztunk, így nem volt szükség a ”b” jelzésre a parancsokat követően. Ezt követően a hatos regiszterbe az alábbi eredmény került:

```
R6          = 0x00FF
R7          = 0x0095
```

Ebben az esetben ha túlcsordulás állhat fent, ha a kapott eredmény nagyobb, mint 65634, ilyenkor az eredmény el fog térni a helyes eredméntől, valamint a carry bit 0-ról 1-es értékre vált.

IV. Harmadik feladat

Végezzen el összeadást két 32 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.

Ezen feladat során már új ismereteket is fel kellett használnunk az előzőekhez képest. Tekintettel arra, hogy a használt program 16 bites regiszterekkel tud csak számolni, így az összeadandó konstansokat két db 16 bites regiszterben kell eltárolni. Az első két regiszterben eltároltuk a két konstans kisebb helyiértékű részét, majd kiszámoltuk ezek összegét. Ezt követően újabb regisztereken eltároltuk a konstansok nagyobb helyiértékű részeit, majd ezek összegét is kiszámoltuk, ám ebben az esetben már a ”ADDC” parancsot használva, hogy figyelembe vegye a carry értéket is figyelembe vegye. Ezzel a paranccsal már meg is kaptuk a 32 bites konstansok összegét. Használt programkód:

```
mov.b #8,R8
mov.b #5,R9
add R8,R9
mov.b #12,R10
mov.b #13,R11
addc R10,R11
```

Megfigyelhető, hogy a második összeadás során az ”ADDC” parancs segítségével a carry bitet is figyelembe tudtuk venni, ezzel gondoskodva arról az esetről, ha az előző összeadás során túlcsordulás keletkezett, akkor azt az érték megjelenjen ebben a számításban. Ez alapján az összeadás eresménye:

```
R8          = 0x0008
R9          = 0x000D
R10         = 0x000C
R11         = 0x0019
```

V. Negyedik feladat

Végezzen el összeadást két 64 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
Az előző feladathoz hasonlóan a 64 bit-es számokat is fel kell osztani 16 bit-es egységekre, amelyekkel külön kell elvégezni az összeadásokat. Ezáltal létre kell hozni négy darab 16 bites egységet, ezzel kiadva az összesen 64 bitet. Ebben az esetben csak az első művelet során használjuk az ”ADD” parancsot, az összes többi esetben az ”ADDC” parancsot kell használni. Ez esetben a legnagyobb megjeleníthető szám a 2^{64}.

A lértehozott programkód:

```
mov.b #9,R4
mov.b #8,R5
add R4,R5
mov.b #15,R6
mov.b #16,R7
addc R6,R7
mov.b #21,R8
mov.b #22,R9
addc R8,R9
mov.b #30,R10
mov.b #31,R11
addc R10,R11
```

A letrehozott programkódban mefigyelhető, hogy csak az első esetben használtuk az ”ADD” parancsot, ezt követően csak az ”ADDC” parancsot alkalmaztuk, ezzel figyelembe véve a korábbi zámytásokból esetlegesen fennmaradt carry értékeket, így elkerülve a fals eredmények létrejöttét. Az ezáltal létrejött eredmények:

R4	= 0x0009
R5	= 0x0011
R6	= 0x000F
R7	= 0x001F
R8	= 0x0015
R9	= 0x002B
R10	= 0x001E

VI. Ötödik feladat

A tanultakat ellenőrizze az 1;2;3; feladat megoldásával előjeles környezetben is. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
Előjeles környezetben nem változik a programok létrehozása, valamint a megírt kódok sem változnak, arra kell figyelni, hogy a számok tárolása során az előjel bit miatt a 8 bit helyett csak 7 biten tudunk számokat tárolni. Elsőjeles összeadás során a kettes komplemens számábrázolást érdemes használni, ami azt jeleni, hogy a lefoglalt, legnagyobb helyiértékű bitjén tároljuk a szám előjelét. Ha a szám nem negatyv, akkor a bit értéke 0, ha viszont a szám negatív, akkor a bit 1 értéket vesz fel. Annak érdekében, hogy tudjuk, hogy a kapott eredmény pozitív vagy negatív-e, az úgynevezett ”negativ flaget” (N flag) kell figyelni, ha ennek az értéke 0, akkor nem negatív, ha 1 az értéke, akkor pedig negatív számot kaptunk eredményül. Azokban az esetekben, amikor a végeredmény a fent leírt, bitek számának megfelelő, tartományon kívül esik, az úgynevezett ”overflow flag” (O flag) értéke 0-ról 1-re változik. A carry bit akkor 1, ha a túlcsordulás a tartomány pozitív felén történik.

V	= 0
SCG1	= 0
SCG0	= 0
OscOff	= 0
CPUOff	= 0
GIE	= 0
N	= 0
Z	= 0
C	= 0

VII. Hatodik feladat

Végezzen el kivonást két 8 bites előjel nélküli szám között. Helyezze a kissebítendőt az egyik míg a kivonandót egy másik regiszterbe, majd végezze el az adatok kivonását. Az eredményt ellenőrizze a Registers ablakban. A program működését lépésenkénti futtatással lehet ellenőrizni. Ismételje meg a feladatot más konstansokkal is. Ellenőrizze, hogy mi történik akkor ha az eredmény túllép a számábrázolási határon. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
A soron következő feladatok során a program szintaktikai megjelenése nagyon hasonló az összeadások létrehozásakor megírt programrészlettel, de ebben az esetben az ”ADD” parancs helyett ”SUB” azaz kivonás parancsot kell alkalmazni. A létrehozott programkód:

```
mov.b #10,R4
mov.b #2,R5
sub.b R4,R5
```

Ezt a programkódot lefuttatva az R4-es regiszterbe bele is került a két szám különbsége:

R4	= 0x000A
R5	= 0x00F8

Ha a kivonás elvégzése során a minimálisan ábrázolható számnál kisebb eredményt kapunk (mint a fenti példában tapasztalható), akkor a kapott eredményt kettes komplemensben kell értelmezni és az N flag értéke 1 lesz.

VIII. Hetedik feladat

Végezzen el kivonást két 16 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a borrow bit értékét. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
Ebben az esetben a programkód csaknem megegyezik az ötödik feladatban létrehozott kóddal, csak el kell távolítani az ”ADD” parancskat és meg kell változtatni őket ”SUB”, azaz kivonás parancsokká. A létrehozott programkód:

```
mov.b #200,R12
mov.b #50,R13
sub.b R12,R13
```

Hasonlóan az előző feladathoz, itt is az elsőként megadott, az R12-es regiszterbe kerül bele a kivonás eredménye:

R12	= 0x00C8
R13	= 0x006A

IX. Nyolcadik feladat

Végezzen el kivonást két 32 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a borrow bit értékét. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
Mivel a csak 16 bites regiszterek állnak rendelkezésre, így az kissebitendő és a kivonandó konstansokat két-két regiszterben kellett eltárolni. Első két regiszterben eltároltam a két szám kisebb helyiértékű részét, majd kiszámoltam ezek különbségét a ”SUB” paranccsal. Újabb két regiszterben eltároltam a konstansoknagyobb helyiértékű részét, akércsak az összeadás esetén, majd ezek különbségét is kiszámoltam a megfelelő parancs segítségével, így megkapva a keresett értéket. A felhasznált programkód:

```
mov.b #8,R4
mov.b #5,R5
sub R4,R5
mov.b #15,R6
mov.b #13,R7
subc R6,R7
```

Hasonlóan a 32 bites összeadáshoz, itt is észrevehető, hogy a második kivonás parancsnál nem az ”SUB”, hanem az ”SUBC” utasítást használtam, melyben a ”C” arra utal, hogy ez a parancs figyelembe veszi a carry értékét is. Ez azért fontos, mert így ha az előző művelet elvégzése során keletkezett maradéktag, akkor az a második kivonás során meg fog jelenni. Hasonlóan a 8 és 16 bites kivonáshoz, itt is kettes komplmensként kell kezelni az értéket, ha az kisebb nullánál. Ebben az esetben is az N flag 1-re vált. A számolás során kapott eredmények:

```
R4  = 0x0008
R5  = 0xFFFD
R6  = 0x000F
R7  = 0xFFFD
```

X. Kilencedik feladat

Végezzen el kivonást két 64 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a borrow bit értékét. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.
Ezen feladat során is nagyon hasonlatos a létrehozott kód a 64 bites számok összeadásához, hiszen mindent ugyanúgy kell megadni, kivéve a parancsot, ahol első esetben a ”SUB”, későbbi esetekben a ”SUBC” parancsot használtuk, ezáltal figyelembe véve a továbbvivendő maradéktagot, amyelyet a carry flag tárol. A létrehozott programkód:

```
mov.b #100,R8
mov.b #90,R9
sub R8,R9
mov.b #80,R10
mov.b #70,R11
subc R10,R11
mov.b #50,R12
mov.b #40,R13
subc R12,R13
mov.b #30,R14
mov.b #15,R15
subc R14,R15
```

Ezen számolások eredményét is lépésenként végigkövethetjük a regiszterek értékeiben, amelyek folyamatosan változnak, ahogy futtatjuk a programot. A kapott eredmények a következők:

```
R8  = 0x0064
R9  = 0xFFF6
R10 = 0x0050
R11 = 0xFFF5
R12 = 0x0032
R13 = 0xFFF5
R14 = 0x001E
R15 = 0xFFF0
```

XI. Tizedik feladat

A tanultakat ellenőrizze az 5;6;7; feladat megoldásával előjeles környezetben is. A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is.

Az összezadáshoz hasonlóan ebben az esetben is megegyeznek a prograkódok, az előjel nélküli verziókhoz, de ebben az esetben jelölni kell a konstans számok előjelét. Kivonás esetén ugyanúgy teljesül az ábrázolási tartomány, mint összeadás esetén. Ezen esetben is igaz, hogy ha egatyv szám a végeredmény, akkor az N flag 1 értéket vesz fel, valamint kettes komplemens óként kell kezelni. Ezen kívül az is teljesül, hogy az overflow flag jelzi, ha a művelet során túlcsordulás történt, ezáltal az ábrázolási tartományba nem tartozik bele a kapott éték. A carry bit ebben az esetben ellentétesen működik, hiszen ebben az esetben pakkor vált 1-es értékre, ha negatív irányból törénik túlcsordulás.

References

[1] https://wwwfiles.iar.com/maxq/guides/EW-UserGuide.pdf
[] https://hu.wikipedia.org/wiki/Regiszter-(számítástechnika)
[2] https://hu.wikipedia.org/wiki/Számábrázolás
[3] https://hu.wikipedia.org/wiki/Számrendszer
[4] https://hu.wikipedia.org/wiki/Fixpontosszámábrázolás