Mikrokontroller II.

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, szorzás megvalósítása, osztás megvalóstása**
Keywords – Mikorkontroller, Regiszter, Műveletek, Számábrázolás, Számrendszerek

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.
- Bináris szorzás: A négy alapművelet egyike a szorzás. Ezt a műveletet a számítógép aritmetika ismételt összeadások sorozatával végzi el. Számos algoritmust dolgoztak ki a géptervezők és a matematikusok. Közös vonásuk a bináris összeadás és léptetés (shiftelés). Az operandusok speciális rekeszekbe, regiszterekbe kerülnek. A regiszter fogalmát a 6.2 fejezetben ismertetjük. A szorzáshoz 3 regiszter szükséges (vannak olyan módszerek, amelyeknél több), ezek az AR (Accumulator Register), RR (Reserved Register) és QR (Quotient Register). A bináris szorzást számpéldán mutatjuk be.

II. Első feladat

Végezzen el a kettővel való szorzást egy 8 bites előjel nélküli számon. Helyezze a szorzandót, mint konstanst, egy regiszterbe, majd végezze el az adott feladatot, oly módon, hogy egyszer balra lépteti a regiszter értékét. 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. Gondolja végig, a 2-vel 4-el 8-al való szorzás menetét!

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. A minta programkód alapján létrehoztuk a szorzást, mely kettővel megszorozza az adott regiszterben tárolt értéket. Ezt a rla.b paranccsal tudtuk elvégezni, mely arra szolgál, hogy egyszer balra lépteti a regiszter értékét, ezéltal elvégezve a kettővel való szorzást. Első esetben a négyet szoroztuk meg kettővel, melynek az eredménye az R4-es regiszterbe került. Az esetünkben használt programkód az alábbi volt:

```
mov.b #4,R4
rla.b R4

mov.b #255,R5
rla.b R5
```

Ezt letesztetük olyan esetben is, ha a szorzás átlápi a megjeleníthető legnagyobb értéket, esetünkben ehhez a 255 értékét választottuk, hiszen annak a kettővel való szorzása már nem jeleníthető meg a megadott bit számon. Ilyenkor megfigyelhető az N és C flag értéke 1-re változik. A kettővel, néggyel

```
R4          = 0x0008
R5          = 0x00FE
```

valamitn nyolccal való szorzás nem más mint ahanyadik kettő hatvánnyal szorozzuk annyiszor kell alkalmazni a rla parancsot, hiszen ez balra tolja el az eredeti számot, ezáltal nő a helyiértékek értéke.

III. Második feladat

Végezzen el a tízzel való szorzást egy 8 bites előjel nélküli számon. Helyezze a szorzandót, mint konstanst, egy regiszterbe, majd végezze el az adott feladatot, oly módon, hogy egyszer balra lépteti a regiszter értékét.
Az alábbiakban létrehozott kód nagyban hasonlít az előzőekben megírtakra, viszont itt kezelnünk kellett azt, hogy nem kettő hatvánnyal szoroztunk, hanem a tízes számmal. Éppen ezért a tízzel való szorzás megfelel annak, mintha nyolccal szoroznánk és ezt követően még kétszer összeadást végeznénk el a számmal. Ebben az esetben is 8 biten számoltunk azaz a parancsok mögé el kellett helyezni a .b kiegészítést, amely megadja, hogy 16 bit helyett 8 biten végezzük el a műveleteet. Ezen kívül a másoik összeadásnál figyelembe kellett venni a korábban esetlegesen fennmaradt carry értéket, amelyet az addc paranccsal tuduk megadni. Éppen ebből az okból a létrehozott kódunk az alábbi:

```
mov.b #3,R6
rla.b R6
rla.b R6
rla.b R6
mov.b #3,R7
add.b R6,R7
addc.b R6,R7
```

Ezen számolást elvégezve a használt regiszterekbe bekerült érékek az alábbiak voltak:

```
R6          = 0x0018
R7          = 0x0033
```

IV. Harmadik feladat

Végezzen el a két 8 bites előjel nélküli szám szorzását. Helyezze a szorzandót és a szorzót, mint konstansokat egy-egy regiszterbe, majd végezze el az adott feladatot, oly módon, hogy egyszer balra lépteti a regiszter értékét. Az eredményt egy 16 bites regiszterbe helyezze.
Ezalatt a feladat során már olyan nehézségbe ütköztünk, hogy két teljes mértékben tetszőleges számnak kellett összeszorozn, így azt is bele kellett írni a szorző algoritmusba, hogy kezelje, hogy az mennyi kettes számrendszerbeli helyiértéket kell lépni, azon kívül pedig mennyi összeadást kell elvégezni az adott számokkal. Az adott regiszterekbe először betöltöttük a tetszőleges számokat, majd egy regiszterből kitöröltük az esetlegesen korábban bennemaradt adatokat. Ezt követően létrehoztuk magát a szorzást elvégző programrészletet. Először is összeadtuk az R9-es és a kinullázott R10-es regiszter értékeit, majd tízest osztást végetünk a dec paranccsal. Ezt követően a jne paranccsal meghívtuk az megadott programrészletet, ezzel elvégezve a két szám szorzását. Az általunk létrehozott programkód:

```
mov.b #20,R8
mov.b #33,R9
clr R10

AA:
add R9,R10
dec R8
cmp R8
jne AA
```

V. Negyedik feladat

Végezzen el szorozást 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. Az eredményt 32 biten ábrázolja!
Ezen feladat megoldásához hasonló programkódot hoztunk létre, mint az előző feladat során. Itt is kezelnünk kellett azt a helyzetet, hogy nem tudjuk, hogy pontosan mennyi lesz a szám értéke, ezáltal nem lehet tudni, hogy hány alkalommal alkalmazhatjuk a balra eltolát kettes szorzásként, és mennyi alkalommal kell hozzáadni a szám saját értékét önmagához. Ezen programkód során is 16 biten ábrázoltunk, így a parancsok után nem szükséges a .b utasítás kiegészítés. Az általunk létrehozott programkód:

```
mov #40,R11
mov #23, R12
clr R13

AA:
add R12, R12
dec R11
cmp R11
jne AA
```

VI. Ötödik feladat

Végezzen el szorzást két 32 bites előjel nélküli szám között. A művelet elvégzése során vizsgálja a carry bit értékét. Az eredményt 64 biten ábrázolja! A jegyzőkönyve csatolja az elkészített programokat, valamint az ellenőrzés eredményének értékelését is. Rögzítse a jegyzőkönyvbe a szorzással kapcsolatban szerzett ismereteit.
A korábbi feladatokhoz hasonlóan itt is egy komplikáltabb programkódot kellett létrehoznunk a szorzás miatt, valamint, hogy a szorzandó számok 32 biten kerüljenek be a programba és a kiszámított eredményt 64 biten ábrázoljuk.

VII. Hatodik feladat

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

VIII. Hetedik feladat

Ezen feladatok során a már korábban létrehozott osztást elvégző programrészletet egészítettük ki, értelmeztük, valamint mellette kommenttel jelöltük, hogy melyik programrészlet milyen feladatot végez el. Ezt az alábbiakban megfigyelhető:

```
divide: clr.w R15 ; kinullázza az R15-öt
        push.w R9 ; az R9 értéke bekerül
a stack pointer által mutatott helyre
        push.w R10 ; az R10 értéke bekerül
 a stack pointer által mutatott helyre
        push.w R11 ; az R11 értéke bekerül
 a stack pointer által mutatott helyre
        clr.w R10 ; kinullázza az R10-et
        clr.w R11 ; kinullázza az R11-et
        mov.w #0x20,R9
; adatmozgatás R9-be a
megadott értéket

divloop:
        rla.w R12  ;jobb oldalra 0-t
 shiftel be
        rlc.w R13  ;rla, csak a carryt
shifteli be
        rlc.w R10 ;rla, csak a carryt
 shifteli be
        rlc.w R11 ;rla, csak a carryt
 shifteli be
        sub.w R14,R10
;kivonás R10-R14, az eredményt
 az R10-be tárolja el
        subc.w R15,R11
;kivonás R11-R15, az eredményt
az R11-be tárolja el a carry érték
figyelembe vételével
        jnc div001 ;atugrás az adott
helyre, ha a carry értéke 0
        bis.w #0x1, R12
;beállítja a biteket a megfelelő
 rendeltetési helyükre

        add.w #0xFFFF,R9;összeadás a számot
az R9-be
        jne divloop ;ugorjon a megadott
helyre, ha nem egyenlő
        jmp div002 ;ugorjon a megadott
 helyre

div001
        add.w R14,R10
;összeadja az R14-be az R14-et
 és az R10-et
        addc.w R15,R11
;összeadja az R15-be az R11-et
 és az R11-et a carry érték figyelembe vételével
        add.w #0xFFFF,R9;összeadja az R9-be az
R9-et és a megadott szám értékét
        jne divloop ;ugorjon a megadott
helyre, ha nem egyenlő


div002
        mov.w R10,R14
;adatmozgatás R14-be az R10
értékét
        mov.w R11,R15
;adatmozgatás R15-be az R11
értékét
        pop.w R11  ; az R11 felveszi a stack
pointer által mutatott helyen lévő értéket
        pop.w R10 ; az R10 felveszi a stack
 pointer által mutatott helyen lévő értéket
        pop.w R9 ; az R9 felveszi a stack
pointer által mutatott helyen lévő értéket
        ret ; visszatérés a szubrutinból

multiply:
        mov.w R12, &MPY   ;adatmozgatás R12-bol
 signed multiply-al
        mov.w R14, &OP2   ;a jel kiterjeszti a
 2. operandust
        mov.w &RESLO, R12  ;adatmozgatás a reslo
 (low word) címbol az R12-be
        mov.w &RESHI, R13 ;a reshi tartalmazza
a high word eredményt
        ret  ;visszatérés a szubrutinból
```

IX. A mérés hibája

A mérés során nem merültek fel a program használatában probléma, a számolási hibákat a feladatok feldolgozása során figyelembe vettük a carry értékek továbbvitelével, valamint a további flagek vizsgálatával.

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