Mikrokontroller gyakorlati megismerése

Heiszman Henrik
Neptun kód: ENV2R9
Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
1083 Budapest, Práter utca 50/A
heiszman.henrik@hallgato.ppke.hu

***Téma*–A mikrokontrollerek gyakorlati megismerése, digitálisan kezelt 1 bites perifériák ki és bemenetének elérése. Feltételvizsgálat megvalósítása, aritmetikai összehasonlítások alkalmazása.**

I. A JEGYZŐKÖNYVBEN HASZNÁLT FOGALMAK

Bit: az információ, valamint az információt hordozó rész hosszának alapegysége is egyben. Lehetséges értékei: 0 és 1 (igaz és hamis). Az elnevezés a Binary digit kifejezésből származik.

Flag bitek: olyan bitek, amelyek az egyes műveletek során különböző információkat közölnek a fejlesztővel/felhasználóval. Példa flag bit-re: Carry bit (átviteli flag), amely elsősorban az aritmetikai utasítások műveleteinek átvitelét mutatja. Zero flag (nulla jező bit), amely ha egy valódi aritmetikai művelet vagy egy összehasonlítás elvlégzése során nulla eredmény keletkezik, akkor a ZF beáll egyesre.

Regiszter: számítógépek központi feldolgozó egységeinek, illetve mikroprocesszorainak gyorsan írható-olvasható, ideiglenes tartalmú tárolóegységei.

II. FELTÉTELES VEZÉRLÉS ÁTADÁSA

Feltételes vezérlés átadásról akkor beszélhetünk, ha az nem feltétel nélkül történik (például JMPL, amely feltétel nélküli ugrás). Ilyen feltétel lehet például a JNZ utasítás, amely használata során akkor történik átadás, ha a Zero flag 0.

III. KÉT SZÁM EGYENLŐSÉGÉNEK VIZSGÁLATA

Ebben az esetben a program eldönti két számról, melyek különböző regiszterben vannak eltárolva, hogy egyenlők-e.

Felhasznált programkód:

```
mov #5, R4
mov #5, R5
cmp R4,R5
jnz NoAction
mov #0, R15

NoAction:
  ret
```

Ebben a kódban a már megszokott „mov” utasítással eltároltam két különböző regiszterben (R4, R5) az ötös számot. Ezután a „cmp” paranccsal meghatároztam, hogy a program hasonlítsa össze a két regiszterben tárolt adatokat, majd a „jnz” (Jump if No Zero) paranccsal utasítottam, hogy abban az esetben, ha az összehasonlítás után a két szám különbsége nem nulla, tehát a zero flag értéke 0, a program álljon le. Viszont ha az összehasonlítás során a két érték különbsége nulla (Zero flag = 1), akkor a program fusson tovább és végezze el a következő utasítást, mely szerint az R15-ös regiszterbe tároljon el egy 0-t, ezzel jelezve, hogy a két szám egyenlő.

A program működését lépésenkénti futtatással ellenőrizetem. *(1-3. ábra)*

R4	0x0005
R5	0x0005

1. ábra
Az R4-es és R5-ös regiszterbe bekerültek a kívánt értékek

N	0
Z	1

2. ábra
Az R4-es és R5-ös regiszterek értékei megegyeznek, így a Zero flag 1-re vált

R14	0xCDCD
R15	0x0000

3. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül a 0

Az elkészült program futtatása során nem tapasztaltam semmilyen anomáliát, a várt eredményt kaptam.

Észrevehető, hogy a program nem csak abban az esetben működik, ha két egyenlő számot tárolok el. Ha a két összehasonlítandó szám különbsége nem nulla, azaz nem egyenlők, akkor a program nem tárol el 0-t az R15-ös regiszterben. Természetesen ez nem a legelegánsabb megoldás,

így elkészítettem egy külön programot az egyenlőtlenség kezelésére.

IV. KÉT NEM EGYENLŐ SZÁM VIZSGÁLATA

Ebben az esetbe a programot úgy kódoltam, hogy 1-es értékkel térjen vissza egy előre meghatározott regiszterben, ha két összehasonlított szám különbsége nem nulla, tehát nem egyenlőek.

Felhasznált programkód:

```
mov #8, R4
mov #5, R5
cmp R4,R5
jeq NoAction
mov #1, R15

NoAction:
  ret
```

Ebben a programban kevés voltozás történ az egyenlőségét vizsgáló párjához képest. A legszembetűnőbb az, hogy az eddig használt Jump if No Zero helyett egy „jeq” nevezetű (Jump if Equal) parancsot használtam. Ezen kívül még módosítottam, hogy tizenötös számú regiszterbe ne 0, hanem 1 kerüljön.

A Jump if Equal (ahogy azt a neve is sejteti), abban az esetben ugrik a program végére, a „cmp” parancs segítségével összehasonlított két szám egyenlő (Zero flag = 1).

A program működését ebben az esetben is lépésenkénti futtatással ellenőrizetem. *(4-6. ábra)*

R4	0x0008
R5	0x0005

4. ábra
Az R4-es és R5-ös regiszterbe bekerültek a kívánt értékek

N	0
Z	0

5. ábra
Az R4-es és R5-ös regiszterek értékei különböznek, így a Zero flag 0 marad

R14	0xCDCD
R15	0x0001

6. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 1-es számjegy

Az elkészült program az elvárásaimnak megfelelően futott többszöri alkalmazás után is.

V. KÉT SZÁM RELÁCIÓJÁNAK VIZSGÁLATA

Az elkészített programok ebben az esetben két szám relációját fogják vizsgálni. Első esetben, amikor az R4-es tag a nagyobb, második esetben pedig, R5-ös tag.

Első esetben felhasznált programkód (R4 > R5):

```
mov.b #9, R4
mov.b #7, R5
cmp R4,R5
jge NoAction
mov #2, R15

NoAction:
  ret
```

Ebben az esetben „jge” (Jump if Greater or Equal) parancsot használtam, amely akkor szakítja meg a programot, ha a kisebb sorszámú regiszterben (R4) tárolt szám kisebb vagy egyenlő a nagyobb sorszámú regiszterben (R5) tárolt számnál. Ha R4 > R5 (két szám különbsége nagyobb, mint 0), akkor a program eltárol egy kettest az R15-ös regiszterben.

A programot ebben az esetben is lépésenként futtattam és így ellenőriztem működését. *(7-9. ábra)*

R4	0x0009
R5	0x0007

7. ábra
Az R4-es és R5-ös regiszterbe bekerültek a kívánt értékek

N	1
Z	0
C	0

8. ábra
Az R4-es regiszterben lévő érték nagyobb, mint az R5-ben lévő, így az N bit 1-re vált

R14	0xCDCD
R15	0x0002

9. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 2-es számjegy

A második esetben felhasznált programkód (R4 < R5):

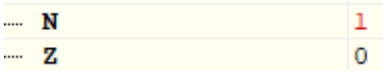
```
mov.b #6, R4
mov.b #7, R5
```

```
    cmp R5,R4
    jge NoAction
    mov #3, R15

NoAction:
    ret
```

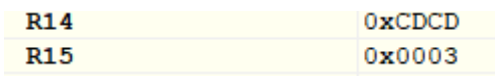
Ebben az esetben egyszerűen megcseréltem az összehasonlítás parancs során a két regisztert, így most a program az R5 > R4 relációt vizsgálja, és minden pontosan megegyezik ezen kívül az előzőkben leírtakkal.

A program helyességét ebben az esetben is debuggolással ellenőriztem. *(10-12.ábra)*

10. ábra
Az R4-es és R5-ös regiszterbe bekerültek a kívánt értékek



11. ábra
Az R4-es regiszterben lévő érték kisebb, mint az R5-ben lévő, így az N bit 1-re vált (különbségük negatív)



12. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 3-es számjegy

A futtatás során nem találtam hibát. Nem megfelelő értékek megadásakor (például a két szám egyenlő) a program helyesen nem tárolja el az R15-ös regiszterben a kívánt számot.

VI. KÉT SZÁM RELÁCIÓJÁNAK VIZSGÁLATA, AHOL ENGEDÉLYEZETT AZ EGYENLŐSÉG IS

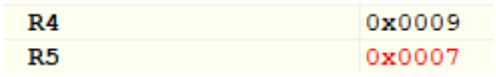
Az elkészített programok ebben az esetben is két szám relációját fogják vizsgálni, viszont itt az egyenlőség is elfogadott (kisebb vagy egyenlő, nagyobb vagy egyenlő). Első esetben, amikor az R4-es tag a nagyobb vagy egyenlő, második esetben pedig, R5-ös tag.

Első esetben felhasznált programkód (R4 >= R5):

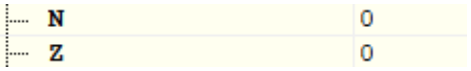
```
    mov.b #9, R4
    mov.b #7, R5
    cmp R5,R4
    jl NoAction
    mov #4, R15

NoAction:
    ret
```

Ebben az esetben „jl” (Jump if Less) parancsot használtam, amely akkor szakítja meg a programot, ha a kisebb sorszámú regiszterben (R4) tárolt szám nagyobb a nagyobb sorszámú regiszterben (R5) tárolt számnál. Itt is látszik, hogy a „cmp” parancs után R5 R4 sorrendben állnak a regiszterek. Az R15 regiszterben akkor tárolja el a program a 4-es számot, ha a Negativ flag értéke nulla az összehasonlítás után.

A program futását lépésenkénti ellenőrzéssel vizsgáltam meg. *(13-15. ábra)*



13. ábra
Az R4-es és R5-ös regiszterbe bekerültek a kívánt értékek



14. ábra
A Zero flag nulla maradt az elvárásainknak megfelelően

15. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 4-es számjegy

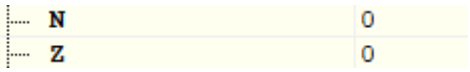
Első esetben felhasznált programkód (R4 <= R5):

```
    mov.b #6, R4
    mov.b #7, R5
    cmp R4,R5
    jl NoAction
    mov #5, R15

NoAction:
    ret
```

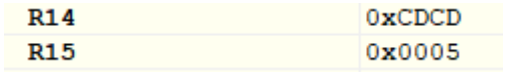
Ebben az esetben egyszerűen megcseréltem az összehasonlítás parancs során a két regisztert (úgy, mint az előző részben használt program készítése során), így most a program az R5 >= R4 relációt vizsgálja, és minden pontosan megegyezik ezen kívül az előzőkben leírtakkal.

Ebben az esetben is debuggolással ellenőriztem. *(16-18.ábra)*

16. ábra
Az R4-es és R5-ös regiszterbe bekerültek a kívánt értékek



17. ábra
A Zero flag nulla maradt az elvárásainknak megfelelően



18. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 5-ös számjegy

Program futtatása során nem találtam hibát, minden az elvárásaimnak megfelelően történt. A program helyesen kezeli a nem megfelelő paramétereket is.

VII. „ÉS” FELTÉTEL HASZNÁLATA

Ebben a részben megírt két program két-két számot hasonlít össze és ezek kapcsolatának függvényében tárol el egy általam választott számot az R15-ös regiszterben. Itt már az előzőektől eltérő módon nem kettő, hanem négy regiszterre volt szükség a számok tárolásához. A két új regiszter az R6 és az R7.

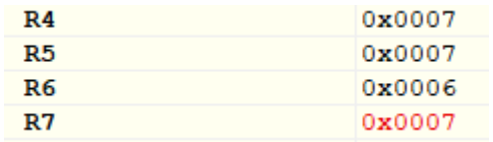
Az első program eltárol egy hatos számot az R15-ben, ha az R4 és az R5 regiszterben tárolt számok megegyeznek, és emellett az R6-ban tárolt szám kisebb, mint az R7-ben tárolt. Ellenkező esetben a program kilép.

Első esetben felhasznált programkód:
(R4==R5 && R6 < R7)

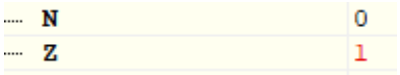
```
mov.b #7, R4
mov.b #7, R5
mov.b #6, R6
mov.b #7, R7
cmp R4,R5
jnz NoAction
cmp R7,R6
jge Action
mov #6, R15
Action:

NoAction:
ret
```

Ahhoz, hogy ez a program helyesen működjön, az előző fejezetekben leírt és használt részeket kellett alkalmazni. Az egyik ilyen volt az egyenlőség vizsgálata a „jnz” paranccsal, a másik pedig az egyenlőség nélküli reláció vizsgálata a „jge”. Ezek működési elvét fentebb taglaltam. Ebben az esetben a program csak akkor tárol el az R15-ös regiszterben 6-ot, ha mind a két feltétel igaz (Hamis-Igaz, Igaz-Hamis, Hamis-Hamis esetekben kilép).

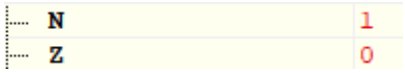
A programot minden létező igaz-hamis kombinációra teszteltem és megfelelően működött. A következő ábrák azt az esetet mutatják, amikor mind a két feltétel igaz és az R15-ben megjelenik a hatos szám. *(19-22. ábra)*



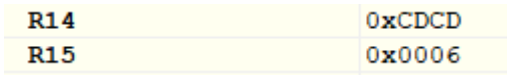
19. ábra
A regiszterekbe bekerültek a kívánt értékek



20. ábra
Az elvárásoknak megfelelően a Zero flag 1-re váltott az R4 és R5 egyenlőségének vizsgálata után



21. ábra
Az elvárásoknak megfelelően a Zero flag ujra 0 lett és a Negatív flag 1-re váltott az R6 és R7 relációjának vizsgálata után



22. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 6-os számjegy

Az második program eltárol egy hetes számot az R15-ben, ha az R4 és az R5 regiszterben tárolt számok megegyeznek, és emellett az R6-ban tárolt szám szigorúan nagyobb, mint az R7-ben tárolt. Ellenkező esetben a program kilép.

Első esetben felhasznált programkód:
(R4==R5 && R7 < R6)

```
mov.b #7, R4
mov.b #7, R5
mov.b #7, R6
mov.b #6, R7
cmp R4,R5
jnz NoAction
cmp R6,R7
jge Action
mov #7, R15
Action:

NoAction:
ret
```

Ahogy azt már tapasztaltuk a relációvizsgálat során, itt is egy egyszerű cserével a „cmp” parancs után átalakítható úgy a programot, hogy az a feltételnek megfelelően működjön. Ebben az esetben is csak akkor tárolja el a hetest a regiszterben, ha mind a két feltétel egyszerre igaz. Ezt azzal értem el, hogy az Action és a NoAction is kiléptet.

A programot ebben az esetben is minden létező igaz-hamis kombinációra teszteltem és megfelelően működött. A következő ábrák azt az esetet mutatják, amikor mind a két feltétel igaz és az R15-ben megjelenik a hetes szám. *(23-26. ábra)*

R4	0x0007
R5	0x0007
R6	0x0007
R7	0x0006

23. ábra
A regiszterekbe bekerültek a kívánt értékek

N	0
Z	1

24. ábra
Az elvárásoknak megfelelően a Zero flag 1-re váltott az R4 és R5 egyenlőségének vizsgálata után

N	1
Z	0

25. ábra
Az elvárásoknak megfelelően a Zero flag újra 0 lett és a Negatív flag 1-re váltott az R6 és R7 relációjának vizsgálata után

R14	0xCDCD
R15	0x0007

26. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 7-es számjegy

VIII. „VAGY” FELTÉTEL HASZNÁLATA

Ebben a részben megírt két program szintén két-két számot hasonlít össze és ezek kapcsolatának függvényében tárol el egy általam választott számot az R15-ös regiszterben. Előző fejezettől eltérően itt „vagy” kapcsolatot alkalmaztam a két feltétel között, ami azt eredményezte, hogy minden esetben eltárolja a választott számot, kivéve, ha mind a két feltétel egyszerre hamis. (Más szóval, ha legalább az egyik feltétel igaz, akkor lefut a program.)

Az első program eltárol egy nyolcas számot az R15-ben, ha az R4 és az R5 regiszterben tárolt számok megegyeznek, vagy az R6-ban tárolt szám kisebb, mint az R7-ben tárolt. Ellenkező esetben a program kilép.

Első esetben felhasznált programkód:
(R4==R5 || R6 < R7)

```
mov.b #7, R4
mov.b #7, R5
mov.b #6, R6
mov.b #7, R7
cmp R4,R5
jnz NoAction
mov #8, R15
NoAction:
cmp R7,R6
jge Action
mov #8,R15
Action:

ret
```

Az előző fejezthez hasonlóan itt is ahhoz, hogy ez a program helyesen működjön, az előző fejezetekben leírt és használt részeket kellett alkalmazni. Az egyik ilyen volt az egyenlőség vizsgálata a „jnz” paranccsal, a másik pedig az egyenlőség nélküli reláció vizsgálata a „jge”. Ezek működési elvét fentebb taglaltam. Az előző fejezetben látható volt, hogy az „Action” is kiléptette a programot csak úgy, mint a „NoAction”. Viszont ebben azt kértem a programtól, hogy ha a „jge” utasítás teljesül, akkor vizsgálja csak meg az R6 és R7 relációját, ezzel leprogramozva a „vagy” feltételt.

A programot ebben az esetben is minden létező igaz-hamis kombinációra teszteltem és megfelelően működött. A következő ábrák azt az esetet mutatják, amikor csak az első feltétel igaz és az R15-ben megjelenik a nyolcas szám. *(27-28. ábra)*

R4	0x0007
R5	0x0007
R6	0x0007
R7	0x0007

27. ábra
A regiszterekbe bekerültek a kívánt értékek

R15	0x0008

28. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 8-as számjegy

A második program eltárol egy kilences számot az R15-ben, ha az R4 és az R5 regiszterben tárolt számok megegyeznek, vagy az R6-ban tárolt szám nagyobb, mint az R7-ben tárolt. Ellenkező esetben a program kilép.

Első esetben felhasznált programkód:
(R4==R5 || R6 > R7)

```
        mov.b #7, R4
        mov.b #7, R5
        mov.b #7, R6
        mov.b #6, R7
        cmp R4,R5
        jnz NoAction
        mov #9, R15
    NoAction:
        cmp R6,R7
        jge Action
        mov #9,R15
Action:

        ret
```

Ahogy azt már sokszor tapasztaltuk a relációvizsgálat során, itt is egy egyszerű cserével a „cmp” parancs után átalakítható úgy a programot, hogy az a feltételnek megfelelően működjön.

A programot szintén az összes esetben teszteltem és megfelelően működött. A következő ábrák azt az esetet szemléltetik, amikor csak a második feltétel teljesül. *(29-30. ábra)*

R4	0x0008
R5	0x0007
R6	0x0007
R7	0x0006

29. ábra
A regiszterekbe bekerültek a kívánt értékek

R14	0xCDCD
R15	0x0009

30. ábra
A programnak megfelelően az R15-ös regiszterbe bekerül az 9-es számjegy

IX. OSZTHATÓSÁG

Ebben a részben megírt program egy tízes számot ad értékül az R15 regiszternek abban az esetben, ha R4 osztható R5-tel. Ellenkezőleg a program leáll.

Felhasznált programkód:

```
    mov #8, R4
    mov #4, R5
    mov R4, R12
    mov R5, R14
    call #divide
    cmp #0,R14
    jnz Action
    mov #10, R15

Action:
    ret
```

Ebben a programban a „divide” függvényt használtam, amely segítségévek el tudtam osztani a két számot egymással. Ahhoz, hogy osztható legyen egy szám a másikkal, az kell, hogy a maradék tag nulla legyen. Ezt a „jzn” paranccsal vizsgáltam meg. Az osztás eredményét a program futtatása után az R12 regiszterben kapjuk meg.

A program helyességet lépésenkénti futtatással ellenőriztem. *(31-33. ábra)*

R4	0x0008
R5	0x0004
R6	0xCDCD
R7	0xCDCD
R8	0xCDCD
R9	0xCDCD
R10	0xCDCD
R11	0xCDCD
R12	0x0008
R13	0xCDCD
R14	0x0004
R15	0xCDCD

31. ábra
A regiszterekbe bekerültek a kívánt értékek

N	0
Z	1
C	1

32. ábra
A Carry flag és a Zero flag is az elvártaknak megfelelően 1-re vált

R12	0x0002
R13	0x0000
R14	0x0000
R15	0x000A

33. ábra
Bekerültek a várt eredmények a megfelelő regiszterekbe

Az ellenőrzés során mindent rendben találtam.

X. PÁROS SZORZAT

Ebben a részben megírt program egy tizenegyet ad értékül az R15 regiszternek abban az esetben, ha R4 és R5 szorzata páros. Ellenkezőleg a program leáll.

Felhasznált programkód:

```
mov #1, R4
mov #2, R5
mov R4, R12
mov R5, R14
call #multiply
mov #2,R5
call #divide
cmp #0,R14
jnz Action
mov #11, R15

Action:
  ret
```

Ennél a programnál a „multiply” függvényt használtam a két szám szorzására. Annak érdekében, hogy megállapítsam, hogy az szorzat páros-e, megvizsgáltam a szorzat és kettő hányadosának maradékát. Ha ez a maradék 0, akkor a szorzatuk páros, így a program eltárolja a tizenegyet az R15-ben. A szorzás eredményét ebben az esetben is az R12 regiszterben láthatjuk.

A program helyességet ebben az esetben is lépésenkénti futtatással ellenőriztem. *(34-36. ábra)*

R4	0x0001
R5	0x0002
R6	0xCDCD
R7	0xCDCD
R8	0xCDCD
R9	0xCDCD
R10	0xCDCD
R11	0xCDCD

34. ábra
A regiszterekbe bekerültek a kívánt értékek

N	0
Z	1
C	1

35. ábra
A Carry flag és a Zero flag is az elvártaknak megfelelően 1-re vált

R12	0x0002
R13	0xCDCD
R14	0xCDCD
R15	0x000B

36. ábra
Bekerültek a várt eredmények a megfelelő regiszterekbe

A program ellenőrzése során nem találtam hibát.

FELHASZNÁLT FORRÁSOK

Mikrovezérlők II mérési segédlet

Mérési utasítások

Szoftvertechnológia, flagek

SZÁMÁBRÁZOLÁS, MIKROKONTROLLER ELŐADÁS