# Mikrokontroller alaputasítások

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, egyszerű alapműveletek végrehajtása. A kettes komplemens bináris összeadás illetve kivonás lehetőségeinek, valamint a műveletek végrehajtása során keletkező jelző (flag) bitek jelentésének elsajátítása.**

## I. A JEGYZŐKÖNYVBEN HASZNÁLT FOGALMAK

Gépi számábrázolás: a számok (számító)gépek memóriájában vagy egyéb egységében történő tárolását vagy valamely adathálózaton történő továbbítás formátumát adja meg.

Nem negatív egész számok: kettes számrendszerbe átváltott forma. Fix hosszon, egész byte méretű ábrázolás (nullákkal kiegészítve). Legkisebb ábrázolható érték N biten a 0, legnagyobb a $2^N - 1$.

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.

Kettes komplemens ábrázolás: a negatív számot eggyel növeljük, majd az így kapott szám abszolút értékét binárisan ábrázoljuk a megadott számú biten, végül az így kapott számjegyeket negáljuk (0 helyett 1-et, 1 helyett 0-t írunk).

Számrendszer: a szám írott formában történő megjelenítésére alkalmas módszer.

Számrendszer alapja és a számjegyek: ha a számrendszer A alapú, akkor a legkisebb felhasználható számjegy a 0, a legnagyobb pedig az A-1.

Gyakran használt számrendszerek:

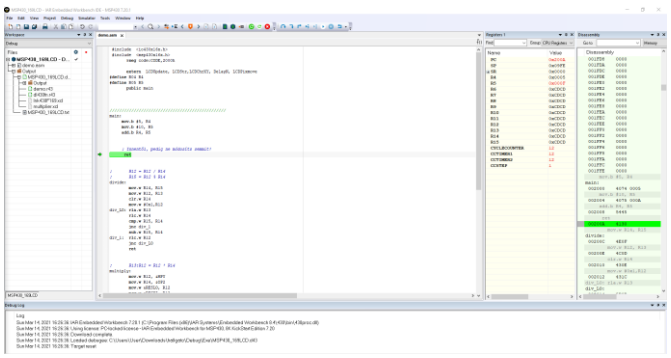
| **Számrendszer** | **Alap** | **Jele** | **Példa** |
|---|---|---|---|
| Decimális | 10 | | 159 |
| Bináris | 2 | [2] vagy b | $100_{[2]}, 0b100$ |
| Oktális | 8 | [8] vagy 0 | $63_{[8]}, 063$ |
| Hexadecimális | 16 | [16] vagy 0x vagy h | $345_{[16]}, 0x243$ |

Hexadecimális számrendszerben ábrázolás során felhasználunk betűket is a következő módon: 10…15 - A…F.

## II. A MUNKAFELÜLET LÉTREHOZÁSA

A mikrokontroller programozásához egy IAR Embedded Workbench nevű programot használtam, melynek segítségével az MSP430F169 típusú mikrokontrollerrel végzett különböző műveleteket szimuláló programokat írhattam.

A program indítása és a laboron során készen kapott file-ok betöltése után a következő felület volt látható. *(1. ábra)*



*1. ábra*
A kezelői felület

Az alkalmazás középső, legnagyobb ablaka szolgál arra, hogy megírjam a mikrokontroller vezérléséhez szükséges programkódot. Baloldalon látható az úgynevezett „Registers 1” ablak, amelyen látható az egyes regiszterekben szereplő értékek hexadecimális számrendszerbe átváltva. Ez az ablak nagyon fontos volt a számomra, mert a debugger használata során itt láttam lépésről lépésre, hogy mi történt az adott program futtatása során. Alul látható a „Build/Debug Log” ablak, amely a hibás futás során jelzéseket ad/ naplózza program fejlesztése során történt eseményeket.

## III. ELŐJEL NÉLKÜLI 8 BITES SZÁMOK ÖSSZEADÁSA

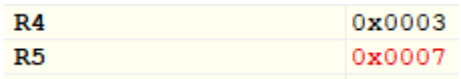
Ebben a részben használt program összeadja az egy-egy regiszterbe helyezett konstansokat (összeadandókat).

Felhasznált programkód:

```
mov.b #3, R4
mov.b #7, R5
add.b R5,R4
```

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 regiszterben.

A program működését lépésenkénti (úgynevezett debuggolással) ellenőriztem. *(2-3. ábra)*




*2. ábra*
Az R4-es és az R5-ös regiszterbe bekerültek az összeadandók


*3. ábra*
Az R4-es regiszterbe bekerültek az eredmény

Tudjuk, hogy a program az egyes regiszterekben lévő számokat hexadecimális, azaz tizenhatos számrendszerben ábrázolja. Ellenőrzésképpen, ha a 0x000A számot decimális számrendszerbe átváltom, akkor éppen 10-et kapok, amely megegyezik a 7+3 értékével. Ebben az esetben az eredményből látszik, hogy a létrehozott program helyesen működött.

Észrevehető, hogy az ábrázolt számok mindegyikénél az első két helyérték mindig nulla volt. Ez annak köszönhető, hogy az utasítások hívószava után szereplő „.b” jelölés 8-ra redukálta a felhasználható bitek számát. Tudjuk, hogy 8 biten a legnagyobb ábrázolható szám, az I. pontban leírtak alapján, 255. Ha úgy adtam volna meg a két összeadandó konstans, hogy az összegük nagyobb legyen ennél a számnál, akkor túlcsordulást tapasztaltam volna.

Például:

```
mov.b #255, R4
mov.b #1, R5
add.b R5, R4
```

Ennek az összeadásnak az elvégzése során azt tapasztaltam, hogy a 256 helyett 0-t kaptam eredményül, amely azt bizonyítja, hogy a művelet elvégzése során túlcsordulás történt, ekkor a carry bit értéke 1-re változott. A helyes érték a kapott értéktől $2^8$-nal tér el (a kapott értékhez hozzá kell adni 256-ot).

## IV. Előjel nélküli 16 bites számok összeadása

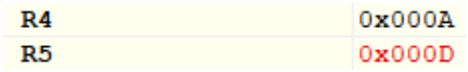
Ebben az esetben a program nagyon hasonlít a III. pontban használt programhoz. Az eltérés a két program között az, hogy ebben az esetben nem 8, hanem 16 bites számokkal dolgoztam, így az legnagyobb ábrázolható szám 65 535.

Felhasznált programkód:

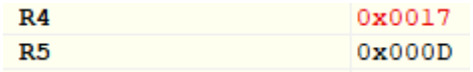
```
mov #10, R4
mov #13, R5
add R5,R4
```

A fent leírt kód szinte ugyanúgy működik, mint az előző esetben, annyi különbséggel, hogy mivel itt már 16 biten dolgoztam, nem volt szükség a „.b” jelzésre.

A program működését ebben az esetben is lépésenkénti futtatással ellenőrizetem. *(4-5. ábra)*

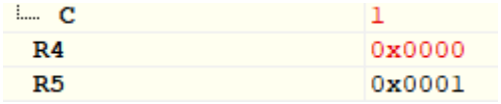



*4. ábra*
Az R4-es és az R5-ös regiszterbe bekerültek az összeadandók




*5. ábra*
Az R4-es regiszterbe bekerültek az eredmény

Ebben az esetben túlcsordulás során (az összeadás eredménye nagyobb, mint 65 534) a helyes eredmény $2^{16}$-nal fog eltérni a kapott eredménytől. A carry bit értéke ebben az esetben is 1-re vált. *(6. ábra)*




*6. ábra*
A carry bit (C) értéke túlcsordulás során

## V. Előjel nélküli 32 bites számok összeadása

Ebben a programban már új dolgot kellett használnom, eltérőt, mint az eddigiekben. Mivel a csak 16 bites regiszterek állnak rendelkezésre, így az összeadandó konstansokat két-két regiszterben kellett eltárolni. Első két regiszterben eltároltam a két szám kisebb helyiértékű részét, majd kiszámoltam ezek összegét. Újabb két regiszterben eltároltam az összeadandók nagyobb helyiértékű részét, majd ezek összegét is kiszámoltam egy új parancs segítségével, így megkapva két 32 bites konstans összegét.

Felhasznált programkód:

```
mov #6, R4
mov #3, R5
add R5,R4
mov #10, R6
mov #11, R7
addc R7,R6
```

Észrevehető, hogy a második összeadás parancsnál nem az „add”, hanem az „addc” utasítást használtam, melyben a „c” arra utal, hogy ez az összeadás figyelembe veszi a carry bitet is. Ez azért fontos, mert ezzel gondoskodtam arról az esetről, ha az első összeadás során túlcsordulás keletkezik, akkor az megjelenjen a második művelet végzése során.

A program működését ebben az esetben is lépésenkénti futtatással ellenőrizetem.

A számolás során a végerdményben csak akkor lesz túlcsordulás, ha a második összeadás túlcsordul, ebben az esetben a carry bit 1-re vált. Túlcsordulás esetén a helyes eredmény $2^{32}$-nel fog eltérni a helyes eredménytől.

## VI. Összeadás előjeles környezetben

Előjeles környezetben a programok létrehozása során megírt kódok nem változnak, csak arra kell figyelni, hogy a számok tárolása során az előjelet meg kell adni. Előjeles összeadás során kettes komplemens számábrázolást használunk, ami azt jelenti, hogy a szám előjelének tárolására az adott bitek számából lefoglalunk egyet (legnagyobb helyiértékű bitet). Ha ennek a bitnek az értéke 0, akkor a szám nem negatív, ha a bit értéke 1, akkor a szám negatív.

Kettes komplemes használata során változnak az ábrázolható tartománya. Kettes komplemens során az ábrázolható tartomány N biten:

- legkisebb szám: $-(2^{N-1})$
- legnagyobb szám: $2^{N-1} - 1$

Ezzel a formulával megkaptam az egyes előjeles összeadások tartományát:

- 8 bites: $-(2^7)$-től $2^7 - 1$-ig [-128 ; 127]
- 16 bites: $-(2^{15})$-től $2^{15} - 1$-ig [-32768 ; 32767]
- 32 bites: $-(2^{31})$-től $2^{31} - 1$-ig

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 |

*7. ábra*
Különböző flagek értékének megjelenítése a programáltal

## VII. Előjel nélküli 8 bites számok kivonása

Kivonás során a szintaktika teljes mértékben megegyezik az összeadás során használtakkal, egyedül az „add” utasítás helyett a kivonás, azaz „sub” parancsot kell alkallmazni.

Felhasznált programkód:

**mov.b #3, R4**
**mov.b #7, R5**
**sub.b R5,R4**

A program ebben az esetben is debuggolással ellenőriztem. *(8-9.ábra)*

| | |
|---|---|
| R4 | 0x0003 |
| R5 | 0x0007 |

*8. ábra*
Az R4-es és az R5-ös regiszterbe bekerültek a konstansok, melyeknek a különbségére kíváncsiak vagyunk

| | |
|---|---|
| R4 | 0x00FC |

*8. ábra*
Az R4- regiszterbe bekerültek a különbség

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. *(10. ábra)*

| | |
|---|---|
| N | 1 |
| Z | 0 |
| C | 0 |
| R4 | 0x00FC |
| R5 | 0x0007 |

*10. ábra*
Az előző kivonás során az negatív eredményt kaptam, így az N flag értéke 1 lett

## VIII. Előjel nélküli 16 bites számok kivonása

Ebben az esetben a programkód majdnem megegyezik a VII. pontban használt kóddal, csak el kell távolítani „.b” utasításokat, amelyek arra utaltak, hogy 8 bites számokkal dolgozunk.

Felhasznált programkód:

**mov #3, R4**
**mov #7, R5**
**sub R5,R4**

Ebben az esetben is kettes komplemensként kell kezelni a nullánál kisebb eredményeket, ilyenkor az N flagértéke 1-re vált.

## IX. Előjel nélküli 32 bites számok kivonása

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. Újabb két regiszterben eltároltam a konstansok nagyobb helyiértékű részét, majd ezek különbségét is kiszámoltam egy új parancs segítségével, így megkapva a keresett értéket.

Felhasznált programkód:

```
mov #6, R4
mov #3, R5
add R5,R4
mov #10, R6
mov #11, R7
subc R7,R6
```

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 aparancs figyelembe veszi a carry bitet 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.

## X. Kivonás előjeles környezetben

Hasonlóan az összeadáshoz, itt is megegyeznek a programkódok az előjel nélküli párjukhoz, de itt jelölni kell a konstansok előjelét.

Kivonásra is ugyanúgy teljesül az ábrázolási tartomány, mint összeadásra.

Ebben az esetben is igaz az, hogy ha negatív számot kaptunk végeredményül, akkor az N flag 1-re vált és kettes komplemensként kell kezelni. Tovább az is teljesül, hogy az overflow flag jelzi, ha a művelet elvégzése során az ábrázolási tartományban nem tartozik bele a kapott érték. A carry bit pedig pont ellentétesen működik, mint azt megszokhattuk az összeadásnál, itt akkor vált 1-re, ha negatív irányból történik túlcsordulás.

## Felhasznált források


Számrendszerek és számábrázolás

Mérési segédlet

Számábrázolás, mikrokontroller előadás

Számrendszerek közötti átváltás

Mérési utasítások