Digitális technika 2.

BMEVIIIAA06

5. előadás

Assembly programozás

(további utasítások, elágazások, ciklusok)

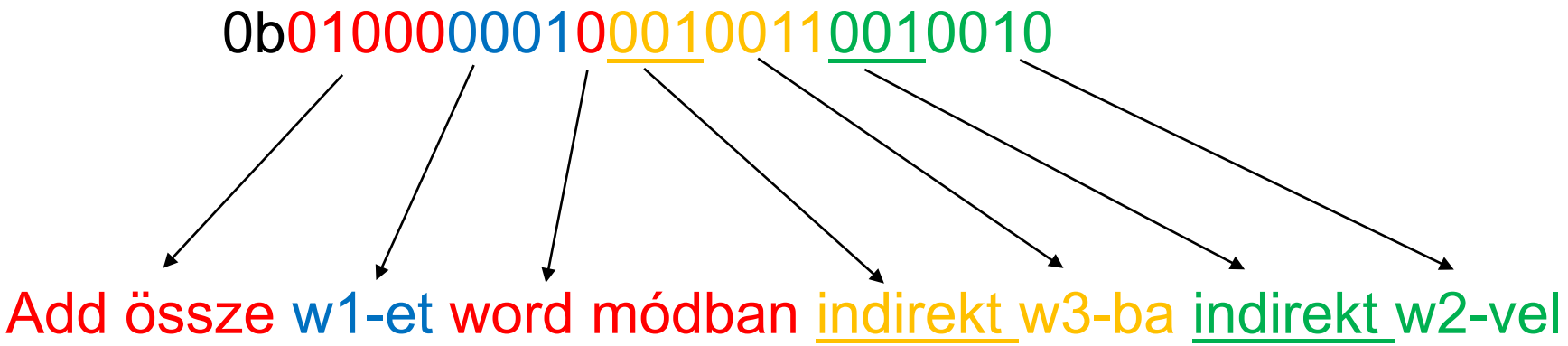
Láttuk eddig:

- A CPU csak utasításokat hajt végre
- Minden utasítás egy-egy szám a memóriában
  - » Mi írhatjuk oda, de megjegyezhető formában
- Az assembler program feladata, hogy a megjegyezhető „utasítás neveket” (mnemonik) számokra (opkód) cserélje
- Láttunk két utasítást: mov és add
- Láttuk, hogy ugyanannak az utasításnak több formája van az operandusoktól függően

Utasítások kódolása

add w1, [w2], [w3] ; [w3]=w1+[w2]

add w1,[w2],[w3]

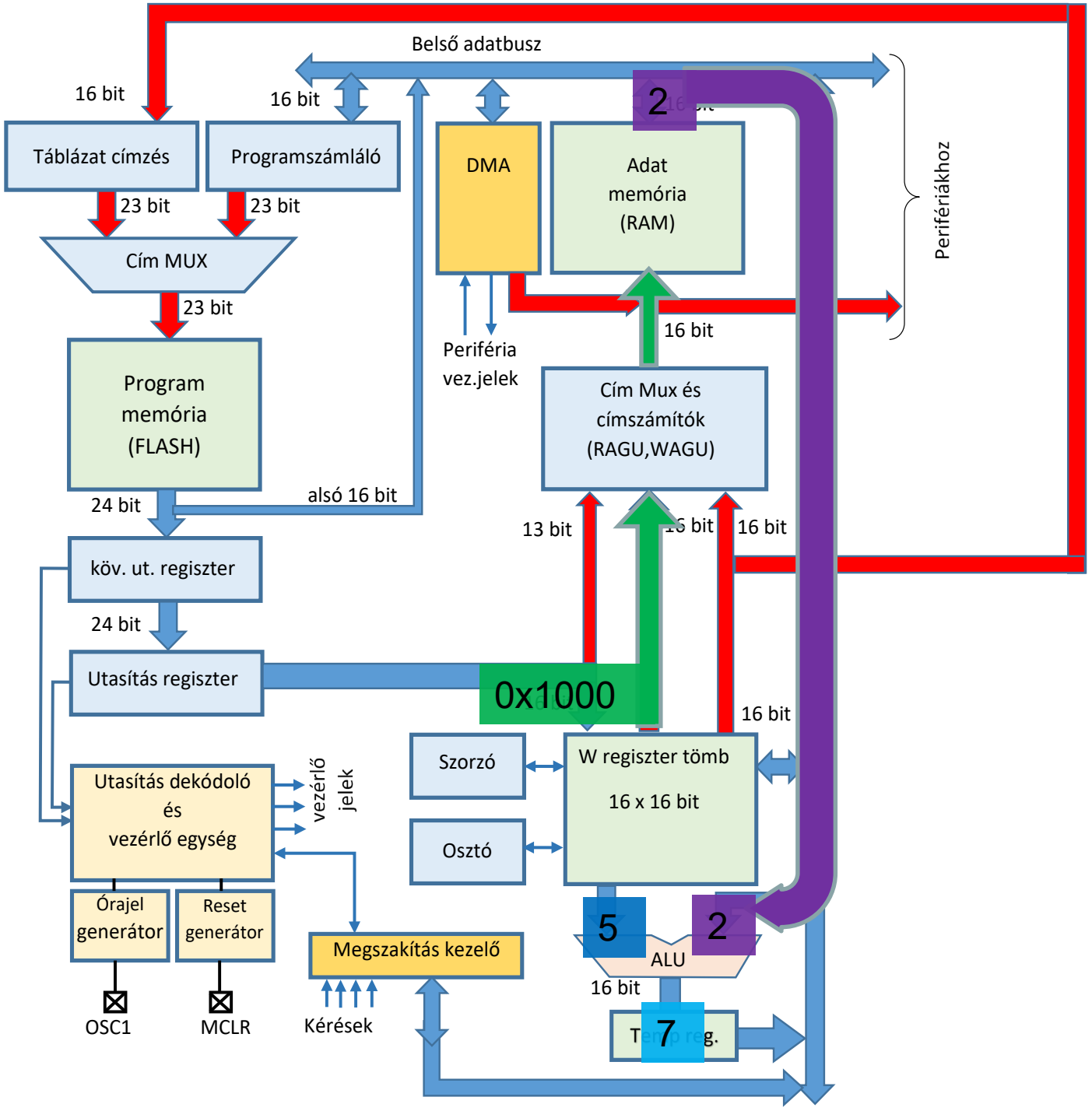


Végrehajtás lépései

add w1,[w2],[w3]

- W1=5
- W2=0x1000
- W3=0x2000
- [W2] tartalma: 2

1. Fázis: [W2] olvasása

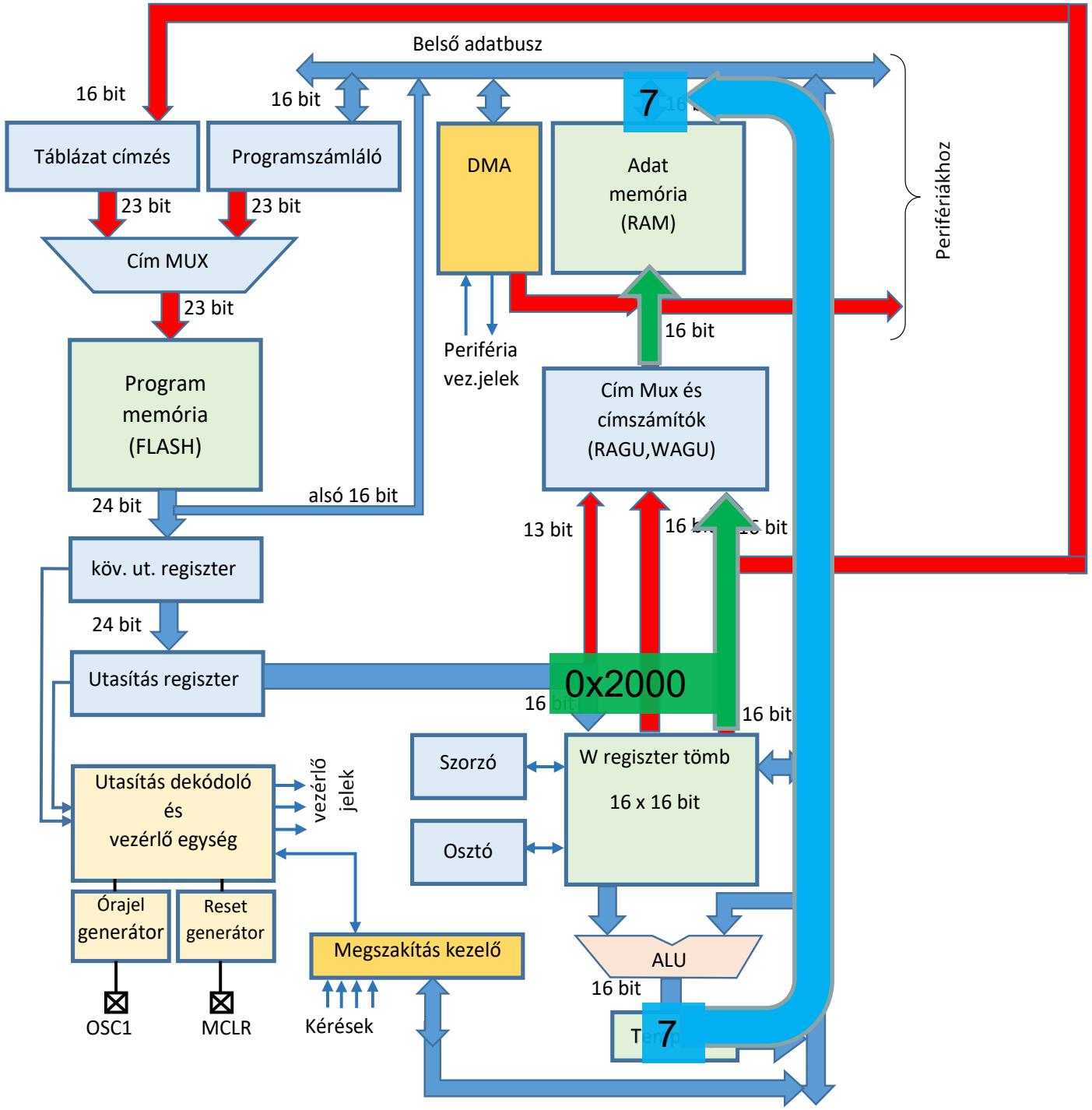


Végrehajtás lépései

add w1,[w2],[w3]

- W1=5
- W2=0x1000
- W3=0x2000
- [W2] tartalma: 2

2. Fázis: [W3] írása



Változók megadása

- Láttuk, hogy a memóriacímeket akár be is írhatjuk a programba közvetlenül (pl. mov 0x1000,w0)
- C-ben megszoktuk, hogy vannak változók (pl. i, atlag, stb.)
- Az assembler a mov-ot is ki tudja cserélni „0b10000”-re…

- Változók definiálása:

1. Nevezzük meg a memóriát ahova a változó kerül
2. Adjuk meg a kezdőcímét, a memórián belül (ha nem az előző után szeretnénk)
3. Adjuk meg hány bájt helyet foglal

Változók megadása

Számformátumok:

- A C nyelven megszokott szabályok érvényesek
- Alapértemlezésben decimális (pl. 10 → decimális tíz)
- 0x: hexadecimális (pl. 0x10=16 → decimális tizenhat)
- 0b: bináris (pl. 0b10=2 → decimális kettő)

- *Vezető 0 oktális számábrázolást jelent, lehetőleg ezt ne használjuk (pl. 010 = 8).*

Változók megadása

```
.bss
.org 0x1000
        atlag:      .space 2
        osszeg:     .space 4
.org 0x1100
        tomb:       .space 400
```

- Változók definiálása:

1. Nevezzük meg a memóriát ahova a változó kerül
2. Adjuk meg a kezdőcímét, a memórián belül (ha nem az előző után szeretnénk)
3. Adjuk meg hány bájt helyet foglal

.bss — **block starting symbol,** a nem inicializált adatok helye általában
Az adott mikrokontroller esetében a RAM terület elején kezdődik.
A ***RAM a 0x800-as címen kezdődik*** az adatbuszon (előtte perifériák)

.org *n* — **origin,** a következő sorban lévő dolgot ettől a címtől kezdje
az adott memórián belül érvényes, azaz a RAM esetén 0x800+n címet jelent
ha nincs megadva, a következő üres helyre kerül a következő változó
ha sosem volt megadva akkor a legelsőn, a 0x800-on

.space m — Ennyi helyet hagyjon a következő változónak

Változók megadása

```
.bss
.org 0x1000
        atlag:      .space 2
        osszeg:     .space 4
.org 0x1100
        tomb:       .space 400
```

- Változók definiálása:

1. Nevezzük meg a memóriát ahova a változó kerül
2. Adjuk meg a kezdőcímét, a memórián belül (ha nem az előző után szeretnénk)
3. Adjuk meg hány bájt helyet foglal

A megadott változók:

atlag: 0x1800 címen kezdődő 2 bájtos változó (kisebb helyiértékű bájtja 0x1800, nagyobb: 0x1801)

osszeg: 0x1802 címen kezdődő 4 bájtos változó (azaz a 0x1805-ig tart, ott a legnagyobb helyiértéke van)

tomb: összefüggő adatterület ami a 0x1900 címtől kezdődik és 400 bájt hosszú

rajtunk múlik hogy használjuk, de lehet pl. egy 200db 16 bites szóból álló tömb

→ int16_t tomb[200]

Változók megadása

```
.bss
.org 0x1000
        atlag:      .space 2
        osszeg:     .space 4
.org 0x1100
        tomb:       .space 400
```

- Változók definiálása:

1. Nevezzük meg a memóriát ahova a változó kerül
2. Adjuk meg a kezdőcímét, a memórián belül (ha nem az előző után szeretnénk)
3. Adjuk meg hány bájt helyet foglal

A megadott változók címei:

atlag: 0x1800

osszeg: 0x1802

tomb: 0x1900

.bss
.org
.space

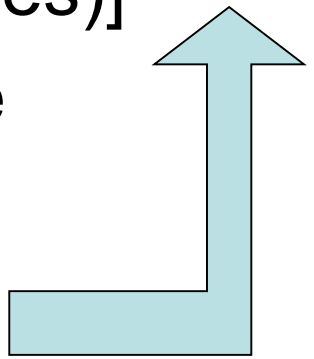
Ezek nem utasítások, hanem a fordítónak szóló **„direktívák”**

→ az utasítás mindig gépi kódra fordul, a direktíva a fordítást vezérli

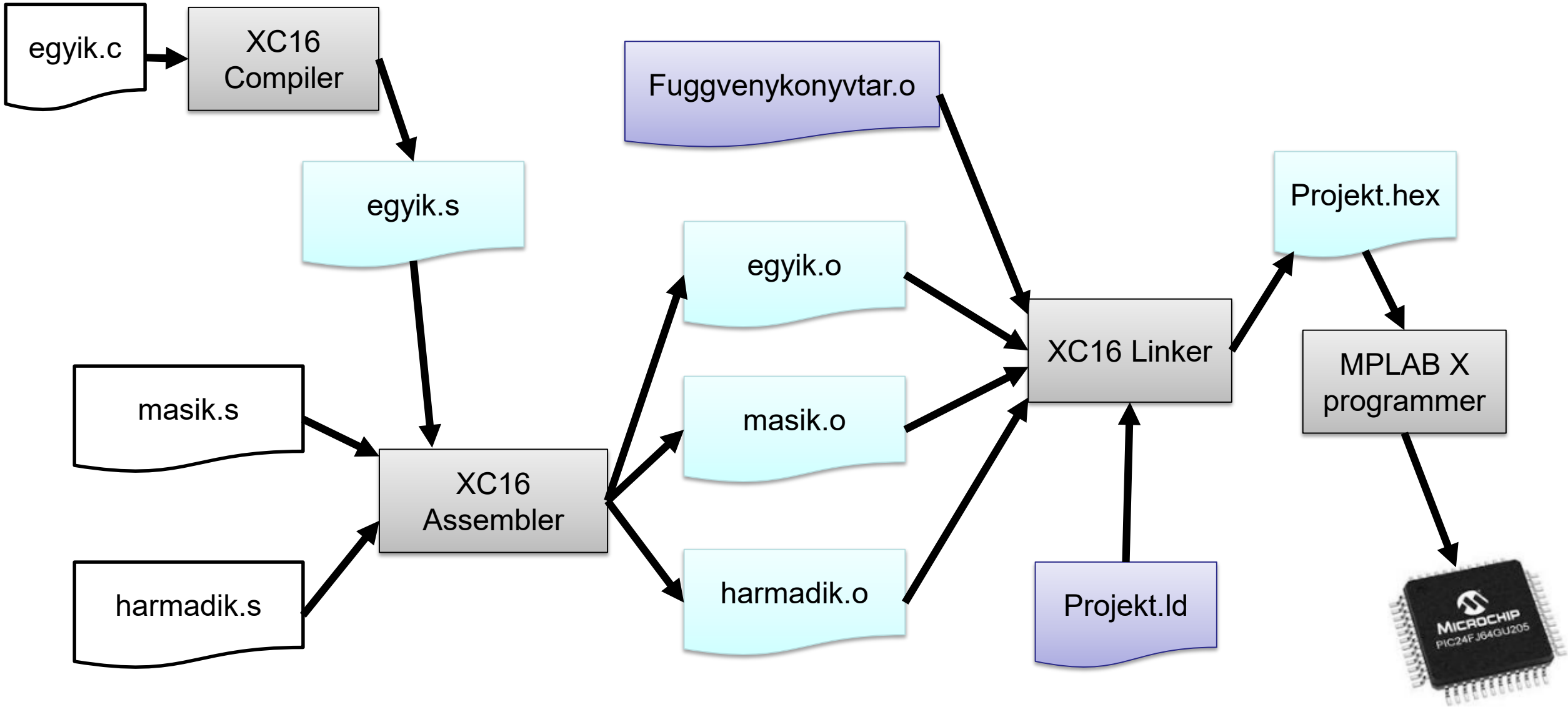
Ezután ha bárhova a változók neveit írjuk, az pont olyan, mintha ezeket a számokat (kezdőcímüket) írnánk oda.

A fejlesztés folyamata

- Program írás általában PC-n →szövegszerkesztő a kód elkészítéséhez
- Szövegfájl fordítása (cross-assembler)→bináris tartalom előállítása
- [több programmodul esetén összefűzés (linkelés)]
- Beprogramozás („égetés”) a mikrokontrollerbe vagy a szimulátorba
- Hibakeresés, javítás



Hogyan kerül a program a memóriába?



Fontosabb fogalmak

- Mnemonik→utasítás rövid neve
- Gépi kód, OP. kód
- Direktíva→fordítónak szóló *„utasítás”*
- Szimbólum→érték helyettesítés
  - Címke→belépési pont, hivatkozási pont
  - Konstans→állandó érték megadása
- Szintaxis→előírt formátum pl.:
  `címke: **utasitás** operandus1,… operandusn ; komment`
- Assembler:
  - Az assembly forrásfájlt gépi kódra fordítja
- Compiler:
  - A C (vagy más magas szintű) nyelvi forrásfájlt assembly kódra fordítja

    (van olyan compiler is, ami egyből gépi kódot fordít)
- Linker:
  - A különböző tárgykód fájlokat összeszerkeszti („link”-eli) és kialakítja a futtatható memóriatartalmat

Fontosabb fogalmak 2

- Tárgykód fájl (.o fájl, object code):
  - fájlonként lefordított gépi kód, amiben még nincs minden szimbólum behelyettesítve
  - Előfordul, hogy bizonyos függvénykönyvtárakat eleve így kapunk
- Linker script (.ld fájl):
  - utasítások a linkernek: pl. hol kezdődik a memória az adott eszközben, hova lehet tenni programot, stb.
  - Általában az eszköz/processzor gyártója adja

    (de előfordul, hogy bele kell nyúlni… ebben a tárgyban nekünk nem kell…)
- .hex fájl:
  - A memóriába kerülő tartalom leírása
  - Utasításokat tartalmaz a programozó eszköznek
    - » „Recept” memória beprogramozására

További utasítások

Utasítás csoportok:

- Adatmozgató utasítások (ezeket láttuk)
- **Aritmetikai műveletek**
- Logikai műveletek
- Bitműveletek
- Vezérlés átadás
- Vezérlő műveletek

Aritmetikai műveletek

- Összeadást láttuk
  - » add wb, ws, wd ;(wd=wb+ws)
- Kivonás
  - » sub wb, ws, wd ;(wd=wb-ws)
- Növelés
  - » inc valtozo ;valtozo+=1
- Csökkentés
  - » dec valtozo ;valtozo-=1
- Szorzás
  - » mul(.ss,.us,.uu) wb,ws,wd ;wd:wd+1=wb*ws, wd csak páros lehet
- Osztás (később)

Aritmetikai műveletek

- Az aritmetikai műveletek vagy bájtos (8 bit) vagy word-ös (16 bit) adatokkal számolnak
  - add.b w0,w1,w2, vagy add w0, w1, w2

- Ha nagyobb méretű számokkal számolnánk, egy utasítás nem elég
- Meg kell jegyezni az átvitelt a két művelet között
  - » Erre szolgál a C (Carry, átvitel) flag
    - » Emlékszünk: A flagek a Státusz Regiszterben (SR) vannak:

SR															
-	-	-	-	-	-	-	DC	P2	P1	P0	RA	N	OV	Z	C

Aritmetikai műveletek

- Az aritmetikai műveletek változtatják a C flaget:
  - Összeadás akkor állítja 1-be, ha az utolsó (15-ös) bitből átvitel történt
  - Kivonás 1-be állítja, ha a kisebbítendő nagyobb a kivonandónál (eredmény pozitív lesz)
    - » *A kivonás igazából a kivonandó kettes komplemensének hozzáadása (a C csak az összeadó átvitel kimenete)*
- Carry-t használó aritmetikai műveletek:
  - add**c** wb, ws, wd ;wd=wb+ws+C
    - » c: carry
  - sub**b** wb, ws, wd ;wd=wb-ws-$\overline{C}$
    - » b: borrow

Magyarul a „carry”-t és a „borrow”-t is átvitelnek vagy maradéknak hívjuk

32 bites aritmetika

- 2 utasítás szükséges minden számhoz
- Hogyan tárolunk hosszabb számokat?
  - A memóriát vagy bájtosan vagy 16 bitesen érjük el
    - » Kisebb címen a kisebb helyiértékű bájt van a 16 bites szóban is
  - Ugyanez a logika érvényes hosszabb számokra
    - » Kisebb helyiértéken kisebb word! *(„little end first”, „little endian”)*
- Pl. a „hosszuszam” változót az 0x1000 címen tároljuk és az értéke épp 0xDEADBEEF:

```
.bss
.org 0x0200
        hosszuszam:     .space 4
```

Cím	Adat
0x1000	0xEF
0x1001	0xBE
0x1002	0xAD
0x1003	0xDE

32 bites aritmetika

- A memóriában először az alsó helyiérték van
  - » Összeadni is hátulról előrefele kell…

C

```
Int32_t egyik,masik, eredmeny;

eredmeny=egyik + masik;
```

Assembly

```
.bss
        egyik:      .space 4
        masik:      .space 4
        eredmeny:   .space 4
.text ; a változók megadása után így váltunk vissza a kód írására
mov egyik,w0
mov masik,w1
add w0, w1, w2  ;beállítja a C-t ha volt carry
mov egyik+2, w0  ;az egyik+2-t az assembler kiszámolja
mov masik+2 w1
addc w0, w1, w3  ;itt használjuk fel, a C-t
mov w2, eredmeny  ;visszarakjuk az eredményt
mov w3, eredmeny+2
```

32 bites aritmetika

- Pontosan ugyanez érvényes kivonásra is

C

```
Int32_t egyik,masik, eredmeny;

eredmeny=egyik - masik;
```

Assembly

```
.bss
        egyik:      .space 4
        masik:      .space 4
        eredmeny:   .space 4
.text ; a változók megadása után így váltunk vissza a kód írására
mov egyik,w0
mov masik,w1
sub w0, w1, w2  ;beállítja a C-t ha volt carry
mov egyik+2, w0  ;az egyik+2-t az assembler kiszámolja
mov masik+2 w1
subb w0, w1, w3  ;itt használjuk fel, a C-t
mov w2, eredmeny  ;visszarakjuk az eredményt
mov w3, eredmeny+2
```

Ha még hosszabb számaink lennének, ugyanezt lehetne ismételni (64 bitnél 4-szer, stb…)

Kiterjesztett aritmetika

- A PIC24 egy 16 bites processzor, tehát egyetlen utasítással vagy 8 vagy 16 bites számokkal dolgozik.
- Ez a Carry flag használatával tetszőlegesen kiterjeszthető
- Nagyobb számokkal számolni mindig lassabb:
  - » 32 biten 2-szer, 64 biten 4-szer, stb…
- Érdemes meggondolni, mekkora számtartományra van szükségünk, és csak akkora változókat használni
- A mai PC 64 bites processzorán ez kevésbé okoz gondot
  - » 64 bites processzor ugyanúgy egy utasítással végzi a 8-16-32-64 bites műveleteket is, de pl. 128 bithez már neki is kétszer annyi utasítás kell
  - » *A mai x86_64 processzoron a 8 és 16 bites műveletek párhuzamosan is elvégezhetők (egy utasítás több kisebb összeadást is elvégezhet egyszerre) ezért mégis érdemes meggondolni, mekkora változót használunk…*

Logikai műveletek

Az ALU-val logikai műveletek is végezhetők

→bitenként (mind a 16 bitre)

- ÉS
  - » and wb, ws, wd ;(wd=wb&ws)
- VAGY
  - » ior wb, ws, wd ;(wd=wb|ws)
- Kizáró vagy (XOR)
  - » xor wb, ws, wd ;(wd=wb^ws)

Logikai műveletek

Példák:

W0	0b1110101111101011
W1	0b0000110011101011

- ÉS
  - » and w0, w1, w2 ;w2 = 0b0000100011101011
- VAGY
  - » ior w0, w1, w2 ;w2 = 0b1110111111101011
- Kizáró vagy (XOR)
  - » xor w0, w1, w2 ;w2 = 0b1110011100000000

Közvetlen adatmegadás

Konstanst csak regiszterbe tudunk mozgatni *(láttuk)*

Tetszőleges memóriacímet 0-ba és 0xFFFF-be viszont állíthatunk:

Csupa 1-esbe állít (set):

- setm valtozo ;valtozo=0xFFFF
- setm.b valtozo ;valtozo=0xFF

0-ra állít (clear)

- clr valtozo ;valtozo=0
- clr.b valtozo

Bitműveletek

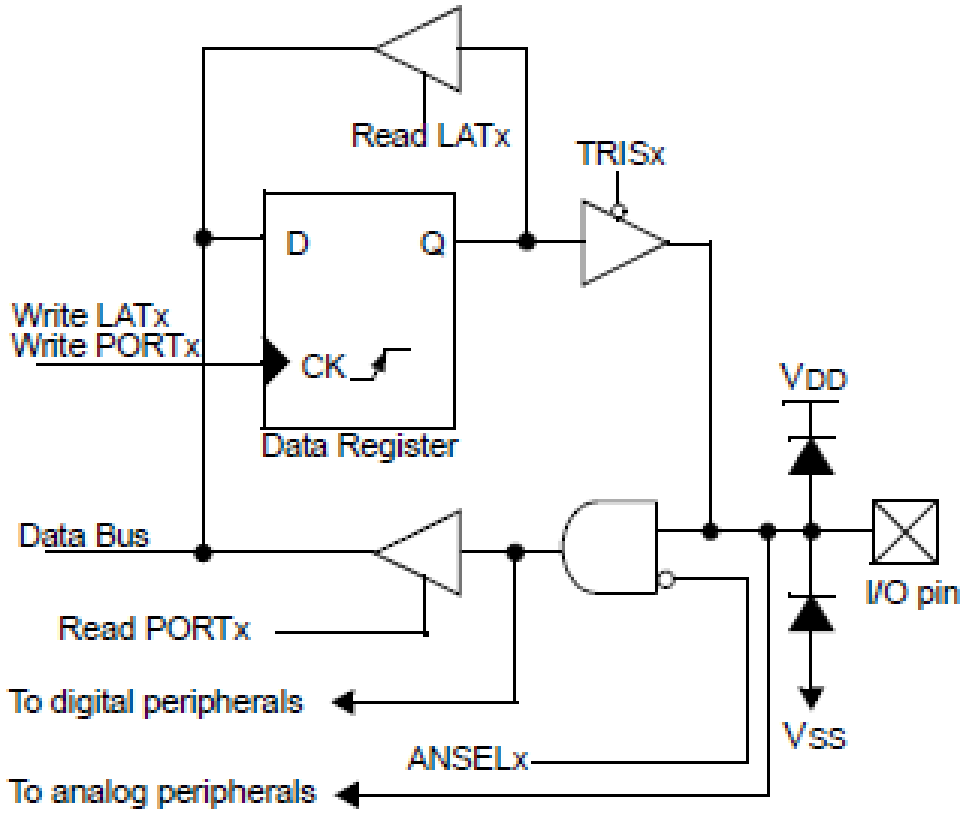
Tetszőleges regiszter vagy cím tetszőleges bitjére

- 1-be állít (set)
  - » bset valtozo, #k ;„valtozo” k. bitjét 1-be állítja (regiszter is lehet)
- 0-ba állít (clear)
  - » bclr valtozo, #k ;„valtozo” k. bitjét 0-ba állítja (regiszter is lehet)
- Negálás (toggle)
  - » btg valtozo, #k ;„valtozo” k. bitjét átfordítja (negálja)
- Ellenőrzés (test)
  - » btst valtozo, #k ;Z flag 1-essel jelzi, ha „valtozo” k. bitje 0

» #k: 0…15 konstans minden esetben

A legegyszerűbb periféria: be/kimenet port

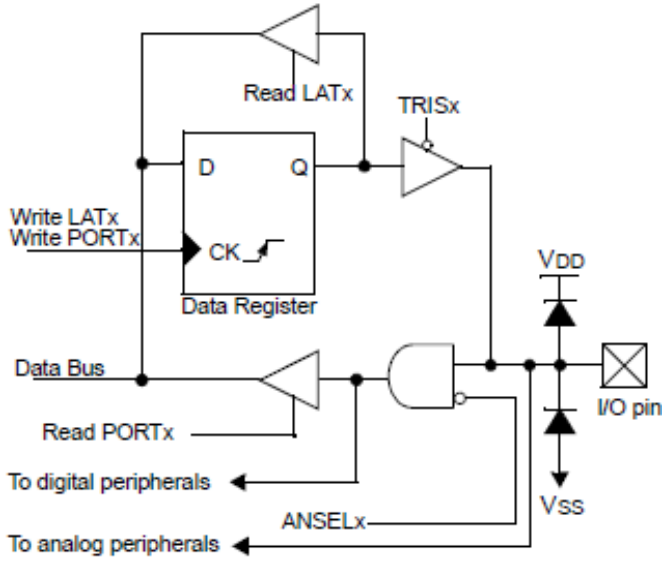
- Kapcsolódás a külvilághoz
- Több „betűvel jelölt” port (A,B,C)
- Portonként több (8-16 bit)
- Lábanként állítható a működési mód
- Lehet bemenet, kimenet, analóg üzemmód
- Regiszterek értékei vezérlik:
  - ANSELx (1: analóg be, 0: digitális be/ki)
  - TRISx: irány (0: ki, 1:be)
  - LATx: kimenet (írni szoktuk)
  - PORTx: bemenet (olvasni szoktuk)
- Reset után minden portláb bemenet és ha lehet akkor analóg mód



A legegyszerűbb periféria: be/kimenet port

- Erre nagyon jók a bitműveletek
- Pl: B port 14-es bitjét 1-esbe szeretnénk állítani, miután korábban kimenetre állítottuk:

```
bset LATB,#14          ;portB.14 legyen 1
```



- Ugyanezt lehetne máshogy is:

```
mov LATB,w0
mov #0b0100000000000000,w1
ior w0, w1, w0
mov w0,LATB
```

→ van olyan processzor, ahol ezt csak így lehet ☹

Bit eltolás

Tetszőleges regiszter tetszőleges számú bittel tolható tetszőleges irányba

- Logikai shift
  - » sl wb, k, wd ;wb-t k-val nagyobb helyiérték felé tolja (Shift Left)
  - » lsr wb, k, wd ;wb-t k-val kisebb helyiérték felé tolja (Logical Shift Right)
    - » Mindig 0 „lép be” az „új” helyekre
- Aritmetikai shift
  - » asr wb, k, wd ;wb-t k-val kisebb helyiérték felé tolja (Arithmetic Shift Right)
    - » Előjel-kiterjeszt, a legnagyobb helyiértékű bit ismétlődik!

» k: 0…15 konstans vagy egy regiszter minden esetben

Vezérlés átadás

A processzor az utasításokat sorban egymás után hajtja végre, kivéve, ha másra utasítják.
A végrehajtási sorrendtől eltérés: „ugrás”.

Kétszer két fajtája van:

- feltételes ←→ feltétel nélküli
- abszolút ←→ relatív

Feltétel nélküli ugrások

Speciális utasítás, ami előírja a következő utasítás helyét. A program mindig a kijelölt helyen folytatódik tovább.

Két fajtája van:

- abszolút: Az utasítás közvetlenül az adott címet tartalmazza, ahol folytatni kell a programot:

```
goto cím                ;ugorj a „cím” címre
```

- relatív: Az aktuális utasításhoz képest valamennyivel előbb vagy visszább kell ugrani:

```
bra #100        ;ugorj 100-zal előre
bra #-100       ;ugorj 100-zal vissza
```

Ugrási címek megadása

- Az ugrás címét megadhatjuk közvetlenül, de ritkán szoktuk:
  - » goto #0x200 ;ugorj a 0x200-s címen lévő kódra
  - » bra #-1 ;ugorj egy sorral vissza (konkrétan ugyanerre, végtelen ciklus)
- Általában címkéket használunk:
  - » Címke: adott memóriacím „beszédes” neve.
  - » Megadása assemblyben:

```
ide:    ...             ;mindig a sor elejére kell írni, „:”-tal ér véget
        goto ide        ;a fordító kicseréli az „ide” címére
oda:    …
        bra oda         ;a fordító kiszámolja a különbséget és behelyettesít
```

  - » Vegyük észre: a változók neve is csak címke
    - » A változónév egy címke az adatmemóriában

Feltételes vezérlésátadás

- Ebben a processzorban csak relatív van belőle
  - » De van olyan processzor is, amiben van mindkét féle
- Adott feltétel teljesülése esetén ugrik, egyébként nem

bra feltétel, címke

- Feltétel valamelyik flag (vagy ezek kombinációja) szerint
  - » Gyakorlatilag az utolsó aritmetikai művelet eredménye alapján

SR															
-	-	-	-	-	-	-	DC	P2	P1	P0	RA	N	OV	Z	C

Flagek

- C: aritmetikai átvitel történt
- Z: az utolsó aritmetikai művelet eredménye 0
- OV: előjeles túlcsordulás történt
- N: az utolsó aritmetikai művelet eredménye negatív

→Több érvényes kombináció is van, egyszerre több flaget is vizsgálhatunk

SR															
-	-	-	-	-	-	-	DC	P2	P1	P0	RA	N	OV	Z	C

Feltételes vezérlésátadás

Feltétel kódok:

C	Carry (not Borrow)	C
GE	Signed greater than or equal	$(\overline{N}\&\&\overline{OV}) \,||\, (N\&\&OV)$
GEU[2]	Unsigned greater than or equal	C
GT	Signed greater than	$(\overline{Z}\&\&\overline{N}\&\&\overline{OV}) \,||\, (\overline{Z}\&\&N\&\&OV)$
GTU	Unsigned greater than	$C\&\&\overline{Z}$
LE	Signed less than or equal	$Z \,||\, (\overline{N}\&\&OV) \,||\, (N\&\&\overline{OV})$
LEU	Unsigned less than or equal	$\overline{C} \,||\, Z$
LT	Signed less than	$(\overline{N}\&\&OV) \,||\, (N\&\&\overline{OV})$
LTU[3]	Unsigned less than	$\overline{C}$
N	Negative	N
NC	Not Carry (Borrow)	$\overline{C}$
NN	Not Negative	$\overline{N}$
NOV	Not Overflow	$\overline{OV}$
NZ	Not Zero	$\overline{Z}$
OV	Overflow	OV
Z	Zero	Z

SR															
-	-	-	-	-	-	-	DC	P2	P1	P0	RA	N	OV	Z	C

Elágazás

```
eredmeny=a-b
If(eredmeny==0)
{
        …/*akkor ág*/
}
else
{
        …/*különben ág*/
}
…/*következő rész*/…
```

```
        mov a, w0
        mov b, w1
        sub w0, w1, w2          ;igazából ez állít flageket
        mov w2, eredmeny
        bra NZ, nemnulla        ;ez ugrik el, ha nem nulla

        …                       ;akkor ág, átugorjuk ha nem kell
        bra vege                ;ugorjuk át a különben ágat mindenképp
nemnulla:
        …                       ;különben ág
vege:
…                               ;következő rész
```

Összehasonlítás

Speciális, az eredményt nem tároló kivonás művelet, ami csak a flagek értékét állítja be a kivonás eredményétől függően.
Két gyakori fajtája van:

- cp wb, #lit5
  → Regiszter és konstans szám összehasonlítása (a konstans sajnos max. 5 bites)
- cp wb, ws
  → két regiszter összehasonlítása (egyik lehet indirekt címzéssel is)
- cp0 f →tetszőleges változó 0-hoz hasonlítása

Syntax:	{label:}	CP{.B}	Wb,	#lit5
Operands:	Wb ∈ [W0 ... W15] lit5 ∈ [0 ... 31]			
Operation:	(Wb) – lit5			
Status Affected:	DC, N, OV, Z, C			

Syntax:	{label:}	CP{.B}	Wb,	Ws [Ws]
Operands:	Wb ∈ [W0 ... W15] Ws ∈ [W0 ... W15]			
Operation:	(Wb) – (Ws)			
Status Affected:	DC, N, OV, Z, C			

Nagyobb számok összehasonlítása

- A 16 bitnél nagyobb számokat az aritmetikához hasonlóan csak több „részletben” lehet összehasonlítani.
- Carry-t használó összehasonlító utasítás is van:

```
cpb wb,#lit5
cpb wb,ws
```

  - » „Compare with borrow”, hasonlóan a kivonáshoz a C negáltját is kivonja

Nagyobb számok összehasonlítása

```
int32_t egyik,masik;
…
If(egyik<masik)
{
        …/*akkor ág*/
}
…/*következő rész*/…
```

```
        mov egyik, w0
        mov masik, w1
        cp w0, w1               ;also helyiértékek
        mov egyik+2, w0
        mov masik+2,w1
        cpb w0,w1               ;felső helyiértékek
        bra GE, nemteljesult    ;ez ugrik el, ha nagyobb vagy egyenlő

        …                       ;akkor ág, átugorjuk ha nem kell
nemteljesult:
        …                               ;következő rész
```

A feltételek nevei (LT, GE, LE, stb.) úgy lettek kitalálva, hogy azok a cp utasításhoz igazodnak, gyakorlatilag az a relációs jel, ami a cp utasítás két operandusa közé kitehető pl:

```
cp w0, w1       ;utasítássorozat akkor ugrik el, ha w0 <= w1,
bra LE,…        ;mert LE: less than or equal, vagyis épp ahogy kimondanánk a „<=„-t
```

Speciális eset: 0-val hasonlítás

Ha több szavas adatot hasonlítunk nullával, minden egyes részét össze kell hasonlítani. Ebben a C flag nem segít. A Z flaget pedig minden egyes „cp” utasítás állítja

Ezt mindig megoldhatjuk több egymást követő összehasonítással, ami nem biztos, hogy a leghatékonyabb…

```
int32_t egyik;
…
If(egyik==0)
{
        …/*akkor ág*/
}
…/*következő rész*/…
```

```
mov egyik, w0
cp  w0, #0              ;also helyiértékek
bra  NZ, nemteljesult
mov egyik+2, w0
cp  w0, #0              ;felső helyiértékek
bra  NZ, nemteljesult   ;ez ugrik el, NZ

…                       ;akkor ág, átugorjuk ha nem kell
nemteljesult:
…                               ;következő rész
```

Speciális eset: 0-val hasonlítás

Ha több szavas adatot hasonlítunk nullával, minden egyes részét össze kell hasonlítani.

Az összehasonlítások helyett akár össze is VAGY-olhatjuk a két helyiértéket,
→Ha a vagy kapcsolatuk 0, akkor mindkettő 0 volt.

```
int32_t egyik;
…
If(egyik==0)
{
        …/*akkor ág*/
}
…/*következő rész*/…
```

```
        mov egyik, w0
        mov egyik+2, w1
        ior w0,w1,w0            ;ennek az eredménye nem is érdekes
        bra  NZ, nemteljesult   ;ez ugrik el, ha nem nulla

        …                       ;akkor ág, átugorjuk ha nem kell
nemteljesult:
        …                               ;következő rész
```

Speciális eset: 0-val hasonlítás

Ha több szavas adatot hasonlítunk nullával, minden egyes részét össze kell hasonlítani.
Ebben a C flag nem segít.
A Z flag viszont igen, mert a carry-t használó aritmetikai műveletekben a Z flag „sticky”, azaz nem lehet beállítani, ha egyszer már 0-t mutat.
→ Ha egy tetszőlegesen hosszú aritmetikai kifejezés minden része 0-ra jön ki, az utolsó művelet után a Z flag mindig helyes, az egész akárhány bites eredményre vonatkozóan…

```
int32_t egyik;
…
If(egyik==0)
{
        …/*akkor ág*/
}
…/*következő rész*/…
```

```
        mov egyik, w0
        cp  w0, #0              ;also helyiértékek
        mov egyik+2, w0
        cpb  w0,#0              ;felső helyiértékek
        bra  NZ, nemteljesult   ;ez ugrik el, NZ

        …                       ;akkor ág, átugorjuk ha nem kell
nemteljesult:
        …                               ;következő rész
```

Itt cpb nem azért kell, mert használja a C-t, hanem azért mert nem állítja 1-be Z-t ha csak a felső helyiértékeken van 0.

Számolós ciklus

```
for(i=0;i<2000;i++)
{
        …/*ciklusmag*/
}
 …/*következő rész*/…
```

```
                clr i                   ;i lesz a ciklusváltozónk
ciklusteszt:    mov #2000,w1            ;w1-be hozzunk be 2000-et
                mov i,w0
                cp w0,w1                ;i<2000
                bra GE, ciklusvege      ;ez ugrik el, ha nem kell folytatni
                …                       ;ciklusmag
                inc i                   ;ciklusváltozó növelése
                bra ciklusteszt         ;vissza az elejére
ciklusvege:
…                                       ;következő rész
```

Ez ~ 2000*(ciklusmag+7) ciklusig tart → 7*2000= 14000 ciklus gyakorlatilag felesleges…

Számolós ciklus lefele

```
for(i=2000;i>0;i--)
{
        …/*ciklusmag*/
}
 …/*következő rész*/…
```

```
                mov #2000,w0
                mov w0, i               ;i lesz a ciklusváltozónk
ciklusteszt:    cp0 i                   ;közvetlenül megnézhetjük 0-e
                bra LE, ciklusvege      ;nem kell folytatni, ha <=0
                …                       ;ciklusmag
                dec i                   ;ciklusváltozó csökkentése
                bra ciklusteszt         ;vissza az elejére
ciklusvege:
…                                       ;következő rész
```

Ez ~ 2000*(ciklusmag+5) ciklusig tart → 2*2000= 4000 ciklust nyertünk…

Lefelé számolni sokkal hatékonyabb, mert 0-val tudunk közvetlen összehasonlítani.

Számolós ciklus lefele, máshogy…

```
for(i=2000;i>0;i--)
{
        …/*ciklusmag*/
}
 …/*következő rész*/…
```

```
                    mov #2000,w0
                    mov w0, i           ;i lesz a ciklusváltozónk
cikluseleje:                            ;egyszer biztosan lefut…
                    …                   ;ciklusmag
                    dec i               ;ciklusváltozó csökkentése
                    bra NZ cikluseleje  ;vissza az elejére ha kell
ciklusvege:
                    …                   ;következő rész
```

Ha kihasználjuk, hogy egyszer biztosan le kell fusson, még összehasonlítani sem kell, a csökkentés állítja a flageket!

Ez ~ 2000*(ciklusmag+3) ciklusig tart → még 4000 ciklust nyertünk…