Digitális technika 2.

BMEVIIIAA06

4. előadás

Mikroprocesszor bevezetés

Fontosabb fogalmak

- **ALU**
- **Vezérlőegység**
- **Memória**
- **Perifériák**

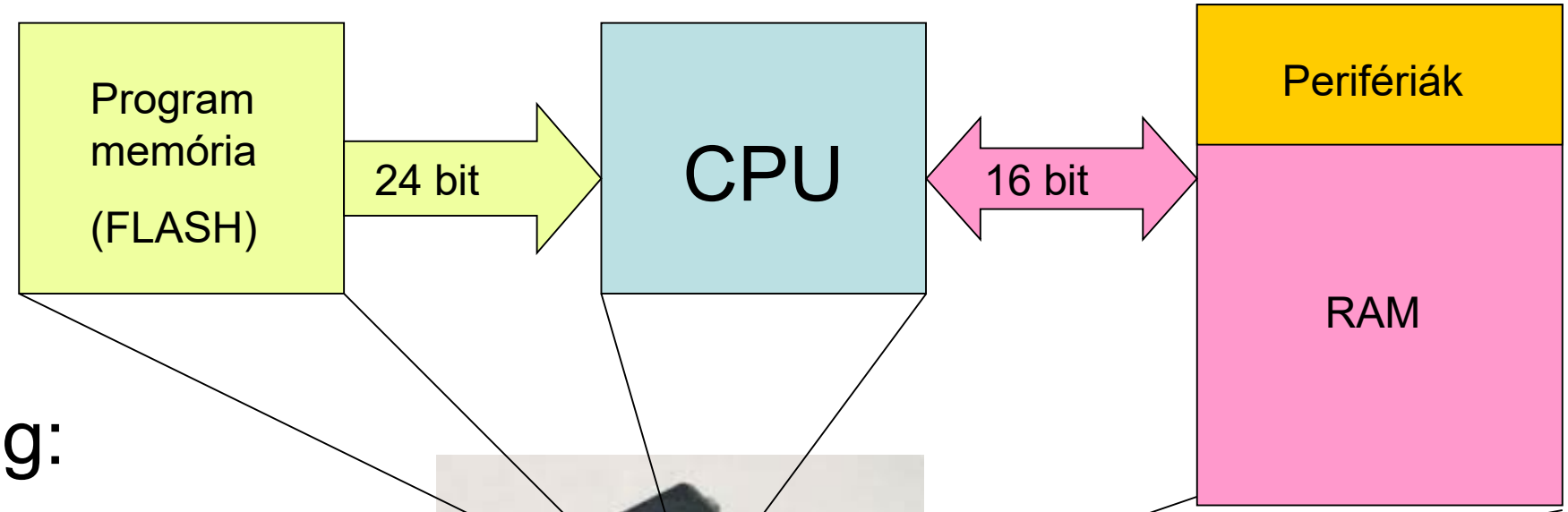
CPU

Mikrokontroller

- **Neumann architektúra (láttuk)**
  azonos memóriatartományban a kód és adat
- **Harvard architektúra**
  külön memóriatartományban a kód és adat (eltérő kezelés, lehet különböző formátum is)
  →Módosított Harvard architektúra

PIC24 mikrokontroller felépítése

- Laboron használt típus: PIC24FJ256GA705

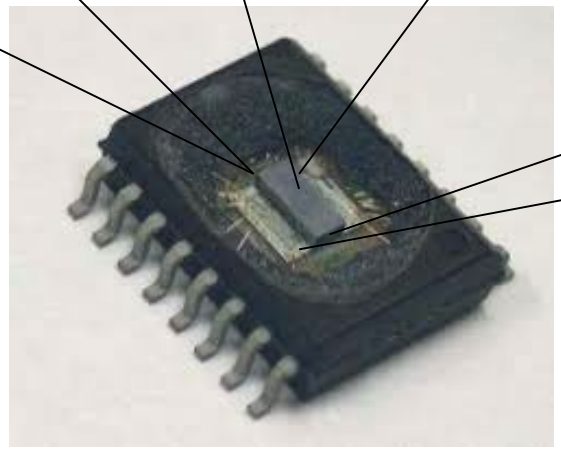


- Fizikailag:

12MB program

→ 4 millió utasítás

64kB RAM címezhető



- Gyakorlatilag:

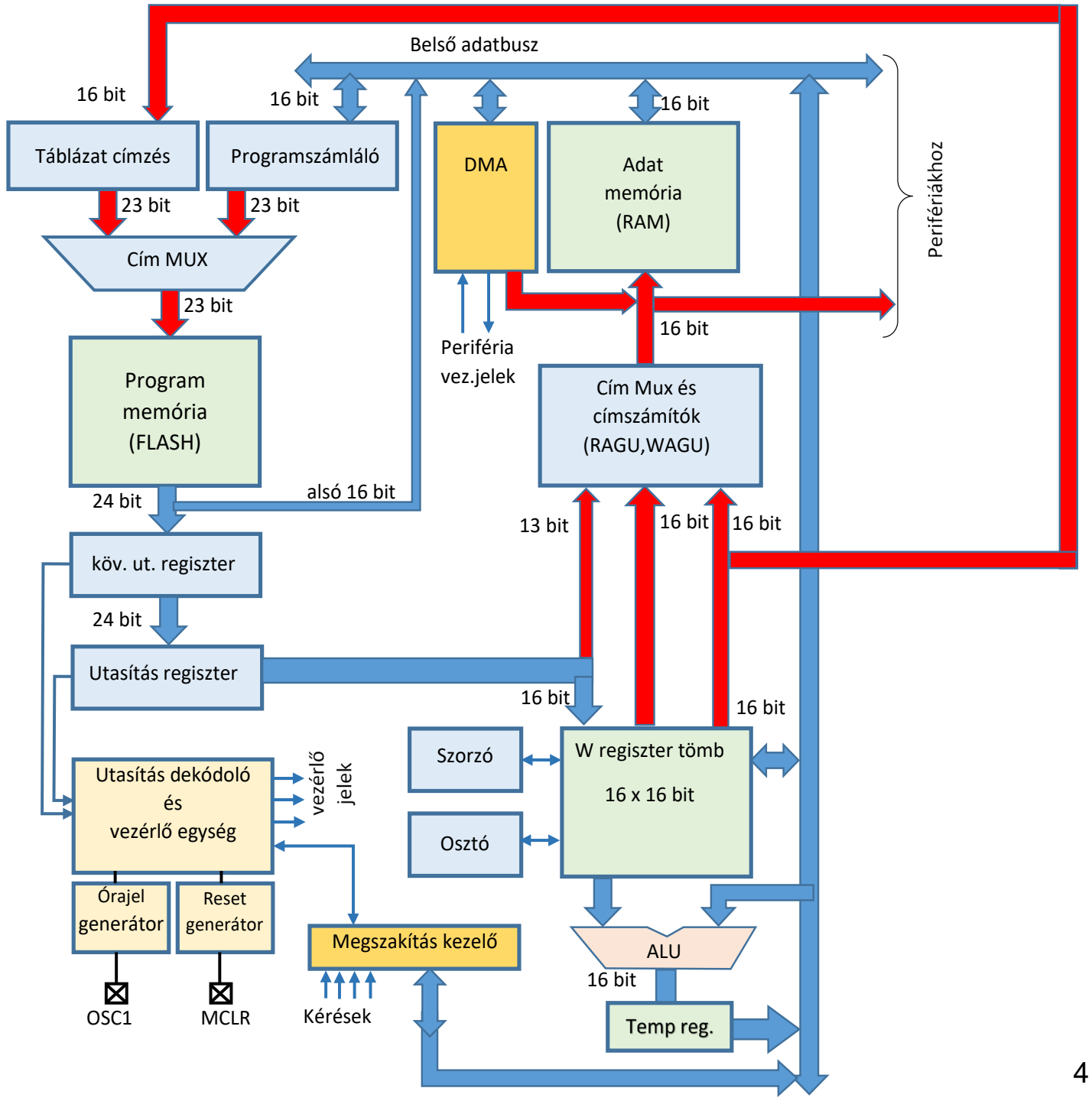
256kB program

16kB RAM van

CPU

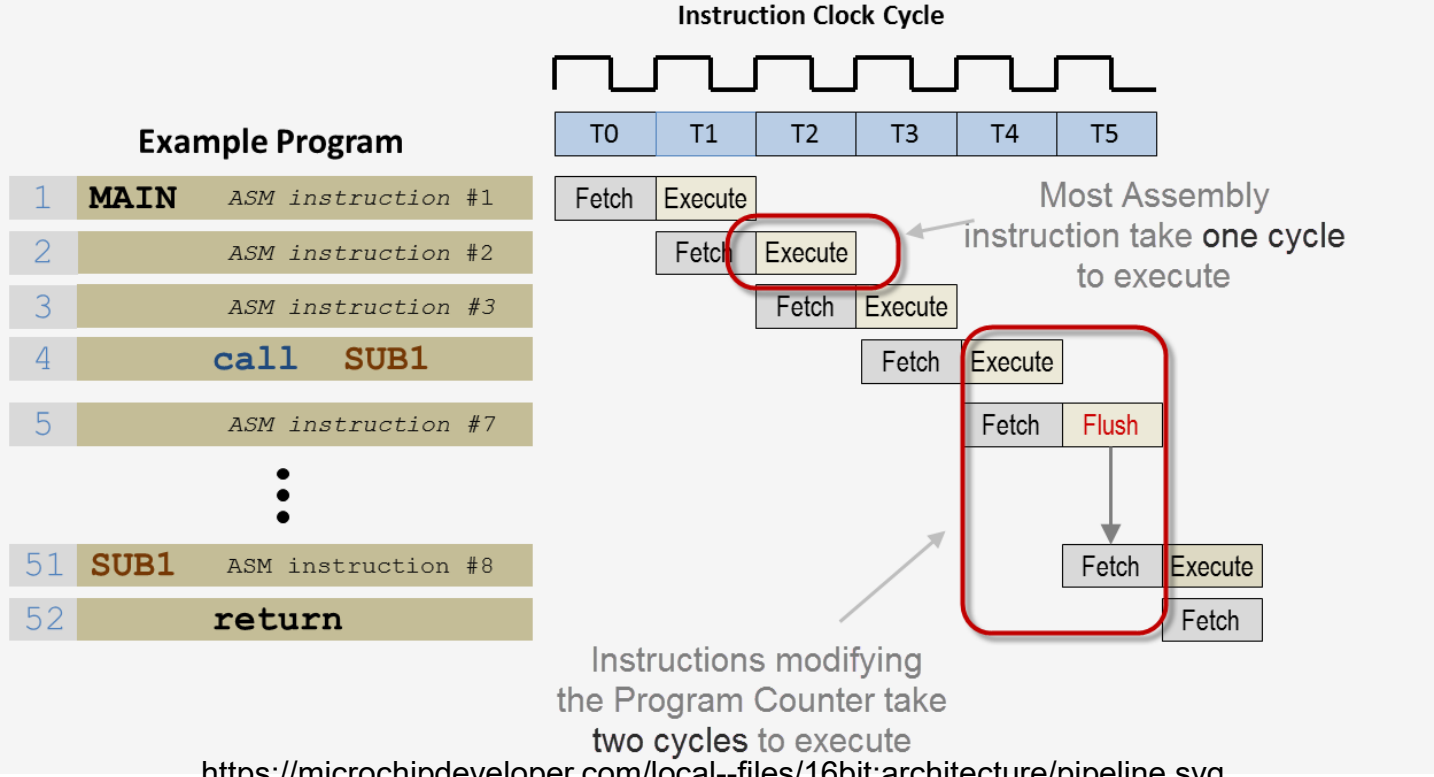
- 16 bites belső és külső adatutak
- 16db 16 bites regiszter
- 16 bites ALU

  + szorzó (16*16=32 bites)
  + osztó (32/16=16+16 bites)



Működés

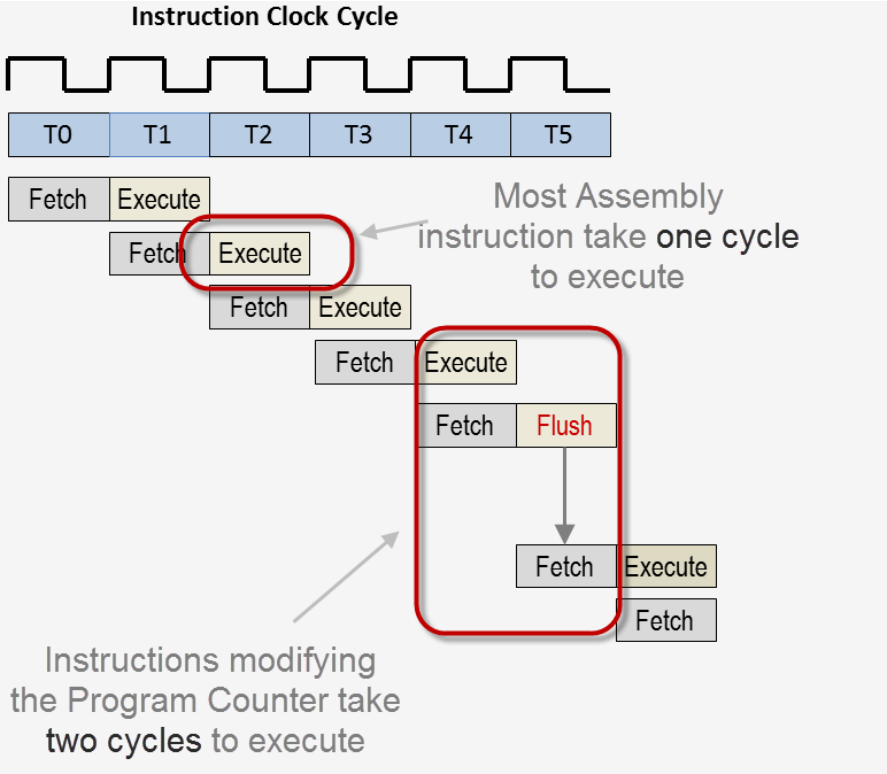
- Szinkron sorrendi hálózat
- Utasítások feldolgozása órajellel ütemezett elemi lépésekből áll
- 1 gépi ciklus (instruction cycle, Tcy) = 2 órajel



https://microchipdeveloper.com/local--files/16bit:architecture/pipeline.svg

Működés

- 1 gépi ciklus (instruction cycle, Tcy)
  = 2 órajel
- A legtöbb utasítás végrehajtása 1 ciklus
- A programot tároló flash memória elérése is pontosan 2 órajel (1 ciklus)
- A következő utasítás felolvasása (fetch) és az aktuális végrehajtása (execute) egyszerre történik
  - » Pipeline-elv (modernebb processzorokban sokkal több részművelet történik egyszerre)
- Ha valamelyik utasítás máshova ugrik, rossz utasítást olvastunk előre, ezért az 2 ciklusig tart



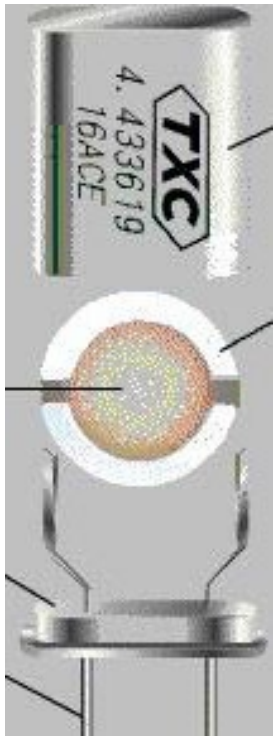
Fontos kiegészítők

- Órajel előállítás (oszcillátor)
- Alaphelyzet (Reset áramkörök)
  - Külső (MCLR)
  - Belső (POR, PWRT)
  - Tápfeszültség hibára (Brown out)
  - Programhibára (Stack over/underflow)
- Program betöltés és Debug támogatás

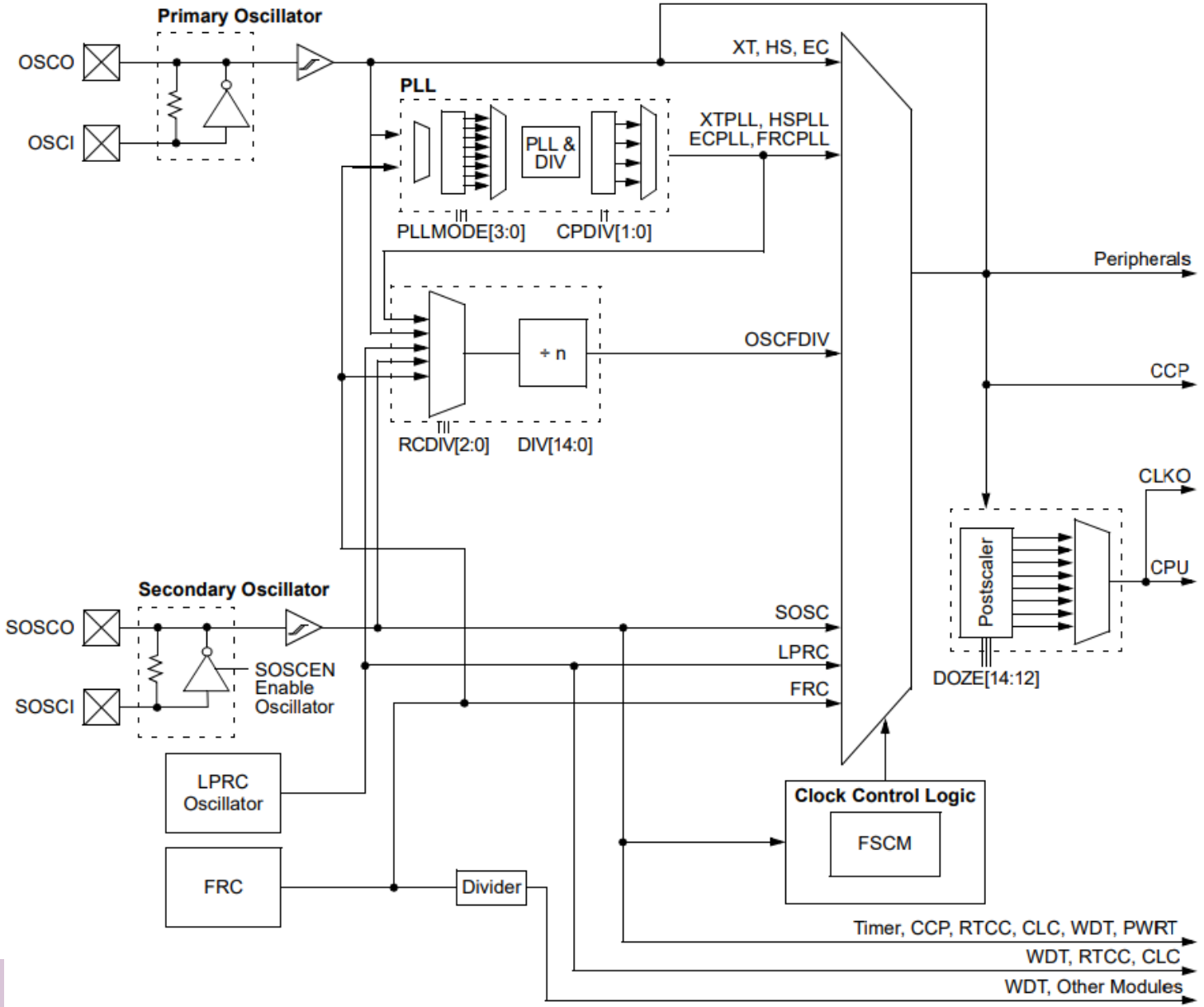
- I/O portok
- Perifériák

Oszcillátor

- Órajel előállítása adott frekvenciával
- Frekvencia meghatározó tag:
  piezo kristály, vagy RC tag
- Több választási lehetőség (konfiguráció)
- Belső szorzási lehetőség (PLL)
- Frekvencia átkapcsolási lehetőséggel
  →energiatakarékosság!

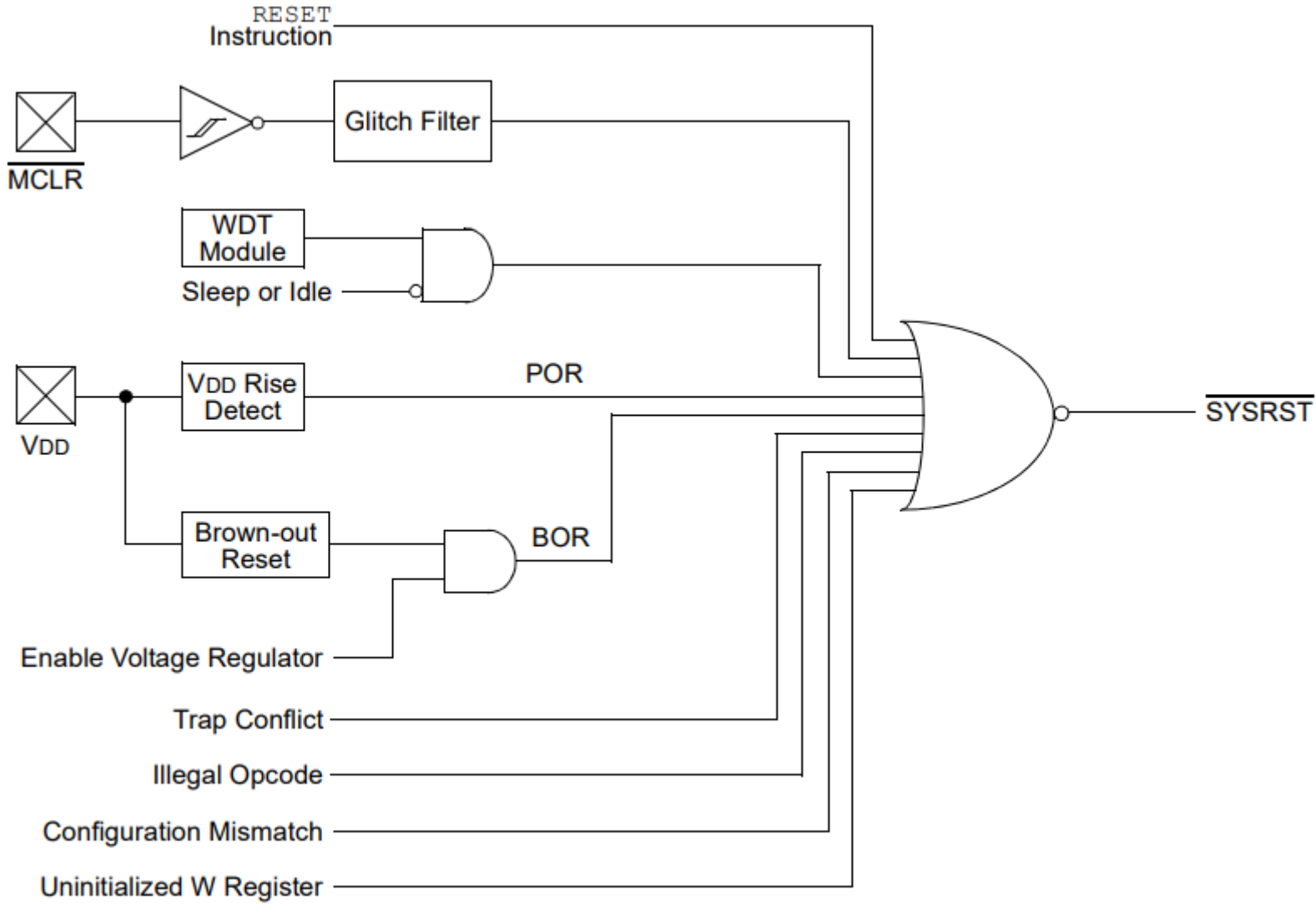


Belső órajelkezelő hálózat



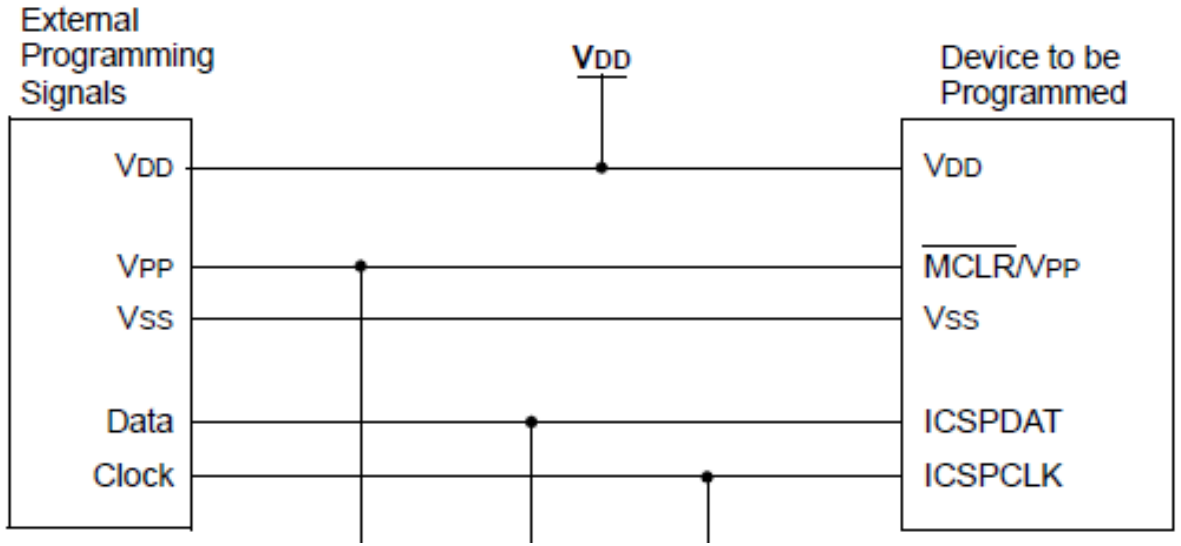
Reset

- Reset hatására:
  - Minden periféria-vezérlő regiszter az előírt alaphelyzetébe áll
  - A programszámláló 0-ra áll.
  - A memória tartalma nem változik!

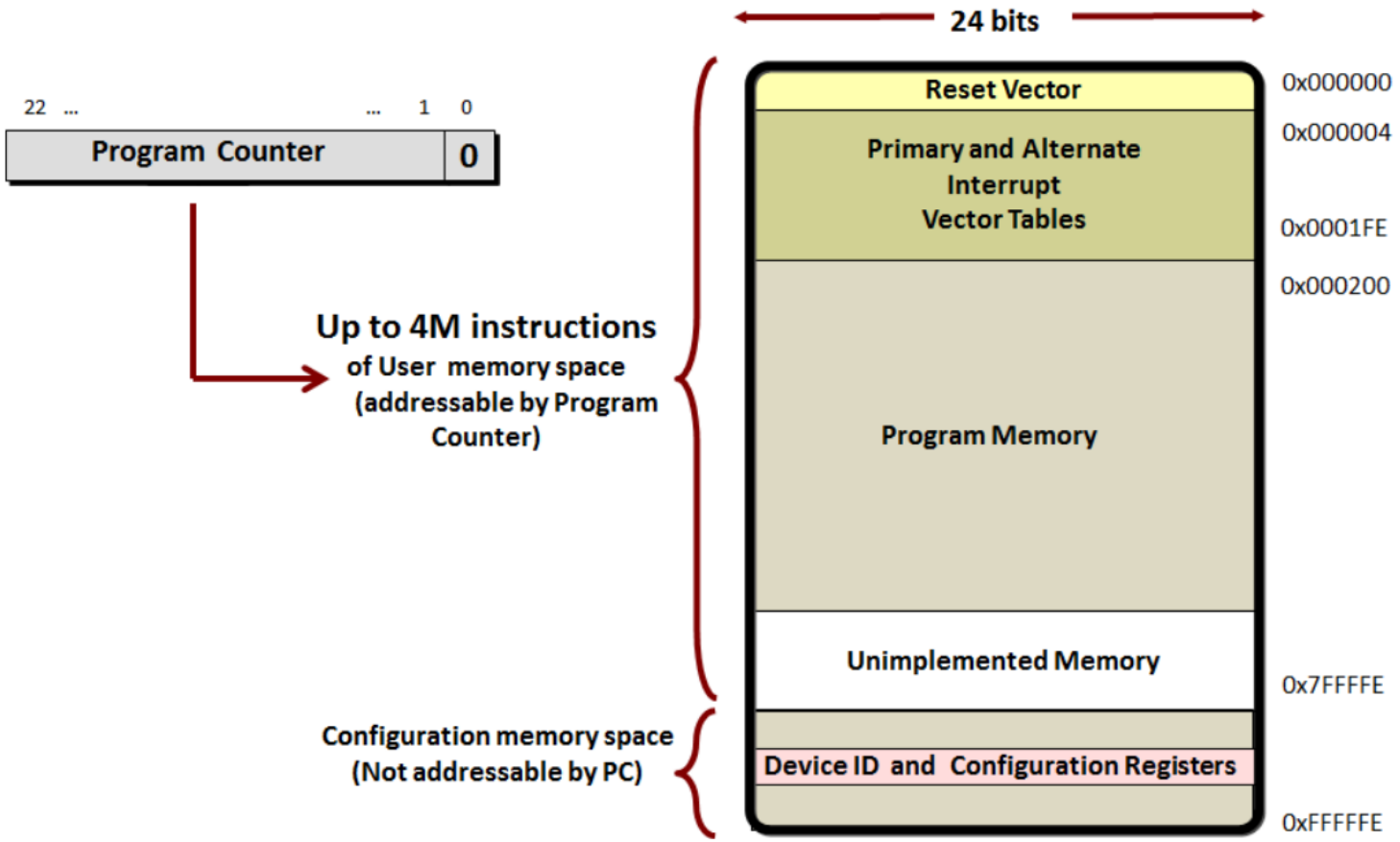


Programozás-Debug

- Áramkörben néhány IC-lábon keresztül lehet a mikrokontrollert programozni
- Ugyanezeken a lábakon lehet megfelelő eszközzel hibakeresést is végezni (debug)
  → később még lesz róla szó…



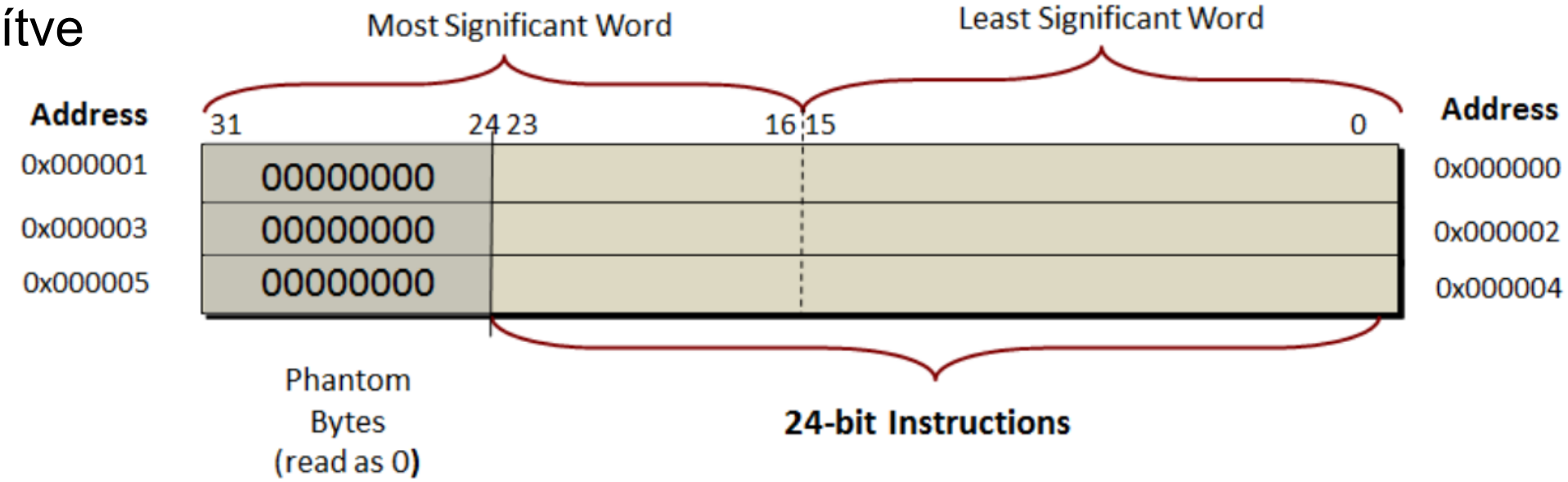
Program memória szervezés



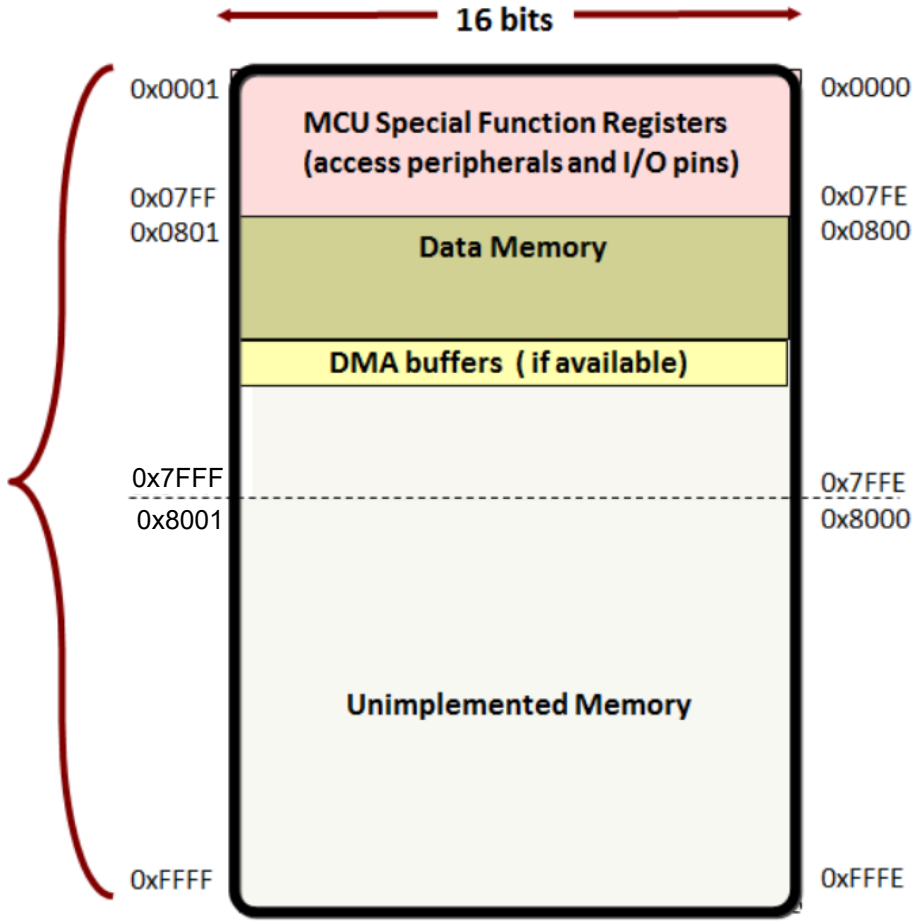
- FLASH memória
- Programkódot (utasításokat) tárol, ezek olvasása 24 bitesen történik
- Olvasása 1 ciklus idő (de ez átlapolva történik)
- Adatok is tárolhatók benne→módosított Harvard arch.
  (Az adatok benne is 16 bitesen olvashatók/írhatók)
- Konfigurációs területet is tartalmaz (később)

Program memória szervezés

- Az összes utasítás egy (vagy kettő) összetartozó 24 bites érték
  - » összesen 2db dupla szavas utasítás van, ezek speciálisan vannak kódolva
- A programszámláló (PC) legalsó bitje fixen 0
  - » azaz mindig 2-esével lép és sosem lehet páratlan
- Minden lehetséges PC értékhez érvényes utasítás tartozik
- Adat olvasásnál páratlan címről is olvashatunk, ilyenkor a felső 8 bitet kapjuk 0-kkal kiegészítve

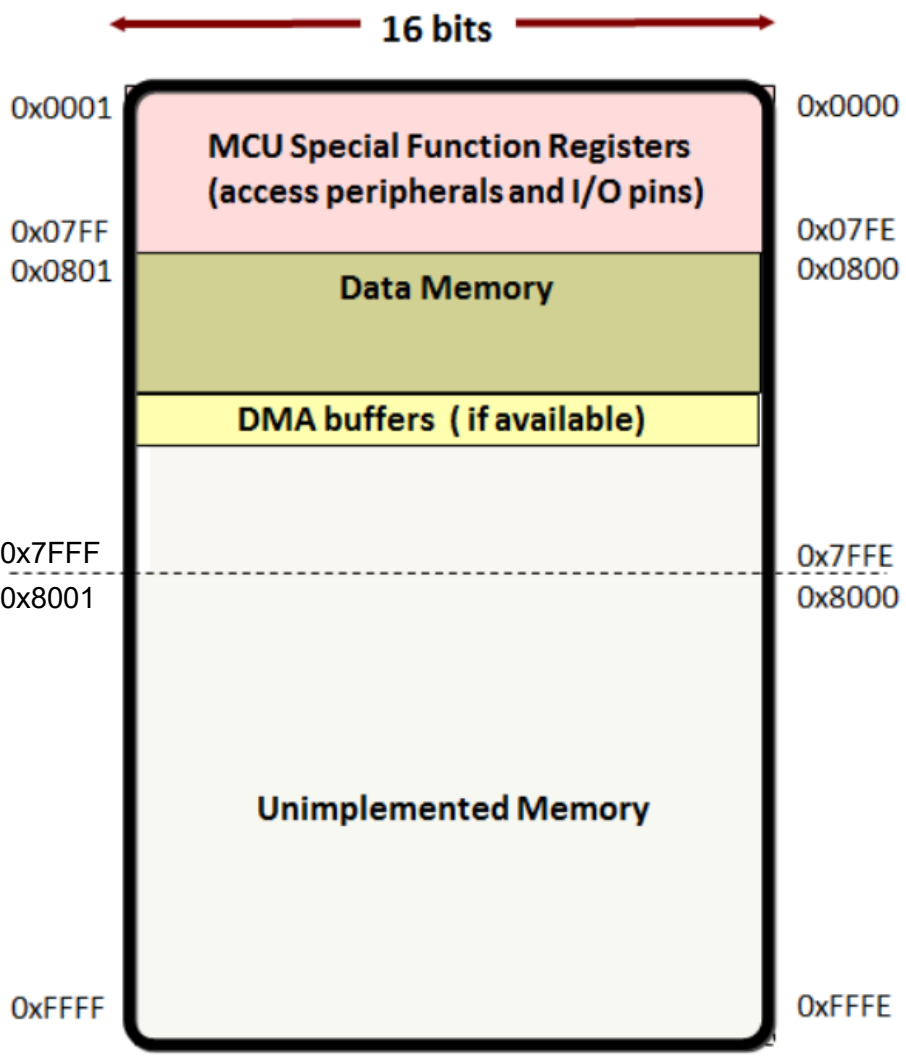


RAM/adat memóriatartomány



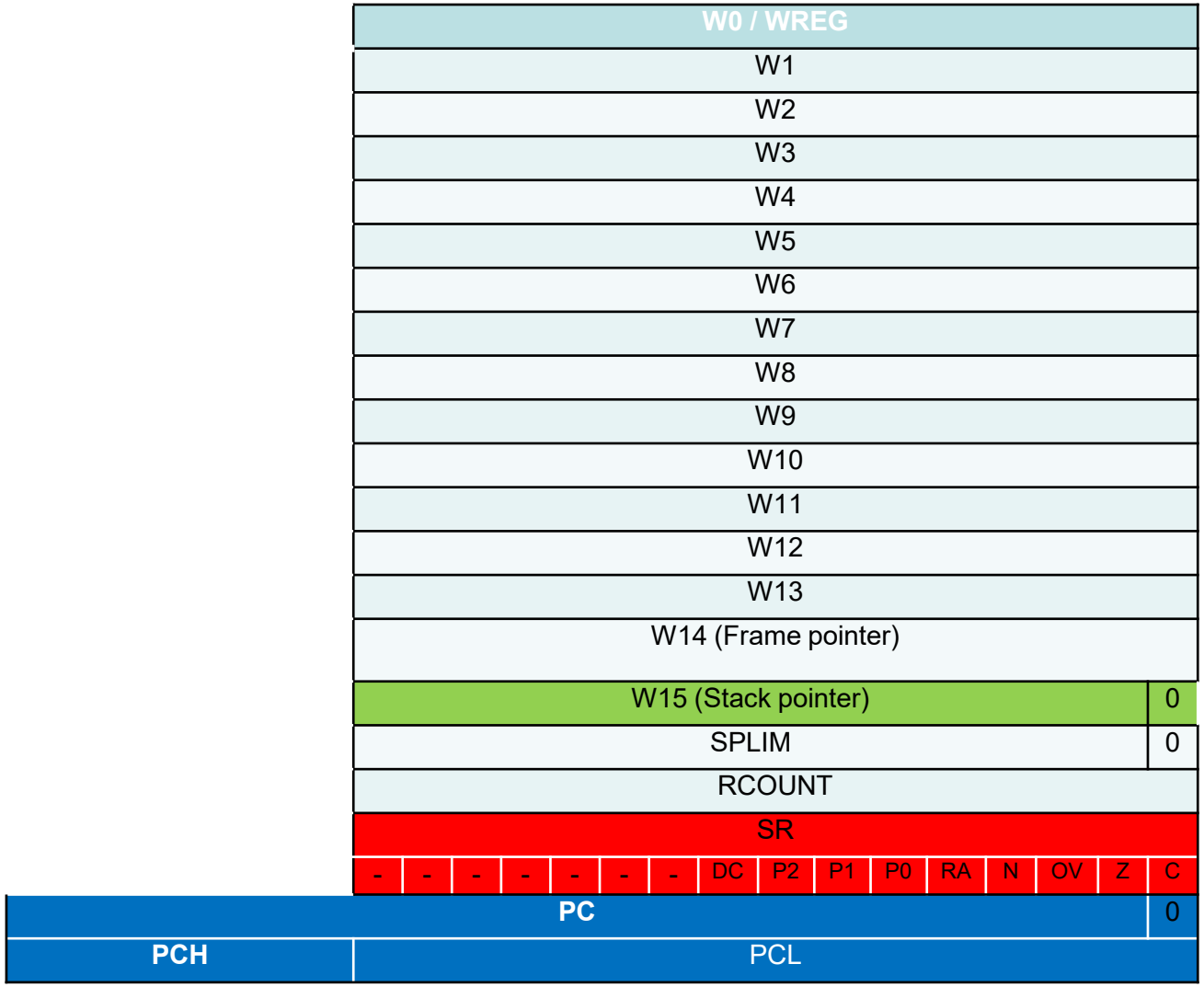
- Perifériák vezérlő regiszterei és a tényleges adatmemória itt található
- 16 bites szervezésű
- Általában a belső memória nem tölti ki a teljes lehetséges címteret
- A külső busz is ebben a címtérben van

Adat memória címzése



- 16 bites cím, 16 vagy 8 bites adat
- A 16 bites adatok úgynevezett little endian formátumban vannak (a kisebb helyiérték van a kisebb címen)
  - » 16 bites adat csak páros címen kezdődhet!
- A tényleges adatmemória a 0x800 címtől kezdődik.
- Előtte periféria vezérlő és egyéb speciális regiszterek vannak.
- Minden ciklusban egy olvasás és egy írás is történhet (ugyanarra a címre is)
- Egy része a stack terület (később)

Fontosabb regiszterek



- 15db „számozott” „munka” regiszter (W0-W15)
- W15 a stack címzésére is szolgál (W14 is részben)
  - » később
- SR (Status Register)
  - Jelzőbiteket (flag) tartalmaz az utolsó művelet eredményére vonatkozóan
- PC (programszámláló)

Gépi utasítások

- A mikrokontroller által végrehajtható műveletek
- Bináris formában vannak tárolva 24 biten, a program-memóriában

→gépi kód

- A gépi kód egy része (5-10 bit) a műveletet határozza meg, másik része tartalmazza a paramétereket (operandus)
- A programot írhatnánk gépi kódban, számokkal is *(de szörnyű lenne)*
- Az egyes bináris kódokhoz megjegyezhető neveket (mnemonik) rendelünk → assembly programozási nyelv
- Az assembler (assembly „fordító”) igazából csak a megjegyezhető neveket nehezen megjegyezhető számokra cseréli le…

Gépi utasítások

- A műveletek paraméterei lehetnek regiszterek, konstansok, vagy memóriában tárolt értékek (változók)
  - A konkrét utasítás határozza meg, hogy hány és milyen típusú paraméterei lehetnek
- Az egyes utasításoknak 0, 1, 2 vagy 3 paramétere (operandusa is lehet)
- Utasítás csoportok:
  - Adatmozgató utasítások
  - Aritmetikai műveletek
  - Logikai műveletek
  - Bitműveletek
  - Vezérlés átadás
  - Vezérlő műveletek

Gépi utasítások - mozgatás

- Adatmozgatás
  - » Igazából másolás, de így hívják
- Két operandusa van: honnan, hova

mov(.b) *honnan*, *hova*

- Mozgathat csak 8 bitet egyszerre (.b végződéssel) vagy 16 bitet egyszerre (.b végződés nélkül)
- A két operandusa többféle formában megadható
  - » Ezeket hívják címzési módoknak
  - » Elég sok címzési mód van, csak keveset fogunk használni…

Mozgatás – regiszterek között

- Egyik (számmal rendelkező) munkaregiszterből a másikba mozgat (másol)
- Pl.: mov W3, W1 ;tedd a W3 tartalmát a W1-be

  Az eredeti tartalom W3-ban is ott marad, de W1-be is ugyanaz kerül

- mov.b W3,W1

  Csak az alsó 8 bitet másolja át, W1 felső 8 bitjét változatlanul meghagyja

Mozgatás – regiszterek között

- Gépi kódban:

  mov W11,W7

  01111 0000 0 000 0111 000 1011

Általában minden bit jelent valamit, részletesen a reference manual-ban megtalálható *(érteni kell, de egyesével tudni nyilván nem kell, ahogy gépi kódot olvasni sem…)*

mov.b W11,W7
01111 0000 1 000 0111 000 1011

Mozgatás – közvetlen cím

- Az egyik munkaregiszterből memóriába vagy memóriából munkaregiszterbe másol
- Pl.: mov 0x1000, W1
  - » Az 1000h címen lévő számot teszi a W1-be (0x előtag, mint a C-ben, 16-os számrendszer)
- Hasonlóan:

  mov W2, 0x800
  - » A W2 tartalmát teszi a 800h címre a memóriába

Mozgatás – közvetlen cím

- Ugyanez gépi kódban:

  mov 0x1000, W1

  10000 0001000000000000 0001

  mov W2, 0x800

  10001 0000100000000000 0010

- Vegyük észre:
  - Ugyanaz a mnemonik különböző gépi kód lehet (az operandusok alapján egyértelműen)
  - Így is csak 15 bit címnek van hely a gépi kódban (ez nem baj, mert a 16 bites adat úgyis páros címen kezdődik)
  - Egynél több memóriacím 24 biten nem fér el, ezért a ~~mov 1000, 2000~~ érvénytelen

Mozgatás – közvetlen adat

- Regiszterek és memória közt mozgatni jó, de valahogy először számoknak kellene bekerülni a regiszterekbe.
- Pl.: mov #23, W1 ;tedd a 23-as számot a W1 regiszterbe
  - » Közvetlenül a kódban lévő számot mozgatja regiszterbe
- A # jel jelenti azt, hogy közvetlenül a szám értékről van szó és nem címet jelent!
  - » A # jelek lehagyása számok elől a leggyakoribb hibaok ☺
- Bármilyen 16 bites szám mozgatható bármelyik munkaregiszterbe
- Ennek fordítva nincs értelme…
  - » ~~mov W1,#10~~ érvénytelen.

Mozgatás – indirekt címzés

- Összetett adatszerkezetek (tömb, struktúra) használatánál szükség van pointerekre.
  - » A megcímzett memóriarekesz helyét is egy memóriarekeszben (vagy regiszterben) találjuk
- Leggyakoribb formája a regiszter indirekt címzés:
- Pl: mov W11,[W1]
  - » A W11-ben lévő számot másolja oda, ahová a W1 mutat
- A []-jelek között lévő regiszter indirekciót jelöl.
- Forrás és cél is lehet indirekt
  - » és ezen kívül is van még más indirekt címzési mód is, ezekről később…
- Bármely munkaregiszter használható indirekt módon

Mozgatás – indirekt címzés

- Vegyük észre, hogy az alábbi két kód ugyanazt végzi el:

```
mov #1000, W2
mov W6, [W2]
```

```
mov w6, 1000
```

- A bal oldali látszólag hosszabb, viszont W2 tartalmát előtte akár ki is számolhattuk volna, nem csak beírtuk.
- A jobb oldalon a fix címet a program már lefordításkor tartalmazza.

Aritmetika

- Az aritmetikai utasításoknak létezik 3 vagy 2 operandusos formája is.
  - » *(1 operandusos is van, ilyenkor W0 az egyik operandus, nem fogjuk használni…)*
- 3 operandusos: A=B+C, 2 operandusos: A=A+B
  - » Leggyakrabban a három operandusos formát használjuk
- Legegyszerűbb aritmetikai utasítás: összeadás
- Pl. **add(.b) B, C, A** ; A=B+C műveletet számítja ki
- A 3 operandusos műveletek operandusai regiszterek mindig lehetnek *(de néha más is)*
- *A 3-ból az utolsó kettő operandus indirekt címzési módokat is használhat*

Aritmetika

- Példa1: adjunk hozzá az 0x1000 címen lévő 16 bites számhoz 100-at, az eredményt tegyük az 0x1002-es címre:

mov 0x1000, W0 →először behozzuk ami az 1000h-n van
mov #100, W1 →behozzuk a 100-as számot (#!)
add W0, W1, W2 →elvégezzük a műveletet
mov W2, 0x1002 →visszaírjuk az eredményt a helyére

Aritmetika

- Példa2: adjunk hozzá az 0x1000 címen lévő 16 bites számhoz 100-at, az eredményt tegyük vissza az eredeti helyre: (formálisan ugyanaz csinálhatjuk)

mov 0x1000, W0 →először behozzuk ami az 1000h-n van

mov #100, W1 →behozzuk a 100-as számot (#!)

add W0, W1, W2 →elvégezzük a műveletet

mov W2, 0x1000 →visszaírjuk az eredményt a helyére

Aritmetika

- Példa2: adjunk hozzá az 0x1000 címen lévő 16 bites számhoz 100-at, az eredményt tegyük vissza az eredeti helyre:
- *Eggyel hatékonyabb megoldás (mert tanultunk indirekt címzést):*

mov #0x1000, W0 →az 1000 címet töltjük W0-ba (#!)

mov #100, W1 →behozzuk a 100-as számot (#!)

add W1, [W0], [W0] →indirekt címzést használunk

» Nyertünk egy utasításnyi időt…

Aritmetika

- Példa2: adjunk hozzá az 0x1000 címen lévő 16 bites számhoz 100-at, az eredményt tegyük vissza az eredeti helyre:
- *Még hatékonyabb megoldás, ha olvastuk az utasításkészletet:*

mov #100, W0 →a 100 számot töltjük W0-ba (#!)

add 0x1000 →egy operandusos add, közvetlen címzés

» Tanulság: Assemblyben rengeteg jó megoldás van, rengeteg „trükkös” lehetőséggel, a legegyszerűbbet megtalálni *nagyon* nehéz.

A feladatmegoldásoknál mindig egy működő és sosem a legegyszerűbb megoldást kérjük.