

PIC programozása (PIC16F887)

1. Alapismeretek
Számrendszerek, számok ábrázolása
2. Mikrovezérlő, PIC
3. Programozás elmélet
4. C nyelv, alapismeretek
változók, elágazás (if), ciklusok (while)
5. PIC, digitális kimenetek vezérlése
6. C nyelv, tömbök, bitműveletek, ciklusok (for)

1.1. Egy kis számolgatás

- 10-es számrendszer (decimális)

10 db számjegy \rightarrow '0' '1' '2' '3' ... '8' '9'


helyi értékek \rightarrow ... 10000 1000 100 10 1

tehát pl. a 7439 azt jelenti hogy van 7db 1000-esünk meg 4 db 100-asunk
meg 3db 10-esünk meg 9db 1-esünk

- 2-es számrendszer (bináris)

csak 2 számjegy \rightarrow '0' és '1' helyi értékek \rightarrow ... 32 16 8 4 2 1

pl. $1011_2 = 1*8 + 0*4 + 1*2 + 1*1 = 11$


8 4 2 1

$1101010_2 = \dots ? \rightarrow 64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1$ melyikből mennyi van?

$348 = \dots_2 ? \rightarrow 256 \ 128 \ 64 \ 32 \ 16 \ 8 \ 4 \ 2 \ 1$ melyikből mennyi kell?

1.2. Egy kis számolgtás

- 1db bináris számjegy (helyiérték) → bit

8 bit = byte 1024 bit = 1kilobit

- 10-es → 2-es átalakítás algoritmus

sorozatos osztás 2-vel, és a maradékok adják a számjegyeket

pl. $25 = \dots_2 ?$ $25 \rightarrow 12 \rightarrow 6 \rightarrow 3 \rightarrow 1 \rightarrow 0$

| | | | | | |
|--|--------------|--------|--------|--------|---------|
| | ↓ 1 | ↓ 0 | ↓ 0 | ↓ 1 | ↓ 1 |
| | 1 helyiérték | 2 h.é. | 4 h.é. | 8 h.é. | 16 h.é. |

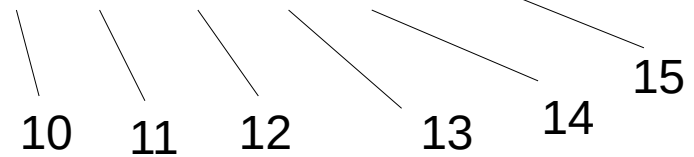
eredmény: 11001_2

1.3. Egy kis számolgtatás

- 16-os számrendszer (hexadecimális)

16 számjegy → '0' '1' '2' '8' '9' 'A' 'B' 'C' 'D' 'E' 'F'

helyi értékek → 256 16 1



pl. $3A4_{16} = 3 \cdot 256 + 10 \cdot 16 + 4 \cdot 1 = 932$

$1EC_{16} = \dots ? \rightarrow 256 \ 16 \ 1$ melyikből mennyi van?

hexa → bináris konverzió:
számjegyenként 4 bitre !

pl. $2E_{16} \rightarrow 00101110_2$

0010 1110

bináris → hexa konverzió:

4 bites csoportokra osztás jobbról,
csoportonként hexa számjegyekké alakítás!

pl. $1111010111_2 \rightarrow 3D7_{16}$

3 13 7
D

- Miért használjuk a 16-os számrendszert ?

Mert nagy számokat egyszerűbb leírni így (kevesebb számjegy)
mint kettes számrendszerben

1.4. Negatív számok

- Negatív számok

az előjel ábrázolására/tárolására → plusz egy előjel bit (a legelső)

előjel bit: 0 → pozitív szám 1 → negatív szám

de ez még nem elég, a műveletvégzés így még okozhat hibákat !

pl. +2 és -2 összeadása → 0010+1010=1100 → -4 !!!

előjel

- A negatív számokat 2-es komplementes kódban ábrázoljuk

2-es komplementes kód: a megfelelő pozitív szám bitenkénti negáltja, majd utána a számhoz hozzáadunk még 1-et

pl. 4 bites számok (ebből egy előjel)

| | | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-----------|-----------|----------|
| 0 → 0000 | 1 → 0001 | 2 → 0010 | 3 → 0011 | 4 → 0100 | 5 → 0101 | 6 → 0110 | 7 → 0111 |
| -1 → 1111 | -2 → 1110 | -3 → 1101 | -4 → 1100 | -5 → 1011 | -6 → 1010 | -7 → 1001 | |
| -8 → 1000 | | | | | | | |

1.5. Negatív számok

- Másik oldalról a negatív számok használata felére csökkenti a használható számtartományt !

pl. ha 8 bites számokkal dolgozunk akkor két eset lehetséges

Csak pozitív számokat használunk!

tehát: 0 → 00000000 1 → 00000001

2 → 00000010 3 → 00000011

4 → 00000100

253 → 11111101 254 → 11111110

255 → 11111111

Pozitív és negatív számokat használunk!

tehát: 0 → 00000000 1 → 00000001

2 → 00000010

126 → 01111110 127 → 01111111

-1 → 11111111 -2 → 11111110

-3 → 11111101

-127 → 10000001

-128 → 10000000

1.6. Törtek ábrázolása

- Fixpontos számábrázolás

a bináris pont helye rögzített → 'n' bites szám, ebből 'k' bit a törtrész, 'n-k' bit az egész rész + előjel



pl. -12,75 → 32 biten kódolva (20 bites egész rész, 11 bit tört rész)

| | | | | |
|---|---|---|--------------------------|---------------|
| +12,75 → 1100,11 | → | 0 | 00000000000000001100 | 110000000000 |
| 1-es komplement kód (bitenként negálás) → | | 1 | 1111 1111 1111 1111 0011 | 0011 1111 111 |
| 2-es komplement kód (1-es kompl. , +1) → | | 1 | 1111 1111 1111 1111 0011 | 0100 0000000 |

Hátránya:

- pazarló, sok bit kell a nagy pontossághoz
- nagyon nagy és nagyon kicsi számok ábrázolása problémás

1.7. Törtek ábrázolása

- Lebegőpontos számábrázolás

- a bináris pont helye nem fix
- a bináris szám felírása a következő alakban → $N = \pm M * 2^{\pm E}$
- csak M és E értékét kell tárolni!
M – normalizált mantissza, E – karakterisztika
- előjel + abszolút érték ábrázolása

Normalizálás:

- törtre → mantissza $\frac{1}{2}$ és 1 közé essen →
törtrész első bitje mindig 1-es, nem tároljuk
- egészre → mantissza 1 és 2 közé essen →
egész rész utolsó bitje mindig 1-es, nem tároljuk

Karakterisztika ábrázolása:

- eltolt (offset) karakterisztika → $c = E + d$ (karakterisztika + eltolás)
egész szám hozzáadása hogy pozitív legyen
- eltolás mértéke (d), ha a karakterisztika 'k' bites $d = 2^{k-1}$
 $d = 2^{k-1} - 1$

1.8. Törtek ábrázolása

- Lebegőpontos számábrázolás

ANSI / IEEE szabvány

- szimpla pontosságú lebegőpontos szám, 32 bites
karakterisztika 8 bites, mantissza 23 bites + előjel bit
- dupla pontosságú lebegőpontos szám, 64 bites
karakterisztika 10 bites, mantissza 53 bites + előjel bit

$$c = E + d$$

$$d = 2^{k-1} - 1$$

Szimpla pontosságú:

$$k=8 \rightarrow d=127$$



$$\text{pl. } N = -12,75 \rightarrow N = -1100,11 \rightarrow N = -1,10011 \cdot 2^3$$

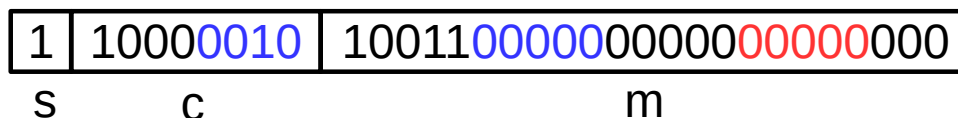
$$s=1$$

$$E=3$$

$$c=3+127$$

$$m=10011000000000000000000$$

$$c=10000010$$

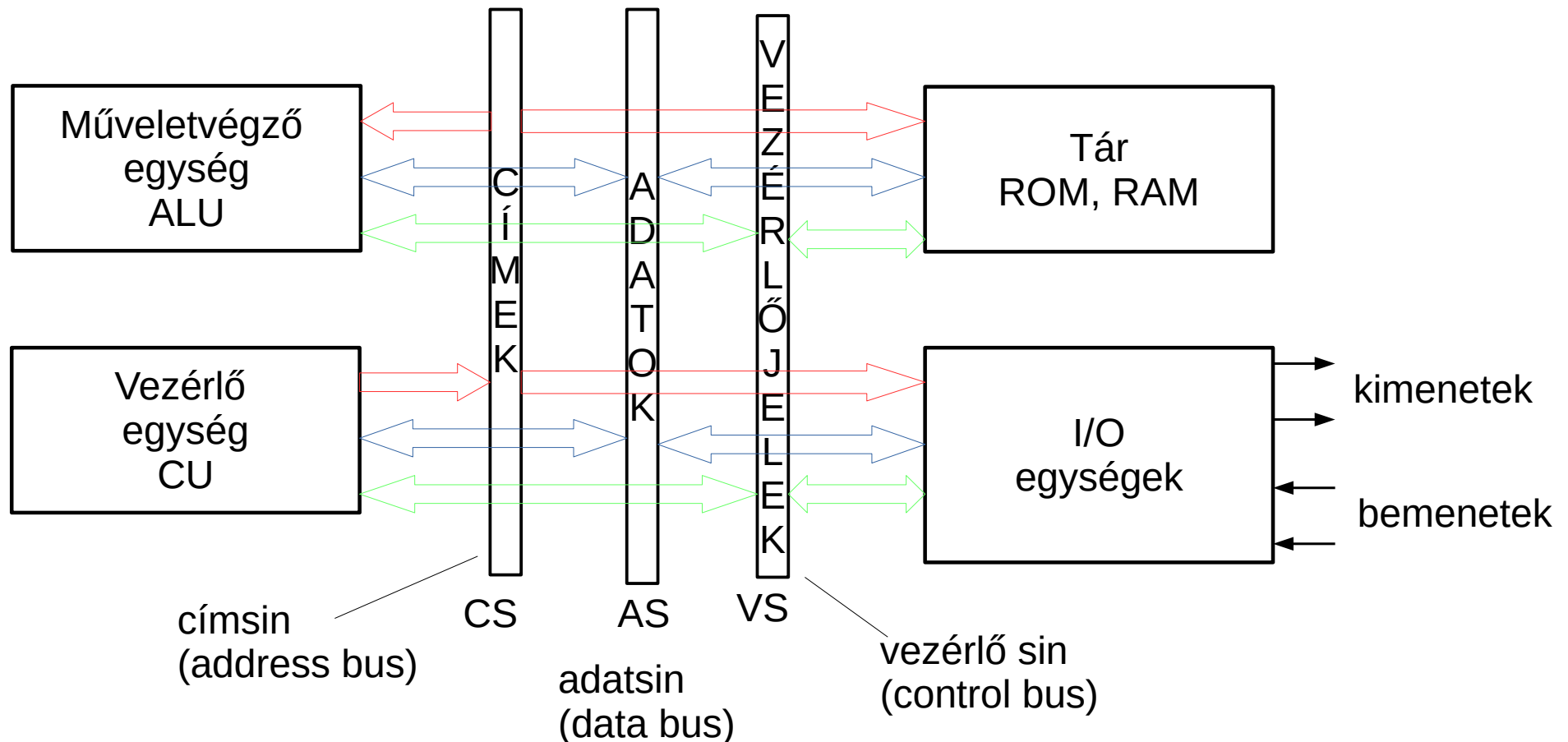


2.1. Mikroszámítógép

Mikroszámítógépek felépítése:

Az egységek közötti kapcsolatot, az adatok áramlását a sínrendszer (vezetékek) biztosítja.

Funkció alapján a vezetékek 3 csoportba oszthatók → 3 sín (bus)



2.2. Mikroprocesszor

- Microprocessor $\rightarrow \mu P$
- egy integrált áramkörben (IC-ben) megvalósított CPU egység, a számítógép központi egysége
- Tehát tartalmaz: vezérlőegységet és műveletvégző egységet
$$CPU = CU + ALU \rightarrow \mu P$$
- Egy mikroprocesszorhoz csak memóriákat és periféria illesztőket, perifériákat kell hozzá csatlakoztatni, hogy egy komplett számítógépet kapjunk
- Általában nagyon sok kivezetése (lába) van \rightarrow a három sín vezetékei miatt

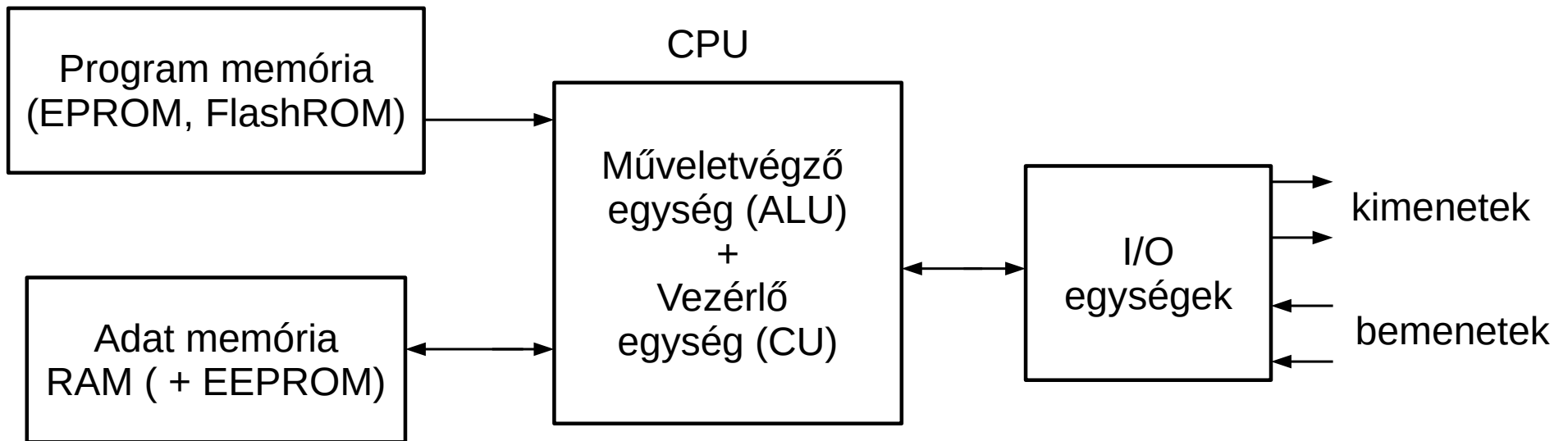
2.3. Mikrovezérlő

- Microcontroller → μC
- komplett kis számítógép egy integrált áramkörben (IC-ben)
- Tehát tartalmaz: mikroprocesszort, többféle memóriát, és különféle perifériákat, kiegészítő áramköröket
- $\mu\text{C} = \text{CU} + \text{ALU} + \text{memory} + \text{I/O}$
- Több gyártó cég is van: Atmel, Microchip, Texas, Intel, Analog Devices,
- Talán a két legelterjedtebb mikrovezérlő család, az AVR-ek (Atmel) és a PIC-ek (Microchip)
- Mivel számítógépekről van szó ---> programozni kell őket az adott feladat elvégzésére

2.4. Mikrovezérlő

Mikrovezérlők felépítése

Nem teljesen a hagyományos számítógép felépítést követik →
az adat és program memória külön van választva → Harvard architektúra



Jellemző perifériák: digitális bemenetek, digitális kimenetek, analóg bemenetek, időzítők, számlálók, komparátorok, kommunikációs portok (RS232, SPI, I²C, USB)

2.5. PIC

- Microchip cég gyártja ezen mikrovezérlőket
- PIC rövidítés,----> Programmable Interface Controller
 \swarrow eredetileg: Peripheral Interface Controller ?

PIC-ek csoportosítása

- * utasításhossz alapján lehet: 12,14,16,24 vagy 32 bites
- * adathossz alapján lehet: 8,16 vagy 32 bites

| | 12bit | 14bit | 16bit | 24bit | 32bit |
|-------|----------------|----------------|-------|----------------|-------|
| 8bit | PIC10 PIC12 | PIC14 PIC16 | PIC18 | | |
| 16bit | | | | PIC24 dsPIC | |
| 32bit | | | | | PIC32 |

pl.
PIC16F887,
PIC18F2550

2.6. PIC16F887 jellemzői

- Többféle tokozással készül, DIP → 40 kivezetés
- Tápfeszültség: 2 – 5,5V között (V_{DD} láb(11,32) → + V_{SS} láb(12,31) → -)
- Órajel: - 0 – 20 MHz között, külső oszcillátor OSC1(13), OSC2(14) lábakra
- de tartalmaz belső RC oszcillátort (31kHz - 8 MHz)
- Utasítás végrehajtás
 - RISC processzor, (csökkentett utasításkészletű) csak 35 utasítás
 - egy gépi ciklusa (belső ciklus) 4 órajel ciklus alatt játszódik le
 - az utasítások nagy része 1 gépi ciklus alatt végrehajtódik
 - az utasítások 14 bitesek, és 8 bites adatokkal dolgozik
- Memóriák:
 - program memória, FlashROM 8kszó (8k x 14 bit)
 - adatmemória, RAM 368 byte + regiszterek (hardver, perifériák vezérlésére)
EEPROM 256 byte, 'háttértár' → adatok stabil tárolására
- Üzem módjai:
 - normál program végrehajtás → \overline{MCLR}/VPP (1) lábra tápfeszültség
 - RESET (újra indítás) → \overline{MCLR}/VPP (1) lábra 0 szint
 - programozás (ICSPDAT-40, ICSPCLK-39 lábakon) → \overline{MCLR}/VPP (1) lábra 12V

2.7. PIC16F887 jellemzői

A legtöbb kivezetésnek/lábnak több funkciója is van

a kívánt funkciót programozással, a megfelelő regiszterek (SFR) bitjeinek állításával lehet kiválasztani

Perifériák

- 35 (36) digitális bemenet/kimenet, (programból kell állítani, hogy be vagy ki)
- 14 analóg bemenet (AN0, AN1, AN2, ... AN13)
- 3 időzítő/számláló (timer)
- kommunikációs portok, USART (RS232,RS485), MSSP (SPI, I²C) ...

A digitális bemenetek/kimenetek szervezése

8-as csoportokba vannak szervezve, és így vezérlő regiszterekhez rendelve:

- RA0, RA1,RA2, ...RA7 → PORTA, TRISA regiszterek
 - RB0, RB1,RB2, ...RB7 → PORTB, TRISB regiszterek
 - RC0, RC1,RC2, ...RC7 → PORTC, TRISC regiszterek
 - RD0, RD1,RD2, ...RD7 → PORTD, TRISD regiszterek
 - RE0, RE1,RE2, (RE3) → PORTE, TRISE regiszterek
-
- TRISx regiszter bitjei állítják be az irányokat, ha 0 → kimenet, ha 1 → bemenet
 - PORTx regiszter bitjein keresztül pedig a hozzá rendelt lábakra lehet írni, vagy be lehet olvasni a láb értékét (attól függően, hogy éppen be- vagy kimenetnek van beállítva

2.8. PIC16F887 jellemzői

TRISx, PORTx regiszterek

| | | | | | | | | |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TRISD | RD7 | RD6 | RD5 | RD4 | RD3 | RD2 | RD1 | RD0 |

RD0, RD1, RD2, ...RD7 lábak irány beállítása,
1 → bemenet, 0 → kimenet

| | | | | | | | | |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PORTD | RD7 | RD6 | RD5 | RD4 | RD3 | RD2 | RD1 | RD0 |

RD0, RD1, RD2, ...RD7 lábak írása vagy olvasása (attól függ bemenet vagy kimenet)

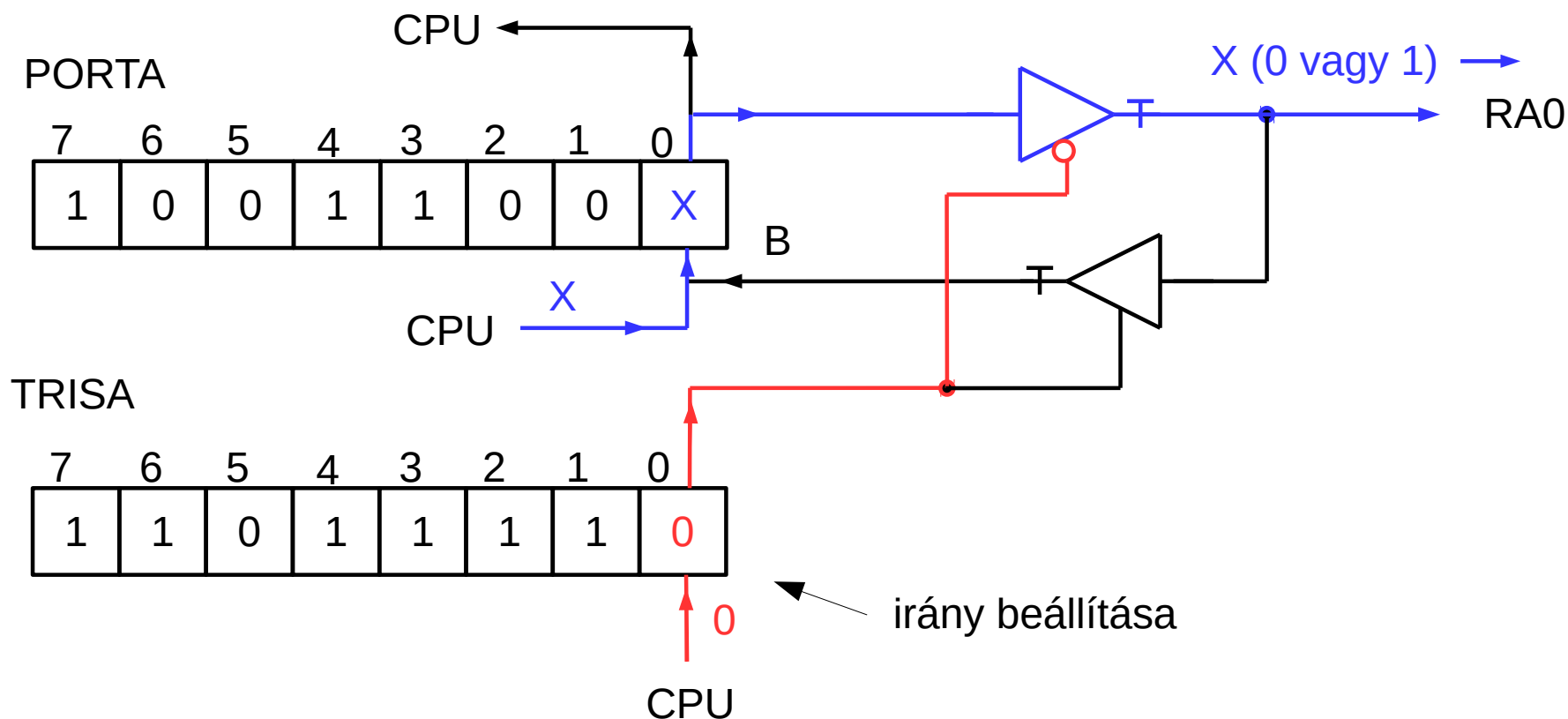
| | | | | | | | | |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TRISC | RC7 | RC6 | RC5 | RC4 | RC3 | RC2 | RC1 | RC0 |

RC0, RC1, RC2, ...RC7 lábak irány beállítása,
1 → bemenet, 0 → kimenet

PORTC, TRISB, PORTB, TRISA, PORTA, TRISE, PORTE
hasonlóan

2.9. PIC16F887 jellemzői

Digitális bemenetek/kimenetek, kimenetként



TRISA 0. bit és PORTA 0. bit → RA0 láb vezérlése, írása

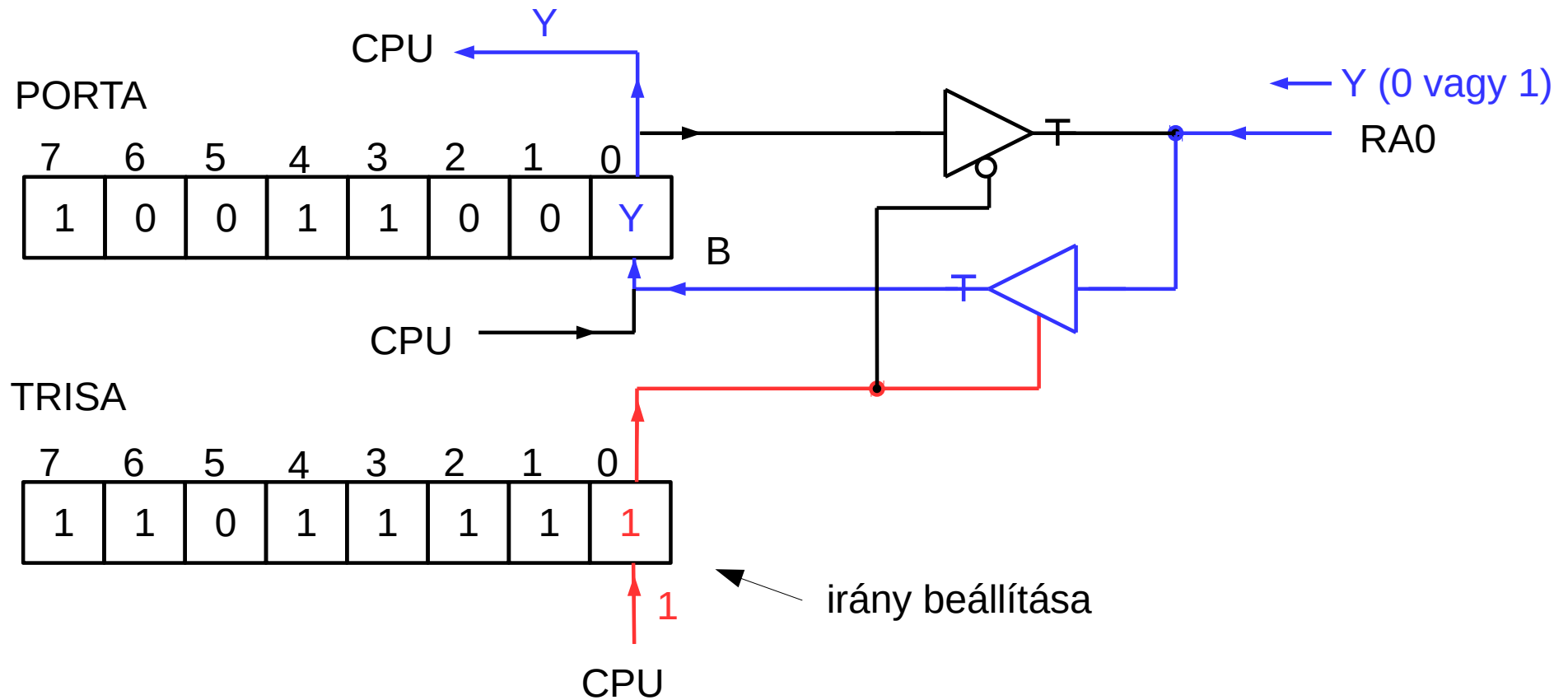
TRISA 1. bit és PORTA 1. bit → RA1 láb

TRISA 2. bit és PORTA 2. bit → RA2 láb

.....

2.10. PIC16F887 jellemzői

Digitális bemenetek/kimenetek, bemenetként



3.1. Programozási lehetőségek

- Gépi kód

Az adott mikroprocesszor (mikrovezérlő) saját nyelve.

Bináris !

pl. 0100 101 0000011 → a 3. regiszter 5. bitjének nullázása (BCF 0x03,5)
pl.2 00000100000000 → W regiszter nullázása (CLRW)

Hátránya: használata nehézkes, időigényes, → lassú program fejlesztés!

- Assembly nyelv

szimbolikus nyelv → a gépi kód minden utasításához egy rövid név (mnemonik) tartozik

pl. BCF 0x03,5 (BCF f,b → bit clear f reg.)
pl.2 CLRW → W regiszter nullázása (clear W)

előnye: gyors működésű, kis méretű programok készítése

hátránya: használata még így is nehézkes, fordítóprogram kell hozzá ! → assembler (pl. MPASM)

3.2. Programozási lehetőségek

- Magas szintű programozási nyelv

közelebb van az emberi logikához, nyelvhez

előnye: használata egyszerű → gyors szoftver fejlesztés

hordozható programok (elrejtí a hardvert! → ez hátrány is lehet)

hátránya: nagyobb méretű, lassúbb programok

fordítóprogram kell hozzá !

ilyen nyelvek: Pascal, C, Java, ...

3.3. Programozás elmélet alapok

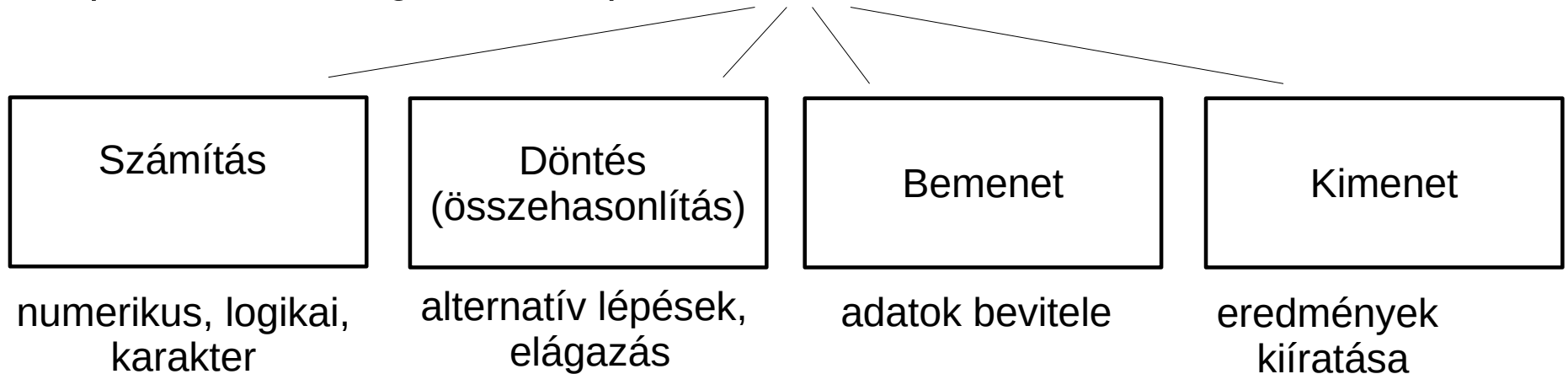
Számítógépes probléma megoldás

Bemenő adatok → kimenő adatok, és köztük összefüggések

Megfelelő műveletekkel a kimeneti adatok meghatározása

Algoritmus

- Műveletek sorozata, amely lehetővé teszi a feladat megoldását
- lépésekből áll, algoritmus lépés → művelet



3.4. Programozás elmélet alapok

Program tervezés lépései

-Lényege a megfelelő algoritmus összeállítása, kidolgozása

-algoritmus ellenőrzése

Nincs hiba

Hiba ! → újratervezés

-algoritmus lekódolása (valamilyen program nyelven)

-program tesztelése

OK

Hiba ! → Kód javítása, újrakódolás

Dokumentálás

Programtervezési módszerek

Strukturált
programozás

Moduláris
programozás

Felülről lefelé
programozás

Folyamatábra
készítés

Objektum orientált
programozás

3.5. Programozás elmélet alapok

Moduláris programozás

- a teljes programot részekre osztjuk → modulok (alprogramok)
- egy-egy kis modul egyszerűbb, könnyebben átlátható



- hátrány: bonyolult, szövevényes kapcsolatok, egymásra hatások

Felülről lefelé programozás

- először a komplett program megírása leegyszerűsítve, nem részletesen kidolgozva
(az egyes részeknek, eljárásoknak csak a neve, feladata ismert még)
- az egyes részeket, alprogramokat tovább finomítjuk, bontjuk
- tehát fokozatosan haladunk lefelé a részletek kidolgozásával

3.6. Programozás elmélet alapok

Folyamatábra módszer

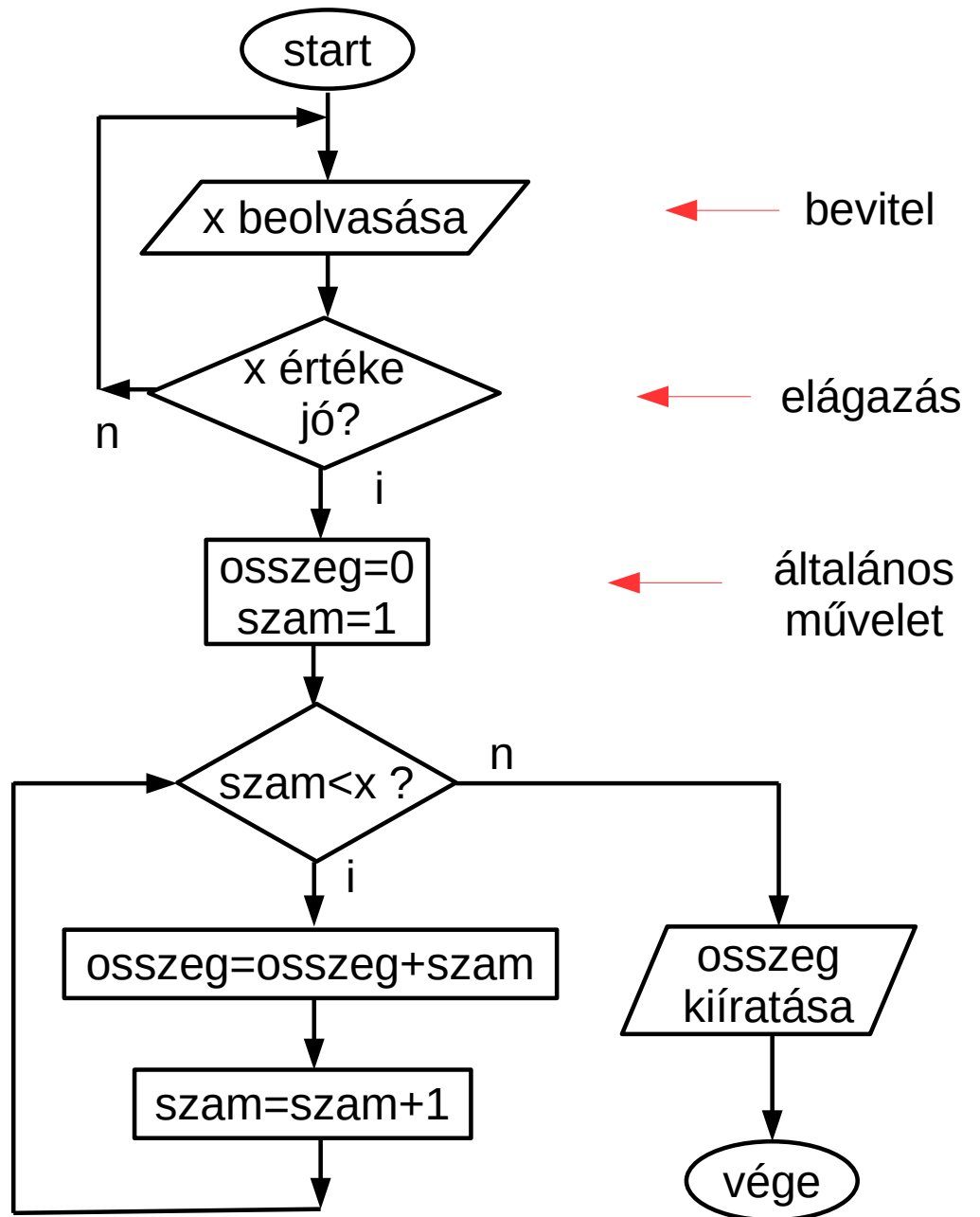
- grafikus szimbólumok
- milyen részletes legyen ?
- nem mindig egyszerű leködni

pl. az első 'x' db egész szám
összege

folyamat vonal

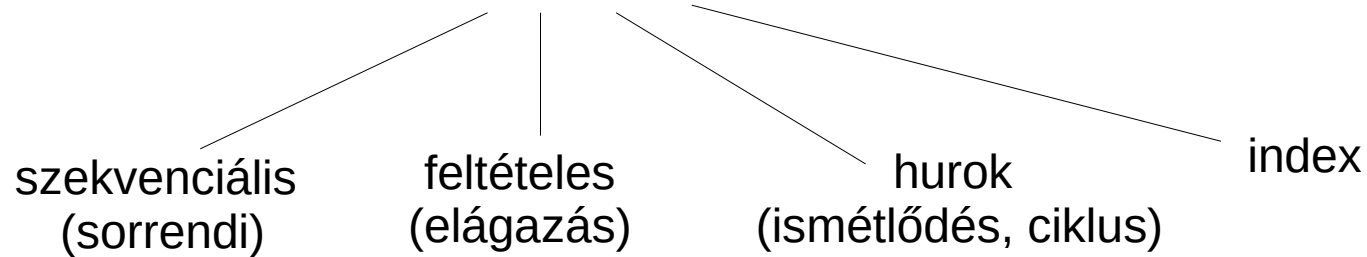
hozzáadás

szam növelése

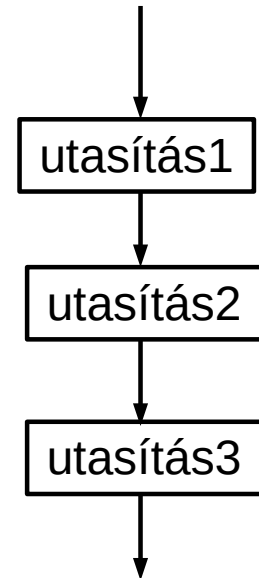


3.7. Programozás elmélet alapok

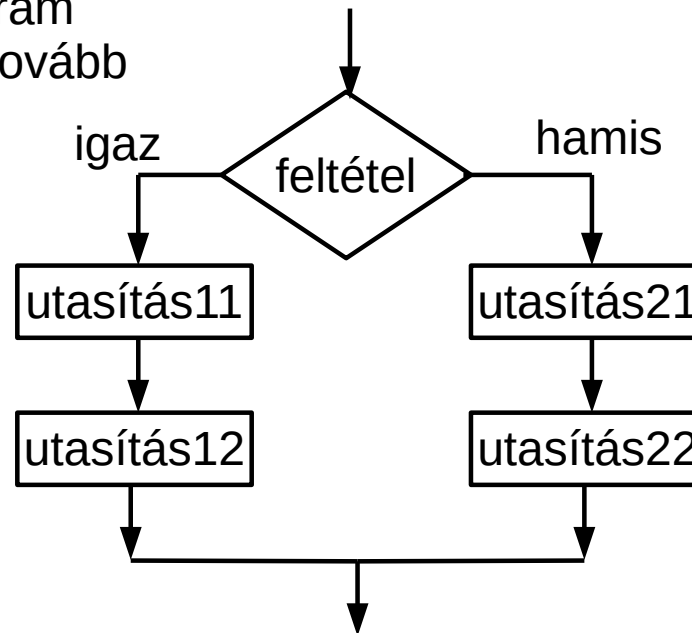
Egy program alapvető struktúrákból épül fel



-szekvenciális struktúra: az utasítások egymás után sorban (ez az alap)

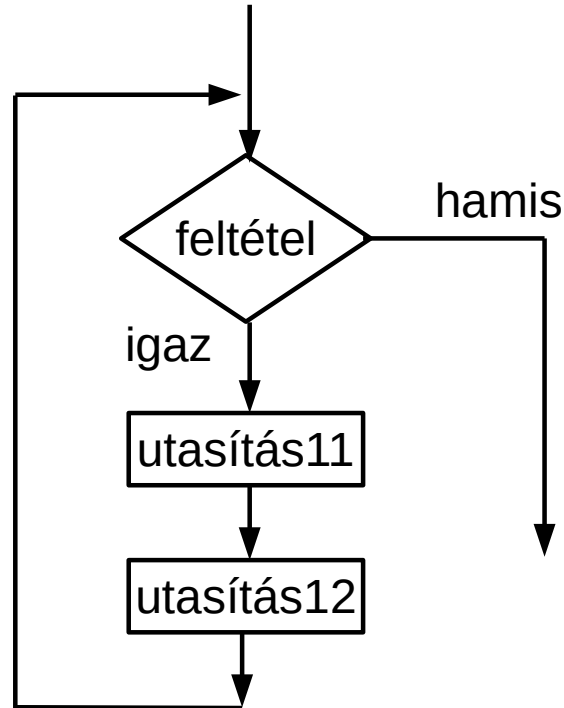


-feltételes struktúra: a program végrehajtás két úton megy tovább (igaz és hamis ág)



3.8. Programozás elmélet alapok

-hurok struktúra (ciklus): feltételtől függően ismétlünk utasításokat



-index struktúra: hasonló mint a feltételes struktúra, de a program végrehajtás nem két úton hanem több úton futhat tovább az 'index' értékétől függően

3.9. Feladatok

- 1. feladat

Készíts folyamatábrát a következő problémára: 1 és 100 közötti, 3-mal osztható számok összegének kiszámítása

- 2. feladat

Készíts folyamatábrát a következő problémára: 50 és 200 közötti, 5-el osztható számok kiírása, nagyságuk szerinti csökkenő sorrendben

- 3. feladat

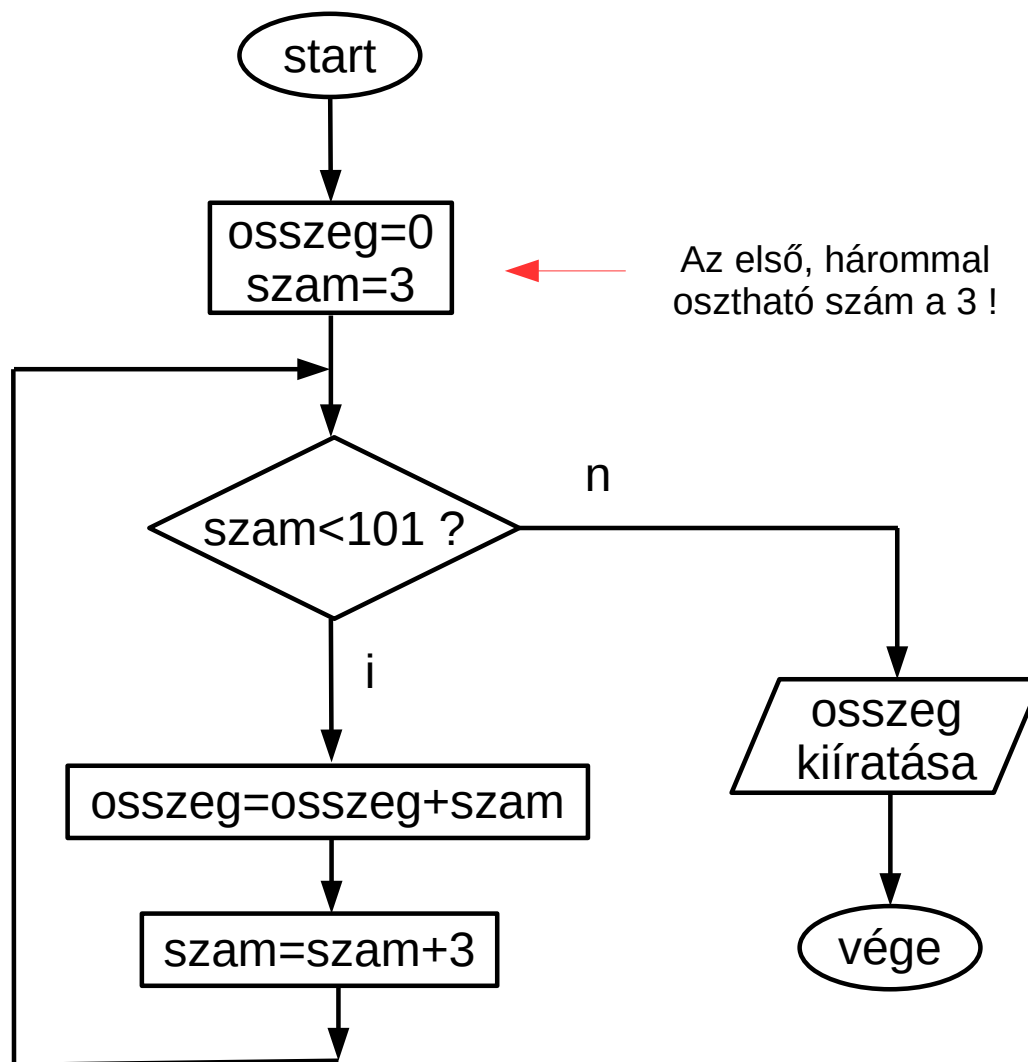
Készíts folyamatábrát a következő problémára: 30 és 90 közötti, 3-al osztható számok kiírása, majd utána 100 és 200 közötti páratlan számok kiírása

3.10. Feladatok megoldásai

- 1. feladat

1 és 100 közötti,
3-mal osztható számok
összegének kiszámítása

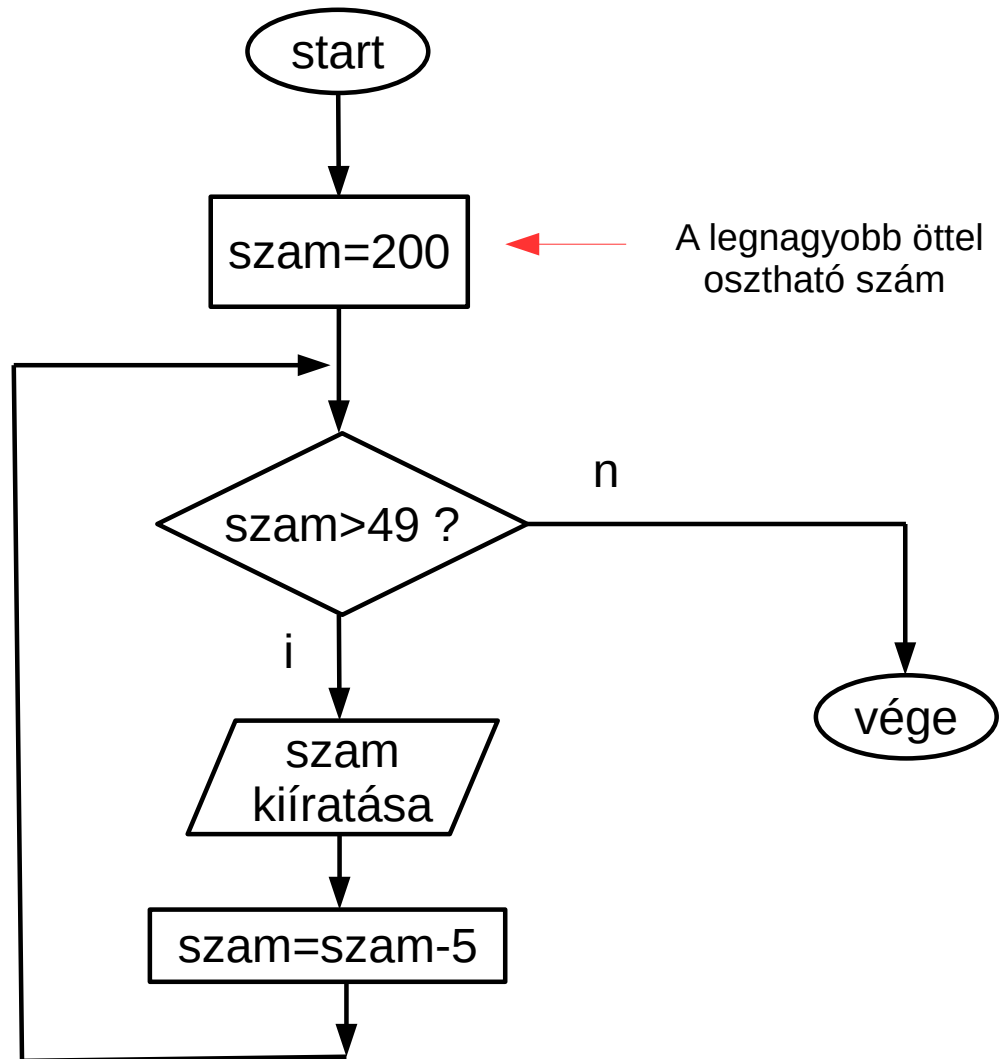
Ciklus, meddig
kell összeadni? →



3.11. Feladatok megoldásai

- 2. feladat

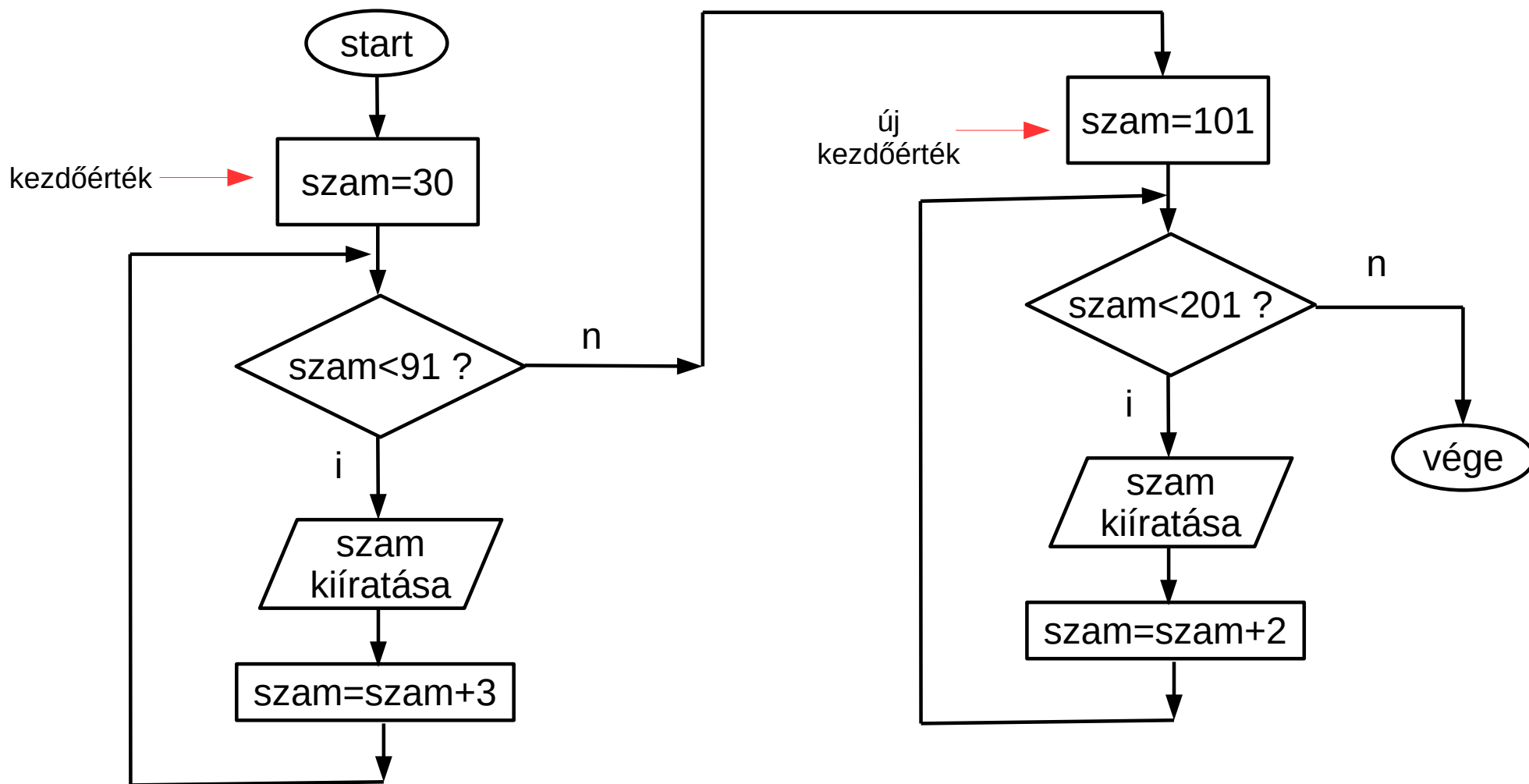
50 és 200 közötti, öttel
osztható számok kiírása,
nagyságuk szerinti
csökkenő sorrendben



3.12. Feladatok megoldásai

- 3. feladat

30 és 90 közötti, 3-al osztható számok kiírása,
majd utána 100 és 200 közötti páratlan számok kiírása



4.1. C nyelv, bevezetés

- C nyelv: magas szintű, de mégis nagyon hardver közeli, tömör nyelv
- Sokféle processzor típusra létezik C fordító (általában többféle is)

PIC-ek esetén C nyelven programozhatunk:

- a '**MikroC pro for PIC**' programmal (MikroElektronika – www.mikroe.com), amely fejlesztő környezet és C fordító egyben
- az 'MPLAB' fejlesztő környezetben, 'XC8' fordítóval (8 bites PIC-ekhez !)
(Microchip – www.microchip.com)

AVR-ek esetén C nyelven programozhatunk:

- a 'MikroC pro for AVR' programmal (MikroElektronika), amely fejlesztő környezet és C fordító egyben
- 'WinAVR + Code::Blocks' vagy 'avr-gcc + Code::Blocks' programokkal
(www.codeblocks.org)
- az 'AVR Studio' fejlesztő környezetben (Atmel – www.atmel.com)

4.2. C nyelv, bevezetés

Egy C nyelvű program szerkezete

- függvényekből (alprogramokból) áll, minimum 1 függvény kell, ez a **'main'** függvény
- a main függvény a fő függvény, vele kezdődik a program végrehajtása, nem hagyható el !
- egy függvény fejrészét (visszatérési érték típusa, függvény neve, és zárójelek között paraméterek) a függvény törzse követi → { } zárójelek között

egy egyszerű C program tehát így néz ki:

```
void main( )      // void → nincs visszatérési érték
{
    definíció1;    // sorok végén megjegyzések lehetnek '/' után !!!
    definíció2;    // definíciók (deklarációk) után pontosvessző kell (;)
    ...
    utasítás1;     // utasítások után szintén pontosvessző kell !!
    utasítás2;
    utasítás3;
    ...
}
```

4.3. C nyelv, bevezetés

Adat típusok

C nyelven sokféle típusú adattal dolgozhatunk, a legfontosabb néhány:

- **char** → egy karakter tárolására szolgál, pl. 'g' 'k' '5' '+'
igazából egy 8 bites számként (0 - 255) tárolódik (ASCII kód)
- **int** → előjeles egész szám, 16 bites (-32768 – +32767)
- **float** → valós szám, lebegőpontos, 32 bites
($\pm 1,17 \cdot 10^{-38}$ – $\pm 6,8 \cdot 10^{38}$)
- **long int** → előjeles egész szám, 32 bites
- **signed char** → előjeles egész, 8 bites, (-128 – +127)
- **unsigned int** → előjel nélküli egész szám, 16 bites, (0 – 65535)
- ...

Változó

- egy adatot tárol, értéke (a tárolt adat) általában változik a program futása során
- van neve (betűkből, számokból és az aláhúzás karakterből állhat) → igazából egy memória rekeszt címez meg (a RAM-ban tárolódnak)
- van típusa ! (milyen típusú adatot tárol)

4.4. Változók, konstansok

Változók használata

- használatuk előtt deklarálni, definiálni kell őket → **típus név (= érték);**
pl.

```
int szam1;           // szam1 nevű, egész típusú változó
char betu;           // betu nevű, karakter típusú változó
char betu2='B';       // betu2 nevű változó, B kezdő értékkel
int sorszam_3=2;      // sorszam_3 nevű egész, 2 kezdő értékkel
char szam2=0b10101100; // kezdőérték 2-es számrendszerben
char szam16=0xA7;     // kezdőérték 16-os számrendszerben
```

- **értékadás**, később bármikor a programban '=' használatával
pl.

```
szam1=5;             // most már szam1 értéke 5 lesz, amíg
                     // meg nem változtatjuk
betu2='C';            // most már betu2 értéke C lesz (nem B)
szam2=0x24;           // szam2 értéke 2416 lesz (36)
```

4.5. Változók, konstansok

Változók használata

- vannak foglalt nevek, ilyen nevű változókat nem hozhatunk létre
(a C nyelv utasításai, típusnevei ... pl. if, for, while, char)

Konstans

- értéke állandó, nem változtatható meg !!
- lehet: egész, karakter, valós (lebegőpontos)
- használatuk előtt deklarálni kell őket → **const név = érték;**

pl.

```
const MAX=200;    // egész, megadás 10-es számrendszerben
const MIN=0x2A;   // egész, megadás 16-os számrendszerben (0x...)
const SZAM=0b10011100; // egész, 2-os számrendszerben (0b...)
const BETU='T';   // karakter konstans
const PI=3.14;    // valós konstans
const NMAX=2.5E6; // valós konstans,  $2.5 \cdot 10^6$ 
```

4.6. Műveletek, operátorok, kifejezések

Aritmetikai műveletek

- a négy alpművelet operátorai: $+$ $-$ $*$ $/$
- maradékos osztás: $\%$
- növelés 1-el (increment): $++$ - csökkentés 1-el (decrement): $--$

Értékadás

változónév=kifejezés;

először kiértékelődik az egyenlőség jel jobb oldalán lévő kifejezés →
és a kapott értéket veszi fel a baloldali változó

```
void main( ) {  
    int szam1, szam3; // két egész változó létrehozása  
    szam1=4+5*2;      // szam1 értéke 14 lesz !!  
                      // (először a szorzás lesz elvégezve)  
    szam3=(szam1-2)/2; // szam3 értéke (14-2)/2=6 lesz  
    szam1=9%4;        // szam1 értéke 1 lesz  
                      // (az egész osztás maradéka)  
    szam1++;          // szam1 értéke 1-el növelődik ! → 2 lesz  
    szam3=szam1*sam1; // szam3 értéke 22=4 lesz  
    szam3--;          // szam3 értéke 1-el csökken ! → 3 lesz  
}
```

4.7. Műveletek, operátorok, kifejezések

Logikai műveletek, operátorok

- ÉS (AND): **&&** pl. **a&&b**
- VAGY (OR): **||** pl. **c||d**
- NEM (NOT): **!** pl. **!d**

Relációs műveletek

- nagyobb: **>**
- kisebb: **<**
- egyenlő: **==**
- nem egyenlő: **!=**
- nagyobb vagy egyenlő: **>=**
- kisebb vagy egyenlő: **<=**

pl.

```
x<10           // igaz értéket ad ha x kisebb mint 10
y==5           // igaz értéket ad ha y egyenlő 5-el
(x>10)&&(x<20)  // igaz értéket ad ha x 10 és 20 közé esik
                // (nagyobb mint 10 ÉS kisebb mint 20)
```

C program nyelven ha valahol logikai értékre van szükség, de szám van ott → a számok automatikusan átkonvertálódnak logikai értékre !!

0 → hamis, bármilyen 0-tól különböző szám → igaz

4.8. Feltételes utasítás

Feltételes utasítás: **if**

elágazást hoz létre a programban

szintaktikája:

```
if (feltétel) { igaz ág utasításai }  
else { hamis ág utasításai }
```

pl.

```
if(x<10) { szam=2; x++; }  
else { szam=1; ... } // az else rész elhagyható !
```

De gyakoribb a következő alak:

```
if(x<10)  
{  
    szam=2;  
    x++;  
}  
else  
{  
    szam=1;  
    ...  
}
```

igaz ág

hamis ág

4.9. Feltételes utasítás

Feltételes utasítás használata

pl. a programban folyamatosan számolni kell 1-től 8-ig, majd kezdeni előlről (1,2,3,4,5,6,7,8,1,2,3,)

```
char szamlal=1;
```

```
...
```

```
...
```

```
if(szamlal<8) szamlal++;
```

```
else szamlal=1;
```

```
// ha csak egy utasítás van
```

```
// akkor { } elhagyható !
```

```
...
```

```
...
```

megoldás másféleképpen:

```
...
```

```
szamlal++; // először növelünk
```

```
if(szamlal>8) szamlal=1;
```

```
// ha túlléptük a határt →
```

```
// kezdőérték beállítása újra
```

```
...
```


4.10. Ciklus utasítás

Ciklus utasítás:

ha többször ismételni akarunk utasításokat

- többféle is van ! → **while**, for, do-while

- while szintaktikája:

while (feltétel) { ismétlendő utasítások }

amíg a feltétel igaz addig ismétli az utasításokat

pl.

```
char x=1;
int szam=0;

while(x<4)
{
    szam=szam+x;
    x++;
}
```



Megfelel ennek:

```
char x=1;
int szam=0;

szam=szam+x; // (x=1) szam=0+1
x++;         // x=1+1
szam=szam+x; // (x=2) szam=1+2
x++;         // x=2+1
szam=szam+x; // (x=3) szam=3+3
x++;         // x=3+1
```

- `szam=szam+x;` → a `szam` változó jelenlegi értékéhez
x értékét → ez lesz a `szam` új értéke

hozzáadjuk

4.11. Ciklus utasítás

Ciklus utasítás használata

pl. adjuk össze az 1 és 101 közötti páros számokat
(tehát $2+4+6+8+\dots+100$)

```
char szam=2;  
int osszeg=0;
```

```
while(szam<101) // csak 100-ig adjon össze  
{  
    osszeg=osszeg+szam;  
    szam=szam+2; // szam növelése 2-vel  
}
```

Végtelen ciklus

- ha a ciklus feltétele mindig igaz → soha nem lesz vége az ismétlésnek !!
- ez gyakran hiba eredménye, de lehet hogy mi szeretnénk ezt
- mikrovezérlőknél a fő program résznek végtelen ciklusban célszerű futnia

4.12. C és a mikrovezérlők

C nyelvű program szerkezete mikrovezérlők esetén

- a 'main' függvényen belül a kezdeti deklarációk, értékadások, beállítások után egy végtelen ciklusban futtatjuk a keretprogramot
- így amíg a mikrovezérlő tápfeszültséget kap fut rajta a program

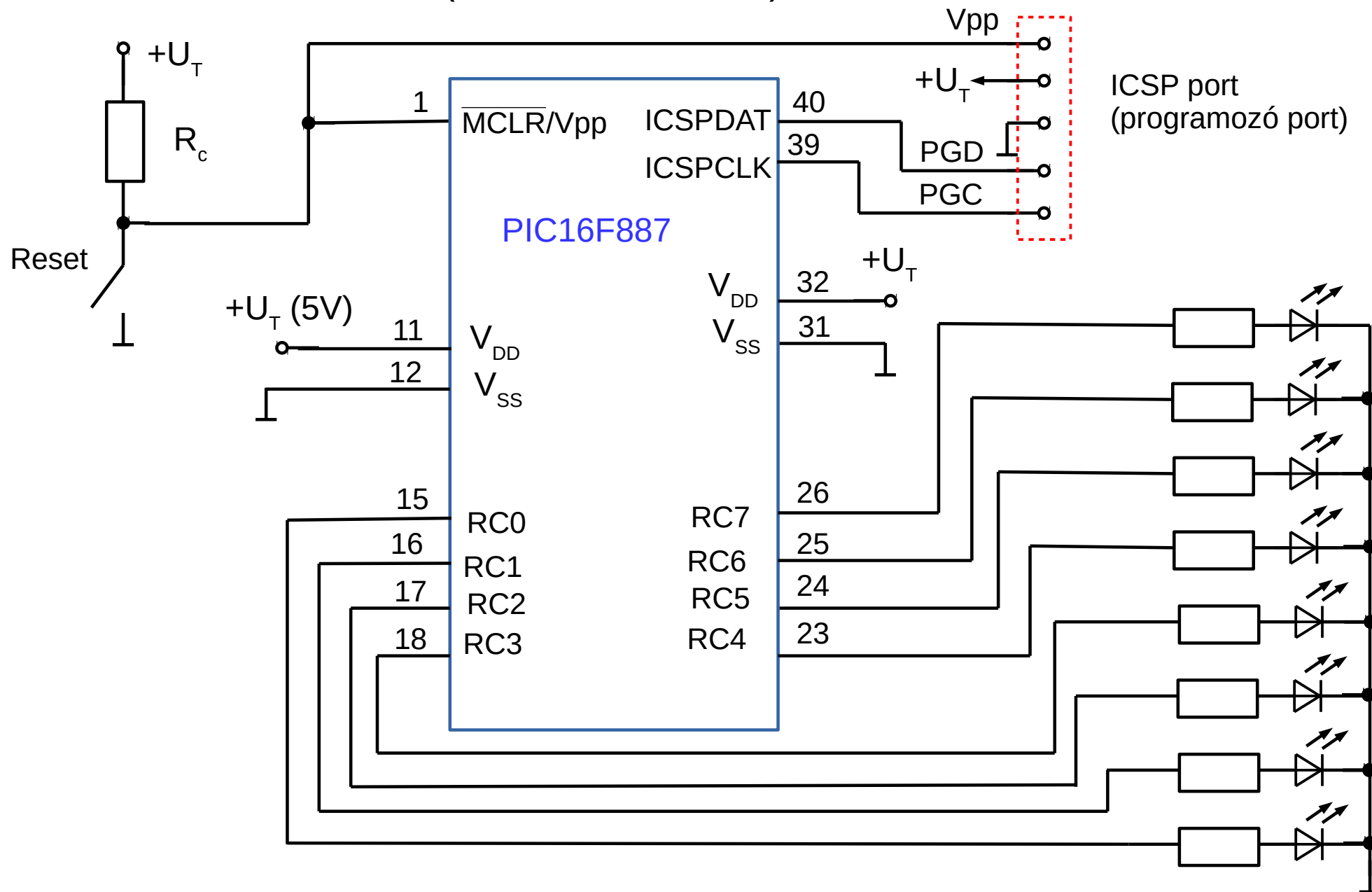
egy egyszerű C program tehát így néz ki mikrovezérlőn:

```
void main( )  
{  
    definíció1;      // kezdeti beállítások  
    ...  
    kezd_utasítás1;  
    kezd_utasítás2;  
    ...  
    while (1) {  
        utasítás1;  
        utasítás2;  
        ....  
    }  
}
```

← végtelen ciklus

5.1. PIC kimenetek vezérlése

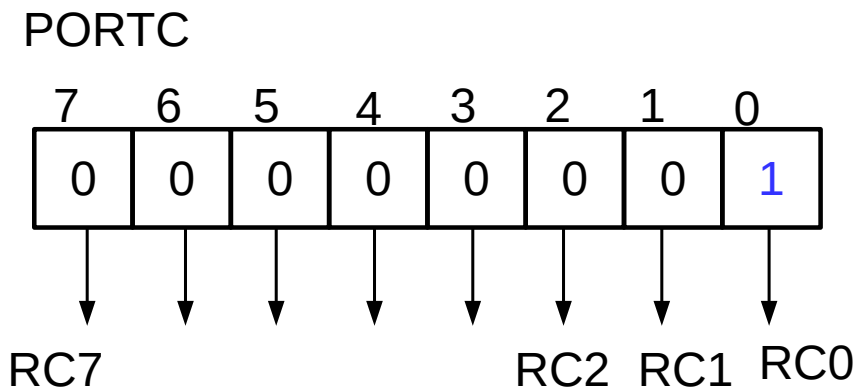
LED-ek a PORTC lábakon (RC0, RC1, ... RC7)



5.2. A LED-ek vezérlése

Az RCx kivezetéseket PORTC, TRISC regiszterekkel vezéreljük

- TRISC bitjeivel beállítjuk, hogy **a lábak kimenetek** legyenek →
TRISC=0b00000000; (vagy TRISC=0;)
- egy LED felkapcsolása → a megfelelő lábra '1' szint (~5V)
 - RC0 láb**on lévő LED → **PORTC=0b00000001;** (vagy PORTC=1;)
 - RC1 láb**on lévő LED → **PORTC=0b00000010;** (vagy PORTC=2;)
 - RC2 láb**on lévő LED → **PORTC=0b00000100;** (vagy PORTC=4;)
 - RC3 láb**on lévő LED → **PORTC=0b00001000;** (vagy PORTC=8;)
 - ...
 - RC7 láb**on lévő LED → **PORTC=0b10000000;** (vagy PORTC=128;)
- egy LED lekapcsolása → a megfelelő lábra '0' szint (~0V)



5.3. A LED-ek vezérlése

1. mintafeladat

- RC2 lábon lévő LED felvillantása kétszer (közben 5s szünet)
- késleltetésnek a MikroC beépített Delay_ms() függvényét használjuk

// 1. RC2 LED kétszer

```
void main( )
```

```
{
```

```
    TRISC=0b00000000;
```

```
// minden RCx láb kimenet
```

```
    PORTC=0b00000100;
```

```
// RC2 lábra 5V → LED világít
```

```
    Delay_ms(5000);
```

```
// 5000ms=5s késleltetés
```

```
    PORTC=0b00000000;
```

```
// RC2 lábra 0V → LED elalszik
```

```
    Delay_ms(5000);
```

```
// 5000ms=5s késleltetés
```

```
    PORTC=0b00000100;
```

```
// RC2 lábra 5V → LED világít
```

```
    Delay_ms(5000);
```

```
// 5000ms=5s késleltetés
```

```
    PORTC=0b00000000;
```

```
// RC2 lábra 0V → LED elalszik
```

```
}
```

FONTOS !! Először egy jó darabig digitális kimenetek és bemenetek számára a PORTC-re és PORTD-re kapcsolt lábakat fogjuk használni. Használhatjuk ilyen célokra természetesen **PORTA és PORTB** lábait is, **DE** azok többsége analóg bemenet is lehet, és az analóg bemenet az alapbeállítás !!! Hogy digitális ki(be)menetként használjuk azokat → **az ANSEL és ANSELH regisztereket kell megfelelően nullázni !!**

5.4. Ciklus használata

2. mintafeladat

- RC2 lábon lévő LED felvillantása 30-szor !! (közben 2s szünet)
- nyilván lehetne az előbbi sorokat 15-ször egymás alá másolni, de célszerűbb és szebb ciklussal megoldani

// 2. RC2 LED 30-szor

```
void main( )
{
    char szam=1;                // e változó tárolja az aktuális számot
    TRISC=0b00000000;          // minden RCx láb kimenet
    while (szam<31)              // ismétlés 30-szor ! szam: 1,2,..30
    {
        PORTC=0b000000100;      // RC2 lábra 5V → LED világít
        Delay_ms(2000);          // 2000ms=2s késleltetés
        PORTC=0b00000000;      // RC2 lábra 0V → LED elalszik
        Delay_ms(2000);
        szam++;                  // szam növelése 1-el
    }
}
```

5.5. Végtelen ciklus használata

3. mintafeladat

- RC2 lábon lévő LED felvillantása sokszor !! (közben 2s szünet)
- tehát addig villogjon amíg tápfeszültség van
- nyilván az eddigi megoldások nem használhatók, ha sokszor egymás alá másoljuk a sorokat akkor is előbb-utóbb leáll
→ végtelen ciklus kell !!

// 3. RC2 LED sokszor

```
void main( )
{
    TRISC=0b00000000;    // minden RCx láb kimenet
    while (1)            // ismétlés sokszor !
    {
        PORTC=0b00000100;    // RC2 lábra 5V → LED világít
        Delay_ms(2000);      // 2000ms=2s késleltetés
        PORTC=0b00000000;    // RC2 lábra 0V → LED elalszik
        Delay_ms(2000);      // 2000ms=2s késleltetés
    }
}
```


5.6. Végtelen ciklus használata

4. mintafeladat, futófény

- A LED-ek sorban, egymás után világítsanak (1s-ig)
- tehát először RC0, majd RC1, RC2, RC7, majd előlről, RC0, RC1,...

lábak → RC7 RC6 RC2 RC1 RC0

128 64 32 16 8 4 2 1 ← helyi értékek

PORTC
bitek →

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

→ 1

→ 2

→ 4

→ 8

→ 16

→ 32

→ 64

→ 128

érdeemes megfigyeln
hogy milyen számokat
kell egymás után a
PORTC regiszterbe
beírni, hátha lehet
egyszerű algoritmust
is használni !!

idő

5.7. Végtelen ciklus használata

4. mintafeladat, futófény, a. megoldás

egyszerű, de nem elegáns → a számokat sorban kiírjuk PORTC-re

```
void main( ) {           // 5.4.a futófény
    TRISC=0b00000000;    // minden RCx láb kimenet
    while (1)            // ismétlés sokszor !
    {
        PORTC=0b00000001; // RC0 lábra 5V → LED0 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b00000010; // RC1 lábra 5V → LED1 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b00000100; // RC2 lábra 5V → LED2 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b00001000; // RC3 lábra 5V → LED3 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b00010000; // RC4 lábra 5V → LED4 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b00100000; // RC5 lábra 5V → LED5 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b01000000; // RC6 lábra 5V → LED6 világít
        Delay_ms(1000);   // 1s késleltetés
        PORTC=0b10000000; // RC7 lábra 5V → LED7 világít
        Delay_ms(1000);   // 1s késleltetés
    }
}
```

5.8. Végtelen ciklus használata

4. mintafeladat, futófény, b. megoldás

ugyanaz mint az előző, de 10-es számrendszerben adjuk meg a számokat

```
void main( ) { // 5.4.b futófény
    TRISC=0b00000000; // minden RCx láb kimenet
    while (1) // ismétlés sokszor !
    {
        PORTC=1; // RC0 lábra 5V → LED0 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=2; // RC1 lábra 5V → LED1 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=4; // RC2 lábra 5V → LED2 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=8; // RC3 lábra 5V → LED3 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=16; // RC4 lábra 5V → LED4 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=32; // RC5 lábra 5V → LED5 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=64; // RC6 lábra 5V → LED6 világít
        Delay_ms(1000); // 1s késleltetés
        PORTC=128; // RC7 lábra 5V → LED7 világít
        Delay_ms(1000); // 1s késleltetés
    }
}
```

5.9. Végtelen ciklus és elágazás használata

4. mintafeladat, futófény, c. megoldás

Kicsit szebb megoldás, felhasználva, hogy sorban az
1,2,4,8,16,32,64,128,1,2,4,8,... számokat kell a PORTC-re írni

// 4.c futófény

```
void main( )
{
    char szam=1;           // e változó tárolja az aktuális számot
    TRISC=0b00000000;     // minden RCx láb kimenet
    while (1)              // ismétlés sokszor !
    {
        PORTC=szam;        // RCx lábra 5V → LEDx világít
        Delay_ms(1000);    // 1s késleltetés
        if(szam<128) szam=2*szam; // *2 → következő szám
        else szam=1;       // de ha elértük a 128-at → kezdés előlről
    }
}
```

5.10. Végtelen ciklus és elágazás használata

5. mintafeladat, futófény2

- A LED-ek sorban, egymás után világítanak (1s-ig), de kettő egyszerre
- tehát először RC0-RC1, majd RC1-RC2, RC6-RC7, majd előlről

lábak → RC7 RC6 RC2 RC1 RC0

128 64 32 16 8 4 2 1 ← helyi értékek

PORTC
bitek →

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|-------|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | → 3 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | → 6 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | → 12 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | → 24 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | → 48 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | → 96 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | → 192 |

idő

5.11. Végtelen ciklus és elágazás használata

5. mintafeladat, futófény2 megoldás

Sorban a 3,6,12,24,48,96,192,3,6,12,24,... számokat kell a PORTC-re írni

// 5. futófény2

```
void main( )
{
    char szam=3;           // e változó tárolja az aktuális számot
    TRISC=0b00000000;     // minden RCx láb kimenet
    while (1)              // ismétlés sokszor !
    {
        PORTC=szam;        // RCx lábra 5V → LEDx világít
        Delay_ms(1000);    // 1s késleltetés
        if(szam<192) szam=2*szam; // *2 → következő szám
        else szam=3;        // de ha elértük a 192-őt → kezdés előlről
    }
}
```

5.12. Feladatok

Írj programokat a PORTC-re kapcsolt LED-ek vezérlésére

- 1. feladat
 - A két szélső (RC0,RC7) LED felvillantása felváltva, 4s szünettel
 - ezt hatszor kell megismételni, majd a program leáll
- 2. feladat
 - A két szélső LED felvillantása felváltva, 5s szünetekkel, háromszor ismételve !
 - ezután villanjon fel ötször a két középső (RC3, RC4) LED, 2s szünetekkel
 - majd a program leáll
- 3. feladat
 - A két középső LED felvillantása egyszerre (2s), majd a két mellettük lévő egyszerre (RC2-RC5) 2s-ig, majd a következő kettő (RC1-RC6), utána RC0-RC7, mindegyik 2s-ig világít
 - ezután folyamatosan futófény: egyszerre mindig három LED világít 1s ideig
0-1-2 → 2-3-4 → 4-5-6 → 6-7-0 → 0-1-2 → 2-3-4 →

6.1. Tömb létrehozása

Tömb

- olyan változó, amely sok azonos típusú elemet (pl. egész számot) tárol
- tömb deklarációja

típus tömb_név[elemek_száma];

pl.

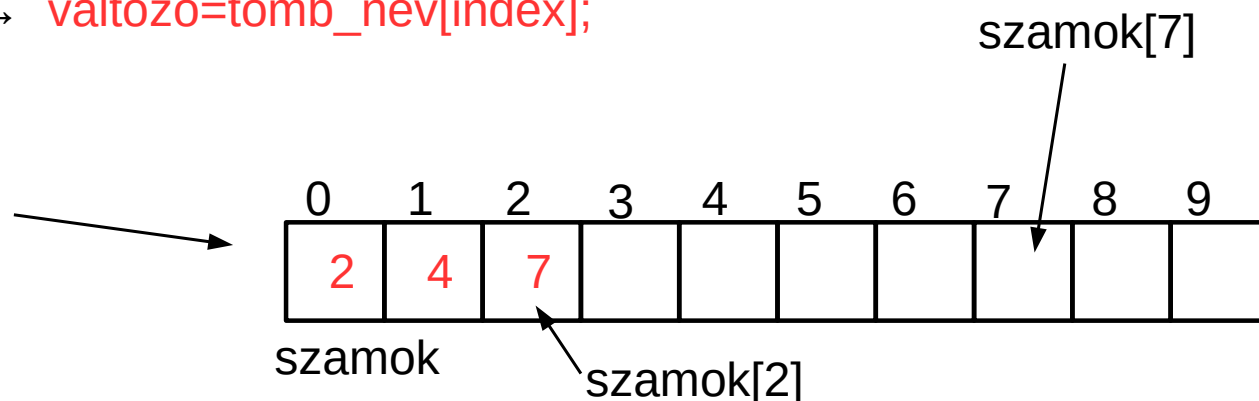
int szamok[10]; // 10 elemű tömb, 10db egész szám tárolására
// az elemek számozása 0-val kezdődik !!!



- hivatkozás a tömb elemeire → tömb_név[index]
értékkadás → tömb_név[index]=érték;
kiolvasás → változó=tömb_név[index];

pl.

szamok[0]=2;
szamok[1]=4;
szamok[2]=szamok[1]+3;



6.2. Tömb használata

Tömb és ciklus

- ha az összes tömbelemet akarjuk kiolvasni vagy értéket adni nekik → ciklus felhasználásával tudunk egyszerűen végig lépkedni a tömb elemeken

pl.

```
char szamok[10];           // 10 elemű tömb létrehozása
i=0;                       // index változó
while (i<10)               // ismétlés 10-szer, i=0,1,2,...9
{
    szamok[i]=2*i;         // tömbelem értéke legyen 2*index
    i++;                  // index növelése (következő ciklus számára)
}
```



| | | | | | | | | | |
|---|---|---|---|---|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 |

szamok

6.3. Tömb használata

Tömb, kezdőérték adás


- a tömb deklarálásakor azonnal megadhatjuk a tömbelemek értékeit is, ha már ismertek , így nem kell a feltöltéssel később vesződni

típus tömb_név[elemek_száma]={1.elem, 2.elem, 3.elem, ... k.elem};
vagy típus tömb_név[]={1.elem, 2.elem, 3.elem, ... k.elem};

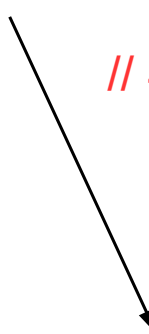
pl.

char szam8[]={3,4,5,6,27,25,23,68}; // 8 elemű tömb létrehozása

int szam4[4]={600,800,1200,100}; // 4 elemű tömb létrehoz



| | | | | |
|-------|-----|-----|------|-----|
| | 0 | 1 | 2 | 3 |
| szam4 | 600 | 800 | 1200 | 100 |



| | | | | | | | | |
|-------|---|---|---|---|----|----|----|----|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| szam8 | 3 | 4 | 5 | 6 | 27 | 25 | 23 | 68 |

6.4. Tömb használata

1. mintafeladat (futófény2) megoldása tömb használatával

Az ismétlendő 3,6,12,24,48,96,192 számokat kell tömbben tárolni !

// 6.1 futófény2 tömbbel

```
void main( )
{
    char led2[ ]={3,6,12,24,48,96,192}; // 7 elemű tömb (0,1,2,...6 !!)
    char i=0;                          // index változó
    TRISC=0b00000000;                  // minden RCx láb kimenet
    while (1)                           // ismétlés sokszor !
    {
        PORTC=led2[i];                 // az aktuális szám kiírása PORTC-re
        Delay_ms(1000);                 // 1s késleltetés
        i++;                            // index növelése
        if(i>6) i=0;                    // ha végig lépkedtünk a tömbön → kezdjük újra
    }
}
```

6.5. Bitműveletek

Bitműveletek

- bitenkénti ÉS (AND): $\&$

pl. $\text{szam1}\&\text{szam2}$
 $0b1011\&0b1101 \rightarrow 0b1001$

- bitenkénti VAGY (OR): $|$

pl. $\text{szam1}|\text{szam2}$
 $0b1010|0b0100 \rightarrow 0b1110$

- bitenkénti NEM (NOT): \sim

pl. $\sim\text{szam1}$
 $\sim 0b1011 \rightarrow 0b0100$

- kizáró VAGY (XOR): \wedge

pl. $\text{szam1}\wedge\text{szam2}$
 $0b1110\wedge 0b0100 \rightarrow 0b1010$

- eltolás balra (left shift) \ll

pl. $\text{szam1}\ll x$
→ eltolás x bittel
 $0b00001001\ll 2 \rightarrow 0b00100100$

- eltolás jobbra (right shift) \gg

pl. $\text{szam1}\gg x$
→ eltolás x bittel
 $0b11001010\gg 3 \rightarrow 0b00011001$

6.6. For ciklus

Ciklus utasítás

ha többször ismételni akarunk utasításokat

- többféle is van ! → while, **for**, do-while
- **for** szintaktikája:

for (ciklus változó kezdőérték; feltétel; ciklus változó léptetése)
{ ismétlendő utasítások }

amíg a feltétel igaz addig ismétli az utasításokat

- főleg akkor célszerű használni, ha adott (előre ismert) számú ismétlés szükséges (pl. tömbök kezelése)

// pl. a számok összeadása 1-től 20-ig
for ciklussal

```
char i;  
int osszeg=0;  
for(i=1;i<21;i++)  
    { osszeg=osszeg+i; }
```



// while ciklussal

```
char i=1;  
int osszeg=0;  
while(i<21)  
{  
    osszeg=osszeg+i;  
    i++;  
}
```

6.7. For ciklus

2. minta feladat

tömb feltöltése az első 30 páratlan számmal

```
void main( )
{
    char i;
    char szam;
    char tomb[ 30];    // 30 elemű tömb (0-29 !!)
    szam=1;            // az első páratlan szám az 1
    for(i=0;i<30;i++)  // ismétlés 30-szor (0-29)
    {
        tomb[i]=szam;  // az aktuális szám betöltése a tömbbe (i. elemébe)
        szam=szam+2;   // a következő páratlan szám meghatározása
    }
}
```

Működése:

- 1. ciklus → szam=1 és i=0 → tomb[0]=1 és szam=3
- 2. ciklus → szam=3 és i=1 → tomb[1]=3 és szam=5
- 3. ciklus → szam=5 és i=2 → tomb[2]=5 és szam=7
- 4. ciklus → szam=7 és i=3 → tomb[3]=7 és szam=9

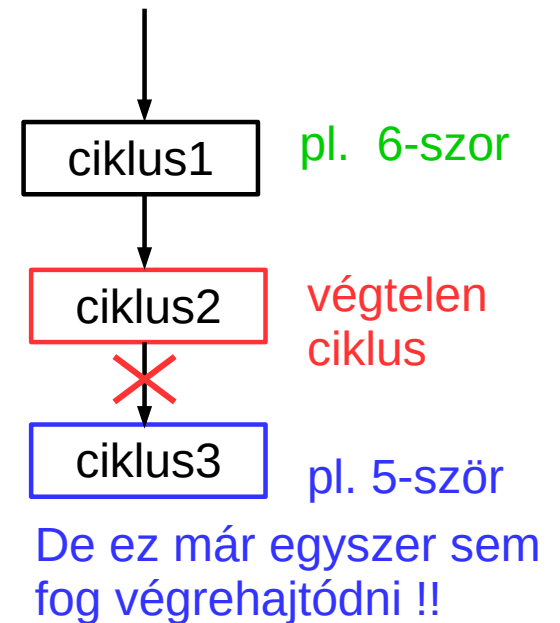
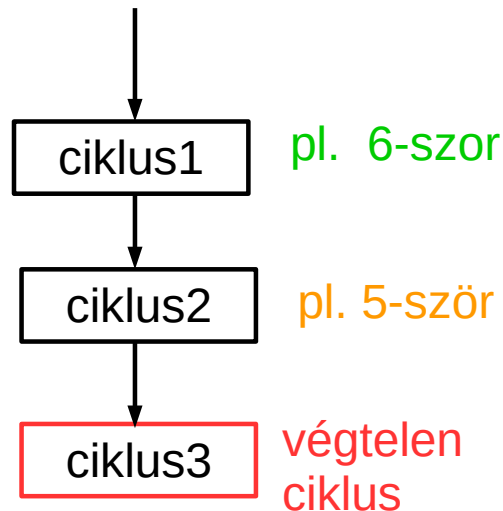
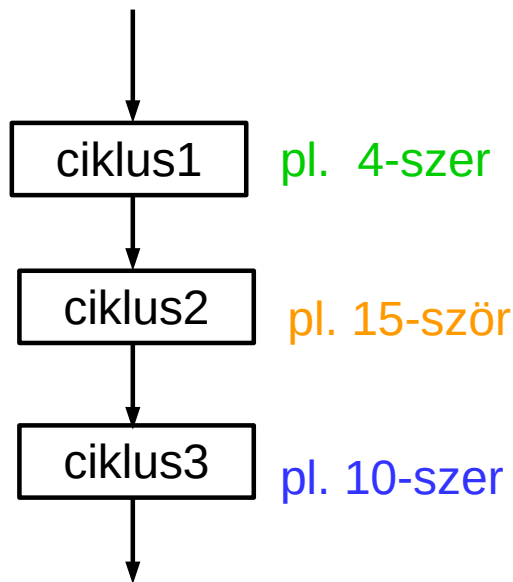
.....

30. ciklus → szam=59 és i=29 → tomb[29]=59 és szam=60 → vége

6.8. Több ciklus egymás után

Több ciklus egymás után

- Akkor használunk ilyen szerkezeteket, ha egymás után különböző dolgokat kell ismételni, elvileg bármennyi ciklust használhatunk egymás után
- viszont nyilvánvalóan csak az utolsó ciklus lehet végtelen ciklus !! →
 - mert a végtelen ciklus utáni utasítások (így a ciklusok is) már soha nem fognak végrehajtódni
- ciklusként természetesen akár for akár while is használható, bármilyen kombinációban



6.9. Több ciklus egymás után

3. mintafeladat

- A két szélső (RC0,RC7) LED felvillantása felváltva, 1s szünetekkel, **hatszor** ismételve !
- ezután villanjon fel **ötször** a két középső (RC3, RC4) LED egyszerre, 2s szünetekkel, majd a program leáll → két ciklus, első hatszor, második ötször

// 6.3 RC0-RC7 felváltva 6-szor, RC3,RC4 felvillan 5-ször

```
void main( )
{
    char i;                                // index változó
    TRISC=0b00000000;                      // minden RCx láb kimenet
    for(i=0;i<6;i++)                        // ismétlés 6-szor (0-5)
    {
        PORTC=0b00000001;                  // RC0 lábon lévő LED világít
        Delay_ms(1000);                     // 1s késleltetés
        PORTC=0b10000000;                  // RC7 lábon lévő LED világít
        Delay_ms(1000);                     // 1s késleltetés
    }
    for(i=0;i<5;i++)                        // ismétlés 5-ször (0-4)
    {
        PORTC=0b00011000;                  // RC3-RC4 lábakon lévő LED-ek világítanak
        Delay_ms(2000);                     // 2s késleltetés
        PORTC=0b00000000;                  // LED-ek nem világítanak
        Delay_ms(2000);                     // 2s késleltetés
    }
}
```

1. ciklus

2. ciklus

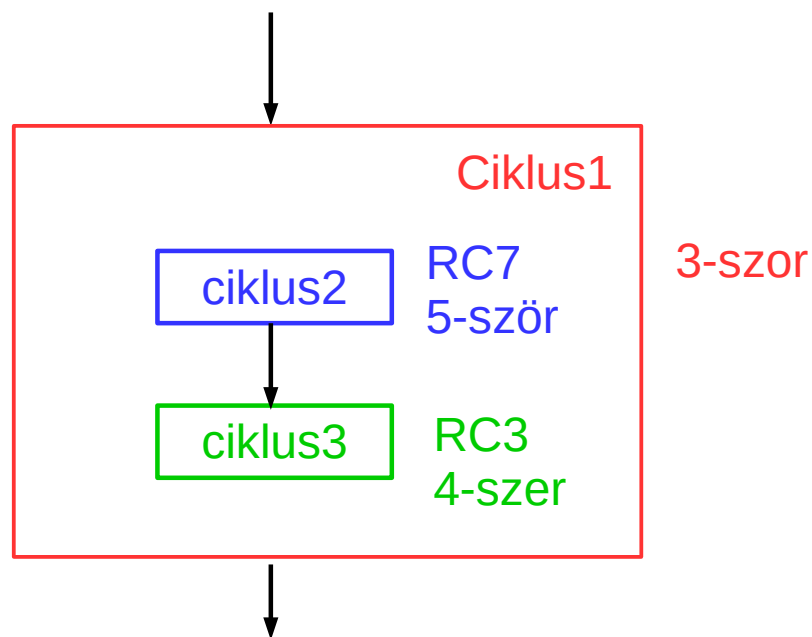
6.10. ciklus a ciklusban

Ciklus a ciklusban

- nemcsak egymás után lehetnek ciklusok, hanem egymáson belül is
- viszont nyilvánvalóan csak a legkülső ciklus lehet végtelen ciklus !!

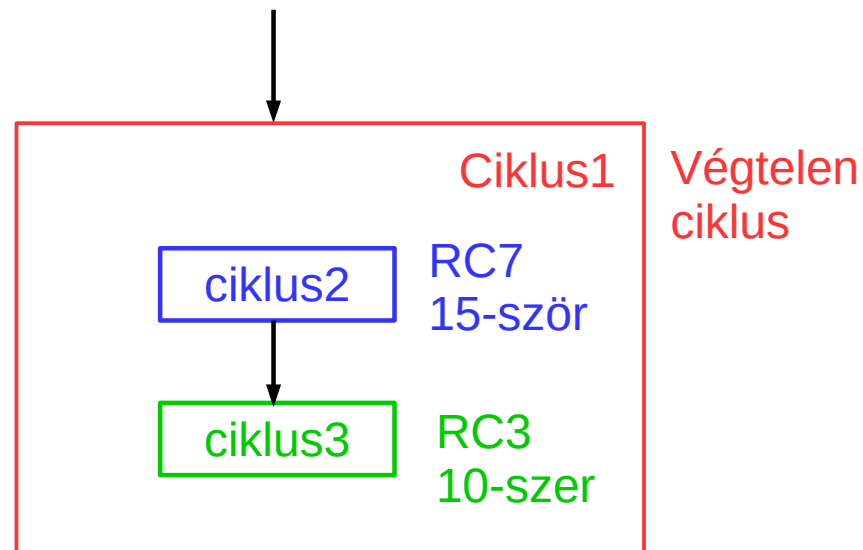
Példa1. PORTC-re kapcsolt LED-ek villogjanak a következőképpen:

Először RC7 5-ször, majd RC3 4-szer
majd RC7 5-ször, majd RC3 4-szer
majd RC7 5-ször, majd RC3 4-szer



Példa2. PORTC-re kapcsolt LED-ek villogjanak a következőképpen:

RC7 15-ször, majd RC3 10-szer
majd RC7 15-ször, majd RC3 10-szer,
... és így ismétlődjön folyamatosan



6.11. ciklus a ciklusban

4. mintafeladat

Futófény előre egyszer, majd hátra egyszer, majd előre, majd hátra, ... és így tovább amíg tápfeszültséget kap

```
void main( )      // 6.4 futófény előre-hátra-előre-hátra-...
{
    char led[ ]={1,2,4,8,16,32,64,128}; // 8 elemű tömb (0,1,2,...7 !!)
    int i=0;           // index változó, 16 bites előjeles egész !!
    TRISC=0b00000000; // minden RCx láb kimenet
    PORTC=led[i];      // utolsó LED világít (RC0)
    Delay_ms(1000);    // 1s késleltetés
    while (1)          // 1. ciklus, végtelen → ismétlés sokszor !
    {
        for(i=1;i<8;i++) // 2. ciklus, ismétlés 7-szer (1-2-...-7)
        {                // futófény előre
            PORTC=led[i];
            Delay_ms(1000); // 1s késleltetés
        }
        for(i=6;i>=0;i--) // 3. ciklus, ismétlés 7-szer (6-5-4-...-0)
        {                // futófény hátra
            PORTC=led[i];
            Delay_ms(1000); // 1s késleltetés
        }
    }
}
```

6.12. Feladatok

Írj programokat a PORTC-re kapcsolt LED-ek vezérlésére

- 1. feladat (ciklusokkal megoldani !)
 - RC0-RC1-RC2 LED felvillantása egyszerre, **4-szer**
 - majd RC7-RC6-RC5 LED felvillantása egyszerre, **3-szor**,
 - majd RC3-RC4 LED felvillantása egyszerre, **4-szer**,
 - majd a program leáll
- 2. feladat (ciklusokkal megoldani !)
 - A két szélső LED felvillantása felváltva, 1s szünetekkel, **4-szer** ismételve !
 - ezután villanjon fel **7-szer** a két középső (RC3, RC4) LED, 2s szünetekkel
 - majd a program leáll
- 3. feladat (ciklusokkal megoldani !)
 - A két szélső LED felvillantása egyszerre, 1s szünetekkel, **6-szor** ismételve !
 - ezután villanjon fel a két középső (RC3, RC4) LED, 2s szünetekkel sokszor!
→ folyamatosan, amíg tápfeszültséget kap