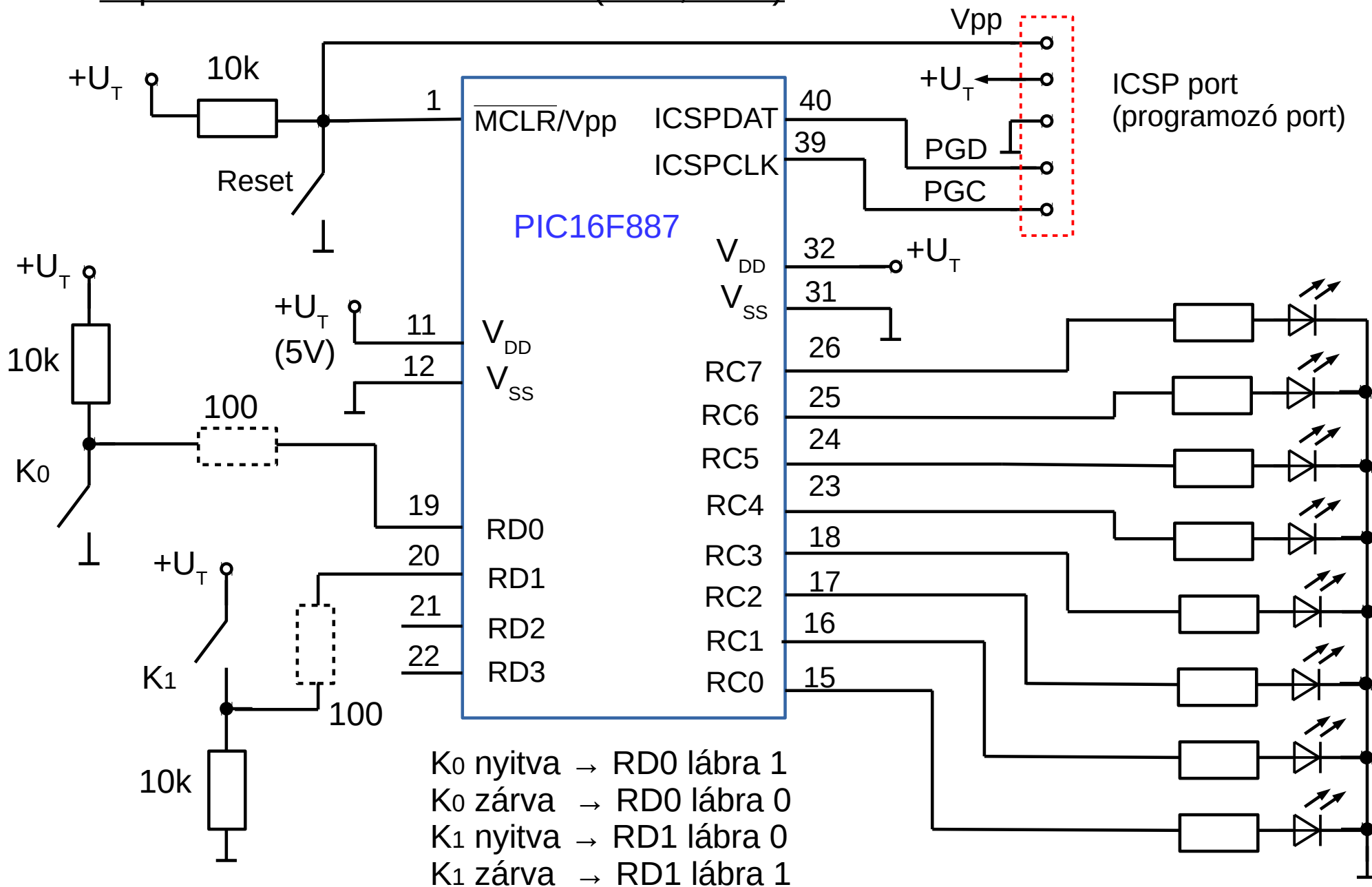


# PIC programozása 2. (PIC16F887)

7. PIC, digitális bemenetek kezelése
8. Hétsegmentes kijelző vezérlése
9. C programozás, függvények, megszakítás
10. LCD kijelző vezérlése

## 7.1. PIC digitális bemenetek

kapcsolók a PORTD lábakon (RD0, RD1)



## 7.2. A kapcsolók érzékelése

- Kapcsolók bekötése

Kétféleképpen köthetjük be a kapcsolókat

- kapcsolhatunk vele 1 szintet (tápfeszültséget) → a rajzon K1
- kapcsolhatunk vele 0 szintet → a rajzon K0

K1 nyitva → RD1 lábra 0

K1 zárva → RD1 lábra 1

K0 nyitva → RD0 lábra 1

K0 zárva → RD0 lábra 0

- Digitális bemenet beállítása

A TRISx regiszter megfelelő bitjét 1 értékűre kell állítani

- most RD0 és RD1 lábakra van szó → TRISD 0. és 1. bitjét 1-re  
(ha a többi RDx lábra nincs kötve semmi akkor mindegy hogy azokat hogyan állítjuk be)
- tehát `TRISD=0b00000011;`

- Bemenet értékének lekérdezése

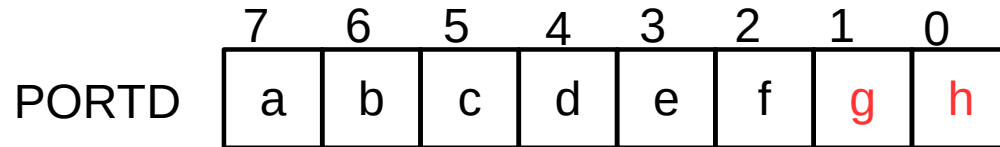
PORTx regiszter megfelelő bitjét kell lekérdezni (most PORTD 0. vagy 1. bitjét)

Lehetőségek:

- bitenkénti ÉS művelettel a szükséges egy bit lekérdezése
- a mikroC fejlesztő környezetben egy regiszter egy bitje önállóan lekérdezhető → `regiszternév.Bbitszám` vagy `regiszternév.Fbitszám`

## 7.3. A kapcsolók érzékelése

### Bemenet értékének lekérdezése bitenkénti ÉS művelettel



PORTD g bit vizsgálható a  
következő művelettel:

```
if(PORTD & 0b00000010)
```

→ igaz, ha g=1 hamis, ha g=0

PORTD h bit vizsgálható a  
következő művelettel:

```
if(PORTD & 0b00000001)
```

→ igaz, ha h=1 hamis, ha h=0

↓

mert:

a b c d e f g h

& 0 0 0 0 0 0 1 0

= 0 0 0 0 0 0 g 0 → 0, ha g=0 vagy 2, ha g=1,  
de minden 0-tól különböző szám igaz logikai  
értékké konvertálódik

## 7.4. A kapcsolók érzékelése

### Bemenet értékének lekérdezése

- mikroC fejlesztő környezetben egyszerűen:

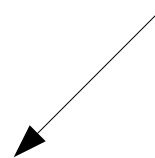
regiszternév.Bbitszám  
vagy  
regiszternév.Fbitszám

	7	6	5	4	3	2	1	0
PORTD	a	b	c	d	e	f	g	h

PORTD g bit lekérdezése:  
PORTD.B1 vagy PORTD.F1



PORTD h bit lekérdezése:  
PORTD.B0 vagy PORTD.F0



if(PORTD.B0==0) → //igaz ha K<sub>0</sub> lenyomva  
vagy if(PORTD.F0==0) → //igaz ha K<sub>0</sub> lenyomva

if(PORTD.B1==1) → //igaz ha K<sub>1</sub> lenyomva  
vagy if(PORTD.F1==1) → //igaz ha K<sub>1</sub> lenyomva

## 7.5. Feladatok

### 1. mintafeladat, a. verzió

- RC2 lábon lévő LED felvillantása 1s-ra, sokszor ( 1s szünetekkel)
- ha K0 kapcsolót lenyomjuk → leáll a villogás, felengedése után folytatódik
- **while(feltétel);** → üres ciklus, amíg a feltétel igaz → addig fut → várakozás egy eseményre → most a kapcsoló felengedésére

// 7.1.a RC2 LED sokszor, K0 kapcsolóval

```
void main( )
{
    TRISC=0b00000000;    // minden RCx láb kimenet
    TRISD=0b00000011;    // RD0 és RD1 láb bemenet
    while (1)             // ismétlés sokszor !
    {
        while(PORTD.B0==0); // amíg K0 lenyomva → fut ez a ciklus
        PORTC=0b00000100;  // RC2 lábra 5V → LED világít
        Delay_ms(1000);    // 1s késleltetés
        PORTC=0b00000000;  // RC2 lábra 0V → LED elalszik
        Delay_ms(1000);    // 1s késleltetés
    }
}
```

## 7.6. Feladatok

### 1. mintafeladat, b. verzió

- RC6 lábon lévő LED felvillantása 1s-ra, sokszor ( 1s szünetekkel)
- ha K1 kapcsolót lenyomjuk → leáll a villogás, felengedése után folytatódik
- **while(feltétel);** → üres ciklus, amíg a feltétel igaz → addig fut → várakozás egy eseményre → most a kapcsoló felengedésére

// 7.1.b RC6 LED sokszor, K1 kapcsolóval

```
void main( )
{
    TRISC=0b00000000;    // minden RCx láb kimenet
    TRISD=0b00000011;    // RD0 és RD1 láb bemenet
    while (1)             // ismétlés sokszor !
    {
        while(PORTD.B1==1); // amíg K1 lenyomva → fut ez a ciklus
        PORTC=0b01000000;  // RC6 lábra 5V → LED világít
        Delay_ms(1000);    // 1s késleltetés
        PORTC=0b00000000;  // RC6 lábra 0V → LED elalszik
        Delay_ms(1000);    // 1s késleltetés
    }
}
```

## 7.7. Feladatok

### 2. mintafeladat (futófény2 kapcsolóval)

- A LED-ek sorban, egymás után világítanak (1s-ig), de kettő egyszerre tehát először RC0-RC1, majd RC1-RC2, .... RC6-RC7, majd előlről  
→ a 3,6,12,24,48,96,192,3,6,.... számokat kell sorban a PORTC-re írni
- ha K<sub>0</sub> kapcsolót lenyomva tartjuk → a haladási irány az ellenkező legyen !  
tehát ilyenkor 192,96,48,24,12,6,3,192,96,.... számokat kell sorban a PORTC-re írni
- K<sub>0</sub> felengedése után újra az eredeti irányban haladjunk



## 7.8. Feladatok

### 2. mintafeladat (futófény2, b. verzió)

```
void main( )           // 7.2. futófény2, K0 kapcsolóval
{
    char szam=3;        // e változó tárolja az aktuális számot
    TRISC=0b00000000;   // minden RCx láb kimenet
    TRISD=0b00000011;   // RD0 és RD1 láb bemenet
    while (1)           // ismétlés sokszor !
    {
        PORTC=szam;     // két lábra 5V → két LED világít
        Delay_ms(1000); // 1s késleltetés
        if(PORTD.B0) {   // ha K0 nincs lenyomva (RD0 láb 1 értékű)
            if(szam<192) szam=2*szam; // *2 → léptetés balra
            else szam=3;           // de ha elértük a 192-őt → kezdés előlről
        }
        if(!PORTD.B0) {  // ha K0 le van nyomva (RD0 láb 0 értékű)
            if(szam>3) szam=szam/2; // /2 → léptetés jobbra
            else szam=192;          // de ha elértük a 3-at → kezdés előlről
        }
    }
}
```

## 7.9. Feladatok

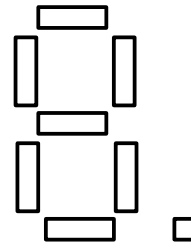
### 3. mintafeladat (futófény2, c. verzió)

```
void main( )           // 7.3. futófény2, K0 kapcsolóval, tömb használatával
{
    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
    TRISD=0b00000011;   // RD0 és RD1 láb bemenet
    while (1)           // ismétlés sokszor !
    {
        PORTC=led2[i ]; // az aktuális szám kiírása PORTC-re
        Delay_ms(1000); // 1s késleltetés
        if(PORTD.B0) {   // ha K0 nincs lenyomva (RD0 láb 1 értékű)
            i++;         // index növelése
            if(i>6) i=0; // ha végig lépkedtünk a tömbön → kezdjük újra
        }
        else {           // egyébként, ha K0 le van nyomva
            if(i>0) i--;  // index csökkentése → ellenkező irány
            else i=6;     // ha végig lépkedtünk a tömbön → kezdjük újra
        }
    }
}
```

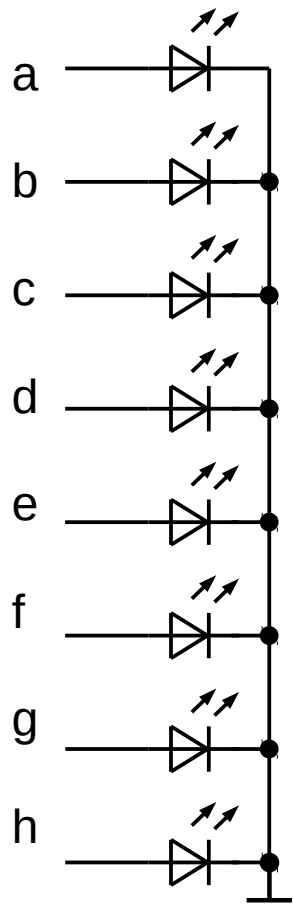
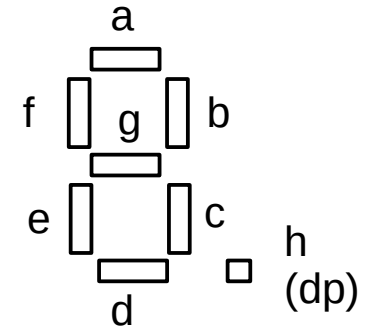
# 8.1. Hétszegmenses kijelző

## Hétszegmenses kijelző

7 + 1 szegmensből áll  
amelyek LEDek igazából  
→ kétféle bekötés !!



szegmens nevek

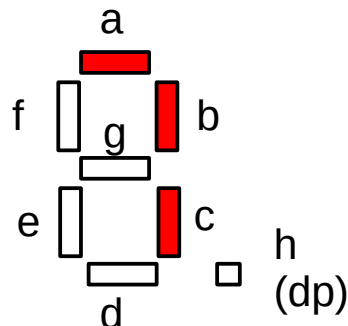


### Közös katódú

vezérlés:

- amelyik szegmensre **1-et adunk** → **világít**
- amelyekre **0-át adunk** → **nem világít**

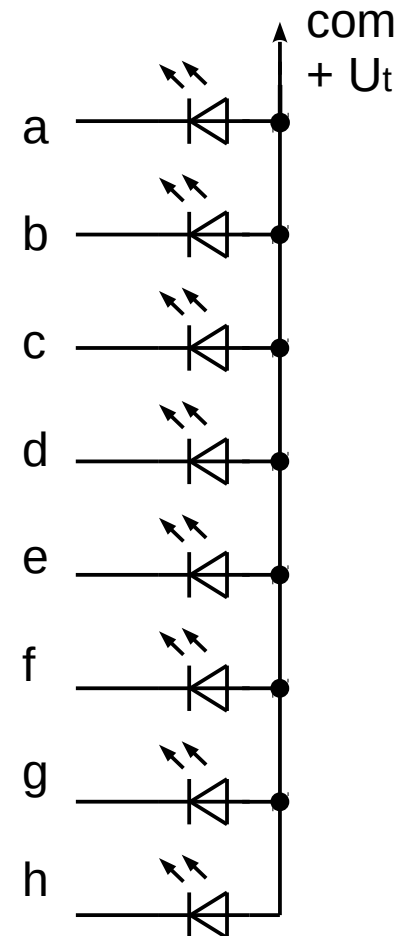
pl. a=b=c=1,  
a többi 0



### Közös anódú

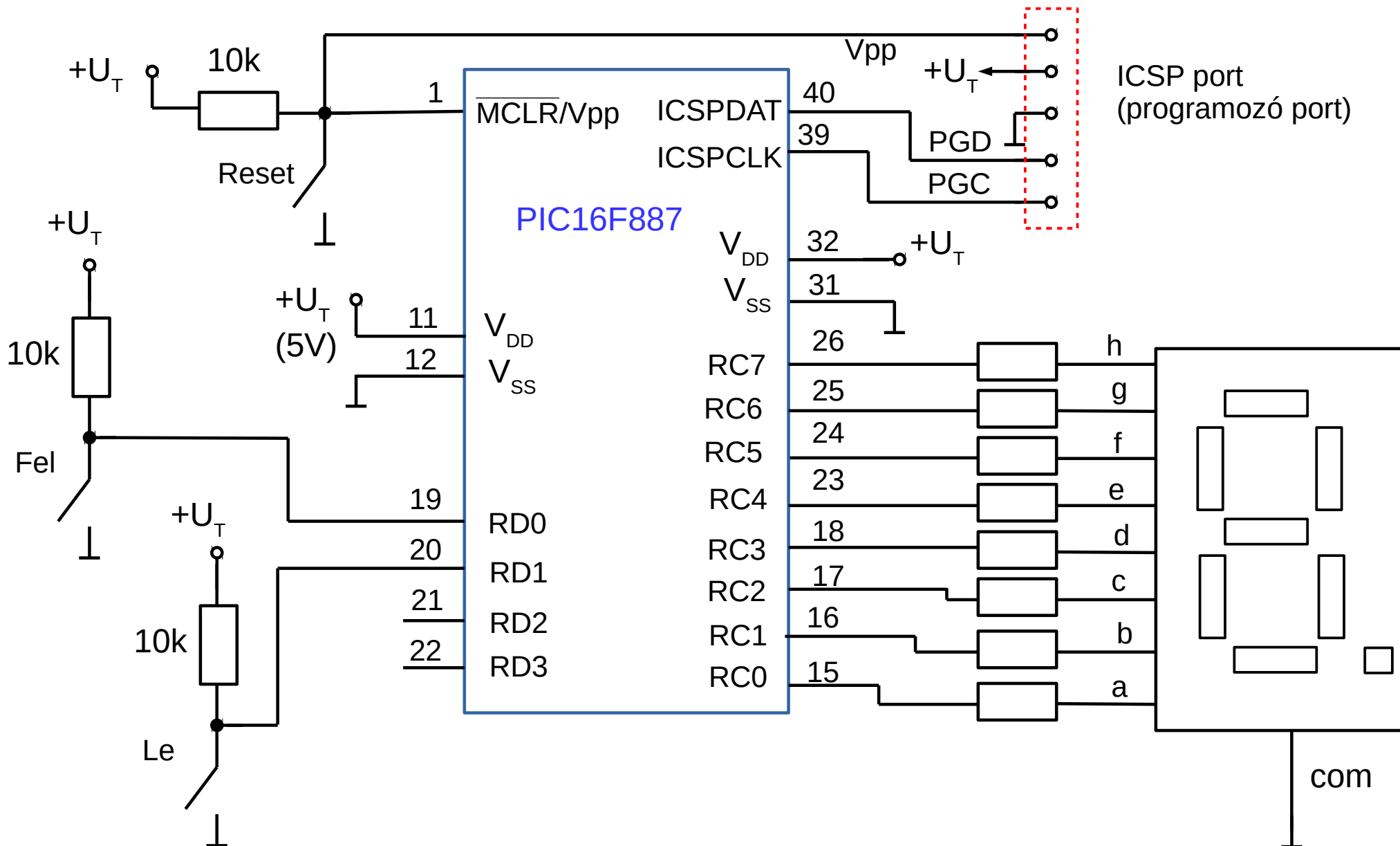
vezérlés:

- amelyik szegmensre **0-át adunk** → **világít**
- amelyekre **1-et adunk** → **nem világít**



## 8.2. A hardver

Hétszegmenses kijelző (közös katódú) a PORTC lábakon



## 8.3. A vezérlő program elkészítése

### 1. mintafeladat

- A 0-2-4-0-2-4-0 ... értékek kijelzése folyamatosan ( 2s-ig mindegyik)
- helyi értékek a bekötés miatt: h-g-f-e-d-c-b-a → 128-64-32-16-8-4-2-1
- 0 → a,b,c,d,e,f szegmensre 1-es → 00111111=0x3F
- 2 → a,b,d,e,g szegmensre 1-es → 01011011=0x5B
- 4 → b,c,f,g szegmensre 1-es → 01100110=0x66

// 8.1. 7 szegmenses kijelző, 0-2-4-0-2-4-0-...

```
void main( )
{
    TRISC=0b00000000;    // minden RCx láb kimenet
    TRISD=0b00000011;    // RD0 és RD1 láb bemenet
    while (1)            // ismétlés sokszor !
    {
        PORTC=0x3F;       // a,b,c,d,e,f szegmens → 0
        Delay_ms(2000);   // 2s késleltetés
        PORTC=0x5B;       // a,b,d,e,g szegmens → 2
        Delay_ms(2000);   // 2s késleltetés
        PORTC=0x66;       // b,c,f,g szegmens → 4
        Delay_ms(2000);   // 2s késleltetés
    }
}
```

## 8.4. Feladatok

Írj programokat 7 szegmenses kijelző (PORTC-n) vezérlésére

- 1. feladat
  - Az 1-3-5-7-9 értékek kijelzése egymás után ( 1s-ig mindegyik)
  - ezt még egyszer megismételni, majd a program leáll
- 2. feladat
  - A 0-2-4-6-8-0-2-4-6-8-0-... értékek kijelzése folyamatosan ( 3s-ig mindegyik)
- 3. feladat
  - Induláskor a 0-9-0-9 értékek kijelzése egymás után ( 1s-ig mindegyik)
  - Majd ezután folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-... (2s-ig mindegyik)
  - ha 'Le' kapcsolót lenyomjuk → leáll a számlálás, felengedése után folytatódik !

## 8.5. Tömb használata

### 2. mintafeladat

- Induláskor a '0' érték kijelzése
  - A FEL kapcsoló nyomva tartása esetén számolás felfelé 1 másodpercenként, 9 elérése esetén nem számol tovább
  - A LE kapcsoló nyomva tartása esetén számolás lefelé 1 másodpercenként, 0 elérése esetén nem számol tovább
  - Hét szegmenses kijelző vezérlése esetén célszerű a kiküldendő kódokat egy tömbben tárolni, mégpedig a következőképpen →
    - '0' számnak megfelelő kód → tömb 0. elemében tárolva
    - '1' számnak megfelelő kód → tömb 1. elemében tárolva
    - ... hasonlóan tovább
    - '9' számnak megfelelő kód → tömb 9. elemében tárolva
- ```
char tomb7[ ]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
```
- Ha valamelyik szám kódját kell a portra küldeni akkor azt most már nem közvetlenül tesszük meg, pl. '2' szám esetén → PORTC=0x5B; !!  
hanem a tömb megfelelő elemét átadva → `PORTC=tomb7[2];`  
hasonlóan pl. a '8'-as szám esetén → `PORTC=tomb7[8];`

## 8.6. Tömb használata

### 2. mintafeladat megoldása

// 8.2. 7 szegmenses kijelző, fel-le

```
void main( )
{
    char tomb7[ ]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
    // a hétszegmenses kijelzőre kiküldendő kódok
    char szam=0;                // tárolja, hogy hol tartunk a számlálásnál
    TRISC=0b00000000;          // minden RCx láb kimenet
    TRISD=0b00000011;          // RD0 és RD1 láb bemenet
    while (1)                   // ismétlés sokszor !
    {
        PORTC=tomb7[szam];      // a 'szam' értékének megfelelő kód küldése
        Delay_ms(1000);         // 1s késleltetés
        if(PORTD.B0==0 && szam<9) { szam++; }
                                // ha FEL lenyomva és szam<9 → szam növelése
        if(PORTD.B1==0 && szam>0) { szam--; }
                                // ha LE lenyomva és szam>0 → szam csökkentése
    }
}
```



## 8.7. Tömb használata

### 3. mintafeladat, 1. megoldása tömbbel

// 8.3. 7 szegmenses kijelző, 0-2-4-0-2-4-0-...

```
void main( )
```

```
{
```

```
    char tomb7[ ]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
```

```
    // a hétszegmenses kijelzőre kiküldendő kódok
```

```
    TRISC=0b00000000;    // minden RCx láb kimenet
```

```
    TRISD=0b00000011;    // RD0 és RD1 láb bemenet
```

```
    while (1)            // ismétlés sokszor !
```

```
    {
```

```
        PORTC=tomb7[0];    // '0' kódjának kiküldése
```

```
        Delay_ms(2000);    // 2s késleltetés
```

```
        PORTC=tomb7[2];    // '2' kódjának kiküldése
```

```
        Delay_ms(2000);    // 2s késleltetés
```

```
        PORTC=tomb7[4];    // '4' kódjának kiküldése
```

```
        Delay_ms(2000);    // 2s késleltetés
```

```
    }
```

```
}
```

## 8.8. Több hétszegmenses kijelző vezérlése

A hardver szempontjából többféleképpen megoldható több hétszegmenses kijelző, de mindenféleképpen bonyolódni fog a dolog

### 1. eset

- Teljesen függetlenül kezeljük a 7szegm. kijelzőket → ezzel egy nagy probléma van → minden kijelző 8 (vagy 7, ha a pontot nem használjuk) digitális kimenetet használ el !! → nem sokat lehetséges így rákötni a mikrovezérlőre  
Persze hogy mennyit, az függ a mikrovezérlő típusától (mennyi digitális kimenete van) és attól hogy milyen egyéb digitális bemeneteket, kimeneteket kell használnunk

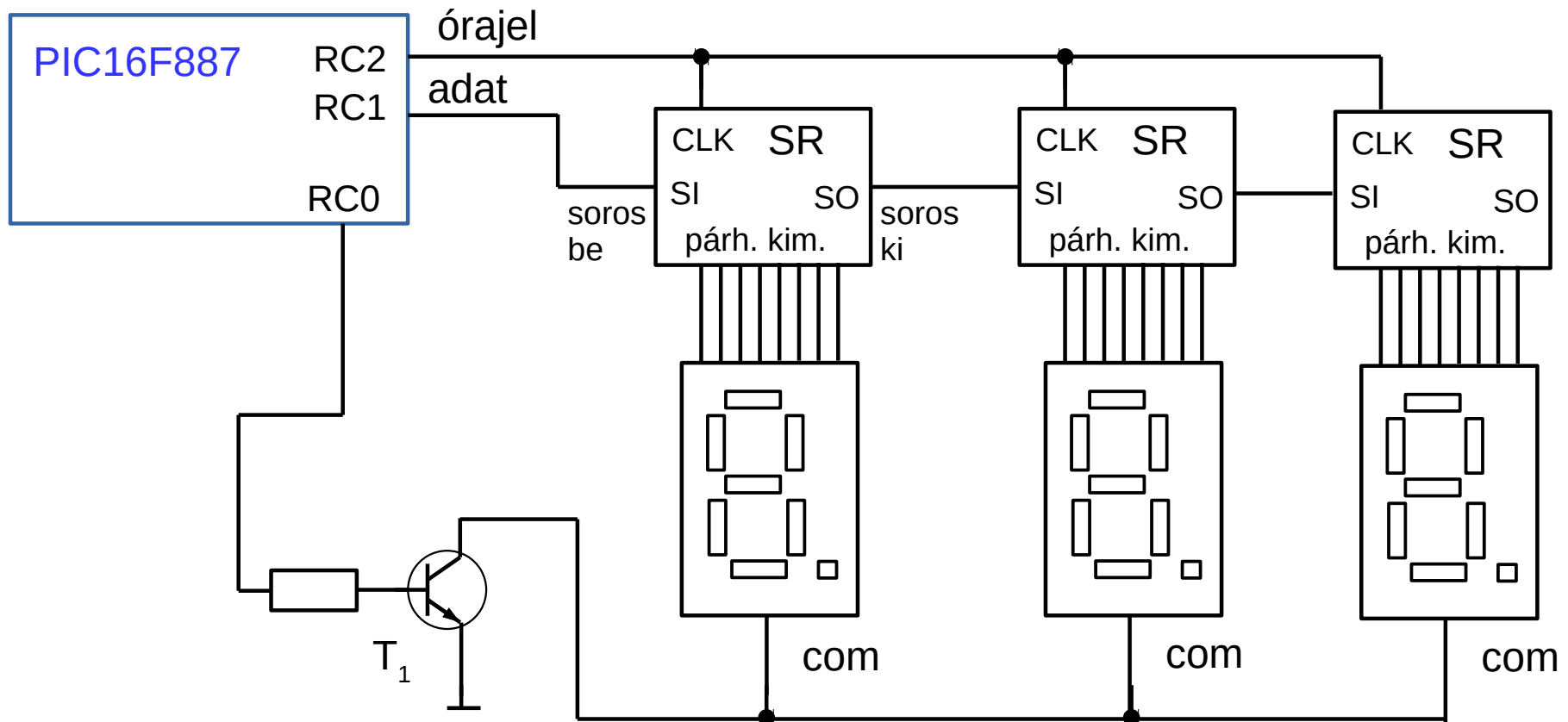
### 2. eset, multiplex vezérlés

- A 7szegmenses kijelzőket párhuzamosan kötjük rá a 8 (vagy 7) digitális kimenetre
- természetesen így mindig ugyanazt az értéket látnánk egyszerre mindegyiken  
→ meg kell oldani, hogy egyenként tudjuk őket bekapcsolni, kikapcsolni
- a közös (COM) bemenetükre nem kapnak közvetlenül 0 (vagy Ut) értéket, hanem tranzisztorokkal kapcsoljuk ezeket
- a tranzisztorokat a mikrovezérlővel kapcsolgatjuk → ez kijelzőként egy plusz digitális kimenetet igényel !
- ha elég gyorsan kapcsolgatjuk a kijelzőket → úgy tűnik mintha egyszerre világítanak

## 8.9. Több hétszegmenses kijelző vezérlése

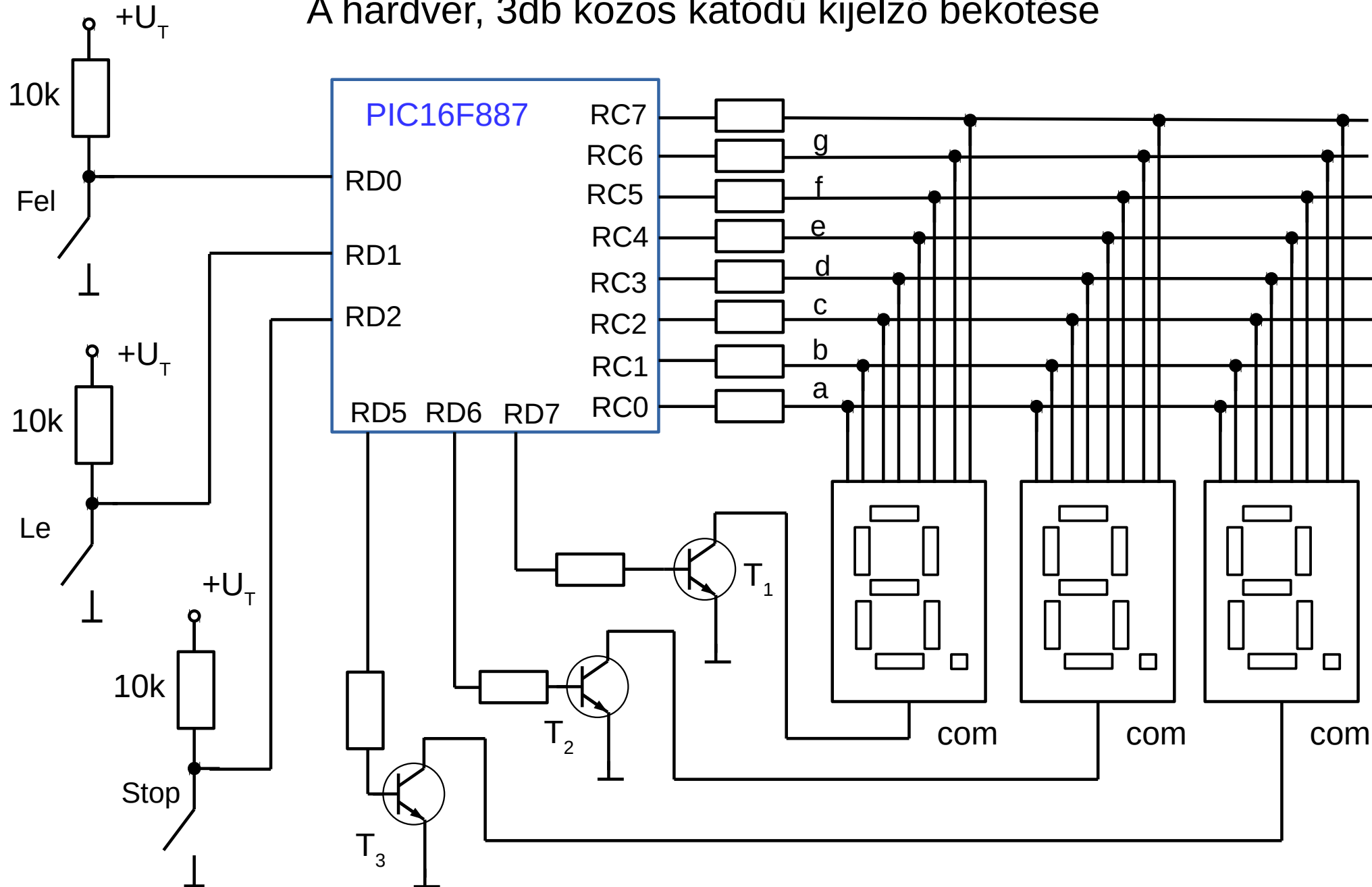
### 3. eset, soros vezérlés shift regiszterekkel

- A 7szegm. kijelzőket közvetlenül shift regiszterekre kötjük
- a shift regisztereket pedig sorban felfűzzük és sorosan küldjük ki nekik a biteket a mikrovezérlőről
- nagyon kevés digitális kimenettel megoldható (2, vagy 3) sok kijelző vezérlése



## 8.10. Hétszegmenses kijelzők multiplex vezérlése

## A hardver, 3db közös katódú kijelző bekötése



## 8.11. Hétszegmenses kijelzők multiplex vezérlése

### 4. mintafeladat

- A 0-2-4-0-2-4-0 ... értékek kijelzése folyamatosan ( 2s-ig mindegyik)
  - **de most mindegyik szám másik kijelzőn !!**
    - '0' szám → 1. kijelzőre      '2' szám → 2. kijelzőre
    - '4' szám → 3. kijelzőre
  - 0 → a,b,c,d,e,f szegm. → PORTC-re 0x3F és  
T1 tranzisztor bekapcsolása, T2 és T3 kikapcsolása →  
PORTD-re 0b10000000
  - 2 → a,b,d,e,g szegm. → PORTC-re 0x5B és  
T2 tranzisztor bekapcsolása, T1 és T3 kikapcsolása →  
PORTD-re 0b01000000
  - 4 → b,c,f,g szegm. → PORTC-re 0x66 és  
T3 tranzisztor bekapcsolása, T1 és T2 kikapcsolása →  
PORTD-re 0b00100000
- A tranzisztorok bekapcsolása → bázisra 1-es szint (+5V)  
A tranzisztorok kikapcsolása → bázisra 0-ás szint (0V)

## 8.12. Hétszegmenses kijelzők multiplex vezérlése

### 4. mintafeladat megoldása

// 8.4. multiplex 7 szegmenses kijelző, 0-2-4-0-2-4-0-...

```
void main( )
{
    TRISC=0b00000000;    // minden RCx láb kimenet → szegmensek
    TRISD=0b00000111;    // RD0, RD1, RD2 lábakon kapcsolók → bemenetek
                          // RD7, RD6 és RD5 lábakon tranzistorok
                          // ismétlés sokszor !

    while (1)
    {
        PORTD=0b00000000;    // mindegyik tranzisztor kikapcsolása
        PORTC=0x3F;          // a,b,c,d,e,f szegmens → 0
        PORTD=0b10000000;    // T1 tranz. bekapcs. → 1. kijelző világít
        Delay_ms(2000);      // 2s késleltetés
        PORTD=0b00000000;    // mindegyik tranzisztor kikapcsolása
        PORTC=0x5B;          // a,b,d,e,g szegmens → 2
        PORTD=0b01000000;    // T2 tranz. bekapcs. → 2. kijelző világít
        Delay_ms(2000);      // 2s késleltetés
        PORTD=0b00000000;    // mindegyik tranzisztor kikapcsolása
        PORTC=0x66;          // b,c,f,g szegmens → 4
        PORTD=0b00100000;    // T3 tranz. bekapcs. → 3. kijelző világít
        Delay_ms(2000);      // 2s késleltetés
    }
}
```

## 8.13. Hétszegmenses kijelzők multiplex vezérlése

### 4. mintafeladat megoldása tömb használatával

// 8.4.b multiplex 7 szegmenses kijelző, 0-2-4-0-2-4-0-...

```
void main( )
{
    char tomb7[ ]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
    TRISC=0b00000000;    // minden RCx láb kimenet → szegmensek
    TRISD=0b00000111;    // RD0, RD1, RD2 lábakon kapcsolók → bemenetek
                        // RD7, RD6 és RD5 lábakon tranzisztorok
                        // ismétlés sokszor !

    while (1)
    {
        PORTD=0b00000000;    // mindegyik tranzisztor kikapcsolása
        PORTC=tomb7[0];      // '0' kódjának kiküldése
        PORTD=0b10000000;    // T1 tranz. bekapcs. → 1. kijelző világít
        Delay_ms(2000);      // 2s késleltetés
        PORTD=0b00000000;    // mindegyik tranzisztor kikapcsolása
        PORTC=tomb7[2];      // '2' kódjának kiküldése
        PORTD=0b01000000;    // T2 tranz. bekapcs. → 2. kijelző világít
        Delay_ms(2000);      // 2s késleltetés
        PORTD=0b00000000;    // mindegyik tranzisztor kikapcsolása
        PORTC=tomb7[4];      // '4' kódjának kiküldése
        PORTD=0b00100000;    // T3 tranz. bekapcs. → 3. kijelző világít
        Delay_ms(2000);      // 2s késleltetés
    }
}
```

## 8.14. Hétszegmenses kijelzők multiplex vezérlése

5. mintafeladat, mutassák a kijelzők folyamatosan a 024 számot

// 8.5. multiplex 7 szegm. 024 a késleltetés kicsi legyen !!

```
void main( )
{
    TRISC=0b00000000;    // minden RCx láb kimenet → szegmensek
    TRISD=0b00000111;    // RD0, RD1, RD2 lábakon kapcsolók → bemenetek
                          // RD7, RD6 és RD5 lábakon tranzistorok
                          // ismétlés sokszor !

    while (1)
    {
        PORTD=0b00000000;    // mindegyik tranzistor kikapcsolása
        PORTC=0x3F;          // a,b,c,d,e,f szegmens → 0
        PORTD=0b10000000;    // T1 tranz. bekapcs. → 1. kijelző világít
        Delay_ms(5);         // késleltetés
        PORTD=0b00000000;    // mindegyik tranzistor kikapcsolása
        PORTC=0x5B;          // a,b,d,e,g szegmens → 2
        PORTD=0b01000000;    // T2 tranz. bekapcs. → 2. kijelző világít
        Delay_ms(5);         // késleltetés
        PORTD=0b00000000;    // mindegyik tranzistor kikapcsolása
        PORTC=0x66;          // b,c,f,g szegmens → 4
        PORTD=0b00100000;    // T3 tranz. bekapcs. → 3. kijelző világít
        Delay_ms(5);         // késleltetés
    }
}
```



## 8.15. Feladatok

Írj programokat az előző 3db multiplexelt 7 szegmenses kijelző vezérlésére

- 1. feladat
  - folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-... DE sorban mindig másik kijelzőn ! → '0' az elsőn, '1' a másodikon, '2' a harmadikon, '3' az elsőn, '4' a másodikon, ....
- 2. feladat
  - folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-... az első kijelzőn ( 1s-ig mindegyik)
  - 'Stop' kapcsolót lenyomva → leáll a számlálás, felengedése után folytatódik !
- 3. feladat
  - folyamatos számlálás 0-1-2-3-4-5-6-7-8-9-0-1-2-... a második kijelzőn ( 1s-ig mindegyik)
  - 'Fel' kapcsolót lenyomva → a számlálás a harmadik kijelzőn történik, felengedése után újra a középsőn folytatódik !
  - 'Le' kapcsolót lenyomva → a számlálás az első kijelzőn történik, felengedése után újra a középsőn folytatódik !

## 8.16. Feladatok

Írj programokat az előző 3db multiplexelt 7 szegmenses kijelző vezérlésére

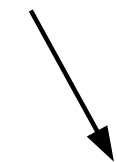
- 4. feladat
    - 'Fel' kapcsolót lenyomva → folyamatos számlálás indul mindhárom kijelzőt használva → 000-001-002-....-010-011-....-998-999-000-001-...
    - 'Stop' kapcsolót lenyomva → leáll a számlálás
    - 'Le' kapcsolót lenyomva → folyamatos számlálás indul lefelé (visszafelé) mindhárom kijelzőt használva
  - 5. feladat
    - 'Fel' kapcsolót lenyomva → egyet lép felfelé a számláló, mindhárom kijelzőt használjuk
    - 'Le' kapcsolót lenyomva → egyet lép lefelé a számláló, mindhárom kijelzőt használjuk
    - induláskor a számláló 000 állásban
- Tipp a megoldáshoz:
- célszerű a MikroC beépített Button() függvényének használata (élfigyelés)
  - vagy saját, élfigyelést alkalmazó kód írása

## 9.1. Függvény

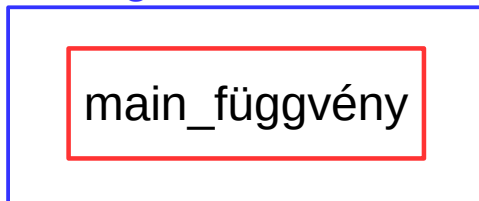
### Egy C nyelvű program szerkezete

- 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 egyszerű program, csak 1 függvény



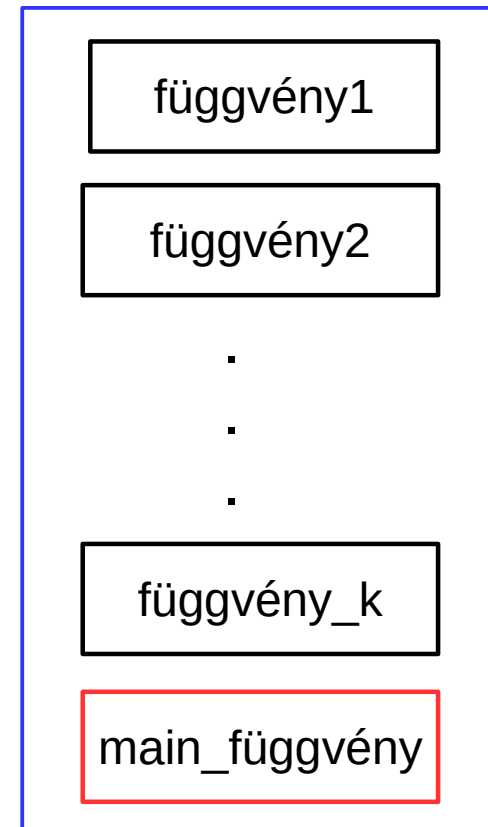
Program



Egy bonyolult program, sok függvény



Program



## 9.2. Függvény

- Függvények
  - vannak előre definiált, beépített függvények:
    - a C nyelv könyvtári függvényei,
    - a használt fejlesztési környezet saját függvényeiezeket csak használni kell, pl. MikroC esetén a Delay\_ms()
  - de természetesen létrehozhatunk saját függvényeket is
  - egy függvény fejrészét (visszatérési érték típusa, függvény neve, és zárójelek között paraméterek) a függvény törzse követi → { } zárójelek között

```
visszatérési_típus függvény_neve(paraméterek) // fejrész
{
    utasítások;                               // a függvény törzse
}
```

- Miért használunk függvényeket?
  - függvények segítségével tudjuk a programunkat kisebb egységekre, alprogramokra osztani
  - így programunk átláthatóbb, egyszerűbb felépítésű lesz →
  - könnyebben módosítható, hibakeresés egyszerűbb (?), könnyebb újra felhasználás

## 9.3. Függvény használata

- Kapcsolat egy függvény és a program többi része között
  - a függvény bemenete → a paraméterek, segítségével tudunk adatokat átadni a függvénynek (amikor meghívjuk)
  - a függvény kimenete → a visszatérési érték, segítségével tudunk adatot visszaadni a hívó programrésznek
  - C nyelven függvényen belül nem lehet függvényt létrehozni !!
  - függvény hívása → **függvénynév(bemenő\_paraméterek);**  
pl. **Delay\_ms(3000);**
  - elképzelhető, hogy nincs paramétere a függvénynek
  - ha van paramétere, akkor ugyanannyi darab (és ugyanolyan típusú) paramétereket kell átadnunk mikor meghívjuk !
  - visszatérési érték felhasználása  
pl. **szam=atlag(10,36,60);**
  - paraméterek definiálása → típus név  
**vissza\_típus függvény\_név(típus1 név1,típus2 név2, ...,típusx névx)**  
  
pl. **int atlag3(int a, int b, int c)**
  - érték visszaadása a **return** utasítással → pl. **return 10**


## 9.4. Függvény használata

### Függvény hívása, érték visszaadása

- pl. 3 szám átlagát kiszámoló függvény (9.1. mintafeladat)

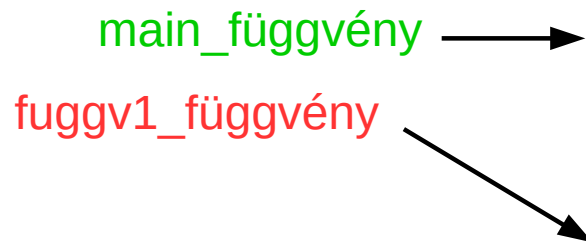
```
float atlag3(float a,float b,float c) // 6. meghívás → a=10, b=20, c=30
{
    float atl;           // 7.
    atl=(a+b+c)/3;       // 8. atl=60/3=20
    return atl;          // 9. vissza a hívóhoz ! → 20 visszaadása
}

void main( )             // itt indul a program !! 1.
{
    float szam1;          // 2.
    float szam2=20;       // 3.
    float szam3;          // 4.
    szam1=atlag3(10,szam2,30); // 5. → atlag3 meghívása → ugrás
                                // 10. → szam1=20
}
```

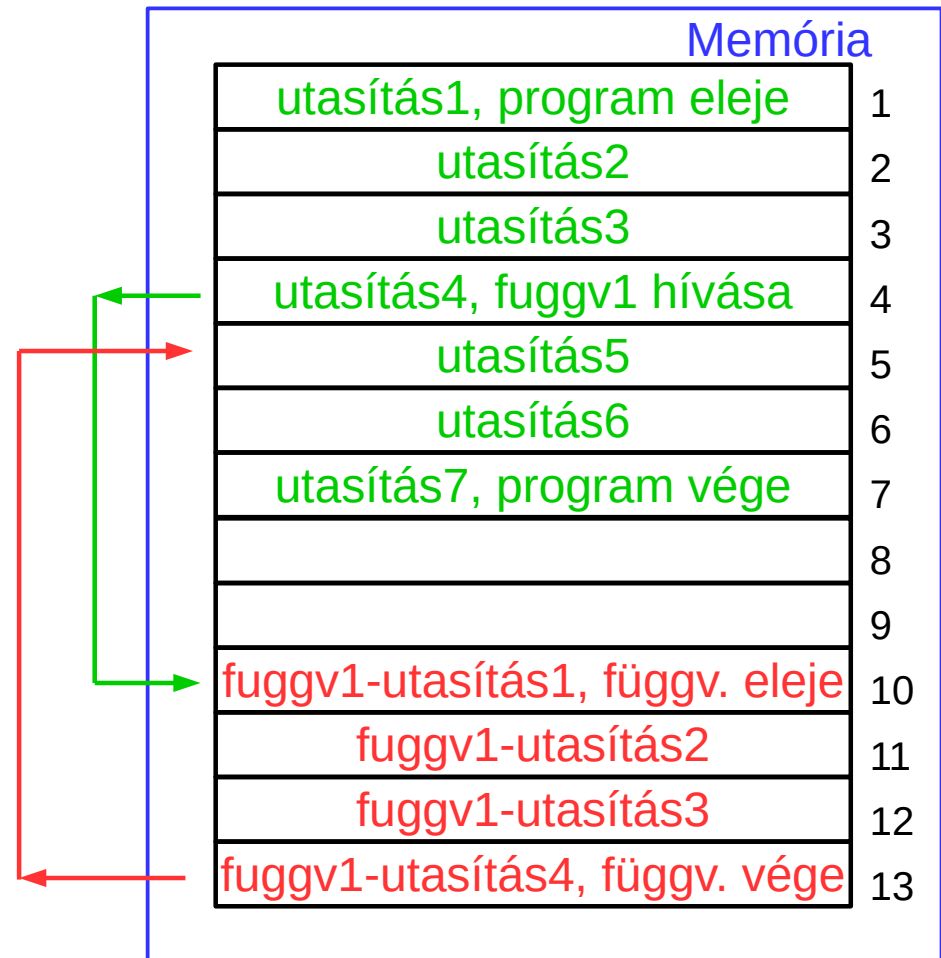


## 9.5. Függvény használata

- Függvény hívása, majd vissza:
  - lényegében **ugró utasítással történik** ! (gépi kód szinten)
  - alapvetően az utasítások végrehajtása ugyanis egymás után történik, ahogyan a memóriában következnek egymás után (olyan sorrendben, ahogyan mi leírtuk az utasításokat)
  - függvény hívás esetén azonban nem a következő memória rekeszben lévő utasítás hajtódik végre, hanem az adott memória címre ugrunk, és ott folytatjuk tovább



A program végrehajtás sorrendje ebben az esetben tehát (melyik számú memóriarekeszben lévő utasítás hajtódik végre)  
→ 1-2-3-4-10-11-12-13-5-6-7



## 9.6. Függvény használata

### Függvény meghívása többször

- sokszor meghívható egy függvény, más-más paraméterekkel
- ez csökkenti a sorok számát (a függvényt csak egyszer kell leírni)

```
float atlag3(float a,float b,float c) // 6. meghívás → a=10, b=20, c=30
{                                     // 12. meghívás → a=20, b=20, c=38
    float atl;           // 7. // 13.
    atl=(a+b+c)/3;       // 8. atl=60/3=20 // 14. atl=78/3=26
    return atl;          // 9. ugrás, 20 visszaadása // 15. ugrás, 26 vissza
}

void main( )                // itt indul a program !! 1.
{
    float szam1;           // 2.
    float szam2=20;        // 3.
    float szam3;           // 4.
    szam1=atlag3(10,szam2,30); // 5. → atlag3 meghívása → ugrás
                                // 10. → szam1=20
    szam3=atlag3(szam2,szam2,38); // 11. → atlag3 meghívása → ugrás
                                    // 16. szam3=26
}
```



## 9.7. Helyi és globális változók

- Helyi változók

- A függvényen belül definiált változók → a függvény helyi (saját) változói → más függvények ezeket nem érik el !!
- **különböző függvényeknek lehetnek azonos nevű helyi változói → de ezen változók teljesen függetlenek egymástól !!**  
(hasonlóan mint egy Miskolcon lévő Petőfi utcának semmi köze egy Debrecenben lévő Petőfi utcához)

```
fuggv1() {  
    char i;  
    char szam1;  
    ..... }
```

```
fuggv2() {  
    char j;  
    char szam1;  
    ..... }
```

```
main() {  
    char szam1;  
    char szam2;  
    ..... }
```

**szam1**, **szam1** és **szam1** változó 3 különböző változó !! → semmi közük egymáshoz

fuggv1 számára 'j' és 'szam2' változó nem létezik

fuggv2 számára 'i' és 'szam2' változó nem létezik

main számára 'j' és 'i' változó nem létezik

## 9.8. Helyi és globális változók

- Helyi változók (9.2. mintafeladat)

// helyi változók

char fuggv1(char a,char b) // meghívás → a=10, b=6 fuggv1 helyi változói

```
{
    char szam1=5;    // fuggv1 'szam1' helyi változója
    char i=1;        // fuggv1 'i' helyi változója
    PORTC=szam1;    // PORTC-re → 5
    i=(a+b)*szam1;  // i=(10+6)*5=80
    a++;            // a=11
    return i;       // vissza a hívóhoz ! → 80 visszaadása
}
```

void main( ) // itt indul a program !!

```
{
    char szam1=10;    // main 'szam1' helyi változója
    char szam2=6;     // main 'szam2' helyi változója
    char a=2;         // main 'a' helyi változója
    TRISC=0b00000000;
    PORTC=szam1;      // PORTC-re → 10
    a=fuggv1(10,szam2); // fuggv1 meghívása → ugrás
                        // a=80
    PORTC=szam1;      // PORTC-re → 10
    PORTC=a;          // PORTC-re → 80
}
```

## 9.9. Helyi és globális változók

- globális változók
  - A függvényeken kívül definiált változók (a program elején) !!
  - minden függvény eléri ezeket a változókat  
(kivéve ha ugyanilyen néven helyi változója is van !!)

```
char i;  
char szam3;
```

```
fuggv1() {  
    char i;  
    char szam1;  
    ..... }  
}
```

```
fuggv2() {  
    char j;  
    char szam1;  
    ..... }  
}
```

```
main() {  
    char szam1;  
    char szam2;  
    ..... }  
}
```

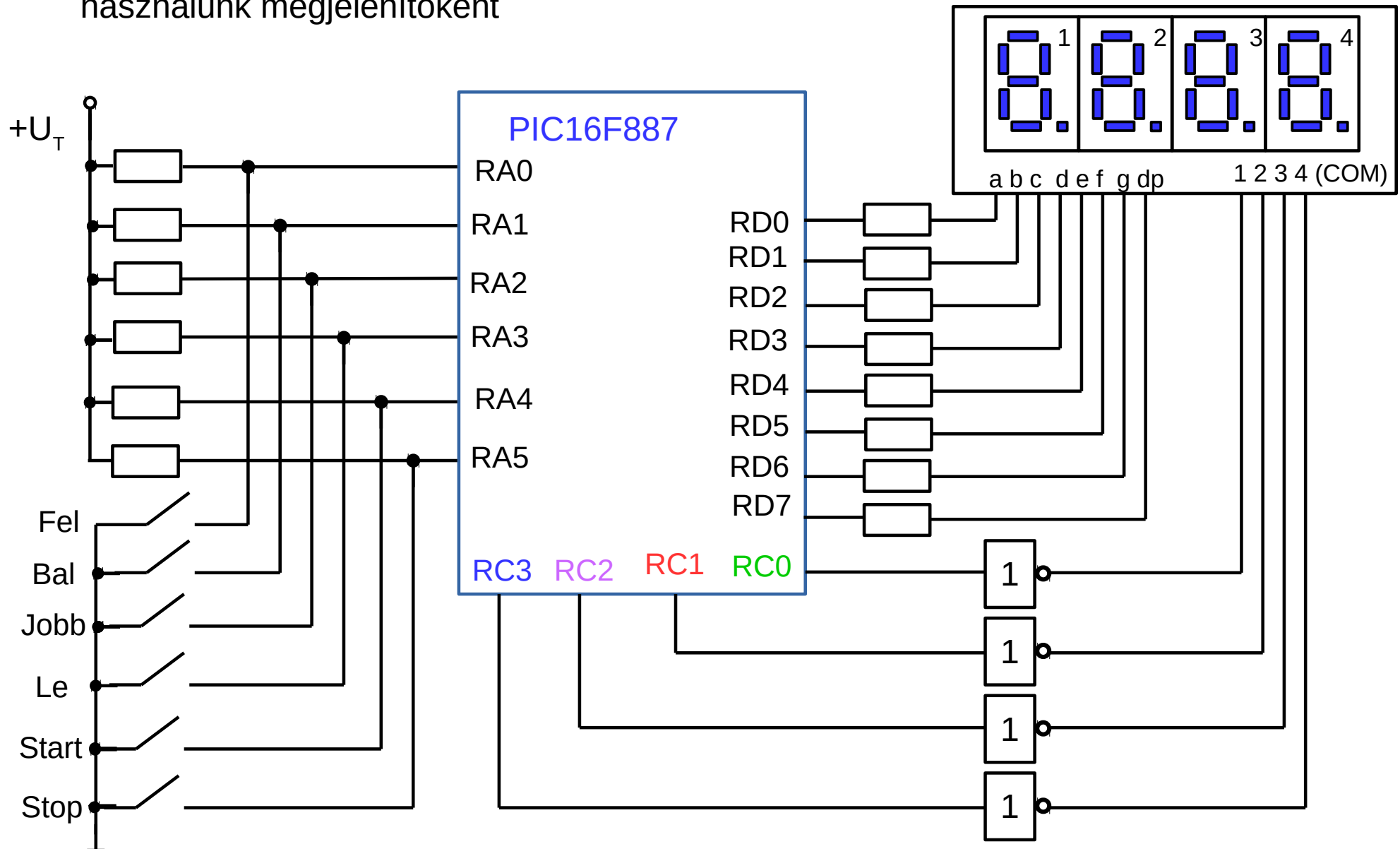
A globális 'i' és 'szam3' változókat minden függvény látja → tudja azokat használni

DE !! függv1 tartalmaz helyi 'i' változót → a globális 'i' helyett azt látja !!

## 9.10. Hétszegmenses kijelzők multiplex vezérlése 2.

### A hardver

- 4 digites, multiplexelt hétszegmenses kijelzőt használunk megjelenítőként



## 9.11. Hétszegmenses kijelzők multiplex vezérlése 2.

### 9.3 minta feladat

- Induláskor a program kiír egy üzenetet, pl. 'Helo', majd vár 'Start' kapcsoló lenyomására
- 'Start' kapcsolót lenyomva → folyamatos számlálás indul (4 számjegy !) mindegyik kijelzőt használva → 0000-0001-0002-....-0010-0011-....-9998-9999-0000-0001-...
- 'Stop' kapcsolót lenyomva → leáll a számlálás
- 'Le' kapcsolót lenyomva → folyamatos számlálás indul lefelé (visszafelé)
- 'Fel' kapcsolót lenyomva → folyamatos számlálás felfelé (előre)

// ez már bonyolultabb program → függvényeket használunk !!  
// globális változók és egy lehetséges függvény egy számjegy kijelzőre írására

```
char tomb7[ ]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,  
               0b01110110,0b01111001,0b00111000,0b00111111}; // hétszegmens kijelző kódok  
               // 0,1,2,3,4,5,6,7,8,9,H,E,L,O  
void szegm7re(char kijelzo, char kod) // hányadik kijelzőre, milyen számjegyet  
{  
    char szam;  
    szam=kod%14; // kod csak 0 és 13 között lehet !  
  
    PORTC=0b00000000; // mindegyik szegmens kikapcsolása  
    PORTD=tomb7[szam]; // szám kódjának kiírása a nyolc szegmens vezetékekre  
    if(kijelzo==1) PORTC=0b00000001; // 1. kijelző világít  
    if(kijelzo==2) PORTC=0b00000010; // 2. kijelző világít  
    if(kijelzo==3) PORTC=0b00000100; // 3. kijelző világít  
    if(kijelzo==4) PORTC=0b00001000; // 4. kijelző világít  
    Delay_ms(4);  
}
```

## 9.12. Hétszegmenses kijelzők multiplex vezérlése 2.

### 9.3 minta feladat

// kezdeti beállítások, üzenet és várakozás START-ra

void kezdes( )

```
{
    OSCCON = 0x70; // oszcillátor beállítása -> külső oszcillátor
    TRISD=0;      // PORTD kimenet → szegmensek (a,b,c,d,e,f,g,dp)
    TRISC=0;      // PORTC → szegmensek vezérlése, RC0 – 1  RC1 – 2  RC2 – 3  RC3 – 4
    TRISA=0b11111111; // PORTA lábak bemenetek (kapcsolók)
    ANSEL=0;      // PORTA lábak digitálisak
    PORTD=0;
    PORTC=0;      // szegmensek lekapcsolva
    // üzenet
    while(PORTA.B4==1) // várunk 'START' lenyomására
    {
        szegm7re(1, 10); // 'H'
        szegm7re(2, 11); // 'E'
        szegm7re(3, 12); // 'L'
        szegm7re(4, 13); // 'O'
    }
}
```

## 9.13. Hétszegmenses kijelzők multiplex vezérlése 2.

### 9.3 minta feladat

```
void main()    // főprogram
{
    char ciklus=0;                // ciklusváltozó
    int szamlalo=0;               // ez tárolja a számot, amit ki kell írni
    int szam;
    char elore=1;                 // állapotváltozó (flag), 1 – előre számlálás 0 – visszafelé
    char leall=0;                 // állapotváltozó (flag), 1 – számlálás leáll 0 – számlálás
    char szamj1,szamj2,szamj3,szamj4;
    kezdes();                     // kezdeti beállításokat elvégző függvény meghívása
    Delay_ms(200);
    while (1)                     // ismétlés végtelenszer !
    {
        szam=szamlalo;
        szamj4=szam%10;           // 4. számjegy
        szam=szam/10; szamj3=szam%10; // 3. számjegy
        szam=szam/10; szamj2=szam%10; // 2. számjegy
        szamj1=szam/10;           // 1. számjegy
        szegm7re(1, szamj1); szegm7re(2, szamj2);
        szegm7re(3, szamj3); szegm7re(4, szamj4);
        if(PORTA.B0==0) { elore=1; leall=0; } // 'FEL' gomb lenyomása
        if(PORTA.B3==0) { elore=0; leall=0; } // 'LE' gomb lenyomása
        if(PORTA.B5==0) { leall=1; } // 'STOP' gomb lenyomása
        ciklus++;
        if(!leall&&(ciklus%20==0)) // ha NEM leall, csak minden 20. ciklusban számlálunk
        { if(elore) szamlalo++;
          else szamlalo--;
        }
    }
}
```

## 9.14. Feladatok

Írj programokat az előző rajzon lévő 4db, multiplexelt 7 szegmenses kijelző vezérlésére

- 1. feladat
  - folyamatos számlálás párosával (csak egy számjegy!): 0-2-4-6-8-0-2-4-... DE sorban mindig másik kijelzőn ! → '0' az elsőn, '2' a másodikon, '4' a harmadikon, '6' a negyediken, '8' az elsőn, ....
  - 'Stop' kapcsolót lenyomva → leáll a számlálás, felengedése után folytatódik !
  - 'Le' kapcsolót lenyomva → a számlálás visszafelé folytatódik
  - 'Fel' kapcsolót lenyomva → a számlálás újból előre történik

- 2. feladat

Módosítsd a 9.3. minta feladatot a következőképpen:

- hármassal számoljon, ne egyesével
- gyorsabb legyen a számlálás, kb. kétszeres sebességű
- kezdéskor, a Start lenyomása után, a végtelen ciklus előtt fusson le egy 'tesztelő' ciklus → minden kijelzőn ugyanaz a szám legyen, és ez növekedjen 0-tól 9-ig (tehát 0000-1111-2222-3333-....)



## 9.15. Megszakítás

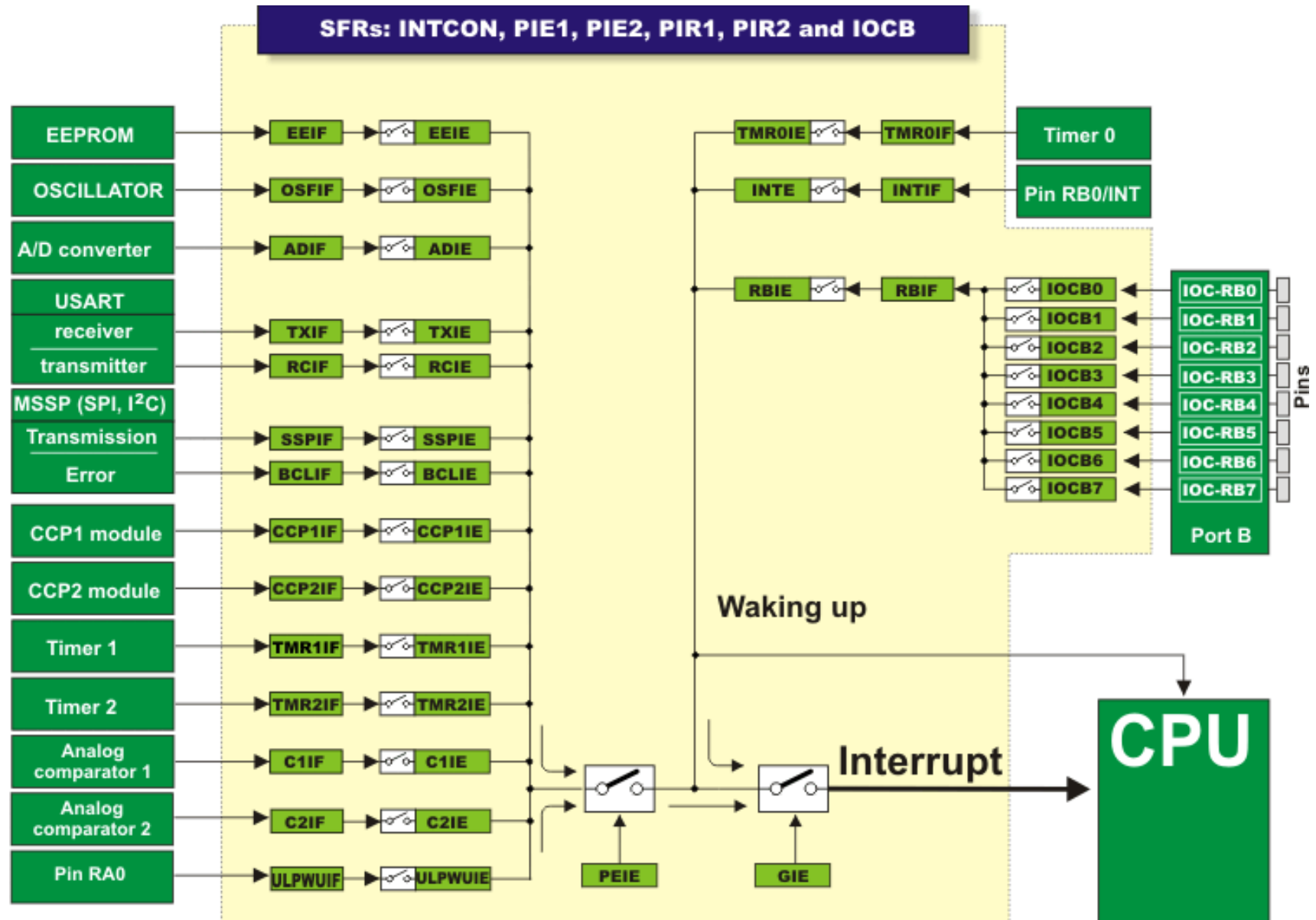
- Megszakítás (interrupt)
  - a program normál végrehajtását egy rövid időre felfüggesztjük, megszakítjuk  
→ hogy valamilyen nagyon fontos, abban a pillanatban bekövetkező (általában külső) esemény hatását lekezeljük, az eseményre reagáljunk
  - az eseményre szükséges válaszlépések elvégzése után folytatódik a normál program végrehajtása (attól a ponttól ahol megszakítottuk)
  - általában valamilyen periféria generálja a mikroprocesszor felé → külső megszakítás bemenetük van a processzoroknak !! (akár több is), de generálható megszakítás szoftveres úton is
- Megszakítás használata
  - a megszakítás lekezelése hasonló mint egy függvény használata → ilyenkor is ugrás történik → egy speciális, megszakításkezelő függvény hívódik meg !!
  - De a különbség az, hogy **megszakításkor automatikusan történik a megszakításkezelő függvény meghívása !!**  
Hiszen a megszakításnál pont az a lényeg, hogy nem tudjuk mikor fog megtörténni !! (ilyen szempontból váratlan) → de nagyon gyorsan reagálnunk kell rá
  - A feladatunk csak az, hogy a megszakításkezelő függvényben lévő utasításokat megadjuk → mi történjen az esemény bekövetkezte esetén
  - A megszakítások általában engedélyezhetők/tilthatók → ez is a mi feladatunk !!

## 9.16. Megszakítás

- Megszakítások mikrovezérlőknél
  - megszakítást tudnak kiváltani a beépített periféria áramkörök
    - időzítő (timer) így jelzi hogy vége az időzítésnek
    - analóg-digitális átalakító (A/D konverter) így jelzi, hogy vége az átalakításnak
  - megszakítást tudunk kiváltani a mikrovezérlő megfelelő lábára adott külső jellel
    - PIC-ek esetén általában az RB0/INT kivezetés szolgálhat megszakítás bemenet céljára is
    - PIC16F887 esetén az összes PORTB láb okozhat megszakítást
  - egy speciális, megszakításkezelő függvény hívódik meg !!
    - MikroC esetén a függvény neve → **interrupt()**
    - más fordítók esetén más név lehet !! pl. Ext-isr()
  - a megszakítások hatására állítódnak a mikrovezérlő különböző regisztereiben lévő megszakítás jelző bitek (IF - interrupt flag) → ezek lekérdezésével meg lehet állapítani, hogy milyen megszakítási esemény történt → ez akkor különösen hasznos ha több megszakítás is engedélyezve van
  - a megszakítások engedélyezéséhez/tiltásához állítani kell a mikrovezérlő különböző regisztereiben lévő megszakítás engedélyező biteket (IE - interrupt enable)
  - PIC16F887 esetén a következő regiszterekkel lehet a megszakításokat engedélyezni, az interrupt flag-eket lekérdezni:
    - INTCON, OPTION\_REG, PIE1, PIE2, PIR1, PIR2, DCON, IOCB

## 9.17. PIC16F887 megszakítás

- Megszakítások PIC16F887 mikrovezérlőknél



Az ábra forrása → PIC Microcontrollers - Programming in C

## 9.18. PIC16F887 megszakítás

Megszakítással kapcsolatos regiszterek PIC16F887 mikrovezérlőknél

|            |       |        |      |      |     |     |     |     |
|------------|-------|--------|------|------|-----|-----|-----|-----|
|            | 7     | 6      | 5    | 4    | 3   | 2   | 1   | 0   |
| OPTION_REG | RBPUP | INTEDG | T0CS | T0SE | PSA | PS2 | PS1 | PS0 |

RBPUP - Port B Pull up Enable bit.  
PORTB felhúzó ellenállások eng.  
1 – tiltás 0 – engedélyezés

INTEDG - Interrupt Edge Select bit.  
Megszakítás él kiválasztás RB0/INT láb  
1 – felfutó él (rising edge)  
0 – lefutó él (falling edge)

|        |     |      |      |      |      |      |      |      |
|--------|-----|------|------|------|------|------|------|------|
|        | 7   | 6    | 5    | 4    | 3    | 2    | 1    | 0    |
| INTCON | GIE | PEIE | T0IE | INTE | RBIE | T0IF | INTF | RBIF |

GIE - Global Interrupt Enable bit → engedélyezi (1) vagy tiltja (0) az összes lehetséges megszakítást

PEIE - Peripheral Interrupt Enable bit → a perifériák megszakítás engedélyezéséhez,  
kivéve timer0 (TMR0) és RB0/INT láb

T0IE - TMR0 Overflow Interrupt Enable bit → timer0 megszakítás engedélyezés (ha 1-re állítjuk)

INTE - RB0/INT External Interrupt Enable bit → RB0/INT láb külső megszakítás engedélyezése (ha 1)

RBIE - RB Port Change Interrupt Enable bit. → PORTB lábak logikai állapotának változása okozta megszakítás

T0IF - TMR0 Overflow Interrupt Flag bit → Timer0 megszakítás jelző bit (1-je jelzi)

INTF - RB0/INT External Interrupt Flag bit → RB0/INT láb külső megszakítás jelző bit

RBIF - RB Port Change Interrupt Flag bit → PORTB lábak állapotának változása okozta megszakítás jelző bit (1 ha legalább 1 láb állapota megváltozott !)

## 9.19. PIC16F887 megszakítás

Megszakítással kapcsolatos regiszterek PIC16F887 mikrovezérlőknél

|      |   |      |      |      |       |        |        |        |
|------|---|------|------|------|-------|--------|--------|--------|
|      | 7 | 6    | 5    | 4    | 3     | 2      | 1      | 0      |
| PIE1 | – | ADIE | RCIE | TXIE | SSPIE | CCP1IE | TMR2IE | TMR1IE |
|      | 7 | 6    | 5    | 4    | 3     | 2      | 1      | 0      |
| PIR1 | – | ADIF | RCIF | TXIF | SSPIF | CCP1IF | TMR2IF | TMR1IF |

**PIE1 → Peripheral interrupt enable bits 1.**

0 – tiltás 1 – engedélyezés

ADIE - A/D Converter Interrupt Enable

RCIE - EUSART Receive Interrupt Enable

TXIE - EUSART Transmit Interrupt Enable

SSPIE - Master Synchronous Serial Port (MSSP) Interrupt Enable

CCP1IE - CCP1 Interrupt Enable

TMR2IE - TMR2 to PR2 Match Interrupt Enable

TMR1IE - TMR1 Overflow Interrupt Enable

**PIR1 → Peripheral interrupt flag bits 1.**

|      |       |      |      |      |       |         |   |        |
|------|-------|------|------|------|-------|---------|---|--------|
|      | 7     | 6    | 5    | 4    | 3     | 2       | 1 | 0      |
| PIE2 | OSFIE | C2IE | C1IE | EEIE | BCLIE | ULPWUIE | – | CCP2IE |

OSFIE - Oscillator Fail Interrupt Enable

C2IE - Comparator C2 Interrupt Enable

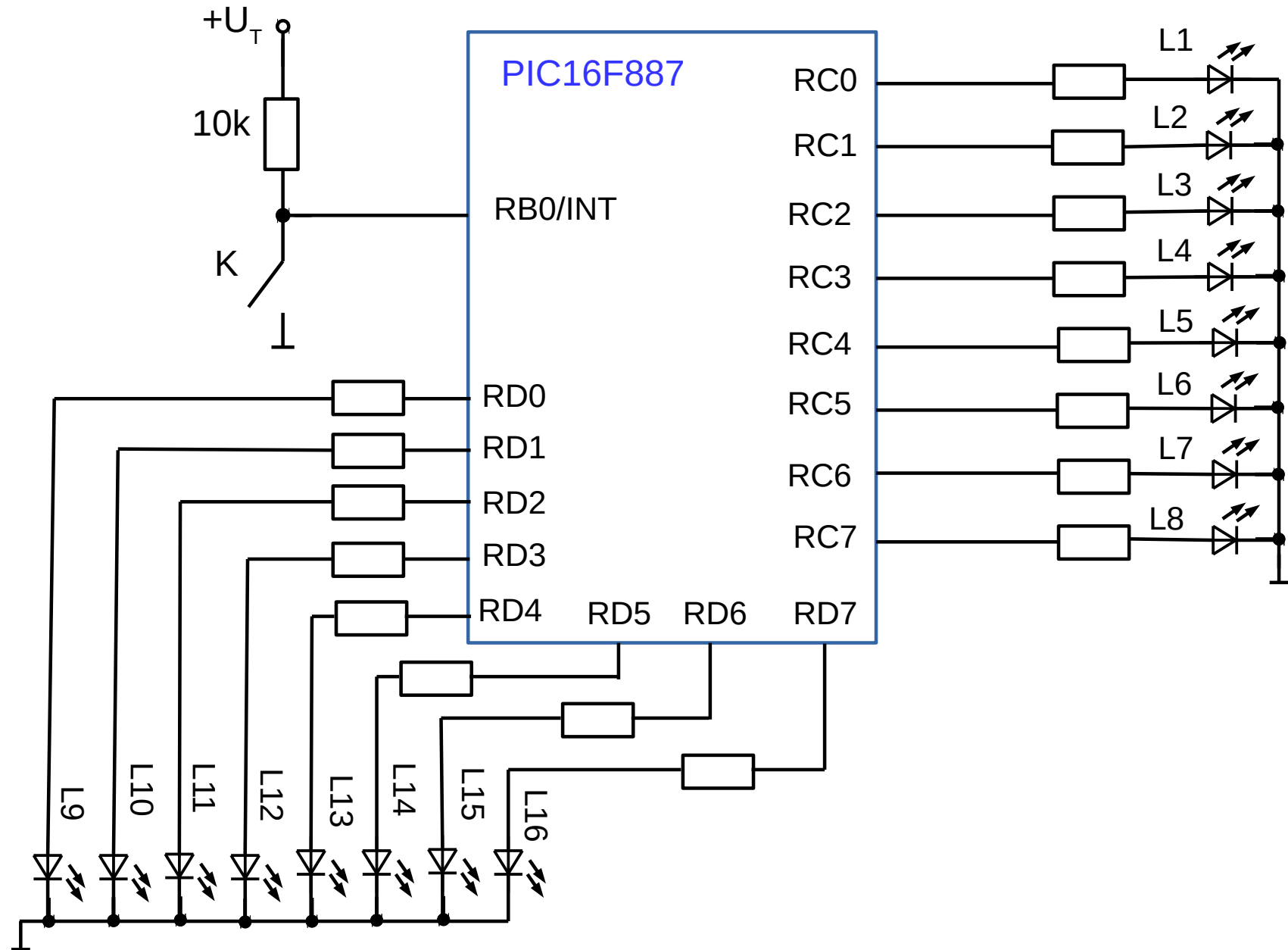
EEIE - EEPROM Write Operation Interrupt Enable

BCLIE - Bus Collision Interrupt Enable

ULPWUIE - Ultra Low-Power Wake-up Interrupt Enable

## 9.20. Megszakítás mintafeladat

A hardver, megszakítás bemenet az RB0 láb (INT)



## 9.21. Megszakítás mintafeladat

### 9.4. mintafeladat

a PORTD-re kapcsolt LED-ek sorban, egymás után világítanak (4s-ig) → futófény → L9-L10-L11-L12-L13-L14-L15-L16-L9...

a K kapcsolót megnyomva pedig megszakítást alkalmazva, számoljuk hányszor nyomtuk meg és binárisan kiírjuk a PORTC-re (LED-ekre)

Induláskor a main függvényben engedélyeznünk kell az INT(RB0) külső megszakítást lefutó élre ! ( K kapcsolót zárva → RB0 lábra 1 helyett 0 kerül)

```
OPTION_REG.B6= 0;    // INTEDG=0 ---> INT/RB0 megszakítás lefutó élre  
INTCON=0x90;         // GIE=INTE=1 INT/RB0 interrupt engedélyezve
```

A **PORTA, PORTB, PORTE** lábak használatához digitális bemenetként nem elég a TRISA, TRISB, TRISE regisztereket megfelelően beállítani !! → ugyanis ezen lábak többsége analóg bemenet is lehet → az **ANSEL és ANSELH** regiszterekkel lehet beállítani, hogy melyik láb legyen

- digitális bemenet (**megfelelő bit 0**)

- vagy analóg bemenet (megfelelő bit 1) → ez az alapértelmezett !!

Tehát most ANSELH regisztert nulláznunk kell (ebben szerepelnek a PORTB lábak)

K lenyomásának számolására egy globális változót (hányszor) használunk, amelyet a megszakítás kezelő függvényben (interrupt) növelünk mindig, és kiíratjuk azonnal a PORTC-re

Az interrupt függvény végén a megszakítást jelző flag-et törölni kell és a megszakítást újra kell engedélyezni !!

## 9.22. Megszakítás mintafeladat

### 9.4. mintafeladat

// 9.4. megszakítás (RB0/INT)

char hanyszor=0; // globális változó, K kapcsoló lenyomásának számolására

```
void interrupt() { // INT(RB0) megszakítás kezelő függvény
    hanyszor++; // K kapcsoló lenyomásának számolása
    PORTC=hanyszor;
    INTCON=0x90; // INT/RB0 interrupt engedélyezése, interrupt flag törlése
}
```

```
void main( ) {
    char szam=0;
    char tomb[]={1,2,4,8,16,32,64,128}; // futófény értékei (egyszerre csak egy LED ég)
    TRISC=0b00000000; // minden PORTC láb kimenet → LED-ek
    TRISD=0b00000000; // minden PORTD láb kimenet → LED-ek
    ANSELH=0; // PORTB lábak nem analóg, hanem digitális bemenetek
    TRISB=0b00000001; // RB0 láb bemenet
    PORTC=hanyszor;
    OPTION_REG.B6= 0; // INTEDG=0 ---> INT/RB0 megszakítás lefutó élre
    INTCON=0x90; // GIE=INTE=1 INT/RB0 interrupt engedélyezve
    while (1) // ismétlés sokszor (végtelenszer) !
    {
        PORTD=tomb[szam]; // futófény
        Delay_ms(3000); // késleltetés
        szam++;
        if(szam>7) szam=0;
    }
}
```



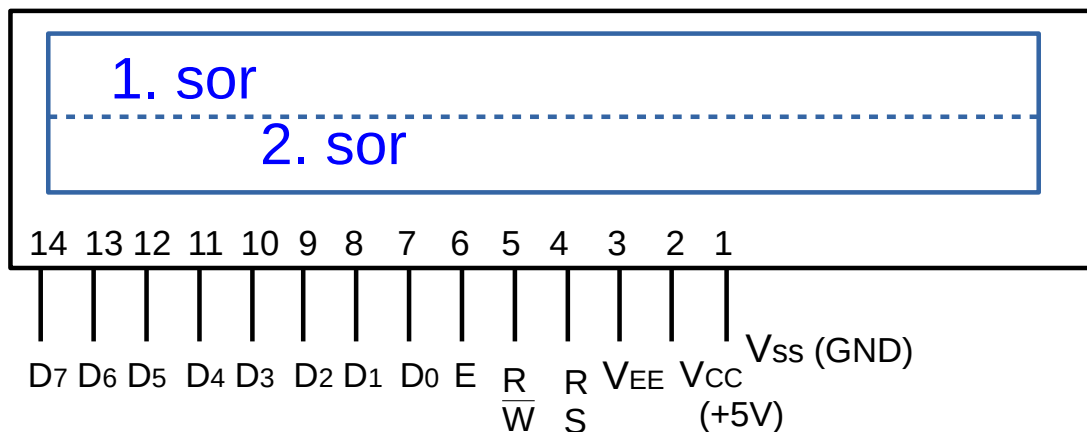
# 10.1. Alfanymerikus LCD kijelző

## Alfanymerikus LCD kijelzők

- LCD (folyadék kristályos) kijelző sokféle létezik, most csak az alfanymerikus kijelzőkkel foglalkozunk, azok közül is a HD44780 (kvázi ipari szabvány) vezérlővel ellátott kijelző modulokkal
  - ebből is létezik többféle: 2 soros, 4 soros, ... → 2x16, 4x20, .. karakteres
  - A 2x16 karakteres kijelző programozását fogjuk áttekinteni
- Erről van egy jó leírás a következő oldalon → <http://esca.atomki.hu/PIC24/lcd.html>

## 2x16 karakteres LCD kijelző

- 16 vagy 14 kivezetéssel rendelkeznek



### adat vezetékek

- 8 bites mód → D0-D7 (vagy DB0-DB7) lábak
- 4 bites mód → D4-D7 (vagy DB4-DB7) lábak

$V_{EE}$  kontraszt beállítása → 0-5V közötti feszültséggel (poti)

### E (vagy EN) - engedélyezés (órjel)

- 6-os láb
- egy ideig (0,3-0,5us) 1-es szintre kell állítani (a pozitív élre olvassa be a bemeneteket)

### RS - Regiszter kiválasztása

- RS=0 → vétel utasítás regiszterbe (IR) → parancs
- RS=1 → vétel az adat regiszterbe (DR) → adat

### R/W – olvasás/írás mód kiválasztása

- R/W=1 → adatok olvasása a kijelzőről
- R/W=0 → adatok küldése a kijelzőnek

## 10.2. HD44780 LCD kijelző

### 1. HD44780 LCD display parancsai

00000001 clear display

0000001x cursor return home

000001MS entry mode set

M-cursor move, 1--> right, 0--> left

S - display shift, 1--> yes, 0--> no

00001DCB display control

D - display, 1--> on, 0--> off

C - cursor, 1--> on, 0--> off

B - cursor blink, 1--> on, 0--> off

0001SDxx cursor/display shift

S-shift-move, 1--> display shift,

0--> cursor move

D-direction, 1--> right, 0--> left

001BRFxx funtion set

B - bit mode, 1--> 8bit, 0--> 4bit

R - row, 1--> 2 rows, 0--> 1 row

F - font, 1--> font 5x10, 0--> font 5x8

01aaaaaa set CGRAM address (user character ---> 5x8)

user character (max. 8):

|                      |               |         |
|----------------------|---------------|---------|
| 1.row B4 B3 B2 B1 B0 | pl. - - x - - | 0b00100 |
| 2.row B4 B3 B2 B1 B0 | - x x x -     | 0b01110 |
| 3.row B4 B3 B2 B1 B0 | x - x - x     | 0b10101 |
| 4.row B4 B3 B2 B1 B0 | - - x - -     | 0b00100 |
| 5.row B4 B3 B2 B1 B0 | - - x - -     | 0b00100 |
| 6.row B4 B3 B2 B1 B0 | - - x - -     | 0b00100 |
| 7.row B4 B3 B2 B1 B0 | - - x - -     | 0b00100 |
| 8.row B4 B3 B2 B1 B0 | - - x - -     | 0b00100 |

1aaaaaaa set DDRAM address **karakter pozíciók címei !!**

első sor 00 01 02 03 ..... 0F

második s. 40 41 42 .... 4F

## 10.3. HD44780 LCD kijelző

### 2. HD44780 LCD display vezérlése

Adatok küldése a kijelzőnek (írás) → R/W lábra 0 szint

A HD44780 vezérlőnek 2db 8 bites regiszterébe írhatunk, az RS lábra adott jellel választjuk ki, hogy melyikbe:

- ha parancsot akarunk küldeni → RS lábra 0 adása → a küldött 8 bit az IR regiszterbe kerül
- ha adatot akarunk küldeni → RS lábra 1 adása → a küldött 8 bit a DR regiszterbe kerül
- RS=0 → vétel utasítás regiszterbe (IR) → parancs
- RS=1 → vétel az adat regiszterbe (DR) → adat

Adatok kérése a kijelzőtől (olvasás) → R/W lábra 1 szint

Ha ezt a funkciót nem akarjuk használni akkor R/W láb fixen 0 szintre (GND) köthető !

- RS=0 esetén → D7 lábon a busy flag értékét, a D6-D0 lábakon a cím számláló értékét (kurzor pozíció) olvashatjuk
- RS= 1 esetén → ?

### 4 bites – 8 bites kommunikáció

- A kijelző modullal kommunikálhatunk 4 (D7-D4) vagy 8 (D7-D0) adat vezetéken → általában a 4 biteset használjuk mert így a mikrovezérlőnél 4 lábbal kevesebb is elég a kijelző vezérléséhez !!
- De mindkét esetben 8 bites számokat küldünk át ! → 4 bites kommunikáció esetén két lépésben visszük át a 8 bitet → először a felső 4 bit, majd az alsó 4 bit

# 10.4. HD44780 LCD kijelző

## 3. HD44780 LCD display beállítási, vezérlési példa

### 4 bites üzemmódban 4 bit küldésének ütemezése

- R/W lábra 0 szint → írás
- RS lábra 0 szint (vagy 1) → parancs küldése (vagy adat)
- D7,D6,D5,D4 lábakra → a parancs vagy adat, felső vagy alsó 4 bitje
- kis késleltetés, ~ 50-100ns !!
- E lábra 1 szint → engedélyezés, kijelző beolvassa a biteket
- kis késleltetés, ~ 300-500ns !!
- E lábra 0 szint → adatfogadás tiltás
- kis késleltetés, ~ 20-50 !!

A nyolc bites parancsokat, adatokat tehát két részletben, hasonló ütemezéssel kell küldenünk

### Üzem mód beállítás (kijelző inicializálása)

Mielőtt bármit is ki tudnánk írni be kell állítani a használni kívánt üzemmódot

- legelőször RS=0 és 0010 kód küldése → megmondjuk, hogy 4 bites üzemmódot használunk  
(ez a parancs felső 4 bitje, az alsó 4 bitjét most nem küldjük el !)
- RS=0 és 00101100 kód küldése → 4 bites üzemmód, 2 sor, karakter méret 5x10
- RS=0 és 00001100 kód küldése → Display ON, Cursor Off
- RS=0 és 00000001 kód küldése → Display törlése
- RS=0 és 00000110 kód küldése → Entry Mode, Increment cursor position, No display shift

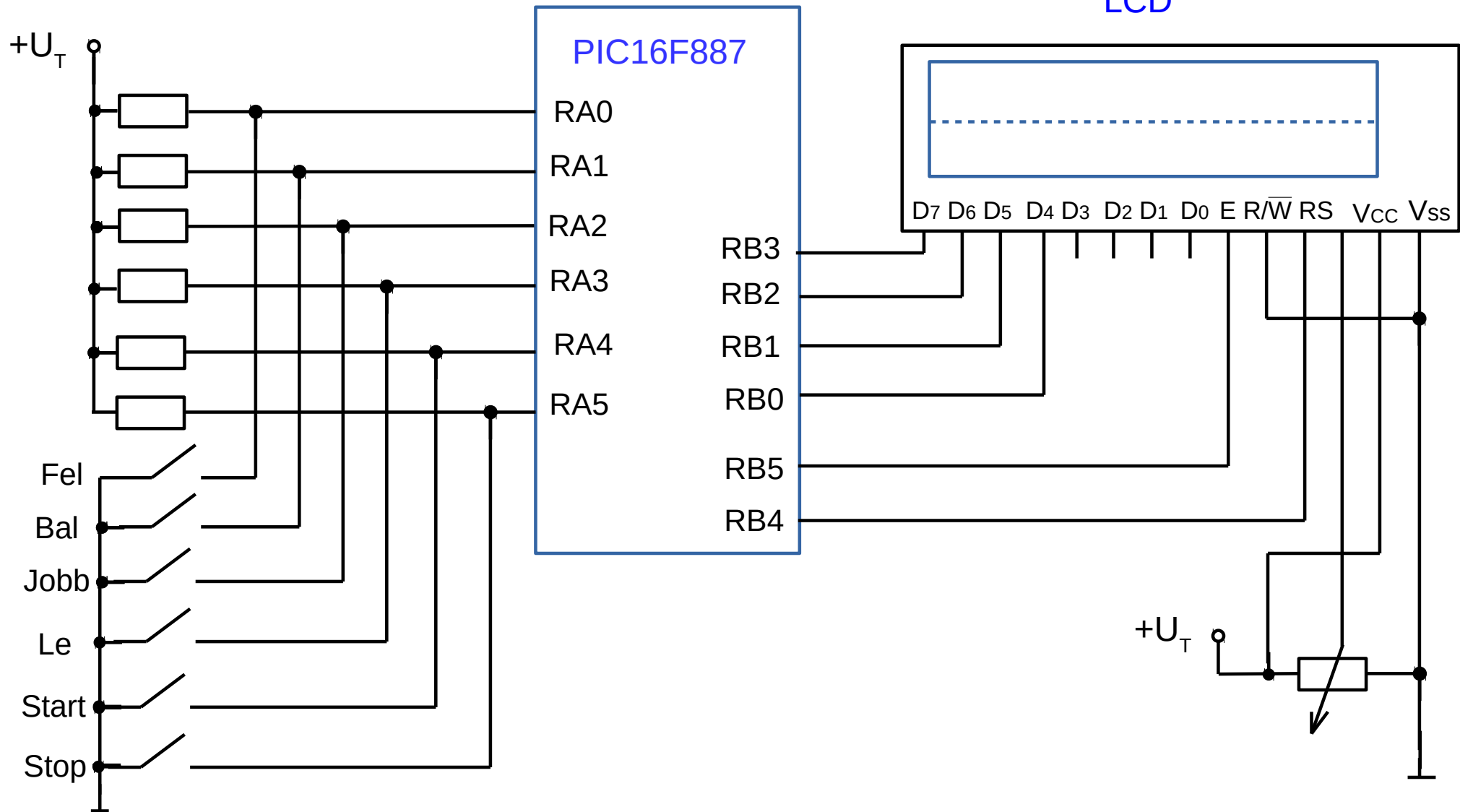
### Karakter kiírása a kijelzőre

Kétféleképpen lehetséges

- először egy parancsként elküldjük a karakter pozíciójának címét (sor, karakter) → 1aaaaaaaa  
majd utána a karakter 8 bites ASCII kódját
- vagy egyből a karakter 8 bites ASCII kódját küldjük → ekkor az aktuális kurzor pozícióba íródik  
pl. RS=0 és 10000000 kód küldése → karakter pozíció beállítása 1. sor 1. karakter  
RS=1 és 00110000 kód küldése → '0' karakter kiírása

## 10.5. HD44780 LCD kijelző vezérlése

A hardver



## 10.6. HD44780 LCD kijelző vezérlése

### Függvények, 4 bit vagy 8 bit küldésére az LCD kijelzőre

A 4 bites információk küldését az LCD-re, az időzítési beállítások kezelését egy függvénnyel oldjuk meg → `send_felbajt(char rs,char adat)` → első paramétere (rs) → az RS láb vezérlését végzi, 0-parancs, 1-adat  
második paramétere (adat) → a küldendő 4 bitet tartalmazza, (az alsó 4 bitje !! mivel ez 8 bites változó)  
Minden egyes 4 bit küldésekor ezt a függvényt kell meghívunk

```
void send_felbajt(char rs,char adat)    // 4 bit (adat változóban) küldése LCD-re
{
    PORTB=PORTB&0b11110000;           // PORTC alsó 4 bitjének törlése
    PORTB=PORTB|(adat&0b00001111);     // a 4 bit beírása PORTC alsó 4 bitjébe
    PORTB.B4=rs;                       //RS láb beállítása (parancs vagy adat?)
    Delay_us(4);
    PORTB.B5=1;                        // órajel (ENG) 1-be
    Delay_us(10);
    PORTB.B5=0;                        // órajel (ENG) 0-ba
    Delay_us(4);
}
```

Függvény, amely a kapott 8 bites számot felbontja két részre (felső és alsó 4 bit)  
és átküldi azokat az LCD kijelzőre

```
void send_bajt(char rs,char szam) // 8 bites szám küldése LCD-re
{
    char felsoresz;
    char alsoresz;
    felsoresz=(0b11110000&szam)>>4;    // szám felső 4 bitje
    alsoresz=0b00001111&szam;          // szám alsó 4 bitje
    send_felbajt(rs,felsoresz);
    send_felbajt(rs,alsoresz);
}
```

## 10.7. HD44780 LCD kijelző vezérlése

### Függvény a kezdeti beállításokra

A kezdeti regiszter beállításokat és LCD üzemmódjának beállítását egy függvénnyel oldjuk meg → `kezd_beall()`

```
void kezd_beall( )
{
    TRISB=0b00000000;    // PORTB lábak kimenetek → LCD D4, D5, D6 ,D7, RS és E
    ANSELH=0;             // PORTB lábak digitálisak (nem analóg bemenetek)
    TRISA=0b11111111;    // PORTA lábak bemenetek
    ANSEL=0;              // PORTA lábak digitálisak (nem analóg bemenetek)
    PORTB=0;
    Delay_ms(100);
    send_felbajt(0,0b0010); //4 bites üzemmód
    Delay_ms(5);
    send_bajt(0,0b00101100); //4 bites üzemmód, 2 sor, karakter méret 5x10
    send_bajt(0,0b00001100); //Display ON, Cursor Off
    send_bajt(0,0b00000001); //Clear Display
    send_bajt(0,0b00000110); //Entry Mode, Increment cursor position, No display shift
}
```

A következő minta feladatokban ezeket a saját függvényeket fogjuk használni → be kell őket másolni mindig a main függvény elé !!

Természetesen használhatunk más függvényeket is → pl. a MikroC fejlesztő környezetben nagyon jó kényelmes LCD kezelő függvények vannak, azokat is lehet használni

## 10.8. HD44780 LCD kijelző vezérlése

Kezdeti beállítások, ha a MikroC függvényeit használjuk

```
sbit LCD_RS at RB4_bit;           // Melyik lábra vannak kötve az LCD kivezetései
sbit LCD_EN at RB5_bit;
sbit LCD_D4 at RB0_bit;
sbit LCD_D5 at RB1_bit;
sbit LCD_D6 at RB2_bit;
sbit LCD_D7 at RB3_bit;
sbit LCD_RS_Direction at TRISB4_bit;
sbit LCD_EN_Direction at TRISB5_bit;
sbit LCD_D4_Direction at TRISB0_bit;
sbit LCD_D5_Direction at TRISB1_bit;
sbit LCD_D6_Direction at TRISB2_bit;
sbit LCD_D7_Direction at TRISB3_bit;

void kezd_beall2( )
{
    TRISB=0b00000000;           // PORTB lábak kimenetek → LCD D4, D5, D6 ,D7, RS és E
    ANSELH=0;                   // PORTB lábak digitálisak (nem analóg bemenetek)
    TRISA=0b11111111;           // PORTA lábak bemenetek
    ANSEL=0;                     // PORTA lábak digitálisak
    PORTB=0;
    Lcd_Init();                  // MikroC LCD kezdeti beállító függvénye
    Delay_ms(100);
    Lcd_Cmd(_LCD_TURN_ON);       //MikroC, parancs küldés LCD-re → Display ON
    Lcd_Cmd(_LCD_CURSOR_OFF);    // Cursor Off
    Lcd_Cmd(0b00101100);         // function set: 4 bites mód, 2 sor, font 5x10
    Lcd_Cmd(_LCD_CLEAR);         //Clear Display
}
```



## 10.9. Minta programok

### 1. minta feladat (saját függvények használatával)

Írjuk ki az LCD kijelző 1. sorába a 'Hello' szöveget, a betűk lassan egymás után jelenjenek meg, majd a 2. sorban menjen folyamatosan egy számláló → 0-1-2-3-4-5-6-7-8-9-0-1-2-...

```
void main( )
{
    char szam=0;
    kezd_beall();
    send_bajt(0,0b10000000); // 1. sor 1. pozíció megadása
    Delay_ms(300);
    send_bajt(1,'H');          // 'H' kiírása
    Delay_ms(2000);
    send_bajt(1,'e');          // 'e' kiírása
    Delay_ms(2000);
    send_bajt(1,'l');          // 'l' kiírása
    Delay_ms(2000);
    send_bajt(1,'l');          // 'l' kiírása
    Delay_ms(2000);
    send_bajt(1,'o');          // 'o' kiírása
    while (1)                  // ismétlés sokszor (végtelenszer) !
    {
        send_bajt(0,0b11000000); // 2. sor 1. pozíció megadása
        send_bajt(1,48+szam);      // szám küldése LCD-re, a '0' ASCII kódja 48 !
        Delay_ms(1000);           // késleltetés
        szam++;
        if(szam>9) szam=0;
    }
}
```

## 10.10. Minta programok

### 1.b minta feladat (MikroC függvényeit használva)

Írjuk ki az LCD kijelző 1. sorába a 'Hello' szöveget, a betűk lassan egymás után jelenjenek meg, majd a 2. sorban menjen folyamatosan egy számláló → 0-1-2-3-4-5-6-7-8-9-0-1-2-...

```
void main( )
{
    char szam=0;
    kezd_beall2();
    Lcd_Chrr(1,1,'H');    // 'H' kiíratása  1. sor 1. pozícióba
    Delay_ms(2000);
    Lcd_Chrr_Cp('e');      // 'e' kiíratása (aktuális kurzor pozícióba)
    Delay_ms(2000);
    Lcd_Chrr_Cp('l');      // 'l' kiíratása
    Delay_ms(2000);
    Lcd_Chrr_Cp('l');      // 'l' kiíratása
    Delay_ms(2000);
    Lcd_Chrr_Cp('o');      // 'o' kiíratása
    while (1)              // ismétlés sokszor (végtelenszer) !
    {
        Lcd_Chrr(2,1,48+szam);    // szám küldése LCD-re, 2. sor 1. pozícióba ( '0' → 48 )
        Delay_ms(1000);           // késleltetés
        szam++;
        if(szam>9) szam=0;
    }
}
```

# 10.11. Minta programok

## 2. minta feladat (saját függvények használatával)

Írjuk ki az LCD kijelző 1. sorába a 'Hi!' szöveget, a betűk lassan egymás után jelenjenek meg, majd a 2. sorban menjen folyamatosan egy számláló → 0-1-2-3-4-5-6-7-8-9-0-1-2-...

'LE' kapcsolót megnyomva a számlálás lefelé folytatódjon, 'FEL' kapcsolót megnyomva újra felfelé !!

```
void main( )
{
    char szam=0;
    char fel=1;           // állapot jelző → 1 – felfelé számlálás 0 – lefelé számlálás
    kezd_beall();
    send_bajt(0,0b10000000); // 1. sor 1. pozíció megadása
    Delay_ms(300);
    send_bajt(1,'H');       // 'H' kiírása
    Delay_ms(2000);
    send_bajt(1,'i');       // 'i' kiírása
    Delay_ms(2000);
    send_bajt(1,'!');       // '!' kiírása
    Delay_ms(1000);
    while (1)              // ismétlés sokszor (végtelenszer) !
    {
        send_bajt(0,0b11000000); // 2. sor 1. pozíció megadása
        send_bajt(1,48+szam);     // szám küldése LCD-re, a '0' ASCII kódja 48 !
        Delay_ms(1000);          // késleltetés
        if(fel==1) { szam++; if(szam>9) szam=0; }
        if(fel==0) { if(szam>0) szam--; else szam=9; }
        if(PORTA.B0==0) { fel=1; } // 'FEL' gomb → számlálás felfelé
        if(PORTA.B3==0) { fel=0; } // 'LE' gomb → számlálás lefelé
    }
}
```

## 10.12. Minta programok

### 3. minta feladat (MikroC függvényeit használva)

Az LCD kijelzőn menjen induláskor egy számláló az 1. sor, 1. karakter pozícióban  
→ 0-1-2-3-4-5-6-7-8-9-0-1-2-...

A 'LE' kapcsolót megnyomva a számlálás a 2.sor 1. pozíciójában folytatódjon,

A 'FEL' kapcsolót megnyomva a számlálás az 1.sor 1. pozíciójában folytatódjon

```
void main( )
{
    char szam=0;           // a számlálást végző változónk
    char fent=1;           // flag, 1-es értéke jelzi hogy a felső sorba kell írunk
    char lent=0;           // flag, 1-es értéke jelzi hogy az alsó sorba kell írunk
    kezd_beall2();
    Delay_ms(500);
    while (1)              // ismétlés sokszor (végtelenszer) !
    {
        if(fent)           // felső sorba kell írunk
        { Lcd_Chr(1,1,48+szam); // szám küldése LCD-re, 1. sor 1. pozícióba
          Lcd_Chr(2,1,' '); } // 2. sor 1. pozíciójában törlés !! (space beírása)
        if(lent)           // alsó sorba kell írunk
        { Lcd_Chr(2,1,48+szam); // szám küldése LCD-re, 2. sor 1. pozícióba
          Lcd_Chr(1,1,' '); } // 1. sor 1. pozíciójában törlés !! (space beírása)
        szam++;
        if(szam>9) szam=0;
        if(PORTA.B0==0) { fent=1; lent=0; } // 'FEL' gomb lenyomása
        if(PORTA.B3==0) { fent=0; lent=1; } // 'LE' gomb lenyomása
        Delay_ms(900);      // késleltetés
    }
}
```

## 10.13. Minta programok

### 4. minta feladat (MikroC függvényeit használva)

Az LCD kijelzőn menjen induláskor egy számláló az 1. sor, 1. karakter pozícióban

→ 0-1-2-3-4-5-6-7-8-9-0-1-2-...

A 'JOB' kapcsolót megnyomva a számlálás az 1.sor 14. pozíciójában folytatódjon,

A 'BAL' kapcsolót megnyomva a számlálás az 1.sor 1. pozíciójában folytatódjon

```
void main( )
{
    char szam=0;           // a számlálást végző változónk
    char balra=1;          // flag, 1-es értéke jelzi hogy az 1. pozícióba kell írunk
    char jobbra=0;         // flag, 1-es értéke jelzi hogy a 14. pozícióba kell írunk
    kezd_beall2();
    Delay_ms(500);
    while (1)              // ismétlés végtelenszer
    {
        if(balra)          // felső sorba kell írunk
        { Lcd_Chr(1,1,48+szam); // szám küldése LCD-re, 1. sor 1. pozícióba
          Lcd_Chr(1,14,' '); } // 1. sor 14. pozíciójában törlés !! (space beírása)
        if(jobbra)         // alsó sorba kell írunk
        { Lcd_Chr(1,14,48+szam); // szám küldése LCD-re, 1. sor 14. pozícióba
          Lcd_Chr(1,1,' '); } // 1. sor 1. pozíciójában törlés !! (space beírása)
        szam++;
        if(szam>9) szam=0;
        if(PORTA.B1==0) { balra=1; jobbra=0; } // 'BAL' gomb lenyomása
        if(PORTA.B2==0) { balra=0; jobbra=1; } // 'JOB' gomb lenyomása
        Delay_ms(900); // késleltetés
    }
}
```

# 10.14. Feladatok

## 1. feladat

Az LCD kijelzőn menjen induláskor egy számláló az 1. sor, 1. karakter pozícióban

→ 0-1-2-3-4-5-6-7-8-9-0-1-2-...

- a 'JOB' kapcsolót megnyomva a számlálás a 14. karakter pozíciójában folytatódjon,
- a 'BAL' kapcsolót megnyomva a számlálás az 1. karakter pozíciójában folytatódjon
- a 'LE' kapcsolót megnyomva a számlálás a 2. sorban folytatódjon,
- a 'FEL' kapcsolót megnyomva a számlálás az 1. sorban folytatódjon  
(a 11.3 és 11.4 minta feladatok összekombinálása)

## 2. feladat

Induláskor →

- az 1. sorban írunk ki egy üzenetet (pl. 'Helo'), majd a 2. sor 1. karakter pozícióban egy számláló számoljon folyamatosan → 0-1-2-3-4-5-6-7-8-9-0-1-2-....  
(gyorsan, kb. fél másodpercenként), amíg a 'START' kapcsolót nem nyomjuk le

A 'START' kapcsoló megnyomása után →

- a számlálás a 2. sor, 10. karakter pozícióban folytatódjon !!
- a 'LE' kapcsolót megnyomva a számlálás visszafelé (lefelé) történjen !!
- a 'FEL' kapcsolót megnyomva a számlálás újból előre (felfelé) történjen

## 3. feladat

- induláskor → 1. sorban írunk ki egy üzenetet (pl. 'Helo'), majd a 2. sorban villogjon egy másik üzenet → „ JOB gomb → jobbra BAL gomb → balra !” ( kb. 2-3 másodpercenként), amíg a 'STOP' kapcsolót nem nyomjuk le
- a 'STOP' kapcsoló megnyomása után a 2. sorban kiírva → „stop”
- a 'JOB' kapcsoló megnyomása után a 2. sorban kiírva → „jobbra” → a kiírás lépjen egyet jobbra 1 másodpercenként
- a 'BAL' kapcsoló megnyomása után a 2. sorban kiírva → „balra” → a kiírás lépjen egyet balra 1 másodpercenként

# 10.15. Feladatok

## 4. feladat

Az LCD kijelzőn menjen induláskor egy számláló az 1. sor, 1-2. karakter pozícióban  
→ 0-1-2-3-4-5-6-7-8-9-10-11-12-.....- 97-98-99-0-1-2-..... (két számjegyű !!)

- a 'LE' kapcsolót megnyomva a számlálás visszafelé folytatódjon !!
- a 'FEL' kapcsolót megnyomva a számlálás újból előre folytatódjon

## 5. feladat

Az LCD kijelzőn induláskor írjuk ki az angol ABC-t az 1. sor, 1. karakter pozícióban  
→ A-B-C-D-E-F-G-...-Z-A-B-...

- a 'JOBB' kapcsolót megnyomva a kiírás egy pozícióval jobbra folytatódjon,
- a 'BAL' kapcsolót megnyomva a kiírás egy pozícióval balra folytatódjon
- a 'LE' kapcsolót megnyomva a kiírás a 2. sorban folytatódjon,
- a 'FEL' kapcsolót megnyomva a kiírás az 1.sorban folytatódjon

## 6. feladat

- induláskor → 1. sorban írjunk ki egy üzenetet (pl. 'Helo'), majd a 2. sorban villogjon egy másik üzenet → „Nyomd meg a START gombot !” ( kb. 2-3 másodpercenként), amíg a 'START' kapcsolót nem nyomjuk le
- a 'START' kapcsolót megnyomva a számlálás két számjegyű legyen !! → a 2. sor, 1-2. karakter pozícióban → 0-1-2-3-4-5-6-7-8-9-10-11-12-.....- 97-98-99-0-1-2-.....
- a 'LE' kapcsolót megnyomva a számlálás visszafelé (lefelé) történjen !!
- a 'FEL' kapcsolót megnyomva a számlálás újból előre (felfelé) történjen