

PIC programozása 3. (PIC16F887)

11. Léptetőmotor vezérlése,élfigyelés
12. PIC, analóg bemenetek
13. PIC, időzítő (timer)
14. PIC, konfigurációs bitek, oszcillátor beállítások

11.1. Léptető motorok

1. Léptető motorok

- szénkefe nélküli egyenáramú motorok, amelyek adott számú lépésből tesznek meg egy fordulatot → általában néhány fokos szögelfordulás jellemző lépésenként (pl. $1,8^\circ$ → 200 lépés egy körülforgás)
- vezérlésük digitális → a tekercsekre megfelelő sorrendben kell feszültséget kapcsolni (majd lekapcsolni)
- elég pontosan egy adott pozícióba lehet forgatni ! (1-5%)
- fordulatszámuk viszont elég alacsony lehet, néhány száz fordulat percenként (pl. 500)

2. Léptető motor típusok

Felépítés alapján lehet:

- változó mágneses ellenállású (VR) → a forgórész fogazott, lágy mágneses anyag
- állandó mágneses (PM) → a forgórészen állandó mágnes(ek)
- hibrid motor (HB) → az előbbi kettő kombinációja

Vezérlés szempontjából:

- unipoláris, az álló rész minden pólusán két tekercs (egy tekercs középleágazással)
- bipoláris, minden fázishoz egy tekercs → az egyes tekercseken az áram irányát is változtatni kell ! → bonyolultabb vezérlés

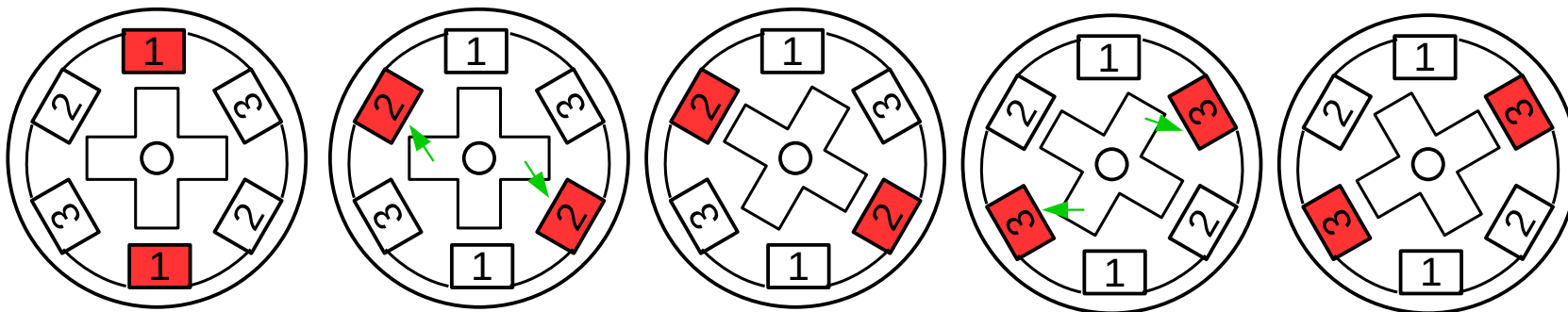
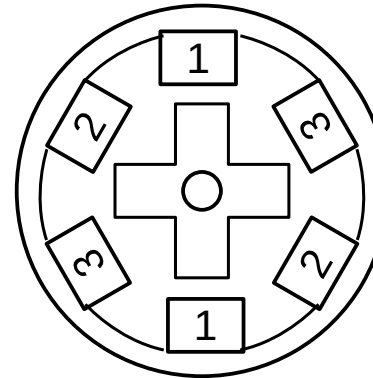
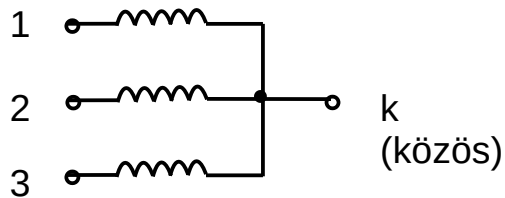
Nagyon jó, szemléletes leírás található a következő oldalon:

http://qtp.hu/elektro/leptetomotor_mukodese.php

11.2. Léptető motorok

3. Változó mágneses ellenállású léptető motorok

- variable reluctance (VR), a forgórész fogazott, lágy mágneses anyag
- az álló részen 3 darab vezérelhető tekercs, 4 kivezetéssel (vagy 5 tekercs, 6 kivezetéssel)
- a tekercsekre megfelelő sorrendben feszültséget kapcsolva (1,2,3,1,2,3,1,...) lépeget az egyik irányba, a sorrendet megfordítva (3,2,1,3,2,1,3,...) a másik irányba



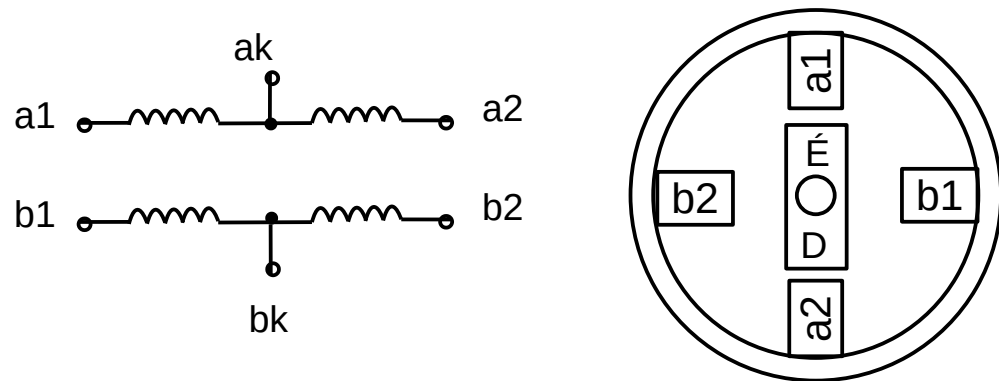
11.3. Léptető motorok

4. Unipoláris léptető motor

- állórészen általában 4 darab vezérelhető tekercs → 5 vagy 6 kivezetéssel
- forgórészen állandó mágnes(ek) → PM motor, vagy esetleg plusz fogazás is → HB motor
- többféle vezérlés lehetséges, de az a lényeg hogy a tekercsre megfelelő sorrendben feszültséget kapcsolva lépeget az egyik, a sorrendet megfordítva a másik irányba

Unipoláris, 6 vezetékes

az 5 vezetékes lényegében ugyanez,
csak a két közös vezeték (ak és bk)
össze van kötve

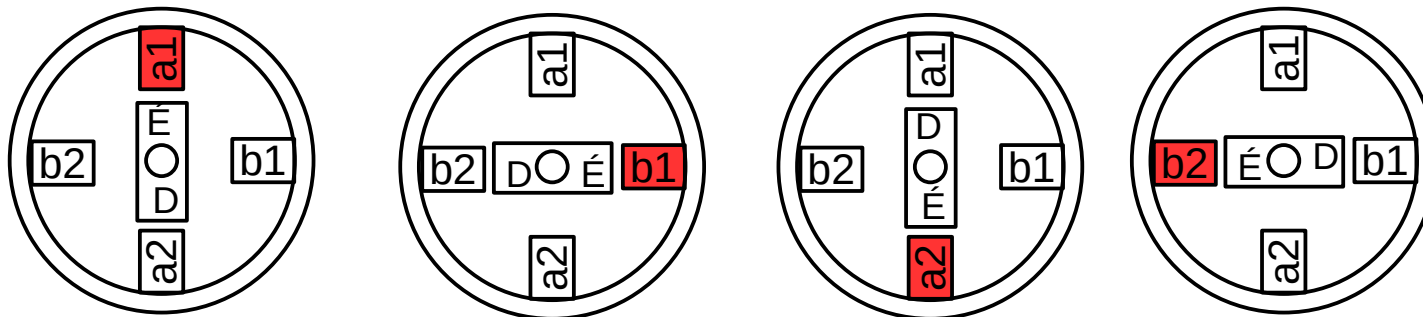


Vezérlési módok:

- egyfázisú (hullám)
- féllépéses (half stepping)
- teljes lépéses (full stepping)
- mikro lépéses (micro stepping)

egyfázisú (hullám) vezérlés

- a 4 tekercsre egyenként, sorban kapcsolunk feszültséget → a1-b1-a2-b2-a1-b1-a2-b2-a1-...



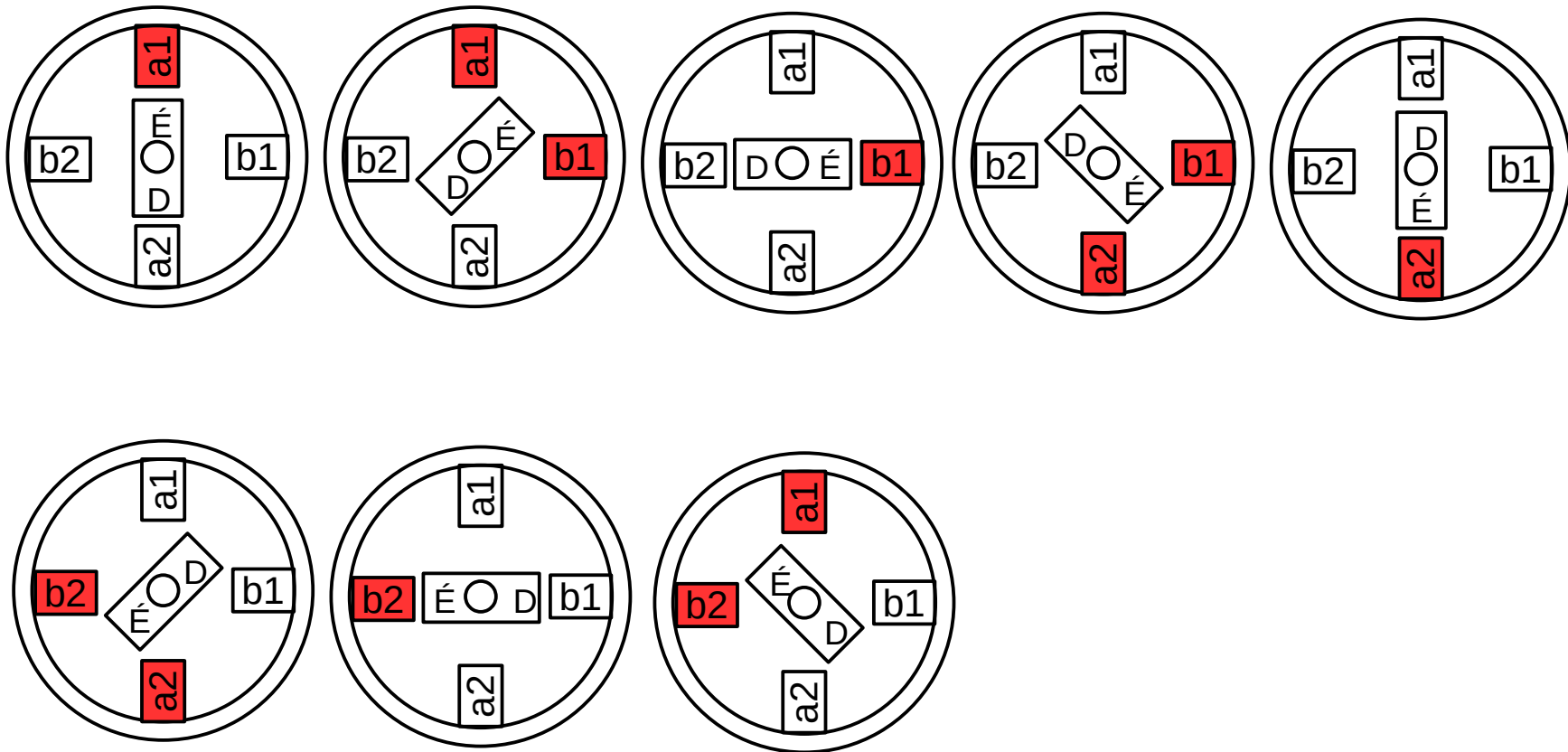
11.4. Léptető motorok

4. Unipoláris léptető motor

féllépéses (half stepping) vezérlés

-az egyfázisú vezérlés lépései közé plusz lépéseket iktatunk be úgy, hogy egyszerre két tekercsre kapcsolunk feszültséget → a lépések száma megduplázódik !

Tehát a vezérlés → a1 – a1,b1 – b1 – b1,a2 – a2 – a2,b2 – b2 – b2,a1 – a1 – a1,b1 – b1 – ...



11.5. Léptető motorok

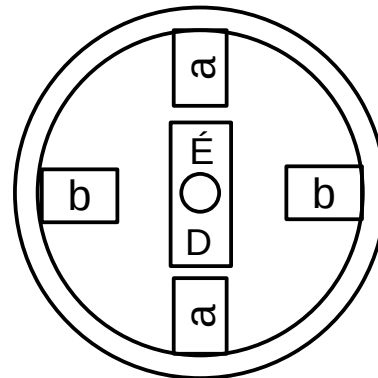
5. Bipoláris léptető motor

- állórészen általában 2 darab vezérelhető tekercs, 4 kivezetés
vagy lehet több tekercs is → pl. 4 tekercs (8 kivezetés)
- forgórészen állandó mágnes(ek) → PM motor, vagy esetleg plusz fogazás is → HB motor
- az egyes tekercseken az áram irányát is változtatni kell ! → H-híd vezérlés

Bipoláris, 4 vezetékes

a1  a2

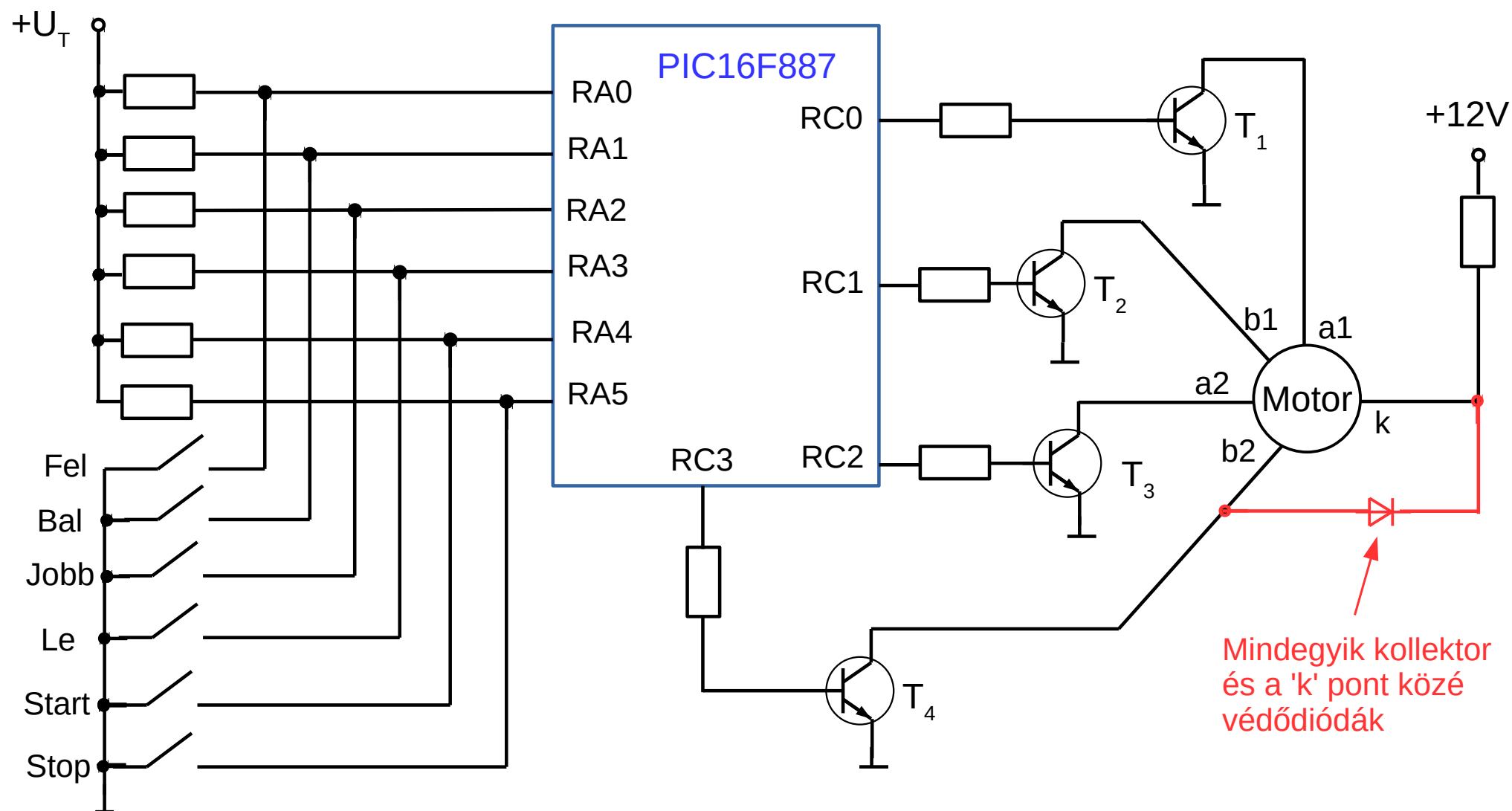
b1  b2



11.6. Unipoláris léptető motor vezérlése

A hardver

- a motor 4 tekercsére tranzisztorokkal kapcsolunk feszültséget, mert jellemzően 5V-nál nagyobb feszültséget igényelnek, és nagy az áram felvételük is !
- tehát a négy tranzisztort kell sorban kapcsolgatni (a bázisukra 1 ill. 0 szint kapcsolásával)



11.7. Unipoláris léptető motor vezérlése

1. mintafeladat

a PORTC-re kapcsolt (RC0-a1 RC1-b1 RC2-a2 RC3-b2) unipoláris léptető motor vezérlése, a következőképpen:

- A JOBB kapcsolót nyomva tartva → a motor lépkedjen 'jobbra' (óramutató járásával megegyezően)
- A BAL kapcsolót nyomva tartva → a motor lépkedjen ellenkező irányba

A kapcsolók a PORTA lábakra vannak kötve !! →

A **PORTA lábak használatához digitális bemenetként** nem elég a TRISA regisztert megfelelően beállítani !! → ugyanis ezen lábak többsége analóg bemenet is lehet → az **ANSEL** regiszterekkel lehet beállítani, hogy melyik láb legyen

- digitális bemenet (**megfelelő bit 0**)
- vagy analóg bemenet (megfelelő bit 1) → ez az alapértelmezett !!

Tehát most ANSEL regisztert nulláznunk kell

A motor léptetése jobbra → feszültség (1-es szint) kapcsolása sorban egyenként
a1,b1,a2,b2 tekercsekre → RC0,RC1,RC2,RC3 lábakra

A motor léptetése balra → feszültség (1-es szint) kapcsolása sorban egyenként
b2,a2,b1,a1 tekercsekre → RC3,RC2,RC1,RC0 lábakra

Tehát egyfázisú, hullám vezérlést használunk

- egyik irány → a1-b1-a2-b2-a1-b1-a2-b2-a1-...
- másik irány → b2-a2-b1-a1-b2-a2-b1-a1-b2-...

11.8. Unipoláris léptető motor vezérlése

1. mintafeladat

// 11.1. léptető motor jobbra-balra

```
void main( ) {
    char i=0;
    char motor[]={1,2,4,8}; // motor tekercsek vezérlése (egyszerre csak egy !)
    TRISC=0b00000000;        // PORTC láb kimenet → motor tekercsek
    ANSEL=0;                  // PORTA lábak nem analóg, hanem digitális bemenetek
    TRISA=0b00111111;        // PORTA lábakon kapcsolók
    PORTC=0;

    while (1)                 // ismétlés sokszor (végtelenszer) !
    {
        while(PORTA.B2==0)    // motor jobbra ha 'JOB' gomb lenyomva
        {
            PORTC = motor[i];
            i++;
            if(i>3) i=0;
            Delay_ms(500);
        }
        while(PORTA.B1==0)    // motor balra ha 'BAL' gomb lenyomva
        {
            PORTC = motor[i];
            if(i>0) i--;
            else i=3;
            Delay_ms(500);
        }
    }
}
```

11.9. Unipoláris léptető motor vezérlése

2. mintafeladat

a PORTC-re kapcsolt (RC0-a1 RC1-b1 RC2-a2 RC3-b2) unipoláris léptető motor vezérlése, a következőképpen:

- A START kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra' (óramutató járásával megegyezően)
- A STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van

Célszerű a motor állapotát külön változóban (vagy változókbán) tárolni → jelenleg egy változó is elég →

a motor_jar változót használjuk:

- 1 értékű, ha forognia kell (START kapcsolót megnyomtuk)
- 0 értékű, ha állnia kell (még el sem indítottuk vagy a STOP kapcsolót megnyomtuk)

A motor léptetése jobbra →

egyfázisú, hullám vezérlést használunk → a1-b1-a2-b2-a1-b1-a2-b2-a1-...

11.10. Unipoláris léptető motor vezérlése

2. mintafeladat

// 11.2. léptető motor jobbra forog-leáll

```
void main( ) {  
    char i=0;  
    char motor_jar=0;           // motor állapotának számontartása → 0-áll 1-jár  
    char motor[]={1,2,4,8};    // motor tekercsek vezérlése (egyszerre csak egy !)  
    TRISC=0b00000000;          // PORTC láb kimenet → motor tekercsek  
    ANSEL=0;                    // PORTA lábak nem analóg, hanem digitális bemenetek  
    TRISA=0b00111111;          // PORTA lábakon kapcsolók  
    PORTC=0;  
  
    while (1)                   // ismétlés sokszor (végtelenszer) !  
    {  
        if(motor_jar==1)       // motor jobbra ha elindítottuk  
        {  
            PORTC = motor[i];  
            i++;  
            if(i>3) i=0;  
        }  
        if(PORTA.B4==0) { motor_jar=1; } // motor indítása ha 'START' gombot lenyomjuk  
        if(PORTA.B5==0) { motor_jar=0; } // motor leállítása ha 'STOP' gombot lenyomjuk  
        Delay_ms(300);  
    }  
}
```

11.11. Unipoláris léptető motor vezérlése

3. mintafeladat, (a 2. feladat módosítása)

a PORTC-re kapcsolt (RC0-a1 RC1-b1 RC2-a2 RC3-b2) unipoláris léptető motor vezérlése, a következőképpen:

- a START kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra' (óramutató járásával megegyezően)
- a STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van
- a FEL kapcsoló lenyomása után → a motor gyorsabban lépkedjen (kb. 3-szoros sebességgel)
- a LE kapcsoló lenyomása után → a motor térjen vissza az alap sebességhez

Célszerű a motor állapotát külön változóban (vagy változókbán) tárolni → jelenleg két változó célszerű →

motor_jar változó

- 1 értékű, ha forognia kell (START kapcsolót megnyomtuk)
- 0 értékű, ha állnia kell (még el sem indítottuk vagy a STOP kapcsolót megnyomtuk)

motor_gyors változó

- 1 értékű, ha nagyobb sebességű
- 0 értékű, ha lassú

A sebességet legegyszerűbben a késleltetés módosításával tudjuk megváltoztatni →
nagy késleltetés → lassú
Kis késleltetés → gyors

11.12. Unipoláris léptető motor vezérlése

3. mintafeladat

```
void main( ) {  
    char i=0;  
    char motor_jar=0;  
    char motor_gyors=0;  
    char motor[]={1,2,4,8};  
    TRISC=0b00000000;  
    ANSEL=0;  
    TRISA=0b00111111;  
    PORTC=0;  
    while (1)  
    {  
        if(motor_jar==1) // motor jobbra ha elindítottuk  
        {  
            PORTC = motor[i];  
            i++;  
            if(i>3) i=0;  
        }  
        if(PORTA.B4==0) { motor_jar=1; } // motor indítása, 'START' gomb  
        if(PORTA.B5==0) { motor_jar=0; } // motor leállítása, 'STOP' gomb  
        if(PORTA.B0==0) { motor_gyors=1; } // motor gyorsítása, 'FEL' gomb  
        if(PORTA.B3==0) { motor_gyors=0; } // motor lassítása, 'LE' gomb  
        if(motor_gyors==1) { Delay_ms(200); } // gyors → kis késleltetés  
        else { Delay_ms(600); } // lassú → nagy késleltetés  
    }  
}
```

11.13. Feladatok

Írj programokat unipoláris léptető motor vezérlésére (a kapcsolás az előzővel megegyező)

- 1. feladat
 - a program indulásakor a motor lépjen 10-et jobbra gyorsan (300ms késleltetés), majd 5-öt balra lassan (1000ms késleltetés)
- 2. feladat
 - a program indulásakor a motor lépjen 10-et jobbra gyorsan (300ms késleltetés),
 - a START kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'balra'
 - a STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van
- 3. feladat
 - a START kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra'
 - a STOP kapcsoló lenyomására → a motor álljon meg abban a pozícióban stabilan, ahol van
 - a FEL kapcsoló lenyomása után → a motor gyorsabban lépkedjen
(kb. 4-szeres sebességgel → 0,5 másodpercenkénti lépések)
 - a LE kapcsoló lenyomása után → a motor térjen vissza az alap sebességhez, 2 másodpercenként lépjen
 - Megoldandó feladat: a nagy késleltetések ellenére, reagáljon gyorsan a kapcsolók lenyomására

11.14. Élfigyelés

Élfigyelés

A bemenetek lekérdezésekor vannak olyan esetek amikor nemcsak az a kérdés, hogy adott kapcsoló/nyomógomb le van-e éppen nyomva (vagy nem), hanem fontos azt is tudnunk, hogy

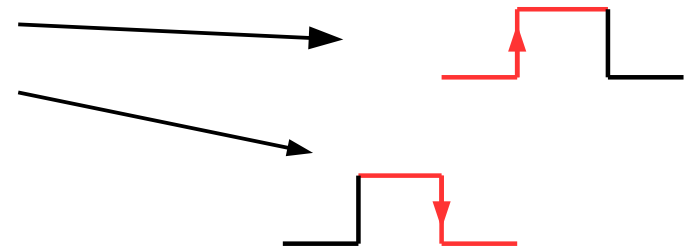
- folyamatosan nyomva tartjuk már egy ideje
- vagy már felengedtük és most újra lenyomtuk

Tehát számolni szeretnénk a lenyomásokat

- azért mert léptetni szeretnénk valamit
- vagy minden új lenyomásra egy új állapotba kell kerülnie a vezérelt rendszerünknek

Az élfigyelés kétféle lehet, figyelhetünk

- felfutó élt → váltás 0 állapotból 1 állapotba
- lefutó élt → váltás 1 állapotból 0 állapotba



Élfigyelés megvalósítása

Többféleképpen lehetséges

- megszakítás használatával → bemenet állapot változásának figyelése
PIC16F887 esetén PORTB lábainak állapotváltozása → pl. 9.3 mintafeladat
- egyszerűen ciklikusan lekérdezzük a bemenet állapotát és összehasonlítjuk az aktuális és az előző lekérdezés értékeit
 - tárolni kell egy változóban a régi (előző) lekérdezés eredményét
 - a lekérdezési frekvenciát annak megfelelően válasszuk meg, hogy milyen gyorsan kell reagálnunk a kapcsoló működtetésére

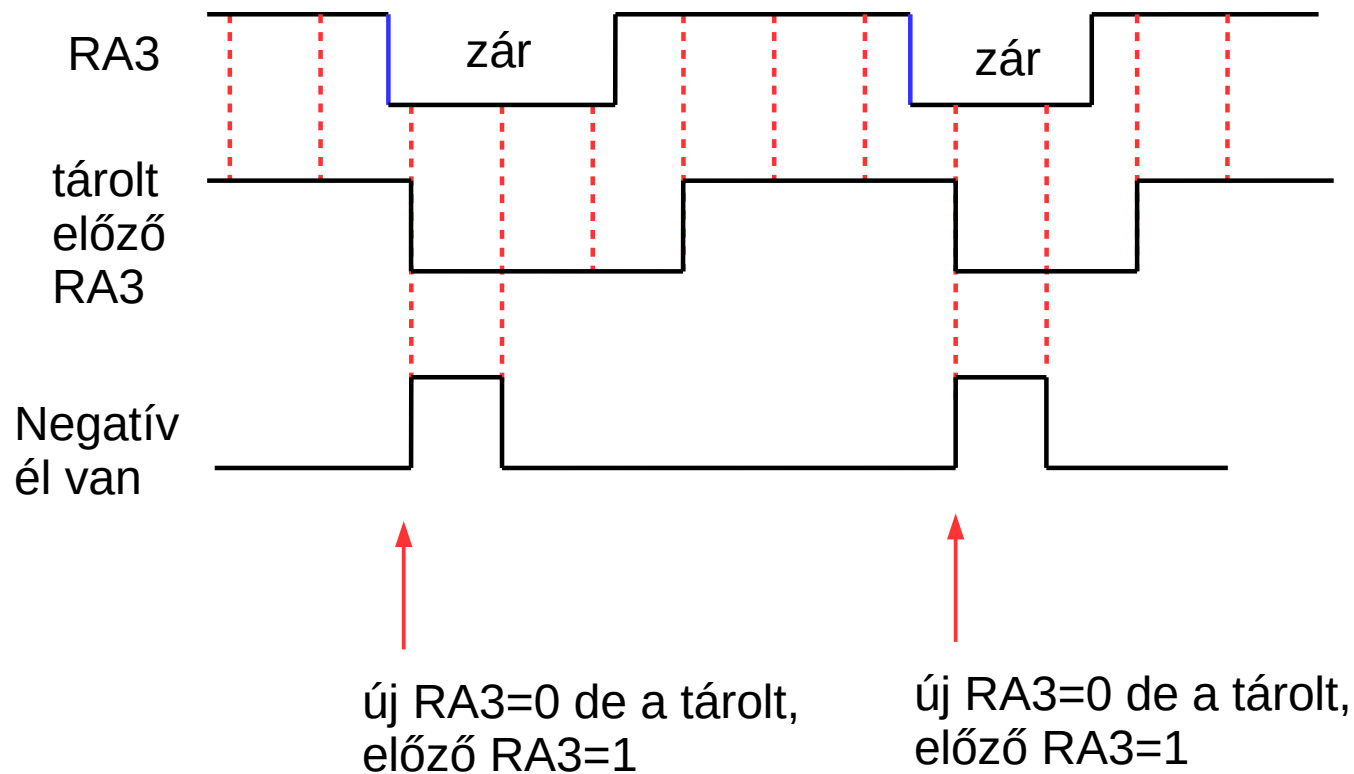
11.15. Élfigyelés

Élfigyelés bemenet ciklikus lekérdezésével

R_{xy} láb figyelése → PORT_x.B_y bit lekérdezése

pl. RA3 láb → PORTA.B3 bit

bekötés: kapcsoló nyitva – H szint, kapcsoló zárva – L szint → negatív él figyelése !!



11.16. Élfigyelés

4. mintafeladat

- A JOBB kapcsoló (RA2) minden lenyomására → a motor lépjen egyet 'jobbra'

// 11.4. motor léptetése egyesével

```
void main( ) {
    char i=0;
    char jobb_uj=1;           // RA2 új lekérdezésének értékét tárolja
    char jobb_regi=1;         // RA2 előző lekérdezésének értékét tárolja
    char motor[]={1,2,4,8};   // motor tekercsek vezérlése (egyszerre csak egy !)
    TRISC=0b00000000;         // PORTC láb kimenet → motor tekercsek
    ANSEL=0;                  // PORTA lábak nem analóg, hanem digitális bemenetek
    TRISA=0b00111111;         // PORTA lábakon kapcsolók
    PORTC = motor[0]; PORTC = motor[1]; // beállítás egy stabil pozícióba
    PORTC = motor[2]; PORTC = motor[3];
    while (1)                 // ismétlés végtelenszer
    {
        if(PORTA.B2==0) { jobb_uj=0; } // 'JOBB' gomb lenyomva
        if(PORTA.B2==1) { jobb_uj=1; } // 'JOBB' gomb felengedve
        if((jobb_uj==0)&&(jobb_regi==1)) // negatív él !! → 1 lépés
        {
            PORTC = motor[i];
            i++;
            if(i>3) i=0;
        }
        jobb_regi=jobb_uj;      // 'JOBB' gomb állapotának eltárolása !
        Delay_ms(100);
    }
}
```

11.17. Feladatok

Írj programokat unipoláris léptető motor vezérlésére (a kapcsolás az előzővel megegyező)

- 1. feladat
 - a FEL kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra'
 - a LE kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'balra'
 - a STOP kapcsoló lenyomására → a motor álljon meg stabilan a pozícióban
 - a JOBB kapcsoló minden lenyomására → a motor lépjen **egy**et 'jobbra'

Késleltetés a lépések között legyen 400ms
- 2. feladat
 - a program indulásakor várjon a START kapcsoló lenyomására ! → ezután a motor lépjen 20-at jobbra gyorsan (250ms késleltetés), majd álljon le
 - ezután
 - a FEL kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'jobbra'
 - a LE kapcsoló lenyomása után → a motor folyamatosan lépkedjen 'balra'
 - a STOP kapcsoló lenyomására → a motor álljon meg stabilan a pozícióban
 - a JOBB kapcsoló minden lenyomására → a motor lépjen **egy**et 'jobbra'
 - a BAL kapcsoló minden lenyomására → a motor lépjen **egy**et 'balra'

Késleltetés a lépések között legyen 400ms

12.1. Analóg-digitális átalakítás

1. Analóg bemenetek

- A mikrovezérlőkben általában van beépítve egy analóg-digitális átalakító (ADC) mint speciális periféria. Segítségével a mikrovezérlő analóg jeleket tud fogadni a külvilág felől, és azt digitálisan fel tudja dolgozni.
- Általában több analóg bemenet is szokott lenni (analóg csatornák) de A/D átalakító csak egy van (közös!) → egy időben egyszerre csak egy analóg bemenet mintájának kódolása folyhat → nekünk kell leprogramozni, hogy adott pillanatban melyik analóg csatorna kerüljön az A/D konverterre

2. A/D a PIC16F887 mikrovezérlőben

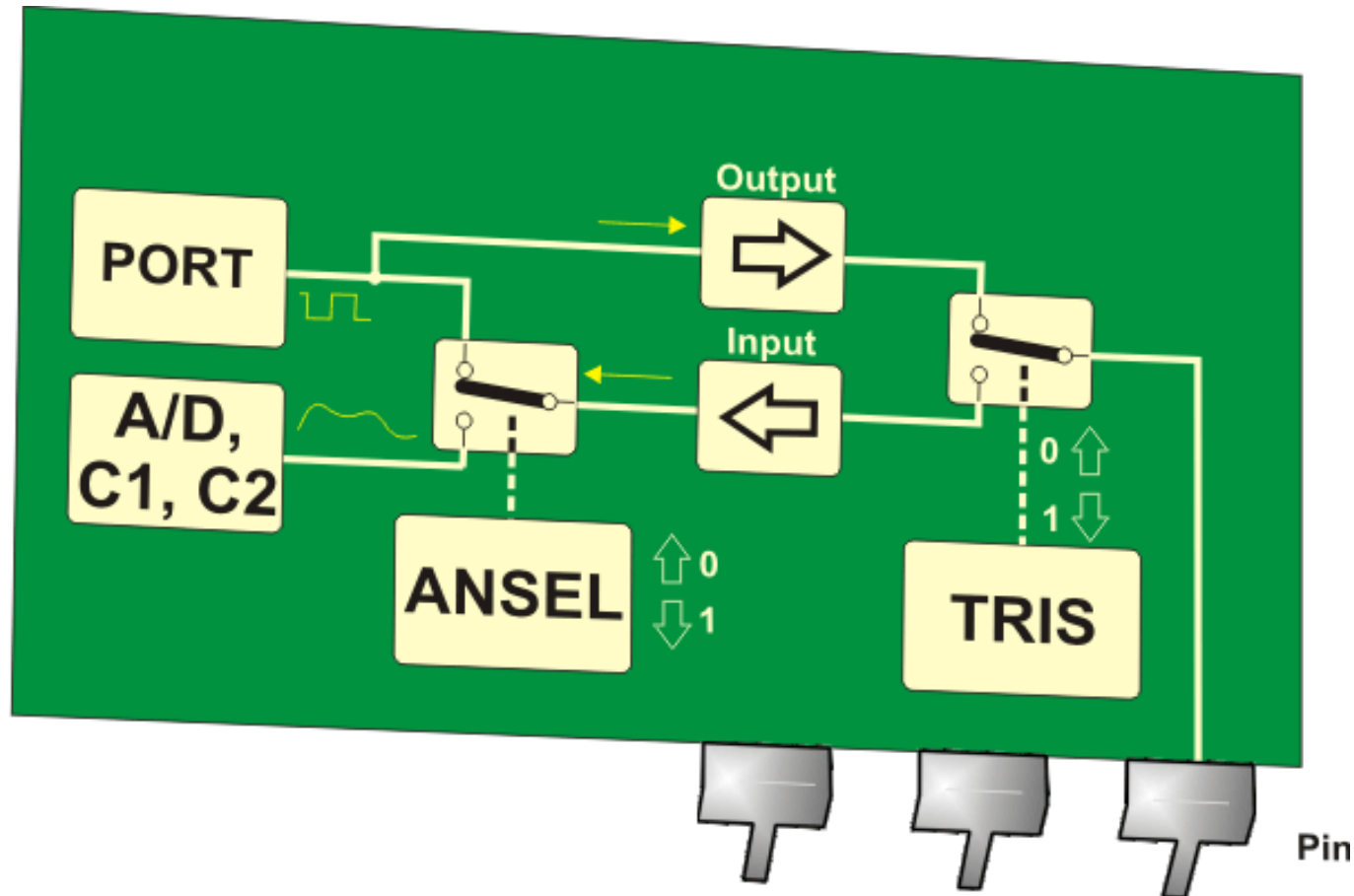
- A mikrovezérlőben egy 10 bites analóg-digitális átalakító van. → a konverzió eredménye ezért két regiszterben tárolódik !! → ADRESH és ADRESL
- 14 db analóg csatorna van (ANSx), és ezek osztoznak a lábakon a digitális be/kimenetekkel !

ANS0 – RA0	ANS5 – RE0	ANS10 – RB1
ANS1 – RA1	ANS6 – RE1	ANS11 – RB4
ANS2 – RA2	ANS7 – RE2	ANS12 – RB0
ANS3 – RA3	ANS8 – RB2	ANS13 – RB5
ANS4 – RA5	ANS9 – RB3	

- megfelelő regiszterekben be kell állítanunk, hogy egy láb most analóg bemenet vagy digitális bemenet/kimenet legyen → ANSEL és ANSELH regiszterek

12.2. Analóg-digitális átalakítás

3. Analóg-digitális bemenet választás



Az ábra forrása → PIC Microcontrollers - Programming in C

12.3. PIC16F887 A/D átalakító beállítása

A/D átalakító regiszterek

	7	6	5	4	3	2	1	0
ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	GO/DONE	ADON

ADCS1, ADCS0 → clock

órajel beállítása

00 → $F_{osc}/2$ 01 → $F_{osc}/8$

10 → $F_{osc}/32$ 11 → RC

GO/DONE → indítás és állapot jelző

1-be állítása → átalakítás indítása

Állapot jelzése

1 → átalakítás folyamatban

0 → átalakítás vége → ezt kell figyelni !!

CHS3, CHS2, CHS1, CHS0 → channel

analóg csatorna kiválasztása

0000 – RA0 0101 – RE0 1010 – RB1

0001 – RA1 0110 – RE1 1011 – RB4

0010 – RA2 0111 – RE2 1100 – RB0

0011 – RA3 1000 – RB2 1101 – RB5

0100 – RA5 1001 – RB3

ADON → A/D engedélyezése (bekapcsolása)

1 → bekapcsolva

0 → kikapcsolva

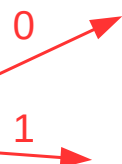
ADRESH, ADRESL

Az A/D konverzió eredményét tárolják 10 biten(E9-E0)

ADRESH

ADRESL

Az eredmény kétféleképpen tárolódhat
ADFM bit



E9	E8	E7	E6	E5	E4	E3	E2	E1	E0	–	–	–	–	–	–
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
–	–	–	–	–	–	E9	E8	E7	E6	E5	E4	E3	E2	E1	E0

12.4. PIC16F887 A/D átalakító beállítása

A/D átalakító regiszterek

	7	6	5	4	3	2	1	0
ADCON1	ADFM	–	VCFG1	VCFG0	–	–	–	–

ADFM

Az eredmény igazítása (kétféleképpen tárolódhat)

VCFG1

Negatív referencia feszültség
0 (alapért.) → GND (V_{SS})
1 → külső, RA2 láb (4)

VCFG0

Pozitív referencia feszültség
0 (alapért.) → V_{DD} (+5V)
1 → külső, RA3 láb (5)

	7	6	5	4	3	2	1	0
ANSEL	ANS7	ANS6	ANS5	ANS4	ANS3	ANS2	ANS1	ANS0
	RE2	RE1	RE0	RA5	RA3	RA2	RA1	RA0

Bemenet típusa (analóg/digitális)

1 → analóg

0 → digitális

1 az alapértelmezett !!

	7	6	5	4	3	2	1	0
ANSELH	–	–	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8
			RB5	RB0	RB4	RB1	RB3	RB2

12.5. A/D átalakító használata

A/D beállítása, használata

1. lépés

Analóg láb beállítása

- ANSEL, ANSELH regiszterek → megfelelő láb értéke 0 legyen ! (analóg)
pl. `ANSELH=0x1F;` → RE0, RE1, RE2 lábak analóg bemenetek
- TRISx → megfelelő láb értéke 1 legyen ! (bemenet)
pl. `TRISE=0x07;` → RE0, RE1, RE2 lábak bemenetek

2. lépés

A/D konfigurálása

- ADCON1 → referencia feszültség, eredmény igazítása
pl. `ADCON1=0x80;` → eredmény jobbra igazítva (ADRESH alsó két bit + ADRESL),
referencia feszültség V_{DD} és GND
- ADCON0 → órajel, analóg csatorna beállítása
pl. `ADCON0=0x94;` → órajel - $F_{osc}/32$ analóg csatorna - RE0

3. lépés

A/D interrupt beállítások, nem feltétlen szükséges !!

- INTCON, PIE1 regiszterek

Az első három lépés utasításait jellemzően a program elején, a kezdeti beállítások szakaszában helyezzük el, de természetesen később a programban bármikor lehet módosítani a beállításokat (jellemzően a bemeneti csatornát kell állítani, ha több analóg bemenet is van)

12.6. A/D átalakító használata

A/D beállítása, használata

4. lépés

A/D konverter bekapcsolása (használat előtt)

- ADCON0 → ADON bit 1-be állítása

`ADCON0=ADCON0|1;` // bitenkénti VAGY művelettel az utolsó bit 1-be állítása (ADON)

5. lépés

A/D konverzió indítása (amikor kell az analóg bemenet aktuális értéke)

- ADCON0 → GO(/DONE) bit 1-be állítása

`ADCON0=ADCON0|2;` // bitenkénti VAGY művelettel az utolsó előtti bit 1-be állítása (GO)

6. lépés

Várakozás a konverzió eredményére → ez kétféleképpen történhet:

- GO/DONE bit figyelése ! → amikor 0 lesz újra → akkor megvan az átalakítás eredménye

pl. `while((ADCON0&2)!=0);`

- A/D interrupt → ha engedélyezve volt és GO/DONE bit = 0 → automatikusan

7. lépés

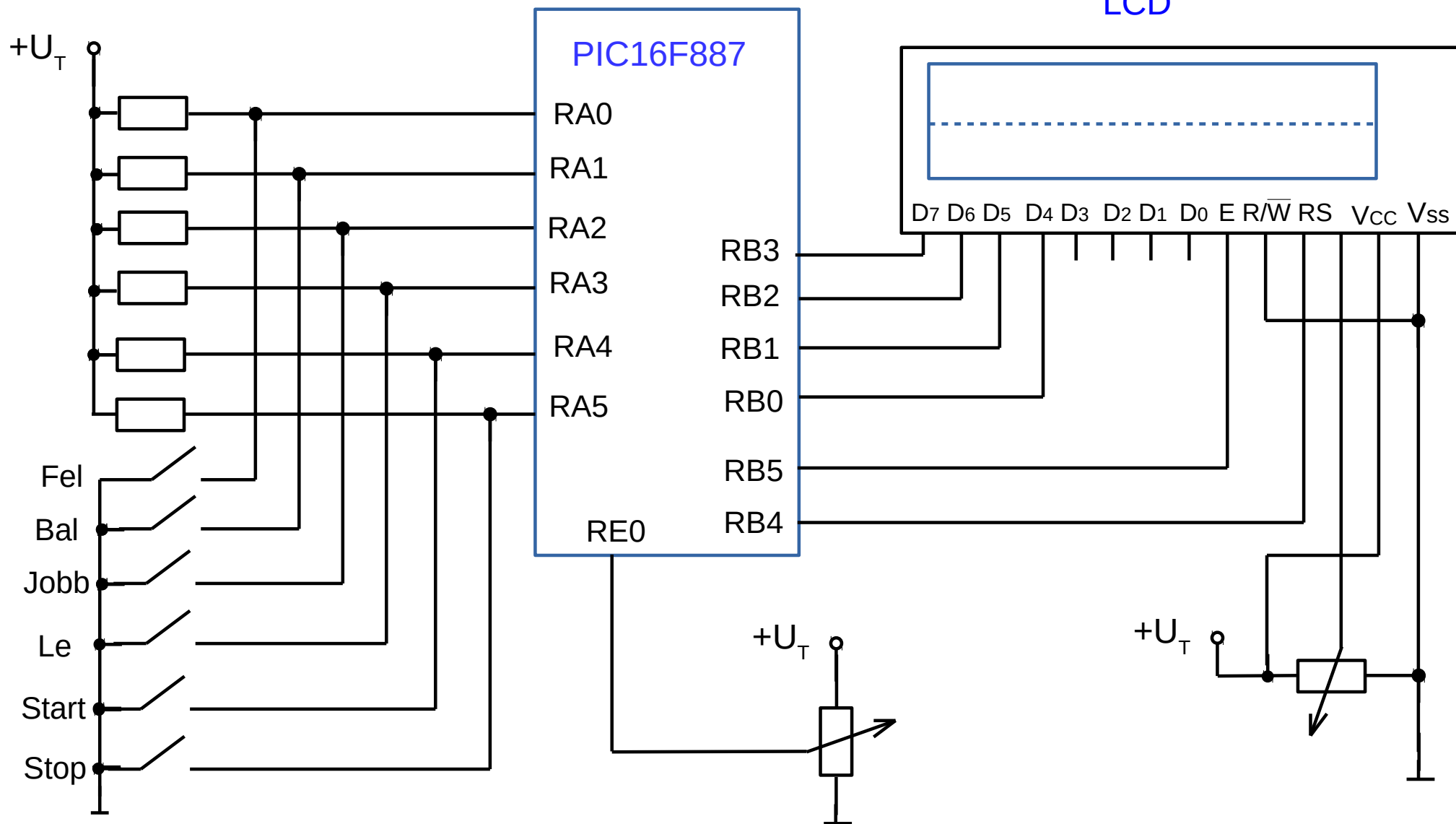
Eredmény kiolvasása

- ADRESH, ADRESL regiszterekből → az ADFM bitnek megfelelően értelmezve !!

pl. `eredmeny=((ADRESH&3)<<8)+ADRESL;` → ha az eredmény jobbra igazítva,
eredmeny 16 bites (int) legyen !!

12.7. A/D átalakító, minta program

A hardver



12.8. A/D átalakító, minta program

1. minta feladat (MikroC függvényeit használva)

Írjuk ki az LCD kijelző 1. sorába egy tájékoztató szöveget
a 2. sorban pedig folyamatosan jelenjen meg a potméterrel beállított feszültségérték

Globális változók, LCD bekötésének definiálása (a program legelején !)

```
int poti=0;           // globális változók
int fesz=0;           // az A/D átalakítás eredményre
char szam1=0;         // feszültség mV-ban
char szam2=0;         // első számjegy
char szam3=0;         // 2. számjegy
                     // 3. számjegy

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;
```

12.9. A/D átalakító, minta program

1 minta feladat (MikroC függvényeit használva)

Regiszterek beállítását végző függvény

```
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=0b11100000;    // PORTE lábak analóg bemenetek, 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
    /* ADC_Init */
    ADCON1= 0b10000000;    // ADFM bit → 1 → eredmény: ADRESH alsó két bit + ADRESL
    ADCON0= 0b11010100;    // AN5 (RE0) lesz a bemeneti csatorna,
    ADCON0=ADCON0|1;      // A/D engedélyezés
}
```

12.10. A/D átalakító, minta program

1. minta feladat (MikroC függvényeit használva)

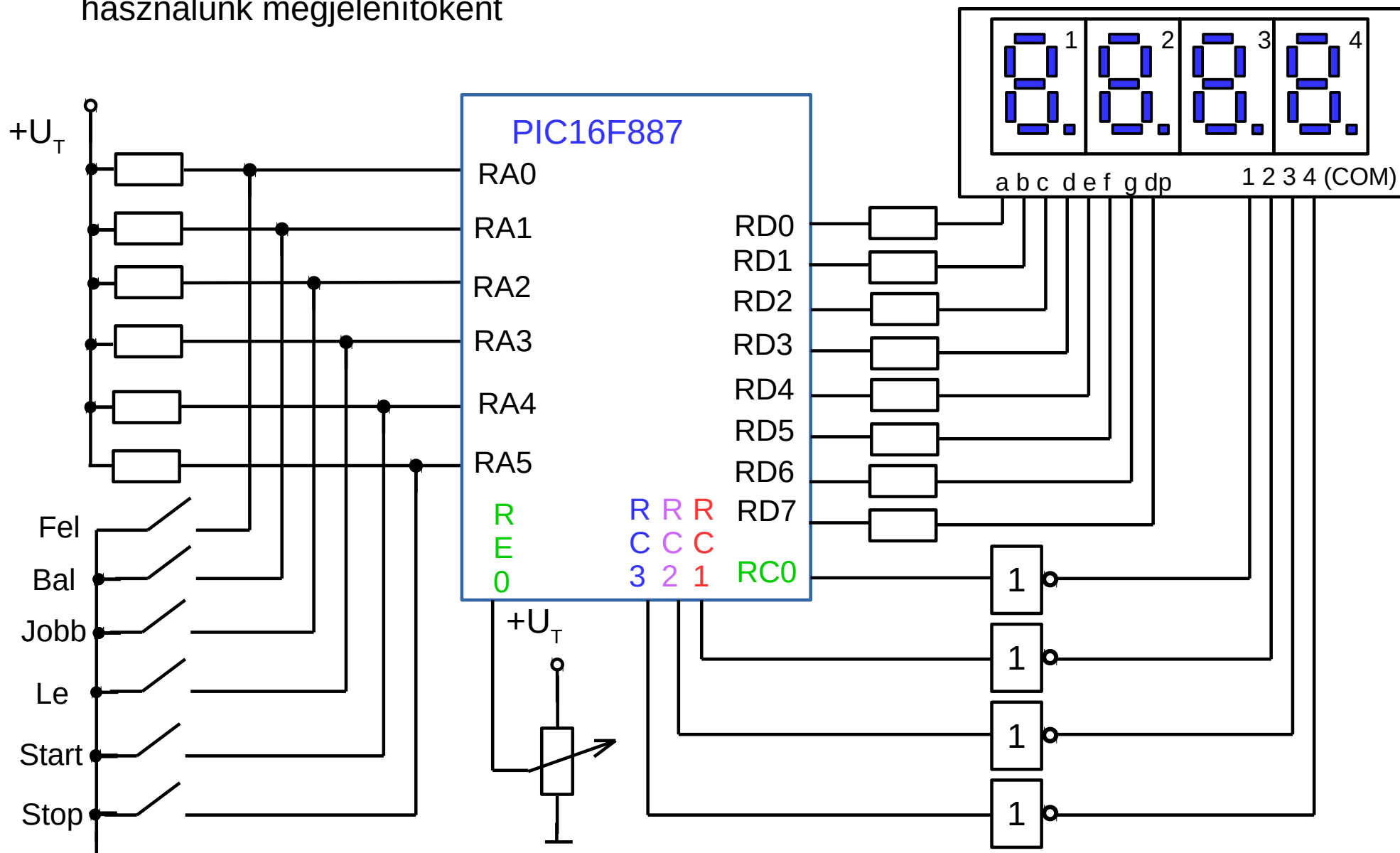
A fő program:

```
void main( )
{
    kezd_beall();
    Delay_ms(500);
    Lcd_out(1,1,"A feszultseg:"); // szöveg kiíratása az első sorba
    while (1)                     // ismétlés sokszor (végtelenszer) !
    {
        ADCON0=ADCON0|2; // GO bit -->1 A/D indítása
        while((ADCON0&2)!=0); // várakozás A/D eredményre amikor DONE=0 lesz
        poti=ADRESH&3; // ADRESH alsó két bitje kell csak !
        poti=(poti<<8)+ADRESL; // eredmény: ADRESH alsó két bit + ADRESL
        fesz=5000.0*poti/1024+2500.0/1024; // a feszültség 0 és 5000 mV között lehet
        if(fesz%10>4) fesz=fesz+5; // kerekítés
        fesz=fesz/10; // 3. tizedesjegy levágása
        szam3=fesz%10; // 2. tizedesjegy
        fesz=fesz/10;
        szam2=fesz%10; // 1. tizedesjegy
        szam1=fesz/10; // egész érték
        Lcd_Chr(2,1,48+szam1); // 1. számjegy (egész) küldése LCD-re, 2. sor 1. pozícióba
        Lcd_Chr_Cp('.');
        Lcd_Chr_Cp(48+szam2); // 2. számjegy küldése (1. tizedesjegy)
        Lcd_Chr_Cp(48+szam3); // 3. számjegy küldése (2. tizedesjegy)
        Lcd_Chr_Cp('V');
        Delay_ms(500);
    }
}
```

12.11. A/D átalakító, minta program 2.

A hardver

- 4 digites, multiplexelt 7 szegmenses kijelzőt használunk megjelenítőként



12.12. A/D átalakító, minta program 2.

2. minta feladat

Digitalizáljuk az RE0 lábon lévő analóg jelet (változtatása potméterrel) és jelenítsük meg a kapott digitális értéket (0-1023) **VAGY** a megfelelő feszültség értéket mV-ban (0-5000) a hétszegmenses kijelzőn. A FEL és LE kapcsolókkal lehet váltani a megjelenítendő értékek között !

// globális változók és kezdeti beállítások

char tomb7[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F}; // hétszegm. kijelző kódok

void kezd_beall()

{

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=0b11100000; // PORTE lábak analóg bemenetek, PORTA lábak digitálisak

TRISE=0b00000111; // PORTE lábak bemenetek

PORTD=0;

PORTC=0; // szegmensek lekapcsolva

/* ADC_Init */

ADCON1= 0b10000000; // ADFM bit → 1 → eredmény: ADRESH alsó két bit + ADRESL

ADCON0= 0b11010100; // AN5 (RE0) lesz a bemeneti csatorna,

ADCON0=ADCON0|1; // A/D engedélyezés

}

12.13. A/D átalakító, minta program 2.

2. minta feladat

Függvény a szám megjelenítésére:

```
void szam_szegm7re(int szam)
{
    char szamj1,szamj2,szamj3,szamj4;
    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
    PORTC=0b00000000;        // mindegyik kijelző kikapcsolása
    PORTD=tomb7[szamj1];      // 1. számjegy
    PORTC=0b00000001;        // 1. kijelző bekapcsolása
    Delay_ms(4);
    PORTC=0b00000000;        // mindegyik kijelző kikapcsolása
    PORTD=tomb7[szamj2];      // 2. számjegy
    PORTC=0b00000010;        // 2. kijelző bekapcsolása
    Delay_ms(4);
    PORTC=0b00000000;        // mindegyik kijelző kikapcsolása
    PORTD=tomb7[szamj3];      // 3. számjegy
    PORTC=0b00000100;        // 3. kijelző bekapcsolása
    Delay_ms(4);
    PORTC=0b00000000;        // mindegyik kijelző kikapcsolása
    PORTD=tomb7[szamj4];      // 4. számjegy
    PORTC=0b00001000;        // 4. kijelző bekapcsolása
    Delay_ms(4);
}
```

12.14. A/D átalakító, minta program 2.

2. minta feladat

A fő program:

```
void main( )
{
    int poti=0;                // az A/D átalakítás eredménye (0-1023)
    int fesz=0;                // feszültség mV-ban (0-5000)
    char ciklus=0;              // ciklusváltozó
    char fent=0;                // állapotváltozó (flag), 0 – poti    1 – fesz    kijelzése
    kezd_beall();               // kezdeti beállításokat elvégző függvény meghívása
    Delay_ms(200);
    while (1)                   // ismétlés végtelenszer !
    {
        if(ciklus%20==0)        // A-D átalakítás ne legyen túl gyakran !! → minden 20. ciklusban (~0,4s)
        {
            ADCON0=ADCON0|2;    // GO bit -->1    A/D indítása
            while((ADCON0&2)!=0); // várakozás A/D eredményre amikor DONE=0 lesz
            poti=ADRESH&3;        // ADRESH alsó két bitje kell csak !
            poti=(poti<8)+ADRESL; // eredmény: ADRESH alsó két bit + ADRESL
            fesz=5000.0*poti/1024+2500.0/1024; // a feszültség 0 és 5000 mV között lehet
        }
        if(fent) { szam_szegm7re(fesz); } // fent=1 → feszültség    kijelzése
        else { szam_szegm7re(poti); }     // fent=0 → digitális érték    kijelzése
        if(PORTA.B0==0) { fent=1; }       // 'FEL' gomb lenyomása
        if(PORTA.B3==0) { fent=0; }       // 'LE' gomb lenyomása
        ciklus++;
    }
}
```


13.1. Időzítők a mikrovezérlőkben

1. Időzítő (Timer)

A mikrovezérlőkben általában van beépítve egy vagy több időzítő (timer) mint speciális periféria.

Ezek igazából számláló áramkörök:

- számolhatnak valamelyik külső lábon érkező impulzusokat (fel- vagy lefutó élt)
- számolhatnak egy meghatározott frekvenciájú belső órajel impulzusait → a számláló értékei adott időtartamnak felelnek meg → időzítésre használható

Használatuk: amikor a számlálást végző regiszter végállapotból újra a 0 állapotba fordul → ez megszakítást okoz → jelzi a megadott idő leteltét (vagy megadott számú impulzus beérkezését)

Általában programozható előosztó is tartozik hozzájuk (esetleg még utó osztó is), amellyel a számlálási frekvencia csökkenthető

2. Időzítők a PIC16F887 mikrovezérlőben

- 3 timer van ebben a mikrovezérlőben:

Timer0 – 8 bites Timer1 – 16 bites Timer2 – 8 bites

- mindegyik timerhez több regiszter is tartozik → amelyekben a számlálás folyik
+ a beállító regiszterek (a számlálás forrása, előosztó beállítása, ...)

-

13.2. PIC16F887 Timer0 beállításai

Timer0-val kapcsolatos regiszterek

OPTION_REG

7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

T0CS Timer0 clock select

0 – belső óra ($F_{osc}/4$) → időzítő (timer) mód
1 – külső ! (T0CKI/RA4 láb) → számláló mód (counter)

T0SE külső jel (RA4/T0CKI) él

0 – felfutó él (rising edge)
1 – lefutó él (falling edge)

PSA előosztó hozzárendelés
(Prescaler – PS2, PS1, PS0)

0 – Prescaler Timer0-hoz
1 – Prescaler WDT-hez

PS2, PS1, PS0 előosztó (Prescaler)

000 – 1/2	100 – 1/32
001 – 1/4	101 – 1/64
010 – 1/8	110 – 1/128
011 – 1/16	111 – 1/256

TMR0

Ez tárolja az időzítő aktuális értékét

INTCON

7	6	5	4	3	2	1	0
GIE	PEIE	TOIE	INTE	RBIE	T0IF	INTF	RBIF

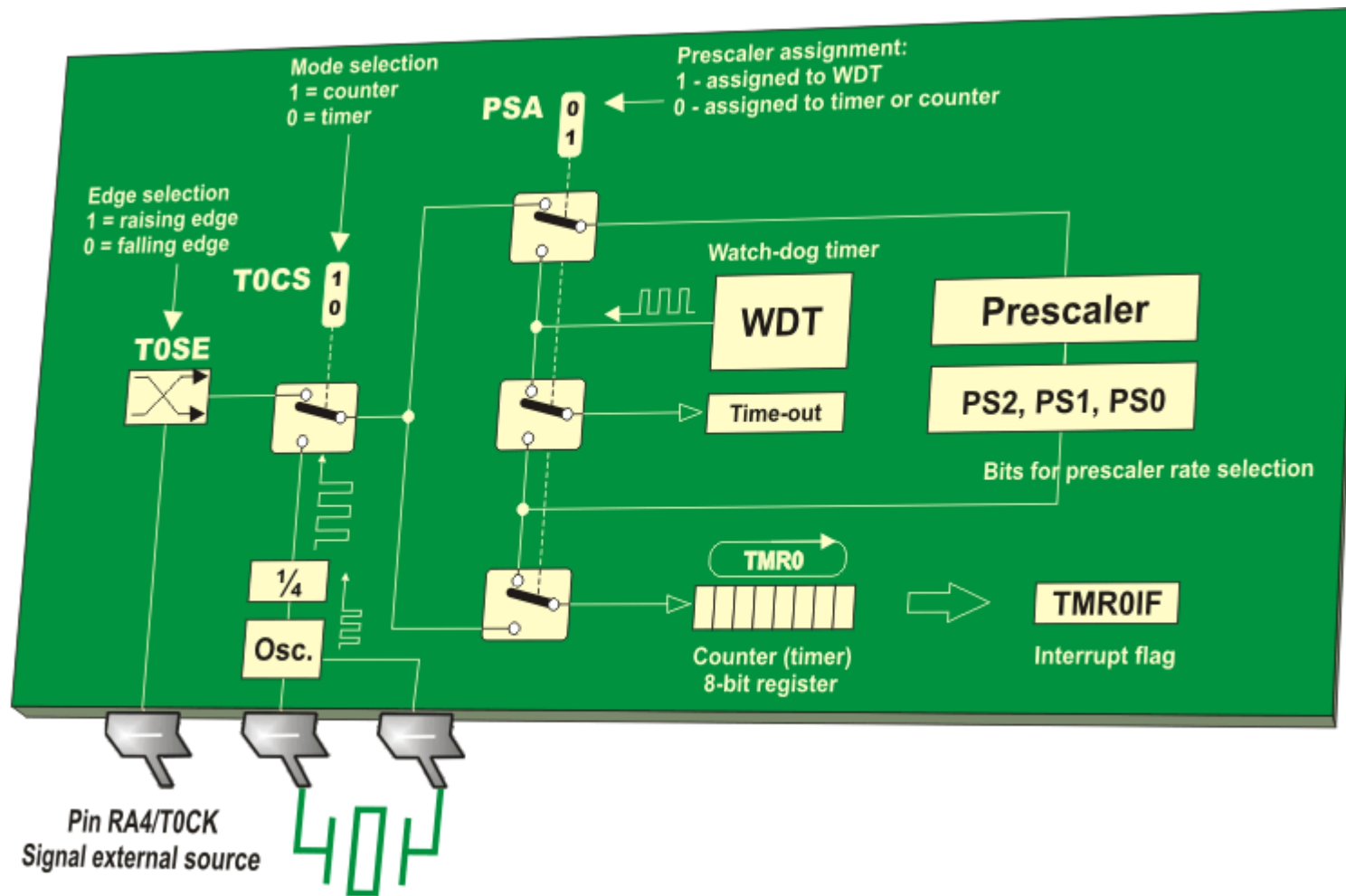
GIE - Global Interrupt Enable bit → engedélyezi (1) vagy tiltja (0) az összes lehetséges megszakítást

TOIE - TMR0 Overflow Interrupt Enable bit → Timer0 megszakítás engedélyezés (ha 1-re állítjuk)

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

13.3. PIC16F887 Timer0 beállításai

Timer0 vezérlése



Az ábra forrása → PIC Microcontrollers - Programming in C

13.4. PIC16F887 Timer0 beállításai

Timer0 beállítási példák

Belső időzítés

main() függvényben

...

```
OPTION_REG=0x06; // 00000110 → T0CS=0 (időzítő), PS2=1, PS1=1, PS0=0 → 1/128  
TMR0=130; // kezdő érték =130 → számol 255-ig → frekvenciát osztja 125-el (255-130)  
INTCON=0xA0; // 10100000 → GIE=1 és T0IE=1 → Timer0 megszakítás engedélyezése
```

...

interrupt() függvényben

...

```
TMR0=130; // kezdő érték = 130  
INTCON=0xA0; // 10100000 → Timer0 megszakítás engedélyezése, T0IF flag törlése
```

Tehát az időzítés frekvenciája → $f_{T0} = ((F_{osc}/4)/128)/125 = F_{osc}/64000$
pl. ha $F_{osc}=f_{clock}=8\text{MHz}$ → $f_{T0}=125\text{Hz}$

Ha TMR0 kezdőértéket megváltoztatjuk

pl. $TMR0=230$; → kezdő érték = 230 → számol 255-ig → frekvenciát osztja 25-el
→ $f_{T0} = ((F_{osc}/4)/128)/25 = F_{osc}/12800$
ha $f_{clock}=8\text{MHz}$ → $f_{T0}=625\text{Hz}$

13.5. PIC16F887 Timer1 beállításai

Timer1-el kapcsolatos regiszterek

T1CON

T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
7	6	5	4	3	2	1	0

T1GINV

Timer1 gate invert

0 – aktív alacsony szint

1 – aktív magas szint

TMR1GE

Timer1 gate enable

Timer1 gate → $\overline{\text{T1G}}$ láb (RB5)

0 – Timer1 on

1 – Timer1 on, ha T1G láb aktív

T1CKPS1, T1CKPS0

Órajel előosztó (Prescaler)

00 – 1/1 10 – 1/4

01 – 1/2 11 – 1/8

TMR1H (felső 8 bit) és

TMR1L (alsó 8 bit) → Ezek tárolják az időzítő aktuális értékét

INTCON és **PIE1** és **PIR1**

Timer1 megszakítás engedélyezése, figyelése

TMR1CS

Timer1 clock select

0 – belső óra ($F_{osc}/4$) → timer mód

1 – külső ! (T1CK1/RC0 láb) → számláló mód

TMR1ON

TMR1 bekapcsolása

0 – off 1 – on

T1SYNC

Külső órajel szinkron vezérlő bit

0 – 1 –

T1OSCEN

Timer1 oszcillátor (32,768kHz)

engedélyezés (T1OSI/RC1-T1OSO/RC0)

0 – LP oszc. off

1 – LP oszc. on

13.6. PIC16F887 Timer1 beállításai

Timer1 beállítási példák

Belső időzítés

main() függvényben

```
...  
T1CON=0b00110101; // TMR1CS=0 → belső órajel (időzítő),  
                    // előosztó → T1CKPS1=1, T1CKPS0=1 → 1/8  
TMR1H=11;  
TMR1L=219; // kezdő érték = 11*256+219=3035 → számol 65535-ig →  
           // frekvenciát osztja 62500-al (65535-3035)  
INTCON=0xC0; // 11000000 → GIE=1 és PEIE=1 → periféria megszakítások engedélyezése  
PIE1=0x01;   // TMR1IE=1 → Timer1 megszakítás engedélyezése
```

interrupt() függvényben

```
...  
PIR1=0;      // jelzőbit törlése  
TMR1H=11;  
TMR1L=219;   // kezdő érték = 3035  
INTCON=0xC0; // 10100000 → GIE=1 és PEIE=1 → periféria megszakítások engedélyezése  
PIE1=0x01;   // TMR1IE=1 → Timer1 megszakítás engedélyezése
```

Tehát az időzítés frekvenciája → $f_{T1} = ((F_{osc}/4)/8)/62500 = F_{osc}/2000000$

pl. ha $F_{osc}=f_{clock}=8\text{MHz}$ → $f_{T1}=4\text{Hz}$

14.1. Konfigurációs memória

1. Program memória felosztása

Mikrovezérlőknél a program memória alapvetően két részből áll

- a nagyobb része a **felhasználói program memória** → ide töltődik be a program
a PIC16F887 esetében ez 8k szó kapacitású → $8 \times 1024 \times 14$ bit
ennek a címtartomány $0000h - 1FFFh$
- egy kis méretű **konfigurációs memória** → ez különféle azonosítókat, beállításokat tárol
mérete általában csak néhány byte, de mérete és tartalma mikrovezérlő típustól függ
a PIC16F887 esetében ennek a címtartománya elvileg $2000h - 3FFFh$,
de gyakorlatilag csak $2000h - 2009h$ van használatban →
 10 szó kapacitású → 10×14 bit

2. A konfigurációs memória (PIC16F887)


Több részből áll:

- azonosító mező ($2000h - 2003h$) → azonosító, ellenőrző összeg, ...
- típusazonosító ($2006h$)
- **konfigurációs szó** (szavak) → **konfigurációs bitek**, 3 db regiszterbe rendezve
CONFIG1 ($2007h$), CONFIG2 ($2008h$), CONFIG3 ($2009h$)

A konfigurációs biteket programból nem tudjuk módosítani !! → a program beírásakor lehet ezeket is beállítani → vagy a programozó szoftverben tudjuk őket módosítani,
→ vagy a program fejlesztő környezetben tudjuk őket beállítani →
fordításkor bekerülnek a hexa állományba

14.2. Konfigurációs bitek

CONFIG1

13	12	11	10	9	8	7	6	5	4	3	2	1	0
DEBUG	LVP	FSCM	IESO	BODEN1	BODEN0	CPD	CP	MCLRE	PWRTE	WDTEN	FOSC2	FOSC1	FOSC0
on	off	off	off	off	off	on	on	on	on	off	 default		

LVP

Alacsony feszültségű programozás engedélyezése

FSCM (FCMEIV)

Fail-safe clock monitor
Átkapcsolás engedélyezése
belső oszcillátorra, ha a külsővel
probléma van

IESO

Int-ext switchover
Átkapcsolás engedélyezése belső
oszcillátorra, amíg a külső nem
stabilizálódik
(sleep módból kilépéskor)

CP

Program memória
védelem

CPD

adat memória
védelem

MCLRE

Master clear enable
(1-es, reset láb)

PWRTE

Power up timer enable

WDTEN

Watch dog timer
enable

FOSC bitek

Oszcillátor típusának megadása

111 → RC, OSC2 órajel ki

110 → RCIO, OSC2 I/O

101 → INTOSC, belső órajel
OSC2 órajel ki, OSC1 I/O

100 → INTOSCIO, belső órajel
OSC2 I/O, OSC1 I/O

011 → EC, OSC1 órajel bemenet
OSC2 I/O

010 → HS, kvarc/kerámia, >4MHz

001 → XT, kvarc/kerámia, <4MHz

000 → LP, kvarc, <200kHz

14.3. Konfigurációs bitek

CONFIG2

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	WRT1	WRT0	BOR4V	1	1	1	1	1	1	1	1

CONFIG3

13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	FCAL5	FCAL5	FCAL4	FCAL3	FCAL2	FCAL1	FCAL0	POR2	POR1	POR0	BOR2	BOR1	BOR0

FCAL6-FCAL0

-

POR2-POR0

-

BOR2-BOR0

-

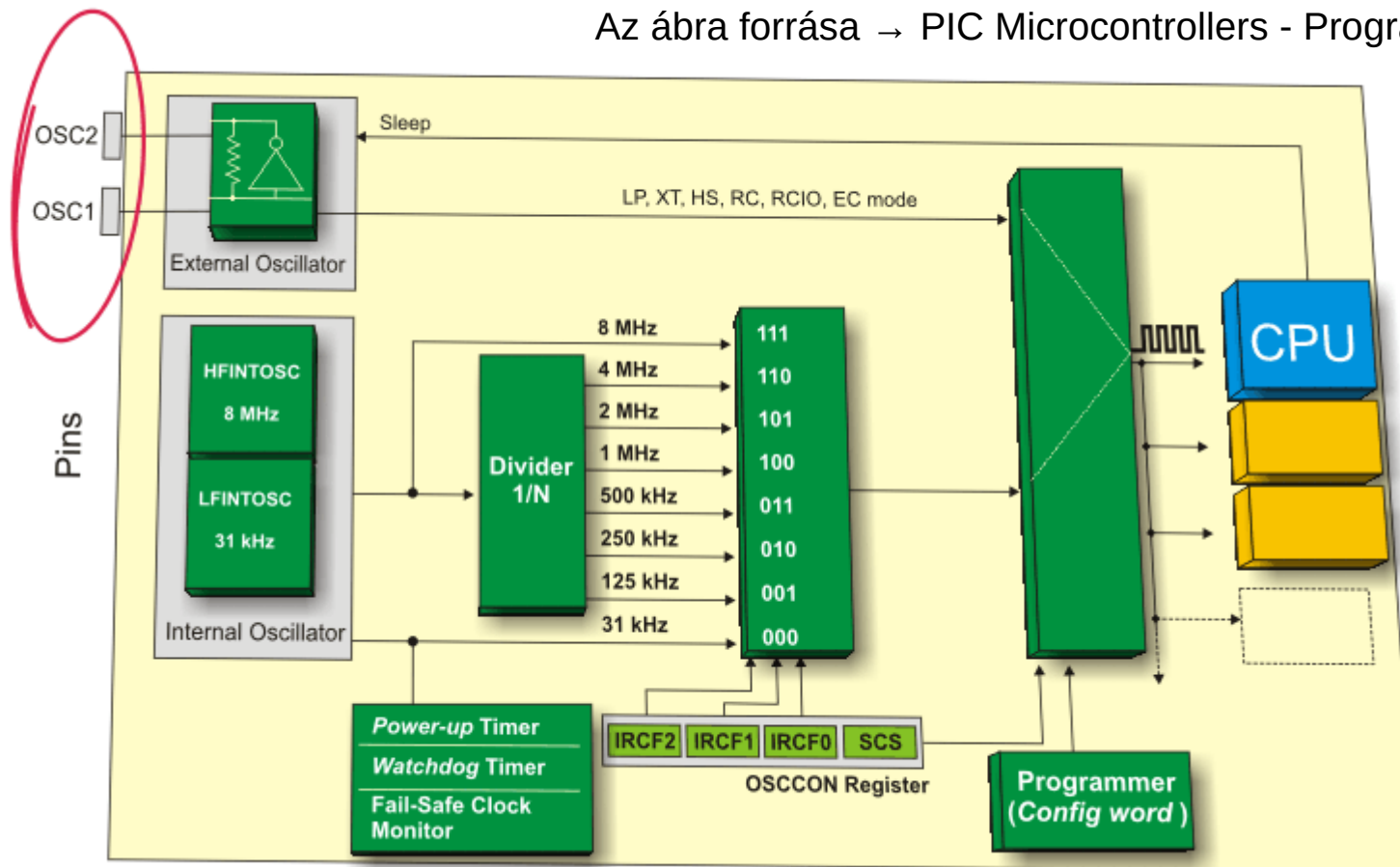
14.4. Oszcillátor beállítások

Órajel hardver

A PIC16F887 mikrovezérlő többféleképpen kaphat órajelet

- külső oszcillátor az OSC1 lábon → pontos megadás konfigurációs bitekkel (CONFIG1 regiszter) → EC
- külső kvarc (vagy kerámia rezonátor) rákapcsolásával → CONFIG1 regiszter
 - LP – kvarc, <200kHz XT – kvarc/kerámia 100kHz-4MHz HS – kvarc/kerámia >4MHz
 - RC – RC oszcillátor, <4MHz
- belső 8MHz-es oszcillátor → ehhez utó osztó (Postcaler) használható → OSCCON regiszter
- belső 31kHz-es oszcillátor

Az ábra forrása → PIC Microcontrollers - Programming in C



14.5. Oszcillátor beállítások

Oszcillátor kontroll regiszter

	7	6	5	4	3	2	1	0
OSCCON	-	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS

IRCF2, IRCF1, IRCF0

utóosztó (Postcaler) HFINTOSC órajelhez
(belső 8MHz-es),
vagy az LFINTOSC órajel kapcsolása
(belső 31kHz-es)

000 → 31kHz
001 – 1/64 → 125kHz
010 – 1/32 → 250kHz
011 – 1/16 → 500kHz
100 – 1/8 → 1MHz
101 – 1/4 → 2MHz
110 – 1/2 → 4MHz
111 – 1/1 → 8MHz

OSTS

0 – belső órajel (internal clock)
1 – külső órajel (external clock)

HTS

1 – HFINTOSC is stable
0 – not stable

LTS

1 – LFINTOSC is stable
0 – not stable

SCS

System clock select bit

1 – belső órajel (internal clock)
0 – külső órajel (external clock) → CONFIG1 regiszter
(FOSC bitek)