În acest laborator vom învăţa să folosim pachetul de programe MPLAB IDE, vom urmări regiştrii STATUS, OPTION şi INTCON, vom analiza un exemplu folosind adresare indirectă şi în cele din urmă ne vom opri asupra unui program scris în limbaj de asamblare.

Registrul STATUS

R/W-0	R/W-0	R/W-0	R -1	R -1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit7							

Legendă:
R = Bit de citire **W** = Bit de scriere
U = Bit neinplementat, citit ca '0' - n = Valoare la resetul power-on

bit 0: **C** (Carry) Transfer
Bit care este afectat de operaţiile de adunare, scădere şi transfer.
1= transferul produs din bitul cel mai înalt al rezultatului
0= transferul nu s-a produs
Bitul C este afectat de instrucţiunile ADDWF, ADDLW, SUBLW, SUBWF.
bit 1: **DC** (Digit Carry) DC Transfer
Bit afectat de operaţiile de adunare, scădere şi transfer. Spre deosebire de bitul C, acest bit reprezintă transferul din al patrulea loc rezultat. Este setat de adunare când se întâmplă un transport de la bitul 3 la bitul 4, sau de scădere când se întâmplă împrumut de la bitul 4 la bitul 3, sau de transfer în ambele direcţii.
1= transfer produs la al patrulea bit conform cu ordinea, al rezultatului.
0= transferul nu s-a produs
Bitul DC este afectat de instrucţiunile ADDWF, ADDLW, SUBLW, SUBWF.
bit 2: **Z** (Zero bit) Indicarea unui rezultat zero.
Acest bit este setat când rezultatul unei operaţii aritmetice sau logice executate este zero.
1= rezultatul egal cu zero
0= rezultatul nu este egal cu zero
bit 3: **PD** (Power-down bit)
Bit ce este setat când microcontrolerul este alimentat atunci când începe să funcţioneze, după fiecare reset obişnuit şi după executarea instrucţiunii CLRWDT. Instrucţiunea SLEEP îl resetează când microcontrolerul intră în regimul consum/uzaj redus. Setarea lui repetată este posibilă prin reset sau prin pornirea sau oprirea sursei. Starea poate fi triggerată de asemenea de un semnal la pinul RB0/INT, de o schimbare la portul RB, de terminarea scrierii în EEPROM-ul de date intern, şi de watchdog de asemenea.
1= după ce sursa a fost pornită
0= executarea instrucţiunii SLEEP
bit 4: **TO** Time-out ; depăşirea-overflow watchdog-ului.
Bitul este setat după pornirea sursei şi executarea instrucţiunilor CLRWDT şi SLEEP. Bitul este resetat când watchdog-ul ajunge la sfârşit semnalând că ceva nu este în ordine.
1= depăşirea-oveflow nu s-a produs
0= depăşirea-overflow s-a produs
bit6,5: **RP1:RP0** (Register Bank Select bits-Biţi de Selectare a Bancului de Regiştri)
Aceşti doi biţi sunt partea superioară a adresei la adresarea directă. Pentru că instrucţiunile ce adresează memoria direct au doar şapte biţi, ei au nevoie doar de încă un bit pentru a adresa cei 256 bytes adică câţi are PIC16F84. Bitul RP1 nu este folosit, dar este lăsat pentru expansiuni viitoare ale acestui microcntroler.
01= primul banc
00= bancul zero
bit 7: **IRP** (Register Bank Select bit-Bit de Selectare a Bancului de Regiştri)
Bit al cărui rol este de a fi al optulea bit la adresarea indirectă a RAM-ului intern.
1= bancul 2 şi 3
0= bancul 0 şi 1 (de la 00h la FFh)

Registrul STATUS conţine starea aritmetică ALU (C, DC, Z), starea RESET (TO, PD) şi biţii pentru selectarea bancului de memorie (IRP, RP1, RP0). Considerând că selecţia bancului de memorie este controlată prin acest registru, el trebuie să fie prezent în fiecare banc.

Registrul OPTION

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0
bit7							

Legendă:
R = Bit de cititire **W** = Bit de scriere
U = Bit neinplementat, citit ca '0' - n = Valoarea la resetul power-on

***bit 0,2*: PS0, PS1, PS2** (Prescaler Rate Select bit-Bit Selecţie Rată Prescaler) Aceşti trei biţi definesc bitul de selecţie a ratei prescalerului. Ce este un prescaler şi cum pot afecta aceşti biţi funcţionarea unui microcontroler va fi explicat în secţiunea depre TMRO.

***bit 3:* PSA** (Prescaler Assignment bit-Bit de Asignare Prescaler)
Bit ce asignează prescalerul între TMRO şi watchdog.
1= prescalerul este asignat watchdogului
0= prescalerul este asignat timer-ului liber (ree-run) TMRO

Bits	TMR0	WDT
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

***bit 4:* T0SE** (TMR0 Source Edge Select bit-Bit Selecţie a Frontului Sursei TMR0) Dacă este permis de a se triggera TMRO prin impulsurile de la pinul RA4/T0CKI, acest bit determină dacă aceasta va fi la frontul descrescător sau crescător al unui semnal.
1= front crescător
0= front descrescător

***bit 5:* TOCS** (TMR0 Clock Source Select bit-Bit Selecţie Sursă Ceas TMR0)
Acest pin permite timerului liber (free-run) să incrementeze starea lui fie de la oscilatorul intern la fiecare ¼ a ceasului oscilatorului, fie prin impulsuri externe la pinul RA4/T0CKI.
1= impulsuri externe
0= ceas intern 1/4

***bit 6:* INTEDG** (Interrupt Edge Select bit-Bit de Selecţie a Frontului Întrerupere)
Dacă întreruperea este activată este posibil ca acest bit să determine frontul la care o întrerupere va fi activată la pinul RB0/INT.
1= front crescător
0= front descrescător

***bit 7:* RBPU** (PORTB Pull-up Enable bit-Bit Enable-Activare Pull-up PORTB)
Acest bit porneşte şi opreşte rezistorii interni 'pull-up'-scoatere la portul B.
1= Rezistori oprire "pull-up"
0= Rezistori pornire "pull-up"

Registrul INTCON

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GIE	EEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF
bit7							

Legendă
R = Bit de cititire **W** = bit de scriere
U = Bit neinplementat, citit ca '0' **–n** = Valoarea la resetul power-on

***bit 0:* RBIF** (RB Port Change Interrupt Flag bit-bit Steguleţ de Întrerupere a Schimbării Portului RB) Bit ce informează despre schimbările de la pinii 4, 5, 6 şi 7 ai portului B.
1=cel puţin un pin şi-a schimbat starea
0=nu s-a întâmplat nici o schimbare la vreun pin.

***bit 1:* INTF** (INT External Interrupt Flag bit-bit Steguleţ de Întrerupere Externă INT) A avut loc o întrerupere externă.
1=a avut loc o întrerupere
0=nu a avut loc o întrerupere
Dacă s-a detectat un front crescător sau descrescător la pinul RB0/INT, (ce este definit cu bitul INTEDG în registrul OPTION), bitul INTF este setat. Bitul trebuie să fie şters în subprogramul întrerupere pentru a detecta următoarea întrerupere.

***bit 2:* T0IF** (TMR0 Overflow Interrupt Flag bit-bit Steguleţ Depăşire Întrerupere TMRO) Depăşirea contorului TMRO.
1=contorul şi-a schimbat starea de la FFh la 00h.
0=depăşirea nu a avut loc

Bitul trebuie să fie şters în program pentru ca o întrerupere să fie detectată.
bit 3: **RBIE** (RB port change Interrupt Enable bit-bit Permite Întreruperea schimbării portului RB) Permite să aibă loc întreruperi la schimbarea stării pinilor 4, 5, 6, şi 7 ai portului B.
1=permite întreruperi la schimbarea stării
0=întreruperi interzise la schimbarea stării
Dacă RBIE şi RBIF au fost simultan setate, va avea loc o întrerupere.
bit 4: **INTE** (INT External Interrupt Enable bit-bit Permite Întrerupere externă INT) Bit ce permite întreruperea externă de la pinul RB0/INT.
1=întrerupere externă permisă
0=întrerupere externă interzisă
Dacă INTE şi INTF au fost setate simultan, va avea loc o întrerupere.
bit 5: **T0IE** (TMR0 Overflow Interrupt Enable bit-bit Permite Depăşire Întrerupere TMRO) Bit ce permite întreruperile în timpul depăşirii contorului TMRO.
1=întrerupere permisă
0=întrerupere interzisă
Dacă T0IE şi T0IF au fost simultan setate, va avea loc întreruperea.
Bit 6: **EEIE** (EEPROM Write Complete Interrupt Enable bit-bit Permite Întrerupere Completă a Scrierii EEPROM) Bit ce permite o întrerupere la sfârşitul unei rutine de scriere în EEPROM
1= întrerupere permisă
0= întrerupere interzisă
Dacă EEIE şi EEIF (ce este în registrul EECON1) au fost simultan setate, va avea loc o întrerupere.

PIC16F84 are patru surse de întrerupere:

1. Terminarea scrierii datelor în EEPROM
2. Întrerupere TMR0 cauzată de depăşirea timer-ului
3. Întrerupere în timpul schimbării la pinii RB4, RB5, RB6 şi RB7 ai portului B.
4. Întrerupere Externă de la pinul RB0/INT al microcontrolerului

Bit 7: **GIE** (Global Interrupt Enable bit-bit Permite Întrerupere Globală) Bit ce permite sau interzice toate întreruperile.
1=toate întreruperile sunt permise
0=toate întreruperile sunt interzise

Pachetul de programe MPLAB IDE

MPLAB IDE - an Integrated Development Environment (mediu integrat de dezvoltare), este un program complex pus la dispoziţie de firma **MICROCHIP TECHNOLOGY INC.** El conţine următoarele programe:
MPLAB Project Manager
MPLAB Editor
MPLAB ICD In-Circuit Debugger
MPLAB SIM Simulator
MPLAB ICE 2000 In-Circuit Emulator
MPASM - Assembler
MPLAB Cxx C Compilers
PRO MATE II and PICSTART Plus Programmers
PICMASTER and PICMASTER CE Emulators

Din cele 9 „unelte" puse la dispoziţie de complexul program MPLAB, ne vom opri doar la 4 dintre acestea, suficient cât să facem un proiect, să editäm un program, apoi să-l asamblăm (obţinând codul hex) şi nu în cele din urmă să simulăm acest program ca să observăm cum se va comporta microcontrolerul după ce este programat (înscrierea codului hex în memoria program) cu ajutorul unui programator.

În cele ce urmează voi prezenta o cale cât mai simplă şi rapidă de utilizare a acestui program.
Start MPLAB IDE. Din bara „MENU", selectează „Option" iar aici selectează „Development Mode". În acest moment va apărea o fereastră de dialog. Pentru început, execută următoarele setări:
selectează MPLAB-SIM Simulator alege microcontrolerul PIC16F84 aplică APPLY, apoi OK.

Până în acest moment ţi-ai ales microcontrolerul şi modul de utilizare al programului → simulator.
Acum va trebui să creezi un proiect, ce va include apoi programul scris în limbaj de asamblare. Un proiect poate include mai multe programe (scrise în limbaj de asamblare sau în C) putându-se face legătura între ele. Tot în cadrul acestui proiect va fi creat şi codul hex (după asamblare). Selectează File > New , şi în acest moment apare o

fereastră de dialog. Selectează YES pentru a crea un nou proiect. În acest moment apare o nouă fereastră de dialog, unde se cere numele proiectului nou şi destinaţia acestuia. După ce s-au completat, selectaţi OK.
Atenţie: daţi nume proiectului cu extensia „.pjt".
După ce a-ţi apăsat OK, apare fereastra de dialog „EDIT PROJECT". În josul ferestrei de dialog este un fişier: numele proiectului pe care l-aţi dat, dar cu extensia „.hex" → acesta fiind numele viitorului cod hex ce se obţine după asamblarea programului. Da-ţi clik pe acesta şi în acest moment se va activa butonul „Node Properties". Aplicaţi acest buton. Acum selectaţi tipul de fişier hex pe care îl doriţi; apoi apăsaţi OK. În acest moment a-ţi revenit la fereastra de dialog „EDIT PROJECT". Dacă NU aveţi scris deja fişierul în limbaj de asamblare, ieşiţi din această fereastră apăsând OK. Dacă aveţi deja scris fişierul în limbaj de asamblare (sau în C), aplicaţi „ADD NODE" şi aduceţi fişierul (cu extensia „.asm") de unde îl aveţi memorat pe hard disk. Atenţie: fişierul cu extensia „.asm" pe care îl aduceţi, trebuie să aibă acelaşi nume cu numele proiectului în care vă aflaţi. Selectaţi şi apoi aplicaţi OK. Iarăşi a-ţi revenit în fereastra „EDIT PROJECT". Aplicaţi OK. Dacă aveţi scris fişierul cu extensia „.asm" puteţi trece la asamblarea acestuia, dar dacă nu, atunci:
Fişierul „UNTILED1" în care vă aflaţi, salvaţi-l cu FILE > SAVE AS, la „FILE NAME" daţi nume fişierului păstrând extensia „.asm", apoi aplicaţi OK pentru a părăsi fereastra. Acum va trebui să ataşaţi noul fişier salvat, la proiectul în care vă aflaţi: PROJECT > EDIT PROJECT , şi iarăşi a-ţi revenit în fereastra " EDIT PROJECT". Aplicaţi butonul „ADD NODE". Din noua fereastră ce apare, selectaţi fişierul pe care l-aţi salvat, apoi OK → OK.
În acest moment puteţi trece la editarea programului. După ce l-aţi scris, SALVAŢI !, şi apoi PROJECT > MAKE PROJECT (sau apăsaţi tasta F10). Acum are loc asamblarea programului pe care l-aţi editat. Dacă a apărut vreo eroare, daţi clik pe mesajul de eroare, iar mouse – ul se va duce automat la linia din program unde există eroarea. Dacă nu aveţi nici o eroare, trebuie să scrie: „build completed successfully". Puteţi închide această nouă fereastră apărută. Din acest moment puteţi trece la SIMULAREA programului pe care l-aţi editat. Pentru uşurinţa urmăririi simulării aduceţi ferestrele: ROM, RAM şi SFR. Plasaţi-le pe desktop astfel încât să vă vină uşor să le urmăriţi pe toate simultan. Ajutându-vă de butoanele din TOOLBAR puteţi realiza simularea pas cu pas sau în mod automat.

Exemplu de adresare indirectă
Se cere ştergerea a 16 locaţii de memorie RAM folosind adresarea indirectă.

```
movlw 0x20      ;încarcă valoarea 20H în Acumulator
mowf FSR        ;încarcă 20H în registru FSR
start   clrf INDF       ;când ştergi INDF, defapt ştergi ceea ce se află la adresa conţinută în FSR
incf FSR;incrementând FSR, selectezi următoarea locaţie (octet) pentru a fi şters
btfss FSR,4     ;am şters 16 locaţii de memorie ?
goto start      ;sari la eticheta start
```

Observaţi că la etichete nu mai este obligatoriu să fie urmate de „:".

Aplicaţie:

În continuare este prezentat un program scris în limbaj de asamblare pentru microcontrolerele PIC (în partea din stânga a tabelului), folosit pentru aprinderea şi stingerea unui LED conectat la PORTB – cel mai nesemnificativ bit, iar în partea din dreapta, este prezentată o aplicaţie asemănătoare , dar scrisă în limbaj de asamblare pentru familia de microcontrolere 8051.

Programul va fi editat, asamblat şi simulat cu ajutorul pachetului de programe MPLAB.

```
name_test_11
    #include   <p16F84.inc>       ;foloseste „trasaturile”
                                  ;microcontrolerului pic16F84

Start
   BSF 03,5       ;seteaza bitul 5 dela adresa 03H , adica pe RP0
   MOVLW 00h      ;muta 00 in Acumulator
   MOVWF 0x05     ;muta din acumulator in registrul TRISA
   MOVWF 06h      ;muta din acumulator in registrul TRISB
   BCF 03,5       ;reseteaza bitul 5 dela adresa 03H , adica pe
                    ;RP0
   MOVLW 02h      ;muta 02 in Acumulator
   MOVWF 05h      ;muta in portul A valoarea din Acumulator

ionel
   MOVLW 01h      ;muta 01 in Acumulator
   MOVWF 06h      ;muta in PORTB valoarea din Acumulator;
                  ;aprinde LED-ul conectat la LSB din PORTB
   CALL Delay     ;apeleaza o intarziere
   MOVLW 00       ;muta 00 in Acumulator
   MOVWF 06h      ;stinge LED-ul prin mutarea in PORTB a valorii
                  ;00 din Acumulator
   CALL Delay     ;apeleaza o intarziere
   GOTO ionel     ;du-te la eticheta ionel

Delay
   MOVLW .13       ;muta constanta ZECIMALA 13, in
                   ;Acumulator
   MOVWF 1Ah       ;muta din Acumulator la adresa 1A hexa
Delay1
   DECFSZ 1Bh,1    ;daca pana acuma nu am lucrat deloc cu locatia
                   ;1BH,
   GOTO Delay1     ; inseamna ca ea are valoarea FFH, si de aici se
                   ;decrementeaza ...
    DECFSZ 1Ch,1
   GOTO Delay1
   DECFSZ 1Ah,1
   GOTO Delay1
   RETURN

END                ;sfarsit de program
```

```
name test_delay     ;nume program
org 0000H           ;inscrie programul in memoria program
                    ;incepand de la adresa 0000H
LED EQU P3.4        ;eticheteaza pinul de port P3.4
START:
          CLR LED  ;"stinge LED-ul"
          CALL DELAY1       ;apeleaza o interziere mai mare
          SETB LED          ;"aprinde LED - ul"
          CALL DELAY2       ;apeleaza o interziere mai mica
          JMP START         ;repeta

DELAY1:           ;subrutina de intarziere mare
          MOV R7,#6
          MOV R6,#100
          MOV R5,#100
DELAY10:
          DJNZ R5,DELAY10
          DJNZ R6,DELAY10
          DJNZ R7,DELAY10
          RET               ;revenire din apelare

DELAY2:           ;subrutina de intarziere mai mica
          MOV R7,#1
          MOV R6,#100
          MOV R5,#100
DELAY20:
          DJNZ R5,DELAY20
          DJNZ R6,DELAY20
          DJNZ R7,DELAY20
          RET               ;revenire din apelare
END                         ;sfarsit de program
```