Microcontrolere PIC on-line

GRATIS!

Pagina anterioar•font> | Conþinut | Pagina urm•are

Author: Nebojsa Matic

Scopul acestei c•i nu este de a face din dvs. un expert •microcontrolere, ci unul care are r•unsuri tehnice la unele •reb•.

All credits for translation goes to **Cristian Secrieru** Romanian editor**.**

Trimiteþi un e-mail unui prieten despre acest articol

Sisteme de dezvoltare

C O N •I N U T

CAPITOLUL 1 Introducere •Microcontrolere

Introducere
Istorie
Microcontrolere contra microprocesoare

1.1 Unitatea de memorie
1.2 Unitatea de procesare
1.3 Bus-ul
1.4 Unitatea intrare-ie°ire
1.5 Comunicaþie serial•BR>1.6 Unitatea de timer
1.7 Watchdog-ul
1.8 Convertorul Analog-Digital
1.9 Programul

CAPITOLUL 3 Set Instructiuni

Introducere

3.1 Set de instructiuni •familia microcontrolerului PIC16Cxx
3.2 Transfer Date
3.3 Aritmetica si logica
3.4 Operatii cu biti
3.5 Directionarea debitului de program
3.6 Perioada de executie a instructiunilor
3.7 Lista de cuvinte

CAPITOLUL 5 MPLAB

Introducere

5.1 Instalarea pachetului de program MPLAB
5.2 Introducere • MPLAB
5.3 Alegerea modului de dezvoltare
5.4 Conceperea unui proiect
5.5 Proiectarea unui fisier de asamblare
5.6 Scrierea unui program
5.7 Simulator MPSIM
5.8 Toolbar

Anexa A Set Instrucþiuni

CAPITOLUL 2 Microcontrolerul PIC16F84

Introducere

CISC, RISC
Aplicatii
Ciclu de clock/instructiune
Pipelining
Semnificatia pinilor

2.1 Generator-oscilator de ceas
2.2 Reset
2.3 Unitatea de procesare centrala
2.4 Porturi
2.5 Organizarea memoriei
2.6 •treruperi
2.7 Timer-ul liber TMRO
2.8 Memoria de date EEPROM

CAPITOLUL 4 Programare •Limbaj de Asamblare

Introducere
Un exemplu de program scris
Directive de control
Fisiere create ca rezultat al translarii de program
Macro-uri

CAPITOLUL 6 Mostrele

Introducere

6.1 Alimentarea microcontrolerului
6.2 Macrouri folosite •programe

- Macrourile WAIT, WAITX
- Macroul PRINT

6.3 Exemple

- Light emitting diodes – LEDuri
- Tastatura

Anexa B Sisteme numerice

Introducere

B.1 Sistem numeric zecimal
B.2 Sistem numeric binar
B.3 Sistem numeric hexazecimal

Concluzie

Anexa C Glosar

- Optocuploare
  - Izolarea galvanic• liniilor de intrare folosind optocuploare
  - Izolarea galvanic• liniilor de ieºire folosind optocuploare
- Relee
- Generarea unui sunet
- Regiºtri de deplasare
  - Registru de deplasare de intrare
  - Registru de deplasare de ieºire
- Afiºoare 7–segmente (multiplexare)
- Afiºor LCD
- Convertor AD pe 12 biþi
- Comunicaþia serial•SPAN>

Trimiteþi-ne un comentariu despre carte

Subiect :
Comentariu despre cartea Microcontrolere PI
Nume :
Tara :
Romania
E-mail :

Mesajul tau:

Submit Reset

Microcontrolere PIC on-line

GRATIS!

Author: Nebojsa Matic

Scopul acestei c•i nu este de a face din dvs. un expert •microcontrolere, ci unul care are r•unsuri tehnice la unele •reb•.

All credits for translation goes to **Cristian Secrieru** Romanian editor**.**

Trimiteþi un e-mail unui prieten despre acest articol

Sisteme de dezvoltare

C O N •I N U T

Anexa B Sisteme numerice

Introducere

B.1 Sistem numeric zecimal
B.2 Sistem numeric binar
B.3 Sistem numeric hexazecimal

Concluzie

Anexa C Glosar

- Optocuploare
  - Izolarea galvanic• liniilor de intrare folosind optocuploare
  - Izolarea galvanic• liniilor de ieºire folosind optocuploare
- Relee
- Generarea unui sunet
- Regiºtri de deplasare
  - Registru de deplasare de intrare
  - Registru de deplasare de ieºire
- Afiºoare 7–segmente (multiplexare)
- Afiºor LCD
- Convertor AD pe 12 biþi
- Comunicaþia serial•SPAN>

Trimiteþi-ne un comentariu despre carte

Subiect :
Comentariu despre cartea Microcontrolere PI
Nume :
Tara :
Romania
E-mail :

Mesajul tau:

Submit Reset

CAPITOLUL 1

Introducere în Microcontrolere

Introducere

Circumstan•ele în care ne g•sim ast•zi în domeniul microcontrolerelor •i-au avut începuturile în dezvoltarea tehnologiei circuitelor integrate. Aceast• dezvoltare a f•cut posibil• înmagazinarea a sute de mii de tranzistoare într-un singur cip. Aceasta a fost o premiz• pentru produc•ia de microprocesoare, •i primele calculatoare au fost f•cute prin ad•ugarea perifericelor ca memorie, linii intrare-ie•ire, timer-i •i altele. Urm•toarea cre•tere a volumului capsulei a dus la crearea circuitelor integrate. Aceste circuite integrate con•in atât procesorul cât •i perifericele. A•a s-a întâmplat cum primul cip con•inând un microcalculator, sau ce va deveni cunoscut mai târziu ca microcontroler a luat fiin••.

Istorie

Este anul 1969, •i o echip• de ingineri japonezi de la compania BUSICOM sosesc în Statele Unite cu cererea ca unele circuite integrate pentru calculatoare s• fie f•cute folosind proiectele lor. Propunerea a fost f•cut• c•tre INTEL, iar Marcian Hoff a fost desemnat responsabil cu acest proiect. Pentru c• el era cel ce avea experien•• în lucrul cu un calculator (PC) PDP8, i-a venit s• sugereze o solu•ie diferit• fundamental în locul construc•iei propuse. Aceast• solu•ie presupunea c• func•ionarea circuitului integrat este determinat• de un program memorat în el. Aceasta a însemnat c• configura•ia ar fi fost mult mai simpl•, dar aceasta ar fi cerut mult mai mult• memorie decât ar fi cerut proiectul propus de inginerii japonezi. Dup• un timp, cu toate c• inginerii japonezi au încercat s• caute o solu•ie mai simpl•, ideea lui Marcian a câ•tigat, •i a luat na•tere primul microprocesor. În transformarea unei idei într-un produs finit, Frederico Faggin a fost de un ajutor major pentru INTEL. El s-a transferat la INTEL, •i doar în 9 luni a reu•it s• scoat• un produs din prima sa concep•ie. INTEL a ob•inut drepturile de a vinde acest bloc integral în 1971. În primul rând ei au cump•rat licen•a de la compania BUSICOM care nu au avut idee ce comoar• avuseser•. În timpul acelui an a ap•rut pe pia•• un microprocesor numit 4004. Acela a fost primul microprocesor de 4 bi•i cu vitez• 6000 opera•ii pe secund•. Nu mult dup• aceea, compania american• CTC a cerut de la INTEL •i de la Texas Instruments s• fac• un microprocesor pe 8 bi•i pentru folosin•• în terminale. Cu toate c• CTC a renun•at la aceast• idee pân• la sfâr•it, INTEL •i Texas Instruments au continuat s• lucreze la microprocesor •i în aprilie 1972 a ap•rut pe pia•• primul microprocesor de 8 bi•i sub numele de 8008. Putea s• adreseze 16Kb de memorie •i avea 45 de instruc•iuni •i viteza de 300.000 de opera•ii pe secund•. Acel microprocesor a fost predecesorul tuturor microprocesoarelor de ast•zi. INTEL au continuat dezvolt•rile lor pân• în aprilie 1974 •i au lansat pe pia•• microprocesorul de 8 bi•i sub numele de 8080 ce putea adresa 64Kb de memorie •i avea 75 de instruc•iuni, iar pre•ul începuse de la 360$.

Într-o alt• companie american• Motorola, •i-au dat seama repede ce se întâmpla, a•a c• au lansat pe pia•• un microprocesor de 8 bi•i 6800. Constructor •ef era Chuck Peddle •i pe lâng• microprocesorul propriu-zis, Motorola a fost prima companie care s• fac• alte periferice ca 6820 •i 6850. La acel timp multe companii au recunoscut marea importan•• a microprocesoarelor •i au început propriile lor dezvolt•ri. Chuck Peddle p•r•se•te Motorola pentru a se muta la MOS Technology •i continu• s• lucreze intensiv la dezvoltarea microprocesoarelor.

La expozi•ia WESCON din Statele Unite din 1975 a avut loc un eveniment critic în istoria microprocesoarelor. MOS Technology a anun•at c• produce microprocesoarele 6501 •i 6502 la 25$ bucata pe care cump•r•torii le puteau

cump•ra imediat. Aceasta a fost atât de senza•ional încât au crezut c• este un fel de în•el•ciune, gândind c• competitorii vindeau 8080 •i 6800 la 179$. Ca un r•spuns la competitorii lor atât INTEL cât •i Motorola au sc•zut pre•urile lor în prima zi a expozi•iei pân• la 69.95$ pe microprocesor. Motorola intenteaz• repede proces contra lui MOS Technology •i contra lui Chuck Peddle pentru copierea protejatului 6800. MOS Technology înceteaz• de a mai produce 6501 dar continu• s• produc• 6502. 6502 este un microcontroler pe 8 bi•i cu 56 de instruc•iuni •i o capabilitate de adresare direct• de 64Kb de memorie. Datorit• costului sc•zut, 6502 devine foarte popular, a•a c• este instalat în calculatoare ca :KIM-1, Apple I, Apple II, Atari, Comodore, Acorn, Oric, Galeb, Orao, Ultra •i multe altele. Curând apar câ•iva produc•tori de 6502 (Rockwell, Sznertek, GTE, NCR, Ricoh •i Comodore preiau MOS Technology) ce era în momentul prosperit••ii sale vândut la o rat• de 15 milioane de microprocesoare pe an!

Al•ii totu•i nu au cedat. Federico Faggin p•r•se•te INTEL, •i î•i porne•te propria sa companie Zilog Inc.
În 1976 Zilog anun•• Z80. În timpul cre•rii acestui microprocesor, Faggin ia o decizie crucial•. •tiind c• un mare num•r de programe fuseser• dezvoltate pentru 8080, Faggin î•i d• seama c• mul•i vor r•mâne fideli acelui microprocesor din cauza marii cheltuieli care ar rezulta în urma refacerii tuturor programelor. Astfel el decide c• un nou microprocesor trebuie s• fie compatibil cu 8080, sau c• trebuie s• fie capabil s• execute toate programele care deja fusese scrise pentru 8080. În afar• acestor caracteristici, multe altele noi au fost ad•ugate, a•a c• Z80 a fost un microprocesor foarte puternic la vremea lui. Putea adresa direct 64Kb de memorie, avea 176 instruc•iuni, un num•r mare de registre, o op•iune incorporat• pentru reîmprosp•tarea memoriei RAM dinamice, o singur• surs•, vitez• de lucru mult mai mare etc. Z80 a fost un succes mare •i toat• lumea a f•cut conversia de 8080 la Z80. Se poate spune c• Z80 comercial, a fost f•r• nici o îndoial•, cel mai de succes micropocesor de 8 bi•i a acelui timp. În afar• de Zilog, al•i noi produc•tori apar de asemenea ca: Mostek, NEC, SHARP •i SGS. Z80 a fost inima a multor calculatoare ca: Spectrum, Partner, TRS703, Z-3.

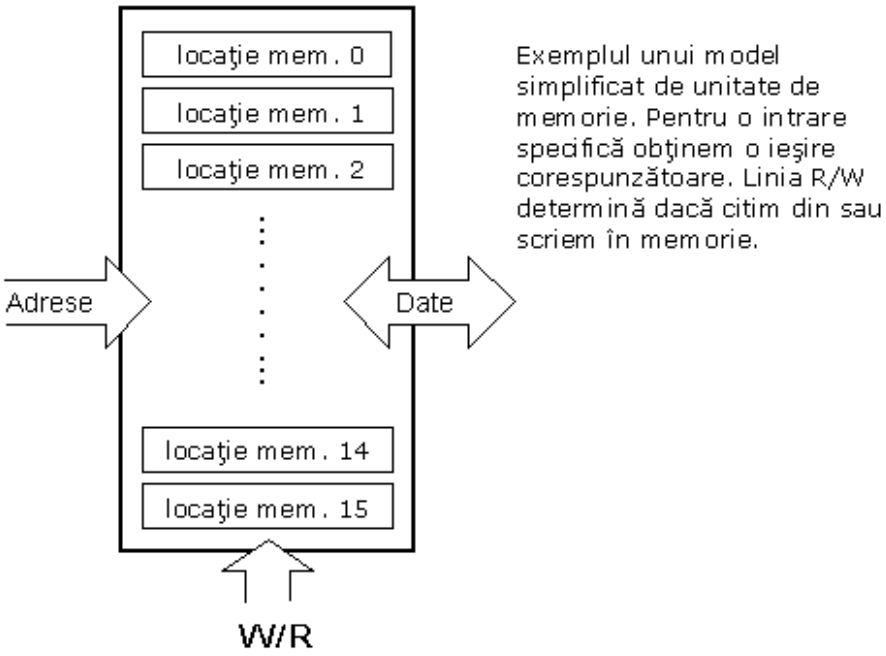
În 1976, INTEL iese pe pia•• cu o versiune îmbun•t••it• de microprocesor pe 8 bi•i numit 8085. Totu•i, Z80 era cu mult mai bun încât INTEL curând a pierdut b•t•lia. Chiar dac• au ap•rut pe pia•• înc• câteva microprocesoare (6809, 2650, SC/MP etc.), totul fusese de fapt deja hot•rât. Nu mai erau de f•cut îmbun•t••iri importante ca s•-i fac• pe produc•tori s• se converteasc• spre ceva nou, a•a c• 6502 •i Z80 împreun• cu 6800 au r•mas ca cei mai reprezentativi ai microprocesoarelor de 8 bi•i ai acelui timp.

Microcontrolere contra Microprocesoare

Microcontrolerul difer• de un microprocesor în multe feluri. În primul rând •i cel mai important este func•ionalitatea sa. Pentru a fi folosit, unui microprocesor trebuie s• i se adauge alte componente ca memorie, sau componente pentru primirea •i trimiterea de date. Pe scurt, aceasta înseamn• c• microprocesorul este inima calculatorului. Pe de alt• parte, microcontrolerul este proiectat s• fie toate acestea într-unul singur. Nu sunt necesare alte componente externe pentru aplicarea sa pentru c• toate perifericele necesare sunt deja incluse în el. Astfel, economisim timpul •i spa•iul necesare pentru construirea de aparate.

1.1 Unitatea de memorie

Memoria este o parte a microcontrolerului a c•rei func•ie este de a înmagazina date.
Cel mai u•or mod de a explica este de a-l descrie ca un dulap mare cu multe sertare. Dac• presupunem c• am marcat sertarele într-un asemenea fel încât s• nu fie confundate, oricare din con•inutul lor va fi atunci u•or accesibil. Este suficient s• se •tie desemnarea sertarului •i astfel con•inutul lui ne va fi cunoscut în mod sigur.

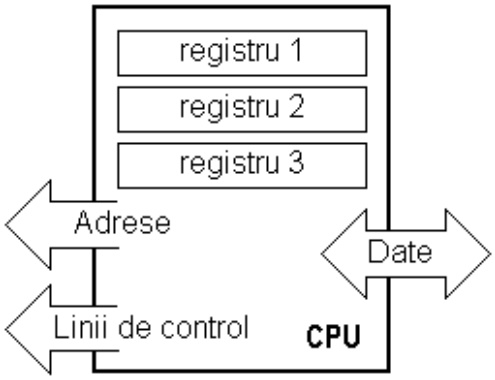


Componentele de memorie sunt exact a•a. Pentru o anumit• intrare ob•inem con•inutul unei anumite loca•ii de memorie adresate •i aceasta este totul. Dou• noi concepte ne sunt aduse: adresarea •i loca•ia de memorie. Memoria

const• din toate loca•iile de memorie, •i adresarea nu este altceva decât selectarea uneia din ele. Aceasta înseamn• c• noi trebuie s• select•m loca•ia de memorie la un cap•t, •i la cel•lalt cap•t trebuie s• a•tept•m con•inutul acelei loca•ii. În afar• de citirea dintr-o loca•ie de memorie, memoria trebuie de asemenea s• permit• scrierea în ea. Aceasta se face prin asigurarea unei linii adi•ionale numit• linie de control. Vom desemna aceast• linie ca R/W (cite•te /scrie). Linia de control este folosit• în urm•torul fel: dac• r/w=1, se face citirea, •i dac• opusul este adev•rat atunci se face scrierea în loca•ia de memorie. Memoria este primul element, dar avem nevoie •i de altele pentru ca microcontrolerul nostru s• func•ioneze.

1.2 Unitatea de procesare central•

S• ad•ug•m alte 3 loca•ii de memorie pentru un bloc specific ce va avea o capabilitate incorporat• de înmul•ire, împ•r•ire, sc•dere •i s•-i mut•m con•inutul dintr-o loca•ie de memorie în alta. Partea pe care tocmai am ad•ugat-o este numit• "unitatea de procesare central•" (CPU). Loca•iile ei de memorie sunt numite regi•tri.

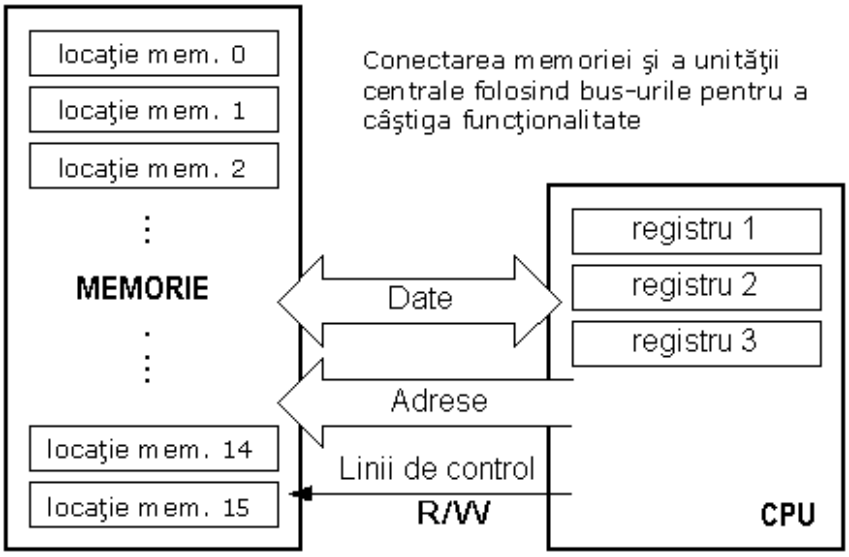


Exemplul unei de unităţi simplificate de procesare centrală cu trei regiştri

Regi•trii sunt deci loca•ii de memorie al c•ror rol este de a ajuta prin executarea a variate opera•ii matematice sau a altor opera•ii cu date oriunde se vor fi g•sit datele. S• privim la situa•ia curent•. Avem dou• entit••i independente (memoria •i CPU) ce sunt interconectate, •i astfel orice schimb de informa•ii este ascuns, ca •i func•ionalitatea sa. Dac•, de exemplu, dorim s• ad•ug•m con•inutul a dou• loca•ii de memorie •i întoarcem rezultatul înapoi în memorie, vom avea nevoie de o conexiune între memorie •i CPU. Mai simplu formulat, trebuie s• avem o anumit• "cale" prin care datele circul• de la un bloc la altul.

1.3 Bus-ul

Calea este numit• "bus"- magistral•. Fizic, el reprezint• un grup de 8, 16, sau mai multe fire. Sunt dou• tipuri de bus-uri: bus de adres• •i bus de date. Primul const• din atâtea linii cât este cantitatea de memorie ce dorim s• o adres•m, iar cel•lalt este atât de lat cât sunt datele, în cazul nostru 8 bi•i sau linia de conectare. Primul serve•te la transmiterea adreselor de la CPU la memorie, iar cel de al doilea la conectarea tuturor blocurilor din interiorul microcontrolerului.

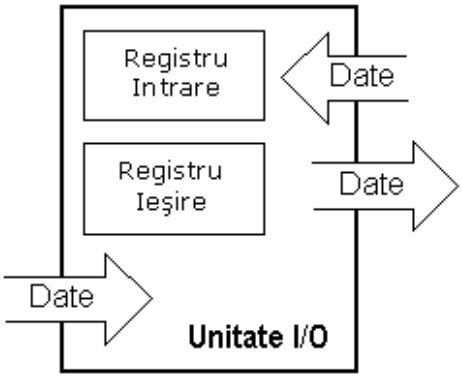


Conectarea memoriei şi a unităţii centrale folosind bus-urile pentru a câştiga funcţionalitate

În ceea ce prive•te func•ionalitatea, situa•ia s-a îmbun•t••it, dar o nou• problem• a ap•rut de asemenea: avem o unitate ce este capabil• s• lucreze singur•, dar ce nu are nici un contact cu lumea de afar•, sau cu noi! Pentru a înl•tura aceast• deficien••, s• ad•ug•m un bloc ce con•ine câteva loca•ii de memorie al c•ror singur cap•t este conectat la bus-ul de date, iar cel•lalt are conexiune cu liniile de ie•ire la microcontroler ce pot fi v•zute cu ochiul liber ca pini la componenta electronic•.

1.4 Unitatea intrare-ie•ire

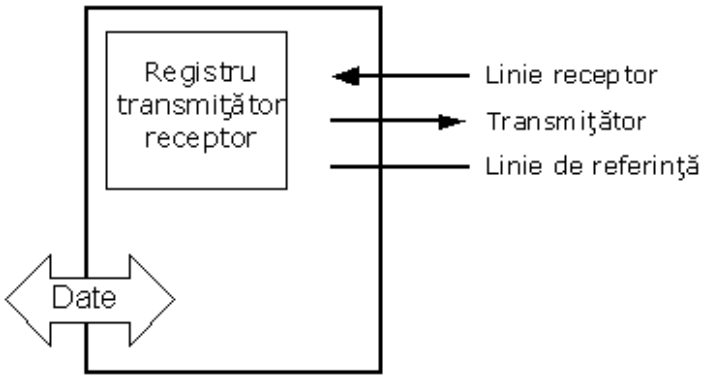
Aceste loca•ii ce tocmai le-am ad•ugat sunt numite "porturi". Sunt diferite tipuri de porturi: intrare, ie•ire sau porturi pe dou•-c•i. Când se lucreaz• cu porturi, mai întâi de toate este necesar s• se aleag• cu ce port urmeaz• s• se lucreze, •i apoi s• se trimit• date la, sau s• se ia date de la port.



Când se lucreaz• cu el portul se comport• ca o loca•ie de memorie. Ceva este pur •i simplu scris în sau citit din el, •i este posibil de a remarca u•or aceasta la pinii microcontrolerului.

1.5 Comunica•ia serial•

Cu aceasta am ad•ugat la unitatea deja existent• posibilitatea comunic•rii cu lumea de afar•. Totu•i, acest mod de comunicare are neajunsurile lui. Unul din neajunsurile de baz• este num•rul de linii ce trebuie s• fie folosite pentru a transfera datele. Ce s-ar întâmpla dac• acestea ar trebui transferate la distan•• de câ•iva kilometri? Num•rul de linii înmul•it cu num•rul de kilometri nu promite costuri eficiente pentru proiect. Nu ne r•mâne decât s• reducem num•rul de linii într-un a•a fel încât s• nu sc•dem func•ionalitatea. S• presupunem c• lucr•m doar cu 3 linii, •i c• o linie este folosit• pentru trimiterea de date, alta pentru recep•ie •i a treia este folosit• ca o linie de referin•• atât pentru partea de intrare cât •i pentru partea de ie•ire. Pentru ca aceasta s• func•ioneze, trebuie s• stabilim regulile de schimb ale datelor. Aceste reguli sunt numite protocol. Protocolul este de aceea definit în avans ca s• nu fie nici o neîn•elegere între p•r•ile ce comunic• una cu alta. De exemplu, dac• un om vorbe•te în francez•, •i altul vorbe•te în englez•, este pu•in probabil c• ei se vor în•elege repede •i eficient unul cu altul. S• presupunem c• avem urm•torul protocol. Unitatea logic• "1" este setat• pe linia de transmisie pân• ce începe transferul. Odat• ce începe transferul, coborâm linia de transmisie la "0" logic pentru o perioad• de timp (pe care o vom desemna ca T), a•a c• partea receptoare va •ti c• sunt date de primit, a•a c• va activa mecanismul ei de recep•ie. S• ne întoarcem acum la partea de transmisie •i s• începem s• punem zero-uri •i unu-uri pe linia de transmisie în ordinea de la un bit a celei mai de jos valori la un bit a celei mai de sus valori. S• l•s•m ca fiecare bit s• r•mân• pe linie pentru o perioad• de timp egal• cu T, •i la sfâr•it, sau dup• al 8-lea bit, s• aducem unitatea logic• "1" înapoi pe linie ce va marca sfâr•itul transmisiei unei date. Protocolul ce tocmai l-am descris este numit în literatura profesional• NRZ (Non-Return to Zero).

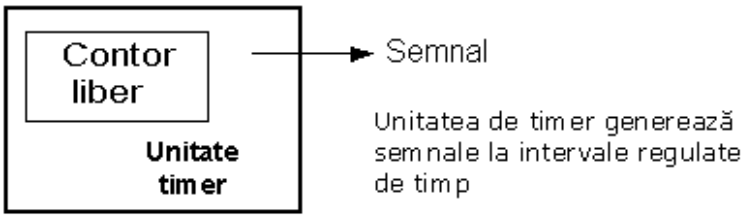


Unitatea serial• folosit• pentru a trimite date, dar numai prin trei linii

Pentru c• avem linii separate de recep•ie •i de transmitere, este posibil s• recep•ion•m •i s• transmitem date (informa•ii) în acela•i timp. Blocul a•a numit full-duplex mode ce permite acest mod de comunicare este numit blocul de comunicare serial•. Spre deosebire de transmisia paralel•, datele sunt mutate aici bit cu bit, sau într-o serie de bi•i, de unde vine •i numele de comunica•ie serial•. Dup• recep•ia de date trebuie s• le citim din loca•ia de transmisie •i s• le înmagazin•m în memorie în mod opus transmiterii unde procesul este invers. Datele circul• din memorie prin bus c•tre loca•ia de trimitere, •i de acolo c•tre unitatea de recep•ie conform protocolului.

1.6 Unitatea timer

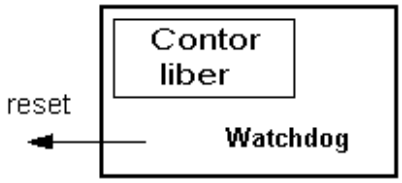
Acum c• avem comunica•ia serial•, putem recep•iona, trimite •i procesa date.



Totu•i, pentru noi ca s• putem s• îl folosim în industrie mai avem nevoie de câteva blocuri. Unul din acestea este blocul timer care este important pentru noi pentru c• ne d• informa•ia de timp, durat•, protocol etc. Unitatea de baz• a timer-ului este un contor liber (free-run) care este de fapt un registru a c•rui valoare numeric• cre•te cu unu la intervale egale, a•a încât luându-i valoarea dup• intervalele T1 •i T2 •i pe baza diferen•ei lor s• putem determina cât timp a trecut. Acesta este o parte foarte important• a microcontrolerului al c•rui control cere cea mai mare parte a timpului nostru.

1.7 Watchdog-ul

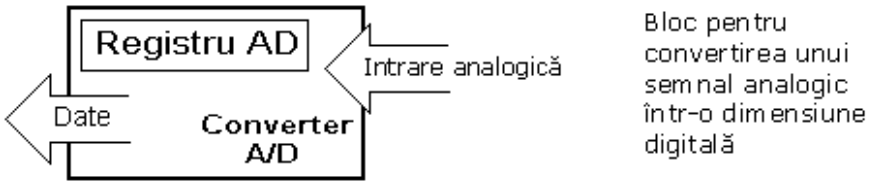
Înc• un lucru ce necesit• aten•ia noastr• este func•ionarea f•r• defecte a microcontrolerului în timpul func•ion•rii. S• presupunem c• urmare a unei anumite interferen•e (ce adesea se întâmpl• în industrie) microcontrolerul nostru se opre•te din executarea programului, sau •i mai r•u, începe s• func•ioneze incorect.



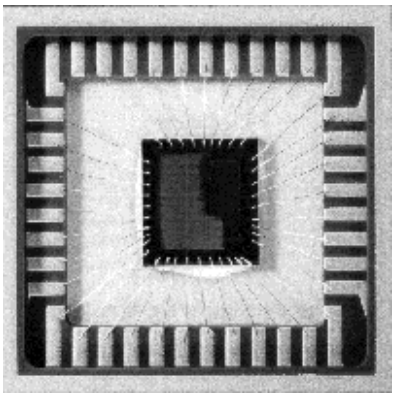
Bineîn•eles, când aceasta se întâmpl• cu un calculator, îl reset•m pur •i simplu •i va continua s• lucreze. Totu•i, nu exist• buton de resetare pe care s•-l ap•s•m în cazul microcontrolerului care s• rezolve astfel problema noastr•. Pentru a dep••i acest obstacol, avem nevoie de a introduce înc• un bloc numit watchdog-câinele de paz•. Acest bloc este de fapt un alt contor liber (free-run) unde programul nostru trebuie s• scrie un zero ori de câte ori se execut• corect. În caz c• programul se "în•epene•te", nu se va mai scrie zero, iar contorul se va reseta singur la atingerea valorii sale maxime. Aceasta va duce la rularea programului din nou, •i corect de aceast• dat• pe toat• durata. Acesta este un element important al fiec•rui program ce trebuie s• fie fiabil f•r• supravegherea omului.

1.8 Convertorul Analog-Digital

Pentru c• semnalele de la periferice sunt substan•ial diferite de cele pe care le poate în•elege microcontrolerul (zero •i unu), ele trebuie convertite într-un mod care s• fie în•eles de microcontroler. Aceast• sarcin• este îndeplinit• de un bloc pentru conversia analog-digital• sau de un convertor AD. Acest bloc este responsabil pentru convertirea unei informa•ii despre o anumit• valoare analogic• într-un num•r binar •i pentru a o urm•ri pe tot parcursul la un bloc CPU a•a ca blocul CPU s• o poat• procesa.

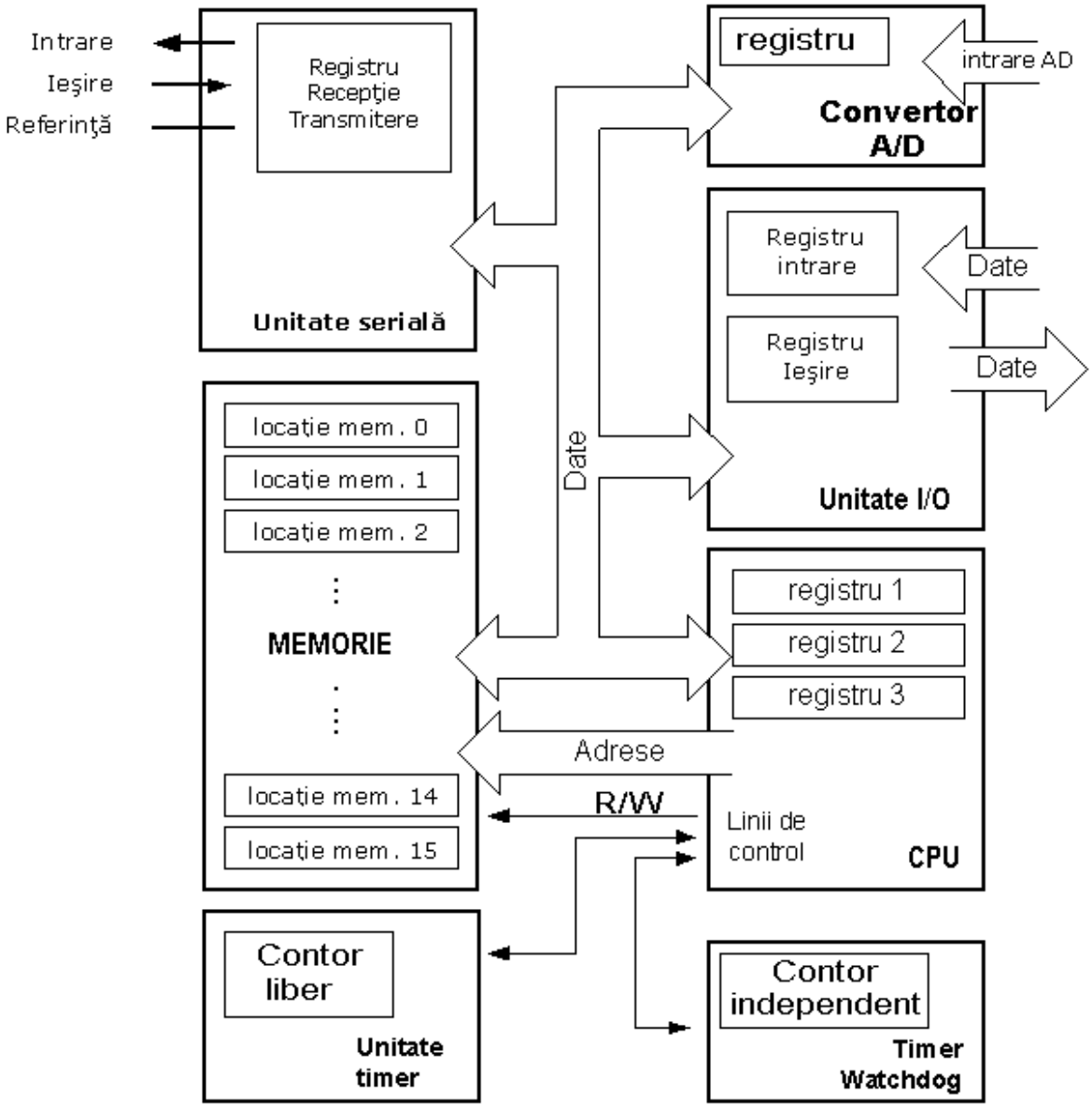


Astfel microcontrolerul este acum terminat, •i tot ce mai r•mâne de f•cut este de a-l pune într-o component• electronic• unde va accesa blocurile interioare prin pinii exteriori. Imaginea de mai jos arat• cum arat• un microcontroler în interior.



Configura•ia fizic• a interiorului unui microcontroler

Liniile sub•iri ce merg din interior c•tre p•r•ile laterale ale microcontrolerului reprezint• fire conectând blocurile interioare cu pinii capsulei microcontrolerului. Schema urm•toare reprezint• sec•iunea central• a microcontrolerului.



Schiţa microcontrolerului cu elementele lui de bază şi conexiunile sale interne

Pentru o aplica•ie real•, un microcontroler singur nu este de ajuns. În afar• de microcontroler, avem nevoie de un program pe care s•-l execute, •i alte câteva elemente ce constituie o interfa•• logic• c•tre elementele de stabilizare (ce se va discuta în capitolele urm•toare).

1.9 Programul

Scrierea programului este un domeniu special de lucru al microcontolerului •i este denumit "programare". S• încerc•m s• scriem un mic program ce îl vom crea singuri •i pe care oricine va fi în stare s•-l în•eleag•.

```
START
REGISTER1=MEMORY LOCATION_A
REGISTER2=MEMORY LOCATION_B
PORTA=REGISTER1 + REGISTER2

END
```

Programul adun• con•inutul a dou• loca•ii de memorie, •i vede suma lor la portul A. Prima linie a programului este pentru mutarea con•inutul loca•iei de memorie "A" într-unul din regi•tri unit••ii de procesare centrale. Pentru c• avem nevoie •i de celelalte date de asemenea, le vom muta de asemenea în cel•lalt registru al unit••ii de procesare centrale. Urm•toarea instruc•iune instruie•te unitatea de procesare central• s• adune con•inutul celor doi regi•tri s• trimit• rezultatul ob•inut la portul A, încât suma acestei adun•ri s• fie vizibil• pentru toat• lumea de afar•. Pentru o problem• mai complex•, programul care s• lucreze la rezolvarea ei va fi mai mare.

Programarea poate fi f•cut• în câteva limbaje ca Assembler, C •i Basic care sunt cele mai folosite limbaje. Assembler apar•ine limbajelor de nivel sc•zut ce sunt programate lent, dar folosesc cel mai mic spa•iu în memorie •i d• cele mai bune rezultate când se are în vedere viteza de execu•ie a programului. Pentru c• este cel mai folosit limbaj în programarea microcontrolerelor va fi discutat într-un capitol ulterior. Programele în limbajul C sunt mai u•or de scris, mai u•or de în•eles, dar sunt mai lente în executare decât programele în Assembler. Basic este cel mai u•or de înv••at, •i instruc•iunile sale sunt cele mai aproape de modul de gândire a omului, dar ca •i limbajul de programare C este de asemenea mai lent decât Assembler-ul. În orice caz, înainte de a v• hot•rî în privin•a unuia din aceste limbaje trebuie s• studia•i cu aten•ie cerin•ele privind viteza de execu•ie, m•rimea memoriei •i timpul disponibil pentru asamblarea sa.

Dup• ce este scris programul, trebuie s• instal•m microcontrolerul într-un aparat •i s•-l l•s•m s• lucreze. Pentru a face aceasta trebuie s• ad•ug•m câteva componente externe necesare pentru func•ionarea sa. Mai întâi trebuie s• d•m via•• microcontrolerului prin conectarea sa la o surs• (tensiune necesar• pentru operarea tuturor instrumentelor electronice) •i oscilatorului al c•rui rol este similar inimii din corpul uman. Bazat pe ceasul s•u microcontrolerul execut• instruc•iunile programului. Îndat• ce este alimentat microcontrolerul va executa un scurt control asupra sa, se va uita la începutul programului •i va începe s•-l execute. Cum va lucra aparatul depinde de mul•i parametri, cel mai important fiind priceperea dezvoltatorului de hardware, •i de experien•a programatorului în ob•inerea maximului din aparat cu programul s•u.

Pagina anterioar•	Con•inut	Pagina urm•toare

CAPITOLUL 2

Microcontrolerul PIC16F84

Introducere

PIC16F84 aparþine unei clase de microcontrolere de 8 biþi cu arhitectur•ISC. Structura lui general•ste ar•t•n schiþa urm•are reprezent• blocurile de baz•

Memoria program (FLASH)-pentru memorarea unui program scris.
Pentru c•emoria ce este f•t•n tehnologia FLASH poate fi programat• ºi ºtears•ai mult dec•odat•aceasta face microcontrolerul potrivit pentru dezvoltarea de component•

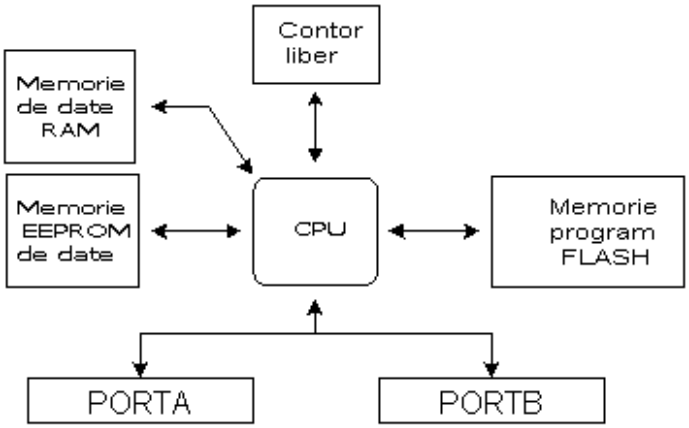
EEPROM-memorie de date ce trebuie s•ie salvate c• nu mai este alimentare.
Este •mod uzual folosit•entru memorarea de date importante ce nu trebuie pierdute dac•ursa de alimentare se •rerupe dintr-o dat•De exemplu, o astfel de dat•ste o temperatur•restabilit•n regulatoarele de temperatur•nbsp;Dac•n timpul •reruperii aliment•i aceast•at•e pierde, va trebui s•acem ajustarea •• dat•a revenirea aliment•i. Astfel componenta noastr•ierde •privinþa auto-menþinerii.

RAM-memorie de date folosit•e un program •timpul execut•i sale.
• RAM sunt memorate toate rezultatele intermediare sau datele temporare ce nu sunt cruciale la •reruperea sursei de alimentare.

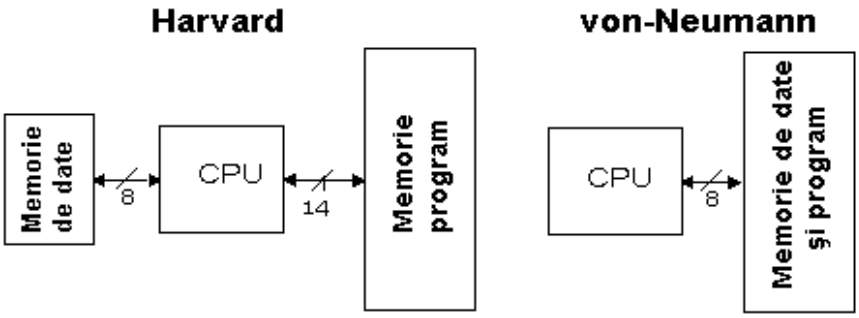
PORTUL A ºi PORTUL B sunt conexiuni fizice •re microcontroler ºi lumea de afar• Portul A are 5 pini, iar portul B are 8 pini.

TIMER-UL LIBER (FREE-RUN) este un registru de 8 biþi •interiorul microcontrolerului ce lucreaz• independent de program. La fiecare al patrulea impuls de ceas al oscilatorului • •rementeaz•aloarea lui p• ce atinge maximul (255), ºi apoi •epe s•umere tot din nou de la zero. Dup•um ºtim timpul exact dintre fiecare dou•ncrement• ale conþinutului timer-ului, poate fi folosit pentru m•rarea timpului ce este foarte util la unele componente.

UNITATEA DE PROCESARE CENTRALà are rolul unui element de conectivitate •re celelalte blocuri ale microcontrolerului. Coordoneaz•ucrul altor blocuri ºi execut•rogramul utilizatorului.



Schiţa microcontrolerului PIC16F84



Bloc de arhitecturi Harvard vs. von Neumann

CISC, RISC

S-a spus deja c•IC1684 are o arhitectur•ISC. Acest termen este adeseori g•t • literatura despre calculatoare, ºi are nevoie s•ie explicat aici mai • detaliu. Arhitectura Harvard este un concept mai nou dec• von-Neumann. S-a n•ut din nevoia de m•re a vitezei microcontrolerului. • arhitectura Harvard, bus-ul de date ºi bus-ul de adrese sunt separate. Astfel este posibil un mare debit de date prin unitatea de procesare central•ºi bine•eles, o vitez•ai mare de lucru. Separarea programului de memoria de date face posibil ca mai departe instrucþiunile s•u trebuiasc••ie cuvinte de 8 biþi. PIC16F84 foloseºte 14 biþi pentru instrucþiuni ceea ce permite ca toate instrucþiunile s•ie instrucþiuni dintr-un singur cuv•. Este de asemenea tipic pentru arhitectura Harvard s•ib•ai puþine instrucþiuni dec•von-Newmann ºi s•ib• nstrucþiuni executate uzual intr-un ciclu._

Microcontrolerele cu arhitectur•arvard sunt de asemenea numite "microcontrolere RISC". RISC •eamn•educed Instruction Set Computer. Microcontrolerele cu arhitectura von-Newmann sunt numite "microcontrolere CISC". Titlul CISC •eamn•omplex Instruction Set Computer.

Pentru c• PIC16F84 este un microcontroler RISC, aceasta •eamn••re un set redus de instrucþiuni, mai precis 35 de instrucþiuni (de ex. microcontrolerele INTEL ºi Motorola au peste 100 de instrucþiuni). Toate aceste instrucþiuni sunt executate •r-un ciclu cu excepþia instrucþiunilor jump ºi branch. Conform cu ceea ce spune constructorul, PIC16F84 ajunge la rezultate de 2:1 •compresia cod ºi 4:1 •vitez•n comparaþie cu alte microcontrolere de 8 biþi din clasa sa.

Aplicaþii

PIC16F84 se potriveºte perfect •multe folosinþe, de la industriile auto ºi aplicaþiile de control casnice la instrumentele industriale, senzori la distanþ•m•re electrice de uºi ºi dispozitivele de securitate. Este de asemenea ideal pentru cardurile smart ca ºi pentru aparatele alimentate de baterie din cauza consumului lui mic.

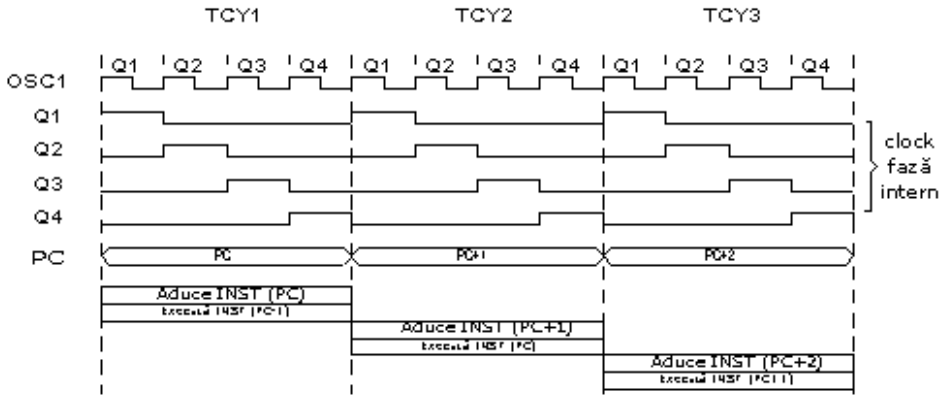
Memoria EEPROM face mai uºoar•plicarea microcontrolerelor la aparate unde se cere memorarea permanent• diferitor parametri (coduri pentru transmiþ•are, viteza motorului, frecvenþele receptorului, etc.). Costul sc•t, consumul sc•t, m•irea uºoar•i flexibilitatea fac PIC16F84 aplicabil chiar ºi •domenii unde microcontrolerele nu au fost prev•te •inte (exemple: funcþii de timer, •ocuirea interfeþei • sistemele mari, aplicaþiile coprocesor, etc.).
Programabilitatea sistemului acestui cip (•reun•u folosirea a doar doi pini •transferul de date) face posibil•lexibilitatea produsului, dup• ce asamblarea ºi testarea au fost terminate. Aceast•apabilitate poate fi folosit•entru a crea producþie pe linie de asamblare, de a •agazina date de calibrare disponibile doar dup•estarea final•sau poate fi folosit pentru a •un•þi programele la produsele finite.

Clock-ul /ciclul instrucþiune

Clock-ul sau ceasul este starter-ul principal al microcontrolerului, ºi este obþinut dintr-o component•e memorie extern• bsp; numit•quot;oscilator". Dac•r fi s•ompar•un microcontroler cu un ceas de timp, "clock-ul" nostru ar fi un tic• pe care l-am auzi de la ceasul de timp. • acest caz, oscilatorul ar putea fi comparat cu arcul ce este r•cit astfel ca ceasul de timp s•earg•De asemenea, forþa folosit•entru a •oarce ceasul poate fi comparat•u o surs•lectric•

Clock-ul de la oscilator intr•ntr-un microcontroler prin pinul OSC1 unde circuitul intern al microcontrolerului divide clock-ul •4 clock-uri egale Q1, Q2, Q3 ºi Q4 ce nu se suprapun. Aceste 4 clock-uri constituie un ciclu de o singur• nstrucþiune (numit de asemenea ciclu maºin••timpul c•ia instrucþiunea este executat•

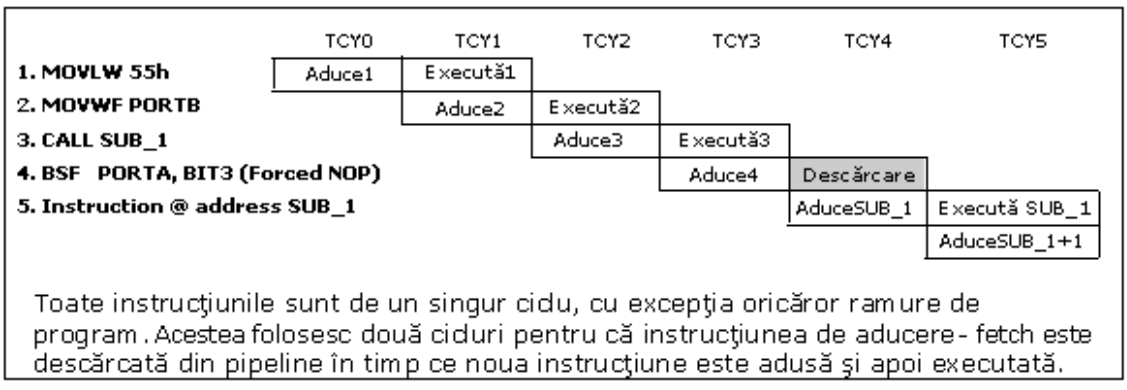
Executarea instrucþiunii •epe prin apelarea unei instrucþiuni care este urm•area •linie. Instrucþiunea este apelat•in memoria program la fiecare Q1 ºi este scris•n registrul de instrucþiuni la Q4. Decodarea ºi executarea instrucþiunii sunt f•te •re urm•arele cicluri Q1 ºi Q4. • urm•area diagram•utem vedea relaþia dintre ciclul instrucþiunii ºi clock-ul oscilatorului (OSC1) ca ºi aceea a clock-urilor interne Q1-Q4. Contorul de program (PC) reþine informaþia despre adresa urm•arei instrucþiuni.



Ciclul Clock/Instrucţiune

Pipelining

Ciclul instrucþiune const•in ciclurile Q1, Q2, Q3 ºi Q4. Ciclurile de instrucþiuni de apelare ºi executare sunt conectate •r-un aºa fel •• pentru a face o apelare, este necesar un ciclu cu o instrucþiune, ºi mai este nevoie de ••nul pentru decodare ºi executare. Totuºi, datorit• pipelining-ului (folosirea unei pipeline-conduct•ºi este aducerea unei instrucþiuni din memorie •timp ce se execut•lta), fiecare instrucþiune este executat•fectiv •r-un singur ciclu. Dac• nstrucþiunea cauzeaz• schimbare •contorul programului, ºi PC-ul nu direcþioneaz•pre urm•area ci spre alte adrese (poate fi cazul cu subprogramele jumps sau calling), 2 cicluri sunt necesare pentru executarea unei instrucþiuni. Aceasta este pentru c•nstrucþiunea trebuie procesat•in nou, dar de data aceasta de la adresa corect•Ciclul •epe cu clock-ul Q1, prin scrierea •registrul instruction register (IR). Decodarea ºi executarea •epe cu clock-urile Q2, Q3 ºi Q4.



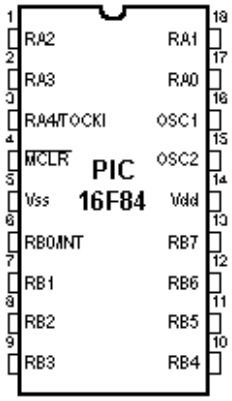
Debitul Pipeline-ului instrucţiunii

TYC0 citeºte instrucþiunea MOVLW 55h (nu are importanþ•span> pentru noi ce instrucþiune a fost executat•ce explic• e ce nu este un dreptunghi desenat •partea de jos).
TCYI execut•nstrucþiunea MOVLW 55h ºi citeºte MOVWF PORTB.
TCY2 execut•OVWF PORTB ºi citeºte CALL SUB_1.
TCY3 execut• apelare a subprogramului CALL SUB_1, ºi citeºte instrucþiunea BSF PORTA, BIT3. Pentru c•nstrucþiunea aceasta nu este aceea de care avem nevoie, sau nu este prima instrucþiune a subprogramului SUB_1 a c•i execuþie este urm•area •ordine, instrucþiunea trebuie citit•in nou. Acesta este un bun exemplu a unei instrucþiuni av• nevoie de mai mult de un ciclu.
TCY4 ciclul instrucþiunii este total folosit pentru citirea primei instrucþiuni din subprogram la adresa SUB_1.

TCY5 execut•rima instrucþiune din subprogram SUB_1 ºi citeºte urm•area.

Semnificaþia pinilor

PIC16F84 are un num•total de 18 pini. Cel mai adesea se g•ºte •r-o capsul•e tip DIP18 dar se poate g• de asemenea ºi •r-o capsul•MD care este mai mic•a cea DIP. DIP este prescurtarea de la Dual In Package. SMD este prescurtarea de la Surface Mount Devices suger• c• g•ile pentru pini unde s•ntre aceºtia, nu sunt necesare •lipirea acestui tip de component•/span>



Pinii microcontrolerului PIC16F84 au urm•area semnificaþie:

Pin nr.1 **RA2** Al doilea pin la portul A. Nu are funcþie adiþional•br> Pin nr.2 **RA3** Al treilea pin la portul A. Nu are funcþie adiþional•br> Pin nr.3 **RA4** Al patrulea pin la portul A. TOCK1 care funcþioneaz• ca timer se g•ºte de asemenea la acest pin.
Pin nr.4 **MCLR** Reseteaz•ntrarea ºi tensiunea de programare Vpp a microcontrolerului.
Pin nr.5 **VSS** Alimentare, mas•/span>
Pin nr.6 **RB0** Pin de zero la portul B. Intrarea •/span>ntrerupere este o funcþie adiþional•/span>
Pin nr.7 **RB1** Primul pin la portul B. Nu are funcþie adiþional•/span>
Pin nr.8 **RB2** Al doilea pin la portul B. Nu are funcþie adiþional•/span>
Pin nr.9 **RB3** Al treilea pin la portul B. Nu are funcþie adiþional•/span>
Pin nr.10 **RB4** Al patrulea pin la portul B. Nu are funcþie adiþional•/span>
Pin nr.11 **RB5** Al cincilea pin la portul B. Nu are funcþie adiþional•br> Pin nr.12 **RB6** Al ºaselea pin la portul B. Linia de 'Clock' •mod programare.
Pin nr.13 **RB7** Al ºaptelea pin la portul B. Linia 'Data' •mod programare.
Pin nr.14 **Vdd** Polul pozitiv al sursei.
Pin nr.15 **OSC2** Pin desemnat pentru conectarea la un oscilator.
Pin nr.16 **OSC1** Pin desemnat pentru conectarea la un oscilator.
Pin nr.17 **RA2** Al doilea pin la portul A. Nu are funcþie adiþional•br> Pin nr.18 **RA1** Primul pin la portul A. Nu are funcþie adiþional•/span>

Pagina anterioar•font>	Conþinut	Pagina urm•are

CAPITOLUL 3

Set Instruc•iuni

Introducere

Am men•ionat deja c• microcontrolerul nu este ca orice alt circuit integrat. Când ies din produc•ie cele mai multe circuite integrate sunt gata de a fi introduse în aparate ceea ce nu este cazul cu microcontrolerele. Pentru a "face" microcontrolerul s• îndeplineasc• o sarcin•, trebuie s•-i spunem exact ce s• fac•, sau cu alte cuvinte trebuie s• scriem programul pe care microcontrolerul s•-l execute. Vom descrie în acest capitol instruc•iunile care alc•tuiesc assembler-ul, sau limbajul de programare cu nivel sc•zut pentru microcontrolerele PIC.

Set de Instruc•iuni în Familia Microcontrolerului PIC16Cxx

Setul complet care cuprinde 35 de instruc•iuni este dat în tabela urm•toare. Un motiv pentru un num•r a•a de mic de instruc•iuni st• în primul rând în faptul c• discut•m despre un microcontroler RISC ale c•rui instruc•iuni sunt bine optimizate având în vedere viteza de lucru, simplitatea arhitectural• •i compactitatea codului. Singurul neajuns este c• programatorul trebuie s• controleze o tehnic• "neconfortabil•" în a utiliza un set modest de 35 de instruc•iuni.

Transfer de Date

Transferul de date într-un microcontroler este f•cut între registrul de lucru (W) •i un registru 'f' ce reprezint• orice loca•ie în RAM-ul intern (indiferent dac• ace•tia sunt regi•tri speciali sau de scop general).

Primele trei instruc•iuni (a se vedea urm•torul tabel) fac ca o constant• s• fie înscris• în registrul W (MOVLW este prescurtarea pentru MOVe Literal to W), •i ca datele s• fie copiate din registrul W în RAM •i datele din RAM s• fie copiate în registrul W (sau în aceea•i loca•ie RAM, la care punct numai starea stegule•ului Z se schimb•).
Instruc•iunea CLRF scrie constanta 0 în registrul 'f ', iar CLRW scrie constanta 0 în registrul W. Instruc•iunea SWAPF schimb• locurile câmpului de nibbles- buc••i de 4 bi•i în interiorul unui registru.

Aritmetic• •i logic•

Din toate opera•iile aritmetice, PIC ca majoritatea microcontrolerelor, accept• doar sc•derea •i adunarea. Stegule•ele C, DC •i Z sunt setate func•ie de rezultatul adun•rii sau sc•derii, dar cu o excep•ie: pentru c• sc•derea se face ca o adunare a unei valori negative, eticheta C este invers• urmând sc•derii. Cu alte cuvinte, este setat• dac• opera•ia este posibil•, •i este resetat• dac• un num•r mai mare a fost sc•zut din unul mai mic.

Unitatea logic• a PIC-ului are capabilitatea de a face opera•iile AND (•I), OR (SAU), EX-OR (SAU-EXCLUSIV), complementare (COMF) •i rota•ie (RLF •i RRF).
Instruc•iunile ce rotesc con•inutul registrului mut• bi•ii în interiorul registrului prin eticheta C cu un spa•iu la stânga (c•tre bitul 7), sau la dreapta (c•tre bitul 0). Bitul ce "iese" din registru este scris în stegule•ul C, •i valoarea stegule•ului C este scris• într-un bit al "p•r•ii opuse" a registrului.

Opera•ii cu bi•i

Instruc•iunile BCF •i BSF fac setarea sau •tergerea unui singur bit oriunde în memorie. Chiar dac• pare o simpl• opera•ie, este executat• în a•a fel ca CPU cite•te mai întâi întregul byte, schimb• un bit în el •i apoi scrie întregul byte în acela•i loc.

Direc•ionarea debitului unui program

Instruc•iunile GOTO, CALL •i RETURN sunt executate în acela•i fel ca •i în celelalte microcontrolere, numai stiva este independent• de RAM-ul intern •i limitat• la opt nivele.
Instruc•iunea 'RETLW k' este identic• cu instruc•iunea RETURN, cu excep•ia c• înainte de a se întoarce dintr-un subprogram, constanta definit• operandul de instruc•iuni este scris• în registrul W. Aceast• instruc•iune ne permite s• proiect•m u•or tabelele (listele) Look-up. Cel mai mult le folosim la determinarea pozi•iei datelor în tabelul nostru ad•ugând-o la adresa la care încep tabelele, •i apoi citim datele din acea loca•ie (care este uzual g•sit• în memoria program).

Tabelul poate fi format ca un subprogram ce const• dintr-o serie de instruc•iuni 'RETLW k', unde constantele 'k' sunt membri ai tabelului.

```
Main    molov 2
        call Lookup
Lookup  addwf PCL, f
        retlw k
        retlw k1
        retlw k2
          :
          :
        retlw kn
```

Scriem pozi•ia unui membru al tabelului nostru în registrul W, •i folosind instruc•iunea CALL apel•m un subprogram care creaz• tabelul. Prima linie de subprogram ADDWF PCL, f adaug• pozi•ia unui membru al registrului W la adresa de start a tabelului nostru, g•sit• în registrul PCL, •i astfel ob•inem adresa datelor reale în memoria program. Când ne întoarcem dintr-un subprogram vom avea în registrul W con•inutul unui membru al tabelului adresat. În exemplul anterior, constanta 'k2' va fi în registrul W urmând unei întoarceri dintr-un subprogram.

RETFIE (RETurn From Interrupt - Interrupt Enable) este o întoarcere dintr-o rutin• de întrerupere •i difer• de o RETURN numai în aceea c• seteaz• automat bitul GIE (Global Interrupt Enable). La o întrerupere, acest bit este automat •ters. Când începe întreruperea, numai valoarea contorului de program este pus• în vârful stivei. Nu este prev•zut• memorarea automat• a valorilor •i st•rii registrului.

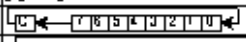
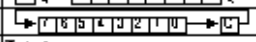
Jump-urile (salturile) condi•ionale sunt sintetizate în dou• instruc•iuni: BTFSC •i BTFSS. Func•ie de starea bitului în registrul 'f' ce este testat, instruc•iunile sar sau nu peste instruc•iunea de program urm•toare.

Perioada de Execu•ie a Instruc•iunii

Toate instruc•iunile sunt executate într-un ciclu cu excep•ia instruc•iunilor ramur• condi•ionale dac• condi•ia a fost adev•rat•, sau dac• con•inutul contorului de program a fost schimbat de o anumit• instruc•iune. În acest caz, execu•ia cere dou• cicluri de instruc•iuni, iar al doilea ciclu este executat ca NOP (No Operation-F•r• opera•ii). Patru clock-uri oscilator fac un ciclu instruc•iune. Dac• folosim un oscilator cu frecven•a de 4 MHz, timpul normal pentru execu•ia instruc•iunii este 1 µs, •i în caz de branching-ramificare condi•ional•, perioada de execu•ie este 2 µs.

List• de cuvinte

f orice loca•ie de memorie într-un microcontroler
W registru de lucru
b pozi•ie bit în registru 'f'
d bit destina•ie
label grup de opt caractere ce marcheaz• începutul unei p•r•i de program
TOS vârful stivei
[] op•iune
<> pozi•ie bit în registru

Mnemonic		Descriere	Operaţie	Steguleţ	Ciclu	Notă
		Transfer date				
MOVLW	k	Mută constanta în W	k → W		1	
MOVWF	f	Mută W în f	W → f		1	
MOVF	f, d	Mută f	f → d	Z	1	1,2
CLRW	-	Şterge W	00h → W, 1 → Z	Z	1	
CLRF	f	Şterge f	00h → f, 1 → Z	Z	1	2
SWAPF	f, d	Interschimbă nibble-urile în f	f(7:4) → (3:0), f(3:0) → (7:4)		1	1,2
		Aritmetică şi logică				
ADDLW	k	Adună constanta cu W	W+k → W	C,DC,Z	1	
ADDWF	f, d	Adună W cu f	W+f → d	C,DC,Z	1	1,2
SUBLW	k	Scade W din constanta	k-W → W	C,DC,Z	1	
SUBWF	f, d	Scade W din f	f-W → d	C,DC,Z	1	1,2
ANDLW	k	ŞI literal cu W	W.AND.k → W	Z	1	
ANDWF	f, d	ŞI W cu f	W.AND.f → d	Z	1	1,2
IORLW	k	SAU inclusiv constanta cu W	W.OR.k → W	Z	1	
IORWF	f, d	SAU inclusiv W cu f	W.OR.f → d	Z	1	1,2
XORLW	k	SAU exclusiv constanta cu W	W.XOR.k → W	Z	1	1,2
XORWF	f, d	SAU exclusiv W cu f	W.XOR.f → d	Z	1	
INCF	f, d	Incrementează f	f+1 → f	Z	1	1,2
DECF	f, d	Decrementează f	f-1 → f	Z	1	1,2
RLF	f, d	Roteşte la stânga prin Carry		C	1	1,2
RRF	f, d	Roteşte la dreapta prin Carry		C	1	1,2
COMF	f, d	Complement f	$\overline{f}$ → d	Z	1	1,2
		Operaţii cu biţi				
BCF	f, b	Şterge bitul f	0 → f(b)		1	1,2
BSF	f, b	Setează bitul f	1 → f(b)		1	1,2
		Direcţionarea unui debit de program				
BTFSC	f, b	Testează bitul f, Sari dacă este şters	salt dacă f(b)=0		1 (2)	3
BTFSS	f, b	Testează bitul f, Sari dacă este setat	salt dacă f(b)=1		1 (2)	3
DECFSZ	f, d	Decrementează f, Sari dacă este 0	f-1 → d, salt dacă Z=1		1(2)	1,2,3
INCFSZ	f, d	Incrementează f, Sari dacă este 0	f+1 → d, salt dacă Z=1		1(2)	1,2,3
GOTO	k	Du-te la adresă	W.AND.k → W		2	
CALL	k	Apelează subrutina	W.AND.f → d		2	
RETURN	-	Întoarcere din Subrutină	TOS → PC		2	
RETLW	k	Întoarcere cu constanta în W	k → W, TOS → PC		2	
RETFIE	-	Întoarcere din întrerupere	TOS → PC, 1 → GIE		2	
		Alte instrucţiuni				
NOP	-	Fără Operaţii			1	
CLRWDT	-	Şterge Timer-ul Watchdog	0 → WDI, 1 → IQ, 1 → PD	$\overline{TO}$, $\overline{PD}$	1	
SLEEP	-	Du-te în mod standby	0 → WDI, 1 → IQ, 0 → PD	$\overline{TO}$, $\overline{PD}$	1	

*1 Dac• portul I/O este operand surs•, este citit• starea pinilor microcontrolerului
*2 Dac• aceast• instruc•iune este executat• în registrul TMR0 •i dac• d=1, prescaler-ul asignat acelui timer va fi automat •ters
*3 Dac• PC s-a modificat, sau rezultatul testului =1, instruc•iunea s-a executat în dou• cicluri

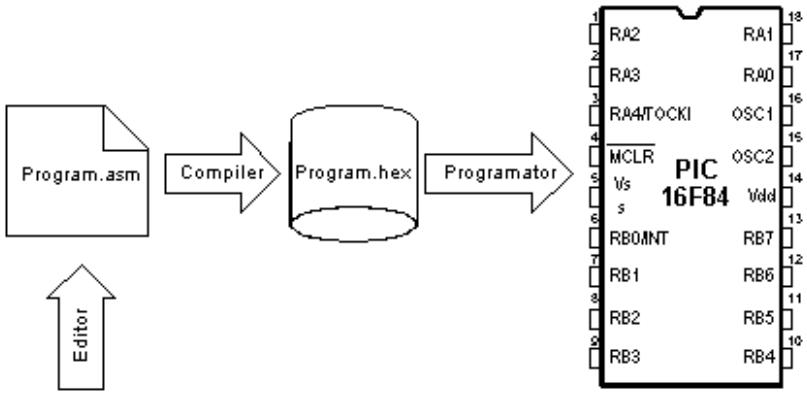
Pagina anterioar• | Con•inut | Pagina urm•toare

CAPITOLUL 4

Programare în Limbaj de Asamblare

Introducere

Abilitatea de a comunica este de mare importan•• în orice domeniu. Totu•i, este posibil• numai dac• amândoi partenerii de comunicare cunosc acela•i limbaj, sau urm•resc acelea•i reguli în timpul comunic•rii. Folosind aceste principii ca un punct de plecare, putem de asemenea defini comunicarea ce are loc între microcontrolere •i om. Limbajul pe care microcontrolerul •i omul îl folosesc pentru a comunica este numit "limbaj de asamblare". Titlul însu•i nu are un în•eles deosebit, •i este analog numelor altor limbaje, de ex. engleza •i franceza. Mai precis, "limbajul de asamblare" este doar o solu•ie trec•toare. Programele scrise în limbaj de asamblare trebuie traduse într-un "limbaj de zero-uri •i unu-uri" pentru ca un microcontroler s•-l în•eleag•. "Limbajul de asamblare" •i "assembler-ul" sau asamblorul sunt dou• no•iuni diferite. Primul reprezint• un set de reguli folosite în scrierea unui program pentru un microcontroler, iar cel•lalt este un program în computerul personal care traduce limbajul de asamblare într-un limbaj de zero-uri •i unu-uri. Un program ce este tradus în "zero-uri" •i "unu-uri" este numit "limbaj ma•in•".



Procesul comunicarii dintre un om si un microcontroler

Fizic, "**Program**" reprezint• un fi•ier pe discul computerului (sau în memorie dac• este citit într-un microcontroler), •i este scris conform cu regulile de asamblare sau ale altui limbaj pentru programarea microcontrolerului. Omul poate în•elege pentru c• este constituit din semne •i cuvinte ale alfabetului. Când se scrie un program, trebuie urm•rite unele reguli pentru a se ob•ine un efect dorit. Un **Translator** interpreteaz• fiecare instruc•iune scris• în limbajul de asamblare ca o serie de zero-uri •i unu-uri ce au o semnifica•ie pentru logica intern• a microcontrolerului.

S• lu•m de exemplu instruc•iunea "RETURN" pe care microcontrolerul o folose•te pentru a se întoarce dintr-un sub-program.

Când asamblorul îl traduce, ob•inem o serie de zero-uri •i unu-uri pe care microcontroleul •tie cum s•-l interpreteze.

Exemplu: RETURN 00 0000 0000 1000

Similar propozi•iei de mai sus, fiecare instruc•iune de asamblare este interpretat• ca •i corespunzând unei serii de zero-uri •i unu-uri.

Locul unde aceast• traducere a limbajului de asamblare se g•se•te , se nume•te un fi•ier de "execu•ie". Vom întâlni adesea numele de fi•ier "HEX". Acest nume vine de la o reprezentare hexazecimal• a acelui fi•ier, ca •i de la apendicele "hex" din titlu, de ex. "run through.hex". Odat• ce este generat, fi•ierul de execu•ie este citit în microcontroler printr-un programator.

Un program în **Limbaj de Asamblare** este scris într-un program pentru procesarea textului (editorul) •i este capabil de a produce un fi•ier ASCII pe discul computerului sau în zone specializate ca MPLAB – ce se va explica în capitolul urm•tor.

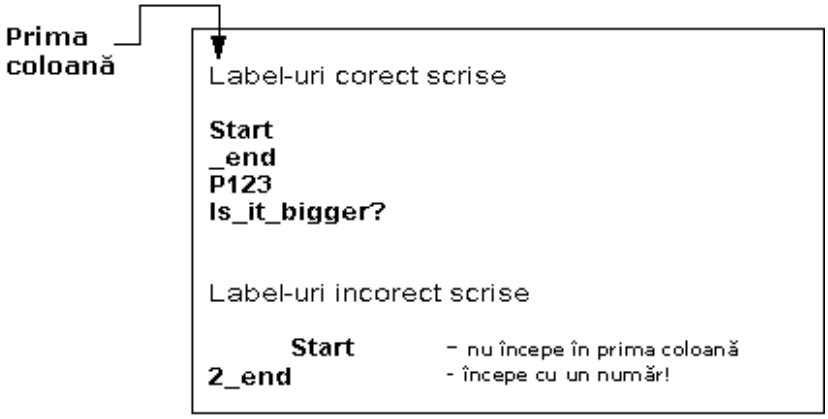
Limbaj de Asamblare

Elementele de baz• ale limbajului de asamblare sunt:

- Label-uri sau Etichete
- Instruc•iuni
- Operanzi
- Directive
- Comentarii

Label-uri

Un **Label** este o desemnare textual• (în general un cuvânt u•or de citit) pentru o linie într-un program, sau sec•iunea unui program unde micro-ul poate s•ri – sau chiar începutul unui set de linii a unui program. Poate fi folosit de asemenea pentru a executa ramificare de program (ca Goto…….) •i programul poate chiar avea o condi•ie ce trebuie îndeplinit• pentru ca instruc•iunea Goto s• fie executat•. Este important pentru un label de a începe cu o liter• a alfabetului sau cu o subliniere "_". Lungimea label-ului poate fi de pân• la 32 caractere. Este de asemenea important ca un label s• înceap• de la primul rând.



Instruc•iuni

Instruc•iunile sunt deja definite prin folosirea unui microcontroler specific, a•a c• ne r•mâne doar s• urm•m instruc•iunile pentru folosirea lor în limbajul de asamblare. Modul în care scriem o instruc•iune mai este numit "sintaxa" instruc•iunii. În exemplul urm•tor putem recunoa•te o gre•eal• în scriere pentru c• instruc•iunile movlp •i goto nu exist• pentru microcontrolerul PIC16F84.

Instrucţiuni corect scrise

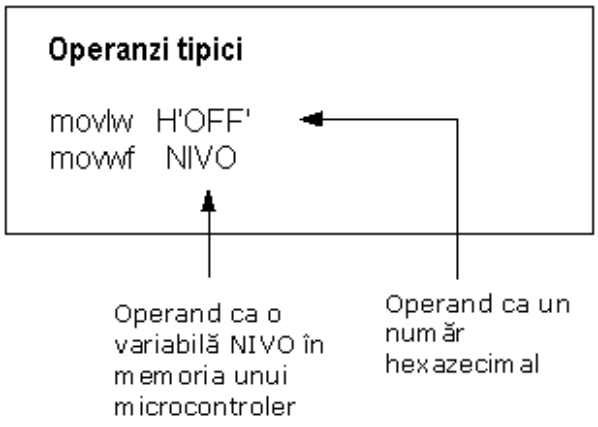
```
movlw     H'01FF'
goto      Start
```

Instrucţiuni incorect scrise

```
movlp     H'01FF'
gotto     Start
```

Operanzi

Operanzii sunt elemente ale instruc•iunii pentru instruc•iunea ce este executat•. Ei sunt de obicei **regi•tri** sau **variabile** sau **constante.** Constantele sunt numite "literal-e". Cuvântul literal înseamn• "num•r".



Comentarii

Comentariul este o serie de cuvinte pe care programatorul le scrie pentru a face programul mai clar •i mai u•or de citit. Se plaseaz• dup• o instruc•iune , •i trebuie s• înceap• cu punct •i virgul•";".

Directive

O **directiv•** este similar• unei instruc•iuni, dar spre deosebire de o instruc•iune este independent• de modelul microcontrolerului, •i reprezint• o caracteristic• a limbajului de asamblare însu•i. Directivelor le sunt date uzual în•elesuri de scop prin variabile •i regi•tri. De exemplu, LEVEL poate fi o desemna•ie pentru o variabil• în memoria RAM la adresa 0Dh. În felul acesta, variabila la acea adres• poate fi accesat• prin desemna•ia LEVEL. Aceasta este mult mai u•or pentru un programator s• în•eleag• decât s• încerce s•-•i aduc• aminte c• adresa 0Dh con•ine informa•ii despre LEVEL.

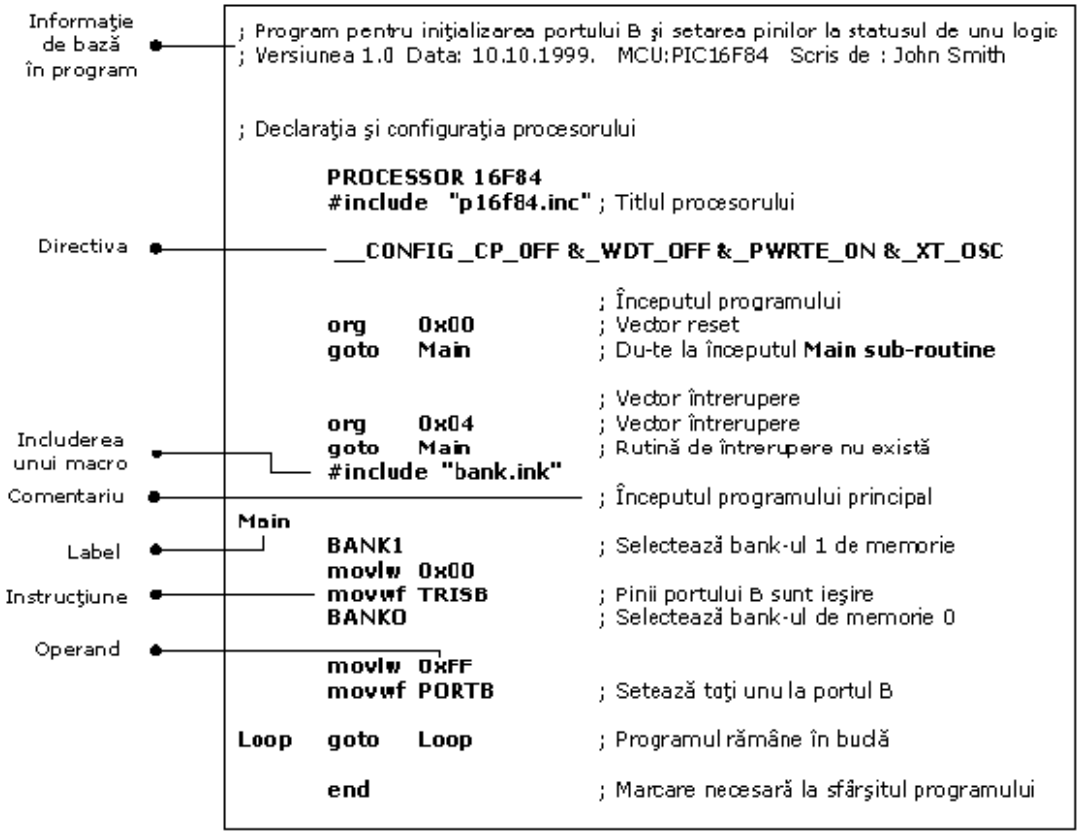
Unele directive frecvent folosite:

```
PROCESSOR 16F84
#include  "p16f84.inc"

__CONFIG _CP_OFF &_WDT_OFF &_PWRTE_ON &_XT_OSC
```

Un exemplu de program scris

Urm•torul exemplu ilustreaz• un program simplu scris în limbaj de asamblare respectând regulile de baz•.

Când se scrie un program, înafar• de regulile obligatorii, sunt de asemenea unele reguli ce nu sunt scrise dar trebuie urmate. Una din ele s• scrii numele programului la început, ce face programul, versiunea lui, date când a fost scris, tipul microcontrolerului pentru care a fost scris, •i numele programatorului.



Pentru c• aceste date nu sunt importante pentru translatorul de asamblare, este scris ca •i **comentarii**. Trebuie remarcat c• un comentariu începe totdeauna cu punct •i virgul• •i c• poate fi plasat într-un rând nou sau poate urma dup• instruc•iune. Este cel mai bine •inut în rândul al treilea pentru a face traseul u•or de urm•rit.

Dup• deschiderea comentariului ce a fost scris, trebuie inclus• **directiva**. Aceasta este ar•tat în exemplul de mai sus.

Pentru a func•iona corect, trebuie s• definim câ•iva parametri ai microcontrolerului ca:
- tipul oscilatorului
- dac• timer-ul watchdog este pe deschis, •i
- dac• circuitul de resetare intern este activ.

Toate acestea sunt definite prin urm•toarea directiv•:

_CONFIG _CP_OFF&_WDT_OFF&PWRTE_ON&XT_OSC

Când toate elementele necesare au fost definite, putem începe scrierea unui program. În primul rând, este necesar de a determina adresa de unde începe microcontrolerul, dup• pornirea sursei de alimentare. Aceasta este (org 0x00). Adresa de la care începe programul dac• are loc o întrerupere este (org 0x04). Pentru c• acesta este un program simplu, va fi suficient s• direc•ion•m microcontrolerul la începutul programului cu o instruc•iune "**goto Main**".

Instruc•iunile g•site în **Main sub-routine** selecteaz• bank-ul 1 al memoriei (BANK1) pentru a accesa registrul TRISIB, a•a încât portul B s• fie declarat ca o ie•ire (movlw 0x00, movwf TRISIB).

Urm•torul pas este de a selecta bank-ul de memorie 0 •i de plasa statusul unu-lui logic la portul B(movlw 0Xff, movwf PORTB), •i astfel programul principal este terminat. Trebuie s• facem o alt• bucl• unde microcontrolerul s• fie •inut ca s• nu se "r•t•ceasc•" dac• se întâmpl• o eroare. Pentru acest scop, se face o bucl• infinit• unde micro-ul este re•inut în timp ce sursa este conectat•. Necesarul "sfâr•it" de la concluzia fiec•rui program informeaz• translatorul de asamblare c• nu mai sunt instruc•iuni în program.

Directive de control

4.1 #DEFINE Schimb• o bucat• de text pentru o alta

Sintax•:
#define<name> [<text ce schimb• numele>]

Descriere:
De fiecare dat• când apare <name> în program , va fi înlocuit cu <text ce schimb• numele>.

Exemplu:
#define turned on 1
#define turned off 0

Directive similare: #UNDEFINE, IFDEF,IFNDEF

4.2 INCLUDE Include un fi•ier adi•ional într-un program

Sintax•:
#include <file_name>
#include "

Descriere:
O aplica•ie a acestei directive are efect ca •i cum întregul fi•ier a fost copiat într-un loc unde directiva "include" a fost g•sit•. Dac• numele fi•ierului este în paranteze p•trate, avem de a face cu un fi•ier de sistem, •i dac• este în interiorul ghilimelelor de citare, avem de a face cu fi•ier de utilizator. Directiva "include" contribuie la un traseu mai bun al programului principal.

Exemplu:
#include <regs.h>
#include "subprog.asm"

4.3 CONSTANT D• o valoare numeric• constant• desemn•rii textuale

Sintax•:
Constant <name>=<value>

Descriere:
De fiecare dat• când apare <name> în program, va fi înlocuit cu <value>.

Exemplu:
Constant MAXIMUM=100
Constant Length=30

Directive similare: SET, VARIABLE

4.4 VARIABLE D• o valoare numeric• variabil• desemn•rii textuale

Sintax•:
Variable<name>=<value>

Descriere:
Folosind aceast• directiv•, desemnarea textual• se înlocuie•te cu o valoare particular•. Difer• de directiva CONSTANT în aceea c• dup• aplicarea directivei, valoarea desemn•rii textuale poate fi înlocuit•.

Exemplu:
variable level=20
variable time=13

Directive similare: SET, CONSTANT

4.5 SET Definirea variabilei asamblorului

Sintax•:
<name_variable>set<value>

Descriere:
Variabilei <name_variable> îi este ad•ugat• expresia <value>. Directiva SET este similar• lui EQU, dar cu directiva SET numele variabilei poate fi redefinit urmând o defini•ie.

Exemplu:
level set 0
length set 12
level set 45

Directive similare: EQU, VARIABLE

4.6 EQU Definind constanta asamblorului

Sintax•:
<name_constant> equ <value>

Descriere:
To the name of a constant <name_constant> is added value <value>

Exemplu:
five equ 5
six equ 6
seven equ 7

Instruc•iuni similare: SET

4.7 ORG Define•te o adres• de unde programul este înmagazinat în memoria microcontrolerului

Sintax•:
<label>org<value>

Descriere:
Aceasta este cea mai frecvent folosit• directiv•. Cu ajutorul acestei directive definim unde o anumit• parte a programului va fi în memoria program.

Exemplu:
```
Start org  0×00
        movlw
        movwf
```

Primele dou• instruc•iuni ce urmeaz• dup• prima directiv• 'org' sunt memorate de la adresa 00, •i celelalte dou• de la adresa 10.

4.8 END Sfâr•it de program

Sintax•:
end

Descriere:
La sfâr•itul fiec•rui program este necesar de a plasa directiva 'end' a•a ca translatorul de asamblare s• •tie c• numai sunt instruc•iuni în program.

Exemplu:
```
.
.
movlw 0xFF
movwf PORTB
end
```

Instruc•iuni condi•ionale

4.9 IF Ramificare de program condi•ional•

Sintax•:
if<conditional_term>

Descriere:
Dac• condi•ia în <conditional_term> este îndeplinit•, parte a programului ce urmeaz• directivei IF va fi executat•. •i dac• nu este, partea ce urmeaz• directivei ELSE sau ENDIF va fi executat•.

Exemplu:
```
if nivo=100
goto PUNI
else
goto PRAZNI
endif
```

Directive similare: #ELSE, ENDIF

4.10 ELSE 'IF' alternativ• la blocul program cu termeni condi•ionali

Sintax•:
Else

Descriere:
Folosit cu directiva IF ca o alterntiv• dac• termenul condi•ional este incorect.

Exemplu:
```
If time< 50
goto SPEED UP
else goto SLOW DOWN
endif
```

Instruc•iuni similare: ENDIF, IF

4.11 ENDIF Sfâr•itul sec•iunii de program condi•ionale

Sintax•:
endif

Descriere:
Directiva este scris• la sfâr•itul blocului condi•ional pentru translatorul de asamblare pentru a •ti c• este sfâr•itul blocului condi•ional

Exemplu:
```
If level=100
goto LOADS
else
goto UNLOADS
endif
```

Directive similare: ELSE, IF

4.12 WHILE Execu•ia sec•iunii programului cât timp condi•ia este îndeplinit•

Sintax•:
while<condition>
.
endw

Descriere:
Liniile de program între WHILE •I ENDW vor fi executate cât timp condi•ia este îndeplinit•. Dac• condi•ia se opre•te din a mai fi valid•, programul continu• executarea instruc•iunilor urmând linia ENDW. Num•rul de instruc•iuni dintre WHILE •i ENDW poate fi cel mult 100, •i num•rul de execu•ii 256.

Exemplu:
While i<10
i=i+1
endw

4.13 ENDW Sfâr•itul p•r•ii condi•ionale a programului

Sintax•:
endw

Descriere:
Instuc•iunea este scris• la sfâr•itul blocului WHILE condi•ional, a•a ca translatorul de asamblare s• •tie c• este sfâr•itul blocului condi•ional

Exemplu:
while i<10
i=i+1

endw

Directive similare: WHILE

4.14 IFDEF Execu•ia unei p•r•i de program dac• simbolul este definit

Sintax•:
ifdef<designation>

Descriere:
Dac• desemnarea <designation> este definit• anterior (cel mai adesea prin instruc•iunea#DEFINE), instruc•iunile ce urmeaz• sunt executate pân• ce nu se ajunge la directivele ELSE •i ENDIF.

Exemplu:
#define test
.
ifdef test ;how the test is defined
.....; instructions from these lines will execute
endif

Directive similare: #DEFINE, ELSE, ENDIF, IFNDEF, #UNDEFINE

4.15 IFNDEF Execu•ia unei p•r•i de program dac• simbolul este definit

Sintax•:
ifndef<designation>

Descriere:
Dac• desemnarea <designation> nu a fost definit• anterior, sau dac• defini•ia ei a fost •tears• cu directiva directive #UNDEFINE, instruc•iunile ce urmeaz• sunt executate pân• ce nu se ajunge la directivele ELSE •i ENDIF.

Exemplu:
#define test
.....
#undefine test
.....
ifndef test ;how the test is undefined
..... .; instructions from these lines will execute
endif

Directive similare: #DEFINE, ELSE, ENDIF, IFDEF, #UNDEFINE

Directive de Date

4.16 CBLOCK Definind un bloc pentru constantele numite

Sintax•:
Cblock [<term>]
<label>[:<increment>], <label>[:<increment>].....
endc

Descriere:
Directiva este folosit• pentru a da valori constantelor numite. Fiecare termen ce urmeaz• prime•te o valoare mai mare cu unu decât precursorul lui. Dac• parametrul <increment> este de asemenea dat, atunci valoarea dat• în parametrul <increment> este ad•ugat• constantei urm•toare. Valoarea parametrului <term> este valoarea de pornire. Dac• nu este dat•, este considerat• a fi zero.

Exemplu:
```
Cblock 0x02
First, second, third  ;first=0x02, second=0x03, third=0x04
endc

cblock 0x02
first : 4, second : 2, third  ;first=0x06, second=0x08, third=0x09
endc
```

Directive similare: ENDC

4.17 ENDC Sfâr•itul defini•iei blocului constante

Sintax•:
```
endc
```

Descriere:
Directiva este folosit• la sfâr•itul defini•iei unui bloc de constante ca translatorul de asamblare s• •tie c• nu mai sunt constante.

Directive similare: CBLOCK

4.18 DB Definind date de un byte

Sintax•:
```
[<term>]db <term> [, <term>,.....,<term>]
```

Descriere:
Directiva rezerv• un byte în memoria de program. Când sunt mai mul•i termeni ce au nevoie s• li se desemneze un byte de fiecare, ei vor fi desemna•i unul dup• altul.

Exemplu:
```
db 't', 0×0f, 'e', 's', 0×12
```

Instruc•iuni similare: DE, DT

4.19 DE Definind Byte-ul de memorie EEPROM

Sintax•:
```
[<term>] de <term> [, <term>,....., <term>]
```

Descriere:
Directiva este folosit• pentru definirea byte-ului de memorie EEPROM. Chiar dac• a fost ini•ial inten•ionat• doar pentru memoria EEPROM, poate fi folosit• pentru oricare alt• loca•ie de memorie.

Exemplu:
```
org H'2100'
de "Version 1.0" , 0
```

Instruc•iuni similare: DB, DT

4.20 DT Definin tabelul de date

Sintax•:
```
[<term>] dt <term> [, <term>,....., <term>]
```

Descriere:
Directiva genereaz• seria RETLW de instruc•iuni, o instruc•iune de fiecare termen.

Exemplu:
```
dt "Message", 0
dt first, second, third
```

Directive similare: DB, DE

Configurând o directiv•

4.21 _CONFIG Setarea the bi•ilor configura•ionali

Sintax•:
· -config<term> or__config<address>,<term>

Descriere:
Sunt definite oscilatorul, aplica•ia timer watchdog •i circuitul intern de reset. Înainte de folosirea acestei directive, procesorul trebuie definit folosind directiva PROCESSOR.

Exemplu:

_CONFIG _CP_OFF&_WDT_OFF&_PWRTE_ON&_XT_OSC

Directive similare: _IDLOCS, PROCESSOR

4.22 PROCESSOR Definind modeul microcontrolerului

Sintax•:
Processor <microcontroller_type>

Descriere:
Instruc•iunea seteaz• tipul microcontrolerului unde programarea este f•cut•.

Exemplu:
processor 16F84

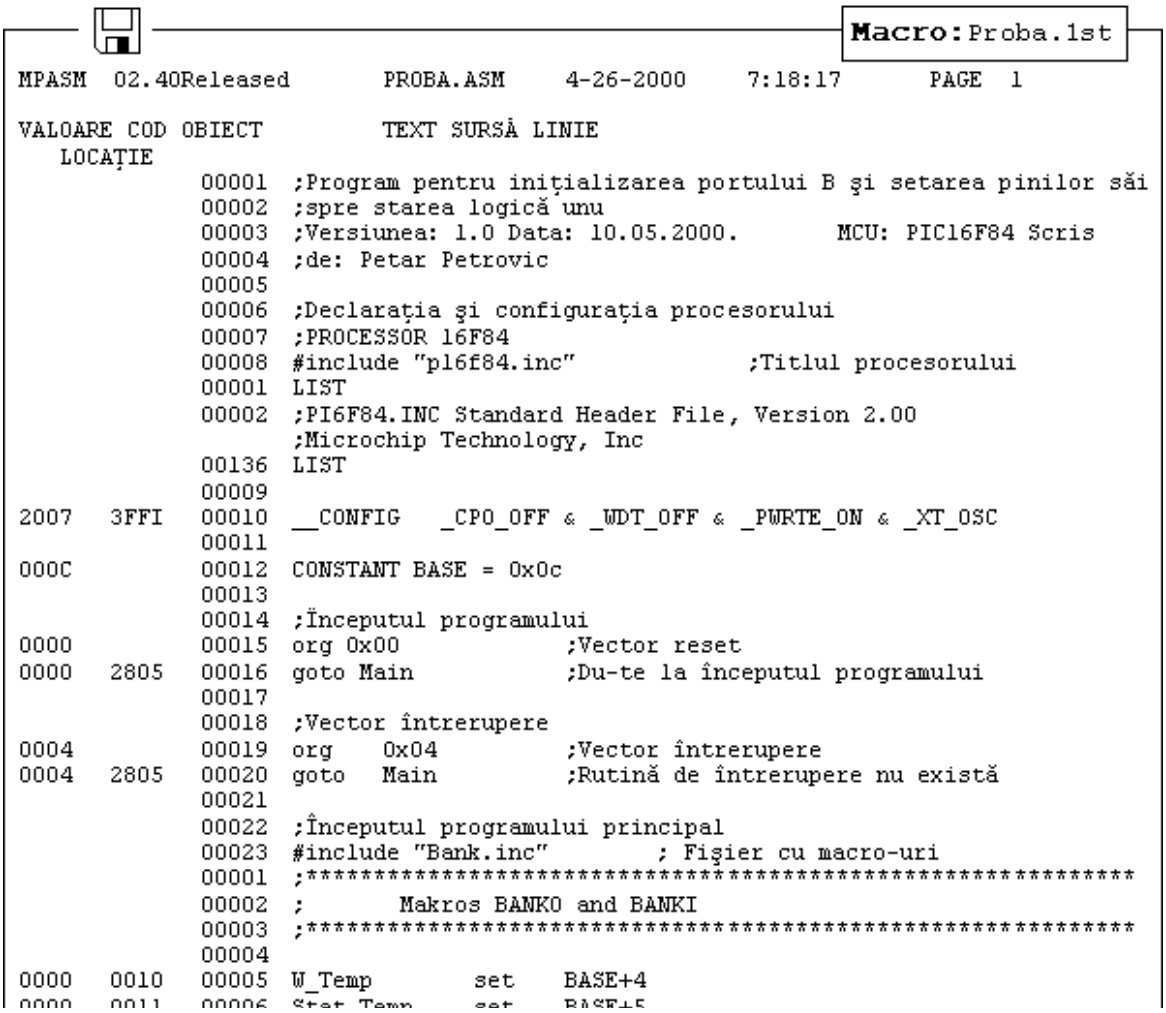
Fi•iere create ca rezultat al transl•rii programului

Ca un rezultat al procesului transl•rii unui program scris în limbaj de asamblare ob•inem fi•iere ca:

- Fi•ier de executare (Program_Name.HEX)
- Fi•ier de erori program (Program_Name.ERR)
- Fi•ier list• (Program_Name.LST)

Primul fi•ier con•ine programul translat ce este citit în microcontroler prin programareare. Con•inutul lui nu poate da orice informa•ie programatorului, a•a c• nu ne vom mai referi la ele în continuare.
Al doilea fi•ier con•ine posibile erorile ce au fost f•cute în procesul scrierii, •i ca au fost observate de translatorul de asamblare în timpul procesului de translare. Erorile pot fi descoperite de asemenea într-un fi•ier "list•". Acest fi•ier este mai potrivit de•i când programul este mare •i vederea fi•ierului "list•" dureaz• mai mult.
Al treilea fi•ier este cel mai folositor programatorului. În el sunt con•inute multe informa•ii, ca informa•ii despre instruc•iunile de pozi•ionare •i variabilele din memorie, sau semnalizarea erorii.

Exemplu unui fi•ier "list•" pentru program urmeaz• în acest capitol. În cap•tul fiec•rei pagini se g•sesc informa•ii despre numele fi•ierului, data când a fost translat •i num•rul paginii. Prima coloan• con•ine o adres• din memoria programului unde este plasat• o instruc•iune din acel rând. A doua coloan• con•ine o valoare a oric•rei variabile definit• de una din directive: SET, EQU, VARIABLE, CONSTANT or CBLOCK. A treia coloan• este rezervat• pentru forma unei instruc•iuni translate pe care PIC-ul o execut•. A patra coloan• con•ine instruc•iunile asamblorului •i comentariile programatorului. Posibile erori vor apare între rânduri urmând o linie în care s-a produs eroarea.



Macro: Proba.lst

```
MPASM  02.40Released       PROBA.ASM     4-26-2000     7:18:17       PAGE  1

VALOARE COD OBIECT         TEXT SURSĂ LINIE
   LOCAŢIE
             00001  ;Program pentru iniţializarea portului B şi setarea pinilor săi
             00002  ;spre starea logică unu
             00003  ;Versiunea: 1.0 Data: 10.05.2000.       MCU: PIC16F84 Scris
             00004  ;de: Petar Petrovic
             00005
             00006  ;Declaraţia şi configuraţia procesorului
             00007  ;PROCESSOR 16F84
             00008  #include "p16f84.inc"             ;Titlul procesorului
             00001  LIST
             00002  ;PI6F84.INC Standard Header File, Version 2.00
                    ;Microchip Technology, Inc
             00136  LIST
             00009
2007   3FFI  00010  __CONFIG   _CPO_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
             00011
000C         00012  CONSTANT BASE = 0x0c
             00013
             00014  ;Începutul programului
0000         00015  org 0x00              ;Vector reset
0000   2805  00016  goto Main             ;Du-te la începutul programului
             00017
             00018  ;Vector întrerupere
0004         00019  org    0x04          ;Vector întrerupere
0004   2805  00020  goto   Main          ;Rutină de întrerupere nu există
             00021
             00022  ;Începutul programului principal
             00023  #include "Bank.inc"         ; Fişier cu macro-uri
             00001  ;*****************************************************************
             00002  ;       Makros BANK0 and BANK1
             00003  ;*****************************************************************
             00004
0000   0010  00005  W_Temp        set    BASE+4
0000   0011  00006  Stat_Temp     set    BASE+5
```

```
0000   0010   00005  W_Temp        set    BASE+4
0000   0011   00006  Stat_Temp     set    BASE+5
0000   0012   00007  Option_Temp   set    BASE+6
              00008
              00009
              00010  BANK0  macro
              00011  bcf    STATUS,RP0    ; Selectează bank-ul de memorie 0
              00012  endm
              00013
              00014  BANK1  macro
              00015  bsf    STATUS,RP0    ; Selectează bank-ul de memorie 1
              00016  endm
              00017
0005          00024  Main
              00025  BANK1                ; Selectează bank-ul de memorie 1
0005   1683   M      bsf    STASTUS,RP0   ; Selectează bank-ul de memorie 1
0006   3000   0026   movlw  00x0
Message[302] :Înregistrează în operand nu în bank-ul 0. Asigură-te că biţii de bank
sunt corecţi.
0007   0086   00027  movwlw TRISB         ;Pinii portului B sunt ieşire
              00028
              00029  BANK0                ;Selectează bank-ul de memorie 0
0008   1283   M      bcf    STATUS,RP0    ;Selectează bank-ul de memorie 0
0009   30FF   00030  movlw  0xFF
000A   0086   00031  movwf  PORTB         ;Setează toţi unu la portul B
              00032
000B   280B   00033  Loop   goto   Loop   ;Programul rămâne în buclă
              00034
              00035  END                  ;Marcare necesară a sfârsitului programului

MEMORY USAGE MAP ('X' = Used, '-' = Unused)

0000 : X---XXXXXXXX---- ---------------- ---------------- ----------------
2000 : -------X-------- ---------------- ---------------- ----------------

Toate celelalte blocuri sunt nefolosite.

Cuvinte folosite în memoria program:    9
Cuvinte libere în memoria program:      1015

Erori:       0
Avertismente: 0 raportate,              0 înlăturate
Mesage:       1 raportate,              0 înlăturate
```

La sfâr•itul fi•ierului "list•" este un tabel cu simboluri folosite în program. Un element folositor al fi•ierului "list•" este un grafic de utilizare a memoriei. La sfâr•it de tot, este o statistic• de erori ca •i cantitatea de program r•mas•.

Macro-uri

Macros-urile sunt elemente foarte folositoare în limbajul de asamblare. Ei ar putea fi pe scurt descri•i ca "grup definit al utilizatorului de instruc•iuni ce vor intra în programul de asamblare unde a fost apelat macro-ul". Este posibil de a scrie un program chiar f•r• folosirea macro-urilor. Dar cu folosirea lor programul scris este mult mai u•or de în•eles, în special dac• mai mul•i programatori lucreaz• la acela•i program. Macro-urile au acela•i scop ca func•ii ale limbajelor de programare complexe.

Cum s• le scriem:

<label> macro [<argument1>,<argument2>,.....<argumentN>]
.....
.....
endm

Din modul în care sunt scrise, vedem c• macro-urile pot accepta argumente, ceea ce este foarte folositor în programare. Când apare argumentul în corpul macro-ului, va fi înlocuit cu valoarea <argumentN>.

Exemplu:

```
NA_PORTB        macro ARG1
                BANK0           ;Selectează bank-ul de memorie 0
                movlw ARG1      ;Valoarea din argumentul ARG1
                                ;este memorată registrul de lucru
                movwf PORTB     ;Valoarea din argumentul
                                ;ARG1 plasată la portul B
                endm            ;macro s-a sfârşit
```

Exemplu de mai sus arat• un macro a c•rui scop este de a înlocui la portul B argumentul ARG1 ce a fost definit în timp ce a fost apelat macro-ul. Folosirea lui în program ar fi limitat• la scrierea unei linii: ON-_PORTB 0xFF , •i astfel am plasa valoarea 0xFF la PORTB. Pentru a folosi un macro în program, este necesar de a include fi•ierul macro în programul principal cu instruc•iunea include "macro_name.inc". Con•inutul unui program este copiat automat într-un loc unde instruc•iunea este scris•. Aceasta poate fi cel mai bine v•zut într-un fi•ier list• anteriror unde fi•ierul cu macro-uri este copiat mai jos de linia #include"bank.inc".

Pagina anterioar•	Con•inut	Pagina urm•toare

CAPITOLUL 5

MPLAB

Introducere

MPLAB este un pachet de program Windows ce face scrierea •i dezvoltarea unui program mai u•oar•. Poate fi descris cel mai bine ca un mediu de dezvoltare pentru un limbaj de programare standard ce este inten•ionat pentru programarea unui computer PC. Unele opera•ii ce erau f•cute din linia de instruc•iuni cu un num•r mare de parametri pân• la descoperirea IDE-ului, "Integrated Development Environment", sunt acum f•cute mai u•oare prin folosirea MPLAB. Totu•i, gusturile noasre difer•, a•a c• chiar ast•zi unii programatori prefer• editoarele standard •i compilatoarele din linia de instruc•iuni. În orice caz, programul scris este u•or de citit, •i este disponibil un help bine documentat.

5.1 Instalarea programului -MPLAB

MPLAB const• din câteva p•r•i:

- Gruparea fi•ierelor aceluia•i proiect într-un singur proiect (Project Manager)
- Generarea •i procesarea unui program (Text Editor)
- Simulator de program scris folosit pentru simularea func•ion•rii programului în microcontroler.

Înafar• de acestea, sunt sisteme de sus•inere pentru produsele Microchip ca PICStart Plus •i ICD (In Circuit Debugger). Pentru c• aceast• carte nu acoper• acestea, ele vor fi men•ionate doar ca op•iuni.

Cerin•ele minime pentru computer pentru rularea lui MPLAB sunt:

• Computer compatibil PC 486 sau mai recent
• Microsoft Windows 3.1x sau Windows 95 •i noile versiuni ale sistemului de operare Windows
• VGA graphic card
• 8MB memorie (32MB recomandat)
• 20MB spa•iu pe hard disc
• Mouse

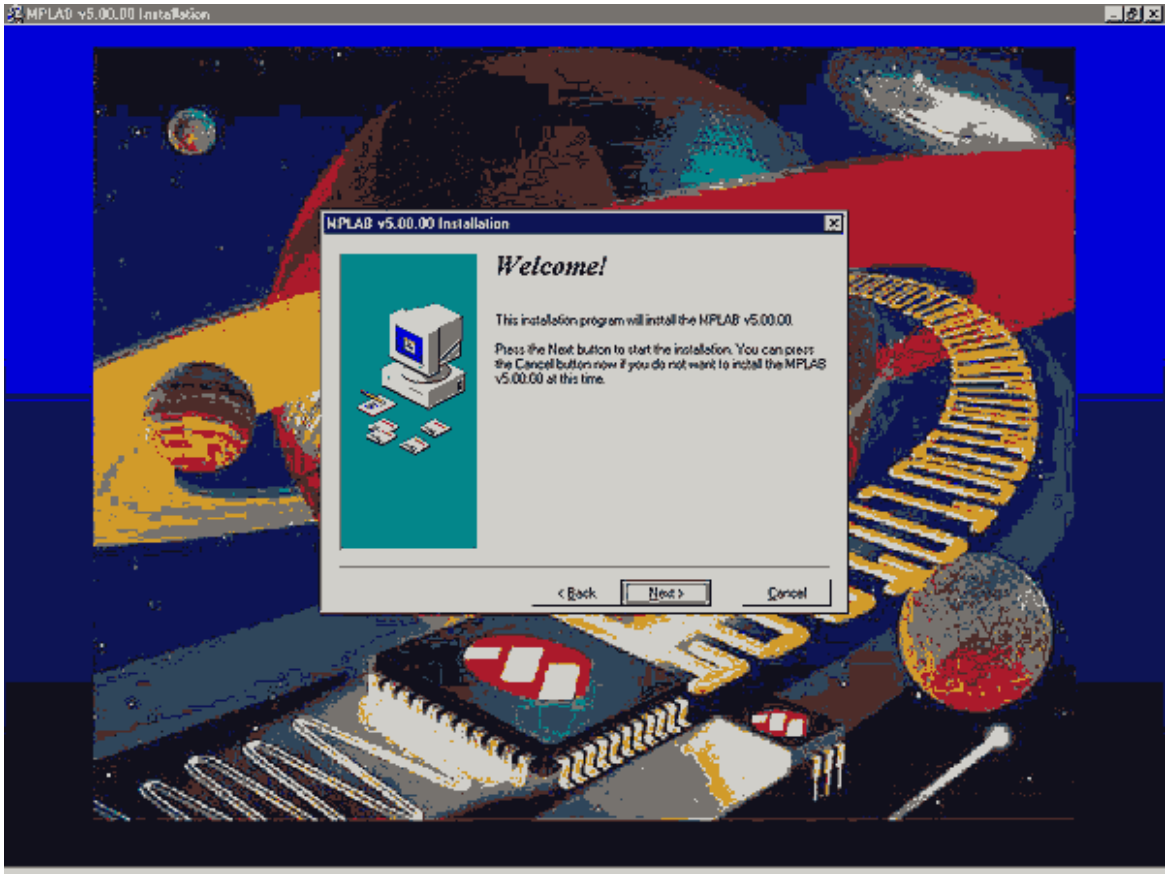
Pentru a porni MPLAB-ul trebuie s•-l instal•m. Instalarea este un proces de copiere a fi•ierelor de pe CD pe un hard disc al computerului. Este o op•iune pentru fiecare fereastr• ce v• ajut• s• v• întoarce•i la cea precedent•, a•a ca erorile s• nu prezinte o

problem• sau s• devin• o experien•• stresant•. Instalarea propriu-zis• are loc ca la majoritatea programelor Windows. Mai întâi apare ecranul Windows, apoi pute•i alege op•iunile urmate de instalarea propriu-zis•, •i în sfâr•it, apare mesajul care spune programul dumneavoastr• instalat este gata de start.

Pa•i pentru instalarea MPLAB:

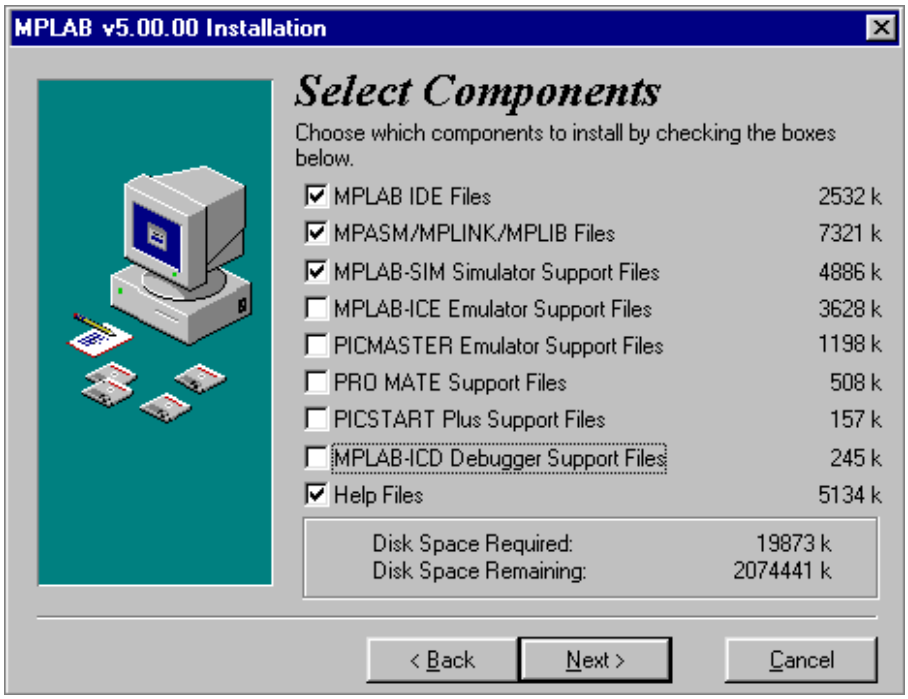
1. Porni•i Windows-ul Microsoft
2. Pune•i the discul CD Microchip în CD ROM
3. Face•i clic pe START în partea stâng• de jos a ecranului •i alege•i op•iunea RUN
4. Face•i clic pe BROWSE •i selecta•i driver-ul CD ROM-ului computerului.
5. G•si•i directorul numit MPLAB pe CD ROM-ul dumneavoastr•
6. Face•i clic pe SETUP.EXE •i apoi pe OK .
7. Face•i clic din nou pe OK în fereastra dumneavoastr• RUN

Instalarea începe dup• ace•ti •apte pa•i. Urm•toarele imagini explic• în•elesul unor pa•i ai instal•rii.



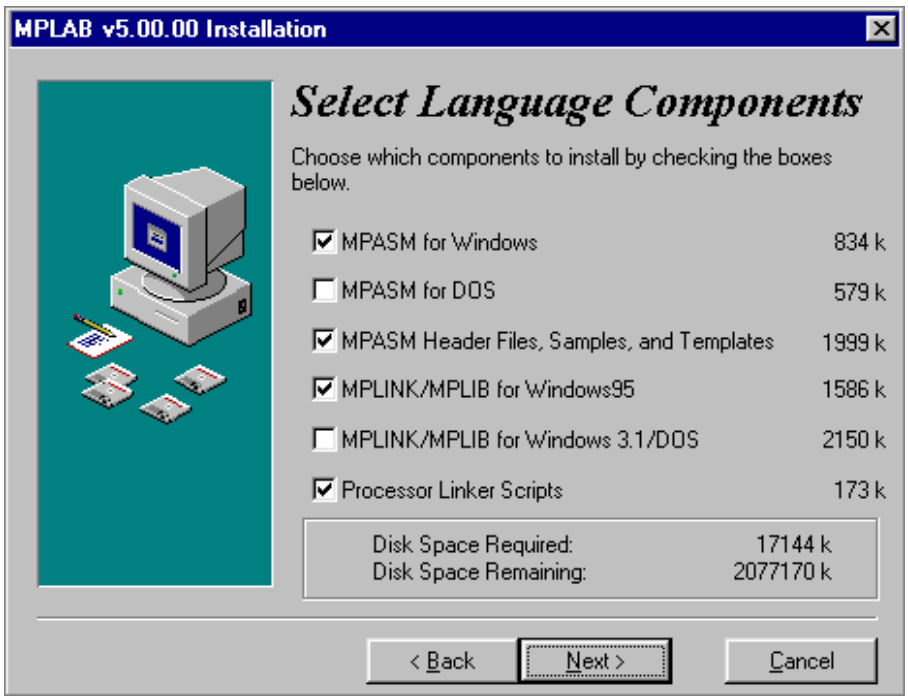
Ecran de bun venit la începutul instal•rii MPLAB

La început de tot, este necesar de a selecta acele componente MPLAB cu care vom lucra. Pentru c• nu avem nici o component• hardware original• Microchip ca programatori sau emulatoare, vom instala doar mediul MPLAB, Assembler-ul, Simulatorul •i instruc•iunile.



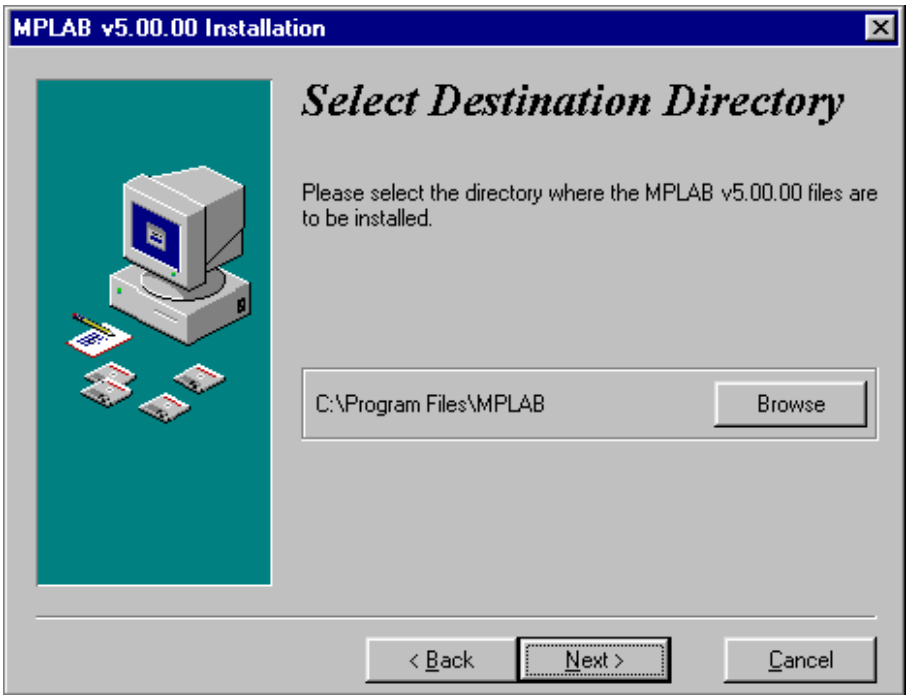
Selectarea componentelor mediului de dezvoltare MPLAB

Întrucât se estimeaz• c• ve•i lucra cu Windows 95 (sau un sistem mai nou), tot ce este în leg•tur• cu sistemul DOS de operare a fost scos în timpul selec•iei limbajului de asamblare. Totu•i dac• dori•i s• lucra•i în DOS, trebuie s• deselecta•i toate op•iunile referitoare la Windows, •i s• alege•i componentele potrivite pentru DOS.



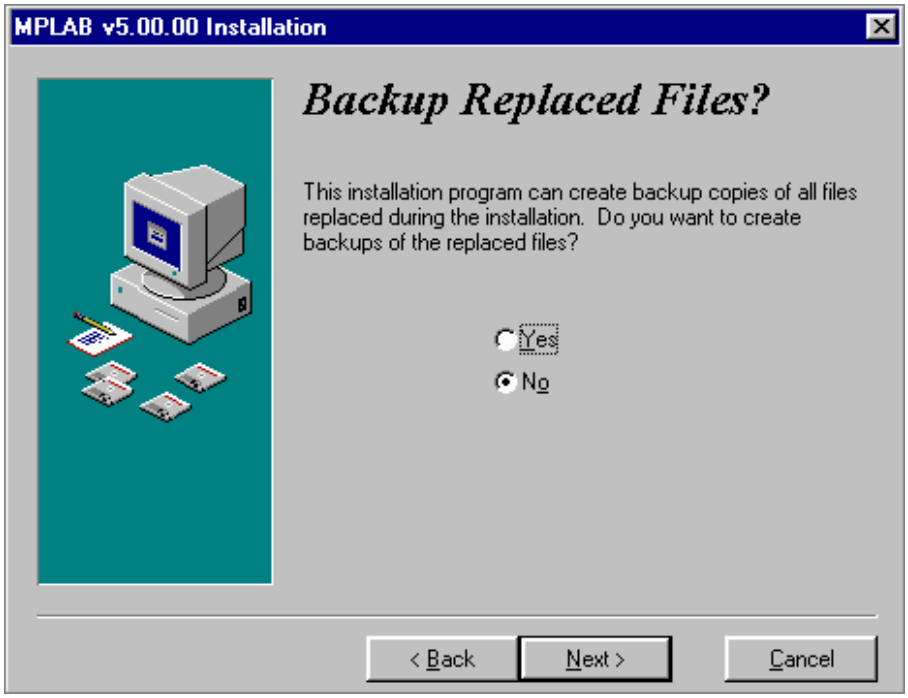
Selectarea assembler-ului •i a sistemului de operare

Ca orice program, MPLAB va trebui instalat într-un director. Aceast• op•iune se poate schimba în orice director de pe orice hard disc al computerului dumneavoastr•. Dac• nu ave•i o nevoie mai presant•, va fi trebui l•sat la locul selectat.



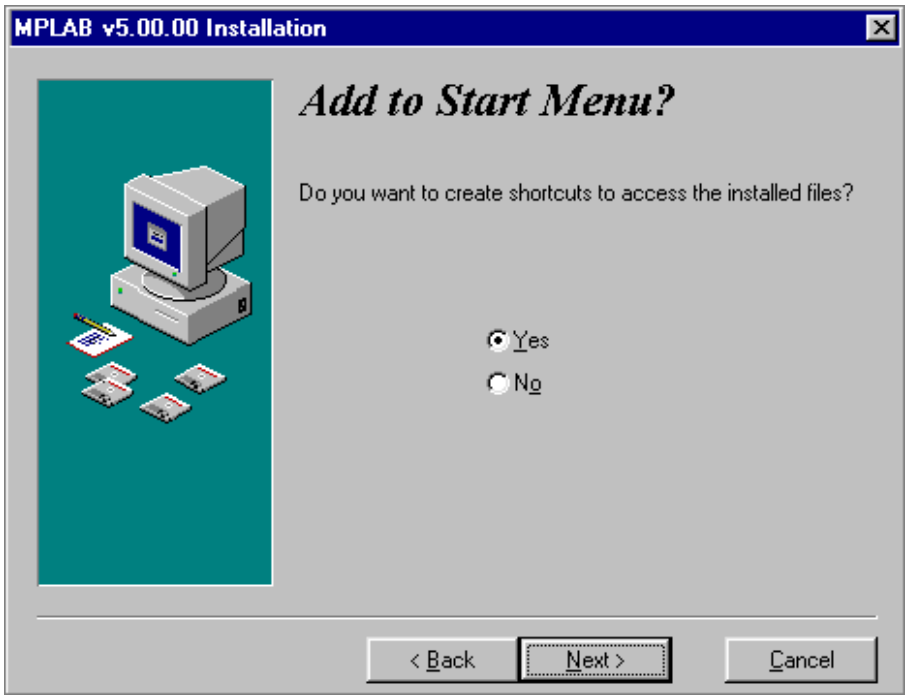
Alegerea directorului unde MPLAB va fi instalat

Utilizatorii care au avut deja MPLAB (o versiune mai veche decât aceasta) au nevoie de urm•toarea op•iune. Scopul acestei op•iuni este de a salva copii a tuturor fi•ierelor ce sunt modificate în timpul unei treceri la o nou• versiune MPLAB. În cazul nostru ar trebui s• l•s•m selectat NO din cauza presupunerii c• aceasta este prima instalare a MPLAB-ului în computerul dumneavoastr•.



Op•iune pentru utilizatorii care instaleaz• o versiune nou• peste o versiune deja instalat• de MPLAB

Start meniu este un grup de pointeri de program, •i este selectat prin clic pe op•iunea START în col•ul de jos stâng al ecranului. Pentru c• MPLAB se va porni de aici, trebuie s• l•s•m aceast• op•iune a•a cum este.



Ad•ugarea MPLAB la start menu

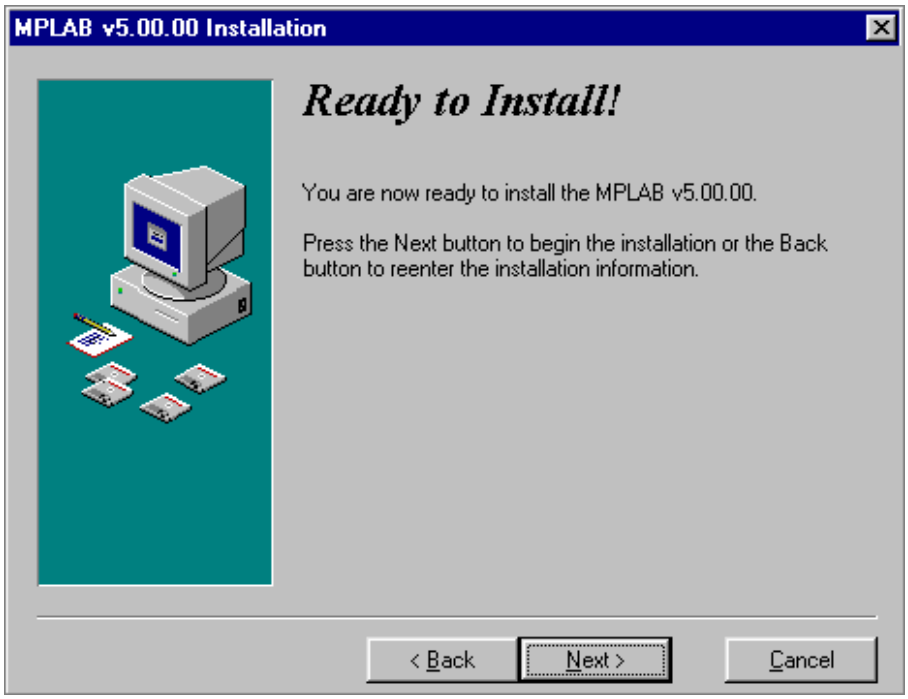
Loca•ia care va fi men•ionat• de aici încolo, are de a face cu o parte a MPLAB în a c•rui explica•ie nu este nevoie s• intr•m. Prin selectarea unui director special, MPLAB va •ine toate fi•ierele în conexiune cu linker-ul într-un director separat.

Determinând un director pentru fi•ierele linker-ului

Orice program Windows are fi•ierele de sistem în mod uzual memorate într-un director con•inând programul Windows. Dup• un num•r de instal•ri diferite. Directorul Windows devine supraaglomerat •i prea mare. Astfel, unele programe permit ca fi•ierele lor de sistem s• fie •inute în aceia•i directori cu programele. MPLAB este un exemplu de asemenea program, •i trebuie selectat• op•iunea de jos.

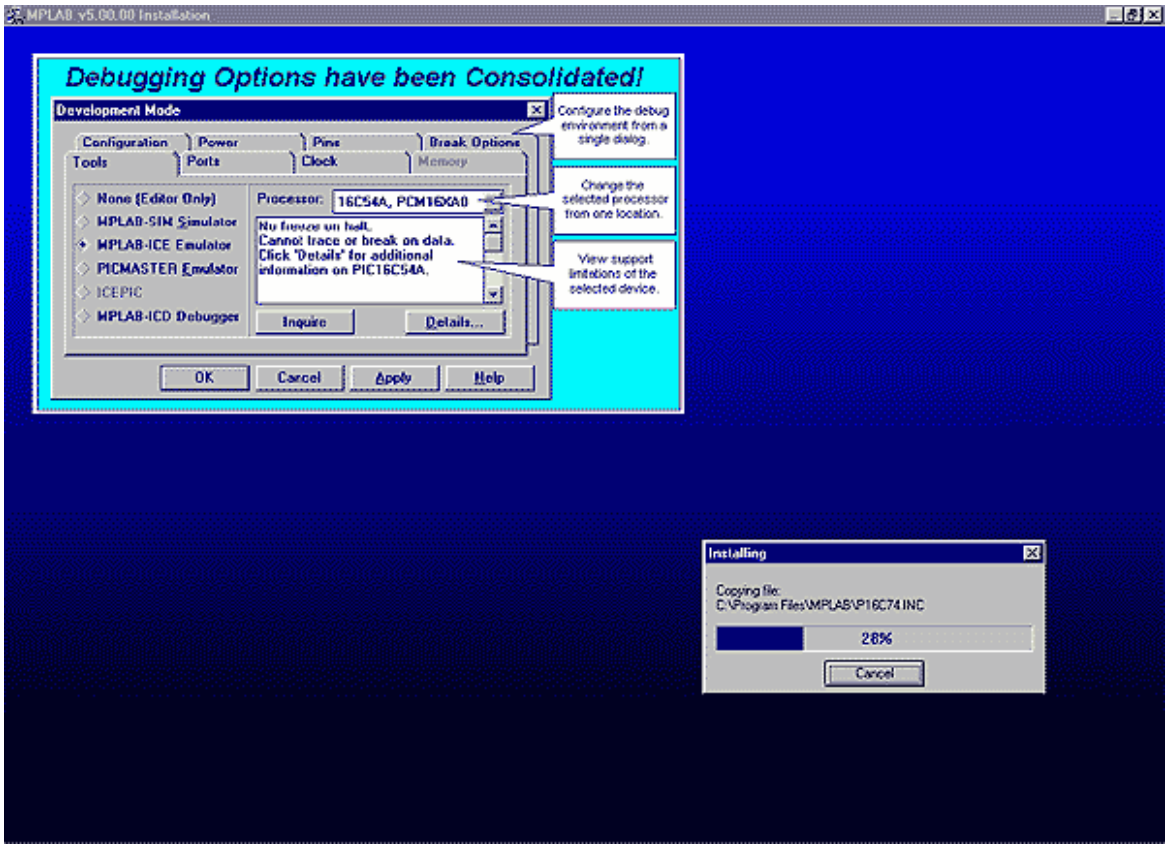
Selectând un director pentru fi•ierele de sistem

Dup• pa•ii de mai sus, instalarea începe f•când clic pe 'Next'.



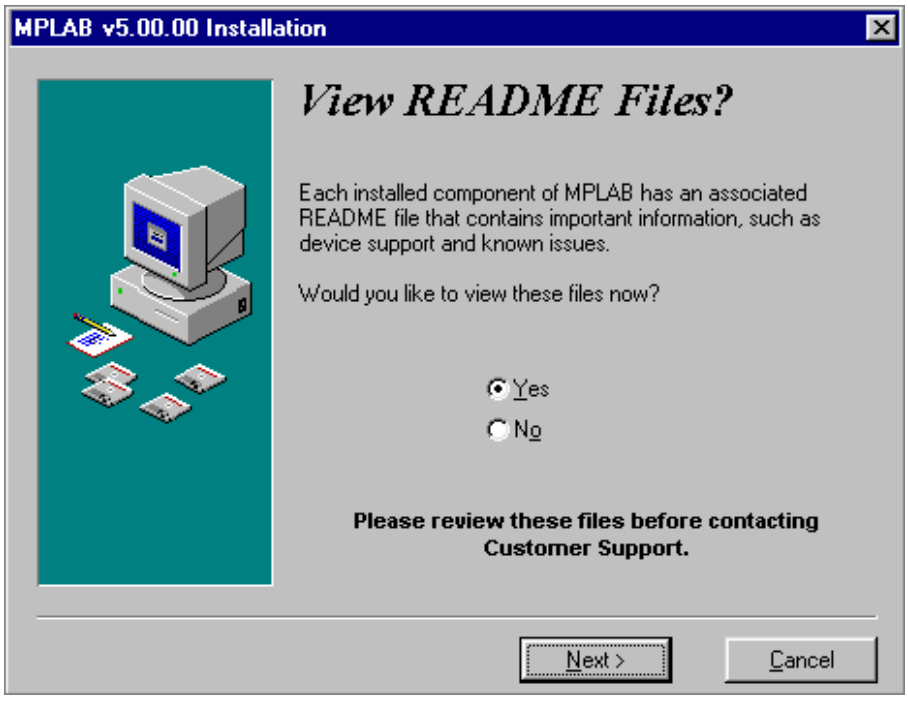
Ecran anterior instal•rii

Instalarea nu dureaz• mult, •i procesul copierii fi•ierelor poate fi v•zut într-o fereastr• mic• în col•ul din dreapta ecranului.



Desf••urarea instal•rii

Dup• ce instalarea este gata, sunt dou• ecrane de dialog, unul pentru informa•ia de ultim moment privind versiunile programului •i corec•iile, iar cel•lalt este un ecran de binevenit. Dac• s-au deschis fi•ierele text (Readme.txt), ele trebuie închise.



Informa•ii de ultim moment privind versiunile programului •i corec•iile

F•când clic pe Finish, instalarea MPLAB este terminat•.

CAPITOLUL 6

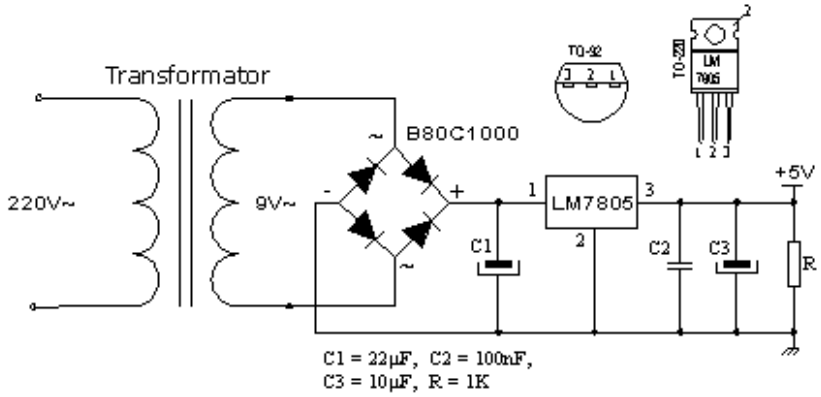
Mostrele

Introducere

Exemplele oferite în aceast capitol v• vor ar•ta cum s• conecta•i microcontrolerul PIC cu alte componente sau dispozitive periferice când produce•i propriul sistem bazat pe microcontroler. Fiecare exemplu con•ine descriere detaliat• a p•r•ii hardware cu schema electric• •i comentarii despre program. Toate programele pot fi luate direct din prezentarea de pe internet „MikroElektronika”.

Alimentarea microcontrolerului

În general, alimentarea corect• este de o importan•• maxim• pentru func•ionarea corect• a sistemului cu microcontroler. Poate fi u•or comparat• cu respira•ia unui om în aer. Este mai probabil ca un om care respir• în aer curat va tr•i mai mult decât un om care locuie•te într–un mediu poluat. Pentru o func•ionare corect• a oric•rui microcontroler, este necesar s• oferim o surs• stabil• de alimentare, un reset sigur în momentul în care îl porni•i •i un oscilator. Conform specifica•iilor tehnice oferite de produc•torul microcontrolerului PIC, tensiunea de alimentare ar trebui s• se încadreze între 2.0V •i 6.0V pentru toate versiunile. Cea mai simpl• solu•ie este folosirea stabilizatorului de tensiune LM7805 care ofer• tensiune stabil• de +5V la ie•ire. O astfel de surs• este ilustrat• în figura de mai jos.



Pentru a func•iona corect sau pentru a avea o tensiune stabilizat• la 5V la ie•ire (pinul 3), tensiunea de intrare pe pinul 1 la LM7805 ar trebui s• fie între 7V •i 24V. În func•ie de curentul consumat de montaj vom folosi tipul corespunz•tor de stabilizator de tensiune LM7805. Sunt diferite versiuni de LM7805. Pentru consum de curent de pân• la un 1A ar trebui s• folosim versiunea în capsul• TO-220 cu posibilitatea de r•cire adi•ional•. Dac• consumul total este de 50mA, putem s• folosim 78L05 (versiune de stabilizator în capsul• mic• TO-92 pentru curent de pân• la 100mA).

Pagina anterioar•	Con•inut	Pagina urm•toare

Anexa A

Set Instruc•iuni

Introducere

Anexa con•ine toate instruc•iunile prezentate separat cu exemple pentru folosirea lor. Sintaxa, descrierea •i efectele ei asupra st•rii bi•ilor sunt pentru fiecare instruc•iune.

A.1 MOVLW Scrie constanta în registrul W

Sintaxă: [*label*] MOVLW **k**
Descriere: constanta **k** de 8 biţi este scrisă în registrul **W.**
Operare: **k** ⇒ (**W**)
Operand: 0 ≤ **k** ≤ 255
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 MOVLW 0x5A

După instrucţiune: W=0x5A

Exemplu 2 MOVLW REGISTAR

Înainte de instrucţiune: W=0x10 şi REGISTAR=0x40
După instrucţiune: W=0x40

A.2 MOVWF Copiaz• W în f

Sintaxă: [*label*] MOVLW **f**
Descriere: conţinutul registrului **W** este copiat în registrul **f.**
Operaţie: **W** ⇒ (**f**)
Opeand: 0 ≤ **f** ≤ 127
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 MOVWF OPTION _REG

Înainte de instrucţiune: OPTION _REG=0x20
W=0x40
După instrucţiune: OPTION _REG=0x40W
W=0x40

Exemplu 2 MOVWF INDF

Înainte de instrucţiune: W=0x17
FSR=0xC2
conţinutul adresei 0xC2=0x00
După instrucţiune: W=0x17
FSR=0xC2
conţinutul adresei 0xC2=0x17

A.3 MOVF Copiaz• f în d

Sintaxă: [*label*] MOVF **f,d**
Descriere: Conţinutul registrului **f** este memorat în locaţia determinată de operandul **d**.
Dacă **d=0**, destinaţia este registrul **W**
Dacă **d=1**, destinaţia este registrul **f** însuşi
Opţiunea **d=1** este folosită pentru testarea conţinutului registrului **f** pentru că execuţia acestei instrucţiuni afectează eticheta Z în registrul STATUS.

Operaţie: **f** ⇒ (**d**)
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 MOVF FSR, 0

Înainte de instrucţiune: FSR=0xC2
W=0x00
După instrucţiune: W=0xC2
Z=0

Exemplu 2 MOVF INDF, 0

Înainte de instrucţiune: W=0x17
FSR=0xC2
conţinutul adresei 0xC2=0x00
După instrucţiune: W=0x17
FSR=0xC2
conţinutul adresei 0xC2=0x00
Z=1

A.4 CLRW Scrie 0 în W

Sintaxă: [*label*] CLRW
Descriere: conţinutul registrului **W** face ieşirea zero, şi eticheta Z în registrul STATUS este setat la unu.

Operaţie: 0⇒ (**W**)
Operand: -
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu CLRW

Înainte de instrucţiune: W=0x55
După instrucţiune: W=0x00
Z=1

A.5 CLRF Scrie 0 în f

Sintaxă: [*label*] CLRF **f**
Descriere: conţinutul registrului '**f**' face ieşirea zero, şi eticheta Z în registrul STATUS este setată la unu

Operaţie: 0 ⇒ **f**
Operand: 0 ≤ **f** ≤ 127
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 CLRF STATUS

Înainte de instrucţiune: STATUS=0xC2
După instrucţiune: STATUS=0x00
Z=1

Exemplu 2 CLRF INDF

Înainte de instrucţiune: FSR=0xC2
conţinutul adresei 0xC2=0x33
După instrucţiune: FSR=0xC2
conţinutul adresei 0xC2=0x00
Z=1

A.6 SWAPF Copiaz• the buc••elele din f în d în diagonal•

Sintaxă: [*label*] SWAPF **f**, **d**
Descriere: Jumătatea de sus şi de jos a registrului **f** îşi schimbă locurile
Dacă **d**=0, rezultatul este memorat în registrul **W**
Dacă **d**=1, rezultatul este memorat în registrul **f**

Operaţie: **f**<0:3> ⇒ **d**<4:7>, **f**<4:7> ⇒ **d**<0:3>;
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 SWAP REG, 0

Înainte de instrucţiune: REG=0xF3
După instrucţiune: REG=0xF3
W=0x3F

Exemplu 2 SWAP REG, 1

Înainte de instrucţiune: REG=0xF3
După instrucţiune: REG=0x3F

A.7 ADDLW Adun• W la o constant•

Sintaxă: [*label*] ADDLW **k**
Descriere: Conţinutul registrului **W** este adunat la o constantă de 8 biţi k şi rezultatul este memorat în registrul **W**.
Operaţie: (**W**) + **k** ⇒ **W**
Operand: 0 ≤ **k** ≤ 255
Etichetă: C, DC, Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 ADDLW 0x15

Înainte de instrucţiune: W=0x10
După instrucţiune: W=0x25

Exemplu 2 ADDLW REG

Înainte de instrucţiune: W=0x10
conţinutul registrului REG=0x37
După instrucţiune: W=0x47

A.8 ADDWF Adun• W la f

Sintaxă: [*label*] ADDWF **f**, **d**
Descriere: Adună conţinutul registrului **W** la registrul **f**.
Dacă **d**=0, rezultatul este memorat în registrul **W**
Dacă **d**=1, rezultatul este memorat în registrul **f**

Operaţie: (W) + (f) ⇒ d
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: C, DC, Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 ADDWF FSR, 0

Înainte de instrucţiune: W=0x17
FSR=0xC2
După instrucţiune: W=0xD9
FSR=0xC2

Exemplu 2 ADDLW INDF, 1

Înainte de instrucţiune: W=0x17
FSR=0xC0
conţinutul adresei 0xC2=0x20
După instrucţiune: W=0x17
FSR=0xC2
conţinutul adresei 0xC2=0x37

A.9 SUBLW Scade W dintr-o constant•

Sintaxă: [*label*] SUBLW **k**
Descriere: Conţinutul registrului **W** este scăzut din constanta **k**, şi rezultatul este memorat în registrul **W**.
Operaţie: **k** - (**W**) ⇒ **W**
Operand: 0 ≤ **f** ≤ 255
Etichetă: C, DC, Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 SUBLW 0x03

Înainte de instrucţiune: W=0x01, C=x, Z=x
După instrucţiune: W=0x02, C=1, Z=0 Result > 0

Înainte de instrucţiune: W=0x03, C=x, Z=x
După instrucţiune: W=0x00, C=1, Z=1 Result = 0

Înainte de instrucţiune: W=0x04, C=x, Z=x
După instrucţiune: W=0xFF, C=0, Z=0 Result < 0

Exemplu 2 SUBLW REG

Înainte de instrucţiune: W=0x10
conţinutul REG=0x37
După instrucţiune: W=0x27
C=1 Result > 0

A.10 SUBWF Scade W din f

Sintaxă: [*label*] SUBLW **f**, **d**
Descriere: Conţinutul registrului **W** este scăzut din conţinutul registrului **f**
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Operaţie: (**f**) - (**W**) ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: C, DC, Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 SUBLW REG, 1

Înainte de instrucţiune: REG=3, W=2, C=x, Z=x
După instrucţiune: REG=1, W=2, C=1, Z=0 Result > 0

Înainte de instrucţiune: REG=2, W=2, C=x, Z=x
După instrucţiune: REG=0, W=2, C=1, Z=1 Result = 0

Înainte de instrucţiune: REG=1, W=2, C=x, Z=x
După instrucţiune: REG=0xFF, W=2, C=0, Z=0 Result < 0

A.11 ANDLW W AND(•I) logic cu o constant•

Sintaxă: [*label*] ADDWF **k**
Descriere: Face operaţia AND(ŞI) logic asupra conţinutului registrului **W** şi constantei **k**.
Rezultatul este memorat în registrul **W**.
Operaţie: (**W**) .AND. **k** ⇒ **W**
Operand: 0 ≤ **f** ≤ 255
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 ANDLW 0x5F

```
Înainte de instrucţiune:    W=0xA3      ; 0101 1111 (0x5F)
După instrucţiune:          W=0x03      ; 1010 0011 (0xA3)
                                        ------------------
                                        ; 0000 0011 (0x03)
```

Exemplu 2 ANDLW REG

```
Înainte de instrucţiune:    W=0xA3      ; 1010 0011 (0xA3)
                            REG=0x37    ; 0011 0111 (0x37)
După instrucţiune:          W=0x23      ------------------
                                        ; 0010 0011 (0x23)
```

A.12 ANDWF W AND(•I) logic cu f

Sintaxă: [*label*] ANDWF **f**, **d**
Descriere: Face operaţia AND(ŞI) logic asupra conţinutului registrelor **W** şi **f**.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**
Operaţie: (**W**) .AND. **f** ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 ANDWF FSR, 1

```
Înainte de instrucţiune:    W=0x17, FSR=0xC2        ; 0001 0111 (0x17)
După instrucţiune:          W=0x17, FSR=02          ; 1100 0010 (0xC2)
                                                    ------------------
                                                    ; 0000 0010 (0x02)
```

Exemplu 2 ANDLW FSR, 0

```
Înainte de instrucţiune:    W=0x17, FSR=0xC2        ; 0001 0111 (0x17)
După instrucţiune:          W=0x02, FSR=0xC2        ; 1100 0010 (0xC2)
                                                    ------------------
                                                    ; 0000 0010 (0x02)
```

A.13 IORLW W OR(SAU) logic cu o constant•

Sintaxă: [*label*] IORLW **k**
Descriere: Face operaţia lOR(SAU) logic asupra conţinutului registrului **W** şi asupra constantei **k** de 8 biţi, şi rezultatul este memorat în registrul **W**.
Operaţie: (**W**) .OR. (**k**) ⇒ **W**
Operand: 0 ≤ **k** ≤ 255
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 IORLW 0x35

Înainte de instrucţiune: W=0x9A
După instrucţiune: W=0xBF
Z=0

Exemplu 2 IORLW REG

Înainte de instrucţiune: W=0x9A
conţinut REG=0x37
După instrucţiune: W=0x9F
Z=0

A.14 IORWF W OR(SAU) logic cu f

Sintaxă: [*label*] IORWF **f**, **d**
Descriere: Face operaţia OR(SAU) logic asupra conţinutului registrelor **W** şi **f**.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**
Operaţie: (**W**) .OR.(**f**) ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 IORWF REG, 0

Înainte de instrucţiune: REG=0x13, W=0x91
După instrucţiune: REG=0x13, W=0x93
Z=0

Exemplu 2 IORWF REG, 1

Înainte de instrucţiune: REG=0x13, W=0x91
După instrucţiune: REG=0x93, W=0x91
Z=0

A.15 XORLW W OR(SAU) logic exclusiv cu o constant•

Sintaxă: [*label*] XORLW **k**
Descriere: Face operaţia OR(SAU) exclusiv asupra conţinutului registrului **W** şi constantei **k**, şi rezultatul este memorat în registrul **W**.
Operaţie: (**W**) .XOR. **k** ⇒ **W**
Operand: 0 ≤ **k** ≤ 255
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 XORLW 0xAF

```
Înainte de instrucţiune:    W=0xB5      ; 1010 1111 (0xAF)
După instrucţiune:          W=0x1A      ; 1011 0101 (0xB5)
                                        -------------------
                                        ; 0000 0011 (0x1A)
```

Exemplu 2 XORLW REG

```
Înainte de instrucţiune:    W=0xAF      ; 1010 0011 (0xA3)
                            REG=0x37    ; 0011 0111 (0x37)
După instrucţiune:          W=0x18      --------------------
                            Z=0         ; 0001 1000 (0x18)
```

A.16 XORWF W logic exclusiv OR(SAU) cu f

Sintaxă: [*label*] XORWF **f**, **d**
Descriere: Face operaţia OR(SAU) logic exclusiv asupra conţinutului registrelor **W** şi **f**.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**
Operaţie: (**W**) .XOR. (**f**) ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ε [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 XORWF REG, 1

```
Înainte de instrucţiune:    W=0xAF, W=0x85   ; 1010 1111 (0xAF)
După instrucţiune:          W=0x1A, W=0x85   ; 1100 0101 (0xB5)
                                             -------------------
                                             ; 0001 1010 (0x1A)
```

Exemplu 2 XORWF REG, 0

```
Înainte de instrucţiune:    REG=0xAF, W=0x85      ;  1010 1111 (0xAF)
După instrucţiune:          REG=0xAF, W=0x1A      ;  1011 0101 (0xB5)
                                                  --------------------
                                                  ; 0001 1010 (0x1A)
```

A.17 INCF Incrementeaz• f

Sintaxă: [*label*] INCF **f**, **d**
Descriere: Incremnetează registrul **f** cu unu.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Operaţie: (**f**) + 1 ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 INCF REG, 1

Înainte de instrucţiune: REG=0xFF
Z=0
După instrucţiune: REG=0x00
Z=1

Exemplu 2 INCF REG, 0

Înainte de instrucţiune: REG=0x10
W=x
Z=0
După instrucţiune: REG=0x10
W=0x11
Z=0

A.18 DECF Decrementeaz• f

Sintaxă: [*label*] DECF **f**, **d**
Descriere: Decremnetează registrul **f** cu unu.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Operaţie: (**f**) - 1 ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 DECF REG, 1

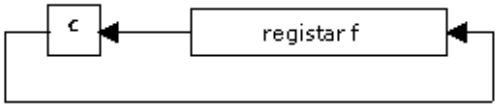
Înainte de instrucţiune: REG=0x01
Z=0
După instrucţiune: REG=0x00
Z=1

Exemplu 2 DECF REG, 0

Înainte de instrucţiune: REG=0x13
W=x
Z=0
După instrucţiune: REG=0x13
W=0x12
Z=0

A.19 RLF Rote•te f la stânga prin CARRY

Sintaxă: [*label*] RLF **f**, **d**
Descriere: Conţinutul registrului **f** este rotit un spaţiu la stânga prin eticheta C (Carry).
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Operaţie: (**f**<n>) **d**<n+1>, **f**<7> ⇒ C, C ⇒ **d**<0>;
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: C
Număr de cuvinte: 1
Număr de cicluri: 1



Exemplu 1 RLF REG, 0

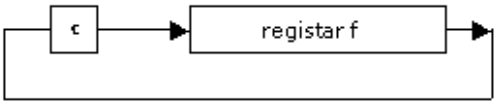
Înainte de instrucţiune: REG=1110 0110
C=0
După instrucţiune: REG=1110 0110
W=1100 1100
C=1

Exemplu 2 RLF REG, 1

Înainte de instrucţiune: REG=1110 0110
C=0
După instrucţiune: REG=1100 1100
C=1

A.20 RRF Rote•te f la dreapta prin CARRY

Sintaxă: [*label*] RRF **f**, **d**
Descriere: Conţinutul registrului **f** este rotit un spaţiu la dreapta prin eticheta C (Carry).
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Operaţie: (**f**<n>) ⇒ **d**<n-1>, **f**<0> ⇒ C, C ⇒ **d**<7>;
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: C
Număr de cuvinte: 1
Număr de cicluri: 1



Exemplu 1 RRF REG, 0

Înainte de instrucţiune: REG=1110 0110
W=x
C=0
După instrucţiune: REG=1110 0110
W=0111 0011
C=0

Exemplu 2 RRF REG, 1

Înainte de instrucţiune: REG=1110 0110
C=0
După instrucţiune: REG=0111 0011
C=0

A.21 COMF Complement f

Sintaxă: [*label*] COMF **f**, **d**
Descriere: Conţinutul registrului **f** este complementat.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Operaţie: (**f**) ⇒ **d**
Operand: 0 ≤ **f** ≤ 127
d ∈ [0,1]
Etichetă: Z
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 COMF REG, 0

```
Înainte de instrucţiune:    REG=0x13    ; 0001 0011 (0x13)
După instrucţiune:          REG=0x13    ; complement
                            W=0xEC      ---------------------
                                        ; 1110 1100 (0xEC)
```

Exemplu 2 COMF INDF, 1

```
Înainte de instrucţiune:    FSR=0xC2
                            conţinutul adresei (FSR)=0xAA
După instrucţiune:          FSR=0xC2
                            conţinutul adresei (FSR)=0x55
```

A.22 BCF Reseteaz• bitul b în f

Sintaxă: [*label*] BCF **f**, **b**
Descriere: Resetează bitul **b** în registrului **f**.
Operaţie: (0) ⇒ **f**<**b**>
Operand: 0 ≤ **f** ≤ 127
0 ≤ **b** ≤ 7
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 BCF REG, 7

```
Înainte de instrucţiune:    REG=0xC7    ; 1100 0111 (0xC7)
După instrucţiune:          REG=0x47    ; 0100 0111 (0x47)
```

Exemplu 2 BCF INDF, 3

```
Înainte de instrucţiune:    W=0x17
                            FSR=0xC2
                            conţinutul adresei (FSR)=0x2F
După instrucţiune:          W=0x17
                            FSR=0xC2
                            conţinutul adresei (FSR)=0x27
```

A.23 BSF Seteaz• bitul b în f

Sintaxă: [*label*] BSF **f**, **b**
Descriere: Setează bitul **b** în registrului **f**.
Operaţie: 1 ⇒ **f**<**b**>
Operand: 0 ≤ **f** ≤ 127
0 ≤ **b** ≤ 7
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu 1 BSF REG, 7

```
Înainte de instrucţiune:    REG=0x07    ; 0000 0111 (0x07)
După instrucţiune:          REG=0x17    ; 1000 0111 (0x17)
```

Exemplu 2 BSF INDF, 3

```
Înainte de instrucţiune:    W=0x17
                            FSR=0xC2
                            conţinutul adresei (FSR)=0x20
După instrucţiune:          W=0x17
                            FSR=0xC2
                            conţinutul adresei (FSR)=0x28
```

A.24 BTFSC Testeaz• bitul b în f, sari dac• = 0

Sintaxă: [*label*] BTFSC **f**, **b**
Descriere: Dacă bitul **b** în registrului **f** este egal cu zero, sărim următoarea instrucţiune.
Dacă bitul **b** este egal cu zero, în timpul execuţiei instrucţiunii curente, execuţia următoarei instrucţiuni este dezactivată, şi se execută instrucţiunea NOP în schimb, făcând astfel din cea curentă o instrucţiune de două cicluri.
Operaţie: Sari următoarea instrucţiune dacă (**f**<**b**>)=0
Operand: 0 ≤ **f** ≤ 127
0 ≤ **b** ≤ 7
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1 sau 2 depinzând de bitul **b**

Exemplu

```
LAB _01     BTFSC  REG, 1      ; Testează bitul nr.1 în REG
LAB_02      . . . . . .        ; Sari această linie dacă=0
LAB_03      . . . . . .        ; Sari aici dacă=1
```

Înainte de instrucţiune, contorul programului a fost la adresa LAB _01.

După instrucţiune, dacă primul bit în registrul REG a fost zero, contorul programului indică spre adresa LAB_03.

Dacă primul bit în registrul REG a fost unu, contorul programului indică spre adresa LAB_02.

A.25 BTFSS Testeaz• bitul b în f, sari dac• =1

Sintaxă: [*label*] BTFSS **f**, **b**
Descriere: Dacă bitul **b** în registrului **f** este egal cu unu, atunci sărim următoarea instrucţiune.
Dacă bitul **b** este egal cu unu, în timpul execuţiei instrucţiunii curente, următoarea este dezactivată, şi se execută instrucţiunea NOP în schimb, făcând astfel din cea curentă o instrucţiune de două cicluri.
Operaţie: Sari următoarea instrucţiune dacă (**f**<**b**>)=1
Operand: 0 ≤ **f** ≤ 127
0 ≤ **b** ≤ 7
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1 sau 2 depinzând de bitul **b**

Exemplu

```
LAB _01     BTFSS  REG, 1      ; Testează bitul nr.1 în REG
LAB_02      . . . . . .        ; Sari această linie dacă=1
LAB_03      . . . . . .        ; Sari aici dacă=0
```

Înainte de instrucţiune, contorul programului a fost la adresa LAB _01.

După instrucţiune, dacă primul bit în registrul REG a fost unu, contorul programului indică spre adresa LAB_03.

Dacă primul bit în registrul REG a fost zero, contorul programului indică spre adresa LAB_02.

A.26 INCFSZ Incrementeaz• f, sari dac•=0

Sintaxă: [*label*] INCFSZ **f**, **d**
Descriere: Conţinutul registrului **f** este incrementat cu unu.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Dacă rezultatul =0, următoarea instrucţiune este executată ca NOP făcând din cea curentă o instrucţiune de două cicluri.
Operaţie: (**f**) + 1 ⇒ d
Operand: 0 ≤ **f** ≤ 127
d € [0,1]
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1 sau 2 depinzând de rezultat

Exemplu

```
LAB _01     INCFSZ  REG, 1      ; incrementează  conţinutul REG cu unu
LAB_02      . . . . . .         ; Sari această linie dacă=0
LAB_03      . . . . . .         ; Sari aici dacă=0
```

Conţinutul acestui contor de program înainte de instrucţiune, PC=adresa LAB _ 01

Conţinutul registrului REG după executarea unei instrucţiuni, REG=REG+1, dacă REG=0, contorul programului indică spre adresa următoarei instrucţiuni LAB_03. Altfel, contorul programului indică spre adresa următoarei instrucţiuni sau spre LAB_02.

A.27 DECFSZ Decrementeaz• f, sari dac• = 0

Sintaxă: [*label*] DECFSZ **f**, **d**
Descriere: Conţinutul registrului **f** este decrementat cu unu.
Dacă **d**=0, rezultatul este memorat în registrul **W**.
Dacă **d**=1, rezultatul este memorat în registrul **f**.
Dacă rezultatul =0, următoarea instrucţiune este executată ca NOP
făcând din cea curentă o instrucţiune de două-cicluri.
Operaţie: (**f**) - 1 ⇒ d
Operand: 0 ≤ **f** ≤ 127
d € [0,1]
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1 sau 2 depinzând de rezultat

Exemplu

```
LAB _01     DECFSZ  CNT, 1     ; decrementează  conţinutul REG cu unu.
LAB_02      . . . . . .        ; Sari această linie dacă=0
LAB_03      . . . . . .        ; Sari aici dacă=1
```

Conţinutul contorului de program înainte de instrucţiune, PC=adresa LAB _ 01

Conţinutul acestui registrului CNT după executarea unei instrucţiuni, CNT=CNT-1, dacă CNT=0, contorul programului indică spre adresa etichetei LAB_03. Altfel, contorul programului indică spre adresa următoarei instrucţiuni, sau spre LAB_02.

A.28 GOTO Salt la adres•

Sintaxă: [*label*] GOTO **k**
Descriere: Salt necondiţionat la adresa **k**.
Operaţie: **k** ⇒ PC<10:0>, (PCLATH<4:3>) ⇒ PC<12:11>
Operand: 0 ≤ **k** ≤ 2048
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 2

Exemplu

```
LAB 00      GOTO LAB_ 01      ; Salt la LAB_01
                :
                :
LAB_01        . . . . . .
```

Înainte de instrucţiune: PC=adresa LAB_00
După instrucţiune: PC=adresa LAB_01

A.29 CALL Apeleaz• un program

Sintaxă: [*label*] CALL **k**
Descriere: Instrucţiunea apelează un program. Mai întâi, adresa de întoarcere (PC+1) este memorată în stivă, apoi operandul direct pe 11 biţi **k**, care conţine adresa subprogramului, este memorat în contorul programului.
Operaţie: (PC)+1 ⇒ Vârful stivei (TOS)
k ⇒ PC<10:0>, (PCLATH<4:3>) ⇒ PC<12:11>
Operand: 0 ≤ **k** ≤ 2048
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 2

Exemplu

```
LAB 01      CALL LAB_ 02      ; Apelează subrutina LAB_02
                  :
                  :
LAB_02        . . . . . .
```

Înainte de instrucţiune: PC=adresa LAB_01
TOS=x
După instrucţiune: PC=adresa LAB_02
TOS=LAB_01

A.30 RETURN Întoarcere dintr-un subprogram

Sintaxă: [*label*] RETURN
Descriere: Conţinutul vârfului stivei este memorat în contorul programului.
Operaţie: TOS ⇒ contorul programului PC
Operand: -
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 2

Exemplu RETURN

Înainte de instrucţiune: PC=x
TOS=x
După instrucţiune: PC=TOS
TOS=TOS-1

A.31 RETLW Întoarcere dintr-un subprogram cu constant• în W

Sintaxă: [*label*] RETLW **k**
Descriere: Constanta de 8 biţi **k** este memorată în registrul **W**. Valoarea vârfului stivei este memorat în contorul programului.
Operaţie: (**k**) ⇒ **W**; TOS ⇒ PC
Operand: 0 ≤ **k** ≤ 255
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 2

Exemplu RETLW 0x43

Înainte de instrucţiune: W=x
PC=x
TOS=x
După instrucţiune: W=0x43
PC=TOS
TOS=TOS-1

A.32 RETFIE Întoarcere dintr-o rutin• de întrerupere

Sintaxă: [*label*] RETFIE
Descriere: Întoarcere dintr-un subprogram. Valoarea din TOS este memorată în contorul programului PC. Întreruperile sunt activate prin setarea unui bit GIE (Global interrupt enable-activează întreruperea globală).
Operaţie: TOS ⇒ PC; 1 ⇒ GIE
Operand: -
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 2

Exemplu RETFIE

Înainte de instrucţiune: PC=x
GIE=0
După instrucţiune: PC=TOS

A.33 NOP F•r• opera•ii

Sintaxă: [*label*] NOP
Descriere: Nu execută nici o operaţie nici nu afectează vreo etichetă.
Operaţie: -
Operand: -
Etichetă: -
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu NOP

Înainte de instrucţiune: PC=x
După instrucţiune: PC=x+1

A.34 CLRWDT Ini•ializeaz• timer-ul watchdog

Sintaxă: [*label*] CLRWDT
Descriere: Timer-ul watchdog este resetat. Prescalerul timer-ului watchdog este de asemenea resetat, şi biţii de stare TO şi PD sunt setaţi de asemenea.
Operaţie: 0 ⇒ WDT
0 ⇒ WDT prescaler
1 ⇒ TO
1 ⇒ PD
Operand: -__ __
Etichetă: TO, PD
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu CLRWDT

Înainte de instrucţiune: contor WDT=x
prescaler=1 :128
După instrucţiune: contor WDT=0x00
prescaler contor=0
TO=1
PD=1
prescaler WDT=1:128

A.35 SLEEP Modul stand by

Sintaxă: [*label*] SLEEP
Descriere: Procesorul intră în modul consum redus. Oscilatorul este oprit. Bitul de stare PD (Power Down-fără alimentare) este resetat. Bitul TO (Timer Out) este setat. WDT (Watchdog Timer) şi prescalerul lui sunt resetate.
Operaţie: 0 ⇒ WDT
0 ⇒ WDT prescaler
1 ⇒ TO
0 ⇒ PD
Operand: -__ __
Etichetă: TO, PD
Număr de cuvinte: 1
Număr de cicluri: 1

Exemplu SLEEP

Înainte de instrucţiune: contor WDT=x
prescaler WDT=x
După instrucţiune: contor WDT=0x00
prescaler WDT=0
TO=1
PD=0

Pagina anterioar• | Con•inut | Pagina urm•toare

Anexa B

Sisteme numerice

Introducere

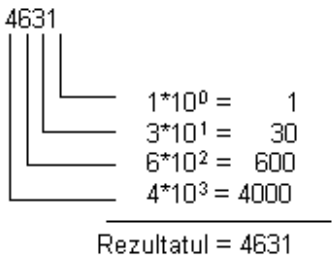
A fost dificil pentru oameni s• accepte faptul c• unele lucruri difer• de ei •i de modul lor de gândire. Acesta este probabil unul din motivele pentru care sistemele numerice care difer• de cele zecimale sunt înc• greu de în•eles. Totu•i, fie c• le vrem ori nu, realitatea este diferit•. Sistemul numeric zecimal pe care oamenii îl folosesc în via•a de fiecare zi este de departe în urma sistemului binar folosit de milioane de calculatoare în lumea întreag•.

Fiecare sistem numeric se bazeaz• pe o funda•ie. La un sistem numeric zecimal, baza este 10, la binar 2, •i la sistemul hexazecimal 16. Valoarea fiec•rui zecimal este determinat• de pozi•ia lui în rela•ie cu întreg num•rul reprezentat în sistemul numeric dat. Suma valorilor fiec•rui zecimal d• valoarea întregului num•r. Sistemele binare •i hexazecimale sunt în special interesante pentru subiectul acestei c•r•i. În afar• de acestea vom discuta de asemenea un sistem zecimal, pentru a-l compara cu celelalte dou•. Chiar dac• un sistem numeric zecimal este un subiect cu care suntem bine familiariza•i, îl vom discuta aici din cauza leg•turii sale cu alte sisteme numerice.

B.1 Sistem numeric zecimal

Sistemul numeric zecimal este definit de baza lui 10 •i de spa•iul zecimal care este num•rat de la dreapta la stânga, •i const• din numerele 1, 2, 3, 4, 5, 6, 7, 8, 9. Aceasta înseamn• c• num•rul din cap•tul din stânga a sumei totale este multiplicat cu 1, urm•torul cu 10, urm•torul cu 100, etc.

Exemplu:



Opera•iile de adunare, sc•dere, împ•r•ire •i înmul•ire într-un sistem numeric zecimal sunt folosite într-un fel care ne este deja cunoscut, a•a c• nu-l vom discuta mai departe.

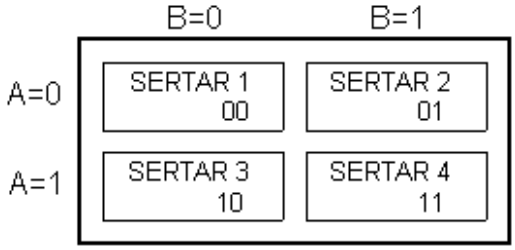
B.2 Sistem numeric binar

Sistemele numerice binare difer• în multe aspecte de sistemul zecimal pe care îl folosim în via•a de zi cu zi. Baza lui numeric• este 2, •i fiecare num•r poate avea doar dou• valori, '1' sau '0'. Sistemul numeric binar este folosit în calculatoare •i microcontrolere pentru c• este de departe mai potrivit pentru procesare decât un sistem zecimal. Uzual, num•rul binar const• din numerele 8, 16, sau 32, •i nu este important având în vedere con•inutul de a discuta de ce. Este destul acum de a accepta aceast• informa•ie.

Exemplu:

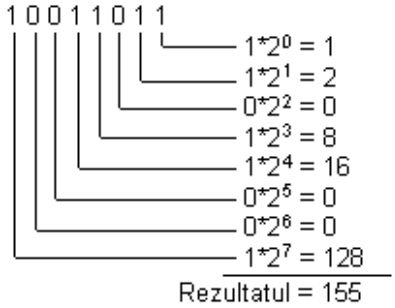
10011011 num•r binar cu 8 digi•i

Pentru a în•elege logica numerelor binare, vom considera un exemplu. S• spunem c• avem un mic dul•pior cu patru sertare, •i c• trebuie s• spunem cuiva s• ne aduc• ceva din unul din sertare. Nimic nu este mai simplu, vom spune partea stâng•, jos (sertarul), •i sertarul dorit este clar definit. Totu•i, dac• ar fi trebuit s• facem aceasta f•r• folosirea instruc•iunilor ca stânga, dreapta, jos, sus, etc., atunci am fi avut o problem•. Sunt multe solu•ii la aceast• problem•, dar noi ar trebui s• c•ut•m una care este cea mai benefic• •i practic•! S• desemn•m rândurile cu A, •i tipurile cu B. Dac• A=1, se refer• la rândul de sus a sertarelor, •i pentru A=0, rândul de jos. Similar cu coloanele, B=1 reprezint• coloana stâng•, •i B=0, dreapt• (urm•toarea imagine). Acum este deja mult mai u•or de a explica din care sertar avem nevoie de ceva. Trebuie doar s• formul•m una din cel patru combina•ii: 00, 01, 10 sau 11. Aceast• caracteristic• denumind fiecare sertar individual nu este decât reprezentarea numeric• binar•, sau conversia numerelor comune dintr-o form• zecimal• într-una binar•. Cu alte cuvinte, referirile ca "primul, al doilea, al treilea •i al patrulea" sunt schimbate cu "00, 01, 10 •i 11".



Ceea ce ne r•mâne este s• ne acomod•m cu logica care este folosit• la sistemul numeric binar, sau cu cum s• ob•inem o valoare numeric• dintr-o serie de zero-uri •i unu-uri într-un fel în care s•-l în•elegem, bineîn•eles. Aceast• procedur• se nume•te conversia dintr-un num•r binar într-unul zecimal.

Exemplu:



Dup• cum pute•i vedea, convertirea unui num•r binar într-unul zecimal se face prin calcularea expresiei din partea stâng•. Depinzând de pozi•ie într-un num•r binar, cifrele poart• diferite valori care sunt multiplicate cu ele însele, •i prin ad•ugarea lor ob•inem un num•r zecimal pe care îl putem în•elege. S• presupunem mai departe c• în fiecare sertar sunt câteva bile: 2 în primul, 4 în al doilea sertar, 7 în al treilea •i 3 în al patrulea sertar. S• spunem de asemenea celui care deschide sertarele s• foloseasc• reprezentarea binar• ca r•spuns. În aceste condi•ii, întrebarea ar fi: "Câte bile sunt în 01?", •i r•spunsul va fi: "Sunt 100 de bile în 01." Trebuie remarcat c• atât întrebarea cât •i r•spusul sun foarte clare chiar dac• nu am folosit nume standard. Trebuie mai departe de observat c• pentru numerele zecimale de la 0 la 3 este suficient de a avea dou• cifre binare, •i c• pentru toate valorile de mai sus trebuie s• ad•ug•m cifre binare noi. A•a c•, pentru numere de la 0 la7 este suficient s• avem trei cifre, pentru numere de la 0 la 15, patru, etc. Mai simplu spus, cel mai mare num•r ce poate fi reprezentat de o cifr• binar• este cel ob•inut când baza 2 este gradat• cu un num•r de cifre binare într-un num•r binar •i astfel num•rul derivat este decrementat cu unu.

Exemplu:

$$2^4 - 1 = 16 - 1 = 15$$

Aceasta înseamn• c• este posibil de a se reprezenta numere zecimale de la 0 la 15 cu 4 cifre binare, incluzând numerele '0' •i '15', sau 16 valori diferite.

Opera•iile ce exist• în sistemul numeric zecimal exist• de asemenea într-un sistem binar. Din motive de claritate •i descifrabilitate, vom revedea adunarea •i sc•derea doar în acest capitol.

Regulile de baz• care se aplic• adun•rii binare sunt:

$$\begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \end{array} \qquad \begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \end{array}$$

Adunarea se face a•a încât cifrele din aceea•i pozi•ie nmeric• sunt adunate, similar sistemuluii numeric zecimal. Dac• ambii digi•i de adunat sunt zero, suma lor r•mâne zero, •i dac• sunt '0' •i '1', rezultatul este '1'. Suma a doi de unu d• zero, dar cu transferarea unui '1' la pozi•ia de mai sus care este adunat cifrelor din acea pozi•ie.

Exemplu:

```
    1010   Primul număr
-   1001   Al doilea număr
-------
   10011   Rezultat
```

Putem verifica dac• rezultatul este corect transferând aceste numere în sistemul numeric zecimal •i f•când adunarea în el. La transfer ob•inem valoarea 10 ca primul num•r, valoarea 9 ca al doilea, •i valoarea 19 ca sum•. Astfel am dovedit c• opera•ia s-a f•cut corect. Probleme apar când suma este mai mare decât ceea ce poate fi reprezentat de un num•r binar cu un num•r dat de cifre binare. Se pot aplica diferite solu•ii atunci, una este s• cre•tem num•rul de cifre binare în sum• ca în exemplul anterior.

Sc•derea, ca •i adunarea se face pe acela•i principiu. Rezultatul sc•derii dintre doi de zero, sau doi de unu r•mâne zero. Când sc•dem unu din zero, trebuie s• împrumut•m unu de la cifra binar• care este pe o pozi•ie mai sus în num•rul binar.

Exemplu:

```
    1010   Primul număr
-   1001   Al doilea număr
-------
    0001   Rezultat
```

Verificând rezulatul cum am f•cut •i cu adunarea, când transl•m aceste numere binare ob•inem numerele zecimale 10 •i 9. Diferen•a lor corespunde num•rului 1 care este ceea ce ob•inem din sc•dere.

B.3 Sistem numeric hexazecimal

Sistemul numeric hexazecimal are num•rul 16 ca baz• a sa. Pentu c• baza unui sistem numeric este 16, sunt 16 cifre care se pot g•si într-un num•r hexazecimal. Aceste cifre sunt "0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F". Literele A, B, C, D, E •i F nu sunt altceva decât valorile 10, 11, 12, 13, 14 •i 15. Ele s-au introdus ca un înlocuitor pentru a face scrierea mai u•oar•. Ca •i la sistemul binar, aici de asemenea, putem determina cu aceea•i formul• care este cel mai mare num•r zecimal pe care îl putem reprezenta cu un num•r specific de cifre hexazecimale.

Exempul:

$$16^2 - 1 = 256 - 1 = 255$$

Uzual, num•rul hexazecimal este scris cu un semn "$" sau "0x" înanitea lui, pentru a eviden•ia sistemul numeric. Astfel, num•rul A37E ar fi scris mult mai corect ca $A37E sau 0xA37E. Pentru a transla un num•r hexazecimal într-un sistem numeric binar nu este necesar de a face nici un calcul ci simpla schimbare de cifre hexazecimale cu cifre binare. Pentru c• valoarea maxim• a num•rului hexazecimal este 15, aceasta înseamn• c• este suficient s• se foloseasc• 4 cifre binare pentru o cifr• hexazecimal•.

Exempul:

$$\$E4 = \underbrace{1110}_{E}\ \underbrace{0100}_{4}$$

Veifică nd, sau transferând ambele numere într-un sistem numeric zecimal, ob•inem num•rul 228 ceea ce dovede•te acurate•ea ac•iunii noastre.

Pentru a ob•ine un zecimal echivalent a unui num•r hexazecimal, trebuie s• înmul•im fiecare cifr• a unui num•r cu num•rul 16 care este gradat prin pozi•ia acelei cifre în num•rul hexazecimal.

Exempul:

```
A37E
|||
||+-- 14*16^0 =     14
|+--- 7*16^1  =    112
+---- 3*16^2  =    768
      10*16^3 = 40960
-----------------------
Rezultat = 41854
```

Adunarea, ca •i în precedetele dou• exemple, se face într-o manier• similar•.

Exempul:

	$3A2B	Primul număr
+	$A9C1	Al doilea număr
	$E3EC	Rezultat

Trebuie s• ad•ug•m cifrele corespunz•toare ale num•rului; •i, dac• suma lor este mai mare ca 16, trebie s• scriem num•rul '0' acolo. Valoarea peste 16 trebuie adunat• urm•toarelor dou• cifre mai mari în valoare. Verificând, ob•inem 14891 ca prim num•r, •i al doilea este 43457. Suma lor este 58348, care este num•rul $E3EC când este transferat în sistemul numeric zecimal. Sc•derea este un proces identic celor dou• sisteme numerice anterioare. Dac• num•rul pe care îl sc•dem este mai mic, împrumut•m din urm•torul loc al valorii mai mari.

Exemplu:

	$2D46	Primul număr
+	$1752	Al doilea număr
	$15F4	Rezultat

Verificând rezultatul, ob•inem valorile 11590 pentru prmul num•r •i 5970 pentru al doilea, când diferen•a lor este 5620, ceea ce coresponde num•rului $15F4 dup• transferul într-un sistem numeric zecimal.

Concluzie

Sistemul numeric binar este înc• cel mai folosit, cel zecimal cel mai u•or de în•eles, iar cel hexazecimal este între cele dou• sisteme. Conversia lui u•oar• într-un sistem numeric binar •i memorarea lui u•oar• îl fac, împreun• cu sistemele binar •i zecimal, unul din cele mai importante sisteme numerice.

Pagina anterioar•	Con•inut	Pagina urm•toare

Anexa C

Glosar

Introducere

Introducere

Pentru c• toate domeniile de activitate ale omului sunt în mod obi•nuit bazate pe termeni adecva•i •i deja adopta•i (prin care au ap•rut alte no•iuni •i defini•ii), tot a•a în domeniul microcontrolerelor putem selecta câ•iva termeni frecvent folosi•i. Ideile sunt adesea conectate a• c• în•elegerea corect• a unei no•iuni este necesar• pentru a deveni familiariza•i cu una sau mai alte idei.

Microcontroler

Microprocesor cu periferice într-o singur• component• electronic•.

Pin I/O

Pin de conecor extern al microcontrolerului care poate fi configurat ca intrare sau ca ie•ire. În cele mai multe cazuri pinul I/O activeaz• microcontrolerul pentru a comunica, controla sau citi informa•ia.

Software

Informa•ia de care mocrocontrolerul are nevoie pentru a func•iona. Sotware-ul nu poate avea erori dac• vrem ca programul •i dispozitivul s• func•ioneze corect. Software-ul poate fi scris în diferite limbaje ca: Basic, C, Pascal sau assembler. Fizic, el este un fi•ier pe un disc de calculator.

Hardware

Mirocontrolerul, memoria, sursa, circuitele de semnal •i toate componentele conectate cu microcontrolerul.

Cel•lalt mod de a-l vedea (în special dac• nu func•ioneaz•) este, c•, hardware-ul este ceva ce pute•i atinge.

Simulator

Pachet software pentru PC care simuleaz• func•ionarea intern• a microcontrolerului. Este ideal pentru verificarea rutinelor software •i a tuturor p•r•ilor codului care nu au conexiuni de supra cerere cu exteriorul. Op•iunile sunt instalate pentru a supraveghea codul, mi•carea în program înapoi •i înainte •i pas cu pas, •i debugging-ul.

ICE

ICE (In Circuit Emulator), emulator intern, parte foarte folositoare a echipamentului care conecteaz• un PC în locul unui microcontroler la un dispozitiv care este în dezvoltare. Permite software-ului de a func•iona la un calculator PC, dar s• apar• ca •i cum un microcontroler real exist• în dispozitiv. ICE v• permite s• v• mi•ca•i în program în timp real, pentru a vedea ce se întâmpl• în microcontroler •i cum comunic• cu exteriorul.

Emulator EPROM

Emulatorul EPROM este un dispozitiv care nu emuleaz• întregul microcontroler ca emulatorul ICE, ci emuleaz• doar memoria lui. Este cel mai mult folosit la microcontrolerele ce au memorie extern•. Prin folosirea lui evit•m •tergerea •i scrierea constant• a memoriei EPROM.

Assembler

Pachet software care transleaz• codul surs• într-un cod pe care microcontrolerul îl poate în•elege. Con•ine o sec•iune pentru descoperirea erorilor. Aceast• parte este folosit• când depan•m un program de erorile f•cute când programul a fost scris.

Fi•ier HEX

Acesta este un fi•ier f•cut de translatorul assembler când se transleaz• un fi•ier surs•, •i are o form• "în•eleas•" de microcontroler. O continuare a fi•ierului este uzual File_name.HEX de unde vine numele fi•ierului HEX.

Fi•ier List

Acesta este un fi•ier f•cut de translatorul assembler •i con•ine toate instruc•iunile din fi•ierul surs• cu adresele •i comnetariile pe care le-a scris programatorul. Este un fi•ier foarte util pentru a urm•ri erorile în program. Extensia fi•ierului este LIST de unde vine •i numele lui.

Fi•ier Source

Fi•ier scris în limbjul în•eles de om •i de translatorul assembler. Prin translarea fi•ierului surs•, ob•inem fi•ierele HEX •i LIST.

Debugging

Eroare f•cut• în scrierea programului, eroare de care nu suntem în cuno•tiin••. Erorile pot fi chiar simple ca erori de tastare, •i chiar complexe ca folosirea incorect• a limbajului programului. Assembler-ul va g•si majoritatea acestor erori •i le va raporta fi•ierului '.LST'. Alte erori se vor c•uta prin încercarea •i urm•rirea func•ion•rii dispozitivului.

ROM, EPROM, EEPROM, FLASH, RAM

Tipuri de memorie pe care le întâlnim la folosirea microcontrolerului. Prima nu poate fi •tears•, ceea ce a•i scris în ea r•mâne pentru totdeauna, •i nu poate fi •ters. A doua este posibil de •ters electric cu sursa adus• separat, •i tensiunea peste aceea la care func•ioneaz• microcontrolerul. A treia poate de asemenea fi •tears• electric, dar folose•te tensiunea la care func•ioneaz• microcontrolerul. A patra este electric posibil de •ters, dar spre deosebire de memoria EEPROM, dar nu are un num•r a•a de mare de cicluri de scriere •i •tergere în loca•iile de memorie. A cincea este rapid•, dar nu re•ine con•inutul ca •i cea anterioar• când se întrerupe alimentarea. Astfel, programul nu este memorat în ea, dar serve•te pentru diferite variabile •i inter-rezultate.

Adresarea

Determin• •i asigneaz• unele loca•ii de memorie.

ASCII

Prescurtare pentru "American Standard Code for Information Interchange-Codul Standard American pentru Interschimb Informa•ii". Este un tip de cod larg acceptat unde fiecare num•r •i liter• au codul lor de opt bi•i.

Carry

Bit de transfer conectat cu opera•ii aritmetice.

Code

Fi•ier sau sec•iune a unui fi•ier ce con•ine instruc•iuni de program.

Byte, Kilobyte, Megabyte

Termen desemnând cantitatea de informa•ie. Unitatea de baz• este un byte, •i are 8 bi•i. Kilobyte-ul are 1024 bytes, •i un megabyte are 1024 kilobytes.

Flag

Bi•i din registrul de stare. Prin activarea lor, programatorul este informat de unele ac•iuni. Programul activeaz• r•spunsul lui dac• este necesar.

Vector întrerupere sau întreruperi

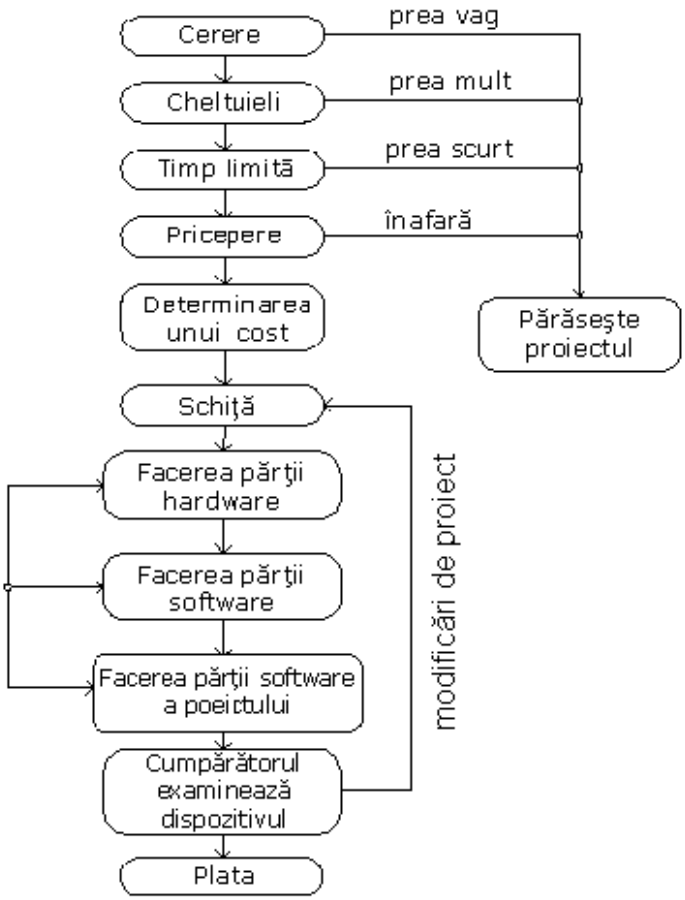
Loca•ie în memoria microcontrolerului. Microcontrolerul ia din aceast• loca•ie informa•ia despre o sec•iune a programului ce trebuie executat• ca un r•spuns la unele evenimente de interes ale programatorului •i dispozitivului.

Programator

Dispozitiv ce face posibil• scrierea software-ului în memoria microcontrolerului, permi•ând astfel microcontrolerului s• lucreze independent. Const• din sec•iunea hardware uzual conectat• cu unul din porturi •i sec•iune software folosit• de calculator ca un program.

Produs

Dezvoltarea produsului este o combina•ie de succes •i experien••. Termenele scurte sau limitele de timp trebuie evitate pentru c• chiar •i cu cele mai multe simple asign•ri, este nevoie de mult timp pentru a dezvolta •i îmbun•t••i. Când se creaz• un proiect, avem nevoie de timp, lini•te , minte logic• •i cel mai important, o în•elegere complet• a nevoilor consumatorului. Cursul tipic în crearea unui produs va avea urm•torul algoritm:



Pagina anterioar• | Con•inut | Pagina urm•toare

2.1 Generator de ceas – oscilator

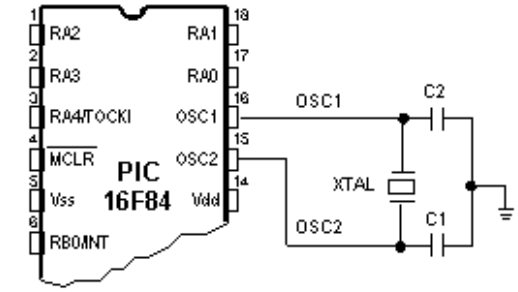
Circuitul oscilator este folosit pentru a da microcontrolerului un ceas-clock. Ceasul este necesar pentru ca microcontrolerul s• execute programul sau instruc•iunile din program.

Tipuri de oscilatoare

PIC16F84 poate lucra cu patru configura•ii diferite de oscilator. Pentru c• configura•iile cu oscilator cu cristal •i rezistor-condensator (RC) sunt cele utilizate cel mai frecvent, doar pe ele le vom men•iona aici. Tipul de microcontroler cu oscilator cu cristal este desemnat ca XT, iar microcontrolerul cu perechea rezistor-condensator are desemnarea RC. Aceasta este important pentru c• trebuie s• numi•i tipul de oscilator când cump•ra•i un microcontroler.

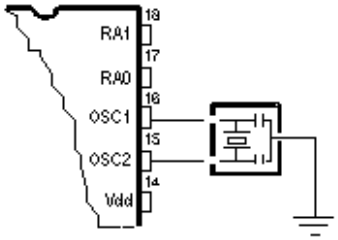
Oscilatorul XT

Oscilatorul cu cristal se afl• intr-o carcas• metalic• cu doi pini pe care este înscris• frecven•a la care cristalul oscileaz•. Mai este necesar câte un condensator ceramic de 30pF cu cel•lalt cap•t la mas• de a fi conecta•i la fiecare pin.



Conectarea oscilatorului cu cuarț pentru a da ceasul microcontrolerului

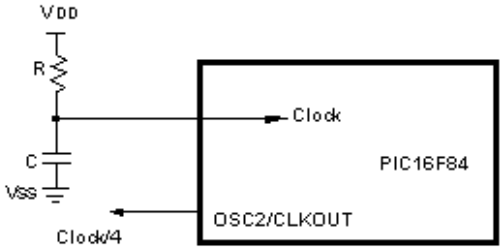
Oscilatorul •i condensatorii pot fi încapsula•i împreun• într-o carcas• cu trei pini. Un asemenea element se nume•te rezonator ceramic •i este reprezentat în scheme ca cel de mai jos. Pinii centrali ai elementului sunt masa, iar pinii terminali sunt conecta•i la pinii OSC1 •i OSC2 ai microcontrolerului. Când se proiecteaz• un aparat, regula este s• plasa•i oscilatorul cât mai aproape de microcontroler, pentru a elimina orice interferen•• de pe liniile pe care microcontrolerul prime•te tactul de ceas.



Conectarea unui rezonator la un microcontroler

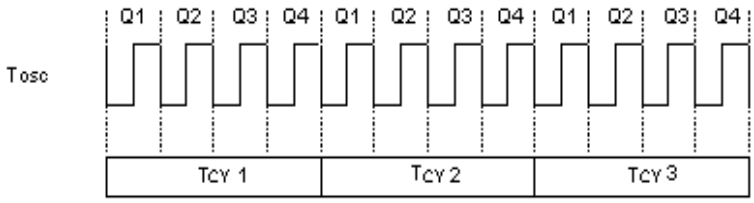
Oscilatorul RC

În aplica•iile unde nu este nevoie de o mare precizie de timp, oscilatorul RC permite economii adi•ionale la cump•rare. Fecven•a de rezonan•• a oscilatorului RC depinde de valoarea tensiunii de alimentare, rezistorul R, condensatorul C •i temperatura de lucru. Trebuie de men•ionat c• frecven•a de rezonan•• este de asemenea influen•at• de varia•iile normale ale parametrilor de proces, de toleran•a extern• a componentelor R •i C, etc.



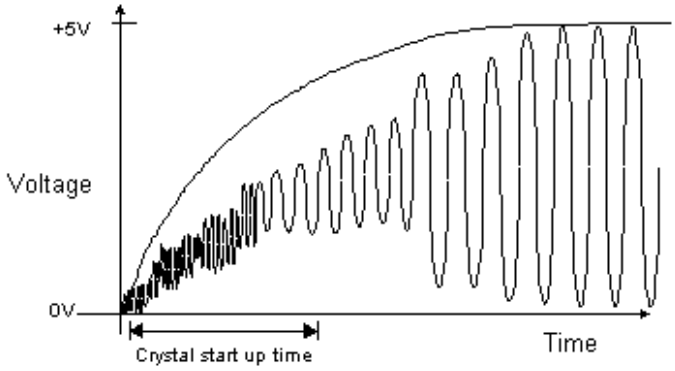
Notă: Acest pin poate fi configurat ca pin input/output

Diagrama de mai sus arat• cum este conectat oscilatorul RC la PIC16F84. La valoarea rezistorului mai mic• 2.2k, oscilatorul poate deveni instabil, sau oscila•ia se poate chiar opri. La valori mari a lui R (ex.1M) oscilatorul devine foarte sensibil la zgomot •i umezeal•. Se recomand• ca valoarea rezistorului R s• fie între 3 •i 100k. Chiar dac• oscilatorul va lucra f•r• un condensator extern (C=0pF), trebuie totu•i folosit un condensator de peste 20pF pentru zgomot •i stabilitate. Indiferent de ce oscilator este folosit, pentru a ob•ine un ceas la care s• func•ioneze microcontrolerul, ceasul trebuie divizat la 4. Un ceas al oscilatorului divizat cu 4 se poate ob•ine la pinul OSC2/ CLKOUT, •i poate fi folosit pentru testarea sau sincronizarea altor circuite logice.



Relatia dintre ceas şi numărul de cicluri instructiune

Dup• alimentare, oscilatorul începe s• oscileze. Oscila•ia la început are o perioad• •i o amplitudine instabile, dar dup• un timp devin stabilizate.



Semnalul unui oscilator de ceas după alimentarea microcontrolerului

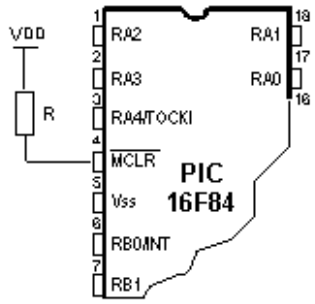
Pentru a preveni ca un asemenea ceas inexact s• influen•eze performan•ele microcontrolerului, trebuie s• •inem microcontrolerul în starea reset pe durata stabiliz•rii ceasului oscilatorului. Diagrama de mai sus arat• o form• tipic• de semnal pe care microcontrolerul o prime•te de la oscilatorul cu cuar• dup• alimentare.

Pagina anterioar•	Con•inut	Pagina urm•toare

2.2 Reset-ul

Resetul este folosit pentru a pune microcontrolerul într-o condi•ie 'cunoscut•'. Aceasta înseamn• practic c• microcontrolerul poate s• se comporte incorect în unele condi•ii nedorite. Pentru a continua s• func•ioneze corect trebuie resetat, însemnând c• to•i registrii vor fi pu•i într-o stare de start. Resetul nu este folosit numai când microcontrolerul nu se comport• cum vrem noi, dar poate de asemenea s• fie folosit când se încearc• un montaj ca o întrerupere într-un program de execu•ie sau când se preg•te•te un microcontroler de a citi un program.

Pentru a preveni ajungerea unui zero logic la pinul MCLR accidental (linia de deasupra înseamn• c• resetul este activat de un zero logic), MCLR trebuie s• fie conectat printr-un rezistor la polul pozitiv al sursei de alimentare. Rezistorul trebuie s• fie între 5 •i 10k. Acest rezistor a c•rui func•ie este de a men•ine o anumit• linie la starea logic• unu ca o prevenire, se nume•te o scoatere-pull up.



Folosirea circuitului intern de reset

Microcontrolerul PIC16F84 are câteva surse de reset:

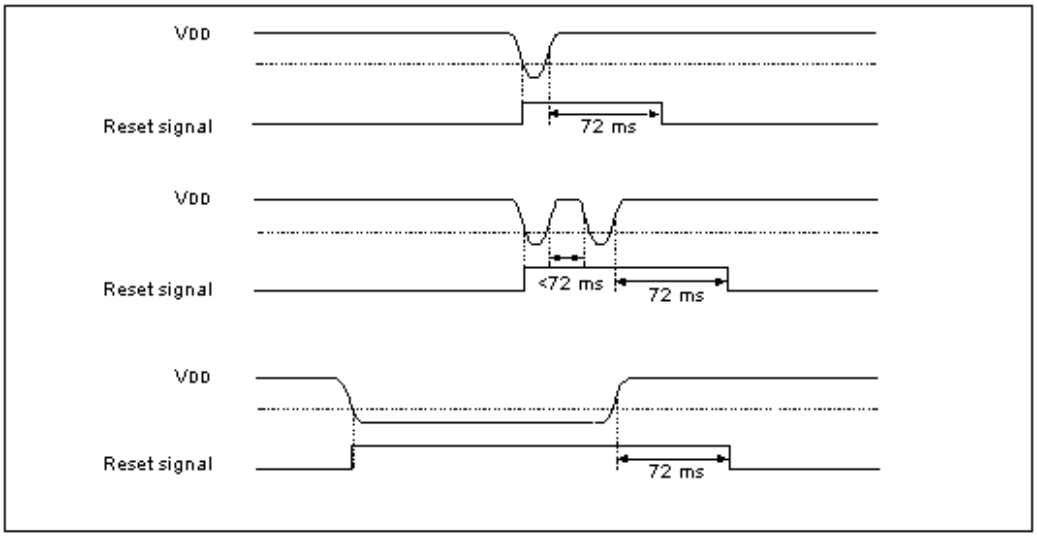
a) Reset la alimentare, POR (Power-On Reset)
b) Reset în timpul lucrului obi•nuit prin aducerea unui zero logic la pinul MCLR al microcontrolerului.
c) Reset în timpul regimului SLEEP
d) Reset la dep••irea timer-ului watchdog (WDT)
e) Reset în timpul dep••irii WDT în timpul regimului SLEEP.

Cele mai importante resurse de reset sunt a) •i b). Prima are loc de fiecare dat• când este alimentat microcontrolerul •i serve•te la aducerea toturor regi•trilor la starea ini•ial• a pozi•iei de start. A doua este pentru a aduce un zero logic la pinul MCLR în timpul opera•iei normale a microcontrolerului. Este des folosit• în dezvoltarea de programe.

În timpul unui reset, loca•iile de memorie RAM nu sunt resetate. Ele sunt necunoscute la alimentare •i nu sunt schimbate la nici un reset. Spre deosebire de acestea, regi•trii SFR sunt reseta•i la o stare ini•ial• a pozi•iei de start. Unul din cele mai importante efecte ale resetului este setarea contorului de program (PC) la zero (0000h), ceea ce permite programului s• înceap• executarea de la prima instruc•iune scris•.

Resetul la sc•derea tensiunii de alimentare dincolo de limita permisibil• (Brown-out Reset)

Impulsul pentru resetare în timpul cre•terii tensiunii este generat de microcontrolerul însu•i când detecteaz• o cre•tere în tensiunea Vdd (în domeniul de la 1.2V la 1.8V). Acest impuls dureaz• 72 ms ceea ce este un timp suficient pentru oscilator ca s• se stabilizeze. Aceste 72 ms sunt asigurate de un timer intern PWRT care are oscilatorul lui RC. Microcontrolerul este în modul reset cât timp PWRT este activ. Totu•i, când montajul func•ioneaz•, probleme apar când sursa nu scade la zero ci când scade mai jos de limita ce garanteaz• func•ionarea corect• a microcontrolerului. Acesta este un caz real din practic•, în special în mediile industriale unde perturba•iile •i instabilit••ile sursei de alimentare sunt ceva foarte curent. Pentru a rezolva aceast• problem• trebuie s• ne asigur•m c• microcontrolerul este într-o stare de reset de fiecare dat• când tensiunea sursei scade sub limita admis•.



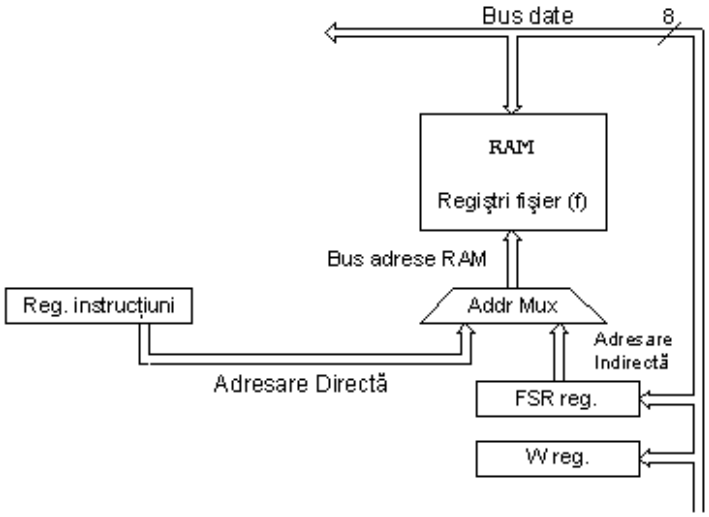
Exemple de cădere a sursei de tensiune mai jos de nivelul admis

Dac•, conform cu specifica•iile electrice, circuitul intern de resetare a microcontrolerului nu poate satisface aceste cerin•e, se pot folosi componente electronice speciale ce sunt capabile s• genereze semnalul de reset dorit. În afar• de aceast• func•ie, ele pot func•iona pentru supravegherea tensiunii de alimentare. Dac• tensiunea scade mai jos de nivelul specificat, un zero logic va apare la pinul MCLR ce •ine microcontrolerul în starea de reset pân• ce tensiunea nu este în limitele ce garanteaz• func•ionarea corect•.

Pagina anterioar•	Con•inut	Pagina urm•toare

2.3 Unitatea de Procesare Central•

Unitatea de procesare central• (CPU) este creierul microcontrolerului. Aceast• parte este responsabil• cu g•sirea •i aducerea (citirea din memorie)-fetching instruc•iunii corecte ce trebuie executat•, cu decodarea acelei instruc•iuni, •i în final cu executarea ei.

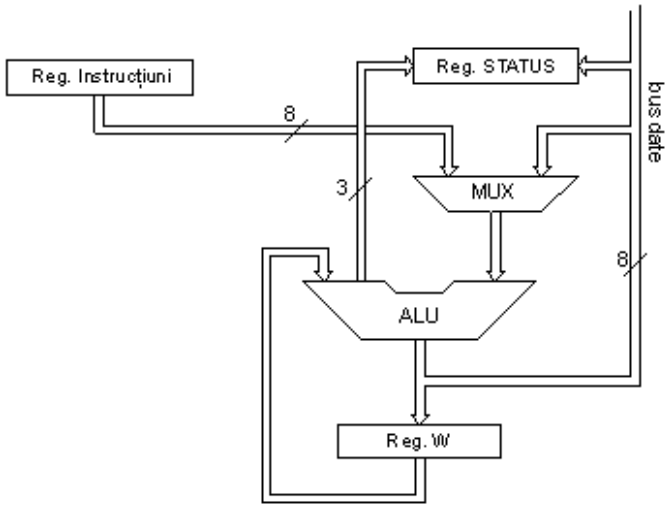


Schiță a unității de procesare centrale

Unitatea de procesare central• conecteaz• toate p•r•ile microcontrolerului într-un întreg. Desigur, func•ia sa cea mai important• este s• decodeze instruc•iunile de program. Când programatorul scrie un program, instruc•iunile au o form• clar• ca MOVLW 0x20. Totu•i, pentru ca microcontrolerul s• în•eleag• aceasta, aceast• form• de 'scrisoare' a unei instruc•iuni trebuie tradus• într-o serie de zero-uri •i unu-uri ce se nume•te 'opcode'. Aceast• tranzi•ie de la o scrisoare la o form• binar• este f•cut• de translatori ca translatorul assembler (cunoscut ca •i assembler sau asamblor). Instruc•iunea astfel adus•-fetched din memoria programului trebuie s• fie decodat• de unitatea de procesare central•. Putem apoi selecta din tabela tuturor instruc•iunilor un set de ac•iuni ce execut• o sarcin• desemnat• definit• de instruc•iune. Pentru c• instruc•iunile pot s• con•in• în ele asign•ri ce cer diferite transferuri de date dintr-o memorie în alta, din memorie la porturi, sau alte calcule, CPU trebuie s• fie conectat• cu toate p•r•ile microcontrolerului. Aceasta este posibil printr-un bus de date •i un bus de adrese.

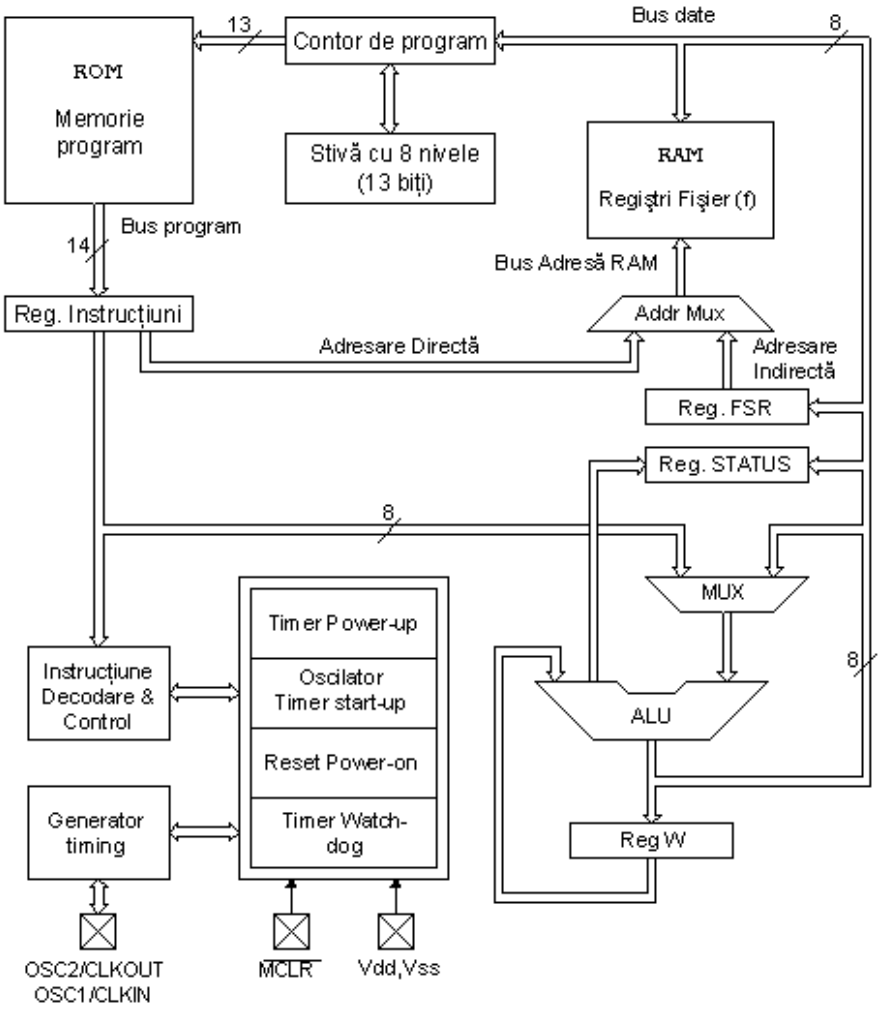
Unitatea de Logic• Aritmetic• (ALU)

Unitatea de logic• aritmetic• este responsabil• de executarea opera•iilor ca adunarea, sc•derea, mutarea (la stânga sau la dreapta într-un registru) •i de opera•iile logice. Mutarea datelor într-un registru se mai nume•te 'shifting'-transferare. PIC16F84 con•ine o unitate logic• aritmetic• de 8 bi•i •i regi•tri de lucru de 8 bi•i.



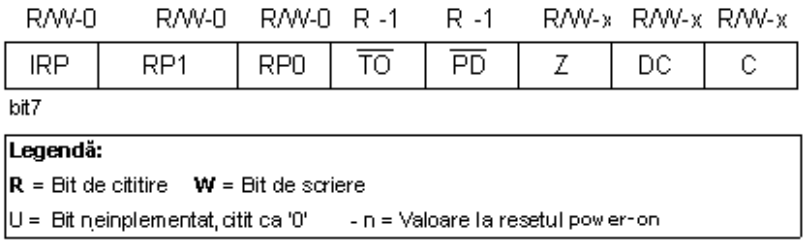
Unitatea de logică-aritmetică şi cum lucrează

În instruc•iunile cu doi operanzi, în mod obi•nuit un operand este în registrul de lucru (registrul W), iar cel•lalt este unul din regi•tri sau o constant•. Prin operand în•elegem con•inutul asupra c•ruia se fac unele opera•ii, iar un registru este oricare din regi•trii GPR sau SFR. GPR este o prescurtare de la 'General Purposes Registers'-Regi•tri cu Scopuri Generale, iar SFR de la 'Special Function Registers'-Regi•tri cu Func•ie Special•. În instruc•iunile cu un operand, un operand este fie registrul W fie unul din regi•tri. Pe lâng• opera•iile aritmetice •i logice, ALU controleaz• bi•ii de stare (bi•i g•si•i în registrul STATUS). Executarea unor instruc•iuni afecteaz• bi•ii de stare, de care depinde rezultatul însu•i. Depinzând de ce instruc•iune este executat•, ALU poate afecta valorile bi•ilor Carry (C), Digit Carry (DC), •i Zero (Z) în registrul STATUS.



Schiță bloc mai detaliată a microcontrolerului PIC16F84

Registru STATUS



bit 0 **C** (Carry) Transfer
Bit care este afectat de opera•iile de adunare, sc•dere •i transfer.
1= transferul produs din bitul cel mai înalt al rezultatului
0= transferul nu s-a produs
Bitul C este afectat de instruc•iunile ADDWF, ADDLW, SUBLW, SUBWF.

bit 1 **DC** (Digit Carry) DC Transfer
Bit afectat de opera•iile de adunare, sc•dere •i transfer. Spre deosebire de bitul C, acest bit reprezint• transferul din al patrulea loc rezultat. Este setat de adunare când se întâmpl• un transport de la bitul 3 la bitul 4, sau de sc•dere când se întâmpl• împrumut de la bitul 4 la bitul 3, sau de transfer în ambele direc•ii.
1= transfer produs la al patrulea bit conform cu ordinea, al rezultatului.
0= transferul nu s-a produs
Bitul DC este afectat de instruc•iunile ADDWF, ADDLW, SUBLW, SUBWF.

bit 2 **Z** (Zero bit) Indicarea unui rezultat zero.
Acest bit este setat când rezultatul unei opera•ii 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 s• func•ioneze, dup• fiecare reset obi•nuit •i dup• executarea instruc•iunii CLRWDT. Instruc•iunea SLEEP îl reseteaz• când microcontrolerul intr• în regimul consum/ uzaj redus. Setarea lui repetat• este posibil• prin reset sau prin pornirea sau oprirea sursei. Starea poate fi triggerat• de asemenea de un semnal la pinul RB0/INT, de o schimbare la portul RB, de terminarea scrierii în EEPROM-ul de date intern, •i de watchdog de asemenea.
1= dup• ce sursa a fost pornit•
0= executarea instruc•iunii SLEEP

bit 4 **TO** Time-out ; dep••irea-overflow watchdog-ului.
Bitul este setat dup• pornirea sursei •i executarea instruc•iunilor CLRWDT •i SLEEP. Bitul este resetat când watchdog-ul ajunge la sfâr•it semnalând c• ceva nu este în ordine.
1= dep••irea-oveflow nu s-a produs
0= dep••irea-overflow s-a produs

bit6:5 **RP1:RP0** (Register Bank Select bits-Bi•i de Selectare a Bancului de Regi•tri)
Ace•ti doi bi•i sunt partea superioar• a adresei la adresarea direct•. Pentru c• instruc•iunile ce adreseaz• memoria direct au doar •apte bi•i, ei au nevoie doar de înc• un bit pentru a adresa cei 256 bytes adic• câ•i are PIC16F84. Bitul RP1 nu este folosit, dar este l•sat 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 Regi•tri)
Bit al c•rui rol este de a fi al optulea bit la adresarea indirect• a RAM-ului intern.
1= bancul 2 •i 3
0= bancul 0 •i 1 (de la 00h la FFh)

Registrul STATUS con•ine starea aritmetic• ALU (C, DC, Z), starea RESET (TO, PD) •i bi•ii pentru selectarea bancului de memorie (IRP, RP1, RP0). Considerând c• selec•ia bancului de memorie este controlat• prin acest registru, el trebuie s• fie prezent în fiecare banc. Bancul de memorie se va discuta mai în detaliu în capitolul Organizarea memoriei. Registrul STATUS poate fi o destina•ie pentru orice instruc•iune, cu oricare alt registru. Dac• registrul STATUS este o destina•ie pentru instruc•iunile ce afecteaz• bi•ii Z, DC or C, atunci scrierea în ace•ti trei bi•i nu este posibil•.

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 citire **W** = Bit de scriere
U = Bit neimplementat, citit ca '0' - n = Valoarea la resetul power-on

bit 0:2 **PS0, PS1, PS2** (Prescaler Rate Select bit-Bit Selec•ie Rat• Prescaler)
Ace•ti trei bi•i definesc bitul de selec•ie a ratei prescalerului. Ce este un prescaler •i cum pot afecta ace•ti bi•i func•ionarea unui microcontroler va fi explicat în sec•iunea depre 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 3 **PSA** (Prescaler Assignment bit-Bit de Asignare Prescaler)
Bit ce asigneaz• prescalerul între TMRO •i watchdog.
1= prescalerul este asignat watchdogului
0= prescalerul este asignat timer-ului liber (ree-run) TMRO

bit 4 **T0SE** (TMR0 Source Edge Select bit-Bit Selec•ie a Frontului Sursei TMR0)
Dac• este permis de a se triggera TMRO prin impulsurile de la pinul RA4/T0CKI, acest bit determin• dac• aceasta va fi la frontul descresc•tor sau cresc•tor al unui semnal.
1= front cresc•tor
0= front descresc•tor

bit 5 **TOCS** (TMR0 Clock Source Select bit-Bit Selec•ie Surs• Ceas TMR0)
Acest pin permite timerului liber (free-run) s• 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

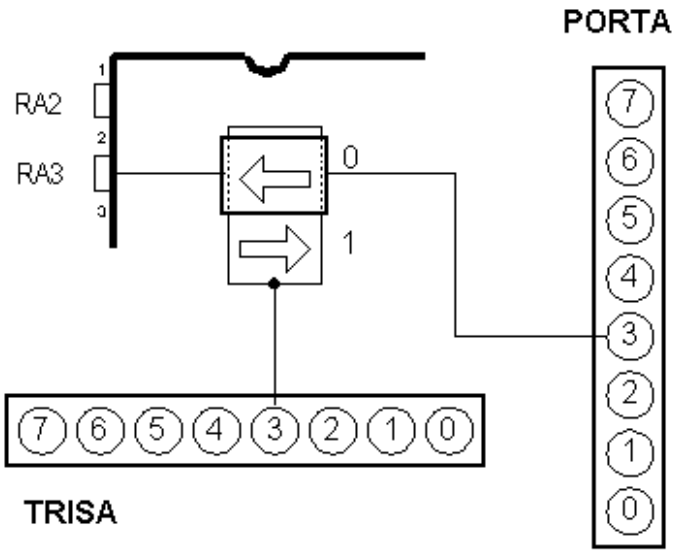
bit 6 I**NTEDG** (Interrupt Edge Select bit-Bit de Selec•ie a Frontului Întrerupere)
Dac• întreruperea este activat• este posibil ca acest bit s• determine frontul la care o întrerupere va fi activat• la pinul RB0/INT.
1= front cresc•tor
0= front descresc•tor

bit 7 **RBPU** (PORTB Pull-up Enable bit-Bit Enable-Activare Pull-up PORTB)
Acest bit porne•te •i opre•te rezistorii interni 'pull-up'-scoatere la portul B.
1= Rezistori oprire "pull-up"
0= Rezistori pornire "pull-up"

Pagina anterioar• **Con•inut** **Pagina urm•toare**

2.4 Porturi

Portul se refer• la un grup de pini ai unui microcontroler ce pot fi accesa•i simultan, sau la care putem seta combina•ia dorit• de zero-uri •i unu-uri, sau de la care putem citi o stare existent•. Fizic, portul este un registru în interiorul unui microcontroler ce este conectat cu fire la pinii microcontrolerului. Porturile reprezint• conexiunea fizic• a Unit••ii de Procesare Central• cu lumea exterioar•. Microcontrolerul le folose•te pentru a monitoriza sau controla alte componente sau aparate. Datorit• func•ionalit••ii, unii pini au rol dublu ca RA4/TOCKI de exemplu, care este simultan al patrulea bit la portul A •i o intrare extern• pentru contorul liber (free-run). Selec•ia uneia din aceste dou• func•ii ale pinului se face în unul din regi•trii configura•ionali. O ilustra•ie a acesteia este al cincilea bit T0CS în registrul OPTION. Selectând una din func•ii cealalt• este dezactivat•.



Relația dintre TRISA şi registrul PORTA

To•i pinii portului pot fi defini•i ca intrare sau ie•ire, conform cu nevoile unui montaj ce este în dezvoltare. Pentru a defini un pin ca pin de intrare sau ca pin de ie•ire, trebuie scris• combina•ia corect• de zero-uri •i unu-uri în registrul TRIS. Dac• în locul potrivit este scris "1" logic în registrul TRIS, acel pin este pin de intrare, iar dac• este valabil contrariul, este un pin de ie•ire. Fiecare port are registrul lui TRIS. Astfel, portul A are TRISA la adresa 85h, iar portul B are TRISB la adresa 86h.

PORTB

PORTB are 8 pini lega•i la el. Registrul adecvat pentru direc•ia datelor este TRISB la adresa 86h. Setarea unui bit în registrul TRISB define•te pinul portului corespunz•tor ca pin de intrare, •i resetarea unui bit în registrul TRISB, define•te pinul portului corespunz•tor ca pin de ie•ire. Fiecare pin la PORTB are un rezistor slab intern pull-up (scoatere) (rezistor care define•te o linie la unu logic) care poate fi activat prin resetarea celui de-al •aptelea bit RBPU în registrul OPTION. Ace•ti rezistori 'pull-up' se închid automat când pinul portului este configurat ca o ie•ire. Când porne•te microcontrolerul, 'pull-up'-ii sunt dezactiva•i.

Patru pini ai portului PORTB, RB7:RB4 pot cauza o întrerupere, care se întâmpl• când starea lor se schimb• de la unu logic la zero logic •i invers. Numai pinii configura•i ca intrare pot cauza aceast• întrerupere s• se întâmple (dac• fiecare pin RB7:RB4 este configurat ca o ie•ire, nu va fi generat• o întrerupere la schimbarea st•rii). Aceast• op•iune de întrerupere cu rezistorii 'pull-up' fac mai u•oar• rezolvarea problemelor din practic•, ca de exemplu o tastatur• matriceal•. Dac• rândurile tastaturii sunt conectate la ace•ti pini, fiecare ap•sare a unei clape va cauza o întrerupere. Microcontrolerul va determina care clap• este ap•sat• în timp ce se proceseaz• o întrerupere. Nu se recomand• s• apela•i la portul B în timp ce se proceseaz• întreruperea.

```
clrf   STATUS       ;Bank0
clrf   PORTB        ;PORTB=0
bsf    STATUS,RP0   ;Bank1
movlw  0x0F         ;Defining input and output  pins
movwf  TRISB        ;Writing to TRISB register
```

Exemplul de mai sus arat• cum pinii 0, 1, 2, •i 3 sunt declara•i ca intrare, •i pinii 4, 5, 6 •i 7 ca ie•ire.

PORTA

PORTA are 5 pini lega•i la el. Registrul corespunz•tor pentru direc•ia datelor este TRISA la adresa 85h. Ca •i la portul B, setarea unui bit în registrul TRISA define•te de asemenea pinul portului corespunz•tor ca un pin de intrare, •i resetarea unui bit în registrul TRISA define•te pinul portului corespunz•tor ca pin de ie•ire.
Al cincilea pin al portului A are func•ie dual•. La acel pin se afl• de asemenea o intrare extern• pentru timer-ul TMRO. Una din aceste dou• op•iuni este aleas• prin setarea sau resetarea bitului TOCS (TMR0 Clock Source Select bit-bit de Selec•ie a Sursei Ceasului TMRO). Acest pin permite timer-ului TMRO sa-•i creasc• starea fie de la oscilatorul intern fie prin impulsuri externe la pinul RA4/T0CKI.

```
bcf    STATUS,RP0   ;Bank0
clrf   PORTA        ;PORTA=0
bsf    STATUS,RP0   ;Bank1
movlw  0x1F         ;Defining input and output pins pinova
movwf  TRISA        ;Writing to TRISA register
```

Exemplul arat• cum pinii 0, 1, 2, 3, •i 4 sunt declara•i ca intrare iar 5, 6 •i 7 ca pini de ie•ire.

Pagina anterioar•	Con•inut	Pagina urm•toare

2.5 Organizarea memoriei

PIC16F84 are dou• blocuri separate de memorie, unul pentru date •i cel•lalt pentru programe. Memoria EEPROM •i regi•trii GPR în memoria RAM constituie un bloc, •i memoria FLASH constituie un bloc de programe.

Memoria program

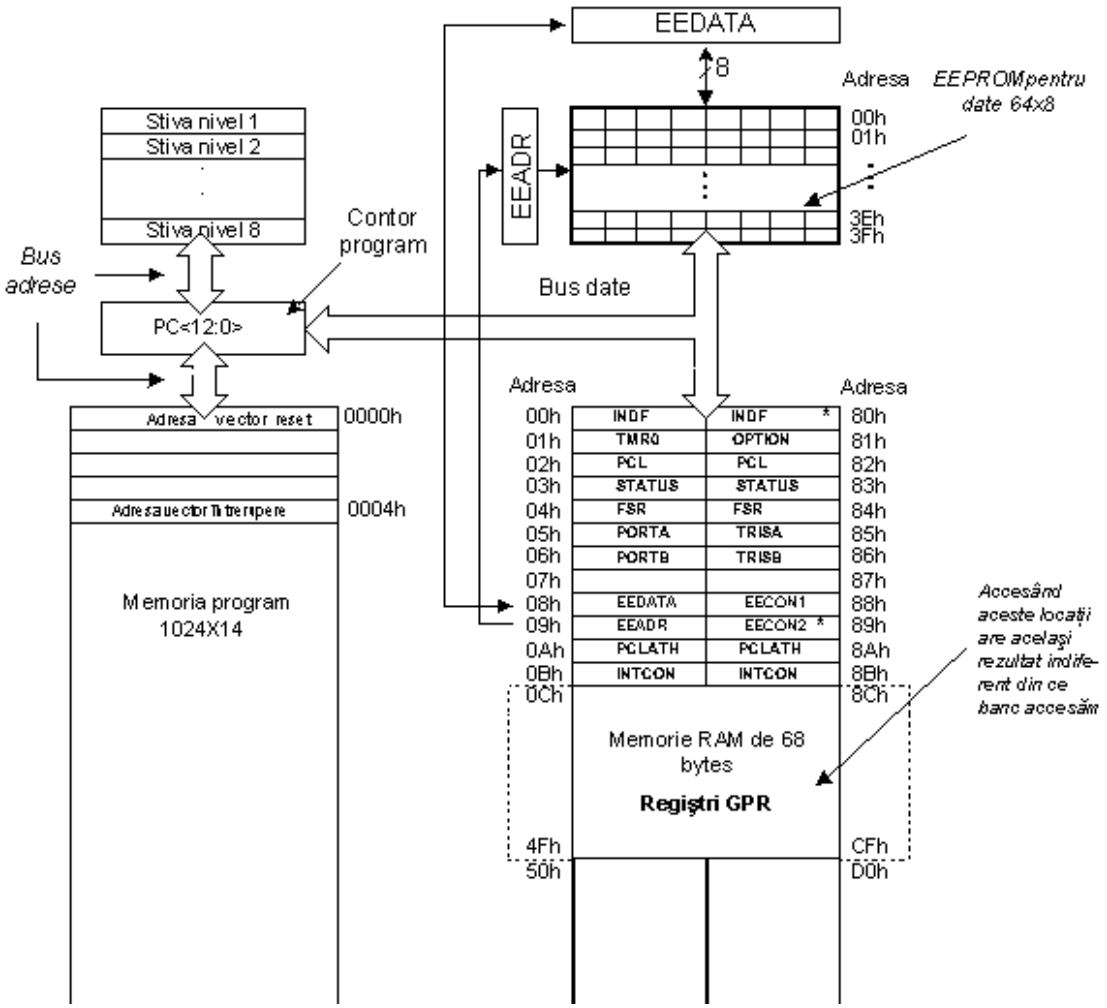
Memoria program a fost realizat• în tehnologia FLASH ceea ce face posibil de a programa un microcontroler de mai multe ori înainte de a fi instalat într-un montaj, •i chiar dup• instalarea sa dac• se întâmpl• unele schimb•ri în program sau parametri de proces. M•rimea memoriei program este de 1024 loca•ii cu l••ime de 14 bi•i unde loca•iile zero •i patru sunt rezervate pentru reset •i pentru vectorul întrerupere.

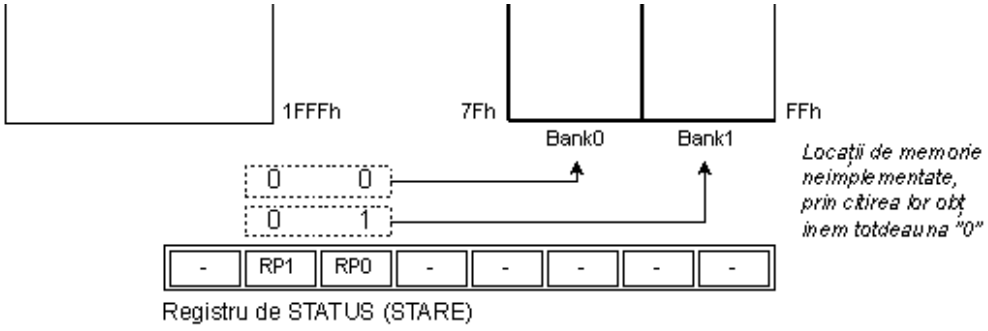
Memoria de date

Memoria de date const• din memoriile EEPROM •i RAM. Memoria EEPROM const• din 64 de loca•ii de opt bi•i a c•ror con•inut nu este pierdut în timpul opririi sursei de alimentare. EEPROM-ul nu este direct adresabil, dar este accesat indirect prin regi•trii EEADR •i EEDATA. Pentru c• memoria EEPROM este folosit• curent la memorarea unor parametri importan•i (de exemplu, o temperatur• dat• în regulatoarele de temperatur•), exist• o procedur• strict• de scriere în EEPROM ce trebuie urmat• pentru a preveni scrierea accidental•. Memoria RAM pentru date ocup• un spa•iu într-o hart• a memoriei de la loca•ia 0x0C la 0x4F ceea ce înseamn• 68 de loca•ii. Loca•iile memoriei RAM sunt de asemenea denumite regi•tri GPR care este o abreviere General Purpose Registers-Regi•tri cu Scop General. Regi•trii GPR pot fi accesa•i indiferent de ce banc este selectat la un moment.

Regi•tri SFR

Regi•tri ce ocup• primele 12 loca•ii în bancurile 0 •i 1 •i sunt regi•tri ai func•iei specializate asignat• cu unele blocuri ale microcontrolerului. Ace•tia sunt numi•i Special Function Registers-Regi•tri ai Func•iei Speciale.





Organizarea memoriei microcontrolerului PICF84

Bancuri de Memorie

În afar• de aceast• diviziune în 'lungime' a regi•trilor SFR •i GPR, harta memoriei este de asemenea împ•r•it• în 'l••ime' (vezi harta precedent•) în dou• zone numite 'bancuri'. Selectarea unuia din bancuri se face de bi•ii RPO •i RP1 în registrul STATUS-stare.

Exemplu:

```
bcf STATUS, RP0
```

Instruc•iunea BCF •terge bitul RPO (RP0=0) în registrul STATUS •i astfel seteaz• bancul 0.

```
bsf STATUS, RP0
```

Instruc•iunea BSF seteaz• bitul RPO (RP0=1) în registrul STATUS •i astfel seteaz• bancul 1.

Uzual, grupurile de instruc•iuni care sunt adesea în uz, sunt conectate într-o singur• unitate ce poate fi u•or apelat• într-un program, •i a c•rei nume are o semnifica•ie clar•, a•a-numitul Macros-macrocomand•. Cu ajutorul lor, selec•ia dintre dou• bancuri devine mai clar• •i programul mult mai elegibil.

```
BANK0 macro
      Bcf STATUS, RP0   ;Select memory bank 0
      Endm

BANK1 macro
      Bsf STATUS, RP0   ;Select memory bank 1
      Endm
```

Loca•iile 0Ch - 4Fh sunt regi•tri cu scop general (GPR) ce sunt folosi•i ca memorie RAM. Când sunt accesate loca•iile 8Ch - CFh în Bancul 1, acces•m de fapt exact acelea•i loca•ii în Bancul 0. Cu alte cuvinte, când dori•i s• accesa•i unul din regi•trii GPR, nu trebuie s• v• îngrijora•i c• nu •ti•i în ce banc sunte•i!

Contorul de Program

Contorul de program (PC) este un registru de 13 bi•i ce con•ine adresa instruc•iunii ce se execut•. Prin incrementarea sau schimbarea sa (ex. în caz de salturi) microcontrolerul execut• instruc•iunile de program pas-cu-pas.

Stiva

PIC16F84 are o stiv• de 13 bi•i cu 8 nivele, sau cu alte cuvinte, un grup de 8 loca•ii de memorie de 13 bi•i l••ime cu func•ii speciale. Rolul s•u de baz• este de a p•stra valoarea contorului de program dup• un salt din programul principal la o adres• a unui subprogram. Pentru ca un program s• •tie cum s• se întoarc• la punctul de unde a pornit, trebuie s• înapoieze valoarea contorului programului din stiv•. Când se mut• dintr-un program într-un subprogram, contorul programului este împins în stiv• (un exemplu de acesta este instruc•iunea CALL). Când se execut• instruc•iuni ca RETURN, RETLW sau RETFIE ce au fost executate la sfâr•itul unui subprogram, contorul programului a fost luat dintr-o stiv•, a•a ca programul s• poat• continua de unde a fost oprit înainte de a fi întrerupt. Aceste opera•ii de plasare într-o •i luare dintr-o stiv• de contor de program sunt numite PUSH •i POP, •i sunt numite conform cu instruc•iunile similare ale unor microcontrolere mai mari.

Programarea În Sistem

Pentru a programa o memorie de program, microcontrolerul trebuie s• fie setat pentru un mod de lucru special prin

aducerea pinului MCLR la 13.5V, iar sursa de tensiune Vdd trebuie s• fie stabilizat• între 4.5V •i 5.5V. Memoria program poate fi programat• serial folosind doi pini 'data/clock' ce trebuie s• fie mai întâi separa•i de liniile montajului, a•a ca s• nu apar• erori în timpul program•rii.

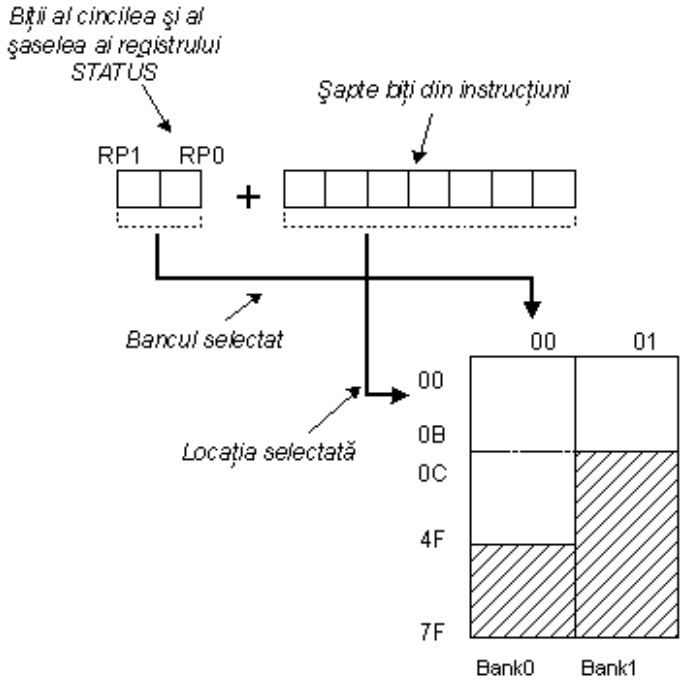
Moduri de adresare

Loca•iile de memorie RAM pot fi accesate direct sau indirect.

Adresarea Direct•

Adresarea Direct• se face printr-o adres• de 9 bi•i. Aceast• adres• este ob•inut• prin conectarea celui de-al •aptelea bit al adresei directe a unei instruc•iuni cu doi bi•i (RP1, RP0) din registrul STATUS dup• cum se arat• în figura urm•toarea. Orice acces la regi•trii SFR poate fi un exemplu de adresare direct•.

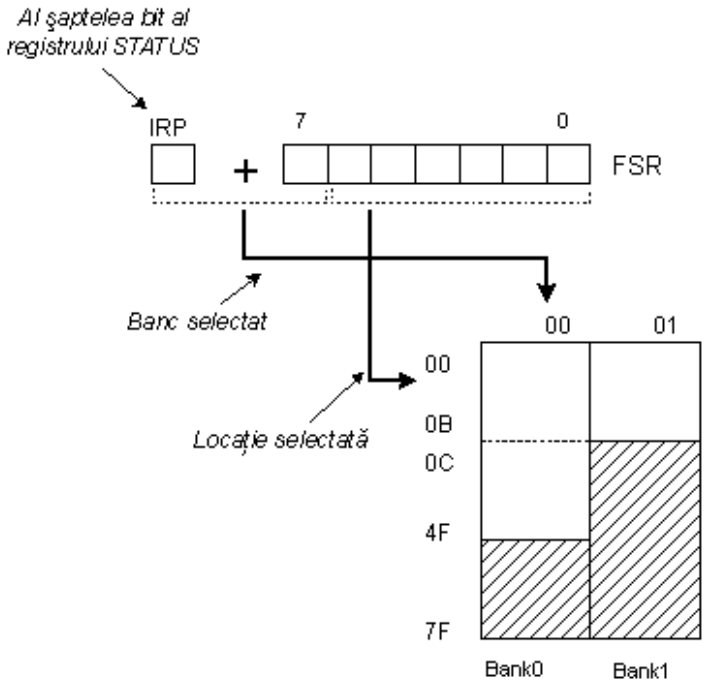
```
Bsf STATUS, RP0 ;Bankl
movlw 0xFF      ;w=0xFF
movwf TRISA     ;address of TRISA register is taken from
                ;instruction movwf
```



Adresarea Direct•

Adresarea Indirect•

Adresarea indirect• spre deosebire de cea direct• nu ia o adres• dintr-o instruc•iune ci o creeaz• cu ajutorul bitului IRP a regi•trilor STATUS •i FSR. Loca•ia adresat• este accesat• prin registrul INDF care de fapt •ine o adres• indicat• de un FSR. Cu alte cuvinte, orice instruc•iune care folose•te INDF ca registrul al ei, în realitate acceseaz• datele indicate de un registru FSR. S• spunem, de exemplu, c• un registru cu scop general (GPR) la adresa 0Fh con•ine o valoarea 20. Prin scrierea unei valori 0Fh în registrul FSR vom ob•ine un registru indicator la adresa 0Fh, iar prin citirea din registrul INDF, vom ob•ine valoarea 20, ceea ce înseamn• c• am citit din primul registru valoarea lui f•r• accesarea lui direct• (dar prin FSR •i INDF). Se pare c• acest tip de adresare nu are nici un avantaj fa•• de adresarea direct•, dar exist• unele nevoi în timpul program•rii ce se pot rezolva mai simplu doar prin adresarea indirect•.



Adresarea Indirectă

Un asemenea exemplu poate trimite un set de date prin comunica•ia serial•, lucrând cu bufere •i indicatoare (ce vor fi discutate în continuare într-un capitol cu exemple), sau s• •tearg• o parte a memoriei RAM (16 loca•ii) ca în urm•torul exemplu.

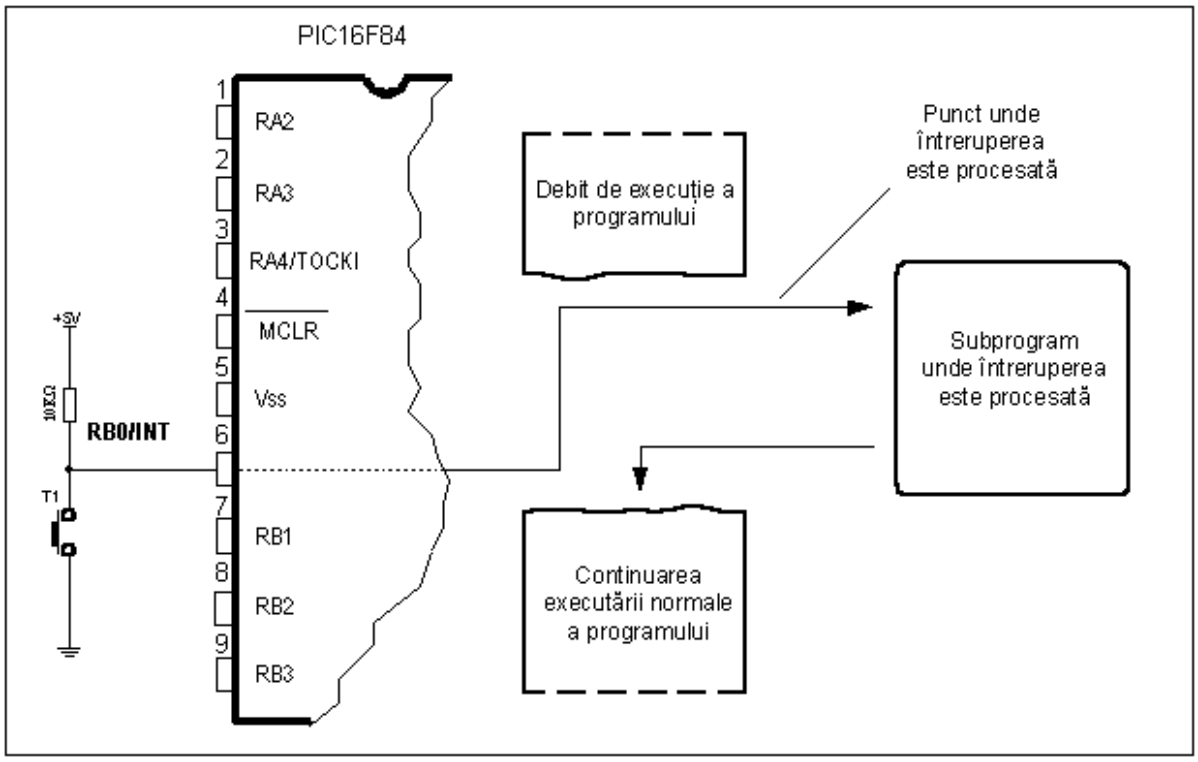
```
      Movlw 0x0C        ;initialization of starting address
      Movwf FSR         ;FSR indicates address 0x0C
LOOP  clrf INDF         ;INDF = 0
      incf FSR          ;address = initial address + 1
      btfss FSR,4       ;are all locations erased
      goto loop         ;no, go through a loop again
CONTINUE
      :                 ; yes, continue with program
```

Citind datele din registrul INDF când con•inutul registrului FSR este egal cu zero, întoarce valoarea zero, •i scrie în el rezultatul în opera•ia NOP (no operation- nu opereaz•).

Pagina anterioar•	Con•inut	Pagina urm•toare

2.6 Întreruperi

Întreruperile sunt un mecanism a unui microcontroler ce îi permit s• r•spund• la unele evenimente la momentul când se întâmpl•, indiferent de ce face atunci microcontrolerul. Aceasta este o parte foarte important•, pentru c• permite conexiunea microcontrolerului cu lumea de afar•. În general, fiecare întrerupere schimb• debitul programului, îl întrerupe •i dup• executarea unui subprogram (rutine de întrerupere), continu• din acela•i punct.



Una din posibilele surse de întreruperi şi cum afectează programul principal

Registrul de control al unei întreruperi se nume•te INTCON •i se g•se•te la adresa 0Bh. Rolul lui este de a permite sau interzice cererile de întreruperi, •i în caz c• nu sunt permise, înregistreaz• cererile de întrerupere singulare prin bi•ii lui.

Registru 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 neimplementat, citit ca '0'	**-n** = Valoarea la resetul power-on

bit 0 **RBIF** (RB Port Change Interrupt Flag bit-bit Stegule• de Întrerupere a Schimb•rii Portului RB) Bit ce informeaz• despre schimb•rile de la pinii 4, 5, 6 •i 7 ai portului B.
1=cel pu•in un pin •i-a schimbat starea
0=nu s-a întâmplat nici o schimbare la vreun pin

bit 1 **INTF** (INT External Interrupt Flag bit-bit Stegule• de Întrerupere Extern• INT) A avut loc o întrerupere extern•.
1=a avut loc o întrerupere
0=nu a avut loc o întrerupere

Dac• s-a detectat un front cresc•tor sau descresc•tor la pinul RB0/INT, (ce este definit cu bitul INTEDG în registrul OPTION), bitul INTF este setat. Bitul trebuie s• fie •ters în subprogramul întrerupere pentru a detecta urm•toarea întrerupere.

bit 2 **T0IF** (TMR0 Overflow Interrupt Flag bit-bit Stegule• Dep••ire Întrerupere TMRO) Dep••irea contorului TMRO.
1=contorul •i-a schimbat starea de la FFh la 00h.
0=dep••irea nu a avut loc
Bitul trebuie s• fie •ters în program pentru ca o întrerupere s• fie detectat•.

bit 3 **RBIE** (RB port change Interrupt Enable bit-bit Permite Întreruperea schimb•rii portului RB) Permite s• aib• loc întreruperi la schimbarea st•rii pinilor 4, 5, 6, •i 7 ai portului B.
1=permite întreruperi la schimbarea st•rii
0=întreruperi interzise la schimbarea st•rii
Dac• RBIE •i RBIF au fost simultan setate, va avea loc o întrerupere.

bit 4 **INTE** (INT External Interrupt Enable bit-bit Permite Întrerupere extern• INT) Bit ce permite întreruperea extern• de la pinul RB0/INT.
1=întrerupere extern• permis•
0=întrerupere extern• interzis•
Dac• INTE •i INTF au fost setate simultan, va avea loc o întrerupere.

bit 5 **T0IE** (TMR0 Overflow Interrupt Enable bit-bit Permite Dep••ire Întrerupere TMRO) Bit ce permite întreruperile în timpul dep••irii contorului TMRO.
1=întrerupere permis•
0=întrerupere interzis•
Dac• T0IE •i T0IF au fost simultan setate, va avea loc întreruperea.

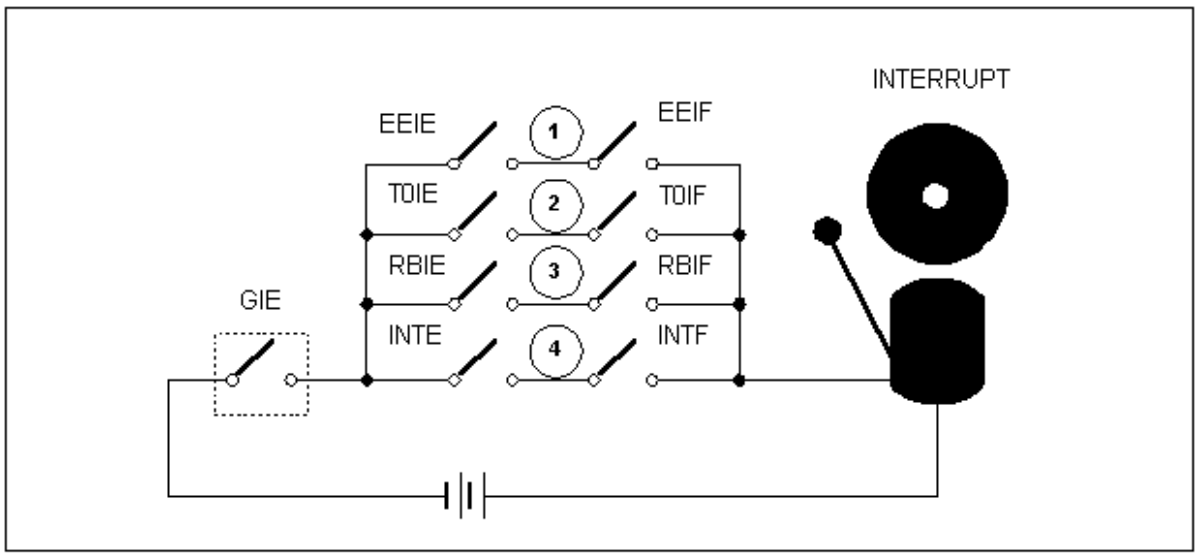
Bit 6 **EEIE** (EEPROM Write Complete Interrupt Enable bit-bit Permite Întrerupere Complet• a Scrierii EEPROM) Bit ce permite o întrerupere la sfâr•itul unei rutine de scriere în EEPROM
1= întrerupere permis•
0= întrerupere interzis•
Dac• EEIE •i EEIF (ce este în registrul EECON1) au fost simultan setate, va avea loc o întrerupere.

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

PIC16F84 are patru surse de întrerupere:

1. Terminarea scrierii datelor în EEPROM
2. Întrerupere TMR0 cauzat• de dep••irea timer-ului
3. Întrerupere în timpul schimb•rii la pinii RB4, RB5, RB6 •i RB7 ai portului B.
4. Întrerupere Extern• de la pinul RB0/INT al microcontrolerului

În general, fiecare surs• de întrerupere are doi bi•i lega•i la ea. Unul permite întreruperea, iar cel•lalt detecteaz• când au loc întreruperi. Exist• un bit comun numit GIE ce poate fi folosit pentru a interzice sau permite toate întreruperile simultan. Acest bit este foarte folositor când se scrie un program pentru c• permite ca toate întreruperile s• fie interzise pentru o perioad• de timp, a•a ca execu•ia unei p•r•i importante a programului s• nu fie întrerupt•. Când instruc•iunea ce reseteaz• bitul GIE a fost executat• (GIE=0, toate întreruperile interzise), fiecare întrerupere ce r•mâne nerezolvat• trebuie ignorat•.



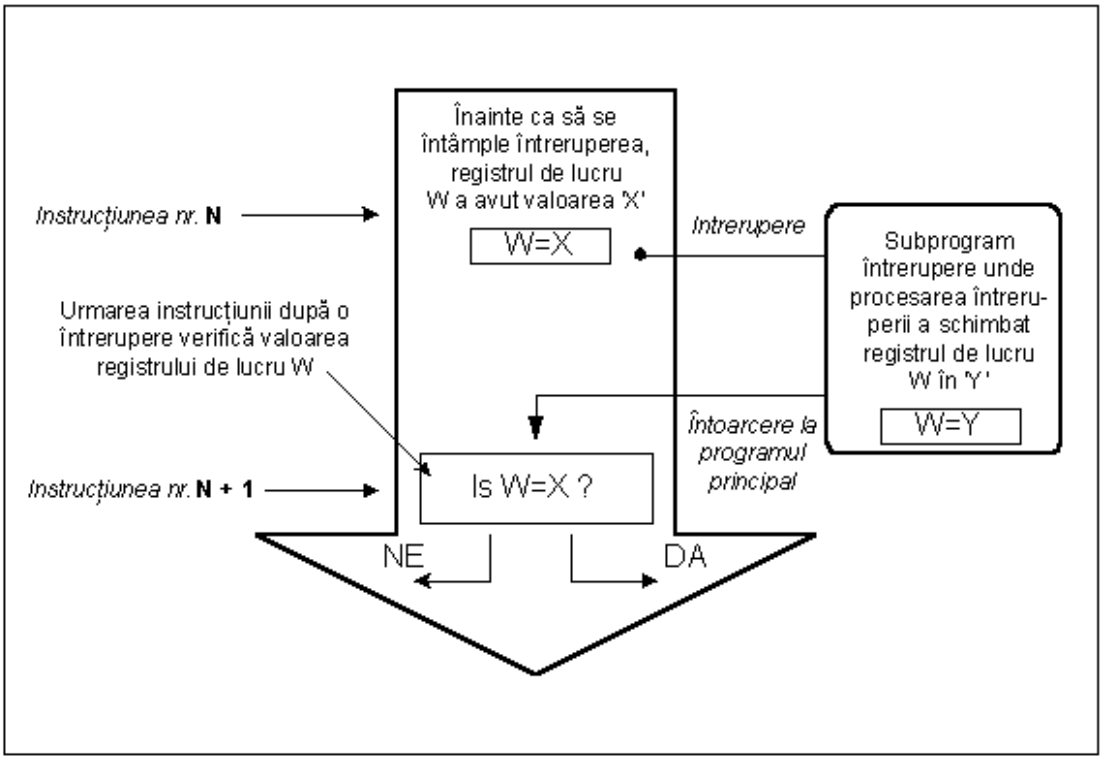
Schiță simplificată a întreruperilor microcontrolerului PIC16F84

Întreruperile ce r•mân nerezolvate •i ce au fost ignorate, sunt procesate când bitul GIE (GIE=1, toate întreruperile sunt permise) va fi •ters. Când i s-a r•spuns întreruperii, bitul GIE a fost •ters, a•a c• orice întreruperi adi•ionale vor fi interzise, adresa de întoarcere a fost trimis• în stiv•, iar adresa 0004h a fost scris• în contorul programului – numai dup• aceasta începe r•spunsul la o întrerupere! Dup• ce este procesat• întreruperea, bitul a c•rui setare a cauzat o întrerupere trebuie •ters, sau rutina de întrerupere va fi procesat• automat tot mereu în timpul întoarcerii la programul principal.

P•strarea con•inutului regi•trilor importan•i

Doar valoarea de întoarcere a contorului programului este înmagazinat• într-o stiv• în timpul unei întreruperi (prin valoare de întoarcere a contorului programului în•elegem adresa instruc•iunii ce trebuie executat•, dar nu a fost executat• pentru c• a avut loc întreruperea). P•strând doar valoarea contorului programului adesea nu este suficient. Unii regi•tri ce sunt în uz în programul principal pot fi de asemenea în uz în rutina de întrerupere. Dac• ei nu sunt re•inu•i, programul principal va ob•ine valori complet diferite în acei regi•tri în timpul întoarcerii dintr-o rutin• de întrerupere, ceea ce va cauza erori în program. Un exemplu de asemenea caz este con•inutul registrului de lucru W. Dac• presupunem c• programul principal a folosit registrul de lucru W pentru unele din opera•iile sale, •i c• a p•strat în el o valoare ce este important• pentru urm•toarea instruc•iune, atunci o întrerupere ce se va întâmpla înainte de acea instruc•iune va schimba valoarea registrului de lucru W, ce va influen•a direct programul principal.

Procedura de înregistrare de regi•tri importan•i înainte de a merge la o rutin• de întrerupere se nume•te PUSH, în timp ce procedura ce aduce valorile înregistrate înapoi, se nume•te POP. PUSH •i POP sunt instruc•iuni ale altor microcontrolere (Intel), dar sunt atât de larg acceptate c• o întreag• opera•ie este numit• dup• ele. PIC16F84 nu are instruc•iuni ca PUSH •i POP, •i ele trebuie s• fie programate.



Unul din posibilele cazuri de erori dacă nu s-a făcut salvarea când s-a mers la un sub-program al unei întreruperi

Datorit• simplit••ii •i folosirii frecvente, aceste p•r•i ale programului pot fi f•cute ca macro-uri. Conceptul unui Macro este explicat în "Limbaj de asamblare program". În urm•torul exemplu, con•inuturile regi•trilor W •i STATUS sunt memorate în variabilele W_TEMP •i STATUS_TEMP înainte de rutina de întrerupere. La începutul rutinei PUSH trebuie s• verific•m bancul selectat în prezent pentru c• W_TEMP and STATUS_TEMP nu se g•sesc în bancul 0. Pentru schimbul de date între ace•ti regi•tri, instruc•iunea SWAPF se folose•te în loc de MOVF pentru c• nu afecteaz• starea bi•ilor registrului STATUS.

Exemplul este un program asamblor pentru urm•torii pa•i :
1. Testarea bancului curent
2. Stocarea registrului W indiferent de bancul curent
3. Stocarea registrul STATUS în bancul 0
4. Executarea rutinei de întrerupere pentru procesul de întrerupere (ISR)
5. Restaureaz• registrul STATUS
6. Restaureaz• registrul W

Dac• mai sunt •i alte variabile sau regi•tri ce trebuie stoca•i, atunci ei trebuie s• fie p•stra•i dup• stocarea registrului STATUS (pasul 3), •i adu•i înapoi înainte ca registrul STATUS s• fie restaurat (pasul 5).

```
Push
        BTFSS STATUS, RP0           ; Bank0
        GOTO RP0CLEAR               ; Yes
        BCF STATUS, RP0             ; NO, go to Bank0
        MOVWF W_TEMP                ; Save W register
        SWAPF STATUS, W             ; W <- STATUS
        MOVWF STATUS_TEMP           ; STATUS_TEMP <- W
        BSF STATUS_TEMP, 1          ; RP0(STATUS_TEMP)=1
        GOTO ISR_Code               ; Push completed
RP0CLEAR
        MOVWF W_TEMP                ; Save W register
        SWAPF STATUS, W             ; W <- STATUS
        MOVWF STATUS_TEMP           ; STATUS_TEMP <- W
;
ISR_Code
        :
        : (Interrupt subprogram )
        :
;
Pop
        SWAPF STATUS_TEMP, W        ; W <- STATUS_TEMP
        MOVWF STATUS                ; STATUS <-W
        BTFSS STATUS, RP0           ; Bank1?
        GOTO Return_WREG            ; NO,
        BCF STATUS, RP0             ; YES, go to Bank0
        SWAPF W_TEMP, F             ; Return contents of  W register
        SWAPF W_TEMP, W             ;
        BSF STATUS, RP0             ; Return to Bank1
        RETFIE                      ; POP complete
Return_WREG
        SWAPF W_TEMP, F             ; Return contents of  W register
        SWAPF W_TEMP, W             ;
        RETFIE                      ; POP completed
```

Acela•i exemplu se poate realiza utilizând macro-uri, f•când astfel programul mai eligibil. Macro-urile ce sunt deja definite, pot fi folosite pentru scrierea de noi macro-uri. Macro-urile BANK1 •i BANK0 ce sunt explicate în capitolul "Organizarea memoriei" sunt folosite cu macro-urile 'push' •i 'pop'.

```
push    macro
        movwf  W_Temp               ;W_Temp <- W
        swapf  W_Temp,F             ;Swap them
        BANK1                       ;Macro for switching to Bank1
        swapf  OPTION_REG,W         ;W <- OPTION_REG
        movwf  Option_Temp          ;Option_Temp <- W
        BANK0                       ;macro for switching to Bank0
        swapf  STATUS,W             ;W <- STATUS
        movwf  Stat_Temp            ;Stat_Temp <-W
        endm                        ;End of push macro

pop     macro
        swapf  Stat_Temp,W          ;W <- Stat_Temp
        movwf  STATUS               ;STATUS <- W
        BANK1                       ;Macro for switching to Bank1
        swapf  Option_Temp,W        ;W <- Option_Temp
        movwf  OPTION_REG           ;OPTION_REG <- W
        BANK0                       ;Macro for switching to Bank0
        swapf  W_Temp,W             ;W <- W_Temp
        endm                        ;End of a pop macro
```

Întrerupere extern• la pinul RB0/INT al microcontrolerului

Întreruperea extern• la pinul RB0/INT este triggerat• de frontul cresc•tor (dac• bitul INTEDG=1 în registrul OPTION<6>), sau de frontul descresc•tor (dac• INTEDG=0). Când apare semnalul corect la pinul INT, bitul INTF este setat la registrul INTCON. Bitul INTF (INTCON<1>) trebuie resetat în rutina de întrerupere, a•a ca întreruperea s• nu aib• loc din nou în timpul întoarcerii la programul principal. Acesta este un pas important al programului pe care programatorul nu trebuie s•-l uite, sau programul va merge constant în rutina de întrerupere. Întreruperea poate fi închis• prin resetarea bitului de control INTE (INTCON<4>).

Întreruperea în timpul dep••irii contorului TMRO

Dep••irea contorului TMRO (de la FFh la 00h) va seta bitul T0IF (INTCON<2>). Aceasta este o întrerupere foarte important• pentru c• multe probleme reale se por rezolva folosind aceast• întrerupere. Unul din exemple este

m•surarea timpului. Dac• •tim cât timp are nevoie contorul pentru a completa un ciclu de la 00h to FFh, atunci num•rul de întreruperi înmul•it cu acea durat• de timp va da timpul total scurs. În rutina de întrerupere unele variabile vor fi incrementate în memoria RAM, valoarea acelei variabile înmul•ite cu timpul de care are nevoie contorul pentru a contoriza într-un ciclu întreg, va da timpul total scurs. Întreruperea poate fi pornit•/oprit• prin setarea/resetarea bitului T0IE (INTCON<5>).

Întrerupere pe timpul unei schimb•ri la pinii 4, 5, 6 •i 7 ai portului B

Schimbarea semnalului de intrare la PORTB <7:4> seteaz• bitul RBIF (INTCON<0>). Patru pini RB7, RB6, RB5 •i RB4 ai portului B, pot triggera o întrerupere ce are loc când starea la ei se schimb• de la unu la zero logic, sau viceversa. Pentru ca pinii s• fie sensibili la aceast• schimbare, trebuie defini•i ca intrare. Dac• oricare din ei este definit ca ie•ire, întreruperea nu va fi generat• la schimbarea st•rii. Dac• ei sunt defini•i ca intrare, starea lor curent• este comparat• cu vechea valoare ce a fost stocat• la ultima citire de la portul B. Întreruperea poate fi pornit•/oprit• prin setarea/ resetarea bitului RBIE în registrul INTCON.

Întreruperea la terminarea subrutinei write în EEPROM

Aceast• întrerupere este doar de natur• practic•. Pentru c• scrierea într-o loca•ie EEPROM dureaz• cam 10ms (care este o durat• lung• în termenii microcontrolerului), nu este rentabil de a a•tepta pân• la cap•t scrierea. Este ad•ugat astfel mecanismul de întrerupere ceea ce permite microcontrolerului s• continue executarea programului principal, în timp ce scrierea în EEPROM este f•cut• în plan secundar. Când scrierea este terminat•, întreruperea informeaz• microcontrolerul c• scrierea s-a terminat. Bitul EEIF, prin care se face aceast• informare, se g•se•te în registrul EECON1. Producerea unei întreruperi poate fi interzis• prin resetarea bitului EEIE în registrul INTCON.

Ini•ializarea întreruperii

Pentru a folosi un mecanism de întrerupere a unui microcontroler, trebuie f•cute unele sarcini preg•titoare. Aceste proceduri sunt pe scurt numite "ini•ializare". Prin ini•ializare definim la ce va r•spunde microcontrolerul, •i ce va ignora. Dac• nu set•m bitul ce permite o anumit• întrerupere, programul nu va executa un subprogram întrerupere. Prin aceasta putem ob•ine controlul asupra producerii întreruperii, ceea ce este foarte folositor.

```
clrf INTCON             ; all interrupts disabled
movlw B'00010000'       ; external interrupt only is enabled
bsf INTCON, GIE         ; occurrence of interrupts allowed
```

Exemplul de mai sus arat• ini•ializarea unei întreruperi externe la pinul RB0 al microcontrolerului. Unde se vede unu setat, înseamn• c• întreruperea este permis•. Producerea altor întreruperi nu este permis•, •i toate întreruperile împreun• sunt interzise pân• ce bitul GIE este •inut în unu.

Urm•torul exemplu arat• o cale tipic• de a dirija întreruperile. PIC16F84 are doar o loca•ie unde adresa unui subprogram întrerupere este memorat•. Aceasta înseamn• c• mai întâi trebuie s• detect•m ce întrerupere este la îndemân• (dac• mai mult de o surs• de întreruperi este disponibil•), •i apoi putem executa acea parte a programului ce se refer• la acea întrerupere.

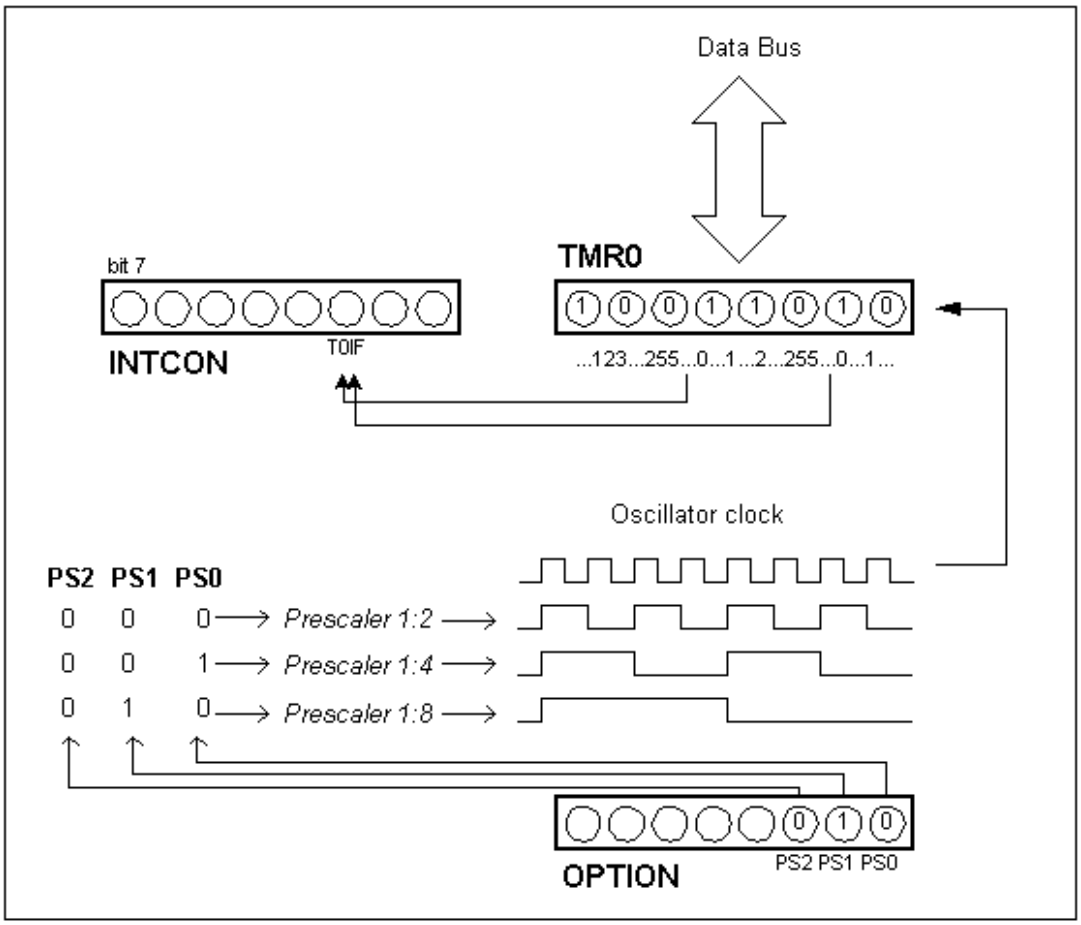
```
org ISR_ADDR            ;ISR_ADDR is interrupt routine address
btfsc INTCON, GIE       ;GIE bit turned off?
goto ISR_ADR            ;no, go back to the beginning
PUSH                    ;keep the contents of important registers
btfsc INTCON, RBIF      ;change on pins 4, 5, 6 and 7 of port B?
goto ISR_PORTB          ;jump to that section
btfsc INTCON,INTF       ;external interrupt occured?
goto ISR_RBO            ;jump to that part
btfsc INTCON, TOIF      ;overflow of timer TMRO?
goto ISR_TMRO           ;jump to that section
BANK1                   ;Bank1 because of EECON1
Btfsc EECON1, EEIF      ;writing to EEPROM completed?
goto ISR_EEPROM         ;jump to that section
BANKO                   ;BankO

ISR_PORTB
      :                 ;section of code which is processed by an
                        ;interrupt ?
      :
      goto END_ISR      ;jump to the exit of an interrupt
ISR_RBO
      :                 ;section of code processing an interrupt?
      :
      goto END_ISR      ;jump to exit of an interrupt.
ISR_TMRO
      :                 ;section of code processing an interrupt
      :
      goto END_ISR      ;jump to the exit of an interrupt
ISR_EEPROM
      :                 ;section of code which processes an interrupt
      :
      goto END_ISR      ;jump to an exit from an interrupt.
END_ISR                 ;
      POP               ;bringing back the contents of important
                        ;registers
      RETFIE            ;return and setting of GIE bit
```

Reîntoarcerea dintr-o rutin• de întrerupere poate fi f•cut• cu instruc•iunile RETURN, RETLW •i RETFIE. Se recomand• ca s• fie utilizat• instruc•iunea RETFIE pentru c• acea instruc•iune este singura ce seteaz• automat bitul GIE, ceea ce permite s• se produc• o nou• întrerupere.

Pagina anterioar•	Con•inut	Pagina urm•toare

2.7 Timer-ul liber TMR0

Timer-ele (temporizatoarele) sunt de obicei cele mai complicate p•r•i ale unui microcontroler, a•a c• este necesar s• rezerv•m mai mult timp pentru a le explica. Odat• cu aplicarea lor este posibil s• se creeze rela•ii între o dimensiune real• ca "timp" •i o variabil• ce reprezint• starea timer-ului într-un microcontroler. Fizic, timer-ul este un registru a c•rui valoare cre•te continuu pân• la 255, •i apoi porne•te de la cap•t: 0, 1, 2, 3, 4...255....0,1, 2, 3.....etc.



Relaţia dintre timer-ul TMRO şi prescaler

Aceast• incrementare se face în fundalul a tot ceea ce face un microcontroler. Depinde de programator "s• g•seasc• o cale" de cum s• profite de aceast• caracteristic• pentru nevoile lui. Una din c•i este s• creasc• o variabil• la fiecare dep••ire a timer-ului. Dac• •tim cât timp are nevoie timer-ul s• fac• o rund• complet•, atunci înmul•ind valoarea variabilei cu acel timp ob•inem timpul total scurs.

PIC16F84 are un timer de 8 bi•i. Num•rul de bi•i determin• pân• la ce valoare contorizeaz• timer-ul înainte de a începe s• contorizeze de la zero din nou. În cazul unui timer de 8 bi•i, acel num•r este 256. O schem• simplificat• a rela•iei dintre un timer •i un prescaler-divizor este reprezentat• în diagrama anterioar•. Prescalerul este numele acelei p•r•i din microcontroler ce divide ceasul oscilatorului înainte de a ajunge la logica ce cre•te starea timer-ului. Num•rul ce divide un ceas este definit prin trei bi•i în registrul OPTION. Cel mai mare divizor este 256. Aceasta înseamn• de fapt c• doar la al fiecare 256-lea ceas, valoarea timer-ului va cre•te cu unu. Aceasta ne d• posibilitatea de a m•sura perioade de timp mai lungi.

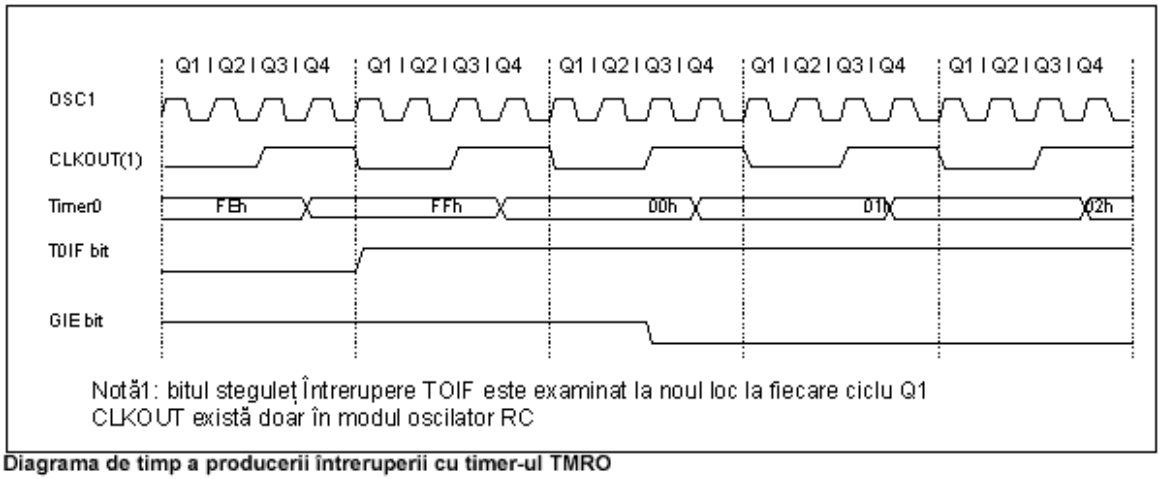
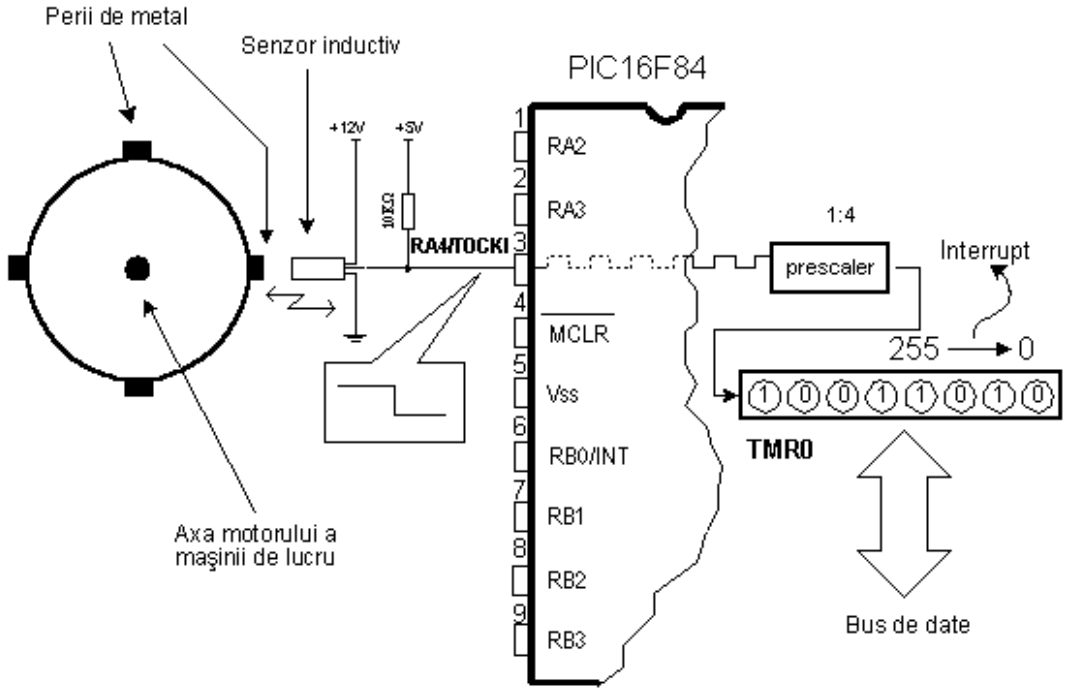


Diagrama de timp a producerii întreruperii cu timer-ul TMR0

Dup• fiecare num•r•toare pân• la 255, timer-ul î•i reseteaz• valoarea la zero •i începe cu un nou ciclu de contorizare pân• la 255. În timpul fiec•rei tranzi•ii de la 255 la zero, bitul TOIF în registrul INTCON este setat. Dac• se permit întreruperi, de aceasta se poate profita în generarea •i în procesarea rutinei de întrerupere. Depinde de programator s• reseteze bitul TOIF în rutina de întrerupere, a•a ca noua întrerupere, sau noua dep••ire s• fie detectate. În afar• de ceasul oscilator intern, starea timer-ului poate de asemenea s• creasc• prin ceasul extern la pinul RA4/TOCKI. Alegerea uneia din aceste dou• op•iuni se face în registrul OPTION prin bitul TOCS. Dac• a fost aleas• aceast• op•iune de ceas extern, va fi posibil s• se defineasc• frontul unui semnal (cresc•tor sau descresc•tor), la care timer-ul s•-•i creasc• valoarea.



Determinarea numărului rotațiilor complete ale axului motorului

În practic•, unul din exemplele tipice ce este rezolvat prin ceas extern •i unde timer-ul contorizeaz• rota•iile complete ale unui ax al unei ma•ini de produc•ie, ca bobinatorul de transformator de exemplu. S• rotim patru •uruburi de metal pe axul unui bobinator. Aceste patru •uruburi vor reprezenta convexitatea metalic•. S• plas•m acum un senzor inductiv la o distan•• de 5 mm de cap•tul unui •urub. Senzorul inductiv va genera semnalul descresc•tor de fiecare dat• când capul •urubului este paralel cu capul senzorului. Fiecare semnal va reprezenta o p•trime dintr-o rota•ie, •i suma tuturor rota•iilor se va g•si în timer-ul TMR0. Programul poate u•or citi aceste date din timer printr-un bus de date.

Urm•torul exemplu ilustreaz• cum s• se ini•ializeze timer-ul la fronturile descresc•toare ale semnalului din sursa extern• cu un prescaler 1:4. Timer-ul lucreaz• în mod "polig-împingere".

```
      clrf TMR0         ;TMR0=0
      clrf INTCON       ;Interrupts and T0IF=0 disallowed
      bsf STATUS,RP0    ;Bank1 because of OPTION_REG
      movlw B'00110001' ;prescaler 1:4, falling edge selected external
                        ;clock source and pull up ;selected resistors
                        ;on port B activated
      movwf OPTION_REG ;OPTION_REG <- W
T0_OVFL
      btfss INTCON, T0IF      ;testing overflow bit
      goto T0_OVFL            ;interrupt has not occured yet, wait
;
; (Part of the program which processes data regarding a number of turns)
;
goto T0_OVFL                  ;waiting for new overflow
```

Acela•i exemplu poate fi realizat printr-o întrerupere în modul urm•tor:

```
push  macro
      movwf W_Temp            ;W_Temp <- W
      swapf W_Temp,F          ;Swap them
      BANK1                   ;Macro for switching to Bank1
      swapf OPTION_REG,W      ;W <- OPTION_REG
      movwf Option_Temp       ;Option_Temp <- W
      BANK0                   ;macro for switching to Bank0
      swapf STATUS,W          ;W <- STATUS
      movwf Stat_Temp         ;Stat_Temp <-W
      endm                    ;End of push macro

pop   macro
      swapf Stat_Temp,W       ;W <- Stat_Temp
      movwf STATUS            ;STATUS <- W
      BANK1                   ;Macro for switching to Bank1
      swapf Option_Temp,W     ;W <- Option_Temp
      movwf OPTION_REG        ;OPTION_REG <- W
      BANK0                   ;Macro for switching to Bank0
      swapf W_Temp,W          ;W <- W_Temp
      endm                    ;End of a pop macro
```

Prescalerul poate fi asignat fie de timer-ul TMRO fie de watchdog. Watchdogul este un mecanism pe care microcontrolerul îl folose•te s• se apere împotriva bloc•rii programelor. Ca orice alt circuit electric, la fel •i cu microcontrolerul se pot întâmpla defect•ri, sau unele stric•ciuni. Din nefericire microcontrolerul are de asemenea un program unde se pot întâmpla probleme. Când se întâmpl• aceasta, microcontrolerul se va opri din func•ionare •i va r•mâne în acea stare pân• ce cineva îl reseteaz•. Din cauza aceasta, a fost introdus mecanismul watchdog. Dup• o anumit• perioad• de timp, watchdogul reseteaz• microcontrolerul (de fapt microcontrolerul se reseteaz• singur). Watchdogul luceaz• pe baza unui principiu simplu: dac• se întâmpl• dep••irea timer-ului, microcontrolerul este resetat, •i începe executarea programului mereu din nou. Astfel, se va întâmpla un reset atât în cazul unei func•ion•ri corecte cât •i incorecte. Urm•torul pas este prevenirea resetului în cazul unei func•ion•ri corecte, ce se face prin scrierea unui zero în registrul WDT (instruc•iunea CLRWDT) de fiecare dat• când se apropie de dep••ire. Astfel programul va preveni un reset cât timp este executat corect. De îndat• ce s-a blocat, nu se va scrie zero, va avea loc dep••irea timer-ului WDT •i un reset ce va duce microcontrolerul înapoi la func•ionarea corect• din nou.

Prescalerul este acordat cu timer-ul TMRO, sau cu timer-ul watchdogului prin bitul PSA în registrul OPTION. •tergând bitul PSA, prescalerul va fi acordat cu timer-ul TMRO. Când prescalerul este acordat cu timer-ul TMRO, toate instruc•iunile de scriere în registrul TMRO (CLRF TMR0, MOVWF TMR0, BSF TMR0,...) vor •terge prescalerul. Când prescalerul este asignat timerului watchdog, numai instruc•iunea CLRWDT va •terge prescalerul •i timer-ul watchdog în acela•i timp. Schimbarea prescalerului este complet• sub controlul programatorului, •i poate fi schimbat în timp ce se ruleaz• programul.

Exist• doar un prescaler •i un timer. Func•ie de nevoi, ele sunt asignate fie timer-ului TMRO fie watchdog-ului.

Registrul control 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
$\overline{\text{RBPU}}$ (1)	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

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 Selectare Rat• Prescaler)
Subiectul prescaler, •i cum afecteaz• ace•ti bi•i lucrul unui microcontroler va fi abordat în sec•iunea despre 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 3 **PSA** (Prescaler Assignment bit-bit Asignare Prescaler)
Bit ce asigneaz• prescalerul între TMRO •i timer-ul watchdog).
1=prescalerul este asignat la timer-ul watchdog
0=prescalerul este asignat la timer-ul free-liber

bit 4 **T0SE** (TMR0 Source Edge Select bit-bit Selectare Front Surs• TMRO)

Dac• triggerul TMRO a fost activat cu impulsuri de la pinul RA4/T0CKI, acest bit va determina dac• va fi la frontul cresc•tor sau descresc•tor al semnalului.
1=front descresc•tor
0=front cresc•tor

bit 5 **T0CS** (TMR0 Clock Source Select bit-bit Selectare Surs• Ceas TMRO)
Acest bit permite unui timer free-run s•-•i incrementeze valoarea fie de la oscilatorul intern, de exemplu ¼ din ceasul oscilatorului, sau prin impulsuri externe la pinul RA4/T0CKI.
1=impulsuri externe
0=1/4 ceas intern

bit 6 I**NTEDG** (Interrupt Edge Select bit-bit Selectare Front Întreruperi)
Dac• a fost permis• producerea de întreruperi, acest bit va determina la ce front va avea loc întreruperea la pinul RB0/INT.
1=front cresc•tor
0=front descresc•tor

bit 7 **RBPU** (PORTB Pull-up Enable bit-bit Permite Pull-up-tragerea PORTB)
Acest bit deschide sau închide rezistorii interni la portul B.
1=rezistorii 'pull-up' deschi•i
0=rezistorii 'pull-up' închi•i

Pagina anterioar• **Con•inut** **Pagina urm•toare**

2.8 Memoria de date EEPROM

PIC16F84 are 64 de bytes de loca•ii de memorie EEPROM la adresele de la 00h la 63h unde se poate scrie sau de unde se poate citi. Cea mai important• caracteristic• a acestei memorii este c• nu pierde con•inutul în timpul închideri sursei de alimentare. Aceasta înseamn• practic c• ceea ce a fost scris în ea va r•mâne chiar •i când microcontrolerul este închis. Datele pot fi re•inute în EEPROM f•r• sursa de alimentare pân• la 40 de ani. (dup• cum declar• produc•torul lui PICD16F84), •i se pot executa 10000 de cicluri de scriere.

În practic•, memoria EEPROM este folosit• pentru stocarea unor date importante sau a unor parametri de proces. Un asemenea parametru este o temperatur• dat•, asignat• când se seteaz• un regulator de temperatur• la un proces. Dac• nu s-a re•inut, va fi nevoie s• se ajusteze temperatura dat• dup• fiecare întrerupere a aliment•rii. Pentru c• aceasta este foarte nepractic (chiar periculos), produc•torii de microntrolere au început s• instaleze un tip mai mic de memorie EEPROM.

Memoria EEPROM este plasat• într-un loc special al memoriei •i poate fi accesat• prin regi•tri speciali. Ace•ti regi•tri sunt:

- **EEDATA** la adresa 08h, care re•ine datele de citit sau cele de scris.
- **EEADR** la adresa 09h, ce con•ine o adres• a loca•iei EEPROM ce este accesat•.
- **EECON1** la adresa 88h, ce con•ine bi•i de control.
- **EECON2** la adresa 89h. Acest registru nu exist• fizic •i serve•te la protejarea EEPROM-ului de scrieri accidentale.

Registrul EECON1 la adresa 88h este un registru de control cu 5 bi•i implementa•i.
Bi•ii 5, 6 •i 7 nu sunt folosi•i, •i prin citire sunt totdeauna zero. Interpretarea bi•ilor registrului EECON1 urmeaz•.

Registrul EECON1

U-0	U-0	U-0	R/W-1	R/W-1	R/W-x	R/S-0	R/S-x
—	—	—	EEIF (1)	WRERR	WREN	WR	RD
bit 7							bit 0

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

bit 0 **RD** (Read Control bit-bit Control Citire)
Setarea acestui bit ini•ializeaz• transferul de date definit în EEADR la registrul EEDATA. Pentru c• timpul nu este esen•ial în citirea datelor ca la scriere, datele din EEDATA pot fi deja folosite în urm•toarea instruc•iune.
1=initializeaz• citirea
0=nu ini•ializeaz• citirea

bit 1 **WR** (Write Control bit-bit Control Scriere)
Setarea acestui bit ini•ializeaz• scrierea datelor din registrul EEDATA la adresa specifcat• prin registrul EEADR.
1=initializeaz• scrierea
0=nu ini•ializeaz• scrierea

bit 2 **WREN** (EEPROM Write Enable bit-bit Permite Scrierea EEPROM) Permite scrierea în EEPROM
Dac• acest bit nu a fost setat, microcontrolerul nu va permite scrierea în EEPROM.
1=scriere permis•
0=scriere interzis•

bit 3 **WRERR** (Write EEPROM Error Flag-Stegule• Eroare Scriere EEPROM) Eroare în timpul scrierii în EEPROM
Acest bit a fost setat doar în caz c• scrierea în EEPROM a fost intrerupt• de un semnal sau prin terminarea timpului din timer-ul watchdog (dac• este activat).
1=a avut loc eroare
0=nu a avut loc eroare

bit 4 **EEIF** (EEPROM Write Operation Interrupt Flag bit-bit Stegule• Întrerupere Opera•ie Scriere EEPROM) Bit folosit pentru a informa c• scrierea datelor s-a terminat.
Când s-a terminat scrierea, acest bit va fi setat automat. Programtorul trebuie s• •tearg• bitul EEIF în programul s•u pentru a detecta noua terminare a scrierii.
1=scrierea terminat•
0=scrierea înc• neterminat•, sau înc• nu a început

Citirea din memoria EEPROM

Setarea bitului RD ini•ializeaz• transferul de date de la adresa g•sit• în EEADR la registrul EEDATA. Ca •i la citirea datelor nu avem nevoie de atât de mult timp ca la scriere, datele luate din registrul EEDATA pot deja fi folosite mai departe în urm•toarea instruc•iune.

O mostr• a p•r•ii programului ce cite•te datele în EEPROM, ar putea ar•ta ca mai jos:

```
bcf   STATUS, RP0       ;bank0, because EEADR is at 09h
movlw 0x00              ;address of location being read
movwf EEADR             ;address transferred to EEADR
bsf   STATUS, RP0       ;bank1 because EECON1 is at 88h
bsf   EECON1, RD        ;reading from EEPROM
bcf   STATUS, RP0       ;Bank0 because EEDATA is at 08h
movf  EEDATA, W         ;W <-- EEDATA
```

Dup• ultima instruc•iune de program, con•inutul de la o adres• EEPROM zero poate fi g•sit în registrul w.

Scrierea în memoria EEPROM

Pentru a scrie datele în loca•ia EEPROM, programatorul trebuie mai întâi s• scrie adresa în registrul EEADR •i datele în registrul EEDATA. Numai atunci este folositor de a seta bitul WR ce pune totul în mi•care. Bitul WR va fi resetat, •i bitul EEIF setat urmând o scriere ce poate fi folosit• în procesarea întreruperilor. Valorile 55h •i AAh sunt prima •i a doua cheie care interzic ca scrierea accidental• în EEPROM s• se întâmple. Aceste dou• valori sunt scrise în EECON2 care serve•te doar pentru acel scop, de a primi aceste dou• valori •i de a preveni orice scriere accidental• în memoria EEPROM. Liniile de program marcate ca 1, 2, 3 •i 4 trebuie s• fie executate în acea ordine în intervale egale de timp. De aceea este foarte important, s• închide•i întreruperile ce ar putea schimba timpul necesar pentru executare instruc•iunilor. Dup• scriere, întreruperile, pot fi permise din nou.

Exemplu unei p•r•i a programului ce scrie datele 0xEE în prima loca•ie în memoria EEPROM ar putea ar•ta ca mai jos:

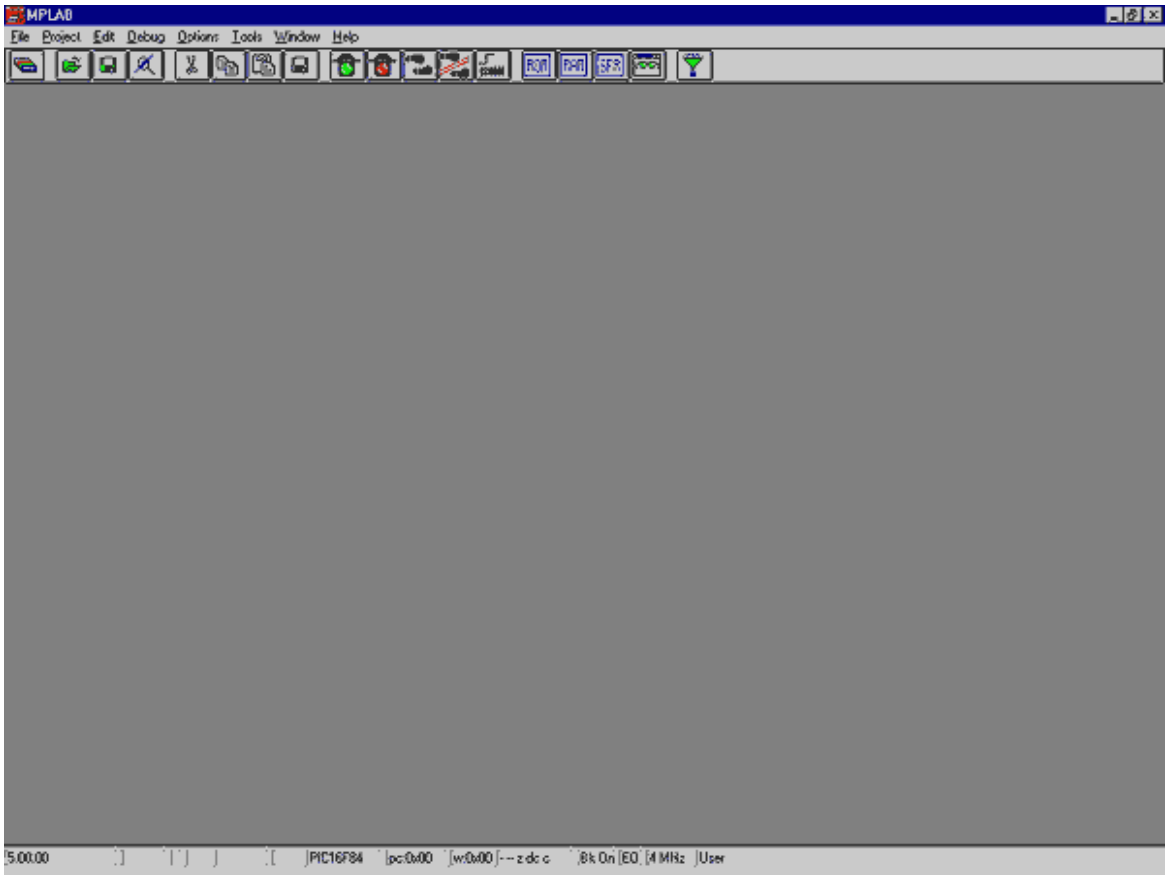
```
        bcf   STATUS, RP0       ;bank0, because EEADR is at 09h
        movlw 0x00              ;address of location being
                                ;written to
        movwf EEADR             ;address being transferred to
                                ;EEADR
        movlw 0xEE              ;write the value 0xEE
        movwf EEDATA            ;data goes to EEDATA register
        bsf STATUS, RP0         ;Bank1 because EEADR is at 09h
        bcf INTCON, GIE         ;all interrupts are disabled
        bsf EECON1, WREN        ;writing enabled
        movlw 55h
1)      movwf EECON2            ;first key 55h --> EECON2
2)      movlw AAh
3)      movwf EECON2            ;second key AAh --> EECON2
4)      bsf EECON1,WR           ;initializes writing
        bsf INTCON, GIE         ;interrupts are enabled
```

Este recomandat ca WREN s• fie închis tot timpul cu excep•ia scrierii datelor în EEPROM, a•a c• posibilitatea unei scrieri accidentale va fi minim•. Scrierea în EEPROM va fi automat •tears•!

Pagina anterioar• | **Con•inut** | **Pagina urm•toare**

5.2 Introducere în MPLAB

Urmând procedura de instalare, ve•i ob•ine un ecran al programului însu•i. Dup• cum vede•i, MPLAB arat• ca cele mai multe programa Windows. În apropierea zonei de lucru este un "menu" (în partea de sus colorat în albastru cu op•iunile File, Edit…etc.), "toolbar" (o zon• cu ilustra•ii de m•rimea unor p•trate mici), •i linia de stare în partea de jos a ferestrei. Este o regul• în Windows de a lua cele mai frecvent folosite op•iuni de programe •i de a le plasa mai jos de menu, de asemenea. Astfel le putem accesa mai u•or •i s• gr•bim lucrul. Cu alte cuvinte, ceea ce ave•i în în toolbar ave•i de asemenea în menu.



Ecranul dup• startarea MPLAB

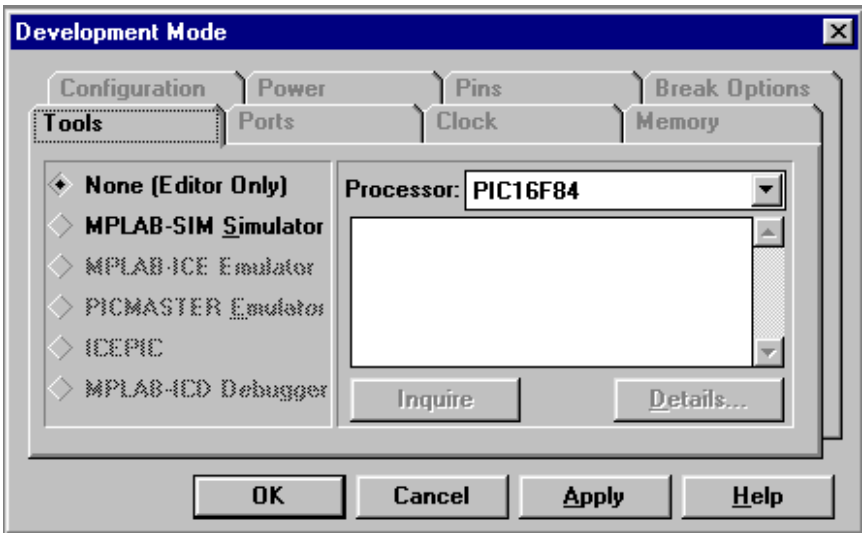
Scopul acestui capitol este ca s• deveni•i familiar cu mediul de dezvoltare MPLAB •i cu elementele de baz• ale MPLAB ca:

Alegerea modului de dezvoltare
Conceperea unui proiect
Conceperea unui fi•ier pentru programul original
Scrierea un program elementar în limbajul de programare asamblor
Translarea unui program în cod executiv
Startarea programului
Deschiderea unei noi ferestre pentru un simulator
Deschiderea unei noi ferestre pentru variabile a c•ror valori le urm•rim (Watch Window)
Salvarea unei fereastre cu variabile a c•ror valori le urm•rim
Setarea punctelor de întreupere într-un simulator (Break point)

Preg•tirea unui program de a fi citit într-un microcontroler se poate rezuma în câ•iva pa•i:

5.3 Alegerea modului de dezvoltare

Setarea unui mod dezvoltare este necesar• a•a ca MPLAB s• poat• •ti ce instrumente vor fi folosite pentru a executa programul scris. În cazul nostru, avem nevoie s• set•m simulatorul ca un instrument ce este folosit. F•când clic pe OPTIONS---> DEVELOPMENT MODE, o nou• fereastr• apare ca în imaginea de mai jos:



Setarea unui mod de dezvoltare

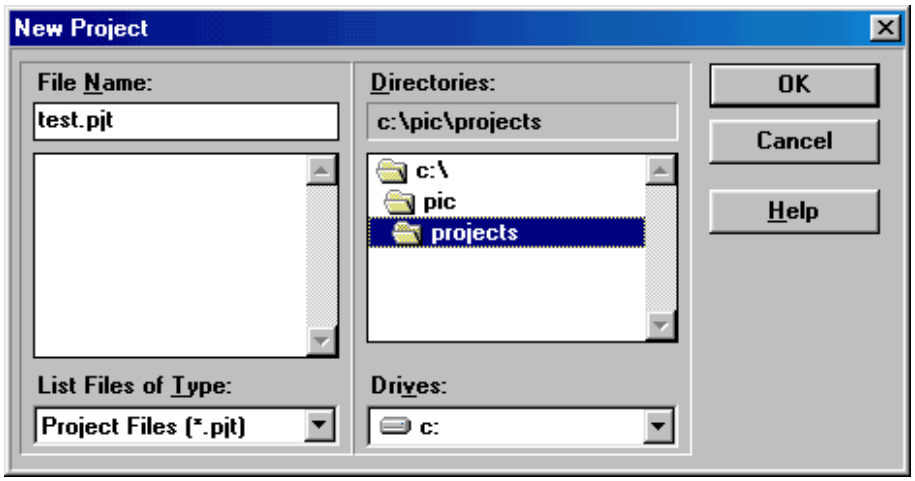
Trebuie s• select•m op•iunea 'MPLAB-SIM Simulator' pentru c• acolo se va testa programul. În afar• de aceast• op•iune, este de asemenea disponibil• op•iunea 'Editor Only'. Aceast• op•iune este folosit• doar dac• dorim s• scriem un program •i prin programator s• scriem ' hex file' într-un microcontoler. Selec•ia modelului microcontrolerului este f•cut• în partea dreapt•. Pentru c• aceast• carte este bazat• pe PIC16F84, trebuie selectat acest model.

De obicei când începem s• lucr•m cu microcontrolere, folosim un simulator. Dup• cum nivelul cunoa•terii va cre•te, programul se va scrie într-un microcontroler imediat dup• translare. Sfatul nostru este ca s• folosi•i totdeauna simulatorul. Chiar dac• programul va p•rea c• se dezvolt• lent, se va merita la sfâr•it.

5.4 Conceperea unui proiect

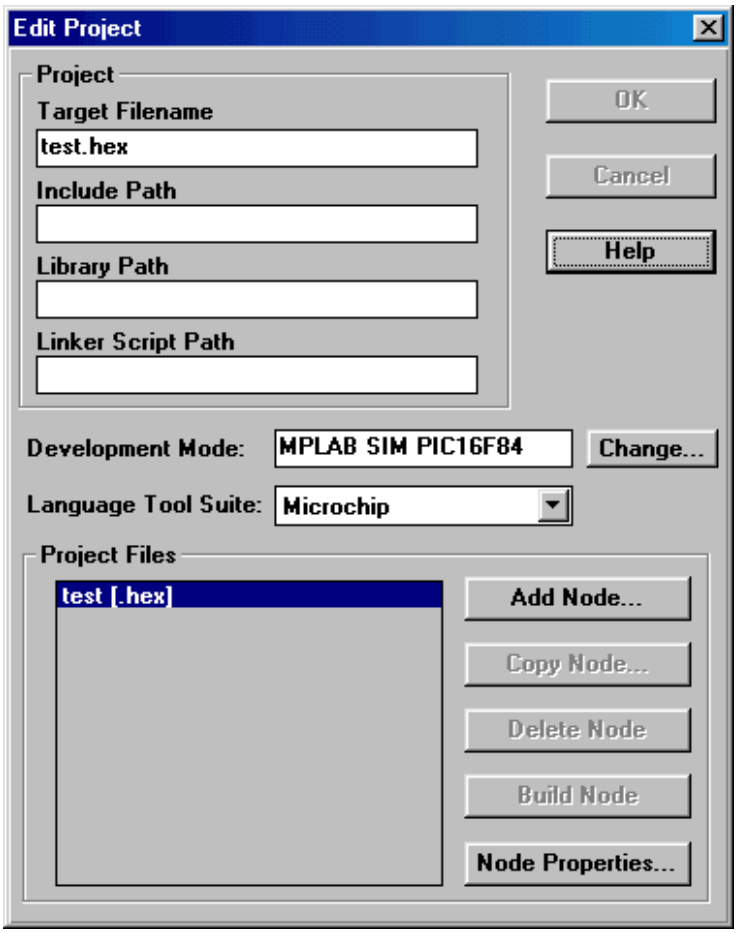
Pentru a începe s• scrie•i un program ave•i nevoie s• crea•i mai intâi un proiect. F•când clic pe PROJECT --> NEW PROJECT pute•i s• v• denumi•i proiectul •i s•-l memora•i într-un director pe care-l dori•i. În imaginea de mai jos, este creat un proiect numit 'test.pjt' •i memorat în directorul c:\PIC\PROJEKTS\.

Acest director este ales pentru c• autorii au ales acest director în calculatorul lor. În general, un director cu fi•iere este plasat de obicei într-un director mai mare a c•rui nume este asociat negre•it cu con•inutul lui.



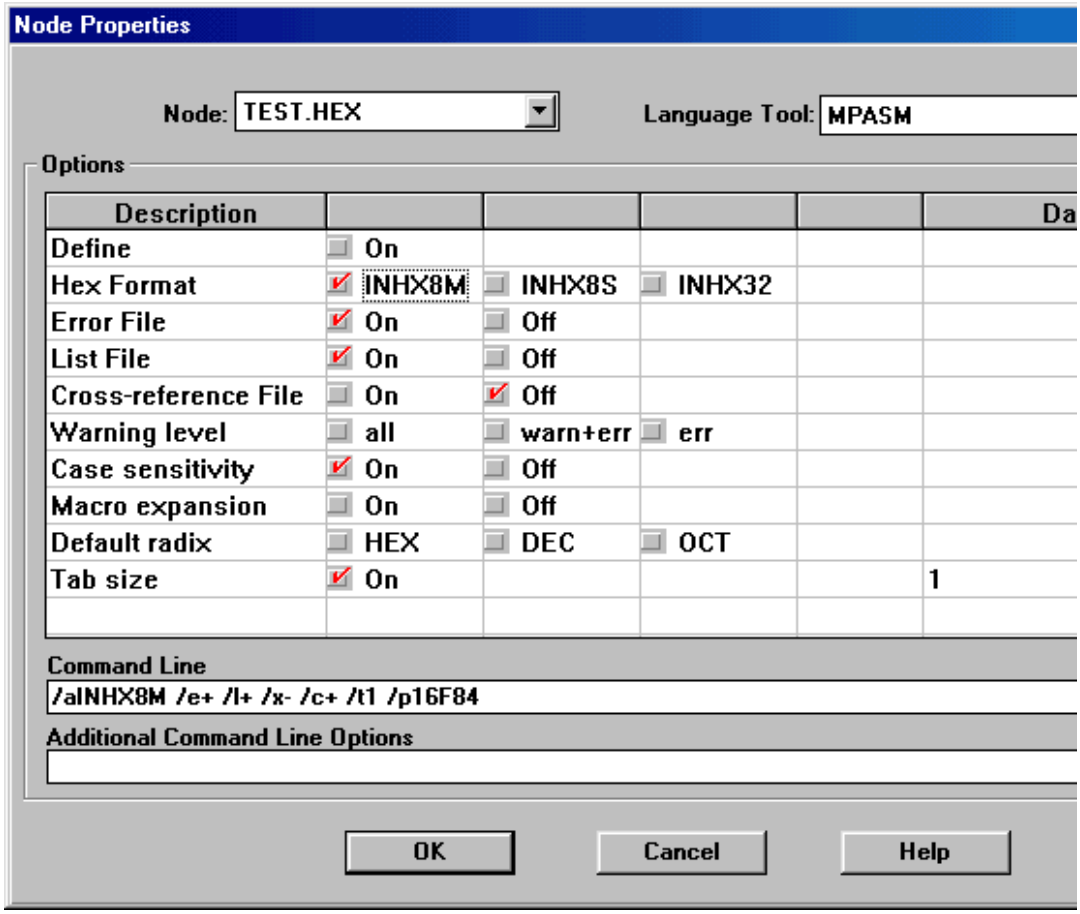
Deschiderea unui proiect nou

Dup• denumirea unui proiect, clic pe OK. O nou• fereastr• apare în imaginea um•toare.



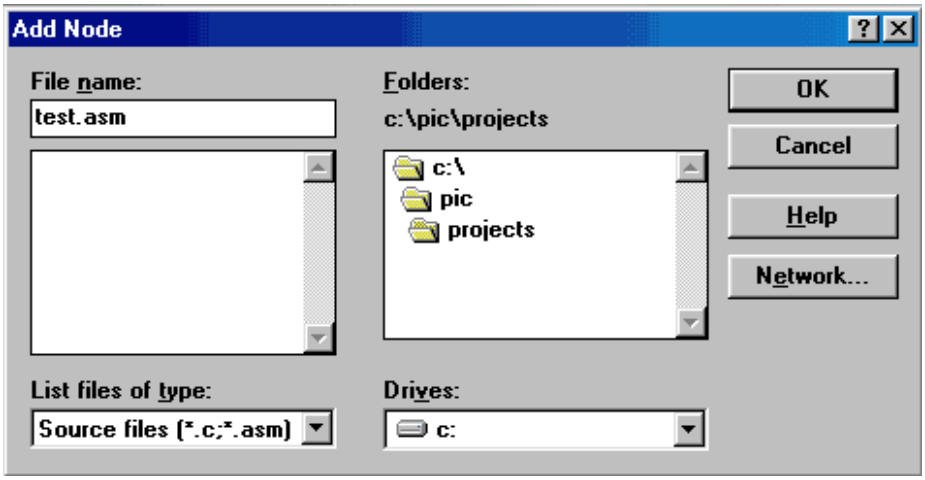
Ajustând elementele proiectului

F•când un clic pe "test [.hex]" se activeaz• op•iunea 'Node properties' în col•ul din dreapta jos a ferestrei. F•când clic pe ea ob•ine•i urm•toarea fereastr•.



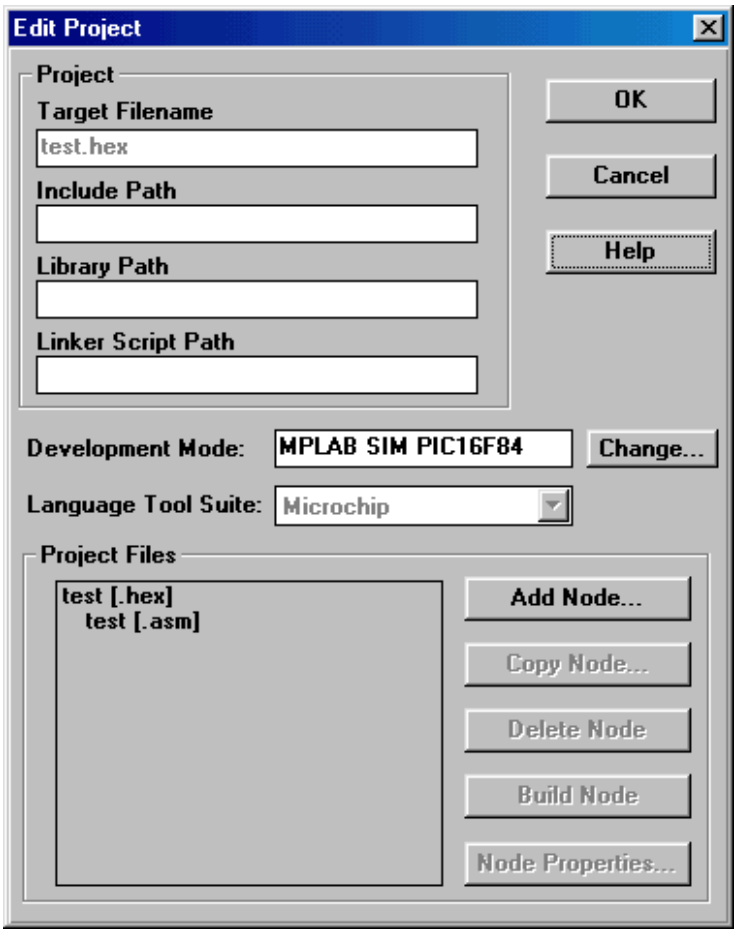
Definind parametrii asamblorului MPASM

Din aceast• imagine observ•m c• sunt diferi•i parametri. Fiecare fel corespunde la un parametru în "Command line". Pentru c• memorarea acestor parametri este foarte necomfortabil•, chiar interzis• pentru încep•tori, s-a introdus ajustarea grafic•. Din imagine observ•m ce op•iuni trebuie deschise. F•când clic pe OK ne întoarcem la fereastra anterioar• unde "Add node" este o op•iune activ•. F•când clic pe ea ob•inem urm•toarea fereastr• unde ne denumim programul asamblor. S•-l denumim"Test.asm" pentru c• acesta este primul nostru program în MPLAB.



Deschizând un proiect nou

F•când clic pe OK ne întoarcem la fereastra de început unde observ•m ad•ugat un fi•ier asamblor.



Fi•ier asamblor ad•ugat

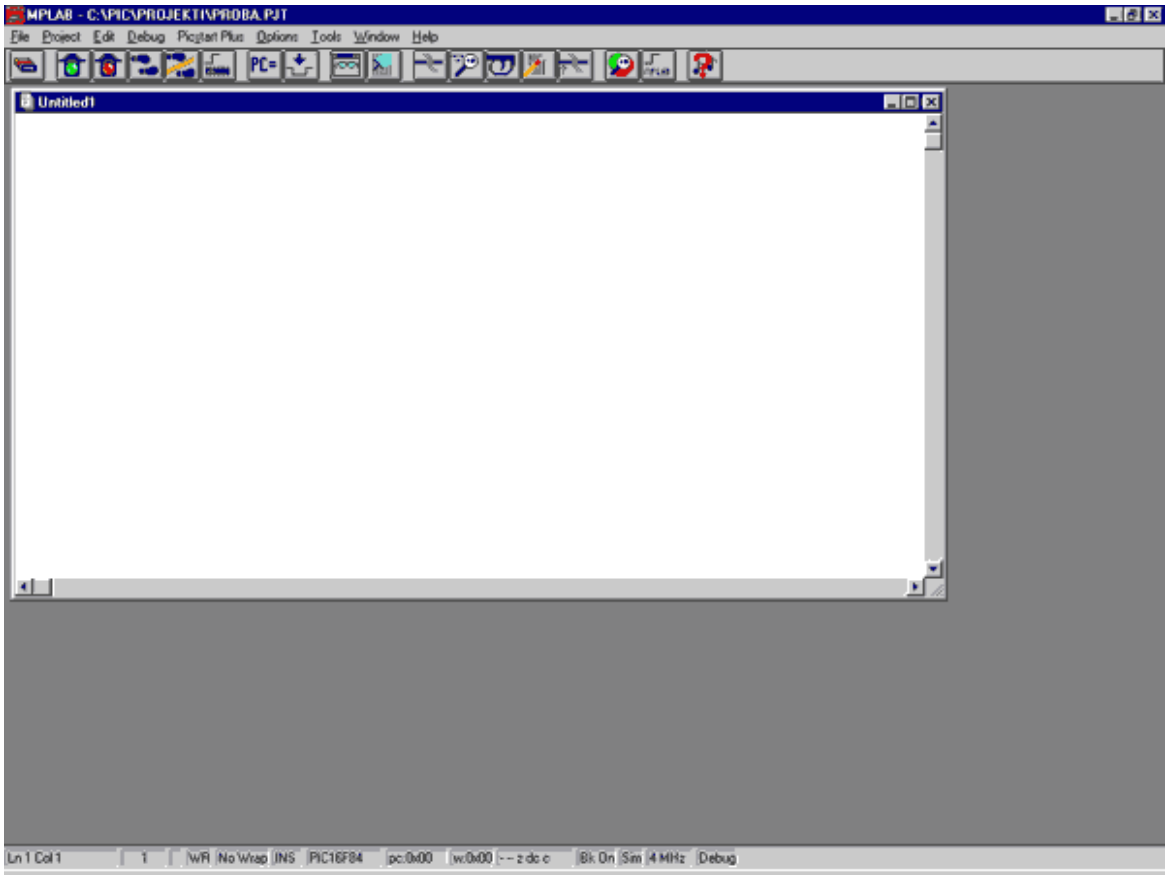
F•când clic pe OK ne întoarcem la mediul de dezvoltare MPLAB.

Pagina anterioar•	Con•inut	Pagina urm•toare

5.5 Conceperea unui nou fi•ier asamblor(scrierea un program nou)

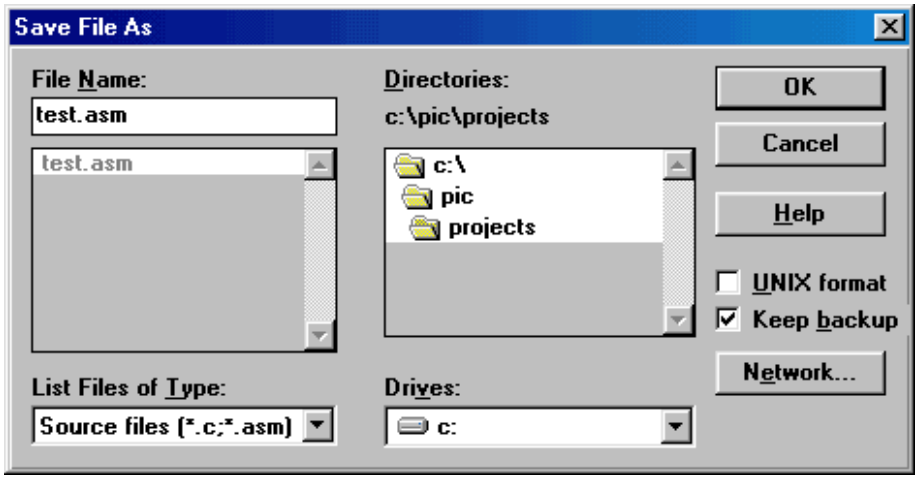
Când partea "proiect" a lucrului este terminat•, trebuie s• începem s• scriem un program. Cu alte cuvinte, un nou fi•ier trebuie deschis, •i se va denumi "test.asm". În cazul nostru, fi•ierul trebuie denumit "test.asm" pentru c• în proiecte ce au doar un fi•ier (ca al nostru), numele proiectului •i numele fi•ierului surs• trebuie s• fie acelea•i.

Un nou fi•ier este deschis f•când clic pe FILE>NEW. Astfel ob•inem o fereastr• text în interiorul spa•iului de lucru MPLAB.



Fi•ier nou asamblor deschis

Fereastra nou• reprezint• un fi•ier unde va fi scris programul. Pentru c• fi•ierul nostru trebuie denumit "test.asm", îl vom denumi a•a. Denumirea se face (ca la toate programele Windows) prin clic pe FILE>SAVE AS. Ob•inem apoi o fereastr• ca imaginea urm•toare.



Denumirea •i salvarea unui fi•ier asamblor nou

Când ob•inem aceast• fereastr•, trebuie s• scrierm'test.asm' mai jos de 'File name:', •i facem clic pe OK. Dup• aceea, vom observa numele fi•ierului 'test.asm' în partea de sus a ferestrei noastre.

Pagina anterioar•	Con•inut	Pagina urm•toare

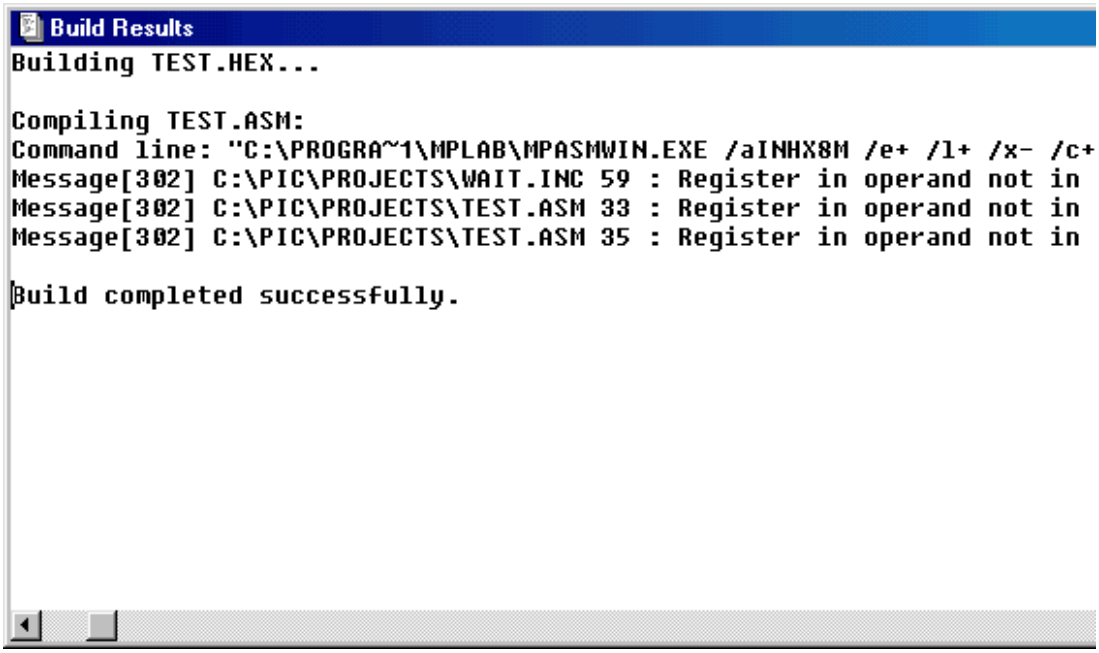
5.6 Scrierea unui program

Numai dup• ce toate opera•iile precedente au fost terminate suntem capabili s• începem s• scriem un program. Pentru c• un program simplu a fost deja scris în sec•iunea c•r•ii "Programare în Limbaj de Asamblare", vom folosi acela•i program aici, de asemenea.

Programul: Proba.asm

```
;Program pentru iniţializarea portului B şi setarea pinilor
;la starea de unu logic
;Versiunea 1.0  Data: 25.04.200.0   MCU:PIC16F84
;Scris de: Petar Petrovic

; Declaraţia şi configuraţia procesorului

      PROCESSOR 16F84
      #include         "p16f84.inc"    ; Titlul procesorului

      __CONFIG     _CP_OFF &_WDT_OFF &_PWRTE_ON &_XT_OSC

      org    0x00         ; Vector reset
      goto   Main         ; Du-te la începutul programului principal
      org    0x04         ; Vector întrerupere
      goto   Main         ; Rutină de întrerupere nu există

      #include    "bank.ink"         ; Macro-uri BANK0 şi BANK1

; Începutul programului principal

Main
      BANK1               ; Selectează bank-ul 1 de memorie
      movwlw 0x00
      movwf TRISB         ; Pinii portului B sunt ieşire
      BANK0               ; Selectează bank-ul de memorie 0

      movlw 0xFF
      movwf PORTB         ; Setează toţi unu la portul B

      Loop goto Loop      ; Programul rămâne în buclă

      end                 ; Marcare necesară la
                          ; sfârşitul programului
```

Programul trebuie s• fie scris într-o fereastr• care este deschis•, sau copiat• de pe un disc, sau luat din prezentarea Mikroelektronika Internet folosind op•iunile copy •i paste. Când programul este copiat în "test.asm" window, putem folosi comanda PROJECT -> BUILD ALL (dac• nu sunt erori), •i o nou• fereastr• va apare ca în imaginea urm•toare.



Fereastr• cu mesaje dup• translarea programului asamblor

Putem vedea din imagine c• ob•inem fi•ierul "test.hex" ca rezultat al procesului de translare, pentru care este folosit programul MPASMWIN, •i c• este doar un mesaj. În toate aceste informa•ii, ultima propozi•ie în fereastr• este cea mai important• pentru c• arat• dac• translarea a fost sau nu f•cut• cu succes. 'Build completed successfully' este un mesaj afirmând c• translarea a fost de succes •i c• nu sunt alt erori.

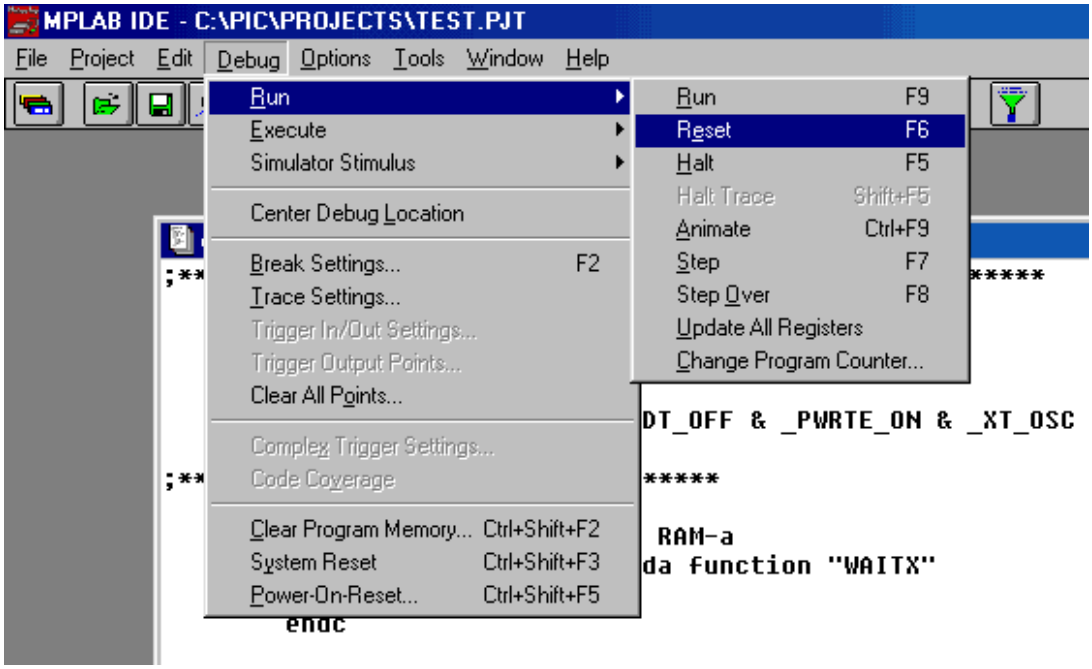
În caz c• apare o eroare, trebuie s• facem dublu clic pe mesajul eroare în fereastra 'Build Results'. Aceasta v• va transfera automat în programul asamblor •i în linia unde a fost eroarea.

Pagina anterioar•	Con•inut	Pagina urm•toare

5.7 Simulatorul MPSIM

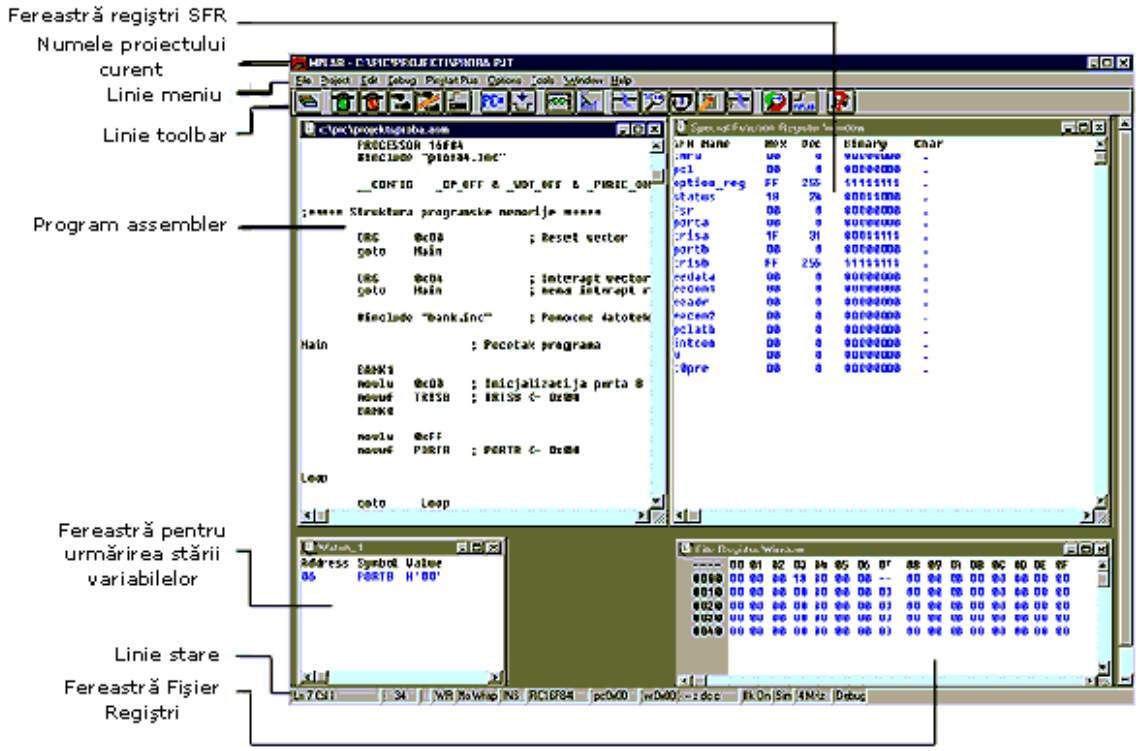
Simulatorul este o parte a mediului MPLAB care d• o mai bun• imagine a lucr•rilor unui microcontroler. Printr-un simulator, putem monitoriza valorile curente ale variabilelor, valorile registrului •i starea pinilor portului. Este adev•rat, simulatorul nu are aceea•i valoare în toate programele. Dac• un program este simplu (ca cel dat aici ca exemplu), simulrea nu este foarte important• pentru c• setarea pinilor portului B la unu logic nu este o sarcin• dificil•. Totu•i, simulatorul poate fi de mare de mare ajutor la programele mai complicate ce includ timer-i, condi•ii diferite unde ceva se întâmpl• , •i alte cerin•e similare (în special cu opera•ii matematice). Simularea, dup• cum indic• numele " simuleaz• lucrul unui microcontroler". În timp ce simulatorul este conceput ca microcontrolerul s• execute instruc•iunile una câte una, programatorul se mi•c• într-un program pas-cu-pas (linie-cu-linie) •i urm•re•te ce se întâmpl• cu datele în microcontroler. Când scrierea s-a terminat, este un obicei bun ca programatorul s•-•i verifice mai întâi programul s•u în simulator, •i apoi s•-l ruleze într-o situa•ie real•. Din nefericire, a• cum se întâmpl• cu multe alte obiceiuri bune, acesta este mai pu•in sau mai mult luat în seam•. Motivele pentru aceasta sunt în parte personalitatea, •i în parte lipsa unor simulatoare bune.

Primul lucru pe care trebuie s•-l facem este, ca într-o situa•ie real•, este de a reseta un microcontroler cu comanda DEBUG > RUN > RESET. Aceast• comand• rezult• în linia îngro•at• pozi•ionat• la începutul unui program, •i contorul programului este pozi•ionat la zero ceea ce poate fi observat în linia de stare (pc: 0x00).



Începerea simul•rii programului, resetarea microcontrolerului

Una din principalele caracteristici a simulatorului este abilitatea de a vedea starea regi•trilor din microcontroler. Ace•ti regi•tri sunt numi•i regi•tri de func•ie special•, sau SFR. Putem ob•ine o fereastr• cu regi•tri SFR f•când clic pe WINDOW->SPECIAL FUNCTION REGISTERS, sau pe icon-ul SFR. Înafar• de regi•trii SFR, este util de a avea o avea o privire în interiorul fi•ierului regi•trilor. Fereastra cu fi•ierul regi•trilor poate fi deschis• f•când clic pe WINDOW->FILE REGISTERS. Dac• sunt variabile în program, este bine de a le vedea de asemenea. Fiec•rei variabile îi este desemnat• o fereastr• (Watch Windows) f•când clic pe WINDOW->WATCH WINDOWS.



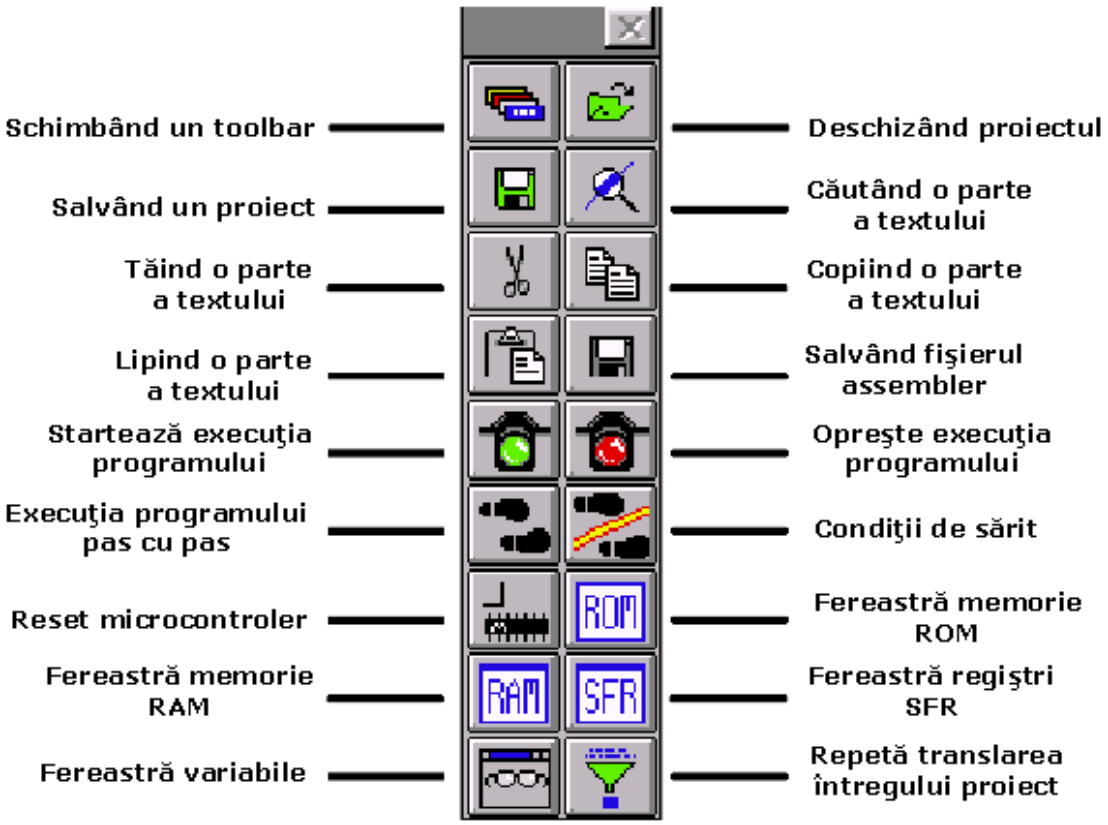
Simulator cu ferestre deschise pentru regi•tri SFR, fi•ierul regi•trilor •i variabile.

Comanda urm•toare într-un simulator este DEBUG>RUN>STEP care începe pa•ii no•tri prin program. Aceea•i comand• ar fi putut fi desemnat• de la o tastatur• cu tasta <F7> (în general, toate comenzile importante au taste desemnate de le claviatur•). Folosind tasta F7, programul este executat pas cu pas. Când ob•inem un macro, fi•ierul con•inând un macro este deschis (Bank.inc), •i continu•m cu macro. Într-o fereastr• cu regi•tri SFR putem observa cum registrul W prime•te valoarea 0xFF •i pe care o trimite la portul B. F•când clic pe tasta F7 din nou, nu ob•inem nimic pentru c• programul a ajuns într-o "infinite loop"-bucl• infinit•. Bucla infinit• este un termen pe care îl întâlnim adesea. Reprezint• bucla din care un microcontroler nu poate ie•i pân• nu se întâmpl• întreruperea (dac• este folosit• într-un program), sau p•n• ce micorcontrolerul va fi resetat.

Pagina anterioar•	Con•inut	Pagina urm•toare

5.8 Toolbar

Pentru c• MPLAB are mai mult de o component•, fiecare component• are bara sa de instrumente, toolbar-ul s•u. Totu•i, este un toolbar care este un fel de compila•ie a tuturor toolbar-ilor, •i poate servi ca un toolbar folosit în mod uzual. Acest toolbar este de ajuns pentru nevoile noastre, •i va fi descris în detaliu. În figura de mai jos putem vedea un toolbar pentru care avem nevoie de o scurt• explica•ie pentru fiecare icon. Din cauza formatului limitat a cestei c•r•i, acest toolbar este reprezentat ca un toolbar suspendat. În general, este plasat orizontal mai jos de menu, de-a lungul întregului ecran.



Toolbar universal cu scurte explica•ii ale icon –urilor

Descriere a icon-urilor toolbar-ului

	Dac• toolbar-ul curent nu r•spunde datorit• diferitor motive la un clic pe acest icon, apare urm•torul. Schimbarea total• este repetat• a•a încât la al patrulea clic vom ob•ine acela•i toolbar.
	Icon pentru deschiderea unui proiect. Proiectul deschis în acest fel con•ine toate ajust•rile ecranului •i ajustarea tuturor elementelor care sunt cruciale pentru proiectul curent.
	Icon pentru salvarea unui proiect. Proiectul salvat va p•stra toate ajust•rile ferestrei •i toate ajust•rile parametrilor. Când citim un program din nou, totul se va întoarce pe ecran ca atunci când s-a închis proiectul.
	C•utarea unei p•r•i de program, sau cuvinte este opera•ia de care avem nevoie când c•ut•m printr-un asamblor mare sau alte programe. Folosindu-l, putem g•si repede o parte a programului, label, macro, etc.
	T•ind o parte a textului. Acesta •i urm•toarele trei icon-uri sunt standard în toate programele care au de a face cu procesarea fi•ierelor textuale. Pentru c• fiecare program este de fapt un fi•ier text obi•nuit, aceste opera•ii sunt folositoare.
	Copiind o parte a textului. Este o diferen•• între acesta •i iconul precedent.Cu opera•ia de t•iere, când t•ia•i o parte a textului, dispare din ecran (•i din program) •i este copiat dup• aceea. Dar cu opera•ia copy, textul este copiat •i nu t•iat, •i r•mâne pe ecran.

	Când o parte a textului este copiat•, este mutat• într-o parte a memoriei ce serve•te pentru transferarea datelor în sistemul opera•ional Windows. Mai târziu, f•când clic pe acest icon poate fi lipit-'pasted' în textul unde este cursorul.
	Salvând un program (fi•ier asamblor).
	Starteaz• execu•ia programului la vitez• maxim•.Se recunoa•te prin apari•ia unei linii de stare galbene. Cu acest fel de execu•ie de program, simulatorul execut• un program la vitez• maxim• pân• ce este întrerupt de un clic pe iconul cu lumin• ro•ie de trafic.
	Opre•te execu•ia programului la vitez• maxim•. Dup• clic pe acest icon, linia de stare devine gri din nou, •i execu•ia programului poate continua pas cu pas.
	Pas cu pas execu•ia programului. F•când clic pe acest icon, începem executarea unei instruc•iuni din linia urm•toare în leg•tur• cu cea curent•.
	Cerere de a s•ri-skip. Pentru c• simulatorul este totu•i o simulare de software de lucru real, este posibil de a s•ri pur •i simplu peste unele cereri ale programului. Aceasta este în special la îndemân• cu instruc•iuni ce a•teapt• o anumit• cerere dup• care programul poate s• continue. Acea parte a programului ce urmeaz• unei cerei este partea ce este interesant• pentru un programator.
	Resetând un microcontroler. F•când clic pe acest icon, contorul programului este pozi•ionat la începutul programului •i simularea poate începe.
	F•când clic pe acest icon ob•inem o fereastr• cu un program, dar de aceast• dat• ca memorie de program unde putem veea ce instruc•iune este g•sit• •i la ce adres•.
	Cu ajutorul acestui icon ob•inem o fereastr• cu con•inutul memoriei RAM a microcontrolerului.
	F•când clic pe acest icon, apare fereastra cu registrul SFR. Pentru c• regi•trii SFR sunt folosi•i în fiecare program, este recomandat ca în simulator aceast• fereastr• s• fie totdeauna activ•.
	Dac• un program con•ine variabile ale c•ror valoare trebuie s• le urm•rim (ex. contorul), o fereastr• are nevoie s• fie ad•gat• pentru fiecare din ele, ceea ce se face prin folosirea acestui icon.
	Când unele erori într-un program sunt eviden•iate în timpul procesului de simulare, programul trebuie corectat. Pentru c• simulatorul folose•te fi•ier HEX ca intrare a sa, trebuie s• transl•m un program din nou a•a ca toate schimb•rile s• fie transferate într-un simulator. F•când clic pe acest icon, întregul proiect este translat din nou, •i ob•inem versiunea mai nou• a fi•ierului HEX pentru simulator.

Pagina anterioar• **Con•inut** **Pagina urm•toare**

Macrouri folosite în programe

Exemplele din sec•iunile urm•toare ale acestui capitol utilizeaz• deseori WAIT, WAITx •i PRINT, de aceea ele vor fi explicate în detaliu.

Macrourile WAIT, WAITx

Fi•ierul Wait.inc con•ine dou• macrouri: WAIT •i WAITx. Prin intermediul acestor macrouri este posibil s• repartiz•m întârzieri de timp în intervale variate. Amândou• macrouri folosesc dep••irea contorului TMR0 ca un interval de timp de baz•. Prin schimbarea prescaler-ului putem schimba lungimea intervalului dep••irii contorului TMR0.

WAIT.inc

```
;***** Declaring constants *****

      CONSTANT PRESCstd = b'00000001' ; Standard prescaler value for TMR0

;***** Macros *****

WAIT   macro  timeconst_1

      movlw  timeconst_1
      call   WAITstd
      endm

WAITX  macro  timeconst_2, PRESCext

      movlw  timeconst_2
      mowwf  WCYCLE                 ; Set the delay time period
      movlw  PRESCext               ; Write specific prescaler value
      call   WAIT_x
      endm

;***** Subprograms *****

WAITstd
      mowwf  WCYCLE                 ; Set the delay time period
      movlw  PRESCstd               ; Write specific prescaler value
WAIT_x
      clrf   TMR0
      BANK1
      mowwf   OPTION_REG            ; Assing the prescaler to TMR0 timer
      BANK0

WAITa
     bcf     INTCON,T0IF        ; Erase TMR0 Overflow Flag
WAITb
      btfss  INTCON,T0IF        ; Check whether it is erased, skip if it isn`t
      goto    WAITb             ; Wait loop
      decfsz  WCYCLE,1          ; Repeat the loop if delay period has not run out
      goto    WAITa
      RETURN
```

Dac• folosim un oscilator (rezonator) de 4MHz, pentru valorile prescaler-ului 0,1 •i 7 care divid ceasul de baz• al oscilatorului, intervalul urmat de o dep••ire a contorului TMR0 va fi 0.512, 1.02 •i 65.3ms. Practic, aceasta înseamn• c• cea mai mare întârziere va fi 256x65.3ms care este egal• cu 16.72 secunde.

Prescalerul	Divizor	Depăşire
b'00000000'	1:2	0.512 ms
b'00000001'	1:4	1.02 ms
b'00000111'	1:256	65.3 ms

Pentru a utiliza macrouri în programul principal este necesar s• declar•m variabilele wcycle •i prescWAIT dup• cum vom vedea în exemplele ce vor urma acestui capitol. Macroul WAIT are un singur argument. Valoarea standard atribuit• prescaler-ului acestui macro este 1 (1.02ms), •i nu poate fi schimbat•.

WAIT timeconst_1

timecons-t_1 este un num•r de la 0 la 255. Prin multiplicarea acestui num•r cu perioada de timp de dep••ire (overflow) vom ob•ine durata total• a întârzierii: TIME = timeconst_1 x 1.02ms.

Exemplu: WAIT .100

Exemplul arat• cum s• ob•inem o întârziere de 100x1.02ms, sau durata total• de 102ms.

Spre deosebire de macroul WAIT, macroul WAITX mai are un argument care poate atribui o valoare prescaler-ului. Macroul WAITX are dou• argumente:

timeconst_2 este un num•r de la 0 la 255. Prin multiplicarea acestui num•r cu perioada de timp de dep••ire (overflow) vom ob•ine durata total• a întârzierii: TIME = timeconst_1 x 1.02ms x PRESCext.

PRESCext este un num•r de la 0 la 7 care seteaz• rela•ia dintre tact •i timer-ul TMR0.

Exemplu: WAITX .100,7

Exemplul arat• cum s• ob•inem o întârziere de 100x65.3ms, sau durata total• de 653ms.

Macroul PRINT

Macroul PRINT este localizat în fi•ierul Print.inc. El u•ureaz• lucrul pentru trimiterea unui •ir de date la unul dintre dispozitivele de ie•ire, cum ar fi: LCD, RS232, imprimant• matricial•...etc. Cea mai u•oar• cale pentru a forma o **serie** este prin folosirea unei directive dt (define table). Aceast• instruc•iune memoreaz• o **serie** de date în cadrul memoriei programului ca un grup de instruc•iuni retlw al c•rui operand este data din •ir.

PRINT.INC

```
PRINT macro Addr, Start, End, Var, Out

        Local   Next    ; Local label
        Local   Exit

        movlw   Start   ; Address of the first member in the string
        mowwf   Var
Next
        movf    Var,w   ; Pointer -> W reg.
        call    Addr    ; String(W reg.) -> W reg.
                        ; Data is in W reg.
        Out             ; Display data on the output device
        movf    Var,w
        xorlw   End-1
        btfsc   STATUS,Z  ; Is it the end of the string ?
        goto    Exit      ; If it is, get out of the loop
        incf    Var,f     ; If isn`t, set the Pointer to next member
                          ; of the string
        goto    Next      ; Repeat loop
Exit
        endm
```

Modalitatea prin care o astfel de secven•• este format• folosind instruc•iunea dt este ar•tat• în urm•torul exemplu:

```
org 0x00
goto Main

        String movwf PCL
        String1 dt "acesta este un sir 'ASCII"
        String2 dt "al doilea sir"
        End
        Main

movlw .5
call String
:
```

Prima instruc•iune dup• eticheta Main scrie pozi•ia unui membru al •irului în registrul W. Execut•m un salt cu instruc•iunea call la eticheta •irului unde pozi•ia membrului •irului este adunat• la valoarea PC (Program Counter): PCL = PCL + W. În continuare avem în program counter o adres• a instruc•iunii retlw cu membrul dorit al •irului. În momentul în care aceast• instruc•iune este executat•, membrul •irului va fi în registrul W, •i adresa instruc•iunii care va fi executat• dup• instruc•iunea call va fi în program counter. Eticheta end este o metod• elegant• de a marca adresa la care •irul se termin•.

Macroul PRINT are cinci argumente:

PRINT macro Addr, Start, End, Var, Out

Addr este o adres• unde unul sau mai multe •iruri (situate unul dup• altul) încep.
Start este o adres• a primului membru al •irului.
End este o adres• unde •irul se termin•.
Var este variabila care are rolul de a ar•ta (pointa) membrii •irului.
Out este un argument pe care îl folosim pentru a trimite adresa rutinelor existente atribuite dispozitivelor de ie•ire cum ar fi: LCD, RS-232, etc.

Exemplu:

```
        org     0x00
        goto    Main

Series  movwf PCL
Message dt "mikroElektronika"
        End

Main
        PRINT Series, Message, End, Pointer, LCDw
              :
```

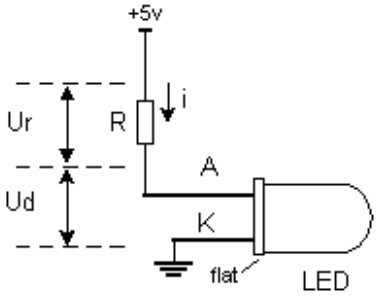
Macroul PRINT scrie la ie•ire un •ir „mikroElektronika" format din caractere ASCII la un dispozitiv de afi•are LCD. •irul **takes one part of program memory** începând cu adresa 0x03.

Example

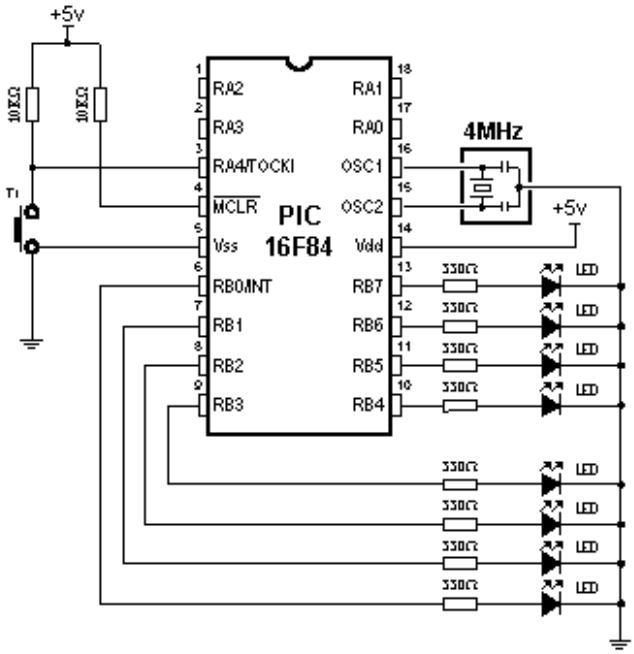
Light Emitting Diodes –LEDuri

Ledurile sunt unele dintre cele mai folosite elemente în electronic•. LED este o abreviere pentru „Light Emitting Diode”. În momentul în care alegem un led, sunt mai mul•i parametri de care trebuie s• •inem seama: diametrul, care este deobicei 3 sau 5mm (milimetri), curentul de func•ionare care este în jur de 10mA (poate fi mai mic decât 2mA pentru ledurile cu randament maxim: emisie de lumin• puternic•) •i bineîn•eles culoarea, care poate fi ro•ie sau verde de•i mai sunt leduri portocalii, albastre, galbene... . Ledurile trebuie conectate corect pentru a emite lumin• •i rezisten•a care limiteaz• curentul trebuie s• fie de o valoare corect• pentru ca ledul s• nu se ard• (supraînc•lzire). Tensiunea pozitiv• de alimentare este legat• la ANOD, iar catodul este legat la tensiunea negativ• sau la masa circuitului. Pentru a identifica fiecare pin, catodul este cel mai scurt pin iar corpul are în general o te•itur• pe partea catodului. Diodele vor emite lumin• numai dac• curentul circul• de la ANOD spre CATOD. Altfel jonctiunea PN este polarizat• invers •i curentul nu va circula. Pentru a conecta corect un led trebuie ad•ugat• o rezisten•• în serie pentru a limita de curentul prin diod•, pentru ca aceasta s• nu se ard•. Valoarea rezisten•ei este determinat• de curentul care vre•i s• circule prin led. Curentul maxim care poate **curge** printr-un led a fost stabilit de produc•tor. Ledurile cu randament maxim pot produce rezultate bune cu un curent mai mic de de 2mA.

Pentru a determina valoarea rezisten•ei serie, trebuie s• cunoa•tem valoarea tensiunii de alimentare. De aici sc•dem tensiunea care cade pe led. Aceast• valoare va varia de la 1,2v la 1,6v, depinzând de culoarea ledului. R•spunsul este valoarea lui Ur. Folosind aceast• valoare •i curentul care vrem s• circule prin LED (între 0.002A •i 0.01A) putem s• afl•m valoarea rezisten•ei cu ajutorul formulei: **$R=U_R / I$**.

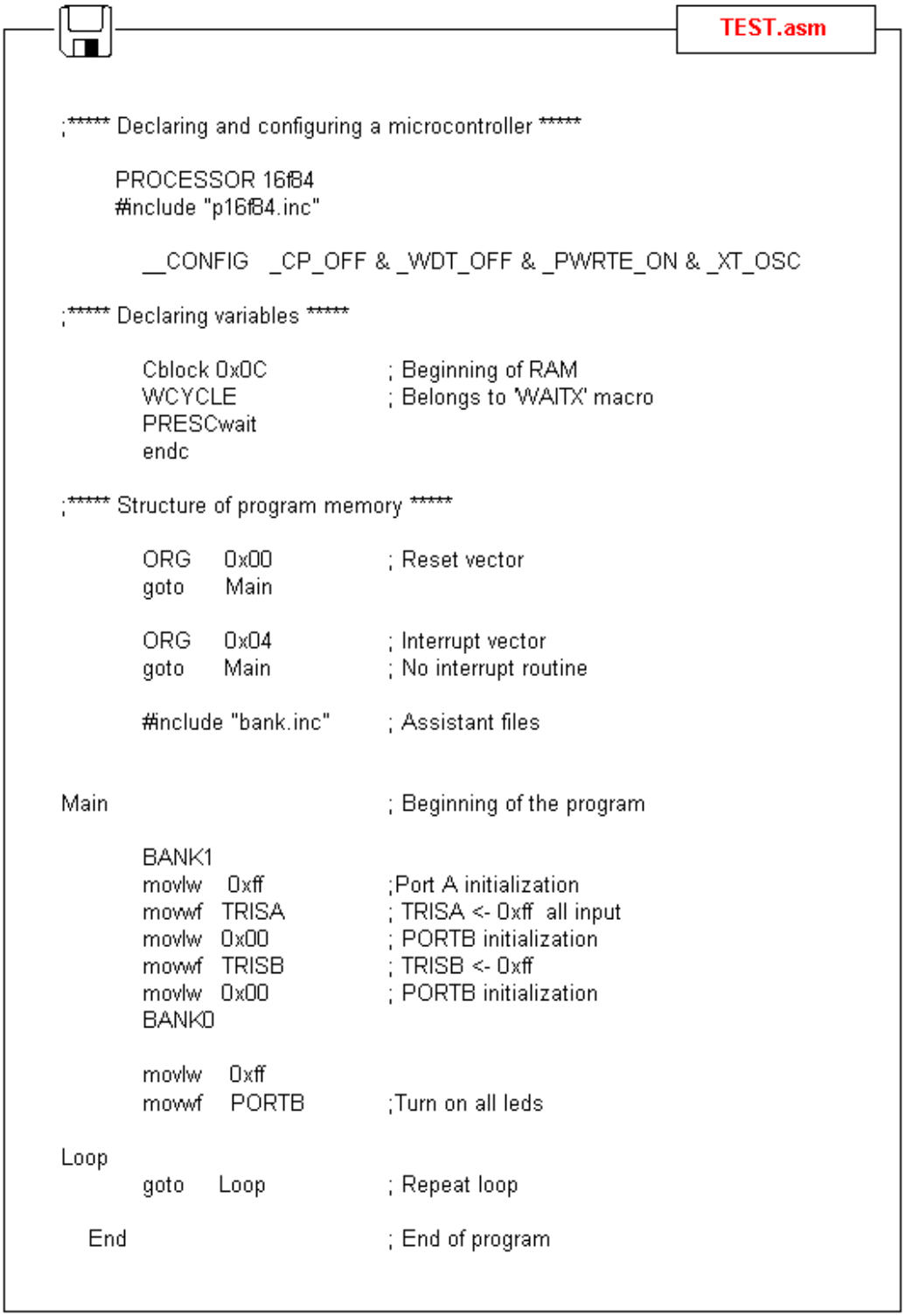


Ledurile sunt conectate la microcontroler în dou• metode. Una este s• le activ•m cu zero logic •i a doua este s• le activ•m cu unu logic. Prima metod• este numit• logic• NEGATIV• iar cea de-a doua este numit• logic• POZITIV•. Figura de mai sus ilustreaz• modalitatea de conectare prin logic• POZITIV•. Deoarece logica POZITIV• ofer• o tensiune de +5v diodei •i rezisten•ei serie, ledul va emite lumin• de fiecare dat• când un pin al portului B este în starea 1 logic (1 = ie•ire HIGH). Logica NEGATIV• necesit• ca ledul s• fie întors •i terminalele de tip anod s• fie conectate împreun• la borna pozitiv• a sursei. În momentul în care este livrat• o ie•ire LOW de la microcontroler c•tre anod •i rezisten••, ledul va lumina.



Connecting LED diodes to PORTB microcontroller

Exemplul urm•tor initializeaz• portul B ca port de ie•ire •i seteaz• unu logic pe fiecare pin al portului B pentru a activa toate ledurile.



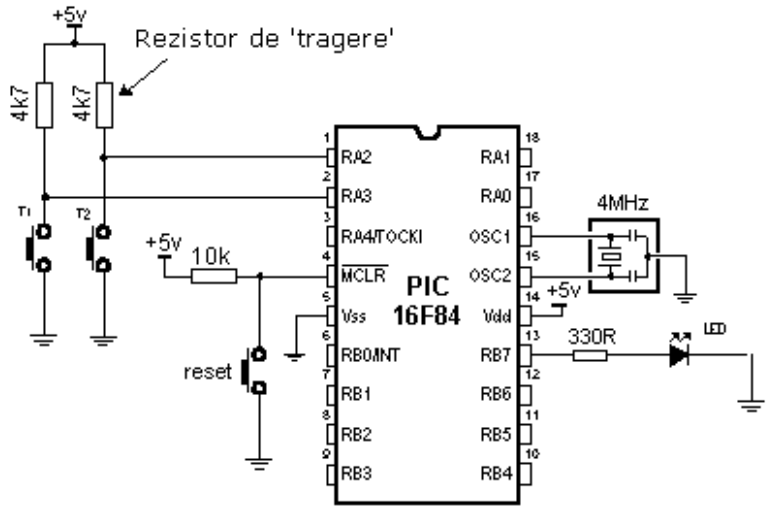
TEST.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        endc

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    Main            ; No interrupt routine

        #include "bank.inc"     ; Assistant files


Main                            ; Beginning of the program

        BANK1
        movlw   0xff            ;Port A initialization
        mowwf   TRISA           ; TRISA <- 0xff  all input
        movlw   0x00            ; PORTB initialization
        mowwf   TRISB           ; TRISB <- 0xff
        movlw   0x00            ; PORTB initialization
        BANK0

        movlw   0xff
        mowwf   PORTB           ;Turn on all leds

Loop
        goto    Loop            ; Repeat loop

   End                          ; End of program
```

Pagina anterioar• | Con•inut | Pagina urm•toare

Tastatura

Tastaturile sunt dispozitive mecanice utilizate pentru a executa o întrerupere sau pentru a realiza o conexiune între dou• puncte. Ele au diferite m•rimi •i au diferite scopuri. Tastele care sunt utilizate aici sunt denumite „taste dip”. Ele sunt lipite direct pe o plac• de circuit •i sunt deseori întâlnite în electronic•. Au patru pini (doi pentru fiecare contact), ceea ce le ofer• stabilitate mecanic•.



Exemplu pentru conectarea tastelor la pinii microcontrolerului

Func•ia tastei este simpl•. În momentul în care ap•s•m o tast•, dou• contacte sunt unite •i se realizeaz• o conexiune. Totu•i, nu toate lucrurile sunt simple. Problema const• în natura tensiunii ca valoare, •i în imperfec•iunea contactelor mecanice. Înainte ca un contact s• fie realizat sau decuplat, exist• o perioad• scurt• de timp când pot ap•rea vibra•ii (oscila•ii) ca rezultat al imperfec•iunii contactelor mecanice, sau din cauza vitezei diferite de ap•sare (acest lucru depinde de persoana care apas• tasta). Termenul atribuit acestui fenomen este denumit **switch (contact) debounce**. Dac• acest lucru nu este prev•zut în momentul în care un program este conceput, poate ap•rea o eroare sau programul poate produce mai mult decât un singur impuls la ie•ire pentru o singur• ap•sare de tast•. Pentru a evita acest lucru, putem introduce o mic• întârziere când detect•m închiderea unui contact. Aceasta va asigura faptul c• ap•sarea unei taste este interpretat• ca un singur impuls. Întârzierea de **debounce** este produs• în software •i durata întârzierii depinde de buton •i de scopul butonului. Problema poate fi par•ial rezolvat• prin ad•ugarea unui condensator în paralel la tast•, dar un program bine realizat ofer• rezultate mai bune. Programul poate fi ajustat pân• când detec•ia fals• este complet eliminat•. În anumite cazuri o simpl• întârziere poate fi suficient• dar dac• vre•i ca programul s• se ocupe de mai multe lucruri în acela•i timp, o simpl• întârziere va însemna c• procesorul nu va face nimic pe o lung• perioad• de timp •i poate rata alte intr•ri sau poate decupla portul de ie•ire c•tre un afi•or. Solu•ia este s• avem un program care s• urm•reasc• ap•sarea unei taste cât •i decuplarea unei taste. Macroul de mai jos poate fi folosit pentru **keypress debounce**.

BUTTON.inc

```
Button  macro HiLo, Port, Bit, Delay, Adrress

        Local   Exit    ; Local labels
        Local   Loop

        if HiLo == 0    ; Is the key pressed ?
        btfsc Port,Bit  ; Is input line LOW ?
        else
        btfss Port,Bit  ; Is input line HIGH ?
        endif
        goto Exit       ; If key hasn`t been pressed, exit the macro

        WAIT Delay      ; Delay for key debounce
Loop
        if HiLo == 0
        btfss Port,Bit  ; Is the key released ?
        else
        btfsc Port,Bit
        endif
        goto Loop

        WAIT Delay      ; Delay for key debounce

        call Adrress    ; Call the service subprogram
Exit                    ; Exit the macro
        endm            ; End of macro
```

Macroul precedent are mai multe argumente care trebuiesc explicate:

BUTTON macro HiLo, Port, Bit, Delay, Address

HiLo poate fi ’0’ sau ’1’ care reprezint• frontul cresc•tor sau c•z•tor unde subrutinele pot fi executate în momentul în care ap•sa•i o tast•.
Port este un port al microcontrolerului la care trebuie conectat• tasta. În cazul microcontrolerului PIC16F84, el poate fi PORT A sau PORT B.
Bit este un pin al portului la care tasta este conectat•.
Delay este un num•r de la 0 la 255, folosit pentru a atribui timpul necesar pentru a detecta **key debounce – contact oscillation – to stop**. El este calculat astfel: TIME = Delay x 1ms.
Adress este adresa la care microcontrolerul se duce dup• ce este detectat un eveniment provenit de la tastatur•. Subrutina de la aceast• adres• execut• instruc•iunile necesare pentru ap•sarea unei taste.

Exemplu 1 BUTTON 0, PORTA, 3, .100, Tester1_above

Tasta-1 este conectat• la RA0 (prima ie•ire a portului A) cu o întârziere de 100 milisecunde •i cu o reac•ie la zero logic. Subrutina care proceseaz• tasta este localizat• la adresa etichetei Tester1_above.

Exemplu 2 BUTTON 1, PORTA, 2, .200, Tester1_below

Tasta-2 este conectat• la RA1 (a doua ie•ire a portului A) cu 200ms întârziere •i cu reac•ie la unu logic.

Exemplul urm•tor arat• modul de folosire într-un program. BUTTON.ASM aprinde •i stinge LEDul. LEDul este conectat la cea de-a •aptea ie•ire a portului B. Tasta-1 este folosit• pentru a aprinde LEDul. Tasta-2 stinge LEDul.

BUTTON.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        endc

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    Main            ; No interrupt routine

        #include "bank.inc"     ; Assistant files
        #include "button.inc"
        #include "wait.inc"

Main                            ; Beginning of the program
        BANK1
        movlw   0xff            ; PORTA initialization
        mowwf   TRISA           ; TRISA <- 0xff
        movlw   0x00            ; PORTB initialization
        mowwf   TRISB           ; TRISB <- 0x00
        BANK0

        clrf    PORTB           ; PORTB <- 0x00

Loop
        Button 0, PORTA, 2, .100, On    ; Button 1
        Button 0, PORTA, 3, .100, Off   ; Button 2
        goto    Loop

On
        bsf     PORTB,7         ; Turn on LED
        return

Off
        bcf     PORTB,7         ; Turn off LED
        return

   End          ; End of program
```

Pagina anterioar• Con•inut Pagina urm•toare

Optocuplor

Optocuplorul combin• un LED •i un fototranzistor în aceea•i capsul•. Rolul unui optocuplor este acela de a separa dou• p•r•i de circuit.

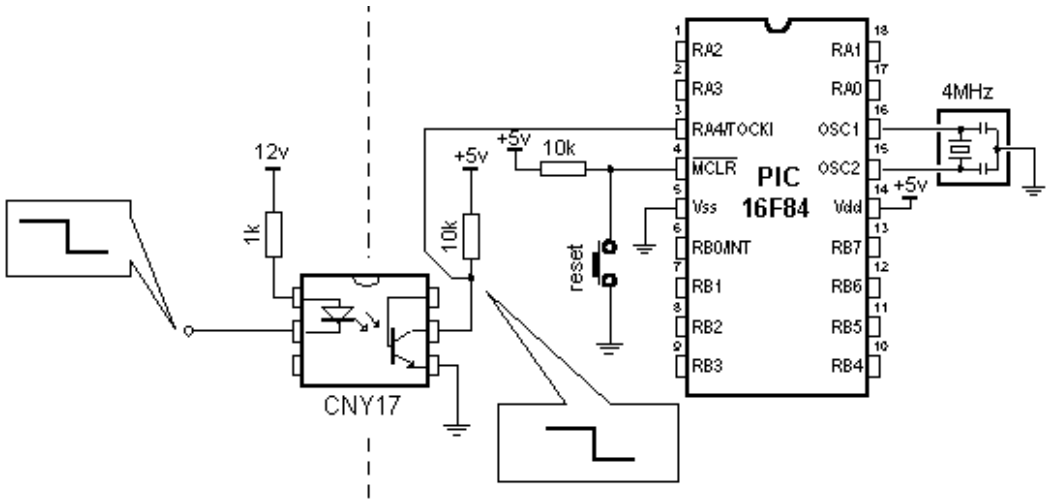
Aceasta este realizat• pentru un num•r de motive:

- **Interferen•a.** O parte a unui circuit poate fi într-o zon• unde este influen•at de interferen•e (cum ar fi cele de la motoarele electrice, echipamente de sudur•, motoare termice etc.). Dac• ie•irea acestui circuit trece printr-un optocuplor spre alt circuit, numai semnalele dorite vor trece prin optocuplor. Semnalele de interferen•• nu vor avea destul• „putere” s• activeze LEDul din optocuplor •i de aceea ele sunt eliminate. Exemplele tipice sunt unit••ile industriale care au mai multe interferen•e care afecteaz• semnalele pe cablu. Dac• aceste interferen•e afecteaz• func•ia unei sec•iuni de control, vor apare erori •i unitatea nu va mai func•iona.
- **Separare simultan• •i intensitatea semnalului.** Un semnal mai mic de 3v este capabil s• activeze un optocuplor •i ie•irea optocuplorului poate fi conectat• la o linie de intrare a microcontrolerului. Microcontrolerul are nevoie de un impuls de intrare de 5v •i în caz semnalul de 3v este amplificat la 5v. Poate fi folosit pentru a amplifica curentul semnalului. Uita•i-v• mai jos pentru utilizarea unei linii de ie•ire a microcontrolerului pentru amplificare de curent.
- **Separare de tensiune mare.** Optocuploarele au calit••i înn•scute pentru separarea tensiunilor mari. Deoarece LEDul este complet separat de fototranzistor, optocuploarele pot da dovad• de izolare de tensiune de 3Kv sau chiar mai mare.

Optocuploarele pot fi folosite ca dispozitive de intrare sau ie•ire. Ele au func•ii adi•ionale cum ar fi Schmitt triggering (ie•irea unui Schmitt trigger este 0 sau 1 – se schimb• încet ridicând •i coborând forma de und• în valori definite LOW sau HIGH). Optocuploarele sunt împachetate ca o singur• unitate sau în grupuri de dou• sau mai multe într-o singur• capsul•. Ele mai sunt denumite foto-întrerup•toare în care un disc cu fante este introdus într-un l•ca• între LED •i fototranzistor •i de fiecare dat• când lumina este întrerupt•, tranzistorul produce un impuls. Fiecare optocuplor are nevoie de dou• aliment•ri pentru a func•iona. Ele pot fi folosite cu o alimentare, dar capacitatea de izolare a tensiunii este pierdut•.

Optocuplor pe o linie de intrare

Modul de func•ionare este simplu: când ajunge un semnal, LEDul din optocuplor este aprins •i lumineaz• pe baza fototranzistorului din aceea•i carcas•. În momentul în care tranzistorul este activat, tensiunea dintre colector •i emitor cade la 0.5v sau mai pu•in •i microcontrolerul sesizeaz• acest lucru ca zero logic pe pinul RA4. Exemplul de mai jos este un contor, folosit pentru num•rarea produselor de pe o linie de produc•ie, pentru determinarea vitezei motorului, pentru contorizarea num•rului de revolu•ii a unei axe etc. Consider•m senzorul ca un microîntrerup•tor. De fiecare dat• când întrerup•torul este închis, LEDul este luminat. LEDul „transfer•” semnalul c•tre fototranzistor •i opera•ia fototranzistorului livreaz• LOW c•tre intrarea RA4 a microcontrolerului. Un program în microcontroler va fi necesar pentru a preveni contoriz•rile false •i un indicator conectat la oricare dintre ie•irile microcontrolerului va indica starea curent• a contorului.



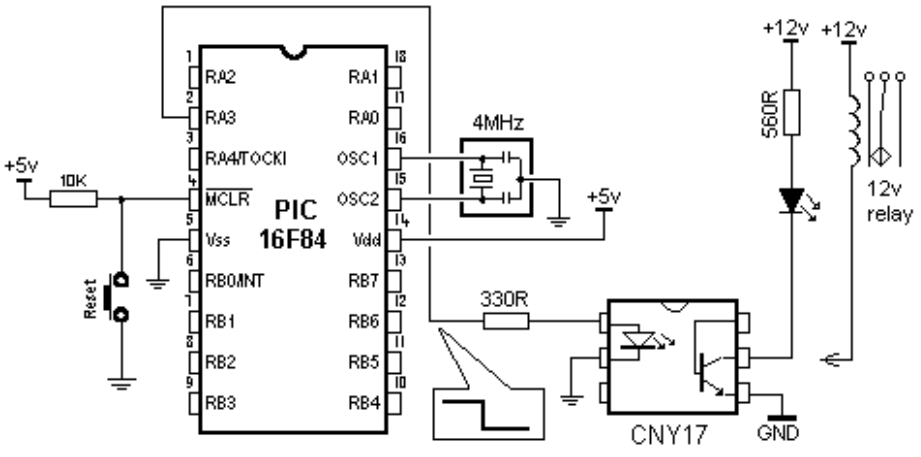
Exemplu de linie de intrare cu optocuplor

OPTO_IN.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Structure of program memory *****

        ORG    0x00            ; Reset vector
        goto   Main

        ORG    0x04            ; Interrupt vector
        goto   Main            ; No interrupt routine

        #include "bank.inc"    ; Assistant files

Main                           ; Beginning of the program
        BANK1
        movlw  0xff            ; PORTA initialization
        mowwf  TRISA           ; TRISA <- 0xff (input)
        movlw  0x00            ; PORTB initialization
        mowwf  TRISB           ; TRISB <- 0x00 (output)
        movlw  b'00110000'     ; RA4 -> TMR0,
        mowwf  OPTION_REG      ; Increment TMR0 on failing edge
        BANK0

        clrf   PORTB ; PORTB <- 0
        clrf   TMR0  ; TMR0 <- 0

Loop

        movf   TMR0,w          ; Copy TMR0 in W reg.
        mowwf  PORTB           ; send value of W reg. on PORTB
        goto   Loop            ; Repeat the loop

   End          ; End of program
```

Optocuplor pe o linie de ie•ire

Un optocuplor poate fi folosit pentru a separa semnalul de ie•ire a unui microcontroler fa•• de un dispozitiv de ie•ire. Acest lucru poate fi necesar pentru separarea tensiunilor înalte sau pentru amplificare. Ie•irea unor anumite microcontrolere este limitat• la 25mA. Optocuplorul va lua semnal de curent sc•zut din microcontroler •i tranzistorul de ie•ire va comanda un LED sau un releu, cum este exemplificat mai jos:



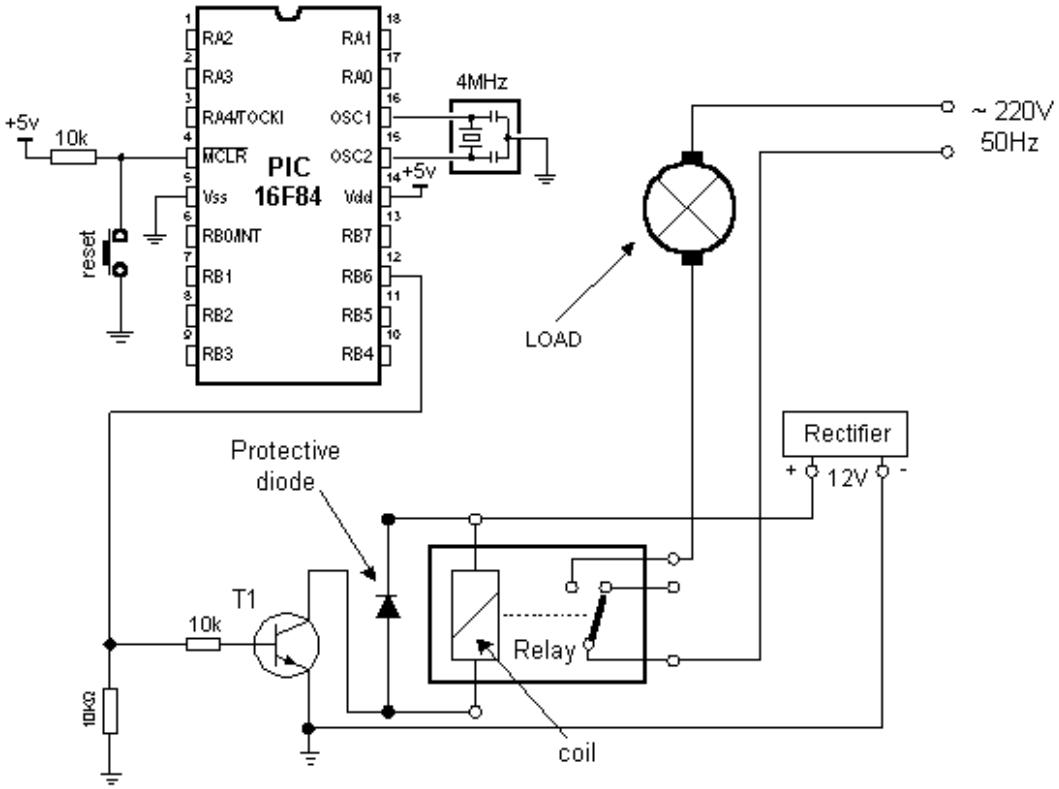
Output line optocoupler example

Programul pentru acest exemplu este simplu. Prin livrarea unui ‚1' logic în pinul 4 al portului A, LEDul se va aprinde •i tranzistorul va fi activat în optocuplor. Orice dispozitiv conectat la ie•irea optocuplorului va fi activat. Curentul limit• pentru tranzistor este în jur de 250mA.

Pagina anterioar•	Con•inut	Pagina urm•toare

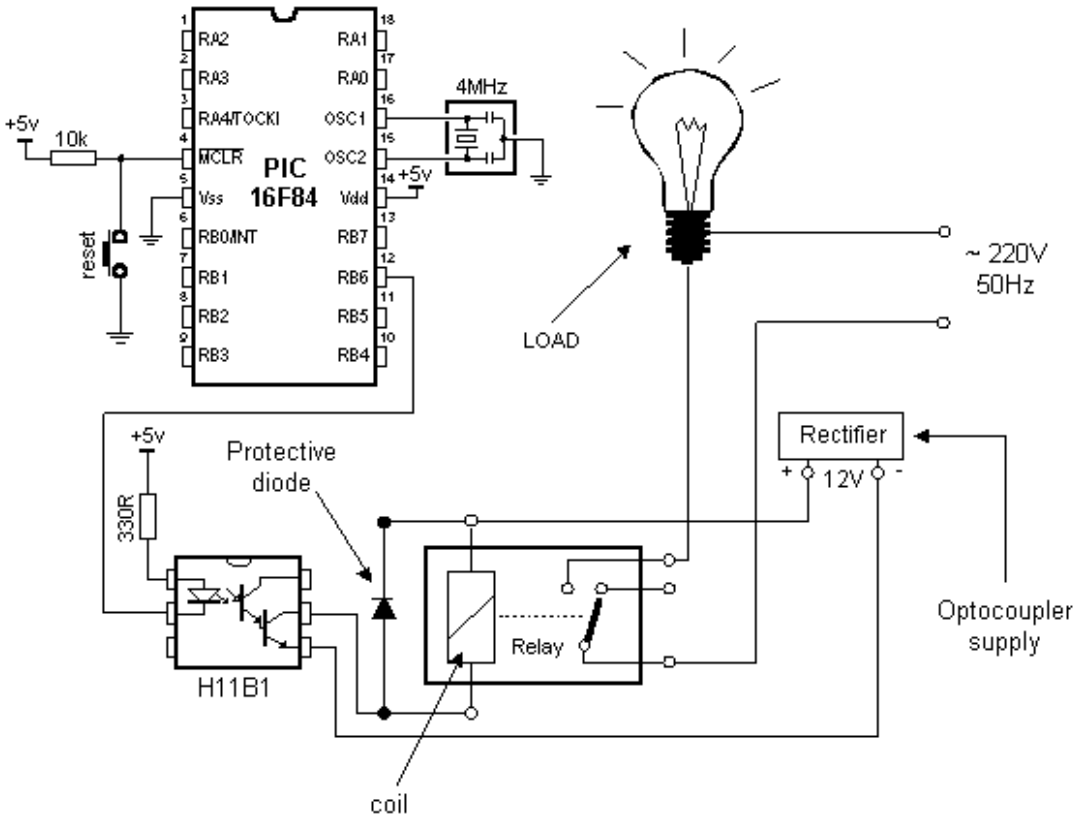
Releul

Releul este un dispozitiv electromecanic care transform• un semnal electric într-o mi•care mecanic•. El este alc•tuit dintr-o bobin• din conductori izola•i înf••ura•i pe un nucleu metalic •i o arm•tur• metalic• cu unul sau mai multe contacte. În momentul în care o tensiune de alimentare este aplicat• la bornele unei bobin•, curentul circul• •i va fi produs un câmp magnetic care mi•c• arm•tura pentru a închide un set de contacte •i/sau pentru a deschide un alt set. Când alimentarea este dezactivat• din releu, cade fluxul magnetic din bobin• •i se produce o tensiune înalt• în direc•ia opus•. Aceast• tensiune poate strica tranzistorul de comand• •i de aceea este conectat• o diod• cu polarizare invers• de-a lungul bobinei pentru a scurtcircuita vârfurile de tensiune în momentul în care apar.



Conectarea unui releu la microcontroler prin intermediul unui tranzistor

Multe microcontrolere nu pot comanda un releu direct •i de aceea un tranzistor de comand• este necesar. Un HIGH pe baza tranzistorului activeaz• tranzistorul •i acesta la rândul lui activeaz• releul. Releul poate fi conectat la orice dispozitiv electric prin intermediul contactelor. Rezisten•a de 10K din baza tranzistorului limiteaz• curentul dinspre microcontroler la o valoare solicitat• de tranzistor. Rezisten•a de 10K dinspre baz• •i bara negativ• previne ca tensiunile de zgomot aplicate în baza tranzistorului s• activeze releul. De aceea numai un semnal clar de la microcontroler va activa releul.



Connecting the optocoupler and relay to a microcontroller

Un releu poate fi de altfel activat prin intermediul unui optocuplor care în acel•i timp amplific• curentul provenit de la ie•irea microcontrolerului •i ofer• un grad înalt de izolare. Optocuploarele **HIGH CURRENT** deobicei con•in un tranzistor cu o ie•ire „Darlington" pentru a oferi curent mare de ie•ire. Conectarea prin intermediul unui optocuplor este recomandat• în mod special pentru aplica•iile microcontroler unde motoarele sunt activate •i zgomotulele de comuta•ie provenite de la motor pot ajunge în microcontroler prin intermediul liniilor de alimentare. Optocuplorul comand• un releu iar releul activeaz• motorul. Figura de mai jos arat• programul necesar pentru activarea releului •i include câteva din macrourile deja discutate.

RELAY.asm

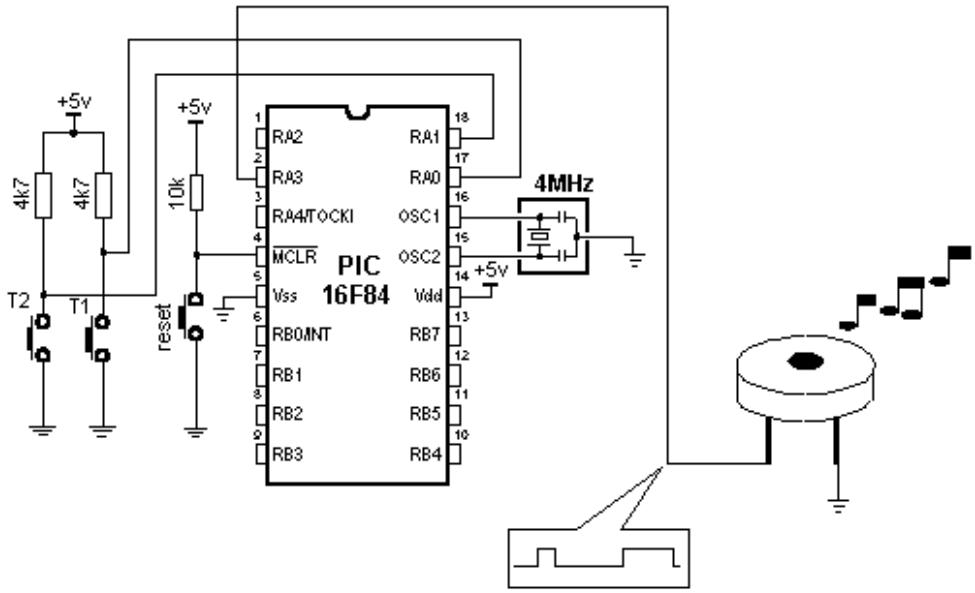
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG   _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        endc

;***** Declaring the hardware *****

        #define RELAY PORTA,3

;***** Structure of program memory *****

        ORG    0x00             ; Reset vector
        goto   Main

        ORG    0x04             ; Interrupt vector
        goto   Main             ; No interrupt routine

        #include "bank.inc"      ; Assistant files
        #include "button.inc"
        #include "wait.inc"
```

```
        #include "button.inc"
        #include "wait.inc"

Main                            ; Beginning of the program
        BANK1
        movlw   0x17            ; PORTA initialization
        mowwf   TRISA           ; TRISA <- 0x17
        movlw   0x00            ; PORTB initialization
        mowwf   TRISB           ; TRISB <- 0x00
        BANK0

        clrf    PORTB           ; PORTB <- 0x00

Loop
        Button 0, PORTA, 0, .100, On    ; Button 0
        Button 0, PORTA, 1, .100, Off   ; Button 1

        goto    Loop            ; Repeat the loop

On
        bsf RELAY               ; Turn on relay
        return

Off
        bcf RELAY               ; Turn off relay
        return

   End          ; End of program
```

Pagina anterioar• | Con•inut | Pagina urm•toare

Generarea unui sunet

Un buzzer piezo poate fi ad•ugat• la o linie de ie•ire a unui microcontroler pentru a livra tonuri „audio”, piuituri •i semnale. Este important de •tiut c• sunt dou• mari tipuri de dispozitive piezoelectrice emi••toare de sunet. Una are componente active în•untrul carcasei •i are nevoie numai de alimentare de curent continuu pentru a emite un ton sau un beep. În general tonurile sau beep-urile emise de aceste difuzoare sau piuitoare nu pot fi schimbate – ele sunt fixe din cauza circuitelor interne. Acesta nu este tipul despre care discut•m în acest articol. Cel•lalt tip const• dintr-un buzzer piezo •i necesit• semnal livrat în ea pentru a func•iona. Depinzând de frecven•a formei de und•, ie•irea poate fi ton, melodie, alarm• sau chiar un mesaj vocal. Pentru ca ele s• func•ioneze trebuie s• livr•m un ciclu care este alc•tuit din semnale HIGH •i LOW. Tranzi•ia de la HIGH la LOW sau de la LOW la HIGH cauzeaz• mi•c•ri diafragmei pentru a produce secven•e de sunete. Forma de und• poate avea o schimbare fin• de la o valoare la alta (denumit• und• sinusoidal•) sau o schimbare rapid• (denumit• und• dreptunghiular•). Un calculator este ideal pentru producerea de unde dreptunghiulare. Livrarea de unde dreptunghiulare produce o ie•ire u•or grosier•. Conectarea unui buzzer piezo este foarte u•oar•. Un pin este conectat la linia negativ•, iar cealalt• la o ie•ire a microcontrolerului, dup• cum este ilustrat în figura de mai jos. Acesta va livra o form• de und• de 5v c•tre buzzerul piezo. Pentru a produce o tensiune mai mare, forma de und• trebuie amplificat• •i aceasta necesit• un tranzistor de comand• •i o bobin•.



Conectarea unui buzzer piezo la un microcontroler

Ca •i în cazul tastaturii, pute•i folosi un macro care va furniza o rutin• BEEP într-un program când va fi necesar.

BEEP **macro** freq, duration

freq: frecven•a sunetului. Un num•r mai mare produce o frecven•• mai înalt•.
duration: durata sunetului. Un num•r mai mare reprezint• un sunet mai lung.

Exemplu 1: BEEP 0xFF, 0x02

Ie•irea buzzerului piezo are cea mai înalt• frecven•• •i durata de **2 cicluri de 65.3ms**, ceea ce rezult• 130.6ms.

Exemplu 2: BEEP 0x90, 0x05

Ie•irea buzzerului piezo are frecven•a de 0x90 •i durata de 5 cicluri de 65.3ms. Este bine ca argumentele macroului s• fie determinate prin experimente •i astfel s• fie ales sunetul care se potrive•te cel mai bine pentru aplica•ie. În continuare este prezentat macroul BEEP:

BEEP.inc

```
;***** Declaring constants *****

        CONSTANT PRESCbeep = b'00000111'            ; 65,3 ms per cycle

;***** Macros *****

BEEP    macro   freq,duration

        movlw   freq
        mowwf   Beep_TEMP1
        movlw   duration
        call    BEEPsub
        endm

BEEPinit macro

        bcf     BEEPport
        BANK1
        bcf     BEEPtris
        BANK0
        endm

;***** Subroutines *****

BEEPsub
        mowwf   Beep_TEMP2          ; Set the value of sound duration
        clrf    TMR0                ; Initialize the counter
        bcf    BEEPport
        BANK1
        bcf    BEEPport
        movlw   PRESCbeep           ; Set the prescaler for TMR0
        mowwf   OPTION_REG          ; OPTION <- W
        BANK0
BEEPa
         bcf    INTCON,T0IF     ; Erase TMR0 Overflow Fleg
BEEPb
        bsf     BEEPport
        call    B_Wait          ; Duration of logic "1"
        bcf     BEEPport
        call    B_Wait          ; Duration of logic "0"
        btfss   INTCON,T0IF     ; Check the TMR0 Overflow Fleg,
        goto    BEEPb           ; Skip of it is set
        decfsz  Beep_TEMP2,1    ; Is the Beep_TEMP2 = 0 ?
        goto    BEEPa           ; If not, jump to BEEP again
        RETURN

B_Wait
        movfw   Beep_TEMP1
        mowwf   Beep_TEMP3
B_Waita
        decfsz  Beep_TEMP3,1
        goto    B_Waita
        RETURN
```

Urm•torul exemplu arat• întrebuin•area unui macro într-un program. Programul produce dou• melodii care sunt ob•inute prin ap•sarea T1 sau T2. Câteva din macrourile discutate anterior sunt incluse în program.

BEEP.asm

```
;***** Declaring and configuring a microcontroller *****

        PROCESSOR 16f84
```

```
      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock  0x0C            ; Beginning of RAM-a
        WCYCLE                  ; Belongs to 'WAITX' macro
        PRESCwait
        Beep_TEMP1              ; Belongs to 'BEEP' macro
        Beep_TEMP2
        Beep_TEMP3
        endc

;***** Declaring the hardware *****

        #define  BEEPport PORTA,3 ; Port and pin for mini speaker
        #define  BEEPtris TRISA,3

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    Main            ; No interrupt routine

       #include "bank.inc"      ; Assistant files
       #include "button.inc"
       #include "wait.inc"
       #include "beep.inc"

Main                            ; Beginning of the program
        BANK1
        movlw   0x17            ; Port A initialization
        mowwf   TRISA           ; TRISA <- 0x00
        movlw   0x00
        mowwf   TRISB
        BANK0

        BEEPinit                ; Mini speaker initialization

        clrf    PORTB

Loop
        Button 0, PORTA, 0, .100, Play1         ; Button 1
        Button 0, PORTA, 1, .100, Play2         ; Button 2
        goto    Loop

Play1
        BEEP 0xFF, 0x02
        BEEP 0x90, 0x05
        BEEP 0xC0, 0x03
        BEEP 0xFF, 0x03                         ; First melody
        return

Play2
        BEEP 0xbb, 0x02
        BEEP 0x87, 0x05
        BEEP 0xa2, 0x03
        BEEP 0x98, 0x03                         ; Second melody
        return

      End               ; End of program
```

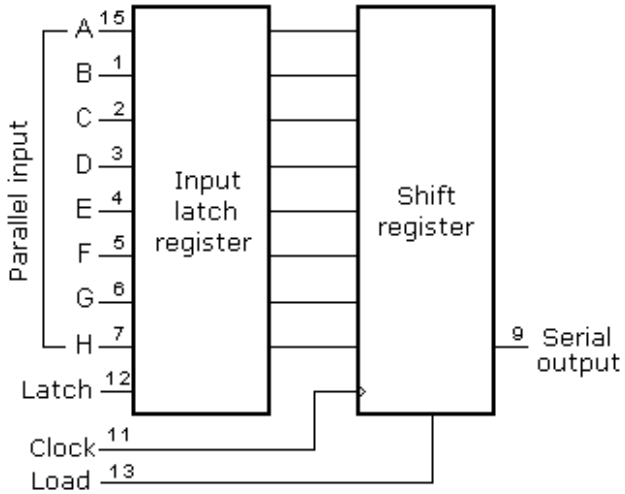
Pagina anterioar• Con•inut Pagina urm•toare

Regi•trii de deplasare

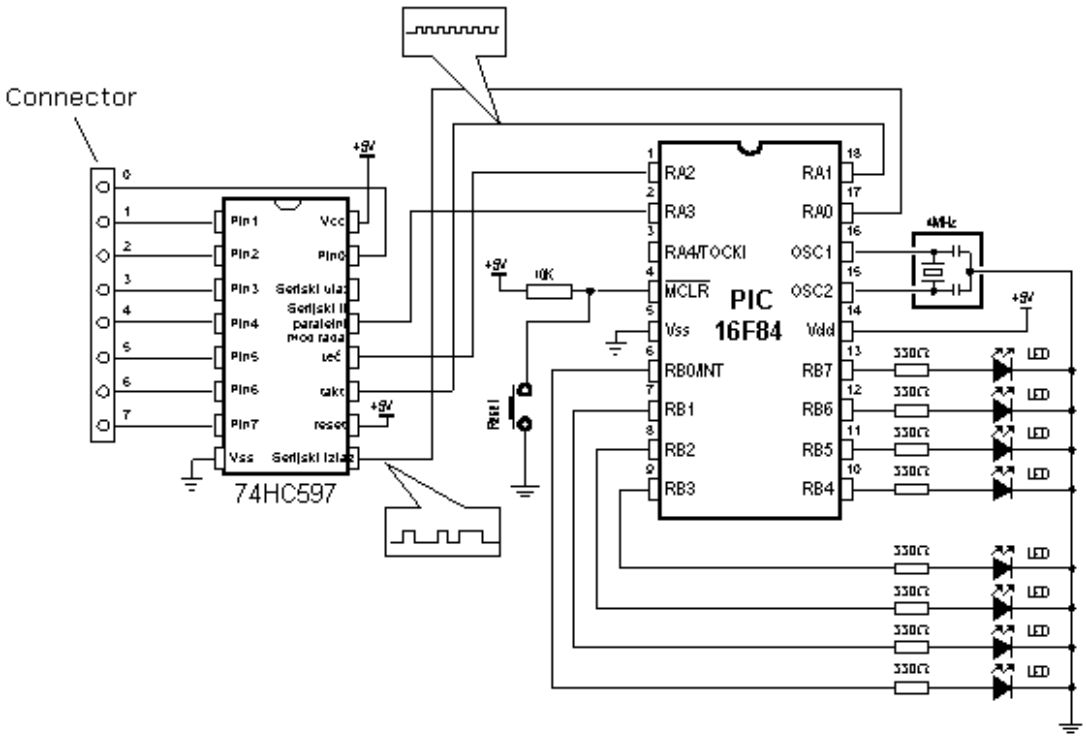
Exist• dou• tipuri de regi•trii de deplasare: **de intrare •i de ie•ire**. **Regi•trii de intrare** încarc• datele paralel, prin intermediul a 8 linii, •i apoi le trimite serial prin intermediul a dou• linii c•tre microcontroler. **Regi•trii de ie•ire** opereaz• în direc•ie opus•: primesc date serial •i la un semnal pe linia „latch", transform• datele în date paralele. Regi•trii de deplasare sunt folosi•i în general pentru a m•ri num•rul de intr•ri – ie•iri ale unui microcontroler. Ei nu prea mai sunt folosi•i pentru c• microcontrolerele moderne au un num•r mare de linii intrare – ie•ire. Oricum, utilizarea lor cu microcontrolere cum ar fi PIC16F84 este foarte important•.

Regi•trii de deplasare de intrare 74HC597

Rei•trii de deplasare de intrare transform• datele paralele în date seriale •i le transfer• c•tre microcontroler. Modul lor de func•ionare este simplu. Sunt patru linii pentru transferul datelor: **clock, latch, load •i data**. Datele sunt citite de la pinii de intrare de un registru intern prin intermediul unui semnal „latch". Apoi, cu un semnal „load", datele sunt transferate de la registrul „latch" de intrare c•tre registrul de deplasare, iar de acolo sunt transferate serial c•tre un microcontroler prin intermediul liniilor „data" •i „clock".



O schem• de leg•tur• a registrului de deplasare 74HC597 la un microcontroler este prezentat• mai jos:



Modalitatea de conectare a unui registru de deplasare de intrare la un microcontroler

Pentru simplificarea programului principal, un macro poate fi utilizat pentru registrul de deplasare de intrare. Macroul HC597 are dou• argumente:

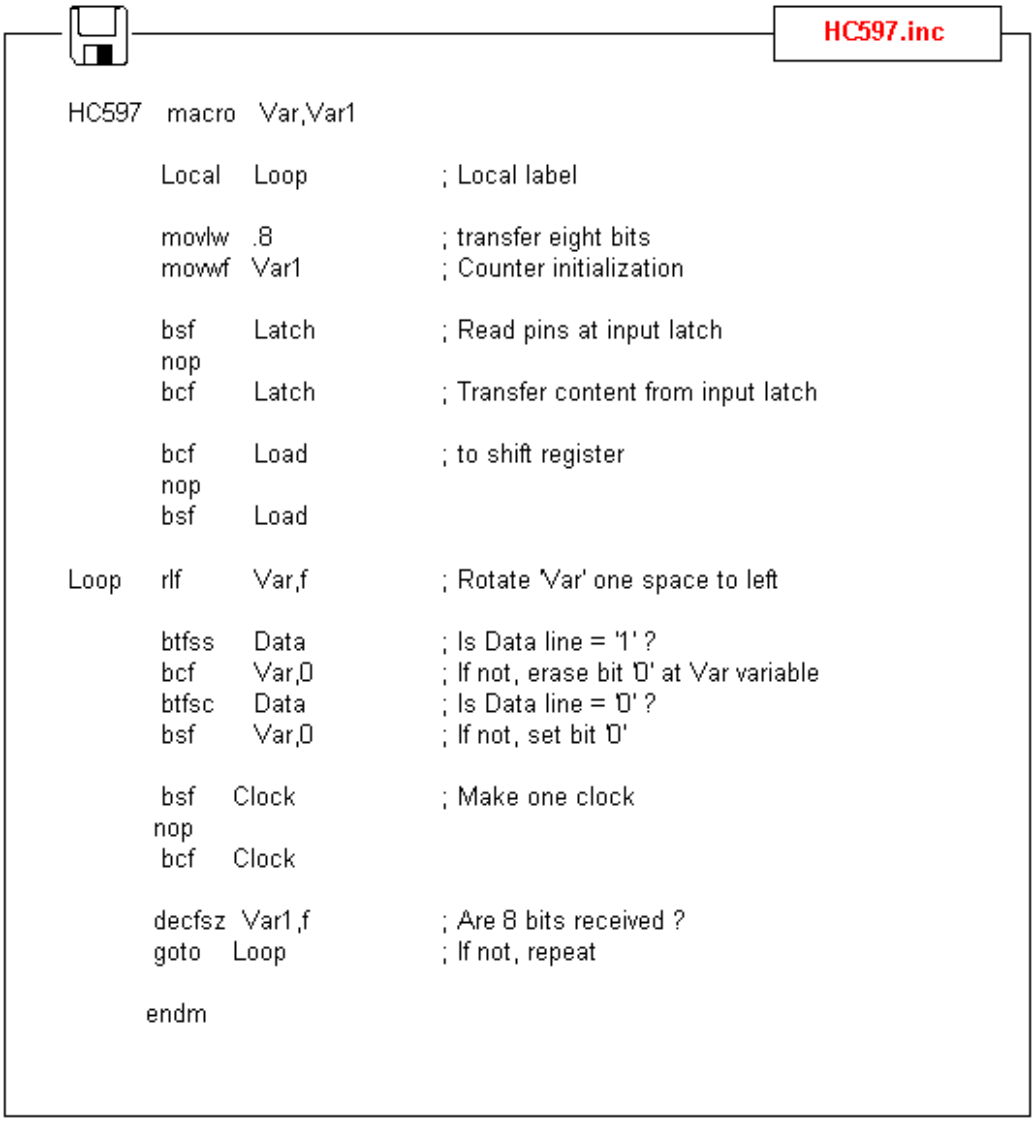
HC597 macro Var, Var1

Var variabil• unde datele provenite de la pinii registrului de deplasare de intrare sunt transferate.
Var1 contor bucl•.

Exemplu: HC597 data, counter

Datele provenite de la pinii registrului de deplasare sunt stocate în variabila data. Variabila Time/counter este folosit• pe post de contor bucl•.

Textul macroului:



HC597.inc

```
HC597   macro   Var,Var1

        Local   Loop            ; Local label

        movlw   .8              ; transfer eight bits
        mowwf   Var1            ; Counter initialization

        bsf     Latch           ; Read pins at input latch
        nop
        bcf     Latch           ; Transfer content from input latch

        bcf     Load            ; to shift register
        nop
        bsf     Load

Loop    rlf     Var,f           ; Rotate 'Var' one space to left

        btfss   Data            ; Is Data line = '1' ?
        bcf     Var,0           ; If not, erase bit '0' at Var variable
        btfsc   Data            ; Is Data line = '0' ?
        bsf     Var,0           ; If not, set bit '0'

        bsf    Clock            ; Make one clock
       nop
        bcf    Clock

        decfsz  Var1,f          ; Are 8 bits received ?
        goto    Loop            ; If not, repeat

       endm
```

Exemplul care v• arat• cum s• folosi•i macroul HC597 este în programul urm•tor. Programul recep•ioneaz• date de la intrarea paralel• a registrului de deplasare •i le mut• serial în variabila RX a microcontrolerului. LEDurile conectate la portul B vor indica rezultatul datelor de intrare.

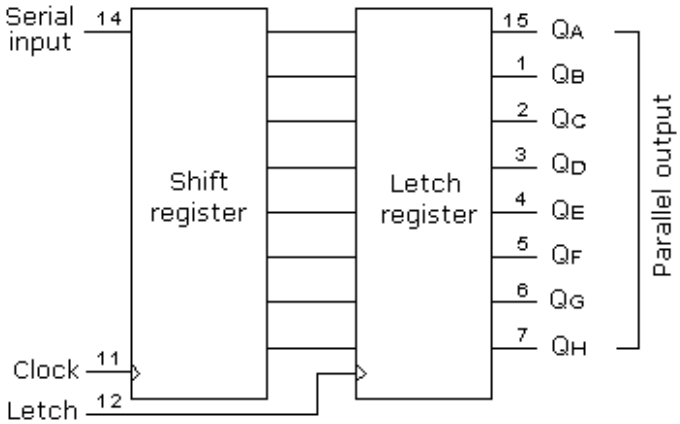
HC597.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock  0x0C            ; Beginning of RAM
        RX
        CountSPI
        endc

;***** Declaring the hardware *****

        #define  Data    PORTA,0
        #define  Clock   PORTA,1
        #define  Latch   PORTA,2
        #define  Load    PORTA,3

;***** Program memory structure *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    Main            ; No interrupt routine

       #include "bank.inc"      ; Assistant files
       #include "hc597.inc"

Main                            ; Beginning of the program
        BANK1
        movlw   b'00010001'     ; Port A initialization
        mowwf   TRISA           ; TRISA <- 0x11
        clrf    TRISB
        BANK0

        clrf    PORTA           ; PORTA <- 0x00

        bsf     Load            ; Enable SHIFT register

Loop
        HC597   RX, CountSPI

                                ; Status of input pins of SHIFT register
        movf    RX,W            ; are found in variable RX
        mowwf   PORTB           ; Set the contents of RX variable to port B
        goto    Loop            ; Repeat the loop

        End                     ; End of program
```

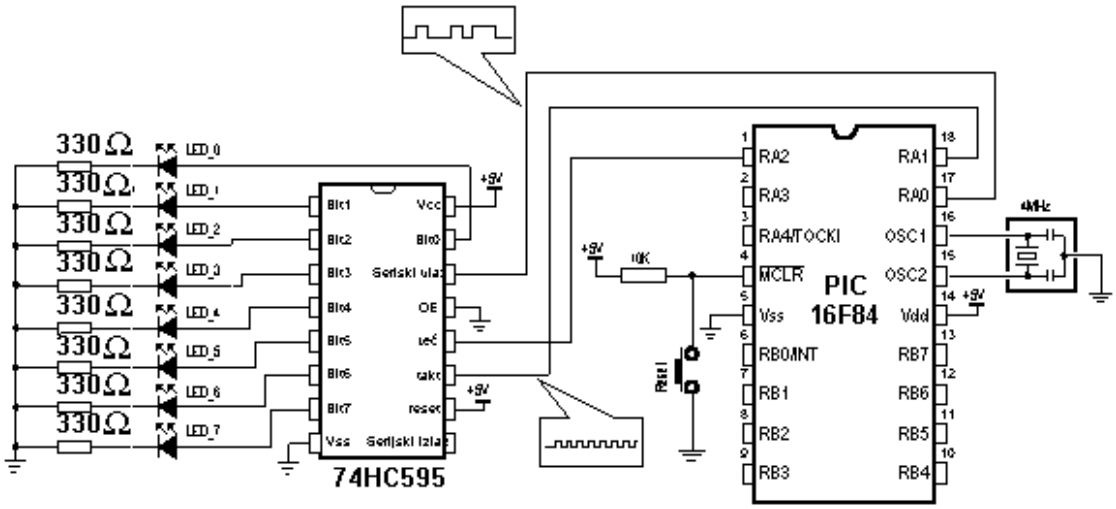
Registru de deplasare de ie•ire

Regi•trii de deplasare de ie•ire transform• datele seriale în date paralele. Pe fiecare front cresc•tor al tactului, registrul de deplasare cite•te valoarea de la linia de date, o memoreaz• într-un registru temporar, apoi repet• acest ciclu de 8 ori. La un semnal de la linia

„latch", datele sunt copiate din registrul de deplasare în registrul de intrare, apoi datele sunt transformate din date seriale în date paralele.



O schem• a registrului de deplasare este prezentat• mai jos:



Conectarea unui registru de deplasare de ie•ire la un microcontroler

Macroul folosit în acest exemplu este localizat în fisierul HC595.INC •i se nume•te HC595.

Macroul HC595 are dou• argumente:

Var variabil• a c•rei con•inut este transferat la ie•irea registrului de deplasare.
Var1 contor bucl•.

Exemplu: HC595 Data, Counter

Datele pe care vrem s• le transfer•m sunt stocate în variabila Data, iar variabila Counter este folosit• pe post de contor bucl•

HC595.inc

```
HC595  macro  Var,Var1

       Local  Loop          ; Local label

       movlw  .8            ; transfer eight bits
       movwf  Var1          ; Counter initialization

Loop   rlf    Var,f         ; Rotate 'Var' one space to left

       btfss  STATUS,C      ; Is carry = '1' ?
       bcf    Data          ; If not, set Data line to '0'
       btfsc  STATUS,C      ; Is carry = '0' ?
       bsf    Data          ; If not, set Data line to '1'

       bsf    Clock         ; Make one clock
       nop
       bcf    Clock

       decfsz Var1,f    ; Are eight bits sent ?
       goto   Loop      ; If not, repeat

       bsf    Latch     ; If all 8 bits have beem sent, move the
       nop              ; contents form SHIFT register to output latch
       bcf    Latch

       endm
```

Un exemplu al utiliz•rii macroului HC595 este în programul urm•tor. Datele provenite de la variabila TX sunt transferate serial în registrul de deplasare. LEDurile conectate la ie•irea paralel• a registrului de deplasare vor indica starea liniilor. În acest exemplu valoarea 0xCB (11001011) este transmis• astfel încât LEDurile 8, 7, 4, 2 •i 1 sunt iluminate.

HC595.asm

```
;***** Declaring and configuring a microcontroller *****

     PROCESSOR 16f84
     #include "p16f84.inc"

       __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock  0x0C         ; Beginning of RAM
        TX                   ; Belongs to 'HC595' macro
        CountSPI
        endc

;***** Declaring the hardware *****

       #define Data    PORTA,0
       #define Clock   PORTA,1
       #define Latch   PORTA,2

;***** Structure of program memory *****

       ORG    0x00          ; Reset vector
       goto   Main

       ORG    0x04          ; Interrupt vector
       goto   Main          ; No interrupt routine

       #include "bank.inc"  ; Assistant files
```

```
        #include "bank.inc"      ; Assistant files
        #include "hc595.inc"

Main                             ; Beginning of the program

        BANK1
        movlw  0x18              ; Port A initialization
        mowwf  TRISA             ; TRISA <- 0x18
        BANK0

        clrf     PORTA           ; PORTA <- 0x00

        movlw  0xCB              ; Fill the TX buffer
        mowwf  TX                ; TX <- '11001011'

        HC595  TX, CountSPI

Loop    goto     Loop    ; Infinite loop

      End                ; End of program
```

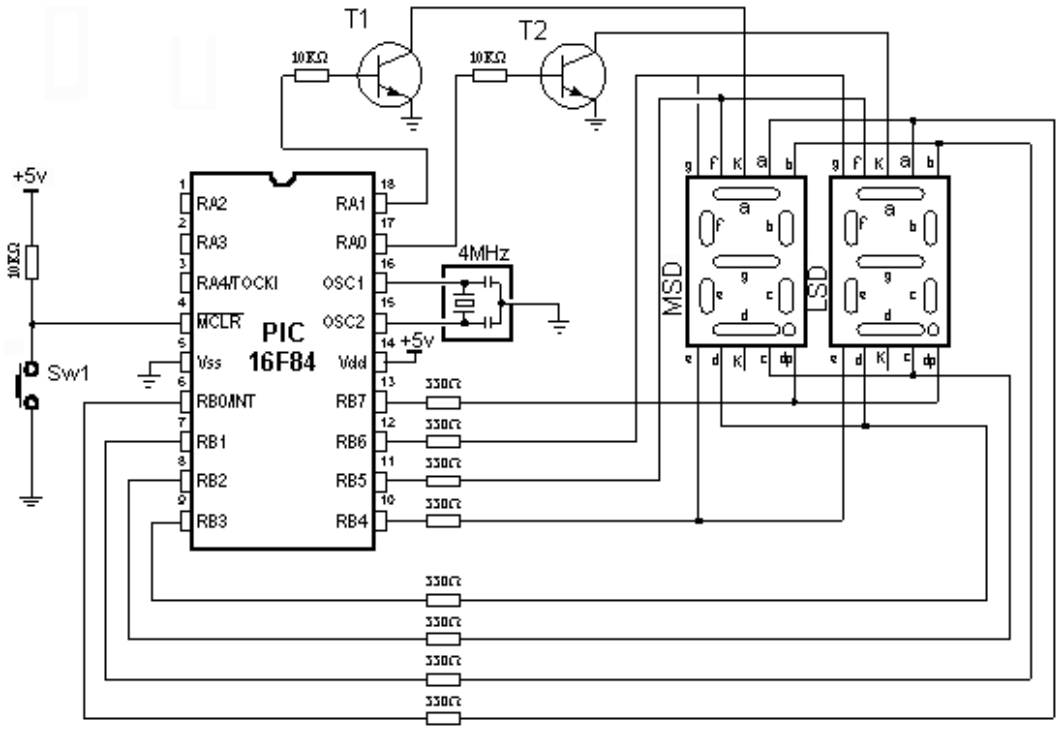
Pagina anterioar• Con•inut Pagina urm•toare

Afi•aj cu 7 segmente (multiplexare)

Segmentele într-un afi•aj cu 7 segmente sunt aranjate astfel încât s• formeze un singur digit de la 0 la F, dup• cum se observ• în desen:

Putem afi•a un num•r pe mai mul•i digi•i prin conectarea de afi•aje adi•ionale. Chiar dac• este mult mai confortabil s• lucr•m cu LCDuri, afi•ajele cu 7 segmente sunt înc• un standard în industrie. Aceasta din cauza rezisten•ei la temperatur•, vizibilit••ii •i unghiului larg de observare. Segmentele sunt marcate cu litere mici: a, b, c, d, e, f, g •i dp, unde dp este punctul zecimal. Cele 8 LEDuri din cadrul fiec•rui afi•aj pot fi aranjate cu catod comun sau cu anod comun. La un afi•aj cu catod comun, catodul comun trebuie s• fie conectat la linia de 0v •i LEDurile sunt activate cu unu logic. Afi•ajele cu anod comun trebuie s• prezinte anodul comun conectat la linia de +5v. Segmentele sunt activate cu zero logic. Dimensiunea afi•ajului este m•surat• în milimetri; se m•soar• doar în•l•imea digitului (nu carcasa, doar digitul!). Afi•ajele sunt disponibile cu digi•i de în•l•imi de 7, 10, 13.5, 20 sau 25 milimetri. Sunt de diferite culori incluzând: ro•u, portocaliu •i verde. Cea mai simpl• metod• pentru a comanda un afi•aj este prin intermediul unui driver de afi•aj. Acestea sunt disponibile pentru pân• la 4 afi•aje. Alternativ, afi•ajele pot fi comandate de un microcontroler, •i, dac• este necesar mai mult decât un afi•aj, metoda de comandare se nume•te „multiplexare”. Principala diferen•• dintre cele dou• metode este num•rul de linii de comand•. Un driver special poate avea numai o singur• linie de tact •i integratul de comand• va accesa toate segmentele •i va incrementa afi•ajul. Dac• avem doar un singur afi•aj de comandat de c•tre microcontroler, vor fi necesare 7 linii plus una pentru punctul zecimal. Pentru fiecare afi•aj zecimal, este necesar• doar câte o linie în plus. Pentru a produce un afi•aj cu 4, 5 sau 6 digi•i, toate afi•ajele cu 7 segmente vor fi conectate în paralel. Linia comun• (linia catodului comun) este conectat• separat •i aceast• linie este conectat• la zero logic pentru o perioad• scurt• de timp pentru a activa afi•ajul. Fiecare afi•aj este activat de 100 ori pe secund• •i vor da impresia c• toate afi•ajele sunt active în acela•i timp. În timp ce fiecare afi•aj este activat, informa•ia trebuie livrat• astfel încât el va afi•a informa•ia corect•. Pot fi accesate pân• la 6 afi•aje în acest mod f•r• ca str•lucirea fiec•rui afi•aj s• fie afectat•. Fiecare afi•aj este activat efectiv pentru 1/6 din timp •i persisten•a vizual• a ochilor d• impresia c• afi•ajul este pornit tot timpul. Toate semnalele de sincronizare pentru afi•aj sunt produse de program, avantajul unui afi•aj controlat de un microcontroler este flexibilitatea. Afi•ajul poate fi configurat ca un contor cresc•tor, contor descresc•tor, •i poate produce un num•r de mesaje folosind literele alfabetului care pot fi u•or de afi•at.

Exemplul de mai jos arat• cum s• control•m dou• afi•aje.



Conectarea unui microcontroler cu afi•aje cu 7 segmente în mod multiplexat

Fi•ierul LED.INC con•ine dou• macrouri: LED_Init •i LED_Disp2. Primul macro este folosit pentru initializarea afi•ajului. Aici este definit• perioada de reîmprosp•tare cât •i pinii microcontrolerului utiliza•i pentru conectarea afi•ajelor.

Macroul LED_Disp2 are un argument:

LED_Disp2 macro first

first este num•rul de la 0 la 99 care trebuie afi•at pe digi•ii MSD •i LSD.

Exemplu: LED_Disp2 0x34

Num•rul 34 va fi afi•at.

LED.inc

```
;***** Macros *****

LED_Init macro
      call  InitPorts
      call  InitTimers
      endm

LED_Disp2 macro num
      movlw num
      mowwf LO
      call  UpdateDisplay
      endm

;***** Subprograms *****

InitPorts
      BANK1
      clrf   LEDtrisA          ; Pins RA0-4 are output
      clrf   LEDtrisB          ; Port B is output
      BANK0
      clrf   LEDportA          ; Set all outputs to '0'
      clrf   LEDportB          ;      /
      bsf    LEDportA,3        ; Turn on MSD display

      RETURN

InitTimers
      BANK1
      movlw  B'10000100'       ; Move the prescaler to TMR0
      mowwf  OPTION_REG        ; ps = 32
      BANK0
      movlw  B'00100000'       ; Enable TMR0 interrupt
      mowwf  INTCON            ;
      movlw  .96               ;
      mowwf  TMR0              ; Start the timer

      RETFIE


;***** ISR - Interrup service routine ******

ISR
      bcf    INTCON,GIE        ; Disable all interrupts
      btfsc  INTCON,GIE        ; Check whether they are diasbled
      goto   ISR

      movlw  .96               ; Initialize the TMR0
      mowwf  TMR0
      bcf    INTCON,T0IF       ; Erase the T0IF flag
      call   UpdateDisplay     ; "Refresh" dispaly

      RETFIE
```

```
        RETFIE

UpdateDisplay
        movf    LEDportA,W          ; Display status -> w register
        clrf    LEDportA            ; Turn off all 7-seg. displays
        andlw   0x0f                ; Separate the lower halfbyte
           mowwf   TempC            ; Save display status in TempC
        bsf     TempC,4             ; Beginning status of LSD display
        rrf     TempC,F             ; Set the status of the next display
        btfss   STATUS,C  ; c=1 ?
        bcf     TempC,3             ; If not, turn off the LSD display
        btfsc   TempC,0             ; If it is, check the status of MSD display
        goto    UpdateMsd           ; If it is turned on, display the MSD digit
UpdateLsd
        call    ChkMsdZero          ; msd = 0 ?
        btfss   STATUS,Z            ; If it is, skip
        movf    LO,W                ; Third LSD digit -> W
        andlw   0x0f                ;         /
        goto    DisplayOut          ; Show it on the display
UpdateMsd
        swapf   LO,W                ; Msd digit -> W
        andlw   0x0f                ;         /
        btfsc   STATUS,Z            ; msd != 0 ?
        movlw   0x0a                ; If it is, skip
DisplayOut
        call    LedTable            ; Take the mask for a digit
        mowwf   LEDportB            ; Set the mask on port B
        movf    TempC,W             ; Turn on displays
        mowwf   LEDportA

        RETURN

LedTable
        addwf   PCL, F
        retlw   B'00111111'         ; mask for digit 0
        retlw   B'00000110'         ; mask for digit 1
        retlw   B'01011011'         ; mask for digit 2
        retlw   B'01001111'         ; mask for digit 3
        retlw   B'01100110'         ; mask for digit 4
        retlw   B'01101101'         ; mask for digit 5
        retlw   B'01111101'         ; mask for digit 6
        retlw   B'00000111'         ; mask for digit 7
        retlw   B'01111111'         ; mask for digit 8
        retlw   B'01101111'         ; mask for digit 9
        retlw   B'00000000'         ; no digit.....


ChkMsdZero                          ; Checking the leading zero
        movf    LO,W                ; Msd digit -> W
        btfss   STATUS,Z            ; = 0 ? skip
        RETURN                      ; If it is, skip
        retlw   .10                 ; If not, go back from 10 to W reg.
```

Realizarea macroului arat• modalitatea de utilizare a macrourilor într-un program. Programul afi•eaz• num•rul ‚21' în 2 digi•i cu **7 segmente**.

LED.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC


;***** Declaring variables *****

        Cblock  0x0C            ; Beginning of RAM
        TempC                   ; Belongs to "LED_Disp2" macro
        LO
        endc

;***** Declaring the hardware *****

        LEDtrisA equ    TRISA
        LEDportA equ    PORTA

        LEDtrisB equ    TRISB
        LEDportB equ    PORTB

;***** Structure of program memory *****

        ORG     0x00            ; Reset vector
        goto    Main

        ORG     0x04            ; Interrupt vector
        goto    ISR             ; Interapt rutina is found
                                ; in 7-seg.inc file

        #include "bank.inc"     ; Assistant files
        #include "7-seg.inc"

Main                            ; Beginning of the program
        LED_Init

        LED_Disp2 0x21          ; Display on two 7-seg. displays
                                ; broj "21"
loop    goto    loop            ; Stay here

      End                       ; End of program
```

Pagina anterioar• Con•inut Pagina urm•toare

Afi•aj LCD

Multe dispozitive cu microcontroler folosesc LCDuri inteligente pentru a afi•a informa•ia vizual•. Urm•torul material se ocup• de conectarea unui afi•aj LDC Hitachi la un microcontroler PIC. Afi•ajele LCD proiectate cu HD44780, modulul pentru LCD fabricat de Hitachi, nu sunt scumpe •i sunt u•or de folosit, •i chiar posibil s• produc• verificarea datelor afi•ate folosind cei 8x80 pixeli ai afi•ajului. Afi•ajele LCD Hitachi con•in un set de caractere ASCII plus simboluri japoneze, grece•ti •i matematice.

A 16x2 line Hitachi HD44780 display

Fiecare dintre cei 640 de pixeli ai afi•ajului trebuie s• poat• fi accesat individual •i aceasta se poate realiza cu un num•r de integrate SMD pentru control montate pe spatele afi•ajului. Aceasta ne salveaz• de o cantitate enorm• de fire •i de un control adecvat astfel încât sunt necesare doar câteva linii pentru a accesa afi•ajul. Putem comunica cu afi•ajul prin intermediul unui bus de date pe 8 bi•i sau de 4 biti. Pentru un bus de 8 biti, afi•ajul are nevoie de o tensiune de alimentare de +5v •i 11 linii I/O. Pentru un bus de 4 bi•i sunt necesare doar liniile de alimentare •i 7 linii. Când afi•ajul LCD nu este pornit liniile de date sunt TRI-STATE, ceea ce înseamn• c• ele sunt în stare de înalt• impedan•• (ca •i cum ar fi deconectate) •i astfel nu interfereaz• cu func•ionabilitatea microcontrolerului când afi•ajul nu este adresat. LCDul necesit• de altfel 3 linii de control de la microcontroler.

Linia **Enable (E)** permite accesul la afi•aj prin intermediul liniilor R/W •i RS. Când aceast• linie este LOW, LCDul este dezactivat •i ignor• semnalele de la R/W •i RS. Când linia (E) este HIGH, LCDul verific• starea celor dou• linii de control •i r•spunde corespunz•tor.

Linia **Read/Write (R/W)** stabile•te direc•ia datelor dintre LCD •i microcontroler. Când linia este LOW, datele sunt scrise în LCD. Când este HIGH, datele sunt citite de la LCD.

Cu ajutorul liniei **Register select (RS)**, LCD interpreteaz• tipul datelor de pe liniile de date. Când este LOW, o instruc•iune este scris• în LCD. Când este HIGH, un caracter este scris în LCD.

Starea logic• a liniilor de control:

E 0 Accesul la LCD dezactivat
1 Accesul la LCD activat

R/W 0 Scrie date în LCD
1 Cite•te date din LCD

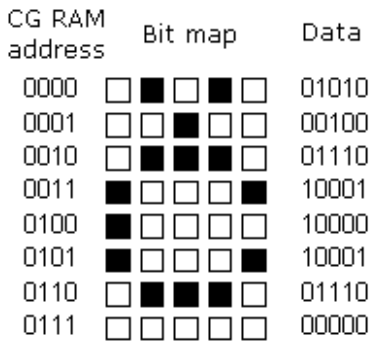
RS 0 Instruc•iuni
1 Caracter

Scrierea datelor în LCD se realizeaz• în câ•iva pa•i:

se seteaz• bitul R/W LOW
se seteaz• bitul RS în 0 sau 1 logic (instruc•iune sau caracter)
se trimit datele c•tre liniile de date (dac• se execut• o scriere)
se seteaz• linia E HIGH
se citesc datele de la liniile de date (dac• se execut• o citire)

Citirea datelor de la LCD se realizeaz• similar, cu deosebirea c• linia de control R/W trebuie s• fie HIGH. Când trimitem un HIGH c•tre LCD, el se va reseta •i va accepta instruc•iuni. Instruc•iunile tipice care sunt transmise c•tre un afi•aj LCD dup• reset sunt: pornirea afi•ajului, activarea cursorului •i scrierea caracterelor de la stânga spre dreapta. În momentul în care un LCD este ini•ializat, el este preg•tit sa primeasc• date sau instruc•iuni. Dac• recep•ioneaz• un caracter, el îl va afi•a •i va muta cursorul un spa•iu la dreapta. Cursorul marcheaz• loca•ia urm•toare unde un caracter va fi afi•at. Când dorim s• scriem un •ir de caractere, mai întâi trebuie s• set•m adresa de start, •i apoi s• trimitem câte un caracter pe rând. Caracterele care pot fi afi•ate pe ecran sunt memorate în memoria video DD RAM (Data Display RAM). Capacitatea memoriei DD RAM este de 80 bytes.

Afi•ajul LCD mai con•ine 64 bytes CG RAM (Character Generator RAM). Aceast• memorie este rezervat• pentru caracterele definite de utilizator. Datele din CG RAM sunt reprezentate sub form• de caractere bitmap de 8 biti. Fiecare caracter ocup• maxim 8 bytes în CG RAM, astfel num•rul total de caractere pe care un utilizator poate s• le defineasc• este 8. Pentru a afi•a caracterul bitmap pe LCD, trebuie setat• adresa CG RAM la punctul de start (de obicei 0) •i apoi s• fie scrise datele în afi•aj. Definirea unui caracter ‚special' este exemplificat• în figur•.



Înainte de a accesa DD RAM, dup• definirea unui caracter special, programul trebuie s• seteze adresa în DD RAM. Orice scriere •i citire a datelor din memoria LCD este realizat• de la ultima adres• care a fost setat•, folosind instruc•iunea set-adress. Odat• ce adresa DD RAM este setat•, un caracter nou va fi afi•at în locul potrivit pe ecran. Pân• acum am discutat opera•ia de scriere •i citire a memoriei unui LCD ca •i cum ar fi o memorie obi•nuit•. Acest lucru nu este adev•rat. Controlerul LCD are nevoie de 40 pân• la 120 microsecunde (us) pentru scriere •i citire. Alte opera•ii pot dura pân• la 5 ms. În acest timp microcontrolerul nu poate accesa LCDul, astfel un program trebuie s• •tie când un LCD este ocupat. Putem rezolva aceasta în dou• metode.

Set DD RAM address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

Set CG RAM address

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	A	A	A	A	A	A

Write in data to RAM

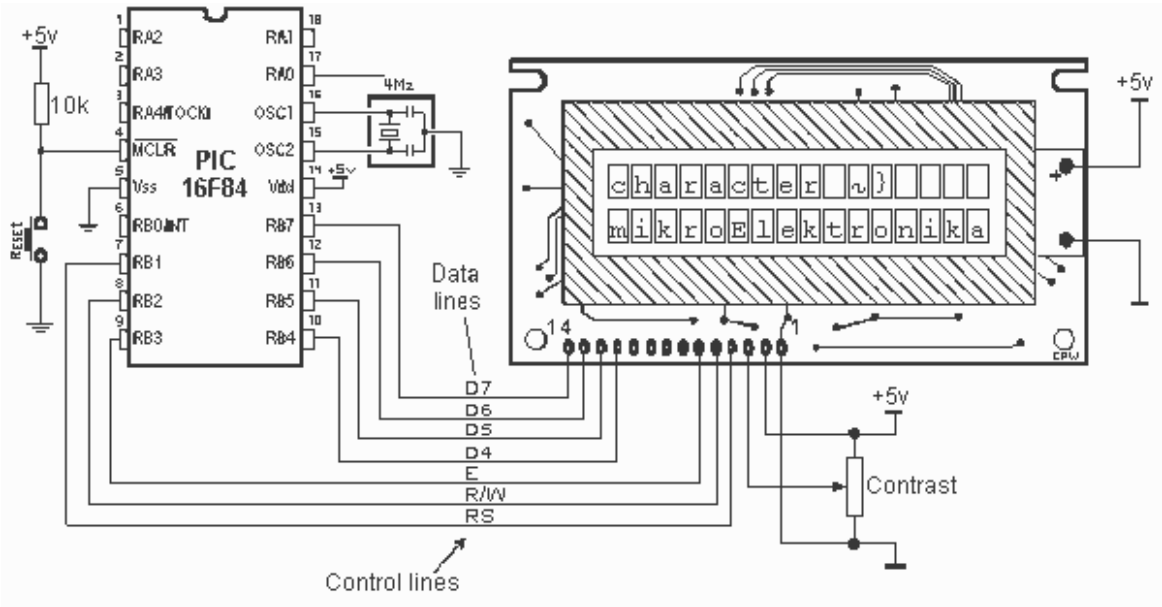
RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	0	D	D	D	D	D	D	D	D

Read data from RAM

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D	D	D	D	D	D	D	D

A=address **D=data**

O metod• este verificarea bitului BUSY de pe linia de date D7. Aceasta nu este cea mai bun• metod• pentru c• LCDul se poate bloca •i programul va sta într-o bucl• infinit• verificând bitul BUSY. O alt• metod• este introducerea unei întârzieri în program. Întârzierea trebuie s• fie destul de lung• pentru ca LCDul s• termine opera•ia în desf••urare. Instruc•iunile pentru scriere •i citire cu memoria LCDului sunt afi•ate mai sus. La început am men•ionat c• avem nevoie de 11 linii I/O pentru a comunica cu un LCD. Oricum, putem comunica cu un LCD printr-un bus de 4 linii. Putem reduce num•rul total de linii de comunica•ie la 7. Schema pentru conectarea printr-un bus de 4 biti este în imaginea de mai jos. În acest exemplu folosim un afi•aj LCD cu 2x16 caractere, denumit LM16x212 fabricat de produc•torul japonez Sharp. Mesajul ‚character' este scris pe prima linie urmat de dou• caractere speciale ‚~' •i ‚}'. Pe a doua linie este scris cuvântul ‚mikroElektronika'.



Conectarea unui afi•aj la un microcontroler

Fi•ierul **LCD.inc** con•ine un grup de macrouri pentru lucrul cu afi•ajele LCD.

LCD.inc

```
;***** Declaring the hardware *****

      RS     equ    1              ; Register Select
      RW     equ    2              ; Read/Write
      EN     equ    3              ; Enable Output / "CLK"

;***** LCD commands *****

      CONSTANT  LCDEM8 = b'00110000'       ; 8 Bit Mode, 2 Lines
      CONSTANT  LCDDZ  = b'10000000'       ; Write 0 u DDRAM
      CONSTANT  LCDEM4 = b'00100000'       ; 4 Bit Mode, 2 Lines

;***** Standard commands for LCD initializatoion *****

      CONSTANT  LCD2L   = b'00101000'      ; Function: 4 bit 2 lines
      CONSTANT  LCDCONT = b'00001100'      ; Display control: Display ON,
                                           ; Cursor OFF, blink OFF
      CONSTANT  LCDSH   = b'00101000'      ; Display mode: AutoInc cursor
                                           ; NoDisplayAutoShift

;***** Standard LCD commands *****

; In order to send one of these commands to LCD, we need to use LCDcmd
; macro, ex. "LCDcmd LCDCLR"

      CONSTANT  LCDCLR  = b'00000001'      ; clears display, resets cursor
      CONSTANT  LCDCH   = b'00000010'      ; cursor home
      CONSTANT  LCDCR   = b'00000110'      ; cursor move direction right
      CONSTANT  LCDCL   = b'00000100'      ; cursor move direction left
      CONSTANT  LCDSL   = b'00011000'      ; shifts display content left
      CONSTANT  LCDSR   = b'00011100'      ; shifts display content right
      CONSTANT  LCDL1   = b'10000000'      ; selects line 1
      CONSTANT  LCDL2   = b'11000000'      ; selects line 2


;***** Macros *****

LCDinit macro
      call    LCD_init                     ; LCD initialization
      endm

LCDchar macro   LCDarg                     ; write out the character on LCD
      movlw   LCDarg
      call    LCDdata
      endm

LCDw    macro
      call    LCDdata
      endm

LCDcmd  macro   LCDcommand                 ; send the command to LCD
      movlw   LCDcommand
      call    LCDcomd
      endm

LCDline macro   line_num
      IF (line_num == 1)
            LCDcmd  LCDL1        ; Start macro with "First Line" instruction
      ELSE
         IF (line  num == 2)
```

```
        ELSE
            IF (line_num == 2)
              LCDcmd  LCDL2       ; Start macro with "Second Line" instruction
            ELSE
            ENDIF
        ENDIF
        endm

LCD_DDAdr macro DDRamAddress
        Local   value = DDRamAddress | b'10000000'     ; Beginning of DDRAM
        IF (DDRamAddress > 0x67)
              ERROR "Wrong DDRAM address in LCD_DDAdr"
        ELSE
              movlw   value
              call    LCDcomd
        ENDIF
        endm

LCD_CGAdr macro CGRamAddress
        Local   value = CGRamAddress | b'01000000'     ; Beginning of CGRAM
        IF (CGRamAddress > b'00111111')
              ERROR "Wrong CGRAM address in LCD_CGAdr"
        ELSE
              movlw   value
              call    LCDcomd
        ENDIF
        endm
```

```
;***** Subprograms *****

LCDcomd clrf    LCDbuf          ; Clear Data flag
        goto    LCDwr

LCDdata clrf    LCDbuf
        bsf     LCDbuf,RS       ; Set Data flag

LCDwr   mowwf   LCDtemp         ; Command/Data in Temp
        andlw   b'11110000'     ; set aside the upper halfbyte
        iorwf   LCDbuf,0        ; set aside Data flag
        mowwf   LCDport         ; send the upper halfbyte to LCD port
        call    LCDclk
        clrf    LCDport
        swapf   LCDtemp,0       ; exchange the upper and lower halfbyte
                                ;  again
        andlw   b'11110000'     ; set aside the lower halfbyte
        iorwf   LCDbuf,0        ; set aside the Data fleg
        mowwf   LCDport         ; send the low half byte to LCD port
        call    LCDclk
        clrf    LCDport
        RETURN

LCDclk  WAITX   0x02, 0x00      ; Enable access to LCD for data and
                                ; commands to be writen in
        bsf     LCDport,EN
        bcf     LCDport,EN
        WAIT    0x02
        RETURN

LCD_init
        clrf    LCDport         ; Prepare LCDport
        BANK1
        clrf    OPTION_REG
        movlw   b'00000000'
        mowwf   LCDtris
        BANK0
        WAIT    0x02
```

```
        BANK0
        WAIT    0x02

        movlw   LCDEM8          ; Start initialization
        movwf   LCDport         ; with "8-bit mode"
        call    LCDclk
        clrf    LCDport
        WAIT    0x02


        movlw   LCDDZ           ; write 0 in DDRAM
        movwf   LCDport
        call    LCDclk
        clrf    LCDport

        movlw   LCDEM4          ; go to 4 bit mode
        movwf   LCDport
        call    LCDclk
        clrf    LCDport

        LCDcmd  LCD2L           ; Function: 2 lines, 4-bit mode
        LCDcmd  LCDCONT         ; Display ON, no cursor
        LCDcmd  LCDSH           ; Mode displaying AutoInc, NoAutoShift
        LCDcmd  LCDCLR          ; Clear display, address counter to zero
        call    LCDspecialChars ; read in characters defined by the user
                                ; to CGRAM
        RETURN

LCDspecialChars                 ; Maximum number of characters
                                ; that user can define is 8

; *** first special Character is "C" at the position 0x00 ***
; *** is called form "LCDchar 0x00" ***

        LCD_CGAdr 0x00          ; Send CGRAM address
        LCDchar b'00001010'     ; Write data to CGRAM address
        LCD_CGAdr 0x01
        LCDchar b'00000100'
        LCD_CGAdr 0x02
        LCDchar b'00001110'
        LCD_CGAdr 0x03
        LCDchar b'00010001'
        LCD_CGAdr 0x04
        LCDchar b'00010000'
        LCD_CGAdr 0x05
        LCDchar b'00010001'
        LCD_CGAdr 0x06
        LCDchar b'00001110'
        LCD_CGAdr 0x07
        LCDchar b'00000000'

; *** first special Character is "C" at the position 0x01 ***
; *** is called form "LCDchar 0x01" ***

        LCD_CGAdr 0x08
        LCDchar b'00000010'
        LCD_CGAdr 0x09
        LCDchar b'00000100'
        LCD_CGAdr 0x0A
        LCDchar b'00001110'
        LCD_CGAdr 0x0B
        LCDchar b'00010001'
        LCD_CGAdr 0x0C
        LCDchar b'00010000'
        LCD_CGAdr 0x0D
        LCDchar b'00010001'
        LCD_CGAdr 0x0E
        LCDchar b'00001110'
        LCD_CGAdr 0x0F
        LCDchar b'00000000'
```

```
LCD_CGAdr 0x0F
LCDchar b'00000000'

LCD_DDAdr 0x00          ; Reset DDRAM

RETURN
```

Macro pentru lucrul cu LCD

Macroul **LCDinit** este utilizat pentru a ini•ializa portul conectat la LCD. LCDul este configurat s• mearg• în modul de 4 bi•i.
Exemplu: LCDinit

LCDchar LCDarg scrie un caracter ASCII. Argumentul este caracterul ASCII.
Exemplu: LCDchar ‚d'

LCDw scrie caracterul din registul W.
Exemplu: movlw ‚p'
LCDw

LCDcmd LCDcommand trimite comenzi.
Exemplu: LCDcmd LCDCH

LCD_DDAdr DDRamAddress seteaz• adresa DD RAM
Exemplu: LCD_DDAdr .3

LCDline line_num seteaz• pozi•ia cursorului la începutul primei sau celei de-a doua linie.
Exemplu: LCDline 2

Când lucr•m cu microcontrolere numerele sunt reprezentate în form• binar•. Din aceast• cauz• ele nu pot fi afi•ate. Pentru aceasta este necesar s• schimb•m numerele dintr-un sistem binar într-un sistem zecimal pentru ca ele s• fie u•or de în•eles. Sursele celor dou• macrouri **LCDval_08** •i **LCDval_16** sunt prezentate mai jos.

Macroul **LCDval_08** realizeaz• conversia unui num•r binar de 8 bi•i într-un num•r zecimal de la 0 la 255 •i îl afi•eaz•. Este necesar s• declar•m urm•toarele variabile în programul principal: TEMP1, TEMP2, LO, LO_TEMP, Bcheck. Num•rul binar de 8 bi•i este în variabila LO. Când macroul este executat, echivalentul zecimal al acestui num•r este afi•at. Zerourile precedente num•rului nu sunt afi•ate.

LCDv8.inc

```
;***** Macros *****

LCDval_08 macro
        call LCDval08
        endm

;***** Subprograms *****

LCDval08
        movfw   LO              ;LO->LO_TEMP
        mowwf   LO_TEMP         ;Initialize zero leading indicator
        clrf    Bcheck

        movlw   d'100'          ;Converts 100-th part of number
        mowwf   TEMP2
        call    VALcnv

        movlw   d'10'           ;Converts 10-th part of number
        mowwf   TEMP2
        call    VALcnv

        movlw   d'1'            ;Converts 1-st part of number
        mowwf   TEMP2
        bsf     Bcheck,0        ;Clear indicator if there aren't leading zeroes

        call    VALcnv
        RETURN

VALcnv
        clrf    TEMP1           ;Counter initialization
```

```
VALcnv
        clrf    TEMP1           ;Counter initialization
        movfw   TEMP2
VALc01
        subwf   LO_TEMP,0       ;Test if LO>99, set carry (negative result)
        skpc
        goto    LCDval2         ;Exit if c is not set  (negative result)

        incf    TEMP1,1         ;Increment counter
        movfw   TEMP2
        subwf   LO_TEMP,1       ;Store TEMP1=TEMP1-100
        bsf     Bcheck,0
        goto    VALc01

        LCDval2 movlw '0'       ;Writing a digit on LCD
        addwf   TEMP1,0         ;Add to counter
        btfss   Bcheck,0
        movlw   ' '             ;Clera leading zero
        LCDw                    ;Write digit on LCD
        RETURN
```

Macroul **LCDval_16** realizeaz• conversia unui num•r binar de 16 bi•i într-un num•r zecimal de la 0 la 65535 •i îl afi•eaz•. Urm•toarele variabile trebuie declarate în programul pricipal: TEMP1, TEMP2, TEMP3, LO, HI, LO_TEMP. HI_TEMP, Bcheck. Num•rul binar de 16 bi•i este în variabilele LO •i HI. Când macroul este executat, echivalentul zecimal al acestui num•r este afi•at. Zerourile precedente num•rului nu sunt afi•ate.

LCDv16.inc

```
;***** Macros *****

LCDval_16 macro
      call LCDval16
      endm

;***** Subprograms *****

LCDval16
      movfw   LO              ;LO->LO_TEMP
      mowwf   LO_TEMP         ;SUB-LO
      movfw   HI              ;SUB-HI
      mowwf   HI_TEMP         ;HI->HI_TEMP
      clrf    Bcheck          ;Initialization leading zero indicator

      movlw   b'00010000'     ;Convert 10000-th part of number
      mowwf   TEMP2           ;SUB-LO
      movlw   b'00100111'
      mowwf   TEMP3           ;SUB-HI
      call    VALcnv

      movlw   b'11101000'     ;Convert 1000-th part of number
      mowwf   TEMP2           ;SUB-LO
      movlw   b'00000011'
      mowwf   TEMP3           ;SUB-HI
      call    VALcnv

      movlw   b'01100100'     ;Convert 100-th part of number
      mowwf   TEMP2           ;SUB-LO
      clrf    TEMP3           ;SUB-HI if zero
      call    VALcnv

      movlw   b'00001010'     ;Convert 10-th part of number
      mowwf   TEMP2           ;SUB-LO
      clrf    TEMP3
      call    VALcnv
```

```
        call    VALcnv

        movlw   b'00000001'     ;Convert 1st part of number
        movwf   TEMP2           ;SUB-LO
        clrf    TEMP3
        bsf     Bcheck,0        ;Clear indicator if there aren't leading zeroes

        call    VALcnv
        RETURN

VALcnv                          ;Counter initialization
        clrf    TEMP1
Vcnv1
        movfw   TEMP3
        subwf   HI_TEMP,0
        skpc                    ;Skip if HI>=0
        goto    LCDval2         ;Exit if not
        bnz     Vcnv2

        movfw   TEMP2
        subwf   LO_TEMP,0
        skpc                    ;Skip if LO>=0
        goto    LCDval2         ;Exit if not

Vcnv2
        movfw   TEMP3
        subwf   HI_TEMP,1       ;HI=HI-TEMP3
        movfw   TEMP2
        subwf   LO_TEMP,1       ;LO=LO-TEMP2
        skpc                    ;Skip if LO>=0
        decf    HI_TEMP,1       ;Decrement HI
        incf    TEMP1,1         ;Increment counter
        bsf     Bcheck,0
        goto    Vcnv1

LCDval2 movlw   '0'             ;Writting a number at LCD
        addwf   TEMP1,0         ;Add offset to counter
        btfss   Bcheck,0
        movlw   ' '
        LCDw
        RETURN
```

Programul principal este o demontra•ie pentru utilizarea afi•ajelor LCD •i desenarea de noi caractere. La începutul programului, trebuie s• declar•m variabilele **LCDbuf** •i **LCDtemp** folosite în subrutinele pentru LCD, cât •i portul microcontrolerului conectat la LCD. Programul scrie mesajul ‚characters:' pe prima linie urmat de dou• caractere speciale ‚~' •i ‚}'. Pe cea de-a doua linie este afi•at ‚mikroElektronika'.

LCD.asm

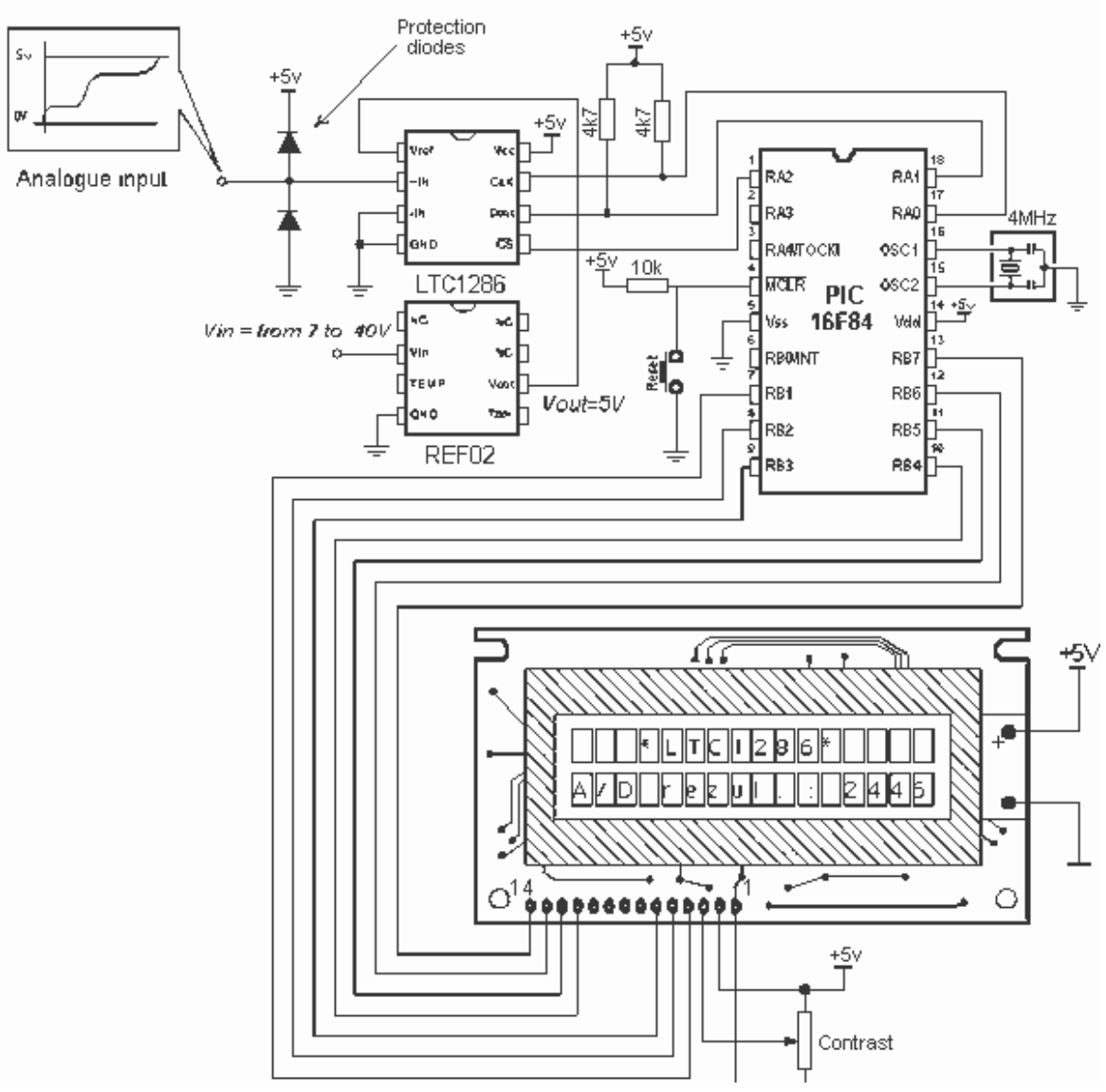
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

        Cblock 0x0C             ; Beginning of RAM
        LCDbuf                  ; Belongs to 'LCDxxx' macros
        LCDtemp
```

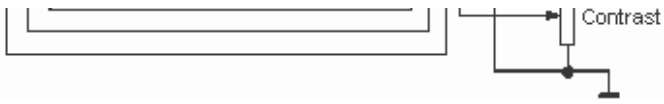
```
        LCDtemp
        WCYCLE              ; Belongs to 'WAITX' macro
        PRESCwait
        Pointer             ; Pointer on characters in message
        endc

;***** Declaring the hardware *****

        LCDtris equ     TRISB
        LCDport equ    PORTB

;***** Structure of program memory *****

        ORG    0x00    ; Reset vector
        goto   Main

        ORG    0x04    ; Interrupt vector
        goto    Main    ; No interrupt routine

Messages                ; Beginning of the messages

      mowf   PCL

                        ; Display messages

Message1 dt "mikRoEleKtrOnIkA"
Message2 dt "bla, bla"
Message3 dt "example"

END_messages                        ; End of messages

        #include "bank.inc"       ; Assistant files
        #include "wait.inc"
        #include "lcd.inc"
        #include "print.inc"

Main                    ; Beginning of the program

        bcf PORTB,2

        LCDinit         ; LCD initialization

        LCDchar 'C'     ; Display character on LCD
        LCDchar 'a'
        LCDchar 'r'
        LCDchar 'a'
        LCDchar 'c'
        LCDchar 't'
        LCDchar 'e'
        LCDchar 'r'
        LCDchar 's'
        LCDchar ':'
        LCDchar ' '

        LCDchar         0x00    ; Display special characters
        LCDchar         0x01

        LCDline 2       ; Second line

; Display message1 on LCD

        PRINT   Messages, Message1, Message2, Pointer, LCDw

Loop    goto    Loop    ; Infinite loop

        End             ; End of program
```

Pagina anterioar• Con•inut Pagina urm•toare

Convertor analog – digital de 12 bi•i

Deoarece totul în lumea microcontrolerelor este reprezentat cu 0 •i 1, ce facem cu un semnal care este 0.5 sau 0.77? Aproape toat• lumea exterioar• unui calculator consist• din semnale analogice. În afar• de vorbire •i muzic•, sunt multe cantit••i care trebuie cuantificate într-un calculator. Umiditatea, temperatura, presiunea aerului, culoarea, **turbidity**, •i concentra•ia metanului sunt doar o parte. R•spunsul este s• lu•m un num•r de linii digitale pe care s• le combin•m asfel încât ele s• exprime o valoare analogic•. O valoare analogic• este orice valoare dintre 0 •i 1. O pute•i numi „valoare frac•ional•". Toate cantit••ile de mai sus trebuie s• fie convertite la o valoare cuprins• între 0 •i 1 astfel s• fie utilizat• într-un calculator. Acesta este conceptul cel mai r•spândit. El devine ceva mai pu•in complex în aplica•ii. Dac• lu•m 8 linii •i le aranj•m astfel încât s• accepte valori binare, rezultatul total va fi 256 (acesta este ob•inut de o num•rare pân• la 255 plus valoarea 0). Dac• conect•m aceste 8 linii într-o „cutie neagr•", vor fi numite linii de ie•ire •i astfel trebuie s•-i furniz•m o singur• linie de intrare. Cu acest aranjament putem detecta pân• la 255 increment•ri între „0" •i „1". Aceast• cutie neagr• este denumit• CONVERTOR •i pentru c• noi convertim din **A**nalog în **D**igital, convertorul se nume•te convertor **AD** sau **ADC (A**nalog to **D**igital **C**onvertor). Convertoarele AD pot fi clasificate dup• parametrii diferi•i. Cei mai importan•i parametri sunt precizia •i modul de transfer al datelor. Referitor la precizie, domeniul este: 8 bi•i, 10 bi•i, 12 bi•i, 14 bi•i, 16 bi•i. Deoarece conversia pe 12 bi•i este un standard industrial, exemplul de mai jos a fost realizat cu ajutorul unui convertor pe 12 bi•i. Cel•lalt parametru important este modul prin care datele sunt transferate în microcontroler. Poate fi paralel sau serial. Transmisia paralel• este mai rapid•. Oricum, aceste convertoare sunt deobicei mai scumpe. Transmisia serial• este mai lent•, dar considerând pre•ul sc•zut •i num•rul redus de linii de intrare la un microcontroler, este transmisia preferat• pentru multe aplica•ii. Semnalele analogice pot dep••i limitele de intrare ale unui ADC. Aceast lucru poate duce la stricarea convertorului. Pentru a proteja intrarea sunt conectate dou• diode, dup• cum se observ• în schem•. Acestea vor proteja de tensiuni de peste 5V •i sub 0V. În exemplul nostru am folosit LTC1286, un ADC pe 12 biti (Linear Technology). Convertorul este conectat cu microcontrolerul prin intermediul a trei linii: data, clock •i CS (Chip Select). Linia CS este folosit• pentru a selecta un dispozitiv de intrare ca s• fie posibil s• fie conectate •i alte dispozitive de intrare (ex. registru de deplasare de intrare, registru de deplasare de ie•ire, ADC serial) pe acelea•i linii ale microcontrolerului. Circuitul de mai jos arat• cum s• conect•m un ADC, tensiune de referin•• •i un afi•aj LCD la un microcontroler. Afi•ajul LCD a fost ad•ugat pentru a vizualiza rezultatele conversiei AD.





Conectarea unui convertor AD cu tensiune de referin•• la un microcontroler

Macroul folosit în acest exemplu este LCD86 •i este localizat în fi•ierul LTC1286.inc.

LTC1286.inc

```
LTC86   macro Var_LO, Var_HI, Var

        Local   Loop            ; Local label
        Local   Loop1

        clrf    Var_LO          ; Erase data buffer
        clrf    Var_HI

        movlw   .4
        mowwf   Var             ; Counter initialization

        bcf     CS              ; Enable AD coverter

        call    CLK             ; Enable Dout line
        call    CLK             ;         /
        call    CLK             ; NULL bit

Loop    rlf     Var_HI,f        ; RRotate 'Var_HI' one space to left

        btfss   Data            ; is Dout line = '1' ?
        bcf     Var_HI,0        ;if not, clear bit '0' in Var_HI variable
        btfsc   Data            ; is Dout line = '0' ?
        bsf     Var_HI,0        ; if not, set bit '0' in Var_HI variable

        call    CLK

        decfsz  Var,f           ; Are 4 bits received ?
        goto    Loop            ; If not, repeat

        movlw   .8
        mowwf   Var             ; Counter initialization

Loop1   rlf     Var_LO,f        ; Rotate 'Var_LO' one space to left

        btfss   Data            ; is Dout line = '1' ?
        bcf     Var_LO,0        ; if not, clear bit '0' in Var_LO variable
        btfsc   Data            ; is Dout line = '0' ?
        bsf     Var_LO,0        ; if not, set bit '0' in Var_LO variable

        call    CLK

        decfsz  Var,f           ; Are 8 bits received ?
        goto    Loop1           ; If not, repeat

        bsf     CS              ; Disable AD converter

        endm

CLK
        bsf     Clock           ; Make one clock
        nop
        nop
        nop
    bcf     Clock

        return
```

Macroul LTC86 are trei argumente:

LTC86 macro Var_LO, Var_HI, Var

Variabila **Var_LO** este unde se stocheaz• rezultatul conversiei byte-ului mai pu•in semnificativ.
Variabila **Var_HI** este unde se stocheaz• rezultatul conversiei byte-ului cel mai semnificativ.
Var este un contor bucl•.

Exemplu: LTC86 LO, HI, Count

Cei patru bi•i ai celei mai mari valori sunt în variabila **HI**, iar primii 8 bi•i ai rezultatului conversiei sunt în variabila **LO**. **Count** este o variabil• de **asisten••** care num•r• trecerile prin bucle.

Urm•torul exemplu arat• cum macrourile sunt folosite în program. Programul cite•te valoarea provenit• de la ADC •i o afi•eaz• pe LDC. Rezultatul este dat în **quantums**. Ex: pentru 0V rezultatul este 0, iar pentru 5V este 4095.

LTC1286.asm

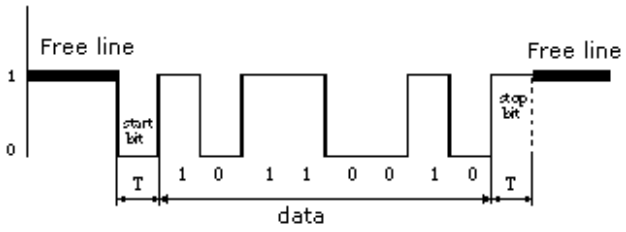
```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

         __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC

;***** Declaring variables *****

      Cblock  0x0C              ; Beginning of RAM
         LCDbuf                 ; Belongs to 'LCDxxx' macros
         LCDtemp
         WCYCLE                 ; Belongs to 'WAITX' macro
         PRESCwait

         TEMP1                  ; Belongs to "LCDval_16" macro
         TEMP2
         TEMP3
         LO                     ; LO_Data buffer
         HI                     ; HI_Data buffer
         LO_TEMP
         HI_TEMP
         Bcheck

         Count                  ; Belongs to "LTC86" macro
         Pointer                ; Pointer on characters in message
   endc

;***** Declaring the hardware *****

         #define  Data     PORTA,0
         #define  Clock    PORTA,1
         #define  CS       PORTA,2

         LCDtris equ      TRISB
         LCDport equ      PORTB

;***** Structure of program memory *****

         ORG    0x00            ; Reset vector
         goto   Main

         ORG    0x04            ; Interrupt vector
         goto   Main            ; No interrupt routine

Messages                        ; Beginning of the messages
```

```
Messages                    ; Beginning of the messages

   mowwf   PCL

                            ; Display messages

Message0 dt "* LTC1286 *"
Message1 dt "A/D rezul.:"

END_messages                ; End of messages


        #include "bank.inc"              ; Assistant files
        #include "ltc1286.inc"
        #include "wait.inc"
        #include "lcd.inc"
        #include "lcdv16.inc"
        #include "print.inc"

Main                                    ; Beginning of the program
        BANK1
        movlw   0xf1                    ; Port A initialization
        mowwf   TRISA                   ; TRISA <- 0xf1
        BANK0

        LCDinit                         ; LCD initialization

        clrf    PORTA                   ; PORTA <- 0x00

        LCD_DDAdr .3

        PRINT   Messages, Message0, Message1, Pointer, LCDw
        ; Read value on AD coverter input

Loop
   LTC86        LO, HI, Count

                                        ; Display AD result
        call    Out                     ; on LCD display

        goto    Loop                    ; Repeat loop

Out
        LCDline 2                       ; Second line

        PRINT   Messages, Message1, END_messages, Pointer, LCDw

        LCDval_16

        return

   End          ; End of program
```

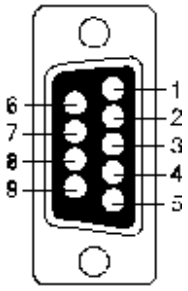
Pagina anterioar• Con•inut Pagina urm•toare

Comunica•ia serial•

SCI este o abrevire pentru Serial Communication Interface, •i ca un subsistem special exist• la majoritatea microcontrolerelor. Când nu este disponibil, cum ar fi în cazul lui PIC16F84, poate fi creat în software.



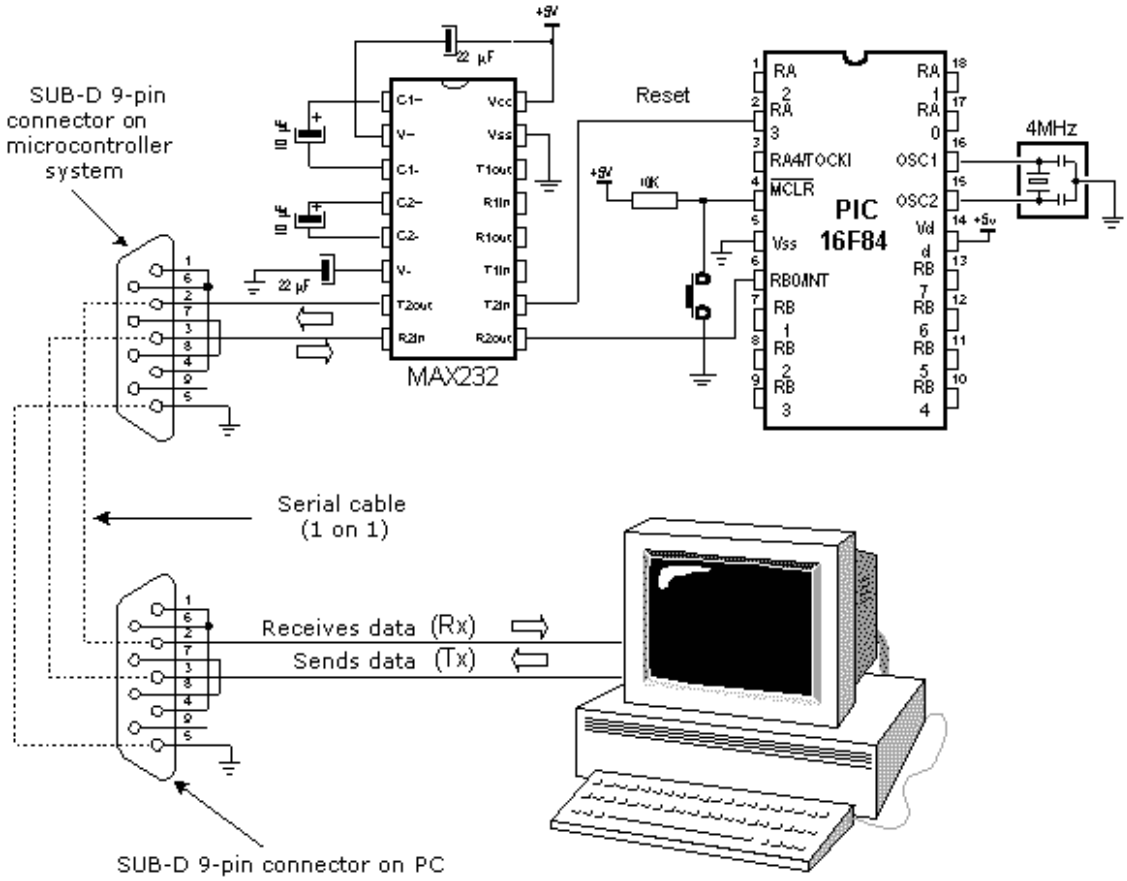
Ca •i în cazul comunica•iei hardware, folosim formatul standard NRZ (Non Return to Zero) cunoscut ca 8 (9)-N-1 sau 8 sau 9 bi•i de date, f•r• paritate •i cu un bit de stop. **Linia liber•** este definit• starea **unu logic**. Startul transmisiei – **Bitul de Start**, are starea **zero logic**. Dup• bi•ii de date care urmeaz• bitului de start (primul bit este cel mai pu•in semnificativ bit) urmeaz• un **Bit de Stop** care are starea **unu logic**. Durata bitului de stop ‚T' depinde de viteza transmisiei •i este ajustat dup• necesit••ile transmisiei. Pentru o vitez• de transmisie de 9600 baud, T este 104us.



1. CD (Carrier Detect)
2. RXD (Receive Data)
3. TXD (Transmit Data)
4. DTR (Data terminal Ready)
5. GND (Ground)
6. DSR (Data Set Ready)
7. RTS (Request To Send)
8. CTS (Clear To Send)
9. RI (Ring Indicator)

Descrierea pinilor ai unui conector RS232

Pentru a conecta un microcontroler la un port serial al unui calculator PC, trebuie s• ajust•m nivelul semnalelor pentru ca s• aib• loc comunica•ia. Nivelul semnalului la un PC este -10V pentru zero logic •i +10V pentru unu logic. Din cauz• c• nivelul semnalului la un microcontroler este de +5V pentru unu logic •i 0V pentru zero logic, avem nevoie de un stadiu intermediar care s• realizeze conversia nivelurilor. Un integrat special proiectat pentru aceast• sarcin• este MAX232. Schema interfe•ei este în diagrama de mai jos:



Conectarea unui microcontroler la un PC prin intermediul unei interfe•e realizate cu MAX232.

Fi•ierul RS232.inc con•ine un grup de macrouri folosite pentru comunica•ia serial•.

RS232.inc

```
;***** Declaring the hardware *****

        #define  RXport PORTB,0
        #define  RXtris TRISB,0

;***** Declaring constants *****

        CONSTANT LF  =      d'10'    ; Line Feed
        CONSTANT CR  =      d'13'    ; Carriage Return
        CONSTANT TAB =      d'9'     ; Tabulator
        CONSTANT BS  =      d'8'     ; Backspace

;***** Macros *****

RS232init macro
        call    RS_init
        endm

SEND    macro   S_string        ; "send'x' ", sending ASCII caracters
        movlw   S_string
        call    SENDsub
        endm

SENDw macro
        call    SENDsub
        endm

RECEIVE         macro
```

```
RECEIVE     macro
            call    RECsub
            endm

;***** Subprograms *****

RS_init bcf     TXport
        BANK1
        clrf    OPTION_REG
        bcf     TXtris              ;Tx pin
        bsf     RXtris              ;Rx input with pull-up
        BANK0
        bsf     TXport              ;The beginning state onTX line :logic "1"
        movlw   b'10010000'
        movwf   INTCON              ;Enable RB0 -interrupt
        RETURN

SENDsub     movwf   TXD             ;Tx data register
        bcf     TXport              ;Start bit
        movlw   0x08
        movwf   RS_TEMP1            ;The number of bit to send 9600-8-N-1
        call    S_Wait
SENDa btfsc     TXD,0               ;Send LSB first
        goto    SENDb
        bcf     TXport
        goto    SENDc
SENDb bsf       TXport
SENDc rrf       TXD,1
        call    S_Wait
     decfsz  RS_TEMP1,1
        goto    SENDa
        goto    SENDd
SENDd bsf       TXport              ;Stop bit
        call    S_Wait
        call    S_Wait              ;Re-sinchronization
        RETURN

S_Wait movlw    0x1E                ;Pause between  two bits
        movwf   RS_TEMP2            ;9600 baud => 104uS in sending
        goto    X_Wait

Rs_Wait     movlw   0x0C            ;Pause 52uS for start bit
        movwf   RS_TEMP2            ;in receiving
        goto    X_Wait

R_Wait movlw    0x1D                ;Pause between two bits in receiving
         movwf   RS_TEMP2
        goto    X_Wait

X_Wait  decfsz  RS_TEMP2,1
        goto    X_Wait
        RETURN

RECsub      call    Rs_Wait
        btfsc   RXport
        goto    REENTRY             ;Not start bit
        movlw   0x08
         movwf   RS_TEMP1           ;The nimber receiving bits 9600-8-N-1
        goto    RECa                ;Pause
RECa    call    R_Wait
        btfss   RXport
        goto    RECb
        bsf     RXD,0x07
        goto    RECc
RECb    bcf     RXD,0x07
RECc    decfsz  RS_TEMP1,0          ;Skip if RS_TEMP=1
        rrf     RXD,1               ;Repeat this seven times
        decfsz  RS_TEMP1,1
        goto    RECa
```

```
            decisz  RS_TEMP1,f
            goto    RECa
            call    R_Wait
            btfss   RXport              ;Check if stop bit
            clrf    RXD
            RETURN

REENTRY     clrf    RXD             ;Unvalid data
            goto    ISRend
```

Utilizarea macroului:

Macroul **RS232init** este folosit pentru ini•ializarea pinului RB0 •i liniei pentru transmisia de date (pinul TX).
Exemplu: RS232init

SEND S_string trimite un caracter ASCII. Argumentul este este semnul ASCII.
Exemplu: SEND ‚g'

SENDw trimite data din registrul W.
Exemplu: movlw ‚t'
SENDw

Macroul **RECEICE** este o subrutin• de tratare a unei întreruperi care recep•ioneaz• datele pentru RS232 •i le memoreaz• în registrul RXD.

Exemplu:

```
ORG    0x04
       goto  ISR
ISR    bcf   INTCON,GIE
       btfsc INTCON,GIE
       goto  ISR
       RECEIVE
          :
          :
ISRend bcf   INTCON,INTF
       RETFIE
```

La începutul programului principal, trebuie s• declar•m variabilele RS_TEMP1, RE_TEMP2, TXD, RXD •i pinul TX al microcontrolerului. Dup• resetarea microcontrolerului programul trimite un mesaj de întâmpinare c•tre calculatorul PC: **$ PIC16F84 on line $**, •i apoi este gata de a recep•iona date de pe linia RX. Putem transmite •i recep•iona date de la calculatorul PC prin acela•i program de comunica•ie. Când microcontrolerul recep•ioneaz• datele, va transmite un mesaj: Character received from PIC16F84: x, pentru confirmarea succesului recep•ion•rii.

Programul principal:

RS232.asm

```
;***** Declaring and configuring a microcontroller *****

      PROCESSOR 16f84
      #include "p16f84.inc"

        __CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC


;***** Declaring variables *****

        Cblock  0x0C        ; Beginning of RAM
        RS_TEMP1            ; Belongs to 'RS232' functions
        RS_TEMP2
        TXD
        RXD
        Pointer             ; Pointer to message's caracters
        endc
```

```
        Pointer             ; Pointer to message's caracters
        endc

;***** Deklaracija hardvera *****

        #define  TXport PORTA,3
        #define  TXtris TRISA,3

        LCDtris equ TRISB
        LCDport         equ PORTB

;***** Structure of program memory *****

        ORG    0x00             ; Reset vector
        goto   Main

        ORG    0x04             ; Interrupt vector
        goto   ISR              ; Jump on interrupt routine

Messages

   movwf        PCL     ; Messages to be shown at LCD
                        ; sent by RS232

Message0 dt "Received character from PIC16F84: "
Message1 dt "$ PIC16F84 on line $"

END_messages                    ; End of messages


        #include "bank.inc"         ; Assistant files
        #include "rs232.inc"
        #include "print.inc"

;***** Interrupt routine *****

ISR     bcf     INTCON,GIE          ; Disable all interrupts
        btfsc   INTCON,GIE          ; Check if disabled
        goto    ISR

        RECEIVE                 ; Store received data in RX variable


        SEND TAB
                                ; Send massage 0 trough RS232 line

        PRINT   Messages, Message0, Message1, Pointer, SENDw

        movfw   RXD     ; Return back received data as confirmation
        SENDw           ; of successful receiving

        SEND    CR      ; Carriage Return
        SEND    LF      ; Line Feed
        SEND    LF

ISRend bcf INTCON,INTF ; Clear interrupt fleg RB0/INT
        RETFIE          ; Enable all interrupts

Main                    ; Start of the program

        RS232init       ; RS232 initialization

                        ; Send message 1

        PRINT   Messages, Message1, END_messages, Pointer, SENDw

        SEND    CR      ; Carriage Return
        SEND    LF      ; Line Feed
        SEND    LF
```

```
        SEND  LF

Loop    goto  Loop   ; Infinite loop
        End          ; End of program
```

Pagina anterioar• Con•inut Pagina urm•toare