3. Structuri de program

Pentru a realiza programe scalabile şi lizibile, codul şi datele trebuie grupate. *Macrourile* şi *subrutinele* joacă un rol important în gruparea codului, datele fiind grupate în structuri denumite *tablouri*.

3.1 Macrouri şi subrutine.

Structurarea codului folosind macrouri şi subrutine reprezintă o practică bună de programare, aceste structuri formând baza unor limbaje consacrate precum 'C' şi 'Pascal'. În limbaje de asamblare, funcţiile nu sunt atât de vizibile precum în limbajele amintite însă numai prin stăpânirea acestor structuri se vor putea dezvolta programe scalabile.

3.1.1 Macrouri.

În limbajul 'C', *macrourile* (cunoscute şi sub denumirea de *macrodefiniţii* sau *macroinstrucţiuni*) sunt definiţii care înlocuiesc constante, variabile sau expresii, fiind declarate cu ajutorul directivei *#define*. În cadrul asamblorului considerat, macrourile sunt secvenţe de instrucţiuni care înlocuiesc un apel al macroului.

Sintaxa de definire a unui macro:

```
etichetă macro [arg1, arg2, …]
     [instrucţiune 1]
     [instrucţiune 2]
     […]
endm
```

Exemplul 3.1. Definirea unui macro de stocare a două valori *val1* şi *val2* parametrice la locaţiile de memorie având adresele respectiv 0 şi 15.

```
ram_0  EQU  d'0'  ;Constanta asamblor cu valoarea 0
ram_15 EQU  0x0F  ;Constanta asamblor cu valoarea 15

;Definire macro Stocare
Stocare macro val1, val2
     MOVLW val1
     MOVWF ram_0  ;Adresa RAM 0
     MOVLW val2
     MOVWF ram_15 ;Adresa RAM 15
endm

;Apelul macroului
     Stocare h'3F', h'F0'
```

Ceea ce este specific macrourilor este faptul că fiecare apel al acestora echivalează cu o inserare a setului de instrucţiuni din corpul macroului la adresa de apel. Folosirea acestor structuri se pretează de regulă în situaţii în care acestea nu conţin multe instrucţiuni şi integrarea altor structuri (cum ar fi cele cu salt) ar duce la o scădere a performanţelor sistemului.

Macrourile oferă o claritate în plus programelor întrucât chiar şi simple seturi de instrucțiuni pot fi înlocuite cu o expresie familiară programatorului. Un asemenea exemplu este construcția din exemplul 2.

Exemplul 3.2. Macro pentru selectarea BANK-ului 2 de memorie.

```
;Definire macro BANK2
BANK2 macro
     MOVLW 0x02          ;Selectarea bank-ului dorit se face
     MOVWF BSR           ;prin stocarea valorii corespunzătoare
endm                     ;în registrul special BSR

;Apelul macroului
     BANK2
```

Exemplul 3.3. Definirea unui macro de adunare a două numere folosind o locatie de memorie temporară situată la adresa 0. Rezultatul se va stoca în registrul de lucru w.

```
ram_temp  EQU  0x00

;Definire macro Adunare
Adunare macro val1, val2
     MOVLW val1
     MOVWF ram_temp
     MOVLW val2
     ADDWF ram_temp, 0
endm

;Apelul macroului
     Adunare 0x23, 0x43
```

Avantaje:

- Claritatea codului la scriere prin posibilitatea redenumirii unor seturi de instrucțiuni folosind expresii familiare programatorului.
- Reducerea dimensiunii codului la scriere prin definirea unui macro pentru seturi de instrucțiuni care se repetă.
- Eficiență ridicată la execuția setului de instrucțiuni inclus în macro întrucât acestea sunt inserate în cod la compilare, nemaifiind necesară efectuarea unui salt.

Dezavantaje:

- Pot introduce erori subtile de programare cum ar fi modificarea unor zone de memorie nedorite datorate ignorării conținutului macroului (poate apare cu uşurință în cadrul unor proiecte mari unde se folosesc macrouri scrise de alte persoane).
- Creşterea în dimensiuni a codului compilat prin folosirea unor apeluri de macro de multe ori. Această problemă este deosebit de supărătoare în cazul în care se folosesc macrouri foarte mari (de exemplu, pentru realizarea unei întârzieri de 100 microsecunde la 20 Mhz trebuie incluse 500 de instrucțiuni NOP).

3.1.2 Subrutine.

Pentru a executa aceleaşi instrucţiuni de mai multe ori fără a recurge la reinserarea codului, se pot folosi subrutine. În limbajul 'C' subrutinele se regăsesc sub forma unor funcţii, având un nume, tip returnat şi un set de parametrii. În cadrul limbajelor de asamblare, subrutinele iau forma unor etichete opţionale şi instrucţiuni de revenire la o adresă de salt.

Sintaxa de definire a unei subrutine:

```
[etichetă:]
      [instrucţiune 1]
      [instrucţiune 2]
      […]
      RETURN | RETLW | RETFIE
```

Exemplul 3.4. Subrutină de setare PORTC , în comentariile aferente fiind precizate şi adresele instrucţiunilor.

```
     GOTO Main          ;0000 Salt peste subrutină

Setare_Port:
     MOVLW 0xF0         ;0004
     MOVWF TRISC        ;0006
     RETURN             ;0008 Revenire din subrutină

Main:
     CALL Setare_Port   ;000A Apelul subrutinei
```

Instrucţiunea de apel al unei subrutine este CALL, urmată de o etichetă reprezentând adresa la care se face salt. În exemplul anterior, adresa la care se află eticheta `Setare_Port` este `0004`, adică adresa primei instrucţiuni din subrutină.

Pentru ca la apelul unei subrutine microprocesorul să poată realiza un salt la o anumită adresă unde să execute un set de instrucţiuni, trebuie să se parcurgă următorii paşi:

(1) Se salvează pe stivă adresa de revenire (valoarea numărătorului de program)

STACK.push(PC)

(2) Se încarcă în numărătorul de program adresa etichetei

Adresa(*etichetă*) → PC

(3) Se execută instrucţiunile din subrutină.

Prin paşii enumeraţi se salvează adresa de revenire pe stivă iar "saltul" la adresa indicată de etichetă echivalează de fapt cu o încărcare a unei adrese în numărătorul de program.

Aceşti paşi fac parte din execuţia unei instrucţiuni `CALL`, iar din cauza saltului, timpul de execuţie este de două ori mai mare decât al unor instrucţiuni simple de manipulare a regiştrilor (cum ar fi `MOVLW`, `MOVWF`, etc.).

Revenirea dintr-o subrutină se face la întâlnirea unei instrucţiuni `RETURN` sau `RETLW`. Pentru revenirea dintr-o rutină de tratare a unei întreruperi se execută o instrucţiune `RETFIE` (despre această instrucţiune se va vorbi mai mult în cadrul laboratorului de *Întreruperi*). Paşii executaţi:

(1) Se stochează în numărătorul de program vârful stivei (care la o execuție corectă este de fapt adresa de revenire salvată)

STACK.pop() → PC

(2) Se execută instrucțiunile existente după `CALL`

Avantaje:

- Ca şi în cazul folosirii macrourilor, includerea unor subrutine în cod poate duce la o creştere a clarității programului.
- Subrutinele sunt des folosite la distribuirea codului într-o echipă ce include mai mulți programatori, împreună cu macrourile făcând posibilă modularizarea.
- Însă cel mai important avantaj îl constituie posibilitatea execuției aceleiaşi secvențe de cod din puncte diferite de program folosind salturi fără re-includerea codului, aşa cum este cazul macrourilor.

Dezavantaje:

- Datorită saltului efectuat la adresa de început a subrutinei şi saltului de revenire, va rezulta un timp de execuție mai mare.

3.2 Tablouri.

Pentru gruparea datelor în structuri denumite "tablouri", se poate folosi una din metodele următoare: instrucțiuni microprocesor, adresarea indirectă, instrucțiuni tabelare.

3.2.1 Implementarea tablourilor folosind instrucțiuni microprocesor.

Această metodă presupune stocarea valorilor tabloului în memoria program, fiind folosită în deosebi în situațiile în care valorile respective nu se doresc a fi modificate. Cu toate că valorile tabloului nu se pot modifica[1], acestă metodă are avantajul că permite stocarea unui număr mult mai mare de valori decât ar fi posibil prin folosirea RAM-ului.

Fiindcă se folosesc instrucțiuni microprocesor în realizarea tablourilor, programatorul poate implementa o rutină de accesare a valorilor după bunul său plac. O metodă foarte populară care poate fi folosită atât la seria 16F cât şi la seria 18F (se poate folosi de fapt la toate microcontrollere care permit manipularea numărătorului de program) implică instrucțiuni de manipulare a numărătorului de program şi instrucțiunea `RETLW`.

Exemplul 3.5. Rutină de simulare al unui tablou cu 8 elemente aleatoare. Accesul la valori se realizează indexat folosind ca "variabilă index", registrul de lucru `W`.

```
;Rutină de simulare al unui tablou pentru microcontrollere ;PIC cu
instrucțiuni RETLW pe 2 octeți (18Fxxxx) (a) şi pe 1 ;octet (16Fxxxx)
(b)
```

[1] Modificarea memoriei program prin intermediul unui program este posibilă doar dacă există instrucțiuni speciale. Altfel, această modificare se poate realiza doar printr folosirea unui circuit extern, fapt care echivalează cu inscripționarea microcontrollerului. Aceste "instrucțiuni speciale" fac parte din setul de instrucțiuni normale la microcontrollere PIC din seria 18Fxxxx.

```
Tabela_Test:
     MULLW  D'2'
     MOVF   PRODL, W
     MOVFF  PCL, 0x00
     ADDWF  PCL, F
     RETLW  D’101’
     RETLW  D’115’
     RETLW  D’90’
     RETLW  D’83’
     RETLW  D’85’
     RETLW  D’99’
     RETLW  D’150’
     RETLW D’149’
```

(a)

```
Tabela_Test:
     MOVWF 0X20
     MOVLW high $
     MOVWF PCLATH
     MOVF  0X20, W
     ADDWF  PCL, F
     RETLW  D’101’
     RETLW  D’115’
     RETLW  D’90’
     RETLW  D’83’
     RETLW  D’85’
     RETLW  D’99’
     RETLW  D’150’
     RETLW D’149’
```

(b)

```
;Accesarea valorii cu index-ul 3 (valorile indecşilor pornesc de la 0)
     MOVLW  D’3’
     CALL   Tabela_Test
;În acest moment, în registrul W se află valoarea D’83’
```

În exemplul 3.5 se poate observa faptul că deoarece întregul tablou reprezintă de fapt un set de instrucțiuni, acesta va fi stocat în memoria program la inscripționarea microcontrollerului. În cazul rutinei (a), prezența instrucțiunii de multiplicare este justificată de numărul de octeți alocați reprezentării instrucțiunii `RETLW`. Astfel, pentru a returna o valoare de la un anumit index, mai întâi trebuie înmulțit index-ul respectiv cu 2 (rezultatul fiind stocat în `PRODL:PRODH`) după care rezultatul este adunat la numărătorul de program, astfel, instrucțiunea care va fi executată după `ADDWF PCL, F` va avea adresa:

Adresă_Instr_Următoare = $ + INDEX * 2,

unde $ reprezintă adresa instrucțiunii curente (`ADDWF PCL, F`).

În acelaşi exemplu, instrucțiunea `MOVFF PCL, 0x00` stochează valoarea inferioară a numărătorului de program la adresa RAM 0x00. Prezența acestei instrucțiuni se justifică prin problemele pe care le introduce structura numărătorului de program pe 21 de biți.

Astfel, cei 21 de biți sunt împărțiți în 3 regiştrii: `PCL`, `PCH` şi `PCU`. Dintre aceştia, doar `PCL` se poate citi şi scrie direct. Pentru `PCH` şi `PCU` s-au introdus doi regiştrii imagine: `PCLATH` şi `PCLATU`. Astfel, valorile reale din `PCH` şi `PCU` se transferă în `PCLATH` respectiv `PCLATU` numai la o citire a registrului `PCL` (`CALL`, `GOTO` sau citire directă prin instrucțiuni). Dacă se modifică valoarea `PCL`-ului, atunci valorile care se află în `PCLATH` şi `PCLATU` sunt transferate respectiv în `PCH` şi `PCU`.

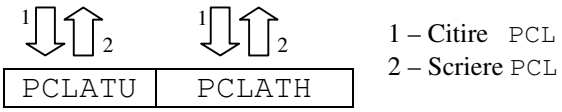

Figura 3.1. Structura numărătorului de program pentru PIC18Fxxxx

În cazul rutinei (b), indexul tabloului se stochează mai întâi într-o variabilă temporară situată la adresa 0x20. La fel ca şi în cazul punctului (a), pentru a face un salt la un anumit index din tablou, mai întâi trebuie stocată o valoare pentru PCLATH, pentru ca la modificarea registrului PCL, PCH să fie încărcat cu o valoare corespunzătoare.

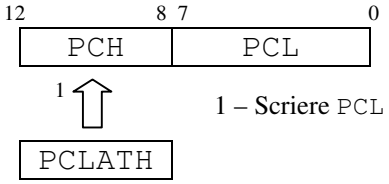

Figura 3.2. Structura numărătorului de program pentru PIC16Fxxxx

3.2.2 Implementarea tablourilor folosind adresarea indirectă.

Adresarea indirectă oferă posibilitatea accesării unei locaţii de memorie RAM fără precizarea unei adrese fixe în instrucţiune. Astfel, dacă se implementează tablourile folosind această metodă, se vor putea stoca valori modificabile pe parcursul execuţiei programului (prin folosirea RAM-ului ca şi zonă de stocare).

Microcontrollerele PIC fac posibilă adresarea indirectă prin implementarea unui set de regiştrii speciali de selecţie a adresei de memorie, denumiţi *File Select Registers* (FSR) (vezi capitolul *Organizarea memoriei*).

Exemplul 3.6. Implementarea unui tablou de 8 elemente folosind adresarea indirectă.

```
;Rutină de stocare a valorilor care vor fi accesate ulterior
;prin rutina de returnare a unei valori
Stocare_Valori:
     LFSR   FSR0, H'200'      ;Bank-ul 2 (cu numărul 3)
     MOVLW  D'101'
     MOVWF  POSTINC0          ;Stocarea valoarii şi imcrementarea
     MOVLW  D'115'            ;automată a registrului FSR0
     MOVWF  POSTINC0
     MOVLW  D'90'
     MOVWF  POSTINC0
     MOVLW  D'83'
     MOVWF  POSTINC0
     MOVLW  D'85'
     MOVWF  POSTINC0
     MOVLW  D'99'
     MOVWF  POSTINC0
     MOVLW  D'150'
     MOVWF  POSTINC0
     MOVLW D'149'
```

```
     MOVWF   POSTINC0
     RETURN

;Rutină de returnare a valorii având indexul stocat în
;registrul de lucru
Ret_Valoare:
     LFSR    FSR1, H'200'      ;Se poate folosi şi FSR0
     MOVWF   FSR1L
     MOVF    INDF1, W
     RETURN
```

Astfel, pentru a putea folosi tablouri implementate prin adresarea indirectă, valorile tabloului trebuie să fie încărcate mai întâi în memoria RAM. Aceasta se poate realiza sub forma unei rutine de inițializare a memoriei, precum în exemplul 3.6.

3.2.3 Implementarea tablourilor folosind instrucțiuni dedicate (valabile doar pentru seria 18Fxxxx).

O altă variantă de implementare a tablourilor în memoria program o reprezintă folosirea instrucțiunilor tabelare oferite de microcontrollerele din seria 18F. Acestea oferă posibilitatea citirii memoriei program octet cu octet şi scrierea acesteia prin blocuri de minim 32 de octeți. Instrucțiunile folosite pentru citire/scriere:

- Citire tabelară: `TBLRD`
- Scriere tabelară: `TBLWT`

Registrul folosit pentru adresare este `TBLPTR`, un registru pe 22 de biți, prezentat în figura 3.3.

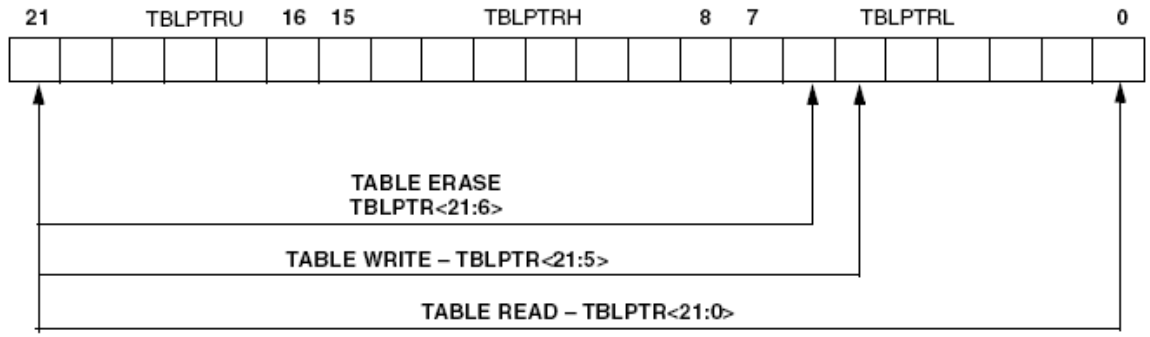

Figura 3.3 Structura registrului TBLPTR

Astfel, pentru adresarea unei locații de memorie FLASH, se vor folosi cele trei componente ale registrului special `TBLPTR` (`TBLPTRU`:`TBLPTRH`:`TBLPTRL`). Aceşti regiştrii sunt folosiți de instrucțiunile `TBLRD` şi `TBLWT` pentru adresarea unei zone de memorie, instrucțiuni care oferă şi posibilitatea modificării automate a regiştrilor adresă (printr-o singură instrucțiune).

Instrucțiunea	Operația executată
`TBLRD*` `TBLWT*`	Se realizează o citire/scriere; `TBLPTR` nu se modifică
`TBLRD*+`	Se realizează o citire/scriere;

TBLWT*+	TBLPTR este incrementat
TBLRD*- TBLWT*-	Se realizează o citire/scriere; TBLPTR este decrementat
TBLRD+* TBLWT+*	TBLPTR este incrementat; Se realizează o citire/scriere

Tabelul 3.1 Parametrii instrucțiunilor TBLRD şi TBLWT

Zona de memorie în care se transferă un octet la execuția unei instrucțiuni de citire este un registru special pe 8 biți: TABLAT. Formele instrucțiunilor de manipulare a memorie FLASH sunt prezentate în Tabelul 3.1.

Pentru stocarea unui şir de valori în memoria program se pot folosi fie instrucțiuni ale căror coduri sunt bine cunoscute de programator şi sunt astfel strategic plasate în program, fie prin definirea unei secvențe de octeți folosind operatorul db, precum în exemplul 3.7.

Exemplul 3.7. Stocarea unei secvențe de octeți în memoria program şi citirea acesteia folosind instrucțiuni TBLRD.

```
Tablou_valori:
     db 0x12, 0x13, 0x90, 0x20, 0x18, 0x23, 0x11, 0x55

;Rutină de inițializare a regiştrilor de adresare
;unde: -'low' - instrucțiune adresată asamblorului
;           pentru determinarea părții inferioare a adresei la
;           care se află eticheta Tablou_valori (i.e. adresa
;           primului element din tablou)
;       - 'high'  - determinarea părții superioare ...
;       - 'upper' - determinarea părții de sus  ...
Init_Adresa:
     MOVLW  low Tablou_valori
     MOVWF  TBLPTRL
     MOVLW  high Tablou_valori
     MOVWF  TBLPTRH
     MOVLW  upper Tablou_valori
     MOVWF  TBLPTRU
     RETURN

;Mai întâi se apelează rutina de inițializare
     CALL Init_Adresa

;După care se pot accesa elementele pe rând folosind ;postincrementarea
     TBLRD*+
     MOVF   TABLAT, W

;Sau se poate accesa un element de la un anumit index
;Însă dacă se modifică 'manual' valorile din regiştrii de ;adresă
trebuie avut grijă la depăşirea valorii 255, când ;trebuie modificat şi
registrul imediat superior
     MOVLW  D'3'        ;se va returna elementul cu index-ul 3
     MOVWF  TBLPTRL
     TBLRD*
     MOVF   TABLAT, W
```

Inițializarea regiştrilor de adresare şi citirea unei zone de memorie FLASH este ilustrată în exemplul 3.7. Realizarea unei modificări a memoriei program (i.e. scrierea memoriei FLASH) folosind instrucțiuni `TBLRD` şi `TBLWT` este puțin mai complexă, cuprinzând operații asupra EEPROM-ului şi instrucțiuni de inițializare bine stabilite.

Prin aceste instrucțiuni, utilizatorul poate construi aplicații tip "Boot-loader" sau poate implementa tablouri modificabile ale căror dimensiune limită este stabilită doar de dimensiunea memoriei program. Pentru mai multe informații legate de scrierea memorie program, se poate consulta site-ul www.microchip.com sau exemplul prezentat în ANEXA 1.

Probleme.

P1. Se consideră 20 de valori stocate sub forma unui tablou (implementarea tabloului se consideră a fi la alegere) şi 8 LED-uri legate la PORTC. Să se afişeze valorile stocate în tabelă pe PORTC.

P2. Se consideră 8 LED-uri legate la PORTC. Să se afişeze pe portul considerat octeții din memoria program folosind instrucțiuni tabelare (metoda numărul 3). Pentru a putea observa valorile afişate, se va folosi un timer pe 8 sau 16 biți.