PROGRAMAREA ÎN LIMBAJ DE ASAMBLARE

GENGE BÉLA

Capitolul 7
Limbajul de asamblare și limbajul C.

Compilatorul C MPLAB C18

- MCC18: compilator ce transformă un program scris în limbajul C într-un set de instrucțiuni în limbaj de asamblare pentru seria PIC18F.
- Față de limbajul C standard există diferențe importante ce pot fi înțelese doar din perspectiva limbajului de asamblare studiat.

Primul program C

- Atenție! Ieșirea **printf** e redirecționată către fereastra de debug sau către interfața de comunicare serială UART, dacă este activat.

```
#include <stdio.h>
void main()
{
     printf("Hello world!");
}
```

- Se păstrează sintaxa C standard.

Primul program C

- Tipuri de date.

Tipul	Dimensiune	Valoarea minimă	Valoarea maximă
char	8 biţi	-128	127
signed char	8 biţi	-128	127
unsigned char	8 biţi	0	255
int	16 biţi	-32768	32767
unsigned int	16 biţi	0	65535
short	16 biţi	-32768	32767
unsigned short	16 biţi	0	65535
short long	24 biţi	-8388608	8388607
unsigned short long	24 biţi	0	16777215
long	32 biţi	-2147483648	2147483647
unsigned long	32 biţi	0	4294967295
float	32 biţi	2^{-126}	$2^{128}*(2-2^{-15})$
double	32 biţi	2^{-126}	$2^{128}*(2-2^{-15})$

Primul program C

- Inițializarea variabilelor.

```
unsigned char c1 = 0x10;              //hexa
unsigned char c2 = 12;                //zecimal
unsigned char c3 = 0b11110101;        //binar
unsigned char c4 = 'Z';               //caracter
```

Secțiuni

- Secțiuni în memoria program:
  - Secțiune de cod: cuv. cheie **code**.
  - Secțiuni de date: cuv. cheie **romdata**.
- Memoria de date:
  - Secțiune de date inițializate: cuv. cheie **idata**. (variabile statice inițializate)
  - Secțiune de date neinițializate: cuv. cheie **udata**. (variabile statice neinițializate)
- Declararea unei secțiuni:

```
#pragma tip_secţiune [nume_secţiune [=adresă]]
```

Secțiuni de program

- Secțiunile se clasifică în:
  - Absolute: se specifică adresa secțiunii.
  - Asignate: numele și adresa secțiunii se specifică într-un script de link-editare.
  - Neasignate: adresa va fi determinată în funcție de setările din scriptul de link-editare.

Secțiuni de program

Declararea unei secțiuni de cod absolute la adresa 0x850:

```
#pragma code secţiune_test=0x850
void f( void )
{
      …
}
```

Secțiuni de program

Declararea unei secţiuni de cod asignate, la adresa 0x900:

```
#pragma code secţiune_test1
void f( void )
{
      …
}
```

Scriptul de link-editare va conţine următoarele:

```
…
CODEPAGE    NAME=page1   START=0x800 END=0x5FFF
…
SECTION     NAME=secţiune_test1      ROM=page1
```

Declararea unei secţiuni de cod neasignate:

```
#pragma code
void f( void )
{
      …
}
```

Secțiuni de program

- Dacă nu se specifică nici o secțiune absolută, compilatorul alocă o secțiune implicită a cărei caracteristici sunt preluate din script-ul de link-editare.
- Prin CODEPAGE memoria program este împărțită în mai multe *pagini:*

```
CODEPAGE   NAME=page   START=0x800 END=0x5FFF
```

Secțiuni de program

- Dacă specifică o secțiune absolută, compilatorul va scrie toate secvențele de cod ce urmează începând cu adresa dată:
  - Se va folosi directiva ORG.
- Declararea unei secțiuni de cod absolute urmată de o secțiune neasignată:

```
#pragma code sect1=0x1000
void f( void )
{
     …
}
#pragma code              // Se revine la secţiunea implicită
void f1( void )
{
     …
}
```

Secțiuni de date

- Compilatorul MCC18 alocă implicit o secțiune de date inițializate și o secțiune de date neinițializate în RAM.
- În funcție de setările compilatorului, aceste secțiuni implicite se pot aloca în mai multe bank-uri (ceea ce necesită schimbarea bank-ului pentru accesare) sau sunt plasate într-un singur bank.

Secțiuni de date

- În cazul secțiunilor inițializate, datele tebuie încărcate în memoria RAM la început, pentru ca acestea să poată fi accesibile pe parcurs.
- Această încărcare se realizează printr-o secvenţă de cod de iniţializare inclusă de regulă în fişierul cu care programul este link-editat: c018i.o.

Secțiuni de date

- Exemplu de declarare a unei secţiuni de date iniţializate, urmată de o secţiune de date neiniţializate:

```
#pragma idata sect1
int i = 1;
char j = 10;
#pragma udata sect2
unsigned char x1;
unsigned char x2;
void f( void )
{
      …
}
```

Secțiuni de date

- Rezolvarea automată a secțiunilor:

```
// Se va aloca în secţiunea idata implicită
int x = 10, y = 20;

// Se va aloca în secţiunea udata implicită
char ch;
void f( void )
{
      ...
}
```

Secțiuni de date

- Declararea constantelor în memoria program:
  - Cuvîntul cheie **rom**, declararea explicită a secțiunii.

```
rom char ch;
void f( void )
{
      ...
}
```

  - Cuvântul cheie **romdata**:

Secțiuni de date

- Declararea constantelor în memoria program:
  - Cuvântul cheie **romdata**:

```
#pragma romdata mem_prog
const rom char x1 = 10;          // Constante de program
const rom x2 = 11;
#pragma data                     // Secţiune de dată implicită
ram int a;                       // Declarare explicită
const char b = 1;
```

- Atenție! Variabilele declarate în memoria program pot fi inițializate (la declarare) și pot fi citite. Modificarea lor NU este posibilă cu instrucțiuni simple, dar NU va fi raportată ca o eroare!

Secțiuni de date

- Exemplu: codul generat la modificarea unei variabile:

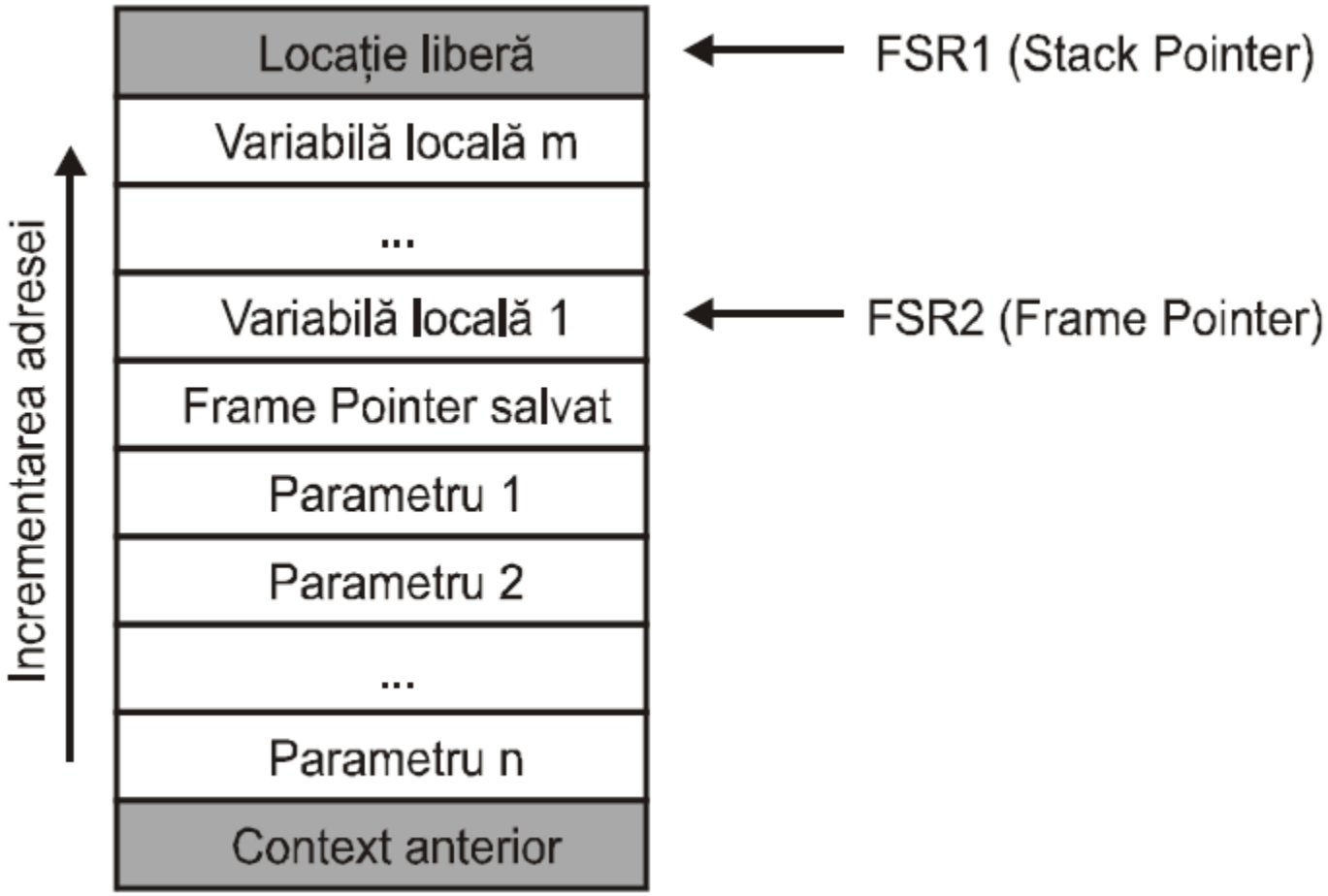
Cod C	Cod asamblare	Comentarii

`rom char c = 0;`
`void f( void )`
`{`
`  ++c;`
`}`
	…
	`MOVLW  low c`
	`MOVWF  TBLPTRL`
	`MOVLW  high c`
	`MOVWF  TBLPTRH`
	`MOVLW  upper c`
	`MOVWF  TBLPTRU`
	`TBLRD*`
	`INCF  TABLAT, F`
	`TBLWT*`
		- Inițializarea celor trei regiştrii de adresare a memoriei program cu cele trei componente de adresă a variabilei ‘c’
		
		- Citirea memoriei program
		- Incrementarea valorii citite
		- Scrierea valorii în mem. Prog.

Stiva de date

- Stivă software diferită de cea hardware.
- Implementarea stivei soft utilizează adresarea indirectă și regiștrii speciali FSR 1, 2 (en. File Select Register).
- Variabilele sunt implicit declarate *auto.*
- FSR1 – este indicatorul vârfului stivei (en. Stack Pointer). Arată către următoarea adresă disponibilă.
- FSR2 – este indicatorul de pagină (en. Frame Pointer). Arată către începutul zonei varibilelor locale, oferind acces rapid la variabilele locale și parametrii.

Stiva de date



Stiva de date

Cod C	Cod asamblare	Comentarii	Pas

//Apelul
funcţiei
f(23, 59);
	
	MOVLW 0x3B
	MOVWF POSTINC1
	MOVLW 0x17
	MOVWF POSTINC1
	CALL @f
	MOVF POSTDEC1
	MOVF POSTDEC1
		
		- 59 -> WREG
		- Stocare val. 59 pe stivă
		- 23 -> WREG
		- Stocare val. 23 pe stivă
		- Apelul rutinei f
		- Scoatere de pe stivă val. 23
		- Scoatere de pe stivă val. 59
			
			1
			1
			1
			1
			2
			11
			11

//Definiţia
funcţiei
void f(char a,
char b)
{
char c;
...
}
	
	MOVFF FSR2L, POSTINC1
	MOVFF FSR1L,FSR2L
	MOVF POSTINC1, F
	...
	MOVF POSTDEC1, F
	MOVF POSTDEC1, F
	MOVFF INDF1, FSR2L
	RETURN
		
		- Salvare FSR2 pe stivă
		- FSR2: început var. locale
		- Declarare var. ‘c’ de 1 octet
		
		- Eliberare mem. ocupată de ‘c’
		- Deplasare la val. salvată FSR2
		- Restaurare val. FSR2
		- Revenire din rutină
			
			3
			4
			5
			6
			7
			8
			9
			10

Stiva de date – returnarea valorilor

- 8 biți: registrul de lucru WREG;
- 16 biți: perechea de regiștrii PRODH:PRODL;
- 24 biți: set de 3 regiștrii alesi de compilator din RAM-ul de uz general:
  - (AARGB2 + 2):(AARGB2 + 1):AARGB2
- 32 biti: set de 4 regiștrii aleși de compilator din RAM-ul de uz general:
  - (AARGB3 + 3):(AARGB3 + 2):(AARGB3 + 1):AARGB3
- 32 biți: se aloca pe stivă, iar FSR0 arată către octetul inferior al valorii returnate.

Variabile statice

- Variabile locale statice: cod redus de accesare, fiind stocate în memoria globală **idata** sau **udata**.
  - Față de C standard variabilele statice pot fi declarate fără să fie inițializate.

Cod C	Cod asamblare	Comentarii

//Definiţia funcţiei
void f(void)
{
  static char c = 0;
  ++c;
  ...
}
	
	MOVFF
	FSR2L,POSTINC1
	MOVFF FSR1L,FSR2L
	INCF c, F
	...
	MOVF POSTDEC1, F
	MOVFF INDF1, FSR2L
	RETURN
		
		- Salvare FSR2 pe stivă
		- FSR2: început var. locale
		
		- Incrementarea variabilei ‘c’
		
		- Deplasare la val. salvata FSR2
		- Restaurare val. FSR2
		- Revenire din rutină

Variabile statice

- Variabile statice locale în memoria program:
  - Variabilele locale declarate în memoria program sunt considerate statice și trebuie să includă cuvântul cheie static în declarație.

```
void f ( void )
{
     static rom char i = 0;
     // Codul generat în asamblare trebuie completat
     // pentru ca operaţia de modificare să se execute
     ++i;
     …
}
```

Pointeri

Tipul memoriei accesate	Calificator	Memoria de alocare	Declarare	Dimensiune (biţi)
Date	near	Date	`char* p;`	16
Date	near	Program	`static char* rom p;`	16
Date	far	Date	`far char* p;`	24
Date	far	Program	`rom far char* p;`	24
Program	near	Date	`rom near char* p;`	16
Program	near	Program	`static rom near char* rom p;`	16
Program	far	Date	`rom far char* p;`	24
Program	far	Program	`static rom far char* rom p;`	24

Tablouri

- Atenție! Tablourile pot epuiza cu ușurință memoria RAM:
  - int a[100].
- Atenție! Există maximum 1-2 bank-uri ce pot fi folosite pentru variabile de uz general!
  - Recomandare: pentru constante să se folosească memoria program!

```
rom char v[] = "ABCDEFG";
```

- Datele vor fi citite prin instrucțiuni TBLRD.

Tablouri

- Exemple.

```
// Memoria ocupată: 20*4 octeţi din memoria program pentru
// şiruri şi 2 octeţi din RAM pentru variabila pointer pStr1
rom const char pStr1[][ 20 ] = { "Sir1", "Sir2",
                                 "Sir3", "Sir4" };

// Memoria ocupată: 5*4 octeţi din memoria program pentru
// şiruri şi 2*4 octeţi din RAM pentru var. pointer pStr2
rom const char* pStr2[]      = { "Sir1", "Sir2",
                                 "Sir3", "Sir4" };
```

Tablouri

- Exemple.

```
// Memoria ocupată: 5*4 octeţi din memoria program pentru
// şiruri şi 2*4 octeţi din memoria program pentru
// variabilele pointer pStr3
rom const char* rom pStr3[]  = { "Sir1", "Sir2",
                                 "Sir3", "Sir4" };

// Memoria ocupată: 5*4 octeţi din memoria RAM pentru
// şiruri şi 2*4 octeţi din memoria RAM pentru
// variabilele pointer pStr4
const char* pStr4[]          = { "Sir1", "Sir2",
                                 "Sir3", "Sir4" };
```

Tablouri

- Mai multe exemple în cartea:
- *B. Genge, P. Haller: Proiectarea sistemelor dedicate și încorporate cu microcontrolerul PIC.*

Optimizarea codului

- Atenție! Nu întotdeauna codul generat e optim:

Cod C	Cod asamblare

`//Var. de tip char`
`c = c + 1;`
 	
`…`
`c = c - 1;`
	`;&c = 0x6A`
	`INCF  0x6A, W, BANKED`
	`MOVWF 0x6A, BANKED`
	`...`
	`DECF  0x6A, W, BANKED`
	`MOVWF 0x6A, BANKED`

`//Var. de tip char`
`++c;`
`…`
`--c;`
	`;&c = 0x6A`
	`INCF  0x6A, F, BANKED`
	`...`
	`DECF  0x6A, F, BANKED`

`//Var. de tip char`
`c++;`
`…`
`c--;`
	`;&c = 0x6A`
	`INCF  0x6A, F, BANKED`
	`...`
	`DECF  0x6A, F, BANKED`

Optimizarea codului

- Atenție! Nu întotdeauna codul generat e optim:

Cod C	Cod asamblare

`//Var. de tip char`
`c = c + 120;`
 	
…
`c = c - 120;`
	`;120 = 0x78, &c = 0x6A`
	`MOVLW 0x78`
	`ADDWF 0x6A, W, BANKED`
	`MOVWF 0x6A`
	`...`
	`MOVLW 0x78`
	`SUBWF 0x6A, W, BANKED`
	`MOVWF 0x6A`

`//Var. de tip char`
`c += 120;`
 	
…
`c -= 120;`
	`;120 = 0x78, &c = 0x6A`
	`MOVLW 0x78`
	`ADDWF 0x6A, F, BANKED`
	`...`
	`MOVLW 0x78`
	`SUBWF 0x6A, F, BANKED`

Inserarea unui cod în ASM

- Codul în ASM este inserat între _asm și _endasm.

```
void portConfig( void )
{
      _asm
            CLRF   PORTA, ACCESS
            CLRF   LATA, ACCESS
            CLRF   PORTB, ACCESS
            CLRF   LATB, ACCESS
            CLRF   TRISA, ACCESS              //B1-B7 de ieşire
            BSF    TRISA, 0, ACCESS           //B0 de intrare
            CLRF   TRISB, ACCESS              //A toate de ieşire
      _endasm
}
```

Inserarea unui cod în ASM

- Codul în ASM este inserat între _asm și _endasm.

```
char a;
void f1()
{
}
void f2()
{
}
void testari( void )
{
      _asm
            MOVLW 0x0A
            CPFSEQ      a, BANKED
            BRA         INEGALITATE
         EGALITATE:
            RCALL f1
            BRA         SFARSIT
         INEGALITATE:
            RCALL f2
         SFARSIT:
      _endasm
}
```

Accesarea regiștrilor speciali

- Se utilizează convențiile:
  - Pentru regiștrii: denumirea registrului (din ASM).
  - Pentru biți: denumirea registrului urmat de “bits.” și urmat de denumirea bitului.
- Exemplu: configurarea PORTD:
  - ADCON1 = 0X0F;
  - CMCON = 0X07;
  - TRISDbits.TRISD0 = 1;
  - TRISDbits.TRISD1 = 1;
  - TRISDbits.TRISD2 = 0;
  - TRISDbits.TRISD3 = 0;
  - PORTD = 0;

Accesarea regiștrilor speciali

- Exercițiu:
  - Scrieți un program C care la apăsarea butonului de pe RD0 aprinde toate LED-urile de pe PORTB, iar la eliberarea butonului le stinge.

Funcții pentru temporizare

- Funcția Nop().
- Funcții din biblioteca delays.h:

Function	Description
`Delay1TCY`	Delay one instruction cycle.
`Delay10TCYx`	Delay in multiples of 10 instruction cycles.
`Delay100TCYx`	Delay in multiples of 100 instruction cycles.
`Delay1KTCYx`	Delay in multiples of 1,000 instruction cycles.
`Delay10KTCYx`	Delay in multiples of 10,000 instruction cycles.

- Exercițiu:
  - Să se implementeze funcția Delay100TCYx().

Exercițiu

- Să se configureze în C Timer0.

7							0
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

Bit 7 **TMR0ON** – Bit de Pornire / Oprire Timer0

1 = Pornire Timer0

0 = Oprire Timer0

Bit 6 **T08BIT** – Bit selecție numărător pe 8 / 16 biți

1 = Timer0 este configurat ca numărător pe 8 biți

0 = Timer0 este configurat ca numărător pe 16 biți

Bit 5 **T0CS** – Bit selecție sursă tact de incrementare

1 = impulsuri externe (tranziții pe pinul T0CKI)

0 = impulsuri interne (tactul de execuție al instrucțiunilor $F_{OSC}/4$)

Bit 4 **T0SE** – Bit selecție front de numărare pentru impulsurile externe

1 = front descrescător pe pinul T0CKI

0 = front crescător pe pinul T0CKI

Exercițiu

- Să se configureze în C Timer0.

7							0
TMR0ON	T08BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0

Bit 3 **PSA** – Bit Utilizare / Dezactivare predivizor

1 = Dezactivare predivizor

0 = Utilizare predivizor

Biții 2-0 **T0PS2:T0PS0** – Biți de selecție a valorii de predivizare

111 = Raport de predivizare 1:256

110 = Raport de predivizare 1:128

101 = Raport de predivizare 1:64

100 = Raport de predivizare 1:32

011 = Raport de predivizare 1:16

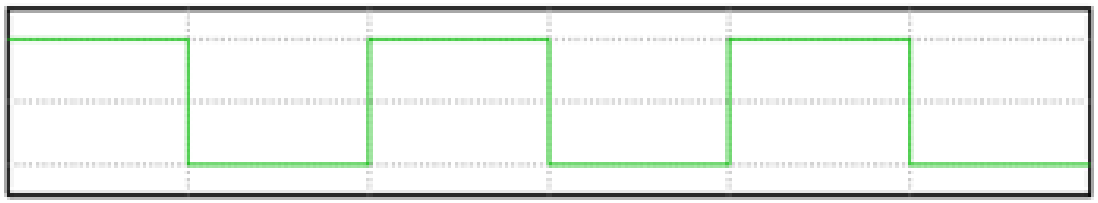
010 = Raport de predivizare 1:8

001 = Raport de predivizare 1:4

000 = Raport de predivizare 1:2

Exercițiu

- Să se configureze în C Timer0.
  - Generarea notelor muzicale pe RB7,RB6:
- C4 - F = 262 Hz, T = 3.8167 mSec, T/2 = 1.9083
- D4 - F = 294 Hz, T = 3.4013 mSec, T/2 = 1.7006
- E4 - F = 330 Hz, T = 3.0303 mSec, T/2 = 1.5151
- F4 - F = 349 Hz, T = 2.8653 mSec, T/2 = 1.4326
- G4 - F = 392 Hz, T = 2.5510 mSec, T/2 = 1.2755
- A4 - F = 440 Hz, T = 2.2727 mSec, T/2 = 1.13635
- B4 - F = 494 Hz, T = 2.0246 mSec, T/2 = 1.0123
- C4 - F = 524 Hz, T = 1.9110 mSec, T/2 = 0.9555



Exercițiu

- Exemplu de calcul: pentru generarea T/2 pentru C4:
  - Tdorit = T/2 = 1.9083ms = 1908.3us = $19083*10^{-1}$us.
  - Tdorit = NrIncr * Prediv * Tinstr.
  - 19083 = NrIncr * Prediv * 2
  - NrIncr = 19083 / (Prediv*2)
    - NrIncr = 9541.5 = 9542 (Prediv = 1, T0=16bit), Val. Init = 65535-9542=0xDAB9.
    - NrIncr = 149.085 = 149 (Prediv = 64, T0=8bit), Val. Init = 0x6A.