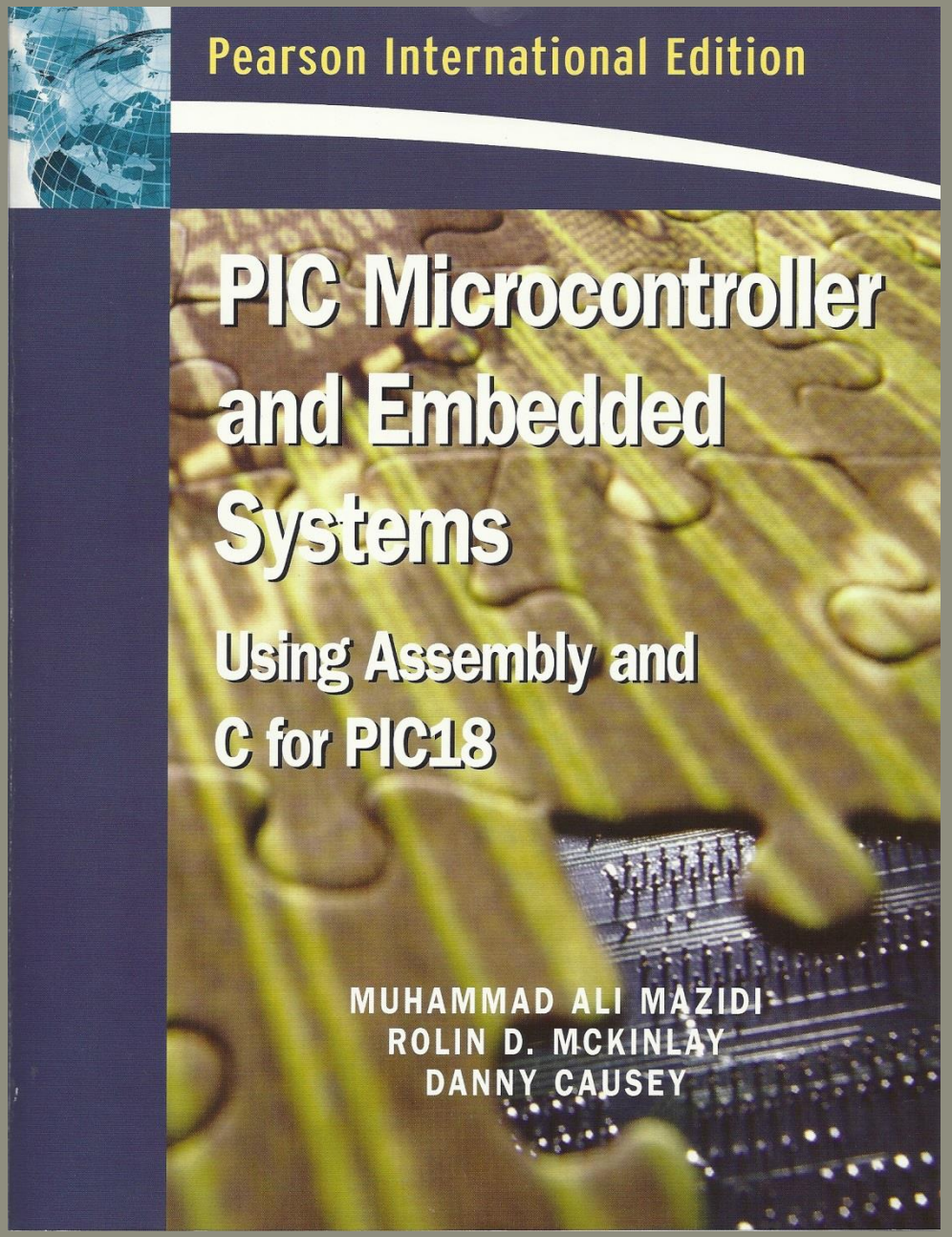
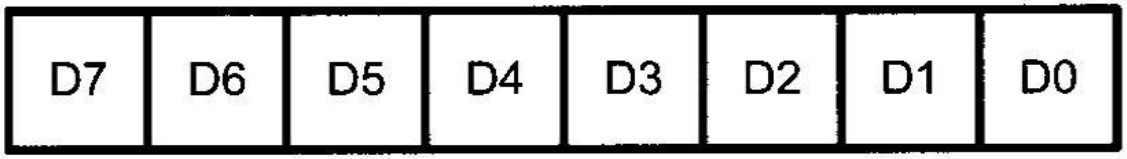
Chapter 2

PIC Architecture & Assembly Language Programming



The WREG Register

- WREG – working register
- The vast majority of PIC register are 8-bit register.



WOVLW Instruction

- Move an 8-bit literal value into WREG register.

  MOVLW K

- The L stands for literal, which means, literally a number must be used; similar to the immediate value in other microprocessors.

  MOVLW 87H

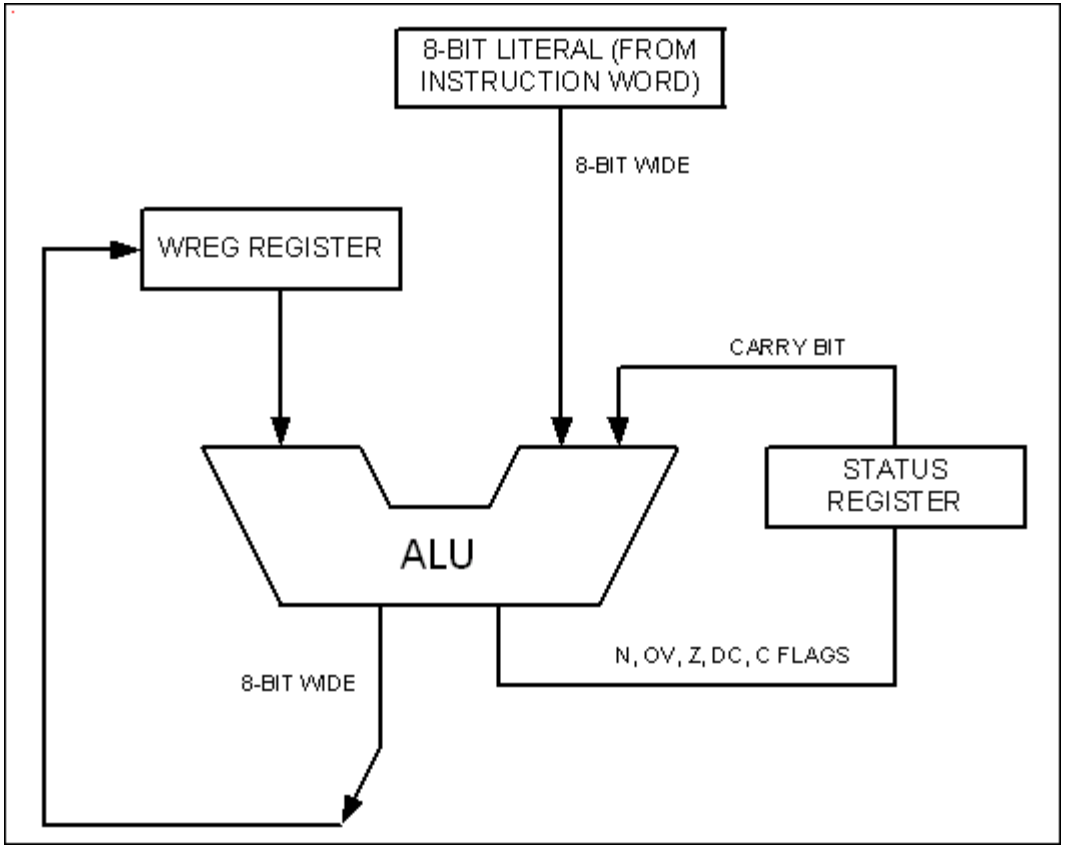
ADDLW Instruction

```
ADDLW   K
```

- ADD a literal value to WREG.

```
MOVLW   25H
ADDLW   34H
;WREG = 59H
```

Figure 2-1. PIC WREG and ALU Using Literal Value




```
MOVLW   12H
ADDLW   16H
ADDLW   11H
ADDLW   43H

MOVLW   7F2H
MOVLW   60A5H
```

The PIC File Register

- Data memory space vs. Program memory space
- The data memory is also called the file register.
- The file register data RAM has a byte-size width, just like WREG.
- (a) Special Function Register (SFR)

  (b) General-Purpose Register (GPR)

  or General-Purpose RAM (GP RAM)

SFRs

- dedicated to specific functions such as ALU status, timers, serial communication, I/O ports, ADC, and so on.
- The more timers we have in a PIC chip, the more SFR registers we will have.

GPRs

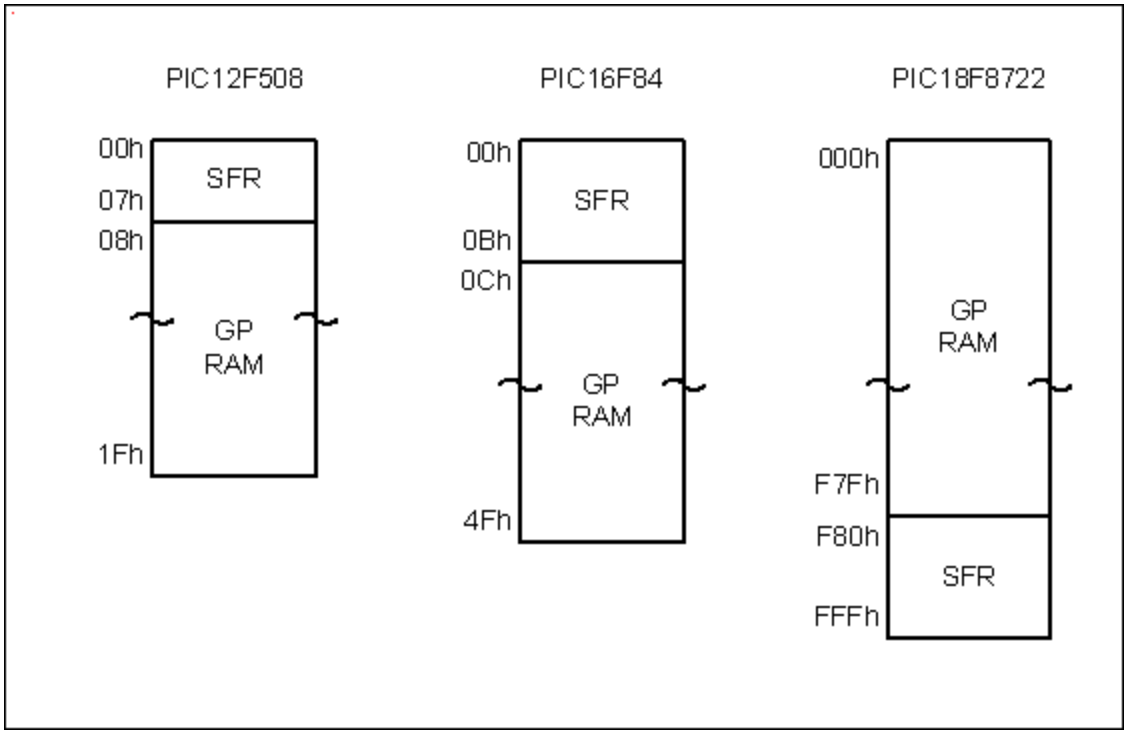
- A group of RAM locations in the file register that are used for data storage and scratch pad.
- The space that is not allocated to SFRs typically is used for general-purpose registers.
- GP RAM vs. EEPROM in PIC chips
- GPRs are used by the CPU for internal data storage.
- EEPROMs are considered as an add-on memory.

File register size for PIC chips

Table 2-1: File Register Size for PIC Chips

	File Register (Bytes)	=	SFR (Bytes)	+	Available space for GPR (Bytes)
PIC12F508	32		7		25
PIC16F84	80		12		68
PIC18F1220	512		256		256
PIC18F452	1792		256		1536
PIC18F2220	768		256		512
PIC18F458	1792		256		1536
PIC18F8722	4096		158		3938

Figure 2-2. File Registers of PIC12, PIC16, and PIC18



File Register and Access Bank

- The PIC18 family can have a maximum of 4096 (4K) bytes for the file register.
- has the addresses of 000-FFF.
- is divided into 256-byte banks.
- A maximum of 16 banks (16×256 = 4096)
- Every PIC18 family member has at least one bank for the file register.
- This bank is called the access bank.
- The access bank is the default bank when we power up the PIC18 chip.

File Register and Access Bank

- The 256 access bank is divided into two equal sections of 128 bytes.
- The 128 bytes from locations 00H to 7FH are set aside for general-purpose registers.
- The other 128 bytes from locations F80H to FFFH are set aside for special function registers.
- A file register of more than 256 bytes will necessitate bank switching.

Figure 2-3. File Register for PIC18 Family

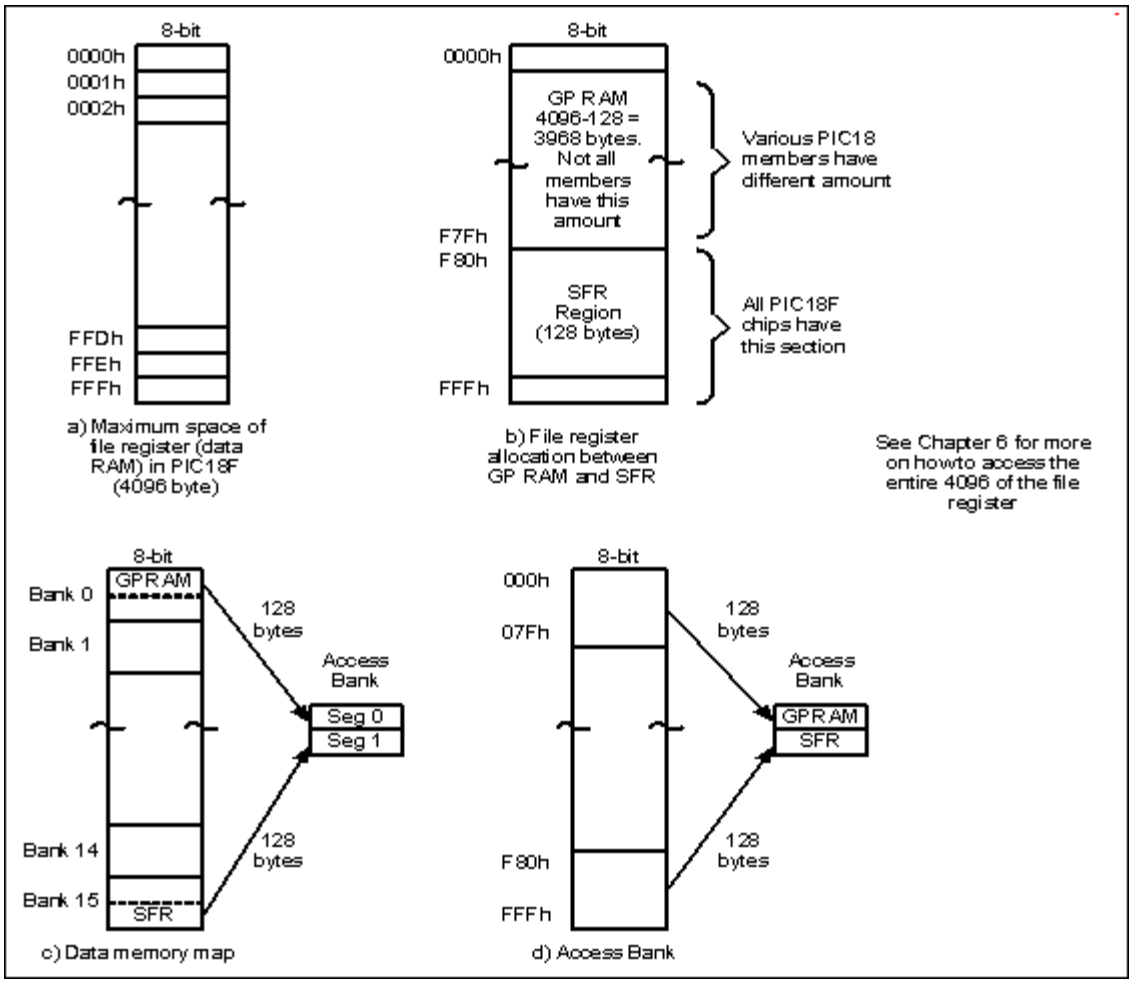


Figure 2-4. Special Function Registers of the PIC18 Family.

F80h	PORTA	FA0h	PIE2	FC0h	----	FE0h	BSR
F81h	PORTB	FA1h	PIR2	FC1h	ADCON1	FE1h	FSR1L
F82h	PORTC	FA2h	IPR2	FC2h	ADCON0	FE2h	FSR1H
F83h	PORTD	FA3h	----	FC3h	ADRESL	FE3h	PLUSW1 *
F84h	PORTE	FA4h	----	FC4h	ADRESH	FE4h	PREINC1 *
F85h	----	FA5h	----	FC5h	SSPCON2	FE5h	POSTDEC1 *
F86h	----	FA6h	----	FC6h	SSPCON1	FE6h	POSTINC1 *
F87h	----	FA7h	----	FC7h	SSPSTAT	FE7h	INDF1 *
F88h	----	FA8h	----	FC8h	SSPADD	FE8h	WREG
F89h	LATA	FA9h	----	FC9h	SSPBUF	FE9h	FSR0L
F8Ah	LATB	FAAh	----	FCAh	T2CON	FEAh	FSR0H
F8Bh	LATC	FABh	RCSTA	FCBh	PR2	FEBh	PLUSW0 *
F8Ch	LATD	FACh	TXSTA	FCCh	TMR2	FECh	PREINC0 *
F8Dh	LATE	FADh	TXREG	FCDh	T1CON	FEDh	POSTDEC0 *
F8Eh	----	FAEh	RCREG	FCEh	TMR1L	FEEh	POSTINC0 *
F8Fh	----	FAFh	SPBRG	FCFh	TMR1H	FEFh	INDF0 *
F90h	----	FB0h	----	FD0h	RCON	FF0h	INTCON3
F91h	----	FB1h	T3CON	FD1h	WDTCON	FF1h	INTCON2
F92h	TRISA	FB2h	TMR3L	FD2h	LVDCON	FF2h	INTCON
F93h	TRISB	FB3h	TMR3H	FD3h	OSCCON	FF3h	PRODL
F94h	TRISC	FB4h	----	FD4h	----	FF4h	PRODH
F95h	TRISD	FB5h	----	FD5h	T0CON	FF5h	TABLAT
F96h	TRISE	FB6h	----	FD6h	TMR0L	FF6h	TBLPTRL
F97h	----	FB7h	----	FD7h	TMR0H	FF7h	TBLPTRH
F98h	----	FB8h	----	FD8h	STATUS	FF8h	TBLPTRU
F99h	----	FB9h	----	FD9h	FSR2L	FF9h	PCL
F9Ah	----	FBAh	CCP2CON	FDAh	FSR2H	FFAh	PCLATH
F9Bh	----	FBBh	CCPR2L	FDBh	PLUSW2 *	FFBh	PCLATU
F9Ch	----	FBCh	CCPR2H	FDCh	PREINC2 *	FFCh	STKPTR
F9Dh	PIE1	FBDh	CCP1CON	FDDh	POSTDEC2 *	FFDh	TOSL
F9Eh	PIR1	FBEh	CCPR1L	FDEh	POSTINC2 *	FFEh	TOSH
F9Fh	IPR1	FBFh	CCPR1H	FDFh	INDF2 *	FFFh	TOSU

* - These are not physical registers.

MOVWF Instruction

- Move (in reality, copy) the source register of WREG (W) to a destination in the file register (F).
- Mnemonic instructions.

```
MOVLW   55H
MOVWF   PORTB
MOVWF   PORTC
MOVWF   2H
MOVWF   3H
```

- Notice that you cannot move literal values directly into the general-purpose RAM locations in the PIC18.

Example 2-1

State the contents of file register RAM locations after the following program:

```
MOVLW 99H          ;load WREG with value 99H
MOVWF 12H
MOVLW 85H          ;load WREG with value 85H
MOVWF 13H
MOVLW 3FH          ;load WREG with value 3FH
MOVWF 14H
MOVLW 63H          ;load WREG with value 63H
MOVWF 15H
MOVLW 12H          ;load WREG with value 12H
MOVWF 16H
```

Solution:

After the execution of `MOVWF 12H` fileReg RAM location 12H has value 99H;
After the execution of `MOVWF 13H` fileReg RAM location 13H has value 85H;
After the execution of `MOVWF 14H` fileReg RAM location 14H has value 3FH;
After the execution of `MOVWF 15H` fileReg RAM location 15H has value 63H;
And so on, as shown in the chart.

Address	Data
012	99
013	85
014	3F
015	63
016	12

ADDWF Instruction

- ADDWF fileReg, D
- Adds together the contents of WREG and a file register locations.
- D indicates the destination bit. If D=0, the destination is WREG. If D=1, then the result will be placed in the file register.
- The PIC assembler allows us to use the letters W or F instead of 0 or 1 to indicate the destination.

Example 2-2

State the contents of file register RAM locations 12H and WREG after the following program:

```
MOVLW 0          ;move 0 WREG to clear it (WREG = 0)
MOVWF 12H        ;move WREG to location 12 to clear it
MOVLW 22H        ;load WREG with value 22H
ADDWF 12H, F     ;add WREG to loc 12H, loc 12 = sum
ADDWF 12H, F     ;add WREG to loc 12H, loc 12 = sum
ADDWF 12H, F     ;add WREG to loc 12H, loc 12 = sum
ADDWF 12H, F     ;add WREG to loc 12H, loc 12 = sum
```

Solution:

The program clears both the WREG and RAM location 12H in the file register. Then it loads WREG with value 22H. From then on, it adds the WREG register and location 12 together and saves the result in location 12H. It does that four times. At the end, location 12H of GP RAM has the value of 88H ($4 \times 22H = 88H$) and WREG = 22H.

After each "`ADDWF 12, F`" instruction

Address	Data
011	
012	22
013	

WREG = 22H

Address	Data
011	
012	44
013	

WREG = 22H

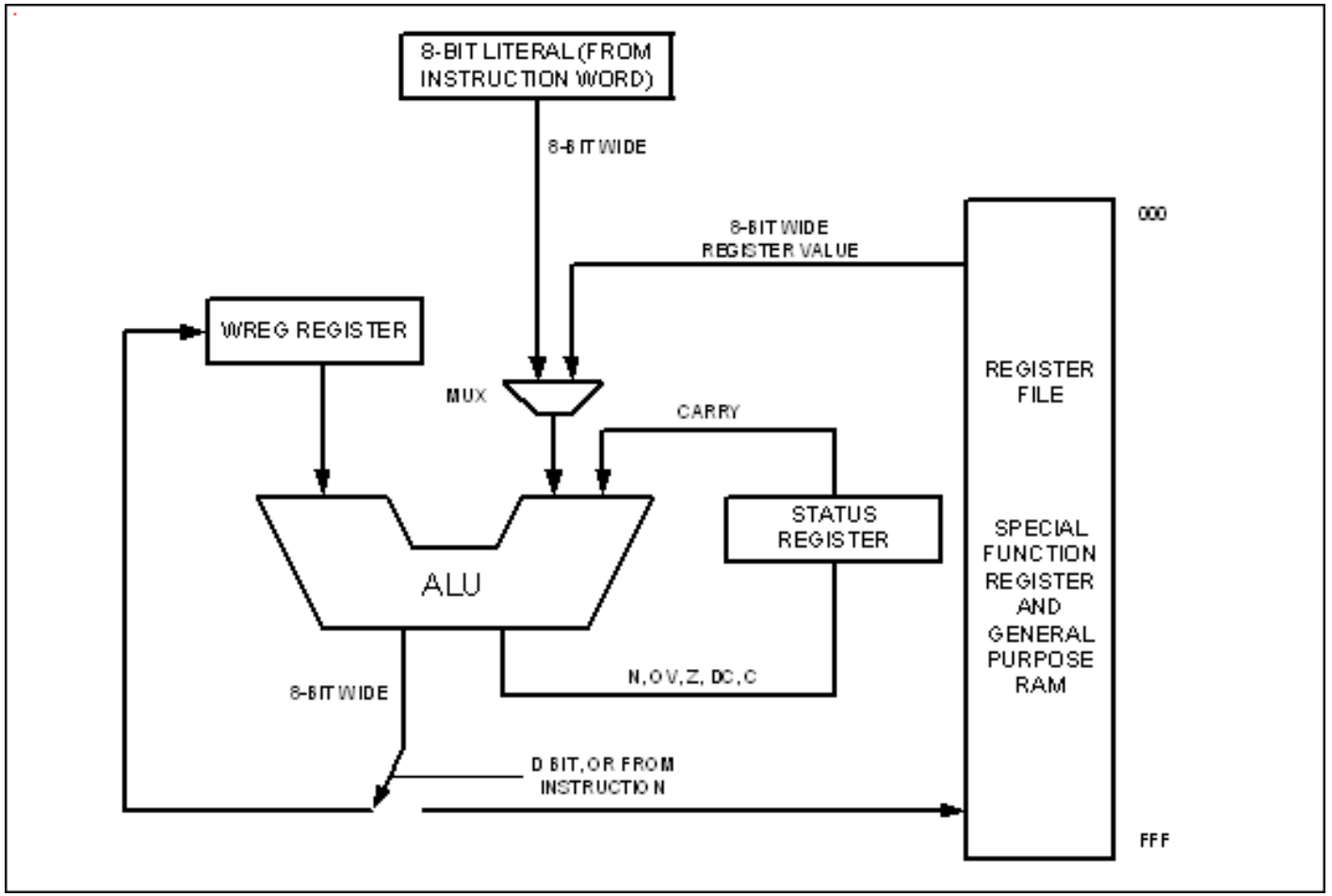
Address	Data
011	
012	66
013	

WREG = 22H

Address	Data
011	
012	88
013	

WREG = 22H

Figure 2-5. WREG, fileReg, and ALU in PIC18



COMF fileReg, d

- Complements the contents of fileReg and places the result in WREG or fileReg.
- This is a example of “Read-Modify-Write”.

Example 2-4

Write a simple program to toggle the SFR of PORT B continuously forever.

Solution:

```
        MOVLW  55H          ;WREG = 55h
        MOVWF  PORTB        ;move WREG to Port B SFR (PB = 55h)
B1      COMF   PORTB, F     ;complement Port B and place it in Port B
        GOTO   B1           ;repeat forever (See Chapter 3 for GOTO)
```

DECF fileReg, d

- Decrements the contents of fileReg and places the result in WREG or fileReg.

```
MOVLW   3
MOVWF   20H
DECF    0x20, F
DECF    0x20, F
DECF    0x20, F
```

MOVF fileReg, d

- Is intended to perform MOVFW.
- The only time we let d=‘F’ (to copy data from fileReg to itself) is when we want to affect the flag bits of the status register.

Example 2-5

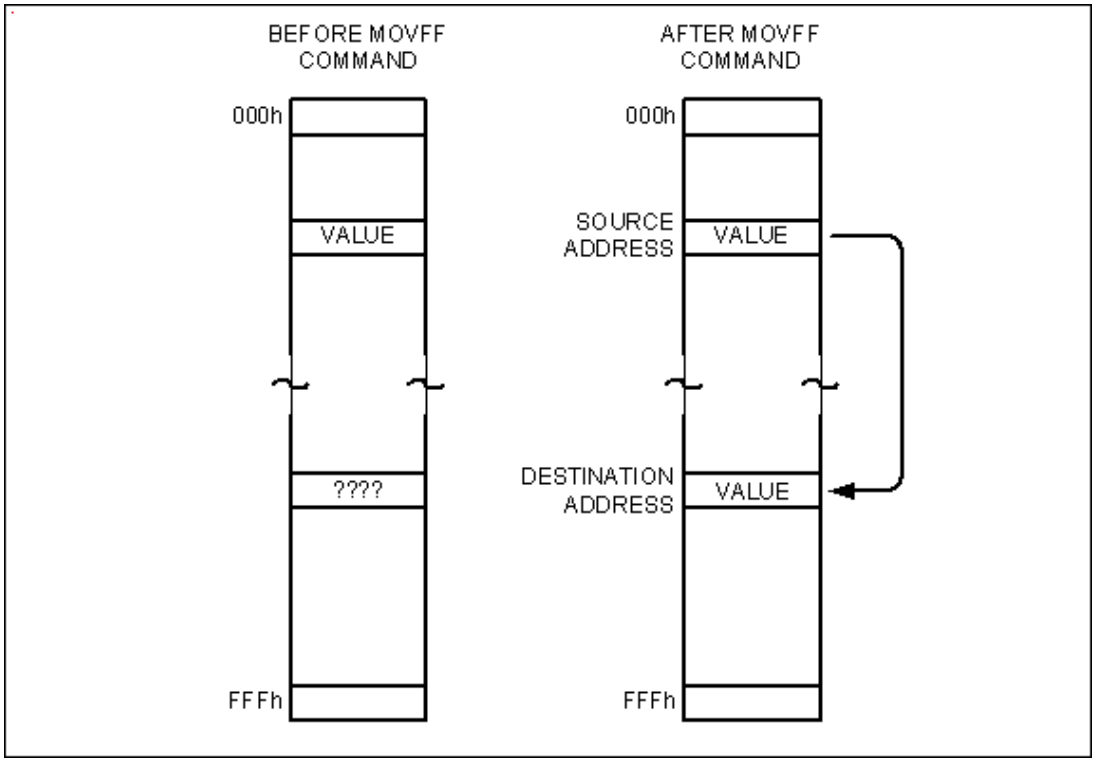
Write a program to get data from the SFRs of Port B and send it to the SFRs of PORT C continuously.

Solution:

```
AGAIN MOVF   PORTB, W     ;bring data from PortB into WREG
      MOVWF  PORTC        ;send it to Port C
      GOTO   AGAIN        ;keep doing it forever
```

MOVFF instruction

- Copies data from one location in fileReg to another location of fileReg.



Example 2-7

Write a program to get data from the SFRs of Port B and send it to the SFRs of PORT C continuously using MOVFF. Compare this to Example 2-5 and explain the difference.

Solution:

```
AGAIN MOVFF PORTB, PORTC        ;copy data from Port B to Port C
      GOTO AGAIN                ;keep doing it forever
```

In Example 2-5 we have:

```
AGAIN MOVF PORTB, W             ;bring data from Port B into WREG
      MOVWF PORTC               ;send it to Port C
      GOTO  AGAIN               ;keep doing it forever
```

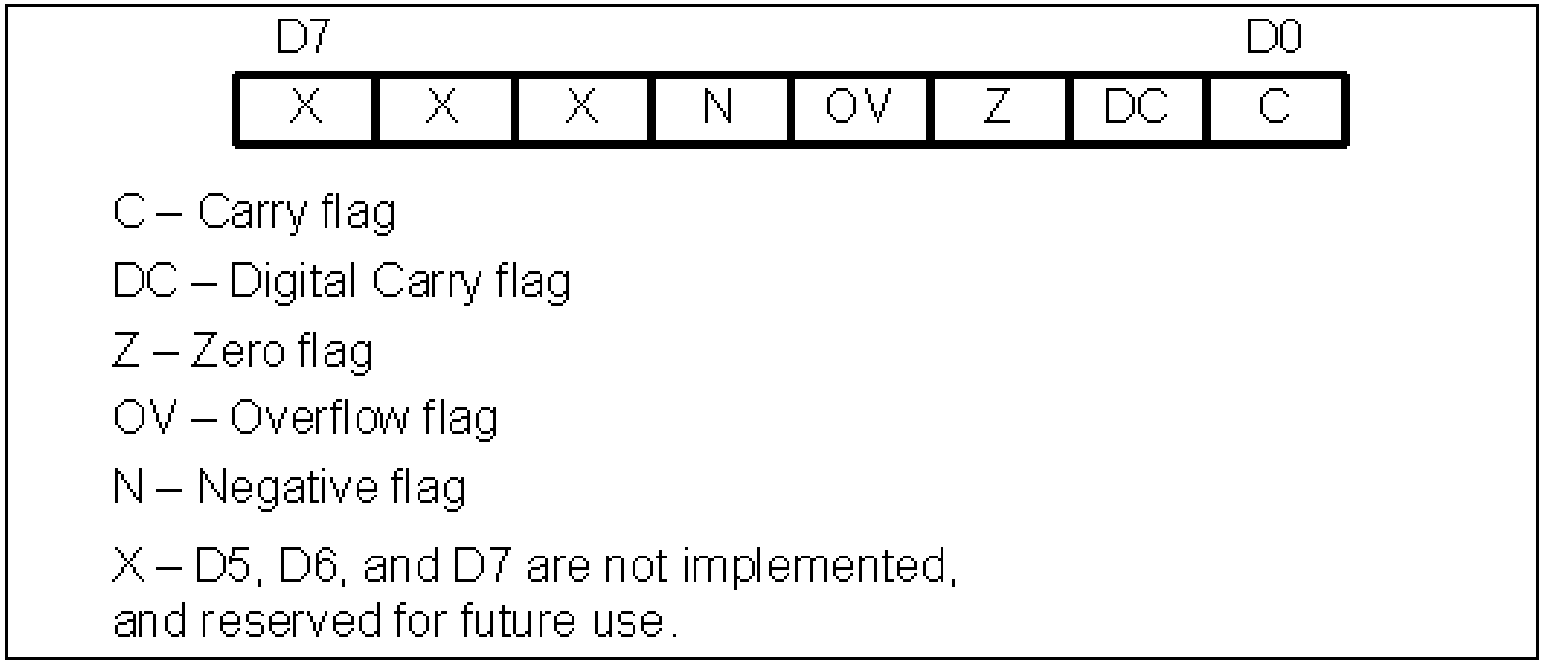
Using MOBVFF we simply copy data from one location to another location. But when we use WREG we can perform arithmetic and logic operations on data before it is moved.

PIC18 Status Register

- Also called flag register.
- Five flags are called conditional flags.

- C, there is a carry out. Usually for unsinged number.
- DC, a carry from D3 to D4. (or AC flag)
- Z, zero.
- OV, overflow. Usually for singed number.
- N, negative. Usually for unsinged number.

Figure 2-7. Bits of Status Register



Example 2-8

Show the status of the C, DC, and Z flags after the addition of 38H and 2FH in the following instructions:

```
MOVLW 38H
ADDLW 2FH          ;add 2FH to WREG
```

Solution:

38H	0011 1000	
+ 2FH	0010 1111	
67H	0110 0111	WREG = 67H

C = 0 because there is no carry beyond the D7 bit.
DC = 1 because there is a carry from the D3 to the D4 bit.
Z = 0 because the WREG has a value other than 0 after the addition.

Example 2-9

Show the status of the C, DC, and Z flags after the addition of 9CH and 64H in the following instructions:

```
MOVLW 9CH
ADDLW 64H          ;add 64H to WREG
```

Solution:

9CH	1001 1100	
+ 64H	0110 0100	
100H	0000 0000	WREG = 00

C = 1 because there is a carry beyond the D7 bit.
DC = 1 because there is a carry from the D3 to the D4 bit.
Z = 1 because the WREG has a value 0 in it after the addition.

Table 2-4: Instructions That Affect Flag Bits

Instruction	C	DC	Z	OV	N
ADDLW	X	X	X	X	X
ADDWF	X	X	X	X	X
ADDWFC	X	X	X	X	X
ANDLW			X		X
ANDWF			X		X
CLRF			X		
COMF			X		X
DAW	X				
DECF	X	X	X	X	X
INCF	X	X	X	X	X
IORLW			X		X
IORWF			X		X
MOVF			X		
NEGF	X	X	X	X	X
RLCF	X		X		X
RLNCF			X		X
RRCF	X		X		X
RRNCF			X		X
SUBFWB	X	X	X	X	X
SUBLW	X	X	X	X	X
SUBWF	X	X	X	X	X
SUBWFB	X	X	X	X	X
XORLW			X		X
XORWF			X		X

Note: X can be 0 or 1.

Table 2-5: PIC18 Branch (Jump) Instructions Using Flag Bits

Instruction	Action
BC	Branch if C = 1
BNC	Branch if C ≠ 0
BZ	Branch if Z = 1
BNZ	Branch if Z ≠ 0
BN	Branch if N = 1
BNC	Branch if N ≠ 0
BOV	Branch if OV = 1
BNOV	Branch if OV ≠ 0

Data Format representation

- There are four ways to show hex numbers.

```
MOVLW   99H
MOVLW   0x99
MOVLW   99
MOVLW   h'99'
```

- Binary numbers

```
MOVLW   B'10011001'
```

Data Format representation

- There are two ways to show decimal numbers.

```
MOVLW   D’12’
MOVLW   .12
```

- ASCII character

```
MOVLW   A’2’
MOVLW   ’2’
```

Assembler Directives

- Instructions tell CPU what to do.
- Directives (pseudo instructions) give directions to the assembler.
- EQU associates a constant number with a data or an address label.

```
COUNT   EQU     25H
        MOVLW   COUNT
```

- SET and EQU directives are identical. The only difference is the value assigned by the SET may be reassigned later.

Assembler Directives

- ORG – the beginning of the address.
- END – the END of the source (asm) file.
- LIST – the assembler the specific PIC chip for which the program should be assembled.

  LIST P=18F452
- #include – libraries used for compiling.
- Radix – numbering system is hexadecimal or decimal.

PIC Assembly Programming

- Machine language
- Assembly language
- Assembler, objective code
- Low-level language
- Complier , high-level language

Structure of Assembly Language

[label] mnemonic [operands] [;comment]

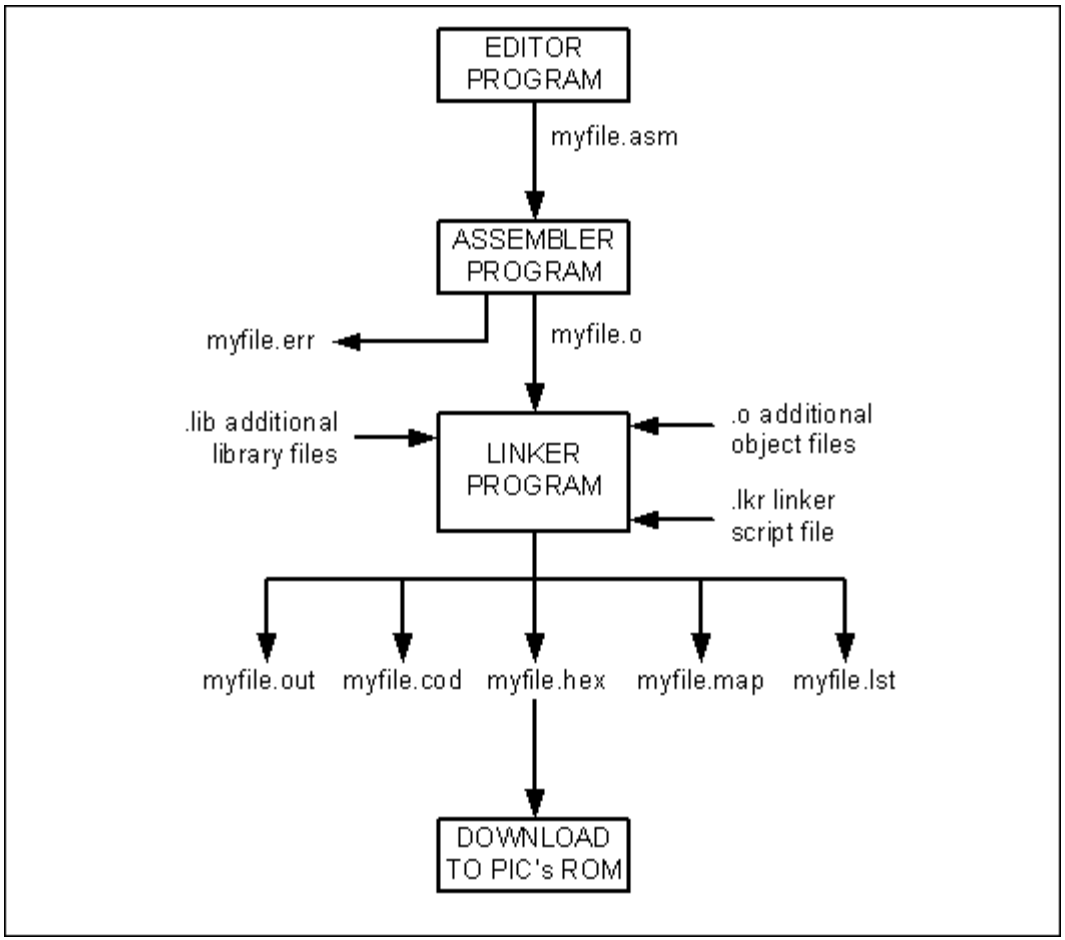
```
        ;PIC Assembly Language Program To Add Some Data.
        ;store SUM in fileReg location 10H.

SUM     EQU   10H           ;RAM loc 10H for SUM

        ORG   0H            ;start at address 0
        MOVLW 25H           ;WREG = 25
        ADDLW 0x34          ;add 34H to WREG
        ADDLW 11H           ;add 11H to WREG
        ADDLW D'18'         ;W = W + 12H = 7CH
        ADDLW 1CH           ;W = W + 1CH = 98H
        ADDLW B'00000110'   ;W = W + 6 = 9EH
        MOVWF SUM           ;save the SUM in loc 10H
HERE    GOTO HERE           ;stay here forever
        END                 ;end of asm source file
```

Program 2-1: Sample of an Assembly Language Program

Figure 2-8. Steps to Create a Program



```
Warning[207]  C:\MDEPIC\EXAMPLE 2-1.ASM 6   : Found label after column 1. (R4)
Warning[207]  C:\MDEPIC\EXAMPLE 2-1.ASM 13  : Found label after column 1. (movle)
Error[122]    C:\MDEPIC\EXAMPLE 2-1.ASM 13  : Illegal opcode (d)
Warning[207]  C:\MDEPIC\EXAMPLE 2-1.ASM 17  : Found label after column 1. (DEC)
Error[122]    C:\MDEPIC\EXAMPLE 2-1.ASM 17  : Illegal opcode (COUNT)
Warning[203]  C:\MDEPIC\EXAMPLE 2-1.ASM 20  : Found opcode in column 1. (movwf)
Warning[207]  C:\MDEPIC\EXAMPLE 2-1.ASM 21  : Found label after column 1. (addl)
Error[108]    C:\MDEPIC\EXAMPLE 2-1.ASM 21  : Illegal character (0)
Error[116]    C:\MDEPIC\EXAMPLE 2-1.ASM 29  : Address label duplicated or differ-
ent in second pass (AGAIN)
```

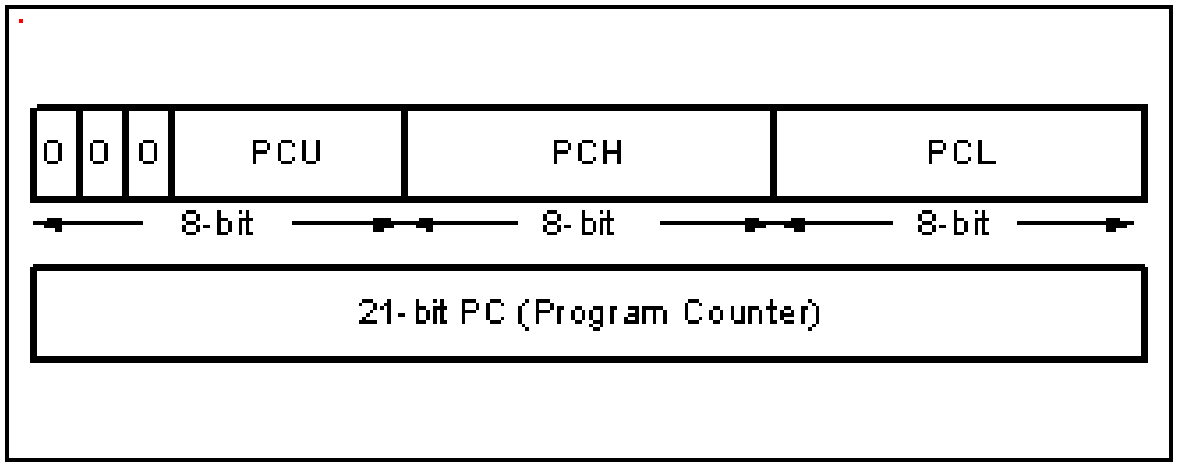
Program 2-1: Sample of a PIC Error (err file)

```
LOC   OBJECT CODE LINE SOURCE TEXT        VALUE

                  00001
                  00002 ;PIC Asm Language Program To Add Some Data
                  00003 ;store SUM in fileReg location 10H
  00000010        00004 SUM EQU 10H       ;RAM loc 10H for Sum
                  00005
000000            00006 ORG 0H            ;start at address 0
000000 0E25       00007 MOVLW 25H         ;WREG = 25
000002 0F34       00008 ADDLW 0x34        ;add 34H to WREG
000004 0F11       00009 ADDLW 11H         ;add 11H to WREG
000006 0F12       00010 ADDLW D'18'       ;W = W + 12H = 7CH
000008 0F1C       00011 ADDLW 1CH         ;W = W + 1CH = 98H
00000A 0F06       00012 ADDLW B'00000110' ;W = W + 6 = 9EH
00000C 6E10       00013 MOVWF SUM         ;save the SUM in loc 10H
00000E EF07 F000  00014 HERE GOTO HERE    ;stay here forever
                  00015 END               ;end of asm source file
```

Program 2-1: List File

Figure 2-9. Program Counter in PIC18



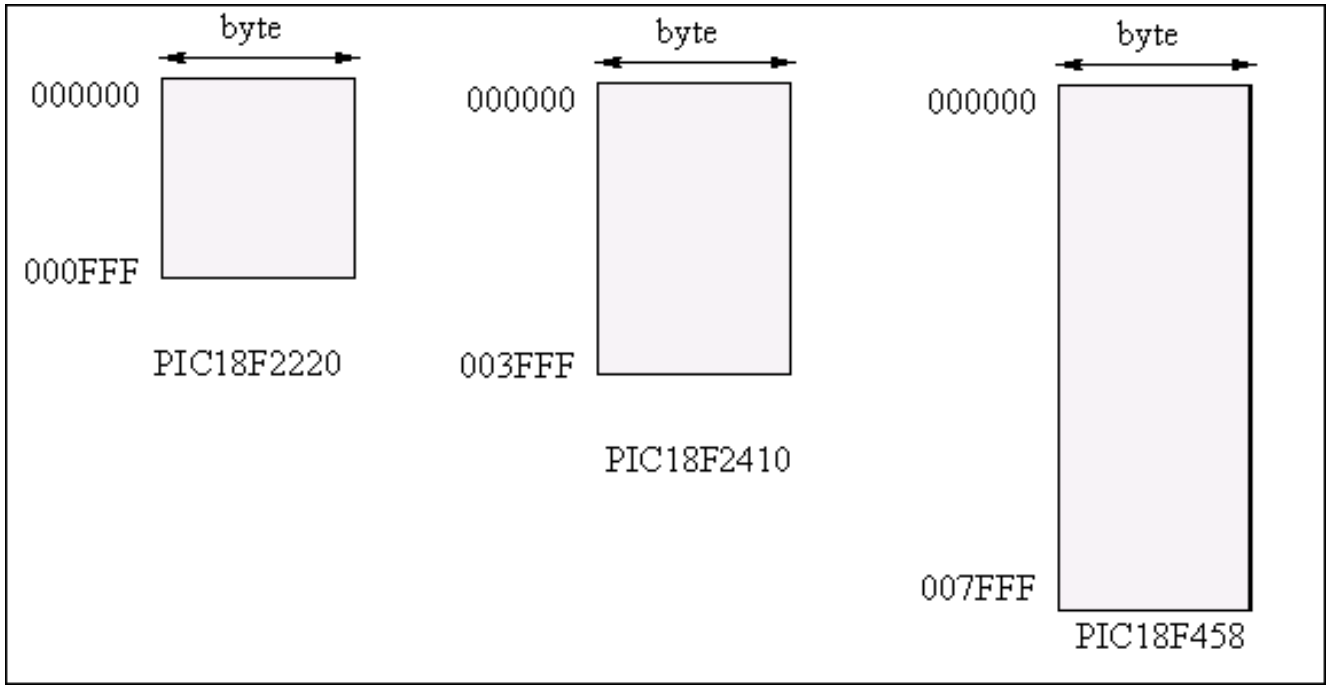
Example 2-11

Find the ROM memory address of each of the following PIC chips:
(a) PIC18F2220 with 4 KB
(b) PIC18F2410 with 16 KB
(c) PIC18F458 with 32 KB

Solution:

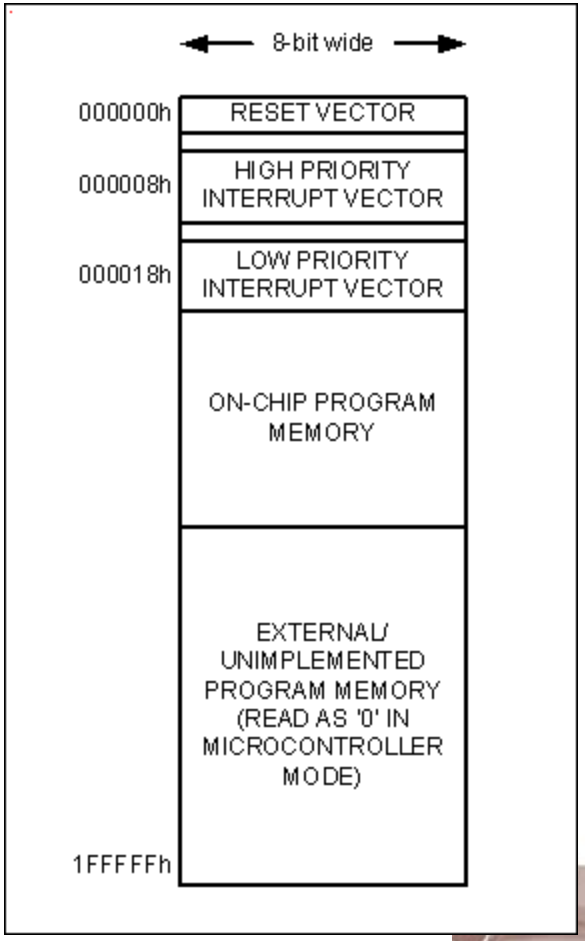
(a) With 4K of on-chip ROM memory space, we have 4096 bytes ($4 \times 1024 = 4096$). This maps to address locations of 0000 to 0FFFH. Notice that 0 is always the first location.
(b) With 16K of on-chip ROM memory space, we have 16,384 bytes ($16 \times 1024 = 16,384$),which gives 0000–3FFFH.
(c) With 32K we have 32,768 bytes ($32 \times 1024 = 32,768$). Converting 32,768 to hex, we get 8000H; therefore, the memory space is 0000 to 7FFFH.

Figure 2-10. PIC18 On-Chip Program (code) ROM Address Range



PIC18 Program ROM Space

- The PIC microcontroller wakes up at memory address 0000 when it is powered up.
- We achieve this by using the ORG statement in the source program as shown earlier.



```
LOC   OBJECT CODE LINE SOURCE TEXT        VALUE

                  00001
                  00002 ;PIC Asm Language Program To Add Some I
                  00003 ;store SUM in fileReg location 10H
  00000010        00004 SUM EQU 10H       ;RAM loc 10H for Sum
                  00005
000000            00006 ORG 0H            ;start at address 0
000000 0E25       00007 MOVLW 25H         ;WREG = 25
000002 0F34       00008 ADDLW 0x34        ;add 34H to WREG
000004 0F11       00009 ADDLW 11H         ;add 11H to WREG
000006 0F12       00010 ADDLW D'18'       ;W = W + 12H = 7CH
000008 0F1C       00011 ADDLW 1CH         ;W = W + 1CH = 98H
00000A 0F06       00012 ADDLW B'00000110' ;W = W + 6 = 9EH
00000C 6E10       00013 MOVWF SUM         ;save the SUM in loc
00000E EF07 F000  00014 HERE GOTO HERE    ;stay here forever
                  00015 END               ;end of asm source fi
```

Program 2-1: List File

Program 2-1: ROM Contents

Address	Code
000000	0E
000001	25
000002	0F
000003	34
000004	0F
000005	11
000006	0F
000007	12
000008	0F
000009	1C
00000A	0F
00000B	06
00000C	6E
00000D	10
00000E	07
00000F	EF
000010	00
000011	F0
000012	

Figure 2-12. Program ROM Width for the PIC18

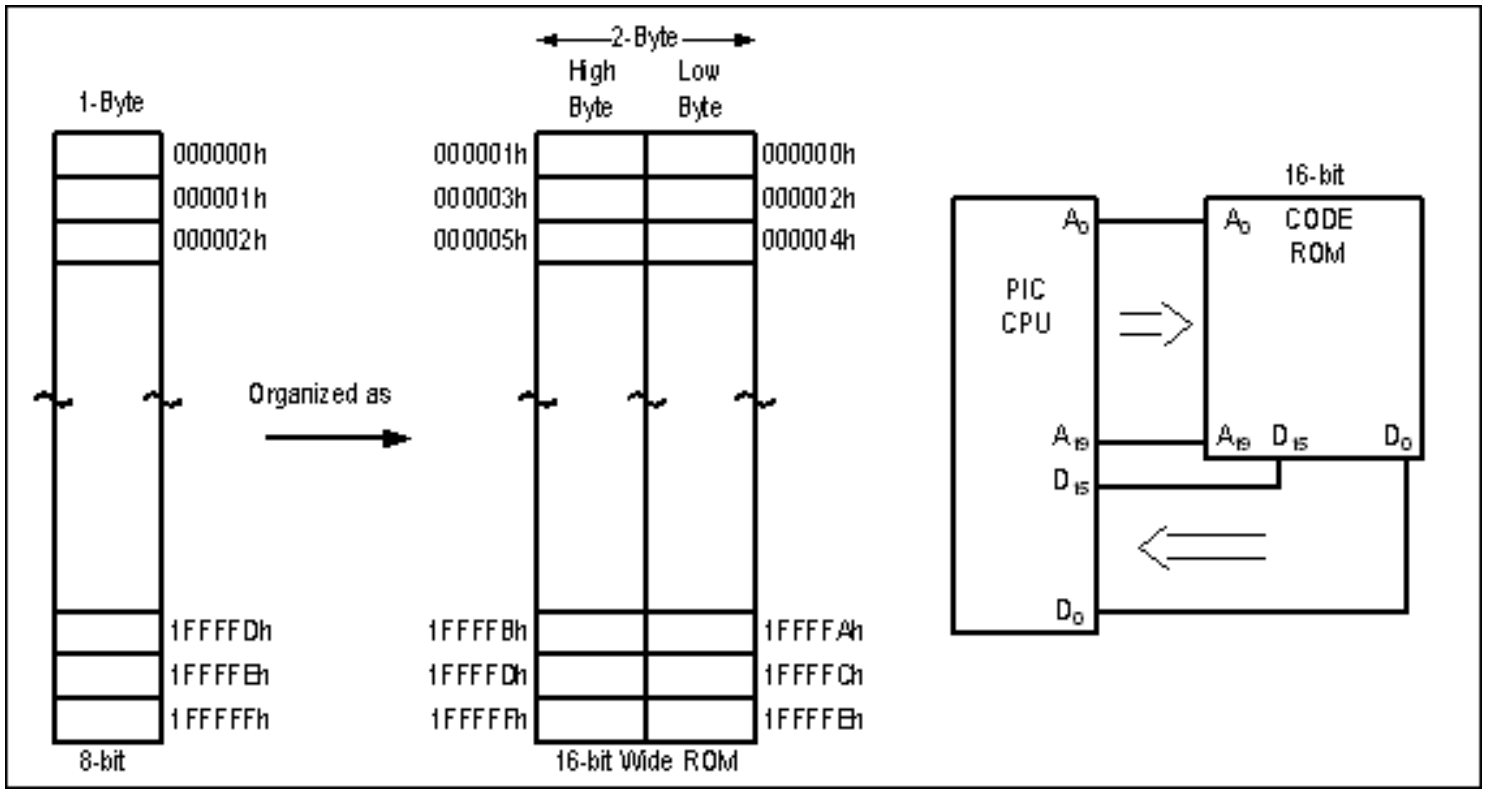


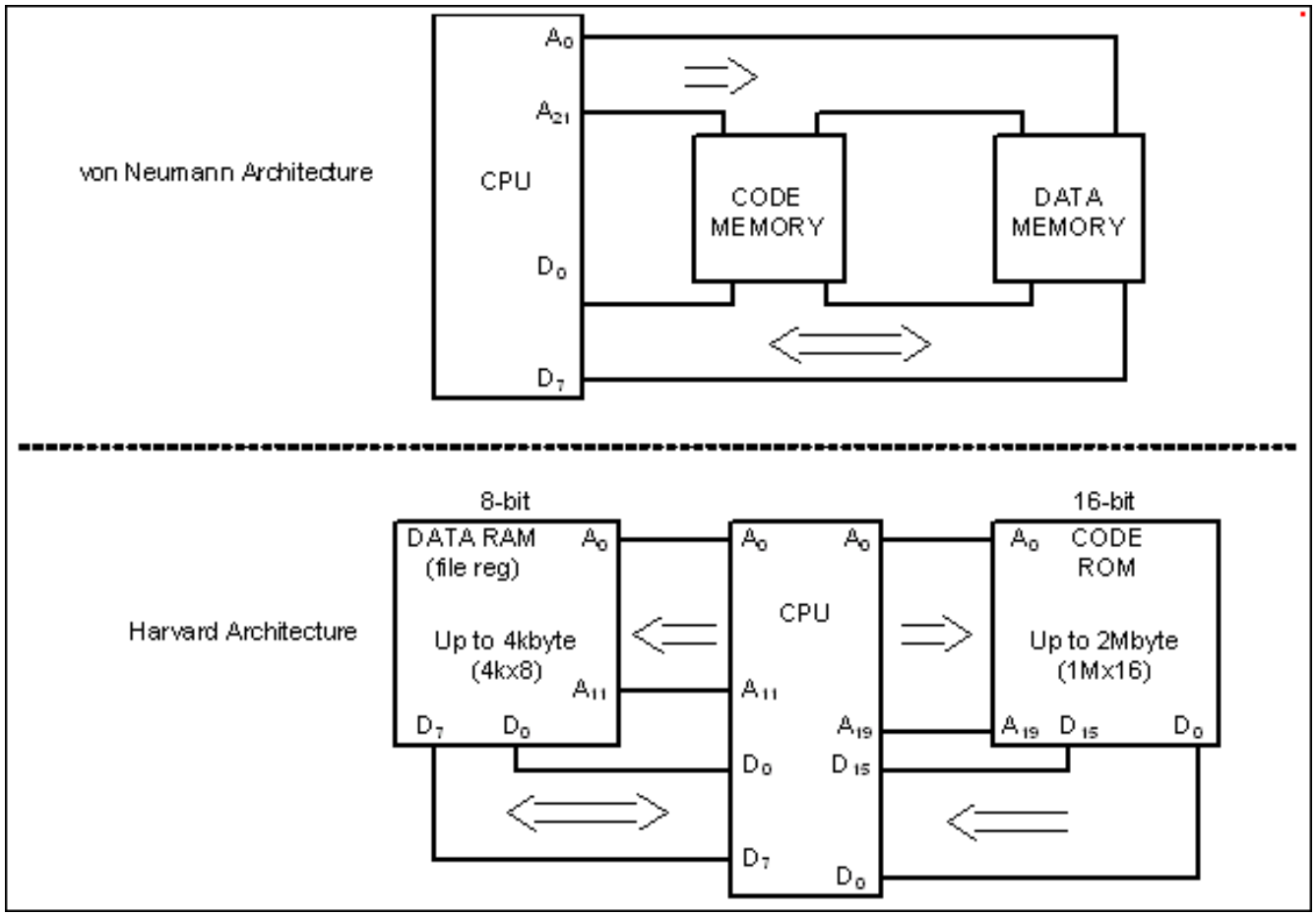
Figure 2-13. PIC18 Program ROM Contents for Program 2-1 List File

- Little endian – The lower byte goes to the low memory location and the high byte goes to the high memory address.

WORD ADDRESS	HIGH BYTE	LOW BYTE
000000h	0Eh	25h
000002h	0Fh	34h
000004h	0Fh	11h
000006h	0Fh	12h
000008h	0Fh	1Ch
00000Ah	0Fh	06h
00000Ch	6Eh	10h
00000Eh	EFh	07h
000010h	0Fh	00h

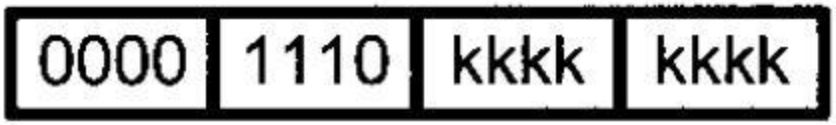
- Big endian

Figure 2-14. von Neumann vs. Harvard Architecture



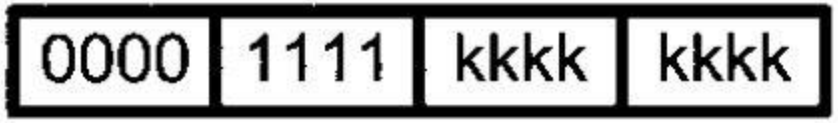
Instruction Size

- MOVLW



$0 \leq k \leq FF$

- ADDLW



$0 \leq k \leq FF$

Instruction Size

- MOVWF

0110	111a	ffff	ffff

$0 \leq f \leq FF$

a = 0 : access bank is used.

a = 1 : access bank is specified by the BSR register.

- MOVFF

1100	ssss	ssss	ssss
1111	dddd	dddd	dddd

Source (f_s)

Destination (f_d)

$0 \leq f_s \leq FFF$

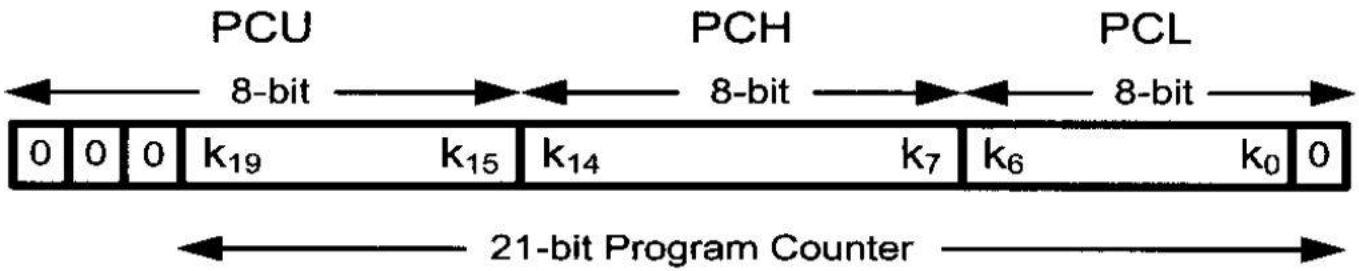
$0 \leq f_d \leq FFF$

Instruction Size

- GOTO

1110	1111	k_7kkk	$kkkk_0$
1111	$k_{19}kkk$	kkkk	$kkkk_8$

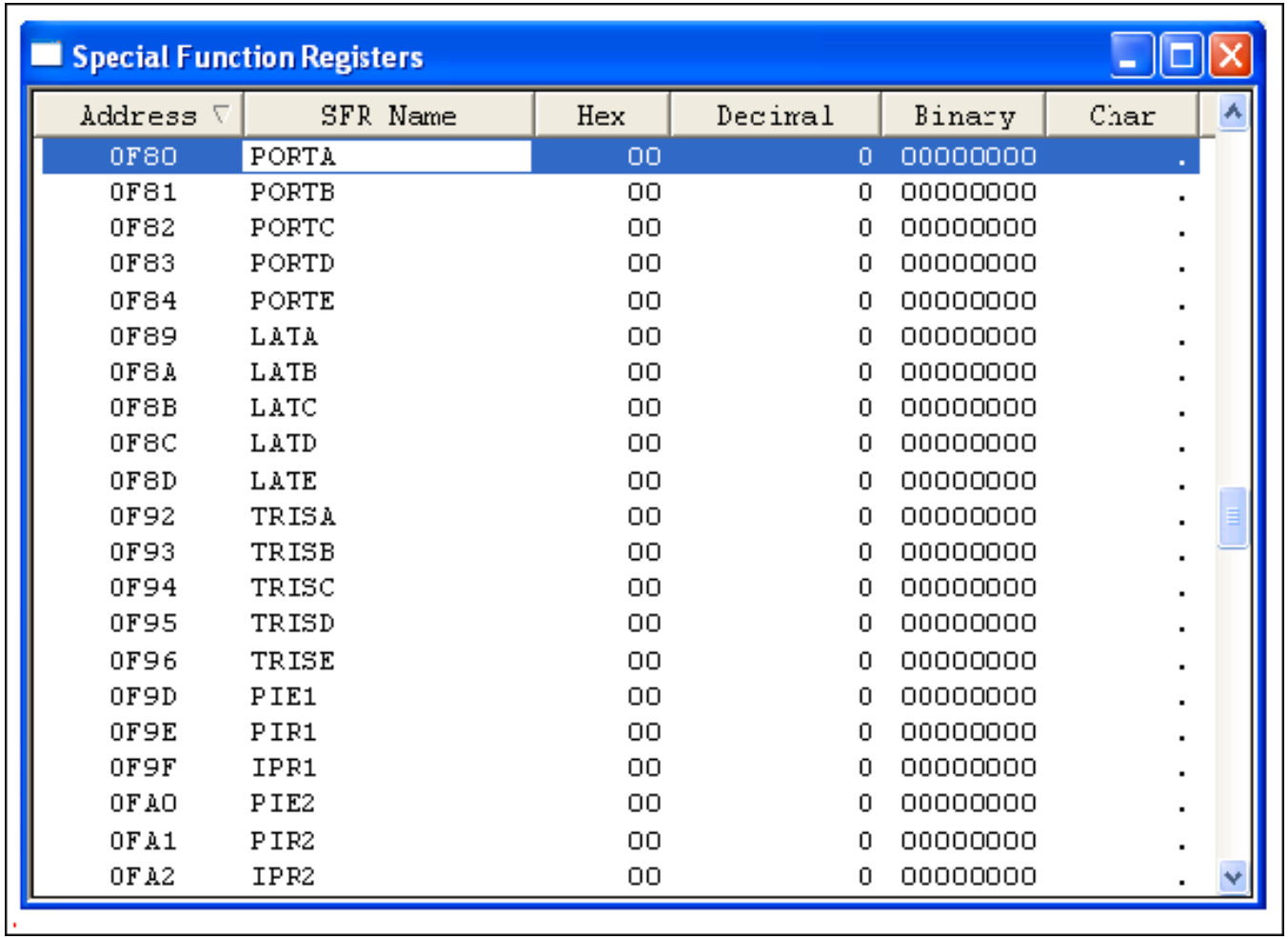
$$0 \leq k \leq FFFFF$$



RISC Architecture

1. Fixed instruction size
2. A large number of registers
3. A small instruction set
4. 95% instructions are executed with only one clock cycle
5. Separate buses for data and code (Havard architecture)
6. Hardwire method. (no microinstructions)
7. Load/store architecture

Figure 2-15. SFR Window in MPLAB Simulator



Address	SFR Name	Hex	Decimal	Binary	Char
0F80	PORTA	00	0	00000000	.
0F81	PORTB	00	0	00000000	.
0F82	PORTC	00	0	00000000	.
0F83	PORTD	00	0	00000000	.
0F84	PORTE	00	0	00000000	.
0F89	LATA	00	0	00000000	.
0F8A	LATB	00	0	00000000	.
0F8B	LATC	00	0	00000000	.
0F8C	LATD	00	0	00000000	.
0F8D	LATE	00	0	00000000	.
0F92	TRISA	00	0	00000000	.
0F93	TRISB	00	0	00000000	.
0F94	TRISC	00	0	00000000	.
0F95	TRISD	00	0	00000000	.
0F96	TRISE	00	0	00000000	.
0F9D	PIE1	00	0	00000000	.
0F9E	PIR1	00	0	00000000	.
0F9F	IPR1	00	0	00000000	.
0FA0	PIE2	00	0	00000000	.
0FA1	PIR2	00	0	00000000	.
0FA2	IPR2	00	0	00000000	.

Figure 2-16. File Register (Data RAM) Window in MPLAB Simulator

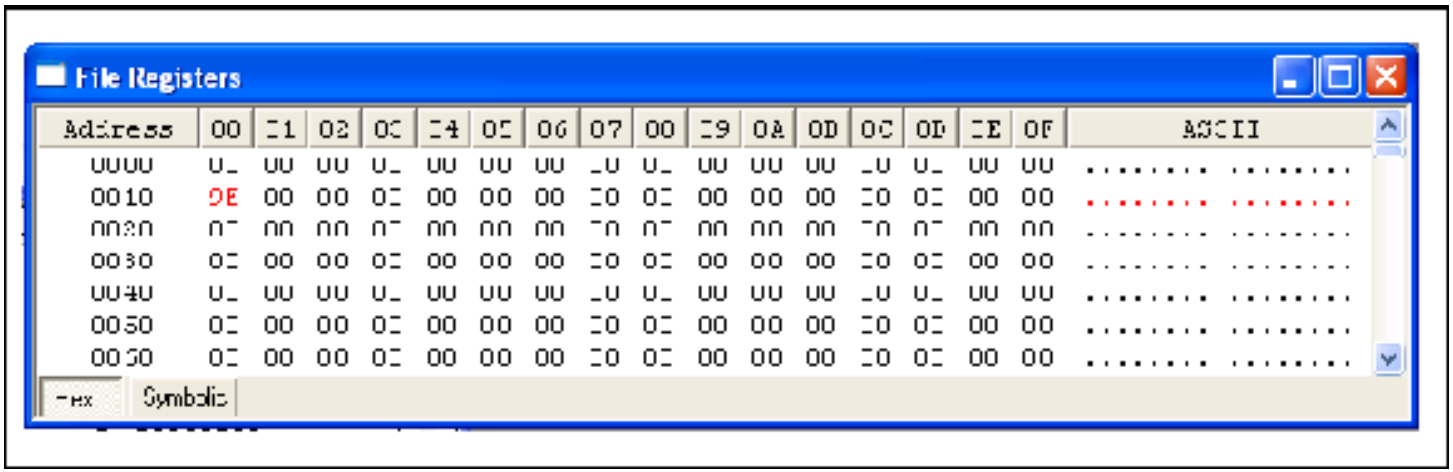


Figure 2-17. Program (Code) ROM Window in MPLAB Simulator