جامعة فلســـــطين التقنية - خضـــــوري
Palestine Technical University - Kadoorie

Microcontrollers
CH 2

Dr. Jafar Saifeddin Jallad

Dept. of Electrical Engineering

Palestine Technical University

Tulkaram, Palestine

Chapter 2: Core SFRs

Features and Function

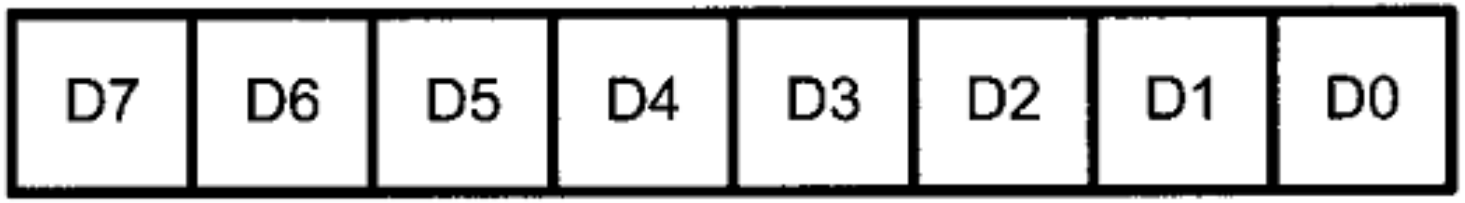
The special function registers can be classified into two categories:

- Core (CPU) registers - control and monitor operation and processes in the central processor. Even though there are only a few of them, the operation of the whole microcontroller depends on their contents.
- Peripheral SFRs- control the operation of peripheral units (serial communication module, A/D converter etc.). Each of these registers is mainly specialized for one circuit and for that reason they will be described along with the circuit they are in control of.

The core (CPU) registers of the PIC16F887 microcontroller are described in this chapter. Since their bits control several different circuits within the chip, it is not possible to classify them into some special group. These bits are described along with the processes they control.

WREG register

In the CPU, registers are used to store information temporarily. That information could be a byte of data to be processed, or an address pointing to the data to be fetched. The vast majority of PIC registers are 8-bit registers. In the PIC there is only one data type: 8-bit. The 8 bits of a register are shown in the diagram below. These range from the MSB (most-significant bit) D7 to the LSB (least-significant bit) D0. With an 8-bit data type, any data larger than 8 bits must be broken into 8-bit chunks before it is processed.



STATUS Register

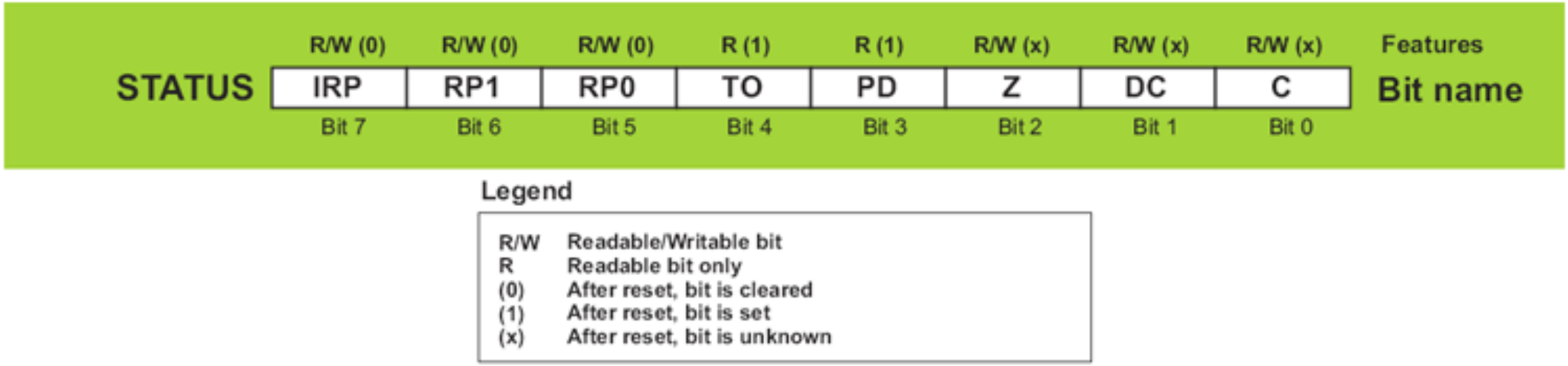


Fig. 2-1 STATUS Register

The STATUS register contains: the arithmetic status of the W register, the RESET status and the bank select bits for data memory. One should be careful when writing a value to this register because if you do it wrong, the results may be different than expected. For example, if you try to clear all bits using the `CLRF STATUS` instruction, the result in the register will be 000xx1xx instead of the expected 00000000. Such errors occur because some of the bits of this register are set or cleared according to the hardware as well as because the bits 3 and 4 are readable only. For these reasons, if it is required to change its content (for example, to change active bank), it is recommended to use only instructions which do not affect any Status bits (C, DC and Z). Refer to “Instruction Set Summary”.

	R/W (0)	R/W (0)	R/W (0)	R (1)	R (1)	R/W (x)	R/W (x)	R/W (x)	Features
STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

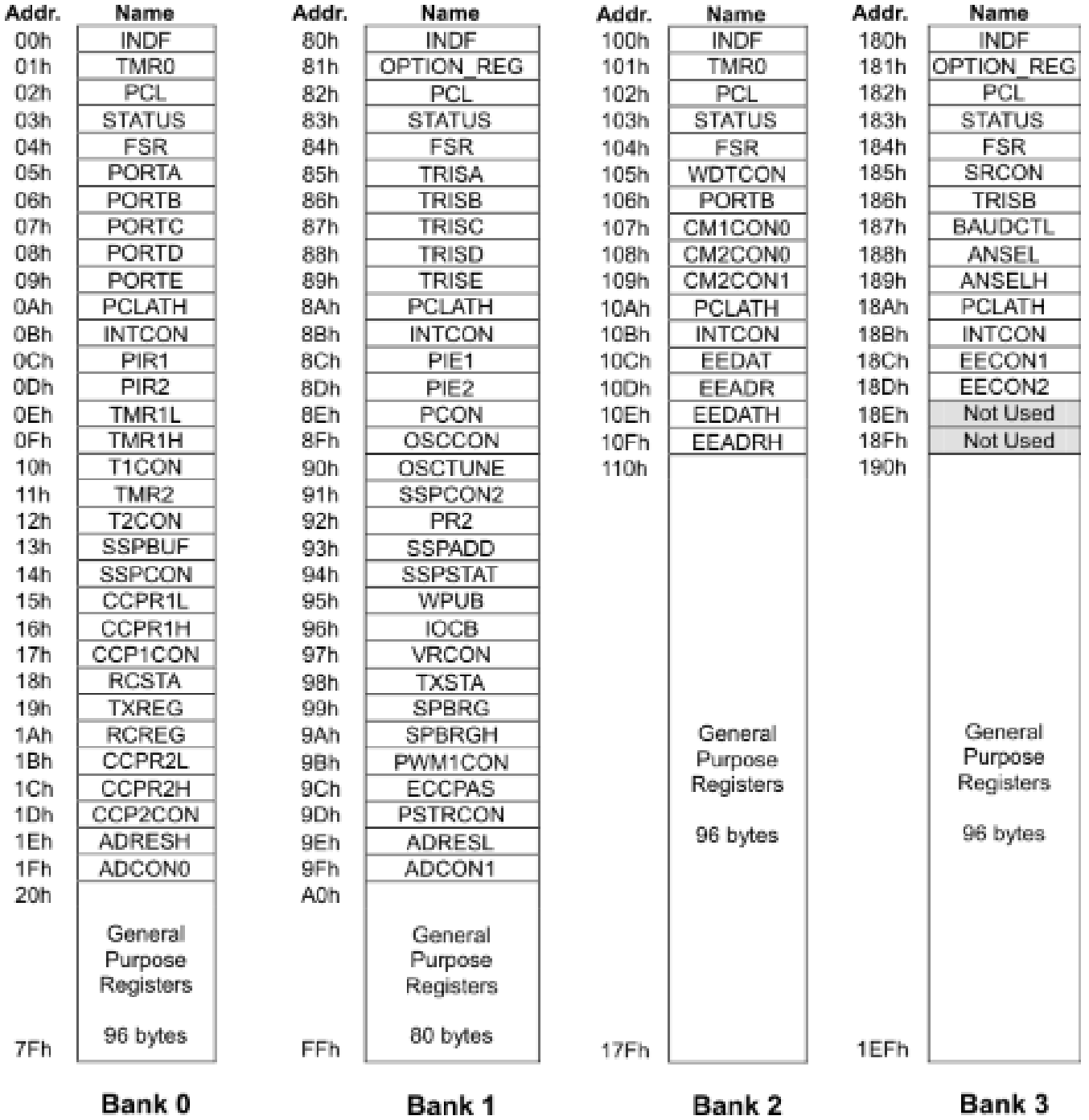
- IRP - Bit selects register bank. It is used for indirect addressing.
  - 1 - Banks 0 and 1 are active (memory location 00h-FFh)
  - 0 - Banks 2 and 3 are active (memory location 100h-1FFh)

- RP1,RP0 - Bits select register bank. They are used for direct addressing.

RP1	RP0	Active Bank
0	0	Bank0
0	1	Bank1
1	0	Bank2
1	1	Bank3

RP1	RP0	Bank	Address	Total	Function
0	0	0	00 – 20	32	Special function registers
			20 – 7F	96	General purpose registers
0	1	1	80 – 9F	32	SFRs, some repeat
			A0 – EF	80	GPRs
			F0 – FF	16	Repeat 70-7F
1	0	2	100 – 10F	16	SFRs, some repeat
			110 – 16F	96	GPRs
			170 – 17F	16	Repeat 70-7F
1	1	3	180 – 18F	16	SFRs, some repeat
			190 – 1EF	96	GPRs
			1F0 – 1FF	16	Repeat 70-7F
			000 – 1FF	96	SFRs
				368	GPRs

Table 1.5 Register bank select


Addr.
Name
00h INDF
01h TMR0
02h PCL
03h STATUS
04h FSR
05h PORTA
06h PORTB
07h PORTC
08h PORTD
09h PORTE
0Ah PCLATH
0Bh INTCON
0Ch PIR1
0Dh PIR2
0Eh TMR1L
0Fh TMR1H
10h T1CON
11h TMR2
12h T2CON
13h SSPBUF
14h SSPCON
15h CCPR1L
16h CCPR1H
17h CCP1CON
18h RCSTA
19h TXREG
1Ah RCREG
1Bh CCPR2L
1Ch CCPR2H
1Dh CCP2CON
1Eh ADRESH
1Fh ADCON0
20h
General Purpose Registers
96 bytes
7Fh
Bank 0
Addr.
Name
80h INDF
81h OPTION_REG
82h PCL
83h STATUS
84h FSR
85h TRISA
86h TRISB
87h TRISC
88h TRISD
89h TRISE
8Ah PCLATH
8Bh INTCON
8Ch PIE1
8Dh PIE2
8Eh PCON
8Fh OSCCON
90h OSCTUNE
91h SSPCON2
92h PR2
93h SSPADD
94h SSPSTAT
95h WPUB
96h IOCB
97h VRCON
98h TXSTA
99h SPBRG
9Ah SPBRGH
9Bh PWM1CON
9Ch ECCPAS
9Dh PSTRCON
9Eh ADRESL
9Fh ADCON1
A0h
General Purpose Registers
80 bytes
FFh
Bank 1
Addr.
Name
100h INDF
101h TMR0
102h PCL
103h STATUS
104h FSR
105h WDTCON
106h PORTB
107h CM1CON0
108h CM2CON0
109h CM2CON1
10Ah PCLATH
10Bh INTCON
10Ch EEDAT
10Dh EEADR
10Eh EEDATH
10Fh EEADRH
110h
General Purpose Registers
96 bytes
17Fh
Bank 2
Addr.
Name
180h INDF
181h OPTION_REG
182h PCL
183h STATUS
184h FSR
185h SRCON
186h TRISB
187h BAUDCTL
188h ANSEL
189h ANSELH
18Ah PCLATH
18Bh INTCON
18Ch EECON1
18Dh EECON2
18Eh Not Used
18Fh Not Used
190h
General Purpose Registers
96 bytes
1EFh
Bank 3

	R/W (0)	R/W (0)	R/W (0)	R (1)	R (1)	R/W (x)	R/W (x)	R/W (x)	Features
STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- TO - Time-out bit.
  - 1 - After power-on or after executing `CLRWDT` instruction which resets watch-dog timer or SLEEP instruction which sets tl microcontroller into low-consumption mode.
  - 0 - After watch-dog timer time-out has occurred.

- PD - Power-down bit.
  - 1 - After power-on or after executing `CLRWDT` instruction which resets watch-dog timer.
  - 0 - After executing `SLEEP` instruction which sets the microcontroller into low-consumption mode.

- **Z** - Zero bit
  - **1** - The result of an arithmetic or logic operation is zero.
  - **0** - The result of an arithmetic or logic operation is different from zero.

- DC - **Digit carry/borrow** bit is changed during addition and subtraction if an "overflow" or a "borrow" of the result occurs.
  - **1** - A carry-out from the 4th low-order bit of the result has occurred.
  - **0** - No carry-out from the 4th low-order bit of the result has occurred.

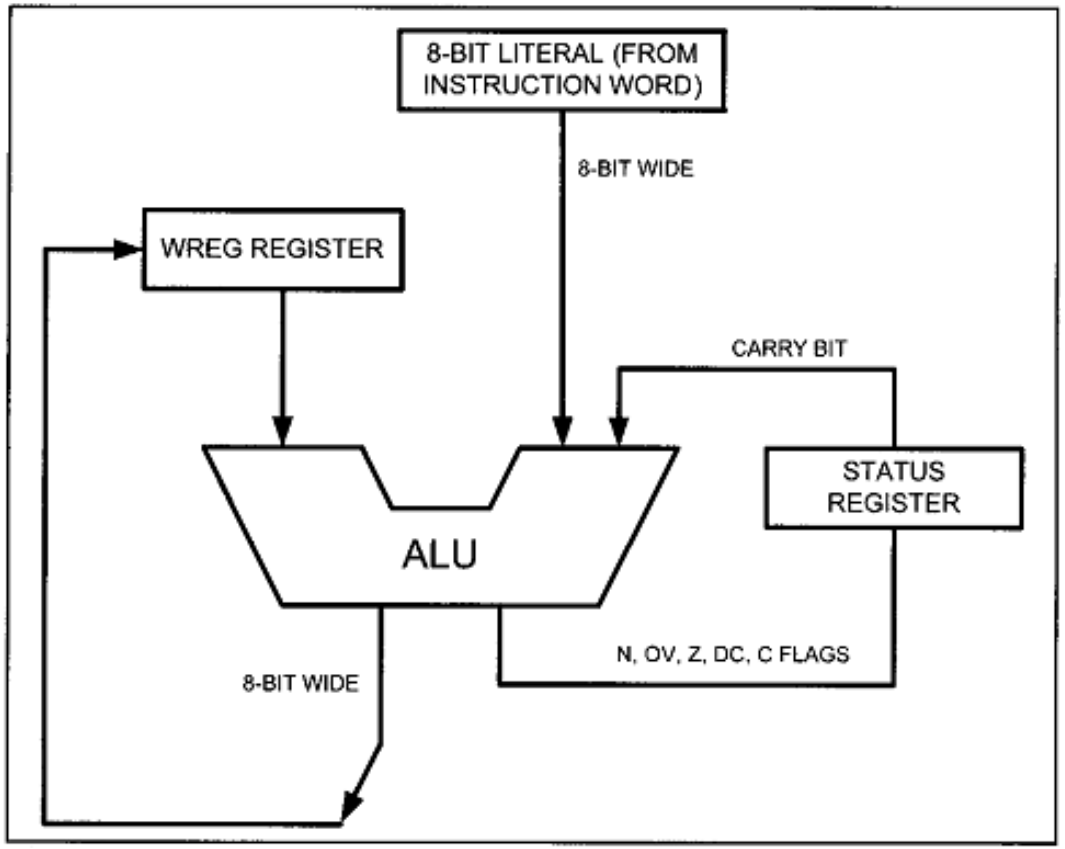
- C - **Carry/Borrow bit** is changed during addition and subtraction if an "overflow" or a "borrow" of the result occurs, i.e. if the result is greater than 255 or less than 0.
  - **1** - A carry-out from the most significant bit of the result has occurred.
  - **0** - No carry-out from the most significant bit of the result has occurred.

PIC WREG and ALU Using Literal Value

MOVLW instruction

Simply stated, the MOVLW instruction moves 8-bit data into the WREG register. It has the following format:

```
MOVLW K      ;move literal value K into WREG
```




```
MOVLW 12H    ;load value 12H into WREG (WREG = 12H)
ADDLW 16H    ;add 16 to WREG (WREG = 28H)
ADDLW 11H    ;add 11 to WREG (WREG = 39H)
ADDLW 43H    ;add 43 to WREG (WREG = 7CH)
```

ADDLW instruction and the status register

Next we examine the impact of the ADDLW instruction on the flag bits C, DC, and Z of the status register. Some examples should clarify their meanings.

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

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.

Example 2-10

Show the status of the C, DC, and Z flags after the addition of 88H and 93H in the following instructions:

```
MOVLW 88H
ADDLW 93H          ;add 93H to WREG
```

Solution:

88H	1000 1000	
+ 93H	1001 0011	
11BH	0001 1011	WREG = 1BH

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

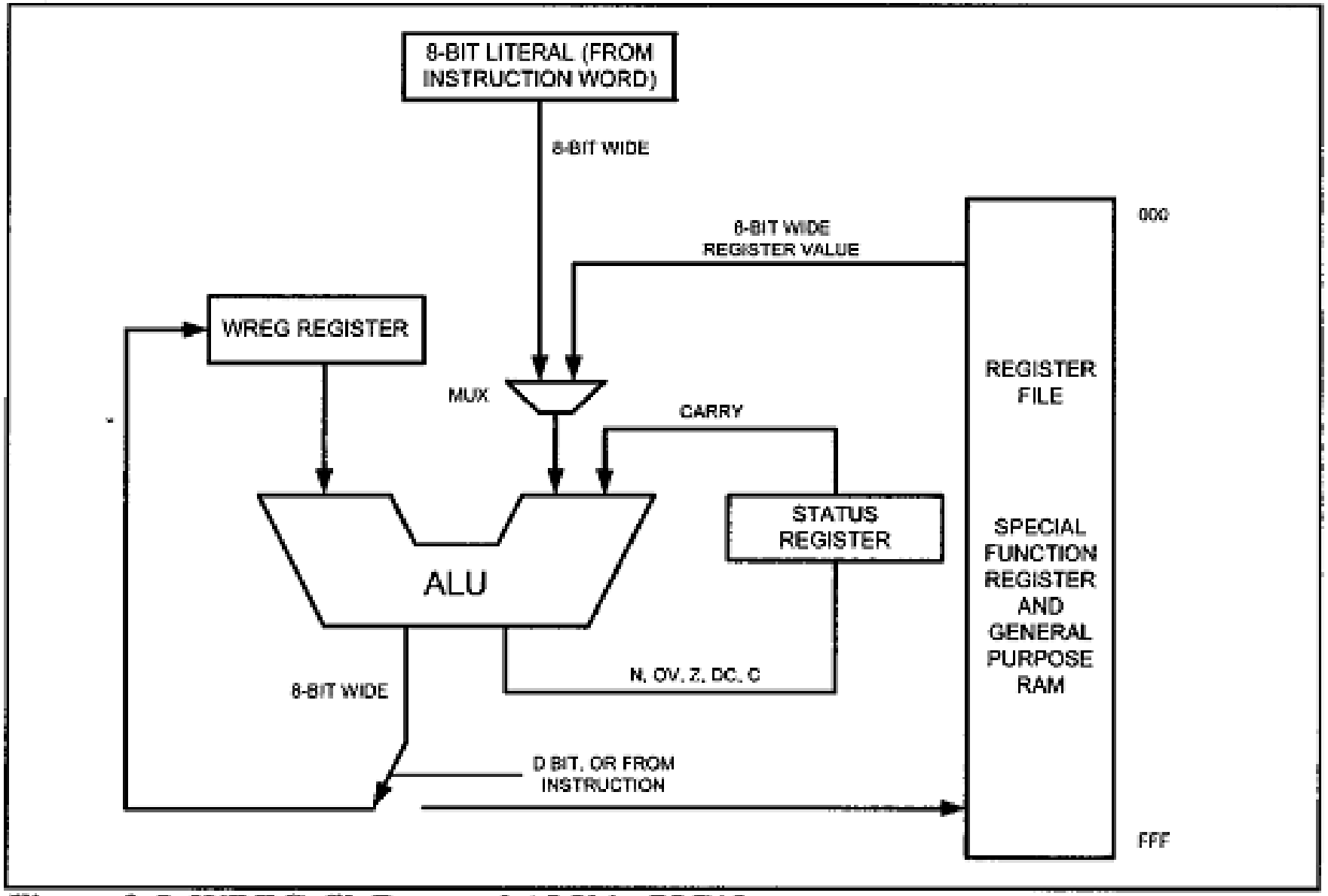


Figure 2-5. WREG, fileReg, and ALU in PIC18

Data format representation

There are four ways to represent a byte of data in the PIC assembler. The numbers can be in hex, binary, decimal, or ASCII formats. The following are examples of how each work.

Hex numbers

There are four ways to show hex numbers:

1. We can use h (or H) right after the number like this: `MOVLW 99H`
2. Put 0x (or 0X) in front of the number like this: `MOVLW 0x99`
3. Put nothing in front or back of the number like this: `MOVLW 99`
4. Put h in front of the number, but with single quotes around the number like this: `MOVLW h'99'`

Here are a few lines of code that use the hex format:

```
MOVLW 25      ;WREG = 25H
ADDLW 0x11    ;WREG = 25H + 11H = 36H
ADDLW 12H     ;WREG = 36H + 12H = 48H
ADDLW H'2A'   ;WREG = 48H + 2AH = 72H
ADDLW 2CH     ;WREG = 72H + 2CH = 9EH
```

The following are invalid:

```
MOVLW E5H     ;invalid, it must be MOVLW 0E5H
ADDLW C6      ;invalid, it must be ADDLW 0C6
```

Binary numbers

There is only one way to represent binary numbers in a PIC assembler. It is as follows:

```
MOVLW B'10011001' ;WREG = 10011001 or 99 in hex
```

The lowercase b will also work. Note that ' is the single quote key, which is on the same key as the double quote ". This is different from other assemblers such as the 8051 and x86. Here are some examples of how to use it:

```
MOVLW B'00100101' ;WREG = 25H
ADDLW B'00010001' ;WREG = 25H + 11H = 36H
```

Decimal numbers

There are two ways to represent decimal numbers in a PIC assembler. One way is as follows:

```
MOVLW D'12'          ;WREG = 00001100 or 0C in hex


MOVLW .12            ;WREG = 00001100 = 0CH = 12
```

ASCII character

To represent ASCII data in a PIC assembler we use the letter A as follows:

```
MOVLW A'2'   ;WREG = 00110010 or 32 in hex (See Appendix F)


MOVLW A'9'   ;WREG = 39H, which is hex number for ASCII '9'
ADDLW A'1'   ;WREG = 39H + 31H = 70H
             ;(31 hex is for ASCII '1')
MOVLW '9'    ;WREG = 39H another way for ASCII
```

ROM memory map

THE PIC FILE REGISTER

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

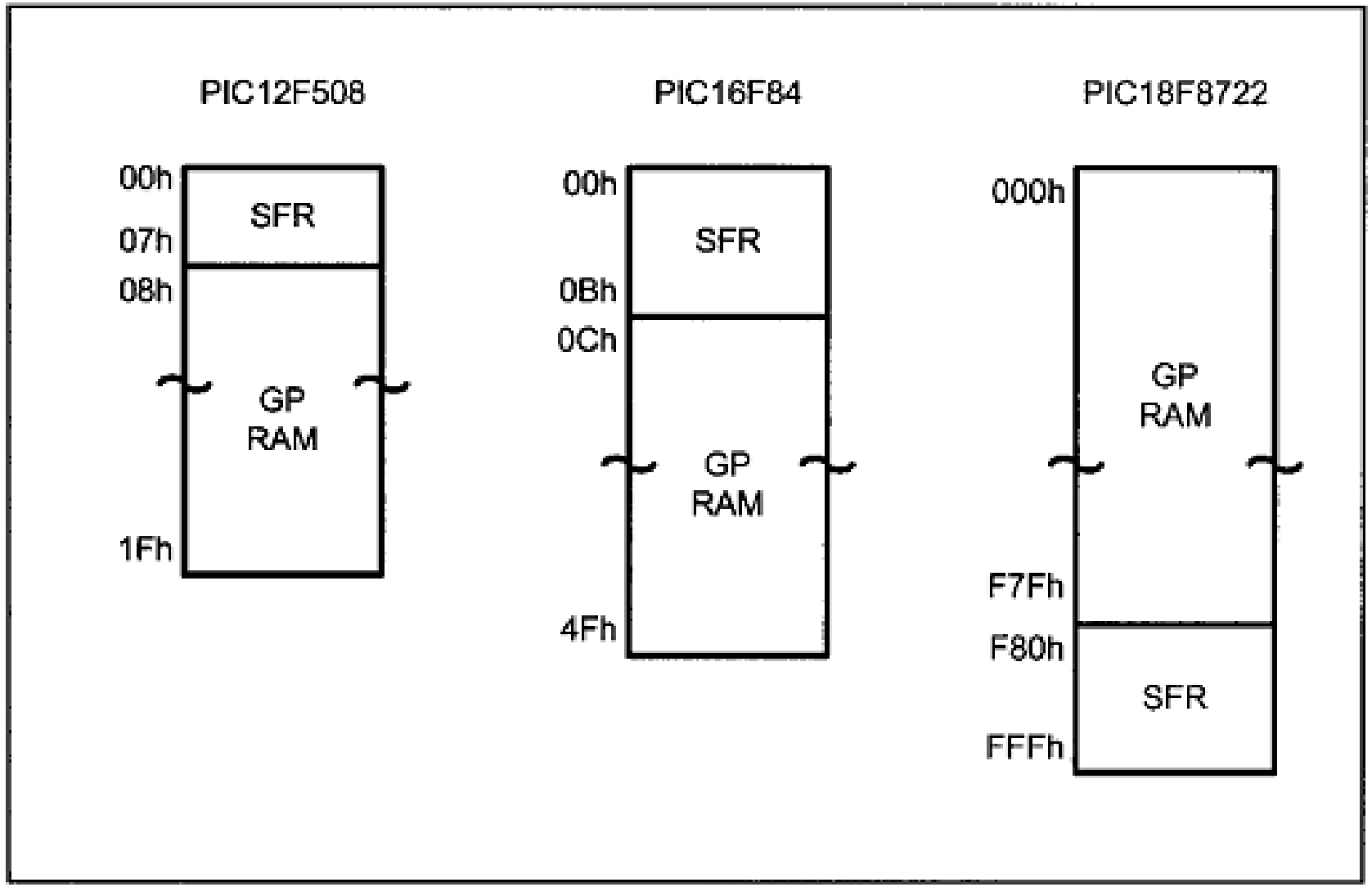


Figure 2-2. File Registers of PIC12, PIC16, and PIC18

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.

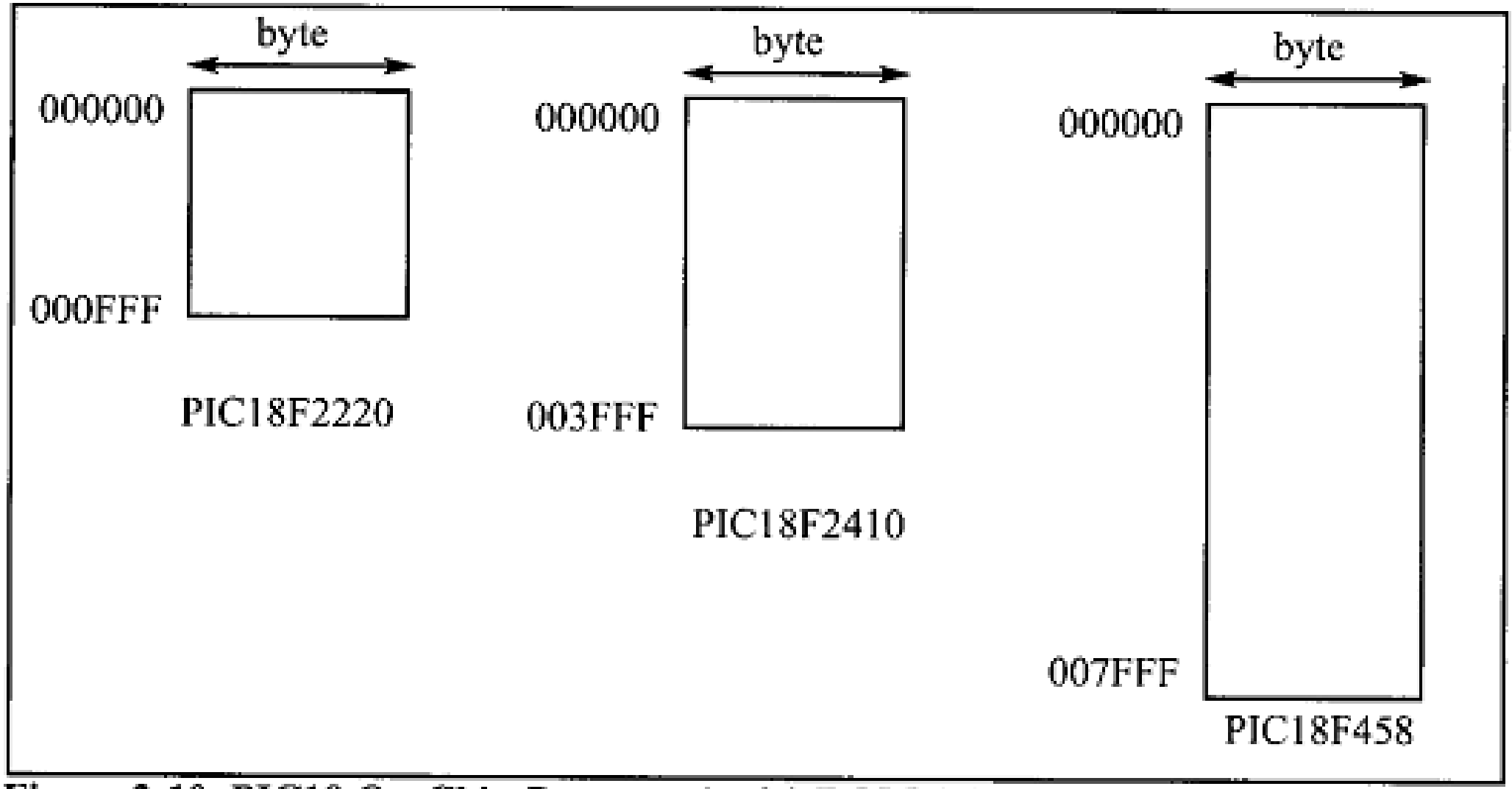


Figure 2-10. PIC18 On-Chip Program (code) ROM Address Range

PCL and PCLATH Registers

The size of the program memory of the PIC16F887 is 8K. Therefore, it has 8192 locations for program storing. For this reason the program counter must be 13-bits wide (2^13 = 8192). In order that the contents of some location may be changed in software during operation, its address must be accessible through some SFR. Since all SFRs are 8-bits wide, this register is "artificially" created by dividing its 13 bits into two independent registers: PCLATH and PCL.

If the program execution does not affect the program counter, the value of this register is automatically and constantly incremented +1, +1, +1, +1... In that way, the program is executed just as it is written- instruction by instruction, followed by a constant address increment.

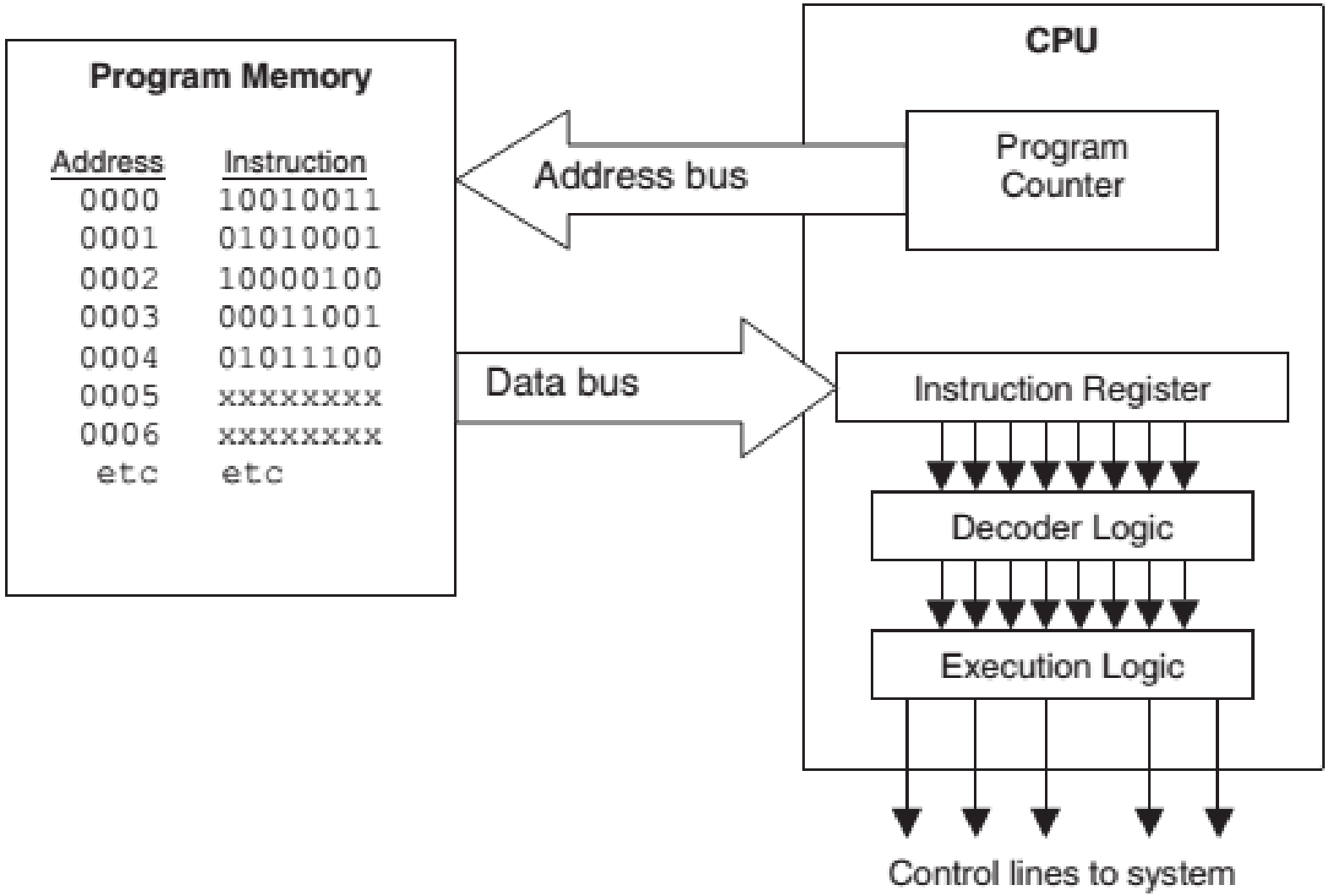
Fig. 2-16 PCL and PCLATH Registers

If the program counter is changed in software, then there are several things that should be kept in mind in order to avoid problems:

- Eight lower bits (the low byte) come from the PCL register which is readable and writable, whereas five upper bits coming from the PCLATH register are writable only.
- The PCLATH register is cleared on any reset.

- In assembly language, the value of the program counter is marked with PCL, but it obviously refers to 8 lower bits only. One should take care when using the "`ADDWF PCL`" instruction. This is a jump instruction which specifies the target location by adding some number to the current address. It is often used when jumping into a look-up table or program branch table to read them. A problem arises if the current address is such that addition causes change on some bit belonging to the higher byte of the PCLATH register. Do you see what is going on?

Executing any instruction upon the PCL register simultaneously causes the Prog ram Counter bits to be replaced by the contents of the PCLATH register. However, the PCL register has access to only 8 lower bits of the instruction result and the following jump will be completely incorrect. The problem is solved by setting such instructions at addresses ending by xx00h. This enables the program to jump up to 255 locations. If longer jumps are executed by this instruction, the PCLATH register must be incremented by 1 for each PCL register overflow.



1.2 Processor program execution

- On subroutine call or jump execution (instructions CALL and GOTO), the microcontroller is able to provide only 11-bit addressing. For this reason, similar to RAM which is divided in “banks”, ROM is divided in four “pages” in size of 2K each. Such instructions are executed within these pages without any problems. Simply, since the processor is provided with 11-bit address from the program, it is able to address any location within 2KB. Figure 2-17 below illustrates this situation as a jump to the subroutine PP1 address.

However, if a subroutine or jump address are not within the same page as the location from where the jump is, two “missing”-higher bits should be provided by writing to the PCLATH register. It is illustrated in figure 2-17 below as a jump to the subroutine PP2 address.

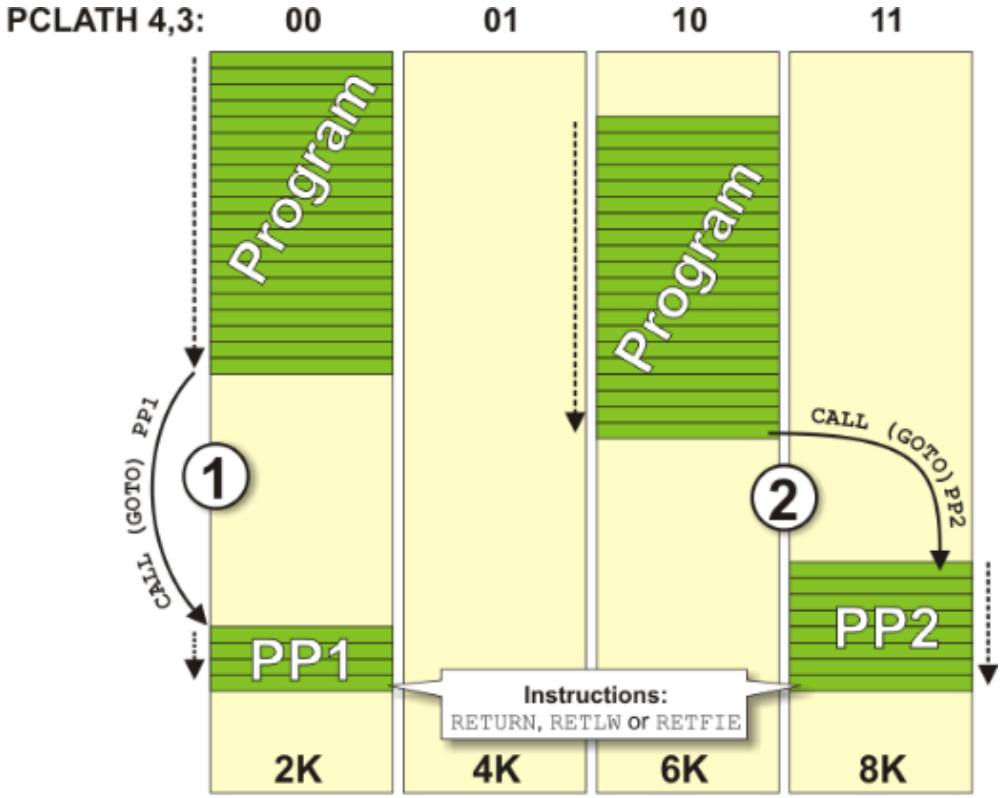
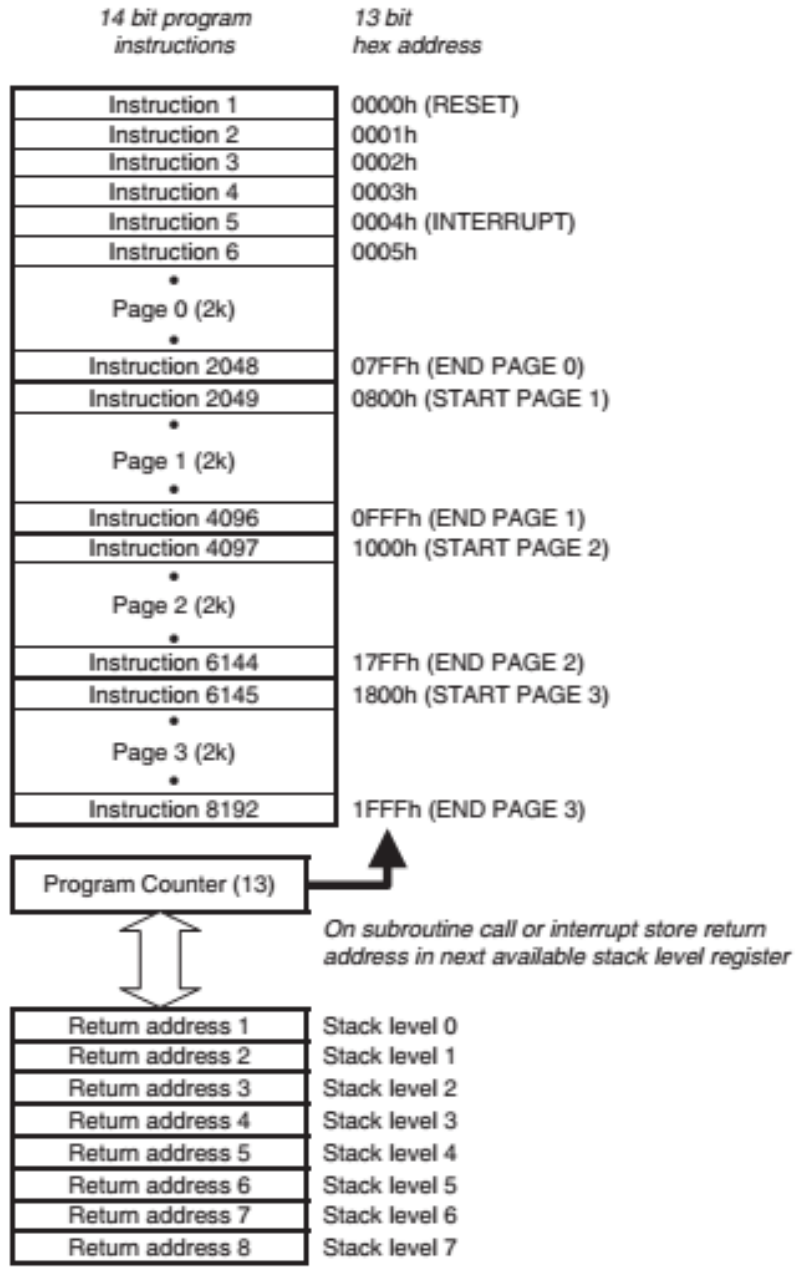


Fig. 2-17 PCLATH Registers



P16F877 program memory and stack

Indirect addressing

In addition to direct addressing which is logical and clear by itself (it is sufficient to specify address of some register to read its contents), this microcontroller is able to perform indirect addressing by means of the INDF and FSR registers. It sometimes considerably simplifies program writing. The whole procedure is enabled because the INDF register is not true one (physically does not exist), but only specifies the register whose address is located in the FSR register. Because of this, write or read from the INDF register actually means write or read from the register whose address is located in the FSR register. In other words, registers' addresses are specified in the FSR register, and their contents are stored in the INDF register. The difference between direct and indirect addressing is illustrated in the figure 2-18 below:

As seen, the problem with the "missing addressing bits" is solved by "borrowing" from another register. This time, it is the seventh bit called IRP from the STATUS register.

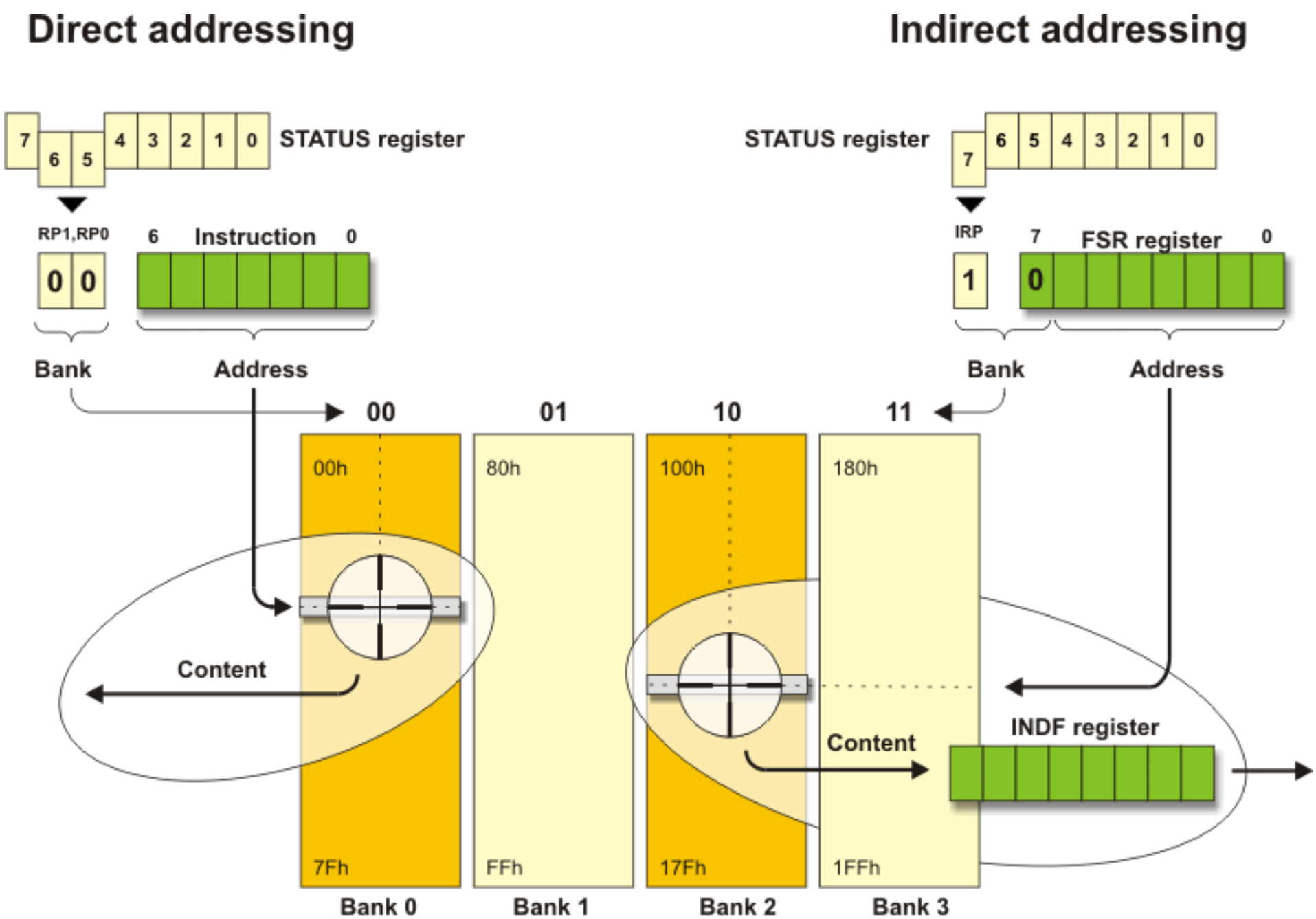


Fig. 2-18 Direct and Indirect addressing

Indirect File Register Addressing

File register 00 (INDF) is used for indirect file register addressing. The address of the register is placed in the file select register (FSR). When data is written to or read from INDF, it is actually written to or read from the file register pointed to by FSR. This is most useful for carrying out a read or write on a continuous block of GPRs, for example, when saving data being read in from a port over a period of time. Since nine bits are needed to address all file registers (000–1FF), the IRP bit in the status register is used as the extra bit. Direct and indirect addressing of the file registers are compared in the data sheet (Figure 2–6).

Instruction Set

It has been already mentioned that microcontrollers differs from other integrated circuits. Most of them are ready for installation into the target device just as they are, this is not the case with the microcontrollers. In order that the microcontroller may operate, it needs precise instructions on what to do. In other words, a program that the microcontroller should execute must be written and loaded into the microcontroller. This chapter covers the commands which the microcontroller "understands". The instruction set for the 16FXX includes 35 instructions in total. Such a small number of instructions is specific to the RISC microcontroller because they are well-optimized from the aspect of operating speed, simplicity in architecture and code compactness. The only disadvantage of RISC architecture is that the programmer is expected to cope with these instructions.

Instruction	Description	Operation	Flag	CLK	*
Data Transfer Instructions					
MOVLW k	Move constant to W	k -> w		1	
MOVWF f	Move W to f	W -> f		1	
MOVF f,d	Move f to d	f -> d	Z	1	1, 2
CLRW	Clear W	0 -> W	Z	1	
CLRF f	Clear f	0 -> f	Z	1	2
SWAPF f,d	Swap nibbles in f	f(7:4),(3:0) -> f(3:0),(7:4)		1	1, 2

Instruction	Description	Operation	Flag	CLK	*
Arithmetic-logic Instructions					
ADDLW k	Add W and constant	W+k -> W	C, DC, Z	1	
ADDWF f,d	Add W and f	W+f -> d	C, DC ,Z	1	1, 2
SUBLW k	Subtract W from constant	k-W -> W	C, DC, Z	1	
SUBWF f,d	Subtract W from f	f-W -> d	C, DC, Z	1	1, 2
ANDLW k	Logical AND with W with constant	W AND k -> W	Z	1	
ANDWF f,d	Logical AND with W with f	W AND f -> d	Z	1	1, 2
ANDWF f,d	Logical AND with W with f	W AND f -> d	Z	1	1, 2
IORLW k	Logical OR with W with constant	W OR k -> W	Z	1	
IORWF f,d	Logical OR with W with f	W OR f -> d	Z	1	1, 2
XORWF f,d	Logical exclusive OR with W with constant	W XOR k -> W	Z	1	1, 2
XORLW k	Logical exclusive OR with W with f	W XOR f -> d	Z	1	
INCF f,d	Increment f by 1	f+1 -> f	Z	1	1, 2
DECF f,d	Decrement f by 1	f-1 -> f	Z	1	1, 2
RLF f,d	Rotate left f through CARRY bit		C	1	1, 2
RRF f,d	Rotate right f through CARRY bit		C	1	1, 2
COMF f,d	Complement f	f -> d	Z	1	1, 2

Bit-oriented Instructions					
BCF f,b	Clear bit b in f	0 -> f(b)		1	1, 2
BSF f,b	Clear bit b in f	1 -> f(b)		1	1, 2
Program Control Instructions					
BTFSC f,b	Test bit b of f. Skip the following instruction if clear.	Skip if f(b) = 0		1 (2)	3
BTFSS f,b	Test bit b of f. Skip the following instruction if set.	Skip if f(b) = 1		1 (2)	3
DECFSZ f,d	Decrement f. Skip the following instruction if clear.	f-1 -> d skip if Z = 1		1 (2)	1, 2, 3
INCFSZ f,d	Increment f. Skip the following instruction if set.	f+1 -> d skip if Z = 0		1 (2)	1, 2, 3
GOTO k	Go to address	k -> PC		2	
CALL k	Call subroutine	PC -> TOS, k -> PC		2	
RETURN	Return from subroutine	TOS -> PC		2	
RETLW k	Return with constant in W	k -> W, TOS -> PC		2	
RETFIE	Return from interrupt	TOS -> PC, 1 -> GIE		2	

Other instructions					
NOP	No operation	TOS -> PC, 1 -> GIE		1	
CLRWDT	Clear watchdog timer	0 -> WDT, 1 -> TO, 1 -> PD	TO, PD	1	
SLEEP	Go into sleep mode	0 -> WDT, 1 -> TO, 0 -> PD	TO, PD	1	

Table 9-1 16Fxx Instruction Set

*1 When an I/O register is modified as a function of itself, the value used will be that value present on the pins themselves.

*2 If the instruction is executed on the TMR register and if d=1, the prescaler will be cleared.

*3 If the PC is modified or test result is logic one (1), the instruction requires two cycles.

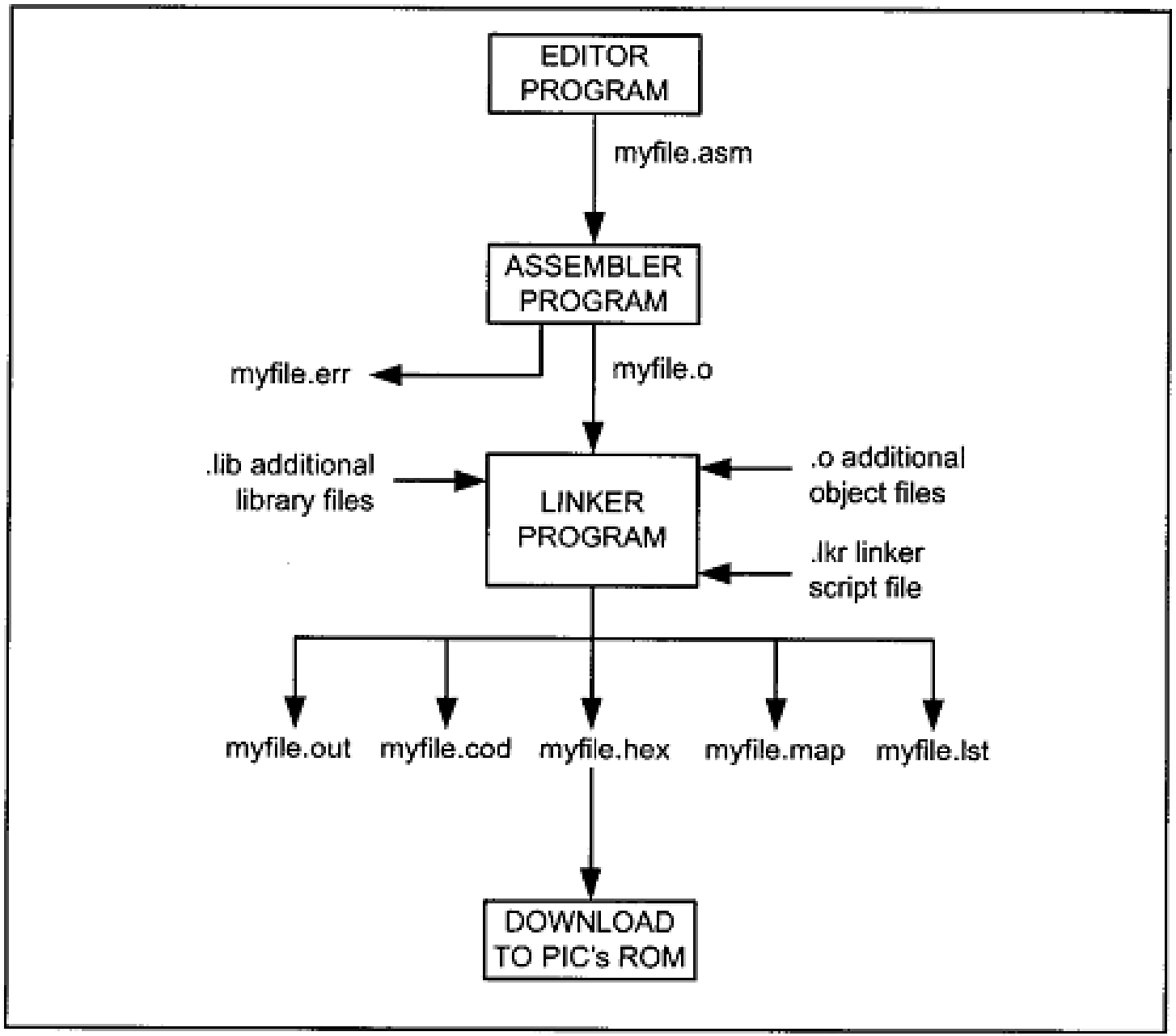


Figure 2-8. Steps to Create a Program

The program is written as a source code (a simple text file) on a PC host computer. Any text editor such as Notepad™ can be used, but an editor is provided with the standard PIC development system software MPLAB (downloadable from www.microchip.com). The instructions are selected from the pre-defined PIC instruction set (Table 13-2 in the data sheet) according to the operational sequence required. The source code file is saved as PROGNAME.ASM. More details of the assembler program syntax are given later.

The source code is assembled (converted into machine code) by the assembler program MPASM, which creates the list of binary instruction codes. As this is normally displayed as hexadecimal numbers, it is saved as PROGNAME.HEX. This is then downloaded to the PIC chip from the PC by placing the MCU in a programming unit which is attached to the serial port of PC, or by connecting the chip to a programmer after fitting it in the application board (in-circuit programming). The hex code is transferred in serial form via Port B into the PIC flash program memory. A list file is created by the assembler, which shows the source code and machine code in one text file. The list file for a simple program which outputs a binary count at Port B is shown in Program 1.1.

Memory Address	Hex Code	Line Number	Address Label	Operation Mnemonic	Operand
		00001		PROCESSOR	16F877
		00002			
0000	3000	00003		MOVLW	00
0001	0066	00004		TRIS	06
		00005			
0002	0186	00006		CLRF	06
0003	0A86	00007	again	INCF	06
0004	2803	00008		GOTO	again
		00009			
		00010		END	

Notes:

The program listing includes the source code at the right, with source line numbers, the hex machine code and the memory location where each instruction is stored (0000–0004). Notice that some statements are assembler directives, not instructions: PROCESSOR to specify the MCU type and END to terminate the source code. These are not converted into machine code.

The ‘877 has 8k of program memory, that is, it can store a maximum of $1024 \times 8 = 8192$ 14-bit instructions. By default, it is loaded, and starts executing, from address zero. In real-time (control) applications, the program runs continuously, and therefore loops back at the end. If it does not, be careful – it will run through the blank locations and start again at the beginning!

Notes:

Let us look at a typical instruction to see how the program instructions are executed.

Source code: MOVLW 05A

Hex code: 305A (4 hex digits)

Binary code: 0011 0000 0101 1010 (16 bits)

Instruction: 11 00xx kkkk kkkk (14 bits)

The instruction means: Move a Literal (given number, 5Ah) into the Working register.

PIC INSTRUCTION SET

F = Any file register (specified by address or label), example is labelled GPR1
L = Literal value (follows instruction), example is labelled num1
W = Working register, W (default label)
Labels Register labels must be declared in include file or by register label equate (e.g. GPR1 EQU 0C)
Bit labels must be declared in include file or by bit label equate (e.g. bit1 EQU 3)
Address labels must be placed at the left margin of the source code file (e.g. start, delay)

Operation	Example	
Move		
Move data from F to W	`MOVF`	`GPR1,W`
Move data from W to F	`MOVWF`	`GPR1`
Move literal into W	`MOVLW`	`num1`
Test the register data	`MOVF`	`GPR1,F`

Register

Clear W (reset all bits and value to 0)	`CLRW`	
Clear F (reset all bits and value to 0)	`CLRF`	`GPR1`
Decrement F (reduce by 1)	`DECF`	`GPR1`
Increment F (increase by 1)	`INCF`	`GPR1`
Swap the upper and lower four bits in F	`SWAPF`	`GPR1`
Complement F value (invert all bits)	`COMF`	`GPR1`
Rotate bits Left through carry flag	`RLF`	`GPR1`
Rotate bits Right through carry flag	`RRF`	`GPR1`
Clear (= 0) the bit specified	`BCF`	`GPR1,but1`
Set (= 1) the bit specified	`BSF`	`GPR1,but1`

Arithmetic

Add W to F, with carry out	`ADDWF`	`GPR1`
Add F to W, with carry out	`ADDWF`	`GPR1,W`
Add L to W, with carry out	`ADDLW`	`num1`
Subtract W from F, using borrow	`SUBWF`	`GPR1`
Subtract W from F, placing result in W	`SUBWF`	`GPR1,W`
Subtract W from L, placing result in W	`SUBLW`	`num1`

Logic

AND the bits of W and F, result in F	`ANDWF`	`GPR1`
AND the bits of W and F, result in W	`ANDWF`	`GPR1,W`
AND the bits of L and W, result in W	`ANDLW`	`num1`
OR the bits of W and F, result in F	`IORWF`	`GPR1`
OR the bits of W and F, result in W	`IORWF`	`GPR1,W`
OR the bits of L and W, result in W	`IORLW`	`num1`
Exclusive OR the bits of W and F, result in F	`XORWF`	`GPR1`
Exclusive OR the bits of W and F, result in W	`XORWF`	`GPR1,W`
Exclusive OR the bits of L and W	`XORLW`	`num1`

Test & Skip

Test a bit in F and Skip next instruction if it is Clear (= 0)	`BTFSC`	`GPR1,but1`
Test a bit in F and Skip next instruction if it is Set (= 1)	`BTFSS`	`GPR1,but1`
Decrement F and Skip next instruction if F = 0	`DECFSZ`	`GPR1`
Increment F and Skip next instruction if F = 0	`INCFSZ`	`GPR1`

Jump

Go to a labelled line in the program	`GOTO`	`start`
Jump to the label at the start of a subroutine	`CALL`	`delay`
Return at the end of a subroutine to the next instruction	`RETURN`	
Return at the end of a subroutine with L in W	`RETLW`	`num1`
Return From Interrupt service routine	`RETFIE`	

Control

No Operation - delay for 1 cycle	`NOP`
Go into standby mode to save power	`SLEEP`
Clear watchdog timer to prevent automatic reset	`CLRWDT`

Note 1: For MOVE instructions data is copied to the destination but retained in the source register.

Note 2: General Purpose Register 1, labelled ' GPR1', represents all file registers (00-4F). Literal value 'num1' represents all 8-bit values 00-FF. File register bits 0–7 are represented by the label 'but1'.

Note 3: The result of arithmetic and logic operations can generally be stored in W instead of the file register by adding, 'W' to the instruction. The full syntax for register operations with the result remaining in the file register F is ADDWF GPR1,F etc. F is the default destination, and W the alternative, so the instructions above are shortened to ADDWF, GPR1, etc. This will generate a message from the assembler that the default destination will be used.