THE WREG REGISTER IN THE PIC

PIC microcontrollers have many registers for arithmetic and logic operations. Among them is the WREG register. Because there are a large number of registers inside the PIC, we will concentrate on the widely used register WREG in this section. General-purpose registers and special function registers re covered later.

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 range goes 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.

The 8-bit WREG register is the most widely used register in the PIC micro controller. WREG stands for working register, as there is only one. The WREG register is the same as the accumulator in other microprocessors. The WREG register is used for all arithmetic and logic instructions. To understand the use of the WREG register, we will show it in the context of two simple instructions: MOVE and ADD.

MOVLW Instruction

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

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

K is an 8-bit value that can range from 0-255 in decimal, or 00-FF in hex. The L stands for literal, which means, literally, a number must be used. In other words, if we see the word literal in any instruction, we are dealing with an actual value that must be provided right there with the instruction. This is similar to the immediate value we see in other microprocessors. Notice that in MOVLW, the letter L (literal) comes first and then the letter W (WREG), which means "move a literal value to WREG, " the destination. The following instruction loads the WREG register with a literal value of 25H (i.e., 25 in hex).

```
MOVLW    25H        ; move value 25H into WREG (WREG = 25H)
```

The following instruction loads the WREG register with value 87H (87 in hex).

```
MOVLW    87H        ; load 87H into WREG (WREG = 87H)
```

The following instruction loads the WREG register with value 15H (15 in hex and 21 in decimal).

```
MOVLW    15H        ; load 15H into WREG (WREG = 15H)
```

ADDLW Instruction

The ADDLW instruction has the following format:

```
ADDLW    K          ; ADD literal value K to WREG
```

The ADD instruction tells the CPU to add the literal value K to register WREG and put the result back in the WREG register. Notice that in ADDLW, first comes the letter L (literal) and then the letter W (WREG), which means "add a literal value to WREG," the destination. To add two numbers such as 25H and 34H, one can do the following:

```
MOVLW   25H        ; load 25H  into  WREG
ADDLW   34H        ; add value 34  to W (W = W + 34H)
```

Executing the above lines results in WREG = 59H (25H + 34H = 59H)
Figure 2-1 shows the literal value and WREG being fed to the PIC ALU.

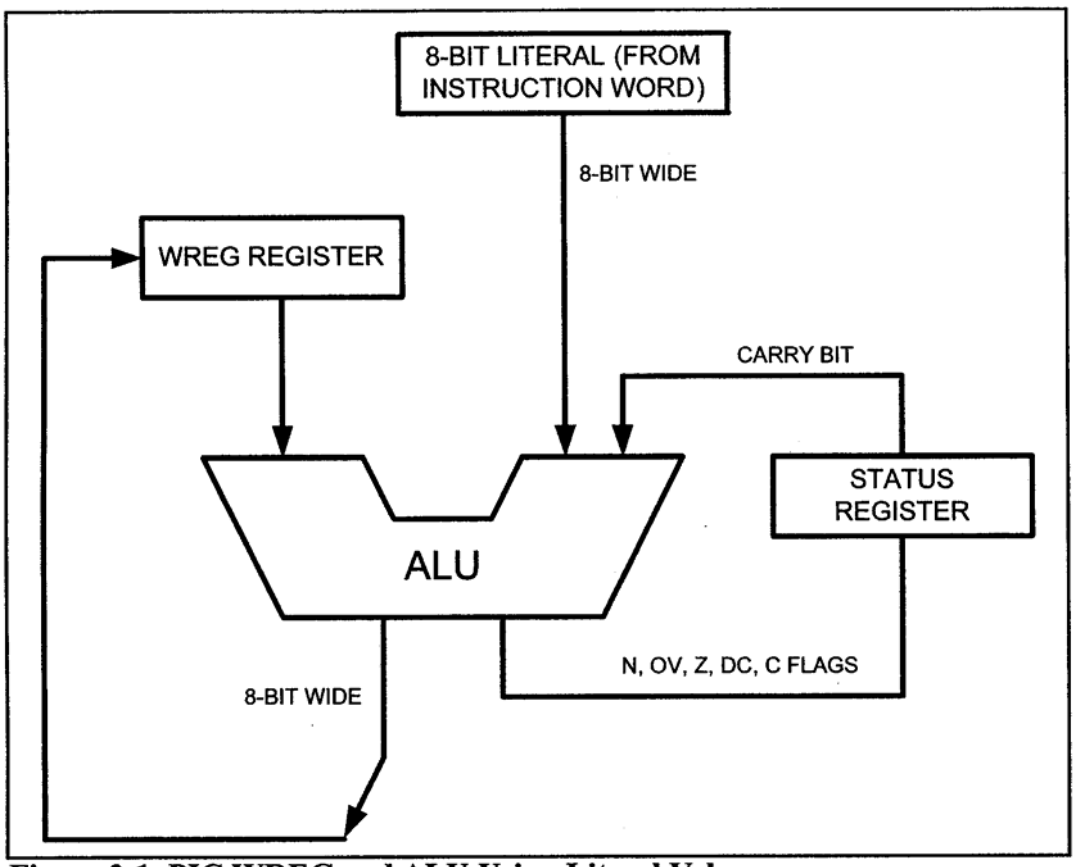

Figure 2-1. PIC WREG and ALU Using Literal Value

The following program will add values 12H, 16H, 31H, and 43H:

```
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)
```

When programming the WREG register of the PIC microcontroller with a literal value, the following points should be noted:

- Values can be loaded directly into the WREG register. There is no need for a preceding pound sign or dollar sign to indicate that a value is an immediate value as is the case with some other microcontrollers.
- If values 0 to F are moved into an 8-bit register such as WREG, the rest of the bits are assumed to be all zeros. For example, in "MOVLW 5H" the result will be WREG = 05H; that is, WREG = 00000101 in binary.

- Moving a value larger than 255 (FF in hex) into the WREG register will truncate the upper byte and cause a warning in the .err file.

```
MOVLW 7F2H  ;ILLEGAL 7F2H > 8 bits (FFH), becomes F2H
MOVLW 456H  ;ILLEGAL 456H > FFH, becomes 56H
MOVLW 60A5H ;ILLEGAL but becomes A5H
```

SECTION 2.2: THE PIC FILE REGISTER

The PIC microcontroller has many other registers in addition to the WREG register. They are called data memory space to distinguish them from program (code) memory space. The data memory space in PIC is a read/write (static RAM) memory. In the PIC microcontroller literature, the data memory is also called the
file register.

File register (data RAM) space allocation in PIC

The file register is read/write memory used by the CPU for data storage, and registers for internal use and functions. As with WREG, we can perform arithmetic and logic operations on many locations of the file register data RAM. The PIC microcontrollers' file register size ranges from 32 bytes to several thousand bytes depending on the chip. Even within the same family, the size of the file register data RAM varies from chip to chip. Notice that the file register data RAM has a byte-size width, just like WREG. The file register data RAM in PIC is divided into two sections:

- Special Function Registers (SFR)
- General-Purpose Registers (GPR)

SFRs (Special Function Registers)

The Special Function Registers (SFRs) are dedicated to specific functions such as ALU status, timers, serial communication, I/O ports, ADC, and so on. The function of each SFR is fixed by the CPU designer at the time of design because it is used for control of the microcontroller or peripheral. The PIC SFRs are 8-bit registers. The number of locations in the file register set aside for SFR depends on the pin numbers and peripheral functions supported by that chip. That number can vary from chip to chip even among members of the same family. Some have as few as 7 (8-pin PIC12C508 with no on-chip analog-to-digital converter) and some have over a hundred (40-pin PIC18F458 with on-chip analog-to digital converter). For example, the more timers we have in a PIC chip, the more SFR registers we will have.

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.

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

```
      ;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

Table 6-1: Selected PIC18 Special Function Register (SFR) Addresses

Symbol	Name	Address
WREG	Working register	FE8H
PORTA	Port A	F80H
PORTB	Port B	F81H
PORTC	Port C	F82H
LATA	Output latch, Port A	F89H
LATB	Output latch, Port B	F8AH
LATC	Output latch, Port C	F8BH
TRISA	Data direction, Port A	F92H
TRISB	Data direction, Port B	F93H
TRISC	Data direction, Port C	F94H
INDF0	Indirect addressing register 0	FEFH
INDF1	Indirect addressing register 1	FE7H
FSR0L	Indirect data memory address pointer 0 low	FE9H
FSR0H	Indirect data memory address pointer 0 high	FEAH
FSR1L	Indirect data memory address pointer 1 low	FE1H
FSR1H	Indirect data memory address pointer 1 high	FE2H
PLUSW0	Indirect indexed address register	FEBH
PREINC0	Preincrement register 0	FECH
POSTDEC0	Post-decrement register 0	FEDH
POSTINC0	Post-increment register 0	FEEH
TBLPTRL	Table pointer, low byte	FF6H
TBLPTRH	Table pointer, high byte	FF7H
TBLPTRU	Table pointer, upper byte	FF8H
TABLAT	Program memory table latch	FF5H
STATUS	Status flag byte	FD8H

SFR registers and their addresses

PIC 18 registers for Ports A, B, and so on are part of the group of registers commonly referred to as SFRs (special function registers). There are many special function registers and they are widely used.

The SFRs can be accessed by their names (which is much easier) or by their addresses. For example, Port B has address F81H, and Port C the address F82H, as shown in Table 6-1. Notice how the following pairs of instructions mean the same thing:

```
MOVWF 0xF81        ;is the same as
MOVWF PORTB        ;which means copy WREG into Port B

CLRF 0xF82         ;is the same as
CLRF PORTC         ;which means clear Port C

BSF 0xFD8,0        ;is the same as
BSF STATUS,C       ;which make C = 1
```

The following two points should be noted about special function registers (SFRs) addresses:

1. The special function registers have addresses between F80H and FFFH. These addresses are below FFFH, because the PIC18 starts assigning SFR addresses at FFFH and goes down until all SFRs

supported by that chip are assigned. Not all the members of the PIC18 family have the same peripherals; therefore, the number of locations used for SFRs varies among the PIC18 family.
2. Not all the address space of F80H to FFFH is used by the SFR. The unused .locations F80H to FFFH are reserved and must not be used by the PICI8 programmer.

Example 6-1

Write code to send 55H to Port B. Include

(a) The register names.
(b) Their addresses.

Solution:

(a)
```
CLRF   TRISB        ;Port B output
MOVLW  0x55         ;WREG = 55H
MOVWF  PORTB        ;Port B = 55H
```

(b) From Table 6-1, TRISB address = F93H and PORTB address = F81H

```
CLRF   0xF93        ;Port B output
MOVLW  0x55H        ;WREG = 55H
MOVWF  0xF81        ;Port B = 55H
```

Regarding direct addressing mode in the PIC18, notice the following

SECTION 6.2: REGISTER INDIRECT ADDRESSING MODE

We can use register direct or register indirect addressing modes to access data stored in the general purpose RAM section of the file register. In the last section we showed how to use direct addressing mode, which is also called register direct. The register indirect addressing mode is a very important addressing mode in the PIC 18. This topic will be discussed thoroughly in this section.

Register indirect addressing mode

In the register indirect addressing mode, a register is used as a pointer to the data RAM location. In the PIC18, three registers are used for this purpose: FSR0, FSR1, and FSR2. FSR stands for *file select register* and must not be confused with SFR (special function register). The FSR is a 12-bit register allowing access to the entire 4096 bytes of data RAM space in the PIC18. We use LFSR (load FSR) to load the RAM address. In other words, when FSRx are used as pointers, they must be loaded first with the RAM addresses as shown below.

```
LFSR 0, 0x30      ;load FSR0 with 0x30
LFSR 1, 0x40      ;load FSR1 with 0x40
LFSR 2, 0x6F      ;load FSR2 with 0x6F
```

Because FSR0, FSR1, and FSR2 are 12-bit registers they cannot fit into the SFR address space unless they are split into pieces of an 8-bit size. That is exactly what PIC18 has done. The FSR registers have the low-byte and high-byte parts called FSRxL and FSRxH, as shown in the SFR table of Table 6-1. In Table 6-1 we see FSR0L and FSR0H, representing the low and high parts of the 12-bit FSR0 register. Note that the FSRxH is only 4-bit and the upper 4 bits are not used. Another register associated with the register indirect addressing mode is the INDF (indirect register). Each of the FSR0, FSR1, and FSR2 registers has an INDF register associated with it, and these are called INDF0, INDF1, and INDF2. When we move data into INDFx we are moving data into a RAM location pointed to by the FSR. In the same way, when we read data from the INDF register, we are reading data from a RAM location pointed to by the FSR. This is shown below.

```
LFSR  0, 0x30     ;FSR0 = 30H RAM location pointer
MOVWF INDF0       ;copy contents of WREG into RAM
                  ;location whose address is held by
                  ;12-bit FSR0 register
```

Advantages of register indirect addressing mode

One of the advantages of register indirect addressing mode is that it makes accessing data dynamic rather than static, as with direct addressing mode. Example 6-2 shows three cases of copying 55H into RAM locations 40H to 45H. Notice in solution (b) that two instructions are repeated numerous times. We can create a loop with those two instructions as shown in solution (c). Solution (c) is the most efficient and is possible only because of the register indirect addressing mode. In Example 6-2, we must use "INCF FSR0L, F" to increment the pointer because there is no such instruction as "INCF FSR0, F". Looping is not possible in direct addressing mode, and that is the main difference between the direct andregister indirect addressing modes. For example, trying to send a string of data located in consecutive locations of data RAM is much more efficient and dynamic using register indirect addressing mode thanusing direct addressing mode. See Example 6-3.