CENG 336

INT. TO EMBEDDED SYSTEMS DEVELOPMENT

Spring 2006

Recitation 03

OUTLINE

- PIC Instruction Set
- Code Structure
- MPLAB Demonstration

Register Addition

ADDWF f,d

Add reg[f] to W and store in either W or reg[f] depending on d,

if d=0 then store in W, else in reg[f]

If reg[24h] =6 and W=4 then

```
ADDWF 24h,1        ; hexadecimal 24
```

Sets reg[24h] to 10

Addition of a constant

ADDLW k

Add k to W and store in W

If W=4 then

```
ADDLW   H'24'  ;  hexadecimal 24
```

Sets W to 28h

ANDWF

ANDWF f,d

And reg[f] with W and store in either W or reg[f] depending on d,

if d=0 then store in W, else in reg[f]

If W = 0001 1111 and reg[20h]= 1111 0100

```
ANDWF 20h,0
```

will set W to 0001 0100

ANDLW

ANDLW k

And k with W and store in W

If W = 0001 1111 and k= 6 (0000 0110)

```
ANDLW 0x6
```

will set W to 0000 0110

Clear registers

CLRF f ; set reg[f] to zero

```
CLRF 0x40  ; reg[40h]:=0
```

CLRW ; set w register to zero

Move operations

`MOVFW  f`

– Moves contents of register f to the W register

`MOVWF  f`

– Moves the W reg to register f

`MOVLW  k`

– Moves the literal constant to the W register

Last two letters are memonics FW,WF,LW

NOP

- NOP stands for NO oPeration
- It is an opcode that does nothing

COMF

COMF f,d ; sets either reg[f] or W to 1's complement of f register content

Example:

```
COMF REG1, 0

Before Instruction
REG1= 0x13

After Instruction
REG1= 0x13
W = 0xEC
```

SUBWF

SUBWF f,d

Subtract W from f

This has two forms

- » SUBWF f,0 ; W := reg[f] – W
- » SUBWF f,1 ; reg[f] := reg[f] - W

```
MOVF    0x33,0    ; W := y
SUBWF   0x32,1    ; x := x - W
```

SUBLW

SUBLW k

Subtract W from k and store in W

If W = 0001 1111 and k= 0010 1111

```
SUBLW          b' 0010 1111 '  ; binary representation
```

will set W to 0001 0000

Decrement register

DECF f,d

Decrements reg[f] once and stores result in either reg[f] or W depending on d

```
DECF  0x50,1
```

Subtracts 1 from register 50

```
DECF 0x50,0
```

Sets W := reg[50h] -1

Decrement and skip

DECFSZ f,d

If the result of decrementing is zero, skip the next instruction

```
Top:
        ;some instructions
        DECFSZ 0x38,1
        GOTO Top
        ;some other instructions
```

Reg[38h] holds the number of times to go round loop

Incrementing

- INCF and INCFSZ work like DECF and DECFSZ except that they increment
- In this case you would load a negative number into your count register and count up towards zero.
- Alternatively, count up, and skip when the result would have been 256.

```
INCFSZ 0x50,1
```

reg[50h] := reg[50h]+1

if reg[50h] is 0 then skip next instruction

Inclusive OR

IORWF f,d

Example

If W=1100 0001 and reg[40h]=0001 0001

`IORWF 0x40,0`

Will set W= 1101 0001

11000001
00010001 or
11010001

Inclusive OR literal

IORLW k

Example

If W=1100 0001

`IORLW 0x7`

Will set W= 1100 0111

11000001
<u>00000111</u> or
11000111

Exclusive OR

XORWF f,d

Example

If W=1100 0001 and reg[40h]=0001 0001

`XORWF 0x40,0`

Will set W= 1101 0000

11000001
00010001 xor
11010000

Exclusive OR literal

XORLW k

Example

If W=1100 0001

XORLW 0x7

Will set W= 1100 0110

11000001
00000111 xor
11000110

Bit operations

```
BCF     f,b  ; set bit b of register f to 0
BSF     f,b  ; set bit b of register f to 1
```

Example

```
BCF    0x34,1
```

Clears bit 1 of register 34

Bit test operations

BTFSC f,b ; bit test skip if clear

BTFSS f,b ; bit test skip if set

Example

```
INCF    0x33
BTFSC   0x33,3
GOTO    OVERFLOW        ; goto overflow when
                        ; reg 33 >7
```

SWAPF

SWAPF f,d

Swaps nibbles in f, stores result in

either reg[f] or W depending on d

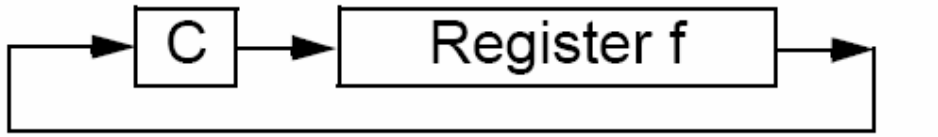
```
SWAPF   W,W   ; will not work!
              ; because W can not be addressed in
              ; 16F877.
```

Rotate right

RRF f,d

- The contents of register ’f’ are rotated one bit to the right through the Carry Flag.

- If ’d’ is 0, the result is placed in the W register. If ’d’ is 1, the result is stored back in register ’f’.

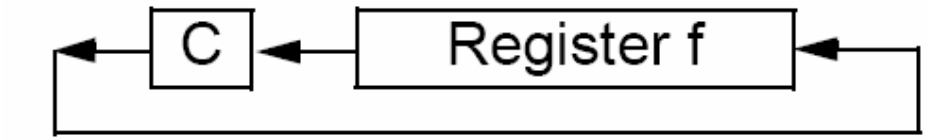


Rotate left

RLF f,d

- The contents of register ’f’ are rotated one bit to the left through the Carry Flag.

- If ’d’ is 0, the result is placed in the W register. If ’d’ is 1, the result is stored back in register ’f’.



GOTO

```
GOTO label
```

Example:

```
 GOTO home
…..
 home
     MOVLW 0x7
….
```

Transfers control to the label

CALL and RETURN

Thare used for subroutines or procedures.

```
CALL foo
….
 foo  ; start of procedure
….    ; body of procedure
RETURN; end of procedure
```

CALL and RETURN

- When a call occurs the PC+1 is pushed onto the *stack* and then the PC is loaded with the address of the label
- When return occurs the stack is popped into the PC transferring control to the instruction after the original call.

RETLW

RETLW k

- The W register is loaded with the eight bit literal 'k'.
- The program counter is loaded from the top of the stack (the return address).

CODE STRUCTURE

```
LIST    P=16F877                ; list directive to define processor
#INCLUDE P16F877.INC            ; processor specific variable definitions

__CONFIG  ( _CP_OFF&_WDT_OFF&_PWRTE_OFF&_XT_OSC&_LVP_OFF)

; variable and constant definitions

ORG         0x000       ; processor reset vector
goto        init        ; go to beginning of initialization

ORG         0x004       ; Interrupt Vector
goto        $

init
                        ; initialization code

mainline
                        ;  main code
goto        mainline
end
```

The ORG directive says where the instruction starts in ROM. Address 0 is where the hardware starts running, Address 4 is where the hardware goes on an interrupt

The registers which will be used in the program should be carefully initialized before main code!

Variable and constant definitions

Definition:

```
count1  equ  H‘35‘
```

Usage:

```
movlw   count1 ; count1 is a constant
movwf   count1 ; count1 is a variable

---------------------------------

#define myPort    PORTB
#define myLed     PORTB,3
```

ASSEMBLER PROCESS