CENG 336

INT. TO EMBEDDED SYSTEMS DEVELOPMENT

Spring 2006

Recitation 02
OUTLINE

- PIC16F877 Overview and Memory Organization
- PIC Instruction Set
- Code Structure
PIC16F877 DEVICE OVERVIEW

Single accumulator
Working register
PIC16F877 Program Memory Organization

- 13 Bit Program Counter
  - capable of addressing an 8K x 14 program memory
- 8 level stack
- Reset Vector: The program comes to this address in the first start or after the microcontroller is reset.
- Interrupt Vector: When an interrupt occurs, the program automatically comes to this address.
## PIC16F877 DATA MEMORY

The data memory is partitioned into multiple banks

- General Purpose Registers
- Special Function Registers
  - Core (CPU) Registers
    - STATUS
    - OPTION_REG
    - INTCON
    - PIE1
    - PIE2
    - PIR1
    - PIR2
  - Peripheral Registers
    - ex. RCSTA, PR2...

### Core (CPU) Registers

- STATUS
- OPTION_REG
- INTCON
- PIE1
- PIE2
- PIR1
- PIR2
- PCON

### Peripheral Registers

- ex. RCSTA, PR2...
### PIC16F877 DATA MEMORY – Cont.

**REGISTER 2-1:**

**STATUS REGISTER** *(ADDRESS 03h, 83h, 103h, 183h)*

<table>
<thead>
<tr>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R/W-0</th>
<th>R-1</th>
<th>R-1</th>
<th>R/W-x</th>
<th>R/W-x</th>
<th>R/W-x</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRP</td>
<td>RP1</td>
<td>RP0</td>
<td>TO</td>
<td>PD</td>
<td>Z</td>
<td>DC</td>
<td>C</td>
</tr>
</tbody>
</table>

**bit 7**: IRP: Register Bank Select bit (used for indirect addressing)
- 1 = Bank 2, 3 (100h - 1FFh)
- 0 = Bank 0, 1 (00h - FFh)

**bit 6-5**: RP1:RP0: Register Bank Select bits (used for direct addressing)
- 11 = Bank 3 (100h - 1FFh)
- 10 = Bank 2 (100h - 17Fh)
- 01 = Bank 1 (80h - FFh)
- 00 = Bank 0 (00h - 7Fh)

Each bank is 128 bytes

**bit 4**: TO: Time-out bit
- 1 = After power-up, CLEARDT instruction, or SLEEP instruction
- 0 = A WDT time-out occurred

**bit 3**: PD: Power-down bit
- 1 = After power-up or by the CLEARDT instruction
- 0 = By execution of the SLEEP instruction

**bit 2**: Z: Zero bit
- 1 = The result of an arithmetic or logic operation is zero
- 0 = The result of an arithmetic or logic operation is not zero

**bit 1**: DC: Digit carry/borrow bit *(ADDWF, ADDLW, SUBLW, SUBFW instructions)*
*(for borrow, the polarity is reversed)*
- 1 = A carry-out from the 4th low order bit of the result occurred
- 0 = No carry-out from the 4th low order bit of the result

**bit 0**: C: Carry/borrow bit *(ADDWF, ADDLW, SUBLW, SUBFW instructions)*
- 1 = A carry-out from the Most Significant bit of the result occurred
- 0 = No carry-out from the Most Significant bit of the result occurred

**Note**: For borrow, the polarity is reversed. A subtraction is executed by adding the two’s complement of the second operand. For rotate *(RRF, RLF)* instructions, this bit is loaded with either the high, or low order bit of the source register.
PIC INSTRUCTION SET
INSTRUCTION FORMATS

Some General Properties:
• Instructions are encoded in binary in ROM.
• The instructions are fixed format, each occupying 14 bits.
• The division of the bits into fields is flexible.

Categories of instructions:
• Byte-oriented file register operations
• Bit-oriented file register operations
• Literal and Control operations
## Instruction Formats

Byte-oriented file register operations

<table>
<thead>
<tr>
<th></th>
<th>13</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>d</td>
<td>f (FILE #)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- d = 0 for destination W
- d = 1 for destination f
- f = 7-bit file register address
INSTRUCTION FORMATS

Example

ADD W register content to 47h address content and save the result to W register

\[ W := W + \text{Reg[47h]} \]

Add W and register
Destination is W
Register number: 47h
INSTRUCTION FORMATS

Example

ADD W register content to 47h address content and save the result to 47h address

Reg[47h] := W + Reg[47h]
INSTRUCTION FORMATS

Bit-oriented file register operations

<table>
<thead>
<tr>
<th>13</th>
<th>10</th>
<th>9</th>
<th>7</th>
<th>6</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>b (BIT #)</td>
<td>f (FILE #)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b = 3-bit bit address
f = 7-bit file register address
INSTRUCTION FORMATS

Example

Clear 5th bit of 29h register content

```
0100  101  010 1001
```

- Clear specified bit of the register
- Bit 5 is to be cleared
- Register number: 29h
INSTRUCTION FORMATS

Literal operations

<table>
<thead>
<tr>
<th>13</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opcode</td>
<td>k (literal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ k = \text{8-bit immediate value} \]
INSTRUCTION FORMATS

Example
ADD 0x17 value to W register content and save the result to W register

```
11 1110
```
```
0001 0111
```

“Add literal” instruction
Value to add: 17h
### INSTRUCTION FORMATS

#### Control operations

<table>
<thead>
<tr>
<th>13</th>
<th>11</th>
<th>10</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPCODE</td>
<td>k (literal)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ k = \text{11-bit immediate value} \]
INSTRUCTION FORMATS

Example

GOTO 0x32

| 101 | 000 0011 0010 |

Binary for GOTO  Binary for 32h
Program Memory Paging

- The upper 2 bits of the address are provided by PCLATH<4:3>.

- When doing a CALL or GOTO instruction, the user must ensure that the page select bits are programmed so that the desired program memory page is addressed.
### TABLE 13-2: PIC16F87X INSTRUCTION SET

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Cycles</th>
<th>MSb</th>
<th>LSb</th>
<th>Status Affected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADDWF</strong> f, d</td>
<td>Add W and f</td>
<td>1</td>
<td>00 0111 dffe fffe</td>
<td>C,DC,Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>ANDWF</strong> f, d</td>
<td>AND W with f</td>
<td>1</td>
<td>00 0101 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>CLRF</strong> f</td>
<td>Clear f</td>
<td>1</td>
<td>00 0001 lffe fffe</td>
<td>Z</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>CLRwf</strong> - W</td>
<td>Clear W</td>
<td>1</td>
<td>00 0001 0xxx xxxx</td>
<td>Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>COMF</strong> f, d</td>
<td>Complement f</td>
<td>1</td>
<td>00 1001 lffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>DECF</strong> f, d</td>
<td>Decrement f</td>
<td>1</td>
<td>00 0111 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>DECSFSZ</strong> f, d</td>
<td>Decrement f, Skip if 0</td>
<td>1(2)</td>
<td>00 1011 dffe fffe</td>
<td>Z,1,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INCF</strong> f, d</td>
<td>Increment f</td>
<td>1</td>
<td>00 1010 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>INCFSZ</strong> f, d</td>
<td>Increment f, Skip if 0</td>
<td>1(2)</td>
<td>00 1111 dffe fffe</td>
<td>Z,1,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IORWF</strong> f, d</td>
<td>Inclusive OR W with f</td>
<td>1</td>
<td>00 0100 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>MOVf</strong> f, d</td>
<td>Move f</td>
<td>1</td>
<td>00 1000 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>MOVF</strong> f, d</td>
<td>Move W to f</td>
<td>1</td>
<td>00 0000 lffe fffe</td>
<td>Z</td>
<td>1,2</td>
<td></td>
</tr>
<tr>
<td><strong>NOP</strong> -</td>
<td>No Operation</td>
<td>1</td>
<td>00 0000 0xx0 0xxx</td>
<td>0xxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RLF</strong> f, d</td>
<td>Rotate Left f through Carry</td>
<td>1</td>
<td>00 1101 dffe fffe</td>
<td>C,1,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RRF</strong> f, d</td>
<td>Rotate Right f through Carry</td>
<td>1</td>
<td>00 1100 dffe fffe</td>
<td>C,1,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SUBWF</strong> f, d</td>
<td>Subtract W from f</td>
<td>1</td>
<td>00 0010 dffe fffe</td>
<td>C,DC,Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SWAPF</strong> f, d</td>
<td>Swap nibbles in f</td>
<td>1</td>
<td>00 1110 dffe fffe</td>
<td>Z,1,2</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XORWF</strong> f, d</td>
<td>Exclusive OR W with f</td>
<td>1</td>
<td>00 0110 dffe fffe</td>
<td>Z,1,2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### BYTE-ORIENTED FILE REGISTER OPERATIONS

#### BIT-ORIENTED FILE REGISTER OPERATIONS

| **BCF** f, b      | Bit Clear f | 1      | 01 00bb bffe fffe | Z,1,2        |                |       |
| **BSF** f, b      | Bit Set f   | 1      | 01 01bb bffe fffe | Z,1,2        |                |       |
| **BTFSC** f, b    | Bit Test f, Skip if Clear | 1(2)  | 01 10bb bffe fffe | Z,3          |                |       |
| **BTFSS** f, b    | Bit Test f, Skip if Set | 1(2)  | 01 11bb bffe fffe | Z,3          |                |       |

#### LITERAL AND CONTROL OPERATIONS

| **ADDLW** k      | Add literal and W | 1     | 11 111x kkkk kkkk | C,DC,Z       |                |       |
| **ANDLW** k      | AND literal with W | 1     | 11 1001 kkkk kkkk | Z,1,2        |                |       |
| **CALL** k       | Call subroutine   | 2     | 10 00kk kkkk kkkk |                |                |       |
| **CLRWDT** -     | Clear Watchdog Timer | 1     | 00 0000 0110 0100 | TO,PD        |                |       |
| **GOTO** k       | Go to address    | 2     | 10 1xxx kkkk kkkk |                |                |       |
| **IORLW** k      | Inclusive OR literal with W | 1    | 11 1000 kkkk kkkk | Z,1,2        |                |       |
| **MOVLW** k      | Move literal to W | 1     | 11 00xx kkkk kkkk | Z,1,2        |                |       |
| **RETIE** -      | Return from interrupt | 2     | 00 0000 0000 1001 | TO,PD        |                |       |
| **RETLW** k      | Return with literal in W | 2     | 11 01xx kkkk kkkk | TO,PD        |                |       |
| **RETURN** -     | Return from Subroutine | 2     | 00 0000 0110 0111 | TO,PD        |                |       |
| **SLEEP** -      | Go into standby mode | 1     | 00 0000 0110 0111 | TO,PD        |                |       |
| **SUBLW** k      | Subtract W from literal | 1     | 11 101x kkkk kkkk | C,DC,Z       |                |       |
| **XORLW** k      | Exclusive OR literal with W | 1    | 11 1010 kkkk kkkk | Z            |                |       |
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
CLRF f ; set reg[f] to zero
CTCF 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 $f,0$ ; $W := \text{reg}[f] - W$

» SUBWF $f,1$ ; \text{reg}[f] := \text{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

\[
\text{reg}[50h] := \text{reg}[50h]+1
\]
\[
\text{if reg}[50h] \text{ is 0 then skip next instruction}
\]
Inclusive OR

IORWF  f,d

Example
If $W=1100\ 0001$ and $\text{reg}[40h]=0001\ 0001$

$\text{IORWF}\ 0x40,0$

Will set $W=1101\ 0001$

\[
\begin{array}{c}
11000001 \\
00010001 \\
\hline
11010001
\end{array}
\] or

\[
\begin{array}{c}
11000001 \\
00010001 \\
11010001
\end{array}
\]
Inclusive OR literal

IORLW \text{k}

Example
If W=1100 0001
IORLW 0x7
Will set W= 1100 0111
Exclusive OR

XORWF     f,d

Example
If W=1100 0001 and reg[40h]=0001 0001
XORWF 0x40,0
Will set W= 1101 0000

\[
\begin{array}{c}
11000001 \\
00010001 \\
\hline
\text{xor} \\
11010000
\end{array}
\]
Exclusive OR literal

XORLW k

Example
If W=1100 0001
XORLW 0x7
Will set W=1100 0110

\[
\begin{array}{c|c}
11000001 & 00000111 \\
00000111 & \text{xor} \\
11000110 & \end{array}
\]
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, 4
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'.

![Diagram of RRF operation]
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

These are 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 ;

goto end

01.03.2007 CEng336

__CONFIG' directive is used to embed configuration word within .asm file. The labels following the directive are located in the respective .inc file. See data sheet for additional information on configuration word settings.

The ORG directive says where the instruction start in ROM. Address 0 is where the hardware starts running Address 4 is where the hardware goes on an interrupt
Variable and constant definitions

Definition:
\[ \text{count1} \text{ equ } \text{H}^{'35'} \]

Usage:
\[ \text{movlw} \text{ count1} ; \text{count1 is a constant} \]
\[ \text{movwf} \text{ count1} ; \text{count1 is a variable} \]

#define myPort PORTB
#define myLed PORTB,3
ASSEMBLER PROCESS

code.asm  \rightarrow  MPASM Assembler  \rightarrow  code.hex  \rightarrow  Programmer  \rightarrow  PIC