CENG 336

INT. TO EMBEDDED SYSTEMS DEVELOPMENT

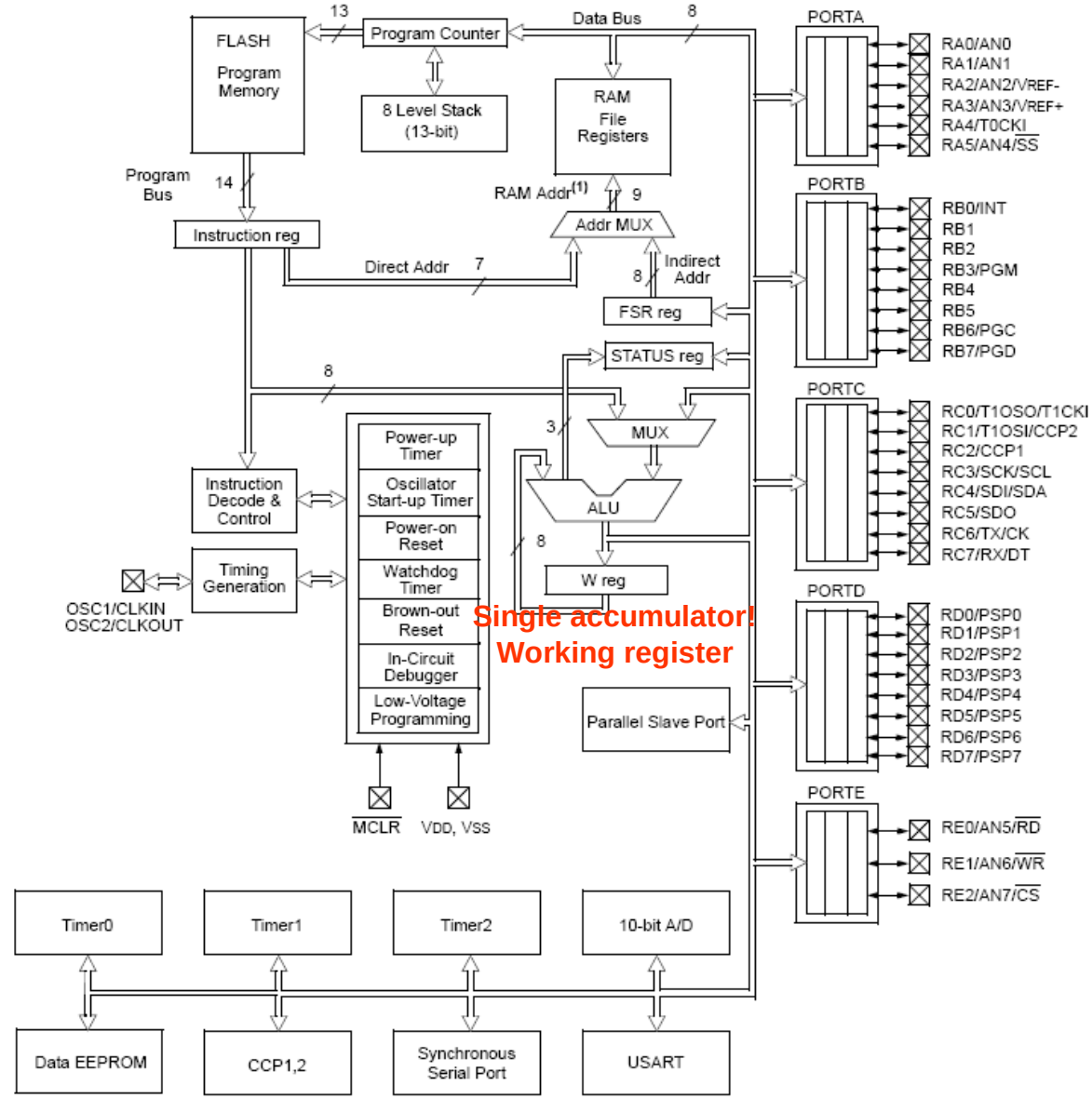
Spring 2006

Recitation 02

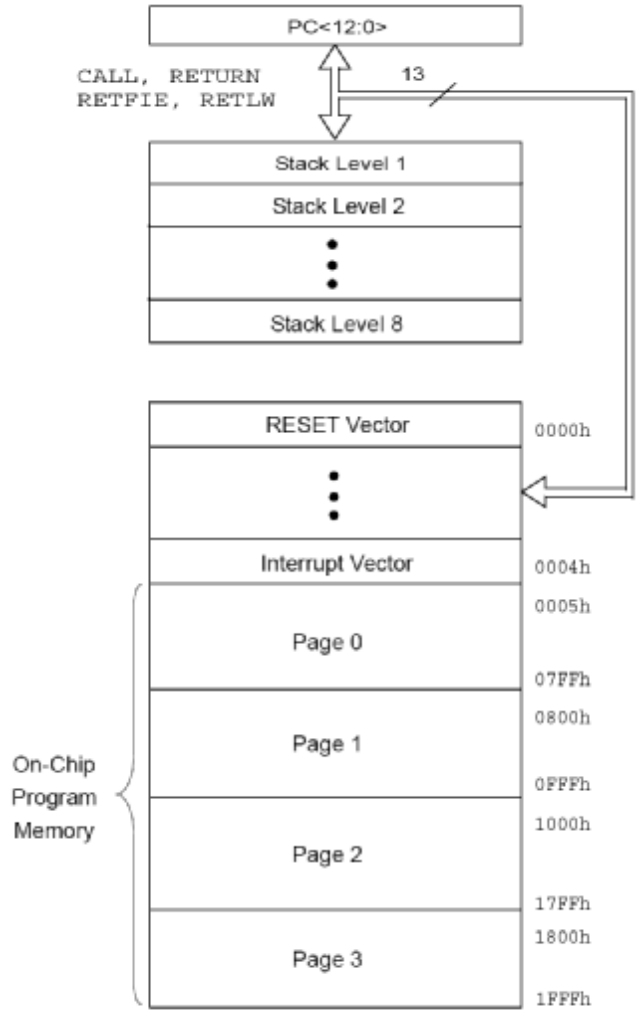
OUTLINE

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

PIC16F877 DEVICE OVERVIEW

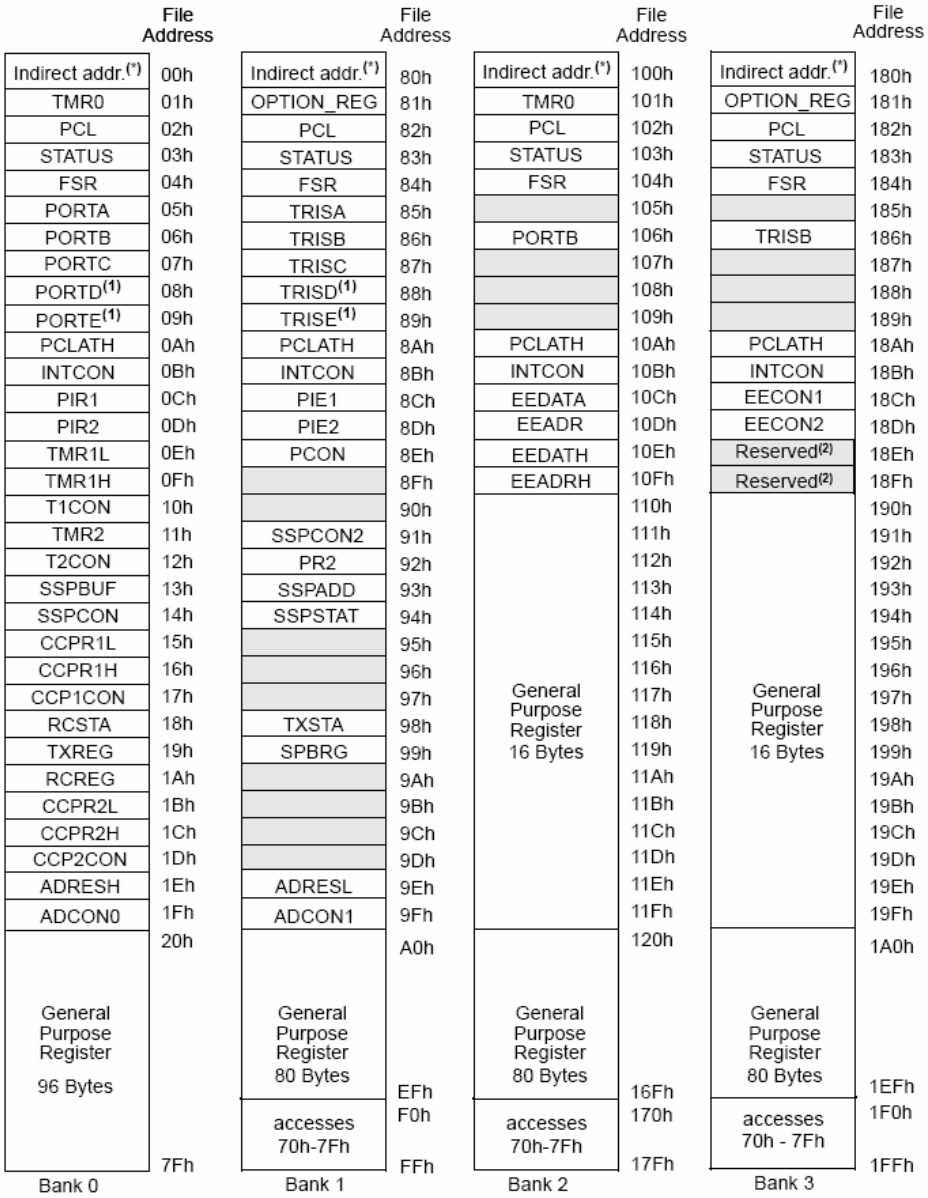


PIC16F877 Program Memory Organization



- 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



Unimplemented data memory locations, read as '0'.
* Not a physical register.

The data memory is partitioned into multiple banks

→ General Purpose Registers

→ Special Function Registers

- § Core (CPU) Registers
  - STATUS
  - OPTION_REG
  - INTCON
  - PIE1
  - PIR1
  - PIE2
  - PIR2
  - PCON
- § Peripheral Registers
  - § ex. RCSTA, PR2...

PIC16F877 DATA MEMORY – Cont.

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h, 103h, 183h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

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 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 **$\overline{TO}$**: Time-out bit
1 = After power-up, CLRWDT instruction, or SLEEP instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$**: Power-down bit
1 = After power-up or by the CLRWDT 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/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, 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/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF 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 $\overline{borrow}$, 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:

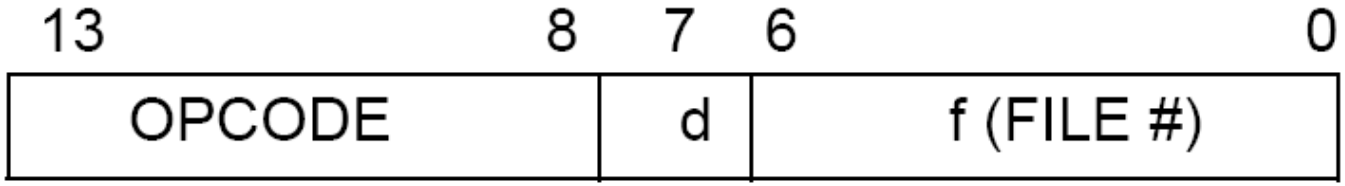
- Instructions are encoded in binary in ROM.
- The instructions are fixed format, each occupying 14 bits.
- The division of the bits into fields is flexible.

Categories of instructions:

- Byte-oriented file register operations
- Bit-oriented file register operations
- Literal and Control operations

INSTRUCTION FORMATS

Byte-oriented file register operations

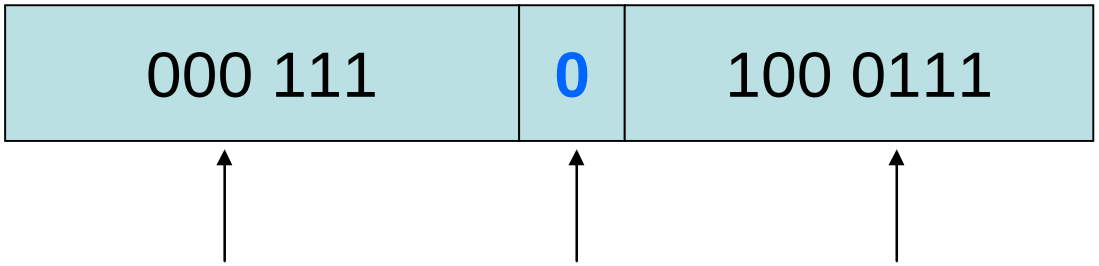


d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

INSTRUCTION FORMATS

Example

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

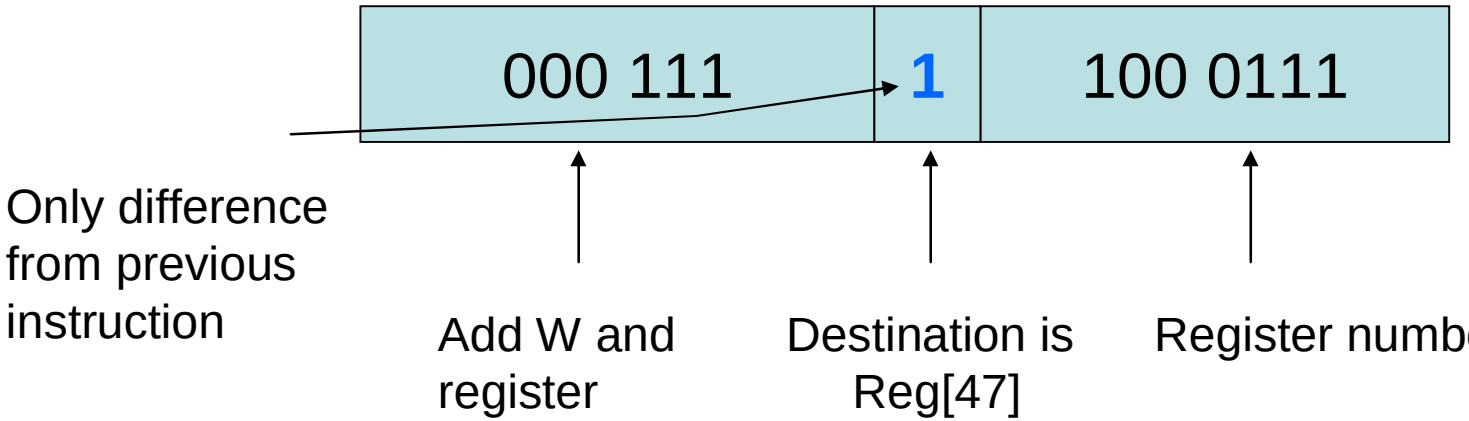


W := W + Reg[47h]

INSTRUCTION FORMATS

Example

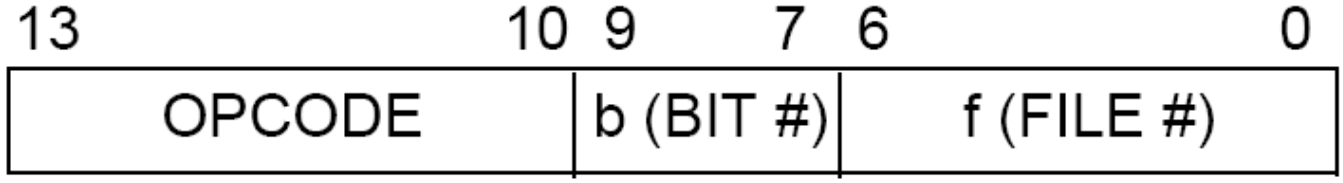
ADD W register content to 47h address content and save the result to 47h address



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

INSTRUCTION FORMATS

Bit-oriented file register operations

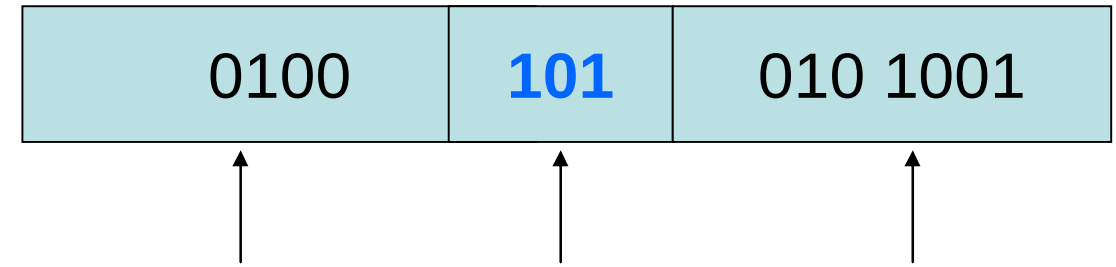


b = 3-bit bit address
f = 7-bit file register address

INSTRUCTION FORMATS

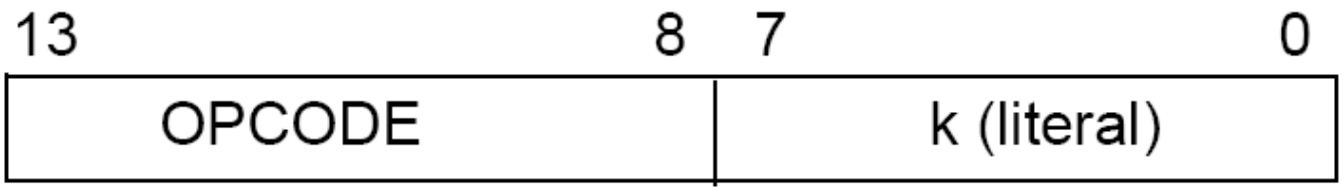
Example

Clear 5th bit of 29h register content



INSTRUCTION FORMATS

Literal operations

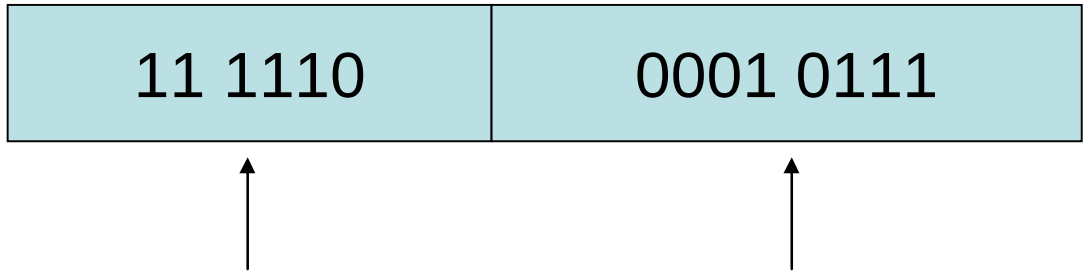


k = 8-bit immediate value

INSTRUCTION FORMATS

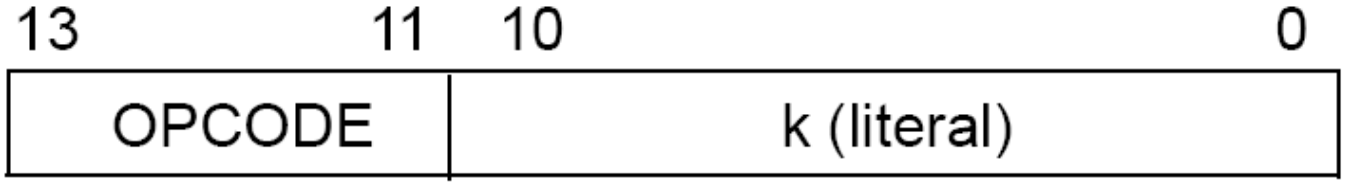
Example

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



INSTRUCTION FORMATS

Control operations

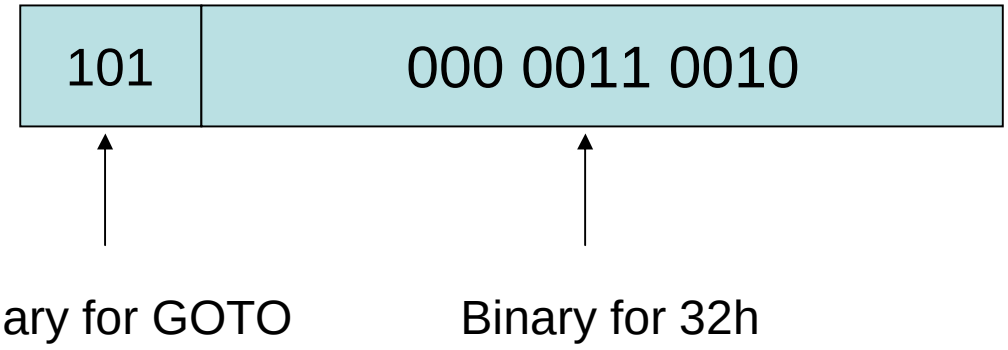


k = 11-bit immediate value

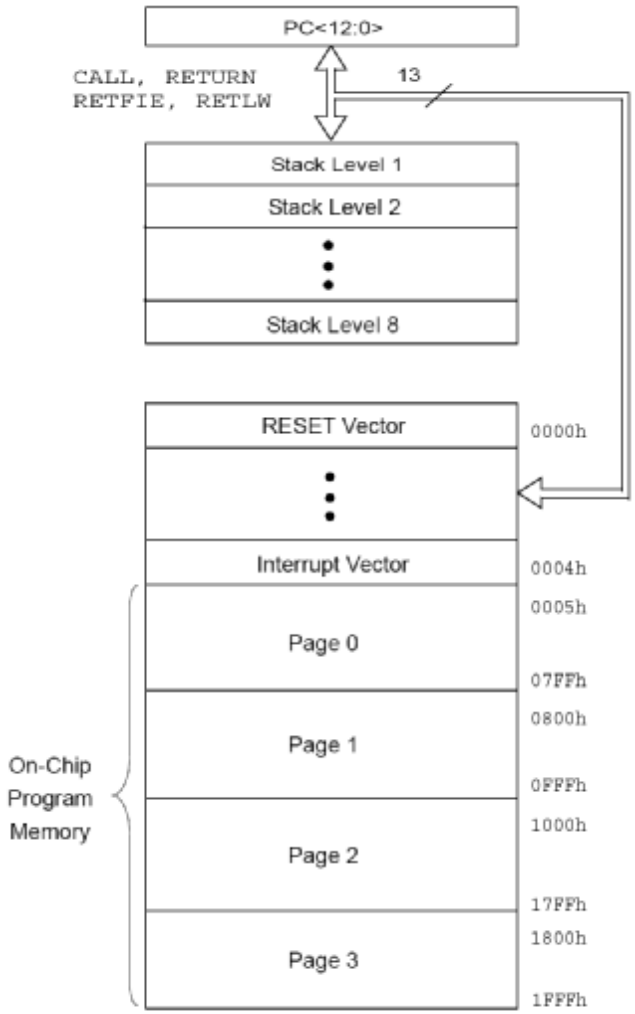
INSTRUCTION FORMATS

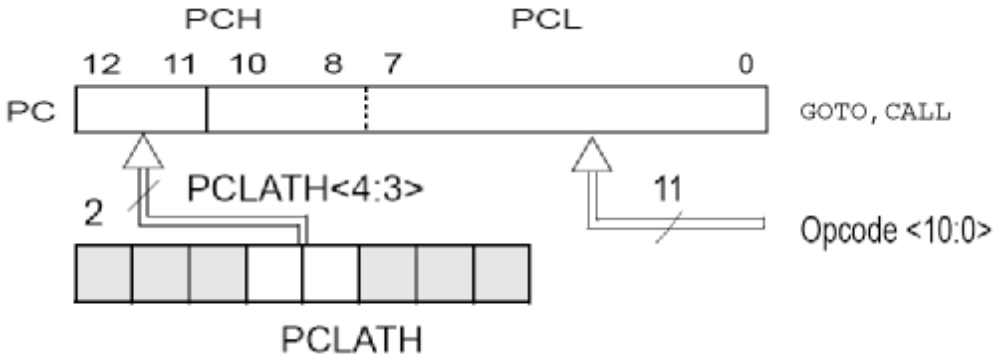
Example

GOTO 0x32



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

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS						
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2
CLRF	f	Clear f	1	00 0001 lfff ffff	Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx	Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff	Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff		1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff	Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff	Z	1,2
MOVWF	f	Move W to f	1	00 0000 lfff ffff		
NOP	-	No Operation	1	00 0000 0xx0 0000		
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS						
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff		1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff		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	
CALL	k	Call subroutine	2	10 0kkk kkkk kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk		
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk	Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk		
RETFIE	-	Return from interrupt	2	00 0000 0000 1001		
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk		
RETURN	-	Return from Subroutine	2	00 0000 0000 1000		
SLEEP	-	Go into standby mode	1	00 0000 0110 0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x 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

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   0x33,4
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

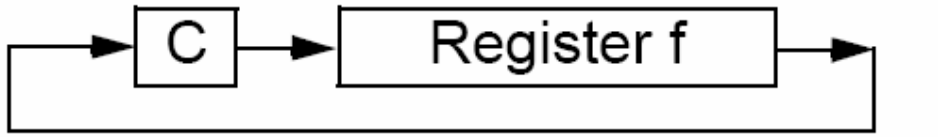
```
SWAPF   W,W   ; will not work!
              ; because W can not be addressed in
              ; 16F877.
```

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’.

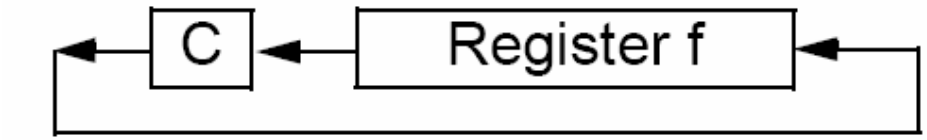


Rotate left

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
```

'__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:

```
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