PIC Assembler

Background

Back in the 1960's, compters were programmed in machine code. The operator would set switches according to the binary code corrsponding to each line of code, push a button, and set the switches for the next line of code.

Machine code is very cryptic. A program for a PIC which counts on PORTC looks like the following:

```
060000000A128A11F92F1B
0E0FF2008316031387018312870187 0AFE2FDF
00000001FF
```

Assembler is *much* superior to machine code. Semi-meaningful names represent the valid machine operations, as described in the previous notes. The previous code would look like the following

```
            _main
            bsf       STATUS, RP0
            bcf       STATUS, RP1
            clrf      TRISC
            bcf       STATUS, RP0
            clrf      PORTC
_loop       incf      PORTC,F
            goto      _loop
```

This is a lot easier to understand than the machine code. It is still very cryptic, however. In addition, assembler has a limited set of commands.

Instruction Sets

Only 75 instructions are used in the PIC18F4620 family. This allows the hardware to be optimized for these 75 instructions, saving size, power, and increasing execution speed (at present, a PIC processor can execute up to 5 million instructions per second while costing as little as $1.27 each)

Pretty much all a PIC can do is

- Set and clear bits
- Read and write from memory (8-bits at a time)
- Logic and / or / exclisuve or (8-bits at a time)
- Add, subtract
- Multiply by two (shift left), and shift right
- Multiply two 8-bit numbers

Anything else must be built up using these simple instructions.

The formatting of an instruction is

Label operation REGISTER, F (W)

Label: optional name you can jump to with a 'goto' command (1st letter cap)

operation: assembler mnemonic for some operation (like clear) (lower case)

REGISTER: RAM address to be operated on

F: Save the result in the register

W: Save the result in the working register

Memory Read & Write		
MOVWF PORTA	memory write	PORTA = W
MOVFF PORTA PORTB	copy	PORTB = PORTA
MOVF PORTA,W	memory read	W = PORTA
MOVLW 123	Move Literal to WREG	W = 123
Memory Clear, Negation		
CLRF PORTA	clear memory	PORTA = 0x00
COMF PORTA	toggle bits	PORTA = !PORTA
NEGF PORTA	negate	PORTA = -PORTA
Addition & Subtraction		
INCF PORTA,F	increment	PORTA = PORTA + 1
ADDWF PORTA, F	add	PORTA = PORTA + W
ADDWFC PORTA, W	add with carry	W = PORTA + W + carry
ADDLW	Add Literal and WREG	
DECF PORTA,F	decrement	PORTA = PORTA - 1
SUBFWB PORTA,F	subtract with borrow	PORTA = W - PORTA - c
SUBWF PORTA,F	subtract no borrow	PORTA = PORTA - W
SUBWFB PORTA,F	subtract with borrow	PORTA = PORTA - W - c
SUBLW 223	Subtract WREG from #	W = 223 - W
Shift left (*2), shift right (/2)		
RLCF PORTA,F	rotate left through carry (9-bit rotate)	
RLNCF PORTA,F	rotate left no carry	
RRCF PORTA,F	rotate right through carry	
RRNCF PORTA,F	rotate right no carry	
Bit Operations		
BCF PORTA, 3	Bit Clear f	clear bit 3 of PORTA
BSF PORTA, 4	Bit Set f	set bit 4 of PORTA
BTG PORTA, 2	Bit Toggle f	toggle bit 2 of PORTA
Logical Operations		
ANDWF PORTA, F	logical and	PORTA = PORTA and W
ANDLW 0x23	AND Literal with WREG	W = W and 0x23
IORWF PORTA,F	logical or	PORTA = PORTA or W
IORLW 0x23	Inclusive OR Literal	W = W or 0x23

XORWF PORTA,F	logical exclusive or	PORTA = PORTA xor W
XORLW 0x23	Exclusive OR Literal	W = W xor 0x23
Tests (skip the next instruction if...)		
CPFSEQ PORTA	Compare PORTA to W, skip if PORTA = W	
CPFSGT PORTA	Compare PORTA to W, Skip if PORTA > W	
CPFSLT PORTA	Compare PORTA to W, Skip if PORTA < W	
DECFSZ PORTA,F	decrement, skip if zero	
DCFSNZ PORTA,F	decrement, skip if not zero	
INCFSZ PORTA,F	increment, skip if zero	
INFSNZ PORTA,F	increment, skip if not zero	
BTFSC PORTA, 5	Bit Test f, Skip if Clear	
BTFSS PORTA, 1	Bit Test f, Skip if Set	
Flow Control		
GOTO Label	Go to Address 1st word	
CALL Label	Call Subroutine 1st word	
RETURN	Return from Subroutine	
RETLW 0x23	Return with 0x23 in WREG	
RETFIE	Return from Interrupt	
Other Stuff....		
NOP	No Operation	
MULLW	Multiply Literal with WREG	
MULWF PORTA	multiply	
TSTFSZ PORTA	test, skip if zero	

Sample Code:

Note: All actions usually pass through the W register.

Examples:

A = 5;

```
movlw   5           ; move 5 to W
movwf   A           ; move W to A
```

A += 5

```
movlw   5           ; move 5 to W
addwf   A,W         ; add to A, store the result in W
movwf   A           ; move W to A

movlw   5           ; move 5 to W
addwf   A,F         ; add to A, store the result in A
```

A = B

```
movff   B,A
```

if (A == B) X = 10;

```
                movf    A,W             ; move A to W
                cpfseq  B               ; compare A to B, skip if equal
                goto    End             ; no skip, done
                movlw   10              ; move 10 to W
                movwf   X               ; move W to X
 End:           nop
```

if (A > B) X = 10; else X = 12;

```
                movf    B,W             ; move B to W
                cpfsgt  A               ; if A > B, skip
                goto    Else            ; false, goto else
 If:
                movlw   10              ; true, move 10 to X
                movwf   X
                goto    End
 Else:
                movlw   12              ; move 12 to X
                movwf   X
 End:
                nop
```

for (i=1, i<10, i++);

```
                movlw   1               ; i = 1
                movwf   i
 Loop:
                incf    i,F             ; i++
                movlw   10
                cpfslt  i               ; skip next command if (i < 10)
                goto    End             ; false - exit
                goto    Loop            ; true, keep looping
 End:
                nop
```

do { x = x + 1; } while (x <= 10);

```
 Loop:
                incf    X,F             ; x = x + 1;
                movlw   10
                cpfsgt  X               ; skip next command if (x > 10)
                goto    Loop
 End:
                nop
```

Note: There are several way to do the same thing. Some are more efficient than others. As a result

- Different C compilers will give different versions of the compiled code
- Decompilers exist (Convert assembler to C) - but you have to know what C compiler you used.
- An expert assembler programmer will always give more efficient code than a C compiler. (Typical 3x to 10x smaller code). Some C compilers claim 80% efficiency - but that's fr specific test cases.
- Assembler is difficult to write and almost impossible to read.

Note: A very useful register is the STATUS register:

STATUS								
Pin	7	6	5	4	3	2	1	0
Name	-	-	-	N	OV	Z	DC	C

N: Negative bit: This bit is used for signed arithmetic (2's complement). It indicates whether the result was negative (ALU MSB = 1).

- 1 = Result was negative
- 0 = Result was positive

bit 3 OV: Overflow bit: This bit is used for signed arithmetic (2's complement). It indicates an overflow of the 7-bit magnitude which causes the sign bit (bit 7) to change state.

- 1 = Overflow occurred for signed arithmetic (in this arithmetic operation)
- 0 = No overflow occurred

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. For ADDWF, ADDLW, SUBLW and SUBWF instructions:

- 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. For ADDWF, ADDLW, SUBLW and 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 occurreRP1: RP0:

Sample Programs

Display {1, 2, 3, 4} on {PORTA, PORTB, PORTC, PORTD}

```
#include <p18f4620.inc>

      org 0x800
      clrf TRISA
      clrf TRISB
      clrf TRISC
      clrf TRISD
      movlw 0x0F
      movwf ADCON1

      movlw 1
      movwf PORTA
      movlw 2
      movwf PORTB
      movlw 3
      movwf PORTC
      movlw 4
      movwf PORTD

Loop:
      goto Loop
      end
```

When you compile, this creates several files. The .lst file shows

- The address of each instruction (LOC)
- The machine code for that instuction (OBJECT)
- The corresponding assembly command

```
LOC  OBJECT CODE     LINE SOURCE TEXT
000800                00003       org 0x800
000800 6A92           00004       clrf TRISA
000802 6A93           00005       clrf TRISB
000804 6A94           00006       clrf TRISC
000806 6A95           00007       clrf TRISD
000808 0E0F           00008       movlw 0x0F
00080A 6EC1           00009       movwf ADCON1
                      00010
00080C 0E01           00011       movlw 1
00080E 6E80           00012       movwf PORTA
000810 0E02           00013       movlw 2
000812 6E81           00014       movwf PORTB
000814 0E03           00015       movlw 3
000816 6E82           00016       movwf PORTC
000818 0E04           00017       movlw 4
00081A 6E83           00018       movwf PORTD
                      00019
00081C                00020 Loop:
00081C EF0E F004      00021       goto Loop
                      00022       end
```

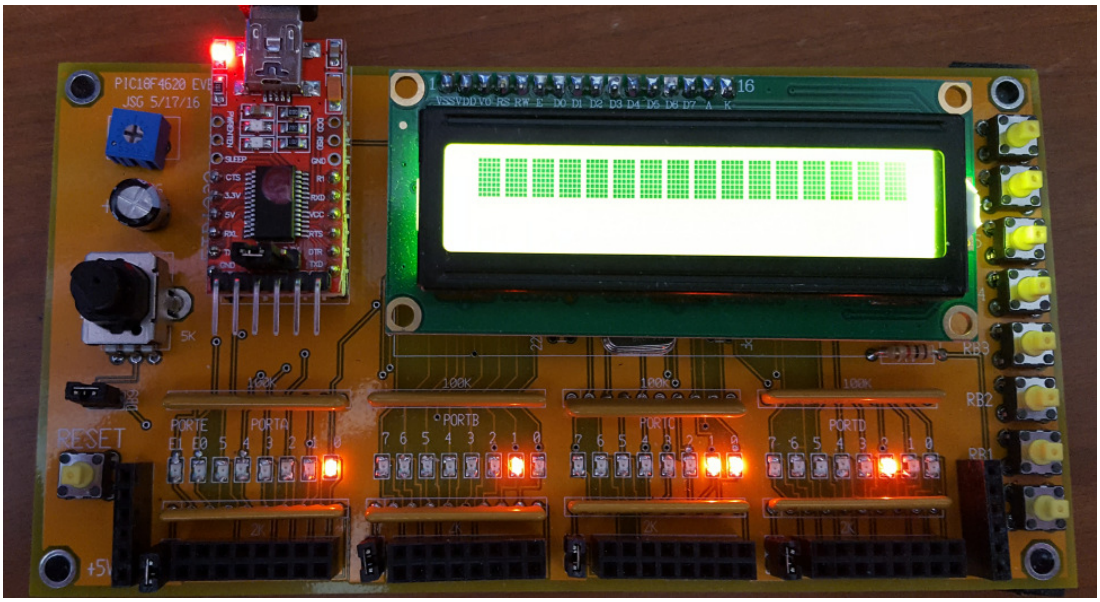
1234.lst file

The .hex file contains the machine code: the thing you download to the PIC processor

```
:020000040000FA
:10080000926A936A946A956A0F0EC16E010E806EA9
:10081000020E816E030E826E040E836E0EEF04F0E4
:00000001FF
```

1234.hex: Machine code that the PIC processor wants

When you download the .hex file to the PIC processor, it executes the program (lecture #3 goes through how to download code)



PIC Board running progrm that sends {1,2,3,4} to {PORTA, PORTB, PORTC, PORTD}

Note that the program worked!

- PORTA = 1
- PORTB = 2
- PORTC = 3
- PORTD = 4

Also note that only engineers get excited when a light turns on. This may not seem like much, but it's a big deal. What this means is

- Your program compiled
- You were able to download your program to the PIC board
- The PIC board is running your program

It took several hours of soldering, debugging, installing software, compiling, etc. just to get to this point. A light turning on reall is a big deal.

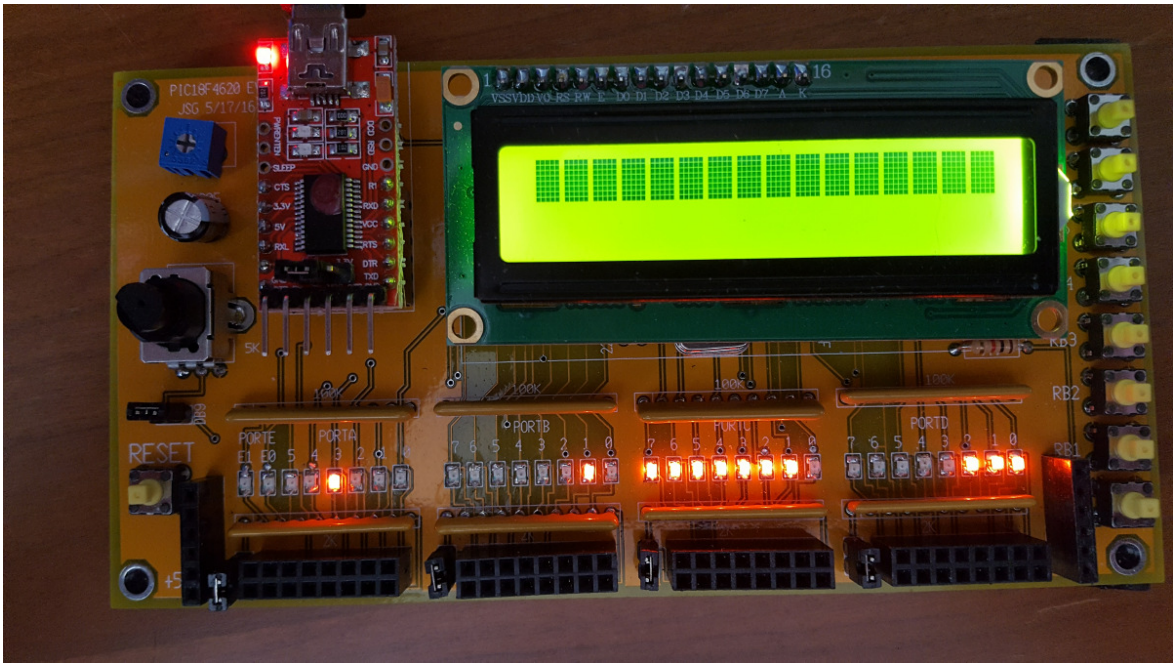
Example 2: Do some operations in assembler

- A = 3
- B = 5
- PORTA = A + B
- PORTB = A - B
- PORTC = B - A
- PORTD = A or B

Code:

```
#include <p18f4620.inc>

A equ 0
B equ 1

      org 0x800
      clrf TRISA
      clrf TRISB
      clrf TRISC
      clrf TRISD
      movlw 0x0F
      movwf ADCON1

      movlw 3
      movwf A
      movlw 5
      movwf B

      movf   A,W
      addwf  B,W
      movwf  PORTA

      movf   A,W
      subwf  B,W
      movwf  PORTB

      movf   B,W
      subwf  A,W
      movwf  PORTC

      movf   A,W
      iorwf  B,W
      movwf  PORTD

Loop:
      goto Loop
      end
```

The result when you download your code is:



PIC Board running program for doing math in assembler

Note that

- PORTA = 3 + 5
- PORTB = 5 - 3
- PORTC = 3 - 5 (twos compliment for -2)
- PORTD = 3 or 5