PIC Assembler

ECE 376 Embedded Systems

Jake Glower - Lecture #2

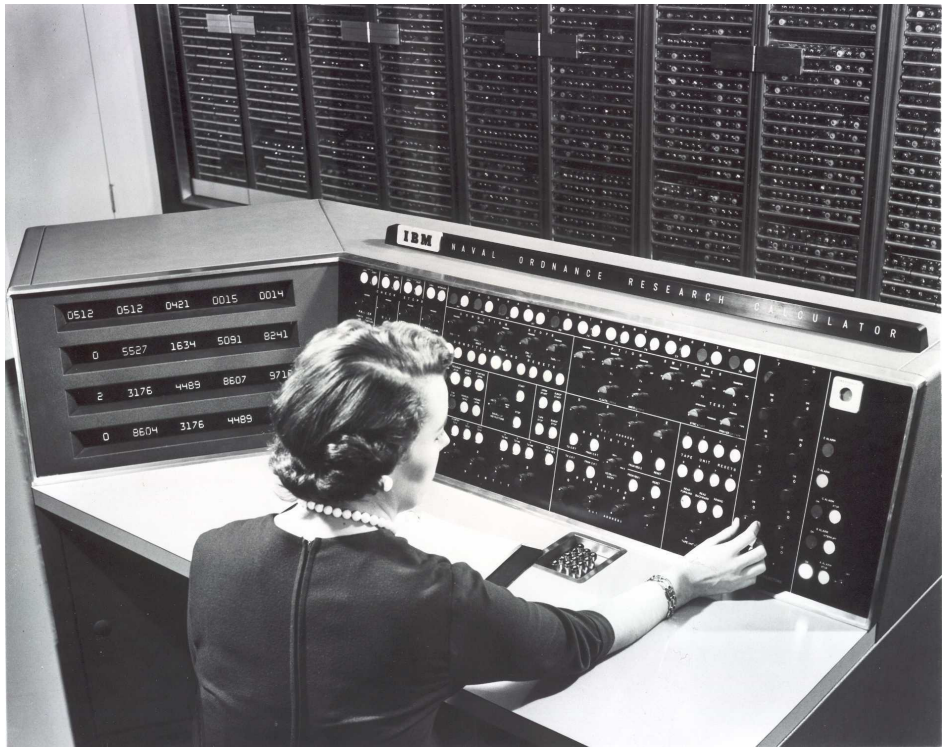
Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Programming

The first computer programs were written in binary (machine code)

- Define each bit with a switch
- Hit program to record the first line of code
- Resulting code looked like this

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



http://www.columbia.edu/cu/computinghistory/norc-4.jpg

Assembler (1951+)

- Dates back to 1951: "The preparation of programs for an electronic digital computer," Wilkes, Wheeler and Gill (Wikipedia)
- Compilers turn assembler in to machine code
- *Much* easier to write than machine code
- Still really cryptic

```
    _main
     movlw      0x0F
     movwf      ADCON1
     clrf       TRISC
     clrf       PORTC
_loop           incf        PORTC,F
     goto       _loop
```

MOVWF	**Move W to f**
Syntax:	MOVWF f {,a}
Operands:	0 ≤ f ≤ 255 a ∈ [0,1]
Operation:	(W) → f
Status Affected:	None
Encoding:	0110 | 111a | ffff | ffff
Description:	Move data from W to register 'f'. Location 'f' can be anywhere in the 256-byte bank. If 'a' is '0', the Access Bank is selected. If 'a' is '1', the BSR is used to select the GPR bank. If 'a' is '0' and the extended instruction set is enabled, this instruction operates in Indexed Literal Offset Addressing mode whenever f ≤ 95 (5Fh). See **Section 24.2.3 "Byte-Oriented and Bit-Oriented Instructions in Indexed Literal Offset Mode"** for details.

Higher-Level Languages

- FORTRAN: 1950's IBM (*Wikipedia)*
- C: 1972 Bell Labs *(Wikipedia)*
- Python: 1991 Guido van Rossum *(wikipedia*)
- Many others

Compiler

- Converts C to assembler
- Assembler to machine code

Each level

- Increases code size 3 - 10x
- Reduces speed 3 - 10x

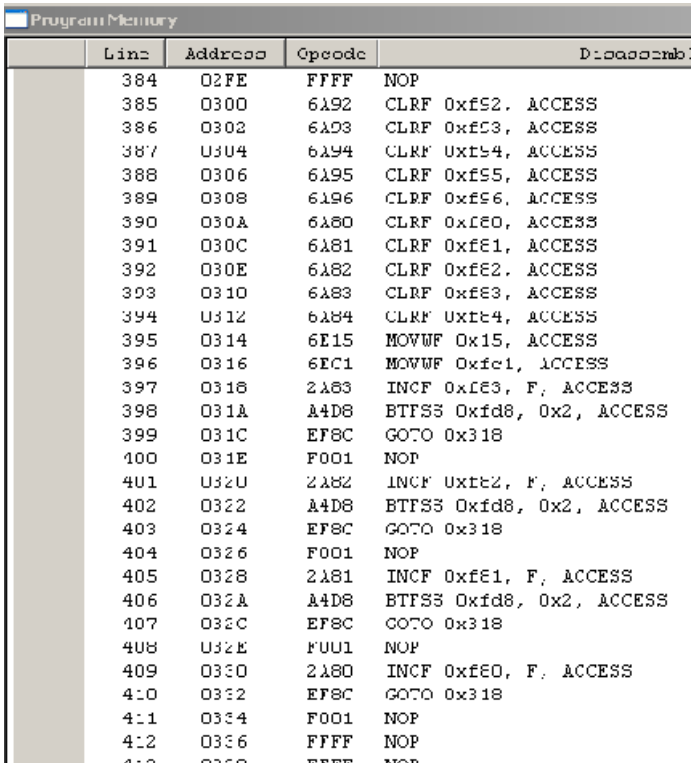
https://www.geeksforgeeks.org/c-programming-language/

Why Assembler?

- Closest thing to machine code: it's how computers actually operate
- C code is convertered into assembler
  - To understand how your C code executes, look at the assembler listing
- Direct access to hardware
  - 100% control of the processor
- Fast and efficient
  - Flight controller for the F16 is 16k
  - Written in assembler

Why Not Assembler?

- Hard to write
  - Very cryptic
- Throw away code
  - Very hard to understand, debug, test, maintain



CISC vs. RISC

CISC: Complex Instruction Set Computing.

- Intel Pentium chip: 500+ instructions
- Floating point arctangent is one instruction
- Fast: Anything you want to do probably has an instruction for it

RISC: Reduced Instruction Set Computing

- Only a few instructions are actually used
- Optimize the computer for these instructions
- Fast: Computer is optimized for the instructions you actually use

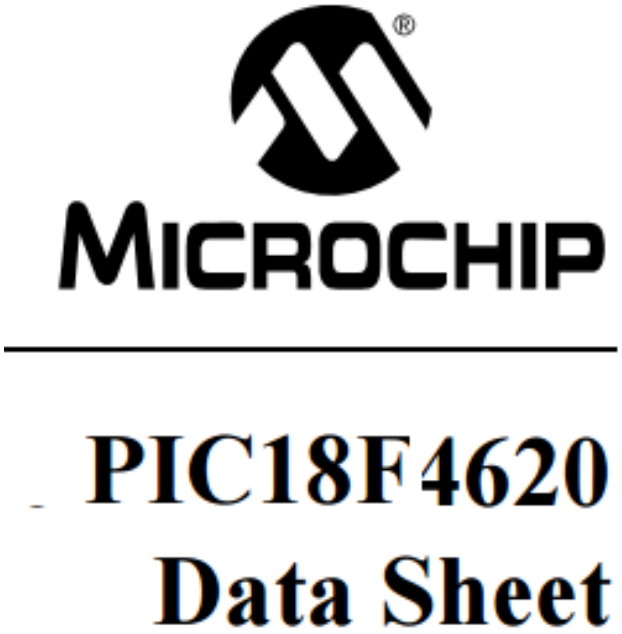
PIC Instructions

- Only 75 instructions with PIC18F4620 (RISC)
- Easier to learn (only 75 instructions)
- Harder to use (requires some convoluted logic)

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.



PIC Assembler

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	234	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

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.

Status Register

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

N: Negative bit:

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

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

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:

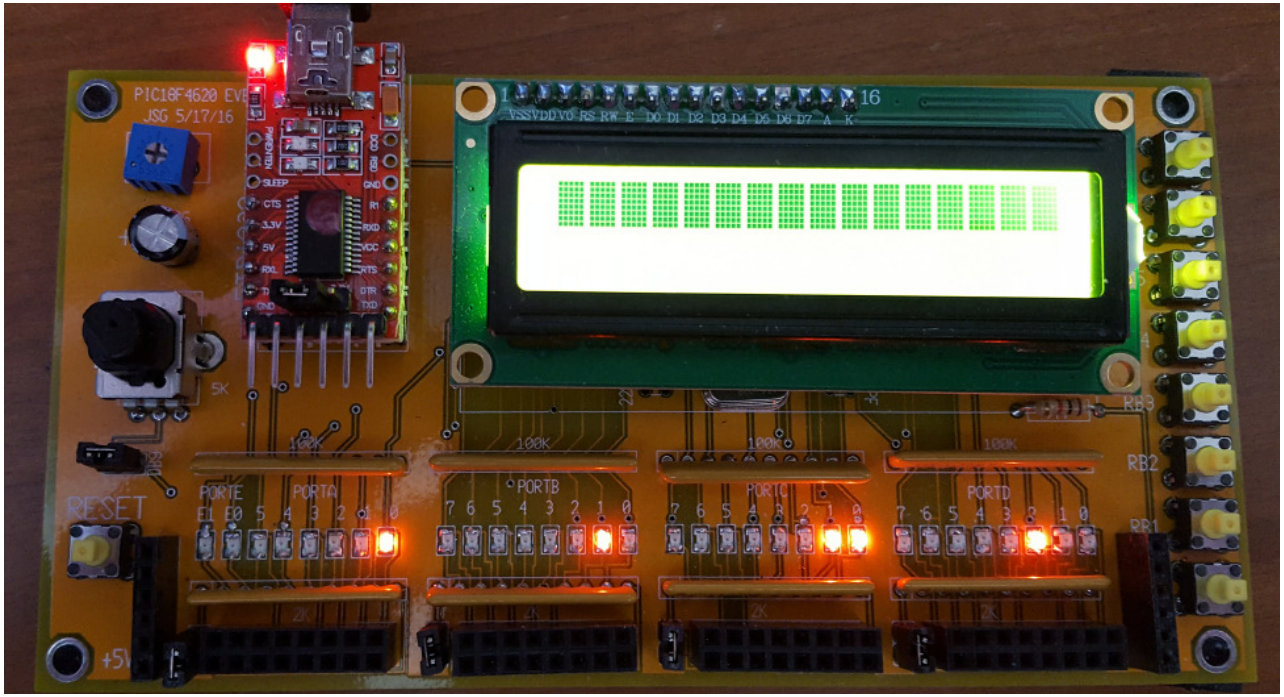
Sample Programs

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
```



1234.lst file

- Gives the memory location, machine code, and assembler 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.hex file

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

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

Example 2: Assembler Operations

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

```
#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
```