Microcontroller Programming
How to make something almost do something else

Raffi Krikorian
MAS.863
3 November 2003
What’s wrong with a P4?

**Pentiums**
- 50 million transistors
- $200
- Watts @ idle
- Complicated instruction set and usage model

**Microcontrollers**
- < 150,000 transistors
- $0.50 - $5.00
- 0.01s Watts while active
- “Simple” programming model
PIC16F876A
What is it?

- 8-bit processor that can be clocked from 50 kHz - 20 MHz
- 8K Flash program memory and 368 bytes SRAM
- 22 I/O pins (5 of which could be ADCs)
- 35 Instructions
- Hardware USART
- 2 Comparators
Memory

- Flash memory is where your "program" is stored
- SRAM is general purpose memory
- Registers can be memory mapped
Instructions

- Processors work with instructions
  - Move, Add, Jump, etc.
- Programs are just a series of instructions that the processor “steps” through

<table>
<thead>
<tr>
<th>Mnemonic,</th>
<th>Description</th>
<th>Cycles</th>
<th>16-Bit Opcode</th>
<th>Status Affected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDWF</td>
<td>Add W and f</td>
<td>1</td>
<td>00 0011 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>ADDWF</td>
<td>Add W with f</td>
<td>1</td>
<td>00 0011 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear f</td>
<td>1</td>
<td>00 0001 00000 0000</td>
<td>Z</td>
<td>2</td>
</tr>
<tr>
<td>COMF</td>
<td>Complement f</td>
<td>1</td>
<td>05 0101 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>DECF</td>
<td>Decrement f</td>
<td>1</td>
<td>00 0011 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>DECFNZ</td>
<td>Decrement f, Skip #0</td>
<td>1</td>
<td>00 0011 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>INC</td>
<td>Increment f</td>
<td>1</td>
<td>00 0101 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>INCFSZ</td>
<td>Increment f, Skip #0</td>
<td>1</td>
<td>00 0101 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>INCF</td>
<td>Increment f</td>
<td>1</td>
<td>00 0101 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>INCFNZ</td>
<td>Increment f, Skip #0</td>
<td>1</td>
<td>00 0101 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>CLRWF</td>
<td>Clear WF</td>
<td>1</td>
<td>00 0000 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>MOVWF</td>
<td>Move W to f</td>
<td>1</td>
<td>00 0000 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>00 0000 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>BCF</td>
<td>Bit Clear f</td>
<td>1</td>
<td>02 00bb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>BFCF</td>
<td>Bit Set f</td>
<td>1</td>
<td>03 00bb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>STFW</td>
<td>Bit Test, Sbit f</td>
<td>1</td>
<td>02 00bb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>STFB</td>
<td>Bit Test, Sbit f</td>
<td>1</td>
<td>03 00bb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>ADCW, ADNW</td>
<td>Add Literal and W</td>
<td>1</td>
<td>11 1101 00000 0000</td>
<td>Z, DC, Z</td>
<td>1,2</td>
</tr>
<tr>
<td>CALL</td>
<td>Call Subroutine</td>
<td>2</td>
<td>20 00bb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>GOTO</td>
<td>Go to Address</td>
<td>2</td>
<td>30 0abb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>GOF</td>
<td>Go to Address</td>
<td>2</td>
<td>30 0abb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>MOVW</td>
<td>Move Literal W</td>
<td>1</td>
<td>11 00bb 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>RET</td>
<td>Return from Interrupt</td>
<td>2</td>
<td>00 0000 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>RETURN</td>
<td>Return from Subroutine</td>
<td>2</td>
<td>00 0000 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>SLEEP</td>
<td>Go into Sleep mode</td>
<td>1</td>
<td>05 00aa 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>SUBIW, SUBN</td>
<td>Subtract W from Literal</td>
<td>1</td>
<td>11 1110 00000 0000</td>
<td>Z, DC, Z</td>
<td>1,2</td>
</tr>
<tr>
<td>XORW</td>
<td>Exclusive OR Literal with W</td>
<td>1</td>
<td>11 1110 00000 0000</td>
<td>Z</td>
<td>1,2</td>
</tr>
</tbody>
</table>
Adding two numbers

- Numbers are defined in locations in memory
- Move NUMBER1 to the W registers (working register)
- Add NUMBER2 to W and store the result back in W
- Move the value in W to the NUMBER3’s memory location

```asm
// NUMBER3 = NUMBER1 + NUMBER2

NUMBER1 EQU 0x20
NUMBER2 EQU 0x21
NUMBER3 EQU 0x22

MOVF NUMBER1, W
ADDWF NUMBER2, W
MOVWF NUMBER3
```
Counting down v1.0

- \( W \leftarrow 10 \)
- \( \text{COUNT} \leftarrow W \)
- Do some stuff
- If the Z bit is set in STATUS (the last operation == 0), then skip the next line
- If the GOTO is not skipped, then jump back to the do_loop

COUNT EQU 0x20

MOV LW d’10’
MOVWF COUNT

do_loop:
    // do stuff
    DECF COUNT, F
    BTFSS STATUS, Z
    GOTO do_loop
Counting down v2.0

- There are optimizations for common operations
- DECFSZ decrements the value in COUNT, stores it into COUNT, and if COUNT == 0 (if the Z bit is set), it skips the next instruction

```assembly
COUNT EQU 0x20
MOVLW d'10'
MOVWF COUNT
do_loop:
    // do stuff
    DECFSZ COUNT, F
    GOTO do_loop
```
Labels

- Labels allow you to mark a place in the code to GOTO or CALL
- GOTO jumps to a label
- CALL saves the current position, then jumps to a label
  - Allows for a RETURN to the current position
Simple Output

- Setup PORTC pin 0 (RC0) to be an output
- Turn PORTC pin 0 on
- Turn PORTC pin 0 off

BSF STATUS, RP0
BCF TRISC, 0
BSF PORTC, 0
BCF PORTC, 0
Simple Input

- Setup PORTB pin 0 (RB0) to be an input
- If RB0 is “low” (reads 0), then skip
  - this is the button press
- If RB0 is “high”, then do next instruction
  - Keeps us looping until the button press

```
BCF STATUS, RP0
BSF TRISB, 0
BSF STATUS, RP0

button_test:
  BTFSC PORTB, 0
  GOTO button_test
  // button pressed
```
Using the USART

- USART RX on RC7, TX on RC6
  - Make sure that RC7 is an input, and RC6 is an output in your code
- Load baud rate into SPBRG
- Receiver enable with CREN bit in RCSTA, transmitter enable with TXEN bit in TXSTA
- Put value you want to transmit into TXREG
- Loop on PIR1 bit RCIF to wait for bytes
- See sample code!
Assembler is fast! But…

• Large programs are hard to manage
• Allocating memory locations in your head is a pain
• Remembering the nuances of all the instructions can get annoying
• “Porting” your code to a different processor is almost impossible
Higher level languages

- C, Basic, Java, Lisp
- All “abstract” out the processor and let you focus on code
  - The compiler handles the conversion from the high level language to the assembly instructions
- There is a penalty, however...
  - Size of code
  - Execution speed
C vs. Assembler

**Assembler**
- MOVLW d’10’
- MOVWF COUNT
- flash:
  - BSF PORTC, 0
  - BCF PORTC, 0
  - DECFSZ COUNT, F
  - GOTO flash

**C**
- count = 10;
- while( count-- > 0 ) {
  - port_c = 1;
  - port_c = 0;
- }
Raffi vs. CCS compiled

**Raffi-written ASM**

- MOV LW d’10’
- MOVWF COUNT
- flash:
  - BSF PORTC, 0
  - BCF PORTC, 0
  - DecFSZ COUNT, F
- GOTO flash

**CCS generated ASM**

- MOV LW d’10’
- MOVWF COUNT
- flash:
  - MOVF COUNT, W
  - DecF COUNT, F
  - XORLW d’0’
  - BTFSC STATUS, Z
  - GOTO flash_done
- GOTO flash
- flash_done:
Getting the job done
Software

- MPLAB IDE: Microchip’s integrated development environment
- PICC: CCS C compiler for PICs
  - Integrates into MPLAB
- gpasm: open source assembler
Hardware

• PICSTART Plus or equivalent programmer

• Project ideas
  – Program a “bootloader” into the software and then load code over the serial port
  – Build a PIC programmer (you can easily do it with another PIC and some simple circuitry)
Attaching your board

- Pin 1 goes to 15V when programming, pins 28 and 27 bidirectionally talk to programmer
- Attach a header and connect that to the programmer
- Also connect power (5V) and ground
Compiling your code (MPLAB)
Getting ready to program (MPLAB)
Burn baby, burn (MPLAB)
What is it?

• 16-bit processor that can be clocked from 30 kHz - 8 MHz
• 8K Flash program memory and 256 bytes RAM
• 22 I/O pins (8 of which could be ADCs)
• Hardware USART
Why would you want to use it?

- This is where we’re going
- GCC as the compiler/toolchain
- JTAG programming/debugging port
- 350 μA max current draw (PIC on avg. draws 6 mA)
- Easy to bridge into much more powerful micros