Microcontroller Programming

How to make something almost do something else

Raffi Krikorian

MAS.863

3 November 2003

What’s wrong with a P4?

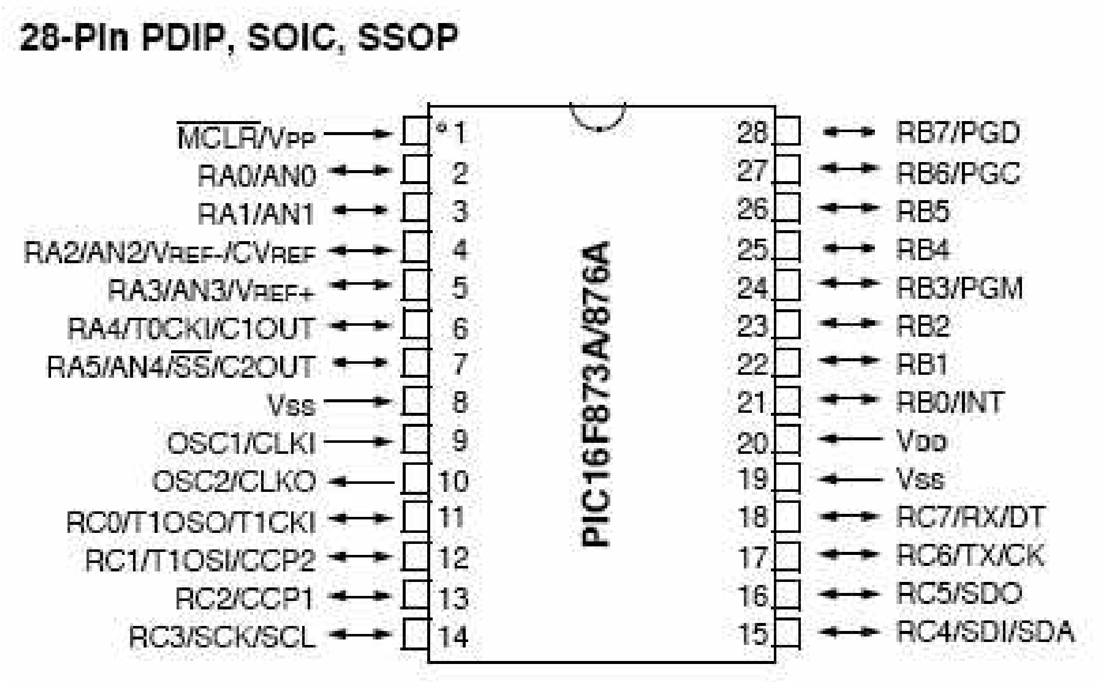
Pentiums

- 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

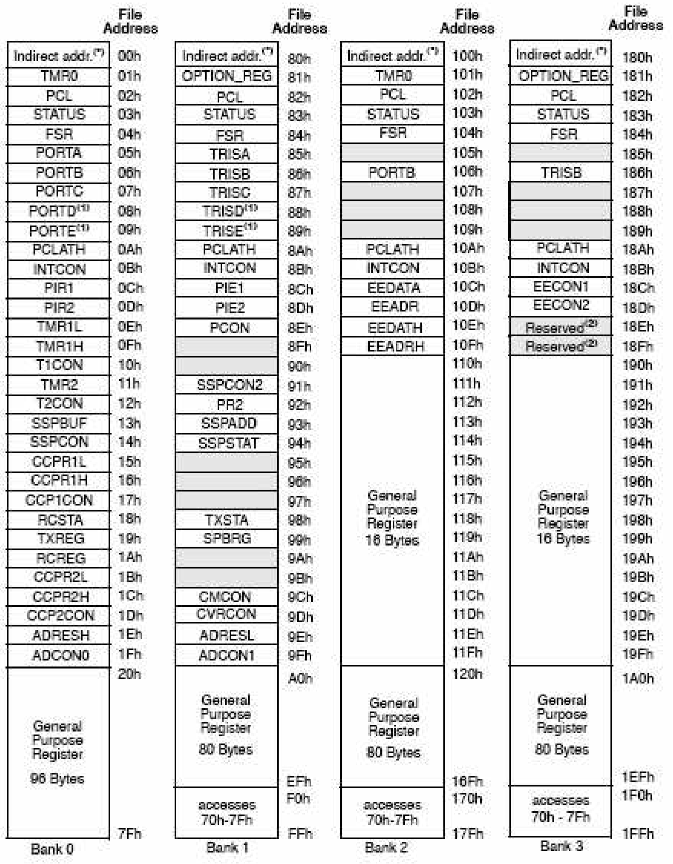
PIC16F876A


28-Pin PDIP, SOIC, SSOP
PIC16F873A/876A
MCLR/VPP 1
RA0/AN0 2
RA1/AN1 3
RA2/AN2/VREF-/CVREF 4
RA3/AN3/VREF+ 5
RA4/T0CKI/C1OUT 6
RA5/AN4/SS/C2OUT 7
VSS 8
OSC1/CLKI 9
OSC2/CLKO 10
RC0/T1OSO/T1CKI 11
RC1/T1OSI/CCP2 12
RC2/CCP1 13
RC3/SCK/SCL 14
28 RB7/PGD
27 RB6/PGC
26 RB5
25 RB4
24 RB3/PGM
23 RB2
22 RB1
21 RB0/INT
20 VDD
19 VSS
18 RC7/RX/DT
17 RC6/TX/CK
16 RC5/SDO
15 RC4/SDI/SDA

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



	File Address		File Address		File Address		File Address
Indirect addr.[*]	00h	Indirect addr.[*]	80h	Indirect addr.[*]	100h	Indirect addr.[*]	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD[1]	08h	TRISD[1]	88h		108h		188h
PORTE[1]	09h	TRISE[1]	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh	Reserved[2]	18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh	Reserved[2]	18Fh
T1CON	10h		90h	General Purpose Register 16 Bytes	110h	General Purpose Register 16 Bytes	190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch	CMCON	9Ch		11Ch		19Ch
CCP2CON	1Dh	CVRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Register 96 Bytes	20h	General Purpose Register 80 Bytes	A0h	General Purpose Register 80 Bytes	120h	General Purpose Register 80 Bytes	1A0h
			EFh		16Fh		1EFh
		accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h - 7Fh	1F0h
	7Fh		FFh		17Fh		1FFh
Bank 0		Bank 1		Bank 2		Bank 3	

- 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

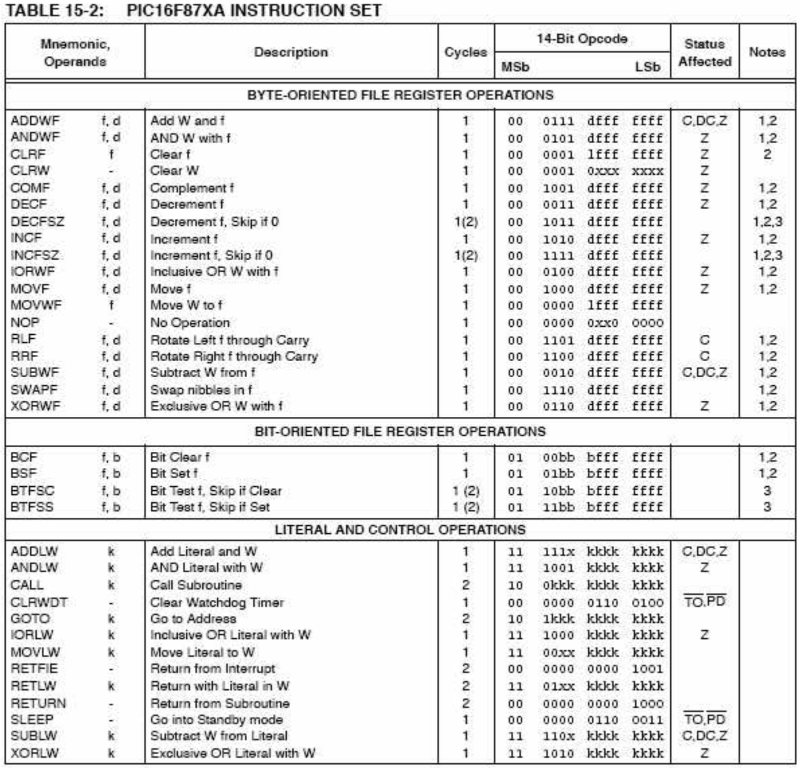


TABLE 15-2: PIC16F87XA INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb			LSb		
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	

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

```
// 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 <- 10
- COUNT <- 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

 MOVLW  d’10’
 MOVWF  COUNT
do_loop:
 // do stuff
 DECF   COUNT, F
 BTFSS  STATUS, Z
 GOTO   do_loop
```

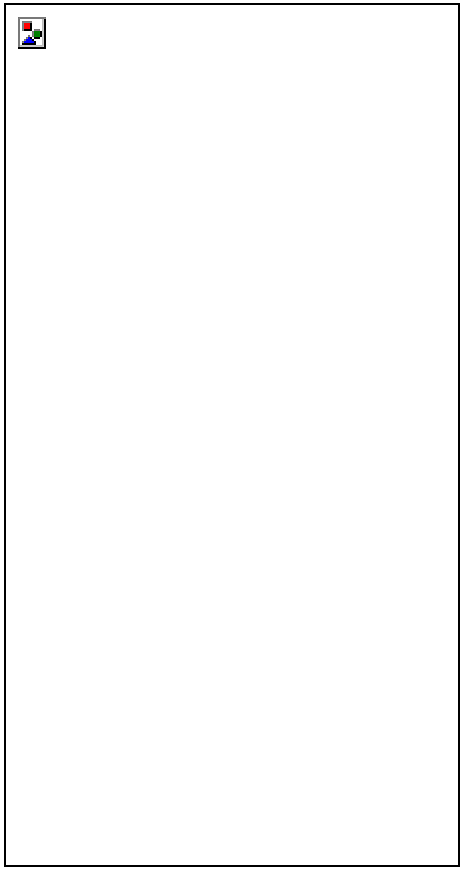
Counting down v2.0

- 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

```
COUNT EQU 0x20

 MOVLW  d'10'
 MOVWF  COUNT
do_loop:
 // do stuff
 DECFSZ COUNT, F
 GOTO   do_loop
```

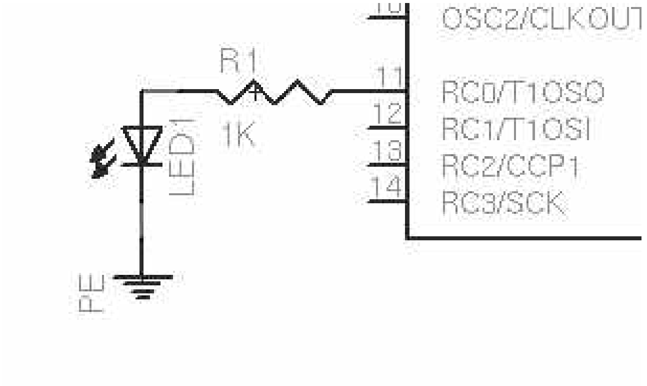
Labels



- 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

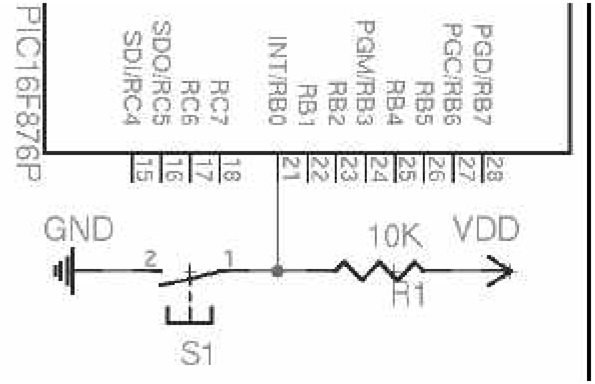


```
BSF STATUS, RP0
BCF TRISC, 0
BCF STATUS, RP0

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

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

CCS generated ASM

```
 MOVLW  d’10’
 MOVWF  COUNT
flash:
 MOVF   COUNT, W
 DECF   COUNT, F
 XORLW  d’0’
 BTFSC  STATUS, Z
 GOTO   flash_done
 MOVLW  d’1’
 MOVWF  PORTC
 CLRF   PORTC
 GOTO   flash
flash_done:
```

Getting the job done

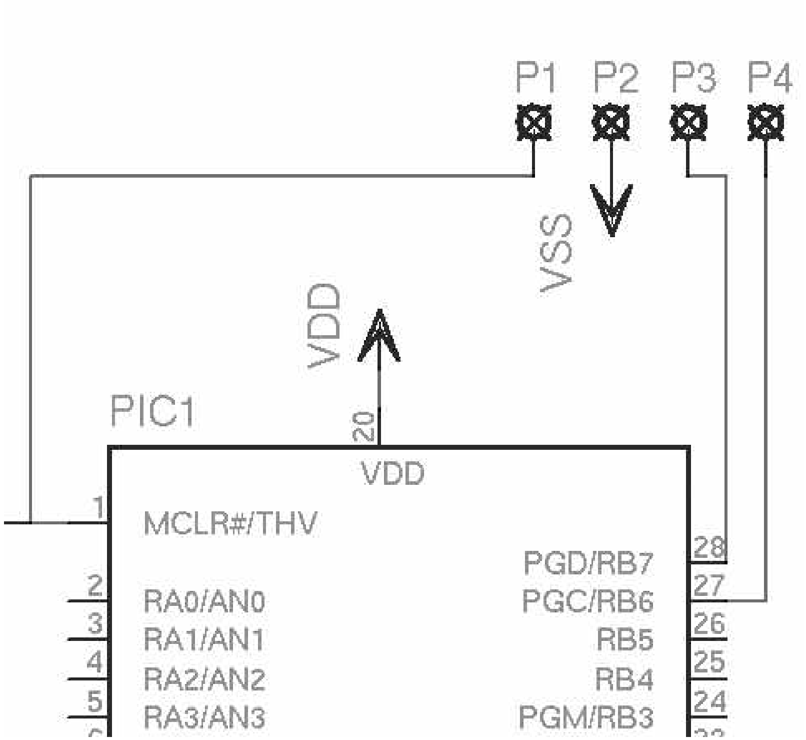
Software

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

Hardware

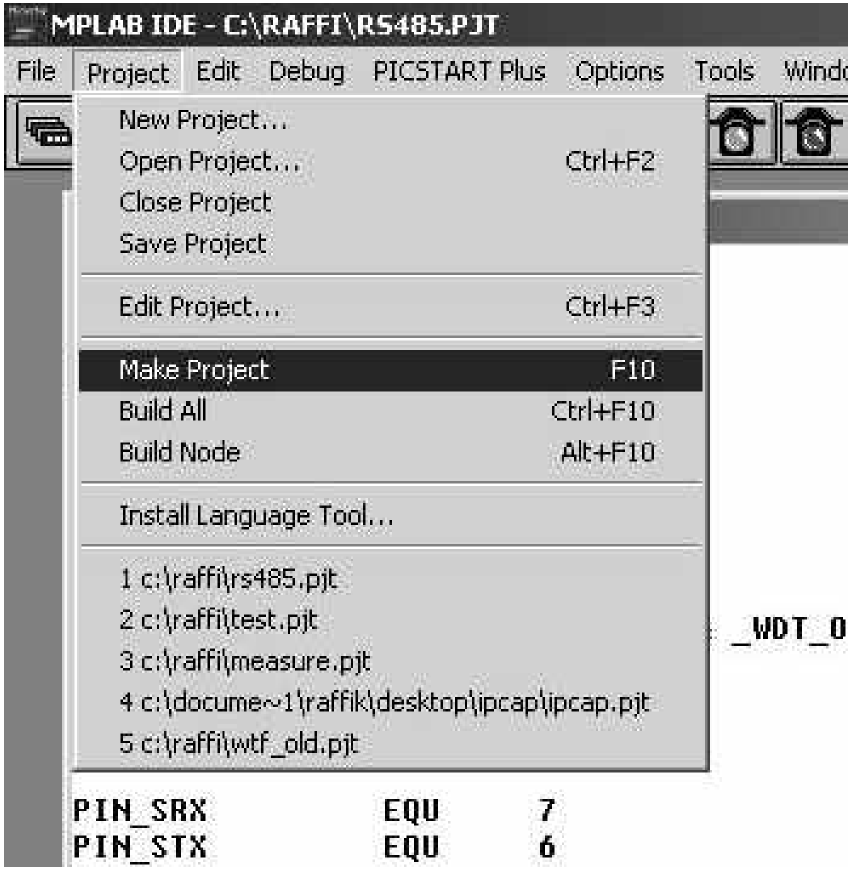
- PICSTART Plus or equivalent programmer
- Project ideas
  - Program a “bootloader” into the software and then load code over the serial port
  - Build a PIC programmer (you can easily do it with another PIC and some simple circuitry)

Attaching your board



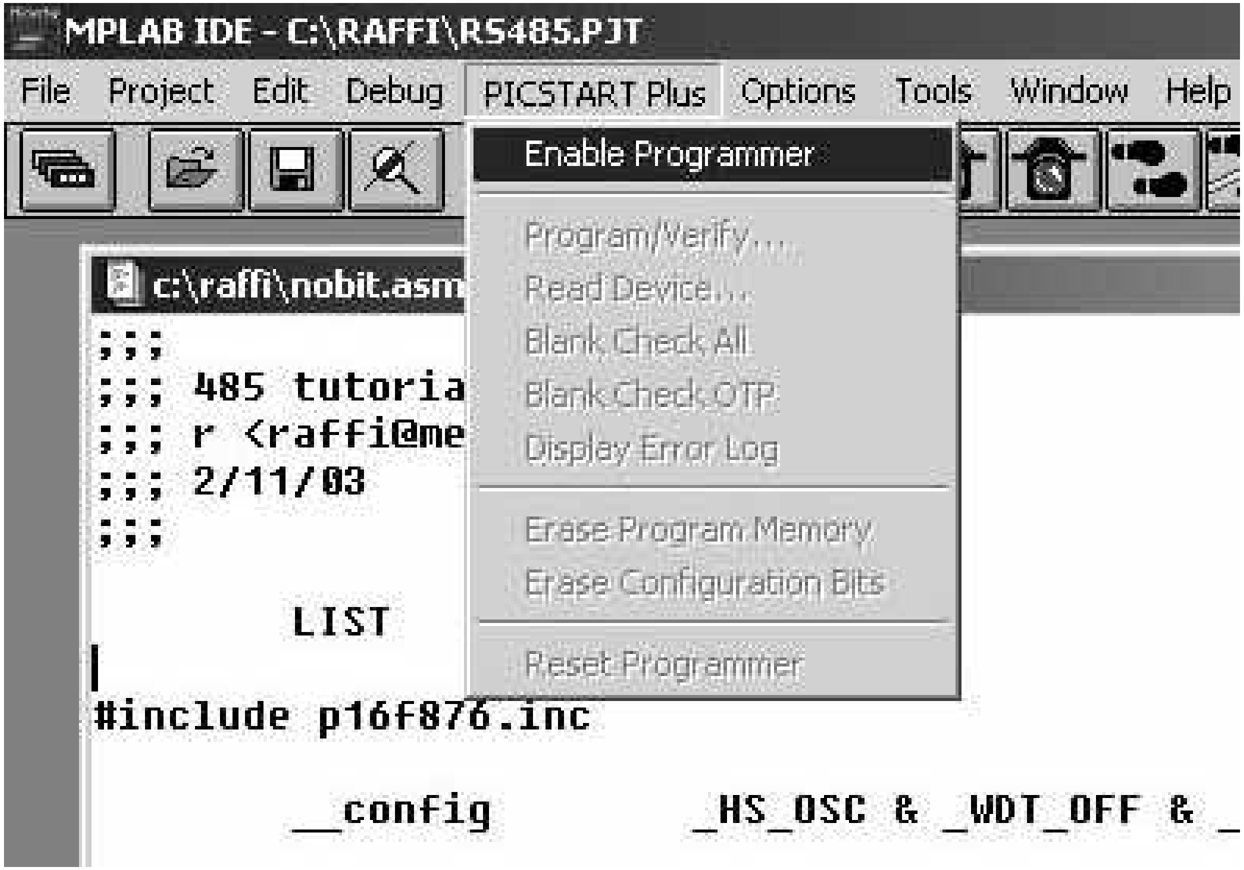
- Pin 1 goes to 15V when programming, pins 28 and 27 bidirectionally talk to programmer
- Attach a header and connect that to the programmer
- Also connect power (5V) and ground

Compiling your code (MPLAB)



```
_WDT_O
PIN_SRX          EQU      7
PIN_STX          EQU      6
```

Getting ready to program (MPLAB)



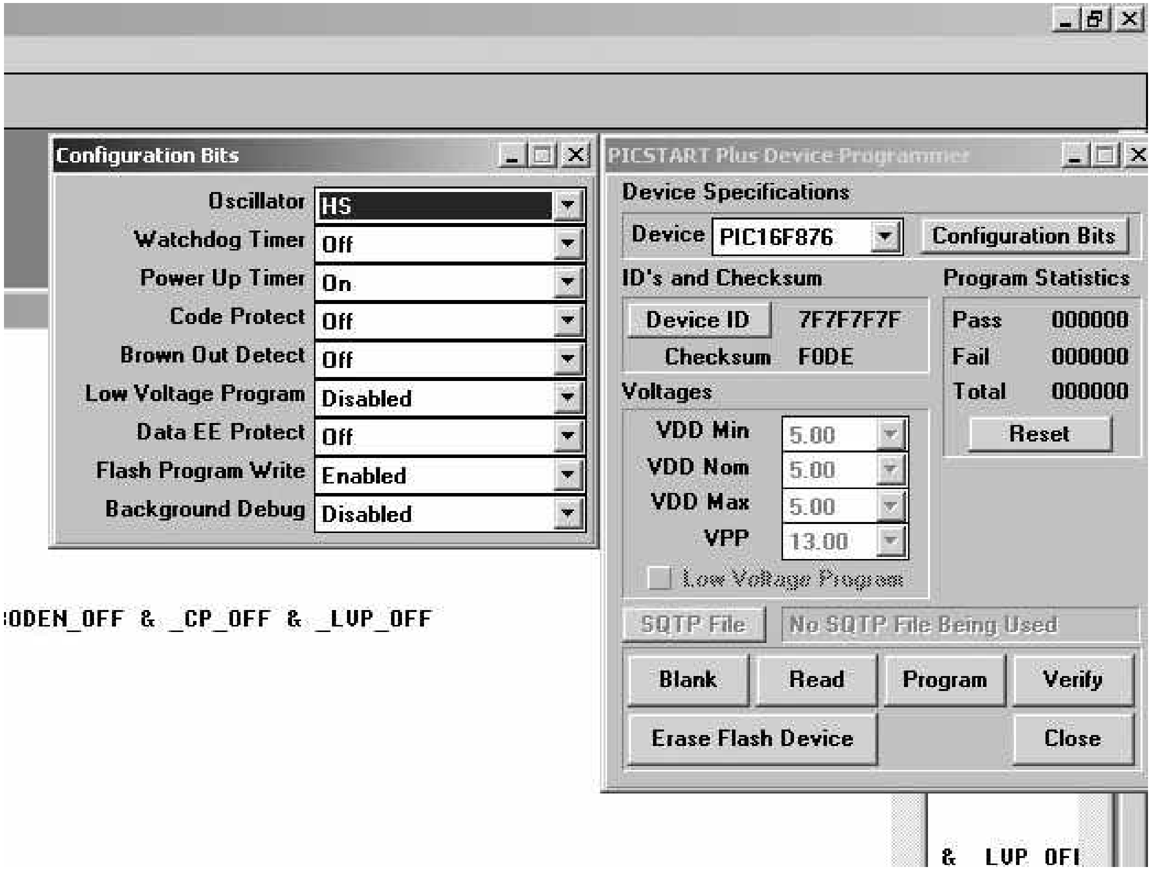
```
;;;
;;; 485 tutoria
;;; r <raffi@me
;;; 2/11/03
;;;

       LIST

#include p16f876.inc

       __config       _HS_OSC & _WDT_OFF & _
```

Burn baby, burn (MPLAB)



Configuration Bits

Oscillator	HS
Watchdog Timer	Off
Power Up Timer	On
Code Protect	Off
Brown Out Detect	Off
Low Voltage Program	Disabled
Data EE Protect	Off
Flash Program Write	Enabled
Background Debug	Disabled

PICSTART Plus Device Programmer

Device Specifications

Device PIC16F876

ID's and Checksum

Device ID 7F7F7F7F

Checksum F0DE

Program Statistics

Pass	000000
Fail	000000
Total	000000

Voltages

VDD Min	5.00
VDD Nom	5.00
VDD Max	5.00
VPP	13.00

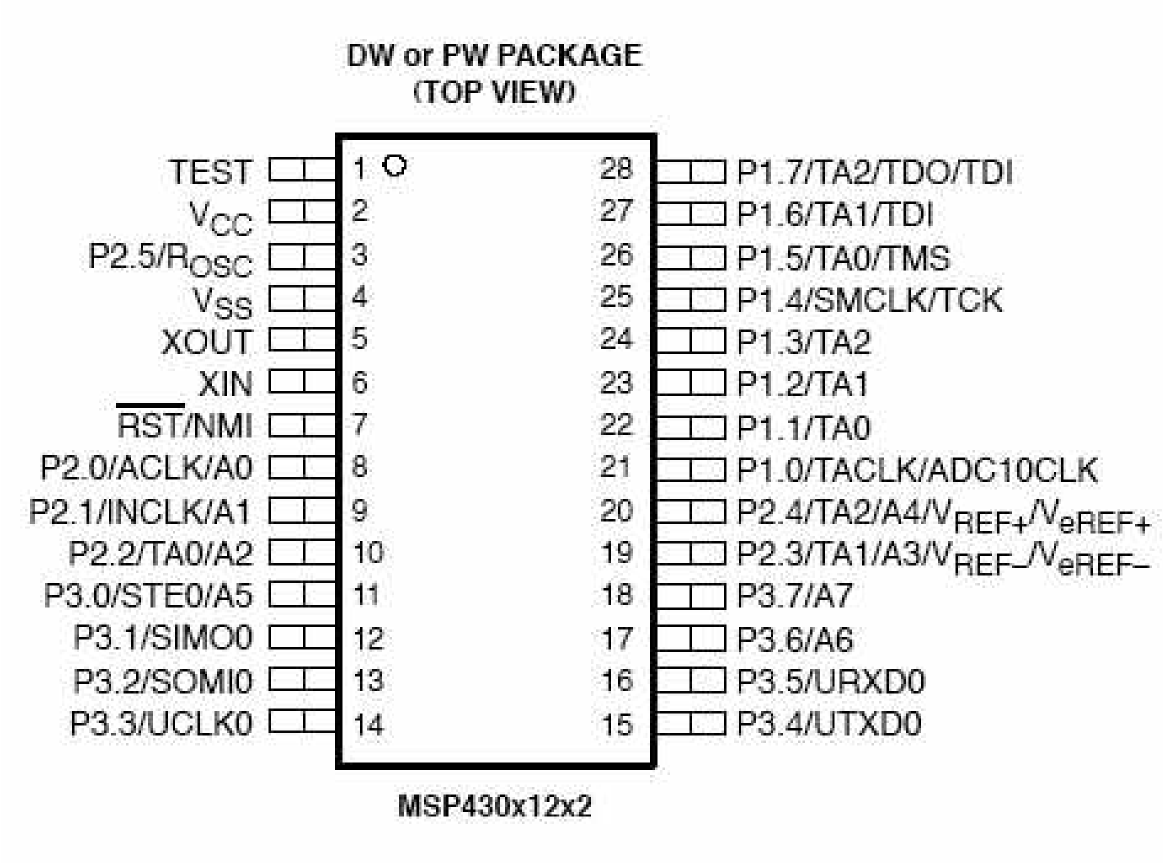
□ Low Voltage Program

No SQTP File Being Used

```
ODEN_OFF & _CP_OFF & _LVP_OFF
```

```
& LVP OFI
```

MSP430F1232



DW or PW PACKAGE
(TOP VIEW)

MSP430x12x2

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 uA max current draw (PIC on avg. draws 6 mA)
- Easy to bridge into much more powerful micros