Detalhes da Programação Assembly para a família Mid-Range PIC

Producing Executable Code

Assembly

Develop code on PC

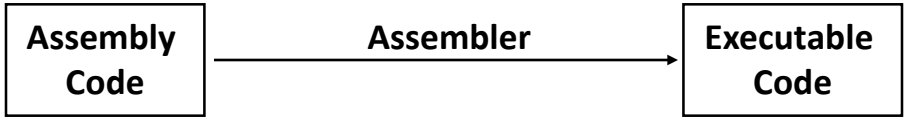
Edit / assemble / link / debug

Assembly

Convert assembly language code to machine language

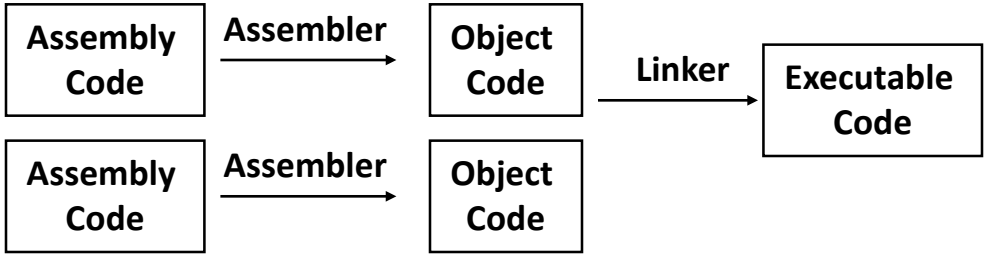
Absolute (executable) code

- Single module systems
- Real (absolute) addresses



Relocatable (object) code

- Complex multiple module systems
- Relative addresses



Linking

Combine + convert multiple object modules to executable code

Set real addresses

Producing Executable Code

Compiling

Develop code on PC

Edit / assemble / compile / link / debug

Compile

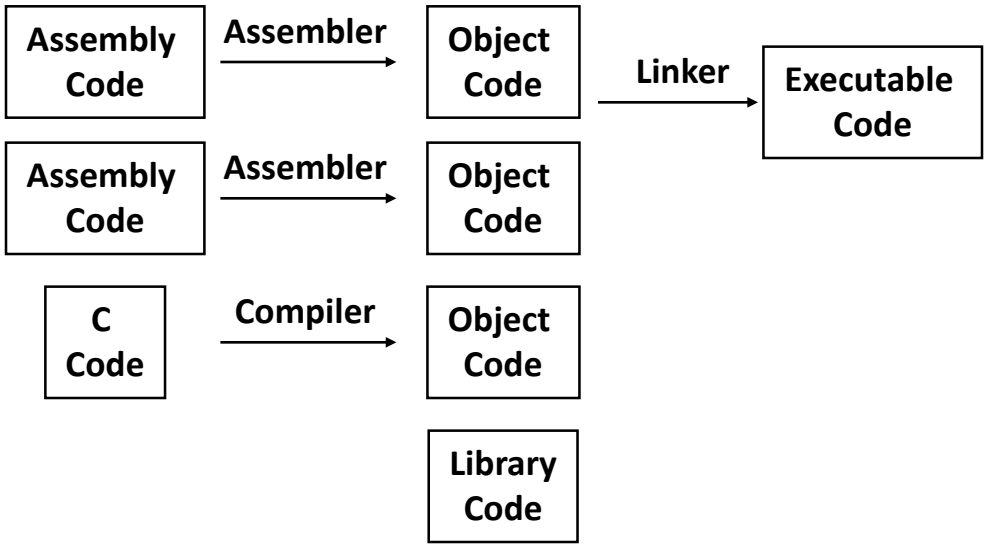
Convert C language code to object code

C compiler available for Mid-Range PICs and higher

Linking

Combine + convert multiple object modules to executable code

Set real addresses



MPLAB

Integrated Development Environment (IDE)

Text editor

Colorized formatting of PIC assembly code + directives

Assembler

`MPASM.EXE / MPASMWIN.EXE`

Linker

`MPLINK.EXE`

Library manager

`MPLIB.EXE`

Creates library (.lib) from several assembler programs

Simulator/Debugger

Simulates microcontroller on PC

Permits (limited) testing and debugging

Programmer

Program microcontroller device

Requires additional hardware

Elements of an Assembly Program

Instructions

- Assembly language instructions in PIC ISA
  - Assembly language instruction $\leftrightarrow$ machine language instruction

Labels

- Constant
  - Symbolic literal
- Variable
  - Pointer to data register holding value
- Line label
  - Pointer to location of instruction word
  - `GOTO label` $\Rightarrow$ `GOTO address_of_instruction_at_label`

Directives

- Commands to assembler executed at assembly time
- Examples
  - Arithmetic operations on addresses and labels
  - Assembly configuration

Program Organization

Absolute code

- Directives
  - Include header files
  - Set up constants + variables + macros
- Reset and interrupt code sections
- Main code section
  - Specify absolute (real) addresses of code sections
  - Absolute addresses $\Rightarrow$ direct addressing modes
- Subroutine code sections
- **`END`** directive

Relocatable code

- Similar to absolute code
- Differences
  - Variables defined in separate data section
  - No address specification for code sections
  - Direct / indirect addressing permitted

Assembler Input / Output Files

File	Extension	Contents	Generated by
Source	`.asm`	Assembly program	User
Include file	`.inc`	Device-specific definitions	Microchip
Listing	`.lst`	Program listing	Assembler (absolute code) or Linker (relocatable code)
Error File	`.err`	Error listing	
Hex File	`.hex, .hxl, .hxh`	Binary executable file	
Cross Reference	`.xrf`	Listing of symbols	
Object File	`.o`	Relocatable object code	Assembler

Constants

Symbolic literal — no change at run time

Defining constant

Assign value to symbol

Assembler converts symbol to literal

Directives

- **`equ`** — no reassignment
- **`set`** — reassignment as constant

Example

```
CONST1 equ 0xA5      ; assign
BIT equ 3            ; assign
prog1:
    movlw CONST1     ; W ← A5h
    addlw BIT        ; W ← 3 + W
```

Specifying constant values May be preceded by + or –	
Decimal	`D'167'` `.'167'`

Hexadecimal	`H'A7'`
	`0xA7`
	`0A7H`

Octal	`O'247'` `247O`
Binary	`B'10100111'`
ASCII	`A'Z'` `'Z'`

Variables

Pointer to data register holding value

Defining variable

- Relocatable code
  - Reserve memory space in data section
- Absolute code
  - Assign pointer to symbol
  - Use symbols as register name

Example (absolute)

```
CONST1 equ 0xA5     ; assign
REG1 equ 20h        ; assign
BIT equ 3           ; assign
prog1:
    movlw CONST1    ; W ← A5h
    movwf REG1      ; REG1 (address 20h) ← W
    bcf REG1, BIT   ; REG1<BIT> = bit 3 in reg 20h ← 0
```

Operations on Constants

Arithmetic at assembly time

Example

```
CONST1 equ 0xA5          ; assign
REG1 equ 20h             ; assign
INDEX equ 4              ; assign
BIT set 3                ; assign BIT ← 3
BIT set BIT + INDEX      ; reassign BIT ← 7
prog1:
    movlw CONST1    ; W ← A5h
    movwf REG1      ; REG1 (address 20h) ← W
    bcf REG1, BIT   ; REG1<BIT> = bit 7 in reg 20h ← 0
```

Operators on constants

Evaluated at assembly time

+	Addition	A1 + A2	Evaluate arithmetically
-	Subtraction	A1 - A2	
*	Multiplication	A1 * A2	
/	Division	A1/A2	
%	Modulo	A1%A2	
~	NOT	~A1	Evaluate bitwise
&	AND	A1 & A2	
|	OR	A1 | A2	
^	XOR	A1 ^ A2	
>>	Right shift	A1 >> 1	
<<	Left shift A1	<< 2	
!	NOT	!A1	Evaluate to TRUE = 1 or FALSE = 0
&&	AND	A1 && A2	
||	OR	A1 || A2	
>	Higher than	A1 > A2	
<	Less than	A1 < A2	
>	Higher or equal to	A1 >= A2	
<	Less or equal to	A1 <= A2	
=	Equal to	A1 == A2	
!=	Different than	A1 != A2	

Operators on Variable Pointers

Evaluated at assembly time

=	Logic or arithmetic assignment	var = 0	var = 0
++	Increment	var ++	var = var + 1
--	Decrement	var --	var = var - 1
+=	Add and assign	var += k	var = var + k
-=	Subtract and assign	var -= k	var = var - k
*=	Multiply and assign	var *= k	var = var * k
/=	Divide and assign	var /= k	var = var / k
%=	Mod and assign	var %= k	var = var % k
&=	AND and assign	var &= k	var = var & k
|=	OR and assign	var |= k	var = var | k
^=	XOR and assign	var ^= k	var = var ^ k
>>=	Right shift and assign	var >>= k	var = var >> k
<<=	Left shift and assign	var <<= k	var = var << k

Note

`var = 0` **equivalent to** `var set 0`

Define Block of Constants

For absolute code

Specify starting absolute address

Defines list of named symbols at sequential addresses

Used as variable pointers

Syntax

```
cblock [expr]
    label[:increment][,label[:increment]]
endc
```

Example

```
cblock 0x20             ; name_0 ← 20h
  name_0, name_1        ; name_1 ← 21h
  name_2, name_3        ; name_2 ← 22h
endc                    ; name_3 ← 24h
```

Address Operators and General Directives

Operator	Operation	Example
`$`	Current address	`goto $ ; loop in place`
`low`	Address low byte	`movlw low label ; W ← label<7:0>`
`high`	Address high byte	`movlw high label ; W ← 000.label<12:8>`

Directive	Operation	Example
`list`	Define device and default number system	`list p = 16f84a, r = dec`
`processor`		`processor 16f84a`
`radix`		`radix dec`
`#include`	Include file in source code	`#include file`
		`#include "file"`
		`#include <file>`
`org`	Assign address to instruction in absolute coding Start of code section	`org 0 ; set next instruction address 0` `; (reset section)`
		`org 4 ; set next instruction address 4` `; (interrupt section)`
		`org 20 ; set next instruction address` `; 20h`

Device Header File (Fragment)

```
; P16F84.INC Standard Header File, Version 2.00 Microchip Technology, Inc.
; This header file defines configurations, registers, and other useful bits of
; information for the PIC16F84 microcontroller. These names are taken to match
; the data sheets as closely as possible.
;
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
; Register Definitions
;
= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
W                EQU H'0000'
F                EQU H'0001'
;----- Register Files------------------------------
INDF             EQU H'0000'
TMR0             EQU H'0001'
PCL              EQU H'0002'
STATUS           EQU H'0003'
FSR              EQU H'0004'
PORTA            EQU H'0005'
PORTB            EQU H'0006'
                 :
;----- STATUS Bits --------------------------------
IRP              EQU H'0007'
RP1              EQU H'0006'
RP0              EQU H'0005'
NOT_TO           EQU H'0004'
NOT_PD           EQU H'0003'
Z                EQU H'0002'
DC               EQU H'0001'
C                EQU H'0000'
;----- INTCON Bits --------------------------------
                 :
```

Special Function Registers (SFR)
Not reserved names
Defined in header files

Skeleton for Absolute Code — 1

```
list p = 16f873             ; Declare device
#include <p16f873.inc>      ; include header file
;
; Define constants
;
DATA1 EQU 0x1
DATA2 EQU 0x2
;
; Define variables
;
w_temp equ 0x20
status_temp equ 0x21
X equ 0x22
Y equ 0x23
```

Alternative:

```
cblock 0x20
  w_temp
  status_temp
  X, Y
0x20endc
```

Skeleton for Absolute Code — 2

```
;
; Body of program
;
   org 0x000          ; Reset vector address
   movlw high PP      ; W ← 000.PP<12:8>
   movwf PCLATH       ; PCLATH ← PP<12:8>
   goto PP            ; PCL ← PP<7:0> = start of main
;
```

Skeleton for Absolute Code — 3

```
   org 0x004                ; Interrupt vector address
   movwf w_temp             ; w_temp ← W (no flags)
   movf STATUS, W           ; W ← STATUS (write Z)
   bcf STATUS, RP0          ; Select bank 0
   movwf status_temp        ; status_temp ← W = STATUS
;                                          (no write Z)
; Interrupt Service Routine Here
;
   bcf STATUS, RP0          ; Select bank 0
   movf status_temp, W      ; Restore STATUS (write Z)
   movwf STATUS             ;                (no write Z)
   swapf w_temp, f          ; swap nibbles to w_temp
   swapf w_temp, W          ; re-swap nibbles to W
                            ;                (no write Z)
   retfie                   ; Return from interrupt
```

Skeleton for Absolute Code — 4

```
PP:
   clrf X                    ; zero variables
   clrf Y
;
; main program
;
; subroutine call
;
   movlw high SR1            ; W ← 000.SR1<12:8>
   movwf PCLATH              ; PCLATH ← SR1<12:8>
   call SR1                  ; push PC
                             ; PCL ← SR1<7:0>
   goto $                    ; spin loop (jumps to here)
SR1:
; code of subroutine SR1
return                       ; Return to main
```

Section Declarations for Relocatable Code

Initialized data section

Syntax

```
[label] idata [RAM_address]
```

Defaults

`label = .idata`

`RAM_address` set by linker

Example

```
        idata
LimitL  dw 0
LimitH  dw D'300'
Gain    dw D'5'
Flags   res 1
String  db 'Hi there!'
```

Data Directives

db

Inserts data byte at next memory address

dw

Inserts 2 data bytes in little endian order

res n

Inserts **n** data 0 bytes

Section Declarations for Relocatable Code

Uninitialized Data Section

Syntax

```
[label] udata [RAM_address]
```

Defaults

`label = .udata`

`RAM_address` set by linker

Example

```
        udata
Var1    res 1
Double  res 2
```

Data Directive

`res n`

reserves **n** data bytes

Section Declarations for Relocatable Code

Shared Uninitialized Data Section

Syntax

```
[label] udata_shr [RAM_address]
```

Registers shared across memory

Values copied to file address in all banks

Default `label = .udata_shr`

Example

```
        udata_shr
Var1    res 1
Double  res 2
```

Var1	Var1	Var1	Var1
Double	Double	Double	Double
Bank 0	Bank 1	Bank 2	Bank 3

Section Declarations for Relocatable Code

Overlayed Uninitialized Data Section

Syntax

`[label] udata_ovr [RAM_address]`

Registers declared in section overlayed

- Other `udata_ovr` sections with same name overwrite same space
- Multiple temporary variable sets declared at one memory location

Default `label = .udata_ovr`

Example

```
Temps      udata_ovr
  Temp1      res 1
  Temp2      res 1
;
; work with Temp1, Temp2
;
Temps      udata_ovr
  NewTemp1   res 1     ; reallocate location Temp1
  NewTemp2   res 1     ; reallocate location Temp2
```

Section Declarations

Overlayed Uninitialized Da

Example (program)

```
        include  "p16f84.inc"
; The SQR subroutine
; ********************************************************
; * FUNCTION: Squares one byte to give a 2-byte result   *
; * EXAMPLE : X = 10h (16), SQUARE = 0100h (256)         *
; * ENTRY   : X in W                                     *
; * EXIT    : SQUARE:2 in shared uninitialized data      *
; ********************************************************
; Static data
        udata
SQUARE  res     2       ; High:Low byte of square
; Local data
        udata_ovr
X       res     1       ; Place for X
X_COPY_L res    1       ; Holds a copy of X
X_COPY_H res    1       ; Copy X overflow hi byte
TEXT    code
; Task 1: Zero double-byte square
SQR     clrf    SQUARE
        clrf    SQUARE+1
; Task 2: Copy and extend X to 16-bits
        movwf   X           ; Put X away into Data memory
        movwf   X_COPY_L    ; Copy of X
        clrf    X_COPY_H    ; and extend to double byte
; Task 3: DO
  ; Task 3A: Shift X right once
SQR_LOOP bcf    STATUS,C    ; Clear carry
        rrf     X,f         ; Shift
   ; Task 3B: IF Carry == 1 THEN add 16-bit shifted X to square
        btfss   STATUS,C    ; IF C == 1 THEN do addition
         goto   SQR_CONT    ; ELSE skip this task
        movf    X_COPY_L,w  ; DO addition
        addwf   SQUARE+1,f  ; First the low bytes
        btfsc   STATUS,C    ; IF no carry THEN do high bytes
         incf   SQUARE,f    ; ELSE add carry
        movf    X_COPY_H,w  ; Next the high bytes
        addwf   SQUARE,f
     ; Task 3C: Shift 16-bit copy of X right once
SQR_CONT bcf    STATUS,C    ; Zero Carry-in
        rlf     X_COPY_L,f
        rlf     X_COPY_H,f
     ; WHILE X not zero
        movf    X,f         ; Test multiplier for zero
        btfss   STATUS,Z
         goto   SQR_LOOP    ; IF not THEN go again
FINI    return              ; ELSE finished
        global SQUARE, SQR
        end
```

Section Declarations for Relocatable Code

Code Section

Syntax

```
[label] code [RAM_address]
```

Defaults

`label = .code`

`RAM_address` set by linker

Code Directive

```
pagesel  start
```

Generates code:

```
movlw high start
movwf PCLATH
```

Example

```
RST     CODE     0x0      ; placed at address 0x0
    pagesel  start
    goto     start
PGM     CODE              ; relocatable code section
start:
  clrw
  goto  $
        CODE              ; relocatable code section
  nop                     ; default section name .code
  end
```

Skeleton for Relocatable Code — 1

```
list p = 16f873             ; Declare device
#include <p16f873.inc>      ; include header file
;
; Define constants
;
DATA1 EQU 0x1
DATA2 EQU 0x2
;
; Define variables
;
udata_shr                   ; data shared across banks
  w_temp res 1
  status_temp res 1
  X res 1
  Y res 1
```

Skeleton for Relocatable Code — 2

```
;
; Body of program
;
Rst_vector code 0          ; Reset vector address
  pagesel PP
  goto PP
;
Intr_vector code 4         ; Interrupt vector address
  goto SR_Int
;
```

Skeleton for Relocatable Code — 3

```
Intr_Prog code 5          ; ISR
SR_Int:
  movwf w_temp            ; w_temp ← W (no flags)
  movf STATUS, W          ; W ← STATUS (write Z)
  bcf STATUS, RP0         ; Select bank 0
  movwf status_temp       ; status_temp ← W = STATUS
;                                    (no write Z)
; Interrupt Service Routine Here
;
  bcf STATUS, RP0         ; Select bank 0
  movf status_temp, W     ; Restore STATUS (write Z)
  movwf STATUS            ;              (no write Z)
  swapf w_temp, f         ; swap nibbles to w_temp
  swapf w_temp, W         ; re-swap nibbles to W
                          ;              (no write Z)
  retfie                  ; Return from interrupt
```

Skeleton for Relocatable Code — 4

```
Prog_Principal code
PP:
   clrf X                    ; zero variables
   clrf Y
;
; main program
;
; subroutine call
;
   pagesel
   call SR1
   goto $                    ; spin loop (jumps to here)
Subroutines code
SR1:
; code of subroutine SR1
return                       ; Return to main
```

Define — Single Line Macros

Syntax

```
#define name [string]
```

Text substitution

`name` in assembly code replaced by string

Permits parameter substitution

Example

```
#define length 20
#define width 30
#define depth 40
#define circumference(X,Y,Z) (X + Y + Z)
:
Size equ circumference(length, width, depth)
```

`Size` evaluates to 20+30+40 = 90

Macros

Syntax

```
macro_name macro [arg_def1, arg_def2,…]
                 [ local label [, label, label,…]]
;
; Body of macroinstruction
;
endm
```

Optional arguments

`arg_def1, arg_def2`

`local labels` — local to macro definition

Call macro

```
macro_name [arg1, arg2,…]
```

Macro Example

```
Convert macro HEXA, ASCII          ; Declare macro
local add30, add37, end_mac        ; local labels
  movf HEXA, W                     ; HEXA ← W
  sublw 9                          ; W ← 9 - W
                                   ; C ← (W > 9)
     movf HEXA, W                  ; C not changed
     btfsc STATUS, C               ;  if (C == 0){
     goto add30   ← C != 0         ;     W ← W + 37h
  add37:                           ;  }
     addlw 37h    ← C = 0          ;  else {
     goto end_mac                  ;     W ← W + 30h
  add30:                           ;  }
     addlw 30h
 end_mac:
 movwf ASCII                       ; ASCII ← W
 endm                              ; End of macro
 Convert HX, ASC                   ; insert macro code here
```

Macros for Register Save / Restore

```
PUSH_MACRO MACRO          ; Save register contents
  MOVWF W_TEMP            ; Temporary register ← W
  SWAPF STATUS,W          ; W ← swap STATUS nibbles
  MOVWF STATUS_TEMP       ; Temporary register ← STATUS
ENDM                      ; End this Macro


POP_MACRO MACRO           ; Restore register contents
SWAPF STATUS_TEMP,W       ; W ← swap STATUS
MOVWF STATUS              ; STATUS  ← W
SWAPF W_TEMP,F            ; W_Temp ← swap W_Temp
SWAPF W_TEMP,W            ; W ← swap W_Temp s
                          ; no affect on STATUS
ENDM                      ; End this Macro
```

Typical Interrupt Service Routine (ISR) — 1

```
org ISR_ADDR          ; store at ISR address
PUSH_MACRO            ; save context registers W, STATUS
CLRF STATUS           ; Bank0
     ; switch implementation in PIC assembly language
BTFSC PIR1, TMR1IF    ; skip next if (PIR1<TMR1IF> == 1)
GOTO T1_INT           ; go to Timer1 ISR
BTFSC PIR1, ADIF      ; skip next if (PIR1<ADIF> == 1)
GOTO AD_INT           ; go to A/D ISR
BTFSC PIR1, LCDIF     ; skip next if (PIR1<LCDIF> == 1)
GOTO LCD_INT          ; go to LCD ISR
BTFSC INTCON, RBIF    ; skip next if (PIR1<RBIF> == 1)
GOTO PORTB_INT        ; go to PortB ISR
GOTO INT_ERROR_LP1    ; default ISR
```

Typical Interrupt Service Routine (ISR) — 2

```
T1_INT                  ; Timer1 overflow routine
    :
    BCF PIR1, TMR1IF    ; Clear Timer1 overflow interrupt flag
    GOTO END_ISR        ; Leave ISR
AD_INT                  ; Routine when A/D completes
    :
    BCF PIR1, ADIF      ; Clear A/D interrupt flag
    GOTO END_ISR        ; Leave ISR
LCD_INT                 ; LCD Frame routine
    :
    BCF PIR1, LCDIF     ; Clear LCD interrupt flag
    GOTO END_ISR        ; Leave ISR
PORTB_INT               ; PortB change routine
    :
END_ISR                 ; Leave ISR
    POP_MACRO           ; Restore registers
    RETFIE              ; Return and enable interrupts
```

Accessing External Modules

Import Label

`extern label [, label...]`

Declare symbol

- Used in current module
- Defined as global in different module

Must appear before label used

Export Label

`global label [, label...]`

Declare symbol

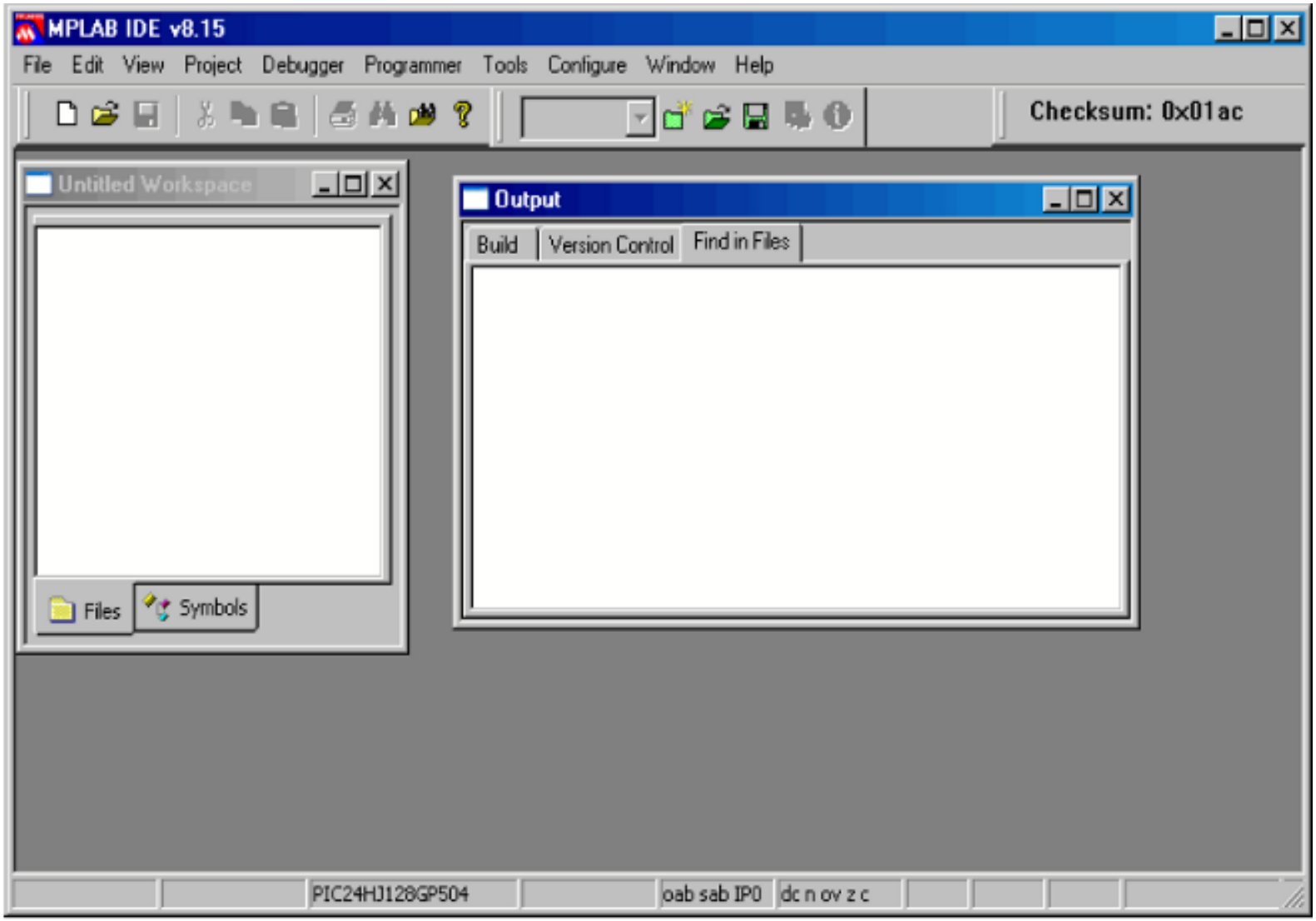
- Defined in current module
- Available to other modules

Example

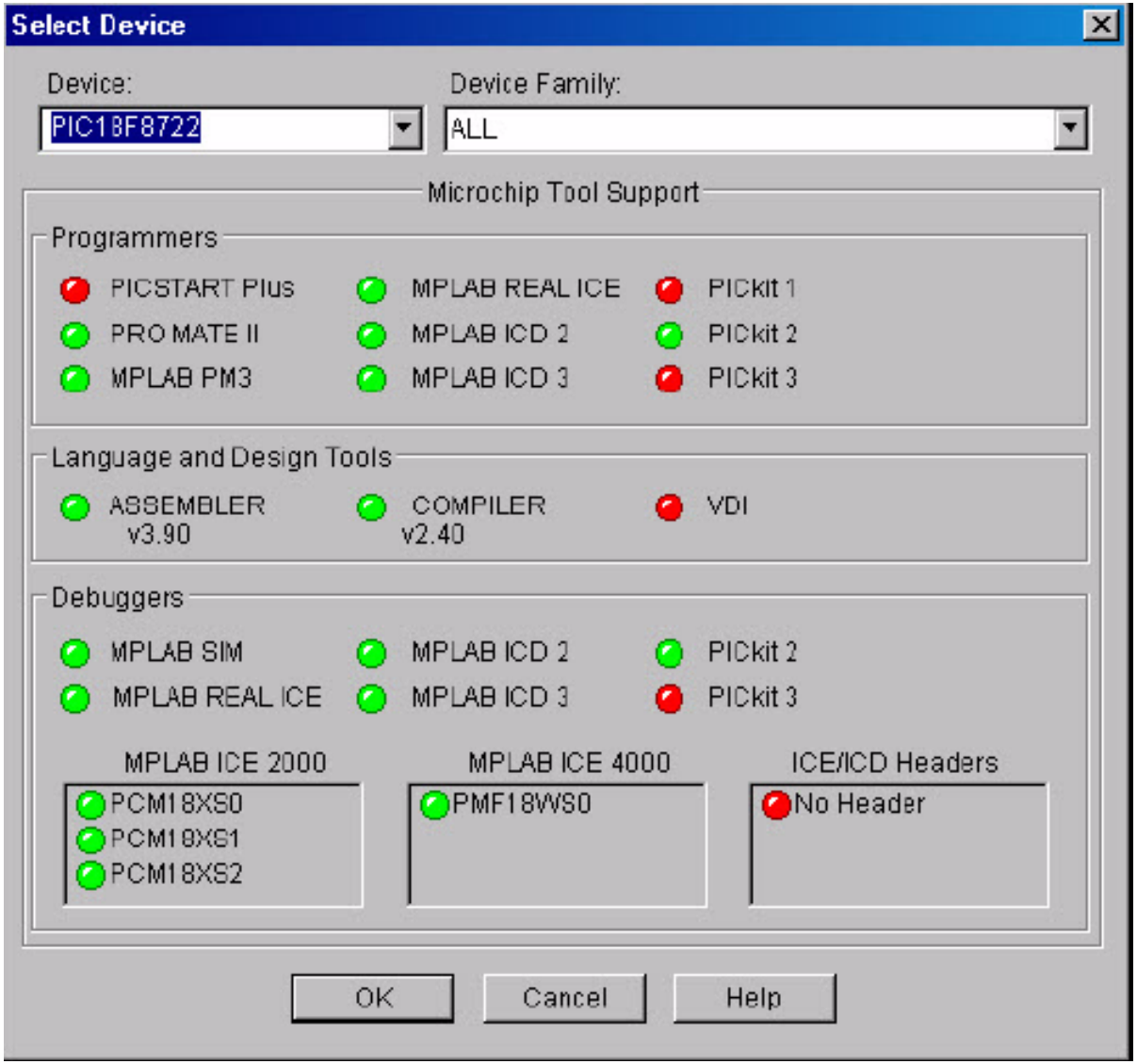
```
; in module 1
      global Var1, Var2
      global AddThree
;
        udata
Var1    res 1
Var2    res 1
        code
AddThree:
        addlw 3
        return

; in module 2
      extern Var1, Var2
      extern AddThree
clrf Var1
clrf Var2
call AddThree
```

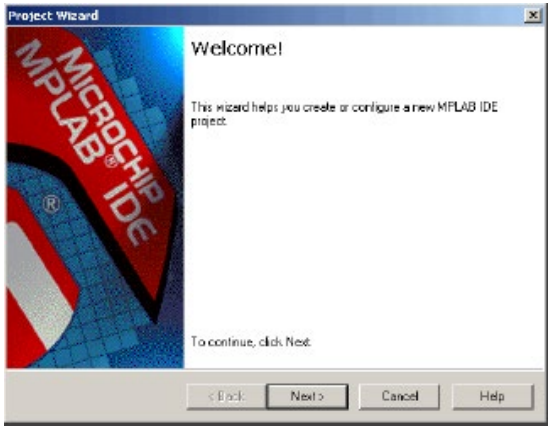
Start MPLAB IDE



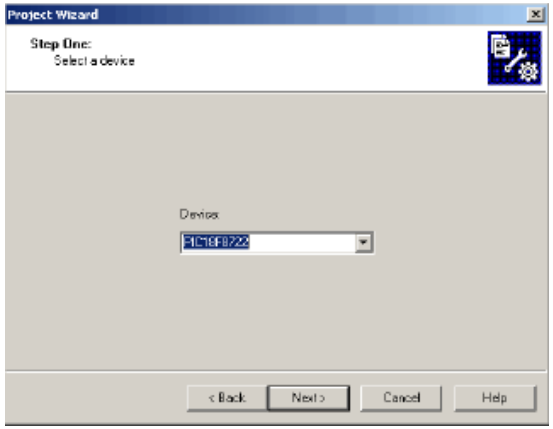
Configure > Select Device



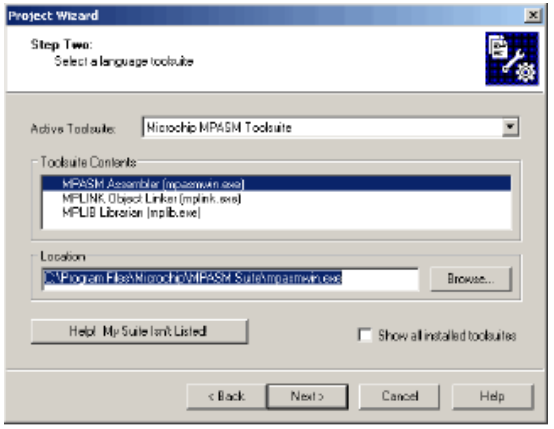
Project > Project Wizard



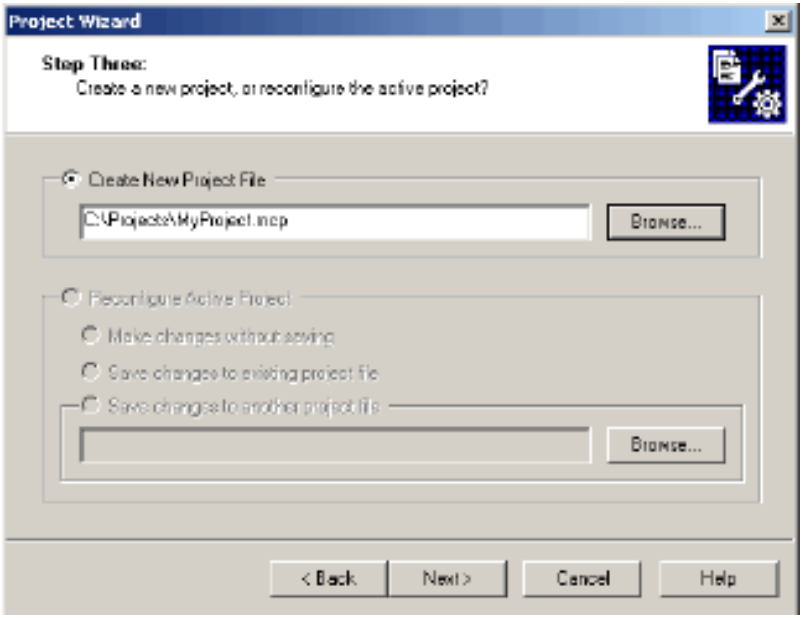
Wizard



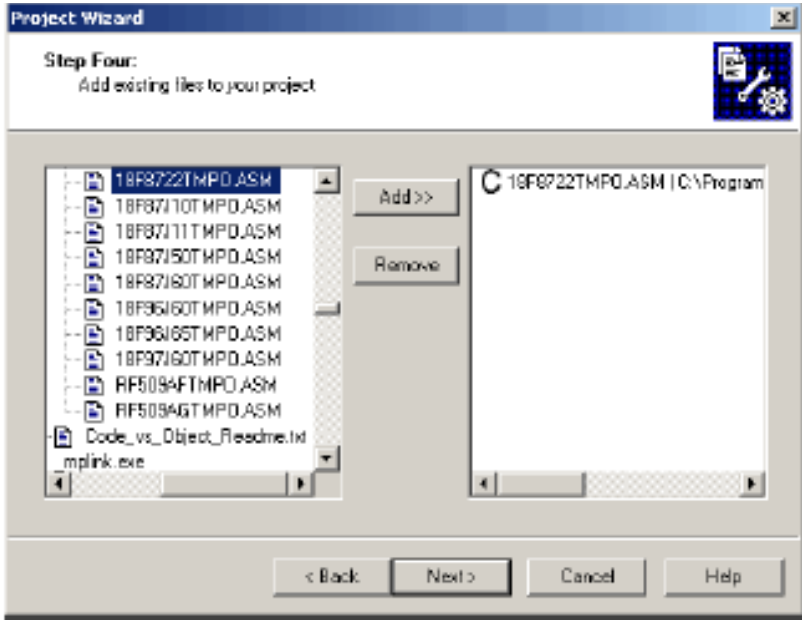
Select PIC Device



Select Language Tools



Save Project by Pathname



Add Device Template File

Build Project

Either

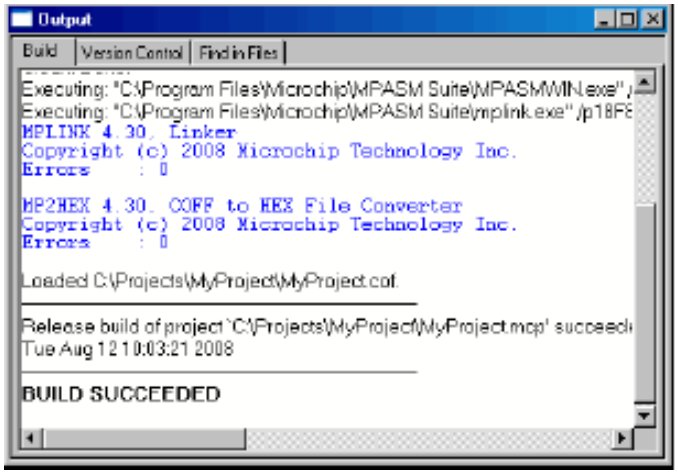
Project > Build All

Right click on project name in Project Window > Build All

Click Build All icon on Project toolbar

Output window shows result of build process

Should be no errors or warnings for default template file



Code

Add constants / variables / code / directives / macros

Rebuild

Testing Code with Simulator

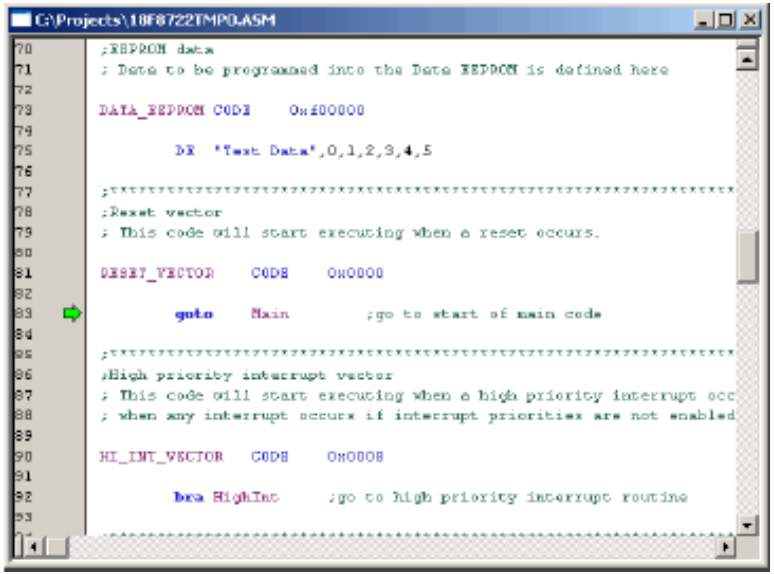
Debugger > Select Tool > MPLAB SIM

Debug toolbar opens

Debugger > Reset > Processor Reset

Assembly code editor opens

Green arrow points to program start (main)



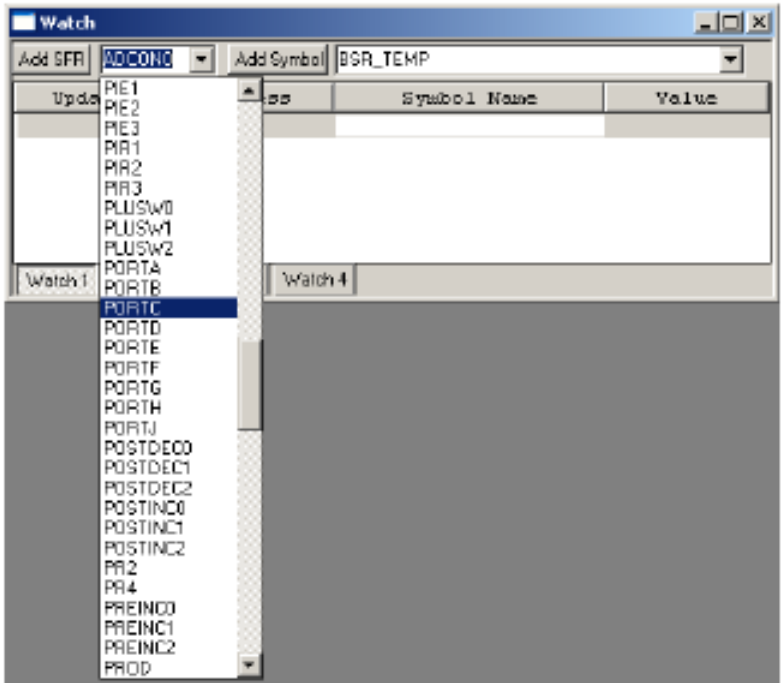
Step Into

Run program in trace mode (single step)

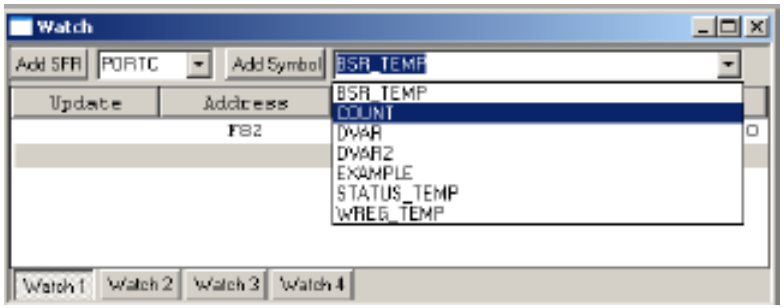
View > Watch

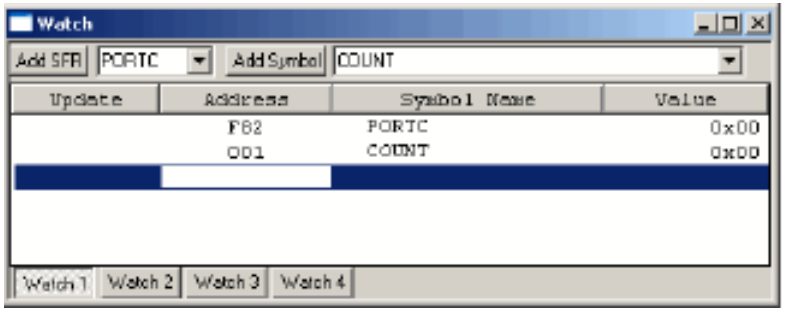
Choose + Add items to watch list

SFRs



Symbols





Breakpoints

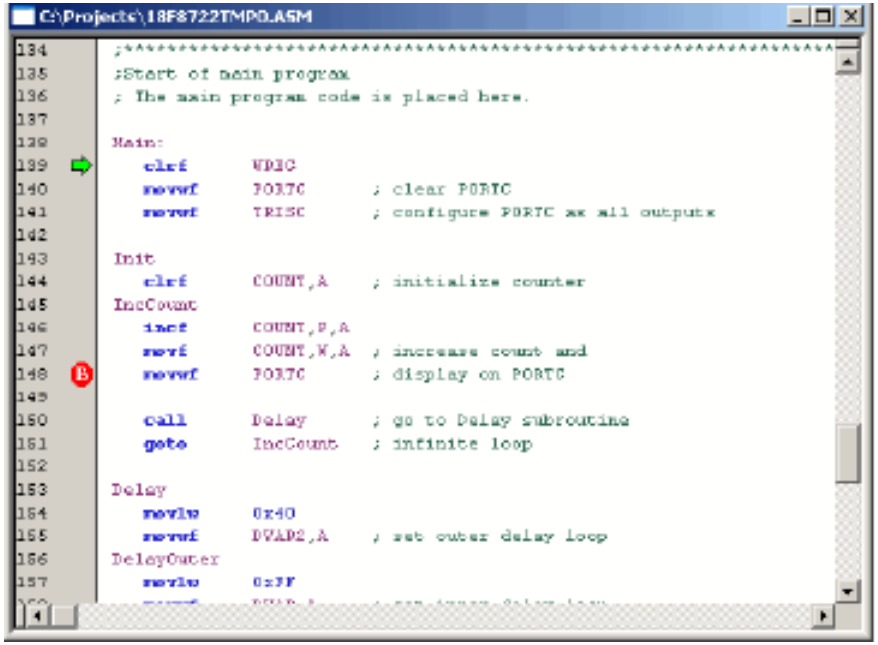
Set breakpoint

Double-click on line of code

Right click > choose Set Breakpoint from menu

Run

Program stops before breakpoint

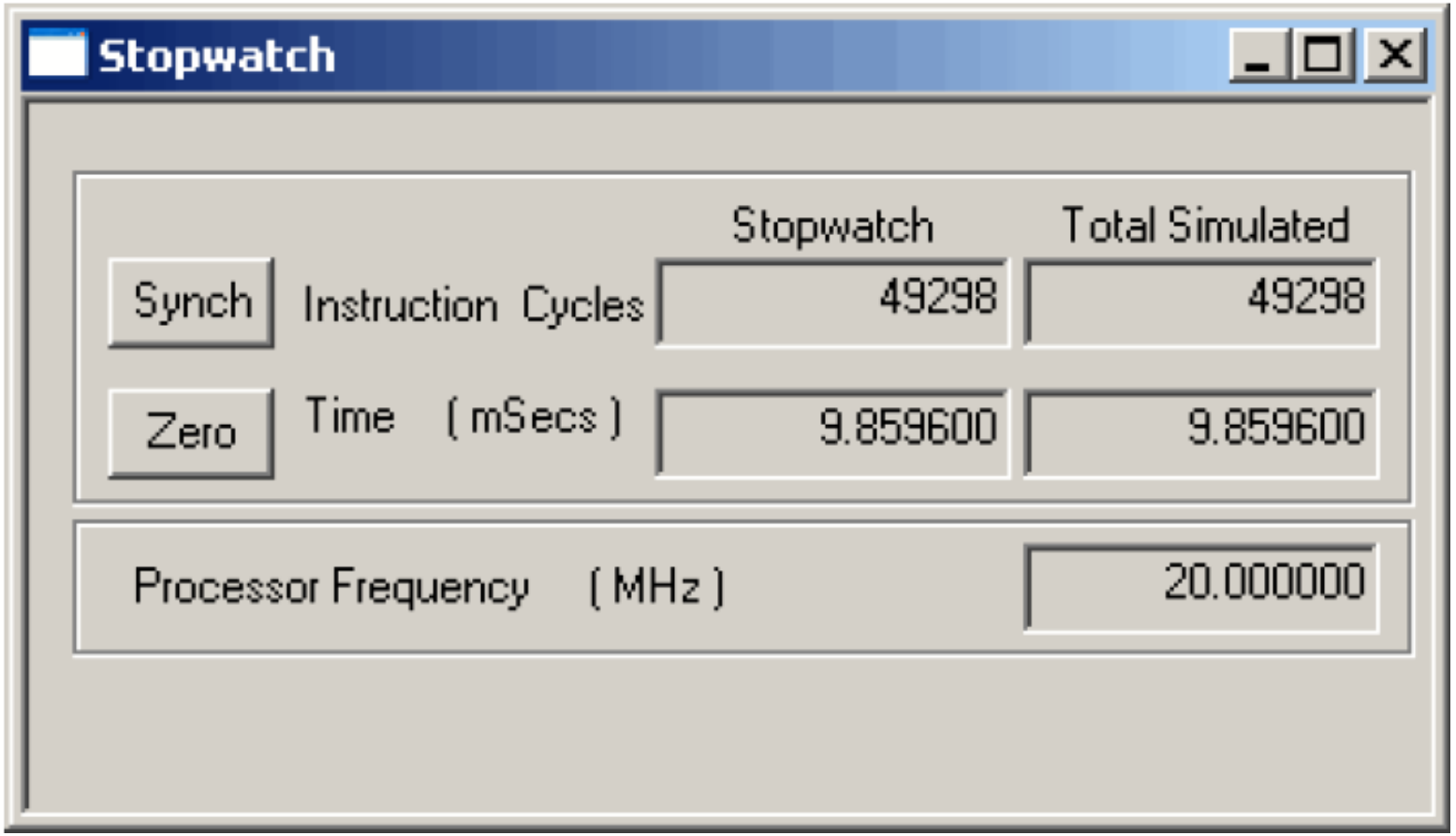


Stopwatch

At breakpoint

Reports clock cycles

Estimates runtime



Delay Timer with Timer0 — 1

Internal RC oscillator

$T_{CY} = 4 \times 1 / (4\ MHz) = 1\ \mu s = 0.001\ ms$

1 ms = 1000 counts

Prescale

`PS<2:0>` ← **`010`** for 1 / 8 division ⇒ 125 counts

2 cycle delay in synchronizer ⇒ 123 counts

Preset

Timer0 interrupts when **`FFh`** = 256 rolls over to 0

Preset counter to 256 – 123 = 133

N ms delay

`AUX ← N` for **`N × 1 ms delay`**

Delay Timer with Timer0 — 2

```
                List p = 16F873
                include "P16F873.INC"
                AUX equ 0x20            ; Auxiliary variable
InitTimer0:     bcf INTCON, T0IE        ; Disable Timer0 interrupt
                bsf STATUS, RP0         ; Bank 1
                movlw 0xC2              ; Configure timer mode
                movwf OPTION_REG        ; Prescaler = 8
                bcf STATUS, RP0         ; Bank 0
                clrf TMR0               ; TMR0 ← 0
                bcf INTCON, T0IF        ; Clear overflow flag
                return
Del1ms:         movlw .133              ; Preset value = 133 (decimal)
                movwf TMR0              ; TMR0 ← preset
Del1ms_01:      btfss INTCON, T0IF      ; Skip next if (T0IF == 1)
                goto Del1ms_01          ; Keep waiting
                bcf INTCON, T0IF        ; Clear T0IF = 0
                return                  ; Return after 1 ms
DelNms:         movwf AUX               ; AUX ← number of ms
DelNms_01:      ; Call Del1ms AUX times
                call Del1ms             ; Wait 1 ms.
                decfsz AUX, f           ; AUX--  Skip next if (AUX == 0)
                goto DemNms_01          ; Keep waiting
                return                  ; Return after AUX interations
                end
```

Measure Interval Between External Pulses — 1

Internal RC oscillator

$T_{CY} = 4 \times 1 / (4\ MHz) = 1\ \mu s = 0.001\ ms$

Timer1

Synchronous timer mode

Prescale ← 1

`TMR1++` every microsecond

CPP1 in capture mode

Capture values of Timer1

`CCP1CON` ← **`00000101`** (capture mode on rising edge)

Trigger at 2 external pulses

`CCP1IF` ← **`1`** on rising edge

Capture2 – Capture1 = interval (in microseconds)

Measure Interval Between External Pulses — 2

```
                List p = 16F873
                include "P16F873.INC"
                N1H equ 20h             ; High byte of first capture
                N1L equ 21h             ; Low byte of first capture
                NH equ 22h              ; High byte of difference
                NL equ 23h              ; Low byte of difference
Init_capture:   clrf T1CON              ; Timer mode with prescaler = 1
                clrf CCP1CON            ; Reset module CCP1
                bsf STATUS, RP0         ; Bank 1
                bsf TRISC, 2            ; Set CCP1 pin as input
                bcf PIE1, TMR1IE        ; Disable Timer1 interrupt
                bcf PIE1, CCP1IE        ; Disable CCP1 interrupt
                bcf STATUS, RP0         ; Bank 0
                clrf PIR1               ; Clear interrupt flags
                movlw 0x05              ; Capture mode on raising edge
                movwf CCP1CON           ;
                bsf T1CON, TMR1ON       ; Start Timer1
                return
```

Measure Interval Between External Pulses — 3

```
Capture:    bcf PIR1, CCP1IF      ; Clear capture flag
            btfss PIR1, CCP1IF    ; Skip next if (CCP1IF == 1)
            goto Capture
            bcf PIR1, CCP1I       ; Clear capture indicator
            movf CCPR1L, W        ; Store captured value in N1H and N1L
            movwf N1L
            movf CCPR1H, W
            movwf N1H
Capture2:   btfss PIR1, CCP1IF    ; Skip next if (CCP1IF == 1)
            goto Capture2
            bcf PIR1, CCP1IF      ; Clear capture indicator
            movf N1L, W
            subwf CCPR1L, W       ; Subtract captured values
            movwf NL
            btfss STATUS, C
            goto Subt1
            goto Subt0
Subt1:      decf CCPR1H, f
Subt0:      movf N1H, W
            subwf CCPR1H, W
            movwf NH
            return
            end
```

16-bit arithmetic

$$A_H:A_L = B_H:B_L - C_H:C_L$$

$$A_L \leftarrow B_L - C_L$$

$$\text{if } (C == 1)\ B_H\text{--}$$

$$A_H \leftarrow B_H - C_H$$

Real Time Clock (RTC) — 1

Internal RC oscillator

$T_{CY} = 4 \times 1 / (4\ \text{MHz}) = 1\ \mu\text{s} = 0.001\ \text{ms}$

Timer0

Timer0 interrupts when **`FFh`** = 256 rolls over to 0

Prescale = 32

Interrupt every 0.001 ms × 256 × 32 = 8.192 ms

Seconds

1 second per clock tick

(1 second / tick) / (8.192 ms / interrupt) = 122.07 interrupts / tick

1 second = 122 interrupts

Minutes

1 minute = 60 clock ticks

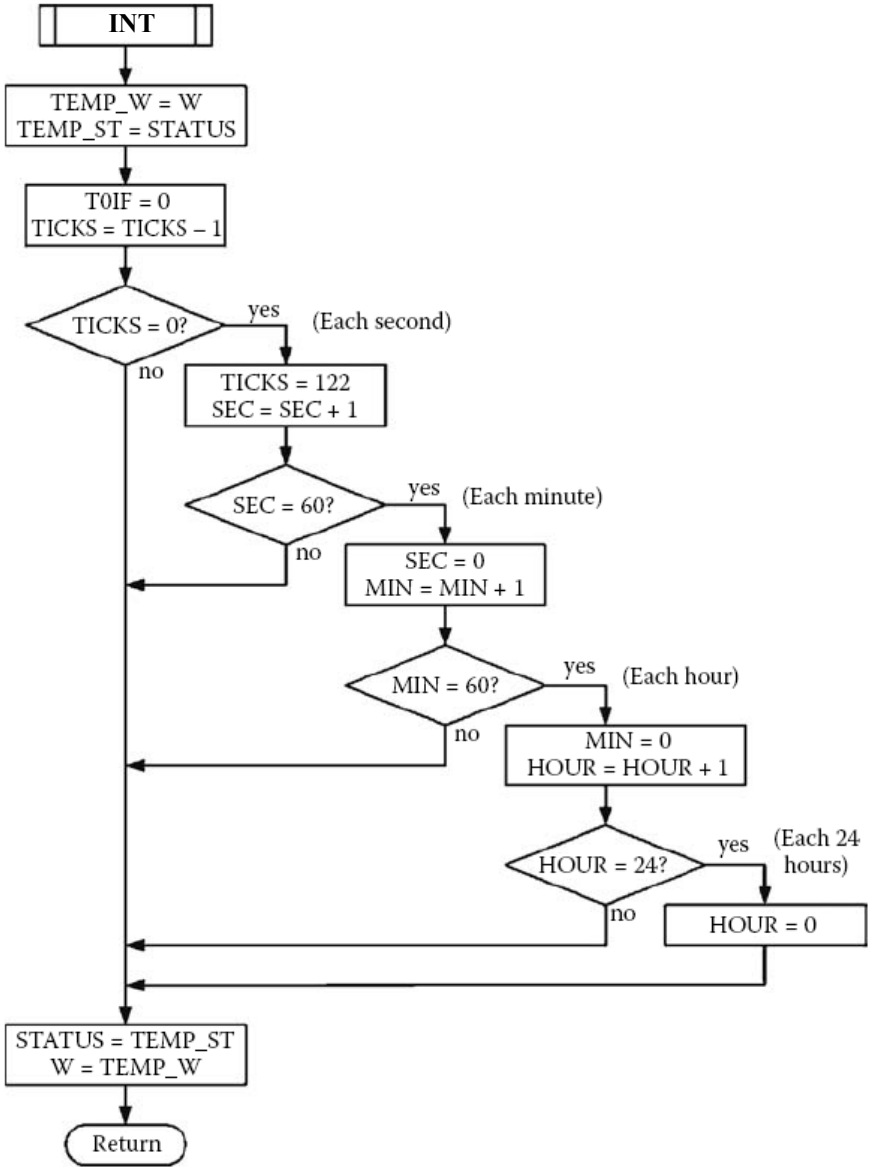
Hours

1 hour = 60 minutes

Days

1 day = 24 hours

Real Time Clock (RTC) — 2



Real Time Clock (RTC) — 3

```
            list p = 16f873
            #include <p16f873.inc>
            TICKS equ 0x20          ; Ticks counter
            SEC equ 0x21            ; Seconds counter
            MIN equ 0x22            ; Minutes counter
            HOUR equ 0x23           ; Hours counter
            TEMP_W equ 0x24
            TEMP_ST equ 0x25
            org 0
            goto init
            org 4
            goto rtc
init:       clrf INTCON             ; Disable interrupts
            bsf STATUS, RP0         ; Bank 1
            movlw 0xC4              ; Prescaler 32
            movwf OPTION_REG        ; Assigned to Timer0
            bcf STATUS, RP0         ; Bank 0
            movlw 0                 ; Count module = 256
            movwf TMR0              ; in Timer0
            movlw .122              ; Ticks per second
            movwf TICKS             ; in tick counter
            clrf SEC                ; Clear Seconds counter
            clrf MIN                ; Clear Minutes counter
            clrf HOUR               ; Clear Hour counter
            bsf INTCON, T0IE        ; Enable Timer0 interrupt
            bsf INTCON, GIE         ; Enable global interrupts
```

Exercício (Moodle)

```
prog:       nop
            goto prog           ; Infinite loop
rtc:        bcf STATUS, RP0     ; Bank 0
            PUSH_MACRO          ; Save STATUS, TEMP_ST
            bcf INTCON, T0IF    ; Clear overflow flag for Timer0
            decfsz TICKS, f     ; TICKS-- Skip next if (TICKS == 0)
            goto end_rtc
rtc_sec:    movlw .122          ; Re-init TICKS
            movwf TICKS
            incf SEC, f         ; seconds++
            movf SEC, W
            xorlw .60           ; Z ← 1 if (SEC == 60)
            btfsc STATUS, Z     ; Skip next on (Z == 1)
            goto end_rtc
rtc_min:    clrf SEC            ; Clear seconds
            incf MIN, f         ; minutes++
            movf MIN, W
            xorlw . 60          ; Z ← 1 if (MIN == 60)
            btfsc STATUS, Z     ; Skip next on (Z == 1)
            goto end_rtc
rtc_hour:   clrf MIN            ; Clear minutes
            incf HOUR, f        ; hours++
            movf HOUR, W
            xorlw .24           ; Z ← 1 if (HOUR == 60)
            btfsc STATUS, Z     ; Skip next on (Z == 1)
            goto end_rtc
rtc_day:    clrf HOUR           ; Clear hours
end_rtc:    POP_MACRO           ; Retrieve STATUS, TEMP_ST
            retfie              ; Return to interrupted program.
            end                 ; End of source code.
```

Exercício para entrega (Moodle): Real Time Clock (RTC)

Implementar um relógio que conte os segundos (0 até 60) utilizando a Placa MCLAb1 no simulador PICSimLab. Os segundos devem ser exibidos nos displays de 7 segmentos da placa.

A placa deve ser programadas em assembly do PIC e o código funciional deve ser enviado com o nome relogio.asm.

Utilizar os exemplos do exercício sobre display de 7 segmentos e o exemplo do RTC como base.