PIC18
and
Embedded
C Programming

Mid-Range → PIC18 Migration

	Mid-Range	PIC18
Data width	8 bits	8 bits
Instruction word	**14 bits**	**16 bits = 2 addressed bytes**
Instructions	**35**	**83**
Instruction address width	**13 bits**	**21 bits**
Program memory	256 – 8192 words	2 Kwords – 64 Kwords
ROM	Flash	Flash
Data memory (bytes)	56 – 368	256 – 4K
Interrupts	int / ext	int / ext
Pins	6 – 64	18 – 100
I/O pins	4 – 54	16 – 70
Stack	**8 levels**	**31 levels**
Timers	**2 – 3**	**2 – 5**
Bulk price	$0.35 – $2.50	$1.20 - $8.50

Migration to PIC18
Data memory / registers

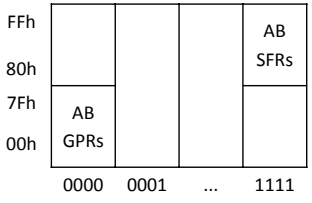
Data address space

12 bit address ⇒ memory ≤ 2^{12} = 4096 bytes = 4 KB

Memory partitioned into banks

Bank = 2^8 = 256 = **`100h`** registers (1/4 KB)

8 bit file address = **`00h … FFh`**



Banks in address space

≤ 2^{12-8} = 16 banks **`0 ... Fh`**

2 to 16 banks implemented in device

Bank select

Bank Select Register (**BSR**)

SFR at address **`FE0h`**

Access Bank (AB)

Virtual bank = addresses **`000h – 07Fh, F80h – FFFh`**

Access bit instructions (bit a = 1 in code) → **`AB 00h – FFh`**

Ignore **BSR**

Migration to PIC18
Special Function Registers (SFR) — all in bank 15 (Fh)

Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2[1]	FBFh	CCPR1H	F9Fh	IPR1
FFEh	TOSH	FDEh	POSTINC2[1]	FBEh	CCPR1L	F9Eh	PIR1
FFDh	TOSL	FDDh	POSTDEC2[1]	FBDh	CCP1CON	F9Dh	PIE1
FFCh	STKPTR	FDCh	PREINC2[1]	FBCh	CCPR2H	F9Ch	—[2]
FFBh	PCLATU	FDBh	PLUSW2[1]	FBBh	CCPR2L	F9Bh	OSCTUNE
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	—[2]
FF9h	PCL	FD9h	FSR2L	FB9h	—[2]	F99h	—[2]
FF8h	TBLPTRU	FD8h	STATUS	FB8h	BAUDCON	F98h	—[2]
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	PWM1CON[3]	F97h	—[2]
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	ECCP1AS[3]	F96h	TRISE[3]
FF5h	TABLAT	FD5h	T0CON	FB5h	CVRCON	F95h	TRISD[3]
FF4h	PRODH	FD4h	—[2]	FB4h	CMCON	F94h	TRISC
FF3h	PRODL	FD3h	OSCCON	FB3h	TMR3H	F93h	TRISB
FF2h	INTCON	FD2h	HLVDCON	FB2h	TMR3L	F92h	TRISA
FF1h	INTCON2	FD1h	WDTCON	FB1h	T3CON	F91h	—[2]
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	—[2]
FEFh	INDF0[1]	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—[2]
FEEh	POSTINC0[1]	FCEh	TMR1L	FAEh	RCREG	F8Eh	—[2]
FEDh	POSTDEC0[1]	FCDh	T1CON	FADh	TXREG	F8Dh	LATE[3]
FECh	PREINC0[1]	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD[3]
FEBh	PLUSW0[1]	FCBh	PR2	FABh	RCSTA	F8Bh	LATC
FEAh	FSR0H	FCAh	T2CON	FAAh	—[2]	F8Ah	LATB
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	—[2]
FE7h	INDF1[1]	FC7h	SSPSTAT	FA7h	EECON2[1]	F87h	—[2]
FE6h	POSTINC1[1]	FC6h	SSPCON1	FA6h	EECON1	F86h	—[2]
FE5h	POSTDEC1[1]	FC5h	SSPCON2	FA5h	—[2]	F85h	—[2]
FE4h	PREINC1[1]	FC4h	ADRESH	FA4h	—[2]	F84h	PORTE[3]
FE3h	PLUSW1[1]	FC3h	ADRESL	FA3h	—[2]	F83h	PORTD[3]
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA

1 — Not a physical register
2 — Unimplemented registers read 0
3 — Not present on 28-pin devices

Migration to PIC18
Status Register

	7	6	5	4	3	2	1	0
Name				**N**	**OV**	**Z**	**DC**	**C**
Writable	Unimplemented in PIC18			R/W	R/W	R/W	R/W	R/W
Reset value	0	0	0	1	1	x	x	x

<7:5>	—	Unimplemented (bank select on Mid-Range devices)
N	Negative	**N** ← **1** on negative ALU result **N** ← **0** on positive ALU result
OV	Overflow	**OV** ← **1** on overflow in signed arithmetic **OV** ← **0** = on no overflow
Z	Zero flag	**Z** ← **1** on ALU zero **Z** ← **0** on non-zero
DC	Half-byte carry (bits 3,4)	**DC** ← **1** on carry (Addition) **DC** ← **0** on borrow (Subtraction)
C	Carry out	**C** ← **1** on carry (Addition) **C** ← **0** on borrow (Subtraction)

Migration to PIC18
Indirect addressing

Indirect addressing

Program writes to File Select Registers (**FSRs**)

Instructions can increment/decrement **FSR** values

INDF virtual registers track contents of **FSR** registers

Migration

Mid-range **FSR** → PIC18 pairs **FSRnH:FSRnL , n = 0, 1, 2**

FSRn<11:0> = full 12-bit address

FSRn<15:12> not used

Mid-range **INDF** → PIC18 **INDFn , n = 0, 1, 2**

Example

```
[005] = 10h
[006] = 0Ah
Load FSR2 ← 005 ⇒ [INDF2] = 10h
FSR2++ ⇒ [INDF2] = 0Ah
```


Migration to PIC18
Indirect addressing with automatic increment/decrement

Similar to INDFn

```
[005] = 10h
[006] = 0Ah
Load FSRn ← 005h ⇒ [FSRn] = [INDFn] = 10h
```

POSTDEC

```
W ← POSTDECn ⇒ W ← 10h , FSRn ← 004h
```

POSTINC

```
W ← POSTINCn ⇒ W ← 10h , FSRn ← 006h
```

PREINC

```
W ← PREINCn ⇒ FSRn ← 006h , W ← 0Ah
```

PLUSW

```
W ← PLUSWn ⇒ FSRn ← 005h + W (signed)
W ← [new FSRn] = [005h + W]
```


Migration to PIC18
Program memory migration

Instruction width

16 bit instruction = 2 bytes

PC points to byte (not instruction)

PC ← **PC + 2** on non-branch instruction

Address partitioning

PC width = 21 bits

Mid-range **PCH:PCL** → PIC18 **PCU:PCH:PCL**

Mid-range **page:offset** description not used

21 bits = **PCU<20:16>:PCH<15:8>:PCL<7:0>**

PCU<23:21> not used

PC access

PCL direct R/W

PCH ← **PCLATH**

PCU ← **PCLATU**

	PC																							
	PCU								PCH								PCL							
	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

	4	3	2	1	0	7	6	5	4	3	2	1	0
	PCLATU					PCLATH							

Migration to PIC18

10 bit A/D converter on 13 inputs

13 inputs

AN0 , ... , AN12

Physical pin assignments device dependent

Each pin configurable as

- Digital I/O configured with TRISx
- Analog input

Not all input channels available on all devices

Registers

A/D Result High Register (**ADRESH**)

A/D Result Low Register (**ADRESL**)

A/D Control Register 0 (**ADCON0**)

A/D Control Register 1 (**ADCON1**)

A/D Control Register 2 (**ADCON2**)

Migration to PIC18

Register changes

Mid-Range		PIC18		
Register	Bit	Register	Bit	
OPTION_REG	NOT_RBPU	INTCON2	NOT_RBPU	New interrupt register
OPTION_REG	INTEDG	INTCON2	INTEDG0	
OPTION_REG	T0CS	T0CON	T0CS	New Timer0 register
OPTION_REG	T0SE	T0CON	T0SE	
OPTION_REG	PSA	T0CON	PSA	New WDT postscaler
OPTION_REG	PS2	T0CON	T0PS2	WDTPS2, CONFIG2H<3>
OPTION_REG	PS1	T0CON	T0PS1	WDTPS1, CONFIG2H<2>
OPTION_REG	PS0	T0CON	T0PS0	WDTPS0, CONFIG2H<1>
PCON	NOT_POR	RCON	NOT_POR	
PCON	NOT_BOR	RCON	NOT_BOR	
STATUS	NOT_TO	RCON	NOT_TO	
STATUS	NOT_PD	RCON	NOT_PD	
INDF	—	INDF0	—	
FSR	—	FSR0L	—	
TMR0	—	TMR0L	—	
SSPCON	—	SSPCON1	—	
ADRES	—	ADRESH	—	

Migration to PIC18

Example code changes

Mid-Range Code	PIC18 Code	
bcf STATUS,RP0	clrf BSR	; first occurrence only
bcf STATUS,RP1		; otherwise remove
movf INDF,w	movf INDF0,w	
movf TMR0,w	movf TMR0L,w	
movf FSR,w	movf FSR0L,w	
movf SSPCON,w	movf SSPCON1,w	
movf ADRES,w	movf ADRESH,w	
movf OPTION_REG,w	movf T0CON,w	; TIMER0 operations
movf PCON,w	movf RCON,w	

Interrupt vector

Mid-range address = **0x04** → PIC18 address **0x08** (byte addressing)

Migration to PIC18

Differences in status bit operation

Instruction	STATUS Bits		
	Mid-Range	PIC18	
ADDLW	C, DC, Z	C, DC, Z, OV, N	
ADDWF	C, DC, Z	C, DC, Z, OV, N	
ANDLW	Z	Z, N	
ANDWF	Z	Z, N	
COMF	Z	Z, N	
DECF	Z	C, DC, Z, OV, N	
INCF	Z	C, DC, Z, OV, N	
IORLW	Z	Z, N	
IORWF	Z	Z, N	
MOVF	Z	Z, N	
RETFIE	GIE	GIE/GIEH, PEIE/GIEL	
RLF	C, DC, Z	C, Z, N	New
RRF	C, DC, Z	C, Z, N	New
SUBLW	C, DC, Z	C, DC, Z, OV, N	
SUBWF	C, DC, Z	C, DC, Z, OV, N	
XORLW	Z	Z, N	
XORWF	Z	Z, N	

Migration to PIC18

New instructions — 1

ADDWFC	f, d, a (access)	Add WREG and Carry bit to f
BTG	f, d, a	Bit Toggle f
CPFSEQ	f, a	Compare f with WREG, Skip =
CPFSGT	f, a	Compare f with WREG, Skip >
CPFSLT	f, a	Compare f with WREG, Skip <
DAW	–	Decimal Adjust WREG
LFSR	f, k	Literal k to FSR(f)
MOVF	f, d, a	Move f
MOVFF	fs, fd	Move fs to fd
MOVLB	k	Move Literal to BSR<3:0>
MULLW	k	Multiply k with WREG → PRODH:PRODL
MULWF	f, a	Multiply WREG with f → PRODH:PRODL
NEGF	f, a	Negate f
POP	–	Pop (delete) Top of Return Stack (TOS)
PUSH	–	Push (PC+2) to Top of Return Stack (TOS)
RCALL	n	Relative Call
RESET		Software Device Reset
RLNCF	f, d, a	Rotate Left f (No Carry)
RRNCF	f, d, a	Rotate Right f (No Carry)
SETF	f, a	Set f
SUBFWB	f, d, a	Subtract f from WREG with Borrow
SUBWFB	f, d, a	Subtract WREG from f with Borrow
TBLRD		Read byte from program memory table
TBLWT		Write byte to program memory table

Migration to PIC18

New instructions — 2

BZ	n	Branch if Zero
BC	n	Branch if Carry
BN	n	Branch if Negative
BNC	n	Branch if Not Carry
BNN	n	Branch if Not Negative
BNOV	n	Branch if Not Overflow
BNZ	n	Branch if Not Zero
BOV	n	Branch if Overflow
BRA	n	Branch Unconditionally
DCFSNZ	f, d, a	Decrement f, Skip if Not 0
INFSNZ	f, d, a	Increment f, Skip if Not 0
TSTFSZ		Skip next on file == 0

CALL k, s	PUSH return address If s == 1 WS ← W , STATUSS ← STATUS , BSRS ← BSR PC ← k
RETFIE s	If s == 1 W ← WS , STATUS ← STATUSS , BSR ← BSRS POP return address

C Programming for PIC18

High Level Language

Why not assembly language

- Laborious and tedious
- Difficult to manage complex programs
  - Program flow
  - Mathematical tasks
  - Debugging
- Required for critical path
  - Very fast code
  - Exact timing — code timed in clock cycles

Why C

- Standard and portable
- Designed for simple processors (1970s machines)
- "Close" to hardware
- Straightforward compilation
  - Human-readable disassembly
  - Hands-on optimization

Adapting C to PIC

Standard compiler strategies

- Similar for all Instruction Set Architectures
- Recursive procedures
- Variable allocation
- Mathematical functions

PIC-specific strategies

- Module programming
  - Timers, A/D, comparators, ports, ...
  - Controlled with Special Function Registers (SFRs)
- Device header file
  - Define SFRs as reserved words
  - Program reads / writes SFRs
- I/O
  - Read / write ports as SFRs
  - Standard output → USART

Example 1

```
#include <p18f242.h>              // 18F242 header file
                                  // predefines TRISB, PORTB

unsigned char counter;

void main (void)
{
     TRISB = 0;                   // set PORTB = outputs
     counter = 1;

     while (1)
     {
       PORTB = counter;           // PORTB ← counter value
/* compiled code copies value of variable counter to
     variable PORTB, stored at address of SFR PORTB */
       counter = counter + 1;
     }
}
```


MPLAB C18 Compiler

C compiler for PIC18

- Specific to PIC18 ISA

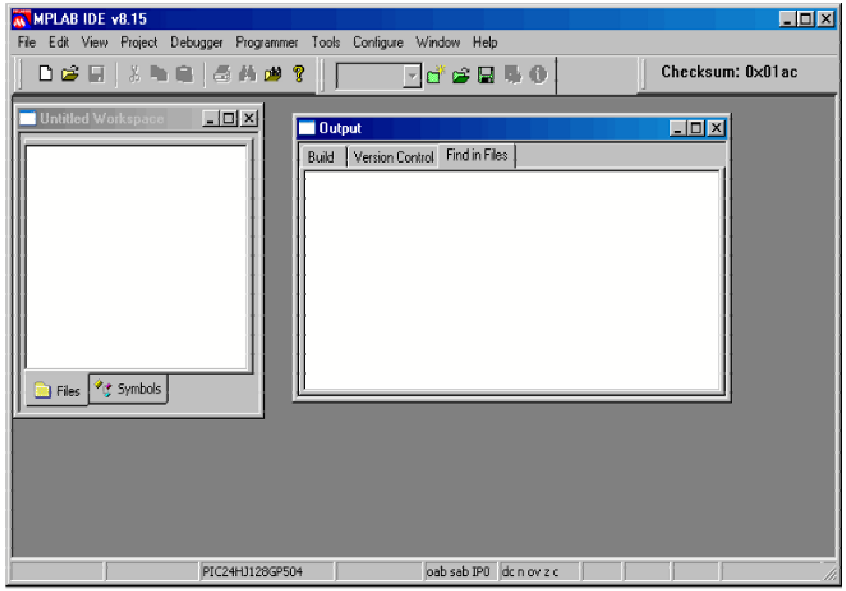
Commercial software

- $150 from Microchip
- Academic version
  - Learning tool
  - Time limited

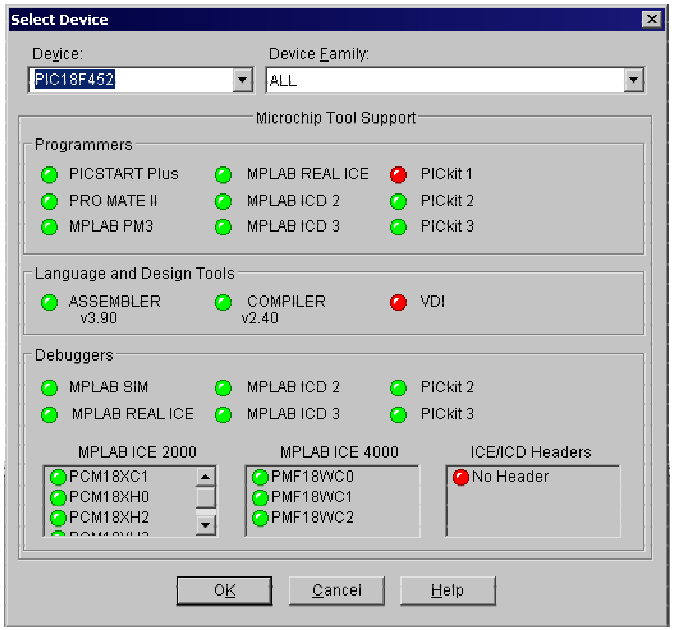
Integrates nicely with PLAB

- Coding in PLAB IDE
- Integrated debugging
  - View disassembly
    - Executable machine code displayed as assembly code
  - C-level breakpoints
  - Assembly-level breakpoints
  - Assembly-level trace (single-step execution)

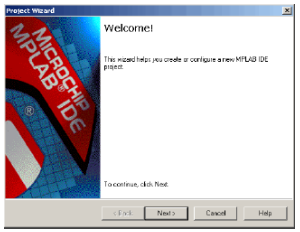
Start MPLAB IDE



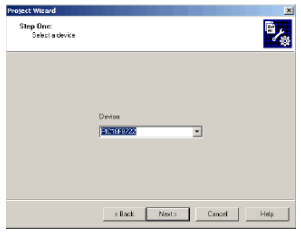
Configure > Select Device



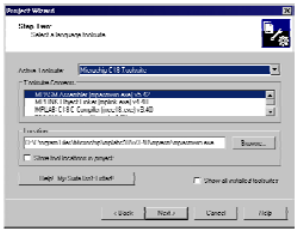
Project > Project Wizard



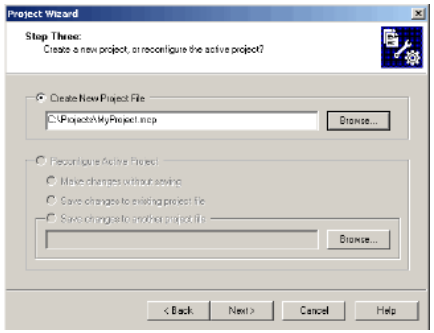
Wizard



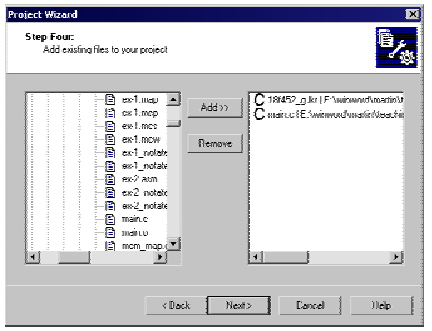
Select PIC Device



Select C18 Toolsuite



Save Project by Pathname



Add linker Template and C File

Build Project

Either

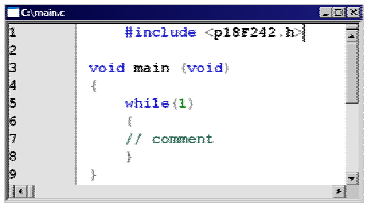
Project > Build All

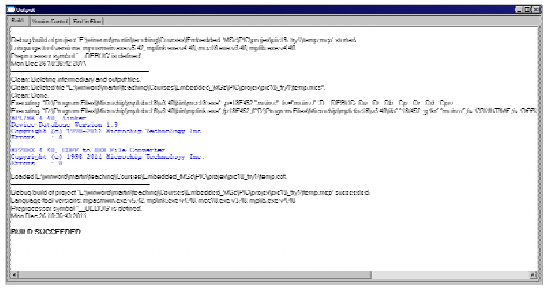
Right click on project name in Project Window > Build All

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

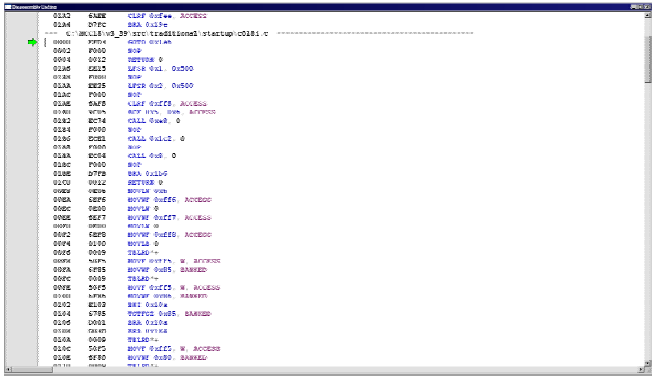
Debugger > Select Tool > MPLAB SIM

Debug toolbar opens

Debugger > Step Into → Debugger > Reset > Processor Reset

Assembly code editor opens (disassembly)

Green arrow points to program start (main)



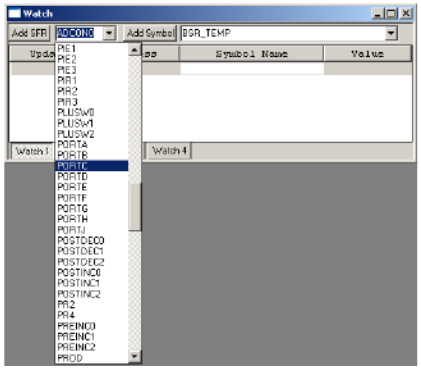
Step Into

Run program in trace mode (single step)

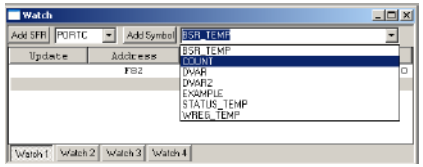
View > Watch

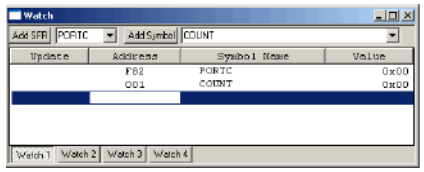
Choose + Add items to watch list

SFRs



Symbols





Breakpoints

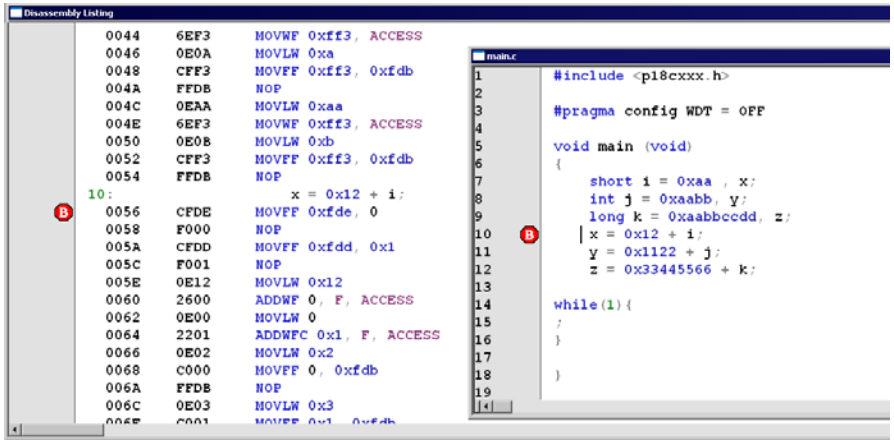
Set breakpoint

Double-click on line of code

Right click > choose Set Breakpoint from menu

Run

Program stops before breakpoint



More Details

Radix

```
TRISA = 0b'10000110';    // initialise PORTA
TRISB = 0x86;            // initialise PORTB
TRISC = 134;             // initialise PORTC
                         // 10000110₂ = 86₁₆ = 134₁₀
```

Linker scripts

Device-specific definitions for linker

In directory ...\bin\LKR

Memory Models

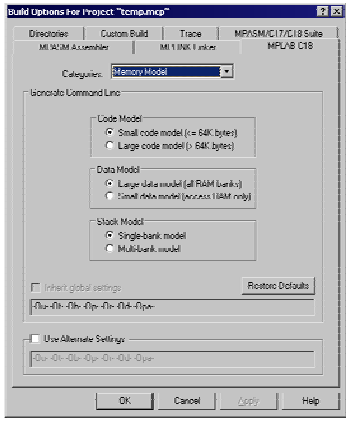
Project>Build Options>Project

Code

- Small ≤ 64 KB
- Large > 64 KB

Data

- Large — all register banks
- Small — ACCESS bank only



Example 2

C code

```
#include <p18cxxx.h>                  // PIC18 header file

#pragma config WDT = OFF

void main (void)
{
     short i = 0xaa , x;              // Short = 8 bits
     int j = 0xaabb, y;               // Standard integer = 16 bits
     long k = 0xaabbccdd, z;          // Long = 32 bits
    x = 0x12 + i;
    y = 0x1122 + j;
    z = 0x33445566 + k;

while(1){
;
}

}
```

Example 2

Disassembly — 1

```
---  C:\MCC18\v3_39\src\traditional\proc\p18F452.asm  ----------------------------------------
  0186    0E00     MOVLW 0
:
---  C:\MCC18\v3_39\src\traditional\startup\c018i.c  -----------------------------------------
  0000    EFD3     GOTO 0x1a6
:
01BA    EC04     CALL 0x8, 0
:
---  C:\MCC18\v3_39\src\traditional\stdclib\__init.c  -------
  01C2    0012     RETURN 0
--- C:\MCC18\pic18_ex1\main.c  ---------------
  0008    CFD9     MOVFF 0xfd9, 0xfe6
  000A    FFE6     NOP
  000C    CFE1     MOVFF 0xfe1, 0xfd9
  000E    FFD9     NOP
  0010    0E10     MOVLW 0x10
  0012    26E1     ADDWF 0xfe1, F, ACCESS
  0014    0EAA     MOVLW 0xaa       ; i = 0xaa
  0016    6EDE     MOVWF 0xfde, ACCESS
  0018    6ADD     CLRF 0xfdd, ACCESS
  001A    0EBB     MOVLW 0xbb       ; j = 0xaabb
  001C    6EF3     MOVWF 0xff3, ACCESS
  001E    0E04     MOVLW 0x4
  0020    CFF3     MOVFF 0xff3, 0xfdb
  0022    FFDB     NOP
  0024    0EAA     MOVLW 0xaa
  0026    6EF3     MOVWF 0xff3, ACCESS
  0028    0E05     MOVLW 0x5
  002A    CFF3     MOVFF 0xff3, 0xfdb
 :
```

FSR1L = FSR2L + 15	z_3
FSR2L + 14	z_2
FSR2L + 13	z_1
FSR2L + 12	z_0
FSR2L + 11	k_3
FSR2L + 10	k_2
FSR2L + 9	k_1
FSR2L + 8	k_0
FSR2L + 7	y_1
FSR2L + 6	y_0
FSR2L + 5	j_1
FSR2L + 4	j_0
FSR2L + 3	0
FSR2L + 2	x
FSR2L + 1	0
FSR2L = 0xfd9	i

Example 2

Disassembly — 2

```
--- C:\MCC18\pic18_ex1\main.c  ---------------
1:                  #include <p18cxxx.h>
2:
3:                  #pragma config WDT = OFF
4:
5:                  void main (void)
  0008    CFD9      MOVFF FSR2L, POSTINC1
                            ; [FSR2L] -> [FSR1] , FSR1++
  000A    FFE6      NOP
  000C    CFE1      MOVFF FSR1L, FSR2L
                            ; [FSR1L] -> [FSR2L]
  000E    FFD9      NOP
  0010    0E10      MOVLW 0x10      ; 16 = 2*2 + 2*2 + 2*4
                                    ; bytes allocated to data
  0012    26E1      ADDWF FSR1L, F, ACCESS
                            ; [FSR1L] -> [FSR1L] + 16
```


Example 2

Disassembly — 3

```
6:   {
7:      short i = 0xaa , x;
  0014    0EAA      MOVLW 0xaa
  0016    6EDE      MOVWF POSTINC2, ACCESS
                                  ; AAh -> [FSR2L] = i, FSR2L++
  0018    6ADD      CLRF POSTDEC2, ACCESS          ; 0 -> FSR2L--
8:      int j = 0xaabb, y;
  001A    0EBB      MOVLW 0xbb
  001C    6EF3      MOVWF PRODL, ACCESS            ; BBh -> PRODL
  001E    0E04      MOVLW 0x4
  0020    CFF3      MOVFF PRODL, PLUSW2
                            ; PRODL = BBh -> [FSR2L + 4] = j_0
  0022    FFDB      NOP
  0024    0EAA      MOVLW 0xaa
  0026    6EF3      MOVWF PRODL, ACCESS            ; AAh -> PRODL
  0028    0E05      MOVLW 0x5
  002A    CFF3      MOVFF PRODL, PLUSW2
                            ; PRODL = AAh -> [FSR2L + 5] = j_1
  002C    FFDB      NOP
```


Example 2

Disassembly — 4

```
9:    long k = 0xaabbccdd, z;
  002E    0EDD      MOVLW 0xdd
  0030    6EF3      MOVWF PRODL, ACCESS      ; DDh -> PRODL
  0032    0E08      MOVLW 0x8
  0034    CFF3      MOVFF PRODL, PLUSW2      ; PRODL -> [FSR2L + 8] = k_0
  0036    FFDB      NOP
  0038    0ECC      MOVLW 0xcc
  003A    6EF3      MOVWF PRODL, ACCESS      ; CCh -> PRODL
  003C    0E09      MOVLW 0x9
  003E    CFF3      MOVFF PRODL, PLUSW2      ; PRODL -> [FSR2L + 9] = k_1
  0040    FFDB      NOP
  0042    0EBB      MOVLW 0xbb
  0044    6EF3      MOVWF PRODL, ACCESS      ; BBh -> PRODL
  0046    0E0A      MOVLW 0xa
  0048    CFF3      MOVFF PRODL, PLUSW2      ; PRODL -> [FSR2L + 10] = k_2
  004A    FFDB      NOP
  004C    0EAA      MOVLW 0xaa
  004E    6EF3      MOVWF PRODL, ACCESS      ; AAh -> PRODL
  0050    0E0B      MOVLW 0xb
  0052    CFF3      MOVFF PRODL, PLUSW2      ; PRODL -> [FSR2L + 11] = k_3
  0054    FFDB      NOP
```

Example 2
Disassembly — 5

```
10:                     x = 0x12 + i;
  0056    CFDE      MOVFF POSTINC2, 0
                            ; [FSR2] = i_0 -> [0] , FSR2++
  0058    F000      NOP
  005A    CFDD      MOVFF POSTDEC2, 0x1
                            ; [FSR2] = i_1 -> [1] , FSR2--
  005C    F001      NOP
  005E    0E12      MOVLW 0x12
  0060    2600      ADDWF 0, F, ACCESS  ; [0] + 0x12 -> [0]
  0062    0E00      MOVLW 0
  0064    2201      ADDWFC 0x1, F, ACCESS
                                  ; [1] + 0x00 + C -> [1]
  0066    0E02      MOVLW 0x2
  0068    C000      MOVFF 0, PLUSW2     ; [0] -> [FSR2 + 2] = x_0
  006A    FFDB      NOP
  006C    0E03      MOVLW 0x3
  006E    C001      MOVFF 0x1, PLUSW2   ; [1] -> [FSR2 + 3] = x_1
  0070    FFDB      NOP
```


Example 2
Disassembly — 6

```
11:                     y = 0x1122 + j;
  0072    0E04      MOVLW 0x4
  0074    CFDB      MOVFF PLUSW2, 0    ; [FSR2 + 4] = j_0 -> [0]
  0076    F000      NOP
  0078    0E05      MOVLW 0x5
  007A    CFDB      MOVFF PLUSW2, 0x1  ; [FSR2 + 5] = j_1 -> [1]
  007C    F001      NOP
  007E    0E22      MOVLW 0x22
  0080    2600      ADDWF 0, F, ACCESS ; [0] + 0x22 -> [0]
  0082    0E11      MOVLW 0x11
  0084    2201      ADDWFC 0x1, F, ACCESS
                                  ; [1] + 0x11 + C -> [1]
  0086    0E06      MOVLW 0x6
  0088    C000      MOVFF 0, PLUSW2    ; [0] -> [FSR2 + 6] = y_0
  008A    FFDB      NOP
  008C    0E07      MOVLW 0x7
  008E    C001      MOVFF 0x1, PLUSW2  ; [1] -> [FSR2 + 7] = y_1
  0090    FFDB      NOP
```


Example 2
Disassembly — 7

```
12:                 z = 0x33445566 + k;
  0092    0E08      MOVLW 0x8
  0094    CFDB      MOVFF PLUSW2, 0                 [0] = k_0 -> [FSR2 + 8]
  0098    0E09      MOVLW 0x9
  009A    CFDB      MOVFF PLUSW2, 0x1               [1] = k_1 -> [FSR2 + 9]
  009E    0E0A      MOVLW 0xa
  00A0    CFDB      MOVFF PLUSW2, 0x2               [2] = k_2 -> [FSR2 + 10]
  00A4    0E0B      MOVLW 0xb
  00A6    CFDB      MOVFF PLUSW2, 0x3               [3] = k_3 -> [FSR2 + 11]
  00AA    0E66      MOVLW 0x66
  00AC    2600      ADDWF 0, F, ACCESS              [0] + 0x66 -> [0]
  00AE    0E55      MOVLW 0x55
  00B0    2201      ADDWFC 0x1, F, ACCESS           [1] + 0x55 + C -> [1]
  00B2    0E44      MOVLW 0x44
  00B4    2202      ADDWFC 0x2, F, ACCESS           [2] + 0x44 + C -> [2]
  00B6    0E33      MOVLW 0x33
  00B8    2203      ADDWFC 0x3, F, ACCESS           [3] + 0x33 + C -> [3]
  00BA    0E0C      MOVLW 0xc
  00BC    C000      MOVFF 0, PLUSW2                 [0] -> [FSR2 + 12] = z_0
  00C0    0E0D      MOVLW 0xd
  00C2    C001      MOVFF 0x1, PLUSW2               [1] -> [FSR2 + 13] = z_1
  00C6    0E0E      MOVLW 0xe
  00C8    C002      MOVFF 0x2, PLUSW2               [0] -> [FSR2 + 14] = z_0
  00CC    0E0F      MOVLW 0xf
  00CE    C003      MOVFF 0x3, PLUSW2               [1] -> [FSR2 + 15] = z_1
13:
14:                 while(1){
  00D2    D7FF      BRA 0xd2
15:                 ;
16:                 }
```


Example 3

```
#include <stdio.h>
#pragma config WDT = OFF
void main (void)
{
printf ("Hello, world!\n");
while (1)
;
}
```

Library function `printf`
Character output to
- Hardware USART
- User defined function

SIM UART1
- PLAB function for output capture

Stimulus

Simulating events

Level change or pulse to I/O port pin

Value in SFR or GFR data memory

Stimulus dialog

Table of events and triggers

Asynchronous

One time change to I/O pin or register

Triggered by user

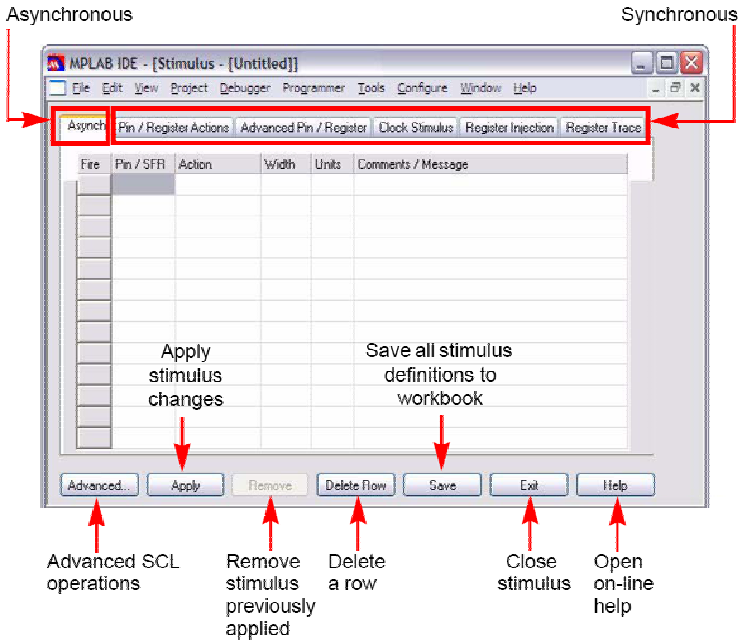
"Fire" button on stimulus GUI within MPLAB IDE

Synchronous

Automatic triggering

Predefined signal / data changes to I/O, SFR or GPR

Stimulus Dialog



C18 Software Library Functions

ANSI 1989 standard C libraries

Character classification, data conversion, string / memory

Character output library

`fprint`, **`fprintf`**, **`putc`**, ...

Device software libraries

UART-type I/O

`ReadUART`, **`WriteUART`**, **`getcUART`**, **`putcUART`**, ...

External device drivers

LCD, serial communications, ...

Delay functions

`Delay1TCY`	**`Delay one instruction cycle`**
`Delay10TCYx`	**`Delay multiples of 10 instruction cycles`**
`Delay100TCYx`	**`Delay multiples of 100 instruction cycles`**
`Delay1KTCYx`	**`Delay multiples of 1,000 instruction cycles`**
`Delay10KTCYx`	**`Delay multiples of 10,000 instruction cycles`**

Reset

Report cause of previous reset

Example 4 — 1

Inputs (switches)
RB4:RB5 on portB
RB7 on portC
Outputs (LED)
RC6:RC0 on portC

```
      #include <p18F242.h>
      #include <delays.h>
      void initialize (void);
      void diagnostic (void);
void main (void)
{
      initialize();            // initialize ports
      diagnostic();            // turn on LEDs for 1 sec
loop:
      if (PORTBbits.RB4 == 0) // copy switch state on RB4 to RC6
        PORTCbits.RC6 = 0;
      else PORTCbits.RC6 = 1;

      if (PORTBbits.RB5 == 0) // copy switch state on RB5 to RC5
        PORTCbits.RC5 = 0;
      else PORTCbits.RC5 = 1;

      goto loop;
}
```

Example 4 — 2

```
void initialize (void)
{
     TRISA = 0b00000000;       // set portA as outputs
     TRISB = 0b00110000;       // set RB5:RB4 as inputs
                               // rest of portC as outputs
     TRISC = 0b10000000;       // set RC7 as input
                               // rest of portC as outputs
     PORTA = 0;                // reset portA ... portC
     PORTB = 0;
     PORTC = 0;
}

void diagnostic (void)
{
     PORTCbits.RC6 = 1;        // turn on RC6
     PORTCbits.RC5 = 1;        // turn on RC5
     Delay10KTCYx(100);        // delay 100 × 10 × 1000 TCY = 10^6 TCY
     PORTCbits.RC6 = 0;        // turn off RC6
     PORTCbits.RC5 = 0;        // turn off RC5
     Delay10KTCYx(100);        // delay 10^6 TCY
}
```


C18 Hardware Library Functions

Peripheral module libraries

I/O port B

Configure interrupts, pull-up resistors, on PORTB pins

A/D converter

Configure, start + check + read conversion, close device

CPP module

Configure, start, read, stop input capture

Configure, start, stop PWM output

Timers

Configure, start, read, write, stop timer

USART

Configure, start, check, read, write, stop USART

Serial communications

Proprietary chip-to-chip data communication

I^2C, Microwire, SPI

TimerX, X = 0, 1, 2, 3, 4

Timer library functions

`unsigned int ReadTimerX( void );`

Read unsigned integer from TimerX

`void WriteTimerX( unsigned int );`

Write unsigned integer to TimerX

`void CloseTimer0( void );`

Close TimerX

`void OpenTimerX( unsigned char )`

Configure TimerX with unsigned character configuration bit mask

mask = binary value → SFRs **`TMRXH:TMRXH`**

Constructed from reserved words

Example

```
mask = TIMER_INT_OFF & T2_PS_1_4 & T2_POST_1_2
                     & T0_EDGE_RISE
```

Sets interrupts disabled, prescaler 1:4, postscaler 1:2, rising edge trigger

Motor Control with One PWM

Connections

Motor enable

Output pin RA5

Motor speed control

PWM **CCP1** output pin (device-dependent — usually **RC2** on PIC18)

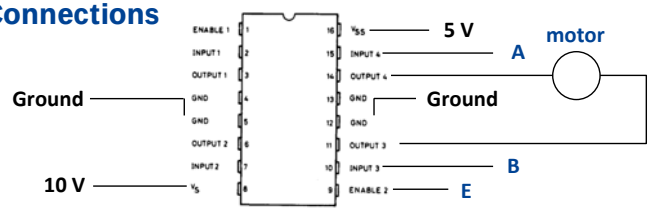
Control A = **CCP1**

Control B = **NOT(CPP1)**

PWM duty cycle > 50% ⇒ $A_{average} > B_{average}$ ⇒ forward motion

PWM duty cycle < 50% ⇒ $A_{average} < B_{average}$ ⇒ reverse motion

Connections



E	A	B	Motor
1	0	0	Brake
1	1	1	
1	1	0	Full Forward
1	0	1	Full Reverse
0	x	x	Coast

Motor Control with One PWM

PWM parameters

Internal oscillator

f_{OSC} = 4 MHz ⇒ T_{OSC} = 0.25 µs

Require PWM frequency > human hearing threshold

f_{PWM} = 25 kHz ⇒ T_{PWM} = 40 µs

T_{PWM} = 40 µs = 4 × (PR2 + 1) × P × 0.25 µs = (PR2 + 1) × P × 1 µs

(PR2 + 1) × P = 40

Preset and PR2

P = 1 ⇒ PR2 + 1 = 40 ⇒ PR2 = 39 = 0x27

ΔT_{ON} = P × T_{OSC} = 0.25 µs ⇒ ΔT_{ON} / T_{PWM} = 0.25 µs / 40 µs = 0.625%

Breaking duty cycle = 50 %

T_{ON} = 0.50 × 40 µs = 20 µs ⇒ DC = 20 µs / 0.25 µs = 80 = 0x50

Maximum duty cycle

T_{ON} = 40 µs ⇒ DC = 40 µs / 0.25 µs = 160

Forward	**80 < DC < 160**
Reverse	**0 < DC < 80**

Motor Control with One PWM

Code

```
#include <timers.h>
#include <pwm.h>
     void motor(unsigned short);
void main (void)
{
     unsigned short speed = 0x50;
     TRISA = 0b00000000;     // portA = outputs, RA2 = motor enable
     TRISC = 0b00000000;     // portC = outputs, RC2 = CCP1 PWM
     PORTA = 0;              // disable motor
     PORTC = 0;              // PWM output = 0
     OpenTimer2(TIMER_INT_OFF & T2_PS_1_1 & T2_POST_1_1);
                             // Enable Timer2, set P = 1
     OpenPWM1(0x27);        // Enable PWM1, set PR2
/*  program sets speed and calls motor(speed) */
}
     void motor(unsigned int speed);
{
     SetDCPWMx(speed);         // set duty cycle
     PORTAbits.RA2 = 1;        // enable motor
}
```


A/D Converter

Functions

Function	Description
`void OpenADC( unsigned char, unsigned char );`	`Configure A/D converter`
`void SetChanADC( unsigned char );`	`Select channel`
`void ConvertADC( void );`	`Start conversion`
`char BusyADC( void );`	`Is converter busy?`
`int ReadADC( void );`	`Read conversion result`
`void CloseADC( void );`	`Disable converter`

Header file

```
#include adc.h
```


A/D Converter

Configuration words

First Word

Clock source (divide T_CY or use internal RC)

`ADC_FOSC_X , X = 2, 4, 8, 16, 32, 64, RC`

Result justification (Result in Least / Most Significant bits)

`ADC_X_JUST , X = RIGHT (LSB), LEFT (MSB)`

Voltage reference source

```
ADC_xANA_yREF ,  x = number of analog inputs
                 y = voltage reference
                      0 – VREF+ = VDD, VREF- = VSS
                      1 – VREF+ = AN3, VREF- = VSS
                      2 – VREF+ = AN3, VREF- = AN2
```

Second Word

Channel

`ADC_CHX , X = channel number`

A/D Interrupts

`ADC_INT_X , X = ON , OFF`

A/D Converter
Example

```
#include <p18F242.h>
#include <adc.h>
void main (void)
{
     int data;
     // Enable ADC – portA<3:0> = analog inputs, internal reference
     // right justify result, channel 0, no interrupt
     OpenADC(ADC_FOSC_8 & ADC_RIGHT_JUST & ADC_3ANA_0REF ,
               ADC_CH0 & ADC_INT_OFF);
     SetChanADC(ADC_CH0);      // convert channel 0
     Delay10TCYx(2);           // delay 20 us for acquisition
     ConvertADC();
     while (BusyADC());        // wait for conversion
     data = ReadADC()          // read data
     SetChanADC(ADC_CH3);      // switch channel
     Delay10KTCYx (2);         // delay 20 us for acquisition
     ConvertADC();
}
```


Inline Assembly

Insert assembly code into C program

- **_asm** begins assembly section
- **_endasm** ends assembly section
- Assembled by C18 compiler

Uses

- Processor-specific actions
  - **SLEEP**, **CLRWDT**, etc
- Critical path
  - Very fast / clock counting procedures

Special requirements

- No assembler directives — only PIC18 ISA
- Comments in C or C++ format
- Operands fully specified
- Default radix = decimal
- Literals specified in C notation
- Labels must end with colon

Inline Assembly Example

```
void main (void)
{
     while (1)
     {
        check_function_1();    // some C-oriented work
        check_function_2();
        _asm
            CLRWDT             // reset watchdog timer
        _endasm
     }
}
```


Specifying Code Sections

Code sections

- Overrides usual C procedure assignments
- Provide assembly-type section declarations for relocatable code

Pragmas

```
#pragma code (section name) (=address)
#pragma romdata (section name) (=address)
#pragma udata (section name) (=address)
#pragma idata (section name) (=address)
```

Examples

```
#pragma code _entry_scn=0x00  // defines reset routine
void _entry (void)            // at address 0
{
    _asm goto _startup _endasm
}
#pragma code _startup_scn     // defines startup routine
void _startup (void)          // linker chooses address
{ ...
```

Default: start at user `main()`

Configuration Bits

Static hardware configuration

Device dependent

Written to program memory during EEPROM programming

Not program accessible at run time

Pragma

```
#pragma config
```

Example

For typical PIC18

```
#pragma config OSC = HS, OSCS = OFF
// oscillator = HS, oscillator switch off
#pragma config PWRT = ON, BOR = OFF
// power-up timer on, brown-out detect off
#pragma config WDT = OFF
// watchdog timer is off
```


Interrupts

Interrupts in C program

Enable interrupt

Interrupt Service Routine (ISR)

- Located at fixed interrupt address
- Written in C or inline assembly
- Cannot pass parameters to / from ISR
- Access global variables / define local variables

Priority

High / low priority interrupts in PIC18

```
#pragma interrupt function_name (save = list)
```

- Declares high priority ISR
- Uses Fast Register Stack to save STATUS, WREG, BSR registers
- Interrupt ends fast return (restores context)

```
#pragma interruptlow function_name (save = list)
```

- Declares low priority ISR
- Saves / restores STATUS, WREG, BSR using software stack

Interrupt Example

main starts timer0
timer0 counts 2^8 ⇒ interrupt ⇒ counter++ → portB

```
#include <p18F242.h>
#include <timers.h>
unsigned char counter = 0;
#pragma code high_vector=0x08    // code section allocated to address 0x08h
void interrupt (void)            // section written in C → inline assembly
{
   _asm GOTO timer0_isr _endasm    //jump to ISR
}
#pragma code                     // default code section
#pragma interrupt timer0_isr     // specify code as high-priority ISR
void timer0_isr (void)
{
   PORTB = ++counter;
   INTCONbits.TMR0IF = 0;        // Clear TMR0 interrupt flag
}
void main (void)
{
   TRISB = 0b00000000;           // portB = outputs
   PORTB = 0;                    // outputs = 0
   OpenTimer0(TIMER_INT_ON & T0_SOURCE_INT & T0_8BIT & T0_PS_1_4);
            // enable interrupt, 8-bit count, internal clock, prescaler 1:4
   INTCONbits.GIE = 1;           // Enable global interrupt
   while (1)                     // wait for interrupt
   {
   }
}
```


Practical Example

Elevator Controller

Embedded Elevator Controller

General requirements

Travel up + down

- Store + display current position (floor)
- Store current direction (up / down)

Store + classify floor requests

- External — call to floor
- Internal — call from passenger
- Direction — up / down from current position

Passenger Call

Grant floor requests in order of current direction

- Announce floor
- Stop
- Clear request from memory
- Open door
- Delay
- Check obstacle in door
- Close door

Example of floor request ordering

Car at floor 0
Call to floor 4 → up
Passenger at 4 requests floor 1 → down
Down call to floor 3 → stop at 3
Up call to floor 2 → no stop at 2
Stop at floor 0
Up to floor 2
Passenger at 2 requests floor 5 → up

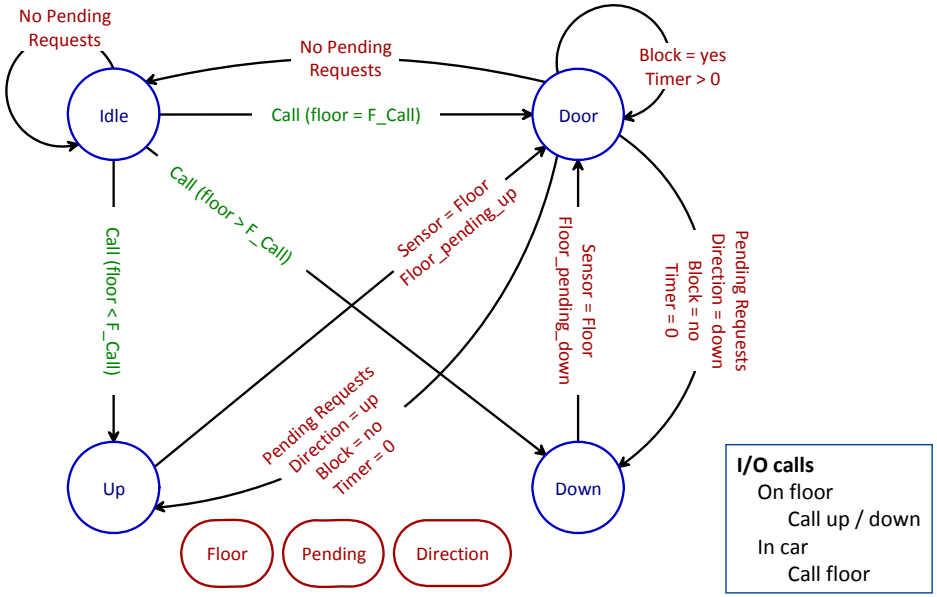
Embedded Elevator Controller

State machine

Elevator States	
Up	car traveling up
Down	car traveling down
Door	car stopped with door open
Idle	car not traveling + door closed + no pending requests
I/O Events	
Car	internal passenger request — call for floor **`f_pass`**
Call	external request — call from floor **`f_call`** to **`d_call`** (up/down)
Block	passenger in elevator doorway
Internal state (memory)	
Floor	current elevator position
Pending	bitmap of stored floor and direction requests
Direction	car executing up / down cycle

Embedded Elevator Controller

State transition diagram



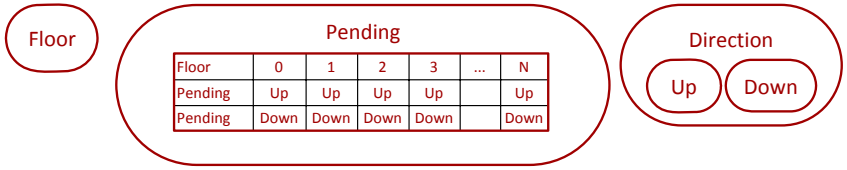
I/O calls

On floor
- Call up / down

In car
- Call floor

Embedded Elevator Controller

Sequential model for pending state



Floor	0	1	2	3	...	N
Pending	Up	Up	Up	Up		Up
Pending	Down	Down	Down	Down		Down

Pending bitmaps

```
pending_up / pending_down
```

Car — internal passenger request for floor f_car

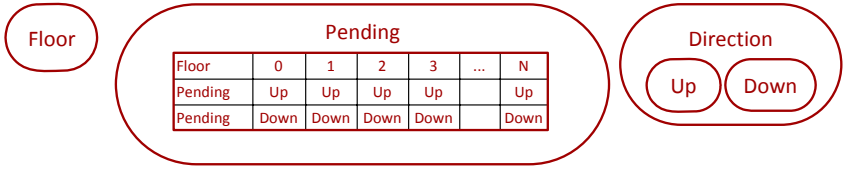
```
if (f_car > floor) pending_up.f_car ← 1
if (f_car < floor) pending_down.f_car ← 1
```

Call — external request from floor f_call to direction d_call

```
if (d_call ≠ floor){
    if (d_call = up) pending_up<f_ call> ← 1
    if (d_call = down) pending_down<f_ call> ← 1
}
```

Embedded Elevator Controller

Direction + idle



Floor	0	1	2	3	...	N
Pending	Up	Up	Up	Up		Up
Pending	Down	Down	Down	Down		Down

Idle

`pending_up = pending_down = 0`

Up / Down

Car executing up / down cycle

`up / down ← 1`

`pending_up ← 0 ⇒ up ← 0`

`pending_down ← 0 ⇒ down ← 0`

Embedded Elevator Controller

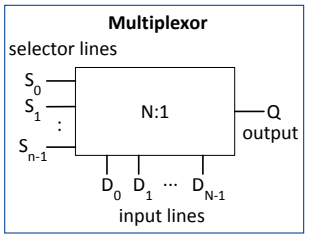
Call I/O

Floor call buttons (switches)

Ground floor — up call button

Top floor — down call button

Middle floors — up + down call buttons



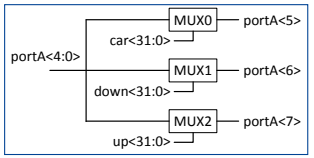
Floor buttons (switches)

Car — floor call buttons

Three N:1 multiplexor (MUX)

Select output Q among N = 2^n inputs

Selector = n-bit number $S_{n-1}...S_0$



3 multiplexors for port allocation

MUX0, MUX1, MUX2 selectors ← **`PortA<4:0>`** ⇒ ≤ 2^5 = 32 floors

Car call buttons <31:0>	**Multiplexor 0**	**`PortA<5>`** ← Q_0
Down call buttons <31:0>	**Multiplexor 1**	**`PortA<6>`** ← Q_1
Up call buttons <31:0>	**Multiplexor 2**	**`PortA<7>`** ← Q_2

Embedded Elevator Controller

Car position control

Open loop control

Elevator model

N floors in building

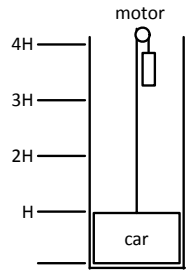
H meters per floor

Not reliable enough for human safety environment

Controller model

Initialize — car at ground floor (**`height = 0`**)

Up H × N meters to floor N



Closed loop control

Elevator model

`N` floors in building

If (car positioned at floor) sensor ← 1

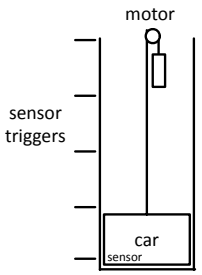
Controller model

Reset car at ground floor (**`floor ← 0`**)

Count sensor rising edge

Floor = **`floor`**

Floor sensor on `portB<3>`



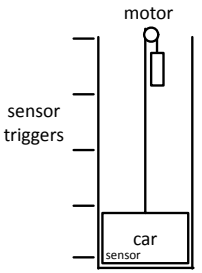
Embedded Elevator Controller

Motor control

Binary control

Motor on / off

Car jumps / stops with jerk



Slow acceleration / deceleration

Slowly increase / decrease speed from off to max

H-bridge with PWM control (output **`CPP1x = RC2`**)

Acceleration calculation

V_{max} = maximum speed

T_{max} = transition time for 0 to V_{max}

A = acceleration = V_{max} / T_{max}

T_{max} = V_{max} / A_{max}

Reasonable numbers for elevator

V_{max} = 1 meter/sec

A_{max} = 0.1 g = 0.1 × 10 meter/sec^2 = 1 meter/sec^2

T_{max} = V_{max} / A_{max} = 1 meter/sec / (1 meter/sec^2) = 1 second

Stopping at floor F

At floor F-1 start delay

After delay slow to 0.1 V_{max}

Stop on floor sensor = 1

Embedded Elevator Controller

Safety controls

Limit switches

Stop car at top and bottom of shaft

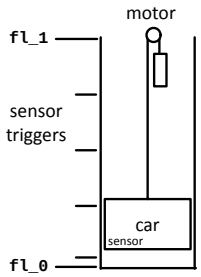
Input: if (car near top floor) switch **fl_1 = 1**

PortB<7> ← fl_1

Input: if (car near ground floor) **fl_0 = 1**

PortB<6> ← fl_0

Interrupt: change on **portB<7:4>**



Door open / closed

Output: **PortB<0> ← close_door_actuator**

Input: if (door open) **switch dr_open = 1**

PortB<1> ← dr_open

Door blocked

Input: if (door blocked) **switch dr_blk = 1**

PortB<2> ← dr_blk

Embedded Elevator Controller

Code skeleton

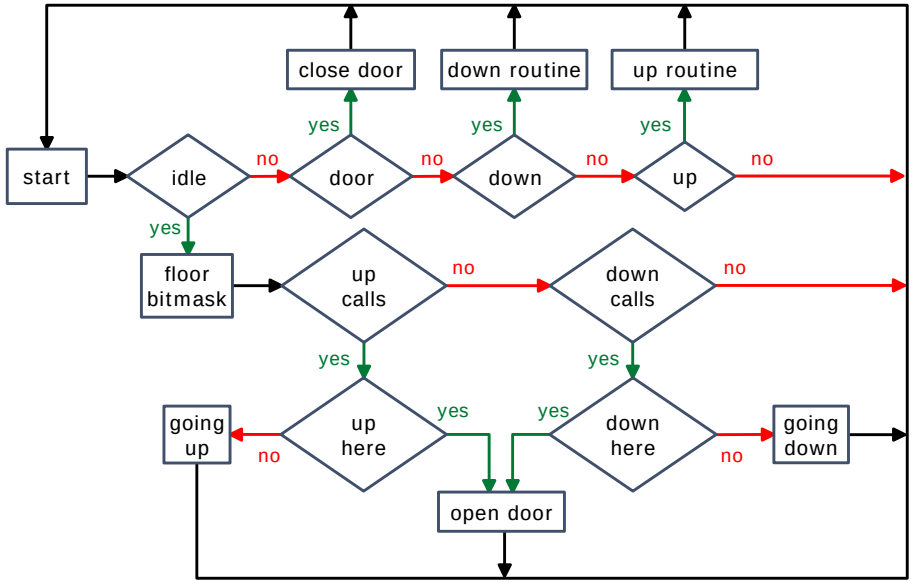
```
includes, defines, macros
reset + initialize                          // start elevator on ground floor
main
{
while (1){
            reset WDT
            read call + passenger buttons
            switch (state) {
              case idle:
                if (new pending calls) begin up/down cycle
                break
              case door open:
                close door
                break
              case going down:
                down routine
                break
              case going up:
                up routine
                break
              default:
                break
            }
          }
}
functions
```

Up / Down Routines

```
if (reaching floor) {
        update floor
        stop
        open door
        close door
        if (pending up/down) continue up/down
}
```


Embedded Elevator Controller

Flow chart of main loop



Embedded Elevator Controller

Code — 1

```
// header files

#include <p18F2420.h>
#include <delays.h>
#include <timers.h>
#include <pwm.h>
#include <portb.h>


#define FLOORS 32

// function prototypes

void open_door( void );        // open door
void close_door( void );       // close door
void init_up( void );          // start car traveling up
void init_dn( void );          // start car traveling down
void up( void );               // process tasks for car moving up
void down( void );             // process tasks for car moving down
void stop_up( void );          // stop car traveling up
void stop_dn( void );          // stop car traveling down
void limit_isr( void );        // interrupt routine
                               // triggered at top / ground floor
```

Embedded Elevator Controller

Code — 2

```
// global variables

unsigned int floor;         // car position
unsigned short state;       // state bitmap
                            // 0 = idle, 1 = door, 2 = down, 3 = up
unsigned short cycle;       // direction cycle (0 = down / 1 = up)
unsigned long pending_dn;   // bitmap:  up call to floor i => bit i = 1
unsigned long pending_up;   // bitmap:  down call
unsigned long mask;         // index into bitmap
unsigned int selector;      // select floor to read buttons
unsigned short safety;      // copy of portB
unsigned short prev_RB3;    // previous reading of portB.bit3 (at floor)
unsigned int DC;            // PWM duty cycle

// configuration bits

#pragma config PWRT = ON        // power-up timer on
#pragma config WDT = ON         // watchdog timer on
#pragma config CCP2MX = PORTC   // CCP2 on RC1
#pragma config PBADEN = OFF     // PORTB<4:0> pins = digital I/O
#pragma config OSC = INTIO67    // oscillator = internal 8 MHz
                                // T_CY = 0.5 us = 500 ns
```


Embedded Elevator Controller

Code — 3

```
// interrupt vector
#pragma code high_vector=0x08    // code section at address 0x08h
void interrupt (void)
{
   _asm GOTO limit_isr _endasm   //jump to ISR
}
// interrupt service routine

#pragma code                     // place in default code section
#pragma interrupt limit_isr      // specify code as high-priority ISR
void limit_isr(void)
{
      short limit;
      limit = PORTB;             // read portB<7:6> => reset interrupt
      if (limit & 128) {
            stop_up();
            floor = FLOORS;
      }
      if (limit & 64) {
            stop_dn();
            floor = 0;
      }
      close_door();              // insure door closed
      state = 1;                 // car stopped and idle
}
```


Embedded Elevator Controller

Code — 4

```
void main (void)
{
//
// Reset code
//
      TRISA = 0b11100000;    // portA<7:5> -- inputs
                             //   portA<7> -- up call button
                             //   portA<6> -- down call button
                             //   portA<5> -- car call button
                             // portA<4:0> -- outputs
                             //   portA<4:0> -- 5-bit floor selector
      PORTA = 0;             // reset portA

      TRISB = 0b11111110;    // portB<7:1> -- inputs
                             //   portB<7> -- car near top floor
                             //   portB<6> -- car near ground floor
                             //   portB<5:4> -- pulled up to logic 1
                             //   portB<3> -- car positioned at floor
                             //   portB<2> -- door blocked
                             //   portB<1> -- door open
                             // portB<0> -- output
                             //   portB<0> -- door motor (1 = open)
      PORTB = 0;             // reset portB
```


Embedded Elevator Controller

Code — 5

```
//
// motor PWM configuration
//

TRISC = 0b11111111;  // portC = outputs
PORTC = 0;           // portC<2> = CPP1 -- PWM output
                     // portC<0> -- 1 = motor enabled
PORTCbits.RC0 = 0;   // disable motor

      // enable Timer2 with no interrupt and P = 2
OpenTimer2(TIMER_INT_OFF & T2_PS_1_1 & T2_POST_1_2);

      // enable PWM1 with 50% duty cycle
OpenPWM1(39);        // Enable PWM1, set PR2+1 = 40
                     // T_PWM = 4*40*2*0.125 us = 40 us

DC = 80;             // DC = 80 => T_ON = 80*2*0.125 us = 20 us
SetDCPWM1(DC);       // set PWM1 duty cycle DC
```

Embedded Elevator Controller

Code — 6

```
// initialize car at ground floor
cycle = 0;                                  // down cycle
pending_dn = 0;                             // clear pending calls
pending_up = 0;
close_door();
safety = PORTB;                             // read portB
if (!(safety & 64)) init_dn();              // go to ground floor

      // read fl_0 until reach ground floor
      // T_CY = 0.125 us => delay 1.25 ms
while (!(PORTB & 64)) Delay10KTCYx(1);
stop_dn();                                  // stop car
floor = 0;
state = 1;                                  // idle
safety = PORTB;                             // read portB
prev_RB3 = PORTB & 8;                       // copy of RB3
// enable interrupt on changes to portB<7:4>
// enable pull-ups
OpenPORTB( PORTB_CHANGE_INT_ON & PORTB_PULLUPS_ON);
```


Embedded Elevator Controller

Code — 7

```
// main loop
     while(1){
// reset watchdog timer
     _asm CLRWDT _endasm
// read call + passenger buttons
     safety = PORTB;    // read portB: interrupt on change in portB<7:6>
     mask = 1;
     for (selector = 0 ; selector < 32 ; selector++){
        // write to portA: read input pins followed by write of outputs
        // output lines portA<4:0> select floor to read
        PORTA = selector;
        if (PORTA & 32){                          // RA5 = car button
            if (selector == floor) open_door();        // call here
            if (selector > floor) pending_up |= mask;  // set bit
            if (selector < floor) pending_dn |= mask;  //   in bitmap
        }
        if (PORTA & 64){                          // RA6 = down call
            if (floor == selector) open_door();    // call to this floor
            else pending_dn |= mask;               // set bit in bitmap
        }
        if (PORTA & 128){                         // portA<7> = up call
            if (floor == selector) open_door();    // call to this floor
            else pending_up |= mask;               // set bit in bitmap
        }
        mask << 1;                                // next mask
     }
```


Embedded Elevator Controller

Code — 8

```
        switch (state) {
            case 1:   // idle state
                      if (pending_up != 0) {
                           cycle = 1;    // starting up cycle
                           init_up();    // travel up
                           break;
                      }
                      if (pending_dn != 0) {
                           cycle = 0;    // starting down cycle
                           init_dn();    // travel down
                           break;
                      }
                      break;
            case 2:   // door open state
                      close_door();      // close door
                      break;
            case 4:   // traveling down state
                      down();            // process down tasks
                      break;
            case 8:   // traveling up state
                      up();              // process up states
                      break;
            default:
                      break;
        }
    }   // end main while loop
}       // end main function
```


Embedded Elevator Controller

Code — 9

```
void open_door( void ){
     while(!(PORTB & 2)) {
          PORTBbits.RB0 = 1;   // try to open door until open
          Delay10KTCYx(10);    // T_CY = 0. 5 us
                               //   delay 5 ms
     }
     state = 2;                // door open state
}


void close_door( void ){
     while( (PORTB & 2) && !(PORTB & 4) ) {
          PORTBbits.RB0 = 0;   // close door unless blocked
          Delay10KTCYx(10);    // T_CY = 0. 5 us
                               //   delay 5 ms
     }
}
```

Embedded Elevator Controller

Code — 10

```
void init_up( void ){
      // accelerate motor from DC = 80 to 160
      //     in 80 steps over 1 second
      // (12.5 ms / step)
      int speed;
      state = 8;                              // traveling up state
      PORTCbits.RC0 = 1;                      // enable motor
      for (speed = 80 ; speed <= 160 ; speed++){
           SetDCPWM1(speed);                  // PWM1 duty cycle
           Delay1KTCYx(25);                   // T_CY = 0.5 us
                                              //   delay 12.5 ms
      }
}
void init_dn( void ){
      // accelerate motor from 80 to 0
      //     in 80 steps over 1 second
      int speed;
      state = 4;                              // traveling down state
      PORTCbits.RC0 = 1;                      // enable motor
      for (speed = 80 ; speed >= 0 ; speed--){
           SetDCPWM1(speed);                  // PWM1 duty cycle
           Delay1KTCYx(25);                   // T_CY = 0.5 us
                                              //   delay 12.5 ms
      }
}
```


Embedded Elevator Controller

Code — 11

```
void up( void ){
      unsigned long floormask = 1;            // bitmap index
      // floor++ if car at floor for first time
      if ( (PORTB & 8) && (prev_RB3 == 0) ) floor++;
      prev_RB3 = PORTB & 8;                   // save last RB3
      floormask << floor + 1;                 // update index
      // if next floor is pending begin to stop
      if (pending_up & floormask == 1) stop_up();
}
void down( void ){
      unsigned long floormask = 1;            // bitmap index
      // floor-- if car at floor for first time
      if ( (PORTB & 8) && (prev_RB3 == 0) ) floor--;
      prev_RB3 = PORTB & 8;                   // save last RB3
      floormask << floor - 1;                 // update index
      // if next floor is pending begin to stop
      if (pending_dn & floormask == 1) stop_dn();
}
```


Embedded Elevator Controller

Code — 12

```
void stop_up( void ){
     int speed;
     for (speed = 160 ; speed >= 88 ; speed--){
     // 80 + 10% * 80 = 88
           SetDCPWM1(speed);                  // set PWM1 duty cycle speed
           Delay1KTCYx(25);                   // T_CY = 0.5 us
                                              //   delay 12.5 ms
     }
     while (!(PORTB & 8)) Delay10KTCYx(10);
     // wait for floor
     SetDCPWM1(80);                           // stop
     PORTCbits.RC0 = 0;                       // disable motor
     open_door();
     Delay10KTCYx(3000);                      // delay = 15 sec
     close_door();
     if (pending_up != 0) init_up();
     else if (pending_dn != 0) {
           cycle = 0;                         // start down cycle
           init_dn();
     }
     else {
           state = 1;                         // idle state
           cycle = 0;
     }
}
```


Embedded Elevator Controller

Code — 13

```
void stop_dn( void ){
     int speed;
     for (speed = 0 ; speed <= 72 ; speed++){
     // 80 - 10% * 80 = 72
           SetDCPWM1(speed);                  // set PWM1 duty cycle speed
           Delay1KTCYx(25);                   // T_CY = 0.5 us
                                              //   delay 12.5 ms
     }
     while (!(PORTB & 8)) Delay10KTCYx(10);
     // wait for floor
     SetDCPWM1(80);                           // stop
     PORTCbits.RC0 = 0;                       // disable motor
     open_door();
     Delay10KTCYx(3000);                      // delay = 15 sec
     close_door();
     if (pending_dn != 0) init_dn();
     else if (pending_up != 0) {
           cycle = 0;                         // start up cycle
           init_up();
     }
     else {
           state = 1;                         // idle state
           cycle = 0;
     }
}
```