Programming the PIC18 Using C- Coding

Dr. Farahmand

Updated: 3/14/12

The C Compiler

- The C18 compiler is a free program for students used for programing the PIC in C-Language.
- Programming in C-Language greatly reduces development time.
- C is NOT as efficient as assembly
  - A *good* assembly programmer can *usually* do better than the compiler, no matter what the optimization level – C WILL use more memory

PIC Compiler

A compiler converts a high-level language program to machine instructions for the target processor

A cross-compiler is a compiler that runs on a processor (usually a PC) that is different from the target processor

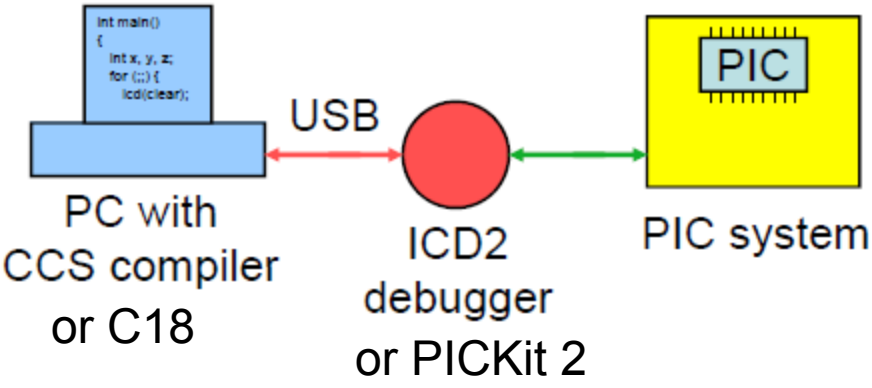
Most embedded systems are now programmed using the C/C++ language

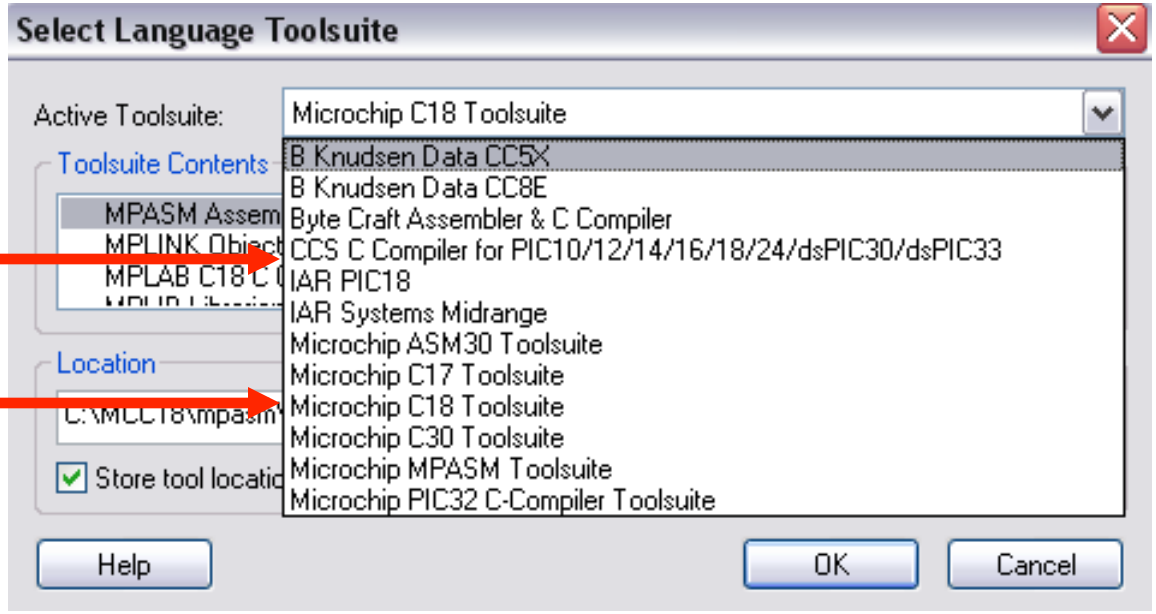
Third Party C-Compilers Available:
HI-TECH - PICCTM, www.htsoft.com
IAR - EWB-PIC, www.iar.com
CCS PIC18 C Compiler, www.ccsinfo.com

CCS Compiler

- **Provides C++ like reference parameters to functions**
- **C Compiler Quick Start Webinar**
  - http://www.ccsinfo.com/videos/quickStart.asx
  - For more click here: http://www.personal.rdg.ac.uk/~stsgrimb/teaching/programming_pic_microcontrollers.pdf

C and Assembly Compilers





The C18 Compiler

- Mixed language programming using C-language with assembly language is supported by the C18 compiler.
  - Assembly blocks are surrounded with at _asm and a _endasm directives to the C18 compiler.
  - Assembly code can also be located in a separate asm file
- The compiler normally operates on 8-bit bytes (char), 16-bit integers (int), and 32-bit floating-point (float) numbers.
- In the case of the PIC, 8-bit data should be used before resorting to 16-bit data.
  - Floats should only be used when absolutely necessary.

The C Library Reference Guide

- Read the practice exercise to see how you can combine C and ASM codes together

For example:

```
_asm
  /* User assembly code */
  MOVLW 10        // Move decimal 10 to count
  MOVWF count, 0

  /* Loop until count is 0 */
  start:
    DECFSZ count, 1, 0
    GOTO done
    BRA start
  done:
_endasm
```

Note: It is possible to use extended assembly instructions using C18 compiler – must be purchased! (Brey p.414)

```
void main (void)
{
    unsigned    char    RegALSBL;
    unsigned    char    count = 0;
    unsigned    char    asm_count;
    // Initialization
```

Note that we use:
// for comments!
A bit must be defined!

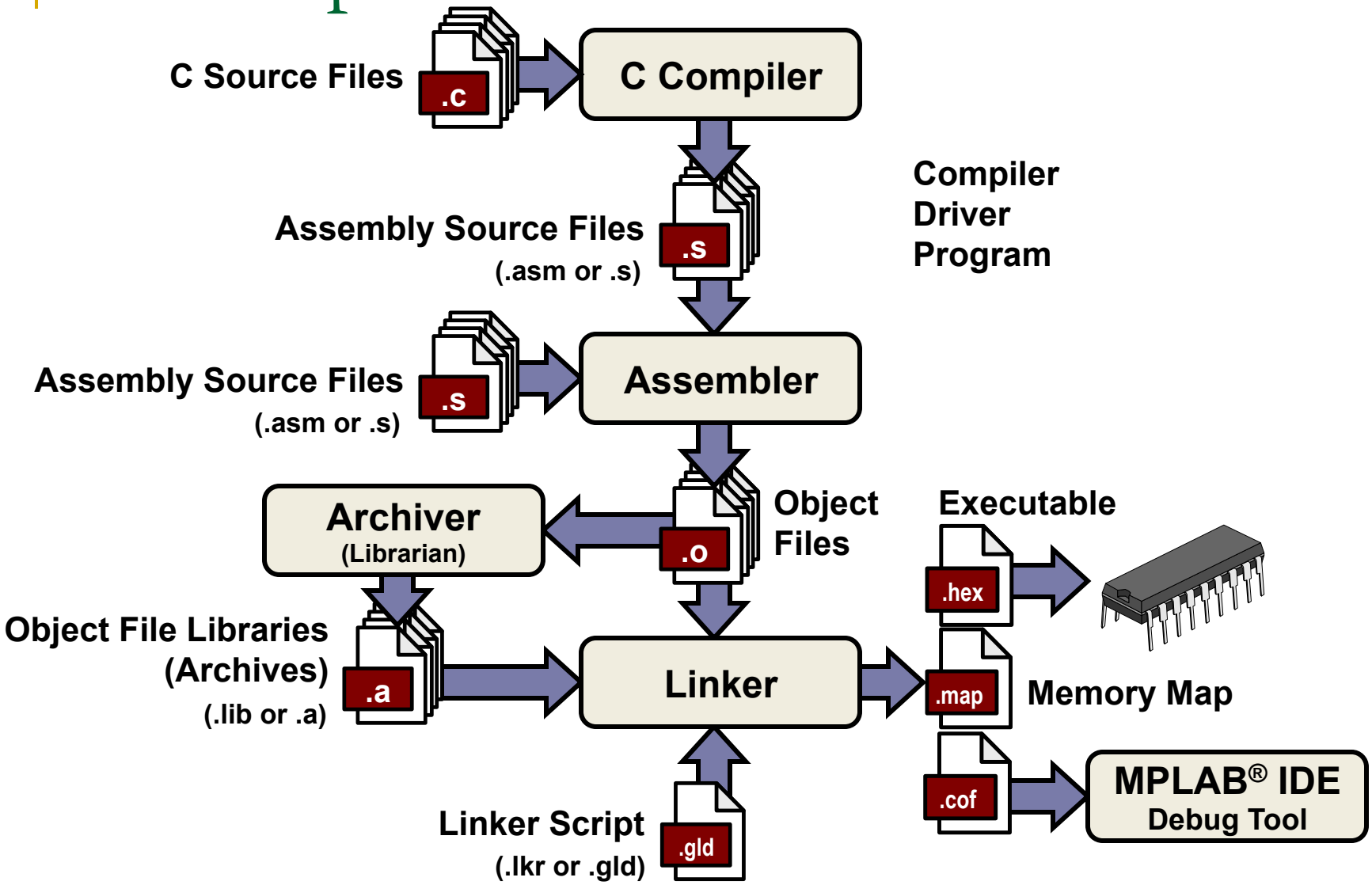
```
    _asm
     /* User assembly code */
        MOVLW 0x10
        MOVLW 0x20
        MOVWF asm_count,0
     /* Loop until count is 0
     start:
       DECFSZ asm_count, 1, 0
       GOTO done
       BRA start
     done:
     endasm
```

For more information: http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf#51

Setting up the IDE in MPLAB

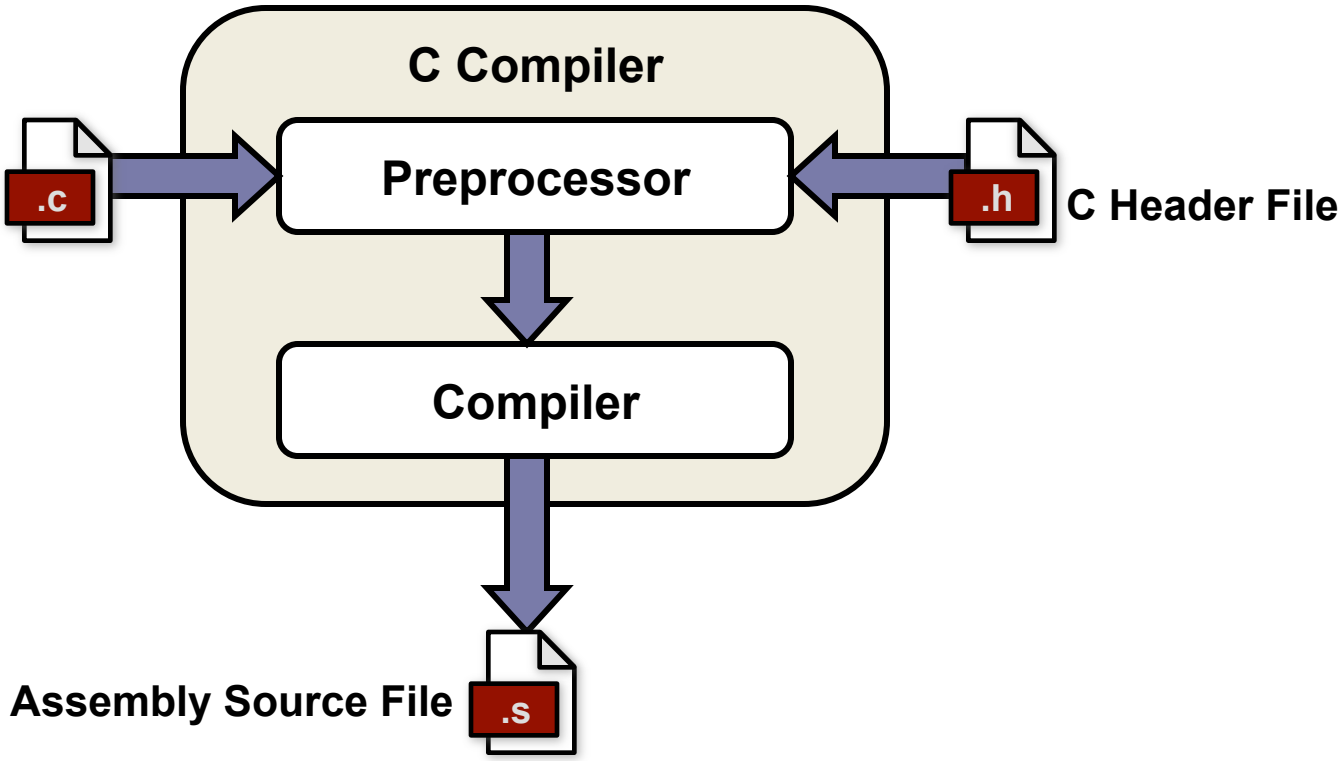
- When writing software in C-Language, the IDE is setup in a similar fashion to assembly language.
- Start the IDE and perform the steps outlined on the next few slides.

Development Tools Data Flow



Cof: Common object file format

Development Tools Data Flow



C Runtime Environment

- C Compiler sets up a runtime environment
  - Allocates space for stack
  - Initializes stack pointer
  - Copies values from Flash/ROM to variables in RAM that were declared with initial values
  - Clears uninitialized RAM
  - Disables all interrupts
  - Calls main() function (where your code starts)

C Runtime Environment

- Runtime environment setup code is automatically linked into application by most PIC® compiler suites
- *Usually* comes from either:
  - crt0.s / crt0.o (crt = C RunTime)
  - startup.asm / startup.o
- User modifiable if absolutely necessary

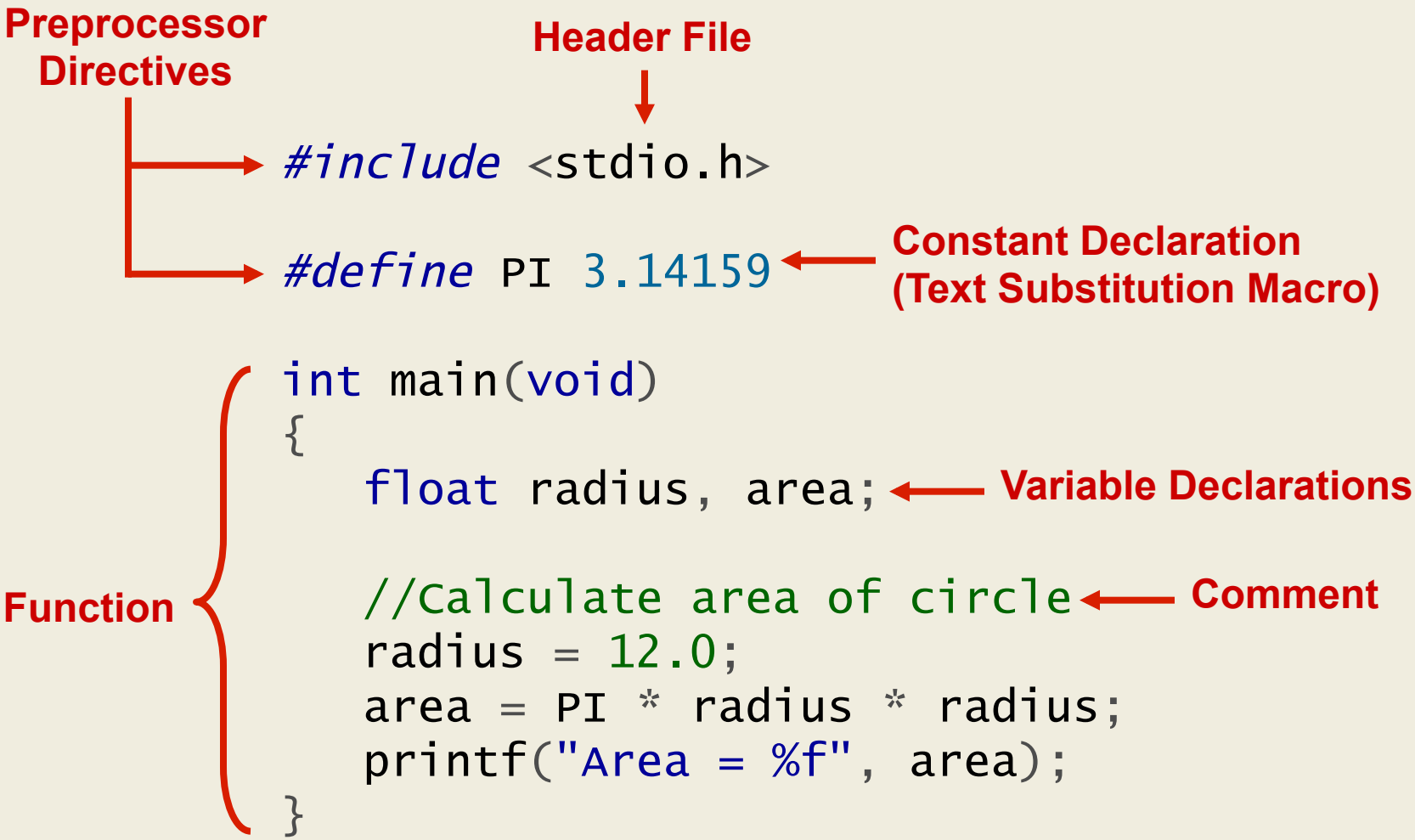
A little about C Programming ….

- For the most part you are on your own!
- Read the handouts in the library for more information.

Fundamentals of C

A Simple C Program

Example



Comments

Two kinds of comments may be used:

Block Comment

/* This is a comment */

Single Line Comment

// This is also a comment

```
/*************************************************
* Program: hello.c
* Author:  A good man
*************************************************/
#include <stdio.h>

/* Function: main() */
int main(void)
{
  printf("Hello, world!\n"); /* Display "Hello, world!" */
}
```

Variables and Data Types

A Simple C Program

Example

```
#include <stdio.h>

#define PI 3.14159

int main(void)
{
    float radius, area;

    //Calculate area of circle
    radius = 12.0;
    area = PI * radius * radius;
    printf("Area = %f", area);
}
```

Data Types

Variable Declarations

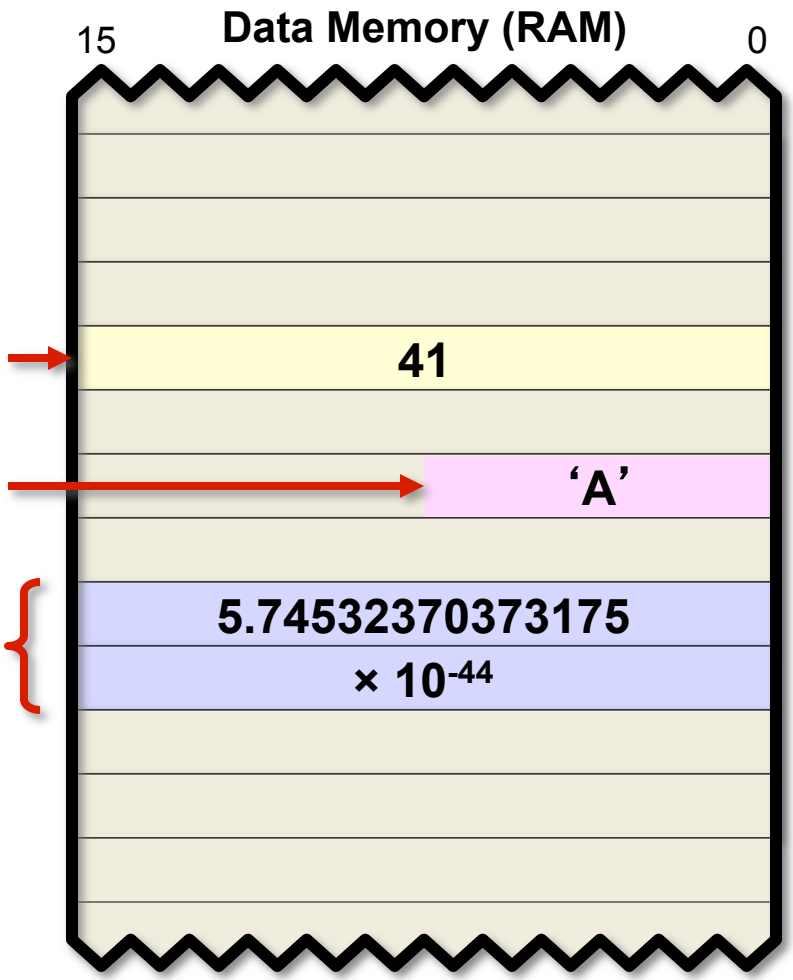
Variables in use

Variables

Variables are names for storage locations in memory

Variable declarations consist of a unique identifier (name)…

```
int warp_factor;
char first_letter;
float length;
```



Data Types

Type	Description	Bits
`char`	single character	8
`int`	integer	16
`float`	single precision floating point number	32
`double`	double precision floating point number	64

The size of an `int` varies from compiler to compiler.

- MPLAB-C30 `int` is 16-bits
- MPLAB-C18 `int` is 16-bits
- CCS PCB, PCM & PCH `int` is 8-bits
- Hi-Tech PICC `int` is 16-bits

Data Type Qualifiers

Modified Integer Types

Qualifiers: `unsigned`, `signed`, `short` and `long`

Qualified Type	Min	Max	Bits
`unsigned char`	0	255	8
`char`, `signed char`	-128	127	8
`unsigned short int`	0	65535	16
`short int`, `signed short int`	-32768	32767	16
`unsigned int`	0	65535	16
`int`, `signed int`	-32768	32767	16
`unsigned long int`	0	$2^{32}-1$	32
`long int`, `signed long int`	-2^{31}	-2^{31}	32
`unsigned long long int`	0	$2^{64}-1$	64
`long long int`, `signed long long int`	-2^{31}	-2^{31}	64

Literal Constants

- A literal is a constant, but a constant is not a literal
  - `#define MAXINT 32767`
  - `const int MAXINT = 32767;`
  - Constants are labels that represent a literal
  - **Literals** are values, often assigned to symbolic constants and variables

- Literals or Literal Constants
  - Are "hard coded" values
  - May be numbers, characters, or strings
  - May be represented in a number of formats (decimal, hexadecimal, binary, character, etc.)
  - Always represent the same value (5 always represents the quantity five)

Literal Constants

Example

```
unsigned int a;
unsigned int c;
#define b 2

void main(void)
{
   a = 5;
   c = a + b;
   printf("a=%d, b=%d, c=%d\n", a, b, c);
}
```

Literal

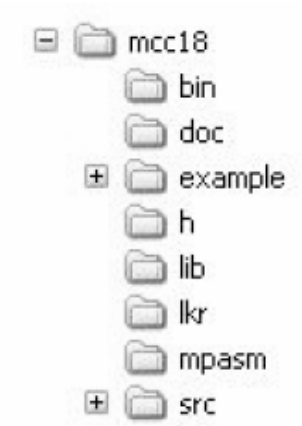
Literal

Using IDE

MPLAB C18 Directory Structure

- MPLAB C18 can be installed anywhere on the PC. Its default installation directory is the C:\mcc18 folder.
- MPLAB IDE should be installed on the PC prior to installing MPLAB C18.

MPLAB® C18 DIRECTORY STRUCTURE



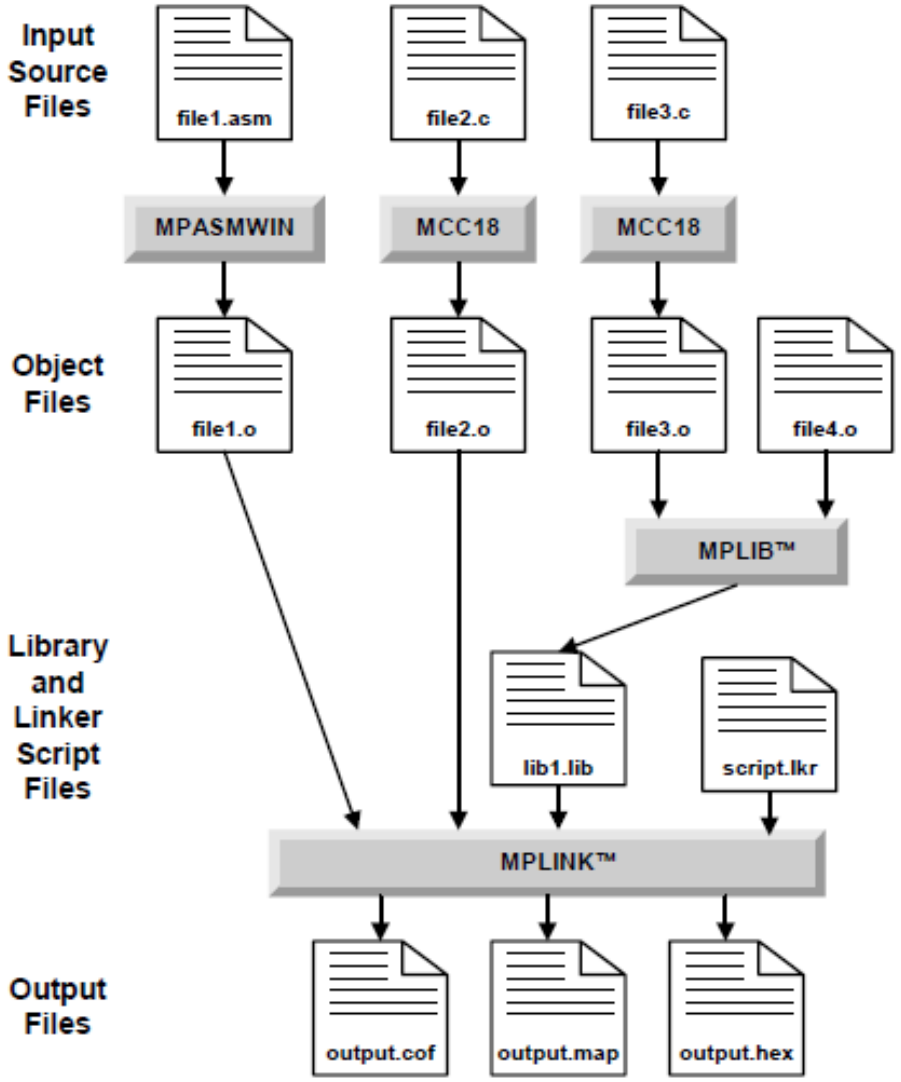
h: Contains the header files for the standard C library and the processor-specific libraries for the supported PICmicro® MCUs.

lib: Contains the standard C library (clib.lib or clib_e.lib), the processor-specific libraries (p18*xxxx*.lib or p18*xxxx*_e.lib, where *xxxx* is the specific device number) and the start-up modules (c018.o, c018_e.o, c018i.o, c018i_e.o, c018iz.o, c018iz_e.o).

lkr: Contains the linker script files for use with MPLAB C18.

mpasm: Contains the MPASM assembler and the assembly header files for the devices supported by MPLAB C18 (p18xxxx.inc).

LANGUAGE TOOLS EXECUTION FLOW



Installation Notes for MCC 18

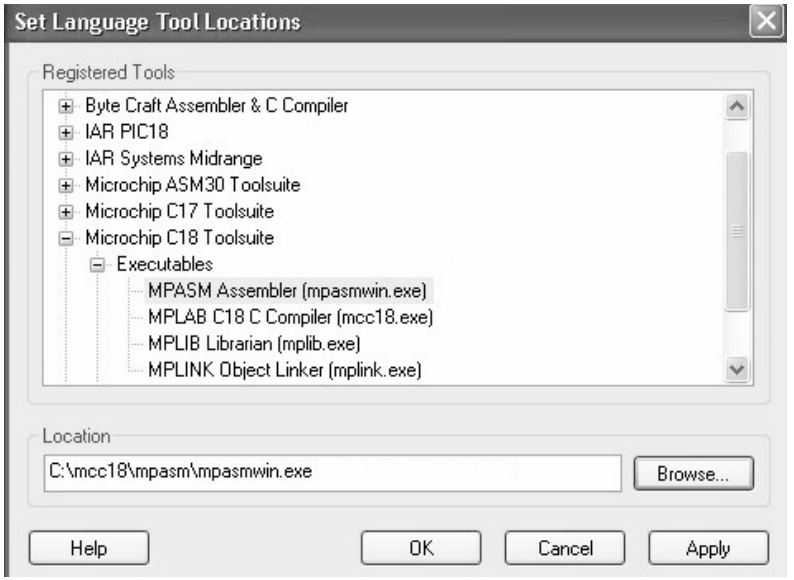
- Make sure executable file locations are properly assigned

MPASM Assembler should point to the assembler executable, `MPASMWIN.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\mpasm\MPASMWIN.exe`

MPLAB C18 C Compiler should point to the compiler executable, `mcc18.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\bin\mcc18.exe`

MPLINK Object Linker should point to the linker executable, `MPLink.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\bin\MPLink.exe`

MPLIB Librarian should point to the library executable, `MPLib.exe`, under "Location". If it does not, enter or browse to the executable location, which is by default:
`C:\mcc18\bin\MPLib.exe`



Setting up the IDE in MPLAB

1. Under the "Configure" label on the menu bar, select "Select Device" from the dropdown menu and choose the microcontroller for the project.

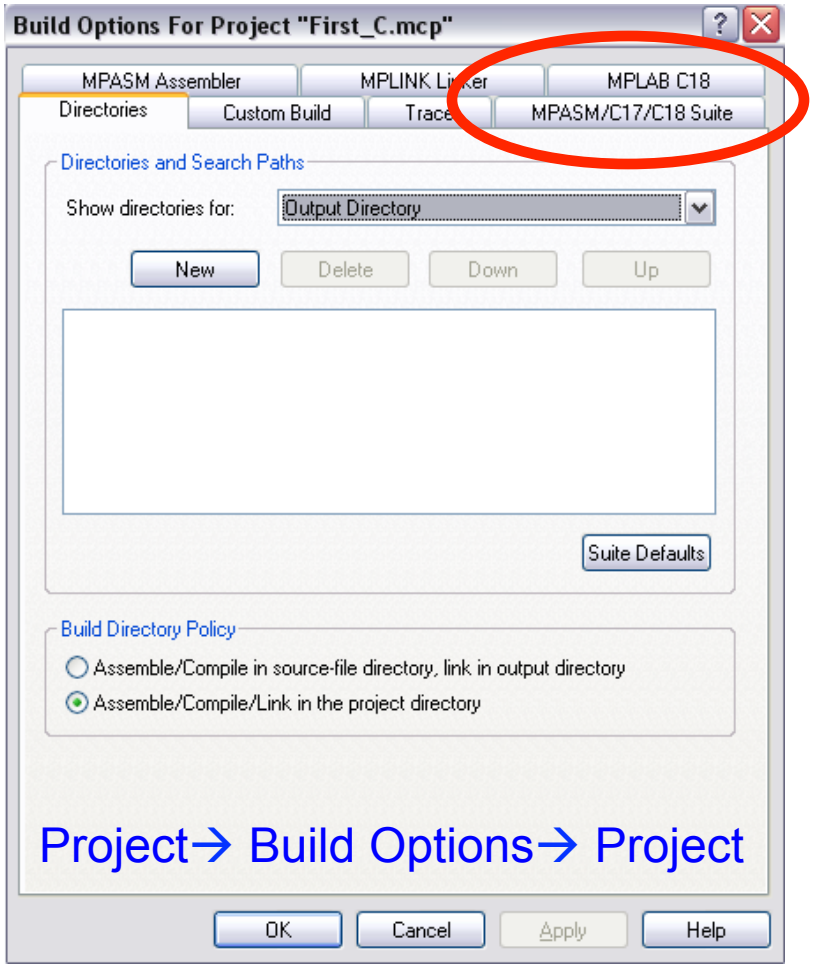
2. Under the "Project" label on the menu bar, select "Project Wizard" from the dropdown menu and again select the microcontroller for the project.

3. In Step 2 of the project wizard, select the "Microchip C18 Toolsuite" and click next. The paths are all correct of the C18 compiler is installed properly.

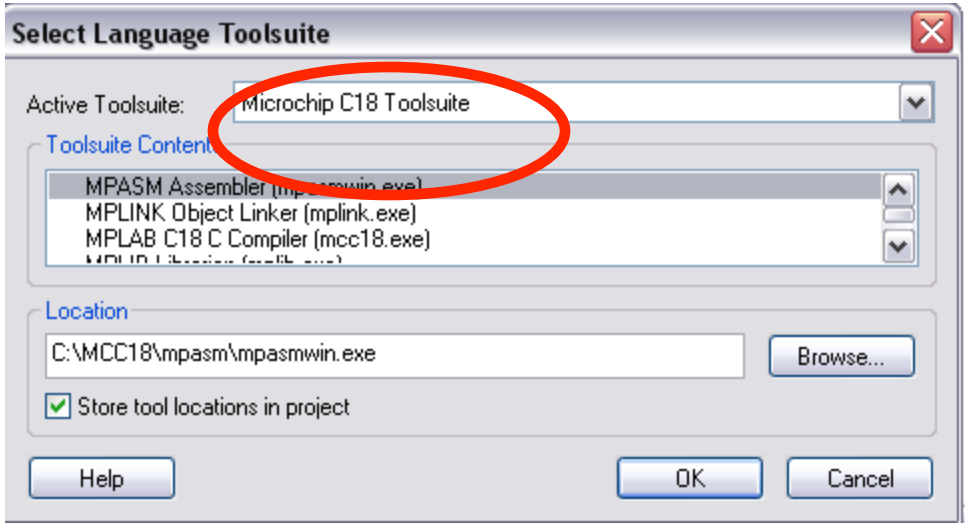
4. Enter a name for the project and a directory and then click on next.

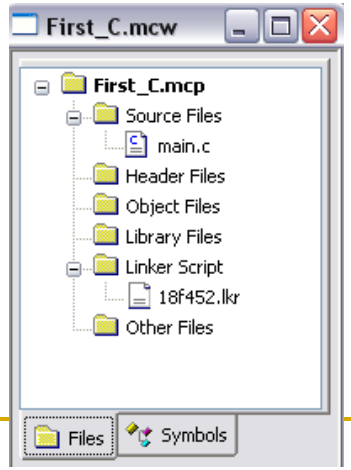
Installing C18 Compiler

Project→ Select Language Toolsuite



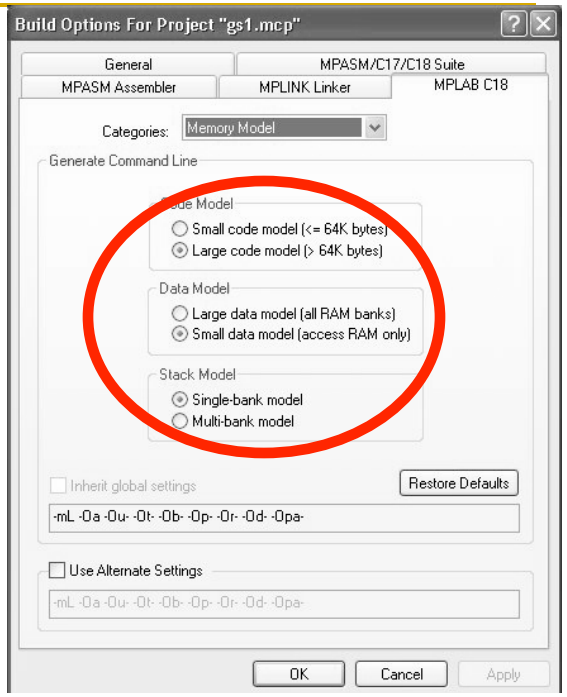
Project→ Build Options→ Project

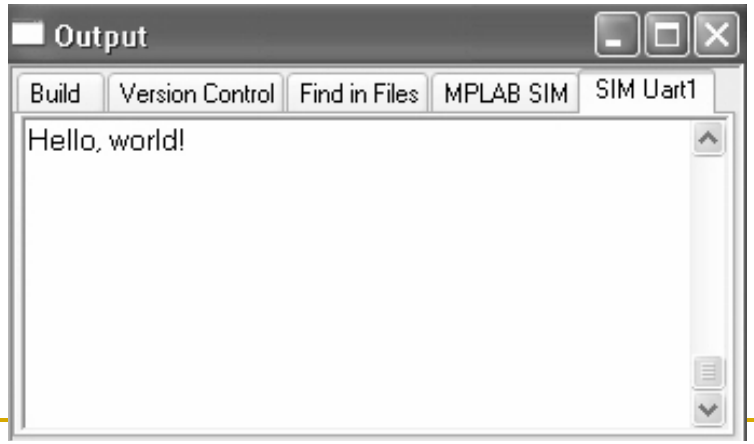


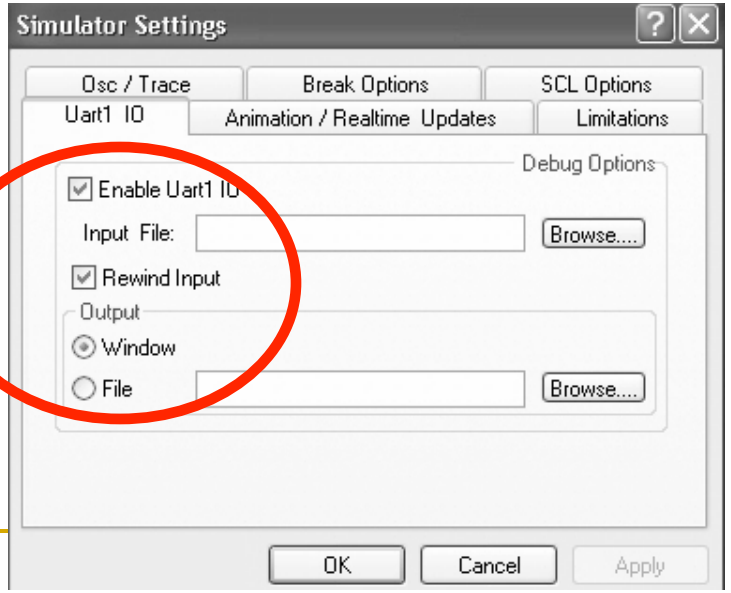


Program Examples

- Make and Build the project
- If you are using standard outputs:
  - Select *Debugger>Settings* and click on the **Uart1 IO** tab. The box marked
  - **Enable Uart1 IO** should be checked, and the **Output** should be set to **Window**
- **Select large code model**







Example

#pragma is Directive instructing the compiler program – indicating the setup for configuration bits

```
/** C O N F I G U R A T I O N   B I T S ******************************/

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF                  // CONFIG1H
#pragma config PWRT = OFF, BOREN = SBORDIS, BORV = 30                   // CONFIG2L
#pragma config WDTEN = OFF, WDTPS = 32768                               // CONFIG2H
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = ON, CCP2MX = PORTC   // CONFIG3H
#pragma config STVREN = ON, LVP = OFF, XINST = OFF                      // CONFIG4L
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF               // CONFIG5L
#pragma config CPB = OFF, CPD = OFF                                     // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF           // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF                       // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF       // CONFIG7L
#pragma config EBTRB = OFF                                              // CONFIG7H


/** I N C L U D E S **************************************************/
#include "p18f46k20.h"
#include "delays.h"

/** V A R I A B L E S ************************************************/
#pragma udata // declare statically allocated uninitialized variables
unsigned char LED_Number;  // 8-bit variable

/** D E C L A R A T I O N S ******************************************/
// declare constant data in program memory starting at address 0x180
#pragma romdata Lesson3_Table = 0x180
const rom unsigned char LED_LookupTable[8] = {0x01, 0x02, 0x04, 0x08,
                                              0x10, 0x20, 0x40, 0x80};

#pragma code    // declare executable instructions

void main (void)
{
```

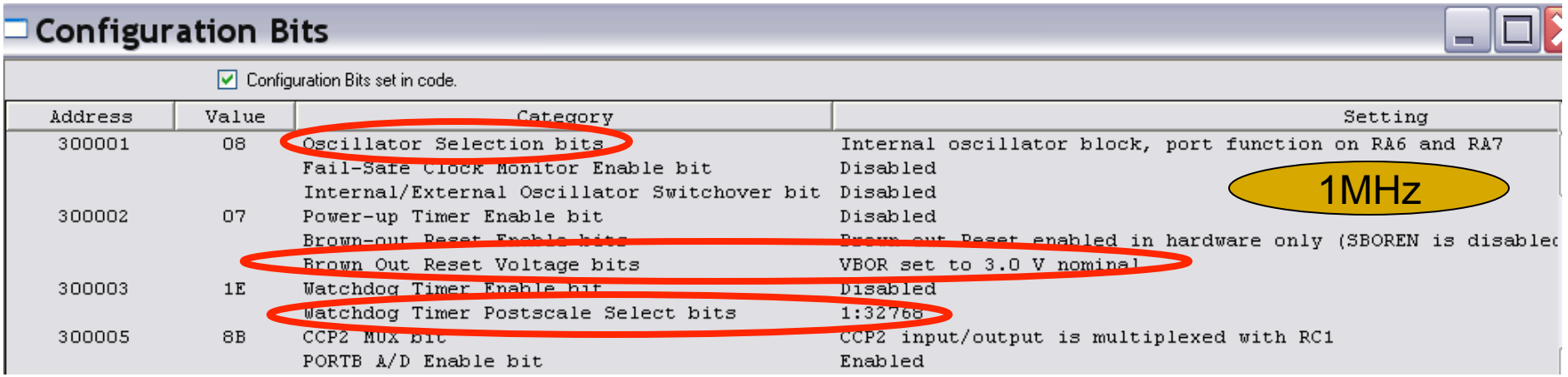
Configuration Bits

Modified for PIC18F4xK20

#pragma is used to declare directive;
Config directive allows configuring MCU operating modes; device dependent

```
/** C O N F I G U R A T I O N   B I T S ******************************/

#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF                  // CONFIG1H
#pragma config PWRT = OFF, BOREN = SBORDIS, BORV = 30                   // CONFIG2L
#pragma config WDTEN = OFF, WDTPS = 32768                               // CONFIG2H
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = ON, CCP2MX = PORTC   // CONFIG3H
#pragma config STVREN = ON, LVP = OFF, XINST = OFF                      // CONFIG4L
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF               // CONFIG5L
#pragma config CPB = OFF, CPD = OFF                                     // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF           // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF                       // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF       // CONFIG7L
#pragma config EBTRB = OFF                                              // CONFIG7H
```



Other #pragma Uses

Modified for PIC18F4xK20

- Two directives can be used to allocate data in specific RAM address
  - udata: uninitialized data is stored anywhere in the file register space
  - :idata Initialized data in file registers before the program execution begins

```
#pragma udata mysection = 0x300
unsigned char LED_Number;  // 8-bit variable
unsigned int AnotherVariable;
```

In the RAM starting with 0x300 we reserve space for mysection and all the variables defined after that)

Other #pragma Uses

Modified for PIC18F4xK20

- Two directives defining program memory sections:
  - code: is used to compile all the subsequent instructions into the program memory space
    - For example: #pragma code main = 0x3A14 // Indicates the MAIN program starts from location 0x3A14
  - romdata: data stored in the program memory

```
/** D E C L A R A T I O N S *****************************************/
// declare constant data in program memory starting at address 0x180
#pragma romdata Lesson3_Table = 0x180
const rom unsigned char LED_LookupTable[8] = {0x01, 0x02, 0x04, 0x08,
                                              0x10, 0x20, 0x40, 0x80};

#pragma code    // declare executable instructions

void main (void)
{
```

Program Memory:

```
0170   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ........ ........
0180   0201 0804 2010 8040 FFFF FFFF FFFF FFFF  ..... @. ........
0190   FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF  ........ ........
```

Other #pragma Uses

Modified for PIC18F4xK20

```
#pragma udata start_storing = 0x300
unsigned char random_values_stored[100];  // save the last 100 random numbers

///// Storing values in ROM - Not accessible when the program starts
#pragma romdata storing_in_rom = 0x4000
rom char rom_string[] = "These are the saved values:";
//rom char random_values_stored[100];
```

Program Memory

Address	00	02	04	06	08	0A	0C	0E	ASCII
3FE0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
3FF0	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	FFFF	....
4000	6854	7365	2065	7261	2065	6874	2065	6173	These ar e the sa
4010	6576	2064	6176	756C	7365	003A	FFFF	FFFF	ved valu es:.....

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
2D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
2E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
2F0	00	00	00	00											00	00	....
300	01	36	2F	BC											00	00	.6/..... ...
310	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

After executing the program

Example /*

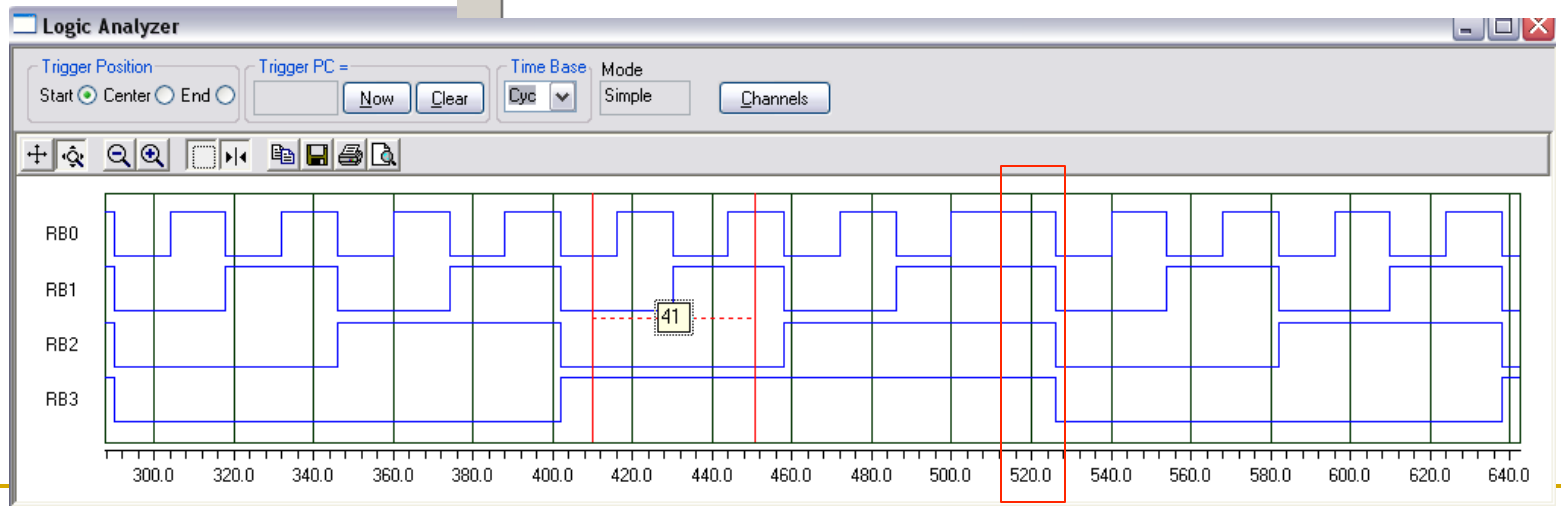
```
 * This is example 1 from "Getting Started with MPLAB C18".
 */
#include <p18cxxx.h>   /* for TRISB and PORTB declarations */
/* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
int counter;
void main (void)
{
   counter = 1;
   TRISB = 0;                    /* configure PORTB for output */
   while (counter <= 15)
   {
     PORTB = counter;     /* display value of 'counter' on the LEDs */
     counter++;
   }
}
```

Example

Configuration bits

Select digital signals

```
#include <p18cxxx.h>    /* for TRISB and PORTB declarations */
/* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 *   - set HS oscillator
 *   - disable watchdog timer
 *   - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
int counter;
void main (void)
{
    counter = 1;
    ADCON1 = 0x7F;
    TRISB = 0;                  /* configure PORTB for output */
    while (1)
    {
        counter = 0;
        while (counter <= 15)
        {
            PORTB = counter;      /* display value of 'counter' on the LEDs */
            counter++;
        }
    }
}
```



Due to end of the loop (BRANCH)

Another Example

```
#include <p18cxxx.h>          //include port specifications
#include <delays.h>           //include time delays
/* Set configuration bits
 *   - set HS oscillator
 *   - disable watchdog timer
 *   - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
// ******************** DATA MEMORY VARIABLES ************
int counter;
// ********************** CONSTANTS ********************
#define MSEC Delay1KTCYx(1)      // MSEC = 1 millisecond
```

```
// ******************** MAIN PROGRAM ******************************
void main (void)
{
    ADCON1 = 0x7F;          // configure PORTS A and B as digital
                            //  this might need to be changed depending
                            //  on the microcontroller version.

    TRISB = 0;              // configure PORTB for output
    TRISA = 0xFF;           // configure PORTA for input
    PORTB = 0;              // LEDs off
    while ( 1 )             // program infinite loop
    {
        counter = 0;        // initialize counter
        while ( ( PORTA & 16 ) == 0 )    // while pushbutton is down
        {
            MSEC;           // wait 1 msec
            counter++;
            if ( counter == 1000 )       // 1 second
            {
                PORTB++;
                counter = 0;
            }
        }
```

Basic idea:
Initially RA4 is one
When switch is pressed RA4 =0
Then RA & 16 = 0 → Start Counting
& display the number on PORTB

Note: 16d = 0x10

Assume using 4MHz internal clock
(250 nsec)→ one instruction cycle is 1 usec
Delay1KTCYx(1)→ Delay of 1000 instruction cycles
Delay100TCYx(10)→ Delay of 1000 instruction cycles

Another Example

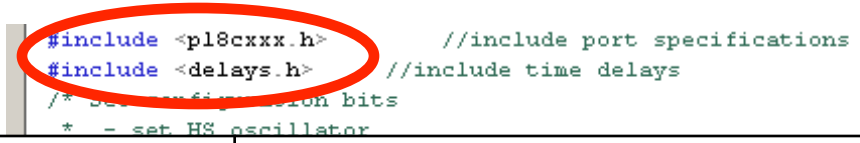
```
#include <p18cxxx.h>          //include port specifications
#include <delays.h>           //include time delays
/* Set configuration Bits
 *   - set HS oscillator
 *   - disable watchdog timer
 *   - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
// ***** ************** DATA MEMORY VARIABLES **************
int counter;
// ****** ************** *** CONSTANTS **********************
#define MSEC Delay1KTCYx(1)     // MSEC = 1 millisecond
```

```
// ******************* MAIN PROGRAM ****************************
void main (void)
{
    ADCON1 = 0x7F;          // configure PORTS A and B as digital
                            //  this might need to be changed depending
                            //  on the microcontroller version.

    TRISB = 0;              // configure PORTB for output
    TRISA = 0xFF;           // configure PORTA for input
    PORTB = 0;              // LEDs off
    while ( 1 )             // program infinite loop
    {
        counter = 0;        // initialize counter
        while ( ( PORTA & 16 ) == 0 )   // while pushbutton is down
        {
            MSEC;           // wait 1 msec
            counter++;
            if ( counter == 1000 )      // 1 second
            {
                PORTB++;
                counter = 0;
            }
        }
```

Alternatively, we could have:
➔ While (PORTAbits.RA4 == 0)

Time Delay Functions



Function	Example	Note
Delay1TCY	Delay1TCY();	Inserts a single NOP instruction into the program
Delay10TCYx	Delay10TCYx(10);	Inserts 100 instruction cycles (number must be between 0 and 255) (0 causes a delay of causes 2560)
Delay100TCYx	Delay100TCYx(10);	Inserts 1000 instruction cycles (number must be between 0 and 255) (0 causes a delay of 25,600)
Delay1KTCYx	Delay1KTCYx(3);	Inserts 3000 instruction cycles (number must be between 0 and 255) (0 causes a delay of 256,000)
Delay10KTCYx	Delay10KTCYx(20);	Inserts 20,000 instruction cycles (number must be between 0 and 255) (0 causes a delay of 2,560,000)

Random Number Generator

```
#include <p18cxxx.h>
/* Set configuration bits
 *   - set HS oscillator
 *   - disable watchdog timer
 *   - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
int seed;
```

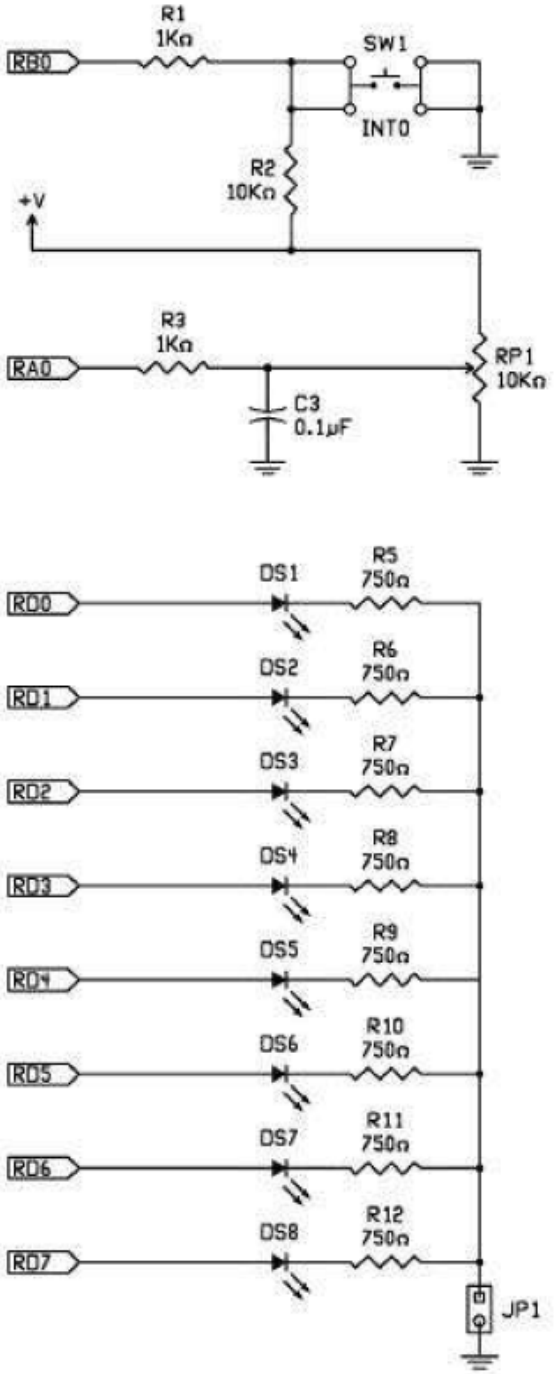
```
void main (void)
{
    ADCON1 = 0x7F;          // configure PORTS A and B as digital
                            //  this might need to be changed depending
                            //  on the microcontroller version.
    TRISB = 0;                      // configure PORTB for output
    TRISA = 0xFF;           // configure PORTA for input
    PORTB = 0;              // LEDs off
    seed = 1;               // self generated random number
    while ( 1 )             // repeat forever
    {
        while ( PORTAbits.RA4 == 0 )     // while pushbutton is down
        {
            seed++;
            if ( seed == 10 )        // if seed hits 10
                seed = 1;
            PORTB = seed;
        }
    }
}
```

Display a random number when RA4 is pressed
Random number will be between 0-9

Random Number Generator
Modified for PIC18F4xK20

```
int  seed;
void main (void)
{
      TRISD = 0b00000000; // PORTD bits 7:0 are all outputs (0)
      INTCON2bits.RBPU = 0; // enable PORTB internal pullups
      WPUBbits.WPUB0 = 1; // enable pull up on RB0
      ANSELH = 0x00; // AN8-12 are digital inputs (AN12 on RB0)
      TRISBbits.TRISB0 = 1; // PORTB bit 0
                            // (connected to switch) is input (1)

      TRISB=0xFF;
      PORTD=0;
      seed = 1;
      while (1)
      {
            while (PORTBbits.RB0 == 0)
            {
                  seed++;
                  if (seed == 10)
                        seed = 1;
                  PORTD = seed;
            }
      }
}
```



We can also use RAND()

```
#include <p18cxxx.h>
#include <delays.h>
#include <stdlib.h>
/* Set configuration bits
 *  - set HS oscillator
 *  - disable watchdog timer
 *  - disable low voltage programming
 */
#pragma config OSC = HS
#pragma config WDT = OFF
#pragma config LVP = OFF
void main (void)
{
        ADCON1 = 0x7F;          // configure PORTS A and B as digital
                                //      this might need to be changed depending
                                //      on the microcontroller version.
        TRISB = 0;              // configure PORTB for output
        PORTB = 0;              // LEDs off
        srand(1);           // Sets the seed of the random number
        while ( 1 )         // repeat forever
        {
                Delay10KTCYx(50);   // wait 1/2 second
                PORTB = rand();     // display a random number
                                    // rand() returns a random value

        }
}
```

Random Number Generator
Modified for PIC18F4xK20

```
#include "p18f46k20.h"
#include <stdio.h>
void main (void)
{
    TRISD = 0b00000000;       // PORTD bits 7:0 are all outputs (0)
    INTCON2bits.RBPU = 0;     // enable PORTB internal pullups
    WPUBbits.WPUB0 = 1;       // enable pull up on RB0
    ANSELH = 0x00;            // AN8-12 are digital inputs (AN12 on RB0)
    TRISBbits.TRISB0 = 1;     // PORTB bit 0 (connected to switch) is input (1)

    srand(1);                 // seed the random number
    TRISB=0xFF;
    PORTD=0;
    while (1)
    {
        while (PORTBbits.RB0 == 0)
        {
            Delay10KTCYx(50);  // wait 1/2 second
            PORTD = rand();              // display a random number
        }
    }
}
```

Common Conversion Functions in <stdlib.h>

Function	Example	Note
atob	atob(buffer)	Converts the number from string form in buffer; returned as a byte signed number (+127 to -128)
atof	atof(buffer)	Converts the number from string form in buffer; returned as a floating point number
atoi	atoi(buffer)	Converts the number from string form in buffer; returned as a 16-bit signed integer (+ 32,767 to -32, 768)
atol	atol(buffer)	Converts the number form string format in buffer; returned as a 32-bit signed integer (+ 2, 147, 483, 647 to – 2, 417, 483, 648)
btoa	btoa(num, buffer)	Converts the signed byte into a string stored at buffer
itoa	itoa(num, buffer)	Converts the signed 16-bit integer to a string stored at buffer
Itol	itol(num, buffer)	Converts the 32-bit signed integer to a string stored at buffer
rand	rand()	Returns a 16 bit random number (0 to 32, 767)
srand	srand(seed)	Sets the seed values to 16-bit integer seed
tolower	tolower(letter)	Converts byte-sized character letter from uppercase; returned as lowercase
toupper	toupper(letter)	Converts byte-sized character letter from lowercase, returns uppercase
ultoa	ultoa(num, buffer)	Same as itol, except num is unsigned

The C Library Reference Guide http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

Cool Example….

Generating various patterns on PORTD

```
#include <delays.h>
#include <p18F8720.h>
unsigned rom char led_tab [] = {0x00,0xFF,0x00,0xFF,0x00,0xFF,0x00,0xFF,
                                0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,
                                0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,
                                0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,
                                0x7F,0xBF,0xDF,0xEF,0xF7,0xFB,0xFD,0xFE,
                                0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,
                                0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,
                                0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,
                                0xFE,0xFD,0xFB,0xF7,0xEF,0xDF,0xBF,0x7F,
                                0x7E,0xFF,0x7E,0xFF,0x7E,0xFF,0x7E,0xFF,
                                0xBD,0xFF,0xBD,0xFF,0xBD,0xFF,0xBD,0xFF,
                                0xDB,0xFF,0xDB,0xFF,0xDB,0xFF,0xDB,0xFF,
                                0xE7,0xFF,0xE7,0xFF,0xE7,0xFF,0xE7,0xFF,
                                0xE7,0xFF,0xE7,0xFF,0xE7,0xFF,0xE7,0xFF,
                                0xDB,0xFF,0xDB,0xFF,0xDB,0xFF,0xDB,0xFF,
                                0xBD,0xFF,0xBD,0xFF,0xBD,0xFF,0xBD,0xFF,
                                0x7E,0xFF,0x7E,0xFF,0x7E,0xFF,0x7E,0xFF};
void main (void)
{
        unsigned char i;
        TRISD = 0x00; /* configure PORTB for output */
        while (1) {
                for (i = 0; i < 136; i++) {
                        PORTD = led_tab[i]; /* output a new LED pattern */
                        Delay 10KTCYx(200)); /* stay for about half a second */
                }
        }
}
```

Function	*Example*	*Note*
memchr	memchr (area51, 'a' , 23)	Search the first 23 bytes of area51 for an 'a'
memchrpgm	memchrpgm (area1, 65, 5)	Search the first 5 bytes of area1 for a 65 (if found, a pointer is returned to the character; if not, a null is returned)
memcmp	memcmp (area1, area2, 4)	Compare area1 with area2 for 4 bytes
memcmppgm	memcmppgm (area3, area4, 2)	Compare program memory area3 with program memory area4 for 2 bytes
memcmppgm2ram	memcmppgm2ram (a1, a2, 5)	Compare a1 with program memory a2 for 5 bytes
memcmpram2pgm	memcmpram2pgm (a3, a4, 6)	Compare program memory a3 with a4 for 6 bytes (returns <0 is first less than second returns ==0 if strings are equal returns >0 if first string is greater than second string)
memcpy	memcpy (a1, a2, 4)	Copies from a2 to a1 for 4 bytes
memcpypgm	memcpypgm (a3, a4, 5)	Copies program memory a4 to program memory a3 for 5 bytes
memcpypgm2ram	memcpypgm2ram (a5, a6, 7)	Copies program memory a6 to a5 for 7 bytes
memcpyram2pgm	memcpyram2pgm (a7, a8, 2)	Copies a8 to program memory a7 for 2 bytes
memmove	memmove (a1, a2 , 3)	Same as memcpy except overlapping regions are allowed
memmovepgm	memmovepgm (a3, a4, 3)	
memmovepgm2ram	memmovepgm2ram (d, e, 3)	
memmoveram2pgm	memmoveram2pgm (f, g, 45)	
strcat	strcat (str1, str2)	Append str1 with str2
strcatpgm	strcatpgm (str3, str4)	Append str3 in the program memory with str4
strcatpgm2ram	strcatpgmram (str5, str6)	Append str5 with program memory string str6
strcatram2pgm	strcatpgmram (str3, str4)	Same as strcatpgm
strchr	strchr (str1, 'a')	Find the first letter a in str1
strchrpgm	strchrpgm (str2, '0')	Find the first zero in str2
strcmp	strcmp (str1, str2)	Compares str1 to str2
strcmppgm	strcmppgm (str3, str4)	Compares str3 in program memory to program memory str4
strcmppgm2ram	strcmppgmram (str5, str6)	Compares str5 to program memory str6
strcmpram2pgm	strcmprampgm (str3, str4)	Compares program memory str3 to str4 (returns >0 if first string is less than second string returns == 0 if strings are equal returns <0 if first string is greater then second string)

Read these!
string.h

Copy data from program memory to data memory

Function	Description
memcpypgm2ram	Copy a buffer from ROM to RAM
memmovepgm2ram	Copy a buffer from ROM to RAM
strcatpgm2ram	Append a copy of the source string located in ROM to the end of the destination string located in RAM
strcpypgm2ram	Copy a string from RAM to ROM
strncatpgm2ram	Append a specified number of characters from the source string located in ROM to the end of the destination string located in RAM
strncpypgm2ram	Copy characters from the source string located in ROM to the destination string located in RAM

Example of <string.h> and <stlib.h>

- Using strlen() and atob()

```
char buffer[]= "The time is 8 o'clock";
char hour;
int a;

void main (void)
{

    for (a = 0; a < strlen(buffer); a++)
    {
        //printf ("a value is %d \n", a);
        if (buffer[a] >= '0' && buffer[a] <= '9')
        {
            //printf ("the buffer value %s \n", buffer[a]);
            break;
        }
    }
        hour = atob (buffer + a);
```

a

File Register

F00	54	68	65	20	74	69	6D	65	20	69	73	20	38	20	6F	27	The time is 8 o'
F10	63	6C	6F	63	6B	00	E2	11	00	00	00	00	00	20	00	00	clock... ....
F20	15	0C	00	FE	FF	00	00	00	00	00	00	00	00	00	00	00	....

The program finds a number in the string
Note: atob is defined in the stdlib.h table

Using ROM directives

- Rom directive → tell the compiler to place data in program memory
  - Near ROM → 16-bit address
  - Far ROM → 21-bit address

NEAR and FAR determine the address size in ROM (16 or 21 bit); Read the handout!

```
rom near char lookUpTable[][20] =
{       "my first message",
        "my second message",
        "my third message",
};

// data memory data
char buffer [20];

void main (void)
{
    strcpypgm2ram (buffer, lookUpTable[1]);
}
```

Second row

Program memory

```
00A0   5B1A D7BF 0012 796D 6620 7269 7473 6D20  .[....my  first m
00B0   7365 6173 6567 0000 0000 796D 7320 6365  essage.. ..my sec
00C0   6E6F 2064 656D 7373 6761 0065 0000 796D  ond mess age...my
00D0   7420 6968 6472 6D20 7365 6173 6567 0000   third m essage..
```

File register

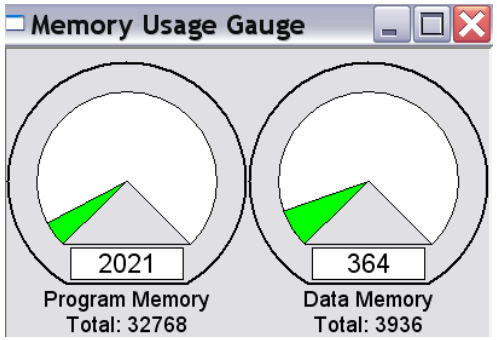
```
F00   6D 79 20 73 65 63 6F 6E 64 20 6D 65 73 73 61 67  my secon d messag
F10   65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  e....... ........
```

Example of using Math Functions

```
float Fr[10];
float L=1.0e-3;
float C=1.0e-6;
float mysqrtbuff;

void main (void)
{
    int a;
    for (a = 0; a < 10; a++)
    {
        Fr[a]= 1 / (6.2831853 * sqrt (L * C));
        mysqrtbuff = sqrt (L * C);
        L += 1.0e-6; // inductor value from 1mH to 10mH
    }
```

- Math function uses significant amount of memory
- Use <math.h>



Memory Usage with math.h

Verifying the Results

	A	B	C	D
1	L	C	Fr	SQRT
2	1.00000000E-03	1.00000000E-06	5.03292121E+03	3.16228E-05
3	1.00100000E-03	1.00000000E-06	5.03040664E+03	3.16386E-05
4	1.00200000E-03	1.00000000E-06	5.02789583E+03	3.16544E-05
5	1.00300000E-03	1.00000000E-06	5.02538877E+03	3.16702E-05
6	1.00400000E-03	1.00000000E-06	5.02288547E+03	3.1686E-05
7	1.00500000E-03	1.00000000E-06	5.02038590E+03	3.17017E-05
8	1.00600000E-03	1.00000000E-06	5.01789005E+03	3.17175E-05
9	1.00700000E-03	1.00000000E-06	5.01539793E+03	3.17333E-05
10	1.00800000E-03	1.00000000E-06	5.01290952E+03	3.1749E-05
11	1.00900000E-03	1.00000000E-06	5.01042480E+03	3.17648E-05
12	1.01000000E-03	1.00000000E-06	5.00794378E+03	3.17805E-05
13	1.01100000E-03	1.00000000E-06	5.00546644E+03	3.17962E-05
14	1.01200000E-03	1.00000000E-06	5.00299277E+03	3.18119E-05
15	1.01300000E-03	1.00000000E-06	5.00052277E+03	3.18277E-05

Symbol Name	Value
Fr	
[0]	5032.921
[1]	5030.406
[2]	5027.896
[3]	5025.388
[4]	5022.885
[5]	5020.385
[6]	5017.890
[7]	5015.397
[8]	5012.909
[9]	5010.425
L	0.001010000
mysqrtbuff	3.176476e-005

Note: Without math.h the program does compile!

However, correct results cannot be achieved!

Understanding Data Storage Using C18 C compiler-Example

Answer the following questions (LAB):

1- where is mydata stored? Which register?
2- Where is Z variable located at?
3- Where is e variable located at?
4- where is midata?
5- where does the main program start at?

```
#pragma code main = 0x50

rom near char midata[] = "HOLA";
unsigned    char    e;

void main(void)
{
    unsigned char mydata[]= "HELLO";
    unsigned    char    z;

    TRISD = 0;
    e = 9;
    for (z=0; z<5; z++)
        PORTD = mydata[z];
}
```

Program: Second_C

Passing Parameters Between C and ASM Codes

C Code

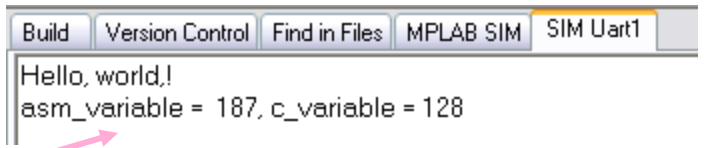
```
void main (void)
{
    your_assembly_code (); // call the assembly function

    //asm_variable = 0xA; //we can change the variable in C
    //c_variable = 0x12;

    _asm
        MOVLW   asm_variable
    _endasm

    printf ("Hello, world,!\n");
    printf( "asm_variable =  %d, c_variable = %d \n",
            asm_variable, c_variable );

}
```

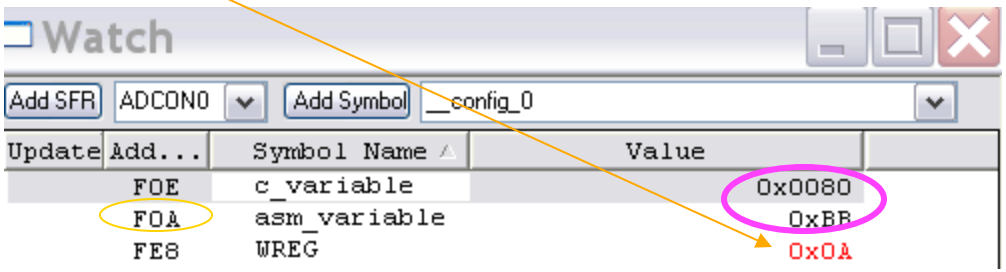


ASM Code

```
;; This is your actual assembly code....
Main:
    ; changing the variable in assembly
    movlw   0x80    ; clear bit 0 in W register
    movwf   c_variable

    movlw   0xBB    ; clear bit 0 in W register
    movwf   asm_variable

; End of your assembly code
GLOBAL your_assembly_code ; export so linker can see it
GLOBAL asm_variable   ; define the assembly variable
END
```



(Refer to Example Code: passing_parameters.c)

References

- Microchip.com
- Brey chapter 5
- Huang