



# **EC-310 Microprocessor & Microcontroller based Design**

---

**Dr. Ahsan Shahzad**

**Department of Computer and Software Engineering (DC&SE)**

**College of Electrical and Mechanical Engineering (CEME)**

**National University of Sciences and Technology (NUST)**

# PIC Programming in C

---

- Compilers produce hex files that we download into the ROM of the microcontroller.
- The size of hex file is critical as Microcontrollers have limited on-chip ROM.
- Assembly language produces hex file much smaller than C.

## Pros of Programming PIC18 in C

- Easier and less time consuming than assembly
- C programs easier to modify and update
- Availability of function libraries
- C code is portable to other microcontrollers

## Cons

- Increased size of hex file

# PIC C18 Compilers

---

## Role of compiler in PIC18 programming in C

- The data types supported, the types of operations performed, and the syntax depends upon the compiler used
- Even the number of bits allocated to each data type depend on the compiler used
- Generally, all the compilers use the syntax derived from ANSI C standard
- However, they are not 100 percent compliant with all the ANSI C standards

### Examples:

1. [Microchip C18 compiler](#): really the best and easiest to use. Perfect for professional use.
2. HI-TECH: Used when Microchip does not work (it was for a PIC16).
3. [CCS](#)
4. SourceBoost

# MPLAB C18 Compiler

---

- The MPLAB C18 compiler is designed for the PIC18 family of microcontrollers
- The focus in this course is on the C18 compiler.
- This is ANSI C 89 Compliant
- For further details visit
- [http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB\\_C18\\_Users\\_Guide\\_51288d.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_C18_Users_Guide_51288d.pdf)

# C Data types for the PIC18

**Goal:** C18 programmers is to create smaller hex file

- Good understanding and proper use of data types is essential for this.

**TABLE 2-1: INTEGER DATA TYPE SIZES AND LIMITS**

Type	Size	Minimum	Maximum
<code>char<sup>(1,2)</sup></code>	8 bits	-128	127
<code>signed char</code>	8 bits	-128	127
<code>unsigned char</code>	8 bits	0	255
<code>int</code>	16 bits	-32,768	32,767
<code>unsigned int</code>	16 bits	0	65,535
<code>short</code>	16 bits	-32,768	32,767
<code>unsigned short</code>	16 bits	0	65,535
<code>short long</code>	24 bits	-8,388,608	8,388,607
<code>unsigned short long</code>	24 bits	0	16,777,215
<code>long</code>	32 bits	-2,147,483,648	2,147,483,647
<code>unsigned long</code>	32 bits	0	4,294,967,295

**Note 1:** A plain `char` is signed by default.

# C Data types for the PIC18

**Goal:** C18 programmers is to create smaller hex file

- Good understanding and proper use of data types is essential for this.

## 2.1.2 Floating-point Types

32-bit floating-point types are native to MPLAB C18 using either the `double` or `float` data types. MPLAB C18 utilizes the IEEE-754 floating-point standard to represent floating-point types. The ranges of the floating-point type are documented in Table 2-2.

**TABLE 2-2: FLOATING-POINT DATA TYPE SIZES AND LIMITS**

Type	Size	Minimum Exponent	Maximum Exponent	Minimum Normalized	Maximum Normalized
float	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e + 38$
double	32 bits	-126	128	$2^{-126} \approx 1.17549435e - 38$	$2^{128} * (2-2^{-15}) \approx 6.80564693e + 38$



# Example: Unsigned Char

## Example 7-1

Write a C18 program to send values 00–FF to Port B.

### Solution:

```
#include <P18F458.h>           //for TRISB and PORTB declarations
void main(void)
{
    unsigned char z;
    TRISB = 0;                  //make Port B an output
    for(z=0; z<=255; z++)
        PORTB = z;
    while(1);                   //NEEDED IF RUNNING IN HARDWARE
}
```

Run the above program on your simulator to see how Port B displays values 00–FFH in binary. Notice that “while(1)” is needed if this program is running in hardware.

# Example: Strings – Character Array

## Example 7-2

Write a C18 program to send hex values for ASCII characters of 0, 1, 2, 3, 4, 5, A, B, C, and D to Port B.

### Solution:

```
#include <P18F458.h>
void main(void)
{
    unsigned char mynum[] = "012345ABCD"; //data is stored in RAM
    unsigned char z;
    TRISB = 0; //make Port B an output
    for(z=0; z<10; z++)
        PORTB = mynum[z];
    while(1); //stay here forever
}
```

Run the above program on your simulator to see how Port B displays values 30H, 31H, 32H, 33H, 34H, 35H, 41H, 42H, 43H, and 44H (the hex values for ASCII 0, 1, 2, etc.). Notice that the last statement “while(1)” is needed only if we run the program in hardware. This is like “GOTO \$” or “BRA \$” in Assembly language.



# Example: Signed Char

## Example 7-4

Write a C18 program to send values of -4 to +4 to Port B.

### Solution:

```
//sign numbers
#include <P18F458.h>
void main(void)
{
    char mynum[] = {+1, -1, +2, -2, +3, -3, +4, -4};
    unsigned char z;
    TRISB = 0;                      //make Port B an output
    for(z=0; z<8; z++)
        PORTB = mynum[z];
    while(1);                       //stay here forever
}
```

Run the above program on your simulator to see how PORTB displays values of 1, FFH, 2, FEH, 3, FDH, 4, and FCH (the hex values for +1, -1, +2, -2, etc.). See Chapter 5 for discussion of signed numbers.

# Example: Unsigned short long

## Example 7-6

Write a C18 program to toggle all bits of Port B 100,000 times.

### Solution:

```
//toggle PB 100,00 times
#include <P18F458.h>
void main(void)
{
    unsigned short long z;
    unsigned int x;
    TRISB = 0;                //make Port B an output
    for(z=0;z<=100000;z++)
    {
        PORTB = 0x55;
        PORTB = 0xAA;
    }
    while(1);                 //stay here forever
}
```

# Time Delays in C

---

Two ways to create time delay in C18

- For **loop** – approximate method
- PIC18 Timers - Accurate

## Time delay using for loop

Two factors that can affect the accuracy of delay:

- **Crystal frequency** – connected to OSC1-OSC2 input pins
- **Compiler**- different compilers produce different size hex code

# Time Delays in C

---

Two ways to create time delay in C18

- For **loop** – approximate method
- PIC18 **Timers** - Accurate

## Time delay using for loop

Two factors that can affect the accuracy of delay:

- **Crystal frequency** – connected to OSC1-OSC2 input pins
- **Compiler**- different compilers produce different size hex code
- **NOTE:** For above reasons, when we write time delays in C using loops, we must use the oscilloscope to measure the exact duration.

# Time Delays in C


## Example 7-7

Write a C18 program to toggle all the bits of Port B ports continuously with a 250 ms delay. Assume that the system is PIC18F458 with XTAL = 10 MHz.

### Solution:

```
#include <P18F458.h>
void MSDelay(unsigned int);
void main(void)
{
    TRISB = 0;                //make Port B an output
    while(1)                  //repeat forever
    {
        PORTB = 0x55;
        MSDelay(250);
        PORTB = 0xAA;
        MSDelay(250);
    }
}

void MSDelay(unsigned int itime)
{
    unsigned int i; unsigned char j;
    for(i=0;i<itime;i++)
        for(j=0;j<165;j++);
}
```



# Instruction Cycle – for PIC

---

The amount of time required by the CPU to execute an instruction is referred to as **Instruction cycle** (also known as machine cycle in some other CPUs OR fetch-decode-execute cycle).

- Because all the instructions in the PIC18 are either 2-byte or 4- byte, most instructions take no more than **one or two instruction cycles** to execute.
- The crystal oscillator, along with on-chip circuitry, provide the clock source for the PIC MCU.
- In PIC, one instruction cycle consists of four oscillator periods (clock cycles)
- Therefore, **Instruction cycle =  $\frac{1}{4}$  of crystal frequency**



# Delay calculation for PIC18 – Assembly code

For crystal frequency 4MHz, calculate the delay introduced by following program.

From **Appendix A**, we have the following machine cycles for each instruction of the DELAY subroutine:

## Instruction Cycle

DELAY	MOVLW	0xFA	1
	MOVWF	MYREG	1
AGAIN	NOP		1
	NOP		1
	NOP		1
	DECF	MYREG, F	1
	BNZ	AGAIN	2
	RETURN		1

# Delay calculation for PIC18 – Assembly code

For crystal frequency 4MHz, calculate the delay introduced by following program.

From **Appendix A**, we have the following machine cycles for each instruction of the DELAY subroutine:

*Instruction Cycle*

DELAY	MOVLW	0xFA	1
	MOVWF	MYREG	1
AGAIN	NOP		1
	NOP		1
	NOP		1
	DECF	MYREG, F	1
	BNZ	AGAIN	2
	RETURN		1

Therefore, we have a time delay of  $[(250 \times 6) + 1 + 1 + 1] \times 1 \mu s = 1503 \mu s$ .