

# Junebug Tutorial 1

## Light a LED Part I

### Introduction

At this point it is assumed that you have downloaded and installed MPLAB and the C18 compiler.

To program a PIC using C18 you need information from several sources. Rather than have you hunt for the needed information it has been included here. The source of information is provided to enable you to check the original text if needed.

Many people think of computer and micro controllers as complex devices. The truth is that they are a large collection of simpler things. This series of tutorials attempts to teach the topic from that standpoint.

In this tutorial you will light a LED. The challenge is getting to the point where you are ready to light the LED.

From a hardware viewpoint we configure the PIC18F1320. More importantly we will start using information found in the processor data sheet and processor header file.

From a software viewpoint we will learn some C basics.: We will introduce C variables, functions, and data types.

Have fun.

Abbreviation: uC for micro controller.

MPLAB is a trademark of Microchip.  
Etc,

# Junebug Tutorial 1

## Light a LED Part I

### Program Demo1A.c

In most computer languages it is traditional to call the first program “Hello World” and have it print the string “Hello World”. This is not the case when dealing with uC's, often we will blink an LED. For our first program we will turn on a single LED,

The program demo1A.c is a simple program that lights a single LED. In this tutorial we will examine it in detail.

```
// FILE demo1A.c *** Junebug 18F1320 Static LED demo ***
//
// Turn on LED1
// DIP Switch (SW6) must have TUTOR on (SW6-1,2,3) all other switches off
//
// NOTE: Project requires a linker scrpt.
// Add either 18f1320i.lkr (for debug) or 18f1320.lkr (no debug) to your project.

#pragma config OSC=INTIO2, WDT=OFF, LVP=OFF, DEBUG=ON
#include <p18f1320.h>

void main(void)
{
    ADCON1 = 0x7F; // make RA0 digital

    TRISA = 0b10111110;
    PORTA = 0b00000001; // Turn LED on

    while(1); // loop forever
}
```

The program consists of several parts.

- Comments – tell people about the program and how it works.
- Configuration – tell the compiler what it needs to know about the uC
- Processor Header File – processor specific definitions
- Initialization – setup the chip so that it will work in the way we want it to
- Action – cause the chip to do what we want (turn on the LED)

# Junebug Tutorial 1

## Light a LED Part I

### Comments

The first few lines known as comments and are ignored by the compiler. We write comments to make the program easier to understand. In this case we provide information that will be useful to anyone reading the program. The syntax for comments is simple. When the compiler sees the two characters `//` it ignores them and all the remainder of the text on that line. The compiler colors various parts of code. Notice that MPLAB shows comments in green. This is done to make it easier to read and understand the code.

```
// FILE demo1A.c *** Junebug 18F1320 Static LED demo ***  
//  
// Turn on LED1  
// DIP Switch (SW6) must have TUTOR on (SW6-1,2,3) all other switches off  
//  
// NOTE: Project requires a linker scrpt.  
// Add either 18f1320i.1kr (for debug) or 18f1320.1kr (no debug) to your project.
```

There are two terms you need to know at this point.

### Syntax

All C code must follow the syntax rules that define C. Syntax rules define the grammar of the C language. The syntax for C is complex. The C language syntax is the final authority on C grammar.

We will learn how to program C by understanding example code. We can depend on the compiler to tell us when we have the wrong syntax. It does not do a very good job of telling us how to correct the problem. When in doubt we can consult the syntax.

### Keywords

Keywords or Reserved Words are strings that have special meaning to the C language.

# Junebug Tutorial 1

## Light a LED Part I

### Memory

Definition: variable data that changes during the execution of a program.

PICs in general have several types of memory. We will use the three following memory types.

Memory Type	Written By	Used For
Configuration	Programmer Tool	Processor and peripheral settings
Flash	Programmer Tool	Program code
RAM	Executing program code	Variables

*Table 1.*

It is important to note that the program code created by the compiler is unchanging and stored in flash memory. The data or variables are read and written by the running program and stored in RAM.

# Junebug Tutorial 1

## Light a LED Part I

### Configuration

The `#pragma` keyword is used to pass information to the compiler. It does not cause code or instructions to be generated.

The PIC18F1320 has a special section of memory used to configure the chip. We use “`#pragma config`” to tell the compiler what values to use when programming the configuration memory. The list of available configuration directives for the uC can be found using HELP from the MPLAB main menu.

HELP>TOPICS>LANGUAGE\_TOOLS/PIC18\_CONFIG SETTINGS

```
#pragma config OSC=INTIO2, WDT=OFF, LVP=OFF, DEBUG=ON
```

The config setting for this program from the help file are illustrated in Table 2.

<b>Oscillator Selection:</b>	OSC = INTIO2	Internal RC, OSC1 as RA7, OSC2 as RA6
<b>Watchdog Timer:</b>	WDT = OFF	Disabled
<b>Low Voltage ICSP:</b>	LVP = OFF	Disabled
<b>Background Debugger Enable:</b>	DEBUG = ON	Enabled

*Table 2.*

We are using the internal oscillator, both RA6 and RA7 will be used as IO pins. We do not use the Watchdog Timer or Low Voltage Programming. They are turned off. The Background Debugger will be used. It allows us to use the MPLAB IDE Debugger.

If the configuration seems a bit overwhelming at this point do not get overly concerned. It is enough to know that the provided configuration works for this demo. We will spend more time on the details as required.

# Junebug Tutorial 1

## Light a LED Part I

### Processor Header File

There are header files other than processor header files and we will introduce them as required.

The processor header file is used to describe the processor and peripheral registers. We use the `#include` directive to tell the compiler which processor header file to use. The processor we are using is the PIC18F1320, the processor header file can be found at:

```
C:\MCC18\h\p18F1320.h
```

Our MPLAB is setup such that the compiler knows where to find p18F1320.h. All we need in our program is the line.

```
#include <p18f1320.h>
```

We refer to the processor registers TRISA, and PORTA and the analog to digital configuration register ADCON1, These registers are defined in p18F1320.h. If it were not included the compiler would generate error similar to these.

```
...\3v0\C18_Projects\Junebug1\demo1A.c:14:Error [1105] symbol 'ADCON1' has not been defined
...\3v0\C18_Projects\Junebug1\demo1A.c:14:Error [1101] lvalue required
...\3v0\C18_Projects\Junebug1\demo1A.c:15:Error [1105] symbol 'TRISA' has not been defined
...
```

The header files contains a large number of definitions that we will use in our programs. The definitions used in this program follows.

```
extern volatile near unsigned char    TRISA;
extern volatile near unsigned char    PORTA;
extern volatile near unsigned char    ADCON1;
```

We will learn what these definitions mean in the following section.

# Junebug Tutorial 1

## Light a LED Part I

### Declaring Data

Our example program uses two types of data.

The first is the integer data type **char**. We know integers as counting or whole numbers.

The second is the **void** type. It is the empty data type used where there is no data.

Table 3 shows the data types used in our program.

char	8 bit integer
void	Empty, or without type

*Table 3.*

We declare a variable using the following syntax or form.

```
modifiers type name;
```

The header file declared our three registers as (these variables are names for the registers)

```
extern volatile near unsigned char name;
```

The keyword (modifier) **extern** tells the compiler that it does not need to allocate or assign RAM to the variable. This is because the register already exists in memory. The modifier **volatile** tells the compiler that the variable is subject to sudden change for reasons which cannot be predicted from a study of the program itself. Modifier **near** tells the compiler how to find or address the variable. We will revisit variable declaration in greater detail later.

We have completed the configuration and setup needed to run our simple program. In the next tutorial we will examine the code that turns on the LED.