

1 El Compilador MikroC y los PICMicro

Un microcontrolador es un dispositivo electrónico encapsulado en un circuito de alto nivel de integración. Los microcontroladores se pueden adquirir comercialmente de diferentes casas fabricantes como: Freescale, Motorola, Intel, Philips, y Microchip.

Microchip en particular es una empresa fabricante de dispositivos electrónicos, en sus líneas de producción se encuentran los microcontroladores PICMicro, los cuales se pueden adquirir en diferentes familias, algunas de ellas son: 12F, 16F, 18F, 24F, 30F, y 33F.

En función de la necesidad del proyecto el desarrollador debe escoger la familia y la referencia que más se acerque a su necesidad, por ejemplo el microcontrolador 12F675 es un PIC de 8 pines con módulos integrados básicos como: Timer y ADC. Un microcontrolador como el 16F877 cuenta con 40 pines y módulos como: Timer, ADC, USART, I2C, PWM, entre otros. Fácilmente se pueden apreciar diferencias que permiten crear aplicaciones diferentes entre estos dos ejemplos.



Figura 1-1

La información técnica, la masiva comercialización y la increíble información publicada acerca de los microcontroladores PIC, los hace ideales para aprender y estudiar su funcionamiento, la empresa Microchip cuenta con el portal WEB www.microchip.com en donde se puede descargar información y aplicativos de software que facilitan los desarrollos con sus microcontroladores.

Básicamente implementar un desarrollo con un microcontrolador PIC consiste en identificar la problemática del desarrollo, crear, editar y depurar un programa de máquina y programar eléctricamente el microcontrolador con un programador específico para los PICMicro. Microchip suministra programadores especializados en diferentes escalas, tal vez el más popular es el PICSTART Plus, sin embargo existen otros como el PICKit2, PICKit3. A pesar de que existan programadores comerciales un desarrollador puede construir o adquirir un programador didáctico de bajo costo, de estos últimos existe amplia información publicada en Internet.



Figura 1-2

Un microcontrolador tiene una arquitectura básica que es similar a la de un computador de escritorio, cuenta con un bloque de memoria OTP o Flash en la cual se guardan las instrucciones del programa esta sección es similar al disco duro del computador, el PICMicro cuenta con una memoria RAM, que cumple las mismas funciones de la memoria RAM de un ordenador personal, el microcontrolador posee puertos de entrada y salida que son similares a los periféricos de entrada y salida del computador tales como puertos para el ratón, impresora, monitor, teclado y demás. Estas características hacen que un microcontrolador sea ideal para crear aplicaciones a pequeña escala que tengan interfaz de usuario, adecuando teclados, botones, lectura de memorias de almacenamiento masivo, sensores de diversas variables como: temperatura, humedad, presión, luminosidad, proximidad, entre otros. De igual manera, es posible crear ambientes de visualización con displays numéricos, alfanuméricos y gráficos. Los puertos seriales como la USART y USB permiten crear comunicaciones seriales y comunicaciones inalámbricas con otros dispositivos. En síntesis las posibilidades son infinitas.

1.1 El compilador MikroC PRO

La programación de microcontroladores se basa en un código de máquina que es conocido como código ensamblador, este código contiene una a una las instrucciones del programa, este código ensamblador o también conocido como código assembler es minucioso, y tedioso de editar. El assembler crea códigos de programa extensos y de difícil comprensión. La creación de compiladores de alto nivel facilitó la edición y creación de programas en todo modo de programación lógica, por supuesto los microcontroladores no fueron la excepción, comercialmente existen varios compiladores de diferentes fabricantes y diferentes lenguajes de alto nivel.

Es posible adquirir compiladores como el PICC, CCS, PIC Basic, entre otros. El estudio de este libro se centra en el compilador MikroC PRO, que es un compilador en lenguaje C para microcontroladores PICMicro de la familia 12F, 16F, y 18F.

MikroC PRO es un paquete de software con una amplia variedad de ayudas y herramientas que facilita la creación de proyectos y aplicativos para los microcontroladores PICMicro.

El estudio de este entorno de desarrollo es posible debido a que el estudiante puede descargar una versión demo o estudiantil, que tiene las mismas características de la versión completa, la única limitación es la dimensión del código de máquina que no puede exceder 2K bytes, sin embargo es una capacidad suficiente al tratarse de un primer aprendizaje. La versión demo se puede descargar de la pagina: www.mikroe.com.

En la siguiente figura se puede apreciar la apariencia visual del entorno de desarrollo.

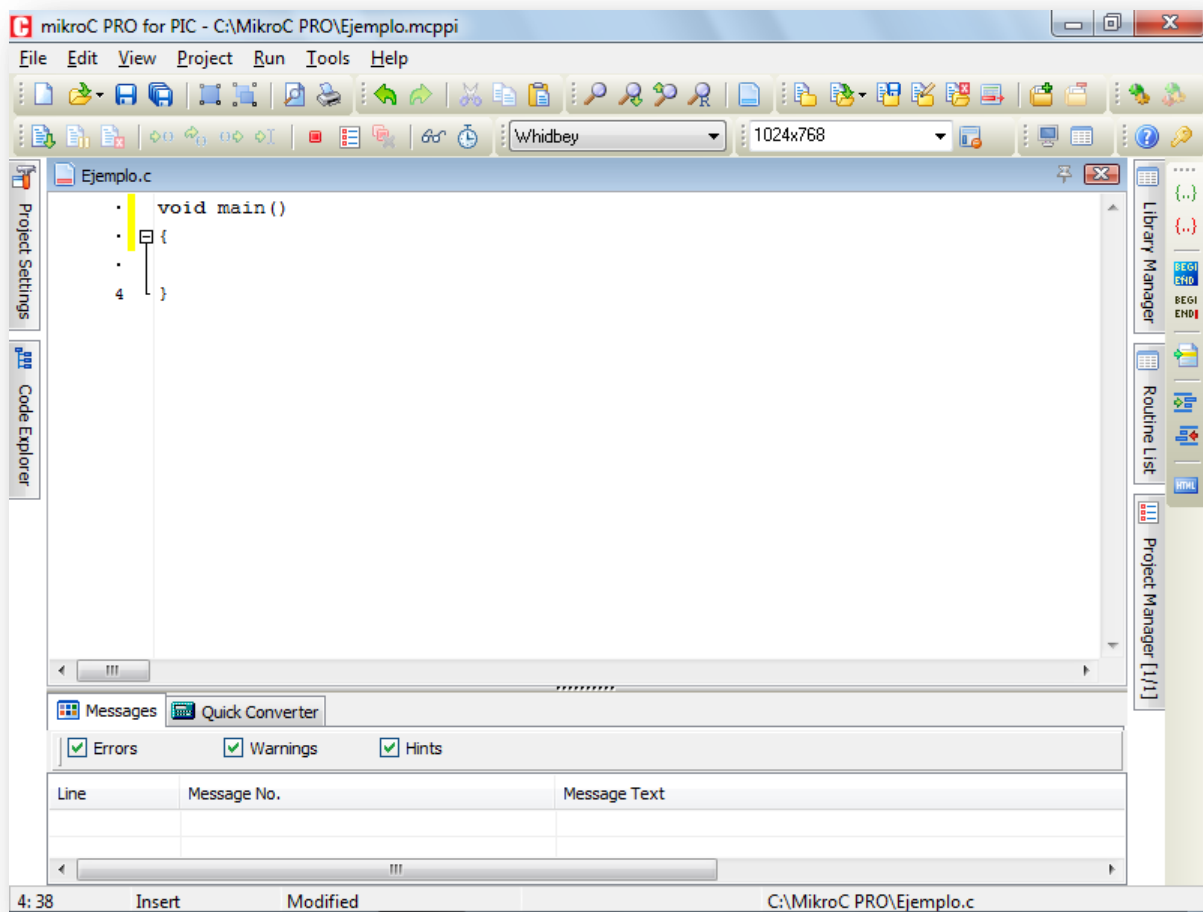


Figura 1-3

El compilador de alto nivel en lenguaje C utiliza estructuras que facilitan la programación, optimiza las operaciones matemáticas y los procesos, por medio del uso de funciones predefinidas y las no predefinidas que el desarrollador puede crear, así como el uso de un conjunto de variables, de tipo carácter, entero, y punto decimal. El compilador crea automáticamente el código ensamblador y a su vez un código similar consignado en un archivo con extensión *.hex, este archivo es el resultado primordial del compilador dado que con este se programa eléctricamente el microcontrolador o con el mismo se puede realizar una simulación computacional.

2 Fundamentos de lenguaje C

El lenguaje C data del año 1972; fue creado por los laboratorios Bell como resultado de la necesidad de reescribir los sistemas operativos UNIX con el fin de optimizar el conocido código ensamblador. De igual manera el lenguaje C fue la evolución de lenguajes previos llamados B, y BCPL. El nuevo lenguaje C, rápidamente tomó fuerza por su funcionalidad y facilidad en la implementación en diversos sistemas computacionales que requerían códigos de máquina.

La forma del lenguaje C, se fundamenta en un complejo estructural que requiere un amplio estudio de las mismas, sin embargo para la programación de los microcontroladores el estudiante requiere una porción fundamental que le permita iniciar y crear los primeros proyectos en MikroC PRO, para este fin el actual capítulo se centra en conocer las nociones necesarias para iniciar el estudio de los microcontroladores con este libro.

2.1 Declaración de variables en lenguaje C

Las variables básicas en este compilador específico son:

- *bit*
- *char*
- *short*
- *int*
- *long*
- *float*
- *double*

Las variables *bit* permiten almacenar un valor lógico es decir verdadero o falso, 0 ó 1.

Las variables *char* se utilizan para almacenar caracteres codificados con el código ASCII, son útiles para guardar letras o textos.

Una variable *short* almacena un número entero de 8 bits corto puede valer de: -127 a 127.

Las variables tipo *int* guardan números enteros de 16 bits, esta variable permite guardar números de: -32767 a 32767.

La variable tipo *long* almacena números enteros largos de 32 bits, su rango puede ser de: -2147483647 a 2147483647.

Las variables tipo *float* y *double* permiten guardar números con punto decimal.

Las anteriores variables pueden declararse incluyendo el signo positivo y negativo, o se pueden declarar por medio de la opción sin signo con la directriz *unsigned*.

En la siguiente tabla se pueden apreciar las características de las variables.

Tipo de variable	Tamaño en Bytes	Valores que soporta
<i>bit</i>	1	0 ó 1
<i>char</i>	1	-127 a 127
<i>short</i>	1	-127 a 127
<i>int</i>	2	- 32767 a 32767
<i>long</i>	4	- 2147483647 a 2147483647
<i>float</i>	4	-1.5x10 ⁴⁵ a 3.4x10 ³⁸
<i>double</i>	4	-1.5x10 ⁴⁵ a 3.4x10 ³⁸
<i>unsigned char</i>	1	0 a 255
<i>unsigned short</i>	1	0 a 255
<i>unsigned int</i>	2	0 a 65535
<i>unsigned long</i>	4	0 a 4294967295

Tabla 2-1

La declaración de variables se realiza indicando el tipo de variable seguido de un nombre que el desarrollador asigna arbitrariamente a la variable. En el punto de la declaración de una variable es posible dar un valor inicial a cada una de las variables sin embargo este último no es estrictamente necesario. Por último la declaración debe culminar con el carácter punto y coma (;).

En los siguientes ejemplos se puede apreciar como se hacen las declaraciones:

```
bit VARIABLE1_BIT;      //Declaración de una variable tipo bit.
char CHARACTER;        //Declaración de una variable tipo char.
char CHARACTER2='J';   //Declaración de una variable tipo char inicializada con el
//valor ASCII del carácter J.
int ENTERO=1234;        //Declaración de una variable tipo entera inicializada con
//el valor 1234.
float DECIMAL=-12.45;   //Declaración de una variable con punto decimal
//inicializada con el valor -12,45.
double DECIMAL2=56.68; //Declaración de una variable con punto decimal
//inicializada con el valor 56,68.
long ENTERO2=-954261;  //Demacración de una variable de tipo entero largo
//inicializada con el valor -954261.
```

Los siguientes ejemplos muestras como declarar variables sin signo:

```
unsigned char CHARACTER; //Declaración de una variable tipo char sin signo.
unsigned int ENTERO;      //Declaración de una variable tipo entera sin signo.
unsigned long ENTERO2;   //Demacración de una variable de tipo entero largo sin signo.
```

Las variables también pueden ser declaradas en un formato que asocia varias variables a un mismo nombre, este formato se conoce como una cadena de variables, o un vector e incluso puede ser una matriz de variables, en conclusión este tipo de declaraciones pueden ser de una o más dimensiones.

El siguiente ejemplo muestra un vector de caracteres, o también conocido como una cadena de caracteres:

```
char Texto[20];      //Cadena de caracteres con 20 posiciones de memoria.
```

De igual manera las cadenas de caracteres o de variables pueden ser declaradas con un valor inicial, este tipo de declaración se puede ver en el siguiente ejemplo:

```
char Texto[20] = "Nuevo Texto"; //Declaración de una cadena de caracteres
//inicializada con el texto: Nuevo Texto.
int Enteros[5]={5,4,3,2,1};      //Declaración de una cadena de enteros con
//valores iniciales.
float Decimales[3]={0.8,1.5,5.8}; //Declaración de una cadena de números con
//punto decimal inicializadas.
```

La declaración de las variables debe respetar algunas reglas básicas que evitan errores y contradicciones en la compilación del código, para este fin tenga presente las siguientes recomendaciones:

- Las variables no deben tener nombres repetidos.
- Las variables no deben empezar por un número.
- Una variable no puede utilizar caracteres especiales como: / * ' ; { } - \ ! . % &.

A continuación se muestran ejemplos de declaraciones de variables que no se pueden hacer:

```
bit 1_VARIABLE-;
char -CARÁCTER!;
int 3ENTERO*;
```

De igual manera es posible crear estructuras de información como un nuevo tipo de variable creada por el desarrollador. Estas estructuras se pueden realizar con las variables ya predefinidas y pueden ser declaraciones de variables o de arreglos de variables. Este tipo de estructuras se declaran, por medio de la directriz: *typedef struct*, y la forma de declarar una variable creada por el desarrollador es la siguiente:

```
typedef struct
{
    char Nombre[10];
    int Edad;
}Usuario;
```

La siguiente es la forma de usar una variable personalizada por el desarrollador:

```
Usuario U;
U.Edad = 25;
U.Nombre[0]='J';
U.Nombre[1]='u';
U.Nombre[2]='a';
```

U.Nombre[3]='n';
U.Nombre[4]=0;

2.2 Formatos numéricos usados en el lenguaje C

Los aplicativos en lenguaje C usan números en diferentes bases numéricas, a pesar de que para el trabajo de bajo nivel del microcontrolador todos sus números están procesados en base 2 es decir en números binarios. El ser humano no está acostumbrado a pensar y procesar operaciones en esta base numérica. Desde las primeras etapas de la formación académica las escuelas y colegios enseñan a pensar y procesar todos los cálculos en base 10, es decir con números decimales. Es por esto que los compiladores en lenguaje C trabajan los números decimales facilitando los diseños para el desarrollador. Además del sistema decimal el compilador en lenguaje C puede trabajar otras bases tales como el binario, y hexadecimal haciendo más simple realizar cálculos y tareas que en decimal serían más complejas. Los sistemas numéricos en base 2, 10, y 16, son los implementados en este compilador en lenguaje C. La escritura de números decimales, es la forma de mayor simplicidad ya que se escriben en lenguaje C de la misma manera convencional que se aprende desde los primeros cursos de matemáticas. Los números binarios se escriben con el encabezado 0b seguidos del número en binario, un ejemplo de esta escritura es: 0b10100001 que es equivalente al número decimal 161. Los números en base 16 o hexadecimales se denotan con el encabezado 0x precedidos de un número en hexadecimal, la expresión 0x2A, es un ejemplo de un número hexadecimal en lenguaje C, y equivale a 42 en decimal. Los números binarios solo pueden tener dos dígitos que son el 0 y el 1. Los números hexadecimales pueden tener 16 dígitos que son; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F.

2.3 Operadores en lenguaje C

El lenguaje C permite hacer operaciones aritméticas o matemáticas básicas entre números contenidos en variables o constantes. Las operaciones aritméticas disponibles son las siguientes:

- Suma
- Resta
- Multiplicación
- División
- Modulo

La suma aritmética entre dos o más números, se puede hacer como se ve en el siguiente ejemplo:

```
int A;  
int B;  
int C;  
C = A+B; //Esta expresión guarda la suma de A y B en la variable C.  
C = A+B+C; //Esta expresión guarda la suma de A, B y C en la variable C.
```

La resta aritmética entre dos o más números, se puede hacer como se ve en el siguiente ejemplo:

```
int A;  
int B;  
int C;  
C = A-B; //Esta expresión guarda la diferencia entre A y B en la variable C.  
C = A-B-C; //Esta expresión guarda la diferencia entre A, B y C en la variable C.
```

La operación matemática de multiplicación se puede realizar entre dos o más números, la operación se relaciona con el carácter asterisco (*), esta expresión se puede apreciar con mayor claridad en los siguientes ejemplos:

```
int A;
```

```
int B;
```

```
int C;
```

```
C = A*B; //Esta expresión guarda la multiplicación entre A y B en la variable C.
```

```
C = A*B*C; //Esta expresión guarda la multiplicación entre A, B y C en la variable C.
```

La división aritmética en lenguaje C se especifica por medio de la barra inclinada (/), en el siguiente ejemplo se puede observar su implementación:

```
int A;
```

```
int B;
```

```
int C;
```

```
C = A/B; //Esta expresión guarda la división A entre B en la variable C.
```

La operación módulo calcula el módulo de una división aritmética es decir calcula el residuo de una división, el cálculo del módulo se denota con el carácter de porcentaje (%), la aplicación de esta operación se puede ver en el siguiente ejemplo:

```
int A;
```

```
int B;
```

```
int C;
```

```
C = A%B; //Esta expresión guarda el residuo de la división de A entre B en la variable C.
```

Las operaciones aritméticas pueden ser usadas en forma combinada, es decir que se pueden mezclar varias operaciones en una misma expresión, para ver esto con mayor claridad observe los siguientes ejemplos:

```
int A;
```

```
int B;
```

```
int C;
```

```
C = (A+B)/C; //Esta expresión es equivalente a C = (A+B)÷C.
```

```
C = (A/B)*C; // Esta expresión es equivalente a C = (A÷B) X C.
```

Otros operadores matemáticos abreviados pueden ser utilizados cuando se requiere de conteos o cambios de una variable de forma constante, por ejemplo es posible incrementar o decrementar una variable en términos de un número, de igual manera es posible hacer esta operación con la multiplicación y la división. Para entender de forma más clara este concepto observe los siguientes ejemplos:

```
int A=100;
```

```
int B=10;
```

```
A++; //Este operador incrementa en una unidad el valor de A.
```

```
A--; //Este operador decrementa en una unidad el valor de A.
```

```
A+=4; //Este operador incrementa en 4 el valor de A.
```

```
A-=5; //Este operador decrementa en 5 el valor de A.
```

```
A/=4; //Este operador divide el valor de A en 4.
```

```
A*=3; //Este operador multiplica el valor de A por 3.
```


$A+=B$; //Este operador incrementa el valor de A en el valor de B unidades.

$A*=B$; //Este operador multiplica el valor de A por B veces.

Otras operaciones matemáticas de mayor complejidad se pueden realizar en lenguaje C, por medio de funciones predefinidas en la librería math, esta temática será tratada posteriormente cuando se estudie el uso y declaración de funciones.

Los operadores lógicos permiten realizar acciones u operaciones que respetan la lógica digital planteada por el matemático inglés George Boole, en el siglo XIX. Boole planteó las operaciones OR, AND, NOT, XOR, NOR, NAND, XNOR. Estas operaciones son ampliamente estudiadas en los cursos de sistemas digitales combinacionales y secuenciales.

Las operaciones lógicas se realizan en lenguaje C entre dos variables o constantes, estas se hacen bit a bit, del mismo peso en una variable o número. Para ver y entender los ejemplos recuerde primero las tablas de verdad de las operaciones lógicas que se muestran a continuación:

<i>Inversor NOT</i>	
<i>Entrada</i>	<i>Salida</i>
0	1
1	0

<i>OR inclusiva</i>		
<i>Entrada 1</i>	<i>Entrada 2</i>	<i>Salida</i>
0	0	0
0	1	1
1	0	1
1	1	1

<i>NOR inclusiva</i>		
<i>Entrada 1</i>	<i>Entrada 2</i>	<i>Salida</i>
0	0	1
0	1	0
1	0	0
1	1	0

<i>OR exclusiva o XOR</i>		
<i>Entrada 1</i>	<i>Entrada 2</i>	<i>Salida</i>
0	0	0
0	1	1
1	0	1
1	1	0

<i>NOR exclusiva o XNOR</i>		
<i>Entrada 1</i>	<i>Entrada 2</i>	<i>Salida</i>
0	0	1
0	1	0
1	0	0
1	1	1

<i>AND</i>		
<i>Entrada 1</i>	<i>Entrada 2</i>	<i>Salida</i>
0	0	0
0	1	0
1	0	0
1	1	1

<i>NAND</i>		
<i>Entrada 1</i>	<i>Entrada 2</i>	<i>Salida</i>
0	0	1
0	1	1
1	0	1
1	1	0

Las operaciones lógicas en lenguaje C, se realizan con caracteres específicos, que denotan cada una de ellas, en el siguiente ejemplo pueden verse las aplicaciones de los operadores lógicos:

Operación lógica NOT, negación, se denota con el carácter virgulilla (~);

```
unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.  
unsigned short RESULTADO;  
RESULTADO = ~VALOR1; //La variable RESULTADO guarda el complemento de  
//VALOR1, 175.
```

Operación lógica OR, o inclusiva, está se denota con el carácter barra vertical (|);

```
unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.  
unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.  
unsigned short RESULTADO;  
RESULTADO = VALOR1|VALOR2; //El valor de la operación lógica o, es guardado en  
//RESULTADO, 95.
```

Operación lógica XOR, o exclusiva, está se denota con el carácter acento circunflejo (^);

```
unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.  
unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.  
unsigned short RESULTADO;  
RESULTADO = VALOR1^VALOR2; //El valor de la operación lógica AND  
//es guardado en RESULTADO, 15.
```

Operación lógica AND, y, está se denota con el carácter ampersand (&);

```
unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.  
unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.  
unsigned short RESULTADO;  
RESULTADO = VALOR1&VALOR2; //El valor de la operación lógica o  
//exclusiva, es guardado en RESULTADO, 80.
```

La implementación de las operaciones lógicas NAND, NOR, XNOR, son similares a AND, OR, y XOR, a agregando el carácter de negación virgulilla, observe los siguientes ejemplos:

```
unsigned short VALOR1=0b01010000; //Variable inicializada en binario con el número 80.  
unsigned short VALOR2=0b01011111; //Variable inicializada en binario con el número 95.  
unsigned short RESULTADO;  
RESULTADO = ~(VALOR1|VALOR2); //El valor de la operación lógica o negada  
// exclusiva, es guardado en RESULTADO.  
RESULTADO = ~(VALOR1&VALOR2); //El valor de la operación lógica y negada,  
// es guardado en RESULTADO.  
RESULTADO = ~(VALOR1^VALOR2); //El valor de la operación lógica o negada  
// exclusiva negada, es guardado en RESULTADO.
```

El desplazamiento de bits dentro de una variable es útil para realizar procesos y tareas que impliquen la manipulación de datos a nivel de los bits. El corrimiento a la derecha en una variable o valor constante, en lenguaje C se realiza por medio del doble carácter mayor que, (>>), de la misma manera el corrimiento a la izquierda se ejecuta con el doble carácter menor que, (<<). La operación de corrimiento hace perder los valores de los bits que salen, e ingresa ceros en

los nuevos bits. En los siguientes ejemplos se puede observar la implementación de estos operadores:

short Dato=0xFF; //Declaración de variables.

short Resultado;

Resultado = Dato>>4; //Está operación guarda en la variable Resultado el corrimiento de 4 bits //a la derecha de la variable Dato, valor final de Resultado es 0x0F.

Resultado = Dato<<4; //Está operación guarda en la variable Resultado el corrimiento de 4 bits //a la izquierda de la variable Dato, valor final de Resultado es 0xF0.

En la siguiente tabla se resumen las operaciones lógicas y aritméticas contempladas anteriormente:

<i>Descripción de la operación</i>	<i>Operador</i>
<i>Suma</i>	+
<i>Restá</i>	-
<i>División</i>	/
<i>Módulo o residuo de la división entera</i>	%
<i>Incremento</i>	++
<i>Decremento</i>	--
<i>Operación OR</i>	
<i>Operación AND</i>	&
<i>Operación XOR</i>	^
<i>Complemento a 1</i>	~
<i>Corrimiento a la derecha</i>	>>
<i>Corrimiento a la izquierda</i>	<<

Tabla 2-2

2.4 Funciones en lenguaje C

Una función es una fracción de código que realiza una tarea específica cada vez que está es invocada por el flujo del programa principal. Las funciones cuentan con dos características fundamentales, uno o más parámetros de entrada, y un parámetro de salida. Cualquiera de estas dos características puede ser una variable o un arreglo de ellas, de igual manera estos parámetros de entrada y salida pueden ser vacíos.

La estructura de las funciones se puede apreciar en los siguientes ejemplos:

void Funcion (**void**) //Función con parámetros de entrada y salida vacíos.

{ //Apertura de la función con corchete.

//Porción de código donde se ejecutan la rutina de la función.

} //Cierre de la función con corchete.

void Funcion (**int** A) //Función con un parámetro de entrada y salida vacía.

{ //Apertura de la función con corchete.

//Porción de código donde se ejecutan la rutina de la función.

} //Cierre de la función con corchete.

```
int Funcion ( void ) //Función con parámetro de entrada vacío y salida entera.
{
    // Apertura de la función con corchete.
    //Porción de código donde se ejecutan la rutina de la función.
}
    //Cierre de la función con corchete.
```

```
int Funcion ( int A ) //Función con un parámetro de entrada y salida enteras.
{
    // Apertura de la función con corchete.
    //Porción de código donde se ejecutan la rutina de la función.
}
    //Cierre de la función con corchete.
```

Los nombres que se designan a las funciones cumplen con las mismas reglas para nombrar las variables. El siguiente ejemplo muestra una función que es capaz de calcular el producto de dos números con punto decimal:

```
float Producto ( float A, float B ) //Función para calcular el producto de A y B.
{
    // Apertura de la función con corchete.
    float RESULTADO;
    RESULTADO = A*B; //Producto de A y B.
    return RESULTADO; //La función retorna el resultado guardado en RESULTADO.
}
    //Cierre de la función con corchete.
```

Una función puede recurrir a otra función para realizar funciones más complejas, para demostrar esta situación se puede ver el siguiente ejemplo, que realiza el cálculo del área de una circunferencia en función de su radio:

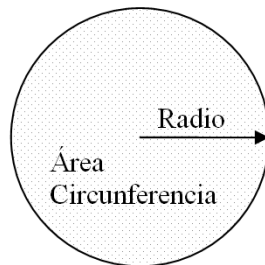


Figura 2-1

El área de la circunferencia se calcula por medio de la ecuación $A=\pi r^2$, donde el valor de π es aproximadamente 3,1416.

```
float Valor_PI ( void ) //Función que retorna el valor de  $\pi$ .
{
    // Apertura de la función con corchete.
    float PI=3.1416;
    return PI;
}
    //Cierre de la función con corchete.
```

En el caso en que una función requiera variables internas esta declaración se debe hacer al principio de la misma, antes de realizar cualquier otra acción o instrucción.

```
float Area_Circunferencia ( float Radio ) //Función para calcular el área del círculo.
{
    // Apertura de la función con corchete.
    float Area;
```

```

    Area = Valor_PI()*Radio*Radio; //Cálculo del área del círculo.
    return Area;
} //Cierre de la función con corchete.

```

Las funciones son creadas por el desarrollador, sin embargo existen algunas que están predefinidas en el compilador y pueden ser consultadas e implementadas respetando los parámetros de entrada y salida, la más importante de ellas es la función *main* o principal. La importancia de esta función radica en el hecho de que es sobre ella que corre todo el programa del microcontrolador y esta se ejecuta automáticamente cuando el microcontrolador se energiza. La función *main* en el caso particular de los microcontroladores no contiene parámetros de entrada ni de salida, su declaración se puede apreciar en el siguiente ejemplo:

```

void main( void ) //Declaración de la función main.
{
    //Apertura de la función main.
    //Código del programa principal del microcontrolador.
} //Cierre de la función main.

```

Las declaraciones de funciones hechas por el desarrollador deben ser declaradas antes de la función *main*, y en el caso de una función que invoque a otra función debe ser declarada antes de la función que hace el llamado.

Las funciones matemáticas trigonométricas y otras como logaritmos, exponenciales, y potenciación pueden ser usadas con la librería predefinida por el compilador. Esta librería se denomina *C_Math*.

2.5 Creación de un programa en lenguaje C

La estructura de un programa en lenguaje C es relativamente simple, primero es indispensable declarar las variables globales que el desarrollador considere necesarias para el funcionamiento del programa, estas variables globales son reconocidas por todos los puntos de código del programa incluidas las funciones propias del desarrollador y la función *main*. El paso a seguir es hacer las declaraciones de funciones diseñadas por el desarrollador para las tareas específicas en su programa. Posteriormente se declara la función *main* y al comienzo de esta se deben declarar las variables que se requieran dentro de la misma. El código que sigue debe configurar e inicializar los puertos y módulos del microcontrolador que sean indispensables en la aplicación. Por último se edita el código que contiene el aplicativo concreto del programa. En el siguiente ejemplo se puede ver cómo hacer una estructura para un programa:

```

int Variable1; //Declaración de variables globales.
char Variable2;
float Variable3;

//Declaración de funciones propias del desarrollador.
float Valor_PI ( void )
{
    float PI=3.1416;
    return PI;
}

```

```

float Calculo_Area_Circulo( float Radio )
{
    float Area;
    Area = Valor_PI()*Radio*Radio; //Cálculo del área.
    return Area;
}

void main( void ) //Declaración de la función main o principal
{
    float Valor_Radio; //Declaración de variables de la función main.
    float Valor_Area;
    TRISB=0; //Configuración de puertos y módulos.
    TRISC=0;
    PORTB=0;
    PORTC=0;
    while( 1 ) //Código concreto del aplicativo.
    {
        Valor_Area=Calculo_Area_Circulo( Valor_Radio );
    }
}

```

2.6 Condicionales e iteraciones en lenguaje C

Las formas condicionales e iterativas hacen parte vital de las estructuras de cualquier código en lenguaje C. Estas permiten hacer ciclos en función de conteos y condiciones de variables que hacen efectivo el flujo del programa. Las sentencias condicionales *if*, *if else*, y *switch case*, permiten realizar menús, tomar decisiones, y controlar el flujo del programa o aplicativo. Las estructuras iterativas *while*, *do while* y *for*, permiten crear bucles iterativos, que cuentan o gobiernan tareas y procesos desarrollados en el programa.

2.6.1 Creación de condiciones lógicas en lenguaje C

Las condiciones lógicas permiten tomar decisiones en función de una evaluación. Las condiciones lógicas solo retornan dos posibles valores: falso o verdadero. Cuando la condición lógica se hace directamente sobre una variable o número, la condición es falsa cuando su valor es 0 y retorna verdadero para cualquier otro valor diferente de 0. Las formas condicionales usadas en lenguaje C son: NOT, o negación, OR o o inclusiva, AND o y. También se pueden usar comparaciones de magnitud entre dos valores tales como: mayor que, menor que, mayor o igual que, menor o igual que, diferente que, e igual que. El uso de la negación se realiza con el carácter admiración (!), la condición OR, se realiza con los caracteres doble barra vertical (||), la condición AND o y, se usa con los caracteres doble ampersand (&&), la condición mayor que se usa con el carácter que tiene su mismo nombre (>), la condición menor que, se hace con el carácter menor que (<), la condición mayor o igual que, se realiza con los caracteres mayor que, e igual (>=), la condición menor o igual que, se realiza con los caracteres menor que, e igual (<=), la condición de igualdad se hace con los caracteres doble igual (==), y la condición diferente que, se usa con los caracteres de igualdad y admiración (!=). Para entender con claridad estos operadores observe y analice los siguientes ejemplos:

```

short A;

```

short B;

//La condición es verdadera si A vale 10 y B vale 20.

(A==10)&&(B==20)

//La condición es verdadera si A es diferente de 5 y B es mayor que 2.

(A!=5)&&(B>2)

//La condición es verdadera si A es menor o igual que 4 o si B es mayor o igual que 6.

(A<=4)&&(B>=6)

//La condición es verdadera si A no es mayor que 3 y si B es diferente de 4.

!(A>3)&&(B!=4)

2.6.2 La sentencia condicional if e if else

La estructura de la sentencia *if* evalúa una condición lógica y ejecuta una porción de código si y solo si el resultado de la evaluación es verdadera. Observe la estructura de la sentencia *if* en el siguiente ejemplo:

short Valor;

if(Valor>100) //Este if evalúa si la variable Valor es mayor que 100.

{

// Porción de código que se ejecuta si el if es verdadero.

}

La estructura *if else* es igual que la sentencia *if*, su única diferencia es que en el dado caso de que la evaluación de la condición sea falsa se ejecuta la porción de código asociada al *else*. Para entender esta situación observe el siguiente ejemplo:

short Valor;

if(Valor>100) //Este if evalúa si la variable Valor es mayor que 100.

{

// Porción de código que se ejecuta si la condición del if es verdadero.

}

else

{

// Porción de código que se ejecuta si la condición del if es falsa.

}

Las sentencias *if* e *if else* se pueden anidar para crear estructuras más completas y complejas, esta característica se puede ver con claridad en el siguiente ejemplo:

short Valor1; *//Declaración de variables.*

short Valor2;

if(Valor1>30) //Sentencia if.

{

// Código que se ejecuta cuando el primer if tiene una condición verdadera.

if((Valor2==4)&&(Valor1<50)) //Segunda sentencia if anidad.

{

//Código que se ejecuta cuando el segundo if tiene una condición verdadera.

}

}

2.6.3 La sentencia switch case

La sentencia *switch case*, funciona de manera similar a la sentencia *if*, difiere en que no se evalúa una condición si no el valor de una variable, y ejecuta una porción de código para cada valor posible de la variable. Los valores contemplados en los casos o *case* posibles de la variable los escoge el desarrollador arbitrariamente en función de su criterio y necesidad. Los casos o valores que no son de interés para el desarrollador se ejecutan en un fragmento de código por defecto, por medio de la directriz *default*. Cada uno de los fragmentos de código editados deben terminar con la directriz *break* para romper el flujo de la estructura *switch case*, esta acción es necesaria dado que si dos casos están seguidos los fragmentos de código seguidos se ejecutarán hasta encontrar la directriz *break* o hasta finalizar la estructura *switch case*. Para comprender de forma clara el funcionamiento de la sentencia *switch case* observe el siguiente ejemplo:

```
short Valor;  
switch( Valor ) //Evaluación de la variable Valor.  
{  
    case 0: //Fragmento de código correspondiente al valor 0 de la variable Valor.  
        break; //Ruptura del caso 0.  
    case 1: //Fragmento de código correspondiente al valor 1 de la variable Valor.  
        break; //Ruptura del caso 1.  
    case 10: //Fragmento de código correspondiente al valor 10 de la variable Valor.  
        break; //Ruptura del caso 10.  
    case 20: //Fragmento de código correspondiente al valor 20 de la variable Valor.  
        break; //Ruptura del caso 20.  
    case 50: //Fragmento de código correspondiente al valor 50 de la variable Valor.  
        break; //Ruptura del caso 50.  
    default: //Código correspondiente para cualquier otro valor por defecto.  
        break; //Ruptura del default.  
}
```

2.6.4 El ciclo iterativo while y do while

El ciclo iterativo *while* repite o ejecuta un fragmento de programa siempre y cuando la condición contenida en el *while* sea verdadera. Para conocer la forma de declarar este tipo de ciclo observe el siguiente ejemplo:

```
short CONT=0; //Declaración de variable entera que cuenta hasta 100.  
while( CONT<100 ) //Declaración del ciclo while.  
{  
    CONT++; //Incremento de la variable CONT.  
}
```

La implementación de un ciclo *do while*, es similar al ciclo *while*, con la diferencia puntual de que en este ciclo el fragmento de código se ejecuta y después evalúa la condición del *while*. En conclusión el código se ejecuta por lo menos una vez antes de evaluar la condición. En el siguiente ejemplo se puede ver la forma de usar este tipo de ciclo:

```
short CONT=0; //Declaración de variable entera que cuenta hasta 100.  
do //Declaración del ciclo do while.  
{
```



```

    CONT++; //Incremento de la variable CONT.
} while( CONT<100 );

```

2.6.5 El ciclo iterativo for

El ciclo iterativo *for* es una estructura parecida al ciclo *while*, la diferencia es una estructura más compleja en los paréntesis de la condición. La condición cuenta con tres parámetros, el primero es un espacio de código para dar valores iniciales a una o más variables, el segundo campo permite crear una condición lógica que hace repetir el fragmento de código del *for* cuando está es verdadera, el último espacio realiza cambio sobre una o más variables, tal como: incrementos, decrementos, etc. Las variables que se cambian en el último campo son generalmente las mismas que se inician en el primer campo, pero está no es una regla. Observe la estructura del *for* en el siguiente modelo:

```

for( _INICIO_ ; _CONDICION_ ; _CAMBIO_ )
{
}

```

Para conocer el funcionamiento del ciclo *for* observe el siguiente ejemplo:

```

short A; //Declaración de variables.
short B;
for( A=0, B=100; (A<100)||(B>20); A++, B-- ) //Estructura del ciclo for.
{
    //Fragmento de código que se ejecuta cuando la condición del for es verdadera.
}

```

2.6.6 Uso anidado de ciclos iterativos

Muchas de las aplicaciones incrementan su funcionalidad y optimizan el tamaño del código de máquina final al usar los ciclos iterativos de forma anidada. La anidación de ciclos consiste en ejecutar un ciclo dentro de otro. Un ciclo puede contener uno o más ciclos anidados. Para comprender este concepto de mejor manera observe el siguiente ejemplo:

```

short COL; //Declaración de variables.
short FIL;
short MATRIZ[10][10]; //Declaración de un vector bidimensional.
for( COL=0; COL<10; COL++ ) //Declaración de ciclo for.
{
    //Fragmento del primer ciclo.
    for( FIL=0; FIL<10; FIL++ ) //Declaración de ciclo for anidado.
    {
        //Fragmento del ciclo anidado.
        MATRIZ[COL][FIL] = COL*FIL;
    }
}

```

El ejemplo anterior, guarda valores en una matriz de 10 por 10, recorriendo una a una las posiciones de la misma.

3 El simulador ISIS de Proteus

El simulador ISIS de Proteus es un poderoso paquete de software, desarrollado por la compañía labcenter electronics, que se ha posicionado desde hace mas de 10 años, como una de las herramientas más útiles para la simulación de los microcontroladores PICMicro. El ISIS permite la simulación de las familias de los PICMicro más populares tales como la: 12F, 16F, 18F. Además de los PIC, el ISIS puede simular una gran variedad de dispositivos digitales y analógicos, entre los dispositivos digitales es posible simular displays de siete segmentos, de caracteres, y gráficos. ISIS puede simular sensores de temperatura, humedad, presión, y luminosidad, entre otros. El simulador permite, simular actuadores como: motores dc, servo motores, luces incandescentes entre otros. Es posible simular periféricos de entrada y salida como teclados, y puertos físicos del ordenador como: RS232, y USB. Este simulador cuenta con una amplia variedad de instrumentos de medición como voltímetros, amperímetros, osciloscopios, y analizadores de señal. En conclusión estás y otras características hacen del ISIS de Proteus, una herramienta ideal para el diseño y estudio de los PICMicro. Una versión demostrativa del paquete de software se puede descargar de la página: www.labcenter.com. En la siguiente imagen se puede apreciar la apariencia visual del entorno de desarrollo del ISIS:

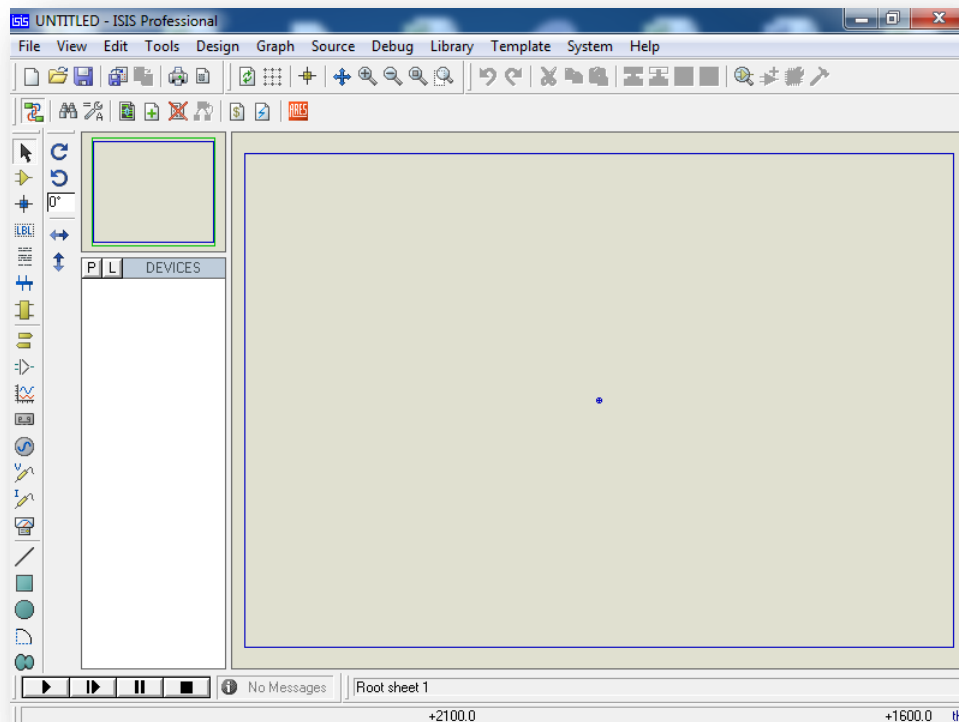


Figura 3-1

El uso y trabajo bajo en el torno de ISIS demanda una amplia cantidad de herramientas y opciones que se deben conocer pero se conocerán paulatinamente en el transcurso de los ejemplos.

3.1 Características básicas del ISIS para simular

En la siguiente sección se mostrarán las características básicas para hacer la primera simulación en ISIS. Como primera medida se debe identificar la paleta de dispositivos, que está a la izquierda de la pantalla, en esta paleta el desarrollador debe identificar el botón P, para mayor claridad observe la siguiente imagen:

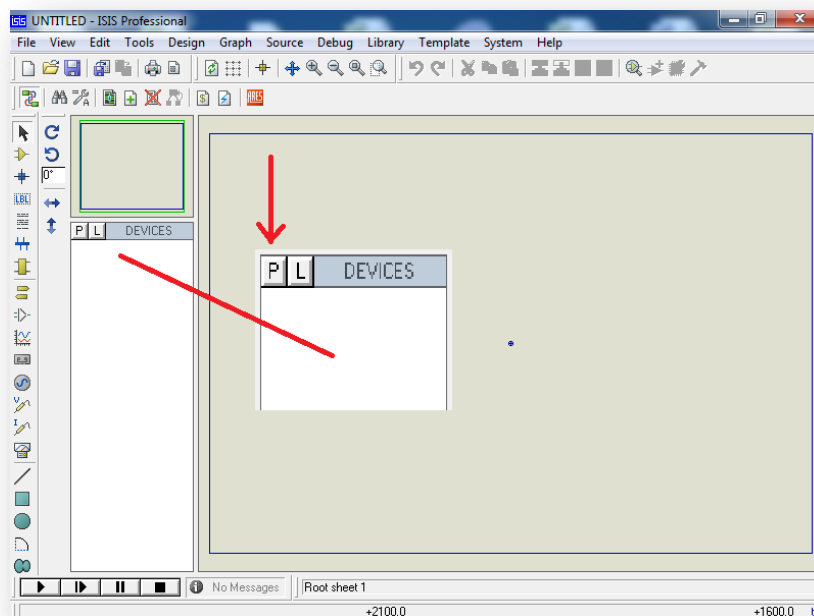


Figura 3-2

Al picar el botón P, el programa abre una nueva ventana que permite buscar los dispositivos electrónicos por medio de la referencia comercial, o bajo la clasificación que el mismo ISIS tiene. Para buscar un dispositivo por medio de la referencia se digita la referencia en la casilla: Keywords, el programa genera una lista en la parte derecha de la ventana con los dispositivos relacionados a la solicitud del usuario. Para seleccionar un dispositivo de esta lista se da doble clic sobre cada uno de los numerales de la lista. Para iniciar busque los siguientes dispositivos: BUTTON, LED-RED, RES, que corresponden a un pulsador, un LED de color rojo, y una resistencia. Después de este proceso, en la paleta de dispositivos debe verse lo siguiente:

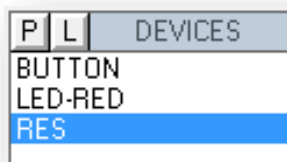



Figura 3-3

El paso siguiente es buscar las terminales de referencia y poder, para este fin se debe pulsar el icono siguiente:  este se encuentra en la paleta de herramientas que está en la parte izquierda de la ventana del programa. Cuando este botón se pica permite ver una lista de terminales, en las

cuales se encuentra GROUND, y POWER, las cuales corresponden respectivamente a la referencia eléctrica y al poder o Vcc. La terminal de poder tiene una diferencia de potencial por defecto de 5 voltios. Para colocar estas terminales en el área de trabajo se pica los ítems en la paleta de terminales y posteriormente en el área de trabajo, después de esta acción en el área de trabajo se debe ver lo siguiente:

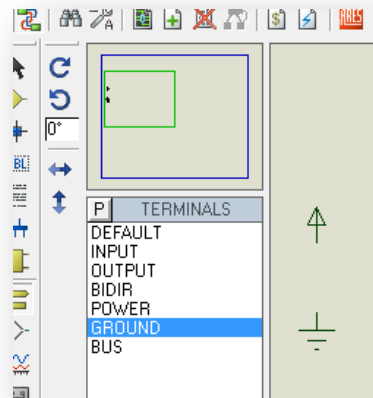



Figura 3-4

El paso siguiente es pegar los dispositivos en el área de trabajo para esto pique el botón:  que se encuentra en la paleta de herramientas de la izquierda. Para pegar los dispositivos en el área de trabajo se sigue el mismo procedimiento de las terminales, al finalizar se debe tener la siguiente vista:

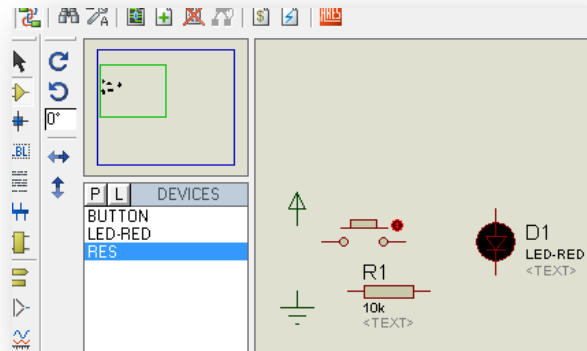


Figura 3-5

El paso siguiente es realizar las conexiones eléctricas, para este ejemplo se conectan todos los elementos en serie, para este fin el cursor del ratón adopta la forma de lápiz, para hacer la conexión entre dos terminales se pica una terminal y después la siguiente. El programa termina rotando la conexión. Al terminar las conexiones se ve la siguiente vista:

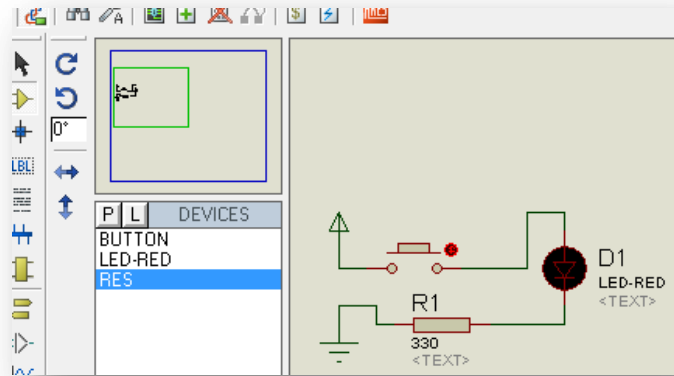


Figura 3-6

Para finalizar se puede cambiar el valor de la resistencia dando clic izquierdo sobre la resistencia, con esta acción sale una ventana que permite editar el valor de la resistencia, que por defecto es de 10k, lo que se debe hacer es cambiarlo por 330.

El paso siguiente es ejecutar o correr la simulación para esto se utiliza la paleta de reproducción que está en la parte inferior izquierda de la pantalla. La apariencia de esta paleta es la siguiente:

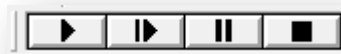


Figura 3-7

Por último se pica el botón de play y la simulación se ejecuta, cuando la simulación esté corriendo se puede pulsar el BUTTON, y ver el efecto de la simulación. Este simple ejemplo permite mecanizar el proceso de creación de un circuito digital, para su posterior simulación.

La creación de circuitos en ISIS, implica hacer otra serie de acciones, que serán tratadas en el transcurso de los ejemplos.

4 Creación del primer programa en MikroC PRO

El proceso siguiente debe ser mecanizado para ser implementado cada vez que se haga un proyecto o programa nuevo para un PICMicro. Al correr el MikroC PRO, se identifica en el menú superior el ítem *Project*, se pica y dentro de este se pica *New Project...*, con esta acción el programa despliega un asistente fácil de usar para crear el nuevo proyecto. La apariencia visual de este asistente es la siguiente:

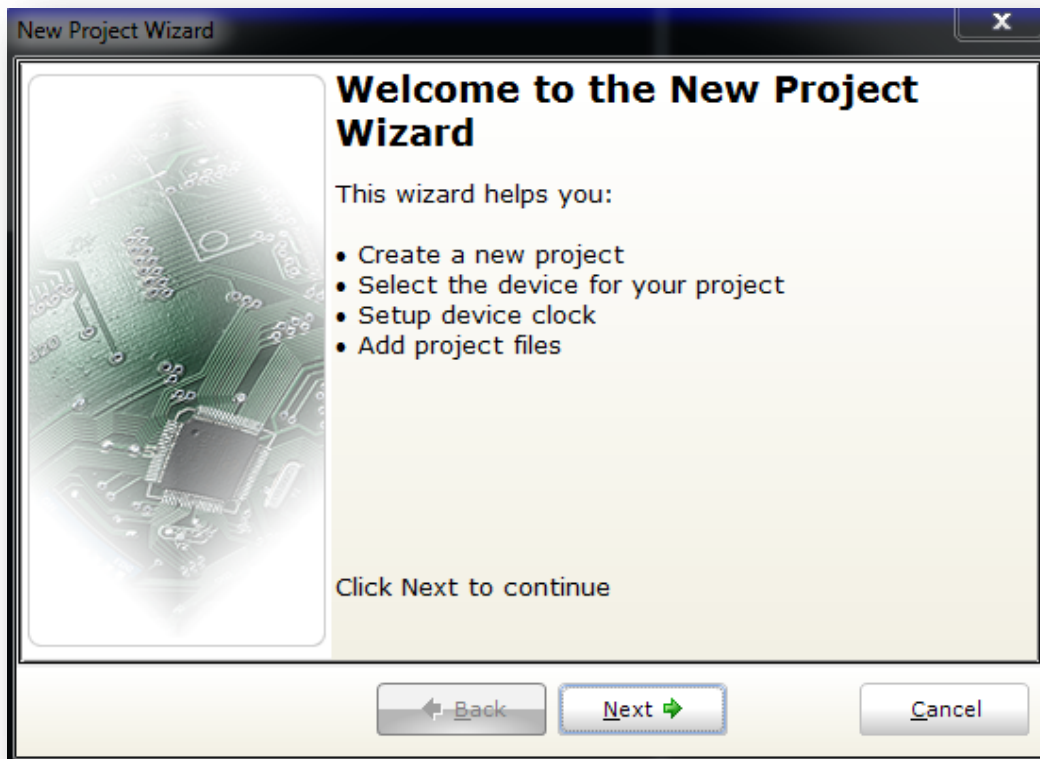


Figura 4-1

La siguiente acción es pulsar el botón *Next*, con este paso el asistente muestra una casilla para seleccionar la referencia del PICMicro, que se desea usar. En esta opción seleccione el PIC P16F84A. El siguiente paso es definir la frecuencia de oscilación con la cual trabajará el PIC, en este ejemplo se selecciona 4.000000 MHz. La siguiente opción permite definir el directorio en donde el desarrollador guardará el proyecto, en este directorio el programa guardará todos los archivos requeridos, en los cuales la fuente del código será el archivo con extensión *.c*, y el ejecutable del PIC es el archivo *.hex*. El siguiente paso solicita adicionar archivos que serán anexados al proyecto. Cuando se realiza un proyecto nuevo este paso se puede omitir y se pulsa *Next*. El último ítem del asistente pregunta si el desarrollador quiere seleccionar las librerías que usará en este trabajo, por defecto este ítem selecciona todas las librerías habilitadas para este PIC,

lo mejor es dejar todas las librerías activas. Por último se termina la configuración y se crea el proyecto, al terminar debe aparecer una vista como la siguiente:

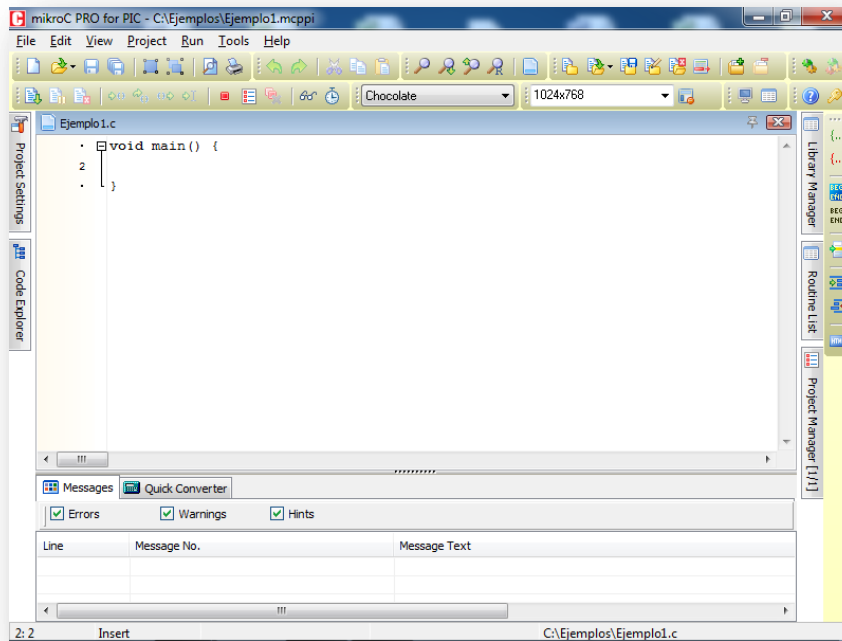



Figura 4-2

Cuando un cambio se realice sobre el código del programa se debe compilar el código picando el siguiente botón:  que está ubicado en la parte superior del programa dentro de la paleta de herramientas. Esta acción genera los resultados de la compilación que se encuentran en la parte inferior de la ventana del programa. Los mensajes se deben terminar con un texto de finalización exitosa.

Para iniciar la edición del proyecto, se configuran los puertos del PIC, y posteriormente se encierra el programa en un bucle infinito. El PIC 16F84A, tiene dos puertos el A y el B, para configurar los puertos como salida o como entrada se manipula el registro TRIS. Cada puerto cuenta con su respectivo registro TRIS, que hace alusión a los tres estados posibles, alto, bajo, y alta impedancia. Los registros TRIS cuentan con el mismo número de bits del puerto, por ejemplo el puerto B o PORTB de este PIC tiene 8 bits, por lo tanto el TRISB también tiene 8 bits. Los bits de los registros TRIS, son correspondientes al puerto, y definen bit a bit el estado del puerto. Si un bit en el TRIS es 0 el mismo bit en el puerto es de salida, y si el bit del TRIS es 1 el mismo bit del puerto es de entrada o está en alta impedancia. Para entender este concepto con mayor claridad observe y analice el siguiente ejemplo:

```
TRISB = 0b11110000; // Configura los cuatro bits de menor peso como salida, y  
//los cuatro bits de mayor peso como entrada.  
PORTB=0b00000000; //Los bits de salida asumen un 0 lógico.
```

Este ejemplo usa un pulsador y un par de LEDs para ver el comportamiento del programa. Observe y analice el programa a continuación:

void main (void)

```

{
    unsigned int CONTADOR=0;
    TRISB = 0b11110000; // Configura los cuatro bits de menor peso como salida, y
    //los cuatro bits de mayor peso como entrada.
    PORTB=0b00000000; //Los bits de salida asumen un 0 lógico.
    while( 1 ) //Bucle infinito
    {
        if( PORTB.F7==0 ) //Evalúa si bit RB7 es 0
        {
            if( PORTB.F0==1 ) //Evalúa el valor del bit RB0 y conmuta su valor.
                PORTB.F0=0;
            else
                PORTB.F0=1;
            while( PORTB.F7==0 ); //Espera a que el RB7 cambie a 1.
        }
        CONTADOR++; //Incrementa el valor del CONTADOR.
        //La siguiente condición if cambia automáticamente el estado del bit RB1
        if( CONTADOR&0x0100 ) //Evalúa si el bit 8 del CONTADOR es 1
            PORTB.F1=1;
        else
            PORTB.F1=0;
    }
}

```

El paso siguiente es hacer la simulación en ISIS, las resistencias de los LEDs deben ser cambiadas a 330Ω, la terminal Master Clear, MCLR debe ser conecta a Vcc para que el PIC, no se reinicie al terminar se debe ver de la siguiente forma:

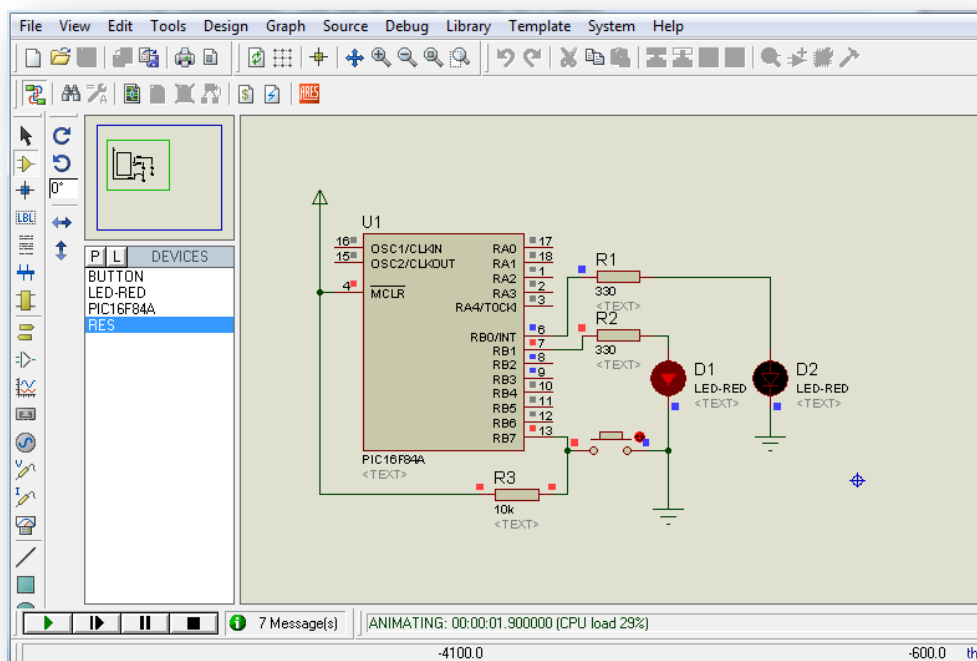


Figura 4-3

Antes de correr la simulación se debe cargar el archivo .hex, para este proceso se hace clic izquierdo sobre el PIC, y aparece una ventana que permite buscar el archivo .hex, en esta ventana también se ajusta la frecuencia de oscilación. Por defecto este valor es 1MHz, para efectos de simulación en este caso se debe seleccionar 4MHz, la vista de esta ventana es la siguiente:

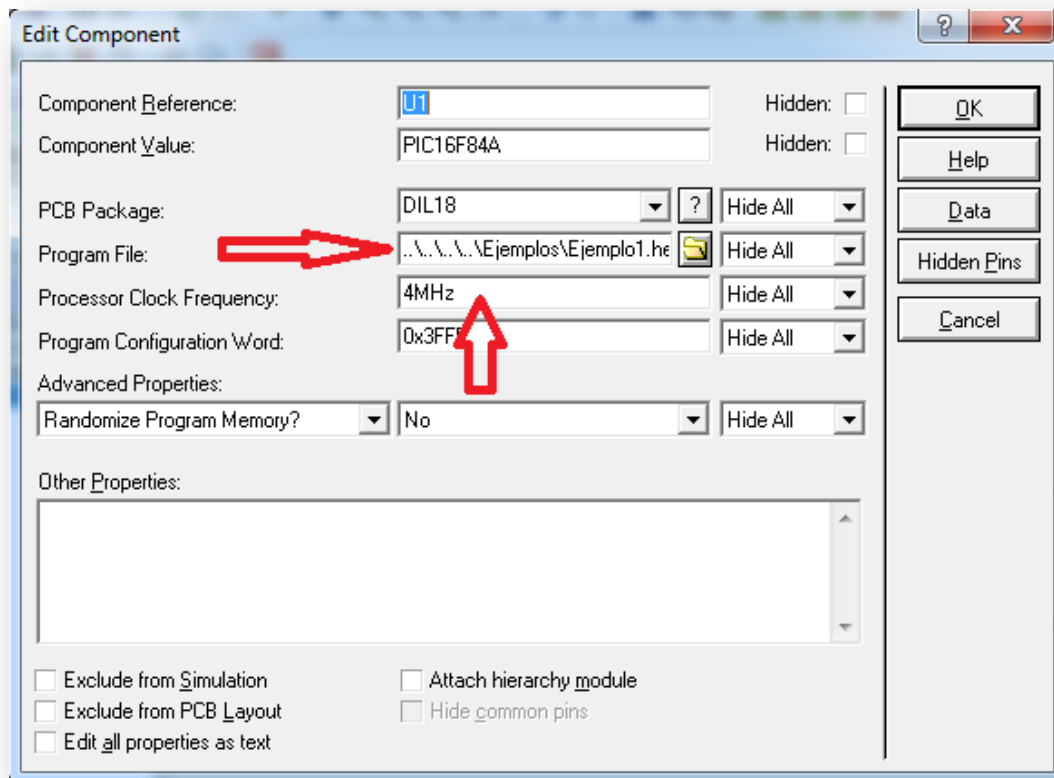


Figura 4-4

Para fines de la programación física del microcontrolador es importante tener presente las condiciones de configuración del PIC, cada uno de los microcontroladores de Microchip, poseen un campo de memoria que caracteriza el comportamiento de real del PIC, entre estos se pueden destacar; el tipo de oscilador o reloj de procesamiento, por ejemplo cuando se usa un cristal de 4M Hz, el oscilador del PIC se debe configurar como XT, si el oscilador es de 20M Hz se usa la configuración HS, si se trata de un cristal de baja velocidad como 400K Hz se usa la configuración LP, y si se implementa un oscilador por medio de circuito RC, o circuito de resistencia en serie con capacitor se usa la opción RC. De la misma forma es posible configurar otras opciones como el uso del perro guardián, la protección de código en memoria FLASH y EEPROM, esta última opción es de suma importancia para proteger la información del microcontrolador y evitar que el programa o los datos de la memoria sean leídos. Para modificar la configuración del PIC, se debe buscar el ítem *Edit Project...* que se encuentra en la pestaña *Project*, del menú principal del programa MikroC PRO.

El acceso a este ítem se puede apreciar en la siguiente figura:

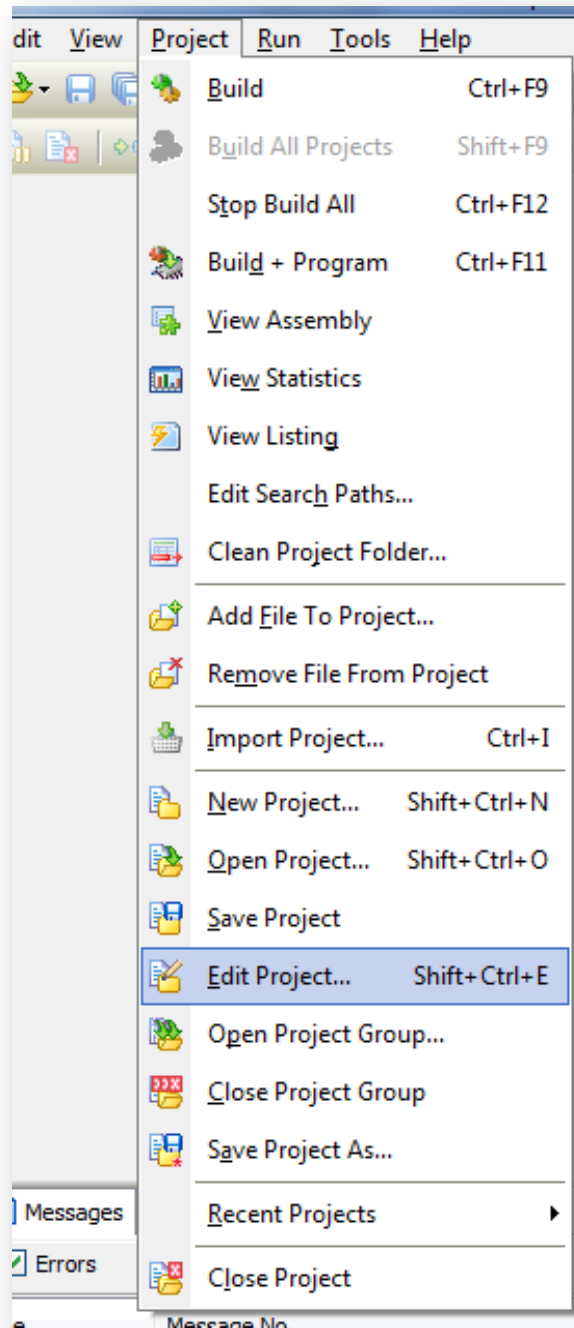


Figura 4-5

Después de picar esta opción el compilador MikroC PRO, despliega una ventana con casillas de selección para editar las opciones de configuración del PIC, en función del microcontrolador esta ventana puede cambiar de apariencia dado que no todos los PIC, cuentan con las mismas configuraciones, sin embargo en la siguiente figura se puede apreciar la apariencia de esta ventana para un PIC 16F877A:

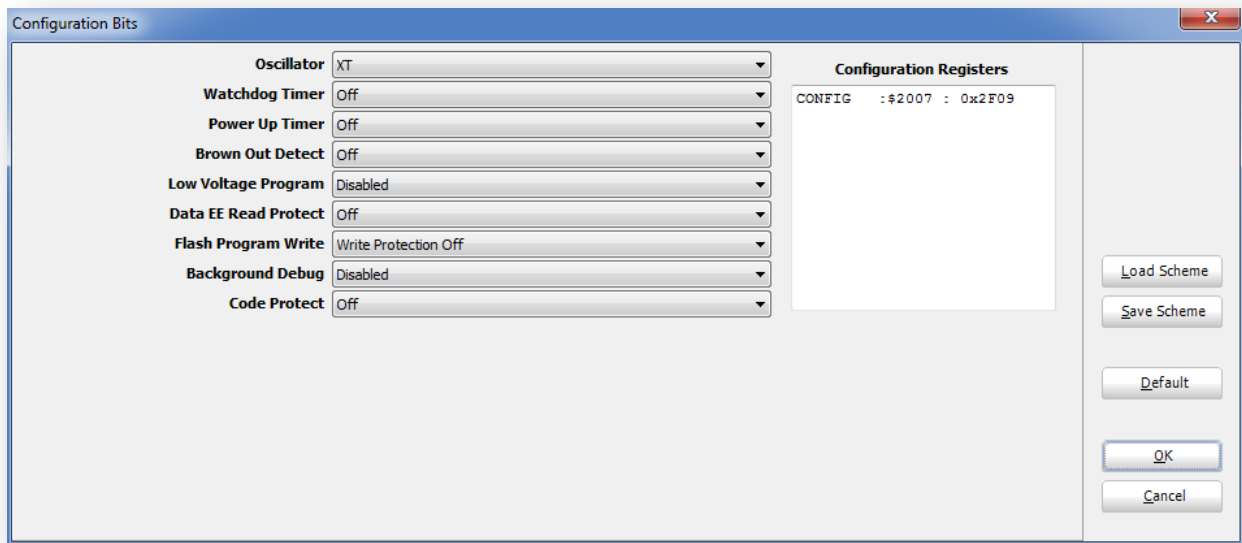


Figura 4-6