

PROGRAMACIÓN DE MICROCONTROLADORES EN C.

Si queremos realizar la programación de los microcontroladores PIC en un lenguaje como el C, es preciso utilizar un compilador de C.

Dicho compilador nos genera ficheros en formato Intel-hexadecimal, que es el necesario para programar (utilizando un programador de PIC) un microcontrolador de 6, 8, 18 ó 40 patillas. El compilador de C que vamos a utilizar es el PCW de la casa CCS Inc. A su vez, el compilador lo integraremos en un entorno de desarrollo integrado (IDE) que nos va a permitir desarrollar todas y cada una de las fases que se compone un proyecto, desde la edición hasta la compilación pasando por la depuración de errores. La última fase, a excepción de la depuración y retoques hardware finales, será programar el PIC.

OPERADORES Y EXPRESIONES

- Operadores de asignación

Una expresión de asignación tradicional es de la forma $\text{expr1} = \text{expr1 operador expr2}$, es decir, $i = i + 5$. Esta expresión se puede representar por otra forma más corta: $\text{expr1 operador} = \text{expr2}$ siguiendo con el mismo ejemplo $i += 5$.

Es en las expresiones complejas, y no en una tan simple como la del ejemplo, donde se puede apreciar la conveniencia de usar esta notación. La siguiente tabla resume los operadores de asignación compuesta y su significado.

Operador	Descripción
<code>+=</code>	Asignación de suma
<code>-=</code>	Asignación de resta
<code>*=</code>	Asignación de multiplicación
<code>/=</code>	Asignación de división
<code>%=</code>	Asignación de resto de división
<code><<=</code>	Asignación de desplazamiento a la izquierda
<code>>>=</code>	Asignación de desplazamiento a la derecha
<code>&=</code>	Asignación de AND de bits
<code> =</code>	Asignación de OR de bits
<code>^=</code>	Asignación de OR exclusivo de bits
<code>~=</code>	Asignación de negación de bits

- Operadores aritméticos

Los operadores aritméticos se usan para realizar operaciones matemáticas. Se listan en la siguiente tabla:

Operador	Descripción	Ejemplo
+	Suma (enteros o reales)	resul = var1 + var2
-	Resta (enteros o reales)	resul = var1 - var2
*	Multiplicación (enteros o reales)	resul = var1 * var2
/	División (enteros o reales)	resul = var1 / var2
-	Cambio de signo en enteros o reales	-var1
%	Módulo; resto de una división entera	rango = n [A1]% 256

- Operadores relacionales

Su misión es comparar dos operandos y dar un resultado entero:
1 (verdadero); 0 (falso).

La siguiente tabla ilustra estos operadores:

Operador	Descripción
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a
!=	Distinto de

- Operadores lógicos

Al igual que los operadores relacionales, éstos devuelven 1 (verdadero), 0 (falso) tras la evaluación de sus operandos. La tabla siguiente ilustra estos operadores.

Operador	Descripción
!	NO lógico
&&	Y lógico
	O lógico

- Operadores de manejo de bits

Estos operadores permiten actuar sobre los operandos a nivel de bits y sólo pueden ser de tipo entero (incluyendo el tipo char). Son los que siguen:

Operador	Descripción
~	Negación de bits (complemento a 1)
&	Y de bits (AND)
^^	O exclusivo de bits (EXOR)
	O de bits (OR)

- Operadores de incremento y decremento

Aunque estos operadores forman parte del grupo de operadores de asignación, he preferido separarlos en aras a una mayor claridad. Su comportamiento se asemeja a las instrucciones de incremento `incf f,d` del ensamblador del μ controlador PIC 16x84 o `inc variable` del Intel 8051.

Operador	Descripción
++	Incremento
--	Decremento

- Operadores de desplazamiento de bits

Los operadores de desplazamiento otorgan al C capacidad de control a bajo nivel similar al lenguaje ensamblador. Estos operadores utilizan dos operandos enteros (tipo `int`): el primero es el elemento a desplazar y el segundo, el número de posiciones de bits que se desplaza. Se resumen en la siguiente tabla:

Operador	Descripción
>>	Desplazamiento a la derecha
<<	Desplazamiento a la izquierda

- Expresiones.

	Constantes
123	Decimal
0123	Octal
0x123	Hex
0b010010	Binario
'x'	Carácter
'\010'	Carácter octal
'\x'	Carácter especial; x puede ser: <code>u,n,t,b,r,f, ', \d,v?</code>
"abcdef"	Cadena (el carácter nulo se agrega al final)

	Identificadores
ABCDE	Hasta 32 caracteres (no puede empezar con números)
ID[X]	Un subíndice
ID[X][X]	Múltiples subíndices
ID.ID	Referencia a una estructura o una unión
ID-->ID	Referencia a una estructura o una unión

INSTRUCCIONES DE CONFIGURACIÓN

```
#include <16F628a.h> /*Se incluye la librería del dispositivo*/
#fuses XT,PUT,NOWDT /*Aquí se configura tales como el perro guardián y, el tipo de reloj*/
#use delay (clock=4M) /*Configuración de la velocidad del reloj*/

#byte port_a=0x05 /*Dirección de configuración puerto a 6 bits*/
#byte port_b=0x06 /*Dirección de configuración puerto b 8 bits “un BYTE”*/

#byte entrada=0xff
#byte salida=0x00
```

FUNCIONES DE I/O DISCRETA

input(pin)	Devuelve el valor de estado (1, 0) de ese pin	in=input(pin_a0);
output_bit(pin, value)	Escribe (1, 0) en el pin correspondiente	output_bit(pin_b0, 1);
output_toggle(pin);	Cambia el estado del pin “complemento”	output_toggle(pin_e1);

FUNCIONES DE RETARDOS

delay_cycles(count);	Demora de 1 a 255 ciclos	delay_cycles(10);
delay_ms(time);	Demora de 1-65535 milisegundos	delay_ms(1000); //1S
delay_us(time);	Demora de 1-65535 microsegundos	delay_us(1000); //1mS

DEFINICIONES DE DATOS

Especificadores de tipo:

Unsigned define un número de 8 bits sin signo
unsigned int define un número de 8 bits sin signo
int define un número de 8 bits sin signo
char define un número de 8 bits sin signo
long define un número de 16 bits sin signo
long int define un número de 16 bits sin signo
signed define un número de 8 bits con signo
signed int define un número de 8 bits con signo
signed long define un número de 16 bits con signo
float define un número de 32 bits en punto flotante
short define un bit
short int define un bit

DEFINICIÓN DE FUNCIÓN

El formato de la definición de una función es como sigue:

```
tipo_de_función función(parámetros de entrada) {  
proceso;  
retorno;  
}
```

donde **tipo_de_función** puede ser void “vacío”, int “intero”, float “flotante”.

Función: en general se identifica con un verbo.

Parámetros de entrada: Se trata de datos de entrada.

proceso: el cómo lo hace para dar la respuesta.

; el punto y coma define el fin de una instrucción.

Retorno: da la respuesta de la función depende del tipo de la función int, float, void etc..

{ }; El ámbito de la función “Hasta donde puede llegar”.

Ejemplo:

//declaración de la función sumar.

```
int dividir(int a, int b){ //Función que puede dividir el número a en b.  
return a/b;           //El retorno de la función “devuelve” la división tipo entera.  
}                     //fin del ambito
```

si necesitáramos dividir el número 5 en 2 entonces se escribe así:

```
a=5;  
b=2;  
dividir(a, b);
```

La respuesta de la división es 2.5 si la hiciéramos nosotros mismos ¿verdad?, pero la función que creamos solo devuelve valores enteros entonces el retorno de la función es 2.

ESTRUCTURA DE UN PROGRAMA EN C

De forma generalizada, la estructura de un programa en C tiene el siguiente aspecto:

```
declaraciones globales  
prototipos de funciones  
main() {  
variables locales;  
bloque de sentencias;  
llamadas a las funciones;  
}  
función_1() {  
variables locales a función_1;  
bloque de sentencias;  
llamada a otra/s funciones;  
}  
función_2(){ }
```

ESTRUCTURAS DE CONTROL

if

```
if(condición ==a){  
    ejecuta esta instrucción;  
}  
else if(condición==b){  
    ejecuta esta instrucción;  
}  
else{  
    ejecuta esta instrucción;  
}
```

switch

```
swieth (parámetro a evaluar){  
    case a:  
        acciones a ejecutar;  
    case b:  
        acciones a ejecutar;  
    default:  
        acciones a ejecutar;  
}
```

while

```
while(condición a cumplir){  
    acciones a ejecutar;  
}
```

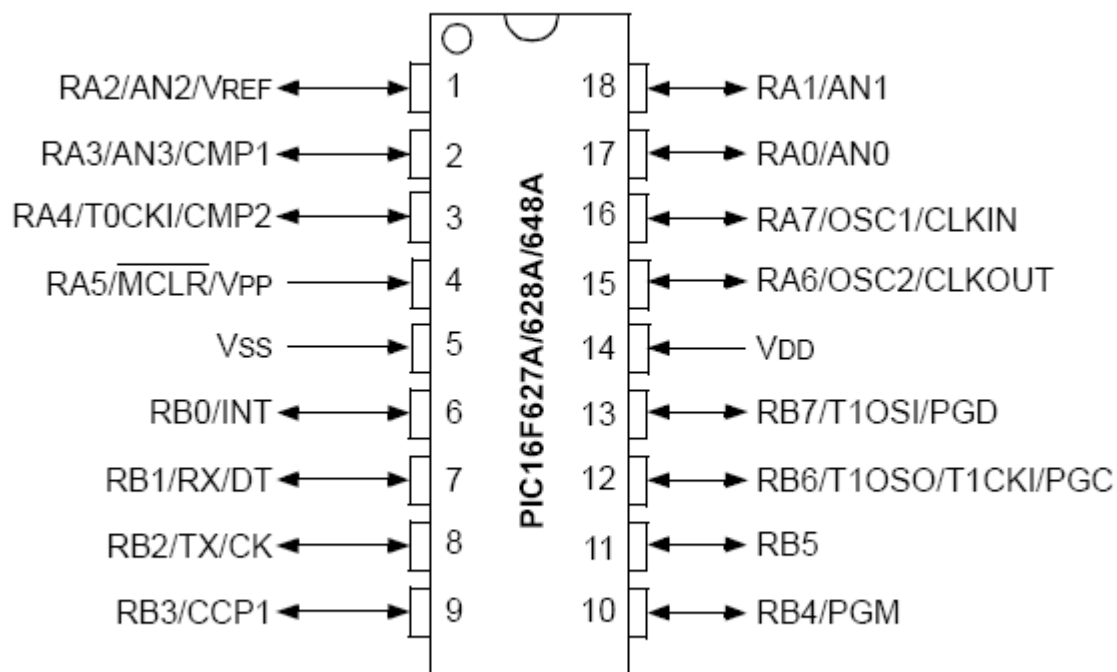
do while

```
do{  
    acciones a ejecutar;  
}  
while(condición a cumplir);
```

for


```
for(asignación inicial, condición; instrucción){  
    bloque de instrucciones;  
}
```

MICROCONTROLADOR PIC16F628A



PUERTO	PIN	BIT
RA0	17	Bit 0 puerto A
RA1	18	Bit 1 puerto A
RA2	1	Bit 2 puerto A
RA3	2	Bit 3 puerto A
RA4	3	Bit 4 puerto A
RA5/reset	4	Bit 5 puerto A
RA6	15	Bit 6 puerto A
RA7	16	Bit 7 puerto A
RB0	6	Bit 0 puerto B
RB1	7	Bit 1 puerto B
RB2	8	Bit 2 puerto B
RB3	9	Bit 3 puerto B
RB4	10	Bit 4 puerto B
RB5	11	Bit 5 puerto B
RB6	12	Bit 6 puerto B
RB7	13	Bit 7 puerto B

								File Address
Indirect addr. ⁽¹⁾	00h	Indirect addr. ⁽¹⁾	80h	Indirect addr. ⁽¹⁾	100h	Indirect addr. ⁽¹⁾	180h	
TMR0	01h	OPTION	81h	TMR0	101h	OPTION	181h	
PCL	02h	PCL	82h	PCL	102h	PCL	182h	
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h	
FSR	04h	FSR	84h	FSR	104h	FSR	184h	
PORTA	05h	TRISA	85h		105h		185h	
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h	
	07h		87h		107h		187h	
	08h		88h		108h		188h	
	09h		89h		109h		189h	
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch	
	0Dh		8Dh		10Dh		18Dh	
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh	
TMR1H	0Fh		8Fh		10Fh		18Fh	
T1CON	10h		90h					
TMR2	11h		91h					
T2CON	12h	PR2	92h					
	13h		93h					
	14h		94h					
CCPR1L	15h		95h					
CCPR1H	16h		96h					
CCP1CON	17h		97h					
RCSTA	18h	TXSTA	98h					
TXREG	19h	SPBRG	99h					
RCREG	1Ah	EEDATA	9Ah					
	1Bh	EEADR	9Bh					
	1Ch	EECON1	9Ch					
	1Dh	EECON2 ⁽¹⁾	9Dh					
	1Eh		9Eh					
CMCON	1Fh	VRCON	9Fh					
	20h		A0h	General Purpose Register 48 Bytes	11Fh			
General Purpose Register 80 Bytes		General Purpose Register 80 Bytes			120h			
					14Fh			
					150h			
	6Fh		EFh		16Fh		1EFh	
16 Bytes	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h	accesses 70h-7Fh	1F0h	
	7Fh		FFh		17Fh		1FFh	
Bank 0		Bank 1		Bank 2		Bank 3		

 Unimplemented data memory locations, read as '0'.
Note 1: Not a physical register.