

Modificado por Luis Frino [www.frino.com.ar](http://www.frino.com.ar)

Fuente [www.micro1.com.ar](http://www.micro1.com.ar)

comprensión de los mismos (por ejemplo, sacar un resultado por una serie de LEDS, en lugar de guardarlos en un registro). Las excepciones son los dos primeros, **suma.asm** y **suma2.asm**, por razones evidentes (tienen muy pocas líneas y son básicos) y **LCD.LIB**, en la que, si bien si fueron corregidos determinados errores, no tenía sentido modificar el programa, ya que gestiona una LCD. Para demostrar nuestra comprensión de la misma se creó **LCDPIC.asm**, inexistente en nuestra bibliografía, y con la que partimos de 0 para su elaboración.

## 6. EL COMPILADOR DE C

### 6.1 Introducción

Prosiguiendo con nuestro estudio sobre las herramientas de programación para la gama media hemos buscado distintas herramientas gratuitas que ofrezcan soporte para el lenguaje C. Microchip se limita a ofrecer tan solo los MPLAB-C17 y MPLAB-C18, limitadas a la gama alta (PIC17CXX y PIC18CXX, respectivamente).

Hemos encontrado varias, pero, entre ellas, quepa tal vez destacar la que aquí estudiaremos, el PIC-C Compiler, al facilitar, además del programa, el código fuente del mismo para posibles posteriores revisiones. Funciona en entorno MS-DOS y es del año 95.

Si bien, como explicamos en su momento, el entorno MPLAB, es del tipo *container*, y carece de lenguajes en su código, dejando la tarea de compilación para otros programas, requiere que estos creen, a su fin, determinados ficheros para atender sus propias necesidades, como la de mostrar errores del programa. Nuestro C es anterior a la creación del MPLAB, y, en consecuencia, no sigue esos criterios, con lo que le resultará imposible incorporarlo.

Su uso es verdaderamente sencillo. Bastará con llamar al programa, que se encuentra en la carpeta Source, seguido de la ruta y nombre del programa a compilar, como, por ejemplo, **pic\_cc prog.c**. La salida del programa es un fichero ensamblador, el cuál, en este caso, se denominaría **prog.s**.

### 6.2 El primer programa en C

Veamos un sencillo programa de los adjuntados en la carpeta **TESTS**, el **test0.c**, el cuál ha sido transcrito sin modificar, pero añadiéndole comentarios:

```
char aa, bb, cc, dd;  ' Creamos cuatro variables de tipo char (caracteres, tamaño 1 byte, lo usual)

main()                ' Proceso principal
{
  char kk;           ' Nuevo char, utilizable solo en main

  aa=3;              ' Damos un par de valores, y calculamos una multiplicación
  bb=5;
  cc=aa*bb;

  kk=cc+1;
  dd=myfunc( kk );  ' Llamada a la función myfunc, con el parámetro kk.
                   ' Su valor de vuelta es asignado a la variable dd
}

myfunc( t )
char t;
{
  char c;

  c=t+4;            ' Calculamos la función deseada
  return( c );     ' Devolvemos el valor c.
}

```

No es nuestro deseo tampoco hacer un estudio profundo del lenguaje c, del que es abundante la bibliografía y con el que pensamos que se hallará familiarizado cualquier lector que se aventure en estas páginas, pero sí hablaremos de sus ventajas, sus desventajas y hasta dónde llega su desarrollo como lenguaje (qué encontraremos en sus librerías).

Por lo pronto, hete aquí la comparación directa con el resultado de la compilación, **test0.s**. Es fácil apreciar lo simple, escueto, legible y fácil de modificar que resulta el código en c.

```
;Small C PIC16C84;
; Coder (1.0 2/10/95)
; Version 0.002

; Front End (PIC Ver 1.0 2/19/95)

include '16c84.h'

; *****code segment cseg*****

org _CSEG

; Begin Function

main_
sub _stackptr, #1
mov _primary, #3
mov aa_ , _primary
mov _primary, #5
mov bb_ , _primary
mov _primary, aa_
call _push_
mov _primary, bb_
call _pop_
call _mul_
mov cc_ , _primary
mov _primary, #0
add _primary, _stackptr
call _push_
mov _primary, cc_
call _push_
mov _primary, #1
call _pop_
add _primary, _secondary
call _pop_
call _putstk_
mov _primary, #0
add _primary, _stackptr
call _indr_
call _push_
; ;(# args passed) mov W, #1
call myfunc_
add _stackptr, #1
mov dd_ , _primary
_1_
add _stackptr, #1
ret

; Begin Function
```

```

myfunc_
  sub _stackptr, #1
  mov _primary, #0
  add _primary, _stackptr
  call _push_
  mov _primary, #2
  add _primary, _stackptr
  call _indr_
  call _push_
  mov _primary, #4
  call _pop_
  add _primary, _secondary
  call _pop_
  call _putstk_
  mov _primary, #0
  add _primary, _stackptr
  call _indr_
  jmp _2_
_2_
  add _stackptr, #1
  ret

;*****need mult/div standard library*****

  include 'mlibpic.h'

; *****data segment dseg*****

  org _DSEG

aa_      ds      1
bb_      ds      1
cc_      ds      1
dd_      ds      1

;0 error(s) in compilation
; literal pool:0
; global pool:112
; Macro pool:51
end

```

Pocos comentarios quedan al respecto. De hecho es prácticamente indescifrable el resultado. Sin embargo, como siempre, una buena programación directa en ensamblador reducirá código, y será de ejecución más rápida. Muchas veces, incluso, resultará imprescindible para determinados módulos. Sin embargo, también necesitará un mayor tiempo de estudio, desarrollo e implementación que en el caso del c.

### 6.3 ¿ Qué podemos usar del c convencional?

Pues, por ejemplo, como apreciaremos del ejemplo **test2.c**, los **punteros**:

```

char val;

main()
{

  char aa,bb,cc,dd;
  load( &aa, &bb, &cc, &dd); ' El carácter & referencia a la dirección de la variable
  val=aa+bb+cc+dd;

```

```
}
```

```
load( a, b, c, d )  
char *a, *b, *c, *d;  
{  
    *a = 1;  
    *b = 2;  
    *c = 3;  
    *d = 4;  
}
```

' Y aquí a, b, c y d son direcciones de memoria, con lo que,  
' para aludir a sus valores, las precedemos de \*.

O, como en el **test1.c**, disfrutaremos de las ventajas de un bucle **while**, o de los operadores lógicos, como *igual*, *distinto*, *>=*, etc.

```
#define TRUE 1  
#define FALSE 0
```

```
char ad, d1, d2, d3;
```

```
main()  
{  
    char cnt;  
    while(TRUE) {  
        ad=GetAD();  
        cnt=0;  
        while (ad>=100) {  
            ad=ad-100;  
            cnt++;  
        }  
        d1=cnt;  
        cnt=0;  
        while (ad>=10) {  
            ad=ad-10;  
            cnt++;  
        }  
        d2=cnt;  
        cnt=0;  
        while (ad>0) {  
            ad=ad-1;  
            cnt++;  
        }  
        d3=cnt;  
    }  
}
```

```
GetAD()  
{  
    char v;  
    v=0xff;  
    return(v);  
}
```

O, como en **test3.c**, de la herramienta **switch**, para escoger entre distintos valores de una variable.

```
char z;
```

```
main()  
{  
    char i;  
    for (i=0; i<5; i++) {  
        switch(i) {
```

```

    case 0: z=f1();
           break;
    case 1: z=f2();
           break;
    case 2: z=f3();
           break;
    case 3: z=f4();
           break;
    case 4: z=f5();
           break;
    }
}
}

```

```

f1() { return(1); }
f2() { return(2); }
f3() { return(3); }
f4() { return(4); }
f5() { return(5); }

```

O, como en el caso de **test4.c**, de la simple sentencia condicional **if**:

```

char z;

main()
{
    char i;
    i=0;
    while(i<5) {
        if (i==0)
            z=f1();
        if (i==1)
            z=f2();
        if (i==2)
            z=f3();
        if (i==3)
            z=f4();
        if (i==4)
            z=f5();
        i++;
    }
}

```

```

f1() { return(1); }
f2() { return(2); }
f3() { return(3); }
f4() { return(4); }
f5() { return(5); }

```

O, como en **test6.c**, del bucle **for**, y de las **cadenas de caracteres** (la sentencia `GetChar` será explicada en el siguiente apartado):

```
/* example of including static strings in compiler */
```

```
char a, b, c, d;
```

```

main() {
    a="hello";
    b="goodbye";
    for (d=0; d<7; d++)
        c=GetChar(b,d); /* index through the string */
    for (d=0; d<5; d++) /* ditto */

```

```

    c=GetChar(a,d);
}

#include "getchar.c"

```

O, incluso, podrá escribir directamente en los puertos, como en **test7.c**. En este caso, hemos vuelto a comentarlo para la más sencilla comprensión:

```

/* test i/o port modes */

#include "io.c"

char a, b;

main()
{
    SetP_A(0x03);          ' Se establece como puerto A la dirección 0x03
    SetP_B(0x0f);          ' Se establece como puerto B la dirección 0x0f
    a=RdPortA();           ' El valor del puerto A se guarda en la variable a
    b=RdPortB();           ' El valor del puerto B se guarda en la variable b
    WrPortA(0x1);          ' Escribimos en el puerto A el valor 0x01
    WrPortB(0x55);         ' Escribimos en el puerto B el valor 0x55
}

```

En realidad es posible asignar como puerto A o B cualquier dirección física de memoria de datos, aunque esta no sea propiamente un puerto. Esto puede ser útil para manejar directamente registros como STATUS o INTCON.

Pero dejamos para el último apartado lo no descrito en los ejemplos, es decir, aquellas funciones incluidas en las librerías.

## 6.4 Librerías y funciones

### 6.4.1 La librería GETCHAR

Esta librería sólo contiene la función GetChar (cadena, índice), que, como pudimos ver en el ejemplo **test6.c**, coge de una cadena de caracteres (definida, por ejemplo, como a = "cadena") el carácter situado en la posición índice. Por ejemplo, b = GetChar (a,2) nos daría b = "d", ya que se empieza a contar desde 0.

### 6.4.2 La librería IO

Esta librería se encarga de la gestión de puertos, y contiene varias funciones:

**SetP\_A(valor)** asigna como dirección del puerto A valor.

**SetP\_B(valor)** asigna como dirección del puerto B valor.

**RdPortA()** y **RdPortB()** devuelven el valor leído en ambos puertos (siempre habrá que definir sus direcciones antes, o tendremos resultados imprevistos).

**WrPortA(valor)** y **WrPortB(valor)**, respectivamente, enviarán valor a las salidas de los puertos A y B. Como en el caso de las anteriores, deben estar previamente definidas sus direcciones para no sufrir imprevistos.

#### 6.4.3 Librería EE\_READ

Contiene la función **ee\_read (addr)**, que leerá un valor de la memoria EEPROM interna del microprocesador, sito en la dirección addr.

#### 6.4.4 Librería EE\_WRITE

Contiene la función **ee\_write (addr)**, que escribirá un valor de la memoria EEPROM interna del microprocesador, sito en la dirección addr.

Estas dos últimas funciones no han sido detalladas en el apartado de ensamblador por no aparecer en la bibliografía asociada. Consultando el Databook en el apartado correspondiente al PIC16C84 se nos mostrará la manera apropiada de hacerlo, a través de los registros EECON1 (dirección 0x08 en la página 0), EECON2 (dirección 0x09 en la página 0), EEDATA (dirección 0x08 en la página 1, 0x88 como dirección absoluta) y EEADR (dirección 0x09 en la página 1, 0x89 como dirección absoluta).

#### **6.4.5 También conviene saber**

Este C asume la directiva **#asm**, a continuación de la cuál seguirá un trozo de código en ensamblador. Para regresar a la programación en C deberemos incluir, tras la secuencia de código, **#endasm**. Puede ver ejemplos editando cualquiera de las librerías, sitas en el directorio **Pic\_lib**. Podrá, a su vez, hacer referencia a los argumentos de la rutina en la que esté el código mediante **#0, #1 ... #n**, siendo el número la posición del argumento en la llamada a la rutina, comenzando por 0.

Las definiciones de constantes en ensamblador están en **Pic\_rt\16c84**. **\_portA** y **\_portB**, por ejemplo, referencian a esos puertos, así como **eedata**, **eeaddr**, **eecon1** y **eecon2**. También contiene macros como **\_push\_**, **\_pop\_**, **\_swap\_**, **\_swaps\_**, o **\_indr\_**, que son usados internamente por el compilador. Puesto que manejan los cuatro registros reservados para el C (estos son **\_primary**, **\_secondary**, **\_temp** y **\_stackptr**) no conviene que los use si no los comprende muy bien, ya que podría alterar el buen funcionamiento de su programa. Las direcciones de esto cuatro registros son 0x2f, 0x2e, 0x2c y 0x2b. Procure no usarlos en su código en ensamblador.

Es posible generar una rutina específica de interrupción. El paso a seguir sería editar el fichero **Pic\_rt\16c84** y alterar la línea:

```
interrupt jmp $ ; set to your interrupt routine
```

Cambiándola por:

```
interrupt call RSI_  
jmp $
```

A partir de ese momento bastará con que cree una rutina llamada RSI() para que sea interpretada como de servicio de interrupción y ejecutada

## 7. EL PROGRAMADOR.

### 7.1 Introducción.

Hemos encontrado una cantidad de programadores importante dispersos por la web y la bibliografía, con diversas complejidades y prestaciones diferentes. Por ello, hemos elegido el programador más sencillo posible pero que mantuviera un nivel de prestaciones alto; con un puñado de componentes poco costosos nuestro programador es capaz de manejar la mayoría de los microcontroladores PIC actuales con una gran facilidad, excluyendo algunas excepciones muy raras.

Aunque no se le puede dar el calificativo de programador de producción, nuestro montaje cumple suficientemente las especificaciones de MICROCHIP para entrar dentro de la categoría que este fabricante llama programadores de desarrollo. En el momento de escribir estas líneas, se pueden programar todos los circuitos PIC de la lista que se proporciona en la Tabla 7.1 pero, teniendo en cuenta su esquema y la evolución constante de su software de control, esta lista es susceptible de evolucionar a medida que se comercializan nuevos circuitos, es más probablemente en el momento en que se lea este texto esta lista ya habrá aumentado. Como muchos de sus homólogos, nuestro montaje se conecta al puerto paralelo del PC, pero no es necesario disponer del último Pentium III a 450 MHz, porque un viejo AT 286 bajo DOS puede ser suficiente, sin tener que renunciar por ello a una excelente facilidad de uso.

0: PIC16F83	18: PIC12CE674	36: PIC16C64	54: PIC16C710
1: PIC16CR83	19: PIC14000	37: PIC16C64A	55: PIC16C71
2: PIC16C84	20: PIC16C54	38: PIC16C64B	56: PIC16C711
3: PIC16F84	21: PIC16C55	39: PIC16CR64	57: PIC16C72
4: PIC16F84A	22: PIC16C56	40: PIC16C65	58: PIC16C73
5: PIC16CR84	23: PIC16C57	41: PIC16C65A	59: PIC16C73A
6: PIC16F873	24: PIC16C57C	42: PIC16C65B	60: PIC16C73B
7: PIC16F874	25: PIC16C58	43: PIC16C66	61: PIC16C74
8: PIC16F876	26: PIC16C554	44: PIC16C67	62: PIC16C74A
9: PIC16F877	27: PIC16C556	45: PIC16C620	63: PIC16C74B
10: PIC12C508	28: PIC16C558	46: PIC16C620A	64: PIC16C76
11: PIC12C508A	29: PIC16C61	47: PIC16C621	65: PIC16C77
12: PIC12C509	30: PIC16C62	48: PIC16C621A	66: PIC16C923
13: PIC12C509A	31: PIC16C62A	49: PIC16C622	67: PIC16C924
14: PIC16C505	32: PIC16C62B	50: PIC16C622B	68: PIC16C642
15: PIC12C671	33: PIC16CR62	51: PIC16CE623	69: PIC16C662
16: PIC12CE673	34: PIC16C63	52: PIC16CE624	70: PIC16C715
17: PIC12C672	35: PIC16C63A	53: PIC16CE625	

Tabla 7.1: Lista de PICs programables.

Nuestro programador necesita, además, una alimentación que puede ser continua o alterna, comprendida entre 12 y 30V y que no es preciso que esté estabilizada. Un enchufe de la red o cualquier alimentación de laboratorio puede ser apropiada, sobre todo teniendo en cuenta que el consumo de corriente es inferior a 100 mA. Como los programadores profesionales, nuestro montaje es capaz, naturalmente, de leer, verificar, programar y comparar los PIC sin ninguna restricción, lo mismo que puede leer y programar sus fusibles de configuración. Por supuesto, también puede borrar los circuitos provistos de memoria de tipo EEPROM y permitir el acceso a la memoria de datos de los circuitos dotados de éstas cuando se realizan, asimismo, con tecnología EEPROM. Ello nos lleva a decir que es verdaderamente completo, por lo que puede satisfacer tanto a un desarrollador ocasional como a un usuario intensivo de circuitos PIC.

## 7.2 De la programación paralela a la programación serie

Mientras que las memorias mas conocidas, las UVPROM, desde la "vieja" 2716 a las más recientes 27512 o 271024, se programan en paralelo (es decir, aplicando simultáneamente a la memoria la dirección a programar y el dato a colocar en esa dirección), numerosas memorias recientes contenidas en los microcontroladores se programan en serie. En el caso de la programación serie sólo se precisan tres lineas de señal, frente a las más de diez necesarias en la programación en paralelo (hasta 26 incluso para una 271024, que es una memoria de 1 megaoctetos y que necesita, por tanto, 8 líneas de datos y 17 líneas de direcciones). La ganancia de espacio es evidente y la simplificación del diseño de circuito impreso que resulta de ello es también enorme.

Los microcontroladores PIC de MICROCHIP están todos provistos de memoria de acceso serie y, como ciertos encapsulados no tienen más que 8 patillas, éstas se reparten según los modos de funcionamiento. La Tabla 7.2 muestra así, en el caso del 12C508 (que es un encapsulado de 8 patillas elegido a título de ejemplo), cómo se realiza la programación respecto a las conexiones del circuito. En la práctica, tres patillas del encapsulado cambian momentáneamente de función durante la fase de programación para dar acceso a la memoria de programa interna; este cambio se desencadena simplemente aplicando la tensión "alta" de programación en la patilla VPP.

<b>Nº DE PATILLA</b>	<b>FUNCIÓN EN MODO NORMAL</b>	<b>FUNCIÓN EN MODO PROGRAMACIÓN</b>
1 (VDD)	Alimentación positiva	Alimentación positiva
2	GP5 o OSC1 o CLKIN	GP5 o OSC1 o CLKIN
3	GP4 o OSC2	GP4 o OSC2
4	GP3 o MCLR	VPP
5	GP2 o TOCKI	GP2 o TOCKI
6	GP1	Reloj de programación
7	GP0	Datos de programación
8 (VSS)	Masa	Masa

Tabla 7.2: Funciones de las patillas del PIC12C508

Aunque las memorias de PIC se programan en serie, nuestro programador se conecta al puerto paralelo del PC. En efecto, por una parte este puerto se puede controlar muy fácilmente por software y, por otra parte, suministra niveles TTL directamente utilizables. Además, debemos disponer de algunas líneas de control para conmutar las diversas alimentaciones del microcontrolador en el curso de la programación, lo cual es mucho más fácil de realizar en un puerto paralelo que en un puerto serie. El esquema completo de nuestro programador se presenta en la Figura 7.1 y vamos a comprobar que se puede analizar fácilmente. Las señales de un puerto paralelo son señales TTL, y por esto, son bastante "maltratadas" en su viaje por los cables de unión un poco largos o de mala calidad. Por esta razón, se restauran un poco por medio de los inversores contenidos en el circuito IC1. Además, como este circuito dispone de salidas a colector abierto, permite controlar fácilmente los tres transistores T1, T2 y T3 que van a continuación. T1 y T2 permiten aplicar la tensión alta de programación VPP a las patillas adecuadas del zócalo universal del programador; patillas que difieren según el tipo de PIC programado. No se puede esperar, en efecto, disponer de la misma asignación de pines en un encapsulado DIL de 8 patillas, que en un DIL de 40. En cuanto al transistor T3, gobierna la tensión normal de alimentación VDD, aplicada igualmente al zócalo universal. Éste permite no alimentar el circuito a programar más que cuando es verdaderamente necesario acceder a él, evitando de esta forma cualquier problema durante su inserción o extracción del zócalo de programación.

Para indicar la aplicación o no de estas tensiones, se utilizan dos LEDs rojos, D1 y D2, gobernados por las dos tensiones VPP. En cuanto al diodo D3, se enciende simplemente cuando el programador está bajo tensión, con el fin de señalar el buen funcionamiento de la alimentación.

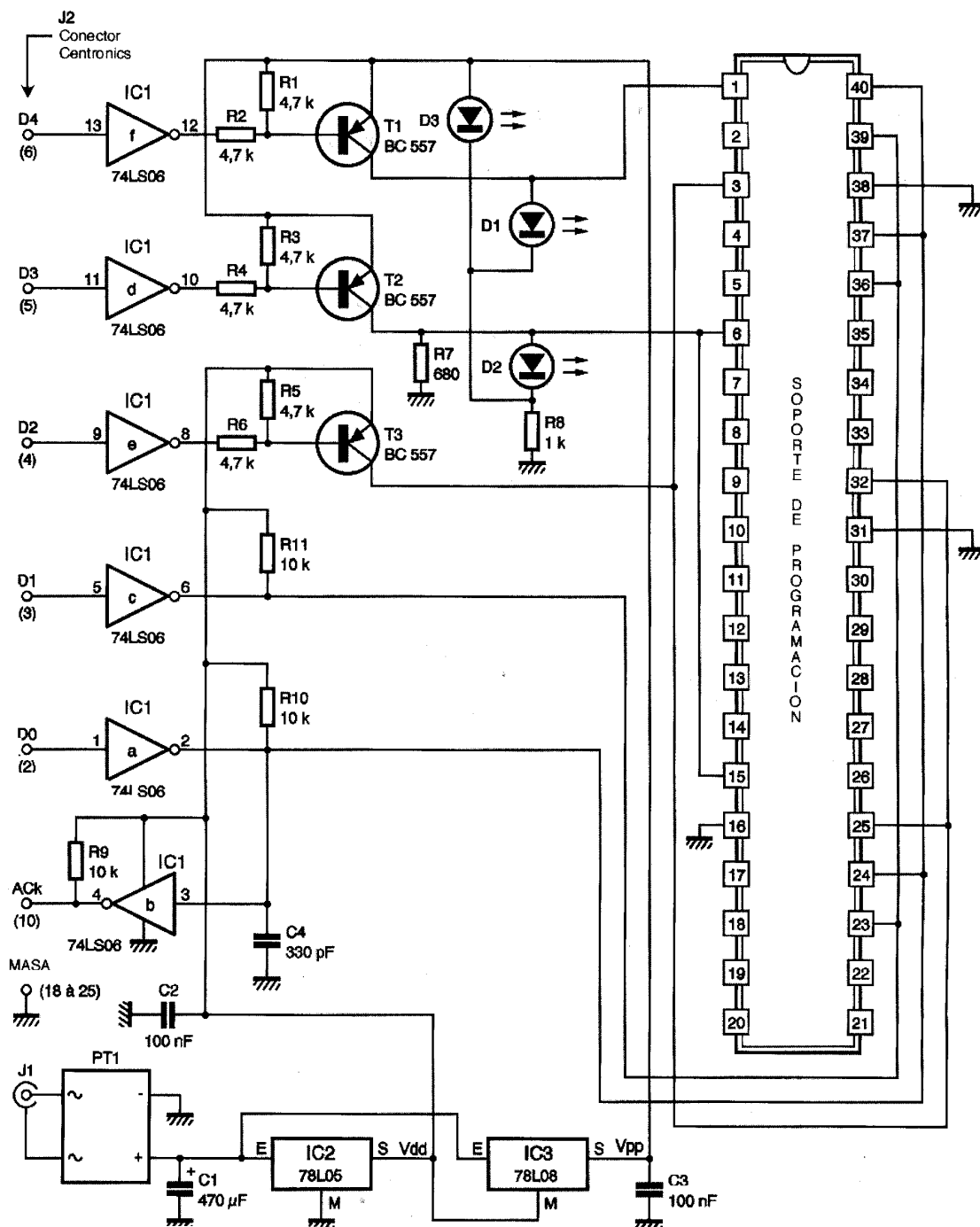


Figura 7.1: Esquema eléctrico del programador

Los datos a programar en el circuito transitan por la puerta IC1a, pasando por IC1b en caso de una segunda lectura del circuito. En cuanto al reloj de programación, pasa por IC1c.

El zócalo destinado a recibir los circuitos a programar es un modelo de 40 patillas un poco especial, como veremos durante la realización práctica, de manera tal que pueda recibir los PIC en encapsulados de 8, 18, 20, 28 y 40 patillas. El cableado de las diferentes alimentaciones, de la línea de datos y de la línea de reloj, se realiza según la asignación de pines de los diferentes circuitos.

La alimentación del programador es muy simple pero muy tolerante. Dos tensiones estabilizadas son, en efecto, necesarias: 5V para la alimentación normal o VDD y 13V para la tensión alta de programación o VPP. IC2, que es un 78L05, se encarga de la producción de 5V, mientras que IC3, que es un 78L08, produce los 13V. Este no está, en efecto, referenciado a masa como se hace normalmente, sino a la salida de IC2, produciendo así  $5 + 8 = 13V$ .

Con el fin de adaptarse a cualquier fuente externa, nuestros reguladores están precedidos por un filtrado generoso y por un puente rectificador. Por esto, se puede aplicar en la entrada J1 cualquier tensión alterna comprendida entre 12 y 20V o cualquier tensión continua comprendida entre 16 y 30V, sabiendo que el consumo de corriente necesario es de aproximadamente 100 mA.

### **7.3 Software y utilización**

Numerosos programas pueden utilizarse con nuestro programador y están disponibles en Internet. Nosotros hemos elegido el programa P16Pro de Bojan Dobaj, que encontrareis en el cdrom adjunto. Este software se adapta perfectamente a nuestro montaje. El programa se llama P16PR363.zip, de donde, una vez descomprimido aparece un cierto número de ficheros, entre ellos el ejecutable P16PRO.EXE

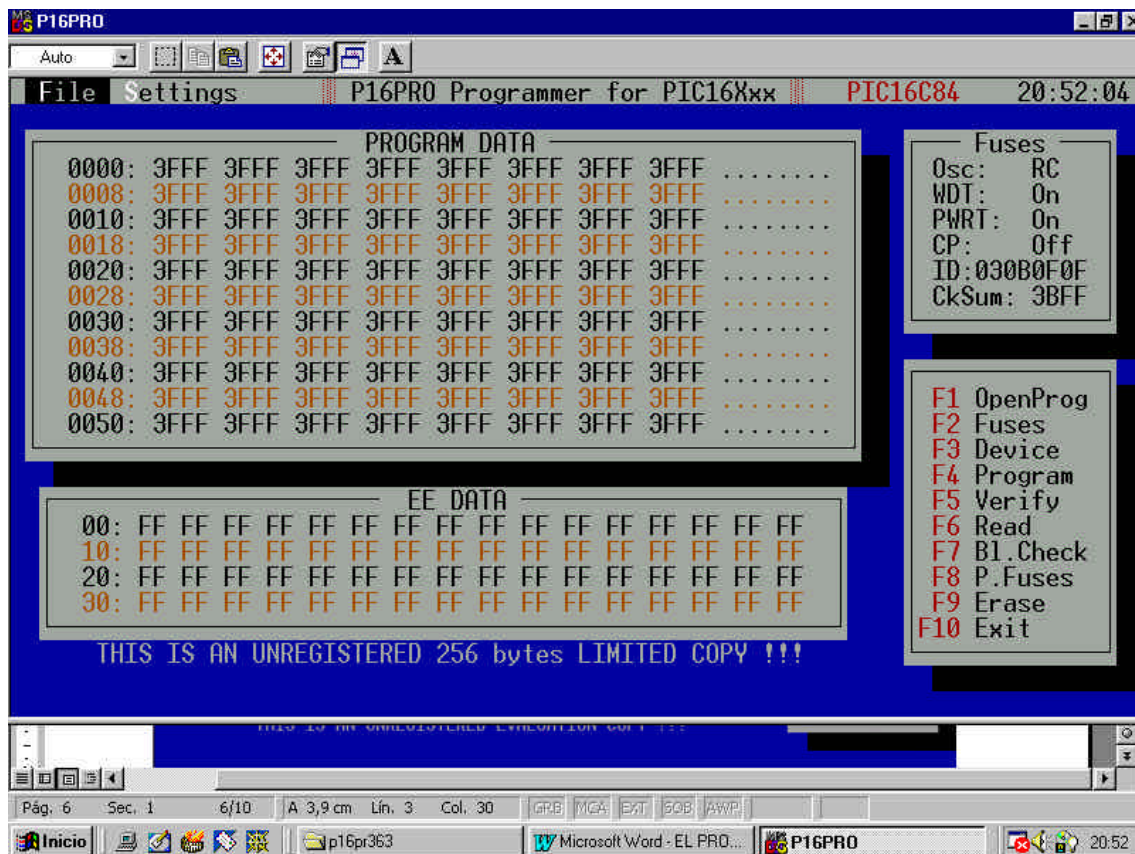
Antes de ejecutar su P16Pro recién instalado, comience por conectar su programador al puerto paralelo del PC. Conecte el programador a una alimentación, y verifique la presencia de 5V en la salida de IC2, y de 13V en la salida de IC3. El LED verde debe estar encendido y los LEDs rojos pueden estar encendidos o apagados según lo que hayan hecho previamente con su puerto paralelo.

Ejecute entonces el programa P16PRO. Tratándose de un programa DOS se ejecuta bajo DOS, en modo DOS si trabajáis con Windows 9X, o bien en una ventana DOS en este último caso.

Cualquiera que sea su modo de ejecución se accede a la pantalla principal reproducida en la Figura 7.2. Esta última muestra, por defecto, tres o cuatro ventanas (dependiendo del tipo de circuito PIC elegido). La ventana principal muestra el contenido de la memoria de programa, la ventana secundaria el estado de los “fusibles” de configuración del circuito y, según el caso, una tercera ventana muestra el contenido de la memoria de datos para los PIC provistos de una EEPROM de datos. Una última ventana, finalmente, resume las principales funciones accesibles de forma inmediata por medio de las teclas F1 a F10.

Figura 7.2: Pantalla de inicio del programador.

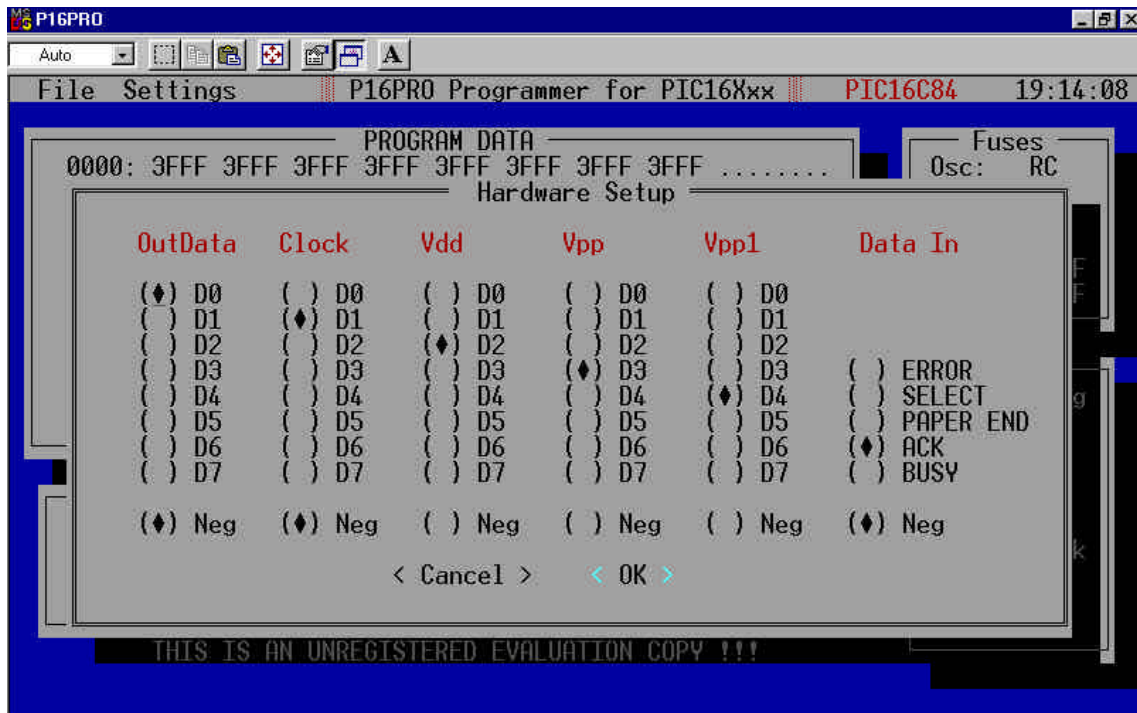
Dos menús desplegables accesibles en la parte superior izquierda de la pantalla, pulsando la tecla Alt, dan acceso a todo lo concerniente a los ficheros (menú FILE) ya a la configuración hardware (menú Settings). Para abrir uno de ellos, basta con pulsar la tecla ENTER cuando el menú elegido se encuentre en vídeo inverso.



La primera operación a realizar consiste en configurar los parámetros en función de su programador. Para esto, vaya al menú Settings y sitúese en Other por medio de las teclas de desplazamiento del cursor (FlechaArriba/FlechaAbajo). Entonces puede usted seleccionar el puerto paralelo al cual se encuentra conectado su montaje. Con este fin, desplace el cursor sobre el nombre del puerto apropiado y hágalo pasar de inactivo a activo por medio de la barra espaciadora. Pulse OK para validar y después vuelva al menú Settings, pero esta vez vaya a Hardware y asegúrese de que aparece una ventana idéntica a la de la Figura 7.3. En caso contrario, modifique su ventana en consecuencia (según el mismo principio que para la elección del puerto), para obtener la misma que nuestra Figura 7.3, a falta de la cual, su programador no podrá funcionar.

Figura 7.3: Configuración del programador.

A continuación, puede usted colocar en el zócalo un circuito PIC seleccionado de entre la lista de circuitos soportados por el programador y respetando las indicaciones de la Figura 7.4, fijarlo en la muesca del zócalo para colocarlo en el sentido correcto. Seleccione la referencia de su circuito por medio de la tecla F3 y luego pruebe las diversas funciones del programador. Si usted utiliza un circuito borrable, por ejemplo, un 16C84 o un 16F84, puede incluso intentar programarlo "para ver", ya que siempre le será posible borrarlo después. Posteriormente, se presenta un resumen de las



instrucciones de uso de este programador, cuyas principales funciones son, sin embargo, suficientemente explícitas, para que usted no encuentre ninguna dificultad en utilizarlas. Si tal fuese el caso, sepa que se encuentra una nota explicativa en inglés dentro del fichero SMANUAL.ENG creado durante la descompresión del fichero inicial.

Un fallo de funcionamiento es muy improbable, vista la simplicidad del montaje; las únicas dificultades que eventualmente podría encontrar son errores de programación o de lectura al utilizar cables de unión de más de 3m entre el PC y el programador. Si, no obstante, se produce dicho fenómeno, que se manifiesta por cambios más o menos aleatorios en los datos leídos o programados, acorte el cable de unión o utilice un cable de mejor calidad.

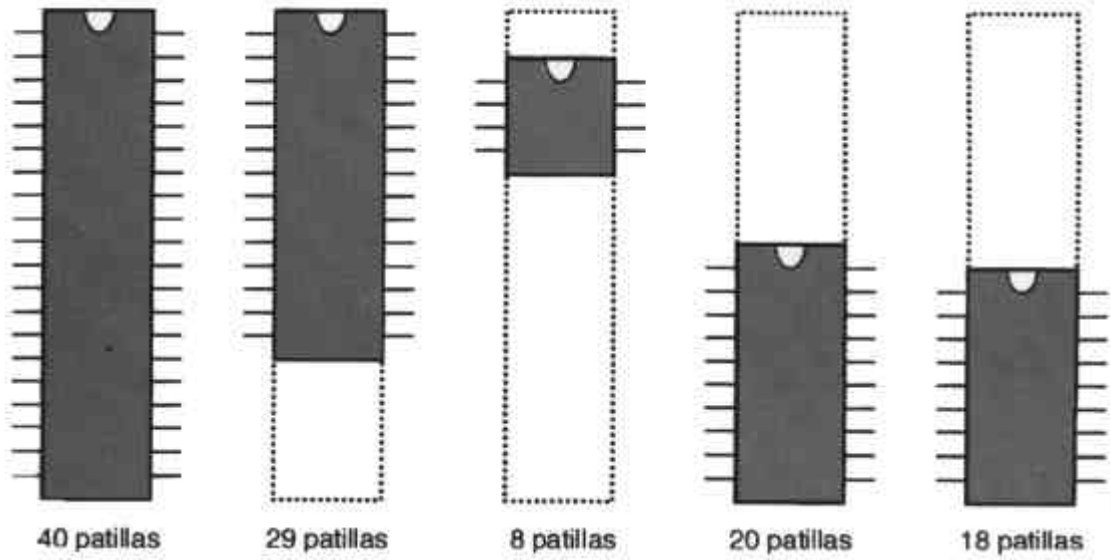


Figura 7.4: Modo de colocar los circuitos integrados en el zócalo universal en función de su tamaño.

#### **7.4 Instrucciones de uso resumidas del programador**

Las teclas F1 a F10 dan acceso directamente a las funciones siguientes:

F1: selecciona el fichero a utilizar en el directorio actual.

F2: da acceso a la ventana de definición de "fusibles" de configuración del circuito a programar, pero no efectúa su programación.

F3: permite elegir el tipo de circuito PIC a programar.

F4: programa el circuito PIC seleccionado con F3, cuyo tipo aparece en la parte superior derecha de la pantalla junto con el fichero previamente seleccionado con F1 o por medio del menú File.

F5: compara el contenido del PIC colocado en el zócalo de programación con el fichero previamente seleccionado con F1 o con el menú File.

F6: lee el contenido del PIC colocado en el zócalo de programación y lo muestra en la ventana o ventanas correspondientes.

F7: comprueba que el PIC no ha sido programado todavía.

F8: programa solamente los fusibles de configuración del PIC conforme al estado definido previamente por medio de F2; estado que se muestra permanentemente en la ventana Fuses.

F9: borra el PIC si éste es borrable (versiones provistas de memoria EEPROM), Si el circuito seleccionado por medio de la tecla F3 no es borrable eléctricamente, este comando se encuentra difuminado e inaccesible.

F10: permite salir del programa.

Los menús File y Settings son accesibles pulsando la tecla Alt del teclado y desplazándose después por medio de las teclas de desplazamiento del cursor (FlechaDerecha/FlechaIzquierda). Para abrirlos hay que pulsar la tecla Enter. Las opciones de menú que aparecen entonces son accesibles, o bien desplazándose por medio de las teclas FlechaAbajo/FlechaArriba, o bien directamente pulsando la letra que se encuentra resaltada (o en vídeo inverso según el modo de presentación). Por ejemplo, la opción Hardware del menú Settings, es accesible pulsando directamente H cuando se encuentra abierto dicho menú. El cierre de una ventana abierta por una de estas opciones de menú puede realizarse situando el cursor sobre Cancel para anular las modificaciones eventualmente hechas por error, o situándose sobre OK para validar las selecciones hechas en la ventana, o bien pulsando Escape.

Cuando una ventana contiene casillas para marcar, como por ejemplo en el caso de la función que permite seleccionar el puerto paralelo, basta con situarse sobre la casilla de su elección por medio de las teclas de desplazamiento del cursor, para poder poner o quitar la marca usando la barra espaciadora.

Habiendo precisado esto, el menú File da acceso a las funciones siguientes:

> Open Program: juega el mismo papel que la tecla F1 y permite cargar en la memoria de programa el contenido del fichero seleccionado. El software reconoce los formatos estándar del ensamblador de MICROCHIP, a saber, **INH8M** o **INH16**.

> Save Program: graba el contenido de la memoria de programa en el fichero de su elección.

> Open Data y Save Data: juegan el mismo papel que los dos comandos que acabamos de ver, pero para los circuitos que contienen memoria EEPROM de datos. En caso contrario, estos comandos están difuminados y son inaccesibles.

> Edit Program: permite editar el contenido de la memoria de programa actual. La dirección y el contenido deseados deben ser introducidos en hexadecimal. Una vez que se ha introducido una dirección, el hecho de pulsar Enter hace pasar automáticamente a la dirección siguiente. Para salir de este modo, basta con pulsar la tecla Escape.

> Fill Program: permite rellenar una zona de memoria con el dato de su elección. Todas las entradas se hacen en hexadecimal.

> Edit Data y Fill Data: juegan el mismo papel que los dos comandos que acabamos de ver pero para los circuitos que contienen memoria EEPROM de datos. En caso contrario, estos comandos están difuminados y son inaccesibles.

> Clear Buffer: repone al estado en blanco (00 o FF según el caso) toda la memoria de programa, así como la memoria EEPROM de datos para los circuitos que la contienen.

> About: muestra el copyright del programa y su número de versión.

> Exit: permite salir del programa, como la tecla F10.

En cuanto al menú Settings, da acceso a las funciones siguientes:

> Device: permite seleccionar el tipo de microcontrolador, como la tecla F3.

> Fuses: permite definir el estado de los fusibles, como la tecla F2.

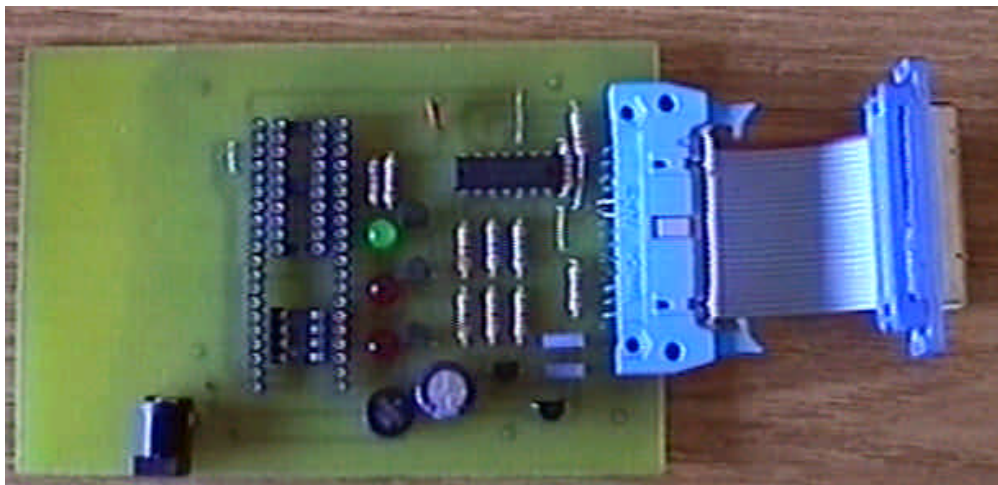
> ID: permite definir el estado de la palabra de identificación contenida en los circuitos PIC. Esta palabra puede ser utilizada como suma de prueba, como número de serie o bien puede ignorarse (a su elección).

> Hardware: da acceso a la ventana de definición de las diferentes líneas de conexión del programador al PC. Esta ventana debe ser rellenada conforme a la que le presentamos en la Figura 7.3 y no debe modificarse o el programador no funcionará.

> Other: permite seleccionar el puerto paralelo utilizado, así como diversas opciones como salvar la configuración al salir del programa, las ventanas de visualización en pantalla, etc.

> Save: permite salvar toda la configuración actual en el fichero P16PRO.INI al salir del programa, incluido el tipo de micro seleccionado, con el fin de poder recuperarla en la siguiente ejecución del programa.

Por último, debajo de estas líneas tenéis como ha quedado nuestro programador:



## 8. APLICACIÓN PRÁCTICA: UN CONTADOR CONTROLADO POR INTERRUPCIÓN

La teoría desarrollada en el presente documento necesita ser apoyada con una demostración práctica por muchos motivos: el primero, personal, es el de demostrar nuestra capacidad ante nuestro tutor para llevar a cabo un diseño real, así como el buen funcionamiento del grabador desarrollado. Sin embargo pensamos que servirá a quien lo lea para comprobar in situ como se "mueve" un PIC y así como para ver un esquema hardware básico sobre el que comenzar a medrar otros posibles diseños.

Para ello hemos decidido hacer una modificación sobre el programa **cuenta.asm**, del punto 6 del apartado de programación, llamado **tablas y subrutinas**, que realizaba una cuenta cíclica de 0 a 9 sobre un 7 segmentos de cátodo común.

En este caso hemos añadido un control por interrupción simulado mediante un pulsador que, al activarse, detendrá la cuenta, y la volverá a cero cuando de suelte el botón.

El problema aparecido es el hecho de que el siete segmentos implementado en esa ocasión controlaba mediante el bit 7 del puerto b el punto decimal, absolutamente innecesario para esta experiencia, mientras que ocupaba el bit 0 del mismo puerto para el *segmento a* del 7 segmentos, que es el único pin disponible para controlar directamente una interrupción externa. Se ha resuelto eliminando el punto decimal y desplazando un bit cada uno de los otros segmento, con lo que observará la tabla de los mismos cambiada.

No cabe destacar más sobre el programa, ya que le suponemos con los conocimientos necesarios, después de leído este manual, como para entender su código, que adjuntamos a continuación. Lo hemos denominado **display.asm**.

```

; *****
; Programa Display.asm
; Contamos hasta 0x5f.
; El valor del contador se visualizará en 8 diodos LED conectados al puerto B
; a partir de la patilla 1, sin gestión de punto decimal
; Preparado para PIC16F84
; Velocidad del reloj: 4 MHz
; Ciclo de instrucción: 1 MHz = 1 microsegundo
; Interrupciones: A través de PB.0, para detener y recomenzar la cuenta.
; Perro guardián: Desactivado
; Tipo de Reloj: XT
; Protección del código: Desactivado
; *****

LIST P = 16F84          ;Indicamos el modelo de PIC a utilizar

; Definición de registros

portb    EQU    0x06    ;Hemos conectado el teclado al puerto B
                    ;La dirección 0x06 corresponde al registro PORTB (puerto B)
                    ;                               en el banco1
TRISB    EQU    0X06    ;                               y TRISB en banco 1
estado   EQU    0X03    ; La dirección del registro de estado es la 0x03
pc       EQU    0x02    ; Contador de Programa, dirección de memoria actual de programa
intcon   EQU    0x0B    ; Registro gestor de interrupciones
opcion   EQU    0x01    ; Registro OPTION. Recordar que está en el banco 1.

; Definición de bits

banco    EQU    0X05    ; Bit del registro de estado correspondiente al banco de datos
Z        EQU    0X02    ; Bit indicador de que el registro W está a cero
int      EQU    0x00    ; Bit de interrupción externa, es el 0 en el puerto B.
intdeg   EQU    0x06    ; Bit 6 de OPTION, que indica si la interrupción PBO es por nivel alto.
intf     EQU    0x01    ; Bit 1 de INTCON, flag de interrupción por PBO.
inte     EQU    0x04    ; Bit 4 de INTCON, habilitador de interrupción por PBO.
GIE      EQU    0x07    ; Bit 7 de INTCON, habilitador de interrupciones.

; Definición de constantes

w        EQU    0      ; Destino de operación = w
f        EQU    1      ; Destino de operación = registro

; Definición de variables

contador EQU    0X0C    ; Contador
digito   EQU    0X0D    ; Para almacenar el dígito

; Comienzo del programa.

ORG 0X00          ; Cubrimos el vector de reset

        GOTO inicio  ; Saltamos a la primera dirección tras el vector de interrupción

ORG 0x04          ; Vector de interrupción

        GOTO RSI

```

; \*\*\*\*\* Inicialización de variables \*\*\*\*\*

ORG 0X05

inicio BSF estado,banco ; Cambiamos a la segunda página de memoria  
CLRFB TRISB ; Programa la puerta B como de todo salidas  
BSF TRISB,int ; Salvo la pata de interrupción PBO, que es de entrada  
BSF opcion,intdeg ; Interrupción PBO cuando esté a nivel alto.  
BCF estado,banco ; Volvemos a la página 0.  
BCF intcon,intf ; Borramos el flag de interrupción por PBO.  
BSF intcon,GIE ; Habilitamos las interrupciones.  
BSF intcon,inte ; Habilitamos la interrupción por PBO.  
CLRFB portb ; Apaga el display, por si había residuos  
CLRFB contador ; Borra el contador (dirección 0x0C)  
CLRFB ; Borramos el registro W

; \*\*\*\*\* Cuerpo Principal \*\*\*\*\*

Reset CLRFB digito ; Comienza a contar por el 0

Siguien MOVFB digito,w ; Coloca el siguiente dígito a evaluar en W  
CALL Tabla ; Llama a la subrutina para coger el dato  
; y hacer la conversión decimal-7 segmentos  
MOVFB portb

Pausa DECFBSZ contador ; Decrementa contador y repite  
GOTO Pausa ; hasta que valga 0  
INCF digito,f ; Incrementa el valor del dígito al siguiente  
MOVFB digito,w ; Pone el valor del dígito en W  
XORLW 0x0A ; Chekea si el dígito sobrepasa el valor 9  
BTFSS estado,Z ; Comprobando con un xor si W vale 0 (Z=1)  
GOTO Reset ; Si Z=1 resetea el dígito y comienza de nuevo la cuenta  
GOTO Siguien ; En caso contrario, continua la cuenta

; \*\*\*\*\* La tabla queda definida aquí \*\*\*\*\*

Tabla ADDFB pc,f ; Suma al contador de programa el valor de offset, es decir,  
; el valor del dígito. Así se genera un PC distinto  
; según su valor,  
; asegurando que vaya a la línea correcta de la tabla  
RETLW 0x7F ; 0 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0x0C ; 1 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0xB6 ; 2 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0x9F ; 3 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0xCC ; 4 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0xDA ; 5 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0xFA ; 6 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0x0F ; 7 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0xFF ; 8 en código 7 segmentos (desplazado a la izquierda)  
RETLW 0xDF ; 9 en código 7 segmentos (desplazado a la izquierda)

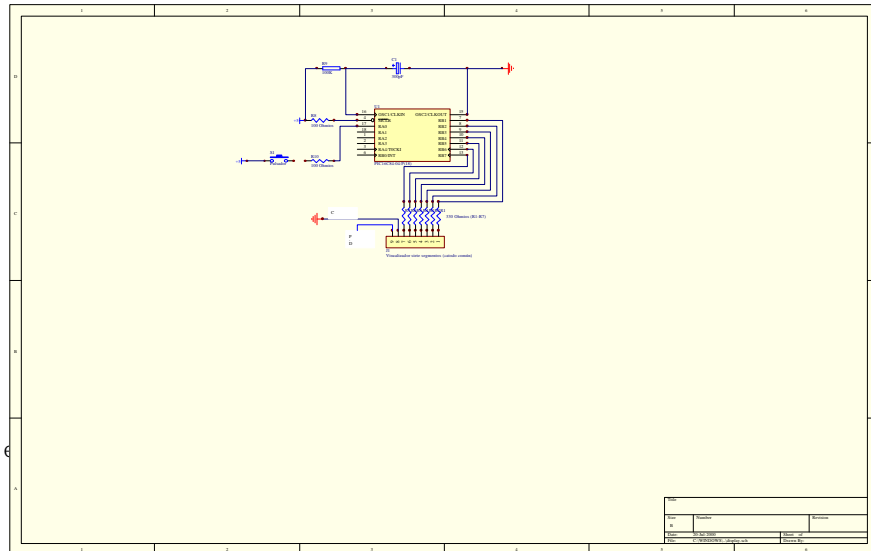
RSI BTFSS intcon,intf ; Si no es interrumpido por PBO, volver al programa  
RETFIE

pulsado BTFSS portb,0 ; Retenemos hasta que se suelte el pulsador  
GOTO pulsado  
MOVLW 0xFF ; Puesto que se habrá de incrementar  
MOVFB digito ; Ponemos el marcador a FF  
BCF intcon,intf ; Borramos la bandera de interrupción  
BSF intcon,inte ; Y rehabilitamos la interrupción por PBO  
RETFIE

END

El esquema electrónico del visualizador lo tenéis a continuación, pero hemos tenido problemas para su correcta visualización desde Word por lo que podéis ampliar la imagen siguiente o podéis acudir al CD adjunto donde se encuentra el esquema y una copia de evaluación de Protel 98, un programa para la creación de circuitos:

Finalmente este e



## 9. BIBLIOGRAFÍA

### 9.1 Bibliografía escrita.

**Microcontroladores PIC. La Solución en un Chip.**

*J. M<sup>a</sup>. Angulo Usategui, E. Martín Cuenca, I. Angulo Martínez*  
Ed. Paraninfo. ( 1997 )

**Microcontroladores PIC. Diseño práctico de aplicaciones.**

*J. M<sup>a</sup>. Angulo Usategui, I. Angulo Martínez*  
Mc Graw Hill [ 1999 ]

**Microcontroladores.**

Vicente Torres.  
Servicio Publicaciones UPV.

**Programming and Customizing the Pic Microcontroller**

Myke Predko  
Mc Graw Hill [ 1999 ]

**Electrónica. Microcontroladores y Microprocesadores.**

Fascículos coleccionables. Editorial Multipress SA.

**PIC16/17 Microcontroller Data Book.**

Microchip Technology Inc. ( 1995 – 96 )

**Technical Training Workbook de Microchip**

Microchip Technology Inc. ( 1999 )

**Embedded Control Handbook**

Microchip Technology Inc. (1995 - 96 )

**July 1999 Technical Library CD-ROM**

Microchip Technology Inc. ( 1999 )

**Microchip Technical CD-ROM First Edition 2000**

Microchip Technology Inc. ( 2000 )

**MPSIM Simulator Quick Reference Guide**

Microchip Technology Inc. ( 1996 )

**MPASM Assembler Quick Reference Guide**

Microchip Technology Inc. ( 1996 )

**MPSIM Simulator User's guide**

Microchip Technology Inc. ( 1995 )

**MPASM Assembler User's Guide**

Microchip Technology Inc. ( 1995 )

