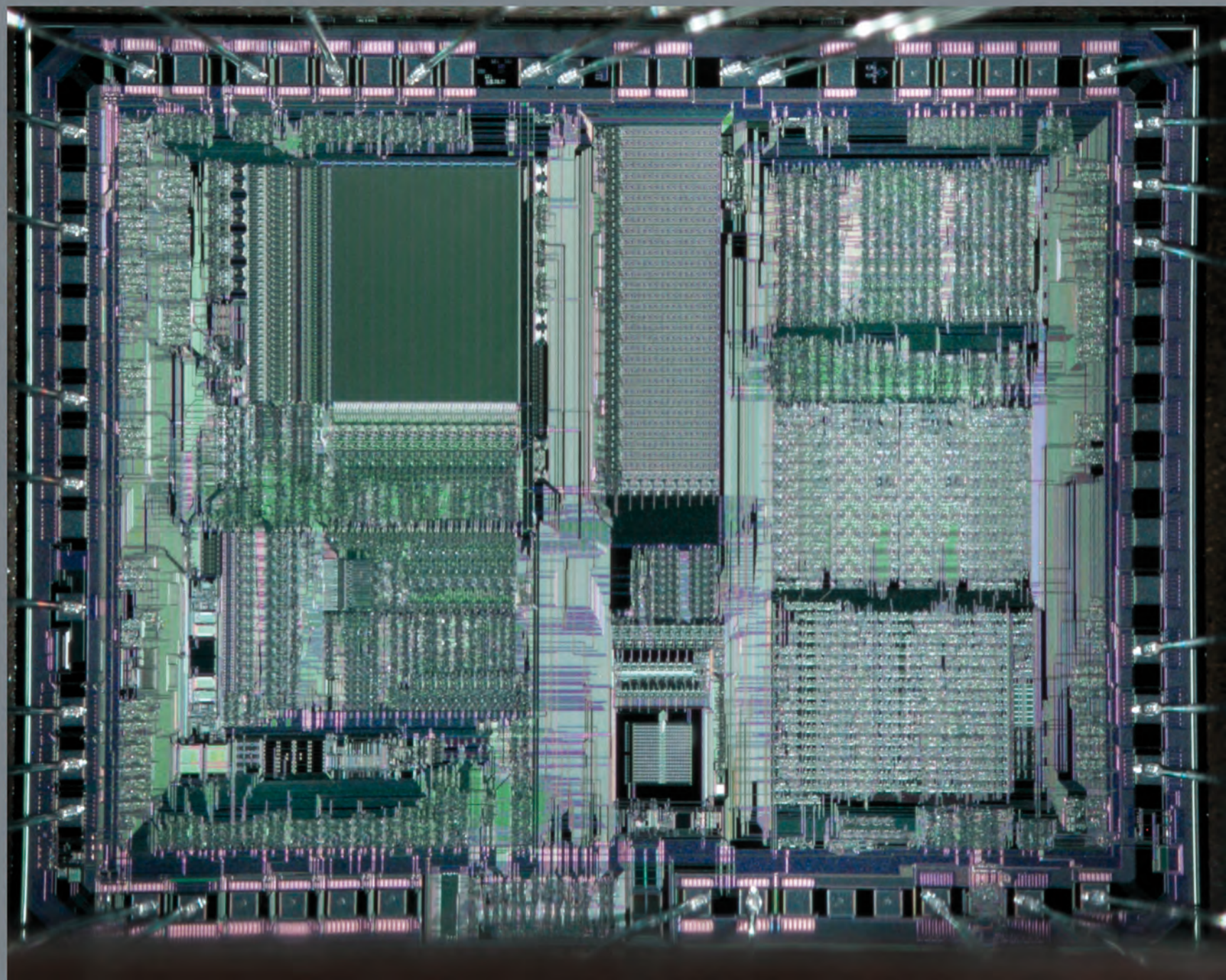
Microcontroladores PIC16

fundamentos y aplicaciones



Alfonso Gutiérrez Aldana

Instituto Politécnico Nacional

Microcontroladores PIC16, fundamentos y aplicaciones
Alfonso Gutiérrez Aldana

Primera edición: 2013

Instituto Politécnico Nacional
Luis Enrique Erro s/n
Unidad Profesional "Adolfo López Mateos"
Zacatenco, Deleg. Gustavo A. Madero
CP 07738, México, DF

Dirección de Publicaciones
Tresguerras 27, Centro Histórico
Deleg. Cuauhtémoc
CP 06040, México, DF

ISBN 978-607-414-392-8

Impreso en México / *Printed in Mexico*
http://www.publicaciones.ipn.mx

Contenido

Introducción 13

Capítulo 1. Conjunto de instrucciones 15
1.1. Conjunto de instrucciones 15
1.1.1. ¿Qué es un mnemónico y por qué se usa? 17
1.1.2. Sistemas de numeración decimal, binario y hexadecimal 17
1.1.3. Memoria en los PIC16 (lo esencial para iniciar) 21
1.1.4. ¿Qué son los registros y las banderas? 22
1.1.5. ¿Qué representan en la tabla 1.1 las letras f, d, b y k? 22
1.1.6. El MPLAB 24
1.2. Las instrucciones y su notación 33
1.2.1. ciclo.asm 33
1.2.2. mult_sumando.asm 38
1.2.3. mult_sumando2.asm 43
1.2.4. multiplicación.asm 47
1.2.5. restas.asm 51
1.3. Descripción resumida del conjunto de instrucciones completo 52

Capítulo 2. Oscilador y reinicio 67
2.1. Osciladores 67
2.1.1. Oscilador RC externo 69
2.1.2. Oscilador RC interno 72
2.1.3. Oscilador con cristal o resonador de cerámica 74
2.1.4. Oscilador externo 76
2.1.5. ¿Cuál oscilador usar? 77
2.2. Circuitos y condiciones de reinicio 80
2.2.1. Reinicio maestro, *Master Clear* (MCLR') 80

2.2.2. Reinicio al encender, *Power-on Reset* (POR) 82
2.2.3. Reinicio por baja tensión, *Brown-out Reset* (BOR) 84
2.2.4. Reinicio por reloj de vigilancia, *Watchdog Timer* (WDT) 87

Capítulo 3. Puertos de uso general 91
3.1. Puertos de uso general 91
3.1.1. Ejemplo: puertos01.asm (una terminal de salida) 94
3.1.2. Ejemplo: puertos02.asm (una terminal de entrada y una de salida)..... 97
3.1.3. Terminales compartidas con periféricos..... 98
3.1.4. Características eléctricas 102
3.1.5. Instrucciones de lectura – modificación – escritura 109
3.1.6. Puertos de uso general para manejar exhibidores de siete segmentos..... 114
3.1.7. Terminales con salida CMOS y drenaje abierto..... 121
3.1.8. Puertos de uso general y teclas..... 123
3.1.9. Levantamiento débil..... 133
3.1.10. Sensor de posición para un eje de rotación 134
3.1.11. Interrupciones..... 142

Capítulo 4. Arquitectura de los PIC16..... 145
4.1. Memoria de datos..... 149
4.1.1. Mapeo..... 149
4.1.2. Direccionamiento directo 151
4.1.3. Precauciones con EQU y CBLOCK-ENDC 153
4.1.4. Registros de uso especial y general..... 155
4.1.5. Direccionamiento indirecto 156
4.1.6. Direccionamiento inmediato 158
4.2. Memoria de programa..... 158
4.2.1. Organización de la memoria de programa 158
4.2.2. Saltos calculados (diseño y uso de tablas) 166
4.2.3. Interrupciones..... 176
4.3. Otras características de la arquitectura que se reflejan en la programación 187
4.3.1. Relación entre el tamaño de las memorias y las instrucciones 188
4.3.2. Ciclo de instrucción..... 190
4.3.3. Paralelismo y tiempo de ejecución..... 190

Capítulo 5. Temporizadores..... 193
5.1. Temporizador 0 193
5.1.1. Reloj 195
5.1.2. Tonos 203

5.2. Temporizador 1 212
5.2.1. Un_segundo 212
5.2.2. Galileo 215
5.3. Temporizador 2 223
5.3.1. Memoria 224

Capítulo 6. Comunicación en serie 249
6.1. Periférico USART 249
6.1.1. El caso asíncrono bidireccional – simultáneo (*full duplex asynchronous*) 249
6.1.2. USART en modo sincronizado 266
6.2. Periférico SSP 271
6.2.1. Modalidad SPI 273
6.2.2. Modalidad I^2C 284

Capítulo 7. Conversión analógico-digital 313
7.1. Intervalo de muestreo 313
7.2. Número de bits en la representación digital 320
7.3. Resolución de la conversión AD. 322
7.4. Convertidor analógico digital de aproximaciones sucesivas 323
7.5. Circuitos que acompañan al convertidor analógico-digital de los PIC16 324
7.6. Conversión AD 327
7.7. Circuitos para acondicionar las señales de los sensores 338

Capítulo 8. Módulo de comparación, captura y modulación por ancho de pulso 341
8.1. Modo de comparación del módulo CCP 343
8.2. Modo de captura del módulo CCP 347
8.3. Modulador por ancho de pulso del módulo CCP 354

Capítulo 9. Exhibidores de cristal líquido 379
9.1. Exhibidores alfanuméricos de cristal líquido 379
9.2. Exhibidores de cristal líquido para gráficos 406

Capítulo 10. Programadores e ICSP 423
10.1. Programadores 423
10.1.1. Programadores desde el MPLAB. 425
10.2. Programación del microcontrolador en el circuito de aplicación (ICSP, *In-Circuit Serial Programming*) 428
10.2.1. Propiedad ICSP 429
10.2.2. Programador basado en el PIC16F628A 432

10.2.3. Comando borra la memoria de programa 434
10.2.4. Programa para leer identificador de dispositivo 439
10.2.5. Formato Intel de 32 bits. 446
10.2.6. lógica_prog.asm 449
10.2.7. programador.asm 456
10.2.8. Programador sin microcontrolador 472
10.2.9. Funciones FIni y FFin 475
10.2.10. Función FBorra. 476
10.2.11. Función FLeeID 479
10.2.12. Función FPrograma 481

Capítulo 11. MPLAB X 489
11.1. MPLAB X en Ubuntu 489
11.2. Creación y simulación de un proyecto 496
11.3. Bits de configuración del microcontrolador 508
11.4. El analizador lógico 511
11.5. Estímulos 512
11.6. Configuración de la frecuencia del oscilador para el simulador y simulación de la USART 514
11.7. Programando los microcontroladores 518

Referencias 521
Anexo. Instrumentación, herramientas y materiales complementarios 523
Herramientas y materiales complementarios 523
Instrumentación 526
Fuentes de alimentación 526
Multímetros 528
Osciloscopios 528

Índice alfabético 531

Introducción

Los avances en electrónica han permitido que las computadoras surjan y se usen en muy diversas áreas del quehacer humano: investigación, ejército, gobierno, negocios, comunicaciones, educación, salud, recreación, etc. Mientras que los microprocesadores se han vuelto más poderosos, también se han especializado y han dado lugar a los ahora llamados procesadores de uso general, a los procesadores digitales de señales (DSP), y a los microcontroladores.

Los microcontroladores se usan ampliamente como "computadoras" que se encuentran dentro de maquinaria, instrumentación, electrodomésticos, vehículos, juguetes y muchos otros aparatos. Debido a la naturaleza de las tareas para las que son diseñados, es común que los microcontroladores no posean una gran capacidad de cálculo, y en su lugar, incorporen facilidades para las comunicaciones, la conversión analógica-digital, la medición de tiempo, la cuenta de eventos y el manejo de "actuadores", entre otras. También debido a la naturaleza de las aplicaciones, las personas que se dedican al desarrollo de proyectos con microcontroladores deben contar con conocimientos y habilidades en por lo menos dos disciplinas: electrónica y computación, situación que se ve reflejada en el contenido del libro.

Con la intención de agilizar el desarrollo de aplicaciones, mientras que son tratados los conceptos básicos referentes a los microcontroladores PIC16, en este libro se presentarán una serie de ejemplos y pequeños proyectos. La programación de estos proyectos se efectuó por completo en lenguaje ensamblador por las razones siguientes:

- Es falsa la idea que usando lenguajes de alto nivel se puede uno librar de conocer a fondo la estructura de los microcontroladores. Es posible programar las computadoras personales conociendo muy poco de su funcionamiento, porque cuentan con un sistema operativo y "manejadores" para gestionar sus recursos que casi siempre son: memoria, puertos, teclados, ratones, impresoras y monitores. El fabricante de los microcontroladores no sabe si su producto será usado con un monitor, un exhibidor de cristal líquido, uno de siete segmentos, unos diodos emisores de luz o, incluso, que no contará con indicadores visuales.

- Aunque más lentamente y requiriendo un poco de dedicación, los humanos podemos gestionar mejor los recursos que los compiladores, por ejemplo: podemos optimizar en forma intercalada algunos segmentos de código para ocupar poca memoria y otros para ejecutarse rápidamente.
- Ya que los compiladores no se hacen solos, las personas con conocimientos de los procesadores y de lenguajes de bajo nivel son requeridas para el diseño y desarrollo de estas herramientas.
- No se requiere adquirir programas para efectuar los desarrollos.

Finalmente, se desea remarcar que la forma de usar este material es experimentando mientras se lee y explorando los códigos y circuitos de ejemplo; las indicaciones para usar el ambiente de desarrollo MPLAB se presentan conforme los ejemplos lo requieren. El lector deberá obtener, a través de Internet, el ambiente de desarrollo MPLAB y la documentación de los procesadores que emplee (ambos están disponibles en forma gratuita en el sitio www.microchip.com); las especificaciones de otros componentes empleados en los desarrollos del texto se pueden obtener en www.alldatasheet.com (también gratis).

CAPÍTULO 1

Conjunto de instrucciones

Como se mencionó en la introducción, y con el fin de que el lector utilice los microcontroladores de la familia PIC16 lo más rápido posible, mientras se presenta el conjunto de instrucciones se darán explicaciones parciales de la arquitectura de los procesadores y del ambiente integral de desarrollo MPLAB. También se hará un pequeño recordatorio de los sistemas de numeración binario y hexadecimal.

Cabe aclarar que la información técnica requerida para el uso de los PIC16 la proporciona en forma gratuita la empresa Microchip y está disponible a través de www.microchip.com. Para escribir este libro se hizo especial uso del documento número 33023a, titulado en el portal como *PIC Mid-Range MCU Family Reference Manual*; ya en el archivo pdf en sí el título es: *PICmicro™ Mid-Range MCU Family Reference Manual*.[1] Es muy recomendable que el lector lo baje para consulta en su computadora; es decir, NO se recomienda imprimirlo completo porque consta de 688 páginas.

1.1. Conjunto de instrucciones

Los microcontroladores PIC16 ejecutan sólo 35 instrucciones diferentes, por lo que se consideran procesadores de tipo RISC (*Reduced Instruction Set Computer*). La tabla 1.1 muestra dicho conjunto de instrucciones.

[1] Referencias al final del libro.

Tabla 1.1. Conjunto de instrucciones de los microcontroladores PIC16

Mnemónico y operndos		Descripción	Ciclos	Instrucción de 14 bits	Banderas afectadas
				MSB LSB	
Operaciones orientadas a un byte en el archivo de registros					
ADDWF	f,d	Suma contenidos de f y W	1	00 0111 dfff ffff	C,DC,Z
ANDWF	f,d	AND entre cont. de f y W	1	00 0101 dfff ffff	Z
CLRF	f	Pone ceros en los bits de f	1	00 0001 1fff ffff	Z
CLRW		Pone ceros en los bits de W	1	00 0001 0000 0000	Z
COMF	f,d	Complementa cont. de f	1	00 1001 dfff ffff	Z
DECF	f,d	Decrece contenido de f	1	00 0011 dfff ffff	Z
DECFSZ	f,d	Decrece cont. f salta si 0	1 (2)	00 1011 dfff ffff	
INCF	f,d	Incrementa contenido de f	1	00 1010 dfff ffff	Z
INCFSZ	f,d	Incrementa cont. f salta si 0	1 (2)	00 1111 dfff ffff	
IORWF	f,d	OR entre cont. de f y W	1	00 0100 dfff ffff	Z
MOVF	f,d	Mueve el contenido de f	1	00 1000 dfff ffff	Z
MOVWF	f	Mueve cont. de W a f	1	00 0000 1fff ffff	
NOP		No ejecuta operación	1	00 0000 0000 0000	
RLF	f,d	Rota a izquierda, pasa por C	1	00 1101 dfff ffff	C
RRF	f,d	Rota a derecha, pasa por C	1	00 1100 dfff ffff	C
SUBWF	f,d	Cont. f menos cont. W	1	00 0010 dfff ffff	C,DC,Z
SWAPF	f,d	Intercambia medios bytes	1	00 1110 dfff ffff	
XORWF	f,d	XOR entre cont. de f y W	1	00 0110 dfff ffff	Z
Operaciones orientadas a un bit en el archivo de registros					
BCF	f,b	Pone a cero un bit de f	1	01 00bb bfff ffff	
BSF	f,b	Pone a uno un bit de f	1	01 01bb bfff ffff	
BTFSC	f,b	Verifica un bit y salta si 0	1 (2)	01 10bb bfff ffff	
BTFSS	f,b	Verifica un bit y salta si 1	1 (2)	01 11bb bfff ffff	
Operaciones con una constante y de control					
ADDLW	k	k más contenido de W	1	11 1110 kkkk kkkk	C,DC,Z
ANDLW	k	k AND contenido de W	1	11 1001 kkkk kkkk	Z
CALL	k	Llamada a subrutina en k	2	10 0kkk kkkk kkkk	
CLRWDT		Reinicia reloj de vigilancia	1	00 0000 0110 0100	TO',PD'
GOTO	k	Salto incondicional a k	2	10 1kkk kkkk kkkk	
IORLW	k	k OR contenido de W	1	11 1000 kkkk kkkk	Z
MOVLW	k	Guarda k en W	1	11 0000 kkkk kkkk	
RETFIE		Regresa de interrupción	2	00 0000 0000 1001	
RETLW	k	Regresa de subrut. k en W	2	11 0100 kkkk kkkk	
RETURN		Regresa de subrutina	2	00 0000 0000 1000	
SLEEP		Pone en ahorro de energía	1	00 0000 0110 0011	TO',PD'
SUBLW	k	k menos contenido de W	1	11 1100 kkkk kkkk	C,DC,Z
XORLW	k	k XOR contenido de W	1	11 1010 kkkk kkkk	Z

En la primera columna de la tabla se encuentran los mnemónicos de las instrucciones, mientras que las instrucciones en sí, están en la cuarta columna. Así es, las instrucciones constan de secuencias de ceros y unos, por ejemplo 00 0010 0000 0001 es una resta. De hecho, describir esta tabla implica tratar, por lo menos en forma parcial, los siguientes puntos:

- ¿Qué es un mnemónico y por qué se usa?
- Sistemas de numeración decimal, binario y hexadecimal.
- Memoria en los PIC16.
- ¿Qué son los registros y las banderas?
- ¿Qué representan en la tabla 1.1 las letras f, d, b y k?
- El MPLAB.
- Las instrucciones en sí, su notación y clasificación.

1.1.1. ¿Qué es un mnemónico y por qué se usa?

Los seres humanos no somos especialmente hábiles para trabajar con secuencias de unos y ceros, secuencias que en este caso codifican las instrucciones que ejecutan los PIC16. Tan sólo en la tabla 1.1, para lograr un poco de claridad, se requirió separar los 14 bits que constituyen una instrucción con sus operandos. En lugar de escribir programas a partir de ceros y unos (lenguaje máquina), usamos secuencias de mnemónicos (lenguaje ensamblador) e incluso programamos en lenguajes "etiquetados" como de nivel intermedio (por ejemplo el C) o de alto nivel (como Java).

Un mnemónico es una palabra que facilita recordar, aunque se debe considerar que los microcontroladores no están diseñados en países donde se habla español. Por ejemplo, RLF es una instrucción que permite rotar a la izquierda el contenido del registro f (*Rotate Left f*). En español el mnemónico sería RIF (rotar izquierda f) o tal vez mejor RICRF (rotar a la izquierda el contenido del registro f); cualquier caso es más fácil que recordar que 001101 es la instrucción para el microcontrolador y que hay que agregar otros 8 bits que corresponderían a los operandos: 00110111000000, que escribiríamos 00 1101 1100 0000, y diría: rotar a la izquierda el contenido del registro 64 y guardar el resultado en ese mismo registro.

1.1.2. Sistemas de numeración decimal, binario y hexadecimal

Quien sabe notación hexadecimal sabe por qué en la tabla 1.1 se agruparon los dígitos de las instrucciones en código máquina en bloques de cuatro elementos contados a partir de la derecha; también sabe que este sistema de numeración es "intermedio" entre el sistema decimal y el binario, formas de numeración para seres humanos y para computadoras.

En el sistema decimal se usan agrupaciones de diez símbolos diferentes (0, 1, 2, 3, 4, 5, 6, 7, 8 y 9) para representar diversas cantidades; se dice que es un sistema base diez. Por ejemplo, al contar elementos, el 0 representa que no hay elementos, el 1 indica que hay un elemento y

el 9 sería usado para nueve elementos; nada nuevo. En caso de representar el conteo de más de nueve elementos, NO se usan símbolos nuevos: se da un significado o peso diferente a cada símbolo de acuerdo con su posición. En los cursos de educación básica las posiciones suelen tener nombre propio: unidades, decenas, centenas, etc.; en los cursos más avanzados se dice que el peso es una potencia de diez.

Tabla 1.2. Sistema decimal

POSICIÓN							
5	4	3	2	1	0		
cien mil	diez mil	mil	cien	diez	uno	Valor por el que multiplicamos el símbolo que aparece en la posición	PESO
			Centena	Decena	Unidad	Nombre propio de la posición o peso	
10^5	10^4	10^3	10^2	10^1	10^0	Notación científica	
ALGUNOS EJEMPLOS						EXPRESIONES DE CÁLCULO (QUE NO SOLEMOS HACER)	VALOR
				1	6	$1 \times \text{diez} + 6 \times \text{uno}$ $1 \times 10 + 6 \times 1$ $1 \times 10^1 + 6 \times 10^0$	16
			0	1	6	$0 \times \text{cien} + 1 \times \text{diez} + 6 \times \text{uno}$ $0 \times 100 + 1 \times 10 + 6 \times 1$ $0 \times 10^2 + 1 \times 10^1 + 6 \times 10^0$	16
		2	0	1	0	$2 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 0 \times 10^0$	2010
1	6	0	9	1	0	$1 \times 10^5 + 6 \times 10^4 + 9 \times 10^2 + 1 \times 10^1$	160910

Como se muestra en el primer ejemplo de la tabla 1.2, para representar dieciséis elementos usamos el símbolo 6 en la posición menos significativa, su valor es seis; también usamos un 1 en la siguiente posición a la izquierda, el valor de este uno se multiplica por el peso de la posición y resulta un diez; sumando estos valores obtenemos el dieciséis.

En el segundo ejemplo se agregó al 16 un cero a la izquierda, es decir, 016; si no usáramos cotidianamente el sistema decimal, tendríamos que efectuar la siguiente operación para determinar el valor que estamos representando: cero veces cien + una vez diez + seis veces uno; el objetivo de este segundo ejemplo es hacer notar que escribiendo en decimal no usamos los ceros a la izquierda porque no aportan información.

En el último ejemplo de la tabla se representa el ciento sesenta mil novecientos diez; una vez cien mil + seis veces diez mil + cero veces mil + nueve veces cien + una vez diez + cero veces uno; una cantidad bastante grande, representada sistemáticamente a partir de diez símbolos que toman diferente valor dependiendo de la posición en que aparecen.

Tabla 1.3. Sistema binario

POSICIÓN									
7	6	5	4	3	2	1	0		
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	PESO	
128	64	32	16	8	4	2	1		
ALGUNOS EJEMPLOS								CÁLCULO (decimal)	VALOR (decimal)
0	0	0	0	0	0	0	0	0 × 128 + 0 × 64 + 0 × 32 + 0 × 16 + 0 × 8 + 0 × 4 + 0 × 2 + 0 × 1	0
0	0	0	0	0	0	0	1	1 × 1	1
0	0	0	0	0	1	0	0	1 × 4	4
1	0	0	0	0	0	0	0	1 × 128	128
1	0	1	0	1	0	1	0	1 × 128 + 1 × 32 + 1 × 8 + 1 × 2	170
1	1	1	1	1	1	1	1	1 × 128 + 1 × 64 + 1 × 32 + 1 × 16 + 1 × 8 + 1 × 4 + 1 × 2 + 1 × 1	255

En el sistema binario hay sólo dos símbolos y se usa en las computadoras digitales porque es relativamente simple fabricar transistores que operen en dos estados diferentes: corte y saturación. Con los transistores trabajando en estos modos se generan los circuitos para representar los símbolos 0 y 1; sin embargo, es importante resaltar que corte NO representa forzosamente alguno de los símbolos y saturación al otro, eso depende de la tecnología usada; en el caso de los PIC16 se usa tecnología CMOS y obtener cada símbolo requiere del uso de dos transistores operando en forma complementaria (el tema se retomará al abordar el uso de los puertos).

Los ejemplos mostrados en la tabla 1.3 se plantean usando ocho posiciones, es decir, 8 bits o un byte. Al contrario que en la notación decimal, al usar el sistema binario solemos representar los ceros que están a la izquierda. El objetivo de esto es proporcionar información del tipo de datos que se pueden procesar; así, en el primer ejemplo de la tabla 1.3 se usa 0000 0000 para representar un cero, y dependiendo del contexto, asumiríamos que estamos trabajando con un procesador de 8 bits, o que tenemos un dato de tipo *char* o *unsigned char* en el lenguaje C, entre otras posibilidades.

En el último ejemplo de la misma tabla se observa que el número entero positivo más grande que podemos representar en sistema binario, al usar 8 bits, es doscientos cincuenta y cinco; se puede usar la ecuación 1.1 para determinar dicho número mayor a partir del número de posiciones en la cifra.

$$mayor = 2^n - 1 \qquad (1.1)$$

donde: n es el número de posiciones de la cifra

Tabla 1.4. Sistema hexadecimal

BINARIO BASE 2								HEXADECIMAL BSE 16		DECIMAL BASE 10		
POSICIÓN								POSICIÓN		POSICIÓN		
7	6	5	4	3	2	1	0	1	0	2	1	0
PESO								PESO		PESO		
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	16^1	16^0	10^2	10^1	10^0
128	64	32	16	8	4	2	1	16	1	100	10	1
ALGUNOS EJEMPLOS DE CIFRAS EN LOS TRES SISTEMAS												
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	1	0	0	1
0	0	0	0	0	0	1	0	0	2	0	0	2
0	0	0	0	0	0	1	1	0	3	0	0	3
0	0	0	0	0	1	0	0	0	4	0	0	4
0	0	0	0	0	1	0	1	0	5	0	0	5
0	0	0	0	0	1	1	0	0	6	0	0	6
0	0	0	0	0	1	1	1	0	7	0	0	7
0	0	0	0	1	0	0	0	0	8	0	0	8
0	0	0	0	1	0	0	1	0	9	0	0	9
0	0	0	0	1	0	1	0	0	A	0	1	0
0	0	0	0	1	0	1	1	0	B	0	1	1
0	0	0	0	1	1	0	0	0	C	0	1	2
0	0	0	0	1	1	0	1	0	D	0	1	3
0	0	0	0	1	1	1	0	0	E	0	1	4
0	0	0	0	1	1	1	1	0	F	0	1	5
0	0	0	1	0	0	0	0	1	0	0	1	6
0	0	1	0	0	0	0	0	2	0	0	3	2
0	1	0	0	0	1	0	0	4	4	0	6	8
1	0	0	0	0	0	0	0	8	0	1	2	8
1	1	1	1	1	1	1	1	F	F	2	5	5

La tabla 1.4 muestra los ejemplos correspondientes al sistema hexadecimal. Para determinar el valor de una cifra seguiríamos el mismo proceso que en los sistemas decimal y binario, sólo que en esta ocasión contamos con dieciséis símbolos: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F (sistema base dieciséis). Del 0 al 9 los símbolos representan lo mismo que en sistema decimal, el símbolo A representa un diez, el B un once y así sucesivamente hasta el F, que representa quince. En esta tabla hay también representaciones binarias y decimales para poder comparar; observe que cuatro dígitos binarios permiten un recorrido de cero a quince, al igual que los símbolos en una sola posición al usar hexadecimal; así, el número binario 0001

0000 tiene su equivalente hexadecimal 10 (que solemos representar 0x10 o 010 para aclarar la base), el 0100 0100 equivale a 0x44. Para hacer la conversión mentalmente, sólo se requiere recordar los primeros cuatro pesos del sistema binario, que serían de derecha a izquierda: 1, 2, 4 y 8; de aquí que las cifras binarias se dividan en grupos de cuatro dígitos. Otra forma de hacer la conversión es memorizar los equivalentes binarios de los dieciséis símbolos hexadecimales y remplazar simplemente un grupo de cuatro bits por un dígito hexadecimal.

Para terminar este breve recordatorio o presentación de los sistemas de numeración, diremos que la ventaja de usar el sistema hexadecimal se empezará a notar cuando tratemos las banderas, y será evidente al hablar de los registros de uso especial y la configuración de los periféricos del PIC.

1.1.3. Memoria en los PIC16 (lo esencial para iniciar)

Las computadoras digitales usan dispositivos electrónicos denominados "memoria" para almacenar la información que procesarán y la forma como efectuarán el proceso, es decir, en la memoria se almacenan datos y programas. En el caso de los microcontroladores PIC16, hay memoria incluida en el circuito integrado y es de dos tipos: el primero NO pierde su contenido cuando falla la energía y se usa para almacenar el programa; el segundo tipo sólo es capaz de retener información mientras el circuito se encuentra energizado y se usa para almacenar datos.

Aunque todos los PIC16 cuentan con memoria para programa y memoria para datos, la capacidad de éstas es distintiva de cada modelo en particular, por ejemplo: el PIC16F84 puede almacenar 1024 instrucciones y cuenta con 68 bytes para datos, mientras que el PIC16F887 almacena hasta 8192 instrucciones y tiene 368 bytes destinados a datos. Para los programadores de computadoras personales o superiores, estas capacidades podrán parecer de juego, pero debido a "los periféricos" que están integrados en los microcontroladores, estas capacidades son apropiadas para gran cantidad de proyectos.

Cada localidad en la memoria de programa es de 14 bits, espacio suficiente para almacenar una instrucción y sus operandos (véase columna 4 de la tabla 1.1). La memoria de programa está dividida en secciones a las que se les llama páginas; cada página es de 2048 espacios y el PIC16 que tiene más memoria de programa, tiene cuatro páginas. La primera localidad se accede a través de la dirección 0. Una dirección en la memoria de programa se puede convertir en operando (dato) cuando se ejecutan instrucciones como CALL (llamada a subrutina) y GOTO (salto incondicional). Es importante notar que el operando para estas instrucciones es una constante de 11 bits.

La memoria para datos cuenta con localidades de 8 bits. Esta memoria también está dividida en secciones; en esta ocasión a dichas secciones se les llama bancos. Cada banco es de 128 bytes y el PIC16 con más bancos, tiene 4. "Para colmo", las primeras 32 localidades de cada banco ya están ocupadas. La primera localidad se accede a través de la dirección 0; tenga en mente que se trata de una localidad diferente a la 0 en la memoria de programa (véase figura 1.1).

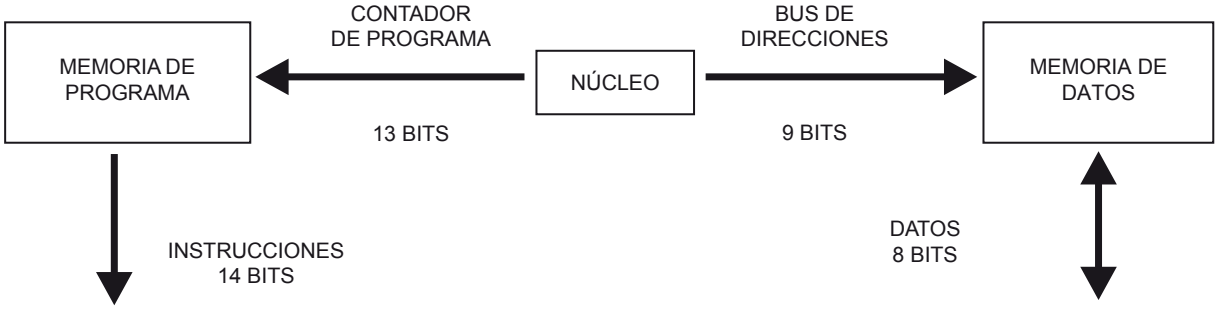


Figura 1.1. Memoria en los PIC16. Se observa un núcleo o unidad central de proceso que es capaz de acceder a dos bloques de memoria; en el de la izquierda se almacenan las instrucciones por ejecutar (cada una de 14 bits), se selecciona la localidad por leer a través del contador de programa (13 líneas); a la derecha está la memoria de datos, aquí las localidades son de 8 bits y se selecciona la que se usará a través del bus de direcciones de 9 bits.

1.1.4. ¿Qué son los registros y las banderas?

Los registros son circuitos capaces de almacenar información codificada en sistema binario; por lo general su capacidad es de 8 bits. Podría considerarse que una memoria es una agrupación de registros en la cual usamos una dirección para referirnos a cada uno de ellos individualmente. Así, en los PIC existen registros de uso especial y de uso general. A cada bit de los registros de uso especial se le ha asignado su funcionalidad desde el momento de diseñar el PIC; a los registros de uso general, el programador les asignará su tarea al momento de diseñar el programa que comandará una aplicación en particular (véase figura 1.2).

El registro de trabajo (W) y el de estado (*STATUS*) son dos de los registros especiales en los PIC. Casi todos los datos que procesa un PIC deben pasar por el registro de trabajo, de ahí su nombre, aunque en otros procesadores se le puede llamar acumulador. El registro de estado "avisa" en qué condiciones se encuentra el PIC.

Cada bit en el registro de estado tiene un nombre y una tarea propios; cuando esta tarea consiste en dar un aviso en particular, al bit se le conoce como bandera. Como ejemplos están: *Z* (*zero*) es una bandera en el registro *STATUS* que avisa que el resultado de la última operación efectuada fue cero; después de efectuar una suma, la bandera *C* (*carry*), también en el registro *STATUS*, contendrá un uno si el resultado implica un acarreo (en la primaria decíamos: "y llevamos uno").

1.1.5. ¿Qué representan en la tabla 1.1 las letras f, d, b y k?

En la primera columna de la tabla 1.1 se presentan los mnemónicos de las instrucciones que manejan los PIC16; después de cada instrucción puede presentarse una **f**, **d**, **b** o **k**. Estas letras representan los datos u operandos que requiere cada instrucción.

	Dirección		Dirección		Dirección		Dirección
INDF	0x00	**INDF**	0x80	**INDF**	0x100	**INDF**	0x180
TMR0	0x01	**OPTION_REG**	0x81	**TMR0**	0x101	**OPTION_REG**	0x181
PCL	0x02	**PCL**	0x82	**PCL**	0x102	**PCL**	0x182
STATUS	0x03	**STATUS**	0x83	**STATUS**	0x103	**STATUS**	0x183
FSR	0x04	**FSR**	0x84	**FSR**	0x104	**FSR**	0x184
PORTA	0x05	TRISA	0x85		0x105		0x185
PORTB	0x06	TRISB	0x86	PORTB	0x106	TRISB	0x186
PORTC	0x07	TRISC	0x87	PORTF	0x107	TRISF	0x187
PORTD	0x08	TRISD	0x88	PORTG	0x108	TRISG	0x188
PORTE	0x09	TRISE	0x89		0x109		0x189
PCLATH	0x0A	**PCLATH**	0x8A	**PCLATH**	0x10A	**PCLATH**	0x18A
INTCON	0x0B	**INTCON**	0x8B	**INTCON**	0x10B	**INTCON**	0x18B
PIR1	0x0C	PIE1	0x8C		0x10C		0x18C
PIR2	0x0D	PIE2	0x8D		0x10D		0x18D
TMR1L	0x0E	**PCON**	0x8E		0x10E		0x18E
TMR1H	0x0F	OSCCAL	0x8F		0x10F		0x18F
T1CON	0x10		0x90		0x110		0x190
TMR2	0x11		0x91		0x111		0x191
T2CON	0x12	PR2	0x92		0x112		0x192
SSPBUF	0x13	SSPADD	0x93		0x113		0x193
SSPCON	0x14	SSPATAT	0x94		0x114		0x194
CCPR1L	0x15		0x95		0x115		0x195
CCPR1H	0x16		0x96		0x116		0x196
CCP1CON	0x17		0x97		0x117		0x197
RCSTA	0x18	TXSTA	0x98		0x118		0x198
TXREG	0x19	SPBRG	0x99		0x119		0x199
RCREG	0x1A		0x9A		0x11A		0x19A
CCPR2L	0x1B		0x9B		0x11B		0x19B
CCPR2H	0x1C		0x9C		0x11C		0x19C
CCP2CON	0x1D		0x9D		0x11D		0x19D
ADRES	0x1E		0x9E		0x11E		0x19E
ADCON0	0x1F	ADCON1	0x9F		0x11F		0x19F
REGISTROS DE USO GENERAL	0x20 – 0x7F	REGISTROS DE USO GENERAL	0xA0 – 0xEF	REGISTROS DE USO GENERAL	0x120 – 0x16F	REGISTROS DE USO GENERAL	0x1A0 – 0x1EF
		MAPEADOS EN EL BANCO 0 0x70 – 0x7F	0xF0 – 0xFF	MAPEADOS EN EL BANCO 0 0x70 – 0x7F	0x170 – 0x17F	MAPEADOS EN EL BANCO 0 0x70 – 0x7F	0x1F0 – 0x1FF

Figura 1.2. A través de la memoria para datos de los PIC16, se puede acceder a una serie de registros relacionados con la operación del núcleo del procesador y de sus periféricos (registros de uso específico); estos registros ocupan las primeras 32 localidades en cada banco. Las celdas sombreadas representan localidades que no existen en los dispositivos.

La letra **f** representa una dirección en el llamado archivo de registros (*Register File*), que por el momento podemos imaginar como la memoria de datos (figura 1.2). En la columna cuatro de la tabla 1.1 se puede observar que el valor de **f** ocupa siete bits cuando la instrucción se escribe en lenguaje máquina, es decir, podemos escribir las direcciones desde 0 hasta 127.

La letra **d** sirve para indicar el destino de un resultado, es decir, el lugar de almacenamiento para dicho resultado. En la columna cuatro de la tabla 1.1 se puede observar que el valor de **d** ocupa un bit cuando la instrucción se escribe en lenguaje máquina, es decir, existen dos posibles destinos: un 0 indica que el resultado se almacenará en el registro de trabajo (registro W); un 1 indica que el resultado se almacenará en la misma localidad de memoria donde se obtuvo el dato.

La **b** es una constante de tres bits y se usa para referirse a un bit dentro de un byte. Con tres dígitos binarios podemos escribir los números decimales del 0 al 7, así que los ocho bits dentro de un byte reciben los nombres 7, 6, 5, 4, 3, 2, 1 y 0. Al bit 0 también lo conocemos como bit menos significativo (LSB), mientras que nos referimos al bit 7 como bit más significativo (MSB).

Finalmente, la **k** representa una constante que puede ser de ocho u once bits. Las constantes de 8 bits contienen un dato que es entregado al núcleo de forma simultánea con la instrucción. Sólo las instrucciones CALL y GOTO usan constantes de 11 bits, éstas representan una dirección en la memoria de programa.

1.1.6. El MPLAB

MPLAB es el nombre del entorno o ambiente integrado de desarrollo que ofrece la compañía Microchip para trabajar con sus productos. Para el desarrollo de este libro se usó la versión 8.53, que era la más nueva al iniciar el proyecto. Entre otras cosas, dicho ambiente integrado de desarrollo IDE (por sus siglas en inglés) permite editar, simular y depurar los programas, así como fungir como controlador del equipo que finalmente programará el circuito integrado.

Al acceder al portal http://www.microchip.com/ se presenta un grupo titulado *Design*; en este grupo hay que seleccionar *MPLAB® IDE* y en la sección *Downloads* encontrará un archivo comprimido con el programa (MPLAB_IDE_8_53.zip). Después de descomprimir, hay que ejecutar *setup*. Se efectuarán las preguntas acostumbradas de aceptación de condiciones, carpeta para instalación, tipo de instalación (completa), etc. Entre las preguntas para instalar aparecerá una ofreciéndole la instalación de un compilador de lenguaje C; evidentemente usted está en libertad de seleccionar lo que desee, pero como se planteó en la introducción, en este libro se trabajará sólo con ensamblador.

Al final de la instalación se ofrecerá, a través de una ventana especial, la posibilidad de seleccionar documentación de microcontroladores, lenguaje ensamblador, programadores, etcétera.

El manual de usuario del MPLAB[2] está disponible también en el sitio de Microchip, pero aquí plantearemos una secuencia de pasos que prepararán el ambiente para el primer ejemplo que desarrollaremos:

1. Una vez que se esté ejecutando el MPLAB, seleccione *Project Wizard...* en el menú *Project* (figura 1.3). Aparecerá una ventana de presentación y será necesario oprimir el botón "siguiente".
2 . La ventana siguiente permite seleccionar el procesador para el que desarrollaremos la aplicación, en este primer ejemplo usaremos el PIC16F84A (figura 1.4).
3. Después obtendremos la ventana en la que se selecciona el lenguaje de desarrollo; para todos nuestros proyectos usaremos el lenguaje ensamblador y será necesario seleccionar *Microchip MPASM Toolsuite*. MUY IMPORTANTE: no avance de esta ventana si en la sección *Toolsuite Contents* aparece algún componente marcado con un tache, se trata de un problema con la trayectoria donde fueron almacenados los archivos y se corrige seleccionando la carpeta (directorio) correcta con el botón *Browse* (figura 1.5).
4. A continuación se debe seleccionar la carpeta que contendrá el proyecto; use el botón *Browse*. Después dé un nombre al proyecto; en nuestro primer ejemplo haremos un ciclo de 10 iteraciones, por lo que sugerimos el nombre "ciclo". El MPLAB agregará la extensión "mcp" (figura 1.6).

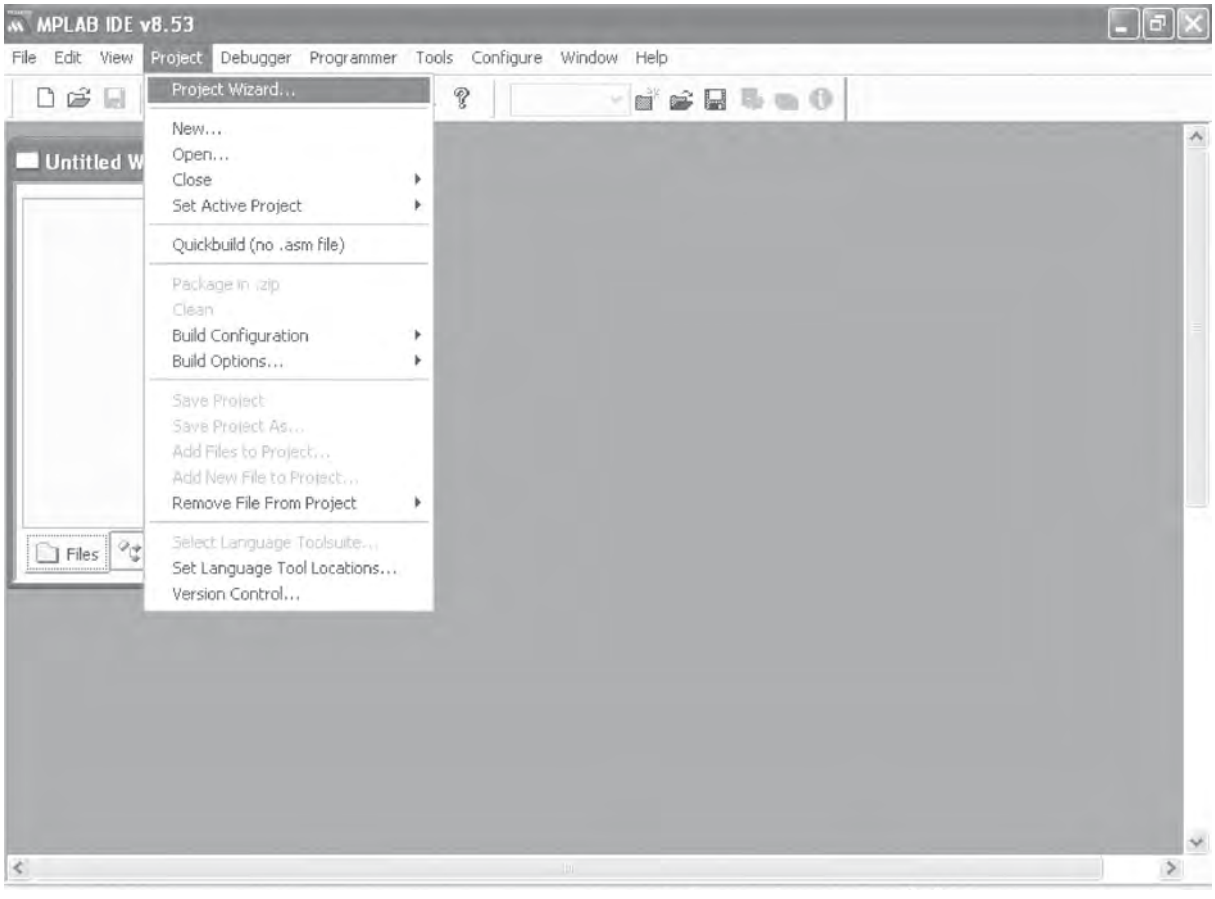


Figura 1.3. Project Wizard...

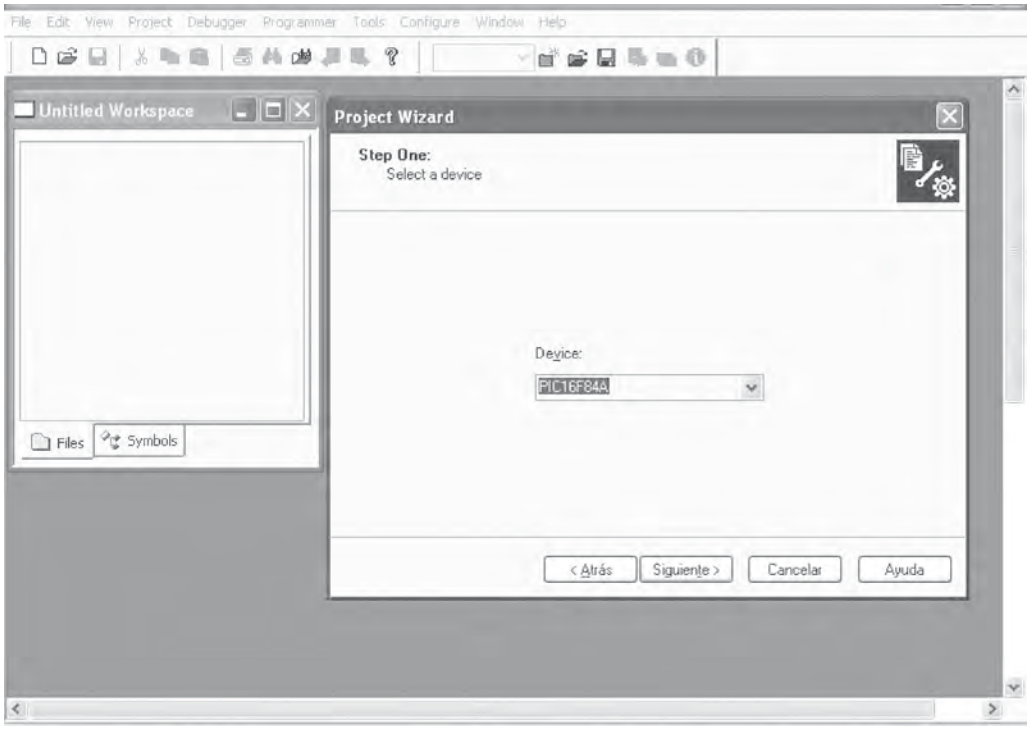


Figura 1.4. Selección del microcontrolador.

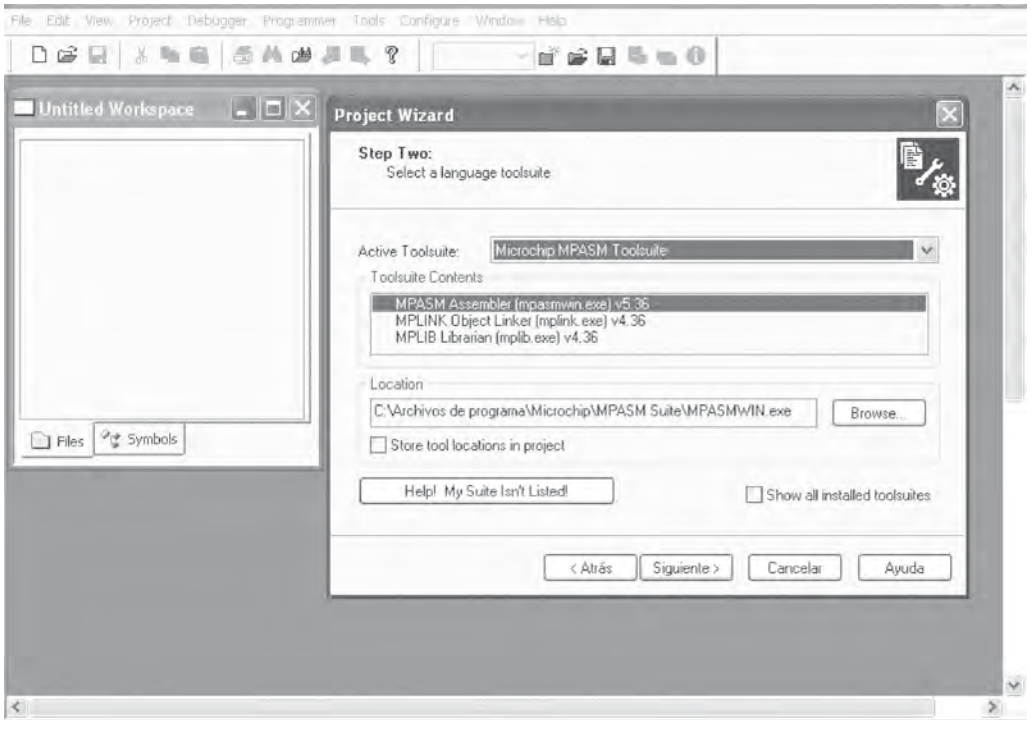


Figura 1.5. Selección de herramientas de desarrollo.

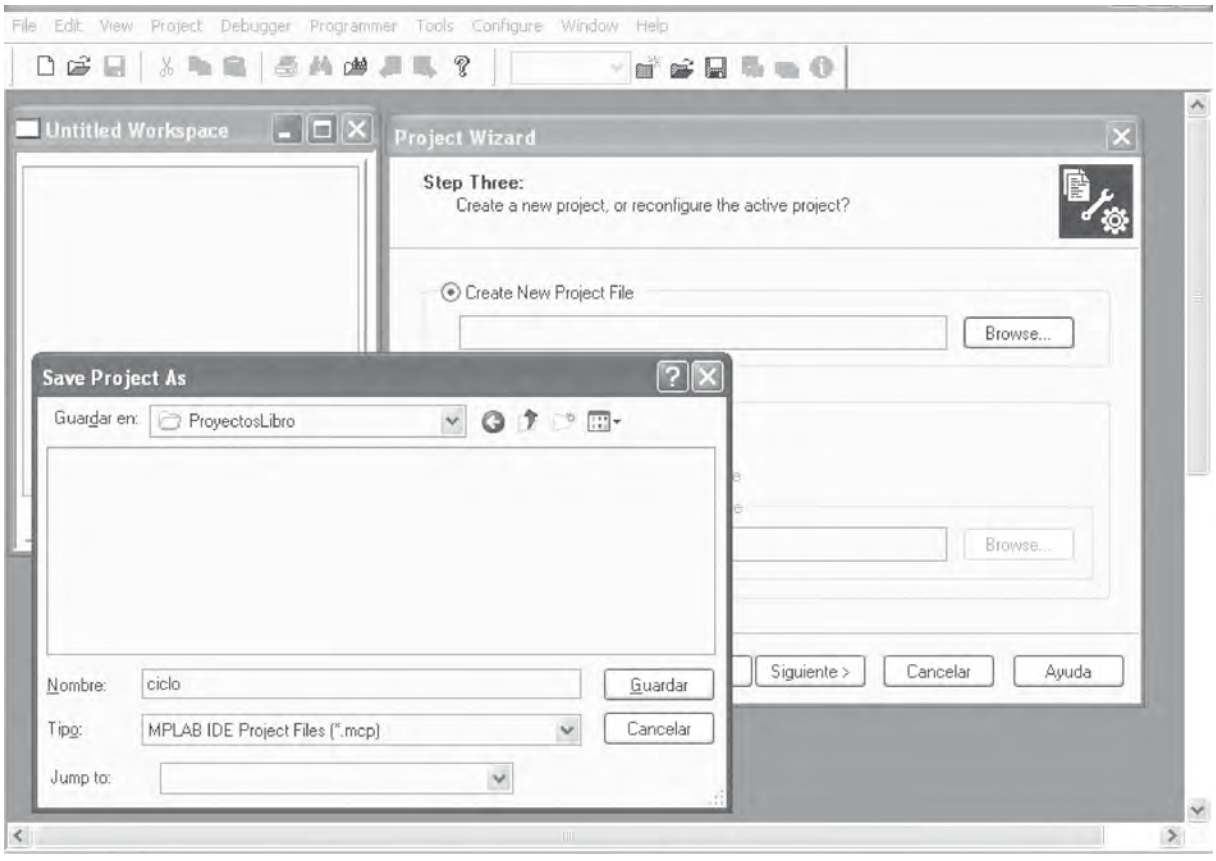


Figura 1.6. Guardar proyecto.

5. Como NO hemos creado un archivo con el código de nuestro programa, en la siguiente ventana no podremos agregarlo al proyecto y nos concretaremos a oprimir "siguiente" (figura 1.7).
6. Aparecerá una ventana resumiendo los datos del proyecto que hemos creado. Se debe oprimir "finalizar" (figura 1.8).
7. Ahora tenemos un proyecto vacío, así que sigue crear el código fuente. Use la opción *New* del menú *File*. En el ambiente aparece una sección donde podemos editar texto. Antes de escribir cualquier cosa, seleccionaremos la opción *Save As...* del menú *File* para darle un nombre al archivo que contendrá el programa; utilizaremos el nombre "ciclo.asm" (figura 1.9). Aunque en el ambiente ya vemos la ventana de edición para el archivo "ciclo.asm", éste aún no forma parte del proyecto; para incorporarlo hay que usar el ratón para colocar el puntero sobre la ventana de proyecto (titulada "ciclo.mcw"). Al oprimir el botón derecho sobre *Source Files*, aparecerá un menú en el que seleccionaremos *Add Files...*; después seleccionaremos "ciclo.asm" y oprimiremos "Abrir"; con esto el archivo ".asm" ya forma parte del proyecto (figura 1.10).

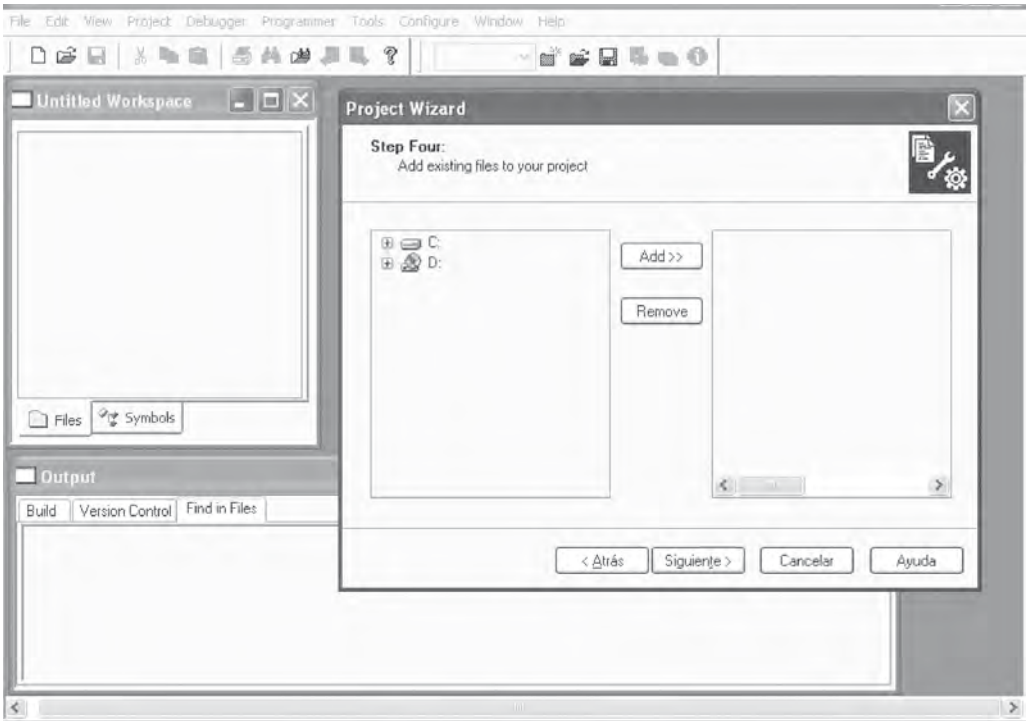


Figura 1.7. No hay archivo con código fuente para incorporar al proyecto.

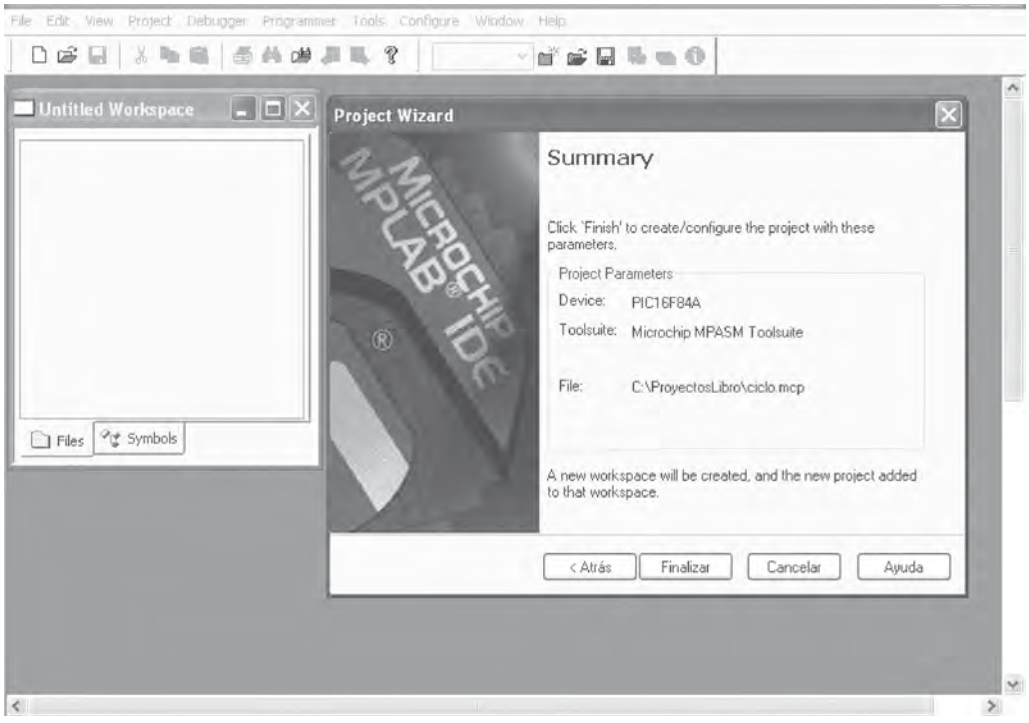


Figura 1.8. Ventana del MPLAB al finalizar la plantilla de un proyecto.

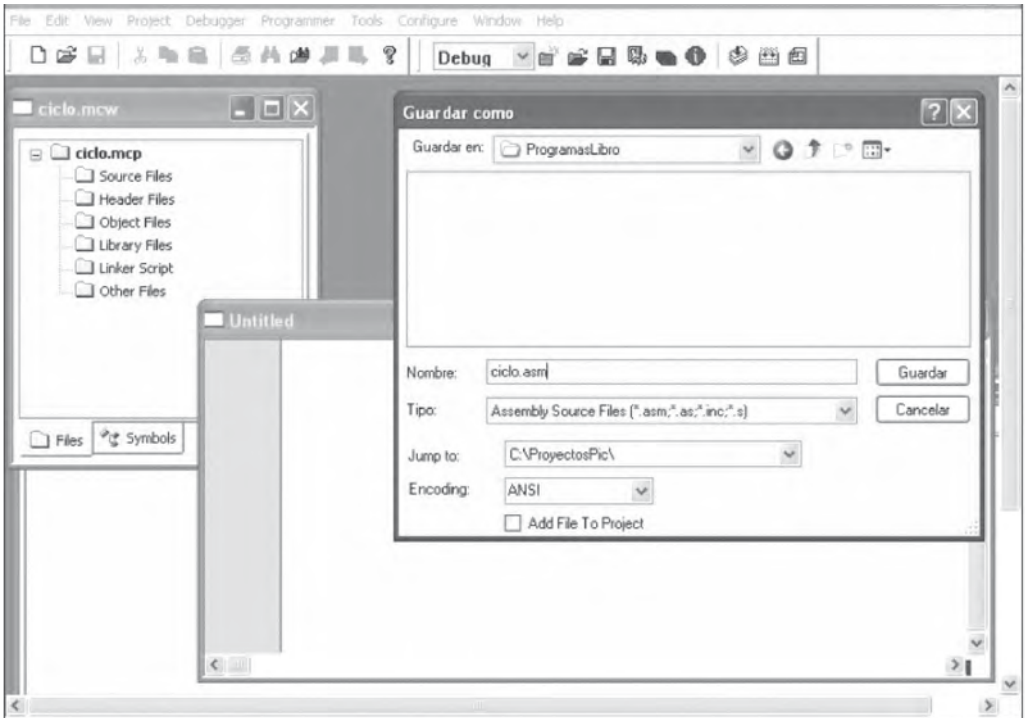


Figura 1.9. Guardando archivo de código fuente.

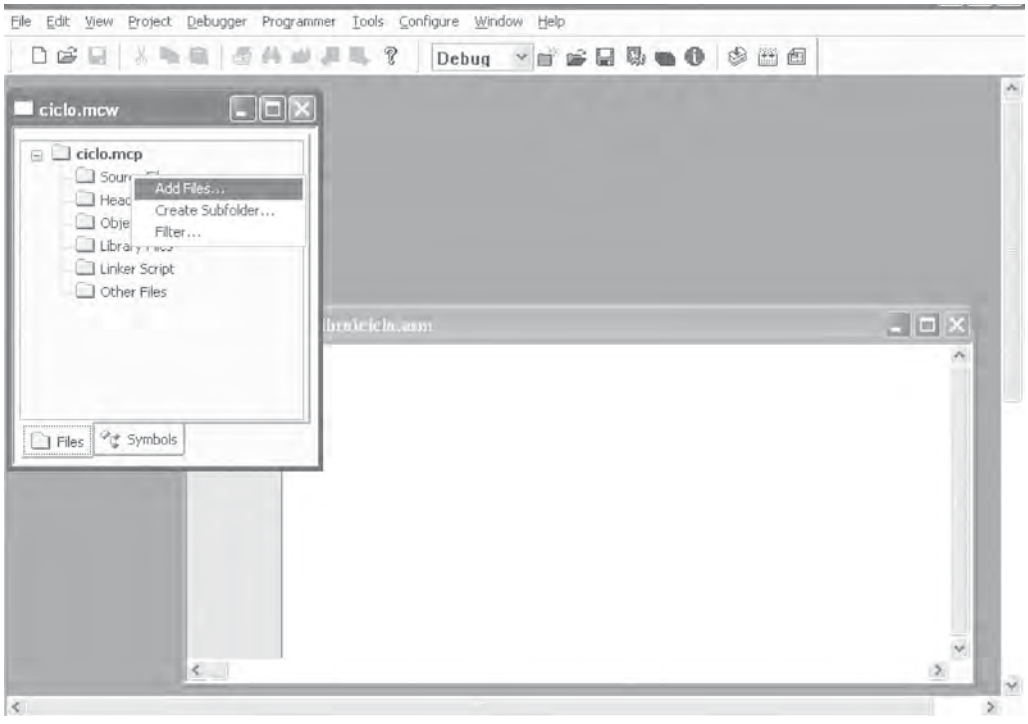


Figura 1.10. Incorporando el código fuente al proyecto.

8. Para evaluar en forma controlada el funcionamiento de los programas se usa un depurador; en este caso usaremos el *MPLAB SIM*, que forma parte del ambiente de desarrollo; para activarlo use la opción *Select Tool* en el menú *Debugger* (figura 1.11).
9. Agregaremos un componente en el ambiente que permitirá observar cómo se desarrolla el programa; para hacerlo seleccione la opción *Watch* en el menú *View* (figura 1.12).
10. En la ventana de programa "ciclo.asm", agregaremos cuatro directivas para el ensamblador (figura 1.12). Es importante notar que éstas no son instrucciones para el microcontrolador y se hablará con detalle de ellas más adelante. Teclearemos **al menos un espacio**, en este caso dos tabuladores, antes de cada directiva. Directiva 1, PROCESSOR 16F84A; indica al MPLAB qué microcontrolador usaremos. Directiva 2, INCLUDE P16F84A.INC; ordena que el archivo "P16F84A.INC" forme parte del proyecto. Directiva 3, ORG 0; ordena al MPLAB que comience a acomodar el programa a partir de la dirección cero de la memoria de programa. Directiva 4, END; ordena que se detenga el proceso de "traducción" de lenguaje ensamblador a lenguaje de máquina; recuerde, NO significa detener la ejecución del programa en el microcontrolador.

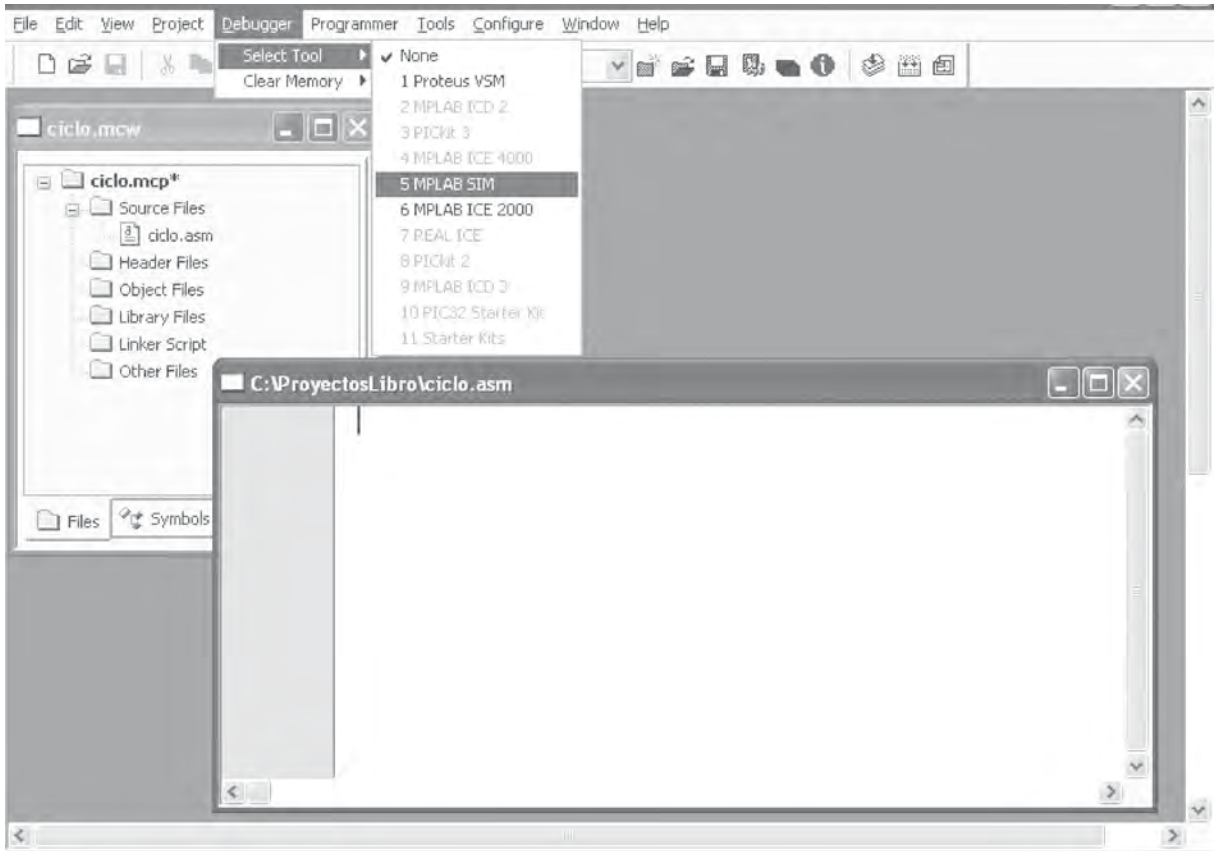


Figura 1.11. Seleccionando la herramienta para depurar.

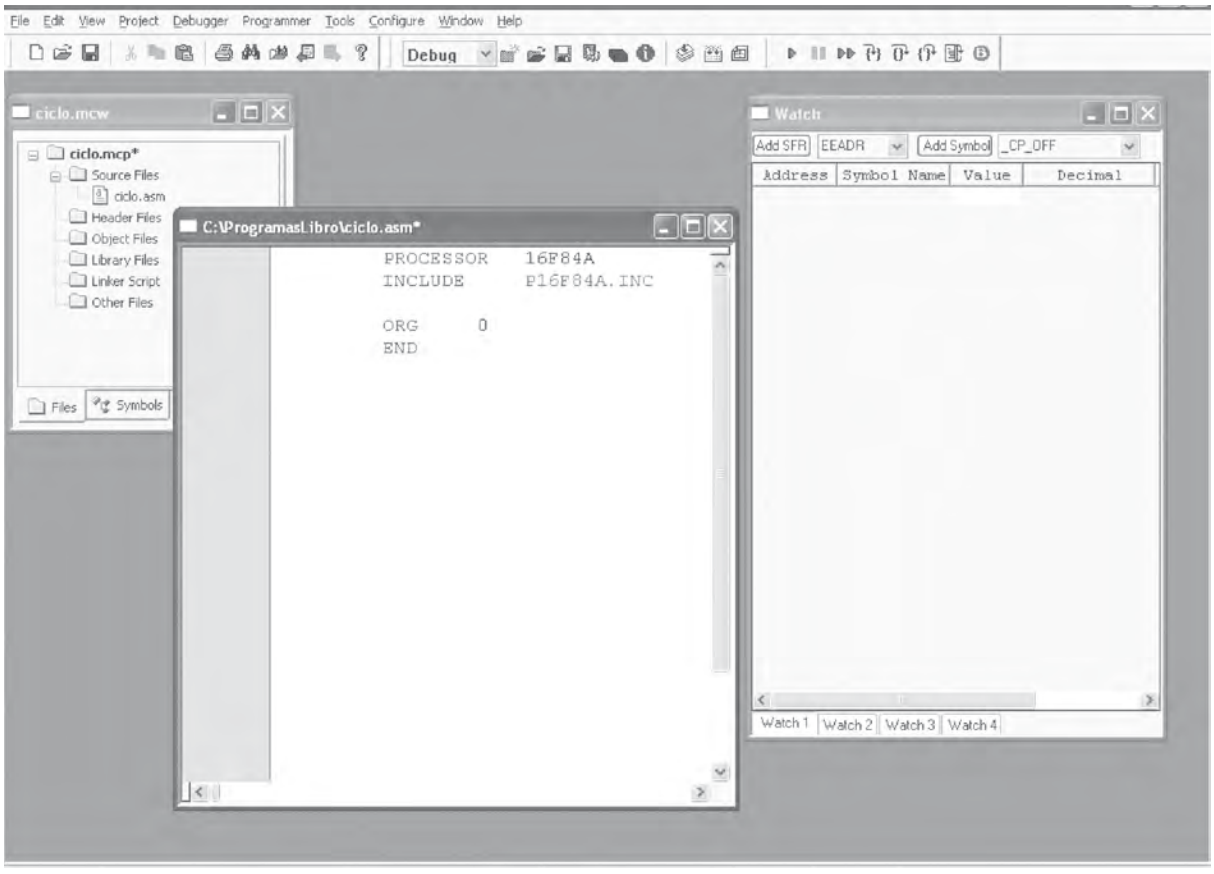


Figura 1.12. Ventana para verificar "variables".

11. Aunque todavía no hemos programado nada, daremos la orden de ejecutar la simulación seleccionando la opción *Run* en el menú *Debugger*. El MPLAB avisará que el proyecto no está actualizado y preguntará si lo queremos construir; responderemos que sí. Después el MPLAB preguntará qué tipo de manejo de memoria deseamos; responderemos que absoluto (figura 1.13). Al concluir el proceso obtendremos una serie de mensajes en la ventana de salida; el último será *BUILD SUCCED*. En este punto tenemos preparado el ambiente de desarrollo para efectuar los programas de ejemplo que usaremos al presentar el conjunto de instrucciones (figura 1.14). Cada vez que se modifique el código y se quieran observar los efectos en el simulador es necesario usar la opción *Build All* en el menú *Project*; como alternativas se tienen el icono *Build All* y las teclas Ctrl+F10.[2]

[2] Referencias al final del libro.

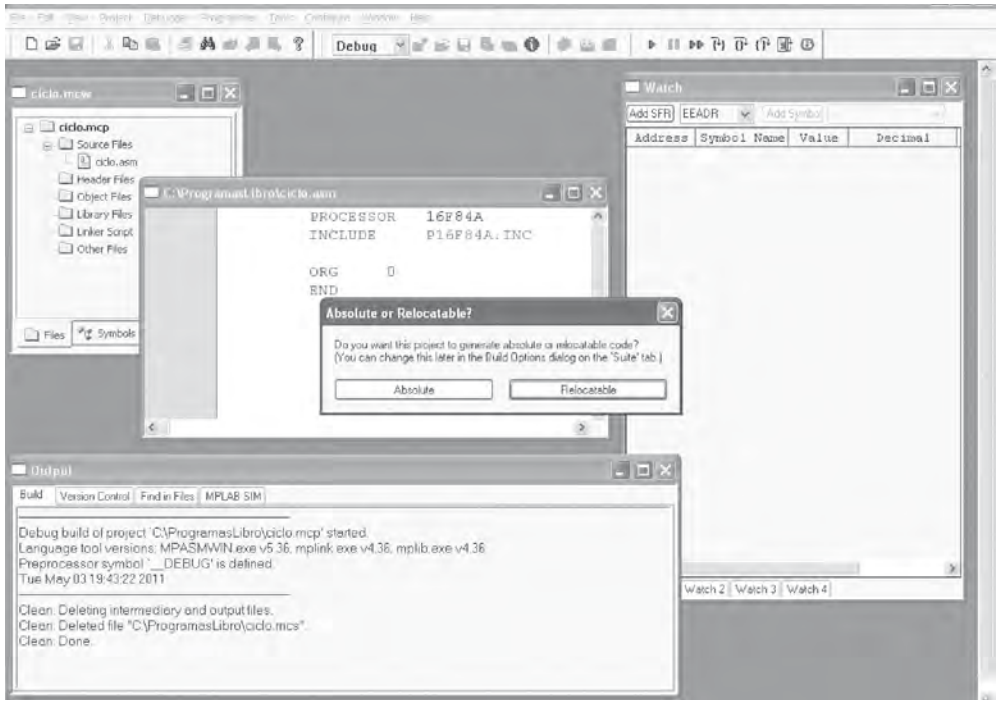


Figura 1.13. Manejo de memoria en forma absoluta; el programador establece y fija en qué localidades de memoria se almacenará el programa.

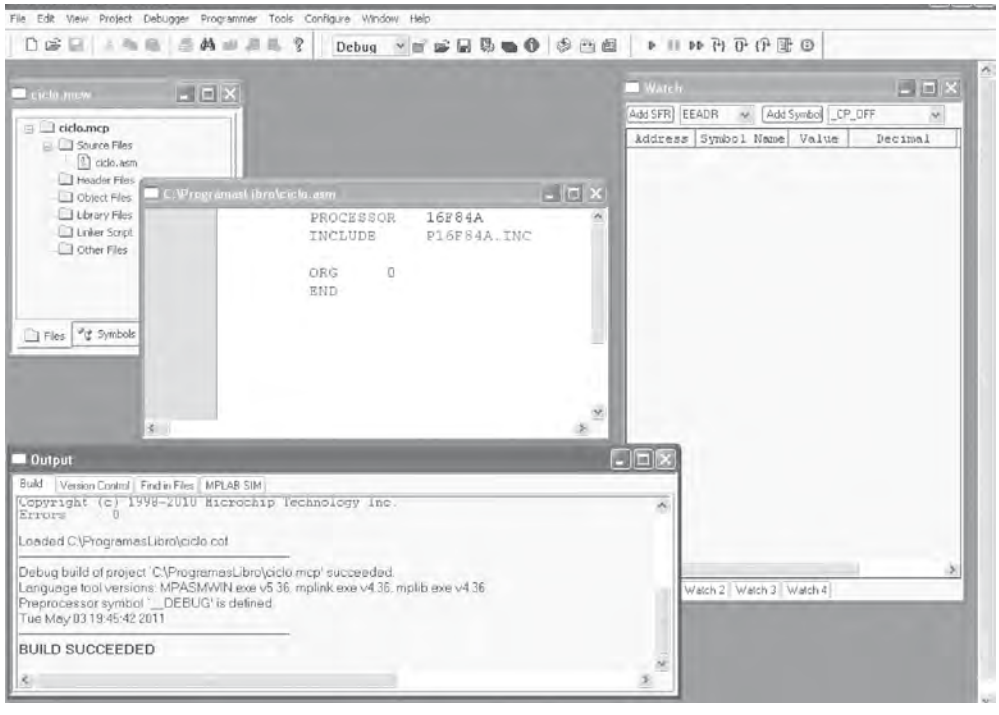


Figura 1.14. Programa vacío traducido exitosamente a código máquina.

1.2. Las instrucciones y su notación

1.2.1. ciclo.asm

Como se mencionó anteriormente, el primer programa que usaremos para estudiar el conjunto de instrucciones efectuará un ciclo de 10 iteraciones. Se seleccionó un ciclo porque, en prácticamente todos los programas, hay un grupo de procesos que se efectúa más de una vez.

En los lenguajes de alto nivel existen varias sentencias que pueden hacer esta tarea, por ejemplo: *for*, *while* y *do_while*. También se podría hacer uso combinado de sentencias de pregunta y salto incondicional (*if* y *goto*); esta última alternativa no se usa porque complicaría el código y "desperdiciaría" las sentencias especiales. Como el lector puede confirmar, ninguna de estas sentencias (o "instrucciones") aparece en la tabla 1.1.

Usaremos las instrucciones NOP, MOVLW, MOVWF, DECFSZ y GOTO, además de la directiva EQU para hacer nuestro programa. Como primera aproximación, éstas sirven para:

NOP: no hace nada.
MOVLW: guarda una constante en el registro de trabajo. El mnemónico hace referencia a mover, pero guardar o almacenar son palabras en español que reflejan mejor lo que sucede en el procesador.
MOVWF: guarda el contenido del registro de trabajo en una localidad de memoria.
DECFSZ: resta uno al contenido de una localidad de la memoria y salta la instrucción siguiente si el resultado es cero. En relación con el mnemónico, solemos decir "decrementar", pero esta palabra no está en el diccionario, por lo tanto usaremos decrecer.
GOTO: ordena que la ejecución del programa continúe en una línea en particular (ir a la línea).
EQU: ordena al MPLAB que remplace un texto por otro (equivale).

Para su descripción, el programa se dividió en secciones como se muestra en la tabla 1.5.

En las computadoras personales el sistema operativo administra la memoria; en los PIC, es el diseñador quien se encarga de esta tarea. Así, se seleccionó la localidad 020 para almacenar el número de veces que se ha efectuado el ciclo.

Buscando más claridad en la escritura de los programas, se prefiere remplazar las direcciones de memoria por nombres asociados al contenido de éstas; por ejemplo: MOVWF CONT, en lugar de MOVWF 020. La directiva EQU hace que el MPLAB remplace, antes de empezar a traducir el código en ensamblador a código máquina, la palabra CONT por el número hexadecimal 020. En la figura 1.15, además de la ventana que muestra el código en ensamblador, se observa otra titulada *Program Memory*; se nota que la directiva EQU no aparece en la memoria de programa y que CONT ha sido reemplazado por 0x20. Para abrir la ventana *Program Memory* se usó la opción del mismo nombre en el menú *View*; una vez abierta, se dio clic en la pestaña *Machine*.

Tabla 1.5. Secciones para describir el programa ciclo.asm

Sección	Código	Función
Antes del programa	CONT EQU 020	Establece una serie de equivalencias. Podría considerarse que se declaran las variables a usar en el código.
Inicialización	MOVLW .10 MOVWF CONT	Almacena un 10 en la localidad de la memoria de datos representada por CONT.
Cuerpo	CICLO NOP	Tareas que se desea repetir. En este caso sólo una instrucción que no hace nada, aunque requiere tiempo para ejecutarse.
Modificación-decisión	DECFSZ CONT,F GOTO CICLO	Cambia el contenido del control y dependiendo del resultado ejecuta el ciclo nuevamente o no.
	NOP NOP	Instrucciones agregadas para hacer pruebas.

Para inicializar el contador (CONT) a diez, requerimos dos instrucciones: una que cargue la constante o literal al registro de trabajo (MOVLW), y otra que pase el contenido del registro de trabajo a una localidad de memoria (MOVWF) (tablas 1.1 y 1.5). Para observar el efecto que tienen estas instrucciones, agregamos CONT (usando *Add Symbol*) y WREG (usando *Add SFR*) en la ventana *Watch* (figura 1.15). Para ejecutar el programa línea por línea se usa la opción *Step Into* del menú *Debuger* también se pueden usar la tecla F7 y el icono *Step Into* (al desplazar el cursor sobre los iconos aparece una etiqueta indicando su nombre o función).

El cuerpo principal del programa sólo tiene la instrucción NOP precedida por la etiqueta CICLO. Esta etiqueta funciona en forma conjunta con la instrucción GOTO, que a su vez sirve para modificar el flujo en los programas. "Normalmente" se ejecuta una instrucción tras otra; al llegar a GOTO CICLO se ordena al microcontrolador ejecutar la instrucción que está en la dirección representada por CICLO. Observe en la figura 1.15 (memoria de programa) que CICLO se reemplazó automáticamente por 0x2 – GOTO 0x2, en lugar de GOTO CICLO.

Antes de que se ejecute por primera vez la instrucción DECFSZ, el contenido de la localidad 020 representada por CONT es diez, después de la ejecución CONT valdrá 9; como este resultado es diferente de 0, no se ordenará saltar la siguiente instrucción. Como resultado de las siguientes iteraciones, CONT valdrá o almacenará: 8, 7, 6, 5, 4, 3, 2, 1 y 0. En la misma iteración en la que DECFSZ causa que CONT valga 0, el microcontrolador dejará de ejecutar (saltará) la instrucción GOTO: la flecha a la izquierda de las ventanas del programa en ensamblador y de la memoria de programa apuntará al penúltimo NOP (fin del ejemplo, figura 1.16).

De la figura 1.17 a la 1.21 se muestra la descripción, similar a la que hace Microchip en el manual de procesadores de mediano rango,[1] de las instrucciones hasta ahora utilizadas. En comparación con la descripción de Microchip, se retiraron: *Words* porque todas las instrucciones ocupan una palabra en la memoria de programa, y *Q Cycle Activity* porque esto será tratado en el capítulo de arquitectura.

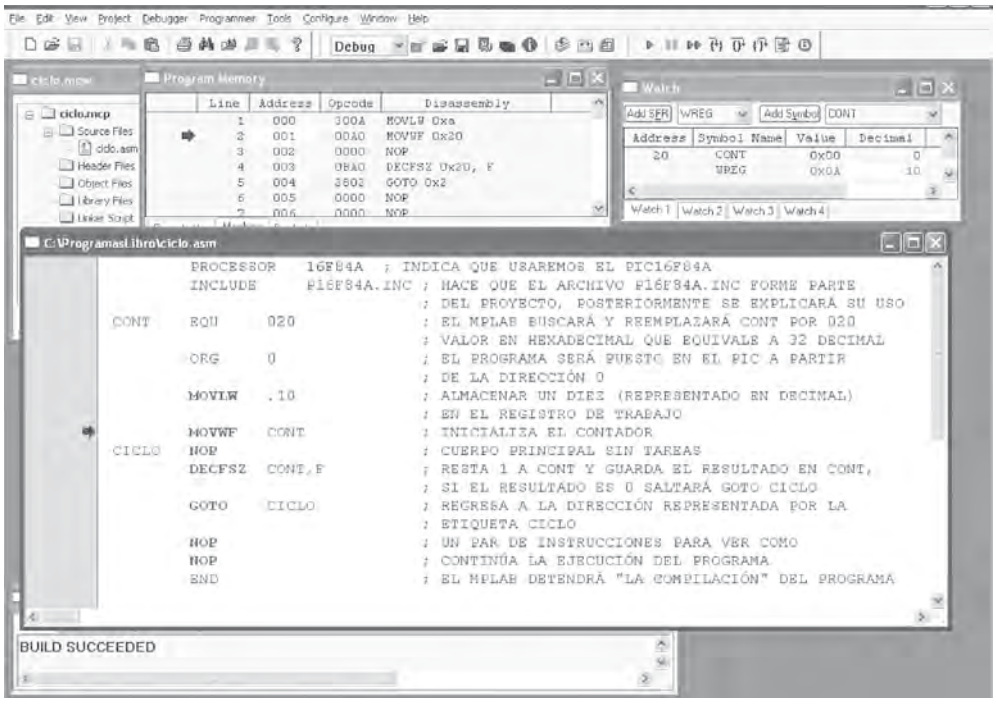


Figura 1.15. Programa ciclo.asm.

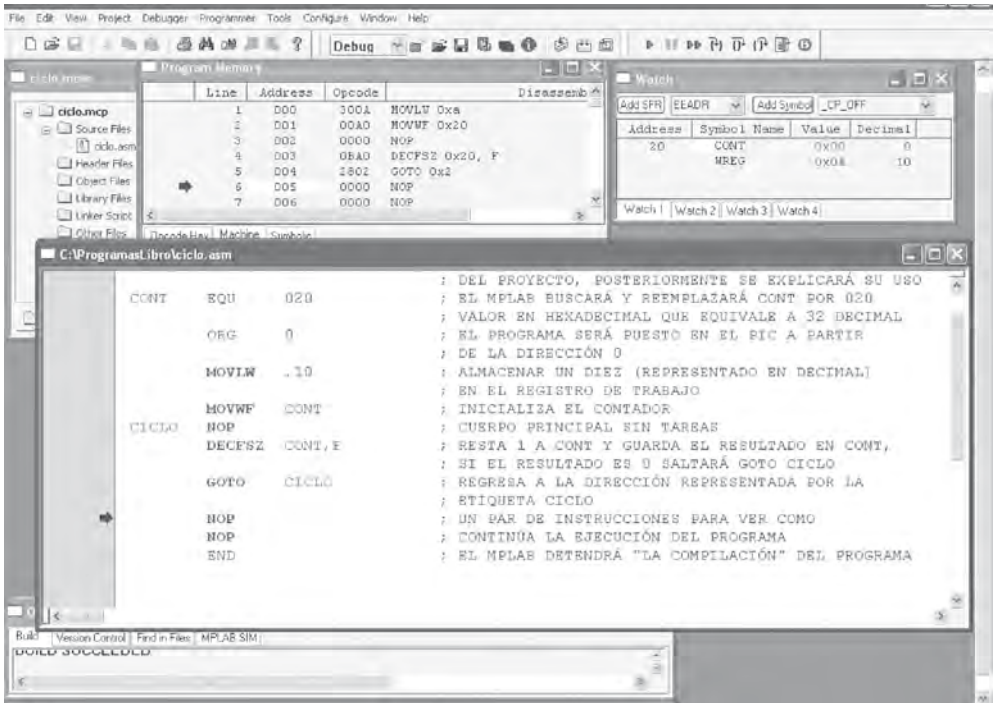


Figura 1.16. Pantalla al detener la ejecución del programa ciclo.asm.
En el microcontrolador la ejecución NO se detendría ahí, ni en la directiva END.

MOVLW Guarda k en W (mueve Literal a W)

Sintaxis:	[Etiqueta] MOVLW k
Operandos:	$0 \leq k \leq 255$
Operación:	$k \rightarrow W$
Registro de estado:	Sin cambios
Codificación:	11 0000 kkkk kkkk
Descripción:	La constante de 8 bits k se almacena en el registro de trabajo W.
Ciclos:	1

Figura 1.17. Descripción de la instrucción MOVLW (*Move Literal to W*).

MOVWF Mueve contenido de W a f

Sintaxis:	[Etiqueta] MOVWF f
Operandos:	$0 \leq f \leq 127$
Operación:	$(W) \rightarrow f$
Registro de estado:	Sin cambio
Codificación:	00 0000 1fff ffff
Descripción:	Almacena el contenido del registro de trabajo W en el registro f.
Ciclos:	1

Figura 1.18. Descripción de la instrucción MOVWF (*Move W to f*).

NOP No ejecuta operación

Sintaxis:	[Etiqueta] NOP
Operandos:	Ninguno
Operación:	Ninguna
Registro de estado:	Sin cambios
Codificación:	00 0000 0000 0000
Descripción:	El núcleo no efectúa tareas durante un ciclo de instrucción.
Ciclos:	1

Figura 1.19. Descripción de la instrucción NOP (*No Operation*).

DECFSZ — Decrece contenido de f, salta si es cero

Sintaxis:	[Etiqueta] DECFSZ f,d
Operandos:	$0 \leq f \leq 127$ $d \in [0,1]$
Operación:	(f) − 1 → destino, salta si resultado = 0
Registro de estado:	Sin cambio
Codificación:	00 1011 dfff ffff
Descripción:	Decrece el contenido del registro f. Si d es 0, el resultado se guarda en el registro de trabajo. Si d es 1, el resultado se guarda en el mismo registro de donde se obtuvo el dato, es decir, f. Si el resultado es 0, salta, es decir; durante el próximo ciclo, en lugar de ejecutar la instrucción que sigue en el código, se ejecuta una instrucción NOP.
Ciclos:	1 cuando no se produce salto, de lo contrario 2.

Figura 1.20. Descripción de la instrucción DECFSZ (*Decremet f, Skip if Zero*).

GOTO — Salto incondicional a k

Sintaxis:	[Etiqueta] GOTO k
Operandos:	$0 \leq k \leq 2047$
Operación:	k → PC<10:0> PCLATH<4:3> → PC<12:11>
Registro de estado:	Sin cambio
Codificación:	10 1kkk kkkk kkkk
Descripción:	Como primera aproximación diremos que: esta instrucción produce el salto incondicional a la dirección en la memoria de programa representada por k. En el capítulo de arquitectura se explicará por completo su funcionamiento.
Ciclos:	2

Figura 1.21. Descripción de la instrucción GOTO (*Go to*).

La notación en las figuras que describen las instrucciones corresponde a la empleada en la documentación de Microchip; su significado es:

[ ] Indica código opcional. En el programa ciclo.asm sólo la instrucción NOP es precedida por una etiqueta (CICLO).
→ Indica que se asigna o almacena en. Por ejemplo, k → W se leería: almacenar la constante k en el registro de trabajo.
( ) Significa el contenido. Por ejemplo, (W) indica el contenido del registro de trabajo.
< : > Habla de un subconjunto de bits. Por ejemplo, el contador de programa (PC, *Program Counter*) tiene un total de 13 bits, por lo que si nos referimos a todo el registro basta escribir PC; si se trata de los 11 bits menos significativos (como en la instrucción GOTO), escribimos PC<10:0>.

Símbolos de matemáticas se usan para establecer qué números pueden ser representados por los operandos: $0 \leq k \leq 255$ indica que k representa a cualquier entero entre 0 y 255 incluyendo a éstos. Para establecer los posibles valores de d (figura 1.20) se usó el símbolo perteneciente a la notación de conjuntos, en este caso d sólo puede valer 0 o 1. El nombre de los operadores lógicos se escribe entre dos puntos (.AND. por ejemplo).

1.2.2. mult_sumando.asm

Para continuar estudiando las instrucciones, haremos un programa básico de multiplicación: sumaremos el valor del multiplicando las veces que diga el multiplicador (tabla 1.6 y figura 1.22). En las primeras líneas se establece que usaremos el símbolo MDO y la dirección 020 para el multiplicando; MDR y 021 para el multiplicador; el resultado será almacenado en la localidad 022 y representado por el símbolo RESULT.

El programa se limita a multiplicar 10×5; para poder modificar los datos sería necesario contar con un teclado y un exhibidor (*display*). La forma de iniciar los valores de MDO y MDR es la misma que se usó para CONT en el ejemplo anterior. Para iniciar a cero RESULT se usó la instrucción CLRF (figura 1.23). Esta instrucción causa además que la bandera cero (*Z*) en el registro de estado (*STATUS*) tome el valor de 1. Para observar este cambio: en la ventana *Wacth* agregue *STATUS* usando la opción *Add SFR*. Puede facilitar la verificación si coloca además el apuntador sobre la opción *decimal* de la ventana *Watch*, da clic con el botón derecho y agrega la modalidad de visualización *Binary* (figura 1.22). El bit *Z* es el número 2, contando de derecha a izquierda a partir de 0 (registro 1.1).

TABLA 1.6. Programa mult_sumando.asm

```
                    ;C:\ProgramasLibro\mult_sumando.asm

          PROCESSOR   16F84A
          INCLUDE     P16F84A.INC

MDO       EQU     020       ; MULTIPLICANDO
MDR       EQU     021       ; MULTIPLICADOR
RESULT    EQU     022       ; RESULTADO

          ORG     0
          MOVLW   .10       ; INICIACIÓN DE VARIABLES
          MOVWF   MDO       ; MDO=10
          MOVLW   .5
          MOVWF   MDR       ; MDR=5
          CLRF    RESULT    ; RESULT=0

          MOVF    MDR,F
          BTFSC   STATUS,Z
          GOTO    FIN       ; EN CASO QUE MULTIPLICADOR SEA 0
          MOVF    MDO,W
CICLO     ADDWF   RESULT,F; SUMA AL RESULTADO EL VALOR DEL MULTIPLICANDO
          DECFSZ  MDR,F     ; DECRECE EL MULTIPLICADOR PARA LLEVAR LA CUENTA
          GOTO    CICLO

FIN       NOP
          GOTO    FIN
          END
```

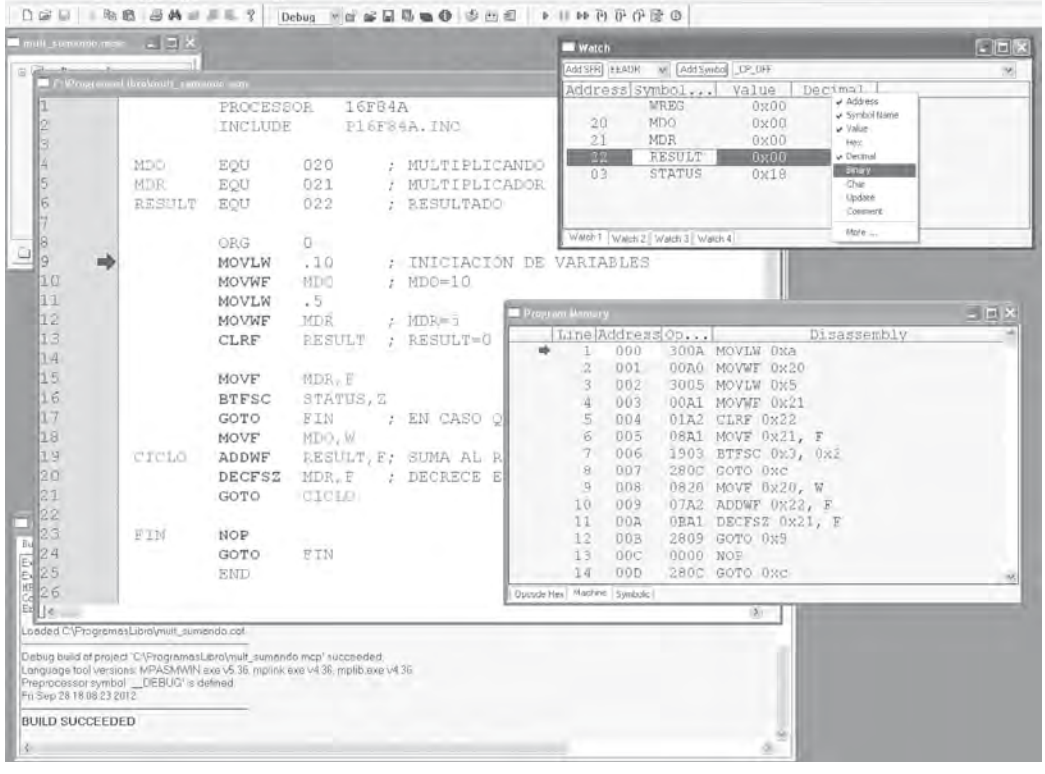


Figura 1.22. Programa mult_sumando.asm.

CLRF Pone ceros en los bits de f

Sintaxis:	[Etiqueta] CLRF f
Operandos:	$0 \leq f \leq 127$
Operación:	0x00 → f 1 → Z
Registro de estado:	Puede cambiar Z
Codificación:	00 0001 1fff ffff
Descripción:	Pone ceros en todos los bits del registro f; pone uno en la bandera Z.
Ciclos:	1

Figura 1.23. Descripción de la instrucción CLRF (*Clear f*).

Registro 1.1. Registro de estado (*STATUS*)

STATUS

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO’	PD’	Z	DC	C
bit 7	6	5	4	3	2	1	bit 0

bit 7 **IRP** Selector de bancos, usado en direccionamiento indirecto
1 = Selecciona bancos 2 y 3 (direcciones 0x100 a 0x1FF)
0 = Selecciona bancos 0 y 1 (direcciones 0x00 a 0xFF)

bit 6:5 **RP1:RP0** Selector de bancos, usado en direccionamiento directo
11 = Selecciona banco 3 (direcciones 0x180 a 0x1FF)
10 = Selecciona banco 2 (direcciones 0x100 a 0x17F)
01 = Selecciona banco 1 (direcciones 0x80 a 0xFF)
00 = Selecciona banco 0 (direcciones 0x00 a 0x7F)

bit 4 **TO’** Tiempo fuera (*Time-out*)
1 = Después de energizar (*power-up*) y de las instrucciones CLRWDT o SLEEP
0 = Se desbordó el reloj de vigilancia (*Watchdog Timer*).

bit 3 **PD’** Bajo consumo (*Power-down*)
1 = Después de energizar o a causa de la instrucción CLRWDT
0 = A consecuencia de la instrucción SLEEP (poner en modo de ahorro de energía)

bit 2 **Z** Cero
1 = El resultado de una operación aritmética o lógica es cero
0 = El resultado de una operación aritmética o lógica es diferente de cero

bit 1 **DC** Acarreo de dígito
1 = Hay acarreo al operar con los 4 bits menos significativos
0 = No hay acarreo de los 4 bits menos significativos

bit 0 **C** Acarreo, funciona en forma negada para las restas
1 = Hay acarreo (el resultado de una suma es mayor a 255)
0 = No hay acarreo (el resultado de una resta es menor que 0)

Aunque de antemano se sabe que el multiplicador es diferente de 0, se verifica esta situación en el código para ejemplificar un posible uso de la instrucción MOVF (figura 1.24). El programa dice MOVF MDR,F, que significa: toma el contenido de la localidad representada por MDR y muévelo a la localidad representada por MDR, ¿y para qué poner algo donde estaba? MOVF pondría a uno la bandera Z si el multiplicador valiera 0.

La instrucción BTFSC STATUS,Z dice: verifica el bit *Z* del registro *STATUS* y salta la siguiente instrucción si vale cero (figura 1.25). Ya que MDR es diferente de 0, la bandera *Z* está en 0 y la instrucción GOTO FIN no se ejecuta.

MOVF MDO,W almacena el valor del multiplicando en el registro de trabajo, mientras que ADDWF RESULT,F suma al contenido de RESULT el contenido de W y guarda lo obtenido en RESULT. Al ejecutar paso por paso observará que RESULT vale 0, 10, 20, 30, 40 y finalmente 50 (10 × 5). La descripción de ADDWF se muestra en la figura 1.26.

MOVF Mueve el contenido de f

Sintaxis:	[Etiqueta] MOVF f,d
Operandos:	$0 \leq f \leq 127$ $d \in [0,1]$
Operación:	(f) → destino
Registro de estado:	Puede cambiar Z
Codificación:	00 1000 dfff ffff
Descripción:	Si d es 0, el contenido de f se guarda en el registro de trabajo. Si d es 1, el contenido de f permanece ahí. Esta operación se usa para saber si el registro contiene un cero.
Ciclos:	1

Figura 1.24. Descripción de la instrucción MOVF (*Move f*).

BTFSC Verifica un bit y salta si es 0

Sintaxis:	[Etiqueta] BTFSC f,b
Operandos:	$0 \leq f \leq 127$ $0 \leq b \leq 7$
Operación:	Salta si (f<b>) = 0
Registro de estado:	Sin cambios
Codificación:	01 10bb bfff ffff
Descripción:	Si el bit b del registro f vale 0, salta, es decir: durante el próximo ciclo, en lugar de ejecutar la instrucción que sigue en el código, se ejecuta una NOP.
Ciclos:	1 cuando no se produce salto, de lo contrario 2.

Figura 1.25. Descripción de la instrucción BTFSC (*Bit Test, Skip if Clear*).

ADDWF	Suma contenidos de f y W
Sintaxis:	[Etiqueta] ADDWF f,d
Operandos:	$0 \leq f \leq 127$ $d \in [0,1]$
Operación:	(f) + (W) $\rightarrow$ destino
Registro de estado:	Pueden cambiar C, DC y Z
Codificación:	00 0111 dfff ffff
Descripción:	Suma el contenido de los registros f y de trabajo. Si d es 0, el resultado se guarda en el registro de trabajo. Si d es 1, el resultado se guarda en el mismo registro de donde se obtuvo el dato, es decir, f.
Ciclos:	1

Figura 1.26. Descripción de la instrucción ADDWF (*Add W and f*).

Ya que el programa suma el valor del multiplicando tantas veces como dice el multiplicador, se usó al multiplicador para llevar la cuenta de las iteraciones; se observa que MDR vale 5, 4, 3, 2, 1 y finalmente 0.

Ya que los microcontroladores ejecutan su programa mientras están energizados, se agregó un ciclo infinito al final del programa para proporcionar una especie de fin artificial.

Para finalizar este ejemplo, observe que en la línea 7 de la ventana *Program Memory* (figura 1.22) se reemplazó *STATUS* por 0x3 y *Z* por 0x2, es decir: el bit 2 del registro 3 (la figura 1.2 muestra la localización de *STATUS* en la memoria de datos; el registro 1.1 muestra su contenido). La directiva *INCLUDE* es la responsable de estas sustituciones; si el lector abre el archivo P16F84A.INC, observará que dentro de él se usó la directiva EQU para hacer el trabajo. Las etiquetas CICLO y FIN son reemplazadas automáticamente y en este ejemplo toman los valores 0x9 y 0xC (dé clic en las opciones *Symbolic* y *Machine* de la ventana memoria de programa). Aunque esta versión del MPLAB no lo muestra, F se sustituye por 1 y W por 0 en las instrucciones de las líneas 6, 9, 10 y 11.

Antes de pasar al siguiente ejemplo:

- Si desea cerrar este proyecto para hacer uno nuevo, use la opción *Close* del menú *Project*. Repita los pasos descritos anteriormente para crear el nuevo proyecto.
- Durante el desarrollo de este libro sólo se modificó el proyecto; el archivo del programa se guardó cambiándole el nombre mediante la opción *Save As...* del menú *File*. Después, en la ventana *mult_sumando.mcw* se eliminó el archivo mult_sumando.asm y se agregó el recientemente creado mult_sumando2.asm.

1.2.3. mult_sumando2.asm

Suponiendo que el programa que se diseña requiere ejecutar cuatro multiplicaciones, tenemos dos opciones: escribir cuatro veces el código de multiplicación o escribirlo una sola vez en una sección a la que llamaremos subrutina y hacer cuatro llamadas a este código (tabla 1.7 y figura 1.27). Al ejecutar mult_sumando2.asm paso por paso (oprimiendo F7), daría la impresión que CALL funciona igual que GOTO, es decir, la ejecución del programa continúa en la etiqueta MULT; la diferencia se nota al llegar a la instrucción RETURN.

TABLA 1.7. Programa mult_sumando2.asm

```
                    ;C:\ProgramasLibro\mult_sumando2.asm

          PROCESSOR   16F84A
          INCLUDE     P16F84A.INC

MDO       EQU     020     ; MULTIPLICANDO
MDR       EQU     021     ; MULTIPLICADOR
RESULT    EQU     022     ; RESULTADO
TEMPO     EQU     023

          ORG     0
          MOVLW   .10     ; INICIACIÓN DE VARIABLES
          MOVWF   MDO     ; MDO = 10
          MOVLW   .5
          MOVWF   MDR     ; MDR = 5
          CALL    MULT    ; PRIMERA LLAMADA A SUBRUTINA
          MOVF    RESULT,W; SE GUARDA EL RESULTADO ANTERIOR
          MOVWF   MDO     ; EN MDO PARA LA SIGUIENTE OPERACIÓN (MDO=50)
          MOVLW   .2
          MOVWF   MDR
          CALL    MULT    ; SEGUNDA LLAMADA PARA MULTIPLICAR
          CLRF    MDR     ; PARA LA TERCERA OPERACIÓN
          CALL    MULT    ; SÓLO SE MODIFICA EL MULTIPLICADOR
          MOVLW   .130    ; HABRÁ PROBLEMAS
          MOVWF   MDO
          MOVLW   .170
          MOVWF   MDR
          CALL    MULT
FIN       GOTO    FIN

MULT      CLRF    RESULT
          MOVF    MDR,W
          MOVWF   TEMPO
          BTFSC   STATUS,Z
          RETURN          ; EN CASO QUE MULTIPLICADOR SEA 0
          MOVF    MDO,W
CICLO     ADDWF   RESULT,F; SUMA AL RESULTADO EL VALOR DEL MULTIPLICANDO
          DECFSZ  TEMPO,F ; DECRECE EL MULTIPLICADOR PARA LLEVAR LA CUENTA
          GOTO    CICLO
          RETURN
          END
```

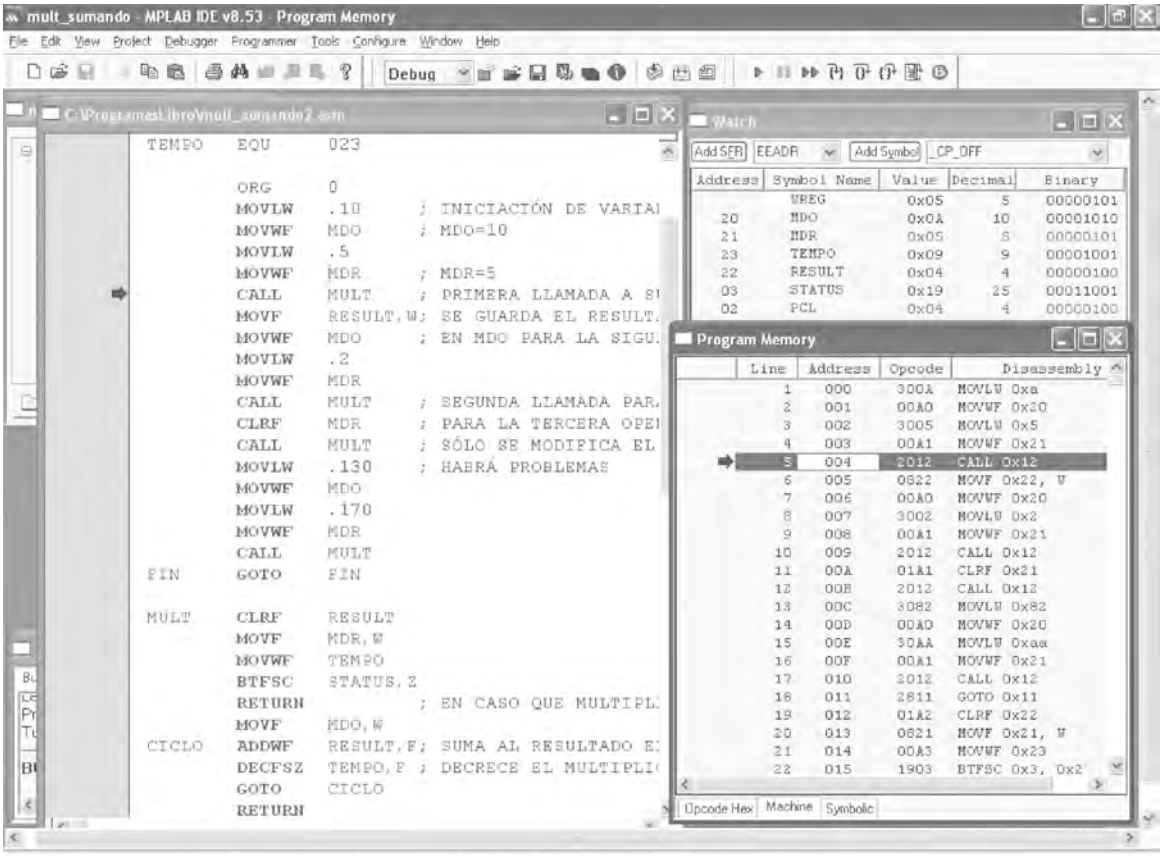


Figura 1.27. Programa mult_sumando2.asm.

Como ya se dejó ver, las instrucciones CALL y RETURN funcionan en forma conjunta. CALL ordena la ejecución de una subrutina, mientras que RETURN la finaliza y hace que el programa continúe en la instrucción que se encuentra después de la llamada. Observe que CALL requiere un operando, que es la dirección donde se encuentra la subrutina y suele ser representada por una etiqueta (en la figura 1.27 se observa que en este ejemplo MULT se sustituyó por 0x12, ventana *Program Memory*).

El contador de programa (PC *Program Counter*) es un registro especial de los procesadores y se encarga de "decir" qué instrucción se ejecuta (el contenido de los 8 bits menos significativos del PC se puede observar a través del registro especial PCL (figura 1.27, ventana *Watch*). Como se describe en la sección "Operación" de la figura 1.28, al ejecutarse una llamada a subrutina, el contenido del PC se incrementa y el resultado se almacena en el tope de pila (TOS *Top of Stack*). La pila es una estructura especial para almacenamiento temporal; en el caso de los PIC16 sólo almacena direcciones pertenecientes a la memoria de programa. Cuando se ejecuta un RETURN (figura 1.29), el contenido del tope de la pila pasa al contador de programa, lo que produce que continúe la ejecución del código en la instrucción siguiente a la llamada a subrutina.

CALL	Llamada a subrutina en k
Sintaxis:	[Etiqueta] CALL k
Operandos:	0 ≤ k ≤ 2047
Operación:	(PC) + 1 → TOS k → PC<10:0> (PCLATH<4:3>) → PC<12:11>
Registro de estado:	Sin cambio
Codificación:	10 0kkk kkkk kkkk
Descripción:	Como primera aproximación diremos que; esta instrucción llama a la subrutina que está en la dirección de la memoria de programa representada por k. En el capítulo de arquitectura se explicará por completo su funcionamiento.
Ciclos:	2

Figura 1.28. Descripción de la instrucción CALL (*Call subrutine*).

RETURN	Regresa de subrutina
Sintaxis:	[Etiqueta] RETURN
Operandos:	Ninguno
Operación:	TOS→ PC
Registro de estado:	Sin cambio
Codificación:	00 0000 0000 1000
Descripción:	Como primera aproximación diremos que esta instrucción ordena la salida de una subrutina. En el capítulo de arquitectura se explicará por completo su funcionamiento.
Ciclos:	2

Figura 1.29. Descripción de la instrucción RETURN (*Return from subrutine*).

En el programa mult_sumando2.asm se ejemplifica también que puede haber más de una instrucción RETURN en una subrutina, pero lógicamente sólo una se ejecutará por cada llamada. RETLW (figura 1.30) es una alternativa a RETURN, cumple sus mismas funciones y además carga una constante en el registro de trabajo antes de abandonar la subrutina. Por ahora sólo mencionaremos que existen rutinas especiales denominadas de interrupción, que su salida obedece a la instrucción RETFIE (figura 1.31), y que el tema de las subrutinas será retomado en capítulos posteriores.

RETLW	Regresa de subrutina, k en W
Sintaxis:	[Etiqueta] RETLW k
Operandos:	$0 \leq k \leq 255$
Operación:	$k \rightarrow W$ $TOS \rightarrow PC$
Registro de estado:	Sin cambio
Codificación:	11 0100 kkkk kkkk
Descripción:	Como primera aproximación diremos que esta instrucción almacena el valor de la constante k en el registro de trabajo W y sale de la subrutina. En el capítulo de arquitectura se explicará por completo su funcionamiento.
Ciclos:	2

Figura 1.30. Descripción de la instrucción RETLW (*Return whith Literal in W*).

RETFIE	Regresa de interrupción
Sintaxis:	[Etiqueta] RETFIE
Operandos:	Ninguno
Operación:	$TOS \rightarrow PC$ $1 \rightarrow GIE$
Registro de estado:	Sin cambio
Codificación:	00 0000 0000 1001
Descripción:	Como primera aproximación diremos que esta instrucción ordena la salida de una subrutina de interrupción. En el capítulo de arquitectura se explicará por completo su funcionamiento.
Ciclos:	2

Figura 1.31. Descripción de la instrucción RETFIE (*Return from interrupt*).

En mult_sumando.asm el valor del multiplicador se modifica mientras se efectúa la multiplicación. Para evitar esta situación, en este ejemplo se usa la localidad de memoria 023 etiquetada como TEMPO. Esto implicó reemplazar MOVF MDR,F por MOVF MDR,W y agregar la instrucción MOVWF TEMPO. Observe que la determinación de si el multiplicador es cero sigue funcionando con la instrucción MOVWF intercalada, ya que ésta no modifica el registro de estado (en particular la bandera *Z*).

En cuanto a las cuatro llamadas a MULT, observe que: al regresar de la primera RESULT vale 50; al regresar de la segunda vale 100; cuando MDR vale 0, RESULT también, y la rutina se abandona por el primer RETURN; la última llamada devuelve un resultado errado ya que

130 × 170 = 22100, y el valor más grande que cabe en un byte es 255. Para esta última llamada, observe que la segunda vez que se ejecuta ADDWF (equivalente a 130 × 2 = 260) enciende el bit C del registro de estado (bit 0 en STATUS), aviso de que se desbordó RESULT.

1.2.4. multiplicación.asm

Las desventajas principales del algoritmo anterior son: 1) Tiempo muy diferente de ejecución dependiendo de los datos (por ejemplo, considere la diferencia entre 2 × 64 y 64 × 2). 2) El resultado de la operación debe ser menor a 256 (se requieren dos bytes para almacenar el producto de dos números de un byte cada uno). El algoritmo que se enseña en las primarias es mejor, y aplicarlo en sistema binario es más fácil que hacerlo en sistema decimal porque sólo requiere saber las "tablas del 0 y del 1". En la parte superior de la figura 1.32 se muestra la multiplicación en sistema binario de 130 por 170.

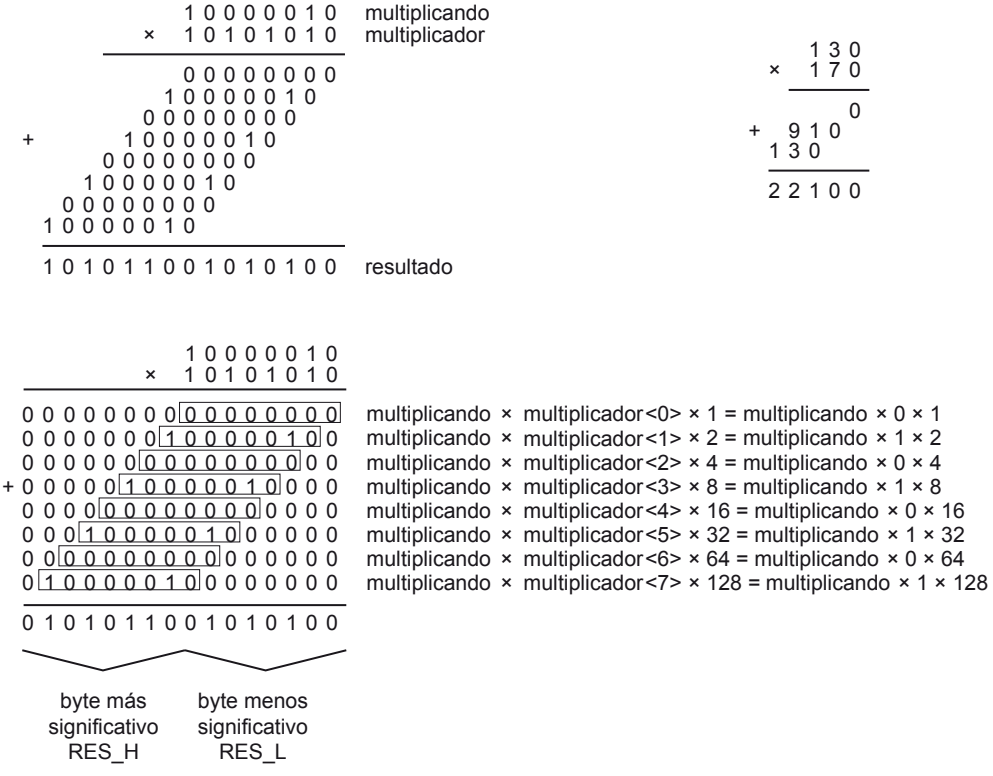


Figura 1.32. Multiplicación de 130 por 170. En la parte superior izquierda, una modalidad binaria del algoritmo que solemos aprender de niños. En la parte inferior, la misma receta con una escritura que facilitará plantear un código en ensamblador.

La receta que alguna vez memorizamos es algo parecido a:

- Usando las tablas, hacer la multiplicación del digito más a la derecha del multiplicador (unidad en el sistema decimal) por el multiplicando y escribir el resultado.
- Para el resto de los dígitos del multiplicador, uno por uno, y de derecha a izquierda usando las tablas hacer la multiplicación del dígito por el multiplicando, y escribir el resultado desplazado un lugar a la izquierda en comparación con el resultado anterior.
- Sumar los resultados parciales.

Un poco de observación en la parte de sumar los resultados parciales desplazados, permite notar que lo que hacemos es: resultado = 130 × 0 + 130 × 70 + 130 × 100. En decimal, multiplicar por diez equivale a desplazar los números una posición a la izquierda. En sistema binario, desplazar a la izquierda una posición equivale a multiplicar por 2 (parte baja de la figura 1.32).

Considerando que en el caso binario los posibles resultados parciales son cero o "una versión desplazada del multiplicando", y analizando la parte baja de la figura 1.32, se deduce que la receta para obtener los resultados parciales, se puede mejorar si se desplaza de izquierda a derecha. En la versión de la primaria, hay un caso que no requiere desplazamiento: implicaría un segmento de código para el caso único y luego un ciclo de 7 iteraciones (si manejamos datos de 8 bits como es el caso con los PIC16). Considerando un resultado de 2 bytes, y desplazando de izquierda a derecha (parte inferior de la figura 1.32), se observa que todos los resultados parciales requieren desplazamiento: 1 para el resultado parcial del bit 7, dos para el del bit 6... 8 para el parcial del bit 0: implica un solo ciclo de 8 iteraciones. La tabla 1.8 y la figura 1.33 muestran el código que implementa esta receta.

TABLA 1.8. Programa multiplicación.asm

```
          ;C:\ProgramasLibro\multiplicación.asm

PROCESSOR   16F84A
INCLUDE     P16F84A.INC

CBLOCK  020
 MDO             ; MULTIPLICANDO
 MDR             ; MULTIPLICADOR
 RES_L           ; RESULTADO PARTE MENOS SIGNIFICATIVA
 RES_H           ; RESULTADO PARTE MÁS SIGNIFICATIVA
 CONT            ; CONTADOR
 TEMPO           ; PARA NO MODIFICAR MDR
ENDC

ORG     0
MOVLW   .130
MOVWF   MDO
MOVLW   .170
```

```
        MOVWF   MDR
        CALL    MULT
FIN     GOTO    FIN

MULT    CLRF    RES_L
        CLRF    RES_H
        MOVLW   .8
        MOVWF   CONT
        MOVF    MDR,W
        MOVWF   TEMPO
        MOVF    MDO,W
CICLO   RRF     TEMPO,F
        BTFSC   STATUS,C
        ADDWF   RES_H,F
        RRF     RES_H
        RRF     RES_L
        DECFSZ  CONT,F  ; DECRECE CONT PARA CONTROLAR ITERACIONES
        GOTO    CICLO
        RETURN
        END
```

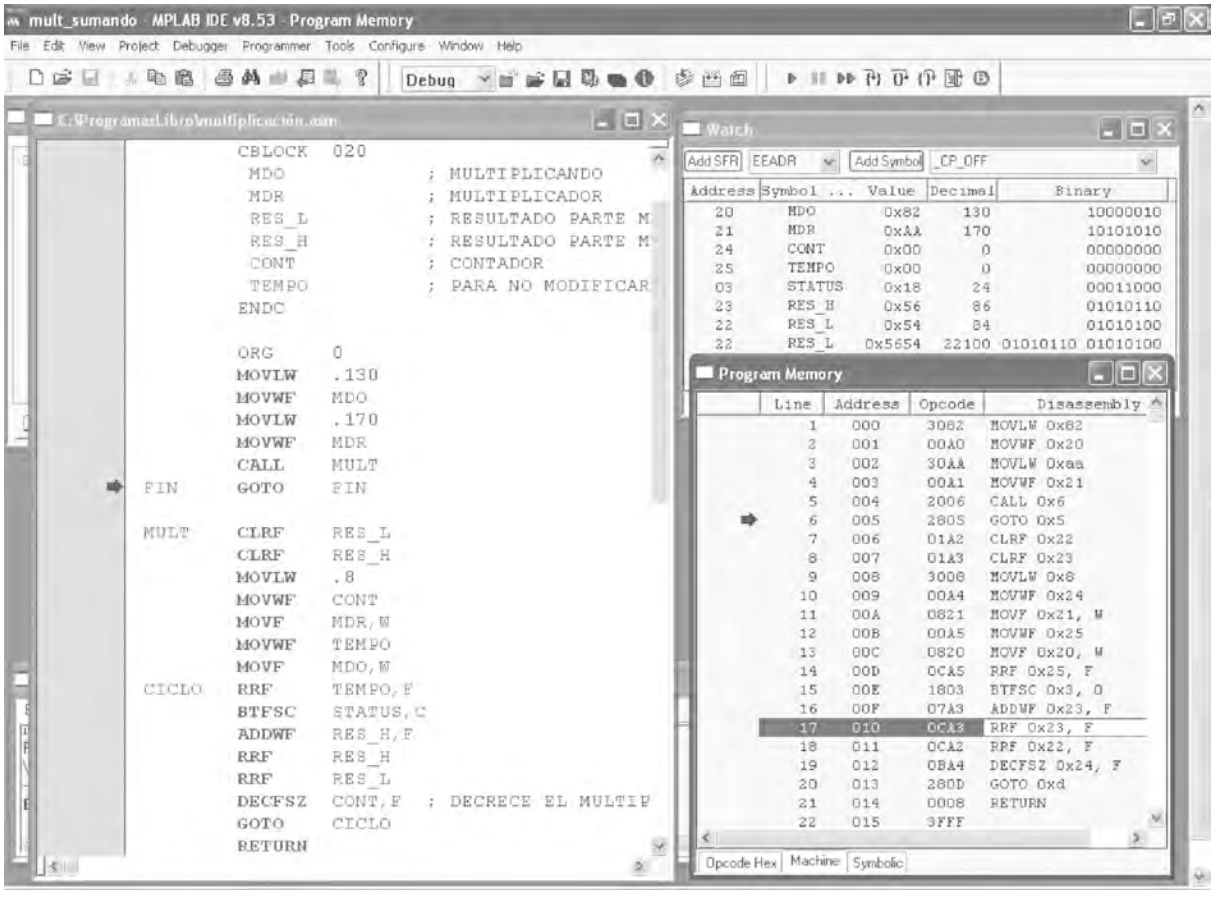


Figura 1.33. Programa multiplicación.asm.

El primer cambio que se aprecia respecto a los códigos anteriores está en la sección de definición de equivalencias: en lugar de la directiva EQU, se usaron CBLOCK y ENDC. Enseguida de CBLOCK se escribe la dirección inicial del bloque de "variables" (en el capítulo de arquitectura se explica por qué no se trata de las variables típicas de programación). El efecto de estas directivas se puede observar en la columna *Address* de la ventana *Watch* en la figura 1.33. En esta misma ventana observe que RES_L aparece dos veces, la segunda se modificó para mostrar 16 bits (el resultado completo); para hacer la modificación se colocó el apuntador sobre RES_L y se dieron clic izquierdo y luego derecho; al aparecer el menú se seleccionó *Properties...*; al abrirse la ventana se seleccionaron *Size* : 16 bits y *Byte Order: High:Low*. El lector ya habrá notado por qué los nombres RES_H y RES_L para las partes más y menos significativas del resultado, respectivamente.

Ya que no hay instrucciones que permitan sumar más de dos operandos, los resultados parciales se suman en cada iteración. Observe también que primero se efectúa la suma (ADDWF), después el desplazamiento (dos instrucciones RRF) y que en realidad se desplaza el resultado.

RRF TEMPO,F (figura 1.34) tiene como objetivo analizar bit por bit el contenido del multiplicador: en caso de ser 1, se efectúa la suma del resultado parcial (no tiene caso sumar un 0). Ya que RRF va colocando el bit de interés en *C* dentro de STATUS, la instrucción BTFSC se aplica a éstos.

RRF Rota a derecha pasa por C

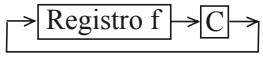
Sintaxis:	[Etiqueta] RRF f,d
Operandos:	$0 \leq f \leq 127$ $d \in [0,1]$
Operación:	
Registro de estado:	Puede cambiar C
Codificación:	00 1100 dfff ffff
Descripción:	El contenido del registro f es rotado a la derecha a través de la bandera de acarreo (el bit 0 pasa a C, el bit 1 pasa al 0, el 2 al 1, etc.). Si d es 0, el resultado se guarda en el registro de trabajo. Si d es 1, el resultado se guarda en el mismo registro de donde se obtuvo el dato, es decir, f.
Ciclos:	1

Figura 1.34. Descripción de la instrucción RRF (*Rotate Right f through Carry*).

Las instrucciones RRF RES_H y RRF RES_L producen el desplazamiento. Note que el bit de acarreo sirve como enlace entre los dos registros. Finalmente observe que no se escribió el destino para estas instrucciones, y que por omisión es 1 (equivalente a F; RRF 0x23,F en la ventana *Program Memory* de la figura 1.33).

1.2.5. restas.asm

Este ejemplo es muy simple y tiene como único objetivo que el lector observe que el bit de acarreo funciona en forma inversa cuando se trata de restas (figuras 1.35 y 1.36).

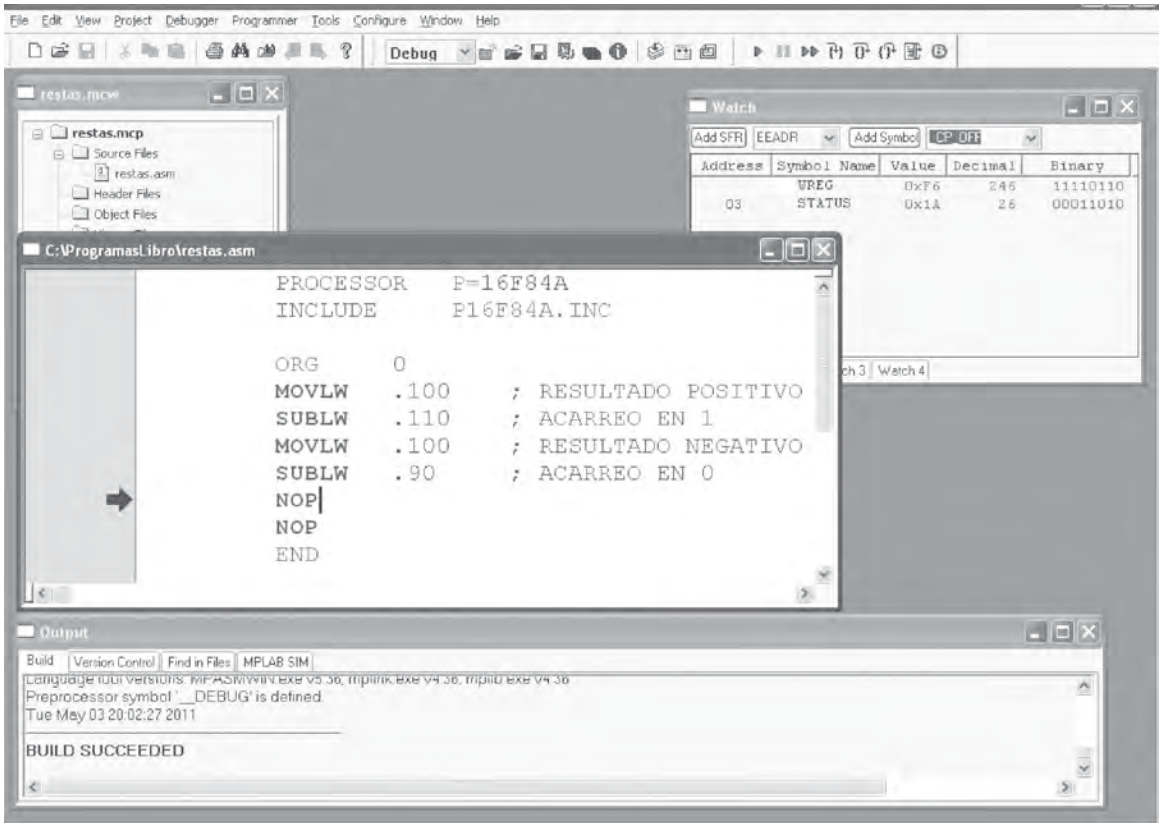


Figura 1.35. Ejemplo de restas.asm.

SUBLW k menos contenido de W

Sintaxis:	[Etiqueta] SUBLW k
Operandos:	$0 \leq k \leq 255$
Operación:	$k - (W) \rightarrow W$
Registro de estado:	Pueden cambiar C, DC y Z
Codificación:	11 1100 kkkk kkkk
Descripción:	A la constante de 8 bits k se le resta el contenido del registro de trabajo W; el resultado se almacena en W.
Ciclos:	1

Figura 1.36. Descripción de la instrucción SUBLW (*Subtract W from literal*).

1.3. Descripción resumida del conjunto de instrucciones completo

Con el fin de tener una sección de consulta, las figuras siguientes muestran la descripción del conjunto de instrucciones en orden alfabético.

ADDLW k más contenido de W

Sintaxis:	[Etiqueta] ADDLW k
Operandos:	$0 \leq k \leq 255$
Operación:	$k + (W) \rightarrow W$
Registro de estado:	Pueden cambiar C, DC y Z
Codificación:	11 1110 kkkk kkkk
Descripción:	La constante de 8 bits k se suma al contenido del registro de trabajo W; el resultado se almacena en W.
Ciclos:	1

Figura 1.37. Descripción de la instrucción ADDLW (*Add Literal and W*).

ADDWF Suma contenidos de f y W

Sintaxis:	[Etiqueta] ADDWF f,d
Operandos:	$0 \leq f \leq 127$ $d \in [0,1]$
Operación:	(f) + (W) → destino
Registro de estado:	Pueden cambiar C, DC y Z
Codificación:	00 0111 dfff ffff
Descripción:	Suma el contenido de los registros f y de trabajo. Si d es 0, el resultado se guarda en el registro de trabajo. Si d es 1, el resultado se guarda en el mismo registro de donde se obtuvo el dato, es decir, f.
Ciclos:	1

Figura 1.38. Descripción de la instrucción ADDWF (*Add W and f*).

Índice alfabético

A
ADCON0, registro, 23, 150, 327
ADCON1, registro, 23, 150, 326
ADDLW, instrucción, 16, 52, 69, 148
ADRES, registro, 23, 150, 327
alias, 317
ADDWF, instrucción, 16, 42, 52, 69, 148
ambiente de desarrollo, 24, 489
ANDLW, instrucción, 16, 53, 69, 148
ANDWF, instrucción, 16, 53, 69, 148
animate, 87, 98, 176
archivo de registros, 23, 149, 150, 157
arquitectura, 145, 187
asíncrono, 249, 255

B
b, operando, 22, 24
banco, 21, 23, 40, 149, 151
bandera, 16, 22, 181
BCF, instrucción, 16, 53, 69, 109, 148
binario, sistema de numeración, 17, 19
bit más significativo, 24
bit menos significativo, 24
bits de configuración, 95, 119, 132, 330, 429, 446, 450, 470, 508
BODEN, 84
BOR, Brown-out Reset, 80, 84
botón, 124
breack point, 88, 344, 450
BSF, instrucción, 16, 54, 69, 109, 148
BTFSC, instrucción, 16, 41, 54, 69, 148
BTFSS, instrucción, 16, 54, 69, 148

C
C, bandera de acarreo, 22, 40, 52, 63, 64
CALL, instrucción, 16, 43, 55, 69, 148, 158, 163
captura, 341, 347
CBLOCK, directiva, 50, 153
CCP, 341
CCPRxH, registro, 23, 150, 341, 343, 347
CCPRxL, registro, 23, 150, 341, 343, 347, 355
CCPxCON, registro, 23, 150, 341, 342
ciclo de instrucción, 16, 67, 69, 148, 190
CLRF, instrucción, 16, 38, 55, 69, 148
CLRW, instrucción, 16, 55, 69, 148
CLRWDT, instrucción, 16, 40, 56, 69, 89, 148
COMF, instrucción, 16, 56, 69, 148
comparación, 341, 343
componente electrónico
- 24LC512 (memoria), 298
- 2N7000 (transistor), 120
- 2SK1445 (transistor), 357
- AT25128 (memoria), 273
- BS250 (transistor), 116, 120
- DS1307 (reloj), 287
- HD44780 (manejador para exhibidor), 379
- HS311 (servo motor), 361
- JHD12864A (exhibidor gráfico de cristal líquido), 406
- JHD162A (exhibidor alfanumérico de cristal líquido), 379
- KS108 (manejador para exhibidor), 406
- IR333C (diodo infrarrojo), 135
- LM35DZ (sensor de temperatura), 335
- MAX232 (TTL - RS232 - TTL), 257, 433, 473
- MAX233 (TTL - RS232 - TTL), 257, 263
- PIC16F628, 92, 182, 185, 211, 269, 343, 432
- PIC16F72, 225, 328, 372
- PIC16F84, 21, 25, 30, 35, 68, 89, 149
- PIC16F877, 99
- PIC16F88, 21
- PIC16F887, 68, 81, 92, 100, 149, 152, 165, 166, 200, 218, 279, 301, 376, 387, 408
- PT331C (fototransistor), 135
- S2010L (SCR), 335

contador de programa, 22, 38, 44, 80, 158, 163, 166, 188, 190
conversión analógico digital, 313
- circuito de muestreo y retención, 323, 325
- multiplexor a la entrada, 323
- número de bits, 320, 322
- por aproximaciones sucesivas, 323
- referencias de voltaje, 324
- resolución, 322

D
d, operando, 22, 24
DC, bandera de acarreo de dígito, 16, 40
DECF, instrucción, 16, 57, 69, 148
DECFSZ, instrucción, 16, 33, 37, 57, 69, 148
debugger, 30, 87, 97, 176, 257, 499
decimal, sistema de numeración, 17
depurador, 30, 87, 97, 176, 257, 499
direccionamiento, 151
- directo, 151
- indirecto, 156
- inmediato, 158

directivas
- CBLOCK, 50, 153
- END, 30
- ENDC, 50, 153
- EQU, 33, 153, 504
- INCLUDE, 30, 42
- ORG, 30, 161, 171
- PROCESSOR, 30
- RES, 502
- UDATA, 502

drenaje abierto, 121, 284

E
END, directiva, 30
ENDC, directiva, 50, 153
EQU, directiva, 33, 504
esclavo, 249, 266, 269, 272, 273, 284
estado, registro de, 23, 38, 40, 80, 150
estimulo, 97, 176, 512
etiqueta, 34, 153, 155, 158, 160
exhibidores
- de 7 segmentos, 114
- de cristal líquido, 379
- alfanuméricos, 379
- gráficos, 406

F
f, operando, 22, 24, 151
filtro
  anti alias, 318
  pasa bajas, 319, 339
formato Intel de 32 bits, 446
frecuencia de muestreo, 315, 346
FSR, registro, 23, 150, 156
fuente de alimentación, 526
full duplex, 249, 259

G
GOTO, instrucción, 16, 33, 58, 69, 148, 158, 163
Gray, código, 136

H
half duplex, 249, 266
herramientas, 523
hexadecimal, sistema de numeración, 17, 20

I
I²C, 271, 284
ICSP, 423, 428
IDE, 24, 490
INCF, instrucción, 16, 58, 69, 148
INCFSZ, instrucción, 16, 59, 69, 148
INCLUDE, directiva, 30, 42
INDF, registro, 23, 150, 156
instalación, 24, 490
instrumentación, 523, 526
INTCON, registro, 23, 150, 182, 349
intervalo de muestreo, 313
IORLW, instrucción, 16, 59, 69, 148
IORWF, instrucción, 16, 60, 69, 148
instrucciones, conjunto de, 15, 33, 52, 69, 148
interrupciones, 159, 176, 349, 353
IRP, 40, 156

K
k, operando, 22, 24

L
lectura-modificación-escritura, instrucciones de, 109
levantamiento débil, 133, 284
Linux, 489
LCD, 379
LSB, 24

M
maestro, 249, 266, 272, 273, 284
mapeo, 149
materiales complementarios, 523
MCLR´ - master clear, 80, 96, 98
Memoria, 21, 188
  memoria de datos, 21, 149
  memoria de programa, 21, 158
mnemónicos, 16, 17
modulación por ancho de pulso, 354
MOVF, instrucción, 16, 60, 69, 148
MOVLW, instrucción, 16, 33, 60, 69, 148
MOVWF, instrucción, 16, 33, 61, 69, 148
MPLAB, 14, 24
  MPLAB SIM, 30
  MPLAB X, 489
MSB, 24
multímetro, 528

N
NOP, instrucción, 16, 33, 61, 69, 148

O
operandos, 16
OPTION_REG, registro, 23, 133, 150, 183
ORG, directiva, 30, 161, 171
OSCCAL, registro, 23, 73, 150
oscilador, 67, 79
  con cristal, 74, 79
  con resonador cerámico, 74, 79
  externo, 76, 79
  RC externo, 69, 79
  RC interno, 72, 79
  selección, 77, 79
osciloscopio, 528

P
página, 158
paralelismo, 145, 190
PCL, registro, 23, 44, 150, 164
PCLATH, registro, 23, 150, 159, 162
PCON, registro, 23, 80, 150
PD', bandera de bajo consumo, 40, 80, 87
pila, 44, 159, 184
polling, 176
POR, Power-on Reset, 80, 82
PORT, 23, 91, 133, 150
preguntando, 176
PROCESSOR, directiva, 30
programación LVP, 429, 434
programadores, 423, 425, 428, 432, 472
programas
  AD00.asm, 329
  aleatorio.asm, 225
  call00.asm, 165
  call01.asm, 166
  carrito.asm, 376
  ciclo.asm, 33
  compara00.asm, 343
  compara01.asm, 345
  compara02.asm, 347
  controlR.asm, 372
  desbordar_pila.asm, 185
  dir_directo.asm, 152
  enciende162A.asm, 387
  equ_cblock.asm, 154
  esc24LC512.asm, 301
  esclavo.asm, 269
  escritura.asm, 279
  FBorra (función en C para PC), 477
  FIni (función en C para PC), 476
  FFin (función en C para PC), 476
  FLeeID (función en C para PC), 479
  FPrograma (función en C para PC), 482
  Galileo.asm, 218
  GalileoG.asm, 350
  goto00.asm, 160
  goto01.asm, 160
  goto02.asm, 160
  goto03.asm, 160
  goto04.asm, 161
  gráfico.asm, 408
  i2c esclavo.asm, 305
  inicializar.asm, 157
  interrumpiendo.asm, 180
  lectura.asm, 283
  lógica_prog.asm, 451
  m887_e88.asm, 306
  maestro.asm, 268
  mapa.asm, 411
  mem_color.asm, 235
  memoria.asm, 239
  memoria0.asm, 230
  México.asm, 399
  México_4bits.asm, 402
  multiplicación.asm, 48
  mult_sumando.asm, 39
  mult_sumando2.asm, 43

n_cuadrada.asm, 171
preguntando.asm, 177
prog_borrar.asm, 435
prog_lee_id.asm, 439
programador.asm, 457
Puente.asm, 369
puertos01.asm, 94
puertos01b.asm, 111
puertos02.asm, 97
PWM00.asm, 356
PWM01.asm, 358
recorrido.asm, 393
reloj01.asm, 198
reloj02.asm, 200
reloj i2c.asm, 290
respaldando.asm, 186
restas.asm, 51
servo00.asm, 365
servoHS311.asm, 362
siete_seg.asm, 118
siete_seg_tm.asm, 126
siete_seg_tm2.asm, 168
tabla_7s.asm, 167
temperatura.asm, 332
terminales_comp.asm, 99
texto.asm, 390
TMR0_0.asm, 194
TMR0_1.asm, 204
tonos.asm, 205
USART_RX.asm, 262
USART_RX_ERR.asm, 264
USART_TX.asm, 256
vel_bici.asm, 174
XY.asm, 417
wdt84.asm, 88
proyecto, generar un, 25, 496
puerto, 91, 133
características eléctricas, 102
punto de paro, 88, 344, 450
PWM, 341, 354
PWRT, Power-up Timer, 85

R

RCREG, registro, 23, 150, 260
RCSTA, registro, 23, 150, 254
rebote, 123
registro, 22
archivo de (*Register File*), 23, 150
de estado, 22, 40, 80, 147, 151, 153, 156, 186
de trabajo, 22, 33, 36, 38, 147, 186
de uso especial, 22, 150, 155
de uso general, 22, 155
reinicio, 67, 80
al encender, 82
maestro, 80
por baja tensión, 84
por reloj de vigilancia, 87
reloj, 67, 249, 253, 274, 276, 287, 294, 310
RES, directiva, 502
RETFIE, instrucción, 16, 45, 61, 69, 148, 158, 164, 179
RETLW, instrucción, 16, 45, 62, 69, 148, 158, 164, 167, 170
RETURN, instrucción, 16, 43, 62, 69, 148, 158, 164
RLF, instrucción, 16, 63, 69, 148
RP1 y RP0 selectores de banco, 40, 151
rotación, 63, 134
RRF, instrucción, 16, 50, 63, 69, 148
RS232, 249, 429. 434

S

saltos calculados, 166
SFRs, registros de uso especial, 23, 150, 155
sensor, 134, 313, 336, 338
señal analógica, 313, 320, 322, 338
serie, comunicación en, 249, 266, 273, 384, 428
simulador de analizador lógico, 94, 176, 225, 344, 361, 437, 511
sincronizado, 76, 249, 255, 266, 271, 273, 284
SLEEP, instrucción, 16, 40, 64, 69, 81, 148
soldadura, 523, 524
SPBRG, registro, 23, 150,
SPI, 273
SSP, 271
SSPADD, registro, 23, 150, 297, 310
SSPBUF, registro, 23, 150, 278, 310
SSPCON, registro, 23, 150, 273, 293, 295
SSPSTAT, registro, 23, 150, 273, 276, 296
STATUS, registro de estado, 23, 38, 40, 80, 150
stopwatch, 87, 210, 230, 256, 344, 456
SUBLW, instrucción, 16, 64, 69, 148
SUBWF, instrucción, 16, 65, 69, 148
SWAPF, instrucción, 16, 65, 69, 148

T

T1CON, registro, 23, 150, 213
tabla, 166, 411
tecla, 123, 125
temporizadores, 193
cero, 193
uno, 212
dos, 223
teorema de muestreo, 314
tiempo de ejecución, 190
TMR1H, registro, 23, 150, 212
TMR1L, registro, 23, 150, 212
TO', bandera de tiempo fuera, 40, 80, 87
tonos, 203
TOS, 44, 55, 61, 163
trabajo, registro de, 22, 33, 36, 38, 147, 186
TRIS, 23, 91, 93, 121, 150
TXREG, registro, 23, 150, 250, 252
TXSTA, registro, 23, 150, 253

U

Ubuntu, 489
UDATA, directiva, 502
USART, 249

V

V_{DD}, voltaje de alimentación, 67
V_{SS}, voltaje de referencia, 67

W

WDT, Watchdog Timer, 56, 80, 87, 176, 183, 196
WREG, registro de trabajo, 22, 33, 36, 38, 147, 186

X

XORLW, instrucción, 16, 66, 69, 148
XORWF, instrucción, 16, 66, 69, 148

Z

Z, bandera de cero, 16, 22, 38, 40

Dirigido a todos aquellos que gustan de usar simultáneamente un texto, un código y circuito de ejemplo, una fuente, un cautín, un multímetro y algunos otros "juguetes", este libro describe el funcionamiento y uso de los microcontroladores PIC16 empleando un enfoque de aprender haciendo.

Se explican y ejemplifican: el conjunto de instrucciones, el núcleo del procesador (memorias, registro de uso especial y general, modos de direccionamiento, mapeo, saltos calculados, etc.), los puertos de uso general, los osciladores, los temporizadores, las comunicaciones en serie (*USART, SPI e I2C*), la conversión analógico-digital, la modulación por ancho de pulso, el manejo de exhibidores de cristal líquido (para texto y para gráficos), la programación del microcontrolador en el circuito de aplicación (*ICSP*) y los circuitos programadores.

Además de las numerosas indicaciones para el uso del ambiente de desarrollo MPLAB 8 en el sistema operativo Windows, también se ofrece un capítulo que describe la instalación y uso del MPLAB X en el sistema operativo Ubuntu (posiblemente la distribución más amigable de Linux).

Es importante destacar que todos los componentes empleados en el desarrollo de los ejemplos se adquirieron en México y que NO son de distribución exclusiva de un proveedor.

ISBN: 978-607-414-392-8

INSTITUTO POLITÉCNICO NACIONAL
DIRECCIÓN DE PUBLICACIONES