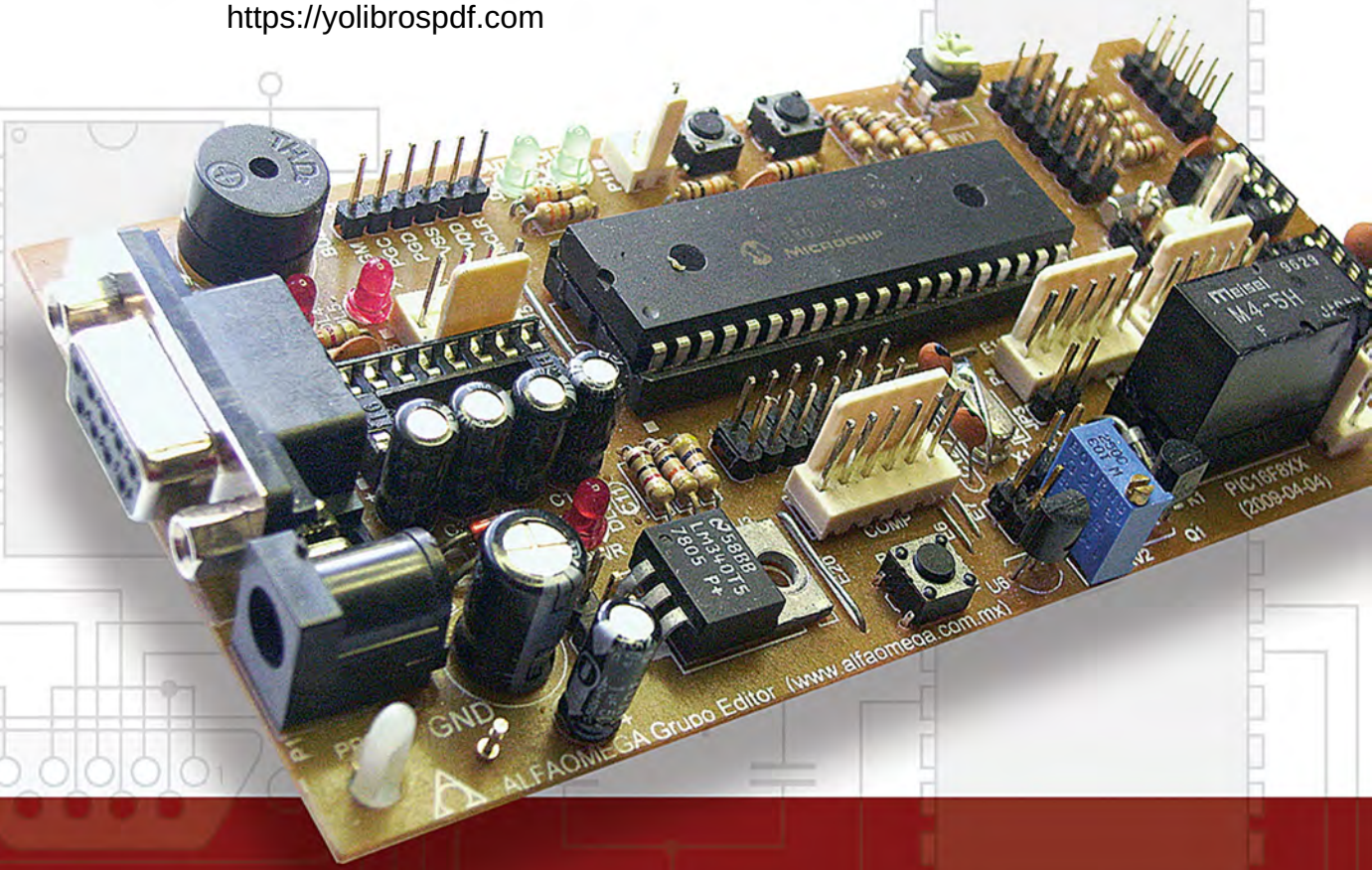
Microcontroladores PIC16F877A y PIC16F887

https://yolibrospdf.com



DANIEL SALVATIERRA

Alfaomega

https://yolibrospdf.com

Microcontroladores PIC16F877A y PIC16F887

Daniel Salvatierra Figueroa

Alfaomega

Buenos Aires • Bogotá • México DF • Santiago de Chile

Formación:
Alfaomega Grupo Editor

Al cuidado de la edición:
Luz Ángeles Lomelí Díaz
lalomeli@alfaomega.com.mx

Gerente Editorial:
Marcelo Grillo Giannetto
mgrillo@alfaomega.com.mx

Datos catalográficos

Salvatierra Figueroa, Daniel,
Microcontroladores PIC16F877A y PIC16F887.
Primera Edición

Alfaomega Grupo Editor, S.A. de C. V., México
ISBN: 978-607-707-463-2
Formato: 17 x 23 Páginas: 747

Microcontroladores PIC16F877A y PIC16F887.
Daniel Salvatierra Figueroa.
Derechos reservados © Alfaomega Grupo Editor S.A. de C.V., México

Primera edición: Alfaomega Grupo Editor, México, Mayo 2012

© 2012 Alfaomega Grupo Editor, S.A. de C.V.
Pitágoras No. 1139, Col. Del Valle, 03100, México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Página Web: **http://www.alfaomega.com.mx**
E-mail: **atenciónalcliente@alfaomega.com.mx**

ISBN: 978-607-707-463-2

Derechos reservados:
Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

NOTA IMPORTANTE:
La información contenida en esta obra tiene un fin exclusivamente didáctico y, por lo tanto, no está previsto su aprovechamiento a nivel profesional o industrial. Las indicaciones técnicas y programas incluidos han sido elaborados con gran cuidado por el autor y reproducidos bajo estrictas normas de control. ALFAOMEGA GRUPO EDITOR S.A de C.V. no será jurídicamente responsable por: errores u omisiones; daños y perjuicios que se pudieran atribuir al uso de la información comprendida en este libro, ni por la utilización indebida que pudiera dársele.

Impreso en México. Printed in Mexico

Empresas del grupo:

México: Alfaomega Grupo Editor, S.A. de C.V. - Pitágoras 1139, Col. Del Valle, México, D.F.
C.P. 03100, Tel.: (52-55) 55 75 50 22 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A. - Carrera 15 No. 64 A 29, Bogotá, Colombia,
Tel.: (57-1) 2100122 - Fax: (57-1) 6068648, E-mail: scliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A. – Dr. La Sierra 1437, Providencia, Santiago, Chile
Tel.: (56-2) 235-4248 – Fax: (56-2) 235-5786, E-mail: agechile@alfaomega.cl

Argentina: Alfaomega Grupo Editor Argentino, S.A. - Paraguay 1307 P.B. Of.11, C.P. 1057,
Buenos Aires, Argentina, Tel/Fax.: (54-11) 4811-0887/ 7183, E-mail: ventas@alfaomegaeditor.com.ar

https://yolibrospdf.com

Acerca del autor



Ing. Daniel Salvatierra Figueroa

Curso sus estudios como Ingeniero Electrónico en la Universidad Pontificia Bolivariana de la ciudad de Medellín-Colombia.

En su desempeño laboral trabajó en empresas en la industria de Computadores, Telefonía, Seguridad, Protecciones de red Eléctrica, y el área Textil; como Jefe de Mantenimiento, Jefe de proyectos, Diseño en área I+D, también como Colaborador en grupos de Investigación.

Presta servicios en forma independiente como asesoría en diseños de tarjetas funcionales en tecnologías THT y SMD, realiza cursos a estudiantes y docentes de diseño de circuitos impresos con el programa EAGLE y lenguaje C para Microcontroladores Freescale y Microchip con programas CodeWarrior y CCS.

Colabora con empresas Representantes e Importadores de componentes electrónicos en la construcción de herramientas de desarrollo para Microcontroladores para aplicaciones con Acelerómetros, Modem Celular, GPS, ZigBee; y en la asesoría a empresas para la implementación con estos dispositivos.

Su experiencia de más de 17 años en la realización de proyectos con Microcontroladores para pequeña y gran escala de producción, le han dado la capacidad y el conocimiento de los retos a los que se enfrentan los desarrolladores a la hora de realizar proyectos.

Contenido

Introducción 1

Parte 1

Conceptos básicos 3

1.0 Componentes de un MCU 3
1.1 Partes básicas 3
1.1.1 ALU (Unidad lógica aritmética) . 4
1.1.2 I/O (INPUT/OUTPUT)..... 4
1.1.3 Memoria (RAM, ROM..... 5
1.2 Partes adicionales 6
1.2.1 Clock 6
1.2.2 Reset..... 7
1.2.3 Interrupciones 7
1.2.4 Watchdog..... 7
1.2.5 Low - Power..... 8
1.2.6 Power V_{DD} - V_{SS} 8
Evaluación..... 8
2.0 Conceptos básicos de herramientas para MCU 9
2.1 ¿Qué es una herramienta? 9
2.2 Hardware 9
2.2.1 Programador (Programmer) 10
2.2.2 Depurador (Debugger) 13
2.2.3 Emulador (Emulator)..... 16
2.3 Software..... 18
Evaluación..... 21
3.0 Información técnica del MCU 22
3.1 Data sheet (Hoja de datos) 22
3.2 Package (Encapsulado)..... 24
3.3 Sistemas de identificación de producto (Product identificación system)..... 24
3.4 Soporte 25
3.5 Muestras..... 25

Parte 2

Arquitectura del MCU 27

4.0 Conceptos de Arquitectura..... 27
4.1 De microprocesadores a microcontroladores..... 27
4.2 Arquitecturas Harvard y Von Neumann 28
4.3 Fases del ciclo de instrucción..... 29
4.4 Tipos de set de instrucciones..... 30
4.5 Ortogonalidad..... 31
5.0 Los microcontroladores PIC..... 31
5.1 La familia PIC 10/12/16 32
5.2 Los grupos PIC16F87XA y PIC16F88X..... 33
6.0 Arquitectura del PIC16F877A..... 35
6.1 Características (Brochure)..... 35
6.2 Pin-Puertos y Periféricos 36
6.2.1 Descripción de Pines (Pinout)..... 36
6.2.2 Puertos y Periféricos 38
6.2.3 Diagrama de bloques..... 40
6.3 Memoria 41
6.3.1 Memoria de programa y stack 41
6.3.2 Memoria de datos..... 44
6.3.3 Registros de funciones especiales (SFR) 47
6.3.4 Memoria EEPROM..... 51

7.0 Arquitectura del PIC16F877 A 51
7.1 Características (Brochure) 51
7.2 Pin-Puertos y Periféricos 53
7.2.1 Descripción de Pines (Pinout) 53
7.2.2 Puertos y Periféricos 54
7.2.3 Diagrama de bloques 57
7.3 Memoria 57
7.3.1 Memoria de programa y stack 58
7.3.2 Memoria de datos 60
7.3.3 Registros de funciones especiales 61
7.3.4 Memoria EEPROM 67
8.0 Registros comunes 67
8.1 Registro W y Status 67
8.2 Contador del programa PCL y PCLATH 69
8.3 FSR (File Select Register), e INDF (Indirect File) 70
8.4 Registro Option_Reg (Option Reg) 72
8.5 Otros registros comunes 73
9.0 Funciones especiales de la CPU 73
9.1 Oscilador 73
9.2 Reset 76
9.2.1 MCLR 77
9.2.2 Power-On Reset (POR) 78
9.2.3 Power-Up Timer (PWRT) 78
9.2.4 Oscillator Start-Up Timer (OST) 78
9.2.5 Brown-Up Reset (BOR) 78
9.2.6 Secuencia Time-Out 78
9.2.7 Power Control Status Register (PCON) 79
9.3 Interrupción 81
9.3.1 Interrupciones enmascarable y no enmascarable 81
9.3.2 Instrucción Retfie 83
9.3.3 Proceso de una interrupción 83
9.3.4 Interrupciones no enmascarables TMRO/PORT/INT 84
9.3.4.1 Interrupción INT 84
9.3.4.2 Interrupción TMRO 84
9.3.4.3 Interrupción por cambio en PORTB 84
9.4 Watchdog Timer (WDT) 85
9.5 Baja potencia (Modo Sleep) 87
9.5.1 Despertando de Sleep (Wake-Up from Sleep) 88
9.5.2 Despertando usando interrupciones 90
9.6 Bits de configuración (Configuration Bits) 91
9.6.1 Código de Protección (Code Protection) 93
9.6.2 Identificador (ID Locations) 93
9.7 Depuración (In-Circuit Debugger) 94
9.8 ICSP (In Circuit Serial Programming) 94
9.8.1 Programando en bajo voltaje ICSP 95
10.0 Set de Instrucciones 96
10.1 Operaciones orientadas a Byte .. 96
10.2 Operaciones orientadas a Bit 96
10.3 Operaciones literales y control ... 97

Parte 3

Programación (I) 99

11.0 Conceptos básicos de programación con MCU 99
11.1 Proceso para elaborar un programa 103
11.1.1 Idea 103
11.1.2 Algoritmo 106
11.1.3 Diagrama de flujo 107
11.1.4 Código 110
11.2 Programa 111
11.2.1 El lenguaje 111
11.2.2 Campos y sintaxis 114
11.2.3 Tipos de archivos 119
Evaluación 123
11.3 Set de instrucciones 123
11.3.1 Instrucciones de carga ... 127
11.3.2 Instrucciones aritméticas 128
11.3.3 Instrucciones lógicas 133
11.3.4 Instrucciones de bit 138
11.3.5 Instrucciones de salto 138
11.3.6 Instrucciones de subrutinas e interrupciones 142
11.3.7 Instrucciones especiales 145
11.3.8 Tips y usos comunes 146
Evaluación 162
11.4 Directivas 163
11.5 Datos 169
11.5.1 Tipo de direccionamiento 169
11.5.2 Tipos de datos 174

11.5.3 Datos en tipos de memoria 178
11.5.4 Formatos de valores 183
11.6 Procesos especiales de programa....183
11.6.1 Subrutina 184
11.6.2 Interrupciones 195
Evaluación....202
12.0 Programador/Depurador-PICKIT 2...203
12.1 Descripción de partes....203
12.2 Descripción de pines ICSP 204
12.3 Modos de operación 205
12.3.1 Programador (Programmer) 205
12.3.2 Modo lógico I/O (Logic I/O Mode) 208
12.3.3 Modo analizador lógico (Logic Analyzer Mode) 209
12.3.4 Herramienta UART (UART Tool) 210
12.3.5 Programador Stand-Alone (Programmer-To-Go) 212
13.0 Sistema de evaluación-PIC16F8XX..213
13.1 Especificaciones....214
13.2 Configuración de pines y conectores....216
13.3 Planos....218
13.4 Lista de componentes....222
13.5 Procedimiento de ensamble....224
13.6 Pruebas de funcionamiento 225
13.7 Módulos....227

Parte 3

Programación (II) 229

14.0 Ambiente de desarrollo MPLAB....229
14.1 Entorno....230
14.1.1 El proceso de diseño 230
14.1.2 Componentes del MPLAB IDE 231
14.1.3 Partes ambiente MPLAB .233
14.2 Pasos parar crear un proyecto en MPLAB....233
14.2.1 Crear proyecto 235
14.2.2 Selección de dispositivo .235
14.2.3 Selección de lenguaje 236
14.2.4 Poner nombre y ruta del proyecto 237
14.2.5 Agregar archivos al proyecto 238
14.2.6 Resumen del proyecto 239
14.2.7 Crear código 240
14.2.8 Compilar el proyecto o Build 242
14.3 Simulación 244
14.4 Programación....255
14.4.1 Programación con PICKIT 2 Programmer 256
14.5 Depuración....263
14.5.1 ICSP 263
14.5.2 Programa ejemplo 264
14.5.3 Pasos para depuración ejemplo....265
14.6 Archivos*.LKR *.INC y Macros...283
14.6.1 Archivos de enlace Linker (*.LKR) 283
14.6.2 Archivos de definiciones (*.INC) 288
14.6.3 Macros 294

Parte 4

Memoria 297

15.0 Memoria....297
15.1 Tipos de memoria 298
15.1.1 Mapa de memoria de programa 299
15.1.2 Memoria de usuario o programa 301
15.1.3 Memoria de configuración 301
Evaluación 307
16.0 Memoria de programa....308
16.1 Ubicación....308
16.2 Acceso....310
16.2.1 Situación 1: Instrucciones con PCL como destino 312
16.2.2 Situación 2: Instrucción GOTO 313
16.2.3 Situación 3: Instrucción CALL 314
16.2.4 Situación 4: Instrucciones RETURN, RETFIE O RETLW 315
16.2.5 Protección CP y WRT 316
16.3 Leer y escribir en la memoria de programa....316
16.3.1 Registros usados 317
16.3.2 Procedimiento 319
16.3.3 Práctica 325
Evaluación 337

17.0 Memoria EEPROM 339
17.1 Diagrama circuital 339
17.2 Funcionamiento 339
17.2.1 Protección CPD 341
17.2.2 Leyendo datos en EEPROM 341
17.2.3 Escribiendo datos en EEPROM 342
17.3 Registros asociados 343
17.4 Procedimiento de uso 344
17.5 Práctica 345
18.0 Memoria RAM 360
18.1 Mapa de registros 360
18.2 SFR 363
18.2.1 Definición 363
18.2.2 Ubicación 363
18.2.3 Inicialización 364
18.3 GPR 365
18.3.1 Definición 365
18.3.2 Ubicación 366
18.3.3 Iniciación 366
18.4 Tipos de direccionamiento 367
18.4.1 Direccionamiento directo 367
18.4.2 Direccionamiento indirecto 368
Evaluación 369

Parte 5

Puertos I/O, Conversores A/D y Comparadores 371

19.0 Puertos I/O (Input/Output) 372
19.1 Puerto A 373
19.1.1 Diagrama circuital 373
19.1.2 Funcionamiento 374
19.1.3 Registros asociados 375
19.1.4 Procedimiento de uso 377
19.1.5 Práctica 378
19.2 Puerto B 381
19.2.1 Diagrama circuital 381
19.2.2 Funcionamiento 382
19.2.3 Registros asociados 383
19.2.4 Procedimiento de uso 384
19.2.5 Práctica 385
19.3 Puerto C 390
19.3.1 Diagrama circuital 390
19.3.2 Funcionamiento 391
19.3.3 Registros asociados 391
19.3.4 Procedimiento de uso 392
19.3.5 Práctica 393
19.4 Puerto D 396
19.4.1 Diagrama circuital 396
19.4.2 Funcionamiento 396
19.4.3 Registros asociados 397
19.4.4 Procedimiento de uso 398
19.4.5 Práctica 398
19.5 Puerto E 404
19.5.1 Diagrama circuital 404
19.5.2 Funcionamiento 404
19.5.3 Registros asociados 405
19.5.4 Procedimiento de uso 408
19.5.5 Práctica 409
19.6 Puerto Paralelo Esclavo 413
19.6.1 Diagrama circuital 413
19.6.2 Funcionamiento 414
19.6.2.1 Escritura 414
19.6.2.2 Lectura 415
19.6.3 Registros asociados 416
Evaluación 416
20.0 Módulo Conversor Análogo a Digital (A/D) 417
20.1 Diagrama circuital 417
20.2 Funcionamiento 418
20.2.1 Resolución 418
20.2.2 Justificación 419
20.2.3 Voltaje de referencia 420
20.2.4 Seleccionando clock 420
20.2.5 Tiempo de adquisición 421
20.2.6 Configuración de pines del puerto análogo 423
20.2.7 Operación 423
20.2.8 Conversión A/D 424
20.3 Registros asociados 426
20.4 Procedimiento de uso 429
20.5 Práctica 432
Evaluación 440
21.0 Comparadores 440
21.1 Módulo comparador (Comparator) 441
21.1.1 Diagrama circuital 441
21.1.2 Funcionamiento 444
21.1.3 Registros asociados 447
21.1.4 Procedimiento de uso 449
21.1.5 Práctica 450
Evaluación 458
21.2 Módulo generador de voltaje de referencia 458
21.2.1 Diagrama circuital 458
21.2.2 Funcionamiento 459
21.2.3 Registros asociados 462

21.2.4 Procedimiento de uso 463
21.2.5 Práctica 464
Evaluación.....472

Parte 6

Timers y PWM 473
22.0 Temporizadores (Timers).....473
22.1 Timer0.....474
22.1.1 Diagrama circuital 474
22.1.2 Funcionamiento 475
22.1.2.1 Modo Temporizador (Timer) 475
22.1.2.2 Modo Contador (Counter) 475
22.1.2.3 Prescaler 476
22.1.2.4 Interrupción (Interrupt) 477
22.1.2.5 Timer0 con clock externo 477
22.1.3 Registros asociados 478
22.1.4 Procedimiento de uso 479
22.1.5 Práctica 479
22.1.5.1 Montaje 1 479
22.1.5.1.1 Procedimiento de uso Timer0 modo temporizador (timer por polling) 480
22.1.5.2 Procedimiento por uso Timer0 modo temporizador (timer por interrupción) 483
22.1.5.2 Montaje 2 487
22.1.5.2.1 Procedimiento de uso Timer0 modo contador (counter por polling) 488
22.1.5.2.2 Procedimiento de uso Timer0 modo contador (counter) por interrupción) 492
22.2 Timer1.....496
22.2.1 Diagrama circuital 496
22.2.2 Funcionamiento 496
22.2.2.1 Modo temporizador (timer) 497
22.2.2.2 Modo contador (counter) 497
22.2.2.2.1 Modo contador sincrónico 498
22.2.2.2.2 Modo contador asincrónico 498
22.2.2.2.3 Modo contador asincrónico, lectura y escritura del Timer1.....499
22.2.2.3 Oscilador 499
22.2.2.4 Reseteo del Timer 1, par de registros (TMR1H:TMR1L) 500
22.2.2.5 Prescaler 500
22.2.3 Registros asociados 500
22.2.4 Procedimiento de uso 501
22.2.5 Práctica 501
22.2.5.1 Montaje 1 501
22.2.5.1.1 Procedimiento de uso Timer1 modo temporizador (timer por polling).....502
22.2.5.1.2 Práctica Timer1 modo temporizador (timer por polling).....503
22.2.5.1.3 Procedimiento de uso Timer1 modo temporizador (timer por interrupción)..505
22.2.5.1.4 Práctica Timer1 modo temporizador (timer por interrupción)..506
22.3 Timer2 509
22.3.1 Diagrama circuital 509
22.3.2 Funcionamiento 509
22.3.3 Registros asociados 510
22.3.4 Procedimiento de uso.....511
22.3.5 Práctica 512
22.3.5.1 Montaje 512
22.3.5.2 Timer2 (Timer por polling) 513
22.3.5.3 Timer2 (Timer por interrupción) ...515
Evaluación 517

23.0 Captura/Comparación/PWM (CCP).....518
23.1 Modo captura (Capture).....520
23.1.1 Diagrama circuital 520
23.1.2 Funcionamiento 520
23.1.3 Registros asociados.....522
23.1.4 Procedimiento de uso.....524
23.1.5 Práctica.....527
23.1.5.1 Montaje 527
23.1.5.1.1 Práctica capture por polling 529
23.1.5.1.2 Práctica capture por interrupción) 533
23.2 Modo de comparación (Compare).....537
23.2.1 Diagrama circuital 537
23.2.2 Funcionamiento 537
23.2.3 Registros asociados.....539
23.2.4 Procedimiento de uso.....540
23.2.5 Práctica.....544
23.2.5.1 Montaje 544
23.2.5.1.1 Práctica compare por polling 544
23.2.5.1.2 Práctica compare por interrupción .547
23.3 Modo PWM Pulse Width Modulator) 550
23.3.1 Diagrama circuital 550
23.3.2 Funcionamiento 551
23.3.3 Registros asociados.....554
23.3.4 Procedimiento de uso.....556
23.3.5 Práctica.....558
23.3.5.1 Montaje 559
23.3.5.1.1 Práctica PWM por polling 559
23.3.5.1.2 Práctica PWM por interrupción .562
Evaluación 564

Parte 7

Módulos de Comunicaciones: SPI, I_2C y SCI o USART 567

24.0 Módulo Puerto Serial Sincrónico Maestro (MSSP).....567
24.1 Módulo Puerto Serial Sincrónico Maestro (MSSP).....568
24.2 Modo Interfaz Periférica Serial (SPI) 568
24.2.1 Protocolo SPI 568
24.2.1.1 Clock (SCK) 570
24.2.1.2 Datos SDI y SDO 571
24.2.1.3 CS o SS 572
24.2.1.4 Modos 572
24.2.2 Módulos MSSP en modo SPI 572
24.2.3 Diagrama circuital 573
24.2.4 Funcionamiento 574
24.2.4.1 Inicializando 574
24.2.4.2 Habilitando SPI/O 574
24.2.4.3 Conexión 575
24.2.4.4 En operación 575
24.2.4.5 Modos del bus SPI 576
24.2.4.6 Modo maestro ...577
24.2.4.7 Modo esclavo 578
24.2.4.8 Operación en Modo Sleep 581
24.2.4.9 Efectos de un reset 581
24.2.5 Registros asociados 581
24.2.6 Práctica 584
24.2.6.1 Código SPI Maestro 585
24.2.6.2 Código SPI Esclavo 586
Evaluación.....589

25.0 Modo Comunicación Inter-Integrados (I2C).....589
25.1 Estándar I2C 589
25.1.1 Hardware.....589
25.1.2 Terminología 591
25.1.3 Protocolo 591
25.1.4 Transferencia de datos...592
25.2 Módulos MSSP en modo I2C 593
25.2.1 Diagrama circuital 594
25.2.2 Funcionamiento 595
25.2.2.1 BRG (Baud Rate Generator) 596
25.2.2.2 Modo esclavo 596
25.2.2.3 Modo maestro 597
25.2.3 Registros asociados 598
25.2.4 Práctica 602
25.3 Memoria EEPROM I2C 24LC32 . 613
25.3.1 Pinout.....613

25.3.2 Recomendaciones de hardware 614
25.3.3 Palabra de control....615
25.3.4 Direccionamiento....615
25.3.5 Operación de escritura....616
25.3.5.1 Escritura de un byte....616
25.3.5.2 Escritura de una página....617
25.3.6 Operación de lectura....617
25.3.6.1 Lectura de dirección actual....618
25.3.6.2 Lectura aleatoria....618
25.3.6.3 Lectura secuencial....619
Evaluación....620
26.0 Módulo Interfaz de Comunicación Serial (SCI USART)....621
26.1 Estándar RS232....621
26.1.1 Asignación de nombres y pines....622
26.1.2 Niveles lógicos....625
26.1.3 Formato de trama....625
26.1.4 Circuito integrado....627
26.1.5 Conexión circuito integrado....627
26.2 Módulo SCI....628
26.2.1 El generador de tasa de Baudios o Baud Rate Generador (BRG)....629
26.2.1.1 Ejemplo 1....630
26.2.2 Circuito de muestreo o sampling circuit....631
26.3 Diagrama circuital....632
26.4 Funcionamiento....633
26.4.1 Modo asíncrono o Asincrónico....634
26.4.1.1 Transmisor Asíncrono....634
26.4.1.2 Receptor Asíncrono....637
26.5 Registros asociados....640
26.6 Práctica....643
26.6.1 Montaje....644
26.6.1.1 Modo de Transmisión y Recepción....645
26.6.1.2 Modo de Transmisión....647
26.6.1.3 Modo de Recepción....649
Evaluación....652

Apéndices

Apéndice A
Características Técnicas....653
Apéndice B
Set de Instrucciones....659
Apéndice C
Operadores....663
Apéndice D
Directivas del MPASM....667
Apéndice E
Registros Especiales....695
Apéndice F
PDF Sistema de Evaluación PIC16F8XX....705
Apéndice G
Código ASCII....717
Apéndice H
Direcciones en Internet....719
Apéndice I
Contenido adicional....721

Bibliografía 723

Índice Analítico 725

Antes de comenzar a leer

RECOMENDACIONES

Este libro está acompañado de la tarjeta de evaluación **PIC16F8XX** con toda la información requerida para su implementación como planos esquemáticos y de board, lista de partes, su forma de ensamble y prueba. Con esta tarjeta se pueden implementar todas las prácticas de cada uno de los módulos. Esto facilita el aprendizaje de los conceptos y sus prácticas.

En nuestra página Web ubicada en http://virtual.alfaomega.com.mx se encuentra información adicional que complementa al contenido de este libro, allí encontrará carpetas que contienen: Ejemplos ASM, *data sheet*, herramientas y software.

Si usted es profesor, le recomendamos ampliamente este libro, pues contiene todos los conceptos básicos y prácticas de cada uno de los módulos del MCU, lo que hace fácil la implementación de un curso con estos MCU.

Si usted es una persona experta, le recomendamos dirigirse a las Partes 4, 5, 6 y 7, que son los **módulos aplicativos**.

Si usted es una persona que ha trabajado alguna vez con otro microcontrolador, le recomendamos que se salte los conceptos básicos e inicie con la **Parte 2, "Arquitectura"**.

La Parte 3 se recomienda ampliamente por los modos de trabajo del MPLAB y las herramientas de trabajo del libro; además de los tips para el **set de instrucciones.**

INTRODUCCIÓN

Este libro está dirigido a lectores novatos, aficionados (hobbistas), estudiantes, desarrolladores expertos y a profesores que deseen implementar en sus cursos el aprendizaje de estos microcontroladores en lenguaje ensamblador (ASM).

La persona que lea este libro deberá tener conceptos claros sobre:

- Sistemas de numeración decimal, binaria, hexadecimal, BCD y Código ASCII.
- Manejo de lógica booleana y compuertas lógicas.
- Lectura de planos electrónicos.

El libro está dividido en 8 Partes, y cada Parte en Capítulos. Las Partes 1 a 3 son **conceptuales,** y las Partes 4 a 7 son **aplicativas**. En la Parte 1 se hace una breve descripción de los **conceptos básicos asociados con los MCU** y sus partes, los conceptos de herramientas de hardware y software, así como el manejo de la información técnica de los MCU.

La Parte 2 trata todo lo relacionado con la **arquitectura**, los tipos de arquitectura, una breve descripción de las familias del MCU Microchip y la familia PIC10/12/16; dentro de esta familia están los Grupos **PIC16F87XA** y **PIC16F88X**, y una detallada explicación de la arquitectura de los micros **PIC16F877A y PIC16F887**, con características y especificaciones, descripción de pines, mapa de memoria, diagrama en bloques, registros y registros comunes a ambos MCU.

En la Parte 3 se estudian los **conceptos básicos de programación**, con explicación de términos como: algoritmo, lenguaje, sintaxis, tipos de archivo y conceptos, cómo es un programa, cómo se elabora y recomendaciones para su manejo; también se detalla el *set* de instrucciones según el tipo de operación, los datos y sus aspectos, como tipos de direccionamiento, constantes y variables y el manejo de las subrutinas y las interrupciones. Luego se ven las herramientas de trabajo de hardware como el programador-depurador PICkit 2, y la herramienta de evaluación **PIC16F8XX**; por último, el **ambiente de desarrollo MPLAB** como herramienta de software en sus formas de simulación, programación y depuración.

A partir de las partes **aplicativas,** se describen los Módulos Específicos MCU; cada uno de ellos contiene: diagrama circuital, funcionamiento, registros asociados, procedimiento de uso y práctica asociada con el módulo.

En la Parte 4 se describen los tipos de **memoria de datos, de programa y EEPROM**, los bits de configuración y el **ID de usuario**, su uso y cuidados. También se verán el **mapa de memoria**, las ubicaciones y la forma de acceso. En la **memoria RAM** se ven los tipos de registro SFR y GPR, y los tipos de direccionamiento.

En la Parte 5 se describen los módulos **puertos A-E y puerto paralelo, conversores análogo a digital, comparadores y generador de voltaje de referencia**. Cada uno de estos módulos se detalla con sus registros de uso y diagramas.

En la Parte 6 se describen los módulos **Timers0, 1 y 2, Captura, Comparación** y **PWM**. En la Parte 7 se describen los módulos de **Comunicaciones**: **SPI**, **I2C** y **SCI** o **USART**.

Por otra parte, se encuentran los **Apéndices**, en donde encontrará información adicional relevante.

PARTE 1

CONCEPTOS BÁSICOS

En esta parte estudiaremos los conceptos básicos concernientes a un microcontrolador, los componentes básicos y adicionales de un MCU. También veremos los conceptos relacionados con las herramientas de hardware y software. Y, por último, la información pertinente a un MCU, como datasheet, addendum, empaque, el sistema de identificación de producto, soporte y muestras.

1.0 COMPONENTES DE UN MCU

Un **microcontrolador** es una máquina electrónica basada en un ordenador; con base en entradas realiza operaciones lógicas y aritméticas que puede almacenar (datos), para generar salidas; estas operaciones se implementan mediante instrucciones en orden secuencialmente lógico (programa).

1.1 PARTES BÁSICAS

Un **MCU** consta de tres partes básicas (Figura 1.1):

1. **ALU:** Unidad Lógica Aritmética.
2. **I/O:** Dispositivos de **entrada** (*input*) y **salida** (*output*).
3. **Memoria:** Almacenamiento de datos y programa.

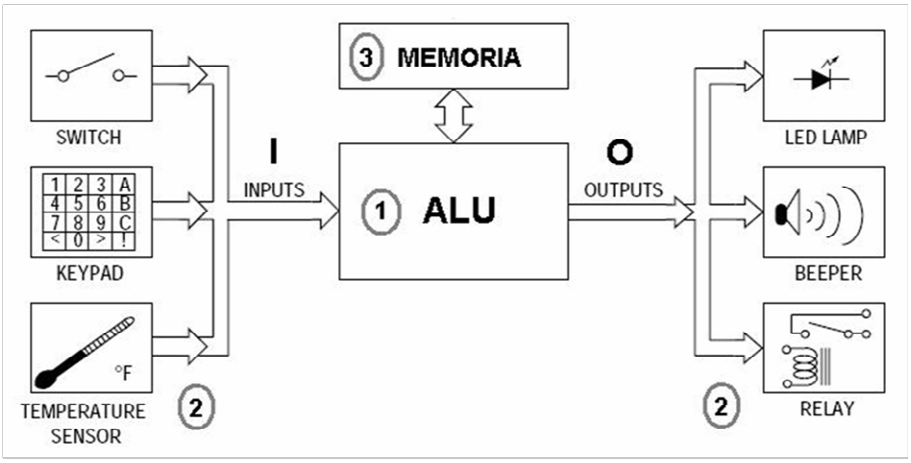


Figura 1.1 Partes del MCU.

1.1.1 ALU (UNIDAD LÓGICA ARITMÉTICA)

La **ALU** es la unidad encargada de realizar operaciones con los datos, realiza operaciones:

- **Aritméticas,** tales como sumar, restar.
- **Lógicas,** como **AND, OR, EXOR**, negación, complemento, rotación.
- **Toma de decisiones,** como bifurcación por prueba de verdadero o falso, por valor 0 o 1, por prueba de valor 0 en contadores.
- **Manejo de datos,** direccionamientos directos o indirectos, lectura y escritura.

1.1.2 I/O (INPUT / OUTPUT)

Éstas son las unidades básicas que tiene un **MCU** para comunicarse con el mundo exterior; por ellas entran y salen los datos. Como unidades básicas, están los puertos digitales (Tabla 1.1) que son agrupaciones de pines bajo un mismo nombre (**PORTA**, **PORTB**); la denominación de los pines contiene el nombre del puerto correspondiente (**RA1**, **RA2** o **RB5**, **RB7**). Estos pines comparten el funcionamiento con otros módulos especializados, como: **A/D** (Análogo a Digital), *comparator* (comparadores), *timers* (temporizadores), **PWM**, **SCI** (*Serial Communication Interface*), **SPI** (*Serial Peripheral Interface*), entre otros. La forma como están distribuidos los pines es llamada *pinout.*

Tabla 1.1 Pines de PUERTO A compartidos con otros módulos periféricos.

			PERIFERICOS		
			TIMER	SPI	
PIN	NOMBRE PIN	PUERTO	A/D	COMP	COMP
2	RA0/AN0	RA0	AN0		
3	RA1/AN1	RA1	AN1		
4	RA2/AN2/VREF-/CVREF	RA2	AN2	VREF-	CVREF
5	RA3/AN3/VREF+	RA3	AN3	VREF+	
6	RA4/T0CKI/C1OUT	RA4	T0CKI	C1OUT	
7	RA5/AN4/SS/C2OUT	RA5	AN4	SS	C2OUT

MODULOS	DESCRIPCION
SPI	Serial Protocol Interface
A/D	Analogo a Digital
TIMER	Temporizadores
COMP	Comparadores

1.1.3 MEMORIA (RAM, ROM)

La **memoria** es aquella unidad que maneja los datos y puede ser: de datos (**RAM**: *Random Access Memory*) y de programa (**ROM**: *Read Only Memory*). Los datos en **RAM** permanece mientras haya suministro de energía, y los datos en **ROM** permanece aunque no haya suministro de energía. Estos bloques de memoria son accedidos por diferentes periféricos mediante buses de datos y direcciones, y sus correspondientes *latch* y *buffer* para el adecuado transporte de los datos. La distribución de estas áreas de memoria es conocida como **mapa de memoria**. En la **RAM** se encuentran los registros especiales propios del MCU para su funcionamiento y registros de propósito general o de usuario.

También encontramos una memoria especial de datos, **EEPROM** (Figura 1.2), la cual guarda datos de una manera especial cuando falta la alimentación de energía. Igualmente, en la **ROM** encontramos áreas o bloques propios del **MCU** para su funcionamiento, y áreas de programa de usuario.

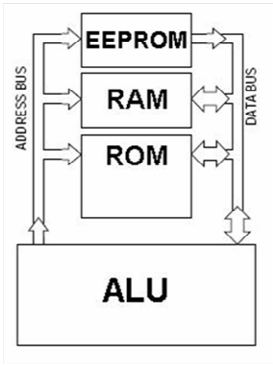


Figura 1.2 Memorias RAM, ROM y EEPROM

1.2 PARTES ADICIONALES

Algunas partes vitales para el funcionamiento adecuado del **MCU** no entran en las categorías ya descritas, como son las que veremos a continuación.

1.2.1 CLOCK

Es el encargado de sincronizar todas las acciones que realiza el MCU. Internamente, los MCU generan **un ciclo de máquina** que contiene varias fases de sincronismo que son las encargadas de que todas las partes actúen en la fase y en los tiempos adecuados. En los MCU se debe seleccionar un oscilador que puede ser interno o externo. Según el tipo de aplicación y exactitud, se escoge el tipo de oscilador requerido (Figura 1.3).

Por lo general, los osciladores internos son de baja precisión (1 a 5%), construidos con circuitos RC que tienen parámetros de calibración de fábrica para ajustar su precisión; éstos dependen del voltaje de alimentación y de la temperatura. Cuando se requiere mayor precisión (<< 1%) se usan, en su orden de importancia, módulos osciladores, luego los cristales y, por último, de no tan buena precisión, los resonadores. Los **circuitos RC** son de muy baja precisión y poco recomendables debido a su gran variación con la temperatura.

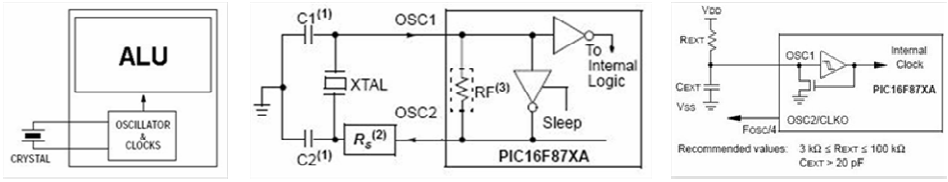


Figura 1.3 Oscilador.

1.2.2 RESET

El pin de *reset* (Figura 1.4) es aquel por donde el usuario puede restablecer abruptamente el funcionamiento del MCU al punto de inicio del programa. Es muy recomendable poner en los circuitos una red RC y un pulsador.

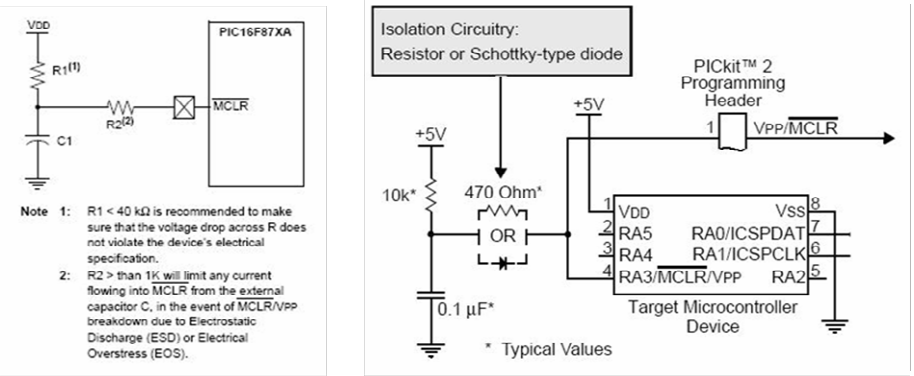


Figura 1.4 Reset.

Este pin también puede ser manejado por otros módulos internos con la misma intención. Este pin puede ser utilizado por dispositivos programadores o depuradores; por eso hay que tener precaución al realizar la conexión con estos.

1.2.3 INTERRUPCIONES

El **MCU**, en su modo de operación, ejecuta un programa principal en forma secuencial, atendiendo las operaciones de las **I/O**. Las interrupciones generan saltos en el funcionamiento del programa principal para realizar rutinas de interrupción cortas, para luego volver a la ejecución del programa principal donde fue interrumpido. Primero se deben habilitar estas interrupciones y, una vez ocasionadas, se debe borrar su bandera de interrupción en la rutina de servicio de interrupción. Las interrupciones son muy útiles porque permiten que el MCU atienda el programa principal, y solamente cuando aparezca un evento de interrupción realiza otras operaciones.

1.2.4 WATCHDOG

El *watchdog* (**perro guardián)** es un oscilador interno que, una vez habilitado, se debe reiniciar cíclicamente para que, a su vez, no genere un *reset*. El *watchdog* es un dispositivo de seguridad que evita que el programa se pierda o se quede en

ciclos infinitos o muy largos; si esto ocurre, el *watchdog* genera un *reset* y el programa se reinicia indicando que hubo un *reset* por **WDT** (*Watchdog Timer*).

1.2.5 LOW – POWER

Cada vez más aplicaciones están siendo orientadas al bajo consumo de energía, por tratar de hacer los equipos portátiles y, por ende, soportados por baterías; esto implica que se tenga muy en cuenta el consumo de energía para que sea el menor posible, haciéndolo eficiente. De esta manera, debemos optimizar los programas con la utilización de los modos de bajo consumo *sleep* y *wait,* con los que el micro queda en modo dormido y de espera.

1.2.6 POWER V_{DD} - V_{SS}

La alimentación de energía es la parte vital de todo sistema electrónico. Debido a la construcción interna de los MCU, el voltaje de alimentación debe ser muy estable y con bajo nivel de ruido. Los pines de alimentación son V_{DD} terminal positivo (usualmente 5 o 3 voltios) y V_{SS} terminal negativo (= 0 voltios) deben estar lo más cerca de la fuente de alimentación. También existen dispositivos internos *brown-out* que avisan cuando hubo una caída de voltaje en la alimentación y, para eso, generar un *reset*, pues por esta causa el MCU podría presentar un comportamiento errático, por lo que es preferible reiniciar el programa.

EVALUACIÓN

P.: 1. Nombre las tres partes más importante de un MCU.
R.:
ALU, I/O, memoria.

P.: 2. ¿Qué tipos de memorias básicas existen?
R.:
Memoria de datos (RAM) y de programa (ROM).

P.: 3. Mencione otras partes adicionales del MCU.
R.:
a). Clock
b). Reset
c). Interrupciones
d). Watchdog
e). Low-power
f). Power V_{DD}-V_{SS}

2.0 CONCEPTOS BÁSICOS DE HERRAMIENTAS PARA MCU

Cuando se decide comenzar a trabajar con **MCU**, se debe tener a la mano una serie de elementos básicos para poder realizar una aplicación. Estos elementos son del tipo **hardware** (circuitos electrónicos) y **software** (programas o aplicaciones).

2.1 ¿QUÉ ES UNA HERRAMIENTA?

Es un conjunto de elementos necesarios para manipular apropiadamente los MCU. Sirve para facilitar el manejo de los recursos internos del MCU, programar el MCU y verificar el funcionamiento adecuado del MCU en el circuito de aplicación con el resultado deseado.

Podemos clasificar las herramientas en:

- **Hardware** (circuitos electrónicos):
  - **Programador** (*Programmer*).
  - **Depurador** (*Debugger*).
  - **Emulador** (*Emulator*).

- **Software** (programas o aplicaciones):
  - **Ambiente de desarrollo** (IDE: Integrated Development Environment).

2.2 HARDWARE

Éstas son las herramientas circuitales que contienen una **base** (*Socket*) para la recepción del MCU, o un conector para comunicación con la **tarjeta** o **board de aplicación** (*target*). Además, posee una conexión con la PC para la comunicación con el software de **Ambiente de Desarrollo Integrado** o **IDE**. Los hay de diferentes tipos; a medida que crece la versatilidad de las funciones de manipulación sobre el MCU, crece la complejidad de la herramienta y también está asociada con su costo.

- **Programador** (*Programmer*).
- **Depurador** (*Debugger*).
- **Emulador** (*Emulator*).

A continuación se hace una descripción de cada uno de estos tipos de herramienta, cada una con la configuración básica, herramientas de Microchip y herramientas en Internet.

2.2.1 PROGRAMADOR (PROGRAMMER)

Ésta es la más básica de las herramientas y la de más bajo costo (Figura 2.1), porque únicamente sirve para programar el **MCU**; puede escribir y leer la memoria de programa y, a veces, la **EEPROM.** Puede cambiar los parámetros básicos de configuración, como alimentación, tipo de oscilador, habilitación de **WDT**, entre otros. Se consigue fácilmente en Internet, tipo "hágalo usted mismo", con toda la documentación necesaria para realizarla, tiene la limitación de que hay que intercambiar el MCU entre el programador y el circuito de aplicación.

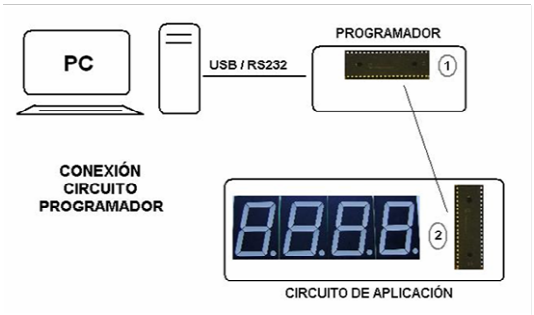


Figura 2.1 Conexión circuito-programador.

De esta manera, el usuario solamente tiene la posibilidad de programar en una base (socket) sobre la herramienta (Paso 1), y luego el MCU se traslada al circuito de aplicación (Paso 2); allí podrá ver el funcionamiento de su programa cuando está corriendo. Ésta es una forma poco idónea para los desarrolladores de aplicaciones, pues se convierte en un modo de ensayo y error. También tiene el gran inconveniente de que el programa funcione muy bien en condiciones de laboratorio, pero en las condiciones reales funcione erráticamente, sin poder ubicar el problema real; otro inconveniente no menos perjudicial es la excesiva manipulación de los pines del MCU. En esta categoría se encuentran herramientas de Microchip con toda la documentación disponible; algunas son:

- **MPLAB PM3**



MPLAB PM3
Núm. de parte: DV007004
Valor: US$895

Figura 2.2 MPLAB PM3

Trabaja con PC o *stand-alone*, *interface* **RS-232** o **USB**, programa toda la línea de microcontroladores **PIC** y también **dsPICs (DSC)**. Tiene un *socket* de 40 pines, y viene con:

- Kit MPLAB PM3
- Cable serial para conexión RS-232 PC
- Cable USB para conexión USB PC
- Cable ICSP
- Adaptador y cables
- CD con MPLAB IDE
- Manual de usuario y documentación técnica en CD

- **PICSTART Plus**



PICSTART Plus
Núm. de parte: DV003001
Valor: US$200

Figura 2.3 PICSTART Plus.

Es un programador de bajo costo para empaques hasta DIP de 40 pines, se conecta vía **RS232**, opera con el **MPLAB** que viene incluido; sus partes son:

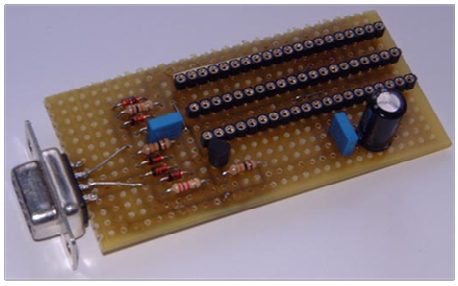
- Kit PICSTART plus
- Cable serial para conexión RS-232 PC
- Adaptador y cables
- CD con MPLAB IDE
- Manual de usuario y documentación técnica en CD
- Copia libre de **PICC Lite Compiler** y una muestra del microcontrolador **PIC16F84**

En Internet, también en esta categoría, se encuentran muchas herramientas descargables gratis, con toda la documentación disponible; algunas son:

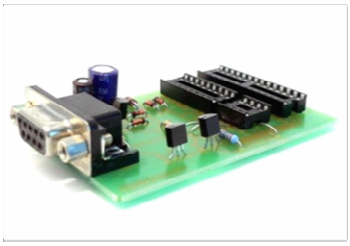
- PICPROG
- Programador JDM (TE-20)
- Programador PIC Pablin II
- Programador PIPO2
- ProPic 2 PIC Programmer.

A) B)





C) D)



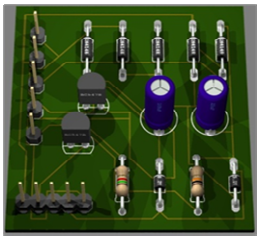


Figura 2.4 Programadores.

Remitimos al lector a la página web http://virtual.alfaomega.com.mx en el apartado **Herramientas**, para la documentación sobre éstas.

2.2.2 DEPURADOR (DEBUGGER)

Esta herramienta (Figura 2.5), ideal para los desarrolladores de aplicaciones, de costos medio y alto, además de programar los MCU, permite también la posibilidad de ver sus registros, y alterarlos, correr el programa paso a paso, hacer **puntos de parada** (*breakpoint*) en el programa, y una variedad de posibilidades para realizar la depuración del programa, con la lectura de los registros del MCU. Esto se realiza directamente mediante la conexión de un cable desde el depurador hasta el circuito de aplicación, sin necesidad de retirar el MCU. Los hay en Internet, pero es más conveniente adquirirlos con un distribuidor.

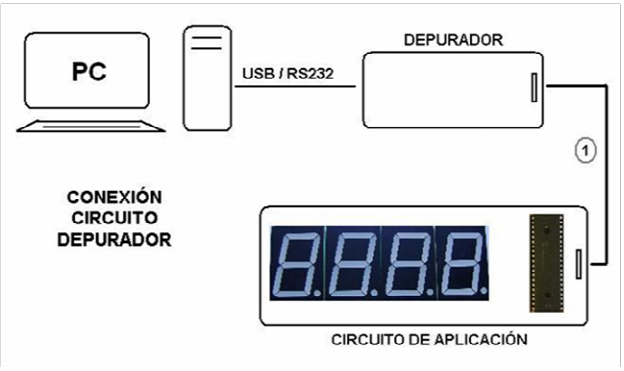


Figura 2.5 Conexión circuito-depurador.

Además, poseen todas las características de los programadores ya vistas. En esta categoría se encuentran herramientas en **Microchip** con toda la documentación disponible, tales como:

- MPLAB ICD 3
- MPLAB ICD 2
- PICkit™ 3
- PICkit™ 2

- **MPLAB ICD 3**



MPLAB ICD 3
Núm. de parte: DV164035
Valor: US$220

Figura 2.6 MPLAB ICD 3.

Este *debugger/programmer*, para los MCU Microchip y dsPIC (DSC), se conecta y alimenta de la conexión USB2.0, y hacia los micros con un conector (RJ-11); y soporta más micros que el MPLAB ICD 2.



MPLAB ICD 2
Núm. de parte: DV164005
Valor: US$160

Figura 2.7 MPLAB ICD 2.

- **MPLAB ICD 2**

Este *debugger/programmer*, de bajo costo, sirve para los **MCU Microchip** y **dsPIC (DSC)**, se conecta por interfaz RS232 o USB, y hacia los micros con un conector (RJ-11).



PICKIT™ 3
Núm. de parte DV164131
Valor: US$70

Figura 2.8 PICKIT™ 3.

- **PICKIT™ 3**

Este *debugger/programmer*, de bajo costo, sirve para los **MCU Microchip** y **dsPIC (DSC)**, se conecta a la PC por interfaz USB, y hacia los micros utiliza, ICSP (*In-Circuit Serial Programming*) con un conector hembra de 6 pines.

El **PICKIT 3 Debug Express** contiene:

- PICKIT 3 **Debugger/Programmer.**
- Board demo de microcontrolador PIC18F45K20 44-pin.
- Cable USB a PC.
- CDROM que incluye información técnica MPLAB y curso con lecciones.

➢ **PICKIT™ 2**



PICKIT™ 2
Núm. de parte DV164121
Valor: US$70

Figura 2.9 PICKIT™ 2.

Este debugger/programmer, de bajo costo, sirve para los MCU Microchip y dsPIC (DSC), se conecta a la PC por interfaz USB, y hacia los micros, utiliza ICSP (In-Circuit Serial Programming) con un conector hembra de 6 pines.

Esta herramienta es muy poderosa, pues contiene:

- Programmer-to-go que permite grabar archivos para luego programar MCU sin la mediación de una PC.
- Programmer UART Tool que es un sniffer de RS232.
- *Programmer Logic Tool Analyzer* que es un analizador lógico de 3 pines.

El PICKIT 2 Debug Express contiene:

- PICKIT 2 Debugger/Programmer.
- Board demo de microcontrolador PIC16F887 44-pin.
- Cable USB a PC.
- CDROM que incluye información técnica MPLAB y curso con lecciones.

En esta categoría, en Internet encontramos gratis muy pocos; observemos la Figura 2.10:

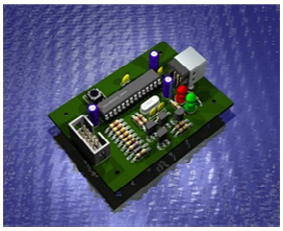


Figura 2.10 PICKIT 2 clone.

Éste es una versión reducida del PICKIT 2 de Microchip. Remitimos al lector a la página web http://virtual.alfaomega.com.mx en el apartado **Herramientas**, para la documentación sobre éstas.

2.2.3 EMULADOR (EMULATOR)

Es la herramienta por excelencia para expertos, su costo es elevado y también su complejidad, pues a partir de circuitos se generan los pines del MCU, con avanzadas características de generación y trazado, con funciones especiales para módulos; son programadores y depuradores mucho más poderosos.

Aquí no hay MCU físico; en vez de eso se emula; se debe manejar con precaución porque los pines son directamente manejados por la tarjeta de la herramienta y tiene una base (*socket*) que se conecta a la aplicación, emulando el funcionamiento del MCU escogido. En este caso, no hay MCU en ninguna parte, porque el emulador, a través del conector, genera todas las funciones del MCU.

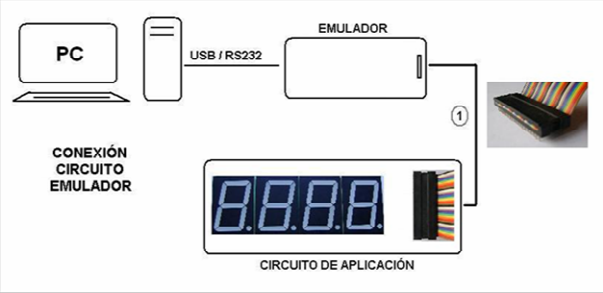


Figura 2.11 Conexión de circuito emulador.

En esta categoría, ofrecidos por **Microchip**, encontramos:

- MPLAB REAL ICE
- MPLAB ICE 2000

- MPLAB REAL ICE



MPLAB REAL ICE
Núm. de parte: DV244005
Valor: US$ 500

Figura 2.12 MPLAB REAL ICE.

Éste es un sistema emulador de bajo costo, programa *In-Circuit* para Microchip Flash DSC (*Digital Signal Controller*); además permite depuración y programación de microcontroladores PIC y dsPIC. Tiene una poderosa interfaz gráfica con el MPLAB. Es conectada a la PC con cable USB y a los micros con RJ11 y conector (CAT5) de bajo ruido y alta velocidad (LVDS).

- MPLAB ICE 2000



MPLAB ICE 2000
Núm. de parte: ICE2000
Valor: US$1 000

Figura 2.13 MPLAB ICE 2000.

Éste es un sistema emulador de alta velocidad (hasta 25 MHz) para muchos MCU Microchip, de fácil intercambio para emular otro MCU, tiene hasta **65 535 breakpoints**, posee sofisticados análisis de trazo y funciones especiales para puerto I/O, timers y otros periféricos. En Internet es muy difícil encontrar esta clase de herramientas gratis, porque requiere un grupo de trabajo grande y un hardware especializado.

En resumen (Figura 2.14), Microchip brinda una gran variedad de herramientas de hardware en un rango amplio de costos.

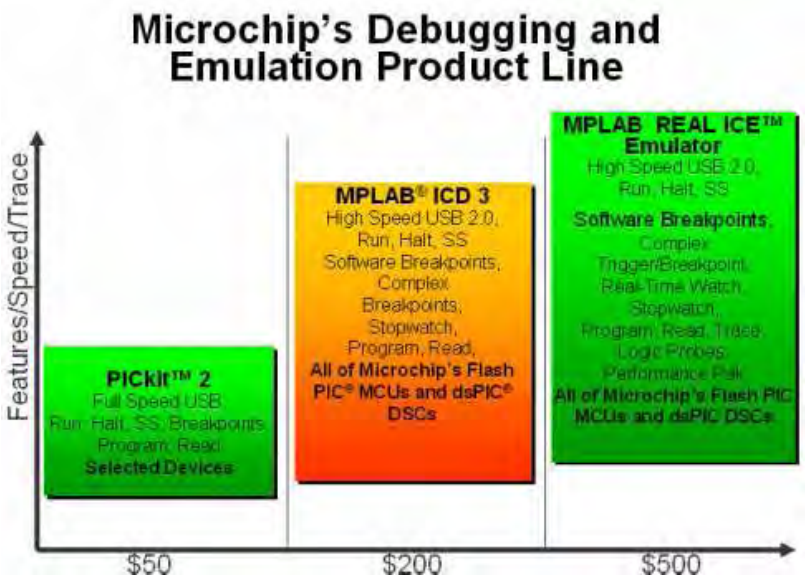


Figura 2.14 Herramientas Microchip.

2.3 SOFTWARE

Estas herramientas son aplicaciones o programas que se instalan en una PC con el propósito de manejar el MCU; se conectan con la herramienta de hardware por un puerto USB, Serial RS232 o **paralelo**. También son llamadas **Ambiente de Desarrollo** o **IDE** (*Integrated Development Environment*); son desarrolladas para editar, ensamblar, compilar, simular, programar, depurar o emular aplicaciones con MCU. Un **ambiente de desarrollo** debe poseer facilidades, tales como: un editor de texto de alto nivel, un compilador que convierta el código de los programas en archivos de texto a código hexadecimal y un protocolo de comunicación para escribir y leer en los MCU. Las herramientas deben tener la capacidad de manejar apropiadamente cada uno de los pines, registros y posiciones de memoria de cada MCU que soporta. Se encuentran en Internet y son descargables (algunas versiones son gratis) para herramientas de hardware como programadores, depuradores y emuladores asociadas con el hardware. Estas aplicaciones pueden trabajar con lenguajes como ASM, C y QuickBasic, entre otros.

- MPLAB IDE de Microchip

Es gratis y se consigue en el sitio de **Microchip** (Figura 2.15); es una excelente herramienta. Soporta todos los micros PIC, dsPIC y DSC, memorias y rfPIC:

- Trabaja en lenguajes **ASM** (ensamblador), **C** y **Basic**.
- Soporta todas las herramientas de Microchip para programadores, depuradores y emuladores y las herramientas de evaluación.
- Soporta el compilador de C de la casa **CCS** y el simulador **Proteus VSM** de **Labcenter Electronics**, también soporta **MatLab**.
- Posee las facilidades para realizar proyectos, definición de micros, visualización de registros, memoria de programa y EEPROM, el registro W y las banderas de Status.

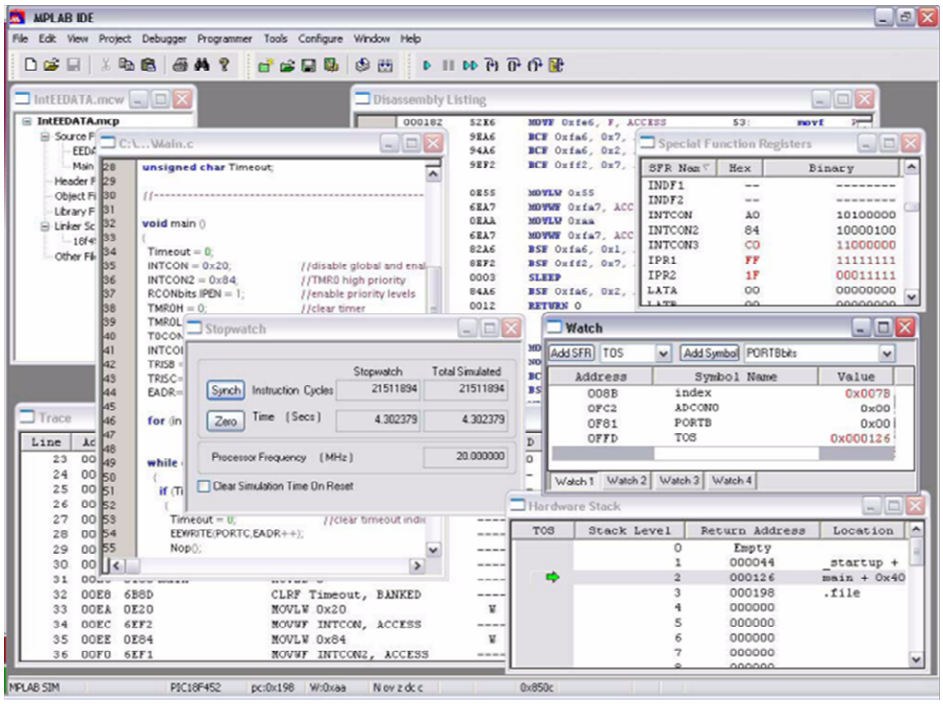


Figura 2.15 MPLAB IDE.

Otras herramientas de software se pueden encontrar en forma libre en Internet, tales como **WinPIC800** e **IC-PROG**.

➢ WinPIC800

Este software (Figura 2.16) es para **programación**, bastante difundido y cuenta con una gran cantidad de herramientas de programación en hardware que lo utiliza por su fácil implementación. Soporta **Microchip –PIC/10/12/16/18,** 16-Bit Flash

Microcontroladores– **dsPIC, 32-Bit PIC, EEPROM Serial I2C, Microwire EEPROM Serial 93XX, Microchip & Compatible**. También soporta **MCU** y **EEPROM** de otras marcas. Puede leer y escribir memoria de programa y **EEPROM**; igualmente puede cambiar los parámetros de configuración.

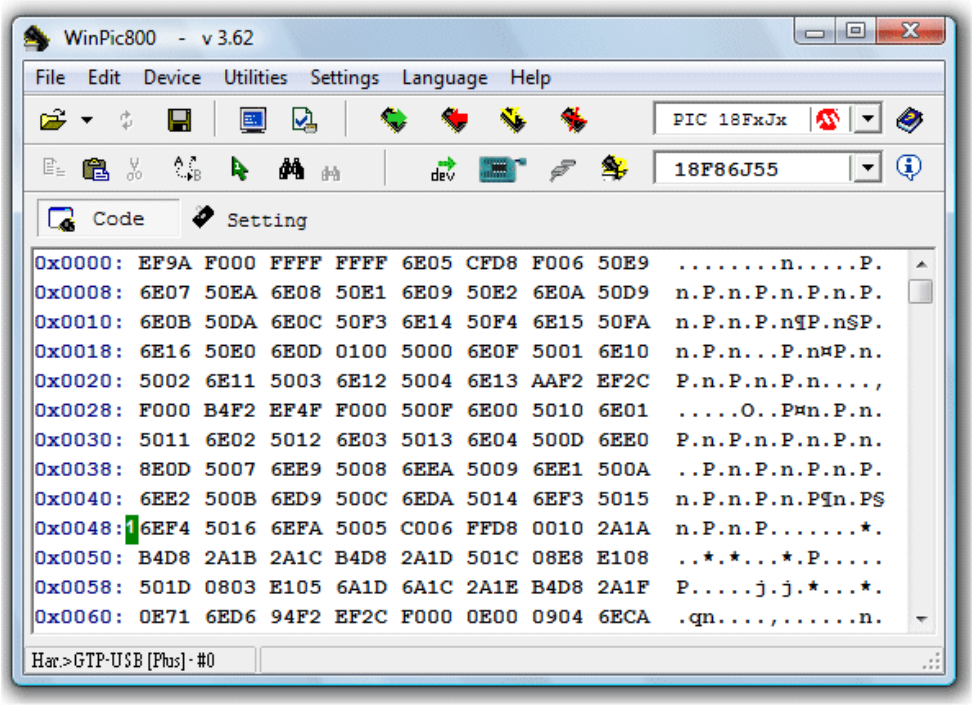


Figura 2.16 WinPIC800.

- IC-PROG

Este software también es para **programación** y es bastante difundido, cuenta con una buena cantidad de herramientas de hardware que lo utiliza por su fácil implementación, por ejemplo, como herramientas de programación de bajo costo. Soporta **Microchip PIC**, memorias **I2C EEPROM** y **4-Wire EEPROM**, **Microwire EEPROM**, lo mismo que dispositivos **MCU** y memorias de otras marcas. Puede leer y escribir memoria de programa y **EEPROM**; también puede cambiar los parámetros de configuración.

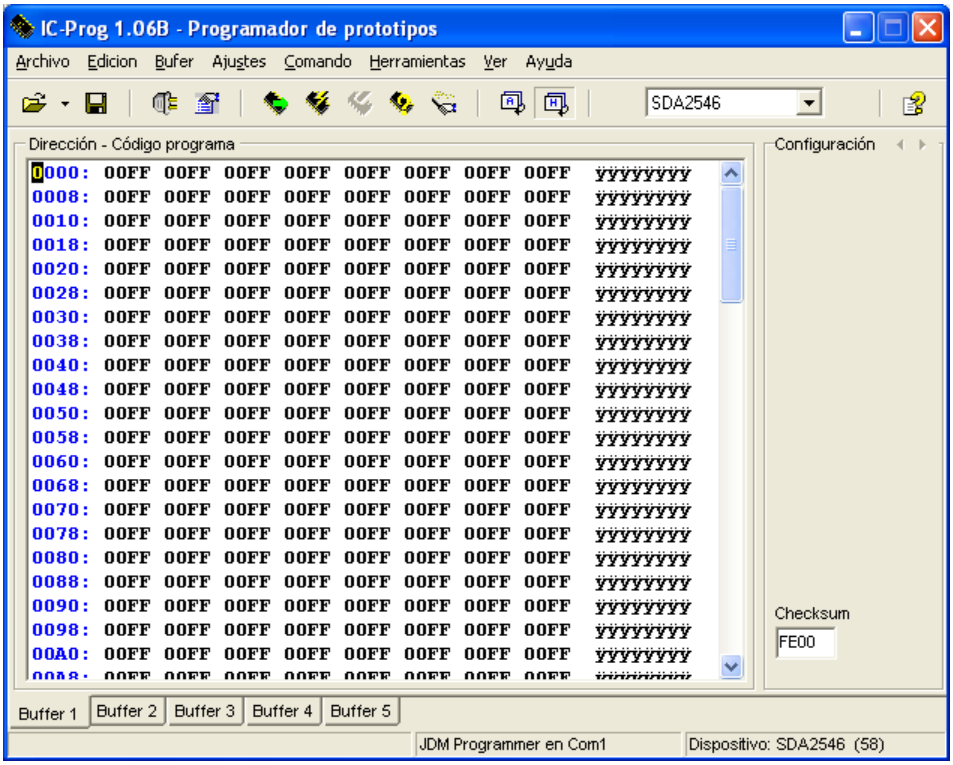


Figura 2.17 IC-PROG.

EVALUACIÓN

P.: 1. ¿Cuáles son los dos grandes tipos de herramientas?
R.:
Hardware y software.

P.: 2. Mencione los 3 tipos de herramientas de hardware.
R.:
a). Programador (*programmer*)
b). Depurador (*debugger*)
c). Emulador (*emulator*)

P.: 3. ¿Un programador (*programmer*) puede ver los registros internos del MCU?
R.:
No, solamente puede programar el MCU.

```
P.: 4. ¿Es un programador la herramienta más apropiada para un
    desarrollador?
R.:
No, la mejor es un depurador (debugger).

P.: 5. ¿En un emulador el MCU no se debe intercambiar?
R.:
Es correcto, no se debe, porque no hay MCU que se pueda
intercambiar.
```

3.0 INFORMACIÓN TÉCNICA DEL MCU

Cuando escogemos un MCU para trabajar, debemos considerar algunos aspectos técnicos relevantes, asociados con la información del MCU.

3.1 DATA SHEET (HOJA DE DATOS)

En la actualidad, todos los fabricantes de **MCU** tienen la información de sus dispositivos disponible en sitios Web en Internet. En el momento de trabajar con un MCU, debemos tener en cuenta lo siguiente:

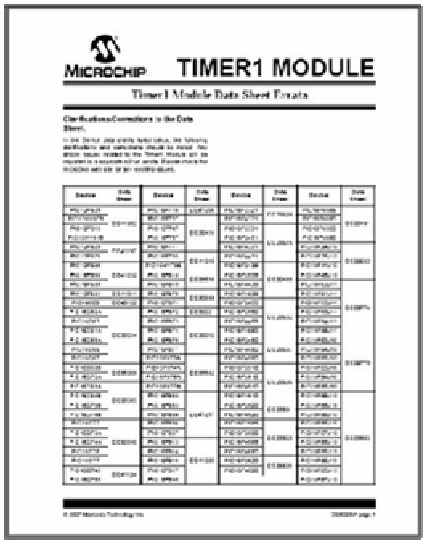
- Descargar su **hoja de datos** (*data sheet*) (Figura 3.1).
- Revisar la fecha de actualización y descargar la última revisión.
- Descargar erratas o addendum, o addenda, que son las adiciones hechas a la **hoja de datos** (*data sheet*), y en donde se hallan errores (*bugs*), problemas encontrados y soluciones planteadas.
- Descargar las **notas de aplicación** (*aplication notes*) y los diseños de referencia, para los módulos que se vayan a trabajar.

Los fabricantes generan esta información para facilitar la labor a los desarrolladores, con las recomendaciones de hardware y software necesarias para salvar los problemas que a veces se presentan, al implementar aplicaciones.

A



B



C



Figura 3.1 Data sheet, Addendum y Notas de aplicación.

3.2 PACKAGE (ENCAPSULADO)

Cuando escogemos un **MCU**, generalmente recordamos la referencia básica de identificación del MCU, y esto puede facilitar el pedido a un proveedor local, pero cuando vamos a otro proveedor no necesariamente el "conocido", que no tiene el mismo lenguaje coloquial que el nuestro, debemos especificar los pines y su disposición. Esta información (Figura 3.2) se encuentra al final de la **hoja de datos** (*data sheet*) del MCU, escogido en la sección *Packaging Information.*

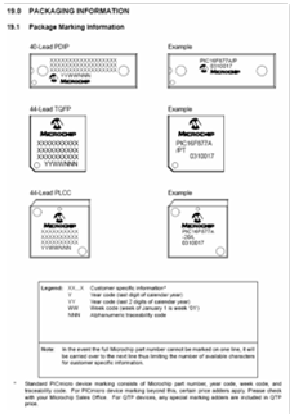


Figura 3.2 Información de encapsulado (package).

3.3 SISTEMAS DE IDENTIFICACIÓN DE PRODUCTO (PRODUCT IDENTIFICATION SYSTEM)

Lo mismo ocurre cuando tratamos de hacer un pedido; por ejemplo, en sitios de Internet como **Digikey** o **Mouser**; debemos especificar con detalle todos los caracteres de la referencia del chip, pues esto afecta el empaque, el rango de la temperatura de funcionamiento y lógicamente el precio. Esta información se encuentra al final de la hoja de datos del MCU escogido, en la sección *Product Identification System* (Figura 3.3).

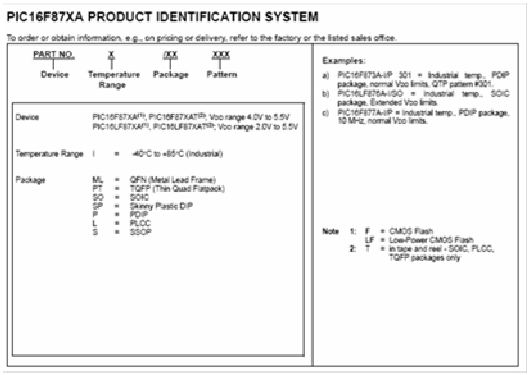


Figura 3.3 Sistema de Identificación del Producto.

3.4 SOPORTE

En la actualidad casi todos los fabricantes de los MCU proveen servicios de soporte (Figura 3.4) a sus usuarios a través de Internet, de varias maneras. El soporte por correo y foros son los tradicionalmente usados, pero también tienen líneas telefónicas y algunos en su idioma; especializados según sea el tema, como microcontroladores, herramientas e IDE, entre otros.

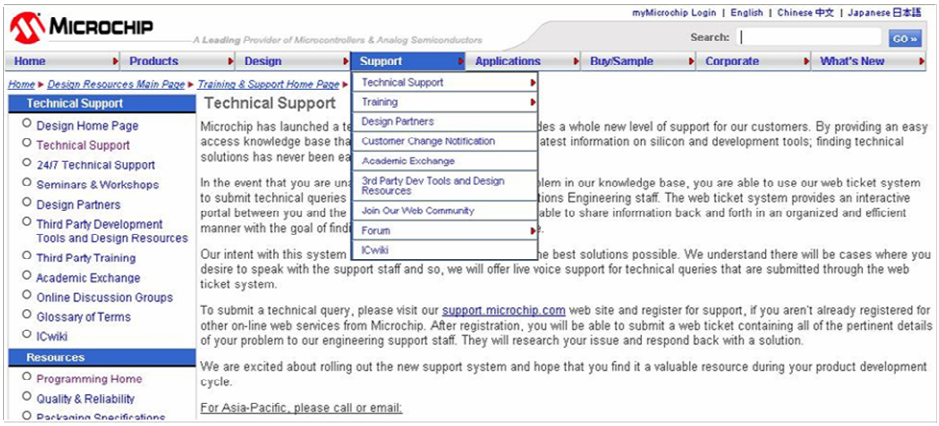


Figura 3.4 Soporte.

3.5 MUESTRAS

Algunos de los fabricantes de **MCU** acostumbran proveer servicio de envío de muestras del MCU, pero usualmente hay que registrar un proyecto y estimar las cantidades de producción. A los fabricantes les interesa que sean clientes serios en el momento de inscribir un proyecto, es decir, que tengan prospectos serios de producción y que respeten los canales formales de distribución del fabricante en la región donde se encuentra; por esta razón hacen seguimiento del estado del proyecto. No se recomienda esta práctica para amateurs, es más aconsejable hacerla con empresas que consuman cantidades mínimas de 5 000 piezas anuales.

PARTE 2

ARQUITECTURA DEL MCU

En esta parte 2 estudiaremos los conceptos y tipos de **arquitecturas,** los microcontroladores **Microchip,** sus familias y los grupos **PIC16F87XA y PIC16F88X**; dentro de estos grupos detallamos la arquitectura de los microcontroladores que trataremos en este libro: **PIC16F877A y PIC16F887** (en adelante **PIC16F877A-887**), con su descripción de pines, puertos y periféricos, la memoria de programa y los datos y, por último, los registros comunes y especiales de cada MCU.

4.0. CONCEPTOS DE ARQUITECTURA

El término **arquitectura** se refiere al diseño conceptual y a la estructura operacional de la máquina, a nivel de hardware, incluyendo las interconexiones entre la ALU, **memorias** y **periféricos**. También son parte importante las funciones especiales del **MCU**, tales como el **oscilador**, el *reset*, el **WDT**, las **interrupciones** y los demás circuitos necesarios para la operación apropiada del MCU. Veremos algunos conceptos y formas de clasificaciones, según la arquitectura.

4.1 DE MICROPROCESADORES A MICROCONTROLADORES

El **microprocesador (µP)** consiste en un chip que tiene una **ALU**, y proporciona el manejo de los buses de datos y direcciones, más la lógica en pines de control, de tal manera que mucha parte del sistema está afuera del chip, se debe poner

externamente el **circuito de reloj** (oscilador), la **memoria de programa ROM**, la **memoria de datos RAM** y los **periféricos**, como **USART** de comunicaciones y la **PIO** de puertos, lo mismo que otros **módulos periféricos**. En esta configuración se pueden agregar los periféricos que se deseen, extendiendo el sistema a otras posibilidades. De la evolución del **microprocesador** se llegó al **microcontrolador,** que es todo en uno. En la Figura 4.1 apreciamos esto más claramente.

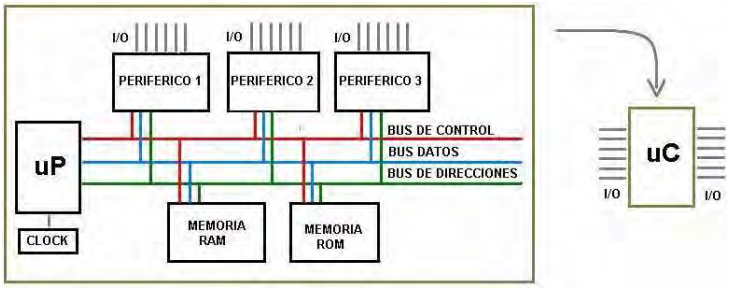


Figura 4.1 Microprocesador y microcontrolador.

Microcontrolador (µC): Un solo chip contiene todo el sistema que vemos en la Figura 4.1, en su interior, de tal manera que con sólo agregar pocos elementos tenemos ya un sistema completo, en el cual se pueden habilitar los periféricos que sean necesarios para nuestra aplicación. En caso de necesitarse más memoria u otros dispositivos periféricos, normalmente debemos cambiar de **µC.**

4.2 ARQUITECTURAS HARVARD Y VON NEUMANN

Según la disposición de los buses de datos y las direcciones, podemos hablar de las **arquitecturas Harvard y Von Neumann** (Figura 4.2). En la **arquitectura Harvard**, por tenerse acceso a la memoria de datos y a la memoria de programa mediante buses de datos y direcciones separados, se puede traer una instrucción y operar con un dato al mismo tiempo, debido a que no hay congestión en los buses. En la **arquitectura Von Neumann** se aprecia la congestión cuando tenemos que acceder la instrucción y luego acceder al dato que se va a manipular, porque se deben utilizar los mismos buses. La congestión que se genera implica un mayor tiempo de ejecución de los programas. La línea de microcontroladores PIC de **Microchip** son microcontroladores de arquitectura Harvard. En la figura 4.2 se aprecian con mayor claridad los diagramas de bloques de conexión.

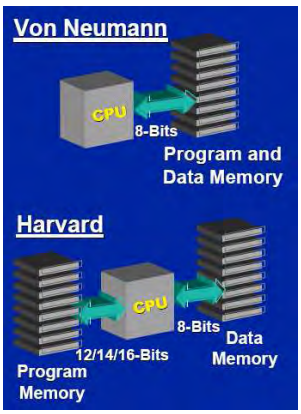


Figura 4.2 Arquitecturas Harvard y Von Neumann.

4.3 FASES DEL CICLO DE INSTRUCCIÓN

El **ciclo de máquina**, o **ciclo de instrucción**, es el tiempo que se toma el micro para ejecutar una instrucción; resulta de dividir la frecuencia del oscilador por cuatro fases; por ejemplo: con un oscilador de 20 MHz el Tcy = 4/(20 MHz) = 0.2 us.

Cada una de las cuatro fases **Q1, Q2, Q3 y Q4** son independientes y, en cuadratura; éstas sincronizan acciones dentro de la **circuitería** del micro. Así es como en cada **Q1** se incrementa el contador de programa PC, la instrucción es traída desde la memoria de programa y guardada en el registro de instrucciones en **Q4** ciclo *fetch INST* (PC). La instrucción es decodificada y ejecutada durante el próximo ciclo **Q1** a **Q4** ciclo *execute INST* (PC). En la Figura 4.3 se muestra un diagrama de tiempos de las fases **Q** y los ciclos Tcy. En un solo ciclo Q1 a Q4 se ejecutan dos acciones *fetch* y *Execute*, *fetch carga de INST* (PC) y *execute ejecución de INST* (PC-1).

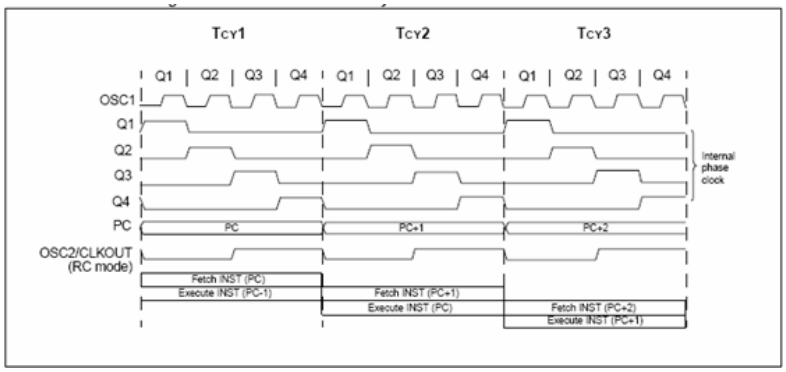


Figura 4.3 Fases del ciclo de instrucción.

En la Figura 4.4 apreciamos cómo es el flujo de instrucciones en una sección de programa. Aquí esta característica se conoce como *pipeline* o **proceso segmentado**; entonces, podemos resumir de la siguiente manera:

- Cada ciclo Tcy contiene cuatro fases = Q1 a Q4.
- Traer una instrucción toma un Tcy.
- Decodificarla y ejecutarla toma un Tcy.
- En un Tcy se trae la instrucción PC (actual) y se decodifica y ejecuta la instrucción PC-1 (anterior).

En el **ciclo de traída** *fetch* la PC se incrementa y trae la instrucción que se guarda en el registro de instrucciones. En el **ciclo de ejecución** *execute* se decodifica y ejecuta la instrucción. Estas dos acciones siempre se ejecutan en el mismo Tcy, sólo que cada acción se realiza con una instrucción diferente. Esto se aprecia mejor en la Figura 4.4.

— En el tiempo Tcy0 se carga la instrucción **1. MOVLW 55h.**

— En el tiempo TCy1 se ejecuta la instrucción **1 MOVLW 55h**, y se carga la instrucción **2. MOVWF PORTB.**

— En el tiempo TCy2 se ejecuta la instrucción **2. MOVWF PORTB**, y se carga la instrucción **3. CALL SUB_1.**

— En el tiempo TCy2 se ejecuta la instrucción **3. CALL SUB_1**, y se carga la instrucción **4. BSF PORTA, BIT3**, y así sucesivamente.

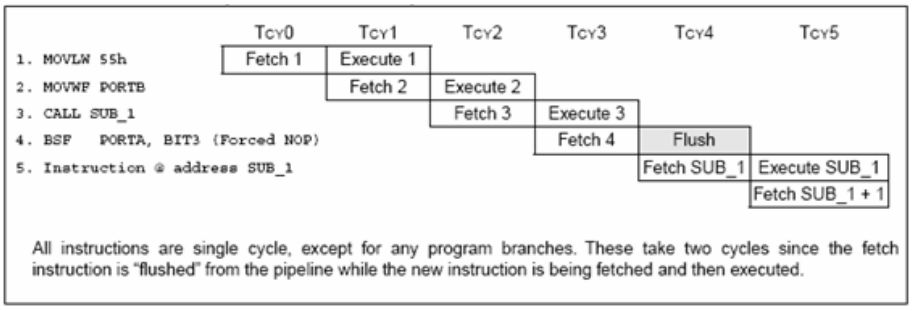


Figura 4.4 Flujo de instrucciones pipeline.

4.4 TIPOS DE SET DE INSTRUCCIONES

Según el tipo de *set* (conjunto) de instrucciones un micro se puede clasificar en:

- CISC (*Complex Instruction Set Computer:* Computadora de Conjunto de Instrucciones Complejo).
- RISC (*Reduced Instruction Set Computer:* Computadora de Conjunto de Instrucciones Reducido).
- SISC (*Specific Instruction Set Computer:* Computadora de Conjunto de Instrucciones Específico).

- **CISC.** Son procesadores con un conjunto de instrucciones complejo (más de 80). Sus instrucciones son muy poderosas, pero requieren muchos ciclos de máquina para la ejecución.
- **RISC.** Son procesadores con un conjunto de instrucciones reducido. Aquí entran los microcontroladores PIC; su conjunto de instrucciones es de 35.
- **SISC.** Son procesadores que poseen un conjunto de instrucciones específico para cada aplicación. Están destinados a aplicaciones muy concretas.

Estas tecnologías son usadas por los fabricantes de computadoras, procesadores, microprocesadores, microcontroladores y sistemas embebidos.

4.5 ORTOGONALIDAD

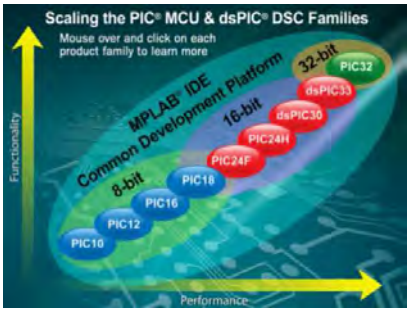
La **ortogonalidad** es una propiedad de las **CPU**. Se dice que un conjunto de instrucciones es ortogonal cuando se puede utilizar cualquier modo de direccionamiento, en cualquier instrucción. La búsqueda de la ortogonalidad hace que el diseño de la **CPU** sea más complejo, pero aporta una mayor facilidad de programación. Por la forma como están construidos, los **PIC** micro permiten que la dirección de cualquier instrucción pueda ser el registro W o cualquier dato de la memoria **RAM**. Recordemos que la memoria **RAM** se puede accesar de manera directa o indirecta con el par de registros **FSR** e **IND**.

5.0 LOS MICROCONTROLADORES PIC

La empresa **Microchip** fabrica varios productos electrónicos; entre ellos, la línea de **Microcontroladores PIC** existe en versiones de 8, 16 y 32 bits. Como apreciamos en la Figura 4.5. a, b, aparecen las familias de **microcontroladores**, así:

- En 8 bits: PIC10/12/16/18
- En 16 bits: PIC24F/H y dsPIC30/33
- En 32 bits: PIC32

a)



b)

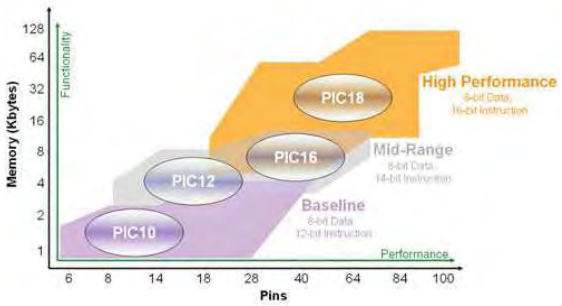


Figura 4.5 Familias de micros Microchip.

En la imagen observamos las familias entre los ejes de número de pines (*pins*) y cantidad de memoria en Kbytes (*memory*). Observamos, por tanto, que los microcontroladores que vamos a trabajar, **PIC16F877A-887**, pertenecen a la región de medio rango (*mid-range*).

5.1 LA FAMILIA PIC 10/12/16

En la Figura 4.6 se muestra la familia de micros **PIC10/12/16**; aquí encontramos los grupos de micros que vamos a trabajar en este libro, el grupo **PIC16F87XA** que contiene el micro **PIC16F877A** y otros de su grupo, y el grupo **PIC16F88X** que contiene el micro **PIC16F887** y otros de su grupo. Observamos que estos grupos y micros están entre los 28 y 40/44 pines; además, la memoria va desde 3.5 k a 14 k palabras (k words). Aquí conviene aclarar la diferencia entre *word* y *byte*; así:

- k words = 2 048 word (14 bits) = 3 584 (bytes) = 3.5 kB
- 8 k words = 8 192 word (14 bits) = 14 336 (bytes) = 14 kB
- 1 words= 14 bits (para estos MCU), 1 byte = 8 bits (cualquier MCU)

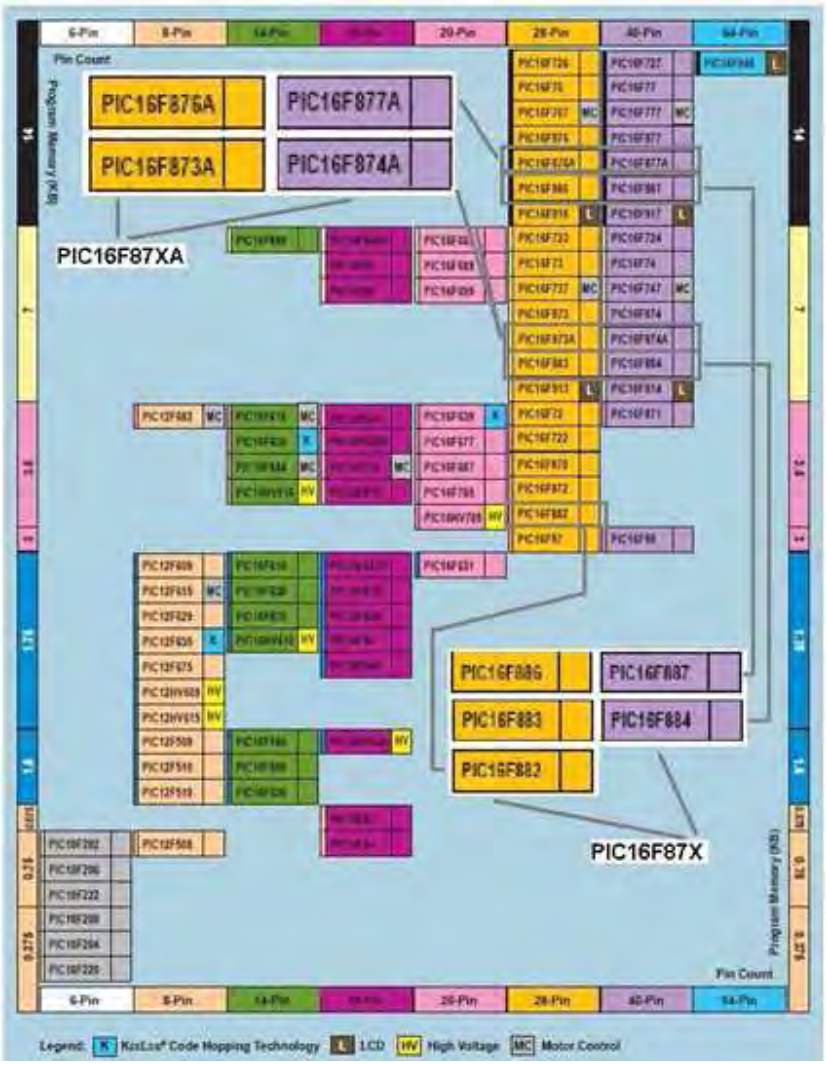


Figura 4.6 Familias PIC10/12/16.

5.2 LOS GRUPOS PIC16F87XA Y PIC16F88X

Como ya hemos visto, estos micros pertenecen a la familia **PIC16**, y entre ellos están los grupos **PIC16F87XA** y **PIC16F88X** que contienen varios dispositivos, entre los cuales son pin a pin compatibles en 28, 40 y 44 pines. En una pequeña tabla resumen (Tabla 5.1) observamos la capacidad de **memoria** y los pines de todos los dispositivos de ambos grupos.

Tabla 5.1 Resumen memoria PIN grupos PIC16F87XA y PIC16F88X.

PIC16F87XA	PIN	FLASH	SRAM	EEPROM
PIC16F873A	28	4096	192	128
PIC16F874A	40/44	4096	192	128
PIC16F876A	28	8192	368	256
PIC16F877A	40/44	8192	368	256

PIC16F88X	PIN	FLASH	SRAM	EEPROM
PIC16F882	28	2048	128	128
PIC16F883	28	4096	256	256
PIC16F884	40/44	4096	256	256
PIC16F886	28	8192	368	256
PIC16F887	40/44	8192	368	256

En las tablas 5.2 y 5.3 se resumen otras características de los grupos **PIC16F87XA** y **PIC16F88X**.

Tabla 5.2 Resumen de las características del grupo PIC16F87XA.

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

Tabla 5.3 Resumen de las características del grupo PIC16F88X.

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	24	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

Observemos que los micros tratados en este libro, **PIC16F877A-87**, son los de la variedad más alta en las Tablas 5.2 y 5.3. Para abordar la **arquitectura** de estos MCU, procederemos por partes, de modo que primero veremos a grandes rasgos las **características del MCU**, luego **Pines-puertos y periféricos**, continuando con el área de **Memoria**, para luego ver **Registros comunes** y **Funciones especiales**.

En **Pines-puertos y periféricos** (Secciones 6.2 y 7.2), vemos el *pinout*, después describimos los **Puertos y periféricos**, detallando los pines, para luego observar un **Diagrama de bloques** que contiene todas las partes del MCU interconectadas. En **Memoria** (Secciones 6.3 y 7.3) las describimos por tipo: de programas o **FLASH (ROM)**, de datos **RAM** con sus registros especiales y de usuario y, por último, la

memoria **EEPROM**. Los **Registros comunes** (Capítulo 8.0) y las **Funciones especiales** (Capítulo 9.0) son partes en común a los **MCU PIC16F877A-887**.

6.0. ARQUITECTURA DEL PIC16F877A

6.1 CARACTERÍSTICAS (BROCHURE)

Estas características (Tablas 6.1 y 6.2) son algunas de las que resaltan el **PIC16F877A**. Con estas características se incluye el grupo de los micros:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

Tabla 6.1 Características (brochure).

RISC CPU

Velocidad de Operación	**DC - 20 Mhz**
Ciclo de instrucción	**DC - 20o ns**
Set de Instrucciones	**35**
Toda las Instrucciones de 1 ciclo, excepción saltos de 2 ciclos	
Memoria de Programa (FLASH)	8K x 14 words
Memoria de Datos	368 x 8 bytes
Memoria de EEPROM	256 x 8 bytes
Pin-Out compatible con otros MCU de 28 o 40/44 pin	
grupos PIC16CXXX y PIC16FXXX	

PERIFERICOS

3 TIMERS (Timer0,1,2)
Timer0 : 8 bit temporizador/contador con 8 bit prescaler
Timer1 : 16 bit temporizador/contador con prescaler
puede ser incrementado durante Sleep con clock externo
Timer2 : 8 bit temporizador/contador con 8 bit registro de periodo
prescaler y postscaler

2 CCP (Capture/compare/PWM) Captura es de 16bit, máxima resolución es de 12.5ns Compare es de 16bit, máxima resolución es de 200ns PWM con máxima resolución de 10 bit
SSP (Sinchronous Serial Port) SPI (Modo Maestro), I2C (Modos: Maestro/Esclavo) USART/SCI (Universal Synchronous Asynchronous Receiver) con 9 bit de detección de dirección PSP Puerto Paralelo Esclavo (Parallel Slave Port) de 8 bit con controles RD,WR y CS (40/44 pin)
CAD (Conversor Análogo a Digital) de 10 bit, hasta 8 canales
Modulo de Comparadores Análogos 2 Comparadores análogos Programable Modulo de voltaje de referencia (Vref) 8 formas de configuración con entradas, salidas y referencia Salidas del comparador accequibles

Tabla 6.2 Características del grupo PIC16F87XA.

Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

6.2 PIN-PUERTOS Y PERIFÉRICOS

6.2.1 DESCRIPCIÓN DE PINES (PINOUT)

La **configuración de los pines** depende del tipo de empaque (Tabla 6.3 y Figura 6.1) que estemos describiendo; de ese modo tenemos varios tipos de empaque, tales como: **PDIP, SOIC, SSOP, QFN, PLCC, TQFP**.

Tabla 6.3 Empaques PIC16F877A y grupo PIC16F87XA

PIC16F87XA	PIN				
PIC16F873A	28	PDIP	SOIC	SSOP	QFN
PIC16F874A	40/44	PDIP	PLCC	TQFP	QFN
PIC16F876A	28	PDIP	SOIC	SSOP	QFN
PIC16F877A	40/44	PDIP	PLCC	TQFP	QFN

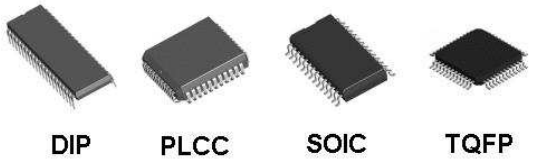


Figura 6.1 Diferentes tipos de Packaged.

Para nuestro caso, el *pinout* de 40 pines **PDIP** (*Plastic Dual In Package*) usado comúnmente para **protoboard**, o con base de 100 mils de separación entre pines y 600 mils de ancho, este empaque y disposición de pines son los que trabajaremos y describiremos a lo largo de este libro (Figura 6.2).

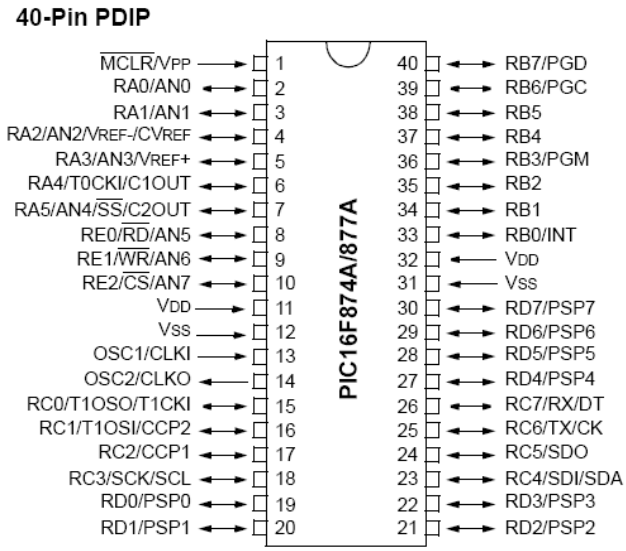


Figura 6.2 Pinout PIC16F877A 40-PIN DIP

6.2.2 PUERTOS Y PERIFÉRICOS

Los **puertos** pueden o no compartir el pin con algún **periférico**; a veces, con más de un periférico, la función con la que el pin queda trabajando depende de la programación que realiza el usuario. Para la configuración de cada periférico, se deben usar adecuadamente los registros asociados y un procedimiento secuencial; todo esto se detalla más adelante en cada módulo. A continuación (Tabla 6.4) apreciamos un resumen de los periféricos que contiene el **PIC16F877A** y cada dispositivo del grupo **PIC16F87XA**.

Tabla 6.4 Puertos y periféricos PIC16F877A y GRUPO PIC16F87XA.

Key Features	PIC16F873A	PIC16F874A	PIC16F876A	PIC16F877A
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Analog Comparators	2	2	2	2

En resumen, todos tienen:

- 3 Timers, TMR0, TMR1, TMR2
- 2 Capture/Compare PWM
- 2 Comparadores análogos
- Comunicación MSSP, USART

Las diferencias están en:

- 28 o 40/44 pines
- 5 u 8 conversores análogos a digital de 10 bits
- 3 o 5 puertos I/O

En la Tabla 6.5 se describen con detalle los pines y puertos con periféricos compartidos para el microcontrolador PIC16F877A. Cada módulo pertenece a un periférico que realiza una función específica, y puede contener más de un puerto. Es bueno anotar que se debe escoger sólo una función o un módulo periférico por cada pin; no aconsejamos usar varias funciones sobre el mismo pin; esto es únicamente para desarrolladores expertos. Debido a que se puede comprometer al MCU, y podría dañar sus pines en casos en los que se pueda confundir una salida con una entrada (un error ocasional es programar un pin al que se conecta un interruptor [switch] como salida, al presionarlo; si la salida está en ALTO, se produce un cortocircuito a GND y se puede dañar el mosfet del circuito de salida del puerto).

Tabla 6.5 Descripción de pines y periféricos PIC16F877A.

PINES PUERTOS PERIFERICOS PIC16F877A						
PIN	NOMBRE PIN	PUERTO	PERIFERICOS			NOTAS
2	RA0/AN0	RA0	AN0			
3	RA1/AN1	RA1	AN1			
4	RA2/AN2/VREF-/CVREF	RA2	AN2	VREF-	CVREF	
5	RA3/AN3/VREF+	RA3	AN3	VREF+		
6	RA4/T0CKI/C1OUT	RA4	T0CKI	C1OUT		
7	RA5/AN4/SS/C2OUT	RA5	AN4	SS	C2OUT	OPEN-DRAIN
33	RB0/INT	RB0	INT			PULLUPS
34	RB1	RB1				PULLUPS
35	RB2	RB2				PULLUPS
36	RB3/PGM	RB3	PGM			PULLUPS
37	RB4	RB4				PULLUPS
38	RB5	RB5				PULLUPS
39	RB6/PGC	RB6	PGC			PULLUPS
40	RB7/PGD	RB7	PGD			PULLUPS
15	RC0/T1OSO/T1CKI	RC0	T1OSO	T1CKI		
16	RC1/T1OSI/CCP2	RC1	T1OSI	CCP2		
17	RC2/CCP1	RC2	CCP1			
18	RC3/SCK/SCL	RC3	SCK	SCL		
23	RC4/SDI/SDA	RC4	SDI	SDA		
24	RC5/SDO	RC5	SDO			
25	RC6/TX/CK	RC6	TX	CK		
26	RC7/RX/DT	RC7	RX	DT		
19	RD0/PSP0	RD0	PSP0			
20	RD1/PSP1	RD1	PSP1			
21	RD2/PSP2	RD2	PSP2			
22	RD3/PSP3	RD3	PSP3			
27	RD4/PSP4	RD4	PSP4			
28	RD5/PSP5	RD5	PSP5			
29	RD6/PSP6	RD6	PSP6			
30	RD7/PSP7	RD7	PSP7			
8	RE0/RD/AN5	RE0	RD	AN5		
9	RE1/WR/AN6	RE1	WR	AN6		
10	RE2/CS/AN7	RE2	CS	AN7		
1	MCLR/VPP	MCLR	VPP			
11	VDD					5V
12	VSS					GND
13	OSC1/CLKI	OSC1	CLKI			
14	OSC2/CLKO	OSC2	CLKO			
31	VDD					5V
32	VSS					GND

MODULOS	DESCRIPCION
SPECIAL	Funciones Especiales MCU
SCI	Serial Comunication Interface
SPI	Serial Protocol Interface
PROG	Programación
A/D	Analogo a Digital
PARLL	Paralelo
SCI-1	Serial Comunication Interface 1
CCP	Capture Compare PWM
I2C	I2C
TIMER	Temporizadores
COMP	Comparadores

6.2.3 DIAGRAMA DE BLOQUES

En el diagrama de bloques (Figura 6.3) podemos observar cada una de las partes del MCU y la interconexión entre ellas; y detallar las grandes partes, tales como **puertos, periféricos, memoria de datos RAM, memoria de programa FLASH**, **ALU** y otras partes (no resaltadas). También podemos ver resaltados los **buses de datos** y de programa.

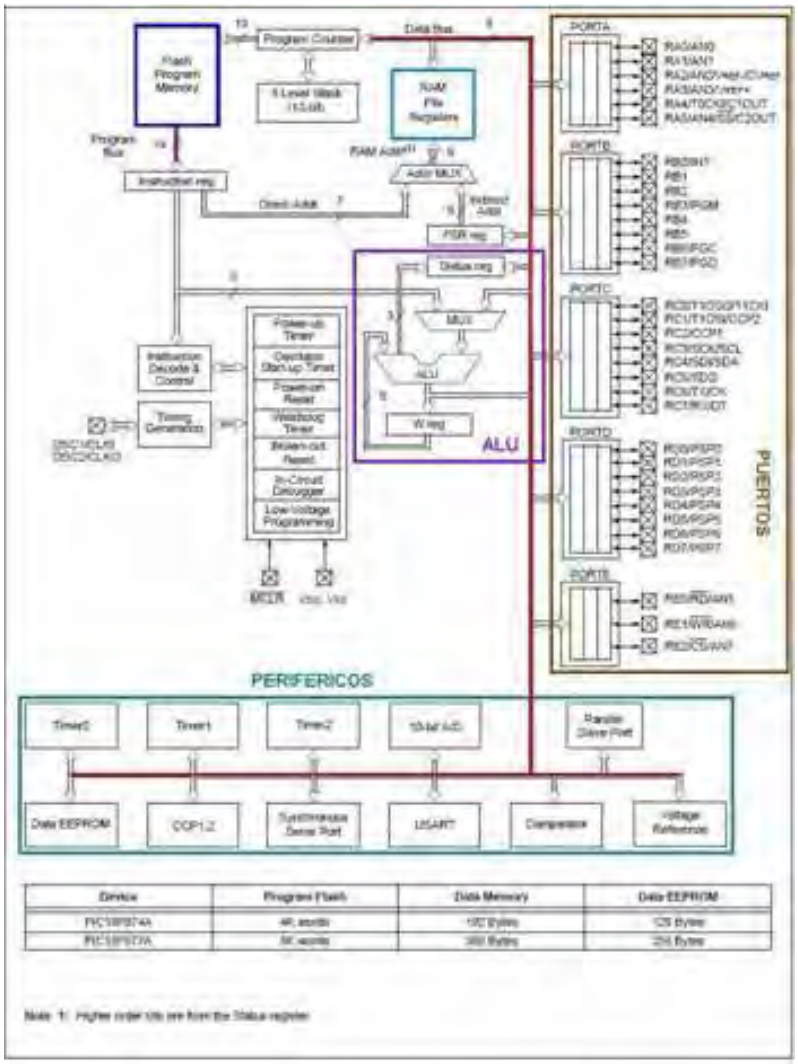


Figura 6.3 Diagrama de bloques PIC16F877A

https://yolibrospdf.com

6.3 MEMORIA

En los **PIC** micros **MCU** hay tres tipos de memoria:

- Memoria de programa (*program memory*) – instrucciones.
- Memoria de datos (*data memory*) – valores de datos variables.
  - Registros de funciones especiales (**SFR**: *Special Function Register*).
    - Controlan la operación del dispositivo.
  - Registros de propósito general (**GPR**: *General Purpose Registers*).
    - Almacenamiento de RAM de datos del usuario.
- Memoria de EEPROM –almacenamiento de datos de usuario no volátil.

En seguida veremos cada una de estas partes de la memoria.

6.3.1 MEMORIA DE PROGRAMA Y STACK

En la **memoria de programa** va alojado el programa con el que el MCU funcionará. En la Figura 6.4 apreciamos el mapa de memoria de programa, el contador de programa PC de 13 bits (12:0), que tiene la posibilidad de direccionamiento de $2^{13} =$ **8 192** posiciones **=> 0 – 1FFFh**. De tal manera que el contador de programa PC puede recorrer todo el espacio del mapa de la memoria de programa.

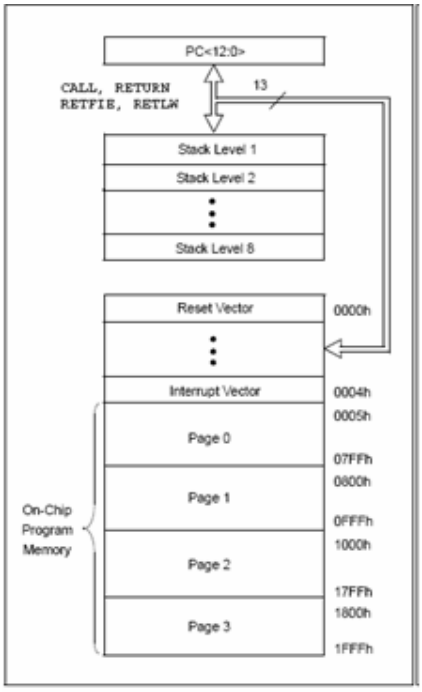


Figura 6.4
Memoria de programa y Stack PIC16F876A/877A.

Cada posición puede alojar una instrucción, perteneciente a un programa. El ancho de cada instrucción se denomina **palabra** (*word*) que para este MCU es de 14 bits. Una palabra de programa contiene el comando o el código de instrucción y su(s) operando(s) (Tabla 6.6).

La instrucción:

bsff,b; pone en 1 el bit (b) del registro (f)

Por ejemplo:
bsf PORTA,4 pone en 1 el bit (4) del registro (PORTA)

El código de la instrucción es 01 01bb bfff ffff

donde bbb es el bit (de 0-7, valor de 3 bits bbb)

y fff ffff es el registro (de 0-127, valor de 7 bits fff ffff)

Si hacemos los reemplazos de los valores, obtenemos:

con el bit b=04h=100

y el registro f=PORTA=05h=000 0101

el código de la instrucción es 01 0110 0000 0101 o 1605h que equivale a
bsf PORTA,4

Tabla 6.6 Ejemplo instrucción.

EJEMPLO DE CONVERSION BINARIO A HEXADECIMAL (14 BIT)																	
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	No. BIT	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	BIT	
		0	1	0	1	b	b	b	f	f	f	f	f	f	f	BINARIO	bsf f,b
		0	1	0	1	1	0	0	0	0	0	0	1	0	1	BINARIO	bsf PORTA,4
1				6				0				5				HEXA	bsf PORTA,4

Observemos que la **instrucción** es de 14 bits de ancho (bit 0 a 13); éste es un ejemplo de una instrucción en un programa, ocupando una sola palabra. El área de programas está dividido en cuatro **páginas** (*page* 0:3) cada uno de 2 048 posiciones (2 k words), para un total de 8 192 posiciones (8 k words). Ésta es la cantidad máxima de instrucciones que debe tener el programa para que se pueda alojar en la memoria del MCU.

```
NOTA: Las posiciones de memoria de programa son de un ancho de 14
bits por palabra (word), (bit 0 a 13); las llamamos palabras porque
no podemos denominarlas byte (byte = 8 bits).
```

Las **páginas** o los **bloques de memoria de programas** (Figura 6.5) se direccionan trabajando con el registro **PC-LATH** (4:3); así:

PC-LATH (4:3) = 00 Page 0, 0000 – 07FFh
PC-LATH (4:3) = 01 Page 1, 0800 – 0FFFh
PC-LATH (4:3) = 10 Page 2, 1000 – 17FFh
PC-LATH (4:3) = 11 Page 3, 1800 – 1FFFh

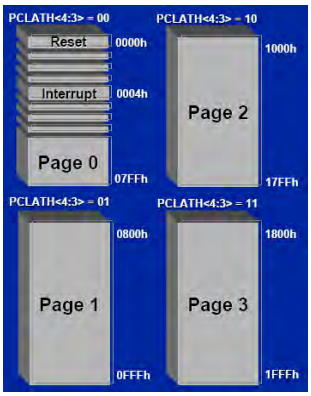


Figura 6.5 Memoria de programa (pages).

Se debe tener esto en cuenta cuando nuestro programa se hace muy extenso, y hay que llamar a subrutinas que quedan en otras páginas. El *stack* tiene 8 niveles, lo que nos indica que podemos usar hasta un máximo de 8 niveles de llamados a subrutinas anidados, como observamos en las Figuras 6.6 y 6.7.

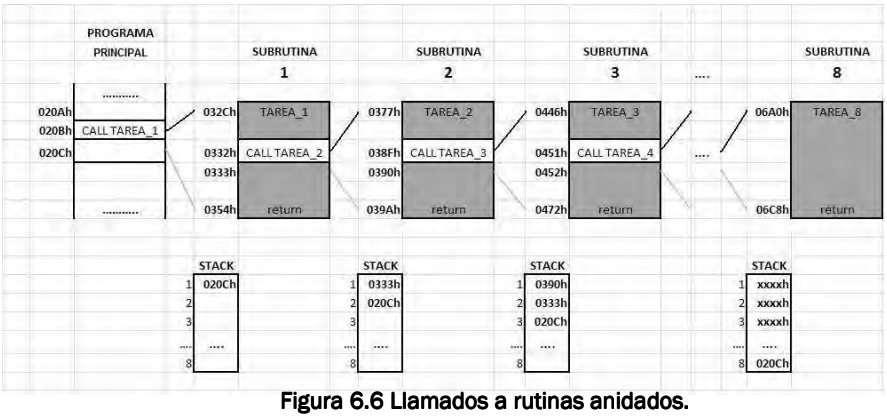


Figura 6.6 Llamados a rutinas anidados.

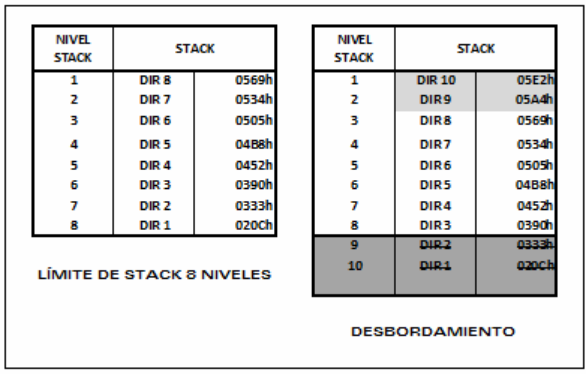


Figura 6.7 Llamados a rutinas anidados.

El vector de *reset* es la posición **0000h** de la memoria de programa y el vector de Interrupción es el **0004h**; significa que al ocurrir estos eventos, el de *reset* o el de interrupción, el contador de programa se direcciona a estas posiciones, y desde allí podemos direccionar a la posición de programa en donde estén alojadas nuestras rutinas de *reset* y la de servicio de **interrupción**.

6.3.2 MEMORIA DE DATOS

La **memoria de datos** es en donde se almacenan todos los registros propios del MCU y los registros propios del usuario. De esta forma se divide en dos partes; a saber:

- **SFR** (*Special Function Registers:* Registros de Funciones Especiales); éstos son los registros propios del MCU asociados con los periféricos y/o puertos y el funcionamiento en general del MCU. Estos registros guardan las configuraciones de periféricos, la lectura y la escritura de valores en el periférico o el puerto; también, la generación de **banderas**, *flags,* habilitación de interrupciones, *buffers* de entrada y salida de datos; en fin, todos los registros necesarios para el funcionamiento de todos los módulos del MCU.
- **GPR** (*General Purpose Registers:* Registros de Propósito General); éstos son los registros en donde se pueden almacenar los datos del usuario.

Es necesario tener en cuenta que la memoria de datos está dividida en cuatro bancos que se deben direccionar previamente para acceder el registro. Esto se logra con los bits RP1 y RP0 del registro STATUS<6:5>, como se muestra en la Figura 6.8 y la Tabla 6.7.

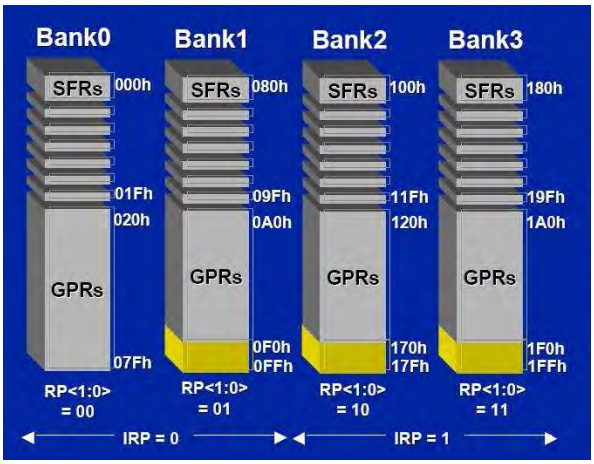


Figura 6.8 Memoria de datos.

Tabla 6.7 Direccionamiento de bancos y posiciones.

STATUS <6:5> RP1:RP0	BANCO	POSICIONES
00	0	000 - 07Fh
01	1	080 - 0FFh
10	2	100 - 17Fh
11	3	180 - 1FFh

A continuación (Tabla 6.8) veremos el área de memoria de datos, con ambos tipos de registros **SFR** y **GPR**, con sus posiciones respectivas.

Tabla 6.8 Mapa de registros.

Mapa de Registros PIC16F877A

File	Addr	File	Addr	File	Addr	File	Addr
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD	08h	TRISD	88h		108h		188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh		18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h	General	116h	General	196h
CCP1CON	17h		97h	Purpose	117h	Purpose	197h
RCSTA	18h	TXSTA	98h	Register (1)	118h	Register (1)	198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch	CMCON	9Ch		11Ch		19Ch
CCP2CON	1Dh	CVRCON	9Dh		11Dh		19Dh
ADRES	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers (2) 96 Bytes	20h 7Fh	General Purpose Registers (3) 80 Bytes	A0h EFh	General Purpose Registers (3) 80 Bytes	120h 16Fh	General Purpose Registers (3) 80 Bytes	1A0h 1EFh
		Mapped in Bank0 70h - 7Fh (4)	F0h FFh	Mapped in Bank0 70h - 7Fh (4)	170h 17Fh	Mapped in Bank0 70h - 7Fh (4)	1F0h 1FFh

Bank0		Bank1		Bank2		Bank3	
[RP1:RP0]	00	[RP1:RP0]	01	[RP1:RP0]	10	[RP1:RP0]	11
[IRP] 0				[IRP] 1			

Observamos que algunos registros **SFR** ocupan la misma posición en cada uno de los bancos, registros como **PCL(02h)**, **STATUS(03h)**, **FSR(04h)**, **PCLTAH(0Ah)**, **INTCON(0Bh)**. Esto quiere decir que solamente hay un registro (por ejemplo, **STATUS**) que comparte cuatro direcciones lógicas. Esto se hace para facilitar la programación en el acceso a estos registros sin tener que conmutar bancos y, por tanto, conservando el banco en el que estamos trabajando.

De la misma manera podemos encontrar el área en los registros **GPR** en las posiciones **70h a 7Fh**, las que siempre serán accedidas desde cualquier banco, lo que indica que únicamente hay **0Fh (16d)** posiciones, que se comparten en todos los bancos. En la Tabla 6.9 vemos las posiciones de memoria y su cantidad para hacer un total de **memoria GPR** o de usuario de 368 bytes.

Tabla 6.9 Registros de usuario (GPR) y posiciones

CANTIDAD DE GPR (RAM)	
POSICIONES	N REG.
020 - 07Fh	96
0A0 - 0EFh	80
110 - 16Fh	96
190 - 1EFh	96
TOTAL	368

6.3.3 REGISTROS DE FUNCIONES ESPECIALES (SFR)

A continuación en la Tabla 6.10 encontramos un resumen de los registros de funciones especiales o **SFR**, y se detallan: el banco al que pertenece, la dirección del registro **xxxh**, el nombre del registro, los bits contenidos, el valor sobre un (*Power-On Reset*) y sobre un (*Brown-Out Reset*); además, la página en donde se describe en la *data sheet* **DS39582B** o **PIC16F87XA**.pdf. Este archivo PDF puede encontrarlo en nuestra página web en la carpeta **Data sheet**. Estos registros se explican de manera amplia en cada uno de los módulos periféricos; algunos de ellos se describen en el **Capítulo 8: "Registros comunes"**.

Tabla 6.10 Resumen de Registros de Funciones Especiales grupo PIC16F87XA.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 0											
00h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
01h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	43, 150
06h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	45, 150
07h	PORTC	PORTC Data Latch when written: PORTC pins when read								xxxx xxxx	47, 150
08h(4)	PORTD	PORTD Data Latch when written: PORTD pins when read								xxxx xxxx	48, 150
09h(4)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	49, 150
0Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
0Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
0Ch	PIR1	PSPIF(3)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	26, 150
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	28, 150
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	60, 150
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	--00 0000	57, 150
11h	TMR2	Timer2 Module Register								0000 0000	62, 150
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	61, 150
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	79, 150
14h	SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	82, 82, 150
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	63, 150
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	63, 150
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	64, 150
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX8D	0000 000x	112, 150
19h	TXREG	USART Transmit Data Register								0000 0000	118, 150
1Ah	RCREG	USART Receive Data Register								0000 0000	118, 150
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	63, 150
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	63, 150
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	64, 150
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	133, 150
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{DONE}$	—	ADON	0000 00-0	127, 150

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.
5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

(*Continúa*)

Tabla 6.10 Resumen de Registros de Funciones Especiales grupo PIC16F87XA.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 1											
80h(3)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
82h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
83h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
84h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	43, 150
86h	TRISB	PORTB Data Direction Register								1111 1111	45, 150
87h	TRISC	PORTC Data Direction Register								1111 1111	47, 150
88h(4)	TRISD	PORTD Data Direction Register								1111 1111	48, 151
89h(4)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	50, 151
8Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
8Bh(3)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
8Ch	PIE1	PSPIE(2)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	25, 151
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	29, 151
8Fh	—	Unimplemented								—	—
90h	—	Unimplemented								—	—
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	83, 151
92h	PR2	Timer2 Period Register								1111 1111	62, 151
93h	SSPADD	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	79, 151
94h	SSPSTAT	SMP	CKE	$D/\overline{A}$	P	S	$R/\overline{W}$	UA	BF	0000 0000	79, 151
95h	—	Unimplemented								—	—
96h	—	Unimplemented								—	—
97h	—	Unimplemented								—	—
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	111, 151
99h	SPBRG	Baud Rate Generator Register								0000 0000	113, 151
9Ah	—	Unimplemented								—	—
9Bh	—	Unimplemented								—	—
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	135, 151
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	141, 151
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	133, 151
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	128, 151

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.
2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.
3: These registers can be addressed from any bank.
4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.
5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

(*Continúa*)

Tabla 6.10 Resumen de Registros de Funciones Especiales grupo PIC16F87XA.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
Bank 2											
100h(4)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
101h	TMR0	Timer0 Module Register								xxxx xxxx	55, 150
102h(4)	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	30, 150
103h(4)	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	22, 150
104h(4)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
105h	—	Unimplemented								—	—
106h	PORTB	PORTB Data Latch when written: PORTB pins when read								xxxx xxxx	46, 150
107h	—	Unimplemented								—	—
108h	—	Unimplemented								—	—
109h	—	Unimplemented								—	—
10Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
10Bh(4)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
10Ch	EEDATA	EEPROM Data Register Low Byte								xxxx xxxx	39, 151
10Dh	EEADR	EEPROM Address Register Low Byte								xxxx xxxx	39, 151
10Eh	EEDATH	—	—	EEPROM Data Register High Byte						--xx xxxx	39, 151
10Fh	EEADRH	—	—	—	—(5)	EEPROM Address Register High Byte				---- xxxx	39, 151
Bank 3											
180h(4)	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								0000 0000	31, 150
181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	23, 150
182h(4)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
183h(4)	STATUS	IRP	RP1	RP0	TO	PD	Z	DC	C	0001 1xxx	22, 150
184h(4)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
185h	—	Unimplemented								—	—
186h	TRISB	PORTB Data Direction Register								1111 1111	46, 150
187h	—	Unimplemented								—	—
188h	—	Unimplemented								—	—
189h	—	Unimplemented								—	—
18Ah(1,3)	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	30, 150
18Bh(4)	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	24, 150
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	34, 151
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	39, 151
18Eh	—	Reserved; maintain clear								0000 0000	—
18Fh	—	Reserved; maintain clear								0000 0000	—

Legend: x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved. Shaded locations are unimplemented, read as '0'.

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a holding register for the PC<12:8>, whose contents are transferred to the upper byte of the program counter.

2: Bits PSPIE and PSPIF are reserved on PIC16F873A/876A devices; always maintain these bits clear.

3: These registers can be addressed from any bank.

4: PORTD, PORTE, TRISD and TRISE are not implemented on PIC16F873A/876A devices, read as '0'.

5: Bit 4 of EEADRH implemented only on the PIC16F876A/877A devices.

6.3.4 MEMORIA EEPROM

La **memoria EEPROM** es una memoria de datos de usuario que tiene la particularidad de no borrarse si falta la alimentación de voltaje. Por tanto, es muy útil en el momento de realizar tablas de datos que se van construyendo a medida que el programa va generando estos datos, de tal manera que se puedan guardar como parámetros de calibración o funcionamiento del dispositivo. A esta memoria EEPROM (Figura 6.9) se accede mediante un procedimiento, no es posible acceder a los datos por un direccionamiento. Observamos que la ubicación de esta memoria está en las direcciones 2100h a 21FFh, que son posiciones de memoria de programa, más allá de los lugares permitidos.

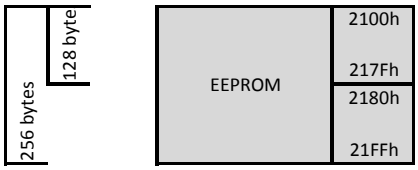


Figura 6.9 Memoria EEPROM.

7.0 ARQUITECTURA DEL PIC16F887

7.1 CARACTERÍSTICAS (BROCHURE)

Estas características (Tabla 7.1) son algunas de las que resaltan el **PIC16F887**. Con estas características se incluye el grupo de los micros:

- PIC16F882
- PIC16F883
- PIC16F884
- PIC16F886
- PIC16F887

Tabla 7.1 Características (Brochure).

RISC CPU	
Velocidad de Operación	DC - 20 Mhz
Ciclo de instrucción	DC - 200 ns
Set de Instrucciones	35
Toda las Instrucciones de 1 ciclo, a excepción saltos de 2 ciclos	

Memoria de Programa (FLASH)	8K x 14 words
Memoria de Datos	368 x 8 bytes
Memoria de EEPROM	256 x 8 bytes
Pin-Out compatible con otros MCU de 28 o 40/44 pin grupos PIC16CXXX y PIC16FXXX	
Oscillator Interno - Calibrado de Fabrica ±1% - Rango de frecuencia 8 MHz to 31 kHz	

PERIFERICOS

3 TIMERS (Timer0,1,2) Timer0 : 8 bit temporizador/contador con 8 bit prescaler Timer1 : 16 bit temporizador/contador con prescaler puede ser incrementado durante Sleep con clock externo Timer2 : 8 bit temporizador/contador con 8 bit registro de periodo prescaler y postscaler
2 ECCP (Enhanced /Capture/compare/PWM) Captura es de 16bit, máxima resolución es de 12.5ns Compare es de 16bit, máxima resolución es de 200ns PWM con máxima resolución de 10 bit, max frec. 20khz PWM con 1,2,4 canales y tiempo muerto programable
MSSP (Sinchronous Serial Port) SPI (Modo Maestro), I2C (Modos: Maestro/Esclavo) EUSART/SCI (Enhanced Universal Synchronous Asynchronous Receiver) con 9 bit de detección de dirección soporta RS-232 y RS-485 y LIN 2.0
CAD (Conversor Análogo a Digital) de 10 bit, hasta 14 canales
Módulo de Comparadores Análogos 2 Comparadores análogos Módulo programable de voltaje de referencia (Vref) 8 formas de configuración con entradas, salidas y referencia

Salidas del comparador accequibles

ESPECIALES

100.000 Ciclos de escritura/borrado para memoria de programa FLASH 1´000.000 Ciclos de escritura/borrado para memoria de datos EEPROM Retención de datos EEPROM > 40 años
Programación ICSP (In Circuit Serial Programming) Depuración In circuit

Tabla 7.2 Características del grupo PIC16F88X.

Device	Program Memory	Data Memory		I/O	10-bit A/D (ch)	ECCP/ CCP	EUSART	MSSP	Comparators	Timers 8/16-bit
	Flash (words)	SRAM (bytes)	EEPROM (bytes)							
PIC16F882	2048	128	128	24	11	1/1	1	1	2	2/1
PIC16F883	4096	256	256	24	11	1/1	1	1	2	2/1
PIC16F884	4096	256	256	35	14	1/1	1	1	2	2/1
PIC16F886	8192	368	256	24	11	1/1	1	1	2	2/1
PIC16F887	8192	368	256	35	14	1/1	1	1	2	2/1

7.2 PIN-PUERTOS Y PERIFÉRICOS

7.2.1 DESCRIPCIÓN DE PINES (PINOUT)

La configuración de los pines depende del tipo de empaque que estemos describiendo, así, tenemos varios tipos de empaque (Tabla 7.3 y Figura 7.1), como: PDIP, SOIC, SSOP, QFN, TQFP.

Tabla 7.3 Empaques PIC16F887 y grupo PIC16F88X.

PIC16F88X	PIN				
PIC16F882	28	PDIP	SOIC	SSOP	QFN
PIC16F883	28	PDIP	SOIC	SSOP	QFN
PIC16F884	40/44	PDIP		TQFP	QFN
PIC16F886	28	PDIP	SOIC	SSOP	QFN
PIC16F887	40/44	PDIP		TQFP	QFN



Figura 7.1 Diferentes tipos de Package.

Para nuestro caso, el *pinout* de 40 pines **PDIP** (*Plastic Dual In Package*) (Figura 7.2), usado comúnmente para **protoboard**, o con base de 100 mils de separación entre pines y 600 mils de ancho; este empaque y la disposición de pines son los que trabajaremos y describiremos a lo largo de este libro.

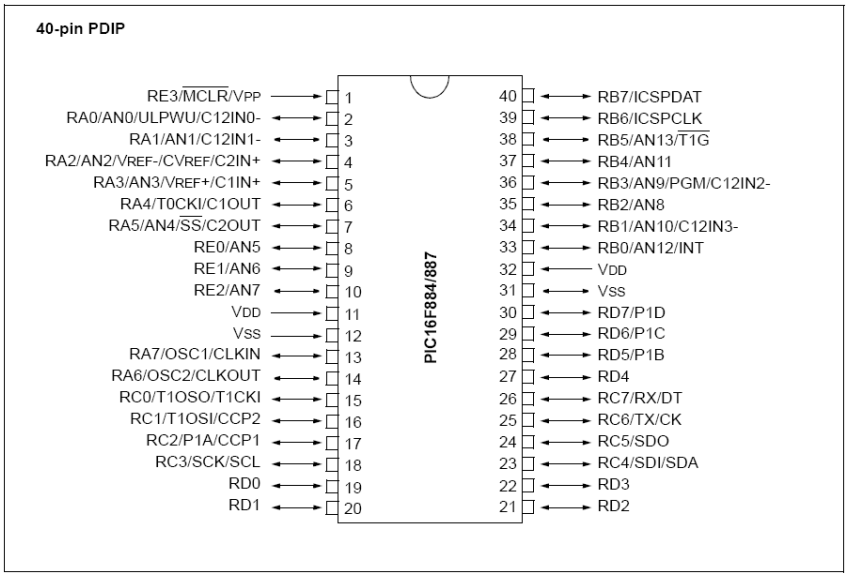


Figura 7.2 Pinout de 40 pines PDIP (Plastic Dual In Package).

7.2.2 PUERTOS Y PERIFÉRICOS

Los **puertos** pueden o no compartir el pin con algún **periférico**; a veces con más de un periférico, la función con la que el pin queda trabajando depende de la programación que realiza el usuario. Para la configuración de cada periférico se deben usar adecuadamente los registros asociados y un procedimiento secuencial; detallamos esto en cada módulo más adelante. A continuación (Tabla 7.4), veremos un resumen de los periféricos que contiene el **PIC16F887** y cada dispositivo del grupo **PIC16F88X**.

Tabla 7.4 Puertos y periféricos PIC16F887 y grupo PIC16F887.

Key Features	PIC16F883	PIC16F884	PIC16F886	PIC16F887	PIC16F882
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C
Timers	3	3	3	3	3
Capture/Compare/PWM modules	2	2	2	2	2
Serial Communications	MSSP, EUSART	MSSP, EUSART	MSSP, EUSART	MSSP, EUSART	MSSP, EUSART
10-bit Analog-to-Digital Module	11 input channels	14 input channels	11 input channels	14 input channels	11 input channels
Analog Comparators	2	2	2	2	2

Resumiendo, todos tienen:

- 3 Timers, TMR0, TMR1, TMR2
- 2 Capture/Compare PWM
- 2 Comparadores Análogos
- Comunicación MSSP, EUSART

Las diferencias están en:

—28 o 40/44 pines

—11 o 14 conversores análogo a digital de 10 bits

—3 o 5 puertos I/O

En la Tabla 7.5 se encuentran descritos detalladamente los pines y puertos con periféricos compartidos para el microcontrolador **PIC16F887**. Cada módulo pertenece a un periférico que realiza una función específica y puede contener más de un puerto. Conviene anotar que se debe escoger solamente una función o un módulo periférico por cada pin; no aconsejamos usar varias funciones sobre el mismo pin; esto únicamente para desarrolladores expertos. Debido a que se puede comprometer al MCU y podría dañar sus pines en casos en los que se pueda confundir una salida con una entrada (un error ocasional es programar un pin al que se conecta un **interruptor** [*switch*] como salida; al presionarlo si la salida está en **alto** se produce un cortocircuito a **GND** que puede dañar el *mosfet* del circuito de salida del puerto).

Tabla 7.5 Descripción de pines y periféricos PIC16F887.

PINES PUERTOS PERIFERICOS PIC16F877A						
PIN	NOMBRE PIN	PUERTO	PERIFERICOS			NOTAS
2	RA0	RA0/AN0/ULPWU/C12IN0-	AN0	ULPWU	C12IN0-	
3	RA1	RA1/AN1/C12IN1-	AN1		C12IN1-	
4	RA2	RA2/AN2/VREF-/CVREF/C2IN+	AN2	CVREF	VREF- / C2IN+	
5	RA3	RA3/AN3/VREF+/C1IN+	AN3		VREF+ / C1IN+	
6	RA4	RA4/T0CKI/C1OUT	T0CKI		C1OUT	
7	RA5	RA5/AN4/SS/C2OUT	AN4	-SS	C2OUT	
33	RB0	RB0/AN12/INT	INT	AN12		
34	RB1	RB1/AN10/C12IN3-		AN10	C12IN3-	
35	RB2	RB2/AN8		AN8		
36	RB3	RB3/AN9/PGM/C12IN2-	PGM	AN9	C12IN2-	
37	RB4	RB4/AN11		AN11		
38	RB5	RB5/AN13/T1G		AN13	T1G	
39	RB6	RB6/ICSPCLK	PGC			
40	RB7	RB7/ICSPDAT	PGD			
15	RC0	RC0/T1OSO/T1CKI	T1CKI	T1OSO		
16	RC1	RC1/T1OSI/CCP2	CCP2	T1OSI		
17	RC2	RC2/P1A/CCP1	CCP1		P1A	
18	RC3	RC3/SCK/SCL	SCL	SCK		
23	RC4	RC4/SDI/SDA	SDA	SDI		
24	RC5	RC5/SDO		SDO		
25	RC6	RC6/TX/CK	TX	CK		
26	RC7	RC7/RX/DT	RX	DT		
19	RD0	RD0				
20	RD1	RD1				
21	RD2	RD2				
22	RD3	RD3				
27	RD4	RD4				
28	RD5	RD5/P1B			P1B	
29	RD6	RD6/P1C			P1C	
30	RD7	RD7/P1D			P1D	
8	RE0	RE0/AN5		AN5		
9	RE1	RE1/AN6		AN6		
10	RE2	RE2/AN7		AN7		
1		RE3/MCLR/VPP	-MCLR			
11		VDD	VDD			5V
12		VSS	VSS			GND
13		RA7/OSC1/CLKIN	OSC1		CLKIN	
14		RA6/OSC2/CLKOUT	OSC2		CLKOUT	
31		VDD	VDD			5V
32		VSS	VSS			GND

MODULOS	DESCRIPCION
SPECIAL	Funciones Especiales MCU
SCI	Serial Comunication Interface
SPI	Serial Protocol Interface
PROG	Programación
AD	Analogo a Digital
SCI-1	Serial Comunication Interface 1
PWM	Capture Compare PWM
I2C	I2C
TIMER	Temporizadores
COMP	Comparadores

7.2.3 DIAGRAMA DE BLOQUES

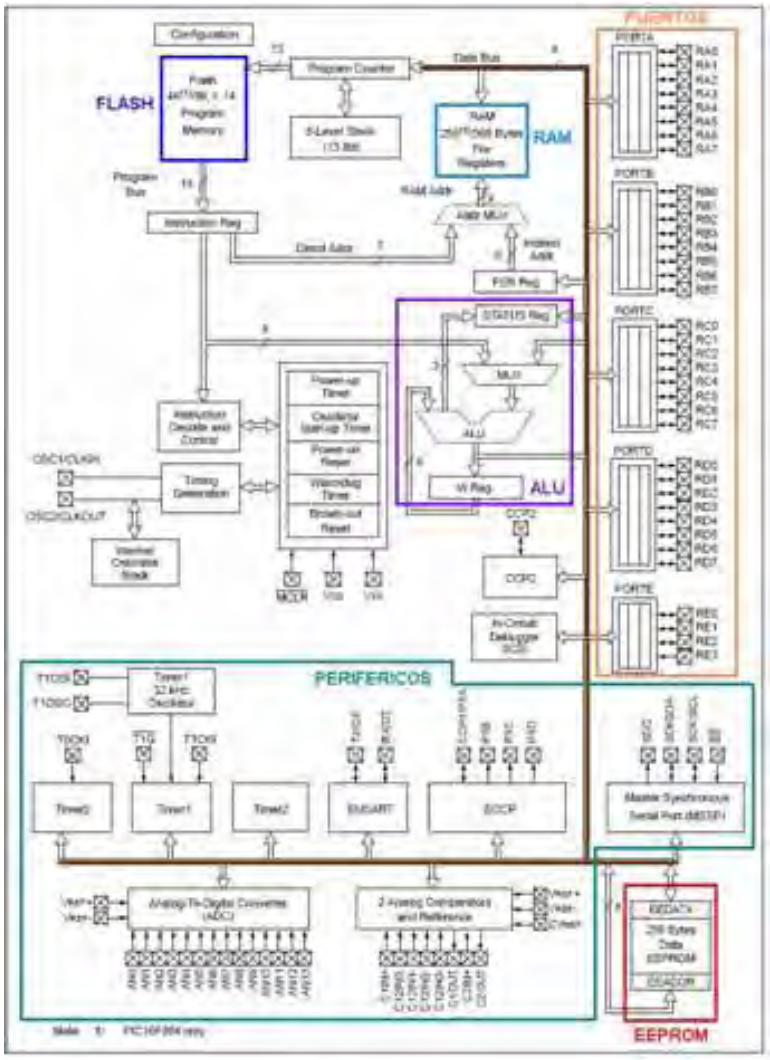


Figura 7.3 Diagrama de bloques PIC 16F884/PIC16F887.

7.3 MEMORIA

Hay tres tipos de memoria en los PIC micros MCU:

- Memoria de programa (*program*: *memory*) –instrucciones.
- Memoria de datos (*data memory*) –valores de datos variables.
  - Registros de funciones especiales (**SFR**: *Special Function Register*), controlan la operación del dispositivo.
  - Registros de Propósito General (**GPR**: *General Purpose Registers*), almacenamiento de RAM.
- Memoria de EEPROM –almacenamiento de datos no volátil.

A continuación veremos cada una de estas partes de la memoria.

7.3.1 MEMORIA DE PROGRAMA Y STACK

En la memoria del programa se encuentra alojado el programa con el que el MCU funcionará. En la Figura 7.4 apreciamos el mapa de memoria de programa del contador de programa PC de 13 bits (12:0), que tiene la posibilidad de direccionamiento de **2^{13} = 8 192 posiciones => 0 – 1FFFh**. De tal manera que el contador de programa PC puede recorrer todo el espacio del mapa de la memoria de programa.

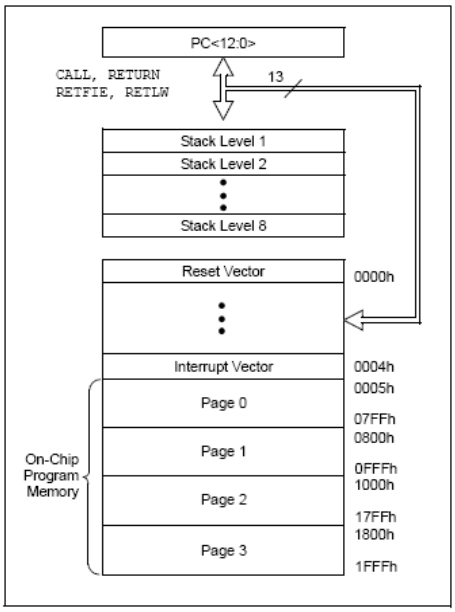


Figura 7.4
Memoria de programa y Stack.

El área de programas está dividida en **cuatro páginas** (**page 0:3**), cada una de 2 048 posiciones (2 k words), con un total de 8 192 posiciones (8 k words). Ésta es la cantidad de instrucciones que como máximo debe tener el programa para que se pueda alojar en la memoria del MCU.

NOTA: Las posiciones de memoria de programa son de un ancho de 14 bits por palabra (word) (bit 0 a 13), las llamamos palabras porque no podemos llamarlas byte (byte = 8 bits).

Las páginas o los bloques de memoria de programas (Figura 7.5) se direccionan trabajando con el registro PC-LATH (4:3), así:

- PC-LATH (4:3) = 00 Page 0, 0000 – 07FFh
- PC-LATH (4:3) = 01 Page 1, 0800 – 0FFFh
- PC-LATH (4:3) = 10 Page 2, 1000 – 17FFh
- PC-LATH (4:3) = 11 Page 3, 1800 – 1FFFh

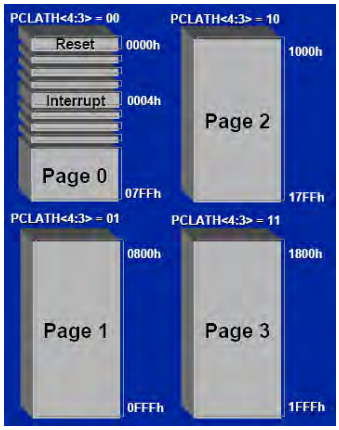


Figura 7.5 Memoria de programa (pages).

Se debe tener esto presente cuando nuestro programa se hace muy extenso, y hay que llamar a subrutinas que quedan en otras páginas. El *stack* con 8 niveles nos indica que podemos usar hasta un máximo de 8 niveles de llamados a subrutinas anidados, como observamos en las Figuras 7.6 y 7.7.

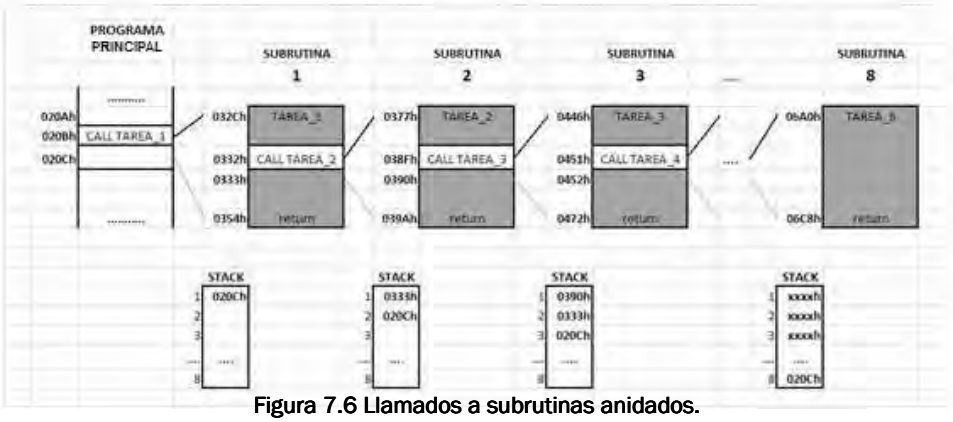


Figura 7.6 Llamados a subrutinas anidados.

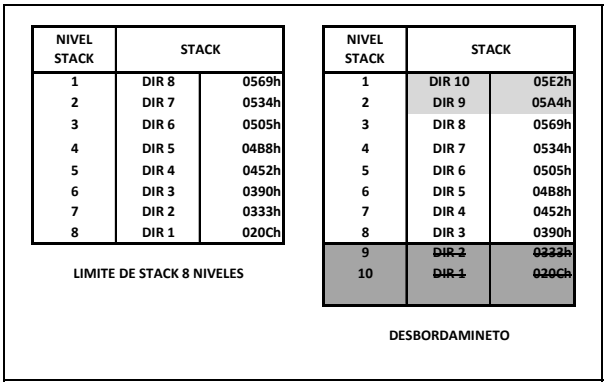


Figura 7.7 Llamados a rutinas anidados.

El vector de *reset* es la posición 0000h y el vector de Interrupción es el 0004h; esto significa que al ocurrir estos eventos, el de *reset* o el de **interrupción**, el contador de programa se direcciona a estas posiciones, y desde allí podemos direccionar a la posición de programa en que estén alojadas nuestras rutinas de *reset* y la de servicio de Interrupción.

7.3.2 MEMORIA DE DATOS

La memoria de datos es donde se almacenan todos los registros propios del MCU y los registros propios del usuario. De esta forma se divide en dos partes, a saber:

- **SFR** (*Special Function Registers*), Registros de Funciones Especiales; éstos son los registros propios del MCU asociados con los periféricos y/o puertos y el funcionamiento en general del MCU.

  Estos registros guardan las configuraciones de periféricos, la lectura y escritura de valores en el periférico o el puerto; también la generación de banderas (*flags),* habilitación de interrupciones, *buffers* de entrada y salida de datos; en fin, todos los registros necesarios para el funcionamiento de todos los módulos del MCU.

- **GPR** (*General Purpose Registers*), Registros de Propósito General; éstos son los registros en donde se puede almacenar la información del programa del usuario.

Hay que tener en cuenta que la memoria de datos está dividida en cuatro bancos que se deben direccionar previamente para acceder el registro. Esto se logra con los bits RP1 y RP0 del registro **STATUS<6:5>**, de la manera como muestran la Figura 7.8 y la Tabla 7.6.

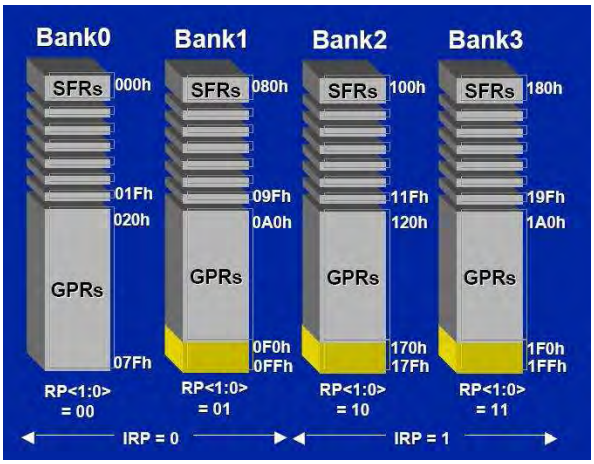


Figura 7.8 Memoria de datos.

Tabla 7.6 Direccionamiento de bancos y posiciones

STATUS <6:5> RP1:RP0	BANCO	POSICIONES
00	0	000 - 07Fh
01	1	080 - 0FFh
10	2	100 - 17Fh
11	3	180 - 1FFh

En seguida veremos el área de memoria de datos, con ambos tipos de registros SFR y GPR, con sus posiciones respectivas.

7.3.3 REGISTROS DE FUNCIONES ESPECIALES

En la Tabla 7.7 se muestra un ejemplo.

Tabla 7.7 Mapa de registros.

Mapa de Registros PIC16F887

File	Addr	File	Addr	File	Addr	File	Addr
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h	WDTCON	105h	SRCON	185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	CM1CON0	107h	BAUDCTL	187h
PORTD	08h	TRISD	88h	CM2CON0	108h	ANSEL	188h
PORTE	09h	TRISE	89h	CM2CON1	109h	ANSELH	189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDAT	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh		18Eh
TMR1H	0Fh	OSCCON	8Fh	EEADRH	10Fh		18Fh
T1CON	10h	OSCTUNE	90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h	WPUB	95h		115h		195h
CCPR1H	16h	IOCB	96h	General	116h	General	196h
CCP1CON	17h	VRCON	97h	Purpose	117h	Purpose	197h
RCSTA	18h	TXSTA	98h	Register (1)	118h	Register (1)	198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah	SPBRGH	9Ah		11Ah		19Ah
CCPR2L	1Bh	PWM1CON	9Bh		11Bh		19Bh
CCPR2H	1Ch	ECCPAS	9Ch		11Ch		19Ch
CCP2CON	1Dh	PSTRCON	9Dh		11Dh		19Dh
ADRESH	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers (2) 96 Bytes	20h 7Fh	General Purpose Registers (3) 80 Bytes	A0h EFh	General Purpose Registers (3) 80 Bytes	120h 16Fh	General Purpose Registers (3) 80 Bytes	1A0h 1EFh
		Mapped in Bank0 70h - 7Fh (4)	F0h FFh	Mapped in Bank0 70h - 7Fh (4)	170h 17Fh	Mapped in Bank0 70h - 7Fh (4)	1F0h 1FFh

Bank0		Bank1		Bank2		Bank3	
[RP1:RP0]	00	[RP1:RP0]	01	[RP1:RP0]	10	[RP1:RP0]	11
[IRP] 0				[IRP] 1			

Observamos que algunos **SFR** ocupan la misma posición en cada uno de los bancos, registros como **PCL(02h), STATUS(03h), FSR(04h), PCLTAH(0Ah), INTCON(0Bh)**. Esto quiere decir que hay solamente un registro (por ejemplo, STATUS) que comparte cuatro direcciones lógicas. Esto se hace para facilitar la programación en el acceso a estos registros, sin tener que conmutar bancos y, por tanto, conservando el banco en el que estamos trabajando. De la misma manera podemos encontrar el área en los **GPR** en las posiciones 70 a 7F, las cuales siempre serán accedidas desde cualquier banco, lo que indica que sólo hay hasta **0Fh (16d)** posiciones, que se comparten en todos los bancos. En la Tabla 7.8 vemos las posiciones de memoria y su cantidad para hacer un total de memoria GPR, o de usuario de 368 bytes.

Tabla 7.8 Registros de usuario (GPR) y posiciones.

CANTIDAD DE GPR (RAM)	
POSICIONES	N REG.
020 - 07Fh	96
0A0 - 0EFh	80
110 - 16Fh	96
190 - 1EFh	96
TOTAL	368

A continuación, en la Tabla 7.9 encontramos un resumen de los **Registros de Funciones Especiales (SFR)**, en esta tabla se detallan: el banco al que pertenecen, la dirección del registro **xxxh**, el nombre del registro, los bits contenidos, el valor sobre un *Power-On Reset* y sobre un *Brown-Out Reset*; además, la página en donde se describe en la *data sheet* DS39582B PIC16F887. Este archivo pdf puede encontrarlo en la Web en la carpeta *data sheet.* Estos registros se explican de manera amplia en cada uno de los módulos periféricos; algunos de ellos se describen en el **Capítulo 8. Registros comunes**.

Tabla 7.9 Resumen de Registros de Funciones Especiales grupo PIC16F88X

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,217
01h	TMR0	Timer0 Module Register								xxxx xxxx	73,217
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,217
03h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,217
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,217
05h	PORTA(3)	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx xxxx	39,217
06h	PORTB(3)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,217
07h	PORTC(3)	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	53,217
08h	PORTD(3,4)	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	57,217
09h	PORTE(3)	—	—	—	—	RE3	RE2(4)	RE1(4)	RE0(4)	---- xxxx	59,217
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter					---0 0000	37,217
0Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,217
0Ch	PIR1	—	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	-000 0000	34,217
0Dh	PIR2	OSFIF	C2IF	C1IF	EEIF	BCLIF	ULPWUIF	—	CCP2IF	0000 00-0	35,217
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	76,217
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	76,217
10h	T1CON	T1GINV	TMR1GE	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON	0000 0000	79,217
11h	TMR2	Timer2 Module Register								0000 0000	81,217
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	82,217
13h	SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	183,217
14h	SSPCON(2)	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	181,217
15h	CCPR1L	Capture/Compare/PWM Register 1 Low Byte (LSB)								xxxx xxxx	126,217
16h	CCPR1H	Capture/Compare/PWM Register 1 High Byte (MSB)								xxxx xxxx	126,217
17h	CCP1CON	P1M1	P1M0	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0	0000 0000	124,217
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	161,217
19h	TXREG	EUSART Transmit Data Register								0000 0000	153,217
1Ah	RCREG	EUSART Receive Data Register								0000 0000	158,217
1Bh	CCPR2L	Capture/Compare/PWM Register 2 Low Byte (LSB)								xxxx xxxx	126,217
1Ch	CCPR2H	Capture/Compare/PWM Register 2 High Byte (MSB)								xxxx xxxx	126,218
1Dh	CCP2CON	—	—	DC2B1	DC2B0	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	125,218
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	99,218
1Fh	ADCON0	ADCS1	ADCS0	CHS3	CHS2	CHS1	CHS0	$\overline{GO/DONE}$	ADON	0000 0000	104,218

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: $\overline{MCLR}$ and WDT Reset do not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.

2: When SSPCON register bits SSPM<3:0> = 1001, any reads or writes to the SSPADD SFR address are accessed through the SSPMSK register. See Registers • and 13-4 for more detail.

3: Port pins with analog functions controlled by the ANSEL and ANSELH registers will read '0' immediately after a Reset even though the data latches are either undefined (POR) or unchanged (other Resets).

4: PIC16F884/PIC16F887 only.

(continúa)

Tabla 7.9 Resumen de Registros de Funciones Especiales grupo PIC16F88X

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 1											
80h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,217
81h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,218
82h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,217
83h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,217
84h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,217
85h	TRISA	TRISA7	TRISA6	TRISA5	TRISA4	TRISA3	TRISA2	TRISA1	TRISA0	1111 1111	39,218
86h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,218
87h	TRISC	TRISC7	TRISC6	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0	1111 1111	53,218
88h	TRISD(3)	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	1111 1111	57,218
89h	TRISE	—	—	—	—	TRISE3	TRISE2(3)	TRISE1(3)	TRISE0(3)	---- 1111	59,218
8Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,217
8Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,217
8Ch	PIE1	—	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	-000 0000	32,218
8Dh	PIE2	OSFIE	C2IE	C1IE	EEIE	BCLIE	ULPWUIE	—	CCP2IE	0000 00-0	33,218
8Eh	PCON	—	—	ULPWUE	SBOREN	—	—	$\overline{POR}$	$\overline{BOR}$	--01 --qq	36,218
8Fh	OSCCON	—	IRCF2	IRCF1	IRCF0	OSTS	HTS	LTS	SCS	-110 q000	62,218
90h	OSCTUNE	—	—	—	TUN4	TUN3	TUN2	TUN1	TUN0	---0 0000	66,218
91h	SSPCON2	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	0000 0000	181,218
92h	PR2	Timer2 Period Register								1111 1111	81,218
93h	SSPADD(2)	Synchronous Serial Port (I^2C mode) Address Register								0000 0000	189,218
93h	SSPMSK(2)	MSK7	MSK6	MSK5	MSK4	MSK3	MSK2	MSK1	MSK0	1111 1111	189,218
94h	SSPSTAT	SMP	CKE	$\overline{D/A}$	P	S	$\overline{R/W}$	UA	BF	0000 0000	189,218
95h	WPUB	WPUB7	WPUB6	WPUB5	WPUB4	WPUB3	WPUB2	WPUB1	WPUB0	1111 1111	49,218
96h	IOCB	IOCB7	IOCB6	IOCB5	IOCB4	IOCB3	IOCB2	IOCB1	IOCB0	0000 0000	49,218
97h	VRCON	VREN	VROE	VRR	VRSS	VR3	VR2	VR1	VR0	0000 0000	97,218
98h	TXSTA	CSRC	TX9	TXEN	SYNC	SENDB	BRGH	TRMT	TX9D	0000 0010	160,218
99h	SPBRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0	0000 0000	163,218
9Ah	SPBRGH	BRG15	BRG14	BRG13	BRG12	BRG11	BRG10	BRG9	BRG8	0000 0000	163,218
9Bh	PWM1CON	PRSEN	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0	0000 0000	145,218
9Ch	ECCPAS	ECCPASE	ECCPAS2	ECCPAS1	ECCPAS0	PSSAC1	PSSAC0	PSSBD1	PSSBD0	0000 0000	142,218
9Dh	PSTRCON	—	—	—	STRSYNC	STRD	STRC	STRB	STRA	---0 0001	146,218
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	99,218
9Fh	ADCON1	ADFM	—	VCFG1	VCFG0	—	—	—	—	0-00 ----	105,218

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented

Note 1: $\overline{MCLR}$ and WDT Reset do not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.

2: Accessible only when SSPCON register bits SSPM<3:0> = 1001.

3: PIC16F884/PIC16F887 only.

(continúa)

Tabla 7.9 Resumen de Registros de Funciones Especiales grupo PIC16F88X.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 2											
100h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,217
101h	TMR0	Timer0 Module Register								xxxx xxxx	73,217
102h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,217
103h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,217
104h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,217
105h	WDTCON	—	—	—	WDTPS3	WDTPS2	WDTPS1	WDTPS0	SWDTEN	---0 1000	225,218
106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	48,217
107h	CM1CON0	C1ON	C1OUT	C1OE	C1POL	—	C1R	C1CH1	C1CH0	0000 -000	88,218
108h	CM2CON0	C2ON	C2OUT	C2OE	C2POL	—	C2R	C2CH1	C2CH0	0000 -000	89,218
109h	CM2CON1	MC1OUT	MC2OUT	C1RSEL	C2RSEL	—	—	T1GSS	C2SYNC	0000 --10	91,219
10Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,217
10Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,217
10Ch	EEDAT	EEDAT7	EEDAT6	EEDAT5	EEDAT4	EEDAT3	EEDAT2	EEDAT1	EEDAT0	0000 0000	112,219
10Dh	EEADR	EEADR7	EEADR6	EEADR5	EEADR4	EEADR3	EEADR2	EEADR1	EEADR0	0000 0000	112,219
10Eh	EEDATH	—	—	EEDATH5	EEDATH4	EEDATH3	EEDATH2	EEDATH1	EEDATH0	--00 0000	112,219
10Fh	EEADRH	—	—	—	EEADRH4(2)	EEADRH3	EEADRH2	EEADRH1	EEADRH0	---- 0000	112,219

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented
Note 1: $\overline{MCLR}$ and WDT Reset does not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.
2: PIC16F886/PIC16F887 only.

(continúa)

Tabla 7.9 Resumen de Registros de Funciones Especiales grupo PIC16F88X.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Page
Bank 3											
180h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	37,217
181h	OPTION_REG	$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	30,218
182h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	37,217
183h	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	29,217
184h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	37,217
185h	SRCON	SR1	SR0	C1SEN	C2REN	PULSS	PULSR	—	FVREN	0000 00-0	93,219
186h	TRISB	TRISB7	TRISB6	TRISB5	TRISB4	TRISB3	TRISB2	TRISB1	TRISB0	1111 1111	48,218
187h	BAUDCTL	ABDOVF	RCIDL	—	SCKP	BRG16	—	WUE	ABDEN	01-0 0-00	162,219
188h	ANSEL	ANS7(2)	ANS6(2)	ANS5(2)	ANS4	ANS3	ANS2	ANS1	ANS0	1111 1111	40,219
189h	ANSELH	—	—	ANS13	ANS12	ANS11	ANS10	ANS9	ANS8	--11 1111	99,219
18Ah	PCLATH	—	—	—	Write Buffer for the upper 5 bits of the Program Counter					---0 0000	37,217
18Bh	INTCON	GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF(1)	0000 000x	31,217
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	113,219
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	111,219

Legend: – = Unimplemented locations read as '0', u = unchanged, x = unknown, q = value depends on condition, shaded = unimplemented
Note 1: $\overline{MCLR}$ and WDT Reset does not affect the previous value data latch. The RBIF bit will be cleared upon Reset but will set again if the mismatch exists.
2: PIC16F884/PIC16F887 only.

7.3.4 MEMORIA EEPROM

La **EEPROM** es una **memoria de datos de usuario** que tiene la particularidad de no borrarse cuando falta la alimentación de voltaje. Por consiguiente, es muy útil en el momento de realizar tablas de datos que se van construyendo a medida que el programa va generando estos datos, de tal manera que se puedan guardar como parámetros de calibración o funcionamiento del dispositivo. A esta **EEPROM** (Figura 7.9) se accede mediante un procedimiento; no es posible acceder a los datos por un direccionamiento. Observemos que la ubicación de esta memoria está en las direcciones **2100h a 21FFh**, que son posiciones de memoria de programa, más allá de los lugares permitidos.

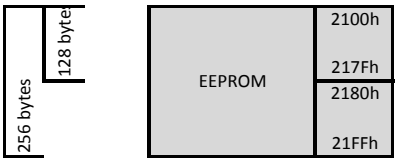


Figura 7.9 memoria EEPROM.

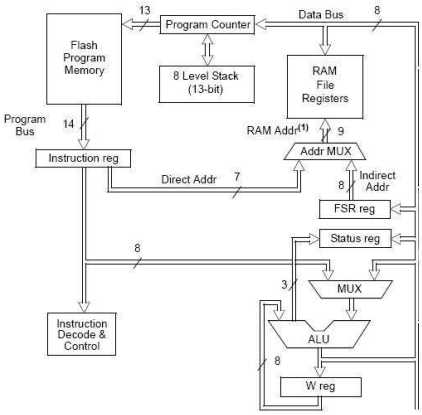
8.0. REGISTRO COMUNES

A continuación veremos los registros comunes a ambos **microcontroladores**, y los de uso más común.

8.1 REGISTRO W Y STATUS

El **registro W** (Figura 8.1) es el que en otros MCU podríamos llamar el **acumulador**, solamente que en este caso tiene la ventaja de la forma en que está alambrado con la ALU; permite tenerlo como dato de trabajo y a la vez como destino; esto quiere decir que al terminar una operación el resultado se guarda en el registro W.

Figura 8.1
Diagrama conexión ALU y registro W.



Las instrucciones que operan con registros y con el registro **W** afectan directamente el registro **STATUS** en los bits **Z, DC** y **C**. El **registro STATUS** contiene el estado aritmético de la **ALU**, el estado de *reset* y los bits para selección de bancos en la memoria de datos. Este registro es una gran ayuda en varios sentidos:

- Para saber el resultado de operaciones aritméticas con los bits Z,C y DC
- Para saber la causa del último reset
- Para seleccionar bancos de memoria IRP, RP1, RP0.

NOTA: El registro STATUS, como cualquier otro, puede ser el destino de cualquier instrucción, pero, al tratar de escribir los bits Z, DC, C, no se permitirá, pues éstos son afectados únicamente por la lógica de la instrucción, y al tratar de escribir los bits de *reset* $\overline{TO}$ y $\overline{PD}$ no se permite, porque sólo son de lectura.

En seguida (Tabla 8.1) se describe el registro STATUS con cada uno de los bits.

Tabla 8.1
Registro STATUS (ADDRES 03h, 83h, 103h, 183h)

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	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7	**IRP:** Register Bank Select bit (used for indirect addressing) 1 = Bank 2, 3 (100h-1FFh) 0 = Bank 0, 1 (00h-FFh)
bit 6-5	**RP1:RP0**: Register Bank Select bits (used for direct addressing) 11 = Bank 3 (180h-1FFh) 10 = Bank 2 (100h-17Fh) 01 = Bank 1 (80h-FFh) 00 = Bank 0 (00h-7Fh) Each bank is 128 bytes.
bit 4	$\overline{TO}$: Time-out bit 1 = After power-up, CLRWDT instruction or SLEEP instruction 0 = A WDT time-out occurred
bit 3	$\overline{PD}$: Power-down bit 1 = After power-up or by the CLRWDT instruction 0 = By execution of the SLEEP instruction
bit 2	Z: Zero bit 1 = The result of an arithmetic or logic operation is zero 0 = The result of an arithmetic or logic operation is not zero
bit 1	DC: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) (for $\overline{borrow}$, the polarity is reversed) 1 = A carry-out from the 4th low order bit of the result occurred 0 = No carry-out from the 4th low order bit of the result
bit 0	C: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions) 1 = A carry-out from the Most Significant bit of the result occurred 0 = No carry-out from the Most Significant bit of the result occurred
Note:	For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (RRF, RLF) instructions, this bit is loaded with either the high, or low order bit of the source register.

Legend:		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared x = Bit is unknown

8.2 CONTADOR DEL PROGRAMA PCL Y PCLATH

El **contador de programa** (Figura 8.2) es un registro muy importante, pues indica en qué posición de la memoria se está ejecutando la presente instrucción. Este registro es un contador que recorre todo el programa en la secuencia determinada por el usuario incluso con sus llamados a subrutinas.

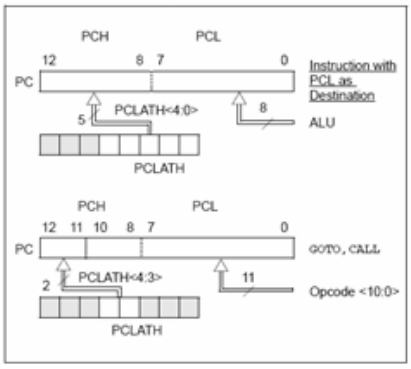


Figura 8.2 Diagrama de PCL y PCLATH.a subrutinas CALL y saltos GOTO.

- El registro PC es de 13 bits PC<12:0>, contiene dos bytes PCH y PCL.
- El byte PCL es un registro **leíble/escribible.**
- El byte PCH es un registro **no leíble**, **sí escribible** indirectamente a través de PCLATH.

```
NOTA: Sobre cualquier reset PCH=0.
```

Tiene dos modos de funcionamiento:

- En instrucciones con PCL como destino, donde la operación de la ALU da la dirección del byte PCL de menor peso y PCLATH escribe su dirección en el registro PCH de mayor peso. Antes de esta instrucción debe ponerse el valor de dirección alta en PCLATH.
- En instrucciones GOTO y CALL donde la dirección PC<10:0> se obtiene del programa (OPCODE), y los bits de página se copian de PCLATH<4:3>.

Estos registros se utilizan mucho para el llamado a subrutinas **CALL** y saltos de programa GOTO. Éstos deben utilizarse con gran cuidado cuando se hagan llamadas entre diferentes páginas (*page*) (límite de página 2 048 palabras). Se

acostumbra mucho realizar tablas con esta combinación que permite el "goto computado", es decir, un apuntador a una tabla con la adición de un *offset*, lo que permite esta facilidad utilizando el PCL. En el ejemplo siguiente leemos un dato de una tabla y lo ponemos en el PUERTO B. El registro **_offset** es un valor que nos dice la posición del dato en la tabla.

```
Cuando _offset=0      PORTB=W="h"
Cuando _offset=3      PORTB=W="a"
        movlw   offset          ; cargue offset para la posición en la Tabla
        call    Valor_Tabla     ; trae el dato en la posición offset de la Tabla
        movwf   PORTB           ; pone el dato W en PUERTO B
Valor_Tabla
        addwf   PCL,F           ; sume offset a PCL, PCL a offset de la Tabla
        retlw   "h"             ; offset 0, carácter h
        retlw   "o"             ; offset 1, carácter o
        retlw   "l"             ; offset 2, carácter l
        retlw   "a"             ; offset 3, carácter a
```

```
NOTA:

Puede ocurrir el error de pasar el límite de 2 k words = 2 048 =
7FFh instrucciones de programa y tener problemas con los llamados a
subrutinas.

Este problema se presenta por pasar de la Página 0 (Page0) a la
Página 1 (Page1), y por no tener en consideración los bits de PC-
LATH<4:3> al intercambiar páginas.
```

Más ampliamente, con ejemplos veremos este manejo del contador de programa PC, en el **Capítulo 11.0, Sección 11.5.3: Datos en tipos de memoria**.

8.3 FSR (FILE SELECT REGISTER), E INDF (INDIRECT FILE)

Este par de registros trabajan juntos para facilitar el manejo del direccionamiento indirecto de la RAM (Figura 8.3); el **FSR** actúa como el apuntador a una posición de la RAM y se opera a través de INDF.

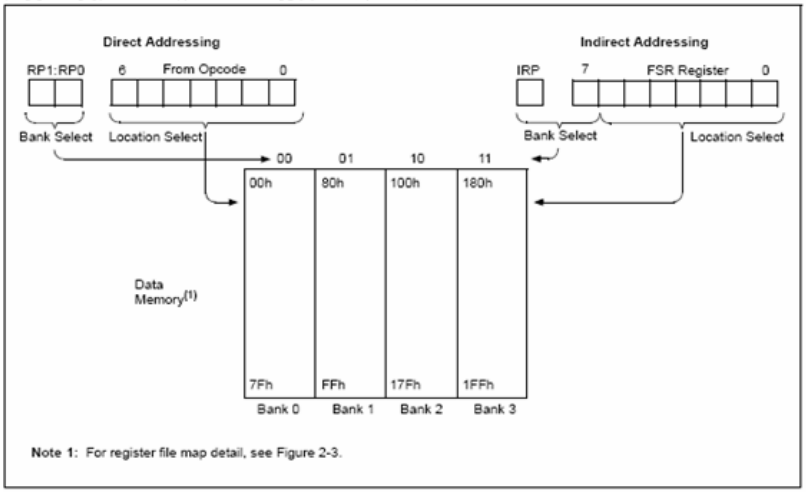


Figura 8.3 Direccionamiento directo e indirecto.

En la Figura 8.3 observamos cómo el cambio de bancos opera de manera diferente:

- Para direccionamiento directo se habilita RP1:RP0.
- Para direccionamiento indirecto se habilita **IRP**, y el bit 7 de FSR.

El ejemplo siguiente ilustra cómo se usan los registros **FSR** e **IND** para realizar la tarea de borrar un bloque de memoria, desde la posición inicial del bloque 0X20, se borrará una cantidad de registros de **0X1F (31d)** guardados en la variable "cant_reg", que irá disminuyendo hasta llegar a 0.

```
        movlw   0×20        ; posición inicial bloque a borrar
        movwf   FSR         ; guarda posición inicial en apuntador FSR
        movlw   0×1F        ; carga cantidad de registros a borrar
        movwf   cant_reg    ; guárdelos en la variable
ciclo
        clrf    INDF        ; borre registro indirecto
        incf    FSR,F       ; incremente apuntador FSR
        decfsz  cant_reg    ; decremente y salte cuando sea 0
        goto    ciclo
        goto    $           ; haga ciclo aquí
```

8.4 REGISTRO OPTION_REG (OPTION REG)

El registro **Option_Reg** (Tabla 8.2) es **leíble/escribibl**e, contiene varios bits de control para configurar el *prescaler* del TMR0/*postscaler* **WDT** (*Watchdog Time*), también la fuente de reloj para el TMR0, la interrupción INT y el *pull-up* del PORTB.

Tabla 8.2 Registro Option Reg.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: When using Low-Voltage ICSP Programming (LVP) and the pull-ups on PORTB are enabled, bit 3 in the TRISB register must be cleared to disable the pull-up on RB3 and ensure the proper operation of the device

El *prescaler* solamente se puede asignar al TMR0 o al WDT, no a ambos de manera simultánea.

8.5 OTROS REGISTROS COMUNES

Otros registros comunes e importantes son:

- INTCON Register
- PIE1 Register
- PIR1 Register
- PIE2 Register
- PIR2 Register
- PCON Register

Todos estos registros se detallan en el **Apéndice E: Registros especiales**.

9.0 FUNCIONES ESPECIALES DE LA CPU

A continuación veremos otras características que maximizan el funcionamiento del MCU.

9.1 OSCILADOR

Este elemento es el responsable de los ciclos de reloj; de su precisión depende toda la precisión de todos los módulos internos. Se pueden escoger entre cuatro tipos de circuitos de oscilador, programando los dos bits <Fosc1:Fosc2> en el registro de configuración de bits.

FOSC1:FOSC0: Bits de selección de oscilador:

- 11 = RC oscilador, resistencia y condensador (muy baja precisión)
- 10 = HS oscilador, alta velocidad >4 MHz a 20 MHz, cristal/resonador (alta precisión)
- 01 = XT oscilador, <4 MHz a 200 kHz, cristal/resonador (mediana precisión)
- 00 = LP oscilador, 20-200 kHz cristal de baja potencia (mediana precisión)

Es más recomendable mientras pueda trabajar en alta frecuencia (20 MHz), para aprovechar las mejores características de la máquina. No obstante, cada usuario evalúa las condiciones de costo y precisión de su aplicación.

NOTA: Sin el circuito de cristal, el oscilador no arranca y tampoco arranca el MCU, a menos que contenga oscilador interno, y éste se halle configurado.

En las configuraciones de **HS, XT** o **LP** de la Figura 9.1 se conecta un cristal o un resonador cerámico. Otro circuito típico puede ser el de la derecha (Figura 9.2).

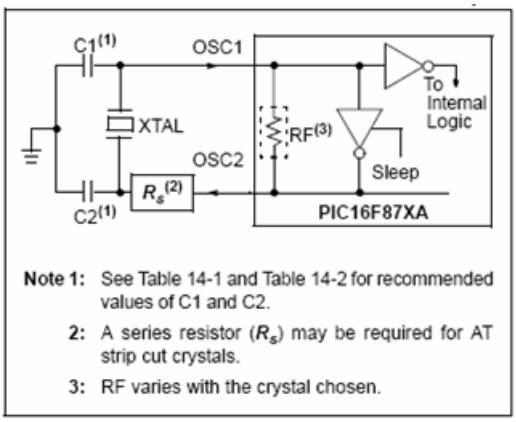


Figura 9.1 Circuito oscilador HS/XT o LP.

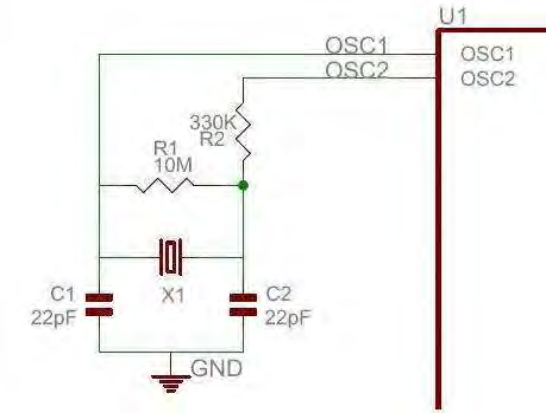


Figura 9.2 Circuito oscilador HS/XT o LP.

Como vemos en la Figura 9.3, también podemos conectar un *clock* externo, proveniente de otra fuente, la cual ya contiene la **circuitería** interna para su propio oscilador, al conectarse sólo requiere un pin (OSC1).

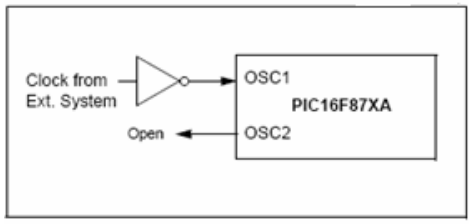


Figura 9.3 Oscilador externo.

Según el tipo de oscilador y de frecuencia, observamos el rango de los valores para los condensadores de acople del cristal.

Tabla 9.1 Selección de condensador para oscilador de cristal.

Osc Type	Crystal Freq.	Cap. Range C1	Cap. Range C2
LP	32 kHz	33 pF	33 pF
	200 kHz	15 pF	15 pF
XT	200 kHz	47-68 pF	47-68 pF
	1 MHz	15 pF	15 pF
	4 MHz	15 pF	15 pF
HS	4 MHz	15 pF	15 pF
	8 MHz	15-33 pF	15-33 pF
	20 MHz	15-33 pF	15-33 pF
These values are for design guidance only. See notes following this table.			
Crystals Used			
32 kHz	Epson C-001R32.768K-A	± 20 PPM	
200 kHz	STD XTL 200.000KHz	± 20 PPM	
1 MHz	ECS ECS-10-13-1	± 50 PPM	
4 MHz	ECS ECS-40-20-1	± 50 PPM	
8 MHz	EPSON CA-301 8.000M-C	± 30 PPM	
20 MHz	EPSON CA-301 20.000M-C	± 30 PPM	

El **circuito RC** (Figura 9.4) es el más bajo en precisión porque tiene muchos factores de variación, tales como voltaje de alimentación, precisión de los componentes y los valores de **R** y de **C** variables con la temperatura.

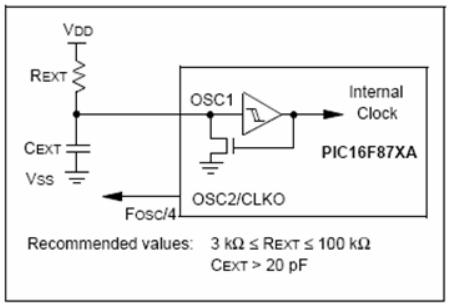


Figura 9.4 Oscilador de circuito RC.

Este circuito es recomendado para bajas frecuencias, y en donde no se necesita precisión.

9.2 RESET

El *reset* es una operación que ocurre por diferentes causas. El **PIC16F87XA** diferencia entre varios tipos de *reset* (dejamos los términos en inglés para reconocerlos más fácilmente), tales como:

- *Power-On Reset* (), al encender el MCU
- *reset*, durante operación normal
- *reset*, durante modo de *sleep*
- *WDT reset*, durante operación normal
- *WDT wake-up*, durante modo de *sleep*
- *Brown-Out Reset* (), falta o ausencia parcial de alimentación

Algunos registros no son afectados en ningún tipo de *reset*, su estado no es conocido durante un *Power-On Reset (POR)* y no es cambiado en ningún otro *reset*. Muchos otros registros son puestos a un estado predeterminado con un *POR*, con un *MCLR* y con un *WDT Reset*, con un *MCLR Reset* durante el modo de *Sleep* y *Brown-out Reset*. Ellos no son afectados por un *WDT wake-up* se entiende que es una reanudación de operación normal.

Los bit $\overline{TO}$ y $\overline{PD}$ son puestos a uno o puestos a cero (1 ó a 0) por diferentes situaciones de *reset*, son utilizados por el software para determinar la causa de la naturaleza del *reset* (Figura 9.5).

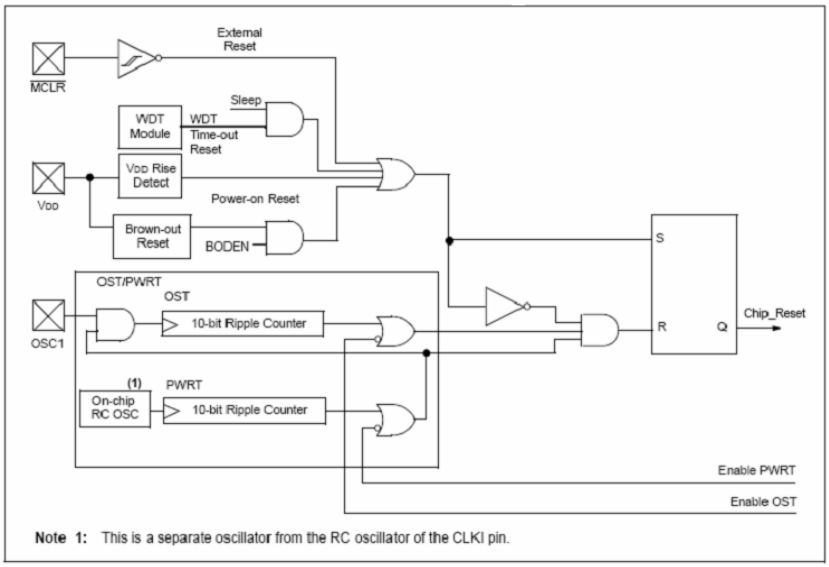


Figura 9.5 Diagrama simplificado de un circuito on-chip reset.

9.2.1 $\overline{\text{MCLR}}$

El circuito tiene un filtro de entrada para prevenir falsos pulsos de $\overline{\text{MCLR}}$. Por este pin también se puede programar con V_{pp} = 14 V_{cd}; por tanto, se recomienda proteger el voltaje de alimentación de V_{dd} = 5 V_{cd} del circuito de programación con una red, como se muestra en la Figura. 9.6

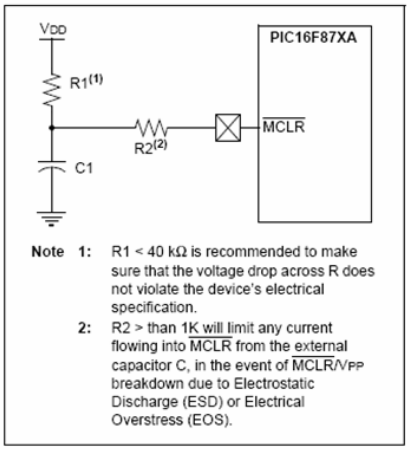


Figura 9.6 Circuito recomendado

9.2.2 POWER-ON RESET (POR)

El *power-on reset (POR)* es un pulso que se realiza cuando el voltaje de V_{DD} pasa de 1.2 a 1.7 V; éste es uno de los pasos para el encendido normal del MCU (saliendo de la condición de *reset*); los parámetros (voltaje, frecuencia, temperatura, etc.) se deben cumplir para garantizar la operación. El *Brown-out reset* puede ser usado para garantizar las condiciones de operación normal de encendido.

9.2.3 POWER –UP TIMER (PWRT)

El temporizador *Power-up Timer* ($\overline{PWRT}$) es de 72 ms y opera una vez haya ocurrido el *POR*; es un circuito RC que permite el tiempo al V_{DD} para llegar a su valor nominal. El chip se mantiene en *reset* tanto tiempo como el $\overline{PWRT}$ dure. Se puede habilitar o no en la configuración de tiempos. Es más recomendable dejarlo habilitado, se deshabilita en aplicaciones donde se necesite un encendido muy rápido del MCU.

9.2.4 OSCILLATOR START-UP TIMER (OST)

El temporizador de encendido $\overline{OST}$ del oscilador nos da un retardo de 1 024 ciclos del oscilador (entrando por OSC1), luego del *PWRT* . Este tiempo ayuda a asegurar que se estabilice el oscilador.

9.2.5 BROWN-OUT RESET (BOR)

El bit BOREN del circuito *Brown-out reset (* $\overline{BOR}$*)* puede ser habilitado o no en los bits de configuración; si V_{DD} baja de V_{BOR} = 4 V, por un tiempo más allá de T_{BOR} = 100 µs, ocurrirá una condición de $\overline{BOR}$ y se reseteará el micro. Se supone V_{DD} = 5 V como voltaje de trabajo, aunque el MCU puede trabajar con V_{dd} = 3 V. Si V_{DD} baja de V_{BOR} = 4 V pero por un periodo de tiempo menor que T_{BOR} = 100 µs, puede que no ocurra una condición de $\overline{BOR}$. Una vez que ocurre el *brown-out reset*, se queda en la condición de $\overline{BOR}$ hasta que V_{DD} suba por encima del valor de V_{BOR} = 4 V, el *power-up timer reset* (*PWRT)* mantiene el micro reseteado durante T_{PWRT} = 72 ms. Si V_{DD} cae abajo de V_{BOR}, durante T_{PWRT}, el proceso de *reset* puede reiniciarse cuando V_{DD} llegue encima de V_{BOR} con el *power-up timer reset* (*PWRT).*

9.2.6 SECUENCIA TIME-OUT

En un encendido la secuencia de *time-out* es como sigue:

- V_{DD} pasa de 1.2 a 1.7 y ocurre un *POR*
- Comienza el temporizador *PWRT* por un periodo de 72 ms (si está habilitado)

- Al terminar, comienza el temporizador de ***OST*** contando 1 024 ciclos del oscilador
- Al terminar, el micro sale de las condición de *reset*

Si se mantiene en bajo el tiempo suficiente (Figura 9.7), una vez el *time-out* termina, la condición de $\overline{\text{MCLR}}$ termina y comienza la ejecución de la primera instrucción en el vector de *reset*.

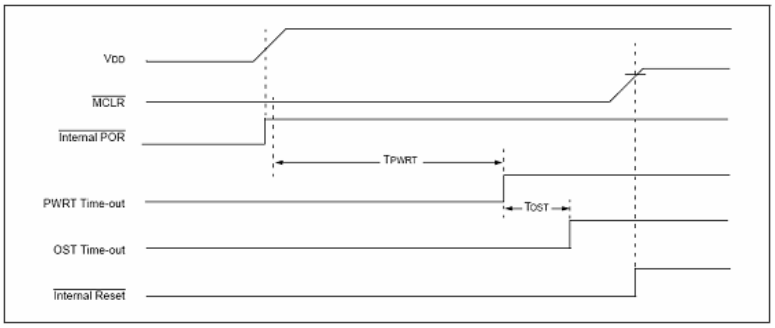


Figura 9.7 Secuencia Time-out en Power-on.

9.2.7 POWER CONTROL STATUS REGISTER (PCON)

El PCON tiene 2 bits:

- **Bit 0** (*Brown-Out Reset Status*). Su estado debido a un *power-on reset* no se conoce. Debe ser puesto a uno para verificarse en el *reset* siguiente, si se aclara, un $\overline{\text{BOR}}$ ha ocurrido. Cuando el *brown-out reset* es deshabilitado, su estado no es predecible y no es válido en ningún momento.

- **Bit 1** (*Power-On Reset Status Bit*). Es puesto a cero por un $\overline{\text{POR}}$ y no afectado de otra forma. El usuario debe poner a uno este bit seguido de un $\overline{\text{POR}}$.

A continuación, las Tablas 9.2, 9.3 y 9.4 nos ayudan a detallar valores y secuencias como característica del circuito de reset.

- Valor de *time-out* en diferentes situaciones y con distintos osciladores.

Tabla 9.2 Time-out en varias situaciones.

Oscillator Configuration	Power-up		Brown-out	Wake-up from Sleep
	$\overline{\text{PWRTE}}$ = 0	$\overline{\text{PWRTE}}$ = 1		
XT, HS, LP	72 ms + 1024 TOSC	1024 TOSC	72 ms + 1024 TOSC	1024 TOSC
RC	72 ms	—	72 ms	—

- Valores de los bits *status* y su significado, son importantes para saber cuál fue la causa del último *reset*.

Tabla 9.3 Bits de status y su significado.

$\overline{POR}$	$\overline{BOR}$	$\overline{TO}$	$\overline{PD}$	Condition
0	x	1	1	Power-on Reset
0	x	0	x	Illegal, $\overline{TO}$ is set on $\overline{POR}$
0	x	x	0	Illegal, $\overline{PD}$ is set on $\overline{POR}$
1	0	1	1	Brown-out Reset
1	1	0	1	WDT Reset
1	1	0	0	WDT Wake-up
1	1	u	u	$\overline{MCLR}$ Reset during normal operation
1	1	1	0	$\overline{MCLR}$ Reset during Sleep or Interrupt Wake-up from Sleep

Legend: x = don't care, u = unchanged

Tabla 9.4 Bits de status y su significado.

$\overline{POR}$	$\overline{BOR}$	$\overline{TO}$	$\overline{PD}$	Condition
0	x	1	1	Power-on Reset
0	x	0	x	Illegal, $\overline{TO}$ is set on $\overline{POR}$
0	x	x	0	Illegal, $\overline{PD}$ is set on $\overline{POR}$
1	0	1	1	Brown-out Reset
1	1	0	1	WDT Reset
1	1	0	0	WDT Wake-up
1	1	u	u	$\overline{MCLR}$ Reset during normal operation
1	1	1	0	$\overline{MCLR}$ Reset during Sleep or Interrupt Wake-up from Sleep

Legend: x = don't care, u = unchanged

La Tabla 9.5 es una lista de las condiciones de *reset* para algunos registros especiales; la Tabla 9.6 da los **valores *reset, watchdog*, OST, Power-out Timer y requerimientos Reset Brown-out.**

Tabla 9.5 Condiciones de reset para registros especiales.

Condition	Program Counter	Status Register	PCON Register
Power-on Reset	000h	0001 1xxx	---- --0x
$\overline{MCLR}$ Reset during normal operation	000h	000u uuuu	---- --uu
$\overline{MCLR}$ Reset during Sleep	000h	0001 0uuu	---- --uu
WDT Reset	000h	0000 1uuu	---- --uu
WDT Wake-up	PC + 1	uuu0 0uuu	---- --uu
Brown-out Reset	000h	0001 1uuu	---- --u0
Interrupt Wake-up from Sleep	PC + 1(1)	uuu1 0uuu	---- --uu

Legend: u = unchanged, x = unknown, - = unimplemented bit, read as '0'

Note 1: When the wake-up is due to an interrupt and the GIE bit is set, the PC is loaded with the interrupt vector (0004h).

Tabla 9.6 Valores de RESET, WATCHDOG, OST, PWRT, BOR.

Param No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
30	TMCL	$\overline{MCLR}$ Pulse Width (low)	2	—	—	µs	VDD = 5V, -40°C to +85°C
31*	TWDT	Watchdog Timer Time-out Period (no prescaler)	7	18	33	ms	VDD = 5V, -40°C to +85°C
32	TOST	Oscillation Start-up Timer Period	—	1024 TOSC	—	—	TOSC = OSC1 period
33*	TPWRT	Power-up Timer Period	28	72	132	ms	VDD = 5V, -40°C to +85°C
34	TIOZ	I/O High-Impedance from $\overline{MCLR}$ Low or Watchdog Timer Reset	—	—	2.1	µs	
35	TBOR	Brown-out Reset Pulse Width	100	—	—	µs	VDD ≤ VBOR (D005)

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

9.3 INTERRUPCIÓN

La **interrupción**, como su nombre lo indica, es la forma que tienen los periféricos de llamar la atención del micro para realizar una tarea urgente o prioritaria. Un ejemplo de la vida práctica que nos ilustra este proceso es: Cuando usted está escribiendo en su computadora y llaman a la puerta (interrupción), usted se debe levantar y atender el llamado de la puerta (rutina de servicio de interrupción).

- Otra forma poco eficiente sería que usted se levantara cada 5 minutos a ver si alguien toca a la puerta (*polling*).

De esta manera es más práctico trabajar con interrupciones que sólo llaman cuando un evento ocurre, en vez de *polling,* en donde tenemos que verificar si el evento ocurrió, esto gasta tiempo y recursos de la máquina. La familia **PIC16F87XA** tiene 15 fuentes de interrupción. Hay un **habilitador global GPIE** y un **habilitador de periféricos PEIE**, cada módulo periférico tiene 2 bits asociados, el de habilitación y el de bandera (por ejemplo, módulo TMR0):

- **TMR0IF** TMR0 *Interrupt Flag* (bandera de interrupción). Se genera (1) cuando ocurre un evento.
- **TMR0IE** TMR0 *Interrupt Enable*. Habilita la interrupción, se habilita con (1) y, si no está habilitada (0), no genera interrupción aunque haya bandera de interrupción.

9.3.1 INTERRUPCIONES ENMASCARABLE Y NO ENMASCARABLE

Como apreciamos en la Figura 9.8, existen interrupciones que son enmascarables y otras que no. Las no enmascarables TMR0/RB0/INT están contenidas en el registro INTCON, junto con los bits de interrupciones global bit GPIE y periféricos bit PEIE. Las otras interrupciones enmascarables (las habilita o no el bit PEIE) se encuentran

en los registros PIE1, PIE2 (Peripheral Interrupt Enable), habilitación de la interrupción, y en los registros PIR1, PIR2 (Peripheral Interrupt Flag: bandera de interrupción). Los registros **INTCON, PIE1, PIE2, PIR1, PIR2** están detallados en el **Apéndice E: Registros especiales.**

NOTA: Enmascarables léase también habilitables. Término proveniente de mask que se refiere a máscara; al poner una máscara se ve el valor de la máscara, no de lo que hay detrás de ella.

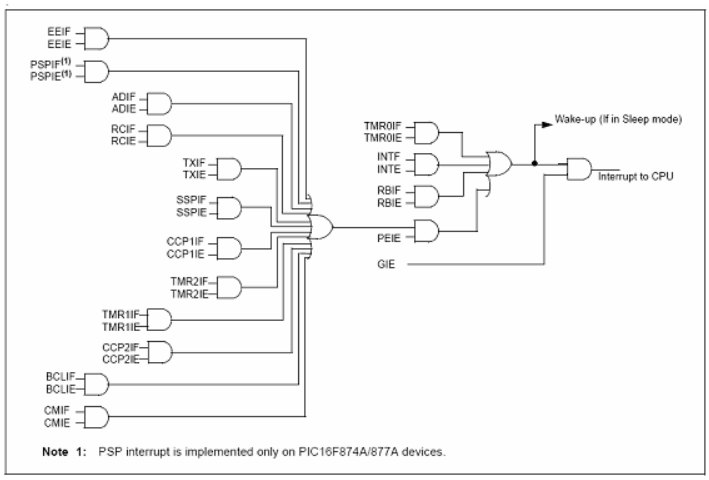


Figura 9.8 Lógica de la interrupción.

El bit **GIE** (*Global Interrupt Enable*), contenido en el registro **INTCON<7>**, habilita todas las interrupciones no enmascarables (puesto a uno, en "1"), o las deshabilita todas (puesto a cero, en "0"). El PEIE habilita o no todas las interrupciones de los periféricos.

NOTA: El GIE y el PIE son puestos a ceros durante un *reset*.

Cuando el GIE es habilitado y la bandera de interrupción y el bit de máscara son puestos a unos ("1"), el programa va al vector de interrupción, atendiendo la rutina de servicio de interrupción. Las interrupciones pueden ser individualmente habilitadas por sus correspondientes bits de habilitación y de bandera de interrupción, y deben habilitar su orden mayor de habilitación con el bit PEIE y el GIE. Como el circuito de la Figura 9.8 lo muestra:

- Sólo si el bit GIE=1, se habilita la interrupción

- Con GIE=1 se habilitan interrupciones de TMR0/INT/RB
- Con PEIE=1 se habilitan las demás interrupciones

```
NOTA: Los bits de banderas de interrupción individuales pueden
resetearse sin importar el estado de PEIE y GIE.
```

9.3.2 INSTRUCCIÓN RETFIE

La instrucción **Return From Interrupt** (**RETFIE**), retorna el programa al punto desde donde se realizó la interrupción cargando al contador de programa PC (con el último valor almacenado en el *stack,* el cual es el punto de origen de la interrupción, para continuar el programa desde allí.

9.3.3 PROCESO DE UNA INTERRUPCIÓN

El proceso de una interrupción se realiza como se detalla en la Figura 9.9. El programa transcurre libremente hasta que hay un llamado de una interrupción; esto detiene el contador de programa PC y es guardado en el stack, luego el PC se carga con la dirección 0004h (vector de interrupción), el bit GEIE es puesto a cero para deshabilitar cualquier interrupción adicional; luego se direcciona a una rutina que detecte el módulo que generó la interrupción, y de allí se direccionará a la rutina de servicio de interrupción (ISR) del módulo periférico; al terminar se debe apagar el bit de la bandera de interrupción antes de retornar de la interrupción; esto evita interrupciones recursivas; con la instrucción RETFIE se carga de nuevo el PC con la dirección guardada en el stack, y el programa continúa corriendo con libertad desde donde se interrumpió.

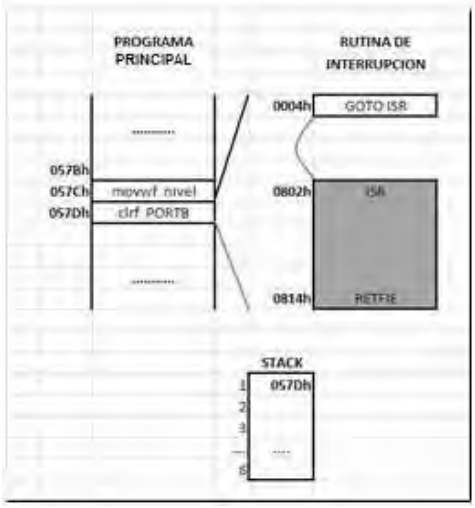


Figura 9.9
Flujo de interrupción

NOTA: **ISR** (Interrupt Service Rutine: Rutina de Servicio de Interrupción) es una porción de programa que hace una labor cuando la interrupción es generada. Debe ser lo más pequeña posible.

Cuando se presentan eventos de interrupción externa como INT o interrupción de cambio en PORTB, la latencia puede ser de 3 a 4 ciclos, dependiendo de cuándo la interrupción ocurrió durante las fases del ciclo de máquina del *clock* del MCU.

NOTA: Latencia es el tiempo que toma el MCU en detectar la interrupción.

9.3.4 INTERRUPCIONES NO ENMASCARABLES TMR0/PORTB/INT

9.3.4.1 INTERRUPCIÓN INT

La interrupción externa sobre el pin RB0/INT puede ser detectada por flanco de subida "1" o de caída "0", con el bit INTEDEG del registro **OPTION_REG<6>**. Cuando un borde válido aparece en el pin, se pone a uno el bit de bandera INTF en el registro INTCON<4>. La bandera INTF debe ser borrada después de ejecutar la rutina de servicio de interrupción, antes de habilitar de nuevo la interrupción. La interrupción INTF puede despertar el micro del modo de *sleep*. El estado de GPIE decide si luego de despertar el procesador salta o no a la rutina de servicio de interrupción.

9.3.4.2 INTERRUPCIÓN TMR0

Un **sobreflujo**, *overflow* (cambio de FFh a 00h) en el registro TMR0 (Figura 9.10), pone a uno el bit TMR0IF en el registro INTCON<2>. Éste puede tener el *prescaler* (entre 2 y 256) asignado con un valor escogido con PS2:PS0 en el OPTION_REG.

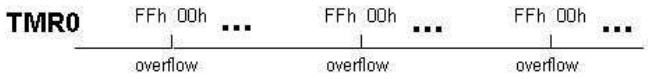


Figura 9.10 TMR0 overflow.

9.3.4.3 INTERRUPCIÓN POR CAMBIO EN PORTB

Esta interrupción llamada *interrupt-on-change* soporta solamente los 4 bits de mayor peso del puerto B (PORTB<7:4>), y actúa únicamente cuando los pines son configurados como entradas, compara el valor de la lectura actual con una lectura

guardada y su comparación genera la interrupción cuando no son iguales las lecturas, es decir, si hubo un cambio en los pines del PORTB. Esto pone a uno el bit RBIF en el registro INTCON<0>, y si el bit GEIE está habilitado genera una interrupción. Esta opción es muy útil cuando estamos en aplicaciones de bajo consumo, cuando se pone en modo de *sleep*, si se tiene un teclado, se podría despertar sólo con oprimir una tecla.

9.4 WATCHDOG TIMER (WDT)

El watchdog es un temporizador (Figura 9.11) cuya finalidad es resetear el micro cuando el programa se pierda, evitando que el micro haga operaciones no previstas. Para evitar este reset, el usuario debe poner la instrucción CLRWDT en partes por donde el programa debe pasar regularmente con un periodo menor que el tiempo del WDT. El watchdog es un timer RC interno que no requiere componentes externos, es separado del oscilador RC del pin OSC1/CLK1; esto significa que el WDT puede aun correr si el clock sobre los pines OSC1/CLKI y OSC2/CLKO ha sido detenido por la instrucción sleep.

Durante la operación normal, el WDT time-out genera un reset (watchdog timer reset) cuando no se encontró la instrucción CLWDT, o se puso en modo sleep. Si el micro está en modo de sleep, el WDT time-out despertará el micro y continuará con la operación normal (watchdog timer wake-up). El bit en el registro **STATUS<4>** será puesto a cero ("0"), cuando haya un WDT time-out.

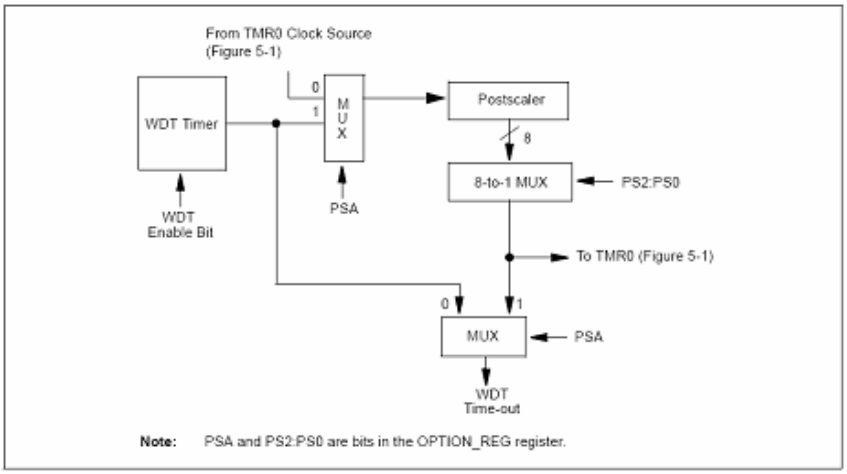


Figura 9.11 Diagrama del watchdog.

El *watchdog* comparte el *prescaler* con el TMR0, se asigna al WDT con el bit PSA puesto a uno ("1") en el **OPTION_REG<3>.** Cuando el *prescaler* está asignado al WDT, para el caso actúa como un *postscaler* que se puede programar con los tres

bits PS2:PS0 en el **OPTION_REG<2:0>.** En la Tabla 9.7 se muestran los valores **de *prescaler* para WDT y TMR0.**

Tabla 9.7 Valores de prescaler para WDT y TMR0

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

El valor del tiempo de ciclo del WDT se extrae de las especificaciones eléctricas en la hoja técnica, y éste aparece como parámetro #31 en la tabla. Normalmente entre 7 y 30 ms; lo típico son 18 ms. Al ejecutar la instrucción **CLRWDT** se resetean el *watchdog timer* y el *prescaler*, pero no la asignación del prescaler. Así comienza de nuevo otro periodo de WDT, hasta encontrarse otra vez con otra instrucción CLRWDT antes de un periodo de WDT. Podemos alcanzar periodos de hasta 2.3 segundos, típicos con la ayuda del **prescaler**, que en este caso actúa como un *postscaler*, multiplicando hasta 128 ciclos de WDT * (18 ms) = 2 304 ms, o sea 2.3 s. En la Tabla 9.8 se dan los valores WDT.

Tabla 9.8 valores WDT.

Param No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
30	TMCL	$\overline{MCLR}$ Pulse Width (low)	2	—	—	µs	VDD = 5V, -40°C to +85°C
31*	TWDT	Watchdog Timer Time-out Period (no prescaler)	7	18	33	ms	VDD = 5V, -40°C to +85°C
32	TOST	Oscillation Start-up Timer Period	—	1024 TOSC	—	—	TOSC = OSC1 period
33*	TPWRT	Power-up Timer Period	28	72	132	ms	VDD = 5V, -40°C to +85°C
34	TIOZ	I/O High-Impedance from $\overline{MCLR}$ Low or Watchdog Timer Reset	—	—	2.1	µs	
35	TBOR	Brown-out Reset Pulse Width	100	—	—	µs	VDD ≤ VBOR (D005)

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

NOTA:

- El WDT puede ser permanentemente deshabilitado, borrando el bit WDTE en los bits de configuración.

- Cuando en un programa se habilita el WDT, se debe garantizar que la instrucción CLRWDT debe estar en un punto donde pase regularmente el programa, con un tiempo menor que el periodo del WDT.

- Si se habilita el WDT y en el programa no se tiene la instrucción CLRWDT o tome más del tiempo que el del periodo del WDT para ejecutar la primera instrucción CLRWDT, el micro realizará un reset cada WDT time-out.

9.5 BAJA POTENCIA (MODO SLEEP)

Ejecutando la instrucción ***sleep*** se entra en el modo de bajo consumo o *power-down* (Tabla 9.9). Si el WDT se habilita, éste puede ser borrado con la instrucción CLWDT, pero sigue corriendo. En el registro STATUS el bit $\overline{PD}$ =0, bit <3> es borrado ("0") y el bit -TO=1, bit <4> es puesto a uno ("1"); y el driver del oscilador se apaga, los puertos mantienen su estado (alto-bajo o alta impedancia), justo antes de que la instrucción **sleep** sea ejecutada.

Para bajos consumos en este modo:

- Ponga todos los I/O ya sea a V_{DD} o V_{SS}.
- Asegúrese de que no haya **circuitería** externa que esté drenando corrientes desde el pin I/O.
- Apague los A/D.
- Deshabilite los *clock*s externos.
- Ponga todos los pines I/O, que son de alta impedancia de entrada, en alto o en bajo externamente, para evitar corrientes de causados por entradas flotantes.
- La entrada T0CK1 debe ponerse a V_{DD} o V_{SS} para bajo consumo de corriente.
- La contribución de los *pull-up* del PORTB debe también ser considerada.
- El $\overline{MCLR}$ debe ser una lógica de alto nivel (V_{IHMC}).

Tabla 9.9 Valores de Power-down.

PIC16LF873A/874A/876A/877A (Industrial)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial				
PIC16F873A/874A/876A/877A (Industrial, Extended)			Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial -40°C ≤ TA ≤ +125°C for extended				
Param No.	Symbol	Characteristic/ Device	Min	Typ†	Max	Units	Conditions
	IDD	**Supply Current(2,5)**					
D010		16LF87XA	—	0.6	2.0	mA	XT, RC osc configurations, FOSC = 4 MHz, VDD = 3.0V
D010		16F87XA	—	1.6	4	mA	XT, RC osc configurations, FOSC = 4 MHz, VDD = 5.5V
D010A		16LF87XA	—	20	35	µA	LP osc configuration, FOSC = 32 kHz, VDD = 3.0V, WDT disabled
D013		16F87XA	—	7	15	mA	HS osc configuration, FOSC = 20 MHz, VDD = 5.5V
D015	ΔIBOR	**Brown-out Reset Current(6)**	—	85	200	µA	BOR enabled, VDD = 5.0V

Legend: Rows with standard voltage device data only are shaded for improved readability.

† Data in "Typ" column is at 5V, 25°C, unless otherwise stated. These parameters are for design guidance only and are not tested.

Note 1: This is the limit to which VDD can be lowered without losing RAM data.

2: The supply current is mainly a function of the operating voltage and frequency. Other factors, such as I/O pin loading, switching rate, oscillator type, internal code execution pattern and temperature, also have an impact on the current consumption.
The test conditions for all IDD measurements in active operation mode are:
OSC1 = external square wave, from rail-to-rail; all I/O pins tri-stated, pulled to VDD;
MCLR = VDD; WDT enabled/disabled as specified.

3: The power-down current in Sleep mode does not depend on the oscillator type. Power-down current is measured with the part in Sleep mode, with all I/O pins in high-impedance state and tied to VDD and VSS.

4: For RC osc configuration, current through REXT is not included. The current through the resistor can be estimated by the formula Ir = VDD/2REXT (mA) with REXT in kΩ.

5: Timer1 oscillator (when enabled) adds approximately 20 µA to the specification. This value is from characterization and is for design guidance only. This is not tested.

6: The Δ current is the additional current consumed when this peripheral is enabled. This current should be added to the base IDD or IPD measurement.

7: When BOR is enabled, the device will operate correctly until the VBOR voltage trip point is reached.

9.5.1 DESPERTANDO DE SLEEP (WAKE-UP FROM SLEEP)

El micro puede despertar del modo *sleep* por uno de los eventos siguientes:

- *Reset* externo por el pin.
- *Time-out* del WDT, si está habilitado (*watchdog timer wake-up*).
- Interrupción por el pin INT o cambio en PORTB (*pin on change*), o interrupción de periférico.

Un $\overline{MCLR}$ externo causará un *reset* del micro. Las otras causas son consideradas como una continuación del programa de ejecución y causan un despertar *wake-up* (son las causas descritas abajo). Los bits $\overline{TO}$ y $\overline{PD}$ pueden usarse para determinar la causa del *reset* del micro. El bit $\overline{PD}$ es puesto a un '1' luego de un encendido *power-up*; es borrado cuando un *sleep* es ejecutado. El bit $\overline{TO}$ es borrado cuando ha ocurrido un WDT *time-out* y causa un despertar *wake-up*. Las interrupciones de los siguientes periféricos pueden despertar el micro del modo *sleep*:

- PSP lectura o escritura (PIC16F874-877 solamente).
- TMR1, debe estar operando como un Contador Asíncrono.
- TMR1, evento especial de *Trigger* en modo asincrónico con *clock* externo.
- CCP en modo de captura.
- SSP bit de detección *start/stop*.
- SSP transmitir o recibir en modo esclavo (SPI/I²C).
- USART Rx o Tx (modo esclavo sincrónico).
- Conversión A/D.
- EEPROM, operación de escritura completada.
- Comparador análogo cambia estado de salida.

Otros periféricos no pueden generar interrupciones durante *sleep*, porque no hay *clock* presente en el chip. Cuando se está ejecutando la instrucción **sleep**, la instrucción siguiente (PC + 1) es precargada. Para que el micro se despierte por una interrupción, el correspondiente bit de habilitación de la interrupción (*interrupt enable*) debe estar puesto a uno (habilitado = "1"). No importa si el estado del bit GIE es cero, el micro se despierta, pero según el estado del GIE tiene dos posibles comportamientos:

- GIE = 0, ejecuta línea siguiente a *sleep* y continúa.
- GIE = 1, ejecuta línea siguiente a *sleep* y salta al vector de interrupción 0004h.

Se debe considerar si la instrucción siguiente no es deseada; se puede agregar un NOP después de la instrucción *sleep*.

9.5.2 DESPERTANDO USANDO INTERRUPCIONES

Cuando el GIE es deshabilitado y cualquier fuente de interrupción tiene ambos bits puestos a unos, el bit de habilitación de interrupción **xxxxIE**, y el bit de bandera de interrupción **xxxxIF**, puede ocurrir una de las siguientes situaciones:

- Si la interrupción ocurre antes de la ejecución de la instrucción **sleep**, la instrucción **sleep** se completa como un NOP, por lo tanto:
  - *Watchdog timer* y *postscaler* pueden no ser borrados.
  - $\overline{TO}$ puede no ser puesto a uno (“1”).
  - $\overline{PD}$ puede no ser puesto a cero (“0”).
- Si la interrupción ocurre durante o después de la ejecución de la instrucción **sleep**, el micro puede despertar inmediatamente del modo de *sleep*. La instrucción **sleep** se completa totalmente antes de despertar; por lo tanto:
  - *Watchdog timer* y *postscaler* son puestos a cero.
  - $\overline{TO}$ es puesto a uno (“1”).
  - $\overline{PD}$ es puesto a cero (“0”).

Incluso si los bits de las banderas se verificaran antes de ejecutar una instrucción **sleep**, puede que el bit de las banderas comience a ponerse a uno antes que la instrucción **sleep** se complete. Para determinar si una instrucción **sleep** se completó, pruebe $\overline{PD}$. Si $\overline{PD}$ es puesto a uno (“1”) la instrucción **sleep** fue ejecutada como un NOP, o sea que la interrupción no es válida.

Si $\overline{PD}$ es puesto a cero (“0”), la instrucción **sleep** se completó, o sea que la interrupción es válida. Para tener la seguridad de que el registro del **WDT** es borrado, la instrucción **CLRWDT** debe ser ejecutada antes que la instrucción **sleep**. En la Figura 9.12 se aprecia un diagrama de tiempos de un proceso de despertar (*wake-up*) con interrupción.

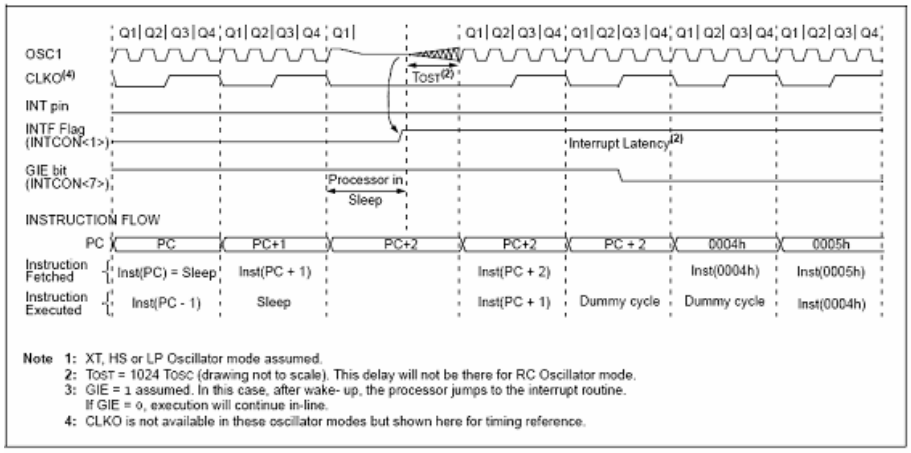


Figura 9.12 Despertar de sleep con interrupción.

9.6 BITS DE CONFIGURACIÓN (CONFIGURATION BITS)

Los **bits de configuración** son los encargados de habilitar algunas propiedades del micro, como proteger la memoria, permitir *debug*, habilitar **WDT** y escoger el oscilador, etc, esto se hace únicamente en el modo de programación y no se pueden cambiar una vez programados. Los bits de configuración pueden ser programados en el código fuente a través de la instrucción **_CONFIG,** que es una directiva del **MPLAB**, y debe ir al principio del programa. El **MPLAB** también tiene la forma de programar estos bits en la opción **configure/configuration bits**, donde se despliega una ventana en donde se pueden cambiar sus valores. Estos bits están en la posición 2007h y solamente pueden ser accesados en modo de programación. Si no son programados, estos 14 bits tienen un valor de 3FFFh todos "1", al ser habilitados o programados, se pone el bit en "0" (Tabla 9.10).

Tabla 9.10 Bits de configuración (Configuration bits).

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	PWRTEN	WDTEN	Fosc1	Fosc0
bit 13													bit0

bit 13 **CP:** Flash Program Memory Code Protection bit
1 = Code protection off
0 = All program memory code-protected

bit 12 **Unimplemented:** Read as '1'

bit 11 **DEBUG:** In-Circuit Debugger Mode bit
1 = In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins
0 = In-Circuit Debugger enabled, RB6 and RB7 are dedicated to the debugger

bit 10-9 **WRT1:WRT0** Flash Program Memory Write Enable bits
For PIC16F876A/877A:
11 = Write protection off; all program memory may be written to by EECON control
10 = 0000h to 00FFh write-protected; 0100h to 1FFFh may be written to by EECON control
01 = 0000h to 07FFh write-protected; 0800h to 1FFFh may be written to by EECON control
00 = 0000h to 0FFFh write-protected; 1000h to 1FFFh may be written to by EECON control
For PIC16F873A/874A:
11 = Write protection off; all program memory may be written to by EECON control
10 = 0000h to 00FFh write-protected; 0100h to 0FFFh may be written to by EECON control
01 = 0000h to 03FFh write-protected; 0400h to 0FFFh may be written to by EECON control
00 = 0000h to 07FFh write-protected; 0800h to 0FFFh may be written to by EECON control

bit 8 **CPD:** Data EEPROM Memory Code Protection bit
1 = Data EEPROM code protection off
0 = Data EEPROM code-protected

bit 7 **LVP:** Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit
1 = RB3/PGM pin has PGM function; low-voltage programming enabled
0 = RB3 is digital I/O, HV on MCLR must be used for programming

bit 6 **BOREN:** Brown-out Reset Enable bit
1 = BOR enabled
0 = BOR disabled

bit 5-4 **Unimplemented:** Read as '1'

bit 3 **PWRTEN:** Power-up Timer Enable bit
1 = PWRT disabled
0 = PWRT enabled

bit 2 **WDTEN:** Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1-0 **Fosc1:Fosc0:** Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
- n = Value when device is unprogrammed		u = Unchanged from programmed state

Note 1: The erased (unprogrammed) value of the Configuration Word is 3FFFh.

9.6.1 CÓDIGO DE PROTECCIÓN (CODE PROTECTION)

El bit de **CP** (Código de Protección) en el registro **Configuration Word <13>** es el encargado de proteger que el programa pueda ser leído o no. Si CP=1, el código de programa del MCU puede ser leído. Si CP=0, el código de programa del MCU está protegido y no puede ser leído.

9.6.2 IDENTIFICADOR (ID LOCATIONS)

Este **código de identificación** consiste en 4 posiciones de memoria, 2000 – 2003h o *ID location*, aquí se puede guardar un *checksum* o cualquier otro código de identificación propio. Estas posiciones no son leíbles durante la ejecución normal del micro, pero se pueden leer y escribir durante la programación/verificación. Se deben usar los 4 bits más bajos o LSB para escribir el *user ID*, véanse las Figuras 9.13 y 9.14).



Figura 9.13 Ventana de id user.

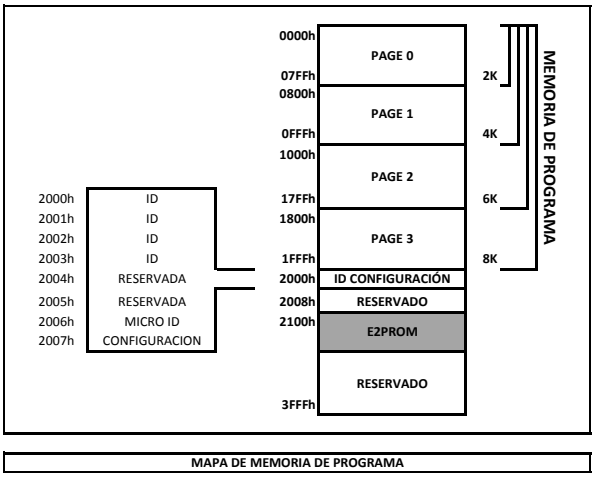


Figura 9.14 Mapa de memoria de posiciones id.

9.7 DEPURACION (IN-CIRCUIT DEBUGGER)

La **depuración** permite, junto con el **MPLAB**, visualizar y cambiar registros, correr paso a paso, poner *breakpoint*, y otras muchas ventajas para el desarrollador.

Cuando la depuración está habilitada (puestos a cero "0") en el **Configuration bits**, algunos recursos no están disponibles para la aplicación (Tabla 9.11).

Tabla 9.11 Recursos en depuración.

I/O pins	RB6, RB7
Stack	1 level
Program Memory	Address 0000h must be NOP
	Last 100h words
Data Memory	0x070 (0x0F0, 0x170, 0x1F0) 0x1EB-0x1EF

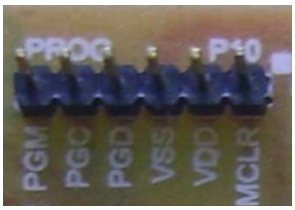
Cuando esté realizando depuración, recomendamos:

- En lo posible, no use los pines RB6 y RB7.
- No use las posiciones de memoria 0X70 (o 0XF0, 0X170, 0X1F0), 0X1EB a 0X1EF en RAM.
- Conserve siempre un nivel de *stack* libre.

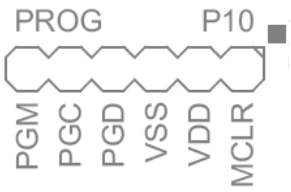
9.8 ICSP (IN-CIRCUIT SERIAL PROGRAMMING)

Esta forma de programar los MCU permite que se haga directamente sobre la tarjeta final de aplicación, y no se deba retirar el micro de la tarjeta. Sólo hay que poner un conector con 6 pines con las señales de *clock* y datos, más las fuentes de V_{DD}, V_{SS} y la de programación /V_{PP} (Figura 9.15 a, b y c).

a)



b)



c)

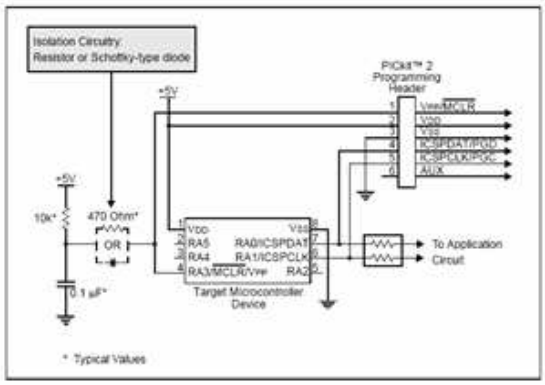


Figura 9.15 Diagrama Conexión ICSP

El circuito de programación es como se muestra a continuación (Figura 9.15): la señal del pin 6 no se conecta. Es necesario poner las **Rs** (resistencias) del circuito: de R = 470 ohm, debido a que por ese pin se aplican 14 V_{CD} de programación, y si los pines RB6 y RB7 no están libres, hay que poner las resistencias R = 4.7 Kohm). Al trabajar con una herramienta debemos tener la precaución de llevar alimentación al conector del ICSP, de tal manera que los V_{DD} = 5 V_{CD} de la alimentación de nuestra aplicación se deben conectar al pin 2 (V_{DD}) del conector ICSP.

Conviene anotar que las herramientas de **Microchip** no están diseñadas para entregar más de 25 mA por el pin 2 (V_{DD}) del conector ICSP. La herramienta toma la alimentación de la conexión USB. La conexión al pin 2 (V_{DD}), sólo es de referencia, y debe conectarse, si no el depurador notará un error.

9.8.1 PROGRAMANDO EN BAJO VOLTAJE ICSP

Se puede programar en alto voltaje V_{PP} = 14 V_{DD} en el pin, o se puede en bajo voltaje con V_{DD} en el pin, y en el pin RB3/PGM se aplica V_{DD} provisto por el bit LVP cuando es puesto a uno “1”.

```
NOTA: En bajo voltaje el pin RB3/PGM no puede ser usado como I/O.
```

Hay que observar cuidadosamente la configuración para la compatibilidad de LVP con otros micros. (Este punto se explica más detalladamente en el manual del PICkit 2 en la Web.)

10.0 SET DE INSTRUCCIONES

Éste es el conjunto de instrucciones con el que cuenta el MCU para hacer todas las operaciones; debido a su **arquitectura RISC**, tenemos solamente 35 instrucciones de trabajo. Cada instrucción en los PIC micro *mid-range* es de 14 bits, divididos en un **opcode**, que especifica el tipo de instrucción y uno o más operandos que, además, especifican la operación de la instrucción. Podemos agrupar las operaciones en tres categorías:

- Operaciones orientadas a byte.
- Operaciones orientadas a bit.
- Operaciones literales y control.

10.1 OPERACIONES ORIENTADAS A BYTE

Mnemónico f,d

Mnemónico = definición en letras de la instrucción

f = dirección del registro que va a ser usado por la instrucción (7 bit, 0 - 255)

d = destino, d=0 destino W, d=1 destino f

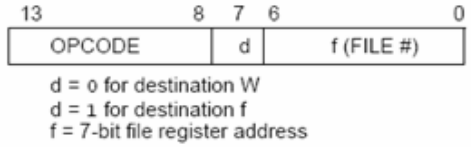


Tabla 10.1 Operaciones orientadas a byte.

10.2 OPERACIONES ORIENTADAS A BIT

Mnemónico f,b

Mnemónico = definición en letras de la instrucción

f = dirección del registro que se afectará (7 bits, 0 - 255)

b = bit afectado (3 bits, 0 -7)

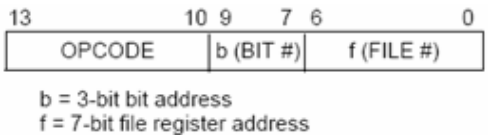


Tabla 10.2 Operaciones orientadas a bit.

10.3 OPERACIONES LITERALES Y CONTROL

General:

Mnemónico k

Mnemónico = definición en letras de la instrucción

k = valor literal (7 bits, 0 - 255)

Instrucciones CALL y GOTO

Mnemónico k

Mnemónico = definición en letras de la instrucción

k = valor literal (11 bits, 0 – 2 K)

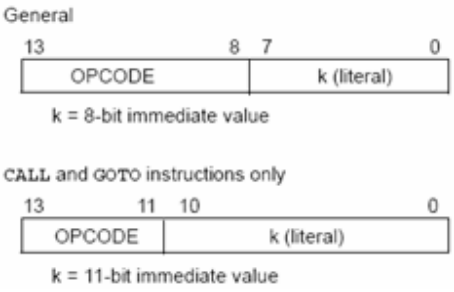


Tabla 10.3 Operaciones literales y control.

En la Tabla 10.4 hay definiciones necesarias para los campos del Opcode del set de instrucciones.

Tabla 10.4 Descripción de campos del OPCODE

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1.
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

Tabla 15.5 Set de instrucciones.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add Literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND Literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call Subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to Address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR Literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move Literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from Interrupt	2	00	0000	0000	1001		
RETLW	k	Return with Literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from Literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR Literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Note: Additional information on the mid-range instruction set is available in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

Todas las instrucciones están detalladas en el Apéndice B: Set de instrucciones.

PARTE 3
PROGRAMACIÓN (I)

Para comprender esta Parte 3 se describe brevemente el proceso de programación con los microcontroladores y las partes que contiene, luego procedemos en la forma secuencial de cómo se concreta una idea o la necesidad, hasta llegar a la realidad con la implementación de una aplicación. El contenido se divide en dos partes: la **elaboración** del programa y luego la **programación** del microcontrolador.

En la **elaboración** del programa veremos los conceptos básicos que involucran desde la idea, el algoritmo, el diagrama de flujo, hasta el código o el programa, ampliando las partes que involucran un programa. Luego, en la **programación** del microcontrolador, se describen las herramientas del ambiente de desarrollo, el programador/depurador y la aplicación; veremos cada una de estas partes y los tres modos de trabajo: simulación, depuración y programación. Con esto queda descrito el proceso de programación desde la idea hasta la aplicación, pasando por todos los conceptos y las herramientas necesarias.

11.0 CONCEPTOS BÁSICOS DE PROGRAMACIÓN CON MCU

En el mundo de los microcontroladores los conceptos de programación están asociados con varios elementos que se deben identificar claramente, y que se describen a continuación. La **idea de programación** en el ámbito de los microcontroladores tiene que ver con el hecho de que un aparato (máquina, dispositivo, etc.), que tiene opciones de funcionamiento, trabaje de la forma que se

espera. En nuestro caso, hacer que un MCU opere como yo espero. Para realizar mi idea o mi proyecto se requiere que se realicen dos pasos:

- Elaborar un programa
- Enviarlo al MCU

```
NOTA:

Aquí se sobreentiende que otras partes del proyecto, como las
mecánicas o de implementación, deben estar ya cubiertas por el
desarrollador, y no son materia de este libro.

Todo el tratamiento de cualquier proyecto será exclusivamente
concentrado en la parte del microcontrolador.
```

Cuando se habla de **elaborar un programa**, se hace referencia a los pasos necesarios desde tomar la idea o la necesidad y concretarla en un programa, para esto se levantan las especificaciones de la idea, pasando por la descripción de un algoritmo en diagrama de flujo, y de aquí pasarlo a código para convertirse en programa. Estos pasos sólo requieren lápiz, papel y el ambiente de desarrollo para escribir el programa (software MPLAB).

Enviarlo al MCU se refiere a la forma en que se envía el programa desde la PC en el ambiente de desarrollo al MCU, en la tarjeta de aplicación, utilizando la herramienta de programación. Este proceso pasa por el manejo de las herramientas de Software y Hardware. Una forma gráfica de describir el proceso de programación es el que apreciamos en la Figura 11.1; este diagrama también muestra un mapa del contenido de esta Parte 3. Se puede ver que los pasos están relacionados con los conceptos y temas que veremos a continuación.

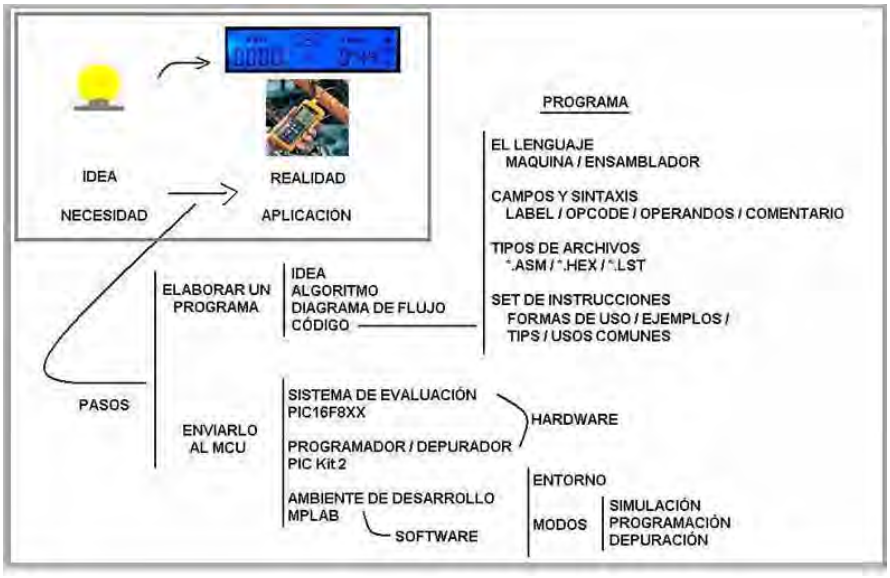


Figura 11.1 Mapa de contenido.

De este modo se puede llevar una **idea** (necesidad) a la **realidad** (aplicación o solución); es decir, el microcontrolador junto con todas sus partes adicionales operan de conformidad con nuestra idea original. Para que se concrete la idea, se supone que se puede implementar con la tarjeta de sistema de evaluación **PIC16F8XX**, junto a otras partes que se conecten a ella; además de esto se requieren otras herramientas. Las herramientas necesarias para realizar un proyecto con microcontrolador son:

1. **Software MPLAB** (u otro IDE) software instalado en PC, para elaborar programa, como proyecto con archivo *.asm, para luego generar archivo *.hex. (MPLAB).

2. **Hardware Programador/Depurador** conectado a la PC, para enviar el programa al MCU, donde queda como instrucciones en la memoria de programa FLASH. En nuestro caso, usaremos el PICkit 2. Sin embargo, hay otras herramientas explicadas en la PARTE 1.

3. **Hardware Tarjeta de aplicación** o protoboard (en nuestro caso, sistema de evaluación **PIC16F8XX**); aquí se encuentra el MCU junto con otros dispositivos que facilitan el manejo de los módulos (Figura 11.2).

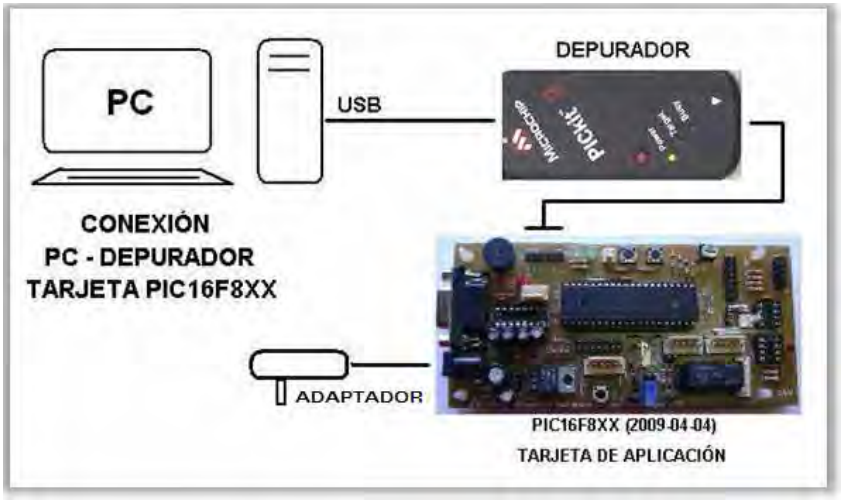


Figura 11.2 Herramientas para proyecto.

El **Software MPLAB** es la herramienta de software que, una vez instalada en la PC, ofrece el ambiente de desarrollo para crear programas, con editor de texto, compilador y otras facilidades para el desarrollador. El **Programador/Depurador** es la herramienta de hardware encargada de recibir de la PC los datos que se deben programar en el microcontrolador, que también está encargado de realizar la depuración y la programación del mismo. Tiene conexión hacia la PC con cable USB y hacia la tarjeta con conector ICSP (de 6 pines). La **Tarjeta de Aplicación** es el hardware final; en nuestro caso, es el sistema **de evaluación PIC16F8XX** que contiene el microcontrolador PIC16F877A; además contiene un conector ICSP (para depuración) y conector para adaptador de alimentación; también conexiones a todos los periféricos para facilitar el aprendizaje del microcontrolador.

Cuando hablamos de **programa**, éste se realiza inicialmente en la PC, en el ambiente de desarrollo MPLAB, cuando creamos un proyecto en uno de sus archivos **(*.asm)** y allí generamos el código o programa. Este archivo es el código fuente, que luego se compila y queda en un **ejecutable (*.hex)** (más adelante veremos que no es propiamente un ejecutable, más bien un formato de comunicaciòn). Luego este archivo será enviado al microcontrolador **PIC16F877A** y quedará en la **memoria de programa FLASH**, como instrucciones en las posiciones que se indicaron en el programa. El **Programador/Depurador** es el que tiene la tarea de comunicarse con el software de **PC MPLAB** y facilitar el envío de los datos de la PC al microcontrolador. Una vez el programa en el micro, se podrá verificar su funcionamiento con la ayuda de la depuración. Ya con la idea clara de lo que involucra la programación, podemos adentrarnos en los conceptos que debemos aclarar.

11.1 PROCESO PARA ELABORAR UN PROGRAMA

Cuando se habla del proceso de elaborar un programa se hace referencia a las diferentes etapas que implica el tomar una idea (necesidad) y volverla realidad (aplicación o solución). Se pueden identificar algunas grandes partes de la manera siguiente:

- Idea
- Algoritmo
- Diagrama de flujo
- Código

En seguida describiremos cada una de estas partes y las relacionaremos con un ejemplo para que se tengan claras en una versión práctica.

11.1.1 IDEA

En este punto es donde se nos ocurre una idea o deseamos suplir una necesidad. Desde aquí parte el esfuerzo para concretar la idea; se debe escribir la idea teniendo en cuenta que sea muy concreta, y tratando de acotar márgenes o niveles o valores, detallando la forma de funcionamiento lo más claro posible; esto también es comúnmente llamado levantamiento de especificaciones. A modo de ejemplo, realizaremos una idea.

Ejemplo: Quiero tomar la temperatura de mi casa.

Con este sencillo ejemplo trataremos de realizar el ejercicio de concretar una idea. Se realizan las preguntas que nos ayuden a definir valores y a acotar márgenes, definir interfaces humanas y de comunicación y, por último, la fuente de alimentación. Las preguntas siguientes relacionadas con este caso específico, son a modo de ejemplo, y se agrupan en los tipos siguientes:

a) Definición de valores

- ¿Por cuánto tiempo quieres sensar la temperatura?

  1 año.
- ¿Con qué resolución la quieres (cada cuánto tiempo se hace la medición)?

  Cada ½ minuto.
- ¿Quieres guardar las mediciones, cada cuánto tiempo?

  Sí, cada semana.

- ¿De qué parte de la casa, interior o exterior?

  Interior.
- ¿Entre qué rango de temperaturas está la medición?

  15 a 33 ºC.

Interfaz humana y comunicaciones

- ¿Cómo vas a leer e ingresar datos como fecha?

  Con LCD y teclado 4×4.
- ¿Quieres comunicación con PC para leer los datos?

  Sí.

Alimentación de energía

La alimentación de energía es de 110 VAC y respaldo con baterías de 12 VCD.

Funcionamiento

Se debe leer la temperatura cada 30 segundos y almacenarla en la memoria; también se deben mostrar en un display LCD (o display de 7 segmentos) el valor actual, el promedio del día, máximo y mínimo. Por medio del teclado y la pantalla LCD, se debe tener un menú de opciones en donde se encuentre:

1. Enviar los datos a la PC
2. Opciones para ingresar fecha y hora

Con la información anterior podemos levantar un diagrama en bloques circuital, como el que aparece en la Figura 11.3; esto nos da una idea del hardware. El nombre más acorde con la función de la aplicación que se describió antes es un registrador de temperatura. El sistema anterior es como el que vemos en la Figura 11.3, está compuesto de MCU (µC), pantalla LCD, teclado 4×4, comunicación RS232 con PC, memoria EEPROM (protocolo I2C), el sensor de temperatura y la fuente de energía.

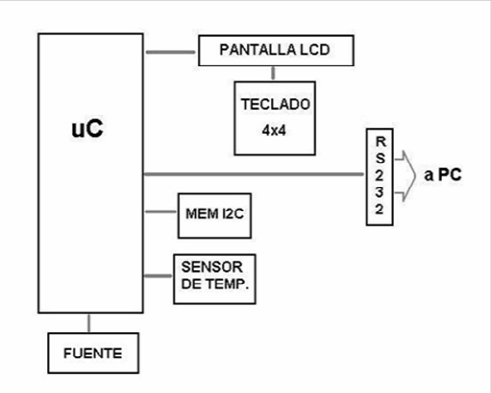


Figura 11.3 Diagrama circuital registrador de temperatura

Tabla 11.1 Resumen de datos del registrador de temperatura.

DESCRIPCIÓN	DATO
Rango de temperatura	15 a 33 ℃
Intervalo de lecturas	30 segundos
Guardar mediciones durante	1 semana
Tiempo total de registro	1 año
Conexión a PC para lectura	Sí
Interfaz humana	LCD y teclado 4×4
Alimentación de energía	Adaptador a 110 VAC Batería 12 VCD

Con esta información estamos ya listos para continuar con el paso siguiente hacia la concreción de la idea. De aquí en adelante nos enfocaremos en el aspecto de la programación, y supondremos el hardware ya solucionado (implementado con PIC16F8XX).

11.1.2 ALGORITMO

El algoritmo es una secuencia bien definida, ordenada y finita de operaciones que permite llegar a la solución de un problema. Al tratar de solucionar o implementar una solución pueden aparecer diferentes formas de realizarlo. Tanto como: formas lógicas de enfrentar el reto, y también las formas de diagramas gráficos. Sin embargo, todas las formas de realizarlo deben solucionar el reto.

```
NOTA:
La intención del algoritmo es hacer un proceso sistemático que al
final se traduce en un programa.
```

Algunos de los puntos comunes que deben cumplir los algoritmos son:

- Fácil de leer y entender la secuencia.
- Que empleen métodos rápidos y eficientes.
- Que sea modular para permitir subdividir en módulos menores, lo que brinda facilidad de manipulación y actualización.

Unas de las implementaciones gráficas y lógicas pueden ser:

- Diagramas de flujo
- Diagrama estructurado
- Diagramas de estado

Cualquiera de estas implementaciones está ampliamente documentada en internet o en textos relacionados, escapan al propósito de este libro y por tanto sólo se mencionan. Utilizaremos los diagramas de flujo como implementación gráfica, utilizando un proceso de diseño *Top-Down*, o sea subdividir por módulos mayores a módulos menores utilizando un seudocódigo para facilitar el entendimiento del diagrama.

```
NOTA:
Top –Down es una forma de trabajo de lo más Alto (Top) a lo más
bajo (Down).
Pseudocódigo es la forma de escribir naturalmente un programa, sin
llegar al código
```

11.1.3 DIAGRAMA DE FLUJO

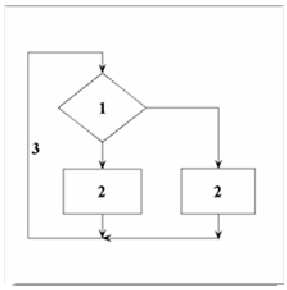
Una de las formas gráficas más ampliamente usadas para expresar los algoritmos es el diagrama de flujo, mediante símbolos que contienen texto o seudocódigo y líneas de sentido o flechas que permiten visualizar el flujo y el sentido de un programa (que al final es el propósito del algoritmo). La diagramación tiene algunos conceptos básicos que veremos a continuación.

a) Estructura

Cualquier programa por complicado que sea se puede representar con tres **tipos de estructuras**, las que podemos apreciar en la Figura 11.4, éstas son:

1. **Bifurcación.** Selección de ejecutar una tarea con base en una decisión
2. **Sentencia.** Toda tarea que se ejecuta
3. **Cíclica.** Repetición de un proceso

Figura 11.4
Estructuras gráficas



Con estas estructuras se pueden simbolizar todas las operaciones posibles en un microcontrolador.

b) Símbolos

Para construir un diagrama de flujo con los tipos de estructura y todas las partes que puede tener un programa, nos ayudaremos de los símbolos siguientes (Figura 11.5):

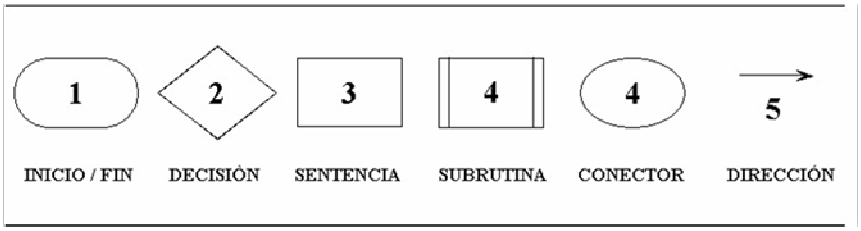
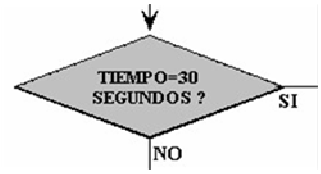


Figura 11.5 Símbolos para diagramas de flujo.

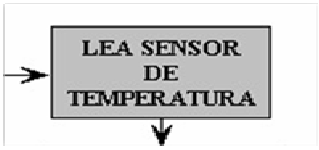
Cada una de estas estructuras tiene las reglas de uso siguientes:



*** INICIO/FIN.** Un rectángulo con lados ovalados representa el inicio o el fin del programa o de rutinas.



***DECISIÓN.** Un rombo representa una decisión que da lugar a que se tomen dos posibles caminos, dependiendo de que la decisión sea Verdadera/Falsa o Sí/No.



***SENTENCIA.** Un rectángulo representa una sentencia u operación, que son la mayor parte de las operaciones realizadas en un programa.



***SUBRUTINA.** Un cuadrado con dos rayas a los lados significa el llamado a subrutinas, que son módulos más pequeños de programa.



***CONECTOR**. Un óvalo o círculo son **conectores** que nos ayudan a segmentar en partes el programa y conectarlos a través de estos símbolos.



***DIRECCIÓN**. Una flecha nos indica el sentido del flujo del programa.

Los diagramas de flujo suelen hacerse en seudocódigo, de tal manera que son expresiones en lenguaje natural que describen con más claridad las tareas, sin llegar a detallar el código.

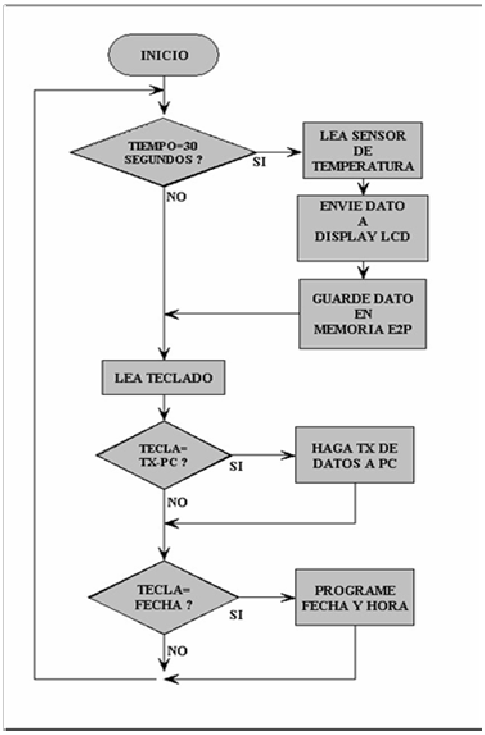


Figura 11.6 Diagrama de flujo registrador de temperatura.

En la Figura 11.6 observamos el diagrama de flujo que describe el funcionamiento del registrador de temperatura, se puede leer naturalmente el proceso que debe seguir el programa para realizar el funcionamiento, tal como se describió en el levantamiento de especificaciones. Está hecho en seudocódigo; donde las tareas se pueden convertir en subrutinas. Cada una de las partes descritas se toma separadamente y se detalla aún más siguiendo el proceso *Top-down*. En la Figura 11.6 observamos cómo las tareas, por ejemplo la sentencia "ENVÍE DATO A DISPLAY LCD", o "HAGA TX DE DATOS A PC" se pueden amplíar en pasos más detallados.

NOTA:
Recordamos que en las aplicaciones en el mundo real generalmente no se tiene un fin de programa, sino un ciclo infinito en donde la aplicación se está ejecutando constantemente. Puede haber incluso estados en donde la máquina entra en modos especiales de bajo consumo (*Sleep*), y aun así el programa puede salir de este modo de bajo consumo y continuar ejecutando la aplicación.

Con todos los pasos anteriores realizados se puede llegar al código que podemos extraer con más facilidad del diagrama de flujo y que nos ayudará a generar el programa.

11.1.4 CÓDIGO

Ésta es la fase final de la concreción de la idea, y contiene lo necesario para realizar la implementación del código a partir del diagrama de flujo. Para explicar con detalle la relación del diagrama de flujo y el código *assembler*, tomaremos un ejemplo de una subrutina para visualizar los datos en un display de 7 segmentos (Figura 11.7).

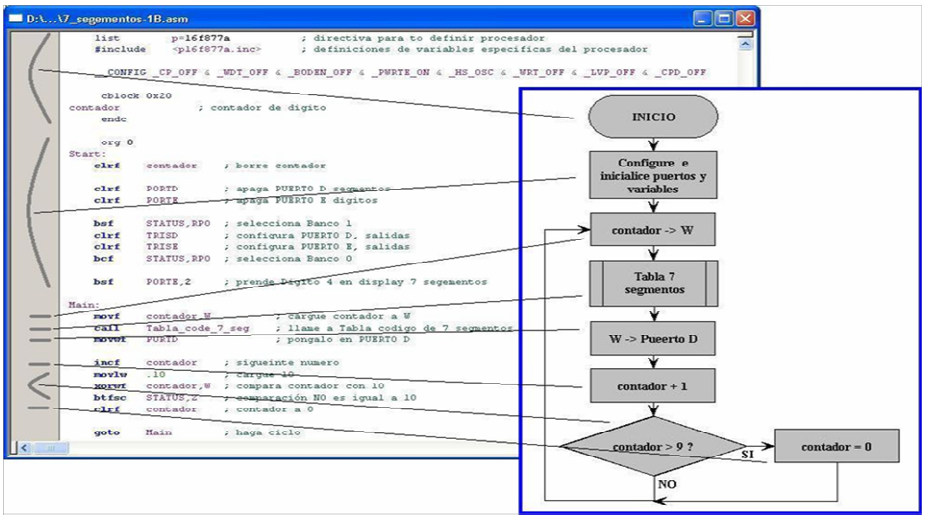


Figura 11.7 Diagrama de flujo y código.

En este ejemplo la idea es visualizar los números decimales de 0 a 9 en un display de 7 segmentos, esto se puede apreciar en la Figura 11.7 donde se encuentran el código assembler y el diagrama de flujo del algoritmo del programa. Vemos cómo no todas las instrucciones en seudocódigo en el diagrama de flujo (a la derecha) son equivalentes a las instrucciones de código assembler (a la izquierda); esto nos quiere decir que al leer un diagrama de flujo, sus símbolos nos pueden corresponder a varias líneas de código assembler. Éste es el último eslabón del proceso para elaborar un programa; se hicieron los pasos necesarios y suficientes para que, de una manera sistemática, se concretara una idea en programa, siguiendo un método sencillo. El anterior es el proceso por el cual debe pasar la idea hasta concretarse en un programa.

11.2 PROGRAMA

Los programas involucran algunos conceptos que debemos tratar, tales como:

- El lenguaje
- Campos y sintaxis
- Tipos de archivos
- Set de instrucciones

De aquí en adelante, abordamos estos temas; resaltando la Sección 11.3, "Set de instrucciones", en donde se amplían las instrucciones en los programas, con tips y formas de uso, los cuales son una gran ayuda para el desarrollador.

11.2.1 EL LENGUAJE

Un **lenguaje** de programación es un conjunto de símbolos, palabras y reglas de sintaxis que definen la estructura del lenguaje y el significado de sus elementos y expresiones. Éstos deben estar relacionados con el **set de instrucciones** de cada microcontrolador y, por ende, con las tareas que éste puede realizar. Los lenguajes pueden ser de bajo y alto nivel, lo cual va relacionado con la capacidad y la facilidad de manejo. En bajo nivel encontramos el **lenguaje de máquina** y *assembler*, que veremos con detalle más adelante.

En alto nivel mencionamos **BASIC** y **lenguaje C**, que con el tiempo se han vuelto un estándar, a los cuales se acogen los fabricantes de microcontroladores, brindando estos lenguajes en sus herramientas de desarrollo. En un nivel superior, tenemos los **lenguajes gráficos** de programación que nos permiten programar a través de **iconos**.

El lenguaje de Máquina

El lenguaje apropiado para el trabajo de los microcontroladores es en 0s y 1s, ya que es una máquina electrónica, toda la circuitería interna está basada en la lógica digital cuyos valores son ON=1 y OFF=0, que se traducen en niveles de voltaje alto o V_{DD}=5V o 1 lógico (V_{DD} puede ser de 3V) y bajo o V_{SS}=0 V o 0 lógico. Por tanto, todas las instrucciones están en lenguaje binario o hexadecimal, o **lenguaje de máquina**, y las instrucciones son llamadas **códigos de máquina.** Las instrucciones hacen parte de los programas, y están alojadas en la memoria de programa; en cada microcontrolador el ancho de la instrucción depende del bus de programa que es de 14 bits en el **PIC16F877A**.

Si acudimos al set de instrucciones y escribimos una instrucción, como: **ponga en 1 el bit 4 del Puerto B**, su resultado en binario sería **1 0110 0000 0110** y su

valor en hexadecimal **1606**. De esta manera, podemos mirar un programa en lenguaje de máquina con sus valores en binario o hexadecimal (Figura 11.8).

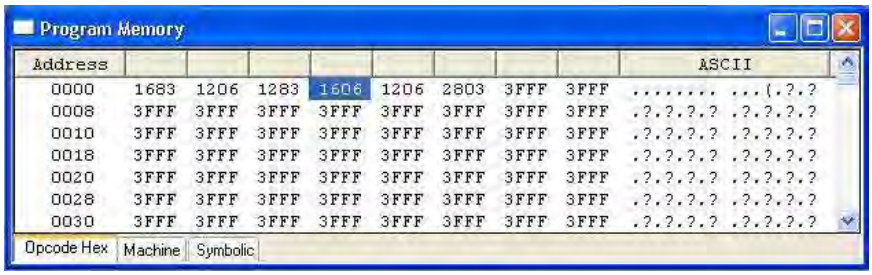


Figura 11.8 Memoria de programa.

En la Figura 11.8 observamos un programa en código hexadecimal; las instrucciones en **3FFFh** son posiciones vacías (o sin quemar todos los bits en "1").

El lenguaje Ensamblador (Assembler)

Para la máquina es muy propicio el lenguaje de 1s y 0s porque es propio de su naturaleza. Sin embargo, el lenguaje del desarrollador es de otra naturaleza, donde lo importante es asociar las instrucciones con abreviaturas que nos den una idea de la tarea que se realiza. Estas abreviaturas se llaman **mnemónicos**, también llamados **opcode**, y en todo su conjunto conforman el **lenguaje ensamblador** que es igual al **set de instrucciones**. De este modo cada mnemónico o instrucción contiene un código binario y su ancho es de 14 bits para este microcontrolador. En la Tabla 11.2 vemos cómo los mnemónicos a la izquierda tienen su descripción y su código de 14 bits, donde están los campos definidos para el código de la instrucción y los operandos f,d de estas instrucciones.

Tabla 11.2 Mnemónicos y códigos.

Mnemonic, Operands	Description	Cycles	14-Bit Opcode MSb LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS					
ADDWF f, d	Add W and f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF f, d	AND W with f	1	00 0101 dfff ffff	Z	1,2

En el momento de escribir un programa, los **mnemónicos** son las instrucciones con las que se elabora; cada una de éstas tiene un código de máquina asociado, los microcontroladores de la serie **10/12/16** marca **Microchip** son de 35 instrucciones; esto nos da la gran ventaja de aprender muy pocas instrucciones para manejar una gran cantidad de dispositivos en diferentes familias.

- **DECODIFICANDO LA INSTRUCCIÓN ADDWF**

Para apreciar un poco mejor el manejo de los mnemónicos a nivel de bit, y para tener una medida del grado de complejidad que involucra el bajo nivel, tomaremos un ejemplo con una instrucción.

Así, por ejemplo, si tomamos un **mnemónico:**

addwf f,d

Adicione el valor del registro W, al valor del registro f y el resultado póngalo en destino d.

Mnemónico	*Descripción*	*Código*
addwf f,d	(W)+(f)-> destino)	00 0111 dfff ffff

Donde:

- d = destino

  d = 0, destino registro W

  d = 1, destino registro f

- f = file o registro RAM

  f = (00 – 7Fh, 0 – 127d)

En nuestro ejemplo, usamos la variable **f = nivel** en la posición 20h, que contiene el valor 1Ah, y en el registro **W** con el valor 2Ch; con estos datos (Tabla 11.3) podemos probar la instrucción addwf.

Tabla 11.3 Instrucción ADDWF.

REGISTROS				
TIEMPO DE	f			W
EJECUCIÓN	nombre	dirección	valor	valor
antes	nivel	20h	1Ah	2Ch
operación				
luego	nivel	20h	1Ah	46h

INSTRUCCIÓN	CODIGO DE INSTRUCCIÓN													
	13	12	11	10	9	8	7	6	5	4	3	2	1	0
addwf f,d	0	0	0	1	1	1	d	f	f	f	f	f	f	f
f=(00h - 7Fh, 0 - 127d)	OPCODE						d	f						
addwf nivel,W	0	0	0	1	1	1	0	0	1	0	0	0	0	0
d=0, destino W	0		7				2				0			

antes	nivel	20h	1Ah	2Ch
operación				
luego	nivel	20h	46h	2Ch

addwf nivel,F	0	0	0	1	1	1	1	0	1	0	0	0	0	0
d=1, destino F	0		7				A				0			

En la Tabla 11.3 apreciamos dos lados, el de la derecha tiene los códigos de la instrucción **addwf**, y en el lado izquierdo encontramos los valores que toman los registros antes y después de ejecutar la operación **addwf.** En consecuencia, si miramos el lado derecho vemos que la instrucción está decodificada en sus 14 bits (bit 0-13), con sus partes de **opcode** (código de la instrucción) y los dos operandos **f,d**. Observamos que la instrucción puede tomar dos valores, según el destino:

Instrucción	Código	Descripción
addwf nivel, W	0720h	Guarda resultado en W, cuando d=0
addwf nivel, F	07A0h	Guarda resultado en F, cuando d=1

En el lado izquierdo de la Tabla 11.3 observamos cómo los registros **f=nivel** y **W** cambian dependiendo de la instrucción o, más exactamente, del destino de la instrucción; el resultado 46h se guarda en **W**, cuando *d*=0 y en **nivel** cuando **d=1**. Cuando se almacena el resultado en **W** el valor que tenía **nivel** se preserva en él y, de forma opuesta, cuando se almacena el resultado en **nivel** el valor que tenía **W** se preserva en él.

11.2.2 CAMPOS Y SINTAXIS

Cada lenguaje tiene asociada una sintaxis, o sea, una forma en que se escriben las instrucciones, en el caso del lenguaje *assembler* igual; el lenguaje *assembler* es muy particular de cada fabricante de microcontroladores, tanto como su **set de instrucciones.** Los IDE (*Integrated Development Environment*) o ambientes de desarrollo tienen sus reglas para escribir los programas en sus editores de texto. Estos campos que describiremos son los más estándares en los editores de texto, que son la herramienta con la que al final se desarrollan los programas, con cada una de sus instrucciones. En cada línea de instrucción (Figura 11.9) se pueden distinguir cuatro campos principales:

- Label (etiqueta)
- Opcode (código de operación)
- Operandos (registros)
- Comentario (para documentar el programa)

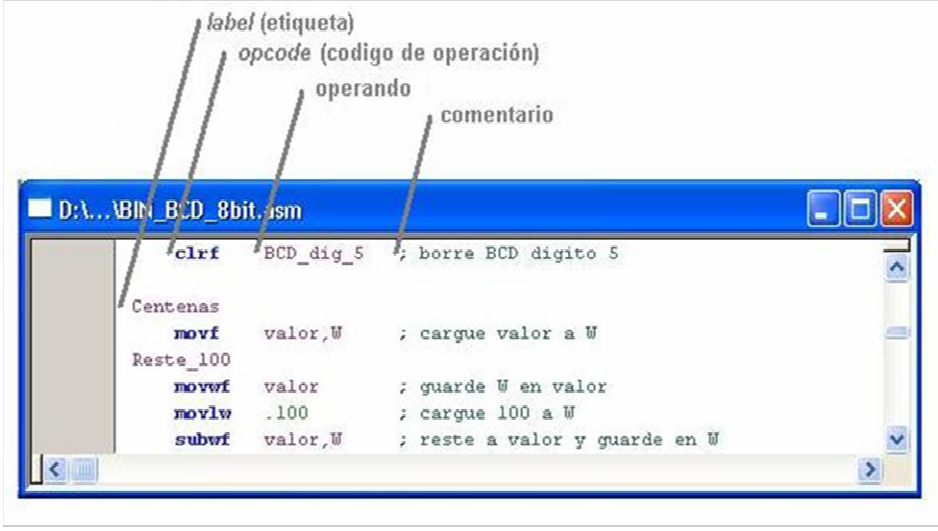


Figura 11.9 Campos y sintaxis.

- **LABEL**

El primer campo es el label o la etiqueta, es de valor alfanumérico y es escogido por el usuario. Una etiqueta está asignada a una dirección o posición de memoria, que contiene una instrucción. De esta manera, las etiquetas se ponen en el programa como reemplazo de las posiciones de memoria, apuntando de esta forma a esa posición donde se encuentra la etiqueta. Facilita el programa para los saltos y establece nombres de rutinas y tablas, entre otros.

```
NOTA: Es más fácil trabajar con nombres alfanuméricos escogidos por
el usuario para la construcción de su propio programa, que recordar
posiciones de memoria.
```

En la Figura 11.10 vemos que las etiquetas Start y MainLoop son label asociadas con las posiciones de memoria 0000h y 0003h que tienen asociadas las instrucciones 1683h bsf STATUS, RP0 y 1606h bsf PORTB,4. De esta manera se hace más fácil recordar estas posiciones con nombres asociados con nuestro programa. Es más descriptivo y útil escribir goto Start, que goto 0000h.

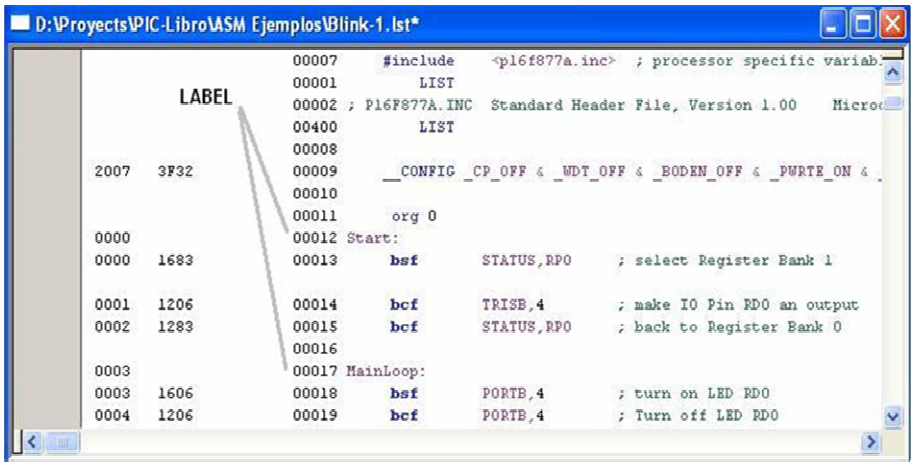


Figura 11.10 Campo *label* o etiqueta.

La etiqueta puede aparecer sola en la línea de programa, sin necesidad de instrucción o seguida de ella. Hay ciertas normas que establece el editor de texto del MPLAB; algunas reglas básicas para el *label* o las etiquetas son las siguientes:

- Deben empezar por letras seguido de letras, números y el carácter "_" subrayado o guión bajo. Ejemplo: Ciclo_1.

- No se pueden insertar espacios o tabulaciones antes de la etiqueta, si no será considerada como una instrucción.

- No se pueden utilizar expresiones reservadas del ensamblador, tales como:

  - Instrucciones.
    Por ejemplo: "nop", "sleep", "goto", etcétera.

  - Nombres de registros especiales.
    Por ejemplo: "TMR0", "PCL", etcétera.

  - Nombre de cada uno de los bits de los registros especiales.
    Por ejemplo: "C", "Z", "DC", etcétera.

  - Directivas del propio ensamblador:
    Por ejemplo: "ORG", "IF", "DB", etcétera.

- **OPCODE**

El segundo campo es el **opcode** o código de operación, que finalmente es el mnemónico de la instrucción. Algunos ejemplos de instrucciones son:

```
instrucción              opcode

addwf  f,d               addwf  f,d
call   k                 call   k
clrwdt                   clrwdt
nop                      nop
```

El opcode puede ser a veces una instrucción sin operandos o sin etiqueta. Aquí el set de instrucciones da la sintaxis, pues en este campo van sólo mnemónicos y deben estar tal como están escritos.

- **OPERANDOS**

Es el campo tercero e indica el o los operandos que necesita la instrucción (a veces hay instrucciones que no tienen operandos: nop, sleep, clrwdt). Según la instrucción, los operandos pueden ser de varios tipos:

```
• Orientada a bit     bsf f,b
                      donde          f=(7 bit, 0-127d, 0-7Fh)
                                     b=(3bit, 0-7)
• Orientada a byte    addwf f,d
                      donde          f=(7 bit, 0-127d, 0-7Fh)
                                     d=destino      d=1, destino f
                                                    d=0, destino W
• Literal      andlw k
                      donde          k=(8bit, 0-255d, 0-7Fh)
               goto k
                      k=(11bit, 0-2047d, 0-7FFh)
```

Cuando los registros o las literales sean cambiados por variables o por constantes, éstas deben estar previamente definidas al inicio del programa. También pueden estar en el archivo de definiciones *.inc. Ejemplos de algunas instrucciones que contienen registros especiales (SFR) (estos registros están previamente definidos en el archivo "P18F877A.inc") son:

```
btfss STATUS,Z          ; aquí STATUS=03h, Z=2h
addwf PCL,F             ; aquí PCL=02h, F=1
```

Ejemplos de algunas instrucciones que contienen registros de usuario (GPR) (estos registros son de usuario y se deben definir al principio del programa) son:

```
      cblock 0X20       ;define un bloque de registros que inicia en la dirección 0X20
contador :1             ;1 byte, posición 20h
nivel:1                 ;1 byte, posición 21h
      endc
      decfsz contador,F ;disminuye el valor del registro de usuario
                        ;definido como "contador" en la posición 20h
      subwf nivel, W    ;resta del registro "nivel" el valor en W y el resultado
                        ;lo guarda en W. El registro "nivel" en la posición 21h
```

- **COMENTARIO**

Éste es el último campo, su objetivo es hacer un comentario acerca de la instrucción, esto es de vital ayuda para el desarrollador, mediante estos comentarios se puede tener una noción más clara de las instrucciones y deducir el sentido del programa. También ayuda a realizar el mantenimiento y la actualización del programa, haciendo más fácil la identificación de las rutinas y facilitando las modificaciones o actualizaciones. El compilador, luego de encontrar un ";", asume un comentario, toma los siguientes caracteres y no los altera, respetando el comentario escrito, puede estar en cualquier parte de la línea, y puede ser de varias líneas, todas comenzando con ";".

NOTA: Un truco para depuración es comentar la instrucción (agregando ";" al principio de la línea), de tal forma que la instrucción no se compila y no se borra y queda como comentario; luego para agregarla como instrucción al programa nuevamente se quita el ";".

Éste es uno de los mecanismos más eficientes para la documentación de programas, es muy recomendable ampliar los comentarios para un fácil entendimiento. Cada uno de los cuatro campos (etiqueta, instrucción u opcode, operandos y comentarios) puede ser opcional, porque no necesariamente deben existir todos en una línea de programa.

11.2.3 TIPOS DE ARCHIVOS

Al trabajar con los editores de texto del ambiente de desarrollo (MPLAB), éste genera archivos adicionales que tienen diferentes formatos y son necesarios para su operación interna. MPLAB genera diferentes archivos, dependiendo del modo en que está trabajando; éstos los veremos más adelante en el Capítulo 14.0, Sección 14.2:"Pasos para crear un proyecto en MPLAB". A continuación veremos los tres tipos de archivos más relevantes:

- *.ASM, archivo *assembler*, contiene el código fuente, las instrucciones en mnemónicos, a partir de éste se generan los otros dos.

- *.LST, archivo listado, es el archivo *assembler* compilado, con los códigos hexadecimales para cada instrucción, con la numeración de líneas de instrucción y líneas de comentarios; además de comentarios de compilación y errores, también contiene todas las definiciones de términos.

- *.HEX, archivo hexadecimal, que resulta de compilar el archivo *assembler* o código fuente. Tiene un formato adecuado para transmisión de datos.

Este archivo se encuentra en nuestra página web ubicada en http://virtual.alfaomega.com.mx en la carpeta: \Ejemplos_ASM\7_segmentos_1B.

- **ARCHIVO * ASM**

Este archivo (Figura 11.11) que contiene el programa en sí, o de código fuente, es un archivo que está en la PC y que contiene todas las instrucciones con los campos adecuados, y con la sintaxis permitida, así como también con los comentarios pertinentes a cada instrucción. Por tanto, aquí es adonde se debe recurrir para hacer cambios o actualizaciones en las labores de mantenimiento de software (se puede escribir en cualquier editor de textos y salvarse en un archivo plano, o sea sin formato). El tamaño de este archivo es pequeño, 2.68 Kbytes, mide el archivo del 7_segmentos-1B.asm, que es el ejemplo que se vio en la Figura 11.10, y su contenido es sólo texto.

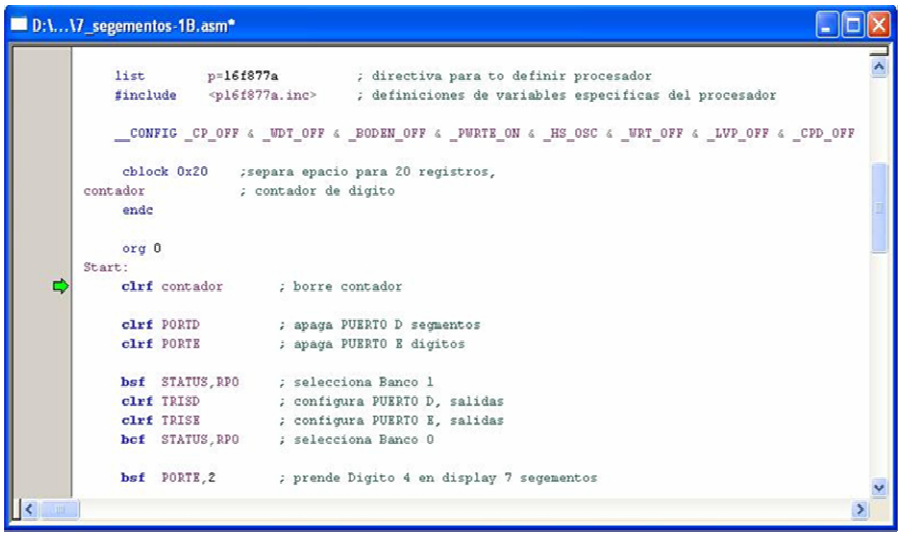


Figura 11.11 Archivos *.ASM.

- **ARCHIVO * LST**

Este archivo, derivado del archivo *assembler* o *.asm, contiene los mismos campos, más el código en hexadecimal de cada instrucción y el número de línea de compilación y el número de línea de texto. También contiene una lista de todos los *label* con su posición correspondiente; incluso, realiza las definiciones que se hallan agregado como archivos *.inc. Este archivo es más utilizado como documentación y también para verificar la compilación. El tamaño de este archivo es el más grande de todos los archivos generados por MPLAB; para el ejemplo del 7_segmentos.lst su tamaño es de 19.2 Kbytes. En la Figura 11.12 vemos todos los campos del archivo *.LST con su nombre correspondiente. Observamos que tiene tres campos más que el *assembler*, y éstos son:

- Posición de memoria de cada instrucción en el microcontrolador
- Código instrucción en hexadecimal (14 bits, max. 3FFFh)
- Núm. línea texto es el consecutivo de las líneas de texto en el programa

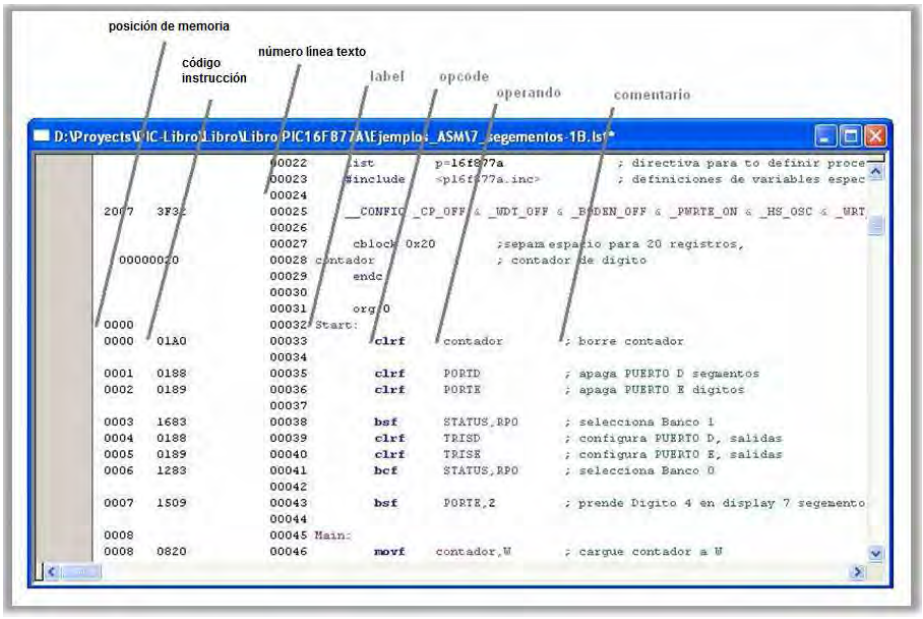


Figura 11.12 Archivos *.LST.

- **ARCHIVO * HEX**

Este archivo es el utilizado para programar el microcontrolador; es generado a partir del archivo código fuente o assembler en el proceso de compilación. Debe contener todas las instrucciones del programa en orden secuencial y codificadas en formato hexadecimal, además empaquetadas en un formato de tramas para transmisión desde la PC al MCU a través del programador/depurador. En la parte inferior de la Figura 11.13 apreciamos el código hexadecimal de todo el programa ejemplo 7_segmentos.asm y en la parte superior están las tramas del archivo, que se corresponden con los contenidos de las posiciones de memoria.

El formato que el compilador MPLAB utiliza para este microcontrolador como salida es INHX8M, la trama aparece explicada en el encabezado de la Figura 11.13. A modo de ejemplo, se verifica sólo la segunda fila. Formato INHX8M, sintaxis y explicación de cada campo:

```
BBAAAATTHHHH....HHHCC  -> trama

BB              No. bytes en la fila, NO CHKSUM
AAAA            Dirección inicial
TT              00-siempre, 01-fila final
HH              Datos: primero-low-LSB y segundo-high-MSB
CC              Checksum todos los anteriores de la fila
                complemento a 2 de 1 byte
```

Si observamos la trama de la segunda fila, tenemos detalladamente la fila siguiente y sus celdas:

BB	AAAA	TT	HHHH	HHHH	HHHH	HHHH	HHHH	HHHH	HHHH	HHHH	CC
10	0000	00	A001	8801	8901	8316	8801	8901	8312	0915	DD

Siguiendo la lectura de los correspondientes campos de la trama podemos deducir que son 16d o 10h pares de bytes antes de *cheksum*, comenzando en la posición 0000h; todos los datos están en parejas de la forma LSB MSB; así, la primera pareja de datos A001, es la primera instrucción 01A0, y la última pareja de datos de la fila es 0915, que es la instrucción 1509, el cheksum es de 1 byte, y es el complemento a 2 de la suma de todos los bytes anteriores.

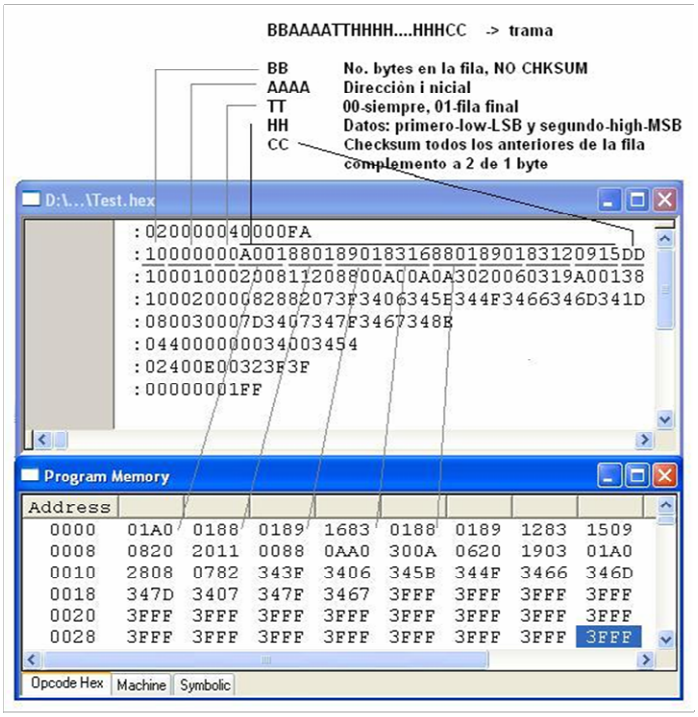


Figura 11.13 Archivos *.LST.

Observamos que los datos están relacionados entre el archivo *.hex y la memoria de programa. Al final los datos deben quedar en el microcontrolador como se muestra en la parte baja de la Figura 11.13, o memoria de programa.

EVALUACIÓN

```
P.:1. Qué son los mnemónicos?
R:
Son las instrucciones con las que se elabora un programa; cada una
de éstas tiene un código de máquina asociado.

P.:2. Los microcontroladores de la serie 10/12/16 marca Microchip
¿cuántos mnemónicos tienen en su set de instrucciones?
R:
Tienen 35 mnemónicos o instrucciones.

P.:3. En la instrucción:
Start: clrf contador ; apague contador
Defina: Label, Instrucción, Operando, Comentario.
R:
Label        Start:
Instrucción  clrf
Operanco     contador
Comentario   ;apague contador

P.:4. ¿Con cuál de estos archivos se puede programar un MCU?
a)  *.asm ,   b)  *.hex ,  c)  *.lst
R:
b) *.hex
```

11.3 SET DE INSTRUCCIONES

Este set es el listado de instrucciones de los PIC micros; son sólo 35 instrucciones y con ellas se pueden programar todos los micros de las familias **PIC 10/12/16**. Ésta es una de las grandes fortalezas de **Microchip** con respecto a los demás fabricantes de micros. El usuario, con aprenderse 35 instrucciones, puede manejar una amplia variedad de micros con diferentes especificaciones de *Pinout,* módulos periféricos y tamaños de memoria. Estas instrucciones se pueden clasificar de diferentes formas; se presentan dos: tipo de operando (bit o byte o literal) y el tipo de operación.

- **SEGÚN EL TIPO DE OPERANDO**

Éste es el orden en que se presentan en el *Datasheet* o pdf original, como aparece en las Tablas 11.4 y 11.5. Para las tablas 11.4 y 11.5, aplican las definiciones referidas en ellas mismas. En la tabla 11.5 vemos que los dos primeros bits de cada instrucción (en la casilla 14-Bit opcode) son los que marcan la diferencia entre los grupos. En esta clasificación encontramos las instrucciones orientadas al tipo de operando:

- Operaciones con registros orientados a byte (00).
- Operaciones con registros orientados a bit (01).
- Operaciones con registros orientados a literales y control (11, 10, 00).

Tabla 11.4 Definiciones para set de instrucciones.

Letra	Descripción
W	Acumulador
f	Registro de RAM (0-7Fh)
d	Destino d=0 destino (W) d=1 destino (f)
b	bit (0-7)
k	Literal (0-FFh) Label (0-2047)
x	no importa (0-1)

Letra	Banderas
Z	Zero
DC	Decimal Carry
C	Carry
TO	Time-Out
PD	Power Down

Tabla 11.5 Set de instrucciones por tipo de operando.

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add Literal and W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	k	AND Literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to Address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR Literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move Literal to W	1	11 00xx kkkk kkkk			
RETFIE	-	Return from Interrupt	2	00 0000 0000 1001			
RETLW	k	Return with Literal in W	2	11 01xx kkkk kkkk			
RETURN	-	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011		$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from Literal	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	k	Exclusive OR Literal with W	1	11 1010 kkkk kkkk		Z	

Note 1: When an I/O register is modified as a function of itself (e.g., MOVF PORTB, 1), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

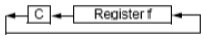
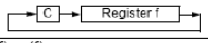
- **SEGÚN EL TIPO DE OPERACIÓN**

Ésta es una clasificación orientada al tipo de instrucción o de operación (Tabla 11.6), que se puede hacer en los grupos siguientes:

— Instrucciones de carga.
— Instrucciones aritméticas.

- Instrucciones lógicas.
- Instrucciones de bit.
- Instrucciones de salto
- Instrucciones de subrutinas e interrupciones.
- Instrucciones especiales.

Tabla 11.6 Set de instrucciones por tipo de operación.

TIPO	MNEMONICO	op	DSCRIPCION	OPCODE	FLAGS	#
CARGA	clrf f	f	00 -> (f)	0 0 0 0 0 1 1 f f f f f f f	Z	1
	clrw		00 -> (W)	0 0 0 0 0 1 0 x x x x x x x	Z	2
	movf f,d	f,d	(f) -> (destino)	0 0 1 0 0 0 d f f f f f f f	Z	3
	movlw k	k	k -> (W)	1 1 0 0 x x k k k k k k k k	Ninguno	4
	movwf f	f	(W) -> (f)	0 0 0 0 0 0 1 f f f f f f f	Ninguno	5
ARITMETICAS	addlw k	k	(W) + k -> (W)	1 1 1 1 x k k k k k k k k	C, DC, Z	6
	addwf f,d	f,d	(W) + (f) -> (f)	0 0 0 1 0 0 d f f f f f f f	C, DC, Z	7
	sublw k	k	k – (W) -> (W)	1 1 1 1 x k k k k k k k k	C, DC, Z	8
	subwf f,d	f,d	(f) – (W) -> (f)	0 0 0 0 1 0 d f f f f f f f	C, DC, Z	9
	decf f,d	f,d	(f) - 1 -> (f)	0 0 0 0 1 1 d f f f f f f f	Z	10
	incf f,d	f,d	(f) + 1 -> (f)	0 0 1 0 1 0 d f f f f f f f	Z	11
LOGICAS	andlw k	k	(W) AND k -> (W)	1 1 1 0 0 1 k k k k k k k k	Z	12
	andwf f,d	f,d	(W) AND (f) -> (f)	0 0 0 1 0 1 d f f f f f f f	Z	13
	iorlw k	k	(W) OR k -> (W)	1 1 1 0 0 0 k k k k k k k k	Z	14
	iorwf f,d	f,d	(W) OR (f) -> (f)	0 0 0 1 0 0 d f f f f f f f	Z	15
	xorlw k	k	(W) XOR k -> (W)	1 1 1 0 1 0 k k k k k k k k	Z	16
	xorwf f,d	f,d	(W) XOR (f) -> (f)	0 0 0 1 1 0 d f f f f f f f	Z	17
	rlf f,d	f,d		0 0 1 1 0 1 d f f f f f f f	C	18
	rrf f,d	f,d		0 0 1 1 0 0 d f f f f f f f	C	19
	comf f,d	f,d	(-f) ->(f)	0 0 1 0 0 1 d f f f f f f f	Z	20
	swapf f,d	f,d	nible (7:4) <-> (3:0) -> (f)	0 0 1 1 1 0 d f f f f f f f	Ninguno	21
BIT	bcf f,b	f,b	0 -> b, (f<b>)	0 1 0 0 b b b f f f f f f f	Ninguno	22
	bsf f,b	f,b	1 -> b, (f<b>)	0 1 0 1 b b b f f f f f f f	Ninguno	23
SALTO	btfsc f,b	f,b	Salta si (b)=0 en (f<b>)	0 1 1 0 b b b f f f f f f f	Ninguno	24
	btfss f,b	f,b	Salta si (b)=1 en (f<b>)	0 1 1 1 b b b f f f f f f f	Ninguno	25
	decfsz f,d	f,d	(f)-1 -> (d), salta si es 0	0 0 1 0 1 1 d f f f f f f f	Ninguno	26
	incfsz f,d	f,d	(f)+1 -> (d), salta si es 0	0 0 1 1 1 1 d f f f f f f f	Ninguno	27
	goto k	k	Salta a la dirección 'k'	1 0 1 k k k k k k k k k k k	Ninguno	28
SUB. INT.	call k	k	Llamada a subrutina	1 0 0 k k k k k k k k k k k	Ninguno	29
	retfie		Retorno de una interrupción	0 0 0 0 0 0 0 0 0 0 0 1 0 0 1	Ninguno	30
	retlw k	k	Retorno con un literal en W	1 1 0 1 x x k k k k k k k k	Ninguno	31
	return		Retorno de una subrutina	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	Ninguno	32
ESPE-CIAL	nop		No operación	0 0 0 0 0 0 0 0 0 0 0 x x	Ninguno	33
	clrwdt		Borra Timer del Watchdog	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	/TO, /PD	34
	sleep		Entra en modo bajo consumo	0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	/TO, /PD	35

A continuación se detallan cada una de las instrucciones, ordenadas según el tipo de operación; también es importante recordar que en el Apéndice B: "Set de instrucciones", está el contenido ampliado de cada instrucción.

```
NOTA: El compilador de MPLAB asume como igual las instrucciones
escritas en mayúsculas o minúsculas. addlw f,d y ADDLW F,D, son
iguales.
```

- **BANDERAS O FLAGS**

Luego de ejecutadas, las instrucciones pueden alterar el estado de las flags o banderas, que son los bits del registro STATUS. Estas banderas operan así:

- Z (Zero). Este bit se pone en "1" si el resultado de la operación es cero (b"0000 0000" o 0×00 o 0 decimal), y se pone Z en "0" con cualquier otro valor.
- C (Carry). Este bit se pone en "1", si en la última operación en el resultado hay un acarreo del bit 7 (MSB); es decir, si el resultado es mayor que (b`1111 1111` o 0×FF o 255 decimal). Se pone en "0" si no hay acarreo del bit 7. (Opera con las instrucciones ADDWF, ADDLW, SUBLW, SUBWF.)
- DC (Digit Carry). Este bit se pone en "1", si hay un acarreo del bit 3 al 4. En otro caso se pone a "0". Al sumar dos bytes (sus nibbles menores <3:0>), si la suma excede a Fh, genera DC. (Opera con las instrucciones ADDWF, ADDLW, SUBLW, SUBWF).

```
NOTA: Nibbles son las dos medias partes de un byte.
Los dos nibles son:
nibble High o nibble MSB <7:4>
nibble Low o nibble LSB <3:0>.
```

11.3.1 INSTRUCCIONES DE CARGA

Éstas son las instrucciones básicas de transferencia de datos entre los registros RAM y el registro W, en todas sus variantes. Se transfieren datos desde una fuente (f) a un destino (d) [d=0 registro (W), d=1 registro (f)]. La fuente puede también ser una constante, cuyo valor (k) es de 8 bits (bit 7.0), que podemos cargar en (W). Se incluyen las instrucciones de borrado, pues carga el valor 00 a (W) o (f). Las instrucciones de carga son:

clrw (clear W)
Borra el contenido del registro (W)
Operación 00 -> (W) Z=1 (bandera de 0)

```
Ejemplo:        clrw                    0 -> (W)
                Antes instrucción:      (W) = 0X7A y Z = 0
                Después instrucción:    (W) = 0X00 y Z = 1
```

clrf (clear f)
Borra el contenido del registro (f), f (0-127d, 0-7Fh) registro en RAM
Operación 00 -> (f) Z=1 (bandera de 0)

```
Ejemplo:        clrf contador           0 -> (contador)
                Antes instrucción:      (contador) = 0×B6 y Z = 0
                Después instrucción:    (contador) = 0×00 y Z = 1
```

movlw k (move Literal to W)
Cargue la literal o constante k a registro (W), k=8 bits (0-255d o 0-FFh)
Operación k -> (W) no afecta las banderas

```
Ejemplo:        movlw 0×C0              C0h -> (W)
                Antes instrucción:      (W) = A9h
                Después instrucción:    (W) = 0×C0
```

movf f,d (move f)
Cargue el contenido del registro (f) en el registro destino (d).
Si d=0, el destino es el registro (W)
Si d=1, el destino es el mismo registro (f).
Operación (f) -> (d) si carga=0, Z=1 (bandera de cero Z)

```
Ejemplo:  movf nivel,0            (nivel) -> (W)
          Antes instrucción:      (nivel) = 0×8E, (W) = 0×37 y Z = 0
          Después instrucción:    (nivel) = 0×8E, (W) = 0×8E y Z = 0
```

movwf f (move W to f)
Carga el contenido del registro (W) al registro (f).
Operación (W) -> (f) no afecta las banderas

```
Ejemplo:        movwf PORTD             (W) -> (PORTD)
                Antes instrucción:      (PORTD) = 0xA5 y (W) = 0x72
                Después instrucción:    (PORTD) = 0x72 y (W) = 0x72
```

11.3.2 INSTRUCCIONES ARITMÉTICAS

Las operaciones aritméticas básicas que podemos realizar son: sumar, restar, incrementar y disminuir.

- **ADICIÓN O SUMA**

Repasando la operación de suma, al observar el ejemplo 1 de suma, tenemos:

```
Ejemplo 1:
    200 (decimal)          1100 1000 (binario) C8 (hex.)
  + 29 (decimal)         + 0001 1101 (binario) 1D (hex.)      C/DC/Z
    229 (decimal)          1110 0101 (binario) E5 (hex.)      0/ 1/ 0
```

La operación es menor que 255. No hay *C* (*Carry*) la suma fue menor que 255d o FFh o b“1111 1111”; sin embargo, sí hay *DC* (*Digit-Carry*) del bit 3 al 4, al sumar los nibble LSB 8h+Dh=15h. No hay bandera *Z* (*Zero*), pues el resultado no es 0.

Si observamos el ejemplo 2 de suma, tenemos:

```
Ejemplo 2:
    178 (decimal)          1011 0010 (binario)  B2 (hex.)
  + 90 (decimal)         + 0101 1010 (binario)  5A (hex.)     C/DC/Z
    268 (decimal)        1 0000 1101 (binario) 10C (hex.)     1/ 0/ 0
```

La operación es mayor que 255. Si hay *C* (*Carry*) la suma fue mayor que **255d** o **FFh** o **b“1111 1111”**, no hay *DC* (*Digit-Carry*). No hay bandera *Z* (*Zero*), pues el resultado no es 0. Con estos dos ejemplos se nota cómo funciona el acarreo (C) y acarreo de dígito (DC). A continuación damos las dos instrucciones de suma:

addlw k (add Literal to W)

Suma el contenido del registro (W) con la literal o constante ‘k’ y almacena el resultado en (W). k=8 bits, (**0-255d o 0-FFh**). Si hay acarreo *C* (*Carry*) se pone a “1”.

Operación (W) + k -> (W) afecta las banderas Z, DC, C

```
Ejemplo 1:
        addlw 0×A2              (W) + A2h -> (W)
        Antes instrucción:      (W) = 0×25, y C = -
        Después instrucción:    (W) = 0×C7, y C = 0
        (0×A2 + 0×25 = 0×C7)

Ejemplo 2:
        addlw 0×79              (W) + 79h -> (W)
        Antes instrucción:      (W) = 0×98, y C = -
        Después instrucción:    (W) = 0×11, y C = 1
        (0×79 + 0×98 = 0×11 más acarreo)
```

addwf f,d (add W and f)

Suma el contenido del registro (W) al contenido del registro (f) y almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)
Si hay acarreo *C* (*Carry*) se pone a "1".

Operación (W) + (f) -> (d) afecta las banderas Z, DC, **C:**

```
Ejemplo 1:   addwf   nivel,W          (nivel) + (W) -> (W)
             Antes instrucción:       (W)= 0x29,(nivel)=0xB1,y C = -
             Después instrucción:     (W)= 0xDA,(nivel)=0xB1,y C = 0
             (0x29 + 0xB1 = 0xDA)
```

```
Ejemplo 2:   addwf contador,F         (contador) + (W) -> (Contador)
             Antes instrucción:       (W)=0xE4,(contador)=0x38,y C = -
             Después instrucción:     (W)=0xE4,(contador)=0x1C,y C = 1
             (0xE4 + 0x38 = 1C más acarreo)
```

- **SUBSTRACCIÓN O RESTA**

Para realizar la resta se hace un truco que consiste en complementar a dos el *sustraendo* y sumarlo al *minuendo*; así obtenemos la *diferencia*. (Complementar a dos es igual que hallar el negativo del número):

```
minuendo     0x7A                          0x7A
sustraendo-  0x42    complemento a 2 ->   + 0xBE
diferencia   0x38                          0x138 (=0x38 positivo)
```

El complemento a 2 consiste en hacer el complemento a 0xFF +1, así:

- complemento a 0xFF
- operando 0x42
- resultado 0xBD + 1 = 0xBE

Conviene anotar que al realizar la resta de 8 bits puede haber tres casos, dependiendo del resultado, así:

- Positivo mayor que cero C = 1 y Z = 0.
- Positivo igual a cero C = 1 y Z = 1.
- Negativo menor que cero C = 0 y Z = 0.

A continuación veremos las dos instrucciones de resta:

sublw k (subtract W from literal)

Resta de la constante 'k' el contenido del registro (W) y almacena el resultado en W.

Operación k - (W) -> (W) afecta las banderas Z, DC, **C:**

```
Ejemplo 1:      sublw  0x03          03h – (W) -> (W)
                Antes instrucción:   (W) = 0x02, C = - y Z = -
                Después instrucción: (W) = 0x01, C = 1 y Z = 0
                Positivo mayor que cero

Ejemplo 2:      sublw  0x02          02h – (W) -> (W
                Antes instrucción:   (W) = 0x02, C = - y Z = -
                Después instrucción: (W) = 0x00, C = 1 y Z = 1
                Positivo igual a cero

Ejemplo 3:      sublw  0x01          01h – (W) -> (W)
                Antes instrucción:   (W) = 0x02, C = - y Z = -
                Después instrucción: (W) = 0xFF, C = 0 y Z = 0
                Negativo menos que cero
```

subwf f,d (subtract W from f)

Resta del contenido del registro (f) el contenido del registro (W), almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación (f) – (W) -> (d) afecta las banderas Z, DC, **C:**

```
Ejemplo 1:
                subwf   tempo,F      (tempo) – (W) -> (tempo)
                Antes instrucción:   (tempo) = 0x11,(W)=0x10,C = - y Z = -
                Después instrucción: (tempo) = 0x01,(W)=0x10,C = 1 y Z = 0
                Positivo mayor que cero

Ejemplo 2:
                subwf   tempo,F      (tempo) – (W) -> (Tempo)
```

```
Antes instrucción:    (tempo) = 0x10,(W)=0x10,C = - y Z = -
Después instrucción:  (tempo) = 0x00,(W)=0x10,C = 1 y Z = 1
Positivo igual a cero
```

```
Ejemplo 3:
          subwf   tempo,F       (tempo) – (W) -> (Tempo)
Antes instrucción:    (tempo) = 0x0F,(W)=0x10,C = - y Z = -
Después instrucción:  (tempo) = 0xFF,(W)=0x10,C = 0 y Z = 0
Negativo menor que cero
```

- **INCREMENTAR Y DISMINUIR**

Las últimas instrucciones aritméticas son las siguientes:

decf f,d (decrement f)

Disminuye el contenido del registro (f) en una unidad, almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación (f) – 1 -> (d) afecta la bandera Z:

```
Ejemplo 1:
          decf    nciclos,F     (nciclos) – 1 -> (nciclos)
Antes instrucción:    (nciclos) = 0x01 y Z = -
Después instrucción:  (nciclos) = 0x00 y Z = 1
```

```
Ejemplo 2:
          decf    contador,W    (contador) – 1 -> (W)
Antes instrucción:    (contador) = 0xE0, (W) = - y Z = -
Después instrucción:  (contador) = 0xE0, (W) = 0xDF y Z = 0
```

incf f,d (increment f).

Incrementa el contenido del registro (f) en una unidad, almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación (f) + 1 -> (d) afecta la bandera Z:

```
Ejemplo 1:
          incf    nivel,F       (nivel) + 1 -> (nivel)
```

```
                Antes instrucción:    (nivel) = 0xFF y Z = -
                Después instrucción:  (nivel) = 0x00 y Z = 1

Ejemplo 2:
                incf   ncajas,W       (ncajas) + 1 -> (W)
                Antes instrucción:    (ncajas) = 0x99, (W) = - y Z = -
                Después instrucción:  (ncajas) = 0x99, (W) = 0x9A y Z = 0
```

11.3.3 INSTRUCCIONES LÓGICAS

Las operaciones lógicas que se pueden realizar son la AND, OR inclusive. OR exclusiva, la rotación, complemento y el intercambio de nibbles:

- AND

Esta operación se simboliza con (.) como (A **.** B) o (A **y** B), ambos deben ser verdaderos, para que la operación sea verdadera (también simbolizada como &).

andlw k (AND Literal with W)

Operación AND lógica entre el contenido del registro (W) y la constante 'k', almacena el resultado en (W).

Operación (W) AND k -> (W) afecta la bandera Z:

```
Ejemplo:        andlw  b'0000 1111'        (W) AND  b'0000 1111' -> (W)
                Antes instrucción:         (W) = b'1010 1010' y Z = -
                Después instrucción:       (W) = b'0000 1010' y Z = 0
```

andwf f,d (AND W with f)

Operación AND lógica entre el contenido del registro (W) y el contenido del registro (f), almacena el resultado según (d).
Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación (W) AND (f) -> (d) afecta la bandera Z:

```
Ejemplo 1:      andwf   PORTE,F       (W) AND (PORTE) -> (PORTE)
                Antes instrucción:    (W) = b'1111 0000',
                                      (PORTE) = b'1100 1010' y Z = -
                Después instrucción:  (W) = b'1111 0000',
                                      (PORTE) = b'1100 0000' y Z = 0

Ejemplo 2:      andwf   PORTE,W       (W) AND (PORTE) -> (W)
                Antes instrucción:    (W) =  b'1111 0000',
```

```
                                (PORTE) = b'1100 1010' y Z = -
Después instrucción:            (W) =  b'1100 0000',
                                (PORTE) = b'1100 1010' y Z = 0
```

- OR (OR INCLUSIVA)

Esta operación se simboliza con (**+**) como (A + B) o (A **o** B), cualquiera debe ser verdadero, para que la operación sea verdadera (también simbolizada como |).

iorlw k (inclusive OR Literal with W)

Operación OR lógica entre el contenido del registro (W) y la constante 'k', almacena el resultado en (W).

Operación (W) OR k -> (W) afecta la bandera Z:

```
Ejemplo:        iorlw  b'1111 0000'.        (W) OR  b'1111 0000' -> (W)
                Antes instrucción:          (W) = b'1001 1010' y Z = -
                Después instrucción:        (W) = b'1111 1010' y Z = 0
```

iorwf f,d (inclusive OR W with f)

Operación OR lógica entre el contenido del registro (W) y el contenido del registro (f), almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación (W) OR (f) -> (d) afecta la bandera Z:

```
Ejemplo 1:      iorwf   nivel,F         (W) OR (nivel) -> (nivel)
                Antes instrucción:      (W) = b'0000 0111',
                                       (nivel) = b'1100 1010' y Z = -
                Después instrucción:    (W) =  b'0000 0111',
                                       (nivel) = b'1100 1111' y Z = 0

Ejemplo:        iorwf   PORTC,W         (W) OR (PORTC) -> (W)
                Antes instrucción:      (PORTC) = b'0111 1010' ,
                                                (W) =  b'0011 0011' y Z = -
                Después instrucción:            (PORTC) = b'0111 1010' ,
                                                (W) =  b'0111 1011' y Z = 0
```

- OREX (OR EXCLUSIVA)

Esta operación se simboliza con (⊕) como (A ⊕ B), uno u otro (no ambos) debe ser verdadero, para que la operación sea verdadera.

xorlw k (exclusive OR Literal with W)

Operación OR-Exclusiva entre el contenido del registro (W) y la constante 'k', el resultado se almacena en (W).

Operación (W) ⊕ k -> (W) afecta solamente la bandera Z:

```
Ejemplo:        xorlw  b'0011 1100'   (W) XOR b'0011 1100' -> (W)
                Antes instrucción:    (W) =  b'1011 0101' y Z = -
                Después instrucción:  (W) =  b'1000 1001' y Z = 0
```

xorwf f,d (exclusive OR W with f)

Operación OR-Exclusiva entre el contenido del registro (W) y el contenido del registro (f), almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación (W) ⊕ (f) -> (d) afecta la bandera Z:

```
Ejemplo 1:      xorwf  nivel,F        (W) XOR (nivel) -> (nivel)
                Antes instrucción:    (nivel) =     b'0001 1001',
                                      (W) =         b'1111 0000' y Z = -
                Después instrucción:  (nivel) =     b'1110 1001',
                                      (W) =         b'1111 0000' y Z = 0

Ejemplo 2:      xorwf  ncajas,W       (W) XOR (ncajas) -> (W)
                Antes instrucción:    (ncajas) =    b'0110 1010',
                                       (W) =        b'1111 0000' y Z = -
                Después instrucción:  (ncajas) =    b'0110 1010',
                                       (W) =        b'1001 1010' y Z = 0
```

- ROTACIONES

Las rotaciones (Figura 11.14) son desplazamientos de una posición o bit, con todo el registro de 8 bits a través de la bandera *C* (*Carry*). Puede ser rotación a la izquierda (*left*, rlf) o a la derecha (*right*, rrf). Es relevante poner el bit C en el valor deseado antes de la operación.

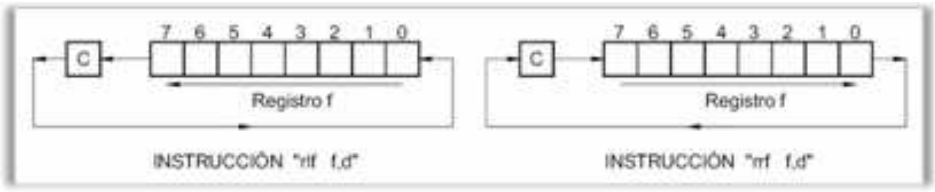


Figura 11.14 Gráfica de rotaciones.

rlf f,d (rotate left f through Carry)

Rote a izquierda el registro (f) a través de *C* (*Carry*), almacena el resultado según (d) (Figura 14.15).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación <<(f) -> (d) afecta la bandera **C:**

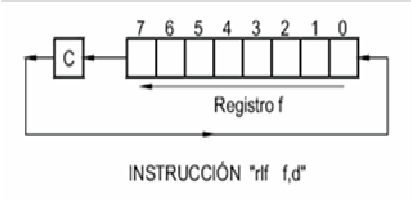


Figura 11.15 Rote a la izquierda a través de carry.

```
Ejemplo 1:      rlf     PORTD,W         <<(PORTD) -> (W)
                Antes instrucción:      (PORTD) = b'0011 1100',W = - y C = 1.
                Después instrucción:    (PORTD) = b'0011 1100'
                                         (W)=     b'0111 1001'y C = 0.

Ejemplo 2:      rlf     nivel,F         <<( nivel) -> (nivel)
                Antes instrucción:      (nivel) = b'0001 1111' y C = 0.
                Después instrucción:    (nivel) = b'0011 1110' y C = 0.
```

rrf f,d (rotate right f through Carry)

Rote a derecha el registro (f) (Figura 11.16) a través de *C* (*Carry*), almacena el resultado según (d).

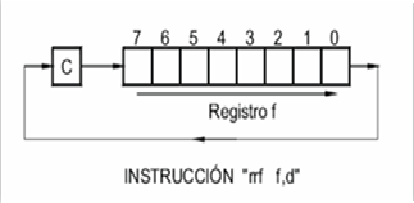


Figura 11.16 Rote a la derecha a través de carry.

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación >>(f) -> (d) afecta la bandera **C:**

```
Ejemplo 1:    rrf    PORTB,W        >>(PORTB) -> (W)
              Antes instrucción:    (PORTB) = b'1100 0011',(W)= - y C = 0
              Después instrucción:  (PORTB) = b'1100 0011',
                                    (W) =    b'0110 0001'y C = 1

Ejemplo 2:    rrf    contador,F     >>(contador) -> (contador)
              Antes instrucción:    (contador) = b'0011 1100', y C = 1
              Después instrucción:  (contador) = b'1001 1110', y C = 0
```

- COMPLEMENTO E INTERCAMBIO DE NIBBLES

Otras dos operaciones lógicas son: El complemento que consiste en cambiar los "1" por "0" y viceversa. El intercambio de nibbles que consiste en intercambiar nibbles (7:4) y (3:0).

comf f,d (complement f)

Complementa el contenido del registro $(\bar{f})$, almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro $(\bar{f})$

Operación $(\bar{f})$-> (d) afecta la bandera Z:

```
Ejemplo 1:    comf   nivel,F        (-nivel) -> (nivel)

              Antes instrucción:    (nivel) = b'1001 1101', y Z = -
              Después instrucción:  (nivel) = b'0110 0010', y Z = 0

Ejemplo 2:    comf   PORTC,W        (-PORTC) -> (W)
              Antes instrucción:    (PORTC) = b'0110 1001',(W)= - y Z = -
              Después instrucción:  (PORTC) = b'0110 1001',
                                    (W) =     b'1001 0110', y Z = 0
```

swapf f,d (swap nibbles in f)

Intercambia los nibbles del registro (f), almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación <->(f) -> (d) afecta la bandera Z:

```
    Ejemplo 1:
                swapf     STATUS,W      <->(f) -> (W)
                Antes instrucción:      (STATUS) = 0x05 y (W) = -
                Después instrucción:    (STATUS) = 0x05 y (W) = 0x50

    Ejemplo 2:
                swapf     dato_rx,F     <->(f) -> (f)
                Antes instrucción:      (dato_rx) = 0x3C
                Después instrucción:    (dato_rx) = 0xC3
```

11.3.4 INSTRUCCIONES DE BIT

Estas instrucciones ponen un bit (b) a nivel lógico "0" o "1", de un registro de la RAM o memoria de datos.

bcf f,b (bit clear f)

Pone a cero el bit (b) del registro (f).
Operación b=0 -> (f<b>) no afecta la bandera:

```
Ejemplo:        bcf    PORTB,DL1      ; 0 -> (PORTB,DL1), DL1 = bit 4
                Antes instrucción:    (PORTB) = b'0011 0100'
                Después instrucción:  (PORTB) = b'0010 0100'
```

bsf f,b (bit set f)

Pone a uno el bit (b) del registro (f).
Operación b=1 -> (f<b>) no afecta la bandera:

```
Ejemplo:        bsf    out_led,bit2   ; 1 -> (out_led,bit2), bit2 = bit 2
                Antes instrucción:    (out_led) = b'1111 1000'
                Después instrucción:  (out_led) = b'1111 1100''
```

NOTA: Recuerde que un byte son 8 bits, estos bits son de 0 a 7.

11.3.5 INSTRUCCIONES DE SALTO

Estas instrucciones permiten realizar saltos en el programa; tomando como base que se cumplan condiciones o no, se pueden realizar de las formas siguientes:

- En función de un bit, según el estado (1 o 0) de un bit en un registro.
- En función de un registro por disminución o incremento al llegar al valor 0×00.
- Incondicional, ejecuta la instrucción *goto k.*

- **SALTOS EN FUNCIÓN DE UN BIT**

Si la prueba (test) de un bit en un registro es verdadera, se realiza el salto (Figura 11-17). La prueba sobre el bit puede ser: clear (0) o set (1); hay dos instrucciones, una para cada opción "btfsc…" y "btfss…".

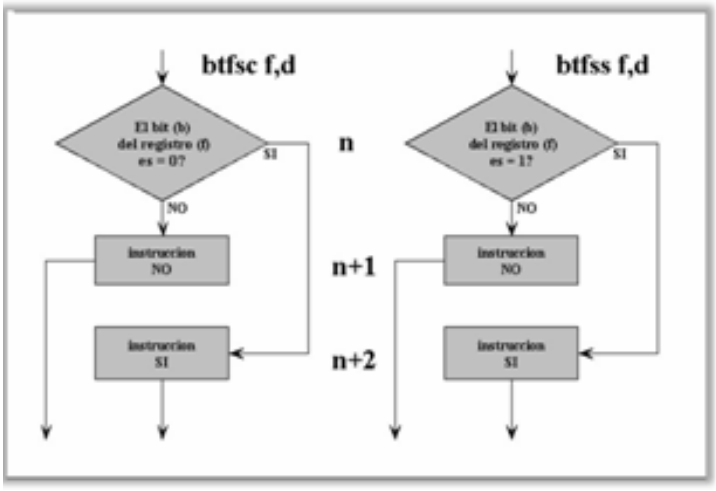


Figura 11.17 Gráfica saltos en función de bit.

btfsc f,b (bit test file, skip if clear)

Esta instrucción puede actuar de dos formas:

• Si el bit (b) del registro (f) es "0", salta la instrucción que sigue.

Recuerde *skip if clear* salta si = "0".

• Si el bit (b) del registro (f) es "1", ejecuta la instrucción que sigue.

Operación Salta si b=0 en (f<b>) no afecta las banderas:

```
Ejemplo:
n btfsc flag,4              ; Si el bit 4 del registro flag es
                            ; "0", salta a n+2
n+1 goto                    ; Respuesta NO
n+2 .......                 ; Respuesta SÍ, salta si "0"
```

```
En instrucción:       PC) = n
Después instrucción:  Si flag<4> = 0, (PC) = n+2 (salta Respuesta SÍ)
                      Si flag<4> = 1, (PC) = n+1 (Respuesta NO)
```

btfss f,b (bit test file, skip if set)

Esta instrucción puede actuar de dos formas:

• Si el bit (b) del registro (f) es "1", salta la instrucción que sigue.

Recuerde *skip if set* salta si = "1".

• Si el bit (b) del registro (f) es "0", ejecuta la instrucción que sigue.

Operación Salta si b=1 en (f<b>) no afecta las banderas:

```
Ejemplo:
n btfss flag,2          ; Si el bit 2 del registro flag es
                        ; "1", salta a n+2
n+1 goto                ; Respuesta NO
n+2 ......              ; Respuesta SÍ, salta si "1"
```

```
En instrucción:       PC) = n
Después instrucción:  Si flag<2> = 1, (PC) = n+2 (salta Respuesta SÍ)
                      Si flag<2> = 0, (PC) = n+1 (Respuesta NO)
```

- **SALTOS EN FUNCIÓN DE UN REGISTRO**

Estos saltos ocurren porque el registro llega al valor de cero, por decremento o incremento del registro (Figura 11.18), con las instrucciones "decfsz..." e "incfsz ...".

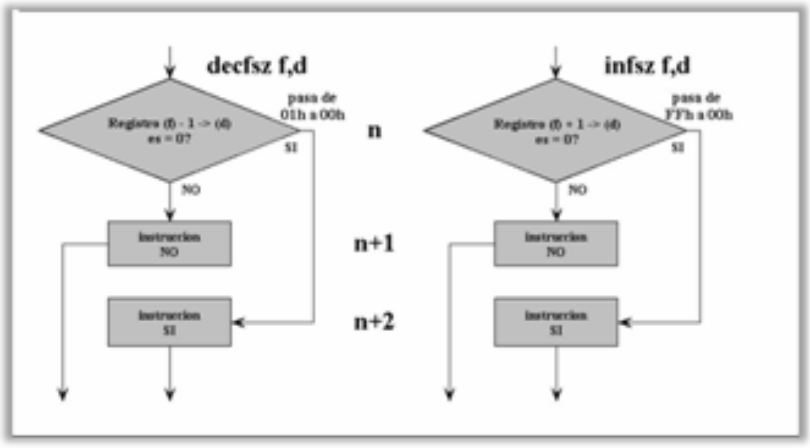


Figura 11.18 Gráfica saltos en función de registro.

decfsz f,d (decrement f, skip if zero)

Disminuye en una unidad el contenido del registro (f), almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación Salta si {(f)-1 -> (d)} = 0 no afecta las banderas:

```
Ejemplo:
n decfsz contador,F          ; Si el registro destino, contador =
                             ; 0? -> salta.
n+1 goto                     ; Respuesta NO
n+2 .......                  ; Respuesta SÍ, salta si llegó a 00
```

```
En instrucción:              (PC) = n
Después instrucción:         Si contador = 0, (PC) = n+2 (salta Respuesta
                             ;SÍ)
                             Si contador <> 0, (PC) = n+1 (Respuesta NO)
```

incfsz f,d (increment f, skip if zero)

Incrementa en una unidad el contenido del registro (f), almacena el resultado según (d).

Si d = 0, se guarda en el registro (W)
Si d = 1, se guarda en el registro (f)

Operación Salta si {(f)+1 -> (d)} = 0 no afecta las banderas:

```
Ejemplo:
n incfsz contador,W          ; Si el registro destino, (W) = 0 ? ->
                             ; salta.
n+1 goto                     ; Respuesta NO
n+2 .......                  ; Respuesta SÍ, salta si llegó a 00

En instrucción:              (PC) = n
Después instrucción:         Si (W) = 0, (PC) = n+2 (salta Respuesta SÍ)
                             Si (W) <> 0, (PC) = n+1 (Respuesta NO)
```

- **SALTO INCONDICIONAL**

Aquí no se analizan condiciones, es un salto sin preguntas.

goto k (goto k)

Vaya a la dirección donde apunta la label 'k'.

Operación k -> (PC) no afecta las banderas:

```
Ejemplo:
               goto Main_4    ; k -> (PC).          ; Posición 0×013A
               ……
Main_4 .......                ; (PC) salta acá      ; Posición 0×01C8

En instrucción:              (PC) = posición 0×013A
Después instrucción:         (PC) = posición Main_4 = 0×01C8
```

11.3.6 INSTRUCCIONES DE SUBRUTINAS E INTERRUPCIONES

Para entender estas instrucciones deben estar claros los conceptos de subrutinas e interrupciones.

NOTA: Los numerales 11.3.6 y 11.3.7 se refieren a procesos especiales del MCU; le recomendamos ampliar los conceptos en otras secciones del libro que contengan los temas de interrupciones y subrutinas.

call k (call k)

Llama la subrutina que está en la dirección donde apunta la label 'k'.

Operación **k -> (PC)** **no afecta las banderas:**

```
Ejemplo:
               Call BIN_BCD   ; k -> (PC).          ; Posición 0x01A8
               ……
BIN_BCD .......               ; (PC) salta acá      ; Posición 0x00E5
```

```
En instrucción:       (PC) = posición 0x01A8, Nivel Stack 1, Stack = 01A9
Después instrucción:  (PC) = posición 0x00E5 = BIN_BCD
```

retlw k (retlw k)

Retorne al último llamado a la subrutina (*Call*), guarde 'k' en registro (W).

Operación **k -> (W), Stack -> (PC), Nivel Stack-1** **no afecta las banderas:**

```
Ejemplo:
0029 call Tabla_code_7_seg    ; llame a Tabla código de 7 segmentos
002A   movwf   PORTD          ; póngalo en PUERTO D
……
Tabla_code_7_seg      ; Tabla para display de 7 segmentos.
0140   addwf   PCL,F
0141   retlw 3Fh      ; número "0".
0142   retlw 06h      ; número "1".
……
014A   retlw 67h      ; número "9".
```

A continuación se realiza el proceso total de traer un dato de la tabla código de 7 segmentos:

```
Dir    Instrucción    Registros
                      (PC) = 0x0029, (W) = 9
0029   call   Tabla_code_7_seg
                      (PC) = 0x0140,(W)= 9, 002A -> Stack  , Nivel Stack 1
0140   addwf    PCL,F
                      (PC) = 0x014A
014A   retlw  67h
                      (PC) = 0x002A,(W)= 0x67, Stack -> (PC),Nivel Stack 0
002A   movwf    PORTD
```

return (return)

Retorne al último llamado a subrutina (Call).

Operación **Stack -> (PC), Nivel Stack-1** **no afecta las banderas:**

```
Ejemplo:
01A8    Call BIN_BCD   ; k -> (PC).
01A9    movf valor,W
                ……
BIN_BCD
00E5    clrw    ……     ; (PC) salta acá, primera instrucción de subrutina
                ……
00F9    return
```

Seguidamente se realiza el proceso total de llamado y *retorno de la subrutina BIN_BCD*:

```
Dir     Instrucción    Registros
                       (PC) = 0x01A8
01A8    Call BIN_BCD   ; k -> (PC).
                       (PC) = 0×00E5  01A9 -> Stack         Nivel Stack 1
                ……
BIN_BCD
00E5    clrw    ……     (PC) salta acá, primera instrucción de subrutina
                ……
                       (PC) = 0×00F9
00F9    return
                       (PC) = 0x01A9  Stack -> (PC)         Nivel Stack 0
01A9    movf valor,W
```

retfie (retfie)

Retorne al último llamado de interrupción.

Operación **Stack -> (PC), Nivel Stack-1** **no afecta las banderas:**

```
Ejemplo:
0089    crlf    valor
008A    clrw                   ; interrupción, PC=0004, 008A -> Stack
008B    movf nivel,W
                ……
```

En seguida se realiza el proceso total de llamado y retorno de interrupción:

```
Dir     Instrucción    Registros
                       (PC) = 0×0089
0089    crlf    valor
                       (PC) = 0×008A
008A    clrw           ; interrupción
                       (PC) = 0×0004, 008B -> Stack, Nivel Stack 1
0004    goto    INT_rutina
                       (PC) = 0×0056
                ……
INT_rutina
```

```
0056    movwf    back_W ; (PC) salta acá, primera instrucción de subrutina
        de interrupción
0057    movf STATUS,W
                ……
                        (PC) = 0×0062
0062    retfie          ; Fin de rutina de interrupción
                        (PC) = 0x008B  Stack -> (PC)  Nivel Stack 0
008B    movf nivel,W
```

Para estas instrucciones remitimos a los ejemplos que se encuentran en el capítulo.

11.3.7 INSTRUCCIONES ESPECIALES

Estas instrucciones no tienen operando.

nop

No haga operación, gasta un ciclo.

Operación nop no afecta las banderas:

```
Ejemplo:
014E    crlf    PORTD
014F    nop             ; esta instrucción no afecta los registros
0150    movf nivel,W
```

clrwdt

Aclare WDT, comienza de nuevo el Time WDT.

Operación 0 -> WDT no afecta las banderas:

```
Ejemplo:
00C4    decf    contador,F
00C5    clrwdt                  ; aclara WDT
00C6    goto    Main
```

sleep (sleep)

Entra a bajo consumo, detiene Oscilador, PC detenido.

Operación -PD=0, PC detenido, Oscilador detenido no afecta las banderas:

Ejemplo:

```
0163    clrwdt
0164    sleep           ; PC detenido
0165    clrf    PORTE
0166    goto    Wake_up
```

A continuación se realiza el proceso total de *sleep* (dormir) y *wake-up* (despertar):

```
Dir     Instrucción     Registros
                        (PC) = 0x0163
0163    clrwdt
                        (PC) = 0x0164
0164    sleep           ; PC detenido
                        (PC) = 0x0165 y detenido, oscilador detenido
        ……              si ocurre un wake-up from sleep
0165    clrf    PORTE   ; despierta y ejecuta la instrucción
                        (PC) = 0x0166
0166    goto    Wake_up
```

11.3.8 TIPS Y USOS COMUNES

En esta sección veremos algunos "tips" o ayudas en la solución de retos, y también el uso común de instrucciones en la realización de programas.

Dividimos estos tips en grupos, así:

- Manejo de bits
- Operaciones aritméticas
- Comparación
- Ciclos iterativos

A continuación los detallamos.

MANEJO DE BITS

Prender o apagar 1 bit

- En el momento de **prender** el pin de un PUERTO:

```
bsf     f,b     ; pone en "1" el bit (b) del registro (f), (f<b>)=1
bsf     PORTB,4 ; pone en "1" o alto o 5V, el bit 4 del registro PORTB
```

	7	6	5	4	3	2	1	0	BIT
bsf PORTB,4	0	1	1	0	1	1	0	1	**PORTB**
	0	1	1	1	1	1	0	1	**PORTB**

- En el momento de **apagar** el bit de un registro:

```
bcf     f,b          ; pone en "0" el bit (b) del registro (f), (f<b>)=0
bcf     STATUS,RP0 ; pone en "0" o bajo o 0V,el bit RP0 del registro STATUS
```

	IRP	RP1	RP0	-TO	-PD	Z	DC	C	BIT
	7	6	5	4	3	2	1	0	**BIT**
bcf STATUS,RP0	0	1	1	0	1	1	0	1	**STATUS**
	0	1	0	1	1	1	0	1	**STATUS**

NOTA: Estas instrucciones no son posibles con el registro W, tampoco con los registros de sólo lectura.

Prender, apagar o invertir varios bits

- Cuando queremos **prender** los bits de un PUERTO:

```
        movlw   k       ; k -> (W)
        iorwf   f,d     ; (W) or (f) -> (d) 0->(W), 1->(f)
```

En el filtro *k*, los bits en 1 **prenden** los bits del PUERTO y los bits en 0 no alteran los bits del PUERTO:

```
        movlw   b'1111 0000     ; carga filtro en (W)
        iorwf   PORTD,F         ; aplica filtro a PORTD y lo guarda en PORTD
```

	7	6	5	4	3	2	1	0	BIT
	1	1	1	1	0	0	0	0	***K***
movlw k	1	1	1	1	0	0	0	0	**W**
	1	0	1	0	0	1	1	1	**PORTD**
iorwf PORTD,F	1	1	1	1	0	1	1	1	**PORTD**

Estas instrucciones *prenden* los bits <7:4> del PUERTO D.

- Cuando queremos **apagar** los bits de un PUERTO:

```
movlw   k       ; k -> (W)
andwf   f,d     ; (W) and (f) -> (d), 0->(W), 1->(f)
```

En el filtro *k*, los bits en 0 **apagan** los bits del PUERTO y los bits en 1 no alteran los bits del PUERTO:

```
movlw   b'1111 0000     ; carga filtro en (W)
andwf   PORTD,F         ; aplica filtro a PORTD y lo guarda en PORTD
```

	7	6	5	4	3	2	1	0	BIT
	1	1	1	1	0	0	0	0	*K*
movlw k	1	1	1	1	0	0	0	0	**W**
	1	0	1	0	0	1	1	1	**PORTD**
andwf PORTD,F	1	0	1	0	0	0	0	0	**PORTD**

Estas instrucciones apagan los bits <3:0> del PUERTO D

Invertir 1 o varios bits

De esta manera podemos cambiar el estado del bit, si está en "1" cambiarlo a "0" y si el bit está en "0" cambiarlo a "1"; este proceso se llama *toggle* o **inversión**.

- Cuando queremos **invertir** los bits de un PUERTO:

```
movlw   k       ; k -> (W)
xorwf   f,d     ; (W) xor (f) -> (d) 0->(W), 1->(f)
```

En el filtro *k*, los bits en 1 **invierten** los bits del PUERTO y los bits en 0 no alteran los bits del PUERTO:

```
movlw   b'0001 0001     ; carga filtro en (W)
xorwf   PORTC,F         ; aplica filtro a PORTC y lo guarda en PORTC
```

	7	6	5	4	3	2	1	0	BIT
	0	0	0	1	0	0	0	1	*K*
movlw k	0	0	0	1	0	0	0	1	**W**
	1	0	1	1	0	1	0	0	**PORTC**
xorwf PORTC,F	1	0	1	0	0	1	0	1	**PORTC**

Estas instrucciones **invierten** los bits <4 y 0> del PUERTO C.

Complementar el registro W

- Ésta es una aplicación del mismo concepto de invertir bits, con el operador *xor*:

	7	6	5	4	3	2	1	0	BIT	hex	dec
	1	1	1	1	1	1	1	1	***K***	**FFh**	**255d**
	1	1	0	1	0	1	1	0	**W**	**D6h**	**214d**
xorlw k	0	0	1	0	1	0	0	1	**W**	**29h**	**41d**

Observamos que la suma en *hex* de los valores del registro *W* son *FFh*, y que la suma en decimal nos da *255d.*

Complementar un registro

- Esta operación se realiza con el operador *comf*:

```
comf f,d          ; (/f) -> (d), 0->(W), 1->(f)
comf nivel,F      ; (/nivel) -> (nivel)
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	0	1	0	0	1	0	1	1	**nivel**	**4Bh**	**75d**
comf nivel,F	1	0	1	1	0	1	0	0	**nivel**	**B4h**	**180d**

OPERACIONES ARITMÉTICAS

- Al incrementar un registro:

```
incf f,d                ; (f) + 1 -> (d)
incf contador,F         ; contador + 1 -> contador
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	0	0	0	1	1	0	0	1	**contador**	**19h**	**25d**
incf contador,F	0	0	0	1	1	0	1	0	**contador**	**1Ah**	**26d**

- Al incrementar el registro W:
- Se usa el truco de sumar 1:

```
addlw k         ; k + (W) -> (W)
addlw  .1       ; 1 + (W) -> (W)
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	1	1	1	1	1	1	1	1	W	FFh	255d
addlw .1	0	0	0	0	0	0	0	0	W	00h	0d

	IRP	R P 1	R P 0	- T O	- P D	Z	D C	C	
	7	6	5	4	3	2	0	1	BIT
	0	0	0	0	0	0	0	0	STATUS
addlw .1	0	0	0	0	0	1	0	1	STATUS

Para este caso ejemplo, observamos que hay un C=1, *Carry* o Acarreo al incrementar de FFh a 00h, o sea acarreo del bit 7, y también bandera Z=1, *Zero* o cero, *Zero*, porque el nuevo valor de W= 0.

Incrementar un registro con una constante:

- Al incrementar un registro con una constante:

```
movlw   k               ; k -> (W)
addwf   f,d             ; (W) + (f) -> (d) 0->(W), 1->(f)
```

```
movlw   h07             ; 7 -> (W)
addwf   contador,F      ; 7 + contador -> contador
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	0	0	0	0	0	1	1	1	*K*	07h	07d
movlw k	0	0	0	0	0	1	1	1	W	07h	07d
	0	0	0	0	1	0	0	1	contador	09h	09d
addwf contador,F	0	0	0	1	0	0	0	0	contador	10h	16d

	IRP	R P 1	R P 0	- T O	- P D	Z	D C	C	
	7	6	5	4	3	2	0	1	BIT
	0	0	0	0	0	0	0	0	STATUS
addwf contador,F	0	0	0	0	0	0	1	0	STATUS

Para este caso ejemplo, observamos que hay un DC=1, *Digit Carry* o Acarreo Digital al incrementar de bit 3 a 4 o de *nibble,* o sea de dígito LSB a dígito MSB.

Incrementar un registro con otro registro

- Al incrementar un registro con otro registro:

```
movf    f,d             ; (f) -> (d) 0->(W), 1->(f)
addwf   f,d             ; (W) + (f) -> (d) 0->(W), 1->(f)
```

Ahora sumaremos el registro conteo_1 a total:

```
movf    conteo_1,W      ; conteo_1 -> (W)
addwf   total,F         ; (W) + total -> total … o también
                        ; conteo_1 + total -> total
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	0	0	1	0	0	1	1	1	conteo_1	27h	39d
movf conteo_1,W	0	0	1	0	0	1	1	1	W	27h	39d
	1	0	1	0	0	1	1	0	total	A6h	166d
addwf total,F	1	1	0	0	1	1	0	1	total	CDh	205d

Disminuir en 1 un registro

- Al disminuir un registro:

```
decf    f,d             ; (f) - 1 -> (d)
decf    contador,F      ; contador - 1 -> contador
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	1	0	1	1	1	0	1	0	contador	BAh	186d
decf contador,F	1	0	1	1	1	0	0	1	contador	B9h	185d

Al disminuir el registro W:

- Se usa el truco de sumar FFh:

```
addlw   k               ; k + (W) -> (W)
addlw   0×FF            ; 0×FF + (W) -> (W)
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	1	1	0	0	0	1	1	0	W	C6h	198d
	1	1	1	1	1	1	1	1	*K*	FFh	255d
addlw 0xFF	1	1	0	0	0	1	0	1	W	C5h	197d

	IRP	R P 1	R P 0	- T O	- P D	Z	D C	C	
	7	6	5	4	3	2	0	1	BIT
	0	0	0	0	0	0	0	0	STATUS
addlw 0xFF	0	0	0	0	0	0	0	1	STATUS

Para este caso ejemplo, observamos que hay un C=1, *Carry* o Acarreo; esto sucede al ser positivo el resultado de la resta (que en realidad es una suma).

Disminuir un registro con una constante

- Al disminuir un registro con una constante:

```
movlw   k               ; k -> (W)
subwf   f,d             ; (f) - (W) -> (d) 0->(W), 1->(f)
```

Debemos anotar que en la resta están los operandos:
minuendo – sustraendo = diferencia
contador – k = contador

```
movlw   h1B             ; 1B -> (W)
subwf   contador,F      ; contador - (W) -> contador
                        ; contador - 1B -> contador
```

	7	6	5	4	3	2	1	0	BIT	hex	dec	
	0	0	0	1	1	0	1	1	*K*	1Bh	27d	Sustraendo
movlw k	0	0	0	1	1	0	1	1	W	1Bh	27d	
	0	1	1	0	1	0	1	0	contador	6Ah	106d	Minuendo
subwf contador,F	0	1	0	0	1	1	1	1	contador	4Fh	79d	Diferencia

	IRP	R P 1	R P 0	- T O	- P D	Z	D C	C	
	7	6	5	4	3	2	0	1	BIT
	0	0	0	0	0	0	0	0	STATUS
subwf contador,F	0	0	0	0	0	0	0	1	STATUS

Para este caso ejemplo, observamos que hay un C=1, *Carry* o Acarreo; esto sucede al ser positivo el resultado de la resta.

Disminuir un registro con otro registro

- Al disminuir un registro con otro registro:

```
movf    f,d             ; (f) -> (d) 0->(W), 1->(f)
subwf   f,d             ; (f) - (W) -> (d) 0->(W), 1->(f)
```

Debemos anotar que en la resta están los operandos:
minuendo – sustraendo = diferencia
total – conteo_1 = total:

```
movf    conteo_1,W      ; conteo_1 -> (W)
subwf   total,F         ; total - (W) -> total … o también
                        ; total - conteo_1 -> total
```

	7	6	5	4	3	2	1	0	BIT	hex	dec	
	0	0	1	0	0	1	0	1	conteo_1	25h	37d	Sustraendo
movf conteo_1,W	0	0	1	0	0	1	0	1	W	25h	37d	
	0	1	1	0	1	1	1	1	total	6Fh	111d	Minuendo
subwf total,F	1	1	0	0	1	1	0	1	total	4Ah	74d	Diferencia

	IRP	R P1	R P0	- T O	- P D	Z	D C	C	BIT
	7	6	5	4	3	2	0	1	BIT
	0	0	0	0	0	0	0	0	STATUS
subwf total,F	0	0	0	0	0	0	0	1	STATUS

Para este caso ejemplo, observamos que hay un C=1, *Carry* o Acarreo; esto sucede al ser positivo el resultado de la resta.

Negar el registro W, (-W)

- Negar el registro W, es un caso especial de la resta, consiste en restar 0-W, es el mismo término que complementar el registro W:

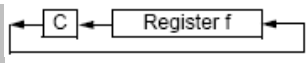
```
sublw   k               ; k - (W) -> (W)
sublw   0               ; 0 - (W) -> (W)
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	0	0	0	0	0	0	0	0	*K*	00h	00d
	1	0	1	0	0	1	1	0	W	A6h	166d
sublw k	0	1	0	1	1	0	0	1	W	59h	89d

Multiplicar un registro por una potencia de 2

- Multiplicar por 2 equivale a rotar a la izquierda todo el registro una vez, si rota de nuevo equivale a multiplicar por 4 y otra vez a multiplicar por 8, y así sucesivamente:

```
rlf     f,d     <<(f)           ;
```



Antes de rotar, hay que tener en cuenta que se debe apagar la bandera C=0, o *Carry* del registro STATUS, porque la instrucción *rlf f,d* pasa por el *Carry*:

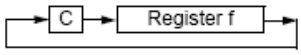
	7	6	5	4	3	2	1	0	BIT	hex	dec	
	0	0	0	1	1	0	0	1	conteo	19h	25d	
rlf conteo,F	0	0	1	1	0	0	1	0	conteo	32h	50d	Multiplica por 2
rlf conteo,F	0	1	1	0	0	1	0	0	conteo	64h	100d	Multiplica por 4
rlf conteo,F	1	1	0	0	1	0	0	0	conteo	C8h	200d	Multiplica por 8

También hay que tener en cuenta que el resultado no debe ser mayor que 255d o FFh, pues sería mayor que 8 bits.

Dividir un registro por una potencia de 2

- Dividir por 2 equivale a rotar a la izquierda todo el registro una vez, si rota de nuevo equivale a dividir por 4 y otra vez a dividir por 8, y así sucesivamente:

```
rrf     f,d     >>(f)           ;
```



Antes de rotar, hay que tener en cuenta que se debe apagar la bandera *C*=0, o *Carry* del registro STATUS, porque la instrucción *rrf f,d* pasa por el *Carry.*

	7	6	5	4	3	2	1	0	BIT	hex	dec	
	1	0	0	1	1	0	0	0	conteo	98h	152d	
rrf conteo,F	0	1	0	0	1	1	0	0	conteo	4Ch	76d	Divide por 2
rrf conteo,F	0	0	1	0	0	1	1	0	conteo	26h	38d	Divide por 4
rrf conteo,F	0	0	0	1	0	0	1	1	conteo	13h	19d	Divide por 8

También hay que tener en cuenta que el resultado no debe afectar el *Carry.*

COMPARACIÓN

Comparación es establecer el valor relativo a una referencia, es una operación que involucra 1 o 2 registros. Cuando realizamos una comparación es necesario tener en cuenta que esta operación es en dos pasos, el primer paso es una operación que puede ser de **carga** o **aritmética**, y el segundo paso es una **decisión** según los bits del registro STATUS (*C* o *Carry*, *Z* o *Zero*).

En el primer paso

La operación de *carga* con la instrucción *movf f,d* altera la bandera *Z* o *Zero* en el registro STATUS. La operación *aritmética* es una resta que altera las banderas *C* o *Carry*, *Z* o *Zero* en el registro STATUS.

En el segundo paso

La **decisión** se toma con las instrucciones **btfsc f,d** o **btfss f,d**, y opera sobre el registro **STATUS** y una de sus banderas *C* o *Carry*, *Z* o *Zero*; esta instrucción tiene dos rutas de salida, una cuando la respuesta es **SÍ** y otra cuando la respuesta es **NO** (Figura 11.18).

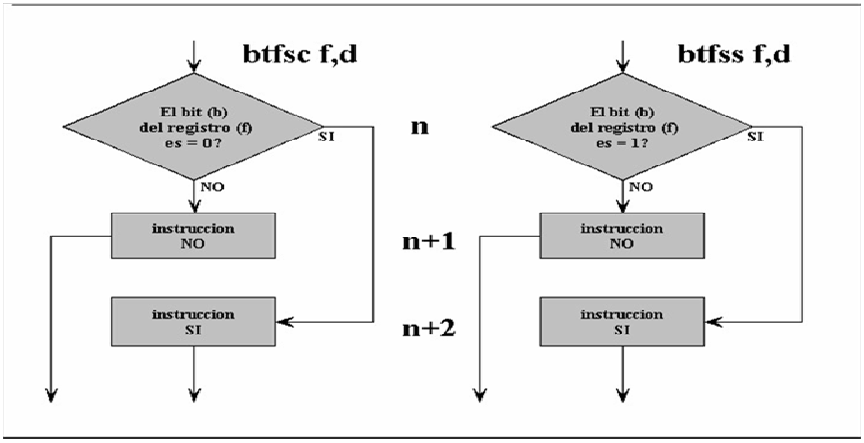


Figura 11.19 Gráfica saltos en función de bit.

Como observamos en la Figura 11.19, la instrucción de decisión se encuentra en la posición n del programa, la ruta de salida SÍ salta una instrucción a la posición n+2, y la ruta de salida NO es la siguiente instrucción en la posición n+1.

NOTA: Tengamos en cuenta que en la salida NO sólo puede colocar una sola instrucción que por lo regular es la instrucción goto k, de tal forma que el flujo del programa irá a una posición donde se encuentran las instrucciones que implementan la salida NO. En algunos casos se puede colocar otra instrucción, que empata con la salida SÍ.

Es criterio del usuario escoger cuál tipo de pregunta (instrucción) hace para que se facilite realizar el programa.

COMPARACIÓN DE UN SOLO REGISTRO

Comprobar valor 0 en un registro

- Comprobar el valor 0 en un registro puede realizarse de varias maneras, la más corta es con la instrucción *movf f,d*:

```
movf    f,d             ; (f) -> (d) 0->(W), 1->(f)
movf    nivel,F         ; nivel -> nivel
```

	7	6	5	4	3	2	1	0	BIT	hex	dec
	0	0	0	0	0	0	0	0	nivel	00h	00d
movf nivel,F	0	0	0	0	0	0	0	0	nivel	00h	00d

	IRP	R1	R0	-TO	-PD	Z	DC	C	
	7	6	5	4	3	2	0	0	BIT
	0	0	0	0	0	0	0	0	STATUS
movf nivel,F	0	0	0	0	0	1	0	0	STATUS

Al ejecutar la instrucción **movf f,d** el registro **STATUS** afecta la bandera *Z*=1, o *Zero*; en caso de que el registro sea **nivel = 0**. Luego de este paso sigue la comprobación de la bandera *Z* en el registro STATUS. Un ejemplo de este procedimiento puede ser la aplicación de trabajo de un tanque, donde se necesita el chequeo de nivel minino=0.

```
Ejemplo:

check_nivel_min
paso 1  movf    nivel,F              ; check nivel mínimo
```

```
paso 2  btfsc   STATUS,Z                ; si Z=0 salte a check_nivel_max
        bsf     alarmas, _nivel_min     ; si Z=1, ponga en "1" bit _nivel_min
                                        ; en registro alarmas
check_nivel_max
        …..
```

Observamos que en el ejemplo anterior, en el *paso 1,* la instrucción movf nivel,F altera el registro STATUS si el registro nivel=0, en el **paso 2** se hace la pregunta con la instrucción btfsc STATUS,Z que tiene dos posibilidades:

- La salida de **NO**, o sea *Z=1* que significa *nivel = 0*, que es la siguiente instrucción bsf alarmas, _nivel_min donde pone en "1" el bit _nivel_min en el registro alarmas.

- La salida de **SÍ**, o sea **Z=0** que significa **nivel != 0**, que es la instrucción de salto hacia la label **check_nivel_max** donde continúa el programa.

En este caso, la ruta de salida de **NO** es solamente una instrucción que empata con la salida **SÍ**.

COMPARACIÓN DE DOS REGISTROS

Cuando se va a comparar dos registros, hay que tener en cuenta tres posibles resultados; en la Tabla 11.7 se resumen estos casos.

Tabla 11.7 Comparación de dos registros.

COMPARACIÒN	OPERACIÒN A - B =	VALOR	STATUS C , Z
A > B	> 0	POSITIVO	C = 1 y Z = 0
A = B	= 0	CERO	C = 1 y Z = 1
A < B	< 0	NEGATIVO	C = 0 y Z = 0

De este modo, para saber si dos registros son iguales, solamente se requiere la pregunta por la bandera Z, y luego la pregunta por la bandera C, si se quiere saber si es mayor o menor.

Comprobar igualdad entre dos registros

- Comprobar la igualdad de 2 registros requiere hacer una operación de sustracción, reg_A – reg_B, y la pregunta si el resultado es igual a 0:

	7	6	5	4	3	2	1	0	BIT	hex	dec
	1	0	0	0	1	1	0	0	reg_A	8Ch	140d
	1	0	0	0	1	1	0	0	reg_B	8Ch	140d
movf reg_B,W	1	0	0	0	1	1	0	0	W	8Ch	140d
subwf reg_A,W	0	0	0	0	0	0	0	0	W	00h	00d

	I R P	R P 1	R P 0	- T O	- P D	Z	D C	C	
	7	6	5	4	3	2	0	0	BIT
	0	0	0	0	0	0	0	0	STATUS
subwf reg_A,W	0	0	0	0	0	1	0	0	STATUS

```
Ejemplo:
Compare por Z=0, no son iguales
paso 1  movf reg_B,W   ; reg_B –>(W)
subwf   reg_A,W        ; reg_A – (W) –>(W)
                       ; reg_A – reg_B –>(W)
paso 2  btfsc STATUS,Z ; si Z=0, no son iguales, salte a No_iguales
Si_iguales....         ; si Z=1, sí son iguales
No_iguales....
```

En el **paso 1** observamos que se realiza la diferencia **reg_A – reg_B -> (W)**, y en el **paso 2** está la pregunta con la instrucción *btfsc STATUS,Z* con sus dos salidas:

- La ruta de salida **NO**, **No_iguales**
- La ruta de salida **SÍ**, **Si_iguales**

```
Ejemplo:
Compare por Z=1, si son iguales
paso 1  movf reg_B,W   ; reg_B –>(W)
subwf   reg_A,W        ; reg_A – (W) –>(W)
                       ; reg_A – reg_B –>(W)
paso 2  btfss STATUS,Z ; si Z=1, si son iguales, salte a Si_iguales
No_iguales ....        ; si Z=0, no son iguales ....
Si_iguales ....
```

En el **paso 1** observamos que se realiza la diferencia **reg_A – reg_B -> (W)**, y en el **paso 2** está la pregunta con la instrucción **btfss STATUS,Z** con sus dos salidas:

- La ruta de salida **NO, No_iguales**
- La ruta de salida **SÍ, Si_iguales**

De estos dos ejemplos deducimos que las salidas son de acuerdo con el tipo de pregunta que se haga sobre el bit **Z**, si es set **"1"** o si es clear **"0"**; de esta manera podemos escoger dónde queden las instrucciones en la salida **SÍ** o en la salida **NO**.

Comprobar mayor o menor entre dos registros

- Comprobar mayor o menor entre 2 registros requiere hacer una operación de sustracción, reg_A – reg_B, en seguida la pregunta si son iguales y luego la pregunta si es mayor:

	7	6	5	4	3	2	1	0	BIT	hex	dec
	1	0	0	0	1	1	0	0	reg_A	8Ch	140d
	1	0	1	1	1	1	1	1	reg_B	BFh	191d
movf reg_B,W	1	0	1	1	1	1	1	1	W	BFh	191d
subwf reg_A,W	0	0	0	0	0	0	0	0	W	00h	00d

	IRP	R P 1	R P 0	- T O	- P D	Z	D C	C	
	7	6	5	4	3	2	0	0	BIT
	0	0	0	0	0	0	0	0	STATUS
subwf reg_A,W	0	0	0	0	0	0	0	0	STATUS

Para este ejemplo **reg_A = 8Ch y reg_B = BFh**, con lo cual se deduce que A<B; sin embargo, nuestro programa es el que nos tiene que dar esta respuesta; por tanto, tenemos que averiguar primero si son iguales y luego si es mayor o menor.

Para este caso, tendremos que hacerlo en 3 pasos:

Paso 1. Realizar la resta reg_A – reg_B -> (W).

Paso 2. Preguntar si son iguales, probando si el bit Z=1 del registro STATUS.

Paso 3. Preguntar si es mayor probando si el bit C=1 del registro STATUS.

```
Ejemplo:
Compare
paso 1  movf reg_B,W          ; reg_B –>(W)
        subwf reg_A,W         ; reg_A – (W) –>(W)
```

```
                              ; reg_A – reg_B –>(W)
paso 2  btfsc STATUS,Z        ; si Z=0 salte a No_iguales
Si_iguales
        goto SR_iguales       ; si Z=1, salte a SR_iguales
No_iguales
paso 3  btfsc STATUS,C        ; si C=0, A_menor_B
A_mayor_B
        goto SR_A_mayor_B     ; si C=1 salte a SR_A_mayor_B
A_menor_B
        goto SR_ A_menor_B    ; salte a SR_ A_menor_B
....
SR_iguales
....
SR_A_mayor_B
....
SR_ A_menor_B
```

Las LABEL SR_iguales, SR_A_mayor_B, SR_ A_menor_B, son las posiciones donde están las rutinas para cada una de estas opciones.

CICLOS ITERATIVOS

Ésta es una de las estructuras que ayudan a simplificar la programación (Figura 11.19). Permite realizar un conjunto de instrucciones de forma infinita o finita.

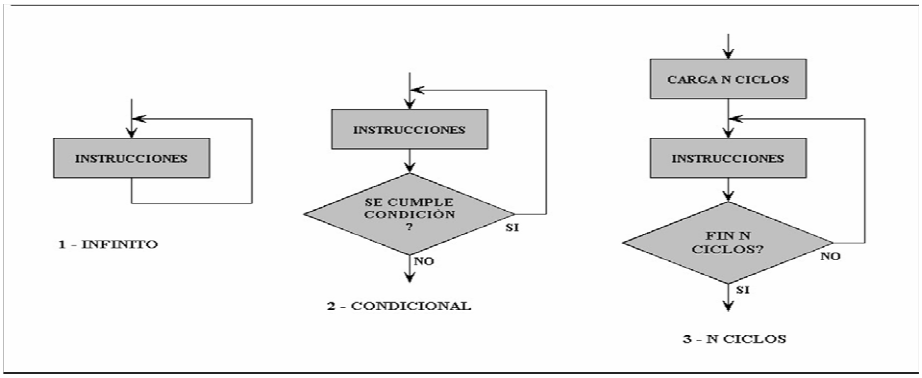


Figura 11.20 Ciclos iterativos.

También conocida como **Ciclos**, **Bucles** o **Lazos**, esta estructura tiene una entrada, un conjunto de instrucciones que se repiten, una condición (o no) y una salida (o no). En la Figura 11.20 apreciamos tres tipos básicos de esta estructura, como infinito, condicional, *n* ciclos.

CICLO ITERATIVO INFINITO

Éste es un ciclo infinito que se repite siempre y usa la instrucción goto k, que es el salto a un punto del programa, usado comúnmente en todos los programas.

```
Ejemplo:

        ....
Main_Loop
        ....
        goto    Main_Loop
```

CICLO ITERATIVO CONDICIONAL

Este ciclo finito se repite mientras la condición se cumpla (o no); esto quiere decir que podemos construir ciclos condicionales que se repitan mientras la condición se cumpla, o ciclos condicionales que se repitan mientras la condición no se cumpla. Esto depende del tipo de pregunta que realice el usuario o del tipo de instrucción usada.

```
Ejemplo:

        ....
check_sw_stop

        ....                            ; hace procedimiento de sw_stop
        btfsc   SWPANEL,STOP            ; pregunta si SW STOP no oprimido
        goto    check_sw_stop           ; aún sigue oprimido SW STOP
Start                                   ; arranca cuando ya no está oprimido
SW STOP
        ....
```

En este caso, el procedimiento de **check_sw_stop** y sus instrucciones se realizarán mientras el bit **STOP** en el registro **SWPANEL** esté oprimido o tenga el valor **"1"**, cuando sea **"0"** o no esté oprimido salta a *Start* y continúa el procedimiento de arranque. Observamos que la instrucción btfsc SWPANEL,STOP es la pregunta que condiciona que el ciclo continúe o no; se acostumbra que en la

salida **NO**, o sea la instrucción siguiente, vaya a **goto k** que es aquella que cierra el ciclo.

CICLO ITERATIVO N CICLOS

En este ciclo finito se realiza un número conocido de ciclos, de tal forma que antes de entrar al ciclo repetitivo se carga el número de ciclos que se va a realizar, y en cada iteración disminuye este valor hasta llegar a 0.

Ejemplo:

```
        ....
        movlw   _n_botellas     ; número de botellas por caja
        movwf   cont_botellas   ; carga a contador de botellas
check_botellas
        ....                    ; hace procedimiento de check de botellas
        decfsz  cont_botellas,F; disminuye cont_botellas, si=0, salte
        goto Check_botellas     ; cont_botellas no=0
Arme_caja                       ; termina conteo de botella y comienza
                                ;arme caja
....
```

En este caso, tenemos una aplicación que cuenta un número de botellas para empacar en una caja, en cada ciclo de check-botellas disminuye el cont_botellas; cuando este valor llega a cero el flujo del programa salta a Arme_caja, terminando el número de ciclos escogido. Observamos que la instrucción decfsz cont_botellas, F es el contador de ciclos y permite que el ciclo continúe o no; se acostumbra que en la salida **NO**, o sea la instrucción siguiente, vaya a **goto k** que es aquella que cierra el ciclo.

EVALUACIÓN

```
P.: 1. Resuelva el valor de nivel en la siguiente instrucción:
nivel   <-   b'0100 1111'
bcf  nivel,6
R:
nivel =  b'0000 1111'

P.:2. Resuelva el valor de vble_1 en las siguientes instrucciones:
vble_1    <-   b'0101 0101'
bsf  STATUS,C
```

```
rrf  vble_1,f
R:
vble_1 = b’1010 1010’

P.:3. Resuelva el valor de sw_in en las siguientes instrucciones:
sw_in    <-   b’0110 1001’
movlw   0X55
xorwf  sw_in,f
R:
sw_in = b’0011 1100’
```

11.4 DIRECTIVAS

Las directivas son comandos especiales que tiene el programa **MPLAB** para ayudar al ensamble y la compilación, tales como: definir el microcontrolador, asignar los bits de configuración, definir bloques de memoria **RAM** y definir datos de **tablas**, entre otros.

Algunos ejemplos de directivas empleadas en un programa son:

- Selección de microcontrolador:

```
LIST P=16F877A
```

- Selección de archivos con definiciones:

```
include "p16f877a.inc"   ; Include file for a PIC16F877A
```

- Selección de bits de configuración:

```
 __CONFIG  _CP_OFF  &  _WDT_OFF  &  _BODEN_OFF  &  _PWRTE_ON  &  _HS_OSC  &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

- Definición de constantes:

```
#define _write_ctrl    b'10100000'    ; byte de control para escritura
#define _read_ctrl     b'10100001'    ; byte de control para lectura
```

- Definición de bloque de memoria RAM para variables de usuario:

```
       cblock  0×20           ; separa espacio para variables
datai          ; buffer de 1 byte Data input
datao          ; buffer de 1 byte Data output
bytecount      ; contador para ciclos de byte
pollcnt        ; contador de ciclos de poll
```

```
loops          ; contador de loop de retardo
loops2         ; contador de loop de retardo
       endc
       cblock  0x2A
buffer :16            ; buffer de 16 datos
       endc
```

- Definición de posición de programa:

```
       org  0x000  ; vector de reset
Main:
       org 0004h
ISR_int               ; rutina de servicio de interrupción
```

A continuación (Figura 11.21) veremos en dónde se encuentran las *Directivas* en el ambiente de desarrollo MPLAB, para su consulta.

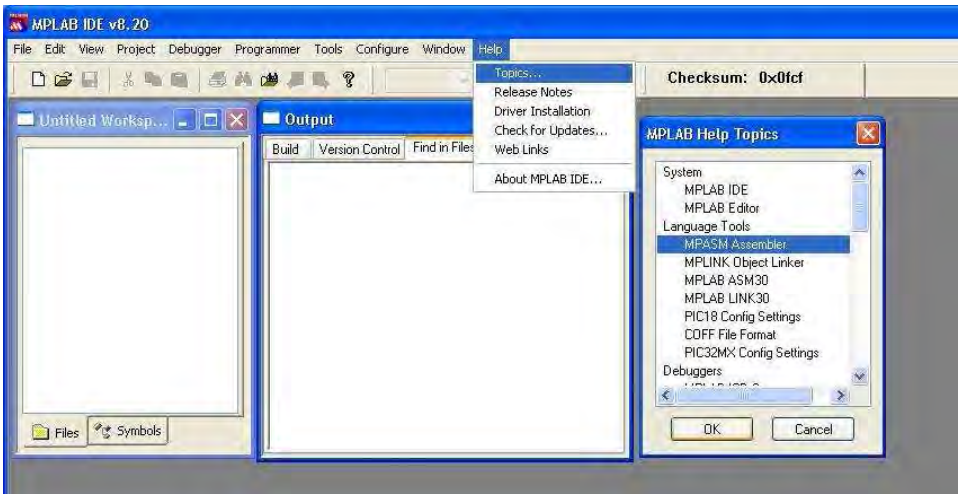


Figura 11.21 Menú Ayuda.

Se encuentran en el programa MPLAB en el menú de opciones **Help\Topics...\Languaje Tools\MPASM Assembler** como se muestra en la Figura 11.20, al dar clic en la casilla **OK** se despliega una ventana adicional de ayuda **MPASM Assembler Help (hlpMPASMAsm .chm** que contiene el menú de ayuda para Assembler, como vemos en la Figura 11.22.

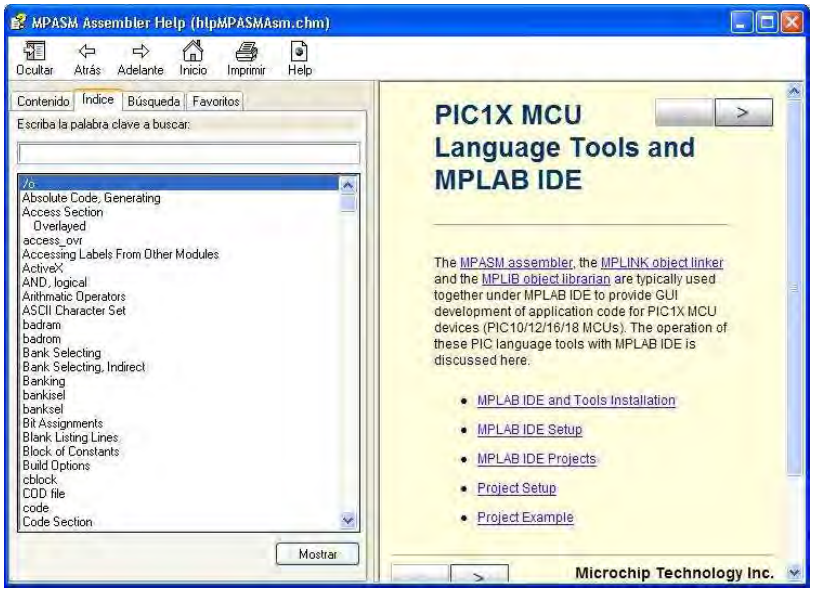


Figura 11.22 Menu Ayuda assembler.

En esta ventana (Figura 11.22) seleccionamos la pestaña **Índice** y en la casilla **Escriba palabra clave para buscar:** escribimos **Directives**, la que nos lleva al contenido de las directivas, como vemos en la Figura 11.23.

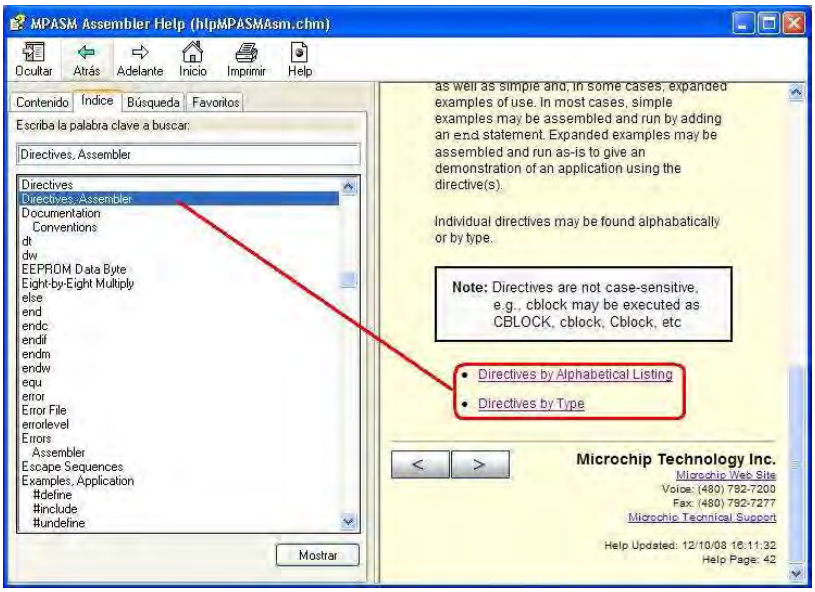


Figura 11.23 Contenido Directivas

Al final encontramos dos opciones:

- Directives by Alphabetical Listing (Directivas en orden afabético)
- Directives by Type (Directivas por tipo)

En esta ventana, al hacer clic en la opción Directives by Type se despliega el contenido de esta opción, tal como se ilustra en la Figura 11.24.

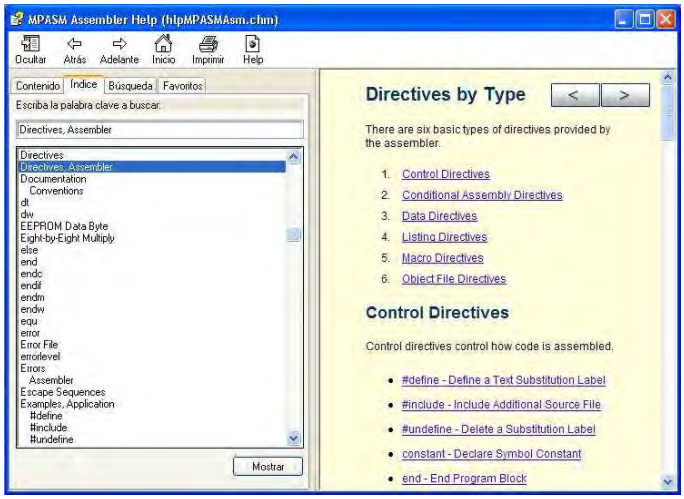


Figura 11.24 Directivas por tipo.

En esta ventana (Figura 11.25), al seleccionar la directiva **dt - Define Table (PIC12/16 MCUs)**, vemos el contenido de esta directiva. En ella encontramos la sintaxis, la descripción y su forma de uso con ejemplos, además de otros temas que se pueden seleccionar relacionados con esta directiva.



Figura 11.25 Directiva DT.

NOTA: Conviene anotar que todas las directivas contenidas en MPASM pueden servir no sólo al Lenguaje Assembler, sino también servir al Lenguaje C y a los desarrolladores o fabricantes de herramientas para Microchip.

Por tanto, no todas las directivas aplican en el momento de emplearlas, y deben ser consultadas en el caso de su aplicación. En seguida encontramos una lista de las directivas más usadas en el desarrollo de programas, como se muestra en la Tabla 11.8.

DIRECTIVA	DESCRIPCIÓN
_BADRAM	Especifica las posiciones del RAM inválidas
BANKISEL	Genera el código que selecciona el banco de memoria de memoria RAM para direccionamiento indirecto
BANKSEL	Genera el código que selecciona el código de memoria RAM
CBLOCK	Define un bloque de variables (inicio de registros de usuario)
CODE	Empieza la sección del código ejecutable
_ _CONFIG	Especifica los bits de configuración
CONSTANT	Declara los símbolos de las constantes
DATA	Crea datos numéricos y de texto
DB	Declara datos de un byte
DE	Define los datos de EEPROM
#DEFINE	Define una etiqueta de sustitución de texto
DT	Define tabla
DW	Declara datos de un word
ELSE	Empieza el bloque alternativo de un IF
END	Fin de bloque de variables (fin de registros de usuario)
ENDC	Acaba un bloque constante
ENDIF	Fin del bloque de instrucciones condicionales
ENDM	Fin de la definición de una Macro
ENDW	Fin de un bucle de While
EQU	Define una constante para el ensamblador
ERROR	Manda un mensaje de error
ERRORLEVEL	itúa el nivel del error
EXITM	Salida de una Macro
EXPAND	Expande una lista de Macro
EXTERN	Declara una etiqueta externa
FILL	Llena la memoria
GLOBAL	Exporta una etiqueta definida
IDATA	Comienza una sección de identificadores (ID)
_ _IDLOCS	Especifica dónde están colocados los Identificadores (ID)
IF	Empieza un bloque de código condicional
IFDEF	Ejecutar si el símbolo ha sido definido
IFNDEF	Ejecutar si el símbolo no ha sido definido
INCLUDE	Incluye ficheros fuente adiccionales
LIST	Opciones listado
LOCAL	Declara una Macro variable como local
MACRO	Declara la definición del Macro
_	Especifica la dirección del RAM máxima
MESSG	Crea mensajes definidos por el usuario
NOEXPAND	Termina la expansión del Macro
NOLIST	Termina el listado
ORG	Pone el origen del programa
PAGE	Inserta el número de página del listado
PAGESEL	Genera el código de selección de la página de ROM
PROCESSOR	El tipo del procesador utilizado
RADIX	Especifica sistema de numeración predefinido
RES	Reserva memoria
SET	Define variable de ensamblador
SPACE	Inserta líneas en blanco
SUBTITLE	Especifica el subtítulo del programa
TITLE	Especifica el título del programa
UDATA	Empieza la sección de datos no inicializados
UDATA_OVR	Empieza la sección de datos no inicializados superpuestos
UDATA_SHR	Empieza la sección de datos no inicializados compartidos
#UNDEFINE	Anula una etiqueta de sustitución
VARIABLE	Declara un símbolo como variable

Tabla 11.8 Directivas.

También se pueden encontrar en el Apéndice D: “Directivas del MPASM”.

11.5 DATOS

Para acceder y manipular datos en un microcontrolador, se deben tener ciertas claridades con respecto al tipo de dato, qué tipo de memoria usa y cómo es el acceso o el direccionamiento; a continuación veremos algunas clasificaciones de los datos, tales como:

- **Tipo de Direccionamiento.** Inmediato, Directo, Bit a Bit, Indirecto, Indexado
- **Tipo de Datos.** Constantes y Variables
- **Datos en Tipos de Memoria.** Memoria de Datos, Memoria de datos en E2PROM, Memoria de Programa.

Y como complemento a esta sección de Datos se tienen **Formatos de valores** y **Operadores.**

11.5.1 TIPO DE DIRECCIONAMIENTO

Ésta es la forma en que se accesan los datos. Según el **tipo de direccionamiento** utilizado para su manejo, se clasifican en:

- Direccionamiento inmediato
- Direccionamiento directo
- Direccionamiento bit a bit
- Direccionamiento indirecto
- Direccionamiento indexado

DIRECCIONAMIENTO INMEDIATO

El valor del dato se carga en forma inmediata, y se agrega al código de la instrucción. Todas las instrucciones que tengan el formato *xxxlw k*.

Ejemplo:

```
andlw  k              ; (W) and k -> (W)
andlw 0X4D            , (W) and 0×4D -> (W)

sublw  k              ; k – (W) -> (W)
sublw _ciclos_60Hz    ; _ciclos_60Hz – (W) -> (W)
```

DIRECCIONAMIENTO DIRECTO

Se carga la dirección de memoria, que contiene el valor del dato que se va a manejar, la dirección se agrega al **opcode** en la instrucción. Todas las instrucciones que tengan el formato *xxxxx f,d* (Figura 11.26).

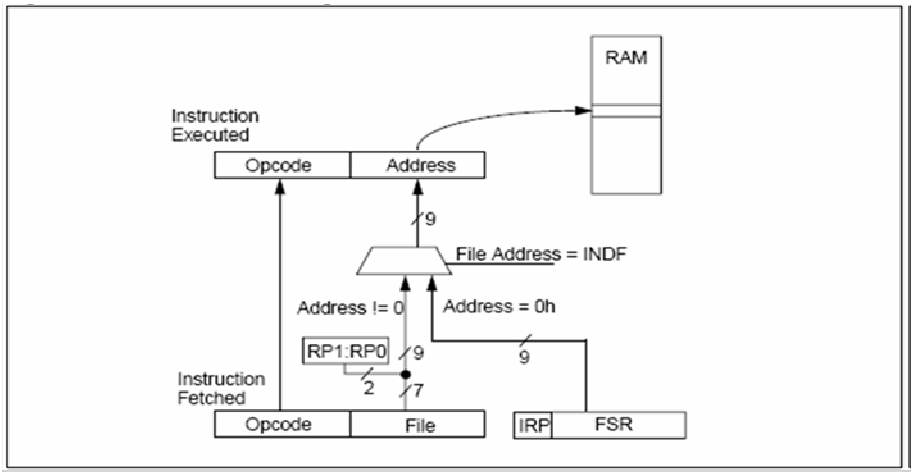


Figura 11.26 Direccionamiento directo conexión.

En la Figura 11.27 vemos cómo se carga la dirección (**!=0, diferente de 0**) cuando está en direccionamiento directo.

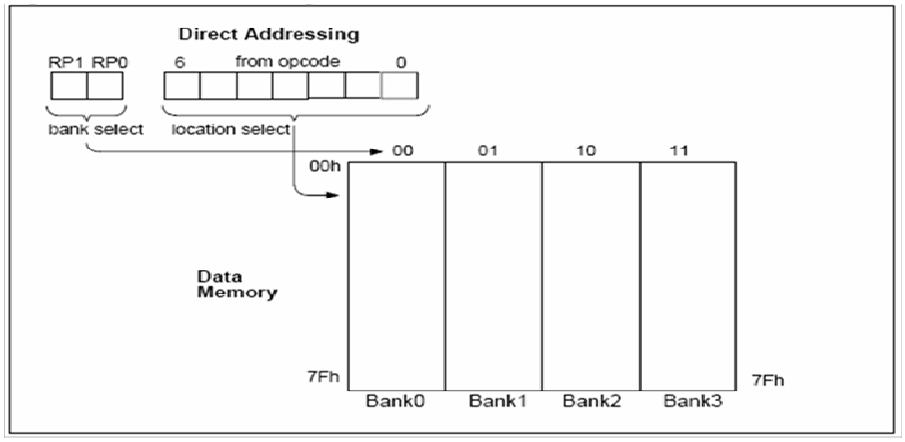


Figura 11.27 Direccionamiento directo diagrama.

En la Figura 11.26 observamos que la dirección está conformada por los 7 bits **<6:0>** (que direccionan de **0-7Fh**) y los bits RP1 y RP0 del registro **STATUS<7:6>**, que direccionan uno de los 4 bancos de memoria RAM (**Bank0, Bank1, Bank2, Bank3**).

Ejemplo:

```
movwf  f                ; (W) -> f
movwf  n_ciclos         ; (W) -> n_ciclos

addwf  f,d              ; (W) + f -> (d)
addwf  contador,F       ; (W) + contador -> contador
```

DIRECCIONAMIENTO BIT A BIT

Manipula el bit de un dato en memoria. Se carga la dirección de memoria del dato que contiene el bit seleccionado. Las instrucciones orientadas a bit que son **4, bsf, bcf, btfss, btfsc** todos terminan con **f,(b)**, **(f)** la dirección de memoria del dato y **(b)** el bit.

Ejemplo:

```
bsf  f,b                ; 1 -> b, f<b>
bsf  PORTA,4   ; 1 -> bit 4, PORTA<4>

btfss  f,b              ; f<b>, si bit b=1 salta
btfss  STATUS,Z         ; STATUS<Z>, Si Z=1 salta
```

NOTA: Tenga en cuenta que para el direccionamiento del registro f se debe escoger antes el banco en donde se ubica, con los bits RP1 y RP0 del registro STATUS, como vimos en el direccionamiento directo.

DIRECCIONAMIENTO INDIRECTO

Se carga el contenido (**INDF**) de la dirección de memoria donde apunta **FSR**. Este modo de direccionamiento es muy práctico debido a que se tiene un apuntador a memoria RAM (Figura 11.28).

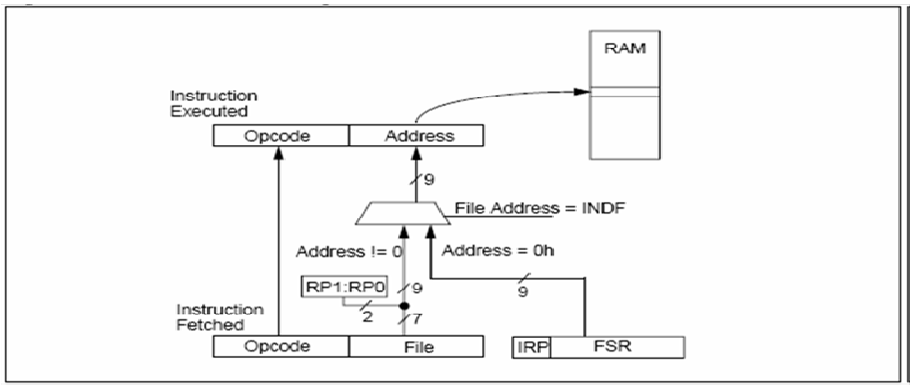


Figura 11.28 Conexión direccionamiento indirecto.

En la Figura 11.29 vemos cómo se carga la dirección (**=0, igual a 0, INDF**) cuando está en direccionamiento indirecto.

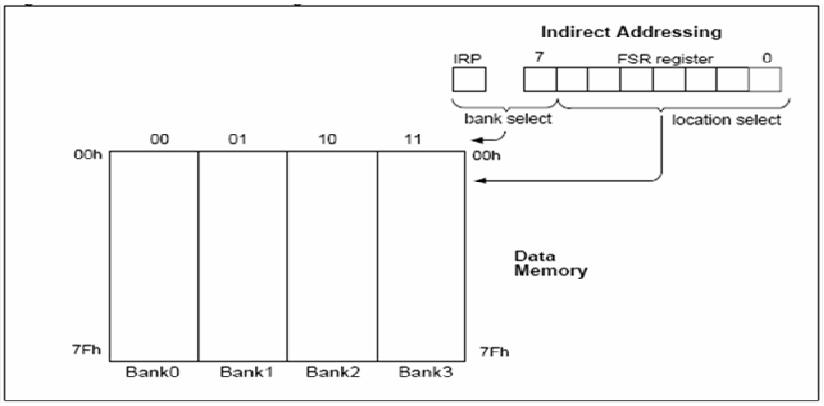


Figura 11.29 Diagrama direccionamiento indirecto.

En la Figura 11.29 observamos cómo la dirección está conformada por los 7 bits **<6:0>** (que direccionan de **0-7Fh, 0-127d**) y los bits IRP del registro **STATUS<7>** y el bit 7 del registro FSR<7>, que direccionan uno de los 4 bancos de la RAM (**Bank0, bank1, Bank2, Bank3**).

Ejemplo:

```
Si FSR=0x23
       clrf INDF      ; 0 -> INDF, donde apunta FSR
                      ; FSR = 0x23, la posición de memoria
                      ;0x23< - 00 (la memoria 0x23 es borrada)
```

```
Si (W) = 0x0F, FSR=0x23, y la posición de memoria (0x23) = 0x3A -> INDF
       andwf INDF,F   ; (W) and INDF –> INDF, donde apunta FSR
                      ; 0x0F and 0x3A -> (0x23) = 0x0A
```

DIRECCIONAMIENTO INDEXADO

Este modo de direccionamiento es un truco de software, debido a que no es implementado en estos micros. Se utiliza comúnmente en las tablas, donde se accede con un puntero (**PC**) y un offset (**W**).

Ejemplo:

```
Main:
       movf contador,W       ; cargue offset a (W)
       call Tabla_code_7_seg ; cargue apuntador (PC) a
                             ; Tabla código de 7 segmentos
       movwf PORTD           ; ponga W en PUERTO D
       ……..
       ……..
Tabla_code_7_seg             ;  (PC)  aquí,  Tabla  para  display  de  7
                             ;segmentos.
       addwf PCL,F           ; sume apuntador PC y offset
                             ;(W) -> PC
       retlw 3Fh             ; número "0".
       retlw 06h             ; número "1".
       ……..
       retlw 7Fh             ; número "8".
       retlw 67h             ; número "9".
       end
```

En el ejemplo anterior, si el contador tiene el valor 8, éste se guarda en (**W**) que es el *offset* para la tabla, con la instrucción **call el puntero (PC)** se ubica al principio de la tabla, y con la instrucción **addwf PCL, F** se suma el **offset** y el **puntero (PC)**, y ésta va a la posición 8 de la tabla, la instrucción **retlw 7Fh**, devuelve en **(W) = 7F**, al volver a la instrucción siguiente, luego del *call*, se ejecuta **movwf PORTD**, que pone el valor de (**W**) en el PUERTO D.

NOTA: Tenga en cuenta que para el direccionamiento de la tabla, se debe escoger antes el banco de memoria de programa donde se ubica, con los bits del registro PCLATH<5:4>.

Recomendamos la lectura de la Parte 4: “Memoria”, en donde aparecen ejemplos de este tipo.

11.5.2 TIPO DE DATOS

Según el tipo de uso, debemos diferenciar los *tipos de datos*:

- **Constantes.** Datos que no cambian durante el programa
- **Variables.** Datos que sí cambian durante el programa

CONSTANTES

Estos datos son los que no cambian durante el programa. **Se definen** en el programa con las **directivas "#DEFINE"** o **"EQU"**. Otra forma particular se encuentra en las **tablas**. Es más sencillo utilizar al principio del programa definiciones de *label* para los valores de constantes, de tal manera que en el desarrollo del programa, cuando se necesite la constante en la instrucción, se incluye la *label* de la constante y no su valor.

Ejemplo:

```
#define  _ciclos _60Hz        0x3C    ; 60 ciclos en hexadecimal
#define  _bit_2               .2      ; 2 en decimal
#define  _filtro              0x0F    ; filtro nibble menor LSB

_nivel_min     EQU            0x10    ; nivel mínimo 10 en hexadecimal
_nivel_max     EQU            hF0     ; nivel máximo F0 en hexadecimal
```

Así se usa mejor la *label* de **_ciclos_60Hz**, en vez del valor **0x3C**.

```
movlw _ciclos _60Hz    ; k -> (W)(mejor)
movlw 0x3C             ; k -> (W)(no adecuado)
```

Al compilar el programa cada instrucción que contenga la constante **_ciclos _60Hz** se cambia por el valor de la definición **0x3C**, que se agrega al **opcode** y se compila. También tiene la ventaja de que cuando se va a actualizar el valor de la constante sólo se hará una vez, por todo el programa, mucho mejor que andar buscando números de constantes en el programa para actualizar su valor.

NOTA: Se debe entender que las constantes no gastan memoria. No ocupan memoria de datos, o RAM, o registros de propósito general GPR (registros de usuario). Tampoco están en posiciones de memoria de programa definidas, ya que son parte de instrucciones. Cada vez que utilice la constante en el programa, allí se anexará esta constante al opcode de la instrucción.

En las **instrucciones** se reconoce como *k*. Si es una constante **dato**, es de 8 bits (0 -255, 0 – 0×FF). Si es una constante **dirección**, es de 11 bits (0 -2 047, 0 – 0×7FF).

```
NOTA: Una dirección en este micro es de 13 bits, el label es de 11
bits en el programa, más 2 bits de PCLATH<5:4>, que escogen en cuál
banco está la dirección. En la memoria de programa hay 4 bancos, de
2 048 instrucciones cada uno.
```

Ejemplo:

```
        addlw k                 ; (W) + k -> (W) (esta k es de 8
        bits)
        addlw 0×25              ; (W) + 0×25 -> (W)
        movlw k                 ; k -> (W) (esta k es de 8 bits)
        movlw _ciclos _60Hz     ; _ciclos _60Hz -> (W)
        goto k                  ; k -> (PC) (esta k es de 11 bits)
        goto MainLoop           ; MainLoop -> (PC), PC=0003
                                ; Posición de MainLoop es 0×0003
```

Este tipo de dato no está o no se ubica directamente en ninguna memoria, sino como parte de instrucciones en la memoria de programa. En la Figura 11.30 apreciamos los diferentes tipos de constante en un programa.

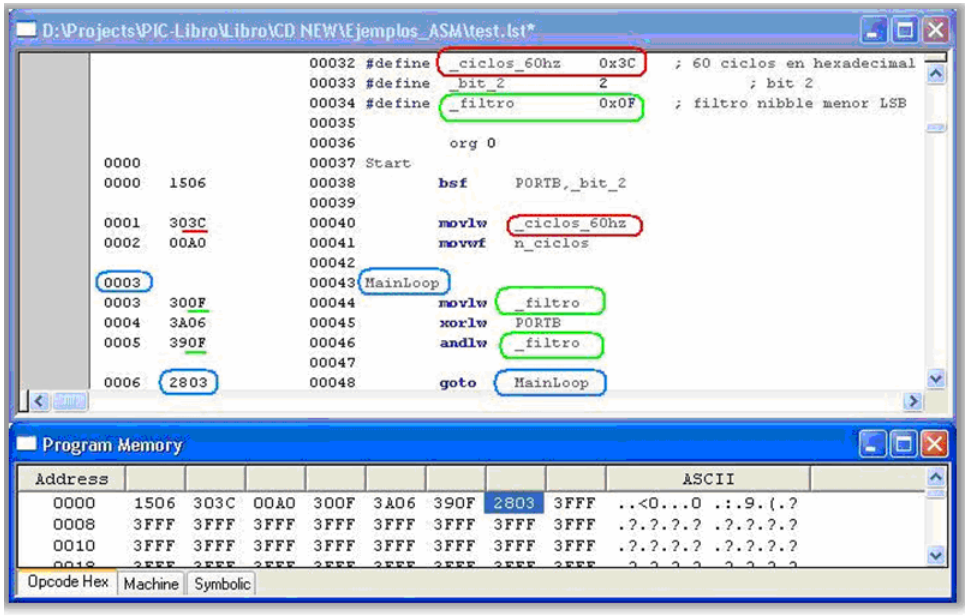


Figura 11.30 Constantes como dato y dirección

COMO DATO

Se define al principio del programa (Figura 11.30):

```
#define  _ciclos_60Hz    0×3C  ; 60 ciclos hexadecimal
#define _filtro          0×0F  ; filtro nibble menor LSB
```

y se reemplaza en cada instrucción que utilice esta constante.

```
movlw       _ciclos_60Hz
andlw       _filtro
```

Al compilar se agrega el valor de 8 bits al **opcode** de la instrucción. Así, la instrucción movlw _ciclos_60Hz se convierte en *303C*, donde *30* es el **opcode** y *3C* es el valor de la constante _ciclos_60Hz que se reemplazó por su valor. Igual para la instrucción andlw _filtro, se convierte en *390F,* donde *39* es el **opcode** y *0F* es el valor de la constante _filtro que se reemplazó por su valor. Cada vez que hay una definición, el compilador reemplaza ese valor definido, en el código de programa, de tal forma que este valor queda insertado en la instrucción, al compilarse el programa.

COMO DIRECCIÓN

Cada vez que ponemos una etiqueta o *label* a una instrucción, el compilador la toma como una dirección, la cual es de 11 bits; se observa en la instrucción goto MainLoop, la dirección **MainLoop** es **0003**, la que se agrega al **opcode** en la codificación de la instrucción, quedando como **2803**.

VARIABLES

Estos datos sí cambian durante el programa. En el programa **se definen** con las **directivas "CBLOCK".**

```
Ejemplo:
        cblock 0×20
Dígito :2              ; posición 0×20, 2 bytes
Dígito_1 :1            ; posición 0×22, 1 byte
Dígito_2 :1            ; posición 0×23, 1 byte
        endc
```

Esto define las variables, pero no las **inicializa**, o sea que quedan definidas para trabajar en el programa, pero no contienen los valores iniciales. Para **inicializar** las variables con valores, procedemos así:

Ejemplo:

```
        movlw   0×A6
        movwf   Dígito_1

        movlw   _ciclos_60Hz
        movwf   Dígito_2

        clrf    contador
        clrf    PORTE
```

En las **instrucciones** se reconoce como **f**. Es un dato, es de 8 bits (0 -255, 0 – 0×FF).

Ejemplo

```
        movwf   f               ; (W) -> f
        movwf PORTD             ; (W) -> PORTD

        addwf   f,d             ; (W) + f -> (d)
        addwf   nivel,F         ; (W) + nivel -> nivel

        decfsz  f,d             ; f – 1 -> (d), salta si f=0
        decfsz  n_ciclos,F      ; n_ciclos -1 -> n_ciclos
```

Estos datos **se ubican** en la **RAM** y pueden ser registros de control del MCU (SFR), o los registros de propósito general (GPR). Los registros que creamos (usuario) se ubican en las posiciones libres de los GPR. Como se ilustra en la Figura 11.31, los registros propios del MCU (SFR: *Special Function Register*), al principio de cada banco y los GPR (*General Purpose Register*), o registros de propósito general, son los registros de usuario, donde se ubicarán las variables creadas en el programa.

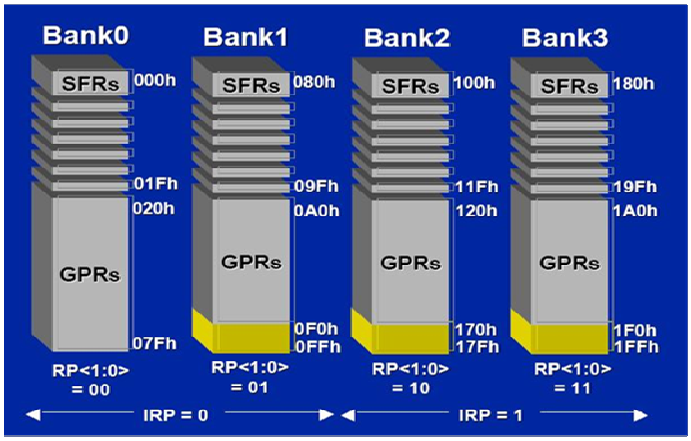


Figura 11.31 Variables en RAM.

11.5.3 DATOS EN TIPOS DE MEMORIA

Se pueden ubicar datos en todos los tipos de memoria; observemos sus características en cada uno de los tipos de memoria:

- **Memoria de Datos.** Datos de variables de control y de usuario (8 bits)
- **Memoria de Datos E2PROM.** Datos no volátiles (8 bits)
- **Memoria de Programa.** Instrucciones (14 bits)

MEMORIA DE DATOS

En esta memoria (Figura 11.32 es donde se guardan los registros de usuario, y se manipulan los Registros de control del MCU.

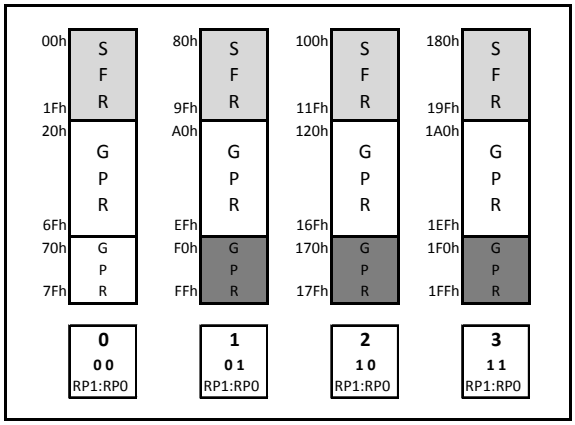


Figura 11.32
Memoria de datos

Los SFR (*Special Function Register*) son los registros encargados del control de todos los módulos del MCU; están al principio de cada uno de los bancos de la RAM; van desde (00 – 1Fh, 0 – 31d) en cada uno de los 4 bancos. Los GPR (*General Purpose Register*) son los registros de propósito general o registros de usuario; en ellos se almacenan las variables creadas en el programa; van desde (20h - 7Fh, 32d – 127d). Estas posiciones están libres para la memoria de usuario, en cada uno de los 4 bancos (Banco0,1,2,3). Estos registros están más detallados en la Parte 2: "Arquitectura", en el Capítulo 6.0, Subsección 6.3.2: "Memoria de Datos". Una amplia descripción de los Datos en RAM se encuentra en la Parte 4: "Memoria", en el Capítulo 15.0, Sección 15.1: "Tipos de memoria", su definición, inicialización y ubicación.

MEMORIA DE DATOS E2PROM

Ésta es la memoria de datos no volátil (Figura 11.33); esto quiere decir que los datos guardados en ella permanecen aunque no haya alimentación de voltaje.

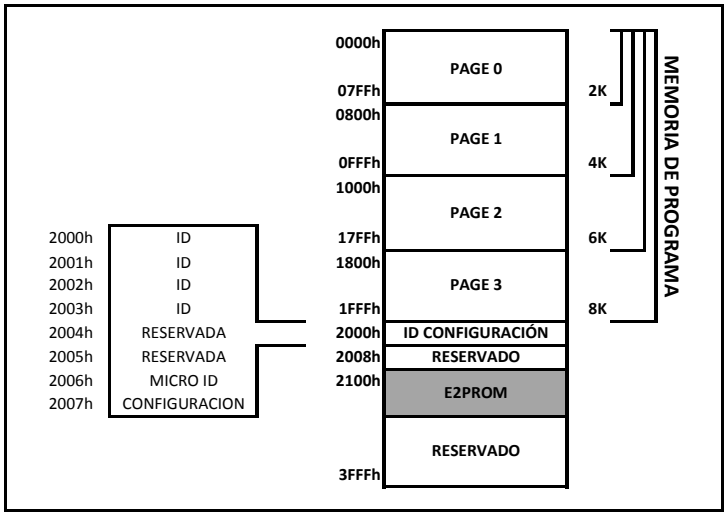


Figura 11.33 Memoria de datos E2PROM.

En la Figura 11.33 observamos la localización de la memoria **E2PROM**, en la posición **2100h,** notamos que se encuentra en una porción más allá de los 8 K que sería el límite de la memoria de programa. Para el usuario, la posición de memoria **E2PROM** comienza en la posición 0. Si tenemos 256 posiciones de memoria para el usuario, estos registros corresponden de **0 – FFh, o 0 – 255d,** sólo que al leer o escribir en la **E2PROM**, el procedimiento asegura que se trabaje en la posición **2100h**.

Se debe aclarar que a los datos de la memoria **EEPROM** sólo se accede mediante un procedimiento y no a un direccionamiento como se hace como en la memoria de datos **GPR**. De tal forma que **no se puede acceder** la posición **2100** y obtener el primer dato de la **EEPROM**. Aun más, no es posible acceder a estas posiciones en modo de **RUN**, sólo mediante un procedimiento. Este aspecto se aclara en la Parte 4: "Memoria", en el Capítulo 14.0, apartado 14.1.3.2: "Memoria EEPROM".

MEMORIA DE PROGRAMA

En esta memoria (Figura 11.34) se encuentran las instrucciones del programa, y los datos en forma de tablas que pueden estar ubicadas en la parte inicial de las 4 páginas de programa que hay.

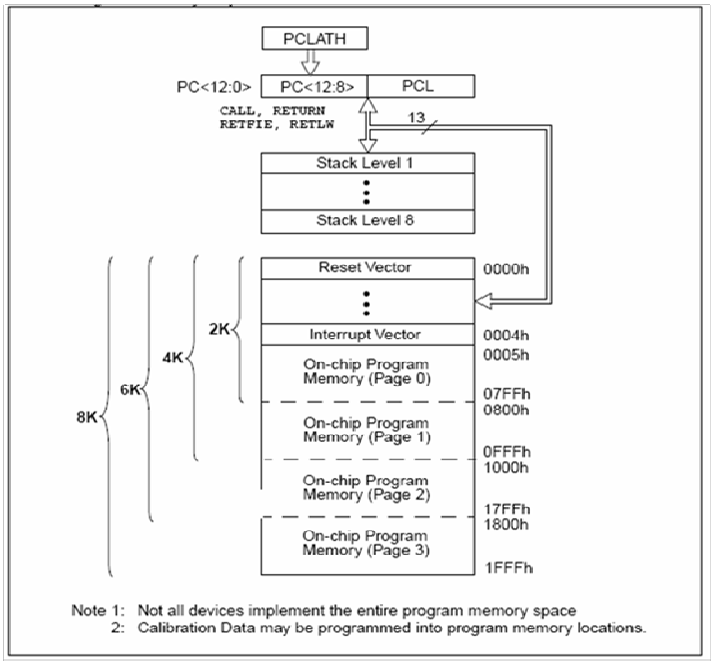


Figura 11.34 Memoria de programa

TABLAS

Las **tablas** (Figura 11.33) deben estar al principio de cada banco, debido a que al realizar la suma con la instrucción addwf PCL, F el valor resultante de la suma no

puede ser mayor de 255 o de 0×FF; si es mayor se sobrescribirán en los primeros valores. Las tablas se componen de:

1. Valores de la tabla
2. Llamado a la tabla

Valores de la tabla

Son todos los valores que integran la tabla, sus partes son:

- La *label* de la tabla es el principio y el nombre de la tabla.
- Luego la instrucción **addwf PCL,F**, que es la que hace un direccionamiento computado, suma el *offset* a la dirección del principio de la tabla.
- Después todos los **valores de la tabla** con la instrucción **retlw xx**, donde *xx* es cada valor de la tabla que debe ser de 8 bits. Cada valor en la tabla se accede con el número de la posición en la tabla, comenzando en 0, 1, 2… y ascendiendo consecutivamente hasta el último valor posible de 255.

Ejemplo:

```
Tabla_Label                  ; nombre o label de la tabla
       addwf PCL,F           ; sume (W) + PCL -> PCL
       retlw _dato_0         ; valor de dato 0
       retlw _dato_1         ; valor de dato 1
       ……..
       retlw _dato_n         ; valor de dato n
```

Cada dato es de 8 bits, del _dato_0 a _dato_n.

Llamado a la tabla

Ésta es la instrucción que llama a la tabla, con el offset cargado en W.

- Cargue el valor de offset en (W).
- Llame a la tabla con la instrucción *call Tabla_Label*.

Ejemplo:

```
       movlw offset          ; cargue offset a (W)
       call Tabla_Label      ; llama al principio de la Tabla
       movwf   PORTD         ; ésta u otra instrucción que utilice el
                             ; valor retornado en W
```

Ésta es la estructura básica de una tabla. En el ejemplo siguiente apreciamos cómo se utiliza una tabla en un programa. El programa muestra los dígitos 0-9 en un *display* de 7 segmentos conectado al Puerto D.

```
Start:
        call Ini_Ptos           ; Inicializa puertos
        clrf contador           ; borre contador
Main:
        movf contador,W         ; cargue contador a W
        call Tabla_code_7_seg   ; llame a Tabla código de 7 segmentos
        movwf PORTD             ; póngalo en PUERTO D

        call Delay              ; Hace retardo
        call Chek_10            ; Verifica si contador=10?
        goto  Main              ; haga ciclo

;-------------------------------------------------------------------------
---------
Tabla_code_7_seg                ; Tabla para display de 7 segmentos.
        addwf   PCL,F
        retlw 3Fh               ; número "0".
        retlw 06h               ; número "1".
        retlw 5Bh               ; número "2".
        retlw 4Fh               ; número "3".
        retlw 66h               ; número "4".
        retlw 6Dh               ; número "5".
        retlw 7Dh               ; número "6".
        retlw 07h               ; número "7".
        retlw 7Fh               ; número "8".
        retlw 67h               ; número "9".
```

Este ejemplo completo lo enontrará usted en nuestra página web http://virtual.alfaomega.com.mx en la carpeta :\Ejemplos_ASM\7_segmentos_1\7_segmentos_1.asm, copiado en el C:\.

11.5.4 FORMATOS DE VALORES

En la Tabla 11.9 se muestra la forma en que se pueden escribir los valores de 8 bits en diferentes formatos.

Tabla 11.9 Variables en RAM.

TIPO	SINTAXIS	EJEMPLO
Decimal	D'<valor>' d'<valor>' .<valor>	movlw D'208' movlw d'208' movlw .208
Hexadecimal	H'<valor>' h'<valor>' 0X<valor> 0x<valor> <valor>H <valor>h	addlw H'E3' addlw h'E3' addlw 0XE3 addlw 0xE3 addlw E3H addlw E3h
Octal	O'<valor>' o'<valor>'	iorlw O'47' iorlw o'47'
Binario	B'<valor>' b'<valor>'	andlw B'10011010' andlw b'10011010'
ASCII	A'<carácter>' a'<carácter>' '<carácter>'	xorlw A'S' xorlw a'S' xorlw 'S'
Cadena de Caracteres	"<string>"	DT "PIC16F8XX"

11.6 PROCESOS ESPECIALES DE PROGRAMA

En el flujo de operación continuo del programa principal se pueden realizar bifurcaciones a otros programas, de dos maneras (procesos especiales de programa), que son:

- **Subrutinas.** Bifurcación a otro programa por llamado desde software.
- **Interrupciones.** Bifurcación a otro programa por llamado desde hardware.

11.6.1 SUBRUTINA

Cuando se crea un programa, partiendo del modelo de trabajo *Top-down*, éste se divide en secciones más pequeñas, de tal forma que pueda quedar modular, y fácilmente descrito para el usuario. Cada una de las tareas que se vayan a realizar se pueden implementar en rutinas pequeñas o **subrutinas**, que luego se pueden usar (llamar, *call*) desde el programa principal para realizar toda la aplicación. Se puede llamar a subrutinas desde el programa principal, o estando en una subrutina, con lo cual se realiza un **anidamiento de subrutinas**. El microcontrolador trae incorporado la facilidad de las subrutinas que describiremos a continuación.

PROCEDIMIENTO DE SUBRUTINA

Las **subrutinas** son pequeñas porciones del programa principal, que las utiliza para realizar completamente la aplicación. Para tener una idea más clara del concepto de subrutinas, observemos este ejemplo: supongamos que tenemos una misma tarea repetida varias veces en un programa, de tal manera que lo que tenemos es una repetición del código de esta tarea en diferentes partes del programa. Esto hace que este programa sea ineficiente debido a que consume mucha memoria de programa.

En la Figura 11.35 observamos la sección de un programa, sin subrutinas (izquierda) y otro con subrutinas (derecha). Se aprecia cómo la **TAREA_X** aparece 3 veces, lo que podría evitarse agrupándola en una sola subrutina, que podemos llamar desde el programa principal donde se necesite.

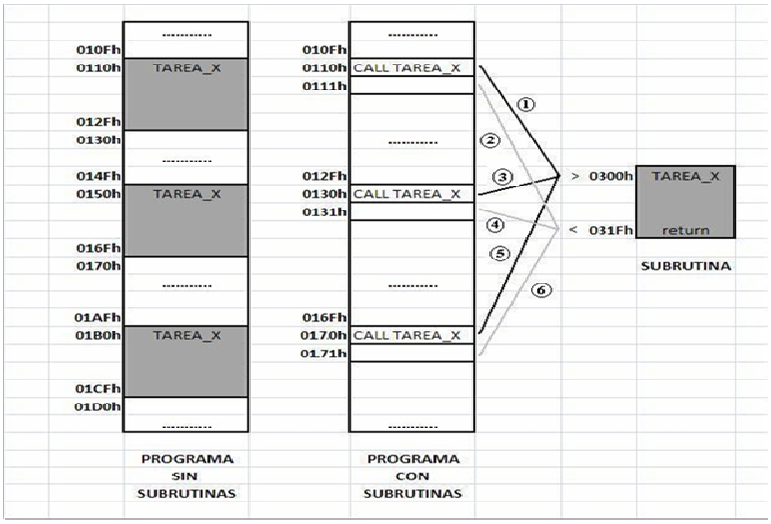


Figura 11.35 Programa com y sin subrutina

La TAREA _X, en el programa sin subrutinas (izquierda en la figura) contiene 1Fh (o 16d) instrucciones, en el programa sin subrutinas, lo que consume 3 × 16 = 48 instrucciones. El mismo programa con subrutina (a su derecha) tiene la subrutina TAREA_X una sola vez y 3 llamados a esta subrutina con las instrucciones call tarea_ más la instrucción return, al final de la subrutina (que causa el retorno), esto nos da 16 + 3 + 1 = 20 instrucciones.

Esto nos da una diferencia de 48 instrucciones en el primer caso sin subrutinas, frente a 20 instrucciones en el segundo caso con subrutinas. Esto sólo nos da una idea del ahorro en instrucciones o memoria de programa; también nos facilita el trabajar en forma de módulos de programa más pequeños. Los componentes de una subrutina (Figura 11.36) son:

- **Nombre de la subrutina**. También llamado label o etiqueta, que finalmente es una dirección de la memoria de programa.
- **Instrucciones**. Instrucciones o cuerpo de la subrutina.
- **Retorno.** Instrucción de retorno, return o retlw k.

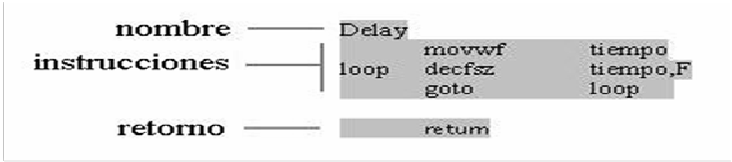


Figura 11.36 Componentes de una subrutina.

El **funcionamiento de la subrutina** es así: el programa principal llama a una subrutina, la ejecuta y luego retorna de ella en el punto en donde la llamó **+1**, para continuar con el programa principal. Estos pasos se resumen en:

- La llamada a la subrutina, instrucción call k, invocada desde el programa principal, guarda la dirección actual **(PC+1) -> Stack**, y carga la dirección destino **(PC) = k**.
- Ejecuta la subrutina, instrucciones contenidas en la subrutina.
- El retorno de la subrutina, instrucción return o retlw k, invocada al final de la subrutina, retoma la dirección guardada en el **Stack, Stack -> (PC)**, y reanuda la ejecución del programa principal.

El flujo del programa principal al llamar a una subrutina (instrucción call k) se interrumpe y salta a otra dirección, guardando la dirección+1 desde donde llama la subrutina; ésta se debe guardar en una memoria llamada apuntador de *Stack*, *Stack pointer* o *Stack*; el contador de programa (**PC)** debe cargarse con la nueva

dirección de la subrutina. A partir de allí el flujo del programa está en la subrutina, de forma que se ejecutan las instrucciones de la subrutina.

Cuando se encuentra la instrucción **return** o **retlw k**, el programa debe retornar el flujo al programa principal; esto lo hace el contador de programa, **PC**, al cargar de nuevo la dirección guardada en el *Stack*, que es la dirección**+1** desde donde se llamó la subrutina; así se restablece el flujo del programa principal.

```
NOTA: Los principales actores en el escenario de las subrutinas son
el contador de programa, PC, y el apuntador de Stack o Stack y las
instrucciones call k, return o retlw k.
```

En la Figura 11.37 se aprecia la secuencia de subrutina, en la columna de PC (contador de programa) están los valores que toma el PC a medida que ejecuta el programa. La columna N contiene el número del proceso en el diagrama de la izquierda, los números 1,3,5 son instrucciones call, y 2,4,6, son instrucciones return. También aparecen el Nivel Stack y el contenido del Stack.

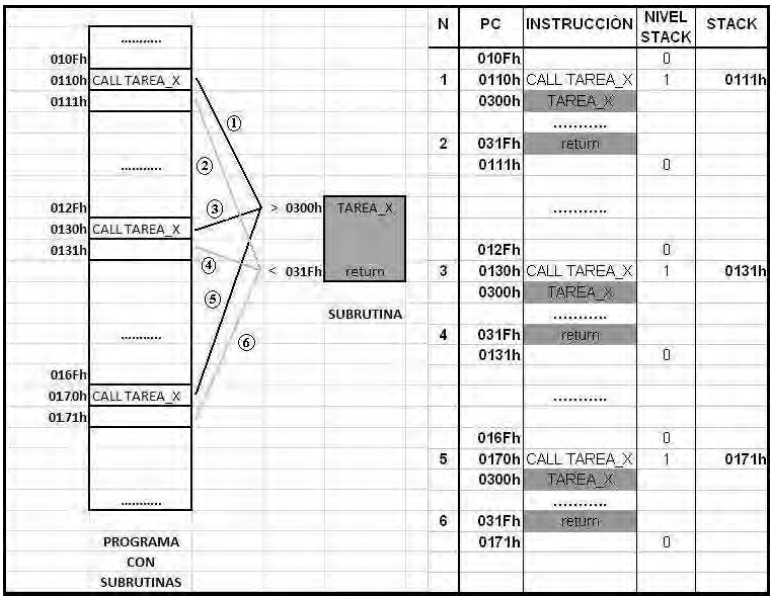


Figura 11.37 Secuencia de subrutina

Si seguimos el proceso **N=1**, que es el llamado a la subrutina, el **PC = 0110h** en la instrucción **call tarea_**, al ejecutarse la instrucción, el Stack guarda la posición **PC+1 = 0111h** en el nivel **1** del *Stack*, y se carga la dirección de la subrutina en el **PC = 0300h**, y a partir de allí se continúan ejecutando las instrucciones de la subrutina, hasta llegar al proceso *N=2*, que es el retorno de la subrutina; en la

dirección **PC = 031Fh** está la instrucción *return*, al ejecutarse la instrucción el Stack trae el valor del **nivel 1** y lo carga en el **PC = 0111h**, y el **PC** va a esta dirección para continuar la ejecución del programa principal.

El mismo procedimiento es el utilizado en los procesos 3-4, 5-6, para llamar de nuevo desde el programa principal a la subrutina **TAREA_X**, ejecutarla y retornar de ella y continuar el programa principal. En el siguiente programa ejemplo se prende y apaga el led **OUT_1** del pin **RB4** por un tiempo de ½ segundo (513 ms = 0.5 s).

Ejemplo:

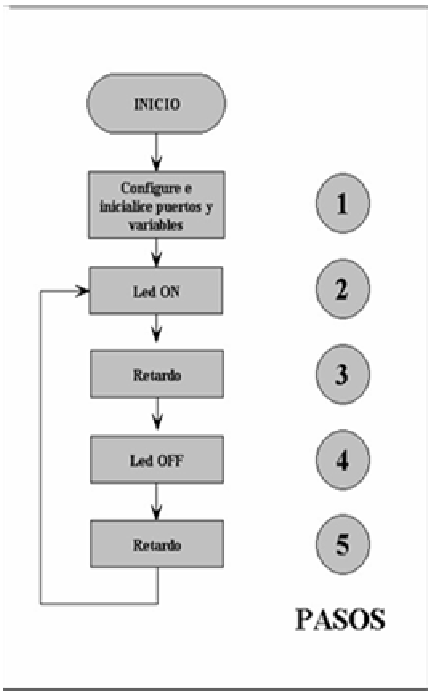


Figura 11.38 Diagrama de flujo sin subrutina.

En la Figura 11.38 observamos que a cada símbolo en el diagrama de flujo le corresponde un paso, encerrado en círculo al lado derecho de la figura. A continuación en el código encontraremos estos **pasos**:

```
;*****************************************************************
INICIO ;***********************

  list    p=16f877a ; define procesador

  #include  <p16f877a.inc> ; definiciones del procesador
```

```
  __CONFIG  _CP_OFF  &  _WDT_OFF  &  _BODEN_OFF  &  _PWRTE_ON  &  _HS_OSC  &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

Hasta aquí se ha definido el MCU con el archivo de definiciones.

```
#define _ini_delay3 0×0C      ; inicializa retardo 3 nivel alto, a 20 MHz
Ton=Toff= 513ms = 0.5 s

  cblock 0×20                 ; define bloque de memoria
delay1                        ; define registros de retardo
delay2
delay3
  endc
```

Aquí se definieron constantes y variables.

```
------------------------------------------------------ PASO 1 ----------
-------------
  org 0
```

Y aquí arranca el programa.

```
Start:
  bsf STATUS,RP0      ; selecciona Banco 1
  bcf TRISB,4         ; Pin RB4 como output
  bcf STATUS,RP0      ; selecciona Banco 0

  clrf  delay1        ; delay1=00h
  clrf  delay2        ; delay2=00h
```

Aquí se configuran el Puerto y las variables.

```
-----------------------------------------------------------PASO 2 ----------
--------------
MainLoop:
Led_ON
  bsf PORTB,4         ; prende LED OUT_1, Pin RB4
```

Aquí se enciende el LED:

```
-----------------------------------------------------------PASO 3 ----------
--------------
  movlw _ini_delay3   ; cargue valor inicial de retardo 3
```

```
  movwf delay3          ; guárdelo en delay3
ON_DelayLoop:
  decfsz delay1,f       ; decremento nivel bajo de retardo
  goto ON_DelayLoop     ;  este  ciclo  bajo  toma  3Tcy  *  256  =  768  Tcy  o
instrucciones
  decfsz delay2,f       ; decremento nivel medio de retardo
  goto ON_DelayLoop     ; este ciclo medio toma 256 * 768 + 3 = 197.376 Tcy
o instrucciones
  decfsz delay3,f       ; decremento nivel alto de retardo
  goto ON_DelayLoop     ; este ciclo alto toma 13 * 197.376 + 3 = 2.565.927
Tcy o instrucciones
                        ; Tcy = 0.2us con xtal = 20MHz, -> 0.513 seg
```

Aquí se hace retardo de 500 ms = 0.5 s:

```
------------------------------------------------------------PASO 4 ----------
--------------
Led_OFF
  bcf PORTB,4           ; apague LED RD0
```

Aquí se apaga el LED:

```
------------------------------------------------------------PASO 5 ----------
--------------
  movlw _ini_delay3     ; cargue valor inicial de retardo 3
  movwf delay3          ; guárdelo en delay3
OFF_DelayLoop:
  decfsz  delay1,f      ; decremento nivel bajo de retardo
  goto  OFF_DelayLoop  ;  este  ciclo  bajo  toma  3Tcy  *  256  =  768  Tcy  o
instrucciones
  decfsz delay2,f       ; decremento nivel medio de retardo
  goto OFF_DelayLoop  ; este ciclo medio toma 256 * 768 + 3 = 197.376 Tcy
o instrucciones
  decfsz delay3,f       ; decremento nivel alto de retardo
  goto OFF_DelayLoop    ; este ciclo alto toma 13 * 197.376 + 3 = 2.565.927
Tcy o instrucciones
                        ; Tcy = 0.2us con xtal = 20MHz, -> 0.513 seg
```

Aquí se hace retardo de 500 ms = 0.5 s:

```
-------------------------------------------------------PASO 6 ----------
-------------
```

```
  goto   MainLoop   ; haga de nuevo
```

Se repite el ciclo:

```
  end
```

Observamos que los pasos 1, 3 y 5 (Figura 11.39) se podrían volver subrutinas, y los pasos 2 y 4 son sólo las instrucciones en encendido y apagado de Led.

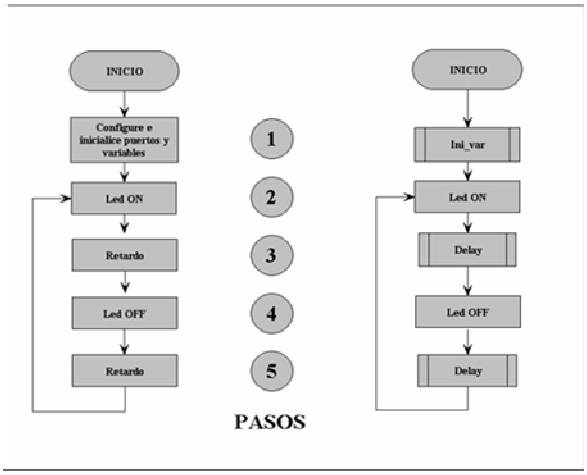


Figura 11.39 Diagrama de flujo con subrutina.

Ahora observamos que se simplifica programa al agrupar código en subrutinas, aunque en el diagrama de flujo (Figura 11.39) solamente se cambian los símbolos de sentencia (lado izquierdo) por los de **llamado a subrutina** (lado derecho):

```
-------------------------------------------------------------------
INICIO ------------------------
```

```
  list    p=16f877a ; define procesador
```

```
  #include  <p16f877a.inc> ; definiciones del procesador
```

```
  __CONFIG  _CP_OFF  &  _WDT_OFF  &  _BODEN_OFF  &  _PWRTE_ON  &  _HS_OSC  &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

Hasta aquí se definió el MCU con el archivo de definiciones.

```
#define _ini_delay3 0×0C       ; inicializa retardo 3 nivel alto, a 20 MHz
Ton=Toff= 513ms = 0.5 s

  cblock 0×20          ; define bloque de memoria
delay1                 ; define registros de retardo
delay2
delay3
  endc
```

Aquí se definieron constantes y variables:

```
  org 0
  goto  Start          ; vaya a programa principal
```

Aquí arranca el programa:

```
------------------------------------------------------------PASO 1 ----------
--------------
;*************************************************************************
********
; Inicicaliza variables
Ini_var
  bsf STATUS,RP0       ; selecciona Banco 1
  bcf TRISB,4          ; Pin RB4 como output
  bcf STATUS,RP0       ; selecciona Banco 0

  clrf  delay1         ; delay1=00h
  clrf  delay2         ; delay2=00h


  return
;*************************************************************************
********
```

Esta **subrutina** configura el **puerto** y las **variables**:

```
------------------------------------------------------PASO 3 y 5----------
-----------


; Retardo de 3 niveles y alcanza hasta 513 ms = 0.5 s
; delay1=delay2=FFh, delay3=0Ch=_ini_delay3
Delay
  movlw _ini_delay3    ; cargue valor inicial de retardo 3
```

```
  movwf delay3          ; guárdelo en delay3
DelayLoop:
  decfsz delay1,f       ; decremento nivel bajo de retardo
  goto DelayLoop        ; este ciclo bajo toma 3Tcy * 256 = 768 Tcy o
instrucciones
  decfsz delay2,f       ; decremento nivel medio de retardo
  goto DelayLoop        ; este ciclo medio toma 256 * 768 + 3 = 197.376 Tcy
o instrucciones
  decfsz delay3,f       ; decremento nivel alto de retardo
  goto DelayLoop        ; este ciclo alto toma 13 * 197.376 + 3 = 2.565.927
Tcy o instrucciones
                        ; Tcy = 0.2us con xtal = 20MHz, -> 0.513 s
  return
;*********************************************************************
********
```

Aquí se hace retardo de 500 ms = 0.5 s.

Éste es el programa principal:

```
Start:
  call Ini_var          ; llame a subrutina Ini_var
MainLoop:
Led_ON
  bsf PORTB,4           ; prende LED OUT_1, Pin RB4
  call Delay            ; llame a subrutina Delay
Led_OFF
  bcf PORTB,4           ; Turn off LED RD0
  call Delay            ; llame a subrutina Delay
  goto MainLoop         ; Do it again...
  end
```

Se agrupan las instrucciones del paso 1 en la subrutina *Ini_var*, las instrucciones de los pasos 3 y 5 en la subrutina *Delay*.

NOTA: Al colocar las subrutinas es mejor ponerlas en la parte inicial del programa y luego el programa principal.

Note cómo, luego de la instrucción **org 0**, se coloca la instrucción **goto Start,** lo que asegura que se comienza en el programa principal, aunque no esté en la primera instrucción. Después de estas instrucciones están las subrutinas.

```
  org 0
  goto Start          ; vaya a programa principal
```

Observamos que en el código del programa principal (*Start*), las instrucciones se disminuyen a sólo 6, con la ayuda de los llamados a subrutina. También se aprecia que se puede hacer una lectura mucho más rápida y global del entendimiento del programa.

LÍMITE DE STACK

El *Stack* (Tabla 11.10) es la memoria que guarda la dirección del *PC* (*Program Counter*) cuando se llama una subrutina, y devuelve esa dirección al *PC* al retornar de la subrutina.

Tabla 11.10 Límite de stack y desbordamiento

NIVEL STACK	STACK	
1	DIR 8	0569h
2	DIR 7	0534h
3	DIR 6	0505h
4	DIR 5	04B8h
5	DIR 4	0452h
6	DIR 3	0390h
7	DIR 2	0333h
8	DIR 1	020Ch

LIMITE DE STACK 8 NIVELES

NIVEL STACK	STACK	
1	DIR 10	05E2h
2	DIR 9	05A4h
3	DIR 8	0569h
4	DIR 7	0534h
5	DIR 6	0505h
6	DIR 5	04B8h
7	DIR 4	0452h
8	DIR 3	0390h
9	~~DIR 2~~	~~0333h~~
10	~~DIR 1~~	~~020Ch~~

DESBORDAMINETO

El *Stack* es una memoria de 8 niveles tipo LIFO (*Last In, First Out*), de tal manera que lo último en entrar es lo primero en salir. Cada vez que hay una llamada a una subrutina o una instrucción *call* entra un nivel el *Stack* (*Push*), y cada vez que hay un retorno de una subrutina o instrucción return o retlw k, el *Stack* sale un nivel (*Pull*). Esta memoria guarda direcciones del PC o memoria de programa; por tanto, es de 13 bits de ancho.

El límite del *Stack* son 8 niveles, como se observa en la Tabla 11.10; cuando se excede este nivel ocurren un desbordamiento y una pérdida de los primeros niveles. Es decir, pierde la forma de volver a las direcciones de esos llamados a subrutina. Esto causa una pérdida del flujo de programa, y es bastante peligroso pues no se sabe el comportamiento del micro en estas condiciones.

```
NOTA: El MCU no da avisos de un desbordamiento del Stack; por lo
tanto, queda al cuidado del desarrollador.
```

ANIDAMIENTO DE SUBRUTINAS

Podemos llamar a una subrutina estando en el programa principal o estando en una subrutina, en cuyo caso lo llamamos *anidamiento*; podemos hacer este procedimiento hasta copar el nivel del Stack; en nuestro caso es de 8 niveles. Lo que significa que podemos hacer llamados a subrutinas anidadas (Figura 11.40) hasta 8 niveles.

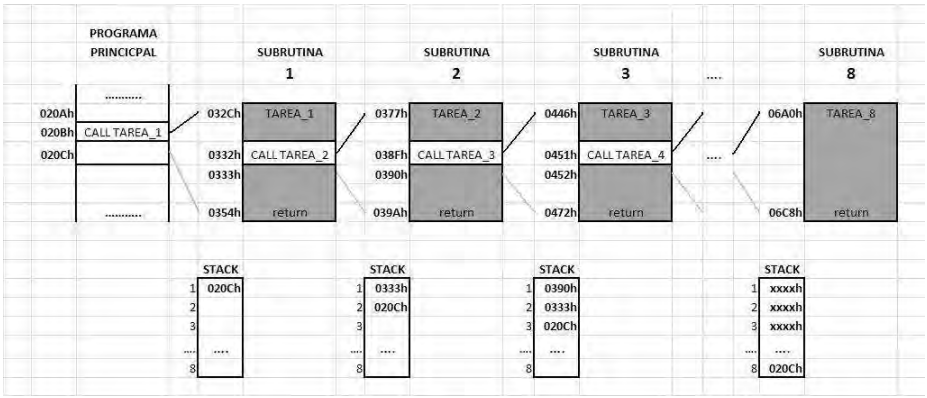


Figura 11.40 Secuencia de subrutinas anidadas

En la Figura 11.39 se puede detallar este suceso; observe que en cada llamado a subrutina puede haber una llamada a otra subrutina, y cada vez el nivel de *Stack* va aumentando de nivel; así, el primer llamado a subrutina ocasiona un primer nivel de **Stack = 020Ch**; el segundo llamado a subrutina ocasiona un segundo nivel de **Stack= 0333h;** el tercer llamado a subrutina ocasiona un tercer nivel de **Stack= 0390h**, y así sucesivamente hasta llegar al nivel 8. No se puede permitir que el programa llene más de los 8 niveles de *Stack*; si esto pasa ocurrirá un desbordamiento, y eso pierde totalmente el flujo del programa.

NOTA: El micro no genera ninguna alerta por desbordamiento. Nunca debe permitirse un desbordamiento del Stack, debido a que el programa pierde la posibilidad de volver a las direcciones que tenía almacenadas. Esto pierde el flujo del programa y el micro puede realizar acciones no deseadas.

11.6.2 INTERRUPCIONES

Como su nombre lo indica, son operaciones que interrumpen el flujo normal del programa, haciendo que se desvíe hacia la dirección del vector de interrupción, y allí se encuentra la rutina de servicio de interrupción; se ejecuta y luego retorna el flujo del programa donde fue interrumpido. Cuando ocurre la interrupción el programa termina de ejecutar la presente instrucción y guarda la dirección +1 en el stack; al terminar de ejecutar la rutina de servicio de interrupción el programa vuelve a esta dirección +1, donde continúa su flujo normal.

FUENTES DE INTERRUPCIÓN

Este microcontrolador tiene 15 fuentes de interrupción como se vio en el Capítulo 9.0, Sección 9.3: "Interrupción" , en donde pueden detallarse cada uno de los módulos que generan interrupción. Todos los módulos periféricos generan interrupciones (Figura 11.41), y cada uno de ellos tiene su bit de control respectivo, los cuales veremos a continuación.

FIGURE 14-10: INTERRUPT LOGIC

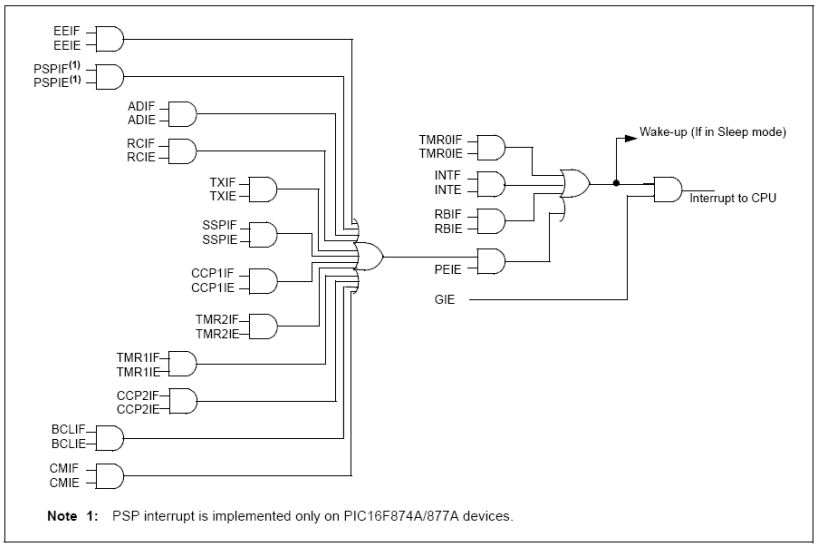


Figura 11.41 Circuitería lógica de las interrupciones

Como se observa en la Figura 11.41, cada módulo tiene dos bits que van a una compuerta *and*, la que sólo genera una salida si ambos bits están en **'1'**; esto solamente ocurre cuando la interrupción está habilitada y la bandera de interrupción está en **'1'**. En el cuadro de la Figura 14.42 se aprecian todas las fuentes de interrupción de los diferentes módulos. Estas fuentes están gobernadas por los bits **PEIE** y **GIE** según el tipo de enmascaramiento **BIT MASK**.

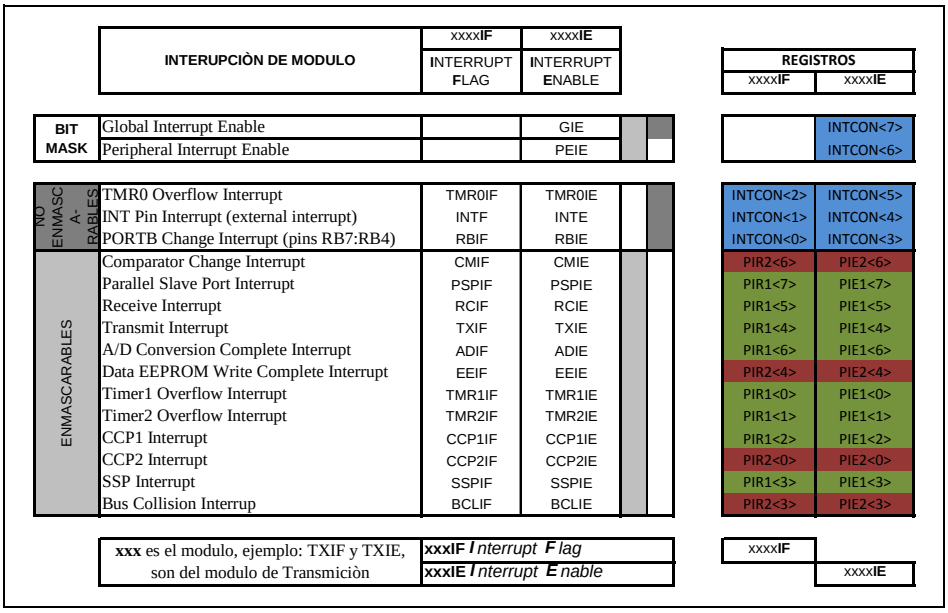


Figura 11.42 Interrupciones y sus banderas.

Cada interrupción posee dos bits:

- El bit **xxxIF Interrupt Flag,** es la bandera de interrupción y se pone en (1) al ocurrir el evento del módulo.
- El bit **xxxIE Interrupt Enable**, es la habilitación de la interrupción; el usuario al ponerla en (1) habilita la interrupción. Si se deja en (0) no habilita interrupción.

```
NOTA: xxx es el módulo, ejemplo: TXIF y TXIE, son del módulo de
Transmisión.
```

También se destacan dos tipos de interrupciones:

- Enmascarables
- No enmascarables

```
NOTA: El término enmascarable proviene de mask, máscara, la cual
permite o no ver el valor real.
```

Las **enmascarables** se pueden habilitar por medio de los bits **PEIE** y **GI**E; si estos dos bits no están habilitados ("1") no se genera interrupción. Como ejemplo, para que se genere la interrupción **Timer2 Overflow Interrupt** del módulo Timer debe ocurrir lo siguiente:

- **GIE=1**, interrupción global habilitada.
- **PEIE=1**, interrupción de periféricos habilitada.
- **TMR2IE=1**, interrupción TMR2 habilitada.
- **TMR2IF=1**, el evento ocurrió, *overflow*.

Las **no enmascarables** no necesitan estar habilitadas con el bit **PEIE** para generar interrupción (o despertar el micro *wake-up* del estado de *sleep*), y sólo necesita estar el bit **GIE** habilitado (1) para generar interrupción. Para que se genere la interrupción **Timer0 Overflow Interrupt** del módulo Timer debe ocurrir lo siguiente:

- **GIE=1**, interrupción global habilitada.
- **TMR0IE=1**, interrupción TMR0 habilitada.
- **TMR0IF=1**, evento ocurrió, *overflow*.

Para **despertar** (*wake-up*) el micro con la interrupción **Timer0 Overflow Interrupt** debe ocurrir lo siguiente:

- El micro en estado de *sleep.*
- **TMR0IE=1**, interrupción TMR0 habilitada.
- **TMR0IF=1**, evento ocurrió, *overflow*.

La habilitación de estos bits se puede deducir de los cuadros sombreados en la Figura 11.42; supongamos que queremos habilitar el módulo CCP2 interrupt; como está en el módulo gris claro, se deben habilitar los bits **xxxxIE** gris claro, o sea **CCP2IE, PEIE** y **GIE**. Si queremos habilitar el módulo **PORTB Change Interrupt**, como está en el módulo gris oscuro, se deben habilitar los bits **xxxxIE** gris oscuro, o sea **RBIE** y **GIE**.

COMPONENTES DE LA SUBRUTINA DE INTERRUPCIÓN

Veamos los componentes de una subrutina de interrupción (Figura 11.43):

- **Nombre de la subrutina de interrupción.** También llamada *label* o etiqueta, que finalmente es una dirección de la memoria de programa.
- **Instrucciones.** Instrucciones de la subrutina de interrupción.
- **Retorno**. Instrucción **retfie**.

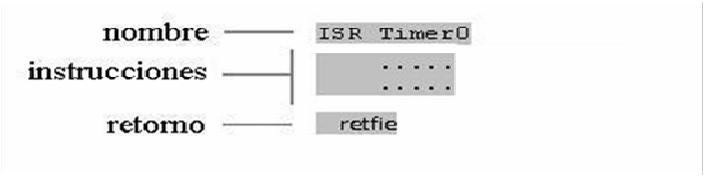


Figura 11.43 Componentes subrutina de interrupción.

Detallando cada uno de estos componentes, tenemos:

NOMBRE DE LA SUBRUTINA DE INTERRUPCIÓN

El nombre sigue las mismas reglas para las *label* o las etiquetas del programa; se recomienda esta sintaxis: **ISR_módulo** donde **ISR_** significa rutina de servicio de interrupción, y **módulo**, el nombre del módulo fuente de interrupción.

Ejemplos:

- **ISR_Timer0** rutina de servicio de interrupción del módulo Timer0.
- **ISR_ADC** rutina de servicio de interrupción del módulo ADC.

INSTRUCCIONES

Es necesario que al principio y al final de la rutina de servicio de interrupción se coloquen estas instrucciones:

1. Haga PEIE=0, deshabilita una nueva interrupción (este paso en caso de que la fuente de interrupción sea enmascarable).
2. Realice las instrucciones de la subrutina de interrupción propias del usuario.
3. Borre la bandera de interrupción bit **IF Interrupt Flag**, del módulo de fuente de interrupción.

4. Haga PEIE=1, habilita interrupción enmascarable (se agrega o no con paso 1).
5. *Retorno*. Instrucción **retfie**. Este paso es el último.

PROCEDIMIENTO DE INTERRUPCIÓN

Para que haya un procedimiento completo de una interrupción se debe contar con cada uno de los pasos siguientes:

- Debe haberse realizado la rutina de interrupción, con el nombre o *label* al principio y al final de ella la instrucción **retfie**.
- Debe ponerse la rutina de interrupción en la posición **0004h** o la instrucción **goto k,** donde **k** es la dirección o *label* de la rutina de interrupción.
- Hay que setear (1) los bits de habilitación de interrupción necesarios para generar la interrupción (véase la Figura 11.42 Interrupciones y sus banderas)
- Debe ocurrir el evento de interrupción del módulo seleccionado.

A continuación (Figura 11.44) detallaremos una secuencia completa de los pasos que se siguen al ocurrir una interrupción.

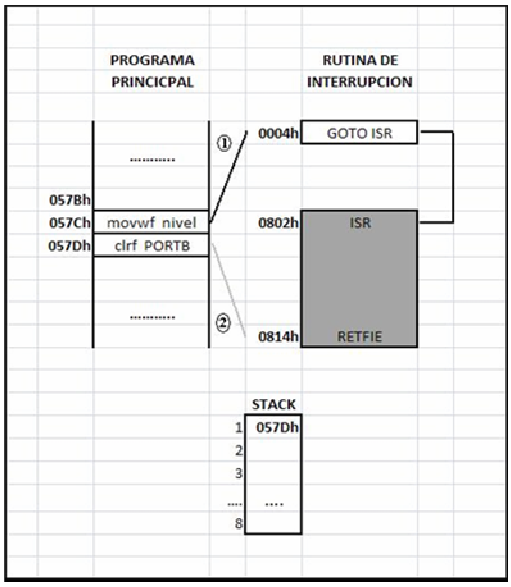


Figura 11.44 Interrupción

En la Figura 11.43 se observa cómo el flujo del programa principal cambia al ocurrir una interrupción en el **paso 1**; esta dirección **+1** donde ocurre la interrupción se guarda en el *Stack* de primer nivel **057Dh**; luego se dirige hacia el vector de interrupción en la dirección **PC=0004h**; de aquí salta *goto ISR* a la dirección de la rutina de servicio de interrupción o ISR (*Interrupt Service Rutine*) que está en la dirección **PC=0802H** (dirección del ejemplo); una vez aquí se realizan las instrucciones de la rutina ISR; al terminar se ejecuta la instrucción **retfie** el **paso 2** que retorna el flujo al programa principal; el *Stack* carga el **PC=057D** que es la dirección **+1** desde donde se generó la interrupción, y el flujo del programa principal se restablece.

```
    Ejemplo:
      list    p=16f877a              ; directiva para definir procesador
  #include  <p16f877a.inc>    ; definiciones de variables específicas del
procesador
  __CONFIG  _CP_OFF  &  _WDT_OFF  &  _BODEN_OFF  &  _PWRTE_ON  &  _HS_OSC  &
_WRT_OFF & _LVP_OFF & _CPD_OFF


       org 0
       goto           Main
;-------------------------------------------------------------- ISR_int --
-------------------
       org 0004h
ISR_int                               ; rutina de servicio de interrupción

       btfsc   PORTB,4         ; si OUT-1 = 0 apagado, vaya a Led1_on
       goto    Led1_off              ;  si  OUT-1  !=  0  prendido,  vaya  a
Led1_off
Led1_on
       bsf  PORTB,4    ; prende led OUT-1
       bcf     PORTB,0         ; OUT-1 = 0, asegura valor=0
       goto Led1_end           ; fin ciclo


Led1_off
       bcf  PORTB,4    ; apaga led OUT-1
       bsf     PORTB,0         ; OUT-1 = 1, asegura valor=1
Led1_end
       bcf     INTCON,INTF           ;   apaga   bandera   de   interrupción
módulo INT
```

```
        retfie
;-------------------------------------------------------------- MAIN -----
----------------
Main:
                                        ; configura puertos
        bsf   STATUS,RP0        ; seleccione Banco 1
        bsf   TRISB,0           ; configure pin RB0, entrada, INPUT-1
        bsf   TRISB,1           ; configure pin RB1, entrada, INPUT-2
        bcf   TRISB,4           ; configure pin RB4, salida, led OUT-1
        bcf   TRISB,5           ; configure pin RB5, salida, led OUT-2
        bcf    OPTION_REG,INTEDG      ; borde de caída de RB0/INT
        bcf   STATUS,RP0        ; seleccione Banco 0

        bsf    INTCON,INTE              ;  habilite  INT,  interrupción  módulo
INT
        bsf    INTCON,GIE               ;    habilite    GIE,    interrupciones
globales

        ; inicializa puertos
        bsf   PORTB,0    ; apague INPUT-1
        bsf   PORTB,1    ; apague INPUT-2
        bcf   PORTB,4    ; apague led OUT-1
        bcf   PORTB,5    ; apague led OUT-2

MainLoop
        btfsc   PORTB,1         ; si INPUT-2 = 0 oprimido, vaya a Led2_on
        goto    Led2_off                ; si INPUT-2 != 0 no oprimido, vaya a
Led2_off

Led2_on
        bsf   PORTB,5    ; prende led OUT-2
        goto   MainLoop   ; haga ciclo

Led2_off
        bcf   PORTB,5    ; apaga led OUT-2
```

```
        goto   MainLoop   ; haga ciclo

        end
```

El programa anterior prende el led **OUT-1** que se alterna cada vez que se presiona el interruptor o switch **INPUT-1**; la interrupción **ISR_int** realiza esto mediante el pin **RB0/INT** en su borde de caída. El led OUT-2, al presionar el interruptor o switch **INPUT-2**, y lo apaga al soltarlo; esto lo realiza en el programa *Main*. También se puede observar que primero está el código de la rutina de servicio de interrupción llamada **ISR_int** en el vector de interrupción posición **org 0004h,** y luego el programa *Main*. Sin embargo, en el vector de reset posición **org 0** se encuentra la instrucción *goto Main* que direcciona el programa a la posición en la que se encuentra la *label Main*.

En el material para esta obra ubicado en nuestra página web: http://virtual.alfaomega.com.mx puede encontrar este ejemplo en la ruta: \Ejemplos_ASM\Interrupt_1\Interrupt_1.asm, el cual se debe copiar en el **C:**\.

EVALUACIÓN

```
P.: 1. ¿Cuáles son los TIPOS DE DIRECCIONAMIENTO?
R:
DIRECCIONAMIENTO INMEDIATO
DIRECCIONAMIENTO DIRECTO
DIRECCIONAMIENTO BIT A BIT
DIRECCIONAMIENTO INDIRECTO
DIRECCIONAMIENTO INDEXADO

P.: 2. En las instrucciones la constante k ¿qué valores toma?
R:
Si es un dato, es de 8 bits (0 -255, 0 – 0×FF).
Si es una dirección, es de 11 bits (0 -2047, 0 – 0×7FF).

P.: 3. ¿Cuáles son las partes que contiene una subrutina?
R:
a) Nombre de la subrutina de interrupción. También llamada label o
   etiqueta, que finalmente es una dirección de la memoria de
   programa.
b) Instrucciones. Instrucciones de la subrutina de interrupción.
c) Retorno. Instrucción retfie.

P.: 4. Hay dos tipos de interrupciones:
R:
a)  Enmascarables
b)  No enmascarables
```

```
P.: 5. ¿Con cuáles bits se habilitan las interrupciones
enmascables?
```

```
R:
Con el bit PEIE (Peripheral Interrupt Enable), en el registro
INTCON<6>, y con el bit GIE (Global Interrupt Enable), en el
registro INTCON<7>

P.: 6. Mencione las tres interrupciones no enmascarables
R:
a)  TMR0 Overflow Interrupt
b)  INT Pin Interrupt (external interrupt)
c)  PORTB Change Interrupt (pins RB7:RB4)
```

12.0 PROGRAMADOR/DEPURADOR - PICKIT 2

Ésta es la herramienta de tipo **programador/depurador** escogida en este libro, debido a todas las bondades que nos brinda para el trabajo con microcontroladores Microchip, además de ser una herramienta de bajo costo. La herramienta **PICkit 2** es un programador/depurador que tiene varias herramientas incorporadas, las que tienen diferentes modos de operación a través de los pines del conector ICSP y con mediación de aplicaciones de software en PC.

12.1 DESCRIPCIÓN DE PARTES

Iniciamos describiendo las partes de la unidad PICkit 2 (Figura 12.1a, b):



Figura 12.1 Descripción de partes PICkit 2 y pines ICSP.

En la Figura 12.1 encontramos la unidad PICkit 2 con sus partes a la derecha, las que describimos a continuación:

1. **Leds de Estado.** Indican el estado de la unidad; tiene 3 leds Power, Target y Busy.
   - **Power (verde).** La potencia se aplica a la unidad vía el puerto USB.
   - **Target (amarillo).** El PICkit 2 está alimentando la tarjeta.
   - **Busy (rojo).** El PICkit 2 está ocupado con una función en progreso, como programando.
2. **Pulsador.** Tiene diferentes usos que encontrará descritos en el manual del usuario del PICkit 2.
3. **Sujetador.** Es un práctico sujetador físico.
4. **Conexión USB.** Es la conexión entre el PC y la unidad a través del cable USB.
5. **Marcador pin 1.** Esta guía indica cuál es el pin 1 en el conector ICSP.
6. **Conector ICSP.** Es el conector que va hacia la tarjeta, por él se establece la comunicación entre el PC y el micro a través de la herramienta PICkit 2 (esto es lo que se denomina propiamente ICSP, la conexión directa a la board de aplicación).

12.2 DESCRIPCIÓN DE PINES ICSP

El protocolo ICSP (*In Circuit Serial Programming*) permite comunicación entre la tarjeta de aplicación que contiene el micro, la herramienta PICkit 2 y el PC. Éste es un estándar de Microchip para sus herramientas de desarrollo con la que se puede comunicar el Software MPLAB con los microcontroladores, permitiendo las funciones de programación/depuración y otras adicionales que trae la herramienta. Este protocolo, en su parte física, contiene un conector de 6 pines con 100mils de separación, y su disposición se aprecia en la Figura 12.1; los pines son:

1. **V_{PP}/-MCLR**
   Voltaje de programación
2. **V_{DD} Tarjeta**
   Voltaje de alimentación Positivo (referencia)
3. **V_{SS} (GND)**
   Voltaje de alimentación Negativo (referencia)

4. ICSPDAT/PGD

Pin de Datos

5. ICSPCLK/PGC
Pin de Reloj

6. Auxiliar Auxiliar

Estos pines toman diferentes funciones según el modo de operación que se está realizando con la herramienta; para mayor información consulte el manual de operación del PICkit 2, que se encuentra en nuestra página web: http://virtual.alfaomega.com.mx en la ruta :\ Herramientas\PICkit 2\Manuales.

12.3 MODOS DE OPERACIÓN

La herramienta PICkit 2 tiene varios modos de operación, cada uno de los cuales requiere un software en PC para su trabajo; la función de cada pin del conector ICSP puede variar según el modo de operación, es decir, cada modo de operación establece la función de cada pin en el conector ICSP; por lo tanto, habrá diferentes conexiones a la tarjeta; más adelante se describen junto al software necesario para la operación. Los modos de operación de la herramienta PICkit 2 son los siguientes:

- *Programmer* (Programador de PICmicro MCU).
- *Logic Tool* [Aplicación (Software)].
  - — *Logic I/O mode* (Generador de estímulos de 4 pines).
  - — *Logic Analyzer mode* (Analizador lógico de 3 canales).
- *UART Tool* (*Sniffer* de RS232 en bajo nivel) (5 V).
- *Programmer-To-Go* (Programador de campo *stand-alone*).

Cada uno de estos modos de operación también requiere un software asociado o una aplicación que determina la forma de funcionamiento. A continuación describiremos cada uno de estos modos. Para mayor detalle de funcionamiento de esta herramienta, lo remitimos al manual del usuario del PICkit 2, que se encuentra en nuestra página web: http://virtual.alfaomega.com.mx en la carpeta: \Herramientas\PICkit 2\Manuales.

12.3.1 PROGRAMADOR (PROGRAMMER)

Ésta es una aplicación de la herramienta, comúnmente utilizada para la programación de los micros.

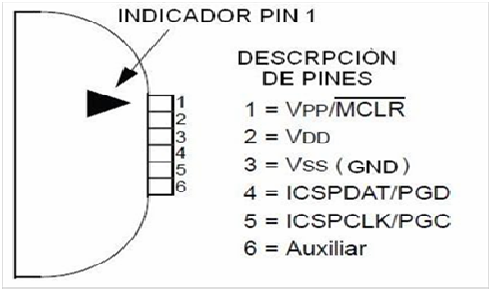


Figura 12.2 ICSP en modo de programador.

Los pines funcionan como originalmente están designados (Figura 12.3); se deben conectar a la tarjeta en un conector de 6 pines para que permitan la programación.

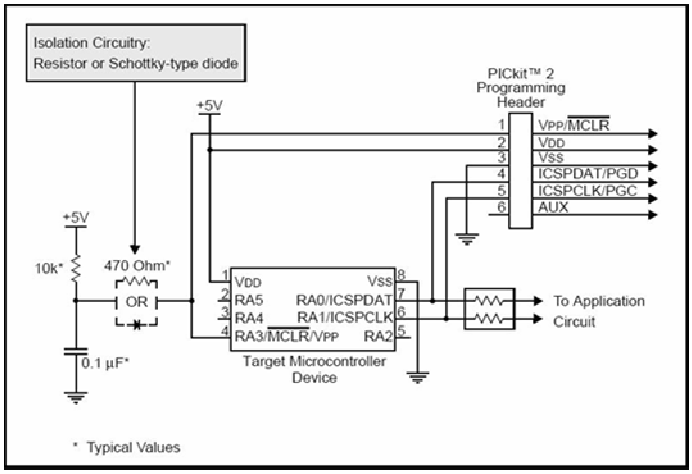


Figura 12.3 ICSP circuito de conexión.

El circuito de conexión del ICSP se muestra en la Figura 12.3 con la descripción de la conexión de cada uno de los pines del conector ICSP y los pines del micro a los cuales van conectados. Se debe resaltar el circuito de reset, debido a que allí se generan 14 VDC en el pin y se hace necesario proteger los +5 VCD con la red que allí aparece.

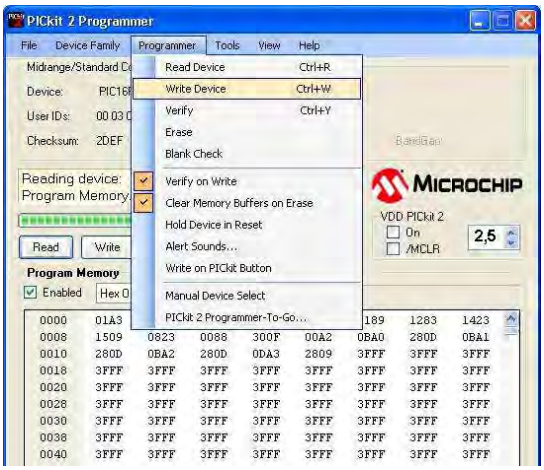


Figura 12.4 Aplicaciones para programador PICkit 2.

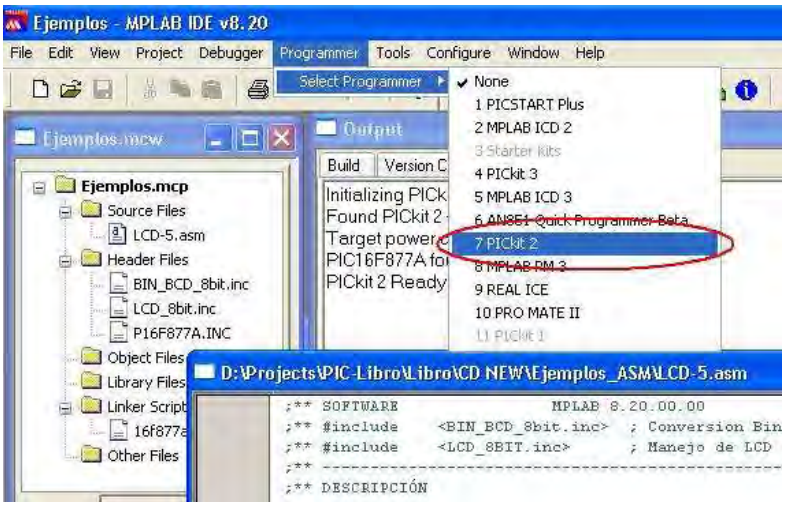


Figura 12.4 Aplicaciones para programador PICkit 2.

Este programador con PICkit 2 se puede llamar desde las 2 aplicaciones de Microchip. Una es aquella con la que viene la herramienta PICkit 2, su nombre es “PICkit 2 programmer” (Figura 12.4 a) y la otra es el ambiente de desarrollo MPLAB (Figura 12.4 b); en su menú de Programmer, aparece el PICkit 2 si ya ha sido previamente instalado. En e Capítulo 14.0, Sección 14.4: “Programación”, se da una explicación detallada sobre el uso de este programador.

12.3.2 MODO LÓGICO I/O (LOGIC I/O MODE)

Ésta es una aplicación de la herramienta, que permite utilizar 4 pines del ICSP como pines I/O que generan estímulos sobre una tarjeta, de tal manera que podemos leer o escribir estados lógicos en los pines. Cada uno de estos pines se puede configurar como Entrada (*Input*) "I", o como Salida (*Output*) "O", con sus niveles lógicos de V_{DD} = "1" o V_{SS} = "O"; de esa manera se escriben o generan estímulos en los pines de salidas, y se leen niveles lógicos en los pines configurados como entradas.

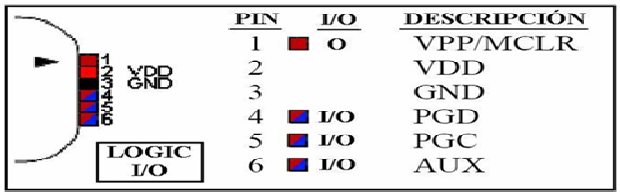


Figura 12.5 ICSP pines en modo Lógico I/O.

En la Figura 12.5 se aprecian los pines del conector ICSP en el modo Lógico I/O; observe que tienen una función diferente de la original. En la Figura 12.6 se aprecia el ambiente de la aplicación en modo Lógico I/O; y se observa cómo se pueden programar los pines y el valor que pueden tomar.



Figura 12.6 Aplicación modo Lógico I/O.

Observe que en la parte superior derecha de la Figura 12.6 están las casillas de los dos modos *Logic I/O* y *Analyzer,* los cuales podemos alternar desde aquí; estos dos modos hacen parte del ambiente o software *Logic Tool.* Llamamos la

herramienta de *Logic Tool* desde la aplicación del programador PICkit 2, en la opción de *Tools*, como se aprecia en la Figura 12.7, donde vemos el despliegue del menú *Tools*.

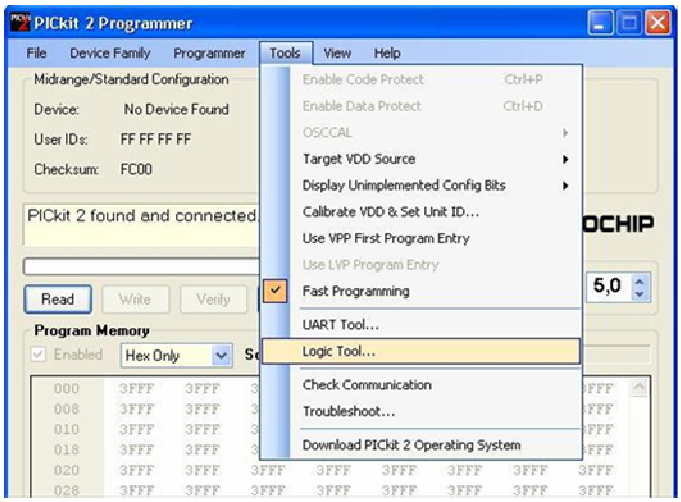


Figura 12.7 Llamado a modo Lógico I/O.

12.3.3 MODO ANALIZADOR LÓGICO (LOGIC ANALYZER MODE)

Ésta es una aplicación de la herramienta, que permite utilizar 3 pines del ICSP como canales del **analizador lógico**, comportándose como un osciloscopio de baja resolución, el cual nos ayuda a visualizar el comportamiento de las señales (canales) en el tiempo. Todos los pines son configurados de entrada y lógicos, de tal manera que sólo puede leer estados lógicos (**"1"** o **"0"**), de rangos entre V_{DD} = "1" o V_{SS} = "0".

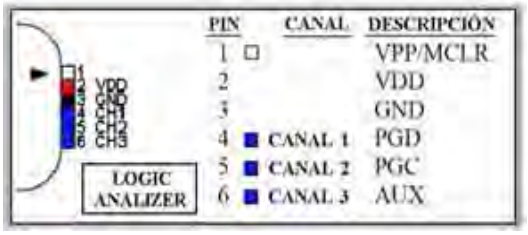


Figura 12.8 ICSP pines en modo analizador lógico.

En la Figura 12.8 se aprecian los pines del conector ICSP en el Modo Analizador Lógico; observe que tienen una función diferente a la original.

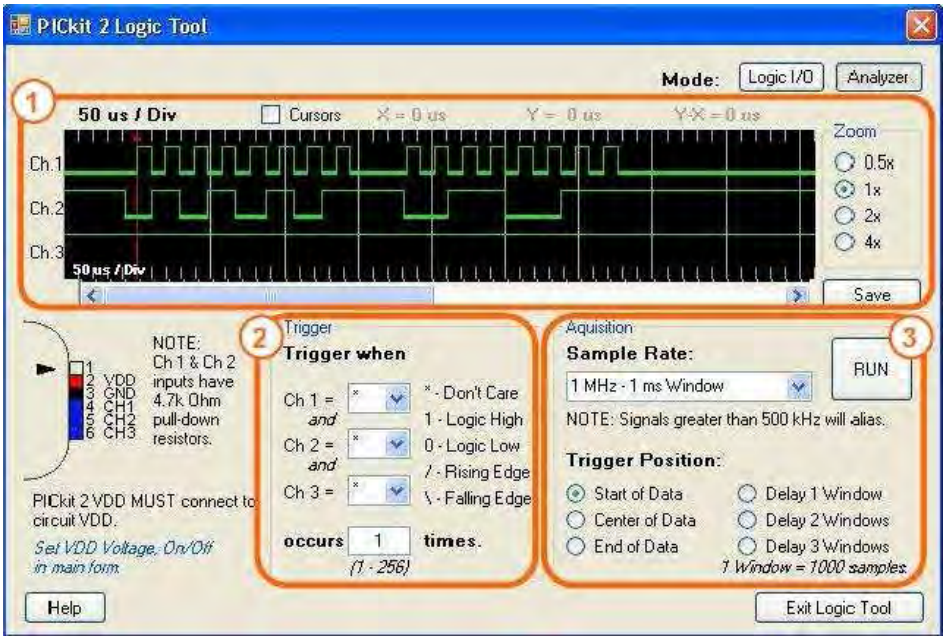


Figura 12.9 Aplicación Modo Analizador Lógico.

Una vez conectados los pines a las señales que se desean ver, se puede seleccionar la forma de *trigger* de cada canal y se escogen los parámetros de muestreo. De esta manera se pueden muestrear las señales recogidas en los pines que están asociados con los canales, y estos datos se ponen en el display (Figura 12.9). También es posible cambiar de escala y también moverse en la pantalla. Otra opción muy útil es grabar el muestreo total en archivos *.bmp. El **analizador lógico** es otro de los modos de la aplicación *Logic Tool* y se escoge en el mismo menú desde la aplicación del programador PICkit 2, como ya hemos visto.

12.3.4 HERRAMIENTA UART (UART TOOL)

Ésta es una aplicación de la herramienta PICkit 2, que permite verificar el funcionamiento de un puerto serial o **UART**, en los pines **Tx** y **Rx**. Se utilizan dos pines del ICSP (Figura 12.10) como pines de Tx (*out*) y Rx (*in*), desde la unidad del PICkit 2, conectados a los pines **Rx_MCU** y **Tx_MCU** respectivamente. Los niveles de alimentación deben estar entre V_{DD} y V_{SS}.

```
NOTA: Tenga en cuenta que está conectando directamente los pines
del MCU, o sea antes del intercambiador de voltajes RS232, los
niveles de voltaje son máx. +5V.
```

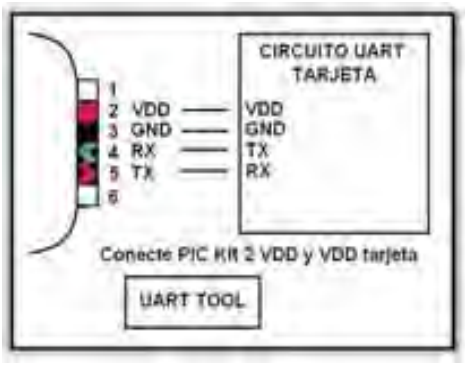


Figura 12.10 ICSP pines en herramienta UART.

En la Figura 12.10 se aprecian los pines del conector **ICSP** como herramienta **UART**, conectado al micro en la tarjeta. Observe que los pines **Tx** y **Rx** están conectados con complemento correspondiente. Es necesario que los pines de V_{DD} estén ambos conectados, pero cada uno con su fuente independiente de alimentación.

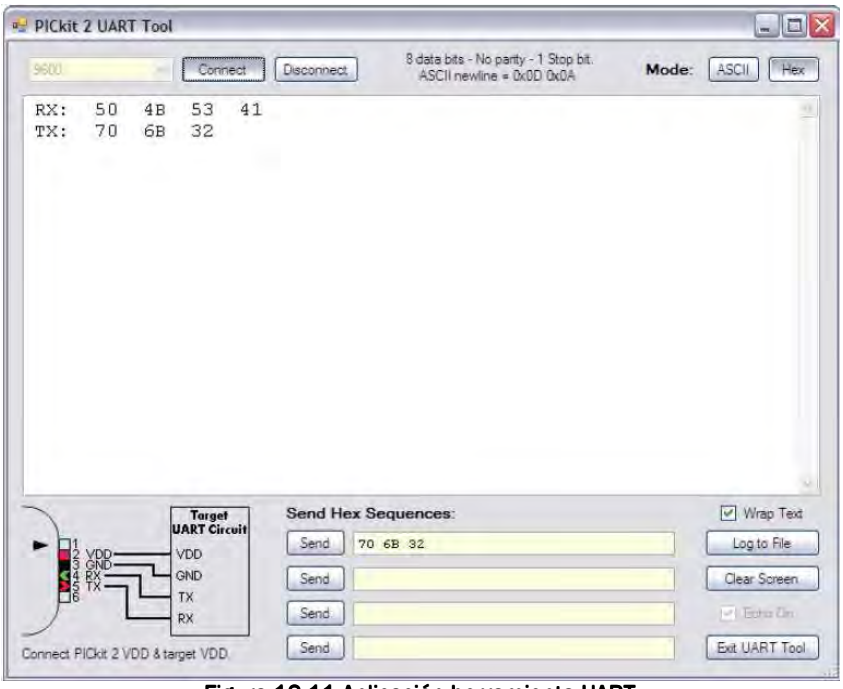


Figura 12.11 Aplicación herramienta UART.

En la aplicación tenemos que configurar la velocidad en **baudios** (o bps: bits por segundo) desde 150 a 38.400, el formato es siempre **8N1**, y el voltaje es entre 2.5 a 5VCD y permite comunicación *full-duplex* hasta 200 líneas. También debemos

escoger si los datos los deseamos ver en **ASCII** o **HEX**. Los datos empiezan a aparecer secuencialmente y se indica el que origina la trama si es **Rx** o **Tx**. En la Figura 12.11 observamos cómo se muestran los datos transmitidos y recibidos, en un ejemplo de 2 tramas. Para mostrar una nueva línea, el **Tx** de la tarjeta debe enviar los caracteres de valores **0x0D** (*carriage return*) y **0x0A** (*line feed*) en esa secuencia. Podemos invocar esta herramienta desde la aplicación del programador PICkit 2, en el menú de *Tool*, en la opción **UART Tool.**

12.3.5 PROGRAMADOR STAND-ALONE (PROGRAMMER-TO-GO)

Ésta es una aplicación de la herramienta PICkit 2, que permite guardar la imagen del programa de un MCU y programar otro sin la intervención de un PC. De esta manera trabaja como un programador de campo *stand-alone*. La forma en que trabajan los pines es la misma que la del programador original (Figura 12.12).



Figura 12.12 ICSP pines en programador *stand-alone*.

En la Figura 12.13 a, b se muestra la aplicación del programador stand-alone.

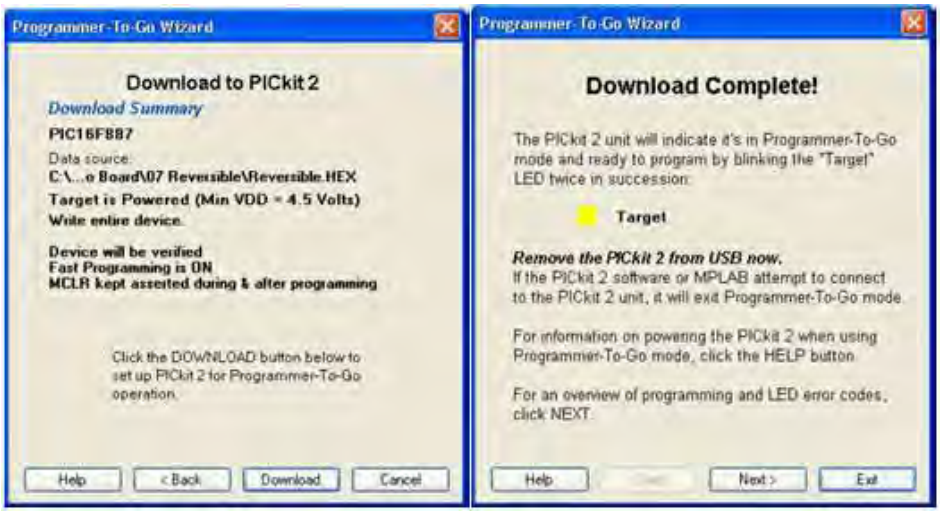


Figura 12.13 Aplicación programador *stand-alone*

Para su funcionamiento se utilizan el **pulsador** y la indicación del estado mediante los leds. Primero se debe descargar el archivo de programa a la unidad PICkit 2, después de esto el programador queda listo para realizar una programación en campo. Cuando se programe el micro en campo (Figura 12.14) debe tenerse la alimentación suficiente en los pines del conector USB.

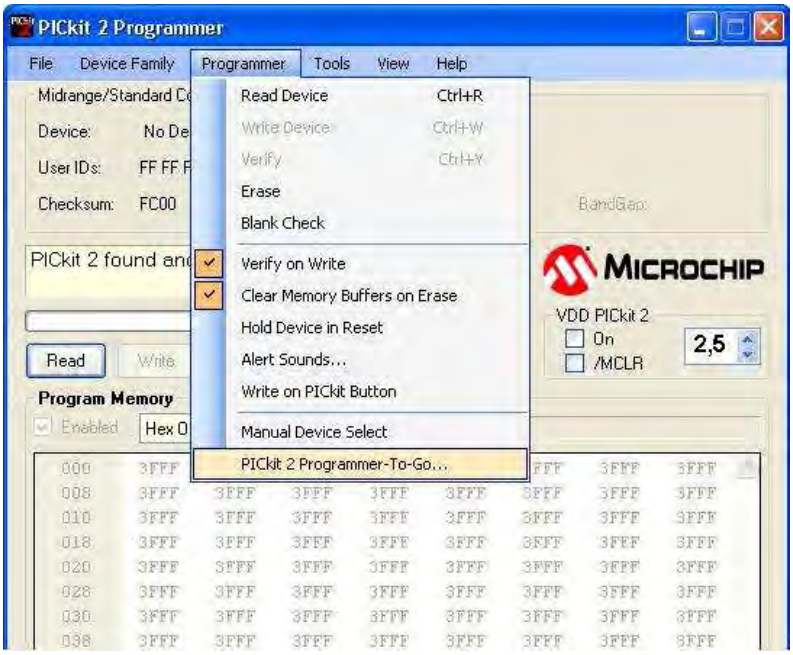


Figura 12.14 Llamado a programador *stand-alone*.

La aplicación del programador *stand-alone* se invoca desde la aplicación PICkit2 programmer, en el menú de *programmer*, tal como indica la Figura 12.14. Para mayor información sobre el funcionamiento de esta aplicación, consulte el manual *PICkit 2 Programmer-To-Go User Guide*.pdf que se encuentra en nuestra página web: http://virtual.alfaomega.com.mx en la carpeta :\Herramientas\PICkit 2\Manuales.

13.0 SISTEMA DE EVALUACIÓN – PIC16F8XX

El Sistema de Evaluación **PIC16F8XX** es la tarjeta de trabajo para todos los ejemplos realizados en este libro; esta tarjeta está diseñada para el micro **PIC16F877A**, pero soporta varios micros de la familia **PIC16F8XX** (abajo MCU soportados). Esta tarjeta

permite el trabajo con todos los módulos del MCU, cada uno de los pines de los módulos están dispuestos en conectores para facilidad de conexión con otras tarjetas, también se agregan otros dispositivos para facilitar las pruebas de los módulos. Toda la información de la tarjeta se encuentra así:

- Especificaciones
- Configuración de pines y conectores
- Planos: Esquemático, Capa Screen, Capa Soldadura, Capa Puentes
- Lista de partes
- Pasos para ensamble
- Pruebas de funcionamiento

Toda esta información (Figura 13.1) la puede encontrar en nuestra página web: http://virtual.alfaomega.com.mx en la carpeta:**\Herramientas\Sistema de Evaluación PIC16F8XX** (2009-04-04).



Figura 13.1 Sistema de evaluación PIC16F8XX.

13.1 ESPECIFICACIONES

Las especificaciones del sistema de evaluación **PIC16F8XX** son:

1. MCU Soportados:
   - PIC16F877A, PIC16F874A
   - PIC16F887, PIC16F884
   - Todos (PDIP 40 PIN)
2. MCU:

- Oscilador Cristal 20MHz
- Pulsador de RESET
- Base 40 pines

3. Programación:
   - Conector ICSP
4. Comunicación:
   - Comunicación RS232 (TX,RX,GND)
   - Leds indicadores de TX y RX
5. Dispositivos de prueba:
   - 2 Leds de prueba OUT-1, OUT-2
   - 2 Pulsadores de prueba INPUT-1, INPUT-2
   - Buzzer de 5V 2mA)
   - Relé de 5VDC 8 pines 1Amp
   - Relé Sensor de temperatura LM335
6. Conectores para :
   - Display LCD (con Potenciómetro de ajuste)
   - Puerto paralelo
   - Relé 6 PIN (2NC, 2P, 2NO)
   - Pin de tierra
7. Dispositivos I2C:
   - Memoria I2C EEPROM tipo 24LCXX
   - Pull-up 2R=10k, base de 8 pines
   - Real Time Clock RTC (DS1307)
   - Xtal 32.768 kHz, conector batería y base de 8 pines
8. Alimentación
   - Voltaje de alimentación 7-14VC

Pines de prueba de fuente (+12V, +5V, GND)
Tamaño: 13.4 $\times$ 6.5 cm
Peso: 150 gr.
Circuito impreso: 1 cara
Material: fenolica

13.2 CONFIGURACIÓN DE PINES Y CONECTORES

Ésta es la tabla resumen de todos los conectores (Tabla 13.1), en ella podrá encontrar:

- Número del conector y nombre del módulo asociado.
- Nombre y número del pin del micro, conectado con el módulo, destacando los diferentes nombres del pin según el puerto y el módulo.
- Diferenciación de los módulos por color o por trama.

CONECTORES Y MODULOS USADO SISTEMA DE EVALUACIÓN PIC16F8XX (2009-04-04)

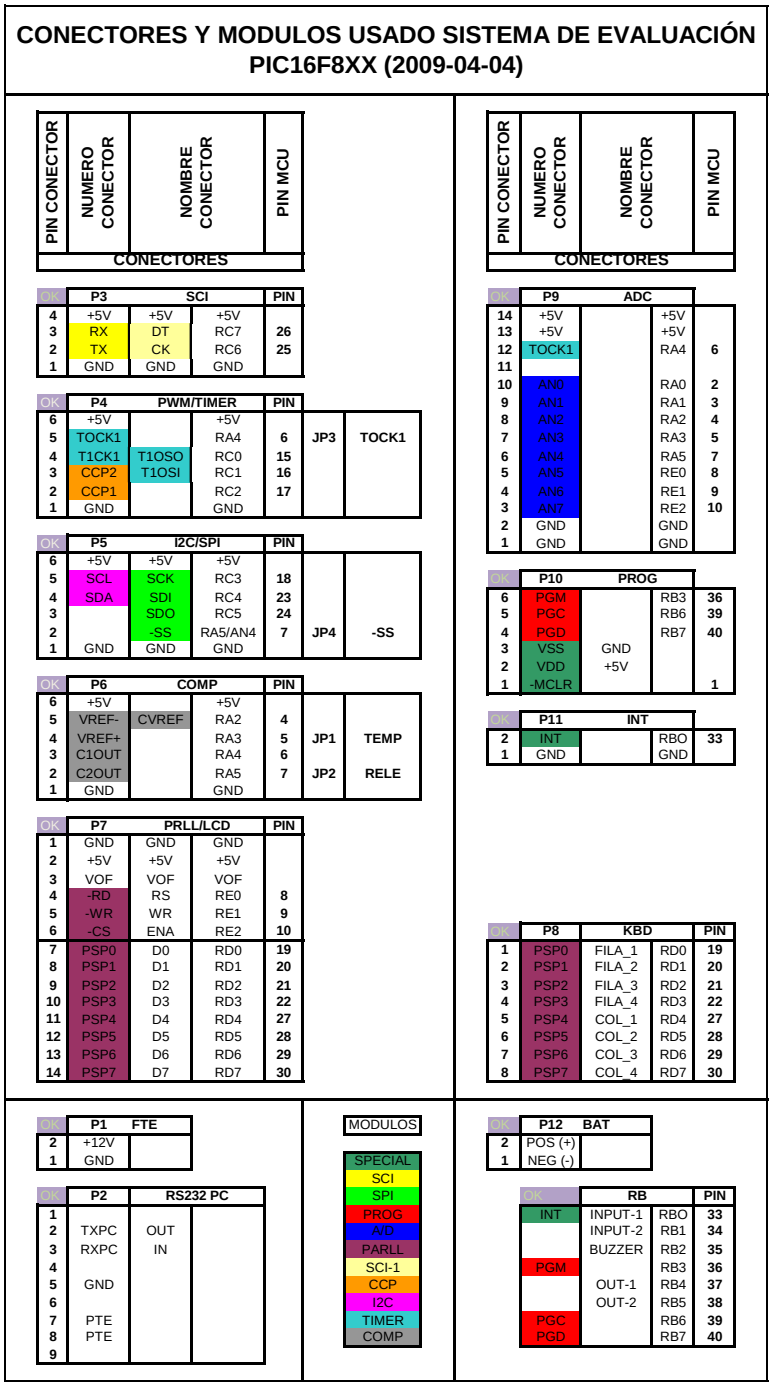


Tabla 13.1 Configuración de módulos y conectores PIC16F8XX.

13.3 PLANOS

A continuación se presentan los planos: esquemático y capas del **circuito impreso** o **board** con presentaciones en arte y real (véanse Figuras 13.2 a 13.9). Toda esta información puede encontrarla en nuestra página web ubicada en: http://virtual.alfaomega.com.mx en la carpeta :\Herramientas\Sistema de Evaluación **PIC16F8XX** (2009-04-04).

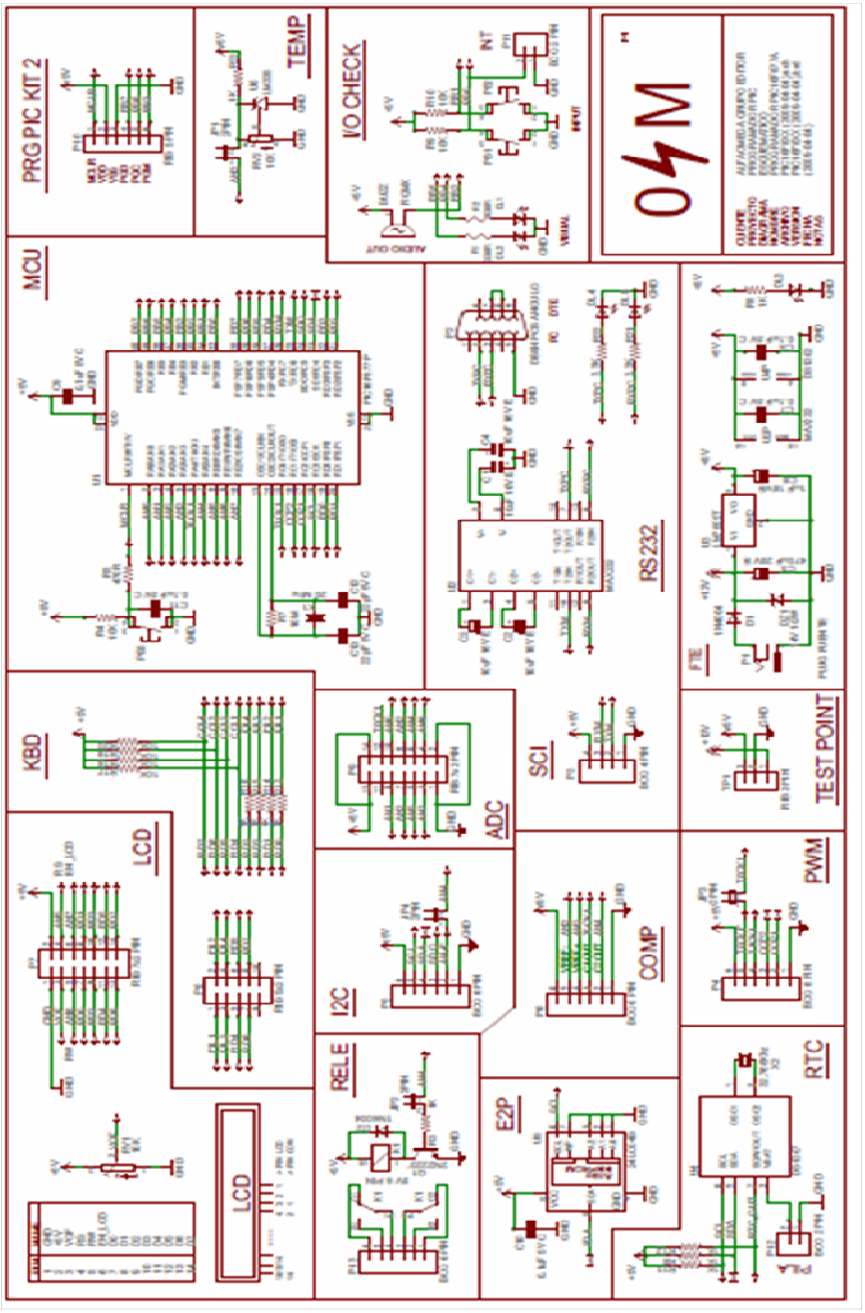


Figura 13.2 Esquemático PIC16F8XX.

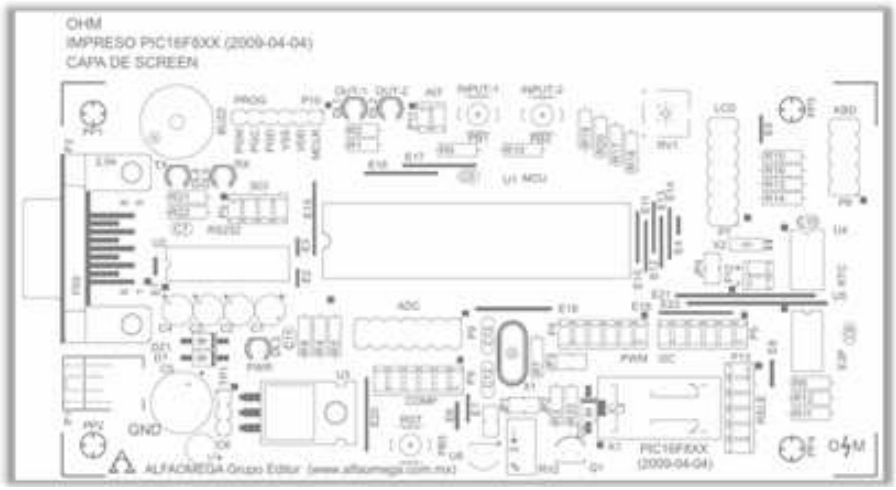


Figura 13.3 Screen PIC16F8XX.



Figura 13.4 Screen real PIC16F8XX.

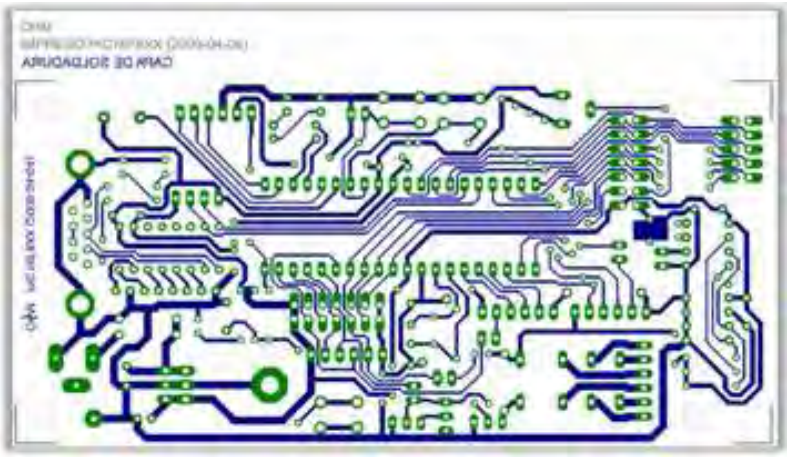


Figura 13.5
Soldadura
PIC16F8XX.

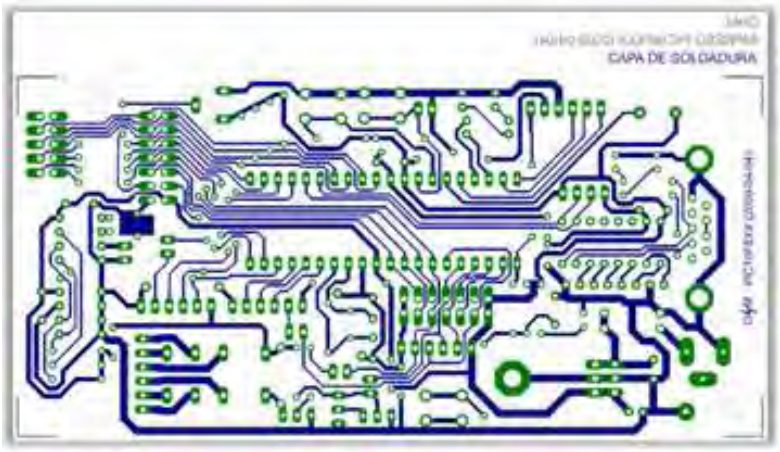


Figura 13.6
Soldadura
rotada
PIC16F8XX.

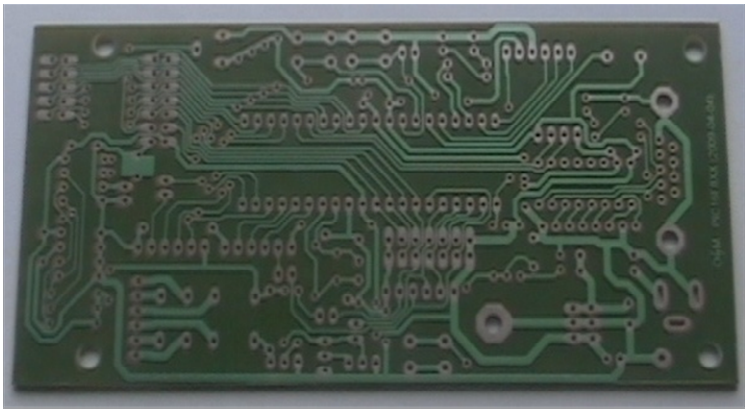


Figura 13.7
Soldadura real
PIC16F8XX.

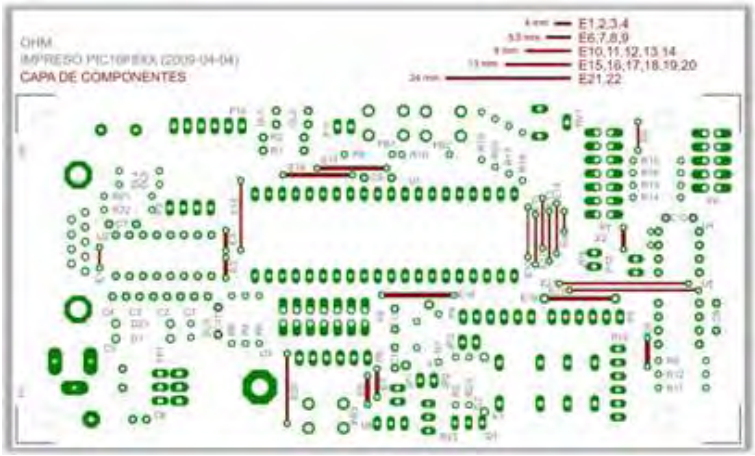


Figura 13.8 Puentes PIC16F8XX.



Figura 13.9 Puentes Real PIC16F8XX.

13.4 LISTA DE COMPONENTES

En la Tabla 13.2 se encuentra la lista de componentes de la tarjeta **PIC16F8XX** (2009-04-04). Cada ítem contiene: la cantidad, la *label* o el nombre en el circuito impreso y el valor o la descripción de componente.

LISTADO COMPONENTES KIT PIC16F8XX (2009-04-04)		
CANT	LABEL	VALOR
1		TARJETA PIC16F8XX (2009-04-04)
1	BUZZ	Buzzer 5 voltios
4	C1-4	10uF 16V E
1	C5	470uF 25V E
1	C6	1uF 16V E
4	C7-11	0.1uF
2	C12-13	22pF 5V C
2	D1-2	1N4004
2	DL1-2	LED3MM VERDE
3	DL3-5	LED3MM ROJO
1	DZ1	14V 1/2W
4	E1,2,3,4	PUENTE 4 MM
4	E6,7,8,9	PUENTE 5.5 MM
5	E10,11,12,13,,14	PUENTE 9 MM
6	E15,16,17,18,19,20	PUENTE 13 MM
2	E21,22	PUENTE 24 MM
0	E5	NO EXISTE
4	JP1-4	2PIN
1	K1	5V 8 PIN
1	P1	PLUG FUENTE
1	P2	DB9H PCB ANGULO
1	P3	BCD 4 PIN
3	P4-6,13	BCD 6 PIN
2	P7-9	RIB 7x2 PIN
1	P8	RIB 5x2 PIN
1	P10	RIB 5 PIN
2	P11-2	BCD 2 PIN
3	PB1-3	PULSADOR MINIATURA 4 PIN
4	PP1-4	SOPORTE PLASTICO
1	Q1	2N2222
2	R1-2	330R
1	R5	470R
10	R3,6,8,11-16,23	1K
7	R4,9,10,17-20	10K
1	R7	10M
2	R21,22	3.3K
1	RV1	10K
1	RV2	10K PRECISION
1	TP1	RIB 3 PIN
1	U1	PIC16F877P
1	U2	MAX232
1	U3	LM7805T
1	U4	DS1307
1	U5	24LC04B
1	U7	LM335
1	X1	20 Mhz
1	X2	32.768Khz
1	U1	BASE 40 PIN
1	U2	BASE 16 PIN
2	U4-5	BASE 8 PIN

Tabla 13.2
Lista de componentes tarjeta PIC16F8XX .

13.5 PROCEDIMIENTO DE ENSAMBLE

En el momento de montar los componentes en una tarjeta, es necesario seguir el orden de inserción de componentes, de bajos a altos, ésta es una ayuda para el ensamble de la tarjeta **PIC16F8XX** (2009-04-04) (Tabla 13.3).

Tabla 13.3 Pasos de ensamble de la tarjeta PIC16F8XX (2009-04-04).

PASOS PARA ENSAMBLE TARJETA PIC16F8XX (2009-04-04)		
1-Limpie tarjeta con Alcohol Isopropílico		
2-Haga y ponga puentes		
4	4mm	E1,2,3,4
4	5.5mm	E6,7,8,9
5	9mm	E10,11,12,13,,14
6	13mm	E15,16,17,18,19,20
2	24mm	E21,22
3-RESISTENCIAS		
1	R7=10M	
1	R5=470R	
2	R1=R2=330R	
2	R21=R22=3.3K	
10	R3,6,8,11,12,13,14,15,16,23=1K	
7	R4,9,10,17,,18,,19,,20=10k	
4-DIODOS		
2	D1,2=1N4004	
1	DZ1=14V 1/2W	
5-REGULADOR		
1	U3=LM7805	
6-BASES INTEGRADOS		
1	40 PIN	
1	16 PIN	
2	8 PIN	
7-PULSADORES		
3	PB1,2,3,4-XTAL	
1	X1=20MHZ	
1	X2=32.987KHz	
8-LEDS		
2	DL1,2	VERDE
3	DL3,4,5	ROJO
9-CONDENSADORE CERAMICOS		
2	C12,13=22pF 50V	
4	C7,8,9,,10=0.1uF 50V	

(continúa)

Tabla 13.3 Pasos de ensamble de la tarjeta PIC16F8XX (2009-04-04).

10-POTENCIOMETRO, TRANSITOR, LM35
1 RV1=10K
1 Q1=2N2222
1 U6=LM335
11-CONECTORES RIBON
2 P7,9=RIBBON 2x7 PIN
1 P8=RIBBON 2x5 PIN
1 P10=RIBBON 5 PIN
1 TP1=RIBBON 3 PIN
4 JP1,2,3,4=2 PIN
12-CONECTORES BLANCOS
4 P4,5,6,13=6 PIN
1 P3=4 PIN
2 P12,12=2 PIN
13-POTENCIOMETRO PRECISION
1 RV2=10K
14-CONDENSADORES ELECTROLITICOS
4 C1,2,3,4=10uF 16V
1 C6=1uF 16V
1 C5=470uF 16V
15-CONECTORES
1 P1=PULG FUENTE
2 P2=DB9 ANGULO
16-BUZZER
1 BUZZ
17-RELE NOTA: PONER UNA BASE DE 16 PINES PARA EL RELE
1 K1=RLE 5V 8 PIN
18-POSTES PLASTICOS
4 PP1,2,3,4
NO MONTE INTEGRADOS ANTES DE HACER CHEQUEO DE FUENTE

13.6 PRUEBAS DE FUNCIONAMIENTO

Estas pruebas de funcionamiento son las pruebas básicas para verificar el ensamble correcto de la tarjeta **PIC16F8XX**.

```
NOTA: La tarjeta debe tener todas las bases vacías, es decir, sin
integrados montados, para realizar estas pruebas.
```

SIN CONECTAR FUENTE

1. Verifique continuidad en las terminales de 5V.
   - TP1 pin 3 (tómelo como referencia)
   - U1 pin 32
   - U2 pin 16
   - U4 pin 8
   - U5 pin 8
   - E7, E8, E16, E18

2. Verifique continuidad en las terminales de 0 V=GND o tierra:
   - TP1 pin 2 (tome éste como referencia o el pin GND)
   - U1 pin 31
   - U2 pin 15
   - U4 pin 4
   - U5 pin 4
   - E2, E3, E6, E17, E22

CONECTAR LA FUENTE

Antes de conectar Fuente (Adaptador):

- Verifique que la terminal (+) esté conectada al centro y el (-) en el exterior del plug.
- El voltaje debe estar en el rango (7-14 VCD). Se supone de 12 VCD.

3. Conecte la fuente:
   - El Led rojo PWR (Power, DL3) debe encender
   - En TP1 el pin 1 debe marcar +12 VCD
   - En TP1 el pin 3 debe marcar +5 VCD

Verifique que los +5 VCD estén en las terminales señaladas en el numeral 1.

DESCONECTAR FUENTE

Desconecte Fuente (Adaptador).

4. Proceda a insertar los integrados en las bases, verifique que la polaridad o punto 1 de cada base e integrado coincidan.

CONECTAR FUENTE

5. Conecte fuente (Adaptador):

   - Los pasos del numeral 3 deben cumplirse
   - Los integrados no deben sobrecalentarse

6. Conecte el PICkit 2 al puerto USB del PC y al conector ICSP en la tarjeta.
7. Abra el programa MPLAB.
8. Abra el archivo de "Test-PIC16F877A".
9. Programe el micro.
10. Verifique funcionamiento.

Este procedimiento se encuentra en el video "Probando Sistema **PIC16F8XX**", que se encuentra en nuestra página web ubicada en: http://virtual.alfaomega.com.mx. En caso de algún problema, verifique las conexiones de la board ensamblada contra el esquemático, verifique las continuidades respectivas para establecer posibles cortos. Por último, verifique su fuente de alimentación, en los rangos y la polaridad adecuada.

13.7 MÓDULOS

Los módulos adicionales para trabajo, junto con la tarjeta de evaluación PIC16F8XX, son:

REFERENCIA	DESCRIPCIÓN
MOD_7S_4D_L (2009-05-06)	Módulo display 7 segmentos 4 dígitos *low*
MOD_8LEDS (2009-05-06)	Módulo 8 leds
MOD_COMP (2009-05-07)	Módulo comparadores 4 potenciómetros o 4 entradas análogas

Esta información se encuentra en la Web en la carpeta **X:\Herramientas\Módulos para Sistema de Evaluación PIC16F8XX** (2009-04-04).

PARTE 3

PROGRAMACIÓN (II)

14.0 AMBIENTE DE DESARROLLO MPLAB

Es un programa de software que corre sobre una PC para desarrollar aplicaciones para microcontroladores Microchip; es llamado IDE (*Integrated Development Environment*) o ambiente de desarrollo integrado (Figura 14.1); este software se llama así porque contiene todas las aplicaciones en un ambiente para desarrollar código para microcontroladores en sistemas embebidos.



Figura 14.1 MPLAB IDE.

Un sistema embebido es un diseño que involucra un microcontrolador como PIC micro MCU o dsPIC o DSCs, con algunos periféricos y circuitos a los que controla, y se agrega a otras partes mecánicas para hacer un producto electrónico. Ejemplos de esto pueden ser un reloj digital o un detector de humo. Un sistema de desarrollo para microcontroladores embebidos es un sistema de programas que corre sobre una PC, para ayudar a escribir, editar, depurar y programar código (la inteligencia de la aplicación) en un microcontrolador. MPLAB IDE corre sobre una PC y contiene todos los componentes necesarios para diseño y desarrollo de aplicaciones de sistemas embebidos, es gratis y se puede descargar de la página de Microchip.

14.1 ENTORNO

14.1.1 EL PROCESO DE DISEÑO

El proceso de diseño, más que un principio y un fin de tareas para realizar, constituye un ciclo (Figura 14.2) en el que el programa se va depurando para que quede como lo requiere el funcionamiento de la aplicación. Todos estos pasos requieren una aplicación para realizarla y **MPLAB** las contiene todas en un solo ambiente de trabajo.

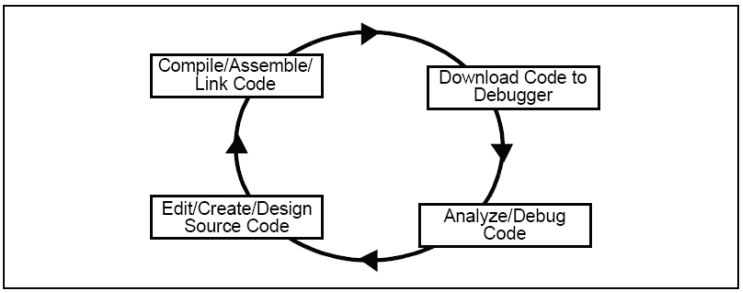


Figura 14.2 Ciclo de diseño.

Las tareas típicas para desarrollar una aplicación con controlador embebido son:

1. Crear un alto nivel de diseño. Escoger el tipo de MCU apropiado a las características deseadas, con los pines y los módulos necesarios, y con el lenguaje apropiado para la construcción del software que controle el hardware apropiadamente.

2. Escribir el código o el programa. En un editor de texto se escribe el programa escogiendo un lenguaje apropiado.

3. Compilar, ensamblar y enlazar el programa para convertirlo en 1 y 0. Convertir el programa en un código ejecutable por el MCU.

4. Probar el código. El programa, una vez escrito, necesita ser probado con el hardware y verificar si el funcionamiento concuerda con el esperado en el mundo real; para esto se hace la depuración que corrige todos los *bug*s, verificando si las variables toman los valores adecuados ajustándolos y ejecutando paso a paso las rutinas para comprobar su funcionamiento.

5. Quemar el micro con el programa realizado.

Cada una de estas tareas puede ser bastante compleja e involucra diferentes programas o aplicaciones, lo cual es transparente para el usuario que se debe concentrar en solucionar su aplicación de la manera más eficiente, aprendiendo a utilizar todas las herramientas que le proporciona el ambiente de desarrollo.

```
NOTA: MPLAB IDE es una envoltura que coordina todas las
herramientas desde una sola interfaz gráfica de usuario.
```

14.1.2 COMPONENTES DEL MPLAB IDE

El **MPLAB IDE** tiene componentes incorporados y módulos *plug-in* para configurar el sistema para una variedad de herramientas de software y hardware para todos los dispositivos PIC micro MCU, dsPIC y DSC.

MPLAB IDE componentes incorporados

Estos componentes se instalan automáticamente con el MPLAB.

- **Manejador de proyectos o** *Project Manager*
  Provee la integración y comunicación entre el IDE y las herramientas de lenguaje (*Language Tools).*

- **Editor**
  Es concebido como un editor de texto para desarrollo de microcontroladores; sirve también como ventana en depuración.

- **Ensamblador/Enlazador** y **Herramientas de lenguaje**
  El **ensamblador** (*Assembler*) puede ser usado para ensamblar un solo archivo, o puede usarse con el **enlazador** (*Linker*) para construir (*build*) un proyecto con archivos de fuentes separados, librerías y objetos recompilados. El **enlazador** (*Linker*) es el responsable de poner el código compilado en las áreas de memoria del micro de destino.

- **Depurador** o *debugger*
  El depurador permite seguir paso a paso el programa, poner *breakpoints*, tener ventanas de visualización para registros y todo lo necesario de un depurador moderno. Trabaja en conjunto con el editor para referenciar la información del código de máquina con el código fuente. Puede ser simulado o conectado a hardware.

- **Motor de ejecución** o *Execution Engine*
  Éste es el software de simulación para todos los PIC micro MCU y dsPIC DSC, los cuales simulan las instrucciones y algunos módulos periféricos. Son opcionales los emuladores y depuradores *in-circuit*, están disponibles para probar código corriendo el programa en hardware.

MPLAB IDE Componentes adicionales opcionales

Los componentes adicionales pueden ser comprados y adicionados al MPALB IDE.

- **Herramientas de Compilación de Lenguaje** o *Compiler Language Tools*
  Compiladores MPLAB C18 y MPLAB C30, C para Microchip totalmente integrados y con optimización de código. Junto con compiladores de fabricantes como: HI-TECH, IAR, microEngineering Labs, CCS y Byte Craft, son invocados por el manejador de proyectos del MPLAB IDE para compilar código, que es cargado de manera automática en la tarjeta de depuración para probar y verificar instantáneamente.

- **Programadores** o *Programmers*
  PICSTART Plus, PICkit 1, PICkit 2, PICkit 3, PRO MATE II, MPLAB PM3 como MPLAB ICD 2 pueden programar código en dispositivos destino. MPLAB IDE ofrece el control total sobre la programación de ambos código y datos, así como los bits de configuración para poner varios modos de operación de los PIC micro MCU, dsPIC y DSC.

- **Depuradores** *In-Circuit Debugger*
  MPLAB ICD 2 y otros depuradores son una alternativa, los cuales pueden bajar el código al microcontrolador insertado en la tarjeta de aplicación, correr paso a paso, insertar *breakpoints* y monitorear registros y variables.

- **Emuladores** *In-Circuit Emulators*
  MPLAB ICE 2000 y MPLAB ICE 4000 son totalmente caracterizados para emular los dispositivos PIC micro, MCU y dsPIC DSC. Se conectan a la PC vía puertos I/O, y permiten un control total del micro en la tarjeta de aplicación.

14.1.3 PARTES AMBIENTE MPLAB

Al iniciar el programa MPLAB aparece una ventana de trabajo (Figura 14.3).

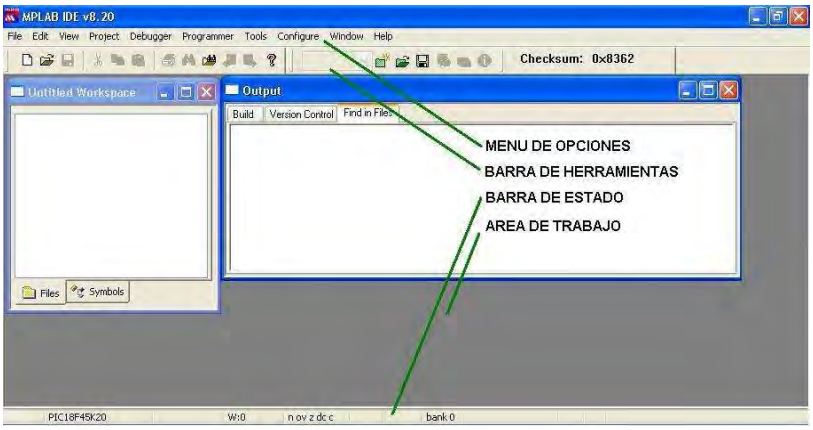


Figura 14.3 Partes ambiente MPLAB.

En esta ventana observamos cada una de las partes:

- Menú de opciones
- Barra de herramientas
- Barra de estado
- Área de trabajo

En el transcurso del aprendizaje del MPAB IDE iremos viendo estas partes.

14.2 PASOS PARA CREAR UN PROYECTO EN MPLAB

Para crear un proyecto supondremos que está instalado el software MPLAB en el subdirectorio **C:\Archivos de Programa**.

```
NOTA: Se recomienda hacer la instalación del MPLAB en el
subdirectorio por defecto C:\Archivos de Programa\; en caso de
haber escogido otra ruta o path diferente, es posible que la
compilación no funcione, si esto sucede deberá buscar los archivos
e insertar los path adecuados.
```

Para crear un proyecto se seguirán los pasos siguientes:

- Crear proyecto
- Selección de dispositivo

- Selección de lenguaje
- Poner el nombre y *path* del proyecto
- Agregar archivos al proyecto
- Resumen del proyecto

Una vez creado el proyecto, los pasos que se siguen son:

- Crear el código
- Construir el proyecto
- Probar el código con el simulador
- Programar el microcontrolador
- Depurar el programa

El proceso de los diferentes programas y archivos que toman parte en un proyecto, hasta llegar al archivo de salida, se puede apreciar en la Figura 14.4.

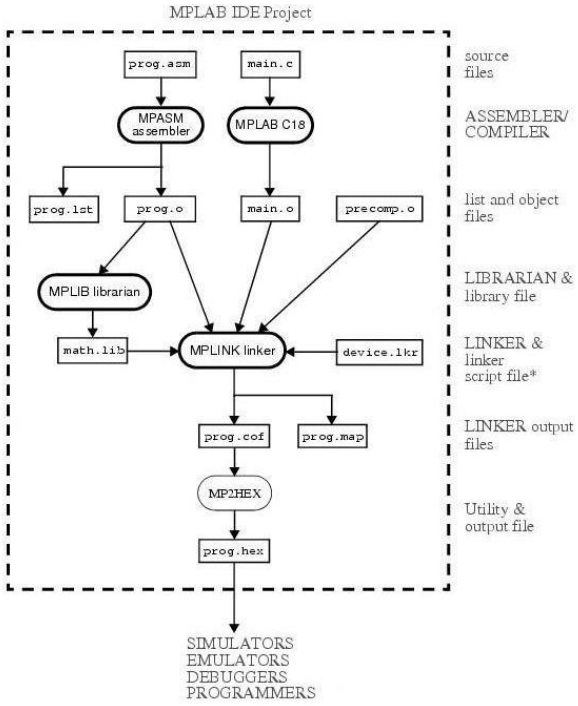


Figura 14.4 Partes MPLAB PROJECT.

14.2.1 CREAR PROYECTO

La creación del proyecto se hace con el **ayudante de proyectos**. Al hacer clic en el menú de opciones *Project/Project Wizard* aparece la ventana de *Project Wizard*, en la cual seleccionamos la casilla **Siguiente>**, como se observa en la Figura 14.5.

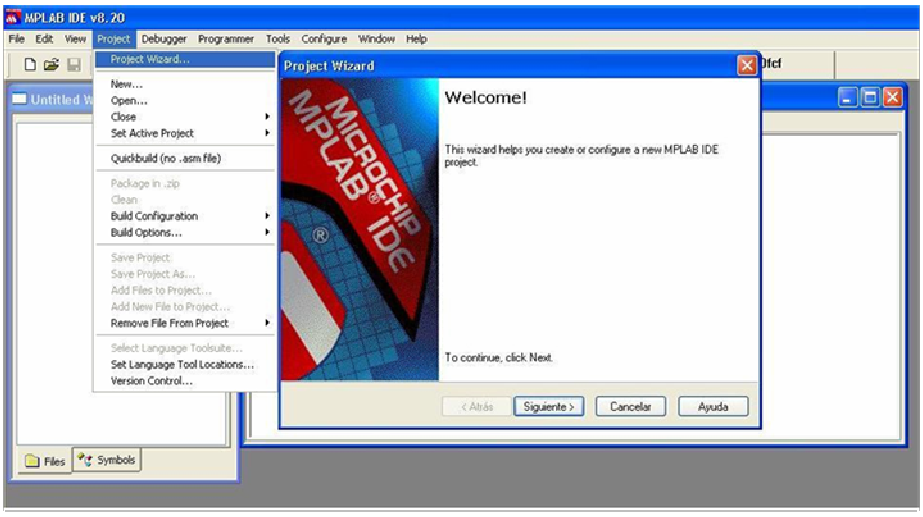


Figura 14.5 Ayudante de proyectos.

14.2.2 SELECCIÓN DE DISPOSITIVO

Luego se selecciona el **dispositivo** como primer paso. En la casilla **Device**: seleccionamos el dispositivo **PIC16F877A** en el menú desplegable que nos aparece, hacemos **clic** en la casilla **Siguiente>**, como se aprecia en la Figura 14.6.

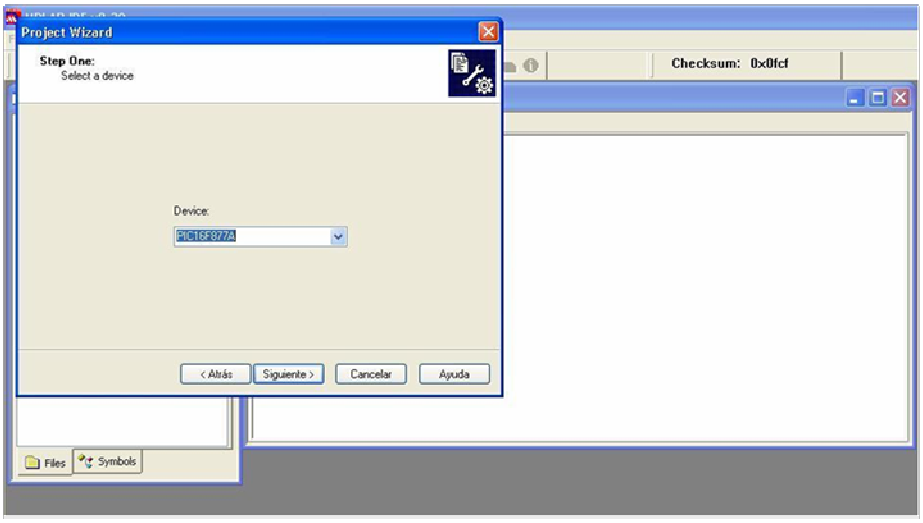


Figura 14.6 Selección de dispositivo.

14.2.3 SELECCIÓN DE LENGUAJE

Luego se selecciona la Herramienta de Lenguaje como segundo paso. En la casilla **Active Toolsuite**: seleccionamos el lenguaje **Microchip MPASM Toolsuite** en el menú desplegable que nos aparece, y hacemos clic en la casilla **Siguiente>**, como muestra la Figura 14.7.

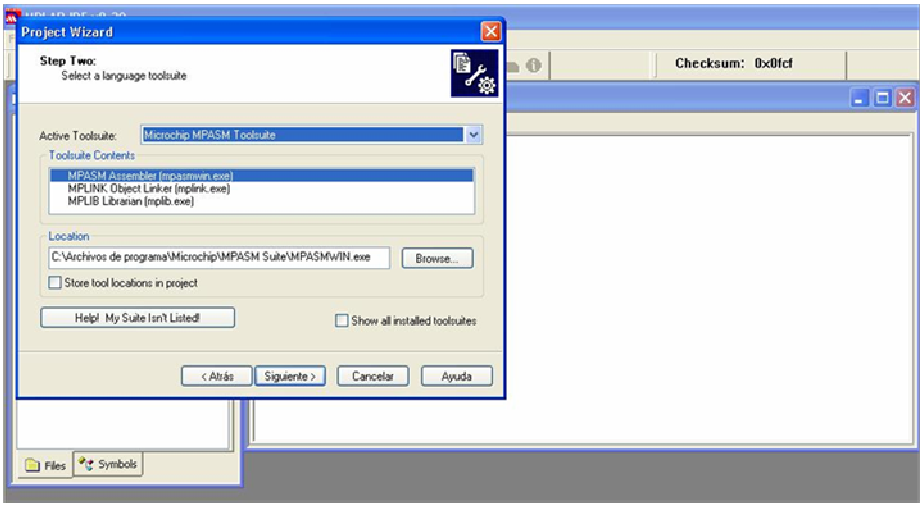


Figura 14.7 Selección de lenguaje.

El **MPASM Toolsuite** usa tres programas con sus respectivos archivos ejecutables que aparecen en el cuadro **Toolsuite Contents** y sus rutas de acceso, que aparecen en el cuadro **Location**; estos archivos ejecutables son:

- MPASM Assembler
  C:\Archivos de Programa\Microchip\MPASM Suite\mpasmwin.exe

- MPLINK linker
  C:\ Archivos de Programa \Microchip\MPASM Suite\mplink.exe

- MPLIB Librarian
  C:\ Archivos de Programa \Microchip\MPASM Suite\mplib.exe

Cuando se instaló el MPLAB se debió hacer en la ruta **C:\Archivos de programa**; en caso de que haya escogido otra ruta o *path*, es posible que la compilación no funcione, y en este caso le tocará buscar los archivos e insertar los *path* adecuados; continuamos haciendo **clic** en la casilla **Siguiente>**.

14.2.4 PONER NOMBRE Y RUTA DEL PROYECTO

Aquí se ponen el nombre del proyecto y el *path* de la carpeta que contendrá los archivos. Debajo de *Create New Project File* escribimos el nombre del proyecto; podemos escoger el *path* asociado en la casilla *Browse* con la ventana emergente que nos da la dirección donde se ubica el proyecto, y hacemos clic en la casilla **Siguiente>**, como muestra la Figura 14.8.

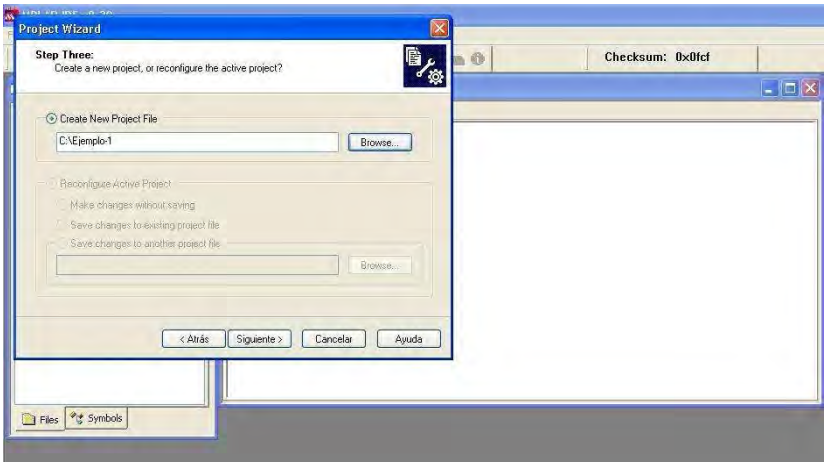


Figura 14.8 Nombre y *path* del proyecto.

14.2.5 AGREGAR ARCHIVOS AL PROYECTO

Se pueden agregar al proyecto archivos ya creados. En la ventana del lado izquierdo se seleccionan los archivos; con la casilla **Add** se agregan a la ventana derecha, y con la casilla **Remove** los quitamos (también se pueden agregar luego de creado el proyecto), como muestra la Figura 14.9.

Se deben agregar los archivos:

- **C:\Archivos de programa\Microchip\MPASM Suite\P16F877A.inc**
  Contiene las definiciones de todos los registros y bit de trabajo del micro.

- **C:\Archivos de programa\Microchip\MPASM Suite\LKR\ 16f877ai.lkr**
  Enlaza todas las áreas de memoria con el compilador y depurador.

Estos dos archivos son de importancia relevante y se tratan en la Sección 14.6.1, "Archivos de enlace Linker (*.LKR)". Una vez agregados los archivos o no (Figura 14.9), continuamos haciendo clic en la casilla **Siguiente>**.

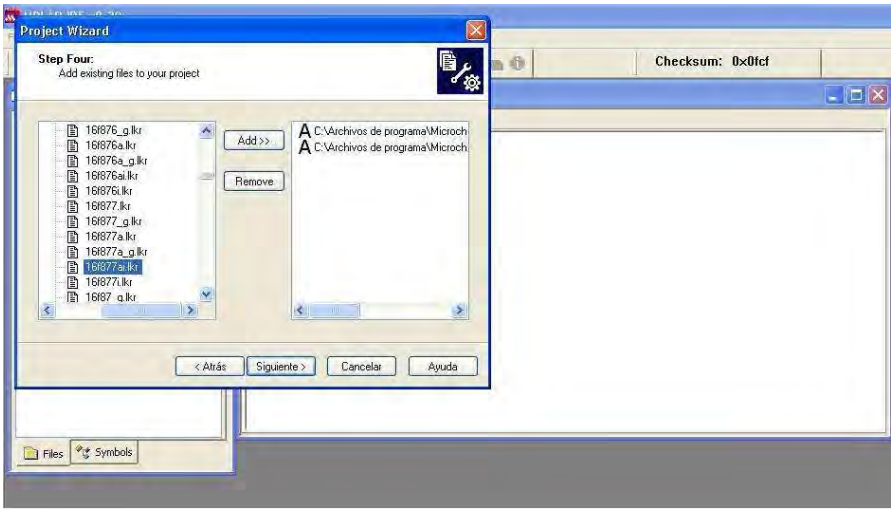


Figura 14.9 Agregar archivos al proyecto.

```
NOTA: También se pueden agregar archivos al proyecto, luego de
haberlo creado.
```

14.2.6 RESUMEN DEL PROYECTO

Para terminar aparece una ventana resumen del proyecto creado. En el cuadro **Project Parameters** se encuentra la información del proyecto, así:

Device: PIC16F877A
Toolsuite: Microchip MPASM Toolsuite
File: C:\Ejemplo-1.mcp
Que fueron los parámetros escogidos.

Como muestra la Figura 14.10, para finalizar hay que hacer **clic** en la casilla **Finalizar.**

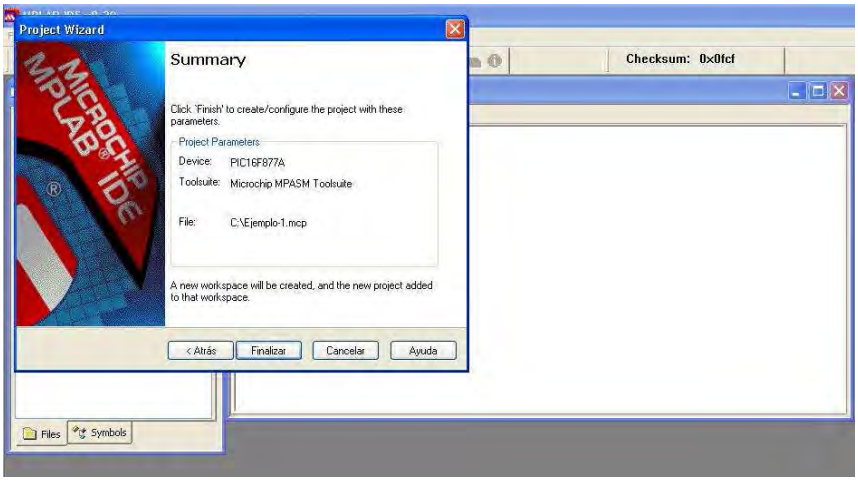


Figura 14.10 Resumen del proyecto.

Hasta aquí ya hemos creado el proyecto, que involucra:

- Escoger el dispositivo o microcontrolador
- Escoger la herramienta de lenguaje
- Escoger el nombre del proyecto y el *path* donde quedarán los archivos
- Agregar los archivos ***.inc** (definiciones) y ***.lkr** (*linker*)

En la Figura 14.11 se muestra la ventana que aparece una vez que se ha creado el proyecto.

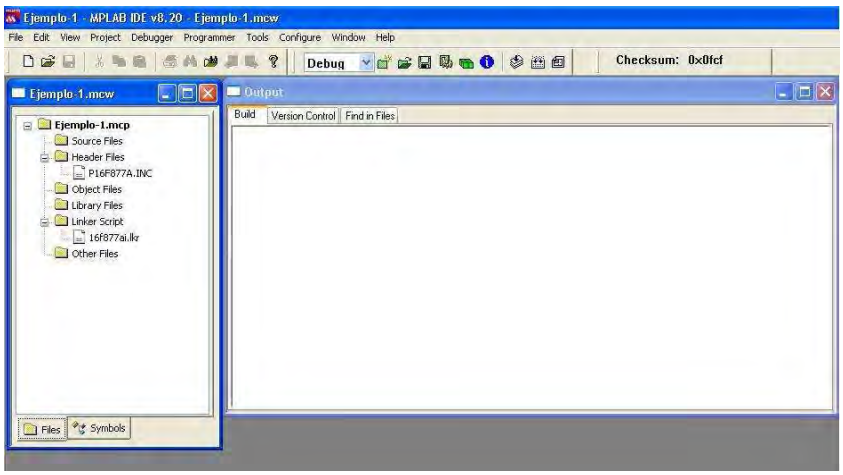


Figura 14.11 Ventana del proyecto.

Observe cómo los archivos que agregamos:

- **`P16F877A.inc`** , está en la carpeta **Header Files**
- **`16f877ai.lkr`**, está en la carpeta **Linker Script**

Una vez creado el proyecto, se siguen los siguientes pasos:

- Crear el código
- Construir el proyecto
- Probar el código con el simulador
- Programar el Microcontrolador
- Depurar el programa

14.2.7 CREAR CÓDIGO

Ahora crearemos el código del archivo: **Blink_1.asm** (dentro del proyecto **C:\Ejemplo-1.mcp** ya creado). En el menú de opciones *File/New* cree un archivo nuevo llamado *Untitled*, y luego guárdelo, con el menú de opciones *File/Save as* con el nombre **C:\Blink_1.asm**. En la ventana ya creada del archivo **C:\Blink_1.asm** copie el código siguiente:

```
;************ ************ Blink_1 **********************************

list p = 16f877a ;  directiva para definir procesador
 #include <p16f877a.inc> ;  definiciones de variables específicas del procesador
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
 _WRT_OFF & _LVP_OFF & _CPD_OFF
                org 0
Start:
        bsf     STATUS,RP0      ; seleccione Banco 1
        bcf     TRISB,4         ; configure pin RB4, salida, led OUT-1
        bcf     STATUS,RP0      ; seleccione Banco 0
MainLoop:
        bsf      PORTB,4        ; prenda led OUT-1
        bcf      PORTB,4        ; apague led OUT-1
        goto     MainLoop       ; haga ciclo
        end
```

Este código consiste en realizar un ciclo infinito **MainLoop** de encendido y apagado del led **OUT-1**, definido en el puerto **PORTB,4**. Observe cómo el archivo cambia su nombre a **C:\Blink_1.asm***, el símbolo asterisco (*****) indica que no se ha guardado el último cambio; se puede salvar haciendo **clic** sobre el ícono  o desde el menú de opciones con **File/Save** (Figura 14.12).

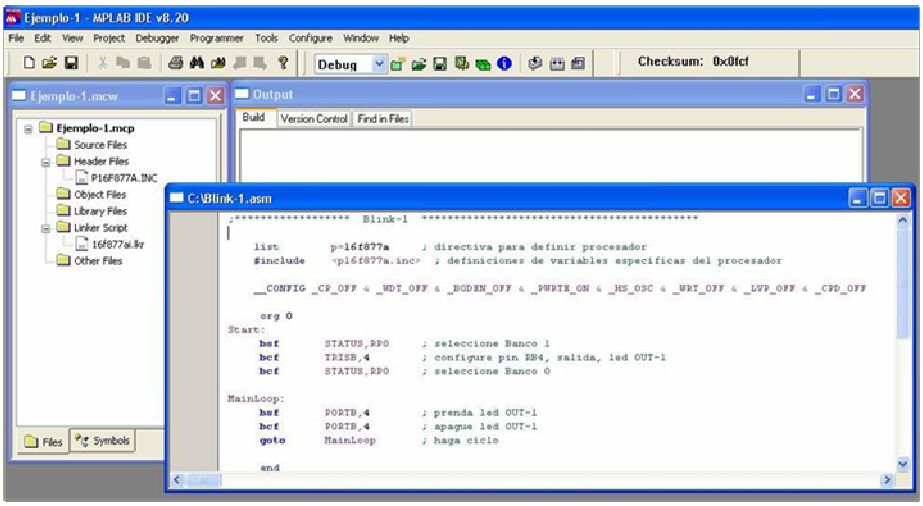


Figura 14.12 Ventana *BLINK_1.ASM*.

Luego de salvar el archivo **C:\Blink_1.asm** podremos tener una vista como la que aparece en la Figura 14.13.

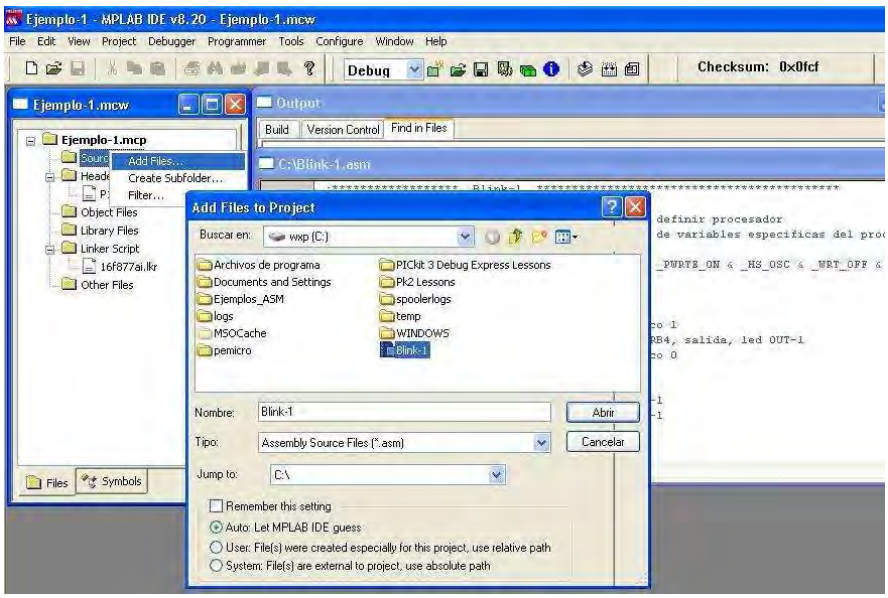


Figura 14.13 Ventana *Adicione Archivos.*

Ahora agregaremos el archivo **C:\Blink_1.asm** a nuestro proyecto, de la manera siguiente, tal como ilustra la Figura 14.13; en la ventana de proyectos **Ejemplo_1.mcw** escogemos la carpeta **Source Files** y detenidos allí con el botón derecho hacemos **clic** en **Add files...**; se despliega la ventana de **Add Files to Project** y buscamos la ruta de nuestro archivo **C:\Blink_1.asm**, y terminamos haciendo **clic** en la casilla **Abrir**. Hasta este punto hemos creado el código y lo hemos agregado al proyecto; a continuación construimos o compilamos el proyecto y realizamos la simulación.

14.2.8 COMPILAR EL PROYECTO O *BUILD*

Construir el proyecto (*build*) o compilarlo consiste en generar los archivos ***.hex** y otros (como se aprecia más adelante); esto se puede hacer de tres maneras. La primera por el menú de opciones en *Project/Build All*, la segunda con el comando abreviado **Ctrl+F10** y la tercera haciendo **clic** en el ícono  . Con cualquiera de las tres opciones anteriores hacemos el procedimiento de construcción o compilación (*build*) del proyecto.

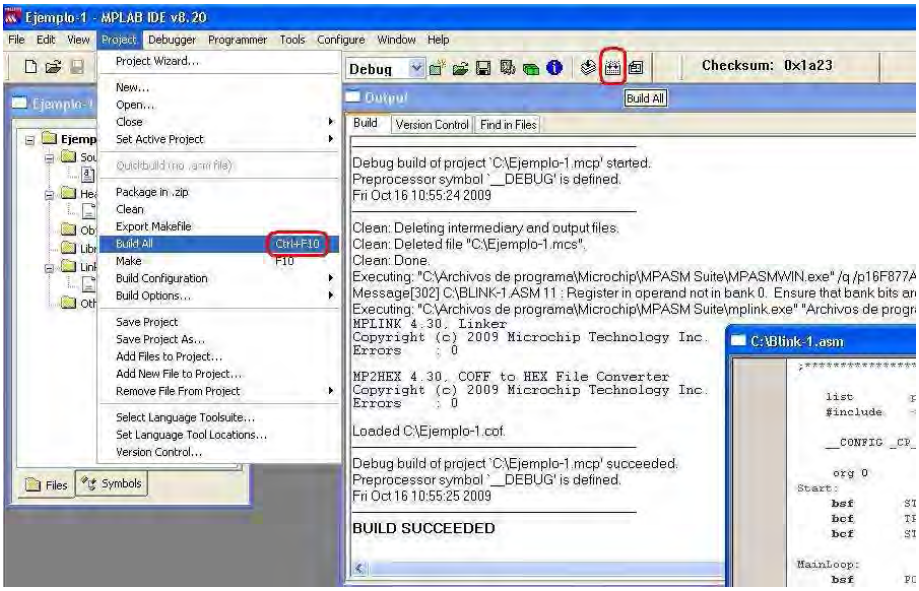


Figura 14.14 Ventana *Construcción del proyecto*.

En la ventana **Output,** en la pestaña **Build**, podemos apreciar, como muestra la Figura 14.14, los pasos realizados para generar la construcción del proyecto **C:\Ejemplo-1.mcp** y nos indica si hay errores en la compilación (a los que podemos acceder haciendo clic sobre ellos); en el caso de compilación exitosa, al final el programa nos entrega el mensaje "Build Succeeded". Una vez compilado con éxito se crean los archivos con extensión: *.MCW, *.MCS, *.MCP, *.MAP, *.HEX, *.COF, correspondientes al proyecto Ejemplo-1 y *.ASM, *.LST, *.ERR, *.O, correspondientes al código Blink_1, como se aprecia en la Tabla 14.1, en la carpeta del proyecto, y en la Tabla 14.2, la correspondiente al proyecto o al código.

Tabla 14.1 Archivos generados.

Nombre	Tamaño	Fecha de modificación
Blink-1.asm	1 KB	16/10/2009 10:55 a.m.
Blink-1.err	1 KB	16/10/2009 10:55 a.m.
Blink-1.lst	14 KB	16/10/2009 10:55 a.m.
Blink-1.o	1 KB	16/10/2009 10:55 a.m.
Ejemplo-1.cof	1 KB	16/10/2009 10:55 a.m.
Ejemplo-1.hex	1 KB	16/10/2009 10:55 a.m.
Ejemplo-1.map	2 KB	16/10/2009 10:55 a.m.
Ejemplo-1.mcp	2 KB	16/10/2009 10:55 a.m.
Ejemplo-1.mcs	1 KB	16/10/2009 01:18 p.m.
Ejemplo-1.mcw	24 KB	15/10/2009 10:38 p.m.

Tabla 14.2 Tipos de archivos generados.

Archivos del Proyecto Ejemplo-1		Archivos del Asembler Blink-1	
*.MCW	Informacion de Workspace	*.ASM	Assembler (MPASM Assembler)
*.MCS	Estado de Build para Make	*.LST	Listado (Ensamblador/Compilador)
*.MCP	Informcaiòn del Poryecto	*.ERR	Errores (Ensamblador/Compilador)
*.MAP	Mapa de registros	*.O	Objeto (Ensamblador/Compilador)
*.HEX	Codigo de maquina formato Hexadecimal		
*.COF	Simbolos y Codigo objeto (MPLINK linker)		

Los pasos siguientes pasan a formar parte de la simulación, la programación y la depuración, los cuales son las formas de operación del MPLAB, cada uno de ellos se toma como sección aparte, por su importancia en el contenido de este libro.

14.3 SIMULACIÓN

La simulación consiste en hacer un seguimiento al código o al programa, paso a paso, en forma *virtual* (solamente en PC) para verificar su funcionamiento adecuado (Figura 14.15).

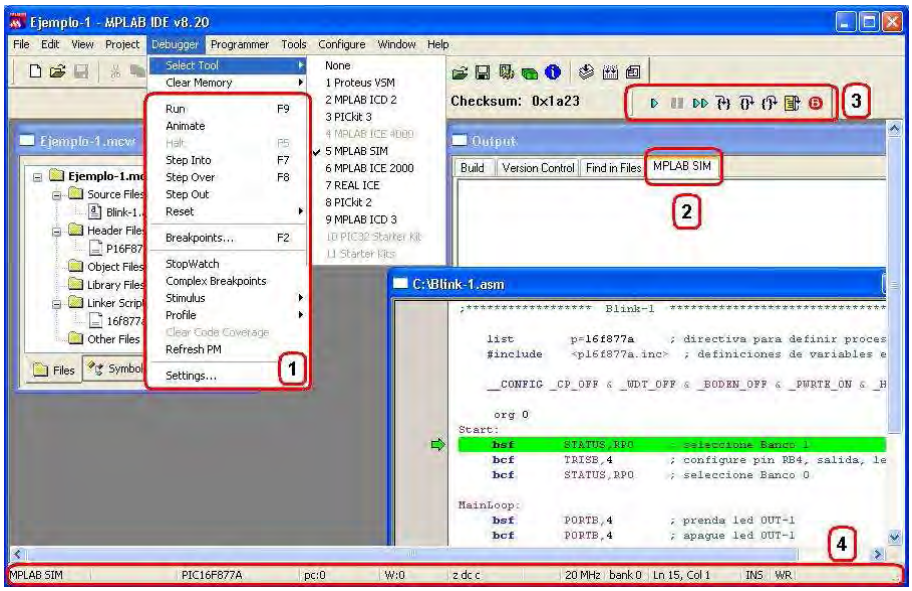


Figura 14.15 Ventana de simulación.

Como ya tenemos el código **C:\Blink_1.asm** creado y el proyecto **C:\Ejemplo-1.mcp** compilado, estamos listos para realizar la **simulación**.

Primero debemos escoger el tipo de depurador, como muestra la Figura14.15; esto se realiza mediante el menú de opciones **Debugger/Select Tool/MPLAB SIM** y, una vez seleccionado el modo de simulación, hay varios cambios que se advierten en la ventana de simulación y que enumeramos a continuación:

1. El menú del *debugger* se amplía con otras opciones para simulación.
2. En la ventana **Output** se genera una pestaña nueva **MPLAB SIM.**
3. En la barra de herramientas aparece una Caja de herramientas de **Depuración.**
4. En la barra de **Status** cambia el modo a **MPLAB SIM** con cajones nuevos.

Detallando el numeral 3, la nueva Caja de herramientas de depuración es:

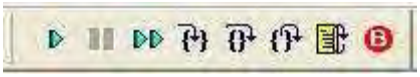


Figura 14.16 Caja de herramientas de depuración.

Esta caja de herramientas corresponde a los íconos de:

 *Run* (correr o arrancar)

 *Halt* (detener)

 *Animate* (animación)

 *Step into* (paso interno)

 *Step over* (paso externo)

 *Step out* (paso de salida)

 *Reset* (reset)

 *Breakpoints* (puntos de ruptura)

La barra de **Status** (parte inferior) cambia de modo **NONE a MPLAB SIM.**

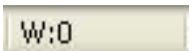


Figura 14.17 Barra de status (parte inferior).

En esta barra podemos destacar:



Contador de programa **pc** = 0



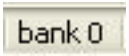
Registro **W** = 0



Banderas **Z** = Zero, **DC** = *Digit Carry*, **C** = *Carry*
En minúsculas z,dc,c = 0 y en mayúsculas Z,DC,C = 1



Velocidad del oscilador = 20 MHz



Banco = 0

Ahora comenzamos a trabajar con la caja de herramienta de depuración.

PASO 1 Generar reset

El primer paso, una vez compilado el proyecto, es generar un *reset* para que el microcontrolador simulado comience con los valores iniciales; esto se hace con el ícono  *Reset* subrayado en la Figura 14.18, la línea verde indica la línea que se va a ejecutar, donde se encuentra la PC contador de programa, a medida que se realiza la simulación. Observe cómo las primeras líneas

```
        list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc> ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

org 0
```

no generan código, son **directivas** que el **compilador** utiliza para:

```
        list p = 16f877a
```

Definir el microcontrolador que se vaya a utilizar.

```
#include <p16f877a.inc>
```

Especifica el archivo con las definiciones de registros del microcontrolador.

```
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF & _CPD_OFF
```

Define cada uno de los bits de conFiguración del microcontrolador.

```
org 0
```

Define la posición o el comienzo del programa.

```
NOTA: Directivas son palabras clave que utiliza el compilador para
definir la información necesaria para la estructura del programa.
```

Véase el **Apéndice D: Directivas del MPASM**.

En la Figura 14.18 se muestra cómo queda la ventana luego del comando *reset*; observe cómo la barra de **Status** no ha cambiado en ningún valor.

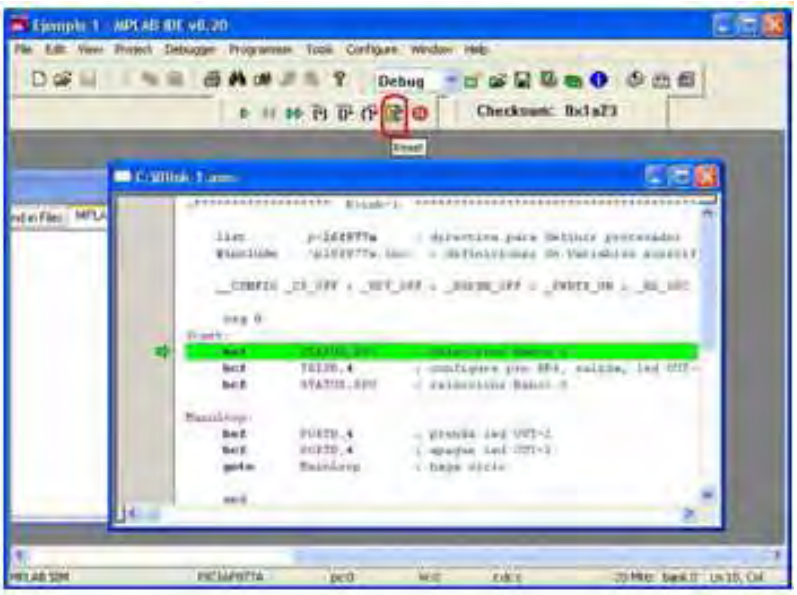


Figura 14.18 Ventana de simulación *reset*.

El **PC = 0** está apuntando a la primera instrucción, pero todavía no se ha ejecutado.

```
bsf STATUS,RP0 ; seleccione Banco 1
```

PASO 2 Primer paso a paso

Los pasos que seguimos a continuación no forman parte de una secuencia obligada, sino de la forma como seguimos este específico programa ejemplo. Luego se realiza el primer paso a paso, haciendo **clic** en el ícono  *step over* (este ícono se recomienda para seguir el programa paso a paso para no entrar en los comandos que tienen varias instrucciones incluidas, tales como subrutinas); este paso genera los primeros cambios en la barra de **Status**.

En la Figura 14.19 se observa cómo se resalta la segunda instrucción **bcf TRISB,4**, y en la barra de *Status* (parte inferior) cómo **pc = 0X1**, el contador de programa apunta a la instrucción **0X1**; esto luego de haberse ejecutado la instrucción **pc = 0X0** bsf STATUS,RP0 o sea el selector de banco **bank = 1**. Es necesaria esta instrucción para tener acceso al registro **TRISB** que está en la posición **086h,** o sea en el **banco 1;** en la barra de *Status* (parte inferior) se observa cómo el banco cambia a **bank 1**.

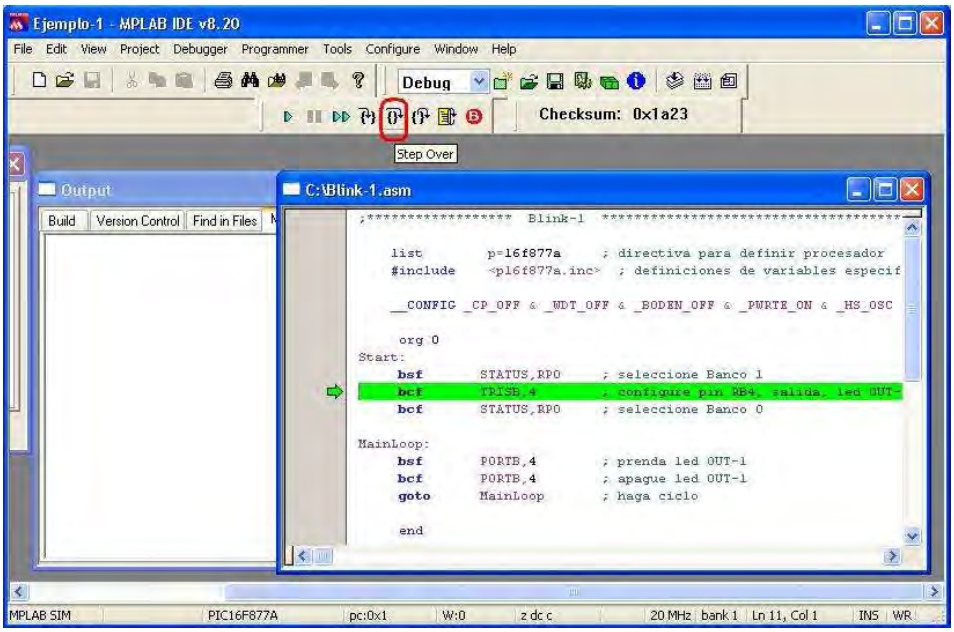


Figura 14.19 Ventana de simulación *step-over 1*.

Como paso siguiente, para visualizar algunos registros, agregaremos la ventana **Watch** (Figura 14.20).

Agregar la ventana Watch y registros SFR

Como se muestra en la Figura 14.20, se agrega la ventana de **Watch**, en la cual se puede ver el contenido de los registros que deseamos verificar; en el menú de opciones **View/Watch** crea la ventana **Watch**, en ella aparecen hasta 4 pestañas marcadas como **Watch 1, Watch 2, Watch 3, Watch 4**, donde podemos agregar los registros que deseemos ver.

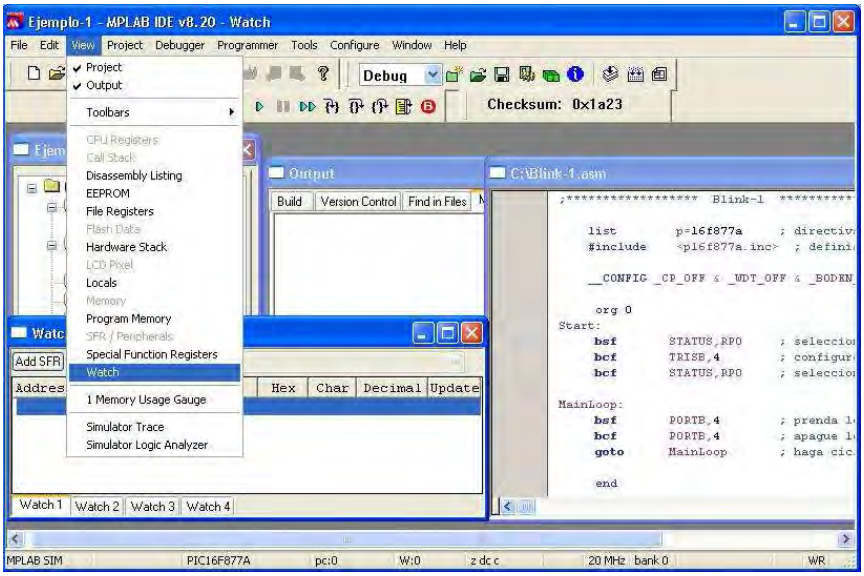


Figura 14.20 Agregar ventana Watch.

Para agregar registros a la ventana **Watch** (Figura 14.21) se selecciona el registro del menú desplegable; para nuestro caso, los registros **TRISB, PORTB** y **STATUS** se adicionan haciendo clic en la casilla **Add SFR.**

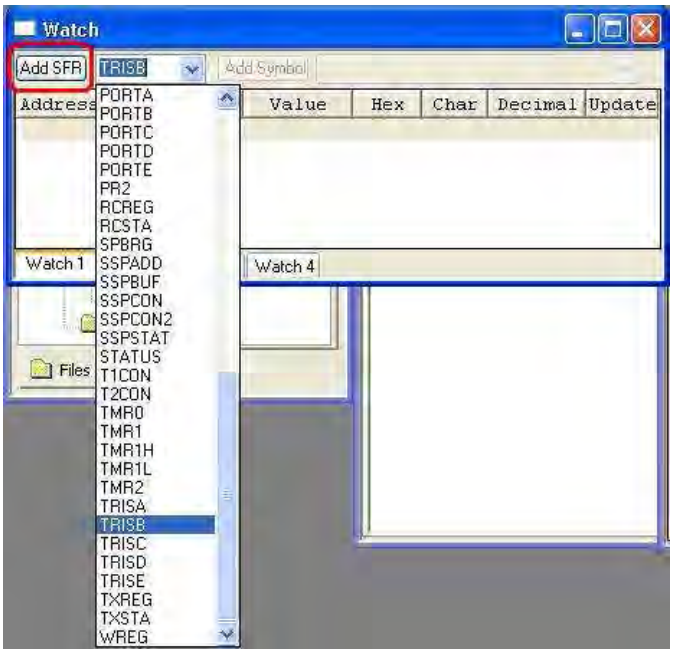


Figura 14.21 Agregar registros a la ventana Watch.

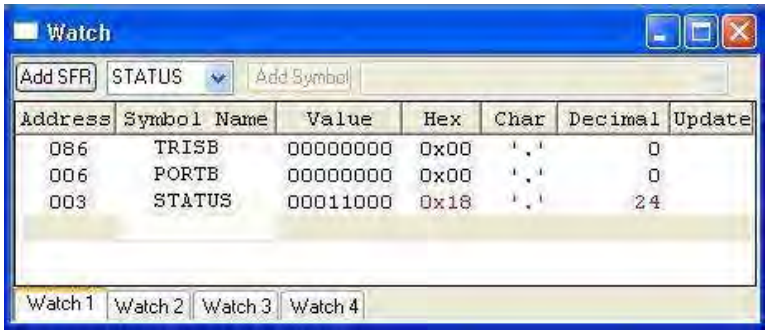


Figura 14.22 Ventana Watch y registros.

Al terminar la ventana **Watch** debe aparecer como se muestra en la Figura 14.22. Luego de agregar la ventana **Watch** podemos ver el cambio de los registros agregados a medida que se ejecuta el paso a paso.

PASO 3 Segundo paso a paso

Con el siguiente paso a paso, con el ícono  *step over*, se ejecuta la segunda instrucción **bcf TRISB,4**, como se muestra en la Figura 14.23, lo que altera el contenido de **TRISB** que pasa de **1111 1111** a **1110 1111**, configurando el bit 4 del **PUERTO B** como salida, que toma los valores de: **11101111** en binario **= 0XEF** en hexadecimal **= 239** en decimal, en la ventana **Watch** y en la barra de Status (parte inferior) el contador de programa cambia a **pc = 0X2** y el selector de banco **bank = 1**.

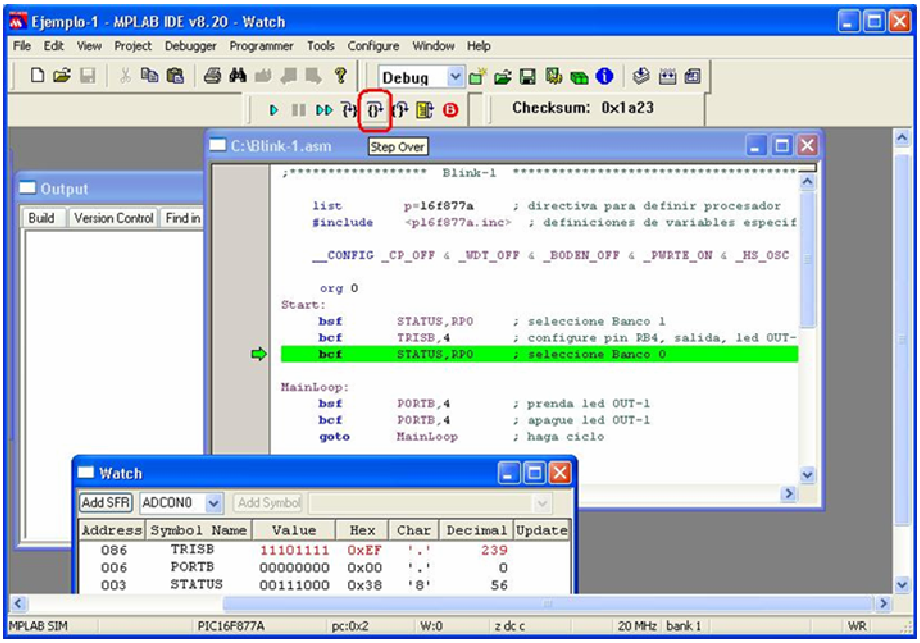


Figura 14.23 Ventana de simulación *step-over 2*.

De esa forma el contenido de la ventana es como se muestra en la Figura 14.24.

PASO 4 Tercer paso a paso

Con el siguiente paso a paso, con el ícono  *step over*, se ejecuta la tercera instrucción, **bcf STATUS,RP0**, se puede observar el cambio en el registro **STATUS** que cambió de **0011 1000 a 0001 1000**, donde los bits **RP1:RP0 <6:5>** cambian de **<01> a <0:0>**, o cambió al **banco = 0**, en la barra de **Status** (parte inferior) el contador de programa cambia a **pc = 0X3** y el selector de banco a **bank = 0**; el resultado se observa en la Figura 14.24.

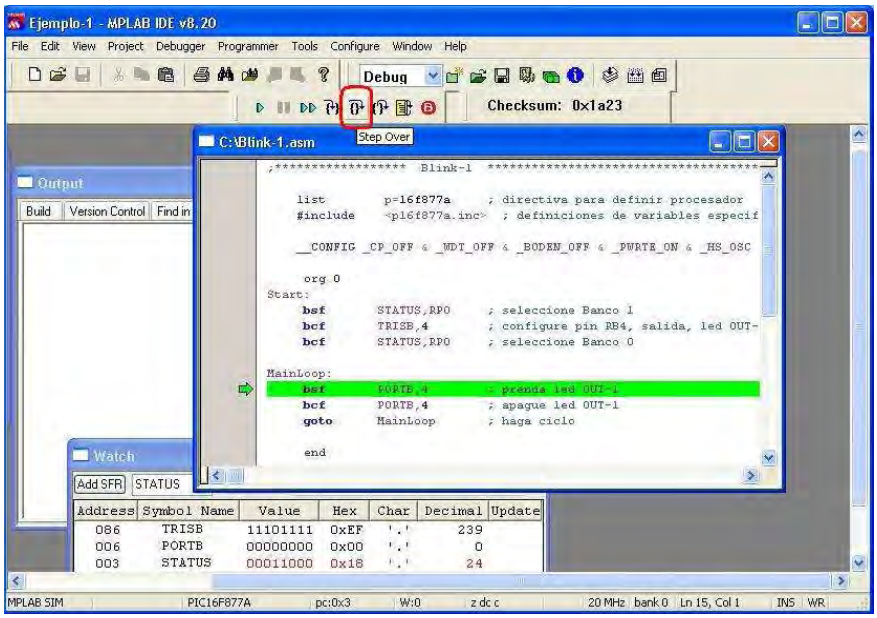


Figura 14.24 Ventana de simulación *step-over 3*.

Hasta aquí lo que hemos hecho es simplemente configurar el puerto **PORTB bit 4** como salida, debido a que el registro **TRISB** se encuentra en la posición **086h**, o sea en el banco 1, debimos cambiar al **banco 1** y luego volver al **banco 0**, que son las tres instrucciones que ejecutamos:

```
        Bsf     STATUS,RP0      ; seleccione Banco 1
        bcf     TRISB,4         ; configure pin RB4, como salida, led OUT-1
        bcf     STATUS,RP0      ; seleccione Banco 0
```

De aquí en adelante el programa entra en el ciclo infinito *MainLoop,* en el cual el led **OUT_1** se prende y apaga de forma continua; las instrucciones que seguiremos paso a paso son las siguientes:

```
MainLoop:
        bsf     PORTB,4         ; prenda led OUT-1
        bcf     PORTB,4         ; apague led OUT-1
        goto    MainLoop        ; haga ciclo
        end
```

PASO 5 Cuarto paso a paso

Con el siguiente paso a paso, con el ícono  *step over*, se ejecuta la cuarta instrucción, **bsf PORTB,4**, se puede observar el cambio en el registro **PORTB** que cambió de **0000 0000 a 0001 0000**, donde el bit **<4>** cambia de **0** apagado a **1** encendido, el **PORTB** toma los valores de **0001 0000** en binario **= 0X10** en hexadecimal **= 16** en decimal, en la ventana **Watch**, en la barra de **Status** (parte inferior) el contador de programa cambia a **pc = 0X4**, como se aprecia en la Figura 14.25, que es la siguiente instrucción a realizar.

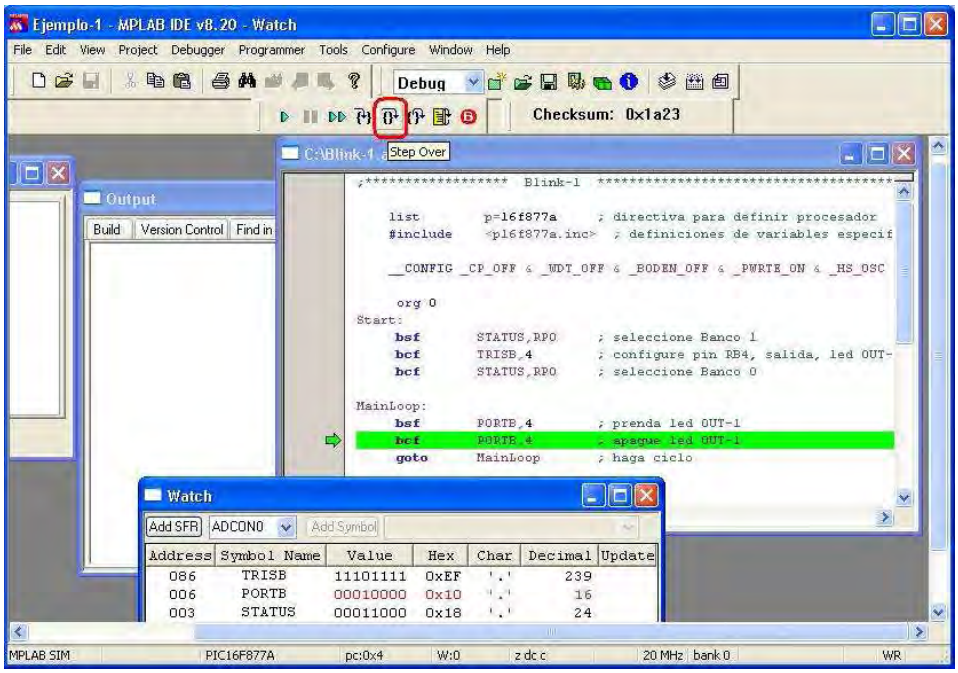


Figura 14.25 Ventana de simulación *step-over 4*.

De esta manera se enciende el led **OUT-1** que está conectado a la salida del **pin 4** del **PUERTOB**. A continuación veremos como se apaga el led **OUT-1**.

PASO 6 Quinto paso a paso

Con el paso a paso siguiente, con el ícono  *step over*, se ejecuta la quinta instrucción, **bcf PORTB,4**,se puede advertir el cambio en el registro **PORTB** que cambió de **0001 0000 a 0000 0000**, donde el bit **<4>** cambia de **1 encendido a 0**

apagado, el PORTB toma los valores de **0000 0000** en binario **= 0x00** en hexadecimal **= 0** en decimal, en la ventana **Watch**, en la barra de **Status** (parte inferior) el contador de programa cambia a **pc = 0X5**, como se aprecia en la Figura 14.26.

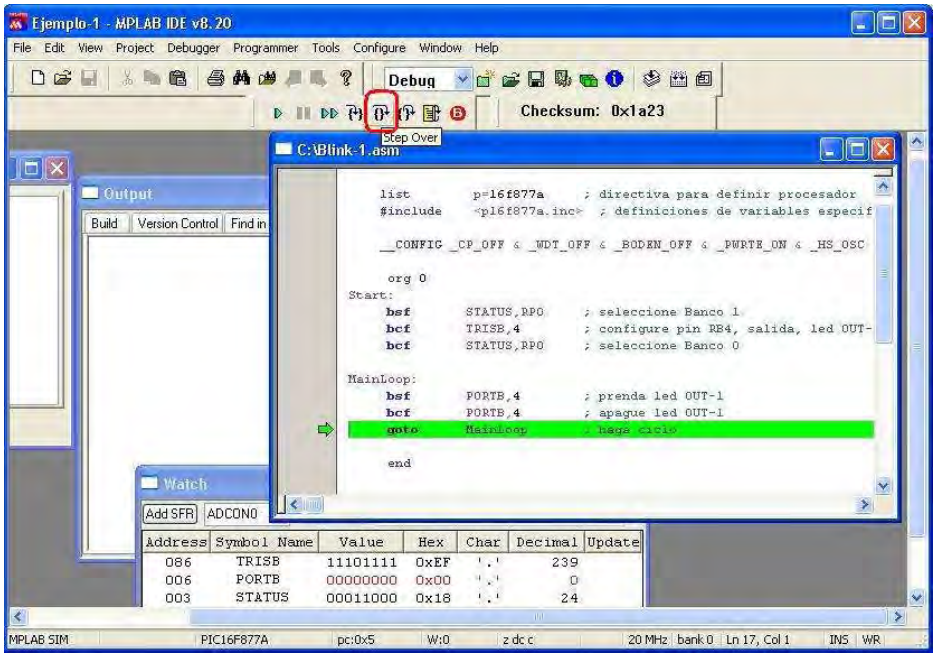


Figura 14.26 Ventana de simulación *step-over* 5.

En el paso siguiente se va de nuevo al comienzo del ciclo infinito **MainLoop,** que es de nuevo el **paso 5**, donde se enciende el led **OUT-1** y así sucesivamente se repite este ciclo infinito. Con el paso a paso siguiente, con el ícono  *step over*, se ejecuta la sexta instrucción, **goto MainLoop,** en la barra de **Status** (parte inferior) el contador de programa cambia a **pc = 0X3**, como se aprecia en la Figura 14.27, que es de nuevo el ciclo de encendido y apagado del led.

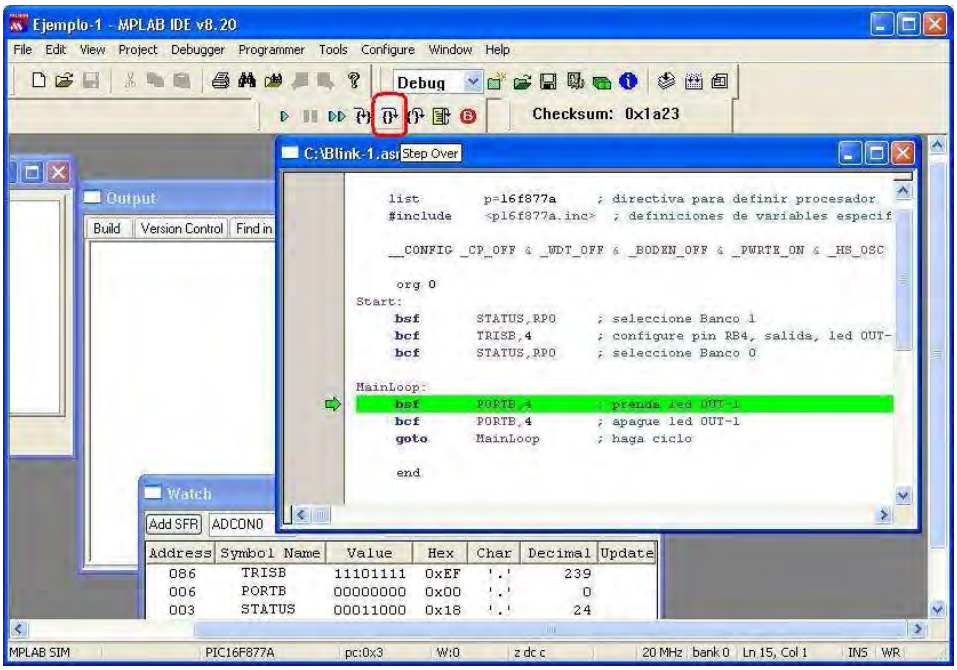


Figura 14.27 Ventana de simulación *step-over* 6.

PASO 7 Sexto paso a paso

Una vez en este punto podemos repetir las acciones desde el paso a paso 5.

14.4 PROGRAMACIÓN

La **programación** consiste en enviar el programa que está en la PC al microcontrolador; para esto hace falta la intervención de varios elementos que son indispensables para que esta labor se realice. Elementos para la **programación**:

- Archivo para programar
- Software de programación
- Programador
- Microcontrolador

ARCHIVO PARA PROGRAMAR

El archivo para programar debe ser un archivo tipo ***.HEX**, o también un archivo ***.ASM** que debemos compilar antes.

SOFTWARE DE PROGRAMACIÓN

El software de programación puede ser cualquiera de los siguientes: **MPLAB IDE, WinPIC800, IC-PROG**, o cualquier otro programa que pueda programar microcontroladores PIC (se encuentran muchos descargables gratis en Internet); para nuestro caso, utilizaremos el **MPLAB IDE,** el cual se asume que ya está instalado en su PC.

PROGRAMADOR

Éste puede ser cualquiera de los programadores descritos en la Parte 1 en el **Capítulo 2.0, Subsección 2.2.1: Programador (*programmer*)**; en nuestro caso, usaremos el **PICkit 2** en conjunto con el sistema de evaluación **PIC16F8XX**. Conviene anotar que muchos usuarios utilizan programadores que tienen bases y se conectan vía **RS232** con la **PC**, lo cual no recomendamos, pues este método requiere que se intercambie el micro entre el programador y la tarjeta de aplicación, lo que trae como consecuencia el daño de los pines por el esfuerzo de introducir y extraer repetidamente el microcontrolador.

MICROCONTROLADOR

El microcontrolador puede ser cualquiera de la serie PIC micro; en nuestro caso utilizamos el **PIC16F877A**, que es uno de los que soporta la herramienta **PIC16F8XX**.

14.4.1 PROGRAMACIÓN CON PICKIT 2 PROGRAMMER

El programa **PICkit 2 Programmer** viene con el kit del **PICkit 2**, o se puede descargar de la página de **Microchip** y se asume ya instalado en la PC. El archivo para programar **C:\Ejemplos_ASM\Blink_2\Blink_2.hex** se debe haber copiado de nuestra página web, de la carpeta **X:\Ejemplos_ASM**. Este programa prende y apaga el led **OUT-1** de forma intermitente cada ½ segundo (0.5 segundos), en la tarjeta del sistema de evaluación **PIC16F8XX**.

PASO 1 Conectar kit de trabajo

Conectar el kit de trabajo conformado por el programador PICkit 2 y el sistema de evaluación PIC16F8XX, tal como se muestra en la Figura 14.28 y físicamente como aparece en la Figura 14.29.



Figura 14.28 Conexión kit de trabajo PICkit 2 y PIC16F8XX.

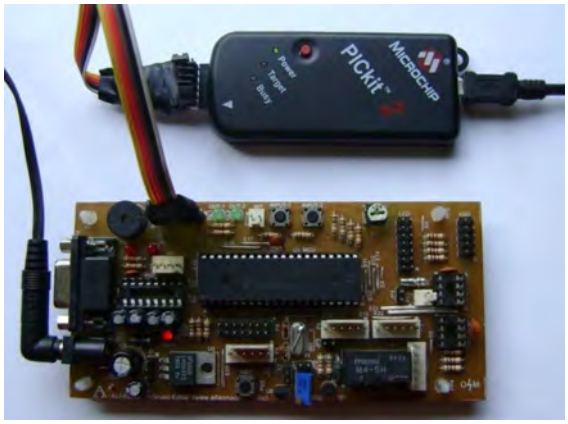


Figura 14.29 Kit de trabajo PICKIT 2 y PIC16F8XX físico.

PASO 2 Cargar PICkit 2 Programmer

Luego se ejecuta el programa **PICkit 2 Programmer** (como se ejecuta usualmente un programa en su PC), y se escoge el microcontrolador **PIC16F877A** en el menú desplegable **Device**: la ventana resultante es como se muestra en la Figura 14.30.

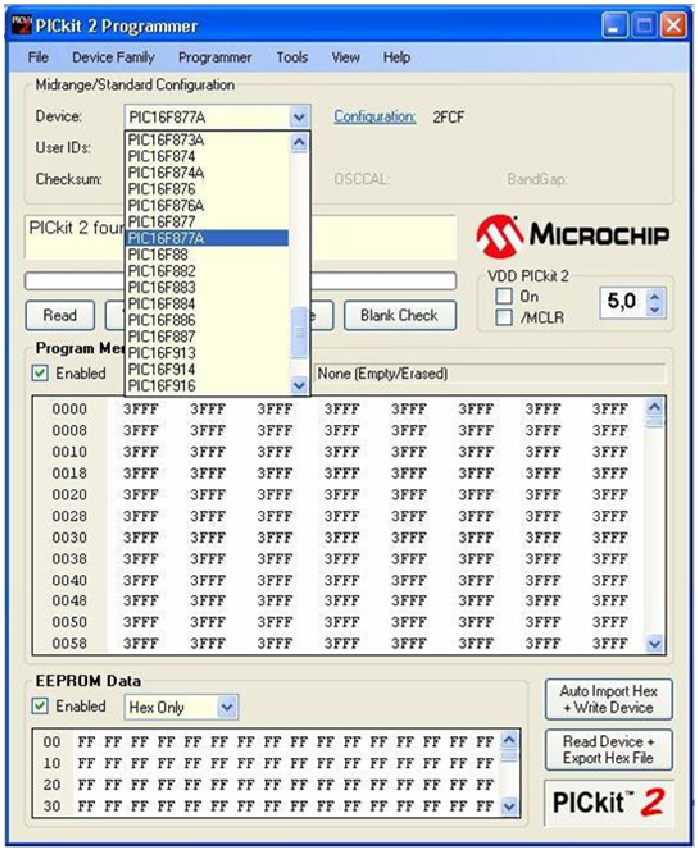


Figura 14.30 PICkit 2 Programmer.

PASO 3 Borrar la memoria de programa

Después se debe borrar el contenido de la memoria de programa del dispositivo, haciendo **clic** en el cuadro **Erase**, en la ventana de memoria todas las posiciones de memoria deben estar en 3FFF (todos los bits en "1"), y en el cuadro de mensajes se debe mostrar el mensaje "Erasing device... Complete"; la ventana es como se muestra en la Figura 14.31.

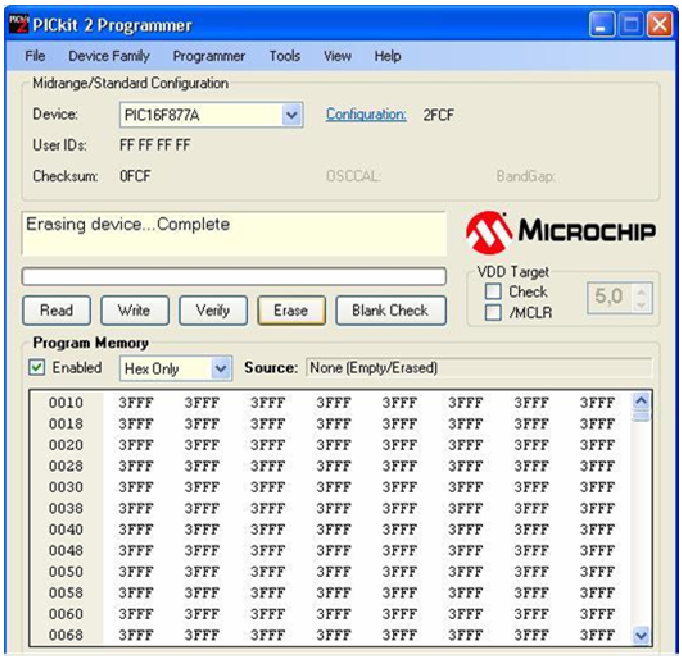


Figura 14.31 PICkit 2 Borrado.

PASO 4 Selección de archivo para programar

Una vez aquí, se selecciona el archivo que se va a programar, desde el menú de opciones **File/Import Hex** que despliega la ventana **Import Hex File**, y allí se selecciona el archivo que se va a cargar; en nuestro caso, se escoge el archivo en la ruta **C:\Ejemplos_ASM\Blink_2\Blink_2.hex**, y damos **clic** en la casilla **Abrir,** tal como muestra la Figura 14.32. Este programa prende y apaga el led **OUT-1** de forma intermitente cada ½ segundo, en la tarjeta de evaluación PIC16F8XX.

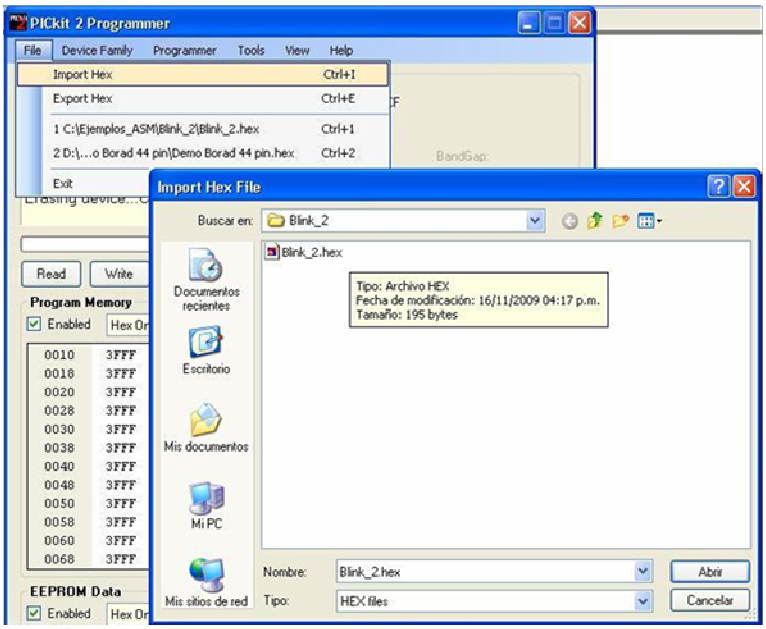


Figura 14.32 PICkit 2 importar archivo Hex.

En el cuadro de mensajes se observa "Hex file sucessfully imported" como se muestra en la Figura 14.33, el archivo se carga en las posiciones de memoria desde 0X0000 a 0X0015; también se aprecian los bits de **Configuración 2F02**, el **User IDs: 00 00 FF FF** y el respectivo *checksum:* **B186**; la ruta del archivo cargado en memoria está en el cuadro **Source: C:\Ejemplos_ASM\Blink_2\Blink_2.hex**.

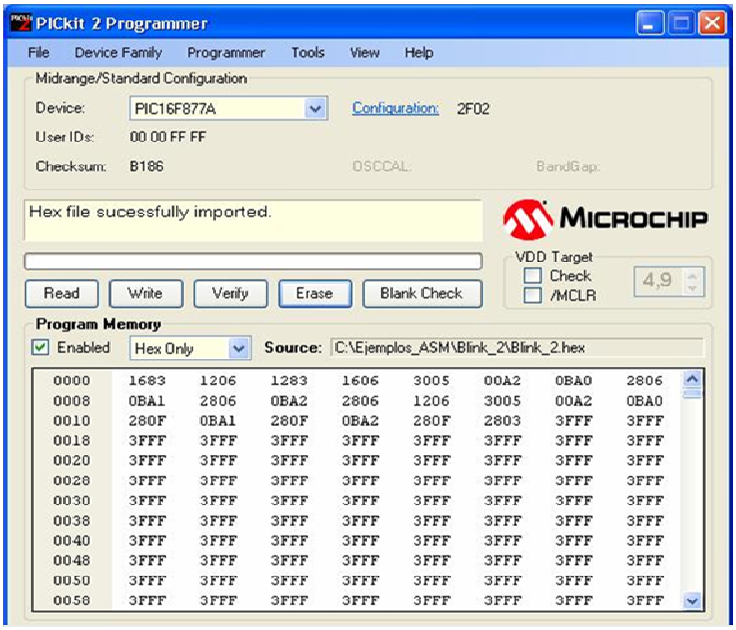


Figura 14.33 PICkit 2 archivo Hex importado.

PASO 5 Programar el Microcontrolador

En este paso ya estamos listos para programar el microcontrolador, haciendo **clic** en el cuadro **Write** se inicia el proceso de programación hasta concluir con el mensaje "*Programming Successful*" que nos indica que ocurrió una programación exitosa, tal como se aprecia en la Figura 14.34.

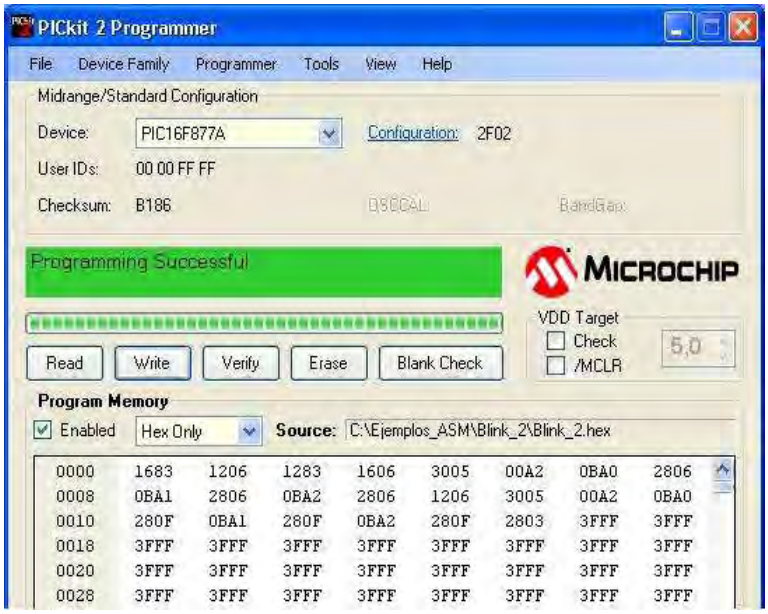


Figura 14.34 PICkit 2 Programación.

PASO 6 Verificar funcionamiento

Ahora (Figura 14.35a, b) puede verificar el funcionamiento del programa en la tarjeta del sistema de evaluación **PIC16F8XX**; este programa prende y apaga el led **OUT-1** de forma intermitente cada ½ segundo.

A **B**

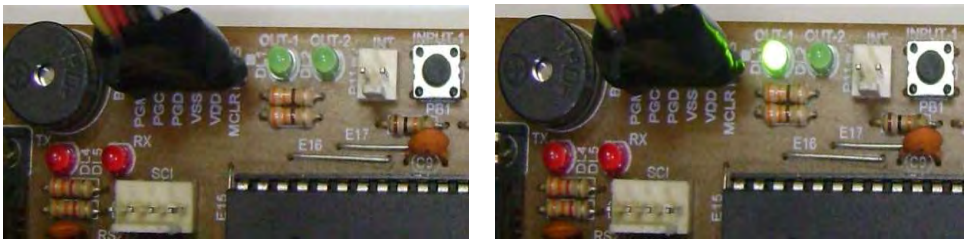


Figura 14.35 Funcionamiento LED_OFF LED_ON.

14.5 DEPURACIÒN

La depuración es lo que llamamos la función *debugger*, y consiste en poder verificar los registros del microcontrolador cuando está insertado en la tarjeta de la aplicación, y el programa se encuentra corriendo (en modo de *run*), de tal forma que podemos verificar si los registros tienen un valor adecuado a medida que el programa está interactuando con los dispositivos a los cuales el microcontrolador está conectado con el mundo real.

14.5.1 ICSP

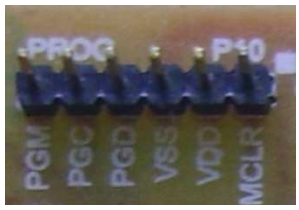
Esto es posible mediante el protocolo **ICSP** (*In-Circuit Serial Programing*) o Programación Serial en Circuito, el cual hace uso de los pines, como se muestra en la Figura 14.36, con el nombre del pin ICSP, el esquemático asociado y los pines a los cuales van conectados en el microcontrolador (Figura 14.37a, b y c).

ICSP In-Circuit Serial Programing

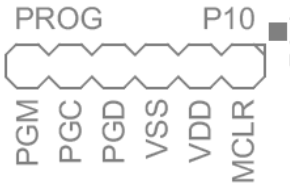
PIN	NAME	BOARD
1	MCLR/VPP	MCLR
2	VDD	VDD
3	VSS	VSS
4	ICSPDAT	RB7
5	ICSPCLK	RB6
6	AUX	NC

Figura 14.36 Programación serial en circuito.

A



B



C

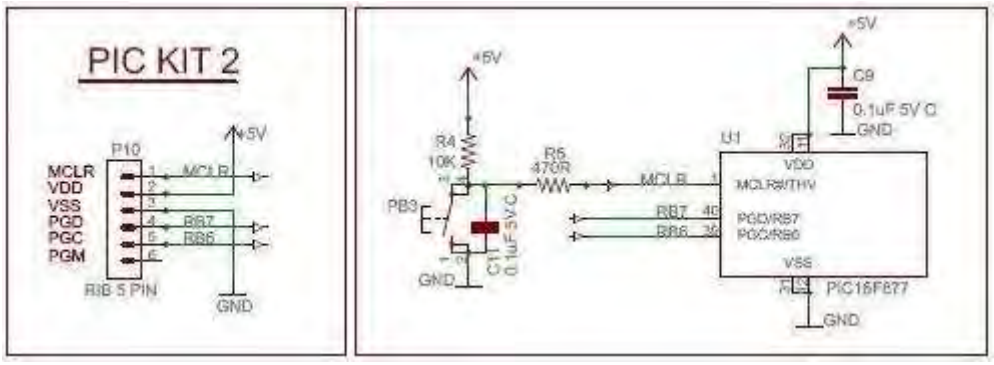


Figura 14.37 Circuito de conexión ICSP.

14.5.2 PROGRAMA EJEMPLO

Este programa consiste en mostrar secuencialmente los dígitos del 0 al 9 en un display de 7 segmentos; los segmentos están conectados al Puerto D y el control del dígito en el Puerto E<2>, todos son de lógica positiva, o sea se acciona o prende con un “1” en la salida. En la Figura 14.38 se aprecia el diagrama de flujo del programa ejemplo.

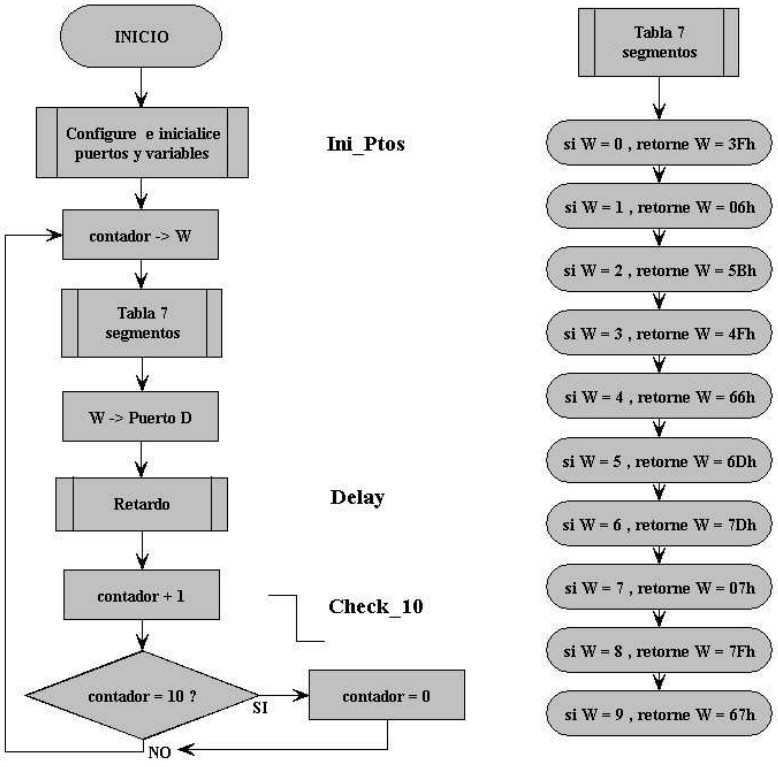


Figura 14.38 Diagrama de flujo ejemplo.

Los nombres de las subrutinas están afuera del diagrama de flujo que se dejó con las funciones que luego se reemplazarán por subrutinas. Este programa **7_segmentos_1A.asm** se encuentra en **X:\Ejemplos_ASM** y se debe copiar en **C:** para quedar como **C:\Ejemplos_ASM\7_segmentos_1\7_segmentos_1A.asm.**

14.5.3 PASOS PARA DEPURACIÓN EJEMPLO

A continuación realizaremos los pasos para ver una depuración del programa ejemplo visto anteriormente.

PASO 1 Conexión del kit de trabajo

La conexión es tal como se vio en las Figuras 14.28 y 14.29 de **Programación,** más la conexión del módulo *Display,* de 7 segmentos, 4 dígitos pequeños **MOD_7S_4D_L (2009-05-06),** debe quedar como se ilustra en la Figura 14.39.



Figura 14.39 Kit de trabajo Depuración.

PASO 2 Cargar el proyecto

Cargar el proyecto significa ejecutar el programa **MPLAB** y cargar el **proyecto** o *workspace* que se va a depurar; la forma más correcta de abrir un proyecto es por la vía del *workspace* desde el menú de opciones *File/Open Workspace…,* como se muestra en la Figura 14.40; en nuestro caso la dirección del proyecto o *workspace* ejemplo está en **C:\Ejemplos_ASM\7_segmentos_1\7_segmentos_1.mcw**.

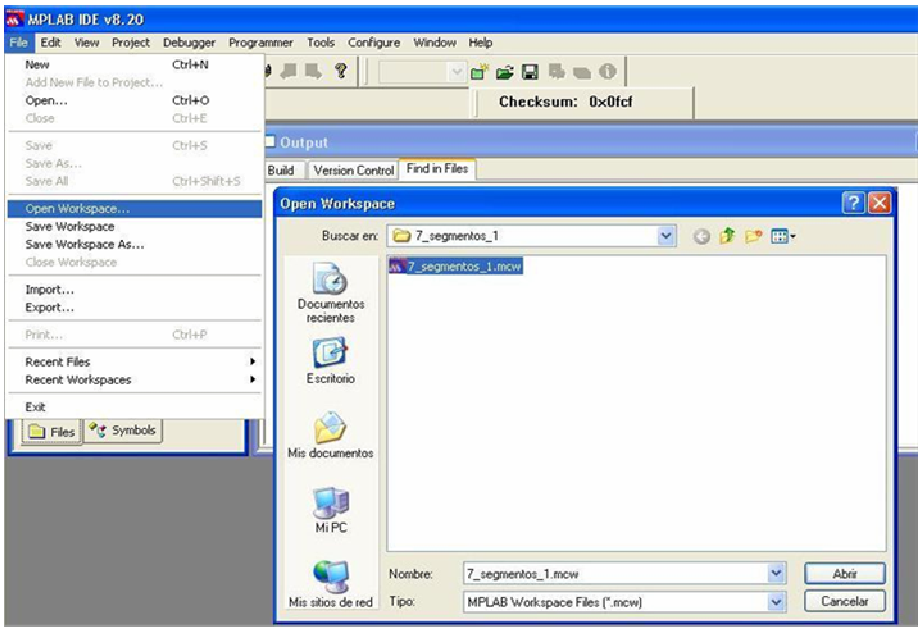


Figura 14.40 Cargando *workspace*.

Conviene anotar que previamente se debió copiar la carpeta incluida en nuestra página web **X:\Ejemplos_ASM** al disco **C:**; esta carpeta contiene todos los ejemplos hechos en ensamblador. Al ser cargado el **proyecto** o *workspace*, carga el ambiente como la última sesión en la cual se salvó; de esta manera aparece la ventana como se muestra en la Figura 14.41.

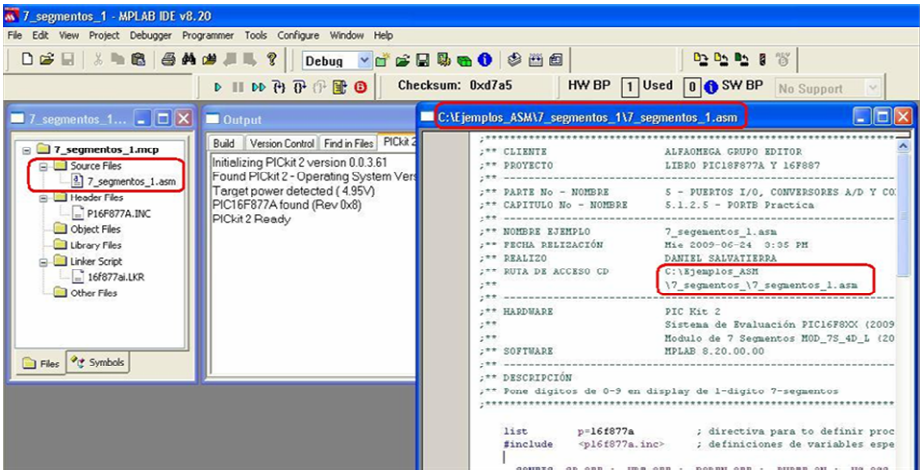


Figura 14.41 Ventana del *workspace* ejemplo.

En la Figura 14.41 se observa cómo en la ventana del **proyecto** está cargado el archivo **C:\Ejemplos_ASM\7_segmentos_1\7_segmentos_1.asm** que es el archivo sin rutinas; ahora cambiaremos este archivo por el archivo con subrutinas archivo **C:\Ejemplos_ASM\7_segmentos_1\7_segmentos_1A.asm**; la forma de hacerlo es la siguiente.

Procedimiento para remover y agregar archivos a un proyecto

Primero removemos el archivo del **proyecto**; en la ventana del **Proyecto.mcw** desde la carpeta **Source Files** seleccionando el archivo que se vaya a remover, y con el botón derecho se despliega un menú y se selecciona **Remove**, y queda removido el archivo del proyecto; estos dos pasos están descritos en la Figura 14.42a y b. Observe cómo la carpeta de **Source Files** queda vacía.

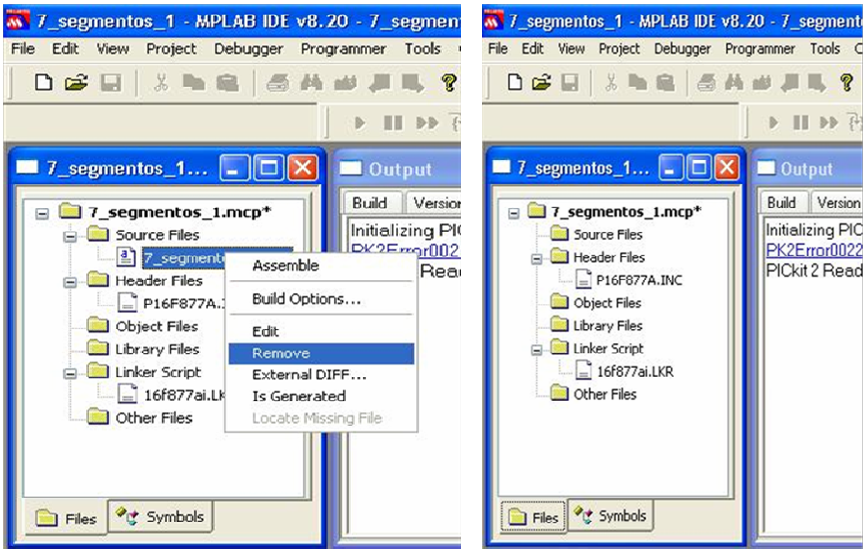


Figura 14.42 Remover archivo del proyecto.

El segundo paso consiste en agregar el archivo **7_segmentos_1A.asm** que se encuentra en la misma carpeta; en la ventana del **Proyecto.mcw** seleccione la carpeta **Source Files** y con el botón derecho se despliega un menú, donde selecciona **Add Files...** el cual despliega la ventana **Add Files to Project**, y allí aparecen los archivos ***.asm** de los que seleccionamos **7_segmentos_1A.asm** haciendo **clic** en la casilla **Abrir**, tal como se muestra en la Figura 14.43**.**

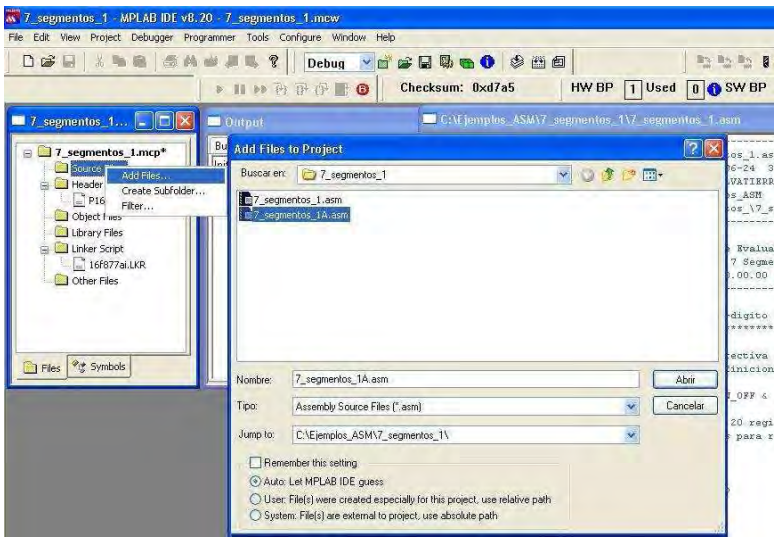


Figura 14.43 Adicionar archivo al proyecto.

Una vez que se ha adicionado el archivo en la ventana de **Proyecto.mcw**, en la carpeta **Source Files** se hace doble **clic** en el archivo **7_segmentos_1A.asm** y aparece en la ventana de trabajo, como se muestra en la Figura 14.44. Observe que pese a que también el archivo anterior **7_segmentos_1.asm** está en la ventana de trabajo, no es considerado por el compilador y no genera errores; sin embargo, se aconseja borrarlo para que no genere errores de procedimiento futuros.

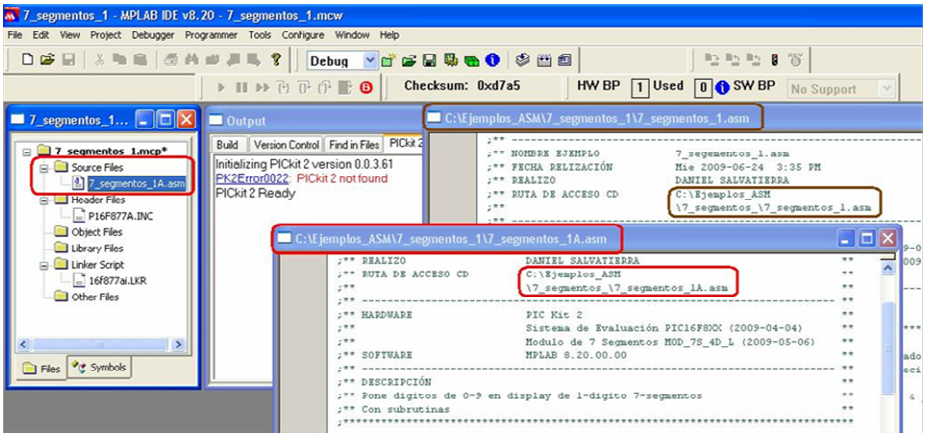


Figura 14.44 Proyecto con archivo nuevo.

PASO 3 Compilar proyecto

Como ya hemos visto, la compilación del proyecto se hace desde las mismas tres partes, desde el menú de opciones *Proyect/Build All*, con las teclas rápidas **Ctrl+F10**, o con el ícono  de la barra de herramientas; en la ventana **Output**, en la pestaña **Build**, se aprecia el resultado de la compilación; si no hay errores aparece el mensaje de "Build Succeeded", tal como se observa en la Figura 14.45.

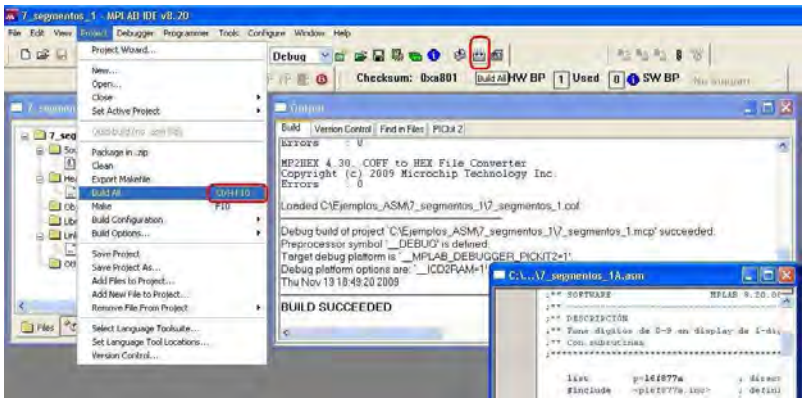


Figura 14.45 Compilar proyecto.

PASO 4 Selección de herramienta de depuración

Se debe escoger la herramienta de depuración (Figura 14.46) desde el menú de opciones **Debugger/Select Tool**, en nuestro caso, el **PICkit 2**.

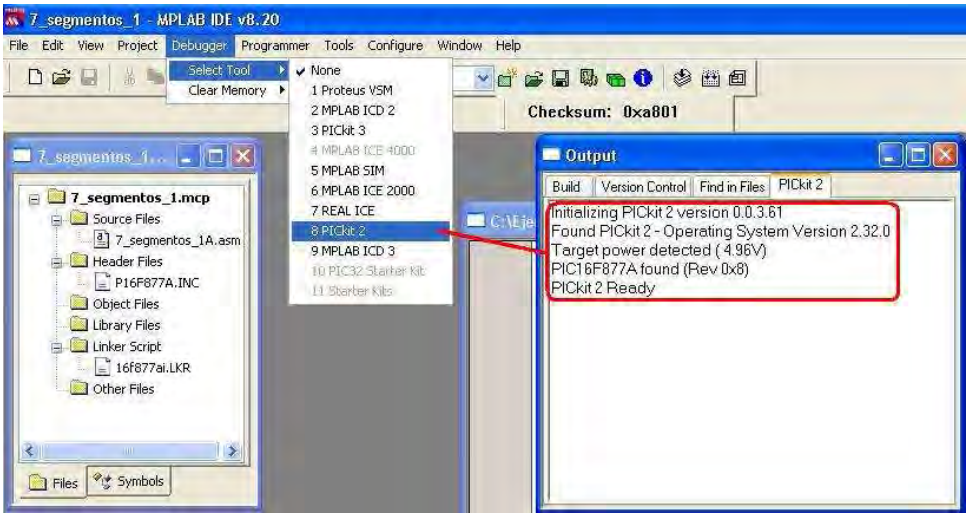


Figura 14.46 Selección de herramienta de depuración

Una vez seleccionada esta opción, aparece un mensaje en la ventana **Output** en la pestaña **PICkit 2**, tal como ilustra la Figura 14.46, indicando que encontró el **PICkit 2** y la versión de su sistema operativo, el voltaje a que está alimentada la tarjeta, que encontró el micro conectado **PIC16F877A** y su revisión, al final dice que el **PICkit 2** está listo para recibir comandos.

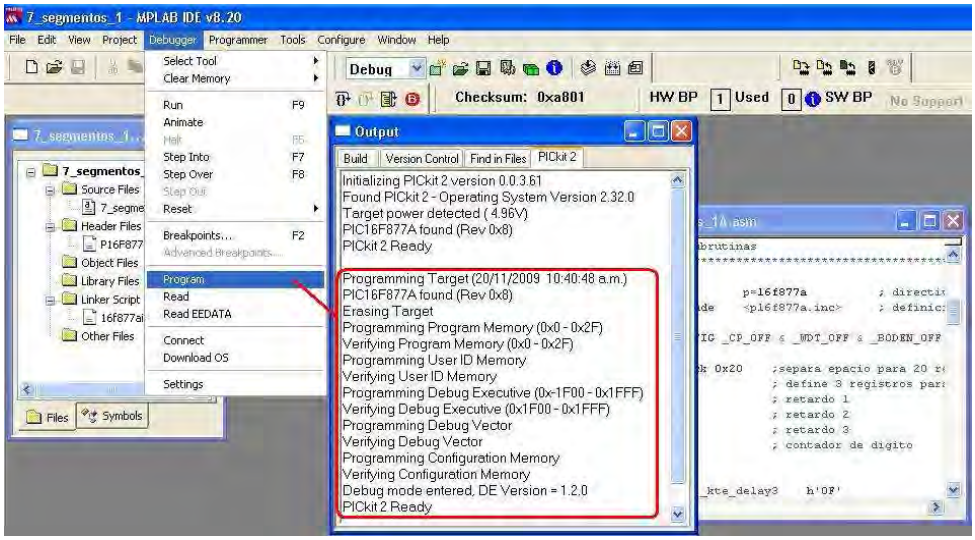


Figura 14.47 Programación del micro.

PASO 5 Programación del micro

Estando en depuración igual, tenemos que programar el micro; este proceso se aprecia en la Figura 14.47, observe cómo cada uno de los pasos realizados por la función de programación aparecen en la ventana **Output** en la pestaña de **PICkit 2**. Este proceso consiste en "quemar" el micro con el programa y en adicionar también una pequeña aplicación para depuración en las posiciones de programa **0X1F00** a **0X1FFF**, y reserva las posiciones de memoria **0X1E5** a **0X1EF** además de **0X70, 0XF0, 0X170, 0X1F0**.

PASO 6 Agregar la ventana Watch y registros GPR

Para seguir el programa paso a paso en el modo de depuración, resulta muy útil revisar los registros; por tanto, se agrega la ventana **Watch** con los registros **PORTD, WREG, STATUS**, tal como se hizo en la Sección 1.4.3, en el Paso 2 Primer paso a paso, en el procedimiento Agregar la ventana Watch y Registros SFR. En esta ventana **Watch** se agregan los registros que están en las posiciones **0X20 delay1, 0X21 delay2, 0X22 delay3, 0X23 contador**; este procedimiento se describe a continuación (Figura 14.48).

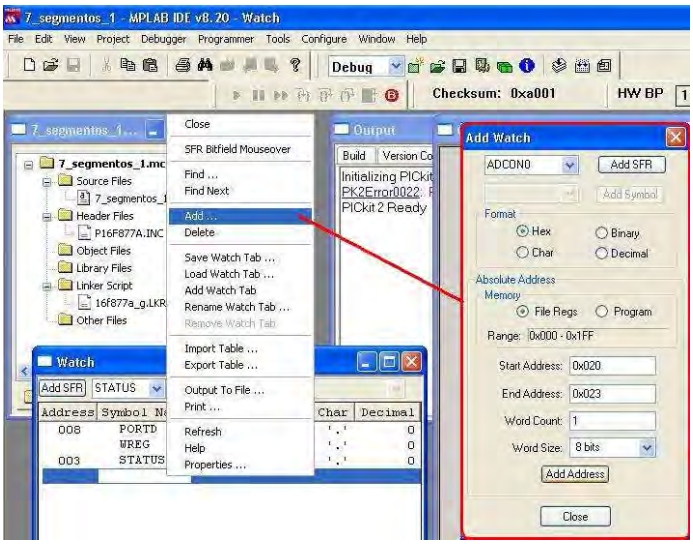


Figura 14.48 Adición de registros GPR.

Detenidos en la ventana **Watch** con el botón derecho se despliega un menú en el cual se selecciona **Add...**, y allí aparece la ventana **Add Watch** donde se agregan las direcciones en la casilla **Star Addres**: con el valor **0X020** y en la casilla **End Address**: con el valor **0X023** y haciendo **clic** sobre el cajón **Add Address** se agregan estas direcciones, para luego aparecer en la ventana **Watch** (Figura 14.49).

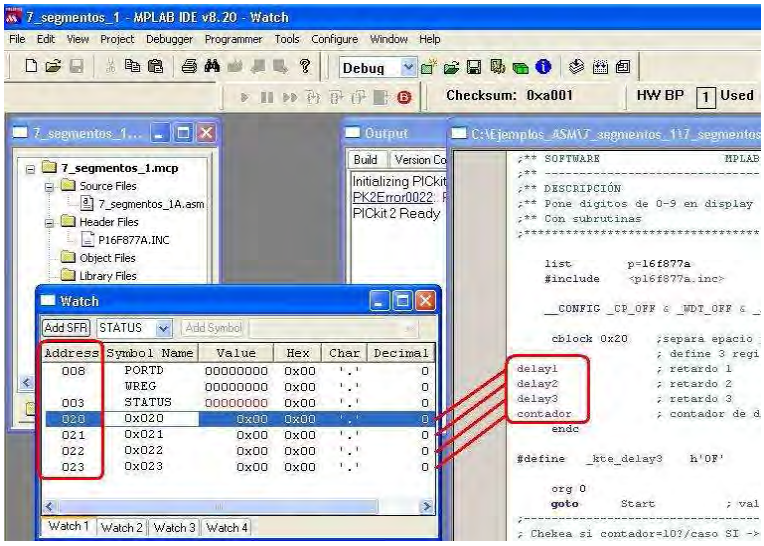


Figura 14.49 Ventana Watch y registros GPR.

Estas direcciones corresponden a las variables definidas en la directiva cblock 0X20, tal como vemos en la Figura 14.51.

PASO 7 Agregar ventana de memoria de programa

Otra de las ventanas que resulta útil al tiempo de realizar la depuración es la ventana de **memoria de programa** o *program memory*, como vemos en la Figura 14.50; para abrir esta ventana desde el menú de opciones hacemos **clic** en *VIEW/Program Memory*.

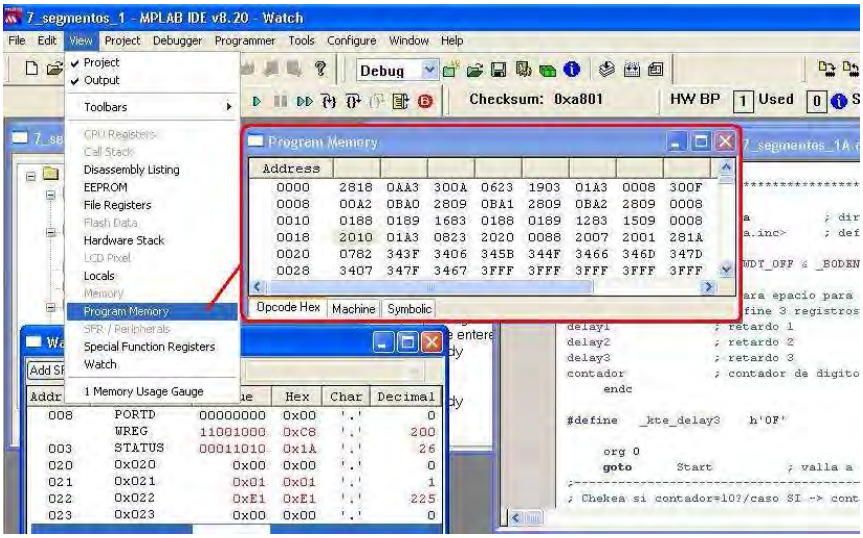


Figura 14.50 Ventana de memoria de programa.

PASO 8 Agregar ventana de listado desensamblado

Otra de las ventanas que resulta útil en el momento de realizar la depuración, es la ventana de **Listado desensamblado** o *Disassembly Listing*, como vemos en la Figura 14.51; para abrir esta ventana desde el menú de opciones hacemos **clic** en *VIEW/Disassembly Listing*.

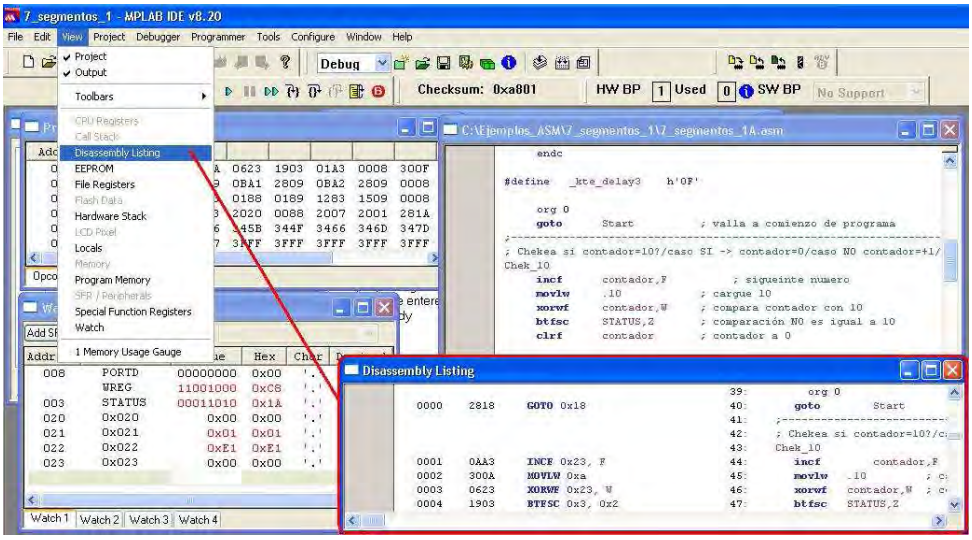


Figura 14.51 Ventana de listado desensamblado.

PASO 9 Generar reset

Ahora, cuando tenemos todas las ventanas necesarias comenzamos la depuración. Al generar el *reset* se obliga al micro ir a su posición inicial (Figura 14.52):

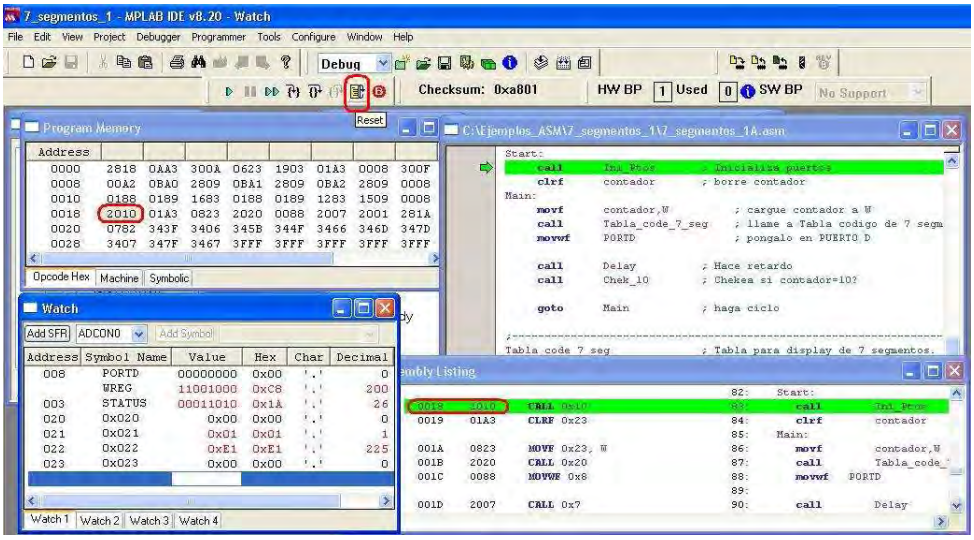


Figura 14.52 Reset.

Observe cómo la primera instrucción está después del *Label Start*, que es donde inicia el programa con la instrucción call Ini_Ptos en la posición **pc = 0X18** cuya instrucción tiene el código **2010h**, que observamos en la ventana de Memoria de Programa.

PASO 10 Primer paso interno

Para entrar a la subrutina **Ini_Ptos** hacemos **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción **call Ini_Ptos** la PC entra a la primera instrucción luego del *Label* **Ini_Ptos** que es la instrucción clrf PORTD en la posición **pc = 0X10** con código **0188h**, como se muestra en la Figura 14.53.

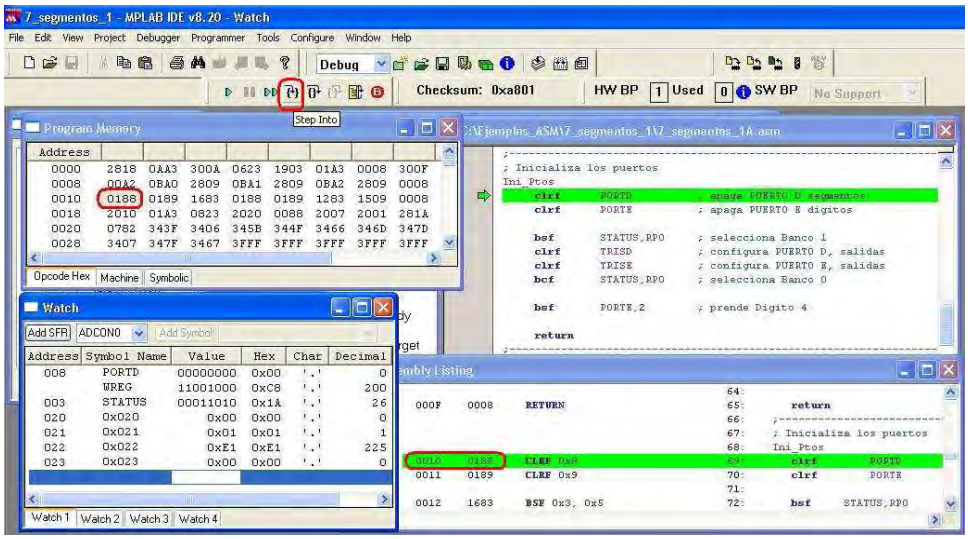


Figura 14.53 Primer paso interno.

PASO 11 Segundo paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción clrf PORTD el PC queda en la segunda instrucción de la subrutina **Ini_Ptos** que es la instrucción clrf PORTE en la posición **pc = 0X11** con código **0189h**, como se muestra en la Figura 14.54.

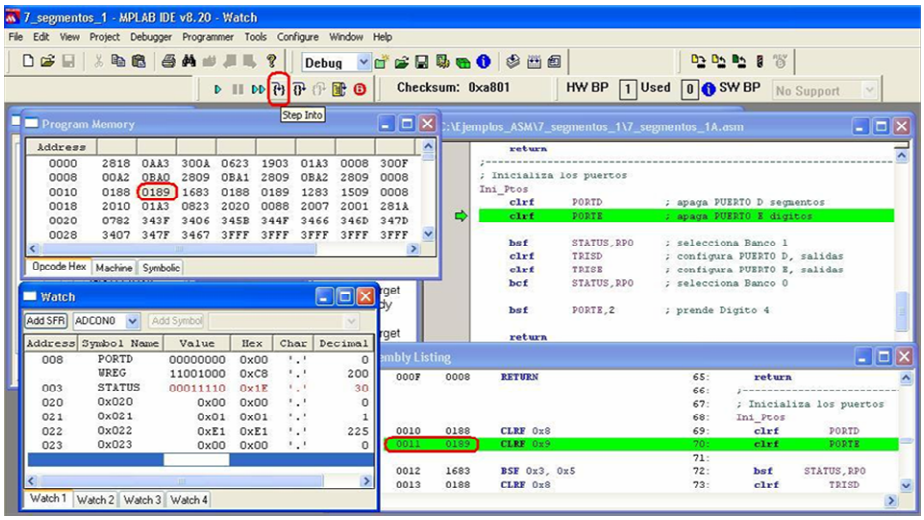


Figura 14.54 Segundo paso interno.

PASO 12 Función Run To Cursor

Podemos recorrer de igual forma, con paso a paso, sucesivamente todas las instrucciones de la subrutina **Ini_Ptos** (Figura 14.55), verificando cada instrucción, su código y la posición de la PC; o también podemos ubicar el cursor en la última instrucción **Return** y allí con el botón derecho desplegar el menú y ejecutar la función **Run To Cursor**, como indica la Figura 14.55 y, de esta forma, ejecutar todas las instrucciones anteriores, incluida aquella donde está el cursor, y al ejecutar la instrucción *return* sale de la subrutina.

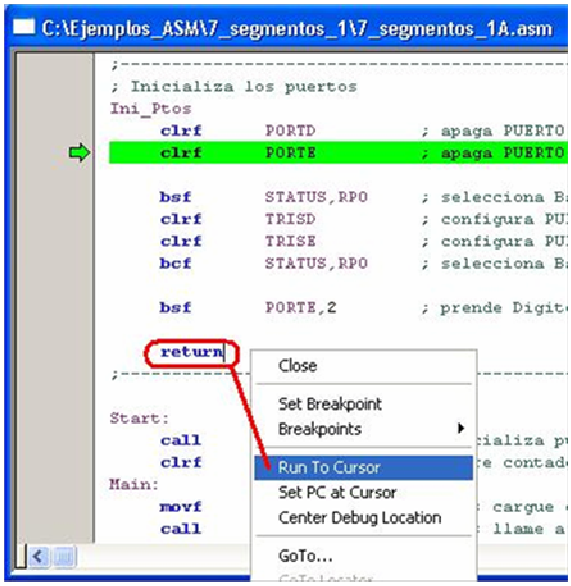


Figura 14.55
Subrutina Ini_Ptos.

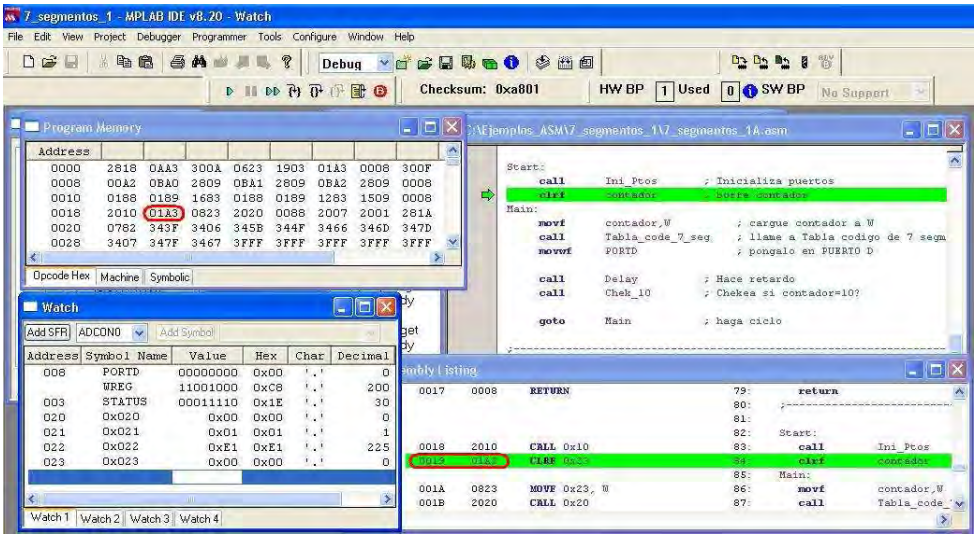


Figura 14.56 Función Run To Cursor.

Luego de la función **Run To Cursor** se observa cómo queda el **PC = 0X19,** o en la segunda instrucción del programa principal en la instrucción **clrf contador**, luego de la llamada a la subrutina **Call Ini_Ptos**, o sea al salir de la subrutina **Ini_Ptos**, como se muestra en la Figura 14.56.

PASO 13 Tercer paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción clrf contador el PC queda en la tercera instrucción del programa principal, que es la instrucción movf contador,W en la posición **PC = 0X1A** con código **0823**, como se muestra en la Figura 14.57.

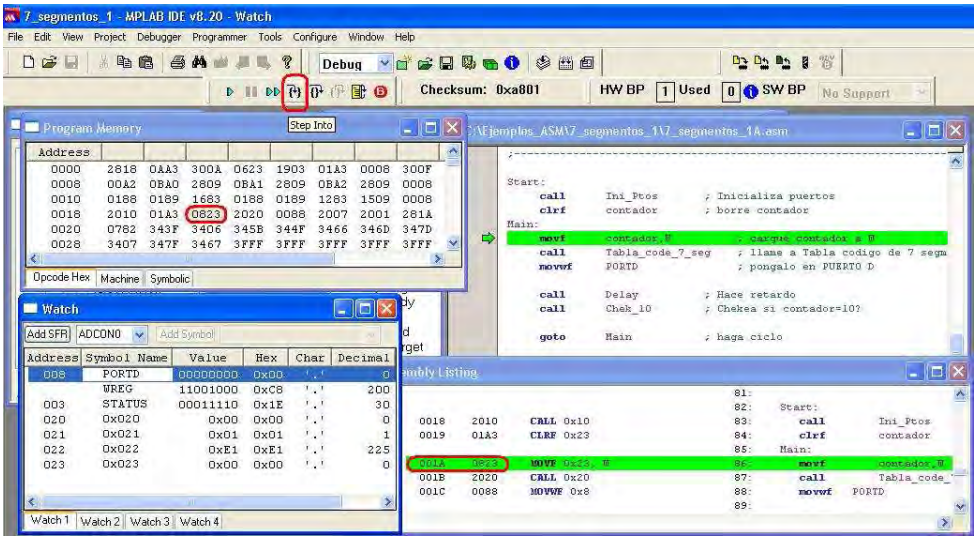


Figura 14.57 Tercer paso interno.

PASO 14 Cuarto paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción movf contador,W se altera el registro **W = 0** y el PC queda en la cuarta instrucción del programa principal que es la instrucción call Tabla_code_7seg en la posición **PC = 0X1B** con código **2020**, como se muestra en la Figura 14.58.

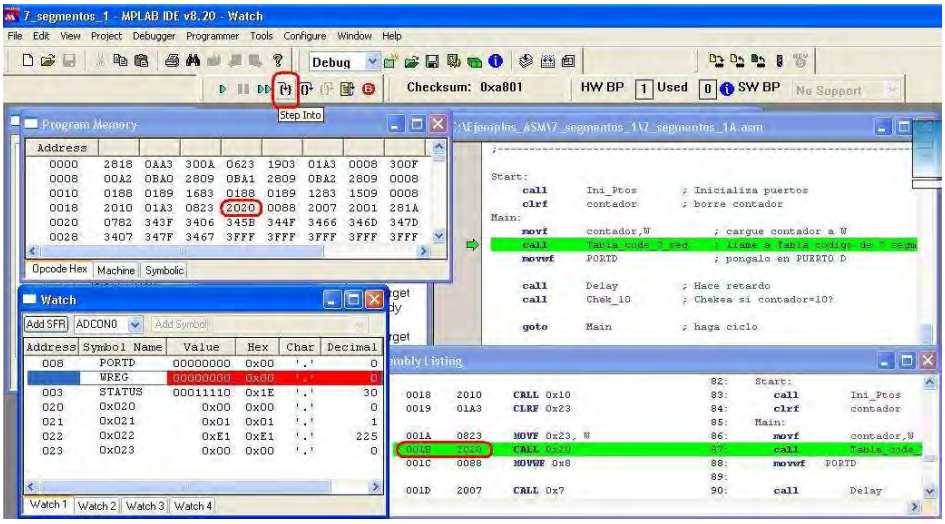


Figura 14.58 Cuarto paso interno.

PASO 15 Quinto paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción **call Tabla_code_7seg** el **PC** queda en la primera instrucción del programa **call Tabla_code_7seg** que es la instrucción **addwf PCL,F** en la posición **PC = 0X20 con código 0782**, como se muestra en la Figura 14.59.

Esta instrucción **addwf PCL,F** suma el contenido de **W + PC** y lo guarda en **PC**, de tal forma que si **W = 0**, la posición del PC, luego de esta instrucción, es **PC = 0X21**, que es la primera instrucción de la tabla.

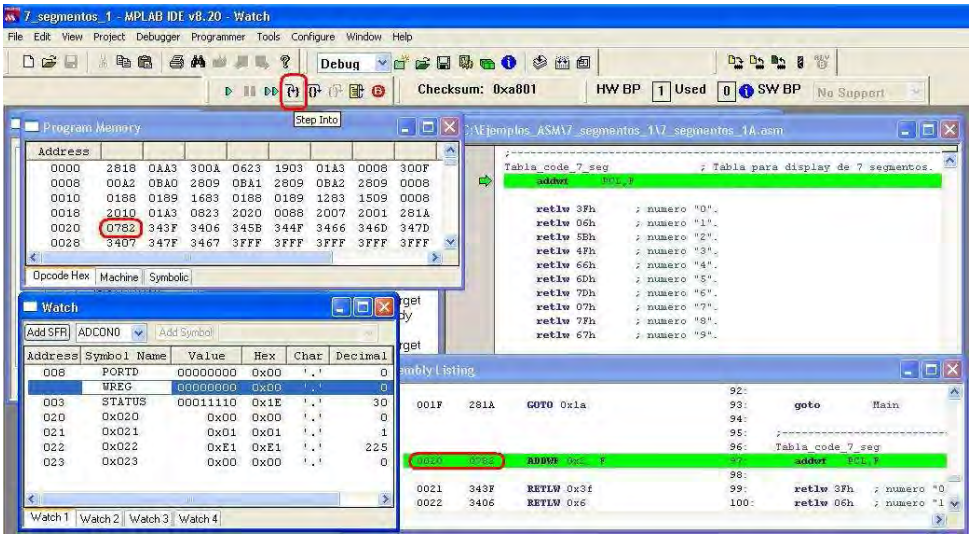


Figura 14.59 Quinto paso interno.

PASO 16 Sexto paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción addwf PCL,F el PC queda en la primera posición de la tabla que es la instrucción retlw 3Fh en la posición **PC = 0X21** con código **343F**, como se muestra en la Figura 14.60.

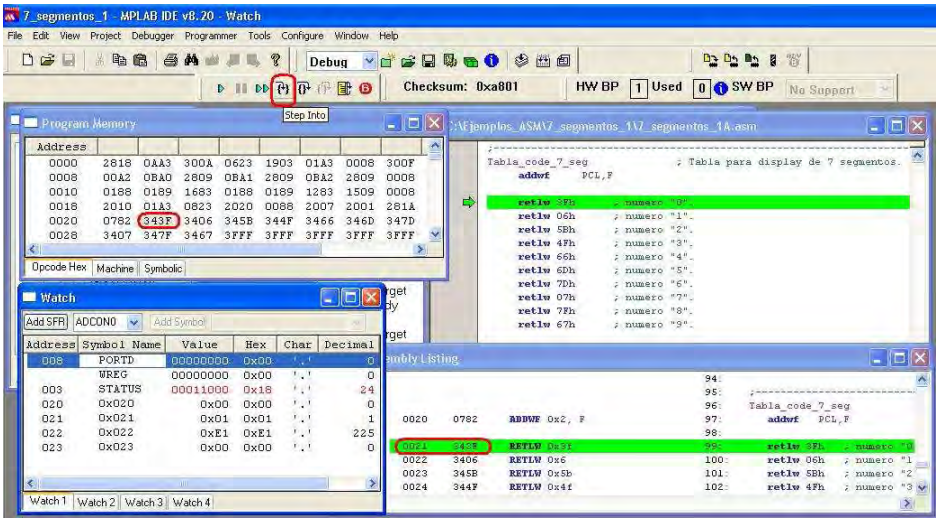


Figura 14.60 Sexto paso interno.

Al quedar en la primera posición de la tabla, el valor que retornará la tabla, Tabla_code_7seg, será el valor **3Fh,** que es el código necesario para que al ponerse en el **puerto D** muestre en el display de 7 segmentos el valor "0".

PASO 17 Séptimo paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo al ejecutar la instrucción retlw 3Fh el PC retorna de la subrutina **Tabla_code_7seg** con un valor en **W = 3F** (como se observa en la ventana *Watch*), en el programa principal en la instrucción movwf PORTD en la posición **PC = 0X1C** con código **0088**, como se muestra en la Figura 14.61.

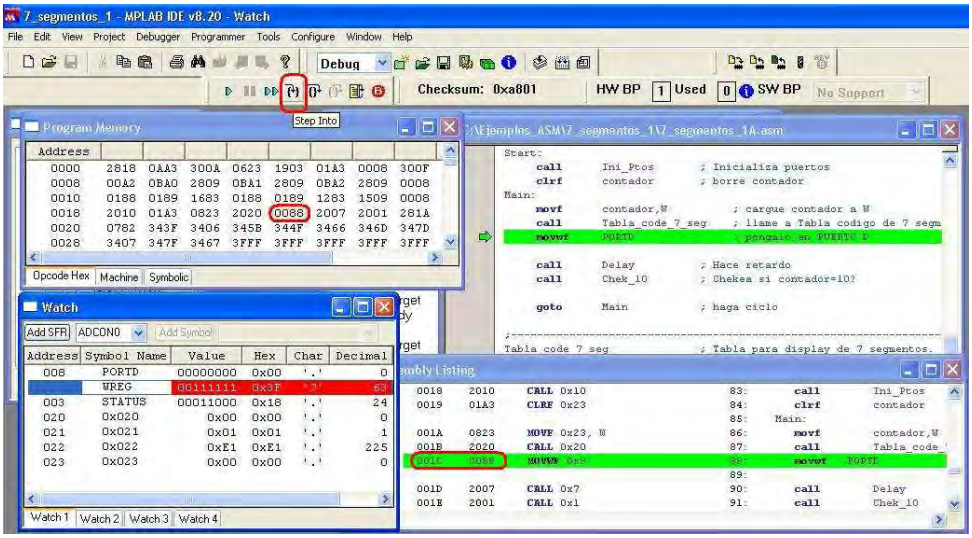


Figura 14.61 Séptimo paso interno.

PASO 18 Octavo paso interno

Seguimos haciendo **clic** en el ícono  *Step Into* **(paso interno)**, y se observa cómo, al ejecutar la instrucción movwf PORTD, el código se puede ver en el display de 7 segmentos como el número **'0'**, el **PC** queda en la instrucción call Delay en la posición **PC = 0X1D** con código **2007**, como se muestra en la Figura 14.62.

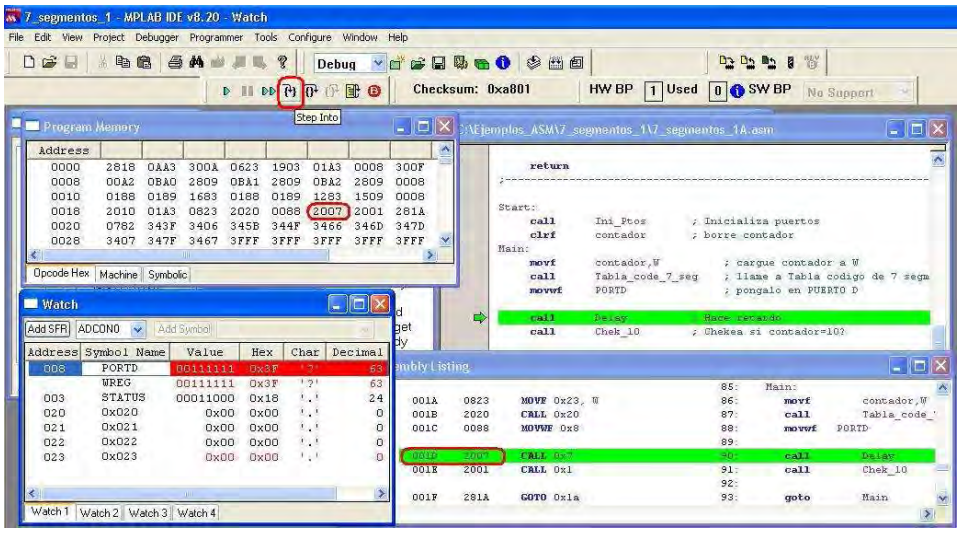


Figura 14.62 Octavo paso interno.

PASO 19 Modo run

A partir de este punto, en el display de 7 segmentos se observan los números del 0 al 9 con un retardo de 0.5 segundos entre ellos; pero el retardo al realizar el paso a paso no es perceptible, solamente si realizamos la función  *run* (**correr**), la cual cambia el micro a modo de *run* (Figura 14.63).



Figura 14.63 Paso 19 Modo *run*.

En la Figura 14.64 se aprecia el kit de trabajo, y en el módulo de display 7 segmentos irán apareciendo los números.

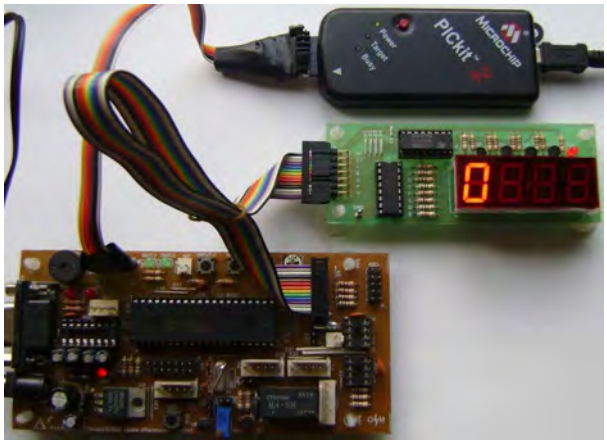


Figura 14.64 Kit con número "0".



Figura 14.65 Números "0" a "9" en 7 segmentos.

En el display de 7 segmentos los dígitos aparecerán como muestra la Figura 14.65.

PASO 20 Detener

En cualquier momento podemos detener el modo de *Run* con el ícono  *Halt*, Detener, para continuar nuevamente con paso a paso.

14.6 ARCHIVOS *.LKR *.INC Y MACROS

14.6.1 ARCHIVOS DE ENLACE LINKER (*.LKR)

Los archivos **Linker** o de enlace (***.lkr**) son los archivos que contienen definiciones sobre las áreas de memoria del microcontrolador que se va a trabajar; definen dónde están ubicados los datos y el código de programa; por lo tanto, habrá un archivo **Linker** asociado para cada microcontrolador. Trabaja asociado con otros módulos del MPLAB para generar los archivos ***.hex**, y con las funciones de **simulación**, **programación** y **depuración**.

UBICACIÓN

Las librerías de los archivos ***.link** se encuentran en la carpeta **C:\Archivos de programa\Microchip\MPASM Suite\LKR**; allí encontrará los archivos asociados con cada microcontrolador. Estos archivos vienen con la instalación del MPLAB. En la Figura 14.66 se muestra cómo se abren los archivos ***.lkr**, estando en la ventana del proyecto **Blink_1.mcw** (para este caso de ejemplo), seleccionando la carpeta **Linker Script**; con el botón derecho en el menú desplegable seleccionamos la opción **Add files...**, y haciendo **clic** en ella aparece la ventana **Add Files to Project**.

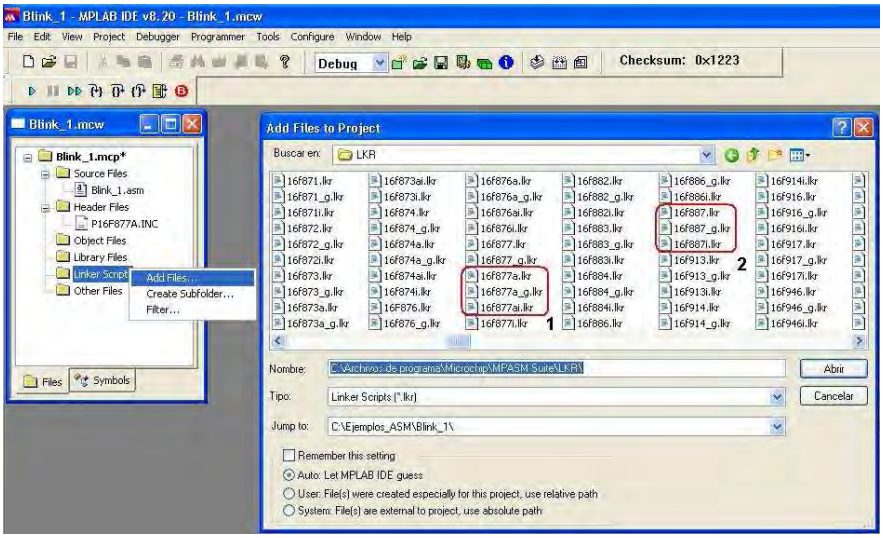


Figura 14.66 Ventana archivos Linker.

En esta ventana, en la casilla **Nombre**, se pone la ruta de los archivos **linker C:\Archivos de programa\Microchip\MPASM Suite\LKR**; en esta dirección podemos encontrar los archivos de los microcontroladores **PIC16F877A** resaltado como (1) y **PIC16F887** resaltado como (2), tal como se muestra en la Figura 14.66.

NOMBRES DE ARCHIVOS LINKER Y SU FUNCIÓN

Para el **PIC16F87A** se encuentran los archivos de la manera siguiente:

- **`16f877a.lkr`** para **programación**
- **`16f877ai.lkr`** para **depuración** o *debugg*
- **`16f877a_g.lkr`** genérico, para ambos **programación** y **depuración** o *debugg*

En forma general se encuentran los archivos con los nombres, así:

- **xxxx.lkr** para **programación**
- **xxxx i.lkr** para **depuración** o *debugg*
- **xxxx _g.lkr** genérico, para ambos **programación** y **depuración** o *debugg*

Donde **xxxx** es el nombre del micro sin las iniciales **PIC**, seguido por las terminaciones resaltadas.

De esta manera, para el microcontrolador **PIC16F887** el nombre del archivo será **16f877** con sus terminaciones; los archivos son:

- **16f887.lkr** para **programación**
- **16f887i.lkr** para **depuración** o *debugg*
- **16f887_g.lkr** genérico, para ambos **programación** y **depuración** o *debugg*

Una vez escogido el archivo **16f877a_g.lkr**, la ventana de proyecto queda como la imagen de la Figura 14.67.

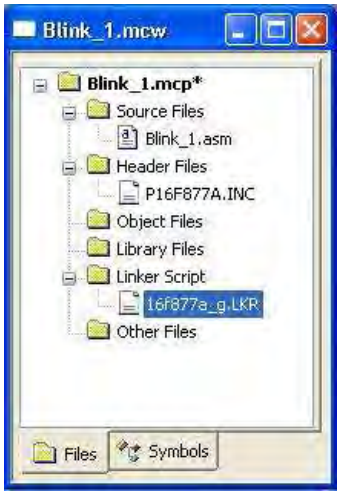


Figura 14.67 Ventana carpeta Linker.

Para desarrollar programas, recomendamos trabajar con el archivo genérico **16f877a_g.lkr**, ya que presta el servicio de trabajar en **programación** y en **depuración**.

RECURSOS DEL MCU PARA DEBUGG

En el modo de **depuración** o *debugg*, el microcontrolador destina ciertos recursos propios para este modo, tales como:

- 2 pines I/O RB7:RB6, para comunicación ICSP.
- 1 nivel de *Stack*, para llamado a rutinas (o interrupciones).
- 256d o 100h posiciones de memoria de programa, donde se ubica un programa residente para el modo de depuración; estas posiciones de memoria de programa son las últimas o de direccionamiento más alto (**1F00h a 1FFFh**).
- 6 posiciones de memoria RAM (**1EBh a 1EFh**); una de ellas es el primer registro en el banco compartido (**070h o 0F0h o 170h o 1F0h**), los otros 5 registros son (**1EBh a 1EFh**).
- En la dirección (**0000h**) debe estar la instrucción **nop**.

```
NOTA: El programa MPLAB realiza esto en forma automática, pero hay
que tener en cuenta estas restricciones en los recursos para no
entrar en conflicto con la aplicación que se vaya a depurar.
```

Estos recursos destinados al modo de **depuración** o *debugg* se muestran en la Tabla 14.3.

Tabla 14.3 Recursos del MCU para depuración.

I/O pins	RB6, RB7
Stack	1 level
Program Memory	Address 0000h must be NOP
	Last 100h words
Data Memory	0x070 (0x0F0, 0x170, 0x1F0) 0x1EB-0x1EF

Estos recursos deben definirse en el archivo ***_g.lkr,** que es el archivo para **depuración** o **programación** del microcontrolador específico. En la sección siguiente veremos cómo se reservan estos recursos en el archivo ***_g.lkr**.

CONTENIDO ARCHIVO 16F877A_G.LKR

A continuación, en la Tabla 14.4 se muestra parte de los contenidos del archivo **16f877a_g.lkr**, resaltado por secciones con definición de áreas de **Datos** y **Código**. Todos los archivos **Linker** se pueden encontrar también en el **Apéndice D: Archivos Linker *.LKR y Definiciones *.INC**.

	PROGRAMACIÒN Y DEPURACIÒN 16f877a_g.lkr	ARCHIVO TIPO
	// Generic linker script for the PIC16F877A processor	// Archivo linker para PIC16F877A
	#IFDEF _DEBUG	En caso de Debug, haga los siguiente
	LIBPATH .	hace uso de librería LIBPATH
PROG	CODEPAGE NAME=page0 START=0x0000 END=0x07FF	define las posiciones de las paginas PROGRAMA
	CODEPAGE NAME=page1 START=0x0800 END=0x0FFF	
	CODEPAGE NAME=page2 START=0x1000 END=0x17FF	
	CODEPAGE NAME=page3 START=0x1800 END=0x1EFF	
	CODEPAGE NAME=debug3 START=0x1F00 END=0x1FFF PROTECTED	**define memoria para debug PROGRAMA**
	CODEPAGE NAME=.idlocs START=0x2000 END=0x2003	define bytes de ID
	CODEPAGE NAME=.config START=0x2007 END=0x2007	define bits de Configuration bits
	CODEPAGE NAME=eedata START=0x2100 END=0x21FF PROTECTED	define posiciones de EEPROM
SFR	DATABANK NAME=sfr0 START=0x0 END=0x1F PROTECTED	define las posiciones de los bancos DATA MCU
	DATABANK NAME=sfr1 START=0x80 END=0x9F PROTECTED	
	DATABANK NAME=sfr2 START=0x100 END=0x10F PROTECTED	
	DATABANK NAME=sfr3 START=0x180 END=0x1BF PROTECTED	
GPR	DATABANK NAME=gpr0 START=0x020 END=0x06F	define las posiciones de los bancos DATA Usuario
	DATABANK NAME=gpr1 START=0x0A0 END=0x0EF	
	DATABANK NAME=gpr2 START=0x110 END=0x16F	
	DATABANK NAME=gpr3 START=0x190 END=0x1E4	
	DATABANK NAME=dbggpr3 START=0x1E5 END=0x1EF PROTECTED	**define memoria para debug**
GPR NO BANK	**SHAREBANK NAME=dbgspr0 START=0x70 END=0x70 PROTECTED**	**define las posiciones de ningun banco DATA Debug**
	SHAREBANK NAME=gprnobnk START=0x71 END=0x7F	define las posiciones de ningun banco DATA Usuario
	SHAREBANK NAME=dbgspr0 START=0xF0 END=0xF0 PROTECTED	
	SHAREBANK NAME=gprnobnk START=0xF1 END=0xFF PROTECTED	
	SHAREBANK NAME=dbgspr0 START=0x170 END=0x170 PROTECTED	
	SHAREBANK NAME=gprnobnk START=0x171 END=0x17F PROTECTED	
	SHAREBANK NAME=dbgspr0 START=0x1F0 END=0x1F0 PROTECTED	
	SHAREBANK NAME=gprnobnk START=0x1F1 END=0x1FF PROTECTED	
PROG DATA	SECTION NAME=PROG1 ROM=page0 // ROM code space - page0	define las areas de memoria de programa para datos
	SECTION NAME=PROG2 ROM=page1 // ROM code space - page1	
	SECTION NAME=PROG3 ROM=page2 // ROM code space - page2	
	SECTION NAME=PROG4 ROM=page3 // ROM code space - page3	
	SECTION NAME=IDLOCS ROM=.idlocs // ID locations	
	SECTION NAME=DEEPROM ROM=eedata // Data EEPROM	

Tabla 14.4 Archivo Linker 16F877A_G.LKR.

En los cuadros **PROG, SFR, GPR, GPR NO BANK, PRG DATA** se observan secciones como áreas de definición. Cada una de estas secciones tiene una función en sus definiciones.

La sección PROG

Esta sección define las áreas de memoria de programa y los registros de configuración. Se define cada página (*page*) y sus direcciones de inicio y fin.

En la línea de definición:

CODEPAGE NAME = page 2 START = 0X1000 END = 0X17FF

Se define la página 2, *page 2,* que tiene dirección de inicio **START = 0X1000** y dirección final, **END = 0X17FF**.

En la línea de definición:

CODEPAGE NAME = .config START = 0X2007 END = 0X2007 PROTECTED

Se define la posición de los bits de configuración; se encuentra en la dirección de inicio, **START = 0X2007h**, y como dirección final, **END = 0X2007h**.

La sección GPR

La sección de datos **GPR** define las áreas de memoria de datos **RAM** para el usuario; cada bloque de registros se define con su dirección de inicio y dirección final; por ejemplo:

DATABANK NAME = gpr0 START = 0X020 END = 0X06F

El bloque de datos **gpr0** tiene dirección de inicio, **START = 0X020**, y dirección final, **END = 0X06F**.

En la línea de definición:

CODEPAGE NAME = debug3 START = 0X1F00 END = 0X1FFF PROTECTED

Esta línea de definición coincide con los recursos reservados de memoria de programa que se encuentra en modo de depuración o *debug*, como dirección de inicio, **START 0X1F00**, y dirección final, **END = 0X1FFF**; estas **256 (100h)** posiciones de memoria son reservadas para un programa residente que permita el modo de depuración. Observe cómo está en las últimas posiciones de memoria. De la manera anterior podemos observar cómo cada una de las secciones contiene las definiciones de las áreas de uso, según la función.

14.6.2 ARCHIVOS DE DEFINICIONES (*.INC)

Los archivos de definición ***.inc** contienen las definiciones de todos los registros y sus bits que contiene el microcontrolador, de tal manera que ayudan al desarrollador a trabajar con estas definiciones, pues de otra manera tendría que hacer las definiciones de todos los registros que se vayan a usar en el programa. Hay un archivo de definición ***.inc** asociado para cada microcontrolador. Al realizar la compilación del proyecto se toma este archivo con sus definiciones para compilarlo junto al archivo de programa.

UBICACIÓN

Las librerías de los archivos ***.inc** se encuentran en la carpeta **C:\Archivos de programa\Microchip\MPASM Suite**; allí encontrará los archivos asociados con cada microcontrolador. Estos archivos vienen con la instalación del MPLAB. En la Figura 14.68 se muestra cómo se abren los archivos ***.inc**; estando en la ventana del proyecto **Blink_1.mcw** (para este caso de ejemplo), seleccionando la carpeta **Header Files**, con el botón derecho en el menú desplegable escogemos la opción **Add files** dando **clic** en ella aparece la ventana **Add Files to Project**.

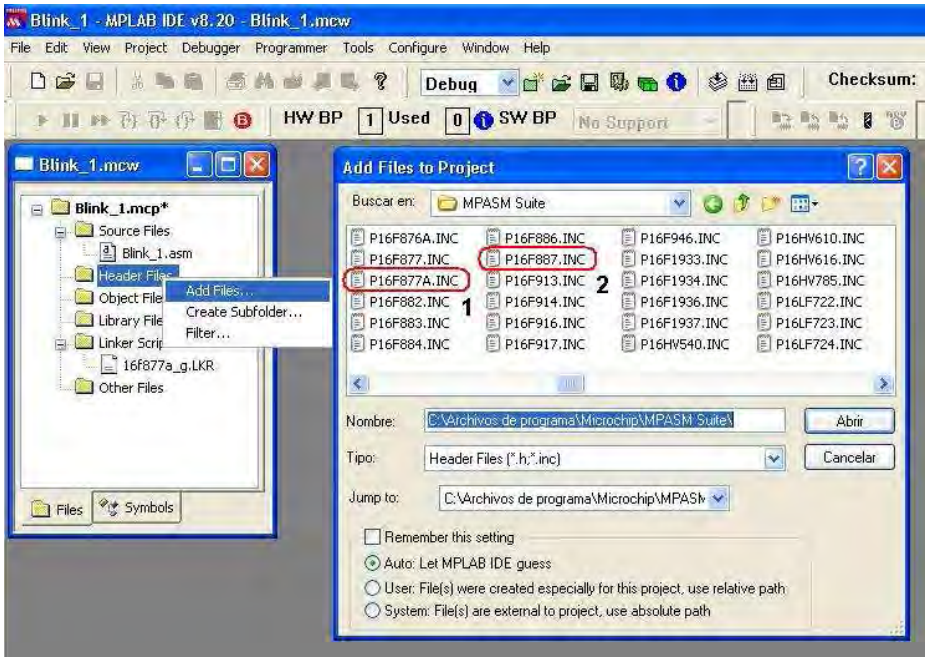


Figura 14.68 Ventana de archivos *Linker*.

En esta ventana, en la casilla **Nombre**, se pone la ruta de los archivos de definición **C:\Archivos de programa\Microchip\MPASM Suite**, en esta dirección podemos encontrar los archivos de los microcontroladores **PIC16F877A** resaltado como (1) y **PIC16F887** resaltado como (2), tal como se muestra en la Figura 14.68. Una vez escogido el archivo **PIC16F877A.INC,** la ventana de proyecto queda como la imagen de la Figura 14.69.

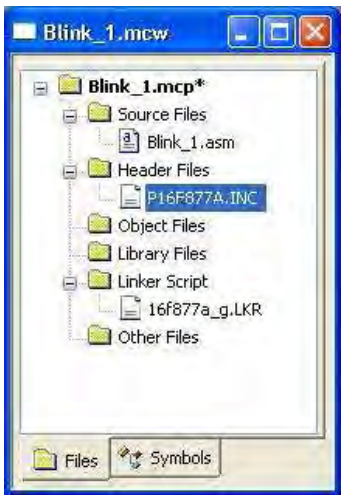


Figura 14.69 Ventana carpeta *Header Files*.

CONTENIDO ARCHIVO "PIC16F877A.INC"

A continuación, en la Figura 14.70, se muestra parte del contenido del archivo **PIC16F877A.INC.**

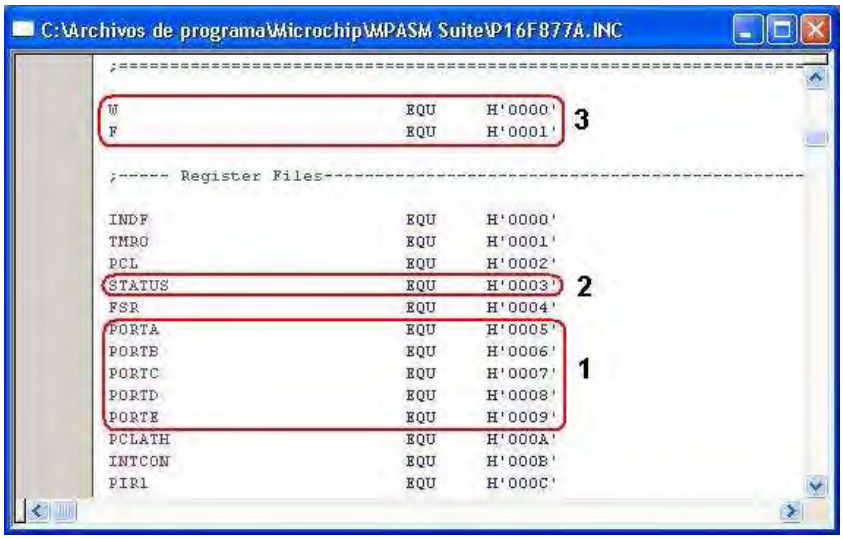


Figura 14.70 Archivo *Linker* 16F877A_G.LKR.

Observe las definiciones de las direcciones de los **puertos A, B, C, D y E** en el ítem resaltado (**1**), la definición de la dirección del registro **STATUS** en el ítem resaltado (**2**), y en el ítem resaltado (**3**) se encuentra la definición de los valores de los registros destino **W** y **F**.

ARCHIVOS *.INC USUARIO

El usuario o el desarrollador también puede construir archivos ***.inc** que contienen definiciones propias de su aplicación. Estas definiciones pueden ayudar a simplificar el desarrollo de la aplicación, evitando tener que redefinir variables, constantes y subrutinas, cada vez que se realiza una aplicación nueva. Un ejemplo práctico de esto es la construcción de un archivo de retardos que diferentes programas pueden usar. A continuación veremos el código y su forma de empleo en un programa. El archivo se llama **Delay.inc** que se encuentra en la ruta **C:\Ejemplos_ASM\Blink_1\Delay.inc**.

```
;*******************************************************************************
;** NOTAS IMPORTANTES DE SOFTWARE
**
;**= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = **
;**
**
;** - Rutinas de retardo de tiempo incluidas aquí **
;** - Delay_1s          ;retardo de 1 seg **
;** - Delay_50 ms       ;retardo de 5 ms
**
;** - Delay_2 ms        ;retardo de 2 ms
**
;** - Delay_150 µs      ;retardo de 150 µs **
;**
**
;** NOTA: Estos valores están calculados para Xtal = 20MHz, Tcy = 0.2 µs **
;*******************************************************************************
        cblock                      ;separa espacio para registros
;define 3 registros para retardo
delay1     ;retardo 1
delay2     ;retardo 2
delay3     ;retardo 3
endc

;ktes para retardo
#define_delay_150 µs                .250          ;retardo para 150 µs
#define_delay_2 ms .13              ; retardo para 2 ms
#define_delay_50 ms                 .2            ;retardo para 50 ms
#define_delay_1s                    .25           ;retardo para 1s
;-------------------------------------------------------------------------
;Retardo 1s
;Delay3 = 250*3*0.2 µs = 150 µs
;Delay2 = 256*150 µs = 38.400 µs s = 40 ms
;Delay1 = 25*40 ms = 1000 ms = 1s
Delay_1s
          clrf delay1; borre Delay 1
          clrf delay2; borre Delay 1
          movlw delay_1s ; cargue retardo de 1s
          movwf delay3 ; a delay3
```

```
Loop4
          decfsz    delay1,F
          goto      Loop4
          decfsz    delay2,F
          goto      Loop4
          decfsz    delay3,F
          goto      Loop4    ;haga ciclo
          return
;-------------------------------------------------------------------------
; Retardo 50 ms
; Delay3 = 256*3*0.2 µs = 153.6 µs
; Delay2 = 256*153.6 µs = 39.321.6 µs = 40 ms
; Delay1 = 2*40 ms = 80 ms
Delay_50 ms
          clrf      delay1          ;borre Delay 1
          clrf      delay2          ;borre Delay 1
          movlw     _delay_50 ms    ;cargue retardo de 50 ms
          movwf     delay3          ;a delay3
Loop3
          decfsz    delay1,F
          goto      Loop3
          decfsz    delay2,F
          goto      Loop3
          decfsz    delay3,F
          goto      Loop3    ;haga ciclo
          return
;-------------------------------------------------------------------------
;Retardo 2 ms
;Delay2 = 256*3*0.2 µs = 153.6 µs
;Delay1 = 13*153.6 µs = 1.996.8 µs = 2 ms
Delay_2 ms
clrf      delay1            ;borre Delay 1
movlw     _delay_2 ms       ;cargue retardo de 2 ms
movwf     delay2            ;a delay2
Loop2
decfsz    delay1,F
goto      Loop2
decfsz    delay2,F
goto      Loop2             ;haga ciclo
return
;-------------------------------------------------------------------------
;Retardo 150 µs
;Delay1 = 256*3*0.2 µs = 153.6 µs
Delay_150 µs
          movlw     delay_150 µs    ;cargue retardo de 150 µs
          movwf     delay1   ;a delay1
Loop1
         decfsz     delay1,F        ;decremente retardo 1
         goto       Loop1           ;haga ciclo
         return
;-------------------------------------------------------------------------------
```

VARIABLES

Observemos cómo se definen algunas variables, tales como:

```
          cblock        ; separa espacio para registros
; define 3 registros para retardo
delay1 ; retardo 1
delay2 ; retardo 2
delay3 ; retardo 3
          endc
```

Se debe anotar que la directiva **cblock** no tiene dirección, y eso implica que estas variables se pondrán en las posiciones de memoria RAM, a continuación de la última variable definida en el programa principal que usa estas definiciones; más adelante veremos cómo se usan estas definiciones, desde otros programas. Estas variables son usadas por las rutinas definidas en este archivo **DELAY.INC**, y también puede usarlas el programa principal que invoca estas definiciones; sin embargo, hay que tener precaución al usar estas variables, debido a que podrían alterar el valor del retardo. Lo más recomendable es no usar las variables, para evitar estos conflictos.

CONSTANTES

También podemos ver las definiciones de constantes:

```
; ktes para retardo
#define     _delay_150 µs.250       ; retardo para 150 µs
#define     _delay_2 ms   .13       ; retardo para 2 ms
#define     _delay_50 ms .2         ; retardo para 50 ms
#define     _delay_1s     .25       ; retardo para 1s
```

Estas constantes se usan para las rutinas definidas en el archivo **DELAY.INC**, y también puede usarlas el programa principal que invoca estas definiciones; como las constantes no gastan posiciones de memoria, se pueden usar sin precauciones, ya que no generan conflictos, como sí ocurre en el caso de las variables.

RUTINAS

Las rutinas definidas aquí son:

```
;**     - Delay_1s          ; retardo de 1 seg      **
;**     - Delay_50 ms       ; retardo de 5 ms       **
;**     - Delay_2 ms        ; retardo de 2 ms       **
;**     - Delay_150 µs      ; retardo de 150 µs     **
```

Cada una de ellas tiene su código correspondiente, y el programa principal que invoca estas definiciones puede usarlas. El llamado a estas subrutinas se hace de la manera usual como se llama a las subrutinas con la instrucción **call** nombre_subrutina; por ejemplo, **call Delay_1s** realiza un retardo de 1 segundo.

```
NOTA: Se puede usar (o llamar) a estas subrutinas, constantes y
variables, desde el programa principal o desde otros archivos de
definición.
```

FORMA DE USO DE ARCHIVOS *.INC

A continuación se da un ejemplo del uso de los archivos ***.inc**, usamos el programa principal *Blink_1B.asm* desde el cual se invoca la rutina *Delay_1s* del archivo de definiciones *Delay.inc.*

Este ejemplo se puede encontrar en nuestra página web ubicada en http://virtual.alfaomega.com.mx siguiendo la ruta **X:\Ejemplos_ASM**, y se debe copiar al **C:**; para encontrarlo como **C:\Ejemplos_ASM \ MACRO_1\ MACRO_1.asm**.

```
list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc> ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF

cblock 0x20   ; separa espacio para registros

endc

org 0
Start:
bsf STATUS,RP0 ; seleccione Banco 1
bcf TRISB,4 ; configure pin RB4, salida, led OUT-1
bcf STATUS,RP0 ; seleccione Banco 0

MainLoop:
bsf PORTB,4 ; prenda led OUT-1
call    Delay_1s                ; retardo de 1 seg
bcf PORTB,4 ; apague led OUT-1
call     Delay_1s               ; retardo de 1 seg
goto MainLoop ; haga ciclo

        #include        Delay.inc                ; incluye definiciones de retardos

end
```

La inclusión del archivo de definiciones *Delay.inc* se hace en la línea:

```
        #include   Delay.inc                ; incluye definiciones de retardos
```

Ésta es la forma de invocar los archivos de definiciones.

14.6.3 MACROS

Ésta es una directiva de MPLAB que se utiliza para resumir la escritura de código semejante, es decir, cuando en el código del programa se usa varias veces la misma

operación con diferentes argumentos, se puede agrupar esta operación en una **macro,** en la cual lo único que varía son los argumentos. Si se usa la **macro** definida en el código de programa, el código de la **macro** se copia en esta parte del programa, cuando se hace la compilación.

DEFINICIÓN DE MACRO

La sintaxis de la directiva **macro** es como se muestra en seguida:

```
label   MACRO [argumento1, ….argumentoN]
        instrucciones (código o cuerpo de la MACRO)

        ENDM
```

Cada una de estas partes significa:

- **label** es el nombre de la **macro**, se pone al inicio de la definición de la **macro** y con esta *Label* se invoca en el programa.
- **[argumento1, …..argumentoN]** se usan en la definición de la **macro** en las instrucciones; cuando la **macro** es invocada se pasan estos valores con los registros usados en el programa.
- **instrucciones** son las porciones de código de programa que se busca reemplazar; éstas usan los argumentos como parte del código.
- **ENDM** es el fin de la **macro**.

Para entender este concepto más claramente daremos un ejemplo: En el ejemplo siguiente se crea la **macro** Check_Mínimo para verificar si el Valor es mayor o igual a Mínimo, únicamente cuando el **Valor < Mínimo**, se guarda **Valor en Mínimo**. A continuación se detalla cada una de las partes:

1. El *Label* de la **macro** es Check_Mínimo o definición de la **macro**.

```
Check_Mínimo        MACRO    Valor, Mínimo
```

2. Los **argumentos** son Valor, Mínimo, que son los valores que se van a trabajar en la **macro**.

3. Las **instrucciones** o el código o el cuerpo de la **macro** son:

```
        movf   Mínimo,W          ; cargue Mínimo -> W
        subwf  Valor,W           ; reste Valor - Mínimo -> W
        movf   Valor,W           ; cargue Valor -> W (truco: tener Valor -> W, para
                                 ; salida no, siguiente pregunta)
        btfss  STATUS,C          ; C = 1? diferencia > = 0?. Sí, salte.
                                 ;(Valor > Mínimo), Mínimo permanece
        movwf  Mínimo            ; No, guarde W (Valor) -> en Mínimo (cambia)
```

- El fin de la **macro**.

```
    ENDM
```

A continuación veremos la forma de uso en el programa principal.

INVOCACIÓN MACRO

La forma de invocar (o llamar) la **macro** es así:

```
Check_Mínimo   lectura, lectura_Mín      ; haga Macro Check_Mínimo
```

Esta línea va en el programa principal, y su función es reemplazar los valores definidos en la **macro** como `Valor`, `Mínimo` por los registros `lectura`, `lectura_Mín`. Por tanto, si lectura `< lectura_Mín`, al ejecutar la macro se guarda lectura en `lectura_Mín`.

Este ejemplo se puede encontrar en nuestra página web, ubicada en http://virtual.alfaomega.com.mx siguiendo la ruta X:\Ejemplos_ASM, y se debe copiar al C:\; para encontrarlo como C: \Ejemplos_ASM \ MACRO_1\ MACRO_1.asm.

PARTE 4

MEMORIA

En esta Parte 4 veremos todos los aspectos relacionados con la **memoria**, los tipos de la memoria, según la **arquitectura** o la **función** y de acuerdo con la **fabricación**. Para conocer estas **memorias** las veremos en su orden: de **programa, EEPROM** y **RAM**. Cada una de ellas con su mapa de memoria, sus direcciones y contenidos. La forma de cómo se accesan y modifican, su funcionamiento y prácticas.

La memoria de **Programa** muestra la ubicación de sus páginas y las direcciones límite de cada una. La memoria **EEPROM** está descrita totalmente como un módulo con sus capítulos de: **diagrama circuital, funcionamiento, registros asociados y práctica.** La memoria **RAM** contiene todo lo relacionado con los registros de la **MCU** o **SFR**, y los registros **GPR** o de usuario; se describen cada una de las siguientes áreas: **definición, ubicación e iniciación.** Además, se describen los tipos de direccionamiento directo e indirecto.

15.0 MEMORIA

Para conocer las **memorias** las veremos en su orden: de p**rograma, EEPROM** y **RAM**. Cada una de ellas con su mapa de memoria, sus direcciones y contenidos. La forma de cómo se accesan y modifican, su funcionamiento y prácticas.

15.1 TIPOS DE MEMORIA

Cuando hablamos de los tipos de memoria hacemos referencia a todos los registros que son modificables o no en el microcontrolador. Hay varias formas de clasificar la memoria en un microcontrolador. Según la **arquitectura** o la **función**, los datos que almacenan en:

- **Memoria de programa** o *Program Memory* o ROM (o *flash*)
  Aquí se almacenan instrucciones o programas o ROM (o *flash*); estos datos no varían durante la ejecución del programa.

- **Memoria de datos** o *Data Memory* o RAM
  Aquí están los datos o variables; estos datos sí varían durante la ejecución del programa.

  - **Registros de funciones especiales** o *Special Function Registers* (**SFR**)
    - Aquí están todos los registros necesarios para el control de operación del micro.
  - **Registros de Propósito General** o *General Purpose Registers* (**GPR**)
    - Aquí está la memoria de usuario.
  - **Datos EEPROM** o *Data EEPROM Memory*
    - Datos de usuario no volátiles.

Según la fabricación, los tipos de memoria ROM son:

- **ROM** o *Read Only Memory*

  Memoria de sólo lectura. Es la más barata, pero solamente es recomendable cuando el código es muy estable y para grandes cantidades de producción. El fabricante Microchip puede entregar los MCU ya programados con este tipo de memoria.

- **OTP** u *One-Time-Programmable*

  Programable una sola vez. Son las siguientes en bajo costo.

- **EPROM** o *Erasable Programmable Read Only Memory*

Memoria borrable y programable de sólo lectura. Son del mismo costo que los OTP y varían únicamente en el precio del empaque.

- **FLASH (EEPROM)** o *Electrically Erasable Programmable Read Only Memory*

  Memoria eléctricamente borrable y programable de sólo lectura. Permiten un tiempo muy corto de borrado y escritura que facilita una eficiencia alta en el desarrollo del código y datos no volátiles. Es de anotar que la memoria de datos EEPROM interna también pertenece a este tipo de memoria.

FLASH es la tecnología de memorias utilizada actualmente en la mayoría de los micros, por su facilidad de implementación y alta durabilidad de los datos y un tiempo muy corto de borrado y escritura; pertenece al tipo EEPROM y se utiliza para guardar las instrucciones o los programas. Todos estos tipos de memoria según su función se describirán a continuación.

15.1.1 MAPA DE MEMORIA DE PROGRAMA

La **memoria de programa** almacena las instrucciones y otros datos, como se muestra en el mapa de memoria de la Figura 15.1, observamos cada uno de los bloques que tienen dos grande grupos:

- **Memoria de usuario** (0000h a 1FFFh)
- **Memoria de configuración** (2000h a 3FFFh)

Los datos encontrados en esta sección no son bytes = 8 bits, son *word* = 14 bits (palabras o instrucciones). Hay algunas salvedades que veremos más adelante.

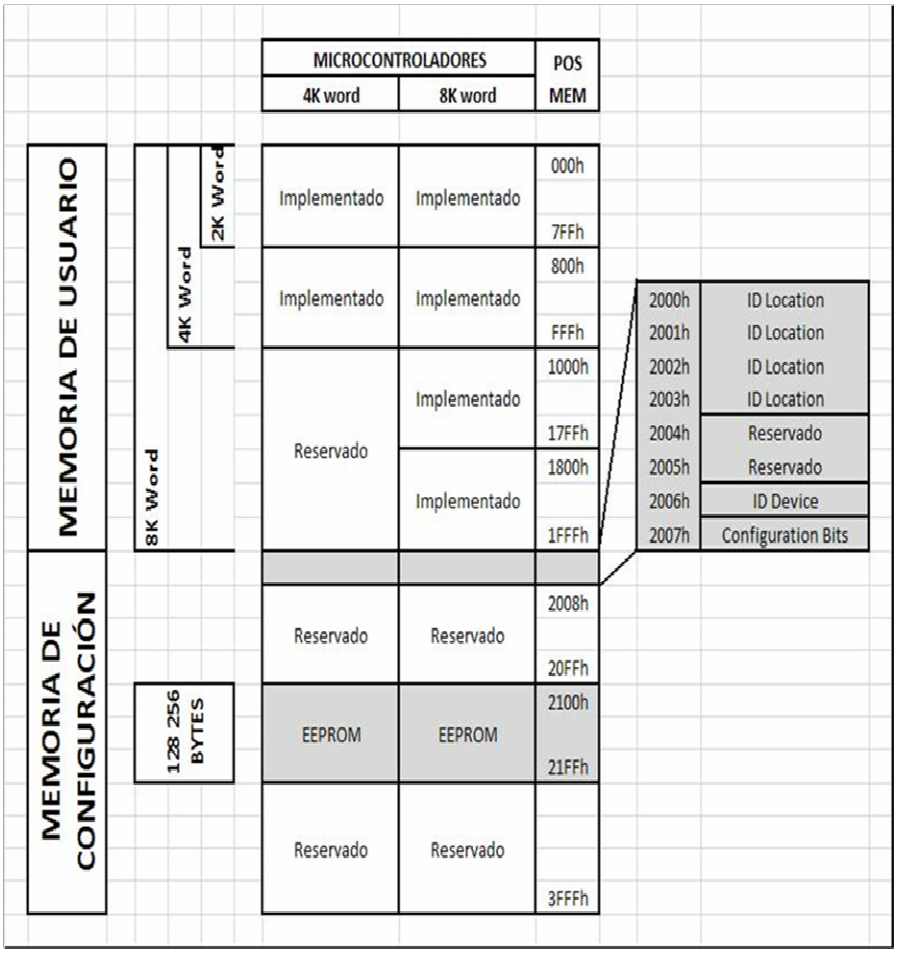


Figura 15.1 Mapa de memoria de programa.

MEMORIA DE USUARIO

Es la primera mitad, se encuentra entre las posiciones **0000h** a **1FFFh** de 4-8 K words (palabras o instrucciones), y contiene las instrucciones o los programas. En **modo de programación**, la PC incrementa desde **0000h** a **1FFFh** y da la vuelta a **0000h**.

MEMORIA DE CONFIGURACIÓN

Está en la segunda mitad; se encuentra entre las posiciones **2000h** a **3FFFh** y contiene los registros de configuración; también se encuentra aquí la memoria EEPROM. En **modo de programación,** la **PC** incrementa desde **2000h** a **3FFFh** y da la vuelta a **2000h** no a **0000h**. En esta sección encontramos: registros de configuración y la memoria EEPROM mapeada lógicamente, la cual se accede mediante un procedimiento (explicado más adelante), no en forma directa o indirecta.

15.1.2 MEMORIA DE USUARIO O PROGRAMA

La **memoria de programa** se encuentra entre las posiciones **0000h** a **1FFFh** de 4-8 K *word* (palabras o instrucciones). Debido al gran contenido de esta sección y al orden de presentación de los temas, dedicamos un capítulo aparte, el **Capítulo 16.0: Memoria de programa**.

15.1.3 MEMORIA DE CONFIGURACIÓN

Esta memoria se encuentra entre las posiciones **2000h** a **3FFFh;** contiene:

- Registros de configuración (**2000h** a **2007h**)
- Memoria EEPROM (mapeada lógicamente en **2100h** a **21FFh**)

En seguida veremos con detalle estas posiciones. El área de memoria no mencionada o faltante, entre **2000h** a **3 FFFh**, son espacios reservados por el fabricante **Microchip** (no usados en este microcontrolador).

REGISTROS DE CONFIGURACIÓN (SPECIAL LOCATION)

Las posiciones **2000h** a **200Fh** están implementadas físicamente, pero solamente se pueden acceder de **2000h** a **2007h**, sólo en **modo de programación**; las otras están reservadas. Cuando el bit más alto de la PC está en '1' entra a memoria de configuración; a este bit no tiene acceso el usuario; únicamente lo utilizan internamente el ambiente MPLAB y la herramienta **PICkit 2** (en nuestro caso). Las posiciones de memoria siguientes corresponden a:

- **ID Location** 2000h a 2003h
- **Reservadas** 2004h a 2005h
- **Device ID** 2006h
- **Configuration Bits** 2007h

ID Location

Éste es un código de identificación para el usuario o para su programa. Son cuatro (4) posiciones de memoria que pueden guardar un ID del usuario; se recomienda utilizar solamente los cuatro (4) últimos bits donde la palabra completa sería así: **xx xxxx xxxx XXXX** (14 bits), donde las X mayúsculas son los bits que se utilizarán. Hay dos opciones; puede escoger un código propio o puede generarlo con base en el *Checksum*. En el entorno MPLAB, desde el menú de opciones **Configure/ID Memory...**, y allí se despliega la ventana de **User ID Memory,** en donde se puede escribir un código propio de 4 bytes, o si marca la casilla **Use Unprotect Checksum**, la próxima vez que abra la ventana de **ID Memory**, aparece un código de 4 bytes correspondiente al *checksum* de los datos del programa como se ilustra en la Figura 15.2.

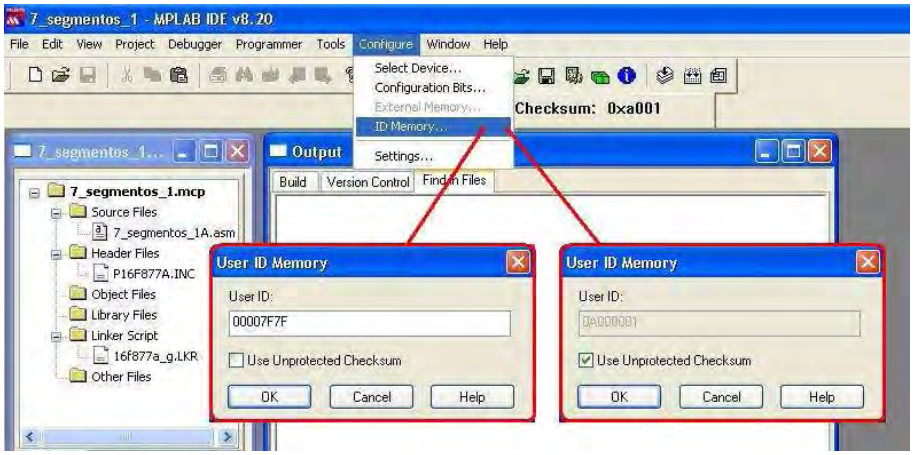


Figura 15.2 ID de usuario.

Device ID

El **Device ID** es un identificador del micro; se puede leer la versión del micro desde el **PICkit 2**. Desde el entorno MPLAB, cuando nos conectamos al micro con una herramienta **PICkit 2** (para nuestro caso), podemos apreciar la versión del micro (Tabla 15.1 y Figura 15.3).

Tabla 15.1 Device ID value.

Device	Device ID Value	
	Dev	Rev
PIC16F873A	00 1110 0100	XXXX
PIC16F874A	00 1110 0110	XXXX
PIC16F876A	00 1110 0000	XXXX
PIC16F877A	00 1110 0010	XXXX

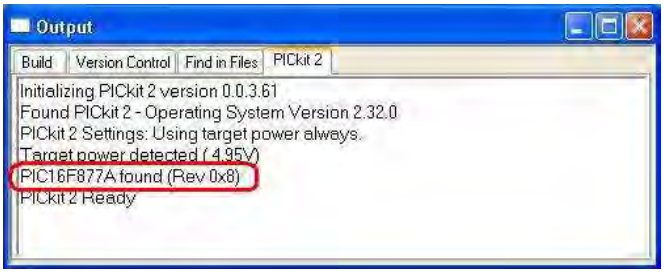


Figura 15.3 Device ID.

Configuration Bits

Los **bits de configuración** son los que definimos en el programa al principio con la directiva __CONFIG; por ejemplo en el código:

```
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC &
_WRT_OFF & _LVP_OFF & _CPD_OFF
```

Tal como se muestra en la Figura 15.4, éste es el punto en donde se seleccionan los bits de configuración en el código del programa.

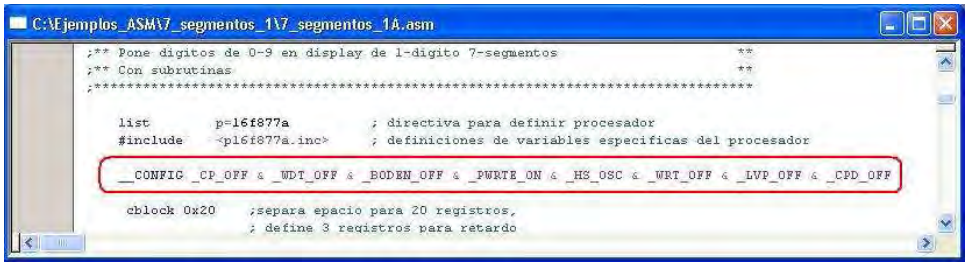


Figura 15.4 Bits de configuración en programa.

Con esta directiva podemos encontrar varios tipos de selección de cada uno de los bits, como se ilustra en la Tabla 15.2.

Tabla 15.2 Opciones de directiva __CONFIGURA

__CONFIG Directive Symbols (From Microchip Header Files)

Feature	SYMBOLS
Oscillators	_RC_OSC
	_EXTRC_OSC
	_EXTRC_OSC_CLKOUT
	_EXTRC_OSC_NOCLKOUT
	_INTRC_OSC
	_INTRC_OSC_CLKOUT
	_INTRC_OSC_NOCLKOUT
	_LP_OSC
	_XT_OSC
	_HS_OSC
Watch Dog Timer	_WDT_ON
	_WDT_OFF
Power-up Timer	_PWRTE_ON
	_PWRTE_OFF
Brown-out Reset	_BODEN_ON
	_BODEN_OFF
Master Clear Enable	_MCLRE_ON
	_MCLRE_OFF
Code Protect	_CP_ALL
	_CP_ON
	_CP_75
	_CP_50
	_CP_OFF
Code Protect Data EEPROM	_DP_ON
	_DP_OFF
Code Protect Calibration Space	_CPC_ON
	_CPC_OFF

Note 1: Not all configuration bit symbols may be available on any one device. Please refer to the MIcrochip include file of that device for available symbols.

Conviene anotar que no todas las opciones de la directiva **_CONFIG** aplican para todos los microcontroladores, estas opciones se pueden verificar en cada **hoja de datos** (*data sheet*). A continuación, en la Tabla 15.3, se detalla la definición de todos los bits del registro de configuración de bits; a manera de ejemplo, se implementa una configuración de bits. En la parte superior apreciamos los bits seleccionados, **10 1111 0000 0010** que corresponden a **2F02h**, el código hexadecimal que es la línea a continuación. Como siguientes están los valores que asigna el microcontrolador **por defecto** (*default*). Se puede apreciar cómo el bit CP tiene asignado **R/P**-1, lo que indica que este bit es **R** de lectura, **P** de programación, **-1** es el estado **por defecto** (*default*). Igual para todos los demás bits. **U-0** indica **U** que es no implementado, y **-0**, su valor por defecto.

Tabla 15.3 Definición de bits de configuración.

EJEMPLO CONFIGURATION WORD 2F0 2H															
1	0	1	1	1	1	0	0	0	0	0	0	1	0		
2		F				0				2					

CONFIGURATION WORD (ADDRESS 2007h)(1)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	PWRTEN	WDTEN	FOSC1	FOSC0

BIT 13	CP	Flash Program Memory Code Protection bit			
		1 – Code protection off	_CP_OFF	H'3FFF'	
		0 – All program memory code-protected	_CP_ALL	H'1FFF'	
BIT 12	—	Unimplemented: Read as '1'			
BIT 11	DEBUG	In-Circuit Debugger Mode bit			
		1 – In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins	_DEBUG_OFF	H'3FFF'	
		0 – In-Circuit Debugger enabled, RB6 and RB7 are dedicated to the debugger	_DEBUG_ON	H'37FF'	
BIT 10	WRT1	Flash Program Memory Write Enable bits			
BIT 9	WRT0	For PIC16F876A/877A:			
		11 – Write protection off; all program memory may be written to by EECON control	_WRT_OFF	H'3FFF'	No prog memmory write protection
		10 – 0000h to 00FFh write-protected; 0100h to 1FFFh may be written to by EECON control	_WRT_256	H'3DFF'	First 256 prog memmory write protected
		01 – 0000h to 07FFh write-protected; 0800h to 1FFFh may be written to by EECON control	_WRT_1FOURTH	H'39FF'	First quarter prog memmory write protected
		00 – 0000h to 0FFFh write-protected; 1000h to 1FFFh may be written to by EECON control	_WRT_HALF	H'29FF'	First half memmory write protected
		For PIC16F872A/874A:			
		11 – Write protection off; all program memory may be written to by EECON control	_WRT_OFF	H'3FFF'	No prog memmory write protection
		10 – 0000h to 00FFh write-protected; 0100h to 0FFFh may be written to by EECON control	_WRT_256	H'3DFF'	First 256 prog memmory write protected
		01 – 0000h to 03FFh write-protected; 0400h to 0FFFh may be written to by EECON control	_WRT_1FOURTH	H'39FF'	First quarter prog memmory write protected
		00 – 0000h to 07FFh write-protected; 0800h to 0FFFh may be written to by EECON control	_WRT_HALF	H'29FF'	First half memmory write protected
BIT 8	CPD	Data EEPROM Memory Code Protection bit			
		1 – Data EEPROM code protection off	_CPD_OFF	H'3FFF'	
		0 – Data EEPROM code-protected	_CPD_ON	H'3EFF'	
BIT 7	LVP	Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit			
		1 – RB3/PGM pin has PGM function; low-voltage programming enabled	_LVP_ON	H'3FFF'	
		0 – RB3 is digital I/O, HV on MCLR must be used for programming	_LVP_OFF	H'3F7F'	
BIT 6	BOREN	Brown-out Reset Enable bit			
		1 – BOR enabled	_BODEN_ON	H'3FFF'	
		0 – BOR disabled	_BODEN_OFF	H'3FBF'	
BIT 5	—	bit 5-4 Unimplemented: Read as '1'			
BIT 4	—				
BIT 3	PWRTEN	Power-up Timer Enable bit			
		1 – PWRT disabled	_PWRTE_OFF	H'3FFF'	
		0 – PWRT enabled	_PWRTE_ON	H'3FF7'	
BIT 2	WDTEN	Watchdog Timer Enable bit			
		1 – WDT enabled	_WDT_ON	H'3FFF'	
		0 – WDT disabled	_WDT_OFF	H'3FFB'	
BIT 1	FOSC1	Oscillator Selection bits			
BIT 0	FOSC0	11 – RC oscillator(3K<R<100K, C>20pF)	_RC_OSC	H'3FFF'	
		10 – HS oscillator (>4Mhz y <20Mhz)	_HS_OSC	H'3FFE'	
		01 – XT oscillator (>200Khz y <4Mhz)	_XT_OSC	H'3FFD'	
		00 – LP oscillator (>32Khz y <200Khz)	_LP_OSC	H'3FFC'	

Para visualizar el registro de configuración de bit desde el menú de opciones en el MPLAB, seleccione **Configure/Configurations Bits...,** y luego se despliega la ventana de **Configuration Bits**; allí se observa cada uno de los bits con su estado de selección seleccionados y el valor respectivo.

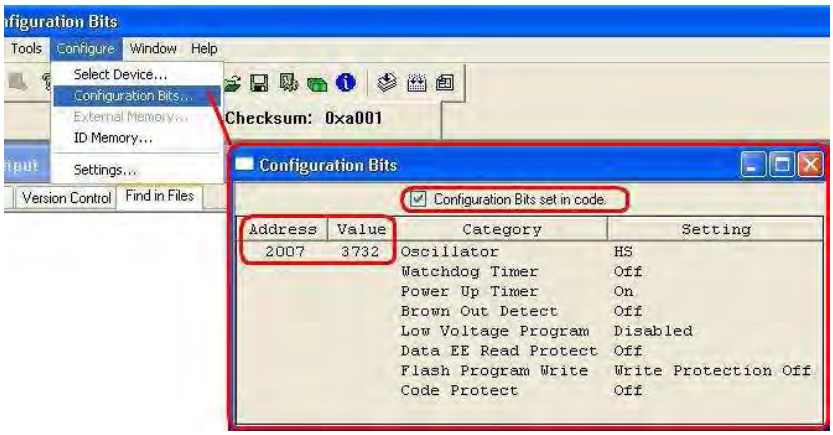


Figura 15.5 Bits de configuración.

Observe que en la Figura 15.5 la casilla **Configuration Bits Set in Code** nos indica que el valor **3732h** corresponde a los bits seleccionados; la dirección de los bits de configuración es la de **2007h**. También podemos editar estos bits deseleccionando la casilla **Configuration Bits Set in Code**, y de esta manera podemos cambiar el valor de cualquiera de los bits, que al ser seleccionado se despliega una ventana donde se encuentran el valor actual y sus opciones; allí podremos cambiarlo, como muestra la Figura 15.6.

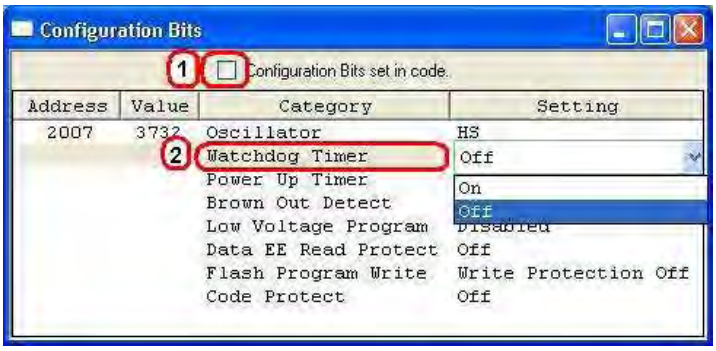


Figura 15.6 Edición Bits de configuración.

MEMORIA EEPROM

Es la misma memoria de datos no volátil; se encuentra entre las posiciones mapeadas lógicamente entre **2100h** a **21FFh** de **128-256 bytes** (datos). No intente acceder esta memoria de otra forma que no sea el procedimiento descrito para leer

o escribir en la memoria EEPROM. Debido al gran contenido de esta sección y al orden de presentación de los temas, le dedicamos por separado todo el **Capítulo 16.0: Memoria EEPROM**.

EVALUACIÓN

```
P.: 1. Cuáles tipos de memoria existen en el MCU 16F877A y 16F887?
R.:
Memoria de programa o Program Memory o ROM (o flash). Aquí se
almacenan instrucciones o programas o ROM (o flash); estos datos no
varían durante la ejecución del programa.
Memoria de datos o Data Memory o RAM. Aquí están los datos o
variables; estos datos sí varían durante la ejecución del programa.

P.: 2. Describa los tipos de memoria RAM que existen en el MCU
16F877A y 16F887.
R.:
Registros de funciones especiales o Special Function Registers
(SFR). Aquí están todos los registros necesarios para el control de
operación del micro.
Registros de Propósito General o General Purpose Registers (GPR).
Aquí está la memoria de usuario.
Datos EEPROM o Data EEPROM Memory, o datos de usuario no volátiles.

P.: 3. Según la fabricación, ¿qué tipos de memoria existen en el
MCU 16F877A y 16F887?
R.:
ROM o Read Only Memory.
Memoria de sólo lectura. Es la más barata, pero solamente es
recomendable cuando el código es muy estable y para grandes
cantidades de producción. El fabricante Microchip puede entregar
los MCU ya programados con este tipo de memoria.
OTP u One-Time-Programmable.
Programable una sola vez son las siguientes en bajo costo:
EPROM o Erasable Programmable Read Only Memory.
Memoria borrable y programable de sólo lectura. Son del mismo costo
que los OTP y varían únicamente en el precio del empaque.
Flash (EEPROM) o Electrically Erasable Programmable Read Only
Memory.
Memoria eléctricamente borrable y programable de sólo lectura.
Permiten un tiempo muy corto de borrado y escritura que facilita
una eficiencia alta en el desarrollo del código y datos no
volátiles. Es de anotar que la memoria de datos EEPROM interna
también pertenece a este tipo de memoria.
```

```
P.: 4. Nombre las 2 áreas de memoria de programa y sus posiciones
que existen en el MCU 16F877A y 16F887.
R.:
Memoria de usuario (0000h a 1FFFh)
Memoria de configuración (2000h a 3FFFh)

P.: 5. ¿Cuáles son los REGISTROS DE CONFIGURACIÓN (SPECIAL
LOCATION) que existen en el MCU 16F877A y 16F887?
R.:
ID Location         2000h a 2003h
Reservadas          2004h a 2005h
Device ID           2006h
Configuration Bits  2007h

P.: 6. ¿Con cuál directiva se habilitan los bits de Configuration?
R.:
Directiva _CONFIG
```

16.0 MEMORIA DE PROGRAMA

La **memoria de programa** o **de usuario** es lo mismo; el programa realizado por el usuario y quemado en el microcontrolador se aloja en estas posiciones, o sea, en las posiciones **0000h a 1FFFh** cuya extensión es de **4-8K**; esto quiere decir que se puede realizar un programa del tamaño de **4096–8192** instrucciones. Cada instrucción tiene un tamaño de 14 bits; estas instrucciones son las que contiene el **set de instrucciones** de cada microcontrolador. Cada instrucción emplea una posición de memoria de programa entre las direcciones **0000h a 1FFFh**. Para esto el contador de programa PC debe tener 13 bits; de esta manera puede direccionar $2^{13} = 8\,192$ posiciones de memoria de programa.

```
NOTA: Es muy común confundir el número de bits de cada instrucción
= 14 bits, con el número de bits del contador de programa PC = 13
bits. 2^13 = 8192 posiciones de memoria de programa.
```

16.1 UBICACIÓN

Las posiciones de memoria se aprecian en la Figura 16.1 que nos muestra el mapa de **memoria de programa** o **de usuario**.

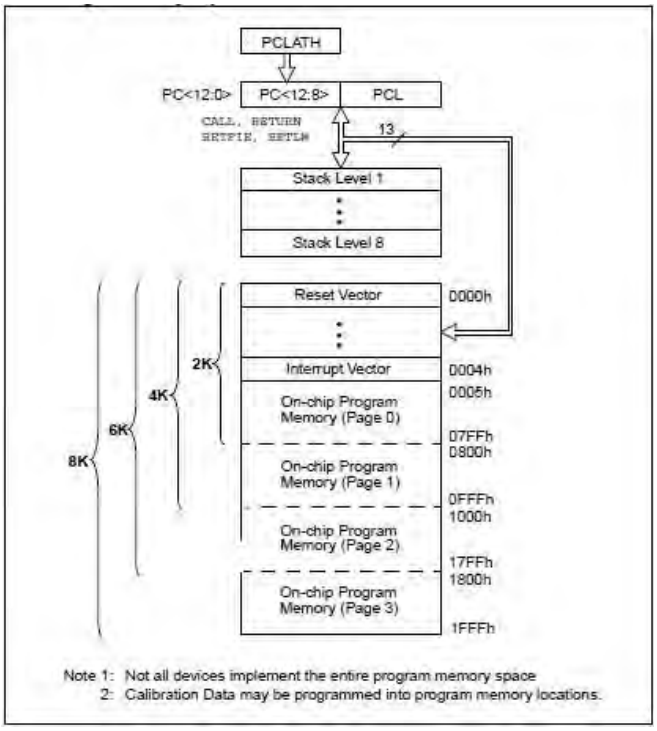


Figura 16.1 Diagrama mapa de memória.

Se pueden observar los 4 bloques de memoria, cada uno con sus direcciones límite y con el tamaño asociado. De este modo, si tenemos una memoria de **4K = 4 096** instrucciones, sus direcciones son desde **0000h a 0FFFh**, con 2 bloques de memoria de **2K = 2 048** instrucciones cada uno. Estos bloques de memoria se denominan **páginas** (*page*). Cada **página** (*page*) tendrá las direcciones siguientes:

Page 0 0000h a 07FFh, 2 048 posiciones de memoria
Page 1 0800h a 0FFFh, 2 048 posiciones de memoria

De otra forma, si tenemos una memoria de **8K = 8 192** instrucciones, sus direcciones son desde **0000h** a **1FFFh,** con 4 bloques de memoria de **2K = 2 048** instrucciones cada uno. Éste es el caso particular de los **PIC16F8XX** (los de este libro). Estos bloques de memoria se denominan **páginas** (*page*) y tienen un direccionamiento con 11 bits, de tal forma que 2^{11} = 2 048. Cada **Página** (*Page*) tendrá las direcciones siguientes:

Page 0 0000h a 07FFh, 2 048 posiciones de memoria
Page 1 0800h a 0FFFh, 2 048 posiciones de memoria

Page 2 1000h a 17FFh, 2 048 posiciones de memoria
Page 3 1800h a 1FFFh, 2 048 posiciones de memoria

Resumen:

- Máximo 8 k word de espacio de memoria (13 bits de dirección)
- Cuatro páginas cada una de 2 k word (11 bits de dirección
- Vector de reset en 0000h
- Vector de interrupción en 0004h
- Acceso a página (*page*) usando PCLATH<4:3>

La forma de seleccionar estos bloques de memoria usando PCLATH<4:3> se discutirá a continuación.

16.2 ACCESO

Al realizar un programa podemos usar cualquiera de los bloques de memoria para poner nuestras instrucciones. En el código del programa esto se realiza mediante directivas que nos ayudan a definir las posiciones de memoria en cualquiera de las cuatro **páginas** (*page*), como se observa en la Figura 16.2. Esas direcciones pueden accederse con el contador de programa PC, mediante diferentes operaciones que veremos a continuación.

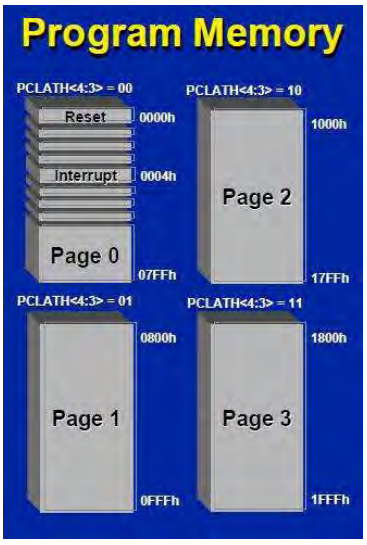


Figura 16.2 Diagrama mapa de memória por páginas.

https://yolibrospdf.com

A estas posiciones de memoria se accede mediante el registro PC. Este registro se compone de dos registros: **PCH:PCL = PC, PCH<4:0> (5 bits)** es el registro **PC high** o alto del **PC, PCL<7:0> (8 bits)** es el registro **PC low** bajo de PC, tal como se ilustra en la Figura 16.3.

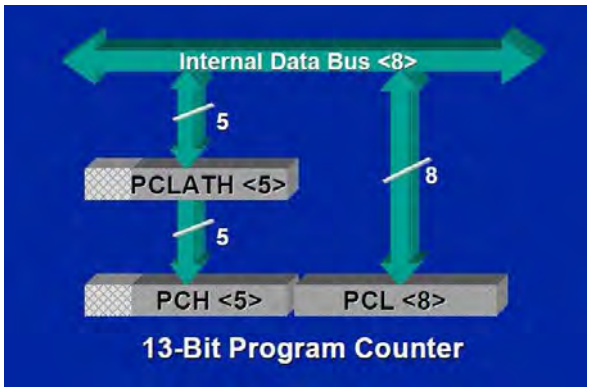


Figura 16.3 Contador de programa.

El registro **PC<12:0>** está conformado por los 13 bits para direccionamiento total del mapa de memoria de $2^{13} = 8\,192$ posiciones.

```
NOTA: Para acceder a cada una de las Páginas (Page) se deben poner
los bits correspondientes en el registro PCLATH<4:3>.
```

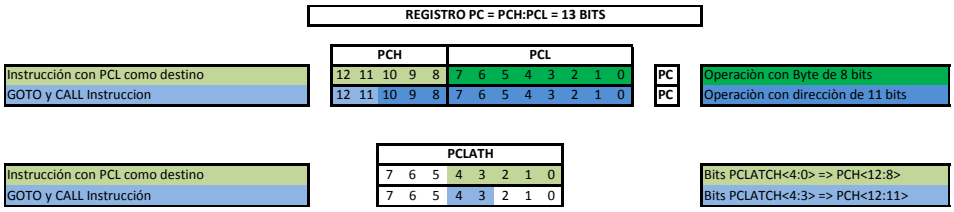


Figura 16.4 Registros PCL y PCLATH.

Según el tipo de instrucción, el registro PC se puede cargar con diferentes tipos de datos; observe la Figura 16.4 en las distintas situaciones que detallaremos después. Existen 4 tipos de situaciones al cargar el contador de programa PC:

- **Situación 1:** Instrucciones con PCL como destino.

  El PC es cargado con una escritura al PCL (y carga **PCH** con **PCLATH<4:0>**).

- **Situación 2:** Instrucción **goto**.

  El PC es cargado durante una instrucción **goto (PC<10:0>** con el *label* o el **opcode** del programa, carga **PC<12:11>** con **PCLATH<4:3>)**.

- **Situación 3:** Instrucción CALL.

  El **PC** es cargado durante una instrucción **CALL (PC<10:0>** con el *label* o el **opcode** del programa, **PC<12:11>** con **PCLATH<4:3>)**. La PC es cargada a la parte alta del *stack* (esto se llama *push* al *stack*).

- **Situación 4:** Instrucciones **RETURN, RETFIE** o **RETLW**.

  El PC es cargado durante una instrucción **RETURN, RETFIE** o **RETLW**, donde el PC es cargado desde la parte alta del *stack* (esto se llama **POP** desde el *stack*).

Cada una de estas situaciones se explica con los respectivos gráficos de los registros involucrados, los cuales nos ayudan a visualizar todo el proceso de cada situación.

16.2.1 SITUACIÓN 1: INSTRUCCIONES CON PCL COMO DESTINO

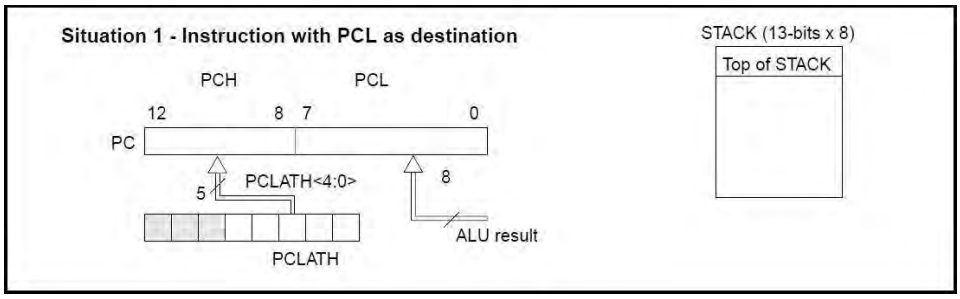


Figura 16.5 Instrucciones con PCL como destino.

Esto sucede con instrucciones del tipo **xxxx f,d** o las instrucciones orientadas a bytes que aparecen en el **set de instrucciones**. Los datos tipo byte contienen 8 bits, que nos dan un direccionamiento de hasta $2^8 = 256$ posiciones de memoria, que es el rango máximo de operación de estas operaciones.

Ejemplo:

```
andwf     PCL,F     ; haga W and PCL -> PCL
movwf     PCL,F     ; haga W -> PCL
```

El resultado de la operación queda guardado en el registro **PCL<7:0>,** la parte baja del registro **PC**, mientras el registro **PCLATH<4:0>** tiene la dirección alta **PCH<4:0>** del registro **PC**, que se carga en el **PCH** o **PC<12:8>**. Esta forma de operación se utiliza más para configurar **tablas,** o también se llama **GOTO computado**; es bueno recordar que el direccionamiento máximo es de hasta 256 posiciones de memoria. Por tanto, las tablas tienen la limitante de 256 bytes.

```
NOTA: Recuerde cargar el PCLATH<4:0> antes de la llamada a la
tabla. De no ser así el resultado es inesperado.
```

En la **Parte 3: Programación I, Capítulo 11.0, Subsección 11.5.3: Datos en tipos de memoria, ítem Tablas,** se da mayor información sobre las tablas.

16.2.2 SITUACIÓN 2: INSTRUCCIÓN GOTO

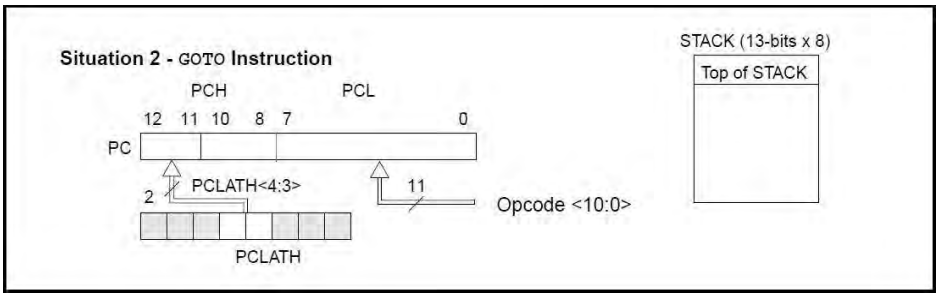


Figura 16.6 Instrucción GOTO.

Con la instrucción **goto** el **PC<10:0>** es cargada con una dirección o *label* provenientes del **opcode** del programa el cual tiene una longitud de 11 bits. Con 11 bits podemos direccionar hasta $2^{11} = 2\ 048$ posiciones de memoria que corresponden a una **Página** (*Page*). Los dos bits faltantes del **PC<12:11>** son cargados desde el **PCLATH<4:3>**. Recuerde que para accesar las 4 **páginas** (*page*) la selección se hace con los bits de **PCLATH<4:3>,** como se observa en la Figura 16.6.

Ejemplo:

```
goto        start       ; start posición 0024h
goto        Main_Loop   ; Main_Loop posición 0012h
```

Recuerde también que si no ha cargado previamente la dirección **PCLATH<4:3>** para seleccionar una **página** (*page*), el programa asume el valor que tiene actualmente.

```
NOTA: Mientras su programa esté en la misma página, no hay
problemas de carga de PCLATH<4:3> para cambiar de Página (Page).
```

16.2.3 SITUACIÓN 3: INSTRUCCIÓN CALL

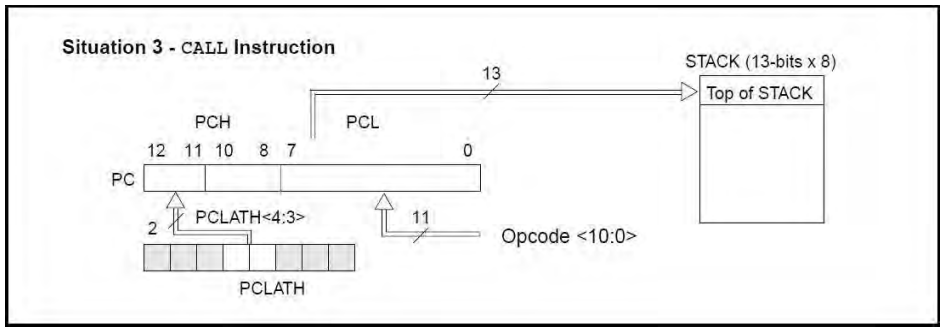


Figura 16.7 Instrucción CALL.

Con la instrucción **CALL** se llama a una subrutina, que luego de ser realizada devuelve el flujo del programa, justamente en la instrucción siguiente a la instrucción **CALL**. Con la instrucción **CALL** el contador de programa **PC<10:0>** se carga con la dirección o el *label* provenientes del **opcode** del programa, el cual tiene una longitud de 11 bits. Los bits **PC<12:11>** son cargados desde el **PCLATH<4:3>**. Además, deshabilita globalmente las interrupciones poniendo a cero el bit **GIE**. Con 11 bits podemos direccionar hasta $2^{11} = 2\ 048$ posiciones de memoria que corresponden a una **Página** (*Page*). El PC es cargado a la parte alta del *stack* (esto se llama *push* al *stack*); este dato es de 13 bits, que son el ancho del *stack*.

```
NOTA: Push es una denominación del procedimiento anterior; no es
una instrucción que se encuentre en el set de instrucciones.
Mientras las subrutinas estén en la misma página, no hay problemas
de carga de PCLATH<4:3> para cambiar de página (page).
```

Una manera de usar esta situación son las denominadas **Tablas**; al respecto, se pueden encontrar más detalles de este uso en la **Parte 3: Programación I**, en el **Capítulo 11.0, Subsección 11.5.1: Tipo de direccionamiento**, en el tema **Direccionamiento indexado**, y también en esta misma parte, en el **Capítulo 11.0, Sección 11.5: Datos,** en la **Subsección 11.5.3: Datos en Tipos de Memoria,** en los temas **Memoria de programa** y **Tablas**.

16.2.4 SITUACIÓN 4: INSTRUCCIONES RETURN, RETFIE O RETLW

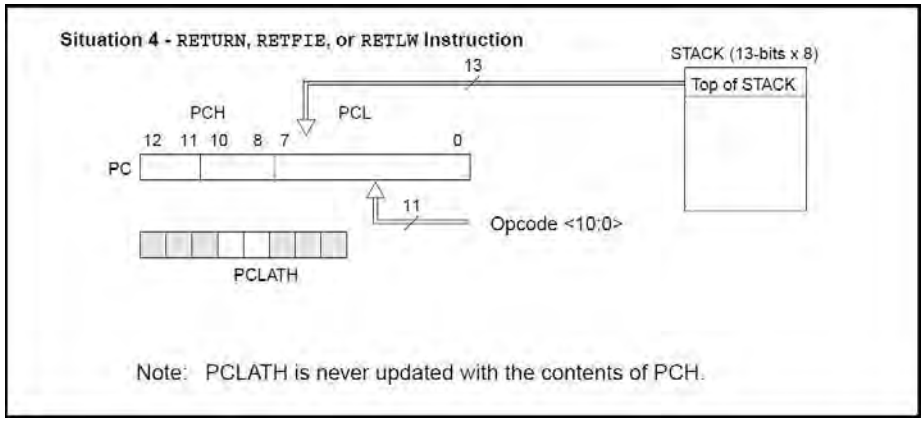


Figura 16.8 Instrucciones RETURN, RETFIE o RETLW.

Con las instrucciones **RETURN, RETFIE** o **RETLW**, el PC es cargado desde la parte alta del *stack* (esto se llama **POP** desde el *stack*). En esta situación los 13 bits del **PC<12:0>** son cargados directamente del *stack*, que contiene una dirección previamente almacenada, desde donde se hizo una llamada a subrutina, o se generó una interrupción.

Las instrucciones **RETURN** y **RETLW** son la manera de terminar una subrutina y volver al programa principal. El **PC<12:0>** es directamente cargado desde el *stack* que contiene la dirección **+1**, desde donde se llamó la subrutina.

La instrucción **RETFIE** es la forma de terminar una rutina de servicio de interrupción y volver al programa principal. El **PC<12:0>** es directamente cargado desde el *stack* que contiene la dirección **+1**, desde donde se generó la interrupción. Además, **RETFIE** habilita globalmente las interrupciones poniendo a uno el bit **GIE**. El PC es cargado desde la parte alta del *stack* (esto se llama **POP** desde el *stack*).

```
NOTA: POP es una denominación del procedimiento anterior; no es una
instrucción que se encuentre en el set de instrucciones.
```

Podrá encontrar ejemplos de las situaciones descritas anteriormente en nuestra página web, en la carpeta:

X:\Ejemplos_ASM, copiada en:
C:\ Ejemplos_ASM\Memory_PC_1\Memory_PC_1.asm, y también en:
C:\ Ejemplos_ASM\Memory_PC_1\Memory_PC_2.asm.

16.2.5 PROTECCIÓN CP Y WRT

Cuando el micro tiene código de protección de programa **CP** habilitado (en el registro de Configuración, visto anteriormente), la memoria de programa no puede ser leída ni escrita desde el exterior con un programador, a menos que se programe de nuevo el microcontrolador con otro programa. Sin embargo, el programa puede correr en forma normal en modo de *Run*. Los bits **WRT<1:0>** (en el registro de Configuración, antes visto) pueden seleccionar cuál área de **memoria de programa** o *flash* será protegida para escritura.

16.3 LEER Y ESCRIBIR EN LA MEMORIA DE PROGRAMA

La **memoria de programa** (*Program Memory*) o memoria *flash* se escribe usualmente mediante el procedimiento de programar o "quemar" el microcontrolador, descrito en la **Parte 3: Programación I,** en el **Capítulo 14.0: Programación.** De esta forma, los contenidos en la **memoria de programa** o *flash* del micro corresponden a las instrucciones del programa realizado como archivo ***** *.asm;* al compilarse este archivo del programa genera el archivo *** .hex,** que es el que contiene los códigos de las instrucciones en el formato **INHX8M**, que se utiliza para programar el micro.

El contenido de los archivos *** .hex** y la explicación del formato de trama **INHX8M** se describen en la **Parte 3: Programación I**, en el **Capítulo 11.0**, **Sección 11.2.3: Tipos de archivos**, en el ítem **Archivo * .HEX**. Para este procedimiento de **programación** se necesita una computadora con el software de programación instalado MPLAB u otro similar, la herramienta de programación **PICkit 2** u otro similar, y el archivo *** .hex**. En este modo se requiere un voltaje de programación de 14 voltios sobre el pin.

También podemos leer o escribir en la memoria de programa *flash* de otra forma que no es en el modo de programación; el microcontrolador puede realizar este nuevo procedimiento en modo de **RUN** o ejecución del programa, con el voltaje de alimentación V_{DD} = 5 voltios. Es decir, podemos realizar una rutina de lectura/escritura de datos en la memoria de programa o *flash*, de manera muy similar a la rutina con la que se leen/escriben datos en la memoria EEPROM. La diferencia entre leer o escribir datos en memoria EEPROM y en memoria de programa o *flash* es el tipo de dato.

- En la memoria EEPROM los datos son tipo byte (8 bits) y las direcciones igualmente son tipo byte (8 bits), 2^8 = 256 posiciones de memoria.

- En la memoria de programa o *flash* los datos son de tipo *word* (14 bits) y las direcciones son el rango de la memoria (13 bits), 2^{13} = 8 192 posiciones de memoria.

En la Tabla 16.1 podemos apreciar esto.

Tabla 16.1 Datos y direcciones en memorias EEPROM y FLASH.

DIFERENCIAS DE RANGOS DE VALORES PARA DATOS EN EEPROM Y FLASH									
		BITS		RANGO DE VALORES			REGISTROS USADOS		EEPROM
	TIPO	No	MSB:LSB		DEC	HEX	HIGH	LOW	
DATO	BYTE	8	7:0	256	0 - 255	00 - FF	- - -	EEDATA	
DIRECCIÒN	BYTE	8	7:0	256	0 - 255	00 - FF	- - -	EEADR	
		BITS		RANGO DE VALORES			REGISTROS USADOS		FLASH
	TIPO	No	MSB:LSB		DEC	HEX	HIGH	LOW	
DATO	WORD	14	13:0	16384	0 -16383	0000 - 3FFF	EEDATH	EEDATA	
DIRECCIÒN	2 BYTE	13	12:0	8192	0 - 8191	0000 - 1FFF	EEADRH	EEADR	

Observe las diferencias entre el número de bytes y bits, rango de valores y registros usados para las memorias EEPROM y *flash*.

16.3.1 REGISTROS USADOS

La memoria de programa o *flash* es de lectura/escritura en modo normal de operación (modo Run); estas posiciones de memoria se accesan mediante un procedimiento (no es de la forma indirecta que se vio en la **Parte 3: Programación I, Capítulo 11.0, Sección 11.5.3, subtítulo: Memoria de programa, tema: Tipo de direccionamiento (o de "acceso")**. En este procedimiento se utilizan los registros:

- EECON1 registro de control 1
- EEDATA registro de datos *low* (bajo)
- EEDATAH registro de datos *high* (alto)
- EEADR registro de dirección *low* (bajo)
- EEADRH registro de dirección *high* (alto)

Para leer o escribir las direcciones de la memoria de programa o *flash* se necesitan 13 bits (2^{13} = 8 192 posiciones de memoria), de manera que necesitamos dos bytes para direccionar una posición de la memoria *flash*, de los cuales usaremos 13 bits, o sea los bits <12:0>. Los dos bytes utilizados para la dirección son EEADRH registro de dirección *high* (alto) y EEADR registro de dirección *low* (bajo).

Para leer o escribir los datos (*word*) de/en la memoria de programa o *flash* se necesitan 14 bits (tomados del set de instrucciones), de manera que necesitamos

dos bytes para un dato (*word*) de la memoria *flash*, de los cuales usaremos 14 bits, o sea los bits **<13:0>**. Los dos bytes utilizados para la dirección son EEDATAH registro de datos *high* (alto), y EEDATA registro de datos *low* (bajo).

```
NOTA: En la memoria flash se almacenan instrucciones; cada posición
de memoria tiene 14 bits denominados word. Cuando se escriben datos
en estas posiciones deben ser del tipo word de 14 bits.
```

En la Figura 16.8 apreciamos la lectura de una posición de la memoria de programa o *flash*, y los contenidos de los registros usados.

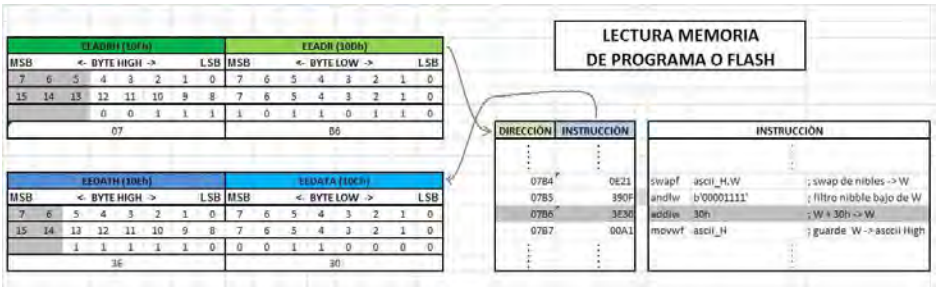


Figura 16.9 Lectura de memoria *flash*

En la Figura 16.9 se muestra un segmento de programa alojado en la memoria de programa o *flash*, en el cual se lee una posición de memoria (**07B6h**) que contiene la instrucción **addlw 30h**, cuyo código es (**3E30h**). Se aprecian claramente los bits usados por los registros **EEADRH** y **EEADR**, cuyo valor (**07B6h**) es la dirección que se va a leer y que usa 13 bits **<12:0>**, y los registros **EEDATAH** y **EEDATA**, cuyo valor (**3E30h**) es el dato leído, el cual usa 14 bits **<13:0>**. En la figura 16.10 se aprecia el registro de control **EECON1** con sus bits de control.

libros.ep-electropc.com

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD**: Program/Data EEPROM Select bit
1 = Accesses program memory
0 = Accesses data memory
Reads '0' after a POR; this bit cannot be changed while a write operation is in progress.

bit 6-4 **Unimplemented**: Read as '0'

bit 3 **WRERR**: EEPROM Error Flag bit
1 = A write operation is prematurely terminated (any $\overline{MCLR}$ or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN**: EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR**: Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD**: Read Control bit
1 = Initiates an EEPROM read; RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Figura 16.10 Registro de control EECON1.

16.3.2 PROCEDIMIENTO

El procedimiento de lectura y de escritura son diferentes, y los describiremos como:

- Procedimiento para lectura.
- Procedimiento para escritura.

A continuación veremos estos procedimientos.

PROCEDIMIENTO DE LECTURA

Para leer datos de la memoria de programa o *flash*, se necesita poner la dirección en los registros EEADRH y EEADR, poner en uno el bit EEPGD que apunta a la **memoria de programa** o *flash*; luego se pone en 1 el bit RD que inicia la lectura; aquí

se esperan dos ciclos que necesita el módulo para obtener el dato (se deben poner 2 instrucciones nop), y luego se lee el dato en los registros EEDATH y EEDATA. Para hacer la lectura en la memoria de programa o *flash*, los pasos son:

1. Ponga en los registros EEADRH y EEADR la dirección del dato que se va a leer.
2. Ponga a uno el bit de control EEPGD en el registro EECON1<7>, que apunta a la memoria de programa o *flash*.
3. Ponga a uno el bit RD en el registro EECON1<0>, el cual realiza la lectura.
4. Espere 2 ciclos (las instrucciones siguientes deben ser nop).
5. Lea el dato en los registros EEDATH y EEDATA.

A continuación realizamos estos pasos con código:

1. Ponga en los registros EEADRH y EEADR la dirección del dato que se va a leer:

```
        bcf     STATUS, RP0         ; cambie banco
        bsf     STATUS, RP1         ; cambie a Banco 2

        movf    address_EE_L,W      ; cargue dirección low -> W
        movwf   EEADR               ; guarde en registro de dirección de memoria flash a leer
        movf    address_EE_H,W      ; cargue dirección high -> W
        movwf   EEADRH              ; guarde en registro de dirección high de memoria flash a leer
```

2. Ponga a uno el bit de control EEPGD en el registro (EECON1<7>), que apunta a la memoria de programa o *flash*:

```
        bsf     STATUS, RP0         ; Bank 3
        bsf     EECON1, EEPGD       ; apunte a memoria de Programa o flash
```

3. Ponga a uno el bit RD en el registro (EECON1<0>), el cual realiza la lectura:

```
        bsf     EECON1, RD          ; habilite bit de lectura
```

4. Espere 2 ciclos (las instrucciones siguientes deben ser **nop**):

```
        nop                         ; ciclo necesario
        nop                         ; ciclo necesario
```

5. Lea el dato en los registros EEDATH y EEDATA:

```
        bcf     STATUS, RP0         ; Bank 2

        bcf     STATUS, RP0         ; Bank 2
```


libros.ep-electropc.com

```
        movf    EEDATA, W       ; EEDATA -> W
        movwf   data_EE_L       ; guarde dato low leído en la memoria flash

        movf    EEDATH, W       ; EEDATH -> W
        movwf   data_EE_H       ; guarde dato high leído en la memoria flash

        bcf     STATUS, RP1     ; Banco 0
```

Más adelante se encuentra el **Ejemplo de lectura** del procedimiento descrito aquí.

PROCEDIMIENTO DE ESCRITURA

Para escribir en la memoria de programa o *flash,* solamente se puede hacer si las posiciones de memoria que se van a escribir están sin *Write-protect,* que son los bits WRT1:WRT0 del registro *Bits de Configuración* descrito en la **Parte 2: Arquitectura,** en el **Capítulo 9.0: Funciones especiales de la CPU,** en la **Sección 9.6: Bits de configuración (*Configuration Bits*)**.

La memoria de programa *flash* se escribe en bloques de 4 datos *word*; estos registros están en direcciones consecutivas, de tal forma que los bits del registro inicial son **EEADR<1:0> = 00**; esto significa que la posición inicial en donde se desea escribir los datos, debe ser múltiplo de 4 exacto. Los dos primeros bits del registro de direcciones se sobrescriben **EEADR<1:0> = 00** a la dirección cargada. Los datos se borran y luego se escriben.

Ejemplo:

Si se van a escribir 4 datos *word*, empezando en la posición (**05C2h)**, el módulo de escritura comenzará a escribir los 4 datos *word* desde la posición (**05C0h**). Si se van a escribir 4 datos *word*, empezando en la posición (**036Eh**), el módulo de escritura comenzará a escribir los 4 datos *word* desde la posición (**036Ch**).

BUFFER Y CICLO DE ESCRITURA

Para escribir un bloque de 4 datos *word* (Figura16.11), primero deben ser cargados al buffer de 4 datos *word*. La escritura se hace secuencialmente en el buffer: dato 1, dato 2, dato 3 y el dato 4 son tipo *word*; y aquí, al llegar a este dato 4, los bits **EEADR<1:0> = 11**, en forma automática se borran los 4 registros del buffer y los 4 datos *word* son escritos en la memoria de programa *flash*.

libros.ep-electropc.com

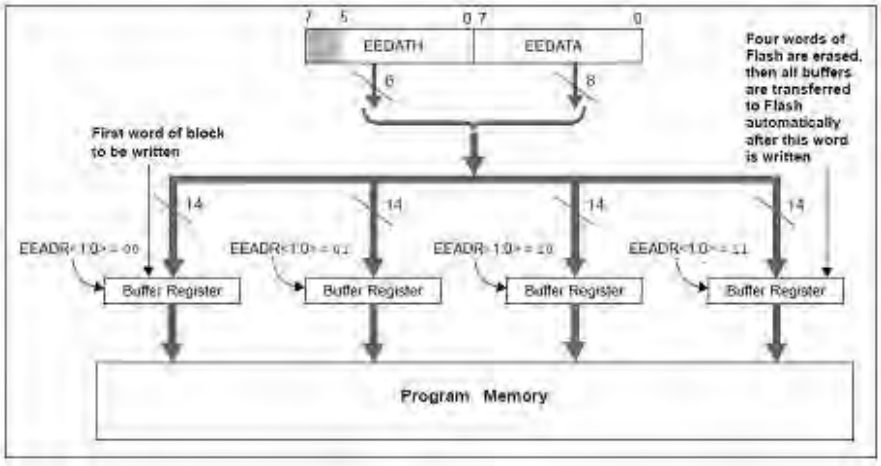


Figura 16.11 Buffer modo de escritura memoria de programa o *flash*.

En la escritura de los datos (dato 1, dato 2, dato 3) que quedan escritos en el buffer, sólo se toman 2 ciclos; únicamente después de escribir el dato 4, el módulo comienza a escribir estos 4 datos en la memoria de programa *flash*. El módulo toma un tiempo de escritura en la memoria de programa *flash*; este tiempo se toma solamente cuando se guarda el último dato en el buffer; esto no sucede con los datos anteriores; este tiempo de escritura es de 4 – 8 ms. En la Tabla 16.2 podrá observar las especificaciones de tiempos de escritura.

Tabla 16.2 Especificaciones de tiempos de escritura de memoria EEPROM y *flash*.

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
		Data EEPROM Memory					
D120	ED	Endurance	100K	1M	—	E/W	-40°C to +85°C
D121	VDRW	VDD for read/write	VMIN	—	5.5	V	Using EECON to read/write, VMIN = min. operating voltage
D122	TDEW	Erase/write cycle time	—	4	8	ms	
		Program Flash Memory					
D130	EP	Endurance	10K	100K	—	E/W	-40°C to +85°C
D131	VPR	VDD for read	VMIN	—	5.5	V	VMIN = min. operating voltage
D132A		VDD for erase/write	VMIN	—	5.5	V	Using EECON to read/write, VMIN = min. operating voltage
D133	TPEW	Erase/Write cycle time	—	4	8	ms	

Todas las 4 posiciones de los registros del buffer deben ser escritas con los datos correctos; es decir, si solamente uno, dos o tres registros están siendo escritos en un bloque de 4 registros, al leer las posiciones de memoria conservarán el dato como si no hubieran sido grabadas. De tal manera que si desea escribirlas debe hacer una secuencia correcta de 4 datos *word*.

SECUENCIA DE ESCRITURA

Para escribir 4 datos en la memoria de programa *flash*, la secuencia es:

1. Ponga la dirección de destino en los registros **EEADRH:EEADR**.
2. Ponga el dato en los registros **EEDATAH:EEDAT**.
3. Ponga a uno el bit **EEPGD** en el registro de control (**EECON1<7>**), el cual apunta a la memoria de programa o *flash*.
4. Ponga a uno el bit **WREN** control (**EECON1<2>**), habilite la escritura en la memoria de programa o *flash*.
5. Escriba (**55h**) y luego (**AAh**) en el registro **EECON2** (secuencia de escritura en *flash*).
6. Ponga a uno el bit **WR** del registro de control (**EECON1<1>**), lo que comienza la escritura de uno de los 4 datos *word* por este ciclo.
7. Ponga a cero el bit **WREN** control (**EECON1<2>**), deshabilite la escritura en la memoria de programa o *flash*.
8. Verifique si es el último dato para buffer de escritura (sí, termina el ciclo/no, continúa).
9. Espere el tiempo de escritura de 4 ms.

En seguida, damos los pasos anteriores con código:

1. Ponga la dirección de destino en los registros **EEADRH:EEADR**:

```
        bsf     STATUS,RP1      ; RP1 = 1
        bcf     STATUS,RP0      ; RP0 = 1, Banco 2

        movf    address_EE_L,W  ; cargue dirección low -> W
        movwf   EEADR           ; guarde en registro de dirección low de memoria flash

        movf    address_EE_H,W  ; cargue dirección high -> W
        movwf   EEADRH          ; guarde en registro de dirección high de memoria flash
```

2 Ponga el dato en los registros **EEDATAH:EEDAT**:

```
        movlw   datos_flash     ; principio de datos flash
```

```
        movwf   FSR             ; cárguelo al apuntador FSR
        bsf     STATUS,IRP      ; pone Banco 2 y 3 para FSR

Write_loop:
        movf    INDF,W          ; cargue indirecto -> W
        movwf   EEDATH          ; guarde en registro dato high memoria flash
        incf    FSR,F           ; FSR + 1

        movf    INDF,W          ; cargue indirecto -> W
        movwf   EEDATA          ; guarde en registro dato low memoria flash
        incf    FSR,F           ; FSR + 1
```

3 Ponga a uno el bit **EEPGD** en el registro de control (**EECON1<7>**), el cual apunta a la memoria de programa o *flash:*

```
        bsf     STATUS,RP0      ; Banco 3
        bsf     EECON1,EEPGD    ; apunta a memoria de programa flash
```

4 Ponga a uno el bit **WREN** control (**EECON1<2>**), habilite la escritura en la memoria de programa o *flash:*

```
        bsf     EECON1,WREN     ; habilita escritura
```

5 Escriba (**55h**) y luego (**AAh**) en el registro **EECON2** (secuencia de escritura en *flash*):

```
        movlw   0x55            ; secuencia obligada
        movwf   EECON2          ; escribe 55h
        movlw   0xAA            ; secuencia obligada
        movwf   EECON2          ; escribe AAh
```

6 Ponga a uno el bit **WR** del registro de control (**EECON1<1>**), lo que comienza la escritura de uno de los 4 datos *word*.

```
        bsf     EECON1,WR       ; pone bit WR para empezar escritura
        nop                     ; ciclo de espera necesario
        nop                     ; ciclo de espera necesario
```

7 Ponga a cero el bit **WREN** control (**EECON1<2>**), deshabilite la escritura en la memoria de programa o *flash*.

```
        bcf     EECON1,WREN     ; des-habilita escritura
```

8 Verifique si es el último dato para buffer de escritura (sí, termina el ciclo/no, continúa):

```
        bcf     STATUS,RP0      ; RP0 = 0, Banco 2

        movf    EEADR,W         ; cargue EEADR -> W
        andlw   0x03            ; filtra bits <1:0>
        Xorlw   0x03            ; compare con 3
        btfsc   STATUS,Z        ; Z = 0? no, son iguales, salte y continúe
        goto    Write_end       ; sí, termine

        incf    EEADR,F         ; EEADR + 1
        goto    Write_loop      ; no, continúe
Write_end
        bcf     STATUS,RP1      ; RP1 = 0, Banco 0
        bcf     STATUS,IRP      ; pone Banco 0 y 1 para FSR
```

9 Espere el tiempo de escritura de 4 ms.

Más adelante se encuentra el **Ejemplo de escritura** del procedimiento descrito aquí.

16.3.3 PRÁCTICA

Para realizar una práctica de los procedimientos descritos realizaremos dos ejemplos: uno de lectura y otro de escritura.

EJEMPLO DE LECTURA

Leeremos un dato de la memoria de programa o *flash*. Para realizar esta práctica, necesitamos el sistema de evaluación **PIC16F877A** y la herramienta **PICkit 2**; el montaje podemos apreciarlo en la Figura 16.12.



Figura 16.12 Montaje práctica *flash*.

Lo que se intenta hacer es leer un dato que está en la posición (**07B5h**) y que pertenece a un bloque de datos tipo *word*, tal como se aprecia en la Figura 16.13.

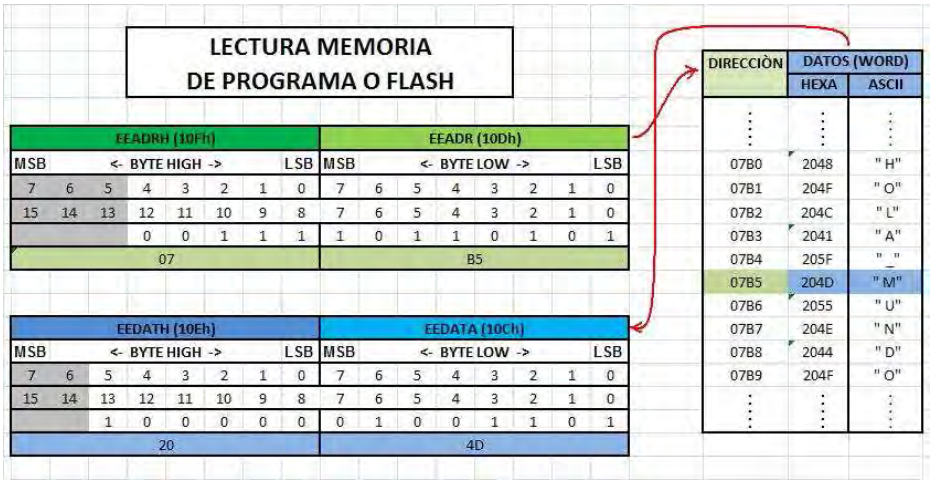


Figura 16.13 Lectura memoria de programa o *flash*.

El código del ejemplo se encuentra en nuestra página Web, en la carpeta:
X:\Ejemplos_ASM que se debe copiar al:
C: \, para encontrarlo como:
C:\Ejemplos_ASM\FLASH_1\FLASH_1.asm.

A continuación damos el código del ejemplo:

```
 list p = 16f877a      ; directiva para definir procesador
#include <p16f877a.inc> ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

 cblock 110h                          ; desde la posición 110h en RAM
                                      ; es el mismo Banco 2, donde están los registros usados
data_EE_L                             ; dato low memoria flash
data_EE_H                             ; dato high memoria flash

address_EE_L                  ; dirección low memoria flash
address_EE_H                  ; dirección high memoria flash
 endc                         ; fin de registros

; la dirección del dato a leer se cambia en las 2 siguientes instrucciones
#define    _address_EE_L      0XB5              ; dirección low memoria flash
#define    _address_EE_H      0x07              ; dirección high memoria flash

 org 0
            goto               start             ; vaya a comienzo de programa
;-------------------------------------------------------------------------
; Lee un dato de la memoria flash,
; entrada dirección del dato (address_EE_H:address_EE_L), hacia registros (EEADRH:EADR)
; salida dato (data_EE_H:data_EE_L) de registros (EEDATAH:EEDATA)
```

```
Read_flash
; dirección del dato a leer de la memoria flash
        BANKSEL address_EE_L        ; cambie Banco 2 a address_EE_L

        movf    address_EE_L,W      ; cargue dirección low -> W
        movwf   EEADR               ; guarde en registro de dirección de memoria flash a leer

        movf    address_EE_H,W      ; cargue dirección high -> W
        movwf   EEADRH              ; guarde en registro de dirección high de memoria flash a leer

        bsf     STATUS, RP0         ; Bank 3
        bsf     EECON1, EEPGD       ; apunte a memoria de programa o flash
        bsf     EECON1, RD          ; habilite bit de lectura
        nop                         ; ciclo necesario
        nop                         ; ciclo necesario

        bcf     STATUS, RP0         ; Bank 2

        movf    EEDATA, W           ; EEDATA -> W
        movwf   data_EE_L           ; guarde dato low leído en la memoria flash

        movf    EEDATH, W           ; EEDATH -> W
        movwf   data_EE_H           ; guarde dato high leído en la memoria flash

        bcf     STATUS, RP1         ; Banco 0

        return
;-----------------------------------------------------------------
Start:
; lee un dato desde la posición (_address_EE_H:_address_EE_L) en la Memoria EEPROM
; y lo guarda en las variables (data_EE_H:data_EE_L)

        BANKSEL address_EE_L        ; cambie banco a address_EE_L

        movlw   _address_EE_L       ; cargue dirección -> W
        movwf   address_EE_L        ; dirección del dato a leer de la memoria flash/EEPROM

        movlw   _address_EE_H       ; cargue dirección high -> W
        movwf   address_EE_H        ; dirección high del dato a leer de la memoria flash

        bcf     STATUS, RP1         ; Banco 0

        call    Read_flash          ; lee en flash

        goto    $
;-----------------------------------------------------------------
        org     0X07B0              ; define posición en memoria de programa
        dw " H O L A _ M U N D O"
;-----------------------------------------------------------------
 End
```

En el programa anterior se adicionaron datos a las posiciones de memoria de programa *flash*; el principio de los datos *word* se ubica con la instrucción `org 0x07B0` en la dirección (**0x7B0**); con la directiva `dw " H O L A _ M U N D O"` se definen los datos y quedan como se ilustra en la Figura 16.14.

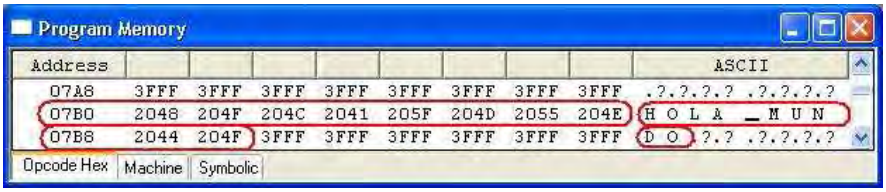


Figura 16.14 Datos *word* en memoria *flash*.

Para cambiar la dirección que se va a leer de la memoria de programa o *flash*, se deben cambiar los valores de las 2 instrucciones (en el programa anterior):

```
; la dirección del dato a leer se cambia en las 2 siguientes instrucciones:
#define   _address_EE_L     0xB5              ; dirección low memoria flash
#define   _address_EE_H     0x07              ; dirección high memoria flash
```

Para hacer el seguimiento del programa anterior en modo de depuración necesitamos agregar dos ventanas, la ventana **Watch** y la ventana **Program Memory**. Para agregar la ventana **Watch** y los registros que se muestran en la Figura 16.13, se sigue el procedimiento descrito en **Parte 3: Programación II**, en el **Capítulo 14.0, Sección 14.5: Depuración, Subsección 14.5.4: Programa ejemplo con subrutinas,** en el ítem **Paso 6 Agregar la ventana Watch y Registros GPR**.

AGREGAR VENTANA PROGRAM MEMORY

Para agregar la ventana **Program Memory,** que es la que muestra los contenidos de la memoria de programa o *flash*, se escoge en el menú de opciones *View/Program Memory,* al hacer clic en esta opción se despliega la ventana **Program Memory**, tal como se ilustra en la Figura 16.15.

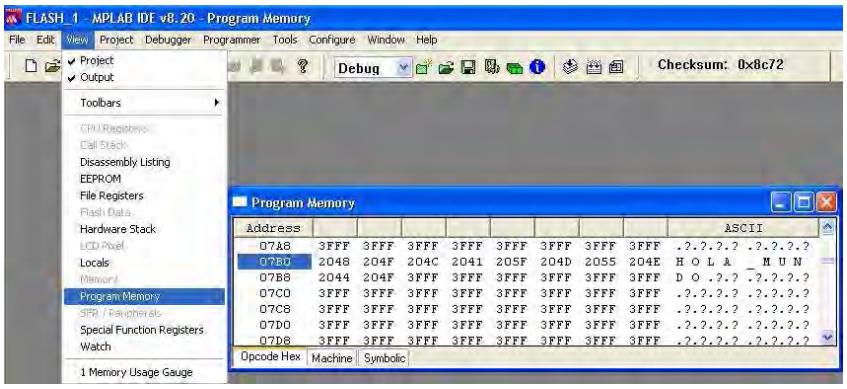


Figura 16.15 Ventana memoria de programa o *flash*.

Con las ventanas **Program Memory** y **Watch** abiertas y con los registros que se van a revisar, cargados en la ventana **Watch**, más el programa cargado, podemos obtener una imagen, como se muestra en la Figura 16.16.

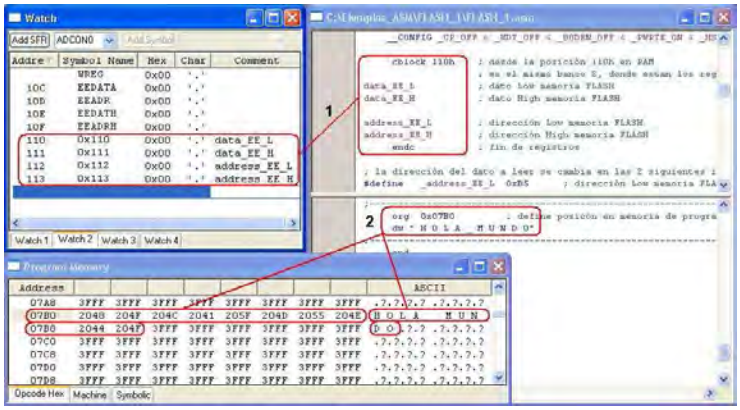


Figura 16.16
Depuración lectura de memoria *flash 1*.

En la Figura 16.16 aparecen dos ítems (**1**) y (**2**) que detallamos aquí:

- Ítem 1

En la ventana **Watch** vemos los registros definidos en el programa; observe que las posiciones de memoria de las variables son definidas desde la posición (110h) en el Banco 2; esto se debe a que los registros de datos EEDDATH: EEDDATA y los registros de dirección EEADRH:EEADR están en las posiciones (10Ch a 10F) en el Banco 2, el programa se facilita, minimizando así el cambio de bancos.

- Ítem 2

En la ventana **Program Memory** está el bloque de los datos *word* "H O L A _ M U N D O" con sus códigos correspondientes, desde la posición (07B0h) hasta la (07B9h). Observe que en la ventana **Watch** los registros están en 0, al iniciar el programa.

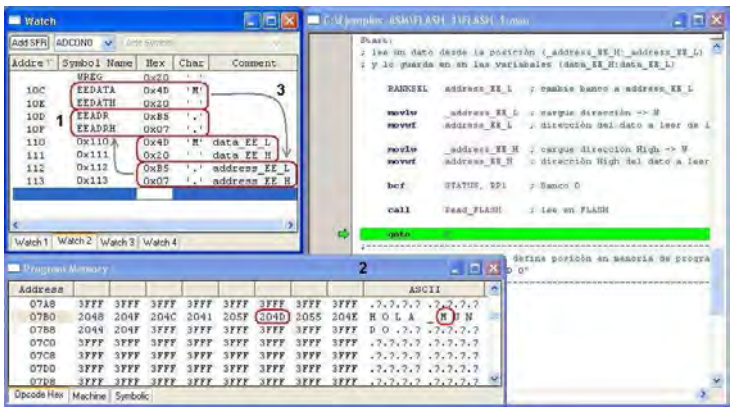


Figura 16.17
Depuración lectura de memoria *flash 2*.

En la Figura 16.17 podemos observar que aparecen tres ítems (**1**), (**2**) y (**3**) que detallamos en seguida.

- Ítem 1

En la ventana **Watch** se aprecia que en los registros de dirección de usuario (address_EE_H: address_EE_H) está la dirección que se va a leer (**07B5h**), y este valor se carga en los registros de dirección de la memoria de programa *flash* (EEADRH:EEADR).

- Ítem 2

En la ventana **Program Memory** se observa el bloque de los datos *word* " H O L A _ M U N D O" con sus códigos correspondientes, desde la posición (**07B0h**) hasta la posición (**07B9h**). El dato que se va a leer en la posición (07B5h) corresponde al código (**204Dh**) o (**"M"**).

- Ítem 3

En la ventana **Watch** se aprecia que en los registros de datos de usuario data_EE_H: data_EE_H está el dato tipo *word* leído (204Dh) y este valor se carga desde los registros de datos de la memoria de programa *flash* (EEDATH:EEDATA).

```
NOTA: Para actualizar los contenidos de la ventana de Program
Memory se debe usar el ícono  en la barra de herramientas.
```

ACTUALIZAR DATOS EN LA VENTANA PROGRAM MEMORY

Para actualizar los datos contenidos en la memoria de programa *flash* del microcontrolador, y para que éstos se puedan ver actualizados en la ventana **Program Memory** del MPLAB, se debe utilizar el ícono que se encuentra en la barra de herramientas, tal como muestra la Figura 16.18.



Figura 16.18 Barra de herramientas.

Al seleccionar el ícono aparece el cuadro de diálogo: **Read the target device memories**, que traduce: **Lea la memoria de programa de la tarjeta destino**. Si no se

realiza el paso anterior, los datos que vemos en la ventana **Program Memory** no corresponderán a los datos que contiene el microcontrolador en la memoria de programa *flash*. El código de un ejemplo similar al anterior, en el que se leen varios datos, se encuentra en nuestra página Web, en la carpeta:

X;\Ejemplos_ASM que se debe copiar al:
C:\, para encontrarlo como:
C:\Ejemplos_ASM\FLASH_1A\FLASH_1A.asm.

EJEMPLO DE ESCRITURA

Escribiremos 4 datos en la memoria de programa o *flash*. Para realizar esta práctica, necesitamos el sistema de evaluación **PIC16F877A** y la herramienta PICkit 2; en la Figura 16.19 podemos apreciar el montaje.



Figura 16.19 Montaje práctica *flash*.

Lo que se intenta hacer es escribir 4 datos tipo *word*, desde una posición en la memoria de programa *flash*. La posición inicial en donde quedarán en la memoria de programa o *flash* está dada por (_address_EE_H : _address_EE_L). Los datos que se van a grabar están en la RAM, desde la posición datos_flash (110h) que está en el Banco 2 (en el mismo banco de los registros de trabajo). Estos 8 datos en RAM se inicializan con los valores 00, 01, 02...07, en la rutina Ini_var cada 2 de ellos se convertirán en un dato *word*, así: (0001h), (0203h), (0405h), (0607h), que serán los datos para escribir en la memoria de programa *flash*. Al final, los datos quedarán desde la posición (07A0h) en la memoria de programa *flash*; lo que se intenta hacer está ilustrado en la Figura 16.20.

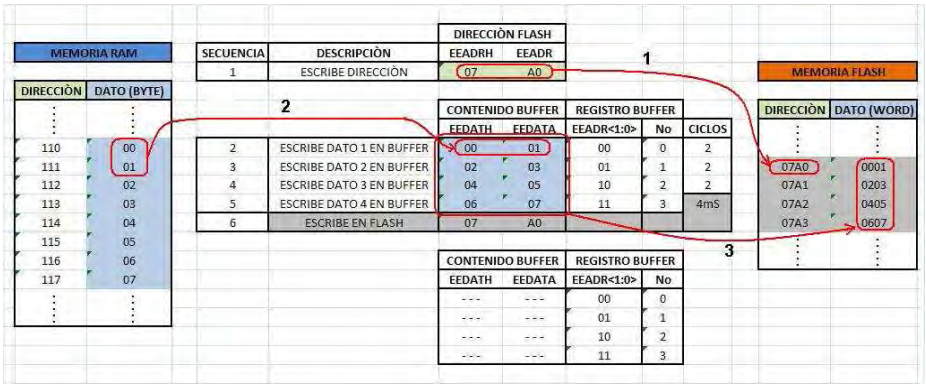


Figura 16.20 Escritura memoria *flash*.

El código del ejemplo se encuentra en nuestra página Web, en la carpeta: X:\Ejemplos_ASM que se debe copiar al: C:\, para encontrarlo como: C:\Ejemplos_ASM\FLASH_1\FLASH_1B.asm.

A continuación damos el código del ejemplo.

```
            list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc>             ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

; datos de retardos
            cblock 0x20             ; desde la posición 20h en RAM
delay1 ; retardo 1 BAJO nivel
delay2 ; retardo 2 MEDIO nivel
            endc                    ; fin de registros

#define    _kte_delay2              h'20'       ; retardo para 5 ms
; datos de flash
            cblock 110h             ; desde la posición 110h en RAM
                                    ; es el mismo Banco 2, donde están los registros usados
datos_flash             :0          ; define principio de datos flash en RAM
                                    ; (comparte posición con dato_flash_L_1)
                                    ; los datos aquí guardados se escribirán en la flash
dato_flash_L_1                      ; dato 1 low a flash
dato_flash_H_1                      ; dato 1 high a flash
dato_flash_L_2                      ; dato 2 low a flash
dato_flash_H_2                      ; dato 2 high a flash
dato_flash_L_3                      ; dato 3 low a flash
dato_flash_H_3                      ; dato 3 high a flash
dato_flash_L_4                      ; dato 4 low a flash
dato_flash_H_4                      ; dato 4 high a flash
```

```
data_EE_L                       ; dato low memoria flash
data_EE_H                       ; dato high memoria flash

address_EE_L                    ; dirección low memoria flash
address_EE_H                    ; dirección high memoria flash

contador                        ; contador
        endc                    ; fin de registros

; la dirección del dato a leer se cambia en las 2 siguientes instrucciones
#define  _address_EE_L          0xA0            ; dirección low memoria flash
#define  _address_EE_H          0x07            ; dirección high memoria flash

#define _bit_3                  3       ; bit 3, conteo hasta 8

 org 0
        goto    start           ; vaya a comienzo de programa
;-------------------------------------------------------------------
Delay
; retardo de 5ms
        clrf     delay1         ; Retardo 1 de BAJO nivel
        movlw   _kte_delay2     ; carga valor a
        movwf    delay2         ; Retardo 2 de MEDIO nivel

Loop:
        decfsz delay1,f ; decremente Retardo de BAJO nivel
        goto Loop ; hay 3 ciclos por loop * 256 loops = 768 instrucciones
        decfsz delay2,f ; decremente Retardo de MEDIO nivel
        goto Loop ; hay por loop (768 + 3) * 256 ciclos = 197.376 ciclos

        return
;-------------------------------------------------------------------
;inicializa valores de los datos_flash (110h en RAM) con 00 01 02..07 en la
Ini_var
        BANKSEL datos_flash     ; cambie a Banco 2 (datos_flash)

        clrf    contador        ; borre contador
        movlw   datos_flash     ; principio de datos flash
        movwf   FSR             ; cárguelo al apuntador FSR
        bsf     STATUS,IRP      ; pone Banco 2 y 3 para FSR

Ini_loop:
        movf    contador,W      ; contador -> W
        movwf   INDF            ; guarde indirecto
        incf    FSR,F           ; FSR +1
        incf    contador,W      ; contador + 1 -> W

        movwf   contador        ; W -> contador
        movwf   INDF            ; guarde indirecto
        incf    FSR,F           ; FSR +1
        incf    contador,F      ; contador + 1

        btfss   contador,_bit_3 ; contador = 8? sí, salte y termine
        goto    Ini_loop        ; no, continúe
```

```
            bcf     STATUS,RP1      ; Banco 0
            bcf     STATUS,IRP      ; pone Banco 0 y 1 para FSR

            return
;-----------------------------------------------------------------
; Escribe datos de EEPROM
; escribe 4 datos desde la posición (_address_EE_H:_address_EE_L) en la Memoria flash
; los datos están en datos_flash que se direccionan con FSR
Write_flash
            bsf     STATUS,RP1      ; RP1 = 1
            bcf     STATUS,RP0      ; RP0 = 1, Banco 2

            movf    address_EE_L,W  ; cargue dirección low -> W
            movwf   EEADR           ; guarde en registro de dirección low de memoria flash

            movf    address_EE_H,W  ; cargue dirección high -> W
            movwf   EEADRH          ; guarde en registro de dirección high de memoria flash

            movlw   datos_flash     ; principio de datos flash
            movwf   FSR             ; cárguelo al apuntador FSR
            bsf     STATUS,IRP      ; pone bancos 2 y 3 para FSR

Write_loop:
            movf    INDF,W          ; cargue indirecto -> W
            movwf   EEDATH          ; guarde en registro dato high memoria flash
            incf    FSR,F           ; FSR + 1

            movf    INDF,W          ; cargue indirecto -> W
            movwf   EEDATA          ; guarde en registro dato low memoria flash
            incf    FSR,F           ; FSR + 1

            bsf     STATUS,RP0      ; Banco 3
            bsf     EECON1,EEPGD    ; apunta a memoria de programa flash
            bsf     EECON1,WREN     ; habilita escritura

            movlw   0x55            ; secuencia obligada
            movwf   EECON2          ; escribe 55h
            movlw   0xAA            ; secuencia obligada
            movwf   EECON2          ; escribe AAh

            bsf     EECON1,WR       ; pone bit WR para empezar escritura
            nop                     ; ciclo de espera necesario
            nop                     ; ciclo de espera necesario
            bcf     EECON1,WREN     ; des-habilita escritura

            bcf     STATUS,RP0      ; RP0 = 0, Banco 2

            movf    EEADR,W         ; cargue EEADR -> W
            andlw   0x03            ; filtra bits <1:0>
            Xorlw   0x03            ; compare con 3
            btfsc   STATUS,Z        ; Z = 0? no, son iguales, salte y continúe
            goto    Write_end       ; sí, termine

            incf    EEADR,F         ; EEADR + 1
            goto    Write_loop      ; no, continúe
```

```
Write_end
        bcf     STATUS,RP1      ; RP1 = 0, Banco 0
        bcf     STATUS,IRP      ; pone Bancos 0 y 1 para FSR

        return

;-----------------------------------------------------------
Start:
        call    Ini_var         ; inicializa registros de datos_flash

        BANKSEL address_EE_L    ; cambie banco a address_EE_L

        movlw   _address_EE_L   ; cargue dirección -> W
        movwf   address_EE_L    ; dirección low memoria flash

        movlw   _address_EE_H   ; cargue dirección high -> W
        movwf   address_EE_H    ; dirección high memoria flash

        bcf     STATUS, RP1     ; Banco 0

        call    Write_flash     ; escribe en flash
        call    Delay           ; retardo de 5 ms

        goto    $
;-----------------------------------------------
        org     0x07A0          ; define posición de memoria de programa flash
        dw      0x3FFF,0x3FFF,0x3FFF,0x3FFF,0x3FFF,0x3FFF,0x3FFF,0x3FFF
        end
;-----------------------------------------------------------
```

Para realizar el seguimiento del programa anterior necesitamos abrir las ventanas **Watch** y **Program Memory**. La forma de abrir estas ventanas y agregar los contenidos se explica en el ejercicio anterior, **Ejemplo de lectura**.

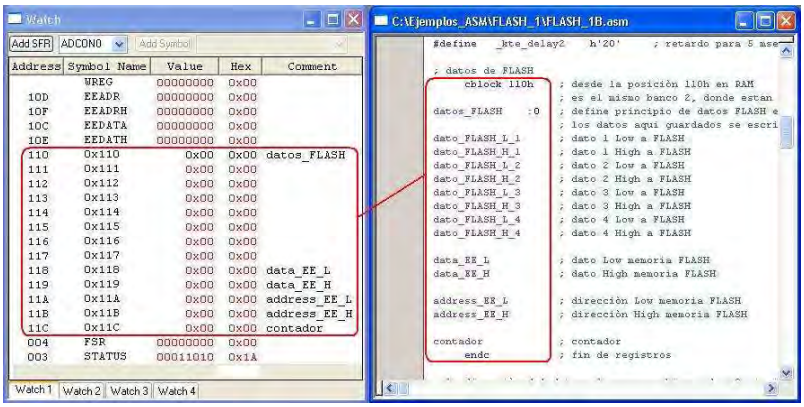


Figura 16.21 Depuración escritura memoria *flash 1*.

En la Figura 16.21 vemos que las variables definidas en el programa están agregadas a la ventana **Watch;** observemos que sus valores están en 0.

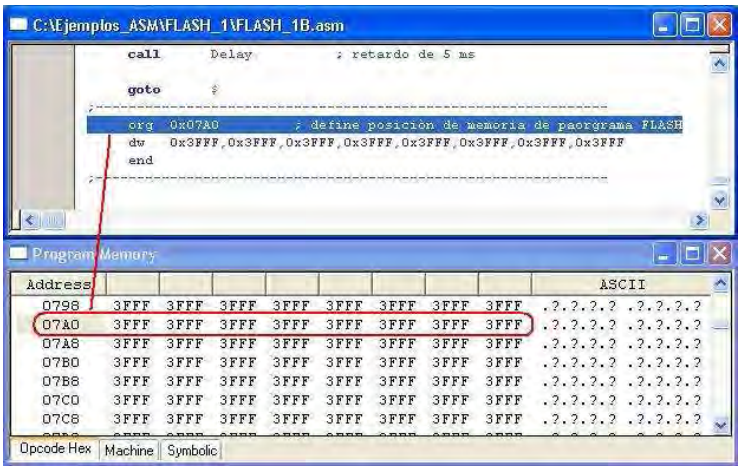


Figura 16.21 Depuración escritura memoria *flash 2*.

En la Figura 16.22, observe cómo las instrucciones org 0x07A0 y dw 0x3FF, 0x3FF—0x3FF limpian las posiciones de memoria (**07A0h a 07A7**), en las cuales escribiremos los 4 datos *word*.

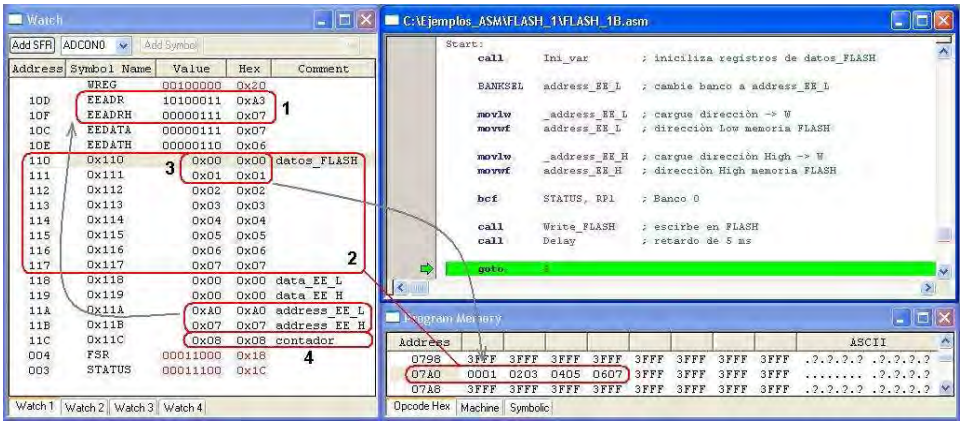


Figura 16.22 Depuración escritura memoria *flash 3*.

En la Figura 16.22 encontramos la imagen del último paso del programa; allí hay varios ítems (**1, 2, 3, 4**) de numeración que detallamos a continuación.

- Ítem 1

Observe que la dirección original (**07A0h**) desde donde se escribirán los 4 datos está en los registros (**address_EE_H: address_EE_L**), y la dirección final (**07A3h**) se encuentra en los registros (**EEADRH:EEADR**); ésta es la dirección del último dato que se va a escribir.

- Ítem 2

Observe que los datos de la memoria RAM (**110h** a **117h**) tienen los contenidos (**00,01,02...07**) y que éstos tomados de 2 en 2 forman el dato *word* que se escribe en la memoria de programa *flash*, desde la posición (**07A0h**).

- Ítem 3

El par de datos (00,01) en la posición (110h y 111h) son convertidos al dato *word* (**0001h**), que está ubicado en la dirección (07A0h), y así sucesivamente se toman de a 2 bytes y se copian consecutivamente hasta 4 datos *word*.

- Ítem 4

El registro contador está en el valor 8, que son los valores que se generan en las posiciones RAM (110h a 117h). De esta manera podemos apreciar que los datos son escritos en la memoria de programa *flash*.

```
NOTA: Para actualizar los contenidos de la ventana de Program
Memory se debe usar el ícono  en la barra de herramientas.
```

Un código de la misma aplicación con un ejemplo similar se encuentra en la Web, en la carpeta: X:\Ejemplos_ASM que se debe copiar al: C:\, para encontrarlo como: C:\Ejemplos_ASM\FLASH_1\FLASH_1C.asm.

EVALUACIÓN

```
P.: 1. ¿Cuántos bytes tiene 1 página (Page) en el MCU 16F877A y el
16F887?
R.:
2 048, Page desde (0000h a 07FFh).

P.: 2. ¿Cuáles son las páginas (Page) y sus direcciones?
R.:
Page 0              0000h a 07FFh, 2 048 posiciones de memoria
Page 1              0800h a 0FFFh, 2 048 posiciones de memoria
Page 2              1000h a 17FFh, 2 048 posiciones de memoria
Page 3              1800h a 1FFFh, 2 048 posiciones de memoria
```


libros.ep-electropc.com

```
P.: 3. ¿Cuántos bits debe tener el PC contador de programa para
direccionar 8K de memoria de programa?

R.:
PC<12..0> (13 bits).

P.: 4. ¿Cuántos bits tiene una instrucción en el MCU 16F877A y
16F887?
R.:
14 bits.

P.: 5. ¿Cuáles son las posiciones de los vectores de Reset e
Interrupción?
R.:
Reset = 0000h, Interrupción = 0004h.

P.: 6. ¿Cuáles son los registros que conforman el registro PC y sus
bits?
R.:
PCH<4..0> (5 BITS), PCL<7..0> (8 BITS), alterno PCLATH<4..0> (5
BITS).

P.: 7. ¿Cuáles son las 4 tipos de situaciones al cargar el contador
de programa PC?
R.:
Situación 1: Instrucciones con PCL como destino.
Situación 2: Instrucción GOTO.
Situación 3: Instrucción CALL.
Situación 4: Instrucciones RETURN, RETFIE, o RETLW.

P.:8. ¿Cómo se protege el código de programa?
R.:
Con el bit protección de programa CP habilitado.
Y con los bits WRT<1:0> se puede seleccionar cuál área de Memoria
de Programa o flash será protegida para escritura.

P.:9. ¿Es posible escribir y leer datos en la Memoria de Programa o
flash?
R.: Sí.

P.: 10. ¿Cuáles son los registros necesarios para escribir datos en
la Memoria de Programa o flash?
R.:
EECON1 registro de control 1
EEDATA registro de datos low (bajo)
EEDATH registro de datos high (alto)
EEADR registro de dirección low (bajo)
```

```
EEADRH registro de dirección high (alto)
```

17.0 MEMORIA EEPROM

La **memoria EEPROM** es una memoria de datos, pese a que está en posiciones de memoria de programa; en ella los datos son un bloque de memoria aparte; se encuentran ubicados a partir de la dirección **2100h**, y se accede a ellos mediante una secuencia especial de instrucciones.

17.1 DIAGRAMA CIRCUITAL

Más que un diagrama circuital (Figura 17.1), este diagrama es una representación de las posiciones de la memoria EEPROM.

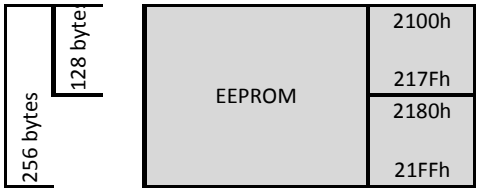


Figura 17.1 Memoria EEPROM.

Conviene anotar que estas posiciones son las direcciones lógicas mapeadas para el dispositivo; con ellas se accede solamente mediante procedimiento, no en forma directa o indirecta. Es decir, si trata de acceder a la posición **2100h** no encontrará la primera posición de memoria de la **EEPROM**.

17.2 FUNCIONAMIENTO

La **memoria EEPROM** es escribible y leíble durante una operación en modo *Run*; se direcciona mediante procedimiento (más adelante se explica) con los registros SFR siguientes (Tabla 17.1).

Tabla 17.1 Registros usados en la EEPROM.

REGISTRO SFR	DESCRIPCIÒM	DIRECCIÒN	BANCO	[RP1:RP0]
EECON1	Control	18Ch	3	11
EECON2	Temporal	18Dh	3	11
EEDATA	Dato	10Ch	2	10
EEADR	Direcciòn	10Dh	2	10

El registro **EEDATA** contiene el dato de 8 bits de lectura o de escritura.
El registro **EEADR** contiene la dirección de 8 bits que está siendo accedida, lo que nos da un acceso de hasta 256 posiciones de datos **EEPROM** (Tabla 17.2).
El registro **EECON1** contiene los bits de control.
El registro **EECON2** es el registro para inicializar la escritura (no es un registro físico y su lectura será de **00h**).

Tabla 17.2 Tamaño EEPROM y rango de direccionamiento.

Data EEPROM Size (1)	Address Range
64	0h - 3Fh
128	0h - 7Fh
256	0h - FFh

Note 1: Presently, devices are only offered with 64 bytes of Data EEPROM.

Se pueden encontrar memorias desde 64 bytes hasta 256 bytes, el rango de direccionamiento se hace con 8 bits. La cantidad de memoria (Tabla 17.3) depende del microcontrolador.

Tabla 17.3 Tamaño EEPROM según referencia de microcontrolador.

Device	# of Bytes
PIC16F873A	128
PIC16F874A	128
PIC16F876A	256
PIC16F877A	256

Los contenidos de la memoria EEPROM tienen la capacidad de estar insertados en el archivo *** .hex**. El dispositivo programador debe ser capaz de leer los datos EEPROM del archivo *** .hex**, y escribir los datos en la EEPROM del MCU, junto con la memoria de programa y los bits de configuración. Los datos de la EEPROM permiten ser leídos y escritos; para que un byte sea escrito automáticamente, primero se borra la posición y luego se escribe el dato nuevo. La memoria EEPROM tiene más **ciclos de borrado y escritura** (*Endurance*) que las posiciones de memoria **FLASH**; es decir, en la memoria EEPROM se pueden grabar datos más ciclos que los ciclos de programación que tiene la memoria *flash*.

```
NOTA: 100 000 ciclos de escritura/borrado para memoria de
programa flash.
```

```
1 000 000 ciclos de escritura/borrado para memoria de datos
EEPROM.

Retención de datos EEPROM > 40 años.
```

En este libro, consulte la **Parte 2: Arquitectura, Capítulo 6.0: Arquitectura del PIC16F887, Sección 6.1: Características (*Brochure*)**. Se recuerda que para la escritura de datos en la EEPROM se tiene un temporizador que puede variar de chip a chip; por tanto, es necesario que se consulten las especificaciones para los límites (tiempo típico de 4-8 ms).

Tabla 17.4 Especificaciones de tiempos de escritura, memorias EEPROM y *flash*.

Param No.	Sym	Characteristic	Min	Typ†	Max	Units	Conditions
		Data EEPROM Memory					
D120	ED	Endurance	100K	1M	—	E/W	-40°C to +85°C
D121	VDRW	VDD for read/write	VMIN	—	5.5	V	Using EECON to read/write, VMIN = min. operating voltage
D122	TDEW	Erase/write cycle time	—	4	8	ms	
		Program Flash Memory					
D130	EP	Endurance	10K	100K	—	E/W	-40°C to +85°C
D131	VPR	VDD for read	VMIN	—	5.5	V	VMIN = min. operating voltage
D132A		VDD for erase/write	VMIN	—	5.5	V	Using EECON to read/write, VMIN = min. operating voltage
D133	TPEW	Erase/Write cycle time	—	4	8	ms	

17.2.1 PROTECCIÓN CPD

Cuando el micro tiene código de protección de datos CPD habilitado (en el registro de configuración, visto antes), la CPU puede leer y escribir datos en la EEPROM. Todos los accesos externos son deshabilitados. Este bit es sólo para protección del código en la memoria de programa.

17.2.2. LEYENDO DATOS EN EEPROM

Para leer un dato de una posición de memoria, el usuario debe escribir la dirección en el registro EEADR, aclarar el bit EEPGD del registro EECON1<7>, el cual apunta a la EEPROM y entonces poner a uno el bit de control RD en el registro EECON1<0>. El dato estará disponible en el ciclo siguiente de instrucción, en el registro EEDATA; por lo tanto, puede ser leído con la instrucción siguiente. EEDATA mantiene este valor hasta otra lectura o hasta que sea escrito por el usuario (durante la operación de escritura).

Ejemplo de código para operación de lectura:

```
bsf     STATUS, RP1     ; RP1 = 1
bcf     STATUS, RP0     ; RP0 = 0, Banco 2
movlw   CONFIG_ADDR     ; cualquier dirección en EEPROM
                        ; este dato puede venir desde w

movwf   EEADR           ; dirección a leer

bsf     STATUS, RP0     ; Banco 3
bcf     EECON1, EEPGD   ; apunte a EEPROM
bsf     EECON1, RD      ; EE Read
bcf     STATUS, RP0     ; Banco 2
movf    EEDATA, W       ; W = EEDATA

bcf     STATUS, RP1     ; Bank 0, precaución volver a banco 0
```

17.2.3 ESCRIBIENDO DATOS EN EEPROM

Para escribir un dato en la posición EEPROM, el usuario debe primero escribir la dirección en el registro EEADR y el dato en el registro EEDATA. Luego debe seguir una secuencia específica para iniciar la escritura de cada byte. La escritura no inicia si la secuencia no es exactamente seguida:

1. Escribir **55h** en registro EECON2
2. Escribir **AAh** en registro EECON2
3. Setear bit **WR** en registro EECON1

```
        bsf     STATUS, RP0     ; Banco 1
        bcf     INTCON, GIE     ; deshabilite las interrupciones
        bsf     EECON1, WREN    ; habilite escritura

;secuencia requerida
        movlw   55h             ;
        movwf   EECON2          ; 55h debe ser escrito en EECON2

        movlw   AAh             ; para empezar secuencia
        movwf   EECON2          ; escriba AAh en EECON2

        bsf     EECON1,WR       ; ponga bit WR, comienza escritura

; interrupciones
;       bsf     INTCON, GIE     ; habilite las interrupciones.(¡esto es opcional!)

;Fin de secuencia
```

```
NOTA: Para cada byte que se desea escribir, se recomienda
deshabilitar las interrupciones durante este segmento de código.
```


libros.ep-electropc.com

Además, el bit WREN en el registro EECON1 debe ser puesto a uno para habilitar la escritura. Este mecanismo previene escrituras accidentales de datos a la EEPROM, debido a ejecución de código errado o no esperado (por ejemplo, programa perdido). El usuario debe mantener el bit WREN borrado todo el tiempo, excepto cuando actualiza la EEPROM. El bit WREN no es borrado por el hardware.

Después de que la secuencia de escritura ha iniciado, borrar el bit WREN no afectará el ciclo de escritura. El bit **WR** será inhibido de ser puesto a uno, a no ser que el bit WREN sea puesto en uno. Cuando se completa el ciclo de escritura, el bit **WR** es borrado en el hardware y el bit **EEIF** (*EE Write Complete Interrupt Flag*) es puesto a uno. El usuario puede habilitar esta interrupción o hacer *pooling* (preguntar o indagar por); este bit EEIF debe ser borrado por el software.

17.3 REGISTROS ASOCIADOS

Los registros asociados con el manejo de la EEPROM son los que se muestran en las Tablas 17.5, 17.6 y 17.7.

Tabla 17.5 Registros asociados con el manejo de la EEPROM.

R/W-x	U-0	U-0	U-0	R/W-x	R/W-0	R/S-0	R/S-0
EEPGD	—	—	—	WRERR	WREN	WR	RD
bit 7							bit 0

bit 7 **EEPGD**: Program/Data EEPROM Select bit
1 = Accesses program memory
0 = Accesses data memory
Reads '0' after a POR; this bit cannot be changed while a write operation is in progress.

bit 6-4 **Unimplemented:** Read as '0'

bit 3 **WRERR:** EEPROM Error Flag bit
1 = A write operation is prematurely terminated (any $\overline{\text{MCLR}}$ or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN:** EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read; RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

Tabla 17.6 Registros/bits asociados con el manejo de las memorias EEPROM y *flash*.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on Reset	Value on all other Resets
10Ch	EEDATA	EEPROM/Flash Data Register Low Byte								xxxx xxxx	uuuu uuuu
10Dh	EEADR	EEPROM/Flash Address Register Low Byte								xxxx xxxx	uuuu uuuu
10Eh	EEDATH	—	—	EEPROM/Flash Data Register High Byte						xxxx xxxx	---- uuuu
10Fh	EEADRH	—	—	—	EEPROM/Flash Address Register High Byte					xxxx xxxx	---- ----
18Ch	EECON1	EEPGD	—	—	—	WRERR	WREN	WR	RD	x--- x000	---- u000
18Dh	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	---- ----
0Dh	PIR2	—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF	-0-0 0--0	-0-0 0--0
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	-0-0 0--0

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends upon condition
Shaded cells are not used by data EEPROM or Flash program memory.

Tabla 17.7 Registros asociados con el manejo de la EEPROM.

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

17.4 PROCEDIMIENTO DE USO

Hay dos maneras de usar la EEPROM: en lectura y en escritura.

EN MODO DE LECTURA

Para leer datos en la EEPROM siga los pasos siguientes:

1. Escriba la dirección en el registro EEADR.
2. Aclare en bit EEPGD en el registro EECON1<7>, para apuntar a la EEPROM.
3. Ponga a '1' el bit RD en el registro EECON1<0>, para comenzar la lectura.
4. Lea el dato en el registro EEDATA.

A continuación siga los mismos pasos con código:

1. Escriba la dirección en el registro EEADR:

```
        bsf     STATUS, RP1     ; RP1 = 1
        bcf     STATUS, RP0     ; RP0 = 0, Banco 2
        movlw   CONFIG_ADDR     ; cualquier dirección en Datos EEPROM
        movwf   EEADR           ; registro de dirección EEPROM que se va a leer
```

2. Aclare en bit EEPGD en el registro EECON1<7>, para apuntar a la EEPROM:

```
        bsf     STATUS, RP0     ; banco 3
        bcf     EECON1, EEPGD   ; apunte a Datos EEPROM
```

3. Ponga a '1' el bit RD en el registro EECON1<0>, para comenzar lectura:

```
        bsf     EECON1, RD      ; habilite bit de lectura
```

4. Lea el dato en el registro EEDATA:

```
        bcf     STATUS, RP0     ; Bank 2
        movf    EEDATA, W       ; W <- EEDATA

        bcf     STATUS, RP1     ; Banco 0
```

Se hace la subrutina **Read_EEprom**, que lee un dato de la EEPROM. Como entrada en el registro W contiene la dirección del dato que se va a leer de la EEPROM. Como salida en el registro W contiene el dato leído de la EEPROM. Para este procedimiento el código es:

```
; Lee datos de EEPROM,
; entrada dirección para EEADR en W,
; salida dato de EEDATA en W

Read_EEprom
        bsf     STATUS, RP1     ; RP1 = 1
        bcf     STATUS, RP0     ; RP0 = 0, Banco 2
;       movlw   CONFIG_ADDR     ; cualquier dirección en Datos EEPROM
                                ; este dato viene como entrada en el registro W
        movwf   EEADR           ; registro de dirección EEPROM que se va a leer

        bsf     STATUS, RP0     ; banco 3
        bcf     EECON1, EEPGD   ; apunte a Datos EEPROM
        bsf     EECON1, RD      ; habilite bit de lectura
        bcf     STATUS, RP0     ; Bank 2
        movf    EEDATA, W       ; W <- EEDATA

        bcf     STATUS, RP1     ; Banco 0

        return
;-----------------------------------------------------------------
```

El código siguiente carga la dirección **contador** en el registro W (entrada), llama a la subrutina **Read_EEprom** y pone el dato del registro W (salida) en el **PORTD**. La forma de usarlo es:

```
movf    contador,W              ; cargue contador (dirección que se va a leer ) a W
        call    Read_EEprom             ; lea EEPROM
        movwf   PORTD                   ; ponga dato leído en PUERTO D
```


libros.ep-electropc.com

EN MODO DE ESCRITURA

Para escribir datos en la memoria EEPROM los pasos son los siguientes:

1. Si el paso 10 no está implementado, verifique (*check*) el bit **WR** para ver si una escritura está en progreso.
2. Escriba la dirección en EEADR.
3. Escriba el dato en EEDATA.
4. Borre el bit EEPGD para apuntar a la EEPROM.
5. Ponga a uno el bit WREN para permitir escritura.
6. Deshabilite interrupciones (si están habilitadas).
7. Ejecute los 5 pasos de la secuencia obligatoria.
   Escriba 55h en registro EECON2, en dos pasos a W y a EECON2.
   Escriba AAh en registro EECON2, en dos pasos a W y a EECON2.
   Ponga a uno el bit **WR**.
8. Habilite interrupciones (si está usando).
9. Borre el bit **WREN** para deshabilitar escrituras.
10. Al completar el ciclo de escritura, el bit **WR** es borrado '0' y el bit de interrupción EEIF es puesto a uno. (EEIF debe ser borrado por firmware.) Si el paso 1 no es implementado, el firmware debe verificar que el bit EEIF sea puesto a '1' o el bit **WR** borrado '0', para indicar el fin del ciclo de escritura.

Los mismos pasos anteriores con código se ven a continuación:

- Si el paso 10 no está implementado, verifique (*check*) el bit **WR** para ver si una escritura está en progreso:

```
Write_EEprom:
        bsf     STATUS,RP1      ; RP1 = 1
        bsf     STATUS,RP0      ; RP0 = 1, Banco 3
        btfsc   EECON1,WR       ; espera a que escritura
        goto    $-1             ; se complete
```

- Escriba la dirección en EEADR:

```
        bcf     STATUS, RP0     ; Banco 2
        movf    EE_ADR,W        ; dirección -> W
        movwf   EEADR           ; W -> EEADR
```

- Escriba el dato en EEDATA:

```
        movf    EE_DATA,W       ; dato -> W
        movwf   EEDATA          ; W -> EEADR
```

- Borre el bit EEPGD para apuntar a la EEPROM:

libros.ep-electropc.com

```
        bsf     STATUS,RP0      ; Banco 3
        bcf     EECON1,EEPGD    ; apunta a EEPROM
```

- Ponga a uno el bit **WREN** para permitir escritura:

```
        bsf     EECON1,WREN     ; habilita escritura
```

- Deshabilite interrupciones (si están habilitadas):

```
        bcf     INTCON,GIE      ; deshabilita Interrupciones
```

- Ejecute los 5 pasos de la secuencia obligatoria.
  Escriba 55h en el registro EECON2, en dos pasos a W y a EECON2.
  Escriba AAh en el registro EECON2, en dos pasos a W y a EECON2.
  Ponga a uno el bit **WR**:

```
        movlw   0x55            ;
        movwf   EECON2          ; escribe 55h
        movlw   0xAA            ;
        movwf   EECON2          ; escribe AAh
        bsf     EECON1,WR       ; pone bit WR para empezar escritura
```

- Habilite las interrupciones (si está usándolas):

```
        bsf     INTCON,GIE      ; hablita INTs.
```

- Borre el bit **WREN** para deshabilitar las escrituras:

```
        bcf     EECON1,WREN     ; deshabilita escritura

        bcf     STATUS,RP1      ; RP1 = 0
        bcf     STATUS,RP0      ; RP0 = 0, Banco 0
```

- Al completar el ciclo de escritura, el bit **WR** es borrado '0' y el bit de interrupción EEIF es puesto a uno. (EEIF debe ser borrado por *firmware.*). Si el paso 1 no es implementado, el *firmware* debe verificar (*check*) que el bit EEIF sea puesto a '1' o el bit **WR** borrado '0', para indicar el fin del ciclo de escritura.

Se hace la subrutina **Write_EEPROM**, que escribe un dato en la EEPROM. El registro **EE_DATA (0xB7h)** es del dato y el registro **EE_ADR (0x07h)** es la dirección; recuerde que estas posiciones no dependen del banco en que esté. Este programa revisa inicialmente la bandera **WR** si está en 0. El código para este procedimiento es:

```
; Escribe datos de EEPROM
; en EE_ADR la dirección que se va a escribir
; en EE_DATA el dato que se va a escribir
```


libros.ep-electropc.com

```
Write_EEprom:
        bsf     STATUS,RP1      ; RP1 = 1
        bsf     STATUS,RP0      ; RP0 = 1, Banco 3
        btfsc   EECON1,WR       ; espera a que escritura
        goto    $-1             ; se complete

        bcf     STATUS, RP0     ; Banco 2
        movf    EE_ADR,W        ; dirección -> W
        movwf   EEADR           ; W -> EEADR
        movf    EE_DATA,W       ; dato -> W
        movwf   EEDATA          ; W -> EEADR

        bsf     STATUS,RP0      ; Banco 3
        bcf     EECON1,EEPGD    ; apunta a la EEPROM
        bsf     EECON1,WREN     ; habilita escritura
        bcf     INTCON,GIE      ; deshabilita las interrupciones

        movlw   0x55            ;
        movwf   EECON2          ; escribe 55h
        movlw   0xAA            ;
        movwf   EECON2          ; escribe AAh
        bsf     EECON1,WR       ; pone bit WR para empezar escritura
        bsf     INTCON,GIE      ; habilita INT
        bcf     EECON1,WREN     ; deshabilita escritura

        bcf     STATUS,RP1      ; RP1 = 0
        bcf     STATUS,RP0      ; RP0 = 0, Banco 0

        return
```

El código siguiente carga la dirección contador en **EE_ADR**, lee un dato de la **Tabla_code_7_seg** y lo pone en **EE_DATA**, luego llama a la rutina Write_EEprom para escribir el dato **EE_DATA** en la posición **EE_ADR** de la EEPROM.

La forma de usarlo es:.

```
        movf    contador,W              ; cargue contador a W (dirección)
        movwf   EE_ADR                  ; guarde en EE_ADR dirección en EEPROM
        call    Tabla_code_7_seg        ; llame a Tabla código de 7 segmentos, W = dirección
        movwf   EE_DATA                 ; guarde el dato de la tabla en EE_DATA,
                                        ; es dato el dato para EEPROM
                call    Write_EEprom    ; escribe en EEPROM
```

17.5 PRACTICA

Como ya hemos visto, hay dos formas de usar la memoria EEPROM: en lectura y en escritura; así que haremos 2 prácticas.

- EN MODO DE ESCRITURA
- EN MODO DE LECTURA

libros.ep-electropc.com

La una escribe y la otra lee un dato de la memoria EEPROM; a continuación se describe cada una.

EN MODO DE ESCRITURA

En seguida haremos una práctica de escritura de un dato en una dirección de la EEPROM; para el montaje de esta práctica únicamente se necesita el sistema de evaluación **PIC16F8XX** y, por supuesto, la herramienta **PICkit 2**, tal como se muestra en la Figura 17.2.



Figura 17.2 Práctica de montaje de la EEPROM.

El programa siguiente escribe un dato definido como _data_WR de valor 0xB7 en la dirección definida como _address_WR de valor 07h, en la EEPROM.

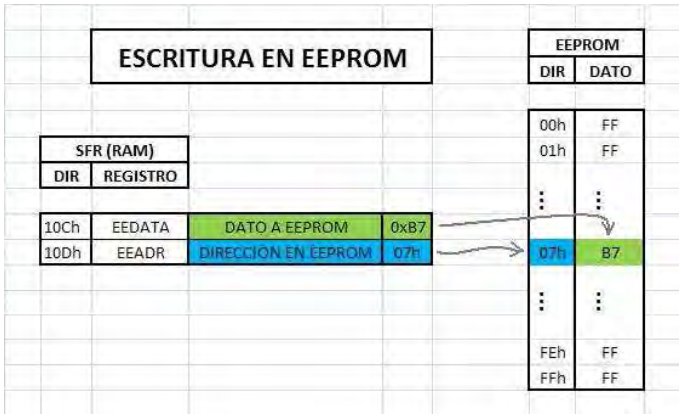


Figura 17.3 Diagrama de escritura de EEPROM.

Lo que se intenta con el programa siguiente está ilustrado en la Figura 17.3.

libros.ep-electropc.com

El código del ejemplo que se encuentra en nuestra página Web, en la carpeta: X:\Ejemplos_ASM y se debe copiar al C:\Ejemplos_ASM\EEPROM_1\EEPROM_1.asm.

```
; * * * * * * * * * * * * * * * * * * * C:\Ejemplos_ASM\EEPROM_1\EEPROM_1.asm * * * * * * * * * *
* * * * * * * * * * * * * * * * *

 list p = 16f877a      ; directiva para definir procesador
 #include <p16f877a.inc> ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

 cblock 0x20           ; desde la posición 20h en RAM
delay1     ; retardo 1 BAJO nivel
delay2     ; retardo 2 MEDIO nivel
 endc                  ; fin de registros

#define    _kte_delay2 h'20'    ; retardo para 5 ms

 cblock 0x71                   ; desde la posición 70h en RAM, estas posiciones
                               ; se accesan desde cualquier banco, sin conmutarlas
EE_DATA                ; dato para EEPROM
EE_ADR                 ; dirección para EEPROM
 endc                          ; fin de registros

#define    _           _data_WR0xB7       ; dato que se va a escribir en EEPROM
#define    _           _address_WR 7      ; dirección donde va el dato que se va a escribir en EEPROM

 org 0
            goto                start       ; vaya al comienzo de programa
;-----------------------------------------------------------------------
; Escribe datos de EEPROM
; en EE_ADR la dirección que se va a escribir
; en EE_DATA el dato que se va a escribir
Write_EEprom
            bsf         STATUS,RP1          ; RP1 = 1
            bsf         STATUS,RP0          ; RP0 = 1, Banco 3
            btfsc       EECON1,WR           ; espera a que escritura
            goto        $-1                           ; se complete

            bcf         STATUS, RP0         ; Banco 2
            movf        EE_ADR,W            ; dirección -> W
            movwf       EEADR               ; W -> EEADR
            movf        EE_DATA,W           ; dato -> W
            movwf       EEDATA              ; W -> EEADR

            bsf         STATUS,RP0          ; Banco 3
            bcf         EECON1,EEPGD        ; apunta a EEPROM
            bsf         EECON1,WREN         ; habilita escritura
            bcf         INTCON,GIE          ; deshabilita INT

            movlw       0x55                ;
            movwf       EECON2              ; escribe 55h
```


libros.ep-electropc.com

```
            movlw   0xAA                ;
            movwf   EECON2              ; escribe AAh
            bsf     EECON1,WR           ; pone bit WR para empezar escritura
            bsf     INTCON,GIE          ; hablita INT
            bcf     EECON1,WREN         ; deshabilita escritura

            bcf     STATUS,RP1          ; RP1 = 0
            bcf     STATUS,RP0          ; RP0 = 0, Banco 0

            return

;-----------------------------------------------------------------
Start:

            movlw   _address_WR         ; carga la dirección (0-256d) donde se escribe el dato
            movwf   EE_ADR              ; guarde en EE_ADR dirección en EEPROM

            movlw   _data_WR; carga el dato escribir
            movwf   EE_DATA             ; guarde en EE_DATA dato en EEPROM

            call    Write_EEprom        ; escribe en EEPROM

Delay
; retardo de 5ms
            clrf    delay1              ; Retardo 1 de BAJO nivel
            movlw   kte_delay2          ; carga valor a
            movwf   delay2              ; Retardo 2 de MEDIO nivel

Loop:
            decfsz delay1,f ;           decremente Retardo de BAJO nivel
            goto Loop                   ; hay 3 ciclos por loop * 256 loops = 768 instrucciones
            decfsz delay2,f             ; decremente Retardo de MEDIO nivel
            goto Loop                   ; hay por loop (768 + 3) * 256 ciclos = 197.376 ciclos

            goto $

 end
```

En el ejemplo anterior, si desea cambiar los valores de la dirección y el dato que se va a escribir en la memoria EEPROM, basta con cambiar los valores de las líneas:

```
#define   _         _data_WR        0xB7    ; dato que se va a escribir en EEPROM
#define   _         _address_WR     7       ; dirección donde va el dato que se va a escribir en
                                            ; EEPROM
```

Debemos abrir dos ventanas que podemos utilizar en el seguimiento de los datos y del tiempo de ejecución del programa: la ventana de datos EEPROM y la ventana de tiempo *Stopwatch*, tal como se muestra en la Figura 17.4.

libros.ep-electropc.com

Agregar ventana EEPROM

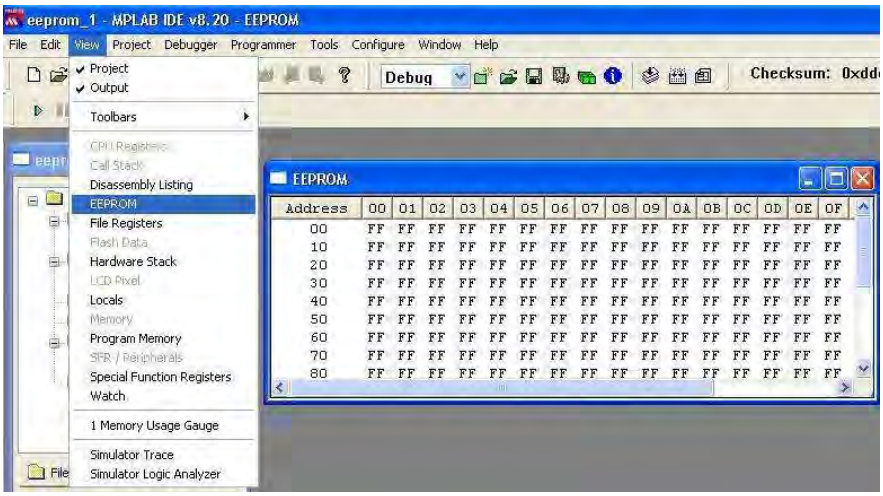


Figura 17.4 Ventana EEPROM.

Para abrir la ventana de datos de EEPROM, se debe seleccionar en el menú de opciones **View/EEPROM**; al dar un clic aparece la ventana EEPROM, tal como ilustra la Figura 17.4; en ella podemos verificar los datos contenidos en la EEPROM.

Agregar ventana Stopwatch

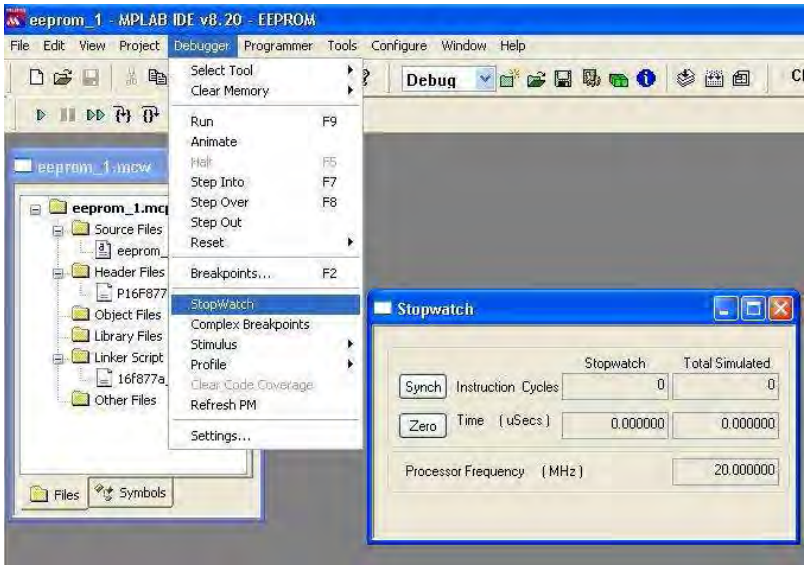


Figura 17.5 Ventana Stopwatch.

libros.ep-electropc.com

Para abrir la ventana de **Stopwatch** se debe seleccionar en el menú de opciones **Debugger/Stopwatch**; al dar un clic aparece la ventana **Stopwatch**, tal como ilustra la Figura 17.5. A esta ventana se accede solamente desde el modo de simulación, y se usa para medir el tiempo transcurrido desde el inicio del programa, o desde un punto en particular. Al correr paso a paso en modo de simulación, en este programa se puede ver cómo se escribe el dato 0□B7 en la posición 07h, y luego se hace un retardo de 5 ms, para esperar el tiempo necesario que toma el módulo de EEPROM para realizar la escritura. En modo paso a paso observe el estado inicial de la memoria EEPROM: todos sus valores están en FFh, que son los valores que indican que la memoria está borrada o que no tiene contenidos, tal como se muestra en la Figura 17.6.

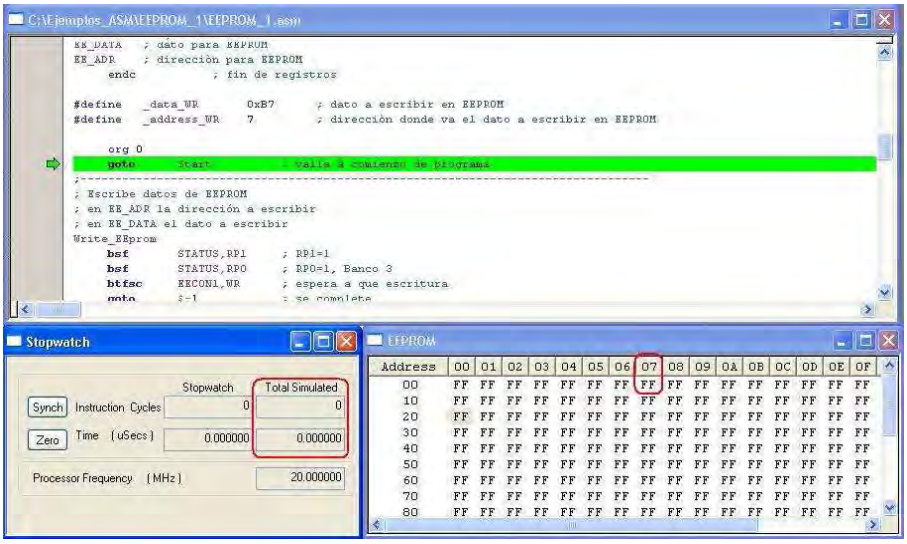


Figura 17.6 Modo simulación de escritura EEPROM 1.

En la Figura 17.7 podemos observar que en la ventana EEPROM el contenido de la memoria cambió en la dirección 07h con el dato B7, al llegar al último paso goto $, luego del retardo de 5 ms para permitir la escritura. En la ventana **Stopwatch** se aprecia el tiempo que transcurre hasta el último paso con el valor 4.948 ms, muy similar a 5 ms, que es el tiempo recomendado para esperar la escritura de los datos en la memoria EEPROM.

libros.ep-electropc.com

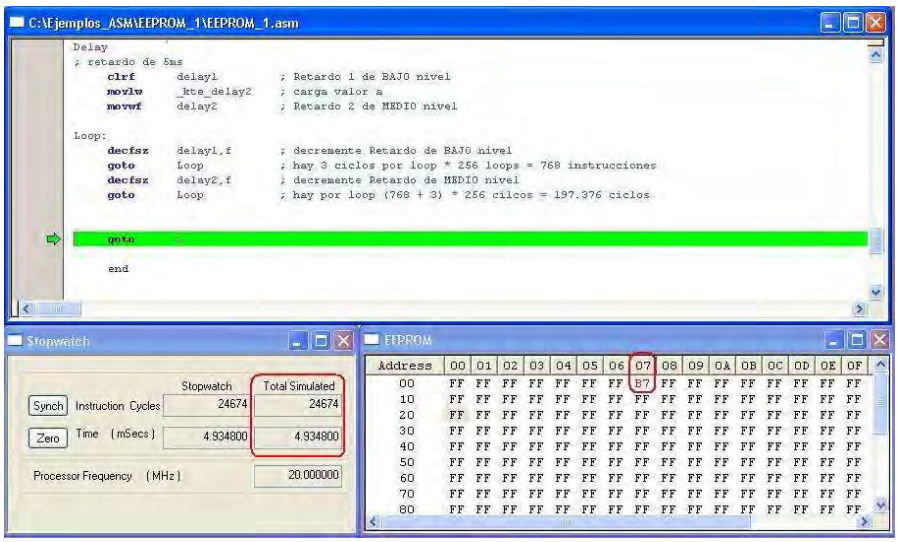


Figura 17.7 Modo simulación de escritura EEPROM 2.

Al programar el microcontrolador en la tarjeta **PIC16F8XX** podemos hacer la depuración o modo *Debug*. De igual manera, al ir a la última instrucción se obtiene el mismo resultado, como se observa en la Figura 17.8.

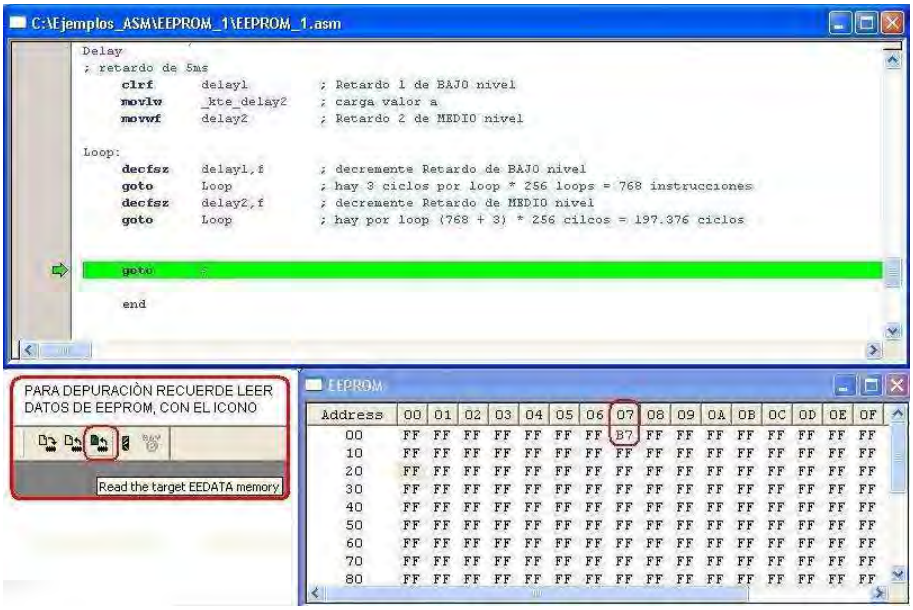


Figura 17.8 Modo depuración de escritura EEPROM.

libros.ep-electropc.com

Actualizar datos en la ventana EEPROM

Para actualizar los datos contenidos en la memoria EEPROM del microcontrolador, y que éstos se puedan ver actualizados en la ventana EEPROM del MPLAB, se debe utilizar el ícono  que aparece en la barra de herramientas, tal como muestra la Figura 17.9.



Figura 17.9 Barra de herramientas.

Al seleccionar el ícono aparece el cuadro de diálogo **Read the target EEDATA memory**, que traduce **Lea los EEDATOS (datos EEPROM) de la tarjeta destino**. Si no se realiza el paso anterior, los datos que vemos en la ventana EEPROM no corresponderán a los datos que contiene el microcontrolador en la memoria EEPROM.

El código del ejemplo con subrutinas y otras funciones que se encuentran en nuestra página Web, en la carpeta
X:\Ejemplos_ASM, se debe copiar al C: para abrirlo como
C:\Ejemplos_ASM\EEPROM_1\EEPROM_1A.asm.

EN MODO DE LECTURA

A continuación haremos una práctica de lectura de un dato en una dirección de la memoria EEPROM; para el montaje de esta práctica se necesita el sistema de evaluación **PIC16F8XX**, tal como se muestra en la Figura 17.10.



Figura 17.10 Práctica de montaje de la EEPROM.

libros.ep-electropc.com

El programa siguiente lee un dato de la memoria EEPROM en la dirección definida como _address_RD 0xC0; este dato, una vez leído, se guarda en el registro definido como dato_RD (Figura 17.11).

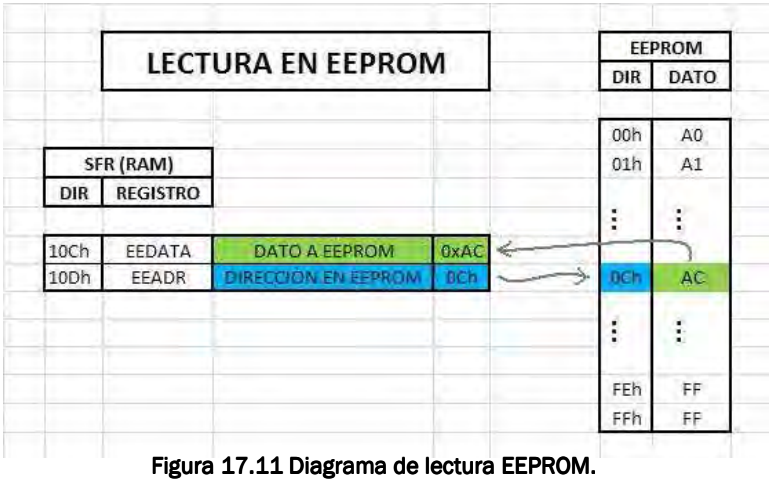


Figura 17.11 Diagrama de lectura EEPROM.

La idea de lo que se intenta realizar con el programa siguiente está ilustrada en la Figura 17.11. El código del ejemplo que se encuentra en nuestra página Web, en la carpeta:

X:\Ejemplos_ASM, se debe copiar al
C:\Ejemplos_ASM\EEPROM_1\EEPROM_1B.asm.

```
; * * * * * * * * * * * * * * * * * * * C:\Ejemplos_ASM\ EEPROM_1\EEPROM_1B.asm * * * * * * * * * * * * * * * * * * * * * * * * * *

            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>          ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

            cblock 0x20             ; desde la posición 20h en RAM
dato_RD                              ; dato leído en la memoria EEPROM
             endc                    ; fin de registros

#define     _address_RD             0x0C                    ; dirección que se va a leer en EEPROM

            org 0
            goto                    start                   ; vaya al comienzo de programa
;-----------------------------------------------------------------------
; Lee datos de EEPROM,
; entrada dirección para EEADR en W,
```


libros.ep-electropc.com

```
; salida dato de EEDATA en W
Read_EEprom
            bsf     STATUS, RP1         ; RP1 = 1
            bcf     STATUS, RP0         ; RP0 = 0, Bank 2
            movwf   EEADR               ; registro de dirección EEPROM que se va a leer
            bsf     STATUS, RP0         ; Bank 3
            bcf     EECON1, EEPGD       ; apunte a Datos EEPROM
            bsf     EECON1, RD          ; habilite bit de lectura
            bcf     STATUS, RP0         ; Bank 2
            movf    EEDATA, W           ; EEDATA -> W

            bcf     STATUS, RP1         ; Banco 0

            return
;-----------------------------------------------------------------
Start:
; lee un dato desde la posición _address_RD en la memoria EEPROM y lo guarda en dato_RD
            movlw   _address_RD         ; carga la dirección (0-256d) donde lee el dato

            call    Read_EEprom         ; lee en EEPROM
            movwf   dato_RD             ; guarde el dato leído en dato_RD

            goto    $

            org 0x2100              ; posiciones lógica mapeadas desde 00 en EEPROM
                                    ; inicia con 16 valores desde 0xA0..AF, desde la posición 00h-0Fh en
                                    ; EEPROM
de          0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF

                                    ; inicia los valores con FF en direcciones de 00 a 1F
;de        0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF

 end
```

Si desea cambiar el valor (en el ejemplo anterior) de la dirección del dato que se va a leer en la memoria EEPROM, basta con cambiar el valor 0x0C de la línea:

```
#define    _address_RD          0x0C                ; dirección que se va a leer en EEPROM
```

Siguiendo el programa anterior paso a paso en modo de simulación, detallaremos varios puntos importantes.

Paso 1

Primero hay que agregar las ventanas **EEPROM** y **Watch,** cuyos procedimientos ya hemos descrito. Para abrir la ventana **EEPROM** se sigue el procedimiento descrito en el ejemplo anterior "En modo de escritura", en el ítem "Agregar ventana EEPROM". Para abrir la ventana **Watch** se sigue el procedimiento descrito en la **Parte 3: Programación II, Capítulo 14,** en la **Sección 14.3: Simulación,** en el ítem**: Paso 2 Primer paso a paso,** explicación**: Agregar la ventana Watch y Registros SFR**. Con el procedimiento descrito podemos agregar a la ventana **Watch** los registros

libros.ep-electropc.com

Registro W, EEDATA y EEADR. Para agregar la variable (registros de usuario GPR o RAM) definida como:

```
dato_RD                         ; dato leído en la memoria EEPROM
```

En la ventana **Watch** seguimos el procedimiento descrito en la **Parte 3: Programación II**, en el **Capítulo 14.0, Subsección 14.5.5: Pasos para depuración ejemplo** en el ítem: **Paso 6 Agregar la ventana Watch y Registros GPR**. Ahora, con las ventanas **EEPROM** y **Watch** abiertas, con los registros que necesitamos ver agregados, y el programa cargado, podemos obtener una imagen como se ilustra en la Figura 17.12.

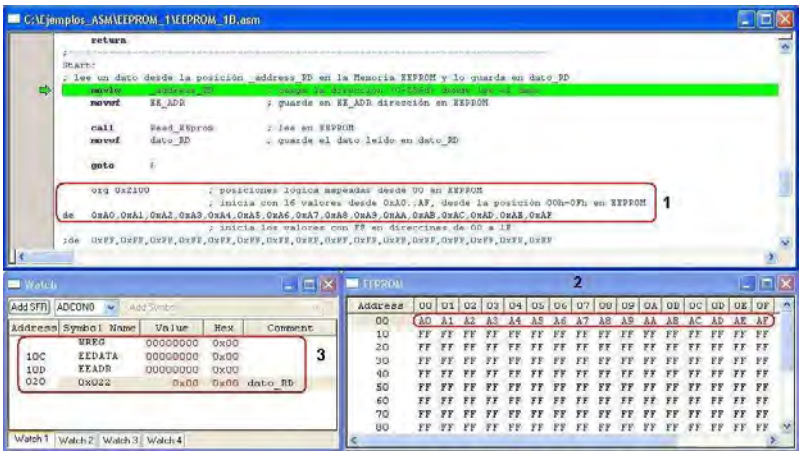


Figura 17.12 Modo simulación de lectura EEPROM 1.

Ahora describiremos los ítems de numeración **1, 2, 3** que se encuentran en la Figura 17.12 y que encierran contenidos para explicar.

- Item 1

En la ventana de código de programa, se definen los datos de la memoria EEPROM con la directiva **DE**, un paso antes se define desde dónde van a quedar con la directiva *org*.

```
NOTA: Recuerde que las posiciones de la memoria EEPROM están
mapeadas lógicamente entre 2100h a 21FFh, que corresponden a las
256 posiciones de la memoria EEPROM entre 00h a FFh.
```

En la línea siguiente se pone la dirección inicial de los datos que se definirán:

```
org         0x2100              ; posiciones lógicas mapeadas desde 00 en EEPROM
```


libros.ep-electropc.com

En la línea siguiente se definen los datos con valores (0xA0 a 0xAF) que se colocarán en la memoria EEPROM, quedarán en las posiciones (00h a 0Fh):

```
de          0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF
```

El MPLAB realiza esto en el modo simulador y se pueden ver en la ventana EEPROM.

NOTA: En los modos de programación y depuración los datos quedan en la memoria EEPROM del MCU y se pueden ver en la ventana EEPROM, actualizándolos con el ícono  , como se describió en el ejemplo anterior en "Actualizar datos en la ventana EEPROM)".

- Ítem 2

En la ventana **EEPROM** se muestran los valores definidos con la directiva **DE** que son los valores **(0xA0 a 0xAF)** en las posiciones **(00h a 0Fh)**, las otras posiciones de memoria están con los datos borrados (con valor FFh).

- Ítem 3

En la ventana **Watch** se pueden ver los registros para revisar; observe que todos están en valor inicial **0x00**.

Paso 2

A continuación, ejecutando la acción **Run to Cursor** sobre la línea de instrucción goto $, se termina el programa. Una vez terminado podemos ver el dato leído y los registros adicionales en la ventana **Watch**.

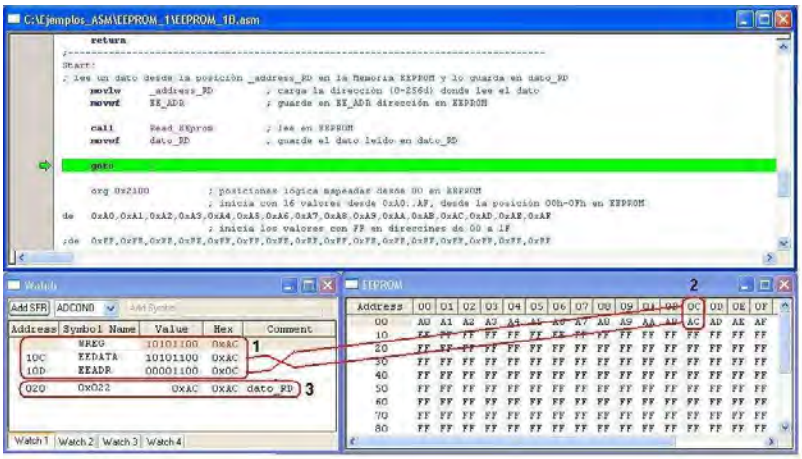


Figura 17.13 Modo simulación de lectura EEPROM 2.

libros.ep-electropc.com

Ahora describimos los ítems de numeración **1, 2 ,3** que se encuentran en la Figura 17.13 y encierran contenidos para explicar**.**

- Ítems 1 y 2

En la ventana **Watch** (**1**) en el registro EEADR = dirección EEPROM del dato (posición 10Dh) se lee el valor EEADR = 0x0C, que corresponde a la posición de memoria en la ventana EEPROM (**2**). En la misma ventana **Watch** (**1**) en el registro EEDATA = dato EEPROM (posición 10Ch) se lee el valor EEDATA = 0xAC que corresponde al dato de la posición de memoria en la ventana EEPROM (**2**).

- Ítem 3

En la ventana **Watch** (**1**) podemos verificar que el dato leído 0xAC se encuentra en la variable dato_RD (registro en la posición 020h).

El código del ejemplo con subrutinas y otras funciones que se encuentra en nuestra página Web, en la carpeta:

X:\Ejemplos_ASM y se debe copiar al C: para abrirlo como C:\Ejemplos_ASM\EEPROM_1\EEPROM_1C.asm.

18.0 MEMORIA RAM

La **memoria RAM** o **de datos o variable** son los datos que sí varían durante la ejecución del programa; está compuesta por:

- **Registros de funciones especiales** o *Special Function Registers (SFR)*
  - Aquí están todos los registros necesarios para el control de operación del micro.

- **Registros de propósito General** o *General Purpose Registers (GPR)*
  - Aquí está la memoria de usuario.

No consideramos los datos almacenados en la memoria EEPROM porque ya se estudiaron en el **Capítulo 17.0: Memoria EEPROM.**

18.1 MAPA DE REGISTROS

La memoria RAM está dispuesta entre los registros **000h a 1FFh**, o sea **512** bytes, los cuales están divididos en 4 bancos de **00h a 7Fh**, o sea **128** bytes, lo cual nos da **4 * 128 = 512**. En su interior se encuentran los registros **SFR** o propios del micro y GPR o registros de usuario.

libros.ep-electropc.com

Mapa de Registros PIC16F877A

File	Addr	File	Addr	File	Addr	File	Addr
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h		107h		187h
PORTD	08h	TRISD	88h		108h		188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch	EEDATA	10Ch	EECON1	18Ch
PIR2	0Dh	PIE2	8Dh	EEADR	10Dh	EECON2	18Dh
TMR1L	0Eh	PCON	8Eh	EEDATH	10Eh		18Eh
TMR1H	0Fh		8Fh	EEADRH	10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h	SSPCON2	91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h	General	116h	General	196h
CCP1CON	17h		97h	Purpose	117h	Purpose	197h
RCSTA	18h	TXSTA	98h	Register (1)	118h	Register (1)	198h
TXREG	19h	SPBRG	99h	16 Bytes	119h	16 Bytes	199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch	CMCON	9Ch		11Ch		19Ch
CCP2CON	1Dh	CVRCON	9Dh		11Dh		19Dh
ADRES	1Eh	ADRESL	9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers (2) 96 Bytes	20h 7Fh	General Purpose Registers (3) 80 Bytes	A0h EFh	General Purpose Registers (3) 80 Bytes	120h 16Fh	General Purpose Registers (3) 80 Bytes	1A0h 1EFh
		Mapped in Bank0 70h - 7Fh (4)	F0h FFh	Mapped in Bank0 70h - 7Fh (4)	170h 17Fh	Mapped in Bank0 70h - 7Fh (4)	1F0h 1FFh

Bank0		Bank1		Bank2		Bank3	
[RP1:RP0]	00	[RP1:RP0]	01	[RP1:RP0]	10	[RP1:RP0]	11
[IRP] 0				[IRP] 1			

Figura 18.1 Mapa de SFR y GPR.

libros.ep-electropc.com

En la Figura 18.1 se observa el mapa del **PIC16F877A** con todos los **SFR** que están en la parte inferior de cada Banco **00h a 1Fh,** y los **GPR** en la parte superior de la memoria en la forma como están distribuidos, posiciones **20h a 7Fh**; cada micro tiene su propio mapa de registros y hay que consultarlos para trabajar con ellos. En la Figura 18.1 puede observar que algunos **SFR** (*Special Function Registers*) aparecen repetidos; eso significa que se acceden en esas posiciones de memoria (por ejemplo, el registro STATUS se puede accesar en 4 posiciones 03h, 83h, 103h y 183h); a nivel de programación esto implica que se puede acceder desde cualquier banco; por consiguiente, no debemos preocuparnos en cuál banco de memoria se está, cuando llamamos el registro STATUS.

Esto ayuda mucho cuando se está programando, porque minimiza el código al no tener los cuidados necesarios de cambio de bancos, los registros repetidos son: **INDF, PCL, STATUS, FSR, PCLATH e INTCON**. Otros registros, como el **TMR0** y el **PORTB** están disponibles en los bancos **0** y **2**, y los registros **OPTION_REG** y **TIRSB** están disponibles en los bancos 1 y 3. En la **Figura** 18.1 todas las posiciones **GPR** (*General Purpose Register*) están marcadas como (**1**) (**2**) (**3**); son posiciones de memoria donde podemos poner nuestras variables de usuario.

Otra particularidad de la memoria es que las posiciones **70h a 7Fh, F0h a FFh, 170h a 17Fh, 1F0h a 1FFh** son las mismas; o sea que estas 16 posiciones superiores de memoria son accedidas desde cualquier banco; esto nos posibilita trabajar con estas posiciones de memoria sin ocuparnos del banco en que estamos, con la ventaja de que son **GPR** o de usuario; por eso, podemos poner allí variables que son de trabajo frecuente en el programa.

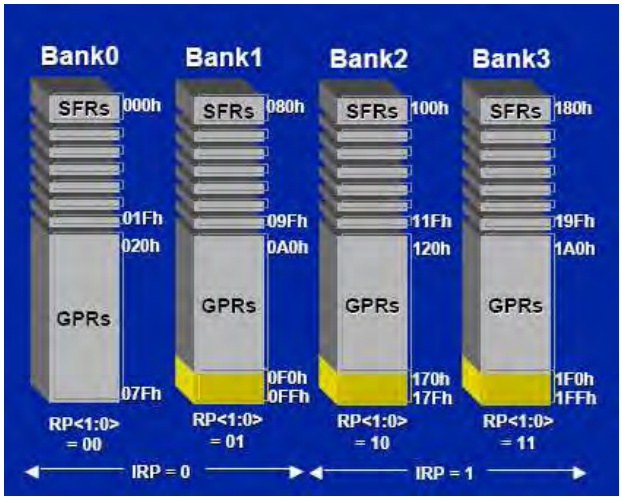


Figura 18.2
Diagrama de SFR y GPR.

En la Figura 18.2 encontramos un cuadro resumido; observe cómo se ubican los bits para acceso directo **RP<1:0>** y para acceso indirecto **IRP,** dependiendo del banco.

libros.ep-electropc.com

18.2 SFR

Registros de control del microcontrolador.

18.2.1 DEFINICIÓN

Los **SFR** ya se encuentran definidos en el **MPLAB** en los archivos *** .inc.** Para el usuario puede pasar inadvertida la dirección que tenga un **SFR** en la memoria, debido a que el **MPLAB** usa el archivo **P16F877A.inc** (agregado al proyecto en la ventana del proyecto en la carpeta *Header Files*), el cual tiene definidos todos los registros y sus bits asociados, y por esa razón no hace falta definirlos nuevamente en el programa, como se muestra en la Figura 18.3.

A) B)

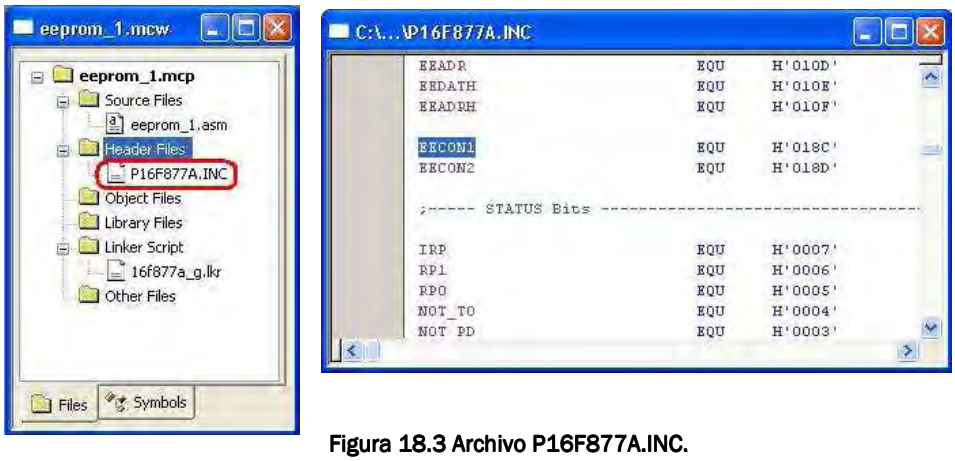


Figura 18.3 Archivo P16F877A.INC.

De esa forma, al escribir un programa en el **MPLAB** (por ejemplo **EECON1**), éste aparece de color rojo indicando que es un registro reconocido y, por tanto, ya definido en el archivo **P16F877A.INC**, tal como se muestra en la Figura 18.3, donde se observa el registro **EECON1** con la posición de memoria **H'018C'**.

18.2.2 UBICACIÓN

Todas las posiciones de estos registros de control del microcontrolador están entre **00h y 1Fh** en cada uno de los bancos (**Banco 0, Banco 1, Banco 2, banco 3**). Una forma de ver la posición de un registro **SFR** es con la ventana *Watch*, al agregar el registro deseado se muestra la dirección que tiene (por ejemplo **EEDTA,** posición **10Ch**), como se ve en la Figura 18.4. (Para ver cómo se agrega la ventana de Watch, consulte la **Parte 3: Programación II**, en el **Capítulo 14.0**, **Sección 14.5: Depuración**,

libros.ep-electropc.com

Subsección 14.5.5: **Pasos para depuración ejemplo**, en el ítem **Paso 6 Agregar la ventana *Watch*** y **Registros GPR.**)

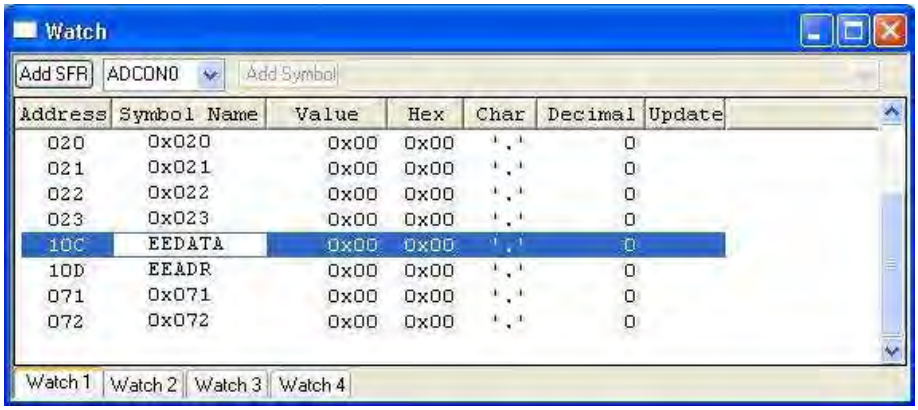


Figura 18.4 Ventana *Watch*.

Otra manera de encontrar la posición de un registro **SFR** es usar la tabla que se encuentra en la **Sección 18.1: Mapa de registros**, o en el archivo **DS39582B-page 18**, que es la *Data Sheet* en el pdf del micro **P16F877A**, al cual remitimos.

18.2.3 INICIALIZACIÓN

Los registros **SFR** son cargados luego de un **POR** (*Power On Reset*) o **BOR** (*Brown On Reset*) (Tabla 18.1) con valores por defecto que son definidos en la *Data Sheet* o en el archivo **DS39582B** en la **Sección 2.2.2: Special Function Registers**, pp, 19-21, a la cual remitimos.

Tabla 18.1 Resumen de la función especial de registros.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Details on page:
02h(3)	PCL	Program Counter (PC) Least Significant Byte								0000 0000	30, 150
03h(3)	STATUS	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	22, 150
04h(3)	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	31, 150
8Dh	PIE2	—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE	-0-0 0--0	27, 151
8Eh	PCON	—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$	---- --qq	29, 151

x = unknown, u = unchanged, q = value depends on condition, - = unimplemented, read as '0', r = reserved.
Shaded locations are unimplemented, read as '0'.

Para inicializar con un valor diferente del defecto, se debe cargar como cualquier otro registro de RAM con cualquiera de las instrucciones explicadas en la **Parte 3: Programación I** en el **Capítulo 11.0, Sección 11.3: Set de instrucciones**. Las formas de cargar un **SFR** con un valor son las siguientes:

libros.ep-electropc.com

Ejemplos:

```
#define   _ini_ram  0x20              ; posición inicial de RAM
          movlw     _ini_ram          ; _ini_ram -> W
          movwf     FSR               ; W -> FSR

          clrf      PORTB             ; 00 -> puerto B

          movlw     0xF0              ; 0F -> W
          movwf     TRISB             ; W -> TRISB

          bsf       PIE2,EEIE         ; Habilita int. De EEPROM (bit EEIE de PIE2)
```

18.3 GPR

Son los registros de memoria de usuario del microcontrolador.

18.3.1 DEFINICIÓN

Al principio del programa es necesario definir los GPR o de usuario, o sea los registros propios del programa; esto se realiza con la directiva cblock y se cierra con endc.

Para definir las variables:

```
          cblock 0x20         ; desde la posición 20h en RAM
cont_SS                       ; contador de Segundos              posición 0x20
cont_MM                       ; contador de Minutos               posición 0x21
cont_HH                       ; contador de Horas                 posición 0x22
cont_año            :2        ; contador de año de 2 byte         posición 0x23
cont_mes                      ; contador de mes                   posición 0x25
cont_dia                      ; contador de día                   posición 0x26
          endc                ; fin de registros
```

Este bloque define las posiciones a partir de la posición **0x20h**. Observe que el registro o la variable **cont_año :2** define dos posiciones de memoria, **0x23** y **0x24**, y el registro siguiente **cont_mes** está en la posición **0x25**.

```
      cblock 0x71    ; desde la posición 71h en RAM
reg_LH    :0         ; no ocupa registro, sólo apunta a esta dirección, registro doble HL
reg_L                ; registro low
reg_H                ; registro high
          endc       ; fin de registros
```

Este bloque define las posiciones a partir de la posición 071h. Observe que el registro o la variable **reg_LH :0** define cero posiciones de memoria, lo que se usa como apuntador, de tal forma que tiene la misma dirección del registro siguiente **reg_L** posición **0x71** y, por último, el registro **reg_H** posición **0x72**.

libros.ep-electropc.com

18.3.2 UBICACIÓN

Al realizar la compilación, podemos observar las variables definidas en el archivo ***.ASM** y que posiciones de memoria quedan en el archivo ***.LST**, como se aprecia en la **Figura** 18.5.

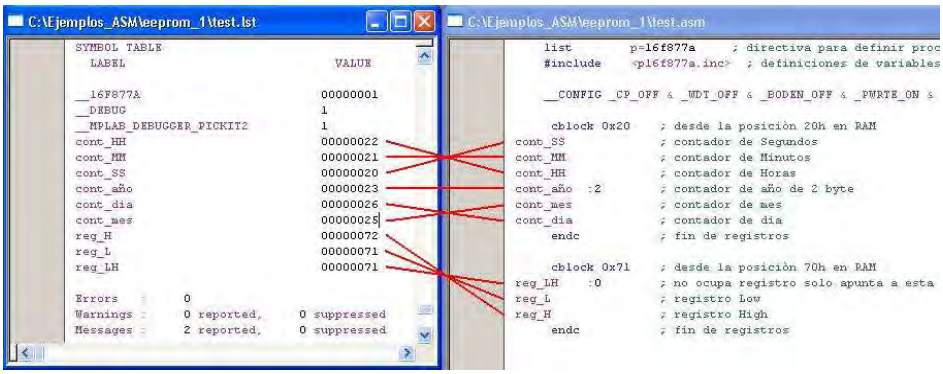


Figura 18.5 Definición de registros GPR.

En el primer bloque **020h**, observe que cada variable ocupa una posición de memoria con la excepción de cont_año :2 que ocupa dos posiciones de memoria **023h** y **024h**, por su definición cont_año :2 que separa dos posiciones. En el segundo bloque **071h**, la definición reg_LH :0 no ocupa posiciones de memoria, funciona solamente como apuntador. Todas las posiciones de estos registros de memoria del microcontrolador están entre 2**0h** y **7Fh** en cada uno de los bancos.

18.3.3 INICIACIÓN

Los **GPR** son cargados luego de un **POR** (*Power On Reset*) con un valor impredecible (aleatorio). Para iniciar con un valor se deben cargar como cualquier otro registro de RAM con cualquiera de las instrucciones explicadas en la **Parte 3: Programación I**, en el **Capítulo 11.0, Sección 11.3: Set de instrucciones**.

Ejemplos:

```
#define   _dia_0   .1          ; día inicial
          movlw    _dia_0              ; _dia_0 -> W
          movwf    cont_dia            ; W -> cont_dia

          clrf     contador            ; 00 -> contador

          movlw    b'0011 1100'        ; 3C -> W
          movwf    filtro              ; W -> filtro

          bsf      nivel,7             ; Habilita bit 7 de nivel
```


libros.ep-electropc.com

18.4 TIPOS DE DIRECCIONAMIENTO

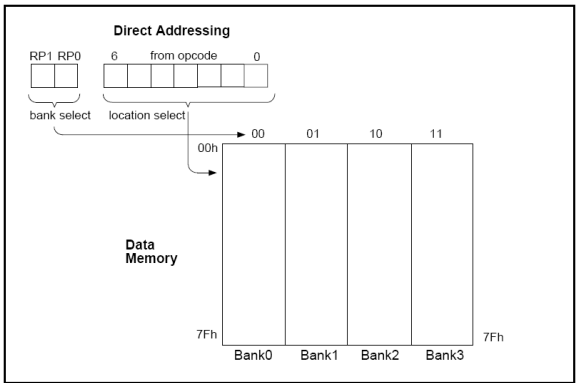
Para acceder a los registros en RAM tenemos dos formas:

- **Direccionamiento directo**
  Con los bits [RP1:RP0] del registro STATUS<6:5> y la instrucción.
- **Direccionamiento indirecto**
  Con el registro FSR, INDF y el bit [IRP] del registro STATUS<7>

18.4.1 DIRECCIONAMIENTO DIRECTO

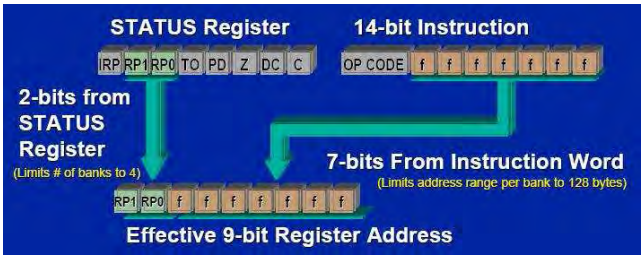
Para acceder los 512 registros (4 bancos de 128), que se encuentran en la memoria RAM (Tabla 18.2) se necesita la dirección que ocupa en el banco (**00h a 7Fh**) más los bits del banco, que son los bits [RP1:RP0] del registro **STATUS<6:5>**.

Tabla 18.2 Bloques de direccionamiento directo.



La dirección de memoria completa necesita 9 bits que nos da $2^9 = 512$ posiciones de toda la memoria RAM, 7 bits (6:0) que es la posición de **00h a 7Fh** son $2^7 = 128$ posiciones en cada banco, más los 2 bits de selección del banco [RP1:RP0] del registro **STATUS<6:5>** que nos da $2^2 = 4$ posibles bancos, tal como se ilustra en la **T**abla 18.2. Estos 9 bits nos dan la posibilidad de acceder las **512** direcciones (**000h a 1FFh**), que son **4** bancos de **128** posiciones (Figura 18.6), **é**sta es el área completa del Mapa de Registros.

Figura 18.6 Direccionamiento directo.



libros.ep-electropc.com

Para cargar un registro es necesario seleccionar previamente el banco apropiado.

```
        bsf     STATUS,RP1      ; RP1 = 1
        bcf     STATUS,RP0      ; RP0 = 0, Banco 0
        movlw   0x55            ; 0x55 -> W
        movwf   EECON2          ; W -> EECON2
```

Otra forma es utilizar la directiva **banksel**:

```
        banksel EECON2          ; selecciona banco de EECON2
        movlw   0x55            ; 0x55 -> W
        movwf   EECON2          ; W -> EECON2
```

Sin embargo, esta directiva podría agregar demasiado código porque pone 2 instrucciones de programa cada vez que se emplea. Por eso se debe tener cuidado con su uso.

18.4.2 DIRECCIONAMIENTO INDIRECTO

El direccionamiento indirecto consiste en poner un apuntador a una posición de la memoria RAM, y desde allí poder acceder esa posición de memoria para operar con ella. En este caso el apuntador es el **FSR**, y la forma de operar indirectamente con el registro es con el **INDF**; conviene **observar** que el INDF no es un registro físico.

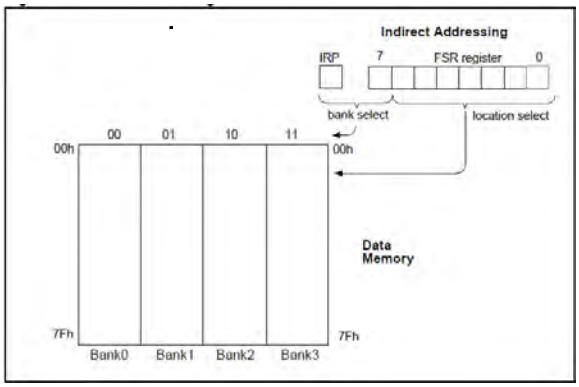


Figura 18.7
Bloques de direccionamiento indirecto.

En este modo de direccionamiento los registros se acceden con el apuntador **FSR**; el registro **INF** y la selección de bancos con el bit 7 del **FSR** y el bit **IRP** del registro **STATUS<7>**.

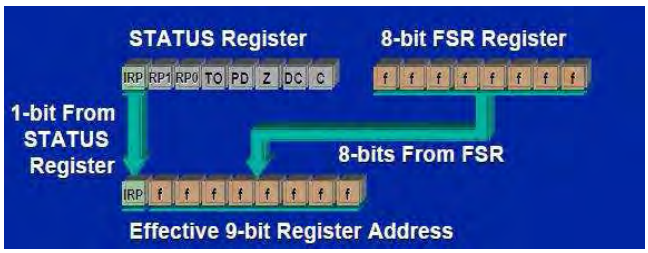


Figura 18.8
Direccionamiento indirecto.

libros.ep-electropc.com

En el ejemplo siguiente se borrarán las posiciones de memoria de 040h a 04Fh.

```
        movlw   0x40    ; inicialice apuntador
        movwf   FSR     ; a RAM
ciclo:
        clrf    INDF    ; borra INDF
        incf    FSR,F   ; incremente apuntador
        btfss   FSR,4   ; ¿terminó? Sí, salta
        goto    ciclo   ; no, borre el siguiente
end_ciclo:
```

EVALUACIÓN

P.: 1. La memoria RAM está compuesta por 2 tipos de registros, ¿cuáles son?
R.:
Special Function Registers (SFR) o Registros de funciones especiales.
General Purpose Registers (GPR) o Registros de propósito general.

P.: 2. ¿Cuántos registros de memoria RAM hay, y en cuántos bancos se separan?
R.:
RAM 512 (00h - 1FF)
Banco 1 (00h - 07Fh)
Banco 2 (80h - 0FFh)
Banco 3 (100h - 17Fh)
Banco 3 (180h - 1FFh)

P.: 3. ¿Cuáles son las directivas para definir los GPR?
R.:
cblock y endc.

P.: 4. ¿Cuáles tipos de direccionamiento hay para la memoria RAM?
R.:
Directo e Indirecto.

P.: 5. ¿Cuáles son los bits usados en el direccionamiento directo para seleccionar bancos de RAM, y en qué registro se encuentran?
R.:
Bits RP0 y RP1
Registro STATUS<5,6>

P.: 6. ¿Cuáles son los bits usados en el direccionamiento indirecto para seleccionar bancos de RAM, y en qué registro se encuentran?
R.:
Bits IRP y FSR<7>
IRP en Registro STATUS<5>, FSR<7> en Registro FSR

libros.ep-electropc.com

P.: 7. ¿Cuál es el registro apuntador y cuál el registro de contenido utilizados en el direccionamiento indirecto?

R.:
Registro apuntador FSR
Registro de contenido INDF

libros.ep-electropc.com

PARTE 5

PUERTOS I/O, CONVERSORES A/D Y COMPARADORES

En esta parte veremos los **puertos I/O** o **puertos de entrada-salida**, que son los periféricos básicos de un microcontrolador, son entradas y salidas digitales de valores '1' y '0'; sus diagramas circuitales y su funcionamiento como puertos y como bits, la inicialización y la configuración de los **5 puertos A, B, C, D** y **E**; cómo cada uno de estos puertos comparten sus pines con otros módulos periféricos. Además el **puerto esclavo paralelo** o **PSP**, que es una forma especial del funcionamiento de los puertos D y E, para la transferencia de datos entre los MCU.

Los **conversores análogo a digital** o **A/D**, que son **entradas analógicas**, es decir, que su valor es continuo entre el voltaje de alimentación V_{DD} y V_{SS}. Este módulo nos ayuda a digitalizar las señales analógicas, de tal manera que el valor puesto en el pin se convierte en un valor digital de 10 bits entre 0 y 1 023, el cual se vuelve manipulable dentro del MCU. Cómo es la estructura circuital del conversor A/D, su configuración y su inicialización y la forma de tomar los valores por *polling* e interrupción. Cómo manejar los **conversores análogo a digital** que nos ayudan a digitalizar las señales analógicas y, por último, veremos la función de **comparador** que interviene con el comportamiento de las señales **análogas** y **digitales**.

El **comparador** es un módulo que relaciona los valores de entrada análoga con su salida digital, funciona como el dispositivo **comparador** que se encuentra en circuitos integrados. Es de gran utilidad porque realiza la operación por hardware de comparación de dos señales análogas, cuyos voltajes oscilan entre V_{DD} y V_{SS}, y

libros.ep-electropc.com

entrega un valor digital '1' o '0', como resultado de su comparación. Si no fuera así, nos tocaría hacerlo por software, digitalizando las entradas y haciendo una resta para arrojar un resultado digital; esta labor nos llevaría varios ciclos de máquina.

El módulo **generador de voltaje de referencia (VREF)**, que nos proporciona un voltaje por una red de 24 resistencias (16 valores seleccionables), conectada entre V_{DD} y V_{SS}. Este módulo se puede conectar al **módulo comparador**, o directamente como pin de salida; y aunque no es de alta precisión, nos da una idea del voltaje de entrada; toma 16 valores entre ¼ a ¾ de V_{DD}, y 0 a ⅔ de V_{DD}.

19.0 PUERTOS I/O (INPUT / OUTPUT)

Los **puertos I/O**, o de **entrada/salida** (*input/output*), son los periféricos básicos de un microcontrolador; denominados **PORTx**, donde **x = A,B,C**, etc.; se tratan como **puerto total** (p. ej., **PORTD** generalmente 8 bits); o como bit (p. ej., **PORTD<5>** directamente el pin 5 del puerto D). Cada bit en el registro **PORTx** corresponde a un valor ("1" = 5V y "0" = 0V) en el pin del MCU, y su dirección será configurada en el registro **TRISx** ("1" entrada y "0" salida). Los puertos generalmente tienen 8 bits (p. ej., PORTD[7:0]) y se numeran de 0 a 7; el 0 es el de menor peso (**LSB**: *Less Significant Bit*), y 7 el de mayor peso (**MSB**: *Most Significant Bit*).

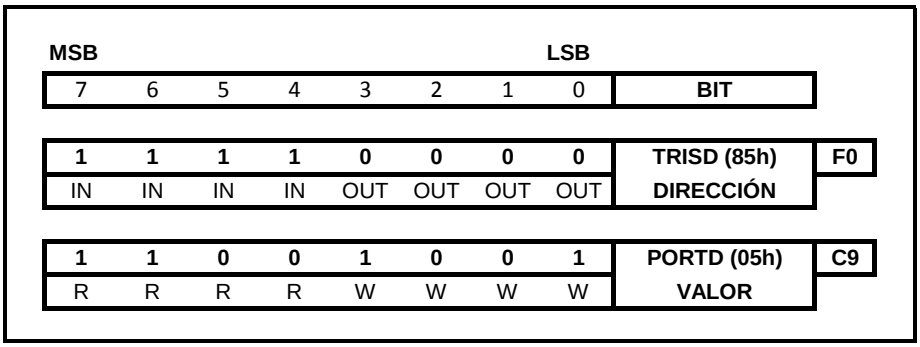


Figura 19.1 Configuración y valor de un puerto.

En la Figura 19.1 apreciamos cómo está configurado el **puerto D** (PORTD), PORTD[3:0] salidas y PORTD[7:4] entradas. También vemos que el valor leído tiene validez en los pines configurados como entrada **IN** (INPUT) marcado como **R** (READ: lectura), y al ser escrito tendrá validez en los pines configurados como salidas OUT (*output*) marcado como **W** (WRITE: escritura). Se debe tener cuidado al usar los valores escritos y leídos en el mismo puerto. Leyendo el registro PORTD lee el estado de los pines, mientras que escribiéndolo, escribimos sobre el *latch* del puerto.

libros.ep-electropc.com

Todas las operaciones de escritura son operaciones de lectura-modificación-escritura. Por lo tanto, una escritura en el puerto implica que los bits del puerto son leídos, los valores son modificados y entonces escribe sobre el *latch* del puerto. Algunos de estos pines I/O (*Input/Output:* entrada/salida) de los puertos son multiplexados con una función alterna; sus características corresponden al periférico escogido. En general, cuando un periférico está habilitado no puede ser utilizado como un I/O pin de propósito general.

19.1 PUERTO A

El **puerto A** (PORTA) es de 6 bits de ancho y bidireccional, y es denominado **RA<5:0>**.

19.1.1 DIAGRAMA CIRCUITAL

En las Figuras 19.2, 19.3 y 19.4 apreciamos tres tipos de configuraciones de pines, en su orden respectivo: **RA0:RA3, RA4/TOCK1** y **RA5**.

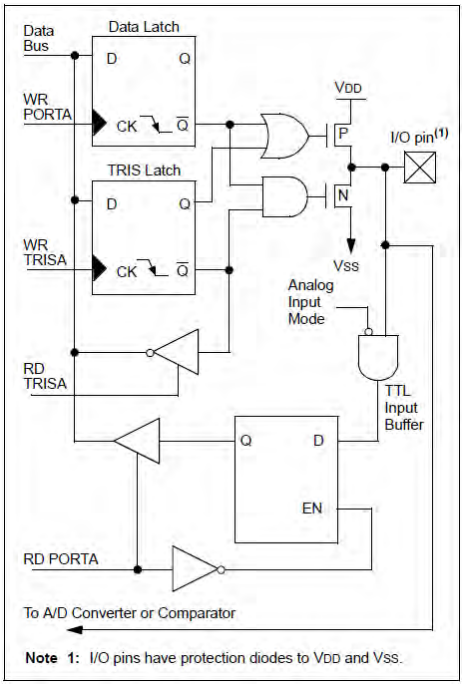


Figura 19.2 Diagrama circuital RA0:RA3

libros.ep-electropc.com

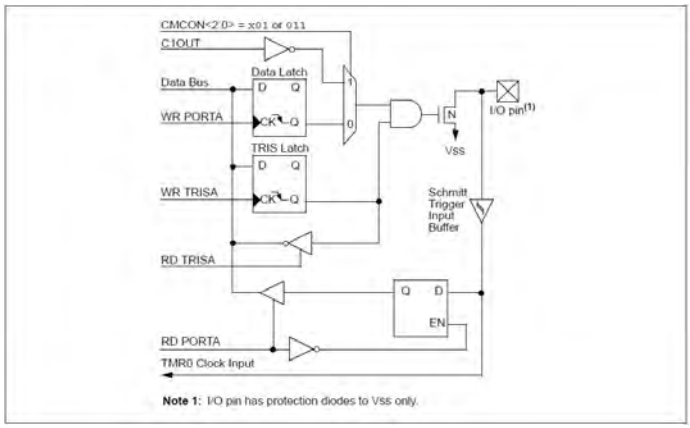


Figura 19.3 Diagrama circuital RA4/TOCK1.

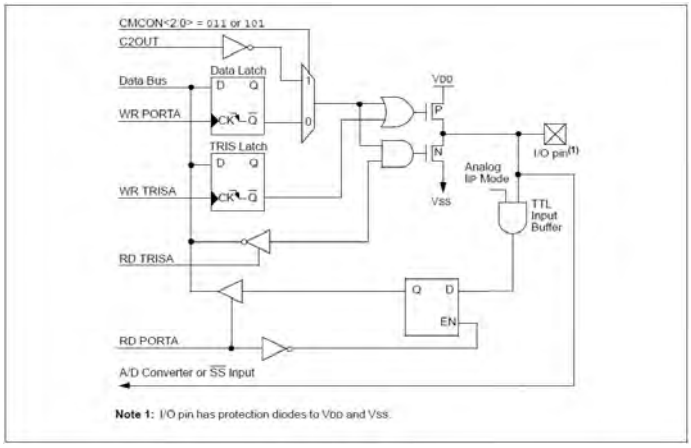


Figura 19.4 Diagrama circuital RA5.

19.1.2 FUNCIONAMIENTO

El **puerto A** es de 6 bits de ancho y bidireccional. La dirección de pines correspondiente es el registro **TRISA**. El pin RA4 es multiplexado con la entrada del módulo de reloj Timer0 para convertirse en el **TOCKI pin**. El **RA4/TOCKI** es una entrada *Schmitt Trigger* y una salida *open-drain* u *open collector.*

```
NOTA: Para leer el dato en una salida open-drain u open collector
hay que agregar R de pull-up.
```


libros.ep-electropc.com

Todos los demás pines del PORTA son niveles TTL de entrada y *full* CMOS *driver* de salida. Otros pines del **puerto A** son multiplexados con entradas analógicas y entrada analógica **VREF** de entrada para los conversores A/D y los comparadores.El funcionamiento de cada pin es seleccionado por borrar o poner a uno los bits de control adecuados en los registros **ADCON1** y/o **CMCON**.

NOTA: Al darse un *power-on reset*, estos pines son configurados como entradas análogas y se leen como "0". Los comparadores están en estado *off* (digital).

El registro **TRISA** controla la dirección de los pines del **puerto A** cuando se usan como entradas análogas, y debe asegurar que los bits correspondientes se mantengan en uno.

19.1.3 REGISTROS ASOCIADOS

En la Tabla 19.1 se describen las funciones de los pines del **puerto A**, y en la Tabla 19.2 los registros asociados al mismo **PUERTO A**.

Tabla 19.1 Funciones de los pines del PUERTO A.

Name	Bit#	Buffer	Function
RA0/AN0	bit 0	TTL	Input/output or analog input.
RA1/AN1	bit 1	TTL	Input/output or analog input.
RA2/AN2/VREF-/CVREF	bit 2	TTL	Input/output or analog input or VREF- or CVREF.
RA3/AN3/VREF+	bit 3	TTL	Input/output or analog input or VREF+.
RA4/T0CKI/C1OUT	bit 4	ST	Input/output or external clock input for Timer0 or comparator output. Output is open-drain type.
RA5/AN4/$\overline{SS}$/C2OUT	bit 5	TTL	Input/output or analog input or slave select input for synchronous serial port or comparator output.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Tabla 19.2 Resumen de registros asociados con el PUERTO A.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by PORTA.

Note: When using the SSP module in SPI Slave mode and $\overline{SS}$ enabled, the A/D converter must be set to one of the following modes, where PCFG3:PCFG0 = 0100, 0101, 011x, 1101, 1110, 1111.

libros.ep-electropc.com

Tabla 19.3 Registro ADCON1 (ADDRESS 9Fh).

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM**: A/D Result Format Select bit

1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2**: A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	FOSC/2
0	01	FOSC/8
0	10	FOSC/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	FOSC/4
1	01	FOSC/16
1	10	FOSC/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented**: Read as '0'

bit 3-0 **PCFG3:PCFG0**: A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: On any device Reset, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

libros.ep-electropc.com

En la Tabla 19.4 se indican los diferentes periféricos que comparte el **PUERTO A**. Cada pin tiene sus distintos nombres y el módulo periférico al que corresponde.

Tabla 19.4 Periféricos.

PIN	PUERTO	NOMBRE PIN	PERIFERICOS		
2	RA0	RA0/AN0	AN0		
3	RA1	RA1/AN1	AN1		
4	RA2	RA2/AN2/VREF-/CVREF	AN2	VREF-	CVREF
5	RA3	RA3/AN3/VREF+	AN3	VREF+	
6	RA4	RA4/T0CKI/C1OUT	T0CKI	C1OUT	
7	RA5	RA5/AN4/SS/C2OUT	AN4	SS	C2OUT

19.1.4 PROCEDIMIENTO DE USO

El proceso consiste en los pasos siguientes:

1. Ponga el valor de inicialización en el puerto **PORTA** (05h).
2. Ponga la dirección de los pines en el puerto **TRISA** (85h).
3. Ponga la configuración de los pines digitales en el registro **ADCON1** (9Fh). (Este paso se requiere para cambiar los pines **ANx** o **A/D** definidos por default a pines digitales.)
4. Lea o escriba datos en el puerto **PORTA** (05h).

A continuación damos el código del ejemplo:

1. Ponga el valor de inicialización en el puerto **PORTA** (05h)

```
movlw   b`00101000      ;RA<5,3> entradas RA5 = RA3 = 1,
                        ;RA<4,2:0> salidas, RA4 = RA2 = RA1 = RA0 = 0
movwf   PORTA           ;ponga en puerto A
```

2. Ponga la dirección de los pines en el puerto **TRISA** (85h):

```
bsf     STATUS,RP1      ;banco 1
movlw   b`00101000      ;RA<5,3> entradas RA5 = RA3 = 1,
                        ;RA<4,2:0> salidas, RA4 = RA2 = RA1 = RA0 = 0
movwf   TRISA
```

3. Ponga la configuración de los pines digitales en el registro **ADCON1** (9Fh):

```
movlw   b'00000110'     ; seleccione todos los ANx (A/D) como digitales
movwf   ADCON1          ; configura en ADCON1
bcf     STATUS,RP1      ; banco 0
```


libros.ep-electropc.com

4. Lea o escriba datos en el puerto **PORTA** (05h):

```
movf    PORTA,W         ; lea puerto A y póngalo en W
movwf   PORTA           ; escriba W en puerto A
```

19.1.5 PRÁCTICA

Esta práctica consiste en hacer titilar como salidas el **PUERTO A**; esto se verifica en los pines del conector **P9** en la tarjeta **PIC16F8XX**; el montaje para la práctica es tal como se muestra en la Figura 19.5.

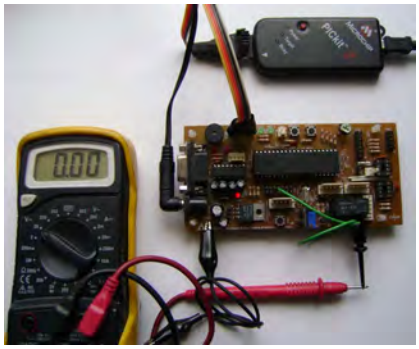


Figura 19.5 Práctica de montaje PUERTO A.

En la tarjeta **PI16F8XX** se especifican el conector y sus pines, como muestra la Figura 19.6.

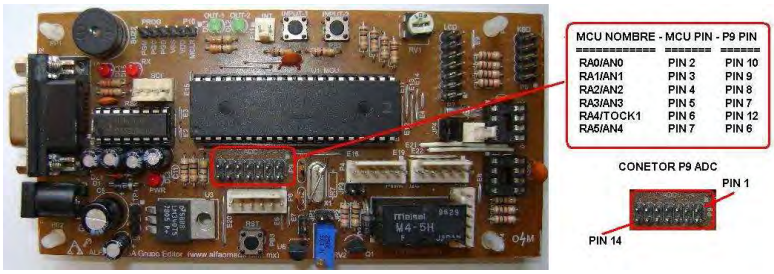


Figura 19.6 PUERTO A en conector P9.

libros.ep-electropc.com

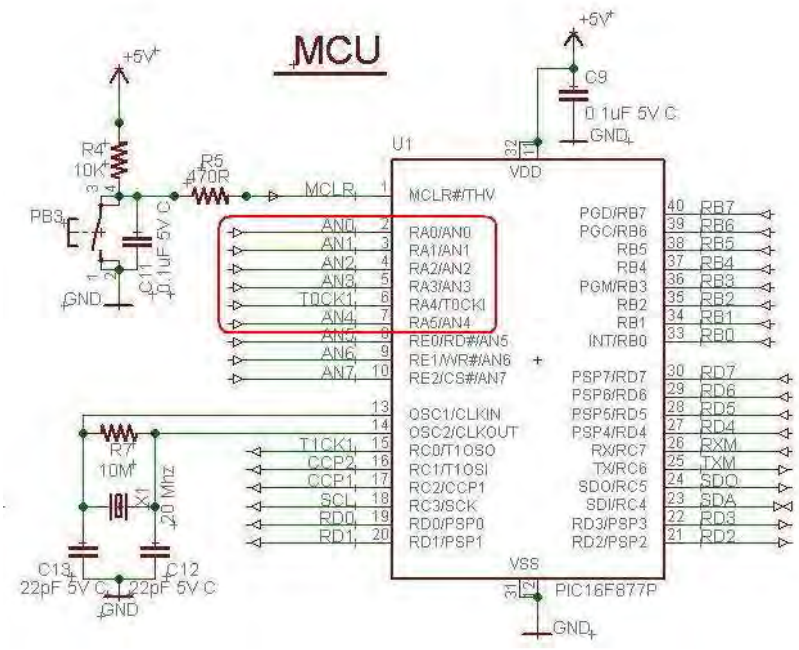
Los pines que trabajaremos con su función se ilustran en la Tabla 19.5.

CONEXION PUERTO A CONECTOR P9			
MCU		P9	
PIN	NOMBRE	PIN	FUNCIÓN
		14	+5V
		13	+5V
6	RA4/T0CKI/C1OUT	12	RA4
		11	
2	RA0/AN0	10	RA0
3	RA1/AN1	9	RA1
4	RA2/AN2/VREF-/CVREF	8	RA2
5	RA3/AN3/VREF+	7	RA3
7	RA5/AN4/SS/C2OUT	6	RA5
8	RE0/RD/AN5	5	
9	RE1/WR/AN6	4	
10	RE2/CS/AN7	3	
		2	GND
		1	GND

Tabla 19.5 Pines y función del PUERTO A.

El plano esquemático de conexión se encuentra descrito a continuación en la Figura 19.7a, b.

a)



libros.ep-electropc.com

b)

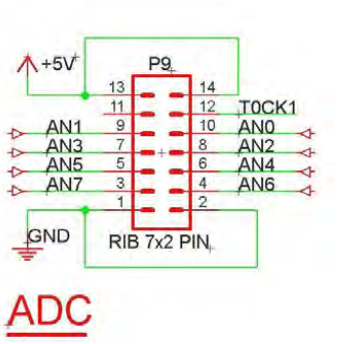


Figura 19.7 Plano esquemático del PUERTO A en conector P9.

En el programa siguiente, en modo de paso a paso, se ponen en '1' y luego en '0' todos los pines del **PUERTO A**. A continuación, el código del programa, el cual se encuentra en nuestra página Web en la carpeta **X:\Ejemplos_ASM**, se debe copiar al **C:** para encontrarlo como **C:\Ejemplos_ASM\ PUERTO_A_1 \ PUERTO_A_1.asm**:

```
        list p= 16f877a          ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

            org 0       ; vector de reset
Start:
            clrf    PORTA           ; apague puerto A
            bsf     STATUS,RP0      ; seleccione Banco 1
            clrf    TRISA           ; configure puerto A como salidas
            movlw   b'00000110'     ; seleccione todos los ANx (A/D) como digitales
            movwf   ADCON1          ; configura en ADCON1
            bcf     STATUS,RP0      ; seleccione Banco 0

Main_Loop:

ON:
            Comf    PORTA,F         ; prenda puerto A
OFF:
            Comf    PORTA,F         ; apague puerto A

            Goto    Main_Loop       ; haga ciclo

            end
```

Al seguir paso a paso este programa, en la Figura 19.8 a), b) se observa cómo el voltaje medido en el multímetro cambia en el pin **RA0** al pasar de **'0' a '1'** en la instrucción comf PORTA,F del paso ON: y en el paso OFF:

libros.ep-electropc.com

A) B)

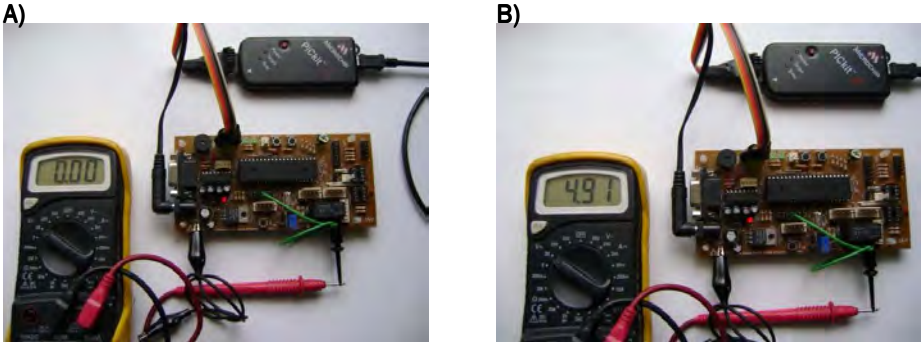


Figura 19.8 Voltaje en RA0 pin 10 en conector P9.

En nuestra página Web puede encontrar este ejemplo con retardos y subrutinas en la carpeta **X:\Ejemplos_ASM**, la que debe copiar en el disco **C:\Ejemplos_ASM\PUERTO_A_1 \ PUERTO_A_1 A.asm**.

19.2 PUERTO B

PORTB es de 8 bits de ancho y bidireccional. Es denominado RB<7:0>.

19.2.1 DIAGRAMA CIRCUITAL

En las Figuras 19.9 y 19.10 podemos apreciar dos tipos de configuraciones de pines: **RB3:RB0** y **RB7:RB4**.

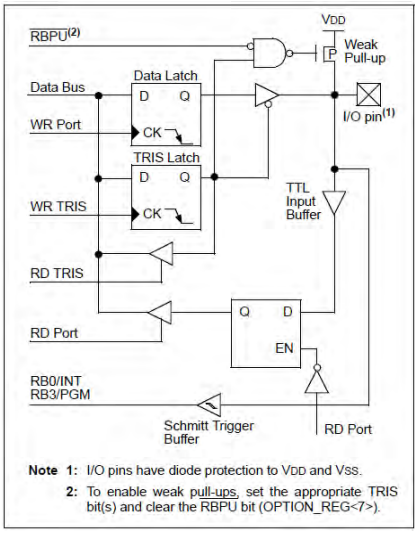


Figura 19.9 Configuraciones de pines: RB3:RB0.

libros.ep-electropc.com

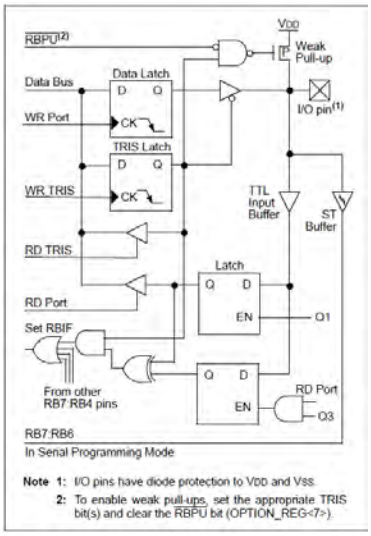


Figura 19.10 Configuraciones de pines: RB7:RB4.

19.2.2 FUNCIONAMIENTO

El **PUERTO B** es de 8 bits de ancho y bidireccional. La dirección de pines correspondiente es el registro TRISB. Tres pines del puerto B son multiplexados con la función de **depurador en circuito** (*in circuit debugger*) y **programación de bajo voltaje** (*low voltage programming*): **RB3/PGM, RB6/PGC** y **RB7/PGD**. El pin **RB3/PGM** es para programación en voltaje bajo (3V). Estos pines se pueden usar, pero lo recomendable es que se dejen vacíos y dedicados a la función de programación y depuración.

Cada uno de los pines del **PUERTO B** tienen un *pull-up* interno, se pueden habilitar o no con un solo bit en el registro **OPTION_REG <7>**; también son automáticamente puestos en *off* cuando se programan como salidas; cuando hay un *power-on reset* los *pull-up*s son deshabilitados. Cuatro de los pines **RB7:RB4** tienen la característica de **interrupción por cambio** (*interrupt on-change)*, esto significa que cuando en la entrada hay un cambio se genera una interrupción. Eso es posible solamente cuando estos 4 pines se configuran como entradas; si están configurados como salida, no generan esta interrupción.

Estos pines de entrada son comparados con los valores viejos de la última lectura de PORTB, y están alambrados con circuitería OR, de tal forma que si ocurre un cambio en algún pin **RB7:RB4** se genera la interrupción de cambio de puerto RB (*interrupt on-change)*, poniendo en uno el bit de bandera **RBIF** en el registro **INTCON<0>**. Esta interrupción puede despertar el MCU si está en modo de *sleep* (dormido). La interrupción se puede apagar en la Rutina de Servicio de Interrupción, de dos maneras:

libros.ep-electropc.com

- Cualquier lectura o escritura del PORTB; esto termina la condición de ocurrencia.
- Borrar el bit de la bandera RBIF.

Una condición de ocurrencia puede estar poniendo a uno el bit de la bandera RBIF; leyendo el PORTB puede terminar la condición de ocurrencia y permite que el bit de la bandera RBIF sea borrada. La característica de *interrupt-on-change* (interrupción-por-cambio) es recomendada para operaciones de *wake-up on key depression* ("despierte por presionar botón" o tecla). Cuando se usa *interrupt-on-change*, no es recomendable hacer *polling* (lecturas) en PORTB. La característica de *interrupt-on-mismatch* ("despierte por ocurrencia"), junto con la configuración de los 4 *pull-up*s de entrada, facilita la interfaz con teclados en aplicaciones de *wake-up on key depresión.*

```
NOTA: Consulte la nota de aplicación, AN552, "Implementing Wake-up
on Key Stroke" (DS00552).
```

El RB0/INT es un pin de interrupción de entrada. Este pin se discute en la **Parte 2: Arquitectura**, en el **Capítulo 9.0, Sección 9.3: Interrupción**, y en la **Parte 3: Programación I**, en el **Capítulo 11.0, Subsección 11.5.2: Interrupciones**.

19.2.3 REGISTROS ASOCIADOS

En la Tabla 19.6 vemos los registros asociados con el **PUERTO B** y en la Tabla 19.7 vemos la descripción de las funciones de los pines del **PUERTO B**.

Tabla 19.6 Registros asociados con el PUERTO B.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
06h, 106h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	uuuu uuuu
86h, 186h	TRISB	PORTB Data Direction Register								1111 1111	1111 1111
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged. Shaded cells are not used by PORTB.

libros.ep-electropc.com

Tabla 19.7 Registros asociados con el PUERTO B.

Name	Bit#	Buffer	Function
RB0/INT	bit 0	TTL/ST(1)	Input/output pin or external interrupt input. Internal software programmable weak pull-up.
RB1	bit 1	TTL	Input/output pin. Internal software programmable weak pull-up.
RB2	bit 2	TTL	Input/output pin. Internal software programmable weak pull-up.
RB3/PGM(3)	bit 3	TTL	Input/output pin or programming pin in LVP mode. Internal software programmable weak pull-up.
RB4	bit 4	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB5	bit 5	TTL	Input/output pin (with interrupt-on-change). Internal software programmable weak pull-up.
RB6/PGC	bit 6	TTL/ST(2)	Input/output pin (with interrupt-on-change) or in-circuit debugger pin. Internal software programmable weak pull-up. Serial programming clock.
RB7/PGD	bit 7	TTL/ST(2)	Input/output pin (with interrupt-on-change) or in-circuit debugger pin. Internal software programmable weak pull-up. Serial programming data.

Legend: TTL = TTL input, ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.

2: This buffer is a Schmitt Trigger input when used in Serial Programming mode or in-circuit debugger.

3: Low-Voltage ICSP Programming (LVP) is enabled by default which disables the RB3 I/O function. LVP must be disabled to enable RB3 as an I/O pin and allow maximum compatibility to the other 28-pin and 40-pin mid-range devices.

En la Tabla 19.8 apreciamos los diferentes periféricos que comparte el **PUERTO B**. Cada pin tiene sus diferentes nombres y módulo periférico al que corresponde.

Tabla 19.8 Periféricos que comparte el PUERTO B.

			PERIFÉRICOS
			SPECIAL
PIN	NOMBRE PIN	PUERTO	PROG
33	RB0/INT	RB0	INT
34	RB1	RB1	
35	RB2	RB2	
36	RB3/PGM	RB3	PGM
37	RB4	RB4	
38	RB5	RB5	
39	RB6/PGC	RB6	PGC
40	RB7/PGD	RB7	PGD

19.2.4 PROCEDIMIENTO DE USO

En el proceso de uso siga los pasos siguientes:

1. Ponga el valor de inicialización en el puerto PORTB (06h).
2. Ponga la dirección de los pines en el puerto TRISB (86h).

libros.ep-electropc.com

3. Lea o escriba datos en el puerto PORTB (06h).

A continuación le damos el código de un ejemplo:

1. Ponga el valor de inicialización en el puerto PORTB (06h):

```
movlw       b'1100 1011'    ;RB <5,4,2> salidas RB5 = RB4 = RB2 = 0,
                            ; RB <7,6,3,1,0> entradas, RB7 = RB6 = RB1 = RB0 = 1
movwf       PORTB           ;ponga en puerto B
```

2. Ponga la dirección de los pines en el puerto TRISB (86h):

```
bsf         STATUS,RP1      ;banco 1
movlw       b'1100 1011'    ;RB <5,4,2> salidas RB5 = RB4 = RB2 = 0, RB <7,6,3,1,0>
                            ; entradas, RB7 = RB6 = RB1 = RB0 = 1
movwf       TRISB
```

3. Lea o escriba datos en el PORTB (06h):

```
movf        PORTB,0         ;lea puerto B y póngalo en W
movwf       PORTB           ;escriba el puerto B con lo que tiene en W
```

19.2.5 PRÁCTICA

Esta práctica consiste en prender y apagar los leds **OUT-1** y **OUT-2**; siguiendo los pulsadores **INPUT-1** e **INPUT-2** del **PUERTO B**, el buzzer BUZZ prende con algún pulsador oprimido y se apaga al soltarlo o con ambos pulsadores oprimidos, en la tarjeta **PIC16F8XX**; en la Figura 19.11 podemos observar el montaje para esta práctica.

```
NOTA: Hay que tener en cuenta que los pines RB7, RB6 y RB3 están
compartidos con los pines del programador ICSP; por eso no se
deberían ocupar.
```



Figura 19.11 Práctica de montaje PUERTO B.

libros.ep-electropc.com

En la tarjeta **PI16F8XX** se especifican el conector y sus pines como se muestra en la Figura 19.12.

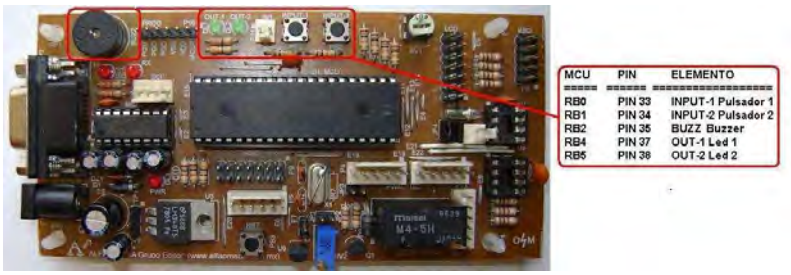


Figura 19.12 PUERTO B leds pulsadores y buzzer.

Los pines que ocuparemos se describen con su función en la Tabla 19.9.

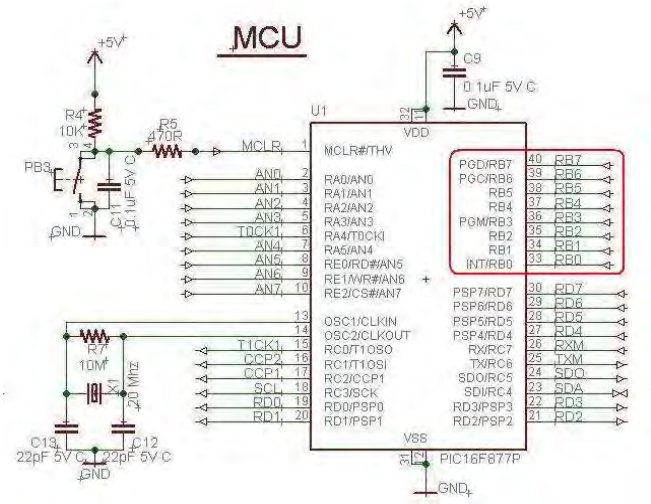
Tabla 19.9 Pines y función de PUERTO B.

CONEXIÓN PUERTO B			
MCU			
PIN	NOMBRE	MOD	FUNCIÒN
33	RB0/INT	INT	INPUT-1
34	RB1		INPUT-2
35	RB2		BUZZER
36	RB3/PGM	PGM	
37	RB4		OUT-1
38	RB5		OUT-2
39	RB6/PGC	PGC	
40	RB7/PGD	PGD	

El plano esquemático de conexión se describe en la Figura 19.13 a, b.

libros.ep-electropc.com

a)



b)

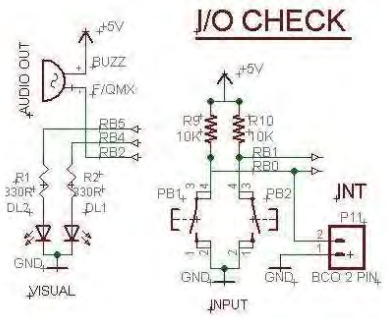


Figura 19.13 Plano esquemático de PUERTO B, leds, pulsadores y buzzer.

En el programa siguiente, en modo de *run*, se prueban las entradas de los pulsadores **INPUT-1** e **INPUT-2**, las salidas son los leds OUT-1 y OUT-2 y el buzzer (BUZZ). Cada vez que se oprime el pulsador INPUT-1, se enciende el led OUT-1 y pita el buzzer (BUZZ). Cada vez que se oprime el pulsador INPUT-2, se enciende el led OUT-2 y pita el buzzer (BUZZ). Si se oprimen el pulsador INPUT-1 e INPUT-2, se enciende el led OUT-1 y OUT-2, y no pita el buzzer BUZZ. En nuestra página Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\ PUERTO_B_1 \ PUERTO_B_1 .asm.

libros.ep-electropc.com

```
        List p = 16f877a          ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

            org 0                           ; vector de reset
Start:
; configura e inicia PORTB
            movlw   b'11001111'     ; todo apagado, leds (en 0), buzzer (en 1) y Pulsadores (en 0)
            movwf   PORTB

            bsf     STATUS,RP0      ; seleccione Banco 1
            movlw   b'11001011'     ; leds, buzzer y pulsadores
            movwf   TRISB           ; programe puerto B
            bcf     STATUS,RP0      ; seleccione Banco 0

MainLoop:
            movlw   b'11001100';     todo apagado, leds (en 0), buzzer (en 1) y Pulsadores (en 0)

            btfss   PORTB,0         ; Pulsador 1 oprimido? no, salte
            xorlw   b'00010100'     ; sí, prende led Out-1 RB4 y buzzer RB2

            btfss   PORTB,1         ; Pulsador 2 oprimido? no, salte
            xorlw   b'00100100'     ; sí, prende led Out-2 RB5 y buzzer RB2

            movwf   PORTB           ; ponga en puerto los cambios
            goto    MainLoop        ; haga ciclo

            end
```

En la Tabla 19.10 se encuentra una secuencia de los posibles valores de los dispositivos (leds, pulsadores, *buzzer*).

Tabla 19.10 Secuencias de leds, pulsadores y buzzer.

SECUENCIA PRÁCTICA PUERTO B										
	PULSADORES				LEDS Y BUZZER					
SECUENCIA	INPUT-1	1/0	INPUT-2	1/0	OUT-1	1/0	OUT-1	1/0	BUZZ	1/0
1	SUELTO	1	SUELTO	1	OFF	0	OFF	0	OFF	0
2	ORPIMIDO	0	SUELTO	1	ON	1	OFF	0	ON	1
3	SUELTO	1	SUELTO	1	OFF	0	OFF	0	OFF	0
4	SUELTO	1	ORPIMIDO	0	OFF	0	ON	1	ON	1
5	ORPIMIDO	0	ORPIMIDO	0	ON	1	ON	1	OFF	0
6	SUELTO	1	SUELTO	1	OFF	0	OFF	0	OFF	0

Este programa, en modo de *run*, puede verificar todas las secuencias que se presentan a continuación con fotos de los dispositivos involucrados. Las secuencias 1 y 2 son: todo apagado y luego el pulsador INPUT-1 oprimido que genera el encendido led OUT-1, en la Figura 19.14a, b.

libros.ep-electropc.com

a) b)

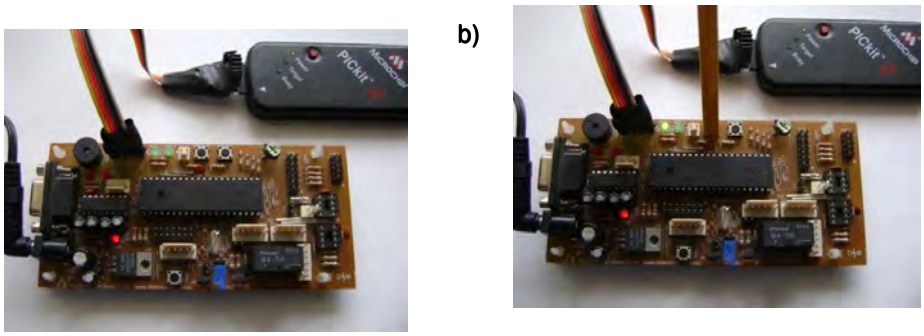


Figura 19.14 Puerto B, led OUT-1, pulsador INPUT-1.

Las secuencias siguientes 3 y 4 son: todo apagado y luego el pulsador INPUT-2 oprimido que genera el encendido led OUT-2, en la Figura 19.15a, b.

a) b)

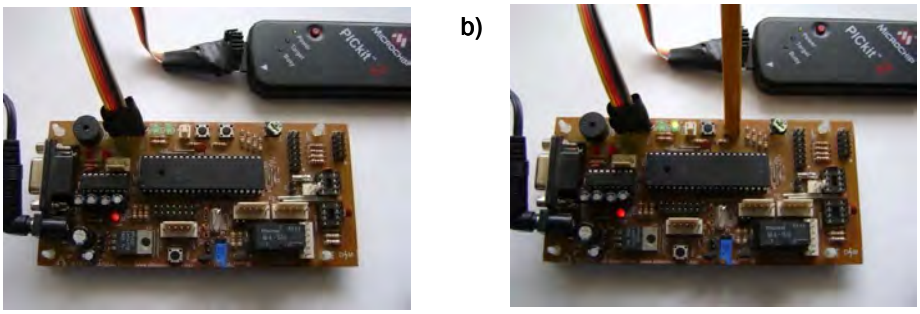


Figura 19.15 Puerto B, led OUT-2, pulsador INPUT-2.

Las últimas secuencias 5 y 6 son: los pulsadores INPUT-1 e INPUT-2 oprimidos generan el encendido de los leds OUT-1 y OUT-2, para luego soltar los pulsadores y se apagan los leds, tal como muestra en la Figura 19.16a, b.

a) b)

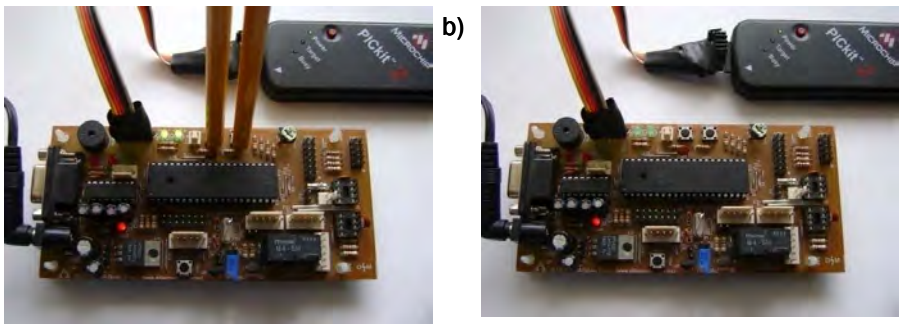


Figura 19.16 Puerto B, led OUT-1-2, pulsador INPUT-1-2

libros.ep-electropc.com

En nuestra página Web puede encontrar otro ejemplo con este Puerto B con retardos y subrutinas en la carpeta **X:\Ejemplos_ASM**, la que debe copiar en el disco para encontrarla como **C:\Ejemplos_ASM\ PUERTO_B_1 \ PUERTO_B_1A.asm**.

19.3 PUERTO C

El PORTC es de 8 bits de ancho y bidireccional, y es denominado RC<7:0>.

19.3.1 DIAGRAMA CIRCUITAL

En las Figuras 19.17 y 19.18 vemos dos tipos de configuraciones de pines: **RC<7:5> <2:0> y RC<4:3>**.

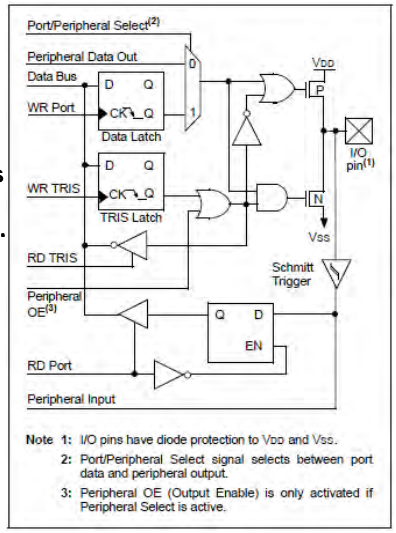


Figura 19.17 Configuraciones de pines: RC<7:5> <2:0>.

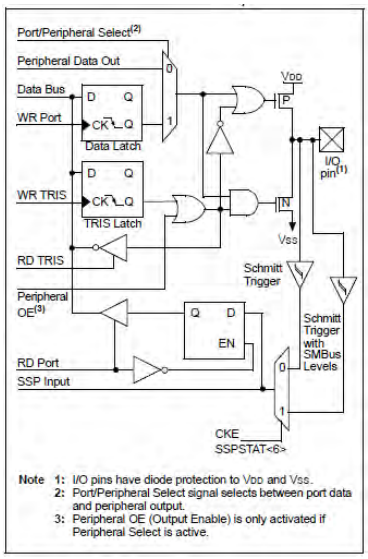


Figura 19.18 Configuraciones de pines: RC<4:3>.

libros.ep-electropc.com

19.3.2 FUNCIONAMIENTO

El Puerto C es de 8 bits de ancho y bidireccional. La correspondiente dirección de pines es el registro **TRISC**. El Puerto C es compartido con varios periféricos, como muestra la Tabla 19.12, y en sus pines tiene entradas de buffer *Schmitt Trigger*. Cuando habilitamos algún periférico se debe tener cuidado al definir los pines en TRISC, debido a que algunos periféricos sobrescriben el TRISC para configurar el pin como entrada o salida, así que hay que referirse al periférico en particular para la propia configuración del pin.

19.3.3 REGISTROS ASOCIADOS

En la Tabla 19.11 se describen las funciones de los pines del Puerto C, y los registros asociados al Puerto C, en la Tabla 19.12.

Tabla 19.11 Funciones del Puerto C.

Name	Bit#	Buffer Type	Function
RC0/T1OSO/T1CKI	bit 0	ST	Input/output port pin or Timer1 oscillator output/Timer1 clock input.
RC1/T1OSI/CCP2	bit 1	ST	Input/output port pin or Timer1 oscillator input or Capture2 input/ Compare2 output/PWM2 output.
RC2/CCP1	bit 2	ST	Input/output port pin or Capture1 input/Compare1 output/ PWM1 output.
RC3/SCK/SCL	bit 3	ST	RC3 can also be the synchronous serial clock for both SPI and I^2C modes.
RC4/SDI/SDA	bit 4	ST	RC4 can also be the SPI data in (SPI mode) or data I/O (I^2C mode).
RC5/SDO	bit 5	ST	Input/output port pin or Synchronous Serial Port data output.
RC6/TX/CK	bit 6	ST	Input/output port pin or USART asynchronous transmit or synchronous clock.
RC7/RX/DT	bit 7	ST	Input/output port pin or USART asynchronous receive or synchronous data.

Legend: ST = Schmitt Trigger input

Tabla 19.12 Resumen de registros asociados con el Puerto C.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
07h	PORTC	RC7	RC6	RC5	RC4	RC3	RC2	RC1	RC0	xxxx xxxx	uuuu uuuu
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged

En la Tabla 19.13 observamos los diferentes periféricos que comparte el Puerto C. Cada pin tiene sus distintos nombres y el módulo periférico al que corresponde.

libros.ep-electropc.com

Tabla 19.13 Periféricos que comparte el Puerto C.

			PERIFERICOS	
			SCI	SCI-1
			SPI	I2C
				PWM
PIN	NOMBRE PIN	PUERTO	TIMER	TIMER
15	RC0/T1OSO/T1CKI	RC0	T1OSO	T1CKI
16	RC1/T1OSI/CCP2	RC1	T1OSI	CCP2
17	RC2/CCP1	RC2		CCP1
18	RC3/SCK/SCL	RC3	SCK	SCL
23	RC4/SDI/SDA	RC4	SDI	SDA
24	RC5/SDO	RC5	SDO	
25	RC6/TX/CK	RC6	TX	CK
26	RC7/RX/DT	RC7	RX	DT

19.3.4 PROCEDIMIENTO DE USO

Para el procedimiento de uso, siga los pasos siguientes:

- Ponga el valor de inicialización en el puerto PORTC (07h).
- Ponga la dirección de los pines en el puerto TRISC (87h).
- Lea o escriba datos en el puerto PORTC (07h).

A continuación le damos el código de un ejemplo:

- Ponga el valor de inicialización en el puerto PORTC (07h):

```
movlw   b'111111000'    ;RC <7:3> entradas, RC7 = RC6 = RC5 = RC4 = RC3 = 1
                        ;RC <2:0> salidas, RC2 = RC0 = 1, RC1 = 0
movwf   PORTC           ;ponga en puerto C
```

- Ponga la dirección de los pines en el puerto TRISC (87h):

```
bsf     STATUS,RP1      ;banco 1
movlw   b'111111000'    ;RC <7:3> entradas, RC7 = RC6 = RC5 = RC4 = RC3 = 1
                        ;RC <2:0> salidas, RC2 = RC0 = 1, RC1 = 0
movwf   TRISC           ; configura puerto C
```

- Lea o escriba los datos en el puerto PORTC (07h):

```
movf    PORTC,W         ;lea puerto C y póngalo en W
movwf   PORTC           ; escriba W en puerto C
```


libros.ep-electropc.com

19.3.5 PRACTICA

Para hacer un ejemplo del Puerto C usando la tarjeta del sistema de evaluación **PIC16F8XX**, habría que usar varios conectores **P3, P4** y **P5**; usamos el **P4** que contiene los pines **RC0, RC1, RC2**. Se deben tener precauciones si hay otros dispositivos conectados a los pines. Estos pines se tratan ampliamente en la **Parte 7: Comunicaciones,** debido a que en este Puerto C están todos los pines del módulo de comunicaciones **I2C, SPI** y **USART**. Esta práctica consiste en prender y apagar los pines RC0, RC1, RC2, titilando en la tarjeta **PIC16F8XX**; el montaje para esta práctica es como se muestra en la Figura 19.19.

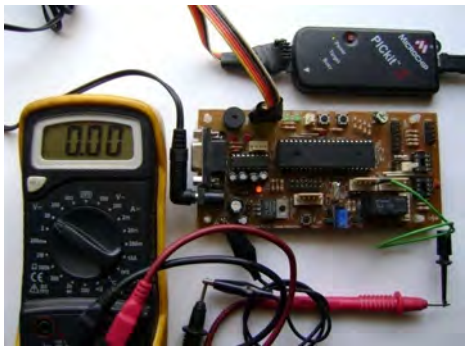


Figura 19.19 Práctica de montaje Puerto C.

Los pines que usaremos en esta práctica están detallados junto al conector **P4** en la tarjeta **PIC16F8XX**, como se aprecia en la Figura 19.20.

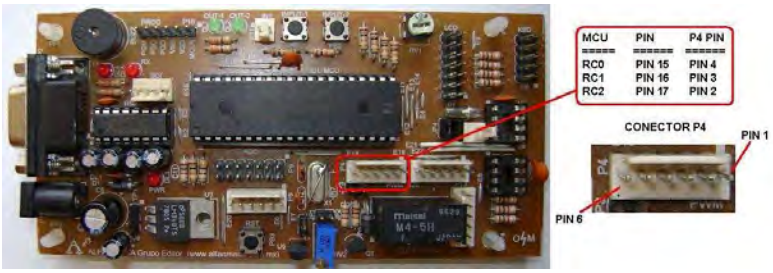


Figura 19.20 Práctica de montaje Puerto C.

Los pines y su función que ocuparemos en el programa siguiente se describen en la Tabla 19.14.

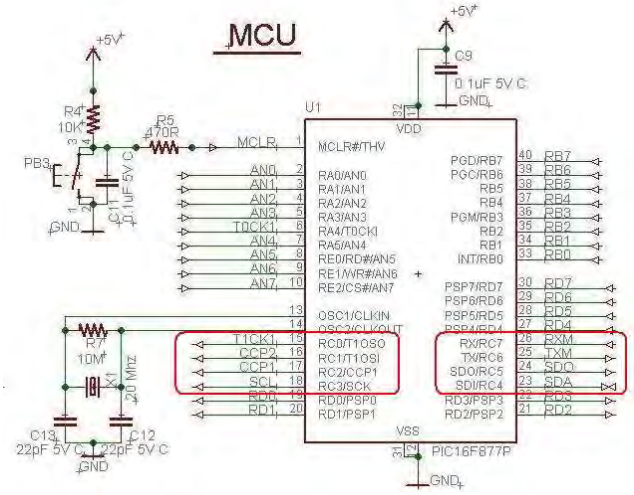
libros.ep-electropc.com

Tabla 19.14 Pines y función del Puerto C.

CONEXIÓN PUERTO C CONECTOR P5					
MCU		P4			
PIN	NOMBRE	PIN	MOD	FUNCIÒN	
15	RC0/T1OSO/T1CKI	6	+5V	+5V	
16	RC1/T1OSI/CCP2	5	TOCK1		
17	RC2/CCP1	4	T1CK1	RC0	
18	RC3/SCK/SCL	3	CCP2	RC1	
23	RC4/SDI/SDA	2	CCP1	RC2	
24	RC5/SDO	1	GND	GND	
25	RC6/TX/CK				
26	RC7/RX/DT				

El plano esquemático de conexión se describe en la Figura 19.21 a, b.

a)



b)

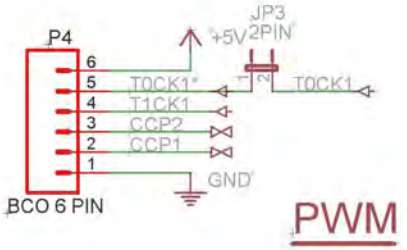


Figura 19.21 Plano esquemático del Puerto C, conector P4.

libros.ep-electropc.com

El programa siguiente, en modo paso a paso, prende **'1'** y apaga **'0'** los pines del Puerto C, para el ejemplo **RC2 = RC1 = RC0** configurados como salidas. En nuestra página Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\ PUERTO_C_1 \ PUERTO_C_1.asm.

```
        List p = 16f877a        ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

        cblock 0x20             ; desde la posición 20 en RAM, separa espacio para estas variables

        org 0                   ; vector de reset
Start:
        clrf    PORTC           ; apague PUERTO C

        bsf     STATUS,RP0      ; seleccione Banco 1
        movlw   b'11111000'     ; salidas RC0 = RC1 = RC2, las otras como entradas
        movwf   TRISC           ; configure PUERTO C
        bcf     STATUS,RP0      ; seleccione Banco 0

MainLoop:

ON:
        comf    PORTC,F         ; prenda PUERTO
OFF:
        Clrf    PORTC           ; apague PUERTO

        Goto    MainLoop        ; haga ciclo

        end
```

Haciendo una depuración del programa anterior (corriéndolo paso a paso), se observa cómo cambia de valores el pin **2** del conector **P4** conectado a **RC2**, en la Figura 19.22a, b; estos valores observados en el multímetro corresponden a los pasos **ON:** y **OFF:** del programa anterior.

a) b)

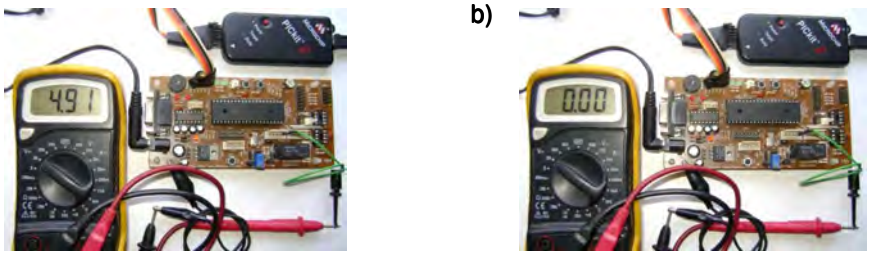


Figura 19.22 Secuencias del Puerto CON/OFF PIN RC2.

libros.ep-electropc.com

En nuestra página Web puede encontrar este ejemplo con retardos y subrutinas en la carpeta **X:\Ejemplos_ASM**, la que debe copiar en el disco para encontrarla como **C:\Ejemplos_ASM\ Puerto_C_1 \ Puerto_C_1A.asm**.

19.4 PUERTO D

El PORTD es de 8 bits de ancho y bidireccional, y es denominado RD<7:0>.

19.4.1 DIAGRAMA CIRCUITAL

En la Figura 19.23 apreciamos un tipo de configuración de pines: RD<7:0>.

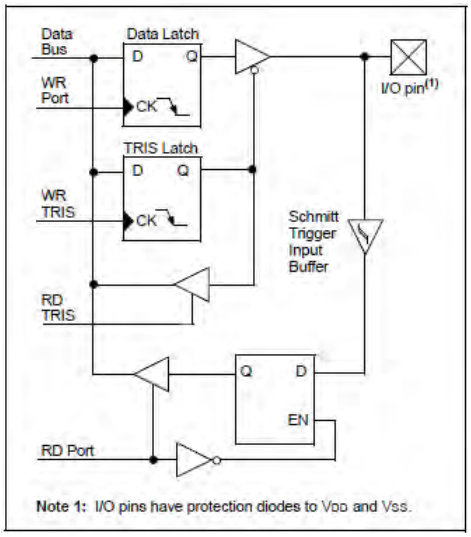


Figura 19.23 Configuración de pines RD<7:0>.

19.4.2 FUNCIONAMIENTO

El Puerto D es de 8 bits de ancho y bidireccional. La dirección de pines correspondiente es el registro TRISD. También puede ser configurado como puerto paralelo, poniendo a '1' el bit PSPMODE en el registro TRISE<4>; en este modo los pines son de entrada con *buffer* TTL.

libros.ep-electropc.com

19.4.3 REGISTROS ASOCIADOS

En la Tabla 19.15 se describen las funciones de los pines del Puerto D y en la Tabla 19.16 los registros asociados con el Puerto D.

Tabla 19.15 Funciones de los pines del Puerto D.

Name	Bit#	Buffer Type	Function
RD0/PSP0	bit 0	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 0.
RD1/PSP1	bit 1	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 1.
RD2/PSP2	bit2	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 2.
RD3/PSP3	bit 3	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 3.
RD4/PSP4	bit 4	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 4.
RD5/PSP5	bit 5	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 5.
RD6/PSP6	bit 6	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 6.
RD7/PSP7	bit 7	ST/TTL(1)	Input/output port pin or Parallel Slave Port bit 7.

Legend: ST = Schmitt Trigger input, TTL = TTL input
Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

Tabla 19.16. Registros asociados con el Puerto D.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
08h	PORTD	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx xxxx	uuuu uuuu
88h	TRISD	PORTD Data Direction Register								1111 1111	1111 1111
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction Bits			0000 -111	0000 -111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTD.

En la Tabla 19.17 observamos los diferentes periféricos que comparte el Puerto D. Cada pin tiene sus distintos nombres y el módulo periférico al que corresponde.

Tabla 19.17 Periféricos que comparte el Puerto D.

			PERIFERICOS
PIN	**NOMBRE PIN**	**PUERTO**	PARLL
19	RD0/PSP0	RD0	PSP0
20	RD1/PSP1	RD1	PSP1
21	RD2/PSP2	RD2	PSP2
22	RD3/PSP3	RD3	PSP3
27	RD4/PSP4	RD4	PSP4
28	RD5/PSP5	RD5	PSP5
29	RD6/PSP6	RD6	PSP6
30	RD7/PSP7	RD7	PSP7

libros.ep-electropc.com

19.4.4 PROCEDIMIENTO DE USO

Para el proceso de uso siga los pasos siguientes:

- Ponga el valor de inicialización en el PORTD (08h).
- Ponga la dirección de los pines en el puerto TRISD (88h).
- Lea o escriba datos en el PORTD (08h).

A continuación le damos el código de un ejemplo:

1. Ponga el valor de inicialización en el PORTD (08h):

```
movlw   b`11110000      ;RD <7:4> entradas, RD <3:0> salidas
movwf   PORTD           ;ponga el latch del puerto D
```

2. Ponga la dirección de los pines en el puerto TRISD (88h):

```
bsf     STATUS,RP1      ;banco 1

movlw   b`11110000      ;RD <7:4> entradas, RD <3:0> salidas
movwf   TRISD
```

3. Lea o escriba datos en el PORTD (08h):

```
bcf     STATUS,RP0      ;banco 0

movf    PORTD,0         ;lea puerto D y póngalo en W
```

19.4.5 PRÁCTICA

En el ejemplo siguiente del Puerto D se muestra cómo se rota un bit en el Puerto D, con el módulo **MOD_8LEDS (2009-05-06)** y con la tarjeta **PIC16F8XX**; el montaje para esta práctica es el que se observa en la Figura 19.24.



Figura 19.24 Práctica de montaje Puerto D.

libros.ep-electropc.com

En la Figura 19.25 observamos en detalle el conector **P7** en la tarjeta **PIC16F8XX**.

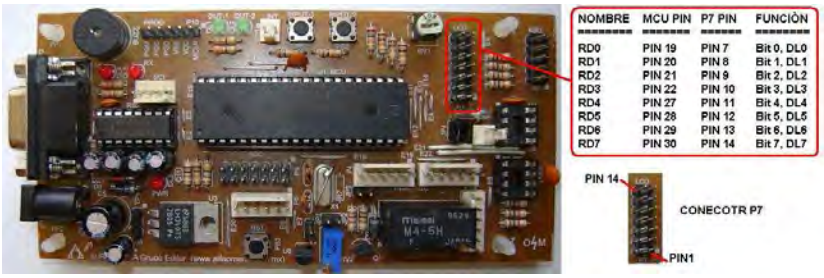


Figura 19.25 Puerto D, conector P 7.

En la Tabla 19.18 se definen los pines y las funciones del conector **P7** con los que trabajaremos.

Tabla 19.18 Pines y funciones del Puerto D

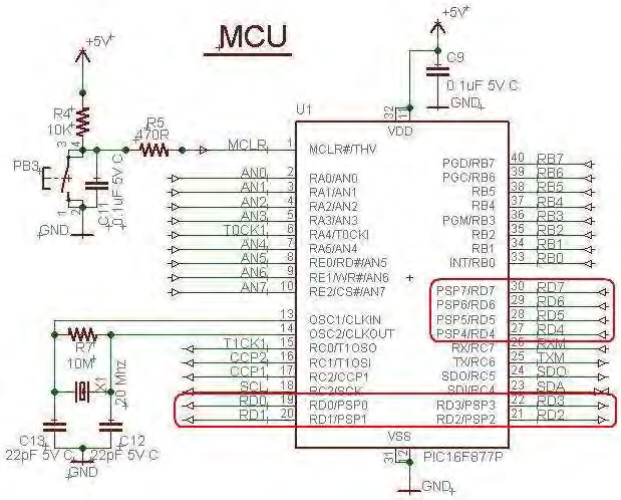
CONEXION PUERTO D
CONECTOR P7 A MOD_8LEDS (2009-05-06)

MCU		P7		
PIN	NOMBRE	PIN	LCD	FUNCIÒN
		1	GND	GND
		2	+5V	+5V
		3	VOF	
		4	-RD	
		5	-WR	
		6	-CS	
19	RD0/PSP0	7	PSP0	BIT0. DL0
20	RD1/PSP1	8	PSP1	BIT1. DL1
21	RD2/PSP2	9	PSP2	BIT2. DL2
22	RD3/PSP3	10	PSP3	BIT3. DL3
27	RD4/PSP4	11	PSP4	BIT4. DL4
28	RD5/PSP5	12	PSP5	BIT5. DL5
29	RD6/PSP6	13	PSP6	BIT6. DL6
30	RD7/PSP7	14	PSP7	BIT7. DL7

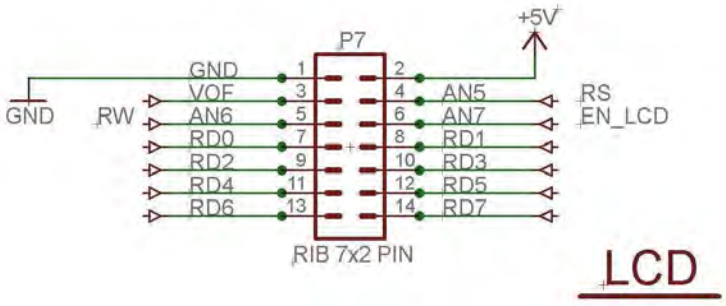
El plano esquemático de conexión se describe en la Figura 19.26a, b, c.

libros.ep-electropc.com

a)



b)



c)

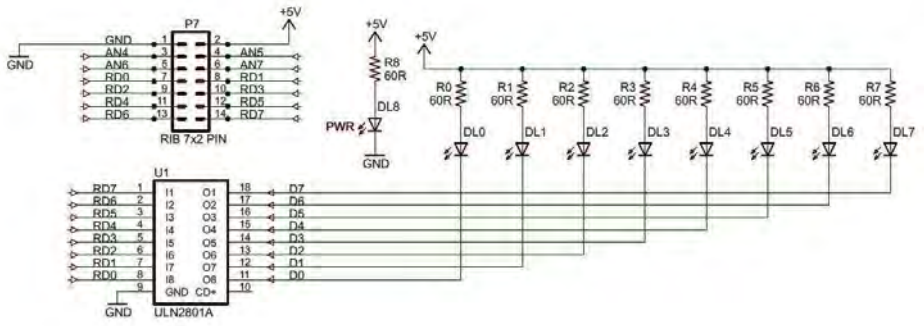


Figura 19.26 Plano esquemático del Puerto D, conector P7.

libros.ep-electropc.com

En la Figura 19.27a, b observamos el módulo **MOD_8LEDS** (2009-05-06).

a)



b)

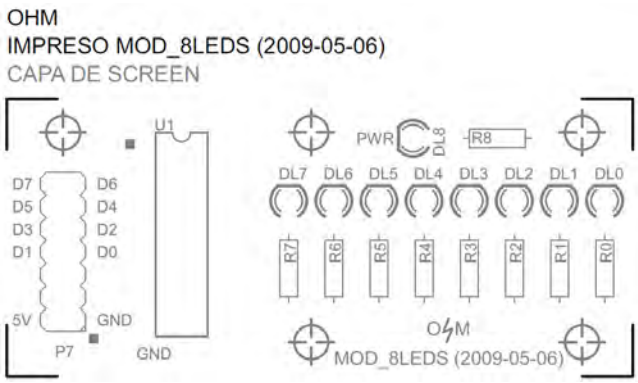


Figura 19.27 Módulo MOD_8LEDS (2009-05-06).

Se debe conectar el conector **P7** de la tarjeta **MOD_8LEDS (2009-05-06)** con el conector **P7** de la tarjeta **PIC16F8XX**.En el ejemplo siguiente, en modo paso a paso, los bits del Puerto D van rotando hacia la izquierda, y se podrán observar con el correspondiente led DLx en el módulo MOD_8LEDS (2009-05-06), de forma que los bits 0 a 7 son los pines **RD0:RD7** del Puerto D, y encienden los bits DL0 a DL7.

En nuestra página Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\ PUERTO_C_1 \ PUERTO_C_1.asm.

```
        List p = 16f877a          ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

                cblock 0x20             ; desde la posición 20 en RAM, separa espacio para estas variables
contador                ; contador de bits
```


libros.ep-electropc.com

```
            endc

            org 0
Start:
            clrf        PORTD               ; apague PUERTO D

            bsf         STATUS,RP0          ; seleccione Banco 1
            movlw       b'00000111'         ; todos los pines digitales
            movwf       ADCON1              ; registro de configuración A/D

            clrf        TRISD               ; configure PUERTO D
            bcf         STATUS,RP0          ; seleccione Banco 0

Main_ini_dig
            movlw       b'00000001'         ; bit 1 prendido
            movwf       contador            ; cargue en contador
            movwf       PORTD               ; cargue a PUERTO D

Main_dig
            bcf         STATUS,C            ; apaga bit C
            rlf         contador,W          ; rote a la izquierda -> W
            movwf       contador            ; cargue W -> contador
            btfsc       STATUS,C            ; C = 0? sí, , continúe siguiente bit, salte
            goto        Main_ini_dig        ; no, último bit

            movwf       PORTD               ; ponga en PUERTO D
            goto        Main_dig            ; continúe ciclo

end
```

Siguiendo paso a paso el programa anterior observaremos la secuencia de bits que se aprecia en la Tabla 19.19.

Tabla 19.19 Secuencia de bits y leds DLX.

	MSB							LSB	PUERTO D
	7	6	5	4	3	2	1	0	BITS
SECUENCIA	DL7	DL6	DL5	DL4	DL3	DL2	DL1	DL0	LEDS DLx
0	0	0	0	0	0	0	0	0	00h
1	0	0	0	0	0	0	0	1	01h
2	0	0	0	0	0	0	1	0	02h
3	0	0	0	0	0	1	0	0	04h
4	0	0	0	0	1	0	0	0	08h
5	0	0	0	1	0	0	0	0	10h
6	0	0	1	0	0	0	0	0	20h
7	0	1	0	0	0	0	0	0	40h
8	1	0	0	0	0	0	0	0	80h

También puede observar las fotos del módulo **MOD_8LEDS (2009-05-06)** de 8 leds en cada una de las secuencias anteriormente vistas, como se muestran en la Figura 19.28 a, b, c, d, e, f, g.

libros.ep-electropc.com

a)



b)



c)



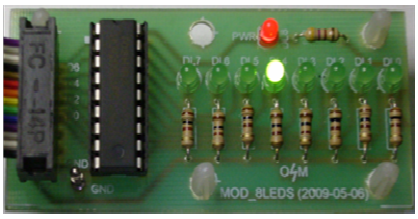
d)



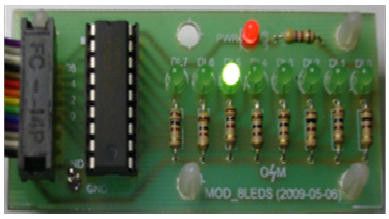
e)



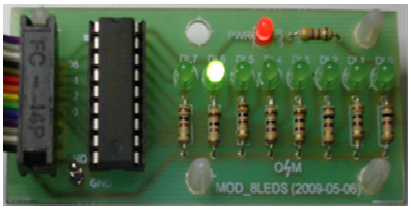
f)



g)



h)



i)



Figura 19.28 Rotación de bits y leds.

libros.ep-electropc.com

En nuestra página Web puede encontrar este ejemplo con retardos y subrutinas en la carpeta **X:\Ejemplos_ASM**, la que debe copiar en el disco para encontrarla como **C:\Ejemplos_ASM\ PUERTO_C_1 \ PUERTO_C_1A.asm**.

19.5 PUERTO E

El PORTE es de 3 bits de ancho y bidireccional, y es denominado **RE<2:0>**.

19.5.1 DIAGRAMA CIRCUITAL

En la Figura 19.29 apreciamos un tipo de configuración de pines del Puerto E: RE<2:0>.

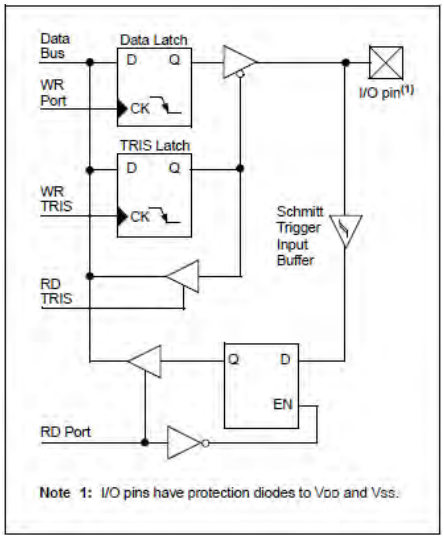


Figura 19.29 Configuración de pines del Puerto E: RE<2:0>.

19.5.2 FUNCIONAMIENTO

El **Puerto E** (PORTE) es de 3 bits de ancho y bidireccional. La correspondiente dirección de pines es el registro **TRISE**. El PUERTO E puede convertirse en los bits de control del puerto paralelo del MCU, cuando ponemos a uno el bit PSPMODE en el registro TRISE<4>; el usuario debe asegurarse de que los pines están configurados como entradas. También necesita asegurarse de que ADCON1 debe ser configurado como entradas digitales. En este modo los pines son entradas de buffer TTL.

libros.ep-electropc.com

En la Tabla 19.23 vemos que el registro TRISE muestra cómo son las funciones de control para la operación de puerto paralelo. Los pines del **Puerto E** también se comparten con entradas analógicas; al seleccionar como entrada analógica estos pines son leídos como "0". El registro **TRISE** controla la dirección de los pines aun cuando sean entradas analógicas; el usuario debe asegurarse de que están configurados como entradas cuando las use como entradas analógicas.

```
NOTA: Cuando hay power-on-reset, estos pines son configurados como
entradas y leídos como "0".
```

19.5.3 REGISTROS ASOCIADOS

En la Tabla 19.20 se describen las funciones de los pines del **Puerto E**, y en la Tabla 19.21 los registros asociados a este mismo puerto. En la tabla 19.22 el registro **TRISE** muestra cómo son las funciones de control para la operación de puerto paralelo.

Tabla 19.20 Funciones de los pines del Puerto E.

Name	Bit#	Buffer Type	Function
RE0/$\overline{RD}$/AN5	bit 0	ST/TTL(1)	I/O port pin or read control input in Parallel Slave Port mode or analog input: $\overline{RD}$ 1 = Idle 0 = Read operation. Contents of PORTD register are output to PORTD I/O pins (if chip selected).
RE1/$\overline{WR}$/AN6	bit 1	ST/TTL(1)	I/O port pin or write control input in Parallel Slave Port mode or analog input: $\overline{WR}$ 1 = Idle 0 = Write operation. Value of PORTD I/O pins is latched into PORTD register (if chip selected).
RE2/$\overline{CS}$/AN7	bit 2	ST/TTL(1)	I/O port pin or chip select control input in Parallel Slave Port mode or analog input: $\overline{CS}$ 1 = Device is not selected 0 = Device is selected

Legend: ST = Schmitt Trigger input, TTL = TTL input
Note 1: Input buffers are Schmitt Triggers when in I/O mode and TTL buffers when in Parallel Slave Port mode.

Tabla 19.21 Registros asociados con el Puerto E.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
09h	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PORTE.

libros.ep-electropc.com

Tabla 19.22 Registro TRISE.

R-0	R-0	R/W-0	R/W-0	U-0	R/W-1	R/W-1	R/W-1
IBF	OBF	IBOV	PSPMODE	—	Bit 2	Bit 1	Bit 0
bit 7							bit 0

Parallel Slave Port Status/Control Bits:

bit 7 **IBF:** Input Buffer Full Status bit
1 = A word has been received and is waiting to be read by the CPU
0 = No word has been received

bit 6 **OBF**: Output Buffer Full Status bit
1 = The output buffer still holds a previously written word
0 = The output buffer has been read

bit 5 **IBOV**: Input Buffer Overflow Detect bit (in Microprocessor mode)
1 = A write occurred when a previously input word has not been read (must be cleared in software)
0 = No overflow occurred

bit 4 **PSPMODE**: Parallel Slave Port Mode Select bit
1 = PORTD functions in Parallel Slave Port mode
0 = PORTD functions in general purpose I/O mode

bit 3 **Unimplemented**: Read as '0'

PORTE Data Direction Bits:

bit 2 **Bit 2**: Direction Control bit for pin RE2/$\overline{CS}$/AN7
1 = Input
0 = Output

bit 1 **Bit 1**: Direction Control bit for pin RE1/$\overline{WR}$/AN6
1 = Input
0 = Output

bit 0 **Bit 0**: Direction Control bit for pin RE0/$\overline{RD}$/AN5
1 = Input
0 = Output

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

En la Tabla 19.23 se describe el registro **ADCON1**.

Tabla 19.23 Registro ADCON1.

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

ADFM: A/D Result Format Select bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

ADCS2: A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

Unimplemented: Read as '0'

PCFG3:PCFG0: A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: On any device Reset, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

libros.ep-electropc.com

En la Tabla 19.24 vemos los diferentes periféricos que comparte el **Puerto E**. Cada pin tiene sus distintos nombres y el módulo periférico al que corresponde.

Tabla 19.24 Periféricos que comparte el Puerto D.

			PERIFERICOS	
PIN	NOMBRE PIN	PUERTO	PARLL	A/D
8	RE0/RD/AN5	RE0	RD	AN5
9	RE1/WR/AN6	RE1	WR	AN6
10	RE2/CS/AN7	RE2	CS	AN7

19.5.4 PROCEDIMIENTO DE USO

En el proceso de uso siga los pasos siguientes:

- Ponga el valor de inicialización en el puerto PORTE (09h).
- Ponga la dirección de los pines en el puerto TRISE (89h).
- Habilite los puertos digitales en el registro ADCON1 (9Fh).
- Lea o escriba datos en el PORTE (09h).

A continuación le damos el código de un ejemplo:

1. Ponga el valor de inicialización en el puerto PORTE (09h):

```
clrf        PORTE              ;ponga en 0 el latch del puerto E
```

2. Ponga la dirección de los pines en el puerto TRISE (89h):

```
bsf         STATUS,RP0         ;banco 1

movlw       b`11111000         ; RE <7:3> entradas,RE <2:0> salidas RE2 = RE1 = RE0 = 0
movwf       TRISE              : configure puerto E
```

3. Habilite los puertos Digitales en el registro ADCON1 (9Fh):

```
movlw       b'00000111'        ; todos los pines digitales
movwf       ADCON1             ; registro de configuración A/D
```

4. Lea o escriba datos en el puerto PORTD (08h):

```
movf        PORTE,W            ;lea puerto E y póngalo en W
movwf       PORTE              ;escriba W en puerto E
```


libros.ep-electropc.com

19.5.5 PRÁCTICA

En el ejemplo siguiente se reúnen los Puertos E y D, para mostrar los 7 segmentos de un display con 4 dígitos con el módulo **MOD_7S_4D_L (2009-05-06)** y con la tarjeta **PIC16F8XX**; el montaje para esta práctica es el que se observa en la Figura 19.30.

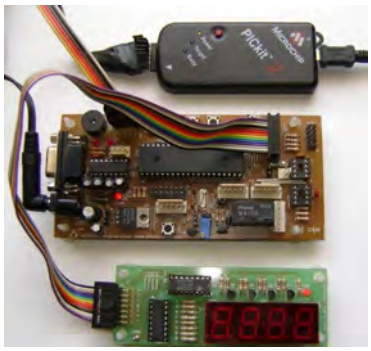


Figura 19.30 Práctica de Montaje Puerto E.

En la Figura 19.31 observamos con detalle el conector **P7** en la tarjeta **PIC16F8XX**.



Figura 19.31 Puerto E, conector P7.

Los pines utilizados en esta práctica están definidos en la Tabla 19.25.

CONEXIÓN PUERTO E
CONECTOR P7 A MOD_7S_4D_L (2009-05-06)

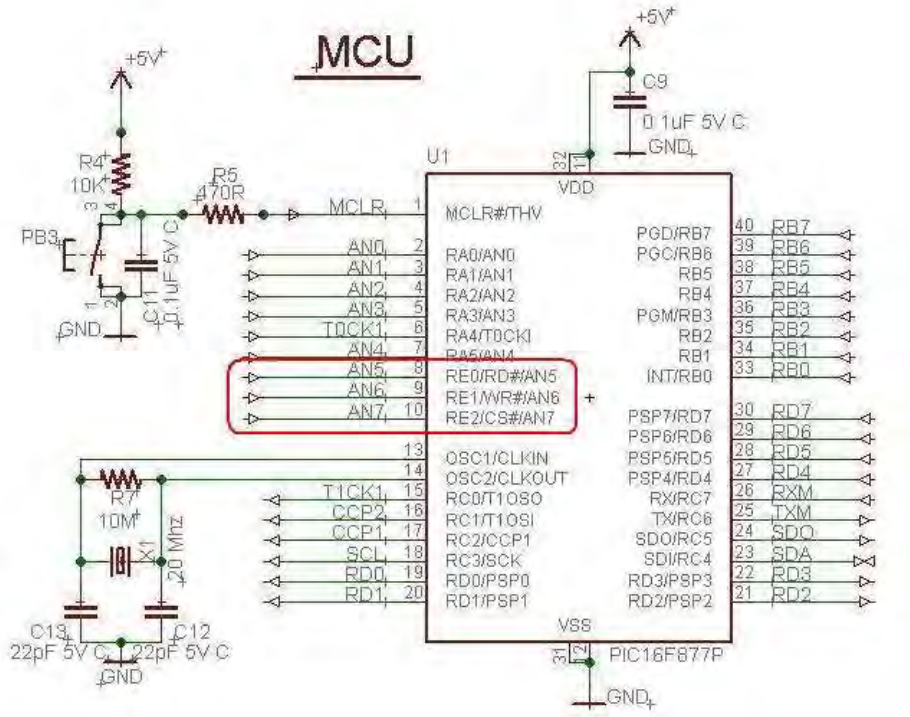
MCU		P7		
PIN	NOMBRE	PIN	LCD	FUNCIÓN
		1	GND	GND
		2	+5V	+5V
		3	VOF	
8	RE0/RD/AN5	4	-RD	DIG_2
9	RE1/WR/AN6	5	-WR	DIG_3
10	RE2/CS/AN7	6	-CS	DIG_4
19	RD0/PSP0	7	PSP0	SEG_A
20	RD1/PSP1	8	PSP1	SEG_B
21	RD2/PSP2	9	PSP2	SEG_C
22	RD3/PSP3	10	PSP3	SEG_D
27	RD4/PSP4	11	PSP4	SEG_E
28	RD5/PSP5	12	PSP5	SEG_F
29	RD6/PSP6	13	PSP6	SEG_G
30	RD7/PSP7	14	PSP7	SEG_P

Tabla 19.25 Pines y función del Puerto E.

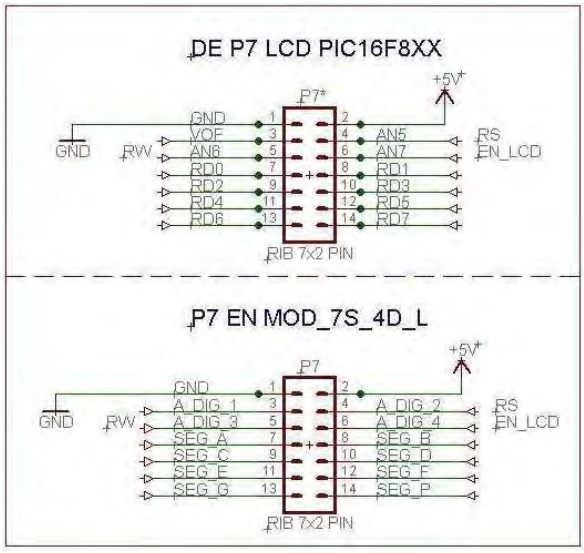
libros.ep-electropc.com

A continuación (Figura 19.32 a,b,c) encontramos descrito el plano esquemático de conexión.

a)



b)



libros.ep-electropc.com

c)

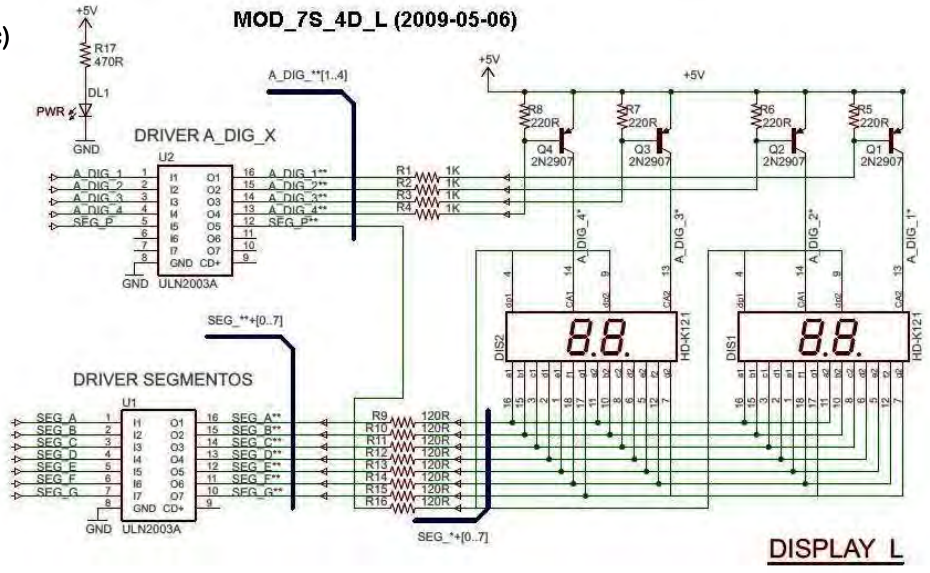
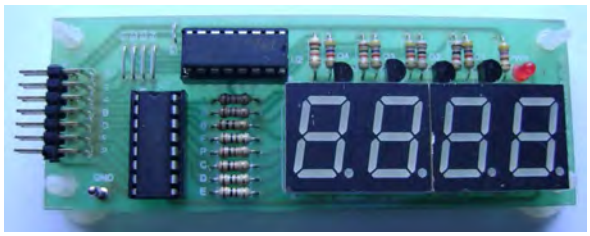


Figura 19.32 Plano esquemático del Puerto E, conector P7.

En la Figura 19.33a, b se muestra la imagen del módulo **MOD_7S_4D_L (2009-05-06)**.

a)



b)

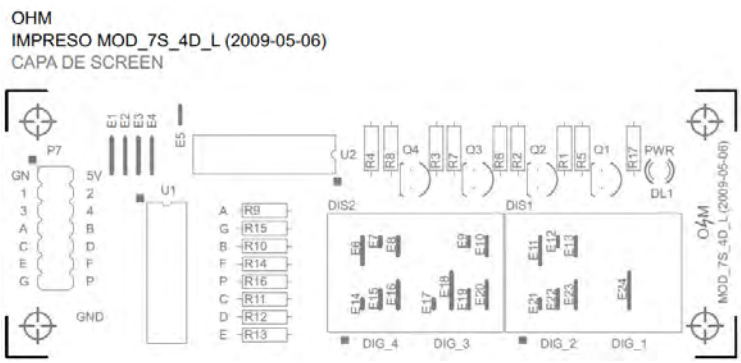


Figura 19.33 Módulo MOD_7S_4D_L (2009-05-06).

libros.ep-electropc.com

El programa siguiente, corriendo paso a paso, conecta un display de 4 dígitos y 7 segmentos, los segmentos (A..P) a los pines del **Puerto D** (0..7), y los dígitos (2..4) al **Puerto E** (0..2). Es de anotar que el dígito 1 no está conectado en esta práctica. Los segmentos (A..P) se irán encendiendo y apagando secuencialmente en el dígito 4, y al llegar al segmento (P) se reinicia el proceso. En la Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\ PUERTO_E_1\ PUERTO_E_1.asm.

```
        List p = 16f877a        ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

        cblock 0x20             ; desde la posición 20 en RAM, separa espacio para estas variables
segmento                        ; dato de segmentos
        endc

#define _bit_DIG_4      2       ; dígito 4 en PORTE, dígitos
        org 0
Start:
        clrf    PORTD           ; apague PUERTO D
        clrf    PORTE           ; apague PUERTO E

        bsf     STATUS,RP0      ; seleccione Banco 1
        movlw   b'00000111'     ; todos los pines digitales
        movwf   ADCON1          ; registro de configuración A/D

        clrf    TRISD           ; configure puerto D
        clrf    TRISE           ; configure puerto E
        bcf     STATUS,RP0      ; seleccione Banco 0

Main_ini_dig
        bsf     PORTE,_bit_DIG_4 ; prenda dígito 4 en PORTE, dígitos
Main_ini_seg
        movlw   b'00000001'     ; bit de segmento A prendido
        movwf   segmento        ; cargue en segmento
        movwf   PORTD           ; cargue a puerto D

Main_seg
        bcf     STATUS,C        ; apaga bit C
        rlf     segmento,W      ; rote a la izquierda -> W
        movwf   segmento        ; cargue W -> segmento
        btfsc   STATUS,C        ; C = 0? sí, continúe
        goto    Main_ini_seg    ; no, vaya a ...

        movwf   PORTD           ; ponga en puerto D
        goto    Main_seg        ; continúe ciclo

        end
```

La secuencia de los segmentos se observa en la Tabla 19.26.

libros.ep-electropc.com

Tabla 19.26 Secuencia de segmentos.

	MSB							LSB	PUERTO D
	7	6	5	4	3	2	1	0	BITS
SECUENCIA	P	G	F	E	D	C	B	A	SEGMENTO
1	0	0	0	0	0	0	0	1	01h
2	0	0	0	0	0	0	1	0	02h
3	0	0	0	0	0	1	0	0	04h
4	0	0	0	0	1	0	0	0	08h
5	0	0	0	1	0	0	0	0	10h
6	0	0	1	0	0	0	0	0	20h
7	0	1	0	0	0	0	0	0	40h
8	1	0	0	0	0	0	0	0	80h

Siguiendo paso a paso el programa anterior, la secuencia de los segmentos se aprecia en el dígito 4, como se muestra en la Figura 19.34.



Figura 19.34 Rotación de segmentos dígito 1.

En nuestra página Web puede encontrar este ejemplo con retardos y subrutinas en la carpeta X:\Ejemplos_ASM, la que debe estar copiada en el disco para encontrarla como C:\Ejemplos_ASM\ PUERTO_E_1\ PUERTO_E_1A.asm.

19.6 PUERTO PARALELO ESCLAVO

El **puerto paralelo esclavo** (**PSP:** *Parallel Slave Port*) es un periférico que usa los pines de dos puertos para su operación. El **Puerto E** como los pines de control, y el Puerto D como los pines de datos propiamente. Se puede interfazar directamente al bus de datos de otro MCU, y éste puede leer y escribir sobre el Puerto D que se comporta como un *latch* de 8 bits.

19.6.1 DIAGRAMA CIRCUITAL

En la Figura 19.35 apreciamos el tipo de configuración de pines del Puerto D: RD<7:0> y RE<2:0>.

libros.ep-electropc.com

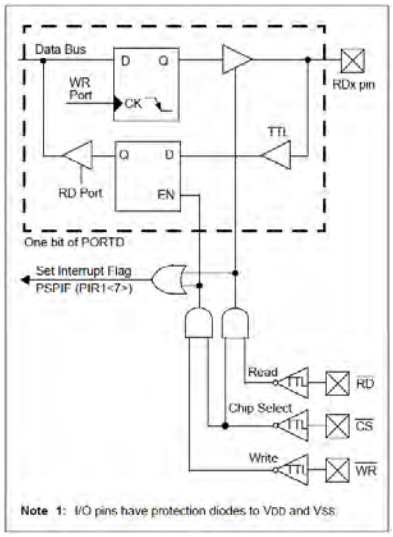


Figura 19.35 Configuración de pines del Puerto D: RD<7:0> y RE<2:0>.

19.6.2 FUNCIONAMIENTO

A través del puerto paralelo podemos interfazar directamente con el bus de datos de un microprocesador externo que sea el maestro, y el nuestro el esclavo, en donde el pin RE0/RD/AN5 es la entrada de lectura RD, RE1/WR/AN6 es la entrada de escritura WR y RE2/CS/AN7 el selector de chip **CS** (*Chip Select*). Para esta configuración el TRISE<2:0> debe ser configurado como entrada en todos los pines. En el módulo A/D también se deben configurar PCFG3:PCFG0 en el registro ADCON1<3:0>, y los pines RE2:RE0 como entradas digitales.

Hay dos latch de 8 bits: uno de salida de datos para escritura y otro de entrada de datos para lectura; observe que es la misma dirección de registro (el PORTD) para ambas operaciones. En este modo el registro TRISD es ignorado, de tal forma que el MCU externo controla la dirección del flujo de los datos.

19.6.2.1 ESCRITURA

En la Figura 19.36 apreciamos un diagrama de tiempos de la operación de escritura. Una escritura ocurre cuando y están en bajo (*low* = "0"), cuando una de las dos o ambas pasan a alto (*high* = "1"), se pone la bandera **IBF** (*Input Buffer Full* TRISE<7>); indicando que la escritura está completa; también se pone la bandera de interrupción **PSPIF** (*Parallel Slave Port Interrupt Flag*). La bandera **IBF** solamente puede aclararse al leer el PORTD (buffer de entrada). La bandera **IBOV** (*Input Buffer Overflow*) en el registro TRISE<5> se pone cuando una segunda escritura se produce y no se hizo lectura del puerto.

libros.ep-electropc.com

19.6.2.2 LECTURA

En la Figura 19.36 observamos un diagrama de tiempos de la operación de lectura. Una lectura ocurre cuando están en bajo (*low* = "0"), inmediatamente la bandera **OBF** (*Output Buffer Full* TRISE<6>) es borrada, indicando que el buffer de salida del Puerto D está esperando ser leído por el bus externo. Cuando cualquiera de los pines pasa a alto (*high* = "1"), también se pone la bandera de interrupción **PSPIF** (*Parallel Slave Port Interrupt Flag*), indicando que la lectura está completa. La bandera OBF permanece en bajo (*low* = "0"), hasta que se escriba sobre el puerto D. Se debe enfatizar que los ciclos de máquina (Q1, Q2, Q3, Q4) deben respetarse para que una secuencia apropiada se realice.

Cuando no hay operación del módulo PSP, las banderas IBF y OBF son mantenidas en bajo (*low* = "0"). Si la bandera IBOV fue previamente puesta (*high* = "12") debe aclararse por software. Una interrupción es generada y almacenada en el bit PSPIF cuando una lectura o una escritura son completadas. La bandera PSPIF debe ser borrada por el software de usuario, y borrando el bit de habilitación de interrupción PSPIE<7> la interrupción puede ser deshabilitada.

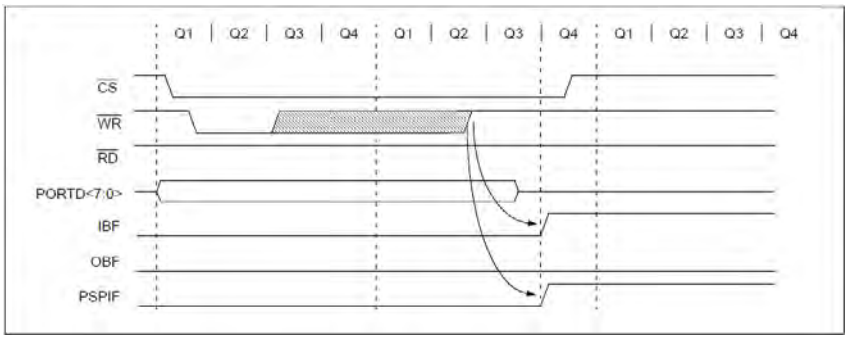


Figura 19.36 Diagrama de tiempos de la operación de escritura.

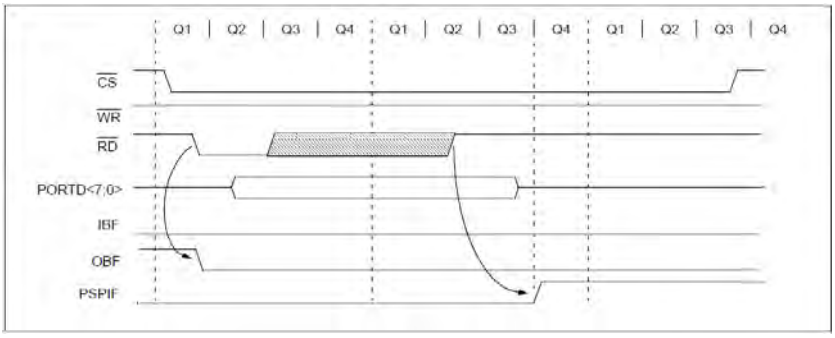


Figura 19.37 Diagrama de tiempos de la operación de lectura.

libros.ep-electropc.com

19.6.3 REGISTROS ASOCIADOS

En la Tabla 19.27 vemos los registros asociados al PSP.

Tabla 19.27 Registros asociados con el Puerto Paralelo Esclavo (PSP).

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
08h	PORTD	Port Data Latch when written; Port pins when read								xxxx xxxx	uuuu uuuu
09h	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu
89h	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Parallel Slave Port.
Note 1: Bits PSPIE and PSPIF are reserved on the PIC16F873A/876A; always maintain these bits clear.

EVALUACIÓN

P.: 1. ¿Qué valores de entrada o salida pueden tener los puertos I/O?
R.:
Valores lógicos 1 o 0.

P.:2. ¿Cómo se nombran los pines de los puertos I/O?
R.:
Se nombran como RXY, donde X = PUERTO (A, B, C, D, E), Y = BIT (7..0).
Por ejemplo, RA4 corresponde al puerto A, pin 4.

P.:3. Los puertos I/O ¿pueden compartir los pines con otros periféricos?
R.:
Sí.

P.:4. ¿Las resistencias de *pull-up* funcionan solamente con pines definidos como salidas?
R.:
Falso, funcionan solamente con pines definidos como entradas.

P.:5. ¿Cuáles son los pasos para el procedimiento de uso de un puerto (con código, p. ej., puerto B)?
R.:
En el proceso de uso siga los pasos siguientes:
1. Ponga el valor de inicialización en el puerto PORTB (06h).
2. Ponga la dirección de los pines en el puerto TRISB (86h).

libros.ep-electropc.com

```
3. Lea o escriba datos en el puerto PORTB (06h).
```

```
A continuación le damos el código de un ejemplo:

1. Ponga el valor de inicialización en el puerto PORTB (06h):
movlw   b'1100 1011'   ;RB <5,4,2> salidas RB5=RB4=RB2=0,
                       ;RB <7,6,3,1,0> entradas, RB7=RB6=RB1=RB0= 1
movwf   PORTB          ; ponga en puerto B

2. Ponga la dirección de los pines en el puerto TRISB (86h):
bsf     STATUS,RP1     ;banco 1
movlw   b'1100 1011'   ;RB <5,4,2> salidas RB5=RB4=RB2=0,
                       ;RB <7,6,3,1,0> entradas, RB7=RB6=RB1=RB0=1
movwf   TRISB

3. Lea o escriba datos en el PORTB (06h):
movf    PORTB,0        ; lea puerto B y póngalo en W.

P.:6. ¿Cuáles son los puertos y pines destinados para el puerto
paralelo?
R.:
Puerto D (pines [7..0]), Puerto E (pines [2..0]).
```

20.0 MÓDULO CONVERSOR ANÁLOGO A DIGITAL (A/D)

Este módulo permite convertir señales continuas del mundo analógico al mundo discreto o digital de 1 y 0; de esta manera la señal analógica se puede manipular dentro del MCU. Conviene anotar que hay un solo conversor A/D que se multiplexa con todos los pines A/D que se estén trabajando en el MCU, también puede conmutar los diferentes voltajes de referencia.

20.1 DIAGRAMA CIRCUITAL

En la Figura 20.1 apreciamos el diagrama en bloques del conversor A/D.

libros.ep-electropc.com

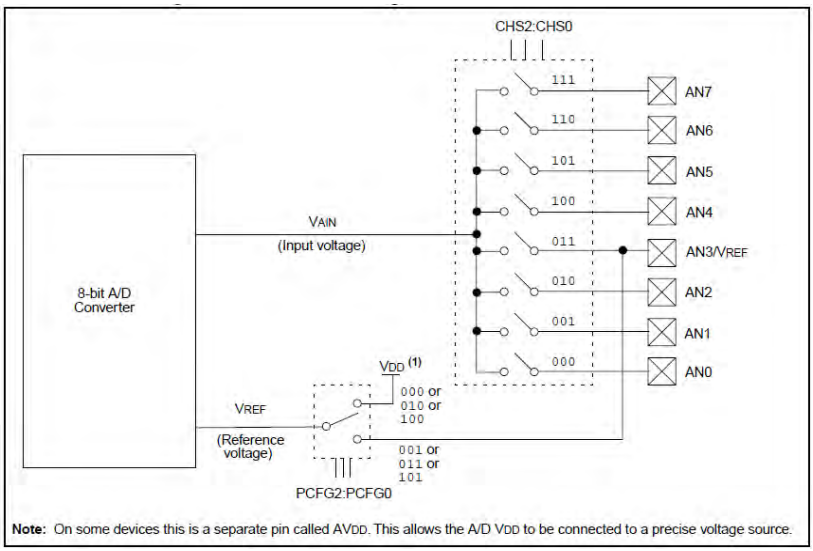


Figura 20.1 Diagrama de bloques del conversor A/D.

20.2 FUNCIONAMIENTO

En la conversión de señales análogas a digitales se deben tener en cuenta varios conceptos que trataremos a continuación.

20.2.1 RESOLUCIÓN

En este MCU el **ADC** (Conversor Análogo a Digital) es de 10 bits de resolución, y esto significa que una señal de entrada que varía entre 0-5V si se divide en 2^{10} = 1024 partes, cada parte equivale a una porción de 4.88mV (aprox. 5mV). Esta resolución es 4 veces mejor que una resolución de 8 bits, como vemos en la Tabla 20.1.

```
NOTA: Aunque las siglas de la traducción al español serían CAD, es
más común llamarlo por sus siglas ADC del inglés
```

También vemos que si bien con 10 bits tenemos 1 024 partes (Tabla 20.1), los valores están entre “0 a 1023d” o “0 a 3FFh”.

libros.ep-electropc.com

Tabla 20.1 Comparación de resoluciones.

RESOLUCIÓN		VALORES		EXACTO	APROX.
BIT	PARTES	DEC	HEX	mV	mV
8	256	0 - 255	0 - FF	19,53125	20
10	1024	0- 1023	0 - 3FF	4,8828125	5

20.2.2 JUSTIFICACIÓN

En este módulo ADC, como es de 10 bits, el resultado se debe almacenar en 2 registros **alto** (*high*) ADRESH y **bajo** (*low*) ADRESL, cada uno de 8 bits; de esta manera tenemos la posibilidad de presentar el valor de la conversión en dos tipos de formato, justificado a la derecha (acostumbrado) y justificado a la izquierda; como vemos en la Figura 20.2, con el bit ADFM en el registro ADCON1<7> se escoge el formato.

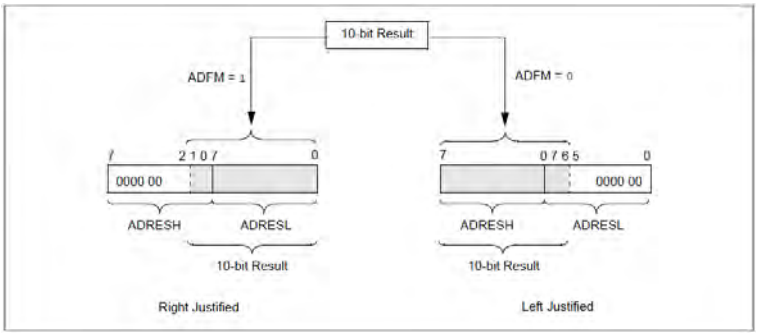
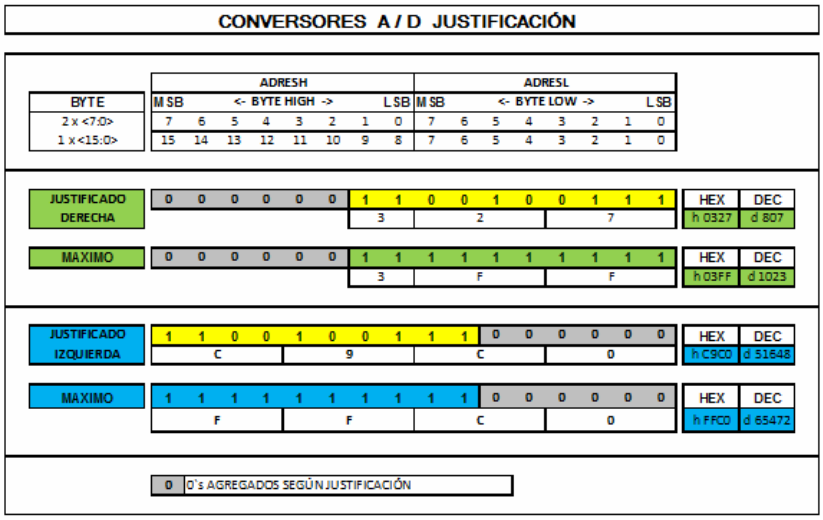


Figura 20.2 Resultado de justificación A/D.

Como apreciamos en la Tabla 20.2, el mismo valor de conversión “11 00 10 01 11” puede presentarse como formato justificado a la derecha que genera el valor “h 0327”, y con formato justificado a la izquierda que genera el valor “h C9C0”, lo cual nos explica la diferencia de justificación. También se observan los valores máximos en ambos formatos, justificado a la derecha “d 1 023” y justificado a la izquierda “d 65 472”. El usuario, según su aplicación, determina cuál de las dos justificaciones usa.

libros.ep-electropc.com

Tabla 20.2 Justificación de conversores A/D.



20.2.3 VOLTAJE DE REFERENCIA

El módulo ADC tiene 2 voltajes de referencia **alto** (*high*) y **bajo** (*low*), que pueden ser seleccionados en una combinación de V_{DD}, V_{SS}, RA3/VREF+, RA2/VREF-, con los bits **PCFG3:PCGF0** en el registro ADCON1<3:0>. Como voltajes de referencia la costumbre es usar Alto = V_{DD} = 5V y Bajo = V_{SS} = 0V, pero el usuario experimentado podrá seleccionar otros valores, teniendo en cuenta los rangos que usa.

20.2.4 SELECCIONANDO CLOCK

El tiempo por bit de conversión es definido como TAD. La conversión necesita un mínimo de 12 TAD para 10 bits. La fuente de reloj es seleccionada por software; las 7 posibles opciones para TAD son:

1. 2 TOSC
2. 4 TOSC
3. 8 TOSC
4. 16 TOSC
5. 32 TOSC
6. 64 TOSC
7. Módulo interno del A/D, oscilador RC (2-6 µs)

Para la conversión correcta del módulo A/D, el reloj (TAD) debe ser seleccionado para asegurar un mínimo tiempo de TAD de 1.6 microsegundos (µs).

libros.ep-electropc.com

En la Tabla 20.3 se muestran los tiempos TAD resultantes de las frecuencias de operación del MCU, y la fuente de reloj seleccionada.

Tabla 20.3 TAD vs Máxima operación de Frecuencia.

AD Clock Source (TAD)		Maximum Device Frequency
Operation	ADCS2:ADCS1:ADCS0	
2 Tosc	000	1.25 MHz
4 Tosc	100	2.5 MHz
8 Tosc	001	5 MHz
16 Tosc	101	10 MHz
32 Tosc	010	20 MHz
64 Tosc	110	20 MHz
RC(1, 2, 3)	x11	(Note 1)

Note 1: The RC source has a typical TAD time of 4 µs but can vary between 2-6 µs.
2: When the device frequencies are greater than 1 MHz, the RC A/D conversion clock source is only recommended for Sleep operation.
3: For extended voltage devices (LF), please refer to **Section 17.0 "Electrical Characteristics"**.

NOTA:
1. La fuente RC tiene un valor típico de tiempo de TAD de 4 µs, pero puede variar entre 2-6 µs.

2. Cuando las frecuencias del MCU son mayores que 1 MHz, la fuente de *clock* de conversión RC A/D es solamente recomendada para operación de *Sleep.*

3. Para la definición extendida (LF), por favor, remítase a la 17.0: Electrical Characteristics.

20.2.5 TIEMPO DE ADQUISICIÓN

Este retardo de tiempo es el requerido antes que una conversión pueda comenzar. Para que el conversor A/D encuentre la exactitud especificada, se debe permitir al condensador de retención de carga (CHOLD) que llegue a su carga total o llegue al nivel de voltaje del canal de entrada. En la Figura 20.3 se muestra el modelo de una entrada análoga. La impedancia de la fuente de señal (RS) y la impedancia del switch de muestreo (RSS) afectan directamente el tiempo requerido para la carga del condensador (CHOLD).

La impedancia del switch de muestreo varía según la alimentación (V_{DD}), como se observa en la Figura 20.3; la impedancia máxima de la señal de fuente análoga es de 2.5 kΩ; por esto, el decremento de esta impedancia disminuye el **tiempo de**

libros.ep-electropc.com

adquisición. Después que el canal es seleccionado (cambiado) se debe hacer este tiempo de adquisición antes que la conversión comience.

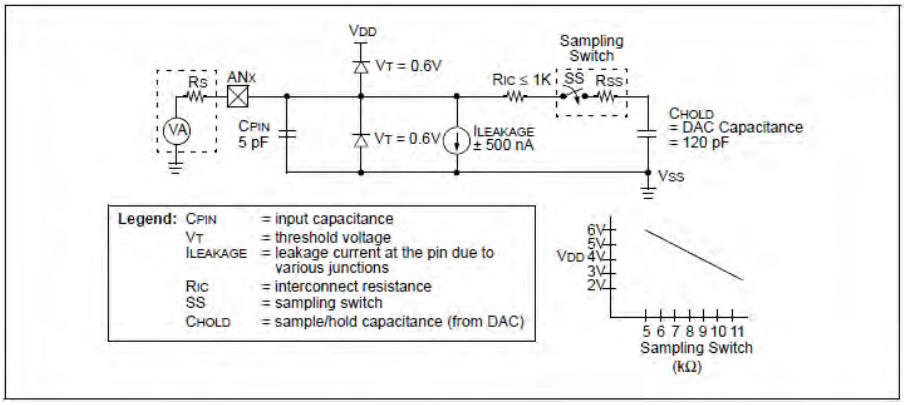


Figura 20.3 Modelo de una entrada análoga.

Para calcular el mínimo **tiempo de adquisición** se puede usar la fórmula de la **Ec. 20.1**; en esta ecuación se asume que se usa un error de ½ LSB (1 024 pasos para el A/D). El error de ½ LSB es el máximo error permitido para la resolución específica. Para un cálculo práctico, si trabajamos con un cristal de 20 MHz, tenemos:

Tiempo de adquisición	= 20 µs
Tiempo total de conversión + adquisición	= 14 ciclos de máquina + 20 µs = 22.8 µs

$$
\begin{aligned}
T_{ACQ} &= \text{Amplifier Settling Time} + \text{Hold Capacitor Charging Time} + \text{Temperature Coefficient} \\
&= T_{AMP} + T_C + T_{COFF} \\
&= 2\,\mu s + T_C + [(\text{Temperature} - 25^\circ C)(0.05\,\mu s/^\circ C)] \\
T_C &= C_{HOLD}(R_{IC} + R_{SS} + R_S)\ln(1/2047) \\
&= -120\,pF\,(1\,k\Omega + 7\,k\Omega + 10\,k\Omega)\ln(0.0004885) \\
&= 16.47\,\mu s \\
T_{ACQ} &= 2\,\mu s + 16.47\,\mu s + [(50^\circ C - 25^\circ C)(0.05\,\mu s/^\circ C) \\
&= 19.72\,\mu s
\end{aligned}
$$

Note 1: The reference voltage (V_{REF}) has no effect on the equation since it cancels itself out.
2: The charge holding capacitor (C_{HOLD}) is not discharged after each conversion.
3: The maximum recommended impedance for analog sources is 2.5 kΩ. This is required to meet the pin leakage specification.

Ec. 201.1 Cálculo de tiempo de adquisición

libros.ep-electropc.com

20.2.6 CONFIGURACIÓN DE PINES DEL PUERTO ANÁLOGO

Los registros **ADCON** y **TRIS** controlan la operación de los pines del puerto A/D. Los pines de puerto que se desean como entradas análogas deben tener su bit correspondiente en el registro **TRIS** configurado como entrada (puertos = 1 Entrada). Si se borra el bit del TRIS quedará como salida y el valor digital (VOH o VOL) será convertido. La operación del módulo A/D es independiente del estado de los bits CHS2:CHS0 y los bits del TRIS. Esto indica que el módulo A/D realiza una operación de conversión sin importar la conexión seleccionada.

NOTA:
1. Cuando leemos el registro del puerto (PORTx), cualquier pin configurado como canal de entrada análoga será leído como borrado = "0" o nivel bajo (*low*).

2. Los pines configurados como entradas digitales serán convertidos como entradas análogas.

3. Los niveles análogos sobre un pin configurado como entrada digital no afectan la exactitud de la conversión.

4. Los niveles análogos sobre un pin que es definido como una **entrada digital** (incluyendo los pines AN7:AN0) pueden causar que el consumo de corrientes, en el buffer de entrada, esté fuera de las especificaciones del dispositivo.

20.2.7 OPERACIÓN

El módulo ADC tiene 4 registros básicos de operación que son:

1. A/D Resultado Registro Alto (ADRESH) (1Eh)
2. A/D Resultado Registro Bajo (ADRESL) (9Eh)
3. A/D Control Registro 0 (ADCON0) (1Fh)
4. A/D Control Registro 1 (ADCON1) (9Fh)

En la Figura 20.4 a, b, apreciamos cómo el registro ADCON0 controla la operación del módulo ADC, y el registro ADCON1 configura las funciones de los pines del puerto; los pines del puerto pueden ser configurados como entradas análogas (RA3 puede ser el voltaje de referencia), o como I/O digitales. Además, observamos ambos registros y sus bits.

libros.ep-electropc.com

a)

ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

b)

ADCON1 REGISTER (ADDRESS 9Fh)

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Figura 20.4 Registros ADCON0 y ADCON1.

Los registros **ADRESH** y **ADRESL** contienen el resultado de la conversión de 10 bits. Ambos registros y sus bits se observan en la Figura 20.5.

ADRESH (1Eh)								ADRESL (9Eh)							
MSB <- BYTE HIGH ->							LSB	MSB <- BYTE LOW ->							LSB
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Figura 20.5 Registros ADRESH y ADRESL.

Cuando la conversión se completa el resultado es guardado en el par de registros **ADRESH:ADRESL**, el bit de bandera **GO/-DONE** de registro **ADCON0<2>** es borrado y el bit de bandera ADIF del registro PIR1 es puesto a uno. Después que el A/D es configurado como se desea, se debe seleccionar el canal deseado, y esperar el retardo de **tiempo de adquisición** antes que la conversión comience; la entrada análoga debe ser programada como entrada en su pin correspondiente en el registro TRIS. (Para determinar el tiempo de muestreo véase la **Sección 20.2.5: Tiempo de adquisición**.) Tan pronto como el tiempo de retardo ha terminado, una conversión A/D puede comenzar.

20.2.8 CONVERSIÓN A/D

Si durante una conversión se borra el bit GO/-DONE, la conversión se aborta; el par de registros no son actualizados porque hay una conversión parcialmente completada. O sea que los registros **ADRESH:ADRESL** siguen conteniendo los valores de la última conversión completa. Después de que una conversión es abortada, la adquisición siguiente comienza automáticamente. El bit GO/-DONE puede ser puesto a uno en cualquier momento para comenzar la conversión siguiente (Figura 20.6).

libros.ep-electropc.com

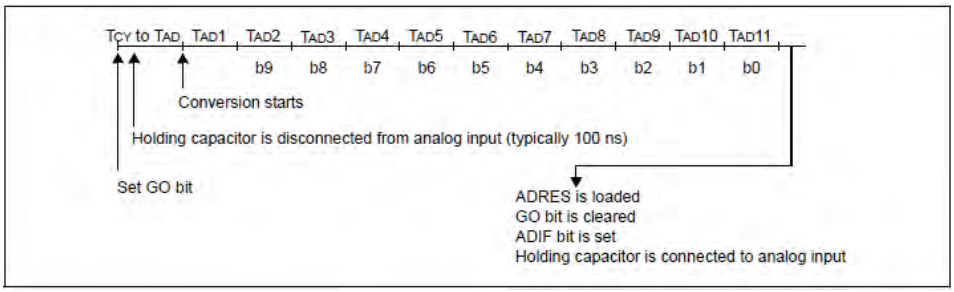


Figura 20.6 Ciclos de conversión A/D.

En la Figura 20.6 apreciamos que después de que el bit GO/-DONE es puesto a uno, el primer segmento tiene un tiempo mínimo de TCY y un tiempo máximo de TAD.

```
NOTA: El bit GO/-DONE no puede ser puesto a uno en la misma
instrucción de encendido del módulo A/D
```

En la Figura 20.7 vemos un diagrama de tiempos de todas las señales que intervienen en una conversión A/D. La especificación de los tiempos (números como 130, 131 en la imagen) se puede encontrar en la hoja de datos o *data sheet*, DS39582B.pdf, p. 195, o en el **Apéndice A: Características técnicas**.

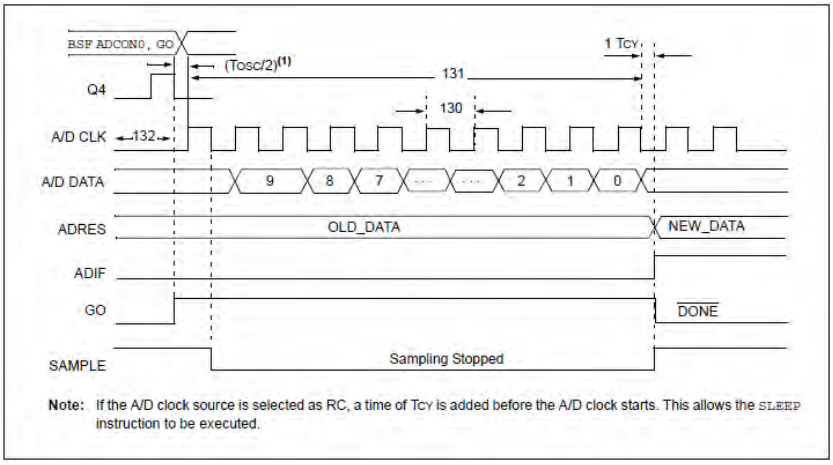


Figura 20.7 Diagrama de tiempos de conversión A/D.

libros.ep-electropc.com

Operación en el modo sleep

El módulo ADC puede operar en modo de *sleep*; para esto se requiere que la fuente de reloj (*clock*) sea el oscilador RC interno que se selecciona con los bits ADSC1:ADSC0 = 11 en el registro ADCON0. Cuando seleccionamos el reloj del RC interno, el módulo A/D espera una instrucción antes de comenzar una conversión. Esto permite que la instrucción *sleep* sea ejecutada, lo cual elimina todos los ruidos de conmutación digital para la conversión. Cuando la conversión es completada, el bit GO/-DONE será borrado y el resultado guardado en los registros ADRES. Si la interrupción del A/D es habilitada, el MCU despertará del modo de *sleep*. Si la interrupción del A/D es deshabilitada, el módulo A/D será apagado; a través del bit ADON en el registro ADCON0<0> puede mantenerse puesto a uno.

Cuando la fuente de reloj es otra (no RC), una instrucción de *sleep* causará que la conversión actual sea abortada y el módulo A/D sea apagado; a través del bit ADON en el registro ADCON0<0> puede mantenerse puesto a uno. Apagar el módulo A/D lo pone en estado de bajo consumo.

```
NOTA: Para que el módulo A/D opere en sleep debe elegirse el
oscilador interno RC (ADCS1:ADCS0 = 11). Para permitir que la
conversión ocurra durante sleep, debe asegurarse que la instrucción
sleep esté inmediatamente a continuación de la instrucción que pone
a uno el bit GO/-DONE.
```

Efectos de un reset

Un *reset* fuerza a todos los registros al estado de *reset*. A la vez, esto fuerza a que se apague el módulo A/D y cualquier conversión sea abortada. Todos los pines de entrada A/D son configurados como entradas análogas. El valor de los registros **ADRESH:ADRESL** no es modificado por una condición de *power-on reset*. Los registros ADRESH:ADRESL contienen un valor desconocido después de una condición *power-on reset*.

20.3 REGISTROS ASOCIADOS

En las Tablas 20.4, 20.5 y 20.6 vemos con detalle los registros ADCON0 y ADCON1 asociados con el conversor A/D.

libros.ep-electropc.com

Tabla 20.4 Registros ADCON0.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON
bit 7							bit 0

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	**00**	Fosc/2
0	**01**	Fosc/8
0	**10**	Fosc/32
0	**11**	FRC (clock derived from the internal A/D RC oscillator)
1	**00**	Fosc/4
1	**01**	Fosc/16
1	**10**	Fosc/64
1	**11**	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Channel 7 (AN7)

Note: The PIC16F873A/876A devices only implement A/D channels 0 through 4; the unimplemented selections are reserved. Do not select any unimplemented channels with these devices.

bit 2 **GO/DONE:** A/D Conversion Status bit

When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit

1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 20.5 Registro ADCON1 (ADDRESS 9Fh).

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: On any device Reset, the port pins that are multiplexed with analog functions (ANx) are forced to be an analog input.

libros.ep-electropc.com

Tabla 20.6 Registros asociados con el conversor A/D.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on MCLR, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h(1)	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h(1)	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used for A/D conversion.
Note 1: These registers are not available on 28-pin devices.

20.4 PROCEDIMIENTO DE USO

Para trabajar con el módulo A/D siga los pasos siguientes:

1. Configure el módulo A/D:

- Configure los pines de entradas análogas, pines digitales, voltaje de referencia en los bits PCFG3:PCFG0 del registro ADCON1<3:0>.
- Seleccione el canal A/D que va a convertir en los bits CHS2:CHS0 del registro ADCON0<5:3>
- Seleccione el *clock* de conversión en el bit ADCS2 del registro ADCON1<6> y los bits ADCS1:ADCS0 en el registro ADCON0<7:6>.
- Prenda el módulo A/D con el bit ADON del registro ADCON0.
- Configure el pin A/D como entrada en el registro TRIS correspondiente (luego de Reset por default en Entradas = '1').

2. Configure la interrupción A/D (si se desea):

- Aclare el bit de la bandera ADIF en el registro PIR1<6>.
- Ponga el bit de la bandera ADIE en el registro PIE1<6>.
- Ponga el bit de la bandera PEIE en el registro INTCON<6>.
- Ponga el bit de la bandera GIE en el registro INTCON<7>.

libros.ep-electropc.com

3. Espere el tiempo requerido de adquisición.

4. Comience la conversión:

- Ponga a uno el bit de la bandera GO/-DONE en el registro ADCON0<2>.

5. Espere que se complete la conversión, ya sea que:

- Pregunte por el bit GO/-DONE en el registro ADCON0<2>, hasta cuando éste sea borrado (interrupción deshabilitada); o
- Espere la interrupción del A/D.

6. Lea el registro del resultado de conversión en el par de registros ADRESH:ADRESL:

Borre el bit de la bandera ADIF en el registro PIR1<6> (si se requiere).

7. Para la próxima conversión vaya a los pasos 1 o 2 (si se requiere):

El tiempo de conversión por bit es definido como TAD.

A continuación repita los pasos con código:

1. Configure el módulo A/D:

- Configure los pines de entradas análogas, pines digitales, voltaje de referencia en los bits PCFG3:PCFG0 del registro ADCON1<3:0>.
- Seleccione el canal A/D que va a convertir en los bits CHS2:CHS0 del registro ADCON0<5:3>.
- Seleccione el reloj de conversión en el bit ADCS2 del registro ADCON1<6> y los bits ADCS1:ADCS0 en el registro ADCON0<7:6>.

```
; configura AD

        bsf     STATUS,RP0      ; seleccione Banco 1
        movlw   b'11000010'     ; Just.Der, Fosc/64, D = AN7,6,5 A/D = AN4,3,2,1,0
                                ; ADFM/ADCS2/0/0/PCFG3/PCFG2/PCFG1/PCFG0
        movwf   ADCON1          ; registro 1 de configuración A/D

        bcf     STATUS,RP0      ; seleccione Banco 0

        movlw   b'10000000'     ; Fosc/64, Canal AN0, GO_Done, 0, AD_ON
                                ; ADCS1/ADCS0/CHS2/CHS1/CHS0/GO_DONE/0/ADON
```


libros.ep-electropc.com

```
        movwf   ADCON0          ; registro 0 de configuración A/D

; hace retardo inicial de Tacq
```

- Prenda el módulo A/D con el bit ADON del registro ADCON0:

```
        bsf     ADCON0,ADON     ; prende AD, ON
```

- Configure el pin A/D como entrada en el registro TRIS correspondiente.
  (No necesario cuando se inicia luego de *reset*, por default en Entradas = '1'.)

  Si es necesario, el código aquí es:

```
        bsf     STATUS,RP0      ; seleccione Banco 1
        clrf    TRISA           ; todos los bits en 0, salidas en puerto A
        comf    TRISA           ; todos los bits en 1, entradas en puerto A
        bcf     STATUS,RP0      ; seleccione Banco 0
```

2. Configure la interrupción A/D (si lo desea):

- Borre el bit de la bandera ADIF en el registro PIR1<6>:

```
        bsf     PIR1,ADIF       ; apaga bandera de AD
```

- Ponga el bit de la bandera ADIE en el registro PIE1<6>:

```
        bsf     STATUS,RP0      ; seleccione Banco 1
        bsf     PIE1,ADIE       ; habilita interrupción de AD
        bcf     STATUS,RP0      ; seleccione Banco 0
```

- Ponga el bit de la bandera PEIE en el registro INTCON<6>:

```
        bsf     INTCON,PEIE     ; habilita interrupciones periféricos
```

- Ponga el bit de la bandera GIE en el registro INTCON<7>:

```
        bsf     INTCON,GIE      ; habilita interrupciones globales
```

3. Espere el tiempo requerido de adquisición:

```
; retardo de tiempo de adquisición
; 20 µs (Tcy = 0.5us, a 20 MHz) son 40 * Tcy de espera

        movlw   0x06            ; cargue valor 6
        movwf   Delay3          ; en retardo 3
DelayLoop3:
        Decfsz  Delay3,f        ; decremente retardo 3
        goto    DelayLoop3      ; si no termina ciclo, vaya a DelayLoop
```

4. Comience la conversion:

libros.ep-electropc.com

- Ponga a uno el bit de la bandera GO/-DONE en el registro ADCON0<2>:

```
bsf     ADCON0,GO_DONE  ; empiece conversión
```

5. Espere que se complete la conversión, ya sea que:

- Pregunte por el bit GO/-DONE en el registro ADCON0<2>, hasta cuando éste sea borrado (interrupción deshabilitada método *polling*); o

```
Main_conversión
        btfsc   ADCON0,GO_DONE  ; terminó conversión = 0? sí, salte
        goto    $-1             ; no, haga ciclo Main_conversión
```

- Espere la interrupción del A/D.

6. Lea el registro del resultado de conversión en el par de registros ADRESH:ADRESL:

```
; lea resultado de conversión ADRESH:L (1Eh:9Eh) y póngalo en reg_ADRESH:L (0X20:0X21)
        movlw   ADRESH          ; cargue dirección a W
        movwf   FSR             ; cargue el apuntador

        movf    INDF,W          ; lea ADRESH
        movwf   reg_ADRESH      ; guarde en registro de memoria ADRESH

        bsf     FSR,bit_7       ; cambie a banco 1
        movf    INDF,W          ; lea ADRESL
        bcf     FSR,bit_7       ; cambie a banco 0
        movwf   reg_ADRESL      ; guarde en registro de memoria ADRESL
```

Aclare el bit de la bandera ADIF en el registro PIR1<6> (si se requiere, sólo para interrupción).

7. Para la próxima conversión vaya a los pasos 1 o 2 (si se requiere):
   El tiempo de conversión por bit es definido como TAD.

20.5 PRÁCTICA

En el ejemplo siguiente se lee el valor de un potenciómetro conectado al conversor **A/D** pin **RA0/AN0** con el módulo **MOD_COMP (2009-05-07)** y la tarjeta **PIC16F8XX**; el montaje para esta práctica se ve en la Figura 20.8.

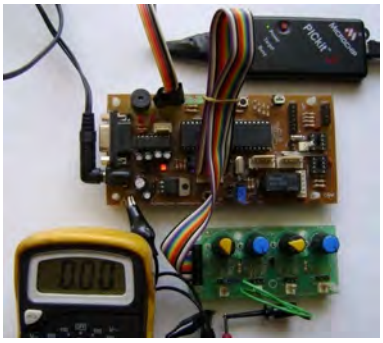


Figura 20.8
Práctica de
montaje A/D

En la Figura 20.9 observamos con detalle el conector **P9** en la tarjeta **PIC16F8XX**.

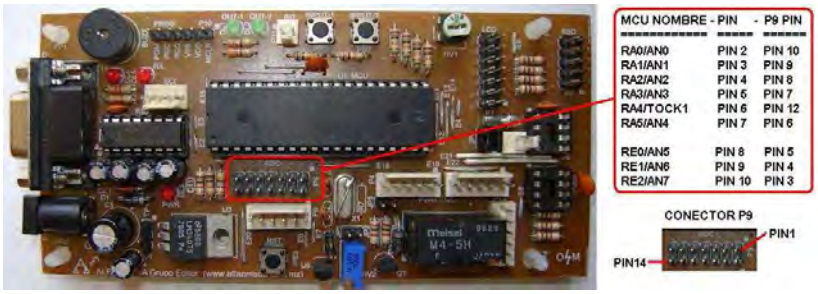


Figura 20.9 Conector A/D, P9.

Los pines que usaremos en esta práctica están definidos en la Tabla 20.7, lo mismo que la función del conector P9 conectado al **MOD_COMP** (2009-05-12) con el que vamos a trabajar. En la tabla los resaltados en gris son los pines para esta práctica.

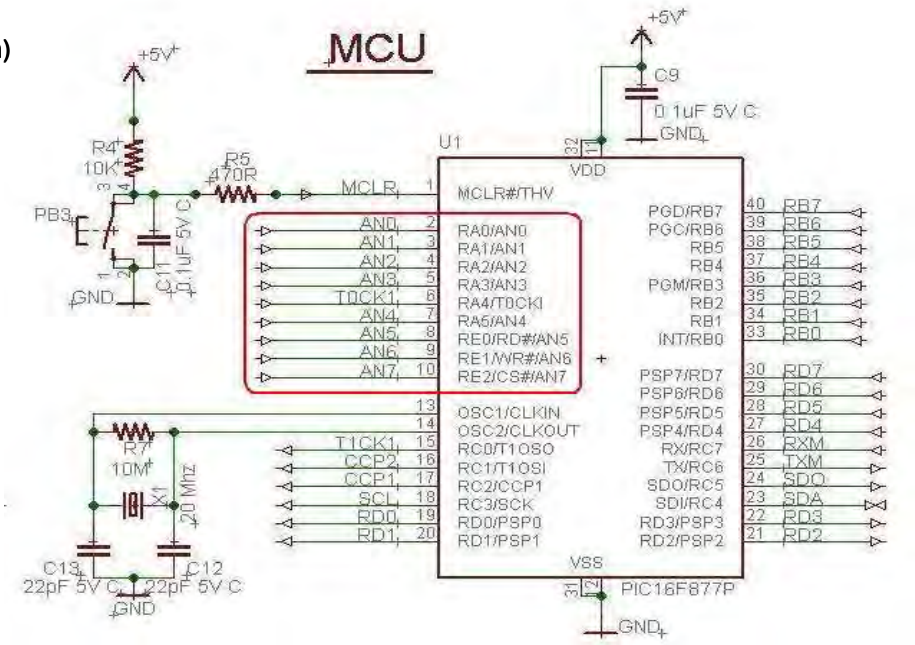
Tabla 20.7
Pines y
función de AD.

CONEXIÓN AD
P9 A MOD_COMP (2009-05-12)

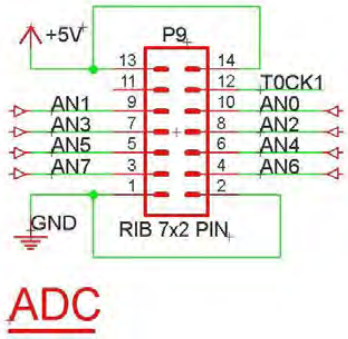
MCU		P9		
PIN	NOMBRE	PIN	ADC	FUNCIÓN
		14	+5V	+5V
		13	+5V	+5V
6	RA4/T0CKI/C1OUT	12	TOCK1	
		11		
2	RA0/AN0	10	AN0	POT. RV0
3	RA1/AN1	9	AN1	POT. RV1
4	RA2/AN2/VREF-/CVREF	8	AN2	POT. RV2
5	RA3/AN3/VREF+	7	AN3	POT. RV3
7	RA5/AN4/SS/C2OUT	6	AN4	
8	RE0/RD/AN5	5	AN5	
9	RE1/WR/AN6	4	AN6	
10	RE2/CS/AN7	3	AN7	
		2	GND	GND
		1	GND	GND

En la Figura 20.10 a, b, c observamos el plano esquemático de la práctica AD.

a)



b)



c)

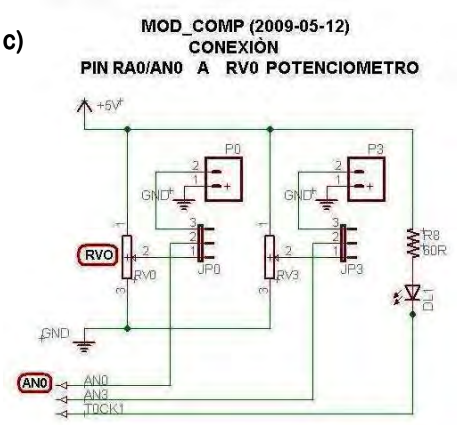


Figura 20.10 Plano esquemático, práctica AD.

libros.ep-electropc.com

En la Figura 20.11 a, b vemos una imagen del módulo MOD_COMP (2009-05-12) con componentes y un *screen* de la tarjeta.

a)



b)

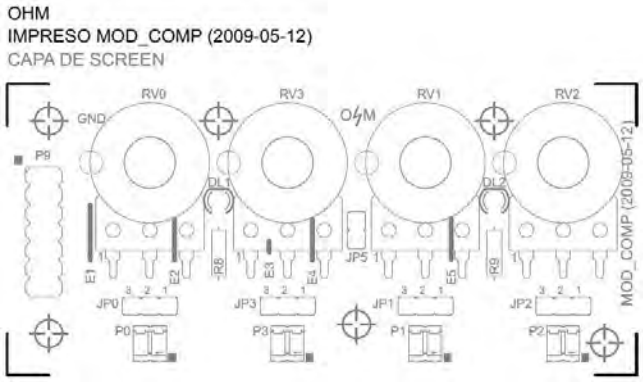


Figura 20.11 Módulo MOD_COMP (2009-05-12).

En esta tarjeta se debe conectar el conector P9 al conector P9 de la tarjeta **PIC16F8X**. También se requiere poner el jumper JP0 entre 1 y 2, para habilitar el potenciómetro RV0 con el pin RA0/AN0. El valor de voltaje leído en un potenciómetro RV0 del módulo conectado al pin RA0/AN0 es convertido, y los valores de la conversión ADRESH y ADRESL se guardan en los registros **reg_ADRESH** y **reg_ADRESL**. La configuración de los pines A/D utilizada es la que se muestra en la Figura 20.12.

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0

Figura 20.12 Configuración AD.

libros.ep-electropc.com

En nuestra página Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\AD_1\AD_1.asm.

```
        List p = 16f877a          ; directiva para definir procesador
#include<p16f877a.inc>            ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

            cblock 0x20           ; desde la posición 20 en RAM, separa espacio para estas variables
                                  ; define 3 registros para retardo
reg_ADRESH                        ; memoria de conversión ADRESH
reg_ADRESL                        ; memoria de conversión ADRESL
Delay1                            ; retardo 1
            endc

#define     bit_7       7         ; define bit 7

            org 0

Start:
; configura AD
            clrf        PORTA               ; apague puerto A

            bsf         STATUS,RP0          ; seleccione Banco 1
            movlw       b'11000010'         ; Just.Der, Fosc/64, D = AN7,6,5 A/D = AN4,3,2,1,0
                                            ; ADFM/ADCS2/0/0/PCFG3/PCFG2/PCFG1/PCFG0
            movwf       ADCON1              ; registro 1 de configuración A/D

            bcf         STATUS,RP0          ; seleccione Banco 0
            movlw       b'10000000'         ; Fosc/64, canal AN0, GO_Done, 0, AD_ON
                                            ; ADCS1/ADCS0/CHS2/CHS1/CHS0/GO_DONE/0/ADON
            movwf       ADCON0              ; registro 0 de configuración A/D

            bsf         ADCON0,ADON         ; prende AD, ON

; hace retardo inicial de Tacq
; retardo de tiempo de adquisición
; 20 µs(Tcy = 0.5 µs , a 20 MHz) son 40 * Tcy de espera
            movlw       0x06                ; cargue valor 6
            movwf       Delay1              ; en retardo 1
DelayLoop1:
            decfsz      Delay1,f            ; decremente retardo 1
            goto        DelayLoop1          ; si no termina ciclo, vaya a DelayLoop

Main_Loop:
; empiece conversión
            bsf         ADCON0,GO_DONE      ; empiece conversión

Main_conversión
; ciclo de conversión
            btfsc       ADCON0,GO_DONE      ; terminó conversión = 0? sí, salte
            goto        $-1                 ; no, haga ciclo Main_conversión
```


libros.ep-electropc.com

```
; lea resultado de conversión ADRESH:L y póngalo en reg_ADRESH:L
        movlw   ADRESH          ; cargue dirección a W
        movwf   FSR             ; cargue el apuntador

        movf    INDF,W          ; lea ADRESH
        movwf   reg_ADRESH      ; guarde en registro de memoria ADRESH

        bsf     FSR,bit_7       ; cambie a banco 1
        movf    INDF,W          ; lea ADRESL
        bcf     FSR,bit_7       ; cambie a banco 0
        movwf   reg_ADRESL      ; guarde en registro de memoria ADRESL

        goto    Main_Loop       ; haga ciclo

        end
```

Siguiendo paso a paso el programa anterior podemos generar los valores de lecturas sobre el potenciómetro RV0, tal como se indica en la Figura 20.12. Cuando se convierte un valor de voltaje de entrada en código hexadecimal, es necesario (y recomendable) tomar varias lecturas sobre el mismo voltaje que se vaya a convertir, y luego hacer un promedio de estos voltajes para tener una mayor aproximación al valor convertido. A continuación (Tabla 20.8) presentamos los datos muestreados para 7 valores de voltaje, obtenidos de 7 posiciones del potenciómetro RV0; para cada voltaje se toman 6 lecturas y luego se hace el promedio de ellas.

libros.ep-electropc.com

V_MAX TEORICO)	5
V_MAX (LEIDO)	4,86
RESOLICIÒN	1024

V calculado = V max * (AD/1024)

LECTURA
PROMEDIO

TESTER	AD	
VOLTIOS	HEX	DEC
0	0	0
0,00	0	0
1,00	00CE	206
	00D6	214
	00CF	207
	00CE	206
	00C4	196
	00D1	209
0,98	CE	206
2,00	01A1	417
	01A8	424
	01AA	426
	01AD	429
	01A0	416
	01B1	433
2,02	1A9	426
2,50	0218	536
	020E	526
	020C	524
	020E	526
	0209	521
	0208	520
2,49	20D	526
3,00	0277	631
	0276	630
	0293	659
	0273	627
	0278	632
	025D	605
2,99	276	631
4,00	034E	846
	034E	846
	0349	841
	0348	840
	034C	844
	0359	857
4,01	34D	846
4,86	03FF	1023
	03FF	1023
	03FF	1023
	03FF	1023
	03FF	1023
	03FF	1023
4,86	3FF	1023

Tabla 20.8 Muestras AD.

libros.ep-electropc.com

En la Tabla 20.8 el valor en **voltios** es medido con el **tester**, y el valor A/D que está en código **HEX** (hexadecimal) son los valores de muestreo. El valor promedio calculado es bastante aproximado a la lectura real, luego de realizar el voltaje (V) calculado. Podemos observar los valores para el cuadro de 1.00 voltio, el valor promedio de las 6 conversiones es 00CEh (206d) con el voltaje máximo real V_{DD} = 4.86. Calculando el valor promedio, tenemos:
V calculado = V máx * (AD/1024) = 4.86 voltios * (276/1 024) = 0.98 voltios
Muy similar a 1.00 voltio.

Lo mismo para el valor leído 2.00 cuyo valor promedio de conversión es 01A9h (426). Calculando el valor promedio, tenemos:

V calculado = V máx * (AD/1 024) = 4.86 voltios * (426/1 024) = 2.02 voltios
Muy similar a 2.00 voltios.

Observe cómo los valores promedio nos dan una lectura de mayor aproximación, y cuya precisión está en 0.02 voltio al realizar el voltaje (V) calculado.

En la Figura 20.13 a, b, c, d, e, f se presentan las lecturas en fotos; observe cómo la posición del potenciómetro RV0 gira de posición, desde un mínimo en 0.00V hasta 4.86V.

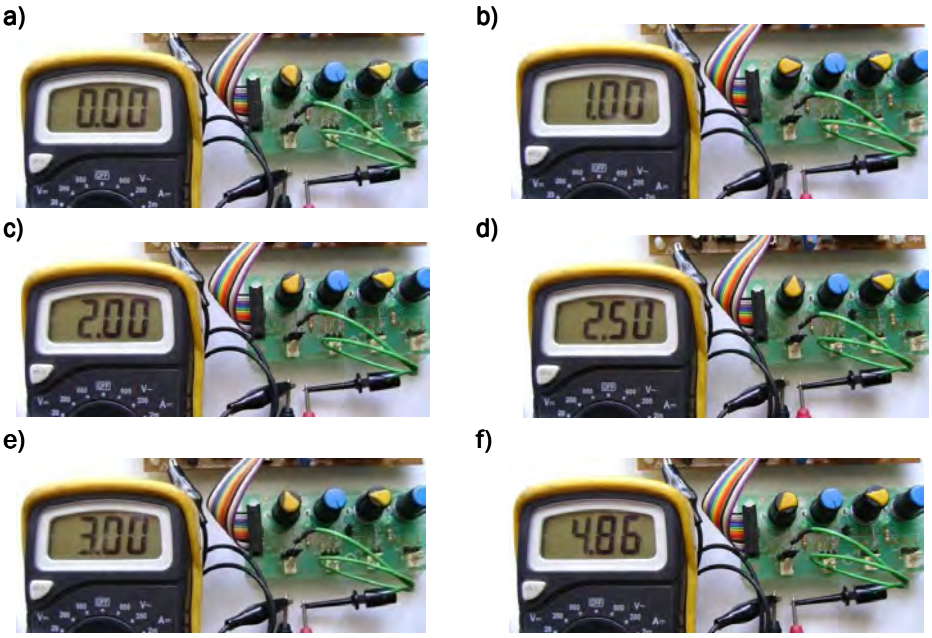


Figura 20.13 Muestras AD.

libros.ep-electropc.com

En nuestra página Web puede encontrar este ejemplo con subrutinas en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco para encontrarla como C:\Ejemplos_ASM\AD_1\AD_1A.asm.

EVALUACIÓN

```
P.: 1. ¿Cuántos conversores A/D hay?
R.:
Hay solamente uno que se multiplexa con todos los pines A/D.

P.: 2. En un conversor A/D con 10 bits de resolución y un voltaje
de alimentación de VDD = 5V, VSS = 0V, ¿en cuántos pasos se divide
la señal de entrada (valores en decimal y hexadecimal)? ¿Y cuál es
el voltaje mínimo que permite medir?
R.:
Se divide en 1024 pasos (0-1 023) (0-3FFh), 4.88mv (o 5mv
aproximado).

P.: 3. ¿Cuáles son los bytes donde se almacena el resultado de la
conversión de un A/D?
R.:
ADRESH (high) Alto y ADRESL (low) bajo.

P.: 4. Si el resultado de una conversión A/D es 11 0010 0111:
¿Qué valor (en decimal y hexadecimal):
Se presenta justificado a la derecha?
Justificado a la izquierda?
R.:
Justificado a la derecha: 0327h = 807d.
Justificado a la izquierda: C9C0h = 51648d.

P.: 5. ¿Cada vez que comienza una conversión A/D sobre el mismo pin
hay que esperar el tiempo de adquisición?
R.:
No, solamente cuando se cambie de pin.
```

21.0 COMPARADORES

Los **comparadores**, como su nombre lo dice, comparan señales análogas y generan salidas digitales. Operan igual que los amplificadores operacionales en configuración de comparadores. En este MCU los comparadores tienen además la posibilidad de ser configurables, y los valores que se requiera comparar no pueden sobrepasar los voltajes de alimentación. También existe la opción de generar un voltaje de referencia interno. Como módulos independientes veremos: los **comparadores** y el **generador de voltaje de referencia**.

libros.ep-electropc.com

21.1 MÓDULO COMPARADOR (COMPARATOR)

El **comparador** se usa para interfazar dos circuitos análogos que, al compararlos, provee una salida digital que indica su magnitud relativa. Son muy útiles al construir bloques de señales mezcladas, lo que permite una funcionalidad independiente de la ejecución del programa. El **módulo comparador** contiene dos comparadores análogos. Las entradas al comparador son multiplexadas con los puertos RA0 a RA3, mientras que las salidas son multiplexadas con los pines RA4 y RA5. El **voltaje de referencia VREF** (*Voltage of Reference*) puede ser una entrada interna a los comparadores. El registro CMCON controla las entradas y salidas multiplexadas.

21.1.1 DIAGRAMA CIRCUITAL

Existen dos comparadores C1 y C2, cada uno con sus tres pines de **V+, V- y OUT**; estos pines se pueden configurar en 8 modos de operación que veremos más adelante; en la Figura 21.1, se observa cómo se conectan con los pines RA<5:0> en un diagrama simplificado de los 2 comparadores.

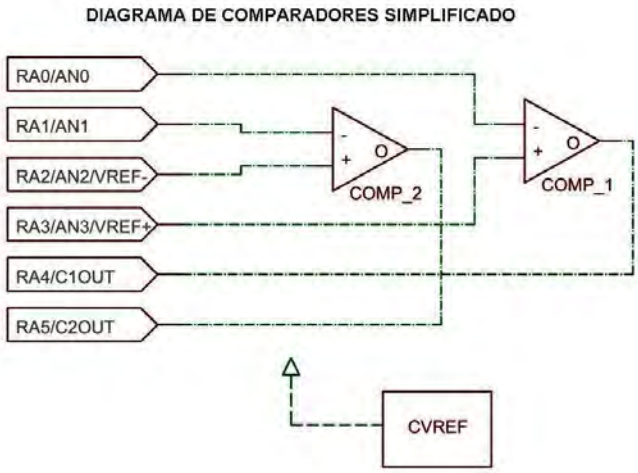
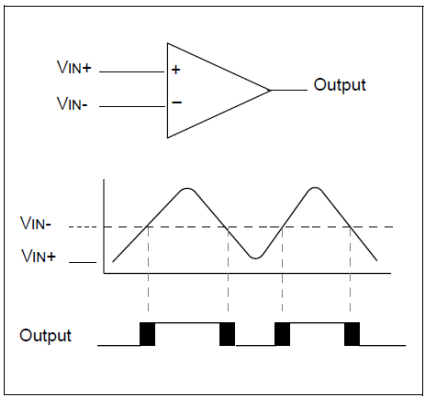


Figura 21.1 Diagrama de comparadores simplificado.

En la Figura 21.2 a, b observamos cómo se conectan y comportan las señales de referencia Vin+ Vin- y de salida (*output*).

libros.ep-electropc.com

a)



b)

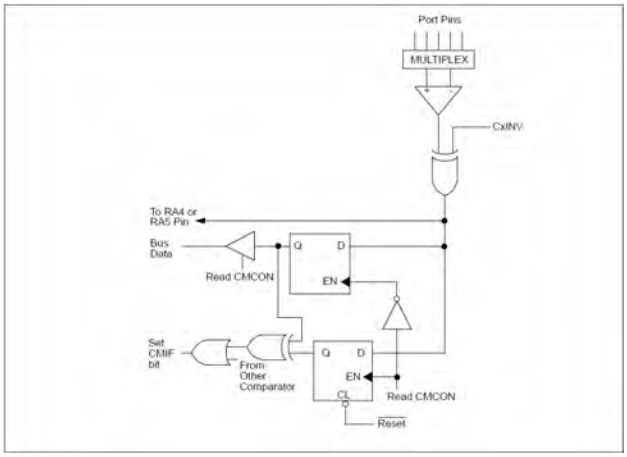


Figura 21.2 Circuito comparador sencillo.

Existen 8 modos de conexión para los comparadores; estos modos se configuran por los bits **CM2:CM0** del registro **CMCOM<2:0>**, como se aprecia en la Figura 21.3.

R-0	R-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
C2OUT	C1OUT	—	—	CIS	CM2	CM1	CM0
bit 7							bit 0

Figura 21.3 Registro CMCON y sus bits.

libros.ep-electropc.com

En la Figura 21.4 observamos los 8 modos de conexión del comparador.

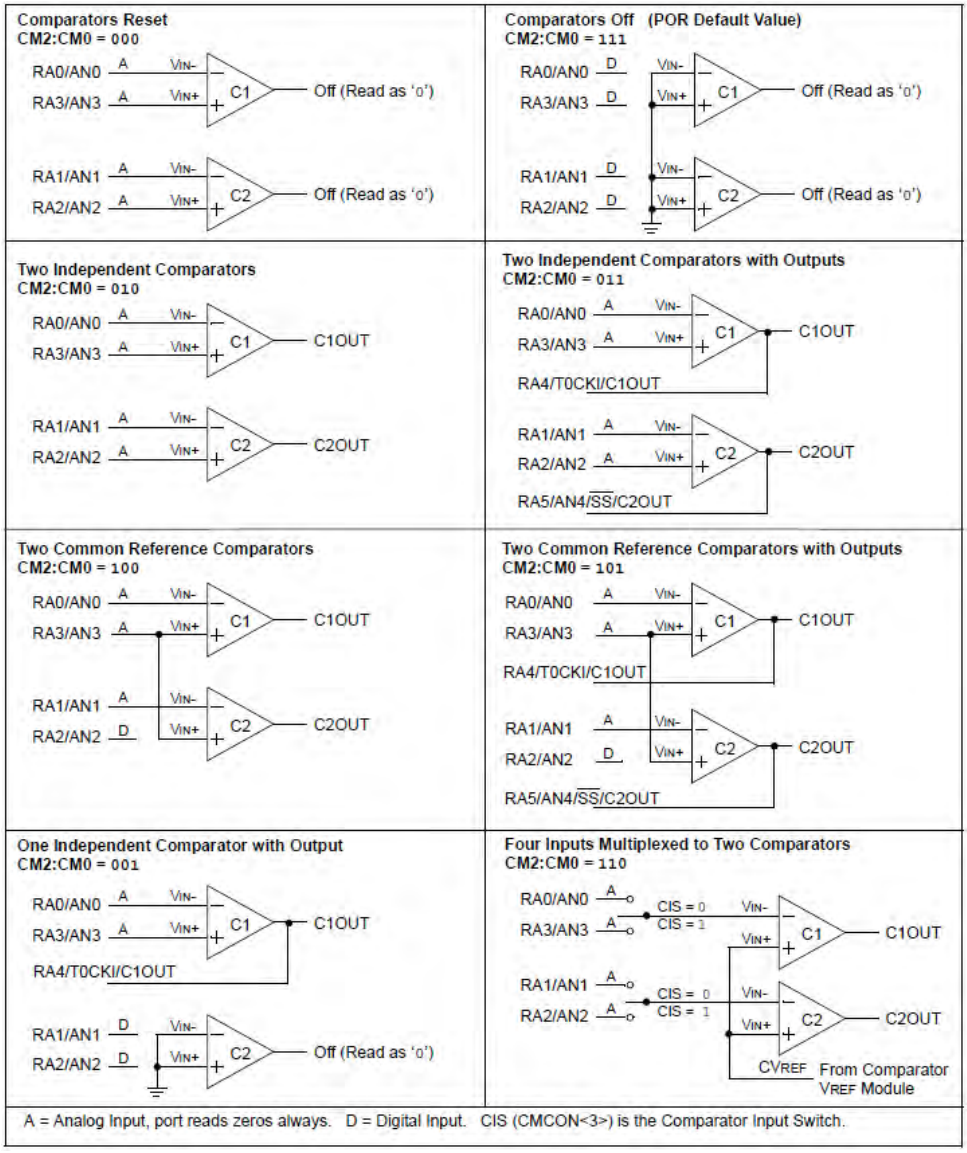


Figura 21.4 Modos de conexión del comparador.

libros.ep-electropc.com

21.1.2 FUNCIONAMIENTO

Modos de operación

Hay ocho modos de operación para los comparadores. El registro CMCOM se usa para seleccionar estos modos. En la Figura 21.4 se muestran los ocho posibles modos. El registro TRISA controla la dirección del pin del comparador para cada modo. Si el modo del comparador es cambiado, el nivel de salida puede no ser válido por el retardo del cambio de modo. (Véase **Apéndice A, Tabla A-4: Características eléctricas, retardo de cambio de modo.**)

```
NOTA: La interrupción del comparador debe ser deshabilitada durante
un cambio de modo. De otra manera puede ocurrir una interrupción
falsa.
```

Operación de un comparador

En la Figura 21.5 se muestra un solo comparador con las relaciones de entrada analógica y salida digital.

- Cuando Vin+ < Vin-, *output* (salida) bajo (*low* = "0")
- Cuando Vin+ > Vin-, *output* (salida) alto = 1 (*high* = "1")

En la Figura 21.5 las zonas oscuras son valores inciertos, debido a *offset* de entrada y al tiempo de respuesta del comparador.

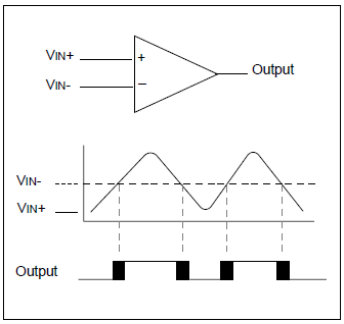


Figura 21.5 Comparador.

Voltaje de referencia

Puede usarse una señal externa o interna de referencia, según el modo de operación. Cuando se usa una referencia externa, los comparadores pueden ser

libros.ep-electropc.com

configurados para que tengan la misma o independiente referencia de voltaje. Sin embargo, el umbral de detección de voltaje de la aplicación debe ser el mismo, entre V_{DD} (5V) y V_{SS} (0V). El módulo de comparadores también permite la selección del generador de voltaje de referencia interno. Un detalle mayor se encuentra en la **Sección 21.2: Módulo generador de voltaje de referencia**. Este módulo sólo trabaja en modo de operación CM<2:0> = 110, cuatro entradas multiplexadas a 2 comparadores.

Tiempo de respuesta

El tiempo de respuesta es el mínimo tiempo después de seleccionar una nueva referencia o un cambio de fuente de entrada, antes que la salida sea válida. Si el generador de referencia interno se cambia, debe tenerse en cuenta el máximo retardo de éste para las salidas del comparador. De otra manera se debe tomar el máximo retardo de los comparadores.

Salidas de los comparadores

Las salidas son leídas en el registro CMCON. Estos bits son de **sólo-lectura** (*read-only*). Las salidas de los comparadores también pueden ser directamente conectadas a los pines RA4 y RA5. Cuando están habilitados los multiplexores (modo de operación 110), las salidas RA4 y RA5 no están sincronizadas y dependen únicamente de la entrada de los comparadores. La incertidumbre de cada comparador está relacionada con los *offset* de entrada y el tiempo de respuesta dado en las especificaciones. El registro TRISA funciona aun como un habilitador/deshabilitador de los pines RA4 y RA5, mientras está en cualquier modo. La polaridad de las salidas de los comparadores puede ser cambiada usando C2INV y C1INV del registro CMCON<4:5>.

```
NOTA:
1.  Cuando leemos el registro del Puerto A, todos los pines
configurados como entradas análogas serán leídos como “0”. Los
pines configurados como entradas digitales convertirán a una
entrada análoga a la especificación del Schmitt Trigger, es decir,
generan un valor bajo o alto como si fuera una entrada digital.

2.  Los niveles análogos de cualquier pin definido como entrada
digital pueden causar, en el buffer de entrada, un consumo mayor de
corriente del especificado.

3.  El RA4 es una salida de colector-abierto (open-collector),
cuando se usa como salida hay que poner una resistencia de pull-up.
```


libros.ep-electropc.com

Interrupciones

La bandera de interrupción del comparador es puesta por el cambio de cualquier salida de los comparadores. El software necesita mantener información acerca del estado de los bits de salida, leyendo el registro CMCOM<7:6>, para determinar el cambio actual ocurrido. En el registro **PIR**, el **CMIF** es la **bandera de interrupción del comparador** (*Comparator Interrupt Flag*), la cual se debe borrar. También puede ser puesta a uno para simular una interrupción.

El bit habilitador de interrupción del comparador **CMIE** (*Comparator Interrupt Enable*), en el registro **PIEx** (*Peripheral Interrupt Enable*), y el bit **PEIE** (*Peripheral Interrupt Enable*), del registro INTCON (*Interrupt Control*), deben ser puestos a uno. También el **GPIE** (*Global Interrupt Enable*) debe ser puesto a uno; si cualquiera de estos bits es borrado, la interrupción no se habilita aunque ocurra una condición de interrupción en el bit CMIF.

```
NOTA: Si un cambio en el registro CMCON (C1OUT o C2OUT) debe
ocurrir, cuando una operación de lectura está siendo ejecutada
(comienzo del ciclo Q2), el bit CMIF (registros PIR) podría no
darse
```

En la rutina de servicio de interrupción el usuario puede borrar la interrupción de la manera siguiente:

1. Cualquier lectura o escritura del registro CMCON puede finalizar la condición de ocurrencia. Borrar el bit de la bandera CMIF.

2. Una condición de ocurrencia puede continuar poniendo en uno el bit de la bandera CMIF. Leyendo el registro CMCON hace que termine la condición de ocurrencia y permite que el bit de la bandera CMIF se borre.

Operación en modo sleep

Cuando el comparador está activo y el MCU es puesto en modo *sleep*, el comparador permanece activo y la función de interrupción se habilita. Esta interrupción puede despertar el MCU del modo de *sleep* cuando se habilita. Cuando el comparador es alimentado, altas corrientes del modo de *sleep* pueden mostrar que una caída de voltaje (*power-down*), está ocurriendo. Cada comparador operacional puede consumir corriente adicional como aparece en las especificaciones del comparador. Para minimizar el consumo de corriente en el modo de *sleep,* apague los comparadores CM<2:0> = 111, antes de entrar al modo

libros.ep-electropc.com

de *sleep*. Si el dispositivo despierta del modo de *sleep,* el contenido del registro CMCON no se afecta.

Efectos de reset

Una condición de *reset* fuerza al registro CMCON al estado *reset*, causando que el módulo comparador se establezca en el modo *off* CM<2:0> = 111. Esto asegura la compatibilidad de los dispositivos **PIC16F87x**.

Consideración de la conexión de entradas análogas

En la Figura 21.6 se muestra un circuito simplificado para las entradas análogas. Como los pines análogos son conectados a una salida digital, tienen diodos de polarización inversa a V_{DD} y V_{SS}. Por esto la entrada análoga debe estar entre V_{SS} y V_{DD}. Si la entrada de voltaje se desvía de este rango en más de 0.6V en cualquier dirección, uno de los diodos es positivamente polarizado y puede ocurrir una condición de *latch-up.* Se recomienda una impedancia máxima de fuente de 10 kΩ. Cualquier componente externo conectado a una entrada análoga, como un **diodo Zener** o un condensador, debe tener muy poca corriente de drenaje.

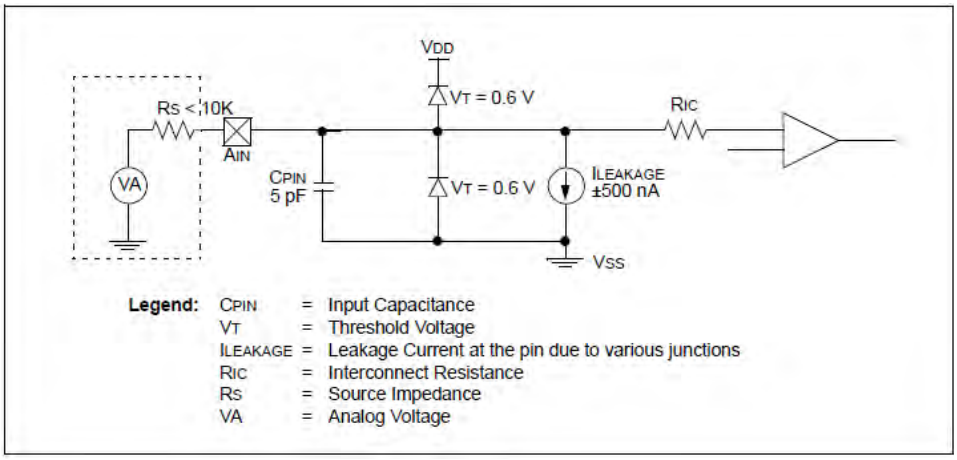


Figura 21.6 Circuito simplificado para las entradas análogas.

21.1.3 REGISTROS ASOCIADOS

En la Tabla 21.1 se describen las funciones de los pines del registro **CMCON** (9Ch), y en la Tabla 21.2, los registros asociados con el módulo de comparadores.

libros.ep-electropc.com

Tabla 21.1 Registro CMCON

R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-1	R/W-1	R/W-1
C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0
bit 7							bit 0

bit 7 **C2OUT**: Comparator 2 Output bit
When C2INV = 0:
1 = C2 VIN+ > C2 VIN-
0 = C2 VIN+ < C2 VIN-
When C2INV = 1:
1 = C2 VIN+ < C2 VIN-
0 = C2 VIN+ > C2 VIN-

bit 6 **C1OUT**: Comparator 1 Output bit
When C1INV = 0:
1 = C1 VIN+ > C1 VIN-
0 = C1 VIN+ < C1 VIN-
When C1INV = 1:
1 = C1 VIN+ < C1 VIN-
0 = C1 VIN+ > C1 VIN-

bit 5 **C2INV**: Comparator 2 Output Inversion bit
1 = C2 output inverted
0 = C2 output not inverted

bit 4 **C1INV**: Comparator 1 Output Inversion bit
1 = C1 output inverted
0 = C1 output not inverted

bit 3 **CIS**: Comparator Input Switch bit
When CM2:CM0 = 110:
1 = C1 VIN- connects to RA3/AN3
C2 VIN- connects to RA2/AN2
0 = C1 VIN- connects to RA0/AN0
C2 VIN- connects to RA1/AN1

bit 2 **CM2:CM0**: Comparator Mode bits
Figure 12-1 shows the Comparator modes and CM2:CM0 bit settings.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Tabla 21.2 Registros asociados con el módulo comparador.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other Resets
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INTIE	RBIE	TMR0IF	INTIF	RBIF	0000 000x	0000 000u
0Dh	PIR2	—	CMIF	—	—	BCLIF	LVDIF	TMR3IF	CCP2IF	-0-- 0000	-0-- 0000
8Dh	PIE2	—	CMIE	—	—	BCLIE	LVDIE	TMR3IE	CCP2IE	-0-- 0000	-0-- 0000
05h	PORTA	—	—	RA5	RA4	RA3	RA2	RA1	RA0	--0x 0000	--0u 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are unused by the comparator module.

libros.ep-electropc.com

21.1.4 PROCEDIMIENTO DE USO

Para poner a trabajar los comparadores siga el procedimiento siguiente:

- Configure las entradas y salidas en el TRISA.
- Configure el modo de operación de los comparadores en CMCON.
- Haga un retardo de 10 µs.
- Lea CMCON para actualizar datos.

A continuación le damos un código ejemplo:

1. Configure las entradas y salidas en el TRISA:

```
clrf    PORTA           ; apaga puerto A
bsf     STATUS,RP0      ; seleccione banco de registros 1
movlw   b'11001111'     ; configura entradas RA0,1,2,3 = Input, RA4,5 = Output
movwf   TRISA
```

2. Configure el modo de operación de los comparadores en CMCON:

```
movlw   b'00000011'     ; configura modo de operación: Independientes con salidas
                        ; Vin>, Vin>, No-Inv, No-Inv, -, 011
                        ; C2OUT/C1OUT/C2INV/C1INV/CIS/CM2-CM1-CM0/
movwf   CMCON
bcf     STATUS,RP0      ; retorna al banco de registros 0
```

3. Haga un retardo de 10 µs:

```
;delay_10 µs
;Retardo de carga = 10 µs , ; (Tcy = 0.2 µs, a 20 MHz) son 50 * Tcy de espera
        movlw   .16             ; cargue valor 16d
        movwf   Delay1          ; en retardo 1
DelayLoop1:
        decfsz  Delay1,f        ; decremente retardo 1
        goto    DelayLoop1      ; si no termina ciclo, vaya a DelayLoop
```

4. Lea CMCON para actualizar los datos:

```
bsf STATUS,RP0 ; seleccione Banco 1
MainLoop:
        movf    CMCON,F         ; lea puerto A y guárdelo
        goto    MainLoop        ; ciclo
```

NOTA: Observe que este módulo comparador no requiere un encendido; necesita únicamente configuración y está listo para funcionar.

libros.ep-electropc.com

21.1.5 PRÁCTICA

En el ejemplo siguiente se conectan dos comparadores c/u con sus pines de entrada **V-, V+** a dos potenciómetros y la salida OUT a un led (*a pull-up*) de lógica negativa, por lo que se pueden leer los valores en los potenciómetros con multímetro y la comparación en la salida con el led. El montaje de esta práctica incluye el sistema de evaluación **PIC16F8XX** y el módulo **MOD_COMP (2009-05-12)**, como se aprecia en la Figura 21.7.

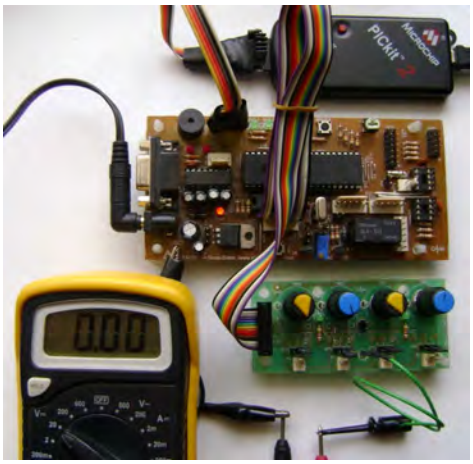


Figura 21.7 Práctica de montaje de comparadores.

Los pines que usaremos en esta práctica están detallados junto al conector **P9** en la tarjeta **PIC16F8XX**, como se aprecia en la Figura 21.8.

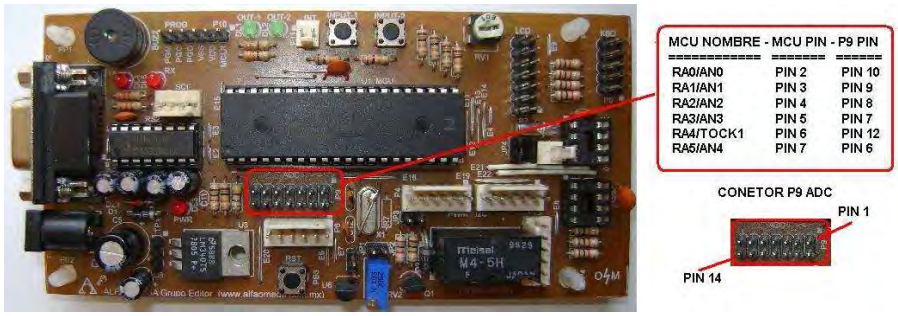


Figura 21.8 Plano esquemático comparador del conector P9.

libros.ep-electropc.com

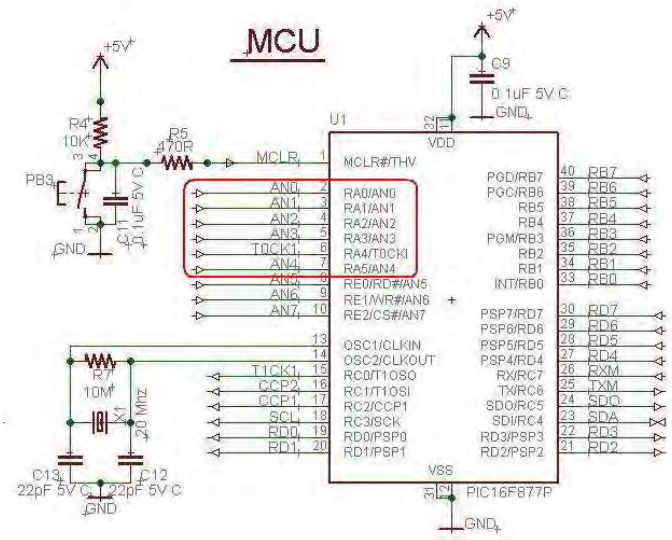
En la Tabla 21.3 se describen los pines y sus funciones que vamos a ocupar en el programa siguiente.

Tabla 21.3 Pines y función de AD.

CONEXIÓN COMPARADORES P9 A MOD_COMP (2009-05-12)				
MCU		P9		
PIN	NOMBRE	PIN	ADC	FUNCIÓN
		14	+5V	+5V
		13	+5V	+5V
6	RA4/T0CKI/C1OUT	12	TOCX1	LED DL1
		11		
2	RA0/AN0	10	AN0	POT. RV0
3	RA1/AN1	9	AN1	POT. RV1
4	RA2/AN2/VREF-/CVREF	8	AN2	POT. RV2
5	RA3/AN3/VREF+	7	AN3	POT. RV3
7	RA5/AN4/SS/C2OUT	6	AN4	LED DL2
8	RE0/RD/AN5	5	AN5	
9	RE1/WR/AN6	4	AN6	
10	RE2/CS/AN7	3	AN7	
		2	GND	GND
		1	GND	GND

El plano esquemático de conexión se describe en la Figura 21.9 a, b, c, d.

a)



libros.ep-electropc.com

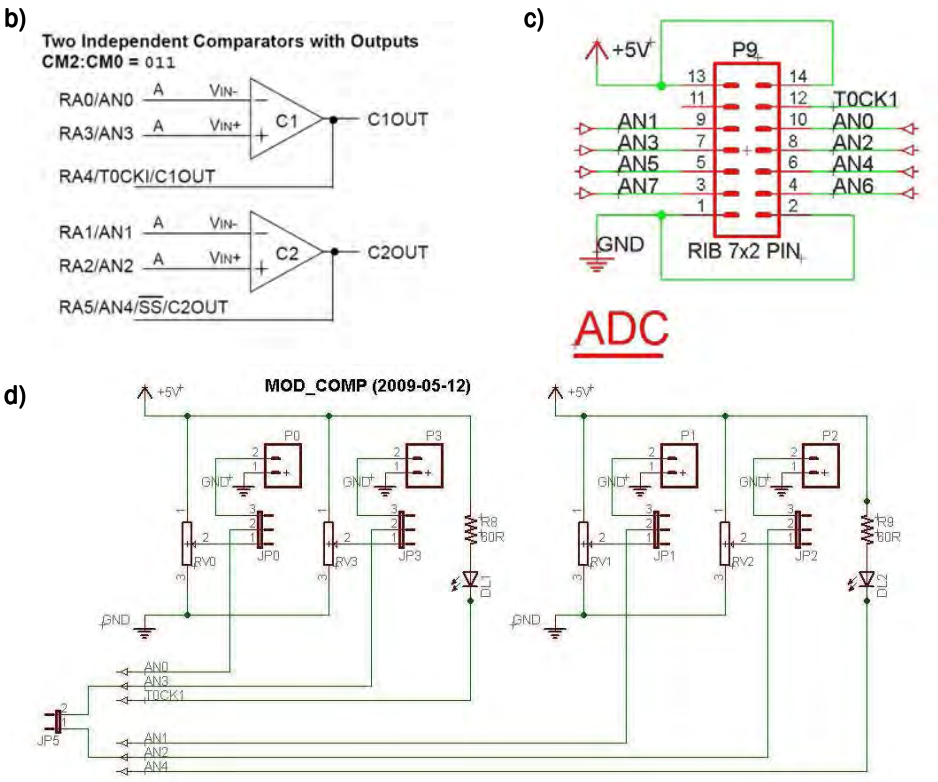


Figura 21.9 Plano esquemático comparador del conector P9.

Para realizar el ejemplo, la configuración escogida para los comparadores es 011, que se denomina:

Two Independent Comparators with Outputs CM2:CM0 = 011

En la Figura 21.10 a, b observamos una imagen del módulo **MOD_COMP (2009-05-12)** con componentes y un *screen* de la tarjeta.

a)



libros.ep-electropc.com

b)

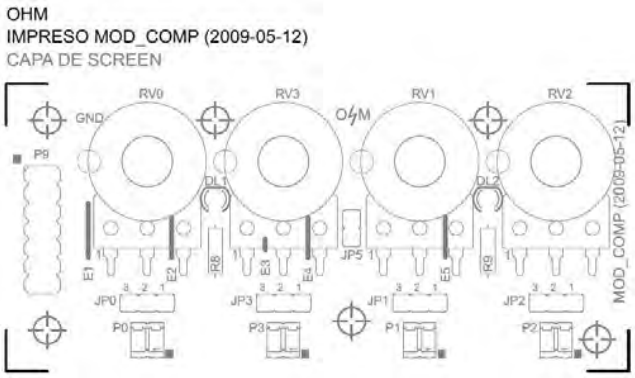


Figura 21.10 Módulo MOD_COMKP (2009-05-125).

En esta tarjeta se debe conectar el conector **P9** al conector **P9** de la tarjeta **PIC16F8X**. También se debe poner el *jumper* JP0, JP1, JP2, JP3 entre 1 y 2, para habilitar los potenciómetros RV0, RV1, RV2, RV3, con los pines AN0, AN1, AN2, AN3, AN4. En la Figura 21.10 vemos los componentes de la tarjeta **MOD_COMP (2009-05-12)**. En el circuito podemos observar cómo:

- Entre los dos potenciómetros RV0 y RV3 está el led DL1
- Entre los dos potenciómetros RV1 y RV2 está el led DL2

En esta práctica se prueban los dos comparadores con el módulo **MOD_COMP (2009-05-12)**, el cual contiene 4 potenciómetros y 2 leds con *pull-up*, así:

En el comparador COMP1:

- La salida RA4/TOCK1/C1OUT conectada a DL1 un led (*a pull-up*) de lógica negativa
- La entrada RA0/AN0 o V-, conectado al potenciómetro RV0
- La entrada RA3/AN3 o V+, conectado al potenciómetro RV3

El comparador COMP2:

- La salida RA5/AN4/C2OUT conectada a DL2 un led (*a pull-up*) de lógica negativa,
- La entrada RA2/AN2 o V-, conectada al potenciómetro RV2
- La entrada RA1/AN1 o V+, conectada al potenciómetro RV1

En nuestra página Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\ Comparador_1\ Comparador_1.asm:

libros.ep-electropc.com

```
        List p = 16f877a          ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador
__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF
            cblock 0x20          ; desde la posición 20 en RAM, separa espacio para estas variables
Delay1                           ; retardo 1
            endc
             org 0
            Start:
            clrf    PORTA                ; apague puerto A
            comf    PORTA                ; prende puerto A
; configura COMP
 bsf STATUS,RP0 ; seleccione Banco 1
            movlw   b'11001111'          ; RA0,1,2,3 Entradas R4,5 = salidas
            movwf   TRISA                ; configure puerto A
            movlw   b'00000011'          ; Vin>, Vin>, No-Inv, No-Inv, -, 011
                                         ; C2OUT/C1OUT/C2INV/C1INV/CIS/CM2-CM1-CM0/
            movwf   CMCON                ; registro 1 de configuración COMPARADOR
            bcf     STATUS,RP0           ; seleccione Banco 0
Main_Loop:
;Retardo de carga = 10 µs , ; (Tcy = 0.2 µs , a 20 MHz) son 50 * Tcy de espera
            movlw   .16                  ; cargue valor 16d
            movwf   Delay1               ; en retardo 1
DelayLoop1:
            decfsz  Delay1,f             ; decremente retardo 1
            goto    DelayLoop1           ; si no termina ciclo, vaya a DelayLoop

            bsf     STATUS,RP0           ; seleccione Banco 1
Main_conversión
            movf    CMCON,W              ; registro de configuración COMPARADOR
                                         ; se lee para Actualizar salidas C2OUT/C1OUT
            Goto    Main_Loop            ; haga ciclo
```


libros.ep-electropc.com

```
end
```

En la Tabla 21.4 apreciamos la secuencia de ajustes de potenciómetros y los cambios de estado en los leds.

Tabla 21.4 Secuencia de potenciómetros y leds.

SECUENCIA PRACTICA DE COMPARADORES					
SECUENCIA	**DESCRIPCIÒN**	**POT.**	**VOLTIOS**	**LED**	**1/0**
1	AJUSTE TODOS LOS POTENCIOMETROS A 0.00V	RVx	0.00	DLx	OFF
2	AJUSTE POTENCIOMETRO RVO = 2.50 V	RV0	2.50		
3	AJUSTE POTENCIOMETRO RV3 = 2.00 V	RV3	2.00		
	OBSERVE LED DL1 = OFF			DL1	OFF
4	AJUSTE POTENCIOMETRO RV3 = 3.00 V	RV3	2.00		
	OBSERVE LED DL1 = ON			DL1	ON
5	AJUSTE TODOS LOS POTENCIOMETROS A 0.00V	RVx	0.00	DLx	OFF
6	AJUSTE POTENCIOMETRO RV2 = 2.50 V	RV2	2.50		
7	AJUSTE POTENCIOMETRO RV3 = 2.00 V	RV1	2.00		
	OBSERVE LED DL2 = OFF			DL2	OFF
8	AJUSTE POTENCIOMETRO RV1 = 3.00 V	RV1	2.00		
	OBSERVE LED DL2 = ON			DL2	ON

El programa anterior, en modo de run, y la realización de las secuencias anteriores nos arrojan los resultados que vemos en las imágenes siguientes.



Figura 21.11 Secuencia 1.

Secuencia 1. Todos los potenciómetros están a 0.00V; observe cómo todos están en su tope mínimo.



Figura 21.12 Secuencia 2

libros.ep-electropc.com

Secuencia 2. Ajuste el potenciómetro a RV3 = 2.50V, observe cómo RV3 queda en la mitad de su recorrido.



Figura 21.13 Secuencia 3.

Secuencia 3. Ajuste el potenciómetro a RV0 = 2.00V, observe cómo RV2 se mueve en su recorrido, el led DL1 está apagado (OFF).

Así tenemos que: RV3 = 2.50 > RV0 = 2.00 = > DL1 = OFF
V+ = RA3 > V- = RA0 OUT = C1OUT = 1



Figura 21.14 Secuencia 4.

Secuencia 4. Ajuste el potenciómetro a RV0 = 3.00V, observe cómo RV2 se mueve en su recorrido, el led DL1 está apagado ON.

Así tenemos que: RV3 = 2.50 < RV0 = 3.00 = > DL1 = ON
V+ = RA3 < V- = RA0 OUT = C1OUT = 0



Figura 21.15 Secuencia 5.

libros.ep-electropc.com

Secuencia 5. Todos los potenciómetros están a 0.00V; observe cómo todos los potenciómetros están en su tope mínimo.



Figura 21.16 Secuencia 6.

Secuencia 6. Ajuste el potenciómetro a RV2 = 2.50V; observe cómo RV2 queda en la mitad de su recorrido.



Figura 21.17 Secuencia 7.

Secuencia 7. Ajuste el potenciómetro a RV1 = 2.00V; observe cómo RV1 se mueve en su recorrido, el led DL2 está apagado (OFF).

Así tenemos que:

RV2 = 2.50 > RV1 = 2.00 = > DL2 = OFF
V+ = RA2 > V- = RA1 OUT = C2OUT = 1



Figura 21.18 Secuencia 8.

libros.ep-electropc.com

Secuencia 8. Ajuste el potenciómetro a RV1 = 3.00V; observe cómo RV1 se mueve en su recorrido, el led DL1 está apagado ON.

Así tenemos que: RV2 = 2.50 < RV1 = 3.00 = > DL2 = ON
V+ = RA2 < V- = RA1 OUT = C2OUT = 0

En nuestra página Web puede encontrar este ejemplo con subrutinas en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\ Comparador_1\ Comparador_1A.asm.

EVALUACIÓN

```
P.: 1. ¿Cuáles son los nombres de los pines de un comparador?
R.:
V+,V-,OUT.

P.: 2. ¿Cuántos modos de conexión del comparador existen?
R.:
8, programables con los bits CM2, CM1, CM0 en el registro CMCON.

P.: 3. ¿En cuál de los modos de conexión del comparador se conecta
el voltaje de referencia (CVREF)?
R.:
En el modo 110.
```

21.2 MÓDULO GENERADOR DE VOLTAJE DE REFERENCIA

El **generador de voltaje de referencia** del comparador es un módulo interno que genera un voltaje fijo de referencia controlado por un registro programable. Genera salida para el módulo comparador y como salida directa del pin **4** del **MCU, CVREF** en el pin **RA2**.

21.2.1 DIAGRAMA CIRCUITAL

En la Figura 21.19a, b observamos el diagrama de bloques del Generador de voltaje de referencia, y cómo el circuito resistivo de **24R** se forma de 8R, luego las 16 resistencias (**R**) del arreglo configurable, más la **8R** al terminar; esta disposición permite que pueda haber dos circuitos en donde se pueden dar dos tipos de valores.

libros.ep-electropc.com

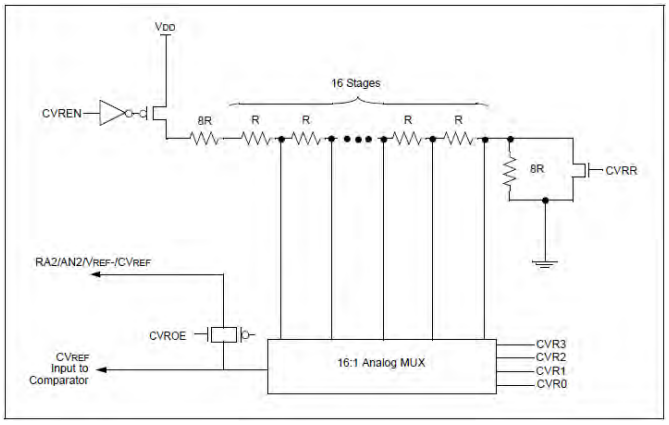


Figura 21.19 Esquema del voltaje de referencia.

21.2.2 FUNCIONAMIENTO

El **generador de voltaje de referencia** del comparador es un arreglo resistivo de 16 pasos, que provee un voltaje fijo de referencia, el registro programable **CVRCON** controla la función del generador de referencia. Este voltaje se puede usar como referencia para el módulo de comparación en el modo **"110"**, o como salida directa en el pin **4** del **MCU, CVREF**. En la Tabla 21.6 se muestra la configuración de los bits del registro **CVRCON** (*Comparator Voltage Reference Control*), el cual contiene todos los bits de control para el manejo total del módulo de voltaje de referencia.

Como vemos en la Figura 21.19, el arreglo resistivo puede tomar la forma de dos circuitos, uno de 32R y otro de 24R; según se active o no el bit **CVRR** (*Comparator VREF Range Selection*), es segmentado para dos rangos de valores de CVREF; también tiene una función de **bajo consumo** (*power-down*), cuando la referencia no está siendo usada, con el bit **CVREN** (*Comparator Voltage Reference Enable*). La alimentación de voltaje del comparador de referencia (también referenciada como CVRSRC) está directamente conectada a V_{DD}. Sin embargo, hay que anotar que el voltaje mayor del arreglo es CVRSRC-VSAT, donde VSAT es el voltaje de saturación del transistor de "switcheo" (conmutación). Esa referencia puede ser tan exacta como los valores de CVRSRC y VSAT.

La salida del generador de referencia puede ser conectada al pin 4 del MCU, RA2/AN2/VREF-/CVREF. El usuario puede usar esta salida como una simple función D/A (digital a análogo), por si es colocada una alta impedancia de salida. El objetivo primordial de esta función es proveer un punto de prueba para verificar la función del módulo generador de referencia.

libros.ep-electropc.com

También se puede conectar la salida de Vref al modulo de comparadores en el modo CM2:CM0 = 110 como se ilustra en la Figura

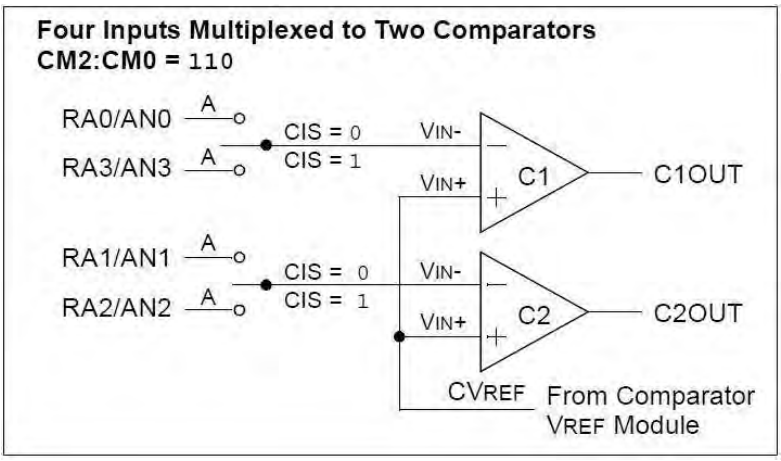


Figura 21.20 Voltaje de referencia a comparadores.

NOTA: Al trabajar con la salida del generador de referencia por el pin RA2 debe ser cuidadoso, pues un sobrevoltaje o una sobrecorriente por este pin, en este modo de uso, podría dañar el módulo generador de referencia.

A continuación (Tabla 21.5) revisaremos los rangos y valores que se pueden generar con estos dos arreglos resistivos.

libros.ep-electropc.com

Tabla 21.5 Cálculos del voltaje de referencia.

CTO		LARGO = 32R	CORTO = 24R
CVRR		0	1
R-HIGH		8	8
R-MIDLE		16	16
R-LOW		8	0
R - STEP		32	24
5V - STEP	Voltios	0,156	0,208
MIN		8/32 = 1/4	0
		0,25*Vdd	0
	Voltios	1,25	0
MAX		24/32 = 3/4	16/24 = 2/3
		0,75*Vdd	0,67*Vdd
	Voltios	3,75	3,35
Vref	Voltios	(Vdd/32)*(VR<3:0>) + (Vdd/4)	(Vdd/24)*(VR<3:0>)

CV3:CV0		Vref (voltios)	
0	0000	1,25000	0
1	0001	1,40625	0,20833
2	0010	1,56250	0,41667
3	0011	1,71875	0,62500
4	0100	1,87500	0,83333
5	0101	2,03125	1,04167
6	0110	2,18750	1,25000
7	0111	2,34375	1,45833
8	1000	2,50000	1,66667
9	1001	2,65625	1,87500
10	1010	2,81250	2,08333
11	1011	2,96875	2,29167
12	1100	3,12500	2,50000
13	1101	3,28125	2,70833
14	1110	3,43750	2,91667
15	1111	3,59375	3,12500
16		3,75000	3,33333

En la Tabla 21.5 vemos cómo los valores se refieren a dos circuitos: uno **largo** de 32R y uno **corto** de 24 R, que corresponden a los dos rangos de selección que se pueden obtener con el bit CVRR Comparador VREF Range bit en el registro CVRCON. Con CVRR = 1, el circuito contiene 32R y los valores están entre ¼ V_{DD} = 1.25V y ¾ V_{DD} = 3.75V, en la Tabla 21.5 la columna de circuito LARGO = 32R. Con CVRR = 0, el circuito contiene 24R y los valores están entre 0 V_{DD} = 0V y 2/3 V_{DD} = 3.35V, en la Tabla 21.5 la columna de circuito CORTO = 24R. Para seleccionar el voltaje que se va a leer en el arreglo resistivo se usan los bits CVR<3:0>; con estos 4 bits se pueden obtener hasta 16 valores diferentes que corresponden a las 16 resistencias conectadas al circuito.

libros.ep-electropc.com

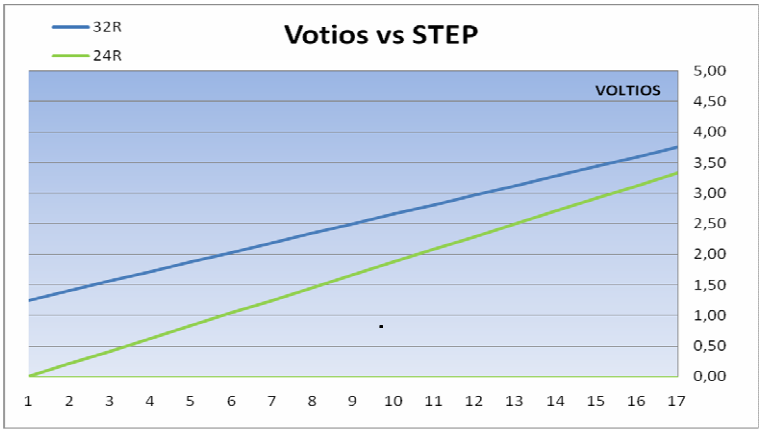


Figura 21.20 Cálculos del voltaje de referencia.

En la Figura 21.20 apreciamos el rango de ambos circuitos con respecto a 5V. La línea superior refleja los valores de voltaje generados con el circuito LARGO = 32R, la línea inferior los valores con el circuito CORTO = 24R.

21.2.3 REGISTROS ASOCIADOS

El registro CVRCON es el registro de control del módulo (Tabla 21.6). En la Tabla 21.7 vemos los registros asociados con el módulo generador de voltaje.

Tabla 21.6 Registro CVRCON.

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0
bit 7							bit 0

bit 7 **CVREN**: Comparator Voltage Reference Enable bit
1 = CVREF circuit powered on
0 = CVREF circuit powered down

bit 6 **CVROE:** Comparator VREF Output Enable bit
1 = CVREF voltage level is output on RA2/AN2/VREF-/CVREF pin
0 = CVREF voltage level is disconnected from RA2/AN2/VREF-/CVREF pin

bit 5 **CVRR**: Comparator VREF Range Selection bit
1 = 0 to 0.75 CVRSRC, with CVRSRC/24 step size
0 = 0.25 CVRSRC to 0.75 CVRSRC, with CVRSRC/32 step size

bit 4 **Unimplemented:** Read as '0'

bit 3-0 **CVR3:CVR0:** Comparator VREF Value Selection bits 0 ≤ VR3:VR0 ≤ 15
When CVRR = 1:
CVREF = (VR<3:0>/ 24) • (CVRSRC)
When CVRR = 0:
CVREF = 1/4 • (CVRSRC) + (VR3:VR0/ 32) • (CVRSRC)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 21.7 Registros asociados con el voltaje de referencia comparador.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR	Value on all other Resets
9Dh	CVRCON	CVREN	CVROE	CVRR	—	CVR3	CVR2	CVR1	CVR0	000- 0000	000- 0000
9Ch	CMCON	C2OUT	C1OUT	C2INV	C1INV	CIS	CM2	CM1	CM0	0000 0111	0000 0111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'.
Shaded cells are not used with the comparator voltage reference.

21.2.4 PROCEDIMIENTO DE USO

Este módulo **generador de voltaje de referencia** puede usar dos configuraciones: como salida directa al pin RA2 = CVREF, o como entrada del módulo comparador.

Como salida directa

Para poner a trabajar el **generador de voltaje de referencia** como salida directa al pin RA2 = CVREF, siga el procedimiento siguiente:

- Seleccione el valor de R (CVR<3:0>).
- Seleccione el rango de operación (cto. 32R o 24R).
- Encienda el módulo de voltaje de referencia.
- Conecte el pin de salida CVREF.
- Haga un retardo de 10 µs.

Lea con voltímetro en el pin el voltaje programado.

A continuación le damos un código ejemplo:

1. Seleccione el valor de R (CVR<3:0>):

```
    bsf STATUS,RP0 ; seleccione Banco 1

; selecciona valor resistivo:
;   movlw   b'00000000'     ; cargue valor para selección de 16R, 0000 -> V = 1.25
    movlw   b'00001000'     ; cargue valor para selección de 16R, 1000 -> V = 2.5
;   movlw   b'00001111'     ; cargue valor para selección de 16R, 1111 -> V = 3.75
                            ; /CVREN/CVROE/CVRR/-/CV3:0/
    movwf   CVRCON          ; cargue en CVR<3:0>
```

2. Seleccione el rango de operación (cto. 32R o 24R):

```
    bcf     CVRCON,CVRR     ; seleccione rango de operación 0 = 32R, 1 = 24R
;   bsf     CVRCON,CVRR     ; seleccione rango de operación 0 = 32R, 1 = 24R
```


libros.ep-electropc.com

3. Encienda el módulo de voltaje de referencia:

```
bsf     CVRCON,CVREN    ; Prenda módulo de voltaje de referencia
```

4. Conecte el pin de salida CVREF:

```
bsf     CVRCON,CVROE    ; conecte a pin CVREF
bcf     STATUS,RP0      ; seleccione Banco 0
```

5. Haga un retardo de 10 µs.

Lea con voltímetro en el pin el voltaje programado. A partir de aquí está disponible el voltaje de $V_{DD}/2$ en el pin RA2 = CVREF.

```
NOTA: En algunos módulos, el orden de los pasos de configuración de
los bits no es tan relevante para su funcionamiento, como en el
módulo generador de voltaje de referencia, siempre y cuando haya un
paso de encendido del módulo, una línea como:
bsf   CVRCON,CVREN  ; prende el módulo de voltaje de referencia.
```

21.2.5 PRÁCTICA

En esta práctica haremos dos ejemplos con el módulo generador de voltaje de referencia. En el **Ejemplo 1** se genera un voltaje de referencia fijo de 2.50 voltios en el pin RA2. En el **Ejemplo 2** se genera un voltaje de referencia que se incremente en los 16 pasos en el pin RA2. Para ambos ejemplos se utiliza el montaje que se observa en la Figura21.21.

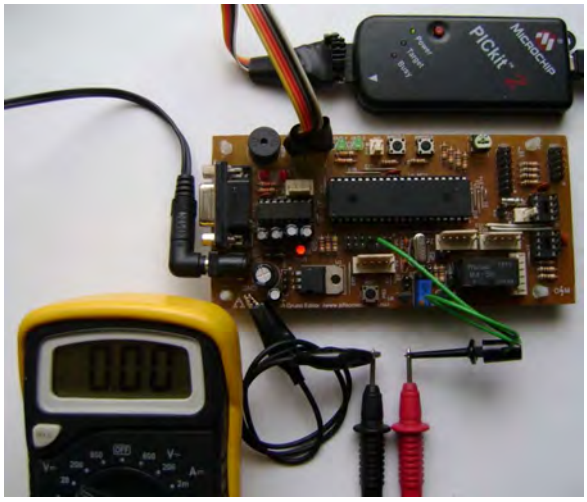


Figura 21.21 Montaje del voltaje de referencia, Ejemplos 1 Y 2

libros.ep-electropc.com

Los pines que usaremos en esta práctica están detallados junto al conector **P9** en la tarjeta **PIC16F8XX**, como se muestra en la Figura 21.22.

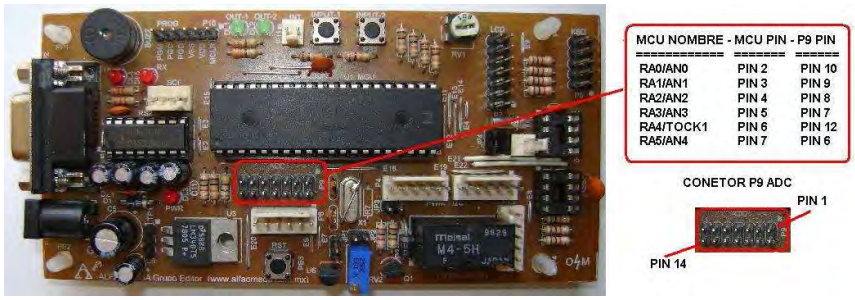


Figura 21.22 Plano esquemático comparador conector P9.

Los pines y sus funciones que ocuparemos en el programa siguiente se describen en la Tabla 21.8.

Tabla 21.8 Pines y función de generador de voltaje de referencia.

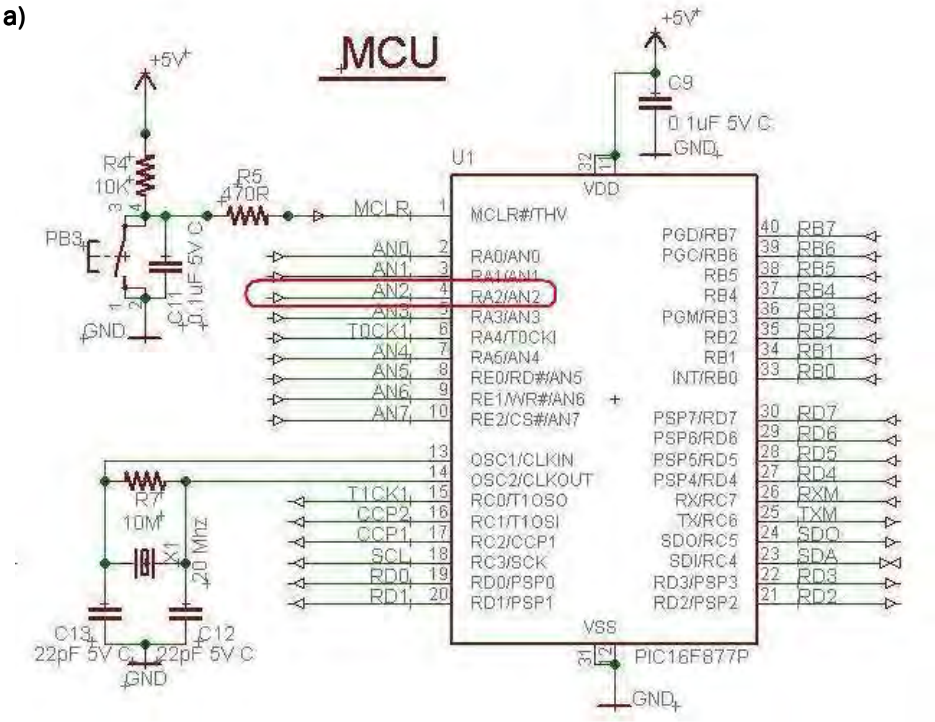
CONEXION VOLTAJE DE REFERENCIA
CONECTOR P9

MCU		P9		
PIN	NOMBRE	PIN	ADC	FUNCIÓN
		14	+5V	+5V
		13	+5V	+5V
6	RA4/TOCKI/C1OUT	12	TOCK1	
		11		
2	RA0/AN0	10	AN0	
3	RA1/AN1	9	AN1	
4	RA2/AN2/VREF-/CVREF	8	AN2	RA2/CVREF
5	RA3/AN3/VREF+	7	AN3	
7	RA5/AN4/SS/C2OUT	6	AN4	
8	RE0/RD/AN5	5	AN5	
9	RE1/WR/AN6	4	AN6	
10	RE2/CS/AN7	3	AN7	
		2	GND	GND
		1	GND	GND

libros.ep-electropc.com

El plano esquemático de conexión se describe en la Figura 21.23 a, b.

a)



b)

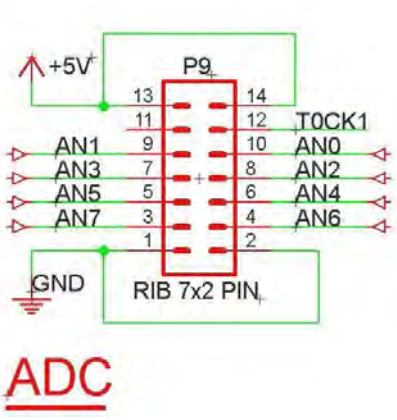


Figura 21.23 Plano esquemático generador de voltaje de referencia

libros.ep-electropc.com

Ejemplo 1

En este ejemplo, en modo de *run*, se pone el valor $V_{DD}/2$ del **voltaje de referencia** directamente sobre el pin **RA2** que se convierte en **CVREF**. El valor del número de resistencias en el circuito **generador de voltaje de referencia** está dado por **CVR<3:0>** en el registro **CVRCON**. En el programa, en el *label* pone valor resistivo donde se cambia este valor, con los últimos 4 bits del byte que se va a cargar en el registro **W**.

Observe que hay 2 valores comentados, para **V = V_{DD} * ¼ = 1.25** y para **V = V_{DD} * ¾ = 3.75**, con los cuales al habilitarlos puede cambiar el valor leído en el pin **RA2**.

En nuestra página Web ubicada en http://virtual.alfaomega.com.mx puede encontrar este ejemplo en la carpeta **X:\Ejemplos_ASM**, la que debe copiar en el disco **C:** para encontrarla como:
C:\Ejemplos_ASM\VOLT_REF_1\VOLT_REF_1.asm.

```
            List p = 16f877a      ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

            org 0
; configura voltaje de referencia
            bsf         STATUS,RP0          ; seleccione Banco 1

; selecciona valor resistivo:
;           movlw       b'00000000'         ; cargue valor para selección de 16R, 0000 -> V = 1.25
            movlw       b'00001000'         ; cargue valor para selección de 16R, 1000 -> V = 2.5
;           movlw       b'00001111'         ; cargue valor para selección de 16R, 1111 -> V = 3.75
                                            ; /CVREN/CVROE/CVRR/-/CV3:0/
            movwf       CVRCON              ; cargue en CVR<3:0>

            bcf         CVRCON,CVRR         ; seleccione rango de operación, 0 = 32R, 1 = 24R
            bsf         CVRCON,CVREN        ; Prenda módulo de voltaje de referencia

            bsf         CVRCON,CVROE        ; conecte a pin CVREF/RA2/AN2
            bcf         STATUS,RP0          ; seleccione Banco 0

            goto        $

            end
```

Corriendo el programa anterior en modo de *run*, se pueden obtener los valores siguientes.

Los valores obtenidos son el resultado de habilitar las líneas de código:

libros.ep-electropc.com

```
;       movlw   b'10000000'     ; cargue valor para selección de 16R, 0000 -> V = 1.25
        movlw   b'10001000'     ; cargue valor para selección de 16R, 1000 -> V = 2.5
;       movlw   b'10001111'     ; cargue valor para selección de 16R, 1111 -> V = 3.75
```

que cambian el valor de CV3:CV0 y, por tanto, el valor generado.

Tabla 21.9 Valores de la práctica.

			VDD	VDD		
			5,00	4,86		
			TEORICO	REAL	CALCULADO	
CV3:CV0		FORMULA	Vref (voltios)	Vref (voltios)	Vref (voltios)	ERORROR (mV)
0	0000	VDD*1/4	1,250	1,180	1,215	0,035
8	1000	VDD*1/2	2,500	2,370	2,430	0,060
15	1111	(Vdd/32)*(VR<3:0>) + (Vdd/4)	3,594	3,430	3,493	0,063
16	Extrapolado	VDD*3/4	3,750			

En la Tabla 21.9 observamos cómo los valores de la columna **real**, son diferentes a los de la columna **teórico**, porque el voltaje de alimentación V_{DD} no es 5.00 voltios, sino de 4.86 V; además, se deben considerar los voltajes de caída en el *mosfet* que se activa con CVREN. Al utilizar este módulo es necesario considerar que los valores de lectura **reales** pueden tener un error con respecto a los valores **teóricos**.

Ejemplo 2

En el ejemplo siguiente, continuando paso a paso, se incrementa el voltaje de referencia en un paso, o sea desde 0R a 15R, cuyos valores generan voltajes de referencia entre V_{DD} * ¼ a V_{DD} * ¾, en el pin RA2 = CVREF. En nuestra página Web puede encontrar este ejemplo en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM\VOLT_REF_1\VOLT_REF_1A.asm.

```
        List p = 16f877a        ; directiva para definir procesador
#include<p16f877a.inc>          ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF &
_CPD_OFF

        cblock  0x71    ; define variables a partir de posición 0x71 en RAM = GPR
contador                ; contador de 0 a 15 valores de tabla
aux                     ; registro auxiliar
        endc

#define _filtro_F0h     0xF0            ; filtro de nibble alto
#define _bit_4          4               ; bit 4

 org 0
; configura voltaje de referencia
```


libros.ep-electropc.com

```
            Bsf     STATUS,RP0          ; seleccione Banco 1

            movlw   b'00000000'         ; 0R de 16R, para valor         RA2 = CVREF = 1,250
            movwf   CVRCON              ; cargue en CVR<3:0>
            bcf     CVRCON,CVRR         ; seleccione rango de operación, 0 = 32R, 1 = 24R
;           bsf     CVRCON,CVRR         ; seleccione rango de operación, 0 = 32R, 1 = 24R

            bsf     CVRCON,CVREN        ; Prenda módulo de voltaje de referencia

            bsf     CVRCON,CVROE        ; conecte a pin CVREF
            bcf     STATUS,RP0          ; seleccione Banco 0

Main_ini:
            clrf    contador  ; borre contador

Main_Loop:
; selecciona valor resistivo:
            movf    contador,W          ; cargue contador -> W
            call    Tabla_CV30          ; traiga valor de configuración CV3:CV0 en W

            movwf   aux                 ; guarde W -> auxiliar

 bsf STATUS,RP0 ; seleccione Banco 1
            movlw   filtro_F0h          ; haga filtro a nibble alto
            andwf   CVRCON,W            ; filtre CVRCON -> W
            iorwf   aux,W               ; haga or con aux1 en nibble bajo (CVR<3:0>)
            movwf   CVRCON              ; guarde CVRCON, cargo CVR<3:0>
            bcf     STATUS,RP0          ; seleccione Banco 0

; check si contador = 16
            incf    contador            ; contador + 1
            btfss   contador,_bit_4     ; check si contador = 16. Sí, salte
            goto    Main_Loop           ; no, vaya a ...
            goto    Main_ini            ; vaya a .., inicia en 0 contador

Tabla_CV30:                 ; valores de CV3:CV0 para selección de resistencias en arreglo de 16R
            addwf   PCL,F   ; sume W + PCL -> PCL

            retlw   b'00000000'         ; 0R de 16R, para valor         RA2 = CVREF = 1,250
            retlw   b'00000001'         ; 1R de 16R, para valor         RA2 = CVREF = 1,406
            retlw   b'00000010'         ; 2R de 16R, para valor         RA2 = CVREF = 1,563
            retlw   b'00000011'         ; 3R de 16R, para valor         RA2 = CVREF = 1,719
            retlw   b'00000100'         ; 4R de 16R, para valor         RA2 = CVREF = 1,875
            retlw   b'00000101'         ; 5R de 16R, para valor         RA2 = CVREF = 2,031
            retlw   b'00000110'         ; 6R de 16R, para valor         RA2 = CVREF = 2,188
            retlw   b'00000111'         ; 7R de 16R, para valor         RA2 = CVREF = 2,344
            retlw   b'00001000'         ; 8R de 16R, para valor         RA2 = CVREF = 2,500
            retlw   b'00001001'         ; 9R de 16R, para valor         RA2 = CVREF = 2,656
            retlw   b'00001010'         ; 10R de 16R, para valor        RA2 = CVREF = 2,813
            retlw   b'00001011'         ; 11R de 16R, para valor        RA2 = CVREF = 2,969
            retlw   b'00001100'         ; 12R de 16R, para valor        RA2 = CVREF = 3,125
            retlw   b'00001101'         ; 13R de 16R, para valor        RA2 = CVREF = 3,281
            retlw   b'00001110'         ; 14R de 16R, para valor        RA2 = CVREF = 3,438
            retlw   b'00001111'         ; 15R de 16R, para valor        RA2 = CVREF = 3,594

            end
```


libros.ep-electropc.com

Con el programa anterior, ejecutado paso a paso, se lograron tomar las lecturas con V_{DD} = 4.86V. Observe que hay voltajes **teóricos** que son calculados con V_{DD} = 5V, y **lecturas** que son los valores leídos con el multímetro. En la Tabla 21.10 se encuentran los valores de las lecturas para circuito **largo** de 32R. Observe que los valores van desde 1.25 a 3.75 voltios **teóricos** y las lecturas reales en **LECTURAS**.

Tabla 21.10 Valores de la práctica.

VDD	VDD
5,00	4,86

		LARGO = 32R	
		Vref (voltios)	
CV3:CV0		TEORICO	LECTURAS
0	0000	1,250	1,18
1	0001	1,406	1,33
2	0010	1,563	1,48
3	0011	1,719	1,63
4	0100	1,875	1,77
5	0101	2,031	1,92
6	0110	2,188	2,07
7	0111	2,344	2,22
8	1000	2,500	2,37
9	1001	2,656	2,52
10	1010	2,813	2,68
11	1011	2,969	2,83
12	1100	3,125	2,98
13	1101	3,281	3,13
14	1110	3,438	3,28
15	1111	3,594	3,43
16	Extrapolado	3,750	

En el programa anterior, para cambiar de circuito largo 32R a 24R, se puede cambiar la línea:

```
    bcf     CVRCON,CVRR     ; seleccione rango de operación, 0 = 32R, 1 = 24R
```

Por la línea:

```
    bsf     CVRCON,CVRR     ; seleccione rango de operación, 0 = 32R, 1 = 24R
```

La mejor manera de hacer esto es comentando la primera línea:

```
;   bcf     CVRCON,CVRR     ; seleccione rango de operación, 0 = 32R, 1 = 24R
    bsf     CVRCON,CVRR     ; seleccione rango de operación, 0 = 32R, 1 = 24R
```


libros.ep-electropc.com

En la Tabla 21.11 se encuentran los valores de las lecturas para circuito corto de 24R. Observemos que los valores van desde 0.00 a 3.33 voltios **teóricos** y las lecturas **reales** en **LECTURAS**.

Tabla 21.11 Valores de la práctica.

VDD	VDD
5,00	4,86

		CORTO = 24R	
		Vref (voltios)	
CV3:CV0		TEORICO	LECTURAS
0	0000	0,000	0,00
1	0001	0,208	0,21
2	0010	0,417	0,4
3	0011	0,625	0,6
4	0100	0,833	0,78
5	0101	1,042	0,99
6	0110	1,250	1,18
7	0111	1,458	1,38
8	1000	1,667	1,58
9	1001	1,875	1,77
10	1010	2,083	1,97
11	1011	2,292	2,17
12	1100	2,500	2,38
13	1101	2,708	2,58
14	1110	2,917	2,78
15	1111	3,125	2,98
16	Extrapolado	3,333	

Al trabajar con este módulo, se recomienda realizar las lecturas de todos los valores reales antes de hacer una aplicación; de no ser así se trabaja con valores teóricos que tienen un error del valor real. Este ejemplo y el anterior con subrutinas y retardos puede encontrarlos en nuestra página Web en la carpeta X:\Ejemplos_ASM, la que debe copiar en el disco C:\ para encontrarla como C:\Ejemplos_ASM \VOLT_REF_1\VOLT_REF_1B.

libros.ep-electropc.com

EVALUACIÓN

```
P.: 1. ¿Cuántos tipos de circuitos hay en el módulo generador de
voltaje de referencia?
R.:
Hay dos, los circuitos de 32R y 24R.

P.: 2. ¿Cuáles son los voltajes máximos y mínimos que se pueden
generar con los circuitos de 32R y 24R?
R.:
1.    Con el circuito de 32R.
      Vmín. = 1.25V, Vmáx. = 3.75V.
2.    Con el circuito de 24R.
      Vmín. = 0V, Vmáx. = 3.35V.
```


libros.ep-electropc.com

PARTE 6

TIMERS Y PWM (CCP)

En esta Parte 6 encontramos dos módulos importantes: los **temporizadores** (**TIMERS)** para aplicaciones como temporización y conteo de eventos de gran precisión, y el módulo **PWM** que realmente es el módulo **captura/comparación CCP** (**Capture/Compare/PWM**), utilizados para contar el tiempo entre eventos, y como generador de señales, ambos de gran resolución (16 bits), y una salida de **PWM** para el control de motores de **DC** con 10 bits de resolución. Algunas aplicaciones donde se pueden usar estos dispositivos podrían ser: en bandas transportadoras para medición de anchos de objetos, en el control de velocidad de motores de **DC** y **AC**, paso a paso, controles remotos infrarrojos inalámbricos, encoders, medidores de efecto *Doppler*, etcétera.

22.0 TEMPORIZADORES (TIMERS)

Este **MCU** en particular contiene **3 Timers: Timer0, Timer1** y **Timer2**; cada uno es independiente de los otros en su funcionamiento, y tienen resoluciones de 8 y 16 bits, con *prescaler* y *postscaler*; también con pines de entrada para conteo de eventos o con entrada de ciclos de reloj del MCU.

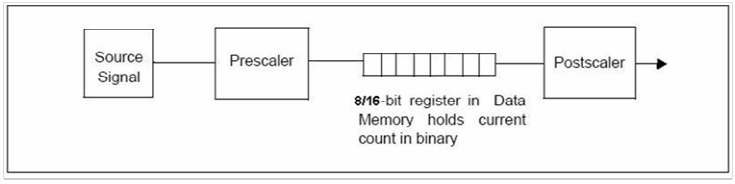


Figura 22.1 Diagrama circuital.

libros.ep-electropc.com

En la Figura 22.1 se muestra un diagrama simplificado de la configuración de los Timers de 8 o 16 bits.

22.1 TIMER0

Este módulo **TIMER0** puede funcionar como **temporizador** (*timer*), o como **contador** (*counter*); tiene un registro de **8 bits TMR0**, y un *prescaler* (**preescalado**) de 8 bits que divide el tiempo o el conteo; es compartido y excluyente entre el *timer* y el *watchdog* (el **perro guardián**). Podemos insertar pulsos de *clock* (**reloj**) para este módulo por el pin **RA4/T0CKI**.

22.1.1 DIAGRAMA CIRCUITAL

En la Figura 22.2 apreciamos el diagrama de bloques del módulo **TIMER0** y el *prescaler* compartido con el *watchdog*.

FIGURE 5-1: BLOCK DIAGRAM OF THE TIMER0/WDT PRESCALER

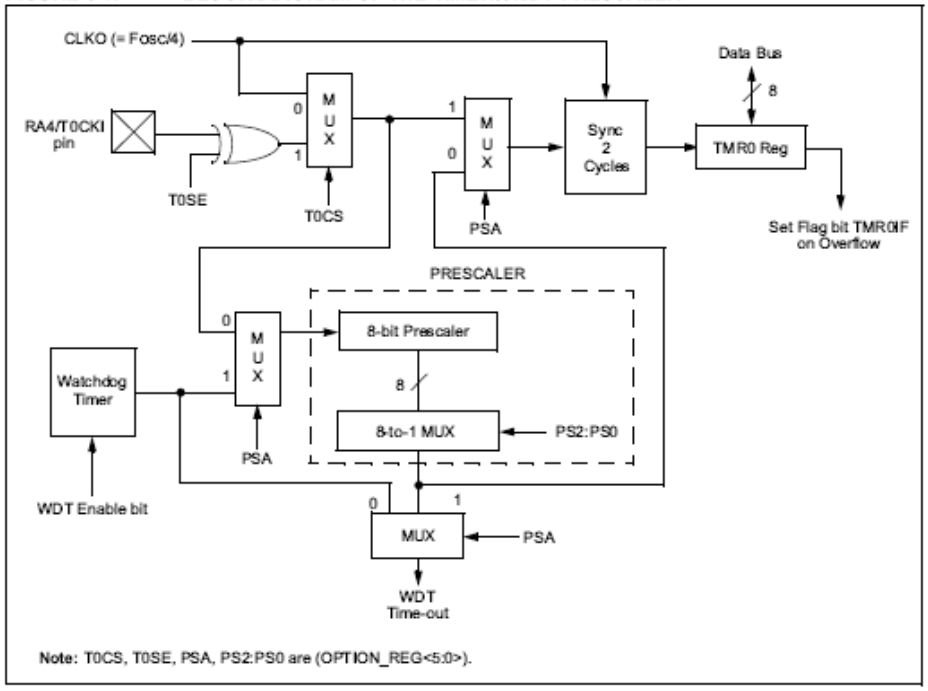


Figura 22.2 Diagrama circuital del TIMER 0/*watchdog timer prescaler*.

libros.ep-electropc.com

22.1.2 FUNCIONAMIENTO

El módulo **TIMER0** tiene dos modos de operación: como **temporizador** (*timer*), o como **contador** (*counter*). Veremos cada uno de los modos de operación y el comportamiento del *prescaler* y la interrupción.

Características:

- Temporizador/contador de 8 bits
- Lectura y escritura
- Preescalado de 8 bits programable por software
- Selector de reloj interno o externo
- Interrupción de sobreflujo (*overflow*) desde FFh a 00h
- Selección de flanco para *clock* externo

22.1.2.1 MODO TEMPORIZADOR (TIMER)

El modo de temporizador es seleccionado por borrar el bit **T0CS** en el registro **OPTION_REG<5>**. En el modo de **temporizador** (timer) el registro **TMR0** puede incrementar cada ciclo de instrucción (sin *prescaler*); y con *prescaler* puede dividir los pulsos hasta por **256**. Si escribimos en el registro **TMR0**, el incremento es inhibido por los dos (**2**) ciclos de instrucción siguientes. El usuario debe tener en cuenta estos ciclos para poner el valor adecuado en el registro **TMR0**.

```
NOTA: En el modo timer los pulsos provienen directamente de CLKO =
Fosc/4, del cristal oscilador /4, y se le puede asignar o no
prescaler.

Sin prescaler la relación es de 1:1.

Con prescaler la relación puede ser desde 1:2 hasta 1:256.
```

Para que observe las relaciones del *prescaler,* lo remitimos a la Tabla 22.1, más adelante.

22.1.2.2 MODO CONTADOR (COUNTER)

El **modo de contador** se selecciona al poner a uno el bit **T0CS** en el registro **OPTION_REG<5>**. En el modo de contador, el registro **TMR0** puede incrementarse con cada pulso, ya sea de **flanco de subida** (*rising*) o el **flanco de caída** (*falling*) sobre el pin **RA4/T0CKI**. El flanco de incremento es determinado por el bit T0SE del registro **OPTION_REG<4>**. Borrando el bit T0SE en el registro OPTION_REG<4>,

libros.ep-electropc.com

selecciona el flanco de subida (*rising*). Las restricciones sobre la entrada del *clock* externo se ven en el apartado "Timer0 con *clock* externo".

```
NOTA: En el modo counter, los pulsos provienen directamente del pin
RA4/TOCKI y no tienen prescaler. Su relación es de 1:1 cuando el
prescaler está asignado al WDT. Pero cuando el prescaler se asigna
al TMR0 con cualquier valor del prescaler PS<2:0>, la relación
siempre es de 1:2.
```

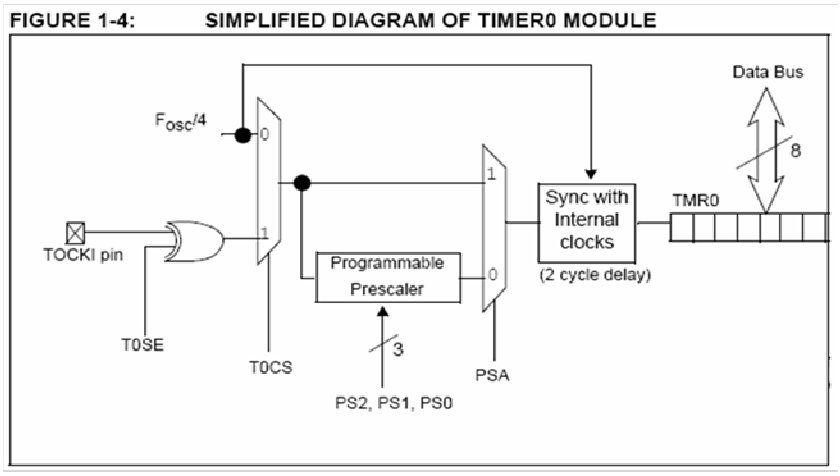


Figura 22.3 Diagrama simplificado del TMR0.

22.1.2.3 PRESCALER

Hay solamente un *prescaler* disponible que es compartido de manera exclusiva entre el *timer* y el *watchdog*. El *prescaler* no es de lectura ni de escritura. El asignar el *prescaler* al temporizador significa que no hay *prescaler* para el *watchdog*, y viceversa. Los bits **PSA** y **PS2:PS0** en el registro **OPTION_REG<3>** y **<2:0>** determinan la asignación y la relación del *prescaler.*

PS2, PS1, PS0	*TMR0 RATE
000	1:2
001	1:4
010	1:8
011	1:16
100	1:32
101	1:64
110	1:128
111	1:256

*If PSA = 1 (Prescaler Assigned to Watchdog Timer), TMR0 Rate = 1:1

Tabla 22.1 Selección de la tasa del *prescaler*.

libros.ep-electropc.com

Cuando el *prescaler* es asignado al temporizador, todas las instrucciones de escritura al registro **TMR0** (p. ej., **CLRF TMR0, MOVWF TMR0, BSF TMR0...**, etc.), aclaran al *prescaler.* Cuando el *prescaler* es asignado al **WDT** (*watchdog timer*), la instrucción **CLRWDT** puede borrar el *prescaler* y el **WDT**.

```
NOTA: Escribiendo el TMR0 cuando el prescaler es asignado al
temporizador, puede borrar el conteo del prescaler, pero no cambia
la asignación.
```

22.1.2.4 INTERRUPCIÓN (INTERRUPT)

La **interrupción** del TMR0 es generada cuando el registro **TMR0** hace un **sobreflujo** (*overflow*) desde **FFh a 00h**; este *overflow* pone a uno el bit **TMR0IF** del registro **INTCON<2>**; la interrupción se puede enmascarar (*masked*) borrrando el bit TMR0IF del registro **INTCON<2>**.

```
NOTA: Enmascarar, cuando se inhabilita una interrupción.
```

El bit TMR0IF debe ser borrado por software en la rutina de servicio de interrupción, antes de rehabilitar esta interrupción. La interrupción del **TMR0** no puede despertar el procesador del modo de *sleep* (**dormido**), si ha sido apagado durante *sleep*.

```
NOTA: El registro TMR0 puede hacer un overflow, ya sea en modo de
temporizador o en modo de contador.
```

22.1.2.5 TIMER0 CON CLOCK EXTERNO

Cuando el *prescaler* no se usa, la entrada del *clock* externo es igual a la salida del *prescaler.* La sincronización de **T0CK1** con la fase del *clock* interno es acoplada por muestrear la salida del *prescaler* con los ciclos **Q2** y **Q4** de las fases del *clock* interno. Por esto es necesario que el *clock* de entrada por el pin **T0CK1** sea al menos 2Tosc en nivel alto (**"1" = 5V**), y al menos **2Tosc** en el nivel bajo (**"0" = 0V**), más un pequeño retardo **RC** de 20 µs en cada nivel. (Remítase a **Timer0 and timer1 external clock requirements**, en el pdf **PIC16F87XA** o *data sheet* **DS39582B**, página 184).

libros.ep-electropc.com

22.1.3 REGISTROS ASOCIADOS

En la Tabla 22.2 se describen cada uno de los bits del registro **OPTION_REG**, y en la Tabla 22.3 están los registros asociados con el módulo **TIMER0**.

Tabla 22.2 Bits del Registro OPTION_REG.

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**

bit 6 **INTEDG**

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: To avoid an unintended device Reset, the instruction sequence shown in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023) must be executed when changing the prescaler assignment from Timer0 to the WDT. This sequence must be followed even if the WDT is disabled.

Tabla 22.3 Registros asociados con el TIMER0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
01h, 101h	TMR0	Timer0 Module Register								xxxx xxxx	uuuu uuuu
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
81h, 181h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented locations read as '0'. Shaded cells are not used by Timer0.

libros.ep-electropc.com

22.1.4 PROCEDIMIENTO DE USO

Como este módulo tiene dos modos de funcionamiento, **modo temporizador** o *timer* y **modo contador** o *counter*, describiremos cada operación con su ejemplo de código asociado; también existen dos métodos que son por *polling* y por interrupción; para cada uno de ellos damos un ejemplo. Veremos cada uno en los casos siguientes:

- TIMER0 modo temporizador (*timer*) por *polling*
- TIMER0 modo temporizador (*timer*) por interrupción
- TIMER0 modo contador (*counter*) por *polling*
- TIMER0 modo contador (*counter*) por interrupción

Para comprender fácilmente el procedimiento de uso explicaremos estos casos al principio de cada práctica.

22.1.5 PRÁCTICA

Haremos las prácticas con cada uno de los casos siguientes:

- TIMER0 modo temporizador (*timer*) por *polling*
- TIMER0 modo temporizador (*timer*) por interrupción
- TIMER0 modo contador (*counter*) por *polling*
- TIMER0 modo contador (*counter*) por interrupción

22.1.5.1 MONTAJE 1

Para las prácticas siguientes se necesita el montaje siguiente:

- Herramientas: depurador PICkit 2, tarjeta de evaluación **PIC16F8XX**, osciloscopio (opcional).

La foto del montaje utilizada en las primeras dos prácticas es tal como se muestra en la Figura 22.4.

libros.ep-electropc.com

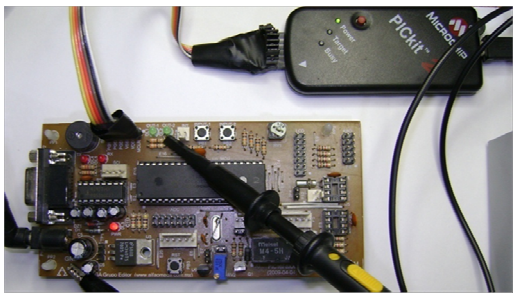


Figura 22.4 Montaje prácticas TIMER0.

Más adelante se muestran las gráficas copiadas del osciloscopio para las dos prácticas siguientes.

22.1.5.1.1 PROCECIMIENTO DE USO TIMER0 MODO TEMPORIZADOR (TIMER POR POLLING)

En este método los pulsos son internos y provienen del **CLKO = Fosc/4**, frecuencia del cristal oscilador/4, y se hace *polling* de la bandera TMR0IF. Los pasos para asignar el **TMR0** como temporizador por *polling* son:

- Borre el registro **TMR0**
- Seleccione la tasa de cambio *prescaler* (*rate prescaler*)
- Asigne la fuente de *clock* CLKO = Fosc/4
- Asigne el *prescaler* al TMR0
- Haga segmento de programa para verificar el *overflow* del Timer0 (*polling*)
- Al ocurrir un *overflow* debe apagar la bandera TMR0IF del registro INTCON

Ahora con el código de cada paso incluido:

1. Borre el registro TMR0:

```
clrf    TMR0            ; borra el registro TMR0
```

2. Seleccione la tasa (*rate*) de cambio *prescaler*:

```
        bsf STATUS,RP0          ; selecciona Banco 1
                                ; asigna prescaler
        movlw   b'00000111'     ; carga valor de prescaler (últimos 3 bits) -> W
        movwf   OPTION_REG      ; guarda valor de prescaler
```

3. Asigne la fuente de clock CLKO = Fosc/4:

```
bcf     OPTION_REG,T0CS ; asigna la fuente de clock al Timer0,
                                ; 0->CLKO = Fosc/4, 1->RA4/T0CKI
```


libros.ep-electropc.com

4. Asigne el *prescaler* al TMR0:

```
    bcf     OPTION_REG,PSA    ; asigna el prescaler al Timer0, Timer0 = 0/WDT = 1
    bcf     STATUS,RP0        ; selecciona Banco 0
```

5. Haga segmento de programa para verificar el *overflow* del Timer0:

```
MainLoop:
; check si hay overflow
    btfss   INTCON,TMR0IF     ; ¿hay overflow? Sí, salte
    goto    MainLoop          ; no, haga ciclo
; código para salida por si hay overflow

                              ; instrucciones usuario
```

6. Al ocurrir un *overflow* debe apagar la bandera TMROIF del registro INTCON:

```
    bcf     INTCON,TMR0IF     ; apague la bandera del overflow
```

En este modo de *polling* se verifica el bit **TMROIF** del registro **INTCON**, cuando pasa a **1**, indica *overflow* o paso del **TMR0** de **FFh** a **00h**. Una vez que se efectúa esto, hay que apagar '0' el bit de la bandera **TMROIF** del registro **INTCON**.

Práctica Timer0 modo temporizador (timer por polling)

El programa siguiente prende y apaga el led **OUT_1**, y el buzzer **BUZ_1** de la tarjeta **PIC1F877A**, con una frecuencia de 38 Hz. Esta frecuencia es generada con el módulo Timer0 en **modo de timer**, procediendo con el método de *polling*. Si tenemos un Xtal = 20 MHz = Fosc, la Fosc/4 = 5 MHz, eso es pulsos cada 0.2 us = Tcy. Se pone el Timer0 con *prescaler* máximo '111' = 256:

Tiempo > Tcy (0.2 us) * Prescaler (256) * max TMR0 (256) = 13.107,2 us = 13.1 ms.

Con cada *overflow* del TMR0 se alterna el valor de las salidas; esto ocurre cada 13.1 ms, o sea el ciclo **ON** y **OFF = 26.2 ms** o **38.1 Hz = 38 Hz**. En las salidas el tono se escucha en el buzzer **BUZ_1,** y el led **OUT_1** titila muy rápido:

```
        list p = 16f877a      ; directiva para definir procesador
#include <p16f877a.inc>       ; definiciones de registros específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define   _toggle_BUZ_1     b'00000100'     ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define   _toggle_OUT_1     b'00010000'     ; filtro para toggle led RB4=_OUT_1=b'0001 0000'
#define   bit_buzzer        .2              ; bit buzzer, puerto B<2>

 org 0
        goto    start

;-----------------------------------------------------------------
```


libros.ep-electropc.com

```
; Inicializa puerto B y módulo Timer0
Ini_Pto
; inicie valores PORTB
            clrf    PORTB               ; apaga el puerto B
            bsf     PORTB,bit_buzzer    ; apaga el buzzer, puerto B<2> = 1

;configura PORTB
            bsf     STATUS,RP0          ; selecciona Banco 1
            movlw   b'11001011'          ;RB7,6,3=entradas ICSP.
                                        ; RB1,0=entradas switches, RB=5,4,2 salidas leds y buzzer
            movwf   TRISB
            bcf     STATUS,RP0          ; selecciona Banco 0

            return
;-----------------------------------------------------------------
Ini_TMR0
; configura TMR0
            clrf    TMR0                ; borra el registro TMR0
;           clrf    INTCON              ; deshabilite interrupciones y borra T0IF

            bsf     STATUS,RP0          ; selecciona Banco 1
; asigna prescaler
            movlw   b'00000111'         ; carga valor de prescaler (últimos 3 bits) -> W
            movwf   OPTION_REG          ; guarda valor de prescaler

            bcf     OPTION_REG,T0CS     ; asigna la fuente de clock al Timer0,
                                        ; RA4/T0CKI = 1 y CLK0 = Fosc/4 = 0
            bcf     OPTION_REG,PSA      ; asigna el prescaler al Timer0, Timer0 = 0/WDT = 1
            bcf     STATUS,RP0          ; selecciona Banco 0

; configura interrupción TMR0

            return
;-----------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
 bcf STATUS,RP0 ; selecciona Banco 0
            movf    PORTB,W             ; carga PORTB -> W
            xorlw   _toggle_OUT_1       ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1       ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB               ; pone W -> PORTB

            return
;-----------------------------------------------------------------

Start:
            call    Ini_Pto             ; inicialice puertos
            call    Ini_TMR0            ; inicialice TMR0

MainLoop:
; check si hay overflow
            btfss   INTCON,TMR0IF       ; ¿hay overflow? Sí, salte
            goto    MainLoop            ; no, haga ciclo

            call    Toggle_pto          ; toggle (alternar) puerto
```


libros.ep-electropc.com

```
        bcf     INTCON,TMR0IF     ; apaga la bandera del overflow

        goto MainLoop             ; haga ciclo
        end
```

Este ejemplo está en nuestra página Web en la carpeta X:\ Ejemplos_ASM\ y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\ TIMER0_1\ TIMER0_1.asm.

a) b)

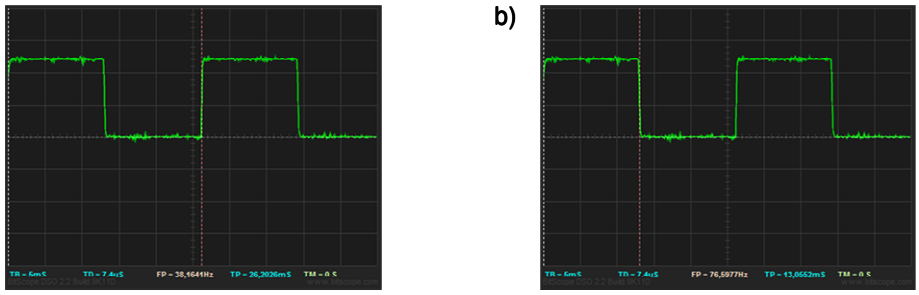


Figura 22.5 Gráficas de 38 Hz.

La Figura 22.5 a, b, c muestra la medición de las señales en el led **OUT_1** y el buzzer BUZ_1; el periodo de la señal es de 26 ms aproximado a la frecuencia de 38 Hz.

22.1.5.1.2 PROCEDIMIENTO DE USO TIMER0 MODO TEMPORIZADOR (TIMER POR INTERRUPCIÓN)

En este método los pulsos internos provienen del CLKO = Fosc/4, frecuencia del cristal oscilador, se hace interrupción con la bandera TMR0IF, habilitando el bit TMR0IE de la interrupción del módulo TMR0. Para asignar el TMR0 como temporizador por interrupción, siga los pasos siguientes:

1. Borre TMR0
2. Seleccione la tasa de cambio de *prescaler*
3. Asigne el *prescaler*
4. Asigne la fuente de *clock* CLKO = Fosc/4
5. Habilite la interrupción del Timer0, bit TMR0IE del registro INTCON
6. Habilite la interrupción global, bit GIE del registro INTCON
7. Haga la rutina de interrupción del TMR0

libros.ep-electropc.com

8. Al final de la rutina de interrupción del TMR0, debe apagar la bandera TMR0IF del registro INTCON
9. Cierre la rutina de interrupción con la instrucción “retfie”

Ahora, realice los mismos pasos con código:

1. Borre TMR0:

```
    clrf    TMR0            ; borra el registro TMR0
```

2. Seleccione la tasa de cambio de *prescaler*:

```
    bsf     STATUS,RP0      ; selecciona Banco 1
                            ; asigna prescaler
    movlw   b'00000111'     ; carga valor de prescaler (últimos 3 bits) -> W
    movwf   OPTION_REG      ; guarda valor de prescaler
```

3. Asigne el *prescaler*:

```
    bcf     OPTION_REG,PSA  ; asigna el prescaler al Timer0, Timer0 = 0/WDT = 1
```

4. Asigne la fuente de *clock* CLKO = Fosc/4:

```
    bcf     OPTION_REG,T0CS ; asigna la fuente de clock al TMR0
```

5. Habilite la interrupción del Timer0, bit TMR0IE del registro INTCON:

```
    bsf     INTCON,TMR0IE   ; habilita interrupción del Timer0
```

6. Habilite la interrupción global, bit GIE del registro INTCON:

```
    bsf     INTCON,GIE      ; habilita interrupciones globales
```

7. Haga rutina de interrupción del TMR0:

```
    org 0004h
ISR_TMR0
; aquí código de la rutina de interrupción
```

8. Al final de la rutina de interrupción del TMR0 debe apagar la bandera TMR0IF del registro INTCON:

```
; apague la bandera del overflow
    bcf     INTCON,TMR0IF   ; apague la bandera del overflow
```

9. Cierre la rutina de interrupción con la instrucción “retfie”:

```
    retfie
```

En este modo de interrupción, cada vez que ocurre un *overflow* del Timer0 (cuando el TMR0 pasa de FFh a 00h) el bit de la bandera TMR0IF pasa a ‘1’, se genera la interrupción, el contador de programa pasa a la posición 004h donde se ejecuta la rutina de interrupción ISR_Timer0. Al terminar se apaga el bit de la bandera TMR0IF y se coloca la instrucción “retfie” para terminar.

libros.ep-electropc.com

Práctica Timer0 modo temporizador (timer por interrupción)

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F877A** con una frecuencia de **38 Hz**. Esta frecuencia es generada con el módulo Timer0, procediendo con el mètodo de interrupción. Este programa, al igual que el anterior, genera las mismas salidas en la misma frecuencia; la diferencia radica en el método de interrupción, donde el código de la rutina de interrupción contiene el encendido y el apagado del led **OUT_1** y el buzzer **BUZ_1**.

```
            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de registros específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1       b'00000100'         ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1       b'00010000'         ; filtro para toggle led RB4=_OUT_1=b'0001 0000'
#define    bit_buzzer          .2                  ; bit buzzer, puerto B<2>

            org 000h                        ; vector de reset
            goto        start

            org 0004h                       ; vector de interrupción
;-------------------------------------------------------------------------
ISR_TMR0
            call        Toggle_pto          ; toggle(alternar) puerto

; apague la bandera del overflow
            bcf         INTCON,TMR0IF       ; apague la bandera del overflow

            retfie
;-------------------------------------------------------------------------
; inicialice puerto B y módulo Timer0
Ini_Pto
; inicie valores PORTB
            clrf        PORTB               ; apaga el puerto B
            bsf         PORTB,bit_buzzer    ; apaga el buzzer, puerto B<2> = 1

;configura PORTB
            bsf         STATUS,RP0          ; selecciona Banco 1
            movlw       b'11001011'         ;RB7,6,3 = entradas ICSP
                                            ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf       TRISB
            bcf         STATUS,RP0          ; selecciona Banco 0

            return
;-------------------------------------------------------------------------
Ini_TMR0
; configura TMR0
            clrf        TMR0                ; borra el registro TMR0
;           clrf        INTCON              ; deshabilite interrupciones y borra T0IF

            bsf         STATUS,RP0          ; selecciona Banco 1
```


libros.ep-electropc.com

```
; asigna prescaler
            movlw   b'00000111'         ; carga valor de prescaler (últimos 3 bits) -> W
            movwf   OPTION_REG          ; guarda valor de prescaler

            bcf     OPTION_REG,T0CS     ; asigna la fuente de clock al Timer0,
                                        ; RA4/T0CKI = 1 y CLKO = Fosc/4 = 0
            bcf     OPTION_REG,PSA      ; asigna el prescaler al Timer0, Timer0 = 0/WDT = 1
            bcf     STATUS,RP0          ; selecciona Banco 0

; configura Interrupción TMR0
            bsf     INTCON,TMR0IE               ; habilita interrupción del Timer0
            bsf     INTCON,GIE                  ; habilita interrupciones globales

            return
;-----------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
            movf    PORTB,W             ; carga PORTB -> W
            xorlw   _toggle_OUT_1       ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1       ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB               ; pone W -> PORTB

            return
;-----------------------------------------------------------------
Start:
            call    Ini_Pto             ; inicialice puertos
            call    Ini_TMR0            ; inicialice TMR0

MainLoop:
            goto    $                   ; haga ciclo

            end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\ TIMER0_1\ TIMER0_1A.asm.

a) b)

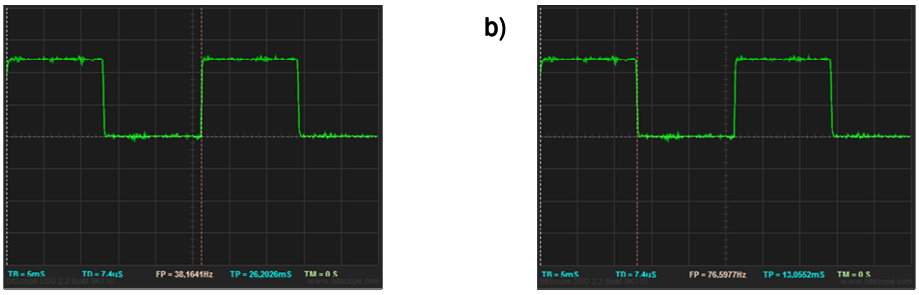


Figura 22.6 Gráficas de 38 Hz.

La Figura 22.6a, b muestra la medición de las señales en el led **OUT_1** y el buzzer **BUZ_1**, el periodo de la señal es de 26 ms aproximado a la frecuencia de **38 Hz**.

libros.ep-electropc.com

22.1.5.2 MONTAJE 2

Para las prácticas siguientes se necesita el montaje siguiente:

- Herramientas: depurador PICkit 2, tarjeta de evaluación **PIC16F8XX**, osciloscopio (opcional).
- Circuito cruce por cero optocoplado con MCT6, con pulsos a cero.

Este montaje es para estas dos últimas prácticas. Las fotos del montaje son las siguientes: el esquema del circuito cruce por cero optocoplado y su montaje. El circuito de cruce por cero se utiliza en estas últimas dos prácticas (Figura 22.7).

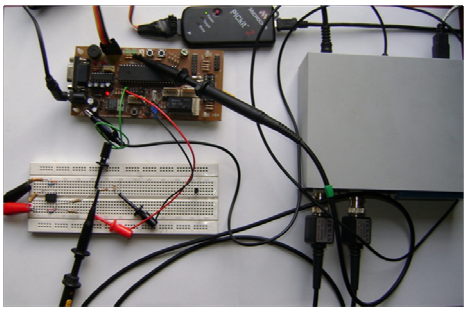


Figura 22.7 Montaje Prácticas TIMER0.

En la Figura 22.8 se observan todas las partes del montaje, incluso el circuito de cruce por cero que se implementó en protoboard.

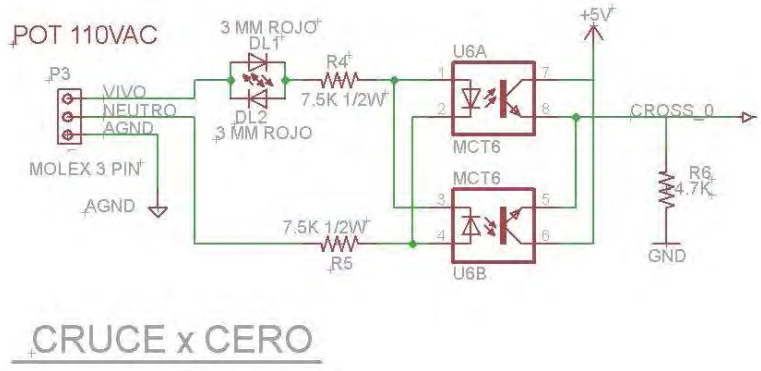


Figura 22.8 Esquema circuito cruce por cero optocoplado.

libros.ep-electropc.com

En el esquema de la Figura 22.9 se muestra el circuito de cruce por cero; el montaje en protoboard se ve en la Figura 22.10; contiene la entrada de 110VAC con caimanes a la izquierda, y los puntos de conexión a la tarjeta con *test point*.

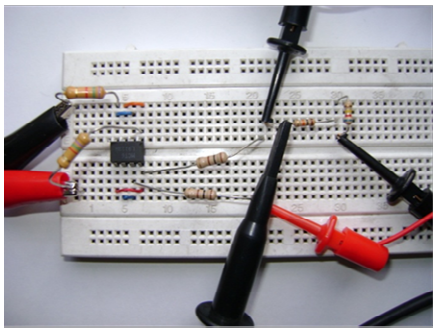


Figura 22.9 Montaje circuito cruce por cero optocoplado.

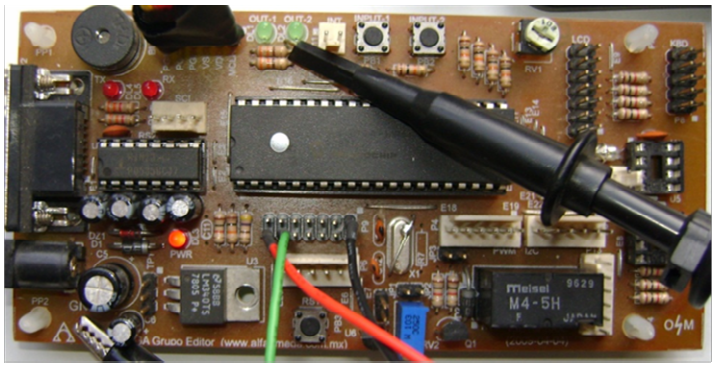


Figura 22.10 Conexión a tarjeta PIC16F8XX.

En la Figura 22.10 se muestran los puntos de prueba conectados a la tarjeta en el conector **P9**, pin1 a **GND** con cable negro, pin **12 a RA4/T0CKI** con cable verde, pin **14 a +5V** con cable rojo. Más adelante se muestran las gráficas copiadas del osciloscopio para las dos prácticas siguientes.

22.1.5.2.1 PROCEDIMIENTO DE USO TIMER0 MODO CONTADOR (COUNTER POR POLLING)

En este método los pulsos externos provienen del pin **RA4/T0CKI**, y se hace *polling* de la bandera **TMR0IF**. Para asignar el TMR0 como contador por *polling*, siga los pasos siguientes:

libros.ep-electropc.com

- Borre el registro TMR0
- Seleccione la tasa (*rate*) de cambio *prescaler*
- Asigne la fuente de *clock* pin RA4/T0CKI
- Asigne el *prescaler* al TMR0
- Haga segmento de programa para verificar el *overflow* del Timer0 (*polling*)
- Al ocurrir un *overflow* debe apagar la bandera TMR0IF del registro INTCON

Ahora, con el código de cada paso incluido:

- Borre el registro TMR0:

```
clrf    TMR0            ; borra el registro TMR0
```

- Seleccione la **tasa** (*rate*) de cambio *prescaler*:

```
    bsf     STATUS,RP0      ; selecciona Banco 1
                            ; asigna prescaler
    movlw   b'00000111'     ; carga valor de prescaler (últimos 3 bits) -> W
    movwf   OPTION_REG      ; guarda valor de prescaler
```

- Asigne la fuente de *clock* pin RA4/T0CKI:

```
bsf     OPTION_REG,T0CS ; asigna la fuente de clock al Timer0,
                        ; RA4/T0CKI = 1 y CLKO = Fosc/4 = 0
```

- Asigne el *prescaler* al TMR0:

```
    bcf     OPTION_REG,PSA  ; asigna el prescaler al Timer0, Timer0 = 0/WDT = 1
    bcf     STATUS,RP0      ; selecciona Banco 0
```

- Haga el segmento de programa para verificar el *overflow* del Timer0 (*polling*):

```
MainLoop:
; check si hay overflow
    btfss   INTCON,TMR0IF   ; ¿hay overflow? Sí, salte
    goto    MainLoop        ; no, haga ciclo
; código para salida por si hay overflow
                            ; instrucciones usuario
```

- Al ocurrir un *overflow* debe apagar la bandera TMR0IF del registro INTCON:

```
    bcf     INTCON,TMR0IF   ; apague la bandera del overflow
```

En este modo de *polling* se verifica el bit **TMR0IF** del registro **INTCON**, cuando pasa a '1', lo que indica un *overflow* o un paso del **TMR0** de **FFh** a **00h**. Una vez que se realiza esto, hay que apagar '0', el bit de la bandera **TMR0IF** del registro **INTCON**.

libros.ep-electropc.com

Práctica Timer0 modo contador (counter por polling)

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F877A**, con una frecuencia de **½ Hz**. La fuente externa conectada al pin **RA4/T0CKI** proviene de un circuito optocoplado de cruce por 0, tomado de la línea de **120VAC / 60 Hz**, que genera **120 Hz**. La frecuencia de **½ Hz** resulta de contar estos **120 pulsos**. Esta frecuencia de **½ Hz** es generada con el módulo **Timer0**, en el modo de **contador** o *counter*, procediendo con el mètodo de *polling*. Al estar asignado el *prescaler* al **WDT**, la relación de los pulsos de entrada al **TMR0** es de **1:1**, o sea que cuenta **120 pulsos** para generar un *overflow*.

Con cada 120 pulsos se desea generar un *overflow* del TMR0; esto implica cargar el TMR0 con un valor inicial de (256-120 = 136) de 136, para que al contar 120 pulsos el TMR0 genere un *overflow* en el pulso número 120 que corresponde al paso de **FFh** a **00h**. Cada un *overflow* equivale a 1 Hz; esto ocurre cada 1 segundo, o sea el ciclo **ON y OFF = 2s** o **½ Hz**. En las salidas, el tono se escucha en el buzzer **BUZ_1** y se ve en el led **OUT_1**, a la frecuencia de **½ Hz**.

```
            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1       b'00000100'         ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1       b'00010000'         ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    bit_buzzer          .2                  ; bit buzzer, puerto B<2>
#define    _136_ciclos         .136                ; 136 ciclos para 1 seg
                                                   ; (lo que falta para overflow 256-120 = 136)

            org 0
            goto        start

;-----------------------------------------------------------------------
; Inicialice puerto B y módulo Timer0
Ini_Pto
; inicie valores PORTB
            clrf        PORTB               ; apaga el puerto B
            bsf         PORTB,bit_buzzer    ; apaga el buzzer, puerto B<2> = 1

;configura PORTB
            bsf         STATUS,RP0          ; selecciona Banco 1
            movlw       b'11001011'         ;RB7,6,3 = entradas ICSP,
                                            ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf       TRISB
            bcf STATUS,RP0 ; selecciona Banco 0

            return
;-----------------------------------------------------------------------
Ini_TMR0
; configura TMR0
            clrf        TMR0                ; borra el registro TMR0
```


libros.ep-electropc.com

```
;           clrf    INTCON          ; deshabilita interrupciones y borra T0IF

            bsf STATUS,RP0 ; selecciona Banco 1
; asigna prescaler
            movlw   b'00000000'     ; carga valor de prescaler (últimos 3 bits) -> W
            movwf   OPTION_REG      ; guarda valor de prescaler

; configura TMR0
            bsf     OPTION_REG,T0CS ; asigna la fuente de clock al TMR0
            bsf     OPTION_REG,PSA  ; asigna el prescaler al WDT
            bcf     STATUS,RP0      ; selecciona Banco 0

; configura Interrupción TMR0

            return
;-------------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
            movf    PORTB,W         ; carga PORTB -> W
            xorlw   _toggle_OUT_1   ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1   ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB           ; pone W -> PORTB

            return
;-------------------------------------------------------------------

Start:
            call    Ini_Pto         ; inicialice puertos
            call    Ini_TMR0        ; inicialice TMR0

Main_ini_TMR0
            movlw   _136_ciclos     ; carga 136 ciclos
            movwf   TMR0            ; guarda en TMR0

MainLoop:
; check si hay overflow
            btfss   INTCON,TMR0IF   ; ¿hay overflow? Sí, salte
            goto    MainLoop        ; no, haga ciclo

            call    Toggle_pto      ; toggle puerto
            bcf     INTCON,TMR0IF   ; apague la bandera del overflow

            goto    Main_ini_TMR0   ; haga ciclo

            end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\TIMER0_1\TIMER0_1B.asm.

libros.ep-electropc.com

a) b)

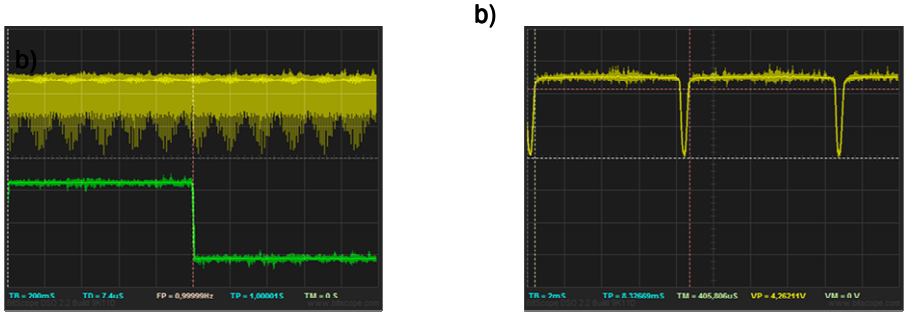


Figura22.11 Gráficas de cruce por cero.

La Figura 22.11 a, b muestra la medición de la señal de salida del buzzer y led de **1** segundo **Ton = Toff** y **2** segundos el periodo **T** (izquierda); también la señal de cruce por cero (derecha), que es de **120 Hz**, un pulso de estos mide 8.33 ms, y dos pulsos 16.66 ms.

22.1.5.2.2 PROCEDIMIENTO DE USO TIMER0 MODO CONTADOR (COUNTER) POR INTERRUPCIÓN

En este método los pulsos externos provienen del pin **RA4/T0CKI**, se hace interrupción con la bandera **TMR0IF**, habilitando el bit TMR0IE de la interrupción del módulo **TMR0**. Para asignar el **TMR0** como contador por interrupción siga los pasos siguientes:

1. Borre TMR0
2. Seleccione la tasa (*rate*) de cambio de *prescaler*
3. Asigne *prescaler*
4. Asigne la fuente de *clock* pin RA4/T0CKI
5. Habilite la interrupción del Timer0, bit TMR0IE del registro INTCON
6. Habilite la **interrupción global**, bit GIE del registro INTCON
7. Haga rutina de interrupción del TMR0
8. Al final de la rutina de interrupción del TMR0 debe apagar la bandera TMR0IF del registro INTCON
9. Cierre la rutina de interrupción con la instrucción “retfie”

libros.ep-electropc.com

Los mismos pasos con código son:

1. Borre TMR0:

```
    clrf        TMR0            ; borra el registro TMR0
```

2. Seleccione tasa de cambio de *prescaler*

```
    bsf         STATUS,RP0      ; selecciona Banco 1
; asigne prescaler
    movlw       b'00000111'     ; carga valor de prescaler (últimos 3 bits) -> W
    movwf       OPTION_REG      ; guarda valor de prescaler
```

3. Asigne *prescaler*:

```
    bsf         OPTION_REG,PSA  ; asigna el prescaler al Timer0, Timer0 = 0/WDT = 1
```

4. Asigne la fuente de *clock* pin RA4/T0CKI:

```
    bsf         OPTION_REG,T0CS ; asigna la fuente de clock al Timer0,
                                ; RA4/T0CKI = 1 y CLKO = Fosc/4 = 0
```

5. Habilite la interrupción del Timer0, bit TMR0IE del registro INTCON:

```
    bsf         INTCON,TMR0IE   ; habilita interrupción del Timer0
```

6. Habilite la interrupción global, bit GIE del registro INTCON:

```
    bsf         INTCON,GIE      ; habilita interrupciones globales
```

7. Haga la rutina de interrupción del TMR0:

```
    org 0004h
ISR_TMR0
; aquí código de la rutina de interrupción
```

8. Al final de la rutina de interrupción del TMR0 debe apagar la bandera TMR0IF del registro INTCON:

```
;apague la bandera del overflow
    bcf         INTCON,TMR0IF   ; apague la bandera del overflow
```

9. Cierre la rutina de interrupción con la instrucción “retfie”:

```
    retfie
```

En este modo de interrupción, cada vez que ocurre un *overflow* del **Timer0** (cuando **TMR0** pasa de **FFh** a **00h**), el bit de la bandera **TMR0IF** pasa a **‘1’**, se genera la interrupción, el contador de programa pasa a la posición **004h** donde se ejecuta la rutina de interrupción **ISR_Timer0**. Al terminar se apaga el bit de la bandera **TMR0IF**, y se coloca la instrucción “retfie” para terminar.

Práctica Timer0 modo contador (counter) por interrupción

El programa siguiente, al igual que el anterior, prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F877A** con una frecuencia de **½ Hz**. Esta frecuencia es generada con el módulo Timer0, procediendo con el mètodo de interrupción. Lo

libros.ep-electropc.com

mismo que en el programa anterior, cada *overflow* equivale a 1 Hz; esto ocurre cada **1** segundo, o sea que el ciclo **ON y OFF = 2s** o **½ Hz**. En las salidas, el tono se escucha en el buzzer **BUZ_1**, y se ve en el led **OUT_1** a una frecuencia de **½ Hz**. La diferencia está en que en este programa el método usado es por interrupción.

```
            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1       b'00000100'         ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1       b'00010000'         ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    bit_buzzer          .2                  ; bit buzzer, puerto B<2>
#define    _136_ciclos         .136                ; 136 ciclos para 1 seg
                                                   ; (lo que falta para overflow 256-120 = 136)

            org 000h            ; vector de reset
            goto    start

            org 0004h           ; vector de interrupción
;-------------------------------------------------------------------------
ISR_TMR0
            call    Toggle_pto          ; toggle puerto

;apague la bandera del overflow
            movlw   _136_ciclos         ; carga 136 ciclos
            movwf   TMR0                ; guarda en TMR0

            bcf     INTCON,TMR0IF       ; apague la bandera del overflow

            retfie
;-------------------------------------------------------------------------
; inicialice puerto B y módulo Timer0
Ini_Pto
; inicie valores PORTB
            clrf    PORTB               ; apaga el puerto B
            bsf     PORTB,bit_buzzer    ; apaga el buzzer, puerto B<2> = 1

;configura PORTB
            bsf     STATUS,RP0          ; selecciona Banco 1
            movlw   b'11001011'         ;RB7,6,3 = entradas ICSP
                                        ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf   TRISB
            bcf     STATUS,RP0          ; selecciona Banco 0

            return
;-------------------------------------------------------------------------
Ini_TMR0
; configura TMR0
            clrf    TMR0                ; borra el registro TMR0
;           clrf    INTCON              ; deshabilite interrupciones y borre T0IF

            bsf     STATUS,RP0          ; selecciona Banco 1
; asigna prescaler
```


libros.ep-electropc.com

```
            movlw   b'00000000'     ; carga valor de prescaler (últimos 3 bits) -> W
            movwf   OPTION_REG      ; guarda valor de prescaler

; configura TMR0
            bsf     OPTION_REG,T0CS ; asigna la fuente de clock al TMR0
            bsf     OPTION_REG,PSA  ; asigna el prescaler al WDT
            bcf     STATUS,RP0      ; selecciona Banco 0

; configura Interrupción TMR0
            bsf     INTCON,TMR0IE   ; habilita interrupción del Timer0
            bsf     INTCON,GIE      ; habilita interrupciones globales

            return
;-----------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
            movf    PORTB,W         ; carga PORTB -> W
            xorlw   _toggle_OUT_1   ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1   ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB           ; pone W -> PORTB

            return
;-----------------------------------------------------------------
Start:
            call    Ini_Pto         ; inicialice puertos
            call    Ini_TMR0        ; inicialice TMR0

Main_ini_TMR0
            movlw   _136_ciclos     ; carga 136 ciclos
            movwf   TMR0            ; guarda en TMR0

            goto    $               ; no, haga ciclo

            end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\TIMER0_1\ TIMER0_1C.asm.

a) b)

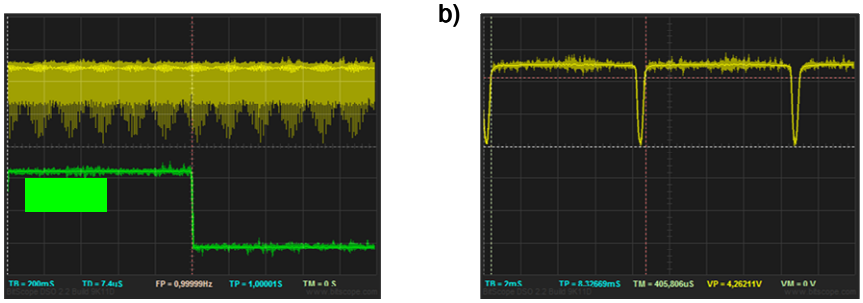


Figura 22.12 Gráficas de cruce por cero.

libros.ep-electropc.com

En la Figura 22.12 a, b se muestra la medición de la señal de salida del buzzer y led de **1** segundo **Ton = Toff** y **2** segundos el periodo **T** (izquierda); también la señal de cruce por cero (derecha), que es de **120 Hz**; un pulso de éstos mide 8.33 ms, y dos pulsos 16.66 ms.

22.2 TIMER1

Este módulo **TIMER1** puede funcionar como **temporizador** (*timer*) o como **contador** (*counter*); tiene un registro de 16 bits compuesto por dos registros de 8 bits (**TMR1H:TMR1L**) y un *prescaler* de 4 bits que divide el tiempo o el conteo. La fuente de clock interno es CLK0 = Fosc/4, o podemos insertar pulsos externos de *clock* para este módulo por el pin **RC0/T1CKI**, o conectar un cristal entre los pines **RC0/T1OSO** y **RC1/T1OSI**.

22.2.1 DIAGRAMA CIRCUITAL

En la Figura 22.13 apreciamos el diagrama de bloques del módulo **TIMER1**.

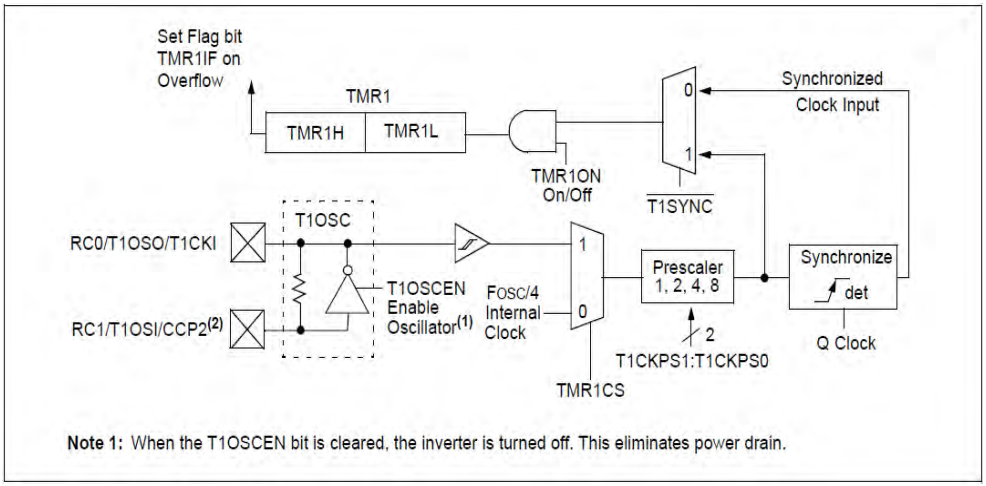


Figura 22.13 Diagrama de bloques del módulo TIMER1.

22.2.2 FUNCIONAMIENTO

El módulo TIMER1 tiene dos modos de operación: como **temporizador** (*timer*) o como **contador** (*counter*), y tiene 16 bits de resolución compuesto por dos registros de 8 bits (**TMR1H:TMR1L**) que son de lectura y **escritura**. Estos registros se incrementan desde **0000h** a **FFFFh** y pasa a **0000h**; en este paso causa un

libros.ep-electropc.com

sobreflujo (*overflow*), ésta es "latcheada" en el bit de la bandera de interrupción **TMR1IF** en el registro **PIR<0>**. La interrupción **TMR1** puede ser habilitada o deshabilitada por setear (poner a "1") o aclarar (poner en "0") el bit de habilitación de interrupción **TMR1IE**, en el registro **PIE1<0>**.

El modo de operación temporizador o contador es determinado por seleccionar el bit **TMR1CS** en el registro **T1CON<1>**. En el modo de temporizador, el **TIMER1** incrementa cada ciclo de instrucción. En el modo de contador, el TIMER1 incrementa con cada flanco de subida (*rising*) de la entrada del *clock* externo. El módulo **TIMER1** puede ser habilitado o deshabilitado por setear o aclarar el bit de control **TMR1ON** del registro **T1CON<0>**.

El módulo **TIMER1** tiene una entrada de *reset* interna; este *reset* puede ser generado por cualquiera de los dos módulos CCP (*véase* el **Cap. 23.0: Captura/comparación/PWM [CCP]**). Cuando el oscilador del **Timer1** es habilitado seteando el bit **T1OSCEN** en el registro **T1CON<3>**, los pines **RC1/T1OSI/CCP2** y **RC0/T1OSO/T1CKI** son pines de entrada; esto significa que el **TRISC<1:0>** es ignorado y estos pines son leídos como **"0"**.

```
NOTA: En el archivo pdf está disponible información adicional sobre
los módulos Timers.
(PICmicro® Mid-Range MCU Family Reference Manual [DS33023]).
```

22.2.2.1 MODO TEMPORIZADOR (TIMER)

El **modo de temporizador** (*timer*) es seleccionado por aclarar el bit **TMR1CS** en el registro **T1CON<1>**. En este modo la entrada de *clock* al *timer* proviene de **Fosc/4**. El bit de sincronización, en el registro **T1CON<2>**, no tiene efecto porque el *clock* interno **Fosc/4** está siempre sincronizado.

22.2.2.2 MODO CONTADOR (COUNTER)

El modo de contador se selecciona al poner a "1" el bit **TMR1CS** en el registro **T1CON<1>**, y puede operar en modo **sincrónico** (síncrono) o **asincrónico** (asíncrono). Cuando el **TIMER1** está siendo incrementado a través de una fuente externa, los incrementos ocurren por un **flanco de subida** (*rising*). Después que el **TIMER1** es habilitado para modo de contador, el módulo debe recibir primero un **flanco de caída** (*falling*), antes que el contador comience el incremento (Figura 22.14).

libros.ep-electropc.com

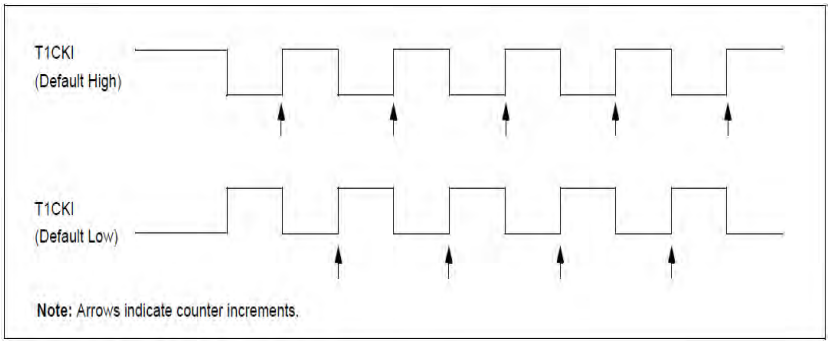


Figura 22.14 TIMER1 incrementado a través de una fuente externa.

22.2.2.2.1 MODO CONTADOR SINCRÓNICO

En este modo el contador incrementa cada flanco de subida (rising) del clock de entrada sobre:

1. El pin RC1/T1OSI/CCP2 cuando seteamos el bit T1OSCEN en el registro T1CON<3>.
2. El pin RC0/T1OSO/T1CKI cuando aclaramos el bit T1OSCEN en el registro T1CON<3>.

Si el bit $\overline{\text{T1SYNC}}$ en el registro **T1CON<2>** es aclarado, la sincronización de la entrada del clock externo se realiza con las fases del clock interno, la sincronización se realiza luego del prescaler que es un contador asincrónico. En esta configuración, durante el modo de sleep, el Timer1 no podrá incrementarse incluso si el clock externo está presente, desde el circuito de sincronización apagado. Sin embargo, el prescaler puede continuar incrementándose.

22.2.2.2.2 MODO CONTADOR ASINCRÓNICO

Si el bit, en el registro T1CON<2>, es seteado, la entrada del clock externo no es sincronizada. El timer continúa incrementándose asincrónico con las fases del clock interno. El timer puede continuar corriendo durante el modo de sleep, y puede generar una interrupción por un overflow, la cual puede despertar el procesador. No obstante, se necesitan precauciones especiales para leer o escribir el timer. En el modo de contador asincrónico, el Timer1 no puede usarse como una base de tiempo para operaciones de comparación o de captura.

libros.ep-electropc.com

22.2.2.2.3 MODO CONTADOR ASINCRÓNICO, LECTURA Y ESCRITURA DEL TIMER1

Leyendo el TMR1H o TMR1L, mientras el timer está corriendo desde un clock externo asincrónico, puede asegurar una lectura válida, pero el usuario debe tener en mente que leer el timer de 16 bits, en sus dos valores de 8 bits, tiene ciertos problemas, porque puede haber un overflow entre lecturas. Por ejemplo, cuando el contador del Timer1 tiene el valor de FFFFh, al leer el valor superior TMR1H = FFh y guardarlo, se pudo incrementar el contador del Timer1 a 0000h, y al leer el valor inferior TMR1L = 00, lo que nos dará una lectura errada de los valores reales, pues se pretendía leer el contador en el valor FFFFh y se leyó FF00h. Para las escrituras, se recomienda que el usuario simplemente detenga el timer y escriba el valor deseado. Una contención de la escritura puede ocurrir por escribir los registros del Timer mientras está corriendo; esto puede producir un valor impredecible en el registro del Timer.

22.2.2.3 OSCILADOR

Un circuito oscilador de cristal es incorporado en los pines entre **RC1/T1OSI** (entrada) y **RC0/T1OSO** (salida del amplificador). Es habilitado por setear el bit de control **T1OSCEN** en el registro **T1CON<3>**. El oscilador es de baja frecuencia, nominal de hasta **200 kHz**. El oscilador puede correr durante el modo de *sleep* y principalmente para usarse con cristal de **32 kHz (32 768 Hz)**. En la Tabla 22.4 se muestra la capacitancia para el oscilador timer. El oscilador **Timer1** es idéntico al oscilador **LP**, el usuario debe proveer un retardo apropiado para el encendido *start-up*.

Tabla 22.4 Capacitancia para el oscilador Timer.

Osc Type	Freq.	C1	C2
LP	32 kHz	33 pF	33 pF
	100 kHz	15 pF	15 pF
	200 kHz	15 pF	15 pF
These values are for design guidance only.			
Crystals Tested:			
32.768 kHz	Epson C-001R32.768K-A	± 20 PPM	
100 kHz	Epson C-2 100.00 KC-P	± 20 PPM	
200 kHz	STD XTL 200.000 kHz	± 20 PPM	

Note 1: Higher capacitance increases the stability of oscillator but also increases the start-up time.

2: Since each resonator/crystal has its own characteristics, the user should consult the resonator/crystal manufacturer for appropriate values of external components.

libros.ep-electropc.com

22.2.2.4 RESETEO DEL TIMER1, PAR DE REGISTROS (TMR1H:TMR1L)

El par de registros TMR1H y TMR1L no son puestos a 00h durante un POR (Power-On-Reset), o cualquier reset, exceptuando eventos de Trigger de los módulos CCP1 y CCP2. El registro T1CON es puesto a 00h sobre un POR o una caída de voltaje brown-out reset, el cual apaga el timer y pone en relación de 1:1 el prescaler. En todos los otros resets los registros no son afectados.

22.2.2.5 PRESCALER

El prescaler es de 4 bits y no es de lectura ni escritura, porque no se tiene acceso a él; tiene una relación de 1,2,4,8 que se selecciona con los bits T1CKPS1:T1CKPS0 en el registro T1CON<5:4>; no es aclarado sobre una escritura de los registros TMR1H o TMR1L.

22.2.3 REGISTROS ASOCIADOS

En la Tabla 22.5 se describen cada uno de los bits del registro **T1CON**, y en la Tabla 22.6 están los registros asociados con el módulo **TIMER1**.

Tabla 22.5 Bits del registro T1CON.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits
11 = 1:8 prescale value
10 = 1:4 prescale value
01 = 1:2 prescale value
00 = 1:1 prescale value

bit 3 **T1OSCEN**: Timer1 Oscillator Enable Control bit
1 = Oscillator is enabled
0 = Oscillator is shut-off (the oscillator inverter is turned off to eliminate power drain)

bit 2 **$\overline{\text{T1SYNC}}$**: Timer1 External Clock Input Synchronization Control bit
When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input
When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS**: Timer1 Clock Source Select bit
1 = External clock from pin RC0/T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON**: Timer1 On bit
1 = Enables Timer1
0 = Stops Timer1

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 22.6 Registros asociados con el módulo TIMER1.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.
Note 1: Bits PSPIE and PSPIF are reserved on the 28-pin devices; always maintain these bits clear.

22.2.4 PROCEDIMIENTO DE USO

Como este módulo tiene dos modos de funcionamiento (**modo temporizador** o *timer* y **modo contador** o *counter*), describiremos cada operación con su ejemplo de código asociado para cada uno de los dos modos, también en sus dos métodos de *polling* y de interrupciones. Veremos cada uno en los casos siguientes:

- TIMER1 Modo Temporizador *timer* por *polling*
- TIMER1 Modo Temporizador (*timer* por interrupción)

En seguida se describe cada procedimiento de uso asociado con cada práctica.

22.2.5 PRÁCTICA

Haremos las prácticas con cada uno de los casos siguientes:

- TIMER1 modo temporizador (*timer* por *polling*)
- TIMER1 modo temporizador (*timer* por interrupción)

22.2.5.1 MONTAJE 1

Para las prácticas siguientes se necesita el montaje siguiente:

- Herramientas: depurador PICkit 2, tarjeta de evaluación **PIC16f8xx**, osciloscopio (opcional).

La foto del montaje utilizada en las primeras dos prácticas es tal como la que se muestra en la Figura 22.15.

libros.ep-electropc.com

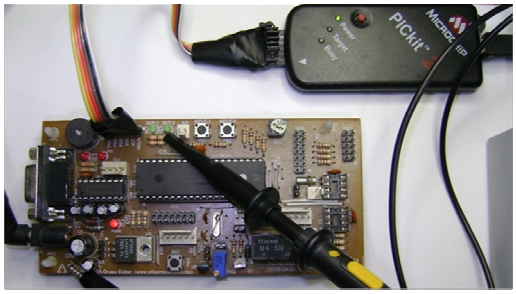


Figura 22.15 Montaje prácticas TIMER1.

Más adelante se muestran las gráficas copiadas del osciloscopio para las dos prácticas siguientes.

22.2.5.1.1 PROCEDIMIENTO DE USO TIMER1 MODO TEMPORIZADOR (TIMER POR POLLING)

En este método los pulsos internos provienen del **CLKO = Fosc/4**, frecuencia del cristal oscilador, y se hace *polling* de la bandera **TMR1IF**. Para asignar el **TMR1** como temporizador por *polling*, siga los pasos siguientes:

1. Borre T1CON
2. Seleccione la tasa (*rate*) de cambio *prescaler*
3. Asigne la fuente de *clock* CLKO = Fosc/4
4. Habilite o prenda el módulo TIMER1
5. Haga segmento de programa para verificar *overflow* del Timer1 (*polling*)
6. Al ocurrir un *overflow* debe apagar la bandera TMR1IF del registro INTCON

Estos mismos pasos con código son:

1. Borre T1CON:

```
clrf    T1CON              ; borra T1CON
```

2. Seleccione la tasa (*rate*) de cambio *prescaler*:

```
iorlw   b'00110000'        ; carga valor de prescaler (bit 5:4) -> W
movwf   T1CON              ; guarda valor de prescaler
```

3. Asigne la fuente de *clock* CLKO = Fosc/4:

```
bcf     T1CON,TMR1CS       ; asigna la fuente de clock al TMR1
                           ; 1 = RC0 externo, 0 = Fosc/4 interno
```

4. Habilite o prenda el módulo TIMER1:

```
bsf     T1CON,TMR1ON       ; habilite o prenda TIMER1, 1 = ON, 0 = OFF
```


libros.ep-electropc.com

5. Haga segmento de programa para verificar *overflow* del Timer1:

```
MainLoop:
; check si hay overflow
      btfss     PIR1,TMR1IF       ; ¿hay overflow? Sí, salte
      goto      MainLoop          ; no, haga ciclo
; código para salto de respuesta si
                                  ; instrucciones
```

6. Al ocurrir un *overflow* debe apagar la bandera TMR1IF del registro INTCON:

```
      bcf       PIR1,TMR1IF       ; apague la bandera del overflow
```

En este modo de *polling* se verifica el bit **TMR1IF** del registro **PIR1**, cuando pasa a '1', lo que indica un *overflow* o paso del **TMR1** de **FFFFh** a **0000h**. Una vez que se realiza esto hay que apagar **'0'** el bit de la bandera **TMR1IF** del registro **PIR1**.

22.2.5.1.2 PRÁCTICA TIMER1 MODO TEMPORIZADOR (TIMER POR POLLING)

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F8XX**, con una frecuencia de **5 Hz** generada con el módulo Timer1, procediendo con el método de *polling*. Si tenemos un **Xtal = 20 MHz = Fosc, la Fosc/4 = 5 MHz**, eso es pulsos cada 0.2 µs = Tcy. Se pone el Timer1 con *prescaler* máximo de '11' = 8. Tiempo = > Tcy (0.2 µs) * Prescaler (8) * max TMR1 (65536) = > 104.8 ms = 0.1s. Con cada *overflow* del TMR1 se alterna el valor de las salidas, esto ocurre cada 0.1 segundos, o sea que el ciclo **ON** y **OFF = 0.2s** o **5 Hz**. En las salidas el tono se escucha en el buzzer **BUZ_1** y el led **OUT_1** titila muy rápido:

```
      list p = 16f877a                  ; directiva para definir procesador
      #include <p16f877a.inc>           ; definiciones de variables específicas del procesador

      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF &
_LVP_OFF & _CPD_OFF

#define   _toggle_BUZ_1     b'00000100'     ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define   _toggle_OUT_1     b'00010000'     ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define   bit_buzzer        .2              ; bit buzzer, puerto B<2>

      org 0
      goto      start

;-------------------------------------------------------------------
; Inicialice puerto B
Ini_Pto
; inicie valores PORTB
      clrf      PORTB             ; apaga el puerto B
      bsf       PORTB,bit_buzzer  ; apague el buzzer, puerto B<2> = 1

      ;configura PORTB
      bsf       STATUS,RP0        ; selecciona Banco 1
      movlw     b'11001011'       ; RB7,6,3 = entradas ICSP
```


libros.ep-electropc.com

```
                                        ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
        movwf   TRISB
        bcf     STATUS,RP0              ; selecciona Banco 0

        return
;-----------------------------------------------------------------------
Ini_TMR1
; configura TMR1
        clrf    T1CON                   ; borra T1CON
        movlw   b'00110000'             ; carga valor de prescaler 1:8 = '11' (bit 5:4) -> W
        movwf   T1CON                   ; guarda valor de prescaler

        bcf     T1CON,TMR1CS            ; asigna la fuente de clock al TMR0
                                        ; 1 = RC0 externo, 0 = Fosc/4 interno
        bsf     T1CON,TMR1ON            ; habilita o prenda TIMER1, 1 = 0N, 0 = OFF

; configura Interrupción TMR1

        return
;-----------------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
        movf    PORTB,W                 ; carga PORTB -> W
        xorlw   _toggle_OUT_1           ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
        xorlw   _toggle_BUZ_1           ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
        movwf   PORTB                   ; pone W -> PORTB

        return
;-----------------------------------------------------------------------
Start:
        call    Ini_Pto                 ; inicialice puertos
        call    Ini_TMR1 ; inicialice TMR1

MainLoop:
; check si hay overflow
        btfss   PIR1,TMR1IF             ; ¿hay overflow? Sí, salte
        goto    MainLoop                ; no, haga ciclo

        call    Toggle_pto              ; toggle puerto
        bcf     PIR1,TMR1IF             ; apague la bandera del overflow
        goto    MainLoop                ; haga ciclo

        end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\TIMER1_1\TIMER1_1.asm.

libros.ep-electropc.com

a) b)

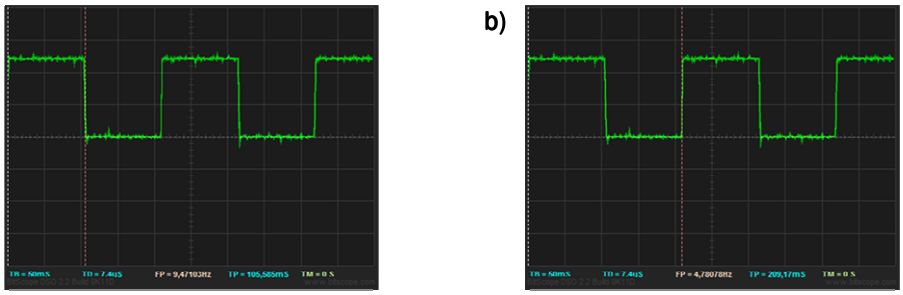


Figura 22.16 Gráficas de 5 Hz.

En la Figura 22.16 a, b se muestra la medición de las señales en el led **OUT_1** y el buzzer **BUZ_1**, el periodo de la señal es de 209 ms aproximado a la frecuencia de **5 Hz**.

22.2.5.1.3 PROCEDIMIENTO DE USO TIMER1 MODO TEMPORIZADOR (TIMER POR INTERRUPCIÓN)

En este método los pulsos internos provienen del **CLKO = Fosc/4**, frecuencia del cristal oscilador, se hace interrupción con la bandera **TMR1IF**, habilitando el bit **TMR1IE** en el registro **PIR1** de la interrupción del módulo **TMR1**. Para asignar el **TMR1** como temporizador por interrupción, haga los pasos siguientes:

1. Borre T1CON
2. Seleccione la tasa (*rate*) de cambio *prescaler*
3. Asigne la fuente de *clock* CLKO = Fosc/4
4. Habilite la interrupción del Timer1, bit TMR1IE del registro PIE1
5. Habilite la interrupción global, bit GIE del registro INTCON
6. Habilite la interrupción periféricos, bit PEIE del registro INTCON
7. Encienda TMR1
8. Haga la rutina de interrupción del TMR1
9. Al final de la rutina de interrupción del TMR1 debe apagar la bandera TMR1IF del registro PIR1
10. Cierre la rutina de interrupción con la instrucción “retfie”

Estos mismos pasos con código son:

1. Borre T1CON:

libros.ep-electropc.com

```
clrf    T1CON           ; borra T1CON
```

2. Seleccione la **tasa** (*rate*) de cambio *prescaler*:

```
iorlw   b'00110000'     ; carga valor de prescaler (bit 5:4) -> W
movwf   T1CON           ; guarda valor de prescaler
```

3. Asigne la fuente de *clock* CLKO = Fosc/4:

```
bcf     T1CON,TMR1CS    ; asigna la fuente de clock al TMR1
                        ; 1 = RC0 externo, 0 = Fosc/4 interno
```

4. Habilite la interrupción del Timer1, bit TMR1IE del registro PIE1:

```
bsf     STATUS,RP0      ; selecciona Banco 1
bsf     PIE1,TMR1IE     ; habilita interrupción del Timer1
bcf     STATUS,RP0      ; selecciona Banco 0
```

5. Habilite la interrupción global, bit GIE del registro INTCON:

```
bsf     INTCON,GIE      ; habilita interrupciones globales
```

6. Habilite la interrupción periféricos, bit PEIE del registro INTCON:

```
bsf     INTCON,PEIE     ; habilita interrupciones periféricos
```

7. Encienda TMR1:

```
bsf     T1CON,TMR1ON    ; habilita o prenda TIMER1, 1 = ON, 0 = OFF
```

8. Haga la rutina de interrupción del TMR1:

```
    org 0004h
ISR_TMR0
; aquí código de la rutina de interrupción
```

9. Al final de la rutina de interrupción del TMR1 debe apagar la bandera TMR1IF del registro PIR1:

```
bcf     PIR1,TMR1IF     ; apague la bandera del overflow
```

10. Cierre la rutina de interrupción con la instrucción **"retfie"**:

```
retfie
```

En este modo de interrupción, cada vez que ocurre un *overflow* del Timer1 (cuando TMR1 pasa de FFFFh a 0000h) el bit de la bandera TMR1IF pasa a '1', se genera la interrupción, el contador de programa pasa a la posición 004h donde se ejecuta la rutina de interrupción **ISR_TMR1**. Al terminar se apaga el bit de la bandera TMR1IF, y se coloca la instrucción "retfie" para terminar.

22.2.5.1.4 PRÁCTICA TIMER1 MODO TEMPORIZADOR (TIMER POR INTERRUPCIÓN)

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F8XX**, con una frecuencia de 5 Hz generada con el módulo Timer1, procediendo con el método de interrupción. Este programa, al igual que el anterior, genera las mismas salidas en la misma frecuencia, la diferencia radica en el método de

libros.ep-electropc.com

interrupción, donde el código de la rutina de interrupción contiene el encendido y apagado del led **OUT_1** y el buzzer **BUZ_1**, y el programa principal sólo es un *loop* infinito sin instrucciones.

```
        list p = 16f877a                    ; directiva para definir procesador
#include <p16f877a.inc>      ; definiciones de variables específicas del procesador

        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF &
_LVP_OFF & _CPD_OFF

#define    _toggle_BUZ_1      b'00000100'      ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1      b'00010000'      ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    bit_buzzer         .2               ; bit buzzer, puerto B<2>

        org 0                       ; vector de reset
        goto        start

        org 0004h     ; vector de interrupción
;-------------------------------------------------------------------------
ISR_TMR1
        call        Toggle_pto      ; toggle puerto

;apague la bandera del overflow
        bcf         PIR1,TMR1IF     ; apague la bandera del overflow

        retfie
;-------------------------------------------------------------------------
; inicialice puerto B
Ini_Pto
; inicie valores PORTB
        clrf        PORTB           ; apaga el puerto B
        bsf         PORTB,bit_buzzer ; apaga el buzzer, puerto B<2> = 1

        ;configura PORTB
        bsf         STATUS,RP0      ; selecciona Banco 1
        movlw       b'11001011'     ;RB7,6,3 = entradas ICSP
                                    ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
        movwf       TRISB
        bcf         STATUS,RP0      ; selecciona Banco 0

        return
;-------------------------------------------------------------------------
Ini_TMR1
; configura TMR1
        clrf        T1CON           ; borra T1CON
        movlw       b'00110000'     ; carga valor de prescaler 1:8 = '11' (bit 5:4) -> W
        movwf       T1CON           ; guarda valor de prescaler

        bcf         T1CON,TMR1CS    ; asigna la fuente de clock al TMR1
                                    ; 1 = RC0 externo, 0 = Fosc/4 interno

; configura interrupción TMR1
        bsf         STATUS,RP0      ; selecciona Banco 1
        bsf         PIE1,TMR1IE     ; habilita interrupción del Timer1
        bcf         STATUS,RP0 ;    selecciona Banco 0
```


libros.ep-electropc.com

```
        bsf     INTCON,GIE          ; habilita interrupciones globales
        bsf     INTCON,PEIE         ; habilita interrupciones periféricos

        bsf     T1CON,TMR1ON        ; habilita o prenda TIMER1, 1 = ON, 0 = OFF

        return
;-----------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
        movf    PORTB,W             ; carga PORTB -> W
        xorlw   _toggle_OUT_1       ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
        xorlw   _toggle_BUZ_1       ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
        movwf   PORTB               ; pone W -> PORTB

        return
;-----------------------------------------------------------------
Start:
        call    Ini_Pto             ; inicialice puertos
        call    Ini_TMR1            ; inicialice TMR1

        goto    $                   ; haga ciclo aquí

        end
```

Este ejemplo está en la Web, en el CD info en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\Timer1_1\Timer1_2.asm.

a) b)

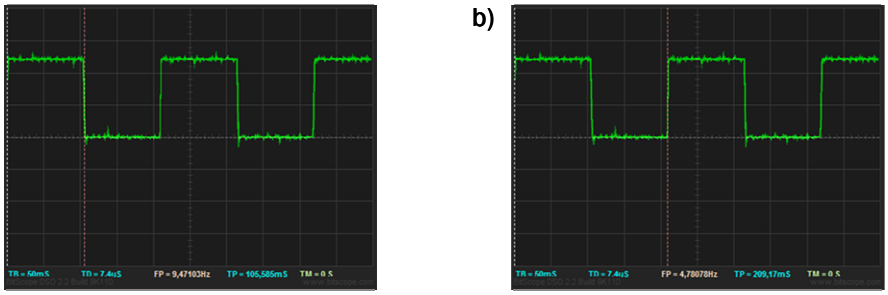


Figura 22.17 Gráficas de 5 Hz.

En la Figura 22.17 a,b se muestra la medición de las señales en el led **OUT_1** y el buzzer **BUZ_1**, el periodo de la señal es de 209 ms aproximado a la frecuencia de 5 Hz.

libros.ep-electropc.com

22.3 TIMER2

El módulo TIMER2 puede funcionar como **temporizador** (*timer*) y tiene un registro de 8 bits con *prescaler* y *postscaler*. Puede usarse como la base de tiempo para el modo **PWM** o módulo **CCP**. El **TMR2** es de lectura-escritura y es aclarado sobre un *reset*.

22.3.1 DIAGRAMA CIRCUITAL

En la Figura 22.18 observamos el diagrama de bloques del módulo TIMER2.

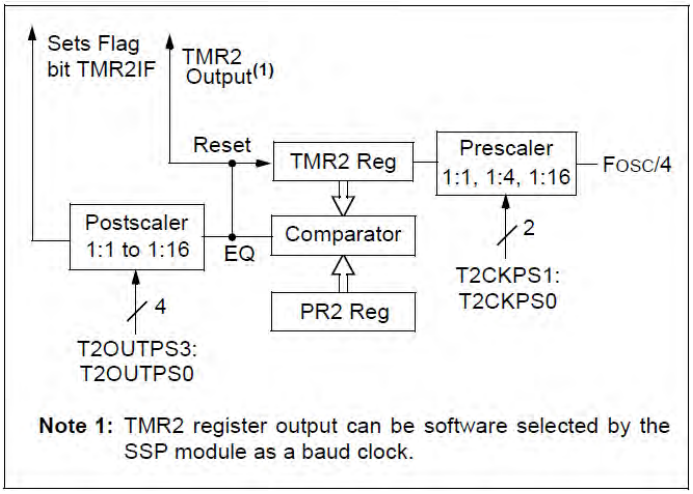


Figura 22.18 Diagrama de bloques del módulo TIMER2.

22.3.2 FUNCIONAMIENTO

El módulo TIMER2 es un temporizador de 8 bits con un *prescaler* y un *postscaler*. Puede usarse como la base de tiempo para el módulo PWM. El registro TMR2 es de lectura y escritura y es aclarado por cualquier reset. La entrada de *clock* (Fosc/4) tiene un *prescaler* de relaciones 1:1, 1:4 o de 1:16, seleccionado por los bits T2CKPS1:T2CKPS0 en el registro T2CON <1:0>. El Timer2 tiene un registro de periodo PR2 de 8 bits, es de lectura y escritura, es inicializado en FFh cuando hay reset. El Timer2 comienza a incrementar desde 00h hasta que iguala a PR2, y entonces resetea a 00h sobre el ciclo siguiente.

La coincidencia de salida del TMR2 se puede postescalar (*postscaler*) con una selección de 16 valores con 4 bits TOUTPS3:TOUTPS0 en el registro T2CON<3:0>, puede generar una interrupción del TMR2. Cuando la coincidencia ocurre se pone

libros.ep-electropc.com

en '1' el bit de la bandera de TMR2IF en el registro PIR1<1>, si se habilita la interrupción poniendo a '1' el bit TR2IE en el registro PIE1<1>, se genera la interrupción del módulo TMR2. El módulo TMR2 se puede apagar, aclarando (poner a '0') el bit TMR2ON en el registro T2CON<2>, para minimizar el consumo de corriente.

Prescaler y postscaler

Los contadores del *prescaler* y el *postscaler* son aclarados cuando cualquiera de estos eventos ocurre:

- Una escritura en el registro TMR2
- Una escritura en el registro T2CON
- Cualquier *reset* de dispositivos (POR, *reset*, WDT *reset* o BOR)

El registro TMR2 no es aclarado cuando se escribe en el registro T2CON.

Salida del TMR2

La salida del TMR2 (antes del postscaler) alimenta al módulo SSP, el cual lo usa opcionalmente para generar el shift clock.

En modo sleep

Durante el modo de sleep, el TMR2 no se incrementa, el prescaler retiene el último conteo y está listo para continuar con la operación una vez que salga del modo de sleep.

22.3.3 REGISTROS ASOCIADOS

En la Tabla 22.7 se describen cada uno de los bits del registro T2CON, y en la Tabla 22.8 están los registros asociados con el módulo TIMER2.

libros.ep-electropc.com

Tabla 22.7 Bits del registro T2CON.

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 postscale
0001 = 1:2 postscale
0010 = 1:3 postscale
•
•
•
1111 = 1:16 postscale

bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Tabla 22.8 Registros asociados con el módulo TIMER como un TIMER/COUNTER.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
92h	PR2	Timer2 Period Register								1111 1111	1111 1111

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer2 module.
Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

22.3.4 PROCEDIMIENTO DE USO

A continuación veremos los pasos necesarios para poner a funcionar el módulo TMR2 en modo de *polling*.

1. Detiene el Timer2
2. Configure el Timer2

libros.ep-electropc.com

3. Encienda el Timer2
4. Pregunte por la bandera TMR2IF en el registro PIR1

A continuación ofrecemos estos pasos con código:

1. Detiene el Timer2:

```
clrf    TMR2            ; detiene TMR2, borra el registro TMR2
```

2. Configure el Timer2:

```
movlw   b'01111011'     ; asigna postscaler = 16,prescaler = 16
movwf   T2CON           ; configura T2CON
```

3. Encienda el Timer2:

```
bsf     T2CON,TMR2ON    ; prende TMR2
```

4. Pregunte por la bandera TMR2IF en el registro PIR1:

```
MainLoop
    btfss   PIR1,TMR2IF     ; ¿hay overflow? Sí, salte
    goto    MainLoop        ; no, haga ciclo

    bcf     PIR1,TMR2IF     ; apague la bandera del overflow
```

22.3.5 PRÁCTICA

Haremos las prácticas para cada uno de los casos siguientes:

- TIMER2 (*timer* por *polling*) TIMER2 (*timer* por interrupción)
-

22.3.5.1 MONTAJE

Para el montaje de las prácticas siguientes se necesitan el depurador **PICkit 2** y la tarjeta de evaluación **PIC16F8XX**, y un osciloscopio (opcional). La foto siguiente es la del montaje utilizado en las prácticas (Figura 22.19).

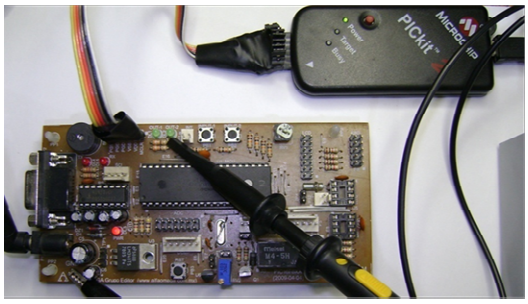


Figura 22.19 Montaje de prácticas Timer2.

libros.ep-electropc.com

Más adelante se muestran las gráficas copiadas del osciloscopio para las dos prácticas siguientes.

22.3.5.2 TIMER2 (TIMER POR POLLING)

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F8XX**, con una frecuencia de 38 Hz generada con el módulo Timer2, procediendo con el método de *polling*.

Si tenemos un Xtal = 20M Hz = Fosc, la Fosc/4 = 5M Hz, eso es pulsos cada 0.2 µs = Tcy.

Se pone el Timer2 con *prescaler* máximo de '11' = 16, y con *postscaler* máximo de '1111' = 16.

Tiempo = > Tcy (0.2 µs) * Prescaler (16) * max TMR2 (256) * Postscaler (16) = > 13,107.2 µs = 13.1 ms.

Tabla 22.9 Timer2 (timer por polling).

Fosc	XTAL - TCY		
XTAL	Tosc	Fosc/4	Tcy
(MHz)	(uS)	(MHz)	(uS)
20	0,05	5	0,2

POSTSCALER - TMR2 - PRESCALER			
postscaler	TMR2	prescaler	T
1111	0xFF	11	(mS)
16	256	16	13,1

T - FCIA			
ON	OFF	CICLO	Fcia
(mS)	(mS)	(mS)	(Hz)
13,1	13,1	26,2	38,2

Con cada *overflow* del TMR2 se alterna el valor de las salidas; esto ocurre cada 13.1 ms, o sea que el ciclo **ON** y **OFF = 26.2 ms** o **38.1 Hz = 38 Hz**. En las salidas el tono se escucha en el buzzer **BUZ_1**, y el led **OUT_1** titila muy rápido:

libros.ep-electropc.com

```
        list p = 16f877a       ; directiva para definir procesador
 #include <p16f877a.inc>       ; definiciones de registros específicas del procesador

        __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF &
_LVP_OFF & _CPD_OFF

        #define     _toggle_BUZ_1     b'00000100'     ;     filtro     para     toggle     buzzer
RB2=BUZ_1=b'0000 0100'
        #define     _toggle_OUT_1     b'00010000'     ; filtro para toggle led RB4 = _OUT_1 =
b'0001 0000'
        #define     bit_buzzer        .2              ; bit buzzer, puerto B<2>

        org 0
        Start
        ;-------------------------------------------------------------------
        ; inicializa puerto B y módulo Timer2
        ; inicie valores PORTB
        clrf    PORTB           ; apaga el puerto B
        bsf     PORTB,bit_buzzer ; apaga el buzzer, puerto B<2> = 1

        ;configura PORTB
        bsf     STATUS,RP0      ; selecciona Banco 1
        movlw   b'11001011'     ;RB7,6,3 = entradas ICSP, RB1,0 = entradas switches, RB =
5,4,2 salidas leds y buzzer
        movwf   TRISB
        bcf     STATUS,RP0      ; selecciona Banco 0

        ; configura TMR0
        clrf    TMR2            ; detiene TMR2, borra el registro TMR2
        movlw   b'01111011'     ; asigna postscaler = 16,prescaler = 16
        movwf   T2CON           ; configura T2CON

        bsf     T2CON,TMR2ON    ; prende TMR2

        ; configura interrupción TMR0
        ;-------------------------------------------------------------------
        MainLoop:
        ; check si hay overflow
        btfss   PIR1,TMR2IF     ; ¿hay overflow? Sí, salte
        goto    MainLoop        ; no, haga ciclo

        bcf     PIR1,TMR2IF     ; apague la bandera del overflow

        Main_overflow
        ; toggle led OUT-1 y buzzer BUZ_1
        bcf     STATUS,RP0      ; selecciona Banco 0
        movf    PORTB,W         ; carga PORTB -> W
        xorlw   _toggle_OUT_1   ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
        xorlw   _toggle_BUZ_1   ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
        movwf   PORTB           ; pone W -> PORTB

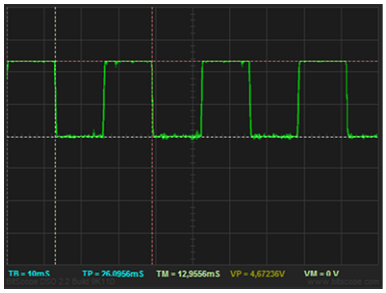
        goto    MainLoop        ; haga ciclo

        end
```


libros.ep-electropc.com

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\TIMER2_1\TIMER2_1.asm.

Figura 22.20 Gráficas de 38 Hz.



En la Figura 22.20 se muestra la medición de las señales en el led **OUT_1** y el buzzer **BUZ_1**, el periodo de la señal es de 26 ms aproximado a la frecuencia de 38 Hz.

22.3.5.3 TIMER2 (TIMER POR INTERRUPCIÓN)

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta PIC1F877a (Tabla 22.10), con una frecuencia de 38 Hz generada con el módulo Timer2, realizando el método de interrupción.

Tabla 22.10 TIMER2 (timer por interrupción).

Fosc	XTAL - TCY		
XTAL	Tosc	Fosc/4	Tcy
(MHz)	(uS)	(MHz)	(uS)
20	0,05	5	0,2

POSTSCALER - TMR2 - PRESCALER			
postscaler	TMR2	prescaler	T
1111	0xFF	11	(mS)
16	256	16	13,1

T - FCIA			
ON	OFF	CICLO	Fcia
(mS)	(mS)	(mS)	(Hz)
13,1	13,1	26,2	38,2

libros.ep-electropc.com

Este programa, al igual que el anterior, genera las mismas salidas en la misma frecuencia; la diferencia radica en el método de interrupción, donde el código de la rutina de interrupción contiene el encendido y apagado del led **OUT_1** y el buzzer **BUZ_1**.

```
        list p = 16f877a       ; directiva para definir procesador
      #include <p16f877a.inc>          ; definiciones de registros específicas del procesador

      __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF &
_LVP_OFF & _CPD_OFF

      #define       _toggle_BUZ_1       b'00000100'       ;       filtro       para       toggle       buzzer
RB2=BUZ_1=b'0000 0100'
      #define       _toggle_OUT_1       b'00010000'       ; filtro para toggle led RB4 = _OUT_1 =
b'0001 0000'
      #define       bit_buzzer          .2                ; bit buzzer, puerto B<2>

        org 0
        goto                start                ; vaya a inicio de programa principal

        org 004h
      ;-------------------------------------------------------------------
      ; rutina de servicio de interrupción TMR2
      ISR_TMR2
      ; toggle led OUT-1 y buzzer BUZ_1
       bcf STATUS,RP0 ; selecciona Banco 0
        movf    PORTB,W          ; carga PORTB -> W
        xorlw   _toggle_OUT_1    ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
        xorlw   _toggle_BUZ_1    ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
        movwf   PORTB            ; pone W -> PORTB

        bcf     PIR1,TMR2IF      ; apaga la bandera del overflow TMR2

        retfie
      ;-------------------------------------------------------------------
      ; inicie puerto B
      Ini_PTOB
      ; inicie valores PORTB
        clrf    PORTB            ; apaga el puerto B
        bsf     PORTB,bit_buzzer ; apaga el buzzer, puerto B<2> = 1

      ;configura PORTB
        bsf     STATUS,RP0       ; selecciona Banco 1
        movlw   b'11001011'      ;RB7,6,3 = entradas ICSP
                                 ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
        movwf   TRISB
        bcf     STATUS,RP0       ; selecciona Banco 0

        return
      ;-------------------------------------------------------------------
      ; inicializa puerto B y módulo Timer0
      Ini_TMR2
      ; configura TMR2
        clrf    TMR2             ; detiene TMR2, borra el registro TMR2
```


libros.ep-electropc.com

```
    movlw   b'01111011'     ; asigna postscaler = 16,prescaler = 16
    movwf   T2CON           ; configura T2CON

; configura interrupción TMR2
    bsf     STATUS,RP0      ; selecciona Banco 1
    bsf     PIE1,TMR2IE     ; habilita interrupción de TMR2
    bcf     STATUS,RP0      ; selecciona Banco 0

    bsf     INTCON,PEIE     ; habilita interrupción de periféricos
    bsf     INTCON,GIE      ; habilita interrupciones globales

    bsf     T2CON,TMR2ON    ; prende TMR2

    return
;-----------------------------------------------------------------
Start:
    call    Ini_PTOB        ; inicialice puerto B
    call    Ini_TMR2        ; inicializa Timer2

MainLoop:
    goto    $               ; haga ciclo

    end
```

Este ejemplo está en la Web, en la carpeta X:\ Ejemplos_ASM\, que se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\TIMER2_1\TIMER2_1A.asm.

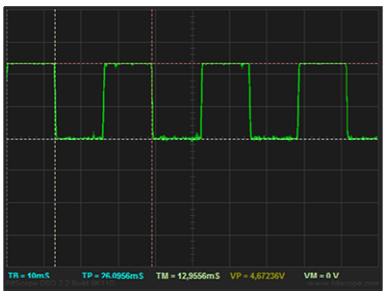


Figura 22.21
Gráficas de 38 Hz.

En la Figura 22.21 se muestra la medición de las señales en el led **OUT_1** y el buzzer **BUZ_1**, el periodo de la señal es de 26 ms aproximado a la frecuencia de 38 Hz.

EVALUACIÓN

```
1. ¿Cuántos TIMERS hay?
R.:
3. Timer0, Timer1, Timer02
```


libros.ep-electropc.com

```
2. ¿De cuántos bits el Timer0 tiene un registro y un prescaler?
```

```
R.:
El registro TMR0 = 8 bits, el prescaler = 8 bits

3. En el Timer0, el prescaler ¿con qué otra función se comparte?
R.: con el watchdog.

4. En el Timer0, ¿cuánto es el máximo de divisiòn de pulsos con
prescaler y TMR0?
R.:
el máximo de división de pulsos es de 65 536 = 256 (prescaler) *
256 (TMR0).

5. ¿De cuántos bits el Timer1 tiene un registro y un prescaler?
R.:
 El registro TMR1 = 16 bits (TMR1H y TMR1L).
El prescaler = 2 bits divide por 1/2/4/8.

6. En el Timer1, ¿de cuánto es el máximo de división de pulsos con
prescaler y TMR1?
R.:
el máximo de división de pulsos es de 524 288 = 8 (prescaler) *
65536 (TMR1).

7. ¿De cuántos bits El Timer2 tiene un registro, un prescaler y un
postscaler?
R.:
El registro TMR2 = 8 bits (TMR2).
El prescaler = 2 bits, divide por 1/4/16.
El postscaler = 4 bits, divide por 1/../16.

8. En el Timer2, ¿de cuánto es el máximo de división de pulsos con
prescaler más postscaler y TMR2?
R.:
el máximo de división de pulsos es de 65 536 = 16 (prescaler) * 16
(postscaler) * 256 (TMR1).
```

23.0 CAPTURA/COMPARACIÓN/PWM (CCP)

Existen dos módulos **CCP: CCP1** y **CCP2**, idénticos en operación, con la excepción de los eventos especiales del *Trigger*. Cada módulo tiene un registro de 16 bits y puede operar como:

libros.ep-electropc.com

1. Registro de 16 bits de captura (*input capture*), asociado con el Timer1
2. Registro de 16 bits de comparación (*output compare*), asociado con el Timer1
3. Registro de 10 bits de **PWM** (*Pulse Width Modulation*), asociado con el Timer2

En la Tabla 23.1 se describen los modos de operación con la fuente asociada y los bits de trabajo.

Tabla 23.1 Modos de operación con la fuente asociada y los bits de trabajo.

Modos de operación, Fuente asociada, No. Bits		
CCP Modo	**Fuente de Timer**	**No Bits**
Capture	Timer1	16 bit
Compare	Timer1	16 bit
PWM	Timer2	10 bit

En la Tabla 23.2 vemos las posibles formas de interacción de los módulos CCP.

Tabla 23.2 Interacción de dos módulos CCP.

TABLE 8-2: INTERACTION OF TWO CCP MODULES

CCPx Mode	CCPy Mode	Interaction
Capture	Capture	Same TMR1 time base
Capture	Compare	The compare should be configured for the special event trigger which clears TMR1
Compare	Compare	The compare(s) should be configured for the special event trigger which clears TMR1
PWM	PWM	The PWMs will have the same frequency and update rate (TMR2 interrupt)
PWM	Capture	None
PWM	Compare	None

En las secciones siguientes la operación del módulo **CCP** se describe para el **CCP1**; **CCP2** opera igual al **CCP1**, a no ser que sea denotado. Al describir los registros o bits lo indicaremos como x para señalar que se trata del módulo **CCP1** o **CCP2**; p. ej., **CCPxCON** se refiere al registro **CCP1CON** o **CCP2CON**, según el módulo que desee trabajar el usuario. Cada módulo de CCP tiene tres registros de trabajo y un pin, como apreciamos en la Tabla 23.3.

Tabla 23.3 Registros de trabajo y pin de cada módulo de CCP.

Generic Name	CCP1	CCP2	Comment
CCPxCON	CCP1CON	CCP2CON	CCP control register
CCPRxH	CCPR1H	CCPR2H	CCP High byte
CCPRxL	CCPR1L	CCPR2L	CCP Low byte
CCPx	CCP1	CCP2	CCP pin

libros.ep-electropc.com

Cada módulo de **capture/compare** y **PWM** contiene:

- Registro de control
- Registro de 16 bits (*high:low*)
- Pin directo del MCU

23.1 MODO CAPTURA (CAPTURE)

El **modo de captura** consiste en capturar el temporizador cuando ocurre un evento en el pin de entrada. Así podemos medir el tiempo transcurrido entre eventos causados en el pin, con la precisión que deseemos.

23.1.1 DIAGRAMA CIRCUITAL

En la Figura 23.1 apreciamos el diagrama de bloques del modo de *capture*.

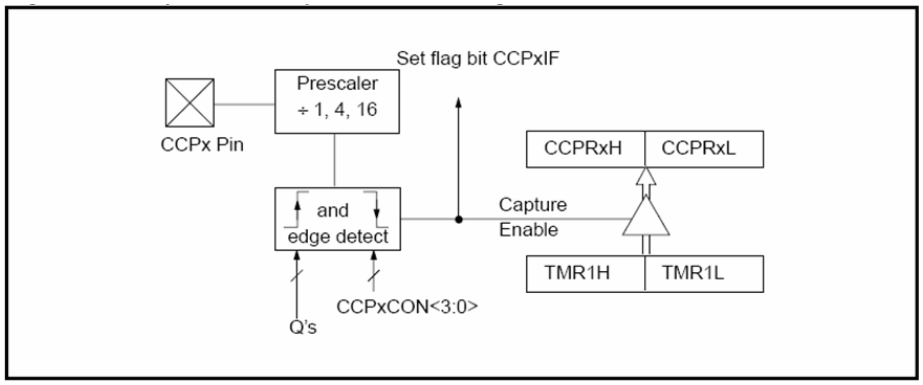


Figura 23.1 Diagrama de bloques del modo de *capture*.

23.1.2 FUNCIONAMIENTO

En el **modo de captura**, un evento en el pin de entrada **RC2/CCP1** o **RC1/CCP2** captura el valor de 16 bits del registro del TMR1 en los registros **CCPRxH:CCPRxL**. Un evento es definido como uno de los siguientes:

1. Cada flanco de caída (*falling*)
2. Cada flanco de subida (*rising*)

libros.ep-electropc.com

3. Cada cuarto (4) flanco de subida (*rising*)
4. Cada decimosexto (16) flanco de subida (*rising*)

El tipo de evento es configurado por los bits de control CCPxM3:CCPxM0 en el registro **CCPxCON<3:0>**. Cuando se hace una captura, el bit de la bandera de interrupción CCPxIF, en el registro PIRx, es seteado. La bandera de interrupción debe ser aclarada por software. Si otra captura ocurre antes que los valores de los registros CCPRx sean leídos, los valores anteriores serán sobrescritos por los valores nuevos. Con el bit CCPxIE puesto a **'1'** en el registro **PIEx** se habilita la interrupción. En la Tabla 23.4 vemos los bits de interrupción y registros correspondientes para cada módulo CCP.

Tabla 23.4 Bits de interrupción y registros correspondientes de cada módulo CCP.

CCPx	Bit Bandera de Interrupción	REGISTRO	Bit Habilitación de Interrupción	REGISTRO	PIN	TRIS
CCP1	CCP1IF	PIR1<2>	CCP1IE	PIE1<2>	RC2/CCP1	TRISC<2>
CCP1	CCP2IF	PIR2<0>	CCP2IE	PIE2<0>	RC1/CCP2	TRISC<1>

Configuración del pin CCP

En **modo de captura** el pin **RC2/CCP1 (**o **RC1/CCP2)** debe ser configurado como entrada, seteando el bit en el registro **TRISC<2> (o <1>)**.

```
NOTA: Si el pin RC2/CCP1 es configurado como una salida, realizar
una escritura en el puerto puede causar una condición de captura.
```

Selección del modo del Timer1

El Timer1 debe estar corriendo en modo de *timer* o en *counter* sincrónico para que el módulo CCP lo use en la función de captura. En el modo de *counter* asincrónico, la función de captura podría no trabajar.

Cambios entre modos de captura

Cuando el modo de captura es cambiado, puede generarse una interrupción falsa. El usuario debe mantener el bit CCPxIE en el registro PIEx aclarado, para evitar una

libros.ep-electropc.com

interrupción falsa, y debe aclarar el bit de la bandera CCPxIF, después de hacer cambios en el modo de funcionamiento.

libros.ep-electropc.com

Prescaler CPP

Hay cuatro ajustes de *prescaler*, especificado por los bits CCP1M3:CCP1M0. Cada vez que el módulo CPP es apagado, o el módulo de CPP no está en modo de captura, el contador del *prescaler* se borra. Cualquier reset borra el contador del *prescaler.* En la captura, cambiando ("switcheando") de un *prescaler* a otro, se puede generar una interrupción. También el contador del *prescaler* no será aclarado; por eso la primera captura puede ser con el *prescaler* no desde "0". En el ejemplo siguiente se muestra el método recomendado para cambiar de *prescaler,* este método también aclara el contador de *prescaler*, y no genera interrupciones falsas.

```
CLRF    CCP1CON      ; Turn CCP module off
MOVLW   NEW_CAPT_PS  ; Load the W reg with
                     ; the new prescaler
                     ; move value and CCP ON
MOVWF   CCP1CON      ; Load CCP1CON with this
                     ; value
```

Figura 23.2 Cambio de prescaler en modo Captura.

En modo de sleep

En este modo el TIMER1 no se incrementa (desde el modo sincrónico), pero el *prescaler* puede seguir contando eventos (no sincronizados). Cuando un evento de captura ocurre, el CCPxIF será puesto, pero el registro de captura no será actualizado. Si la interrupción del CCP es habilitada, podrá despertar al micro del modo de *sleep*. El valor en el registro de 16 bits TMR1 no es transferido al registro de captura, porque el *timer* no ha sido incrementado y carece de significado el valor almacenado. De todas maneras, el CCP puede ser usado como otra interrupción externa.

Reset

El módulo es apagado y el valor en el *prescaler* es forzado a 0.

23.1.3 REGISTROS ASOCIADOS

En la Tabla 23.5 se describe cada uno de los bits del registro CCPxCON, y en la Tabla 23.6 están los registros asociados con los módulos CCP.

libros.ep-electropc.com

Tabla 23.5 Registro CCPxCON.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Tabla 23.6 Registros asociados con los módulos CCP.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by Capture and Timer1.
Note 1: The PSP is not implemented on 28-pin devices; always maintain these bits clear.

libros.ep-electropc.com

23.1.4 PROCEDIMIENTO DE USO

Se pueden hacer dos formas de *input capture*:

- *Capture* por *polling*.
- *Capture* por interrupción.

En cualquiera de las formas anteriores se debe configurar:

- Salida del pin CCPx
- Timer1
- Módulo CCPx

Describiremos los dos métodos con sus pasos respectivos. Para trabajar con el módulo *input capture* en forma de por *polling,* siga los pasos siguientes:

1. Detenga el TMR1, apague el módulo CCPx
2. Configure el módulo CCP para la operación *capture*
3. Configure CCPx pin como salida
4. Detenga el TMR1 y asigne el valor del periodo TMR1H:TMR1L
5. Asigne el valor de *prescaler* y el modo sincrónico de TMR1
6. Encienda el módulo TIMER1
7. Verifique si la bandera de CCP1IF en el registro PIR1 ocurre
8. Guarde las lecturas CCPR1H:CCPR1L
9. Apague la bandera CCPR1IF

Los pasos anteriores con código son:

1. Detiene el TMR1, apague el módulo CCPx:

```
clrf    T1CON           ; detiene TIMER1

clrf    CCP1CON         ; CCP Module is off
```

2. Configure el módulo CCP para la operación *capture*:

```
movlw   b'00000101'     ; modo capture '0101' cada flanco de subida
movwf   CCP1CON         ; configura CCP1CON
```

3. Configure CCPx pin como salida:

```
bsf     STATUS,RP0      ; Bank 1
```


libros.ep-electropc.com

```
    bcf     TRISC,2         ; haga CCP1 = RC2 = Output
    bcf     STATUS,RP0      ; Bank 0
```

4. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L:

```
    clrf    T1CON           ; detiene TIMER1
;   clrf    TMR1H           ; borra registro alto TMR1
;   clrf    TMR1L           ; borra registro bajo TMR1
```

5. Asigne el valor del *prescaler* y el modo sincrónico de TMR1:

```
    movlw   b'00110000'     ; pone prescaler '11' = 1:8
    movwf   T1CON           ; guarda valor de prescaler

    bcf     T1CON,TMR1CS    ; fuente de clock al TIMER1
                            ; 1 = RC0 externo, 0 = CLKO = Fosc/4
```

6. Encienda el módulo TIMER1:

```
    bsf     T1CON,TMR1ON    ; habilita o prenda TIMER1, 1 = ON, 0 = OFF
```

7. Verifique si la bandera de CCP1IF en el registro PIR1 ocurre:

```
Main_CPP:
; check si hay pulso de subida en RC2 = CCP1
    btfss   PIR1,CCP1IF     ; hay match de CCP1? sí, salte
    goto    Main_CPP        ; no, haga ciclo
```

8. Guarde lecturas CCPR1H:CCPR1L:

```
; guarda la lectura en forma indirecta
    movf    CCPR1H,W        ; carga parte alta de CCP1R en W
    movwf   INDF            ; guárdela indirecta/.

    incf    FSR,F           ; FSR + 1

    movf    CCPR1L,W        ; carga parte baja de CCP1R en W
    movwf   INDF            ; guarda la indirecta/.

    incf    FSR,F           ; FSR + 1
```

9. Apaga la bandera CCPR1IF:

```
    bcf     PIR1,CCP1IF     ; apaga la bandera de match CCP1
```

Para trabajar con el módulo *input capture* en forma de interrupción, siga los pasos siguientes:

1. Detiene el TMR1, apague el módulo CCPx
2. Configure el módulo CCP para la operación *capture*
3. Configure CCPx pin como salida
4. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L
5. Designe el valor de *prescaler* y el modo sincrónico de TMR1

libros.ep-electropc.com

6. Habilite las interrupciones globales bit GIE en registro INTCON
7. Habilite las interrupciones de periféricos bit PEIE en registro INTCON
8. Habilite la interrupción de CCPxIE
9. Encienda el módulo TIMER1
10. Haga la rutina de servicio de interrupción para CCPx, lea CCPR1H:CCPR1L
11. Apaga la bandera CCPR1IF

Los pasos anteriores con código son:

1. Detiene el TMR1, apague el módulo CCPx:

```
    clrf    T1CON           ; detiene TIMER1

    clrf    CCP1CON         ; CCP Module is off
```

2. Configure el módulo CCP para la operación *capture*:

```
    movlw   b'00000101'     ; modo capture '0101' cada flanco de subida
    movwf   CCP1CON         ; configura CCP1CON
```

3. Configure CCPx pin como salida:

```
    bsf     STATUS,RP0      ; Bank 1
    bcf     TRISC,2         ; haga CCP1 = RC2 = Output
    bcf     STATUS,RP0      ; Bank 0
```

4. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L:

```
    clrf    T1CON           ; detiene TIMER1
;   clrf    TMR1H           ; borra registro alto TMR1
;   clrf    TMR1L           ; borra registro bajo TMR1
```

5. Asigne el valor del *prescaler* y el modo sincrónico del TMR1:

```
    movlw   b'00110000'     ; pone prescaler '11' = 1:8
    movwf   T1CON           ; guarda valor de prescaler

    bcf     T1CON,TMR1CS    ; fuente de clock al TIMER1
                            ; 1 = RC0 externo, 0 = CLKO = Fosc/4
```

6. Habilite las interrupciones globales bit GIE en registro INTCON:

```
    bsf     INTCON,GIE      ; habilita interrupciones globales
```

7. Habilite las interrupciones de periféricos bit PEIE en registro INTCON:

```
    bsf     INTCON,PEIE     ; habilita interrupciones de periféricos
```

8. Habilite la interrupción de CCPxIE:

```
    bsf     STATUS,RP0      ; selecciona Banco 1
    bsf     PIE1,CCP1IE     ; habilita interrupción de CCP1
    bcf     STATUS,RP0      ; selecciona Banco 0
```

9. Encienda el módulo TIMER1:

```
    bsf     T1CON,TMR1ON    ; habilita o prenda TIMER1, 1 = ON, 0 = OFF
```


libros.ep-electropc.com

10. Haga la rutina de servicio de interrupción para CCPx, lea CCPR1H:CCPR1L:

```
ISR_CCP1

; guarda la lectura en forma indirecta
    movf    CCPR1H,W        ; carga parte alta de CCP1R en W
    movwf   INDF            ; guarda la indirecta/.

    incf    FSR,F           ; FSR + 1

    movf    CCPR1L,W        ; carga parte baja de CCP1R en W
    movwf   INDF            ; guarda la indirecta/.

    incf    FSR,F           ; FSR + 1
```

11. Apaga la bandera CCPR1IF:

```
    bcf     PIR1,CCP1IF     ; apaga la bandera de match CCP1

    retfie
```

A continuación veremos las prácticas para estos dos modos de operación del módulo *input capture*.

23.1.5 PRÁCTICA

Haremos las prácticas para:

- *Capture* por *polling*.
- *Capture* por interrupción.

Ambas prácticas miden el tiempo entre pulsos de una señal proveniente de un circuito de cruce por cero, de la línea de 120VAC / 60 Hz.

23.1.5.1 MONTAJE

Para las prácticas siguientes se necesitan:

- Herramientas: Depurador PICkit 2, tarjeta de evaluación PIC16F8XX, osciloscopio (opcional).
- Circuito cruce por cero optocoplado con MCT6, con pulsos a cero.

Este montaje es para las dos últimas prácticas.

Las fotos siguientes son las del montaje, el esquema del circuito cruce por cero optocoplado y su montaje. El circuito de cruce por cero se utiliza en estas dos prácticas (Figura 23.2).

libros.ep-electropc.com

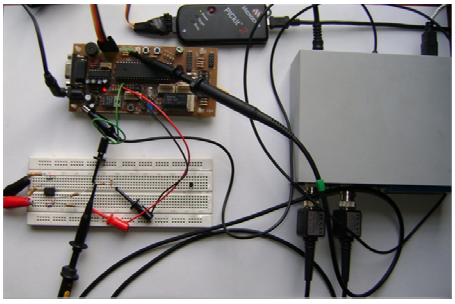


Figura 23.2 Montaje prácticas *Capture*.

En la Figura 23.2 se observan todas las partes del montaje, incluso el circuito de cruce por cero que se implementó en protoboard.

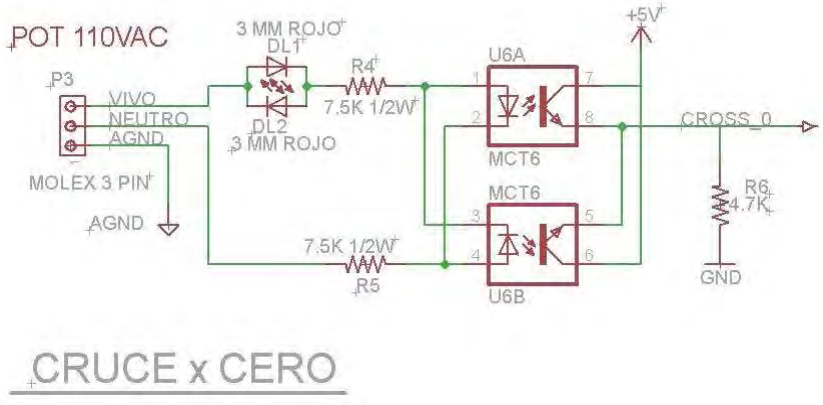


Figura 23.3 Esquema circuito cruce por cero optocoplado.

El esquema de la Figura 23.3 muestra el circuito de cruce por cero, el montaje en protoboard se ve en la Figura 23.4, contiene la entrada de 110VAC con caimanes a la izquierda, y los puntos de conexión a la tarjeta con *test point*.

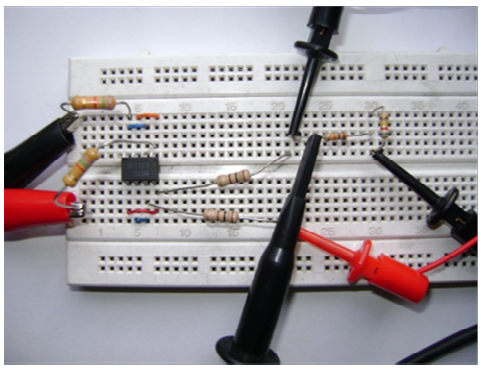


Figura 23.4 Montaje circuito cruce por cero optocoplado.

libros.ep-electropc.com

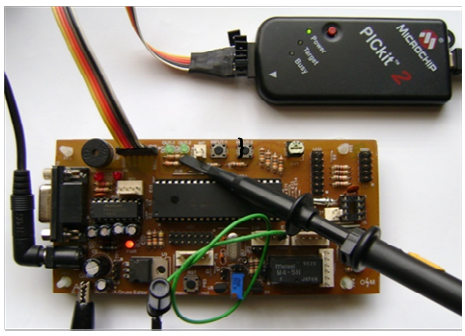


Figura 23.5 Conexión a tarjeta PIC16F8XX.

En la Figura 23.5 se muestran los puntos de prueba conectados a la tarjeta en el conector P4, pin 1 a GND con cable negro, pin 2 a RC2/CCPI con cable verde, pin 6 a +5V con cable rojo. Más adelante están las gráficas copiadas del osciloscopio para las dos prácticas siguientes.

23.1.5.1.1 PRÁCTICA CAPTURE POR POLLING

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta PIC1F877A, con una frecuencia de 60 Hz. Se tiene un circuito optocoplado de cruce por 0, tomado de la línea de 120VAC /60 Hz, que genera 120 Hz. Estos pulsos se conectan a la entrada CCP1 = RC2 en modo de *input capture*. Cada uno de estos pulsos se mide con el *input capture* y los datos son almacenados en las posiciones 0×20 a 0×37, que corresponden a 24 registros, o 12 pares de registros.

Cada par de registros contiene una lectura del *input capture*, de tal forma que los registros 0×20 y 0×21 corresponden al contenido CCPR1H:CCPR1L del *input capture* de la primera lectura o primer cruce por cero, el par de registros 0×22 y 0×23 corresponde al contenido CCPR1H:CCPR1L del *input capture* de la segunda lectura o segundo cruce por cero, y así sucesivamente hasta el par de registros 0×36 y 0×37, que corresponden a la lectura número 12 de cruce por cero. Estos datos son llevados a una tabla de Excel y presentados al final, junto con las gráficas tomadas con el osciloscopio. El código del programa es:

```
            list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1        b'00000100'        ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1        b'00010000'        ; filtro para toggle led RB4 =_OUT_1=b'0001 0000'
#define    bit_buzzer           .2                 ; bit buzzer, puerto B<2>
```


libros.ep-electropc.com

```
            cblock      0x20                ; separa memoria desde 0x20
datos_capture   .24                         ; separa 24 posiciones correspondientes a 12 datos
cont_datos                                  ; contador de datos
            endc

                        org         000h
Start:
;-------------------------------------------------------------------
; inicialice puerto B

            clrf        PORTB               ; apague el puerto B
            bsf         PORTB,bit_buzzer    ; apague el buzzer, puerto B<2> = 1

;configura PORTB
            bsf         STATUS,RP0          ; selecciona Banco 1
            movlw       b'11001011'         ;RB7,6,3 = entradas ICSP,
                                            ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf       TRISB
            bcf         STATUS,RP0          ; selecciona Banco 0
;-------------------------------------------------------------------
; configura input capture
            clrf        T1CON               ; detiene TIMER1
                                            ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3..CCPxM0/
            clrf        CCP1CON             ; CCP Module is off

            movlw       b'00000101'         ; modo capture '0101' cada flanco de subida
            movwf       CCP1CON             ; configura CCP1CON

; configura PORT C
            bsf         STATUS,RP0          ; Bank 1
            bsf         TRISC,2             ; haga CCP1 = RC2 = Input
            bcf         STATUS,RP0          ; Bank 0

; configura TIMER1
                                ; T1CON=/-/-/T1CKPS1/T1CKPS0/T1OSCEN/T1SYNC/TMR1CS/TMR1ON
            clrf        T1CON               ; detiene TIMER1
;           clrf        TMR1H               ; borra registro alto TMR1
;           clrf        TMR1L               ; borra registro bajo TMR1
            movlw       b'00110000'         ; pone prescaler '11' = 1:8
            movwf       T1CON               ; guarda valor de prescaler

            bcf         T1CON,TMR1CS        ; fuente de clock al TIMER1
                                            ; 1 = RC0 externo, 0 = CLKO = Fosc/4 interno
            bsf         T1CON,TMR1ON        ; habilita o prenda TIMER1, 1 = 0N, 0 = OFF
; configura Interrupción TMR1

;-------------------------------------------------------------------
Main_ini:
            movlw       datos_capture       ; pos de datos
            movwf       FSR                 ; guarda en apuntador

            movlw       .24                 ; carga 24 datos
            movwf       cont_datos          ; guarda en contador de datos
;-------------------------------------------------------------------
Main_CPP:
; check si hay pulso de subida en RC2 = CCP1
```


libros.ep-electropc.com

```
            btfss   PIR1,CCP1IF        ; ¿hay match de CCP1? Sí, salte
            goto    Main_CPP           ; no, haga ciclo

; toggle led OUT-1 y buzzer BUZ_1
            movf    PORTB,W            ; carga PORTB -> W
            xorlw   _toggle_OUT_1      ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1      ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB              ; pone W -> PORTB
;-------------------------------------------------------------------
; guarda la lectura en forma indirecta
            movf    CCPR1H,W           ; carga parte alta de CCP1R en W
            movwf   INDF               ; guarda la indirecta/.

            incf    FSR,F              ; FSR + 1

            movf    CCPR1L,W           ; carga parte baja de CCP1R en W
            movwf   INDF               ; guarda la indirecta/.

            incf    FSR,F              ; FSR + 1

            bcf     PIR1,CCP1IF        ; apaga la bandera de match CCP1
;-------------------------------------------------------------------
; verificar si es última lectura
            decf    cont_datos,F       ; contador de datos - 1
            decfsz  cont_datos,F       ; contador de datos - 1, check = 0?
            goto    Main_CPP           ; continúe con otro dato

            goto    Main_ini           ; haga ciclo inicial

            end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\ CAPTURE_1\ CAPTURE_1.asm.

a) b)

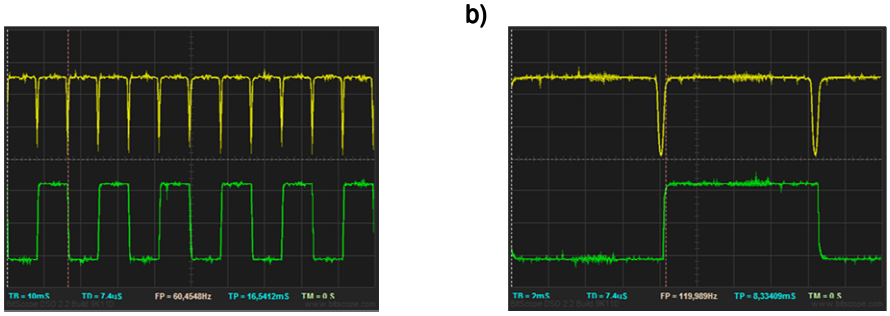


Figura 23.6 Gráficas de *capture*.

En la Figura 23.6 a, b se muestra la medición en el pin CCP1 = RC2 (superior en la gráfica), correspondiente a la señal de cruce por cero, el periodo de la señal es

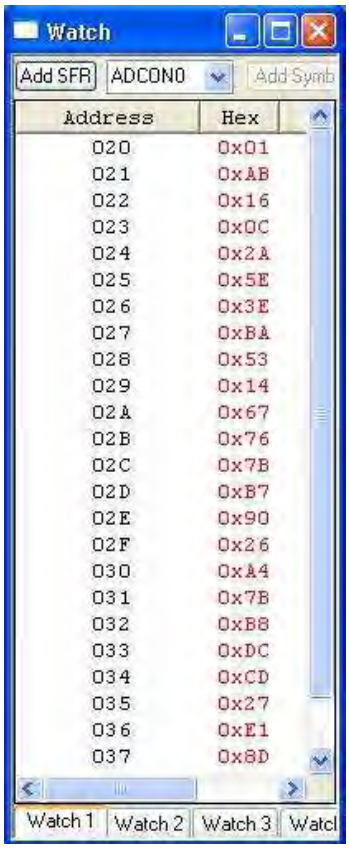
libros.ep-electropc.com

de T = 8.33 ms. En las salidas el tono se escucha en el buzzer **BUZ_1** y se ve en el led **OUT_1** (inferior en la gráfica); estas salidas cambian cada ciclo de cruce por cero, generando una frecuencia de 60 Hz. Para obtener las 12 lecturas realizadas con el modo de *input capture*, debemos correr el programa en modo de depuración, y nos paramos en la última línea de código:

```
goto       Main_ini   ; haga ciclo inicial
```

y allí, con el botón derecho del ratón (*mouse*) en el menú deplegado, escogemos la opción *run to cursor*; así se realizan solamente 12 ciclos, y se obtienen las lecturas resumidas en la Figura 23.7 a, b.

a)



b)

D=A*B*C/1000	T (ms)	8,34
C	Prescaler	8
B	Tcy (uS)	0,2
A	Promedio	5210,4

REGISTROS		PAR DE REGISTROS		DIFERENCIA
HEXA		HEXA	DEC	DEC
0x01	01	01AB	427	5217
0xAB	AB			
0x16	16	160C	5644	5202
0x0C	0C			
0x2A	2A	2A5E	10846	5212
0x5E	5E			
0x3E	3E	3EBA	16058	5210
0xBA	BA			
0x53	53	5314	21268	5218
0x14	14			
0x67	67	6776	26486	5185
0x76	76			
0x7B	7B	7BB7	31671	5231
0xB7	B7			
0x90	90	9026	36902	5205
0x26	26			
0xA4	A4	A47B	42107	5217
0x7B	7B			
0xB8	B8	B8DC	47324	5195
0xDC	DC			
0xCD	CD	CD27	52519	5222
0x27	27			
0xE1	E1	E18D	57741	
0x8D	8D			

Figura 23.7 Cálculos *capture*.

Tomando el primer par de registros 0×20 y 0×21 con sus valores 0×01 y 0×AB se convierte en 01AB que en decimales da el valor de 427; el segundo par de

libros.ep-electropc.com

registros 0×22 y 0×23 con sus valores 0×16 y 0×0C se convierte en 160C, que en decimales es de 5 644. Restando estos valores tenemos 5 644 – 472 = 5 217. De esta manera se hace con todos los demás registros para generar la casilla DIFERENCIA; al hacer un promedio de estos valores tendremos de 5 210.4; este valor es el promedio de todas las diferencias entre lecturas del *input capture*. Luego, al multiplicar por el *prescaler* y el Tcy, obtenemos 8.34 ms que es el valor real de un ciclo de 120 Hz proveniente del circuito de cruce por cero.

23.1.5.1.2 PRACTICA CAPTURE POR INTERRUPCIÓN

El programa siguiente es igual al anterior; prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta **PIC1F877A**, con una frecuencia de 60 Hz. Los datos recogidos con el *input capture* son llevados a una tabla de Excel y presentados al final, junto con las gráficas tomadas con el osciloscopio. A continuación ofrecemos el código del programa:

```
            list p = 16f877a      ; directiva para definir procesador
 #include <p16f877a.inc>      ; definiciones de variables específicas del procesador

  __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1      b'00000100'      ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1      b'00010000'      ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    bit_buzzer.2                        ; bit buzzer, puerto B<2>

            cblock    0x20              ; separa memoria desde 0x20
datos_capture         :.24              ; separa 24 posiciones correspondientes a 12 datos
cont_datos                              ; contador de datos
            endc

                      org       000h              ; vector de reset
                      goto      start

                      org       004h              ; vector de interrupción
;-------------------------------------------------------------------------
ISR_CPP1:
            call      toggle_pto; toggle pin buzzer y led

; guarda la lectura en forma indirecta
            movf      CCPR1H,W          ; carga parte alta de CCP1R en W
            movwf     INDF              ; guarda la indirecta/.

            incf      FSR,F             ; FSR + 1

            movf      CCPR1L,W          ; carga parte baja de CCP1R en W
            movwf     INDF              ; guarda la indirecta/.

            incf      FSR,F             ; FSR + 1

; verifica si es última lectura
            decf      cont_datos,F      ; contador de datos - 1
```


libros.ep-electropc.com

```
            decfsz  cont_datos,F        ; contador de datos - 1, check = 0?
            goto    ISR_end             ; continúe con otro dato

            call    Ini_datos           ; incializa datos para capture

ISR_end
            bcf     PIR1,CCP1IF         ; apaga la bandera de match CCP1

            retfie
;-----------------------------------------------------------------
; inicialice puerto B
Ini_pto
            clrf    PORTB               ; apague el puerto B
            bsf     PORTB,bit_buzzer    ; apague el buzzer, puerto B<2> = 1

;configura PORTB
            bsf STATUS,RP0 ; selecciona Banco 1
            movlw   b'11001011'         ;RB7,6,3 = entradas ICSP
                                        ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf   TRISB
            bcf     STATUS,RP0          ; selecciona Banco 0

            return
;-----------------------------------------------------------------
Ini_CCP
; configura input capture
            clrf    T1CON               ; detiene TIMER1
                                        ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3..CCPxM0/
            clrf    CCP1CON             ; CCP Module is off

            movlw   b'00000101'         ; modo capture '0101' cada flanco de subida
            movwf   CCP1CON             ; configura CCP1CON

; configura PORT C
            bsf     STATUS,RP0          ; Bank 1
            bsf     TRISC,2             ; haga CCP1 = RC2 = Input
            bcf     STATUS,RP0          ; Bank 0

; configura TIMER1
                                        ;       T1CON       =       /-/-/T1CKPS1/T1CKPS0/T1OSCEN/-
T1SYNC/TMR1CS/TMR1ON
            clrf    T1CON               ; detiene TIMER1
;           clrf    TMR1H               ; borra registro alto TMR1
;           clrf    TMR1L               ; borra registro bajo TMR1
            movlw   b'00110000'         ; pone prescaler '11' = 1:8
            movwf   T1CON               ; guarda valor de prescaler

            bcf     T1CON,TMR1CS        ; fuente de clock al TIMER1, 1 = RC0 externo, 0 = CLKO =
Fosc/4 interno

; configura interrupción TMR1
            bsf     INTCON,GIE          ; habilita interrupciones globales
            bsf     INTCON,PEIE         ; habilita interrupciones de periféricos
            bsf     STATUS,RP0          ; selecciona Banco 1
;           bsf     PIE1,TMR1IE         ; habilita interrupción de TMR1
            bsf     PIE1,CCP1IE         ; habilita interrupción de CCP1
```


libros.ep-electropc.com

```
            bcf     STATUS,RP0          ; selecciona Banco 0

            bsf     T1CON,TMR1ON        ; habilita o prenda TIMER1, 1 = ON, 0 = OFF

            return
;-----------------------------------------------------------------
Toggle_pto
; toggle led OUT-1 y buzzer BUZ_1
            movf    PORTB,W             ; carga PORTB -> W
            xorlw   _toggle_OUT_1       ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1       ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB               ; pone W -> PORTB

            return
;-----------------------------------------------------------------
Ini_datos:
            movlw   datos_capture       ; pos de datos
            movwf   FSR                 ; guarda en apuntador

            movlw   .24                 ; carga 24 datos
            movwf   cont_datos          ; guarda en contador de datos

            return
;-----------------------------------------------------------------
Start:
            call    Ini_pto             ; inicialice puerto
            call    Ini_CCP             ; inicialice CCP

            call    Ini_datos           ; inicialice datos para capture

            goto    $                   ; haga ciclo aquí

            end
```

Este ejemplo está en nuestra página Web en la carpeta X:\ Ejemplos_ASM\ y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\ CAPTURE_1\ CAPTURE_1A.asm.

a) b)

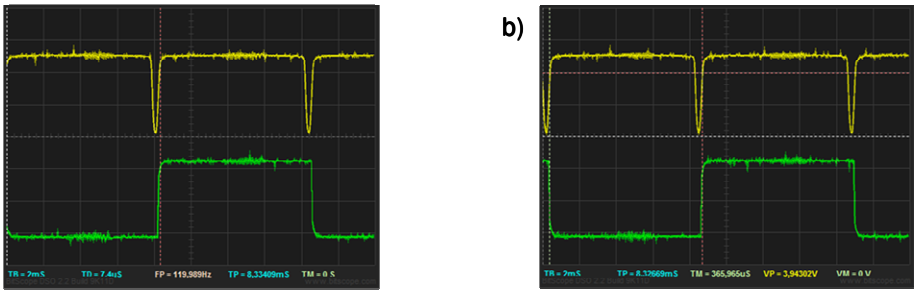


Figura 23.8 Gráficas de *capture*.

En la Figura 23.8 se muestra la medición en el pin CCP1 = RC2 (superior en la gráfica), correspondiente a la señal de cruce por cero, el periodo de la señal es de T

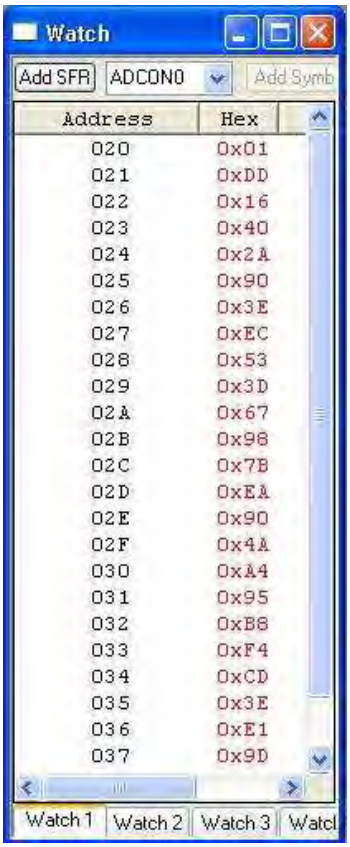
libros.ep-electropc.com

= 8.33 ms. En las salidas el tono se escucha en el buzzer **BUZ_1** y se ve en el led **OUT_1** (inferior en la gráfica), las cuales cambian cada ciclo de cruce por cero, generando una frecuencia de 60 Hz. Para obtener las 12 lecturas realizadas con el modo de *input capture*, debemos correr el programa en modo de depuración, nos paramos en la rutina **ISR_CCP** al final en la línea de código:

```
call        Ini_datos          ; incializa datos para capture
```

y allí, con el botón derecho del ratón (*mouse*) en el menú desplegado, escogemos la opción **run to cursor**; así se realizan solamente 12 ciclos, y se obtienen las lecturas resumidas a continuación (Figura 23.9 a, b).

a)



b)

D=A*B*C/1000	T (ms)	8,33
C	Prescaler	8
B	Tcy (uS)	0,2
A	Promedio	5207,3

REGISTROS		PAR DE REGISTROS		DIFERENCIA
HEXA		HEXA	DEC	DEC
0x01	01	01DD	477	5219
0xDD	DD			
0x16	16	1640	5696	5200
0x40	40			
0x2A	2A	2A90	10896	5212
0x90	90			
0x3E	3E	3EEC	16108	5201
0xEC	EC			
0x53	53	533D	21309	5211
0x3D	3D			
0x67	67	6798	26520	5202
0x98	98			
0x7B	7B	7BEA	31722	5216
0xEA	EA			
0x90	90	904A	36938	5195
0x4A	4A			
0xA4	A4	A495	42133	5215
0x95	95			
0xB8	B8	B8F4	47348	5194
0xF4	F4			
0xCD	CD	CD3E	52542	5215
0x3E	3E			
0xE1	E1	E19D	57757	
0x9D	9D			

Figura 23.9 Cálculos *capture*.

Tomando el primer par de registros 0×20 y 0×21 con sus valores 0×01 y 0×DD, se convierte en 01DD que en decimales da el valor de 477; el segundo par

libros.ep-electropc.com

de registros 0×22 y 0×23 con sus valores 0×16 y 0×40 se convierte en 1 640 que en decimales es de 5 696.

Restando estos valores tenemos 5 696 – 4 772 = 5 219.

De esta manera se hace con todos los demás registros para generar la casilla **Diferencia**; al hacer un promedio de estos valores tendremos 5 210.4; este valor es el promedio de todas las diferencias entre lecturas del *input capture*. Luego, al multiplicar por el *prescaler* y el Tcy, obtenemos 8.33 ms que es el valor real de un ciclo de 120 Hz proveneiente del circuito de cruce por cero.

23.2 MODO DE COMPARACIÓN (COMPARE)

El **modo de comparación** consiste en generar una salida cuando ocurre una coincidencia entre el temporizador y el registro del comparador. Así generamos una señal de salida reloj (*clock*) muy precisa.

23.2.1 DIAGRAMA CIRCUITAL

En la Figura 23.10 apreciamos el diagrama de bloques del modo de **comparación**.

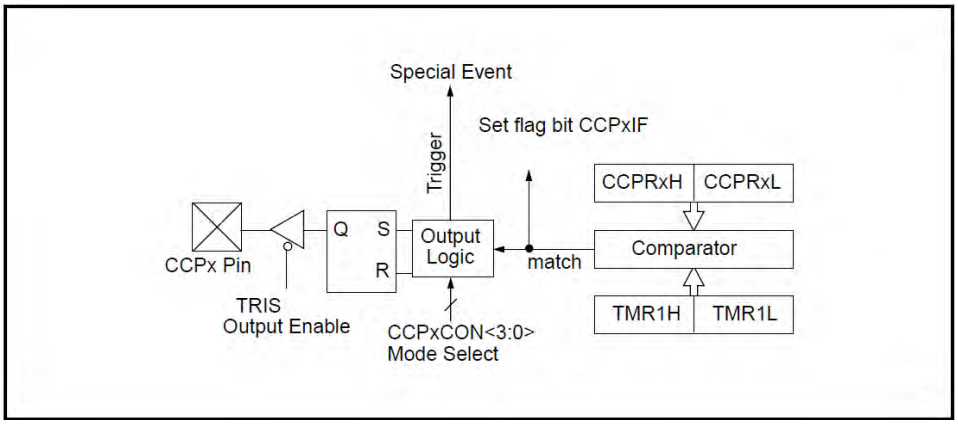


Figura 23.10 Diagrama de bloques del modo de comparación.

23.2.2 FUNCIONAMIENTO

En el modo de **comparación** el registro **CCPRx** de 16 bits es comparado constantemente con el TMR1, cuando ocurre una coincidencia el pin de salida **RC2/CCP1 o RC1/CCP2** queda como:

libros.ep-electropc.com

1. Alto (*high* = "1")
2. Bajo (*low* = "0")
3. Permanece sin cambios

La acción sobre el pin está configurada con los bits de control CCP1M3:CCP1M0 en el registro CCPxCON<3:0>; al mismo tiempo el bit de la bandera de interrupción CCPxIF en el registro PIEx es seteado.

Configuración del pin CCP

En modo de *compare*, el pin **RC2/CCP1 (**o **RC1/CCP2)** debe ser configurado como salida, aclarando el bit en el registro **TRISC<2> (**o **<1>)**.

```
NOTA: Aclarando el registro CCPxCON forzará el pin RC2/CCP1 (o
RC1/CCP2), del latch de salida del comparador al nivel bajo
(default). Este latch no es el del PORTC I/O.
```

Selección del modo del Timer1

El **Timer1** debe estar corriendo en modo de *timer* o en *counter* sincrónico, para que el módulo CCP lo use en la función de comparación. En el modo de **counter** asincrónico, la función de comparación podría no trabajar.

Interrupción de software

Cuando se escoge el modo de generación de **interrupción** de software, el CCPx pin no es afectado. El bit de la bandera CCPxIF es seteado, y causa una interrupción del CCP (si está habilitada).

Eventos especiales del Trigger

En este modo un *Trigger* de hardware interno es generado, y puede ser usado para inicializar una acción. El evento especial del *Trigger* de la salida de CCP1 resetea el par de registros del TMR. Esto permite al registro CCPR1 ser efectivamente un registro de 16 bits de periodo programable para el Timer1. El evento especial de *Trigger* de la salida de CCP2 resetea el par de registros del TMR1, y comienza una conversión A/D (si el módulo A/D es habilitado).

```
NOTA: El evento especial de Trigger para los módulos CCP1 y CCP2
podrá no setear el bit de la bandera de interrupción TMR1IF del
registro PIR1<0>.
```


libros.ep-electropc.com

23.2.3 REGISTROS ASOCIADOS

En la Tabla 23.7 se describen cada uno de los bits del registro CCPxCON, y en la Tabla 23.8 están los registros asociados con los módulos CCP.

Tabla 23.7 Bits del registro CCPxCON.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 23.8 Registros asociados con los módulos CCP.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by Capture and Timer1.
Note 1: The PSP is not implemented on 28-pin devices; always maintain these bits clear.

23.2.4 PROCEDIMIENTO DE USO

Se puede trabajar con el módulo *output compare* de dos maneras:

- *Compare* por *polling*
- *Compare* por interrupción

En cualquiera de las formas anteriores se deben configurar:

- Salida del pin CCPx
- Timer1
- Módulo CCPx

Para trabajar con el módulo *output compare* en forma de por *polling*, siga los pasos siguientes:

1. Configure el CCPx pin como salida
2. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L
3. Asigne el valor del *prescaler* y el modo sincrónico del TMR1
4. Apague el módulo CCPx y asigne el valor del comparador CCPRxH: CCPRxL
5. Configure el módulo CCP para la operación *compare*
6. Encienda el módulo TIMER1

libros.ep-electropc.com

7. Verifique la bandera de TMR1IF en el registro PIR1; ocurre cada periodo
8. Pregunte por la bandera CCP1IF en el registro PIR1; ocurre cuando hay comparación

Los pasos anteriores con código son:

1. Configure CCPx pin como salida:

```
    bsf     STATUS,RP0          ; Bank 1
    bcf     TRISC,2             ; haga CCP1 = RC2 = Output
    bcf     STATUS,RP0          ; Bank 0
```

2. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L:

```
    clrf    T1CON               ; detiene TIMER1
;   clrf    TMR1H               ; borra registro alto TMR1
;   clrf    TMR1L               ; borra registro bajo TMR1
```

3. Asigne el valor de *prescaler* y el modo sincrónico de TMR1:

```
    movlw   b'00110000'         ; pone prescaler '11' = 1:8
    movwf   T1CON               ; guarda valor de prescaler

    bcf     T1CON,TMR1CS        ; fuente de clock al TIMER1, 1 = RC0 externo, 0 = CLKO = Fosc/4
```

4. Apague el módulo CCPx y asigne el valor del comparador CCPRxH: CCPRxL:

```
    clrf    CCP1CON             ; CCP Module is off

    movlw   _CCPR1H             ; carga valor de comparación alto
    movwf   CCPR1H              ; guarda valor de comparación alto

    movlw   _CCPR1L             ; carga valor de comparación bajo
    movwf   CCPR1L              ; guarda valor de comparación bajo
```

5. Configure el módulo CCP para la operación *compare*:

```
    movlw   b'00001000'         ; modo compare '1000' set output on match
    movwf   CCP1CON             ; configura CCP1CON
```

6. Encienda el módulo TIMER1:

```
    bsf     T1CON,TMR1ON        ; habilita o prenda TIMER1, 1 = ON, 0 = OFF
```

7. Verifique la bandera de TMR1IF en el registro PIR1, ocurre cada periodo:

```
Main_TMR1_OVF:
; check si hay overflow
    btfss   PIR1,TMR1IF         ; ¿hay overflow? Sí, salte
    goto    Main_CCP   ; no, haga ciclo

    clrf    CCP1CON   ; CCP Module is off
    movlw   b'00001000'         ; modo compare '1000' set output on match
    movwf   CCP1CON             ; configura CCP1CON

    bcf     PIR1,TMR1IF         ; apague la bandera del overflow
```

8. Pregunte por la bandera CCP1IF en el registro PIR1, ocurre cuando hay comparación:

```
Main_CCP:
; check si hay match CCP1
    btfss   PIR1,CCP1IF         ; hay match de CCP1? sí, salte
```


libros.ep-electropc.com

```
    goto        Main_TMR1_OVF       ; no, haga ciclo

    nop                             ; necesario para apagar CCP1IF,
    nop                             ; cuando prescaler de TMR1 = > 1:8
    bcf         PIR1,CCP1IF         ; apaga la bandera de match CCP1
```

Para trabajar con el módulo *output compare* en forma de interrupción, siga los pasos siguientes:

1. Configure el CCPx pin como salida
2. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L
3. Asigne el valor de *prescaler* y el modo sincrónico de TMR1
4. Apague el módulo CCPx y asigne el valor del comparador CCPRxH: CCPRxL
5. Configure el módulo CCP para la operación *compare*
6. Habilite las interrupciones globales bit GIE en registro INTCON
7. Habilite las interrupciones de periféricos bit PEIE en registro INTCON
8. Habilite la interrupciónde TMR1, bit TMER1IE en registro PIE1
9. También puede habilitar la interrupción de CCPxIE*
10. Encienda el módulo TIMER1
11. Haga la rutina de servicio de interrupción para TMR1
12. Haga la rutina de servicio de interrupción para CCPx

```
*NOTA: Tenga en cuenta que al habilitar la interrupción de módulo
CCPx, la salida CCPx no se afecta.
```

Los pasos anteriores con código son:

1. Configure CCPx pin como salida:

```
    bsf         STATUS,RP0          ; Bank 1
    bcf         TRISC,2             ; haga CCP1 = RC2 = Output
    bcf         STATUS,RP0          ; Bank 0
```

2. Detiene el TMR1 y asigne el valor del periodo TMR1H:TMR1L:

```
    clrf        T1CON               ; detiene TIMER1
;   clrf        TMR1H               ; borra registro alto TMR1
;   clrf        TMR1L               ; borra registro bajo TMR1
```

3. Asigne el valor del *prescaler* y el modo sincrónico de TMR1:

```
    movlw       b'00110000'         ; pone prescaler '11' = 1:8
    movwf       T1CON               ; guarda valor de prescaler
```


libros.ep-electropc.com

```
    bcf     T1CON,TMR1CS    ; fuente de clock al TIMER1
                            ; 1 = RC0 externo, 0 = CLKO = Fosc/4
```

4. Apague el módulo CCPx y asigne el valor del comparador CCPRxH: CCPRxL:

```
    clrf    CCP1CON         ; CCP Module is off

    movlw   _CCPR1H         ; carga valor de comparación alto
    movwf   CCPR1H          ; guarda valor de comparación alto

    movlw   _CCPR1L         ; carga valor de comparación bajo
    movwf   CCPR1L          ; guarda valor de comparación bajo
```

5. Configure el módulo CCP para la operación *compare*:

```
    movlw   b'00001000'     ; modo compare '1000' set output on match
    movwf   CCP1CON         ; configura CCP1CON
```

6. Habilite las interrupciones globales del bit GIE en el registro INTCON:

```
    bsf     INTCON,GIE      ; habilita interrupciones globales
```

7. Habilite las interrupciones de periféricos bit PEIE en registro INTCON:

```
    bsf     INTCON,PEIE     ; habilita interrupciones de periféricos
```

8. Habilite la interrupción de TMR1, bit TMER1IE en registro PIE1:
También puede habilitar la interrupción de CCPxIE*

```
    bsf     STATUS,RP0      ; selecciona Banco 1
    bsf     PIE1,TMR1IE     ; habilita interrupción de TMR1
;   bsf     PIE1,CCP1IE     ; habilita interrupción de CCP1
    bcf     STATUS,RP0      ; selecciona Banco 0
```

9. Encienda el módulo TIMER1:

```
    bsf     T1CON,TMR1ON    ; habilita o prenda TIMER1, 1 = ON, 0 = OFF
```

10. Haga la rutina de servicio de interrupción para TMR1:

```
ISR_TMR1
; check si hay overflow
    clrf    CCP1CON         ; CCP Module is off
    movlw   b'00001000'     ; modo compare '1000' set output on match
    movwf   CCP1CON         ; configura CCP1CON

    bcf     PIR1,TMR1IF     ; apague la bandera del overflow

    retfie
```

11. Haga la rutina de servicio de interrupción para CCPx:

```
ISR_CCP1
; check si hay match CCP1

;   nop                     ; necesario para apagar CCP1IF,
;   nop                     ; cuando prescaler de TMR1 = > 1:8
    bcf     PIR1,CCP1IF     ; apaga la bandera de match CCP1

    retfie
```


libros.ep-electropc.com

```
*NOTA: Tenga en cuenta que al habilitar la interrupción de módulo
CCPx, la salida CCPx no se afecta.
```

A continuación veremos las prácticas para estos dos modos de operación del módulo *output compare*.

23.2.5 PRÁCTICA

Haremos las prácticas para

- *Compare* por *polling*.
- *Compare* por interrupción.

23.2.5.1 MONTAJE

Para el montaje de las prácticas siguientes se necesitan el depurador PICkit 2 y la tarjeta de evaluación **PIC16F8XX**, y un osciloscopio (opcional). La foto del montaje es la siguiente que se utiliza en las prácticas, como muestra la Figura 23.11.

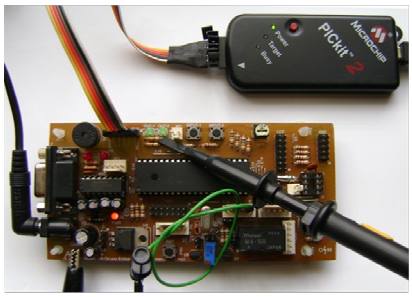


Figura 23.11 Montaje prácticas *compare*.

23.2.5.1.1 PRÁCTICA COMPARE POR POLLING

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta PIC1F877A, con una frecuencia de 9.5 Hz, generada con el módulo *compare*, procediendo con el método de *polling*. Si tenemos un Xtal = 20 MHz, Tcy = 0.2 µs, TMR1 = 0×FFFF = 65 536, con *prescaler* = '11' = 8, obtendremos un periodo de: T = Tcy * TMR1 * Prescaler = 0.2 µs * 65 536 * 8 = 104 857.6 ns = 105 ms, lo que da una Fcia = 9.5 Hz. En este periodo de 105 ms, haremos un *compare* en el valor CCP1 = 0×C000, lo que nos da un Toff = 78.7 ms = 75% y Ton = 26.2 ms = 25%.

libros.ep-electropc.com

Tabla 23.9 Cálculos *compare*.

Fosc	XTAL - TCY		
XTAL	Tosc	Fosc/4	Tcy
(MHz)	(uS)	(MHz)	(uS)
20	0,05	5	0,2

TMR1 - PRESCALER				
TMR1H	TMR1L	prescaler	T	Fcia
0xFF	0xFF	11	(mS)	(Hz)
65536		8	104,9	9,54

CCPR1 - T - Toff - Ton					
CCPR1H	CCPR1L	Toff	Ton	T	
0xC0	0x00	75	25	100	(%)
49152		78,7	26,2	104,9	(mS)

En las salidas el tono se escucha en el buzzer **BUZ_1**, y el led **OUT_1** titila.

Más adelante para la práctica se muestran las gráficas copiadas del osciloscopio.

```
            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1       b'00000100'         ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1       b'00010000'         ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    bit_buzzer          .2                  ; bit buzzer, puerto B<2>
#define    _CCPR1H             0xC0                ; carga valor de comparación alto
#define    _CCPR1L             0x00                ; carga valor de comparación bajo

                       org     000h
Start:
;-------------------------------------------------------------
; inicialice puerto B

            clrf       PORTB               ; apague el puerto B
            bsf        PORTB,bit_buzzer    ; apague el buzzer, puerto B<2> = 1

;configura PORTB
            bsf        STATUS,RP0           ; selecciona Banco 1
            movlw      b'11001011'         ;RB7,6,3 = entradas ICSP
                                           ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf      TRISB
            bcf        STATUS,RP0          ; selecciona Banco 0
;-------------------------------------------------------------
```


libros.ep-electropc.com

```
; configura PORT C
            bsf     STATUS,RP0          ; Bank 1
            bcf     TRISC,2             ; haga CCP1 = RC2 = Output
            bcf     STATUS,RP0          ; Bank 0

; configura TIMER1
                    ; T1CON = /-/-/T1CKPS1/T1CKPS0/T1OSCEN/-T1SYNC/TMR1CS/TMR1ON
            clrf    T1CON               ; detiene TIMER1
;           clrf    TMR1H               ; borra el registro alto TMR1
;           clrf    TMR1L               ; borra el registro bajo TMR1
            movlw   b'00110000'         ; pone prescaler '11' = 1:8
            movwf   T1CON               ; guarda valor de prescaler

            bcf     T1CON,TMR1CS        ; fuente de clock al TIMER1, 1 = RC0 externo, 0 = CLKO =
Fosc/4 interno
; configura Ouput compare
                    ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3..CCPxM0/
            clrf    CCP1CON ; CCP Module is off

            movlw   _CCPR1H             ; carga valor de comparación alto
            movwf   CCPR1H              ; guarda valor de comparación alto

            movlw   _CCPR1L             ; carga valor de comparación bajo
            movwf   CCPR1L              ; guarda valor de comparación bajo

            movlw   b'00001000'         ; modo compare '1000' set output on match
;           movlw   b'00001001'         ; modo compare '1001' clr output on match
            movwf   CCP1CON             ; configura CCP1CON


            bsf     T1CON,TMR1ON        ; habilita o prenda TIMER1, 1 = ON, 0 = OFF
; configura interrupción TMR1

;-------------------------------------------------------------------------
Main_TMR1_OVF:
; check si hay overflow
            btfss   PIR1,TMR1IF         ; ¿hay overflow? Sí, salte
            goto    Main_CCP            ; no, haga ciclo

            clrf    CCP1CON             ; CCP Module is off
            movlw   b'00001000'         ; modo compare '1000' set output on match
            movwf   CCP1CON             ; configura CCP1CON

            bcf     PIR1,TMR1IF         ; apaga la bandera del overflow

            goto    Main_CPP_cod        ; haga toggle


Main_CCP:
; check si hay match CCP1
            btfss   PIR1,CCP1IF         ; ¿hay match de CCP1? Sí, salte
            goto Main_TMR1_OVF          ; no, haga ciclo

            nop                         ; necesario para apagar CCP1IF,
            nop                         ; cuando prescaler de TMR1 = > 1:8
            bcf     PIR1,CCP1IF         ; apaga la bandera de match CCP1
```


libros.ep-electropc.com

```
Main_CPP_cod
; toggle led OUT-1 y buzzer BUZ_1
            movf    PORTB,W             ; carga PORTB -> W
            xorlw   _toggle_OUT_1       ; filtro para toggle (alternar) led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1       ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB               ; pone W -> PORTB

            goto    Main_TMR1_OVF       ; haga ciclo

            end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\ COMPARE_1\ COMPARE_1.asm.

a) b)

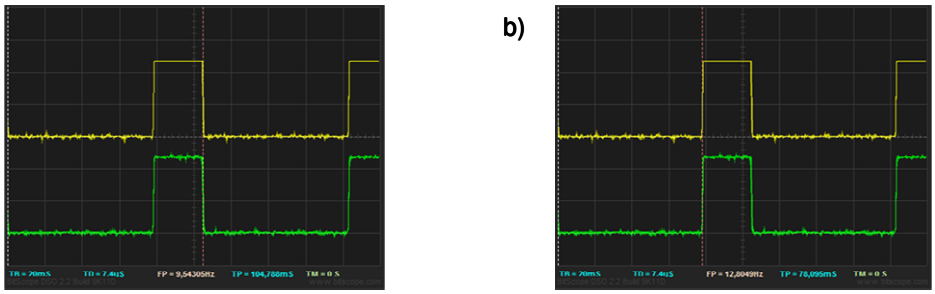


Figura 23.12 Gráficas de *compare por polling*.

En la Figura 23.12 a, b se muestra la medición de las señales en el pin **CCP1 = RC2** (superior), el periodo de la señal es de T = 104 ms, el **DE Toff = 78** ms aproximados, que es 75% del periodo; la frecuencia es de 9.5 Hz; de forma similar se observan las señales del led y el buzzer (inferior).

23.2.5.1.2 PRÁCTICA COMPARE POR INTERRUPCIÓN

El programa siguiente es igual al anterior, prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta PIC1F877A, con una frecuencia de 9.5 Hz, usando el módulo *compare*, procediendo con el método de interrupción. En las salidas el tono se escucha en el buzzer **BUZ_1**, y el led **OUT_1** titila. Más adelante se muestran las gráficas copiadas del osciloscopio para la práctica.

```
            list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc>             ; definiciones de variables específicas del procesador

__CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF
```


libros.ep-electropc.com

```
#define   _toggle_BUZ_1      b'00000100'      ; filtro para togglebuzzer RB2=BUZ_1= b'0000 0100'
#define   _toggle_OUT_1      b'00010000'      ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define   bit_buzzer         .2               ; bit buzzer, puerto B<2>
#define   _CCPR1H            0xC0             ; carga valor de comparación alto
#define   _CCPR1L            0x00             ; carga valor de comparación bajo

                org     000h                  ; vector de reset
                goto    start

                org     004h                  ; vector de interrupción
;---------------------------------------------------------------
ISR_TMR1
; check si hay overflow
          clrf  CCP1CON           ; CCP Module is off
          movlw b'00001000'       ; modo compare '1000' set output on match
          movwf CCP1CON           ; configura CCP1CON

          call  Toggle_Led_Buzz   ; toggle led y buzzer

          bcf   PIR1,TMR1IF       ; apaga la bandera del overflow

          retfie
;---------------------------------------------------------------
ISR_CCP1
; check si hay match CCP1
          call  Toggle_Led_Buzz   ; toggle led y buzzer

;         nop                     ; necesario para apagar CCP1IF,
;         nop                     ; cuando prescaler de TMR1 = > 1:8
          bcf   PIR1,CCP1IF       ; apaga la bandera de match CCP1

          retfie
;---------------------------------------------------------------
; toggle led OUT_1 y buzzer BUZ_1
Toggle_Led_Buzz
          bcf   STATUS,RP0        ; selecciona Banco 0
          movf  PORTB,W           ; carga PORTB -> W
          xorlw _toggle_OUT_1     ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
          xorlw _toggle_BUZ_1     ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
          movwf PORTB             ; pone W -> PORTB

          return
;---------------------------------------------------------------
;inicialice puerto B
Ini_pto
          clrf  PORTB             ; apague el puerto B
          bsf   PORTB,bit_buzzer  ; apague el buzzer, puerto B<2> = 1

;configura PORTB
          bsf   STATUS,RP0        ; selecciona Banco 1
          movlw b'11001011'       ;RB7,6,3 = entradas ICSP
                                  ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
          movwf TRISB
          bcf   STATUS,RP0        ; selecciona Banco 0
```


libros.ep-electropc.com

```
; configura PORT C
            bsf     STATUS,RP0          ; Bank 1
            bcf     TRISC,2             ; haga CCP1 = RC2 = Output
            bcf     STATUS,RP0          ; Bank 0

            return
;-----------------------------------------------------------
Ini_CCP1
; configura TIMER1
                    ; T1CON = /-/-/T1CKPS1/T1CKPS0/T1OSCEN/T1SYNC/TMR1CS/TMR1ON
            clrf    T1CON               ; detiene TIMER1
;           clrf    TMR1H               ; borra el registro alto TMR1
;           clrf    TMR1L               ; borra el registro bajo TMR1
            movlw   b'00110000'         ; pone prescaler '11' = 1:8
            movwf   T1CON               ; guarda valor de prescaler

            bcf     T1CON,TMR1CS        ; fuente de clock al TIMER1, 1 = RC0 externo, 0 = CLKO =
Fosc/4 interno
; configura Ouput compare
                    ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3..CCPxM0/
            clrf    CCP1CON             ; CCP Module is off

            movlw   _CCPR1H             ; carga valor de comparación alto
            movwf   CCPR1H              ; guarda valor de comparación alto

            movlw   _CCPR1L             ; carga valor de comparación bajo
            movwf   CCPR1L              ; guarda valor de comparación bajo

            movlw   b'00001000'         ; modo compare '1000' set output on match
;           movlw   b'00001001'         ; modo compare '1001' clr output on match
            movwf   CCP1CON             ; configura CCP1CON

; configura Interrupción TMR1
            bsf     INTCON,GIE          ; habilita interrupciones globales
            bsf     INTCON,PEIE         ; habilita interrupciones de periféricos

            bsf     STATUS,RP0          ; selecciona Banco 1
            bsf     PIE1,TMR1IE         ; habilita interrupción de TMR1
;           bsf     PIE1,CCP1IE         ; habilita interrupción de CCP1
            bcf     STATUS,RP0          ; selecciona Banco 0

            bsf     T1CON,TMR1ON        ; habilita o prenda TIMER1, 1 = ON, 0 = OFF

            return
;-----------------------------------------------------------
Start:
            call    Ini_pto             ; inicialice puerto
            call    Ini_CCP1            ; inicialice CCP1

            goto    $                   ; haga ciclo aquí

            end
```


libros.ep-electropc.com

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\ COMPARE_1\ COMPARE_1A.asm.

a) b)

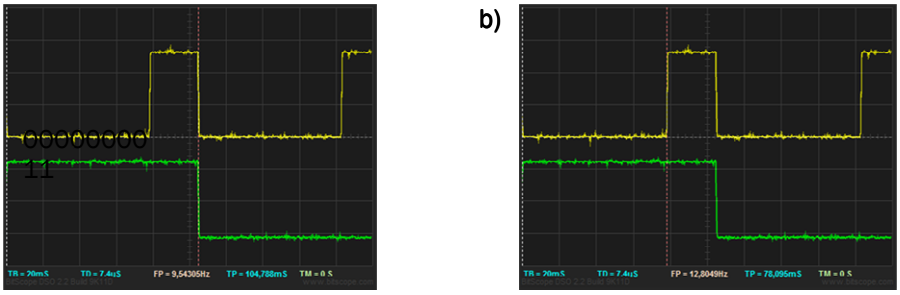


Figura 23.13 Gráficas de *compare* por interrupción.

En la Figura 23.13 se ve la medición de las señales en el pin **CCP1 = RC2** (superior), el periodo de la señal es de T = 104 ms, el del Toff = 78 ms aproximados, que es 75% del periodo, la frecuencia es de 9.5 Hz; de forma similar se observan las señales del led y el buzzer (inferior). Recuerde que la interrupción **CCP1IE**, cuando es habilitada la salida **CCP1 = RC2**, no se afectará; por eso la línea:

```
;               bsf         PIE1,CCP1IE          ; habilita interrupción de CCP1
```

se encuentra comentada.

Observe también cómo la señal de led y buzzer (inferior) solamente cambia cada vez que hay interrupción de TMR2.

23.3 MODO PWM (PULSE WIDTH MODULATOR)

El **modo de PWM** (*Pulse Width Modulation*), como su nombre lo indica, es un módulo que genera pulsos de ancho modulados, con una resolución de 10 bits, es decir, que permite una variación de 1 024 pasos, o sea de **00h a 3FFh**. Este módulo tiene aplicación especial en el control de velocidad de motores de CD.

23.3.1 DIAGRAMA CIRCUITAL

En la Figura 23.14 apreciamos el diagrama de bloques del módulo PWM.

libros.ep-electropc.com

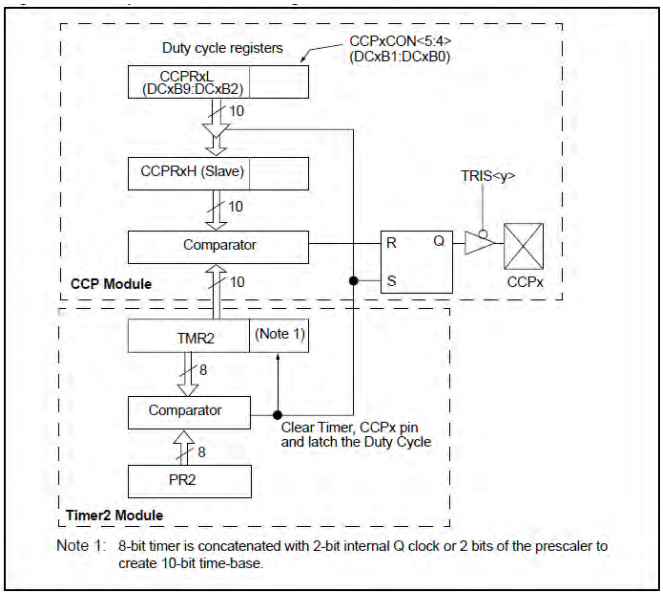


Figura 23.14 Diagrama de bloques del módulo PWM.

23.3.2 FUNCIONAMIENTO

En el **modo PWM**, el pin CCPx produce una salida de 10 bits de resolución de PWM. Así el pin CCPx es multiplexado con el latch de datos del puerto C; el bit correspondiente en el TRISC debe aclararse para que el pin CCPx sea una salida.

```
NOTA: Aclarando, el registro CCPxCON forzará al pin CCPx PWM del
latch de salida al nivel bajo (default). Este latch no es el del
PORTC I/O.
```

En la Figura 23.15 vemos el diagrama de una señal PWM, con un periodo que es el tiempo total de la salida y un *duty cycle* (DC = dureza o ciclo útil) que es el tiempo que la señal está en alto. La frecuencia es el inverso del periodo (1/periodo).

libros.ep-electropc.com

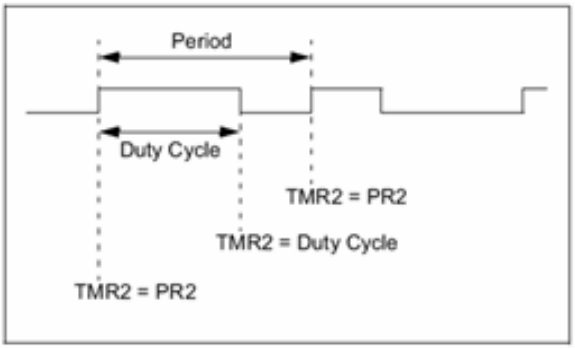


Figura 23.15 Periodo del PWM.

Periodo del PWM

El periodo del PWM se especifica al escribir en el registro PR2, y puede calcularse con la fórmula:

PWM período = [(PR2) + 1] • 4 • TOSC • (TMR2 valor de prescaler)
especificado en unidades de tiempo

Donde el PWM frecuencia (FPWM) se define como 1/[PWM periodo]. Cuando el TMR2 es igual a PR2, en el ciclo siguiente de instrucción pueden ocurrir los eventos siguientes:

1. TMR2 es aclarado (puesto en 00h)
2. El pin CCPx es seteado (puesto a uno, excepto si el *duty cicle* es = 0%, donde el pin CCPx puede no ser seteado)
3. El *duty cicle* es almacenado desde el CCPRxL al CCPRxH

NOTA: El *postscaler* del TMR2 no se usa en el cálculo de la frecuencia del PWM, pero sí puede usarse como salida para realimentar la tasa (*rate*) de frecuencia del PWM.

El duty cicle

El PWM *duty cicle* se especifica al escribir el registro CCPRxL bits DCx9:DCx2 y los bits DCxB1:DCxB0 en el registro CCPxCON<5:4>, para un total de 10 bits de resolución del PWM *duty cicle*. Para calcular el PWM *duty cicle* se usa la fórmula:

libros.ep-electropc.com

PWM *duty cycle* = (DCxB9:DCxB0) • Tosc • (TMR2 valor de *prescaler*)
especificado en unidades de tiempo

Los bits DCx9:DCx0 pueden escribirse en cualquier tiempo, pero el valor del *duty cicle* es almacenado en CCPRxH, sólo hasta que ocurra la coincidencia entre PR2 y TMR2 (esto es al final del periodo actual). En modo PWM, CCPRxH es un registro de sólo lectura. El registro CCPRxH y los 2 bits del latch interno se usan para un doble buffer del PWM *duty cicle*. Este doble buffer es esencial para la operación del PWM sin *glitches*.

Cuando el doble buffer, que contiene el registro CCPRxH y los 2 bits latch, coincide con el valor concatenado del TMR2 con los 2 bits internos Q *clock* (o los 2 bits del *prescaler* TMR2), el pin CCPx es aclarado (puesto a '0'), esto es el final de *duty cicle*. La resolución máxima del PWM (bits) para una frecuencia de PWM es:

$$\text{PWM resolución} = \frac{\log\left(\frac{FOSC}{FPWM}\right)}{\log(2)} \text{ bits}$$

NOTA: Si el PWM *duty cicle* es mayor que el periodo del PWM, el pin CCPx no será aclarado (puesto a '0'). Esto permite un *duty cicle* del 100%.

Resolución mínima

El mínimo de resolución (en tiempo) de cada bit del PWM *duty cicle* depende del *prescaler* del TMR2. Recuerde que la entrada del TMR2 es Fosc/4 = Tcy, debido a los 2 bits internos Q *clock* (o los 2 bits del *prescaler* TMR2), la resolución queda como muestra la Tabla 23.10.

Tabla 23.10 El mínimo de resolución (en tiempo) de cada bit del PWM.

Prescaler Value	T2CKPS1:T2CKPS0	Minimum Resolution (Time)
1	0 0	Tosc
4	0 1	Tcy
16	1 x	4 Tcy

En el ejemplo siguiente se dan el cálculo del PWM y la resolución necesaria para la frecuencia del PWM.

libros.ep-electropc.com

Desired PWM frequency is 78.125 kHz,
Fosc = 20 MHz
TMR2 prescale = 1

$1/78.125\ kHz = [(PR2) + 1] \cdot 4 \cdot 1/20\ MHz \cdot 1$

$12.8\ \mu s = [(PR2) + 1] \cdot 4 \cdot 50\ ns \cdot 1$

$PR2 = 63$

Find the maximum resolution of the duty cycle that can be used with a 78.125 kHz frequency and 20 MHz oscillator:

$1/78.125\ kHz = 2^{PWM\ RESOLUTION} \cdot 1/20\ MHz \cdot 1$

$12.8\ \mu s = 2^{PWM\ RESOLUTION} \cdot 50\ ns \cdot 1$

$256 = 2^{PWM\ RESOLUTION}$

$\log(256) = (PWM\ Resolution) \cdot \log(2)$

$8.0 = PWM\ Resolution$

En el ejemplo anterior, para obtener una frecuencia de PWM de 78.125 kHz, con una Fosc = 20 MHz, tenemos como resultado PR2 = 63, similar a 64d o 3Fh, que es el valor del perlodo. La resolución máxima para la frecuencia del PWM es 78.125 kHz, con Fosc = 20 MHz, que da como resultado 8 bits, o sea que con 8 bits se alcanza 2^8 = 256 o entre 0-255 valores de *duty cicle*; para valores mayores de 255 el *duty cicle* = 100%. La Tabla 23.11 es útil como referencia de valores para Fosc = 20 MHz.

Tabla 23.11 Resolución máxima para la frecuencia del PWM Fosc = 20 MHz.

PWM Frequency	1.22 kHz	4.88 kHz	19.53 kHz	78.12 kHz	156.3 kHz	208.3 kHz
Timer Prescaler (1, 4, 16)	16	4	1	1	1	1
PR2 Value	0xFF	0xFF	0xFF	0x3F	0x1F	0x17
Maximum Resolution (bits)	10	10	10	8	7	5.5

23.3.3 REGISTROS ASOCIADOS

En la Tabla 23.12 se encuentran los registros asociados CCP1CON y CCP2CON, y en la Tabla 23.13, los registros asociados con PWM y TIMWR2.

libros.ep-electropc.com

Tabla 23.12 Registros asociados CCP1CON y CCP2CON.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **CCPxX:CCPxY**: PWM Least Significant bits

Capture mode:
Unused.

Compare mode:
Unused.

PWM mode:
These bits are the two LSbs of the PWM duty cycle. The eight MSbs are found in CCPRxL.

bit 3-0 **CCPxM3:CCPxM0**: CCPx Mode Select bits

0000 = Capture/Compare/PWM disabled (resets CCPx module)
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, set output on match (CCPxIF bit is set)
1001 = Compare mode, clear output on match (CCPxIF bit is set)
1010 = Compare mode, generate software interrupt on match (CCPxIF bit is set, CCPx pin is unaffected)
1011 = Compare mode, trigger special event (CCPxIF bit is set, CCPx pin is unaffected); CCP1 resets TMR1; CCP2 resets TMR1 and starts an A/D conversion (if A/D module is enabled)
11xx = PWM mode

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Tabla 23.13 Registros asociados con PWM y TIMWR2.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by Capture and Timer1.
Note 1: The PSP is not implemented on 28-pin devices; always maintain these bits clear.

libros.ep-electropc.com

23.3.4 PROCEDIMIENTO DE USO

Los pasos necesarios para trabajar con el módulo CCP en el modo PWM se pueden generar mediante dos métodos:

- PWM por *polling*.
- PWM por interrupción.

Básicamente, en ambos métodos se trabaja con el módulo TIMER2, verificando la bandera de interrupción TMR2IF por *polling* y habilitando el bit TMR2IE para la interrupción. Los pasos para implementar el PWM por *polling* son:

1. Asigne el valor del periodo PWM en el registro PR2
2. Asigne el valor del *duty cicle* del PWM escribiendo en bits (DCx9:DCx0), o en registro CCPRxL (bits DCx9:DCx2) y en registro CCPxCON<5:4> (bits DCx1:DCx0).
3. Configure el módulo CCP para la operación PWM
4. Configure CCPx pin como salida
5. Asigne el *prescaler* a TMR2 y encienda el TMR2 con el bit TMR2ON en el registro T2CON.
6. Verifique la bandera de TMR2IF en el registro PIR1

Los mismos pasos anteriores con código son:

1. Asigne el valor del periodo PWM en el registro PR2:

```
clrf    CCP1CON         ; borra CCP1CON

movlw   0xFF            ; periodo máximo
movwf   PR2             ; guarda en PR2
```

2. Asigne el valor del *duty cicle* del PWM, escribiendo en bits (DCx9:DCx0), o en registro CCPRxL (bits DCx9:DCx2) y en registro CCPxCON<5:4> (bits DCx1:DCx0):

```
movlw   0x1F            ; 25% de duty cicle
movwf   CCPR1L          ; guarda en CCPR1L
```

3. Configure el módulo CCP para la operación PWM:

```
                        ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3/2/1/0/
movlw   b'00101100'     ; modo PWM y bit [DCx1:DCx0] en [1:1]
movwf   CCP1CON         ; configura CCP1CON
```

4. Configure CCPx pin como salida:

```
bsf     STATUS,RP0      ; Bank 1
bcf     TRISC,_bit_CCP1 ; configura pin de salida
```


libros.ep-electropc.com

```
    bcf     STATUS,RP0          ; Bank 0
```

5. Asigne el *prescaler* a TMR2 y encienda el TMR2 con el bit TMR2ON en el registro T2CON.

```
                                ; T2CON = /-/TOUTPS3/2/1/0/TMR2ON/T2CKPS1/0/
    clrf    T2CON               ; detiene TMR2, borra el registro TMR2
    movlw   b'00000011'         ; asigna postscaler = 1, prescaler = 16
    movwf   T2CON               ; configura T2CON

    bsf     T2CON,TMR2ON        ; prenda TMR2
```

6. Verifique la bandera de TMR2IF en el registro PIR1:

```
Main_TMR2_OVF:
; check si hay overflow
    btfss   PIR1,TMR2IF         ; ¿hay overflow? Sí, salte
    goto    Main_TMR2_OVF       ; no, haga ciclo

    bcf     PIR1,TMR2IF         ; apaga la bandera del overflow
```

Más adelante veremos una práctica de este modo PWM por *polling*. Para implementar el PWM por interrupción siga estos pasos:

1. Asigne el valor del periodo PWM en el registro PR2
2. Asigne el valor del *duty cicle* del PWM escribiendo en bits (DCx9:DCx0), o en registro CCPRxL (bits DCx9:DCx2) y en registro CCPxCON<5:4> (bits DCx1:DCx0).
3. Configure el módulo CCP para la operación PWM
4. Configure CCPx pin como salida
5. Asigne el *prescaler* a TMR2 y encienda el TMR2 con el bit TMR2ON en el registro T2CON.
6. Habilite interrupción global GIE, de periféricos PEIE en INTCON, habilite interrupción TMR2IE en PIE1.
7. Haga rutina de servicio de interrupción para PWM

Los mismos pasos anteriores con código son:

1. Asigne el valor del periodo PWM en el registro PR2:

```
    clrf    CCP1CON             ; borra CCP1CON

    movlw   0xFF                ; máximo periodo
    movwf   PR2                 ; guarda en PR2
```

2. Asigne el valor del *duty cicle* del PWM, escribiendo en bits (DCx9:DCx0), o en registro CCPRxL (bits DCx9:DCx2) y en registro CCPxCON<5:4> (bits DCx1:DCx0):

```
    movlw   0x1F                ; 25% de duty cicle
```


libros.ep-electropc.com

```
    movwf   CCPR1L          ; guarda en CCPR1L
```

3. Configure el módulo CCP para la operación PWM:

```
                            ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3/2/1/0/
    movlw   b'00101100'     ; modo PWM y bit [DCx1:DCx0] en [1:1]
    movwf   CCP1CON         ; configura CCP1CON
```

4. Configure CCPx pin como salida:

```
    bsf     STATUS,RP0      ; Bank 1
    bcf     TRISC,_bit_CCP1 ; configura pin de salida
    bcf     STATUS,RP0      ; Bank 0
```

5. Asigne el *prescaler* a TMR2 y encienda el TMR2 con el bit TMR2ON en el registro T2CON:

```
                            ; T2CON = /-/TOUTPS3/2/1/0/TMR2ON/T2CKPS1/0/
    clrf    T2CON           ; detiene TMR2, borra el registro TMR2
    movlw   b'00000011'     ; asigna postscaler = 1, prescaler = 16
    movwf   T2CON           ; configura T2CON

    bsf     T2CON,TMR2ON    ; prenda TMR2
```

6. Habilite interrupción global GIE, de periféricos PEIE en INTCON, habilite interrupción TMR2IE en PIE1:

```
    bsf     INTCON,PEIE     ; habilita interrupciones de periféricos
    bsf     INTCON,GIE      ; habilita interrupciones globales

    bsf     STATUS,RP0      ; Bank 1
    bsf     PIE1,TMR2IE     ; habilita interrupción de tmr2
    bcf     STATUS,RP0      ; Bank 0

    bsf     T2CON,TMR2ON    ; prenda TMR2
```

7. Haga rutina de servicio de interrupción para PWM:

```
; rutina de servicio de interrupción a PWM
ISR_PWM

                            ; código

    bcf     PIR1,TMR2IF     ; apaga la bandera de overflow

    retfie
```

Más adelante veremos una práctica de este modo PWM por interrupción.

23.3.5 PRÁCTICA

Haremos las prácticas para:

- PWM por *polling*.
- PWM por interrupción.

libros.ep-electropc.com

23.3.5.1 MONTAJE

Para el montaje de las prácticas siguientes se necesitan el depurador **PICkit 2** y la tarjeta de evaluación **PIC16F8XX**, y un osciloscopio (opcional). La foto siguiente es del montaje utilizado en las prácticas, tal como muestra la Figura 23.16.

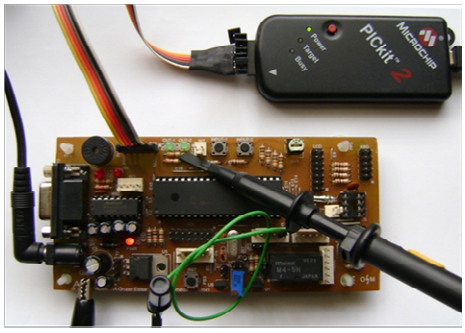


Figura 23.16 Prácticas de montaje PWM.

23.3.5.1.1 PRÁCTICA PWM POR POLLING

El programa siguiente prende y apaga el led **OUT_1** y el buzzer **BUZ_1** de la tarjeta PIC1F877A, con una frecuencia de 1.22 kHz generada con el módulo PWM. Procediendo con el método de *polling*. Si tenemos un Xtal = 20 MHz, Tosc = 0.05 µs, con PR2 = 0×FF, *prescaler* = '11' = 16, y un *duty cicle* deseado de ⅛, calculando tendremos:

PWM periodo = [(PR2) + 1] • 4 • TOSC • (TMR2 valor de prescaler)
PWM periodo = [(255) + 1] • 4 • 0.05 µs • (16) = 819.2 µs

PWM duty cycle = (DCxB9:DCxB0) • Tosc • (TMR2 valor de prescaler)
819.2 µs * (1/8) = (DCxB9:DCxB0) * 0.05 µs * 16
(DCxB9:DCxB0) = 128, dividiendo por 4 (bits DCx1:DCx0) = > 32, o sea 0×1F

PWM resolución = log(Fosc/Fpwm)/log(2) bits
PWM resolución = log(Tpwm/(Tosc * Prescaler))/log(2) bits
PWM resolución = log(20 MHz/(1.22 kHz * 16))/log(2) bits = 10

libros.ep-electropc.com

Tabla 23.14 Cálculos PWM.

Fosc	XTAL - TCY
XTAL	Tosc
(MHz)	(uS)
20	0,05

PR2 - TMR2 - PRESCALER				
PR2	TMR2	prescaler	T	Fcia
0xFF	0xFF	11	(uS)	(KHz)
255	256	16	819,2	1,221

DC - T - Ton - Toff				
DC	DCx9:DCx0	Ton	Toff	T
	0x1F	(mS)	(mS)	(mS)
1/8	32	102,4	716,8	819,2

La señal PWM de salida es de periodo T = 819.2 µs, con un DC = ⅛ que da un Ton = 102.4 µs. En las salidas el tono se escucha en el buzzer **BUZ_1** y el led **OUT_1** titila. Más adelante se muestran las gráficas copiadas del osciloscopio para la práctica.

```
            list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1      b'00000100'        ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1      b'00010000'        ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    _bit_buzzer        2                  ; bit buzzer, puerto B<2>
#define    _bit_CCP1          2                  ; bit de módulo CCP1

                   org        000h
Start:
;-------------------------------------------------------------------------
; inicialice puerto B
            clrf       PORTB                    ; apague el puerto B
            bsf        PORTB,_bit_buzzer        ; apague el buzzer, puerto B<2> = 1

;configura PORTB
            bsf        STATUS,RP0          ; selecciona Banco 1
            movlw      b'11001011'         ; RB7,6,3 = entradas ICSP
                                           ; RB1,0 = entradas switches, RB = 5,4,2 salidas leds y buzzer
            movwf      TRISB
            bcf        STATUS,RP0          ; selecciona Banco 0
```


libros.ep-electropc.com

```
;-------------------------------------------------------------------
; configura CCP1
        clrf    CCP1CON         ; borra CCP1CON

        movlw   0xFF            ; máximo periodo
        movwf   PR2             ; guarda en PR2

        movlw   0x1F            ; 25% de duty cicle
        movwf   CCPR1L          ; guarda en CCPR1L

                                ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3/2/1/0/
        movlw   b'00101100'     ; modo PWM y bit [DCx1:DCx0] en [1:1]
        movwf   CCP1CON         ; configura CCP1CON

        bsf     STATUS,RP0      ; Bank 1
        bcf     TRISC,_bit_CCP1 ; configura pin de salida
        bcf     STATUS,RP0      ; Bank 0


; configura CCP1 y TMR2
                                ; T2CON = /-/TOUTPS3/2/1/0/TMR2ON/T2CKPS1/0/
        clrf    T2CON           ; detiene TMR2, borra el registro TMR2
        movlw   b'00000011'     ; asigna postscaler = 1, prescaler = 16
        movwf   T2CON           ; configura T2CON

        bsf     T2CON,TMR2ON    ; prenda TMR2

; configura Interrupción TMR2

;-------------------------------------------------------------------
Main_TMR2_OVF:
; check si hay overflow
        btfss   PIR1,TMR2IF     ; ¿hay overflow? Sí, salte
        goto    Main_TMR2_OVF   ; no, haga ciclo

        bcf     PIR1,TMR2IF     ; apaga la bandera del overflow

Main_CPP_cod
; toggle led OUT-1 y buzzer BUZ_1
        bcf     STATUS,RP0      ; selecciona Banco 0
        movf    PORTB,W         ; carga PORTB -> W
        xorlw   _toggle_OUT_1   ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
        xorlw   _toggle_BUZ_1   ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
        movwf   PORTB           ; pone W -> PORTB

        goto    Main_TMR2_OVF   ; haga ciclo

 end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\PWM_1\PWM_1.asm.

libros.ep-electropc.com

a) b)

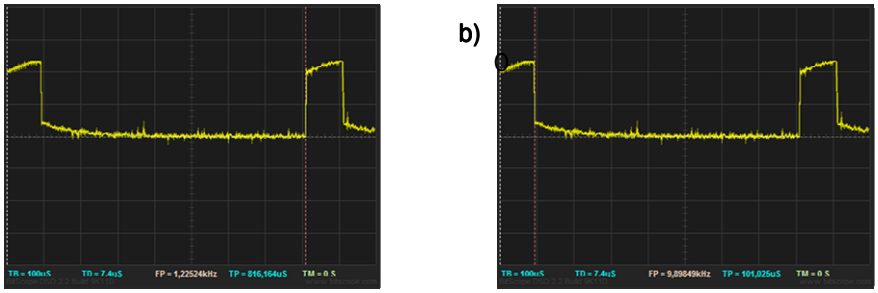


Figura 23.17 Gráficas de PWM por polling.

En la Figura 23.17a, b, c se muestra la medición de las señales en el pin CCP1 = RC2, el periodo de la señal es de T = 816 µs, y el del Ton = 101 µs aproximados, que es ⅛ del periodo; la frecuencia es de 1.22 kHz.

23.3.5.1.2 PRÁCTICA PWM POR INTERRUPCIÓN

El programa siguiente es el mismo que en el ejemplo anterior, procediendo con el método de interrupción. La interrupción se genera al finalizar el periodo, pero no se genera cuando hay *duty cicle*. La señal PWM de salida es de un periodo T = 819.2 µs, con un DC = ⅛ que da un Ton = 102.4 µs. En las salidas el tono se escucha en el buzzer **BUZ_1**, y el led **OUT_1** titila. Más adelante se muestran las gráficas copiadas del osciloscopio para la práctica.

```
            list p = 16f877a        ; directiva para definir procesador
#include <p16f877a.inc>             ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _toggle_BUZ_1        b'00000100'        ; filtro para toggle buzzer RB2=BUZ_1=b'0000 0100'
#define    _toggle_OUT_1        b'00010000'        ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
#define    _bit_buzzer          2                  ; bit buzzer, puerto B<2>
#define    _bit_CCP1            2                  ; bit de módulo CCP1

                    org         0000h              ; vector de reset
                    goto        start

                    org         0004h              ; vector de interrupciones
;-------------------------------------------------------------------------
; rutina de servicio de interrupción a PWM
ISR_PWM

            bcf STATUS,RP0 ; selecciona Banco 0
            movf    PORTB,W             ; carga PORTB -> W
            xorlw   _toggle_OUT_1       ; filtro para toggle led RB4 = _OUT_1 = b'0001 0000'
            xorlw   _toggle_BUZ_1       ; filtro para toggle buzzer RB2 = BUZ_1 = b'0000 0100'
            movwf   PORTB               ; pone W -> PORTB
```


libros.ep-electropc.com

```
            bcf     PIR1,TMR2IF     ; apaga la bandera del overflow

            retfie
;-----------------------------------------------------------------
; inicialice puerto B
Ini_pto
            clrf    PORTB                   ; apague el puerto B
            bsf     PORTB,_bit_buzzer       ; apague el buzzer, puerto B<2> = 1

;configura PORTB
            bsf     STATUS,RP0      ; selecciona Banco 1
            movlw   b'11001011'     ; RB7,6,3 = entradas ICSP
                                    ; RB1,0=entradas switches, RB=5,4,2 salidas leds y buzzer
            movwf   TRISB
            bcf     STATUS,RP0      ; selecciona Banco 0

            return
;-----------------------------------------------------------------
; configura CCP1 en modo PWM y TMR2
Ini_PWM
            clrf    CCP1CON         ; borra CCP1CON

            movlw   0xFF            ; máximo periodo
            movwf   PR2             ; guarda en PR2

            movlw   0x1F            ; 25% de duty cicle
            movwf   CCPR1L          ; guarda en CCPR1L

                                    ; CCP1CON = /-/-/CCPxX/CCPxY/CCPxM3/2/1/0/
            movlw   b'00101100'     ; modo PWM y bit [DCx1:DCx0] en [1:1]
            movwf   CCP1CON         ; configura CCP1CON

            bsf     STATUS,RP0      ; Bank 1
            bcf     TRISC,_bit_CCP1 ; configura pin de salida
            bcf     STATUS,RP0      ; Bank 0

; configura CCP1 y TMR2
                                    ; T2CON = /-/TOUTPS3/2/1/0/TMR2ON/T2CKPS1/0/
            clrf    T2CON           ; detiene TMR2, borra el registro TMR2
            movlw   b'00000011'     ; asigna postscaler = 1, prescaler = 16
            movwf   T2CON           ; configura T2CON

; configura interrupción TMR2
            bsf     INTCON,PEIE     ; habilita interrupciones de periféricos
            bsf     INTCON,GIE      ; habilita interrupciones globales

            bsf     STATUS,RP0      ; Bank 1
            bsf     PIE1,TMR2IE     ; habilita interrupción de TMR2
            bcf     STATUS,RP0      ; Bank 0

            bsf     T2CON,TMR2ON    ; prenda TMR2

            return
;-----------------------------------------------------------------
Start:
```

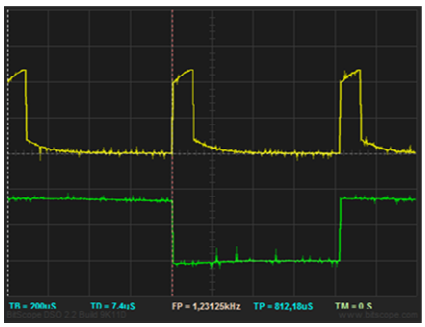

libros.ep-electropc.com

```
        call    Ini_pto         ; inicialice puerto B
        call    Ini_PWM         ; inicialice PWM

        goto    $               ; haga ciclo

end
```

Este ejemplo está en nuestra página Web, en la carpeta X:\ Ejemplos_ASM\, y se debe copiar al C:\ para quedar como C:\ Ejemplos_ASM\\PWM_1\PWM_1A.asm.

a) 

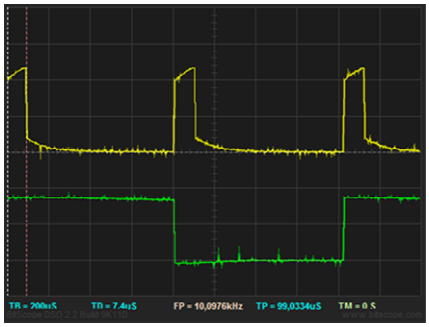
b) 

Figura 23.18 Gráficas de PWM por interrupción.

En la Figura 23.18a, b vemos la medición de las señales en el pin CCP1 = RC2 y en las salidas del led **OUT_1** o el buzzer **BUZ_1**. El periodo de la señal es de T = 812 µs, y el Ton = 99 µs aproximados, que es ⅛ del periodo, la frecuencia es 1.22 kHz (en las gráficas la señal superior). La salida hacia el led **OUT_1** o el buzzer BUZ_ (es la señal inferior) tiene un cambio cada vez que hay interrupción por TMR2.

EVALUACIÓN

```
P.: 1. ¿Cuáles son los modos del módulo CCP?
R.:
Capture/compare/PWM.

P.: 2. ¿Cuántos módulos CCP existen?
R.:
2, CCP1 y CCP2.

P.: 3. ¿Qué timer asociado y cuántos bits de manejo tienen los
modos CCP?
R.:
Capture      Timer1 16 bits
Compare      Timer1 16 bits
PWM   Timer2 10 bits
```


libros.ep-electropc.com

P.: 4. ¿En qué consiste el modo capture?
R.:
El modo de captura consiste en capturar el temporizador cuando ocurre un evento en el pin de entrada. Así podemos medir el tiempo transcurrido entre eventos causados en el pin, con la precisión que deseemos.

P.: 5. ¿En cuáles registros se guarda una captura?
R.:
En los pares de registros CCPR1H:CCPR1L o CCP21H:CCPR2L.

P.: 6. ¿En qué consiste el modo compare?
R.:
El modo de comparación consiste en generar una salida cuando ocurre una coincidencia entre el temporizador y el registro del comparador. Así generamos una señal de salida reloj (*clock*) muy precisa.

P.: 7. ¿En cuáles registros se guarda una comparación?
R.:
En los pares de registros CCPR1H:CCPR1L o CCPR2H:CCPR2L.

P.: 8. ¿En qué consiste el modo PWM?
R.:
El modo de PWM (*Pulse Width Modulation*), como su nombre lo indica, es un módulo que genera pulsos de ancho modulados.

libros.ep-electropc.com

libros.ep-electropc.com

PARTE 7

MODULOS DE COMUNICACIONES: SPI, I²C Y SCI O USART

En esta Parte 7 de comunicaciones veremos varios tipos de módulos que contiene el MCU para intercambiar información con otros dispositivos que están fuera de él. Veremos también cómo el módulo MSSP contiene el hardware para comunicaciones seriales de los protocolos SPI e I2C, que son protocolos standard de comunicaciones on board, o sea dentro de la misma tarjeta. Luego veremos el módulo USART que contiene el hardware para comunicación serial del protocolo RS232, el cual es un protocolo standard de comunicaciones ampliamente difundido, y usado para transferencia de información con computadoras. Otra de las formas es el protocolo RS485, que también se puede implementar con este módulo.

24.0 MÓDULO PUERTO SERIAL SINCRÓNICO MAESTRO (MSSP)

En este capítulo veremos que el módulo MSSP es una interfaz serial con otros periféricos o microcontroladores, y contiene los modos SPI e I2C. Los dispositivos con los que se comunica pueden ser: memorias seriales EEPROM, registros de rotación, drivers de display y conversores A/D, entre otros.

libros.ep-electropc.com

24.1 MÓDULO PUERTO SERIAL SINCRÓNICO MAESTRO (MSSP)

El módulo MSSP es una interfaz serial con otros periféricos o microcontroladores que pueden ser memorias seriales EEPROM, registros de rotación, drivers de display y conversores A/D, entre otros. Este módulo puede operar en dos modos:

1. *Serial Peripheral Interface* (**SPI**)
2. *Inter-Integrated Circuit* (**I²C**)
    1. *Full Master Mode*
    2. *Slave Mode*

El I²C soporta los modos en hardware siguientes:

1. Modo maestro
2. Modo multimaestro
3. Modo esclavo

El módulo MSSP tiene tres registros asociados:

1. SSPSTAT registro de estado
2. SSPCON registro de control
3. SSPCON2 registro de control

El uso de estos registros y sus configuraciones de bits individuales difieren significativamente, según el módulo operado: SPI o I²C. Los detalles adicionales se explican en los capítulos siguientes.

24.2 MODO INTERFAZ PERIFÉRICA SERIAL (SPI)

Antes de ver el módulo MSSP en el modo SPI, veremos el protocolo SPI.

24.2.1 PROTOCOLO SPI

El **SPI** es un protocolo de comunicación sincrónica para intercambio de datos. No es un estándar de algún propietario y, por tanto, cada fabricante lo acomoda a sus dispositivos. Por ejemplo, **Microchip** y **Freescale** trabajan con datos de 8 bits, Texas Instruments y Atmel con 16 bits de ancho de datos, entre otros. Los rasgos principales, a los que todos los fabricantes se acogen, son los que veremos a

libros.ep-electropc.com

continuación. La comunicación SPI se realiza entre dos dispositivos, uno **maestro** y otro **esclavo**, como muestra la Figura 24.1. Los pines son definidos como:

- **SCK** (Serial Clock)
- **MOSI** (Master Ouput Slave Input)
- **MISO** (Master Input Slave Output)
- (Slave Select)

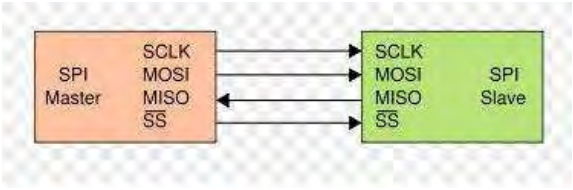


Figura 24.1 SPI conexión estándar maestro-esclavo.

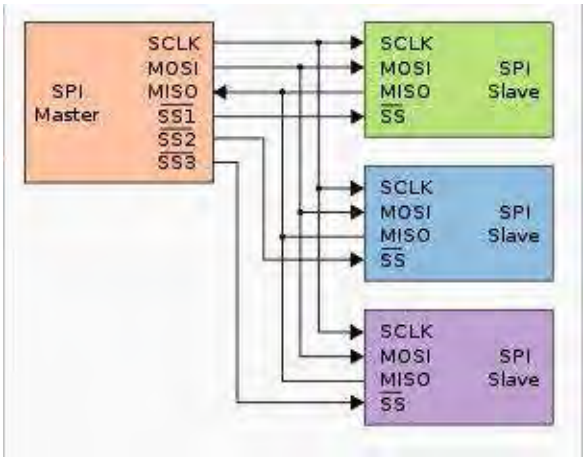


Figura 24.2 SPI conexión estándar maestro-esclavos.

En la conexión con múltiples esclavos existen varios pines de control en el **maestro**, llamados –SS1, -SS2, -SS3 que van conectados al pin de control del dispositivo esclavo (Figura 24.2). Para este caso los pines están definidos como el fabricante **Microchip** los define; en la Figura 24.3 se enseña una conexión típica entre dos microcontroladores **Microchip**; el **procesador 1** actúa como **maestro** y el **procesador 2** como **esclavo**. Los pines utilizados son:

libros.ep-electropc.com

- **Clock serial** o *Serial Clock* (**SCK**)
- **Dato serial de salida** o *Serial Data Out* (**SDO**)
- **Dato serial de entrada** o *Serial Data In* (**SDI**)
- **Selección de esclavo** o *Slave Select* ()

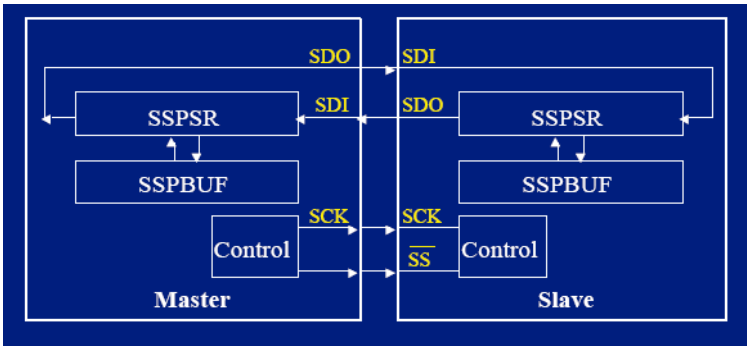


Figura 24.3 Conexión SPI con -SS.

Observe que son las mismas conexiones con nombres diferentes.

24.2.1.1 CLOCK (SCK)

La **señal de reloj** es provista y manejada únicamente por el **maestro** para proveer la sincronización, e indica cuándo un dato puede cambiar y cuándo es válido para lectura. La velocidad de transmisión puede cambiar, y depende solamente del reloj, lo cual es ventajoso para micros que incluso tengan osciladores **RC**. Para calcular la velocidad límite (Tabla 24.1 y Figura 24.4) añadiremos una gráfica de las especificaciones eléctricas que puede encontrar en el pdf **PIC16F87XA DS39582B**-página 188.

Tabla 24.1 Tiempos límite SPI.

Param No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
70*	TssL2scH, TssL2scL	$\overline{SS}$ ↓ to SCK ↓ or SCK ↑ Input	Tcy	—	—	ns	
71*	TscH	SCK Input High Time (Slave mode)	Tcy + 20	—	—	ns	
72*	TscL	SCK Input Low Time (Slave mode)	Tcy + 20	—	—	ns	
73*	TdiV2scH, TdiV2scL	Setup Time of SDI Data Input to SCK Edge	100	—	—	ns	
74*	TscH2diL, TscL2diL	Hold Time of SDI Data Input to SCK Edge	100	—	—	ns	

libros.ep-electropc.com

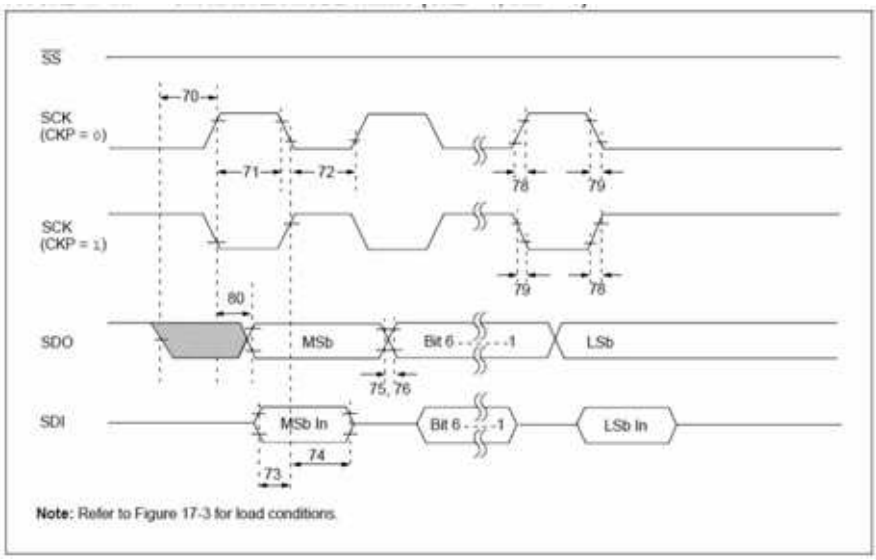


Figura 24.4 Diagrama de tiempos límite SPI.

24.2.1.2 DATOS SDI Y SDO

Cuando un dato es sincronizado saliendo, también se sincroniza un nuevo dato entrando en el mismo dispositivo. Cuando se transmite un dato, el dato entrante debe ser leído antes de transmitir de nuevo. Si el dato entrante no es leído (así no se use), el dato será perdido y como resultado el módulo SPI puede ser deshabilitado.

```
NOTA: Siempre debe leerse el dato luego que una transferencia se
hace, aunque el dato no lo use en la aplicación.
```

Los datos son intercambiados entre dispositivos. No hay un dispositivo que sea un solo "transmisor" o un solo "receptor" en SPI, cada dispositivo tiene una línea para entrada de datos y una línea para salida de datos. El intercambio de datos es controlado por la línea de reloj **SCK** que es controlada por el **maestro**. Un dato es entregado (*out*) durante un flanco del reloj **SCK**. Un dato es leído capturado (*in*) durante el flanco opuesto del reloj **SCK**. El flanco (de subida o bajada) del opuesto se usa para asegurar que el dato es válido durante el tiempo de lectura. En la Figura 25.5 se observa cómo los pulsos de reloj SCK se producen en el flanco de bajada, donde el dato **SDO** de salida cambia, y en el reloj **SCK** opuesto, en el flanco de subida, se lee el dato **SDI**.

libros.ep-electropc.com

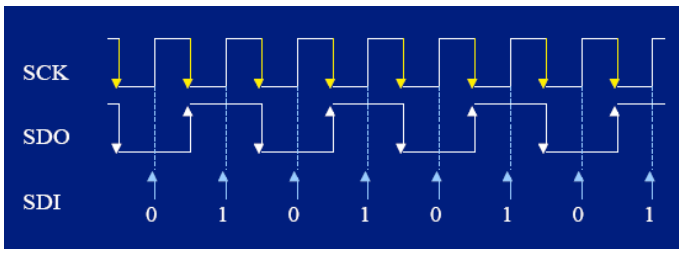


Figura 24.5 Ejemplo SPI, modo 1,1

24.2.1.3 CS O SS

A menudo, una señal controla cuando un dispositivo es accedido; esta señal se denomina *Chip Select* (CS) o *Slave Select* (), se activa frecuentemente en bajo, y se usa cuando hay más de un esclavo en el sistema, pero puede ser opcional cuando solamente hay un esclavo. Como regla general, debe usarse. Se usa comúnmente para mejorar la inmunidad al ruido. Esta función resetea el esclavo, que así queda listo para recibir el byte siguiente.

24.2.1.4 MODOS

Hay cuatro modos de funcionamiento según el tipo de reloj, que se detallan más adelante (Figura 24.6).

SPI Mode	CKP	CKE
0,0	0	1
0,1	0	0
1,0	1	1
1,1	1	0

Figura 24.6
Modos de Clock SPI.

24.2.2 MÓDULO MSSP EN MODO SPI

El modo SPI permite que sean sincrónicamente transmitidos y recibidos de manera simultánea 8 bits de datos. Todos los 4 modos de SPI son soportados. Típicamente se usan tres pines:

- Serial Data Out (SDO) – RC5/SDO
- Serial Data In (SDI) – RC4/SDI/SDA
- Serial Clock (SCK) – RC3/SCK/SCL

libros.ep-electropc.com

Además puede usarse un cuarto pin en **modo de esclavo** (*Slave*): *Slave Select* () – RA5/AN4//C2OUT. Para el modo de operación SPI, el módulo MSSP tiene cuatro registros:

1. MSSP Registro de Control SSPCON1
2. MSSP Registro de Status SSPSTAT
3. Registro de Buffer Serial Receptor/Transmisor SSPBUF
4. MSSP Registro de Movimiento SSPSR (no accesible directamente)

El **SSPCON** y el **SSPTAT** son el registro de **control y estado** en el modo de operación SPI. Los registros son de lectura y escritura. En el registro SSPSTAT, los bits <5:0> son de sólo lectura y los bits **<7:6>** son de lectura y escritura. El registro **SSPSR** se usa para mover los datos entrantes o salientes, y el registro **SSPBUF** es desde donde son leídos o escritos. En la operación de recepción los registros SSPSR y SSPBUF unidos crean un doble buffer de recepción. Cuando el SSPSR recibe un byte completo es transferido al SSPBUF, y el bit de interrupción SSPIF es puesto a uno. Durante la operación de transmisión el registro SSPBUF no se comporta como un doble buffer, la escritura se realiza en ambos registros: SSPBUF y SSPSR.

24.2.3 DIAGRAMA CIRCUITAL

En la Figura 24.7 se muestra el diagrama circuital del módulo SPI.

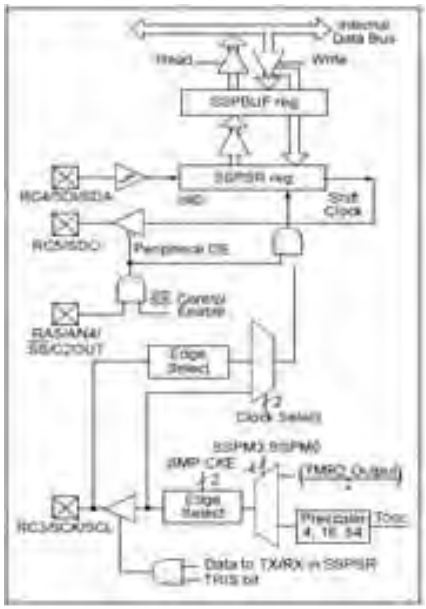


Figura 24.7 Diagrama circuital.

libros.ep-electropc.com

24.2.4 FUNCIONAMIENTO

24.2.4.1 INICIALIZANDO

Cuando inicializamos el SPI se necesita especificar varias opciones. Esto se logra programando los bits apropiados de control (SSPCON<5:0> y SSPSTAT<7:6>). Estos bits de control permiten especificar lo siguiente:

- Modo maestro (SCK es reloj de salida)
- Modo esclavo (SCK es reloj de entrada)
- Polaridad del *clock* (estado del reloj SCK)
- Fase de muestreo del dato de entrada (mitad o final del tiempo del dato de salida)
- Flanco del reloj (dato de salida sobre el flanco de subida/bajada del SCK)
- Tasa (*rate*) de reloj (solamente modo maestro)
- Modo de selección de esclavo (solamente modo esclavo)

24.2.4.2 HABILITANDO SPI I/O

Para habilitar el puerto serial, el bit de habilitación SSP en el registro SSPCON1<5> debe ser puesto a uno; para borrar o reconfigurar el modo SPI se borra el bit SSPEN; se reinicializan los registros en SSPCON1 y entonces se pone a uno el bit SSPEN. Esto configura los pines SDI, SDO, SCK y -SS como pines del puerto serial. Para que se comporten como funciones del puerto serial, algunos deben tener sus bits de datos de dirección en el TRIS programados apropiadamente; o sea que:

- SDI es automáticamente controlado por el módulo SPI
- SDO debe tener el bit TRISC<5> = 0
- SCK (modo maestro) debe tener el bit TRISC<3> = 0
- SCK (modo esclavo) debe tener el bit TRISC<3> = 1
- debe tener el bit TRISA<5> = 1 (solamente en modo esclavo)

Cualquier función de puerto serial que no se desee puede ser sobrescrita, programando el correspondiente bit del dato de dirección en el registro TRIS al valor opuesto.

libros.ep-electropc.com

24.2.4.3 CONEXIÓN

En la Figura 24.8 se indica una conexión típica entre dos microcontroladores: el Procesador 1 actúa como **maestro** y el **procesador 2**, como **esclavo**.

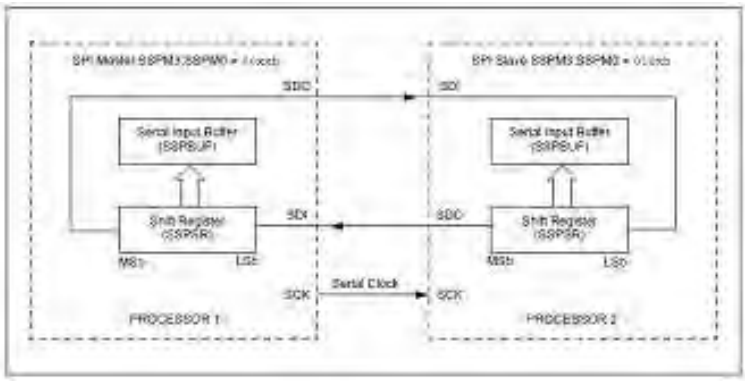


Figura 24.8 Conexión SPI master/slave.

El **procesador 1 maestro** inicia la transferencia de datos, enviando una señal SCK. El dato es llevado afuera del registro de movimiento *shift register* sobre el flanco de reloj programado, y es leído o asegurado sobre el flanco de reloj opuesto. Ambos procesadores deben ser programados sobre la misma polaridad de reloj *clock polarity* (CKP). Entonces ambos procesadores enviarán y recibirán el dato al mismo tiempo. Ya sea que el dato sea importante o no (*dummy data*), eso depende de la aplicación de software. Esto nos lleva a tres escenarios posibles:

1.	Maestro envía dato	Esclavo envía *dummy data*
2.	Maestro envía dato	Esclavo envía dato
3.	Maestro envía *dummy data*	Esclavo envía dato (*dummy data* dato no importante)

24.2.4.4 EN OPERACIÓN

El **módulo MSSP** consiste en un registro de movimiento de transmisión/recepción SSPSR, y un registro buffer SSPBUF. El registro SSPSR mueve los datos entrantes o salientes del dispositivo, **MSB** (*Most Significant Bit*) primero. El SSPBUF mantiene el dato que se escribió al SSPSR hasta que el dato recibido está listo. Una vez que los 8 bits del dato han sido recibidos, el byte es movido al registro SSPBUF. Entonces el bit de detección de *Buffer Full* (**BF**), en el registro SSPSTAT<0>, y el bit de bandera de interrupción SSPIF, en el registro PR1<3> son puestos a uno. El doble buffer del dato recibido SSPBUF permite al byte siguiente comenzar la recepción antes que sea leído el dato que se acabó de recibir.

libros.ep-electropc.com

Cualquier escritura al registro SSPBUF durante una transmisión o una recepción de datos será ignorada, y el bit de *Write Collision* (**WCOL**) en el registro SSPCON<7> será puesto a uno. El software del usuario debe borrar el bit WCOL, para que pueda determinar si la escritura siguiente al registro SSPBUF se completó exitosamente. Cuando el software de aplicación está esperando recibir un dato válido, el registro SSPBUF debe ser leído antes que el dato siguiente sea transferido escribiendo el SSPBUF. El bit **BF** (*Buffer Full*) indica cuándo el SSPBUF ha sido cargado con el dato recibido (transmisión completa). Una vez que el SSPBUF es leído, el bit BF es borrado. Este dato puede ser irrelevante si el SPI es solamente transmisor.

La interrupción del MSSP se usa generalmente cuando la transmisión/recepción se completa. El SSPBUF debe ser leído o escrito. Si no se está usando el método de interrupción, puede hacerse preguntando para asegurar que no ocurra una colisión de escritura. El **registro SSPSR** no es directamente de lectura o de escritura, y puede ser accesible, direccionando el **registro SSPBUF**. El ejemplo siguiente muestra cómo cargar el SSPBUF (SSPSR) para transmisión, cuando se está recibiendo un dato.

```
LOOP    BTFSS   SSPSTAT, BF     ; Dato ha sido recibido (transmission complete)?
        BRA     LOOP ; No
        MOVF    SSPBUF, W       ; carga SSPBUF -> W
        MOVWF   RXDATA          ; guarde dato en RAM
        MOVF    TXDATA, W       ; carga TXDATA -> W
        MOVWF   SSPBUF          ; guarde W -> SSPBUF
```

24.2.4.5 MODOS DEL BUS SPI

La **polaridad de reloj CKP** se selecciona con la programación apropiada en el bit CKP del registro SSPCON1<4>, junto con el bit CKE del registro SSPSTAT<6>; en la Figura 24.9 se observan los diferentes modos de SPI, y en la Figura 24.10, la formación de cada uno de estos modos.

MODOS SPI BUS		
Standard Modo SPI	CKP	CKE
0,0	**0**	**1**
0,1	0	0
1,0	1	1
1,1	**1**	**0**

Figura 24.9 Modos de *clock* SPI

libros.ep-electropc.com

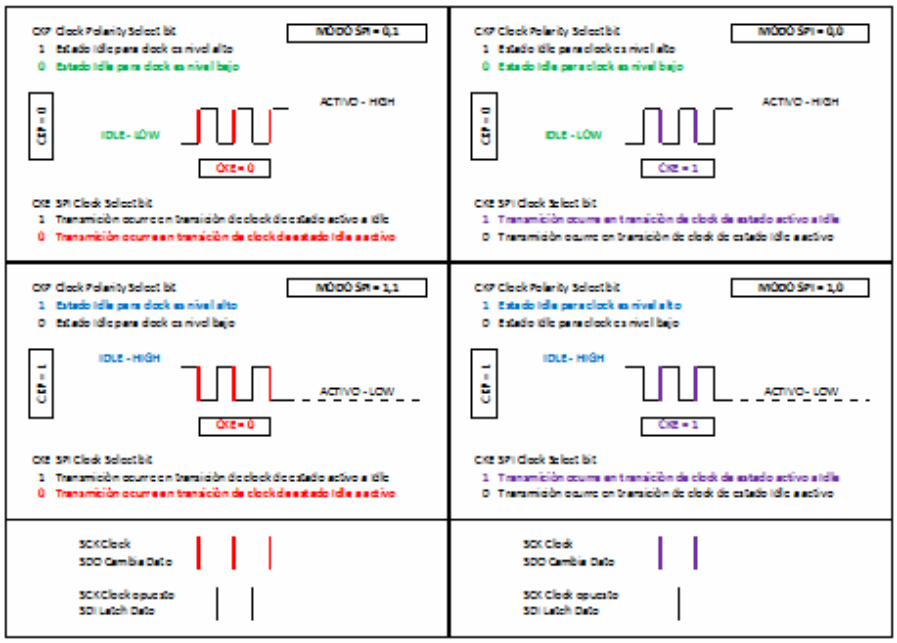


Figura 24.10 Formación modos SPI.

24.2.4.6 MODO MAESTRO

El **modo maestro** puede iniciar la transferencia de datos en cualquier tiempo, porque es el que controla el SCK; además determina cuándo el esclavo transmite datos por el protocolo de software. En la Figura 24.11 apreciamos sus diagramas de tiempo y formas de onda. En el modo maestro (Figura 24.11), el dato es transmitido o recibido tan pronto como el registro SSPBUF es escrito. Si el SPI solamente va a recibir, la salida SDO debe ser deshabilitada (programada como entrada). El registro SSPSR continuará moviendo hacia adentro la señal presente sobre el pin SDI a la tasa de reloj programada. Tan pronto como cada byte es recibido, puede ser cargado al registro SSPBUF como un byte normal es recibido (los bits de Interrupciones y Status apropiadamente puestos). Esto podría ser útil en aplicaciones de recepción, como un **monitor de actividad de línea**.

libros.ep-electropc.com

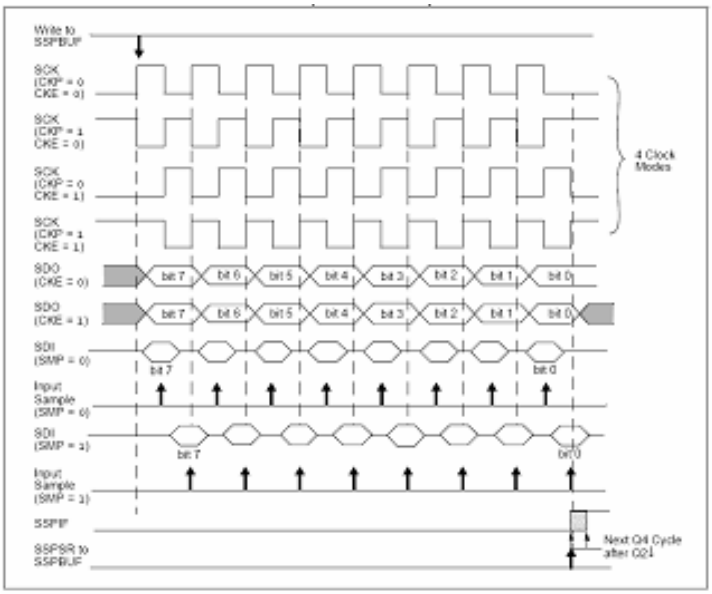


Figura 24.11 Diagrama de tiempos de SPI modo maestro.

El bit MSB es el primero en transmitirse en cualquier modo. Con los bits SSPM3:SSPM0 en el registro SSPCON1<3:0> se puede seleccionar la velocidad de transmisión de datos, así:

1. Fosc/4 o Tcy
2. Fosc/16 o 4 * Tcy
3. Fosc/64 o 16 * Tcy
4. Timer2 *output*/2

Esto permite un máximo de velocidad (a 40 MHz) de 10 Mbps.

24.2.4.7 MODO ESCLAVO

En el **modo esclavo** el dato es transmitido y recibido a medida que los pulsos del reloj externo aparecen sobre el SCK. Cuando el último bit es capturado, el bit de bandera de interrupción SSPIF es puesto a uno. Estando en el **modo esclavo** el reloj es provisto por una fuente de reloj externo sobre el pin SCK. Este reloj externo debe mantenerse un mínimo de tiempo en alto y en bajo, según lo indiquen las especificaciones eléctricas. Estando en modo *sleep*, el esclavo puede transmitir o recibir datos. Cuando el byte es recibido, el MCU puede despertar de *sleep*.

libros.ep-electropc.com

El pin -SS permite una sincronización en **modo esclavo**. El SPI debe estar en modo esclavo con el pin -SS de control habilitado (SSPCON1<3:0> 0100). El pin -SS debe permanecer en bajo para que funcione como entrada. Cuando el pin -SS está en bajo son habilitadas la transmisión y la recepción, y el pin SDO es maniobrado. Cuando el pin -SS va a alto, el pin SDO no es maniobrado aun si la transmisión está en la mitad de un byte, y se torna una salida flotante, pueden necesitarse resistencias de *pull-up* o *pull-down*, según la aplicación.

```
NOTA: Cuando el SPI está en modo esclavo con el pin -SS habilitado
(SSPCON1<3:0> = 0100), el módulo SPI se resetea si el pin -SS se
pone a VDD.

Si el SPI está en modo esclavo con el bit CKE puesto a 1 (en el
registro SSPSTAT<6>), el pin de control -SS debe habilitarse.
```

Cuando el módulo SPI se resetea, el contador de bit es forzado a "0", esto sucede forzando el pin -SS a alto o borrando el bit SSPEN.

Para emular una comunicación de dos hilos. El pin SDO puede ser conectado al pin SDI. Cuando el SPI necesita operar como receptor, el pin SDO puede ser configurado como entrada. Esto deshabilita las transmisiones desde el SDO. El SDI siempre puede ser dejado como una entrada (función SDI), así no se crea un conflicto de bus.

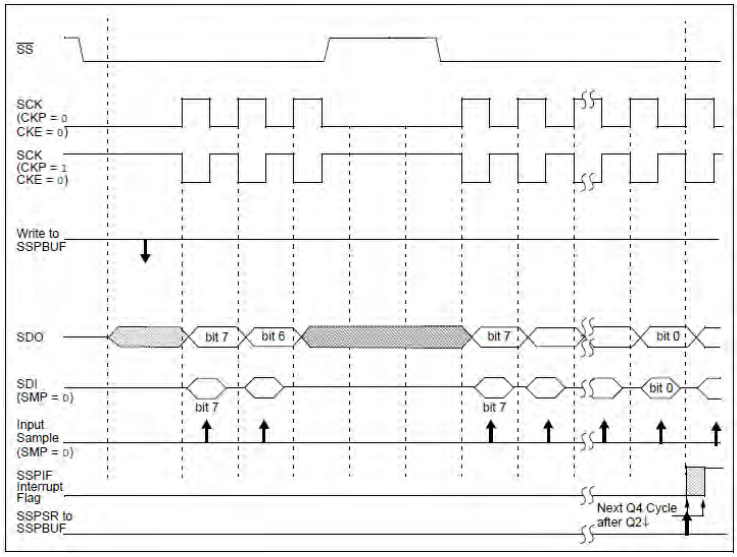


Figura 24.12 Diagrama de tiempos de sincronización en modo esclavo.

libros.ep-electropc.com

En la Figura 24.12 observamos cómo cuando el pin -SS es puesto a V_{DD}, el contador de bit es forzado a cero, y comienza la comunicación de un nuevo byte.

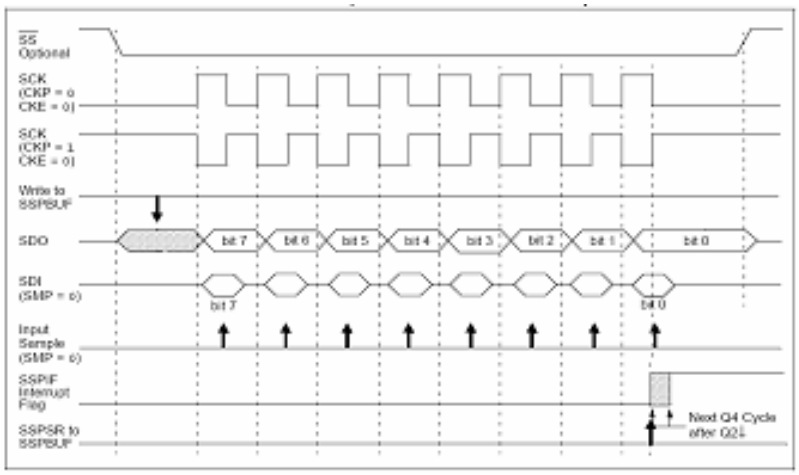


Figura 24.13 Diagrama de tiempos SPI en modo esclavo CKE = 0.

Cuando se trabaja el SPI con CKE = 0, el pin de control -SS es opcional (Figura 24.13).

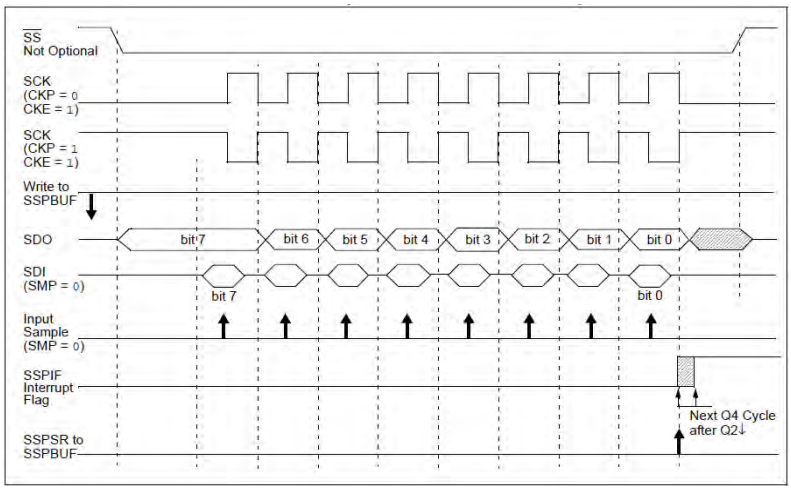


Figura 24.14 Diagrama de tiempos SPI en modo esclavo CKE = 1.

Cuando se trabaja el SPI con CKE = 1, el pin de control -SS es obligatorio (Figura 24.14).

libros.ep-electropc.com

24.2.4.8 OPERACIÓN EN MODO SLEEP

En **modo maestro**, todos los módulos de reloj son detenidos y la transmisión o la recepción permanecerán en este estado hasta que el dispositivo despierte del modo de *sleep*. Después que el dispositivo retorna al modo normal, el módulo continúa transmitiendo o recibiendo datos. En **modo esclavo** el registro **SPI** de movimiento de **transmisión/recepción SSPSR** opera asincrónicamente el dispositivo. Esto permite al dispositivo que sea puesto en modo de *sleep*, y los datos sean movidos en el SSPSR. Cuando todos los 8 bits han sido recibidos, el bit de bandera de interrupción del MSSP puede ser puesto, y si está habilitado puede despertar el dispositivo del modo de *sleep*.

24.2.4.9 EFECTOS DE UN RESET

Un *reset* deshabilita el módulo MSSP y termina la transferencia actual.

24.2.5 REGISTROS ASOCIADOS

El módulo MSSP (Tablas 24.1, 24.2 y 24.3) tiene cuatro registros para el modo de operación SPI:

- MSSP Registro de Control SSPCON1
- MSSP Registro de Status SSPSTAT
- Registro de Buffer Serial Receptor/Transmisor SSPBUF
- MSSP Registro de movimiento SSPSR (no accesible directamente)

libros.ep-electropc.com

Tabla 24.1 Registro SSPCON1.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit (Transmit mode only)
1 = The SSPBUF register is written while it is still transmitting the previous word. (Must be cleared in software.)
0 = No collision

bit 6 **SSPOV:** Receive Overflow Indicator bit
SPI Slave mode:
1 = A new byte is received while the SSPBUF register is still holding the previous data. In case of overflow, the data in SSPSR is lost. Overflow can only occur in Slave mode. The user must read the SSPBUF, even if only transmitting data, to avoid setting overflow. (Must be cleared in software.)
0 = No overflow

Note: In Master mode, the overflow bit is not set, since each new reception (and transmission) is initiated by writing to the SSPBUF register.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
1 = Enables serial port and configures SCK, SDO, SDI, and $\overline{SS}$ as serial port pins
0 = Disables serial port and configures these pins as I/O port pins

Note: When enabled, these pins must be properly configured as input or output.

bit 4 **CKP:** Clock Polarity Select bit
1 = Idle state for clock is a high level
0 = Idle state for clock is a low level

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits
0101 = SPI Slave mode, clock = SCK pin. $\overline{SS}$ pin control disabled. $\overline{SS}$ can be used as I/O pin.
0100 = SPI Slave mode, clock = SCK pin. $\overline{SS}$ pin control enabled.
0011 = SPI Master mode, clock = TMR2 output/2
0010 = SPI Master mode, clock = Fosc/64
0001 = SPI Master mode, clock = Fosc/16
0000 = SPI Master mode, clock = Fosc/4

Note: Bit combinations not specifically listed here are either reserved or implemented in I²C mode only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 24.2 Registro SPSTAT.

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Sample bit
SPI Master mode:
1 = Input data sampled at end of data output time
0 = Input data sampled at middle of data output time
SPI Slave mode:
SMP must be cleared when SPI is used in Slave mode.

bit 6 **CKE:** SPI Clock Select bit
1 = Transmit occurs on transition from active to Idle clock state
0 = Transmit occurs on transition from Idle to active clock state
Note: Polarity of clock state is set by the CKP bit (SSPCON1<4>).

bit 5 **D/$\overline{A}$:** Data/$\overline{Address}$ bit
Used in I²C mode only.

bit 4 **P:** Stop bit
Used in I²C mode only. This bit is cleared when the MSSP module is disabled, SSPEN is cleared.

bit 3 **S:** Start bit
Used in I²C mode only.

bit 2 **R/$\overline{W}$:** Read/$\overline{Write}$ bit information
Used in I²C mode only.

bit 1 **UA:** Update Address bit
Used in I²C mode only.

bit 0 **BF:** Buffer Full Status bit (Receive mode only)
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Tabla 24.3 Registros para el modo de operación SPI.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE/ GIEH	PEIE/ GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
TRISC	PORTC Data Direction Register								1111 1111	1111 1111
SSPBUF	Synchronous Serial Port Receive Buffer/Transmit Register								xxxx xxxx	uuuu uuuu
SSPCON	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	0000 0000	0000 0000
TRISA	—	PORTA Data Direction Register							--11 1111	--11 1111
SSPSTAT	SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF	0000 0000	0000 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'.
Shaded cells are not used by the MSSP in SPI mode.

Note 1: The PSPIF, PSPIE and PSPIP bits are reserved on 28-pin devices; always maintain these bits clear.

libros.ep-electropc.com

24.2.6 PRÁCTICA

Para realizar esta práctica se necesitan dos micros conectados en el modo SPI: el micro MCU-1 es el **maestro** y el micro MCU-2 es el **esclavo**; la conexión se realiza a través del conector P5 en ambos sistemas **PIC16F8XX**, como muestra la Figura 24.15 en el esquema de conexión.

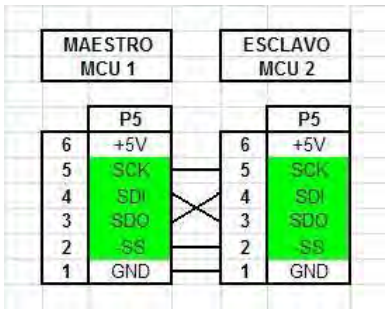


Figura 24.15 Esquema de conexión SPI.

Esta práctica consiste en que el **micro maestro** le envía un byte (8 bits) en forma continua al micro Esclavo que pone el valor en el puerto D, en donde se conecta un display de 7 segmentos y 4 dígitos, y en donde se puede apreciar el código enviado. Por esto debe haber dos programas: uno para el **micro maestro** y otro para el **micro esclavo**. El montaje es como apreciamos en la Figura 24.16.

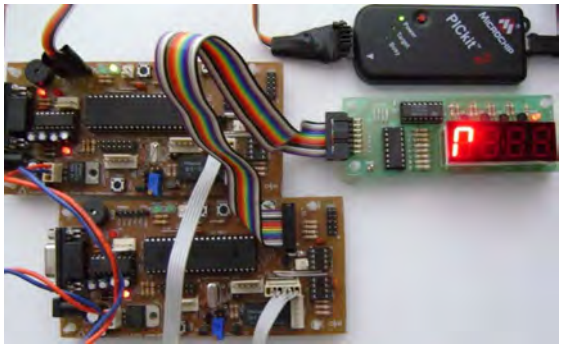


Figura 24.16 Montaje SPI.

libros.ep-electropc.com

24.2.6.1 CÓDIGO SPI MAESTRO

Este micro **maestro** envía un byte (8 bits) **Dato_Tx**, al micro esclavo, todo el tiempo. La conexión SPI = Modo 1,1, Fosc/16.

```
            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de variables específicas del procesador

                                    ; bits de configuración
 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

            cblock 0x20             ; define variables desde posición 20 en RAM
Dato_Tx                 ; Dato variable - enviado por SPI
Dly0                    ; Retardo Variable (low byte)
Dly1                    ; Retardo Variable (high byte)
            endc

#define     _CS         0x05        ; PORTA,5
;-------------------------------------------------- CÓDIGO DEL MAESTRO --------
            org         0x0000      ; inicia programa

            BANKSEL TRISA       ; BANK 1
            movlw   0×00        ; pone PORTA como
            movwf   TRISA       ; todas salidas OUT
            movlw   0×06        ; apague los A/D, deje sólo Digital
            movwf   ADCON1      ; para que PORTA pueda ser usado

            movlw   0x00        ; pone PORTB como CHECK
            movwf   TRISB       ; RB5,4 salidas OUT CHECK

; Set up the SPI Port
;           BANKSEL TRISC       ; BANK 1
            movlw   0x10        ; SCK es salida OUT (maestro), SDI es entrada IN
            movwf   TRISC       ; SDO es salida OUT, todas las otras OUT
            movlw   0x40        ; Modo 1,1 SPI
            movwf   SSPSTAT     ; tiempo de muestreo
            BANKSEL SSPCON      ; BANK 0
            movlw   0x31        ; Modo 1,1 SPI modo maestro, 1/16 Tosc bit tiempo
            movwf   SSPCON      ; SSP encendido

            bcf         PORTB,4             ; apague OUT CHECK

            movlw       0x22                ; valor del dato a Tx
            movwf       Dato_Tx             ; carga en variable Dato_Tx
Send_DT
            bcf         PORTA,_CS           ; habilite Chip Select OUT (low)

            bsf         PORTB,5             ; enciende OUT CHECK

            movf        Dato_Tx,W           ; carga Dato_Tx -> en W
            movwf       SSPBUF              ; póngalo en SSPBUF
            BANKSEL SSPSTAT             ; BANK 1
Char1
```


libros.ep-electropc.com

```
            btfss   SSPSTAT,BF         ; ¿Dato transferido completo? (Buffer Full?)
            goto    Char1              ; no, check de nuevo
            BANKSEL SSPBUF             ; BANK0
            movf    SSPBUF,W           ; lea dato desde SSPBUF (dato entrante)
            bsf     PORTA,_CS          ; Deshabilite Chip Select OUT (high)

            bcf     PORTB,5            ; apaga OUT CHECK

Update_Test_Counter
;           incf    Dato_Tx,F ; incremente variable Dato_Tx

Delay
            movlw   0x01               ; retardo sencillo loop
            movwf   Dly1               ; |
            movlw   0x0F               ; |
            movwf   Dly0               ; |

DlyLoop
            decfsz  Dly0,F             ; |
            goto    DlyLoop            ; |
            decfsz  Dly1,F             ; |
            goto    DlyLoop            ; |
                                       ; Retardo hecho \_/

DoAgain
            goto    Send_DT            ; hecho, envía siguiente Byte.
;-------------------------------------------------------------------
            end
```

Este ejemplo está en nuestra página Web en la ruta X:\Ejemplos_ASM, que debe ser copiado al C:\, para encontrarlo como C:\Ejemplos_ASM\SPI_1\SPI_1_esclavo.asm.

24.2.6.2 CÓDIGO SPI ESCLAVO

El micro Esclavo recibe el dato del **micro maestro** y lo pone en el PORTD, en donde se conecta un display de 7 segmentos, y se aprecia el código enviado.

```
            list p = 16f877a       ; directiva para definir procesador
 #include <p16f877a.inc>           ; definiciones de variables específicas del procesador

                                   ; bits de configuración
 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _CS_TRIS 0x20          ; PORTA,2
;------------------------------------- CÓDIGO DEL ESCLAVO --------
            org     0x0000         ; inicia programa

            BANKSEL TRISA          ; BANK 1
            movlw   _CS_TRIS ; defina pin de Chip Select
            movwf   TRISA          ; es un pin de entrada IN
```


libros.ep-electropc.com

```
            movlw   0x00    ; configure todos los pines PORTE para ser salidas
            movwf   TRISE   ; control de dígitos de display de 7 segmentos

            movlw   0x06    ; apague los A/D, deje sólo Digital
            movwf   ADCON1  ; para que PORTA pueda ser usado

;           BANKSEL TRISD   ; BANK 1
            movlw   0x00    ; ponga PORTD como salidas OUT
            movwf   TRISD   ; salidas de 7 segmentos de display

; configura puerto SPI
;           BANKSEL TRISC   ; BANK 1
            movlw   0x10    ; SCK es salida OUT (maestro), SDI es entrada IN
            movwf   TRISC   ; SDO es salida OUT, todas las otras OUT
            movlw   0x40    ; Modo 1,1 SPI
            movwf   SSPSTAT ; tiempo de muestreo
            BANKSEL SSPCON  ; BANK 0



            movlw   0x34    ; Mode 1,1 SPI modo esclavo / Requerido
            movwf   SSPCON  ; SSP encendido

Chk4Dat
            BANKSEL SSPSTAT         ; BANK 1
            btfss   SSPSTAT,BF      ; ¿Dato transferido completo? (Buffer Full?)
            goto    Chk4Dat         ; no, check de nuevo
                                    ; se puede agregar código antes que se reciba el byte completo

RX_Data
            BANKSEL SSPBUF          ; BANK 0
            movf    SSPBUF,W        ; pone SSPBUF -> en W
;           BANKSEL PORTD           ; BANK 0
            movwf   PORTD           ; muestra resultado en PORTD
            bsf     PORTE,2         ; prende dígito 1 de display
DoAgain1
            ; Optional código para antes de próximo dígito
            goto    Chk4Dat         ; Recibe siguiente Byte

            end
```

Este ejemplo está en nuestra página Web en la ruta: X:\Ejemplos_ASM, que debe ser copiado al C:\, para encontrarlo como C:\Ejemplos_ASM\SPI_1\SPI_1_maestro.asm.

El código es enviado por el micro **maestro** y recibido por el micro **esclavo** y puesto en el **puerto D**, que a su vez genera un carácter en el display de 7 segmentos; estos ejemplos de 3 códigos se presentan a continuación (Figura 24.17).

libros.ep-electropc.com

			128	64	32	16	8	4	2	1	PESO
			7	6	5	4	3	2	1	0	BIT
HEX	HEX	DEC	P	G	F	E	D	C	B	A	SEG
64h	64	100	0	1	1	0	0	1	0	0	0
11h	11	17	0	0	0	1	0	0	0	1	1
33h	33	51	0	0	1	1	0	0	1	1	2



Figura 24.17 Caracteres y código.

Los códigos son 0x64, 0x11 y 0x33 (Figuras 24.18 a, b; 24.19 a, b y 24.20 a, b) generan los patrones en el display de 7 segmentos, como vemos a la derecha en la Figura 24.18a, b; observamos la correspondencia de los bits **A-P** con los bits **0-7** en cada uno de ellos.

a) 

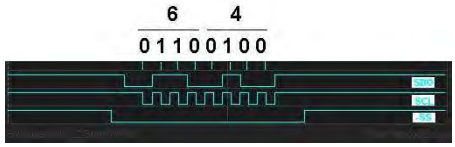
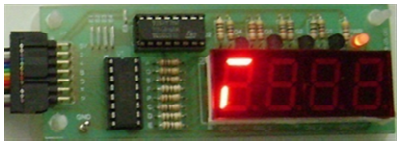
b) 

Figura 24.18 Carácter y trama código 0x64.

a) 

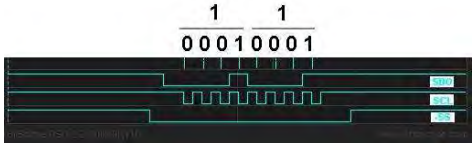
b) 

Figura 24.19 Carácter y trama código 0x11.

a) 

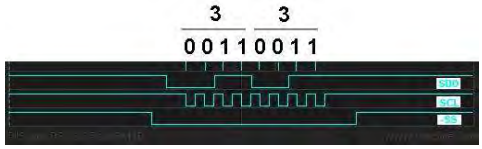
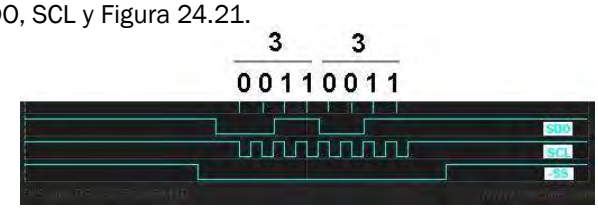
b) 

Figura 24.21 Trama código 0x33.

Cada una de las figuras anteriores contiene el carácter generado en el display de 7 segmentos, y los bits correspondientes en cada una de las tramas de transmisión, observe los pines SDO, SCL y Figura 24.21.

Figura 24.21
Trama código 0x33.



libros.ep-electropc.com

En la trama se detalla cómo está en bajo, mientras se envía la trama de bits, SCL son los pulsos de reloj que, en la subida, cambian el dato SDO, y en la caída se lee el dato.

EVALUACIÓN

```
1. El módulo de comunicaciones MSSP contiene dos modos de
operación, ¿cuáles son?
R.:
Modo SPI y modo I2C
SPI: Serial Peripheral Interface
I2C: Inter-Integrated Circuit

2. ¿Cuáles son los 4 pines usados en el Modo SPI?
R.:
SCK   Serial Clock
MOSI  Master Ouput Slave Input
MISO  Master Input Slave Output
-SS   Slave Select

3. ¿El módulo SPI puede operar como maestro y esclavo?
R.:
Sí.
```

25.0 MODO COMUNICACIÓN INTER-INTEGRADOS (I2C)

Las **comunicaciones I²C** tienen la intención de realizarse con dispositivos en la misma tarjeta o board, a velocidades de 100 kHz. Estos dispositivos son de una amplia variedad: memorias, RTC, drivers para display, sensores de temperatura y DAC, entre otros. Comenzamos por entender qué es el estándar I²C, y luego vemos el módulo MSSP en modo I²C.

25.1 ESTÁNDAR I2C

El **IIC** (*Inter-Integrated Circuit*: **circuitos interintegrados**) es un bus de comunicación serial diseñado por Phillips, en 1992; su velocidad es de 100 Kbps a 400 Kbps, aunque las velocidades han ido creciendo a 1 Mb y 3.4 Mbps.

25.1.1 HARDWARE

El IIC utiliza dos hilos para la comunicación: el SDA datos, el SCL reloj, y un tercero como referencia o GND; los dos primeros son configurados como drenaje abierto;

libros.ep-electropc.com

por esto requieren resistencias de *pull-up*. Los dispositivos conectados a este bus tienen una dirección única por cada dispositivo. El dispositivo maestro inicia la transferencia de datos (Figura 25.1), generando el reloj para todos los dispositivos esclavos; puede haber varios **dispositivos maestros** con la misma capacidad de manejo del bus; por esto este bus es **multimaestro**.



Figura 25.1 Conexión bus I²C.

Cada dispositivo en sus dos líneas de conexión tiene configurada la circuitería de entrada y de salida (Figura 25.2).

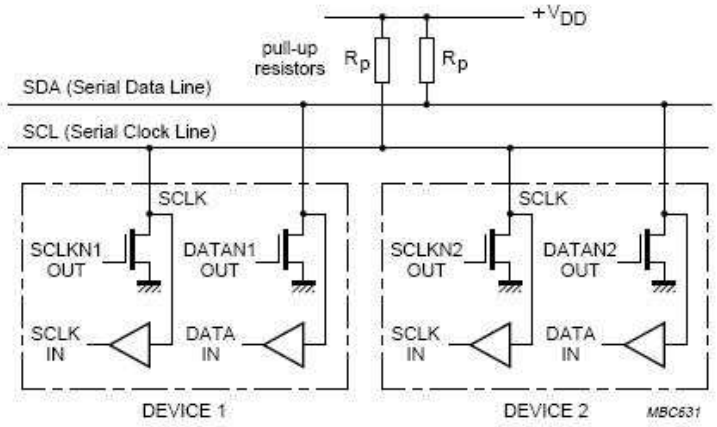


Figura 25.2 Conexión interna de dispositivos I²C.

libros.ep-electropc.com

25.1.2 TERMINOLOGÍA

Tabla 25.1 Terminología de bus I²C.

Transmisor	Dispositivo que envía datos al bus.
Receptor	Dispositivo que recibe datos del bus.
Maestro	Dispositivo que inicia una transferencia, genera señal de reloj, y termina la transferencia de datos.
Esclavo	Dispositivo direccionado por el maestro.
Multimaestro	Más de un maestro puede intentar controlar el bus al mismo tiempo, sin corromper el mensaje.
Arbitraje	Procedimiento para asegurar que si más de un maestro simultáneamente trata de controlar el bus, solamente a uno le es permitido y el mensaje del ganador no es corrompido.
Sincronización	Procedimiento para sincronizar las señales de reloj de dos o más dispositivos.

Dadas las definiciones de la Tabla 25.1, se habla de:

- Maestro transmisor es un caso común.
- Esclavo receptor es otro caso común.
- Maestro receptor, cuando recibe datos de un dispositivo esclavo transmisor.
- Esclavo transmisor, cuando envía datos a un maestro receptor.

25.1.3 PROTOCOLO

Los estados de las líneas de comunicación son como aparecen en la Figura 25.3, en los cuales diferenciamos varios estados.

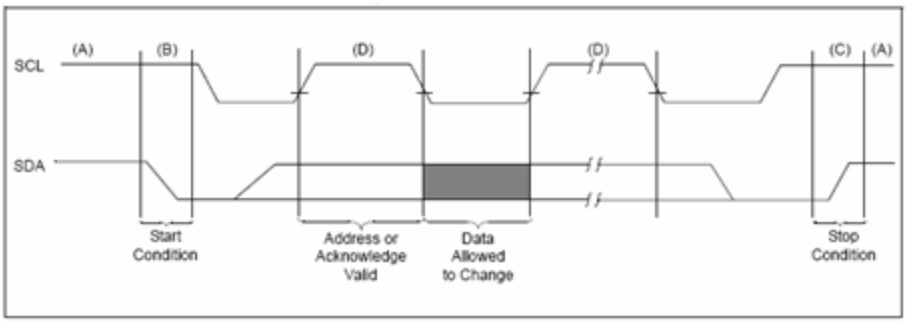


Figura 25.3 Estados del bus I²C.

libros.ep-electropc.com

Las **reglas del protocolo** son:

- La transferencia de datos solamente se puede iniciar cuando el bus no está ocupado.
- Condición de *data valid* **(dato válido**); durante una transferencia de datos la línea datos SDA debe permanecer estable, mientras el reloj SCL está en alto (Figura 25.3, **s**ección D**)**. Los cambios en la línea de datos SDA, cuando el reloj SCL está en alto, se pueden interpretar como una condición de *start* o *stop*.
- Condición de *bus not busy* (**bus no ocupado**); ambas líneas (data SDA y reloj SCL) están en alto (Figura 25.3, **s**ección A**)**.
- Condición de *start* (**arranque**); transición de alto a bajo de la línea de datos SDA, mientras la línea de reloj SCL está en alto (Figura 25.3, sección B**)**. Todos los comandos deber ser precedidos de una condición de *start*.
- Condición de *stop* (**parada);** transición de bajo a alto de la línea de datos SDA, mientras la línea de reloj SCL está en alto (Figura 25.3, sección C**)**. Todos los comandos deber ser terminados con una condición de *stop*.
- Condición de *acknowledge* (**reconocimiento);** cada receptor direccionado, luego de recibir un byte, debe generar un *acknowledge*. El **maestro** debe generar un reloj extra que está asociado con un bit de *acknowledge*.

El **dispositivo que reconoce** (el *acknowledge*) debe poner la líneas SDA en bajo durante el reloj de *acknowledge*; la línea de datos SDA debe ser estable en bajo durante el pulso de reloj SCL. Por supuesto, se deben tomar en cuenta los tiempos de cada dispositivo en particular. Durante lecturas, un **maestro-receptor** debe señalizar un final de datos al Esclavo-Transmisor, generando un bit de *no-Aknowledge* sobre el último byte que ha sido *sincronizado* fuera del Esclavo. En este caso, el Esclavo-Transmisor deja la línea de datos SDA en alto para habilitar al **maestro** para que genere la condición de *stop.*

25.1.4 TRANSFERENCIA DE DATOS

Cada comando debe comenzar por una condición de *start*, y debe terminar con una condición de *stop*. Un dispositivo **maestro** comienza la transferencia y responde un dispositivo Esclavo. Cada dato debe ser de 8 bits, y por esto debe ser acompañado de 8 pulsos de reloj; no hay restricción en la cantidad de datos; cada dato es transferido comenzando en el MSB bit 7 al bit 0 o LSB; cada dato debe ir acompañado de un *acknowledge* que es el bit 9 con el correspondiente pulso 9 de reloj. Si el **dispositivo esclavo** no puede recibir o transmitir otro byte completo hasta realizar una función interna, puede poner el pin SCL en **"0**" para forzar al dispositivo

libros.ep-electropc.com

maestro a que espere (Figura 25.4). La transferencia de datos puede continuar cuando el dispositivo Esclavo esté listo para otro byte y libere la línea SCL.

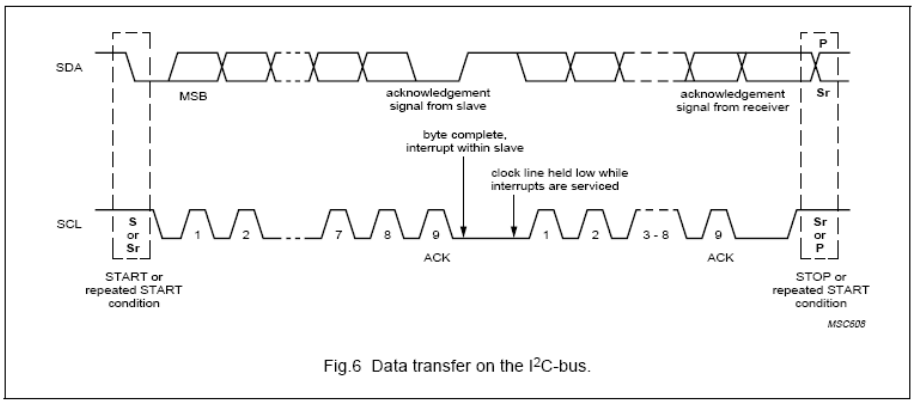


Figura 25.4 Transferencia de datos sobre bus I²C.

El reconocimiento o *acknowledge* es generado por el dispositivo receptor **maestro** o Esclavo.

25.2 MÓDULO MSSP EN MODO I2C

El módulo MSSP en modo I²C es totalmente implementado en las funciones de **Maestro** y **Esclavo**, y provee interrupciones sobre bits de *start* y *stop* en hardware para determinar el bus libre (función multimaestro); también soporta los **estándares** de direccionamiento de 7 y 10 bits.

Los dos pines usados para transferencia de datos son:

- **Reloj Serial** (**SCL**) – RC3/SCK/SCL
- **Dato Serial** (**SDA**) – RC4/SDI/SDA

El usuario debe configurar estos pines como entradas o salidas a través de los bits TRISC<4:3>, o en el bit SSPEN en el registro SSPCON1. Para la operación de I²C, el módulo MSSP tiene 6 registros:

1. *Control Register* (SSPCON1)
2. *Control Register 2* (SSPCON2)
3. *Status Register* (SSPSTAT)
4. *Serial Receive/Transmit Buffer Register* (SSPBUF)

libros.ep-electropc.com

5. *Shift Register* (SSPSR) – *Not directly accessible*
6. *Address Register* (SSPADD)

Los registros SSPCON1, SSPCON2 y SSPSTAT son registros de control y estado en el modo de operación del I²C, los registros SSPCON1 y SSPCON2 son de lectura y escritura, y el registro SSPSTAT en los bits <5:0> es de sólo lectura y los bits <7:6> son de lectura y escritura. El SSPSR, que es el registro de movimiento (*shift register)* usado para mover los datos entrantes o salientes, es el registro de buffer para los datos escritos *en* y leídos *desde*. El registro SSPADD mantiene la dirección del dispositivo esclavo cuando el SSP es configurado en modo esclavo I²C. Cuando el SSP es configurado en modo maestro, los 7 bits más bajos actúan como el recargador de valor del *Baud Rate Generator* o generador de tasa de baudios.

En operaciones de recepción, los registros SSPSR y SSPBUF juntos crean un doble buffer de recepción. Cuando el registro SSPSR recibe un byte completo, éste es transferido al SSPBUF y el bit de interrupción SSPIF es puesto a uno. Durante transmisión, el SSPBUF no es buffer doble. Una escritura al SSPBUF puede escribir a ambos registros SSPBUF y SSPSR.

25.2.1 DIAGRAMA CIRCUITAL

Se muestra en las Figuras 25.5 y 25.6.

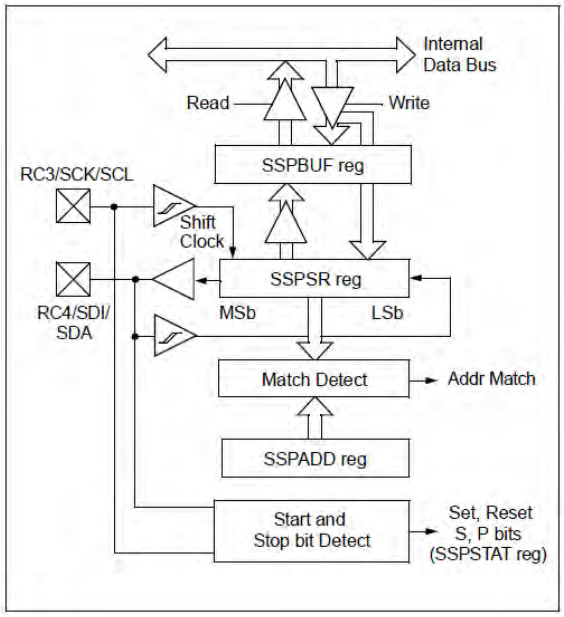


Figura 25.5 Conexión modo I²C.

libros.ep-electropc.com

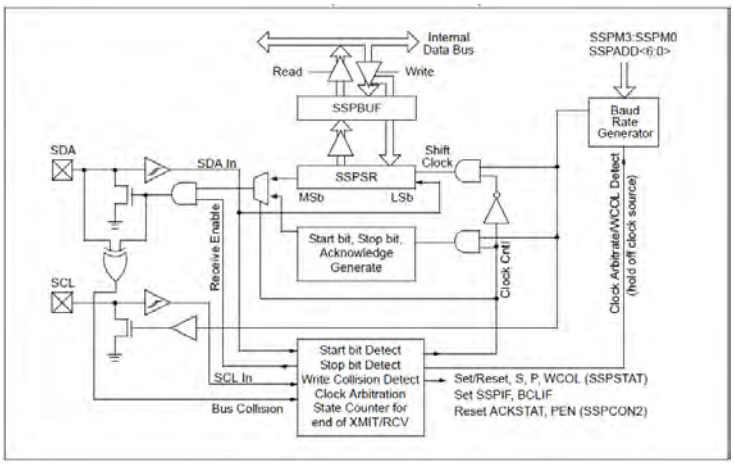


Figura 25.6 Conexión modo maestro I²C.

25.2.2 FUNCIONAMIENTO

Las funciones del módulo MSSP se habilitan poniendo a uno el bit de habilitación del MSSP, o el bit SSPEN en el registro SSPCON1<5>. El registro SSPCON1 permite el control de la operación del I²C. Los 4 bits de selección de modo SSPCON<3:0> permiten que sea seleccionado uno de los modos siguientes:

- I²C modo maestro, reloj = OSC/(4 * (SSPADD+1))
- I²C modo esclavo (7 bits de dirección)
- I²C modo esclavo (10 bits de dirección)
- I²C modo esclavo (7 bits de dirección), con interrupciones habilitadas bit de *start* y *stop*
- I²C modo esclavo (10 bits de dirección), con interrupciones habilitadas bit de *start* y *stop*
- I²C modo maestro controlado por *firmware*, esclavo es habilitado

La selección de cualquier **modo I²C**, con el bit SSPEN puesto a uno, fuerza los pines SCL y SDA a ser abiertos u *open drain*; para configurarlos como entradas se deben poner a uno los bits TRISC necesarios. Para asegurar la operación apropiada del módulo deben ponerse resistencias externas de *pull-up*, sobre los pines SDA y SCL.

libros.ep-electropc.com

25.2.2.1 BRG (BAUD RATE GENERATOR)

El **BRG**, usado por el módulo SPI, se utiliza para definir la frecuencia del reloj SCL, para operaciones de 100 Kpbs, 400 Kpbs y 1 Mbps (Tabla 25.2).

Tabla 25.2 Valores de BRG.

FCY	FCY*2	BRG Value	FSCL (2 Rollovers of BRG)
10 MHz	20 MHz	19h	400 kHz(1)
10 MHz	20 MHz	20h	312.5 kHz
10 MHz	20 MHz	3Fh	100 kHz
4 MHz	8 MHz	0Ah	400 kHz(1)
4 MHz	8 MHz	0Dh	308 kHz
4 MHz	8 MHz	28h	100 kHz
1 MHz	2 MHz	03h	333 kHz(1)
1 MHz	2 MHz	0Ah	100 kHz
1 MHz	2 MHz	00h	1 MHz(1)

Note 1: The I²C interface does not conform to the 400 kHz I²C specification (which applies to rates greater than 100 kHz) in all details, but may be used with care where higher rates are required by the application.

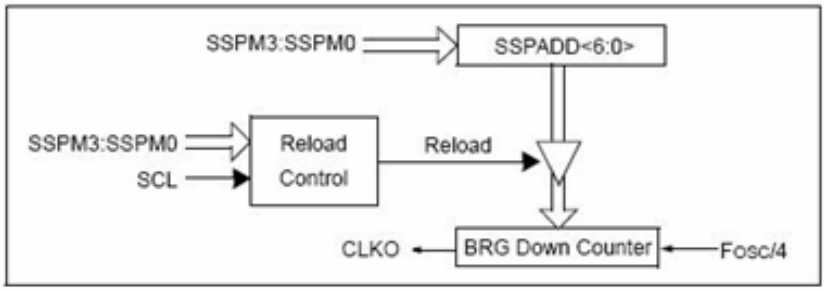


Figura 25.7 Diagrama del BRG.

En el **modo maestro I²C**, el **BRG** (Figura 25.7) es recargado automáticamente, y el valor de recarga es almacenado en los 7 bits de menos peso del registro SSPADD. Cuando ocurre una escritura al SSPBUF, el BRG comienza automáticamente a contar. El BRG cuenta hacia abajo hasta 0 y se detiene, hasta que otra recarga se realiza. El contador BRG es disminuido dos veces por ciclo de instrucción TCY sobre los relojes **Q2** y **Q4**.

25.2.2.2 MODO ESCLAVO

En este modo de operación es cuando el MCU es el módulo esclavo de otro procesador mayor; este modo no se trata en este libro. Los interesados en este

libros.ep-electropc.com

modo de operación deben remitirse al **Capítulo 9.0, Sección 9.4.3: Slave Mode** del archivo **DS39582B**, pdf, p. 83 que es la *data sheet* del microcontrolador PIC16F877A.

25.2.2.3 MODO MAESTRO

En este modo de operación el **MCU** es el **maestro** y se comunica con otros dispositivos esclavos. Este modo es habilitado por poner los bits **SSPM<3:0>** en "1000" y poner a uno el bit **SSPEN** en el registro **SSPCON1<5>**. Los pines **SCL** y **SDA** son manipulados por el hardware del **MSSP**. La operación de modo maestro es soportada por la generación de interrupciones sobre la detección de las condiciones de *start* y *stop*. El bit de *stop* (**P**) y el *start* (**S**) son borrados por un *reset*, o cuando el módulo MSSP es deshabilitado. El control del bus I²C puede ser tomado cuando el bit **P** es puesto a uno, o el bus está libre, con ambos bits **S** y **P** borrados.

En el **modo maestro de firmware controlado**, el código de usuario conduce todas las operaciones del bus I²C, basado en las condiciones de los bits *start* y *stop*.

```
NOTA: El módulo MSSP, cuando es configurado en modo maestro, no
permite cola de eventos. Por ejemplo, al usuario no le es permitido
iniciar una condición de start e inmediatamente escribir sobre el
registro SSPBUF para iniciar una transmisión, antes que la
condición de start se complete. En este caso, el registro SSPBUF no
puede ser escrito, el bit WCOL será puesto a uno, indicando que no
ocurrió una escritura al SSPBUF.
```

El módulo genera todos los pulsos del reloj serial y las condiciones de *start* y *stop*; una transferencia es finalizada con una condición de *stop*, o con una condición de *start* repetida. Así, una condición de *start* repetida es también el comienzo de la transferencia serial siguiente que el bus I²C no pudo realizar. En el modo de **maestro transmisor**, los datos seriales salen a través del pin **SDA**, y los pulsos de reloj salen por el pin **SCL**. El primer byte transmitido contiene la dirección del dispositivo **esclavo receptor** en 7 bits, y el bit **R/W** (*Read/Write*) que será **"0"** lógico. Los ocho bits del dato serial son transmitidos con los correspondientes pulsos de reloj. Luego que el esclavo haya recibido cada byte, envía un reconocimiento *acknowledge*. El bit de *start* y *stop* son el comienzo y el final de una transferencia de datos.

En el modo **maestro receptor** el primer byte transmitido contiene la dirección del **esclavo receptor** del dispositivo transmisor en 7 bits, y el bit **R/W** (*Read/Write*) que será "1" lógico. Los datos seriales son recibidos por el pin **SDA**, y los pulsos de reloj por el pin **SCL**. Después que cada byte es recibido, un bit de *acknowledge* es transmitido. Las condiciones de *start* y *stop* son salidas para indicar el comienzo y el fin de una transferencia serial.

libros.ep-electropc.com

La secuencia de transmisión típica puede ser:

1. El usuario genera una condición de start, poniendo a uno el bit SEN (*start* ENable) del registro **SSPCON2<0>**.
2. El SSPIF es puesto a uno por el hardware, indicando que terminó el tiempo de *start*. El módulo MMSP espera el tiempo *start* requerido antes que otra operación se realice; es decir, espera a que SSPIF = 1.
3. El usuario carga el SSPBUF con la dirección que se va a transmitir.
4. La dirección es movida hacia afuera por el pin SDA hasta que los 8 bits sean transmitidos, SSPIF es puesto a uno por hardware, indicando que terminó de transmitir los 8 bits.
5. El módulo MSSP mueve el bit de *acknowledge* desde el dispositivo Esclavo, y escribe este valor en el bit ACKSTAT del registro SSPCON2<6>, donde se puede verificar para completar el ciclo.
6. El módulo MSSP genera una interrupción al final del noveno ciclo de reloj, seteando el bit SSPIF por hardware.
7. El usuario carga el registro SSPBUFF con los 8 bits del dato.
8. El dato es movido afuera por el pin SDA hasta que todos los 8 bits son transmitidos.
9. El módulo MSSP mueve el bit de *acknowledge* desde el dispositivo Esclavo, y escribe este valor en el bit ACKSTAT del registro SSPCON2<6>.
10. El módulo MSSP genera una interrupción al final del noveno ciclo de reloj, seteando el bit SSPIF.
11. El usuario genera una condición de *stop*, seteando el sto**P** bit **EN**able (**PEN)** en el registro SSPCON2<2>.
12. La interrupción es generada una vez que la condición de *stop* está completa.

25.2.3 REGISTROS ASOCIADOS

Se describen:

1. *Control Register* (SSPCON)
2. MSSP *Control Register 2* (SSPCON2)
3. MSSP *Status Register* (SSPSTAT)
4. *Serial Receive/Transmit Buffer Register* (SSPBUF)

libros.ep-electropc.com

5. MSSP *Shift Register* (SSPSR) – *Not directly accessible*
6. MSSP *Address Register* (SSPADD)

Tabla 25.3 Registro SSPSTAT.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

bit 7 **WCOL:** Write Collision Detect bit

In Master Transmit mode:
1 = A write to the SSPBUF register was attempted while the I²C conditions were not valid for a transmission to be started. (Must be cleared in software.)
0 = No collision

In Slave Transmit mode:
1 = The SSPBUF register is written while it is still transmitting the previous word. (Must be cleared in software.)
0 = No collision

In Receive mode (Master or Slave modes):
This is a "don't care" bit.

bit 6 **SSPOV:** Receive Overflow Indicator bit

In Receive mode:
1 = A byte is received while the SSPBUF register is still holding the previous byte. (Must be cleared in software.)
0 = No overflow

In Transmit mode:
This is a "don't care" bit in Transmit mode.

bit 5 **SSPEN:** Synchronous Serial Port Enable bit
1 = Enables the serial port and configures the SDA and SCL pins as the serial port pins
0 = Disables the serial port and configures these pins as I/O port pins

Note: When enabled, the SDA and SCL pins must be properly configured as input or output.

bit 4 **CKP:** SCK Release Control bit

In Slave mode:
1 = Release clock
0 = Holds clock low (clock stretch). (Used to ensure data setup time.)

In Master mode:
Unused in this mode.

bit 3-0 **SSPM3:SSPM0:** Synchronous Serial Port Mode Select bits
1111 = I²C Slave mode, 10-bit address with Start and Stop bit interrupts enabled
1110 = I²C Slave mode, 7-bit address with Start and Stop bit interrupts enabled
1011 = I²C Firmware Controlled Master mode (Slave Idle)
1000 = I²C Master mode, clock = FOSC/(4 * (SSPADD + 1))
0111 = I²C Slave mode, 10-bit address
0110 = I²C Slave mode, 7-bit address

Note: Bit combinations not specifically listed here are either reserved or implemented in SPI mode only.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 25.4 Registro SSPCON1.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
bit 7							bit 0

bit 7 **GCEN:** General Call Enable bit (Slave mode only)
1 = Enable interrupt when a general call address (0000h) is received in the SSPSR
0 = General call address disabled

bit 6 **ACKSTAT:** Acknowledge Status bit (Master Transmit mode only)
1 = Acknowledge was not received from slave
0 = Acknowledge was received from slave

bit 5 **ACKDT:** Acknowledge Data bit (Master Receive mode only)
1 = Not Acknowledge
0 = Acknowledge

Note: Value that will be transmitted when the user initiates an Acknowledge sequence at the end of a receive.

bit 4 **ACKEN:** Acknowledge Sequence Enable bit (Master Receive mode only)
1 = Initiate Acknowledge sequence on SDA and SCL pins and transmit ACKDT data bit. Automatically cleared by hardware.
0 = Acknowledge sequence Idle

bit 3 **RCEN:** Receive Enable bit (Master mode only)
1 = Enables Receive mode for I^2C
0 = Receive Idle

bit 2 **PEN:** Stop Condition Enable bit (Master mode only)
1 = Initiate Stop condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Stop condition Idle

bit 1 **RSEN:** Repeated Start Condition Enabled bit (Master mode only)
1 = Initiate Repeated Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Repeated Start condition Idle

bit 0 **SEN:** Start Condition Enabled/Stretch Enabled bit
In Master mode:
1 = Initiate Start condition on SDA and SCL pins. Automatically cleared by hardware.
0 = Start condition Idle
In Slave mode:
1 = Clock stretching is enabled for both slave transmit and slave receive (stretch enabled)
0 = Clock stretching is enabled for slave transmit only (PIC16F87X compatibility)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Note: For bits ACKEN, RCEN, PEN, RSEN, SEN: If the I^2C module is not in the Idle mode, this bit may not be set (no spooling) and the SSPBUF may not be written (or writes to the SSPBUF are disabled).

libros.ep-electropc.com

Tabla 25.5 Registro SSPCON2.

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/$\overline{A}$	P	S	R/$\overline{W}$	UA	BF
bit 7							bit 0

bit 7 **SMP:** Slew Rate Control bit
In Master or Slave mode:
1 = Slew rate control disabled for standard speed mode (100 kHz and 1 MHz)
0 = Slew rate control enabled for high-speed mode (400 kHz)

bit 6 **CKE:** SMBus Select bit
In Master or Slave mode:
1 = Enable SMBus specific inputs
0 = Disable SMBus specific inputs

bit 5 **D/$\overline{A}$:** Data/$\overline{Address}$ bit
In Master mode:
Reserved.
In Slave mode:
1 = Indicates that the last byte received or transmitted was data
0 = Indicates that the last byte received or transmitted was address

bit 4 **P:** Stop bit
1 = Indicates that a Stop bit has been detected last
0 = Stop bit was not detected last
Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 3 **S:** Start bit
1 = Indicates that a Start bit has been detected last
0 = Start bit was not detected last
Note: This bit is cleared on Reset and when SSPEN is cleared.

bit 2 **R/$\overline{W}$:** Read/$\overline{Write}$ bit information (I^2C mode only)
In Slave mode:
1 = Read
0 = Write
Note: This bit holds the R/$\overline{W}$ bit information following the last address match. This bit is only valid from the address match to the next Start bit, Stop bit or not $\overline{ACK}$ bit.
In Master mode:
1 = Transmit is in progress
0 = Transmit is not in progress
Note: ORing this bit with SEN, RSEN, PEN, RCEN or ACKEN will indicate if the MSSP is in Idle mode.

bit 1 **UA:** Update Address (10-bit Slave mode only)
1 = Indicates that the user needs to update the address in the SSPADD register
0 = Address does not need to be updated

bit 0 **BF:** Buffer Full Status bit
In Transmit mode:
1 = Receive complete, SSPBUF is full
0 = Receive not complete, SSPBUF is empty
In Receive mode:
1 = Data Transmit in progress (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is full
0 = Data Transmit complete (does not include the $\overline{ACK}$ and Stop bits), SSPBUF is empty

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

25.2.4 PRÁCTICA

En esta práctica se realizan ejemplos de escritura y lectura con la memoria externa **EEPROM 24LC32**, con **protocolo I²C**. Se anota que estas rutinas sirven en general para las **memorias 24XX**; se debe tener en cuenta la referencia de la memoria para verificar los bytes de direccionamiento. En algunas de ellas, el código para realizar rutinas de supervisión para errores está planteado, para que el usuario lo implemente. Para realizar cualquier ejemplo de escritura o lectura, se deben hacer varias rutinas, como observamos más adelante en el código presentado. El montaje para esta práctica se realiza con el sistema de evaluación **PIC16F8XX** y el programador **PICkit 2**; se inserta la **memoria 24C32** en el socket U5 de la tarjeta del sistema de evaluación **PIC16F8XX**, como se aprecia en la Figura 25.8.

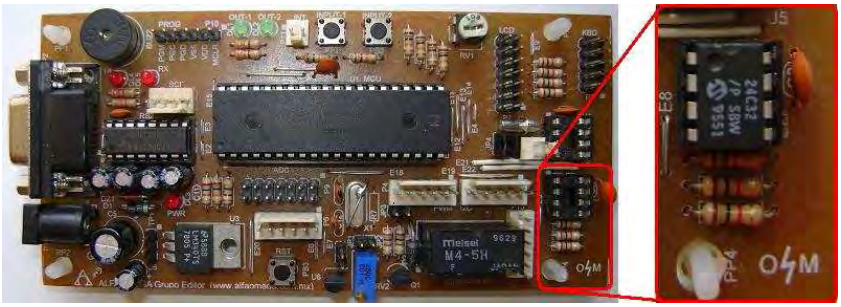


Figura 25.8 PIC16F8XX socket de memoria EEPROM.

El plano esquemático de conexión se describe en la Figura 25.9 a, b.

libros.ep-electropc.com

a)

b)

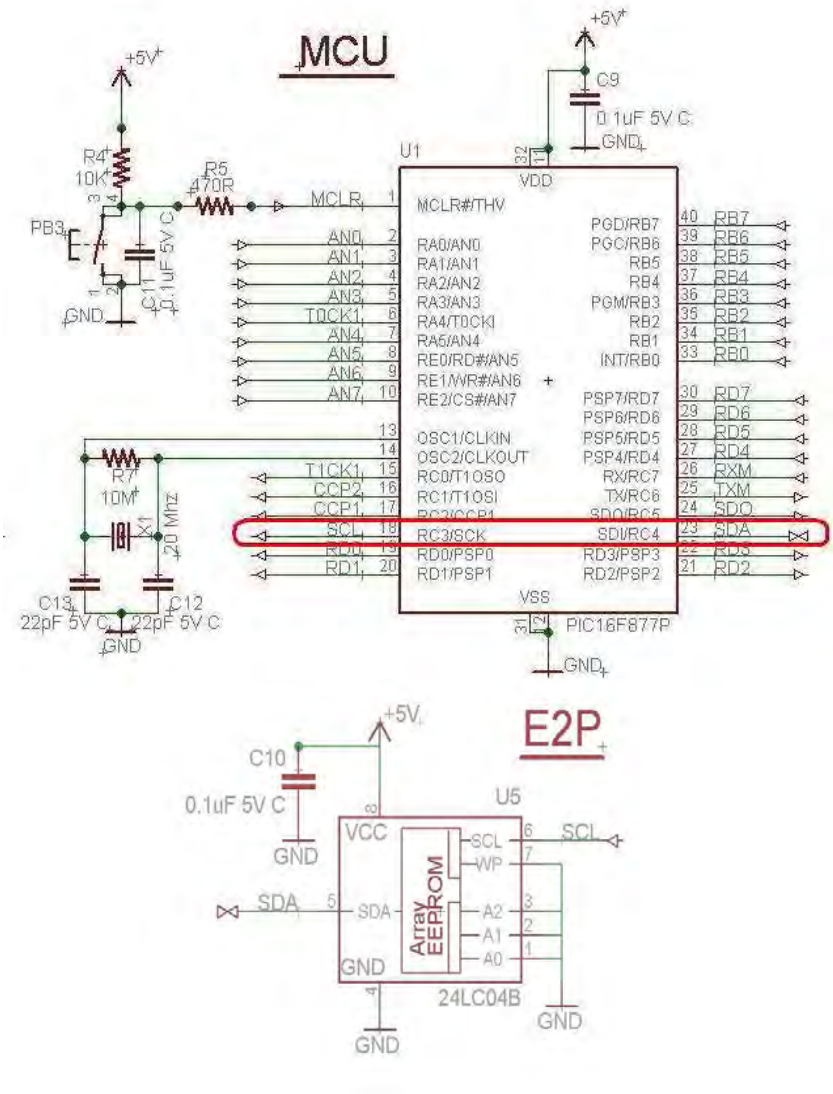


Figura 25.9 Plano esquemático E2P I²C.

A continuación se entrega el código para realizar tareas en la memoria 24LC32 EEPROM externa. Las tareas son de lectura y escritura de byte y bytes. Observe cómo, en el programa principal *start*, se pueden comentar algunas instrucciones para deshabilitarlas. En el siguiente ejemplo se escriben 16 datos en la **memoria EEPROM** desde la posición 0x0AA0, y luego se leen y se colocan en la **memoria RAM** en la posición 0x2A.

libros.ep-electropc.com

El siguiente ejemplo se puede encontrar en nuestra página Web en el CD info en la carpeta X:\Ejemplos_ASM, que debe copiarse al C:\ para encontrase como C:\Ejemplos_ASM\ \I2C_1\I2C_1.asm.

```
;** DESCRIPCIÓN                                                                **
;** ===========                                                                **
;**  Estas rutinas fueron escritas para la memoria 24LC32, tomadas de la AN976  **
;**                                                                             **
;**         start. Comienzo del programa pirncipal, este llama a las demas subrutinas **
;**         Init. inicializaciòn y configuraciòn de Puerto, Pines y variables   **
;**         BSTART. condiciòn de Bit de Start                                   **
;**         BRESTART. Condiciòn de bit de Re-Start                              **
;**         BSTOP. Condiciòn de bit de Stop                                     **
;**         TX. Transmisiòn de un byte "datao"                                  **
;**         RX. Recepciòn de un byte en "datai"                                 **
;**         ByteWrite. Escritura de un byte _DATA en direcciòn (_MSB_ADD:LSB_ADDR) **
;**         PageWrite. Escritura de 16 byte _DATA en direcciòn (_MSB_ADD:LSB_ADDR) **
;**         ByteRead. Lectura de un byte "datao" en direcciòn (_MSB_ADD:LSB_ADDR) **
;**         SequentialRead. Lectura de 16 byte en direcciòn (_MSB_ADD:LSB_ADDR) y los **
;**         pone en la direcciòn del apuntador FSR                              **
;**         Poll. Chack si fin de escritura ok                                  **
;**                                                                             **
;*********************************************************************
; PROGRAMA PRINCIPAL
; ==================
            LIST P=16F877A
            include "p16f877a.inc"     ; Include file for a PIC16F877A
  __CONFIG  _CP_OFF  &  _WDT_OFF  &  _BODEN_OFF  &  _PWRTE_ON  &  _HS_OSC  &  _WRT_OFF  &
_LVP_OFF & _CPD_OFF

;*************************************************************************
; DEFINICIONES
; ============
; Crysta1=10MHz, Tcy=400ns
; PORTC<3>=SCL
; PORTC<4>=SCA

#define    _write_ctrl          b'10100000'        ; byte de control para escritura
#define    _read_ctrl           b'10100001'        ; byte de control para lectura

;*************************************************************************
; VARIABALES
; ==========
            cblock     0x20                    ; separa espacio para varibales
datai       ; buffer de 1 byte Data input
datao       ; buffer de 1 byte Data output
bytecount ; contador para ciclos de byte
pollcnt     ; contador de ciclos de polling
loops       ; contador de loop de retardo
loops2      ; contador de loop de retardo
            endc

            cblock     0x2A
buffer      :16                    ; buffer de 16 datos
```


libros.ep-electropc.com

```
            endc

;*********************************************************************************
******
   org    0x000             ; vector de reset

start
   call   Init              ; initializa I2C
   ;call   ByteWrite         ; escribe un Byte
   ;call   ByteRead          ; lee un byte
   ;call   PageWrite         ; escribe una página (16 byte)
            movlw     buffer
            movwf     FSR
            call      SequentialRead                ; lee en forma secuencial, los pone en FSR

   nop
   goto   $-1               ; loop infinito

;*****************************************************************
; Inicializa el módulo MSSP para I2C Maestro
Init
   bcf    STATUS,RP1        ; Select Bank 01
   bsf    STATUS,RP0
   movlw  b'11111111'
   movwf  TRISC             ; Set PORTC to all inputs

   clrf   SSPSTAT           ; Disable SMBus inputs
   bsf    SSPSTAT,SMP       ; Disable slew rate control

   movlw  0x18              ; Load 0x18 into WREG
   movwf  SSPADD            ; Setup 100 kHz I2C reloj

   clrf   SSPCON2           ; Clear control bits
   bcf    STATUS,RP0        ; Select Bank 00
   movlw  b'00101000'
   movwf  SSPCON            ; Enable SSP, select I2C Master mode

   bcf    PIR1,SSPIF        ; Clear SSP interrupt flag
   bcf    PIR2,BCLIF        ; Clear Bit Collision flag
   retlw  0

;*****************************************************************
; Condicion Bit de Start
BSTART
   bcf    STATUS,RP1
   bcf    STATUS,RP0        ; selecciona Banco 0
   bcf    PIR1,SSPIF        ; borra bandera de interrupciòn SSP

   bsf    STATUS,RP0        ; selecciona Banco 1
   bsf    SSPCON2,SEN       ; generate condicion Start
   bcf    STATUS,RP0        ; selecciona Banco 0

bstart_wait
   btfss  PIR1,SSPIF        ; check si se completo operaciòn
   goto   bstart_wait       ; no, continua check
```


libros.ep-electropc.com

```
   retlw   0

;*****************************************************************
; Condicion Bit de Re-Start
BRESTART
   bcf     STATUS,RP1
   bcf     STATUS,RP0          ; selecciona Banco 0
   bcf     PIR1,SSPIF          ; borra bandera de interrupciòn SSP

   bsf     STATUS,RP0          ; selecciona Banco 1
   bsf     SSPCON2,RSEN        ; generate condicion Restart
   bcf     STATUS,RP0          ; selecciona Banco 0

brestart_wait
   btfss   PIR1,SSPIF          ; check si se completo operaciòn
   goto    brestart_wait       ; no, continua check

   retlw   0

;*****************************************************************
; Condicion Bit de Stop
BSTOP
   bcf     STATUS,RP1
   bcf     STATUS,RP0          ; selecciona Banco 0
   bcf     PIR1,SSPIF          ; borra bandera de interrupciòn SSP

   bsf     STATUS,RP0          ; selecciona Banco 1
   bsf     SSPCON2,PEN         ; generate condicion Stop
   bcf     STATUS,RP0          ; selecciona Banco 0

bstop_wait
   btfss   PIR1,SSPIF          ; check si se completo operaciòn
   goto    bstop_wait          ; no, continua check

   retlw   0

;*****************************************************************
; Transmision de un byte que está en "datao"
; se pone codigo para chek si hay ACK
TX
   bcf     STATUS,RP1
   bcf     STATUS,RP0          ; selecciona Banco 0
   bcf     PIR1,SSPIF          ; borra bandera de interrupciòn SSP

   movf    datao,W             ; copia datao -> W
   movwf   SSPBUF              ; escribe dato a Buffer

tx_wait
   btfss   PIR1,SSPIF          ; check si se completo operaciòn
   goto    tx_wait             ; no, continua check

;  bsf     STATUS,RP0          ; selecciona Banco 1
;  btfsc   SSPCON2,ACKSTAT     ; Check si bit ACK fue recivido
;  goto    ackfailed           ; no, ACK no recivido

   retlw   0
```


libros.ep-electropc.com

```
;*****************************************************************
; Rececepciòn de un byte que se pone en "datai"
; se genera bit de ACK
RX
   bcf    STATUS,RP1
   bcf    STATUS,RP0        ; selecciona Banco 0
   bcf    PIR1,SSPIF        ; borra bandera de interrupciòn SSP

   bsf    STATUS,RP0        ; selecciona Banco 1
   bsf    SSPCON2,RCEN      ; inicia recepcion de un byte
   bcf    STATUS,RP0        ; selecciona Banco 0

rx_wait
   btfss  PIR1,SSPIF        ; check si se completo operaciòn
   goto   rx_wait           ; no, continua check

   movf   SSPBUF,W          ; copia byte de buffer a W
   movwf  datai             ; copia W -> datai

   bcf    PIR1,SSPIF        ; borra bandera de interrupciòn SSP

   bsf    STATUS,RP0        ; selecciona Banco 1
   bsf    SSPCON2,ACKEN     ; genera bit ACK/NO ACK
   bcf    STATUS,RP0        ; selecciona Banco 0

rx_wait2
   btfss  PIR1,SSPIF        ; check si se completo operaciòn
   goto   rx_wait2          ; no, continua check

   retlw  0

;*****************************************************************
; Escribe un byte _DATA en la direcciòn (_MSB_ADDR : _LSB_ADDR)
; genera todo el ciclo de escritura: Start, Control, MSB addr, LSB addr, Data, Stop
; al final chek si escritura completa
ByteWrite
   call   BSTART            ; genera condicion Start

                            ; envia byte de control
   bcf    STATUS,RP0        ; selecciona Banco 0

;                  _write_ctrl                    ; byte de control escritura
   movlw  _write_ctrl       ; carga byte control -> W
   movwf  datao             ; W -> datao
   call       TX            ; Rutina Trasnmita byte control

                            ; envia MSB byte de direcciòn
   bcf    STATUS,RP0        ; selecciona Banco 0
;                  _MSB_ADDR                      ; byte MSB de direcciòn
   movlw  0x08              ; carga MSB byte de direcciòn -> W
   movwf  datao             ; W -> datao
   call   TX                                       ; Rutina Trasnmita MSB byte direcciòn

                            ; envia LSB byte de direcciòn
   bcf    STATUS,RP0        ; selecciona Banco 0
```


libros.ep-electropc.com

```
;         _LSB_ADDR                    ; byte LSB de direcciòn
   movlw   0x00              ; carga LSB byte de direcciòn -> W
   movwf   datao             ; W -> datao
   call    TX              ; Rutina Trasnmita LSB byte direcciòn

                        ; envia byte de dato
   bcf     STATUS,RP0        ; selecciona Banco 0
;                _DATA                        ; byte dato
   movlw   0xBB              ; carga byte de dato -> W
   movwf   datao             ; W -> datao
   call    TX              ; Rutina Trasnmita byte dato

   call    BSTOP             ; generate condicion Stop
   call    Poll            ; Poll escritura completa

   retlw   0

;*****************************************************************
; Escribe 16 byte (page) en la direcciòn (_MSB_ADDR : _LSB_ADDR)

PageWrite
   bcf     STATUS,RP1
   bcf     STATUS,RP0        ; selecciona Banco 0

;                _PAGE_DATA                   ; cantidad de datos por pagina
   movlw   .16                          ; cantidad de datos por pagina -> W
   movwf   bytecount         ; W-> contador de byte

   call    BSTART            ; genera condicion Start
                        ; envia byte de control
   bcf     STATUS,RP0        ; selecciona Banco 0

;                _write_ctrl                  ; byte de control escritura
   movlw   _write_ctrl               ; carga byte control -> W
   movwf   datao             ; W -> datao
   call    TX              ; Rutina Trasnmita byte control

                        ; envia MSB byte de direcciòn
   bcf     STATUS,RP0        ; selecciona Banco 0
;                _MSB_ADDR                    ; byte MSB de direcciòn
   movlw   0x0A              ; carga MSB byte de direcciòn -> W
   movwf   datao             ; W -> datao
   call    TX                                          ; Rutina Trasnmita MSB byte direcciòn

                         ; envia LSB byte de direcciòn
   bcf     STATUS,RP0        ; selecciona Banco 0
;                _LSB_ADDR                    ; byte LSB de direcciòn
   movlw   0xA0              ; carga LSB byte de direcciòn -> W
   movwf   datao             ; W -> datao
   call    TX              ; Rutina Trasnmita LSB byte direcciòn


   bcf     STATUS,RP0        ; selecciona Banco 0
txbyte                  ; Este loop transfiere los numeros del 16 al 1
   movf    bytecount,W       ; usa (bytecount -> W) como dato
;                _DATA                        ; byte dato
```


libros.ep-electropc.com

```
   movwf   datao              ; W -> datao
   call    TX              ; Rutina Trasnmita byte dato

   bcf     STATUS,RP0         ; selecciona Banco 0
   decfsz  bytecount,F        ; Check fin de loop
   goto    txbyte             ; no, continue loop

   call    BSTOP              ; generate condicion Stop
   call    Poll            ; Poll escritura completa

   retlw   0

;*****************************************************************
; Lee un byte _DATA en la direcciòn (_MSB_ADDR : _LSB_ADDR)
; genera todo el ciclo de escritura: Start, Control, MSB addr, LSB addr, Control, Data, Stop
; al final genera No ACK para completar lectura
ByteRead
   call    BSTART             ; genera condicion Start

                        ; envia byte de control
   bcf     STATUS,RP0         ; selecciona Banco 0
;                  _write_ctrl                          ; byte de control escritura
   movlw   _write_ctrl        ; carga byte control -> W
   movwf   datao              ; W -> datao
   call    TX              ; Rutina Trasnmita byte control

                        ; envia MSB byte de direcciòn
   bcf     STATUS,RP0         ; selecciona Banco 0
;                  _MSB_ADDR                            ; byte MSB de direcciòn
   movlw   0x08               ; carga MSB byte de direcciòn -> W
   movwf   datao              ; W -> datao
   call    TX                                                  ; Rutina Trasnmita MSB byte direcciòn

                        ; envia LSB byte de direcciòn
   bcf     STATUS,RP0         ; selecciona Banco 0
;                  _LSB_ADDR                            ; byte LSB de direcciòn
   movlw   0x00               ; carga LSB byte de direcciòn -> W
   movwf   datao              ; W -> datao
   call    TX              ; Rutina Trasnmita LSB byte direcciòn

   call    BRESTART           ; genera condicion Restart

                        ; envia byte de control
   bcf     STATUS,RP0         ; selecciona Banco 0
;                  _read_ctrl                           ; byte de control lectura
   movlw   _read_ctrl         ; Load control byte for read
   movwf   datao              ; W -> datao
   call    TX              ; Rutina Trasnmita byte control

                        ; lea byte de dato
   bsf     STATUS,RP0         ; selecciona Banco 1
   bsf     SSPCON2,ACKDT      ; envie bit NO ACK
   call    RX              ; lea byte data

   call    BSTOP              ; genera condicion Stop
```


libros.ep-electropc.com

```
   retlw  0

;*****************************************************************
; Lee en forma secuencial 16 datos de la posiciòn (_MSB_ADDR : _LSB_ADDR)
; los coloca a partir de la direcciòn del apuntador FSR
SequentialRead
   bcf    STATUS,RP0        ; selecciona Banco 0
   movlw  .16
   movwf  bytecount         ; initialize countador de 16 bytes

   call   BSTART            ; genera condicion Start

                    ; envia byte de control
   bcf    STATUS,RP0        ; selecciona Banco 0
;                  _write_ctrl                    ; byte de control escritura
   movlw  _write_ctrl       ; carga byte control -> W
   movwf  datao             ; W -> datao
   call   TX              ; Rutina Trasnmita byte control

                    ; envia MSB byte de direcciòn
   bcf    STATUS,RP0        ; selecciona Banco 0
;                  _MSB_ADDR                      ; byte MSB de direcciòn
   movlw  0x0A              ; carga MSB byte de direcciòn -> W
   movwf  datao             ; W -> datao
   call   TX                                              ; Rutina Trasnmita MSB byte direcciòn

                    ; envia LSB byte de direcciòn
   bcf    STATUS,RP0        ; selecciona Banco 0
;                  _LSB_ADDR                      ; byte LSB de direcciòn
   movlw  0xA0              ; carga LSB byte de direcciòn -> W
   movwf  datao             ; W -> datao
   call   TX              ; Rutina Trasnmita LSB byte direcciòn

   call   BRESTART          ; genera condicion Restart

                    ; envia byte de control
   bcf    STATUS,RP0        ; selecciona Banco 0
;                  _read_ctrl                     ; byte de control lectura
   movlw  _read_ctrl        ; Load control byte for read
   movwf  datao             ; W -> datao
   call   TX              ; Rutina Trasnmita byte control
rxbyte
                    ; lea byte de datos
   bcf    STATUS,RP0        ; selecciona Banco 0
   decfsz bytecount,F       ; check si lopp termino
   goto   continue          ; no, continue
   bsf    STATUS,RP0        ; selecciona Banco 1
   bsf    SSPCON2,ACKDT     ; envie bit NO ACK
   call   RX              ; lea byte de dato
   movf   datai,W                 ; copie datai -> W
   movwf  INDF                    ; W -> INDF gurade en forma indirecta

   call   BSTOP             ; genera condicion Stop
   retlw  0

continue
```


libros.ep-electropc.com

```
   bsf    STATUS,RP0        ; selecciona Banco 1
   bcf    SSPCON2,ACKDT     ; envie bit ACK
   call   RX                ; Input data from device

   movf    datai,W                  ; copie datai -> W
   movwf   INDF                     ; W -> INDF gurade en forma indirecta
   incf    FSR                      ; incremente apuntador

   goto   rxbyte            ; continue loop

;*****************************************************************
; Poll si la memoria EEPROM esta libre, chek el bit ACK , que indica que el ciclo
; interno de escritura termino
Poll
   bcf    STATUS,RP0        ; selecciona Banco 0
   movlw  .40
   movwf  pollcnt           ; pone maximo ciclo para poll

polling
   call   BRESTART          ; genera condicion Restart

                        ; envia byte de control
   bcf    STATUS,RP0        ; selecciona Banco 0
;                _write_ctrl                     ; byte de control escritura
   movlw  _write_ctrl       ; carga byte control -> W
   movwf  datao             ; W -> datao
   call   TX              ; Rutina Trasnmita byte control

   bsf    STATUS,RP0        ; selecciona Banco 1
   btfss  SSPCON2,ACKSTAT   ; bit ACK bit=0?
   goto   exitpoll          ; si, pare pool
                        ; no, check si ya son40 veces
   bcf    STATUS,RP0        ; selecciona Banco 0
   decfsz pollcnt,F         ; decremente contador de pool=0?
   goto   polling           ; no, poll denuevo
;  goto   TimedOut          ; si, no responde
                        ; termino tiempo, tome una acciòn
exitpoll
   call   BSTOP             ; genera condicion Stop
   retlw  0

   END
```


libros.ep-electropc.com

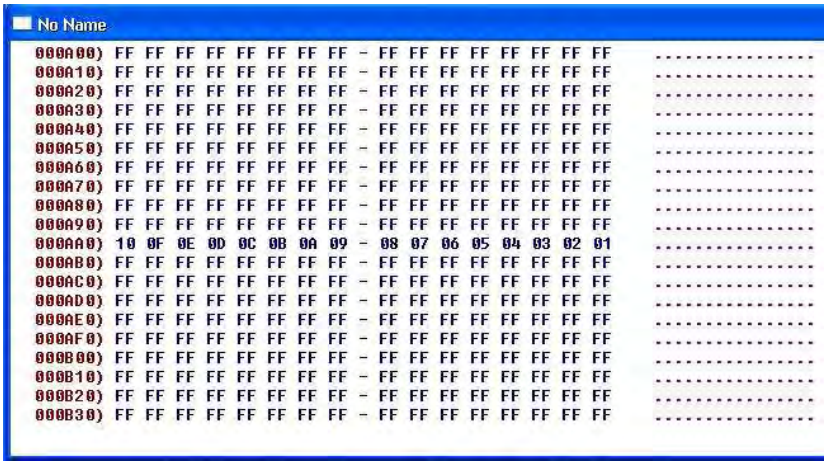


Figura 25.10 Escritura en memoria EEPROM.

Utilizando la rutina **write page** con la dirección **0x0AA0** podemos escribir los valores de **10h** a **01h** en las posiciones de la memoria **24LC32** de **0x0AA0** a **0x0AAF**, como apreciamos en la Figura 25.10.

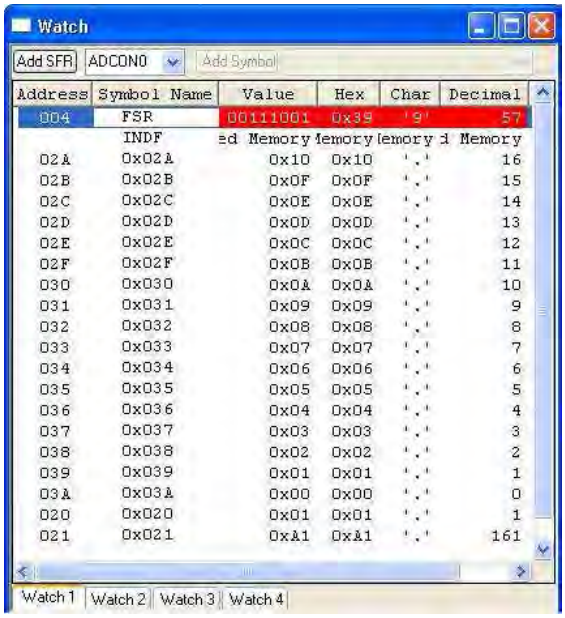


Figura 25.11 Lectura en Memoria EEPROM.

libros.ep-electropc.com

Con la rutina **sequential read** se pueden leer los valores de la memoria **24LC32** en las posiciones de **0xAA0** a **0x0AAF**, y ponerlos en la dirección donde apunta el FSR; para este caso, las posiciones de **RAM** de **0x02A** a **0x3A**; observemos cómo corresponden a los valores vistos anteriormente. En la Figura 25.11 se muestra la ventana **watch** con los registros FSR, INDF y las direcciones 0x02A a 0x39.

25.3 MEMORIA EEPROM I2C 24LC32

Este dispositivo es una memoria serial EEPROM que funciona en **modo esclavo** con protocolo I²C. La memoria 24LC32 tiene una capacidad de 32 Kbit o 4 096 bytes (00h a FFFh) posiciones de memoria, con memoria caché de 8 páginas (1 page = 8 byte) que son 64 bytes (3Fh) posiciones de memoria. Su voltaje de alimentación es desde 2.5 a 5.5 voltios y su consumo de corriente en lectura es de 1mA, y en *standby* es de 1 µA.

25.3.1 PINOUT

En la Figura 25.12 se presenta un diagrama de bloques y el *pinout*.

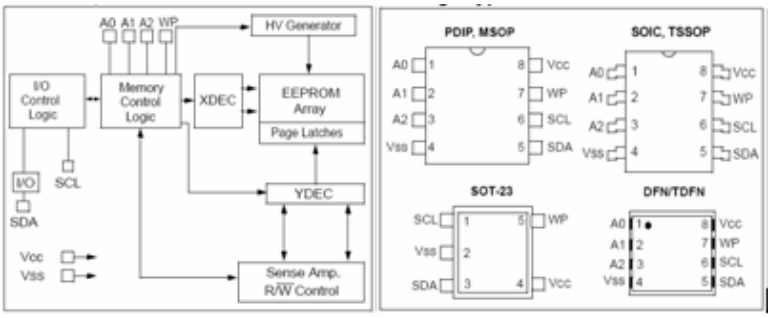


Figura 25.12 Memoria 24LC32. Diagrama de bloques y pinout.

Tabla 25.6 Pinout.

Pin	Nombre	Descripción
1	A0	Selección del chip A0
2	A1	Selección del chip A1
3	A2	Selección del chip A2
4	VSS	Tierra GND
5	SDA	Datos
6	SCL	Clock
7	WP	Protección de escritura
8	VCC	Alimentación

libros.ep-electropc.com

25.3.2 RECOMENDACIONES DE HARDWARE

Estas son las recomendaciones para el Hardware opere bien:

- Los pines A0, A1, A2 configuran la **selección del chip,** cuando son puestos a VCC (5V) son configurados como "1", y cuando son puestos a VSS = 0V son configurados como "0".

- La memoria siempre es de lectura, y con el pin **WP** (*Write Protect*) evita la escritura cuando está a VCC, y la permite cuando se pone a VSS o GND o 0V, hay que colocar con una resistencia de polarización de 10 K, para menejo.

- Entre los pines de alimentación, lo más cercano a ellos se debe colocar un C = 0.1 µF.

- Sobre los pines SDA y SCL debe ponerse una resistencia de polarización típica de 10 K para 100 kHz y 2 K para 400 kHz.

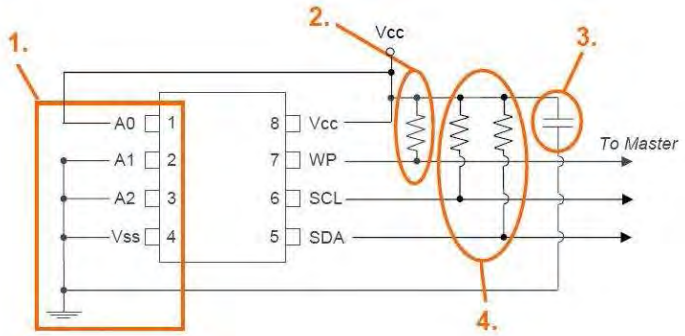


Figura 25.13 Recomendaciones de hardware.

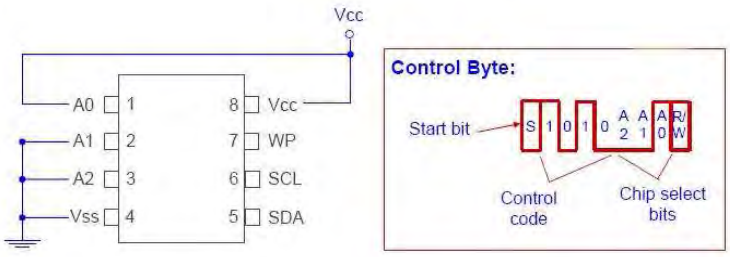
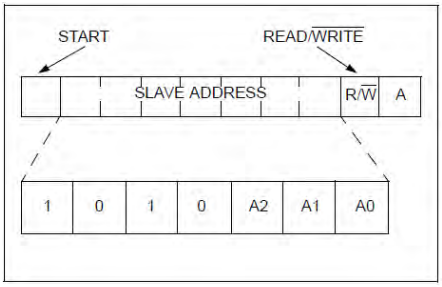


Figura 25.14 Direccionamiento de hardware

libros.ep-electropc.com

25.3.3 PALABRA DE CONTROL

En la Figura 25.15 se puede observar cómo los pines configurados en hardware A2 = 0, A1 = 0, A0 = 1 son los bits que se deben poner en la palabra de control, para seleccionar el chip.



Operation	Control Code	Device Select	R/W
Read	1010	Device Address	1
Write	1010	Device Address	0

Figura 25.15 Palabra de control.

En la Figura 25.15 vemos la secuencia de bits para la palabra de control, el bit de *start* (**S**), los bits de control **1010**, los bits de selección de chip **A2, A1, A0**, el bit de lectura-escritura *R/W* y el bit de *acknowledge* (ACK**)**.

25.3.4 DIRECCIONAMIENTO

Para el direccionamiento de la memoria 24LC32 que tiene una capacidad de 32 Kbits = 8 bit * (4 K =) 4 * 1 024 = 32 768 bits, o en bytes 32 768 bits / 8 bit = 4 096 bytes, o 4 Kbytes; se requeire de 12 bits en dos bytes de dirección: ADDRESH y ADDRESL (Figura 25.16), observemos la posición de los bits en cada byte (Figura 25.17).

ADRESH								ADRESL							
MSB			<- BYTE HIGH ->				LSB	MSB			<- BYTE LOW ->				LSB
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	0	0	0	1	1	0	1	1	1	0	0	0	1	1	1	HEX	DEC
0				D				C				7				h 0DC7	d 3527

Figura 25.16 Bytes de dirección ADDRESH y ADDRESL.

libros.ep-electropc.com

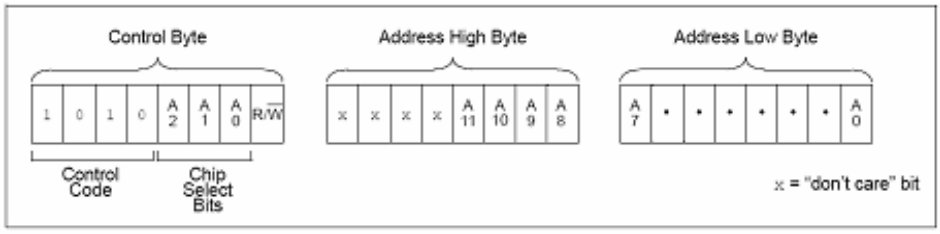


Figura 25.17 Secuencia de control y dirección.

25.3.5 OPERACIÓN DE ESCRITURA

Podemos realizar dos tipos de escritura sobre la memoria:

1. Escritura de 1 byte *Write Byte*
2. Escritura de una página *Write Page*

25.3.5.1 ESCRITURA DE UN BYTE

En la Figura 25.18 vemos el diagrama de tiempos del procedimiento para escribir un byte en la memoria EEPROM.

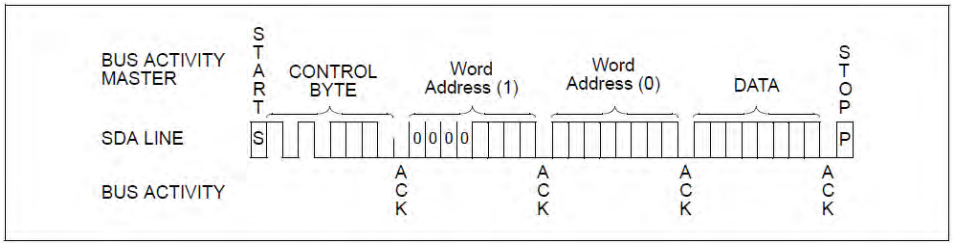


Figura 25.18 Secuencia de escritura de un byte.

La secuencia de escritura de un byte contiene: la palabra de control, la dirección y el dato. Cuando llega la palabra de control al receptor esclavo direccionado, éste genera un *acknowledge* en el noveno bit, indicando al **maestro** que recibió el comando. Así, la siguiente palabra transmitida es la dirección alta, y luego la dirección baja. La memoria recibe esta dirección y la guarda como apuntador, y envía un *acknowledge* al final de cada byte, indicando al **maestro** la recepción de la dirección. Luego de esto, el **maestro** transmite el byte que se va a escribir, y a esto la memoria responde con el último *acknowledge*, y el **maestro** al recibirlo genera un *stop* para indicar que terminó el comando. Desde aquí se genera un tiempo de escritura a nivel interno, en la memoria, la cual no genera señales externas.

libros.ep-electropc.com

25.3.5.2 ESCRITURA DE UNA PÁGINA

En la Figura 25.19 vemos el diagrama de tiempos del procedimiento para escribir una página u 8 bytes en la memoria EEPROM. Recuerde que la memoria caché es de 8 páginas (1 página = page = 8 bytes) que son 64 bytes (3Fh) posiciones de memoria. Se pueden escribir hasta 1 memoria caché completa, o 64 bytes, u 8 páginas (*page*) antes que la memoria tome el tiempo de escritura (4 – 8 ms). Se pueden escribir un total de 64 memorias caché = 64 * 64 bytes = 4 096 bytes = 4K bytes. En medio de cada escritura de una memoria caché se debe tomar el tiempo de escritura que toma la memoria (4 – 8 ms).

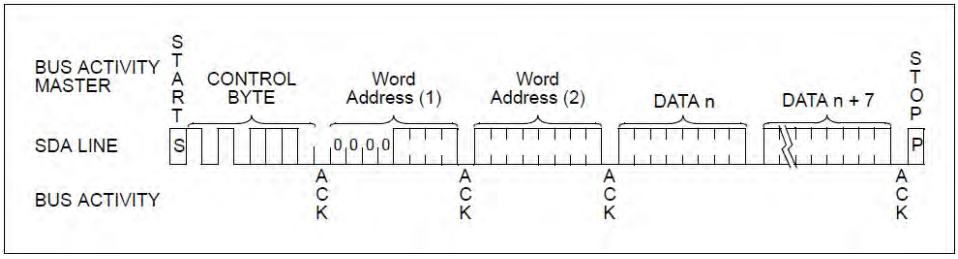


Figura 25.19 Secuencia de escritura de una página.

La secuencia de escritura de una página (*page*), que son 8 bytes, contiene la palabra de control, la dirección alta y baja y los bytes de datos. La secuencia es exactamente la misma hasta la escritura del primer byte; de allí en adelante el maestro envía un bit de *acknowledge* (noveno bit) por cada byte recibido desde la memoria, hasta que llega al byte 8, en el cual se envían un *no-acknowledge* y una condición de *stop*.

Cuando la memoria recibe los bytes de dirección se guardan en un contador de direcciones que incrementa la dirección con cada byte enviado; esto ocurre hasta por 64 bytes (8 páginas: *pages*) que es la memoria caché, si se envía un siguiente byte que se sobrescribe en el primero en la memoria caché, y también los que sigan a continuación. La memoria caché se guarda al recibir un *stop*, en un procedimiento de escritura interna que dura 2ms por página, durante el cual el SCL se mantiene en bajo.

25.3.6 OPERACIÓN DE LECTURA

Se pueden realizar tres tipos de lectura sobre la memoria:

1. Lectura de dirección actual (*current address read*)
2. Lectura aleatoria (*random read*)
3. Lectura secuencial (*sequential read*)

libros.ep-electropc.com

25.3.6.1 LECTURA DE DIRECCIÓN ACTUAL

La memoria tiene un contador de dirección que mantiene la última dirección accedida más uno; así, al enviar solamente la palabra de control de lectura, la memoria devuelve el dato que se encuentra en la dirección actual (Figura 25.20).

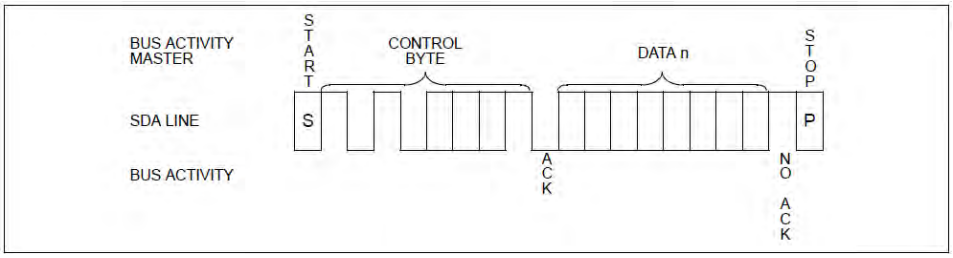


Figura 25.20 Lectura de dirección actual.

El **maestro** empieza con el bit de *start*, luego envía la palabra de control de lectura, y recibe un *acknowledge* de la memoria; allí el **maestro** genera los 8 relojes para el dato que la memoria envía; en seguida el maestro le envía un *no-acknowledge* y el bit de *stop* para terminar la secuencia. Otra alternativa de esta lectura es la secuencial, donde podría el **maestro** seguir recibiendo datos si no genera *no-acknowledge*, y no envía el bit de *stop* para terminar la secuencia; a cambio de esto espera un *acknowledge* de la memoria y genera los 8 relojes para el dato siguiente que la memoria envía, y así sucesivamente hasta terminar con un *no-acknowledge* y el bit de *stop* para terminar la secuencia.

25.3.6.2 LECTURA ALEATORIA

Esta lectura es la de cualquier dirección de la memoria donde se encuentra un dato, la secuencia de lectura aleatoria de un byte contiene la palabra de control de escritura, la dirección alta y baja, de nuevo el bit de *start*, la palabra de control de lectura y el byte de dato (Figura 25.21). El maestro empieza con el bit de *start*, luego envía la palabra de control de escritura, y recibe un *acknowledge* de la memoria; en seguida envía el byte alto de la dirección y espera un *acknowledge* de la memoria; después envía el byte bajo de la dirección y espera un *acknowledge* de la memoria; allí termina y genera un bit de *re-start* y envía de nuevo la palabra de control, pero de lectura, y espera un *acknowledge* de la memoria, y desde allí genera los 8 relojes para el dato que la memoria envía; a continuación el maestro le envía un *no-acknowledge* y envía el bit de *stop* para terminar la secuencia.

libros.ep-electropc.com

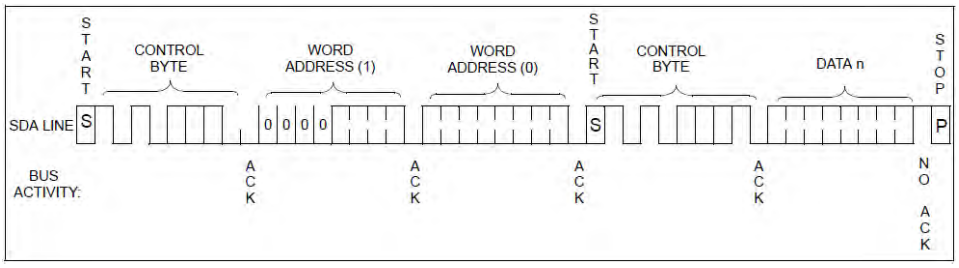


Figura 25.21 Lectura aleatoria.

25.3.6.3 LECTURA SECUENCIAL

Ésta es la lectura secuencial de datos de cualquier dirección de la memoria; la secuencia de lectura secuencial es la misma que la secuencia de la lectura aleatoria, sólo que se leen varios bytes, y contiene la palabra de control de escritura, la dirección alta y baja, de nuevo el bit de *start*, la palabra de control de lectura y los bytes de datos (Figura 25.22). Es de anotar que es la misma secuencia de la **lectura aleatoria** anterior con **n datos**, luego de enviar la dirección inicial desde donde se leerán los datos. Si queremos leer los **n datos** desde la última posición (última dirección + 1, que tiene almacenada la EEPROM) de manera secuencial, observemos esta secuencia en la Figura 25.22.

El maestro empieza con el bit de *start*, luego envía la palabra de control de escritura, y recibe un *acknowledge* de la memoria; y desde allí genera los 8 relojes para el dato que la memoria envía, y el **maestro** le envía *acknowledge* a la memoria confirmando el dato recibido; seguidamente genera de nuevo los 8 relojes para el dato siguiente y le envía otro *acknowledge*, y así sucesivamente hasta terminar con un *no-acknowledge*, y envía el bit de *stop* para terminar la secuencia.

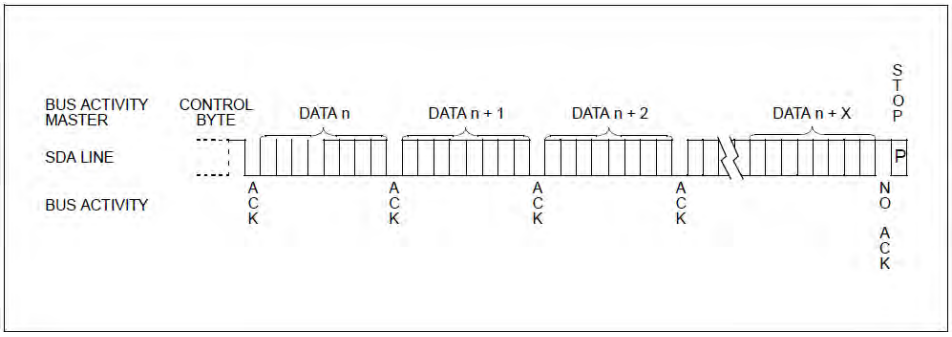


Figura 25.22 Lectura secuencial

libros.ep-electropc.com

NOTA: Es muy importante leer los *data sheets* de las memorias que se utilizarán para saber si usan o no los pines de direccionamiento de dispositivo *chip select bit* A2, A1, A0.

Según el tipo de memoria, también será el número de bytes de direccionamiento (1 o 2).

EVALUACIÓN

P.: 1. ¿Cuáles son los estados posibles del protocolo I2C?
R.:
Data valid (“dato válido”).
Bus not busy (“bus no ocupado”)
Start (“arranque”).
Stop (“parada”).
Acknowledge (“reconocimiento”).

P.: 2. ¿Cuáles son los pines que se usan en el protocolo I2C?
R.:
1. Reloj serial (SCL) – RC3/SCK/SCL.
2. Dato serial (SDA) – RC4/SDI/SDA.
3. GND.

P.: 3. Mencione algunos de los modos del módulo I2C?
R.:
I2C modo maestro, reloj = OSC/(4 * (SSPADD+1)).
I2C modo esclavo (7 bits de dirección).
I2C modo esclavo (10 bits de dirección).
I2C modo esclavo (7 bits de dirección), con interrupciones habilitadas bit de *start* y *stop*
I2C modo esclavo (10 bits de dirección), con interrupciones habilitadas bit de *start* y *stop*.
I2C modo maestro controlado por *firmware*, esclavo es habilitado.

P.: 4. ¿Con qué pines se direcciona la memoria 24LC32?
R.:
Pines A0, A1, A2-

P.: 5. ¿Con qué pin se protege la escritura en la memoria 24LC32?
R.:
WP (Write Protect).

P.: 6. Cuál es la capacidad en bytes y bits de la memoria 24LC32?

libros.ep-electropc.com

```
R.:

32 Kbits = 8 bits * 4 K = 4 * 1024 = 32768 bits, en bytes 32768 / 8
bits = 4096 bytes, 4 Kbytes.

P.: 7. ¿Cuántos bits se necesitan para direccionar 32 Kbits de la
memoria 24LC32?
R.:
Se necesitan:
12 bits
32 Kbits = 4 Kbytes
2^12 = 2^2 * 2`10 = 4 K

P.: 8. ¿Qué tipos de escritura hay en la memoria 24LC32?
R.:
Escritura de 1 byte (write byte).
Escritura de una página (write page).

P.: 9. ¿Cuántos tipos de lectura hay en la memoria 24LC32?
R.:
Lectura de dirección actual (current address read).
Lectura aleatoria (random read).
Lectura secuencial (sequential read).
```

26.0 MÓDULO INTERFAZ DE COMUNICACIÓN SERIAL (SCI USART)

El USART (*Universal Synchronous Asynchronous Receiver Transmitter*) es uno de dos módulos de comunicación, también llamado **SCI** (*Serial Communication Interface*). Antes de tratar sobre este módulo veremos la definición del estándar RS232.

26.1 ESTÁNDAR RS232

Hacia 1960, el Comité **EIA** (*Electronic Industries Association*) hizo una recomendación para el estándar de protocolo de comunicaciones denominado **EIA RS232C**, que consistía en poder comunicar dos equipos que estaban físicamente distanciados. Desde esa época se han actualizado en varias versiones. En aquellos años las *main frame* (**computadoras principales**) necesitaban intercambiar datos digitales y se comunicaban por la red telefónica a largas distancias, en forma de datos seriales a través de un MODEM telefónico. Estos MODEM se encuentran cercanos a las computadoras, las conexiones se aprecian en las Figuras 26.1 y 26.2.

libros.ep-electropc.com

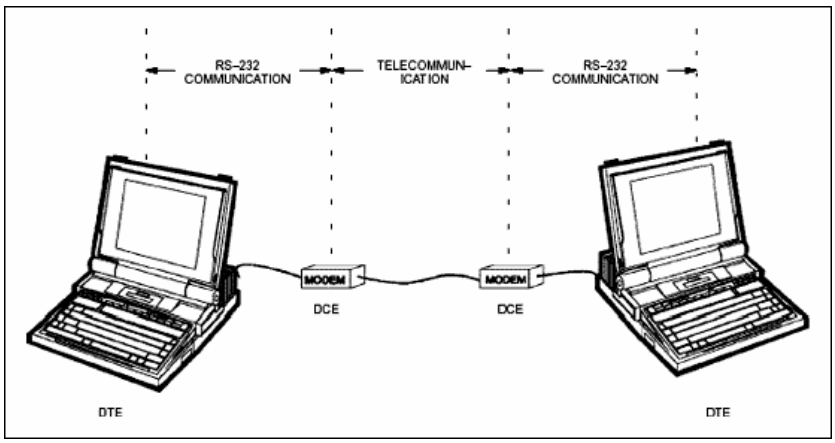


Figura 26.1 Comunicación PC (DTE) a PC (DTE).

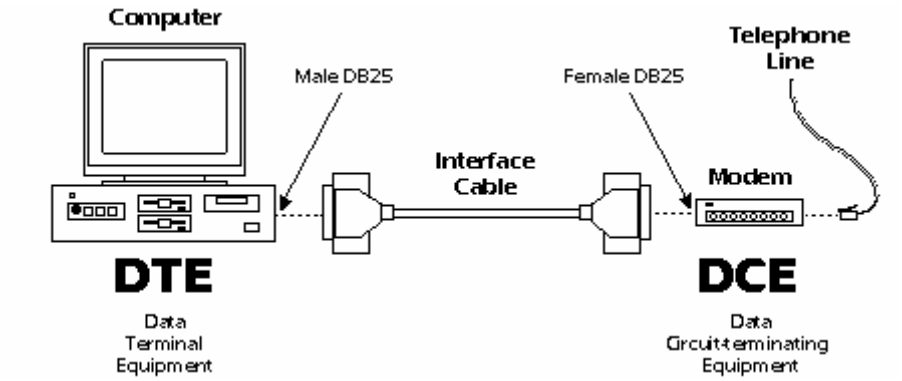


Figura 26.2 Comunicación PC (DTE) a MODEM (DCE).

La comunicación entre **DTE** (*Data Terminal Equipment*) y **DCE** (*Data Communication Equipment*) se hacía por un cable de 25 hilos DB25, y luego se pasó a 9 hilos DB9, usados muy ampliamente en los ratones (*mouses*) de las **PC** (computadoras personales) o en el puerto de comunicaciones RS232.

26.1.1 ASIGNACIÓN DE NOMBRES Y PINES

La asignación de los pines y sus nombres es como se muestra en las Figuras 26.3 y 2.6.4.

libros.ep-electropc.com

Signal		Origin			
Name	Abbreviation	DTE	DCE	DB-25	DE-9
Data Terminal Ready	DTR	●		20	4
Data Carrier Detect	DCD		●	8	1
Data Set Ready	DSR		●	6	6
Ring Indicator	RI		●	22	9
Request To Send	RTS	●		4	7
Clear To Send	CTS		●	5	8
Transmitted Data	TxD	●		2	3
Received Data	RxD		●	3	2
Common Ground	GND	*1	7	5	4
Protective Ground	PG	*1	1	-	-
		*1	COMUN		

Figura 26.3 Definición de pines DB25 y DB9.

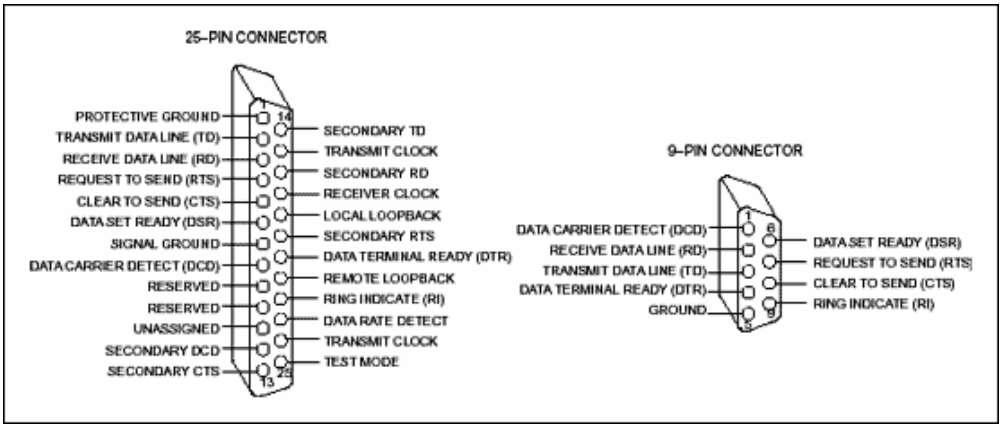


Figura 26.4 Pinout DB25 DB9.

En la comunicación serial se usan comúnmente los pines Tx, Rx y GND. En la Figura 26.5 a, b, c, d, f observamos cómo los tipos de conectores DB9, macho y hembra, se complementan al conectarse, de manera que cada pin se corresponde.

libros.ep-electropc.com



Figura 26.5 Conector DB9 hembra y macho

libros.ep-electropc.com

26.1.2 NIVELES LÓGICOS

Según el estándar de comunicación RS232, los niveles de voltaje de las señales son como se ilustra en la Figura 26.6, donde las señales tienen las características siguientes:

1. Los datos son en forma de lógica negativa, o sea, un voltaje entre +3V y +15V es un dato lógico "0", y un voltaje entre −3V y −15V es un dato lógico "1"
2. Un dato no válido es una señal entre +3V y −3V
3. Los niveles comúnmente usados son entre 9V y 12V
4. La distancia máxima del cable debe ser de 15 metros, pero puede alcanzar más distancia cuando la velocidad es baja.

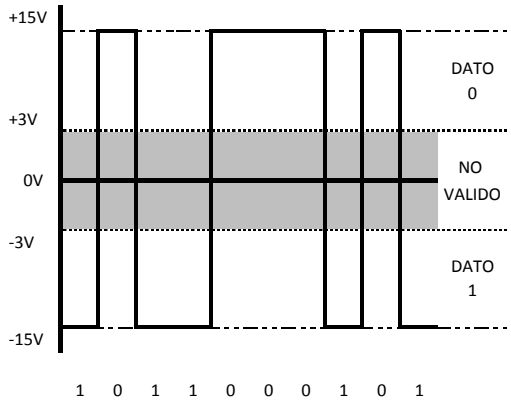


Figura 26.6 Niveles de voltaje y lógicos.

26.1.3 FORMATO DE TRAMA

Una trama es un conjunto de bits que constituyen un paquete de información o datos. Está conformada por:

1. *Start* (inicio o bit de arranque) es la señal de voltaje que cambia de −15V a +15V, o sea de un nivel lógico de "1" a "0"; esto marca el inicio de la trama de transmisión o recepción, a partir de este momento todos los demás bits estarán espaciados en el tiempo de acuerdo con la velocidad escogida.
2. *Data* (datos 8 bits o bytes) es la información como tal; la forma más usual en el orden de los bits transmitidos o recibidos es *less significant bit,* donde el primer bit es el LSB.

libros.ep-electropc.com

3. *Parity* (paridad) es un bit opcional que se envía luego del último bit de dato.

   Paridad par. Si la suma de los bits de datos es par, el bit de paridad par es 1, si es impar, el bit de paridad par es 0.

   Paridad impar. Si la suma de los bits de datos es impar, el bit de paridad impar es 1, si es par, el bit de paridad impar es 0.

   Ejemplo: el byte de datos **"0110 1010"** suma 4.

   Para **paridad par** el 4 es par, el bit de **paridad par = "1".**
   Para **paridad impar** el 4 es par, el bit de **paridad impar = "0".**

Sin paridad. **no** se utiliza el bit de paridad.

4. *Stop* (parada) es el bit de fin de transmisión; se pueden generar 1, 1.5, 2 bits de parada. En este estado es como queda la línea para indicar que no hay transmisión.

En la Figura 26.7 aparece una trama que contiene 1 bit de *start*, 8 bits de **datos;** no hay bit de **paridad** y 1 bit de *stop*. Esto se traduce como **8N1**, donde **8** es el número de datos; **N,** la paridad (no hay), y **1,** el bit de parada.

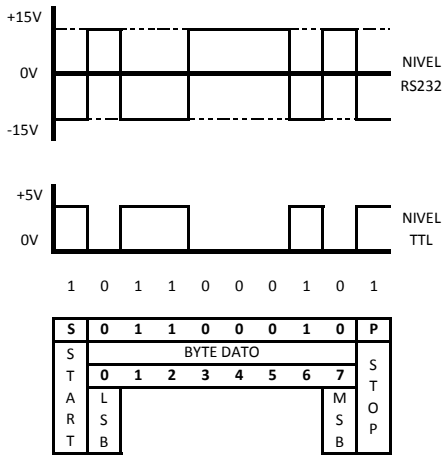


Figura 26.7 Niveles de voltaje y lógicos.

libros.ep-electropc.com

26.1.4 CIRCUITO INTEGRADO

Uno de los circuitos integrados que realiza esta función de intercambiador de niveles lógicos para el estándar RS232, es el MAX3232 que vemos en la Figura 26.8a, b, donde se aprecia el *pinout* en la disposición de pines de board (package) y esquemático.

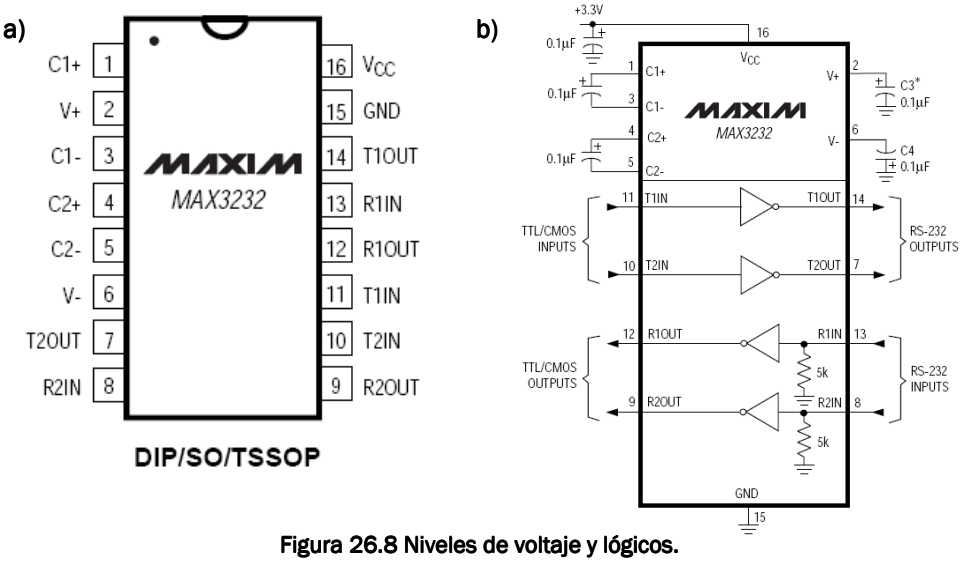


Figura 26.8 Niveles de voltaje y lógicos.

26.1.5 CONEXIÓN CIRCUITO INTEGRADO

En el sistema de desarrollo **PIC16F8XX**, se encuentra el circuito RS232 completo que contiene un circuito integrado MAX232, los condensadores de polarización, los leds de indicación DL4 Tx y DL5 Rx, el conector P3 con los pines +5V, RXM, TXM, GND, y el conector P2 que es un DB9 con sus pines TXPC RXPC GND; apreciamos esto en el circuito Board (Figura 26.9), y el circuito esquemático (Figura 26.10 a, b).

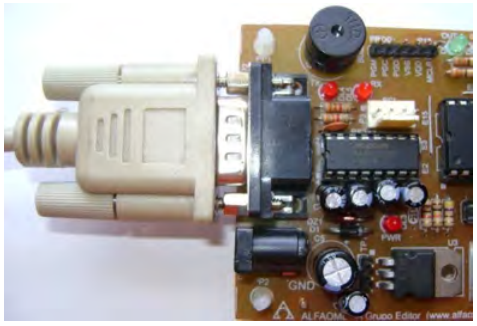


Figura 26.9 PIC16F8XX circuito RS232 integrado en board.

libros.ep-electropc.com

a)

b)

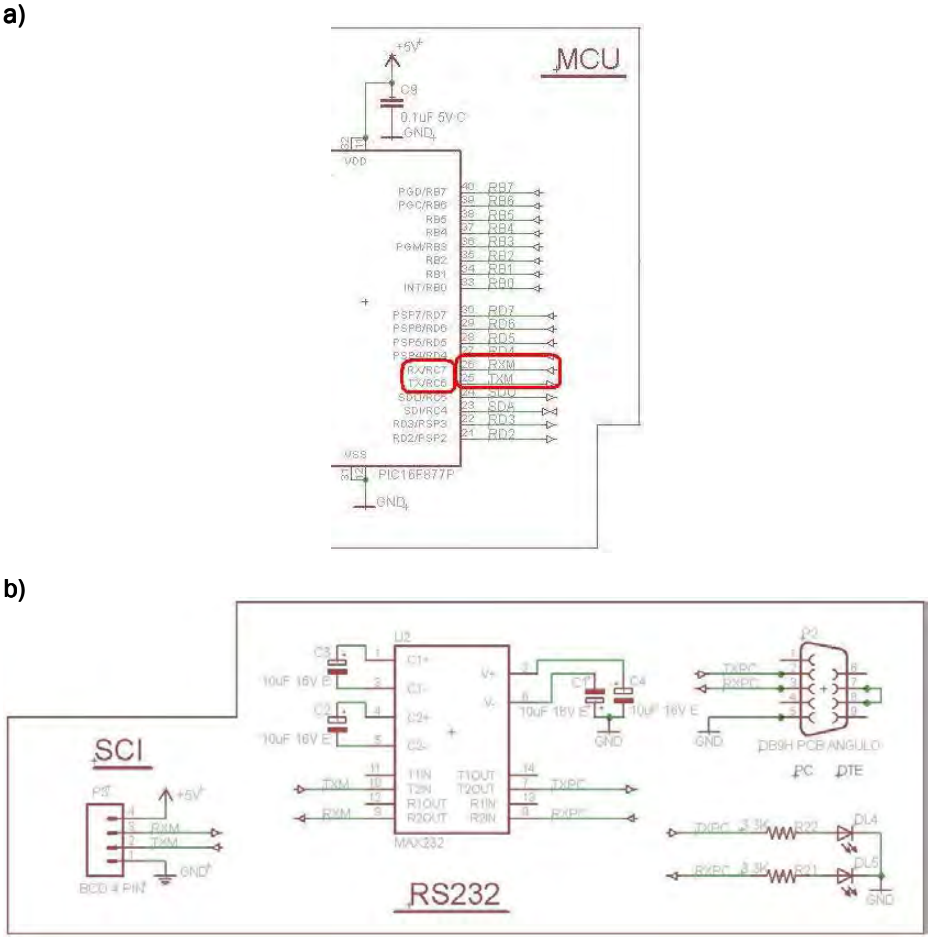


Figura 26.10 PIC16F8XX circuito RS232 esquemático.

26.2 MÓDULO SCI

El SCI está diseñado para comunicarse transmitiendo y recibiendo datos, serialmente, con periféricos fuera de la tarjeta principal. Puede transferir tramas de datos de 8 o 9 bits y detectar errores cuando los datos son sobrescritos o incorrectamente tramados (puestos en la trama). Puede generar interrupciones cuando ocurre una recepción (o se completa una transmisión), y contiene buffer de datos que simplifican los requerimientos de temporización del software que controla la USART.

libros.ep-electropc.com

Algunas partes tienen un direccionamiento USART que usa el bit 9 de los bits de datos para distinguir entre dirección y datos en la recepción. Esto permite un filtrado simple de datos de entrada y a menudo se usa en comunicaciones sobre el estándar RS485. Puede ser configurado en los modos siguientes:

1. Asíncrono (*full-duplex*)
2. Síncrono – maestro (*half duplex*)
3. Síncrono – esclavo (*half duplex*)

Para la configuración se utilizan el bit SPEN del registro RCSTA<7> y los bits en el registro TRIS<7:6>, que configuran los pines RC6/TX/CK y RC7/RX/DT. Hay dos módulos que deben ser atendidos antes de abordar el funcionamiento del SCI, porque tienen conceptos que atañen al desempeño del SCI; estos módulos son:

- Generador de tasa de baudios o *Baud Rate Generator* (**BRG**)
- Circuito de muestreo o *sampling circuit*

26.2.1 EL GENERADOR DE TASA DE BAUDIOS O BAUD RATE GENERATOR (BRG)

El BRG soporta ambos modos de la USART: **asíncrono** y **síncrono**, y está dedicado al generador de tasa de baudios de 8 bits. El registro SPBRG controla el periodo del temporizador de **corrida libre** (*free-running*) de 8 bits. En **modo asíncrono**, el bit BRGH del registro TXSTA<2>, también controla la tasa de baudios. En **modo síncrono** o **sincrónico** el bit BRGH es ignorado. En la Tabla 26.1 se muestra la fórmula de cálculo de la tasa de baudios para los diferentes modos de la USART, que aplican en modo maestro (reloj interno).

Tabla 26.1 Fórmula de tasa de baudios (baud rate).

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = FOSC/(64 (X + 1))	Baud Rate = FOSC/(16 (X + 1))
1	(Synchronous) Baud Rate = FOSC/(4 (X + 1))	N/A

Legend: X = value in SPBRG (0 to 255)

Dada la tasa de baudios deseada y el Fosc, se puede calcular el valor entero más cercano para el registro SPBRG, usando la fórmula de la Tabla 26.1, se puede determinar el error en la **tasa de baudios** (*baud rate formula*), como se aprecia en la Tabla 26.2.

libros.ep-electropc.com

26.2.1.1 EJEMPLO 1

En el ejemplo (Tabla 26.2) se muestra el error de cálculo de la tasa de baudios.

Fosc = 16 MHz
Desired Baud Rate = 9600
BRGH = 0
SYNC = 0

Example 18-1: Calculating Baud Rate Error

Desired Baud rate	=	Fosc / (64 (X + 1))
9600	=	16000000 / (64 (X + 1))
X	=	⌊25.042⌋ = 25
Calculated Baud Rate	=	16000000 / (64 (25 + 1))
	=	9615
Error	=	(Calculated Baud Rate - Desired Baud Rate) / Desired Baud Rate
	=	(9615 - 9600) / 9600
	=	0.16%

Tabla 26.2 Cálculo de error de la tasa de baudios.

Puede ser ventajoso usar la alta tasa de baudios (BRGH = 1) siempre que sean baudios de reloj lentos. Esto porque la ecuación **Fosc/(16(x+1))** puede reducir el error en la tasa de baudios en algunos casos. Escribir un valor nuevo al registro SPBRG mientras se está transmitiendo o recibiendo, puede causar que el temporizador BRG sea puesto a uno (o borrado). Esto asegura que el BRG no espera un sobreflujo (*overflow*) antes de la salida de la nueva tasa de baudios.

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	1.221	1.75	255	1.202	0.17	207	1.202	0.17	129
2.4	2.404	0.17	129	2.404	0.17	103	2.404	0.17	64
9.6	9.766	1.73	31	9.615	0.16	25	9.766	1.73	15
19.2	19.531	1.72	15	19.231	0.16	12	19.531	1.72	7
28.8	31.250	8.51	9	27.778	3.55	8	31.250	8.51	4
33.6	34.722	3.34	8	35.714	6.29	6	31.250	6.99	4
57.6	62.500	8.51	4	62.500	8.51	3	52.083	9.58	2
HIGH	1.221	-	255	0.977	-	255	0.610	-	255
LOW	312.500	-	0	250.000	-	0	156.250	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	0.300	0	207	0.3	0	191
1.2	1.202	0.17	51	1.2	0	47
2.4	2.404	0.17	25	2.4	0	23
9.6	8.929	6.99	6	9.6	0	5
19.2	20.833	8.51	2	19.2	0	2
28.8	31.250	8.51	1	28.8	0	1
33.6	-	-	-	-	-	-
57.6	62.500	8.51	0	57.6	0	0
HIGH	0.244	-	255	0.225	-	255
LOW	62.500	-	0	57.6	-	0

Tabla 26.3 Tasas de baudios (*baud rates*) para el modo asíncrono (BRGH = 0).

libros.ep-electropc.com

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

Tabla 26.4 Tasas de baudios para el modo asíncrono (BRGH = 1).

Tabla 26.5 Registros asociados con el generador de tasa de baudios (BRG).

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
18h	RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000x
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used by the BRG.

26.2.2 CIRCUITO DE MUESTREO O SAMPLING CIRCUIT

El dato sobre el pin RC7/RX/DT es muestreado tres veces por un circuito detector de mayoría, para determinar si es un nivel alto o bajo el que está presente en el pin RX (Figura 26.11).

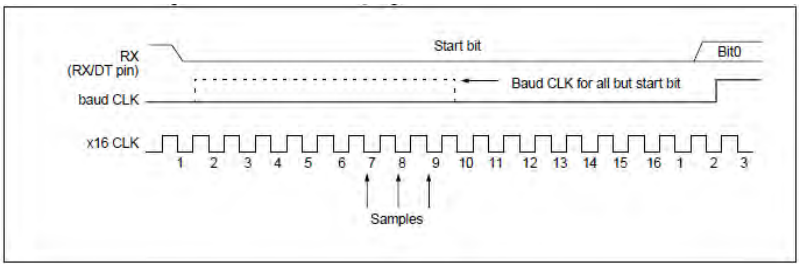


Figura 26.11 Diagrama de tiempos de muestreo.

El **circuito muestreador** o *sampling* detecta la caída del pin, y a partir de allí

libros.ep-electropc.com

comienza a contar 16 ciclos en una señal “0” o “1”; se hace el muestreo en la caída de los ciclos 7, 8 y 9 y se toma este valor.

26.3 DIAGRAMA CIRCUITAL

En las siguientes imágenes (Figuras 26.12 a 26.15) se aprecian los diagramas de bloque y el flujo de datos de los circuitos de transmisión y recepción.

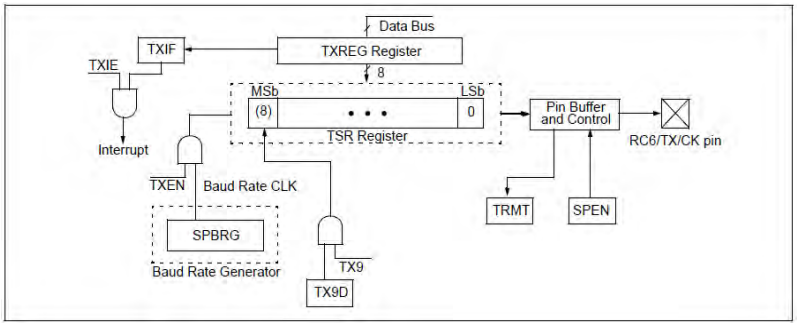


Figura 26.12 Flujo de datos de transmisión asincrónica.

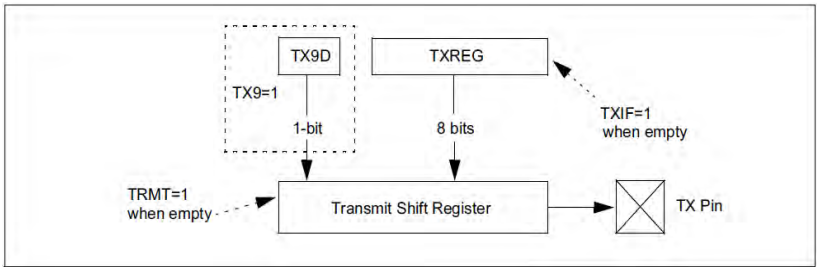


Figura 26.13 Diagrama de bloques del transmisor.

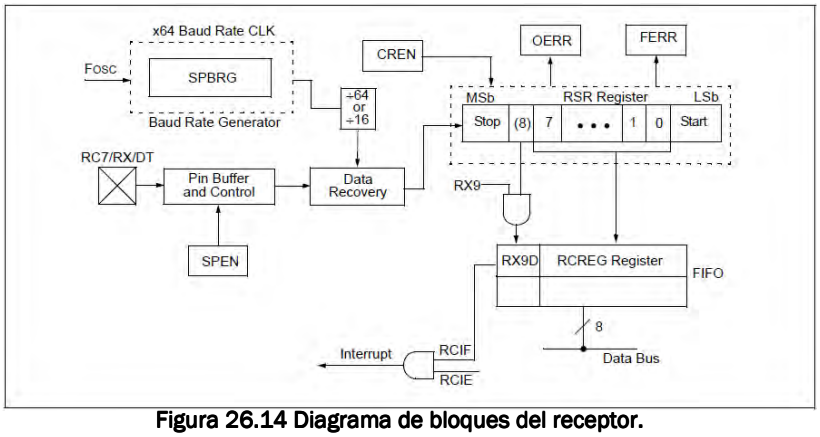


Figura 26.14 Diagrama de bloques del receptor.

libros.ep-electropc.com

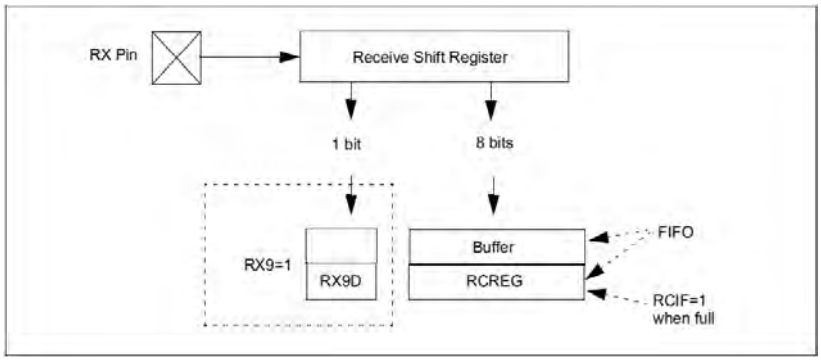


Figura 26.15 Flujo dedatos derecepciónasincrónica.

26.4 FUNCIONAMIENTO

Para entender el funcionamiento del dispositivo, primero entenderemos el protocolo de comunicación RS232. La comunicación RS232 se realiza serialmente, o sea que los datos van por un solo hilo (TX o RX, sin atender otras señales por el momento), y en una secuencia ordenada que indica previamente qué tipo de dato es; así, para nuestro caso, la señal se compone de 1 bit de START, 8 o 9 bits de datos y 1 bit de STOP (Figura 26.16).

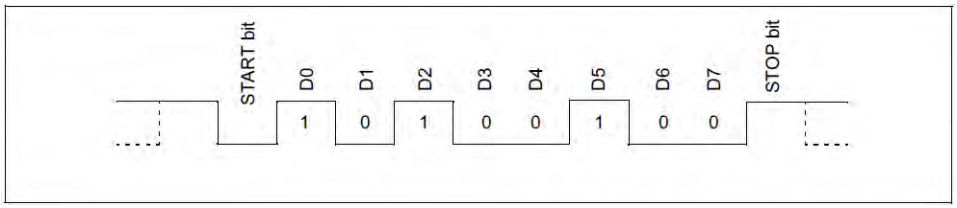


Figura 26.16 Trama serial.

Haciendo el análisis en el lado digital del dispositivo SCI (0 – VDD), la señal es "1" o nivel alto, cuando no hay transmisión (o recepción); cuando está en progreso y va a nivel bajo "0" comienza la transmisión; el receptor usa la transición hacia el nivel bajo para determinar la temporización de los bits siguientes. La señal está en bajo por la duración del bit de START, y es seguida por los bits de datos, primero el menos significativo, y puede recibir 8 o 9 bits de datos; el bit de STOP sigue al último bit de datos y es siempre un "1" o nivel alto. Por eso una transmisión termina cuando el pin queda en "1". Luego que el bit de STOP ha completado la trama, el bit de START de la transmisión siguiente puede ocurrir como se muestra en línea punteada (Figura 26.16). Acerca de la forma de onda de la Figura 26.16, que indica la señal de los pines TX o RX del micro, se debe notar que:

libros.ep-electropc.com

- El bit de START = 0 y el bit de STOP = 1, los bits de datos, son enviados y recibidos como el primer bit, el menos significativo (LSB); así que se tiene que mirar hacia atrás para formar el número binario.
- Los niveles de los datos no son invertidos hasta llegar al chip RS232 que usa los voltajes negativos para representar su lógica. Generalmente, si usamos un RS232, los niveles son invertidos y los niveles cambiados a través de los **intercambiadores** (*transceivers*) del chip para lograr viajar a distancias que no atenúen la señal en los cables.
- Algunas características que usa el USART afectan la señal. El modo de noveno bit se usa cuando hay paridad o si se necesita un bit extra de *stop*. También puede usarse para direccionamiento.

26.4.1 MODO ASÍNCRONO O ASINCRÓNICO

En este modo, la USART usa el formato estándar *Non-Return-to-Zero* (**NRZ**), 1 bit de ***start***, 8 o 9 bits de **datos**, 1 bit de ***stop***. El formato más común es de 8 bits. Un circuito dedicado al **generador de tasa de baudios** de 8 bits puede ser usado para derivar frecuencias estándar de tasa de baudios desde el oscilador. La USART transmite y recibe primero el LSB, el transmisor y el receptor son funcionalmente independientes, pero usan el mismo formato de tasa de baudios. El generador de tasa de baudios produce un reloj, ya sea de 16 o 64 del cambiador de tasa de bit, dependiendo del bit BRGH en el registro TXSTA<2>. La paridad no es soportada por el hardware, pero puede ser implementada en el software (y almacenada en el noveno bit). El modo Asíncrono es detenido durante *sleep*. El **modo asíncrono** es seleccionado por borrar el bit SYNC en el registro TXSTA<4>. El modo asincrónico consiste de los siguientes elementos importantes:

1. Generador de tasa de baudios o baud rate generator
2. Circuito de muestreo o sampling circuit
3. Transmisor asíncrono o asynchronous transmitter
4. Receptor asíncrono o asynchronous receiver

26.4.1.1 TRANSMISOR ASÍNCRONO

La parte principal del **transmisor** es el **TSR** (*Transmit Shift Register*) que obtiene sus datos del TXREG **buffer de transmisión de lectura/escritura** (*Read/Write Transmit Buffer*), y el registro TXREG es cargado con datos en el software. El registro TSR no es cargado hasta que el bit de *stop* ha sido transmitido desde la carga previa.

Tan pronto como el bit de *stop* es transmitido, el TSR es cargado con un nuevo dato desde el registro TXREG (si está disponible). Una vez que el registro TXREG

libros.ep-electropc.com

transfiere el dato al registro TSR (ocurre en un Tcy), el registro TXREG queda vacío y el bit TXIF del registro PIR<4> es puesto a uno. Esta interrupción puede ser habilitada o deshabilitada por poner a uno o por borrar el bit de habilitación TXIE del registro PIE1<4>.

El bit de la bandera TXIF puede ser puesto prescindiendo del estado de habilitación de TXIE, y no puede ser borrado por el software; solamente puede ser repuesto a uno cuando un dato nuevo es cargado en el registro TXREG. Mientras el bit de la bandera TXIF indica el estado del registro TXREG, otro bit TRMT del registro TXSTA<1> muestra el estado del registro TSR. El estado del bit TRMT es de sólo lectura que es puesto a uno cuando el registro TSR está vacío. No hay interrupción lógica para este bit; el usuario debe elegirlo y hacer *polling* (“pollearlo”) para determinar el estado vacío del registro TSR.

```
NOTA: El registro TSR no es mapeado en la memoria de datos, por eso
no está disponible para el usuario. El bit bandera TXIF es puesto a
uno cuando el bit TXEN es puesto. TXIF es borrado por cargar TXREG.
```

La transmisión (Figuras 26.17 y 26.18) es habilitada poniendo a uno el bit TXEN en el registro TXSTA<5>, y habilitar el pin de salida poniendo a uno el bit SPEN en el registro RCSTA<7>. La transmisión actual no puede ocurrir hasta que el registro TXREG sea cargado con un dato y el **BRG** (*Baud Rate Generator*: Generador de Tasa de Baudios) ha producido un desplazamiento del reloj.

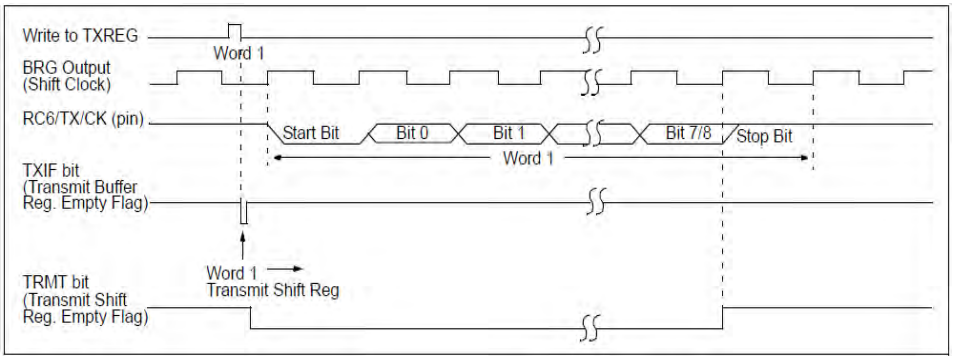


Figura 26.17 Diagrama de tiempos en modo asíncrono maestro de transmisión.

libros.ep-electropc.com

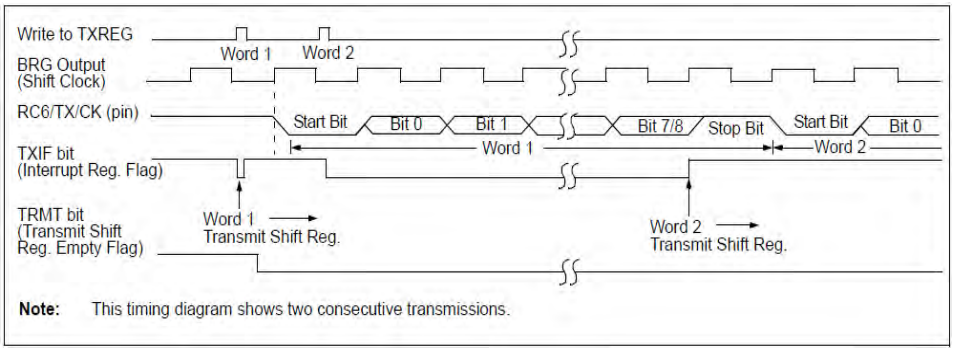


Figura 26.18 Diagrama de tiempos en modo asíncrono maestro de transmisión (*back-to-back*).

La transmisión también se puede iniciar por la primera carga del registro TXREG y, en consecuencia, poner a uno el bit de habilitación TXEN. Normalmente, cuando la transmisión se comienza primero, el registro TSR está vacío. A este punto, transferir al registro TXREG resultará en una transferencia inmediata al TSR, resultando en un vacío en TXREG. Así es posible una transferencia *back-to-back*. Aclarando el bit de habilitación TXEN durante una transmisión puede causar que la transmisión sea abortada, y podrá resetear el transmisor; como resultado el pin RC6/TX/CK puede quedar en impedancia alta.

Para seleccionar una transmisión de 9 bits, el bit TX9 del registro TXSTA<6> debe ser puesto a uno, y el noveno bit debe ser escrito en TX9D en el registro TXSTA<0>. El noveno bit debe ser escrito antes de escribir el dato de 8 bits al registro TXREG. Esto es lo indicado, porque una escritura al registro TXREG puede resultar en una transferencia inmediata del dato al registro TSR (si TSR está vacío). En ese caso, un dato incorrecto del noveno bit será cargado en el registro TSR. Para realizar una transmisión asincrónica, siga los pasos siguientes:

1. Inicialice el registro para SPBRG con la tasa de baudios apropiada. Si desea una velocidad alta, ponga el bit BRGH.
2. Habilite el puerto serial, aclarando el bit SYNC y ponga a uno el bit SPEN.
3. Si desea una interrupción, habilite el bit TXIE.
4. Si desea una transmisión de 9 bits, ponga a uno el bit TX9 de transmisión.
5. Habilite la transmisión poniendo a uno el bit TXEN que también pondrá a uno el bit TXIF.
6. Si selecciona una transmisión de 9 bits, debe cargar el noveno bit en el bit TX9D.
7. Cargue el dato en el registro TXREG (comienza la transmisión).

libros.ep-electropc.com

8. Si usa una interrupción, asegúrese de que los bits GIE y PEIE en el registro INTCON<7:6> sean puestos a uno.

La siguiente es una subrutina para **transmitir datos en modo asíncrono** a 9 600 bps, con su respectiva inicialización:

```
;-----------------------------------------------------------------
; Transmite dato en W, cuando TXIF = 1, o TXREG estén vacíos

Tx_SCI:
        btfss   PIR1,TXIF ; evalúa si TXREG vacío, sí, salta
        goto    $-1             ; no, espera hasta que TXREG esté desocupado
        movwf   TXREG           ; dato -> TXREG

        return
;-----------------------------------------------------------------
; inicialice SCI, como transmisor a 9 600

Ini_SCI:
        bsf     STATUS,RP0      ; Banco 1
        movlw   _9600_SPBRG     ; 9 600 tasa de baudios
        movwf   SPBRG
        movlw   0x24            ; TXEN = 1 habilite transmisión BRGH = 1 high baud rate
        movwf   TXSTA

        bcf     STATUS,RP0      ; Banco 0
        movlw   0x80            ; SPEN = 1, habilite serial
        movwf   RCSTA

        return
;-----------------------------------------------------------------
```

La forma de usarlo es:

```
        call    Ini_SCI         ; inicialice SCI

MainLoop:
        movf    contador,W      ; carga contador -> W
        call    Tabla_msg       ; llama a tabla mensaje
                                ; en W carácter a TX
        call    Tx_SCI          ; Tx dato serial
```

26.4.1.2 RECEPTOR ASÍNCRONO

El dato es recibido en el pin RC7/RX/DT y maneja la recuperación del dato. Una vez que el modo **asincrónico** es seleccionado, la recepción es habilitada poniendo a uno el bit CREN en el registro RCSTA<4>. La parte principal del receptor es el registro **RSR** (*Receive Shift Register*). Después de muestrear el bit de *stop*, el receptor de datos en el RSR es transferido al registro RCREG (si está vacío). Si la transferencia es completa, el bit de bandera RCIF del registro PIR1<5> es puesto a uno. La interrupción actual puede ser habilitada o deshabilitada por poner a uno o por

libros.ep-electropc.com

borrar el bit de habilitación RCIE en el registro PIE1<5>. El bit de bandera RCIF es de sólo lectura y es borrado por el hardware. Es borrado cuando el RCREG ha sido leído y está vacío.

El RCREG es un registro de doble buffer (como un FIFO de 2 niveles). Es posible que 2 bytes de datos sean recibidos y transferidos al RCREG FIFO, y que un tercer byte comience a moverse en el registro RSR. Sobre la detección del bit de *stop* del tercer byte, si el RCREG está aún lleno, el bit de Error de Overrun OERR en el registro RCSTA<1> será puesto a uno. La palabra en el RSR puede ser perdida. El registro RCREG puede ser leído dos veces para recuperar los dos bytes en el FIFO. El bit de Overrun OERR tiene que ser borrado por el software. Esto se hace para inicializar la lógica de recepción (Figura 26.19) (CREN es borrado y luego poniéndolo a uno). Si el bit OERR es puesto a uno, las transferencias desde el registro RSR al registro RCREG son inhibidas, y los datos adicionales no pueden ser recibidos. Por esto es esencial borrar el bit OERR si está puesto a uno.

El bit FERR de **error de trama** (*framming*), en el registro RCSTA<2>, es puesto a uno si el bit de *stop* es detectado como cero. El bit FERR y el noveno bit recibido son almacenados en la misma forma como se recibe el dato. Leyendo el registro RCREG pueden ser cargados los bits RX9D y FERR con valores nuevos; por esto, para el usuario, es esencial leer el registro RCSTA antes de leer el registro RCREG, para no perder la información anterior de los bits FERR y RX9D.

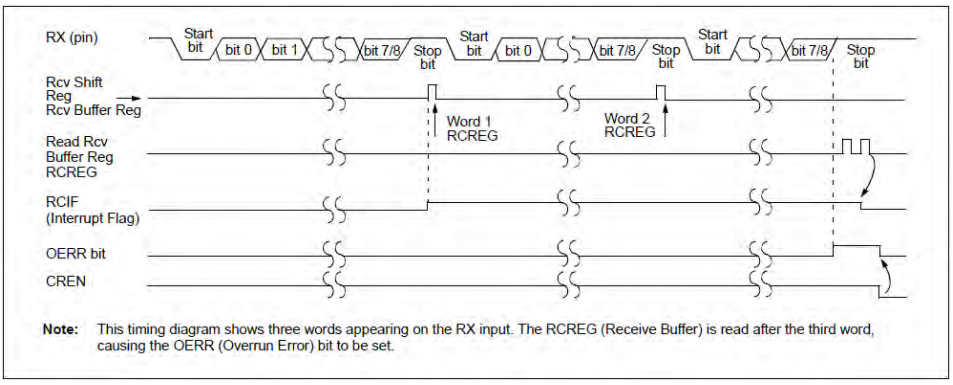


Figura 26.19 Diagrama de tiempos modo asíncrono de recepción.

Para realizar una **recepción asincrónica**, siga los pasos siguientes:

- Inicialice el registro SPBRG para la tasa de baudios apropiado. Si desea una alta velocidad de tasa de baudios, setee el bit BRGH.
- Habilite el **puerto serial asincrónico** por borrar el bit SYNC y ponga a uno el bit SPEN.

libros.ep-electropc.com

- Si desea las interrupciones, habilite el bit RCIE en el registro PIE1<5>.
- Si desea el noveno bit de recepción, setee el bit RX9D en el registro RCSTA<0>.
- Habilite la recepción, seteando el bit CREN en el registro RCSTA<4>.
- El bit de bandera RCIF en el registro PIR1<5> será puesto cuando la recepción es completa, y una interrupción se generará si el bit RCIE está puesto a uno en el registro PIE1<5>.
- Lea el registro RCSTA para dar el noveno bit (si está habilitado), y determine si algún error ha ocurrido en otra recepción.
- Lea el octavo bit de dato recibido, leyendo el registro RCREG.
- Si cualquier error ha ocurrido, aclare el error por borrar el bit de habilitación CREN en el registro RCSTA<4>.
- Si está usando interrupciones, asegúrese de que el bit GIE y PEIE del registro INTCON<7:6> sean puestos a uno.

La siguiente es una rutina para **recibir** datos en modo **asíncrono** a 9 600 baudios, con su inicialización respectiva:

```
;-----------------------------------------------------------------
; inicialice SCI, como receptor asíncrono a 9 600

Ini_SCI:
        bsf     STATUS,RP0          ; Banco 1
        movlw   _9600_SPBRG         ; 9600 tasa de bps
        movwf   SPBRG
        movlw   0x04                ; BRGH = 1 high baud rate
        movwf   TXSTA

        bcf     STATUS,RP0          ; Banco 0
        movlw   0x90                ; SPEN = 1 habilite serial, CREN = 1 recepción continúa
        movwf   RCSTA

        return
;-----------------------------------------------------------------
;Check si hay dato recibido, no = retorna, si dato->W y flags,_nuevo_dato = 1
Rx_SCI
        btfss   PIR1,RCIF ; check si RCREG lleno, sí, salta
        return                      ; no, retorna no hay dato

        btfsc   RCSTA,OERR          ; si ocurre un error de overrun
        goto    Er_SCI_Overr        ; maneje el error
        btfsc   RCSTA,FERR          ; si ocurre un error de trama
        goto    Er_SCI_Frame        ; maneje el error

        movf    RCREG,W             ; dato recibido -> W
        bsf     flags,_nuevo_dato   ;indica nuevo dato recibido
        return
```


libros.ep-electropc.com

```
; Error de Overrun, OERR = 1, se aclara error y continúa
Er_SCI_Overr:
        bcf     RCSTA,CREN      ; reset la lógica del receptor
        bsf     RCSTA,CREN      ; habilite la lógica del receptor
        return

; Error de Frame, FERR = 1, se aclara error y continúa
Er_SCI_Frame:
        movf    RCREG,W         ; descarga dato recibido con error -> W
        return
;-----------------------------------------------------------------
```

La forma de usarlo es:

```
    Main:

        call    Ini_SCI         ; inicialice SCI


    MainLoop:

        call    Rx_SCI          ; check nuevo dato recibido y lo carga en W
```

26.5 REGISTROS ASOCIADOS

Estos registros se muestran en las tablas 26.5, 26.6, 26.7 y 26.8.

libros.ep-electropc.com

Tabla 26.6 Registro TXSTA.

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7							bit 0

bit 7 **CSRC:** Clock Source Select bit
Asynchronous mode:
Don't care.
Synchronous mode:
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9**: 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN**: Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled
Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC**: USART Mode Select bit
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH**: High Baud Rate Select bit
Asynchronous mode:
1 = High speed
0 = Low speed
Synchronous mode:
Unused in this mode.

bit 1 **TRMT**: Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 26.7 Registro RCSTA.

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R-0	R-x
SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (configures RC7/RX/DT and RC6/TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9**: 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN**: Single Receive Enable bit
Asynchronous mode:
Don't care.
Synchronous mode – Master:
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.
Synchronous mode – Slave:
Don't care.

bit 4 **CREN**: Continuous Receive Enable bit
Asynchronous mode:
1 = Enables continuous receive
0 = Disables continuous receive
Synchronous mode:
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **ADDEN:** Address Detect Enable bit
Asynchronous mode 9-bit (RX9 = 1):
1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set
0 = Disables address detection, all bytes are received and ninth bit can be used as parity bit

bit 2 **FERR**: Framing Error bit
1 = Framing error (can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR**: Overrun Error bit
1 = Overrun error (can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of Received Data (can be parity bit but must be calculated by user firmware)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

libros.ep-electropc.com

Tabla 26.8 Registros asociados con la transmisión asíncrona.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous transmission.
Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

Tabla 26.9 Registros asociados con la recepción asíncrona.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented locations read as '0'. Shaded cells are not used for asynchronous reception.
Note 1: Bits PSPIE and PSPIF are reserved on 28-pin devices; always maintain these bits clear.

26.6 PRÁCTICA

Veremos tres ejemplos de esta manera:

1 Modo de Transmisión y Recepción
C:\Ejemplos_ASM\USART_1\usart_1.asm

2 Modo de Transmisión
C:\Ejemplos_ASM\USART_1\usart_1_tx.asm

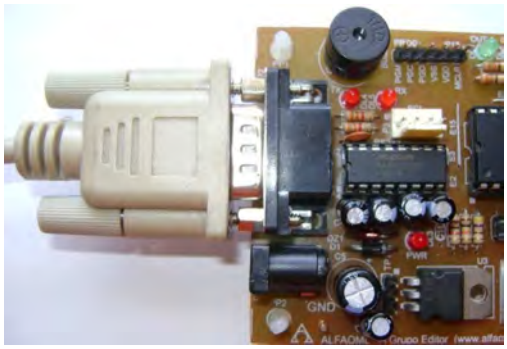
3 Modo de Recepción
C:\Ejemplos_ASM\USART_1\usart_1_rx.asm

libros.ep-electropc.com

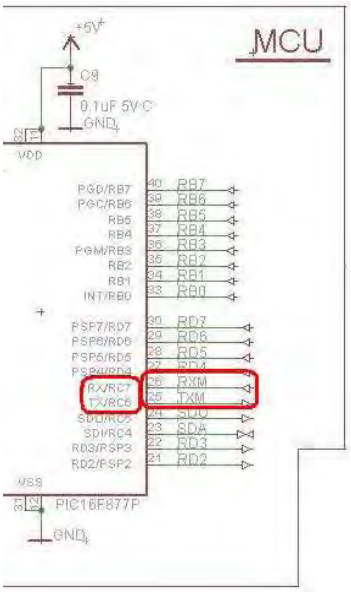
26.6.1 MONTAJE

Para estos ejemplos, el montaje requiere el sistema de evaluación **PIC16F8XX** y **el PICkit 2**; en el *socket* U2 se monta el integrado **MAX232**, y se necesita un cable externo **RS232** macho-hembra cruzado, conectado a la PC, tal como se observa en la Figura 26.20 a, b, c.

a)



b)



c)

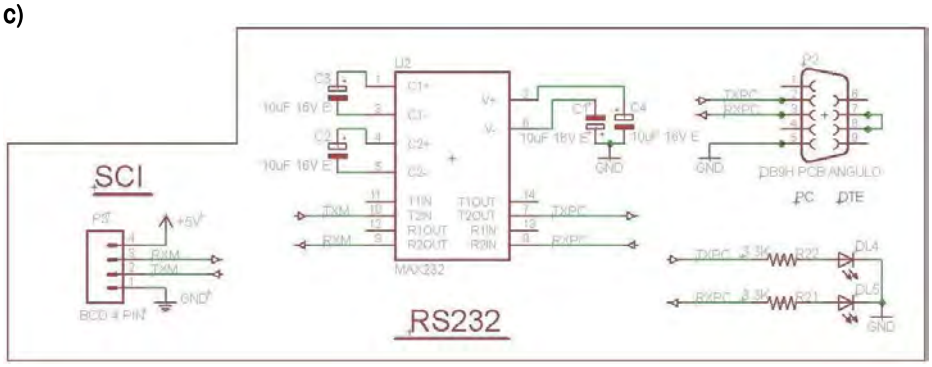


Figura 26.20 PIC16F8XX circuito RS232 board y esquemático

libros.ep-electropc.com

26.6.1.1 MODO DE TRANSMISIÓN Y RECEPCIÓN

Este ejemplo está hecho en **modo asincrónico** a 9600 bps 8N1, para probar el modo de **recepción y transmisión**. Este programa **recibe y transmite** lo digitado en el teclado de la PC, cuando estamos conectados vía RS232 por el puerto serial (COM1) con el programa HyperTerminal, configurado como 9600 8N1, de tal forma que lo observaremos como se ilustra en la Figura 26.21.

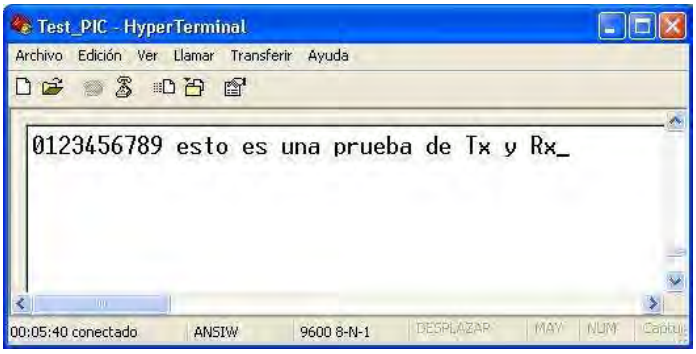


Figura 26.21 Hyperterminal.

El código siguiente es el del ejemplo que está en el CD info en la Web en la ruta X:\ejemplos_ASM que, al copiarlo al C:\, se encuentra como: C:\Ejemplos_ASM\USART_1\usart_1.asm.

```
; * * * * * * * * * * * * * * * * * C:\Ejemplos_ASM\USART_1\usart_1.asm * * * * * * * * * * * * * * *

            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>           ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _9600_SPBRG      .129              ; pone 9600 bps
#define    _nuevo_dato      0                 ; Bit_0 flag, Nuevo dato recibido

            cblock 0x20             ; define variables desde posición 20 en RAM
flags                               ; banderas
            endc
;-----------------------------------------------------------------
            org 0x0000                      ; inicia programa
Main:
            call        Ini_SCI             ; inicialice SCI
MainLoop:
            call        Rx_SCI              ; verifica nuevo dato recibido y lo carga en W
            btfsc       flags,_nuevo_dato   ; check si nuevo dato recibido
            call        Tx_SCI              ; Tx el nuevo dato recibido por serial
            bcf         flags,_nuevo_dato   ; apague hay nuevo dato recibido
```


libros.ep-electropc.com

```
            goto    MainLoop            ; haga ciclo
;-------------------------------------------------------------------
;Check si hay dato recibido, no = retorna, si dato->W y flags,_nuevo_dato = 1
Rx_SCI
            btfss   PIR1,RCIF           ; check si hay dato
            return                      ;retorna si no hay dato

            btfsc   RCSTA,OERR          ; si ocurre un error de overrun
            goto    Er_SCI_Overr        ; maneje el error
            btfsc   RCSTA,FERR          ; si ocurre un error de trama
            goto    Er_SCI_Frame        ; maneje el error

            movf    RCREG,W             ; dato recibido -> W
            bsf     flags,_nuevo_dato   ;indica nuevo dato recibido
            return

; Error de Overrun, OERR = 1, se aclara error y continúa
Er_SCI_Overr:
            bcf     RCSTA,CREN          ; reset la lógica del receptor
            bsf     RCSTA,CREN          ; habilite la lógica del receptor
            return

; Error de Frame, FERR = 1, se aclara error y continúa
Er_SCI_Frame:
            movf    RCREG,W             ; descarga dato recibido con error -> W
            return
;-------------------------------------------------------------------
; inicialice SCI, como transmisor asíncrono a 9600

Ini_SCI:
            bsf     STATUS,RP0          ; Banco 1
            movlw   _9600_SPBRG         ; 9600 tasa de baudios
            movwf   SPBRG
            movlw   0x24                ; TXEN = 1 habilite transmisión BRGH = 1 high baud rate
            movwf   TXSTA

            bcf     STATUS,RP0          ; Banco 0
            movlw   0x90                ; SPEN = 1 habilite serial, CREN = 1 recepción continúa
            movwf   RCSTA

            return
;-------------------------------------------------------------------
; Transmite dato en W, cuando TXIF = 0, o TXREG vacío

Tx_SCI:
            btfss   PIR1,TXIF           ; check si transmisor ocupado
            goto    $-1                 ; espera hasta que transmisor esté desocupado
            movwf   TXREG               ; transmite el dato

            return
;-------------------------------------------------------------------
            end
```


libros.ep-electropc.com

26.6.1.2 MODO DE TRANSMISIÓN

Este ejemplo está hecho en **modo asincrónico** a 9 600 bps 8N1, para probar el **modo de transmisión**. Este programa transmite el mensaje "Hola Mundo" y al final del mensaje se hacen dos comandos *carriage return* y *line feed*. Cada 0.5 segundos una letra es transmitida vía RS232 y se puede recibir por un puerto serial de la PC (COM1) con el programa HyperTerminal, configurado como 9600 8N1, de tal forma que lo observaremos como se ilustra en la Figura 26.22.

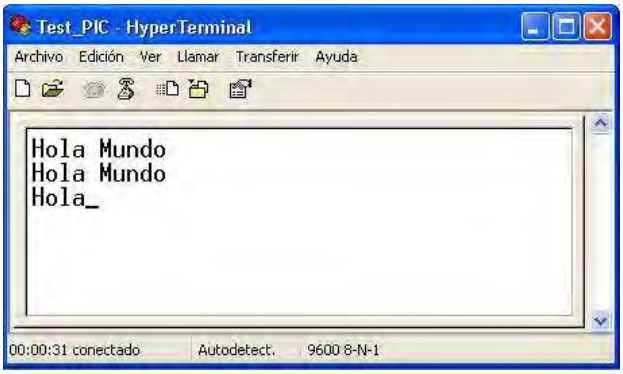


Figura 26.22 HyperTerminal.

El código siguiente es el del ejemplo que está en nuestra página Web en la ruta X:\Ejemplos_ASM, que al copiarlo al C:\, se encuentra como: C:\Ejemplos_ASM\USART_1\usart_1_tx.asm:

```
; * * * * * * * * * * * * * * * * * * * * * * C:\Ejemplos_ASM\USART_1\usart_1_tx.asm * * * * * * * * * * * * * * * * *

            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF & _CPD_OFF

#define    _9600_SPBRG        .129        ; pone 9600 bps
#define    _kte_delay3        h'0F'       ; valor para retardo 0.5 s

            cblock 0x20             ; define variables desde posición 20 en RAM
contador                ; contador de dígito
delay1                  ; retardo 1 ; define 3 registros para retardo
delay2                  ; retardo 2
delay3                  ; retardo 3
            endc
;-----------------------------------------------------------------
```


libros.ep-electropc.com

```
        org 0x0000              ; inicia programa
Main:
        call    Ini_SCI         ; inicialice SCI
        clrf    contador        ; borre contador
        clrf    delay1          ; borre retardo 1
        clrf    delay2          ; borre retardo 2

MainLoop:
        movf    contador,W      ; carga contador -> W
        call    Tabla_msg       ; llama a tabla mensaje

        call    Tx_SCI          ; Tx dato serial

        call    Chequea_12      ; contador = 12?
        call    Delay           ; haga retardo de 0.5 s

        goto    MainLoop        ; haga ciclo

;-----------------------------------------------------------------
; inicialice SCI, como transmisor asíncrono a 9600

Ini_SCI:
        bsf     STATUS,RP0      ; Banco 1
        movlw   _9600_SPBRG     ; 9600 tasa de bps
        movwf   SPBRG
        movlw   0x24            ; TXEN = 1 habilite transmisión BRGH = 1 high baud rate
        movwf   TXSTA

        bcf     STATUS,RP0      ; Banco 0
        movlw   0x80            ; SPEN = 1, habilite serial
        movwf   RCSTA

        return
;-----------------------------------------------------------------
; Transmite dato en W, cuando TXIF = 0, o TXREG vacío

Tx_SCI:
        btfss   PIR1,TxIF       ; check si transmisor ocupado
        goto    $-1             ; espera hasta que transmisor esté desocupado
        movwf   TXREG           ; transmite el dato

        return
;-----------------------------------------------------------------
; Verifica si contador = 12?/caso SI -> contador = 0/caso NO contador = +1/
Check_12
        incf    contador,F      ; siguiente número
        movlw   .12             ; carga 12
        xorwf   contador,W      ; compara contador con 12
        btfsc   STATUS,Z        ; comparación NO es igual a 12
        clrf    contador        ; contador a 0

        return
;-----------------------------------------------------------------
; Hace retardo de 0.5 s
Delay
        movlw   _kte_delay3     ; carga valor a
```


libros.ep-electropc.com

```
            movwf   delay3              ; Retardo 3 de ALTO nivel

Loop:
            decfsz  delay1,f            ; disminuye Retardo de BAJO nivel
            goto    Loop                ; hay 3 ciclos por loop * 256 loopss = 768 instrucciones
            decfsz  delay2,f            ; disminuye Retardo de MEDIO nivel
            goto    Loop                ; hay por loop (768 + 3) * 256 ciclos = 197.376 ciclos
            decfsz  delay3,f            ; disminuye Retardo de ALTO nivel
            goto    Loop                ; hay por loop (197.376 + 3) * 31 ciclos = 6.118.749 ciclos

            return
;-----------------------------------------------------------------
Tabla_msg               ; Tabla mensaje.
            addwf   PCL,F

            dt      "Hola Mundo"        ; mensaje
            dt      0x0D                ;0x0D     carriage return
            dt      0x0A                ;0x0A     line feed
            end
```

26.6.1.3 MODO DE RECEPCIÓN

Este ejemplo está hecho en **modo asincrónico** a 9600 bps 8N1, para probar el **modo de recepción**. Este programa recibe 16 teclas escritas en el teclado de la PC cuando estamos conectados vía RS232 por el puerto serial (COM1) con el programa **HyperTerminal**, configurado como **9600 8N1**, de tal forma que lo observaremos como se ilustra en la Figura 26.23; las teclas oprimidas en la PC se reciben y ponen en la memoria RAM en las posiciones (0x20 a 0x31); en la primera parte ilustra las teclas de 0-9 y a-f; en la segunda, las teclas de a-p.

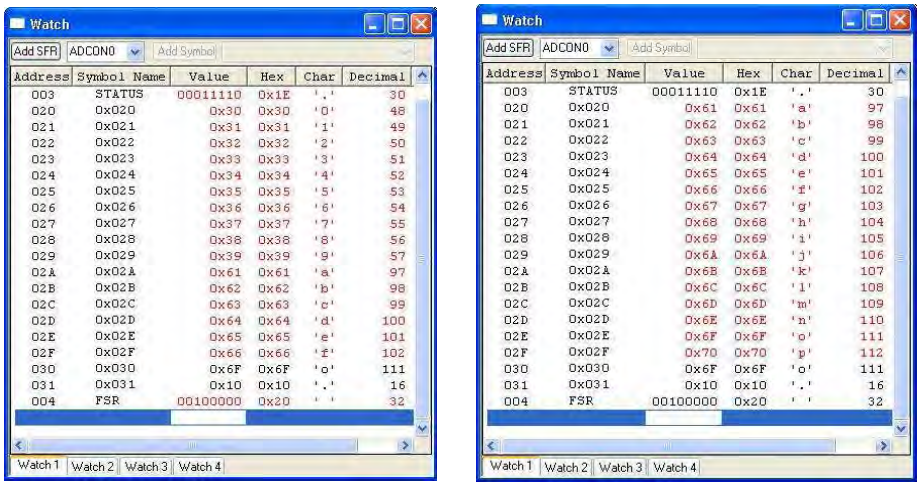


Figura 26.23 Ventana *match*

libros.ep-electropc.com

El código siguiente es el del ejemplo que está en nuestra página Web en la ruta X:\Ejemplos_ASM, que, al copiarlo al C:\, se encuentra como: C:\Ejemplos_ASM\USART_1\usart_1_ rx.asm:

```
; * * * * * * * * * * * * * * * * C:\Ejemplos_ASM\USART_1\usart_1_ rx.asm * * * * * * * * * * * * *

            list p = 16f877a        ; directiva para definir procesador
 #include <p16f877a.inc>        ; definiciones de variables específicas del procesador

 __CONFIG _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_ON & _HS_OSC & _WRT_OFF & _LVP_OFF
& _CPD_OFF

#define    _9600_SPBRG     .129            ; pone 9600 bps
#define    _nuevo_dato     0               ; Bit_0 flag, nuevo dato recibido

            cblock 0x20     ; define variables desde posición 20 en RAM
buffer      :10             ; reserva 16 posiciones de memoria para datos
flags                       ; banderas
contador                    ; contador
            endc

;-----------------------------------------------------------------
            org 0x0000                  ; inicia programa
Main:
            call    Ini_SCI             ; inicialice SCI

Main_Ini_var
            movlw   buffer              ; carga dirección
            movwf   FSR                 ; para dir/to. indirecto
            clrf    contador            ; contador = 0

Main_no_dato
            bcf     flags,_nuevo_dato   ; apague hay nuevo dato recibido

MainLoop:
            call    Rx_SCI              ; check nuevo dato recibido y lo carga en W
            btfss   flags,_nuevo_dato   ; check si nuevo dato recibido
            goto    Main_no_dato

            movwf   INDF                ; guárdelo en INDF
            incf    FSR                 ; incremente apuntador
            incf    contador,F          ; siguiente número
            movlw   .16                 ; carga 16
            xorwf   contador,W          ; compara contador con 16
            btfss   STATUS,Z            ; comparación, si es igual a 16 salte
            goto    Main_no_dato        ; no, continúa
            goto    Main_Ini_var        ; sí, inicialice variables

;-----------------------------------------------------------------
;Check si hay dato recibido, no = retorna, si dato->W y flags,_nuevo_dato = 1
Rx_SCI
            btfss   PIR1,RCIF           ; check si RCREG lleno, sí, salta
            return                      ; no, retorna no hay dato
```


libros.ep-electropc.com

```
            btfsc   RCSTA,OERR         ; si ocurre un error de overrun
            goto    Er_SCI_Overr       ; maneje el error
            btfsc   RCSTA,FERR         ; si ocurre un error de trama
            goto    Er_SCI_Frame       ; maneje el error

            movf    RCREG,W            ; dato recibido -> W
            bsf     flags,_nuevo_dato  ;indica nuevo dato recibido
            return

; Error de Overrun, OERR = 1, se aclara error y continúa
Er_SCI_Overr:
            bcf     RCSTA,CREN         ; reset la lógica del receptor
            bsf     RCSTA,CREN         ; habilite la lógica del receptor
            return

; Error de Frame, FERR = 1, se aclara error y continúa
Er_SCI_Frame:
            movf    RCREG,W            ; descarga dato recibido con error -> W
            return
;-------------------------------------------------------------------
; inicialice SCI, como receptor asíncrono a 9600

Ini_SCI:
            bsf     STATUS,RP0         ; Banco 1
            movlw   _9600_SPBRG        ; 9600 tasa de bps
            movwf   SPBRG
            movlw   0x04               ; BRGH = 1 high baud rate
            movwf   TXSTA

            bcf     STATUS,RP0         ; Banco 0
            movlw   0x90               ; SPEN = 1 habilite serial, CREN = 1 recepción continúa
            movwf   RCSTA

            return
;-------------------------------------------------------------------
; Transmite dato en W, cuando TXIF = 1, o TXREG vacío

Tx_SCI:
            btfss   PIR1,TXIF          ; check si TXREG vacío, sí, salta
            goto    $-1                ; no, espera hasta que TXREG esté desocupado
            movwf   TXREG              ; dato -> TXREG

            return
;-------------------------------------------------------------------
            end
```


libros.ep-electropc.com

EVALUACIÓN

P.: 1. ¿Cuáles son los niveles de voltaje del protocolo RS232?
R.:
1. Los datos son en forma de lógica negativa, o sea, un voltaje entre +3V y +15V es un dato lógico "0", y un voltaje entre -3V y -15V es un dato lógico "1".
2. Un dato no válido es una señal entre +3V y -3V.
3. Los niveles comúnmente usados son entre 9V y 12V.

P.: 2. ¿Qué partes integra una trama de datos en el protocolo RS232?
R.:
- *Start* (inicio o bit de arranque)
- *Data* (datos 8 bits o bytes)
- *Parity* (paridad)
- *Stop* (parada)

P.: 3. ¿Cuáles son los modos de trabajo del módulo SCI?
R.:
- Asíncrono (*full-duplex*)
- Síncrono – maestro (*half duplex*)
- Síncrono – esclavo (*half duplex*)

libros.ep-electropc.com

APÉNDICE A

CARACTERÍSTICAS TÉCNICAS

libros.ep-electropc.com

CARACTERÍSTICAS TÉCNICAS

MICROCHIP

PIC16F87XA

28/40/44-Pin Enhanced Flash Microcontrollers

Devices Included in this Data Sheet:

- PIC16F873A
- PIC16F874A
- PIC16F876A
- PIC16F877A

High-Performance RISC CPU:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: DC – 20 MHz clock input
  DC – 200 ns instruction cycle
- Up to 8K x 14 words of Flash Program Memory, Up to 368 x 8 bytes of Data Memory (RAM), Up to 256 x 8 bytes of EEPROM Data Memory
- Pinout compatible to other 28-pin or 40/44-pin PIC16CXXX and PIC16FXXX microcontrollers

Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- Synchronous Serial Port (SSP) with SPI™ (Master mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) – 8 bits wide with external $\overline{RD}$, $\overline{WR}$ and $\overline{CS}$ controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
  - Two analog comparators
  - Programmable on-chip voltage reference (V_{REF}) module
  - Programmable input multiplexing from device inputs and internal voltage reference
  - Comparator outputs are externally accessible

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

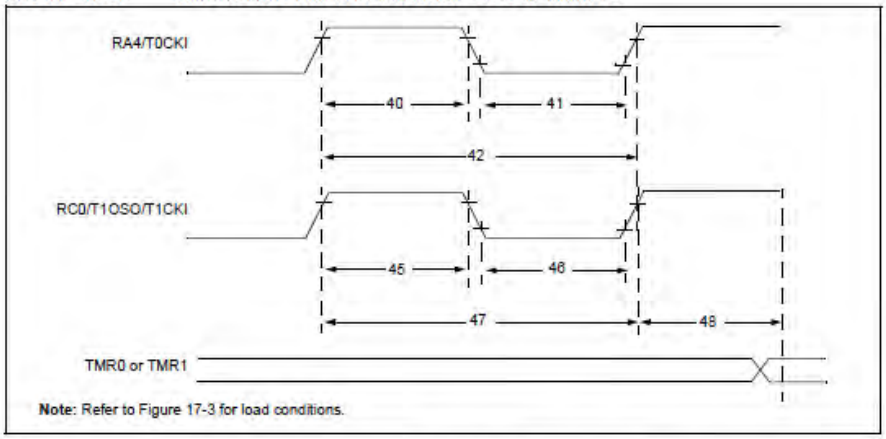
Device	Program Memory		Data SRAM (Bytes)	EEPROM (Bytes)	I/O	10-bit A/D (ch)	CCP (PWM)	MSSP		USART	Timers 8/16-bit	Comparators
	Bytes	# Single Word Instructions						SPI	Master I^2C			
PIC16F873A	7.2K	4096	192	128	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F874A	7.2K	4096	192	128	33	8	2	Yes	Yes	Yes	2/1	2
PIC16F876A	14.3K	8192	368	256	22	5	2	Yes	Yes	Yes	2/1	2
PIC16F877A	14.3K	8192	368	256	33	8	2	Yes	Yes	Yes	2/1	2

© 2003 Microchip Technology Inc.

libros.ep-electropc.com

TIMER0 Y TIMER1 REQUERIMIENTOS CLOCK EXTERNO

FIGURE 17-8: TIMER0 AND TIMER1 EXTERNAL CLOCK TIMINGS



Note: Refer to Figure 17-3 for load conditions.

TABLE 17-6: TIMER0 AND TIMER1 EXTERNAL CLOCK REQUIREMENTS

Param No.	Symbol	Characteristic			Min	Typ†	Max	Units	Conditions
40*	TT0H	T0CKI High Pulse Width		No Prescaler	0.5 TCY + 20	—	—	ns	Must also meet parameter 42
				With Prescaler	10	—	—	ns	
41*	TT0L	T0CKI Low Pulse Width		No Prescaler	0.5 TCY + 20	—	—	ns	Must also meet parameter 42
				With Prescaler	10	—	—	ns	
42*	TT0P	T0CKI Period		No Prescaler	TCY + 40	—	—	ns	
				With Prescaler	Greater of: 20 or TCY + 40 / N	—	—	ns	N = prescale value (2, 4,..., 256)
45*	TT1H	T1CKI High Time	Synchronous, Prescaler = 1		0.5 TCY + 20	—	—	ns	Must also meet parameter 47
			Synchronous, Prescaler = 2, 4, 8	Standard(F)	15	—	—	ns	
				Extended(LF)	25	—	—	ns	
			Asynchronous	Standard(F)	30	—	—	ns	
				Extended(LF)	50	—	—	ns	
46*	TT1L	T1CKI Low Time	Synchronous, Prescaler = 1		0.5 TCY + 20	—	—	ns	Must also meet parameter 47
			Synchronous, Prescaler = 2, 4, 8	Standard(F)	15	—	—	ns	
				Extended(LF)	25	—	—	ns	
			Asynchronous	Standard(F)	30	—	—	ns	
				Extended(LF)	50	—	—	ns	
47*	TT1P	T1CKI Input Period	Synchronous	Standard(F)	Greater of: 30 or TCY + 40 / N	—	—	ns	N = prescale value (1, 2, 4, 8)
				Extended(LF)	Greater of: 50 or TCY + 40 / N				N = prescale value (1, 2, 4, 8)
			Asynchronous	Standard(F)	60	—	—	ns	
				Extended(LF)	100	—	—	ns	
	FT1	Timer1 Oscillator Input Frequency Range (oscillator enabled by setting bit T1OSCEN)			DC	—	200	kHz	
48	TCKEZTMR1	Delay from External Clock Edge to Timer Increment			2 TOSC	—	7 TOSC	—	

* These parameters are characterized but not tested.

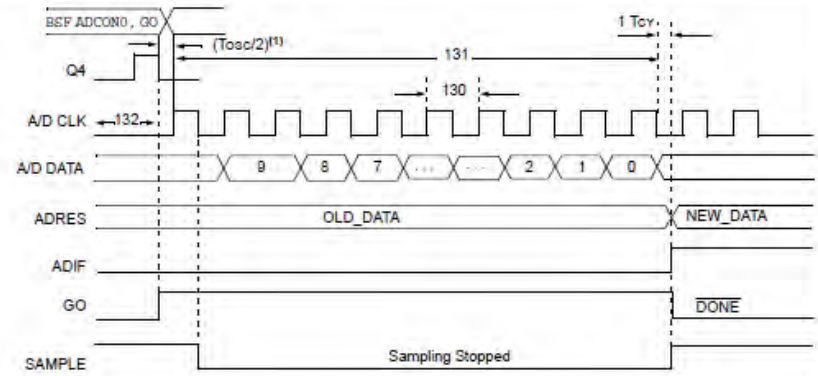
† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

© 2003 Microchip Technology Inc.

libros.ep-electropc.com

DIAGRAMA DE TIEMPOS A/D

FIGURE 17-19: A/D CONVERSION TIMING



Note: If the A/D clock source is selected as RC, a time of TCY is added before the A/D clock starts. This allows the SLEEP instruction to be executed.

TABLE 17-15: A/D CONVERSION REQUIREMENTS

Param No.	Symbol	Characteristic		Min	Typ†	Max	Units	Conditions
130	TAD	A/D Clock Period	PIC16F87XA	1.6	—	—	µs	TOSC based, VREF ≥ 3.0V
			PIC16LF87XA	3.0	—	—	µs	TOSC based, VREF ≥ 2.0V
			PIC16F87XA	2.0	4.0	6.0	µs	A/D RC mode
			PIC16LF87XA	3.0	6.0	9.0	µs	A/D RC mode
131	TCNV	Conversion Time (not including S/H time) (Note 1)			—	12	TAD	
132	TACQ	Acquisition Time		(Note 2)	40	—	µs	
				10*	—	—	µs	The minimum time is the amplifier settling time. This may be used if the "new" input voltage has not changed by more than 1 LSb (i.e., 20.0 mV @ 5.12V) from the last sampled voltage (as stated on CHOLD).
134	TGO	Q4 to A/D Clock Start		—	TOSC/2 §	—	—	If the A/D clock source is selected as RC, a time of TCY is added before the A/D clock starts. This allows the SLEEP instruction to be executed.

* These parameters are characterized but not tested.

† Data in "Typ" column is at 5V, 25°C unless otherwise stated. These parameters are for design guidance only and are not tested.

§ This specification ensured by design.

Note 1: ADRES register may be read on the following TCY cycle.

2: See **Section 11.1 "A/D Acquisition Requirements"** for minimum conditions.

© 2003 Microchip Technology Inc.

libros.ep-electropc.com

COMPARADORES CARACTERÍSTICAS ELÉCTRICAS RETARDO DE CAMBIO DE MODO

PIC16F87XA

TABLE 17-1: COMPARATOR SPECIFICATIONS

Operating Conditions:	3.0V < VDD < 5.5V, -40°C < TA < +85°C (unless otherwise stated) 4.0V < VDD < 5.5V, -40°C < TA < +125°C (unless otherwise stated)						
Param No.	**Sym**	**Characteristics**	**Min**	**Typ**	**Max**	**Units**	**Comments**
D300	VIOFF	Input Offset Voltage	—	± 5.0	± 10	mV	
D301	VICM	Input Common Mode Voltage*	0	-	VDD – 1.5	V	
D302	CMRR	Common Mode Rejection Ratio*	55	-	—	dB	
300 300A	TRESP	Response Time*(1)	—	150	400 600	ns ns	PIC16F87XA PIC16LF87XA
301	TMC2OV	Comparator Mode Change to Output Valid*	—	—	10	µs	

* These parameters are characterized but not tested.

Note 1: Response time measured with one comparator input at (VDD – 1.5)/2 while the other input transitions from VSS to VDD.

TABLE 17-2: VOLTAGE REFERENCE SPECIFICATIONS

Operating Conditions:	3.0V < VDD < 5.5V, -40°C < TA < +85°C (unless otherwise stated) 4.0V < VDD < 5.5V, -40°C < TA < +125°C (unless otherwise stated)						
Spec No.	**Sym**	**Characteristics**	**Min**	**Typ**	**Max**	**Units**	**Comments**
D310	VRES	Resolution	VDD/24	—	VDD/32	LSb	
D311	VRAA	Absolute Accuracy	— —	— —	1/2 1/2	LSb LSb	Low Range (VRR = 1) High Range (VRR = 0)
D312	VRUR	Unit Resistor Value (R)*	—	2k	—	Ω	
310	TSET	Settling Time*(1)	—	—	10	µs	

* These parameters are characterized but not tested.

Note 1: Settling time measured while VRR = 1 and VR<3:0> transitions from '0000' to '1111'.

© 2003 Microchip Technology Inc.

libros.ep-electropc.com

libros.ep-electropc.com

APÉNDICE B

SET DE INSTRUCCIONES. DESCRIPCIÓN DE LOS CAMPOS. FORMATO DE INSTRUCCIONES

libros.ep-electropc.com

SET DE INSTRUCCIONES

TABLE 15-2: PIC16F87XA INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1,2
CLRF	f	Clear f	1	00 0001 1fff ffff		Z	2
CLRW	-	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1,2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1,2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1,2
MOVWF	f	Move W to f	1	00 0000 1fff ffff			
NOP	-	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1,2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1,2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1,2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1,2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add Literal and W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	k	AND Literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$,$\overline{PD}$	
GOTO	k	Go to Address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR Literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move Literal to W	1	11 00xx kkkk kkkk			
RETFIE	-	Return from Interrupt	2	00 0000 0000 1001			
RETLW	k	Return with Literal in W	2	11 01xx kkkk kkkk			
RETURN	-	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011		$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from Literal	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	k	Exclusive OR Literal with W	1	11 1010 kkkk kkkk		Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a `NOP`.

Note: Additional information on the mid-range instruction set is available in the PICmicro® Mid-Range MCU Family Reference Manual (DS33023).

libros.ep-electropc.com

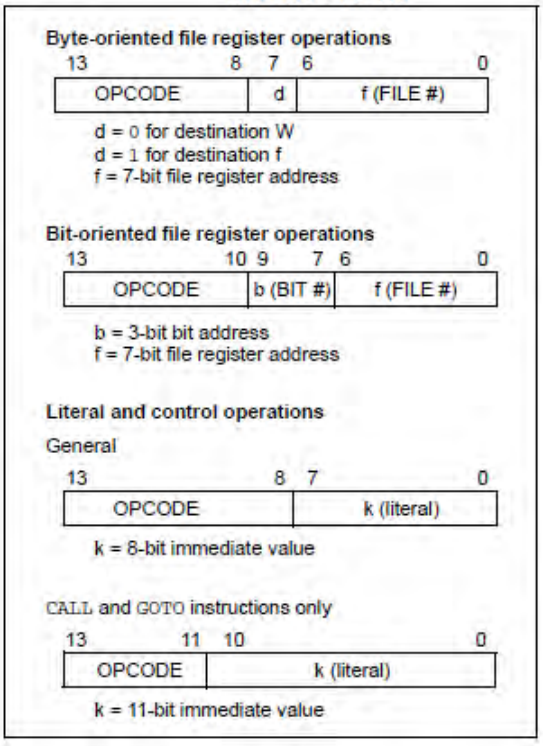
DESCRIPCIÓN DE LOS CAMPOS

TABLE 15-1: OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1.
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

FORMATO DE INSTRUCCIONES

FIGURE 15-1: GENERAL FORMAT FOR INSTRUCTIONS



libros.ep-electropc.com

libros.ep-electropc.com

APÉNDICE C

OPERADORES

libros.ep-electropc.com

OPERADORES

Table: Arithmetic Operators in Order of Precedence

Operator		Example
$	Current/Return program counter	goto $ + 3
(	Left Parenthesis	1 + (d * 4)
)	Right Parenthesis	(Length + 1) * 256
!	Item NOT (logical complement)	if ! (a == b)
-	Negation (2's complement)	-1 * Length
~	Complement	flags = ~flags
low[1]	Return low byte of address	movlw low CTR_Table
high[1]	Return high byte of address	movlw high CTR_Table
upper[1]	Return upper byte of address	movlw upper CTR_Table
*	Multiply	a = b * c
/	Divide	a = b / c
%	Modulus	entry_len = tot_len % 16
+	Add	tot_len = entry_len * 8 + 1
-	Subtract	entry_len = (tot - 1) / 8
<<	Left shift	flags = flags << 1
>>	Right shift	flags = flags >> 1
>=	Greater or equal	if entry_idx >= num_entries
>	Greater than	if entry_idx > num_entries
<	Less than	if entry_idx < num_entries
<=	Less or equal	if entry_idx <= num_entries
==	Equal to	if entry_idx == num_entries
!=	Not equal to	if entry_idx != num_entries
&	Bitwise AND	flags = flags & ERROR_BIT
^	Bitwise exclusive OR	flags = flags ^ ERROR_BIT
|	Bitwise inclusive OR	flags = flags | ERROR_BIT
&&	Logical AND	if (len == 512) && (b == c)
||	Logical OR	if (len == 512) || (b == c)
=	Set equal to	entry_index = 0
+=	Add to, set equal	entry_index += 1
-=	Subtract, set equal	entry_index -= 1
*=	Multiply, set equal	entry_index *= entry_length
/=	Divide, set equal	entry_total /= entry_length
%=	Modulus, set equal	entry_index %= 8
<<=	Left shift, set equal	flags <<= 3
>>=	Right shift, set equal	flags >>= 3
&=	AND, set equal	flags &= ERROR_FLAG
|=	Inclusive OR, set equal	flags |= ERROR_FLAG
^=	Exclusive OR, set equal	flags ^= ERROR_FLAG
++	Increment[2]	i ++
--	Decrement[2]	i --

MPASM Asembler Help
Arithmetic Operators and Precedence

libros.ep-electropc.com

CONSTANTES

MPASM Radix Types Supported

Radix	Syntax	Example
Binary	B‘<binary_digits>’	B‘00111001’
Octal	O‘<octal_digits>’	O‘777’
Decimal	D‘<digits>’ .<digits>	D‘100’ .100
Hexadecimal (default)	H‘<hex_digits>’ 0x<hex_digits>	H‘9f’ 0x9f
Character (ASCII)	A‘<Character>’ ‘<character>’	A‘C’ ‘C’

libros.ep-electropc.com

libros.ep-electropc.com

APÉNDICE D

DIRECTIVAS DEL MPASM

libros.ep-electropc.com

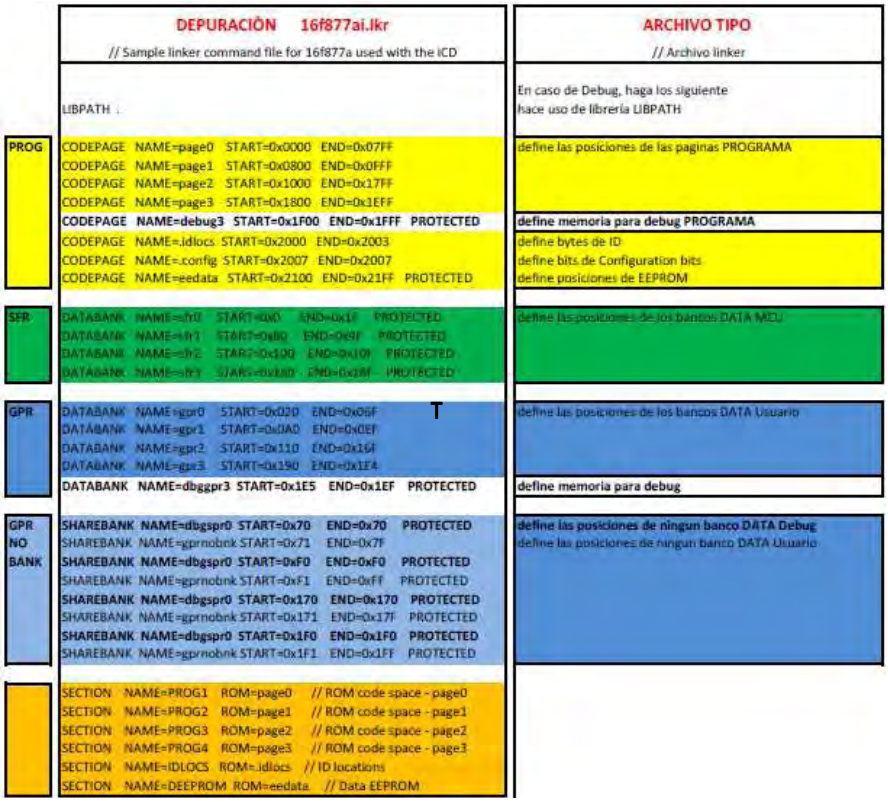
ARCHIVOS LINKER *.LKR

PROGRAMACIÒN_16f877a_lkr.pdf



libros.ep-electropc.com

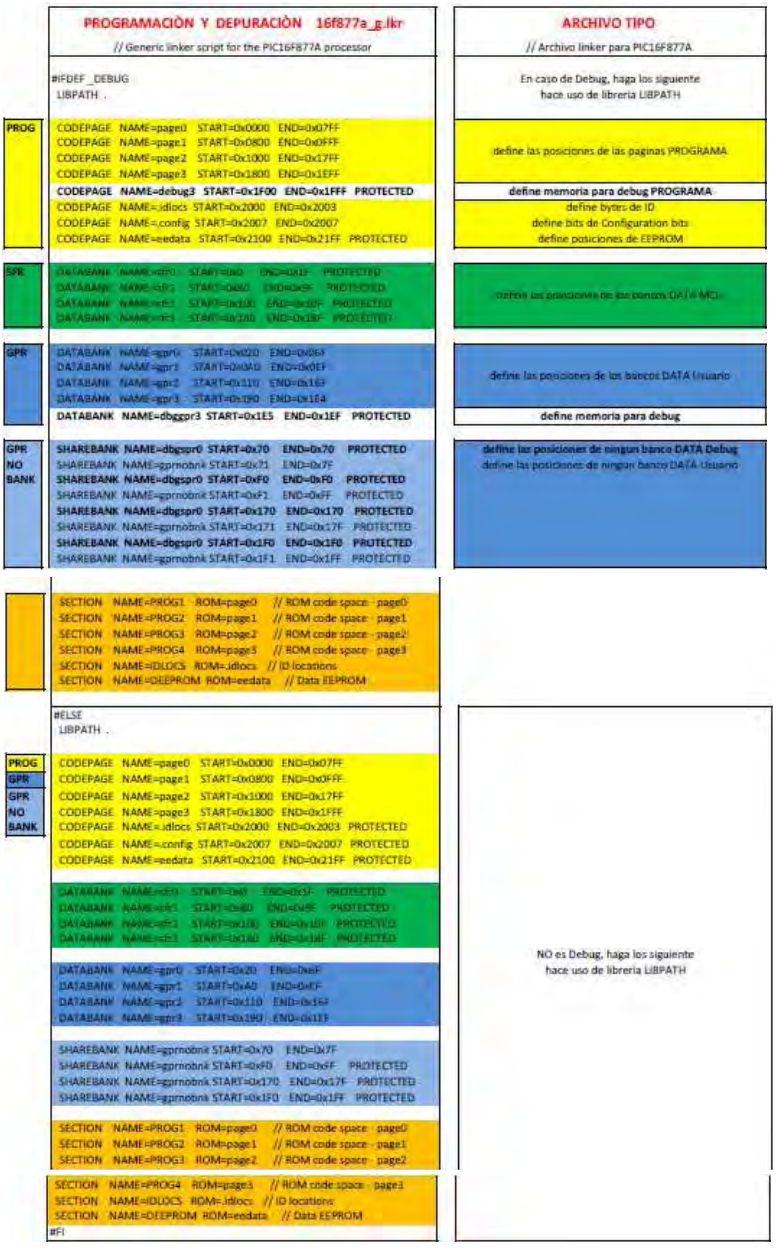
DEPURACIÓN_16F877AI_LKR



	DEPURACIÓN 16f877ai.lkr // Sample linker command file for 16f877a used with the ICD	ARCHIVO TIPO // Archivo linker
	LIBPATH .	En caso de Debug, haga los siguiente hace uso de librería LIBPATH
PROG	CODEPAGE NAME=page0 START=0x0000 END=0x07FF CODEPAGE NAME=page1 START=0x0800 END=0x0FFF CODEPAGE NAME=page2 START=0x1000 END=0x17FF CODEPAGE NAME=page3 START=0x1800 END=0x1EFF	define las posiciones de las paginas PROGRAMA
	CODEPAGE NAME=debug3 START=0x1F00 END=0x1FFF PROTECTED	**define memoria para debug PROGRAMA**
	CODEPAGE NAME=.idlocs START=0x2000 END=0x2003 CODEPAGE NAME=.config START=0x2007 END=0x2007 CODEPAGE NAME=eedata START=0x2100 END=0x21FF PROTECTED	define bytes de ID define bits de Configuration bits define posiciones de EEPROM
SFR	DATABANK NAME=sfr0 START=0x0 END=0x1F PROTECTED DATABANK NAME=sfr1 START=0x80 END=0x9F PROTECTED DATABANK NAME=sfr2 START=0x100 END=0x109 PROTECTED DATABANK NAME=sfr3 START=0x180 END=0x18F PROTECTED	define las posiciones de los bancos DATA MCU
GPR	DATABANK NAME=gpr0 START=0x020 END=0x06F DATABANK NAME=gpr1 START=0x0A0 END=0x0EF DATABANK NAME=gpr2 START=0x110 END=0x16F DATABANK NAME=gpr3 START=0x190 END=0x1F4	define las posiciones de los bancos DATA Usuario
	DATABANK NAME=dbggpr3 START=0x1E5 END=0x1EF PROTECTED	**define memoria para debug**
GPR NO BANK	**SHAREBANK NAME=dbgspr0 START=0x70 END=0x70 PROTECTED** SHAREBANK NAME=gprnobnk START=0x71 END=0x7F **SHAREBANK NAME=dbgspr0 START=0xF0 END=0xF0 PROTECTED** SHAREBANK NAME=gprnobnk START=0xF1 END=0xFF PROTECTED **SHAREBANK NAME=dbgspr0 START=0x170 END=0x170 PROTECTED** SHAREBANK NAME=gprnobnk START=0x171 END=0x17F PROTECTED **SHAREBANK NAME=dbgspr0 START=0x1F0 END=0x1F0 PROTECTED** SHAREBANK NAME=gprnobnk START=0x1F1 END=0x1FF PROTECTED	**define las posiciones de ningun banco DATA Debug** define las posiciones de ningun banco DATA Usuario
	SECTION NAME=PROG1 ROM=page0 // ROM code space - page0 SECTION NAME=PROG2 ROM=page1 // ROM code space - page1 SECTION NAME=PROG3 ROM=page2 // ROM code space - page2 SECTION NAME=PROG4 ROM=page3 // ROM code space - page3 SECTION NAME=IDLOCS ROM=.idlocs // ID locations SECTION NAME=DEEPROM ROM=eedata // Data EEPROM	

libros.ep-electropc.com

PROGRAMACIÓN Y DEPURACIÓN_16F877A_G.LKR



libros.ep-electropc.com

DEFINICIONES *.INC

P16F877a.INC
P16F887.INC

```
        LIST
; P16F877A.INC  Standard Header File, Version 1.00
Microchip Technology, Inc.
        NOLIST

; This header file defines configurations, registers, and
other useful bits of
; information for the PIC16F877A microcontroller.  These
names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file
is
; included.  The processor may be selected the following
ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /PIC16F877A
;       2. LIST directive in the source file
;               LIST   P=PIC16F877A
;       3. Processor Type entry in the MPASM full-screen
interface

;==========================================================
================
;
;       Revision History
;
;==========================================================
================

;Rev:   Date:    Reason:
;1.03   11/17/05 Added the ADCON1 bit ADCS2.
;1.02 05/28/02 Corrected values for _CP_ALL and _CP_OFF in
Configuration Bits section.
;1.01   09/13/01 Added the PIR2 bit CMIF and the PIE2 bit
CMIE
;1.00   04/19/01 Initial Release (BD - generated from
PIC16F877.inc)

;==========================================================
================
```


libros.ep-electropc.com

```
;
;       Verify Processor
;
;==========================================================
================

        IFNDEF __16F877A
           MESSG "Processor-header file mismatch.  Verify
selected processor."
        ENDIF

;==========================================================
================
;
;       Register Definitions
;
;==========================================================
================

W                               EQU     H'0000'
F                               EQU     H'0001'

;----- Register Files------------------------------------
----------------

INDF                            EQU     H'0000'
TMR0                            EQU     H'0001'
PCL                             EQU     H'0002'
STATUS                          EQU     H'0003'
FSR                             EQU     H'0004'
PORTA                           EQU     H'0005'
PORTB                           EQU     H'0006'
PORTC                           EQU     H'0007'
PORTD                           EQU     H'0008'
PORTE                           EQU     H'0009'
PCLATH                          EQU     H'000A'
INTCON                          EQU     H'000B'
PIR1                            EQU     H'000C'
PIR2                            EQU     H'000D'
TMR1L                           EQU     H'000E'
TMR1H                           EQU     H'000F'
T1CON                           EQU     H'0010'
TMR2                            EQU     H'0011'
T2CON                           EQU     H'0012'
SSPBUF                          EQU     H'0013'
SSPCON                          EQU     H'0014'
CCPR1L                          EQU     H'0015'
CCPR1H                          EQU     H'0016'
CCP1CON                         EQU     H'0017'
RCSTA                           EQU     H'0018'
TXREG                           EQU     H'0019'
```


libros.ep-electropc.com

```
RCREG                           EQU     H'001A'
CCPR2L                          EQU     H'001B'
CCPR2H                          EQU     H'001C'
CCP2CON                         EQU     H'001D'
ADRESH                          EQU     H'001E'
ADCON0                          EQU     H'001F'

OPTION_REG                      EQU     H'0081'
TRISA                           EQU     H'0085'
TRISB                           EQU     H'0086'
TRISC                           EQU     H'0087'
TRISD                           EQU     H'0088'
TRISE                           EQU     H'0089'
PIE1                            EQU     H'008C'
PIE2                            EQU     H'008D'
PCON                            EQU     H'008E'
SSPCON2                         EQU     H'0091'
PR2                             EQU     H'0092'
SSPADD                          EQU     H'0093'
SSPSTAT                         EQU     H'0094'
TXSTA                           EQU     H'0098'
SPBRG                           EQU     H'0099'
CMCON                           EQU     H'009C'
CVRCON                          EQU     H'009D'
ADRESL                          EQU     H'009E'
ADCON1                          EQU     H'009F'

EEDATA                          EQU     H'010C'
EEADR                           EQU     H'010D'
EEDATH                          EQU     H'010E'
EEADRH                          EQU     H'010F'

EECON1                          EQU     H'018C'
EECON2                          EQU     H'018D'

;----- STATUS Bits --------------------------------------
----------------

IRP                             EQU     H'0007'
RP1                             EQU     H'0006'
RP0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- INTCON Bits --------------------------------------
----------------

GIE                             EQU     H'0007'
```


libros.ep-electropc.com

```
PEIE                            EQU     H'0006'
T0IE                            EQU     H'0005'
TMR0IE                          EQU     H'0005'
INTE                            EQU     H'0004'
RBIE                            EQU     H'0003'
T0IF                            EQU     H'0002'
TMR0IF                          EQU     H'0002'
INTF                            EQU     H'0001'
RBIF                            EQU     H'0000'

;----- PIR1 Bits --------------------------------------
----------------

PSPIF                           EQU     H'0007'
ADIF                            EQU     H'0006'
RCIF                            EQU     H'0005'
TXIF                            EQU     H'0004'
SSPIF                           EQU     H'0003'
CCP1IF                          EQU     H'0002'
TMR2IF                          EQU     H'0001'
TMR1IF                          EQU     H'0000'

;----- PIR2 Bits --------------------------------------
----------------

CMIF                    EQU     H'0006'
EEIF                            EQU     H'0004'
BCLIF                           EQU     H'0003'
CCP2IF                          EQU     H'0000'

;----- T1CON Bits -------------------------------------
----------------

T1CKPS1                         EQU     H'0005'
T1CKPS0                         EQU     H'0004'
T1OSCEN                         EQU     H'0003'
NOT_T1SYNC                      EQU     H'0002'
T1INSYNC                        EQU     H'0002'    ; Backward
compatibility only
T1SYNC                          EQU     H'0002'
TMR1CS                          EQU     H'0001'
TMR1ON                          EQU     H'0000'

;----- T2CON Bits -------------------------------------
----------------

TOUTPS3                         EQU     H'0006'
TOUTPS2                         EQU     H'0005'
TOUTPS1                         EQU     H'0004'
TOUTPS0                         EQU     H'0003'
TMR2ON                          EQU     H'0002'
```


libros.ep-electropc.com

```
T2CKPS1                         EQU     H'0001'
T2CKPS0                         EQU     H'0000'

;----- SSPCON Bits --------------------------------------
----------------

WCOL                            EQU     H'0007'
SSPOV                           EQU     H'0006'
SSPEN                           EQU     H'0005'
CKP                             EQU     H'0004'
SSPM3                           EQU     H'0003'
SSPM2                           EQU     H'0002'
SSPM1                           EQU     H'0001'
SSPM0                           EQU     H'0000'

;----- CCP1CON Bits -------------------------------------
----------------

CCP1X                           EQU     H'0005'
CCP1Y                           EQU     H'0004'
CCP1M3                          EQU     H'0003'
CCP1M2                          EQU     H'0002'
CCP1M1                          EQU     H'0001'
CCP1M0                          EQU     H'0000'

;----- RCSTA Bits ---------------------------------------
----------------

SPEN                            EQU     H'0007'
RX9                             EQU     H'0006'
RC9                             EQU     H'0006'    ; Backward
compatibility only
NOT_RC8                         EQU     H'0006'    ; Backward
compatibility only
RC8_9                           EQU     H'0006'    ; Backward
compatibility only
SREN                            EQU     H'0005'
CREN                            EQU     H'0004'
ADDEN                           EQU     H'0003'
FERR                            EQU     H'0002'
OERR                            EQU     H'0001'
RX9D                            EQU     H'0000'
RCD8                            EQU     H'0000'    ; Backward
compatibility only

;----- CCP2CON Bits -------------------------------------
----------------

CCP2X                           EQU     H'0005'
CCP2Y                           EQU     H'0004'
CCP2M3                          EQU     H'0003'
```


libros.ep-electropc.com

```
CCP2M2                          EQU     H'0002'
CCP2M1                          EQU     H'0001'
CCP2M0                          EQU     H'0000'

;----- ADCON0 Bits --------------------------------------
----------------

ADCS1                           EQU     H'0007'
ADCS0                           EQU     H'0006'
CHS2                            EQU     H'0005'
CHS1                            EQU     H'0004'
CHS0                            EQU     H'0003'
GO                              EQU     H'0002'
NOT_DONE                        EQU     H'0002'
GO_DONE                         EQU     H'0002'
ADON                            EQU     H'0000'

;----- OPTION_REG Bits ----------------------------------
-----------------

NOT_RBPU                        EQU     H'0007'
INTEDG                          EQU     H'0006'
T0CS                            EQU     H'0005'
T0SE                            EQU     H'0004'
PSA                             EQU     H'0003'
PS2                             EQU     H'0002'
PS1                             EQU     H'0001'
PS0                             EQU     H'0000'

;----- TRISE Bits ---------------------------------------
----------------

IBF                             EQU     H'0007'
OBF                             EQU     H'0006'
IBOV                            EQU     H'0005'
PSPMODE                         EQU     H'0004'
TRISE2                          EQU     H'0002'
TRISE1                          EQU     H'0001'
TRISE0                          EQU     H'0000'

;----- PIE1 Bits ----------------------------------------
----------------

PSPIE                           EQU     H'0007'
ADIE                            EQU     H'0006'
RCIE                            EQU     H'0005'
TXIE                            EQU     H'0004'
SSPIE                           EQU     H'0003'
CCP1IE                          EQU     H'0002'
TMR2IE                          EQU     H'0001'
TMR1IE                          EQU     H'0000'
```


libros.ep-electropc.com

```
;----- PIE2 Bits ----------------------------------------
----------------

CMIE                    EQU     H'0006'
EEIE                            EQU     H'0004'
BCLIE                           EQU     H'0003'
CCP2IE                          EQU     H'0000'

;----- PCON Bits ----------------------------------------
----------------

NOT_POR                         EQU     H'0001'
NOT_BO                          EQU     H'0000'
NOT_BOR                         EQU     H'0000'

;----- SSPCON2 Bits -------------------------------------
-----------------

GCEN                            EQU     H'0007'
ACKSTAT                         EQU     H'0006'
ACKDT                           EQU     H'0005'
ACKEN                           EQU     H'0004'
RCEN                            EQU     H'0003'
PEN                             EQU     H'0002'
RSEN                            EQU     H'0001'
SEN                             EQU     H'0000'

;----- SSPSTAT Bits -------------------------------------
----------------

SMP                             EQU     H'0007'
CKE                             EQU     H'0006'
D                               EQU     H'0005'
I2C_DATA                        EQU     H'0005'
NOT_A                           EQU     H'0005'
NOT_ADDRESS                     EQU     H'0005'
D_A                             EQU     H'0005'
DATA_ADDRESS                    EQU     H'0005'
P                               EQU     H'0004'
I2C_STOP                        EQU     H'0004'
S                               EQU     H'0003'
I2C_START                       EQU     H'0003'
R                               EQU     H'0002'
I2C_READ                        EQU     H'0002'
NOT_W                           EQU     H'0002'
NOT_WRITE                       EQU     H'0002'
R_W                             EQU     H'0002'
READ_WRITE                      EQU     H'0002'
UA                              EQU     H'0001'
BF                              EQU     H'0000'
```


libros.ep-electropc.com

```
;----- TXSTA Bits --------------------------------------
----------------

CSRC                            EQU     H'0007'
TX9                             EQU     H'0006'
NOT_TX8                         EQU     H'0006'    ; Backward
compatibility only
TX8_9                           EQU     H'0006'    ; Backward
compatibility only
TXEN                            EQU     H'0005'
SYNC                            EQU     H'0004'
BRGH                            EQU     H'0002'
TRMT                            EQU     H'0001'
TX9D                            EQU     H'0000'
TXD8                            EQU     H'0000'    ; Backward
compatibility only


;----- CMCON Bits --------------------------------------
----------------
C2OUT                           EQU     H'0007'
C1OUT                           EQU     H'0006'
C2INV                           EQU     H'0005'
C1INV                           EQU     H'0004'
CIS                             EQU     H'0003'
CM2                             EQU     H'0002'
CM1                             EQU     H'0001'
CM0                             EQU     H'0000'

;----- CVRCON Bits --------------------------------------
----------------
CVREN                           EQU     H'0007'
CVROE                           EQU     H'0006'
CVRR                            EQU     H'0005'
CVR3                            EQU     H'0003'
CVR2                            EQU     H'0002'
CVR1                            EQU     H'0001'
CVR0                            EQU     H'0000'

;----- ADCON1 Bits --------------------------------------
----------------

ADFM                            EQU     H'0007'
ADCS2                           EQU     H'0006'
PCFG3                           EQU     H'0003'
PCFG2                           EQU     H'0002'
PCFG1                           EQU     H'0001'
PCFG0                           EQU     H'0000'
```


libros.ep-electropc.com

```
;----- EECON1 Bits --------------------------------------
----------------

EEPGD                           EQU     H'0007'
WRERR                           EQU     H'0003'
WREN                            EQU     H'0002'
WR                              EQU     H'0001'
RD                              EQU     H'0000'

;=========================================================
================
;
;       RAM Definition
;
;=========================================================
================

        __MAXRAM H'1FF'
        __BADRAM H'8F'-H'90', H'95'-H'97', H'9A'-H'9B'
        __BADRAM H'105', H'107'-H'109'
        __BADRAM H'185', H'187'-H'189', H'18E'-H'18F'

;=========================================================
================
;
;       Configuration Bits
;
;=========================================================
================

_CP_ALL                         EQU     H'1FFF'
_CP_OFF                         EQU     H'3FFF'
_DEBUG_OFF                      EQU     H'3FFF'
_DEBUG_ON                       EQU     H'37FF'
_WRT_OFF                        EQU     H'3FFF'    ; No prog
memmory write protection
_WRT_256                        EQU     H'3DFF'    ; First 256
prog memmory write protected
_WRT_1FOURTH                    EQU     H'3BFF'    ; First
quarter prog memmory write protected
_WRT_HALF                       EQU     H'39FF'    ; First
half memmory write protected
_CPD_OFF                        EQU     H'3FFF'
_CPD_ON                         EQU     H'3EFF'
_LVP_ON                         EQU     H'3FFF'
_LVP_OFF                        EQU     H'3F7F'
_BODEN_ON                       EQU     H'3FFF'
_BODEN_OFF                      EQU     H'3FBF'
_PWRTE_OFF                      EQU     H'3FFF'
_PWRTE_ON                       EQU     H'3FF7'
_WDT_ON                         EQU     H'3FFF'
```


libros.ep-electropc.com

```
_WDT_OFF                        EQU     H'3FFB'
_RC_OSC                         EQU     H'3FFF'
_HS_OSC                         EQU     H'3FFE'
_XT_OSC                         EQU     H'3FFD'
_LP_OSC                         EQU     H'3FFC'

        LIST


        LIST
; P16F887.INC  Standard Header File, Version 1.00
Microchip Technology, Inc.
        NOLIST

; This header file defines configurations, registers, and
other useful bits of
; information for the PIC16F887 microcontroller.  These
names are taken to match
; the data sheets as closely as possible.

; Note that the processor must be selected before this file
is
; included.  The processor may be selected the following
ways:

;       1. Command line switch:
;               C:\ MPASM MYFILE.ASM /PIC16F887
;       2. LIST directive in the source file
;               LIST   P=PIC16F887
;       3. Processor Type entry in the MPASM full-screen
interface

;==========================================================
================
;
;       Revision History
;
;==========================================================
================
;
;1.00   11/18/05 Original
;
;==========================================================
================
;
;       Verify Processor
;
;==========================================================
================
```


libros.ep-electropc.com

```
        IFNDEF __16F887
           MESSG "Processor-header file mismatch.  Verify
selected processor."
        ENDIF

;==========================================================
================
;
;       Register Definitions
;
;==========================================================
================

W                               EQU     H'0000'
F                               EQU     H'0001'

;----- Register Files------------------------------------
----------------

INDF                            EQU     H'0000'
TMR0                            EQU     H'0001'
PCL                             EQU     H'0002'
STATUS                          EQU     H'0003'
FSR                             EQU     H'0004'
PORTA                           EQU     H'0005'
PORTB                           EQU     H'0006'
PORTC                           EQU     H'0007'
PORTD                           EQU     H'0008'
PORTE                           EQU     H'0009'
PCLATH                          EQU     H'000A'
INTCON                          EQU     H'000B'
PIR1                            EQU     H'000C'
PIR2                            EQU     H'000D'
TMR1L                           EQU     H'000E'
TMR1H                           EQU     H'000F'
T1CON                           EQU     H'0010'
TMR2                            EQU     H'0011'
T2CON                           EQU     H'0012'
SSPBUF                          EQU     H'0013'
SSPCON                          EQU     H'0014'
CCPR1L                          EQU     H'0015'
CCPR1H                          EQU     H'0016'
CCP1CON                         EQU     H'0017'
RCSTA                           EQU     H'0018'
TXREG                           EQU     H'0019'
RCREG                           EQU     H'001A'
CCPR2L                          EQU     H'001B'
CCPR2H                          EQU     H'001C'
CCP2CON                         EQU     H'001D'
ADRESH                          EQU     H'001E'
ADCON0                          EQU     H'001F'
```


libros.ep-electropc.com

```
OPTION_REG                  EQU     H'0081'

TRISA                       EQU     H'0085'
TRISB                       EQU     H'0086'
TRISC                       EQU     H'0087'
TRISD                       EQU     H'0088'
TRISE                       EQU     H'0089'

PIE1                        EQU     H'008C'
PIE2                        EQU     H'008D'
PCON                        EQU     H'008E'
OSCCON                      EQU     H'008F'
OSCTUNE                     EQU     H'0090'
SSPCON2                     EQU     H'0091'
PR2                         EQU     H'0092'
SSPADD                      EQU     H'0093'
SSPMSK                      EQU     H'0093'
MSK                         EQU     H'0093'
SSPSTAT                     EQU     H'0094'
WPUB                EQU       H'0095'
IOCB                EQU       H'0096'
VRCON               EQU       H'0097'
TXSTA                       EQU     H'0098'
SPBRG                       EQU     H'0099'
SPBRGH                      EQU     H'009A'
PWM1CON                     EQU     H'009B'
ECCPAS                      EQU     H'009C'
PSTRCON                     EQU     H'009D'
ADRESL                      EQU     H'009E'
ADCON1                      EQU     H'009F'

WDTCON                      EQU     H'0105'

CM1CON0                     EQU     H'0107'
CM2CON0                     EQU     H'0108'
CM2CON1                     EQU     H'0109'

EEDATA                      EQU     H'010C'
EEDAT                       EQU     H'010C'
EEADR                       EQU     H'010D'
EEDATH                      EQU     H'010E'
EEADRH                      EQU     H'010F'

SRCON               EQU       H'0185'

BAUDCTL                     EQU     H'0187'
ANSEL               EQU       H'0188'
ANSELH                      EQU     H'0189'

EECON1                      EQU     H'018C'
```


libros.ep-electropc.com

```
EECON2                          EQU     H'018D'

;----- BANK 0 REGISTER DEFINITIONS --------------------------
----------------
;----- STATUS Bits ------------------------------------------
----------------

IRP                             EQU     H'0007'
RP1                             EQU     H'0006'
RP0                             EQU     H'0005'
NOT_TO                          EQU     H'0004'
NOT_PD                          EQU     H'0003'
Z                               EQU     H'0002'
DC                              EQU     H'0001'
C                               EQU     H'0000'

;----- INTCON Bits ------------------------------------------
----------------

GIE                             EQU     H'0007'
PEIE                            EQU     H'0006'
T0IE                            EQU     H'0005'
TMR0IE                          EQU     H'0005'
INTE                            EQU     H'0004'
RBIE                            EQU     H'0003'
T0IF                            EQU     H'0002'
TMR0IF                          EQU     H'0002'
INTF                            EQU     H'0001'
RBIF                            EQU     H'0000'

;----- PIR1 Bits --------------------------------------------
----------------

ADIF                            EQU     H'0006'
RCIF                            EQU     H'0005'
TXIF                            EQU     H'0004'
SSPIF                           EQU     H'0003'
CCP1IF                          EQU     H'0002'
TMR2IF                          EQU     H'0001'
TMR1IF                          EQU     H'0000'

;----- PIR2 Bits --------------------------------------------
----------------

OSFIF                   EQU     H'0007'
C2IF                    EQU     H'0006'
C1IF                    EQU     H'0005'
EEIF                            EQU     H'0004'
BCLIF                           EQU     H'0003'
ULPWUIF                         EQU     H'0002'
CCP2IF                          EQU     H'0000'
```


libros.ep-electropc.com

```
;----- T1CON Bits --------------------------------------
----------------

T1GIV                           EQU     H'0007'
TMR1GE                          EQU     H'0006'
T1CKPS1                         EQU     H'0005'
T1CKPS0                         EQU     H'0004'
T1OSCEN                         EQU     H'0003'
NOT_T1SYNC                      EQU     H'0002'
T1INSYNC                        EQU     H'0002'    ; Backward
compatibility only
T1SYNC                          EQU     H'0002'
TMR1CS                          EQU     H'0001'
TMR1ON                          EQU     H'0000'

;----- T2CON Bits --------------------------------------
----------------

TOUTPS3                         EQU     H'0006'
TOUTPS2                         EQU     H'0005'
TOUTPS1                         EQU     H'0004'
TOUTPS0                         EQU     H'0003'
TMR2ON                          EQU     H'0002'
T2CKPS1                         EQU     H'0001'
T2CKPS0                         EQU     H'0000'

;----- SSPCON Bits --------------------------------------
----------------

WCOL                            EQU     H'0007'
SSPOV                           EQU     H'0006'
SSPEN                           EQU     H'0005'
CKP                             EQU     H'0004'
SSPM3                           EQU     H'0003'
SSPM2                           EQU     H'0002'
SSPM1                           EQU     H'0001'
SSPM0                           EQU     H'0000'

;----- CCP1CON Bits --------------------------------------
----------------

P1M1                    EQU     H'0007'
P1M0                    EQU     H'0006'
DC1B1                           EQU     H'0005'
CCP1X                           EQU     H'0005'   ; Backward
compatibility only
DC1B0                           EQU     H'0004'
CCP1Y                           EQU     H'0004'   ; Backward
compatibility only
CCP1M3                          EQU     H'0003'
```


libros.ep-electropc.com

```
CCP1M2                          EQU     H'0002'
CCP1M1                          EQU     H'0001'
CCP1M0                          EQU     H'0000'

;----- RCSTA Bits --------------------------------------
----------------

SPEN                            EQU     H'0007'
RX9                             EQU     H'0006'
RC9                             EQU     H'0006'    ; Backward
compatibility only
NOT_RC8                         EQU     H'0006'    ; Backward
compatibility only
RC8_9                           EQU     H'0006'    ; Backward
compatibility only
SREN                            EQU     H'0005'
CREN                            EQU     H'0004'
ADDEN                           EQU     H'0003'
FERR                            EQU     H'0002'
OERR                            EQU     H'0001'
RX9D                            EQU     H'0000'
RCD8                            EQU     H'0000'    ; Backward
compatibility only

;----- CCP2CON Bits ------------------------------------
----------------

CCP2X                           EQU     H'0005'   ; Backward
compatibility only
DC2B1                           EQU     H'0005'
CCP2Y                           EQU     H'0004'   ; Backward
compatibility only
DC2B0                           EQU     H'0004'
CCP2M3                          EQU     H'0003'
CCP2M2                          EQU     H'0002'
CCP2M1                          EQU     H'0001'
CCP2M0                          EQU     H'0000'

;----- ADCON0 Bits -------------------------------------
----------------

ADCS1                           EQU     H'0007'
ADCS0                           EQU     H'0006'
CHS3                            EQU     H'0005'
CHS2                            EQU     H'0004'
CHS1                            EQU     H'0003'
CHS0                    EQU       H'0002'
GO                              EQU     H'0001'
NOT_DONE                        EQU     H'0001'
GO_DONE                         EQU     H'0001'
ADON                            EQU     H'0000'
```


libros.ep-electropc.com

```
;----- BANK 1 REGISTER DEFINITIONS ------------------------
----------------
;----- OPTION_REG Bits -----------------------------------
-----------------

NOT_RBPU                        EQU     H'0007'
INTEDG                          EQU     H'0006'
T0CS                            EQU     H'0005'
T0SE                            EQU     H'0004'
PSA                             EQU     H'0003'
PS2                             EQU     H'0002'
PS1                             EQU     H'0001'
PS0                             EQU     H'0000'

;----- PIE1 Bits -----------------------------------------
----------------

ADIE                            EQU     H'0006'
RCIE                            EQU     H'0005'
TXIE                            EQU     H'0004'
SSPIE                           EQU     H'0003'
CCP1IE                          EQU     H'0002'
TMR2IE                          EQU     H'0001'
TMR1IE                          EQU     H'0000'

;----- PIE2 Bits -----------------------------------------
----------------

OSFIE                   EQU     H'0007'
C2IE                    EQU     H'0006'
C1IE                    EQU     H'0005'
EEIE                            EQU     H'0004'
BCLIE                           EQU     H'0003'
ULPWUIE                         EQU     H'0002'
CCP2IE                          EQU     H'0000'

;----- PCON Bits -----------------------------------------
----------------

ULPWUE                          EQU     H'0005'
SBOREN                          EQU     H'0004'
NOT_POR                         EQU     H'0001'
NOT_BO                          EQU     H'0000'
NOT_BOR                         EQU     H'0000'

;----- OSCCON Bits ---------------------------------------
----------------

IRCF2                           EQU     H'0006'
IRCF1                           EQU     H'0005'
```


libros.ep-electropc.com

```
IRCF0                           EQU     H'0004'
OSTS                            EQU     H'0003'
HTS                             EQU     H'0002'
LTS                             EQU     H'0001'
SCS                             EQU     H'0000'

;----- OSCTUNE Bits --------------------------------------
----------------

TUN4                            EQU     H'0004'
TUN3                            EQU     H'0003'
TUN2                            EQU     H'0002'
TUN1                            EQU     H'0001'
TUN0                            EQU     H'0000'

;----- SSPCON2 Bits --------------------------------------
-----------------

GCEN                            EQU     H'0007'
ACKSTAT                         EQU     H'0006'
ACKDT                           EQU     H'0005'
ACKEN                           EQU     H'0004'
RCEN                            EQU     H'0003'
PEN                             EQU     H'0002'
RSEN                            EQU     H'0001'
SEN                             EQU     H'0000'

;----- SSPSTAT Bits --------------------------------------
----------------

SMP                             EQU     H'0007'
CKE                             EQU     H'0006'
D                               EQU     H'0005'
I2C_DATA                        EQU     H'0005'
NOT_A                           EQU     H'0005'
NOT_ADDRESS                     EQU     H'0005'
D_A                             EQU     H'0005'
DATA_ADDRESS                    EQU     H'0005'
P                               EQU     H'0004'
I2C_STOP                        EQU     H'0004'
S                               EQU     H'0003'
I2C_START                       EQU     H'0003'
R                               EQU     H'0002'
I2C_READ                        EQU     H'0002'
NOT_W                           EQU     H'0002'
NOT_WRITE                       EQU     H'0002'
R_W                             EQU     H'0002'
READ_WRITE                      EQU     H'0002'
UA                              EQU     H'0001'
BF                              EQU     H'0000'
```


libros.ep-electropc.com

```
;----- WPUB Bits --------------------------------------------
----------------

WPUB7                   EQU     H'0007'
WPUB6                   EQU     H'0006'
WPUB5                   EQU     H'0005'
WPUB4                   EQU     H'0004'
WPUB3                   EQU     H'0003'
WPUB2                   EQU     H'0002'
WPUB1                   EQU     H'0001'
WPUB0                   EQU     H'0000'

;----- IOCB Bits --------------------------------------------
----------------

IOCB7                   EQU     H'0007'
IOCB6                   EQU     H'0006'
IOCB5                   EQU     H'0005'
IOCB4                   EQU     H'0004'
IOCB3                   EQU     H'0003'
IOCB2                   EQU     H'0002'
IOCB1                   EQU     H'0001'
IOCB0                   EQU     H'0000'

;----- VRCON Bits -------------------------------------------
----------------

VREN                    EQU     H'0007'
VROE                    EQU     H'0006'
VRR                     EQU     H'0005'
VRSS                    EQU     H'0004'
VR3                     EQU     H'0003'
VR2                     EQU     H'0002'
VR1                     EQU     H'0001'
VR0                     EQU     H'0000'

;----- TXSTA Bits -------------------------------------------
----------------

CSRC                            EQU     H'0007'
TX9                             EQU     H'0006'
NOT_TX8                         EQU     H'0006'    ; Backward
compatibility only
TX8_9                           EQU     H'0006'    ; Backward
compatibility only
TXEN                            EQU     H'0005'
SYNC                            EQU     H'0004'
SENDB                           EQU     H'0003'
BRGH                            EQU     H'0002'
TRMT                            EQU     H'0001'
TX9D                            EQU     H'0000'
```


libros.ep-electropc.com

```
TXD8                            EQU     H'0000'    ; Backward
compatibility only

;----- SPBRG Bits --------------------------------------
--------------

BRG7                            EQU     H'0007'
BRG6                            EQU     H'0006'
BRG5                            EQU     H'0005'
BRG4                            EQU     H'0004'
BRG3                            EQU     H'0003'
BRG2                            EQU     H'0002'
BRG1                            EQU     H'0001'
BRG0                            EQU     H'0000'

;----- SPBRGH Bits -------------------------------------
---------------

BRG15                           EQU     H'0007'
BRG14                           EQU     H'0006'
BRG13                           EQU     H'0005'
BRG12                           EQU     H'0004'
BRG11                           EQU     H'0003'
BRG10                           EQU     H'0002'
BRG9                            EQU     H'0001'
BRG8                            EQU     H'0000'

;----- PWM1CON Bits ------------------------------------
----------------

PRSEN                           EQU     H'0007'
PDC6                            EQU     H'0006'
PDC5                            EQU     H'0005'
PDC4                            EQU     H'0004'
PDC3                            EQU     H'0003'
PDC2                            EQU     H'0002'
PDC1                            EQU     H'0001'
PDC0                            EQU     H'0000'

;----- ECCPAS Bits -------------------------------------
----------------

ECCPASE                         EQU     H'0007'
ECCPAS2                         EQU     H'0006'
ECCPAS1                         EQU     H'0005'
ECCPAS0                         EQU     H'0004'
PSSAC1                          EQU     H'0003'
PSSAC0                          EQU     H'0002'
PSSBD1                          EQU     H'0001'
PSSBD0                          EQU     H'0000'
```


libros.ep-electropc.com

```
;----- PSTRCON Bits --------------------------------------
-----------------

STRSYNC                         EQU     H'0004'
STRD                            EQU     H'0003'
STRC                            EQU     H'0002'
STRB                            EQU     H'0001'
STRA                            EQU     H'0000'

;----- ADCON1 Bits ---------------------------------------
----------------

ADFM                    EQU       H'0007'
VCFG1                   EQU       H'0005'
VCFG0                   EQU       H'0004'

;----- BANK 2 REGISTER DEFINITIONS -----------------------
----------------
;----- WDTCON Bits ---------------------------------------
----------------

WDTPS3                          EQU     H'0004'
WDTPS2                          EQU     H'0003'
WDTPS1                          EQU     H'0002'
WDTPS0                          EQU     H'0001'
SWDTEN                          EQU     H'0000'

;----- CM1CON0 Bits --------------------------------------
----------------

C1ON                            EQU     H'0007'
C1OUT                           EQU     H'0006'
C1OE                            EQU     H'0005'
C1POL                           EQU     H'0004'

C1R                             EQU     H'0002'
C1CH1                           EQU     H'0001'
C1CH0                           EQU     H'0000'

;----- CM2CON0 Bits --------------------------------------
----------------

C2ON                            EQU     H'0007'
C2OUT                           EQU     H'0006'
C2OE                            EQU     H'0005'
C2POL                           EQU     H'0004'

C2R                             EQU     H'0002'
C2CH1                           EQU     H'0001'
C2CH0                           EQU     H'0000'
```


libros.ep-electropc.com

```
;----- CM2CON1 Bits --------------------------------------
----------------

MC1OUT                          EQU      H'0007'
MC2OUT                          EQU      H'0006'
C1RSEL                         EQU      H'0005'
C2RSEL                         EQU      H'0004'

T1GSS                           EQU      H'0001'
C2SYNC                          EQU      H'0000'

;----- BANK 3 REGISTER DEFINITIONS -----------------------
----------------
;----- SRCON Bits ----------------------------------------
-----------------

SR1                              EQU     H'0007'
SR0                              EQU     H'0006'
C1SEN                            EQU     H'0005'
C2REN                            EQU     H'0004'
PULSS                            EQU     H'0003'
PULSR                            EQU     H'0002'

FVREN                     EQU     H'0000'

;----- BAUDCTL Bits --------------------------------------
----------------

ABDOVF                          EQU     H'0007'
RCIDL                           EQU     H'0006'

SCKP                            EQU     H'0004'
BRG16                           EQU     H'0003'

WUE                             EQU     H'0001'
ABDEN                           EQU     H'0000'

;----- ANSEL Bits ----------------------------------------
----------------

ANS7                            EQU     H'0007'
ANS6                            EQU     H'0006'
ANS5                            EQU     H'0005'
ANS4                            EQU     H'0004'
ANS3                            EQU     H'0003'
ANS2                            EQU     H'0002'
ANS1                            EQU     H'0001'
ANS0                            EQU     H'0000'

;----- ANSELH Bits ---------------------------------------
----------------
```


libros.ep-electropc.com

```
ANS13                   EQU     H'0005'
ANS12                   EQU     H'0004'
ANS11                           EQU     H'0003'
ANS10                           EQU     H'0002'
ANS9                            EQU     H'0001'
ANS8                            EQU     H'0000'

;----- EECON1 Bits --------------------------------------
----------------

EEPGD                           EQU     H'0007'

WRERR                           EQU     H'0003'
WREN                            EQU     H'0002'
WR                              EQU     H'0001'
RD                              EQU     H'0000'

;=======================================================
================
;
;       RAM Definition
;
;=======================================================
================

        __MAXRAM H'1FF'
        __BADRAM H'18E'-H'18F'

;=======================================================
================
;
;       Configuration Bits
;
;=======================================================
================
_CONFIG1                        EQU     H'2007'
_CONFIG2                        EQU     H'2008'

;----- Configuration Word1 ------------------------------
----------------

_DEBUG_ON                       EQU     H'1FFF'
_DEBUG_OFF                      EQU     H'3FFF'
_LVP_ON                        EQU     H'3FFF'
_LVP_OFF                EQU     H'2FFF'
_FCMEN_ON                       EQU     H'3FFF'
_FCMEN_OFF                      EQU     H'37FF'
_IESO_ON                        EQU     H'3FFF'
_IESO_OFF                       EQU     H'3BFF'
_BOR_ON                         EQU     H'3FFF'
```


libros.ep-electropc.com

```
_BOR_NSLEEP                     EQU     H'3EFF'
_BOR_SBODEN                     EQU     H'3DFF'
_BOR_OFF                        EQU     H'3CFF'
_CPD_ON                         EQU     H'3F7F'
_CPD_OFF                        EQU     H'3FFF'
_CP_ON                          EQU     H'3FBF'
_CP_OFF                         EQU     H'3FFF'
_MCLRE_ON                       EQU     H'3FFF'
_MCLRE_OFF                      EQU     H'3FDF'
_PWRTE_ON                       EQU     H'3FEF'
_PWRTE_OFF                      EQU     H'3FFF'
_WDT_ON                         EQU     H'3FFF'
_WDT_OFF                        EQU     H'3FF7'
_LP_OSC                         EQU     H'3FF8'
_XT_OSC                         EQU     H'3FF9'
_HS_OSC                         EQU     H'3FFA'
_EC_OSC                         EQU     H'3FFB'
_INTRC_OSC_NOCLKOUT             EQU     H'3FFC'
_INTRC_OSC_CLKOUT               EQU     H'3FFD'
_EXTRC_OSC_NOCLKOUT             EQU     H'3FFE'
_EXTRC_OSC_CLKOUT               EQU     H'3FFF'
_INTOSCIO                       EQU     H'3FFC'
_INTOSC                         EQU     H'3FFD'
_EXTRCIO                        EQU     H'3FFE'
_EXTRC                          EQU     H'3FFF'

;----- Configuration Word2 ----------------------------------
----------------

_WRT_OFF                        EQU     H'3FFF'    ; No prog
memmory write protection
_WRT_256                        EQU     H'3DFF'    ; First 256
prog memmory write protected
_WRT_1FOURTH                    EQU     H'3BFF'    ; First
quarter prog memmory write protected
_WRT_HALF                       EQU     H'39FF'    ; First
half memmory write protected

_BOR21V                        EQU     H'3EFF'
_BOR40V                        EQU     H'3FFF'

        LIST
```


libros.ep-electropc.com

APÉNDICE E

REGISTROS ESPECIALES

libros.ep-electropc.com

INTCON REGISTER

REGISTER 2-3: INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 **PEIE**: Peripheral Interrupt Enable bit
1 = Enables all unmasked peripheral interrupts
0 = Disables all peripheral interrupts

bit 5 **TMR0IE**: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 **INTE**: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 **RBIE**: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 **TMR0IF**: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 **INTF**: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 **RBIF**: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state; a mismatch condition will continue to set the bit. Reading PORTB will end the mismatch condition and allow the bit to be cleared (must be cleared in software).
0 = None of the RB7:RB4 pins have changed state

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

copiado de PIC16F87XA.PDF pag..26/234

libros.ep-electropc.com

PIE1 REGISTER

REGISTER 2-4: PIE1 REGISTER (ADDRESS 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
bit 7							bit 0

bit 7 **PSPIE:** Parallel Slave Port Read/Write Interrupt Enable bit(1)
1 = Enables the PSP read/write interrupt
0 = Disables the PSP read/write interrupt

Note 1: PSPIE is reserved on PIC16F873A/876A devices; always maintain this bit clear.

bit 6 **ADIE**: A/D Converter Interrupt Enable bit
1 = Enables the A/D converter interrupt
0 = Disables the A/D converter interrupt

bit 5 **RCIE**: USART Receive Interrupt Enable bit
1 = Enables the USART receive interrupt
0 = Disables the USART receive interrupt

bit 4 **TXIE**: USART Transmit Interrupt Enable bit
1 = Enables the USART transmit interrupt
0 = Disables the USART transmit interrupt

bit 3 **SSPIE**: Synchronous Serial Port Interrupt Enable bit
1 = Enables the SSP interrupt
0 = Disables the SSP interrupt

bit 2 **CCP1IE**: CCP1 Interrupt Enable bit
1 = Enables the CCP1 interrupt
0 = Disables the CCP1 interrupt

bit 1 **TMR2IE**: TMR2 to PR2 Match Interrupt Enable bit
1 = Enables the TMR2 to PR2 match interrupt
0 = Disables the TMR2 to PR2 match interrupt

bit 0 **TMR1IE**: TMR1 Overflow Interrupt Enable bit
1 = Enables the TMR1 overflow interrupt
0 = Disables the TMR1 overflow interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

copiado de PIC16F87XA.PDF pag..27/234

libros.ep-electropc.com

PIR1 REGISTER

REGISTER 2-5: PIR1 REGISTER (ADDRESS 0Ch)

R/W-0	R/W-0	R-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
bit 7							bit 0

bit 7 **PSPIF:** Parallel Slave Port Read/Write Interrupt Flag bit(1)
1 = A read or a write operation has taken place (must be cleared in software)
0 = No read or write has occurred
Note 1: PSPIF is reserved on PIC16F873A/876A devices; always maintain this bit clear.

bit 6 **ADIF**: A/D Converter Interrupt Flag bit
1 = An A/D conversion completed
0 = The A/D conversion is not complete

bit 5 **RCIF**: USART Receive Interrupt Flag bit
1 = The USART receive buffer is full
0 = The USART receive buffer is empty

bit 4 **TXIF**: USART Transmit Interrupt Flag bit
1 = The USART transmit buffer is empty
0 = The USART transmit buffer is full

bit 3 **SSPIF**: Synchronous Serial Port (SSP) Interrupt Flag bit
1 = The SSP interrupt condition has occurred and must be cleared in software before returning from the Interrupt Service Routine. The conditions that will set this bit are:
- SPI – A transmission/reception has taken place.
- I^2C Slave – A transmission/reception has taken place.
- I^2C Master
  - A transmission/reception has taken place.
  - The initiated Start condition was completed by the SSP module.
  - The initiated Stop condition was completed by the SSP module.
  - The initiated Restart condition was completed by the SSP module.
  - The initiated Acknowledge condition was completed by the SSP module.
  - A Start condition occurred while the SSP module was Idle (multi-master system).
  - A Stop condition occurred while the SSP module was Idle (multi-master system).

0 = No SSP interrupt condition has occurred

bit 2 **CCP1IF**: CCP1 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused in this mode.

bit 1 **TMR2IF**: TMR2 to PR2 Match Interrupt Flag bit
1 = TMR2 to PR2 match occurred (must be cleared in software)
0 = No TMR2 to PR2 match occurred

bit 0 **TMR1IF**: TMR1 Overflow Interrupt Flag bit
1 = TMR1 register overflowed (must be cleared in software)
0 = TMR1 register did not overflow

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

copiado de PIC16F87XA.PDF pag.. 28/234

libros.ep-electropc.com

PIE2 REGISTER

REGISTER 2-6: PIE2 REGISTER (ADDRESS 8Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIE	—	EEIE	BCLIE	—	—	CCP2IE
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **CMIE**: Comparator Interrupt Enable bit
1 = Enables the comparator interrupt
0 = Disable the comparator interrupt

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIE**: EEPROM Write Operation Interrupt Enable bit
1 = Enable EEPROM write interrupt
0 = Disable EEPROM write interrupt

bit 3 **BCLIE**: Bus Collision Interrupt Enable bit
1 = Enable bus collision interrupt
0 = Disable bus collision interrupt

bit 2-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IE**: CCP2 Interrupt Enable bit
1 = Enables the CCP2 interrupt
0 = Disables the CCP2 interrupt

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

copiado de PIC16F87XA.PDF pag.. 29/234

libros.ep-electropc.com

PIR2 REGISTER

REGISTER 2-7: PIR2 REGISTER (ADDRESS 0Dh)

U-0	R/W-0	U-0	R/W-0	R/W-0	U-0	U-0	R/W-0
—	CMIF	—	EEIF	BCLIF	—	—	CCP2IF
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6 **CMIF**: Comparator Interrupt Flag bit
1 = The comparator input has changed (must be cleared in software)
0 = The comparator input has not changed

bit 5 **Unimplemented:** Read as '0'

bit 4 **EEIF**: EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **BCLIF**: Bus Collision Interrupt Flag bit
1 = A bus collision has occurred in the SSP when configured for I^2C Master mode
0 = No bus collision has occurred

bit 2-1 **Unimplemented:** Read as '0'

bit 0 **CCP2IF**: CCP2 Interrupt Flag bit
Capture mode:
1 = A TMR1 register capture occurred (must be cleared in software)
0 = No TMR1 register capture occurred
Compare mode:
1 = A TMR1 register compare match occurred (must be cleared in software)
0 = No TMR1 register compare match occurred
PWM mode:
Unused.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

copiado de PIC16F87XA.PDF pag. 30/234

libros.ep-electropc.com

PCON REGISTER

REGISTER 2-8: PCON REGISTER (ADDRESS 8Eh)

U-0	U-0	U-0	U-0	U-0	U-0	R/W-0	R/W-1
—	—	—	—	—	—	$\overline{POR}$	$\overline{BOR}$
bit 7							bit 0

bit 7-2 **Unimplemented:** Read as '0'

bit 1 **$\overline{POR}$**: Power-on Reset Status bit
1 = No Power-on Reset occurred
0 = A Power-on Reset occurred (must be set in software after a Power-on Reset occurs)

bit 0 **$\overline{BOR}$**: Brown-out Reset Status bit
1 = No Brown-out Reset occurred
0 = A Brown-out Reset occurred (must be set in software after a Brown-out Reset occurs)

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

copiado de PIC16F87XA.PDF pag.. 31/234

libros.ep-electropc.com

CONFIGURATION WORD

REGISTER 14-1: CONFIGURATION WORD (ADDRESS 2007h)(1)

R/P-1	U-0	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	R/P-1	U-0	U-0	R/P-1	R/P-1	R/P-1	R/P-1
CP	—	DEBUG	WRT1	WRT0	CPD	LVP	BOREN	—	—	$\overline{\text{PWRTEN}}$	WDTEN	Fosc1	Fosc0
bit 13													bit0

bit 13 **CP:** Flash Program Memory Code Protection bit
1 = Code protection off
0 = All program memory code-protected

bit 12 **Unimplemented:** Read as '1'

bit 11 **DEBUG:** In-Circuit Debugger Mode bit
1 = In-Circuit Debugger disabled, RB6 and RB7 are general purpose I/O pins
0 = In-Circuit Debugger enabled, RB6 and RB7 are dedicated to the debugger

bit 10-9 **WRT1:WRT0** Flash Program Memory Write Enable bits
For PIC16F876A/877A:
11 = Write protection off; all program memory may be written to by EECON control
10 = 0000h to 00FFh write-protected; 0100h to 1FFFh may be written to by EECON control
01 = 0000h to 07FFh write-protected; 0800h to 1FFFh may be written to by EECON control
00 = 0000h to 0FFFh write-protected; 1000h to 1FFFh may be written to by EECON control
For PIC16F873A/874A:
11 = Write protection off; all program memory may be written to by EECON control
10 = 0000h to 00FFh write-protected; 0100h to 0FFFh may be written to by EECON control
01 = 0000h to 03FFh write-protected; 0400h to 0FFFh may be written to by EECON control
00 = 0000h to 07FFh write-protected; 0800h to 0FFFh may be written to by EECON control

bit 8 **CPD:** Data EEPROM Memory Code Protection bit
1 = Data EEPROM code protection off
0 = Data EEPROM code-protected

bit 7 **LVP**: Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit
1 = RB3/PGM pin has PGM function; low-voltage programming enabled
0 = RB3 is digital I/O, HV on $\overline{\text{MCLR}}$ must be used for programming

bit 6 **BOREN**: Brown-out Reset Enable bit
1 = BOR enabled
0 = BOR disabled

bit 5-4 **Unimplemented:** Read as '1'

bit 3 **$\overline{\text{PWRTEN}}$**: Power-up Timer Enable bit
1 = PWRT disabled
0 = PWRT enabled

bit 2 **WDTEN**: Watchdog Timer Enable bit
1 = WDT enabled
0 = WDT disabled

bit 1-0 **Fosc1:Fosc0**: Oscillator Selection bits
11 = RC oscillator
10 = HS oscillator
01 = XT oscillator
00 = LP oscillator

Legend:		
R = Readable bit	P = Programmable bit	U = Unimplemented bit, read as '0'
- n = Value when device is unprogrammed		u = Unchanged from programmed state

Note 1: The erased (unprogrammed) value of the Configuration Word is 3FFFh.

copiado de PIC16F87XA.PDF pag..146

libros.ep-electropc.com

__CONFIG DIRECTIVE SYMBOLS

__CONFIG Directive Symbols (From Microchip Header Files)

Feature	SYMBOLS
Oscillators	_RC_OSC
	_EXTRC_OSC
	_EXTRC_OSC_CLKOUT
	_EXTRC_OSC_NOCLKOUT
	_INTRC_OSC
	_INTRC_OSC_CLKOUT
	_INTRC_OSC_NOCLKOUT
	_LP_OSC
	_XT_OSC
	_HS_OSC
Watch Dog Timer	_WDT_ON
	_WDT_OFF
Power-up Timer	_PWRTE_ON
	_PWRTE_OFF
Brown-out Reset	_BODEN_ON
	_BODEN_OFF
Master Clear Enable	_MCLRE_ON
	_MCLRE_OFF
Code Protect	_CP_ALL
	_CP_ON
	_CP_75
	_CP_50
	_CP_OFF
Code Protect Data EEPROM	_DP_ON
	_DP_OFF
Code Protect Calibration Space	_CPC_ON
	_CPC_OFF

Note 1: Not all configuration bit symbols may be available on any one device. Please refer to the MIcrochip include file of that device for available symbols.

copiado de 33023ª PICmicro Mid-R....pdf pag DF31027-A – pág e 27-6

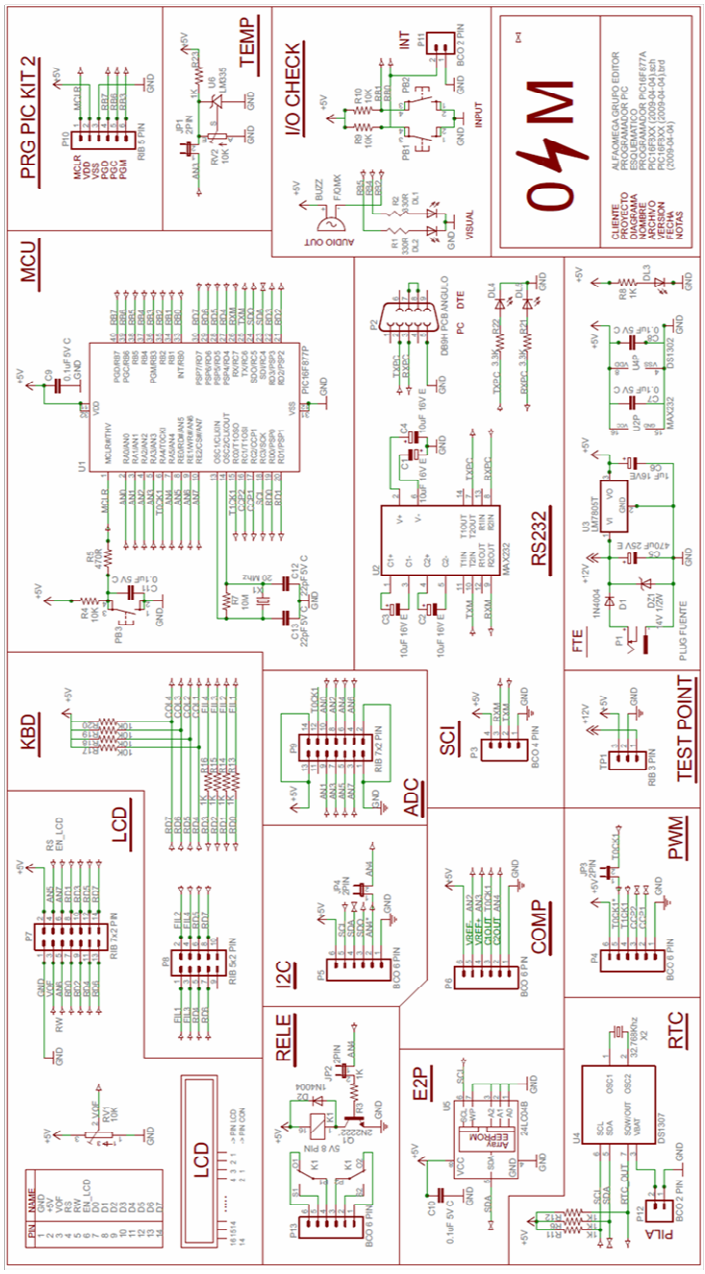
libros.ep-electropc.com

libros.ep-electropc.com

APÉNDICE F

PDF SISTEMA DE EVALUACIÓN PIC16F8XX

libros.ep-electropc.com



PIC16F8XX (2009-04-04) SCH.PDF (ESQUEMÁTICO)

libros.ep-electropc.com

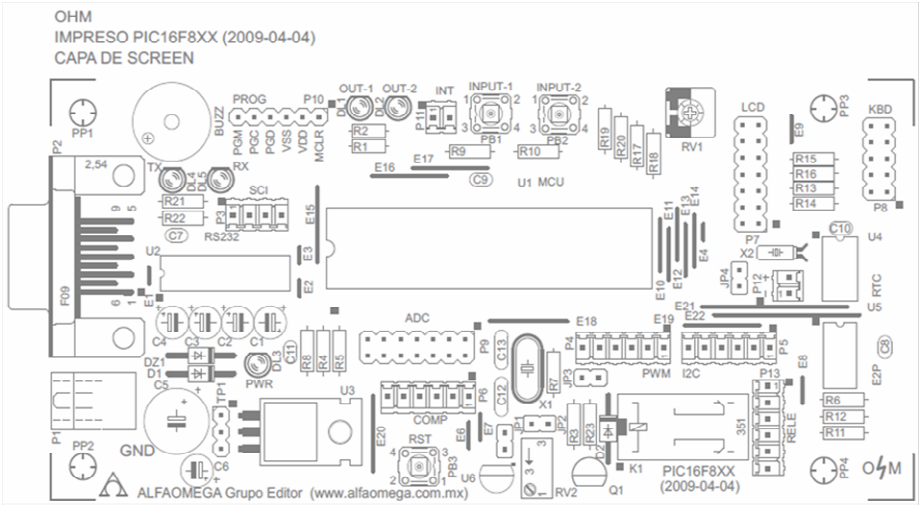


FIGURA F-2 PIC16F8XX (2009-04-04) SCREEN.pdf (capa de screen)

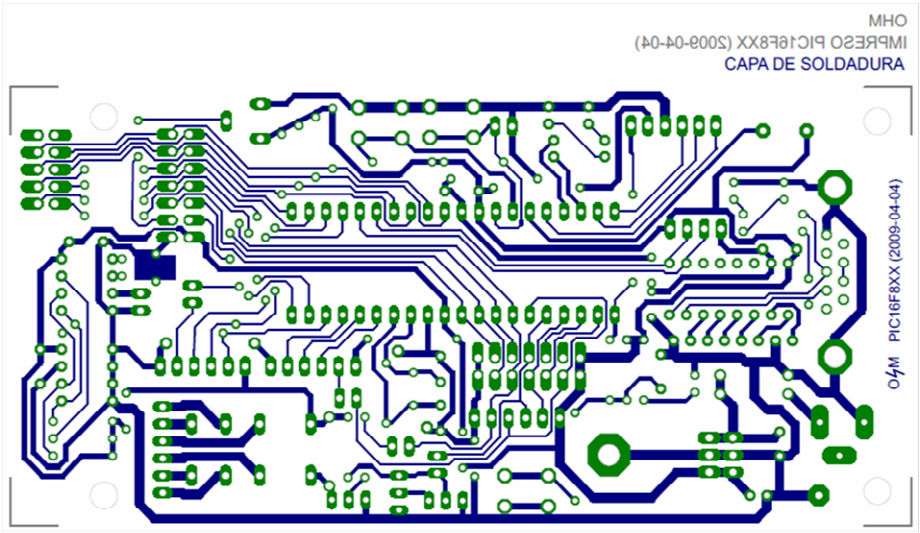


FIGURA F-3 PIC16F8XX (2009-04-04) SOLD -R.pdf (capa soldadura invertida)

libros.ep-electropc.com

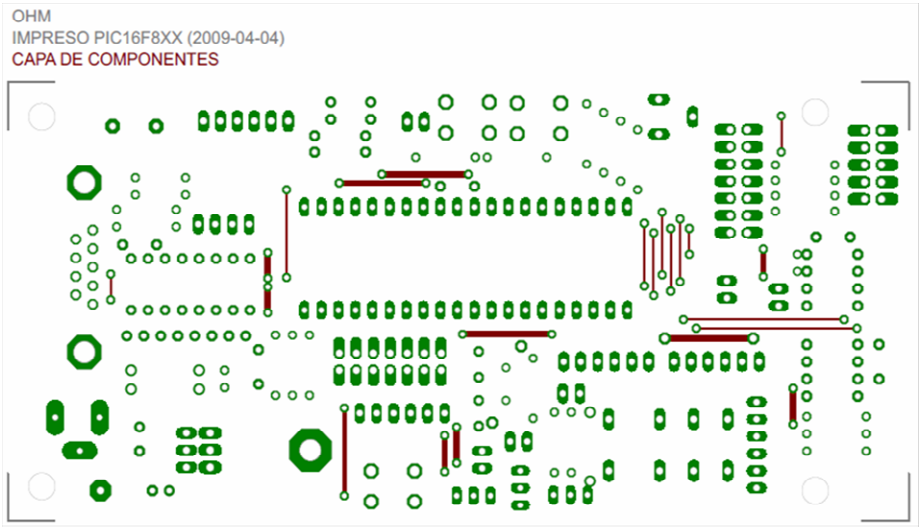
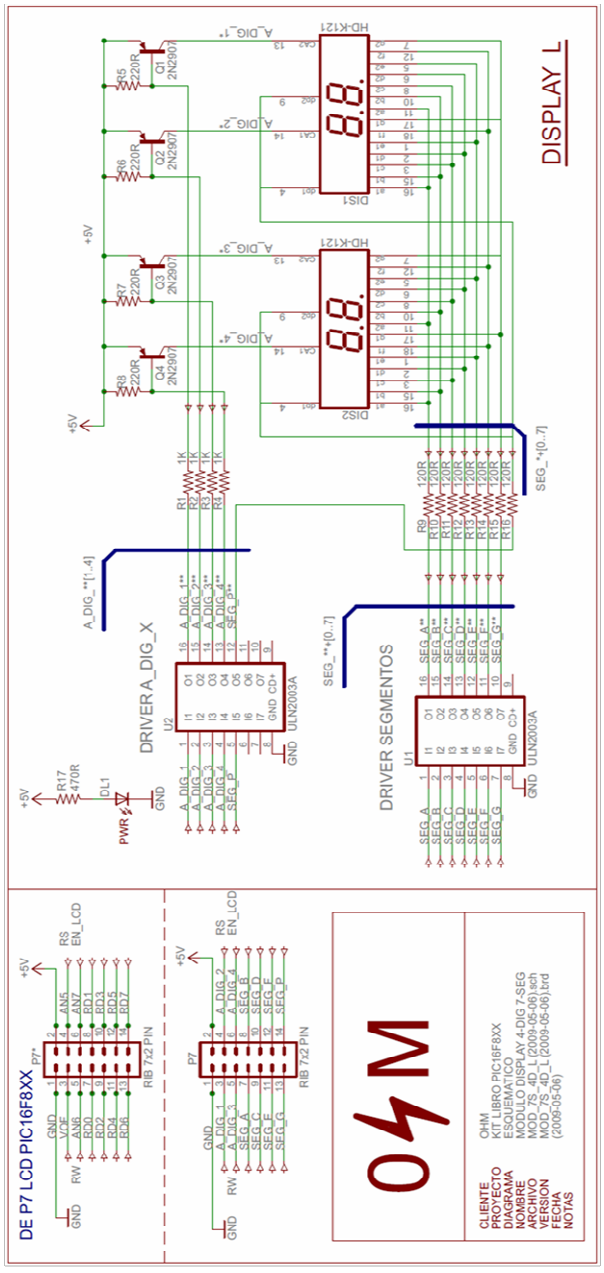


FIGURA F-4 PIC16F8XX (2009-04-04) COMP.pdf (capa de componentes)

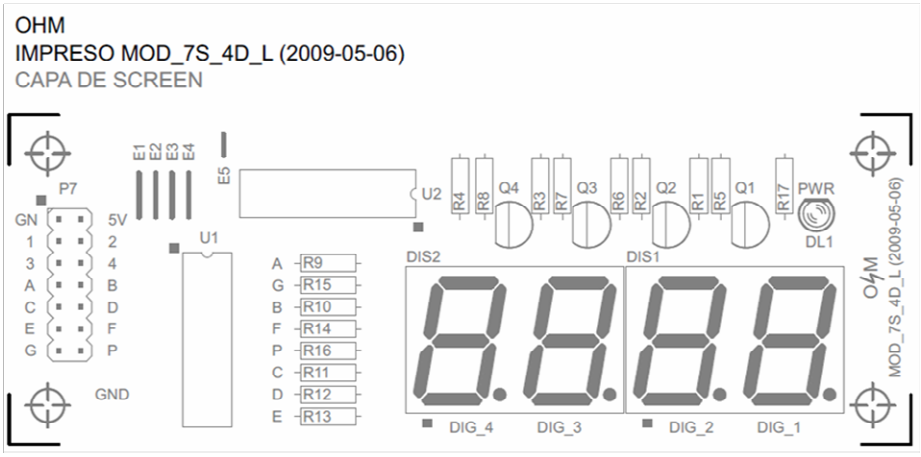
libros.ep-electropc.com

PDF DE MÓDULO DISPLAY 7 SEGMENTOS 4 DÍGITOS

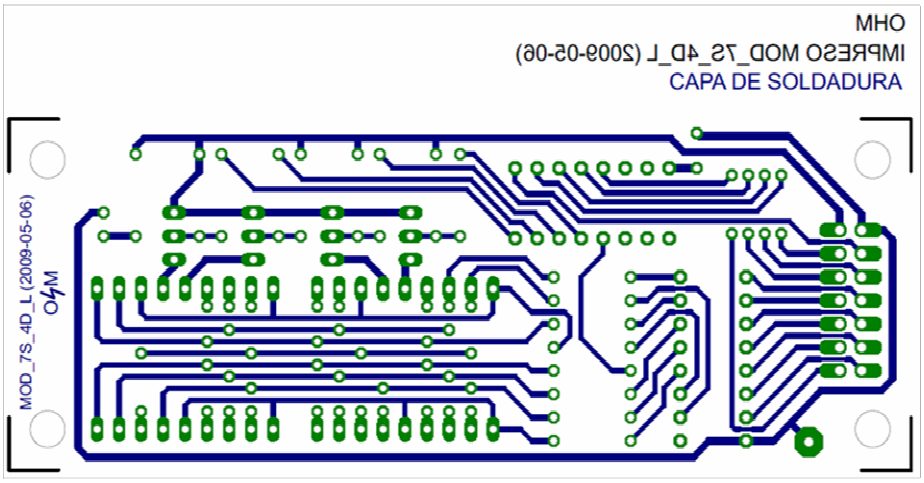


MOD_7S_4D_L (2009-05-06) SCH.pdf (esquemático)

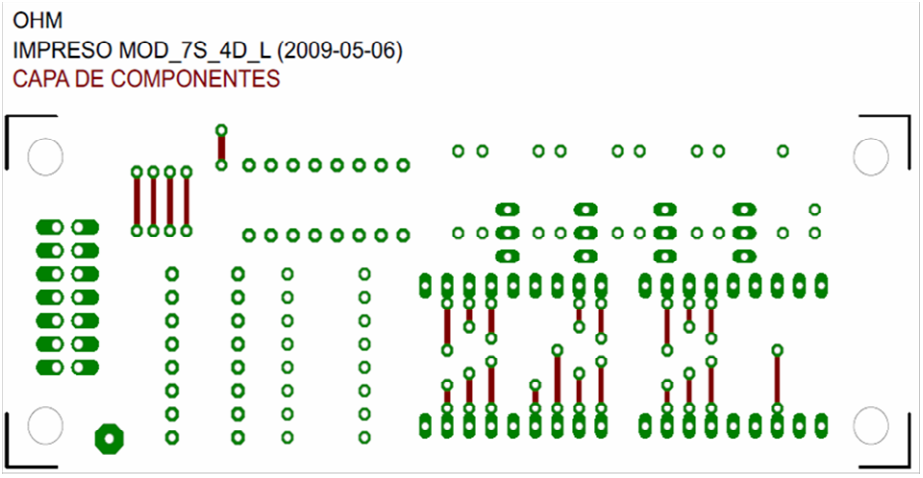
libros.ep-electropc.com



MOD_7S_4D_L (2009-05-06) SCREEN.pdf (capa de screen)

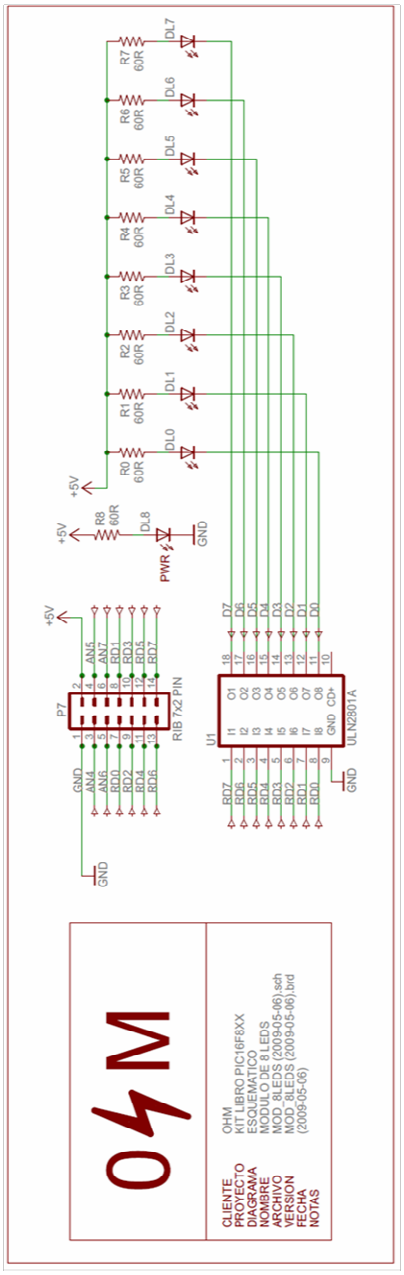


MOD_7S_4D_L (2009-05-06) SOLD -R.pdf (Capa Soldadura Invertida)

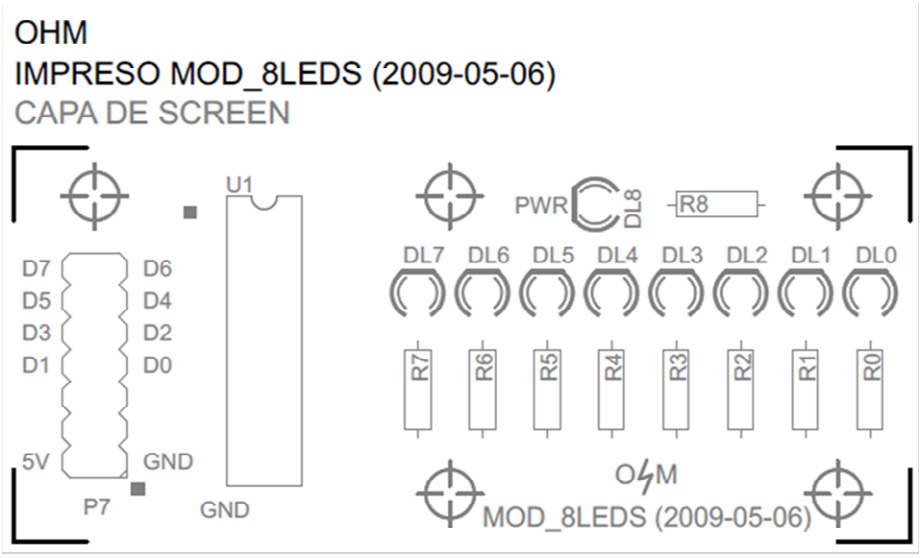


MOD_7S_4D_L (2009-05-06)) COMP.pdf (capa de componentes)

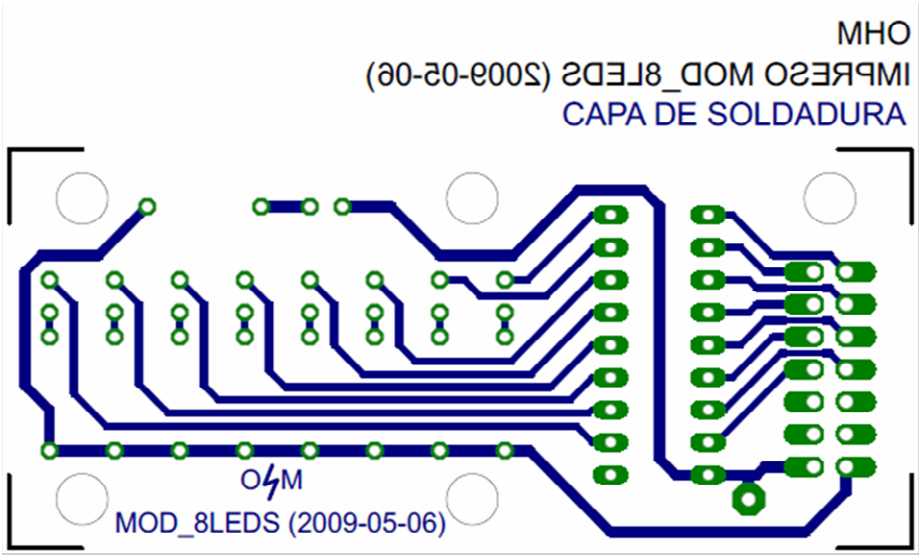
PDF DE MÓDULO DE 8 LEDS



MOD_8LEDS (2009-05-06) SCH.pdf (esquemático)

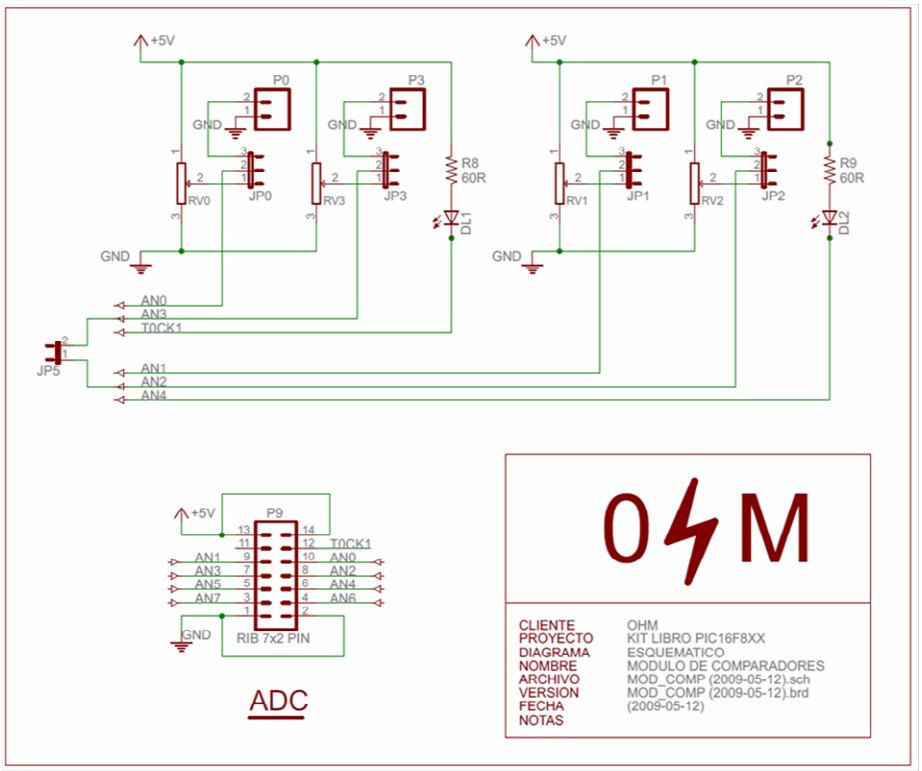


MOD_8LEDS (2009-05-06) SCREEN.pdf (capa de screen)

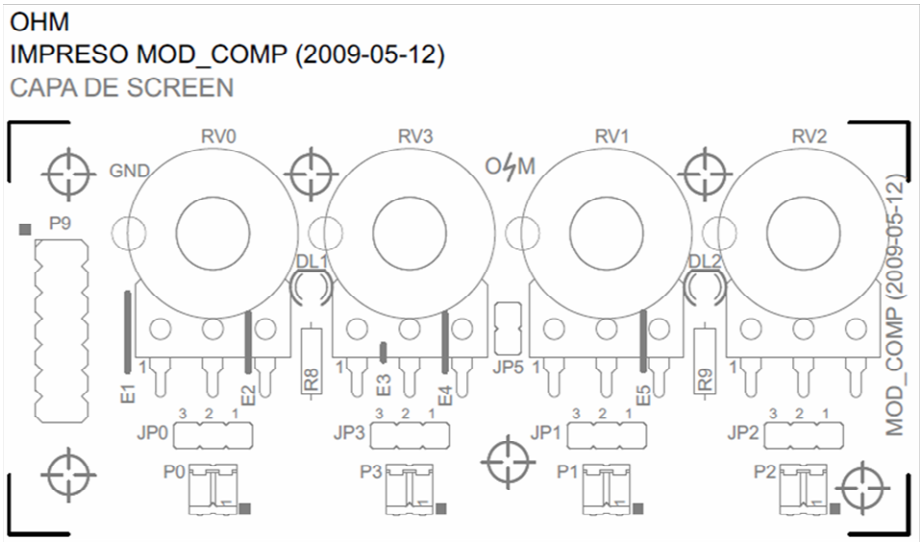


MOD_8LEDS (2009-05-06) SOLD -R.pdf (capa soldadura invertida)

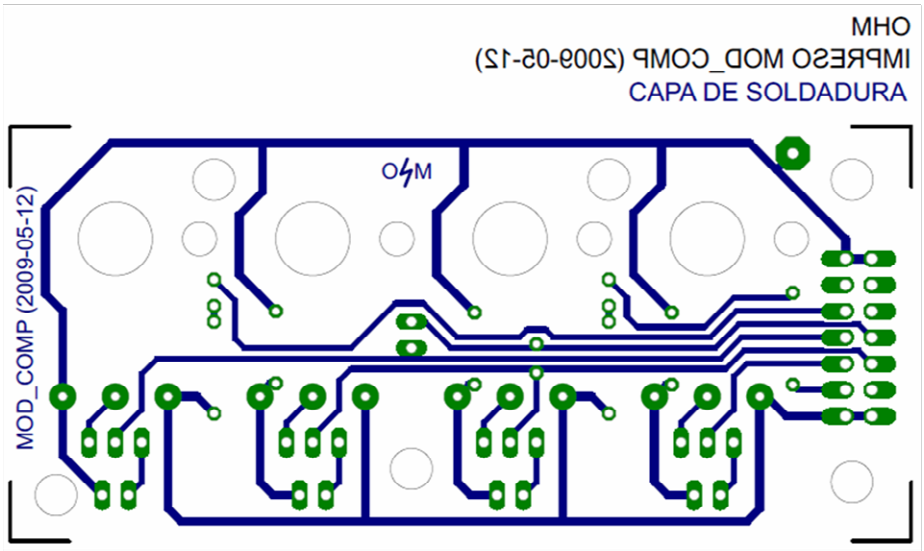
PDF DE MÓDULO DE 2 COMPARADORES



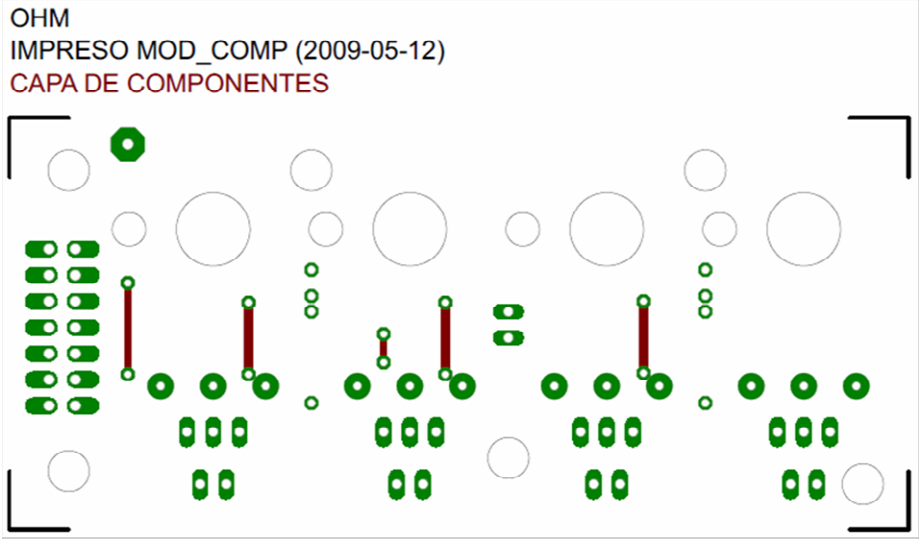
MOD_COMP (2009-05-12) SCH.pdf (esquemático)



MOD_COMP (2009-05-12) SCREEN.pdf (capa de screen)



MOD_COMP (2009-05-12) SOLD -R.pdf (capa soldadura invertida)



MOD_COMP (2009-05-12) COMP.pdf (capa de componentes)

APÉNDICE G

CÓDIGO ASCII

CÓDIGO ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

APÉNDICE H

DIRECCIONES EN INTERNET

Microchip Debugger Tool

Home Products Home Page Development Tools Main Page
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2519&param=en534451&page=wwwdevMPLABEmulatorDebuggers

Microchip Programer Tool

Home Products Home Page Development Tools Main Page
http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2807&devicefamilyid=1242&page=wwwDevToolsHomeProg

Microchip Soporte Tecnico 7/24

http://support.microchip.com/scripts/slxweb.dll/external?name=webticketcust

PicKit2 Clone

http://sergiols.blogspot.com/2009/02/pickit-2-clone-reloaded.html
http://sites.google.com/site/ronnyf7/pickit2
http://electronicaupao.blogspot.com/2010/01/pickit2-clone-full.html

Software de programaciòn Programadores WinPic 8000

http://www.winpic800.com/

Software de programaciòn Programadores IC-PROG

http://www.ic-prog.com/

Maxim RS232

http://www.maxim-ic.com/app-notes/index.mvp/id/882
http://www.datasheetcatalog.org/datasheet/maxim/MAX3222-MAX3241.pdf

Wikipedia SPI

http://es.wikipedia.org/wiki/Serial_Peripheral_Interface

Memoria EEPROM I2C 24LC32A

http://ww1.microchip.com/downloads/en/DeviceDoc/21713L.pdf

APÉNDICE I

CONTENIDO ADICIONAL

DATA SHEET
Contiene las hojas de datos de los MCU PIC16F877A, PIC16F887 y otros dispositivos usados en la tarjeta de evaluación PC16F8XX

EJEMPLOS_ASM
Contiene todos los ejemplos vistos en cada módulo y otros resueltos con sus respectivas carpetas y archivos de proyecto

HERRAMIENTAS
Contiene toda la información de las herramientas de hardware como PicKit2, sistema de evaluación PIC16F8XX y sus módulos

BIBLIOGRAFÍA

Datasheet del grupo de MCU PIC16F87XA

Microchip
PIC16F87XA Data Sheet
28/40/44-Pin Enhanced Flash Microcontrollers
© 2003 Microchip Technology Inc. DS39582B

Datasheet del grupo de MCU PIC16F882/883/884/886/887

Microchip
PIC16F882/883/884/886/887
Data Sheet
28/40/44-Pin, Enhanced Flash-Based 8-Bit
CMOS Microcontrollers with
nanoWatt Technology
© 2009 Microchip Technology Inc. DS41291F

Manual de referencia de Familia de MCU PICmicro™ Mid-Range

PICmicro™ Mid-Range MCU Family
Reference Manual
© 1997 Microchip Technology Inc. DS33023A

Broushure de MCU de 8 Bit

8-Bit PIC® Microcontrollers
Fall 2008
© 2008, Microchip Technology Incorporated. DS39630E

Memoria Datos y Programa

Section 6. Memory Organization
© 1997 Microchip Technology Inc. DS31006A

RAMROM.PDF
© 2001 Microchip Technology Incorporated. All Rights Reserved. S0001A
RAM/ROM(x14)

Memoria EEPROM

Section 7. Data EEPROM
© 1997 Microchip Technology Inc. DS31007A

Memoria Flash

EEPROM Memory Programming Specification PIC16F87X
© 2002 Microchip Technology Inc. DS39025F

FLASH Memory Programming Specification PIC16F87XA
© 2002 Microchip Technology Inc. DS39589B

ÍNDICE ANALÍTICO

A

Actualizar datos en la venta Program Memory 330
Acumulador 67
Adición o suma 128
Agregar archivos al proyecto 238
Agregar ventana Program Memory 328
Algoritmo 106
ALU 3, 4, 27, 40, 68
Analizador lógico 210,
AND 4, 133
Anidamiento de subrutinas 184, 194
Archivo *ASM 119
Archivo *HEX 121
Archivo *LST 120
Archivos *.LKR *.INC y MACROS 283, 291, 294
Archivos de definiciones (*.INC) 288, 289
Argumento 295
Aritméticas 4
Arquitectura RISC 96
Arquitectura 27, 34, 297
Arquitectura Harvard 28
Arquitectura Von Neumann 28
ASCII 212

B

Banderas (flags) 44, 127
Base (Socket) 9
Baudios 211
Bifurcación 107
Bit 96
Bits de configuración 91
Bloques de memoria de programas 43
BOR (Brown-Out Reset) 78
Breakpoint (Puntos de parada) 13, 17
BRG (Generador de tasa de baudios) 629, 634
Buses de datos 40
Busy 204
Byte 96

C

Call 69
Campos y sintaxis 114
Captura 518- 537
Características (Brochure) 35, 51
Cíclica 107
Ciclo de ejecución (execute) 30
Ciclo de instrucción 29
Ciclo de máquina 6, 29
Ciclo de traída (fetch) 30
Ciclos iterativos 160
Circuitería 29, 74, 87
Circuito de muestreo o Sampling Circuit 631
Circuito integrado 627
Circuitos RC 6, 75
CISC 31,
Código de identificación (ID locations) 93
Código de protección (Code protection) 93
Código 110
Códigos de máquina 111
Comentario 118
Como dato 176
Como dirección 176
Comparación 155, 537-550
Comparador 371, 440
Compilador 246
Compilar el proyecto o Build 242, 243, 269
Complemento e intercambio de nibbles 137
Componentes del MPLAB IDE 231
Conector ICSP 204
Conexión circuito integrado 627
Conexión USB 204
Configuración de bits 94, 303
Configuración de pines y conectores 216
Constantes 174, 293
Contador de programa 69
Conversores análogo a digital 371
CPU 31
Crear código 240
Crear proyecto 235

CS o SS 572,
CVRCON (Comparator Voltaje Reference Control) 459
CVRD (Comparator VREF Range Selection) 459
CVREN (Comparator Voltage Reference Enable) 459

D

Data Sheet (Hoja de datos) 22, 24, 47
Dato 175, 169
Datos EEPROM (Data EEPROM Memory) 298
Datos en tipos de memoria 169, 178
Datos SDI y SDO 571,
DCE (Data Communication Equipment) 622
Decisión 155
Delay 193
dePICs (DSC) 11,14, 20
Depuración 94, 263
Depurador (Debugger) 9, 232, 286
Depurar ejemplo 265
Descripción de partes 203
Descripción de pines (pinout) 36, 204
Despertar 197
Device ID 302
Device 235, 258
Diagrama circuital 297, 339, 373
Diagrama de flujo 107
Digikey 24
Dirección 175
Direccionamiento bit a bit 171
Direccionamiento directo 170, 367
Direccionamiento indexado 173
Direccionamiento indirecto 171, 368
Direccionamiento inmediato 169
Directivas en orden alfabético 166
Directivas por tipo 166
Directivas 163, 246
DTE (Data Terminal Equipment) 622

E

EPROM (Erasable Programmable Read Only Memory) 298
EEPROM Serial I2C 20
Elaborar un programa 99, 100
Enmascarables 197
Enlazador (Linker) 231
Ensamblador/Enlazador (Assembler/Linker) 231
Entorno 230
Entrada 3
Enviar programa al MCU 100
Especificaciones 214
Escribiendo datos en EEPROM 342
Erase 259
Estándar RS232 621,
EXOR 4
EEPROM 5, 10, 20, 67, 297
Emulador (Emulator) 9, 16, 232

F

FLASH (EEPROM/Electrically Erasable Programmable Read Only Memory) 299
FSR 31, 70, 71
Fuentes de interrupción 195
Funcionamiento 297, 339, 374
Funcionamiento de la subrutina 185
Formato de trama 625
Formatos de valores 183

G

Generador de voltaje de referencia (VREF) 372, 375, 440, 463, 467
GIE (Global Interrupt Enable) 82, 196, 197
GND 55
GPR (General Purpose Registers) 41, 44, 45, 47, 58, 60, 63, 180, 297, 298, 360, 365, 366

H

Habilitador de periféricos PEIE 81
Habilitador global GPIE 81
Hardware Programador/Depurador 101
Hardware Tarjeta de aplicación 101
Hardware 9
Herramienta UART (UART Tool) 210
Herramientas de compilación de lenguaje 232
Herramientas de lenguaje 231
Hex 212

I

I/O 3, 4, 7
IC-PROG 19
ICSP (In-Circuit Serial Programming) 94, 211, 263
ID Location 302
IDE (Integrated Development Environment) 9, 18, 229
Incrementar y disminuir 132
IND 31, 71
Instrucción 42
Instrucción RETFIE (Return from Interrupt) 83
Instrucciones 175, 198, 295
Instrucciones aritméticas 128
Instrucciones de bit 138,
Instrucciones de carga 127
Instrucciones de salto 138
Instrucciones de subrutinas e interrupciones 142
Instrucciones especiales 145
Instrucciones lógicas 133,
Interrupción 81, 477
Interrupción INT 84
Interrupción TMR0 84
Interrupciones 27, 44, 60, 183, 195
Interrupciones enmascarables 81
Interrupciones no enmascarables 81
Interruptor (switch) 55
IRP 71

L

Labcenter Electronics 19
Label 115, 295
Leds de estado 204
Leer y escribir en la memoria de programa 316
Leible/escribible 69, 72
Lenguaje ASM 19
Lenguaje Basic 19, 111
Lenguaje C 19, 111
Lenguaje de máquina 111
Lenguaje ensamblador 112
Lenguaje 111
Lenguajes gráficos 111
Leyendo datos en EEPROM 341
Límite de snack 193
Linker 283, 284, 285
Lista de componentes 222
Listado desensamblado (Disassembly Listing) 273
Llamado a subrutina 190
Lógicas 4

M

Macros 294, 295, 296
MainLoop 254
Manejador de proyectos 231
Manejo de bits 146
Manejo de datos 4
Mapa de memoria de programa 299
Mapa de memoria 5
Mapa de registros 360
Marcador Pin1 204
MatLab 19
MCLR 77
MCU Microchip 14,
MCU 3,4,5,6,7,9,10, 20, 22, 24, 25, 27, 34, 41, 297, 473
Memoria de configuración 299, 301
Memoria de datos del usuario 67
Memoria de datos 178, 298
Memoria de programa FLASH 40, 102
Memoria de programa 41, 178, 180, 273, 297, 298, 308
Memoria de sólo lectura 298
Memoria de usuario o programa 301, 308
Memoria de usuario 299, 300
Memoria EEPROM I2C 24LC32 613-619
Memoria EEPROM 51, 178, 179, 297, 306, 339
Memoria 3, 5, 33, 297
Memorias 27
Microchip 13, 16, 18, 19, 20, 27, 28, 31, 95, 112
Microcontrolador 3, 28
Microcontroladores PIC 31
Microcontroladores 31, 67
Microprocesador 27, 28
Microwire EEPROM Serial 93XX 20
Mnemónicos 112, 113,
Modo analizador lógico (Logic analyzer mode) 209

Modo asíncrono o asincrónico 634
Modo comunicación Inter-Integrados (I2C) 589-593
Modo de transmision 647
Modo interfaz periférica serial (SPI) 568
Modo lógico I/O (Logic I/O Mode) 208
Modo run 282
Modos de operación 205
Módulo ADC (Conversor Análogo a Digital (A/D) 417, 440
Módulo comparador 441-458
Módulo general de voltaje de referencia 458 -471
Módulo interfaz de comunicación serial (SCI USART) 621
Módulo MSSP en modo I2C 593-612
Módulo MSSP en modo SPI 572
Módulo Puerto Serial Sincrónico Maestro (MSSP) 567, 572
Módulo SCI 628
Módulos periféricos 28
Módulos 227
Motor de ejecución (Execution Engine) 232
Mouser 24
MPASM Toolsuite 237, 238, 289
MPLAB IDE 232, 256
MPLAB SIM 245, 246
MPLAB 11, 91, 94, 101, 102, 230, 233

N

Niveles lógicos 625
No enmascarables 197
No leíble/sí escribible 69
Notas de aplicación (aplication notes) 22
NRZ (Non-Return-to-Zero) 634

O

Opcode 96, 112, 113, 117
Operaciones aritméticas 149
Operandos 117
OR 4, 134
OREX 134
Ortogonalidad 31
Oscilador 27, 28
OST (Oscillator Start-Up Timer) 78, 79
OTP (One-Time-Programmable) 298

P

Package (Encapsulado) 24
Páginas 42, 43, 310
Palabra (word) 42
Paralelo 18
Paso interno 275, 276, 278, 279, 280, 281, 282
Pasos 187
PC-LATH 43
PCON (Power Control Status Register) 79
PDIP (Plastic Dual in Package) 36, 37, 53, 54
PEIE 196, 197
Periféricos 27, 28, 38, 40, 54

PI0 28
PIC 16 33
PIC Lite Compiler 11
PIC 11, 41
PIC10/12/16 32, 112, 123
PIC16F84 11
PIC16F877A 27, 32, 35, 38, 111, 235, 256, 258
PIC16F877A-887 27, 32, 34, 35, 289, 290, 362
PIC16F87XA 27, 32, 33, 34, 38, 76, 81
PIC16F887 27, 32, 51, 54, 55, 285
PIC16F88X 27, 32, 33, 34, 54, 101, 213, 214, 218, 225, 227
PICKit 2 Programmer 256, 257, 258, 270, 271, 302
PICKIT 3 Debug Express 15
Pines-puertos y periféricos 36, 53
Planos 218
PLCC 36
POR (Power-on Reset) 78
Power 204
Prescaler 86, 476
Procedimiento de ensamble 224
Procedimiento de uso de memoria EEPROM 344,
Procedimientos de lectura y escritura 319, 321, 322, 323, 324
Proceso de diseño 230
Proceso de una interrupción 83, 199
Proceso segmentado 30
Procesos especiales de programa 183
Programa 111, 297
Programación 19, 20, 99, 255
Programador (Programmer) 9, 205, 232
Programador Stand-Alone (Programmer-To-Go) 212
Programador/Depurador 102, 203
Programar ejemplo 234
Programar microcontrolador 261
Protección CP y WRT 316
Protección CPD 341
Proteus VSM 19
Protoboard 37, 54
Proyecto.mcw 268, 269
Pruebas de funcionamiento 225
PSP (Puerto esclavo paralelo) 371, 413, 414, 415, 416
Puerto A 373, 374,375,376,377,378,379,380
Puerto B 381, 382, 383, 384, 385, 386, 387, 388, 389, 390
Puerto C 390, 391, 392, 393, 394, 395,
Puerto D 396, 397, 398, 399, 400, 401, 402, 403
Puerto E 404, 405, 406, 407, 408, 409, 410, 411, 412,413
Puertos I/O (de entrada-salida) 371, 372
Puertos 38, 40, 54
Pulsador 204
Pulsador 213
PWM 4, 473
PWN (Pulse Width Modulator) 550- 564
PWRT (Power-Up Timer) 78

Q

QFN 36, 53

R

RA1 4
RA2 4
RAM (Random Access Memory) 5, 28, 31, 40, 297, 360
RB5 4
RB7 4
Receptor asíncrono 637
Registro W 67, 113
Registros asociados 297, 343, 375
Registros de configuración 301
Registros en RAM 367
Registros usados 317
Reset 8, 27
Resumen del proyecto 239
Return 186, 187, 193, 276
RISC 31
ROM (Read Only Memory) 5, 28, 298
Rotaciones 135
Rs (resistencias) 95
RS-232 11, 256
RSR (Receive Shift Register) 637
Run to Cursor 276, 277
Ruta del proyecto 237
Rutinas 293

S

Salida 3,
SCI (Serial Communication Interface) 4
Sección GPR 288
Sección PROG 287
Selección de dispositivo 235
Selección de lenguaje 236
Sentencia 107
Set de instrucciones 111, 112, 114, 123, 308
SFR (Special Function Register) 41, 44, 45, 47, 58, 60, 63, 297, 298, 360, 363, 364
Simulación 244, 245
SISC 31
Sistema de Evaluación 213
Sleep 87, 89, 90
Snack 193, 194
Software 9, 18,
SOIC 36, 53
SPI (Serial Peripheral Interface) 4
SSOP 36, 53
Status 60, 68, 246, 247, 248, 249, 250, 253, 254
Subrutinas 183, 184, 191
Substracción o resta 130
Sujetador 204

T

Tablas 180
Tarjeta o board de aplicación (target) 9, 102, 204, 218, 222
Temporizadores (Timers) 473

Timer0 474-496
Timer1 496- 508
Timer2 509-517
Tipo de instrucción 311, 312, 313, 314, 315
Tipo de operación 125
Tipo de operando 123
Tipos de archivos 119
Tipos de datos 169, 174
Tipos de direccionamiento 169
Tipos de estructuras 107
Toma de decisiones 4
TQFP 36, 53
TSR (Transmit Shift Register) 634

U

UART 211
Ubicación 308
USART 28
USB 11

V

Variables 174, 292
Verificar funcionamiento de microcontrolador 262

W

Watch 7, 248, 249, 250, 253, 271, 272
WinPIC800 19
WDT (Watchdog Timer) 8, 10, 27, 72, 85, 90, 91