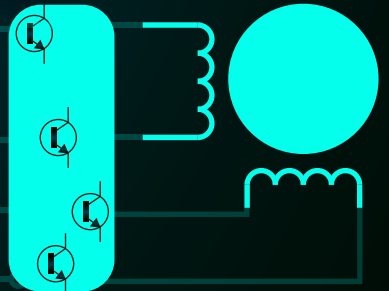
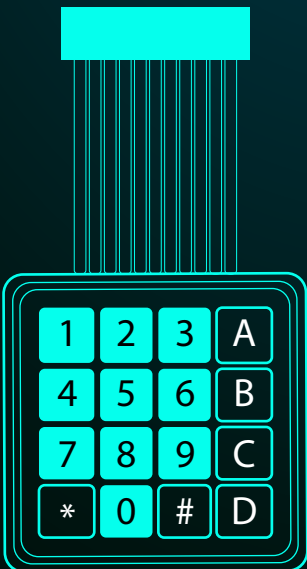


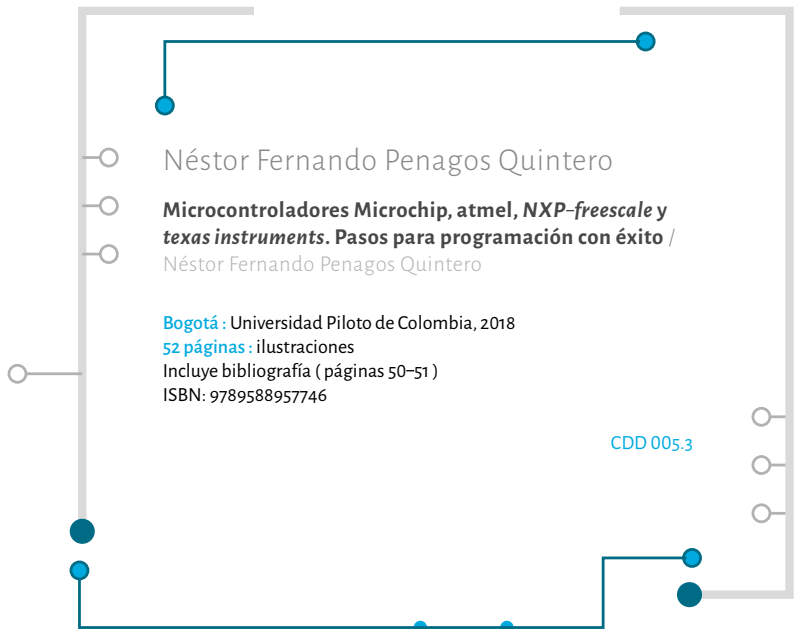
Microcontroladores
Microchip, Atmel,
NXP-Freescale y
Texas Instruments.
*Pasos para una
programación con éxito*

**Néstor Fernando
Penagos Quintero**



The image features a central white rectangular box with a blue border, containing text. The background is a light gray grid of dots, with a network of gray lines and nodes extending from the central box. The text is in a dark blue, sans-serif font. The overall design is clean and technical.

**Microcontroladores
Microchip, Atmel,
NXP-Freescale y
Texas Instruments.
Pasos para una
programación
con éxito**





Microcontroladores Microchip, Atmel, NXP-Freescale y Texas Instruments. Pasos para una programación con éxito

ISBN

XXXXXXXXXX

Primera edición – 2019

Bogotá, Colombia

Autor

Néstor Fernando Penagos Quintero

Diseño

Ivonne Carolina Cardozo Pachón

Departamento de Publicaciones y Comunicación Gráfica

UNIVERSIDAD PILOTO DE COLOMBIA

Presidente

José María Cifuentes Páez

Rectora

Ángela Gabriela Bernal Medina

Director de Publicaciones y Comunicación Gráfica

Rodrigo Lobo-Guerrero Sarmiento

Director de Investigaciones

Mauricio Hernández Tascón

Coordinador General de Publicaciones

Diego Ramírez Bernal

Decano del programa de Ingeniería Mecatrónica

Jaime Durán García

CONTENIDO

Introducción	14
Capítulo I	16
Sistemas de Numeración	17
1.1 Números decimales	18
1.2 Números Binarios	19
1.3 Números Hexadecimales	22
Ejercicios propuestos	26
Capítulo II	27
Memorias y Microprocesadores	28
2.1 Registro	28
2.2 Memorias	31
2.2.1 Escritura	32
2.2.2 Lectura	33
2.2.3 Bus de datos	33
2.2.4 Bus de direcciones	33
2.2.5 Bus de control	34
2.2.6 Familias de la RAM	34
2.2.7 Familias de la ROM	36
2.3 Microprocesador	37
2.3.1 Instrucción	39
2.4 Dispositivos Periféricos	40
2.5 Sistemas Micro-Procesados	40
2.5.1 Desventajas	41
Capítulo III	42
Microcontroladores	43
3.1 Ventajas	43
3.2 Arquitectura de los Microcontroladores	44
3.3 PIC16F628A	47
3.3.1 Registro STATUS	49
3.3.2 Instrucciones	50
3.4. Diagramas de Flujo	82
3.5. Puertos de entrada/salida	89
3.6. Paso de diseño y Programación	91
3.7. Manejo de display 7 segmentos	105
3.7.1 Visualización dinámica con display de 7 segmentos	110
3.7.2. Rotación de datos con display siete segmentos	115
3.8. Manejo de matriz 7x5 unicolor	122
Ejercicios propuestos	136

Capítulo IV	138
Otras aplicaciones	139
4.1. Manejo del teclado matricial y display 7 segmentos	139
4.2. Creación y manejo de librerías	148
4.2.1. Paso para la construcción de librerías en MPLAB	148
4.3. Manejo de LCD	158
4.4. Manejo de la Memoria EEPROM Data	193
4.5. Manejo de motor paso a paso	214
Ejercicio propuestos	227
Capítulo V	228
Otros Microcontroladores	229
5.1. Microcontroladores de ATMEL. Atender	229
5.1.1. ATtiny2313	230
5.1.1.1 Registro STATUS	231
5.1.1.2 Puertos de I/O	234
5.1.1.3. Set de instrucciones	234
5.1.2. Programación	239
5.2. Microcontroladores de NXP-FREESCALE	249
5.2.1. Microcontroladores de 8 BITS	250
5.2.2. MC9S08JS16	251
5.2.3. PINES DE I/O	254
5.2.4. Set de instrucciones	254
5.2.5. Programación	260
5.3. Microcontroladores Texas Instruments	
MSP430	278
5.3.1. MSP430F2272	280
5.3.2. PINES DE I/O	283
5.3.3. Set de instrucciones	284
5.3.4. Programación	285
Ejercicios propuestos	292
Referencias Bibliográficas	294
Índice de figuras	6
Índice de tablas	9
Índice de diagramas	11

ÍNDICE DE FIGURAS

Figura 1.	Ejemplos de registros de 4 bits y 8 bits	29
Figura 2.	Movimiento básico de los registros	30
Figura 3.	Diferentes tamaños de matrices	31
Figura 4.	Tamaño de una Memoria	32
Figura 5.	Proceso de escritura y de lectura de una memoria	33
Figura 6.	Familia de memorias RAM	35
Figura 7.	Diagrama de bloques donde se observa memoria caché L1 y L2	36
Figura 8.	Familia de memorias ROM	36
Figura 9.	Esquema de un microprocesador	38
Figura 10.	Diagrama de bloques de un sistema microprocesador	41
Figura 11.	Memoria RAM del PIC 16F628A	48
Figura 12.	Memoria ROM del PIC 16F628A	49
Figura 13.	Ilustración de la instrucción <i>DECFSZ</i> <i>f,d</i>	59
Figura 14.	Ilustración de la instrucción <i>INCFSSZ</i> <i>f,d</i>	61
Figura 15.	Ilustración de la instrucción <i>RLF</i> <i>f,d</i>	65
Figura 16.	Ilustración de la instrucción <i>RRF</i> <i>f,d</i>	66
Figura 17.	Ilustración de la instrucción <i>SWAPF</i> <i>f,d</i>	68
Figura 18.	Ilustración de la instrucción <i>BTFSZ</i> <i>f,b</i>	73
Figura 19.	Ilustración de la instrucción <i>BTFS</i> <i>f,b</i>	73
Figura 20.	Ilustración de la instrucción <i>CALL</i> <i>k</i>	76
Figura 21.	Generar nuevo proyecto en Mplab X 3.65 paso 1	84
Figura 22.	Generar nuevo proyecto en Mplab X 3.65 paso 2	84
Figura 23.	Generar nuevo proyecto en Mplab X 3.65 paso 3	85
Figura 24.	Generar nuevo proyecto en Mplab X 3.65 paso 4	85
Figura 25.	Generar nuevo proyecto en Mplab X 3.65 paso 5	86
Figura 26.	Generar nuevo proyecto en Mplab X 3.65 paso 6	87
Figura 27.	Simulación en Mplab X 3.65 paso 1	88
Figura 28.	Simulación en Mplab X 3.65 paso 2	88
Figura 29.	Simulación en Mplab X 3.65 paso 3	89
Figura 30.	Distribución de pines del PIC 16F628A	90

Figura 31.	Diagrama de conexiones del ejemplo 8. (Proteus, 2003)	92
Figura 32.	Cambio de banco y programar puertos	96
Figura 33.	Cambio al banco 0 y prender el led	97
Figura 34.	Llamado de la Subrutina de TIEMPO	97
Figura 35.	Apagado del led	98
Figura 36.	Diagrama de conexiones del ejemplo diez. (Proteus, 2003)	102
Figura 37.	Diagrama de conexiones del ejemplo 11. (Proteus, 2003)	106
Figura 38.	Distribución de un display 7 segmentos	106
Figura 39.	Diagrama de conexiones del ejemplo 12. (Proteus, 2003)	111
Figura 40.	Construcción de Matriz 7x5	122
Figura 41.	Diagrama de conexiones con una Matriz 7x5 cátodo común. (Proteus, 2003)	123
Figura 42.	Esquema de teclado numérico matricial 4x4	140
Figura 43.	Diagrama de conexiones del ejemplo 15	141
Figura 44.	Diagrama de conexiones del ejemplo 17. (Proteus, 2003)	162
Figura 45.	Diagrama de conexiones del ejemplo 18. (Proteus, 2003)	173
Figura 46.	Diagrama de grados para el motor paso a paso del ejemplo 20	214
Figura 47.	Esquema de un motor paso a paso de cuatro bobinas, unipolar	215
Figura 48.	Diagrama de conexiones del ejemplo 20. (Proteus, 2003)	217
Figura 49.	Distribución de pines del ATtiny2313	231
Figura 50.	Mapa de memoria de programa del ATtiny2313	231
Figura 51.	Memoria SRAM del ATtiny2313/V	233
Figura 52.	Registros de propósito general y de trabajo del ATtiny2313/V	233
Figura 53.	Crear nuevo proyecto en AtmelStudio 7.0 Paso 1	241
Figura 54.	Crear nuevo proyecto en AtmelStudio 7.0 Paso 2	241
Figura 55.	Crear nuevo proyecto en AtmelStudio 7.0 Paso 3	243

Figura 56. Distribución de pines del MC9S08JS16.	251
Figura 57. Distribución de Memoria del MC9S08JS16. NXP-Freescale	252
Figura 58. Crear proyecto en CodeWarrior 10.1 Paso 1	261
Figura 59. Crear proyecto en CodeWarrior 10.1 Paso 2	262
Figura 60. Crear proyecto en CodeWarrior 10.1 Paso 1	262
Figura 61. Crear proyecto en CodeWarrior 10.1 Paso 4	263
Figura 62. Distribución de pines del MSP430F2272.	280
Figura 63. Distribución de memoria del MSP430F2272.	281
Figura 64. Parte de la CPU del MSP430F2272.	282
Figura 65. Crear proyecto en CodeComposer 6.2 Paso 1	287
Figura 66. Crear proyecto en CodeComposer 6.2 Paso 2	288
Figura 67. Crear proyecto en CodeComposer 6.2 Paso 3	291

Tabla 1.	Los primeros diez números decimales	18
Tabla 2.	Combinaciones con dos dígitos de números decimales	18
Tabla 3.	Primeros números binarios	19
Tabla 4.	Cuatro primeros números binarios	20
Tabla 5.	Se observan los ocho primeros números binarios	20
Tabla 6.	Primeros quince números decimales y su equivalente en binario	21
Tabla 7.	Primeros números hexadecimales con sus equivalentes decimales	22
Tabla 8.	Algunos números hexadecimales con sus equivalentes decimales	23
Tabla 9.	Se observan las primeras 16 combinaciones de la numeración hexadecimal y su equivalente en el sistema decimal y binario	24
Tabla 10.	Comparación de algunos Microcontroladores gama media (Microchip, 2006).	46
Tabla 11.	Instrucciones del PIC16F628A	51
Tabla 12.	Forma de respuesta de la instrucción <i>SUBWF f,d</i>	67
Tabla 13.	Ilustración de la instrucción <i>MOVLW k</i>	78
Tabla 14.	Forma de respuesta de la instrucción <i>SUBLW k</i>	80
Tabla 15.	Códigos para números con display ánodo común	107
Tabla 16.	Códigos para la palabra HOLA del ejemplo 12	112
Tabla 17.	Códigos para la letra l con Matriz 7x5 cátodo común	124
Tabla 18.	Subrutina de la librería <i>libreria_Matriz_7x5</i>	151
Tabla 19.	Distribución de pines LCD 2x16	158
Tabla 20.	Códigos ASCII más usados	160
Tabla 21.	Modos de una LCD 2x16.	160
Tabla 22.	Subrutinas importantes en la librería LCD4bits	172

ÍNDICE DE TABLAS

Tabla 23. Orden de energizar las bobinas de un motor paso a paso de cuatro bobinas, unipolar	216
Tabla 24. Grados por girar	216
Tabla 25. Comparación entre algunos Microcontroladores de 8 Bit´s. Atmel	229
Tabla 26. Comparación entre los Microcontroladores PIC 16F628A y el ATtiny2313	230
Tabla 27. Set instrucciones del ATtiny2313	234
Tabla 28. Familias de Microcontroladores de Freescale	249
Tabla 29. Comparación entre algunos Microcontroladores de 8 Bit´s. Freescale	250
Tabla 30. Comparación entre los Microcontroladores PIC 16F628A, el ATtiny2313 y el MC9S08JS16	251
Tabla 31. Set instrucciones del MC9S08JS16	254
Tabla 32. Forma de uso de la instrucción MOV opr,opr	265
Tabla 33. Comparación entre algunos Microcontroladores de 16 Bit´s. Texas	279
Tabla 34. Comparación entre los Microcontroladores PIC16F62A, ATtiny2313, MC9S08JS16 y el MSP430F2272	279
Tabla 35. Set instrucciones del MSP430F2272	284
Tabla 36. Forma de uso de la instrucción MOV src,dst	289

Diagrama 1.	Programa que suma dos registros 8 bits	83
Diagrama 2.	Programa que prende y apaga un led	93
Diagrama 3.	Tiempo del ejemplo 8	94
Diagrama 4.	Programa juego de luces	100
Diagrama 5.	Programa que prende y apaga un led con dos velocidades	103
Diagrama 6.	Programa para visualizar los numeros decimales en display 7 segmentos	108
Diagrama 7.	Programa para visualizar la palabra hola en 4 display de 7 segmentos	113
Diagrama 8.	Programa para visualizar una frase completa en cuatro displays de siete segmentos.	117
Diagrama 9.	Subrutina ROTAR del ejemplo 13	118
Diagrama 10.	Subrutina VER del ejemplo 13	119
Diagrama 11.	Programa principal para visualizar una palabra completa en una matriz 7x5	125
Diagrama 12.	Subrutina de la letra l	126
Diagrama 13.	Subrutina ROTAR para una matriz 7x5 cátodo común	127
Diagrama 14.	Subrutina VER para una matriz 7x5 cátodo común	128
Diagrama 15.	Programa para manejo de un teclado matricial y visualizar en display siete segmentos	142
Diagrama 16.	Subrutina para el número uno del ejemplo 15	143
Diagrama 17.	Programa principal para uso de librerías para manejo de una matriz 7x5	153
Diagrama 18.	Subrutina de la letra l	154
Diagrama 19.	Programa manejo LCD a 8 bits. Ejemplo 17	164
Diagrama 20.	Subrutina PRENDE_LCD. Ejemplo 17	165
Diagrama 21.	Subrutina LÍNEA 1. Ejemplo 17	165
Diagrama 22.	Subrutina LÍNEA 2. Ejemplo 17	166
Diagrama 23.	Subrutina DATO. Ejemplo 17	166

ÍNDICE DE DIAGRAMAS

Diagrama 24.	Subrutina CLAVE para leer en la EEPROM Data del ejemplo 19	197
Diagrama 25.	Subrutina LEER_EE del ejemplo 19	198
Diagrama 26.	Subrutina CAMBIO para monitorear el teclado del ejemplo 19	198
Diagrama 27.	Subrutina ESCRIBIR_1 para control de la EEPROM Data del ejemplo 19	199
Diagrama 28.	Subrutina ESCRIBIR_EE para escribir en la EEPROM Data del ejemplo 19	200
Diagrama 29.	Programa principal del ejemplo 20	218
Diagrama 30.	Subrutina G_90 del ejemplo 20	218
Diagrama 31.	Subrutina EVALUAR del ejemplo 20	219
Diagrama 32.	Subrutina GIRE_D del ejemplo 20	220
Diagrama 33.	Subrutina GIRE_I del ejemplo 20	221
Diagrama 34.	Programa que prende y apaga un led. Ejemplo 21	240
Diagrama 35.	Programa que prende y apaga un led con dos velocidades. Ejemplo 22	244
Diagrama 36.	Programa para visualizar los números decimales del 0 al 9 en un Display 7 Segmentos. Ejemplo 23	247
Diagrama 37.	Programa que prende y apaga un led. Ejemplo 24	261
Diagrama 38.	Programa juego de luces. Ejemplo 25	269
Diagrama 39.	Programa que prende y apaga un led con dos velocidades. Ejemplo 26	272
Diagrama 40.	Programa para visualizar los números decimales del 0 al 9 en un Display siete Segmentos. Ejemplo 27	276
Diagrama 41.	Programa que prende y apaga un led. Ejemplo 28	286
Diagrama 42.	Tiempo del ejemplo 28	286



Dedicatoria

A mis padres Alba y Marco por su apoyo económico.

A mi esposa Nanda quien con su apoyo espiritual
fue fundamental para la culminación de este libro.

A mi hijo Tomás Fernando quien me hace seguir adelante.



Introducción

El desarrollo y la evolución de los microcontroladores han hecho posible que sus aplicaciones sean cada vez más robustas. Estas aplicaciones se pueden conseguir utilizando microcontroladores con procesadores de 8, 16 y hasta 32 bits con memoria de programa tipo flash de 64K, 128K, 256K Bytes; con memorias RAM de 8K bytes; con más de 60 puertos, entre otras características. Esto hace que se puedan desarrollar programas grandes que antes no se podían realizar en un dispositivo de estos. Por esta razón, se hace necesario conocer y estar a la vanguardia de estas tecnologías para el desarrollo de nuevos proyectos de robótica móvil, sistemas mecatrónicos y de instrumentación, entre otros. Este texto es una introducción a la programación de microcontroladores en lenguaje ensamblador que le brinda al lector los conceptos básicos y algunos ejemplos para profundizar en el tema y desarrollar proyectos propios de mayor complejidad.

En este libro proponemos seis pasos para la programación exitosa de los microcontroladores tanto de Microchip, Atmel, NXP-Freescale y Texas Instruments¹. Como ya mencionamos, la propuesta es una introducción a la programación de aplicaciones bajo microcontroladores de 8 bits.

En el primer capítulo se hace una breve reseña sobre los sistemas de numeración decimal, hexadecimal y binario; y se señalan algunos ejemplos de conversión entre ellos. La clarificación de estos ejemplos es relevante para el buen desarrollo de

¹ Paso 1, saber cómo funciona lo que se quiere controlar; Paso 2, hacer un diagrama de conexiones de la distribución del circuito; Paso 3, hacer el diagrama de flujo; Paso 4, pasar a las instrucciones el diagrama de flujo; Paso 5, programar el microcontrolador; Paso 6, si se requiere, realice las respectivas correcciones.

los procesos de simulación. En el segundo capítulo se describe qué es una memoria y cuáles son los elementos más importantes que la componen y que permiten comprender el proceso de lectura y escritura: registros, bus de datos, bus de control, bus de direcciones, entre otros. Igualmente, se describen las partes y el funcionamiento de un microprocesador. Por último, se analizan los periféricos de Entrada/Salida (*IN/OUT*) y se describe el funcionamiento de un sistema microprocesado. Se recomienda que el lector tenga claro las tablas de verdad de las compuertas lógicas (AND, OR, NOT, XOR) de esa forma puede comprender mejor el funcionamiento del sistema microprocesado.

En los capítulos tres y cuatro se presenta una introducción a los microcontroladores fabricados por Microchip, seleccionado el PIC16F628A para desarrollar diferentes aplicaciones. Así mismo, se explica el proceso para realizar un proyecto en el software MPLAB X y se proponen seis pasos de programación que incluyen el proceso de simulación.

Fialmente, en el capítulo quinto se introducen los microcontroladores fabricados por Atmel (*ATtiny2313*), *NXP-Freescale*, (*MC9So8]S16*) y *Texas Instruments* (*MSP430F2272*). Estos microcontroladores son comparables con el PIC16628A. Sobre estos microcontroladores se desarrollan algunos de los ejemplos trabajados en los dos capítulos anteriores.

En cada uno de los capítulos se incluyen una serie de ejercicios a fin de que el lector practique y logre mayores habilidades para desarrollar sus propios programas.

Los softwares que se usan en este libro son para efectos netamente académicos, como ejercicio para explicar y hacer comparación de diferentes fabricantes de microcontroladores; por tanto, no tienen ningún fin comercial.



Capítulo I



Sistemas de Numeración

A lo largo de la historia los seres humanos han tenido la necesidad de contar: “en la época primitiva solo contaban ovejas, bultos, los días transcurridos entre el cambio de luna, etc.; contar es la primera operación matemática que se requiere” (Ritterman, S., 1986, p. 47). Hoy día el hombre requiere hacer muchas operaciones matemáticas (contar, sumar, restar, comparar, multiplicar, dividir, conversiones, entre otras) para poder controlar diferentes procesos.

Los circuitos digitales pueden realizar operaciones y cálculos que el hombre tardaría mucho tiempo en realizarlas manualmente Ritterman comenta:

los computadores realizan actividades y cálculos rápidamente, utilizando unas pocas fracciones de segundos a través de circuitos electrónicos, que si se realizaran a papel y lápiz se emplearía mayor tiempo. Al realizar estos cálculos la computadora genera una serie de respuestas que pueden ser datos o un conjunto de instrucciones para controlar un proceso automáticamente (Ritterman, S., p. 47).

Estas operaciones se pueden realizar en diferentes sistemas de numeración, los circuitos digitales las realizan en binario. Estos números pueden venir del mundo exterior al circuito digital. Las entradas y salidas (*In/Out*) para una computadora pueden ser señales análogas o digitales y se obtienen a través de un teclado o del estado de los sensores. La información de estas señales puede darse en la forma de un número decimal, binario o hexadecimal. Por esta razón, a continuación se explica cada uno de los sistemas de numeración empleados en este proceso.

1.1 Números decimales

De acuerdo con Ritterman el término 'decimal' viene de la palabra latina decimus que significa "diez" y del término 'dígito' cuyo significado es "dedo" (Cf. Ritterman, 1986, p. 49). La principal ventaja del sistema decimal es que cuenta con pocos símbolos y que su posición indica un valor específico.

En un sistema de numeración posicional antes de utilizar la siguiente posición se comienza con el primer número (0) continuando en una serie hasta el último número (9). (ver tabla 1).

Tabla 1.
Los primeros diez números decimales.

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Cuando se llena la posición más baja, de derecha a izquierda, se pone un cero en ella y se comienza con el número uno en la siguiente posición hasta utilizar el mayor número, ver Tabla 2. Posteriormente se pone el número cero en la posición baja y se comienza el número dos en la segunda posición y así sucesivamente.

La base o raíz de un sistema es el número de símbolos que éste contiene. La base del sistema decimal es diez y nos indica cuántos símbolos se utilizan.

Tabla 2.
Combinaciones con dos dígitos de números decimales.

10
11
12
⋮
20
21
22
⋮
30
31
32
⋮
40
41

42
⋮
50
51
52
⋮
60
61
62
⋮
⋮
90
91
92
⋮
99
100
101
102
⋮

1.2 Números Binarios

El sistema en base dos se conoce como binario. Como se mencionó anteriormente, los aparatos electrónicos trabajan en base de dos estados, lo cual es compatible con el sistema binario. En una computadora los datos decimales se convierten en binarios y todos los cálculos se realizan a partir del sistema binario.

El sistema binario es un sistema posicional. Por lo tanto, para su ejecución se sigue el mismo procedimiento que en el sistema decimal. Sin embargo, la serie comienza en cero y se termina en uno. Ver Tabla 3.

Tabla 3.
Primeros números binarios.

0
1

Para las siguientes combinaciones se comienza con el número uno en la siguiente posición más significativa y se cuenta nuevamente desde la posición menos significativa (ver tabla 4).

Tabla 4.
Cuatro primeros números binarios.

00
01
10
11

Las siguientes combinaciones se explican en la tabla 5:

Tabla 5.
Se observan los ocho primeros números binarios.

000
001
010
011
100
101
110
111

El término ‘dígito’ representa un número en el sistema decimal. En el sistema binario la palabra ‘bit’ representa un número. El bit menos significativo **LSB** (*Least Significant Bit*) es el de la derecha y el bit más significativo **MSB** (*Most Significant Bit*) es el de la izquierda, ver ejemplo 1.

En el sistema decimal las posiciones son unidades, decenas, centenas, etc., porque la base es 10. En el sistema binario la base es 2. Por ende, las posiciones son: $2^n, \dots, 2^5, 2^4, 2^3, 2^2, 2^1, 2^0$. Es decir: $2^n, \dots, 32, 16, 8, 4, 2, 1$. El **LSB** está en la posición 2^0 , la posición siguiente es 2^1 y así sucesivamente. El **MSB** está en la posición 2^n . Para pasar un número binario a decimal se multiplica cada posición por el número binario correspondiente y al final se realiza una suma, ver ejemplo 1.

Ejemplo 1

Hallar el equivalente decimal de 1011010.

Solución

$$\begin{array}{rcccccccc}
 \text{MSB} \rightarrow & 1 & 0 & 1 & 1 & 0 & 1 & 0 & \leftarrow \text{LSB} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 \times 2^6 & 0 \times 2^5 & 1 \times 2^4 & 1 \times 2^3 & 0 \times 2^2 & 1 \times 2^1 & 0 \times 2^0 & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 \times 64 & 0 \times 32 & 1 \times 16 & 1 \times 8 & 0 \times 4 & 1 \times 2 & 0 \times 1 & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 64 + & 0 + & 16 + & 8 + & 0 + & 2 + & 0 = & 90_{10}
 \end{array}$$

Ejemplo 2

Hallar el equivalente decimal de 1100011.

Solución

$$\begin{array}{rcccccccc}
 \text{MSB} \rightarrow & 1 & 1 & 0 & 0 & 0 & 1 & 1 & \leftarrow \text{LSB} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 \times 2^6 & 1 \times 2^5 & 0 \times 2^4 & 0 \times 2^3 & 0 \times 2^2 & 1 \times 2^1 & 1 \times 2^0 & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 1 \times 64 & 1 \times 32 & 0 \times 16 & 0 \times 8 & 0 \times 4 & 1 \times 2 & 1 \times 1 & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\
 & 64 + & 32 + & 0 + & 0 + & 0 + & 2 + & 1 = & 99_{10}
 \end{array}$$

Siguiendo el mismo procedimiento el lector puede observar que el equivalente de los primeros 16 números decimales y binarios son los que se muestran en la tabla 6.

Tabla 6.
Primeros quince números decimales y su equivalente en binario.

Decimal	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

1.3 Números Hexadecimales

La base del sistema hexadecimal es 16 y frecuentemente se le llama *hex*. En este sistema se establecieron los símbolos del cero al nueve, seguidos de las letras **A** hasta **F** para los últimos seis números.

En la tabla 7 se observan los primeros números hexadecimales con sus equivalentes decimales.

Tabla 7.
Primeros números hexadecimales con sus equivalentes decimales.

Hexadecimal	Decimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	10
B	11
C	12
D	13
E	14
F	15

La secuencia hexadecimal funciona de la misma forma que los anteriores sistemas. Las combinaciones siguientes son las que se observan en la tabla 8. La conversión del hexadecimal a decimal es muy sencilla, pues la base del sistema hexadecimal es 16 (ver el ejemplo 3).

Tabla 8.
Algunos números hexadecimales con sus equivalentes decimales.

Hexadecimal	Decimal
10	16
11	17
12	18
13	19
14	20
⋮	⋮
20	32
21	33
22	34
23	35
⋮	⋮
30	48
31	49
32	50
⋮	⋮

Ejemplo 3

Hallar el equivalente decimal de $5CAB_{16}$.

Solución

$$\begin{array}{ccccccc}
 \text{MSB} \rightarrow & 5 & C & A & B & & \leftarrow \text{LSB} \\
 & \downarrow & \downarrow & \downarrow & \downarrow & & \\
 & 5 \times 16^3 & C \times 16^2 & A \times 16^1 & B \times 16^0 & & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & & \\
 & 5 \times 4096 & 12 \times 256 & 10 \times 16 & 11 \times 1 & & \\
 & \downarrow & \downarrow & \downarrow & \downarrow & & \\
 20480 & + & 3072 & + & 160 & + & 11 = 23723_{10}
 \end{array}$$

En la tabla 9 se observan los equivalentes de las primeras 16 combinaciones entre los sistemas decimal, binario y hexadecimal.

Tabla 9.

Se observan las primeras 16 combinaciones de la numeración hexadecimal y su equivalente en el sistema decimal y binario.

Hexadecimal	Decimal	Binario
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Para pasar del sistema decimal al binario hay que seguir el procedimiento del ejemplo 4.

Ejemplo 4

Pasar 50_{10} a binario.

Solución

Para empezar hay que dividir en dos; si el cociente de la división es exacto se coloca cero (0) y si es inexacto se coloca uno (1). El resultado de la primera división es el **LSB** del número binario y así sucesivamente:

$$\begin{array}{l} \frac{50}{2} = 25, \text{ la división es exacta se coloca } 0 \leftarrow \text{LSB} \\ \frac{25}{2} = 12, \text{ la división es inexacta se coloca } 1 \\ \frac{12}{2} = 6, \text{ la división es exacta se coloca } 0 \\ \frac{6}{2} = 3, \text{ la división es exacta se coloca } 0 \\ \frac{3}{2} = 1, \text{ la división es inexacta se coloca } 1 \\ \frac{1}{2} = 0, \text{ la división es inexacta se coloca } 1 \leftarrow \text{MSB} \end{array}$$

El resultado se lee de abajo hacia arriba: $50_{10} = 110010_2$. También se puede pasar de decimal a hexadecimal. En el ejemplo 5 se muestra el procedimiento.

Ejemplo 5

Pasar 987_{10} a hexadecimal.

Solución

Para empezar hay que dividir en 16. El residuo de la división es el **LSB** del número hexadecimal, el cociente se vuelve a dividir en 16. El residuo es el segundo **LSB** del número hexadecimal y así sucesivamente. Cuando la última división del número es menor a 16, entonces no se hace la división, sólo se pasa el número:

$$\begin{array}{l} \frac{987}{16} = 61, \text{ el residuo de la división es: } 987 - (16 \times 61) = 987 - 976 = 11 = B \\ \frac{61}{16} = 3, \text{ el residuo de la división es: } 61 - (16 \times 3) = 61 - 48 = 13 = D \\ \frac{3}{16} = 0, \text{ el residuo de la división es: } = 3 = 3 \end{array}$$

El resultado se lee de abajo hacia arriba: $987_{10} = 3DB_H$.

Ejercicios propuestos

1. Convertir del sistema binario al decimal y hexadecimal:

1101011_2	100100011_2	100111_2
110111_2	11011_2	110101_2
00101000_2	10111011_2	11010101_2
1000101011_2	111101_2	1001000101_2

2. Convertir del sistema decimal al binario y hexadecimal:

120_{10}	151_{10}	250_{10}
42_{10}	57_{10}	33_{10}
100_{10}	265_{10}	341_{10}
133_{10}	479_{10}	570_{10}

3. Convertir del sistema hexadecimal al binario y decimal:

FB_H	$A35_H$	$2F_H$
189_H	$9C2_H$	176_H
$F9A_H$	$E4D_H$	$2AB_H$
$5DB_H$	$F79_H$	$F9AB_H$



Capítulo II



Memorias y Microprocesadores

La unidad mínima que se puede almacenar es **1 bit** (0 ó 1), el dispositivo que se usa para ello es un *Flip-Flop (F-F)*; el cual está construido por compuertas lógicas, y estas a su vez por transistores. También se pueden guardar 4 bits que equivalen a 1 **nibble**, el dispositivo que almacena estos bits es un Registro de 4 bits. Si se toman 2 **nibbles** se forma 1 **byte** que equivale a 8 **bits**, los cuales se guardan en un registro de 8 bits. Cuando se toma un **byte** o más se forma una **palabra** (Floyd, 2006). A continuación presentamos esquemáticamente los conceptos aquí mencionados,

1 **bit** → Unidad mínima de almacenamiento

4 **bits** = 1 **nibble**

1 **byte** = 8 **bits** = 2 **nibbles**.

1 **byte** o más = 1 **palabra**.

2.1 Registro

Hay aplicaciones que requieren almacenar 4 bits o más, para ello se usa un Registro. Por ejemplo almacenar la respuesta de una suma o el estado de sensores digitales, para ello se usan los registros, éstos varían según el tamaño de bits que se desean guardar. Para Floyd los registros “son circuitos lógicos secuenciales, que están íntimamente relacionados con los contadores digitales. Los registros se utilizan principalmente para almacenar datos digitales y, normalmente no poseen una secuencia característica interna de estados como los contadores” (2006, p. 552).

Los registros están contruidos por “un conjunto de F-Fs (Flip-Flops), y son muy importantes en las aplicaciones que precisan almacenar y transferir datos dentro de un sistema digital. En general, un registro se utiliza únicamente para almacenar y desplazar datos (1s y 0s), que introduce en él una fuente externa” (Floyd, p. 552).

Un registro es:

Un circuito digital con dos funciones básicas: *almacenamiento de datos* y *el movimiento de datos*. La primera le convierte en un tipo importante de dispositivo de *memoria*, esta capacidad es el número total de bits (1s y 0s) de un dato digital que puede almacenar, cada etapa (F-F) de un registro representa un bit de su capacidad de almacenamiento. (Floyd, 2006, p. 552)

De acuerdo con Floyd, se puede afirmar que el número de etapas de un registro determina su capacidad. Hay registros de 4 bits, 8 bits, 16 bits, 32 bits, entre otros. Sin embargo, el tamaño de los bits varía, pues depende de cuántos se desean guardar. En la Figura 1 se muestran algunos ejemplos. Hay casos de registros que tienen tamaños de 10 bits ó 14 bits. El tamaño depende de la aplicación y de la cantidad de bits que se requiere almacenar. “La capacidad de desplazamiento de un registro permite el movimiento de los datos de una etapa a otra dentro del registro, o la entrada o salida del mismo, en función de los impulsos de reloj que se apliquen” (Floyd, p. 552).

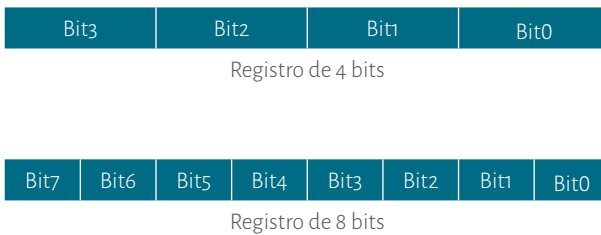


Figura 1. Ejemplos de registros de 4 bits y 8 bits.

Fuente. Elaboración propia

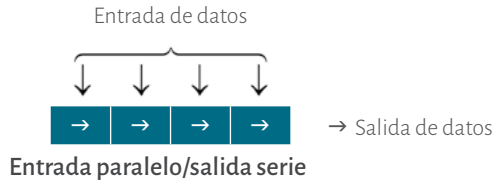
En la Figura 2 se ilustra cómo se mueven los datos en los registros. El bloque representa un registro de 4 bits y las flechas indican la dirección en que se mueven los datos.



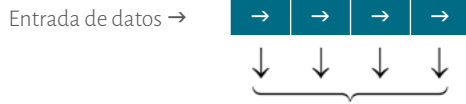
Entrada serie/salida serie con desplazamiento a la derecha



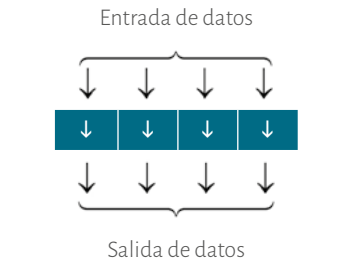
Entrada serie/salida serie con desplazamiento a la izquierda



Entrada paralelo/salida serie



Entrada serie/salida paralelo



Entrada paralelo/salida paralelo



Rotación a la derecha



Rotación a la izquierda

Figura 2. Movimiento básico de los registros.

Fuente. Elaboración propia, adaptado de Floyd, 2006, p. 553

2.2 Memorias

A medida que la aplicación crece, se requiere almacenar muchos más bits y es complicado guardarlos en registros. Por ello, se diseñaron las memorias: “La memoria es la parte de un sistema que almacena datos binarios en grandes cantidades. Las memorias están formadas por matrices de elementos de almacenamiento (F-F o condensadores)” (Floyd, p. 603).

Asimismo, es importante aclarar que “En una memoria la unidad mínima de almacenamiento es una *celda* la cual puede almacenar 1 bit (1 ó 0). Una *matriz* está construida con varias celdas” (Floyd, p. 603). En la Figura 3 se muestra varios ejemplos de matrices. También se puede decir que una memoria está construida a partir de muchos registros donde cada uno de ellos tiene una dirección diferente.

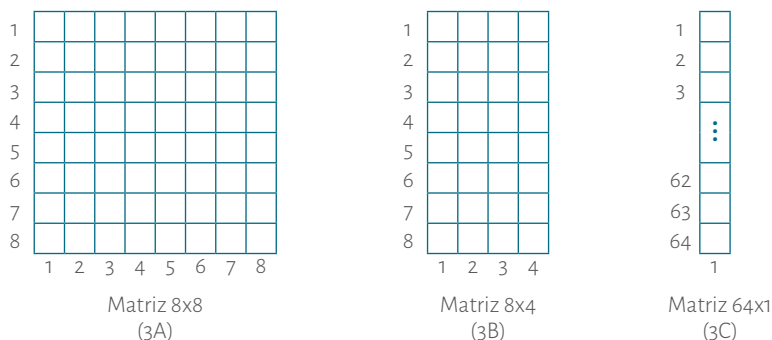


Figura 3. Diferentes tamaños de matrices.

Fuente. Elaboración propia, adaptado de Floyd, 2006, p. 602

Una memoria se identifica por el número de palabras (número de registros) que puede almacenar, multiplicado por el tamaño de la palabra (número de bits de cada registro).

La fórmula es $M \times N$ (Figura 4).

Siendo: **M** el número de palabras.

N el tamaño de la palabra.

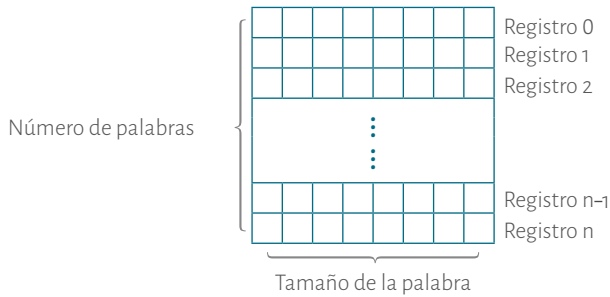


Figura 4. Tamaño de una Memoria.

Fuente. Elaboración propia.

Si se tiene una matriz de $8 \times 8 = 64$ bit o una memoria de 8 bytes (Figura 3A).

Si se tiene una matriz de $8 \times 4 = 32$ bits o una memoria de 8 nibbles (Figura 3B).

Si se tiene una matriz de 64×1 o una memoria de 64 bits (Figura 3C).

Entonces una memoria de $16k \times 8$ almacenará 16384 palabras de 8 bits. También decimos que contiene 131.072 bits, cantidad que equivale a $2^{14} = 16.384$ bytes = 16 k. (Floyd, 2006).

En una memoria se pueden hacer dos procesos fundamentales: escritura y lectura. Cuando se escribe en una memoria se colocan los datos en una posición específica de la memoria y cuando se lee se extraen sus datos. El direccionamiento hace parte de los dos procesos y consiste en seleccionar la posición de memoria donde se quiere escribir o leer. Para ello, hay un circuito que se encarga del proceso de decodificación. Floyd aclara al respecto: “La decodificación de la posición de la memoria a la cual se quiere acceder la decodifica un circuito especial el cual es llamado decodificador de direcciones” (Floyd, 2006, pp. 605-606).

2.2.1 Escritura

Para el proceso de escritura hay que seguir tres pasos fundamentales, ver Figura 5:

1. Se coloca la dirección en la que se quiere escribir el dato en el **bus de direcciones**. El decodificador se encarga de interpretar esa posición.
2. Se coloca el dato que se quiere escribir en el **bus de datos**.
3. Se da la orden de escribir a través del **bus de control**.

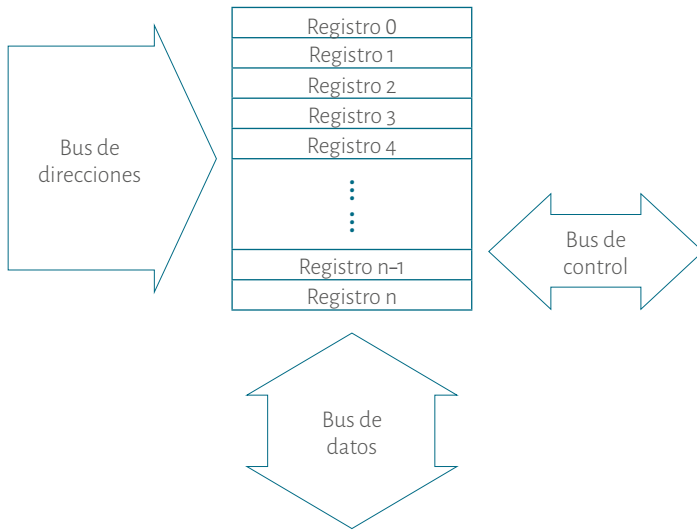


Figura 5. Proceso de escritura y de lectura de una memoria.

Fuente. Elaboración propia

2.2.2 Lectura

Para el proceso de lectura hay que seguir tres pasos fundamentales, ver Figura 5:

1. Se coloca la dirección del dato que se quiere leer en el **bus de direcciones**. El decodificador se encarga de interpretar esa posición.
2. Se da la orden de lectura a través del **bus de control**.
3. En el **bus de datos** se coloca una copia del dato que hay en la dirección seleccionada de la memoria.

2.2.3 Bus de datos

El bus de datos es el que permite ingresar o extraer la información que va a ser escrita o leída en una memoria. Este bus es bidireccional, es decir, sale e ingresa a la memoria (Floyd, 2006). Físicamente este bus es una serie de cables o pistas (en las tarjetas). El tamaño de este bus depende del tamaño de la palabra y puede ser de 8 bits, 16 bits, 32 bits, etc.

2.2.4 Bus de direcciones

El bus de direcciones contiene la información de la dirección del dato que se desea leer o escribir en la memoria. Este bus es unidireccional, pues solo llega a la

memoria (Floyd, 2006). Físicamente este bus es una serie de cables o pistas en las tarjetas. El tamaño de este bus varía según el tamaño de la memoria. Por ejemplo, para direccionar una memoria de 2 kBytes son necesarias 11 líneas, es decir $2^{11} = 2048$ bytes.

2.2.5 Bus de control

Este bus como su nombre lo indica es el que controla el proceso de lectura y de escritura de una memoria. Este bus es bidireccional, es decir sale e ingresa a la memoria (Floyd, 2006).

Hay dos grupos fundamentales de memorias las cuales son RAM y ROM. A continuación se explica en detalle cada una de ellas.

RAM (Random – Access – Memory) Memoria de acceso aleatorio: este tipo de memorias tarda el mismo tiempo en acceder a cualquier posición de memoria. Las memorias RAM se pueden seleccionar en modo de escritura o en el modo de lectura, es decir, se le puede leer o escribir (Floyd, 2006). Esta memoria es volátil porque pierde sus datos al desconectarse la alimentación.

ROM (Read – Only – Memory) Memoria de solo lectura: este tipo de memorias almacenan los datos de forma permanente o semipermanente y solo tienen la opción de ser leídas. No obstante, hay algunas que se les puede escribir, más adelante las analizaremos (Floyd, 2006). Estas memorias no son volátiles pues conservan los datos aunque no tengan alimentación.

De acuerdo con lo anteriormente visto, se puede ver que la principal diferencia de estas dos memorias radica en que RAM es volátil y la ROM no.

2.2.6 Familias de la RAM

Siguiendo a Floyd, esta memoria se divide en dos grandes grupos: SRAM y DRAM.

- **SRAM RAM estática.** Este tipo de memoria está construida con *Flip-Flops*, por ende, almacena los datos hasta que se desconecte.
- **DRAM RAM dinámica.** Este tipo de memoria está construida con condensadores. Por ende, no almacena los datos por mucho tiempo y hay que refrescarla periódicamente a través de un circuito adicional.

Los dos tipos de memoria pierden los datos al desconectar la alimentación. Si se hace una comparación entre las dos encontramos las siguientes características.

SRAM	DRAM
<ul style="list-style-type: none"> • Se puede leer más rápido • Tiene un tamaño mayor 	<ul style="list-style-type: none"> • Se puede leer rápido • Tiene un tamaño menor

Fuente, elaboración propia, datos tomados de Floyd, 2006, p. 608.

En la Figura 6 se muestra la familia de memorias RAM.

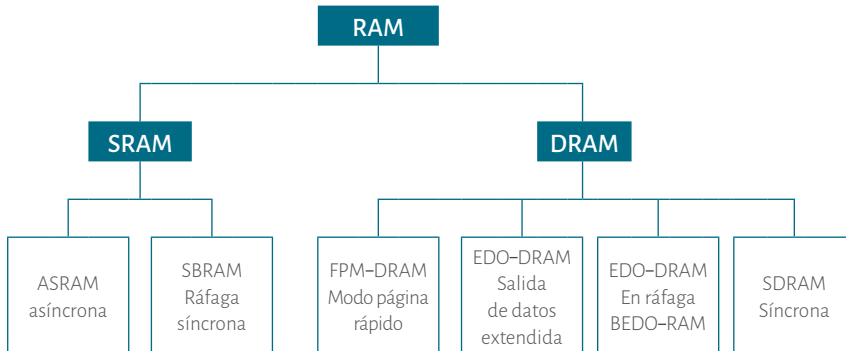


Figura 6. Familia de memorias RAM.

Fuente. Elaboración propia, adaptado de Floyd, Figura 10.7, 2006, p. 609.

A continuación se explica brevemente cada una de las memorias:

- **ASRAM:** el funcionamiento de esta memoria no está sincronizado con el reloj del sistema.
- **SBRAM:** esta memoria está conectada con el reloj del sistema. Es más rápida porque tiene un circuito que le permite obtener los cuatro siguientes datos. Este método es llamado de ráfaga.
- **MEMORIA CACHÉ:** la memoria caché es una aplicación de la SRAM. Es una memoria de alta velocidad. Se utiliza para guardar los datos o las instrucciones más recientes. Hay dos niveles de esta memoria, de acuerdo con Floyd:
 - **Caché L1.** Caché de nivel 1. Esta memoria integrada en el mismo microprocesador y su capacidad de almacenamiento es muy limitada. Es conocida como caché primaria.
 - **Caché L2.** Caché nivel 2. Esta memoria está compuesta por integrados externos al microprocesador y, a diferencia de las memorias del nivel uno, su capacidad es mayor secundaria.

En la Figura 7 se muestran las dos clases de memorias caché:

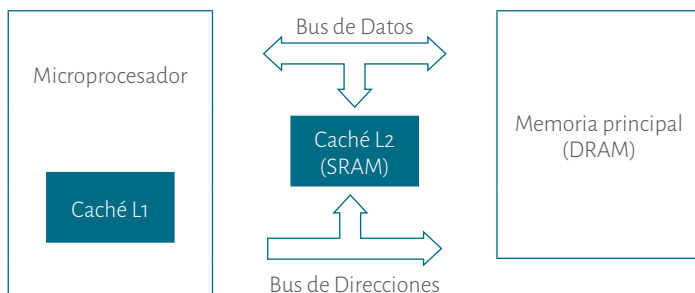


Figura 7. Diagrama de bloques donde se observa memoria caché L1 y L2.

Fuente. Elaboración José Luis Rodríguez. Datos tomados de Floyd, 2006. Fundamentos de sistemas digitales (p. 615).

- **FPM-RAM:** esta memoria depende de que las siguientes direcciones de la memoria, a las que haya que acceder, se encuentren en la misma fila, en la misma página. De esta forma la memoria es más rápida.
- **EDO-RAM:** esta es una memoria con salida de datos extendida. Es muy similar a la FPM-DRAM y es más rápida.
- **BEDO-RAM:** esta memoria cuenta con salida de datos extendida en ráfaga. Es una EDO-DRAM en ráfaga.
- **SD-RAM:** La operación de la memoria está sincronizada con el reloj del sistema que es el mismo del microprocesador. Opera como SBRAM.

2.2.7 Familias de la ROM

En la Figura 8 se muestra la familia de memorias ROM.

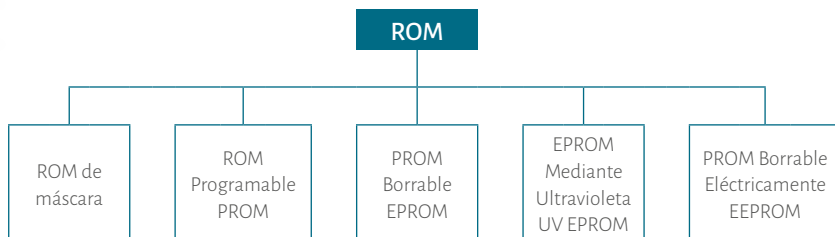


Figura 8. Familia de memorias ROM.

Fuente. Elaboración propia, adaptado de Floyd, Figura 10,22, 2006. p. 624.

A continuación, se explica brevemente cada una de la familia de memorias ROM:

ROM de máscara: se denomina ROM y es programada en el proceso de fabricación. Por ende, esta programación no se puede modificar. Esta memoria almacena funciones básicas.

PROM: este tipo de memorias tiene un sistema de hilo-fusible que se introduce en el proceso de fabricación. El proceso de grabación es irreversible. Una vez programada no se le puede cambiar. Cuando se programa la memoria, los hilos-fusibles se rompen para almacenar 0 (con una corriente lo suficientemente grande para fundirlo) y no se rompen para almacenar 1. Hay tres tipos de tecnologías de fusibles que son:

- Conexión de metal
- Conexión de silicio
- Uniones pn

EPROM: una memoria EPROM es una PROM borrable. Es necesario borrar el programa existente para volver a programarla. Este tipo de memoria requiere del proceso de guardar un programa, con una o varias tareas (instrucciones), en una memoria. Por ejemplo, manejar la secuencia de movimientos de uno o varios motores los cuales mueven un robot.

UV EPROM: para borrarla es necesario exponerla a rayos ultravioleta de alta intensidad. El tiempo de exposición es de varios minutos, dependiendo del tamaño del programa que tenga: a mayor programa mayor tiempo.

EEPROM: esta memoria es programable y borrable eléctricamente. Se puede reprogramar dentro del propio circuito final.

FLASH: esta memoria es ideal. Tiene alta capacidad de almacenamiento y no es volátil. Se puede leer y escribir en el circuito que esté implementada. Es rápida y económica, además, es muy utilizada en los computadores portátiles.

2.3 Micropocesor

Es un circuito integrado dividido en tres bloques: **ALU, unidad de control** y **matriz de registros** (Floyd, 2006). En la Figura 9 se observa el esquema de un microprocesador (μp).



Figura 9. Esquema de un microprocesador.

Fuente. Elaboración propia

De acuerdo con la definición de Ritterman S., (1988) y Floyd (2006), cada bloque del microprocesador:

ALU (Unidad Aritmético Lógica) es el encargado de ejecutar las operaciones o instrucciones matemáticas (suma, resta, multiplicación, división, entre otras) y lógicas (AND, OR, NOT, entre otras).

La unidad de control cumple con dos tareas fundamentales: a) regular el procesamiento de datos y b) generar señales de tiempo y de compuertas. La segunda tarea activa los circuitos apropiados, y la primera sincroniza las operaciones. En otras palabras, la unidad de control es quien se encarga de decodificar las instrucciones que tiene que ejecutar el **ALU**.

La matriz de registros contiene un número determinado de registros en los cuales se pueden almacenar, temporalmente datos, tales como las respuestas de operaciones, además de otras. Entre los registros más importantes se encuentra: (1) el contador de programa, encargado de apuntar a la posición o dirección de la instrucción que se está ejecutando, (2) el acumulador que lleva el resultado de las operaciones que se realizan, (3) bus de datos y (4) bus de direcciones, los cuales interconectan los dispositivos anteriormente mencionados. El bus de datos es el que transporta la información que entra y sale del microprocesador y, por lo general, es de 8 bits. Por su parte, el bus de direcciones es el encargado de direccionar los dispositivos de memoria. Sin embargo además de estos 4 existen más registros que sirven para el funcionamiento del microprocesador.

Según Ritterman el funcionamiento de un microprocesador es así:

(...) el contador de programa inicia en 00_H cuando se inicia el proceso. Esta cuenta se transfiere a un registro de direcciones, y este se va al bus de direcciones. Así, 00_H es la primera dirección cuando se

accesa la ROM. La primera instrucción del usuario se encuentra en la posición 00_H , y esta palabra se transfiere desde la ROM a través del bus de datos. Luego, el registro de datos transfiere esta palabra a un decodificador de instrucciones que es parte de la unidad de control que interpreta la palabra. Después, ésta genera señales de control para el desarrollo de las instrucciones, tales como suma, resta o transferencia de datos. Las señales de control se transmiten por el bus de control. Cuando se codifica una instrucción, el contador de programa se incrementa a 01_H . Normalmente, la posición 01_H contiene datos pertenecientes a la instrucción de la posición 00_H . En cuanto el bus de direcciones se habilita nuevamente, el dato en la posición 01_H se transfiere de la ROM al registro de datos, luego las señales de control apropiadas transfieren el contenido del registro de datos al acumulador. Cuando el contador de programa pasa a 02_H se obtiene la siguiente instrucción, el proceso continúa, una posición a la vez, hasta que se obtiene el resultado que se desea, en otras palabras, hasta que se ejecute todo el programa del usuario (Ritterman, 1988, p. 416).

2.3.1 Instrucción

Una instrucción es toda operación realizada por un microprocesador. El número y el tipo de instrucciones está determinado por la estructura del ALU del microprocesador. Allí radica una de las grandes diferencias de los microprocesadores. Un conjunto de instrucciones es un grupo de instrucciones que un microprocesador puede ejecutar, (Uyemura, 2000).

Según Uyeruma, una instrucción requiere los siguientes ciclos para su ejecución:

1. Ciclo de recuperación de la instrucción (t_p)
2. Decodificación de la instrucción (t_D)
3. Ejecución de la instrucción (t_{EX})
4. Almacenamiento (t_s)

El tiempo de ejecución de una instrucción (t_{INS}) está formado por la suma de estos 4 tiempos así:

$$t_{INS} = t_F + t_{ID} + t_{EX} + t_S$$

Fuente. (Uyeruma, 2006, p. 407)

El tiempo de ejecución de una instrucción depende de la velocidad de reloj. Así lo corrobora Uyeruma: “Como la temporización usualmente está controlada por una señal de reloj aplicada externamente, entre más grande sea la frecuencia del reloj, a mayor velocidad funciona el microprocesador” (2000, p. 407).

2.4 Dispositivos Periféricos

Ritterman afirma: “sin importar cuál sea la aplicación, la entrada para la computadora se obtiene del mundo exterior. Los dispositivos periféricos son las vías entre la computadora, y el mundo exterior” (1988, p. 359). Estos dispositivos son los encargados de entrar o sacar los datos del exterior al microprocesador. Los datos pueden ser digitales o análogos. En realidad sin ellos no se podría controlar ningún dispositivo o ningún proceso.

Para Uyeruma (2000) entre los dispositivos de entrada se encuentran: teclado, el ratón, la unidad de disco, el CD ROM, entre otros. Y entre los de salida: el monitor, unidad de disco en forma de escritura, impresora, entre otros.

2.5 Sistemas Micro-Procesados

Con los dispositivos periféricos se puede realizar un sistema Micro-Procesado, el cual va a cumplir una o varias tareas específicas. Los elementos que se requieren en un sistema Micro-Procesado son: un microprocesador (μp), una memoria RAM, una memoria ROM, un puerto de I/O y un reloj. A medida que la tarea lo requiera se le pueden acondicionar otros periféricos o más memoria. Para interconectar todos estos dispositivos se usan los buses de datos, de direcciones y de control. En la Figura 10 se observan los elementos más importantes que contiene dicho sistema: un microprocesador de 8 bits (bus de datos 8 bits), una memoria RAM 2kx8 que requiere un bus direcciones de 11 bits ($2^{11} = 2.048 \text{ BYTES} = 2 \text{ K}$), una memoria ROM 8kx8 con un bus direcciones de 13 bits ($2^{13} = 8.192 \text{ BYTES} = 8 \text{ K}$) y unos periféricos con 2 puestos de 8 bits cada uno.

A continuación se describe la tarea que cumple cada uno de los elementos que componen dicho sistema:

- **ROM:** tiene el programa que el usuario diseñó (instrucciones).
- **El μp :** es el encargado de ejecutar el programa.
- **RAM:** almacena los datos temporales que necesitan en la ejecución del programa.
- **Los periféricos:** son los que permiten ingresar o sacar los datos del exterior que el programa requiera.

2.5.1 Desventajas

Como el sistema Micro-Procesado, cada circuito periférico es un integrado. Por lo tanto, requiere de:

- Un tamaño considerable (gran tamaño).
- Bastante consumo de corriente.
- Pocos puertos.

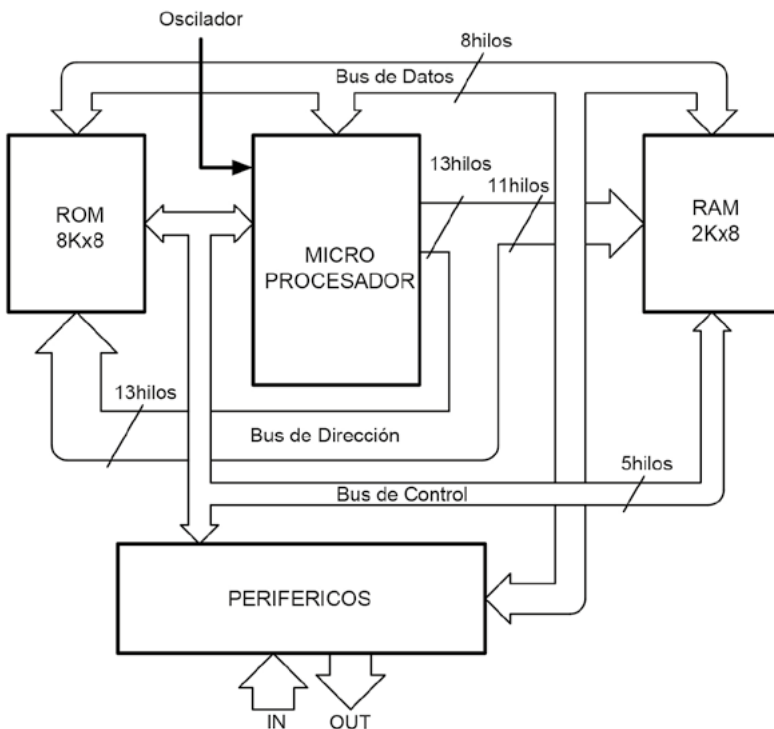


Figura 10. Diagrama de bloques de un sistema microprocesado.

Fuente. Elaboración José Luis González, imagen adaptada del primer diseño IBM Harvard Mark I



Capítulo III

Microcontroladores

Los microcontroladores contienen los mismos elementos del sistema microprocesado pero con la diferencia de que todos estos dispositivos están en un solo integrado. Los pines que tienen son: los periféricos de entrada y salida, los de alimentación y el oscilador.

3.1 Ventajas

El sistema microcontrolado contiene todo en un solo circuito integrado. Por lo tanto, tiene las siguientes ventajas:

- Poco tamaño
- Bajo consumo de corriente
- Es más económico

Hay varios fabricantes de microcontroladores en el mundo, entre los más importantes encontramos:

- Intel
- Phillips
- Zilog
- NXP - Freescale
- Microchip
- Texas Instruments
- Atmel.

Por ser los cuatro últimos fabricantes los más conocidos en Colombia, en este libro se explicarán y se efectuarán aplicaciones en microcontroladores tanto de Microchip como de NXP—Freescale, Texas Instruments y Atmel iniciando por el primero.

Aplicaciones de los microcontroladores

- Robótica
- Periféricos de computadores
 - Teclados
 - Impresoras
 - Discos duros
- Industria automotriz
- Electrodomésticos
- Instrumentación
- Alarmas
- Electromedicina
- Sistemas de navegación espacial
- Entre otros.

3.2 Arquitectura de los Microcontroladores

Los microprocesadores se dividen en dos arquitecturas fundamentales así: la arquitectura Von Newman y la arquitectura Harvard. En la arquitectura Von Newman el bus de datos está compartido con el bus de direcciones. Un ejemplo de ello son los microcontroladores de Freescale de 8 bits, cuyo bus de datos es de 8 bits y el bus de direcciones es de 16 bits. Por lo tanto, solo se comparte la mitad del bus (Galeano, 2009). La arquitectura Harvard “(...) contiene buses separados para la decodificación de instrucciones y de datos (...). Este doble bus permite que la gran mayoría de las instrucciones sean ejecutadas en un solo ciclo de máquina, un ejemplo de ello son los microcontroladores de microchip” (Galeano, 2009, p. 67).

Según el juego o repertorio de instrucciones que un microprocesador es capaz de ejecutar se puede clasificar en tres grandes grupos:

1. **CISC:** → (*Complex Instruction Set Computer*) Juego de instrucciones complejo para computador. Ejemplo: Intel-NXP (Freescale).
2. **RISC:** → (*Reduced Instruction Set Computer*) Juego de instrucciones reducido para computador. Ejemplo: Microchip, Atmel, Texas instruments (Mandado, 2007, p. 23).
3. **SISC:** → (*Specific Intruction Set Cumputer*) Juego de instrucciones específico para computador. Ejemplo: fabricantes de relojes y de juguetes.

Los microcontroladores de Microchip se conocen como PIC, que en su traducción al español significa: Interfase Controlada Programable. Estos microcontroladores se pueden dividir en tres gamas que se caracterizan por su bus de datos, bus de direcciones, número de instrucciones, líneas de I/O, capacidad de memoria entre otras:

- Gama baja → 33 instrucciones
- Gama media → 35 instrucciones
- Gama alta → 64 instrucciones

En este capítulo se desarrollarán los ejemplos con microcontroladores de gama media. En esta gama se han presentado varias evoluciones, pues el PIC16C84 fue remplazado por el PIC16F84 (la F significa que tiene memoria flash) y éste, a su vez, por el PIC16F84A. Posteriormente, se desarrolló el PIC16F628A, considerado como un microcontrolador muy poderoso.

A esta gama también pertenecen los microcontroladores PIC16F873A, PIC16F874A, PIC16F877, PIC16F1X19, el PIC16F877A, entre otros. En la Tabla 10 se muestran algunas de sus características.

Tabla 10.
Comparación de algunos Microcontroladores gama media.
(Microchip, 2006).

Características	PIC-16F84A	PIC-16F628A	PIC-16F873A	PIC-16F874A	PIC-16F876A	PIC-16F877A
Memoria de programa (ROM) FLASH	1K X 14	2K X 14	4K X 14	4K X 14	8K X 14	8K X 14
Memoria RAM	68 X 8	224 X 8	192 X 8	192 X 8	368 X 8	368 X 8
Número de instrucciones (RISC)	35	35	35	35	35	35
Memoria EEPROM Data	64 X 8	128 X 8	128 X 8	128 X 8	256 X 8	256 X 8
USART	NO	SÍ	SÍ	SÍ	SÍ	SÍ
Oscilador interno	NO	SÍ	NO	NO	NO	NO
Pines de I/O	13	15	22	33	22	33
Tipo de empaque	DIP	DIP	DIP	PDIP	DIP	PDIP
Número de pines	18	18	28	40	28	40

Nota. Los datos de **PIC16F84A**, son tomados del documento del fabricante Microchip. Fuente Microchip, 2001 (p.1). Tomado de <http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf>

Los datos de **PIC16F628A** son tomados del documento del fabricante Microchip. Fuente Microchip, 2007, (pp. 1-5). <http://ww1.microchip.com/downloads/en/DeviceDoc/40044F.pdf>

Los datos de **PIC16F873A**, **PIC16F874**, **PIC16F876A** y **PIC16F877A** son tomados del documento del fabricante Microchip. Fuente Microchip, 2003, (p. 1). <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>

A simple vista son claras las ventajas del PIC16F628A, pues es un microcontrolador pequeño y poderoso. Por ello, este microcontrolador se selecciona para desarrollar los ejemplos más representativos a fin de comprender su funcionamiento. El lector podrá comparar y se dará cuenta que, con pocas modificaciones y en algunos casos sin modificación alguna, los ejemplos se pueden aplicar con los otros microcontroladores de la Tabla 10.

3.3 PIC16F628A

En los microcontroladores de Microchip hay **páginas** y **bancos**. Las primeras son como están divididas la memoria ROM (Flash), es decir en la cual se programa las aplicaciones del usuario; los segundos es la forma como está dividida la memoria RAM, en ella hay dos categorías de registros: registros de propósito específico y registro de propósito general, los primeros son los que el microcontrolador usa para su funcionamiento, los segundos son para el usuario, es decir, los puede usar para propósito de su programa. En la figura 11 se muestra la distribución de los bancos de la memoria RAM del PIC16F628A.

En la Figura 12 se muestra que la memoria ROM (Flash) del microcontrolador PIC16F628A es de 2k (desde 000_H hasta $3FF_H$). Además, se observa que en las posiciones 000_H y 004_H están los vectores *reset* y los vectores de interrupciones, *respectivamente*.

Hay dos registros fundamentales para la programación del microcontrolador en *Asembler*: el registro *STATUS* que hace parte de los registros de propósito específico, y el registro **W** (*Work*) el cual hace parte del microprocesador interno que posee el microcontrolador. El registro **W** es el registro principal y contra él se hacen todas las operaciones: suma, resta, comparaciones, entre otras. El registro *STATUS* tiene el estado de las operaciones. Es decir, si al hacer una suma hay acarreo, si al hacer una resta la respuesta es cero, entre otros casos que pueden darse. Ambos registros son de 8 bits.

Banco 0	Dirección	Banco 1	Dirección	Banco 2	Dirección	Banco 3	Dirección
INDF	00 h	INDF	80 h	INDF	100 h	INDF	180 h
TMR0	01 h	OPTION	81 h	TMR0	101 h	OPTION	181 h
PCL	02 h	PCL	82 h	PCL	102 h	PCL	182 h
STATUS	03 h	STATUS	83 h	STATUS	103 h	STATUS	183 h
FSR	04 h	FSR	83 h	FSR	104 h	FSR	184 h
PORTA	05 h	TRISA	85 h	-----	105 h	-----	185 h
PORTB	06 h	TRISB	86 h	PORTB	106 h	TRISB	186 h
-----	07 h	-----	87 h	-----	107 h	-----	187 h
-----	08 h	-----	88 h	-----	108 h	-----	188 h
-----	09 h	-----	89 h	-----	109 h	-----	189 h
PCLACH	0A h	PCLACH	8A h	PCLACH	10A h	PCLACH	18A h
INTCON	0B h	INTCON	8B h	INTCON	10B h	INTCON	18B h
PIR1	0C h	PIE1	8C h	-----	10C h	-----	18C h
-----	0D h	-----	8D h	-----	10D h	-----	18D h
TMR1L	0E h	PCON	8E h	-----	10E h	-----	18E h
TMR1H	0F h	-----	8F h	-----	10F h	-----	18F h
T1CON	10 h	-----	90 h	-----	-----	-----	-----
TMR2	11 h	-----	91 h	-----	-----	-----	-----
T2CON	12 h	PR2	92 h	-----	-----	-----	-----
-----	13 h	-----	93 h	-----	-----	-----	-----
-----	14 h	-----	94 h	-----	-----	-----	-----
CCPR1L	15 h	-----	95 h	-----	-----	-----	-----
CCPR1H	16 h	-----	96 h	-----	-----	-----	-----
CCP1CON	17 h	-----	97 h	-----	-----	-----	-----
RESTA	18 h	TESTA	98 h	-----	-----	-----	-----
TXREG	19 h	SPBRG	99 h	-----	-----	-----	-----
RCREG	1A h	EEDATA	9A h	-----	-----	-----	-----
-----	1B h	EEADR	9B h	-----	-----	-----	-----
-----	1C h	EECON1	9C h	-----	-----	-----	-----
-----	1D h	EECON2	9D h	-----	-----	-----	-----
-----	1E h	-----	9E h	-----	-----	-----	-----
CMCON	1F h	VRCON	9F h	-----	11F h	-----	-----
Registros de propósito general 80 bytes	20 h	Registros de propósito general 80 bytes	A0 h	Registros de propósito general 48 bytes	120 h	-----	-----
-----	-----	-----	-----	-----	14F h	-----	-----
-----	-----	-----	-----	-----	150 h	-----	-----
-----	-----	-----	-----	-----	-----	-----	-----
-----	-----	-----	-----	-----	-----	-----	-----
-----	6F h	-----	EF h	-----	160 h	-----	1EF h
16 Bytes	70 h	Acceso a 70 h-7F h	F0 h	Acceso a 70 h-7F h	170 h	Acceso a 70 h-7F h	1FO h
-----	-----	-----	-----	-----	-----	-----	-----
-----	7F h	-----	FF h	-----	17F h	-----	1FF h

Figura 11. Memoria RAM del PIC 16F628A.

Fuente. (Microchip, 2007, p. 17).

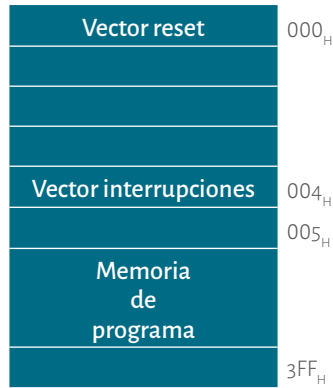


Figura 12. Memoria ROM del PIC 16F628A.

Fuente. (Microchip, 2007, p. 15).

A continuación se explica cómo funciona el registro STATUS bit a bit.

3.3.1 Registro STATUS

<i>IRP</i>	<i>RP1</i>	<i>RP0</i>	\overline{TO}	\overline{PD}	<i>Z</i>	<i>DC</i>	<i>C</i>
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Fuente. (Microchip, 2007, p. 22).

Bit 7: **IRP:** Selección de bancos general

1 → Banco 2, 3 (00h–FFh)

0 → Banco 0, 1 (100h–1FFh).

Bit 6,5: **RP1-RP0:** Selección de bancos específicos

00 → Banco 0 (00h–7Fh)

01 → Banco 1 (80h–FFh)

10 → Banco 2 (100h–17Fh)

11 → Banco 3 (180h–1FFh).

Bit 4: **\overline{TO} :** Tiempo–fuera (*Time–Out*)

1 → Después de encendido, cuando se ejecutan las instrucciones CLRWDT o SLEEP.

0 → Cuando WDT ha terminado su conteo.

- Bit 3: **\overline{PD}** : Prender–contar (*Power–Down*)
- 1 → Después de encendido, cuando se ejecuta la instrucción CL-RWDT
- 0 → Cuando está en ejecución la instrucción *SLEEP*.
- Bit 2: **Z**: Cero (*Zero*)
- 1: El resultado de una operación aritmética o lógica que da cero.
- 0: El resultado de una operación aritmética o lógica que no da cero.
- Bit 1: **DC**: Carry Digital
- 1: En una operación aritmética o lógica hay carry en el bit 4.
- 0: En una operación aritmética o lógica no hay carry en el bit 4.
- Bit 0: **C**: Carry
- 1: El resultado de una operación aritmética o lógica pasa de la capacidad de operación. Es decir que supera el carry de 8 bits.
- 0: El resultado de una operación aritmética o lógica no pasa de la capacidad de operación que es equivalente a 8 bits.

Hay otro tipo de registros de propósito específico que, a medida que se avance en el libro, se explicarán.

3.3.2 Instrucciones

Por lo pronto se explican las 35 instrucciones que tiene el microcontrolador. Estas instrucciones están divididas en 3 grandes grupos, ver Tabla 11: el primer grupo contiene las operaciones entre registros, el segundo grupo está compuesto por las operaciones de manipulación y prueba de bits, y el tercer grupo contiene las operaciones con constantes y de control (Microchip, 2007).

En todas las instrucciones hay ejemplos para que al lector se le facilite entender mejor el funcionamiento de cada una. Recuerde que **W** es el registro principal y **f** puede ser cualquier registro de la RAM. Se va a suponer que hay dos registros de propósito general en el banco cero, **Reg A** y **Reg B**, que están en las posiciones 20_H y 21_H respectivamente. Para propósito de los ejemplos se dará un valor a estos registros además al registro **W**;

$$W = CD_H$$

$$\text{Reg A} = 8E_H$$

$$\text{Reg B} = 7F_H$$

Además, el lector debe tener conocimientos previos en sistemas digitales (tablas de verdad de las compuertas lógicas, complemento a uno, entre otros), con ello se le facilita entender mejor los ejemplos.

3.3.2.1 Instrucciones de Operaciones entre registros. En estas instrucciones se encuentra la letra **f**, la cual se refiere a cualquier registro ya sea propósito específico o de propósito general. También se encuentra la letra **d**, la cual representa en dónde se quiere dejar la respuesta de la instrucción; si **d=0** la respuesta de la instrucción queda en el registro **W** y si **d=1** la respuesta de la instrucción queda en el registro **f**. (Microchip, 2007, p. 115). Enseguida se explican cada una de las instrucciones de este grupo:

Tabla 11.
Instrucciones del PIC16F628A.

No.	Nemónico	Descripción	Bit afectado en STATUS
<i>Primer grupo operaciones entre registros</i>			
1	ADDWF f,d	Operación matemática, SUMA. Así W + Reg F	C, DC, Z
2	ANDWF f,d	Operación lógica, AND. Así W AND Reg F	Z
3	CLRF F	Limpia (poner ceros) el registro f	Z
4	CLRW	Clarea el registro W	Z
5	COMF f,d	Complemento del registro f	Z
6	DECF f,d	Resta en uno el registro f	Z
7	DECFSZ f,d	Resta en uno el registro f y salta si es cero	
8	INCF f,d	Suma uno al registro f	Z
9	INCFSZ f,d	Suma uno al registro f y salta si es cero	
10	IORWF f,d	Operación lógica, OR. Así W OR Reg F	Z
11	MOVF f,d	Copia el registro f	Z
12	MOVWF F	Copia el registro W al registro f	
13	NOP	No hace nada	
14	RLF f,d	Rota a la izquierda el registro f con carry	C
15	RRF f,d	Rota a la dercha el registro f con carry	C
16	SUBWF f,d	Operación matemática, RESTA. Así Reg F – W	C, DC, Z
17	SWAPF f,d	Intercambia los nibbles del registro f	

18	XORWF	f,d	Operación lógica, OR Exclusiva. Así $W \oplus \text{Reg F}$	Z
Segundo grupo operaciones de manipulación y prueba de bits				
19	BCF	f,b	Pone cero (Clear) en un bit (b) del registro f	
20	BSF	f,b	Pone uno (Set) en un bit (b) del registro f	
21	BTFS	f,b	Pregunta por un bit (b) del registro f y salta si es cero (Clear)	
22	BTFS	f,b	Pregunta por un bit (b) del registro f y salta si es uno (Set)	
Tercer grupo operaciones con constantes y de control				
23	ADDLW	k	Operación matemática, SUMA. Así $W + k = W$. (k es un número)	C. DC. Z
24	ANDLW	k	Operación lógica, AND. Así $W \text{ AND } k = W$. (k es un número)	Z
25	CALL	k	Llama a un subrutina (k es una etiqueta)	
26	CLRWD		Clarea el WDT (Watchdog Timer)	$\overline{TO} \overline{PD}$
27	GOTO	k	Salta a la dirección. (k es una etiqueta)	
28	IORLW	k	Operación lógica, OR. Así $W \text{ OR } k = W$. (k es un número)	Z
29	MOVLW	k	Mueve la constante k a W. (k es un número)	
30	RETFIE		Retorna de una interrupción	
31	RETLW	k	Retorna de una subrutina con la constante k en W. (k es un número)	
32	RETURN		Retorna de una subrutina	
33	SLEEP		El microcontrolador va a bajo consumo. (En espera)	$\overline{TO} \overline{PD}$
34	SUBLW	k	Operación matemática, RESTA. Así $k - W = W$. (k es un número)	C. DC. Z
35	XORLW	k	Operación lógica, OR Exclusiva. Así $W \text{ XOR } k = W$ (k es un número)	Z

Fuente. (Microchip, 2007, p. 116)

- **ADDWF** **f,d**

Según el fabricante (Microchip, 2007, p. 117) esta instrucción suma lo que tiene el registro **W** con el registro **f (Reg f)**, y la respuesta puede quedar en el registro **W** o en el registro **f (Reg f)**, así:

$$W + \text{Reg f} = \begin{cases} W & \rightarrow d = 0 \\ \text{Reg f} & \rightarrow d = 1 \end{cases}$$

La instrucción afecta los bits **Z**, **DC**, y **C** del registro STATUS (ver sección **3.3.1**).

Ejemplo instrucción ADDWF **f,d**

Se desea sumar el registro **W** con el registro **Reg B** y la respuesta de la suma dejarla en el **Reg B** así:

$$W + \text{Reg B} = \text{Reg B}$$

Al realizar la suma de los registros **W** y **Reg B** se suma $D_H + F_H = C_H$ (se suma el *nibble* de menor peso de cada registro respectivamente), produce 1 en el bit 4 (**Carry DC**, bit 1 del registro STATUS, ver sección **3.3.1**). Luego se suma $1_H + C_H + 7_H = 4_H$ (se suma el *nibble* de mayor peso de cada registro respectivamente más el **Carry DC**), produce 1 en el bit 8 (**Carry C**, bit 0 del registro STATUS, ver sección **3.3.1**). Por lo tanto, la respuesta queda en 9 bits, así:

$$\begin{array}{rcl} \text{Carry C (Bit 0 del registro STATUS)} \rightarrow & 1 & \\ & 1 & \leftarrow \text{Carry DC (Bit 1 del registro STATUS)} \\ & CD_H & \\ + & 7F_H & \\ \hline & 1 & \\ & 4C_H & \end{array}$$

$$\begin{array}{rcl} \text{Carry C (Bit 0 del registro STATUS)} \rightarrow & 1 & \\ & 0100 & \\ & 1100_2 & \\ & \underbrace{\hspace{2.5cm}} & \\ & 9 \text{ bits} & \end{array}$$

La instrucción se escribe de la siguiente forma:

$$\text{ADDWF} \quad \text{RegB},1$$

Recuerde que el **1** (uno) significa que la respuesta de la instrucción queda en **Reg B**, así lo explica el ejemplo. Antes de ejecutarse la instrucción los registros tienen:

$$\begin{array}{l} W = CD_H \\ \text{Reg A} = 8E_H \\ \text{Reg B} = 7F_H \end{array}$$

Después de ejecutarse la instrucción; los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

$$\begin{array}{l} W = CD_H \\ \text{Reg A} = 8E_H \\ \text{Reg B} = 4C_H \end{array}$$

En este caso el bit C (bit 0 del registro STATUS, ver sección 3.3.1) está en 1, pues se produjo carry en el bit 8.

En este caso el bit DC (bit 1 del registro STATUS, ver sección 3.3.1) está en 1, pues se produjo carry en el bit 4.

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 0, pues ningún registro quedó en 0.

- **ANDWF** *f,d*

Esta instrucción hace la operación lógica **AND** bit a bit entre el registro **W** y el registro **f** y la respuesta queda en **W** o en **Reg f** así:

$$W^{AND} Reg f = \begin{cases} W & \rightarrow d = 0 \\ Reg f & \rightarrow d = 1 \end{cases}$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción **ANDWF** *f,d*

Se desea hacer la operación lógica **AND** entre el registro **Reg W** y el registro **Reg A** y la respuesta de la operación lógica se deja en el **Reg A** así:

$$W \cdot Reg A = Reg A$$

Al realizar la operación lógica **AND** entre los registros **Reg W** y **Reg A** (se hace la operación lógica **AND** bit a bit entre los 8 bits de los 2 registros) así:

	Hex	Bin
Reg W →	CD _H =	1100 1101 ₂
Reg A →	8E _H =	1000 1110 ₂
Respuesta	8C _H =	1000 1100 ₂

La instrucción se escribe de la siguiente forma:

$$ANDWF \quad RegA,1$$

Recuerde que el **1** (uno) significa que la respuesta de la instrucción queda en **Reg A**, así lo muestra en el ejemplo. Antes de ejecutarse la instrucción los registros tienen:

$$W = CD_H$$

$$Reg A = 8E_H$$

$$\mathbf{Reg\ B} = 7F_H$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

$$\mathbf{W} = CD_H$$

$$\mathbf{Reg\ A} = 8C_H$$

$$\mathbf{Reg\ B} = 7F_H$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 0, pues ningún registro quedó en 0.

- **CLRF *f***

Esta instrucción limpia el registro **f**. Es decir, lo pone en 00_H , así:

$$00 = \mathbf{Reg\ f}$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción **CLRF *f***

Se desea limpiar, poner en 0 todos los bits, del **Reg B**:

$$00_H = \mathbf{Reg\ B}$$

La instrucción se escribe de la siguiente forma:

$$\mathbf{CLRF\ \textit{RegB}}$$

Antes de ejecutarse la instrucción los registros tienen:

$$\mathbf{W} = CD_H$$

$$\mathbf{Reg\ A} = 8E_H$$

$$\mathbf{Reg\ B} = 7F_H$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

$$\mathbf{W} = CD_H$$

$$\mathbf{Reg\ A} = 8E_H$$

$$\mathbf{Reg\ B} = 00_H$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 1 pues un registro quedó en 0.

- **CLRW**

Esta instrucción limpia el registro **W**; lo pone en 00_H , así:

$$00_H = W$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción CLRW

Se desea limpiar del **Reg W**:

$$00_H = \text{Reg W}$$

La instrucción se escribe de la siguiente forma:

CLRW

Antes de ejecutarse la instrucción los registros tienen:

$$W = CD_H$$

$$\text{Reg A} = 8E_H$$

$$\text{Reg B} = 7F_H$$

Después de ejecutarse la instrucción los registros **Reg A** y **Reg B** no se ven afectados, en cambio **Reg W** sí:

$$W = 00_H$$

$$\text{Reg A} = 8E_H$$

$$\text{Reg B} = 7F_H$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 1 pues un registro quedó en 0.

- **COMF f,d**

Esta instrucción halla el complemento a 1 (cambiar los **cero** por **unos** y los **unos** por **ceros**) del registro **f** y la respuesta puede quedar en **W** o en **Reg f** así:

$$f = \text{complemento de Reg } f \left\{ \begin{array}{l} W \rightarrow d = 0 \\ \text{Reg } f \rightarrow d = 1 \end{array} \right.$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción **COMF** *f,d*

Se desea hallar el complemento a uno del **Reg B** y la respuesta se deja en el mismo registro:

$$\overline{\text{Reg B}} = \text{Reg B}$$

Al realizar el complemento a uno del registro B se tiene:

	Hex	Bin
Reg B →	7F _H =	0111 1111 ₂
$\overline{\text{Reg B}}$ →	80 _H =	1000 0000 ₂

(Complemento a uno del **Reg B**)

La instrucción se escribe de la siguiente forma:

$$\text{COMF } \text{Reg B}, 1$$

Antes de ejecutarse la instrucción los registros tienen:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg A} = 8\text{E}_{\text{H}}$$

$$\text{Reg B} = 7\text{F}_{\text{H}}$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg A} = 8\text{E}_{\text{H}}$$

$$\text{Reg B} = 80_{\text{H}}$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 0 pues ningún registro quedó en 0.

- **DECF** *f,d*

Esta instrucción decrementa el registro **f** (resta en uno, $f-1$) y la respuesta puede quedar en **W** o en **Reg f** así:

$$f = \text{Reg } f-1 \quad \left\{ \begin{array}{l} W \rightarrow d = 0 \\ \text{Reg } f \rightarrow d = 1 \end{array} \right.$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción *DECf f,d*

Se desea restar 1 al **Reg A** y la respuesta se deja en el mismo registro así:

$$\mathbf{Reg\ A} - 1 = \mathbf{Reg\ A}$$

Al restar 1 del registro A se tiene:

	Hex
Reg A →	$8E_H - 1_H = 8D_H$

La instrucción se escribe de la siguiente forma:

DECf **RegA,1**

Antes de ejecutarse la instrucción los registros tienen:

$$\mathbf{W} = \mathbf{CD}_H$$

$$\mathbf{Reg\ A} = \mathbf{8E}_H$$

$$\mathbf{Reg\ B} = \mathbf{7F}_H$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

$$\mathbf{W} = \mathbf{CD}_H$$

$$\mathbf{Reg\ A} = \mathbf{8D}_H$$

$$\mathbf{Reg\ B} = \mathbf{80}_H$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en cero, pues ningún registro quedó en cero.

- **DECFSZ** **f,d**

Esta instrucción decrementa el registro f (resta en 1) y la respuesta puede quedar en **W** o en **Reg f** así:

$$\mathbf{f} = \mathbf{Reg\ f} - 1 \quad \left\{ \begin{array}{l} \mathbf{W} \rightarrow \mathbf{d} = 0 \\ \mathbf{Reg\ f} \rightarrow \mathbf{d} = 1 \end{array} \right.$$

La instrucción pregunta por el bit **Z** del STATUS. Si al ejecutar la instrucción el registro f queda en 0, se salta una instrucción y, si no es 0, no salta.

Ejemplo instrucción *DECFSZ f,d*

Se desea restar uno al **Reg A**, dejar la respuesta en el mismo y preguntar si este registro es 0. Para entender mejor cómo funciona la instrucción en la Figura 13A se muestra una parte del diagrama de flujo y la Figura 13B ilustra cómo la ejecutaría el microcontrolador.

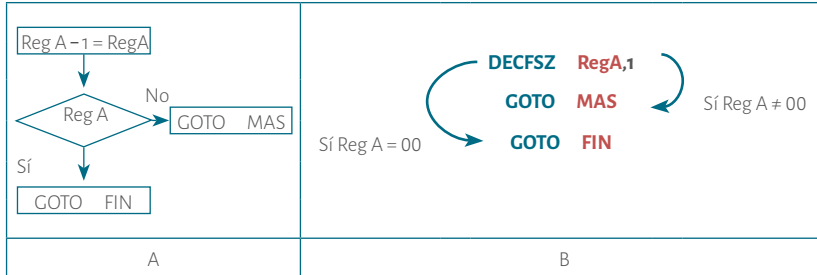


Figura 13. Ilustración de la instrucción **DECFSZ** *f,d*

La instrucción se escribe de la siguiente forma:

DECFSZ *RegA,1*

Antes de ejecutarse la instrucción los registros tienen:

W = CD_H

Reg A = 8E_H

Reg B = 7F_H

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

W = CD_H

Reg A = 8D_H

Reg B = 7F_H

En este caso el bit Z (bit 2 del registro STATUS, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

- **INCF** *f,d*

Esta instrucción incrementa el registro **f** (suma en 1) y la respuesta puede quedar en **W** o en **Reg f** así:

$$\mathbf{Reg\ f = Reg\ f + 1} \begin{cases} W & \rightarrow d = 0 \\ \mathbf{Reg\ f} & \rightarrow d = 1 \end{cases}$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción **INCF f,d**

Se desea sumar 1 al **Reg B** y la respuesta se deja en el mismo registro así:

$$\mathbf{Reg\ B + 1 = Reg\ B}$$

Al restar 1 del registro B se obtiene:

	Hex
Reg B →	$7F_H + 1_H = 80_H$

La instrucción se escribe de la siguiente forma:

$$\mathbf{DECf\ \ \ \ \ \ RegB,1}$$

Antes de ejecutarse la instrucción los registros tienen:

$$\mathbf{W} = CD_H$$

$$\mathbf{Reg\ A} = 8E_H$$

$$\mathbf{Reg\ B} = 7F_H$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

$$\mathbf{W} = CD_H$$

$$\mathbf{Reg\ A} = 8E_H$$

$$\mathbf{Reg\ B} = 80_H$$

En este caso el bit Z (bit 2 del Registro STATUS, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

- **INCFSZ f,d**

Esta instrucción incrementa el registro f (suma en 1). La respuesta puede quedar en **W** o en **Reg f** así:

$$\mathbf{Reg\ f = Reg\ f + 1} \begin{cases} W & \rightarrow d = 0 \\ \mathbf{Reg\ f} & \rightarrow d = 1 \end{cases}$$

La instrucción pregunta por el bit **Z** del STATUS. Si al ejecutar la instrucción el registro *f* queda en 0, se salta una instrucción y, si no es 0, no salta.

Ejemplo instrucción **INCFSZ** *f,d*

Se desea sumar 1 al **Reg A**, dejar la respuesta en el mismo registro y preguntar si este registro es 0. Para entender mejor cómo funciona la instrucción en la Figura 14A se muestra una parte del diagrama de flujo y en la Figura 14B ilustra cómo la ejecutaría el microcontrolador.

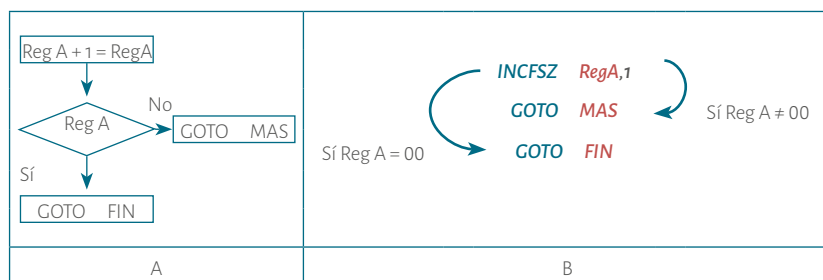


Figura 14. Ilustración de la instrucción **INCFSZ** *f,d*

La instrucción se escribe de la siguiente forma:

INCFSZ *RegA,1*

Antes de ejecutarse la instrucción los registros tienen:

W = CD_H

Reg A = $8E_H$

Reg B = $7F_H$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

W = CD_H

Reg A = $8F_H$

Reg B = $7F_H$

En este caso el bit Z (bit 2 del registro STATUS, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

- **IORWF** *f,d*

Esta instrucción hace la operación lógica **OR** bit a bit entre el registro **W** y el registro **f** y la respuesta puede quedar en **W** o en **Reg f** así:

$$W \text{ or } \text{Reg } f = \begin{cases} W & \rightarrow d = 0 \\ \text{Reg } f & \rightarrow d = 1 \end{cases}$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción **IORWF f,d**

Se desea hacer la operación lógica **OR** entre el registro **Reg W** con el registro **Reg A** y la respuesta de la operación lógica dejarla en el **Reg A** así:

$$W + \text{Reg } A = \text{Reg } A$$

Al realizar la operación lógica **OR** entre los registros **Reg W** y **Reg A**, se hace la operación lógica **OR** bit a bit entre los 8 bits de los 2 registros :

	Hex	Bin
Reg W →	CD _H =	1100 1101 ₂
Reg A →	8E _H =	1000 1110 ₂
Respuesta	CF _H =	1100 1111 ₂

La instrucción se escribe de la siguiente forma:

$$\text{IORWF } \text{Reg } A, 1$$

Antes de ejecutarse la instrucción los registros tienen:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg } A = 8\text{E}_{\text{H}}$$

$$\text{Reg } B = 7\text{F}_{\text{H}}$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

$$W = \text{CD}_{\text{H}}$$

$$\mathbf{Reg A} = \mathbf{CF_H}$$

$$\mathbf{Reg B} = \mathbf{7F_H}$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0. **MOVF *f,d***

Esta instrucción mueve (copia) lo que tiene el registro ***f*** a ***W*** o al mismo registro ***f*** así:

$$\mathbf{Reg f} = \begin{cases} \mathbf{W} & \rightarrow d = 0 \\ \mathbf{Reg f} & \rightarrow d = 1 \end{cases}$$

La instrucción afecta el bit **Z** del registro STATUS.

Ejemplo instrucción **MOVF *f,d***

Se desea copiar lo que tiene el **Reg A** al **Reg W** así:

$$\mathbf{Reg A} = \mathbf{Reg W}$$

La instrucción se escribe de la siguiente forma:

$$\mathbf{MOVF \quad RegA,0}$$

Recuerde que el **0** (cero) significa que la respuesta de la instrucción queda en **Reg W**, como lo muestra el ejemplo.

Antes de ejecutarse la instrucción los registros tienen:

$$\mathbf{W} = \mathbf{CD_H}$$

$$\mathbf{Reg A} = \mathbf{8E_H}$$

$$\mathbf{Reg B} = \mathbf{7F_H}$$

Después de ejecutarse la instrucción los registros **Reg A** y **Reg B** no se ven afectados, en cambio **Reg W** sí:

$$\mathbf{W} = \mathbf{8E_H}$$

$$\mathbf{Reg A} = \mathbf{8E_H}$$

$$\mathbf{Reg B} = \mathbf{7F_H}$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

- **MOVWF *f***

Esta instrucción mueve (copia) lo que tiene el registro **W** al registro **f** así:

$$W = \text{Reg } f$$

La instrucción no afecta ningún bit del registro STATUS.

Ejemplo instrucción **MOVWF f**

Se desea copiar lo que tiene el **Reg W** al **Reg B** así:

$$\text{Reg } W = \text{Reg } B$$

La instrucción se escribe de la siguiente forma:

MOVWF RegB

Antes de ejecutarse la instrucción los registros tienen:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg } A = 8\text{E}_{\text{H}}$$

$$\text{Reg } B = 7\text{F}_{\text{H}}$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg } A = 8\text{E}_{\text{H}}$$

$$\text{Reg } B = \text{CD}_{\text{H}}$$

En este caso el bit Z (bit 2 del Registro STATUS, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

- **NOP**

Esta instrucción no hace nada, no afecta ningún registro. Este microcontrolador cuenta con un oscilador interno el cual funciona a una frecuencia de 4 Mega Hertz (4Mhz) y, a esta velocidad, el microcontrolador ejecuta la instrucción en 1 micro segundo (1μS) (Microchip, 2007). Por ello, es muy usada para sincronismos. En consecuencia, la instrucción no afecta ningún bit del registro STATUS.

- **RLF f,d**

Esta instrucción toma el registro **f** y lo rota la izquierda (corrimiento bit a bit) con el **Carry** y la respuesta puede quedar en **W** o en **Reg f** así:

$\text{Reg } f = \text{rotar a la izquierda registro } f \begin{cases} W \rightarrow d = 0 \\ \text{Reg } f \rightarrow d = 1 \end{cases}$

La instrucción afecta el bit **C** del registro STATUS. En la Figura 15 se muestra cómo se hace la rotación.



Figura 15. Ilustración de la instrucción **RLF** *f,d*

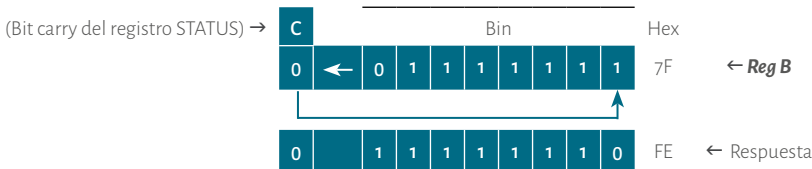
Fuente: elaborado por José Luis González, imagen adaptada de Microchip, 2007, p. 125.

Ejemplo instrucción RLF *f,d*

Se desea hacer la rotación a la izquierda del **Reg B** y la respuesta se deja en el mismo **Reg B** así:

$$\text{Reg B (Rotar a la izquierda)} = \text{Reg B}$$

Al hacer la rotación a la izquierda bit a bit del **Reg B** junto con el bit C del registro STATUS, (se asume que está en 0). En la Figura 15 se puede observar que el bit 7 del registro *f* (para este caso **Reg B**) pasa al bit C del registro STATUS y éste, a su vez, pasa al bit 0 del mismo registro así:



La instrucción se escribe de la siguiente forma:

RLF *RegB,1*

Antes de ejecutarse la instrucción los registros tienen:

$$W = CD_H$$

$$\text{Reg A} = 8E_H$$

$$\text{Reg B} = 7F_H$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

$$W = CD_H$$

$$\text{Reg A} = 8E_H$$

$$\text{Reg B} = FE_H$$

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 0, pues al hacer la rotación el bit 7 del **Reg B** era 0.

- **RRF** *f,d*

Esta instrucción toma el registro *f* y lo rota la derecha con el **Carry** y la respuesta puede quedar en **W** o en **Reg f** así:

$$\text{Reg } f = \text{Rotar a la derecha registro } f \begin{cases} W & \rightarrow d = 0 \\ \text{Reg } f & \rightarrow d = 1 \end{cases}$$

La instrucción afecta el bit **C** del registro STATUS. En la Figura 16 se muestra cómo se hace la rotación.



Figura 16. Ilustración de la instrucción **RRF** *f,d*

Fuente. Fuente. Elaborado por José Luis González, imagen adaptada de Microchip, 2007, p 126.

Ejemplo instrucción **RRF** *f,d*

Se desea hacer la rotación a la derecha del **Reg B** y la respuesta se deja en el mismo **Reg B** así:

Reg B (Rotar a la derecha) = **Reg B**

Al hacer la rotación a la derecha bit a bit del **Reg B** junto con el bit C del registro STATUS, (se asume que está en 0). En la Figura 16 se puede observar que el bit 0 del registro *f* (para este caso **Reg B**) pasa al bit C del registro STATUS y éste, a su vez, pasa al bit 7 del mismo registro:



1 0 0 1 1 1 1 1 1 3F ← Respuesta

La instrucción se escribe de la siguiente forma:

RRF *RegB*,1

Antes de ejecutarse la instrucción los registros tienen:

W = CD_H

Reg A = 8E_H

Reg B = 7F_H

Después de ejecutarse la instrucción los registros **Reg W** y **Reg A** no se ven afectados, en cambio **Reg B** sí:

W = CD_H

Reg A = 8E_H

Reg B = 3F_H

En este caso el bit Z (bit 2 del registro STATUS, ver sección 3.3.1) está en 1, pues al hacer la rotación el bit 0 del **Reg B** era 1.

- **SUBWF** *f*,*d*

Esta instrucción resta lo que tiene el registro **W** del registro **f** y la respuesta puede quedar en **W** o en **Reg f** así:

$$\text{Reg } f - W = \begin{cases} W & \rightarrow d = 0 \\ \text{Reg } f & \rightarrow d = 1 \end{cases}$$

La instrucción afecta los bits **Z**, **DC** y **C** del registro STATUS. Hay tres formas de respuesta que se muestran en la tabla.

Tabla 12.
Forma de respuesta de la instrucción **SUBWF** *f*,*d*.

Cuando	Z	C	Respuesta
$f > W$	0	1	Positiva
$f = W$	1	1	Cero
$f < w$	0	0	Negativa

Fuente. Elaboración propia, adaptado de Microchip, 2007

Ejemplo instrucción `SUBWF` *f,d*

Se desea restar el registro **W** del registro **Reg B** y la respuesta de la suma dejarla en el **Reg W** así:

$$\text{Reg B} - \text{W} = \text{W}$$

Al restar el registro **W** del registro **Reg B** se resta $F_H - D_H = 2_H$ (se resta el *nibble* de menor peso de cada registro respectivamente). Ahora se resta $8_H - 5_H = 3_H$ (se resta el *nibble* de mayor peso de cada registro respectivamente):

$$\begin{array}{r} 8F_H \\ - 5D_H \\ \hline 32_H \end{array}$$

La instrucción se escribe de la siguiente forma:

`SUBWF` *RegB,0*

Recuerde que el 0 (cero) significa que la respuesta de la instrucción queda en el registro **W**, así lo solicitó el ejemplo. Antes de ejecutarse la instrucción los registros tienen:

$$\text{W} = 5D_H$$

$$\text{Reg A} = 9E_H$$

$$\text{Reg B} = 8F_H$$

Después de ejecutarse la instrucción los registros **Reg A** y **Reg B** no se ven afectados, en cambio **Reg W** sí:

$$\text{W} = 32_H$$

$$\text{Reg A} = 9E_H$$

$$\text{Reg B} = 8F_H$$

En este caso el bit Z está en 0 y el bit C está en 1, indicando que la respuesta es positiva según la Tabla 12.

- **`SWAPF` *f,d***

Esta instrucción toma el *nibble* de mayor peso (bits 4 al 7) del registro **f** y lo intercambia con el *nibble* de menor peso (bits 0 al 3) del registro **f**. La respuesta puede quedar en **W** o en **Reg f**, así como se muestra en la Figura 17.



Figura 17. Ilustración de la instrucción SWAPF f,d

Fuente. (Microchip, 2007, 127)

La instrucción no afecta ningún bit del registro STATUS.

Ejemplo instrucción SWAPF f,d

Se desea intercambiar los *nibbles* del registro **Reg A** y la respuesta dejarla en el **Reg A**. Entonces la instrucción intercambia el *nibble* de mayor peso (9) con el *nibble* de menor peso (E) del **Reg A**:

Nibble de mayor peso \rightarrow $9E_H$ \leftarrow Nibble de mayor peso
 $E9_H$

La instrucción se escribe de la siguiente forma:

SWAPF RegA,1

Recuerde que el 1 significa que la respuesta de la instrucción queda en el registro **A**, así se explica en el ejemplo.

Antes de ejecutarse la instrucción los registros tienen:

W = $5D_H$

Reg A = $9E_H$

Reg B = $8F_H$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

W = $5D_H$

Reg A = $E9_H$

Reg B = $8F_H$

La instrucción no afectó ningún bit del registro **STATUS**.

- **XORWF f,d**

Esta instrucción hace la operación lógica **XOR** bit a bit entre el registro **W** y el registro **f**. La respuesta puede quedar en **W** o en **Reg f** así:

$$W \oplus \text{Reg } f = \begin{cases} W & \rightarrow d = 0 \\ \text{Reg } f & \rightarrow d = 1 \end{cases}$$

La instrucción afecta el bit **Z** del registro **STATUS**.

Ejemplo instrucción **XORWF f,d**

Se desea hacer la operación lógica **XOR** entre el registro **Reg W** y el registro **Reg A**. La respuesta de la operación lógica se deja en el **Reg A** así:

$$W \oplus \text{Reg } A = \text{Reg } A$$

Al realizar la operación lógica **XOR** entre los registros **Reg W** y **Reg A** se hace la operación lógica **XOR** bit a bit entre los 2 bits de los 2 registros así:

	Hex	Bin
Reg W →	CD _H =	1100 1101 ₂
Reg A →	8E _H =	1000 1110 ₂
Respuesta	43 _H =	0100 0011 ₂

La instrucción se escribe de la siguiente forma:

XORWF RegA,1

Recuerde que el **1** (uno) significa que la respuesta de la instrucción queda en **Reg A**, así se muestra en el ejemplo. Antes de ejecutarse la instrucción los registros tienen:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg } A = \text{8E}_{\text{H}}$$

$$\text{Reg } B = \text{7F}_{\text{H}}$$

Después de ejecutarse la instrucción los registros **Reg W** y **Reg B** no se ven afectados, en cambio **Reg A** sí:

$$W = \text{CD}_{\text{H}}$$

$$\text{Reg } A = \text{43}_{\text{H}}$$

$$\text{Reg } B = \text{7F}_{\text{H}}$$

En este caso el bit Z (bit 2 del registro **STATUS**, ver sección 3.3.1) está en 0, pues ningún registro quedó en 0.

3.3.2.2 Instrucciones de manipulación y prueba de bits. En estas instrucciones se encuentra la letra **f**, la cual se refiere a cualquier registro, ya sea de propósito específico o de propósito general. También está la letra **b** que representa cualquier bit del 0 al 7.

- **BCF** **f,b**

Esta instrucción toma un bit (0 al 7 que lo representa la letra **b**) del registro **f** y lo pone en cero.

Ejemplo instrucción **BCF** **f,b**

Se desea dejar en 0 (cero) el bit 5 (cinco) del registro **STATUS**. Para el ejemplo se asumirá que el estado de los bits del registro es de la siguiente forma:

Registro STATUS							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
0	1	1	0	0	0	1	1

↑

Bit que será afectado por la instrucción

En este caso el bit 5 (cinco) del registro **STATUS** está 1 (Uno). Al realizar la instrucción este bit queda en 0 (cero).

La instrucción se escribe de la siguiente forma:

BCF **Status,5**

Recuerde que el **5** (cinco) significa que este bit es el afectado por la instrucción, así se aclara en el ejemplo.

Al ejecutarse la instrucción el único bit del registro **STATUS** que se ve afectado es el bit 5:

Registro STATUS							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
0	1	0	0	0	0	1	1

↑

Bit afectado por la instrucción

Después de ejecutarse la instrucción ningún otro registro se ve afectado.

- **BSF** *f,b*

Esta instrucción toma un bit (0 - 7 que lo representa *b*) del registro *f* y lo pone en uno.

Ejemplo instrucción **BSF** *f,b*

Se desea dejar en 1 (uno) el bit 5 (cinco) del registro **STATUS**. Para el ejemplo se asumirá que el estado de los bits del registro es de la siguiente forma:

Registro STATUS							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
0	0	1	0	0	0	1	0

↑

Bit que será afectado por la instrucción

En este caso el bit 6 (seis) del Registro **STATUS** está 0 (cero). Al realizar instrucción este bit queda en 1 (uno).

La instrucción se escribe de la siguiente forma:

BSF *Status,6*

Recuerde que el 6 significa que este bit es el afectado en la instrucción. Así se expone en el ejemplo. Al ejecutarse la instrucción el único bit que se ve afectado es el bit cero:

Registro STATUS							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C
0	1	1	0	0	0	1	0

↑

Bit afectado por la instrucción

Después de ejecutarse la instrucción ningún otro registro se ve afectado.

- **BTFS** *f,b*

Esta instrucción toma un bit (0 a 7 que lo representa **b**) del registro **f** y pregunta si es 0, si lo es, salta una instrucción. De lo contrario, no salta. En la Figura 18A se muestra la parte de un diagrama de flujo y en la Figura 18B se ilustra cómo la ejecutaría el microcontrolador.

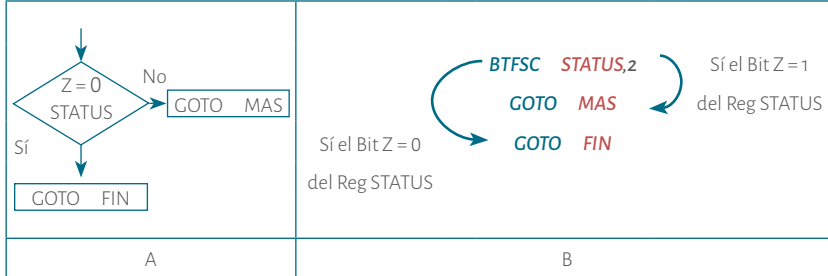


Figura 18. Ilustración de la instrucción **BTFSZ f,b**

Ejemplo instrucción **BTFSZ f,b**

Evaluar si en bit 2 (dos, bit Z) del registro **STATUS** está en 0 (cero).

La instrucción se escribe de la siguiente forma:

BTFSZ Status,2

Al ejecutarse la instrucción se evalúa si el bit 2 (bit Z) del registro **STATUS** está en 0 (cero). El lector puede observar la Figura 18B donde se muestra que si este bit está en 0 (Cero) el microcontrolador salta una instrucción. Es decir que el microcontrolador ejecutaría la instrucción **GOTO FIN** y si este bit está en 1 no salta una instrucción. Esto significa que se ejecutaría la instrucción **GOTO MAS**.

- **BTFSZ f,b**

Esta instrucción toma un bit (0 a 7 que lo representa **b**) del registro **f** y pregunta si es 1. Si lo es, salta una instrucción. De lo contrario, no salta. En la Figura 19A se muestra una parte de un diagrama de flujo y en la Figura 19B se ilustra como la ejecutaría el microcontrolador.

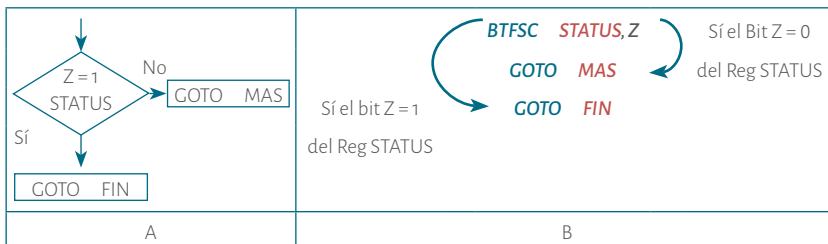


Figura 19. Ilustración de la instrucción **BTFS** *f,b*

Ejemplo instrucción BTFS *f,b*

Evaluar si en bit 2 (dos, bit Z) del registro **STATUS** esta en 0 (cero).

La instrucción se escribe de la siguiente forma:

BTFS *Status,2*

Al ejecutarse la instrucción se evalúa si el bit 2 (bit Z) del registro **STATUS** está en 1. El lector puede observar la Figura 19B donde se muestra que si este bit está en 1 el microcontrolador salta una instrucción. Es decir que se ejecutaría la instrucción **GOTO** *FIN*. Si este bit está en 0 no salta una instrucción y, por lo tanto, se ejecutaría la instrucción **GOTO** *MAS*.

3.3.2.3 Instrucciones de operaciones con constantes y de control. En estas instrucciones se encuentra la letra *k* que en algunas instrucciones representa un constante (un número, ya sea hexadecimal de **00_H** a **FF_H** o binario de **0000 0000** a **1111 1111** o decimal de **0** a **255**); en otros representa una dirección de una subrutina o de una etiqueta que exista en el programa.

- **ADDLW** *k*

Esta instrucción suma lo que tiene el registro **W** con la constante *k* y la respuesta queda en **W** así:

$$W + k = W$$

La instrucción afecta los bits **Z**, **DC** y **C** del registro **STATUS**.

Ejemplo instrucción ADDLW *k*

Se desea sumar el registro **W** con la constante **7F_H** así:

$$W + 7F_H = W$$

Al realizar la suma del registro **W** y la constante **7F_H**, se suma $D_H + F_H = C_H$ (se suma el *nibble* de menor peso de cada uno respectivamente), produce 1 en el bit 4 (**Carry DC**, bit 1 del Registro STATUS, ver sección 3.3.1). Posteriormente, se suma $1_H + C_H + 7_H = 4_H$ (se suma el *nibble* de mayor peso de cada uno respectivamente más el **Carry DC**) y produce 1 en el bit 8 (**Carry C**, Bit 0 del Registro STATUS, ver sección 3.3.1):

Carry C (bit 0 del registro STATUS) → 1 1 ← **Carry DC** (bit 1 del registro STATUS)

$$\begin{array}{r} \text{CD}_H \\ + \text{7F}_H \\ \hline 1 \text{4C}_H \end{array}$$

La instrucción se escribe de la siguiente forma:

ADDLW 0X7F

Recuerde que esta instrucción deja la respuesta en el registro **W**. Es decir, que solo afecta a éste.

Antes de ejecutarse la instrucción se tiene:

$$W = \text{CD}_H$$

$$\text{Constante} = \text{7F}_H$$

Después de ejecutarse la instrucción, el registro **Reg W** se ve afectado:

$$W = \text{4C}_H$$

En este caso el bit C (bit 0 del registro **STATUS**, ver sección 3.3.1) está en 1, pues se produjo *carry* en el bit 8. En este caso el bit DC (bit 1 del registro **STATUS**, ver sección 3.3.1) está en 1, pues se produjo *carry* en el bit 4.

En este caso el bit Z (bit 2 del registro **STATUS**, ver sección 3.3.1) está en 0, pues ningún registro quedó en 0.

- **ANDLW k**

Esta instrucción hace la operación lógica **AND** bit a bit entre el registro **W** y la constante **k** y la respuesta queda en **W** así:

$$W \text{ k} = W$$

La instrucción afecta el bit **Z** del registro **STATUS**.

Ejemplo instrucción ANDLW k

Se desea hacer la operación lógica **AND** entre el registro **Reg W** y la constante **8E_H**:

$$W \text{ 8E}_H = W$$

Al realizar la operación lógica **AND** entre el registro **Reg W** y **8E_H**. Se hace la operación lógica **AND** bit a bit entre los 8 bits:

	Hex	Bin
Reg W →	CD _H =	1100 1101 ₂
Constante →	8E _H =	1000 1110 ₂
Respuesta	8C _H =	1000 1100 ₂

La instrucción se escribe de la siguiente forma:

ANDLW 0X8E

Recuerde que esta instrucción deja la respuesta en el registro **W**. Es decir, que sólo afecta a éste.

Antes de ejecutarse la instrucción los registros tienen:

$$W = CD_H$$

Después de ejecutarse la instrucción el registro **Reg W** se ve afectado:

$$W = 8C_H$$

En este caso el bit Z (bit 2 del registro **STATUS**, ver sección 3.3.1) está en 0, pues ningún registro quedó en 0.

- **CALL k**

Esta instrucción hace un llamado de la subrutina **k**, puede ser cualquiera. En la Figura 20 se muestra un mapa de memoria donde se observa mejor el funcionamiento. Cuando se usa esta instrucción se hace necesario utilizar una instrucción de retorno al final de la subrutina. La instrucción no afecta ningún bit del registro **STATUS**.

PROGRAMA PRINCIPAL

MOTOR

TIEMPO

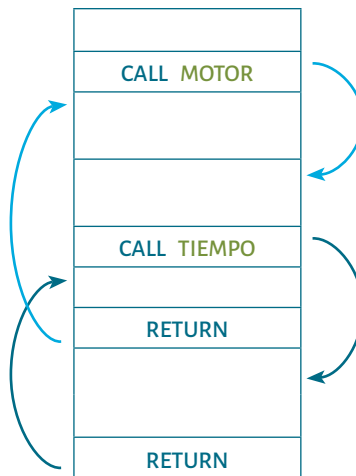


Figura 20. Ilustración de la instrucción **CALL** *k*

Ejemplo instrucción CALL *k*

Se desea ir a la subrutina **TIEMPO**.

La instrucción se escribe de la siguiente forma:

CALL **TIEMPO**

Esta instrucción no afecta ningún bit del registro **STATUS**.

- **CLRWDT**

Esta instrucción limpia el **WDT**. Es decir, lo pone en 00_H . La instrucción afecta los bit \overline{TO} y \overline{PD} (bit 4 y 3 del registro **STATUS**, ver sección 3.3.1).

- **GOTO** *k*

Esta instrucción hace un salto incondicional en el programa a la dirección *k*, puede ser cualquiera. Cuando se cita esta instrucción se hace necesario utilizar una instrucción igual al final de la subrutina. De lo contrario, el microcontrolador no regresa. Este procedimiento solo se realiza si es necesario. La instrucción no afecta ningún bit del registro **STATUS**.

Ejemplo instrucción GOTO *k*

Se desea ir a la subrutina **MAS**.

La instrucción se escribe de la siguiente forma:

CALL **MAS**

Esta instrucción no afecta ningún bit del registro **STATUS**.

- **IORLW** *k*

Esta instrucción hace la operación lógica **OR** bit a bit entre el registro **W** y la constante *k*. La respuesta queda en **W** así:

$$W + k = W$$

La instrucción afecta el bit **Z** del registro **STATUS**.

Ejemplo instrucción IORLW *k*

Se desea hacer la operación lógica **OR** entre el registro **Reg W** con la constante $8E_H$ así:

$$W + 8E_H = \text{Reg A}$$

Al realizar la operación lógica **OR** entre el registro **Reg W** y la constante $8E_H$ se hace la operación lógica **OR** bit a bit entre los 8 bits:

	Hex	Bin
Reg W →	$CD_H =$	$1100\ 1101_2$
Contante A →	$8E_H =$	$1000\ 1110_2$
Respuesta	$CF_H =$	$1100\ 1111_2$

La instrucción se escribe de la siguiente forma:

IORLW 0X8E

Recuerde que esta instrucción deja la respuesta en el registro **W**. Por lo tanto, sólo afecta éste.

Antes de ejecutarse la instrucción el registro **Reg W** tiene:

$$W = CD_H$$

Después de ejecutarse la instrucción el registro **Reg W** se ve afectado:

$$W = CF_H$$

En este caso el bit Z (bit 2 del registro **STATUS**, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

- **MOVLW k**

Esta instrucción carga (mueve) la constante **k**, en el registro **W**, así:

$$k = W$$

Esta constante puede ser hexadecimal, decimal o binaria. En la Tabla 13 se muestra un ejemplo de cómo puede realizar la carga del número 77_{10} al registro **Reg W**.

Tabla 13

Ilustración de la instrucción **MOVLW k**

Forma de uso de la instrucción

Explicación

MOVLW <i>0X4D</i>	Mueve el valor 4D en <i>hexadecimal</i> al registro W
MOVLW <i>.77</i>	Mueve el valor 77 en <i>decimal</i> al registro W
MOVLW <i>b'01001101'</i>	Mueve el valor 01001101 en <i>binario</i> al registro W
MOVLW <i>'M'</i>	Mueve el valor ASCII de la letra M (01001101) al registro W

La instrucción no afecta ningún bit del registro **STATUS**.

- **RETFIE**

Esta instrucción retorna de una interrupción hecha al microcontrolador, sin importar en qué parte del programa se halla hecho.

La instrucción no afecta ningún bit del registro **STATUS**.

- **RETLW** *k*

Esta instrucción retorna de una subrutina (hecha por la instrucción **CALL** *k*) y carga la constante *k* en el registro **W**. Esta instrucción es muy usada para hacer tablas.

La instrucción no afecta ningún bit del registro **STATUS**.

Ejemplo instrucción **RETLW** *k*

Se desea retornar el valor decimal 48 en el registro **Reg W**.

La instrucción se escribe de la siguiente forma:

IORLW *.48*

Después de ejecutarse la instrucción el registro **Reg W** se ve afectado:

$$W = 48_{10}$$

- **RETURN**

Esta instrucción retorna de una subrutina (hecha por la instrucción **CALL** *k*).

La instrucción no afecta ningún bit del registro **STATUS**.

- **SLEEP**

Esta instrucción pone en poco consumo al microcontrolador (*Standby*). Es decir, pasa a un estado de muy bajo consumo de corriente: de unos 100 nA a 2V. Por lo tanto, el dispositivo queda en espera a ser despertado. Este proceso se puede hacer por: **1)** entrada RESET externa en el pin MCLR, **2)** Watchdog Timer wake-up (si WDT estaba habilitado) y **3)** interrupción desde el pin RB0 / INT (Microchip, 2007, pp. 108-109).

La instrucción afecta los bits \overline{TO} y \overline{PD} (bit 4 y 3 del registro **STATUS**, ver sección 3.3.1).

- **SUBLW** *k*

Esta instrucción resta lo que tiene el registro **W** de la constante **k** y la respuesta queda en **W** así:

$$k - W = W$$

La instrucción afecta los bits **Z**, **DC** y **C** del registro **STATUS**. Hay tres formas de respuesta que se muestran en la Tabla 14.

Tabla 14.
Forma de respuesta de la instrucción **SUBLW** *k*.

Cuando	Z	C	Respuesta
$k > W$	0	1	Positiva
$k = W$	1	1	Cero
$k < w$	0	0	Negativa

Fuente. Elaboración propia, datos tomados de (Microchip, 2007, 126).

Ejemplo instrucción **SUBLW** *k*

Se desea restar la constante $8F_H$ menos el registro **Reg W**:

$$8F_H - W = W$$

Al restar el registro **Reg W** de la constante $8F_H$ se resta $F_H - D_H = 2_H$ (se resta el *nibble* de menor peso de cada dato respectivamente). Ahora se resta $8_H - 5_H = 3_H$ (se resta el *nibble* de mayor peso de cada dato respectivamente):

$$\begin{array}{r} 8F_H \\ - 5D_H \\ \hline 32_H \end{array}$$

La instrucción se escribe de la siguiente forma:

SUBLW **0X8F**

Recuerde que esta instrucción deja la respuesta en el registro **W** y sólo afecta a éste.

Antes de ejecutarse la instrucción el registro **Reg W** tiene:

$$W = 5D_H$$

Después de ejecutarse la instrucción el registro **Reg W** se ve afectado:

$$W = 32_H$$

En este caso el bit Z está en *cerro* y el bit C está en *uno*; indicando que la respuesta es positiva según la Tabla 14.

- **XORLW** *k*

Esta instrucción hace la operación lógica **XOR** bit a bit entre el registro **W** y la constante **k**. La respuesta queda en **W** así:

$$W \oplus k = W$$

La instrucción afecta el bit **Z** del registro **STATUS**.

Ejemplo instrucción **XORLW** *k*

Se desea hacer la operación lógica **XOR** entre el registro **Reg W** y la constante **8E_H** así:

$$W \oplus 8E_H = W$$

Al realizar la operación lógica **XOR** entre el registro **Reg W** y la constante **8E_H** se hace la operación lógica **XOR** bit a bit entre los 8 bits de los 2 datos :

	Hex	Bin
Reg W →	CD _H =	1100 1101 ₂
Constante →	8E _H =	1000 1110 ₂
Respuesta	43 _H =	0100 0011 ₂

La instrucción se escribe de la siguiente forma:

XORLW 0X8E

Recuerde que esta instrucción deja la respuesta en el registro **W** y sólo afecta éste.

Antes de ejecutarse la instrucción el registro **Reg W** tiene:

$$W = CD_H$$

Después de ejecutarse la instrucción el registro **Reg W** se ve afectado:

$$W = 43_H$$

En este caso el bit Z (bit 2 del registro **STATUS**, ver sección **3.3.1**) está en 0, pues ningún registro quedó en 0.

Una vez explicadas todas las 35 instrucciones con ejemplos, se desarrollarán varios programas como guías para aprender a usarlas y, de esa forma, el lector puede practicar y familiarizarse con este proceso para realizar sus propios programas y practicar según lo que requiera.

3.4. Diagramas de Flujo

Para empezar a desarrollar los diagramas de flujo hay que tener claro qué tarea se debe hacer, para ello se verán varios ejemplos para que el lector pueda posteriormente realizar los suyos.

Ejemplo 7

Hacer un diagrama de flujo para sumar dos números de 8 bits cada uno en el microcontrolador PIC 16F628A.

Solución:

A cada número se le asigna un registro. Entonces los registros se llamarán sumando 1 (S1) y sumando 2 (S2). La respuesta puede dar en 16 bits, así que hay una respuesta baja (RB) y respuesta alta (RA). Cada uno de estos registros son de 8 bits. El lector debe tener claro que se usan dos registros de 8 bits para la respuesta. Es decir, 16 bits, de los cuales solo se usan 9. 8 que bits están en RB y 1 bit en RA. Éste último es el *carry* de 8 bits. Para mayor claridad ver el ejemplo instrucción **ADDWF** *f.d.*

$$\begin{array}{r} S1 \\ + S2 \\ \hline RA RB \end{array}$$

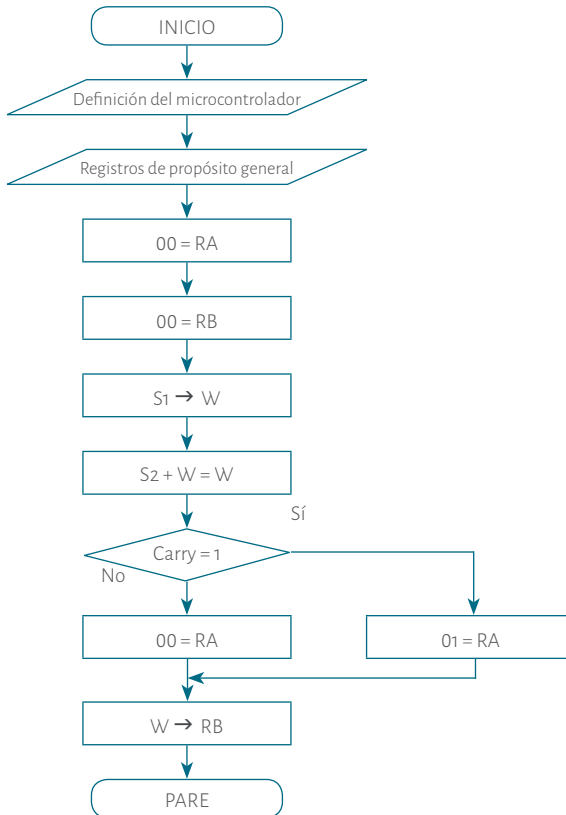
A continuación se plantea el diagrama de flujo (ver diagrama 1):

- (1) Hay que definir el microcontrolador con el que se va a trabajar,
- (2) Definir los registros necesarios (registros de propósito general),
- (3) Plantear la solución a la suma.

Como en los microcontroladores PIC el registro W es el principal, por lo tanto, se debe recuperar alguno de los sumandos a través de este registro para posteriormente sumarlo con el otro sumando. Una vez hecha la suma hay que guardar las respuestas; tanto la respuesta baja (RB) como respuesta la alta (RA).

Una vez realizado el diagrama de flujo hay que pasarlo a instrucciones. Para ello, se debe hacer un nuevo proyecto en MPLAB X IDE 3.65 o en otra versión superior. Este programa puede ser encontrado en internet en la siguiente página: <http://www.microchip.com>. Una vez abierto MPLAB X IDE es necesario que no tenga ningún proyecto abierto, para hacer este procedimiento se realiza lo siguiente: **File/Close Project.**

Diagrama 1
Programa que suma dos registros 8 bits.



Ahora va por: **File/New Project** (Ver Figura 21). En la ventana que se abre seleccionar **Microchip Embedded** y **Standalone Project** y dar clic en **Next**. Así como se muestra en la Figura 21B. En la ventana que se abre en **Family** dejar **All Families**. De esta forma se seleccionan todas las diferentes familias que tiene el Microchip. A medida que el lector avance puede seleccionar la familia que desee. En **Device** se puede seleccionar el microcontrolador PIC16F628A (para ello lo digita o lo busca en la lista) y se da clic en **Next**, como se observa en la Figura 22A. En la siguiente ventana se hace clic en **Next**, como se muestra en la Figura 22B.

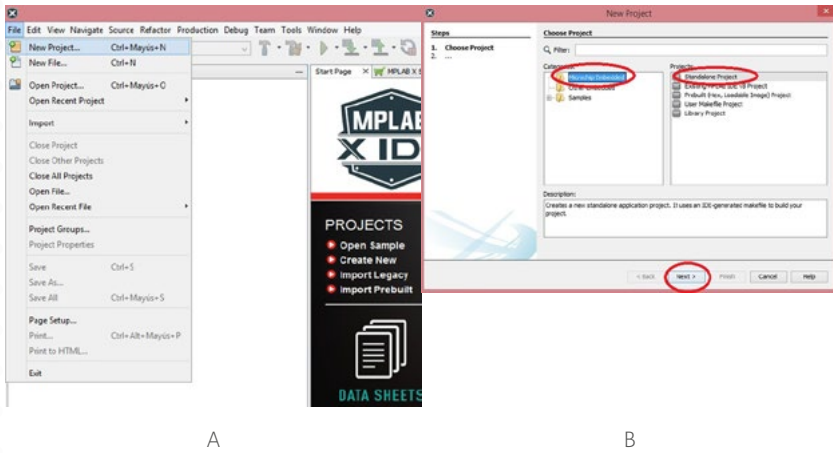


Figura 21. Generar nuevo proyecto en Mplab X 3.65 paso 1

Fuente. Captura de pantalla Mplab X 3.65

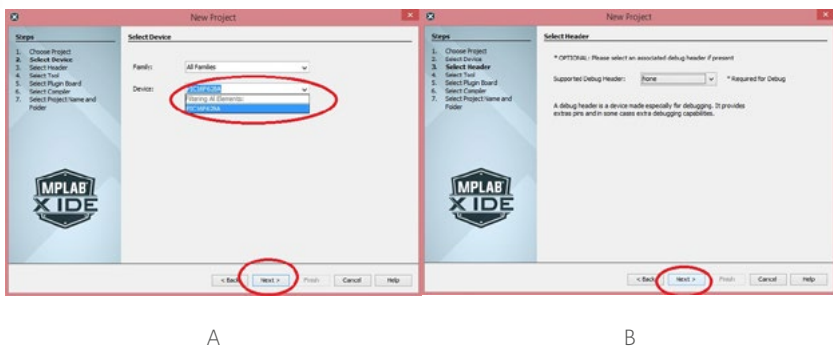


Figura 22. Generar nuevo proyecto en Mplab X 3.65 paso 2

Fuente. Captura de pantalla Mplab X 3.65

Para ver cómo cambian los registros en el proceso de simulación se hace la selección del simulador y se hace clic en **Next**, tal como se observa en la Figura 23A. En la ventana siguiente se debe hacer la selección del compilador **mpasm** y se hace clic en **Next**, tal como se muestra en la Figura 23B. Este compilador es para programar en ensamblador.

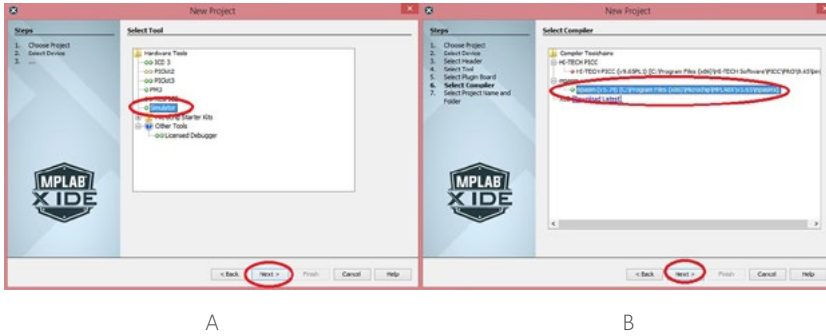


Figura 23. Generar nuevo proyecto en Mplab X 3.65 paso 3

Fuente. Captura de pantalla Mplab X 3.65

En la siguiente ventana se asigna un nombre al proyecto, para este caso SUMA, y se selecciona la carpeta donde se guarda. Para ello, va por **Browse** y se hace clic en **Finish**, tal y como se muestra en la Figura 24A. En la ventana siguiente se hace clic derecho en **Source Files** y se selecciona **New/Other** (ver Figura 24B).

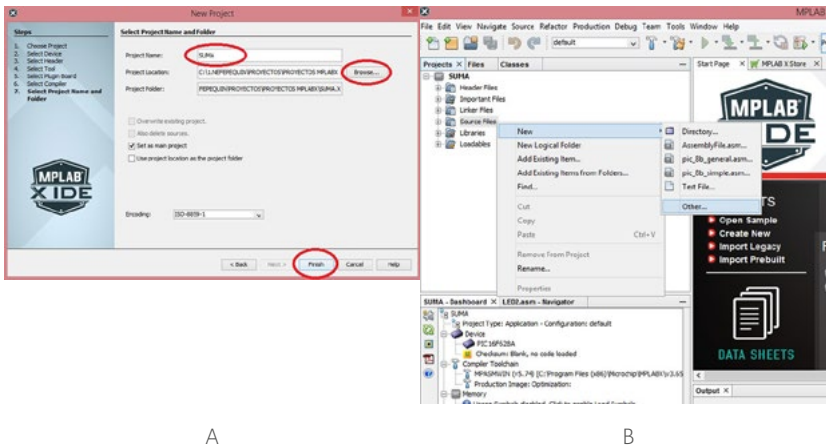


Figura 24. Generar nuevo proyecto en Mplab X 3.65 paso 4

Fuente. Captura de pantalla Mplab X 3.65

Ahora en **Categories**: seleccionar **Assembler**. En **File Types** seleccionar **AssemblerFile.asm** y hacer clic en **Next**, tal como se muestra en la Figura 25A. En la siguiente ventana se le asigna el nombre al archivo, en este caso MAIN, y se hace clic en **Finish**, tal y como se muestra en la Figura 25B.

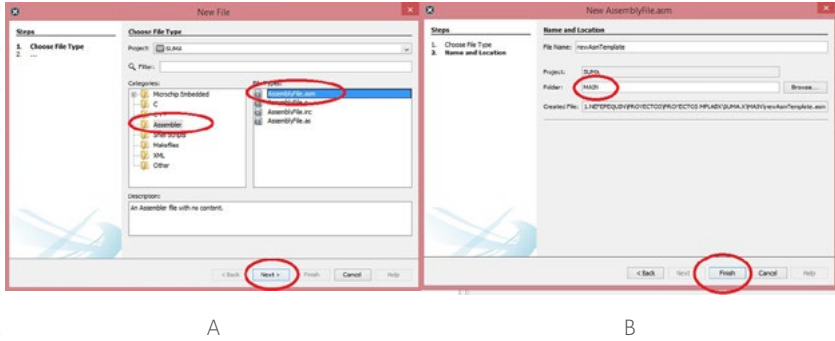


Figura 25. Generar nuevo proyecto en Mplab X 3.65 paso 5

Fuente. Captura de pantalla Mplab X 3.65

El programador debe digitar el código en assembler en la ventana que se abre. Si el programador desea hacer comentarios los puede escribir después de digitar punto y coma. En el diagrama de flujo, lo primero es la librería del microcontrolador (`#INCLUDE<p16f628A.inc>`). Posteriormente se deben escribir los registros de propósito general (`S1, S2, RA, RB`), iniciando en la posición `20H` (ver las instrucciones que están a continuación). En seguida se le da la orden al compilador para que inicie el programa en `0x000H` (`RES_VECT CODE 0X0000`, (ver las instrucciones que están a continuación). Después se comienza a escribir el programa, como lo muestra la Figura 26 A. Al final se debe tener **END** para que el compilador entienda que hasta aquí va el código. A continuación se muestra cómo se da el proceso:

```
#INCLUDE<p16f628A.inc>           ; Definición del microcontrolador. Es decir, la librería.

S1      EQU      0X20           ; SUMANDO 1. Definición de registros de propósito general.
S2      EQU      0X21           ; SUMANDO 2.
RA      EQU      0X22           ; Respuesta parte alta.
RB      EQU      0X23           ; Respuesta parte baja.

RES_VECT CODE 0X0000           ; Vector reset
```

CLRF	RA	; Se asegura que la respuesta alta este cero.
CLRF	RB	; Se asegura que la respuesta baja este cero.
MOVF	S1,0	; Se recupera el sumando 1, es decir S1 = W.
ADDWF	S2,0	; Se realiza la suma W + S2 = W.
BTFSF	STATUS,0	; Se pregunta si la suma dio carry.
BSF	RA,0	; La suma dio carry.
MOVWF	RB	; La suma no dio carry. Se guarda la respuesta parte baja.
PARE		
GOTO	PARE	; Se para la operación del microcontrolador.
		; Como el microcontrolador no tiene una instrucción de pare,
		; entonces hay que pararlo de esa forma.
END		; Final del programa.

Para compilar, pulse **F11** o haga clic en la figura del martillo y la escoba (ver Figura 26B). Si se presentan errores se pueden corregir haciendo clic en el error y el aplicativo remite a la línea donde se presenta dicho error. Previamente se debe leer qué tipo de error es para poder corregirlo. Una vez corregido el error se debe pulsar nuevamente **F11** y repetir el procedimiento hasta que no tenga ningún error.

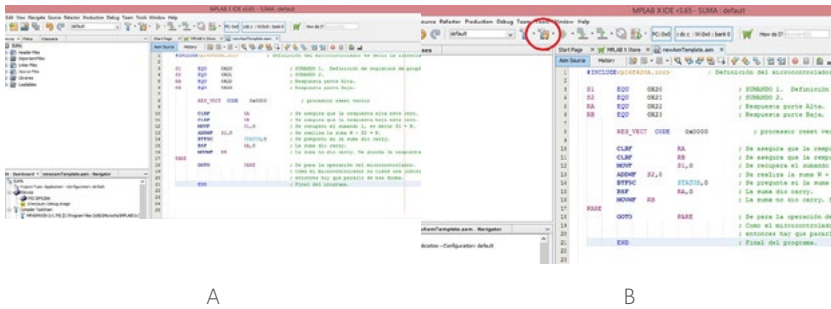


Figura 26. Generar nuevo proyecto en Mplab X 3.65 paso 6

Fuente. Captura de pantalla Mplab X 3.65

Una vez compilado el proyecto hay que simularlo. Es decir, probar que el programa hace la tarea para la cual fue hecho. Para ello, hay que abrir la ventana de los registros de propósito específico y los de registros de propósito general **Window/PIC Memory Views/File Registers** y **Window/PIC Memory Views/SFRs**, (ver Figura 27A). Observe la Figura 27B como quedan organizadas, sólo se seleccionan los que se quieren ver.

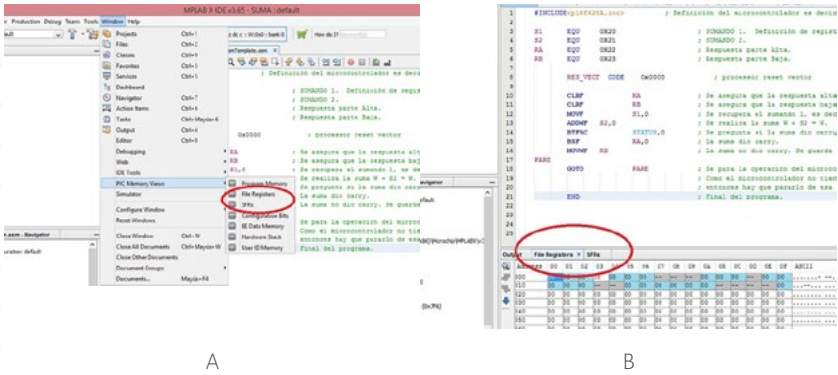


Figura 27. Simulación en Mplab X 3.65 paso 1

Fuente. Captura de pantalla Mplab X 3.65

Ahora hay que habilitar el simulador que va por **Debug/Discrete Debugger Operation/Build Debugging Main Project**. Y tener en cuenta que se deben realizar los dos procedimientos. De lo contrario, no se puede hacer la simulación. Posteriormente, **Debug/Discrete Debugger Operation/Launch Debugger Main Project** (ver Figura 28A). Al hacer esto se habilitan los diferentes íconos de simulación (ver Figura 28B).

La simulación la puede hacer también por teclado (**F7 paso a paso**). El lector podrá explorar otras opciones y otras teclas, a medida que avanza. Para ello, se recomienda ver **Debug**.

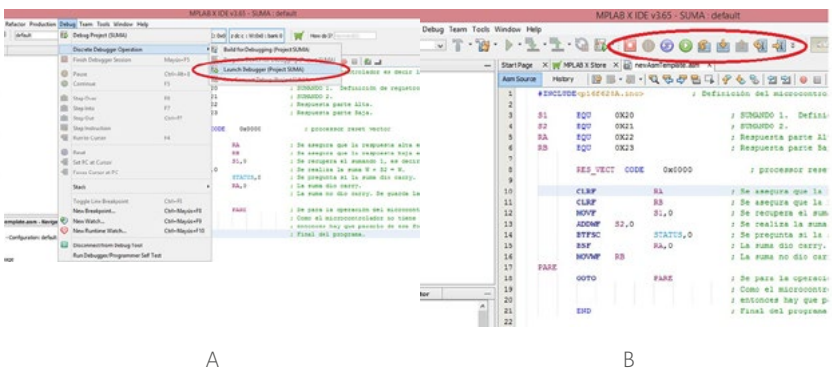


Figura 28. Simulación en Mplab X 3.65 paso 2

Fuente. Captura de pantalla Mplab X 3.65

Para empezar la simulación del programa SUMA hay que hacer clic en **reset** y cargar los registros **S1** y **S2** con los datos que el lector desee sumar. Para ello, hay que colocar el puntero del mouse en las posiciones seleccionadas (20_H y 21_H) y se hace doble clic para modificar el dato (Ver Figura 29A). Con la tecla F7 puede ir simulado paso a paso hasta que el programa termine. En los registros RA y RB (posiciones 22_H y 23_H) está la respuesta de la suma. Para este ejemplo se suma 7A_H + 9B_H = 01_H15_H, (ver Figura 29B).



Figura 29. Simulación en Mplab X 3.65 paso 3

Fuente. Captura de pantalla Mplab X 3.65

Para que el lector se familiarice con la simulación en el siguiente ejemplo se explica, paso a paso, este proceso usando las teclas correspondientes para observar el cambio de los bits de cada registro usados en el programa.

3.5. Puertos de entrada/salida

El microcontrolador PIC16F628A tiene 18 pines, de los cuales 15 son líneas de entrada/salida (*Input/Output*), estos están distribuidos en dos registros llamados PORTA y PORTB, 2 pines de alimentación y 1 pin para el reset (esta línea puede ser usada únicamente como entrada). De esa forma se completan los 18 pines del microcontrolador. El PORT A tiene 7 líneas de I/O y el PORT B tiene 8 líneas de I/O. Estos registros están ubicados en el banco 0 y hacen parte de los registros de propósito

específico. La distribución de estas líneas se puede observar en la Figura 30. Cada línea tiene dos o más tareas. A medida que el lector avance en la programación puede usar estas opciones.

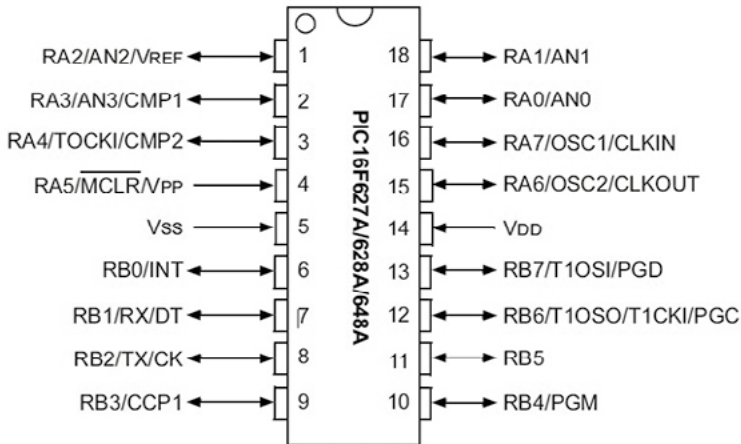


Figura 30. Distribución de pines del PIC 16F628A.

Fuente. (Microchip, 2007, p. 2)

Las líneas de los puertos pueden ser programadas bit a bit. Es decir, el programador tiene la posibilidad, por ejemplo, de tomar todas las 8 líneas del PORT B de salida ó todas de entrada ó 4 bits de salida y 4 bits de entrada ó 2 bits salida y 2 bits de entrada ó 1 bit de salida y 1 bit de entrada, como lo requiera la aplicación que se esté desarrollando.

Para la programación de los puertos se utilizan los registros llamados **TRISA** y **TRISB**, los cuales controlan cada puerto respectivamente. Si se quiere una línea de salida hay que ponerla en 0 (cero) y una línea de entrada hay que ponerla en 1 (uno). Estos registros están ubicados en el banco 1 y hacen parte de los registros de propósito específico. Así, cuando se requiera programar los puertos se ubica el microcontrolador en el banco 1. Una vez programados se regresa al banco 0 para trabajar con los puertos con el fin de sacar o ingresar datos del microcontrolador. Es importante recordar que el registro **STATUS** es el que permite cambiar de bancos a través de los bits 5 (**RP0**) y 6 (**RP1**), (ver subtítulo **3.3.1 Registro STATUS**).

Para que el programador se familiarice con la programación de los puertos, es necesario utilizar algunos ejemplos. Inicialmente se realizarán ejemplos de pro-

gramación de salida. Posteriormente se programarán casos de entrada y salida. De esa forma el lector podrá seguir avanzando en la comprensión del proceso de programación.

3.6. Paso de diseño y Programación

Ejemplo 8. Diseñar un programa que prenda y apague un led durante tiempo indefinido. Para empezar a diseñar este programa se realizan los siguientes pasos:

Paso 1. Saber cómo funciona lo que se quiere controlar (planta), conocer qué voltaje tiene, las corrientes, las formas de onda, entre otras características que se requieren para su correcto funcionamiento. Por ejemplo, si se va a controlar un motor paso a paso, hay que saber cómo funciona o si se va a controlar una matriz de puntos bicolor, hay que saber su funcionamiento .

Paso 2. Hacer un diagrama de conexiones de la distribución del circuito: qué líneas van de salida y cuáles de entrada. Hay que tener en cuenta las líneas de alimentación, de reset y del oscilador. En lo posible se debe tomar en orden los puertos para facilitar programación. Esta distribución va de acuerdo con lo que se quiere controlar.

Paso 3. Hacer el diagrama de flujo de acuerdo con el diagrama de conexiones planteado en el punto anterior.

Paso 4. Pasar a las instrucciones el diagrama de flujo y simularlo. Tener en cuenta el set de instrucciones del microcontrolador, en el cual se está haciendo el programa.

Paso 5. Programar el microcontrolador, armar el circuito y probar.

Paso 6. Si se requiere, se deben realizar las respectivas correcciones y se debe volver al paso tres.

Nota: Si después de hacer varias correcciones no ha logrado realizar el programa para controlar su planta, entonces es necesario regresar al paso dos y en casos más críticos regresar al paso uno.

A continuación, se desarrolla paso a paso:

Paso 1. En este caso hay que controlar un led. Si al led se le garantiza que el cátodo esté en 0 voltios, entonces, cuando al ánodo se le aplican 5 voltios el led prende y cuando se le aplica 0 voltios se apaga. Esto es lo que tiene que hacer el microcontrolador; sacar por un puerto 5 voltios y después 0 voltios.

Paso 2. Hacer un diagrama de conexiones. Hay que decir en cuál de las 15 líneas de *in/out* del microcontrolador se va a poner el led. Por ejemplo, se puede tomar B₃ (ver Figura 31). Es muy importante proteger el led poniéndole una resistencia con él en serie. Recuerde que el led es un testigo, por lo tanto, hay que usarlo como tal. Además, el led tiene un voltaje umbral menor a 5 voltios. Por ello, es necesario poner una resistencia en serie con él para protegerlo. Una de las funciones de un led es que sirve como testigo del funcionamiento de un actuador. Por ejemplo, si hay un motor que está activo hay un led prendido que indica esto o si el motor está inactivo hay un led apagado que lo confirma.

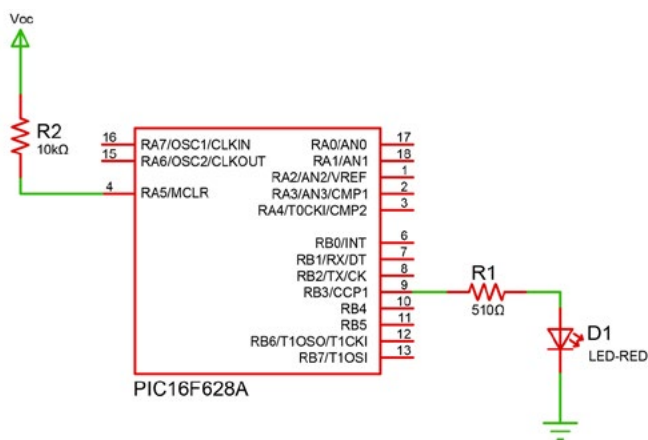
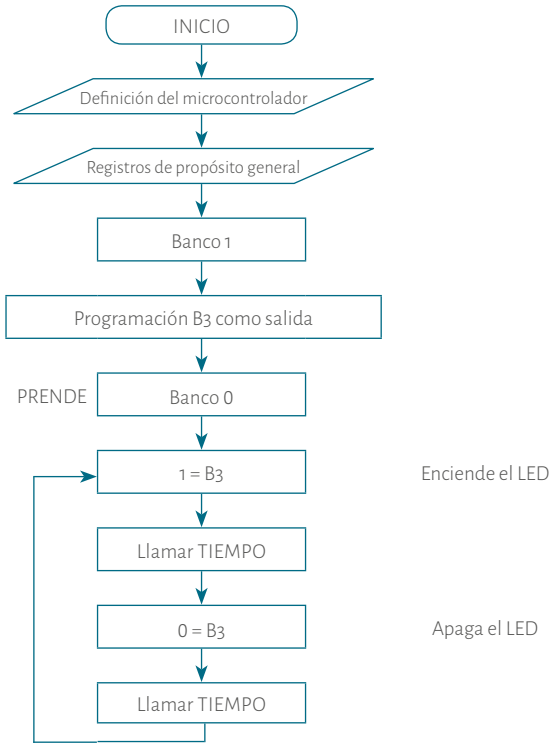


Figura 31. Diagrama de conexiones del ejemplo 8. (Proteus, 2003).

Fuente. Figura elaborada por José Luis González.
(La distribución pines es tomada de Proteus)

Paso 3. Hacer el diagrama de flujo, teniendo en cuenta el diagrama de conexiones planteado en el punto anterior. Programar B₃ como salida para el led. Ver diagrama 2.

Diagrama 2
Programa que prende y apaga un led.

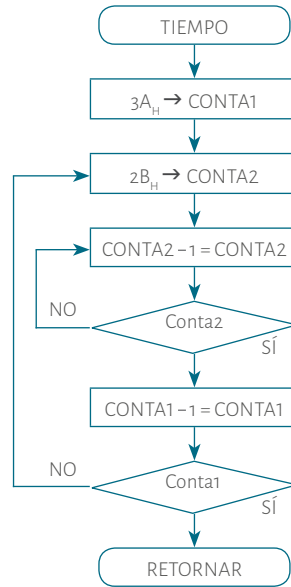


Fuente. Elaboración propia

Dentro del diagrama de flujo se llama la subrutina tiempo, la cual se usa para observar el encendido y el apagado del led. Si no se usara, el ojo humano vería siempre el led prendido porque el microcontrolador trabaja a 4 Mhz.

En el Diagrama de flujo 3 se hace la subrutina tiempo y se observa claramente un típico bucle anidado con dos registros como contadores. El lector puede ver que hasta que los 2 registros contadores estén en 0 el microcontrolador no retorna de la subrutina. Cuando se esté haciendo la simulación se puede ver esto con mayor detalles (paso 4).

Diagrama 3.
Tiempo del ejemplo 8.



Fuente. Elaboración propia

Paso 4. Pasar los diagramas de flujo a instrucciones y, posteriormente, simular. El programa es el siguiente:

```

#include<p16f628A.inc> ; Definición del microcontrolador es decir la librería.

CONTA1 EQU 0X20 ; CONTADOR 1. Definición de registros de propósito general,
                  ; para la subrutina de tiempo.
CONTA2 EQU 0X21 ; CONTADOR 2.

RES_VECT CODE 0X0000 ; Vector reset

BSF STATUS,5 ; Se escoge el banco 1 para programar puertos.
BCF STATUS,6
BCF TRISB,3 ; B3 como salida para prender y apagar el led.
BCF STATUS,5 ; Se escoge el banco 0 para prender y apagar el led.
BCF STATUS,6

PRENDE
BSF PORTB,3 ; Prende el led.
CALL TIEMPO ; Se da tiempo para que el led se vea prendido.
BCF PORTB,3 ; Apaga el led.
CALL TIEMPO ; Se da tiempo para que el led se vea apagado.
GOTO PRENDE ; Regresa a prender el led.

TIEMPO
  
```

	MOVLW	0X3F	; Valor con que se carga el contador 1.
	MOVWF	CONTA1	
NIVEL1			
	MOVLW	0X2B	; Valor con que se carga el contador 2.
	MOVWF	CONTA2	
NIVEL2			
	DEFSZ	CONTA2,1	; Se resta en 1 el contador 2 y se pregunta si es cero.
	GOTO	NIVEL2	; Aun el contador 2 no es cero.
	DEFSZ	CONTA1,1	; Se resta en 1 el contador 1 y se pregunta si es cero.
	GOTO	NIVEL1	; Aun el contador 1 no es cero.
	RETURN		; Retorna de la subrutina.
	END		; Final del programa.

Para la simulación hay que tener en cuenta los valores de los contadores, pues se tardan bastante tiempo. Hay que recordar que se está simulado y no emulado. Es decir, que el proceso no se hace en tiempo real. No se debe olvidar que la tecla para correr el programa paso a paso es F7.

A continuación se explica la simulación del programa. Previamente se debe haber realizado todos los pasos del ejemplo 7 y se debe saber si el registro afectado es de propósito específico o de propósito general (*Window/PIC Memory Views/File Registers* y *Window/PIC Memory Views/SFRs*). El proceso empieza con la programación de los puertos. Para ello, se ubica el microcontrolador en el banco uno usando el registro **STATUS** (dicho registro se encuentra en **SFRs**) con los bits 5 y 6. Las dos primeras instrucciones se explican en la parte inferior la Figura 32A, cómo cambian los bits de este registro al ir ejecutando el programa (oprimiendo **F7** por instrucción, es decir para ello se hizo necesario oprimir dos veces esta tecla). Una vez ubicado el microcontrolador en el Banco 1 se procede a programar la línea B₃ como salida mediante el bit 3 del registro **TRISB** (dicho registro se encuentra en **SFRs**) así como lo muestra la Figura 32B.

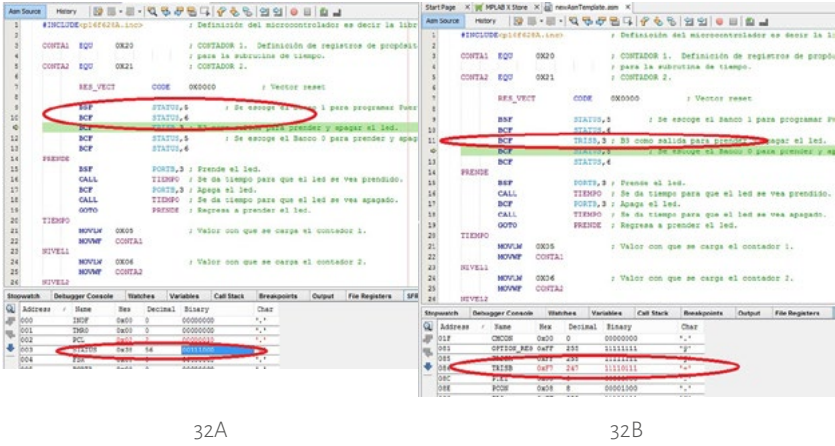


Figura 32. Cambio de banco y programar puertos.

Fuente. Captura de pantalla MPLAB X IDE 3.65

Posteriormente, usando la cuarta y quinta instrucción del programa (**BCF STATUS,5** y **BCF STATUS,6**), el microcontrolador regresa al banco 0, (Ver Figura 33A). Una vez ejecutadas las instrucciones el microcontrolador está listo para prender y apagar el led. Allí está la etiqueta **PRENDE**. Al ejecutar la instrucción **BSF PORTB,3** prende el bit tres del puerto B. En la Figura 33B se observa específicamente cada paso.

En este punto del proceso se usa la subrutina **TIEMPO**. Para ello, se hace un llamado usando la instrucción **CALL TIEMPO**. Cuando se oprime **F7** el microcontrolador salta a esta subrutina (ver el contador de programa PCL, el cual indica la posición del programa que se está ejecutando. Este registro se encuentra en **SFRs**). En la Figura 34A se usan los registros **CONTADOR1** y **CONTADOR2** (dichos registros se encuentran en **File Registers**). Luego se hace un bucle anidado (recordar el diagrama de flujo 3). Al oprimir **F7** la instrucción **MOVLW 0X3F** carga $3F_H$ al registro W y al ejecutar la instrucción **MOVWF CONTADOR1**, el valor $3F_H$ pasa a este registro. Con las dos instrucciones siguientes se hace lo mismo con el **CONTADOR 2**, pero en este caso se carga $2B_H$.

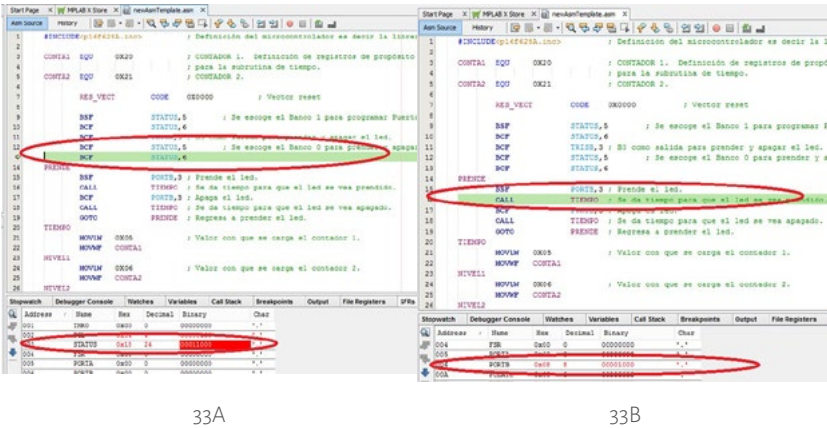


Figura 33. Cambio al banco 0 y prender el led.

Fuente. Captura de pantalla MPLAB X IDE 3.65

Tener en cuenta que mientras el microcontrolador está ejecutando esta subrutina el bit 3 del puerto B sigue prendido.

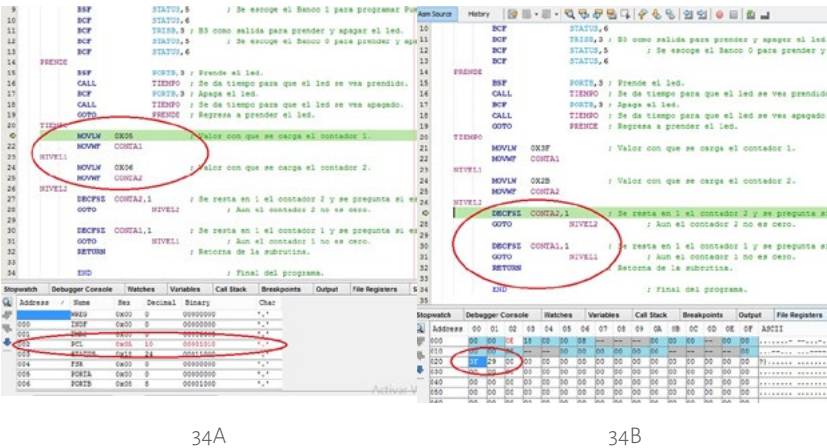
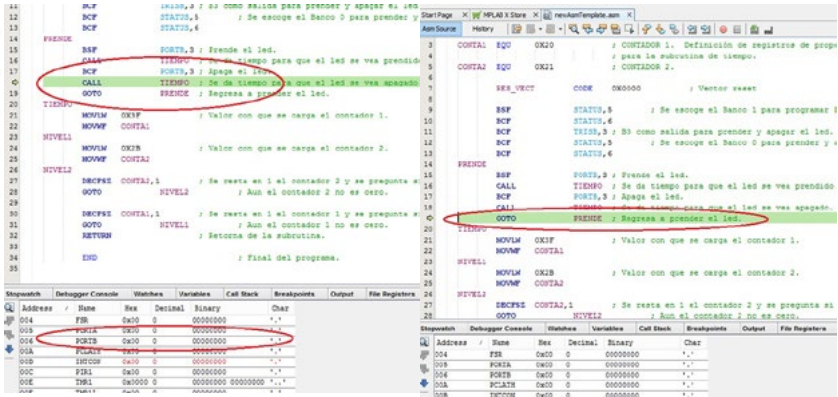


Figura 34. Llamado de la Subrutina de TIEMPO.

Fuente. Captura de pantalla MPLAB X IDE 3.65

Al seguir ejecutado la subrutina **TIEMPO**, la instrucción **DECFSZ CONTADOR2,1** junto con la instrucción **GOTO NIVEL2** decrementan el registro **CONTADOR2** hasta que es equivalente a 00_H . Cuando se cumple esta condición se ejecuta la instrucción **DECFSZ CONTADOR1,1**. Si el registro **CONTADOR1** no es 00_H se ejecuta la instrucción **GOTO NIVEL1**. Esto quiere decir que se ejecutan las dos

instrucciones anteriores hasta cuando el registro **CONTADOR1** sea 00_{H} . Cuando se cumple esta condición se ejecuta la instrucción **RETURN**, la cual hace que el microcontrolador regrese al programa principal (recordar el diagrama de flujo 3). Así como se observa en la Figura 34B. En otras palabras, el microcontrolador no sale de esta subrutina si los dos registros no son 00_{H} . Entre más grande sean los valores de los dos registros contadores (**CONTADOR1** Y **CONTADOR2**) el tiempo es mayor. Al retonar de la subrutina se ejecuta la instrucción **BCF PORTB,3** que hace que el bit tres del puerto B se ponga en 0. Por lo tanto, el led se apaga, como lo muestra la Figura 35A. Allí nuevamente se usa la subrutina **TIEMPO** para que el ojo humano pueda ver el led encendido y apagado. Después se ejecuta la instrucción **GOTO PRENDE**, la cual ejecuta el programa nuevamente hasta que se apague el microcontrolador (ver Figura 35B).



35A

35B

Figura 35. Apagado del led.

Fuente. Captura de pantalla MPLAB X IDE 3.65

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 30).

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso tres.

Ejemplo 9. Diseñar un juego de luces como las del auto fantástico que tengan 8 leds y que enciendan de derecha a izquierda uno a uno. Después de izquierda a derecha uno a uno y se repite el proceso.

A continuación, se desarrollará paso por paso:

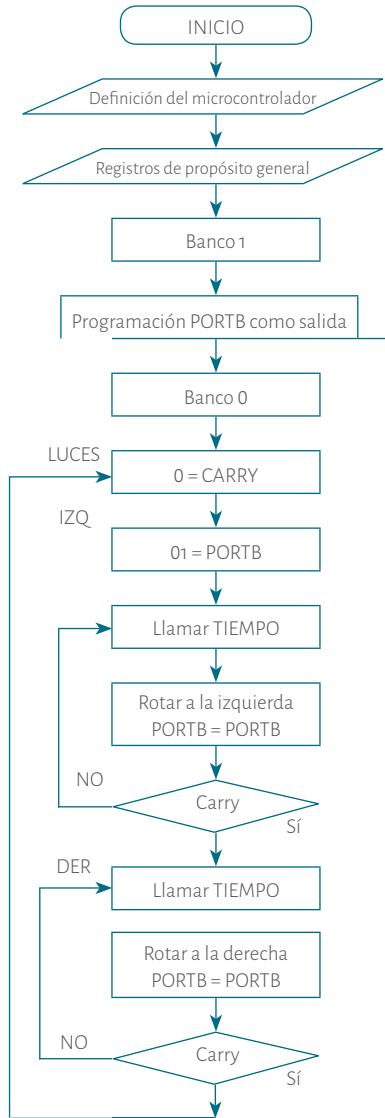
Paso 1. Para este ejemplo ya saben cómo funciona un led, así que el diseñador debe agregar los 7 leds restantes.

Paso 2. Como hay que poner 8 leds, el único puerto que tiene 8 líneas es PORTB, allí están los leds. Las conexiones son muy similares a la Figura 27, sólo que en este montaje hay que agregar 7 leds más, desde B0 hasta B7, cada led con su resistencia en serie para protección.

Paso 3. De acuerdo con el diagrama de conexiones hay que programar todo el PORTB como salida para los 8 leds (ver Diagrama 4). En este programa hay que trabajar con el bit del *carry* del registro **STATUS**, pues al rotar los bits del PORTB se ve afectado el bit *carry*. Esta rotación se hace de derecha a izquierda y viceversa. Las instrucciones que se usan para ello son: **RLF *f,d*** y **RRF *f,d***. Como hay que preguntar por el *carry* las instrucciones son: **BTFSC *f,b*** y **BTFSS *f,b***. La subrutina de tiempo también será usada en este programa entonces será copiada idéntica a la del ejemplo 8.

Paso 4. El programa se muestra después del Diagrama 4. Note que la subrutina de tiempo es la misma del ejemplo anterior pero con otros valores de los contadores. Si se desea que sea más rápido o más lento el juego de luces, sólo se aumenta o se disminuye el valor a los contadores desde 00_{H} a FF_{H} . Una vez digitado el programa y compilado, se puede simular.

Diagrama 4.
Programa juego de luces.



Fuente. Elaboración propia

#INCLUDE<pt16f628A.inc>

CONTA1 EQU 0X20

; Definición del microcontrolador es decir la librería.

CONTA2 EQU 0X21

; CONTADOR 1. Definición de registros de propósito general,
; para la subrutina de tiempo.

; CONTADOR 2.

RES_VECT CODE 0X0000

; Vector reset

BSF STATUS,5

; Se escoge el Banco 1 para programar puertos.

BCF STATUS,6

CLRF TRISB

; Puerto B como salida para los 8 leds.

BCF STATUS,5

; Se escoge el Banco 0 para hacer el juego de luces.

BCF STATUS,6

LUCES

BCF STATUS,0

; Se pone cero en el bit de carry.

MOVLW 0X01

; Se garantiza el primer led encendido así: 0000 0001

MOVWF PORTB

IZQ

CALL TIEMPO

; Se da tiempo para que el led se vea encendido.

RLF PORTB,1

; Se rota el PORTB a la izquierda.

BTFSZ STATUS,0

; Se pregunta si ya se rotó las 8 veces.

GOTO IZQ

; Aun no se ha rotado las 8 veces a la izquierda.

DER

CALL TIEMPO

; Se da tiempo para que el led se vea encendido.

RRF PORTB,1

; Se rota el PORTB a la derecha.

BTFSZ STATUS,0

; Se pregunta si ya se rotó las 8 veces.

GOTO DER

; Aun no se ha rotado las 8 veces a la derecha.

GOTO LUCES

; Regresa a encender el primer led.

TIEMPO

MOVLW 0X7C

; Valor con que se carga el contador 1.

MOVWF CONTA1

NIVEL1

MOVLW 0X5E

; Valor con que se carga el contador 2.

MOVWF CONTA2

NIVEL2

DECFSZ CONTA2,1

; Se resta en 1 el contador 2 y se pregunta si es cero.

GOTO NIVEL2

; Aun el contador 2 no es cero.

DECFSZ CONTA1,1

; Se resta en 1 el contador 1 y se pregunta si es cero.

GOTO NIVEL1

; Aun el contador 1 no es cero.

RETURN

; Retorna de la subrutina.

END

; Final del programa.

Paso 5. Armar el circuito teniendo en cuenta la distribución del microcontrolador y el diagrama planteado en el punto 3. Programar y probar.

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso 3.

Hasta el momento sólo se han obtenido datos del microcontrolador. El siguiente ejemplo se hace para ingresar datos y de esa forma tomar decisiones de acuerdo con el dato en el exterior.

Ejemplo 10. Hacer un programa que permita cambiar la velocidad de un led, es decir, que si un pulsador está abierto (uno) un led encienda y apague a una velocidad y si el pulsador está cerrado (cero) el led encienda y apague a una velocidad más rápida.

A continuación, se desarrolla paso a paso:

Paso 1. Ya se sabe cómo funciona un led. Es preciso saber cómo funciona un pulsador. Hay dos clases de pulsadores: uno normalmente abierto y otro normalmente cerrado. Para este caso se usará uno normalmente abierto, en configuración pull up, como se observa en la Figura 36.

Paso 2. Hay que decir en cuál de las 15 líneas del microcontrolador se va a ubicar el led y en cuál el pulsador. Por ejemplo, se puede tomar B1 para el led y B4 para el pulsador (ver Figura 36).

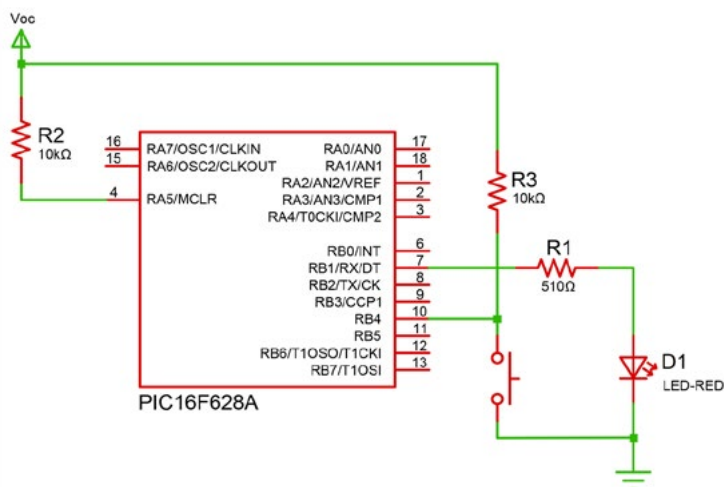


Figura 36. Diagrama de conexiones del ejemplo diez.

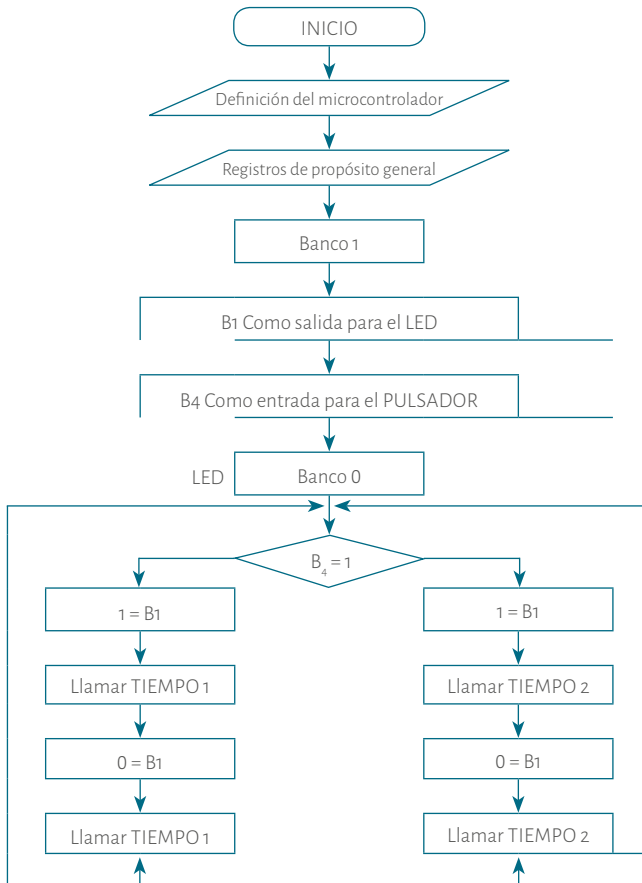
Fuente. Elaboración José Luis González. (La distribución de pines es tomada de Proteus).

Paso 3. De acuerdo con el diagrama de conexiones hay que programar B_1 como salida para el led y B_4 como entrada para el pulsador (ver Diagrama 5). Dentro del diagrama de flujo se llama las subrutinas tiempo 1 y tiempo 2. Estas subrutinas se usan para la diferencia de las velocidades. Es la misma subrutina pero con valores diferentes en los contadores.

Paso 4. En este programa hay que tomar una decisión. Para ello, se usa la instrucción **BTFSF PORTB, 4**. Esta instrucción se usa para ir a **VELO1** ó **VELO2** según el estado del bit 4 del registro PORTB. El lector puede ver cómo funciona esta instrucción (21).

Diagrama 5.

Programa que prende y apaga un led con dos velocidades.



Fuente. Elaboración propia.

	MOVLW	0X83	; Valor con que se carga el contador 1.
	MOVWF	CONTA1	
NIVEL3			
	MOVLW	0X32	; Valor con que se carga el contador 2.
	MOVWF	CONTA2	
NIVEL4			
	DEFSZ	CONTA2,1	; Se resta en 1 el contador 2 y se pregunta si es cero.
	GOTO	NIVEL4	; Aun el contador 2 no es cero.
	DEFSZ	CONTA1,1	; Se resta en 1 el contador 1 y se pregunta si es cero.
	GOTO	NIVEL3	; Aun el contador 1 no es cero.
	RETURN		; Retorna de la subrutina.
	END		; Final del programa.

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 26).

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso tres.

3.7. Manejo de display 7 segmentos

Hasta el momento se han manejado leds como visualizadores, ahora se van a realizar programas para utilizar un display 7 segmentos. Para ello, el siguiente ejemplo ilustra cómo realizarlo.

Ejemplo 11. Se desea visualizar los números decimales en un display 7 segmentos de ánodo común. Sólo se puede utilizar el PIC 16F628A, sino se puede utilizar un decodificador de 7 segmentos.

A continuación, se desarrolla el proceso paso a paso:

Paso 1. Recordar que existen display de ánodo común y cátodo común. Para los primeros en el común hay que garantizar $5 V_{DC}$ (uno lógico), en los segmentos que se quieren prender $0 V_{DC}$ (cero lógico) y en los segmentos que se quieren apagar $5 V_{DC}$ (uno lógico). Para los segundos es lo contrario, en el común hay que garantizar $0 V_{DC}$ (cero lógico), en los segmentos que se quieren prender $5 V_{DC}$ (uno lógico) y en los segmentos que se quieren apagar $0 V_{DC}$ (cero lógico). Los segmentos de los displays tienen la distribución que se muestra en la Figura 38.

Paso 2. Hay que decir en cuáles de las 15 líneas del microcontrolador se va a poner los 7 segmentos del display. Se puede tomar el PORTB, desde B₀ hasta B₆, una línea para cada segmento, (ver Figura 37).

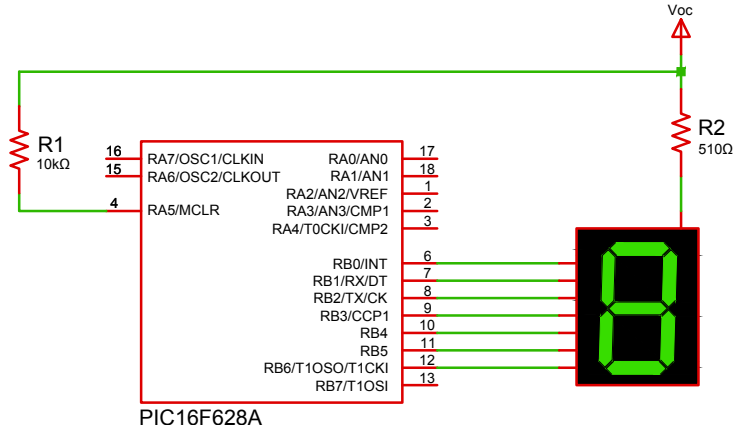


Figura 37. Diagrama de conexiones del ejemplo 11.

Fuente, Elaboración José Luis González. (La distribución de pines es tomada de Proteus).

Paso 3. Para hacer el diagrama de flujo hay que saber qué códigos seleccionar para cada número, para ello recordar la distribución de los segmentos para un display de 7 segmentos (ver Figura 38).

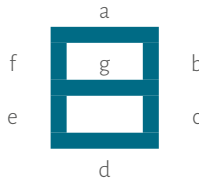


Figura 38. Distribución de un display 7 segmentos.

Fuente. Elaboración propia.










De acuerdo con el diagrama de conexiones planteado en el paso 2, hay que programar todo el PORTB como salida para los 7 segmentos del display. Aquí hay que saber qué datos sacar en el PORTB para visualizar cada número. Para ello, se puede observar la tabla 15 en la que se muestra que para cada línea del PORTB hay un segmento del display. De esa forma se obtiene el código para cada número. Para este ejemplo se trabaja con un display de ánodo común. Por lo tanto, para prender un segmento se garantiza cero (0) y para que apague uno (1). Observe que para este

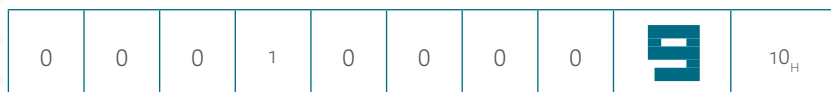
caso no se está usando la línea B7 del puerto. Por consiguiente, esta línea se puede poner en cero o en uno, en este caso se puso en cero. Si se quiere se puede usar para trabajar el punto del display.

Si se quiere observar otro número o alguna letra sólo se tiene que tener en cuenta qué segmento hay que encender y cuál hay que apagar. Sin embargo, es importante saber que con este tipo de display no se puede observar todas las letras, por lo que es preciso usar un display alfanumérico o de 14 segmentos.

El diagrama de flujo es el que se muestra en Diagrama 6. Observe el orden para sacar los códigos de cada número y el tiempo que se da para ver cada uno. Note que el diagrama no está completo. Por lo tanto, el lector puede completarlo, solo debe seguir la secuencia.

Tabla 15.
Códigos para números con display ánodo común.

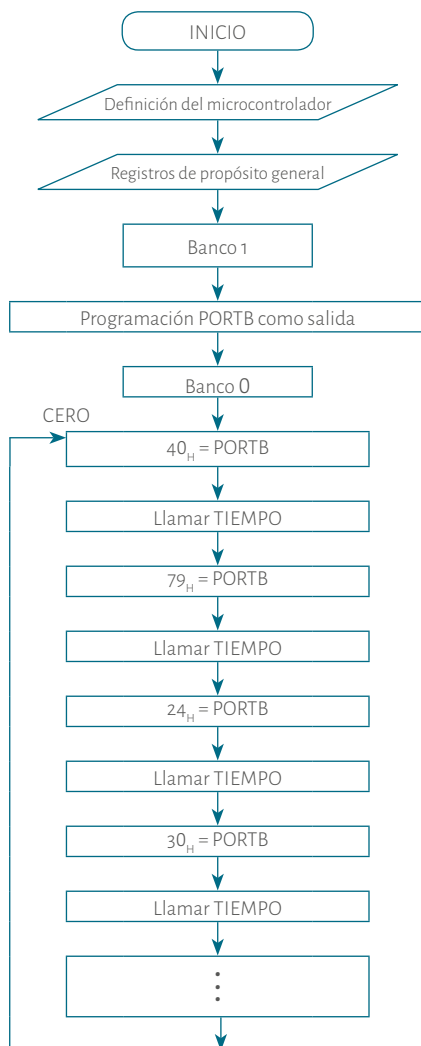
B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	Número	Código
x	g	F	e	d	C	b	a		
0	1	0	0	0	0	0	0		40 _H
0	1	1	1	1	0	0	1		79 _H
0	0	1	0	0	1	0	0		24 _H
0	0	1	1	0	0	0	0		30 _H
0	0	0	1	1	0	0	1		19 _H
0	0	0	1	0	0	1	0		12 _H
0	0	0	0	0	0	1	0		02 _H
0	1	1	1	1	0	0	0		78 _H
0	0	0	0	0	0	0	0		00 _H



Fuente. Elaboración propia

Diagrama 6.

Programa para visualizar los números decimales en display 7 segmentos.



Fuente. Elaboración propia.

Paso 4. El programa es el siguiente: la subrutina de tiempo es la misma que la de los anteriores programas. Si se quiere más tiempo entonces se debe au-

mentar el valor de los contadores. Recuerde que puede cargar los contadores desde 00_H hasta FF_H.

```

#INCLUDE<p16f628A.inc>           ; Definición del microcontrolador es decir la librería.

CONTA1 EQU 0X20                 ; CONTADOR 1. Definición de registros de propósito general,
                                ; para la subrutina de tiempo.
CONTA2 EQU 0X21                 ; CONTADOR 2.

RES_VECT CODE 0X0000           ; Vector reset

BSF STATUS,5                   ; Se escoge el Banco 1 para programar Puertos.
BCF STATUS,6
CLRFB TRISB                     ; PORT B como salida para los siete segmentos del display.
BCF STATUS,5                   ; Se escoge el Banco 0 para hacer el juego de luces.
BCF STATUS,6

CERO

MOVLW 0X40                     ; SE CARGA EL CÓDIGO DEL CERO A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X79                     ; SE CARGA EL CÓDIGO DEL UNO A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X24                     ; SE CARGA EL CÓDIGO DEL DOS A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X30                     ; SE CARGA EL CÓDIGO DEL TRES A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X19                     ; SE CARGA EL CÓDIGO DEL CUATRO A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X12                     ; SE CARGA EL CÓDIGO DEL CINCO A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X02                     ; SE CARGA EL CÓDIGO DEL SEIS A W.
MOVWF PORTB                    ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO                    ; TIEMPO PARA VISUALIZAR EL DATO.

MOVLW 0X78                     ; SE CARGA EL CÓDIGO DEL SIETE A W.

```

```

MOVWF PORTB           ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO           ; TIEMPO PARA VISUALIZAR EL DATO.

MOV LW 0X00           ; SE CARGA EL CÓDIGO DEL OCHO A W.
MOVWF PORTB           ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO           ; TIEMPO PARA VISUALIZAR EL DATO.

MOV LW 0X10           ; SE CARGA EL CÓDIGO DEL NUEVE A W.
MOVWF PORTB           ; SE MUESTRA EN EL DISPLAY
CALL TIEMPO           ; TIEMPO PARA VISUALIZAR EL DATO.

GOTO CERO             ; SE INICIA NUEVAMENTE LA CUENTA

;
TIEMPO

MOV LW 0XFF           ; Valor con que se carga el contador 1.
MOVWF CONTA1

NIVEL1

MOV LW 0XFF           ; Valor con que se carga el contador 2.
MOVWF CONTA2

NIVEL2

DECFSZ CONTA2,1       ; Se resta en 1 el contador 2 y se pregunta si es cero.
GOTO NIVEL2           ; Aun el contador 2 no es cero.
DECFSZ CONTA1,1       ; Se resta en 1 el contador 1 y se pregunta si es cero.
GOTO NIVEL1           ; Aun el contador 1 no es cero.
RETURN                ; Retorna de la subrutina.

END                   ; Final del programa.

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 30) y la distribución de los segmentos del display.

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso 3.

3.7.1 Visualización dinámica con display de 7 segmentos

Ahora se puede manejar más de un display de 7 segmentos. El siguiente ejemplo puede ser útil para visualizar una palabra de 4 letras.

Ejemplo 12. Se desea visualizar la palabra HOLA en el 4 display de 7 segmentos de ánodo común. Sólo se puede utilizar el PIC 16F628A, es decir, sin utilizar otro integrado. A continuación, se desarrolla paso a paso:

Paso 1. Como ya se conoce el funcionamiento de los displays de ánodo común, es preciso saber cómo se van a manejar 4 displays con un solo microcontrolador. Estos displays se pueden multiplexar de la siguiente forma: unir todos los segmentos **a** de los 4 displays, unir todos los segmentos **b** de los 4 displays, unir todos los segmentos **c** de los 4 displays y, así sucesivamente, hasta el segmento **g**. El común de cada display se deja sin unir. En el mercado se consiguen displays que ya vienen multiplexados. De esta forma se evita realizar un proceso cableado complicado

Paso 2. Hay que decir en cuáles de las 15 líneas del microcontrolador se van a poner los 7 segmentos de todos los displays y en cuáles los comunes. Se puede tomar el PORTB para los segmentos y el PORTA para los comunes (ver Figura 39).

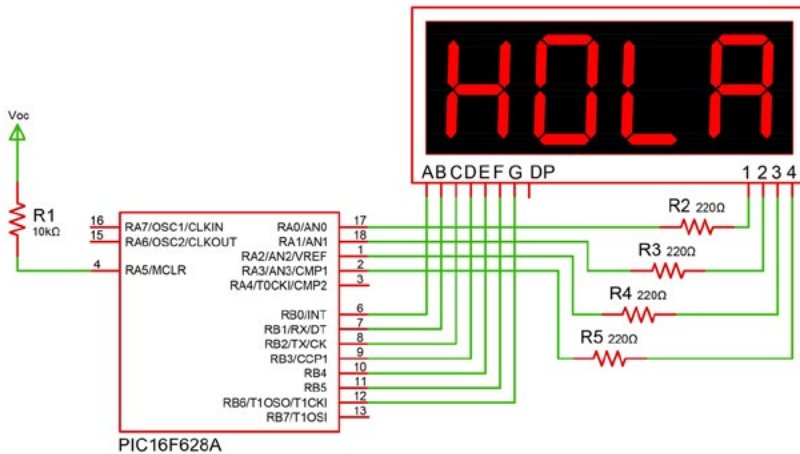


Figura 39. Diagrama de conexiones del ejemplo 12.



Fuente. Elaboración José Luis González. (La distribución de pines es tomada de Proteus).

Para este ejemplo se está utilizando el PORTA. Este microcontrolador en dicho puerto tiene comparadores analógicos (amplificadores operacionales). Por lo tanto, hay que deshabilitarlos para usar el puerto como entrada y salida digital. Para dicho proceso hay que enviar 07_H al registro **CMCON** (Microchip, 2007, p. 31) el cual se encuentra el banco 0.

Paso 3. Para hacer el diagrama de flujo hay que saber qué códigos tiene cada letra, recordar la distribución de los segmentos de un display de 7 segmentos, los cuales van a ser sacados en el PORTB (Ver Figura 38 y ver la Tabla 16).

De acuerdo con el diagrama de conexiones planteado en el paso 2. Hay que programar todo el PORTB como salida para los 7 segmentos de los 4 displays y el PORTA también como salida para los comunes de todos los displays.

Tabla 16.
Códigos para la palabra HOLA del ejemplo 12.

B ₇	B ₆	B ₅	B ₄	B ₃	B ₂	B ₁	B ₀	Letra	Código
x	G	F	e	d	C	b	A		
0	0	0	0	1	0	0	1		09 _H
0	1	0	0	0	0	0	0		40 _H

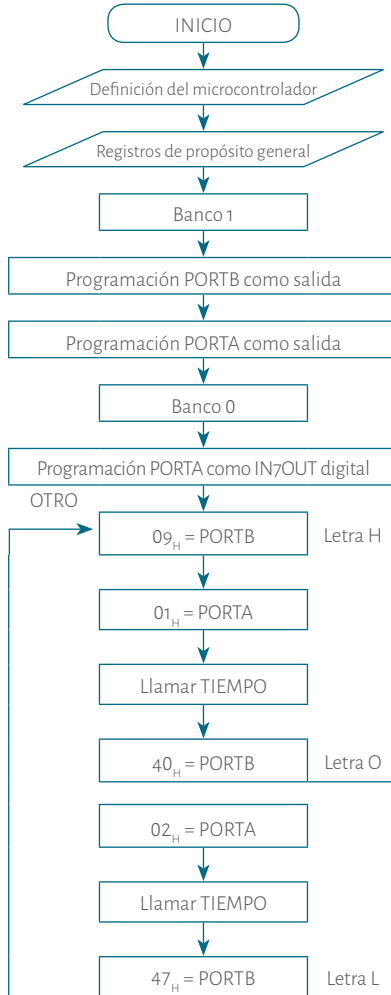
0	1	0	0	0	1	1	1	L	47
0	0	0	0	1	0	0	0	A	08 _H

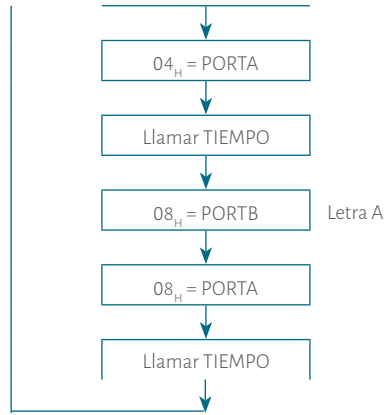
Fuente. Elaboración propia.

El diagrama de flujo es el que se muestra en el Diagrama 7, observe el orden para sacar los códigos de cada letra y el tiempo que se da para ver cada una.

Diagrama 7.

Programa para visualizar la palabra HOLA en 4 display de 7 segmentos.





Fuente. Elaboración propia.

Paso 4. El programa es el siguiente: la subrutina de tiempo es la misma de los anteriores programas pero para este ejemplo hay que visualizar las 4 letras al tiempo. Los valores de los contadores no pueden ser muy grandes porque, de lo contrario, se visualizaría letra por letra. No obstante, no pueden ser muy pequeños porque los 7 segmentos de los 4 displays estarían prendidos. Por lo tanto, no se pueden identificar las letras.

```

#include <p16f628A.inc>           ; Definición del microcontrolador es decir la librería.

CONTA1 EQU 0x20                 ; CONTADOR 1. Definición de registros de propósito general,
                                ; para la subrutina de tiempo.
CONTA2 EQU 0x21                 ; CONTADOR 2.

RES_VECT CODE 0x0000           ; Vector reset

BSF STATUS,5                   ; Se escoge el Banco 1 para programar puertos.
BCF STATUS,6
CLRF TRISB                      ; PORT B como salida para los siete segmentos del display.
CLRF TRISA                      ; PORT A como salida para los comunes de los display.
BCF STATUS,5                   ; Se escoge el Banco 0 para hacer el juego de luces.
BCF STATUS,6
MOVLW 0x07                     ; Dato para deshabilitar los comparadores del PORT A.
                                ; Se deja PORT A como entrada y/o salida digital.

MOVWF CMCON                    ; Registro que controla los comparadores.

OTRO

MOVLW 0x09                     ; Se carga la letra H.
MOVWF PORTB
MOVLW 0x01                     ; Se habilita D1.
MOVWF PORTA
CALL TIEMPO                    ; Tiempo para visualizar la letra H.
  
```

```

MOV LW 0X40 ; Se carga la letra A.
MOV WF PORTB
MOV LW 0X02 ; Se habilita Dz.
MOV WF PORTA
CALL TIEMPO ; Tiempo para visualizar la letra A.

MOV LW 0X47 ; Se carga la letra L.
MOV WF PORTB
MOV LW 0X04 ; Se habilita D3.
MOV WF PORTA
CALL TIEMPO ; Tiempo para visualizar la letra L.

MOV LW 0X08 ; Se carga la letra A.
MOV WF PORTB
MOV LW 0X08 ; Se habilita D4.
MOV WF PORTA
CALL TIEMPO ; Tiempo para visualizar la letra L.

;
TIEMPO
MOV LW 0X30 ; Valor con que se carga el contador 1.
MOV WF CONTA1

NIVEL1
MOV LW 0X20 ; Valor con que se carga el contador 2.
MOV WF CONTA2

NIVEL2
DECFSZ CONTA2,1 ; Se resta en 1 el contador 2 y se pregunta si es cero.
GOTO NIVEL2 ; Aun el contador 2 no es cero.

DECFSZ CONTA1,1 ; Se resta en 1 el contador 1 y se pregunta si es cero.
GOTO NIVEL1 ; Aun el contador 1 no es cero.

RETURN ; Retorna de la subrutina.

END ; Final del programa.

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador y la distribución del display (ver Figura 30).

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso tres.

3.7.2. Rotación de datos con display 7 segmentos

Ya que se ha manejado la visualización dinámica con 4 displays de 7 segmentos, ahora se puede realizar un visualizador de mensajes. Para ello, se puede ver el siguiente ejemplo.

Ejemplo 13. Se desea visualizar una frase completa. Por ejemplo, **PRUEBA DEL PUBLIC - HOLA**, en 4 displays 7 segmentos de ánodo común. Sólo se puede utilizar el PIC 16F628A. Es decir, sin utilizar otro integrado. A continuación se desarrolla paso a paso:

Paso 1. En el ejemplo anterior se trabajó con 4 displays de ánodo común multiplexados. Por lo tanto, el funcionamiento es el mismo.

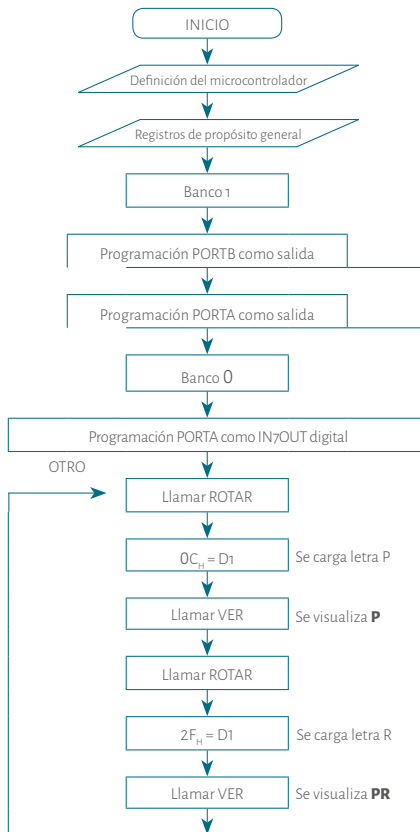
Paso 2. En este montaje se usará la misma distribución del ejemplo anterior (ver Figura 39).

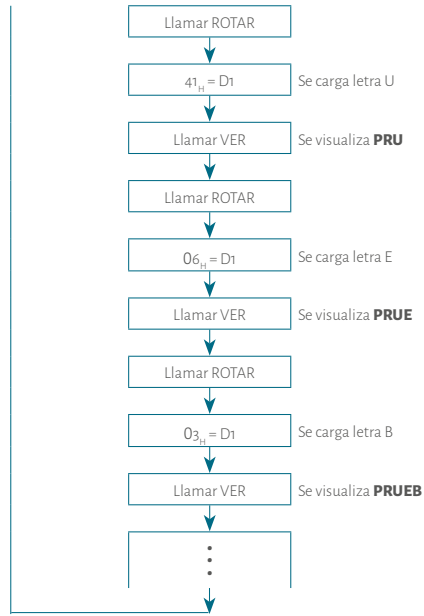
Paso 3. Para hacer el diagrama de flujo hay que saber los códigos de cada letra. Se recomienda usar la tabla de los dos ejemplos anteriores. De esa forma

puede obtener el código de cada una, incluyendo el espacio que va entre las palabras (ver Diagrama 8). Este diagrama de flujo está incompleto. El lector podrá darse cuenta que solo debe seguir la secuencia para completarlo de la siguiente forma: primero se llama la subrutina **Rotar**, Diagrama 9, luego se carga el código de la **letra** en el registro **D1** y, por último, se llama la subrutina (ver Diagrama 10). Para este ejemplo a cada display se le asignará un registro de la RAM (registros de propósito general), los cuales serán **D1**, **D2**, **D3** y **D4**.

La subrutina **Rotar** se encarga de desplazar (rotar) los datos que tienen los registros **D1**, **D2**, **D3** y **D4**. De esa forma se da el efecto de desplazamiento del texto así: lo primero que se hace es pasar lo que tiene el registro **D3** al registro **D4**. Ambos son registros de propósito general, no hay ninguna instrucción para ello entonces hay que pasar del registro **D3** a **W** y luego **W** a **D4**, luego **D2** a **D3** y así sucesivamente.

Diagrama 8.
Programa para visualizar una frase completa
en cuatro displays de siete segmentos.



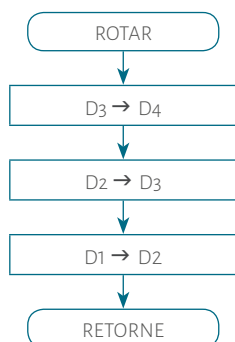


Fuente. Elaboración propia

La subrutina **Ver** se encarga de recuperar la información que tiene los registros **D1**, **D2**, **D3** y **D4** para pasarla al PUERTO B a través del registro **W**. En este momento es donde se visualizan las letras en los display. Recuerde que las letras se muestran

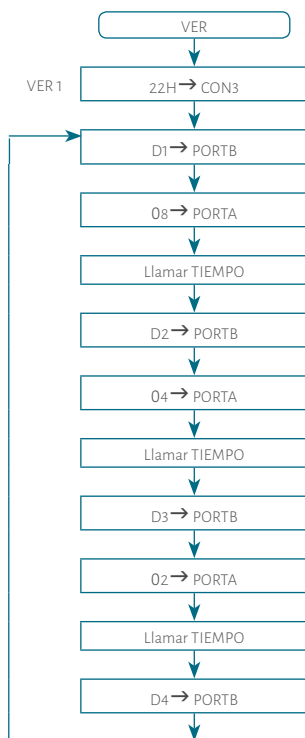
una a una, pero tan rápido que se ven todos encendidos. Como la mayor parte del tiempo el microcontrolador debe estar visualizando las letras que se van ingresando, entonces esta subrutina tiene un bucle que tiene un contador. Entre más grande es el valor del contador, más lento es el ingreso de una nueva letra.

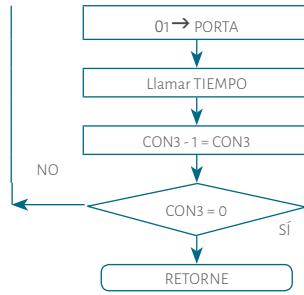
Diagrama 9.
Subrutina ROTAR del ejemplo 13.



Fuente. Elaboración propia

Diagrama 10.
Subrutina VER del ejemplo 13.





Fuente. Elaboración propia.

Paso 4. El programa es el siguiente: la subrutina de tiempo es la misma de los anteriores programas con los mismos valores del anterior ejemplo. Note que el programa no está completo, sólo se debe seguir la secuencia para las otras letras.

```

#include<pt16f628A.inc> ; Definición del microcontrolador es decir la librería.

CONTA1 EQU 0X20 ; CONTADOR 1. Definición de registros de propósito general,
                  ; para la subrutina de tiempo.
CONTA2 EQU 0X21 ; CONTADOR 2.
CON3 EQU 0X22 ; CONTADOR 3.
D1 EQU 0X23
D2 EQU 0X24
D3 EQU 0X25
D4 EQU 0X26
RES_VECT CODE 0X0000 ; Vector reset

BSF STATUS,5 ; Se escoge el Banco 1 para programar puertos.
BCF STATUS,6
CLRF TRISB ; PORT B Como salida para los siete segmentos del display.
CLRF TRISA ; PORT A Como salida para los comunes de los display.
BCF STATUS,5 ; Se escoge el Banco 0 para hacer el juego de luces.
BCF STATUS,6
MOVLW 0X07 ; Dato para deshabilitar los comparadores del PORT A.
            ; Se deja PORT A como entrada o salida digital.
MOVWF CMCON ; Registro que controla los comparadores.

OTRO
CALL ROTAR ; Se hace el desplazamiento para dar cabida a la primer letra.
MOVLW 0X0C ; Se carga la letra P.
MOVWF D1
CALL VER ; Se visualiza P
CALL ROTAR ; Se hace el desplazamiento para dar cabida a la segunda letra.
MOVLW 0X2F ; Se carga la letra R.
MOVWF D1
  
```

```

CALL VER ; Se visualiza PR
CALL ROTAR ; Se hace el desplazamiento para dar cabida a la tercer letra.
MOVLW 0X41 ; Se carga la letra U.
MOVWF D1
CALL VER ; Se visualiza PRU
CALL ROTAR ; Se hace el desplazamiento para dar cabida a la cuarta letra.
MOVLW 0X06 ; Se carga la letra E.
MOVWF D1
CALL VER ; Se visualiza PRUE
CALL ROTAR ; Se hace el desplazamiento para dar cabida a la quinta letra.
MOVLW 0X03 ; Se carga la letra B.
MOVWF D1
CALL VER ; Se visualiza PRUEB

; Para las otras letras es seguir la secuencia.

GOTO OTRO ; Se visualiza nuevamente.

;
ROTAR
MOVF D3,W ; D3 = D4
MOVWF D4
MOVF D2,W ; D2 = D3
MOVWF D3
MOVF D1,W ; D1 = D2
MOVWF D2
RETURN

;
VER
MOVLW 0X22 ; CONTADOR PARA VISUALIZACIÓN.
MOVWF CON3

VER_1
MOVF D1,W ; SE VISUALIZA LO QUE TIENE D1
MOVWF PORTB
MOVLW 0X08
MOVWF PORTA
CALL TIEMPO
MOVF D2,W ; SE VISUALIZA LO QUE TIENE D2
MOVWF PORTB
MOVLW 0X04
MOVWF PORTA
CALL TIEMPO
MOVF D3,W ; SE VISUALIZA LO QUE TIENE D3
MOVWF PORTB
MOVLW 0X02
MOVWF PORTA
CALL TIEMPO
MOVF D4,W ; SE VISUALIZA LO QUE TIENE D4
MOVWF PORTB

```

```

MOV LW    0X01
MOV WF    PORTA
CALL      TIEMPO
DECFSZ    CON3,1           ; SE PREGUNTA SI AUN NO SE HA VISUALIZADO
GOTO      VER_1           ; LO QUE DICE CON3
RETURN

;
TIEMPO
MOV LW    0X30           ; Valor con que se carga el contador 1.
MOV WF    CONTA1

NIVEL1
MOV LW    0X20           ; Valor con que se carga el contador 2.
MOV WF    CONTA2

NIVEL2
DECFSZ    CONTA2,1       ; Se resta en 1 el contador 2 y se pregunta si es cero.
GOTO      NIVEL2        ; Aun el contador 2 no es cero.
DECFSZ    CONTA1,1       ; Se resta en 1 el contador 1 y se pregunta si es cero.
GOTO      NIVEL1        ; Aun el contador 1 no es cero.
RETURN    ; Retorna de la subrutina.
END       ; Final del programa.

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 30) y la distribución del display.

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso tres.

3.8. Manejo de matriz 7x5 unicolor

Una vez se ha explicado el manejo de la visualización dinámica con 4 displays es posible aplicar lo aprendido con una matriz de leds.

Ejemplo 14. Se desea visualizar las palabras: **INGENIERÍA ELECTRÓNICA** en una matriz unicolor 7x5. Solo se puede utilizar el PIC 16F628A, no es posible utilizar otro integrado. A continuación, se desarrolla paso a paso dicho proceso:

Paso 1. Una matriz de puntos está construida con leds, formado por filas y columnas (Figura 40). Allí se muestra una matriz de ánodo común de 7 filas y 5 columnas. Esta matriz es como tener 5 displays de 7 segmentos cada uno. Para explicar el funcionamiento se puede realizar un ejemplo de prender uno de los leds. Por ejemplo, si se colocan 5 voltios (un lógico) en la fila 2 y 0 voltios (0 lógico) en la columna 4 se prende el led correspondiente a esa intersección

y así sucesivamente con el led o los leds que se deseen prender o apagar. Esto es lo que el microcontrolador hace; enviar datos por las filas y habilitar por las columnas.

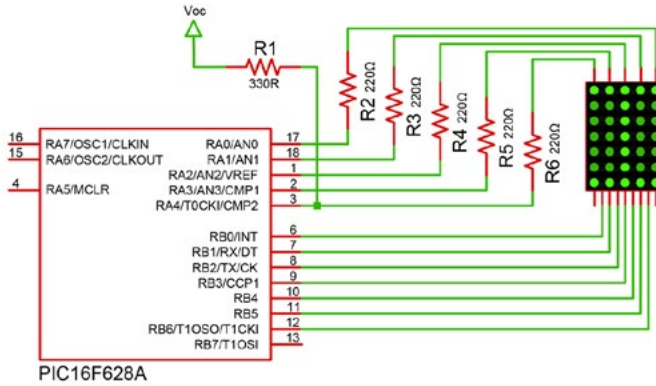


Figura 40. Diagrama de conexiones del ejemplo 14.

Fuente. Elaboración José Luis González. (La distribución de pines es toma de Proteus).

Paso 2. Hay que decir cuáles de las 15 líneas del microcontrolador se usarán para las filas de la matriz y en cuáles las columnas. Se puede tomar el PORTB para las filas y el PORTA para las columnas (ver Figura 41). En este microcontrolador se debe recordar que la línea A4 viene con el colector al aire. Por ello, el colector tiene una resistencia de $330\ \Omega$ a V_{CC} en dicha línea.

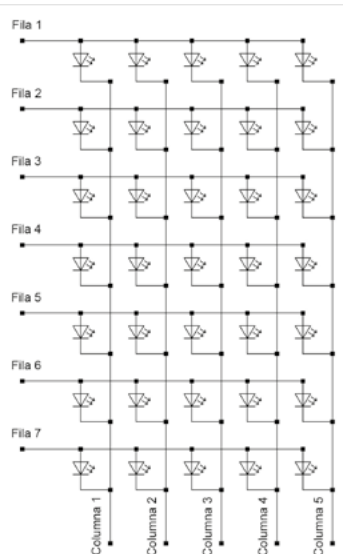


Figura 41. Diagrama de conexiones con una Matriz 7x5 cátodo común. (Proteus, 2003)

Fuente. Elaboración José Luis González

Paso 3. De acuerdo con el diagrama de conexiones planteado en el paso 2, hay que programar tanto el PORTB y el PORTA como salida. El primero para las filas y el segundo para las columnas. Para hacer el diagrama de flujo es necesario tener en cuenta que para formar una letra hay que enviar 5 códigos; uno para cada columna. En la tabla 17 sólo está la letra I, la cual se forma enviando **1** a las filas que se quieren encender y **0** a las que se quieren apagar (PORTB) y habilitando la columna correspondiente con **0** y deshabilitando con **1** (PORTA). Así que solo es seguir enviando los códigos para el resto de las letras incluyendo los espacios entre ellas, para ello es enviar **0** en todas las filas y también tener en cuenta las letras con tildes.

En este diagrama de flujo se plantea un programa principal que está encargado de llamar cada una de las subrutinas de las letras en orden hasta completar la palabra y, al final, regresa a la primera letra (ver Diagrama 11). Tener en cuenta que hay un espacio entre cada letra, para lo cual se usa la subrutina **ESPACIO**.

Este diagrama está incompleto. El lector podrá completarlo solo siguiendo la secuencia.

Tabla 17.
Códigos para la letra I con Matriz 7x5 cátodo común.

Columnas (PORTA) →		C1 A0	C2 A1	C3 A2	C4 A3	C5 A4
Filas (PORTB) ↓						
B0	F1	1	1	1	1	1

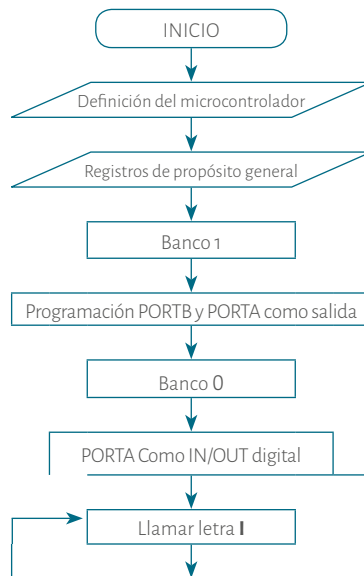
B1	F2	0	0	1	0	0
B2	F3	0	0	1	0	0
B3	F4	0	0	1	0	0
B4	F5	0	0	1	0	0
B5	F6	0	0	1	0	0
B6	F7	1	1	1	1	1
B7	X	0	0	0	0	0
Códigos al PORTB →		41 _H	41 _H	7F _H	41 _H	41 _H

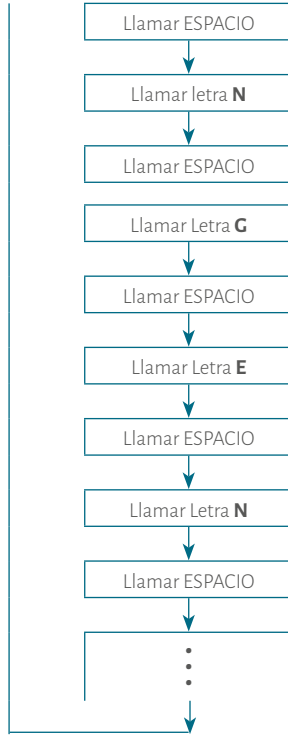
Fuente. Elaboración propia

La subrutina de cada letra está encargada de cargar los 5 códigos que forman cada una de ellas (ver Diagrama 12). Nuevamente para este programa es necesario las subrutinas **ROTA** y **VER**. Estas subrutinas son muy parecidas a las del ejemplo anterior, solo hay que hacer un ligero cambio (ver Diagramas 13 y 14).

En la subrutina **VER** se usará el mismo tiempo que se utilizó en el ejemplo anterior.

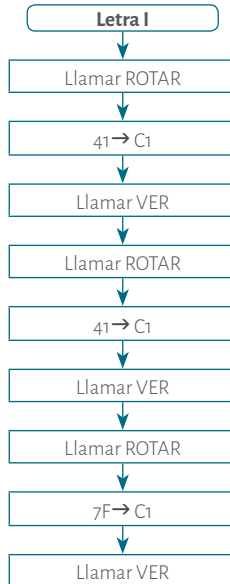
Diagrama 11.
Programa principal para visualizar una palabra completa en una matriz 7x5.

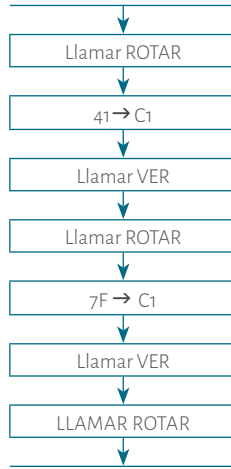


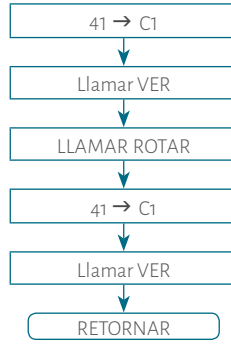


Fuente. Elaboración propia

Diagrama 12.
Subrutina de la letra I



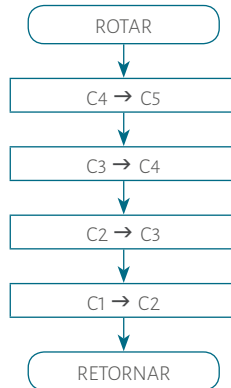




Fuente. Elaboración propia.

Diagrama 13.

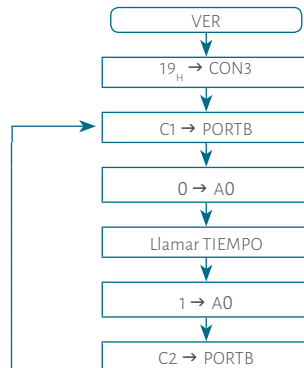
Subrutina ROTAR para una matriz 7x5 cátodo común.

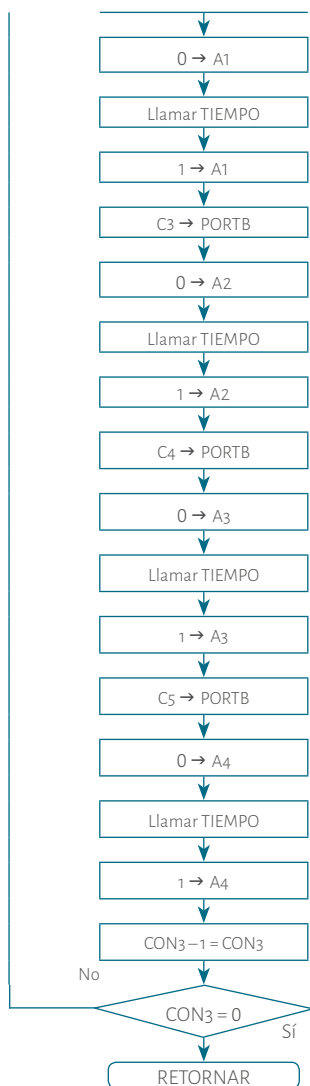


Fuente. Elaboración propia.

Diagrama 14.

Subrutina VER para una matriz 7x5 cátodo común.





Fuente. Elaboración propia.

Paso 4. El programa no está completo. Para las demás letras hay que seguir la secuencia. Hay que tener en cuenta que para las letras se usa una subrutina para cada una de ellas. Por lo tanto, solo se requiere una subrutina por letra y dentro del programa principal hay que citarla cuantas veces se requiera. El lector podrá observar que para la definición de registros de propósito general se usa **CBLOCK 0x20**, la cual sirve para iniciar la posición de los registros (20_H) y cuando se quiere finalizar se utiliza **ENDC**.

```

#include <pt6f628A.inc> ; Definición del microcontrolador es decir la librería.

CBLOCK 0X20 ; Definición de registros de propósito general.
CON1 ; CON1, CON2 y CON3 registros para tiempos.
CON2
CON3
C1 ; C1, C2, C3, C4 y C5 Para las cinco filas de la matriz.
C2
C3
C4
C5
ENDC
;

RES_VECT CODE 0X0000 ; Vector reset

BSF STATUS,5 ; banco 1 para programar puertos.
BCF STATUS,6
CLRF TRISB ; PORTB como salida para las filas de la matriz.
CLRF TRISA ; PORTA como salida para las columnas de la matriz.
BCF STATUS,5 ; Banco 0 para sacar datos.
BCF STATUS,6
MOVLW 0X07 ; Se deshabilitan los comparadores analógicos,
MOVWF CMCON ; el PORTA se toma como salida digital.

PALABRA ; PROGRAMA PRINCIPAL
CALL LETRA_I ; Se llama la letra I.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_N ; Se llama la letra N.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_G ; Se llama la letra G.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_E ; Se llama la letra E.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_N ; Se llama la letra N.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_I ; Se llama la letra I.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_E ; Se llama la letra E.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_R ; Se llama la letra R.

```

CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_I	; Se llama la letra I.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_A	; Se llama la letra I.
CALL	ESPACIO	; Espacio entre letras.
CALL	ESPACIO	; Espacio entre letras.
CALL	ESPACIO	; Espacio entre letras.
		; Para el resto de la frase seguir la secuencia.
GOTO	PALABRA	; Inicia nuevamente
;	LETRA_I	
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la primera parte de la letra.
MOVLW	0X41	; Primera parte de la letra I.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la segunda parte de la letra.
MOVLW	0X41	; Segunda parte de la letra I.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la tercera parte de la letra.
MOVLW	0X7F	; Tercera parte de la letra I.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la cuarta parte de la letra.
MOVLW	0X41	; Cuarta parte de la letra I.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la quinta parte de la letra.
MOVLW	0X41	; Quinta parte de la letra I.
MOVWF	C1	
CALL	VER	; Se visualiza.
RETURN		
;	LETRA_N	
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la primera parte de la letra.
MOVLW	0X7F	; Primera parte de la letra N.

MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la segunda parte de la letra.
MOVLW	0X04	; Segunda parte de la letra N.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la tercer parte de la letra.
MOVLW	0X08	; Tercera parte de la letra N.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la cuarta parte de la letra.
MOVLW	0X10	; Cuarta parte de la letra N.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la quinta parte de la letra.
MOVLW	0X7F	; Quinta parte de la letra N.
MOVWF	C1	
CALL	VER	; Se visualiza.
RETURN		
		; LETRA_G
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la primera parte de la letra.
MOVLW	0X3E	; Primera parte de la letra G.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la segunda parte de la letra.
MOVLW	0X41	; Segunda parte de la letra G.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la tercera parte de la letra.
MOVLW	0X51	; Tercera parte de la letra G.
MOVWF	C1	
CALL	VER	; Se visualiza.
CALL	ROTA	; Se hace el desplazamiento para dar cabida a la cuarta parte de la letra.
MOVLW	0X49	; Cuarta parte de la letra G.
MOVWF	C1	

```

CALL    VER                ; Se visualiza.

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           quinta parte de la letra.

MOVLW  0X32              ; Quinta parte de la letra G.
MOVWF  C1
CALL    VER                ; Se visualiza.
RETURN

;
LETRA_E

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           primera parte de la letra.

MOVLW  0X3E              ; Primera parte de la letra E.
MOVWF  C1
CALL    VER                ; Se visualiza.
CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           segunda parte de la letra.

MOVLW  0X49              ; Segunda parte de la letra E.
MOVWF  C1
CALL    VER                ; Se visualiza.

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           tercera parte de la letra.

MOVLW  0X49              ; Tercera parte de la letra E.
MOVWF  C1
CALL    VER                ; Se visualiza.

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           cuarta parte de la letra.

MOVLW  0X41              ; Cuarta parte de la letra E.
MOVWF  C1
CALL    VER                ; Se visualiza.

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           quinta parte de la letra.

MOVLW  0X41              ; Quinta parte de la letra E.
MOVWF  C1
CALL    VER                ; Se visualiza.
RETURN

;
LETRA_R

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           primera parte de la letra.

MOVLW  0X7E              ; Primera parte de la letra R.
MOVWF  C1
CALL    VER                ; Se visualiza.

CALL    ROTA              ; Se hace el desplazamiento para dar cabida a la
                           segunda parte de la letra.

MOVLW  0X09              ; Segunda parte de la letra R.

```

```

MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA ; Se hace el desplazamiento para dar cabida a la
           tercera parte de la letra.
MOVLW 0X19 ; Tercera parte de la letra R.
MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA
MOVLW 0X29 ; Cuarta parte de la letra R.
MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA
MOVLW 0X46 ; Quinta parte de la letra R.
MOVWF C1
CALL VER ; Se visualiza.
RETURN

;
LETRA_A
CALL ROTA
MOVLW 0X7E ; Primera parte de la letra A.
MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA
MOVLW 0X09 ; Segunda parte de la letra A.
MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA
MOVLW 0X09 ; Tercera parte de la letra A.
MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA
MOVLW 0X09 ; Cuarta parte de la letra A.
MOVWF C1
CALL VER ; Se visualiza.

CALL ROTA
MOVLW 0X7E ; Quinta parte de la letra A.
MOVWF C1
CALL VER ; Se visualiza.
RETURN

;
ESPACIO

```

```

CALL      ROTA      ; Espacio entre letras.
MOVLW    0X00
MOVWF    C1
CALL     VER        ; Se visualiza.
RETURN

;
VER

MOV LW   .25        ; Contador para control de la subrutina VER
MOV WF   CON3

VER_1

MOV F    C1,W       ; Se recupera la fila 1
MOV WF   PORTB
BCF     PORTA,0     ; Se habilita la columna 1
CALL    TIEMPO
BSF     PORTA,0     ; Se deshabilita la columna 1

MOV F    C2,W       ; Se recupera la fila 2
MOV WF   PORTB
BCF     PORTA,1     ; Se habilita la columna 2
CALL    TIEMPO
BSF     PORTA,1     ; Se deshabilita la columna 2

MOV F    C3,W       ; Se recupera la fila 3
MOV WF   PORTB
BCF     PORTA,2     ; Se habilita la columna 3
CALL    TIEMPO
BSF     PORTA,2     ; Se deshabilita la columna 3

MOV F    C4,W       ; Se recupera la fila 4
MOV WF   PORTB
BCF     PORTA,3     ; Se habilita la columna 4
CALL    TIEMPO
BSF     PORTA,3     ; Se deshabilita la columna 4

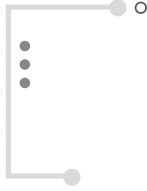
MOV F    C5,W       ; Se recupera la fila 5
MOV WF   PORTB
BCF     PORTA,4     ; Se habilita la columna 5
CALL    TIEMPO
BSF     PORTA,4     ; Se deshabilita la columna 5

DECFSZ   CON3,1    ; Decrementa el CON3 y pregunta si ya se visualizó las
                  ; 25 veces.
GOTO     VER_1     ; Aun no se ha visualizado las 25 veces.
RETURN   ; Ya terminó.

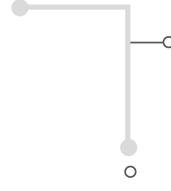
;
ROTA

MOV F    C4,W       ; C4 a C5
MOV WF   C5

```

MOVF C3,W ; C3 a C4



MOVWF C4

MOVF C2,W ; C2 a C3

MOVWF C3

MOVF C1,W ; C1 a C2

MOVWF C2

RETURN

; TIEMPO

MOVLW 0X1A ; Subrutina de tiempo.

MOVWF CON1 ; De acuerdo al diagrama de flujo hasta que

NIVEL1 ; los dos contadores estén en cero no retorna.

MOVLW 0X1C

MOVWF CON2

NIVEL2

DECFSZ CON2,1

GOTO NIVEL2

DECFSZ CON1,1

GOTO NIVEL1

RETURN

;

END ; Fin del programa

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 30) y la distribución de la matriz.

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso tres.



Capítulo IV



En el capítulo anterior se han realizado varios ejercicios sobre el manejo de entradas y salidas. Ahora se realizan ejemplos para profundizar un poco más en la programación de microcontroladores de microchip. Por lo tanto, se trabajarán distintas aplicaciones: manejo de teclado matricial, manejo de librerías, de LCD, de EEPROM data y de motor paso a paso.

4.1. Manejo del teclado matricial y display 7 segmentos

A continuación se desarrolla paso a paso el manejo de un teclado matricial 4x3 con visualización en un display 7 segmentos de ánodo común.

Ejemplo 15. Se desea usar un display de 7 segmentos de ánodo común para visualizar los números de cero (0) a nueve (9), toda vez que se oprima la tecla respectiva desde un teclado numérico 4x3 que es matricial. A continuación se desarrollará paso por paso:

Paso 1. En el ejemplo 11 se trabajó con un display de 7 segmentos de ánodo común. Por lo que se conoce su funcionamiento, sólo se debe definir cómo funciona un teclado matricial 4x3. Palacios explica que los teclados son interruptores normalmente abiertos conectados entre filas y columnas. En la Figura 42 se muestra un teclado matricial 4x4, donde hay 4 filas y 4 columnas (Palacios, 2006).

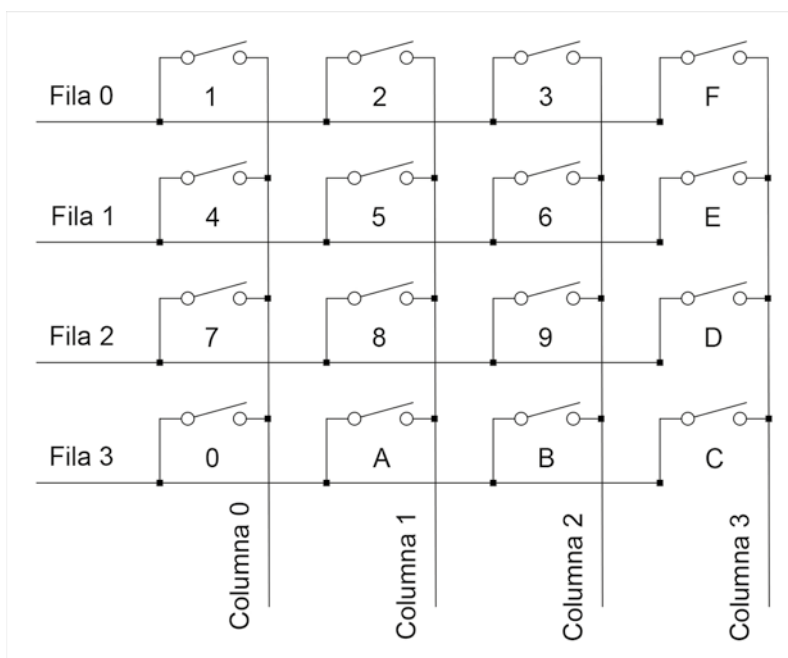


Figura 42. Esquema de teclado numérico matricial 4x4.

Fuente. Elaboración José Luis González.

El funcionamiento del teclado es muy sencillo; cuando se oprimen uno de los interruptores se une una fila con una columna. Por ejemplo, al oprimir el pulsador marcado con la tecla 3 se une la fila 0 con la columna 2 o, si se oprime el pulsador marcado con el número 0, se une la fila 3 con la columna 0 y así sucesivamente. Aquí hay que entender qué son pulsadores. Por lo tanto, el programador le da el nombre a cada uno, pues la tecla marcada con la letra A puede tomar otro nombre o una función que, por ejemplo, puede ser cambiar a letras mayúsculas.

Así, el microcontrolador habilita las filas y monitorea las columnas o viceversa. En otras palabras, para el primer caso, el dispositivo coloca en las filas la combinación en binario 0111 desde la fila 0 hasta la 4 y, en ese momento, se habilita la fila 0. Por consiguiente, sólo quedan por monitorear las 4 columnas en las que están habilitadas las teclas marcadas con (1), (2) (3) y (F). Para la siguiente fila la combinación es 1011 y las teclas habilitadas son (4), (5) (6) y (E) y así sucesivamente. El programador debe tener en cuenta que hay que garantizar las columnas en *pull-up*, utilizando resistencias a Vcc. Es decir, a los mismos 5 voltios de la fuente.

Paso 2. Hay que indicar en cuáles de las quince líneas del microcontrolador se van a poner los 7 segmentos del display. Se puede tomar el PORTB, desde B_0 hasta B_6 , una línea para cada segmento, así como se hizo en el ejemplo 11. En este mismo puerto se puede conectar el teclado así: las filas desde B_0 hasta B_3 y desde B_4 hasta B_6 para las columnas. De esa forma este puerto va a ser usado tanto de entrada como de salida, es decir, para manejar el display será de salida y para manejar el teclado; la parte baja (B_0 - B_3) será de salida (filas); y la parte alta (B_3 - B_6) de entrada (columnas). Ver figura 43.

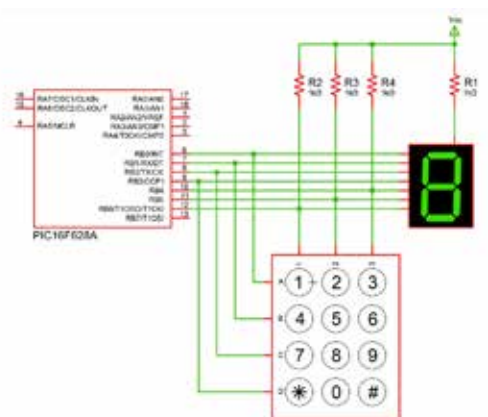


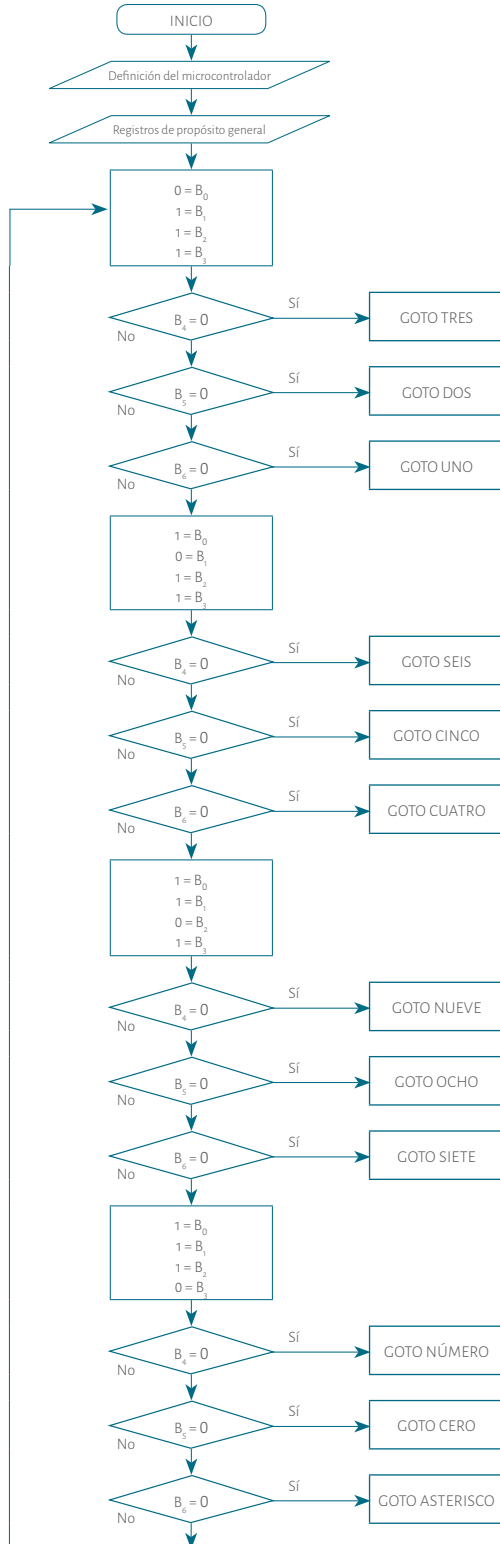
Figura 43. Diagrama de conexiones del ejemplo 15.

Fuente. Elaboración José Luis González. (La distribución de pines es tomada de Proteus).

Paso 3. Para hacer el diagrama de flujo hay que saber qué códigos obtener para cada número, estos se pueden ver en la Tabla 15 del Ejemplo 11. En el Diagrama 15 se observa el diagrama de flujo del programa principal. En este diagrama se ve claramente cómo se habilita cada una de las filas y, cada vez que se habilita una de ellas, se puede observar cómo va el proceso preguntando por cada una de las columnas.

Para cada uno de los números se usa una subrutina, en ellas se obtiene por el Puerto B el código correspondiente del número, pero antes este Puerto es programado como salida, después se programa de entrada (para columnas, B_4 - B_6) y salida (para filas, B_0 - B_3), se da un tiempo y por último se regresa al inicio del programa principal, en el Diagrama 16 se muestra para el número 1, para los otros números solo es cambiar el código para cada uno, también se observa las subrutinas de programación del PORTB tanto de salida como de entrada-salida.

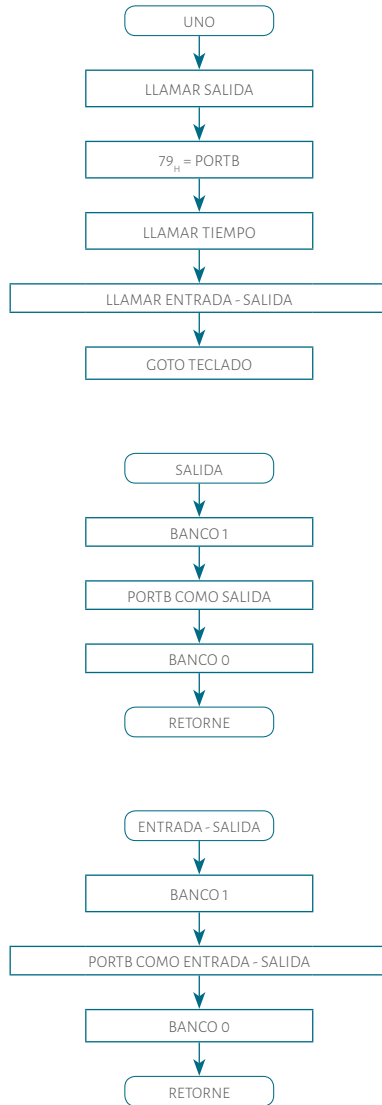
Diagrama 15.
Programa para manejo de un teclado matricial
y visualizar en display siete segmentos.



Fuente. Elaboración propia.

Diagrama 16.

Subrutina para el número uno del ejemplo 15.



Fuente. Elaboración propia.

Paso 4. El programa es el siguiente: la subrutina de tiempo es la misma de los anteriores programas. Si se quiere más tiempo entonces se debe aumentar el valor de los contadores. Recuerde que puede cargarlos desde 00_H hasta FF_H .

#INCLUDE<p16f628A.inc>

; Definición del microcontrolador es decir la librería.

CBLOCK

0x20

; Definición de registros de propósito general.


```

CON1
CON2
ENDC
;

RES_VECT CODE 0X0000 ; Vector reset

TECLADO ; PROGRAMA PRINCIPAL

BCF PORTB,0 ; SE HABILITAN LAS TECLAS 3,2,1
BSF PORTB,1
BSF PORTB,2
BSF PORTB,3

BTFFS PORTB,4 ; ¿OPRIMEN EL TRES?
GOTO TRES ; OPRIMIERON EL TRES

BTFFS PORTB,5 ; ¿OPRIMEN EL DOS?
GOTO DOS ; OPRIMIERON EL DOS

BTFFS PORTB,6 ; ¿OPRIMEN EL UNO?
GOTO UNO ; OPRIMIERON EL UNO
;

BSF PORTB,0 ; SE HABILITAN LAS TECLAS 6,5,4
BCF PORTB,1
BSF PORTB,2
BSF PORTB,3

BTFFS PORTB,4 ; ¿OPRIMEN EL SEIS?
GOTO SEIS ; OPRIMIERON EL SEIS

BTFFS PORTB,5 ; ¿OPRIMEN EL CINCO?
GOTO CINCO ; OPRIMIERON EL CINCO

BTFFS PORTB,6 ; ¿OPRIMEN EL CUATRO?
GOTO CUATRO ; OPRIMIERON EL CUATRO
;

BSF PORTB,0 ; SE HABILITAN LAS TECLAS 9,8,7
BSF PORTB,1
BCF PORTB,2
BSF PORTB,3

BTFFS PORTB,4 ; ¿OPRIMEN EL NUEVE?
GOTO NUEVE ; OPRIMIERON EL NUEVE

BTFFS PORTB,5 ; ¿OPRIMEN EL OCHO?
GOTO OCHO ; OPRIMIERON EL OCHO

```

; CON1, CON2 para la subrutina tiempo.

; Vector reset

; PROGRAMA PRINCIPAL

; SE HABILITAN LAS TECLAS 3,2,1

; ¿OPRIMEN EL TRES?

; OPRIMIERON EL TRES

; ¿OPRIMEN EL DOS?

; OPRIMIERON EL DOS

; ¿OPRIMEN EL UNO?

; OPRIMIERON EL UNO

; SE HABILITAN LAS TECLAS 6,5,4

; ¿OPRIMEN EL SEIS?

; OPRIMIERON EL SEIS

; ¿OPRIMEN EL CINCO?

; OPRIMIERON EL CINCO

; ¿OPRIMEN EL CUATRO?

; OPRIMIERON EL CUATRO

; SE HABILITAN LAS TECLAS 9,8,7

; ¿OPRIMEN EL NUEVE?

; OPRIMIERON EL NUEVE

; ¿OPRIMEN EL OCHO?

; OPRIMIERON EL OCHO

```

BTFS    PORTB,6           ;¿OPRIMEN EL SIETE?
GOTO    SIETE            ; OPRIMIERON EL SIETE

;

BSF     PORTB,0           ; SE HABILITAN LAS TECLAS #,0*
BSF     PORTB,1
BSF     PORTB,2
BCF     PORTB,3

BTFS    PORTB,4           ;¿OPRIMEN EL NUMERO?
GOTO    NUMERO          ; OPRIMIERON EL NÚMERO

BTFS    PORTB,5           ;¿OPRIMEN EL CERO?
GOTO    CERO            ; OPRIMIERON EL CERO

BTFS    PORTB,6           ;¿OPRIMEN EL ASTERISCO?
GOTO    ASTERISCO       ; OPRIMIERON EL ASTERISCO
GOTO    TECLADO         ; NADA SE HA PRESIONADO, VUELVE A REVISAR

;
UNO

CALL    SALIDA           ; Se programa puerto B de salida.
MOVLW  0X79
MOVWF  PORTB            ; Se obtiene el dato por el puerto B
CALL    TIEMPO
CALL    SALIDA_ENTRADA  ; Se programa puerto B de entrada salida.
GOTO    TECLADO         ; Regresa el programa principal.

;
DOS

CALL    SALIDA           ; Se programa puerto B de salida.
MOVLW  0X24
MOVWF  PORTB            ; Se obtiene el dato por el puerto B.
CALL    TIEMPO
CALL    SALIDA_ENTRADA  ; Se programa Puerto B de Entrada salida.
GOTO    TECLADO         ; Regresa el programa principal.

;
TRES

CALL    SALIDA           ; Se programa puerto B de salida.
MOVLW  0X30
MOVWF  PORTB            ; Se obtiene el dato por el puerto B.
CALL    TIEMPO
CALL    SALIDA_ENTRADA  ; Se programa puerto B de entrada salida.
GOTO    TECLADO         ; Regresa el Programa Principal.

;
CUATRO

CALL    SALIDA           ; Se programa puerto B de salida.
MOVLW  0X19
MOVWF  PORTB            ; Se obtiene el dato por el puerto B.
CALL    TIEMPO
CALL    SALIDA_ENTRADA  ; Se programa puerto B de entrada salida.
GOTO    TECLADO         ; Regresa el programa principal.

;

```

CINCO

```
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X12
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
```

;

SEIS

```
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X02
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
```

;

SIETE

```
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X78
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
```

;

OCHO

```
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X00
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
```

;

NUEVE

```
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X10
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el Programa Principal.
```

;

CERO

```
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X40
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
```

;

```

NUMERO
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X09
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
;
ASTERISCO
CALL SALIDA ; Se programa puerto B de salida.
MOVLW 0X1B
MOVWF PORTB ; Se obtiene el dato por el puerto B.
CALL TIEMPO
CALL SALIDA_ENTRADA ; Se programa puerto B de entrada salida.
GOTO TECLADO ; Regresa el programa principal.
;
SALIDA
BSF STATUS,5 ; BANCO 1
BCF STATUS,6
CLRF TRISB ; PORTB COMO SALIDA PARA EL DISPLAY
BCF STATUS,5 ; BANCO 0
BCF STATUS,6
RETURN
;
SALIDA_ENTRADA
BSF STATUS,5 ; BANCO 1
BCF STATUS,6
MOVLW 0XF0 ; PORTB COMO ENTRDA/SALIDA, SALIDA PARA
; LAS FILAS (B0 A B3)
MOVWF TRISB ; ENTRADA PARA LAS COLUMNAS (B4 A B7)
BCF STATUS,5 ; BANCO 0
BCF STATUS,6
RETURN
;
TIEMPO
MOVLW 0X1A ; Subrutina de tiempo.
MOVWF CON1 ; De acuerdo al diagrama de flujo hasta que
NIVEL1 ; los dos contadores estén en cero no retorna.
MOVLW 0X1C
MOVWF CON2
NIVEL2
DECFSZ CON2,1
GOTO NIVEL2
DECFSZ CON1,1
GOTO NIVEL1
RETURN
;
END ; Fin del programa

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que conocer la distribución del microcontrolador (ver figura 30), la distribución del teclado y la del display.

Paso 6. Si se requiere, realice las respectivas correcciones y vuelva al paso tres.

4.2 Creación y manejo de librerías

Cada vez que se avanza en la programación y en las aplicaciones de estos microcontroladores se complica un poco el programa. No obstante, es preciso recordar que si usted organiza su aplicación en un programa principal y éste hace uso de unas subrutinas, es más fácil la programación. Ahora bien, la programación se facilita mucho más si se utilizan de las librerías, pues éstas se pueden construir y citar en las aplicaciones o, en algunos casos, se pueden tomar librerías creadas por otras personas y hacer uso de ellas, respetando los derechos de autor para evitar los problemas de plagio. Para que el lector se familiarice con la construcción y uso de las librerías se realizará un ejemplo.

Ejemplo 16. Se desea que el ejemplo 14, haciendo uso de la misma conexión, utilice una librería con las siguientes subrutinas: PROG_PUERTOS, DATO, ESPACIO, VER, ROTA y TIEMPO. Para el desarrollo de este ejemplo además de seguir los 6 pasos de diseño, hay que incluir 3 pasos más, los cuales se refieren a la construcción y al manejo de librerías. A continuación se describen estos pasos:

4.2.1. Paso para la construcción de librerías en MPLAB

Paso 1, para librerías. Escribir la librería como un programa en assembler. Lo primero que se hace es definir los registros de propósito general que se van usar en la librería, usando el comando **CBLOCK** y terminado con **ENDC**. Posteriormente, se escribe la librería con todas sus etiquetas (subrutinas) y, al final, no hay que escribir END. En este caso no hay que incluir la librería del microcontrolador con el que se va a trabajar. Para realizar librerías que funcionen bien el programador tiene que haber adquirido bastante destreza para realizar programas, pues si realiza un proceso inadecuado ésta no funciona adecuadamente.

Paso 2, para librerías. Guardar el archivo como *.inc* en la siguiente ruta:

C:\Program Files\Microchip\MPLABX\v3.65\mpasmx

Para ello puede usar **MPLABX** o Bloc de notas, recordando que debe guardarlo como .inc

Paso 3, para librerías. Incluir la librería en el programa principal usando #include <xxxx.inc>, y seguir los 6 pasos de diseño propuestos en el ejemplo 8.

A continuación, se desarrollarán los 3 pasos propuestos:

Paso 1, para librerías. Para iniciar a escribir la librería hay que abrir **MPLABX**, así como se vió en el ejemplo 7. Es importante recordar que después de los puntos y coma los enunciados serán tomados como comentarios del programador.

El programa de la librería es el siguiente:

```

CBLOCK                                ; Definición de registros de propósito general.
CON1                                  ; CON1, CON2 y CON3 registros para tiempos.
CON2
CON3
C1                                     ; C1, C2, C3, C4 y C5 Para las cinco filas de la matriz.
C2
C3
C4
C5
AUX
ENDC
;

PROG_PUERTO

    BSF    STATUS,5                    ; Banco 1 para programar puertos.
    BCF    STATUS,6
    BCF    TRISA,0                      ; PORTA como salida para las columnas de la matriz.
    BCF    TRISA,1
    BCF    TRISA,2
    BCF    TRISA,3
    BCF    TRISA,4

    CLRF   TRISB                        ; PORTB como salida para las filas de la matriz.

    BCF    STATUS,5                    ; Banco 0 para sacar datos.
    BCF    STATUS,6
    MOVLW  0X07                          ; Se deshabilitan los comparadores analógicos,
    MOVWF  CMCON                          ; el PORTA se toma como salida digital.

    RETURN

;
DATO

```

```

MOVWF  AUX           ; Guarda la parte de la letra
CALL   ROTA         ; Va a la rutina ROTA
MOVF   AUX,W        ; Recupera la parte de la letra.
MOVWF  C1
CALL   VER          ; Va a visualizar.
RETURN

;
ESPACIO
CALL   ROTA         ; Espacio entre letras.
MOVLW  0X00
MOVWF  C1
CALL   VER          ; Se visualiza.
RETURN

;
VER
MOVLW  .25          ; Contador para control de la subrutina VER
MOVWF  CON3
VER_1
MOVF   C1,W        ; Se recupera la fila 1
MOVWF  PORTB
BCF    PORTA,0     ; Se habilita la columna 1
CALL   TIEMPO
BSF    PORTA,0     ; Se deshabilita la columna 1

MOVF   C2,W        ; Se recupera la fila 2
MOVWF  PORTB
BCF    PORTA,1     ; Se habilita la columna 2
CALL   TIEMPO
BSF    PORTA,1     ; Se deshabilita la columna 2

MOVF   C3,W        ; Se recupera la fila 3
MOVWF  PORTB
BCF    PORTA,2     ; Se habilita la columna 3
CALL   TIEMPO
BSF    PORTA,2     ; Se deshabilita la columna 3

MOVF   C4,W        ; Se recupera la fila 4
MOVWF  PORTB
BCF    PORTA,3     ; Se habilita la columna 4
CALL   TIEMPO
BSF    PORTA,3     ; Se deshabilita la columna 4

MOVF   C5,W        ; Se recupera la fila 5
MOVWF  PORTB
BCF    PORTA,4     ; Se habilita la columna 5
CALL   TIEMPO
BSF    PORTA,4     ; Se deshabilita la columna 5

```

	DEFSZ	CON3,1	; Decrementa el CON3 y pregunta si ya se visualizó las 25 veces.
	GOTO	VER_1	; Aún no se ha visualizado las 25 veces.
	RETURN		; Ya terminó.
;			
ROTA	MOVF	C4,W	; C4 a C5
	MOVWF	C5	
	MOVF	C3,W	; C3 a C4
	MOVWF	C4	
	MOVF	C2,W	; C2 a C3
	MOVWF	C3	
	MOVF	C1,W	; C1 a C2
	MOVWF	C2	
	RETURN		
;			
TIEMPO	MOVLW	0X1A	; Subrutina de tiempo.
	MOVWF	CON1	; De acuerdo con el diagrama de flujo hasta que
NIVEL1			; los dos contadores estén en cero no retorna.
	MOVLW	0X1C	
	MOVWF	CON2	
NIVEL2	DEFSZ	CON2,1	
	GOTO	NIVEL2	
	DEFSZ	CON1,1	
	GOTO	NIVEL1	
	RETURN		

A continuación se explicará brevemente cada una de las subrutinas de la librería (ver la tabla 18).

Tabla 18.
Subrutina de la librería *libreria_Matriz_7x5*.

Subrutina	Comentario
PROG_PUERTO	Esta subrutina se encarga de programar tanto el PORTA y PORTB de salida.
DATO	Esta subrutina se encarga de mostrar una parte de cada letra.
ESPACIO	Esta subrutina se encarga de dejar un espacio entre letras.
VER	Esta subrutina se encarga de recuperar lo que tiene cada una de las columnas, pasarla al PORTB y habilitarla en el PORTA.
ROTA	Esta subrutina se encarga de rotar el dato de todas las columnas.
TIEMPO	Esta subrutina se encarga de dar tiempo para visualización dinámica.

Fuente. Elaboración propia.

Paso 2, para librerías. Guardar el archivo como `.inc` en la siguiente ruta:

C:\Program Files\Microchip\MPLABX\v3.65\mpasmx

Recordar que puede usar **MPLABX** o Bloc de notas y que debe guardarlo como `.inc`.

Para este ejemplo el archivo se guardó con nombre ***Lib_Matriz.inc***. De esa forma cuando se requiera puede ser citada con ese nombre.

Paso 3, para librerías. Incluir la librería en el programa principal usando ***#include <Lib_Matriz.inc>***

Una vez vistos los pasos para escribir una librería se puede continuar con el desarrollo del ejemplo usando las subrutinas de la librería recién creada y, desde luego, hacer la cita desde el programa principal. A continuación se desarrollará el ejemplo paso a paso:

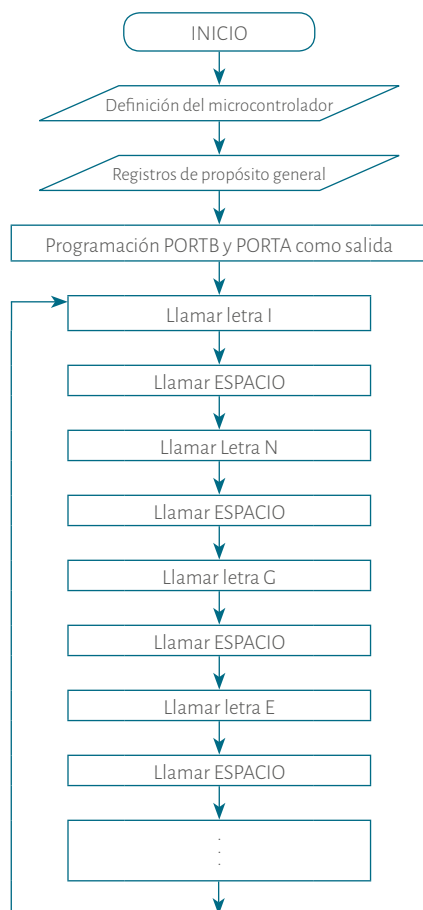
Paso 1. Para este ejemplo ya se explicó cómo funcionan las matrices de cátodo común 7x5. Por lo tanto, no se van a explicar. Se recomienda leer nuevamente el ejemplo 14.

Paso 2. Así como se planteó al inicio del ejemplo se va a usar la misma conexión del ejemplo 14, (ver figura 41).

Paso 3. Para este ejemplo se usará un programa principal y unas subrutinas. El primero se muestra en el Diagrama de flujo 17, en el cual se observa cómo usar la librería y una de las subrutinas se ve en el Diagrama de flujo 18 donde se muestra la letra I. Los cinco códigos de cada letra siguen siendo iguales. Por lo tanto, se recomienda al lector ver la tabla 17. Así como en el ejemplo 14 hay que dejar un espacio entre cada letra. Este diagrama está incompleto, pero el lector podrá completarlo siguiendo la secuencia.

Diagrama 17.

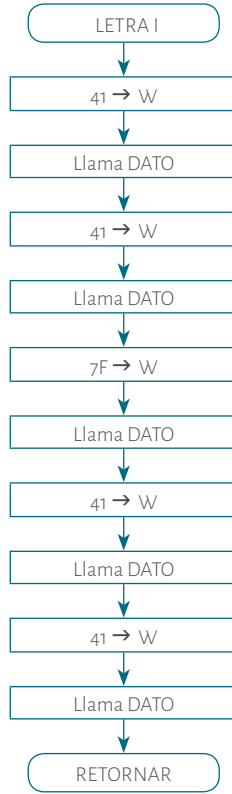
Programa principal para uso de librerías para manejo de una matriz 7x5.



Fuente. Elaboración propia.

Paso 4. El programa se muestra después del diagrama de flujo 18, hay que seguir la secuencia para las demás letras ya que no está completo. El lector puede notar cómo utiliza la librería al final del programa.

Diagrama 18.
Subrutina de la letra I



Fuente. Elaboración propia

```

#include<pt6f628A.inc> ; Definición del microcontrolador es decir la librería.

CBLOCK 0X20 ; Definición de registros de propósito general.

ENDC
;
RES_VECT CODE 0X0000 ; Vector reset

CALL PROC_PUERTO

PALABRA ; PROGRAMA PRINCIPAL

CALL LETRA_I ; Se llama la subrutina de la letra I, desde la librería.
CALL ESPACIO ; Espacio entre letras.

CALL LETRA_N ; Se llama la subrutina de la letra N, desde la librería.
CALL ESPACIO ; Espacio entre letras.
  
```

CALL	LETRA_G	; Se llama la subrutina de la letra G, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_E	; Se llama la subrutina de la letra E, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_N	; Se llama la subrutina de la letra N, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_I	; Se llama la subrutina de la letra I, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_E	; Se llama la subrutina de la letra E, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_R	; Se llama la subrutina de la letra R, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_I	; Se llama la subrutina de la letra I, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	LETRA_A	; Se llama la subrutina de la letra A, desde la librería.
CALL	ESPACIO	; Espacio entre letras.
CALL	ESPACIO	; Espacio entre letras.
		; Para el resto de la frase es seguir la secuencia.
GOTO	FRASE	; Inicia nuevamente la frase.
;		
LETRA_I		
MOVLW	0X41	; Primera parte de la letra I.
CALL	DATO	; Se llama la subrutina DATO, desde la librería.
MOVLW	0X41	; Segunda parte de la letra I.
CALL	DATO	; Se llama la subrutina DATO, desde la librería.
MOVLW	0X7F	; Tercera parte de la letra I.
CALL	DATO	; Se llama la subrutina DATO, desde la librería.
MOVLW	0X41	; Cuarta parte de la letra I.
CALL	DATO	; Se llama la subrutina DATO, desde la librería.
MOVLW	0X41	; Quinta parte de la letra I.
CALL	DATO	; Se llama la subrutina DATO, desde la librería.
RETURN		
;		
LETRA_N		
MOVLW	0X7F	; Primera parte de la letra N.

```

CALL      DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X04      ; Segunda parte de la letra N.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X08      ; Tercera parte de la letra N.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X10      ; Cuarta parte de la letra N.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X7F      ; Quinta parte de la letra N.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.
RETURN

;
LETRA_G

MOVWLW   0X3E      ; Primera parte de la letra G.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X41      ; Segunda parte de la letra G.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X51      ; Tercera parte de la letra G.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X49      ; Cuarta parte de la letra G.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X32      ; Quinta parte de la letra G.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.
RETURN

;
LETRA_E

MOVWLW   0X3E      ; Primera parte de la letra E.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X49      ; Segunda parte de la letra E.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X49      ; Tercera parte de la letra E.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X41      ; Cuarta parte de la letra E.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.

MOVWLW   0X41      ; Quinta parte de la letra E.
CALL     DATO      ; Se llama la subrutina DATO, desde la librería.
RETURN

```

;

```

LETRA_R
    MOVLW    0X7E           ; Primera parte de la letra R.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X09           ; Segunda parte de la letra R.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X19           ; Tercera parte de la letra R.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X29           ; Cuarta parte de la letra R.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X46           ; Quinta parte de la letra R.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.
    RETURN

;
LETRA_A
    MOVLW    0X7E           ; Primera parte de la letra A.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X09           ; Segunda parte de la letra A.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X09           ; Tercera parte de la letra A.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X09           ; Cuarta parte de la letra A.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.

    MOVLW    0X7E           ; Quinta parte de la letra A.
    CALL     DATO           ; Se llama la subrutina DATO, desde la librería.
    RETURN

;
#include<Lib_Matriz.inc>

END

```

Paso 5. Programar el microcontrolador y montarlo en el mismo circuito del Ejemplo 14.

Paso 6. Hacer las correcciones y volver al paso 3. Para este programa hay que tener en cuenta que se están usando las subrutinas de la librería, pues si hay algún problema o error en algunas de estas subrutinas el programa no funcionará.

El lector podrá realizar sus propias librerías y podrá conseguir librerías hechas por otras personas. En ambos casos hay que guardar las librerías en la dirección vista en el paso 2 para librerías. Cuando se requiera usar una librería diseñada por otra persona se recomienda hacer la respectiva cita. En el ejemplo 18 justamente usa una librería diseñada por otra persona.

4.3. Manejo de LCD

Hasta el momento se han trabajado display 7 segmentos multiplexados y una matriz de puntos, ya es hora de hacer algunas aplicaciones con una LCD. En el mercado hay muchas variedades tanto de tamaño, como de precios y de colores. En el siguiente ejemplo se trabajara con una LCD de 16 X 2.

Ejemplo 17. Se desea visualizar una frase completa en una LCD de 16 X 2 y, posteriormente, que el microcontrolador quede prendiendo y apagando un led como se muestra en el ejemplo 8. La frase puede estar distribuida así:

Línea 1 **"Prueba LCD 16x2"**

Línea 2 **"con PIC16F628A"**

A continuación se desarrollará paso por paso:

Paso 1. Según Palacios 2006, las LCD de 16 X 2, están construidas por dos líneas de 16 caracteres cada una, y estos están normalmente formados por una matriz de 5x8. De esta forma, se puede visualizar casi cualquier carácter (Palacios, 2006,). Sin embargo, las que describe Palacios difieren con relación a las existentes en el mercado nacional, ya que éstas últimas contienen 2 pines mas; es decir 16 pines, estos dos últimos son un LED (Back Light), pin15 ánodo (Vcc) y pin 16 cátodo (GND). La descripción general de los pines se muestra en la tabla 19.

Tabla 19.
Distribución de pines LCD 2x16.

Pin	Símbolo	Función
1	V_{SS}	0 V
2	V_{DD}	5 V
3	V_O	Ajuste de Contraste
4	RS	Selector de modo

5	R/\overline{W}	Selector Lectura/ $\overline{\text{Escritura}}$
6	E	Enable
7	DB_0	Línea de datos D_0
8	DB_1	Línea de datos D_1
9	DB_2	Línea de datos D_2
10	DB_3	Línea de datos D_3
11	DB_4	Línea de datos D_4
12	DB_5	Línea de datos D_5
13	DB_6	Línea de datos D_6
14	DB_7	Línea de datos D_7
15	A/V_{EE}	5 V
16	K	0 V

Fuente. Adaptado de XIAMEN AMOTEC DISPLAY. CO. LTD. (2008). SPECIFICATIONS OF LCD MODULE. MODULE NO: ADM1602K-NSW-FBS/3.3V

Según Palacios, las LCD tienen dos tipos de memorias: DDRAM donde se almacenan los datos que se quieren visualizar y CGROM donde se almacenan los datos que se pueden visualizar. La primera memoria tiene una capacidad de 80 caracteres, 40 para cada línea, de los cuales solo se pueden visualizar los 16 primeros de cada una: la línea 1 inicia en la posición 00H, y la línea 2 inicia en la posición 40H (ver Figura 43). Cada vez que se escribe un dato, automáticamente, se apunta a la siguiente posición. La segunda memoria tiene almacenados 192 caracteres, cada uno tiene asignado un número en binario de 8 bits. Este número es el código ASCII. En la tabla 20 se muestran los códigos más usados en una LCD. El lector podrá observar cómo se obtiene los 8 bits para cada símbolo. Por ejemplo, si se quiere la letra E el código es: '0100 0101'.

Datos Visibles en la LCD																					
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	27	Línea 1
40	41	42	43	44	45	46	47	48	49	4 ^a	4B	4C	4D	4E	4F	50	51	52	67	Línea 2

Figura 43. Distribución de la memoria DDRAM.

Fuente. Elaboración propia. Datos tomados de Palacios, 2006

La LCD tiene dos modos de funcionamiento fundamentales:

1. **Modo comando:** cuando en el bus de datos la LCD recibe un comando como: borrar display, mover cursor, entre otras. ($RS = 0$ y $R\overline{W} = 0$). Una

operación de este modo tarda un máximo de 1.64 ms. Los modos comando que puede recibir la LCD se muestran en la tabla 21 (Palacios, 2006, p.191).

2. **Modo carácter o dato:** cuando en el bus de datos la LCD recibe un carácter a guardar en la DDRAM ($RS = 1$ y $R\bar{W} = 0$). Una operación de este modo tarda un máximo de 40 μs (Palacios 2006, p.191).

Tabla 20.

Códigos ASCII más usados.

Parte alta Parte baja	0010 XXXX	0011 XXXX	0100 XXXX	0101 XXXX	0110 XXXX	0111 XXXX
XXXX 0000		0	@	P	`	p
XXXX 0001	!	1	A	Q	a	q
XXXX 0010	“	2	B	R	b	r
XXXX 0011	#	3	C	S	c	s
XXXX 0100	\$	4	D	T	d	t
XXXX 0101	%	5	E	U	e	u
XXXX 0110	&	6	F	V	f	v
XXXX 0111	‘	7	G	W	g	w
XXXX 1000	(8	H	X	h	x
XXXX 1001)	9	I	Y	i	y
XXXX 1010	*	:	J	Z	j	z
XXXX 1011	+	;	K	[k	{
XXXX 1100	,	<	L	¥	l	
XXXX 1101	-	=	M]	m	}
XXXX 1110	.	>	N	^	n	®
XXXX 1111	/	?	O	_	o	¬

Fuente. Elaboración propia, adaptado de Adaptado de XIAMEN AMOTEC DISPLAY. CO. LTD. (2008). SPECIFICATIONS OF LCD MODULE. MODULE NO: ADM1602K-NSW-FBS/3.3V (p.14)

Tabla 21.

Modos de una LCD 2x16.

Comando	RS	R/\bar{W}	DB ₇	DB ₆	DB ₅	DB ₄	DB ₃	DB ₂	DB ₁	DB ₀
<i>Clear isplay</i>	0	0	0	0	0	0	0	0	0	1
<i>Return Home</i>	0	0	0	0	0	0	0	0	1	X
<i>Entry Modo Set</i>	0	0	0	0	0	0	0	1	I/D	S

<i>Display Control</i>	0	0	0	0	0	0	1	D	C	B
<i>Cursor and Display Shift</i>	0	0	0	0	0	1	S/C	R/L	X	X
<i>Funtion Set</i>	0	0	0	0	1	DL	N	F	X	X
<i>Set DDRAM</i>	0	0	1	DDRAM Address						
<i>Write RAM</i>	1	0	Write Data							

Fuente. Adaptado de XIAMEN AMOTEC DISPLAY. CO. LTD. (2008). SPECIFICATIONS OF LCD MODULE. MODULE NO: ADM1602K-NSW-FBS/3.3V

Donde:

Clear Display: borra pantalla.

Return Home: regresa el cursor a la posición 00H.

Entry Modo Set: modo de entrada

S = 0, la información visualizada en la pantalla no se desplaza al ingresar un nuevo dato.

S = 1, la información visualizada en la pantalla se desplaza al ingresar un nuevo dato.

I/D = 0, incremento automático de la posición del cursor.

I/D = 1, decremento automático de la posición del cursor (Palacios, 2006, p.193).

Display Control: control de la pantalla

B = 0, Blink OFF, parpadeo cursor apagado.

B = 1, Blink ON, parpadeo cursor prendido.

C = 0, Cursor OFF, cursor apagado.

C = 1, Cursor ON, cursor prendido.

D = 0, Display OFF, display apagado.

D = 1, Display ON, display prendido.

Cursor and Display Shift: control de los desplazamientos del cursor y de la pantalla (Palacios, 2006, p.193)

R/L = 0, a la izquierda.

R/L = 1, a la derecha.

S/C = 0, el desplazamiento se aplica sobre el cursor.

S/C = 1, el desplazamiento se aplica sobre el cursor, sin afectar la DDRAM.

S/C=0, el desplazamiento aplica sobre todo el display.
(Palacios, 2006, p.193)

Function Set:

características de función de *Hardware*.

F=0, caracteres de 5x7.

F=1, caracteres de 5x10.

N=0, pantalla de 1 línea.

N=1, pantalla de 2 líneas.

DL=0, conexión de bus de 4 bits (DB4 a DB7).

DL=1, conexión de bus de 8 bits (DB0 a DB7) (Cf., Palacios, 2006, p. 193)

Así como se explicó en *Function Set* la LCD se puede conectar en bus de 4 bits (**DL=0**) o de 8 bits (**DL=1**). Para este ejemplo se conectará de la segunda forma es decir a 8 bits. Además sólo se desea escribir en la LCD. Por esta razón, se le garantiza la línea **R/W=0** y se ahorra un pin del microcontrolador.

Paso 2. La distribución del circuito puede ser de la siguiente forma: el puerto B para el bus de la LCD: **B₀** con **DB₀** (pin 7), **B₁** con **DB₁** (pin 8) y así sucesivamente. Las líneas de control **RS** (pin 4) y **E** (pin 6) están conectadas así: **A₀=RS** y **A₁=E**. El led se puede colocar en **A₃**. En la figura 44 se puede apreciarla distribución de estos circuitos.

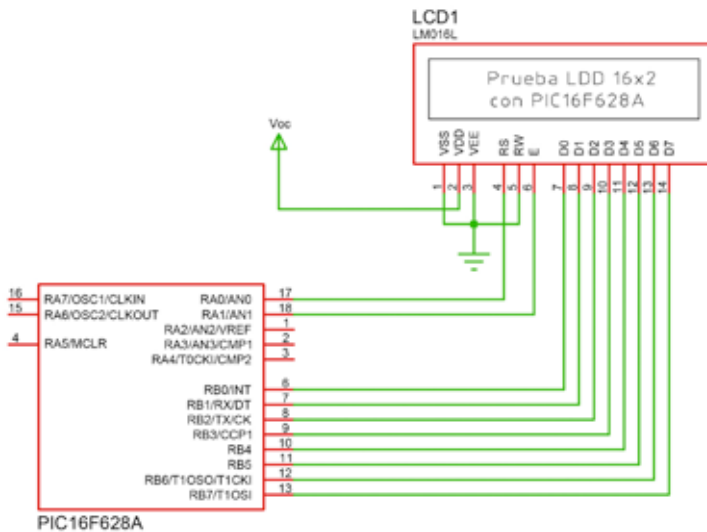
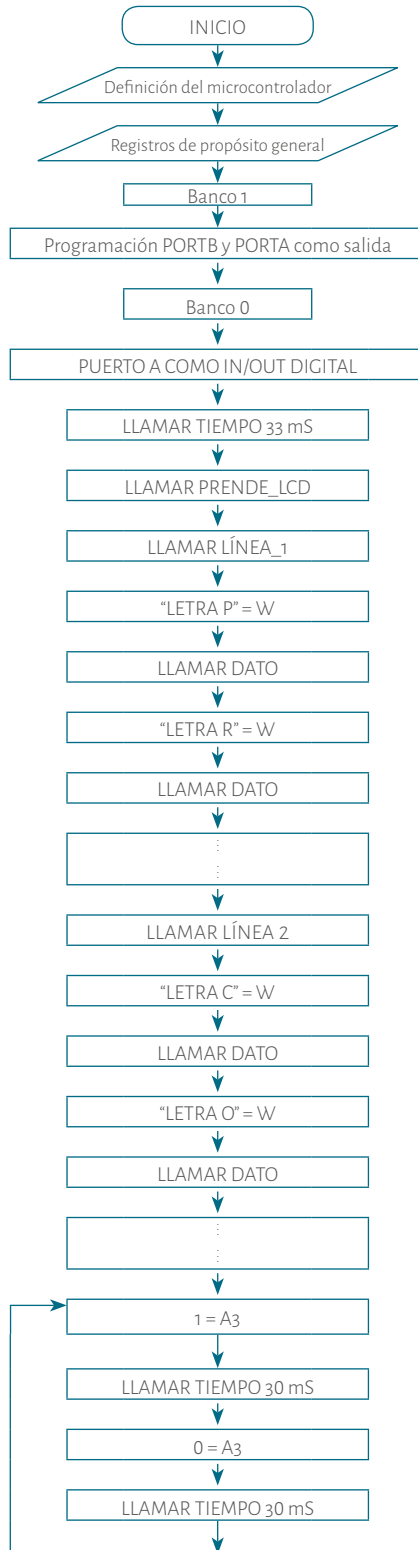


Figura 44. Diagrama de conexiones del ejemplo 17.

Fuente. Elaboración José Luis González. (La distribución de pines es tomada de Proteus).

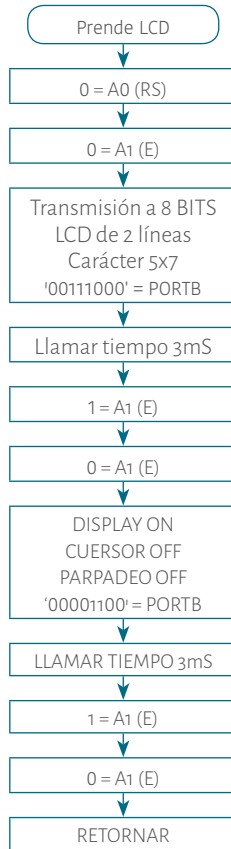
Paso 3. Para este ejemplo se van a crear varias subrutinas que van a ser citadas por el programa principal, este se muestra en el Diagrama 18, —el diagrama no está completo, el lector podrá darse cuenta de la secuencia para las letras tanto en la línea 1 como en la 2 que es cargar el código ASCII de la letra o número en W y luego Llamar la subrutina DATO—. En éste, lo primero que se hace es programar los puertos, luego dar un tiempo, posteriormente llama una subrutina de **PRENDER_LCD**, este se encarga de configurar y prender la LCD, diagrama 19, luego llama la subrutina **LÍNEA_1**, en la cual se selecciona la línea 1 de la LCD posición 00H, diagrama 20, en seguida se carga el dato en ASCII en el registro W, posteriormente se usa la subrutina **DATO** que se encarga de enviar el dato a la LCD y se repite el proceso para todas las letras. Cuando se requiere la línea 2 se usa la subrutina **LÍNEA_2**, la cual selecciona la línea 2 de la LCD; posición 40H, diagrama 21. Por último se garantiza que un led prenda y apague indefinidamente, con ello se prueba que la LCD un vez reciba el dato lo mantiene hasta que se le diga lo contrario.

Diagrama 19.
Programa manejo LCD a 8 bits. Ejemplo 17.



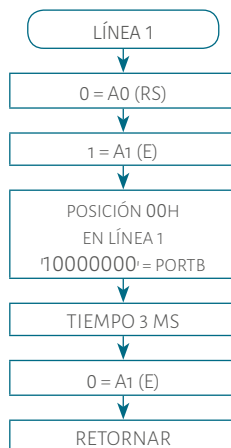
Fuente. Elaboración propia.

Diagrama 20.
Subrutina PRENDE_LCD. Ejemplo 17.



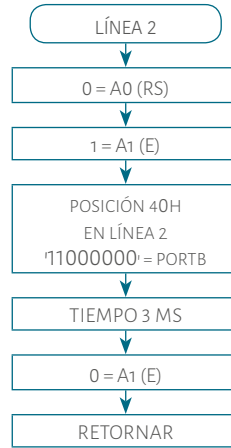
Fuente. Elaboración propia.

Diagrama 21.
Subrutina LÍNEA 1. Ejemplo 17.



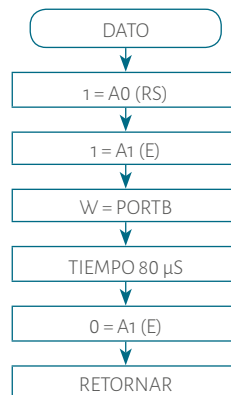
Fuente. Elaboración.

Diagrama 22.
Subrutina LÍNEA 2. Ejemplo 17.



Fuente. Elaboración propia.

Diagrama 23.
Subrutina DATO. Ejemplo 17.



Fuente. Elaboración propia.

Paso 4. El programa es el siguiente, (en este caso el programa está completo para todas las letras). Las subrutinas de tiempos son las mismas que se han usado en los programas anteriores pero con valores adecuados para cumplir con los requerimientos de tiempos de la LCD.

```
#INCLUDE<pt16f628A.inc>
```

```
CBLOCK
```

```
0x20
```

```
CON1
```

```
CON2
```

```
; Definición del microcontrolador es decir la librería.
```

```
; Definición de registros de propósito general.
```

CON3
CON4
ENDC

```

RES_VECT CODE 0X0000 ; Vector reset

;
PUERTOS
BSF STATUS,5 ; Se escoge el banco 1 para programar puertos.
BCF STATUS,6
CLRF TRISB ; PORT B como salida para el bus de datos de la LCD.
CLRF TRISA ; A0=RS, A1=E
BCF STATUS,5 ; Se escoge el banco 0 para sacar datos a la LCD.
BCF STATUS,6
MOVLW 0X07 ; Dato para deshabilitar los comparadores del PORT A.
; Se deja PORT A como entrada o salida digital.
MOVWF CMCON ; Registro que controla los comparadores.

;
CALL TIEMPO ; Se da un tiempo de 33 mS.
PROGRAMA
CALL PRENDER_LCD ; Ejecuta la subrutina de prender y configurar la LCD
CALL LÍNEA_1 ; LCD Línea 1 Posición 00

MOVLW 'P' ; Se carga el código ASCII de la letra P
CALL DATO ; Se visualiza en la LCD

MOVLW 'r' ; Se carga el código ASCII de la letra r
CALL DATO ; Se visualiza en la LCD

MOVLW 'u' ; Se carga el código ASCII de la letra u
CALL DATO ; Se visualiza en la LCD

MOVLW 'e' ; Se carga el código ASCII de la letra e
CALL DATO ; Se visualiza en la LCD

MOVLW 'b' ; Se carga el código ASCII de la letra b
CALL DATO ; Se visualiza en la LCD

MOVLW 'a' ; Se carga el código ASCII de la letra a
CALL DATO ; Se visualiza en la LCD

MOVLW ' ' ; Se carga el código ASCII del espacio
CALL DATO ; Se visualiza en la LCD

MOVLW 'L' ; Se carga el código ASCII de la letra L
CALL DATO ; Se visualiza en la LCD

```


MOVLW	'C'	; Se carga el código ASCII de la letra C
CALL	DATO	; Se visualiza en la LCD
MOVLW	'D'	; Se carga el código ASCII de la letra D
CALL	DATO	; Se visualiza en la LCD
MOVLW	' '	; Se carga el código ASCII del <i>espacio</i>
CALL	DATO	; Se visualiza en la LCD
MOVLW	'1'	; Se carga el código ASCII del Número 1
CALL	DATO	; Se visualiza en la LCD
MOVLW	'6'	; Se carga el código ASCII del Número 6
CALL	DATO	; Se visualiza en la LCD
MOVLW	'x'	; Se carga el código ASCII de la letra x
CALL	DATO	; Se visualiza en la LCD
MOVLW	'2'	; Se carga el código ASCII del Número 2
CALL	DATO	; Se visualiza en la LCD
CALL	LÍNEA_2	; LCD Línea 2 Posición 40
MOVLW	'c'	; Se carga el código ASCII de la letra c
CALL	DATO	; Se visualiza en la LCD
MOVLW	'o'	; Se carga el código ASCII de la letra o
CALL	DATO	; Se visualiza en la LCD
MOVLW	'n'	; Se carga el código ASCII de la letra n
CALL	DATO	; Se visualiza en la LCD
MOVLW	' '	; Se carga el código ASCII del <i>espacio</i>
CALL	DATO	; Se visualiza en la LCD
MOVLW	'P'	; Se carga el código ASCII de la letra P
CALL	DATO	; Se visualiza en la LCD
MOVLW	'1'	; Se carga el código ASCII del número 1
CALL	DATO	; Se visualiza en la LCD
MOVLW	'6'	; Se carga el código ASCII del número 6
CALL	DATO	; Se visualiza en la LCD
MOVLW	'F'	; Se carga el código ASCII de la letra F
CALL	DATO	; Se visualiza en la LCD

```

MOV LW <6> ; Se carga el código ASCII del número 6
CALL DATO ; Se visualiza en la LCD

MOV LW <2> ; Se carga el código ASCII del número 2
CALL DATO ; Se visualiza en la LCD

MOV LW <8> ; Se carga el código ASCII del número 8
CALL DATO ; Se visualiza en la LCD

MOV LW <A> ; Se carga el código ASCII de la letra A
CALL DATO ; Se visualiza en la LCD

;;
LED ; Rutina de prender el LED indefinidamente.
BSF PORTA,3 ; Prende el led en A3
CALL TIEMPO ; Se da tiempo para ver el LED prendido.
BCF PORTA,3 ; Apaga el led en A3
CALL TIEMPO ; Se da tiempo para ver el LED apagado.
GOTO LED ; Vuelve a prender el LED

;;
PRENDER_LCD
BCF PORTA,0 ; RS=0
BCF PORTA,1 ; E=0
MOV LW b'0011000' ; TRANSMISIÓN EN 8 BITS,
; LCD DE 2 LÍNEAS, CARACTER 5X7
MOVWF PORTB ; DL=1, N=1, F=0
CALL TIEMPO_3ms ; Tiempo superior a 1,64 mS
BSF PORTA,1 ; E=1
BCF PORTA,1 ; E=0

MOV LW b'00001100' ; DISPLAY ON, CURSOR OFF, PARPADEO OFF.
MOVWF PORTB ; D=1, C=0, B=0
CALL TIEMPO_3ms ; Tiempo superior a 1,64 mS
BSF PORTA,1 ; E=1
BCF PORTA,1 ; E=0

RETURN

;
LÍNEA_1
BCF PORTA,0 ; RS=0
BSF PORTA,1 ; E=1
MOV LW 0x80 ; POSICIÓN 00, INICIO LÍNEA 1.
MOVWF PORTB
CALL TIEMPO_3ms ; Tiempo superior a 1,64 mS
BCF PORTA,1 ; E=0
RETURN
;

```

```

LÍNEA_2
    BCF     PORTA,0           ; RS=0
    BSF     PORTA,1           ; E=1
    MOVLW  B'11000000'       ; POSICIÓN 40, INICIO LÍNEA 2.
    MOVWF  PORTB
    CALL   TIEMPO_3ms        ; Tiempo superior a 1,64 mS
    BCF     PORTA,1           ; E=0
    RETURN

;
DATO
    BSF     PORTA,0           ; RS=1
    BSF     PORTA,1           ; E=1
    MOVWF  PORTB             ; Se escribe el dato en el bus de datos de la LCD.
    CALL   TIEMPO_80mics     ; Tiempo superior a 40 µS
    BCF     PORTA,1           ; E=0
    RETURN

;;
TIEMPO_3ms                          ; Subrutina de 3 mS
    MOVLW  0X19              ; A 4 MHZ
    MOVWF  CON1
N1
    MOVLW  0X29
    MOVWF  CON2
N2
    DECFSZ CON2,1
    GOTO   N2
    DECFSZ CON1,1
    GOTO   N1
    RETURN

;
TIEMPO_80mics                        ; Subrutina de 80 µS
    MOVLW  0X01              ; A 4 MHZ
    MOVWF  CON1
N3
    MOVLW  0X18
    MOVWF  CON2
N4
    DECFSZ CON2,1
    GOTO   N4
    DECFSZ CON1,1
    GOTO   N3
    RETURN

;
TIEMPO                                ; Subrutina de 197 mS
    MOVLW  0X00              ; A 4 MHZ
    MOVWF  CON1
N5

```

```

MOV LW 0X00
MOV WF CON2

N6
    DECF SZ CON2,1
    GOTO N6
    DECF SZ CON1,1
    GOTO N5
    RETURN

;

END

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard, para ello hay que saber la distribución del microcontrolador (ver figura 30) y la distribución de pines de la LCD (ver tabla 19).

Paso 6. Si hay que hacer correcciones, hacerlas y volver al paso tres.

Ya que se ha visto teclado, LCD y se ha explicado el manejo de librerías, ahora se puede desarrollar un ejemplo que involucre las tres.

Ejemplo 18. Se desea usar una LCD de 16x2 y un teclado matricial 4x4: cada vez que se oprime una tecla aparece el símbolo respectivo en la LCD y se desplaza el texto hacia la izquierda, al estilo de una calculadora. Es necesario tener una tecla de cambio de símbolos para reconocer más símbolos y letras. Adicionalmente es recomendable tener un parlante que suene cada vez que se oprima una tecla. Finalmente, es fundamental no olvidar el led indicador de funcionamiento, pues este indica que el programa esta funcionando adecuadamente. Este indicador prende y apaga cada vez que el programa se ejecuta.

A continuación se desarrollará paso por paso:

Paso 1. En el ejemplo 15 se trabajó con teclado matricial 4x3. En el ejemplo 17 se explicó cómo funciona una LCD de 16 X 2. Allí se comentó que estas se pueden conectar a 8 bits o 4 bits. Para este ejemplo se puede trabajar a 4 bits, para ello se puede usar la librería que Palacios ha desarrollado para un LCD 16x2, usando sólo 6 líneas así: dos de control y cuatro de datos, la cual es una conexión de 4 bits; denominada LCD_4BIT.INC-B.INC; (recordar lo visto en el ejemplo 16 sobre el manejo de librerías). El lector podrá familiarizarse con el uso de la librería e incluso modificarla. Por lo pronto, se explicarán algunas de las subrutinas más importantes en la librería (ver la tabla 22).

Tabla 2.2.
Subrutinas importantes en la librería LCD4bits.

Subrutina	Comentario
LCD_Inicializa	Inicia la LCD para su correcto funcionamiento. Produce un reset por <i>software</i> , borra la memoria DDRAM y enciende la pantalla.
LCD_Caracter	Visualiza en la posición actual del cursor el código ASCII del dato contenido en el registro W.
LCD_Borra	Borra toda la pantalla y coloca el cursor al inicio de la línea 1.
LCD_Línea1	Envía el cursor al inicio de la línea 1.
LCD_Línea2	Envía el cursor al inicio de la línea 2.
LCD_Posición línea1	Envía el cursor a la posición de la línea 1 indicado por el registro W.
LCD_Posición línea2	Envía el cursor a la posición de la línea 2 indicado por el registro W.

Fuente. Elaboración propia, datos tomados de (Palacios, 2006, 194-195)

Para el manejo de esta librería **LCD_4BIT.INC-B.INC** es necesario usar subrutinas de tiempo, para ello, Palacios 2006 ha definido la librería **RETARDOS.INC**, la cual cuenta con varias subrutinas entre los 4 μ s hasta los 20 segundos.

Para el uso de las subrutinas de las dos librerías mencionadas anteriormente se usa la instrucción **CALL**.

Paso 2. Para la distribución del circuito hay que tener en cuenta que la librería maneja el bus de datos de la LCD por la parte alta del PORTB, desde B_4 hasta B_7 , y que las líneas de control **RS** y **E** están conectadas así; $B_0 = RS$ y $B_1 = E$. —El lector podrá darse cuenta que la librería fue modificada, líneas de control **RS** y **E** estaban conectadas así; $A_0 = RS$ y $A_2 = E$. — Por otra parte el teclado se puede conectar así: las filas desde A_0 hasta A_3 y A_4 , A_6 , A_7 , para las columnas; el parlante se puede conectar en B_3 , mediante un transistor como amplificador; el led indicador de funcionamiento del circuito se puede conectar en B_2 , en la Figura 43 se observa todo esto.

Paso 3. Para este ejemplo se usarán varias rutinas principales, ya que son las que controlan el teclado. Estas rutinas utilizan las siguientes subrutinas:

ROTAR: se encarga de rotar los datos que ingresan.

RECUPERAR: se encarga de mostrar los datos en la LCD mediante el uso de la librería.

LCD_4BIT.INC-B.INC, PITO: se encarga de hacer oscilar un pin del microcontrolador a una frecuencia audible y para amplificar corriente se usa un transistor en corte y saturación.

BORRAR: se encarga de borrar el dato. Este proceso es contrario al que se ejecuta con la subrutina **ROTAR**.

El diagrama de flujo de las rutinas principales es muy parecido a los usados en el ejemplo 15. Por tanto sólo es habilitar filas y monitorear columnas; la diferencia es que se tienen cuatro rutinas de teclado, así: primera (0 al 9 y Borra), segunda (A a J y Borra), tercera (K a S y Borra) y cuarta (T a Z y borra). Por ello no se harán diagramas de flujo para este ejemplo.

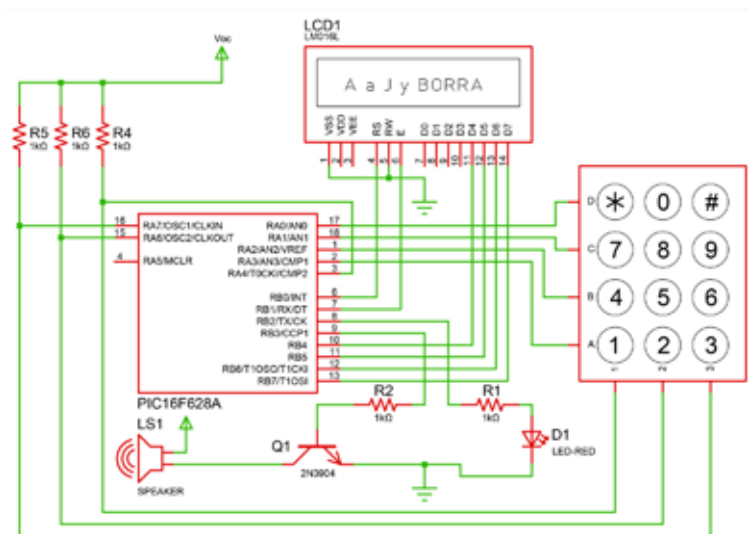


Figura 45. Diagrama de conexiones del ejemplo 18.

Fuente. Elaboración José Luis González. (La distribución de pines es tomada de Proteus).

Paso 4. El programa es el siguiente, en este caso el programa está completo para todas las letras. El lector podrá observar que algunos de los registros de propósito general están separados por comas para ahorrar espacio.

```
#INCLUDE<p16f628A.inc> ; Definición del microcontrolador es decir la librería.

CBLOCK ; Definición de registros de propósito general.
    CON1 ; REGISTROS PARA SUBROUTINAS DE RETARDO.
    CON2
    CON3
    D1, D2, D3, D4, D5, D6, D7, D8 ; REGISTRO PARA SUBROUTINA DE ROTACIÓN
```

```

D9, D10, D11, D12, D13, D14           ; Y VISUALIZACIÓN EN LA LCD.
D15, D16
ENDC
;

RES_VECT CODE 0X0000           ; Vector reset

INICIO

BSF STATUS,5                   ; Banco 1 para programar puertos.
BCF STATUS,6
BCF TRISB,2                     ; B2 SALIDA PARA EL LED
BCF TRISB,3                     ; B3 SALIDA PARA EL PARLANTE
MOVLW 0XF0                     ; PORTA PARA EL TECLADO,
MOVWF TRISA                    ; Filas (A0 hasta A3) y A4, A6, A7, para las columnas
BCF STATUS,5                   ; Banco 0 para sacar datos.
BCF STATUS,6
MOVLW 0X07                     ; Se deshabilitan los comparadores analógicos,
MOVWF CMCON                    ; el PORTA se toma como salida digital.

CALL LIMPIAR                   ; Subrutina de limpiar la LCD
CALL LCD_Inicializa            ; SE INICIALIZA LA LCD
;
T4_A_T1
CALL TONO_CAMBIO               ; Se genera un tono para indicar el cambio de teclado
;
TECLADO1
CALL LED                       ; LED INDICADOR

CALL TECLA1                    ; MENSAJE INFORMANDO QUÉ TECLAS ESTÁN
; HABILITADAS, 0 a 9 y * ES BORRAR
; SE HABILITA TECLAS (UNO, DOS, TRES)
BCF PORTA,3
BSF PORTA,2
BSF PORTA,1
BSF PORTA,0

BTFFS PORTA,4                  ; SE PREGUNTA SI OPRIMEN EL NÚMERO UNO
GOTO UNO
BTFFS PORTA,6                  ; SE PREGUNTA SI OPRIMEN EL NÚMERO DOS
GOTO DOS
BTFFS PORTA,7                  ; SE PREGUNTA SI OPRIMEN EL NÚMERO TRES
GOTO TRES
;
BSF PORTA,3                    ; SE HABILITA TECLAS (CUATRO, CINCO, SEIS)
BCF PORTA,2
BSF PORTA,1
BSF PORTA,0

```

```

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CUATRO
GOTO CUATRO
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CINCO
GOTO CINCO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO SEIS
GOTO SEIS
;
BSF PORTA,3 ; SE HABILITA TECLAS (SIETE, OCHO, NUEVE)
BSF PORTA,2
BCF PORTA,1
BSF PORTA,0
;
BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO SIETE
GOTO SIETE
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO OCHO
GOTO OCHO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO NUEVE
GOTO NUEVE
;
BSF PORTA,3 ; SE HABILITA TECLAS (CAMBIO DE TECLADO (T1 A T2),
BSF PORTA,2 ; CERO, BORRAR)
BSF PORTA,1
BCF PORTA,0
;
BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA TECLA BORRAR
CALL BORRAR
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CERO
GOTO CERO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN CAMBIO DE TECLADO
GOTO T1_A_T2
;
GOTO TECLADO1
;
T1_A_T2
CALL TONO_CAMBIO ; Se genera un tono para indicar el cambio de teclado
;
TECLADO2
CALL LED ; LED INDICADOR
CALL TECLA2 ; MENSAJE INFORMANDO QUÉ TECLAS ESTÁN
; HABILITADAS, A a J y * ES BORRAR
BCF PORTA,3 ; SE HABILITA TECLAS (LETRA_A, LETRA_B, LETRA_C)
BSF PORTA,2
BSF PORTA,1
BSF PORTA,0
;
BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA LETRA_A
GOTO LETRA_A
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN LA LETRA_B

```



```

GOTO LETRA_B
BTFS PORTA,7 ; SE PREGUNTA SI OPRIMEN LA LETRA_C
GOTO LETRA_C

;

BSF PORTA,3 ; SE HABILITA TECLAS (LETRA_D, LETRA_E, LETRA_F)
BCF PORTA,2
BSF PORTA,1
BSF PORTA,0

BTFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA LETRA_D
GOTO LETRA_D
BTFS PORTA,6 ; SE PREGUNTA SI OPRIMEN LA LETRA_E
GOTO LETRA_E
BTFS PORTA,7 ; SE PREGUNTA SI OPRIMEN LA LETRA_F
GOTO LETRA_F

;

BSF PORTA,3 ; SE HABILITA TECLAS (LETRA_G, LETRA_H, LETRA_I)
BSF PORTA,2
BCF PORTA,1
BSF PORTA,0

BTFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA LETRA_G
GOTO LETRA_G
BTFS PORTA,6 ; SE PREGUNTA SI OPRIMEN LA LETRA_H
GOTO LETRA_H
BTFS PORTA,7 ; SE PREGUNTA SI OPRIMEN LA LETRA_I
GOTO LETRA_I

;

BSF PORTA,3 ; SE HABILITA TECLAS (LETRA_J, (CAMBIO DE
BSF PORTA,2 ; TECLADO (T2 A T1), BORRAR)
BSF PORTA,1
BCF PORTA,0

BTFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA TECLA BORRAR
CALL BORRAR
BTFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CERO
GOTO LETRA_J
BTFS PORTA,7 ; SE PREGUNTA SI OPRIMEN CAMBIO DE TECLADO
GOTO T2_A_T3

GOTO TECLADO2

;
T2_A_T3
CALL TONO_CAMBIO ; Se genera un tono para indicar el cambio de teclado

TECLADO3
CALL LED ; LED INDICADOR
CALL TECLA3 ; MENSAJE INFORMANDO QUÉ TECLAS ESTÁN
; HABILITADAS, K a Sy* ES BORRAR
BCF PORTA,3 ; SE HABILITA TECLAS (LETRA_K, LETRA_L, LETRA_M)

```

```

BSF      PORTA,2
BSF      PORTA,1
BSF      PORTA,0

BTFFS   PORTA,4      ; SE PREGUNTA SI OPRIMEN LA LETRA_K
GOTO    LETRA_K
BTFFS   PORTA,6      ; SE PREGUNTA SI OPRIMEN LA LETRA_L
GOTO    LETRA_L
BTFFS   PORTA,7      ; SE PREGUNTA SI OPRIMEN LA LETRA_M
GOTO    LETRA_M

;

BSF      PORTA,3      ; SE HABILITA TECLAS (LETRA_N, LETRA_Ñ, LETRA_O)
BCF      PORTA,2
BSF      PORTA,1
BSF      PORTA,0

BTFFS   PORTA,4      ; SE PREGUNTA SI OPRIMEN LA LETRA_N
GOTO    LETRA_N
BTFFS   PORTA,6      ; SE PREGUNTA SI OPRIMEN LA LETRA_Ñ
GOTO    LETRA_Ñ
BTFFS   PORTA,7      ; SE PREGUNTA SI OPRIMEN LA LETRA_O
GOTO    LETRA_O

;

BSF      PORTA,3      ; SE HABILITA TECLAS (LETRA_P, LETRA_Q, LETRA_R)
BSF      PORTA,2
BCF      PORTA,1
BSF      PORTA,0

BTFFS   PORTA,4      ; SE PREGUNTA SI OPRIMEN LA LETRA_P
GOTO    LETRA_P
BTFFS   PORTA,6      ; SE PREGUNTA SI OPRIMEN LA LETRA_Q
GOTO    LETRA_Q
BTFFS   PORTA,7      ; SE PREGUNTA SI OPRIMEN LA LETRA_R
GOTO    LETRA_R

;

BSF      PORTA,3      ; SE HABILITA TECLAS (LETRA_S, (CAMBIO DE
BSF      PORTA,2      ; TECLADO (T2 A T3), BORRAR)
BSF      PORTA,1
BCF      PORTA,0

BTFFS   PORTA,4      ; SE PREGUNTA SI OPRIMEN LA TECLA BORRAR
CALL    BORRAR
BTFFS   PORTA,6      ; SE PREGUNTA SI OPRIMEN EL NÚMERO CERO
GOTO    LETRA_S
BTFFS   PORTA,7      ; SE PREGUNTA SI OPRIMEN CAMBIO DE TECLADO
GOTO    T3_A_T4

GOTO    TECLADO3

```

```

T3_A_T4
CALL    TONO_CAMBIO    ; Se genera un tono para indicar el cambio de teclado
TECLADO4
CALL    LED            ; LED INDICADOR
CALL    TECLA4        ; MENSAJE INFORMANDO QUÉ TECLAS ESTÁN
                    ; HABILITADAS, T a Z y * ES BORRAR

    BCF    PORTA,3    ; SE HABILITA TECLAS (LETRA_T, LETRA_U, LETRA_V)
    BSF    PORTA,2
    BSF    PORTA,1
    BSF    PORTA,0

    BTFSS   PORTA,4    ; SE PREGUNTA SI OPRIMEN LA LETRA_T
    GOTO   LETRA_T
    BTFSS   PORTA,6    ; SE PREGUNTA SI OPRIMEN LA LETRA_U
    GOTO   LETRA_U
    BTFSS   PORTA,7    ; SE PREGUNTA SI OPRIMEN LA LETRA_V
    GOTO   LETRA_V
;

    BSF    PORTA,3    ; SE HABILITA TECLAS (LETRA_W, LETRA_X, LETRA_Y)
    BCF    PORTA,2
    BSF    PORTA,1
    BSF    PORTA,0

    BTFSS   PORTA,4    ; SE PREGUNTA SI OPRIMEN LA LETRA_W
    GOTO   LETRA_W
    BTFSS   PORTA,6    ; SE PREGUNTA SI OPRIMEN LA LETRA_X
    GOTO   LETRA_X
    BTFSS   PORTA,7    ; SE PREGUNTA SI OPRIMEN LA LETRA_Y
    GOTO   LETRA_Y
;

    BSF    PORTA,3    ; SE HABILITA TECLAS (LETRA_Z, ESPACIO, LETRA_?)
    BSF    PORTA,2
    BCF    PORTA,1
    BSF    PORTA,0

    BTFSS   PORTA,4    ; SE PREGUNTA SI OPRIMEN LA LETRA_Z
    GOTO   LETRA_Z
    BTFSS   PORTA,6    ; SE PREGUNTA SI OPRIMEN ESPACIO
    GOTO   LETRA_ESP
    BTFSS   PORTA,7    ; SE PREGUNTA SI OPRIMEN LA LETRA_?
    GOTO   LETRA_?
;

    BSF    PORTA,3    ; SE HABILITA TECLAS (LETRA_¿, (CAMBIO DE
    BSF    PORTA,2    ; TECLADO (T4 A T1), BORRAR)
    BSF    PORTA,1
    BCF    PORTA,0

```

```

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA TECLA BORRAR
CALL BORRAR
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN TECLA ¿
GOTO LETRA_¿
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN CAMBIO DE TECLADO
GOTO T4_A_T1

GOTO TECLADO4

;
TECLA1

CALL LCD_Línea 2 ; MENSAJE DE TECLADO 1
MOVLW «0» ; Se carga el código ASCII del número 0
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «<<» ; Se carga el código ASCII del espacio
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «a» ; Se carga el código ASCII de la letra a
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «<<» ; Se carga el código ASCII del espacio
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «9» ; Se carga el código ASCII del número 9
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «<<» ; Se carga el código ASCII del espacio
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «y» ; Se carga el código ASCII de la letra y
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «<<» ; Se carga el código ASCII del espacio
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «B» ; Se carga el código ASCII de la letra B
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «O» ; Se carga el código ASCII de la letra O
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «R» ; Se carga el código ASCII de la letra R
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «R» ; Se carga el código ASCII de la letra R
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «A» ; Se carga el código ASCII de la letra A
CALL LCD_Caracter ; Se visualiza en la LCD
RETURN

;
TECLA2

CALL LCD_Línea 2 ; MENSAJE DE TECLADO 2
MOVLW «A» ; Se carga el código ASCII de la letra A
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «<<» ; Se carga el código ASCII del espacio
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW «a» ; Se carga el código ASCII de la letra a
CALL LCD_Caracter ; Se visualiza en la LCD

```

```

MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <J> ; Se carga el código ASCII de la letra J
CALL LCD_Character ; Se visualiza en la LCD
MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <y> ; Se carga el código ASCII de la letra y
CALL LCD_Character ; Se visualiza en la LCD
MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <B> ; Se carga el código ASCII de la letra B
CALL LCD_Character ; Se visualiza en la LCD
MOVW <O> ; Se carga el código ASCII de la letra O
CALL LCD_Character ; Se visualiza en la LCD
MOVW <R> ; Se carga el código ASCII de la letra R
CALL LCD_Character ; Se visualiza en la LCD
MOVW <R> ; Se carga el código ASCII de la letra R
CALL LCD_Character ; Se visualiza en la LCD
MOVW <A> ; Se carga el código ASCII de la letra A
CALL LCD_Character ; Se visualiza en la LCD
RETURN

```

```

;
TECLA3

```

```

CALL LCD_Línea 2 ; MENSAJE DE TECLADO 3
MOVW <K> ; Se carga el código ASCII de la letra K
CALL LCD_Character ; Se visualiza en la LCD
MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <a> ; Se carga el código ASCII de la letra a
CALL LCD_Character ; Se visualiza en la LCD
MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <S> ; Se carga el código ASCII de la letra S
CALL LCD_Character ; Se visualiza en la LCD
MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <y> ; Se carga el código ASCII de la letra y
CALL LCD_Character ; Se visualiza en la LCD
MOVW << ; Se carga el código ASCII del espacio
CALL LCD_Character ; Se visualiza en la LCD
MOVW <B> ; Se carga el código ASCII de la letra B
CALL LCD_Character ; Se visualiza en la LCD
MOVW <O> ; Se carga el código ASCII de la letra O
CALL LCD_Character ; Se visualiza en la LCD
MOVW <R> ; Se carga el código ASCII de la letra R
CALL LCD_Character ; Se visualiza en la LCD
MOVW <R> ; Se carga el código ASCII de la letra R

```

```

CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <A>              ; Se carga el código ASCII de la letra A
CALL    LCD_Caracter    ; Se visualiza en la LCD
RETURN

;
TECLA4

CALL    LCD_Línea 2     ; MENSAJE DE TECLADO 4
MOVLW  <T>              ; Se carga el código ASCII de la letra T
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <<               ; Se carga el código ASCII del espacio
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <a>              ; Se carga el código ASCII de la letra a
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <<               ; Se carga el código ASCII del espacio
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <Z>              ; Se carga el código ASCII de la letra Z
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <<               ; Se carga el código ASCII del espacio
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <y>              ; Se carga el código ASCII de la letra y
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <<               ; Se carga el código ASCII del espacio
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <B>              ; Se carga el código ASCII de la letra B
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <O>              ; Se carga el código ASCII de la letra O
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <R>              ; Se carga el código ASCII de la letra R
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <R>              ; Se carga el código ASCII de la letra R
CALL    LCD_Caracter    ; Se visualiza en la LCD
MOVLW  <A>              ; Se carga el código ASCII de la letra A
CALL    LCD_Caracter    ; Se visualiza en la LCD
RETURN

;
TONO_CAMBIO

CALL    PITO            ; SUBROUTINA DE CAMBIO DE TECLADO, ES
CALL    Retardo_50ms    ; DECIR CAMBIO DE SÍMBOLOS
CALL    PITO            ; SE USA UNA SERIE DE RETARDOS PARA HACER
CALL    Retardo_50ms    ; EFECTO DE PITO LARGO,
CALL    PITO
CALL    Retardo_50ms
RETURN

;
LED

BSF    PORTB,2          ; Prende el LED indicador
CALL    Retardo_50ms    ; Durante este tiempo.

```

```

BCF      PORTB,2      ; Apaga el LED indicador
CALL    Retardo_50ms ; Durante este tiempo.
RETURN

;
CERO

CALL    PITO          ; Subrutina del número CERO.
CALL    ROTA          ; Se genera tono para indicar que se oprimió la tecla.
MOVLW  <0>           ; Se ROTA los datos para efecto de desplazamiento.
MOVWF  D16            ; Se carga el código ASCII del número 0
CALL    RECUPERAR     ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO   TECLADO1       ; Va al programa de TECLADO1

;
UNO

CALL    PITO          ; Subrutina del número UNO.
CALL    ROTA          ; Se genera tono para indicar que se oprimió la tecla.
MOVLW  <1>           ; Se ROTA los datos para efecto de desplazamiento.
MOVWF  D16            ; Se carga el código ASCII del Número 1
CALL    RECUPERAR     ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO   TECLADO1       ; Va al programa de TECLADO1

;
DOS

CALL    PITO          ; Subrutina del número DOS.
CALL    ROTA          ; Se genera tono para indicar que se oprimió la tecla.
MOVLW  <2>           ; Se ROTA los datos para efecto de desplazamiento.
MOVWF  D16            ; Se carga el código ASCII del Número 2
CALL    RECUPERAR     ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO   TECLADO1       ; Va al programa de TECLADO1

;
TRES

CALL    PITO          ; Subrutina del Número TRES.
CALL    ROTA          ; Se genera tono para indicar que se oprimió la tecla.
MOVLW  <3>           ; Se ROTA los datos para efecto de desplazamiento.
MOVWF  D16            ; Se carga el código ASCII del Número 3
CALL    RECUPERAR     ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO   TECLADO1       ; Va al programa de TECLADO1

;
CUATRO

CALL    PITO          ; Subrutina del número CUATRO.
CALL    ROTA          ; Se genera tono para indicar que se oprimió la tecla.
MOVLW  <4>           ; Se ROTA los datos para efecto de desplazamiento.
MOVWF  D16            ; Se carga el código ASCII del número 4
CALL    RECUPERAR     ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote

```

```

;
CINCO
GOTO    TECLADO1    ; Va al programa de TECLADO1

CALL    PITO        ; Subrutina del número CINCO.
CALL    ROTA        ; Se genera tono para indicar que se oprimió la tecla.
MOVWLW <5>         ; Se ROTA los datos para efecto de desplazamiento.
MOVWF   D16         ; Se carga el código ASCII del número 5

CALL    RECUPERAR   ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO    TECLADO1    ; Va al programa de TECLADO1

;
SEIS
CALL    PITO        ; Subrutina del número SEIS.
CALL    ROTA        ; Se genera tono para indicar que se oprimió la tecla.
MOVWLW <6>         ; Se ROTA los datos para efecto de desplazamiento.
MOVWF   D16         ; Se carga el código ASCII del número 6

CALL    RECUPERAR   ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO    TECLADO1    ; Va al programa de TECLADO1

;
SIETE
CALL    PITO        ; Subrutina del número SIETE.
CALL    ROTA        ; Se genera tono para indicar que se oprimió la tecla.
MOVWLW <7>         ; Se ROTA los datos para efecto de desplazamiento.
MOVWF   D16         ; Se carga el código ASCII del número 7

CALL    RECUPERAR   ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO    TECLADO1    ; Va al programa de TECLADO1

;
OCHO
CALL    PITO        ; Subrutina del número OCHO.
CALL    ROTA        ; Se genera tono para indicar que se oprimió la tecla.
MOVWLW <8>         ; Se ROTA los datos para efecto de desplazamiento.
MOVWF   D16         ; Se carga el código ASCII del número 8

CALL    RECUPERAR   ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO    TECLADO1    ; Va al programa de TECLADO1

;
NUEVE
CALL    PITO        ; Subrutina del número NUEVE.
CALL    ROTA        ; Se genera tono para indicar que se oprimió la tecla.
MOVWLW <9>         ; Se ROTA los datos para efecto de desplazamiento.
MOVWF   D16         ; Se carga el código ASCII del número 9

CALL    RECUPERAR   ; Se visualiza en la LCD
CALL    Retardo_200ms ; Retardo para evitar antirebote
GOTO    TECLADO1    ; Va al programa de TECLADO1
;

```



```

LETRA_A                                     ; Subrutina de la letra A.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <A>                                  ; Se carga el código ASCII de la letra A
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO2                             ; Va al programa de TECLADO2

;
LETRA_B                                     ; Subrutina de la letra B.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <B>                                  ; Se carga el código ASCII de la letra B
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO2                             ; Va al programa de TECLADO2

;
LETRA_C                                     ; Subrutina de la letra C.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <C>                                  ; Se carga el código ASCII de la letra C
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO2                             ; Va al programa de TECLADO2

;
LETRA_D                                     ; Subrutina de la letra D.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <D>                                  ; Se carga el código ASCII de la letra D
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO2                             ; Va al programa de TECLADO2

;
LETRA_E                                     ; Subrutina de la letra E.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <E>                                  ; Se carga el código ASCII de la letra E
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO2                             ; Va al programa de TECLADO2

;
LETRA_F                                     ; Subrutina de la letra F.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.

```

```

MOV LW    <F>                ; Se carga el código ASCII de la letra F
MOV WF   D16
CALL     RECUPERAR          ; Se visualiza en la LCD
CALL     Retardo_200ms     ; Retardo para evitar antirebote
GOTO    TECLADO2          ; Va al programa de TECLADO2

;
LETRA_G                ; Subrutina de la letra G.
CALL     PITO              ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA              ; Se ROTA los datos para efecto de desplazamiento.
MOV LW   <G>              ; Se carga el código ASCII de la letra G
MOV WF   D16
CALL     RECUPERAR          ; Se visualiza en la LCD
CALL     Retardo_200ms     ; Retardo para evitar antirebote
GOTO    TECLADO2          ; Va al programa de TECLADO2

;
LETRA_H                ; Subrutina de la letra H.
CALL     PITO              ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA              ; Se ROTA los datos para efecto de desplazamiento.
MOV LW   <H>              ; Se carga el código ASCII de la letra H
MOV WF   D16
CALL     RECUPERAR          ; Se visualiza en la LCD
CALL     Retardo_200ms     ; Retardo para evitar antirebote
GOTO    TECLADO2          ; Va al programa de TECLADO2

;
LETRA_I                ; Subrutina de la letra I.
CALL     PITO              ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA              ; Se ROTA los datos para efecto de desplazamiento.
MOV LW   <I>              ; Se carga el código ASCII de la letra I
MOV WF   D16
CALL     RECUPERAR          ; Se visualiza en la LCD
CALL     Retardo_200ms     ; Retardo para evitar antirebote
GOTO    TECLADO2          ; Va al programa de TECLADO2

;
LETRA_J                ; Subrutina de la letra J.
CALL     PITO              ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA              ; Se ROTA los datos para efecto de desplazamiento.
MOV LW   <J>              ; Se carga el código ASCII de la letra J
MOV WF   D16
CALL     RECUPERAR          ; Se visualiza en la LCD
CALL     Retardo_200ms     ; Retardo para evitar antirebote
GOTO    TECLADO2          ; Va al programa de TECLADO2

;
LETRA_K                ; Subrutina de la letra K.
CALL     PITO              ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA              ; Se ROTA los datos para efecto de desplazamiento.
MOV LW   <K>              ; Se carga el código ASCII de la letra K
MOV WF   D16
CALL     RECUPERAR          ; Se visualiza en la LCD

```

```

CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO3 ; Va al programa de TECLADO3

;
LETRA_L ; Subrutina de la letra L.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <L> ; Se carga el código ASCII de la letra L
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO3 ; Va al programa de TECLADO3

;
LETRA_M ; Subrutina de la letra M.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <M> ; Se carga el código ASCII de la letra M
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO3 ; Va al programa de TECLADO3

;
LETRA_N ; Subrutina de la letra N.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <N> ; Se carga el código ASCII de la letra N
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO3 ; Va al programa de TECLADO3

;
LETRA_Ñ ; Subrutina de la letra Ñ.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <Ñ> ; Se carga el código ASCII de la letra Ñ
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO3 ; Va al programa de TECLADO3

;
LETRA_O ; Subrutina de la letra O.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <O> ; Se carga el código ASCII de la letra O
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO3 ; Va al programa de TECLADO3

;

```

```

LETRA_P                                     ; Subrutina de la letra P.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <P>                                 ; Se carga el código ASCII de la letra P
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO3                             ; Va al programa de TECLADO3
;
LETRA_Q                                     ; Subrutina de la letra Q.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <Q>                                 ; Se carga el código ASCII de la letra Q
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO3                             ; Va al programa de TECLADO3
;
LETRA_R                                     ; Subrutina de la letra R.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <R>                                 ; Se carga el código ASCII de la letra R
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO3                             ; Va al programa de TECLADO3
;
LETRA_S                                     ; Subrutina de la letra S.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <S>                                 ; Se carga el código ASCII de la letra S
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO3                             ; Va al programa de TECLADO3
;
LETRA_T                                     ; Subrutina de la letra T.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <T>                                 ; Se carga el código ASCII de la letra T
MOVWF    D16
CALL      RECUPERAR                           ; Se visualiza en la LCD
CALL      Retardo_200ms                       ; Retardo para evitar antirebote
GOTO     TECLADO4                             ; Va al programa de TECLADO4
;
LETRA_U                                     ; Subrutina de la letra U.
CALL      PITO                                ; Se genera tono para indicar que se oprimió la tecla.
CALL      ROTA                                ; Se ROTA los datos para efecto de desplazamiento.

```

```

MOV LW <U> ; Se carga el código ASCII de la letra U
MOV WF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4

;
LETRA_V ; Subrutina de la letra V.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOV LW <V> ; Se carga el código ASCII de la letra V
MOV WF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4

;
LETRA_W ; Subrutina de la letra W.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOV LW <W> ; Se carga el código ASCII de la letra W
MOV WF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4

;
LETRA_X ; Subrutina de la letra X.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOV LW <X> ; Se carga el código ASCII de la letra X
MOV WF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4

;
LETRA_Y ; Subrutina de la letra Y.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOV LW <Y> ; Se carga el código ASCII de la letra Y
MOV WF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4

;
LETRA_Z ; Subrutina de la letra Z.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOV LW <Z> ; Se carga el código ASCII de la letra Z
MOV WF D16
CALL RECUPERAR ; Se visualiza en la LCD

```

```

CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4
;
LETRA_ESP ; Subrutina del ESPACIO.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW << ; Se carga el código ASCII del ESPACIO
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4
;
LETRA_? ; Subrutina del símbolo ?
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <? ; Se carga el código ASCII del símbolo ?
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4
;
LETRA_¿ ; Subrutina del símbolo ¿
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <¿ ; Se carga el código ASCII del símbolo ¿
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
GOTO TECLADO4 ; Va al programa de TECLADO4
;
LIMPIAR ; SUBRUTINA DE LIMPIAR LA LCD
MOVLW << ; Se carga el código ASCII del ESPACIO
MOVWF D1 ; Solo es garantizar el espacio en todos los registros
MOVWF D2
MOVWF D3
MOVWF D4
MOVWF D5
MOVWF D6
MOVWF D7
MOVWF D8
MOVWF D9
MOVWF D10
MOVWF D11
MOVWF D12
MOVWF D13
MOVWF D14
MOVWF D15

```

```

MOVWF D16
RETURN
;
PITO
MOVWF CON3
MOVWF CON3
; SUBROUTINA DE GENERAR EL TONO
; Número de veces para repetir el ON OFF en B3
; Es decir la duración del pito

BSF PORTB,3
CALL TMPITO
BCF PORTB,3
CALL TMPITO
DECFSZ CON3,1
GOTO PITO1
RETURN
;
ROTA
; SUBROUTINA DE DESPLAZAMIENTO A LA IZQUIERDA EN LA LCD
; Es decir la que permite el efecto de corrimiento de los caracteres

MOVF D2,W
MOVWF D1
MOVF D3,W
MOVWF D2
MOVF D4,W
MOVWF D3
MOVF D5,W
MOVWF D4
MOVF D6,W
MOVWF D5
MOVF D7,W
MOVWF D6
MOVF D8,W
MOVWF D7
MOVF D9,W
MOVWF D8
MOVF D10,W
MOVWF D9
MOVF D11,W
MOVWF D10
MOVF D12,W
MOVWF D11
MOVF D13,W
MOVWF D12
MOVF D14,W
MOVWF D13
MOVF D15,W
MOVWF D14
MOVF D16,W
MOVWF D15
RETURN
;

```

BORRAR

; SUBROUTINA DE DESPLAZAMIENTO A LA DERECHA EN LA LCD

; Es decir la que permite el efecto BORRAR

```

CALL    PITO                ; Se genera tono para indicar que se oprimió la tecla.
MOVWF  D15,W               ; D15 = D16
MOVWF  D16
MOVWF  D14,W               ; D14 = D15
MOVWF  D15
MOVWF  D13,W               ; D13 = D14
MOVWF  D14
MOVWF  D12,W               ; D12 = D13
MOVWF  D13
MOVWF  D11,W               ; D11 = D12
MOVWF  D12
MOVWF  D10,W               ; D10 = D11
MOVWF  D11
MOVWF  D9,W                ; D9 = D10
MOVWF  D10
MOVWF  D8,W                ; D8 = D9
MOVWF  D9
MOVWF  D7,W                ; D7 = D8
MOVWF  D8
MOVWF  D6,W                ; D6 = D7
MOVWF  D7
MOVWF  D5,W                ; D5 = D6
MOVWF  D6
MOVWF  D4,W                ; D4 = D5
MOVWF  D5
MOVWF  D3,W                ; D3 = D4
MOVWF  D4
MOVWF  D2,W                ; D2 = D3
MOVWF  D3
MOVWF  D1,W                ; D1 = D2
MOVWF  D2
CALL    RECUPERAR          ; Se visualiza en la LCD
CALL    Retardo_200ms      ; Retardo para evitar antirebote
RETURN

```

;

RECUPERAR

```

CALL    LCD_Borra          ; Borra la pantalla
CALL    LCD_Línea 1        ; Se escribe en línea 1 de la LCD
MOVWF  D1,W                ; Se recupera D1
CALL    LCD_Caracter       ; Se visualiza en la LCD
MOVWF  D2,W                ; Se recupera D2
CALL    LCD_Caracter       ; Se visualiza en la LCD
MOVWF  D3,W                ; Se recupera D3
CALL    LCD_Caracter       ; Se visualiza en la LCD
MOVWF  D4,W                ; Se recupera D4

```



```

CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D5,W             ; Se recupera D5
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D6,W             ; Se recupera D6
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D7,W             ; Se recupera D7
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D8,W             ; Se recupera D8
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D9,W             ; Se recupera D9
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D10,W            ; Se recupera D10
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D11,W            ; Se recupera D11
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D12,W            ; Se recupera D12
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D13,W            ; Se recupera D13
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D14,W            ; Se recupera D14
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D15,W            ; Se recupera D15
CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D16,W            ; Se recupera D16
CALL    LCD_Character    ; Se visualiza en la LCD
RETURN

;
TMPITO

MOV LW  0x10             ; SUBROUTINA TIEMPO PARA PITO
MOVWF  CON1

DEC6

MOV LW  0x10
MOVWF  CON2

DEC7

DECFSZ CON2,1
GOTO   DEC7
DECFSZ CON1,1
GOTO   DEC6
RETURN

;

#include <LCD_4BIT-B.INC> ; Subrutinas de control del módulo LCD.
#include <RETARDOS.INC>  ; Subrutinas de retardo.

END

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se este cableando en el protoboard, para ello hay que saber la distribución del microcontrolador (ver figura 30), y la distribución de pines de la LCD (ver tabla 19), y del teclado.

Paso 6. Si es necesario hacer las correcciones respectivas y volver al paso tres.

4.4. Manejo de la Memoria EEPROM Data

Los microcontroladores de microchip tienen una memoria llamada EEPROM Data que, para el caso del PIC16F628, es de 128x8. En consecuencia, tiene 128 registros, cada uno de un tamaño de 8 bits (ver tabla 10). Para mayor información el lector podrá consultar el *Data Sheet* del microcontrolador.

Una vez aclarado el manejo del teclado y de la LCD, se puede hacer un ejemplo usando la Memoria EEPROM Data.

Ejemplo 19. Se quiere acceder a la memoria EEPROM usando el montaje del ejemplo 18. El teclado sirve para digitar una clave. Si la clave es correcta aparecerá la frase "**Clave correcta**" y si es incorrecta aparecerá la frase: "**Clave incorrecta**". En ambos casos se producirá un sonido diferente mediante el parlante. Para ingresar la clave se usará la tecla (#) "número" y para el cambio de la clave se usará la tecla (*) "asterisco". Hay que tener en cuenta que para cambiar la clave se requiere la clave anterior. Además, la clave debe ser mínimo de 4 dígitos. Si la clave se digita 3 veces erróneamente deshabilita el teclado por un tiempo aproximado de 2 minutos.

A continuación, se desarrollará paso por paso:

Paso 1. En el ejemplo 18 se trabaja con teclado matricial 4x3, LCD y parlante. A continuación explicaremos entonces cómo funciona la memoria EEPROM Data. Para acceder a esta memoria se utilizan 4 registros: 2 para el control de lectura/escritura y 2 para el direccionamiento indirecto. A continuación, se explica cada uno de ellos y la forma de cómo usarlos en el proceso de escritura y de lectura.

Los registros de direccionamiento indirecto son **EEADR** y **EEDATA**. El primero hace las veces de bus de direcciones y el segundo las veces de bus de datos (Microchip, 2007). Para entender mejor esto, hay que suponer que se desea leer la posición $2C_H$. En dicha posición está el dato $7F_H$ por lo que es preciso cargar la posición ($2C_H$)

al registro **EEDR** a través del registro W. Con ayuda del registro **EEDATA** es posible leer el dato ($7F_{H}$) a través del registro W. Estos mismos pasos se aplican para el proceso de escritura.

Los registros de control son: *EECON1* y *EECON2*. El primero permite habilitar o deshabilitar la memoria EEPROM Data y, a su vez, controla el proceso de lectura o de escritura (Microchip, 2007, p. 90). Este registro solo tiene implementado el nibble de menor peso, el nibble de mayor peso no está implementado; así como se ilustra a continuación:

Registro **EECON1**

				WRERR	WREN	WR	RD
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Fuente. Elaboración propia, datos tomados de (Microchip, 2007, p. 90).

Bit 3: WRERR: EEPROM Error Flag bit. Es el bit de error.

- 1 → Una operación de escritura es terminada prematuramente (cuando se usa MCLR o cuando se produce un reset normal en el WDT).
- 0 → El proceso de escritura es completa.

Bit 2: WREN: EEPROM Write Enable bit. Es el bit que habilita la escritura en la EEPROM.

- 1 → Se habilita los ciclos de escritura en la memoria EEPROM data.
- 0 → Se deshabilita los ciclos de escritura en la memoria EEPROM data.

Bit 1: WR: Write Control. Es el bit de control de escritura.

- 1 → Se inicia el ciclo de escritura en la memoria EEPROM data. Cuando se termina la escritura del dato, por *hardware*, el bit se pone en cero (0).
- 0 → El ciclo de lectura de la memoria EEPROM data se ha completado.

Bit 0: RD: Read Control. Es el bit de control de lectura.

- 1 → Se inicia el proceso de lectura en la memoria EEPROM data. Cuando se termina un ciclo de lectura de un dato, por *hardware*, se pone en cero (0).
- 0 → No se inicia el proceso de lectura de la memoria EEPROM data.

El registro *EECON2* se utiliza sólo en el proceso de escritura como un “contador”. Por esta razón, el fabricante recomienda cargarle dos valores en hexadecimal durante este proceso (Microchip, 2007). Este registro no está implementado físicamente.

Paso 2. Para este ejercicio se usa la misma distribución del ejemplo anterior (ver la figura 45).

Paso 3. Para este ejemplo se usa un programa principal que es igual a la rutina **TECLADO1** del programa anterior. Por lo tanto, sólo se harán los diagramas de flujo de las subrutinas **CLAVE** y **CAMBIO**. La primera es la que valida la clave, pues se encarga de leer la memoria EEPROM data y comparar los datos ingresados por el teclado. Si son iguales, el microcontrolador dará la orden de desplegar en la línea dos de la LCD “**Clave correcta**” y si son diferentes desplegará “**Clave errónea**”. La segunda subrutina se encarga de cambiar la clave. Para ello, primero hay que validar la clave anterior y compararla con los datos que tiene la EEPROM data. Si es igual el sistema pide que se digite la nueva clave desplegando “**Nueva clave**” y si es errónea desplegará “**Clave errónea**”. Después de digitar la clave errónea en tres oportunidades el microcontrolador irá a una subrutina donde deshabilitará el teclado por un tiempo aproximado de dos minutos.

Para hacer el diagrama de flujo hay que identificar qué posiciones, de las 128 de la EEPROM data van a ser usadas para guardar la clave. Por ejemplo, se pueden usar desde 40_H hasta 43_H . Hay que recordar que la clave debe tener mínimo cuatro dígitos. A continuación se explica cada una de las subrutinas.

En la subrutina **CLAVE** se debe leer primero la posición 40_H de la memoria EEPROM data y debe escribirse en el registro **DATO1** (registro de propósito general). Después se lee la posición 41_H y se pasa al registro **DATO2**. Posteriormente, se lee la posición 42_H y se pasa al registro **DATO3**. Por último, se lee la posición 43_H y se pasa al registro **DATO4**. Una vez recuperados los cuatro datos se compara de la siguiente manera: **DATO1** con **D13**, **DATO2** con **D14**, **DATO3** con **D15** y **DATO4** con **D16**. Es importante recordar que **D13**, **D14**, **D15** y **D16** son registros de propósito general definidos para mantener los datos de la línea 1 en la LCD. Ahora bien, al comparar los datos, si todos son iguales se despliega en la LCD “**Clave correcta**” y en caso de que alguno de los datos no sea igual, se debe desplegar “**Clave errónea**” (diagrama de flujo 23). Para este ejemplo se va a aprovechar y explicar direccionamiento indirecto en la RAM, para ello se usan dos registros: **FSR** y **INDF**; el primero es el puntero de

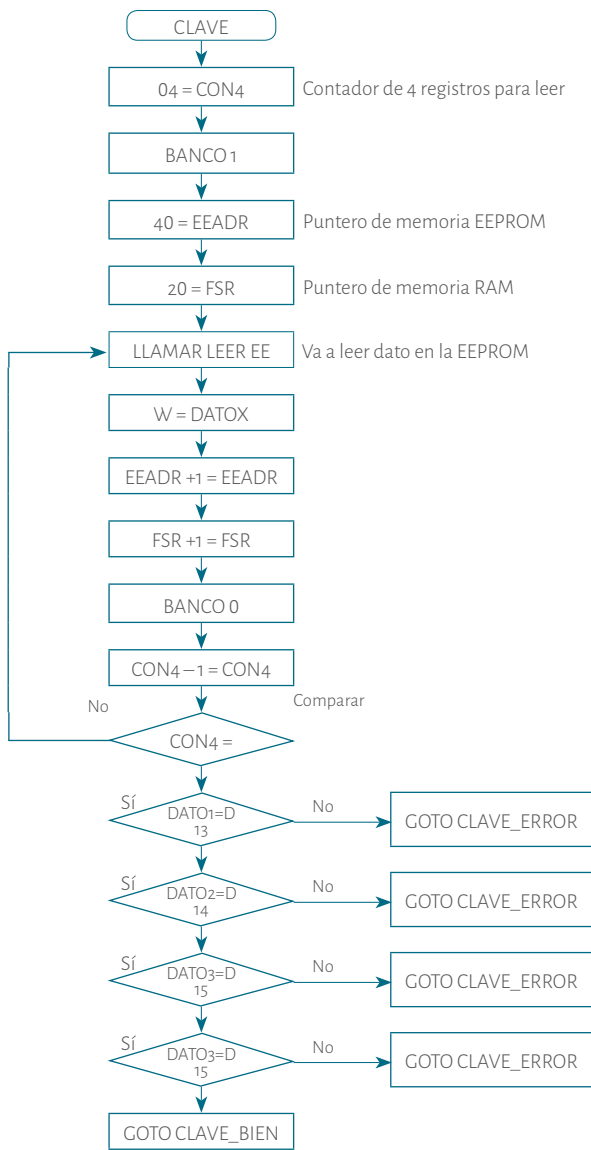
memoria, es decir es el bus de direcciones; el segundo es el que tiene el dato, en otras palabras es el bus de datos. El funcionamiento es: se carga la posición que se desea leer y/o escribir en el registro **FSR** y usando el registro **INDF** se lee y/o se escribe, a través del registro W.

En la subrutina **CLAVE** es llamada otra la cual es **LEER_EE**, que se encarga de habilitar la Memoria EEPROM data, leer el dato y por último deshabilitarla, así como se muestra en el diagrama de flujo 24. También se utilizan **CLAVE_ERROR** y **CLAVE_BIEN**; la primera despliega en la LCD “**clave errónea**”, allí también se llama una subrutina de **PITO_ERROR**. Además se tiene el control de errores los cuales son 3; y la segunda despliega en la LCD “**clave correcta**”. También hay un bit de control de error (bit menos significativo del registro **ERROR_EE**), el cual es controlado por las dos subrutinas anteriores, para ser usado en el subrutina **CAMBIO**, el funcionamiento es muy fácil cuando hay error se garantiza en uno y cuando la clave está correcta se garantiza en cero.

Para el caso de la subrutina **CAMBIO** hay que validar la clave. Por consiguiente, lo primero que debe hacerse es leer la clave. Para ello, se llama a la subrutina **CLAVE** pero con la diferencia de que si la clave es correcta, hay que desplegar en la LCD “**Nueva clave**”. Una vez allí ingresa a la parte de escribir la nueva clave, claro, una vez que se digite la nueva clave se oprime nuevamente la tecla de cambio.

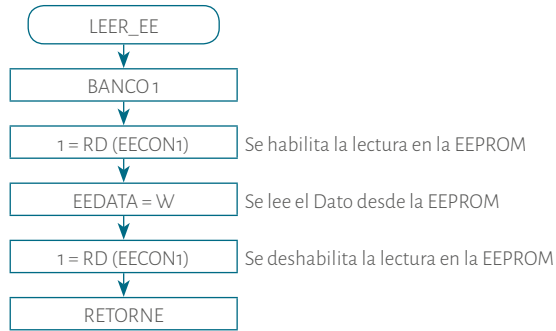
Diagrama 24.

Subrutina CLAVE para leer en la EEPROM Data del ejemplo 19.



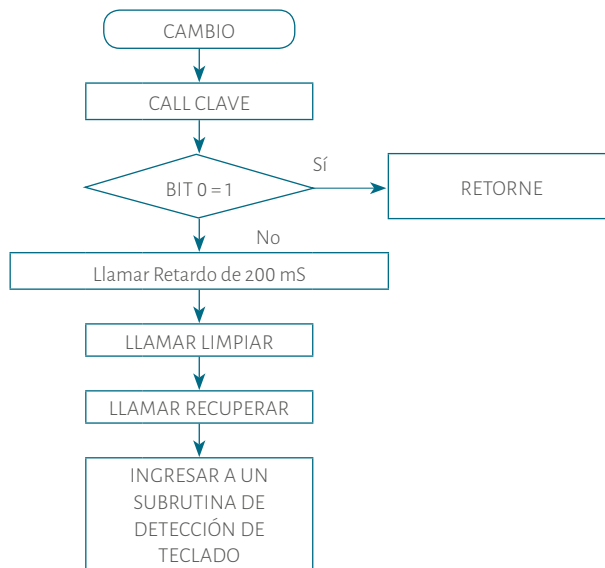
Fuente. Elaboración propia.

Diagrama 25.
Subrutina LEER_EE del ejemplo 19.



Fuente. Elaboración propia

Diagrama 26.
Subrutina CAMBIO para monitorear el teclado del ejemplo 19.

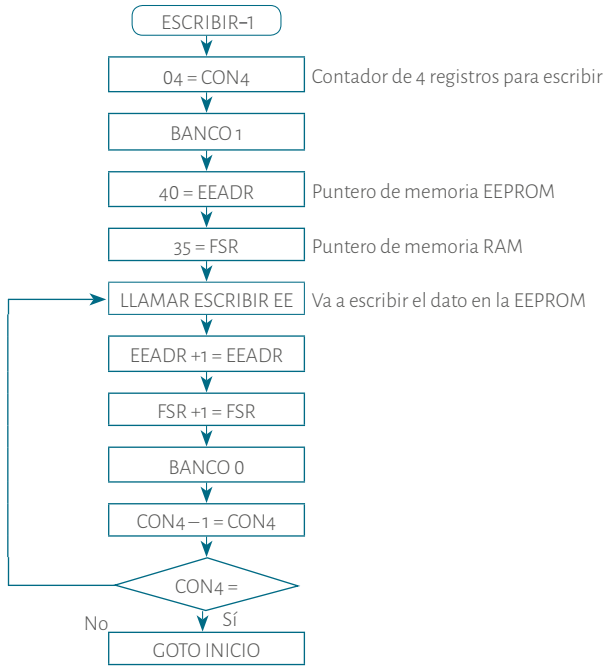


Fuente. Elaboración propia.

En DETECCIÓN DE TECLADO se usan otras subrutinas las cuales son **ESCRIBIR_1** y **ESCRIBIR_EE**. Estas subrutinas se requieren una vez se ha digitado la nueva clave y se encargan de habilitar la memoria EEPROM data, escribir los cuatro datos y deshabilitar la memoria (Diagrama 16 y 17). También se utiliza la subrutina **MENSAJE-CAMBIO**, con la cual se despliega en la LCD **“Nueva clave”**.

Diagrama 27.

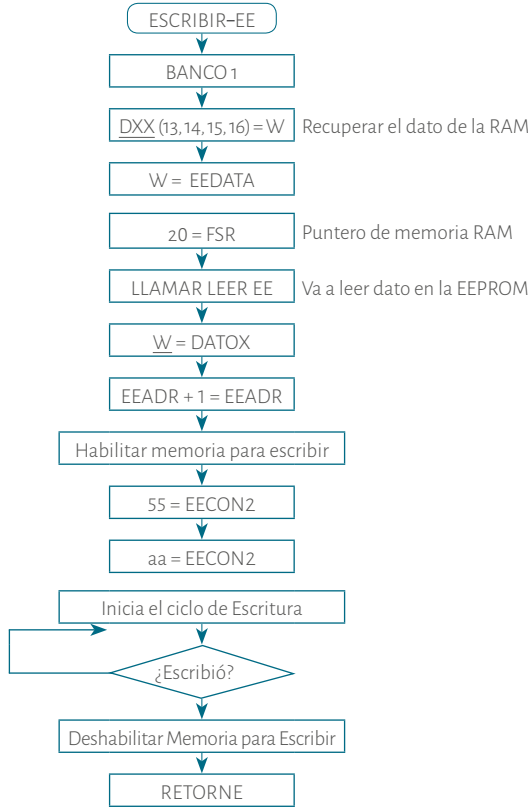
Subrutina ESCRIBIR_1 para control de la EEPROM Data del ejemplo 19.



Fuente. Elaboración propia.

Diagrama 28.

Subrutina ESCRIBIR_EE para escribir en la EEPROM Data del ejemplo 19.



Fuente. Elaboración propia.

Paso 4. El programa es el siguiente (en este caso el programa está completo). El lector podrá observar que en este programa los registros de propósito general están separados con comas, es usado para ahorrar espacio cuando se escribe el programa.

#INCLUDE<p16f628A.inc>

; Definición del microcontrolador es decir la librería.

CBLOCK 0x20

; Definición de registros de propósito general.

DATO1,DATO2,DATO3,DATO4

; REGISTROS QUE CONTIENEN LA CLAVE

CON1,CON2,CON3,CON4

; REGISTROS PARA SUBROUTINAS DE RETARDO.

CON5

; CONTADOR DE ERRORES

D1, D2,D3,D4,D5,D6,D7,D8

; REGISTRO PARA SUBROUTINA DE ROTACIÓN Y

D9,D10,D11,D12,D13,D14,D15,D16

; VISUALIZACIÓN EN LA LCD.

ERROR_EE

ENDC

;

RES_VECT CODE 0X0000 ; Vector reset

PUERTOS

BSF STATUS,5 ; Banco 1 para programar puertos.
 BCF STATUS,6
 BCF TRISB,2 ; B2 SALIDA PARA EL LED
 BCF TRISB,3 ; B3 SALIDA PARA EL PARLANTE
 MOVLW 0XF0 ; PORTA PARA EL TECLADO,
 MOVWF TRISA ; Filas (A₀ hasta A₃) y A₄, A₆, A₇, para las columnas
 BCF STATUS,5 ; Banco 0 para sacar datos.
 BCF STATUS,6
 MOVLW 0X07 ; Se deshabilitan los comparadores analógicos,
 MOVWF CMCON ; el PORTA se toma como salida digital.

MOVLW 0X03 ; SE CARGA CONTADOR DE TRES ERRORES

MOVWF CON5

INICIO

CALL LIMPIAR ; SUBROUTINA DE LIMPIAR LA LCD
 CALL TONO ; SUBROUTINA DE TONO
 CALL LCD_Inicializa ; SE INICIALIZA LA LCD
 CALL RECUPERAR

;

TECLADO1

CALL LED ; LED INDICADOR
 CALL DIGITE_CLAVE ; Mensaje informando para digitar la CLAVE
 BCF PORTA,3 ; SE HABILITA TECLAS (UNO, DOS, TRES)
 BSF PORTA,2
 BSF PORTA,1
 BSF PORTA,0

BTFSF PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO UNO

CALL UNO

BTFSF PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO DOS

CALL DOS

BTFSF PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO TRES

CALL TRES

;;;;

BSF PORTA,3 ; SE HABILITA TECLAS (CUATRO, CINCO, SEIS)

BCF PORTA,2

BSF PORTA,1

BSF PORTA,0

BTFSF PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CUATRO

CALL CUATRO

```

BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CINCO
CALL CINCO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO SEIS
CALL SEIS
;

BSF PORTA,3 ; SE HABILITA TECLAS (SIETE, OCHO, NUEVE)
BSF PORTA,2
BCF PORTA,1
BSF PORTA,0

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO SIETE
CALL SIETE
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO OCHO
CALL OCHO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO NUEVE
CALL NUEVE
;

BSF PORTA,3 ; SE HABILITA TECLAS (CAMBIO, CERO, CLAVE)
BSF PORTA,2
BSF PORTA,1
BCF PORTA,0

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA TECLA CAMBIO
GOTO CAMBIO
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CERO
CALL CERO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN LA TECLA CLAVE
CALL CLAVE

GOTO TECLADO1
;
DIGITE_CLAVE
CALL LCD_Línea 2 ; MENSAJE DIGITE CLAVE
MOVLW 'D' ; Se carga el código ASCII de la letra D
CALL LCD_Character ; Se visualiza en la LCD
MOVLW 'I' ; Se carga el código ASCII de la letra I
CALL LCD_Character ; Se visualiza en la LCD
MOVLW 'C' ; Se carga el código ASCII de la letra C
CALL LCD_Character ; Se visualiza en la LCD
MOVLW 'I' ; Se carga el código ASCII de la letra I
CALL LCD_Character ; Se visualiza en la LCD
MOVLW 'T' ; Se carga el código ASCII de la letra T
CALL LCD_Character ; Se visualiza en la LCD
MOVLW 'E' ; Se carga el código ASCII de la letra E
CALL LCD_Character ; Se visualiza en la LCD
MOVLW ' ' ; Se carga el código ASCII del ESPACIO
CALL LCD_Character ; Se visualiza en la LCD
MOVLW 'C' ; Se carga el código ASCII de la letra C

```

```

CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW 'L' ; Se carga el código ASCII de la letra L
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW 'A' ; Se carga el código ASCII de la letra A
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW 'V' ; Se carga el código ASCII de la letra V
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW 'E' ; Se carga el código ASCII de la letra E
CALL LCD_Caracter ; Se visualiza en la LCD
MOVLW ' ' ; Se carga el código ASCII del ESPACIO
CALL LCD_Caracter ; Se visualiza en la LCD
RETURN

;
TONO_CAMBIO
CALL PITO ; SUBROUTINA DE TONO
CALL Retardo_50ms ; SE USA UNA SERIE DE RETARDOS PARA HACER
CALL PITO ; EFECTO DE PITO LARGO.
CALL Retardo_50ms
CALL PITO
CALL Retardo_50ms
RETURN

;
LED ; LED TESTIGO
BSF PORTB,2 ; Prende LED en B2
CALL Retardo_50ms ; Durante este tiempo.
BCF PORTB,2 ; Apaga LED en B2
CALL Retardo_50ms ; Durante este tiempo.
RETURN

;
CERO ; Subrutina del número CERO.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <0> ; Se carga el código ASCII del número 0
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;
UNO ; Subrutina del número UNO.
CALL PITO ; Se genera tono para indicar que se oprimió la tecla.
CALL ROTA ; Se ROTA los datos para efecto de desplazamiento.
MOVLW <1> ; Se carga el código ASCII del número 1
MOVWF D16
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;

```

```

DOS
CALL PITO ; Subrutina del número DOS.
CALL ROTA ; Se genera tono para indicar que se oprimió la tecla.
MOVWL <2> ; Se ROTA los datos para efecto de desplazamiento.
MOVWF D16 ; Se carga el código ASCII del número 2
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;
TRES
CALL PITO ; Subrutina del número TRES.
CALL ROTA ; Se genera tono para indicar que se oprimió la tecla.
MOVWL <3> ; Se ROTA los datos para efecto de desplazamiento.
MOVWF D16 ; Se carga el código ASCII del número 3
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;
CUATRO
CALL PITO ; Subrutina del número CUATRO.
CALL ROTA ; Se genera tono para indicar que se oprimió la tecla.
MOVWL <4> ; Se ROTA los datos para efecto de desplazamiento.
MOVWF D16 ; Se carga el código ASCII del número 4
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;
CINCO
CALL PITO ; Subrutina del número CINCO.
CALL ROTA ; Se genera tono para indicar que se oprimió la tecla.
MOVWL <5> ; Se ROTA los datos para efecto de desplazamiento.
MOVWF D16 ; Se carga el código ASCII del número 5
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;
SEIS
CALL PITO ; Subrutina del número SEIS.
CALL ROTA ; Se genera tono para indicar que se oprimió la tecla.
MOVWL <6> ; Se ROTA los datos para efecto de desplazamiento.
MOVWF D16 ; Se carga el código ASCII del número 6
CALL RECUPERAR ; Se visualiza en la LCD
CALL Retardo_200ms ; Retardo para evitar antirebote
RETURN ; Retorna desde donde lo llamaron

;
SIETE
CALL PITO ; Subrutina del número SIETE.

```

```

CALL      ROTA      ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <7>      ; Se carga el código ASCII del número 7
MOVWF    D16
CALL     RECUPERAR ; Se visualiza en la LCD
CALL     Retardo_200ms ; Retardo para evitar antirebote
RETURN   ; Retorna desde donde lo llamaron

;
OCHO     ; Subrutina del número OCHO.
CALL     PITO      ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA      ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <8>      ; Se carga el código ASCII del número 8
MOVWF    D16
CALL     RECUPERAR ; Se visualiza en la LCD
CALL     Retardo_200ms ; Retardo para evitar antirebote
RETURN   ; Retorna desde donde lo llamaron

;
NUEVE   ; Subrutina del número NUEVE.
CALL     PITO      ; Se genera tono para indicar que se oprimió la tecla.
CALL     ROTA      ; Se ROTA los datos para efecto de desplazamiento.
MOVLW    <9>      ; Se carga el código ASCII del número 9
MOVWF    D16
CALL     RECUPERAR ; Se visualiza en la LCD
CALL     Retardo_200ms ; Retardo para evitar antirebote
RETURN   ; Retorna desde donde lo llamaron

;
;
LIMPIAR ; SUBROUTINA DE LIMPIAR LA LCD
MOVLW    <<      ; Se carga el código ASCII del ESPACIO
MOVWF    D1      ; Solo es garantizar el espacio en todos los registros
MOVWF    D2
MOVWF    D3
MOVWF    D4
MOVWF    D5
MOVWF    D6
MOVWF    D7
MOVWF    D8
MOVWF    D9
MOVWF    D10
MOVWF    D11
MOVWF    D12
MOVWF    D13
MOVWF    D14
MOVWF    D15
MOVWF    D16
RETURN

;
PITO    ; SUBROUTINA DE GENERAR EL TONO

```

PITO₁

```

MOVLW 0x10 ; Número de veces para repetir el ON OFF en B3
MOVWF CON3 ; Es decir la duración del pito

BSF PORTB,3 ; Prende B3
CALL TMPITO ; Tiempo para el tono
BCF PORTB,3 ; Apaga B3
CALL TMPITO ; Tiempo para el tono
DECFSZ CON3,1 ; Pregunta si ya lo repitio las veces que diga CON3
GOTO PITO1
RETURN

```

;

ROTA

```

; SUBROUTINA DE DESPLAZAMIENTO A LA IZQUIERDA EN LA LCD
; Es decir la que permite el efecto de corrimiento de los caracteres

```

```

MOVF D2,W ; D2 = D1
MOVWF D1
MOVF D3,W ; D3 = D2
MOVWF D2
MOVF D4,W ; D4 = D3
MOVWF D3
MOVF D5,W ; D5 = D4
MOVWF D4
MOVF D6,W ; D6 = D5
MOVWF D5
MOVF D7,W ; D7 = D6
MOVWF D6
MOVF D8,W ; D8 = D7
MOVWF D7
MOVF D9,W ; D9 = D8
MOVWF D8
MOVF D10,W ; D10 = D9
MOVWF D9
MOVF D11,W ; D11 = D10
MOVWF D10
MOVF D12,W ; D12 = D11
MOVWF D11
MOVF D13,W ; D13 = D12
MOVWF D12
MOVF D14,W ; D14 = D13
MOVWF D13
MOVF D15,W ; D15 = D14
MOVWF D14
MOVF D16,W ; D16 = D15
MOVWF D15
RETURN

```

;

CLAVE

```

; Subrutina de control para leer la EEPROM
MOVLW 0x04 ; Son 4 dígitos por leer

```

```

MOVWF CON4
BSF STATUS,5 ; BANCO 1
MOVLW 0X40 ; POSICIÓN DEL PRIMER DATO EN LA EEDATA,
MOVWF EEADR ; PARA COMPARAR.
MOVLW 0X20 ; POSICIÓN DEL PRIMER DATO EN LA RAM,
MOVWF FSR ; PARA COMPARAR.

OTRO_DATO
CALL LEER_EE ; SE LEE EL DATO
MOVWF INDF
INCF EEADR,1 ; SE INCREMENTA POSICIÓN PARA EL
INCF FSR,1 ; SIGUIENTE DATO.
BCF STATUS,5 ; BANCO 0
DECFSZ CON4,1 ; SE PREGUNTA SI YA LEYÓ LOS 4 DATOS.
GOTO OTRO_DATO
;
CAMPARAR ; Subrutina para comparar datos digitados en la RAM
; con los almacenados en la EEPROM
MOVF DATO1,W ; SE COMPARA EL PRIMER DATO
XORWF D13,W
BTSS STATUS,2 ; SE PREGUNTA SI DATO1 = D13
GOTO CLAVE_ERROR ; LA CLAVE ES ERRÓNEA
MOVF DATO2,W ; SE COMPARA EL SEGUNDO DATO
XORWF D14,W
BTSS STATUS,2 ; SE PREGUNTA SI DATO2 = D14
GOTO CLAVE_ERROR ; LA CLAVE ES ERRÓNEA
MOVF DATO3,W ; SE COMPARA EL TERCER DATO
XORWF D15,W
BTSS STATUS,2 ; SE PREGUNTA SI DATO3 = D15
GOTO CLAVE_ERROR ; LA CLAVE ES ERRÓNEA
MOVF DATO4,W ; SE COMPARA EL CUARTO DATO
XORWF D16,W
BTSS STATUS,2 ; SE PREGUNTA SI DATO4 = D16
GOTO CLAVE_ERROR ; LA CLAVE ES ERRÓNEA
GOTO CLAVE_BIEN ; LA CLAVE ES CORRECTA.
;
LEER_EE
BSF STATUS,5 ; BANCO 1
BSF EECON1,0 ; SE HABILITA LA LECTURA EN LA EEPROM DATA
MOVF EEDATA,W ; SE RECUPERA EL DATO
BCF EECON1,0 ; SE DESHABILITA LA LECTURA EN LA EEPROM DATA
RETURN
;
CLAVE_ERROR ; SE INDICA QUE HAY ERROR AL DIGITAR
BSF ERROR_EE,0 ; LA CLAVE; SE ESCRIBE XXXX XXX1 EN
; EL REGISTRO ERROR
CALL LIMPIAR ; SUBRRUTINA DE LIMPIAR LA LCD
CALL RECUPERAR

```



```

CALL    LCD_Línea z           ; MENSAJE DE ERROR
MOVLW  <C>                   ; Se carga el código ASCII de la letra C
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <L>                   ; Se carga el código ASCII de la letra L
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <A>                   ; Se carga el código ASCII de la letra A
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <V>                   ; Se carga el código ASCII de la letra V
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <E>                   ; Se carga el código ASCII de la letra E
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <<                     ; Se carga el código ASCII del ESPACIO
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <E>                   ; Se carga el código ASCII de la letra E
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <R>                   ; Se carga el código ASCII de la letra R
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <R>                   ; Se carga el código ASCII de la letra R
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <O>                   ; Se carga el código ASCII de la letra O
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <N>                   ; Se carga el código ASCII de la letra N
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <E>                   ; Se carga el código ASCII de la letra E
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <A>                   ; Se carga el código ASCII de la letra A
CALL    LCD_Caracter         ; Se visualiza en la LCD
DECSFZ CON5,1                ; UN ERROR MENOS
GOTO   PITO_ERROR            ; TIENE OTRA OPORTUNIDAD
GOTO   ESPERA                ; NO TIENE MÁS OPORTUNIDADES

;
CLAVE_BIEN

BCF    ERROR_EE,0            ; SE INDICA QUE NO HAY ERROR AL DIGITAR
                                           ; LA CLAVE; SE ESCRIBE XXXX XXX0 EN EL
                                           ; REGISTRO ERROR

CALL    LCD_Línea z           ; MENSAJE DE CLAVE CORRECTA
MOVLW  <C>                   ; Se carga el código ASCII de la letra C
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <L>                   ; Se carga el código ASCII de la letra L
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <A>                   ; Se carga el código ASCII de la letra A
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <V>                   ; Se carga el código ASCII de la letra V
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <E>                   ; Se carga el código ASCII de la letra E
CALL    LCD_Caracter         ; Se visualiza en la LCD
MOVLW  <<                     ; Se carga el código ASCII del ESPACIO

```

```

CALL LCD_Character ; Se visualiza en la LCD
MOVLW <C> ; Se carga el código ASCII de la letra C
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <O> ; Se carga el código ASCII de la letra O
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <R> ; Se carga el código ASCII de la letra R
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <R> ; Se carga el código ASCII de la letra R
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <E> ; Se carga el código ASCII de la letra E
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <C> ; Se carga el código ASCII de la letra C
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <T> ; Se carga el código ASCII de la letra T
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <A> ; Se carga el código ASCII de la letra A
CALL LCD_Character ; Se visualiza en la LCD
CALL Retardo_2s ; TIEMPO DE ESPERA
RETURN

;
CAMBIO

CALL CLAVE ; Va a leer la clave
BTFSK ERROR_EE,0 ; Se pregunta si s digito la clave correcta, es decir
; se valida la clave
RETURN ; La clave no es correcta.
CALL Retardo_200ms ; TIEMPO DE ESPERA
CALL LIMPIAR ; SUBRRUTINA DE LIMPIAR LA LCD
CALL RECUPERAR

;
TECLADO_CAMBIO

CALL LED ; LED INDICADOR
CALL MENSAJE_CAMBIO ; DESPLIEGA EN LA LCD CAMBIE LA CLAVE
BCF PORTA,3 ; SE HABILITA TECLAS (UNO, DOS, TRES)
BSF PORTA,2
BSF PORTA,1
BSF PORTA,0

BTFSK PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO UNO
CALL UNO
BTFSK PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO DOS
CALL DOS
BTFSK PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO TRES
CALL TRES

;;

BSF PORTA,3 ; SE HABILITA TECLAS (CUATRO, CINCO, SEIS)
BCF PORTA,2
BSF PORTA,1
BSF PORTA,0

```

```

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CUATRO
CALL CUATRO
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CINCO
CALL CINCO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO SEIS
CALL SEIS
;

BSF PORTA,3 ; SE HABILITA TECLAS (SIETE, OCHO, NUEVE)
BSF PORTA,2
BCF PORTA,1
BSF PORTA,0

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN EL NÚMERO SIETE
CALL SIETE
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO OCHO
CALL OCHO
BTFFS PORTA,7 ; SE PREGUNTA SI OPRIMEN EL NÚMERO NUEVE
CALL NUEVE;
;

BSF PORTA,3 ; SE HABILITA TECLAS (ESCRIBIR_1, CERO)
BSF PORTA,2
BSF PORTA,1
BCF PORTA,0

BTFFS PORTA,4 ; SE PREGUNTA SI OPRIMEN LA TECLA ESCRIBIR_1
GOTO ESCRIBIR_1
BTFFS PORTA,6 ; SE PREGUNTA SI OPRIMEN EL NÚMERO CERO
CALL CERO

GOTO TECLADO_CAMBIO
;
MENSAJE_CAMBIO
CALL LCD_Línea 2 ; MENSAJE DE CAMBIO DE CLAVE
MOVLW <N> ; Se carga el código ASCII de la letra N
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <U> ; Se carga el código ASCII de la letra U
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <E> ; Se carga el código ASCII de la letra E
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <V> ; Se carga el código ASCII de la letra V
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <A> ; Se carga el código ASCII de la letra A
CALL LCD_Character ; Se visualiza en la LCD
MOVLW << ; Se carga el código ASCII del ESPACIO
CALL LCD_Character ; Se visualiza en la LCD
MOVLW <C> ; Se carga el código ASCII de la letra C
CALL LCD_Character ; Se visualiza en la LCD

```

MOVLW	<L>	; Se carga el código ASCII de la letra L
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVLW	<A>	; Se carga el código ASCII de la letra A
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVLW	<V>	; Se carga el código ASCII de la letra V
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVLW	<E>	; Se carga el código ASCII de la letra E
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVLW	<<	; Se carga el código ASCII del ESPACIO
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVLW	<<	; Se carga el código ASCII del ESPACIO
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVLW	<<	; Se carga el código ASCII del ESPACIO
CALL	LCD_Caracter	; Se visualiza en la LCD
RETURN		
;		
RECUPERAR		
CALL	LCD_Borra	; Borra la pantalla
CALL	LCD_Línea 1	; Se escribe en Línea 1 de la LCD
MOVF	D1,W	; Se recupera D1
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D2,W	; Se recupera D2
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D3,W	; Se recupera D3
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D4,W	; Se recupera D4
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D5,W	; Se recupera D5
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D6,W	; Se recupera D6
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D7,W	; Se recupera D7
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D8,W	; Se recupera D8
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D9,W	; Se recupera D9
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D10,W	; Se recupera D10
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D11,W	; Se recupera D11
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D12,W	; Se recupera D12
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D13,W	; Se recupera D13
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D14,W	; Se recupera D14
CALL	LCD_Caracter	; Se visualiza en la LCD
MOVF	D15,W	; Se recupera D15

```

CALL    LCD_Character    ; Se visualiza en la LCD
MOVWF  D16,W            ; Se recupera D16
CALL    LCD_Character    ; Se visualiza en la LCD
RETURN

;
PITO_ERROR
CALL    PITO              ; SUBROUTINA DE PITO DE ERROR DE CLAVE
CALL    PITO
CALL    Retardo_50ms
CALL    PITO
CALL    PITO
CALL    Retardo_50ms
CALL    PITO
CALL    PITO
CALL    Retardo_50ms
CALL    PITO
CALL    PITO
CALL    Retardo_50ms
CALL    Retardo_2s      ; TIEMPO DE ESPERA
CALL    Retardo_2s      ; TIEMPO DE ESPERA
RETURN

;
PITO              ; SUBROUTINA DE GENERAR EL TONO
MOVWLW  0x10        ; Número de veces para repetir el ON OFF en B3
MOVWF  CON3         ; Es decir la duración del pito

PITO1
BSF    PORTB,3      ; Prende B3
CALL   TMPITO       ; Tiempo para el Tono
BCF    PORTB,3      ; Apaga B3
CALL   TMPITO       ; Tiempo para el Tono
DECFSZ CON3,1       ; Pregunta si ya lo repitió las veces que diga CON3
GOTO   PITO1
RETURN

;
TMPITO
MOVWLW  0x10        ; SUBROUTINA TIEMPO PARA PITO
MOVWF  CON1

DEC6
MOVWLW  0x10
MOVWF  CON2

DEC7
DECFSZ CON2,1
GOTO   DEC7
DECFSZ CON1,1
GOTO   DEC6
RETURN

;

```

```

ESPERA
MOVWLW  .120           ; CONTADOR DE 120 SEGUNDOS ES DECIR 2 MINUTOS
MOVWFW  CON4           ; TIEMPO EN QUE EL TECLADO SE DESHABILITA
CALL    PITO_ERROR
CALL    Retardo_50ms
CALL    PITO_ERROR
CALL    Retardo_50ms
CALL    Retardo_1s     ; TIEMPO DE ESPERA
DECFSZ  CON4,1
GOTO    $-2
GOTO    PUERTOS

;;
ESCRIBIR_1
MOVWLW  0X04           ; SUBROUTINA DE ESCRIBIR EN LA MEMORIA
MOVWFW  CON4           ; EEPROM
BSF     STATUS,5      ; BANCO 1
MOVWLW  0X40           ; POSICIÓN DEL PRIMER DATO EN LA EEDATA,
MOVWFW  EEADR          ; PARA ESCRIBIR.
MOVWLW  0X35           ; POSICIÓN DEL PRIMER DATO EN LA RAM,
MOVWFW  FSR            ; PARA ESCRIBIR.

OTRO_DATO_1
CALL    ESCRIBIR_EE   ; SE ESCRIBE EL DATO
INCF   EEADR,1        ; SE INCREMENTA POSICIÓN PARA EL
INCF   FSR,1          ; SIGUIENTE DATO.
BCF    STATUS,5      ; BANCO 0
DECFSZ CON4,1        ; SE PREGUNTA SI YA ESCRIBIÓ LOS CUATRO DATOS.
GOTO   OTRO_DATO_1
GOTO   INICIO

;
ESCRIBIR_EE
BSF    STATUS,5      ; BANCO 1
MOVW   INDFW         ; RECUPERO EL DATO DE LA RAM.
MOVWFW EEDATA        ; ESCRIBO EN LA EEDATA.
BSF    EECON1,2      ; HABILITÓ LA MEMORIA EEDATA PARA ESCRIBIR.
MOVWLW 0X55          ; TIEMPO NECESARIO PARA ESCRIBIR EN LA
MOVWFW EECON2        ; MEMORIA EEDATA.
MOVWLW 0XAA
MOVWFW EECON2
BSF    EECON1,1
BTFS   EECON1,1      ; PREGUNTÓ SI YA FUE ESCRITO EL DATO.
GOTO   $-1
BCF    EECON1,2      ; DESHABILITÓ LA MEMORIA EEDATA PARA ESCRIBIR.
RETURN

;
#include <LCD_4BIT-B.INC> ; Subrutinas de control del módulo LCD.
#include <RETARDOS.INC>  ; Subrutinas de retardo.

END

```

En el programa anterior se usó **GOTO \$-X**; esta es usada para hacer salto atrás del programa, las posiciones que se devuelven están dadas por **X**; si se quiere hacer saltos adelante del programa es **GOTO \$+X**; las posiciones que se adelantan están dadas por **X**.

Paso 5. Programar el microcontrolador, armar el circuito y probar. Recordar que para este ejemplo se usa el mismo del ejercicio anterior.

Paso 6. Si es necesario hacer las correcciones pertinentes y volver al paso tres.

4.5. Manejo de motor paso a paso

Según Thimoty, los motores DC se dividen en tres grupos: 1) Rotor devanado, 2) imán permanente y 3) conmutados electrónicamente o motores paso a paso (Thimoty, 1996, p. 530). Para el siguiente ejemplo utilizaremos el tercer tipo de motor.

Ejemplo 20. Se desea manejar un motor paso a paso de 4 bobinas de 1.8 grados por paso, de 5Vdc, unipolar. Hay 4 pulsadores ubicados en cruz como lo muestra la figura 46. Cuando se oprime el pulsador de 90 grados el microcontrolador ubica el motor en dicha posición, si se oprime el pulsador de 270 grados el motor es ubicado allí, si se oprime el pulsador de 180 grados entonces el microcontrolador sabe que la última posición fue 270 entonces toma la decisión de devolver el motor 90 grados. Lo anterior implica que el programa está en la capacidad de saber cuál es la posición inicial y cuál es la posición final y, de esa forma, toma la decisión para que el motor gire a la derecha o la izquierda, según la diferencia. No se debe olvidar el led indicador de funcionamiento.

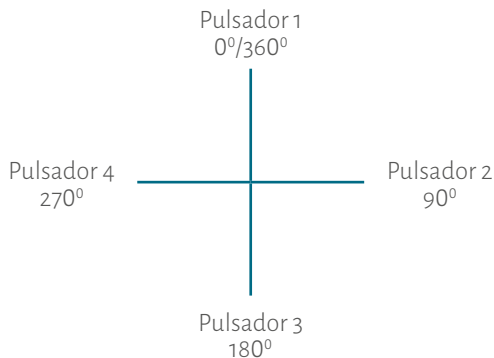


Figura 46. Diagrama de grados para el motor paso a paso del ejemplo 20.

A continuación se desarrollará paso por paso:

Paso 1. Los motores paso a paso están divididos en tres grupos según su construcción: 1) de imán permanente, 2) de reluctancia variable, 3) híbridos. Los primeros son los más usados y conocidos y están contruidos por: 1) el rotor, el cual es un imán permanente en forma de disco, 2) el estator, que tiene forma cilíndrica y en su interior están las bobinas (Thimoty, 1996). Para entender el funcionamiento se explicará con un motor de cuatro bobinas unipolar de 1.8 grados por paso de $5 V_{DC}$. En el esquema que se muestra en la Figura 47 se observa dicho motor.

Se asume que el orden las bobinas es como está en la Figura 47. Para energizar el motor es preciso hacerlo bobina por bobina, tal y como se explica a continuación. El primer paso es energizar L1 y las otras tres no se energizan, como resultado de las fuerzas de atracción-repulsión el rotor gira dentro del campo magnético del estator, solo 1.8 grados. El segundo paso es energizar L2 y las otras tres no se energizan, el motor gira otros 1.8 grados, es decir ya ha girado 3.6 grados. El tercer paso es energizar L3 y las otras tres no se energizan, el motor gira otros 1.8 grados, es decir ya ha girado 5.4 grados. El cuarto paso es energizar L4 y las otras tres no se energizan, el motor gira otros 1.8 grados, es decir ya ha girado 7.2 grados y posteriormente se energiza nuevamente la bobina uno y así sucesivamente, así como lo muestra la tabla 23, en la cual se observa cómo se hace la secuencia de los pasos; es así como el motor gira a la derecha o a la izquierda según el orden de como se energicen las bobinas, es decir; L1, L2, L3, L4 a la derecha y L4, L3, L2, L1 a la izquierda.

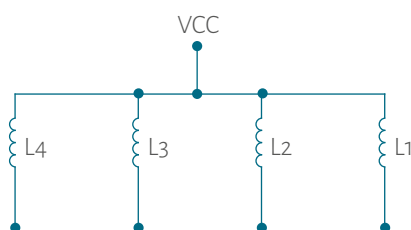


Figura 47. Esquema de un motor paso a paso de cuatro bobinas, unipolar.

Fuente. Elaboración propia.

Tabla 23.

Orden de energizar las bobinas de un motor paso a paso de cuatro bobinas, unipolar.

L1	L2	L3	L4	Grados
GND	Vcc	Vcc	Vcc	1.8
Vcc	GND	Vcc	Vcc	3.6
Vcc	Vcc	GND	Vcc	5.4
Vcc	Vcc	Vcc	GND	7.2

Fuente. Elaboración propia.

En este ejemplo se está trabajando con motor paso a paso de 1.8 grados. Para dar una vuelta completa (360°) hay que darle 200 pasos. Con la siguiente ecuación se pueden determinar cuántos pasos se requieren para un determinado grado:

$$x = \frac{N.Grados * 200 \text{ pasos}}{360 \text{ Grados}}$$

Donde:

X = Número de paso por girar

N. Grados = Número de grados a girar

Así, por ejemplo, si se desea hacer girar el motor 90 grados, el número de pasos que se deben dar al motor es:

$$x = \frac{90 \text{ Grados} * 200 \text{ pasos}}{360 \text{ Grados}}$$

$$x = 50 \text{ pasos}$$

El lector puede usar esta ecuación para cualquier motor, sólo debe saber qué grados giran por paso. En la tabla 24 se puede apreciar el número de pasos que debe girar el motor para cada uno de los grados del ejercicio.

Tabla 24.

Grados por girar.

Grados	Pasos
90	50
180	100
270	150
360	200

Fuente. Elaboración propia.

Paso 2. En este ejemplo se puede tomar la parte baja del puerto B como entrada para los cuatro pulsadores: $B0=90$, $B1=180$, $B2=270$ y $B3=0/360$. La parte alta se toma como salida para las 4 bobinas del motor, usando transistores para el manejo de potencia. Por último, el led indicador se coloca en A0. En la figura 48 se aprecia este proceso.

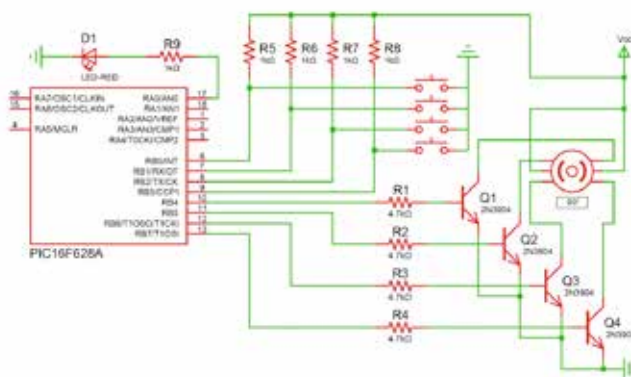


Figura 48. Diagrama de conexiones del ejemplo 20.

Fuente. Elaboración José Luis González. (La distribución de pines fue tomada de Proteus).

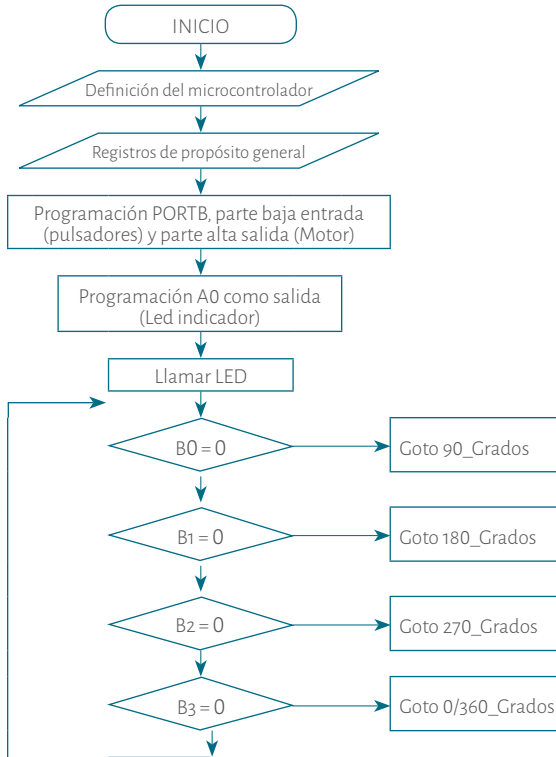
Paso 3. En este ejemplo el programa principal se va a encargar de monitorear los 4 pulsadores. Se debe recordar que se desea girar 90° , 180° , 270° y $0/360^\circ$ (diagrama de flujo 29). Para cada grado hay una subrutina. En el diagrama de flujo 30 se muestra el caso de la subrutina de 90° . En los grados restantes el proceso es el mismo, la única diferencia es el número de grados por girar (ver tabla 24).

Para el control del sentido de giro (derecha o izquierda), y el número de grados que debe girar el motor según la posición actual y la posición deseada, se cuenta con una subrutina llamada **EVALUAR**, esta se muestra en el diagrama de flujo 31, allí se determina si el motor debe girar a la derecha (subrutina **GIRE_D**, diagrama de flujo 32), o debe girar a la izquierda (subrutina **GIRE_I**, diagrama de flujo 33), o no debe girar, es decir que debe ir al programa principal.

En las subrutinas **GIRE_D** y **GIRE_I**, lo primero que se hace es evaluar cuál de las 4 bobinas fue la última energizada, siguiendo la tabla 23, por ejemplo, si se está en la subrutina **GIRE_D** y la última bobina energizada fue la 2, entonces la siguiente bobina a energizar es la tres. Por el contrario, si se encuentra en la subrutina **GIRE_I** y la última bobina energizada fue la dos, la siguiente bobina a energizar es

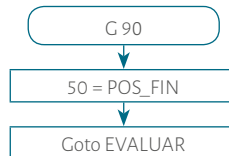
la uno. En el diagrama de flujo 32 se muestra la subrutina **GIRE_D** y en la subrutina **GIRE_I** se observa en el diagrama de flujo 33.

Diagrama 29.
Programa principal del ejemplo 20.



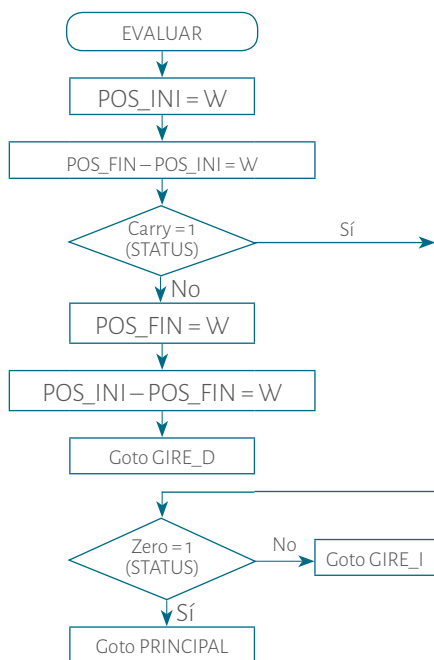
Fuente. Elaboración propia.

Diagrama 30.
Subrutina **G_90** del ejemplo 20.



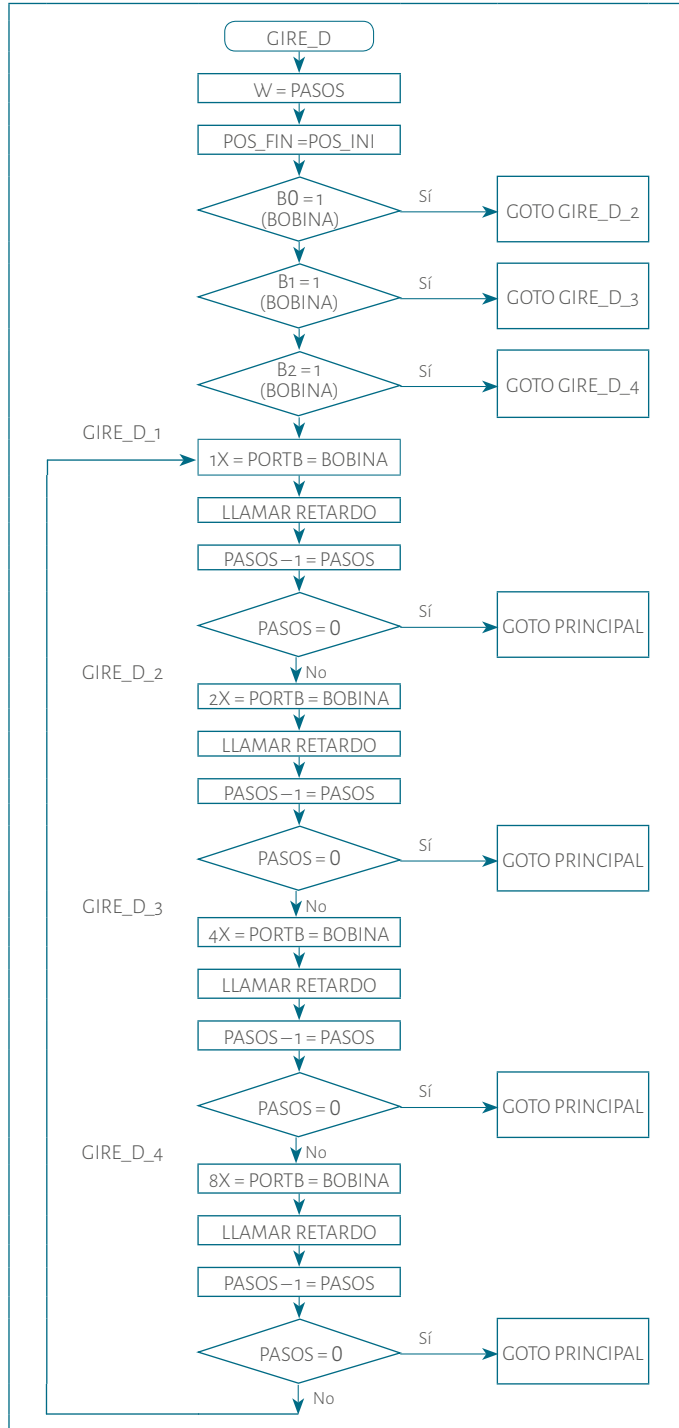
Fuente. Elaboración propia

Diagrama 31.
Subrutina EVALUAR del ejemplo 20.



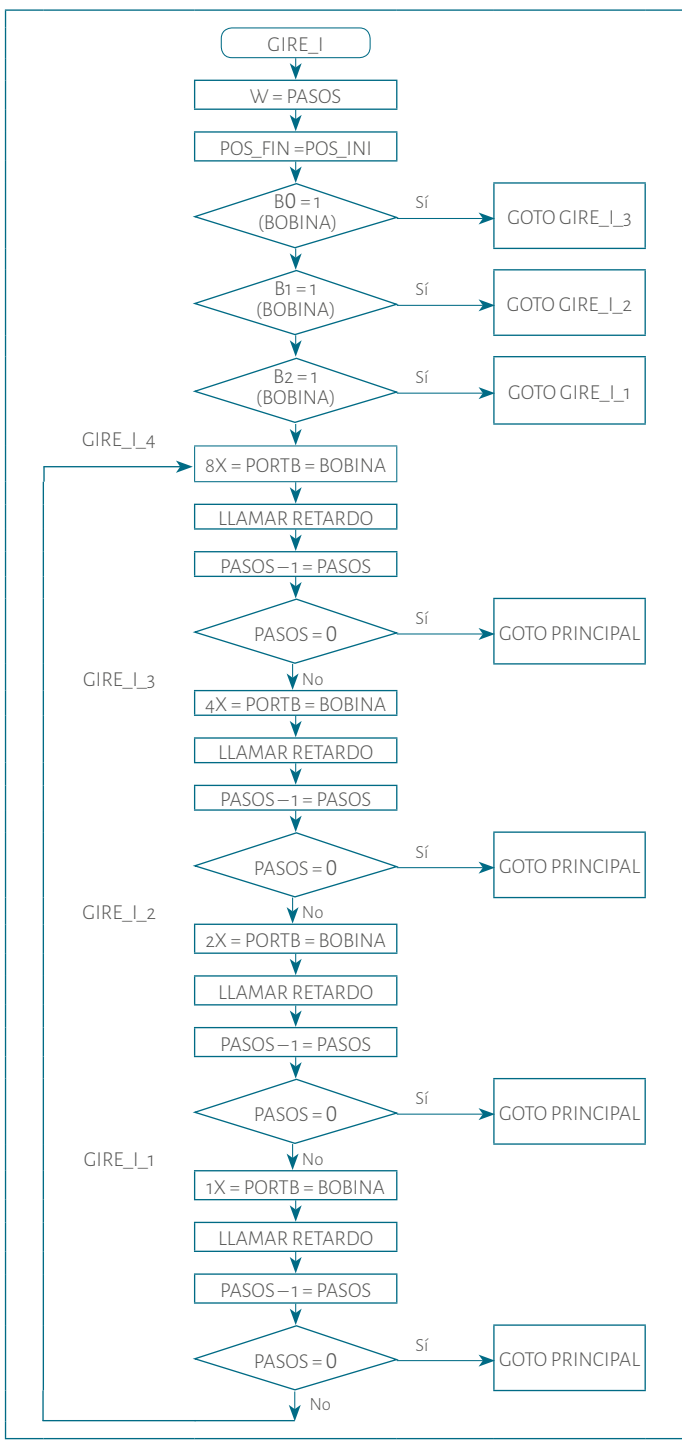
Fuente. Elaboración propia.

Diagrama 32.
Subrutina GIRE_D del ejemplo 20.



Fuente. Elaboración propia.

Diagrama 33.
Subrutina GIRE_I del ejemplo 20.



Fuente. Elaboración propia.

La subrutina de **LED** y de **TIEMPO** son iguales a la de los ejemplos anteriores donde se usan.

Paso 4. El programa es el siguiente (el programa está completo):

```
#INCLUDE<p16f628A.inc>           ; Definición del microcontrolador es decir la librería.
;
CBLOCK    0X20
CON1      ; REGISTRO PARA LA RUTINA DE TIEMPO
CON2
POS_INI   ; REGISTRO QUE TIENE LA POSICIÓN INICIAL
POS_FIN   ; REGISTRO QUE TIENE LA POSICIÓN FINAL.
PASOS     ; REGISTRO PARA CONTADOR DE PASOS.
BOBINA    ; REGISTRO PARA CONTROL DE BOBINAS ENERGIZADAS.
ENDC

RES_VECT  CODE 0X0000           ; Vector reset
;
PUERTOS
BSF       STATUS,5             ; BANCO 1 PARA PROGRAMAR PUERTOS.
BCF       STATUS,6
MOVLW    0X0F                  ; PARTE BAJA DEL PORTB COMO ENTRADA PARA LOS
MOVWF    TRISB                 ; CUATRO PULSADORES. PARTE ALTA DEL PORTB
; COMO SALIDA PARA EL MOTOR.
BCF       TRISA,0              ; A0 COMO SALIDA PARA EL LED TESTIGO
BCF       STATUS,5             ; BANCO 0 PARA SACAR DATOS.
BCF       STATUS,6
MOVLW    0X07                  ; SE DESHABILITAN LOS COMPARADORES
MOVWF    CMCON                 ; ANALÓGICOS EL PORTA
;
PRINCIPAL
CALL     LED
BTFSS   PORTB,0                ; GIRA 45 GRADOS.
GOTO    C_90

BTFSS   PORTB,1                ; GIRA 90 GRADOS.
GOTO    C_180

BTFSS   PORTB,2                ; GIRA 135 GRADOS.
GOTO    C_270

BTFSS   PORTB,3                ; GIRA 180 GRADOS.
GOTO    C_0_360

GOTO    PRINCIPAL
```

```

;;
G_90
    MOVLW    .50
    MOVWF   POS_FIN      ; PASOS PARA GIRAR 90 GRADOS.
    GOTO    EVALUAR

;;
G_180
    MOVLW    .100
    MOVWF   POS_FIN      ; PASOS PARA GIRAR 90 GRADOS.
    GOTO    EVALUAR

;;
G_270
    MOVLW    .150
    MOVWF   POS_FIN      ; PASOS PARA GIRAR 90 GRADOS.
    GOTO    EVALUAR

;;
G_0_360
    MOVLW    .200
    MOVWF   POS_FIN      ; PASOS PARA GIRAR 90 GRADOS.
    GOTO    EVALUAR

;;
EVALUAR
    MOVF    POS_INI,W      ; SE RECUPERA LA POSICIÓN ACTUAL.
    SUBWF   POS_FIN,W      ; POS_FIN - POS_INI = W.
    BTFS   STATUS,0      ; SI EL CARRY ES UNO SALTE.
    GOTO    CASI
    MOVF    POS_FIN,W
    SUBWF   POS_INI,W      ; POS_INI - POS_FIN = W.
    GOTO    GIRE_D        ; GIRA A LA IZQUIERDA

CASI
    BTFS   STATUS,2
    GOTO    GIRE_I        ; GIRA A LA DERECHA
    GOTO    PRINCIPAL    ; NO GIRA NADA

;;
GIRE_D
    MOVWF   PASOS        ; SE RECUPERA EL CONTADOR DE LOS GRADOS A
    MOVF    POS_FIN,W    ; GIRAR.
    MOVWF   POS_INI
    BTFS   BOBINA,0      ; LA PRIMERA BOBINA FUE LA ÚLTIMA ENERGIZADA?
    GOTO    GIRE_D_2
    BTFS   BOBINA,1      ; LA SEGUNDA BOBINA FUE LA ÚLTIMA ENERGIZADA?
    GOTO    GIRE_D_3
    BTFS   BOBINA,2      ; LA TERCERA BOBINA FUE LA ÚLTIMA ENERGIZADA?
    GOTO    GIRE_D_4
    GOTO    GIRE_D_1

;*****
GIRE_D_1

```



```

BSF    PORTB,4    ; SE DA PRIMER PASO
BCF    PORTB,5
BCF    PORTB,6
BCF    PORTB,7
MOVLW 0X01    ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA PRIMERA
MOVWF  BOBINA
CALL   RETARDO
DECFSZ PASOS,1    ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
                    ; DESEADA.

GOTO   GIRE_D_2    ; EL MOTOR NO HA LLEGADO.
GOTO   PRINCIPAL   ; EL MOTOR YA LLEGÓ.

GIRE_D_2
BCF    PORTB,4    ; SE DA SEGUNDO PASO
BSF    PORTB,5
BCF    PORTB,6
BCF    PORTB,7
MOVLW 0X02    ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA SEGUNDA
MOVWF  BOBINA
CALL   RETARDO
DECFSZ PASOS,1    ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
                    ; DESEADA.

GOTO   GIRE_D_3    ; EL MOTOR NO HA LLEGADO.
GOTO   PRINCIPAL   ; EL MOTOR YA LLEGÓ.

GIRE_D_3
BCF    PORTB,4
BCF    PORTB,5
BSF    PORTB,6
BCF    PORTB,7
MOVLW 0X04    ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA TERCERA
MOVWF  BOBINA
CALL   RETARDO
DECFSZ PASOS,1    ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
                    ; DESEADA.

GOTO   GIRE_D_4    ; EL MOTOR NO HA LLEGADO.
GOTO   PRINCIPAL   ; EL MOTOR YA LLEGÓ.

GIRE_D_4
BCF    PORTB,4
BCF    PORTB,5
BCF    PORTB,6
BSF    PORTB,7
MOVLW 0X08    ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA CUARTA
MOVWF  BOBINA
CALL   RETARDO
DECFSZ PASOS,1    ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
                    ; DESEADA.

GOTO   GIRE_D_1    ; EL MOTOR NO HA LLEGADO.
GOTO   PRINCIPAL   ; EL MOTOR YA LLEGÓ.

```

;;;

GIRE_I

```

MOVWF PASOS ; SE RECUPERA EL CONTADOR DE LOS GRADOS A
MOVF POS_FIN,W ; GIRAR.
MOVWF POS_INI
BTFSZ BOBINA,3 ; LA CUARTA BOBINA FUE LA ÚLTIMA ENERGIZADA?
GOTO GIRE_I_3
BTFSZ BOBINA,2 ; LA TERCERA BOBINA FUE LA ÚLTIMA ENERGIZADA?
GOTO GIRE_I_2
BTFSZ BOBINA,1 ; LA SEGUNDA BOBINA FUE LA ÚLTIMA ENERGIZADA?
GOTO GIRE_I_1
GOTO GIRE_I_4

```

GIRE_I_4

```

BCF PORTB,4
BCF PORTB,5
BCF PORTB,6
BSF PORTB,7
MOVLW 0X08 ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA CUARTA
MOVWF BOBINA
CALL RETARDO
DECFSZ PASOS,1 ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
; DESEADA.
GOTO GIRE_I_3 ; EL MOTOR NO HA LLEGADO.
GOTO PRINCIPAL ; EL MOTOR YA LLEGÓ.

```

GIRE_I_3

```

BCF PORTB,4
BCF PORTB,5
BSF PORTB,6
BCF PORTB,7
MOVLW 0X04 ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA TERCERA
MOVWF BOBINA
CALL RETARDO
DECFSZ PASOS,1 ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN

```

; DESEADA.

```

GOTO GIRE_I_2 ; EL MOTOR NO HA LLEGADO.
GOTO PRINCIPAL ; EL MOTOR YA LLEGÓ.

```

GIRE_I_2

```

BCF PORTB,4
BSF PORTB,5
BCF PORTB,6
BCF PORTB,7
MOVLW 0X02 ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA SEGUNDA
MOVWF BOBINA
CALL RETARDO
DECFSZ PASOS,1 ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
; DESEADA.

```

```

GOTO GIRE_I_1 ; EL MOTOR NO HA LLEGADO.
GOTO PRINCIPAL ; EL MOTOR YA LLEGÓ.

GIRE_I_1

BSF PORTB,4
BCF PORTB,5
BCF PORTB,6
BCF PORTB,7

MOVLW 0X01 ; LA ÚLTIMA BOBINA ENERGIZADA FUE LA PRIMERA
MOVWF BOBINA
CALL RETARDO
DECFSZ PASOS,1 ; SE PREGUNTA SI EL MOTOR LLEGÓ A LA POSICIÓN
; DESEADA.

GOTO GIRE_I_4 ; EL MOTOR NO HA LLEGADO.
GOTO PRINCIPAL ; EL MOTOR YA LLEGÓ.

;;
LED

BSF PORTA,0
CALL RETARDO
CALL RETARDO
BCF PORTA,0
CALL RETARDO
CALL RETARDO
RETURN

;
RETARDO

MOVLW 0XA0
MOVWF CON1

DEC4

MOVLW 0X85
MOVWF CON2

DEC3

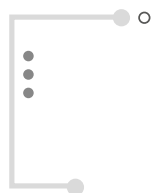
DECFSZ CON2,1
GOTO DEC3
DECFSZ CON1,1
GOTO DEC4
RETURN

;;
END

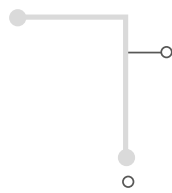
```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard, ya que es preciso conocer la distribución del microcontrolador (ver figura 30) y el orden de las bobinas del motor.

Paso 6. Hacer las correcciones pertinentes y volver al paso tres.



Ejercicios propuestos



1. Diseñar un programa que controle un motor paso a paso de 1.8 grados. Cuando se oprime un pulsador el motor gira 270° a la derecha, luego da una vuelta a la izquierda, luego dos vueltas a la derecha y, por último, gira 45° a la izquierda. En este momento el motor para. Hay un led de testigo. Cuando se vuelve a oprimir el pulsador, éste inicia nuevamente.
2. En el ejemplo 20 se trabajó con un motor paso a paso de cuatro bobinas, de 1.8 grados por paso, de 5Vdc y 4 pulsadores ubicados en cruz (figura 46). Se desea que una LCD visualice en la línea 1 los grados que gira el motor y en la línea 2 el sentido de giro (derecha o izquierda).
3. Se desea diseñar el control de un semáforo en una intersección (calle/ carrera) así: la prioridad la tiene la calle, en la carrera estarán ubicados sensores para detectar el flujo de vehículos. Cuando pasen 10 vehículos en cada uno de los sentidos (norte/sur ó sur/norte) el semáforo debe dar la vía a la carrera, durante 40 segundos. Se deben tener leds indicadores para los 3 colores del semáforo, tanto en la calle como en la carrera.
4. Agregarle un semáforo peatonal al ejercicio anterior. Este debe ser visualizado en una matriz bicolor: cuando el peatón no debe caminar, el semáforo mostrará una silueta de una persona quieta en rojo, cuando el peatón puede caminar la figura se pone de color verde caminado (animación). Asimismo, cuando el tiempo se esté terminado la figura debe caminar más rápido y ponerse de color amarillo (mezcla de rojo y verde) y, cuando el tiempo se acabe, debe pasar a rojo y la silueta de la persona debe estar quieta.



Capítulo V

Otros Microcontroladores

5.1. Microcontroladores de ATMEL

Una vez vistos los microcontroladores de Microchip, ahora se estudiarán los microcontroladores fabricados por Atmel. Este fabricante tiene varios microcontroladores de 8 bits, entre los cuales se pueden mencionar ATmega8A, ATtiny2313, ATmega48PA y el ATmega328P, entre otros. En la Tabla 25 se muestran algunas de las características de cada uno.

Tabla 25.
Comparación entre algunos Microcontroladores de 8 bits. Atmel.

Características	ATmega8A	ATtiny2313	ATmega48PA	ATmega328P
Memoria de Programa (ROM) FLASH	8k x Bytes	2k x Bytes	4k x Bytes	32k x Bytes
Memoria RAM	1k x 8 bits	128 x 8 bits	512 x 8 bits	1k x 8 bits
Pines de I/O	23	18	23	23
Oscilador Interno	Sí	Sí	Sí	Sí
Tipo de Empaque	PDIP/TQFP/QFN/MLF	PDIP/SOIC/MLF	PDIP/TQFP/QFN/MLF	PDIP/TQFP/QFN/MLF
Número de Pines	28/32	20/32	28/32	28/32

Nota. Los datos de Atmega8A se tomó del documento MICROCHIP. Technology Inc. (2017c). (pp. 1-2)

Los datos de Attiny2313 son consultados del documento ATMEL[®]. 8-bit AVR[®]. (2016), (pp.1-2)

Los datos de ATmega48PA son tomados de ATMEL[®]. 8-bit AVR[®]. (2016a). (pp.1-2)

Los datos de ATmega328P son tomados de ATMEL[®]. 8-bit AVR[®].(2016b). (pp. 1-2).

Se selecciona el ATtiny2313, pues es un microcontrolador poderoso. Para desarrollar algunos de los ejemplos que se realizaron en los capítulos III y IV sobre el PIC16F628A. El lector podrá observar que estas aplicaciones pueden servir para los otros microcontroladores que se analizaron en la Tabla 25. Sólo hay que tener en cuenta las pequeñas diferencias entre ellos: el tamaño de la Flash, el tamaño de la RAM, entre otros.

En la Tabla 26 se muestra la comparación entre los microcontroladores PIC-16F628A y el ATtiny2313. Además se mencionan las principales características de cada microcontrolador y, de esa forma, se aprecia que son muy parecidos.

Tabla 26.

Comparación entre los Microcontroladores PIC 16F628A y el ATtiny2313

Microcontroladores	Memoria de Programa (ROM) FLASH	Memoria RAM	Pines de I/O	Oscilador Interno	Tipo de Empaque	Número de Pines
PIC16F628A	2K X 14	128 X 8	15	SÍ	DIP/SSOP/QFN	18/20/32
ATtiny2313	2K X Bytes	128 X 8	18	SÍ	PDIP/SOIC/MLF	20/32

Nota: Los datos de PIC16F628A fueron recuperados MICROCHIP. Technology Inc. (2007a). (p. 1).

Los datos para ATtiny2313 fueron recuperados de ATMEL ®. 8-bit AVR ®. (2016). (pp.1-2)

5.1.1. ATtiny2313

En las Tablas 24 y 25 se comenta que el **ATtiny2313** es de 20/32 pines empaquetado PDIP/SOIC/MLF. En la Figura 49 se muestra la distribución de estos para el PDIP/SOIC.

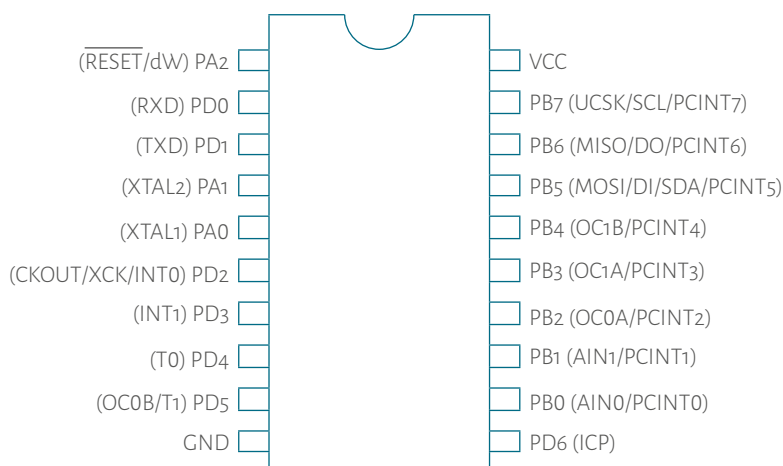


Figura 49. Distribución de pines del ATtiny2313.

Fuente. ATMEL[®]. 8-bit AVR[®]. (2016). Microcontroller with 2K Bytes In-System Programmable Flash. Attiny2313/V. (p. 2)

En la figura 50 se muestra el mapa de memoria del programa, el lector verá que inicia en la posición $0x0000_H$ hasta la posición $0x03FF_H$ que son los 2K que posee el microcontrolador.

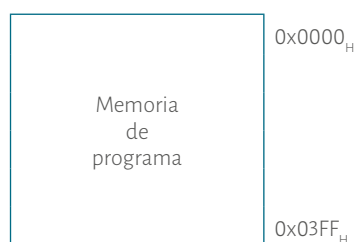


Figura 50. Mapa de memoria de programa del ATtiny2313

Fuente. ATMEL[®]. 8-bit AVR[®]. (2016). Microcontroller with 2K Bytes In-System Programmable Flash. Attiny2313/V. (p.14)

5.1.1.1 Registro STATUS

El registro **STATUS** hace parte del microprocesador interno que posee el microcontrolador. Es preciso recordar que este registro tiene el estado de las operaciones. Por lo tanto, si al hacer una suma hay acarreo o si al hacer una resta la respuesta es 0, entre otros, este es de 8 bits.

A continuación se explica el registro **STATUS** bit a bit.

I	T	H	S	V	N	Z	C
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Fuente. Traducción tomado de ATMEL[®]. 8-bit AVR[®]. (2016), (p. 9)

- Bit 7: **I**: *Bit habilitador de interrupciones*. Este bit activa todas las interrupciones.
 1 → Se habilitan todas las interrupciones.
 0 → Se inhabilitan todas las interrupciones.
- Bit 6: **T**: *Bit Copy Storage*. Este bit puede ser usado con las instrucciones BLD (*Bit Load*) y BST (*Bit Store*).
- Bit 5: **H**: *Carry* en el cuarto bit.
 1: Cuando en una operación aritmética hay *carry* en el bit 4.
 0: Cuando en una operación aritmética no hay *carry* en el bit 4.
- Bit 4: **S**: $S = N \oplus V$
 Este bit es una sale de hacer una XOR entre los bit N y V.
- Bit 3: **V**: Bit de desbordamiento en operaciones de complemento a 2.
- Bit 2: **N**: Indicador de negativo.
 1: Cuando en una operación aritmética o lógica es negativo.
 0: Cuando en una operación aritmética o lógica no es negativo.
- Bit 1: **Z**: Cero (*Zero*).
 1: El resultado de una operación aritmética o lógica es 0.
 0: El resultado de una operación aritmética o lógica no es 0.
- Bit 0: **C**: *Carry*
 1: El resultado de una operación aritmética o lógica pasa de la capacidad de operación. Es decir, supera el *carry* de 8 bits.
 0: El resultado de una operación aritmética o lógica no pasa de la capacidad de operación de 8 bits.

En la Figura 51 se muestra la distribución de la memoria SRAM. El lector puede observar que entre las posiciones 0x0000_H y 0x001F_H hay 32 registros, ver Figura 52. Igualmente, entre las posiciones 0x0020_H y 0x005F_H están los registros de entrada/salida y sus controladores.

32 Registros	0x0000 _H a 0x001F _H
64 I/O Registros	0x0020 _H a 0x005F _H
SRAM (128X8)	0x0060 _H
	0x00DF _H

Figura 51. Memoria SRAM del ATtiny2313/V.

Fuente. ATMEL[®]. 8-bit AVR[®]. (2016). Microcontroller with 2K Bytes In-System Programmable Flash. Attiny2313/V. (p. 15)

En la figura 52 se puede observar en detalle los 32 primeros registros que posee el microcontrolador en la memoria RAM. Es decir que este tipo de microcontroladores tienen 16 registros acumuladores (0x10_H a 0x1F_H), con los cuales se evita el cuello de botella al tener uno o dos de ellos. Estos registros pueden ser usados para propósito general o direccionamiento de memoria.

Registros de propósito general	Registro 0	0x00 _H	
	Registro 1	0x01 _H	
	.	.	
	.	.	
	Registro 14	0x0E _H	
	Registro 15	0x0F _H	
	Registros de trabajo	Registro 16	0x10 _H
		Registro 17	0x11 _H
		.	.
		.	.
		Registro 31	0x1E _H
		Registro 32	0x1F _H

Figura 52. Registros de propósito general y de trabajo del ATtiny2313/V.

Fuente. ATMEL[®]. 8-bit AVR[®]. (2016). Microcontroller with 2K Bytes In-System Programmable Flash. Attiny2313/V. (p.10)

5.1.1.2 Puertos de I/O

El microcontrolador ATtiny2313 tiene 20 pines (Estuche PDIP ó SOIC), de los cuales 18 son líneas de entrada/salida (*Input/Output*). Estas líneas están distribuidas en 3 registros llamados PORTA, PORTB y PORTD. El PORTA tiene 3 líneas de I/O, el PORT B tiene 8 líneas de I/O, y el PORTD tiene 7 líneas de I/O, ver Figura 49. Cada línea tiene dos o más tareas y, a medida que el lector avanza en la programación, puede usar estas opciones.

Para el control de los puertos de entrada o salida tiene los registros de control DDRA, DDRB y DDRD, los cuales controlan respectivamente cada puerto. Para programar una línea de salida se le garantiza uno (1) y para programarla de entrada se le garantiza cero (0). Es importante aclarar que en este procedimiento ocurre lo contrario a lo que ya vimos en los microcontroladores de Microchip. Hay que mencionar que los microcontroladores de Atmel en comparación con los de Microchip en la RAM no tienen bancos.

5.1.1.3. Set de instrucciones

Antes de empezar a desarrollar ejemplos de programas en el microcontrolador, es necesario ver las instrucciones. En el capítulo III se mencionó que este fabricante está en el grupo RISC. En la Tabla 27 se muestran las instrucciones las cuales fueron tomadas del *datasheet* del microcontrolador. Hay que mencionar que la ejecución de las instrucciones en estos microcontroladores es más rápida, pues mientras la primera instrucción está siendo ejecutada, la segunda es recuperada de la memoria del programa (*Fetching*) (ATMEL, 2016, pp.1-2).

Tabla 27.
Set instrucciones del ATtiny2313.

Nemonico	Descripción	Operación	Bandera (Status)	No. Ciclos de reloj
Instrucciones Lógicas y Aritméticas				
ADD Rd,Rr	Suma dos registros	$Rd \leftarrow Rd + Rr$	Z,C,N,V,H	1
ADC Rd,Rr	Suma del <i>Carry</i> con dos registros	$Rd \leftarrow Rd + Rr + C$	Z,C,N,V,H	1
ADIW Rdl,K	Suma inmediata de un registro con constante (K)	$Rdh:Rdl \leftarrow Rdh:Rdl + K$	Z,C,N,V,S	2
SUB Rd,Rr	Resta dos registros	$Rd \leftarrow Rd - Rr$	Z,C,N,V,H	1

SUBI Rd,K	Resta una constante de un registro	$Rd \leftarrow Rd - K$	Z,C,N,V,H	1
SBC Rd,Rr	Resta con <i>carry</i> dos registros	$Rd \leftarrow Rd - Rr - C$	Z,C,N,V,H	1
SBCI Rd,K	Resta una constante y <i>Carry</i> de un registro	$Rd \leftarrow Rd - K - C$	Z,C,N,V,H	1
SBIW RdI,K	Resta inmediata de un registro con constante (K)	$Rdh:Rdl \leftarrow Rdh:Rdl - K$	Z,C,N,V,S	2
AND Rd,Rr	Operación lógica AND entre dos registros	$Rd \leftarrow Rd \text{ AND } Rr$	Z,N,V	1
ANDI Rd,Rr	Operación lógica AND entre un registro y una constante (K)	$Rd \leftarrow Rd \text{ AND } K$	Z,N,V	1
OR Rd,Rr	Operación lógica OR entre dos registros	$Rd \leftarrow Rd \text{ OR } Rr$	Z,N,V	1
ORI Rd,Rr	Operación lógica AND entre un registro y una constante (K)	$Rd \leftarrow Rd \text{ OR } K$	Z,N,V	1
EOR Rd,Rr	Operación lógica XOR entre dos registros	$Rd \leftarrow Rd \oplus Rr$	Z,N,V	1
COM Rd	Complemento a 1 de un registro	$Rd \leftarrow 0xFF_H - Rd$	Z,C,N,V	1
NEG Rd	Complemento a 2 de un registro	$Rd \leftarrow 0x00_H - Rd$	Z,C,N,V,H	1
SBR Rd,K	Pone unos (Setea) en todos los bits de un registro (Rd)	$Rd \leftarrow Rd \text{ OR } K$	Z,N,V	1
CBR Rd,K	Pone ceros (Clarea) en todos los bits de un registro (Rd)	$Rd \leftarrow Rd \text{ AND } (0xFF_H - K)$	Z,N,V	1
INC Rd	Incrementa en 1 un registro	$Rd \leftarrow Rd + 1$	Z,N,V	1
DEC Rd	Decrementa en 1 un registro	$Rd \leftarrow Rd - 1$	Z,N,V	1
TST Rd	Prueba si un registro es cero	$Rd \leftarrow Rd \text{ AND } Rd$	Z,N,V	1
CLR Rd	Clarea un registro	$Rd \leftarrow Rd \oplus Rd$	Z,N,V	1
SER Rd	Setea un registro	$Rd \leftarrow 0xFF_H$	Ninguno	1
Instrucciones de Saltos				
RJMP K	Salta a la etiqueta K	$PC \leftarrow PC + k + 1$	Ninguno	2
IMP K	Salto indirecto a (Z)	$PC \leftarrow Z$	Ninguno	2
RCALL K	Llamado de una subrutina K	$PC \leftarrow PC + k + 1$	Ninguno	3
ICALL K	Llamado indirecto a (Z)	$PC \leftarrow Z$	Ninguno	3
RET	Retorna de una subrutina	$PC \leftarrow \text{STACK}$	Ninguno	4
RETI	Retorna de una interrupción	$PC \leftarrow \text{STACK}$	I	4
CPSE Rd,Rr	Compara los registros (Rd = Rr) y salta si son iguales	Si (Rd = Rr) $PC \leftarrow PC + 2$ or 3	Ninguno	1/2/3
CP Rd,Rr	Compara los registros Rd y Rr	$Rd - Rr$	Z,N,V,C,H	1
CPC Rd,Rr	Compara los registros Rd, Rr y <i>Carry</i>	$Rd - Rr - C$	Z,N,V,C,H	1
CP K	Compara el registro Rd con la constante K	$Rd - K$	Z,N,V,C,H	1
SBRC Rr,b	Pregunta por un bit (b) del registro Rr y salta si es cero (Clear)	Si (Rr(b)=0) $PC \leftarrow PC + 2$ or 3	Ninguno	1/2/3
SBRS Rr,b	Pregunta por un bit (b) del registro Rr y salta si es uno (Set)	Si (Rr(b)=1) $PC \leftarrow PC + 2$ or 3	Ninguno	1/2/3

BRBS s,K	Salta a una etiqueta (K) si el bit (s) del registro STATUS es uno	Si (SREG(s) = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRBS s,K	Salta a una etiqueta (K) si el bit (s) del registro STATUS es uno	Si (SREG(s) = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BREQ K	Salta a una etiqueta (K) si el bit (Z) del STATUS es uno	Si (Z = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRNE K	Salta a una etiqueta (K) si el bit (Z) del registro STATUS es cero	Si (Z = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRCS K	Salta a una etiqueta (K) si el bit (C) del registro STATUS es uno	Si (C = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRCC K	Salta a una etiqueta (K) si el bit (C) del registro STATUS es cero	Si (C = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRSH K	Salta a una etiqueta (K) si el bit (C) del registro STATUS es cero	Si (C = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRLO K	Salta a una etiqueta (K) si el bit (C) del registro STATUS es uno	Si (C = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRMI K	Salta a una etiqueta (K) si el bit (N) del registro STATUS es uno	Si (N = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRPL K	Salta a una etiqueta (K) si el bit (N) del registro STATUS es cero	Si (N = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRGE K	Salta a una etiqueta (K) si el bit (S) del registro STATUS es cero	Si ($N \oplus V = 0$) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRLT K	Salta a una etiqueta (K) si el bit (S) del registro STATUS es uno	Si ($N \oplus V = 1$) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRHS K	Salta a una etiqueta (K) si el bit (H) del registro STATUS es uno	Si (N = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRHC K	Salta a una etiqueta (K) si el bit (H) del registro STATUS es cero	Si (N = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRTS K	Salta a una etiqueta (K) si el bit (T) del registro STATUS es uno	Si (T = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRTC K	Salta a una etiqueta (K) si el bit (T) del registro STATUS es cero	Si (T = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRVS K	Salta a una etiqueta (K) si el bit (V) del registro STATUS es uno	Si (V = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRVC K	Salta a una etiqueta (K) si el bit (V) del registro STATUS es cero	Si (V = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRIE K	Salta a una etiqueta (K) si las interrupciones están habilitadas	Si (I = 1) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
BRID K	Salta a una etiqueta (K) si las interrupciones están deshabilitadas	Si (I = 0) Entonces $PC \leftarrow PC + k + 1$	Ninguno	1/2
Instrucciones de Pruebas Bits				
SBI P,b	Pone un Bit (b) del puerto (P) en uno	$I/O(Pb) \leftarrow 1$	Ninguno	2
CBI P,b	Pone un Bit (b) del puerto (P) en cero	$I/O(Pb) \leftarrow 0$	Ninguno	2
LSL Rd	Rota a la izquierda el registro (Rd)	$Rd(n+1) \leftarrow Rd(n)$, $Rd(0) \leftarrow 0$	Z,C,N,V	1
LSR Rd	Rota a la derecha el registro (Rd)	$Rd(n) \leftarrow Rd(n+1)$, $Rd(7) \leftarrow 0$	Z,C,N,V	1

ROL Rd	Rota a la izquierda el registro (Rd) con <i>Carry</i>	$Rd(0) \leftarrow C, Rd(n+1) \leftarrow Rd(n), C \leftarrow Rd(7)$	Z,C,N,V	1
ROR Rd	Rota a la derecha el registro (Rd) con <i>Carry</i>	$Rd(7) \leftarrow C, Rd(n) \leftarrow Rd(n+1), C \leftarrow Rd(0)$	Z,C,N,V	1
ASR Rd	Rotacion aritmética a la derecha el registro (Rd)	$Rd(n) \leftarrow Rd(n+1), n=0..6$	Z,C,N,V	1
SWAP Rd	Intercambia los <i>nibbles</i> del registro (Rd)	$Rd(3..0) \leftarrow Rd(7..4), Rd(7..4) \leftarrow Rd(3..0)$	Ninguno	1
BSET s	Pone un Bit (s) del registro STATUS en uno	$SREG(s) \leftarrow 1$	SREG(s)	1
BCLR s	Pone un Bit (s) del registro STATUS en cero	$SREG(s) \leftarrow 0$	SREG(s)	1
BST Rd,b	Toma el Bit (b) del registro (Rd) y lo pone en el Bit (T) del registro STATUS	$T \leftarrow Rr(b)$	T	1
BLD Rd,b	Toma el Bit (T) del registro STATUS y lo pone en el Bit (b) del registro (Rd)	$Rd(b) \leftarrow T$	Ninguno	1
SEC	Pone el Bit (C) del registro STATUS en uno	$C \leftarrow 1$	C	1
CLC	Pone el Bit (C) del registro STATUS en cero	$C \leftarrow 0$	C	1
SEN	Pone el Bit (N) del registro STATUS en uno	$N \leftarrow 1$	N	1
CLN	Pone el Bit (N) del registro STATUS en cero	$N \leftarrow 0$	N	1
SEZ	Pone el Bit (Z) del registro STATUS en uno	$Z \leftarrow 1$	Z	1
CLZ	Pone el Bit (Z) del registro STATUS en cero	$Z \leftarrow 0$	Z	1
SEI	Habilita todas las interrupciones	$I \leftarrow 1$	I	1
CLI	Deshabilita todas las interrupciones	$I \leftarrow 0$	I	1
SES	Pone el Bit (S) del registro STATUS en uno	$S \leftarrow 1$	S	1
CLS	Pone el Bit (S) del registro STATUS en cero	$S \leftarrow 0$	S	1
SEV	Pone el Bit (V) del registro STATUS en uno	$V \leftarrow 1$	V	1
CLV	Pone el Bit (V) del registro STATUS en cero	$V \leftarrow 0$	V	1
SET	Pone el Bit (T) del registro STATUS en uno	$T \leftarrow 1$	T	1
CLT	Pone el Bit (T) del registro STATUS en cero	$T \leftarrow 0$	T	1
SEH	Pone el Bit (H) del registro STATUS en uno	$H \leftarrow 1$	H	1
CLH	Pone el Bit (H) del registro STATUS en cero	$H \leftarrow 0$	H	1

Instrucciones de Transferencia de datos				
MOV Rd,Rr	Mueve el registro (Rr) al registro (Rd)	$Rd \leftarrow Rr$	Ninguno	1
MOVW Rd,Rr	Mueve el registro (Word) al registro (Rd)	$Rd+1:Rd \leftarrow Rr+1:Rr$	Ninguno	1
LDI Rd,K	Carga la constante (K) al registro (Rd)	$Rd \leftarrow K$	Ninguno	1
LD Rd,X	Carga indirecta de una posición de memoria (X)	$Rd \leftarrow (X)$	Ninguno	2
LD Rd,X+	Carga indirecta de una posición +1 de memoria (X)	$Rd \leftarrow (X), X \leftarrow X + 1$	Ninguno	2
LD Rd,-X	Carga indirecta de una posición -1 de memoria (X)	$X \leftarrow X - 1, Rd \leftarrow (X)$	Ninguno	2
LD Rd,Y	Carga indirecta de una posición de memoria (Y)	$Rd \leftarrow (Y)$	Ninguno	2
LD Rd,Y+	Carga indirecta de una posición +1 de memoria (Y)	$Rd \leftarrow (Y), Y \leftarrow Y + 1$	Ninguno	2
LD Rd,-Y	Carga indirecta de una posición -1 de memoria (Y)	$Y \leftarrow Y - 1, Rd \leftarrow (Y)$	Ninguno	2
LDD Rd,Y+q	Carga indirecta de una posición de memoria (Y) con desplazamiento	$Rd \leftarrow (Y + q)$	Ninguno	2
LD Rd,Z	Carga indirecta de una posición de memoria (Z)	$Rd \leftarrow (Z)$	Ninguno	2
LD Rd,Z+	Carga indirecta de una posición +1 de memoria (Z)	$Rd \leftarrow (Z), Z \leftarrow Z + 1$	Ninguno	2
LD Rd,-Z	Carga indirecta de una posición -1 de memoria (Z)	$Z \leftarrow Z - 1, Rd \leftarrow (Z)$	Ninguno	2
LDD Rd,Z+q	Carga indirecta de una posición de memoria (Z) con desplazamiento	$Rd \leftarrow (Z + q)$	Ninguno	2
LDS Rd,K	Carga directa de una constante (K) a un registro (Rd) de la SRAM	$Rd \leftarrow (K)$	Ninguno	2
ST X,Rr	Mover indirecto el registro (Rr) a la posición (X)	$(X) \leftarrow Rr$	Ninguno	2
ST X+,Rr	Mover indirecto el registro (Rr) a la posición (X+)	$(X) \leftarrow Rr, X \leftarrow X + 1$	Ninguno	2
ST -X,Rr	Mover indirecto el registro (Rr) a la posición (X-)	$X \leftarrow X - 1, (X) \leftarrow Rr$	Ninguno	2
ST Y,Rr	Mover indirecto el registro (Rr) a la posición (Y)	$(Y) \leftarrow Rr$	Ninguno	2
ST Y+,Rr	Mover indirecto el registro (Rr) a la posición (Y+)	$(Y) \leftarrow Rr, Y \leftarrow Y + 1$	Ninguno	2
ST -Y,Rr	Mover indirecto el registro (Rr) a la posición (Y-)	$Y \leftarrow Y - 1, (Y) \leftarrow Rr$	Ninguno	2
STD Y+q,Rr	Mover indirecto el registro (Rr) a la posición (Y+) con desplazamiento	$(Y + q) \leftarrow Rr$	Ninguno	2
ST Z,Rr	Mover indirecto el registro (Rr) a la posición (Z)	$(Z) \leftarrow Rr$	Ninguno	2
ST Z+,Rr	Mover indirecto el registro (Rr) a la posición (Z+)	$(Z) \leftarrow Rr, Y \leftarrow Y + 1$	Ninguno	2

ST -Z,Rr	Mover indirecto el registro (Rr) a la posición (Z-)	$Z \leftarrow Z-1, (Z) \leftarrow Rr$	Ninguno	2
STD Z+q,Rr	Mover indirecto el registro (Rr) a la posición (Z+) con desplazamiento	$(Z+q) \leftarrow Rr$	Ninguno	2
STS K,Rr	Mover directo el registro (Rr) a la SRAM	$(K) \leftarrow Rr$	Ninguno	2
LPM	Carga la posición (Z) al registro (R0)	$R0 \leftarrow (Z)$	Ninguno	3
LPM Rd,Z	Carga la posición (Z) al registro (Rd)	$Rd \leftarrow (Z)$	Ninguno	3
LPM Rd,Z+	Carga la posición (Z+1) al registro (Rd)	$Rd \leftarrow (Z), Z \leftarrow Z+1$	Ninguno	3
SPM	Mover indirecto los registros (R0 y R1) a la posición (Z)	$(Z) \leftarrow R1:R0$	Ninguno	-
IN Rd,P	Ingresa el puerto (P) al registro (Rd)	$Rd \leftarrow P$	Ninguno	1
OUT P,Rd	Saca el registro (Rd) al puerto (P)	$P \leftarrow Rd$	Ninguno	1
PUSH Rr	Pasa el registro Rr a la PILA (STACK)	$STACK \leftarrow Rr$	Ninguno	2
POP Rd	Pasa la PILA (STACK) al registro (Rd)	$Rr \leftarrow STACK$	Ninguno	2
Instrucciones de control de MCU				
NOP	No hace nada		Ninguno	1
SLEEP	El microcontrolador va a bajo consumo. (En espera)	(Ver dicha función en el Datasheet)	Ninguno	1
WDR	Clarea el WDT (<i>Watchdog Timer</i>)	(Ver dicha función en el Datasheet)	Ninguno	1
BREAK	Break	For On-chip Debug Only	Ninguno	N/A

Fuente. Elaboración propia, datos tomados de ATMEL[®]. 8-bit AVR[®]. (2016). Microcontroller with 2K Bytes In-System Programmable Flash. Attiny2313/V. Para los datos de SLEEP (p. 30) y WDR (p. 38).

5.1.2. Programación

A continuación se acondicionan algunos de los ejemplos que se desarrollaron en los microcontroladores de Microchip vistos en los capítulos III y IV para que el lector pueda desarrollar destreza en ambos fabricantes, siguiendo los mismos pasos de programación, planteados en el ejemplo 8 del capítulo III. Se recomienda al lector leer nuevamente este ejemplo. Para programar y simular los programas de los microcontroladores de Atmel se usará AtmelStudio versión 7.0.

Ejemplo 21. Tomar el ejemplo 8 y acondicionarlo para el microcontrolador ATtiny2313.

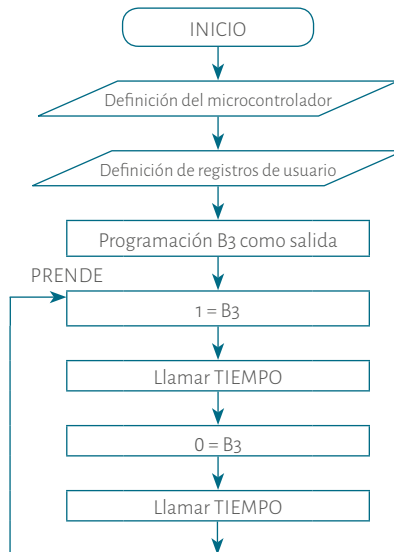
A continuación se desarrolla paso a paso el proceso. Además se explica cómo hacer un proyecto en AtmelStudio.

Paso 1. El funcionamiento de un led ya se conoce. Por ende, este paso se omite.

Paso 2. Hacer un diagrama de conexiones. Hay que decir en cuál de las 18 líneas de *In/Out* del microcontrolador se va a poner el led. Se puede tomar B₃, así como en el ejemplo 8. En los ejemplos desarrollados con microcontroladores de Atmel no se harán figuras con los montajes. El lector entenderá que sólo debe seguir la distribución de pines del microcontrolador con el que está trabajando, en este caso es el ATtiny2313, ver Figura 49. Igualmente, es preciso tener en cuenta las líneas de alimentación y de *reset*.

Paso 3. Hacer el diagrama de flujo. De acuerdo con el diagrama de conexiones planteado en el punto anterior hay que programar B₃ como salida para el led, ver Diagrama 34. La subrutina tiempo es la misma de los anteriores programas. Por ende, no se hace el diagrama de flujo, ver diagrama 3, ejemplo 8.

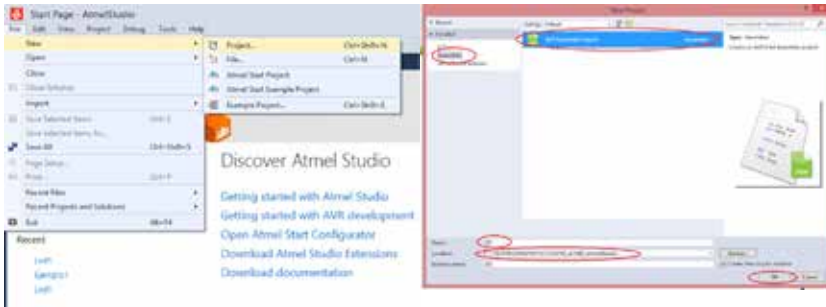
Diagrama 34.
Programa que prende y apaga un led. Ejemplo 21



Fuente. Elaboración propia.

Paso 4. Una vez hecho el diagrama de flujo hay que pasarlo a instrucciones. Para ello, se usa AtmelStudio 7.0 u otra versión superior. Una vez abierto el

programa se debe hacer clic en *File/New/Project...* como se observa en la Figura 53A. En la siguiente ventana seleccionar *Assembler* después *AVR Assembler Project*, darle nombre al proyecto (para este caso LED) y seleccionar la carpeta donde va a guardarse. Para seleccionar esta carpeta se debe ir a *Browse* y después dar clic en OK, ver Figura 53B. Se recomienda al lector tener una carpeta para sus proyectos.



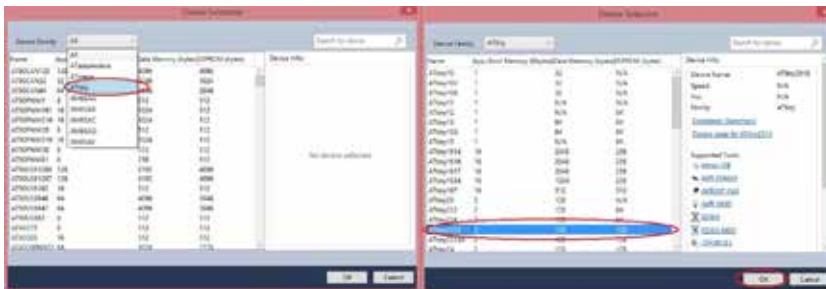
53 A

53 B

Figura 53. Crear nuevo proyecto en AtmelStudio 7.0 Paso 1

Fuente. Captura de pantalla AtmelStudio 7.0 Paso 1.

En la ventana siguiente desplegar en *Device Family* desplegar la lista y seleccionar *Attiny*, ver Figura 54A. Luego se debe buscar el microcontrolador, para este caso ATtiny2313, y hacer clic en OK, ver Figura 54B.



54 A

54 B

Figura 54. Crear nuevo proyecto en AtmelStudio 7.0 Paso 2

Fuente. Captura de pantalla AtmelStudio 7.0 Paso 2.

En la ventana que se abre es donde el programador debe digitar el código en *Assembler*. Antes de empezar a digitar, se selecciona el texto que sale por defecto y se borra. Si el programador desea hacer comentarios, los puede escribir después de digitar un punto y coma.

En el diagrama de flujo, lo primero es seleccionar la librería del microcontrolador (**.INCLUDE "tn2313def.inc"**). Se había mencionado que este tipo de microcontroladores tienen 16 registros acumuladores (0x10_H a 0x1F_H). Por lo tanto, para seleccionar uno de ellos como registro de trabajo se hace **.DEF REGA = R16** (ver Figura 55A). Siguiendo el diagrama de flujo, se definen los registros del usuario CON1 y CON2 que se utilizan para la subrutina tiempo. Se recomienda utilizar la instrucción **RJMP INICIO** para ubicar el microcontrolador en inicio. El programa se muestra a continuación:

```
.INCLUDE "tn2313def.inc"           ; Definición del microcontrolador
.DEF REGA = R16                    ; Definición del registro de trabajo

.DEF CON1 = R17                    ; Definición de registros del usuario
.DEF CON2 = R18

RJMP INICIO                        ; Salta a inicio del programa

INICIO:

    LDI    REGA,0B11111111         ; Se programa todo el PORTB
    OUT    DDRB,REGA               ; como salida

    LDI    REGA,0B00000000         ; Se garantiza que el PORTB
    OUT    PORTB,REGA             ; este en cero

PRENDE:

    SBI    PORTB,3                 ; Prende el B3 del PORTB
    RCALL TIEMPO                   ; Se da tiempo para ver el led prendido
    CBI    PORTB,3                 ; Apaga el B3 del PORTB
    RCALL TIEMPO                   ; Se da tiempo para ver el led apagado
    RJMP  PRENDE                   ; Regresa a prender el led

TIEMPO:

    LDI    CON1,0x3F               ; Carga 3Fh en CON1

N1:

    LDI    CON2,0x2A               ; Carga 2Ah en CON2
```

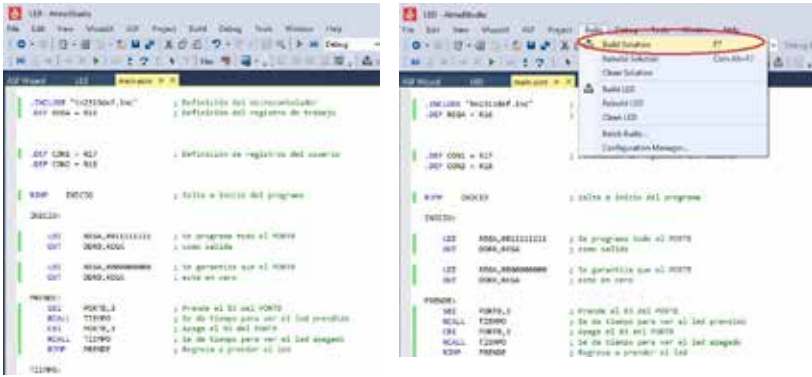
Nz:

```

DEC     CON2           ; Decrementa CON2-1=CON2 y pregunta si es cero
BRNE   Nz             ; Aún CON2 no es cero

DEC     CON1           ; Decrementa CON1-1=CON1 y pregunta si es cero
BRNE   N1             ; Aún CON1 no es cero
RET                                ; Retorna de la subrutina
    
```

Para compilar se va por **Buil/Buil Solution**, como se observa en la Figura 55B con **F7**. Si se presentan errores se pueden corregir haciendo clic en el error y el aplicativo remite a la línea donde se presenta dicho error. Previamente se debe identificar qué tipo de error es para poder corregirlo. Una vez corregido, se debe pulsar nuevamente **F7** y repetir el procedimiento hasta que no tenga ningún error.



55 A

55 B

Figura 55. Crear nuevo proyecto en AtmelStudio 7.0 Paso 3

Fuente. Captura de pantalla AtmelStudio 7.0 Paso 3

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador, ver Figura 49, y del diagrama de conexiones planteado en el paso 2.

Paso 6. Hacer las correcciones pertinentes y volver al paso 3.

Ejemplo 22. Tomar el ejemplo 10 y acondicionarlo para el microcontrolador ATtiny2313. A continuación se desarrolla paso a paso:

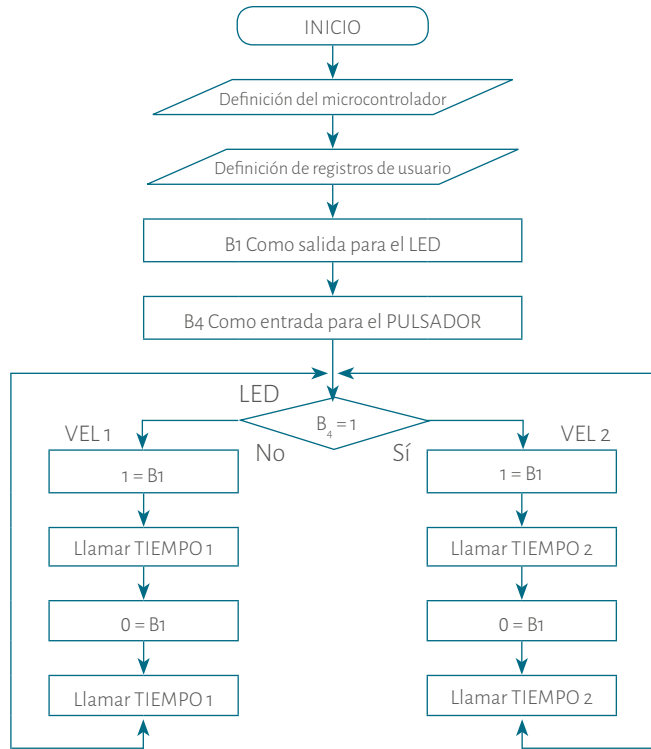
Paso 1. Ya se sabe cómo funciona un led y un pulsador normalmente abierto.

Paso 2. Hay que identificar en cuál de las 18 líneas del microcontrolador se va a ubicar el led y en cuál el pulsador. Por ejemplo, se puede tomar B1 para el led y B4 para el pulsador. Hay que tener en cuenta la distribución de pines del microcontrolador.

Paso 3. Diagrama de flujo: según lo planteado en el punto anterior hay que programar B1 como salida para el led y B4 como entrada para el pulsador, ver diagrama 35.

Diagrama 35.

Programa que prende y apaga un led con dos velocidades. Ejemplo 22.



Fuente. Elaboración propia.

Como se mencionó en el ejemplo 10 hay 2 subrutinas de retardo las que se define la diferencia de velocidades: TIEMPO 1 y TIEMPO 2.

Paso 4. Note los diferentes valores en los contadores para las subrutinas de tiempo. El programa es el siguiente:

```

.INCLUDE "tn2313def.inc"           ; Definición del microcontrolador
.DEF REGA = R16                    ; Definición del registro de trabajo

; Definición de registros del usuario
.DEF CON1 = R17
.DEF CON2 = R18
.DEF CON3 = R19

RJMP INICIO                        ; Salta a inicio del programa

INICIO:

    LDI    REGA,0B11101111        ; Se programa B4 como IN y
    OUT    DDRB,REGA              ; B1 como OUT

    CBI    PORTB,1                ; Se apaga el led

PREGUNTA:
    IN     CON3,PINB              ; Ingresa el PORTB a CON3
    SBRC   CON3,4                 ; Se pregunta si el b4 de CON3 es uno
    RJMP   VEL1                   ; B4 de CON3 es uno
    RJMP   VEL2                   ; B4 de CON3 es cero

VEL1:
    SBI    PORTB,1                ; Prende el B1 del PORTB
    RCALL  TIEMPO1                ; Se da tiempo para ver el led prendido
    CBI    PORTB,1                ; Apaga el B1 del PORTB
    RCALL  TIEMPO1                ; Se da tiempo para ver el led apagado
    RJMP   PREGUNTA              ; Regresa a preguntar

VEL2:
    SBI    PORTB,1                ; Prende el B1 del PORTB
    RCALL  TIEMPO2                ; Se da tiempo para ver el led prendido
    CBI    PORTB,1                ; Apaga el B1 del PORTB
    RCALL  TIEMPO2                ; Se da tiempo para ver el led apagado
    RJMP   PREGUNTA              ; Regresa a preguntar

TIEMPO1:
    LDI    CON1,0x3F              ; Carga 3Fh en CON1

N1:
    LDI    CON2,0x2A              ; Carga 2Ah en CON2

N2:
    DEC    CON2                   ; Decrementa CON2-1=CON2 y pregunta si es cero
    BRNE   N2                     ; Aun CON2 no es cero

    DEC    CON1                   ; Decrementa CON1-1=CON1 y pregunta si es cero

```

	BRNE	N1	; Aún CON1 no es cero
	RET		; Retorna de la subrutina
TIEMPO2:			
	LDI	CON1,0xAF	; Carga AFh en CON1
N3:			
	LDI	CON2,0x6A	; Carga 6Ah en CON2
N4:			
	DEC	CON2	; Decrementa CON2-1=CON2 y pregunta si es cero
	BRNE	N4	; Aún CON2 no es cero
	DEC	CON1	; Decrementa CON1-1=CON1 y pregunta si es cero
	BRNE	N3	; Aún CON1 no es cero
	RET		; Retorna del a subrutina

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 47) y el diagrama de conexiones planteado en el paso 2.

Paso 6. Hacer las correcciones pertinentes y volver al paso tres.

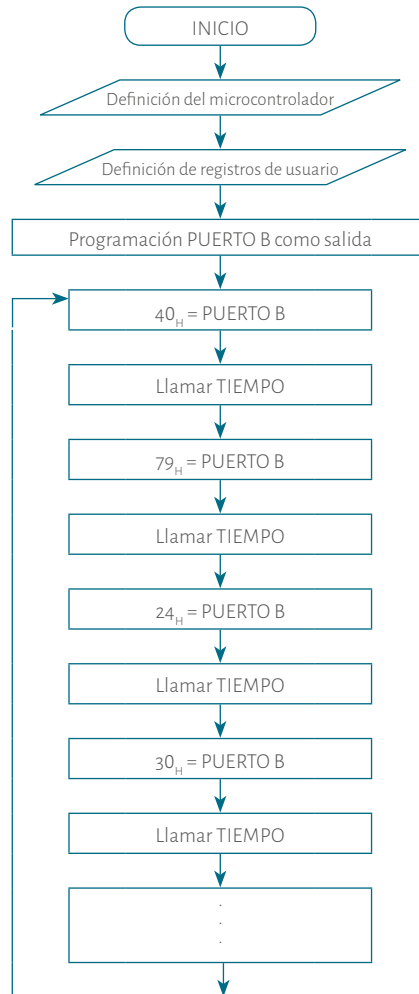
Ejemplo 23. Tomar el ejemplo 11 y acondicionarlo para el microcontrolador ATtiny2313. A continuación se desarrolla este proceso paso a paso:

Paso 1. En el ejemplo 11 se utilizó y se explicó cómo funciona un display de ánodo común, por lo tanto, no se explica.

Paso 2. Para este ejercicio se usará el puerto B, así como se usó en el ejemplo 11. Desde B₀ hasta B₆ se usa una línea del puerto para cada segmento.

Paso 3. Diagrama de flujo: hay que usar el puerto B de salida. El diagrama de flujo 36 muestra cómo usar este puerto y cómo obtener los datos. Es preciso recordar que estos datos son los que se observan en la Tabla 12. Note que el diagrama no está completo, el lector puede completarlo, solo debe seguir la secuencia.

Diagrama 36.
Programa para visualizar los números decimales del 0 al 9 en un Display 7 Segmentos. Ejemplo 23.



Fuente. Elaboración propia.

Paso 4. El programa es el siguiente. La subrutina de tiempo sigue siendo igual a los ejemplos anteriores.

```

.INCLUDE "tn2313def.inc"      ; Definición del microcontrolador
.DEF REGA = R16              ; Definición del registro de trabajo

.DEF CON1 = R17              ; Definición de registros del usuario
.DEF CON2 = R18
  
```



```

RJMP   INICIO           ; Salta a inicio del programa

INICIO:

LDI    REGA,0B11111111    ; Se programa PORTB como OUT
OUT    DDRB,REGA

CERO:

LDI    REGA,0X40         ; Se carga el código del cero al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X79         ; Se carga el código del uno al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X24         ; Se carga el código del dos al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X30         ; Se carga el código del tres al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X19         ; Se carga el código del cuatro al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X12         ; Se carga el código del cinco al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X02         ; Se carga el código del seis al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X78         ; Se carga el código del siete al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X00         ; Se carga el código del ocho al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

LDI    REGA,0X10         ; Se carga el código del nueve al REGA
OUT    PORTB,REGA       ; Se muestra en el display
RCALL  TIEMPO           ; Se da tiempo para ver el número

RJMP   CERO             ; Salta a inicio del programa

```

TIEMPO:

LDI CON1,0x3F ; Carga 3Fh en CON1

N1:

LDI CON2,0x2A ; Carga 2Ah en CON2

N2:

DEC CON2 ; Decrementa CON2-1=CON2 y pregunta si es cero

BRNE N2 ; Aún CON2 no es cero

DEC CON1 ; Decrementa CON1-1=CON1 y pregunta si es cero

BRNE N1 ; Aún CON1 no es cero

RET ; Retorna del a subrutina

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 49) y la distribución del diagrama de conexiones planteado en el paso 2.

Paso 6. Hacer las correcciones y volver al paso 3.

5.2. Microcontroladores de NXP-FREESCALE

Una vez vistos los microcontroladores de Microchip y de Atmel, ahora se estudiarán los fabricados por *NXP-Freescale*. Este fabricante tiene microcontroladores que se pueden dividir por el bus de datos que maneja así como lo menciona Vesga en su libro *Microcontroladores Motorola Freescale* y en familias como lo muestra la Tabla 28.

Tabla 28.
Familias de Microcontroladores de Freescale.

Tamaño del bus datos	Familia	Variedades
Bus de 8 bits	HC08	HC08AB, HC08AP, HC08AS/AZ, HC08BD, HC08EY, HC08G, HC08JK/JL, HC08MR, HC08Q, HC08RF, HC08SR
	HCS08	HCS08AW HCS08G/Q/R
	RS08	MC9RS08KA1/2
Bus de 16 bits	S12	S12XA/D, S12A, S12C, S12D, S12E, S12H, S12G
	HC12	HC912BXX, HC912DXX
	HC16	68HC16R1, 68HC16Y1/Y3, 68HC16Z1/Z3
Bus de 32 bits	68K/ColdFire	ColdFire MCF52XX, 68K M683XX, 68K M68 0X0

Fuente elaboración propia, datos tomados de Vesga, 2007, p. 16-17

5.2.1. Microcontroladores de 8 BITS

Entre los microcontroladores de 8 bits se encuentran el MC9S08JS8, el MC9S08JS16, el MC9S08GB60, el MC9S08GT60, entre otros. En la Tabla 29 se muestran algunas de las características de cada uno de ellos.

Tabla 29.
Comparación entre algunos Microcontroladores de 8 Bits. Freescale.

Características	MC9S08JS8	MC9S08JS16	MC9S-08GB60	MC9S-08GT60
Memoria de Programa (ROM) FLASH	8192 X 8	16384 X 8	60K	60K
Memoria RAM	512 X 8	512 X 8	4K	4K
Pines de I/O	14	14	56	34
Oscilador Interno	Sí	Sí	No	No
Tipo de Empaque	QFN/SOIC	QFN/SOIC	LQFP	QFN/SDIP
Numero de Pines	24/20	24/20	64	44/42

Nota. Los datos de MC9S08JS8 y de MC9S08JS16 fueron tomados de NXP-Freescale. (2009). Los datos de MC9S08GB60 y de MC9S08GT60 son tomados de NXP-Freescale (2004).

Se selecciona el MC9S08JS16, pues es un microcontrolador poderoso. Además con éste se pueden realizar las mismas aplicaciones que se realizaron con los microcontroladores PIC16F628A y ATtiny2313 (Microchip y Atmel). El lector podrá observar que estas aplicaciones pueden servir para los otros mencionados que se analizaron en la tabla 29, sólo hay que tener en cuenta el mapa de memoria de cada uno.

En la Tabla 30 se muestra la comparación entre los microcontroladores PIC16F628A, el ATtiny2313 y el MC9S08JS16. Esta comparación muestran las principales características de cada uno de ellos y, de esa forma, se aprecia que son muy parecidos.

Tabla 30.

Comparación entre los Microcontroladores PIC 16F628A, el ATtiny2313 y el MC9S08JS16

Microcontroladores	Memoria de programa (ROM) FLASH	Memoria RAM	Pines de I/O	Oscilador interno	Tipo de empaque	Número de pines
PIC16F628A	2K X 14	128 X 8	15	Sí	DIP/SSOP/QFN	18/20/32
ATtiny2313	2K X Bytes	128 X 8	18	Sí	PDIP/SOIC/MLF	20/32
MC9S08JS16	16384 X 8	512 X 8	14	Sí	QFN/SOIC	24/20

Notas Los datos de PIC16F628A fueron tomados de NXP-Freescale. (2004), (p.1)

Los datos de ATtiny2313 fueron tomados de NXP-Freescale. (2004), (pp. 1-2)

Los datos de MC9S08JS16 fueron tomados de NXP-Freescale. (2009), (p. 36)

5.2.2. MC9S08JS16

En la Tabla 29 y 30 se explicó que el MC9S08JS16 es de 20 pines empaquetado SOIC. Así, en la Figura 56 se muestra la distribución de estos:

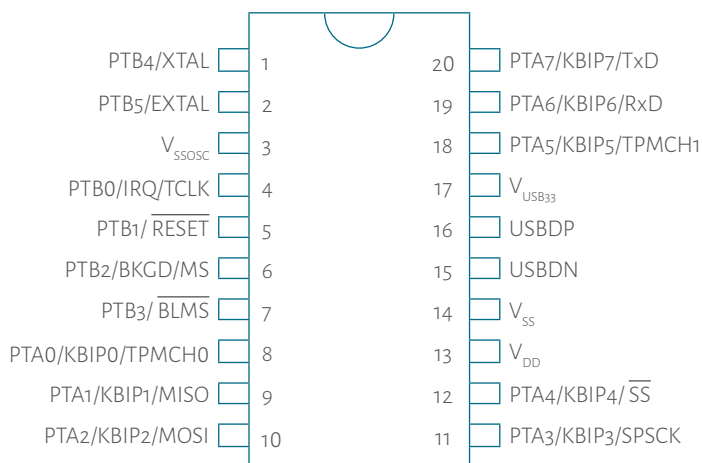


Figura 56. Distribución de pines del MC9S08JS16

Fuente. NXP-Freescale. (2009). p. 22

En la Figura 57 se muestran todas las posiciones de memoria del microcontrolador. El lector podrá interpretar y explorarlas todas las posiciones. Por el momento se quiere avanzar en el desarrollo de aplicaciones en lenguaje *Assembler*. Por consiguiente, se presentan algunos de los registros que posee este microcontrolador:

0X0000 _H	Registros de página directa
0X007F _H	
0X0080 _H	RAM
0X027F _H	512 Bytes
0X0280 _H	No implementado
0X17FF _H	
0X1800 _H	Registros página alta
0X185F _H	
0X1860 _H	RAM
0X195F _H	USB
0X1960 _H	256 Bytes
0X1960 _H	No implementado
0X215F _H	
0X2160 _H	ROM
0X315F _H	de
0X3160 _H	arranque
0X3160 _H	No implementado
0XBFFF _H	
0XC000 _H	FLASH
0XFFFF _H	16384

Figura 57. Distribución de Memoria del MC9S08JS16. NXP-Freescale

Fuente. NXP-Freescale. (2009). HCS08 Microcontrollers.
MC9S08JS16 Reference Manual. p. 36.

Acumulador (A): es un registro de 8 bits, se usa en las operaciones aritméticas y lógicas. Comúnmente se usa para almacenar resultados de cálculos aritméticos y manipulación de datos (NXP-Freescale, 2009, p. 88.). Este registro es como el **registro W** para los microcontroladores de Microchip.

Registro X: es un registro de 8 bits que permite hacer el direccionamiento indexado o puede usarse como un acumulador auxiliar.

Registro H:X: Consta de un par de registros H y X que forman 16 bits. Estos registros se usan para direccionamiento indexado y también como punteros de memoria para cubrir todo el mapa de memoria de la Figura 57.

Registro CCR (Condition Code Register): este registro contiene las banderas, es de 8 bits y es muy similar el registro **STATUS** en los microcontroladores de Microchip y de Atmel. Su funcionamiento es así:

V	1	1	H	I	N	Z	C
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

BIT 7: **V: (OVERFLOW)** BIT DE REBOSAMIENTO DE SIGNO. ES DECIR CUANDO SE SALE DEL RANGO -128 A 127 ASÍ:

1 → REBOSAMIENTO.

0 → NO REBOSAMIENTO.

BIT 4: **H: (HALF-CARRY)** BANDERA DE MEDIO CARRY:

1: HAY CARRY EN EL BIT 4. EN UNA SUMA CON O SIN CARRY.

0: NO HAY CARRY EN EL BIT 4. EN UNA SUMA CON O SIN CARRY.

BIT 3: **I: (INTERRUPT MASK)** MÁSCARA DE INTERRUPCIÓN. PRENDER — CONTAR (*POWER — Down*)

1 → LAS INTERRUPTIONES ESTÁN DESHABILITADAS.

0 → LAS INTERRUPTIONES ESTÁN HABILITADAS.

BIT 2: **N: (NEGATIVE)** BANDERA DE NEGATIVO

1: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA NEGATIVO.

0: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA NO ES NEGATIVO.

BIT 1: **Z: (ZERO)** BANDERA DEL CERO

1: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA O LÓGICA DA CERO

0: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA O LÓGICA NO DA CERO.

BIT 0: **C: (CARRY)** CARRY.

1: EL RESULTADO DE UNA OPERACIÓN SIN SIGNO PASA DE LA CAPACIDAD DE OPERACIÓN, ES DECIR EL CARRY DE 8 BITS.

0: EL RESULTADO DE UNA OPERACIÓN SIN SIGNO PASA DE LA CAPACIDAD DE OPERACIÓN, ES DECIR DE 8 BITS.

(LAS INSTRUCCIONES DE CORRIMIENTO Y ROTACIÓN TAMBIÉN AFECTAN ESTE BIT)

5.2.3. PINES DE I/O

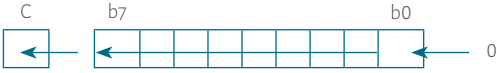
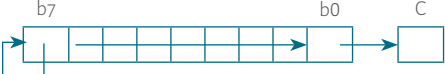
El microcontrolador MC9S08JS16 cuenta con 14 líneas de entrada y salida (I/O) distribuidas en dos puertos (PTAD y PTBD). El *puerto A* tiene 8 bits (A7 a A0), el *puerto B* tiene 6 bits (B5 a B0). Para usarlos como entrada o salida tienen los registros de control PTADD y PTBDD, los cuales controlan cada puerto. Hay que mencionar que los microcontroladores de *NXP-Freescale* en comparación de los de *Microchip* no tienen bancos en la RAM. Por consiguiente, los puertos (PTXD) y los controladores (PTXDD) están en el mismo banco. Para programar una línea de salida se le garantiza uno (1) y para programarla de entrada se le garantiza cero (0). Es importante aclarar que sucede lo contrario con los microcontroladores de *Microchip* y lo mismo de *Atmel*.

5.2.4. Set de instrucciones

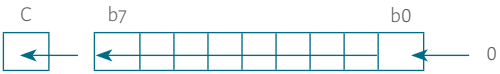
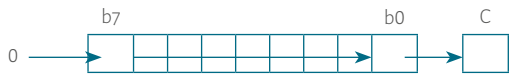
Antes de empezar a desarrollar aplicaciones en el microcontrolador es necesario ver las instrucciones. En el capítulo III se mencionó que este fabricante pertenece al grupo *CISC* y, por lo mismo, maneja muchas instrucciones. El lector notará que, efectivamente, es muy complicado explicar en detalle todas las instrucciones. Razón por la cual a medida que se hagan los ejemplos se explicarán las instrucciones que se van usando. En la Tabla 31 se muestran las instrucciones explicadas por Vega. Es importante recordar que en los microcontroladores de *Microchip* existe el registro **W**, para los de *Freescale* es el **A** (Acumulador).


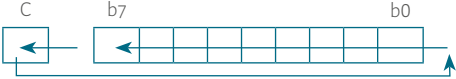
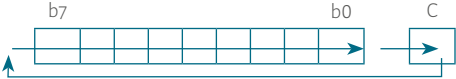
Tabla 31.
Set instrucciones del MC9S08JS16.

INSTRUCCIÓN	OPERACIÓN	CICLOS
ADC #OPR	Suma con Carry	2
ADC OPR		3
ADC OPR,X		3
ADC ,X		2
ADC OPR,SP		4
ADD #OPR	Suma sin Carry	2
ADD OPR		3
ADD OPR,X		3
ADD ,X		2
ADD OPR,SP		4
AIS #OPR	SP = SP + OPR	2
AIX #OPR	H:X = H:X + OPR	2

AND #OPR	Operación lógica AND $A = A \& (M)$	2
AND OPR		3
AND OPR,X		3
AND ,X		2
AND OPR,SP		4
ASL OPR	Rota a la Izquierda el registro así: 	4
ASLA		1
ASLX		1
ASL OPR,X		4
ASL ,X		3
ASL OPR,SP	5	
ASL OPR	Rota a la izquierda el registro así: 	4
ASLA		1
ASLX		1
ASL OPR,X		4
ASL ,X		3
ASL OPR,SP	5	
BCC REL	Salta a una subrutina o etiqueta (REL) si el Bit del Carry del registro de banderas (CCR) es cero	3
BCLR N,OPR	Toma un bit (N) de un registro (OPR) y lo pone en cero (CLR)	4
BCC REL	Salta a una subrutina o etiqueta (REL) si el Bit del Carry del Registro de Banderas (CCR) es Uno	3
BEQ REL	Salta a una subrutina o etiqueta (REL) si el Bit del Cero del Registro de Banderas (CCR) es Uno	3
BHCC REL	Salta a una subrutina o etiqueta (REL) si el Bit de Half Carry del Registro de Banderas (CCR) es Cero	3
BHCS REL	Salta a una subrutina o etiqueta (REL) si el Bit de Half Carry del Registro de Banderas (CCR) es Uno	3
BIH REL	Salta a una subrutina o etiqueta (REL) si el pin IRQ es Cero	3
BIL REL	Salta a una subrutina o etiqueta (REL) si el pin IRQ es Uno	3
BIT #OPR	Probar un bit cualquiera de un registro	
BIT OPR		
BIT OPR,X		
BIT ,X		
BIT OPR,SP		
BMC REL	Salta a una subrutina o etiqueta (REL) si el Bit de interrupción del registro de banderas (CCR) es cero	3
BMI REL	Salta a una subrutina o etiqueta (REL) si el resultado de una operación es Negativo	3
BMS REL	Salta a una subrutina o etiqueta (REL) si el Bit del Interrupción del registro de banderas (CCR) es uno	3
BNE REL	Salta a una subrutina o etiqueta (REL) si no hay cero	3
BPL REL	Salta a una subrutina o etiqueta (REL) si es resultado de una operación es positivo	3
BRA REL	Salta a una etiqueta (REL) siempre	3

BRCLR N,OPR,REL	Salta a una subrutina o etiqueta (REL) si es el Bit (N) de un registro (OPR) está en cero (CLR).	5
BRN REL	Nunca salta a subrutina o etiqueta (REL)	3
BRSET N,OPR,REL	Salta a una subrutina o etiqueta (REL) si es el Bit (N) de un registro (OPR) está en uno (SET).	5
BSET N,OPR	Toma un bit (N) de un registro (OPR) y lo pone en Uno (SET)	4
BSR REL	Llama a una subrutina (REL)	4
CBEQ OPR,REL	Compara el valor del registro A con el valor del registro (OPR) y salta a una etiqueta o subrutina (REL) si son iguales	5
CBEQA #OPR,REL		4
CBEQX #OPR,REL		4
CBEQ OPR,X+,REL		5
CBEQ X+REL		4
CBEQ OPR,SP,REL		6
CLC	Pone cero en el Bit del <i>Carry</i>	1
CLI	Pone cero en el Bit del <i>Interrupt Mask</i> . (Habilita las interrupciones)	2
CLR OPR	Poner 00h en el registro	3
CLRA		1
CLR X		1
CLR H		1
CLR OPR,X		3
CLR ,X		2
CLR OPR,SP		4
CMP #OPR	Compara el valor del registro A con un registro A-(M)	2
CMP OPR		3
CMP OPR,X		3
CMP ,X		2
CMP OPR,SP		4
COM OPR	Complemento	4
COMA		1
COMX		1
COM OPR,X		4
COM ,X		2
COM OPR,SP	5	
CPHX #OPR	Compara el valor de H:X con el valor almacenado en registro OPR o con el valor de OPR	3
CPHX OPR		4
CPX #OPR	Compara el valor del registro X con un registro X-(M)	2
CPX OPR		3
CPX ,X		3
CPX OPR,X		3
CPX OPR,SP		4
DAA	Ajustar a decimal el registro A (A_{10})	2
DBNZ OPR,REL	Decrementa (Resta en uno) el registro (OPR, A, X, SP) y salta a una etiqueta o subrutina (REL) si no es cero	5
DBNZ A REL		3
DBNZ X REL		3
DBNZ OPR,X,REL		5
DBNZ X,REL		4
DBNZ OPR,SP,REL		6

DEC OPR DECA DECX DEC OPR,X DEC ,X DEC OPR,SP	Decrementa (Resta en uno) el registro (OPR, A, X, SP)	4 1 1 4 3 5
DIV	División así: $A = H:A/X$ A = Cociente (Resultado), H = Residuo	7
EOR #,OPR EOR OPR EOR OPR,X EOR ,X EOR OPR,SP	OR EXCLUSIVA Así: $A = A \oplus OPR$	2 3 4 2 4
INC OPR INCA INCX INC OPR,X INC ,X INC OPR,SP	Incrementa (suma uno) el registro (OPR, A, X, SP)	4 1 1 4 3 5
JMP OPR JMP OPR,X JMP ,X	Salto a una dirección OPR	2 4 2
JSR OPR JSR OPR,X JSR ,X	Salta a una subrutina	4 6 4
LDA #OPR LDA OPR LDA OPR,X LDA ,X LDA OPR,SP	Carga en el registro A el valor (#OPR) o el dato almacenado en OPR $A \leftarrow (M)$	2 3 4 2 4
LDHX #OPR LDHX OPR	Carga en el registro H:X el valor (#OPR) o el dato almacenado en OPR $H:X \leftarrow M:M$	3 4
LDX #OPR LDX OPR LDX OPR,X LDX ,X LDX OPR,SP	Carga en el registro X el valor (#OPR) o el dato almacenado en OPR $X \leftarrow (M)$	2 3 4 2 4
LSL OPR LSLA LSLX LSL OPR,X LSL ,X LSL OPR,SP	Rota a la izquierda el registro así: 	4 1 1 4 3 5
LSR OPR LSRA LSRX LSR OPR,X LSR ,X LSR OPR,SP	Rota a la derecha el registro así: 	4 1 1 4 3 5

MOV OPR,OPR	Mueve así	5
MOV OPR,X		4
MOV #OPR,OPR	Destino ← Fuente	4
MOV X+,OPR		4
MUL	Multiplicación sin signo así: $A \times X = X:A$	5
NEG OPR	Complemento a uno	4
NEGA		1
NEGX		1
NEG OPR,X		4
NEG ,X		3
NEG OPR,SP		5
NOP	No hace nada	1
NSA	Toma los 4 bits de mayor peso del registro A y los intercambia con los 4 bit de menor peso así: 	3
ORA #OPR	Hace la operación lógica OR entre el registro A y el valor (#OPR) o el dato almacenado en OPR, así: $A \leftarrow A \text{ OR } (M)$	2
ORA OPR		3
ORA OPR,X		4
ORA ,X		2
ORA OPR,SP		4
PSHA	Inserta el registro A en SP	2
PSHH	Inserta el registro H en SP	2
PSHX	Inserta el registro X en SP	2
PULA	Saca el registro A en SP	2
PULH	Saca el registro H en SP	2
PULX	Saca el registro X en SP	2
ROL OPR	Rota a la izquierda el registro con Carry así: 	4
ROLA		1
ROLX		1
ROL OPR,X		4
ROL ,X		3
ROL OPR,SP		5
ROR OPR	Rota a la derecha el registro con Carry así: 	4
RORA		1
RORX		1
ROR OPR,X		4
ROR ,X		3
ROR OPR,SP		5
RPS	Resetea el SP así: $SP \leftarrow FF$	1
RTI	Retorna de una interrupción	7
RTS	Retorna de una subrutina	4

SBC #OPR		2
SBC OPR	Resta con <i>Carry</i> , así	3
SBC OPR,X		3
SBC ,X	$A = A - (M) - C$	2
SBC OPR,SP		4
SEC	Poner uno (Set) al bit <i>Carry</i> del registro CCR, así: $C \leftarrow 1$	1
SEI	Poner uno (Set) al bit <i>Interrupt Mark</i> del registro CCR, así: $I \leftarrow 1$. (Deshabilita las interrupciones).	2
STA OPR		3
STA OPR,X	Almacenar A en M, así	3
STA ,X	$M \leftarrow A$	2
STA OPR,SP		4
STHX OPR	Almacenar HX en OPR	4
STOP	Habilita el pin IRQ, y para el oscilador	1
STX OPR		3
STX OPR,X	Almacenar X en M, así	3
STX ,X	$M \leftarrow X$	2
STX OPR,SP		4
SUB #OPR		2
SUB OPR	Resta sin <i>Carry</i> , así	3
SUB OPR,X		3
SUB ,X	$A = A - (M)$	2
SUB OPR,SP		4
SWI	Interrupción de Software	9
TAP	Transferir el Registro A al registro CCR, así: $CCR \leftarrow A$	2
TAX	Transferir el Registro A al Registro X, así: $X \leftarrow A$	1
TPA	Transferir el Registro CCR al Registro A, así: $A \leftarrow CCR$	1
TST OPR		3
TSTA		1
TSTX		1
TST OPR,X	Hacer test si la cantidad es Cero o Negativo	3
TST ,X		2
TST OPR,SP		4
TSX	Transferir el Registro SP al Registro H:X, así: $H:X \leftarrow SP$	2
TXA	Transferir el Registro X al Registro A, así: $A \leftarrow X$	1
TXS	Transferir el Registro H:X al Registro SP, así: $SP \leftarrow H:X$	2

Fuente. NXP-Freescale. (2009). MC9S08JS16RM. HCS08 Microcontrollers. Data Sheet, p. 96.

Donde:

A	= Acumulador	SP	= <i>Stack Pointer</i>
M	= Dato o valor almacenado	H:X	= Registro para direccionamiento Indexado de 16 bits
C	= Carry	REL	= Una etiqueta cualquiera, (Una subrutina)
OPR	= Registro (8 ó 16 Bits)	N	= Cualquier bit
X	= Registro para direccionamiento indexado de 8 bits	#OPR	= Valor inmediato

5.2.5. Programación

A continuación se ajustan algunos de los ejemplos que se desarrollaron en los microcontroladores de Microchip para que el lector pueda desarrollar destreza en ambos fabricantes, siguiendo los mismos pasos de programación planteados en el ejemplo 8 del capítulo III. Se recomienda al lector leer nuevamente este ejemplo.

Para programar y simular los programas de los microcontroladores de *NXP-Freescale* se usará *CodeWarrior* versión 10.1.

Ejemplo 24. Tomar el ejemplo 8 y acondicionarlo para el microcontrolador MC9S08JS16.

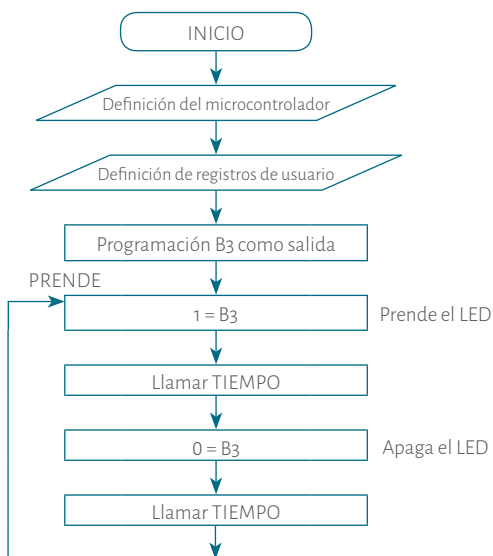
A continuación se desarrolla paso a paso la programación y simulación. Además se explica cómo hacer un proyecto en *CodeWarrior*.

Paso 1. El funcionamiento de un led ya se conoce. Por ende, se omite este paso.

Paso 2. Hacer un diagrama de conexiones. Hay que decir en cuál de las 15 líneas de *In/Out* del microcontrolador se va a poner el led. Se puede tomar B₃, así como el ejemplo 8. En los ejemplos desarrollados con microcontroladores *NXP-Freescale* no se harán Figuras con los montajes. El lector sólo debe seguir la distribución de pines del microcontrolador con el que está trabajando (MC9S08JS16, Figura 56). De la misma forma, se deben tener en cuenta las líneas de alimentación y de reset.

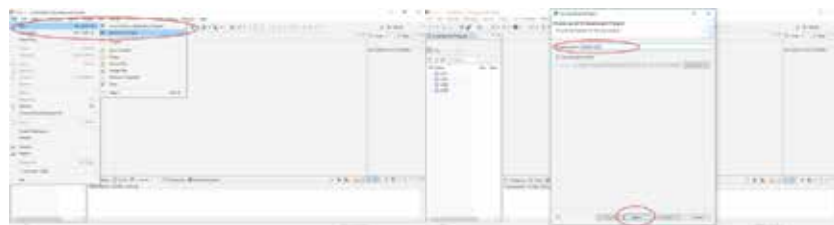
Paso 3. Hacer el diagrama de flujo. De acuerdo con el diagrama de conexiones planteado en el punto anterior hay que programar B₃ como salida para el led (ver diagrama 37). La subrutina tiempo es la misma de los programas anteriores. Por consiguiente, no se hace el diagrama de flujo, ver diagrama 3, ejemplo 8.

Diagrama 37.
Programa que prende y apaga un led. Ejemplo 24



Fuente. Elaboración del autor.

Paso 4: Una vez hecho el diagrama de flujo hay que pasarlo a instrucciones. A continuación se desarrolla el diagrama de flujo bloque por bloque. Cuando se construye un nuevo proyecto se hace la definición del microcontrolador. Para ello, se usa el *CodeWarrior* versión 10.1 u otra versión superior. Cuando se inicia *CodeWarrior* se selecciona una carpeta donde se guardan los proyectos, aunque se recomienda al lector tener una carpeta para ellos. Una vez abierto el programa hacer clic en **File/New/Bareboard Project**, como se observa en la Figura 58A. En la siguiente ventana darle el nombre al proyecto, en este caso ejemplo 24 y hacer clic en **Next** (ver Figura 58B).



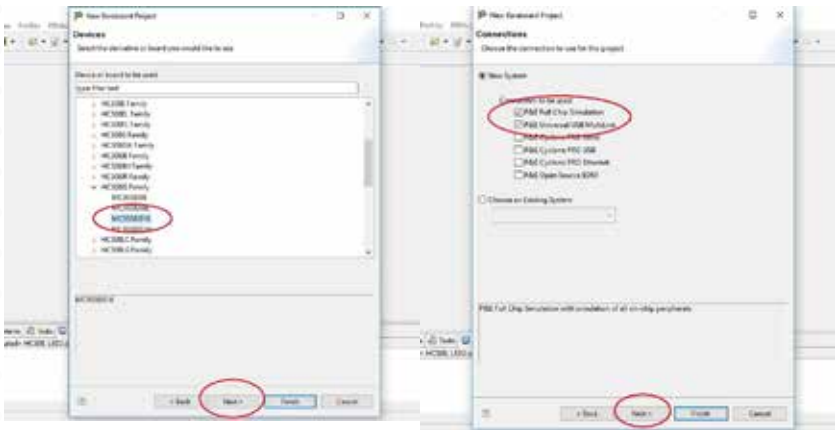
58 A

58 B

Figura 58. Crear proyecto en CodeWarrior 10.1 Paso 1

Fuente. Captura de pantalla CodeWarrior 10.1 Paso 1.

Después hay que seleccionar el microcontrolador en el cual se va a realizar el programa. Para ello, buscar la familia HCS08JS y, por último, seleccionar el MC9S08JS16 y hacer clic en **Next** (Figura 59A). En la ventana siguiente seleccionar *Full Chip Simulation* y *Universal/USB Multilink*. El primero sirve para simular el programa, el segundo es el sistema de desarrollo para programar el microcontrolador. Finalmente, se hace clic en **Next**, ver Figura 59B.



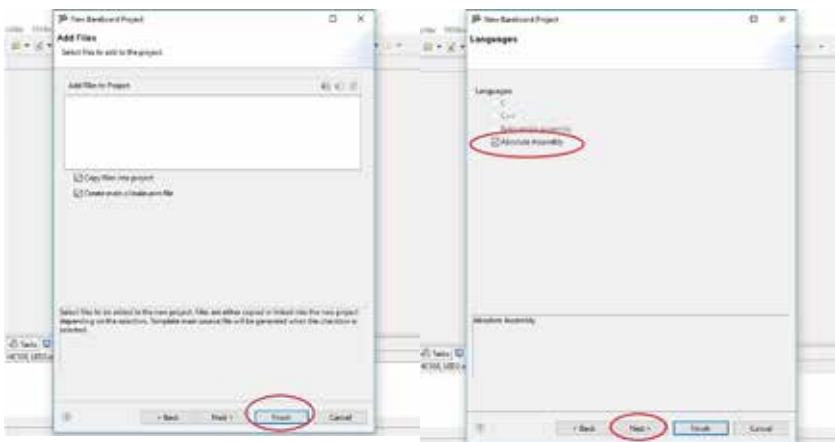
59 A

59 B

Figura 59. Crear proyecto en CodeWarrior 10.1 Paso 2

Fuente. Captura de pantalla CodeWarrior 10.1 Paso 2

En la ventana que aparece se hace clic en Next, Figura 60A. En la siguiente ventana seleccionar *Absolute Assembly* y hacer clic en Next, ver Figura 60B.



60 A

60 B

Figura 60. Crear proyecto en CodeWarrior 10.1 Paso 3

Fuente. Captura de pantalla CodeWarrior 10.1 Paso 3

En la siguiente ventana hacer clic en Finish, ver Figura 61A. En ese momento es donde se crea el proyecto. Hay que seleccionar el proyecto y desplegar todas las carpetas en la carpeta **Sources**. Se debe seleccionar esta carpeta y hacer clic en **mian**. En la ventana que se abre hay una plantilla donde se escribe el programa del usuario. El lector podrá observar dos óvalos rojos en la Figura 61B. En el primer óvalo el usuario puede definir los registros para su uso y puede programar los puertos. En el segundo óvalo se escribe el programa principal y las subrutinas.

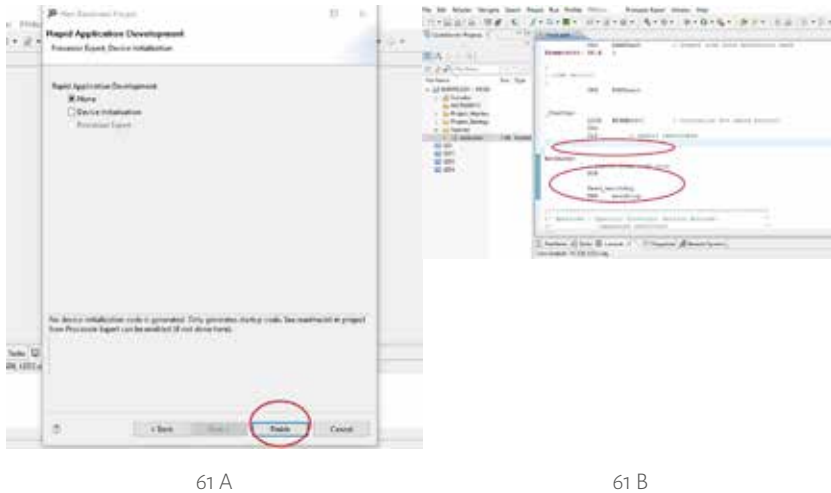


Figura 61. Crear proyecto en CodeWarrior 10.1 Paso 4

Fuente. Captura de pantalla CodeWarrior 10.1 Paso 4

Ahora se hace la definición de los registros del usuario. Recordar que puede iniciar en la posición $0x0080_H$ hasta $0x027F_H$, ver Figura 57. Tener en cuenta que para este compilador asignar una posición en hexadecimal se le antepone el signo pesos (\$). Además que hay usar los dos puntos después del registro:

```
CON1: EQU $80 ;REGISTROS PARA LA SUBROUTINA TIEMPO
CON2: EQU $81
```

Ahora hay que programar B3 como salida para el led, recordar que los registros de control son los **DDRX**, para este caso se usará el **DDRB**:

```
BSET 3,PTBDD ;B3 COMO SALIDA PARA EL LED
```


Observe que se usó la instrucción **BSET**. Esta instrucción tiene el siguiente formato: **BSET n,opr** con el que se toma un bit que lo representa **n** de un registro cualquiera que lo representa **opr** y lo pone en uno (SET). Así, se programa la línea 3 del puerto B como salida. El siguiente bloque del diagrama de flujo es **enciende el led** y allí se inicia el programa principal. Para ello, la plantilla asigna una etiqueta o nivel que se llama **mainLoop**, ver Figura 61b. Para encender el led se usa la misma instrucción anterior pero con la diferencia de que se pone en 1 el bit 3 del puerto B (PTB) así:

```
BSET 3,PTBD ;PRENDE EL LED EN B3
```

Siguiendo el diagrama de flujo ahora hay que llamar tiempo para que el ojo humano pueda observar el led prendido. Para ello, se emplea la instrucción **BSR → rel**. Tener en cuenta que se puede usar esta instrucción siempre y cuando el salto no supere las 128 posiciones, tanto arriba como abajo desde donde se esté llamando.

```
BSR TIEMPO ;SE DA TIEMPO PARA VER EL LED PRENDIDO
```

Posteriormente, hay que apagar el led, utilizando la instrucción **BCLR n,opr**. Esta instrucción toma un bit, que lo representa **n**, de un registro cualquiera, lo representa **opr** y lo pone en cero:

```
BCLR 3,PTBD ;APAGA EL LED EN B3
```

Aquí nuevamente se llama tiempo para que el ojo humano pueda observar el led apagado así:

```
BSR TIEMPO ;SE DA TIEMPO PARA VER EL LED APAGADO
```

Según el diagrama de flujo hay que regresar a prender el led, es decir hasta aquí es el programa principal, para ello se usa la instrucción **BRA rel**; la cual se usa para hacer saltos incondicionales a cualquier parte del programa, para este caso se salta a **mainLoop** así:

```
BRA mainLoop ;SALTA A PRENDER NUEVAMENTE
```

Ya se escribió el programa principal, sólo queda hacer la subrutina tiempo. Siguiendo el diagrama 3 hay que cargar los registros contadores CON1 y CON2 me-

diante la instrucción **MOV opr,opr**. Esta instrucción mueve un valor cualquiera a cualquier registro, ya sea hexadecimal, decimal o binario. Para entender un poco se puede observar la Tabla 32 donde se muestra cómo se usa esta instrucción para cargar el valor 100 en decimal y los equivalentes en hexadecimal, ASCII y en binario con el registro CON1.

Tabla 32.

Forma de uso de la instrucción **MOV opr,opr**

Forma de uso de la instrucción	Explicación
MOV #$\\$64D$,CON1	Mueve el valor 64 en Hexadecimal al registro CON1
MOV #100t,CON1	Mueve el valor 100 en Decimal al registro CON1
MOV #%1100100,CON1	Mueve el valor 01100100 en Binario al registro CON1
MOV #'@',CON1	Mueve el valor ASCII de la letra @ (1100100) al registro CON1

Fuente. Elaboración propia.

El lector puede observar cómo se escribió la instrucción: primero se escribe el valor a cargar (fuente) y después el registro (destino). Siguiendo el diagrama 3 se carga 30H y 20H a los registros CON1 y CON2 respectivamente así:

TIEMPO:
N1:

MOV # $\$30$,CON1 ; MUEVE EL VALOR 30 H AL REGISTRO CON1.

N1:

MOV # $\$20$,CON2 ; MUEVE EL VALOR 20 H AL REGISTRO CON2.

Note que hay la etiqueta N1 es donde se regresa a cargar nuevamente el registro **CON2** cuando **CON1** aún no es equivalente a cero. Siguiendo el diagrama hay que restar en uno el **CON2** y preguntar si es cero, utilizandola instrucción **DBNZ opr,rel**. Esta instrucción decrementa (resta en uno cualquier registro, que lo representa **opr**) y salta, si no es cero (salta a una etiqueta que la representa **rel**):

N2:

DBNZ CON2,N2 ; CON2-1=CON2, SI NO ES CERO SALTA A N2

Ahora hay que restar en uno el **CON1** y preguntar si es cero. Para ello, se aplica la misma instrucción anterior:

```
DBNZ    CON1,N1          ; CON1-1 = CON1, SI NO ES CERO SALTA A N1
```

Por último, se usa la instrucción **RTS** para retornar al programa principal desde la subrutina TIEMPO así:

```
RTS          ; RETORNA AL PROGRAMA PRINCIPAL
```

Una vez escrito el programa y explicada cada una de las partes del diagrama de flujo, se puede observar el programa completo:

```
; Include derivative-specific definitions
INCLUDE 'derivative.inc'

;
; export symbols
;

XDEF _Startup
ABSENTRY _Startup

;
; variable/data section
;

ORG RAMStart      ; Insert your data definition here
ExampleVar: DS.B 1

;
; code section
;

ORG ROMStart

_Startup:
LDHX #RAMEnd+1    ; initialize the stack pointer
TXS
CLI                ; enable interrupts

CON1: EQU $80     ; REGISTROS PARA LA SUBROUTINA TIEMPO
CON2: EQU $81

BSET 3,PTBDD      ; B3 COMO SALIDA PARA EL LED

mainLoop:
BSET 3,PTBD       ; PRENDE EL LED EN B3
BSR TIEMPO        ; SE DA TIEMPO PARA VER EL LED PRENDIDO
BCLR 3,PTBD       ; APAGA EL LED EN B3
BSR TIEMPO        ; SE DA TIEMPO PARA VER EL LED APAGADO
feed_watchdog
BRA mainLoop      ; SALTA A PRENDER NUEVAMENTE
```

```

TIEMPO:      MOV      #30,CON1      ; MUEVE EL VALOR 30 H AL REGISTRO CON1.
N1:          MOV      #20,CON2      ; MUEVE EL VALOR 20 H AL REGISTRO CON2.
N2:          DBNZ     CON2,N2        ; CON2-1 = CON2, SINO ES CERO SALTA A N2
            DBNZ     CON1,N1        ; CON1-1 = CON1, SINO ES CERO SALTA A N1
            RTS                ; RETORNA AL PROGRAMA PRINCIPAL

```

```

;*****
;
; spurious - Spurious Interrupt Service Routine.*
;
; (unwanted interrupt) *
;*****

```

```

spurious:    NOP                ; placed here so that security value
            RTI                ; does not change all the time.

```

```

;*****
;
; Interrupt Vectors *
;*****

```

```

ORG          $FFFA

            DC.W spurious        ;
            DC.W spurious        ; SWI
            DC.W _Startup        ; Reset

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Hay que saber la distribución del microcontrolador (ver Figura 56) y conocer el diagrama de conexiones planteado en el paso dos.

Paso 6. Hacer las correcciones pertinentes y volver al paso tres.

Ejemplo 25. Tomar el ejemplo 9 y acondicionarlo para el microcontrolador MC9S08JS16. A continuación se desarrolla paso a paso:

Paso 1. Se sabe que el funcionamiento de un led, así como se dijo en el ejemplo 9, sólo requiere adicionar 7 leds más en el puerto A para hacer el juego de luces.

Paso 2. Como se mencionó en el punto anterior, se trabajará todo el puerto para los 8 leds. El programador entenderá que le puede agregar al circuito anterior los 7 led's restantes.

Paso 3. Diagrama de flujo: hay que programar todo el PORTA como salida para los 8 leds. Para controlar la rotación tanto para la derecha como para la izquierda se pueden usar las instrucciones de rotación con Carry, tanto para izquierda como para la derecha, ver tabla 30; entonces sólo es preguntar por el Carry, de esa forma se puede tener control de las rotaciones. El Puerto A inicia con 0000 0001 en binario para cuando rota a la izquierda y cuando va a la derecha se le asegura 1000 0000 en binario, ver diagrama 38.

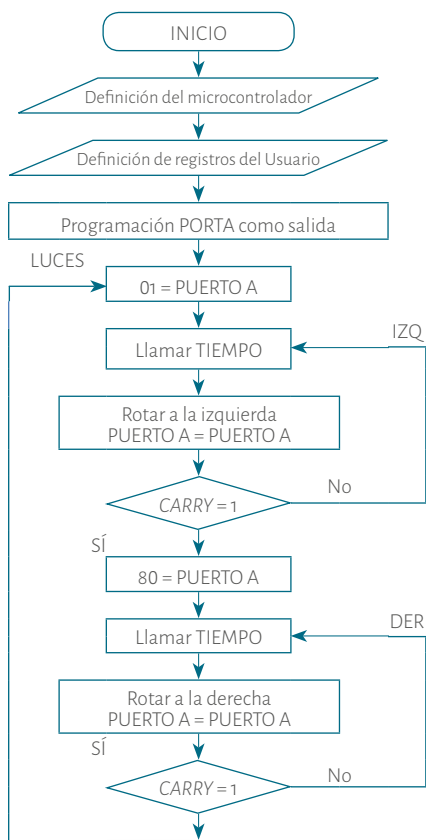
Paso 4. El programa se puede observar después del diagrama 38. Note que la subrutina de tiempo es la misma del ejemplo anterior. Si se desea que el juego de luces sea más rápido o más lento, sólo se aumenta o disminuye el valor a los contadores desde 00_H a FF_H . Una vez digitado el programa se puede compilar. El lector podrá notar que las instrucciones nuevas usadas en este programa son las de rotación, tanto a la izquierda como a la derecha. A continuación se describen estas instrucciones:

LSL	PTAD	; SE ROTA A LA IZQUIERDA EL PUERTO A, CON CARRY
LSR	PTAD	; SE ROTA A LA DERECHA EL PUERTO A, CON CARRY

Para preguntar por el estado del bit del Carry:

BCC	REL	; Salta a una subrutina o etiqueta (REL) si el Bit del Carry del ; Registro de banderas (CCR) es cero
-----	-----	--

Diagrama 38.
Programa juego de luces. Ejemplo 25.



Fuente. Elaboración propia.

Una vez explicadas las nuevas instrucciones que se usarán, sólo queda escribir el programa completo:

```

; Include derivative-specific definitions
INCLUDE derivative.inc

;
; export symbols
;

XDEF _Startup
ABSENTRY _Startup

;
; variable/data section
;

ORG RAMStart ; Insert your data definition here
  
```

ExampleVar: DS.B 1

```

;
; code section
;

ORG ROMStart

; Startup:

LDHX #RAMEnd+1 ; initialize the stack pointer
TXS
CLI ; enable interrupts

CON1: EQU $80 ; CON1 Y CON2 CONTADORES PARA LA SUBROUTINA
; TIEMPO

CON2: EQU $81

MOV #$$FF,PTADD ; PUERTO COMO SALIDA PARA LOS LEDS

LUCES:
MOV #%00000001,PTAD ; SE INICIA CON 0000 0001

IZQ:
BSR TIEMPO ; SE DA TIEMPO PARA VER LA ROTACIÓN
LSL PTAD ; SE ROTA A LA IZQUIERDA EL PUERTO B
BCC IZQ ; SI EL CARRY ES CERO SALTA A IQZ
MOV #%10000000,PTAD ; SE INICIA CON 0000 0001

DER:
BSR TIEMPO ; SE DA TIEMPO PARA VER LA ROTACIÓN
LSR PTAD ; SE ROTA A LA DERECHA EL PUERTO B
BCC DER ; SI EL CARRY ES CERO SALTA A DER
feed_watchdog
BRA LUCES ; REGRESA A INICIAR ROTACIÓN A LA IZQUIERDA

TIEMPO:
MOV #30,CON1 ; MUEVE EL VALOR 30 H AL REGISTRO CON1.

N1:
MOV #20,CON2 ; MUEVE EL VALOR 20 H AL REGISTRO CON2.

N2:
DBNZ CON2,N2 ; CON2-1 = CON2, SINO ES CERO SALTA A N2
DBNZ CON1,N1 ; CON1-1 = CON1, SINO ES CERO SALTA A N1
RTS ; RETORNA AL PROGRAMA PRINCIPAL

;*****
; spurious - Spurious Interrupt Service Routine. *
; (unwanted interrupt) *
;*****

spurious: ; placed here so that security value
NOP ; does not change all the time.
RTI

```

```

;*****
;
;      Interrupt Vectors      *
;*****
;
;      ORG      $FFFA
;
;      DC.W    spurious      ;
;      DC.W    spurious      ;SWI
;      DC.W    _Startup      ;Reset

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 56) y el diagrama de conexiones planteado en el paso dos.

Paso 6. Hacer las correcciones pertinentes y volver al paso tres.

Con los dos ejemplos anteriores se trabajó el manejo de puertos sólo de salida. A partir del siguiente ejemplo se van a realizar ejercicios tanto de entrada como de salida para visualizar cómo se toman decisiones en los microcontroladores dependiendo de un dato del exterior.

Ejemplo 26. Tomar el ejemplo 10 y acondicionarlo para el microcontrolador MC9S08JS16. continuación se desarrolla paso a paso:

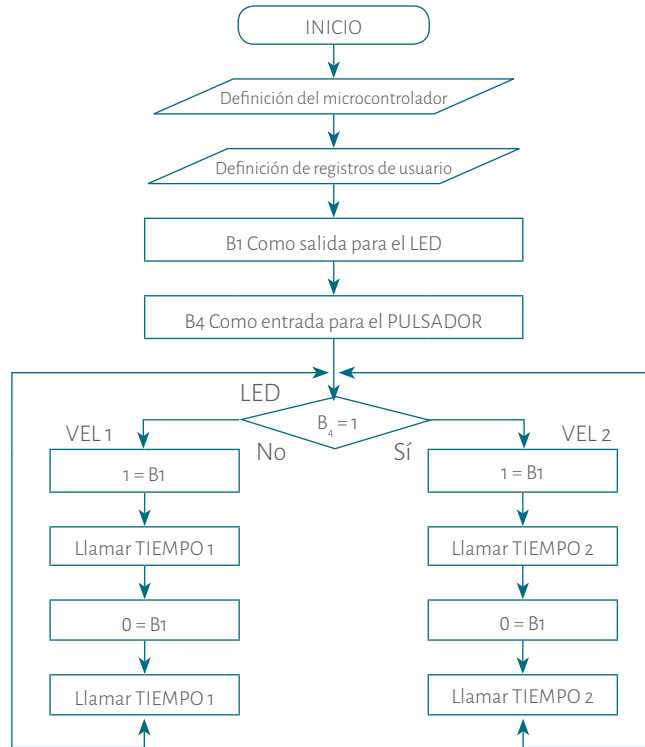
Paso 1. En el ejemplo 8 se explicó el funcionamiento de un led y en el ejemplo 10 se explicó cómo trabaja un pulsador normalmente abierto.

Paso 2. Para este ejercicio se puede tomar la misma distribución del ejemplo 10. Es decir, B₁ para el led y B₄ para el pulsador.

Paso 3. Diagrama de flujo: según lo planteado en el punto anterior hay que programar B₁ como salida para el led y B₄ como entrada para el pulsador (ver diagrama 39).

Diagrama 39.

Programa para prender y apagar un led con dos velocidades. Ejemplo 26.



Fuente. Elaboración propia.

Como se mencionó en el ejemplo 10 hay dos subrutinas de retardo con las que se establecen distintas velocidades: TIEMPO 1 y TIEMPO 2.

Paso 4. Para el desarrollo de este programa se usará una instrucción que no se había trabajado la cual es:

BSET n,opr,rel

Esta instrucción permite preguntar por un bit (que lo representa **(n)**) de un registro (que lo indica **(opr)**) si está en uno (**SET**), salta a una subrutina (**rel**); y si esta en cero no salta, es decir ejecuta la siguiente instrucción. Para el caso de este ejemplo es la que permite tomar la decisión de qué subrutina ejecutar, si B4 es uno ejecuta la subrutina **VEL2** y si B4 es cero ejecuta la subrutina **VEL1**.

Estos microcontroladores también tienen la posibilidad de preguntar por si un bit es cero (CLR) la instrucción es:

```
BCLR n,opr,rel
```

Esta instrucción funciona a la inversa de la interior, es decir que salta a una etiqueta (**rel**) si el bit (**n**) de un registro (**opr**) es cero (**CLR**).

Ahora sí se puede escribir el programa según lo planteado en el diagrama de flujo 38:

```
; Include derivative-specific definitions
INCLUDE 'derivative.inc'

;
; export symbols
;
    XDEF _Startup
    ABSENTRY _Startup
;
; variable/data section
;
    ORG RAMStart ; Insert your data definition here
ExampleVar: DS.B 1
;
; code section
;
    ORG ROMStart

_Startup:
LDHX #RAMEnd+1 ; initialize the stack pointer
TXS
CLI ; enable interrupts

CON1: EQU $80 ; CON1 Y CON2 CONTADORES PARA LA SUBROUTINA
; TIEMPO

CON2: EQU $81
BSET PTBDD,1 ; B1 COMO SALIDA PARA EL LED
BCLR PTBDD,4 ; B4 COMO IN PARA EL PULSADOR

LED:
BRSET 4,PTBD,VEL2 ; SE PREGUNTA SI B4 ES UNO, SI ES UNO SALTA A VEL2
; Y SI ES CERO SALTA A VEL1
```

```

VEL1:
    BSET    PTBD,1           ; SUBROUTINA DE VELOCIDAD UNO
    BSR     TIEMPO1
    BCLR    PTBD,1
    BSR     TIEMPO1
    feed_watchdog
    BRA     LED
    
```

```

VEL2:
    BSET    PTBD,1           ; SUBROUTINA DE VELOCIDAD DOS
    BSR     TIEMPO2
    BCLR    PTBD,1
    BSR     TIEMPO2
    feed_watchdog
    BRA     LED
    
```

```

TIEMPO1:
    MOV     #30,CON1        ; MUEVE EL VALOR 30 H AL REGISTRO CON1.
    
```

```

N1:
    MOV     #20,CON2        ; MUEVE EL VALOR 20 H AL REGISTRO CON2.
    
```

```

N2:
    DBNZ    CON2,N2         ; CON2 - 1 = CON2, SINO ES CERO SALTA A N2
    DBNZ    CON1,N1         ; CON1 - 1 = CON1, SINO ES CERO SALTA A N1
    RTS
    
```

```

TIEMPO2:
    MOV     #50,CON1        ; MUEVE EL VALOR 50 H AL REGISTRO CON1.
    
```

```

N1:
    MOV     #60,CON2        ; MUEVE EL VALOR 60 H AL REGISTRO CON2.
    
```

```

N2:
    DBNZ    CON2,N2         ; CON2 - 1 = CON2, SI NO ES CERO SALTA A N2
    DBNZ    CON1,N1         ; CON1 - 1 = CON1, SI NO ES CERO SALTA A N1
    RTS
    
```

```

;*****
;* spurious - Spurious Interrupt Service Routine. *
;* (unwanted interrupt) *
;*****
    
```

```

spurious:
    NOP                       ; placed here so that security value
    RTI                       ; does not change all the time.
    
```

```

;*****
;* Interrupt Vectors *
;*****
    
```

```

ORG     $FFFA
    
```

```
DC.W spurious
DC.W spurious
DC.W _Startup
```

```
;
;SWI
;Reset
```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 56) y conocer el diagrama de conexiones planteado en el paso 2.

Paso 6. Hacer las correcciones pertinentes y volver al paso tres.

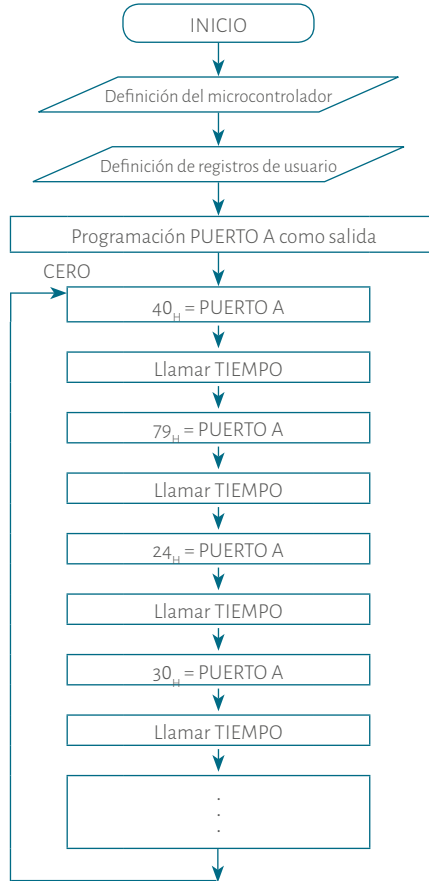
Ejemplo 27. Tomar el ejemplo 11 y acondicionarlo para el microcontrolador MC9S08JS16. A continuación se desarrolla paso a paso:

Paso 1. En el ejemplo once se utilizó y se explicó cómo funciona un display de ánodo común. Por lo tanto, no se explica.

Paso 2. Para este ejercicio se usará el puerto A desde A0 hasta A6, una línea del puerto para cada segmento (ejemplo 11).

Paso 3. Diagrama de flujo: hay que usar el puerto A de salida. El diagrama de flujo 40 muestra cómo obtener los datos. Es importante recordar que estos datos son los que se observan en la Tabla 12. Note que el diagrama no está completo. El lector puede completarlo, solo debe seguir la secuencia.

Diagrama 40.
Programa para visualizar los números decimales del 0 al 9 en un Display 7 segmentos. Ejemplo 27



Fuente. Elaboración propia.

Paso 4. La subrutina de tiempo sigue siendo igual a los ejemplos anteriores. El programa es el siguiente:

```

; Include derivative-specific definitions
INCLUDE 'derivative.inc'

;
; export symbols
;

XDEF_Startup
ABSENTRY_Startup

;
; variable/data section
;

```

```

        ORG RAMStart          ; Insert your data definition here
ExampleVar: DSB 1
;
; code section
;

        ORG ROMStart

_Startup:
LDHX #RAMEnd+1          ; initialize the stack pointer
TXS
CLI                    ; enable interrupts

CON1: EQU $80          ; CON1 Y CON2 CONTADORES PARA LA SUBROUTINA
                        ; TIEMPO
CON2: EQU $81

MOV #FF,PTADD          ; PUERTO A DE SALIDA PARA LOS SEGMENTOS DEL
                        ; DISPLAY

CERO:
MOV #$40,PTAD          ; SE VISUALIZA CERO EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$79,PTAD          ; SE VISUALIZA UNO EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$24,PTAD          ; SE VISUALIZA DOS EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$30,PTAD          ; SE VISUALIZA TRES EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$19,PTAD          ; SE VISUALIZA CUATRO EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$12,PTAD          ; SE VISUALIZA CINCO EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$02,PTAD          ; SE VISUALIZA SEIS EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$78,PTAD          ; SE VISUALIZA SIETE EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

MOV #$00,PTAD          ; SE VISUALIZA OCHO EN EL DISPLAY
BSR TIEMPO             ; TIEMPO PARA VER EL DATO

```

```

MOV    #§10,PTAD           ; SE VISUALIZA DIEZ EN EL DISPLAY
BSR    TIEMPO              ; TIEMPO PARA VER EL DATO

feed_watchdog
BRA    CERO

TIEMPO:
MOV    #§30,CON1          ; MUEVE EL VALOR 30 H AL REGISTRO CON1.
N1:
MOV    #§20,CON2          ; MUEVE EL VALOR 20 H AL REGISTRO CON2.
N2:
DBNZ   CON2,N2            ; CON2 - 1 = CON2, SINO ES CERO SALTA A N2
DBNZ   CON1,N1            ; CON1 - 1 = CON1, SINO ES CERO SALTA A N1
RTS    ; RETORNA AL PROGRAMA PRINCIPAL
;*****
;* spurious - Spurious Interrupt Service Routine. *
;* (unwanted interrupt) *
;*****

spurious:
NOP    ; placed here so that security value
RTI    ; does not change all the time.

;*****
;* Interrupt Vectors *
;*****

ORG    $FFFA

DC.W  spurious
DC.W  spurious
DC.W  _Startup

```

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que saber la distribución del microcontrolador (ver Figura 56) y el diagrama de conexiones planteado en el paso 2.

Paso 6. Hacer las correcciones pertinentes y volver al paso 3.

5.3. Microcontroladores texas instruments MSP430.

Una vez vistos los microcontroladores de Microchip, de Atmel y de NXP-Freescale. Ahora se estudiarán los fabricados por Texas Instruments. Para este fabricante el microcontrolador más pequeño es de 16 bits. Entre los microcontroladores de 16 bits

se encuentran el MSP430F1132, el MSP430F1232 el MSP430F2013, el MSP430F2272 y el MSP430F2274, entre otros. En la Tabla 33 se muestran algunas de las características de cada uno de ellos.

Tabla 33.
Comparación entre algunos Microcontroladores de 16 Bits. Texas.

Características	MSP430F1132	MSP430F1232	MSP430F2013	MSP430F2272	MSP430F2274
Memoria de programa (ROM) FLASH	8KBX16	8KBX16	2KBX16	32KBX16	32KBX16
Memoria RAM	256 X 16	256 X 16	128 X 16	1K X 16	1K X 16
Pines de I/O	14	22	10	32	32
Oscilador interno	S1	S1	S1	S1	S1
Tipo de empaque	TSSOP/QFN	TSSOP/QFN	DIP/ TSSOP	TSSOP/QFN	TSSOP/QFN
Numero de pines	20 28		14	38	38

Notas. MSP430F1132 datos tomados de TEXAS INSTRUMENTS. (2002-2004).

MSP430F1232 datos tomados de TEXAS INSTRUMENTS. (2002-2004).

MSP430F2013 tomado de TEXAS INSTRUMENTS. (2013).

MSP430F2272 y **MSP430F2274** tomado de TEXAS INSTRUMENTS. (2012).

Se hace la selección del MSP430F2272, pues es un microcontrolador poderoso. Además se pueden realizar las mismas aplicaciones que se realizaron con los microcontroladores PIC16F628A, ATtiny2313 y el MC9S08JS16 (Microchip, Atmel y NXP-Freescale). El lector podrá observar que estas aplicaciones pueden servir para los otros mencionados que se analizaron en la Tabla 33, sólo hay que tener en cuenta el mapa de memoria de cada uno. En la Tabla 34 se presenta la comparación entre los microcontroladores PIC16F628A, ATtiny2313, MC9S08JS16 el MSP430F2272, en la que se muestran las principales características de cada uno de ellos.

Tabla 34.
Comparación entre los Microcontroladores PIC16F62A, ATtiny2313, MC9S08JS16 el MSP430F2272

Microcontroladores	Memoria de Programa (ROM) FLASH	Memoria RAM	Pines de I/O	Oscilador Interno	Tipo de Em-paque	Numero de Pines
PIC16F628A	2K X 14	128 X 8	15	Sí	DIP	18
ATtiny2313	2K X Bytes	128 X 8	18	Sí	PDIP/SOIC/MLF	20/32
MC9S08JS16	16384 X 8	512 X 8	14	Sí	QFN/SOIC	24/20
MSP430F2272	32 K x 16	1 K x 16	32	Sí	TSSOP/QFN	38

Nota. Los datos de PIC16F628A fueron tomados de MICROCHIP. Technology Inc. (2007a).

Los datos de ATtiny2313 fueron tomados de ATMEL[®]. 8-bit AVR[®]. (2016). (p. 1).
 Los datos de MC9S08JS16 fueron tomados de NXP-Freescale. (2009).
Los datos de MSP430F2272 fueron tomados de TEXAS INSTRUMENTS. (2012).

5.3.1. MSP430F2272

En la Tabla 33 y 34 se comentó que el **MSP430F2272** es de 38 pines empaquetado TSSOP ó QFN. En la Figura 62 se muestra la distribución de estos.

En la Figura 63 se muestra el mapa de memoria. El lector podrá interpretar y explorar todas las posiciones del mapa de memoria. Por el momento, se quiere avanzar en programación en Assembler. Para tal motivo, se explicarán algunos de los registros que posee este microcontrolador. Sin embargo, para la mayor información se puede consultar el *Data Sheet*:

Figura 62. Distribución de pines del MSP430F2272.



Fuente. TEXAS INSTRUMENTS. (2012). MSP430F22x2, MSP430F22x4. MIXED SIGNAL Microcontroller (p.3)

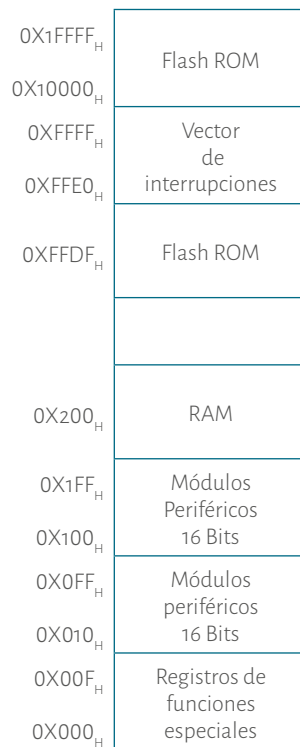


Figura 63. Distribución de memoria del MSP430F272.

Fuente. TEXAS INSTRUMENTS, 2013, (p. 25)

Este tipo de microcontroladores tienen 16 registros acumuladores, con los cuales se evita el cuello de botella al tener uno o dos de ellos al igual que en los microcontroladores de Atmel. Estos registros pueden ser usados para propósito general o direccionamiento de memoria. En la figura No. 64 se muestran.

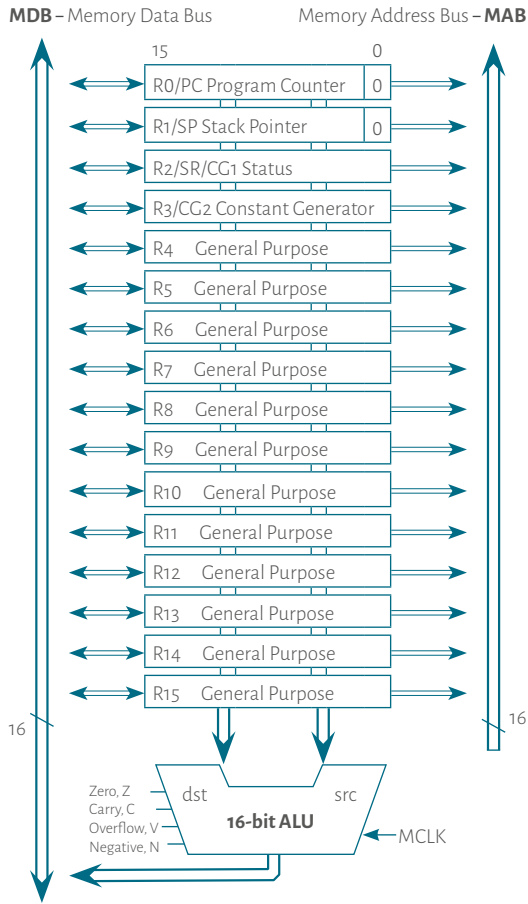


Figura 64. Parte de la CPU del MSP430F2272.

Fuente. TEXAS INSTRUMENTS, 2013, (p. 44)

REGISTRO STATUS/R2: en la Figura 64 se puede apreciar que R2 es el registro que tiene el estado de las operaciones. Este registro es muy similar al registro **STATUS** y al **CCR** en los microcontroladores de Microchip, Atmel y NXP-Frecale.

15	9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reservado	V	SCG1	SCG0	OSC OFF	CPU OFF	GIE	N	Z	C	

Fuente. TEXAS INSTRUMENTS, 2013, (pp. 45-46)

BIT 8: **V: (OVERFLOW)** DETECTA EL CAMBIO ERRÓNEO DE SIGNO EN UNA OPERACIÓN ARITMÉTICA

BIT 7-6: **SCG1-SCG0** SISTEMA GENERADOR DE RELOJ

BIT 5: **OSCOFF:**

BIT 4: **CPU OFF:**

BIT 3: **GIE: (INTERRUPT MASK)** MASCARA DE INTERRUPCIÓN.

1 → LAS INTERRUPCIONES ESTÁN HABILITADAS.

0 → LAS INTERRUPCIONES ESTÁN DESHABILITADAS.

BIT 2: **N: (ZERO)** BANDERA DE NEGATIVO

1: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA O LÓGICA DA CERO

0: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA O LÓGICA NO DA CERO.

BIT 1: **Z: (ZERO)** BANDERA DEL CERO

1: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA O LÓGICA DA CERO

0: EL RESULTADO DE UNA OPERACIÓN ARITMÉTICA O LÓGICA NO DA CERO.

BIT 0: **C: (CARRY)** CARRY.

1: EL RESULTADO DE UNA OPERACIÓN SIN SIGNO PASA DE LA CAPACIDAD DE OPERACIÓN, ES DECIR EL CARRY DE 16 BITS.

0: EL RESULTADO DE UNA OPERACIÓN SIN SIGNO PASA DE LA CAPACIDAD DE OPERACIÓN, ES DECIR DE 16 BITS.

5.3.2. PINES DE I/O

El microcontrolador MSP430F2272 cuenta con 32 líneas de entrada y salida (I/O) distribuidas en 4 puertos (Port1, Port2, Port3, Port4). El puerto 1 tiene 8 un bits, el puerto 2 tiene 8, el puerto 3 tiene 8 bits, el puerto 4 tiene 8 bits. Para el control de los puertos de entrada o salida se tienen los registros de control: P1DIR,P2DIR,P3DIR,P4DIR. Estos registros controlan respectivamente cada puerto. Para programar una línea de salida se le garantiza uno (1) y para programarla de entrada se le garantiza cero (0). Es importante aclarar que es lo contrario de los de Microchip y lo mismo de los de Atmel y *NXP-Freescale*.

5.3.3. Set de instrucciones

Antes de empezar a desarrollar ejemplos de aplicaciones en el microcontrolador, es necesario ver las instrucciones. En el capítulo III se mencionó que este fabricante está en el grupo RISC. En la Tabla 35 se muestran las instrucciones, las cuales fueron tomadas del *Datasheet* del microcontrolador.

Tabla 35.
Set instrucciones del MSP430F2272

INSTRUCCIÓN	OPERACIÓN	BIT DEL STATUS			
		V	N	Z	C
MOV(.B) src,dst	Mueve así: src \oplus dst	-	-	-	-
ADD(.B) src,dst	Suma así: src + dst \oplus dst	*	*	*	*
ADDC(.B) src,dst	Suma así: src + dst + C \oplus dst	*	*	*	*
SUB(.B) src,dst	Resta así: dst + not.src + 1 \oplus dst	*	*	*	*
SUB(.B) src,dst	Resta así: dst + not.src + C \oplus dst	*	*	*	*
CMP(.B) src,dst	Resta así: dst - src Esta instrucción es la de comparación. No afecta ninguno de los dos registros involucrados. Sin embargo, afecta los bits del Status, por los cuales se puede preguntar	*	*	*	*
DADD(.B) src,dst	Suma así: src + dst + C \oplus dst (en decimal)	*	*	*	*
BIT(.B) src,dst	Operación lógica AND así: src AND dst \oplus dst No afecta ninguno de los dos registros involucrados. Sin embargo, si afecta los bits del Status, por los cuales se puede preguntar.	0	*	*	*
BIC(.B) src,dst	Toma un bit (src) de un registro (dst) y lo pone en CERO	-	-	-	-
BIS(.B) src,dst	Toma un bit (src) de un registro (dst) y lo pone en UNO	-	-	-	-
XOR(.B) src,dst	Suma exclusiva así: src \oplus dst \oplus dst	*	*	*	*
AND(.B) src,dst	Operación lógica AND así: src AND dst \oplus dst	0	*	*	*
RRC(.B) dst	Rota el registro dst con carry a la derecha	*	*	*	*
RRA(.B) dst	Rotación aritmética del registro dst a la derecha	*	*	*	*
PUSH(.B) src	SP - 2 = SP Y guarda src en SP	-	-	-	-
SWPB dst	Intercambia los bit de mayor (15 al 8) peso a con los de menor peso (7 al 0) del dst	-	-	-	-
CALL dst	Llamado a un subrutina (dst)	-	-	-	-
RET	Retorna de una subrutina	-	-	-	-
JEQ/JZ Label	Salta a una etiqueta (Label) si el bit Zero es Uno	-	-	-	-

JNE/JNZ	Label	Salta a una etiqueta (Label) si el bit Zero es Cero	-	-	-	-
JC	Label	Salta a una etiqueta (Label) si el bit Carry es Uno	-	-	-	-
JNC	Label	Salta a una etiqueta (Label) si el bit Carry es Cero	-	-	-	-
JN	Label	Salta a una etiqueta (Label) si el bit Negative es Uno	-	-	-	-
JGE	Label	Salta a una etiqueta (Label) si (N XOR V) es Cero	-	-	-	-
JL	Label	Salta a una etiqueta (Label) si (N XOR V) es Uno	-	-	-	-
JMP	Label	Salta a una etiqueta (Label) incondicional	-	-	-	-

Fuente. TEXAS INSTRUMENTS, 2013, p.57

Donde:

src= Fuente	dst= Destino
= Ningún Bit del Status es afectado	0= Un Bit del Status es puesto en CERO
*= Algún Bit del Status es afectado	1= Un Bit del Status es puesto en UNO

Fuente. TEXAS INSTRUMENTS, 2013, (p.57)

En las instrucciones que tienen (B) se pueden usar para hacer operaciones en 8 bits, al poner en b en la instrucción.

5.3.4. Programación

A continuación se acondicionará un ejemplo que se desarrolló en los microcontroladores de Microchip, Atmel y NXP-Freescale, con lo cual el lector tendrá mejores herramientas para desarrollar sus aplicaciones. Para el desarrollo de las aplicaciones en Texas se siguen los mismos pasos de programación, planteados en el ejemplo 8, el cual se vió en el capítulo III. Se recomienda al lector leer nuevamente este ejemplo.

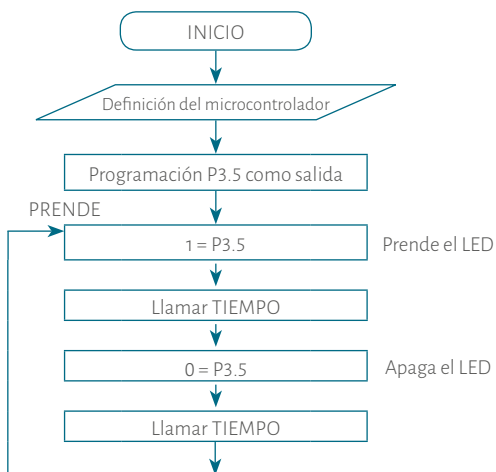
Ejemplo 28. Tomar el ejemplo 8 y acondicionarlo para el microcontrolador MSP430F2272. A continuación se desarrollará paso a paso. Además se explicará cómo hacer un proyecto en CodeComposer.

Paso 1. El funcionamiento de un led ya se conoce por ende se omite este paso.

Paso 2. Hacer un diagrama de conexiones. Hay que decir en cuál de las 32 líneas de In/Out, del microcontrolador se va a poner el led. Se puede tomar P3.5. En los ejemplos desarrollados con microcontroladores Texas Instruments no se harán figuras con los montajes. El lector entenderá que sólo se debe seguir la distribución de pines del microcontrolador, el cual está trabajando. Para este caso, es el MSP430F2272 (Figura 62). De la misma forma, hay que tener en cuenta las líneas de alimentación y de reset.

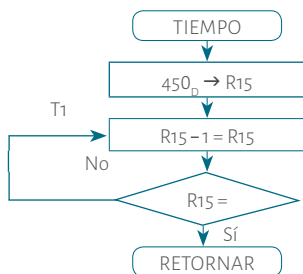
Paso 3. Hacer el diagrama de flujo. De acuerdo al diagrama de conexiones planteado en el punto anterior. Hay que programar P3.5 como salida para el led (ver diagrama 41).

Diagrama 41.
Programa que prende y apaga un led. Ejemplo 28



Para estos microcontroladores la rutina tiempo es más sencilla, pues como se tienen registros de 16 bit con un solo registro se puede hacer una rutina suficiente para ver prender y apagar el led. Es preciso recordar que $2^{16} = 65536$ a diferencia de $2^8 = 256$, ver diagrama 42. Recordar que estos microcontroladores tienen 15 registros que se pueden usar como propósito general. Sin embargo, los R0, R1, R2, R3 ya tienen tareas propias del microcontrolador, ver Figura 64. Por ello, no se recomienda usarlos para propósitos generales. Es mejor usar los registros desde R4. Para esta rutina se usa el R15.

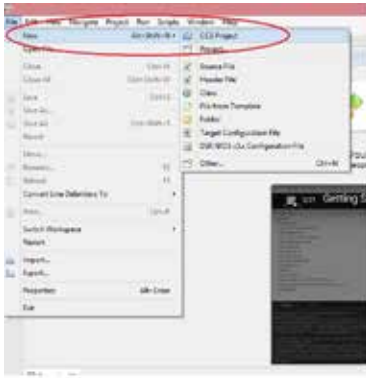
Diagrama 42.
Tiempo del ejemplo 28.



Paso 4. Una vez hecho el diagrama de flujo hay que pasarlo a instrucciones. A continuación se desarrollará el diagrama de flujo bloque por bloque. Pero antes se hace necesario saber cómo se construye proyecto *Code Composer*.

Cuando se construye un nuevo proyecto se hace la definición del microcontrolador, para ello se usa *CodeComposer* versión 6.2 o en otra versión superior, (cuando se inicia *CodeComposer* este le permite seleccionar una carpeta donde guardar los proyectos, se recomienda al lector tener una carpeta para ellos).

Una vez abierto va por la ventana **File/New/CCSProject** como se muestra en la Figura 65A. En la siguiente ventana se debe desplegar y seleccionar el microcontrolador, poner el nombre del proyecto (en este caso Ejemplo 28), seleccionar **Empty Assembler-only Project** y hacer clic en **Finish** (Figura 65B).



65 A



65 B

Figura 65. Crear proyecto en *CodeComposer* 6.2 Paso 1

Fuente. Captura de pantalla *CodeComposer* 6.2 Paso 1

Se ha generado una plantilla en la cual se debe digitar el programa en assembler (el óvalo rojo señala donde escribir el programa), ver figura 66A.

Cuando se abrió el nuevo proyecto se escogió el microcontrolador que es el primer bloque del diagrama de flujo; el siguiente bloque en el diagrama de flujo es programa la línea P3.5 de salida, para ello el lector puede recordar las instrucciones de la tabla 28, y además lo comentado en el apartado 5.3.3, y es para programar una línea de salida se pone en 1, usando el registro P1DIR. La instrucción para poner un UNO en cualquier bit de un registro es: **BIS(B) src,dst**; y para poner un CERO en cualquier bit de un registro es: **BIC(B) src,dst**. La forma como se escribe es:

BIS.B #00i00000B,&P3DIR ; P3.SComo Out

Para el lector debe ser claro que si se quiere otro bit solo debe escribir la combinación del bit. Siguiendo el diagrama de flujo en el siguiente bloque es preciso prender el led, utilizando la misma instrucción pero con el registro P3OUT que es el puerto:

BIS.B #00i00000B,&P3OUT ; Prende LED



66 A



66 B

Figura 66. Crear proyecto en CodeComposer 6.2 Paso 2

Fuente. Captura de pantalla CodeComposer 6.2 Paso 2

Continuado el diagrama se requiere llamar tiempo para ver prendido el led. Para ello, se usa la instrucción CALLdst, así:

CALL #TIEMPO ;Tiempo para ver led prendido

Siguiendo el siguiente diagrama de flujo se requiere apagar el led. Para ello, se usa la misma la instrucción **BIC(B) src,dst,**

BIC.B #00i00000B,&P3OUT ; Apaga LED

Continuado el diagrama ahora se requiere llamar tiempo para ver apagado el led. Para ello, se usa la instrucción CALLdst, ;

CALL #TIEMPO ;Tiempo para ver led apagado

Por último, se quiere volver a prender el led. Para lograrlo, se usa una instrucción de salto incondicional: `JMP Label`. Hay que aclarar que se debe saltar a la etiqueta `PRENDE`:

```
JMP PRENDE ;VA PRENDER EL LED
```

Ahora se requiere hacer un programa de tiempo. El primer paso para hacerlo es cargar el registro `R15`, usando la instrucción `MOV(.B) src,dst`, diagrama de flujo 41:

```
MOV.W #450,R15
```

Para entender un poco más estos procesos se puede observar la Tabla 36, en la cual se muestra cómo se usa esta instrucción para cargar el valor 119 en decimal y los equivalentes en hexadecimal, ASCII y en binario en el registro `R15`.

Tabla 36.
Forma de uso de la instrucción `MOV src,dst`

Forma de uso de la instrucción	Explicación
<code>MOV.W#\$119,R15</code>	Mueve el valor 119 en hexadecimal al registro <code>R15</code>
<code>MOV#77,R15</code>	Mueve el valor 119 en decimal al registro <code>R15</code>
<code>MOV#01110111B,R15</code>	Mueve el valor 01110111 en binario al registro <code>R15</code>
<code>MOV#A',CON1</code>	Mueve el valor ASCII de la letra <code>M</code> al registro <code>R15</code>

Fuente. Elaboración propia.

Siguiendo el diagrama hay que restar en uno el `R15` y preguntar si es cero. Se utilizan las instrucciones: `DEC.W dst;JNZ Label`; la primera resta en uno (`dst-1=dst`) y la segunda salta a una etiqueta cuando aún no es cero:

```
DEC.W R15 ;R15-1=R15  
JNZ L1 ;R15=0
```

Por último, se usa la instrucción `ret` para retornar al programa principal desde la subrutina `TIEMPO` así:

```
RET ;RETORNA AL PROGRAMA PRINCIPAL
```

Ahora que se escribió el programa desarrollando cada una de las partes del diagrama de flujo, se puede observar el programa completo, ver Figura 66B:

```

;
;MSP430 Assembler Code Template for use with TI Code Composer Studio
;
;
;-----
.cdecls C,LIST,"msp430.h" ; Include device header file
;-----
.def RESET ; Export program entry-point to
; make it known to linker.
;-----
.text ; Assemble into program memory.
.retain ; Override ELF conditional linking
; and retain current section.
.retainrefs ; And retain any sections that have
; references to current section.
;-----
RESET mov.w #_STACK_END,SP ; Initialize stackpointer
StopWDT mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog timer
;-----
; Main loop here
;-----
BIS.B #00100000B,&P3DIR ; P3.5 Como Out

PRENDE
BIS.B #00100000B,&P3OUT ; Prende LED
CALL #TIEMPO ; Tiempo para ver led prendido
BIC.B #00100000B,&P3OUT ; Apaga LED
CALL #TIEMPO ; Tiempo para ver led apagado
JMP PRENDE ; VA PRENDER EL LED

TIEMPO
MOV.W #45000,R15

T_1
DEC.W R15 ; R15-1 = R15
JNZ T_1 ; R15=0

RET ;RETORNA AL PROGRAMA PRINCIPAL
;-----
; Stack Pointer definition
;-----
.global _STACK_END

```

```

.sect .stack
;-----
; Interrupt Vectors
;-----
.sect ".reset" ;MSP430 RESET Vector
.short RESET

```

Una vez escrito todo el programa hay que compilar haciendo clic símbolo del martillo, como se muestra en la Figura 67A. Si se presentan errores se pueden corregir haciendo clic en el error y el aplicativo lo remite a la línea donde se presenta dicho error. Previamente es necesario leer qué tipo de error es para poder corregirlo. Una vez corregido, pulsar nuevamente el martillo y repetir el procedimiento hasta que no tenga ningún error (figura 67B).

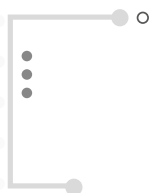


Figura 67. Crear proyecto en *CodeComposer 6.2* Paso 3

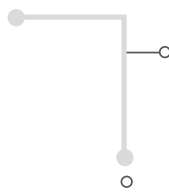
Fuente. Captura de pantalla *CodeComposer 6.2* Paso 3

Paso 5. Programar el microcontrolador, armar el circuito y probar. En este paso hay que tener cuidado cuando se esté cableando en el protoboard. Para ello, hay que conocer la distribución del microcontrolador, ver Figura 62, y el diagrama de conexiones planteado en el paso 2.

Paso 6. Hacer las correcciones pertinentes y volver al paso 3.



Ejercicios propuestos



Realizar los siguientes ejercicios en cada uno de los fabricantes vistos en este capítulo.

1. Se desea que un led aumente su velocidad de encendido y apagado, y que después la disminuya. Es decir, que inicie con la velocidad lenta, vaya hasta una velocidad que el ojo humano no vea y, finalmente, que regrese de la misma forma a la velocidad en la que inició.
2. Hacer un contador en binario de 00_H a $4C_H$ que inicie nuevamente.
3. Hacer un juego de luces que se mueva 3 veces de izquierda a derecha y 5 veces de derecha a izquierda.
4. Se tienen 3 matrices unicolor 7×5 y se desea visualizar la palabra MICROCONTROLADORES en los 3 colores posibles (rojo, verde y ámbar).
5. Hay un concurso de conocimientos en electrónica básica organizado por la facultad de Ingeniería de la Universidad Piloto de Colombia. El concurso tiene cinco participantes. La misión es diseñar el circuito de manera que al momento de hacer una pregunta, alguno de los participantes pueda oprimir un botón, inmediatamente, un bombillo de 110 VAC indique cuál de los participantes desea contestar. En ese momento se deben deshabilitar los botones de los otros participantes. Además se debe contar con un sistema de reset y con un led indicador.
6. Hay cinco motores $12V_{DC}$, cada uno tiene un pulsador con el que se apagan o prenden. Inicialmente, los cinco motores están prendidos. Si se desea apagar alguno de ellos, por ejemplo el motor dos, se oprime el pulsador dos. En este caso los otros motores siguen prendidos. Si se

- quiere volver a prender se oprime de nuevo el pulsador. Esto aplica para todos los motores.
- Hay una tejedora que tiene un motor de $12V_{DC}$, el cual mueve un cabezal. El motor tiene dos microswitch; uno a la izquierda y otro a la derecha. Se desea que el cabezal vaya a la derecha y a la izquierda. Es decir, se debe simular el movimiento de coser. Hay un led indicador que funciona permanentemente.
 - Diseñar un programa que controle un motor paso a paso de 1.8 grados por paso así: Hay un pulsador que una vez oprimido hace girar al motor así: El motor gira 281° a la derecha, luego dos vueltas a la izquierda, luego una vuelta a la derecha y por último 90° a la izquierda. Al final el motor se detiene. Hay un led de testigo. Cuando se vuelve a oprimir el pulsador el proceso inicia nuevamente.
 - En el ejemplo 20 se trabajó con un motor paso a paso de 4 bobinas, de 1.8 grados por paso, de 5Vdc y con 4 pulsadores ubicados en cruz, ver Figura 44. Se desea que una LCD visualice, en la línea uno, los grados que gira el motor y, en la línea dos, el sentido de giro, derecha o izquierda.
 - Se desea diseñar el control de un semáforo en una intersección (calle/carrera) así: la prioridad la tiene la calle; en la carrera estarán ubicados sensores para detectar el flujo cuando pasen 15 vehículos en cada uno de los sentidos (norte/sur o sur/norte); el semáforo debe dar la vía a la carrera durante 50 segundos. Se deben tener leds indicadores para los 3 colores del semáforo, tanto en la calle como en la carrera.
 - Agregar un semáforo peatonal al ejercicio anterior. Este semáforo debe ser visualizado en una matriz bicolor así: cuando el peatón no debe caminar el semáforo mostrará una silueta de una persona en rojo quieta; cuando el peatón puede caminar la figura se pone de color verde caminado (animación); cuando el tiempo se esté terminado, la figura debe caminar más rápido y ponerse de color amarillo (mezcla de rojo y verde) y cuando el tiempo se acabe debe pasar a rojo y debe estar quieto.



Referencias Bibliográficas

- ATMEL[®]. 8-bit AVR[®]. (2016). *Microcontroller with 2K Bytes In-System Programmable Flash. Attiny2313/V*. http://ww1.microchip.com/downloads/en/devicedoc/atmel-2543-avr-attiny2313_datasheet.pdf
- ATMEL[®]. 8-bit AVR[®]. (2016a). *Microcontroller Atmega48PA/88Pa/168PA. Datasheet complete*. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42734-8-bit-AVR-Microcontroller-ATmega48PA-88PA-168PA_Datasheet.pdf
- ATMEL[®]. 8-bit AVR[®]. (2016b). *Microcontrollers. ATmega328/P. Datasheet complete*. http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf
- Camargo , G. (2009). *Programación de sistemas embebidos en C*. Bogotá D.C., Colombia: AlfaOmega.
- Floyd, T. (2006). *Fundamentos de sistemas digitales*. Madrid, España: Pentrice-Hall.
- Galeano, . (2009). *Programación de sistemas embebidos en C*. Bogotá D.C., Colombia: Alfaomega.
- Mandado, E., Méndez, L., Ferreira, L. y Matos, E. (2007) *Microcontroladores PIC. Sistema Integrado Para el Autoaprendizaje*. Barcelona, España: Marcombo.
- MICROCHIP. Technology Inc, (2001). *PIC16F84A Data Sheet 18-pin Enhanced FLASH/EEPROM 8-bit Microcontroller*. <http://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf>
- MICROCHIP. Technology Inc. (2007a). *PIC16F627A/628A/648A Data Sheet Flash-Based, 8-Bit CMOS Microcontrollers with nanoWatt Technology*. <http://ww1.microchip.com/downloads/en/DeviceDoc/40044F.pdf>

- MICROCHIP .Technology Inc. (2007b). *PIC16F87XA Data Sheet 28/40/44-Pin Enhanced Flash Microcontrollers*. <http://ww1.microchip.com/downloads/en/DeviceDoc/39582b.pdf>
- MICROCHIP. Technology Inc. (2017c). *AVR 8-Bit Microcontroller. Atmega8A Data Sheet*. de: <http://ww1.microchip.com/downloads/en/DeviceDoc/Microchip%208bit%20mcu%20AVR%20ATmega8A%20data%20sheet%2040001974A.pdf>
- NXP-Freescale. (2009). *MC9S08JS16RM.. HCS08 Microncontrollers. Data Sheet*, (4), pp. xx-xx, <https://www.nxp.com/docs/en/reference-manual/MC9S08JS16RM.pdf>
- NXP-Freescale. (2004). *MC9S08GB60/D.. HCS08 Microncontrollers. Data Sheet*, (4) <https://www.nxp.com/docs/en/data-sheet/MC9S08GB60.pdf>
- Palacios, E., Remiro, F y López, L. (2006) *Microcontrolador PIC 16F84A. Desarrollo de Proyectos* (2 Ed.),. Madrid, España: Alfa y Omega.
- Ritterman, S. (1988) *Circuitos de computadora*. México D.F, México: McGraaw Hill.
- TEXAS INSTRUMENTS. (2002-2004). *MSP430x11x2, MSP430x12x2. MIXED SIGNAL Microncontroller*. Recuperado de <http://www.ti.com/lit/ds/symlink/msp430f1132.pdf>
- TEXAS INSTRUMENTS. (2012). *MSP430F22x2, MSP430F22x4. MIXED SIGNAL Microncontroller*. <http://www.ti.com/lit/ds/symlink/msp430f2272.pdf>
- TEXAS INSTRUMENTS. (2013). <http://www.ti.com/lit/ug/slau144j/slau144j.pdf>
- Thymoty, Manoley. (1996). *Electrónica Industrial Moderna* (3 Ed.). Mexico D.F., México: Pirson.
- Uyemura, J. (2000). *Diseño de sistemas digitales*. México D.F, México: International Thomson.
- Vega, J. (2007). *Microcontroladores Motorola Freescale. Programacion, familias y sus distintas aplicaciones*. Bogotá D.C, Colombia: AlfaOmega.
- XIAMEN AMOTEC DISPLAY. CO. LTD. (2008). *SPECIFICATIONS OF LCD MODULE. MODULE NO: ADM1602K-NSW-FBS/3.3V. DOC.REVISION: OD.*: <https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf>