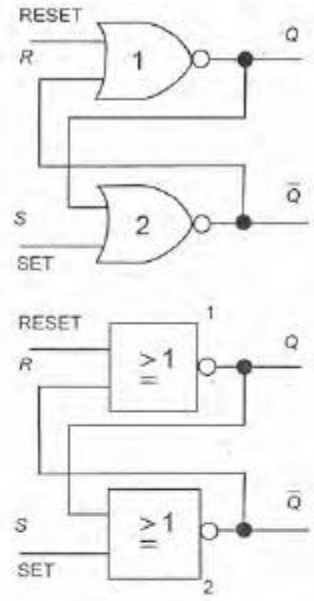


Figura 14.35 Flip–flop SR

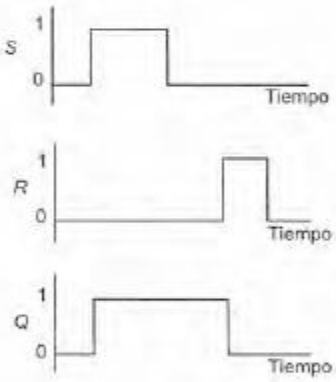


Figura 14.36 Diagrama de tiempo

Figura 14.37 Flip–flop SR

14.7.1 El flip flop

El *flip flop* es un elemento de memoria básico que consta de un conjunto de compuertas lógicas. Existen diversos tipos de flip–flops. La figura 14.35 muestra una forma, el flip–flop SR (set–reset), que tiene compuertas NOR. Si inicialmente se tienen ambas salidas $Q = 0, S = 0$ y $R = 0$, entonces al hacer que S cambie de 0 a 1, la salida de la compuerta NOR 2 será 0. Esto produce que ambas entradas a la compuerta NOR 1 se conviertan en 0 y su salida se vuelva 1. Esta realimentación actúa como entrada de la compuerta NOR 2, en la que ambas entradas son igual a 1 y al final no se produce otro cambio.

Si S cambia de 1 a 0, la salida de la compuerta NOR 1 sigue siendo 1 y la salida de la compuerta NOR 2 permanece en 0. No hay cambio en las salidas cuando la entrada S cambia de 1 a 0. Permanecerá en este estado en forma indefinida si los únicos cambios que se producen son en S. Es capaz de "recordar" el estado al que fue establecido. La figura 14.36 ilustra lo anterior con un diagrama de tiempos, en el que un impulso rectangular se utiliza como la entrada S.

Si R cambia de 0 a 1 cuando S es 0, la salida de la compuerta NOR 1 se convierte en 0 y, por lo tanto, la salida de la compuerta NOR 2 cambia a 1, es decir, el flip–flop se reinicia. Un cambio de R a 0 no tiene efecto en estas salidas.

Así, cuando el valor de S es 1 y R se hace 0, la salida Q cambia a 1 si su valor anterior fue 0, y seguirá siendo 1 si antes fue 1. Ésta es la condición de inicio y permanecerá sin cambio aun cuando S cambie a 0. Cuando S es 0 y R se hace 1, la salida Q se ajusta a 0, si su valor anterior fue 1, o sigue siendo 0 si antes fue 0. Ésta es la condición de reposo. La salida Q que se produce en un instante determinado dependerá de las entradas S y R y también del último valor de la salida. La siguiente tabla de estado ilustra lo anterior:

S	R	$Q_t \rightarrow Q_{t+1}$	$\overline{Q}_t \rightarrow \overline{Q}_{t+1}$
0	0	$0 \rightarrow 0$	$1 \rightarrow 1$
0	0	$1 \rightarrow 1$	$0 \rightarrow 0$
0	1	$0 \rightarrow 0$	$1 \rightarrow 1$
0	1	$1 \rightarrow 0$	$0 \rightarrow 0$
1	0	$0 \rightarrow 1$	$1 \rightarrow 0$
1	0	$1 \rightarrow 1$	$0 \rightarrow 0$
1	1		No se permite
1	1		No se permite

Observe que si S y R se hacen 1 al mismo tiempo, no existe la posibilidad de que haya un estado estable, por lo que esta condición de entrada no se permite. La figura 14.37 muestra el símbolo de bloques simplificado que representa al flip–flop SR.

Un ejemplo sencillo de la aplicación de un flip–flop es un sistema de alarma simple, en el que la alarma suena cuando se obstruye el

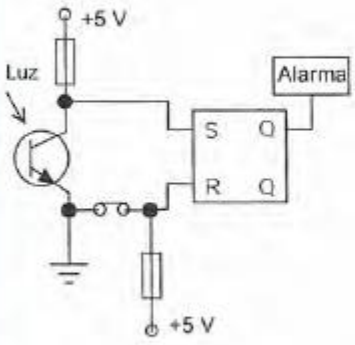


Figura 14.38 Circuito de una alarma

paso de un haz luminoso; la alarma sigue sonando aun cuando ya no se interrumpa el paso de luz. La figura 14.38 muestra este sistema. Se puede usar como sensor un fototransistor configurado de manera que cuando se ilumina produce una entrada a S prácticamente de 0 V, pero cuando la iluminación se interrumpe produce 5 V de entrada en S. Cuando el haz luminoso se interrumpe S se convierte en 1 y la salida del flip–flop se convierte en 1, y suena la alarma. La salida sigue siendo 1 aun cuando S cambie a 0. La desactivación de la alarma se logra sólo si el interruptor de ajuste se abre en forma momentánea para producir 5 V de entrada en R.

14.7.2 Sistemas síncronos

Con frecuencia es necesario definir el ajuste y reinicio de operaciones que deben ocurrir en tiempos específicos. En un sistema no temporizado o *sistema asíncrono*, las salidas de las compuertas lógicas cambian su estado cada vez que una o varias entradas cambian. En un sistema temporizado o *sistema síncrono*, los tiempos exactos en los que alguna de las salidas cambia su estado están determinados por una señal de temporización o señal de reloj. Ésta es en general un tren de pulsos rectangulares y cuando se usa la misma señal de reloj en todas las partes del sistema, las salidas están sincronizadas. La figura 14.39 muestra este principio con compuertas en un flip–flop SR. La señal de ajuste y de reloj se suministran a través de una compuerta AND en la entrada S del flip–flop. Así, la señal de ajuste llega al flip flop sólo cuando la señal de ajuste y la de reloj tienen valor 1. Asimismo, la señal de reinicio junto con la señal de reloj entran a R a través de otra compuerta AND. En consecuencia, el ajuste y el reinicio sólo pueden ocurrir en el momento definido por el temporizador. La figura 14.40 muestra el diagrama de tiempo.

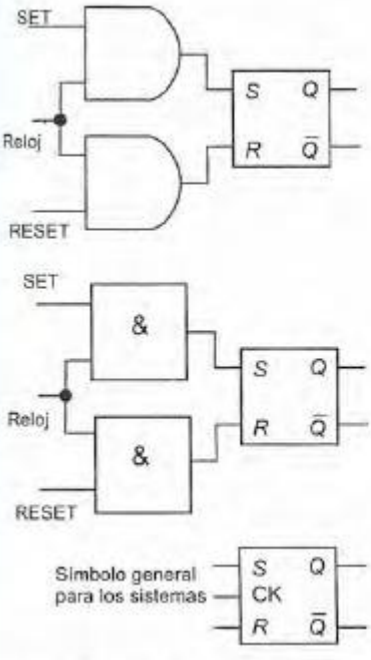


Figura 14.39 Flip–flop de RS con reloj

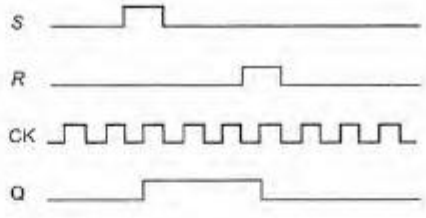


Figura 14.40 Diagrama de tiempo

14.7.3 El flip flop tipo JK

Para muchas aplicaciones no es aceptable el estado de indeterminación que se presenta con un flip–flop SR cuando $S = 1$ y $R = 1$; por ello se emplea otro tipo de flip–flop: el *flip flop JK* (figura 14.41). Éste es un dispositivo biestable que se utiliza mucho. La siguiente es la tabla de verdad de este dispositivo; observe que los únicos cambios de la tabla de estado del flip–flop SR son las líneas cuando ambas entradas son 1.

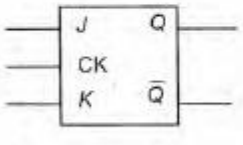


Figura 14.41 Flip–flop tipo JK

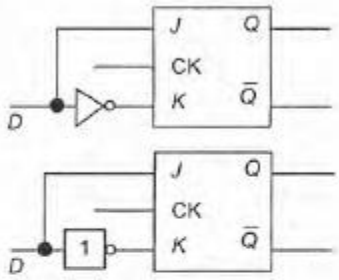


Figura 14.42 Flip–flop tipo D

J	K	$Q_t \rightarrow Q_{t+1}$	$\overline{Q}_t \rightarrow \overline{Q}_{t+1}$
0	0	0 → 0	1 → 1
0	0	1 → 1	0 → 0
0	1	0 → 0	1 → 1
0	1	1 → 0	0 → 0
1	0	0 → 1	1 → 0
1	0	1 → 1	0 → 0
1	1	0 → 1	1 → 0
1	1	1 → 0	0 → 1

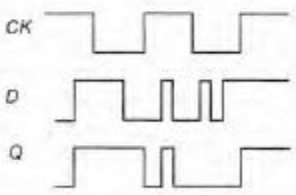


Figura 14.43 Salida de flip–flop tipo D

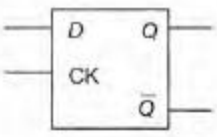


Figura 14.44 Símbolo de flip–flop tipo D

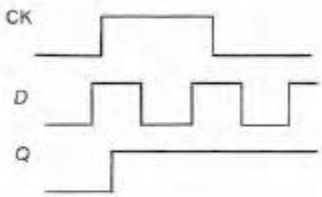


Figura 14.45 Activación por flanco de subida

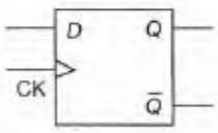


Figura 14.46 Símbolo de un flip–flop D de activación por flanco

Un ejemplo de aplicación de este flip–flop es la necesidad de obtener una salida con valor alto cuando la entrada *A* aumenta y después de cierto tiempo la entrada *B* aumenta. Para determinar si ambas entradas son altas se puede emplear una compuerta AND; sin embargo, su salida será alta sin importar cuál fue la entrada que aumentó primero. No obstante, si las entradas *A* y *B* se conectan a un flip–flop JK, *A* debe aumentar primero para que la salida aumente cuando *B* también aumente.

14.7.4 El flip flop tipo D

El biestable de datos o *flip flop D* es de hecho un flip-flop SR con reloj, o un flip–flop JK cuya entrada *D* se conecta de manera directa a las entradas *S* o *J* y a través de una compuerta NOT a las entradas *R* o *K* (figura 14.42); en el símbolo del flip–flop tipo D, la entrada combinada *R* y *K*, se denominan *D*. De esta manera, una entrada de 0 o de 1 conmuta la salida de manera que siga a la entrada D cuando el pulso del reloj sea 1 (figura 14.43). Una aplicación particular del flip–flop tipo D es garantizar que la salida sólo tome el valor de la entrada *D* en tiempos definidos con precisión. La figura 14.44 muestra el símbolo utilizado para un flip–flop D.

En el flip–flop tipo D, cuando la entrada del reloj o la de activación aumenta, la salida sigue a los datos presentados en la entrada *D*. Se dice que el flip–flop es transparente. Cuando se presenta una transición de alto a bajo en la entrada de activación, la salida *Q* se mantiene al nivel de datos justo anterior a la transición. Se dice que los datos en el punto de transición están *latched* (enclavado). Existen circuitos integrados de flip–flops D. Un ejemplo es el 7475, que contiene cuatro *latched* transparentes D.

La diferencia entre el flip–flop 7474 D y el 7475 consiste en que el primero es un dispositivo de activación por flanco; en el paquete hay dos de estos flip–flops. En un flip–flop D de activación por flanco, las transiciones de *Q* sólo ocurren en el flanco de entrada del pulso de reloj y en el 7474 en el flanco positivo, es decir, de la transición del nivel bajo al alto. La figura 14.45 ilustra lo anterior. La diferencia entre el símbolo básico de un flip–flop D de activación por flanco y el de un flip–flop D es el pequeño triángulo que se coloca en la entrada del reloj CK (figura 14.46). También hay otras dos entradas de-

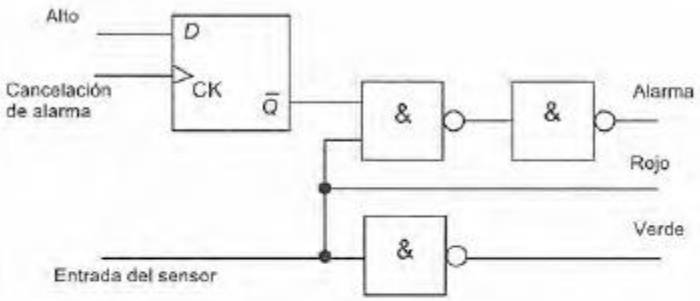


Figura 14.47 Sistema de alarma

nominadas PRESET y CLEAR. Un valor bajo en la entrada del PRESET define la salida Q igual a 1, en tanto que un valor bajo en la entrada del CLEAR borra la salida y hace Q igual a 0.

Un ejemplo de una aplicación sencilla de este flip–flop se ve en la figura 14.47, la cual muestra un sistema que sirve para que aparezca una luz verde cuando la entrada del sensor es baja y una luz roja cuando la entrada aumenta y hace sonar una alarma. La luz roja deberá permanecer encendida en tanto la entrada del sensor siga siendo alta, pero la alarma se puede apagar. Éste podría ser el sistema para monitorear la temperatura de un proceso; el sensor y el trasductor de señal producen una señal baja cuando la temperatura es inferior al nivel de seguridad y una señal alta cuando es superior a ese nivel. El flip–flop tiene una entrada alta. Cuando se aplica una entrada baja a la entrada CK y la entrada del sensor es baja, se enciende la luz verde. Cuando la entrada del sensor se vuelve alta, se apaga la luz verde, se enciende la roja y suena la alarma. Para eliminar la alarma se aplica una señal alta en la entrada CK, pero la luz roja permanece encendida mientras la entrada del sensor sea alta. Para construir este sistema se puede emplear un 7474 y un circuito o circuitos integrados que contengan tres compuertas NAND.

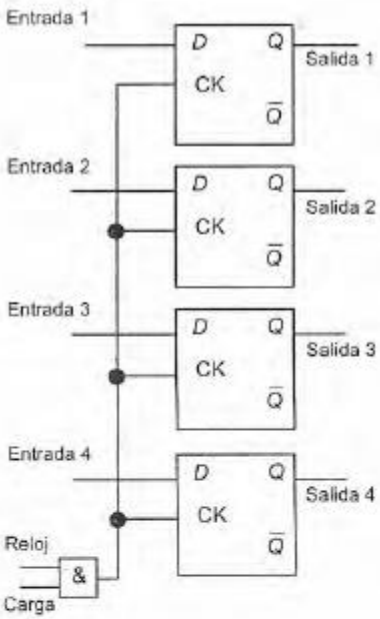


Figura 14.48 Registro

14.7.5 Registros

Un *registro* es un conjunto de elementos de la memoria que sirve para guardar información hasta que se requiera. Se puede construir mediante flip–flops. Cada flip–flop guarda una señal binaria, es decir, un 0 o un 1. La figura 14.48 muestra la configuración de un registro de 4 bits cuando se utilizan flip–flops tipo D. Cuando la señal de carga es 0, no se presenta ninguna entrada de reloj en los flip–flops D, por lo que no hay cambio en los estados de los flip–flops. Cuando la señal de carga es 1, las entradas pueden modificar los estados de los flip–flops. Mientras la señal de carga sea 0, los flip–flops conservarán los valores de sus estados anteriores.

Problemas

1. ¿Cuál es el mayor número decimal que se puede representar utilizando un número binario de 8 bits?
2. Convierta los siguientes números binarios en números decimales: a) 1011 y b) 10 0001 0001.

3. Convierta los números decimales a) 423 y b) 529 en hexadecimales.
4. Convierta los números BCD a) 0111 1000 0001 y b) 0001 0101 0111 en decimales.
5. ¿Cuáles son las representaciones de complemento a dos, de los números decimales a) –90 y b) –35?
6. ¿Cuáles son los bits de paridad par que es necesario añadir a a) 100 1000 y b) 100 1111?
7. Reste los siguientes números decimales utilizando el complemento a dos: a) 21–13 y b) 15–3.
8. Explique qué compuertas lógicas deberán emplearse para controlar las siguientes situaciones:
   a) La venta de boletos en una máquina automática de una estación de trenes.
   b) Un sistema protector de seguridad para la operación de una máquina herramienta.
9. Indique cuáles son las funciones booleanas que permiten describir las siguientes situaciones:
   a) Se produce una salida cuando se cierra el interruptor A y se cierra el interruptor B o el interruptor C.
   b) Se produce una salida cuando el interruptor A o el interruptor B se cierran y el interruptor C o el D están cerrados.
   c) Se produce una salida cuando el interruptor A se abre o bien cuando el interruptor B se cierra.
   d) Existe una salida cuando el interruptor A está abierto y el interruptor B está cerrado.
10. Indique cuáles son las funciones booleanas de los circuitos lógicos que se muestran en la figura 14.49.

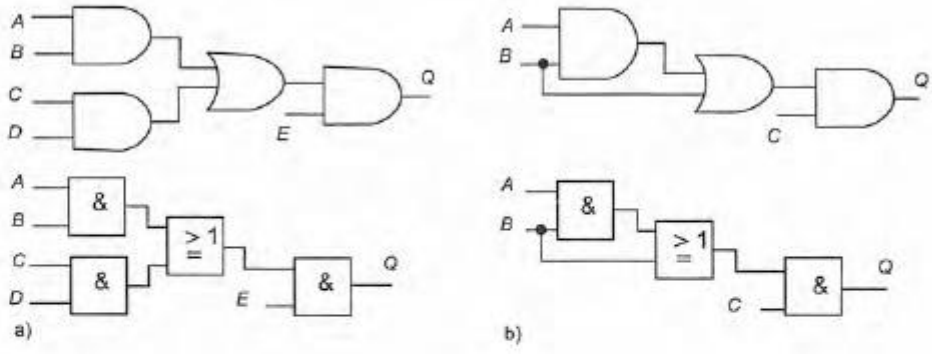


Figura 14.49
Problema 10

11. Elabore la tabla de verdad de la ecuación booleana
    $Q = (A \cdot C + B \cdot C) \cdot (A + C)$
12. Simplifique las siguientes ecuaciones booleanas:
    a) $Q = A \cdot \overline{C} + A \cdot C \cdot \overline{D} + \overline{C} \cdot D$
    b) $Q = A \cdot \overline{B} \cdot D + A \cdot \overline{B} \cdot \overline{D}$
    c) $Q = A \cdot B \cdot C + C \cdot D + C \cdot D \cdot E$
13. Utilice las leyes de De Morgan para mostrar que una compuerta NOR con entradas invertidas equivale a una compuerta AND.

14. Dibuje los mapas de Karnaugh de las siguientes tablas de verdad y determine cuál es la ecuación booleana simplificada para las salidas.

a)

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	1

b)

A	B	C	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

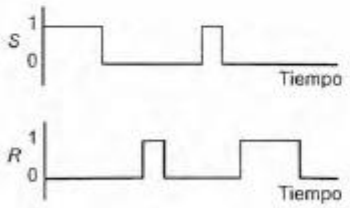


Figura 14.50 Problema 17

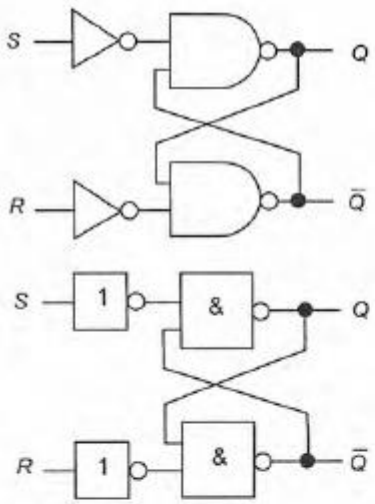


Figura 14.51 Problema 18

15. Simplifique las siguientes ecuaciones booleanas utilizando mapas de Karnaugh:
    a) $Q = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C}$
    b) $Q = \overline{A} \cdot B \cdot \overline{C} \cdot D + A \cdot \overline{B} \cdot \overline{C} \cdot D + \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot D + A \cdot B \cdot \overline{C} \cdot D + A \cdot B \cdot \overline{C} \cdot \overline{D} + A \cdot B \cdot C \cdot D$
16. Diseñe un sistema que permita la apertura de una puerta sólo cuando se oprime la combinación correcta de cuatro botones; cualquier otra combinación activa una alarma.
17. La figura 14.50 muestra el diagrama de tiempo de las entradas S y R de un flip–flop tipo RS. Complete el diagrama añadiendo la salida Q.
18. Explique cómo obtener un flip–flop tipo RS con base en la configuración de la figura 14.51.

15 Microprocesadores

15.1 Control

Si consideramos un problema de control sencillo, como la secuencia de las luces roja, amarilla y verde del semáforo de un crucero, basta recurrir a un sistema de control electrónico que contenga circuitos integrados de lógica combinacional y de lógica secuencial. Sin embargo, en situaciones más complejas se deben controlar muchas más variables pues la secuencia de control es más complicada. La solución más sencilla en este caso no es construir un sistema basado en la interconexión de circuitos integrados de lógica combinacional y secuencial, sino en el uso de un microprocesador para que el software realice las "interconexiones".

Los sistemas de microprocesadores que se estudian en este libro son los que se usan como sistemas de contol y se llaman *microprocesadores embebidos*. Esto se debe a que el microprocesador está dedicado a controlar una función específica y arranca por sí mismo sin requerir la intervención humana, y está totalmente autocontenido con sus propios programas de operación. Para los humanos, no es aparente que el sistema sea de microprocesador. Así, una lavadora de ropa moderna contiene un microprocesador y todo lo que el operador debe hacer para que funcione es seleccionar qué tipo de lavado requiere oprimiendo los botones apropiados o girando un selector y luego oprimiendo el botón de arranque.

Este capítulo presenta un panorama general de la estructura de los microprocesadores y los microcontroladores; en los dos siguientes capítulos se estudia la programación y en el capítulo 17 las interfases. Si el lector desea profundizar en el tema, puede consultar las publicaciones de los fabricantes de microprocesadores, o libros como: *Microprocessor Systems* de W. Bolton (Longman, 2000) y sobre microcontroladores en especial *Software and Hardware Engineering, Motorola M68HC11* de F.M. Cady (Oxford University Press, 1997), *Microcontroller Technology: The 68HC11* de P. Spasov (Prentice-Hall, 1992, 1996), *The 8051 Family of Microcontrollers* de R.H. Barnett (Prentice-Hall, 1995), *PIC: Your Personal Introductory*

Course de J. Morton (Newness, 2001) o bien *PIC Basic: Programming and Projects* de I. Ibrahim (Newness, 2001).

15.2 Sistemas microprocesadores

Los sistemas microprocesadores constan de tres partes:

1. La *unidad central de proceso* (CPU, *central processing unit*), la cual reconoce y ejecuta las instrucciones de un programa. Ésta es la parte que usa el microprocesador.
2. Las *interfases de entrada y salida*, para manejar las comunicaciones entre la computadora y el mundo exterior; el término *puerto* se usa para la interfase.
3. La *memoria*, donde se almacenan instrucciones de programas y datos.

Los microprocesadores que contienen memoria y varios arreglos de entrada y salida en un mismo chip se llaman *microcontroladores*.

15.2.1 Buses

Las señales digitales se desplazan de una sección a otra a través de vías llamadas *buses*. En sentido físico, el bus consta de varios conductores a través de los cuales se transportan diversas señales eléctricas. Éstos pueden ser las pistas de una tarjeta de circuito impreso, o los alambres de un cable plano. La figura 15.1 ilustra la configuración general de un sistema microprocesador y sus buses. Hay tres formas de bus en un sistema microprocesador.

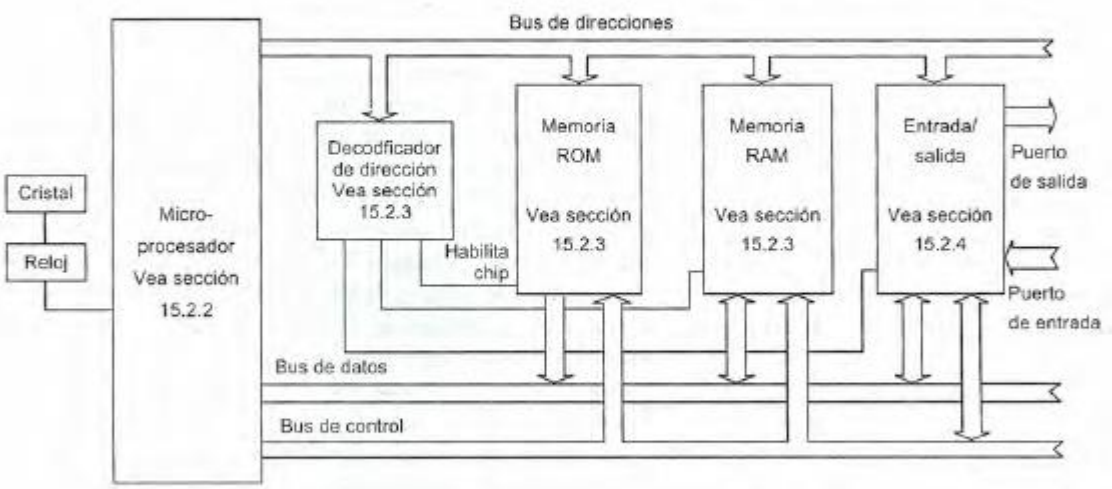


Figura 15.1 Configuración general de un sistema microprocesador

1. *Bus de datos*
   Los datos asociados con las funciones de procesamiento de la CPU fluyen a través del *bus de datos*. De esta manera se utiliza para transportar palabras hacia o desde la CPU y la memoria o las interfases de entrada/salida. La longitud de las palabras puede ser de 4, 8, 16, 32 o 64 bits. En cada línea del bus viaja una señal binaria, es decir, un 0 o un 1. Así, en un bus de cuatro líneas se podría transportar la palabra 1010; en cada cable se transporta un bit, es decir:

Palabra	Línea del bus
0 (bit menos significativo)	Primer línea del bus de datos
1	Segunda línea del bus de datos
0	Tercer línea del bus de datos
1 (bit más significativo)	Cuarta línea del bus de datos

   Entre más líneas tenga el bus de datos, más larga podrá ser la palabra que se utilice. El intervalo de valores que puede adoptar un elemento de datos está restringido al espacio correspondiente a cierta longitud de palabra. Así, para una palabra con longitud de 4 bits, la cantidad de valores es $2^4 = 16$. Supongamos que mediante estos datos se desea representar una temperatura, entonces el intervalo de temperaturas posibles se divide en 16 segmentos suponiendo que el intervalo se representa por una palabra de 4 bits. Los primeros microprocesadores eran dispositivos de 4 bits (longitud de palabra), y todavía se emplean mucho en dispositivos como juguetes, lavadoras y controladores de calefacción central doméstica. Después aparecieron los microprocesadores de 8 bits, por ejemplo, el Motorola 6800, el Intel 8085A y el Zilog Z80. En la actualidad existen microprocesadores de 16, 32 y 64 bits; sin embargo, los microprocesadores de 8 bits aún se utilizan mucho en controladores.

2. *Bus de direcciones*
   El *bus de direcciones* transporta señales que indican dónde se pueden encontrar los datos y hace la selección de alguna localidad de memoria o los puertos de entrada y salida. Cada localidad en la memoria tiene una identificación única, denominada "dirección", de modo que los sistemas son capaces de seleccionar una instrucción o datos específicos en la memoria. Cada interfase entrada/salida tiene también una dirección. Cuando una dirección dada se selecciona, colocándola en el bus de direcciones, dicha localidad será la única que estará abierta a la comunicación que se envía desde la CPU. Es decir, la CPU sólo puede comunicarse con una localidad a la vez. Una computadora con un bus de datos de 8 bits tiene un bus de direcciones de 16 bits, es decir, 16 líneas. La magnitud del bus de direcciones permite 2^{16} localidades direccionadas. La cantidad de 2^{16} corresponde a 65 536 localidades y en general se expresa como 64 K, donde K es

igual a 1024. Entre más memoria direccionable haya, mayor es la cantidad de datos que es posible guardar, así como mayor y más complejo el programa que se puede utilizar.

3. *Bus de control*
   Las señales referentes a las acciones de control se transportan en el *bus de control*. Por ejemplo, es necesario que el microprocesador informe a los dispositivos de memoria si se están leyendo datos de un dispositivo de entrada o se están escribiendo datos a un dispositivo de salida. El término READ se usa para recibir señales y WRITE para enviarlas. El bus de control también se usa para transportar las señales de reloj del sistema que deben sincronizar todas las acciones del sistema microprocesador. El reloj es un oscilador controlado por cristal y produce pulsos de periodos regulares.

15.2.2 El microprocesador

En general se hace referencia al microprocesador como la *unidad de procesamiento central* (CPU, *central processing unit*). Esta es la parte del procesador en la que se procesan los datos, se traen instrucciones de la memoria que se decodifican y se ejecutan. La estructura interna, conocida como *arquitectura*, de un microprocesador depende del microprocesador que se esté considerando. La figura 15.2 indica, en forma simplificada, la arquitectura general de un microprocesador.

Las siguientes son las funciones de las partes que forman a un microprocesador.

1. *Unidad lógica y aritmética* (ALU, *arithmetic and logic unit*)
   La unidad lógica y aritmética es la responsable de llevar a cabo la manipulación de los datos.

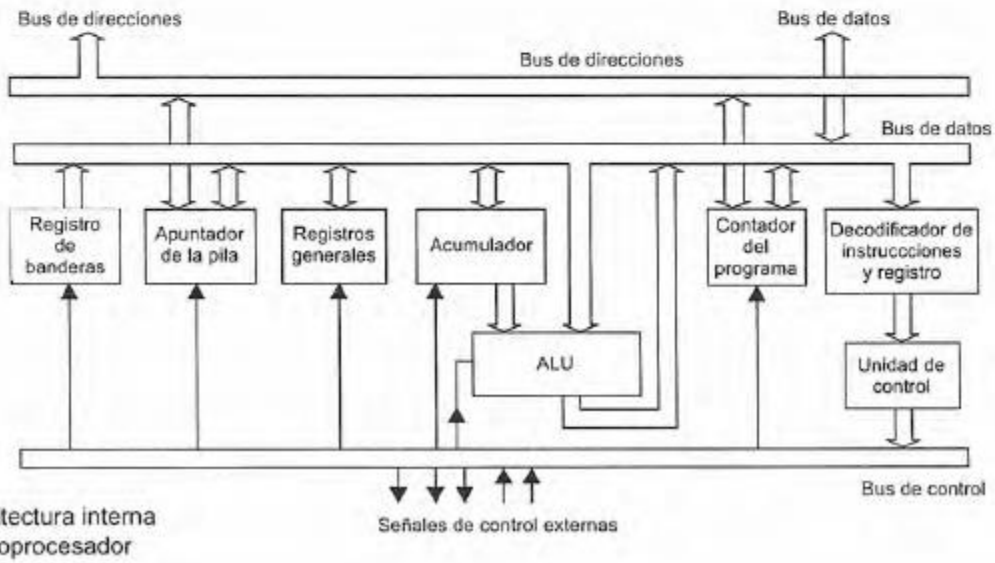


Figura 15.2 Arquitectura interna general de un microprocesador

2. *Registros*
   Los datos internos que la CPU suele utilizar se mantienen temporalmente en un grupo de *registros* mientras se ejecutan las instrucciones. Éstos son localidades de memoria dentro del microprocesador y se usan para almacenar información involucrada en la ejecución de un programa. Un microprocesador contendrá un grupo de registros, cada tipo de registro tiene una función diferente.

3. *Unidad de control*
   La *unidad de control* determina la temporización y secuencia de las operaciones. Ésta genera señales de temporización utilizadas para traer de la memoria una instrucción del programa y ejecutarla. El 6800 de Motorola utiliza un reloj con frecuencia máxima de 1 MHz, es decir, un periodo de reloj de 1 μs; y las instrucciones requieren entre 2 y 12 ciclos de reloj. Las operaciones pertenecientes a los microprocesadores se reconocen por la cantidad de ciclos que se requieren para ejecutarlas.

Existen diversos tipos de registros; la cantidad, dimensión y tipo de los registros varía de un microprocesador a otro. Los siguientes son los registros más comunes:

1. *Registro acumulador*
   El registro denominado acumulador (A o Acc) es donde se guardan los resultados de la unidad lógica y aritmética temporalmente. Para que la CPU pueda habilitar el acceso, es decir, usar las instrucciones o datos guardados en la memoria, es necesario que proporcione la dirección de memoria del dato requerido, utilizando el bus de direcciones. Una vez hecho lo anterior, la CPU podrá usar las instrucciones o datos necesarios por el bus de datos. Dado que sólo es posible leer de una localidad de memoria a la vez, es necesario recurrir a un almacenamiento temporal cuando, digamos, se combinan números. Por ejemplo, al sumar dos números, uno de ellos se trae de una dirección y se deja en el acumulador mientras que la CPU trae el otro número de otra dirección de memoria. A partir de este momento, la unidad lógica y aritmética de la CPU puede operar ambos números. El resultado se transfiere al acumulador. Éste, por lo tanto, es un registro de retención temporal para permitir que la unidad lógica y aritmética haga operaciones con los datos y, una vez terminadas las operaciones, el registro retenga los resultados. Por ello, participa en todas las transferencias de datos asociadas con la ejecución de operaciones aritméticas y lógicas.

2. *Registro de estado*, o *registro de código de condición* o *registro de banderas*
   Este registro contiene información relacionada con el resultado de la última operación realizada en la unidad lógica y aritmética. El registro contiene bits individuales, los cuales tienen un significado especial. Estos bits se conocen como *banderas*. El estado de la última operación se indica con cada bandera que se ajusta o

se restablece, según sea el caso, para indicar un estado específico. Por ejemplo, para indicar si el resultado de la última operación es negativo, es cero, si hay acarreo (por ejemplo, el resultado de la suma de los números binarios 1010 y 1100 es (1)0110, que podría ser mayor que el tamaño de la palabra del microprocesador, por lo que se acarrea un 1 de sobreflujo), si hay desbordamiento, o si existe la posibilidad de interrumpir el programa para permitir que ocurra un evento externo. Las siguientes son las banderas más comunes:

Bandera	Ajuste, es decir, 1	Restablecimiento, es decir, 0
Z	el resultado es cero	el resultado no es cero
N	el resultado es negativo	el resultado no es negativo
C	se genera acarreo	no se genera acarreo
V	se produce desbordamiento	no se produce desbordamiento
I	se ignora la interrupción	la interrupción se procesa de manera normal

A manera de ilustración, considere el estado de las banderas Z, N, C y V para la operación de suma de los números hexadecimales 02 y 06. El resultado es 08. Como no es cero entonces Z es 0. El resultado es positivo, de modo que N es 0. No hay acarreo, de modo que C es 0. El resultado sin signo está en el intervalo –128 a +127 y no hay desbandamiento así que V es 0. Ahora considere las banderas cuando los números hexadecimales sumados son F9 y 08, el resultado es (1)01. El resultado no es cero, así Z es 0. Como es positivo N es 0. El resultado sin signo tiene acarreo y C es 1. El resultado sin signo está en el intervalo –128 a +127 y entonces V es 0.

3. *Contador del programa* (PC, *program counter*) o *apuntador de instrucciones* (IP, *instruction pointer*)
Mediante este registro la CPU controla su posición en un programa. En este registro contiene la dirección de la localidad de memoria que tiene la siguiente instrucción del programa. Cada vez que se ejecuta una instrucción, el registro contador del programa se actualiza de forma que siempre contiene la dirección de la localidad de memoria donde está almacenada la siguiente instrucción que se va a ejecutar. El contador del programa se incrementa cada vez para que la CPU ejecute las instrucciones en secuencia, a menos que una instrucción, como JUMP (salto) o BRANCH (ramificación) la cambie.

4. *Registro de direccionamiento de memoria* (MAR, *memory address register*)
Éste contiene la dirección de los datos. Por ejemplo, al sumar dos números, el registro de direccionamiento de memoria almacena la dirección del primer número. Los datos en esa dirección se transfieren al acumulador. Después el segundo número se almacena en el registro de direccionamiento de memoria. El dato

de esta dirección se suma al dato en el acumulador. El resultado se guarda en una dirección que invoca el registro de direccionamiento de memoria.

5. *Registro de instrucciones* (IR, *instruction register*)
   Este registro guarda instrucciones. Después de traer una instrucción de la memoria a través del bus de datos, la CPU la almacena en el registro de instrucciones. Después de cada traída de instrucción, el microprocesador incrementa el contador del programa en uno y como resultado el contador del programa apunta a la siguiente instrucción que espera ser traída. La instrucción puede entonces decodificarse y usarse para ejecutar una operación. Esta secuencia se conoce como el ciclo de *trae-ejecuta*.

6. *Registros de propósito general*
   Estos registros pueden servir para almacenar datos o direcciones en forma temporal y se utilizan en operaciones de transferencias entre varios registros.

7. *Registro de apuntador de la pila* (SP, *stack pointer register*)
   El contenido de este registro almacena una dirección que define el tope de la pila en la memoria RAM. La pila es un área especial de memoria donde se almacenan los valores del contador de programa cuando se ejecuta una subrutina.

La cantidad y tipo de registros dependerá del microprocesador que se use. Por ejemplo, el microprocesador 6800 de Motorola (figura 15.3) tiene dos registros acumuladores, un registro de estado, un registro de índice, un registro de apuntador de pila y un registro de contador de programa. El registro de estado tiene bits de bandera para indicar signo negativo, cero, acarreo, desbordamiento, medio acarreo e interrupción. El microprocesador 6802 de Motorola es similar, pero incluye memoria RAM y un reloj integrado.

El microprocesador 8085A de Intel es un desarrollo basado en el procesador 8080, el 8080 requería un generador de reloj externo mientras que el 8085A tiene un generador de reloj intergrado. Los programas escritos para el 8080 se pueden correr en el 8085A. El 8085A tiene seis registros de propósito general B, C, D, E, H y L, un apuntador de pila, un contador del programa, un registro de banderas y dos registros temporales. Los registros de propósito general se pueden usar como seis registros de 8 bits o en pares BC, DE y HL como registros de 16 bits. La figura 15.4 muestra un diagrama de bloques representativo de la arquitectura.

Como será aparente a partir de las figuras 15.3 y 15.4, los microprocesadores tienen una gama amplia de entradas y salidas de control y temporización. Éstas proveen salidas cuando un microprocesador está llevando a cabo ciertas operaciones y entradas para influenciar operaciones de control. Adicionalmente existen entradas relacionadas con el control de interrupciones. Éstas se diseñaron para permitir que la operación de un programa se interrumpa como resultado de algún evento externo.

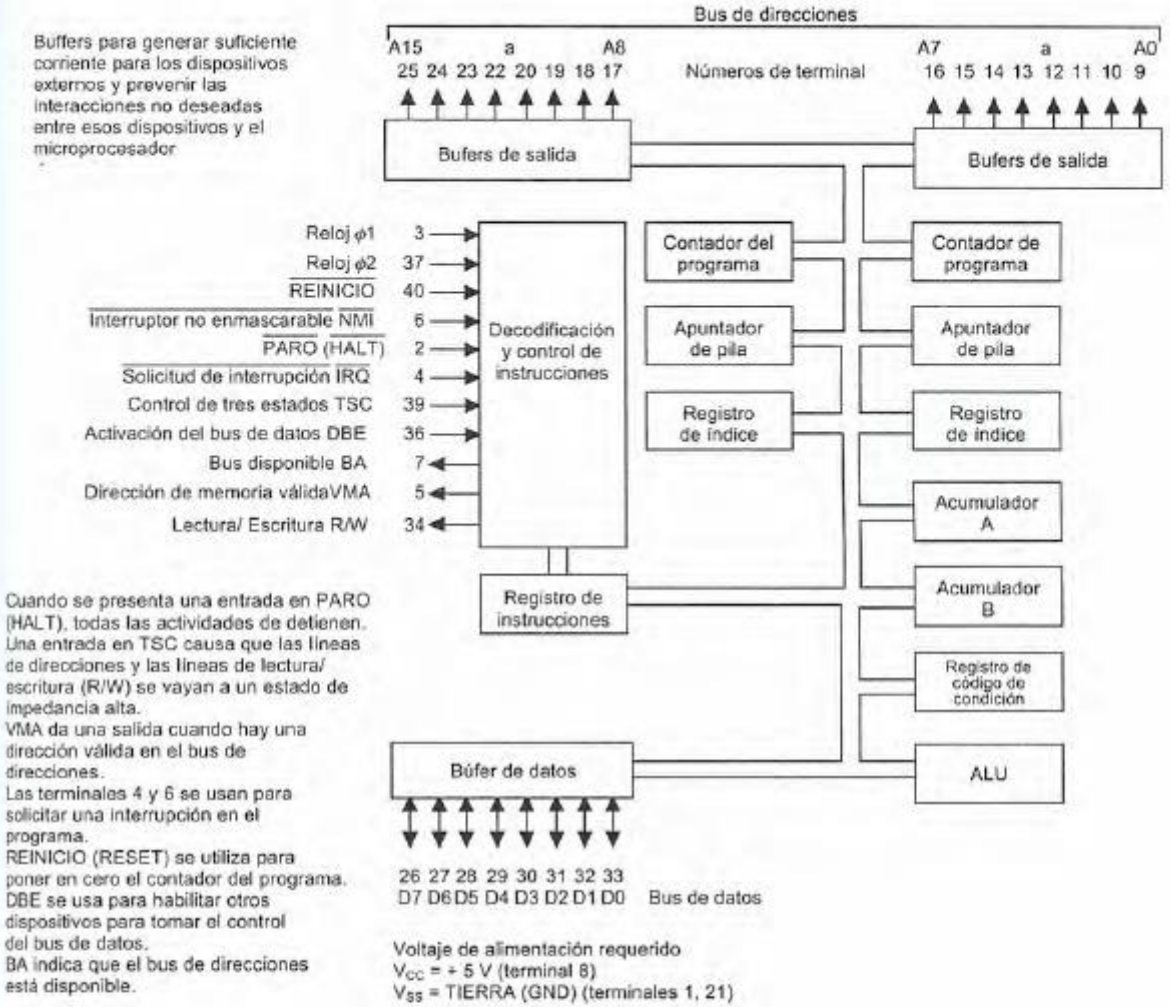


Figura 15.3 Arquitectura del microprocesador 6800 de Motorola

15.2.3 Memoria

La unidad de memoria de un microprocesador guarda datos binarios y toma la forma de uno o varios circuitos integrados. Los datos pueden ser códigos de instrucciones de un programa, o números con los que se realizan operaciones.

El tamaño de la memoria depende de la cantidad de líneas del bus de direcciónes. Los elementos de la unidad de memoria están formados en esencia por grandes cantidades de celdas de memoria, cada una guarda un bit 0 o 1. Las celdas de memoria se agrupan por localidades, y cada localidad puede guardar una palabra. Para accesar la palabra almacenada, se identifica cada localidad por una dirección única. De esta manera, en un bus de dirección de 4 bits se pueden identificar 16 direcciones diferentes, cada una tal vez, capaz de guardar un byte, es decir, un grupo de 8 bits.

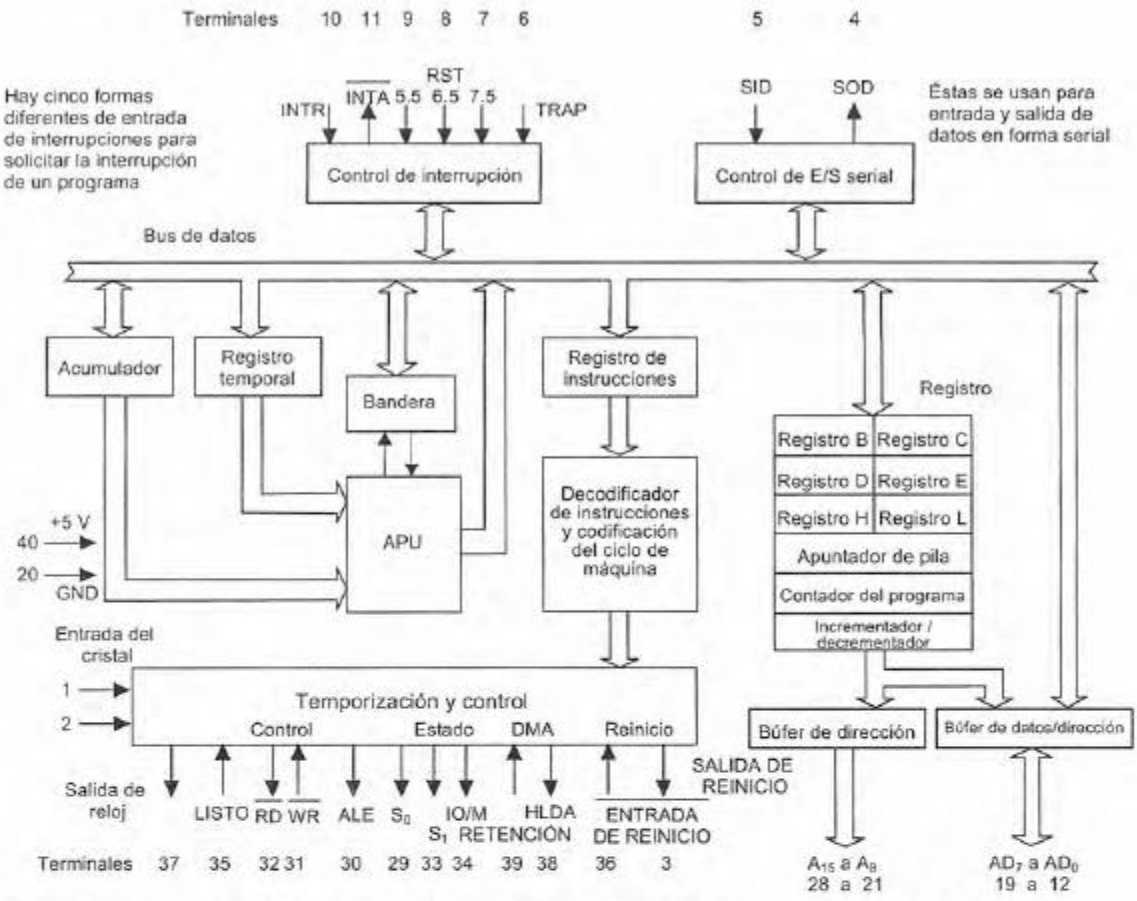


Figura 15.4 Arquitectura del Intel 8085A

Como ilustración del análisis anterior de los datos almacenados en relación con el tamaño del bus de direcciones, la figura 15.5 muestra las posibilidades con un bus de direcciones de 4 bits.

La capacidad de la unidad de memoria se especifica por la cantidad de localidades de memoria disponibles; 1 K es $2^{10} = 1024$ localidades; una memoria de 4 K tiene 4096 localidades.

Existen varios tipos de unidad de memoria:

1. *ROM*
   Cuando se guardan datos en forma permanente, se utiliza un dis-

Figura 15.5 Tamaño del bus de dirección

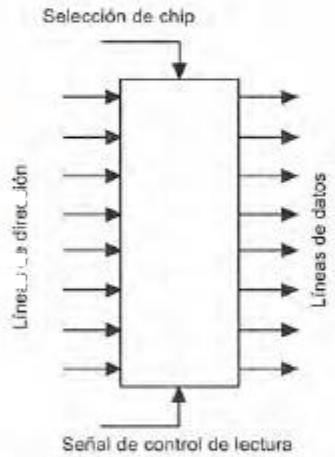


Figura 15.6 Chip de una ROM

positivo de memoria conocido como *memoria de sólo lectura* (ROM, *read only memory*). Las memorias ROM se programan con el contenido que se requiere durante la fabricación del circuito integrado. Mientras el chip de memoria esté en la computadora no es posible escribirle datos, sólo se permite la lectura, y se utiliza para programas fijos, como el sistema de arranque o "boot" de una computadora y programas para aplicaciones específicas de los microprocesadores. Aun cuando se suspenda la alimentación eléctrica, esta memoria no pierde su contenido. La figura 15.6 muestra las conexiones de un chip ROM típico capaz de guardar 1 K × 8 bits.

2. *PROM*
El término *ROM programable* (PROM) se refiere a las memorias ROM que puede programar el usuario. En un principio, las celdas de memoria tienen un fusible como eslabón que mantiene su memoria en 0. Al hacer pasar una corriente por el fusible, se abre de manera permanente y el valor cambia de 0 a 1. Una vez que el eslabón está abierto, los datos se guardan en forma permanente en la memoria y ya no es posible modificarlos.

3. *EPROM*
El término *ROM borrable y programable* (EPROM, *erasable and programmable* ROM) se refiere a memorias ROM que es posible programar y modificar. Un chip de EPROM típico contiene una serie de pequeños circuitos electrónicos, celdas, donde se almacena una carga. Para almacenar el programa se aplican voltajes a las terminales del circuito integrado y se produce una configuración de celdas cargadas y no cargadas. Esta configuración queda permanente en el chip hasta que la borra un haz de luz ultravioleta que pasa por una ventana de cuarzo ubicada en la parte superior del dispositivo. Esto provoca la descarga de todas las celdas. Por lo tanto, es posible volver a programar el chip. La EPROM 2716 de Intel tiene 11 conexiones de dirección y una para activación, la cual se activa con un valor bajo.

4. *EEPROM*
La *PROM eléctricamente borrable* (EEPROM, *electrically erasable* PROM) es similar a las EPROM, pero para el borrado se utiliza un voltaje relativamente alto, en vez de la luz ultravioleta.

5. *RAM*
Los datos temporales, es decir, datos con los que se están realizando operaciones, se guardan en una memoria de lectura/escritura conocida como *memoria de acceso aleatorio* (RAM, *random access memory*); en ella se puede leer y escribir. La figura 15.7 muestra las conexiones típicas del chip de una RAM de 1 K × 8 bits. El chip RAM 6810 de Motorola tiene siete conexiones de dirección y seis para selección; de éstas, cuatro se activan con un valor bajo y dos con uno alto; para activar la RAM, todas se deben activar al mismo tiempo.

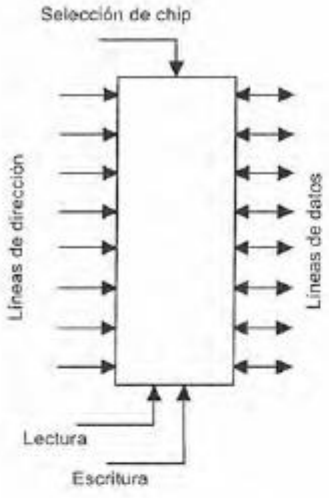


Figura 15.7 Chip de memoria RAM

Cuando en una ROM se guarda un programa, estará disponible y listo cuando se activa el sistema. Los programas que se guardan en una ROM se conocen como *firmware* (microprogramas). Algunos deben estar presentes siempre. Los programas guardados en una RAM se conocen como *software*. Cuando el sistema se activa, el software se puede cargar en la RAM desde el equipo periférico, como el teclado, el disco duro o un disco flexible.

15.2.4 Entrada/salida

La operacion de entrada/salida se define como la transferencia de datos entre el microprocesador y el mundo exterior. El término *dispositivos periféricos* se refiere a las piezas de equipo que intercambian datos con un sistema de microprocesador. Dado que las velocidades y características de los dispositivos periféricos pueden ser muy distintas a las del microprocesador, se conectan a través de circuitos de interfase. Una de las funciones más importantes de uno de estos circuitos es sincronizar la transferencia de datos entre el microprocesador y el dispositivo periférico. En las operaciones de entrada, el dispositivo de entrada coloca los datos en el registro de datos del circuito de interfase; estos datos permanecen ahí hasta que los lee el microprocesador. En las operaciones de salida, el microprocesador coloca los datos en el registro hasta que los lee el dispositivo periférico.

Para que el microprocesador introduzca datos válidos de un dispositivo de entrada necesita estar seguro de que el circuito de interfase ha retenido correctamente los datos de entrada. Para ello realiza un *muestreo* o una *interrupción*. En el primer caso, el chip de interfase recurre a un bit de estado definido como 1 para indicar que los datos son válidos. El microprocesador sigue verificando hasta que aparece este bit de estado en 1. El problema con este método es que el microprocesador debe esperar hasta encontrar el bit de estado. En el método de interrupción, el circuito de interfase envía una señal de interrupción al microprocesador cuando contiene datos válidos; el microprocesador suspende la ejecución de su programa principal y ejecuta la rutina asociada con la interrupción para leer los datos.

15.2.5 Ejemplos de sistemas

La figura 15.8 muestra un ejemplo de un sistema basado en microprocesador que usa el microprocesador 8085A de Intel: tiene un registro de direcciones 74LS373, un decodificador de direcciones de 3 a 8 líneas 74LS138, dos chips 2114 de memoria RAM de 1K × 4, un chip 2716 de memoria EPROM de 2K × 8 y dos chips 74LS244 y 74LS374 que son interfases de entrada y salida, respectivamente.

1. *Registro de direcciones*
   La salida de habilitación del registro de direcciones (ALE, *address latch enable*) proporciona una salida al hardware externo para indicar cuando las líneas AD0-AD7 contienen una dirección y cuando contienen datos. Cuando la ALE está en alto activa el registro y las líneas A0-A7 transfieren la parte baja de la di-

rección a este registro donde se enclava. Entonces cuando la ALE cambia y regresa a bajo, de modo que los datos pueden salir del microprocesador, esta parte de la dirección permanece enclavada (*latched*) en el 74LS373. La parte alta de la dirección se envía a través de las líneas A8-A15 y siempre es válida; la dirección completa está dada por la parte baja en el registro de direcciones y la parte alta en el bus de direcciones del microprocesador.

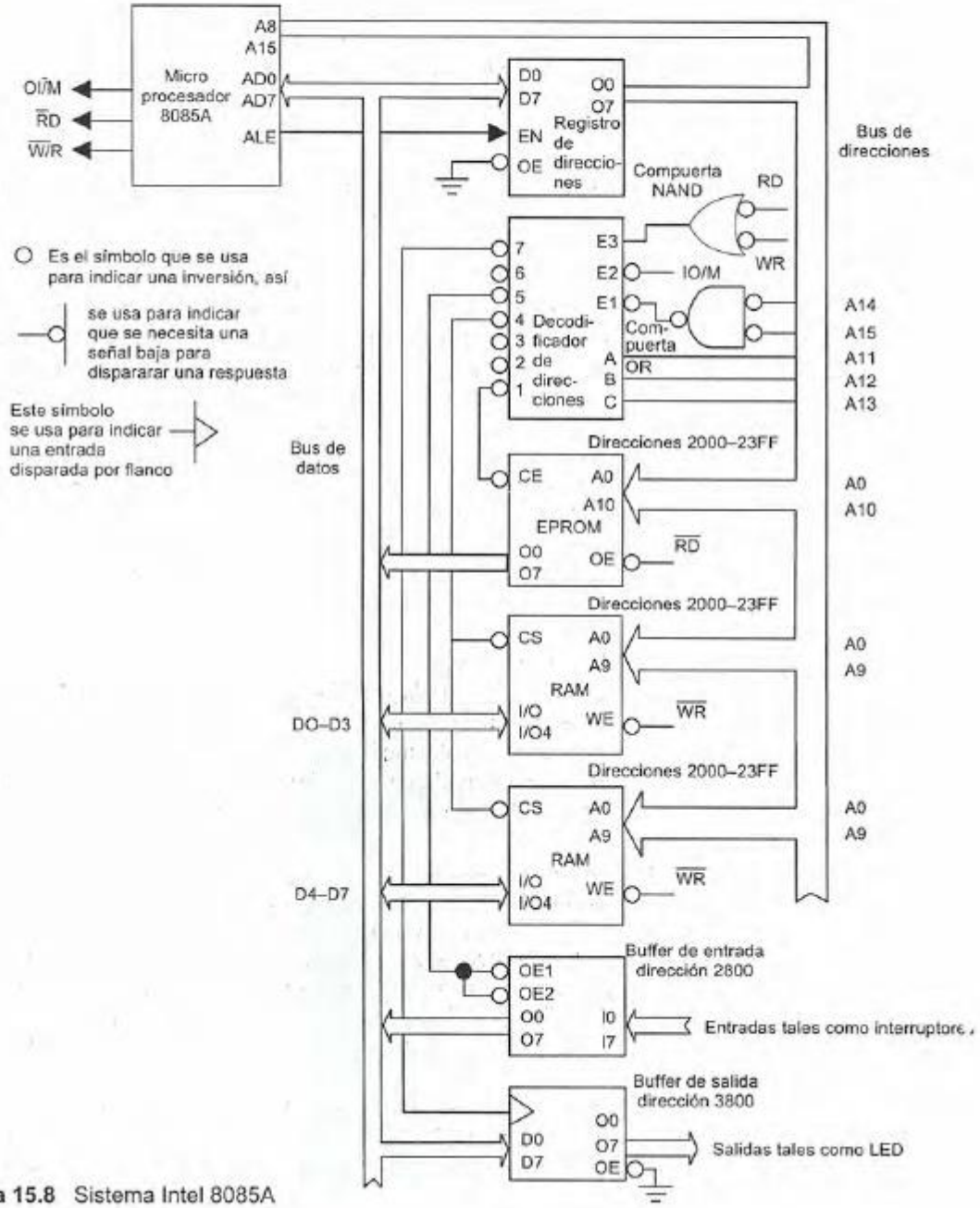


Figura 15.8 Sistema Intel 8085A

2. *Decodificador de direcciones*
   El 74LS138 es un decodificador de 3 a 8 líneas y proporciona una señal activa baja en una de sus ocho salidas; la salida elegida depende de las señales en sus tres líneas de entrada A, B y C (vea la figura 14.32). Antes de poder elegir, debe habilitarse con las entradas de habilitación 1 y 2 en bajo y la 3 en alto.

3. *Memoria EPROM*
   Los bits de dirección A11, A12, A13 y A14 se usan para seleccionar qué dispositivo se va a direccionar. Esto deja a los bits A0-A10 para la dirección, y entonces la EPROM puede tener $2^{11} = 2048$ direcciones, que es el tamaño de la memoria EPROM 2716 de Intel. La EPROM se selecciona siempre que el microprocesador lea una dirección entre 0000 y 07FF y da como salida su contenido de 8 bits al bus de datos a través de las líneas O0-O7. La línea de habilitación de salida (OE, *output enable*) se conecta a la salida de lectura del microprocesador para asegurar que la EPROM sea sólo de escritura.

4. *Memoria RAM*
   Se muestran dos chips de memoria RAM según se utilizan, cada una de $1K \times 4$. En conjunto proporcionan una memoria para señales de 8 bits. Ambos chips utilizan los mismos bits de direcciones de A0-A9 para la selección de memoria, donde un chip proporciona los bits de datos de D0-D3 y el otro los bits de D4-D7. Con 10 bits para la dirección se tienen $2^{10} = 1024$ diferentes direcciones, de 2000 a 23FF. La memoria RAM usa la entrada de habilitación de escritura (WE, *write enable*) para determinar si se lee o escribe en la memoria. Si la entrada está en bajo, se está escribiendo en la dirección de la RAM seleccionada, si está en alto se está leyendo.

5. *Buffer de entrada*
   El buffer de entrada 74LS244 pasa el valor binario de las entradas sobre el bus de datos siempre que OE1 y OE2 estén en bajo. Se accede a éste mediante cualquier dirección entre 2800 y 2FFF, así, se podría utilizar 2800. El buffer es para asegurar que las entradas no sean carga para el microprocesador.

6. *Registro de salida*
   El chip 74LS374 es un registro de salida. Enclava o retiene la salida del microprocesador de manera que los dispositivos de salida tengan tiempo para leerlo, mientras que el microprocesador puede seguir con otras instrucciones de su programa. El registro de salida está dado por un intervalo de direcciones de 3800 a 3FFF y de este modo podría direccionarse usando 3800.

La figura 15.9 muestra un ejemplo de un sistema basado en el uso del microprocesador 6800 de Motorola que sólo tiene un chip de RAM, un chip de ROM y entrada/salida programable. Con este sistema no

es necesaria la decodificación de direcciones debido al reducido número de dispositivos involucrados. Para las estradas/salidas en paralelo se usa un adaptador de interfase periférico (PIA, *peripheral interface adapter*) (sección 18.4) y para entradas/salidas en serie se utiliza un adaptador de interfase asíncrono (ACIA, *asynchonous interface adapter*) (sección 18.5). Éstos se pueden programar para manejar las entradas y salidas y dar el aislamiento requerido.

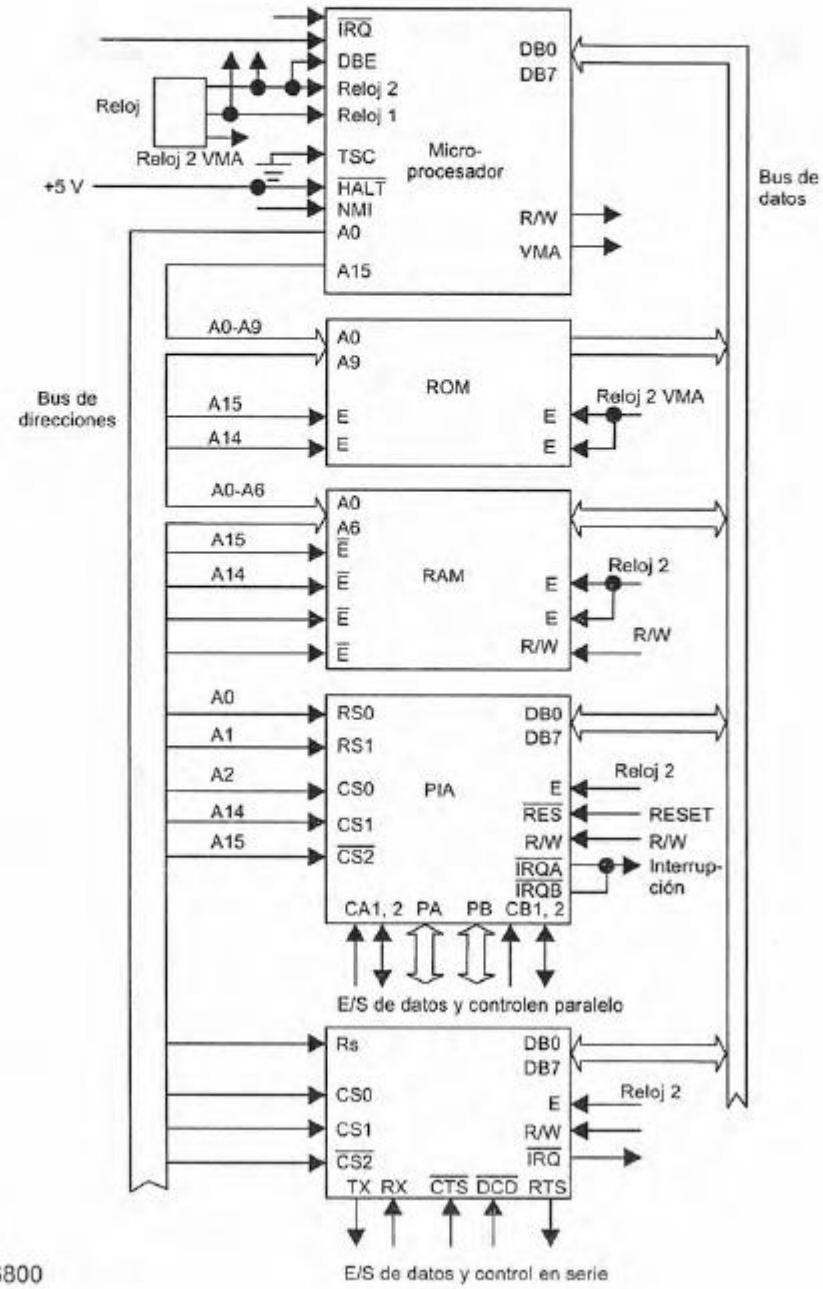


Figura 15.9 Sistema M6800

1. *Memoria RAM*
   Las líneas de direcciones A14 y A15 se conectan a las entradas de habilitación del chip de RAM. Cuando ambas líneas están en bajo, el chip de la memoria RAM está conversando con el microprocesador.
2. *Memoria ROM*
   Las líneas de direcciones A14 y A15 se conectan a las entradas de habilitación del chip de ROM y cuando las señales en ambas líneas están en alto, entonces se está direccionando el chip de la memoria ROM.
3. *Entradas/salidas*
   Las líneas de direcciones A14 y A15 se conectan a las entradas de habilitación de PIA y ACIA. Cuando la señal en la línea A15 es baja y la señal en la A14 es alta entonces se direccionan las interfases entrada/salida. A fin de indicar qué dispositivos se están habilitando, la línea A2 de direcciones se hace alta para el PIA y la línea A3 se hace alta para el ACIA.

15.3 Microcontroladores

Para que un microprocesador pueda funcionar como un sistema aplicado al control, son necesarios chips adicionales, por ejemplo, dispositivos de memoria para almacenar programas y datos, así como puertos de entrada/salida para permitir que se comunique con el mundo exterior y reciba señales desde él. El *microcontrolador* consiste en la integración en un chip de microprocesador con memoria, interfases de entrada/salida y otros dispositivos periféricos como temporizadores. La figura 15.10 muestra un diagrama de bloques general de un microcontrolador.

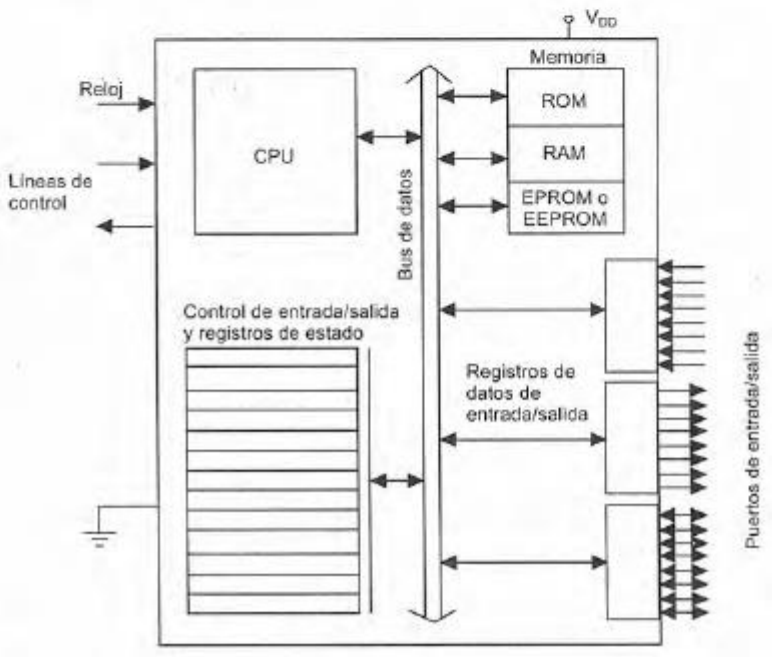


Figura 15.10 Diagrama de bloques de un microcontrolador

Un microcontrolador común tiene terminales para la conexión externa de entradas y salidas, alimentación eléctrica y señales de reloj y de control. Las conexiones de entrada y salida se agrupan en unidades denominadas puertos de entrada/salida. Por lo general, estos puertos tienen ocho líneas para poder transportar una palabra de datos de 8 bits. Para una palabra de 16 bits se utilizan dos puertos, uno para transmitir los 8 bits inferiores, y otro para los 8 bits superiores. Los puertos pueden ser sólo entrada o sólo salida, o programables para funcionar como entrada o salida.

El 68HC11 de Motorola, el 8051 de Intel y el PIC16C6x/7x son ejemplos de microcontroladores de 8 bits en cuanto a que el bus de datos tiene capacidad para 8 bits. El 68HC16 de Motorola es un ejemplo de microcontrolador de 16 bits y el 68300 de Motorola es un microcontrolador de 32 bits. Los microcontroladores tienen cantidades limitadas de ROM y RAM y se usan ampliamente para sistemas de control integrados. Un sistema de microprocesador con memoria separada y chips de entrada/salida es más apropiado para procesar información en un sistema de computadora.

15.3.1 El M68HC11 de Motorola

Motorola cuenta con dos familias básicas de microcontroladores de 8 bits: el 68HC05, que es la versión económica, y el 68HC11, que es la versión con rendimiento superior. La familia M68HC11 de Motorola (figura 15.9) se basa en el microprocesador 6800 de Motorola, el cual es muy utilizado para sistemas de control.

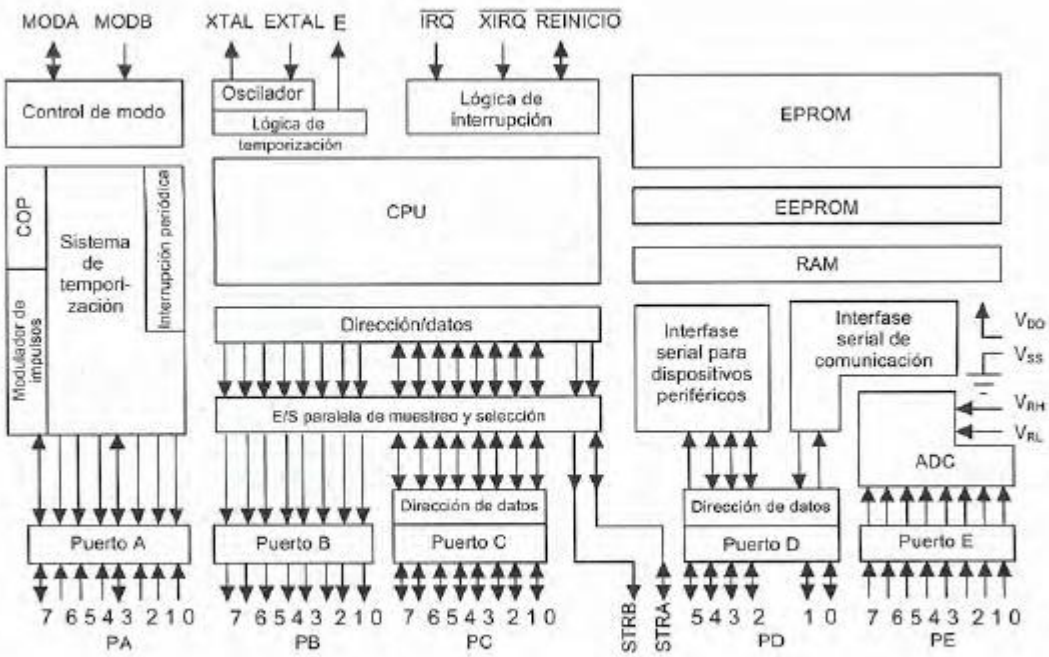


Figura 15.11 Diagrama de bloques de M68HC11

Existen muchas versiones en esta familia, las diferencias se deben al tipo de RAM, ROM, EPROM, EEPROM y las características del registro de configuración. Por ejemplo, una versión (68HC11A8) tiene 8 K de ROM, 512 bytes de EEPROM, 256 bytes de RAM, un sistema de temporización de 16 bits, una interfase serial síncrona, una interfase de comunicación serial sin retorno a cero asíncrona, un convertidor analógico a digital de 8 bits, 8 canales, para las entradas analógicas y cinco puertos A, B, C, D y E.

Registro de datos del puerto A $1000

Bit

7	6	5	4	3	2	1	0

Figura 15.12 Registro del puerto A

1. *Puerto A*
   El puerto A tiene sólo tres líneas de entrada, cuatro líneas de salida y una línea que funciona como entrada o salida. La dirección del registro de datos del puerto A es $1000 (figura 15.12), la dirección del registro de control del acumulador de pulsos es $1026 (figura 15.13); este registro controla la función de cada bit del puerto A. Este puerto también permite el acceso al temporizador interno del microcontrolador, los bits PAMOD, PEDGE, RTR1 y RTRO controlan el acumulador de pulsos y el reloj.

Figura 15.13 Registro del control del acumulador de pulsos

Registro de datos del puerto B $1004

Bit

7	6	5	4	3	2	1	0

Figura 15.14 Registro del puerto B

2. *Puerto B*
   El puerto B sólo funciona como salida y tiene ocho líneas (figura 15.14). No es posible colocar datos de entrada en las terminales del puerto B. Su registro de datos está en la dirección $1004 y para extraer datos es necesario escribir a esta ubicación de memoria.

Registro de datos del puerto C $1003

Bit

7	6	5	4	3	2	1	0

Registro de datos del puerto C $1007

Bit

7	6	5	4	3	2	1	0

Cuando un bit es puesto en 0, el bit correspondiente en el puerto es una entrada, cuando es puesto en 1, una salida

Figura 15.15 Registro del puerto C

3. *Puerto C*
   El puerto C puede ser entrada o salida; los datos se escriben o leen de su registro de datos en la dirección $1003 (figura 15.15). Su dirección se controla mediante el registro de direcciones de datos del puerto en la dirección $1007. Los ocho bits en este registro corresponden a los bits individuales del puerto C y determinan si las líneas son de entrada o salida; cuando el bit del registro de dirección de datos se fija en 0 es una entrada y cuando se fija en 1 es una salida. Las líneas STRA y STRB (cuando funcionan en modo single chip) se vinculan a los puertos B y C y se utilizan para las señales de protocolo (handshake) de dichos puertos. Estas líneas controlan el tiempo de transferencia de datos. El registro de control de E/S en paralelo PIOC, en la dirección $1002 contiene bits para controlar el modo de handshake, así como la polaridad y los flancos activos de las señales de handshake.

Registro de datos del puerto D $1008

Bit

		5	4	3	2	1	0

Registro de datos del puerto D $1009

Bit

		5	4	3	2	1	0

Cuando un bit es puesto en 0, el bit correspondiente en el puerto es una entrada, cuando es puesto en 1, una salida

Figura 15.16 Registro del puerto D

Registro de datos del puerto E $100A

Bit

7	6	5	4	3	2	1	0

Figura 15.17 Registro del puerto E

4. *Puerto D*

El puerto D contiene sólo seis líneas, que pueden ser tanto entrada como salida, y su registro de datos está en la dirección $1008 (figura 15.16); las direcciones se controlan mediante el registro de direcciones del puerto, en la dirección $1009; el bit correspondiente se define como 0 para una entrada y como 1 para una salida. El puerto D también sirve como conexión a los dos subsistemas seriales del microcontrolador. La interfase para comunicación serial es un sistema asíncrono que proporciona una comunicación serial compatible con modems y terminales. La interfase periférica serial es un sistema síncrono de alta velocidad diseñado para comunicar el microcontrolador y los componentes periféricos compatibles con estas velocidades.

5. *Puerto E*

El puerto E es de 8 bits sólo de entrada (figura 15.17) que se puede utilizar como puerto de entrada de propósito general, o para entradas al convertidor interno analógico-digital. Las dos entradas V_{RH} y V_{RL} proporcionan voltaje de referencia al ADC. El registro de datos del puerto E está en la dirección $1002.

El 68HC11 tiene un convertidor analógico-digital interno; los bits del puerto E, 0, 1, 2, 3, 4, 5, 6 y 7 son las terminales de la entrada analógica. Dos líneas V_{RH} y V_{LH} proporcionan los voltajes de referencia al ADC; el voltaje de referencia alto V_{RH} no debe ser menor que V_{DD}, o sea, 5 V, y el voltaje de referencia bajo V_{LH} no debe ser menor que V_{SS} , o sea, 0 V. El ADC debe habilitarse antes de que se pueda usar. Esto se hace estableciendo el bit de control ADPU (encendido A/D) en el registro OPTION (figura 15.18), o sea el bit 7. El bit 6 selecciona la fuente de reloj para el ADC. Se requiere un retardo de cuando menos 100 μs después del encendido para permitir que el sistema se estabilice.

Registro OPTION $1039

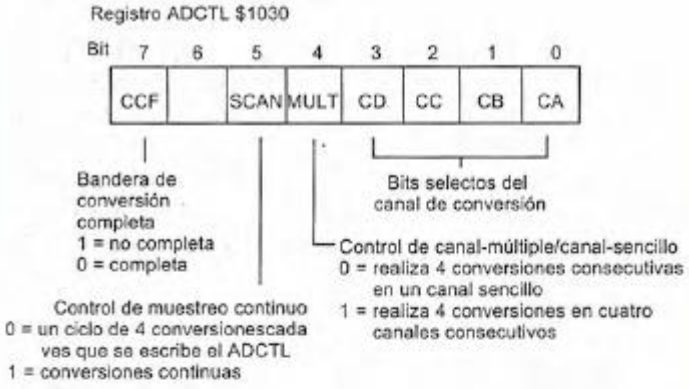
Bit	7	6	5	4	3	2	1	0
	ADPU	CSEL	IRQE	DLY	CME		CR1	CR2

Selecciona reloj
0 = selecciona reloj E, 1 = selecciona oscilador RC interno

0 = A/D no encendido, 1 = A/D encendido

Figura 15.18 Registro OPTION

La conversión analógica a digital se inicia escribiendo al registro ADCTL (registro A/D control/estado) después de necenderlo y del retardo de estabilización (figura 15.19). Esto implica seleccionar canales y modos de operación. La conversión inicia un ciclo de reloj después. Por ejemplo, si se selecciona un canal sencillo haciendo MULT = 0 las cuatro conversiones A/D sucesivas se harán en el canal seleccionado por los bits CD-CA. El resultado de la conversión se guarda en los registros de resultados A/D ADR1-ADR4.



MULT = 0

CD	*CC*	*CB*	*CA*	*Canal convertido*
0	0	0	0	PE0
0	0	0	1	PE1
0	0	1	0	PE2
0	0	1	1	PE3
0	1	0	0	PE4
0	1	0	1	PE5
0	1	1	0	PE6
0	1	1	1	PE7

MULT = 1

				Registros de resultados A/D			
CD	*CC*	*CB*	*CA*	*ADR1*	*ADR2*	*ADR3*	*ADR4*
0	0	X	X	PE0	PE1	PE2	PE3
0	1	X	X	PE4	PE5	PE6	PE7

Figura 15.19 Registro ADCTL

6. *Modos*
MODA y MODB son dos terminales que se pueden usar para definir, durante el encendido, el funcionamiento del microcontrolador en uno de cuatro modos posibles: inicio especial, prueba especial, un solo chip y ampliado.

MODB	MODA	Modo
0	1	Inicio especial
0	1	Prueba especial
1	0	Un solo chip
1	1	Ampliado

En el modo de single chip, el microcontrolador es por completo autosuficiente, excepto por una fuente de reloj externa y un circuito de reinicio. Con este modo, es posible que los recursos propios del microcontrolador no sean suficientes como la memoria; en estos casos se puede usar el modo ampliado para aumentar el número de direcciones. Los puertos B y C proporcionan buses de dirección, datos y control. El puerto B ofrece las ocho terminales para la dirección superior y el puerto C, las terminales para los datos multiplexados y para la dirección inferior. El modo bootstrap permite al fabricante cargar programas especiales en una ROM especial para clientes que utilizan el M68HC11. Cuando el microcontrolador se configura en este modo, se carga el programa especial. El modo test se usa principalmente para pruebas de producción internas en Motorola.

Después de seleccionar el modo, la conexión MODA se puede utilizar para determinar el inicio de la ejecución de una instrucción. La función de la terminal MODB es servir como un medio para que la RAM interna del chip pueda recibir energía cuando se suspende la energía eléctrica normal.

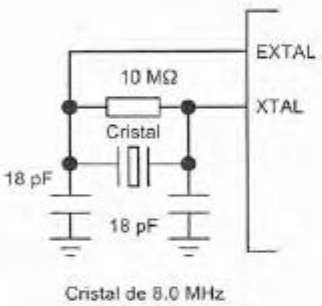


Figura 15.20 Entrada del oscilador

7. *Terminales del oscilador*
Las terminales del sistema oscilador XTAL y EXTAL son conexiones necesarias para accesar al oscilador interno. La figura 15.20 muestra un circuito externo que puede usarse. E es el bus temporizador y funciona a un cuarto de la frecuencia del oscilador y se puede emplear para sincronizar eventos externos.

8. *Controlador de interrupción*
Este controlador permite al microcontrolador interrumpir un programa (vea el capítulo 18). Por las líneas IRQ y XIRQ entran las señales de interrupción. RESET es para el restablecimiento. Una interrupción es un evento que requiere la CPU para detener la ejecución normal de un programa y para realizar algún servicio relacionado con el evento. Las líneas IRQ y XIRQ están asignadas a las fuentes de interrupción externas.

9. *Temporizador*
El M68HC11 contiene un sistema de temporización que tiene un contador de ejecución libre, una función de comparación de cinco salidas, la capacidad para capturar el tiempo cuando se produce un evento externo, una interrupción periódica en tiempo real y un contador, denominado acumulador de impulsos, para eventos externos. El contador de ejecución libre, denominado TCNT, es un contador de 16 bits que empieza a contar en 0000, cuando se restablece la CPU y sigue funcionando en forma continua sin que el programa lo pueda reiniciar. En cualquier momento se puede leer su valor. La fuente del contador es el temporizador de bus del sistema y se puede graduar de manera anticipada definiendo en el registrador TMSK2 los bits PR0 y PR1 como bits 0 y 1 en la dirección $1024 (figura 15.21).

Registro de interruptor del temporizador 2 en la dirección $1024

Bit

7	6	5	4	3	2	1	0
						PR1	PR0

Figura 15.21 Registro TMSK2

Factores de preescala

			Una cuenta	
			Frecuencia del bus	
PR1	*PR0*	*Factor de preescala*	*2 MHz*	*1 MHz*
0	0	1	0.5 ms	1 ms
0	1	4	2 ms	4 ms
	0	8	4 ms	8 ms
	1	16	8 ms	16 ms

La salida de las funciones de comparación permite especificar los tiempos en que ocurrirá una salida cuando termine la cuenta definida. El sistema de captura de entrada consigna el valor del contador cuando se produce una entrada, de manera que captura el tiempo exacto en que ocurre una entrada. Es posible configurar el acumulador de impulsos para que funcione como contador de eventos y cuente los impulsos de temporización externos o como acumulador de tiempo de modo que guarde la cantidad de impulsos que se producen durante cierto intervalo como resultado de la activación del contador y, después de cierto tiempo, se desactive. El registro de control de acumulador de impulsos, PACTL (figura 15.12), que se encuentra en la dirección $1026 se usa para seleccionar el modo de operación. El bit PAEN se establece en 0 para desactivar el acumulador de impulsos y en 1, para activarlo; el bit PAMOD se hace 0 para activar el modo de contador de eventos y 1 para el modo de tiempo activado; el bit PEDGE se hace 0 para que el acumulador de impulsos responda a un flanco descendente cuando opera en el modo contador de eventos y 1 para que responda a un flanco ascendente. En el modo de tiempo accionado, el bit PEDGE se hace 0 para desactivar el conteo cuando el bit 7 del puerto A es 0 y para que acumule cuando ese bit sea 1; cuando el bit PEDGE es 1 en este modo, se desactiva el conteo cuando el puerto A, bit 7 es 1 y se activa cuando es 0.

10. *COP*

Otra función de temporización es la función de la *operación correcta de la computadora* (COP, *computer operating properly*). Consiste en un temporizador que apaga y restablece el sistema si no ha concluido alguna operación dentro de un lapso razonable (sección 21.2). También se le conoce como *temporizador vigilante.*

11. *PWM*

La modulación de ancho de pulso (PWM, *pulse width modulation*) controla la velocidad de los motores de cd (vea la sección 7.5.5) mediante una señal de onda cuadrada; al variar la cantidad de tiempo que la señal está presente, se modifica el valor promedio de la señal. Para generar la onda cuadrada se utiliza un mi-

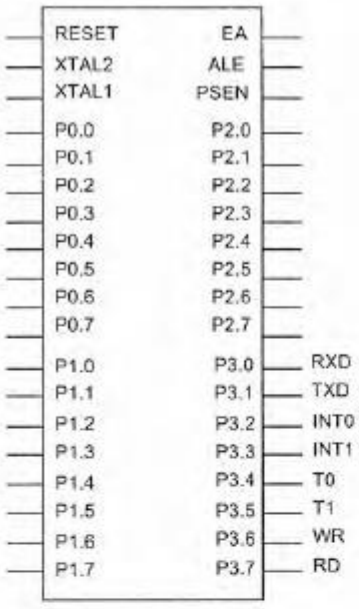


Figura 15.22 Intel 8051

crocontrolador, disponiéndolo para que haya una salida cada medio periodo. Sin embargo, algunas versiones del M68HC11 tienen un módulo de modulación de ancho de pulso de manera que, después de configurar y activar el módulo de PWM, se pueden generar automáticamente las ondas de PWM.

De lo anterior se puede concluir que antes de utilizar un microcontrolador es necesario inicializarlo, es decir, colocar los bits en los registros adecuados para que funcione como se requiere.

Lo anterior es sólo una breve indicación de las conexiones de entrada/salida del microcontrolador M68HC11. Si el lector desea profundizar sobre el tema, se sugiere consultar los manuales publicados por el fabricante o las siguientes obras: *Software and Hardware Engineering, Motorola M68HC11* de F.M. Cady (Oxford University Press, 1977) o *Microcontroller Technology, The 68HC11* de P. Spasov (Prentice-Hall, 1996, 1992).

15.3.2 El 8051 de Intel

Otras familia común de microcontroladores es la 8051 de Intel. La figura 15.22 muestra sus conexiones y la figura 15.23 su arquitectura. El 8051 tiene cuatro puertos de entrada/salida en paralelo: los puertos 0, 1, 2 y 3. Los puertos 0, 2 y 3 también desempeñan funciones alternas. La versión 8051AH tiene una memoria ROM de 4 K, una memoria RAM de 128 bytes, dos temporizadores y un control de interrupción para cinco fuentes.

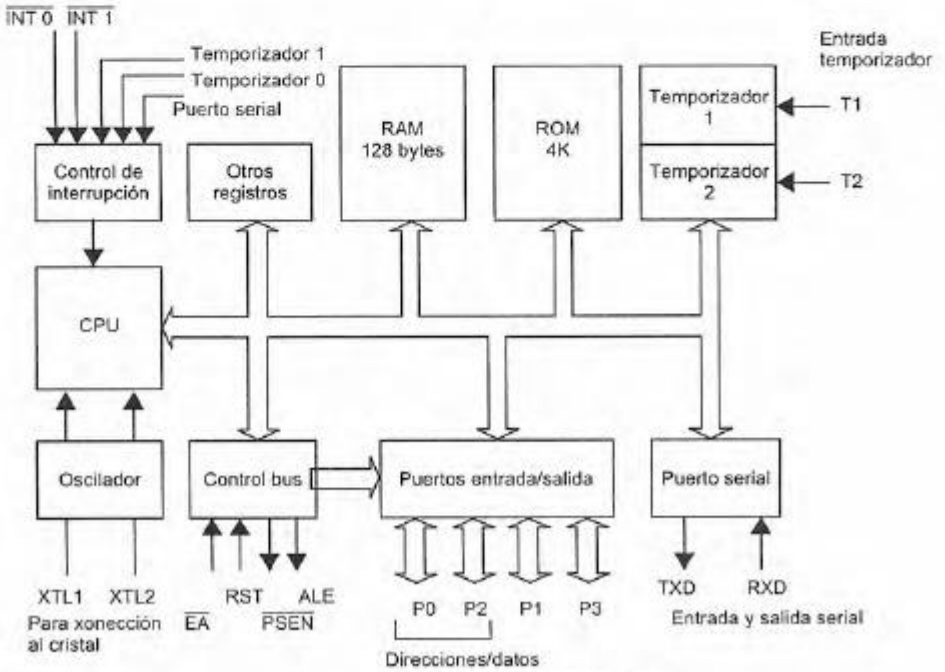


Figura 15.22 Intel 8051

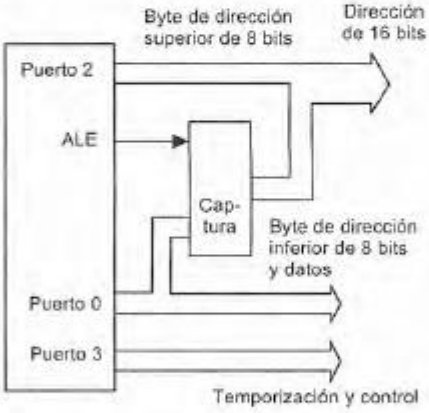


Figura 15.24 Aplicación de ALE

1. *Puertos de entrada/salida*
   El puerto 0 está enla dirección 80H, el puerto 1 en la dirección 90H, el puero 2 en la dirección A0H y el puerto 3 en la dirección B0H (Intel utiliza una H (o h) después de la dirección para indicar que es hexadecimal). Cuando un puerto se usa como puerto de salida, los datos se colocan el el registro de función especial correspondiente. Cuando un puerto se va a utilizar como puerto de entrada, el valor FFH deberá escribirse primero. Todos los puertos son direccionables por bit. Así, por ejemplo, podemos utilizar el bit 6 del puerto 0 para encender o apagar un motor y quizas, para encender o apagar una bomba el bit 7.

   El puerto 0 se utiliza tanto como puerto de entrada como de salida. También se puede emplear para acceder a memoria externa como un bus multiplexado de direcciones y datos. El puerto 1 se utiliza como puerto de entrada y de salida. El puerto 2 se usa tanto como puerto de entrada como de salida. También se puede emplear para acceder a memoria externa por el bus de direcciones altas. El puerto 3 se utiliza como puerto de entrada y de salida, o como puerto de entrada/salida para propósitos especiales. Entre las funciones alternas del puerto 3 están las de salidas de interrupción y temporización, entrada y salida de puerto serial y señales de control de interfase con la memoria externa. RXD es el puerto de entrada serial, TXD el puerto de salida en serie, INT0 la interrupción externa 0 e INT1 la interrupción externa 1, T0 es la entrada externa 0 del temporizador/contador, T1 la entrada externa 1 del temporizador/contador, WR se usa para la selección de escritura de la memoria externa y RD para la selección de lectura de la memoria externa. El término *selección* se refiere a una conexión que sirve para activar o desactivar una función particular.

2. *ALE*
   La conexión para la *habilitación del registro de direcciones* (ALE, *address latch enable*) produce un impulso de salida para capturar el byte de orden inferior de la dirección durante el acceso a la memoria externa. Esto permite utilizar direcciones de 16 bits. La figura 15.24 ilustra esto.

3. *PSEN*
   La terminal para la *activación del almacenamiento del programa* (PSEN, *program store enable*) es la terminal de la señal de lectura para la memoria de programa externa y está activa cuando su valor es bajo. Está conectada con la terminal de activación de salida de una ROM o una EPROM externas.

4. *EA*
   El microprocesador toma el valor bajo de la terminal de *acceso externo* (EA, *external access*) cuando sólo quiere acceder al código de programa externo; cuando toma su valor alto, en forma automática accede al código interno o externo, dependiendo de la dirección. Así, en el primer reinicio del 8051, el contador del programa inicia en $0000 y apunta a la primera instrucción de

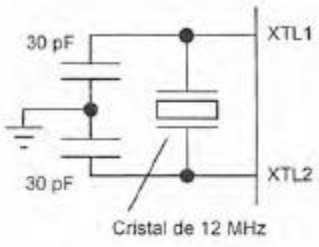


Figura 15.25 Cristal

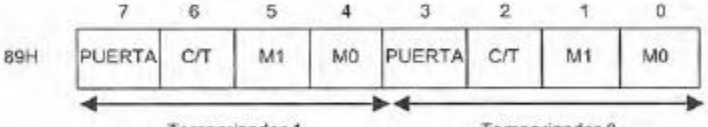
programa en el código de memoria interna a menos que EA se mantenga bajo. Luego el CPU manda un bajo en PSEN para habilitar el uso del código de memoria externo. Esta terminal se usa también, en los microprocesadores con EPROM, para recibir el voltaje de programación para programar los EPROM.

5. *XTAL1, XTAL2*
Son las terminales de conexión de un oscilador de cristal o externo. La figura 15.25 ilustra como se usan con un cristal. La frecuencia de cristal más común es 12 MHz.

6. *RESET*
Cuando en esta conexión hay una señal alta se reinicia el microcontrolador.

7. *Entrada/salida serial*
Escribir en el buffer de datos serial SBUF en la dirección 99H carga los datos para transmisión; leer el SBUF accede a los datos recibidos. El registro direccionable por bit del registro de control del puerto serial SCON en la dirección 98H se usa para controlar los diferentes modos de operación. Vea en el capítulo 10 un análisis de estas interfases.

8. *Tiempos*
El registro de modo del temporizador TMOD en la dirección 89H se usa para fijar los modos de operación para los temporizadores 0 y 1 (figura 15.26). Se carga como una entidad y no es direccionable por bit. El registro de control del temporizador TCON (figura 15.27) contiene los bits de estado y control para los remporizadores 0 y 1. Los cuatro bits superiores se usan para encender y apagar los temporizadores y para indicar saturación del temporizador. Los bits inferiores no tienen que ver con los temporizadores y se usan para detectar e iniciar interrupciones externas.



Puerta 0 = temporizador corre cuando cualquiera TR0/TR1 se fija
1 = temporizador corre sólo cuando INT0/INT1 es alto junto con TR0/TR1
C/T: selector del contador/temporizador
0 = entrada del reloj del sistema, 1 = entrada de TX0/TX1
M0 y M1 fijan el modo

M1	M0	Modo	
0	0	0	contador de 13 bit, 5 inferiores de TL0 y los 8 de TH
0	1	1	contador de 16 bit
1	0	2	temporizador/contador de 8 bit autorecargable
1	1	3	TL0 es un temporizador/contador de 8 bit controlado por los bits de control del temporizador 0. TL0 es un temporizador/contador de 8 bit controlado por los bits de control del temporizador 1. Temporizador 1 está apagado.

Figura 15.26 Registro TMOD

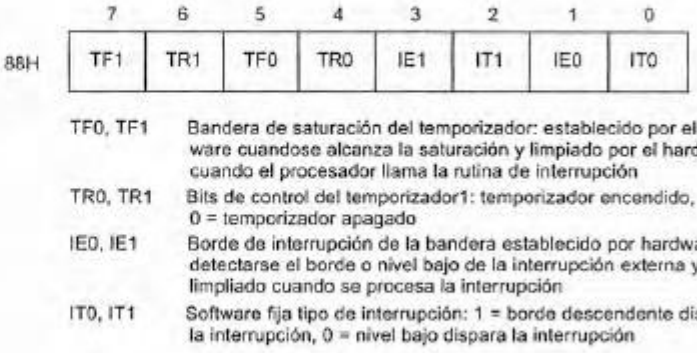


Figura 15.27 Registro TCON

La fuente de los bits contados por cada temporizador se fija por el bit C/T; si el bit es bajo la fuente es el reloj del sistema dividido entre 12 y si es alto se fija para contar de una fuente externa. Los temporizadores se arrancan fijando TR0 o TR1 a 1 y se detienen haciéndolos 0. Otra forma de controlar los temporizadores es fijando GATE a 1, esto permite que el temporizador sea controlado por la terminal INT0 o INT1 del temporizador al hacerse 1. De esta forma un dispositivo externo conectado a estas terminales del microcontrolador puede controlar el encendido/apagado del contador.

9. *Interrupciones*
Las interrupciones fuerzan al programa a llamar una subrutina localizada en una dirección específica de memoria; esto se logra escribiendo en el registro de habilitación de interrupción IE en la dirección A8H (figura 15.28).

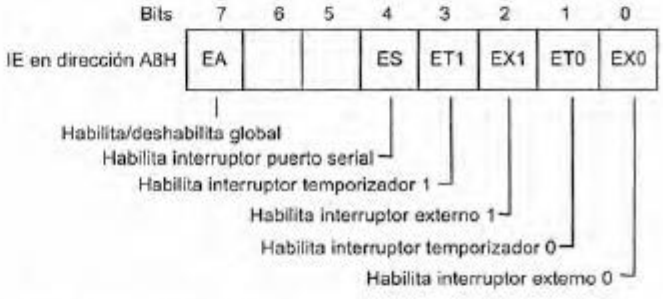


Figura 15.28 Registro IE

8D	TH1	F0	B
8C	TH0	E0	ACC
8B	TL1	D0	PSW
8A	TL0	B8	IP
89	TMOD	B0	P3
88	TCON	A8	IE
87	PCON	A0	P2
83	DPH	99	SBUF
82	DPL	98	SCON
81	SP	90	P1
80	P0		

Figura 15.29 Registros

El término *registros de función especial* se usa para los registros de control de entrada/salida (figura 15.29), como el IE descrito antes, estos se localizan en las direcciones 80 a FF. El acumulador A (ACC) es el registro más grande usado para operaciones con datos; el registro B se usa para multiplicación y división. P0, P1, P2 y P3 son los registos de captura para los puertos 0, 1, 2 y 3.

Para profundizar, se sugiere consultar los manuales del fabricante, o libros como: *Programming and Interfacing the 8051 Microcontroller* de S. Yeralan y A. Ahluwalia (Addison-Wesley, 1993), *The 8051 Microcontroller* de I. Scott MacKenzie (Prentice Hall 1999, 1985, 1992) o *The 8051 Family of Microcontrollers* de R.H. Barnett (Prentice-Hall, 1995).

15.3.3 Microcontroladores de Microchip

Otra familia de microcontroladores de 8 bits muy empleada es la del Microchip PIC16C6x/7x. Usan el término PIC (*peripheral interface controller*) para designar sus microcontroladores de un solo chip. Estos utilizan una forma de arquitectura llamada arquitectura Harvard. Con ella las instrucciones son enviadas desde la memoria del programa utilizando buses distintos a los empleados para las variables de acceso (figura 15.30). Los otros microcontroladores tratados en este capítulo no tienen buses separados y los datos del programa deben esperar a la lectura/escritura de variables y a las operaciones de entrada/salida antes de recibir instrucciones de la memoria. Con la arquitectura Harvard las instrucciones se pueden enviar cada ciclo sin esperar, cada instrucción se puede ejecutar cada ciclo después de su envío. La arquitectura Harverd permite permite una operación más rápida para una frecuencia de reloj dada. La figura 15.31 muestra las conexiones de una de las versiones del controlador PIC16C74A y la figura 15.32 su arquitectura.

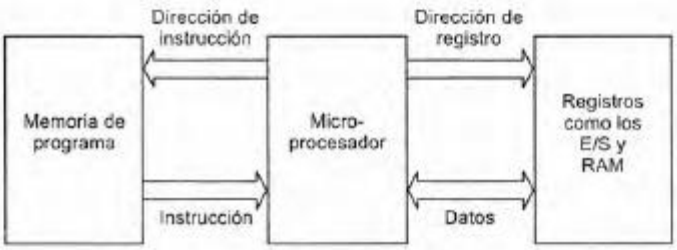


Figura 15.30 Arquitectura Harvard

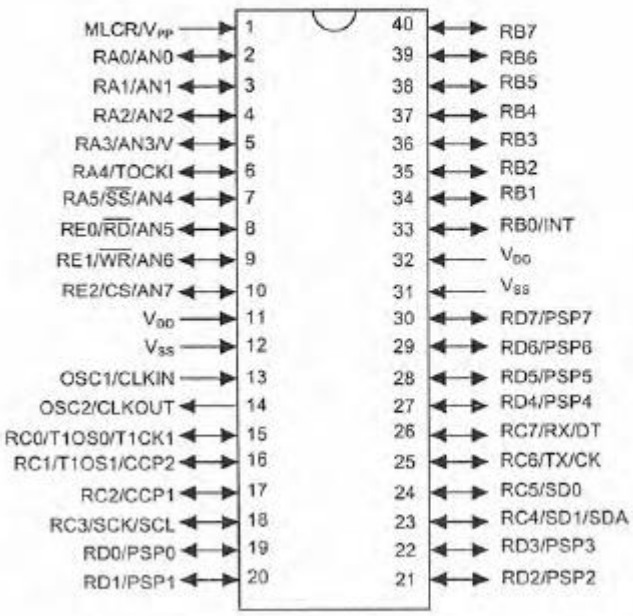


Figura 15.31 PIC16C74A

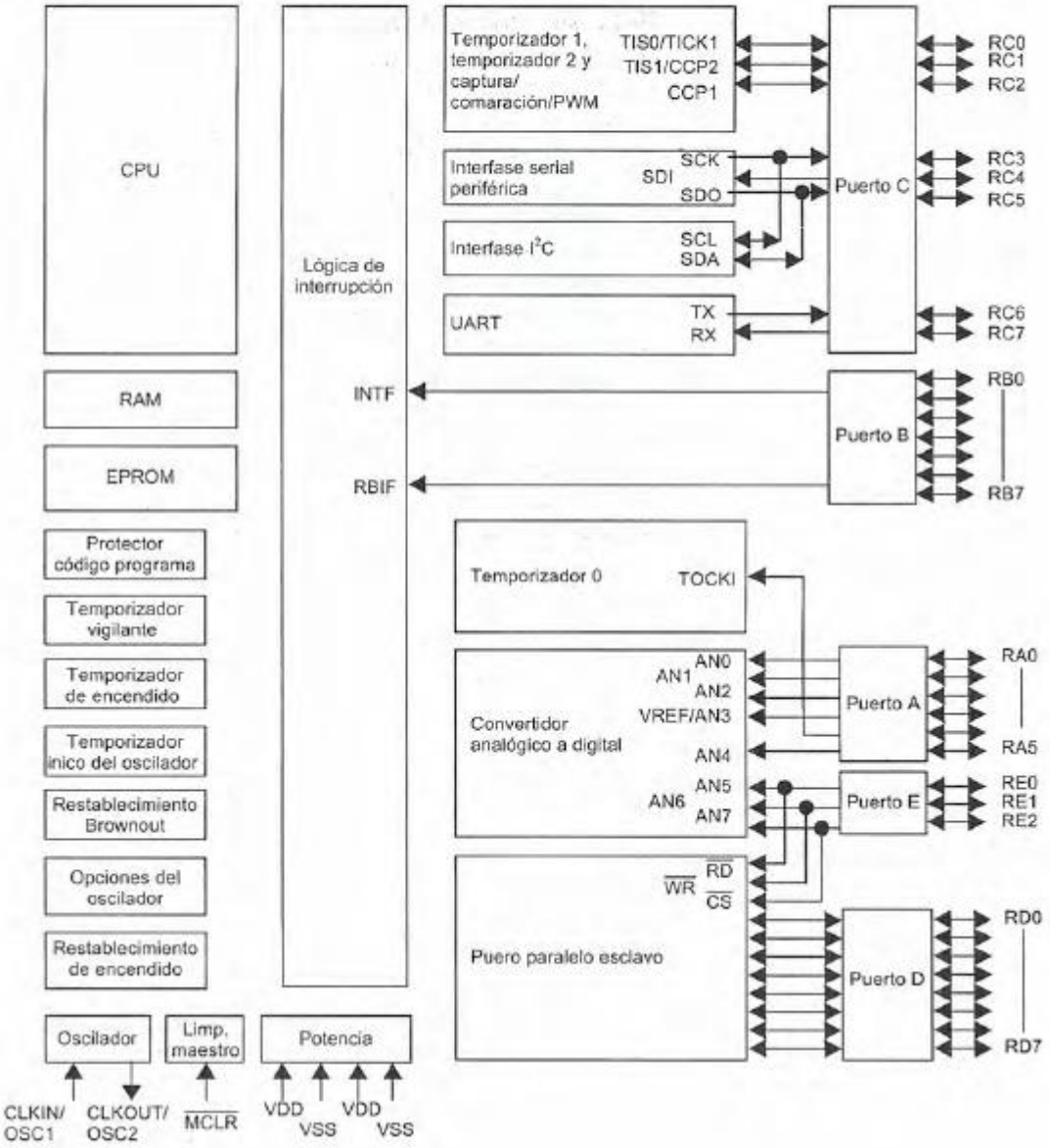


Figura 15.32 PIC16C74/74A

Las características básicas de los microcontroladores son:

1. *Puertos entrada/salida*
   Las terminales 2, 3, 4, 5, 6 y 7 corresponden al puerto A de entrada/salida bidireccional. Como los otros puertos bidireccionales las señales se leen y escriben usando los registros del puerto. La dirección de las señales se controla con los registros de dirección TRIS; hay un TRIS para cada puerto. El TRIS se fija en 1 para lectura y 0 para escritura (figura 15.33).

Figura 15.33 Dirección del puerto

Las terminales 2, 3, 4 y 5 se pueden usar como entradas analógicas, la terminal 6 para una entrada de reloj al temporizador 0; la terminal 7 puede ser la esclava seleccionada para el puerto serial sincrónico (vea más adelante en esta sección).

Las terminales 33, 34, 35, 36, 37, 38, 39 y 40 sirven como puerto B de entrada/salida bidireccional; la dirección de las señales se controla mediante su registro de dirección TRIS correspondiente. La terminal 33 también puede ser terminal de interrupción externa. Las terminales 37, 38, 39 y 40 también funcionan como terminales para interrupciones cuando hay cambios. La terminal 39 también es el reloj de programación en serie y la terminal 40 para los datos de programación en serie.

Las terminales 15, 16, 17, 18, 23, 24, 25 y 26 son para el puerto C de entrada/salida bidireccional; la dirección de las señales se controla mediante su registro de dirección TRIS correspondiente. La terminal 15 se puede utilizar como salida del temporizador 1 o como entrada de reloj del temporizador 1. La terminal 16 es entrada del oscilador del temporizador 1 o entrada de la captura 2/salida de la comparación 2/la salida de la PWM2.

Las terminales 19, 20, 21, 22, 27, 28, 29 y 30 son para el puerto D de entrada/salida bidireccional; la dirección de las señales se controla por su registro de dirección TRIS correspondiente.

Las terminales 8, 9 y 10 corresponden al puerto E de entrada/salida bidireccional; la dirección de las señales se controla mediante su registro de dirección TRIS. La terminal 8 también puede ser el control de lectura del puerto paralelo esclavo, o para la entrada analógica 5. El puerto paralelo esclavo es un elemento que facilita el diseño de los circuitos de interfase con computadoras personales, cuando en una aplicación las terminales de los puertos D y E se asignan a esta operación.

Bits ADCON0			
5	*4*	*3*	*Para entrada analógica encendida*
0	0	0	Puerto A, bit 0
0	0	1	Puerto A, bit 1
0	1	0	Puerto A, bit 2
0	1	1	Puerto A, bit 3
1	0	0	Puerto A, bit 5
1	0	1	Puerto E, bit 0
1	1	0	Puerto E, bit 1
1	1	1	Puerto E, bit 2

2. *Entradas analógicas*
Las terminales 2, 3, 4, 5, 6 y 7 del puerto A y las terminales 8, 9 y 10 del puerto E se pueden usar como entradas analógicas alimentadas a través de un convertidor analógico a digital interno. Los registros ADCON1 y TRISA para el puerto A (TRISE para el puerto E) deben inicializarse para seleccionar el voltaje de referencia que se usará en la conversión y seleccionar los canales como entradas. El ADCON0 debe inicializarse como aparece en la tabla anexa.

3. *Temporizadores*
El microcontrolador tiene tres temporizadores: temporizador 0, temporizador 1 y temporizador 2. El temporizador 0 es un contador de 8 bits en el cual es posible escribir o leer y que puede usarse para contar transiciones de señal externa, generando una interrupción cuando ha ocurrido el número de eventos requeridos. La fuente para el conteo puede ser la señal del reloj interna o una señal digital externa. La selección de la fuente de conteo se hace mediante el bit TOCS en el registro OPTION (figura 15.34).

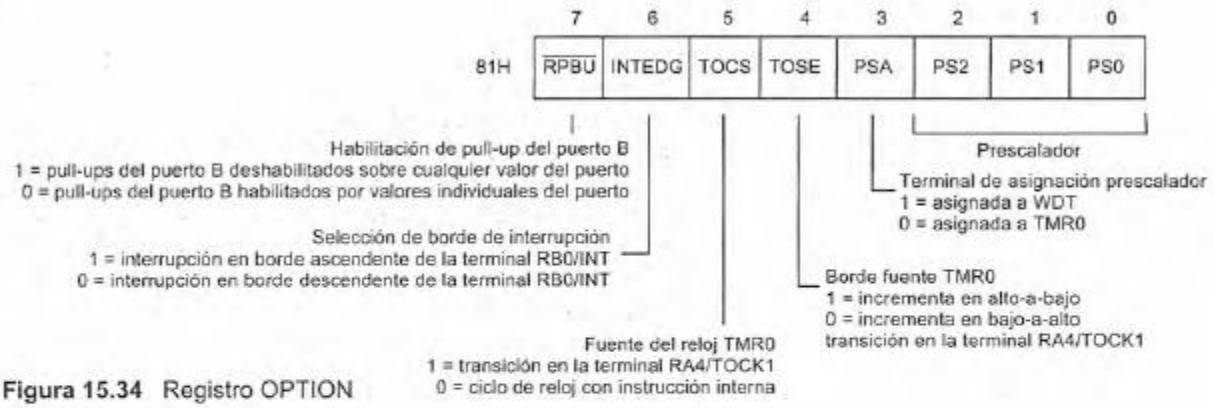


Figura 15.34 Registro OPTION

Si el prescalador no se selecciona la cuenta se incrementa cada dos ciclos de la fuente de entrada. Se usa el prescalador para que la señal pase al contador después de otro número fijo de ciclos de reloj. Enseguida se muestran algunas relaciones de escala posibles. WDT da los factores de escala seleccionados cuando se utiliza un temporizador vigilante. Se usa para para terminar el conteo y reiniciar el sistema si la opreración no concluye en un tiempo razonable; el tiempo es normalmente 18 ms.

Valores de la terminal prescalar			*Relación TMR0*	*Relación WDT*
PS2	*PS1*	*PS0*		
0	0	0	1 : 2	1 : 1
0	0	1	1 : 4	1 : 2
0	1	0	1 : 8	1 : 4
0	1	1	1 : 16	1 : 8
1	0	0	1 : 32	1 : 16
1	0	1	1 : 64	1 : 32
1	1	0	1 : 128	1 : 64
1	1	1	1 : 256	1 : 128

El temporizador 1 es el más versatil de los temporizadores y se puede utilizar para monitorear los tiempos entre señales de transición en una terminal de entrada o controlar los tiempos exactos de transiciones en una terminal de salida. Cuando se usa en los modos de captura o compara permite al microcontrolador controlar los tiempos de salida en la terminal 17.

El temporizador 2 puede usarse para contolar el periodo de un impulso de salida con amplitud modulada (PWM). Los impulsos de salida con amplitud modulada se proporcionan por las terminales 16 y 17.

Dir. arch.	Banco 0	Banco 1	Dir. arch.
00h	INDF	INDF	80h
01h	TMR0	OPTION	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	PORTC	TRISC	87h
08h	PORTD	TRISD	88h
09h	PORTE	TRISE	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	PIR1	PIE1	8Ch
0Dh	PIR2	PIE2	8Dh
0Eh	TMR1L	PCON	8Eh
0Fh	TMR1H		8Fh
10h	T1CON		90h
11h	TMR2		91h
12h	T2CON	PR2	92h
13h	SSPBUF	SSPADD	93h
14h	SSPCON	SSPSTAT	94h
15h	CCPR1L		95h
16h	CCPR1H		96h
17h	CCP1CON		97h
18h	RCSTA	TXSTA	98h
19h	TXREG	SPBRG	99h
1Ah	RCREG		9Ah
1Bh	CCPR2L		9Bh
1Ch	CCPR2H		9Ch
1Dh	CCPR2CON		9Dh
1Eh	ADRES		9Eh
1Fh	ADCON0	ADCON1	9Fh
20h	Registros de propósito general	Registros de propósito general	A0h
7Fh			FFh

Figura 15.36 Registros de propósito especial

4. *Entrada/salida serial*
Los microcontroladores PIC incluyen un módulo de puerto serial sincrónico SSP y un modulo de interfase serial de comunicaciones (SCI). La terminal 18 tiene las funciones alternativas de la entrada del reloj serial sincrónico o la salida para el modo de interfase periférica serial (SPI) y el modo I^2C. El bus I^2C proporciona una interfase de doble alambre bidireccional que puede usarse con muchos otros chips; también se puede usar para conectar un microcontrolador maestro a microcontroladores esclavos. UART, o sea, el receptor transmisor universal asíncrono, se puede usar para crear una interfase serial con una computadora personal.

5. *Puerto paralelo esclavo*
El puerto paralelo esclavo usa los puertos D y E y habilita al microprocesador para proporcionar una interfase con una PC.

6. *Entrada del cristal*
La terminal 13 es para la entrada del cristal del oscilador o la entrada de una fuente externa de reloj; la terminal 14 es la salida del cristal del oscilador. La figura 15.35a muestra el arreglo necesario para un control de frecuencia preciso. La figura 15.35b muestra una solución para el control de frecuencia de poco costo; para una frecuencia de 4 MHz tendríamos $R = 4.7\ k\Omega$ y $C = 33$ pF. La relación interna del reloj es la frecuencia del oscilador dividida entre 4.

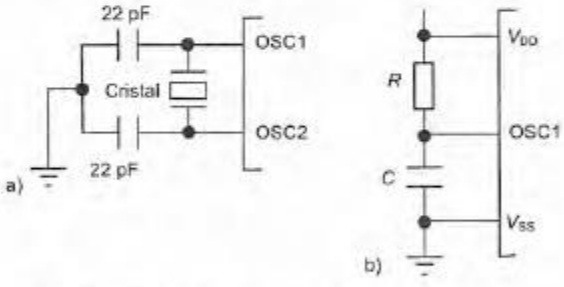


Figura 15.35 Control de frecuencia

7. *Borrado maestro*
La terminal 1 es el borrador maestro, esto es, entrada de restablecimiento y requiere un valor bajo para restablecer la unidad.

Los *registros de propósito especial* (figura 15.36) se usan para control de entrada/salida, como se ilustró para algunos de estos registros. Los registros para el PIC16C73/74 están en dos bancos y antes de seleccionar un registro en particular se debe escoger el banco fijando un bit en el registro de estado (figura 15.37).

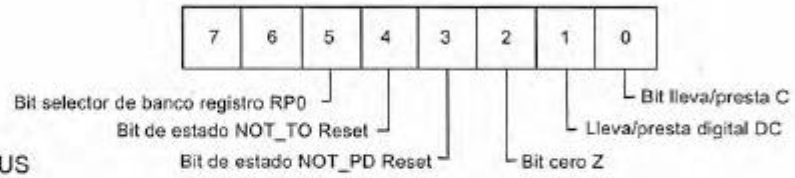


Figura 15.35 Registro STATUS

Si desea más detalles de los microcontroladores PIC se sugieren las publicaciones del fabricante o libros como *Design with PIC Microcontrollers* de J.B. Peatman (Prentice-Hall, 1998).

15.3.4 Selección de un microprocesador

Al elegir un microcontrolador se deben considerar los siguientes factores:

1. *Número de puertos de entrada/salida*
   ¿Cuántos puertos de entrada/salida son necesarios para realizar la tarea respectiva?

2. *Interfaces necesarias*
   ¿Qué interfaces se necesitan? Por ejemplo, ¿se requiere una modulación por ancho de pulso? Muchos microcontroladores tienen salidas PWM, por ejemplo, el PIC17C42 tiene dos.

3. *Necesidades de memoria*
   ¿Qué capacidad de memoria se necesita para realizar la tarea?

4. *Cantidad de interrupciones necesarias*
   ¿Cuántos eventos de interrupción se requieren?

5. *Velocidad de procesamiento requerida*
   El microprocesador requiere tiempo para ejecutar una instrucción (vea la sección 16.2.2), este tiempo está definido por el reloj del procesador.

Como ejemplo de la variación en los microcontroladores disponibles, la tabla 15.1 muestra detalles de algunos de la familia Intel 8051; la tabla 15.2, de la familia PIC16Cxx y la tabla 15.3 de la familia M68HC11.

Tabla 15.1 Características de miembros de la familia Intel 8051

	ROM	*EEPROM*	*RAM*	*Temporizadores*	*Puertos E/S*	*Interrupciones*
8031AH	0	0	128	2	4	5
8051AH	4K	0	128	2	4	5
8052AH	8K	0	128	3	4	6
8751H	0	4K	128	2	4	5

Tabla 15.2 Características de miembros de la familia PIC

	E/S	*EEPROM*	*RAM*	*Canales CAD*	*USART*	*Módulos CCP*
PIC16C62A	22	2K	128	0	0	1
PIC16C63	2	4K	192	0	1	2
PIC16C64A	33	2K	128	0	0	1
PIC16C65A	33	4K	192	0	1	2
PIC16C72	22	2K	128	5	0	1
PIC16C73A	22	4K	192	5	1	2
PIC16C74A	33	4K	192	8	1	3

Tabla 15.3 Características de miembros de la familia M68HC11

	ROM	*EEPROM*	*RAM*	*ADC*	*Temporizador*	*PWM*	*E/S*	*Serial*	*Reloj E MHz*
68HC11A0	0	0	256	8 can., 8-bit	(1)	0	22	SCI, SPI	2
68HC11A1	0	512	256	8 can., 8-bit	(1)	0	22	SCI, SPI	2
68HC11A7	8 K	0	256	8 can., 8-bit	(1)	0	38	SCI, SPI	3
68HC11A8	8 K	512	256	8 can., 8-bit	(1)	0	38	SCI, SPI	3
68HC11C0	0	512	256	4 can., 4-bit	(2)	2 can., 8-bit	36	SCI, SPI	2
68HC11D0	0	0	192	Ninguno	(2)	0	14	SCI, SPI	2

Temporizador: (1) captura de 3 entradas, comparación de 5 salidas, interrupción en tiempo real, temporizador del controlador de secuencia, acumulador de impulsos; (2) captura de 3 o 4 entradas, comparación de 5 o 4 salidas, interrupción en tiempo real, temporizador del controlador de secuencia, acumulador de impulsos. En serie: SCI es una interfase para comunicaciones en serie asíncrona, SPI es una interfase para dispositivos periféricos en serie y síncrona.

15.4 Aplicaciones

Los siguientes son dos ejemplos de cómo se utilizan los microcontroladores. En el capítulo 22 se presentan más casos.

15.4.1 Sistema para medición de temperatura

Para ilustrar en forma breve cómo se puede usar un microcontrolador, la figura 15.38 muestra los principales elementos de un sistema de medición de temperatura que usa un MC68HC11. El sensor de temperatura da un voltaje proporcional a la temperatura (por ejemplo, un termotransistor como el LM35; vea la sección 2.9.4). La salida del sensor de temperatura se conecta a la línea de entrada del ADC del microcontrolador. Éste se programa para convertir la temperatura en una salida BCD con la que se conmutan los elementos de un display de dos dígitos de siete segmentos. Sin embargo, como la temperatura puede fluctuar, es necesario utilizar un registro de memoria para guardar los datos suficiente tiempo para permitir su lectura en el display. El registro de almacenamiento, 74HCT273, es un flip-flop octal tipo D que se reinicia durante el siguiente flanco de elevación positiva de la entrada de reloj del microcontrolador.

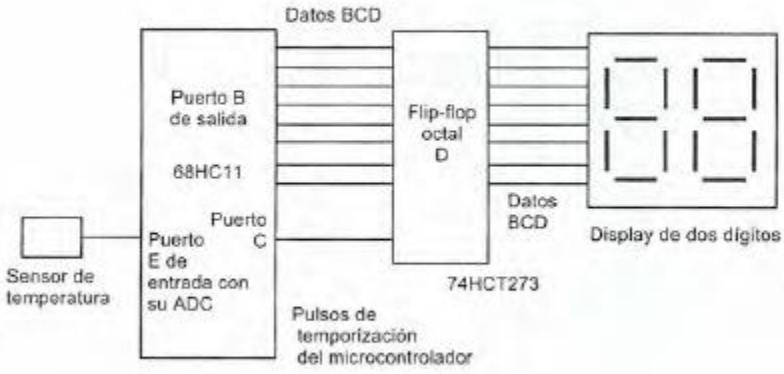


Figura 15.38 Sistema de medición de temperatura

15.4.2 Lavadora doméstica

La figura 15.39 muestra cómo emplear un microcontrolador para operar una lavadora doméstica. El microcontrolador más común es el M68HC05B6 de Motorola; por ser más barato y sencillo que el microcontrolador M68HC11 Motorola mencionado en este capítulo, que se utiliza mucho en aplicaciones de bajo costo.

Las señales de los sensores de la temperatura del agua y la velocidad del motor entran por el puerto de entrada analógico a digital. El puerto A proporciona las salidas para los diversos actuadores que se usan para controlar la máquina y también la entrada del interruptor del nivel del agua. El puerto B da las salidas para el display. El puerto C produce las salidas para el display y también recibe las señales de entrada del teclado que se usa para ingresar en la máquina las selecciones del programa. La sección PWM del temporizador proporciona una modulación por ancho de pulsos para controlar la velocidad del motor. El programa de la máquina se interrumpe y detiene si se abre la puerta de la lavadora.

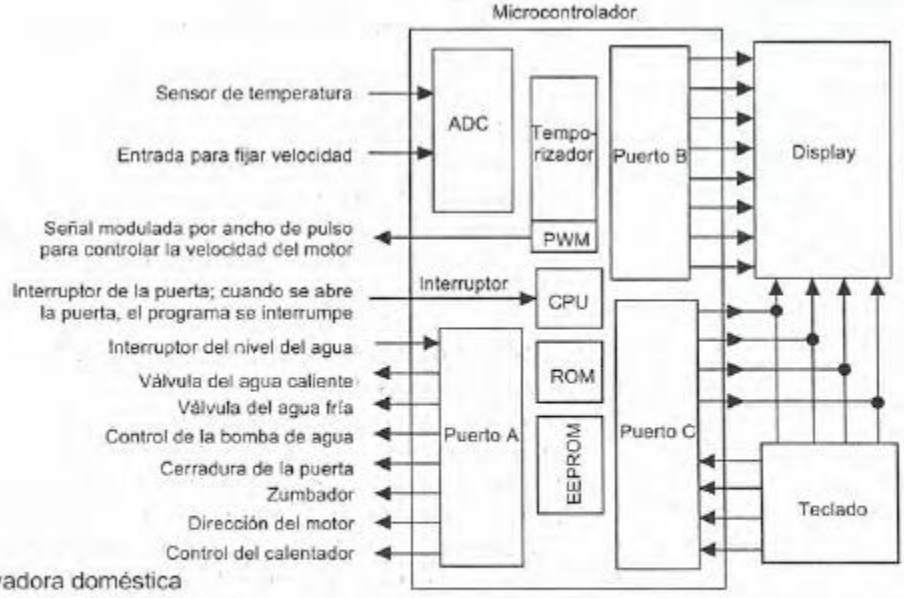


Figura 15.39 Lavadora doméstica

15.5 Programación

Figura 15.40 Símbolos para los diagramas de flujo

Figura 15.41 Diagrama de flujo

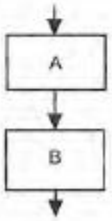


Figura 15.42 Secuencia

Un método de uso común para diseñar programas es el siguiente:

1. Definir el problema, indicando con claridad qué función se espera que ejecute el programa, las entradas y salidas requeridas, las restricciones de velocidad de operación, exactitud, capacidad de memoria, etcétera.

2. Definir el algoritmo. Un *algoritmo* es la secuencia de pasos que definen el método de solución del problema.

3. En sistemas con menos de mil instrucciones, es útil representar el algoritmo mediante un *diagrama de flujo*. La figura 15.40 muestra los símbolos más comunes de estos diagramas. Cada paso del algoritmo se representa por uno o varios de esos símbolos y se unen con líneas que representan el flujo del programa. Otra herramienta de diseño útil es el *seudocódigo*. Éste es una forma de describir los pasos de un algoritmo de una manera informal, que después se puede traducir en un programa.

4. Traducir el diagrama de flujo/algoritmo a instrucciones que el microprocesador pueda ejecutar. Para ello, se escriben las instrucciones en algún lenguaje, por ejemplo, lenguaje ensamblador o C, y luego se convierten ya sea en forma manual o mediante un programa ensamblador, en un código aceptable para el microprosesador, esto es, código de máquina.

5. Probar y depurar el programa. Los errores del programa se conocen como *defectos* o *errores de programa*, y el proceso de rastreo y eliminación se llama *depuración*.

Los diagramas de flujo y el seudocódigo son auxiliares para el diseño de programas sistemáticos y estructurados. La figura 15.41 muestra parte de un diagrama de flujo donde, después del inicio del programa, aparece una operación A seguida de una bifurcación que conduce a la operación B o a la operación C, dependiendo de si la respuesta a la pregunta es sí o no.

15.5.1 Seudocódigo

El seudocódigo se parece a dibujar un diagrama de flujo e implica escribir un programa como una secuencia de funciones u operaciones con el elemento IF-THEN-ELSE (SÍ-ENTONCES-O) y el elemento de repetición WHILE-DO (EN TANTO QUE-HACER). Una secuencia (figura 15.42) se escribiría como:

```
BEGIN A
...
END A
...
```

```
BEGIN B
  ...
END B
```

y una decisión como:

```
IF X
THEN
  BEGIN A
    ...
  END A
ELSE
  BEGIN B
    ...
  END B
ENDIF X
```

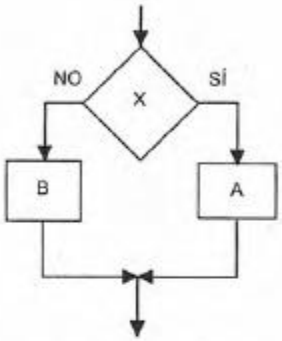


Figura 15.43 SÍ-ENTONCES-O

La figura 15.43 muestra este tipo de decisión en un diagrama de flujo. Una repetición se escribe como:

```
WHILE X
DO
  BEGIN A
    ...
  END A
  BEGIN B
    ...
  END B
ENDO WHILE X
```

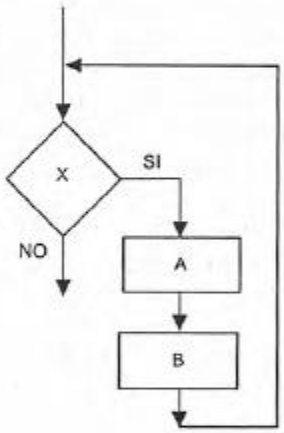


Figura 15.44 EN TANTO (QUE)-HACER

La figura 15.44 ilustra WHILE-DO (EN TANTO (QUE)-HACER) como un diagrama de flujo. Un programa escrito de esta manera sería el siguiente:

```
BEGIN PROGRAM
  BEGIN A
    IF X
      BEGIN B
      END B
    ELSE
      BEGIN C
      END C
    ENDIF X
  END A
  BEGIN D
    IF Z
      BEGIN E
      END E
    ENDIF Z
  END D
```

En el capítulo 16 se mostrará cómo elaborar programas en lenguaje ensamblador y en el capítulo 17 en lenguaje C.

Problemas

1. En el caso de un microprocesador explique el papel de a) un acumulador, b) el estado, c) la dirección de memoria, d) los registros del contador de programa.
2. En un microprocesador se utilizan ocho líneas de dirección para acceder a la memoria. ¿Cuál será la cantidad máxima de ubicaciones de memoria a las que se puede acceder?
3. Un chip de memoria tiene 8 líneas de datos y 16 líneas de dirección. ¿Cuál es su capacidad?
4. ¿Cuál es la diferencia entre un microcontrolador y un microprocesador?
5. Dibuje un diagrama de bloques de un microcontrolador básico y explique la función de cada subsistema.
6. ¿Qué puertos del M68HC11 se utilizan para a) un convertidor A/D, b) un puerto bidireccional, c) una entrada/salida serial, d) funcionar como puerto de sólo salida de 8 bits?
7. ¿Cuántos bits de memoria tiene el M68HC11A7 para la memoria de datos?
8. En el M68HC11 de Motorola el puerto C es bidireccional. ¿Cómo se debe configurar para que funcione como a) entrada, b) salida?
9. El M68HC11 de Motorola se puede utilizar en un solo chip y en modo ampliado. ¿Cuál es el propósito de estos modos?
10. ¿Para qué se utiliza la conexión ALE del 8051 de Intel?
11. ¿Qué entrada se requiere para restablecer el microcontrolador 8051 de Intel?
12. Represente en seudocódigo lo siguiente:
    a) Si A es sí, entonces B o bien C.
    b) En tanto que A es sí, hacer B.

16 Lenguaje ensamblador

16.1 Lenguajes

Con el término *software* se designan todas las *instrucciones* con las que se indica a un microprocesador o microcontrolador qué hacer. El repertorio de instrucciones que el microprocesador reconoce se denomina *conjunto de instrucciones*. Su forma dependerá del microprocesador que se utilice. El conjunto de instrucciones necesarias para llevar a cabo una tarea dada se llama *programa*.

Los microprocesadores trabajan en código binario. Las instrucciones escritas en código binario se conocen como *código de máquina*. Escribir programas en este código es un proceso tedioso que requiere habilidad; está sujeto a errores, dado que el programa es una serie de ceros y unos y no es fácil comprender el significado de las instrucciones con sólo observar la secuencia. Una alternativa es utilizar un código taquigráfico de fácil comprensión para representar las secuencias de 0 y 1. Por ejemplo, agregar datos a un acumulador puede representarse como ADDA. Este código taquigráfico se conoce como *código mnemónico*, y es un código 'auxiliar para la memorización'. Este tipo de código se conoce como *lenguaje ensamblador*. Escribir un programa utilizando mnemónicos es más sencillo, porque son una versión abreviada de la operación que realiza una instrucción. También, dado que las instrucciones describen las operaciones del programa, se facilita su comprensión y se reduce la posibilidad de cometer errores, comparado con las secuencias binarias de la programación en código de máquina. Sin embargo, todavía debe convertirse el programa ensamblador en código de máquina, ya que sólo éste reconoce el microprocesador. Esta conversión se puede hacer a mano, usando las hojas de especificaciones del fabricante que dan el código binario para cada mnemónico. También existen programas de cómputo para hacer la conversión, estos programas se conocen como *compiladores para lenguaje ensamblador*.

Los *lenguajes de alto nivel* proporcionan un tipo de lenguaje de programación que describe de forma más cercana y más accesible el tipo de operaciones que se requieren. Ejemplos de estos lenguajes son BASIC, C, FORTRAN y PASCAL. Sin embargo, aún es necesario convertir estos lenguajes a código de máquina usando un compilador, para que el microprocesador lo pueda utilizar. En este capítulo

se presenta un panorama general de cómo elaborar programas utilizando lenguaje ensamblador; en el capítulo 17 se usa lenguaje C.

16.2 Conjuntos de instrucciones

Las siguientes son las instrucciones más comunes que se dan a los microprocesadores; la lista completa de estas instrucciones se conoce como *conjunto de instrucciones*. En general, las instrucciones se clasifican en:

1. Transferencia de datos
2. Aritméticas
3. Lógicas
4. Control del programa

El conjunto de instrucciones es distinto para cada procesador. No obstante, algunas instrucciones son razonablemente comunes para la mayoría de los microprocesadores. Estas instrucciones son:

Transferencia de datos

1. *Cargar* (*load*)
   Esta instrucción lee el contenido de la localidad de memoria especificada y lo copia a la localidad del registro especificado en la CPU; por ejemplo:

Antes de la instrucción	*Después de la instrucción*
Dato en la localidad de memoria 0010	Dato en la localidad de memoria 0010
	Dato tomado de 0010 en el acumulador

2. *Almacenar* (*store*)
   Esta instrucción copia el contenido de un registro especificado en una localidad de memoria dada; por ejemplo:

Antes de la instrucción	*Después de la instrucción*
Dato en el acumulador	Dato en el acumulador
	Dato copiado a la localidad de memoria 0011

Aritméticas

3. *Sumar* (*add*)
   Esta instrucción suma el contenido de una localidad de memoria especificada con los datos de algún registro; por ejemplo:

Antes de la instrucción	*Después de la instrucción*
Acumulador con dato 0001	Acumulador con dato 0011
Localidad de memoria con datos 0010	

4. *Decrementar* (*decrement*)
Esta instrucción resta 1 del contenido de una localidad especificada. Por ejemplo, supongamos que se tiene el acumulador en la localidad especificada:

Antes de la instrucción	*Después de la instrucción*
Acumulador con dato 0011	Acumulador con dato 0010

5. *Comparar* (*compare*)
Esta instrucción determina si el contenido de un registro es mayor, menor o igual que el contenido de una localidad de memoria dada. El resultado aparece en el registro de estado como una bandera.

Instrucciones lógicas

6. *Operación AND* (*AND*)
Esta instrucción aplica la operación lógica AND al contenido de la localidad de memoria especificada y los datos en un registro dado. A los números se les aplica la operacion bit por bit, por ejemplo:

Antes de la instrucción	*Después de la instrucción*
El acumulador con dato 0011	Acumulador con dato 0001
La localidad de memoria con dato 1001	

En los datos anteriores, sólo en el bit menos significativo hay un 1 en ambos conjuntos de datos y la operación AND sólo produce un 1 en el bit menos significativo del resultado.

7. *Operación or-exclusiva* (*exclusive-or*)
Esta instrucción aplica la operación lógica or-exclusiva al contenido de la localidad de memoria especificada y a los datos en un registro dado; la operación se realiza bit por bit.

8. *Corrimiento lógico* (*a la izquierda o a la derecha*)
Las instrucciones de corrimiento lógico producen el desplazamiento del patrón de bits en el registro, un espacio a la izquierda o a la derecha incluyendo un 0 al final del número. Por ejemplo, para el corrimiento lógico a la derecha se corre un 0 al bit más significativo y el bit menos significativo se desplaza a la bandera de acarreo del registro de estado.

Antes de la instrucción	*Después de la instrucción*
Acumulador con dato 0011	Acumulador con dato 0001 El registro de estado indica acarreo 1

9. *Corrimiento aritmético (a la izquierda o a la derecha)*
Las instrucciones de corrimiento aritmético producen el desplazamiento del patrón de bits en el registro una posición a la izquierda o a la derecha, pero se conserva el bit de signo en la extrema izquierda del número; por ejemplo, en un desplazamiento aritmético a la derecha:

Antes de la instrucción	*Después de la instrucción*
Acumulador con dato 1011	Acumulador con dato 1101 El registro de estado indica acarreo 1

10. *Rotación (a la izquierda o a la derecha) (rotate)*
Las instrucciones de rotación producen el desplazamiento del patrón de bits en el registro una posición a la izquierda o a la derecha y el bit que sale sobrando se escribe ahora en el otro extremo; por ejemplo, en una rotación a la derecha:

Antes de la instrucción	*Después de la instrucción*
Acumulador con dato 0011	Acumulador con dato 1001

Control del programa

11. *Salto (jump)*
Esta instrucción modifica la secuencia de ejecución del programa. En general, el contador del programa ocasiona que se ejecuta de manera secuencial, en estricta secuencia numérica. Ahora bien, con una instrucción de salto el contador del programa pasa a una localidad especificada del programa. Por ejemplo, supongamos que el programa requiere la siguiente secuencia de instrucciones:

Decrementa el acumulador
Salta, si el acumulador no es cero, a la instrucción ...

12. *Ramificación o control de flujo (branch)*
Es una instrucción condicional cuyas opciones son *ramificación si el valor es cero* hacia una dirección o *ramificación si el valor es positivo* hacia otra dirección. Esta instrucción se ejecuta si se cumplen las condiciones requeridas. Por ejemplo, un programa que requiera la secuencia de instrucciones del diagrama de flujo de la figura 16.1.

Figura 16.1 Ejemplo de una ramificación

13. *Paro (halt)*
Esta instrucción detiene la actividad del microprocesador.

Los tipos de mnemónicos utilizados como instrucciones de lenguaje ensamblador dependerán del microprocesador o microcontrolador utilizado. El apéndice C muestra los conjuntos de instrucciones del microcontrolador M68HC11, Intel 8051 y PIC16Cxx. Los detalles completos de este u otro conjunto de instrucciones se encuentran en las publicaciones del fabricante.

Los datos numéricos pueden estar en binario, octal, hexadecimal o decimal. En general, en ausencia de cualquier indicador el ensamblador supone que el número está en decimal. Con dispositivos de Motorola, un número se indica con el prefijo #, un número binario está precedido por % o es seguido por B; un número en octal está precedido por @ o seguido por O; un número hexadecimal está precedido por $ o seguido por H; y un número en decimal no requiere indicación de letra o símbolo. Con dispositivos de Intel, los valores numéricos deben estar precedidos por un # para indicar un número y por B para binarios, O o Q para octales, H para hexadecimales y D o nada para decimales. Con microcontroladores PIC el archivo de encabezado tiene R = DEC para decimal por omisión. Entonces para números binarios el número está entre comillas precedido por B, y para números en hexadecimal por H.

16.2.1 Direccionamiento

Al utilizar un mnemónico, como LDA, para especificar una instrucción, estará seguido de información adicional, para especificar las fuentes y destinos de los datos que requiere la instrucción. Los datos que siguen a una instrucción se conocen como *operandos*.

Existen diversos métodos para especificar la localización de datos, es decir, el direccionamiento, y la manera en que el programa permite al microprocesador obtener sus instrucciones o datos. Los microprocesadores tienen diferentes modos de direccionamiento. El 68HC11 de Motorola tiene seis modos de direccionamiento, inmediato, directo, extendido, indexado, inherente y relativo; el 8051 de Intel tiene cinco modos de direccionamiento, inmediato, directo, a registro, indirecto e indexado; el microcontrolador PIC tiene tres modos, inmediato, directo e indirecto, con el modo indirecto se puede tener indexado. Los siguientes son los métodos más comunes:

1. *Direccionamiento inmediato*
   Los datos que siguen al mnemónico son el valor para operar y se usa para el cargado de un valor predeterminado en un registro o localidad de memoria. Por ejemplo, el código de Motorola LDA B #$25 significa carga el número 25 en el acumulador B. El símbolo # significa modo inmediato y un número, el símbolo $ que el número está en notación hexadecimal. Con el códico Intel se tendría MOV A,#25H para mover el número 25 al acumulador A. El símbolo # indica que es un número y la H, que el número es hexadecimal. Con el código de un PIC se tendría movlw H'25' para cargar el número 25 al registro de trabajo w, la H indica que es un número hexadecimal.

2. *Direccionamiento directo, absoluto, extendido o de página cero*
Con esta forma de direccionamiento el byte de datos que sigue al código de operación da directamente una dirección que define la localidad de los datos a usar en esa instrucción. Con Motorola el término *direccionamiento directo* se usa cuando la dirección dada es únicamente de 8 bits de longitud; el término *direccionamiento extendido* se usa cuando la dirección es de 16 bits. Por ejemplo, con el código Motorola, LDAA $25 significa carga en el acumulador el contenido de la localidad de memoria 0025, el 00 se supuso. Con el código Intel, para la misma operación, se puede tener la instrucción con direccionamiento directo MOVA,20H para copiar los datos de la dirección 20 al acumulador A. Con el código del PIC se tendría movwf Reg1 para copiar el contenido del Reg1 al registro de trabajo, la dirección del Reg1 se definió antes.

3. *Direccionamiento implicado o direccionamiento inherente*
Con este modo de direccionamiento, la dirección está implícita en la instrucción. Por ejemplo, con Motorola e Intel, el código CLR A significa limpia el acumulador A. Con el PIC, clrw significa limpia el registro de trabajo.

4. *Direccionamiento a registro*
Con esta forma de direccionamiento, el operando se especifica como el contenido de uno de los registros internos. Por ejemplo, con Intel, el código ADD R7,A se usa para sumar el contenido del acumulador al registro R7.

5. *Direccionamiento indirecto*
Esta forma de direccionamiento quiere decir que el dato va a encontrarse en una localidad de memoria cuya dirección está dada por la instrucción. Por ejemplo, con el sistema PIC se usan los registros INDF y FSR. La dirección se escribe primero en el registro FSR que sirve como un apuntador de dirección. Un acceso directo de INDF con la instrucción movf INDF,w cargará el registro de trabajo w usando el contenido de FSR como apuntador a la localidad del dato.

6. *Direccionamiento indexado*
Direccionamiento indexado significa que los datos están en una localidad de memoria cuya dirección se mantiene en un registro de índices. El primer byte de la instrucción contiene el código de operación y el segundo byte contiene el offset; el offset se suma al contenido del registro de índices para determinar la dirección del operando. Una instrucción de Motorola pudiera aparecer como LDA A $FF,X; esto quiere decir carga el acumulador A con lo datos que aparecen en la dirección dada por la suma del contenido del registro de índices y FF. Otro ejemplo es: STA A $05,X; esto significa almacenar el contenido del acumulador A en la dirección dada por contenido del registro índices más 05.

7. *Direccionamiento relativo*
   Este tipo de direccionamiento se usa con instrucciones de ramificación. El código operación está seguido por un byte llamado dirección relativa. Ésta indica el desplazamiento en direcciones que se tendrá que sumar al contador de programa si se presenta la ramificación. Por ejemplo, el código de Motorola, BEQ $F1 indica que si el dato es igual a cero, entonces la siguiente dirección en el programa es F1. La dirección relativa de F1 se suma a la dirección de la siguiente instrucción.

Como una ilustración, la tabla 16.1 muestra algunas instrucciones con los modos de direccionamiento utilizados en los sistemas de Motorola.

Tabla 16.1 Ejemplos de direccionamiento

Modo de direccionamiento	Instrucción	
Inmediato	LDA A #$F0	Cargar el acumulador A con el dato F0
Directo	LDA A $50	Cargar el acumulador A con datos en la dirección 0050
Extendido	LDA A$0F01	Cargar el acumulador A con datos en la dirección 0F01
Indexado	LDA A $CF,X	Cargar el acumulador con datos en la dirección dada por la suma del registro de índice más CF
Implícito	CLR A	Borrar acumulador A
Extendido	CLR $2020	Borrar dirección 2020, es decir, guardar todos los 0 en dirección 2020
Indexado	CLR $10,X	Borrar la dirección dada por el registro de índice más 10, es decir, guardar todos los 0 en esa dirección

16.2.2 Desplazamiento de datos

El siguiente es un ejemplo del tipo de información que se puede obtener en una hoja del conjunto de instrucciones de un fabricante (microprocesador 6800 de Motorola).

		Modos de direccionamiento					
		INMED.			DIRECTO		
Operación	Mnemónico	OP	~	#	OP	~	#
Sumar	ADDA	8B	2	2	9B	3	2

~ es el número de ciclos del microprocesador requeridos y # es el número de bytes de programa necesarios.

Esto significa que cuando se usa el modo de direccionamiento inmediato en este procesador, la operación Sumar se representa por el término mnemónico ADDA. El código de máquina para este direccionamiento es 8B y para obtener su expresión completa son necesarios dos ciclos. La operación requiere dos bytes en el programa. El

término *código de operación* se refiere a la instrucción que ejecutará el microprocesador y se expresa en forma hexadecimal. Un byte es un grupo de ocho dígitos binarios que el microprocesador reconoce como una palabra. Entonces, se necesitan dos palabras. En el direccionamiento directo el código de máquina es 9B y se requieren tres ciclos y dos bytes de programa.

Para ejemplificar cómo pasa la información entre la memoria y el microprocesador, considere las siguientes tareas. El direccionamiento de la memoria RAM para guardar un nuevo programa es sólo el más conveniente. En los siguientes ejemplos se usarán las direcciones que comienzan en 0010. Para emplear el direccionamiento directo, las direcciones deberán estar en la página cero, es decir, entre 0000 y 00FF. Los ejemplos se basan en el uso del conjunto de instrucciones del microprocesador M6800.

Tarea: introducir todos los ceros en el acumulador A.

Dirección de memoria	*Código de operación*	
0010	8F	CLR A

La siguiente dirección de memoria que se puede usar es 0011 dado que CLR A sólo ocupa un byte del programa. Éste es el modo de direccionamiento implícito.

Tarea: sumar al contenido del acumulador A el dato 20.

Dirección de memoria	*Código de operación*	
0010	8B 20	ADD A #$20

Aquí se utiliza el direccionamiento inmediato. La siguiente dirección de memoria que se puede utilizar es 0012, dado que en esta forma de direccionamiento, ADD A ocupa dos bytes de programa.

Tarea: cargar el acumulador A con los datos presentes en la dirección de memoria 00AF.

Dirección de memoria	*Código de operación*	
0010	B6 00AF	LDA A $00AF

Esto utiliza el direccionamiento absoluto. La siguiente dirección de memoria que se puede usar es 0013 porque, en este tipo de direccionamiento, LDA A ocupa tres bytes de programa.

Tarea: girar hacia la izquierda los datos que contiene la localidad de memoria 00AF.

Dirección de memoria	*Código de operación*	
0010	79 00AF	ROL $00AF

En este caso se utiliza el direccionamiento absoluto. La siguiente dirección de memoria que se puede usar es 0013 dado que ROL, en este modo, ocupa tres bytes de programa.

Tarea: guardar los datos que contiene el acumulador A en la localidad de memoria 0021.

Dirección de memoria	*Código de operación*	
0010	D7 21	STA A $21

Aquí se utiliza el direccionamiento directo. La siguiente dirección de memoria que se puede utilizar es 0012 porque STA A, en este modo, ocupa dos bytes de programa.

Tarea: si el resultado de la instrucción anterior es cero, avanzar cuatro lugares mediante bifurcación.

Dirección de memoria	*Código de operación*	
0010	27 04	BEQ $04

Se utiliza el direccionamiento relativo. Si el resultado no es cero, la siguiente dirección de memoria es 0012 porque BEQ, en este modo, ocupa dos bytes de programa. Si el resultado es cero, entonces la siguiente dirección es 0012 + 4 = 0016.

16.3 Programas en lenguaje ensamblador

Un programa en lenguaje ensamblador puede considerarse como una serie de instrucciones a un ensamblador, el cual produce el programa en código de máquina. Un programa escrito en lenguaje ensamblador consiste en una serie de instrucciones, una por línea. Una instrucción contiene de una a cuatro secciones o *campos*:

Etiqueta Código de operación Operando Comentario

Se utiliza un símbolo especial para indicar el inicio y el final de un campo; los símbolos empleados dependen del tipo de código de máquina del microprocesador. En el 6800 de Motorola se utilizan espacios. En el Intel 8080 aparecen dos puntos después de la etiqueta, un espacio después del código de operación, comas entre cada entrada del campo de direcciones y punto y coma antes de un comentario. En general, se usa punto y coma para separar los comentarios del operando.

La *etiqueta* es el nombre que recibe una entrada en la memoria. Las etiquetas están formadas por letras, números y algunos otros caracteres. En el 6800 de Motorola, las etiquetas tienen de uno a seis caracteres; el primero debe ser una letra, y no puede ser sólo la letra A, B o X ya que se reservan para referirse al acumulador o al registro de índices. En el 8080 de Intel se aceptan cinco caracteres, el primero debe ser una letra, @ o ?. La etiqueta no debe tener los nombres

reservados para los registros, códigos de instrucciones o seudoperaciones (vea más adelante en esta misma sección). Cada etiqueta en un programa debe ser única. Si no hay etiqueta, entonces se debe agregar un espacio en el campo de etiquetas. En el 6800 de Motorola un asterisco (*) en la etiqueta indica que la instrucción es un comentario, es decir, un comentario insertado para que el programa sea más claro. Como tal, el comentario se ignora en el ensamblador durante el proceso para obtener el programa en código de máquina .

El *código de operación* especifica cómo manejar los datos y se indica por su mnemónico; por ejemplo, LDA A. Este campo es el único que nunca puede estar vacío.

Además, el campo del código de operación puede contener directivas para el ensamblador. Éstas se conocen como *seudoperaciones*, ya que aun cuando aparecen en el campo del código de operación, no se traducen a instrucciones en código de máquina. Estas operaciones pueden definir símbolos, asignar programas y datos a ciertas áreas de la memoria, generar tablas y datos fijos, indicar la terminación del programa, etcétera. Las directivas de ensamblador más comunes son:

Definir contador del programa	
ORG	Define la dirección en memoria de inicio de la parte del programa que sigue. En un programa puede haber varios puntos de origen.
Definir símbolos	
EQU, SET, DEF	Iguala/ajusta/define un símbolo como un valor numérico, otro símbolo o una expresión.
Reservar localidades de memoria	
RMB, RES	Reserva bytes/espacio de la memoria.
Definir constante en la memoria	
FCB	Forma un byte constante.
FCC	Forma una secuencia de caracteres constante.
FDB	Forma una constante de dos byte.
BSW	Almacena bloque de ceros.

La información incluida en el campo del operando depende del mnemónico que le precede y del modo de direccionamiento. Proporciona la dirección de los datos que se manejan durante el proceso especificado en el código de operación. Por ello, con frecuencia se le conoce como *campo de direcciones*. Este campo puede estar vacío si las instrucciones dadas por el código de operación no necesitan datos ni dirección. Los datos numéricos de este campo pueden ser hexadecimales, decimales, octales o binarios. El ensamblador supone que los números son decimales, a menos que se indique lo contrario. En el 6800 de Motorola se escribe $ antes del número hexadecimal, o H al final; @ antes de los números octales, o una O o Q al final; % antes de un número binario, o B al final. En el Intel 8080 un número hexadecimal termina con H, un número octal termina con O o Q y un

número binario con B. Los números hexadecimales deben empezar con un dígito decimal, es decir, 0 a 9, para evitar confusión con los nombres. En el 6800 de Motorola, el modo de dirección inmediato se indica precediendo el operando con #, y en el modo de dirección indexado va seguido del operando X. Para los modos de direccionamiento directo o extendido no se utilizan símbolos especiales. Si la dirección está en la página cero, es decir, FF o menos, el ensamblador asigna en forma automática el modo directo. Si la dirección es mayor que FF, el ensamblador asigna el modo extendido.

El campo de comentarios es opcional y su propósito es permitir al programador incluir comentarios que contribuyan a una mayor legibilidad del programa. Durante la compilación del programa de código de máquina el ensamblador ignora este campo.

Figura 16.2 Diagrama de flujo para la suma de dos números

16.3.1. Ejemplos de programas en lenguaje ensamblador

Los siguientes ejemplos ilustran cómo elaborar algunos programas simples.

Problema: sumar dos números de 8 bits localizados en diferentes direcciones de la memoria y almacenar el resultado otra vez en la memoria.

El algoritmo es:

1. Iniciar.
2. Cargar el primer número en el acumulador. El acumulador es donde se acumulan los resultados de las operaciones aritméticas. Es el registro de trabajo, o sea, es una zona donde se hacen los cálculos antes de que el resultado se tranfiera a algún otro lado. Entonces se debe copiar el dato al acumulador para poder hacer la aritmética. Con los PIC se usa el término registro de trabajo (w).
3. Sumar el segundo número.
4. Guardar la suma en la localidad de memoria designada.
5. Parar.

La figura 16.2 muestra los pasos anteriores en un diagrama de flujo.

Los programas escritos para tres diferentes microcontroladores aparecen a continuación. En ellos la primera columna es la etiqueta, la segunda el código de operación, la tercera el operando y la cuarta los comentarios. Observe que todos los comentarios van precedidos por punto y coma.

Programa M68HC11

```
                ;Suma de dos números

NUM1    EQU     $00     ;posición del número 1
NUM2    EQU     $01     ;posición del número 2
SUM     EQU     $02     ;posición para la suma
```

```
        ORG     $C00        ;dirección inicial del usuario RAM
START   LDAA    $NUM1       ;carga número 1 al acumulador A
        ADDA    $NUM2       ;suma el número 2 a A
        STAA    SUM         ;guarda la suma en $02
        END
```

La primera línea del programa especifica la dirección del primer sumando. La segunda línea especifica la dirección del número que se suma al primer número. La tercera especifica dónde se colocará el resultado de la suma. La cuarta, la dirección de memoria en la que debe empezar el programa. El uso de etiquetas significa que el operando relacionado con los datos no tiene que especificar las direcciones, sólo las etiquetas.

Cuando las seudoperaciones se traducen a código de máquina también se indican las direcciones para los elementos. En código de máquina el programa sería el siguiente:

```
0010 96 70
0012 9B 71
0014 97 72
0016 3F
```

El mismo programa para un Intel 8051 sería:

Programa para 8051

```
        ;Suma de dos números
NUM1    EQU     20H         ;posición del número 1
NUM2    EQU     21H         ;posición del número 2
SUM     EQU     22H         ;posición para la suma

        ORG     8000H       ;dirección inicial del usuario RAM
START   MOV     A,NUM1      ;carga número 1 al acumulador A
        ADD     A,NUM2      ;suma el número 2 a A
        MOV     SUM,A       ;guarda la suma en 22H
        END
```

El mismo programa para un microcontrolador PIC sería:

Programa PIC

```
        ;Suma de dos números
Num1    EQU     H'20'       ;posición del número 1
Num2    EQU     H'21'       ;posición del número 2
Sum     EQU     H'22'       ;posición para la suma

        org     H'000'      ;dirección inicial del usuario RAM
Start   movlw   Num1        ;carga número 1 al acumulador A
        addlw   Num2        ;suma el número 2 a A
        movwf   Sum         ;guarda la suma en 22H
        End
```

Figura 16.3 Un ciclo

Figura 16.4 Diagrama de flujo de la suma de 10 números

En muchos programas existe la necesidad de realizar una tarea repetidas veces. En estos casos, el programa se diseña de manera que la operación pase por la misma sección cierto número de veces. Esto se denomina *procesamiento en ciclos* o *iteraciones*; un ciclo es una sección de un programa que se repite varias veces. La figura 16.3 muestra el diagrama de flujo de un ciclo.Con él cierta operación debe realizarse varias veces antes de proceder con el programa. Cuando la cantidad de operaciones está completa continua la ejecución del programa. El siguiente programa ilustra los ciclos.

Problema: sumar los números ubicados en 10 direcciones distintas (éstas pueden ser, por ejemplo, el resultado generado por 10 sensores para una muestra).

El algoritmo sería:

1. Inicio.
2. Definir el valor del conteo igual a 10.
3. Apuntar a la localidad que se encuentra en el número de dirección de la parte inferior.
4. Sumar el número que aparece en la dirección de la parte inferior.
5. Disminuir en uno el conteo.
6. Sumar 1 al apuntador de ubicación de la dirección.
7. ¿La cuenta es igual a 0? Si no es así, ramificar a 4. Si es así, continuar.
8. Guardar la suma.
9. Parar.

La figura 16.4 ilustra el diagrama de flujo. El programa es:

```
COUNT   EQU     $0010
POINT   EQU     $0020
RESULT  EQU     $0050
        ORG     $0001
        LDA B   COUNT     ;Cargar el contador
        LDX     POINT     ;Inicializar el registro de índi
                          ;ces al inicio de los números
SUM     ADD A   X         ;Sumar el sumando
        INX               ;Sumar 1 al registro de índice
        DEC B             ;Restar 1 al acumulador B
        BNE     SUM       ;Ramificar a suma
        STA A   RESULT    ;Guardar
        WAI               ;Parar el programa
```

El número 10, correspondiente al conteo, se carga en el acumulador B. El registro de índices proporciona la dirección inicial de los datos que se suman. El primer paso es sumar el contenido de la localidad de memoria direccionada por el registro de índices al contenido del acumulador, al inicio considerado como cero (se puede usar la instrucción CLR A para borrarlo al inicio). La instrucción INX suma

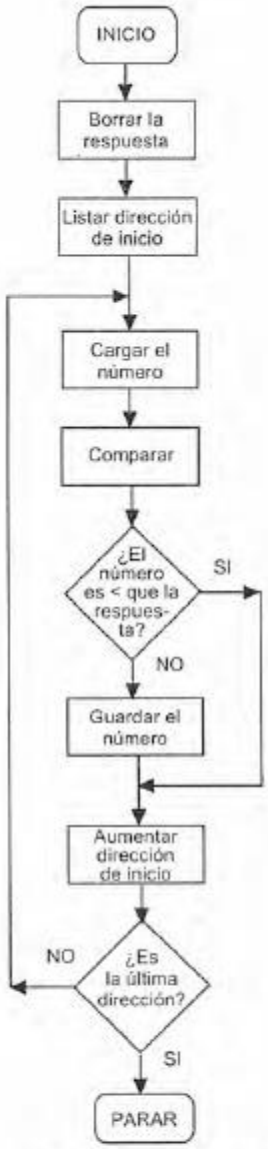


Figura 16.5 Diagrama de flujo para obtener el número mayor

1 al registro de índices, de manera que la siguiente dirección que se elije es 0021. DEC B resta 1 al contenido del acumulador B e indica que el valor del conteo es ahora 9. BNE es la instrucción para ramificar a SUM si no es igual a cero, es decir, si la bandera Z tiene valor 0. El programa itera y repite el ciclo hasta que ACC B es cero.

Problema: determinar cuál de todos los números de una lista es el mayor (podría ser determinar la mayor temperatura de, digamos, la temperatura más alta enviada por varios sensores de temperatura).

El algoritmo sería:

1. Borrar la dirección de la respuesta.
2. Listar la dirección de inicio.
3. Cargar el número de la dirección de inicio.
4. Comparar el número con el número en la dirección de respuesta.
5. Guardar la respuesta si es mayor.
6. De no ser así, guardar el número.
7. Aumentar la dirección de inicio en 1.
8. Ramificar a 3 si la dirección no es la última.
9. Parar.

La figura 16.5 muestra el diagrama de flujo. El programa es:

```
FIRST   EQU     $0030
LAST    EQU     $0040
ANSW    EQU     $0041
        ORG     $0000
        CLR     ANSW        ;Borrar la respuesta
        LDX     FIRST       ;Cargar la primera dirección
NUM     LDA A   $30,X       ;Cargar el número
        CMP A   ANSW        ;Comparar con la respuesta
        BLS     NEXT        ;Ramificar a NEXT si el valor
                            ;es menor o igual
        STA A   ANSW        ;Guardar la respuesta
NEXT    INX                 ;Aumentar registro de índices
        CPX     LAST        ;Comparar registro de índices
                            ;con LAST
        BNE     NUM         ;Ramificar si no es igual a cero
        WAI                 ;Parar el programa
```

El procedimiento borra primero la dirección de la respuesta. A continuación se carga la primera dirección y el número en dicha dirección se coloca en el acumulador A. LDA A $30,X significa cargar el acumulador A con los datos de la dirección dada por el registro de índices más 30. Se compara el número con la respuesta; el número se guarda si la respuesta es mayor que el número que ya está en el acumulador, de otra manera, se bifurca para repetir el ciclo con el siguiente número.

16.4 Subrutinas

Es frecuente el caso de que un bloque de programación, una subrutina, se requiera varias veces en el mismo programa. Por ejemplo, puede necesitarse para producir un retraso en el tiempo. Una opción es duplicar el programa de subrutina varias veces en el programa principal; esto, sin embargo, significa un aprovechamiento ineficiente de la memoria. Otra opción es conservar una copia en la memoria y bifurcar o saltar a la subrutina cada vez que sea necesario. No obstante, esto presenta el problema de saber, una vez concluida la subrutina, a qué parte del programa regresar para reanudar. Lo que se necesita es un mecanismo para regresar al programa principal y continuar en el punto en que se quedó cuando se inició la subrutina. Para ello es necesario guardar el contenido del contador del programa en el momento en que se bifurca a la subrutina para volver a cargar este valor en el contador del programa cuando termine la subrutina. Las dos instrucciones que se proporcionan con los microprocesadores que permiten implantar la subrutina de esta manera son:

1. JSR (salto a la rutina) o CALL (invocar), que permite invocar una subrutina.

2. RTS (regreso de la subrutina) o RET (regresar), que se usa como la última instrucción de una subrutina y regresa al sitio correcto del programa que lo invocó.

Las subrutinas se pueden llamar desde diversos puntos de un programa. Para ello es necesario guardar el contenido del contador del programa de forma que lo último en entrar sea lo primero en salir (LIFO, *last in first out*). Este tipo de registro se conoce como *pila*. Es como una pila de platos en la que el último plato siempre se coloca arriba y el primer plato que se saca es siempre el que está arriba, o sea, el último que se agregó a la pila. La pila puede ser un bloque de registros en un microprocesador o, más comúnmente, una sección de la memoria RAM. Un registro especial en el microprocesador, llamado *registro del apuntador de pila*, se usa para apuntar a la siguiente dirección libre en el área de la memoria RAM que se está usando para la pila.

Además del uso automático de la pila cuando se utilizan subrutinas, el programador puede diseñar un programa en el que la pila se utilice para guardar datos en forma temporal. En este caso, las dos instrucciones son:

1. PUSH (cargar). Mediante esta instrucción los datos de los registros especificados se guardan en la siguiente localidad de la pila que esté libre.

2. PULL (sacar) o POP (subir). Mediante esta instrucción se recogen los datos de la última ubicación en la pila y se transfieren a un registro especificado.

Por ejemplo, antes de ejecutar una subrutina, quizás sea necesario guardar los datos de algunos registros; y después de la subrutina, restaurar esos datos. Los elementos del programa serían, en el 6800 de Motorola:

```
SAVE      PSH A          ;Guardar acumulador A en pila
          PSH B          ;Guardar acumulador B en pila
          TPA            ;Transferir el registro de estado al
                         ;acumulador A
          PSH A          ;Guardar el registro de estado en
                         ;la pila
; Subrutina
RESTORE   PUL A          ;Restaurar el código de condición
                         ;desde la pila al acumulador A
          TAP            ;Restaurar el código de condición
                         ;desde A al registro de estado
          PUL B          ;Restaurar acumulador B desde
                         la ;pila
          PUL A          ;Restaurar acumulador A desde
                         la ;pila
```

16.4.1 Subrutina de retardo

Los *ciclos de retardo*, con frecuencia se requieren cuando el microprocesador tiene una entrada de un dispositivo, como un convertidor analógico a digital. Muchas veces se necesita enviar una señal al convertidor para que inicie la conversión y luego esperar un tiempo fijo antes de leer los datos del convertidor. Esto se puede hacer incluyendo un ciclo mediante el cual el microprocesador realiza diversas instrucciones antes de seguir con el resto del programa. Un programa de retardo sencillo sería el siguiente:

```
DELAY     LDA A    #$05     ;Cargar 05 en el acumulador A
LOOP      DEC A             ;Disminuir en 1 el acumulador A
          BNE      LOOP     ;Bifurcar si el resultado no es igual
                            ;a cero
          RTS               ;Regresar de la subrutina
```

Cada movimiento a través del ciclo implica varios ciclos de máquina. Cuando se recorre un ciclo cinco veces, el programa de retardo necesita:

Instrucción	*Ciclos*	Ciclos en total
LDA A	2	2
DEC A	2	10
BNE	4	20
RTS	1	1

En total, el retraso es de 33 ciclos de máquina. Si cada uno tarda 1 µs, entonces el retraso total es 33 µs. Para un retraso más largo, desde el inicio se pone un número mayor en el acumulador A.

Un ejemplo de la subrutina de ciclo de retardo para un microcontrolador PIC es:

```
        movlw   Valor     ; cargar el valor de cuenta requerido
        movwf   Cuenta    ; contador de ciclos
Delay   decfsz  Cuenta    ; decrementa el contador
        goto    Retardo   ; ciclo
```

La instrucción decfsz toma un ciclo y la instrucción goto toma dos ciclos. El ciclo se repetirá (cuenta – 1) veces. Adicionalmente se tienen las instrucciones movlw y movwf, cada una de ellas toma un ciclo, y cuando la cuenta es igual a 1 se tiene la instrucción decfsz que toma otros dos ciclos. Entonces, el número total de ciclos es:

$$\text{número de ciclos de instrucciones} = 3(\text{cuenta} - 1) + 4$$

Cada ciclo de instrucciones toma cuatro ciclos de reloj por lo que el número de ciclos de retardo introducidos por esta subrutina es:

$$\text{número de ciclos de reloj} = 4[2(\text{cuenta} - 1) + 4]$$

Con un reloj de 4 MHz cada ciclo de reloj toma $1/(4 \times 10^6)$ s.

Con frecuencia el retardo obtenido usando sólo el ciclo sencillo descrito no es suficiente. Una forma de obtener un retardo mayor es utilizar un ciclo anidado. La figura 16.6 muestra el diagrama de flujo de un ciclo de retardo anidado. El ciclo interior es igual al del programa de ciclo sencillo descrito antes. El registro E disminuirá 255 veces antes de que el ciclo termine y se establezca la bandera de cero. El ciclo exterior hace que la rutina del ciclo interior se ejecute repetidamente mientras el registro D disminuye hasta cero. Entonces con el registro D inicialmente con un conteo de ciclos de, digamos 140, el tiempo de retardo será $140 \times 2.298 = 321.72$ ms. El programa es entonces:

```
DELAY   MOV     D,8CH     ; fija D en 8CH, o sea, 140
OLOOP   MOV     E,FFH     ; fija E en FFH, o sea, 255
ILOOP   DEC     E         ; disminuye E, el contador del
                          ; ciclo interno
        JNZ     ILOOP     ; repite el ILOOP 255 veces
        DEC     D         ; disminuye D, el contador del
                          ; ciclo externo
        JNZ     OLOOP     ; repite el ILOOP 140 veces
```

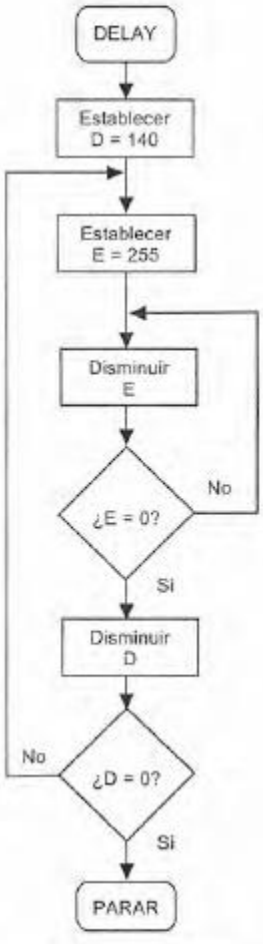


Figura 16.5 Ciclo de retardo anidado

Los siguientes son algunos ejemplos de programas donde las subrutinas de retardo son necesarias.

1. *Problema*: encender y apagar un LED repetidas veces

   Con este problema se usará la subrutina DELAY con ciclos para proporcionar los retardos requeridos; el microprocesador toma un tiempo finito para procesar las instrucciones en un ciclo y completar el ciclo. La estructura del programa es:

1. Si LED encendido
   Apagar LED
   Mientras LED apagado, ejecutar la subrutina RETRASO
2. De otra manera (ELSE)
   Encender LED
   Ejecutar la subrutina RETRASO

Subrutina RETRASO
   Realizar una instrucción, o instrucciones, o un ciclo, o un doble ciclo dependiendo del retardo requerido.

Por el tamaño del retardo necesario, es más conveniente utilizar un doble ciclo. Programando un Intel 8051, es posible utilizar la instrucción DJNZ, disminuye y salta si el resultado no es cero. Disminuye la dirección indicada por el primer operando y salta al segundo operando si el valor resultante no es cero. El LED está conectado a la terminal 0 del puerto 1 del microcontrolador. El programa utilizando las instrucciones en lenguaje ensamblador para el Intel 8051 sería:

```
FLAG     EQU     0FH             ; fijar bandera cuando LED encendido
         ORG     8000H

START    JB      FLAG,LED_OFF    ; salta si LED_OFF, o sea LED encendido
         SETB    FLAG            ; de otra manera fija el bit FLAG
         CLR     P1.0            ; enciende LED
         LCALL   DELAY           ; llama la subrutina DELAY
         SJMP    START           ; salta a START
LED_OFF  CLR     FLAG            ; borra la bandera de LED encendido para indicar LED apagado
         SETB    P1.0            ; apaga LED
         LCALL   DELAY           ; llama la subrutina DELAY
         LJMP    START           ; salta a START

DELAY    MOV     R0,#0FFH        ; valor del ciclo de retardo exterior
ILOOP    MOV     R1,#0FFH        ; valor del ciclo de retardo interior
OLOOP    DJNZ    R1,ILOOP        ; espera mientras ciclo interior
         DJNZ    R0,OLOOP        ; espera mientras ciclo exterior
         RET                     ; regresa de la subrutina

         END
```

2. *Problema*: encender en secuencia ocho LEDS

La instrucción rotar se puede usar para encender en forma sucesiva los LEDS, si tenemos inicialmente un arreglo de bit 0000 0001 el cual se rota para dar 0000 0011, luego 0000 0111 y así sucesivamente. El siguiente es un programa en lenguaje ensamblador para un Motorola 68HC11 que se puede usar, los LEDS están conectados al puerto B; un pequeño retardo se ha incorporado en el programa.

```
COUNT   EQU     8               ; el contador tiene el número de ciclos requeridos
                                ; o sea, el número de bits que van a encenderse
FIRST   EQU     %00000001       ; enciende el bit 0
PORTB   EQU     $1004           ; dirección del puerto B
        ORG     $C000
        LDAA    #FIRST          ; carga el valor inicial
        LDAB    #COUNT          ; carga contador
LOOP    STAA    PORTB           ; enciende bit 1 o sea LED 1
        JSR     DELAY           ; salta a la subrutina DELAY
        SEC                     ; fija el bit de acarreo para rotar en el bit menos significativo para
                                ; mantener el bit como 1
        ROLA                    ; rota hacia la izquierda
        DECB                    ; decrementa el contador

DELAY   RTS                     ; retrdo simple corto

        END
```

16.5 Tablas de consulta

El direccionamiento indexado se puede utilizar para que un programa busque valores en una tabla. Por ejemplo, al determinar el cuadrado de números enteros, un método posible es encontrar el valor correspondiente a un entero en particular en una tabla de cuadrados, en lugar de realizar la operación aritmética para encontrar el cuadrado. Las tablas de consulta son útiles en particular cuando la relación es no lineal y no se describe por ecuaciones aritméticas sencillas, por ejemplo, el sistema de mando de un motor descrito en la sección 1.5.2 donde el tiempo de encendido es una función del ángulo del eje del cigueñal y de la presión en la entrada del multiple. Aquí el microcontrolador tiene que enviar las señales de tiempo que dependen de señales de entrada del sensor de velocidad y de los sensores del eje del cigueñal.

Para ilustrar cómo se pueden usar las tablas de consulta, considere el problema de determinar los cuadrados de enteros. Se puede colocar una tabla de cuadrados de los enteros 0, 1, 2, 3, 4, 5, 6, ...en la memoria del programa y tener los cuadrados 0, 1, 4, 9, 16, 25, 36, ... en direcciones sucesivas. Si el número que se eleva al cuadrado es 4, entonces éste se convierte en el índice para la dirección indexada de los datos en la tabla, donde la primera entrada es el índice 0. El programa suma el índice a la dirección base de la tabla para encontrar la dirección de la entrada correspondiente al entero. De esta forma se tiene:

Índice	0	1	2	3	4	5	6
Entrada en tabla	0	1	4	9	16	25	36

Por ejemplo, con un microcontrolador Motorola 68HC11 se tiene el siguiente programa de búsqueda para determinar los cuadrados.

```
REGBAS   EQU    $B600      ; dirección base para la tabla
         ORG    $E000
         LSAB   $20        ; carga el acc. B con el entero
                           ; que se eleva al cuadrado
         LDX    #REGBAS    ; apunta a la tabla
         ABX               ; suma el contenido del acum. B
                           ; al registro del índice X
         LDAA   $00,X      ; carga el acum. A con el
                           ; valor indexado
```

se pudo haber cargado la tabla en la memoria usando la seudoperación FDB:

```
         ORG    $B600
         FDB    $00,$01,$04,$09   ; dando los valores a los
                                  bloques reservados
                                  de memoria
```

Con el microprocesador de Intel 8051 la instrucción MOVC A, @A+DPTR trae los datos de la localidad de memoria apuntada por la suma de DTPR y el acumulador A y la almacena en el mismo acumulador. Esta instrucción se puede usar para buscar datos en una tabla donde el apuntador de datos DPTR se inicializa al principio de la tabla. Como una ilustración, suponga que se quiere usar una tabla para la conversión de temperaturas en escala Celcius a escala Fahrenheit. El programa pasa los parámetros de las temperaturas que requieren conversión a una subrutina, de forma que puede incluir las siguientes instrucciones:

```
         MOV    A,#NUM             ; carga el
                                   ; valor que
                                   ; va a
                                   ; convertirse
         CALL   LOOK_UP            ; llama a la
                                   ; subrutina
                                   ; LOOK_UP

LOOK_UP  MOV    DPTR,#TEMP         ; apunta a la
                                   ; tabla
         MOVC   A,@A+DTPR          ; obtiene el
                                   ; valor de la
                                   ; tabla
         RET                       ; regresa
                                   ; de la
                                   ; subrutina
TEMP     DB     32, 34, 36, 37, 39, 41, 43, 45  ; dando
                                   ; valores a
                                   ; la tabla
```

Otro ejemplo del uso de una tabla es dar la secuencia de un número de salidas. Esta puede ser la secuencia para operar las luces de un semáforo que controla el tráfico, que dé la secuencia rojo, rojo más ambar, verde, ambar. La luz roja se ilumina cuando hay una salida de

RD0, la ambar se ilumina con RD1 y la verde, con RD2. Los datos de la tabla serían:

	Rojo	Rojo + ambar	Verde	Ambar
Índice	0	1	2	3
	0000 0001	0000 0011	0000 0100	0000 0010

16.5.1 Retardo para un motor paso a paso

En un motor paso a paso se deben utilizar retardos entre cada instrucción para avanzar un paso y permitir que haya tiempo para que ese paso ocurra antes de la siguiente instrucción del programa. El algoritmo de un programa para generar una secuencia continua de impulsos escalón sería el siguiente:

1. Inicio.
2. Definir la secuencia de las salidas necesarias para obtener la secuencia de pasos.
3. Establecer la posición del paso inicial.
4. Avanzar un paso.
5. Saltar a la rutina de retraso para dar tiempo a que se complete el paso.
6. ¿Este es el último paso en la secuencia de pasos para una rotación completa? Si no es así, continúe con el paso siguiente; si es así, regrese al número 3.
7. Continúe hasta infinito.

El siguiente es un programa posible para un motor paso a paso, en la configuración de paso completo y controlado por el microcontrolador M68HC11, usando las salidas de PB0, PB1, PB2 y PB3. Se utiliza una tabla 'de consulta' de la secuencia del código de salida para que las salidas lleven el motor paso a paso a la siguiente secuencia de pasos. La tabla que se utiliza es la siguiente:

Paso	Salidas requeridas desde el puerto B				Código
	PB0	PB1	PB2	PB3	
1	1	0	1	0	A
2	1	0	0	1	9
3	0	1	0	1	5
4	0	1	1	0	6
1	1	0	1	0	A

La secuencia de código que se necesita para operar el motor paso a paso con paso completo es A, 9, 5, 6, A; así, estos valores constituyen la secuencia que el apuntador debe consultar en la tabla. FCB es el código de operación para 'formar un byte constante' y se usa para inicializar los bytes de datos de la tabla.

```
BASE    EQU     $1000
PORTB   EQU     $4          ; Puerto de salida
TFLG1   EQU     $23         ; Registro 1 del indicador de
                            ; interrupción del temporizador
TCNT    EQU     $0E         ; Registro del contador del
                            ; temporizador
TOC2    EQU     $18         ; Registro de comparación 2
                            ; de salida
TEN_MS  EQU     20000       ; 10 ms en el reloj

        ORG     $0000
STTBL   FCB     $A          ; Ésta es la tabla de consulta
        FCB     $9
        FCB     $5
        FCB     $6
ENDTBL  FCB     $A          ; Fin de la tabla de consulta
        ORG     $C000
        LDX     #BASE
        LDA A   #$80
        STA A   TFLG1,X     ; Borrar bandera
START   LDY     #STTBL
BEG     LDA A   0,Y         ; Empezar por la primera
                            ; posición de la tabla
        STA A   PORTB,X
        JSR     DELAY       ; Saltar a demora
        INY                 ; Incremento en la tabla
        CPY     #ENTBL      ; ¿Es el fin de la tabla?
        BNE     BEG         ; Si no es así, bifurcar a BEG
        BRA     START       ; Si es así, ir de nuevo a inicio
DELAY   LDD     TCNT,X
        ADD D   #TEN_MS     ; Aumentar una demora de 10 ms
        STD     TOC2,X
HERE    BRCLR   TFLG1,X,    ; Esperar hasta que haya
                            ; transcurrido la demora
                $80,HERE
        LDA A   #$80
        STA A   TFLG1,X     ; Borrar bandera
        RTS
```

Observe que en la etiqueta TEN_MS hay un espacio subrayado para indicar que tanto TEN como MS son parte de la misma etiqueta.

El retardo aquí se obtiene mediante el bloque temporizador del microcontrolador. Se utiliza un retardo de 10 ms. En un sistema de microcontrolador con un temporizador de 2 MHz un retardo de 10 ms corresponde a 20 000 ciclos de reloj. Para obtener este retardo primero se obtiene el valor del registro del TCNT y se le agregan 20 000 ciclos; con este valor se carga el registro TOC2.

Problemas

1. Con base en el siguiente resumen del juego de instrucciones de un fabricante (6800), determine los códigos de máquina necesarios para la operación de suma con acarreo de los siguientes modos: a) de dirección inmediata; b) de dirección directa.

		Modos de direccionamiento					
		IMMED			DIRECT		
Operación	Mnemónico	OP	~	#	OP	~	#
Sumar con acarreo	ADC A	89	2	2	99	3	2

2. La operación de borrado del conjunto de instrucciones del procesador 6800 de Motorola sólo tiene una entrada en la columna de modo de direccionamiento implicado. ¿Qué significa esto?
3. ¿Cuál es la mnemotecnia del 6800 de Motorola para a) borrar un registro A; b) guardar el acumulador A; c) cargar el acumulador A; d) comparar los acumuladores; e) cargar el registro índice?
4. Escriba una línea de programa ensamblador para a) cargar el acumulador con 20 (hex); b) disminuir el acumulador A; c) borrar la dirección $0020; d) ADD (sumar) al contenido del acumulador A el número en la dirección $0020.
5. Explique las operaciones especificadas en las siguientes instrucciones: a) STA B $35; b) LDA A #$F2; c) CLC; d) INC A; e) CMP A #$C5; f) CLR $2000; g) JMP 05,X.
6. Escriba los siguientes programas en lenguaje ensamblador:
   a) Restar el número hexadecimal en la dirección de memoria 0050 del número hexadecimal en la ubicación de memoria 0060 y guardar el resultado en la dirección 0070.
   b) Multiplicar dos números de 8 bits ubicados en las direcciones 0020 y 0021 y guardar el producto, un número de 8 bits, en la dirección 0022.
   c) Guardar los números hexadecimales del 0 al 10 en las ubicaciones de la memoria que empiezan en 0020.
   d) Desplazar el bloque de 32 números a la dirección $3000, empezando por la dirección $2000.
7. Escribir en lenguaje ensamblador una subrutina que se pueda usar para producir un retardo que pueda ser igual a cualquier valor.
8. Escribir en lenguaje ensamblador una rutina que se pueda usar de manera que si la entrada en la dirección 2000 producida por un sensor tiene un valor alto, el programa salta a una rutina que empieza en la dirección 3000; si es baja, el programa continúa.

17 Lenguaje C

17.1 ¿Por qué el lenguaje C?

C es un lenguaje de alto nivel que a menudo se utiliza en vez del lenguaje ensamblador (vea el capítulo 16) para programar microprocesadores. Cuando se compara con el lenguaje ensamblador, tiene la ventaja de ser más fácil de manejar y que un mismo programa se puede usar con microprocesadores diferentes; para ello, basta usar el compilador apropiado para traducir el programa C al código de máquina del microprocesador involucrado. El lenguaje ensamblador varía dependiendo del tipo de microprocesador mientras que C es un lenguaje estandarizado por el ANSI I(*American National Standars Institute*).

Este capítulo es una introducción al lenguaje C y a la elaboración de programas; un estudio más detallado del tema, se encuentra en obras como *Teach Yourself C Programming in 21 Days* de P. Aitken y B.L. Jones (Sams Publishing, 1995), o bien *C for Engineers and Scientists* de G. Bronson (West, 1993) y, en forma específica para microcontroladores, *Programming Microcontrollers in C* de T. Van Sickle (High Text, 1994), *C Programming for Embedded Systems* de K. Zurell (R & D Books, 2000), *C and the 8050* volúmenes 1 y 2 de T. Schultz (Prentice Hall PTR, 1998), o *Embedded C* de M. J. Pont (Addison Wesley, 2002).

17.2 Estructura de un programa

La figura 17.1 da un panorama de los principales elementos de un programa en C. Existe un comando de preprocesado que invoca un archivo estándar, seguido de la función principal. Dentro de ésta hay otras funciones que se conocen como subrutinas. Cada función contiene cierta cantidad de instrucciones.

17.2.1 Características principales

Las siguientes son las características clave de los programas escritos en lenguaje C. Observe que en los programas en C el compilador ignora tanto los espacios como los cambios de línea y sólo se usan

para comodidad del programador, ya que facilitan la lectura del programa.

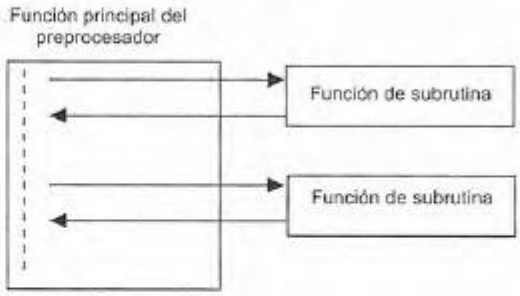


Figura 17.1 Estructura de un programa de lenguaje C

1. *Palabras clave*
   En el lenguaje C ciertas palabras se reservan como palabras clave (keywords) con significado específico. Por ejemplo, *int* se utiliza para indicar que se está trabajando con valores enteros; *if* se utiliza cuando un programa puede cambiar de dirección de ejecución, dependiendo de si una decisión es verdadera o falsa. C requiere que todas las palabras claves se escriban con minúsculas. Estas palabras no se pueden usar para otra cosa. Las siguientes son las palabras clave estándar (de ANSI) en C:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

2. *Instrucciones*
   Las instrucciones son los elementos que componen un programa; cada instrucción termina con un punto y coma. Las instrucciones pueden agruparse en bloques poniéndolas entre corchetes, es decir, {}. Por ejemplo, un grupo de dos instrucciones sería el siguiente:

```
{
instrucción 1;
instrucción 2;
}
```

3. *Funciones*
   El término *función* se utiliza para designar un bloque autónomo de código de programa que realiza un conjunto de acciones y tiene un nombre para referirse a ella (semejante a las subrutinas de los programas en lenguaje ensamblador). Una función se escribe como un nombre seguido de paréntesis; esto es, nombre( ). Los

paréntesis pueden encerrar argumentos; el argumento de una función es un valor que se transfiere a la función cuando se invoca. Para ejecutar una función, se invoca por su nombre como una instrucción del programa. Por ejemplo, tal vez se tenga la instrucción

```
printf("Mechatronics");
```

Esto significa que la palabra Mechatronics se transfiere a la función printf( ), una función preescrita que se invoca por el comando del preprocesador y, como resultado, la palabra se despliega en la pantalla. Para indicar que los caracteres forman una cadena como la palabra Mechatronics, se ponen entre comillas.

4. *Retorno*
   Una función puede regresar un valor a la rutina de invocación. Al frente del nombre de la función aparece el *tipo de retorno*, el cual especifica el tipo del valor que debe regresarse a la función que invoca una vez concluida la ejecución. Por ejemplo, int main( ) se utiliza para indicar que la función main regresa un valor entero. Algunas veces la función no devuelve ningún valor, en estos casos el retorno se especifica como void (vacío); por ejemplo, void main(void). Con frecuencia, el archivo de encabezados contiene esta información del retorno y no tendrá que especificarse cuando hay funciones definidas en el archivo de encabezados.

   Para regresar un valor desde una función hasta el punto donde se invocó, se utiliza la palabra clave *return*; por ejemplo, para regresar el contenido de result, se escribe:

```
return result;
```

   En general la instrucción return finaliza una función.

5. *Funciones de bibliotecas estándar*
   Los paquetes de lenguaje C cuentan con bibliotecas que contienen gran cantidad de funciones predefinidas en código C ya escritas y que ahorran al usuario el tiempo y esfuerzo de escribirlas. Estas funciones se pueden invocar por su nombre. Para utilizar el contenido de una biblioteca dada, se debe especificar en la cabecera del programa. Ejemplos de estas bibliotecas son:

```
math.h para funciones matemáticas
stdio.h para funciones de entrada y salida
time.h para funciones de tiempo y fecha
```

Por ejemplo, la función printf( ) es una función que se puede llamar desde la biblioteca stdio.h y sirve para enviar los resultados a la pantalla del monitor. Otra función es scanf( ) que puede usarse para leer datos del teclado.

6. *Preprocesado*
El *preprocesado* es un programa que se identifica por *comandos de preprocesado*, y que se ejecuta antes de la compilación. Estos comandos se distinguen por el signo # en el principio de la línea. Por ejemplo:

```
# include < >
```

para incluir el archivo que se especifica entre los paréntesis angulares . Cuando se llega a este comando, el archivo especificado se inserta en el programa. Es frecuente emplear este comando para agregar el contenido de los programas de encabezado estándar, los cuales cuentan con diversas declaraciones y definiciones para permitir el uso de las funciones de las bibliotecas estándar. La línea sería

```
# include <stdio.h>
```

Como ejemplo, considere el programa sencillo

```
# include <stdio.h>

main( )
{
  printf("Mechatronics");
}
```

Antes de iniciar el programa principal se agrega el archivo stdio.h. Así, cuando el programa principal empieza ya es posible emplear la función printf( ), que produce la palabra Mechatronics desplegada en la pantalla.

Otro tipo de comando de preprocesado es

```
# define pi 3.14
```

que sirve para definir valores que se insertan siempre que se encuentre cierto símbolo en el programa. Por ejemplo, siempre que se encuentre pi, se utilizará el valor 3.14.

```
# define square(x) (x)*(x)
```

sustituirá el término square(x) en el programa por (x)*(x).

7. *Función main (principal)*
Todo programa escrito en C tiene una función denominada *main*( ). Esta función controla la ejecución del programa y es la primera función invocada. La ejecución empieza con su primera instrucción. Otras funciones pueden ser invocadas en las instrucciones, cada una se ejecuta y el control regresa a la función principal. La instrucción

```
void main(void)
```

indica que ningún resultado regresará al programa principal y que no hay argumento. Por convención, cuando main( ) regresa un valor de 0 indica la terminación normal del programa; es decir

```
return 0;
```

8. *Comentarios*
Para incluir comentarios se usan /* y */. Por ejemplo

```
/* Sigue el programa principal */
```

El compilador ignora los comentarios y sólo se usan para facilitar al programador la comprensión de un programa. Los comentarios pueden ocupar más de una línea, por ejemplo

```
/* Un ejemplo de un programa usado para
ilustrar la programación */
```

9. *Variables*
Una *variable* es una localidad de memoria a la cual se ha asignado un nombre que puede guardar varios valores. Las variables en las que se guardan caracteres se especifican mediante la palabra clave *char*; dicha variable tiene una longitud de 8 bits y en general se usa para guardar un solo carácter. Los enteros con signo, es decir, números sin parte fraccionaria y con signo positivo o negativo, se especifican con la palabra clave *int*. La palabra clave *float* se usa para números de punto flotante, números con parte fraccionaria. La palabra clave *double* también se utiliza para números de punto flotante, pero proporciona el doble de dígitos significativos que *float*. Para declarar una variable antes del nombre se inserta el tipo, por ejemplo

```
int contador;
```

Esta expresión declara que la variable "contador" es de tipo entero. Otro ejemplo sería

```
float x, y;
```

Esto indica que las variables x y y son números de punto flotante.

10. *Asignaciones*
Una instrucción de asignación es aquella donde la variable que aparece a la izquierda del signo = toma el valor de la expresión que aparece a la derecha. Por ejemplo, a = 2 asigna el valor 2 a la variable a.

11. *Operadores aritméticos*
Los operadores aritméticos que se usan son: suma +, resta –, multiplicación *, división /, módulo %, incremento + + y decremento – –. El operador de incremento aumenta el valor de una variable en 1; el operador de decremento lo disminuye en 1. La precedencia de las operaciones es la misma de las operaciones aritméticas. Por ejemplo, 2*4 + 6/2 es 11. El siguiente es un ejemplo de un programa que utiliza operadores aritméticos:

```
/* programa para calcular el área de un círculo */

#include <stdio.h>  /*identifica la biblioteca*/

int radio, area  /*variables radio y área son enteros*/

int main(void)  */inicia programa main, int indica
     que un valor entero regresa,  void indica que
     main( ) no tiene parámetros*/
{
  printf("Ingresa radio:"); */"Ingresa radio" en la pantalla*/
  scanf("%d", &radio); */Lee un entero del
     teclado y lo asigna a la variable radio*/
  area = 3.14 * radio * radio; */Calcula el área*/
  printf("\nArea = %d", area); */En una nueva línea
     imprime  Area = y el valor numérico del área*/
  return 0;  */regresa al punto de llamado*/
}
```

12. *Operadores de relación*
Los operadores de relación se usan para comparar expresiones mediante preguntas como "¿Es *x* igual a *y*?" o "¿Es *x* mayor que 10?". Los operadores de relación son: es igual que = =, no es igual que !=, es menor que <, es menor o igual que <=, es mayor que >, es mayor o igual que >=. Observe que = = se utiliza cuando se pregunta si dos variables son iguales, y = se usa para las asignaciones, es decir, cuando se afirma que ambas variables son la misma. Por ejemplo, la representación de la pregunta "¿Es a igual que 2?" sería (a = = 2).

13. *Operadores lógicos*
Los operadores lógicos son:

Operador	Simbolo
AND	&&
OR	||
NOT	!

Observe que en C el resultado es igual a 1 si es verdadero y 0 si es falso.

14. *Operadores sobre bits (bitwise)*
Los operadores sobre bits manejan sus operandos como una serie de bits individuales, en lugar de un valor numérico; se comparan los bits de cada operando y sólo trabaja con variables enteras. Los operadores son:

Operación sobre bits	Símbolo
AND	&
OR	|
OR-EXCLUSIVA	^
NOT	~
Corrimiento a la derecha	>>
Corrimiento a la izquierda	<<

La siguiente instrucción es un ejemplo:

```
portA = portA | 0x0c;
```

El prefijo 0x indica que el 0c es un valor hexadecimal, donde 0000 1100 está en binario. El valor del puerto A al cual se aplica la operación OR es un número binario que fuerza los bits 2 y 3; todos los demás bits permanecen sin cambio.

```
portA = portA ^ 1;
```

La instrucción causa que todos los bits, excepto el bit 1 del puerto A, quedan sin cambio. Si el bit 0 en el puerto A, es 1, XOR lo cambiará a 0, y si es 0 lo cambiará a 1.

15. *Secuencia*
La serie de caracteres comprendida dentro de comillas, " ", se conoce como cadena. Como su nombre lo indica, estos caracteres se manejan como una entidad vinculada. Por ejemplo,

```
printf("Sum = %d", x)
```

El argumento que está dentro de ( ) especifica qué se transfiere a la función printf. Hay dos argumentos separados con una coma. El primero es la cadena entre comillas y especifica cómo se debe presentar la salida, el %d especifica que la variable se desplegará como entero decimal. Otros especificadores de formato son:

%c	carácter
%d	entero decimal con signo
%e	notación científica
%f	número en punto flotante
%o	octal sin signo

%s	cadena de caracteres
%u	entero decimal sin signo
%x	hexadecimal sin signo
%%	imprime el signo %

El argumento x especifica el valor que se desplegará.

Como otro ejemplo, la instrucción:

```
scanf("%d", &x);
```

lee del teclado un número entero decimal y lo asigna a la variable entera x. El símbolo & que antecede a x es el operador 'dirección de'. Cuando se pone antes del nombre de una variable, ésta devuelve la dirección de dicha variable. El comando permite leer datos y guardarlos usando la dirección dada.

16. *Secuencias de escape*

Las secuencias de escape son caracteres que 'escapan' de la interpretación estándar de los caracteres y se usan para controlar la ubicación de la salida en pantalla moviendo el cursor, o indicando un procedimiento especial. Por ejemplo,

```
printf("\nSum = %d", d)
```

el término \n indica que cada vez que aparezcan datos en la pantalla se debe usar una nueva línea. Las secuencias de escape utilizadas con más frecuencia son:

\a	emite una señal sonora (alarma)
\b	retroceso
\n	línea nueva
\t	tabulador horizontal
\\	diagonal invertida
\?	interrogación
\'	apóstrofo

17.2.2 Ejemplo de un programa en C

Un ejemplo de un programa sencillo para mostrar el uso de algunos de los términos anteriores es:

```
/* Programa sencillo en C*/
# include <stdio.h>
void main(void)
{
```

```
int a, b, c, d; /*a, b, c, y d son enteros*/
a = 4; /*a a se le asigna el valor 4*/
b = 3; /*a b se le asigna el valor 3*/
c = 5; /*a c se le asigna el valor 5*/
d = a * b * c; /*a d se le asigna el valor de a × b × c*/
printf( "a * b * c = %d\n", d);
}
```

La instrucción int a, b, c, d; declara las variables a, b, c y d como de tipo entero. Las instrucciones a = 4, b = 3, c = 5 asignan valores iniciales a las variables; el signo = indica asignación. La instrucción d = a * b * c indica que se debe multiplicar a por b, esto por c y guardar el resultado en d. La parte printf en la instrucción printf("a *b *c = %d\n", d) es la función para desplegar en el monitor. El argumento contiene %d, lo cual indica que se debe convertir a un valor decimal para desplegarlo. Es decir, imprime a * b * c = 60. El carácter \n al final de la cadena indica que en ese punto hay que insertar una nueva línea.

17.3 Control de flujo y ciclos

Las instrucciones que permiten el control de flujo y la realización de ciclos en los programas son *if* (si), *if/else* (si/de otra manera), *for* (para), *while* (mientras) y *switch* (conmutar).

1. *If*
   La instrucción *if* produce una ramificación (figura 17.2). Por ejemplo, si una expresión es verdadera, se ejecuta la instrucción; si no lo es, no se ejecuta y el programa continúa con la siguiente instrucción. La instrucción podría ser de la forma

```
if(condition 1 = = condition 2);
printf("\nCondition is OK.");
```

Figura 17.2 If

Un ejemplo de un programa en el que se utiliza la instrucción if es:

```
#include <stdio.h>

int x, y;
main( )
{
 printf("\nIngresa el valor entero para x:");
 scanf("%d", &x);
 printf("\nIngresa el valor entero para y:");
 scanf("%d", &y);
 if(x = = y)
    printf("x es igual que y");
 if(x > y)
    printf("x es mayor que y");
 if(x < y)
    printf("x es menor que y");
 return 0;
}
```

Figura 17.3 If/else

En la pantalla aparece Ingresa el valor entero para x: y entonces debe introducirse un valor en el teclado. La pantalla muestra Ingresa el valor entero para y: y debe introducirse un valor. La secuencia if determina si los valores introducidos son iguales, o cual es mayor que otro y despliega el resultado en la pantalla.

2. *If/else*

La instrucción *if* se combina con la instrucción *else*. Si el resultado es sí se ejecuta una instrucción; si es no, se ejecuta otra instrucción (figura 17.3). Por ejemplo:

```
#include<stdio.h>

main( )
{
 int temp;
 if(temp > 50)
    printf("Precaución");
 else
    printf("El sistema está bien");
}
```

3. *For*

El término *ciclo* (loop) se usa para la ejecución de una secuencia de instrucciones hasta que una condición determinada resulta verdadera o falsa. La figura 17.4 ilustra esto. Una manera de escribir instrucciones para un ciclo es usar la función *for*. La forma general de esta instrucción es

```
for(expresión inicial; expresión prueba; expresión
    incremento)
instrucción de ciclo;
```

Un ejemplo de cómo se usa es

```
#include<stdio.h>

int contador

main ( )
{
 for(contador = 0; contador < 7; contador ++)
 printf("\n%d", contador);
}
```

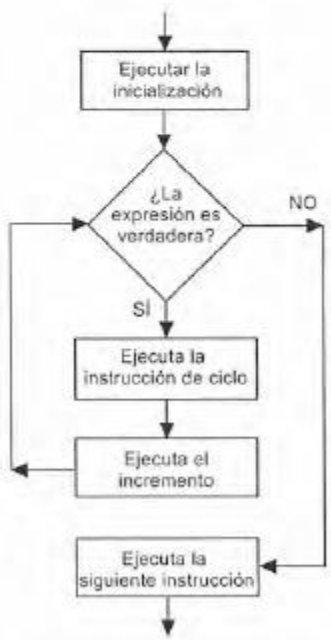


Figura 17.4 For

El valor inicial de contador es 0, se incrementa en 1, se hace un ciclo y se repite la instrucción for en tanto que contador sea menor que 7. El resultado en pantalla muestra 0 1 2 3 4 5 6, donde cada número está en una línea separada.

Figura 17.5 While

4. *While*

Con esta instrucción la repetición de un ciclo continúa mientras la expresión sea verdadera (figura 17.5). Cuando la expresión resulta falsa, el programa continúa con la siguiente instrucción después del ciclo. Un ejemplo es el siguiente programa, donde la instrucción while se ejecuta mientras que el valor de contador es menor que 7, y despliega los resultados.

```
#include <stdio.h>

int contador;
int main ( );
{
 contador = 1;
 while(contador < 7)
    {
     printf("\n%d", contador);
     contador++;
    }
    return 0;
}
```

En pantalla aparece 1 2 3 4 5 6 con cada número en una sola línea.

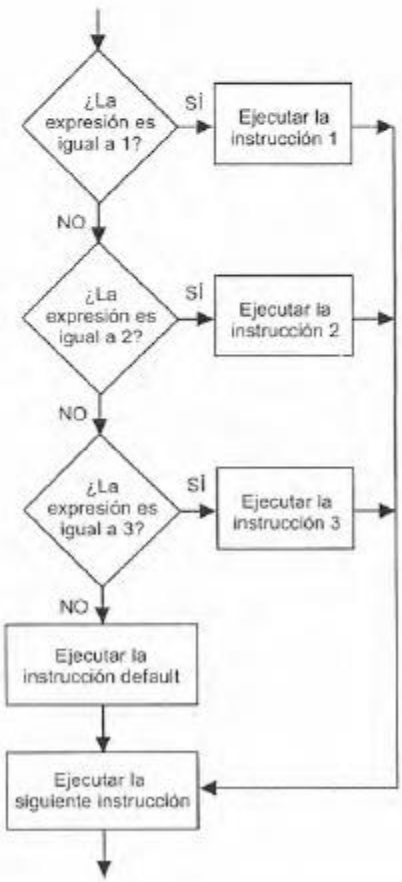


Figura 17.6 Switch

5. *Switch*

Con esta instrucción se elige entre varias alternativas, la condición a probar aparece entre paréntesis. Las posibles opciones se identifican por etiquetas *case*, las cuales identifican los valores esperados de la condición de prueba. Por ejemplo, si ocurre case 1 se ejecutaría la instrucción 1; si ocurre case 2, se ejecuta la instrucción 2, etcétera. Si la expresión no es igual a alguno de los case, entonces, se ejecuta la instrucción default. Después de una instrucción case casi siempre aparece una instrucción break para transferir la ejecución a la instrucción posterior al switch y detener el switch para que no recorra toda la lista de case. La secuencia es la siguiente (figura 17.6):

```
switch(expression)
{
 case 1;
    instrucción 1;
    break
 case 2;
    instrucción 2;
    break;
 case 3;
    instrucción 3;
    break;
 default;
    instrucción default;
```

```
}
next instrucción
```

El siguiente es un ejemplo de un programa que reconoce los números 1, 2 y 3 y despliega el que se introdujo con el teclado.

```
#include<stdio.h>

int main ( );
{
 int x;

 printf("Ingrese un número 0, 1, 2 o 3: ");
 scanf("%d", &x);

 switch (x)
 {
  case 1:
    printf("Uno");
    break;
  case 2:
    printf("Dos");
    break;
  case 3:
    printf("Tres");
    break;
  default;
    printf("No fue 1, 2, o 3");
 }
 return 0;
}
```

17.4 Arreglos

Suponga que se desea registrar la temperatura del mediodía, durante una semana, y después, localizar la temperatura correspondiente a un día en particular. Esto puede realizarse usando un arreglo. Un *arreglo* es una colección de localidades de memoria para almacenar datos, donde cada una tiene el mismo tipo de dato y el mismo nombre de referencia. Para declarar un arreglo con el nombre Temperatura para guardar valores de tipo flotante se especifica la instrucción:

```
float Temperatura[7];
```

El tamaño del arreglo se indica entre paréntesis cuadrados [ ], justo después del nombre del arreglo. En este caso se usó 7 para los datos de cada día de la semana. Para referirse a los elementos individuales del arreglo se utiliza un valor de un índice. Al primer elemento corresponde el número 0, al segundo el 1 y así sucesivamente, de manera que el último elemento de una secuencia de n elementos es el $n - 1$. La figura 17.7 muestra la forma de un arreglo secuencial. Para almacenar valores en el arreglo, se puede escribir:

arreglo[0]	arreglo[1]	arreglo[2]	arreglo[3]

Figura 17.7 Arreglo secuencial de cuatro elementos

```
temperatura[0] = 22.1;
```

```
temperatura[1] = 20.4;
etcétera
```

Si se desea utilizar scanf( ) para introducir un valor en uno de los elementos del arreglo, ponga & delante del nombre del arreglo, por ejemplo,

```
scanf("%d", &temperatura [3]);
```

El siguiente es un ejemplo de un sencillo programa para guardar y desplegar el cuadrado de los números 0, 1, 2, 3, y 4:

```
#include<stdio.h>

int main(void)
{
 int sqrs[5];
 int x;

 for(x = 1; x<5; x++)
    sqrs[x - 1] = x * x;
 for(x = 0; x < 4; x++)
    printf("%d", sqrs[x]);

 return 0;
}
```

Los arreglos pueden tener valores iniciales cuando se les declara por vez primera, por ejemplo,

```
int array[7] = {10, 12, 15, 11, 10, 14, 12};
```

Si se omite el tamaño del arreglo, el compilador creará un arreglo lo suficientemente grande para incluir los valores de inicialización.

```
int array[ ] = {10, 12, 15, 11, 10, 14, 12};
```

arreglo[0][0]	arreglo[0][1]
arreglo[1][0]	arreglo[1][1]
arreglo[2][0]	arreglo[2][1]

Figura 17.8 Arreglo bidimensional

Existe la posibilidad de emplear *arreglos multidimensionales*. Por ejemplo, una tabla de datos es un arreglo de dos dimensiones (figura 17.8), donde x representa la fila en tanto que y es la columna, y se escribe como:

```
array[x][y];
```

17.5 Apuntadores

La dirección de una localidad de memoria es única y proporciona los medios para acceder a los datos guardados en una localidad. Un *apuntador* es una variable especial que puede guardar la dirección de otra variable. Si una variable denominada p contiene la dirección de otra variable denominada x, se dice que p *apunta* a x. Si x se encuentra en la dirección 100 de la memoria, p tendría el valor 100. Como el apuntador es una variable, igual que otras variables, debe ser declarada antes de utilizarse. El siguiente es un ejemplo de cómo se declara un apuntador:

```
type *nombre;
```

El * indica que el nombre se refiere a un apuntador. Es frecuente que los nombres para designar apuntadores se escriban con el prefijo p, es decir, pname. Por ejemplo,

```
int *pnumero;
```

Para inicializar un apuntador y darle una dirección a la cual apuntar se utiliza &, que es el operador de dirección, utilizando una instrucción de la forma:

```
pointer = &variable;
```

El siguiente programa corto ilustra lo anterior:

```
#include<stdio.h>

int main(void)

{
 int *p, x;
 x = 12;
 p = &x;  /*asigna a p la dirección de x*/
 printf("%d", *p); /*muestra el valor de x usando p*/

 return 0;
}
```

El programa despliega el número 12 en la pantalla. El acceso al contenido de una variable usando un apuntador, como en el caso anterior, se conoce como *acceso indirecto*. El proceso de acceder a los datos de una variable direccionada mediante un apuntador se conoce como *referenciación* del apuntador.

17.5.1 Aritmética de los apuntadores

Las variables de apuntador pueden tener los operadores aritméticos +, –, + + y – –. El incremento o decremento de un apuntador da como resultado que apunta al elemento siguiente o al anterior de un arreglo.
Entonces, para incrementar un apuntador al siguiente elemento de un arreglo se puede utilizar

```
pa++; /*usando el operador incrementa en 1*/
```

o bien:

```
pa = pa + 1;  /*sumando 1*/
```

17.5.2 Apuntadores y arreglos

Mediante los apuntadores es posible acceder a elementos individuales en un arreglo. El siguiente programa muestra cómo hacerlo.

```
#include <stdio.h>

int main(void)
{
 int x[5] = (0, 2, 4, 6, 8);
 int *p;
 p = x;  /*asigna a p la dirección de inicio de x*/
 printf("%d %d", x[0], x[2]);

 return 0;
}
```

La instrucción printf ("%d %d", x[0], x[2]); apunta la dirección dada por x, por lo tanto, se muestran los valores de las dos direcciones [0] y [2], es decir 0 y 4, cada uno en una línea.

17.6 Desarrollo de programas

Al desarrollar programas la meta es terminar con un conjunto de instrucciones en lenguaje máquina que se pueda usar para operar un sistema microprocesador/microcontrolador. Estas instrucciones forman el *archivo ejecutable*. Con el fin de llegar a este archivo ocurre la siguiente secuencia de eventos:

1. *Creación del código fuente*
   Consiste en escribir la secuencia de instrucciones en lenguaje C que constituirán el programa. Muchos compiladores tienen un editor para introducir el código fuente; de otra manera, se puede recurrir a Notepad de Microsoft Windows. El uso de un procesador de textos puede presentar problemas, ya que la información adicional de formato podría impedir la compilación, a menos que se opte por guardar el archivo sin la información de formato.

2. *Compilación del código fuente*
   Una vez escrito el código fuente, la compilación es su traducción en código de máquina. Antes de iniciar el proceso de compilación, se ejecutan los comandos del preprocesado. El compilador puede detectar varias formas de error durante la traducción y generar mensajes que indiquen los errores. Algunas veces un solo error produce una secuencia de errores en cascada, todos consecuencia del primer error. En general los errores obligan a regresar a la etapa de edición y reditar el código fuente. El compilador almacena el código de máquina en otro archivo.

3. *Vinculación (linking) para crear un archivo ejecutable*
   Entonces se usa el compilador para vincular, es decir, ligar el código generado con las funciones de biblioteca para obtener un

solo archivo ejecutable. El programa se almacena como un archivo ejecutable.

17.6.1 Archivos de encabezado

Los comandos de preprocesado se usan al principio del programa para definir las funciones utilizadas en ese programa; esto se hace para poder referirse a ellas con etiquetas. Sin embargo, para evitar escribir grandes listas de funciones estándar para cada programa, se puede usar una instrucción de preprocesamiento para indicar que se deberá usar un archivo que incluye las funciones estándar relevantes. Todo eso es necesario para indicar cuál archivo de funciones estándar deberá usar el compilador, este archivo es un *encabezado* puesto que aparece como cabecera del programa. Por ejemplo, <stdio.h> contiene funciones de entrada y salida estándar tales como get (obtener, entradas, es decir lee información de un dispositivo), put (poner, salidas, es decir, escribe información en un dispositivo) y scanf (leer datos); <math.h> contiene funciones matemáticas tales como cos, sen, tan, exp (exponencial) y sqrt (raíz cuadrada).

Los archivos de encabezado también están dispuestos para definir los registros y puertos de los microcontroladores y ahorran al programador tener que definir cada registro y cada puerto escribiendo líneas de preprocesamiento para cada uno. Entonces, para el microcontrolador 8051 de Intel se podría tener el encabezado <reg51.h>, éste define todos los registros, por ejemplo, los puertos P0, P1, P2 y P3, bits individuales en registros direccionables por bits como, TF1, TR1, TF0, TR0, IE1, IT1, IE0 y IT0 en el registro TCON. Así, se pueden escribir instrucciones refiriéndose a las entradas y salidas del puerto 0 usando etiquetas P0 o TF1 para el bit TF1 en el registro TCON. De manera similar, para un MC68HC11E9 de Motorota el encabezado <hc11e9.h> define los registros, por ejemplo, PORTA, PORTB, PORTC y PORTD, y los bits individuales de los registros direccionables por bits, por ejemplo, STAF, STAI, CWOM, HNDS, OIN, PLS, EGA e INVB en el registro PIOC. Así, se pueden escribir instrucciones refiriéndose a las entradas y salidas del puerto A usando simplemente la etiqueta PORTA. Las bibliotecas pueden también proveer rutinas para ayudar en el uso de dispositivos periféricos de hardware tales como teclados y pantallas de cristal líquido.

El programa principal escrito quizás para un microcontrolador específico puede, como resultado del cambio del archivo de encabezado, podrá adaptarse con facilidad para correr en cualquier microcontrolador. Las bibliotecas hacen posible que los programas en C sean transportables.

17.7 Ejemplos de programas

Los siguientes son ejemplos de programas escritos en C para sistemas basados en microcontroladores.

17.7.1 Encendido y apagado de un motor

Suponga que desea programar el microcontrolador M68HC11 para arrancar y detener un motor de cd. El puerto C se usa para las entradas y el B para la salida al motor, pasando por el respectivo amplificador de potencia o driver (figura 17.9). El botón de arranque está conectado a PC0; al accionarlo, la entrada cambia de 1 a 0 cuando arranca el motor. El botón de paro está conectado a PC1 para cambiar la entrada de 1 a 0 cuando se detenga el motor. El registro de direcciones de datos del puerto C, DDRC, se define como 0 y el puerto C queda definido para recibir entradas. El programa correspondiente sería:

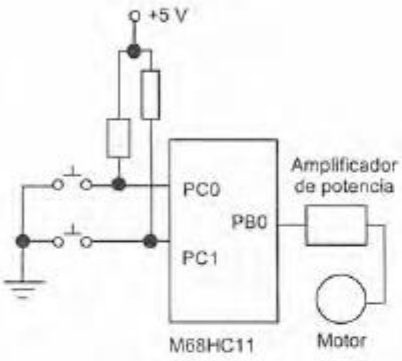


Figura 17.9 Control de un motor

```
#include<hc11e9.h> /*incluye el archivo de encabezado*/
void main(void)
{
 PORTB.PB0 &=0; /*al inicio el motor está apagado*/
DDRC = 0; /*prepara puerto C para entrada*/
while (1)  /*repite mientras se mantiene la condición*/
{
 if(PORTC.PC0 = =0) /*¿se oprimió el botón arranque?*/
     PORTB.PB0 |= 1;  /*salida de arranque si se oprimió*/
else if(PORTC.PC1) = = 0) /*¿se oprimió el botón paro?*/
     PORTB.PB0 &=0;  /*salida de paro si se oprimió*/
 }
}
```

Observe que | es el operador OR y ajusta un bit del resultado a 0 sólo si los bits correspondientes de ambos operandos son 0; de no ser así, se define como 1. Se usa para activar o definir uno o varios bits iguales a un valor. Por ejemplo, en el Puerto B.PB0 |=1, al 1 se aplica el operador OR tomando el valor que está en PB0 y se enciende el motor. Ésta es una manera práctica de conmutar en forma simultánea varios bits de un puerto. El & de PORTB.PB0 & = 0 se usa para aplicar el operador AND al bit PB0 con 0, y puesto que PB0 ya es 1, asigna a PORTB.PB0 el valor de 0.

17.7.2 Lectura de un canal del ADC

Suponga que desea programar un microcontrolador (M68HC11) de manera que sólo lea uno de los canales del ADC. El M68HC11 contiene un ADC de aproximaciones sucesivas de 8 bits y ocho canales multiplexados, a través del puerto E (figura 17.10). En el registro de control/estado del ADC, ADCTL, se encuentra el indicador de fin de conversión CCF en el bit 7 y otros bits que sirven para controlar al multiplexor y la exploración de canales. Si CCF = 0, la conversión no ha finalizado; cuando es 1, ya finalizó. La conversión analógica a digital se inicia escribiendo un 1 en el bit DPU del registro OPTION. Sin embargo, es necesario que el ADC haya estado encendido por lo menos 100 μs antes de leer un valor.

Para convertir la entrada analógica a PE0, hay que definir igual a 0 los primeros cuatro bits del registro ADCTL, es decir, CA, CB, CC

y CD. Si sólo se convierte un canal, el bit SCAN 5 se define igual a 0 y el bit MULT 4 igual a 0. Un programa para leer un canal en particular debe contener lo siguiente: después de encender el ADC, todos los bits del registro ADCTL se cambian a 0, se pone el número del canal y se lee la entrada cuando CCF es 0.

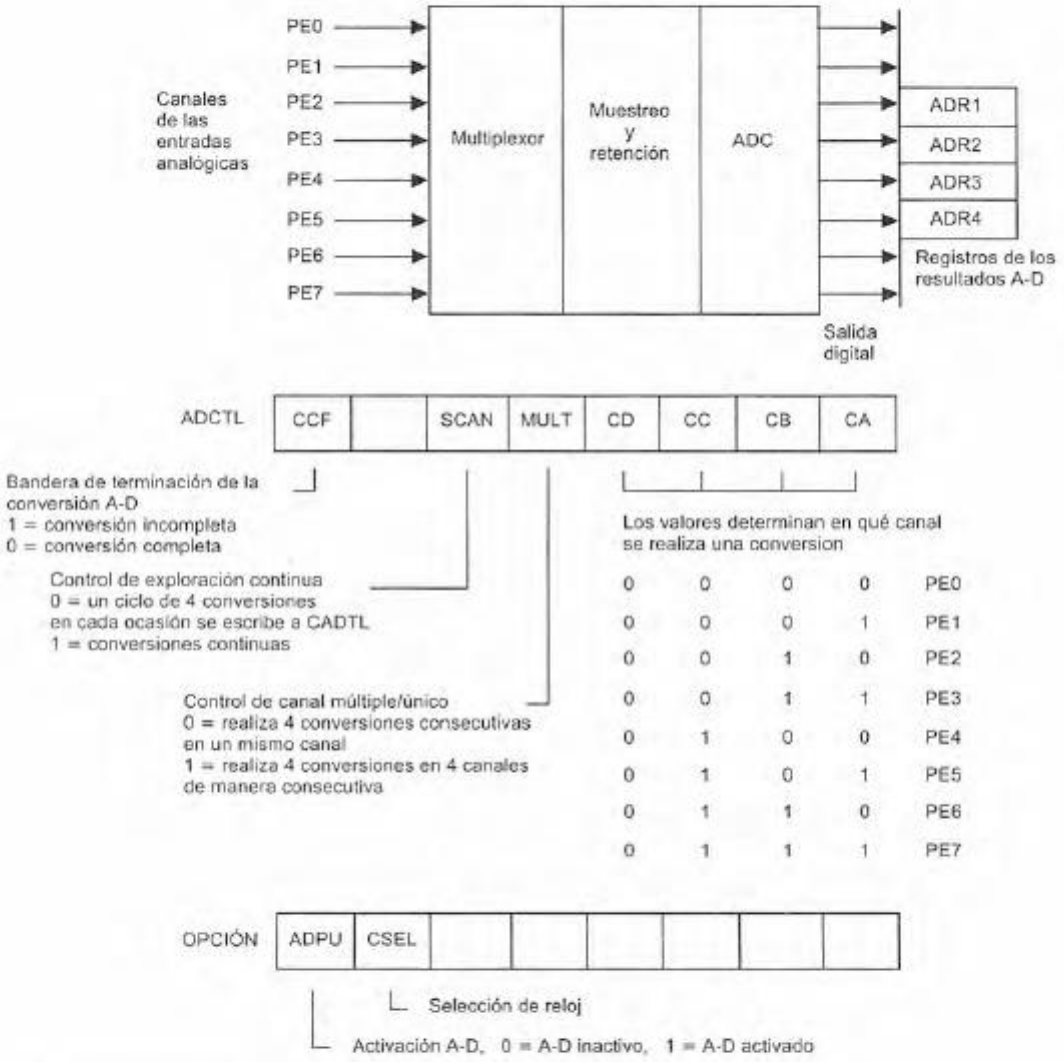


Figura 17.10 Convertidor ADC

El programa sería el siguiente:

```
#include<hc11e9.h>  /*incluye el archivo de encabezado*/
  void main(void)
  {
  unsigned int k; /*dá el número del canal*/
  OPTION=0; /*ésta línea y las siguientes encienden el ADC*/
```

```
OPTION.ADPU=1;

ADCTL &=~0x7; /*borra los bits*/
ADCTL |=k; /*da el número del canal a leer*/
while (ADCTL.CCF = = 0);
return ADR1; /*regresa el valor convertido a la dirección 1*/
}
```

Observe que ~ es el operador complemento y su tarea es invertir los bits de su operando, es decir, todos los 0 cambian a 1 y viceversa. Se define el bit 7. | es el operador OR y en el resultado define un bit como 0 sólo si los bits correspondientes de ambos operandos son 0; de no ser así, define el resultado como 1. Se utiliza para activar o definir, uno o varios bits en un valor. En este caso, con k = 1, sólo se define CA igual a 1. Para asegurar que después del encendido el valor no se lea demasiado rápido, se añade una subrutina de retraso.

Problemas

1. Las siguientes preguntas se refieren a los componentes de un programa.
   a) Señale qué indica el término int en la siguiente instrucción:

   ```
   int counter;
   ```

   b) Señale qué indica la siguiente instrucción:

   ```
   num = 10
   ```

   c) Señale cuál sería el resultado de la siguiente instrucción:

   ```
   printf("Name");
   ```

   d) Indique cuál sería el resultado de la siguiente instrucción:

   ```
   printf("Number %d", 12);
   ```

   e) Señale cuál sería el resultado de lo siguiente:

   ```
   #include <stdio.h>
   ```

2. Para el siguiente programa: ¿por qué se incluye la línea a) #include <stdio.h>; b) los corchetes {}; c) /d; y d) ¿qué aparece en la pantalla cuando se ejecuta el programa?

   ```
   #include <stdio.h>
   main( )
   {
     printf(/d"problema 3");
   }
   ```

3. ¿Qué se desplegará en la pantalla al ejecutar el siguiente programa?

```
#include <stdio.h>

int main(void);
{
  int num;
  num = 20;

  printf("El número es %d", num);

  return 0;
}
```

4. Escriba un programa para calcular el área de un rectángulo cuando se dan en la pantalla su longitud y ancho. La respuesta se despliega precedida de las palabras 'El área es'.

5. Escriba un programa que despliegue los números del 1 al 15, cada uno en una línea.

6. Explique las razones de las instrucciones del siguiente programa para dividir dos números.

```
#include <stdio.h>

int main(void);
{
  int num1, num2;

  printf("Teclee el primer número:");
  scanf("%d", &num1);

  printf("Teclee el segundo número:");
  scanf("%d", &num2);

  if(num2 = = 0)
    printf("No se puede dividir entre cero")
  else
    printf("El resultado es: %d", num1/num2);

  return 0;
}
```

18 Sistemas de entrada/salida

18.1 Interfases

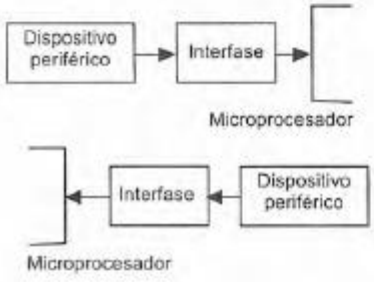


Figura 18.1 Interfaces

Cuando un microprocesador controla un sistema, debe recibir información de entrada, responder a ésta y producir señales de salida para realizar la acción de control requerida. Entonces puede haber señales de entrada desde sensores y señales de salida a dispositivos externos tales como relevadores y motores. El término *dispositivo periférico,* o *periférico* designa un dispositivo, que puede ser un sensor, un teclado, un actuador, etcétera, el cual se conecta con un microprocesador. Por lo general, no es posible conectar en forma directa un dispositivo periférico a un microprocesador por la falta de compatibilidad en la forma y nivel de sus señales; para lograr la compatibilidad necesaria se recurre a un circuito, que se conoce como interfase, que permite el acoplamiento entre los dispositivos periféricos y el microprocesador. La figura 18.1 ilustra esta configuración. La interfase es la parte donde se elimina la incompatibilidad.

Este capítulo estudia los requerimientos de estas interfases y del adaptador de interfase para dispositivo periférico MC6820 de Motorola y del adaptador de interfase para comunicaciones asíncronas MC6850 también de Motorola.

18.2 Direccionamiento entrada/salida

Existen dos formas en que el microprocesador puede seleccionar a los dispositivos de entrada/salida. Algunos microprocesadores, por ejemplo el Zilog Z80, tiene *entradas/salidas aisladas* e instrucciones de entrada especiales, como IN que se utiliza para leer de un dispositivo de entrada, e instrucciones especiales de salida como OUT que se utiliza para escribir en los dispositivos de salida. Por ejemplo, con el Z80 se tendría:

```
IN A,(B2)
```

para leer el dispositivo de entrada B2 y poner el dato en el acumulador A. Una instrucción de salida sería:

```
OUT (C),A
```

para escribir el dato del acumulador A en el puerto C.

Es común que los microprocesadores no tengan instrucciones por separado para la entrada y la salida, sino que usen las mismas instrucciones para escritura en memoria y lectura de memoria. A esto se denomina *entrada/salida de memoria mapeada*. Con este método, cada dispositivo de entrada/salida tiene una dirección, justo como una localidad de memoria. Los microcontroladores 68HC11 de Motorola, 8051 de Intel y los PIC no tienen las instrucciones de entrada/salida por separado y utilizan el mapeo de memoria. De esta forma, con el mapeo de memoria se usaría:

```
LDAA $1003
```

para leer el dato de entrada en la dirección $1003 y:

```
STAA $1004
```

para escribir el dato de salida en la dirección $1004.

Los microprocesadores ingresan y extraen bits de datos a través de puertos paralelos. Muchos dispositivos periféricos requieren varios puertos de entrada/salida; debido a que la palabra de datos del dispositivo periférico es más larga que la de la CPU. La CPU debe transferir los datos por segmentos. Por ejemplo, si se necesita una salida de 16 bits con una CPU de 8 bits, el procedimiento es:

1. La CPU prepara los ocho bits más significativos de los datos.
2. La CPU envía al primer puerto los ocho bits más significativos de los datos.
3. La CPU prepara los ocho bits menos significativos de los datos.
4. La CPU envía al segundo puerto los ocho bits menos significativos de los datos.
5. Así, después de cierto retardo, los 16 bits llegan al dispositivo periférico.

18.2.1 Registros de entrada/salida

El microcontrolador 68HC11 de Motorola tiene cinco puertos A, B, C, D y E (sección 15.3.1). Los puertos A, C y D son bidireccionales y se pueden usar tanto para entrada como para salida. El puerto B es sólo de salida y el E es sólo de entrada. Usar un puerto bidireccional ya sea de entrada o de salida depende del estado de un bit en su registro de control. Por ejemplo, el puerto A en la dirección $1000 se controla mediante el acumulador de pulso del registro de control PACTL en la dirección $1026. Para hacer que el puerto A se use como entrada se requiere que el bit 7 sea 0; para que se use como salida se requiere que el bit 7 sea 1 (figura 15.13) El puerto C es bidireccional y los ocho bits en su registro en la dirección $1003 están controlados por los bits correspondientes en su registro de dirección de datos del puerto en la dirección $1007. Cuando el bit de dirección de datos correspondiente se hace 0, se tiene un puerto de entrada y cuando se hace 1, de salida. El puerto D es bidireccional y contiene sólo seis líneas de entrada/salida en la dirección $1008. Está controlado por el registro de dirección del puerto en la direc-

ción $1009. La dirección de cada línea se controla con el bit correspondiente en el registro de control, éste es 0 para una entrada y es 1 para una salida. Algunos de los puertos también se pueden configurar para realizar otras funciones fijando otros bits en el registro de control.

Para un puerto de dirección fija, por ejemplo, el puerto B del 68HC11 de Motorola es sólo un puerto de salida, las instrucciones para enviar al exterior algún valor, como $FF, son sencillamente aquellas que se necesitan para cargar el dato a esa dirección. La instrucción podría ser:

```
REGBAS EQU  $1000      ; dirección base para los registros de E/S
PORTB  EQU  $04        ; incremento de PORTB a partir de REGBAS
       LDX  #REGBAS    ; cargar el registro de índices X
       LDAA #$FF       ; cargar $FF en el acumulador
       STAA PORTB, X   ; almacenar el valor en la dirección PORTB
```

Para el puerto E de dirección fija, el cual es únicamente de entrada, las instrucciones para leer un byte de ahí podrían ser:

```
REGBAS EQU  $1000      ; dirección base para los registros de E/S
PORTE  EQU  $0A        ; incremento del PORTE a partir de REGBAS
       LDAA PORTE, X   ; cargar el valor en PORTE en el acumulador
```

Para un puerto bidireccional como el puerto C, antes de poder utilizarlo como de entrada se debe configurar de modo que actúe como entrada. Esto significa hacer todos los bits 0. Así, se tendría:

```
REGBAS EQU  $1000      ; dirección base para los registros de E/S
PORTC  EQU  $03        ; incremento de PORTC a partir de REGBAS
DDRC   EQU  $07        ; incremento de la dirección del registro
                       ; datos a partir de REGBAS
       CLR  DDRC,X     ; llenar DDRS con 0
```

Para el microcontrolador 8051 de Intel (vea la sección 15.3.2) existen cuatro puertos de entrada/salida bidireccionales. Cuando el bit de un puerto se va a utilizar como salida, el dato sólo se pone en el bit del registro de funciones especiales correspondiente; cuando se utiliza como entrada se pone un 1 en cada bit concerniente, de esta manera se puede escribir FFH para un puerto completo donde se va a escribir. Considere un ejemplo de las instrucciones del 8051 de Intel para encender un LED cuando se presiona un botón. El botón proporciona una entrada al P3.1 y una salida a P3.0; el botón hace que la entrada se vaya a un estado bajo cuando se presiona.

```
       SETB P3.1      ; hace que el bit P3.1 se vaya a 1 y la entrada
                      ; también
LOOP   MOV  C,P3.1    ; lee el estado del botón
                      ; y lo almacena en la bandera de acarreo
       CPL  C         ; complementa la bandera de acarreo
       MOV  P3.0, C   ; copia el estado de la bandera de acarreo a la
                      ; salida
       SJMP LOOP      ; mantiene la secuencia en repetición
```

Con los microcontroladores PIC la dirección de las señales en sus puertos bidireccionales se fijan mediante el registro de dirección TRIS (sección 15.3.3). El registro TRIS se hace 1 para lectura y 0 para escritura. Los registros para el PIC16C73/74 están acomodados en dos bancos y antes de poder seleccionar un registro en particular, se tiene que elegir el banco poniendo el bit 5 en el registro STATUS. Este registro está en ambos bancos por lo que no se tiene que seleccionar el banco para usar este registro. Los registros TRIS están en el banco 1 y los registros PORT están en el banco 0. De esta manera para fijar el puerto B como salida primero se debe seleccionar el banco 1 y luego hacer TRISB, 0. Luego se puede seleccionar el banco 0 y escribir la salida al PORTB. El banco se selecciona asignando un bit en el registro de STATUS. Las instrucciones para seleccionar el puerto B como salida son:

```
Salida   clrf   PORTB          ; limpia todos los bits en el puerto B
         bsf    STATUS,RP0     ; usa el registro de estado (status)
                               ; para seleccionar el banco 1
                               ; haciendo RP0 igual a 1
         clrf   TRISB          ; limpia los bits de la salida
         bsf    STATUS,RP0     ; usa el registro de estado (status)
                               ; para seleccionar el banco 0
                               ; el puerto B es ahora una salida
                               ; que se hizo 0
```

18.3 Requerimientos de una interfase

Las siguientes son algunas de las acciones que se requieren con frecuencia de un circuito de interfase:

1. *Acoplamiento mediante buffer/aislamiento eléctricos*
   Es necesario cuando un dispositivo periférico funciona con un voltaje o corriente distintos de los del sistema de buses del microprocesador, o cuando sus referencias de tierra son diferentes. El término *buffer* se refiere a un dispositivo que proporciona aislamiento y amplificación de corriente o voltaje. Por ejemplo, si la salida de un microprocesador se conecta a la base de un transistor, la corriente de base necesaria para conmutar el transistor es mayor que la que proporciona el microprocesador de manera que se utiliza un buffer para amplificar la corriente. Muchas veces también se requiere aislamiento entre el microprocesador y el sistema de alimentación eléctrico.

2. *Control de temporización*
   Este control es necesario cuando las velocidades de transferencia de los datos entre el dispositivo periférico y el microprocesador son distintas, por ejemplo, cuando un microprocesador se conecta a un dispositivo periférico más lento. Esto se puede realizar utilizando líneas especiales entre el microprocesador y el dispositivo periférico a fin de controlar la temporización de las transferencias de datos. Estas líneas se conocen como *líneas de*

reconocimiento (*handshaking*), y el proceso como *reconocimiento*. El dispositivo periférico envía la señal DATA READY (datos disponibles) a la sección de entrada/salida. La CPU determina si la señal DATA READY está activa; después lee los datos en la sección de entrada/salida y envía la señal INPUT ACKNOWLEDGED (reconocimiento de entrada) al dispositivo periférico. Esta señal indica que la transferencia se ha realizado y, por ello, el dispositivo periférico puede enviar más datos. Para una salida, el dispositivo periférico envía una señal OUTPUT REQUEST (petición de salida) o una PERIPHERAL READY (periféricos listos) a la sección de entrada/salida. La CPU determina si la señal PERIPHERAL READY está activa y envía los datos al dispositivo periférico. La siguiente señal PERIPHERAL READY sirve para informar a la CPU que la transferencia se ha realizado.

3. *Conversión de código*
   Esta conversión es necesaria cuando los códigos que usan los dispositivos periféricos difieren de los que usa el microprocesador. Por ejemplo, un LED requiere un decodificador para convertir la salida BCD del microprocesador en el código necesario para operar los displays de siete segmentos.

4. *Modificación de la cantidad de líneas*
   La longitud de palabra en los microprocesadores es fija: 4, 8 o 16 bits. Esto determina la cantidad de líneas en el bus de datos del microprocesador. La cantidad de líneas del equipo periférico puede ser diferente, y quizás requerir una palabra más larga que la del microprocesador.

5. *Transferencia de datos en serie a paralelo y viceversa*
   En un microprocesador de 8 bits en general los datos se manipulan 8 bits a la vez. Para transferir de manera simultánea 8 bits a un dispositivo periférico se necesitan 8 rutas de datos. Esta forma de transferencia se llama *transferencia de datos en paralelo*. Sin embargo, no siempre es posible transferir datos de esta forma. Por ejemplo, en la transferencia de datos de un sistema telefónico público puede haber sólo una ruta de datos, por lo que deben transferirse de manera secuencial, un bit a la vez. Este tipo de transferencia se denomina *transferencia de datos en serie* y es más lenta que la transferencia de datos en paralelo. Si se usa la transferencia de datos en serie, es necesario convertir los datos en serie que entran al microprocesador en datos en paralelo y viceversa, cuando salen de él.

6. *Conversión de analógico a digital y viceversa*
   La señal de salida de los sensores es casi siempre analógica, y para que el microprocesador la pueda recibir es necesario convertirla a digital. La señal de salida de un microprocesador es digital y esto puede requerir una conversión a señal en analógica para operar un actuador.

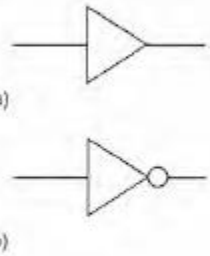


Figura 18.2 Buffers

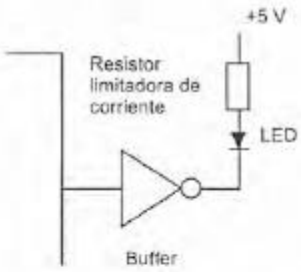


Figura 18.3 Cómo usar un buffer

18.3.1 Buffers

La figura 18.2a muestra el símbolo de un buffer que no modifica la lógica de la entrada, sólo el nivel de corriente o voltaje. Es similar al símbolo de un amplificador, puesto que ésa es la función del buffer. La figura 18.2b muestra el símbolo de un buffer que no sólo amplifica sino también invierte la salida.

Para ilustrar cómo se usa un buffer, considere una salida que activa un indicador de LED (figura 18.3). Cuando la salida del microprocesador es alta, el buffer produce una salida baja. El buffer ofrece a la corriente del LED una trayectoria hacia la tierra de baja resistencia, por lo que el LED se enciende. Cuando la salida del microprocesador es baja, el buffer produce una salida alta y la corriente del LED no tiene trayectoria hacia la tierra, por lo que se apaga.

Con frecuencia los dispositivos periféricos deben compartir las líneas de datos del microprocesador; es decir, deben conectarse al bus de datos y para ello es necesario que el microprocesador active sólo un dispositivo a la vez y los demás permanezcan desactivados. Para ello se utilizan los *buffers de tres estados*. Este término se refiere a que su salida puede ser baja, alta o desconexión. El estado de flotación tiene una impedancia alta, por lo que la salida se ve como un circuito abierto con lo que se conecte a él. La figura 18.4 muestra cómo usar este tipo de buffers y la figura 18.5 sus símbolos. Estos buffers están disponibles como circuitos integrados, por ejemplo, el 74125 con cuatro buffers no inversores tipo activo bajo y el 74126 con cuatro buffers no inversores tipo activo alto.

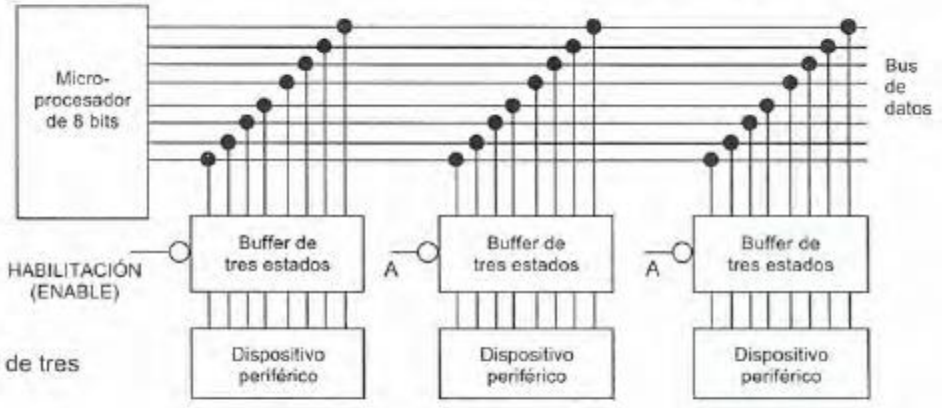


Figura 18.4 Buffer de tres estados

18.3.2 Reconocimiento (*handshaking*)

A menos que dos dispositivos puedan enviar y recibir datos a la misma velocidad, es necesario un reconocimiento para intercambiar datos. Con el reconocimiento el dispositivo más lento controla la velocidad de transferencia. Para la transferencia de datos en paralelo, la forma de reconocimiento más común es la de *muestreo y reconocimiento*. Durante éste, se indica cuando se está listo para recibir se envían los datos y mientras éstos se reciben, el receptor indica que no está listo para recibir más datos; concluida la transferencia, este dispositivo indica que está listo para recibir otra vez.

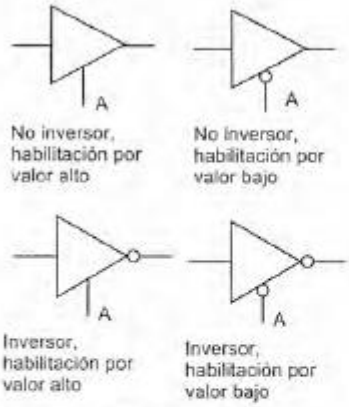


Figura 18.5 Buffers de tres estados

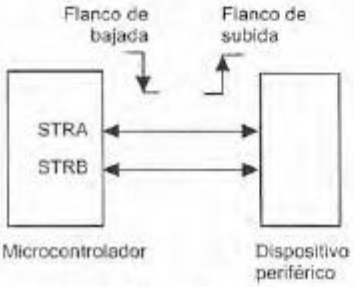


Figura 18.6 Control de reconocimiento

Con el microcontrolador MC68HC11, la operación de entrada/salida muestreada consiste en lo siguiente. Para las señales de control de reconocimiento se usan las terminales STRA y STRB (figura 18.6, vea también en la figura 15.9 el modelo de bloques completo); el puerto C se usa para la entrada muestreada y el puerto B para la salida muestreada. Cuando los datos están listos para que los envíe el microcontrolador, STRA produce un pulso y se envía al dispositivo periférico. Cuando el microcontrolador recibe un flanco de subida o de bajada en STRB, el puerto de salida relevante del microcontrolador envía los datos al dispositivo periférico. Una vez que los datos están listos para enviarlos al microcontrolador, el dispositivo periférico envía una señal a STRA indicando que está listo, y luego un flanco de subida o de bajada en STRB se usa para indicar que está listo para recibir. Antes de que ocurra el reconocimiento, el registro de entrada/salida PIOC en la dirección $1002 es el primero que se configura. La figura 18.7 ilustra los estados necesarios para los bits que están en ese registro.

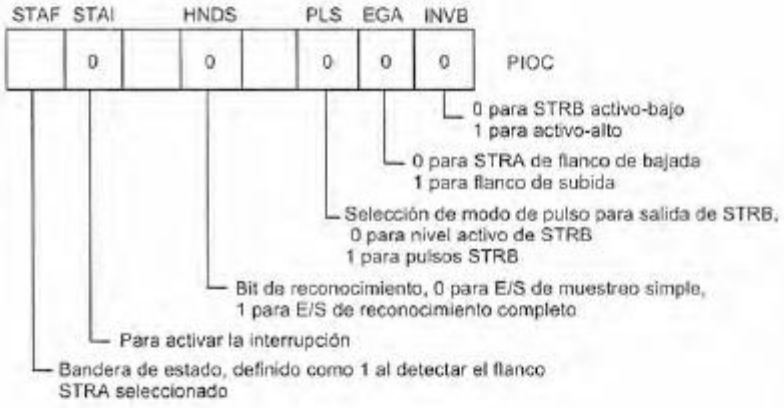


Figura 18.7 PIOC

Entrada/salida con reconocimiento completo, consiste en el envió de dos señales a través de STRB; la primera indica listo para recibir datos y la otra que los datos se leyeron. Esta forma de operación requiere que en el PIOC el bit HNDS sea 1; si PLS se hace 0, se dice que el reconocimiento completo es tipo pulsado y si es igual a 1, que está asegurado. Durante la operación por pulsos, se envía un pulso como reconocimiento; con un STRB asegurado se produce un reinicio (figura 18.8).

18.3.3 Poleo e interrupciones

Suponga una situación en la que todas las transferencias de entrada/salida de datos se controlan en un programa. Cuando los dispositivos periféricos necesitan atención, alertan al microprocesador modificando el nivel de voltaje de una línea de entrada. El microprocesador responde saltando a una rutina de servicio del programa para el dispositivo. Al finalizar la rutina, regresa al programa principal. El control del programa de las entradas/salidas es un ciclo para leer entradas y actualizar salidas, con saltos a rutinas de servicio

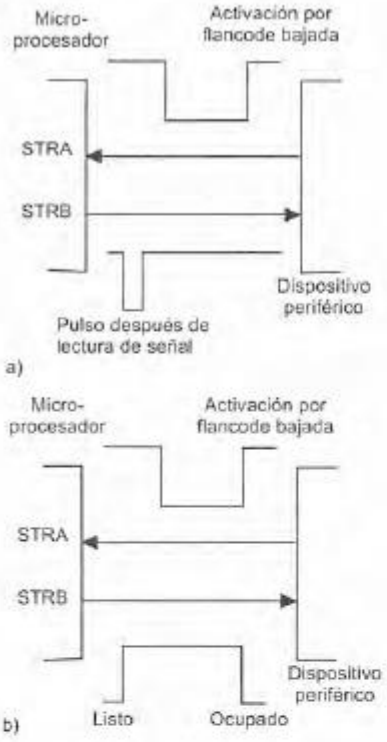


Figura 18.8 Reconocimiento completo: a) pulsado, b) asegurado

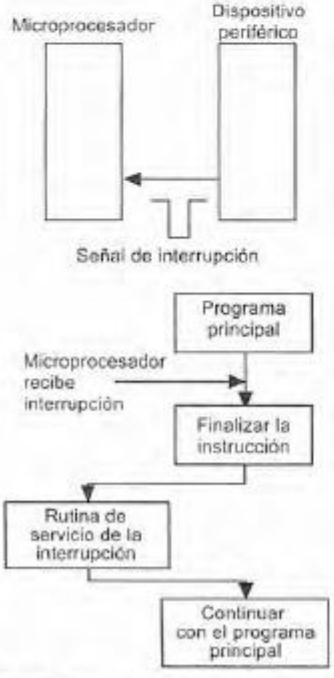


Figura 18.9 Solicitud de interrupción

cuando se requieren. Este proceso, que consiste en repetir la verificación de cada dispositivo periférico para determinar si está listo para enviar o aceptar un nuevo byte de datos se llama *poleo*.

Una opción del control por programa es el *control de interrupciones*. Una interrupción incluye un dispositivo periférico que activa una línea de petición de interrupción especial. Cuando se recibe una interrupción, el microprocesador suspende la ejecución de su programa principal y salta a la rutina de servicio del dispositivo periférico. La interrupción no debe producir una pérdida de datos y la rutina para manejar una interrupción debe estar incorporada al software, de manera que el estado de los registros del procesador y la última dirección del programa principal a la que se haya accedido queden guardadas en localidades de la memoria específicas. Al concluir la rutina de servicio de interrupción, se restaura el contenido de la memoria y el microprocesador reanuda la ejecución del programa principal, en el punto que fue interrumpido. De este modo, cuando ocurre una interrupción (figura 18.9):

1. El CPU espera a que termine la instrucción que está ejecutando antes de manejar la interrupción.
2. Todos los registros del CPU se sitúan en la pila y se modifica un bit para detener interrupciones adicionales durante esta interrupción. La pila es un área especial de memoria en la que los valores del contador del programa se pueden almacenar cuando se ejecuta una subrutina. El contador de programa proporciona la dirección de la siguiente instrucción en un programa y al almacenar este valor habilita el programa para que reanude en el punto donde se detuvo para ejecutar la interrupción.
3. El CPU determina la dirección de la rutina de servicio de la interrupción que se va a ejecutar. Algunos microprocesadores tienen terminales dedicadas a las interrupciones y la terminal que se elige determina la dirección que se va a usar. Otros microprocesadores tienen sólo una terminal para interrupciones y el dispositivo de interrupción debe proporcionar los datos que informan al microprocesador dónde se localiza la rutina de servicio de la interrupción. Algunos microprocesadores tienen ambos tipos de entradas de interrupciones. La dirección de inicio de una rutina de servicio de interrupción se llama *vector de interrupción*. El bloque de memoria asignado para almacenar estos vectores se conoce como *tabla de vectores de interrupciones*. El fabricante de los chips fija las direcciones de los vectores.
4. La CPU se ramifica hacia la rutina de servicio de la interrupción.
5. Después que termina esta rutina, los registros del CPU regresan desde la pila y el programa principal reanuda en el punto donde se quedó.

A diferencia de un llamado de subrutina, que está ubicada en un punto específico en un programa una interrupción se puede llamar desde cualquier punto del programa. Observe que el programa no controla cuándo ocurre una interrupción, el control está en el evento de interrupción.

Con frecuencia las operaciones de entrada/salida usan interrupciones debido a que el hardware no puede esperar. Por ejemplo, un teclado puede generar una señal de entrada de interrupción cuando se presiona una tecla. El microprocesador suspende el programa principal para manipular la entrada del teclado, procesa la información y regresa al programa principal para continuar donde éste se detuvo. Esta capacidad de codificar una tarea como una rutina de servicio de una interrupción y amarrarla a una señal externa simplifica muchas tareas de control, permitiendo manipularlas sin retardo. Es posible programar el microprocesador para que ignore la señal de solicitud de algunas interrupciones a menos que un bit haya sido habilitado. Tales interrupciones se denominan *enmascarables*.

El 68HC11 de Motorola tiene dos señales de entrada externas de interrupción. XIRQ es una interrupción no enmascarable y siempre se ejecuta al terminar la instrucción que se está ejecutando actualmente. Cuando la interrupción XIRQ se presenta, el CPU salta a la rutina de servicio de la interrupción cuyo vector de interrupción se mantiene en la dirección $FFF4/5 (bytes bajo y alto de la dirección). IRQ es una interrupción enmascarable. Cuando el microcontrolador recibe una señal de solicitud de interrupción en la terminal IRQ que va de bajada, el microcontrolador salta a la rutina de servicio de la interrupción indicada por el vector de interrupción $FFF2/3. IRQ se puede enmascarar con la instrucción fija la máscara de la interrupción SEI y se puede desenmascarar con la instrucción limpia la máscara de la interrupción CLI. Al final de la rutina de servicio de la interrupción se usa la instrucción RTI para regresar al programa principal.

Con el 8051 de Intel, las fuentes de interrupción se habilitan y deshabilitan en forma individual a través del registro de bit direccionable IE (habilitación de interrupción) en la dirección 0A8H (vea figura 15.28), un 0 deshabilita una interrupción y un 1 la habilita. Existe además un bit de habilitación/deshabilitación global en el registro IE que se fija en 1 para habilitar todas las interrupciones externas o se hace 0 para deshabilitarlas. El registro TCON (figura 15.27) se usa para determinar el tipo de señal entrada de interrupción que inicializará una interrupción.

Con los microcontroladores PIC, las interrupciones se controlan mediante el registro INTCON (figura 18.10). Para usar el bit 0 del puerto B como una interrupción, debe asignarse como una entrada y el registro INTCON se debe inicializar con un 1 en INTE y un 1 en GIE. Si la interrupción se va a presentar en un flanco de subida, entonces se debe hacer 1 el INTEDG (bit 6) en el registro OPTION (fi-

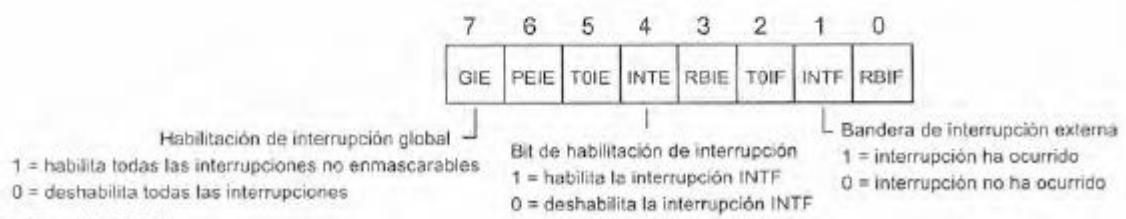


Figura 18.10 INTCON

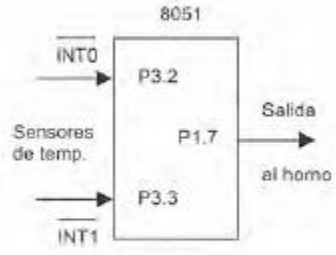


Figura 18.11 Sistema de calefacción central

gura 15.34); si es con flanco de bajada este bit debe hacerse 0. Cuando la interrupción se presenta INTF se modifica. Se puede limpiar mediante la instrucción bcf INTCON,INTF.

Como ilustración de un programa que involucre interrupciones externas, considere un programa de control encendido/apagado sencillo para un sistema de calefacción central que involucra el microcontrolador 8051 de Intel (figura 18.11). El horno del sistema de calefacción central se controla mediante una salida P1.7 y se usan dos sensores de temperatura, uno para determinar cuando la temperatura baja de, digamos, 20.5°C y el otro cuando sube más de 21.0°C. El sensor para la temperatura de 21.0°C se conecta a la interrupción INT0, puerto 3.2 y el sensor para la temperatura de 20.5°C se conecta a la INT1, puerto 3.3. Al elegir que el bit IT1 sea 1 en el registro TCON, las interrupciones externas se disparan por flanco, es decir, se activan cuando hay un cambio de 1 a 0. Cuando la temperatura sube a 21.0°C la interrupción externa INT0 tiene una entrada que cambia de 1 a 0 y la interrupción se activa para que la instrucción CLR P1.7 dé una salida 0 y apague el horno. Cuando la temperatura cae a 20.5°C la interrupción externa INT1 tiene una entrada que cambia de 0 a 1 y la interrupción se activa para que la instrucción SETB P1.7 dé un 1 a la salida y encienda el horno. El programa PRINCIPAL es sólo un conjunto de instrucciones para configurar y habilitar las interrupciones, establecer las condiciones iniciales de que el horno esté encendido si la temperatura es menor que 21.0°C o esté apagado si es mayor, y entonces espera haciendo nada hasta que la interrupción ocurra. Con el programa, se ha supuesto que hay un archivo de encabezado.

```
        ORG   0
        LJMP  MAIN

        ORG   0003H   ; proporciona la dirección de entrada para ISR0
ISR0    CLR   P1.7    ; rutina de servicio de la interrupción para
                      ; apagar el horno
        RETI          ; regreso de la interrupción

        ORG   0013H   ; proporciona la dirección de entrada para ISR1
ISR1    SETB  P1.7    ; rutina de servicio de la interrupción para
                      ; apagar el horno
        RETI          ; regreso de la interrupción

        ORG   30H
MAIN    SETB  EX0     ; para habilitar la interrupción externa 0
        SETB  EX1     ; para habilitar la interrupción externa 1
        SETB  IT0     ; hacer el disparo cuando hay un cambio
                      ; de 1 a 0
        SETB  IT1     ; hacer el disparo cuando hay un cambio
                      ; de 1 a 0
        SETB  P1.7    ; enciende el horno
        JB    P3.2,HE ; si la temperatura es mayor que 21.0°C
              RE      ; salta a HERE y deja encendido el horno;
        CLR   P1.7    ; apaga el horno
```

```
HERE  SJMP  HERE    ; hacer nada hasta que se presente
                    ; una interrupción
      END
```

Los microcontroladores, además de la solicitud de interrupción tienen la interrupción para reinicio y una interrupción no enmascarable. La *interrupción para reinicio* es un tipo especial de interrupción y cuando ocurre el sistema se reinicia, por lo que cuando está activa se detiene todo el sistema, se carga la dirección de inicio del programa principal y se ejecuta la rutina de inicio. El microcontrolador M68HC11 tiene un *sincronizador de controlador de secuencia para el adecuado funcionamiento de la computadora* (COP, *computer operating properly*). Su propósito es detectar errores en el procesamiento del software cuando la CPU no ejecuta ciertas secciones de código dentro del lapso asignado. Cuando esto ocurre, el sincronizador del COP rebasa su tiempo y se procede al reinicio del sistema.

La *interrupción no enmascarable* no se puede enmascarar, lo cual significa que no hay forma de impedir la ejecución de la rutina de la interrupción cuando se conecta en esta línea. Una interrupción de este tipo se reserva para casos de rutinas de emergencia, como cuando se interrumpe el suministro de energía eléctrica y, se recurre a la alimentación de una fuente de respaldo.

18.3.4 Acoplamiento mediante interfase en serie

En la transmisión de datos en paralelo, por cada bit se utiliza una línea; por otra parte, en los sistemas en serie se usa una sola línea para transmitir datos en bits enviados en secuencia. Existen dos tipos básicos de transferencia de datos: asíncrona y síncrona.

En la *transmisión asíncrona*, tanto el receptor como el transmisor usan su propia señal de sincronización, por lo que el receptor no conoce cuándo inicia o termina una palabra. Por ello es necesario que cada palabra de datos transmitida lleve sus propios bits de inicio y terminación a fin de que el receptor pueda saber dónde termina una palabra y comienza otra (figura 18.12). En este modo de transmisión, en general el transmisor y el receptor son remotos (el capítulo 20 da detalles de interfaces estándar). En una *transmisión síncrona*, transmisor y receptor tienen una misma señal de sincronización, lo que permite la sincronización de la transmisión y la recepción.

El microcontrolador MC68HC11 (vea la figura 15.9) tiene una interfase para comunicaciones en serie (SCI, *serial communications interface*) que se utiliza para la transmisión asíncrona y se emplea para comunicarse con dispositivos periféricos remotos. En la SCI la terminal PD1 del puerto D se utiliza como línea de transmisión y el

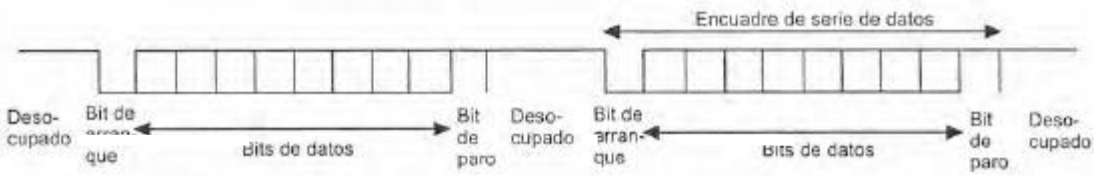


Figura 18.12 Transmisión asíncrona

puerto PD0 como línea de recepción. Estas líneas se activan o desactivan mediante el registro de control de la SCI. El microcontrolador también tiene una interfase para dispositivo periférico en serie (SPI, *serial peripheral interface*) para la transmisión síncrona. Se utiliza en comunicaciones en serie locales; locales significa comunicaciones dentro de la máquina en donde se encuentra el chip.

18.4 Adaptador de interfase para dispositivos periféricos

Es posible diseñar interfases para entrada/salida específicas; pero también existen dispositivos para interfases de entrada/salida programables; los cuales permiten elegir entre diversas opciones de entrada y salida a través del software. Estos dispositivos se conocen como *adaptadores de interfase para dispositivos periféricos*, (PIA, *peripheral interface adapters*).

Una interfase en paralelo PIA de uso común es la *MC6821* de *Motorola*. Es parte de la familia MC6800, por lo que se puede conectar en forma directa a los buses MC6800 y MC68HC11 de Motorola. Se puede decir que este dispositivo consta en escencia de dos puertos de entrada/salida en paralelo, con su lógica de control para conectarse con el microprocesador principal. La figura 18.13 muestra la configuración básica del PIA MC6821, y las conexiones.

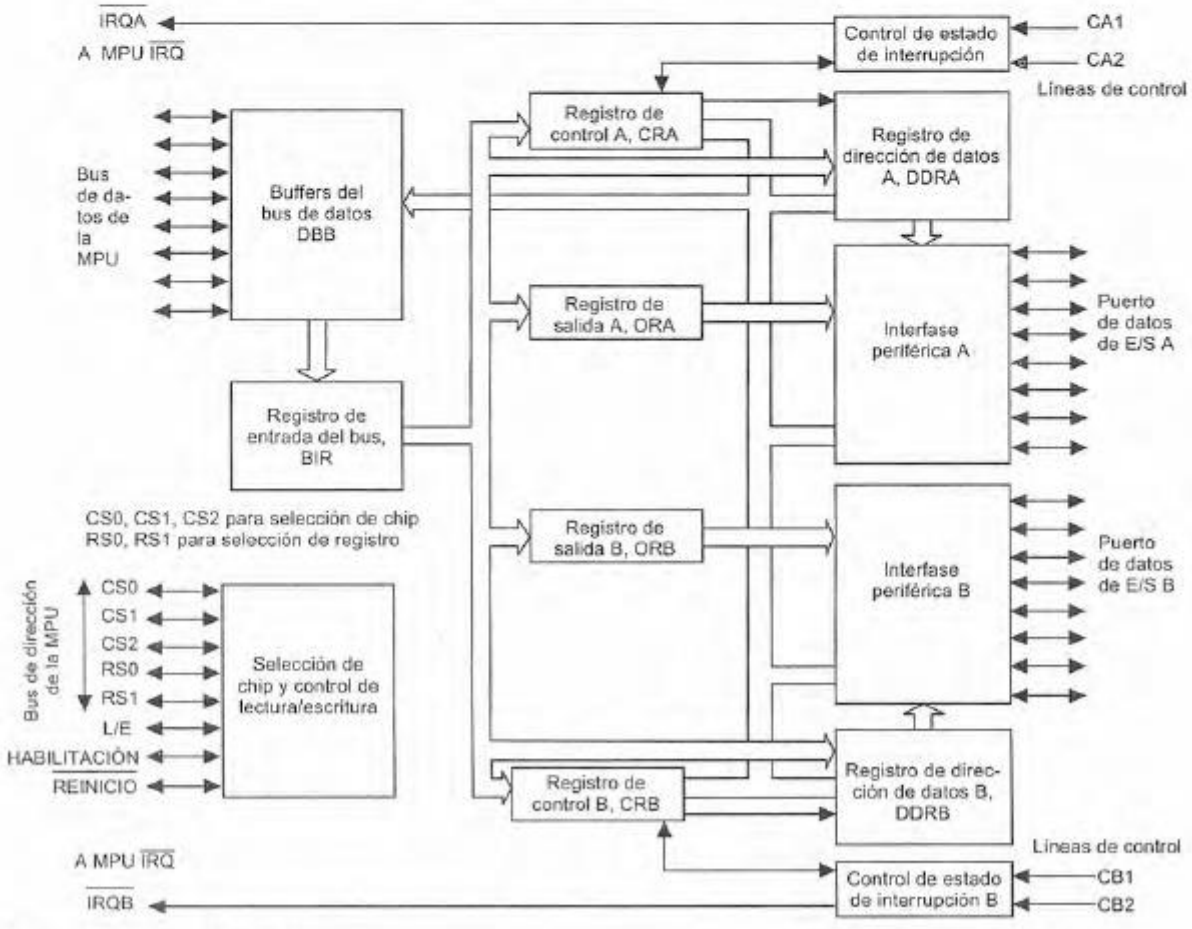


Figura 18.13 PIA MC6821

El PIA contiene dos puertos de datos paralelos de 8 bits, denominados A y B. Cada puerto tiene:

1. Un *registro de interfase para dispositivo periférico*. El funcionamiento de un puerto de salida difiere del de entrada pues debe guardar los datos para el dispositivo periférico. Para la salida se usa un registro que guarda temporalmente los datos. Se dice que el registro está *cerrado*, es decir, conectado, cuando un puerto se usa como salida, y abierto si se usa como entrada.

2. Un *registro de la dirección* o *sentido de los datos* que determina si las líneas de entrada/salida son entradas o salidas.

3. Un *registro de control* para determinar las conexiones lógicas activas en el dispositivo periférico.

4. Dos *líneas de control*, CA1 y CA2, o CB1 y CB2.

Dos líneas de dirección del microprocesador conectan el PIA con dos líneas de selección de registro, RS0 y RS1. Esto da al PIA cuatro direcciones para los seis registros. Si RS1 es bajo, se direcciona el lado A y cuando es alto, el lado B. RS0 direcciona los registros a un lado en particular, es decir, A o B. Cuando RS0 es alto, se direcciona el registro de control, y cuando es bajo, el registro de datos o el registro de dirección de datos. Para un lado en particular, el registro de datos y el registro de dirección de datos tienen la misma dirección. Cuál de ellos se direccione dependerá del bit 2 del registro de control (vea adelante).

Los bits de los registros de control A y B están relacionados con las funciones que se realizan en los puertos. Entonces en el registro de control A están los bits que muestra la figura 18.14. En el registro de control B se utiliza una configuración similar.

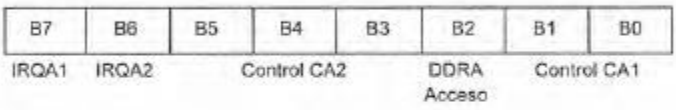


Figura 18.14 Registro de control

Bits 0 y 1

Los primeros dos bits controlan la forma en que funcionan las líneas de control de entrada CA1 o CB1. El bit 0 determina si es posible la salida de la interrupción. B0 = 0 desactiva la interrupción del microprocesador IRQA (B), B0 = 1 activa la interrupción. CA1 y CB1 no están definidos por el nivel estático de la entrada, pero se activan por flancos, es decir, por la variación de una señal. El bit 1 define si el bit 7 se determina por una transición de alto a bajo (flanco de bajada), o por una transición de bajo a alto (flanco de subida). B1 = 0 define una transición de alto a bajo, B1 = 1 define una transición de bajo a alto.

Bit 2

El bit 2 determina si se direccionan los registros de dirección de datos o los registros de datos del dispositivo periférico. Si B2 se define como 0, se direccionan los registros de dirección de datos, y si B2 es 1, se eligen los registros de datos de dispositivos periféricos.

Bits 3, 4 y 5

Estos bits permiten que el PIA realice diversas funciones. El bit 5 determina si la línea de control 2 es una entrada o una salida. Si el bit 5 se define como 0, la línea de control 2 es una entrada; si se define como 1, es una salida. En el modo de entrada, CA2 y CB2 funcionan de la misma manera. Los bits 3 y 4 determinan si la salida de la interrupción está activa y qué tipo de transiciones definen al bit 6.

Cuando B5 = 0, es decir, CA2(CB2) se define como entrada: B3 = 0 desactiva la interrupción del microprocesador IRQA(B) debido a CA2(CB2), B3 = 1 activa la interrupción del microprocesador IRQA(B) debido a CA2(CB2); B4 = 0 determina el que el indicador de interrupción IRQA(B), bit B6, se define por una transición de alto a bajo en CA2(CB2), B4 = 1 determina que se define por una transición de bajo a alto.

B5 = 1 define CA2(CB2) como salida. En el modo de salida CA2 y CB2 se comportan de diferente manera. En CA2: si B4 = 0 y B3 = 0, CA2 disminuye durante la primera transición ENABLE (E) de alto a bajo y a continuación el microprocesador lee el registro A de datos del dispositivo periférico, regresando a alto en la siguiente transición CA1; B4 = 0 y B3 = 1, CA2 disminuye durante la primera transición ENABLE, de alto a bajo y a continuación el microprocesador lee el registro A de datos del dispositivo periférico, regresando a alto durante la siguiente transición ENABLE de alto a bajo. Para CB2: si B4 = 0 y B3 = 0 , CB2 disminuye en la primer transición ENABLE bajo a alto, y a continuación el microprocesador escribe en el registro de datos de dispositivos periféricos B, regresando a alto durante la siguiente transición CB1; B4 = 0 y B3 = 1, CB2 disminuye en la primera transición ENABLE de bajo a alto, y el microprocesador escribe en el registro de datos de dispositivos periféricos B, volviendo a alto durante la siguiente transición ENABLE de bajo a alto. En B4 = 1 y B3 = 0, CA2(CB2) disminuye cuando el microprocesador escribe B3 = 0 en el registro de control. En B4 = 0 y B3 = 1, CA2 (CB2) aumenta cuando el microprocesador escribe B3 = 1 en el registro de control.

Bit 6

Éste es el indicador de interrupción CA2(CB2), definido por las transiciones en CA2(CB2). Si CA2(CB2) es una entrada (B5 = 0), se borra cuando el microprocesador lee el registro de datos A(B). Si CA2(CB2) es la salida (B5 = 1), el indicador es 0 y no lo afectan las transiciones CA2(CB2).

Bit 7

Es el indicador de interrupción CA1(CB1) y se borra si el microprocesador lee el registro de datos A(B).

El proceso de elección de las opciones empleadas se denomina *configuración* o *inicialización* del PIA. La conexión RESET se usa para borrar todos los registros del PIA, el cual se debe inicializar.

18.4.1 Inicialización del PIA

Antes de utilizar el PIA se debe elaborar y utilizar un programa que defina las condiciones del flujo de datos periféricos deseadas. El programa del PIA se coloca al inicio del programa principal para que desde el inicio el microprocesador lea los datos de los dispositivos periféricos. El programa de inicialización sólo se ejecuta una vez.

El programa de inicialización que define cuál puerto es el de entrada y cuál el de salida es como el siguiente:

1. Borre el bit 2 de los registros de control mediante un reinicio, de manera que se direccionen los registros de dirección de datos. El registro de dirección de datos A se direcciona como XXX0 y el registro de dirección de datos B como XXX2.
2. Para que A sea un puerto de entrada, cargue todos los 0s en el registro de dirección A.
3. Para que B sea un puerto de salida, cargue todos los 1 en el registro de dirección B.
4. Cargue 1 en el bit 2 de los dos registros de control. El registro de datos A ahora se direcciona como XXX0 y el registro de datos B como XXX2.

De esta manera, el programa de inicialización en lenguaje ensamblador para definir el lado A como la entrada y el lado B como la salida, después de un reinicio, es:

```
INIT    LDAA    #$00    Carga los 0
        STAA    $2000   Define al lado A como puerto de entrada
        LDAA    #$FF    Carga los 1
        STAA    $2000   Define al lado B como puerto de salida
        LDAA    #$04    Carga 1 en el bit 2, y en los demás bits, 0
        STAA    $2000   Elige el registro de datos del puerto A
        STAA    $2002   Elige el registro de datos del puerto B
```

Con la instrucción LDAA 2000, los datos se leen en el puerto de entrada A y con la instrucción STAA 2002 el microprocesador escribe datos de dispositivo periférico en el puerto de salida.

18.4.2 Conexión de señales de interrupción a través del PIA

El PIA MC6821 de Motorola (figura 18.15) tiene dos conexiones, IRQA e IRQB, a través de las cuales se envían señales de interrupción al microprocesador; cuando CA1, CA2 o CB1, CB2 envían una solicitud de interrupción, impulsan la terminal IRQ del microprocesador al estado activo con valor bajo. Cuando en la sección anterior, se consideró el programa de inicialización de un PIA, sólo el bit 2 del registro de control se definió como 1; los otros se definieron como 0. Estos ceros desactivaron las entradas de las interrupciones. Para utilizar las interrupciones, se debe modificar el paso de la inicialización

que guarda S04 en el registro de control. La forma de modificación dependerá del tipo de cambio de la entrada requerida para iniciar la interrupción.

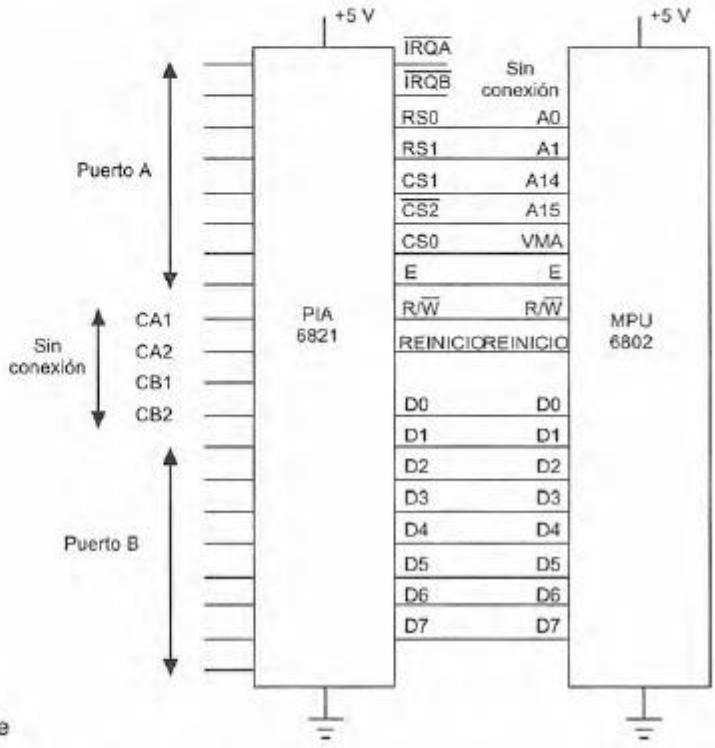


Figura 18.15 Acoplamiento mediante interfase con un PIA

Suponga, por ejemplo, que se requiere que CA1 active una interrupción cuando se presenta una transición de alto a bajo; CA2 y CB1 no se utilizan, y activa CB2, utilizándolo para la salida de definición/reinicio. El formato de registro de control para satisfacer estas especificaciones para CA es:

B0 es 1 para activar la interrupción en CA1.
B1 es 0 para que el indicador de interrupción IRQA1 se defina por una transición de alto a bajo en CA1.
B2 es 1 para dar acceso al registro de datos.
B2 es 1 para dar acceso al registro de datos.
B3, B4 y B5 son 0 porque CA2 está desactivado.
B6 y B7 son indicadores sólo de lectura, por lo que es posible utilizar un 0 o un 1.

Por lo tanto el formato de CA1 podría ser 00000101, es decir, 05 en notación hexadecimal. El formato del registro de control de CB2 es:

B0 es 0 para desactivar CB1.
B1 puede ser 0 o 1 dado que CB1 está desactivado.
B2 es 1 para permitir el acceso al registro de datos.
B3 es 0, B4 es 1 y B5 es 1, para elegir definir/reinicio.
B6 y B7 son indicadores sólo de lectura, por lo que se usan 0 o 1.

Por lo tanto, el formato para CA1 sería 00110100, es decir, 34 en notación hexadecimal. El programa de inicialización sería:

```
INIT  LDAA  #$00   Carga 0s
      STAA  $2000  Define al lado A como puerto de entrada
      LDAA  #$FF   Carga 1s
      STAA  $2000  Define al lado B como puerto de salida
      LDAA  #$05   Carga el formato del registro de control requerido
      STAA  $2000  Elige el registro de datos del puerto A
      LDAA  #$34   Carga el formato del registro de control requerido
      STAA  $2002  Elige el registro de datos del puerto B
```

18.4.3 Ejemplo de conexión de una interfase con un PIA

La figura 18.14 es un ejemplo de conexión de una interfase con un PIA: en ella se muestra un circuito que se usa para un motor paso a paso unipolar (sección 7.7.2). Al conectar los devanados inductivos se puede generar una fem de regreso de magnitud considerable, por lo que es necesario disponer de algún medio para aislar los devanados del PIA. Se pueden usar optoaisladores, diodos o resistencias. Con los diodos se obtiene una interfase sencilla y barata; en cambio, las resistencias no aislan por completo el PIA.

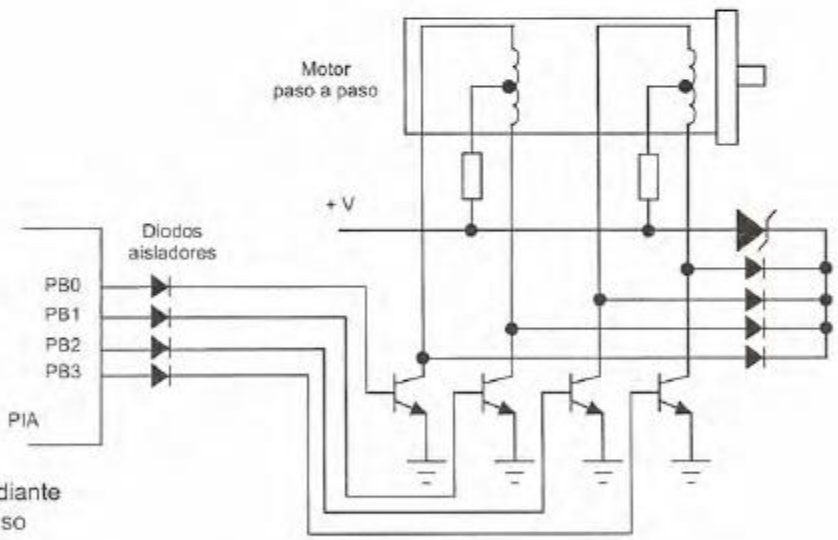


Figura 18.16 Acoplamiento mediante interfase con un motor paso a paso

18.5 Interfase para comunicaciones en serie

El *receptor/transmisor asíncrono universal* (UART, *universal asynchronous receiver/transmitter*) es el elemento esencial de un sistema de comunicaciones en serie; su función es cambiar los datos en serie a datos en paralelo en la entrada y datos en paralelo a datos en serie en la salida. Una forma programable de UART muy común es el *adaptador de interfase para comunicaciones asíncronas* (ACIA, *asynchronous communications interface adapter*) MC6850 de Motorola. La figura 18.17 ilustra un diagrama de bloques de los elementos que lo componen.

El flujo de datos entre el microprocesador y el ACIA se da a través de ocho líneas bidireccionales, D0 a D7. El microprocesador controla la dirección del flujo de datos mediante la entrada de lectura/escritura que se dirige al ACIA. Las tres líneas de selección de chip sirven para seleccionar determinados registros del ACIA. Si la línea de selección de registro tiene valor alto, se eligen los registros de transmisión de datos y de recepción de datos; si el valor es bajo, se eligen los registros de control y de estado. El registro de estado contiene información del estado de las transferencias de datos durante su realización, información que se utiliza para leer las líneas de detección de portadora de datos y de listo para enviar. El registro de control al principio se utiliza para reiniciar el ACIA y después, para definir la velocidad de transferencia de datos en serie y el formato de los datos.

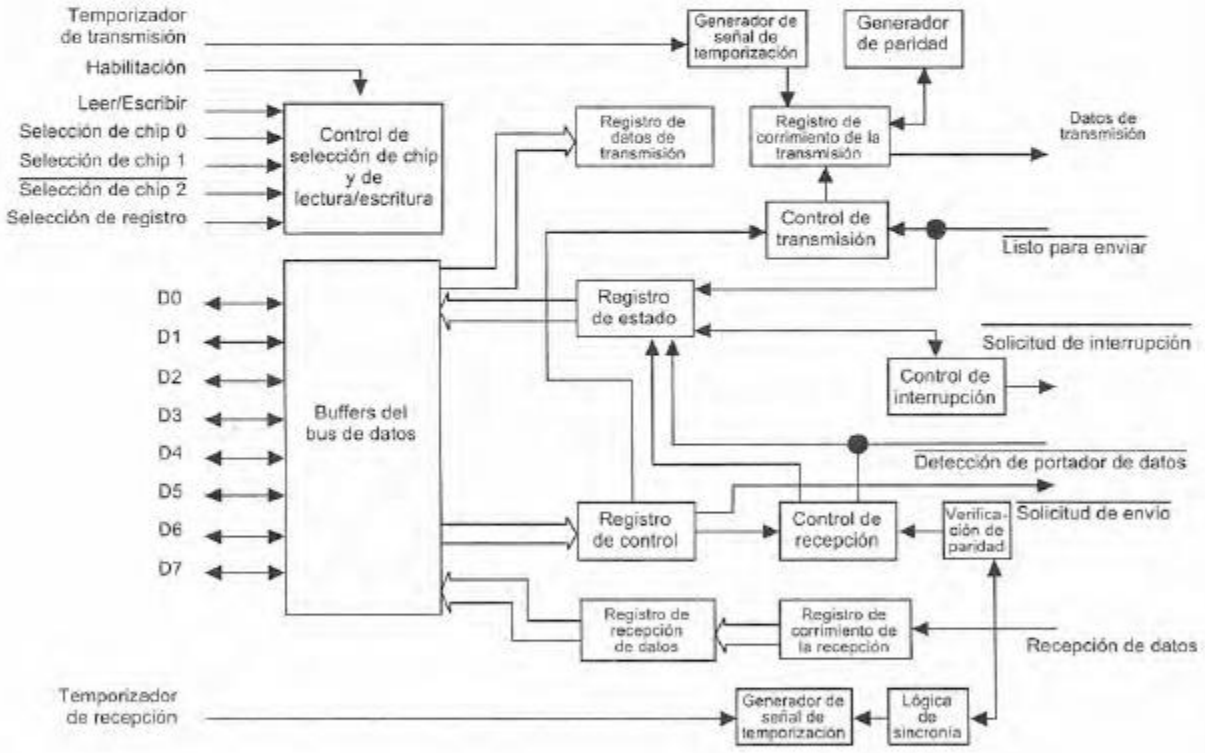


Figura 18.17 ACIA MC6850

La parte para dispositivos periféricos del ACIA incluye dos líneas de datos en serie y tres líneas de control. Los datos se envían por la línea de transmisión de datos y se reciben por la línea de recepción de datos. Se cuenta con señales de control de listo para enviar, señal de control para detección de portadores de datos y señal de solicitud de envío. La figuras 18.18 y 18.19 muestran los formatos de bit de los registros de control y de estado, respectivamente.

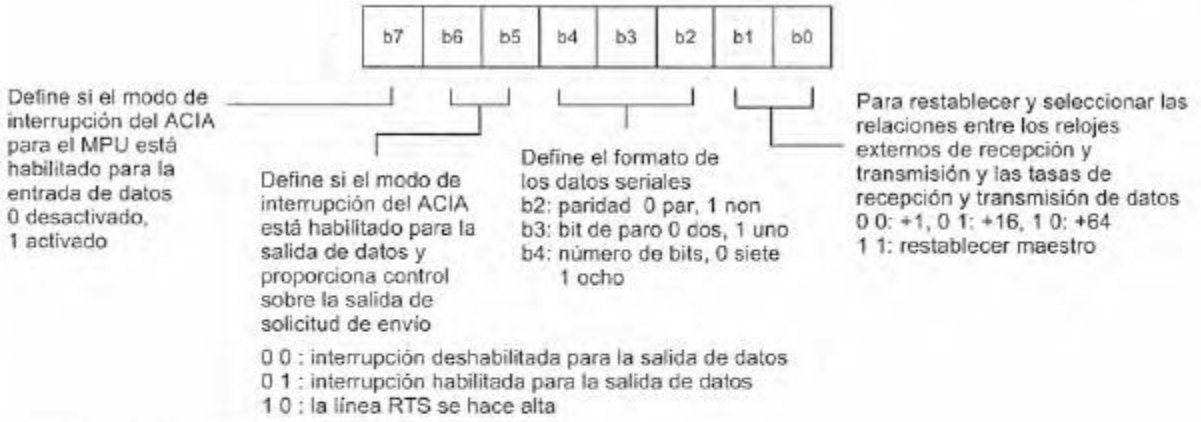


Figura 18.18 Registro de control

Figura 18.19 Registro de estado

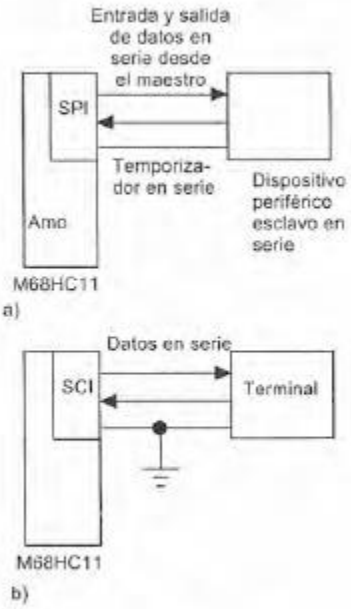


Figura 18.20 a) SPI, b) SCI

La transferencia de datos en serie asíncrona en general se usa para la comunicación entre dos computadoras, ya sea con o sin módem o entre una computadora y una impresora (vea en el capítulo 20 más detalles).

18.5.1 La interfase en serie del M68HC11 de Motorola

Muchos microcontroladores tienen interfases en serie, es decir, UART integrados. Por ejemplo, el M68HC11 tiene una interfase para dispositivos periféricos en serie (SPI), una interfase síncrona y una interfase para comunicaciones en serie (SCI), que es una interfase asíncrona (figura 15.9). La SPI requiere la misma señal de sincronización que usan tanto el microcontrolador como el dispositivo o dispositivos que se conectan en forma externa (figura 18.20a). Es posible conectar a la SPI varios microcontroladores. La SCI es una interfase asíncrona, y por ello es posible utilizar diferentes señales de sincronización entre su sistema y el dispositivo que se conecta de manera externa (figura 18.20b). Los microprocesadores para propósito general no cuentan con interfase para comunicaciones en serie, por lo que para usarlos es necesario utilizar un UART (como el MC6850 de Motorola). En algunas situaciones se requiere más de una interfase de comunicaciones en serie, y es necesario complementar el microcontrolador M68HC11 con una UART.

La SPI se inicializa por los bits del registro de control de SPI (SPCR) y en el registro de control de la dirección de envío de datos del puerto D (DDRD). El registro de estado SPI contiene bits de estado y de error. El SCI se inicializa utilizando el registro de control SCI 1, el registro de control SCI 2 y el registro de control de la velocidad en baudios. Los indicadores de estado están en el registro de estado del SCI. Para ver mayores detalles consulte: *Software and Hardware Engineering: Motorola M68HC11* de F.M. Cady (OUP, 1997) o *Microcontroller Technology: The 68HC11* de P. Spasov (Prentice-Hall, 1992, 1996). Detalles de un sistema similar de microcontroladores PIC16Cx de Microchip se encuentran en *Programming and Customizing the PIC Microcontrollers* de M. Predko (McGraw-Hill,1998).

18.6 Ejemplos de acoplamiento mediante interfase

Los siguientes son ejemplos de acoplamientos mediante interfases.

18.6.1 Acoplamiento mediante interfase en un visualizador de siete segmentos y un decodificador

Considere que se usa un microcontrolador para activar una unidad visualizadora con LED de siete segmentos (vea la sección 4.4). Un LED es un indicador de apagado-encendido; el número que aparezca en el visualizador dependerá de qué LEDs estén encendidos. La figura 18.21 muestra cómo usar un microcontrolador para activar un visualizador de ánodo común utilizando un controlador de decodificador (sección 14.6.3); este último recibe una entrada BCD y la convierte en un código adecuado para el visualizador.

En el decodificador 7447 las terminales 7, 1, 2 y 6 son las terminales de entrada del decodificador para la entrada BCD; las terminales 13, 12, 11, 10, 9, 15 y 14 son las salidas de los segmentos del visualizador. La terminal 9 del visualizador es el punto decimal. La tabla 18.1 muestra las señales de entrada y salida del decodificador.

Poner en blanco significa que ninguno de los segmentos está encendido. Esta acción se usa para evitar un 0 de encabezado cuando, hay, por ejemplo, tres unidades visualizadoras y sólo se desea que

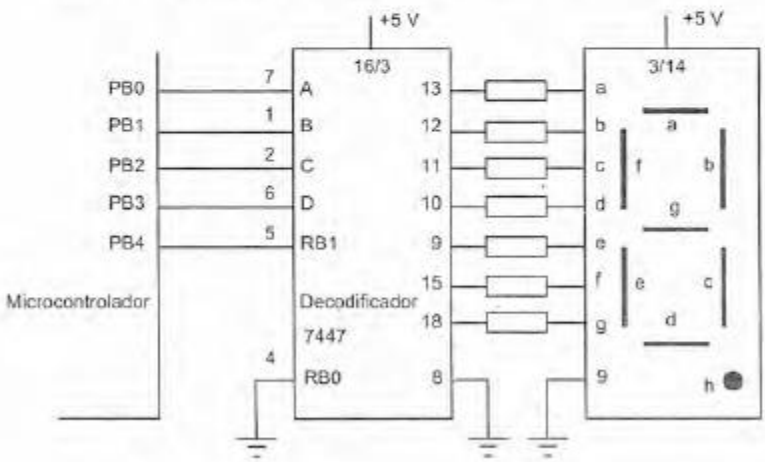


Figura 18.21 Manejo de un visualizador

Tabla 18.1 Decodificador BCD 7447 para un visualizador de siete segmentos.

Visualizador	Terminales de entrada				Terminales de salida						
	6	2	1	7	13	12	11	10	9	15	14
0	L	L	L	L	ON	ON	ON	ON	ON	ON	OFF
1	L	L	L	H	OFF	ON	ON	OFF	OFF	OFF	OFF
2	L	L	H	L	ON	ON	OFF	ON	ON	OFF	ON
3	L	L	H	H	ON	ON	ON	ON	OFF	OFF	ON
4	L	H	L	L	OFF	ON	ON	OFF	OFF	ON	ON
5	L	H	H	L	ON	OFF	ON	ON	OFF	ON	ON
6	L	H	H	L	OFF	OFF	ON	ON	ON	ON	ON
7	L	H	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
8	H	L	H	H	ON	ON	ON	OFF	OFF	OFF	OFF
9	H	L	H	L	ON	ON	ON	OFF	OFF	OFF	OFF

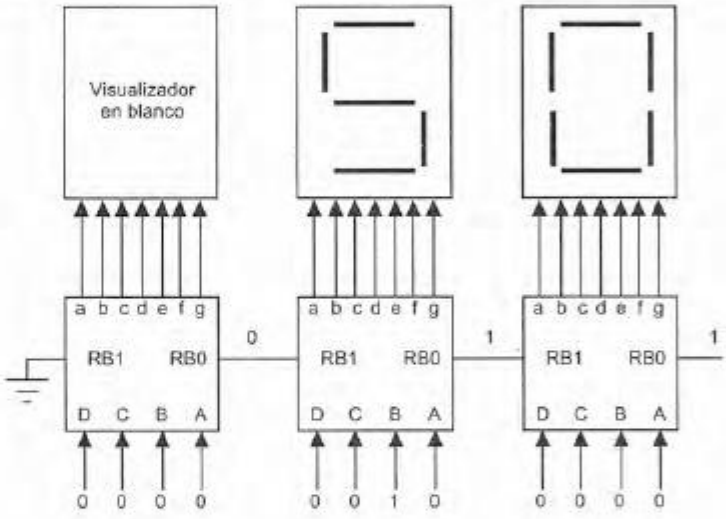


Figura 18.22 Puesta en blanco del acarreo

aparezca la lectura como 10 y no 010; para ello se pone en blanco el 0 de encabezado y se impide su iluminación. Para lograr esto se pone en valor bajo la *entrada para poner en blanco el acarreo*, RBI. Cuando RBI tiene un valor bajo y las entradas BCD A, B, C y D tienen valor bajo, la salida se pone en blanco. Si la entrada no es cero, la salida para poner en blanco el acarreo RBO tiene un valor alto, sin tener en cuenta cuál sea la condición en que se encuentre RBI. La RBO del primer dígito del visualizador se conecta a la RBI del segundo dígito y la RBO del segundo se conecta a la RBI del tercer dígito; así, se pone en blanco sólo el 0 final (figura 18.22).

En los visualizadores que tienen varios elementos, en vez de usar un decodificador por cada elemento, se utiliza la multiplexión y un solo decodificador. La figura 18.23 muestra el circuito del multiplexor de un visualizador de cuatro elementos tipo cátodo común. Los datos BCD salen por el puerto A y el decodificador muestra en todos los visualizadores la salida del decodificador. El cátodo común de éstos se conecta a tierra a través de un transistor. El visualizador no se encenderá a menos que el transistor se encienda como consecuen-

cia de una señal de salida del puerto B. Alternando entre PB0, PB1, PB2 y PB3, la salida del puerto A puede cambiar al visualizador adecuado. Para mantener una visualización constante, éste se enciende con suficiente frecuencia para que no se perciba el parpadeo del visualizador. Para presentar más de un dígito a la vez se puede usar la multiplexión por división de tiempo (vea la sección 3.7.2.).

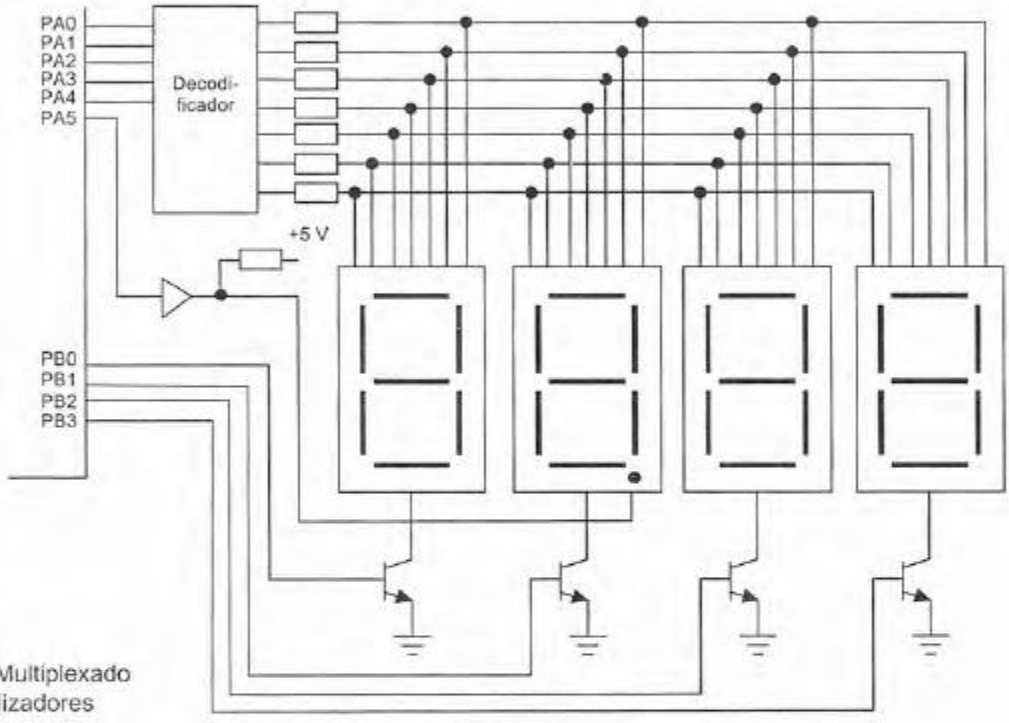


Figura 18.23 Multiplexado de cuatro visualizadores

18.6.2 Acoplamiento mediante interfase para señales analógicas

Cuando es necesario que la señal de salida producida por un microprocesador o un microcontrolador sea de tipo analógico, se lleva a cabo una conversión de señal digital a analógica. Por ejemplo, el DAC AD557 de Analog Devices se utiliza con este propósito. Este convertidor produce un voltaje de salida proporcional a su entrada digital y dispone de un latch de entrada para el acoplamiento mediante interfase del microprocesador. Si los latches no fueran necesarios, las terminales 9 y 10 se conectan a tierra. Los datos se bloquean cuando se produce un flanco positivo, es decir, un cambio de bajo a alto, en algunas de las entradas de la terminal 9 o la terminal 10. Los datos se retienen hasta que ambas terminales regresan al nivel bajo. Cuando esto sucede, los datos se transfieren del latch al convertidor digital a analógico para su conversión en voltaje analógico. La figura 18.24 muestra el AD557, en el cual el latch no se ha utilizado y está conectado a un M68HC11 de Motorola, de manera que al ejecutar el programa, genera un voltaje que es una señal diente de sierra. Otros tipos de forma de onda se pueden generar con facilidad cambiando el programa.

```
BASE    EQU   $1000     Dirección de base de registros de E/S
PORTB   EQU   $04       Desviación de PORTB respecto a BASE

        ORG   $C000
        LDX   #BASE     Punto X a base de registro
        CLR   PORTB,X   Enviar 0 al CAD
AGAIN   INC   PORTB,X   Incrementar en 1
        BRA   AGAIN     Repetir
        END
```

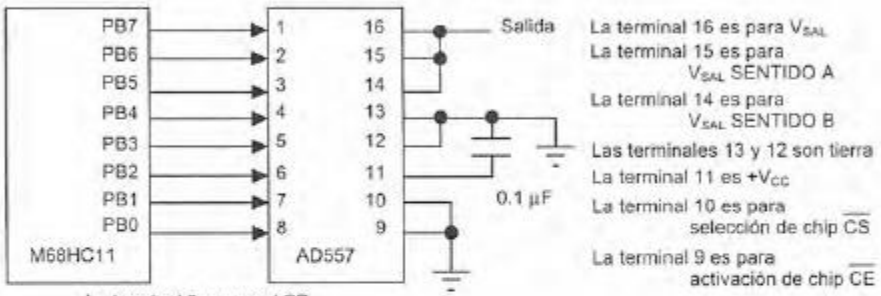


Figura 18.24 Generación de formas de onda

Problemas

1. Describa las funciones que puede realizar una interfase.
2. Explique la diferencia entre una interfase en paralelo y una interfase en serie.
3. Explique qué se entiende por un sistema de mapeo de memoria para entradas/salidas.
4. ¿Cuál es la función de un adaptador de interfase con dispositivo periférico (PIA)?
5. Describa la arquitectura del PIA MC6821 de Motorola.
6. Explique la función del programa de inicialización de un PIA.
7. ¿Qué ventajas ofrece utilizar las interrupciones externas en vez del muestreo por software como medio de comunicación con dispositivos periféricos?
8. En el PIA MC6821 de Motorola, ¿qué valor debe quedar guardado en el registro de control, si hay que desactivar CA1, CB1 debe ser una entrada de interrupción activada definida por una transición de bajo a alto, CA2 debe estar activada y se utiliza como salida para definir/reiniciar y CB2 debe ser activada y asumir un valor bajo durante la primera transición E de bajo a alto, siguiendo al microprocesador? Escriba en el registro de datos de dispositivos periféricos B y vuelva al valor alto durante la siguiente transición de bajo a alto E.
9. Escriba un programa en lenguaje ensamblador para inicializar el PIA MC6821 de Motorola, de manera que se cumplan las especificaciones del problema 8.
10. Escriba un programa en lenguaje ensamblador para inicializar el PIA MC6821 de Motorola, de manera que lea ocho bits de datos del puerto A.

19 Controladores lógicos programables

19.1 Controladores lógicos programables

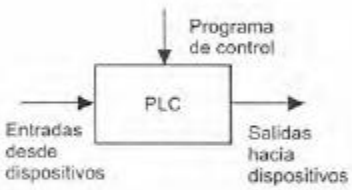


Figura 19.1 Controlador lógico programable (PLC)

Un *controlador lógico programable* (PLC, *programmable logic controller*) es un dispositivo electrónico digital (figura 19.1) que usa una memoria programable para guardar instrucciones y llevar a cabo funciones lógicas, de secuencia, de sincronización, de conteo y aritméticas para controlar máquinas y procesos y que se ha diseñado específicamente para programarse con facilidad. Este tipo de procesadores se denomina *lógico* debido a que la programación tiene que ver principalmente con la ejecución de operaciones lógicas y de conmutación. Los dispositivos de entrada (como interruptores) y los dispositivos de salida (como motores) que están bajo control se conectan al PLC, y después el controlador monitorea las entradas y salidas de acuerdo con el programa almacenado por el operador en el PLC con el que controla máquinas o procesos. En un principio, el propósito de estos controladores fue sustituir la conexión física de relevadores (como en la figura 7.2) de los sistemas de control lógicos y de sincronización. Los PLC tienen la gran ventaja de que permiten modificar un sistema de control sin tener que volver a alambrar las conexiones de los dispositivos de entrada y salida; basta con que el operador digite en un teclado las instrucciones correspondientes. También estos controladores son más rápidos que los sistemas a base de relevadores. El resultado es un sistema flexible que se puede usar para controlar sistemas muy diversos en su naturaleza y su complejidad. Tales sistemas se usan ampliamente para la implementación de funciones lógicas de control debido a que son fáciles de usar y programar.

Los PLC son similares a las computadoras, pero tienen características específicas que permiten su empleo como controladores. Estas características son:

1. Son robustos y están diseñados para resistir vibraciones, temperatura, humedad y ruido.
2. La interfase para las entradas y las salidas está dentro del controlador.
3. Es muy fácil programarlos, así como entender el lenguaje de programación. La programación básicamente consiste en operaciones de lógica y conmutación.

Los primeros PLC fueron concebidos en 1968. Hoy su uso está muy generalizado y hay una gran variedad, desde pequeñas unidades autónomas que cuentan quizás con apenas 20 entradas y salidas, hasta sistemas modulares para manejar grandes cantidades de entradas/salidas, manejar entradas/salidas digitales y analógicas y llevar a cabo modos de control PID.

Este capítulo analiza la estructura básica de los PLC y cómo se emplean en el control de máquinas y procesos. Si se desea profundizar en el tema, se sugieren obras especializadas como *Programmable Controllers, Operation and Application* de I.G. Warnock (Prentice-Hall, 1988), *Programmable Logic Controllers* de W. Bolton (Newnes, 1996, 2003) o *Automation with Programmable Logic Controllers* de P. Rohner (Macmillan, 1996).

19.2 Estructura básica

La figura 19.2 muestra la estructura interna básica de un PLC que, en escencia, consiste en una unidad central de procesamiento (CPU), memoria y circuitos de entrada/salida. La CPU controla y procesa todas las operaciones dentro del PLC. Cuenta con un temporizador cuya frecuencia típica es entre 1 y 8 MHz. Esta frecuencia determina la velocidad de operación del PLC y es la fuente de temporización y sincronización de todos los elementos del sistema. Un sistema de buses lleva información y datos desde y hacia la CPU, la memoria y las unidades de entrada/salida. Los elementos de la memoria son: una ROM para guardar en forma permanente la información del sistema operativo y datos corregidos; una RAM para el programa del usuario y memoria buffer temporal para los canales de entrada/salida.

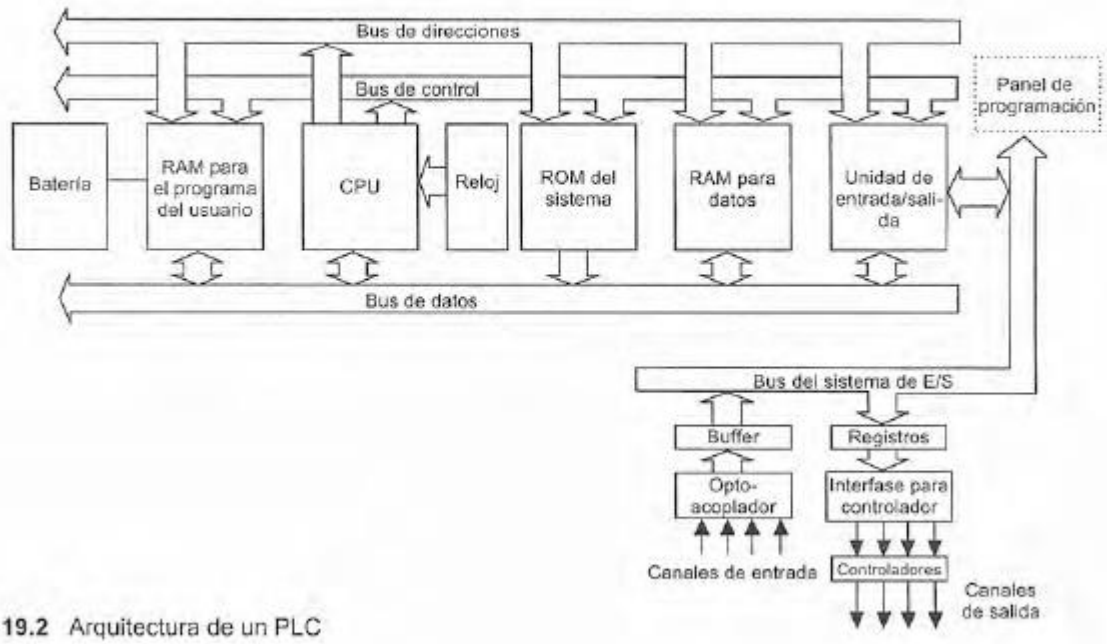


Figura 19.2 Arquitectura de un PLC

El usuario puede modificar los programas en la RAM. Sin embargo, para evitar que estos programas se pierdan durante una interrupción del suministro de energía eléctrica, en el PLC se utiliza una batería, para mantener el contenido de la RAM durante un periodo. Una vez elaborado un programa y guardado en la RAM, se puede cargar en un chip de memoria EPROM para que quede guardado de manera permanente. Las especificaciones de PLC pequeños con frecuencia indican la capacidad de la memoria del programa en función de la cantidad de pasos de programa que es posible guardar. Un paso de programa es la instrucción para que ocurra cierto evento. El programa puede consistir en varios pasos; por ejemplo: examinar el estado del interruptor A y del interruptor B; si A y B están cerrados, entonces dar energía al solenoide P, lo que tal vez resulte en la operación de un actuador. Cuando esto ocurre, puede iniciar otra tarea. Por lo general, un PLC pequeño puede manejar de 300 a 1000 pasos, más que suficiente para la mayoría de las aplicaciones de control.

19.2.1 Entrada/salida

La unidad de entrada/salida es la interfase entre el sistema y el mundo externo. Para introducir programas en esta unidad se usa un tablero, que puede variar de un pequeño teclado con pantalla de cristal líquido, a los que usan unidades de presentación visual (VDU, *visual display unit*) con teclado y pantalla. También es posible introducir los programas al sistema mediante un enlace con una computadora personal (PC) que se carga con un paquete de software apropiado.

Los canales de entrada/salida proporcionan funciones para el acondicionamiento y aislamiento de señales, lo que permite conectarlos directamente a sensores y actuadores, sin necesidad de otros circuitos. La figura 19.3 muestra la configuración básica de un canal de entrada. Los voltajes de entrada comunes son 5 V y 24 V.

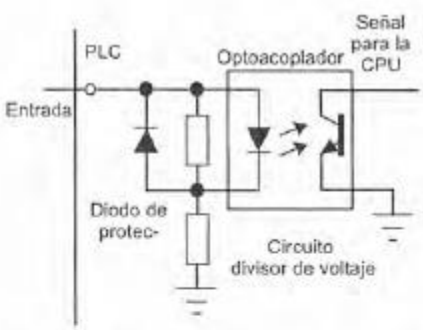


Figura 19.3 Canal de entrada

Los voltajes comunes de salida son 24 V y 240 V. La especificación del tipo de las salidas generalmente es tipo relevador, tipo transistor o tipo triac. En el tipo relevador (figura 19.4), la señal de la salida del PLC se utiliza para operar un relevador; por lo que éste es capaz de conmutar corrientes del orden de unos pocos amperes en un circuito externo. El relevador aísla al PLC del circuito externo, y se emplea tanto para la conmutación de cd como la de ca. Sin embargo, los relevadores funcionan con relativa lentitud. En la salida tipo transistor (figura 19.5) se utiliza un transistor para conmutar corriente a través de un circuito externo. El transistor realiza la conmutación con mayor rapidez. En la figura 19.5a, usando la convención de la dirección del flujo de corriente de positivo a negativo, un dispositivo de salida recibe la corriente de un módulo de salida y lo que se conoce como drenado de corriente (*sinking*), en la figura 19.5b la corriente fluye del módulo de salida hacia una carga y esto se conoce como suministro de corriente (*sourcing*). Los optoaisladores se usan con transistores de conmutación para lograr el aislamiento entre los circuitos externos y el PLC. La salida tipo transistor sólo se utiliza en

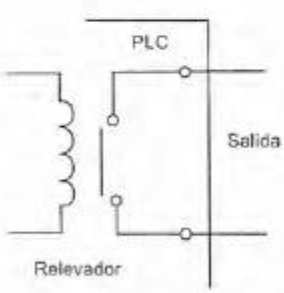


Figura 19.4 Salida tipo relevador

la conmutación de cd. Las salidas tipo triac se usan para controlar cargas externas que se conectan a la fuente de alimentación de ca. En este caso también se emplean optoaisladores.

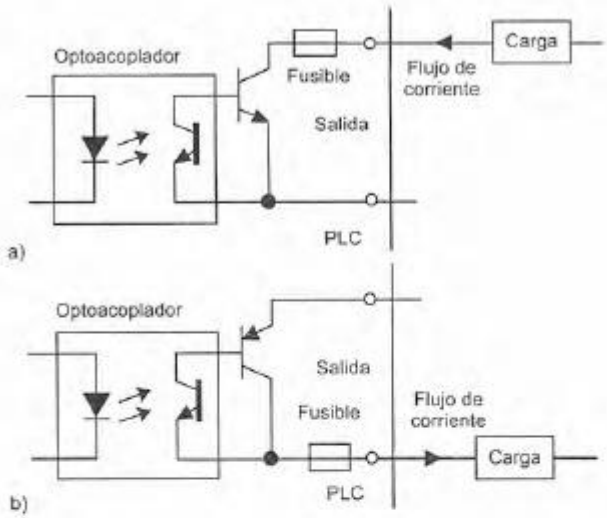


Figura 19.5 Salida tipo transistor a) drenado de corriente, b) suministro de corriente

19.2.2 Ejemplo de un PLC

Las siguientes son algunas de las características de un típico PLC pequeño, el Mitsubishi F2-20MR-ES.

Alimentación eléctrica:	110-120 V/220-240 V c.a., unifásica 50/60 Hz
Lenguaje de programación:	Lógica de escalera
Capacidad de programación:	1000 pasos
Velocidad de ejecución:	7 μs/paso en promedio
Memoria del programa:	CMOS-RAM incorporada; es posible añadir una EPROM
Batería de respaldo:	Batería de litio, de unos 5 años de vida
Temporizadores:	Temporizador de 0.1 s: 24 puntos, temporizadores de retardo a la activación (0.1 a 999 s) Temporizador de 0.01 s: 8 puntos, temporizadores de retardo a la activación (0.01 a 99.9 s)
Contadores (retentivo):	Contador regresivo (0 a 999), 32 puntos
Cantidad de entradas:	12 puntos, todas optoaisladas
Voltaje de entrada:	Incorporado de 24 V c.d., externo de 24 V c.d.
Cantidad de salidas:	8 puntos
Opciones de salida:	Salida tipo relevador: relevador aislado Salida tipo transistor: optoaislado Salida tipo triac: optoaislado

19.3 Procesamiento de la entrada/salida

La forma básica de programación más común en los PLC es la *programación en lenguaje de escalera*. Ésta especifica cada una de las tareas de un programa como si fueran los escalones de una escalera. En cada escalón se especifica, por ejemplo, la revisión de los interruptores A y B (las entradas); si ambos están cerrados, se proporciona energía a un solenoide (la salida). En la siguiente sección se analiza con más detalle la programación en lenguaje de escalera.

La secuencia que sigue un PLC para realizar un programa se resume de la siguiente manera:

1. Explorar las entradas asociadas a un escalón del programa de escalera.
2. Resolver la operación lógica que involucra esas entradas.
3. Encender/apagar las salidas de ese escalón.
4. Continuar con el siguiente escalón y repetir los pasos 1, 2 y 3.
5. Continuar con el siguiente escalón y repetir los pasos 1, 2 y 3.
6. Continuar con el siguiente escalón y repetir los pasos 1, 2 y 3.

Y así sucesivamente, hasta finalizar el programa.

Los escalones del programa en lenguaje escalera se exploran de acuerdo con la secuencia respectiva.

Existen dos métodos para el procesamiento de entradas/salidas:

1. *Por actualización continua*
   En este caso, la CPU explora los canales de entrada de acuerdo con la secuencia del programa. Cada punto de entrada se revisa por separado y se determina su efecto en el programa. Existe un retardo inherente, por lo general de unos 3 ms, cuando se revisa cada una de las entradas para garantizar que el microprocesador sólo lea señales de entrada válidas. Este retardo evita que el microprocesador cometa el error de contar una señal de entrada dos o más veces, si hay rebotes de los contactos en el interruptor. Antes de que el programa envíe la instrucción para ejecutar una operación lógica y se produzca una salida, se exploran varias entradas, cada exploración con un retardo de 3 ms. Las salidas quedan retenidas de manera que su estado se mantiene hasta la siguiente actualización.

2. *Copiado masivo de entradas/salidas*
   Dado que con la actualización continua se produce un retardo de 3 ms por cada entrada, el tiempo total para revisar cientos de puntos de entrada/salida puede ser comparativamente largo. Para que el programa se ejecute más rápido, un área específica de la RAM se utiliza como memoria intermedia o buffer entre la unidad de lógica de control y la unidad de entrada/salida. Cada entrada/salida tiene una dirección en esta memoria. Al inicio de cada ciclo de programa, la CPU muestrea todas las entradas y copia sus estados en las direcciones de entrada/salida de la RAM. Conforme se ejecuta el programa, se leen los datos de entrada guardados en la RAM, según se requiera y se ejecutan las operaciones lógicas. Las señales de salida producidas se guar-

dan en la sección reservada para entrada/salida en la RAM. Al término de un ciclo de programa, las salidas se envían de la RAM a los canales de salida. Las salidas quedan retenidas para que conserven su estado hasta la siguiente actualización.

19.4 Programación

La programación de un PLC basada en *diagramas de escalera* consiste en la elaboración de un programa de manera similar a como se dibuja un circuito de contactos eléctricos. El diagrama de escalera tiene dos líneas verticales que representan las líneas de alimentación. Los circuitos se disponen como líneas horizontales, es decir, como escalones de una escalera, sujetos entre las dos líneas verticales. La figura 19.6 muestra los símbolos estándar básicos que se utilizan, así como un ejemplo de escalones en un diagrama de escalera.

Cuando se dibuja la línea de circuito de un escalón, las entradas siempre preceden a las salidas y debe haber por lo menos una salida por cada línea. Los escalones deben empezar con una o varias entradas y terminar con una salida.

Las entradas y las salidas están numeradas y la notación utilizada depende del fabricante del PLC; por ejemplo, en la serie F de PLC Mitsubishi antes de un elemento de entrada hay una X y antes de un elemento de salida, una Y; la numeración empleada es la siguiente:

Entradas	X400-407, 410-413 X500-507, 510-513	(24 entradas posibles)
Salidas	Y430-437 Y530-537	(16 salidas posibles)

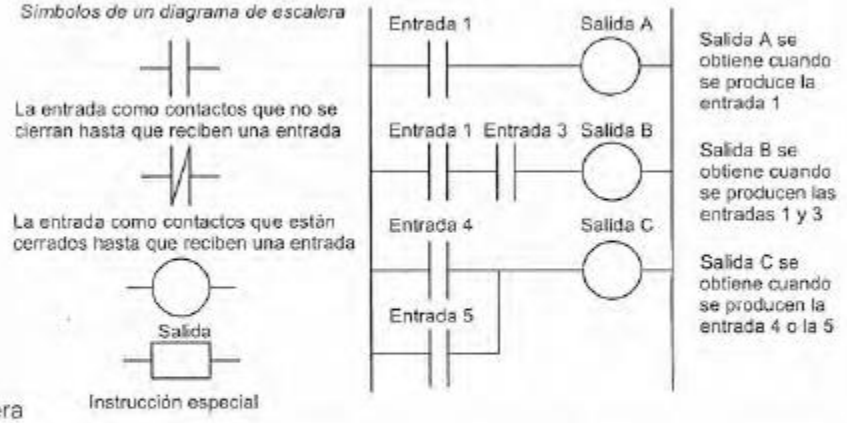


Figura 19.6 Diagrama tipo escalera

Para ilustrar cómo se dibuja un diagrama de escalera, considere la salida de un PLC que sirve para energizar un solenoide cuando el interruptor de arranque normalmente abierto, conectado a la entrada, se activa al cerrarlo (figura 19.7a). El programa necesario se muestra en la figura 19.7b. Empezando por la entrada, encontramos el sím-

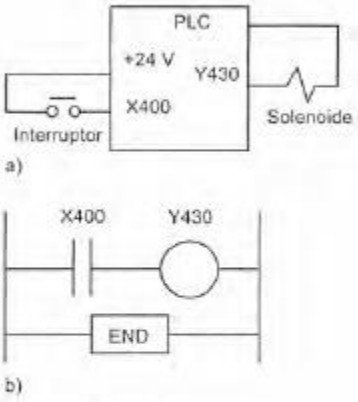


Figura 19.7 Interruptor controlando una solenoide

bolo para normalmente abierto | |, que puede tener dirección de entrada X400. La línea termina en la salida, la solenoide, cuyo símbolo es O y cuya dirección de salida es Y430. Para indicar la terminación del programa se marca el peldaño final. Cuando se cierra el interruptor, se activa la solenoide. Esto podría, por ejemplo, ser una válvula de solenoide que se abre para que entre agua en un recipiente.

Otro ejemplo es un control de temperatura encendido-apagado (figura 19.8), en el cual la entrada varía de un valor bajo a uno alto cuando el sensor de temperatura llega a la temperatura de calibración. En este momento, la salida cambia de encendido a apagado. El sensor de temperatura mostrado en la figura es un termistor en una configuración puente con la salida conectada a un amplificador operacional configurado como comparador (sección 3.2.7). El programa muestra la entrada como un contacto normalmente cerrado, produciendo la señal de encendido y la salida. Cuando se abre el contacto, se produce la señal de desconexión y la salida se apaga.

Para introducir estos programas de escalera se pueden utilizar teclados especiales, o seleccionarlos en la pantalla de una computadora usando el ratón. También se especifican mediante lenguaje mnemónico. Una vez introducidos, el PLC traduce estos programas a lenguaje de máquina para que el microprocesador y sus elementos respectivos puedan utilizarlos.

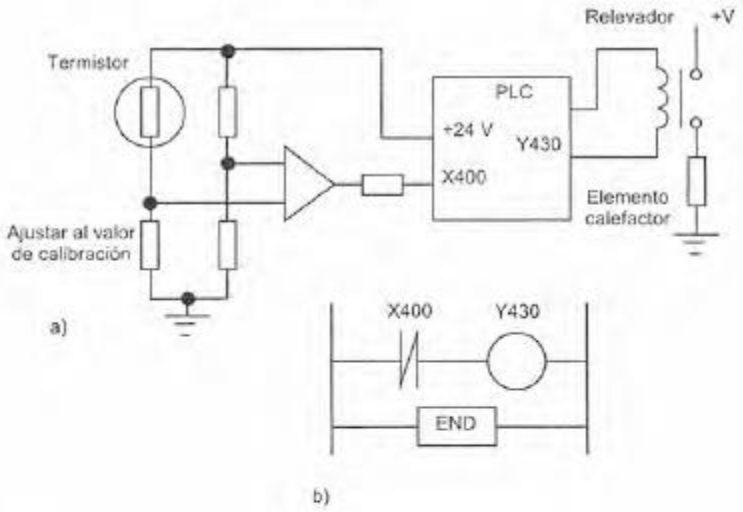


Figura 19.8 Sistema de control de temperatura

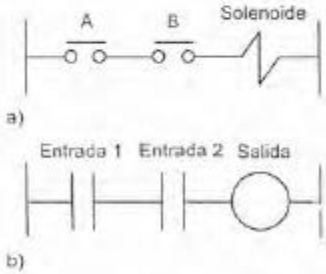


Figura 19.9 Un sistema AND

19.4.1 Funciones lógicas

Las funciones lógicas se pueden obtener con una combinación de interruptores (sección 14.3), ahora se verá cómo se pueden escribir programas tipo escalera para esas combinaciones.

1. AND
   La figura 19.9a muestra una bobina que no se energiza a menos que dos interruptores, en general abiertos, se cierren. Si los interruptores A y B están cerrados, se obtiene la función lógica

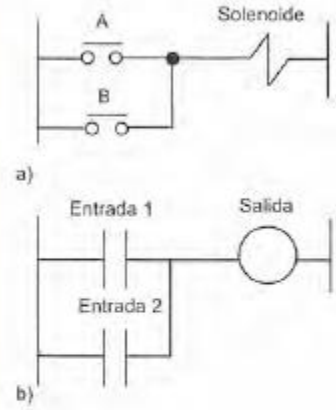


Figura 19.10 Un sistema OR

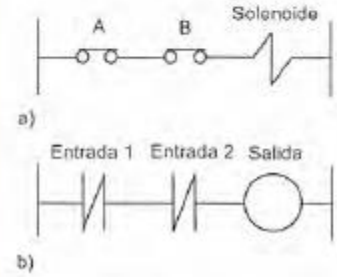


Figura 19.11 Un sistema NOR

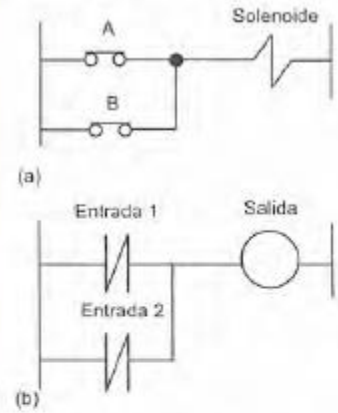


Figura 19.12 Un sistema NAND

Figura 19.13 Un sistema XOR

AND. El diagrama de escalera empieza con | |, que es la entrada identificada como 1 y representa al interruptor A conectado en serie con | |, entrada identificada como 2, la cual representa al interruptor B. La línea termina con ○ para representar a la salida. La figura 19.9b muestra la línea.

2. OR
   La figura 19.10a ilustra una bobina que no se energiza hasta que uno de los interruptores A o B, en general abiertos, se cierra, situación que corresponde a una compuerta lógica OR. El diagrama de escalera empieza con | |, denominado entrada 1, que representa al interruptor A, el cual está conectado en paralelo con | |, denominado entrada 2, que representa al interruptor B. La línea termina con ○, que representa a la salida. La figura 19.10b muestra la línea.

3. NOR
   La figura 19.11 muestra cómo representar el diagrama del programa de escalera para una compuerta NOR. Dado que debe haber una salida cuando ni A ni B tengan entrada, entonces cuando existe entrada en A o en B no hay salida, el programa escalera muestra la entrada 1 en serie con la entrada 2, ambas representadas por contactos normalmente cerrados.

4. NAND
   La figura 19.12 muestra una compuerta NAND. No hay salida cuando tanto A como B tienen una entrada. El diagrama del programa de escalera indica que para que haya salida se requiere que no haya entradas en la entrada 1 ni en la 2.

5. XOR
   La figura 19.13 muestra cómo dibujar el diagrama de un programa escalera para una compuerta XOR, donde no hay salida cuando no hay entrada para la entrada 1 ni para la entrada 2 y tampoco cuando hay entrada tanto en la entrada 1 como en la entrada 2. Observe que las entradas están representadas por dos juegos de contactos, uno normalmente abierto y otro normalmente cerrado.

Considere una situación en la que el interruptor A, normalmente abierto, debe activarse junto con uno de los otros interruptores B o C, normalmente abiertos, para activar un solenoide. Esta configuración se representa como la conexión del interruptor A en serie con dos interruptores en paralelo, B y C (figura 19.14a). Para energizar la bobina A y B o C deben estar cerrados. El interruptor A, con los interruptores en paralelo produce una situación lógica AND. Los dos interruptores que están en paralelo producen una situación lógica OR. De esta manera, se presenta una combinación de dos compuertas. La tabla de verdad es la siguiente:

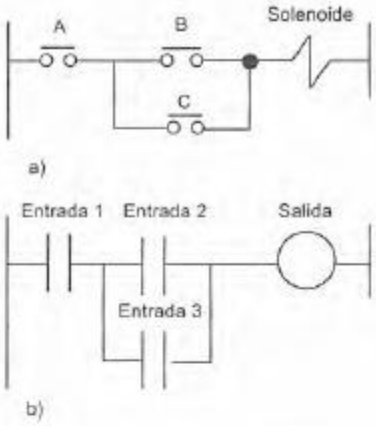


Figura 19.14 Solenoide controlado por interruptores

Entradas			Salida
A	B	C	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

El diagrama de escalera empieza con | | identificado como entrada 1 para representar al interruptor A. Éste se conecta en serie con dos | | en paralelo, denominados entrada 2 y entrada 3, que representan a los interruptores B y C. La línea termina con ○ para representar la salida, es decir, la solenoide. La figura 19.14b muestra el diagrama.

Un ejemplo sencillo de un programa que usa compuertas lógicas es el siguiente. Suponga que se desea producir una salida a la solenoide que controla la válvula con la que se abre la puerta de una tienda cuando el encargado cierra un interruptor para abrir la tienda y cuando un cliente se aproxima a la puerta y es detectado por un sensor que produce una señal. La tabla de verdad de este sistema es:

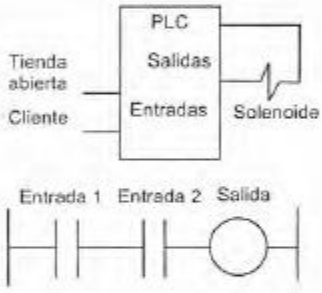


Figura 19.15 Sistema de la puerta de una tienda

Interruptor para abrir la tienda	Sensor de cliente aproximándose	Salida del solenoide
Apagado	Apagado	Apagado
Apagado	Encendido	Apagado
Encendido	Apagado	Apagado
Encendido	Encendido	Encendido

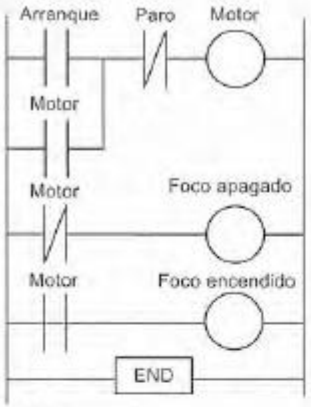
La tabla de verdad anterior corresponde a la de una compuerta AND, por lo que el programa para controlar la puerta es el que se muestra en la figura 19.15.

19.4.2 Enclavamiento

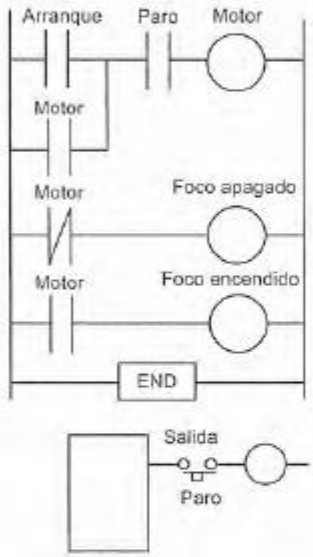
Con frecuencia se presentan situaciones en las que es necesario mantener energizada una bobina, aun cuando ya no exista la entrada que proporciona la energía. Para lograrlo se utiliza lo que se conoce como *circuito de enclavamiento*. Éste es un circuito de autosostenimiento, ya que después de ser energizado mantiene ese estado hasta que recibe otra entrada. Es decir, recuerda su último estado.

Figura 19.16 Circuito de enclavamiento

La figura 19.16 ilustra un circuito de enclavamiento. Cuando la entrada 1 se energiza y se cierra, se produce una salida. Sin embargo, cuando hay una salida, el contacto asociado a ella se energiza y se cierra. Estos contactos aplican el operador OR a los contactos de la entrada 1. Entonces, aun cuando el contacto de la entrada 1 se abra,



a) Sistema inseguro



b) Sistema seguro

Figura 19.17 Retención de un motor

el circuito mantendrá energizada la salida. La única manera de liberar la salida es accionando el contacto de la entrada 2 el cual en general está cerrado.

Para ejemplificar cómo se usa un circuito de enclavamiento, suponga que se requiere controlar un motor mediante un PLC de manera que al oprimir por un momento el botón de arranque, el motor comienza a trabajar; cuando se acciona el interruptor de paro, el motor se apaga; usando focos indicadores se sabe si el motor está encendido o apagado. La figura 19.17 muestra el diagrama equivalente. Cuando no hay entradas, el foco del motor indica que está apagado. Los contactos del motor, normalmente cerrados, encienden el foco de apagado. Al oprimir el botón de arranque, el contacto, normalmente abierto, se cierra y el motor se enciende. El contactos del motor, que está en paralelo con el contacto de arranque, retiene (enclava) la condición anterior. Además, el otro contacto del motor, normalmente cerrado, se abre, y se apaga el foco de apagado; el contacto del motor, normalmente abierto, se cierra y se activa el foco de encendido. El interruptor de paro abre el contacto y para el motor. Pudiera parecer que un arreglo de un interruptor de paro sencillo sería usar un conjunto de interruptores programables normalmente cerrados y abrirlos para detener el motor. Sin embargo, esto tiene la desventaja de que si existe una falla en el PLC entonces el motor no se puede detener. Utilizar un interruptor real normalmente cerrado en la salida del PLC, y luego programar el PLC como si los interruptores estuvieran normalmente abiertos para conmutarlos porque encuentra los contactos cerrados, es una opción más segura.

19.4.3 Secuenciación

Con frecuencia se presentan dos situaciones de control que requieren secuencias de salidas, con la conmutación de una a otra salida controlada por sensores. Suponga que se requiere un programa de escalera para un sistema neumático (figura 19.18) en el cual se controlan dos cilindros biestables, A y B, mediante válvulas de doble solenoide; en este caso se usan los sensores de inicio y fin de carrera a−, a+, b− y b+ para detectar los límites del movimiento del vástago de los pistones; se requiere una secuencia de activación de los cilindros correspondiente a A+, B+, A− y B−.

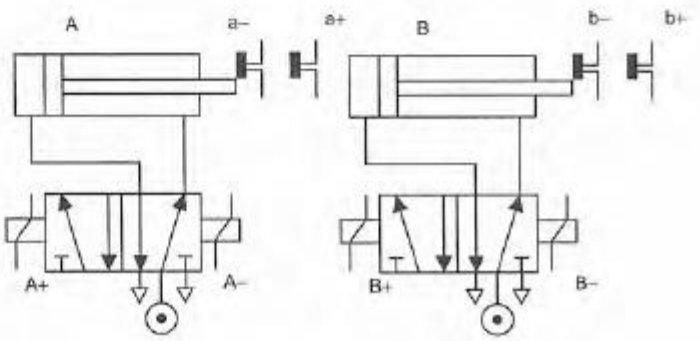


Figura 19.18 Puesta en secuencia de un pistón

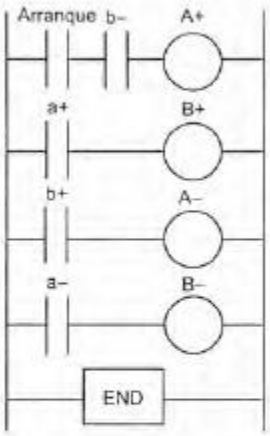


Figura 19.19 Puesta en secuencia de un pistón

La figura 19.19 muestra una opción para configurar el programa anterior. En el primer escalón se induce la entrada del interruptor de arranque A. La extensión del cilindro de A, es decir, cuando se energiza la solenoide A+, tiene lugar sólo cuando el interruptor de inicio está cerrado y también cuando el interruptor b− está cerrado; este último indica que el cilindro B está retraído. Cuando se extiende el cilindro A, el interruptor a+, que indica la extensión de A, se activa. Esto produce una salida que se envía al solenoide B+ y como resultado B se extiende. Esto cierra el interruptor, lo cual indica la extensión de B, es decir, del interruptor b+, y lleva a la salida a la solenoide A– y la retracción del cilindro A. Esta retracción cierra el interruptor límite a– de modo que da la salida a la solenoide B−, lo cual produce la contracción de B. Esto concluye el ciclo del programa y se regresa al primer escalón; el programa queda en espera de que se cierre el interruptor de inicio antes de que se repita el ciclo.

19.5 Mnemónicos

Cada uno de los escalones de un programa en escalera representa una línea del programa; la escalera constituye el programa completo en 'lenguaje de escalera'. Para introducir el programa en el PLC, el programador emplea un teclado con los símbolos gráficos de los elementos de escalera o selecciona los símbolos en una pantalla de computadora con un ratón; el panel o computadora que contiene el programa traduce los símbolos a lenguaje de máquina que se guardan en la memoria del PLC.

Otra manera de introducir un programa es traducir el programa escalera en instrucciones llamadas mnemónicos, donde cada línea de código corresponde a un elemento de la escalera; después éstos se introducen en el panel de programación o en la computadora y se traducen a lenguaje de máquina. Los mnemónicos difieren de un fabricante a otro. Para los PLC de la serie F de Mitsubishi, los mnemónicos se muestran al margen. En los ejemplos del resto de este capítulo, donde no se usan las descripciones generales, se utilizarán los mnemónicos de Mitsubishi. Los mnemónicos de otros fabricantes no difieren mucho y los principios en que se basa su aplicación son los mismos.

Mnemónicos Mitsubishi

LD	Iniciar un escalón con un contacto en general abierto
OUT	Una salida
AND	Un elemento en serie y, por lo tanto, una instrucción lógica AND
OR	Elementos en paralelo y, por lo tanto, una instrucción lógica OR
I	Una instrucción lógica NOT
...I	Se emplea junto con otras instrucciones para indicar lo inverso de éstas
ORI	Una función lógica OR NOT
ANI	Una función lógica AND NOT
LDI	Inicia un escalón con un contacto en general cerrado
ANB	AND utilizado con dos subcircuitos
ORB	OR utilizado con dos subcircuitos
RST	Restablecimiento de registro de corrimiento/contador
SHF	Corrimiento
K	Insertar una constante
END	Fin de la escalera

Los siguientes casos muestran cómo introducir escalones individuales en una escalera. Con base en los mnemónicos de Mitsubishi, la compuerta AND mostrada en la figura 19.20 se introduciría como:

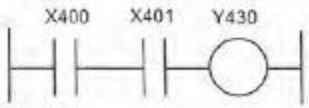


Figura 19.20 Un sistema AND

Paso	Instrucción	
0	LD	X400
1	AND	X401
2	OUT	Y430

La compuerta OR de la figura 19.21 se introduciría como:

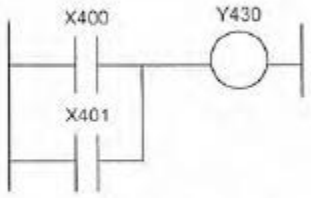


Figura 19.21 Un sistema OR

Paso	Instrucción	
0	LD	X400
1	OR	X401
2	OUT	Y430

La compuerta NOR de la figura 19.22 se introduciría como:

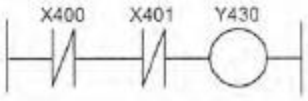


Figura 19.22 Un sistema NOR

Paso	Instrucción	
0	LDI	X400
1	ANI	X401
2	OUT	Y430

La compuerta NAND de la figura 19.23 se introduciría como:

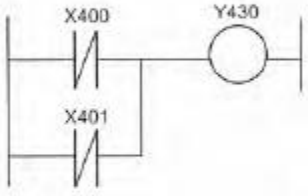


Figura 19.23 Un sistema NAND

Paso	Instrucción	
0	LDI	X400
1	ORI	X401
2	OUT	Y430

La compuerta XOR de la figura 19.24 se introduciría como:

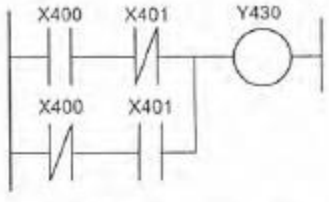


Figura 19.24 Un sistema XOR

Paso	Instrucción	
0	LD	X400
1	ANI	X401
2	LDI	X400
3	AND	X401
4	ORB	
5	OUT	Y430

Después de leer las dos primeras instrucciones, la tercera instrucción inicia una nueva línea. Pero la primera línea no ha finalizado con una salida. En consecuencia, la CPU reconoce que en la segunda línea hay una línea paralela y lee todos los elementos listados, hasta que llega a la instrucción ORB. El mnemónico ORB (unión en paralelo de dos ramas o bloques) indica a la CPU que debe aplicar un operador OR (O) a los resultados de los pasos 0 y 1, junto con los de los pasos 2 y 3.

19.6 Temporizadores, relevadores internos y contadores

En secciones anteriores de este capítulo se mencionaron tareas que requieren conexiones en serie y en paralelo de los contactos de entrada. Sin embargo, existen tareas en las que se requieren retardos y conteo de eventos. Para estos casos, pueden emplearse algunos dispositivos de los PLC como temporizadores y contadores, los cuales se controlan mediante instrucciones lógicas y se pueden representar en diagramas de escalera.