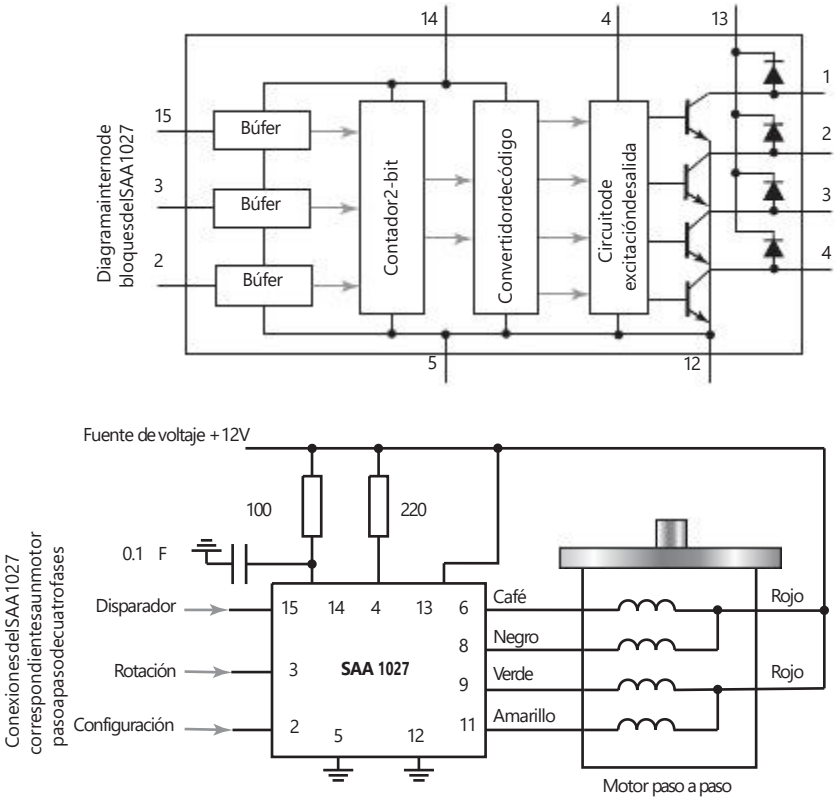


Figura 9.30 Circuito integrado SAA 1027 utilizado en un motor paso a paso.

en un valor bajo se produce una rotación en el sentido de las manecillas del reloj; cuando se mantiene a un valor alto, la rotación se da en sentido inverso.

Algunas aplicaciones requieren ángulos de paso muy pequeños. Si bien para reducir el tamaño del ángulo de paso se aumenta la cantidad de dientes del rotor y/o la cantidad de fases, es común que no usen más de cuatro fases ni más de 50 a 100 dientes. En su lugar se utiliza una técnica de **minipasos**, que consiste en dividir cada paso en cierta cantidad de subpasos de igual tamaño. Para ello se utilizan diferentes corrientes en los devanados, de manera que el rotor se desplace a posiciones intermedias entre las posiciones de un paso normal. Por ejemplo, es posible subdividir un paso de 1.8° en diez subpasos iguales.

Los motores paso a paso se usan para producir pasos de rotación controlados, así como una rotación continua, controlando su velocidad de rotación con el control de la frecuencia de aplicación de los pulsos que provocan el avance paso a paso. De esta manera se obtiene un motor de velocidad variable controlado muy útil que tiene muchas aplicaciones.

Dado que las bobinas del motor paso a paso tienen inductancia y que la aplicación de las cargas inductivas conmutadas pueden generar fuerzas contraelectromotrices considerables, al conectar los motores paso a paso a los puertos de salida de un microprocesador es necesario incluir una protección para evitar daños al microprocesador. Esta protección se logra conectando resistores a las líneas para limitar la corriente; el valor de estos resistores debe elegirse con mucho cuidado para obtener esa protección, pero sin limitar el valor de la corriente necesaria para conmutar los transistores. Los diodos conectados en los devanados impiden que haya corriente en dirección inversa, por lo que también brindan protección. Otra alternativa son los optoaisladores (vea la sección 3.3).

9.7.3 Selección de un motor paso a paso

1 Requerimientos del par de operación de la aplicación. El par de velocidad debe ser lo bastante alto para acomodar el par y el requerimiento de velocidad de progresión. También las características par velocidad deben ser las adecuadas.
2 El ángulo de paso deber ser de una resolución lo bastante alta para proporcionar los incrementos de movimiento de salida requeridos.
3 Costo.

Todo esto necesitará contemplar las especificaciones de datos para motores paso a paso. A continuación se presentan algunos valores comunes tomados de una hoja de datos de un productor para un motor paso a paso unipolar (Canon 42M048CIU-N):

Voltaje de operación c.d.	5V
Resistencia por devanado	9.1 æ
Inductancia por devanado	8.1 mH
Par de retención	66.2 mNm/9.4 pulg/onza
Momento de inercia del rotor	12.5 - 10^{-4}gm2
Par de retén	12.7 mNm/1.8 pulg/onza
Ángulo de paso	7.5º
Tolerancia de ángulo de paso	± 0.5º
Pasos por revolución	48

El par de detención es el requerido para el motor paso a paso cuando no están energizados los devanados del motor.

Una vez seleccionado el motor, se necesitará encontrar un sistema motriz que sea compatible con el motor. Por ejemplo, para utilizar el Cybernetics CY512 con un motor unipolar, se podría si fuera aceptable un voltaje de entrada máximo de 7 V y una corriente máxima por fase de 80 mA. El SAA1027 de Sgnetis es un controlador ampliamente utilizado en motores paso a paso unipolares pequeños con un voltaje de entrada máximo de 18 V y una corriente máxima por fase de 350 mA. Para un motor bipolar de dos fases un motor unipolar de cuatro fases, habría que considerar el SCS-Thomson L297/L298, ya que es un conjunto de controlador lógico de dos chips. El chip L297 genera las secuencias de fase del motor de cuatro fases TTL de señales lógicas para motores unipolares de dos y de cuatro fases, y el L298 es un controlador puente diseñado para aceptar este tipo de señales y cargas inductivas de control, en este caso un motor paso a paso. Un motor bipolar puede ser controlado por corrientes de devanado de hasta 2 A.

Si se proporciona un pulso a un motor paso a paso tenemos esencialmente una entrada a un circuito de resistor e inductor y el par resultante se aplica a la carga, de donde resulta una aceleración angular. En consecuencia, el sistema tendrá una frecuencia natural; no irá directamente a la posición del paso siguiente pero por lo general tendrá oscilaciones amortiguadas en su entorno antes de descender hasta el valor permanente (Figura 9.31). Vea la sección 24.1.2 para un comentario de esto y una derivación de la frecuencia natural y del factor de amortiguamiento.

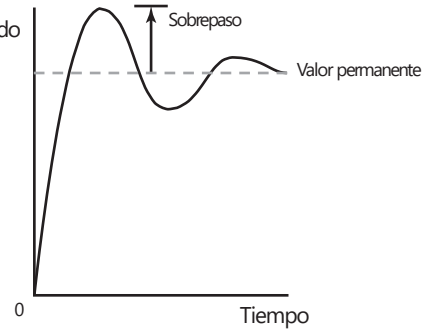


Figura 9.31 Oscilaciones en tono al ángulo permanente.

9.8 Selección de un motor

Al seleccionar un motor para una aplicación en particular, entre los factores que hay que tener en cuenta están:

1 Acoplamiento de inercia
2 Requerimientos de par
3 Requerimientos de potencia

9.8.1 Acoplamiento de inercia

El concepto de acoplamiento de impedancia que se presenta en la sección 3.8 para impedancias eléctricas se puede ampliar a sistemas mecánicos, y a una situación análoga a lo que ahí se describió para los circuitos eléctricos referente a un motor, una fuente de par, que rotan directamente una carga (Figura 9.32a). El par requerido para que dé una carga con un momento I_L de inercia y una aceleración angular a es $I_L a$. El esfuerzo de torsión requerido para acelerar el eje del motor es $T_M = I_M a_M$ y el que se requiere para acelerar la carga es $T_L = I_L a_L$. El eje del motor tendrá, en ausencia del engrane, la misma aceleración angular y la misma velocidad angular. La potencia que se necesita para acelerar todo el sistema es $T_M V$, donde V es la velocidad angular. Entonces:

$$\text{potencia} = (I_M + I_L) a V$$

La potencia se produce por el par del motor T_M y así debe ser igual a $T_M V$. Por lo tanto,

$$T = (I_M + I_L) a$$

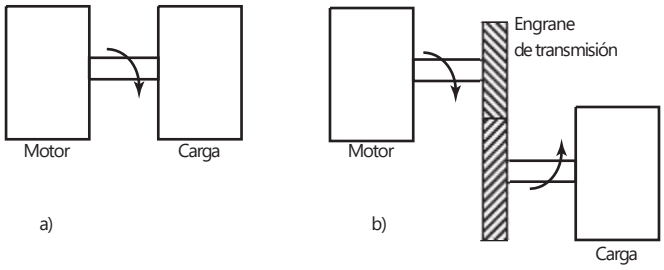


Figura 9.32 a) Motor girando una carga, b) motor con engrane de transmisión girando una carga.

El par para obtener una aceleración angular dada se minimizará cuando $I_M = I_L$. Por lo tanto para una transferencia de potencia máxima, el momento de inercia de la carga debe ser igual al del motor.

En el caso del motor que gira la carga mediante un engrane de transmisión (Figura 9.32b), la condición para la transferencia de máxima potencia es que el momento de inercia del motor sea igual al momento de inercia reflejado de la carga, que es $n^2 I_L$, donde n es el coeficiente de reducción e I_L el momento de inercia de la carga (vea la sección 10.2.2).

Por lo tanto, para la transferencia de máxima potencia, el momento de inercia del motor debe acoplarse con el de la carga o la carga reflejada cuando se emplean engranes. Esto implicará que el par para obtener una aceleración dada se minimizará. Esto es particularmente útil si el motor se va a utilizar para un posicionamiento rápido. Con un sistema de engranes se puede utilizar el ajuste del coeficiente de reducción para que se logre un acoplamiento.

9.8.2 Requerimientos de par

La Figura 9.33 muestra las curvas de funcionamiento de un motor común. Para un giro continuo no se debe exceder el valor de par de paro. Éste es el valor de par máximo al cual no ocurrirá un calentamiento. En uso intermitente también son posibles pares mayores. Al incrementarse la velocidad angular, disminuye la capacidad del motor para que entregue par. Por lo tanto, si se requieren velocidades y pares mayores que puedan ser proporcionados por un motor en particular, se necesita elegir un motor más potente.

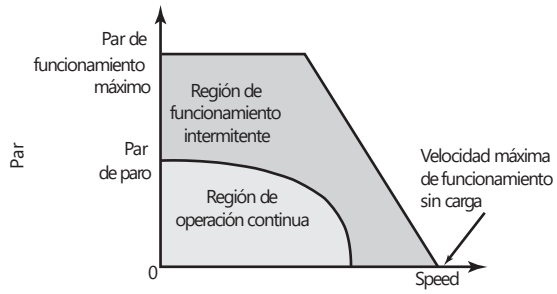


Figura 9.33 Gráfica de par velocidad.

Supongamos que se requiere un motor para operar un montacargas de tipo tambor y que levante una carga (Figura 9.34). Con un diámetro de tambor de, digamos, 0.5 m y una carga máxima m de 1000 kilos, la tensión en el cable será $mg = 1000 \cdot 9.81 = 9810$ N. El par en el tambor será $9810 \cdot 0.25 = 24\,525$ Nm, o alrededor de 2.5 kNm. Si el montacargas funciona a una velocidad constante v de 0.5 m/s, la velocidad angular del tambor V es $v/r = 0.5/0.25 = 2$ rad/s, o $2/2p = 0.32$ revs/s. El motor controla el eje mediante un engrane. Hay que decidir si el coeficiente de reducción debe ser tal que la velocidad máxima del motor debiera estar en alrededor de 1500 rev/min, o 25 rev/s. Esto indica un coeficiente de reducción n de 25/0.32 o casi lo bastante para 80:1. El par de carga en el motor será reducido por un factor de 80 a partir del del tambor y será de 2500/80 = 31.25 Nm. Si se permite cierta fricción en el engrane, entonces el par máximo permisible en el motor puede ser de alrededor de 35 Nm.

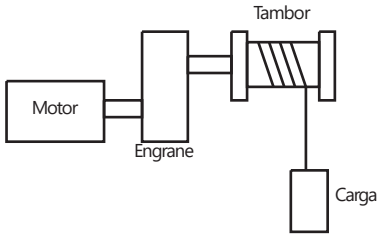


Figura 9.34 Motor levantando una carga.

No obstante, éste es el único par máximo cuando la carga se debe elevar a una velocidad constante. Se necesita agregarle el par que se requiere para acelerar la carga desde la base a la velocidad de 0.5 m/s. Si, digamos, lo que se requiere es alcanzar esta velocidad desde la base en 1 s, entonces el par de aceleración que se necesita es $I\alpha$, donde I es el momento de inercia y a es la aceleración angular. El momento de inercia efectivo de la carga como se ve por el motor a través del engrane es $(1/n^2)$ - el momento de inercia de la carga mr^2 y así $(1/80)^2$ - 1000 - $0.252 = 0.0098$ kg m^2 o alrededor de 0.01 kg m^2. El momento de inercia referido del tambor y del engrane puede agregar 0.02 kg m^2. Para encontrar el momento de inercia total implicado en el izamiento de la carga también hay que agregarle el momento de inercia del motor. Las hojas de datos del fabricante deben dar un valor de, digamos, 0.02 kg m^2 y entonces el momento de inercia total implicado en el izamiento debe ser $0.01 + 0.02 + 0.02 = 0.05$ kg m^2. Se requiere la velocidad del motor para levantar de 0 a 25 rev/s en 1 s, por lo que la aceleración angular es 25 - 2p)/1 = 157 rad/s^2 o alrededor de 160 rad/s^2. Entonces, el par de aceleración requerido es 0.05 - 160 = 8 Nm. En consecuencia, el par máximo que se debe permitir es el que se requiere para levantar la carga a una velocidad constante más la que se necesita para acelerarla a esta velocidad desde la base, y por lo tanto es $35 + 8 = 43$ Nm.

Se pueden escribir en modo algebraico los argumentos implicados en el ejemplo anterior como sigue. El par T_m requerido de un motor es el que se necesita por la carga T_L, o T_L/n para una carga ajustada a la relación del engrane n, y la que se necesita para acelerar el motor $I_m\alpha_m$, donde I_m es el momento de inercia del motor y α_m su ángulo de aceleración:

$$T = \frac{T_L}{n} + I_m\alpha_m$$

La aceleración angular de la carga α_L está dada por

$$\alpha_m = n\alpha_L$$

Como habrá un par T_f requerido para superar la fricción de la carga, el par utilizado para la carga será $(T_L \quad T_f)$ y por lo tanto

$$T_L \quad T_f = I_L\alpha_L$$

Por lo tanto podemos escribir

$$T_m = \frac{1}{n}\left[T_L + \alpha_L\left(I_L + n^2 I_m\right)\right]$$

9.8.3 Requerimientos de potencia

El motor debe estar apto para correr a la máxima velocidad requerida sin calentamiento excesivo. La potencia P total es la suma de la potencia requerida para superar la fricción y la que se necesita para la carga. Como la potencia es

el producto del par y de la velocidad angular, la potencia requerida para superar el par friccional T_f es $T_f \omega$ y la que se necesita para acelerar la carga con la aceleración angular a es $(I_L a)\omega$, donde I_L es el momento de inercia de la carga. Por lo tanto:

$$P = T_f \omega + I_L a \omega$$

Resumen

Los **relevadores** son interruptores operados de forma eléctrica en el que si se cambia una corriente a un circuito eléctrico se cambia a una corriente de encendido o apagado a otro circuito.

A un **diodo** puede considerársele como un dispositivo que pasa corriente en una sola dirección, la otra dirección tiene una resistencia muy alta.

Un **tiristor** puede ser considerado como un diodo que tiene una puerta que controla las condiciones bajo las cuales el diodo puede encenderse. Un **triac** es similar a un **tiristor** y es equivalente a un par de tiristores conectados en antiparalelo en el mismo chip.

Los **transistores bipolares** se pueden usar como interruptores al cambiar la actual base entre cero y un valor que conduce el transistor en la saturación. Los **MOSFET** son similares y también se pueden usar como interruptores.

El principio básico de un motor de c.d. es un circuito de alambre, la armadura, la cual gira libre en el campo de un imán como resultado de una corriente que pasa a través de un circuito. El campo magnético puede provenir de un imán permanente o un electroimán, por ejemplo un devanado de campo. La velocidad de un motor magnético permanente depende de la corriente a través del devanado de la armadura; con un motor de devanado de campo éste depende ya sea de la corriente a través del devanado de la armadura o a través del devanado de campo. Estos motores de c.d. requieren un conmutador y escobillas para invertir de manera periódica la corriente a través de cada devanado de armadura. El **motor de c.d. de imán permanente sin escobillas** tiene un rotor de imán permanente y una secuencia de bobinas de estator a través de las cuales la corriente cambia su secuencia.

Los **motores de c.a.** se pueden clasificar en dos grupos, monofásico y polifásico, cada grupo subdividido en motores de inducción y motores síncronos. Los motores de una sola fase tienden a utilizarse para requerimientos de potencia baja, mientras que los motores polifásicos se utilizan para potencias más altas. Los motores de inducción tienden a ser más baratos que los motores síncronos y, por lo tanto, tienen un uso mucho más amplio.

La selección de un motor requiere tener en cuenta el **acoplamiento de inercia**, así como el **par** y los **requerimientos de potencia**.

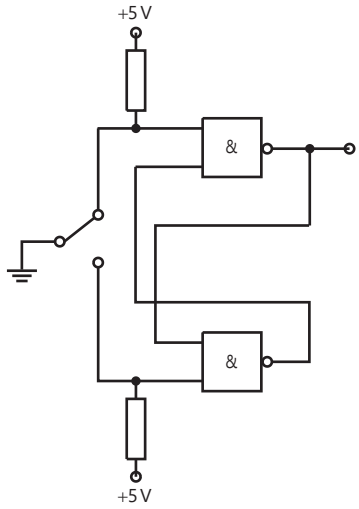
Problemas

9.1 Explique cómo usar el circuito de la Figura 9.35 para eliminar el rebote del interruptor.

9.2 Explique cómo usar un tiristor para controlar el nivel de un voltaje de c.d. seccionando la salida de una fuente de voltaje constante.

9.3 Se necesita un motor de c.d. con el que se obtenga: a) un par alto a velocidades bajas, para desplazar cargas grandes; b) un par de valor casi constante

Figura 9.35 Problema 9.1.



independientemente de la velocidad. Sugiera tipos de motor que sean adecuados para este propósito.

9.4 Sugiera posibles tipos de motores, ya sea de c.d. o de c.a., para aplicaciones en las que se obtenga: a) una operación barata y con par constante, b) velocidades altas controladas, c) velocidades bajas, d) reducir al mínimo las necesidades de mantenimiento.

9.5 Explique el principio de un motor de c.d. de imán permanente sin escobillas.

9.6 Explique los principios de la operación del motor paso a paso de reluctancia variable.

9.7 Si el ángulo de paso de un motor paso a paso es 7.5°, ¿cuál será la frecuencia de la entrada digital para obtener una rotación de 10 rev/s?

9.8 ¿Cuál será el ángulo de paso para un motor paso a paso híbrido con ocho devanados del estator y diez dientes de rotor?

9.9 Un motor de c.d. de imán permanente tiene una resistencia de armadura de 0.5 & y cuando un voltaje de 120 V es aplicado al motor éste alcanza una velocidad de rotación de estado de equilibrio de 20 rev/s y señala 40 A. ¿Cuál será a) la entrada de potencia para el motor, b) la pérdida de potencia en la armadura, c) el par generado a esa velocidad?

9.10 Si un motor de c.d. produce un par de 2.6 N m cuando la corriente de la armadura es 2 A, ¿cuál será el par con una corriente de 0.5 A?

9.11 ¿Cuántos pasos/pulsos por segundo de salida necesitará un microprocesador para un motor paso a paso si a éste se le da una salida de 0.25 rev/s y tiene un ángulo de paso de 7.5º?

9.12 Un motor paso a paso se utiliza para girar una polea de 240 mm de diámetro y, por lo tanto, una banda que está moviendo una masa de 200 kg. Si esta masa se acelera de manera uniforme desde el reposo hasta 100 mm/s en 2 segundos y hay una fuerza de fricción constante de 20 N, ¿cuál será el par requerido de empuje para el motor?

Parte IV
Sistemas de microprocesadores

Capítulo diez Microprocesadores y microcontroladores

Objetivos

Después de estudiar este capítulo, el lector debe ser capaz de:

- **Describir la estructura básica de un sistema de microprocesador.**
- **Describir la arquitectura de microprocesadores y la manera en la que se pueden incorporar a sistemas microprocesadores.**
- **Describir la estructura básica de microcontroladores y la manera en la que sus registros pueden configurarse para llevar a cabo tareas.**
- **Explicar la manera en la que los programas se pueden desarrollar con el uso de diagramas de flujo y seudocódigo.**

10.1 Control

Si se considera un problema de control sencillo, como la secuencia de las luces roja, ámbar 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 control 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 el ser humano no es aparente que el sistema sea de microprocesador. Así, una moderna lavadora de ropa contiene un microprocesador y todo lo que el operador debe hacer para que funcione es seleccionar qué tipo de lavado requiere al oprimir los botones apropiados o girar un selector y luego oprimir 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 13 las interfaces.

10.2 Sistemas de microprocesadores

Los sistemas de microprocesadores constan de tres partes: la **unidad central de procesamiento** (CPU), la cual reconoce y ejecuta las instrucciones de un programa. Ésta es la parte que usa el microprocesador, las **interfaces de entrada y salida**, para manejar las comunicaciones entre la computadora y el mundo exterior; el término **puerto** se usa para la interfaz, y la **memoria**

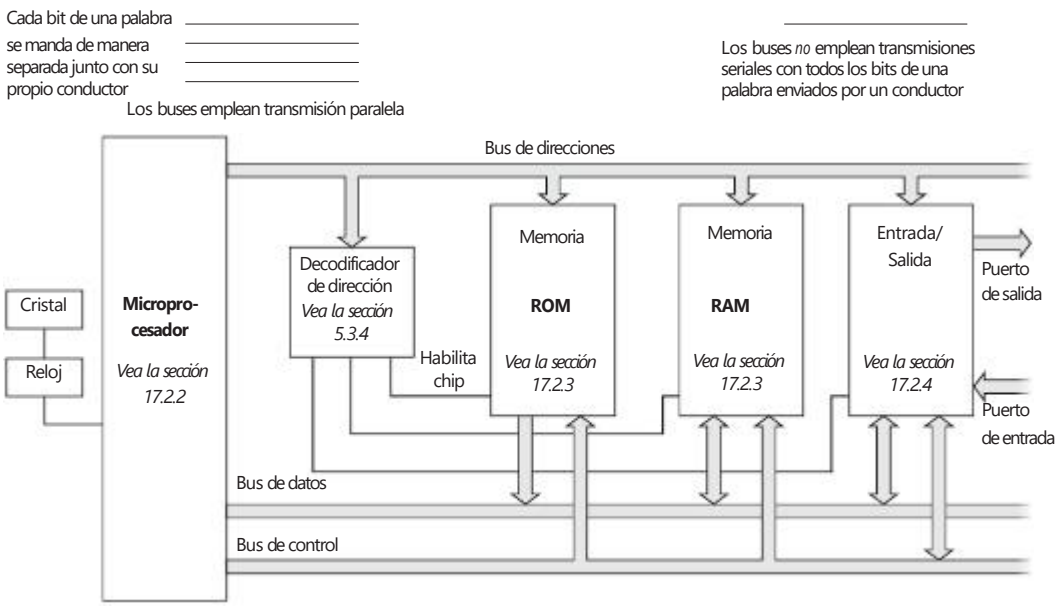


Figura 10.1 Forma general de un sistema microprocesador y sus buses. Todos los componentes comparten el mismo bus de datos y de direcciones. A este arreglo se le conoce como la arquitectura de Von Neumann.

es donde se almacenan instrucciones de programas y datos. La Figura 10.1 ilustra un arreglo general de un sistema microprocesador.

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

10.2.1 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 y son vías que pueden compartir todos los chips en el sistema. Esto se debe a que si sus conexiones separadas se utilizaran entre los chips, habría una cantidad muy grande de conductores de conexiones. Cuando se utilizan buses de conexiones compartidas significa que el chip pone los datos en el bus, el otro chip tiene que esperar su turno hasta que termine la transferencia de datos antes de que uno de ellos pueda poner sus datos en el bus. Por lo general, un bus tiene 16 o 32 conexiones paralelas de manera que cada una pueda llevar 1 bit de una palabra de datos simultáneamente. Esto agiliza la transmisión que al tener una conexión en serie envía una palabra completa en una secuencia de bits por un conductor.

Hay tres formas de bus en 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 interfaces de entrada/salida. 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)	Primera línea del bus de datos
1	Segunda línea del bus de datos
0	Tercera 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 $2^4 = 16$. Suponga 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 interfaz 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 un cristal y produce pulsos de periodos regulares.

10.2.2 El microprocesador

En general se hace referencia al microprocesador como la unidad de procesamiento central (CPU). Ésta es la parte del procesador en la que se procesan

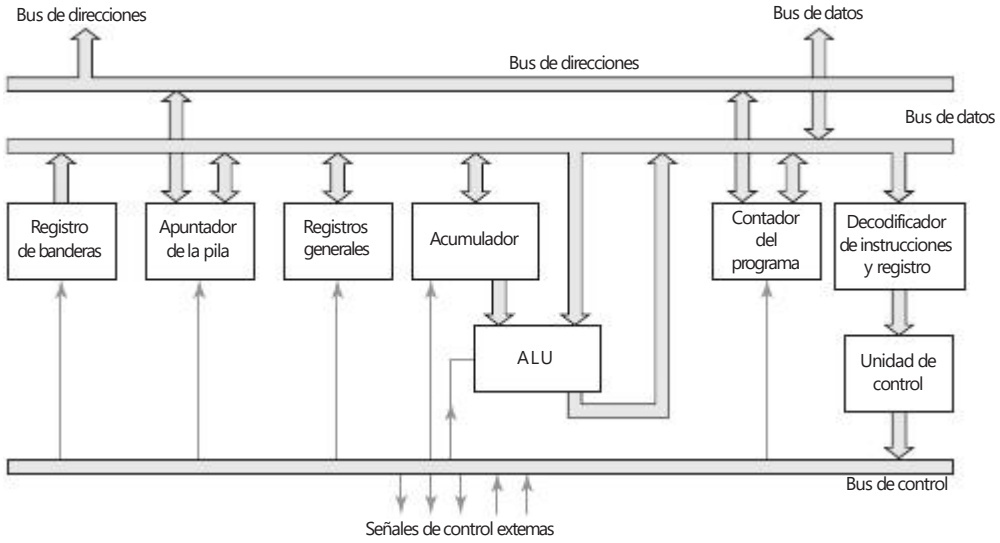


Figura 10.2 Arquitectura interna general de un microprocesador.

los datos, se traen instrucciones y datos. La estructura interna, conocida como **arquitectura** de un microprocesador, depende del microprocesador que se esté considerando. La Figura 10.2 indica, en forma simplificada, la arquitectura general de un microprocesador.

Las siguientes son las funciones de las partes que forman un microprocesador:

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

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. La 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 dos y doce 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, la dimensión y el tipo de los registros varía de un microprocesador a otro. Los siguientes son los registros más comunes:

1 *Registro acumulador*
El registro 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, por ejemplo, se combinan números: 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 desbordamiento, 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) o *apuntador de instrucciones* (IP)
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)
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)
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 **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)
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 10.3) 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, éste requería un generador de reloj externo mientras que el 8085A tiene un generador de reloj integrado. 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 17.4 muestra un diagrama de bloques representativo de la arquitectura.

Como será aparente a partir de las Figuras 10.3 y 10.4, 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.

10.2.3 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.

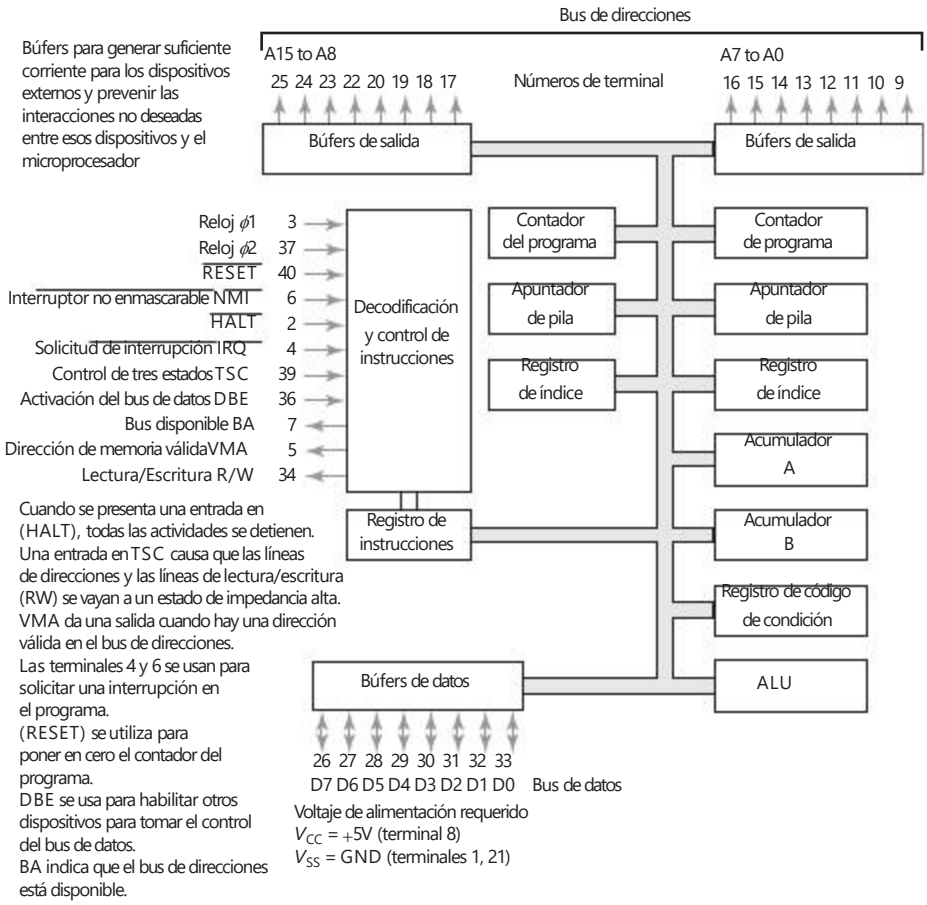


Figura 10.3 Arquitectura del microprocesador 6800 de Motorola.

El tamaño de la memoria depende de la cantidad de líneas del bus de direcciones. 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 acceder a 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 10.5).

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 dispositivo de memoria conocido como **memoria de sólo lectura** (ROM). 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. Aun cuando se suspenda

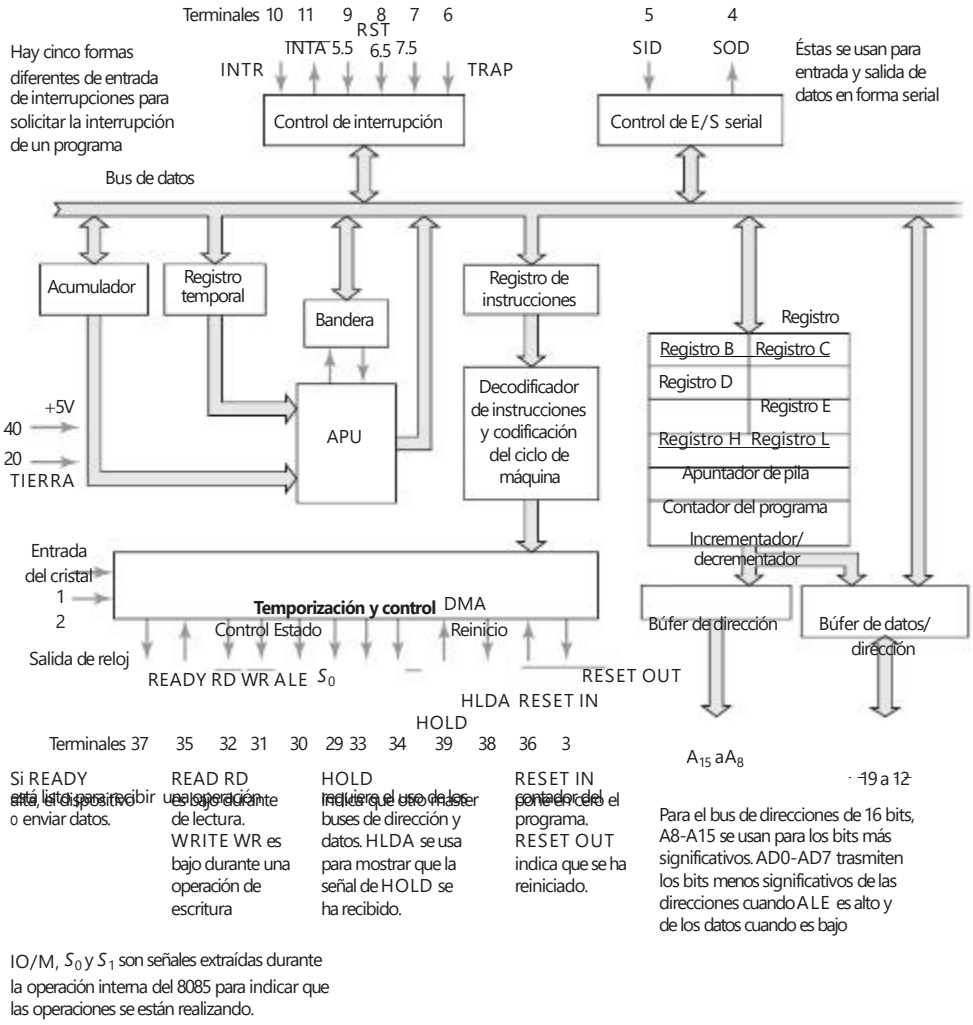


Figura 10.4 Arquitectura del microprocesador Intel 80885A.

Figura 10.5 Tamaño del bus de dirección.

la alimentación eléctrica, esta memoria no pierde su contenido. La Figura 10.6a) muestra las conexiones de un chip de 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) 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

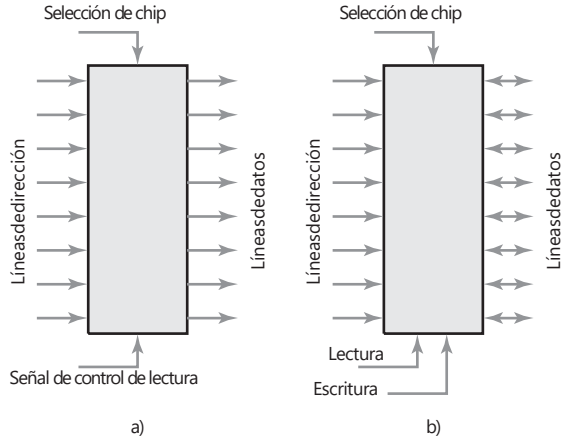


Figura 10.6 a) chip de ROM, b) chip de RAM.

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) 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); en ella se puede leer y escribir. La Figura 17.6b) muestra las conexiones típicas del chip de una RAM de 1 K * 8 bits. El chip de 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.

Cuando en una ROM se guarda un programa, estará disponible y listo cuando se active 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.

10.2.4 Entrada/salida

La operación 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 interfaz. 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 interfaz; 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 interfaz ha retenido correctamente los datos de entrada. Para ello realiza un **muestreo** o una **interrupción**. En el primer caso, el chip de interfaz 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 interfaz 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.

10.2.5 Ejemplos de sistemas

La Figura 10.7 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, respectivamente, interfaces de entrada y salida.

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 dirección a este registro donde se enclava. Entonces cuando la ALE cambia y regresa a baja, 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.

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. Antes de poder elegir, debe habilitarse con las entradas de habilitación 1 y 2 en bajo y la 3 en alto.

3 *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) se conecta a la salida de lectura del microprocesador para asegurar que la EPROM sea sólo de escritura.

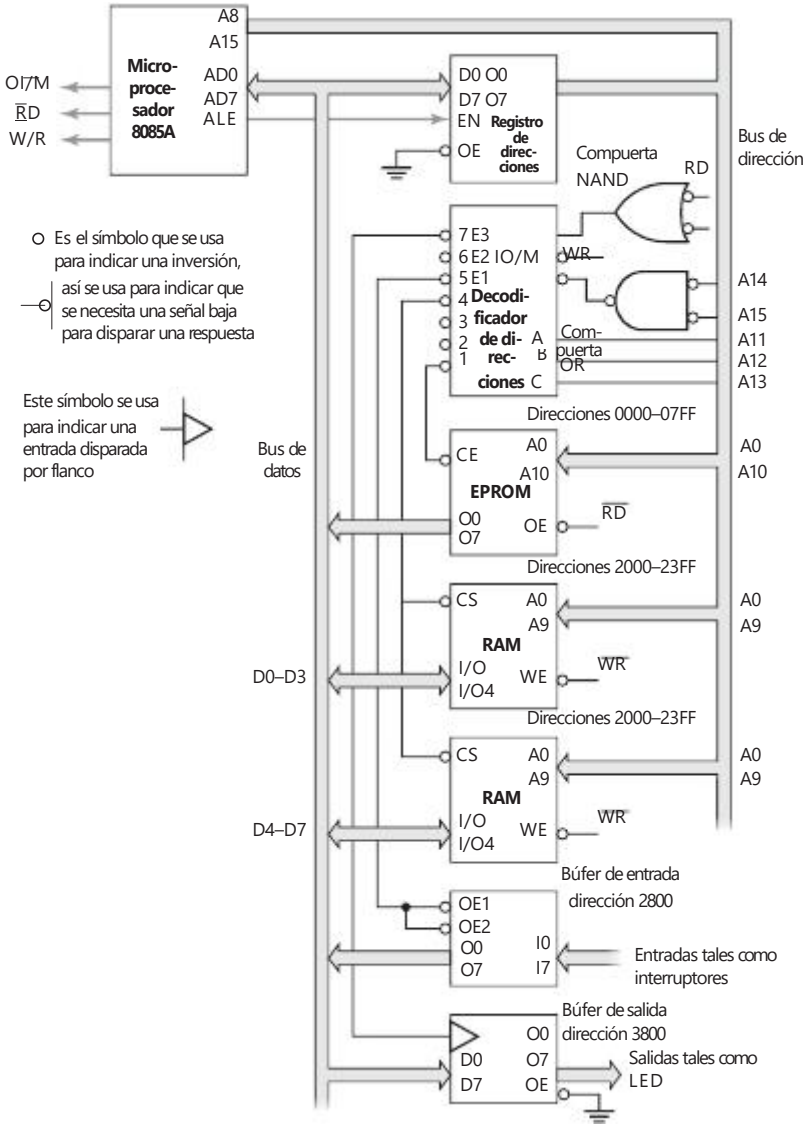


Figura 10.7 Sistema Intel 8085A.

4 *RAM*

Se muestran dos chips de memoria RAM según se utilizan, cada una de 1K * 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) 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 *Búfer de entrada*

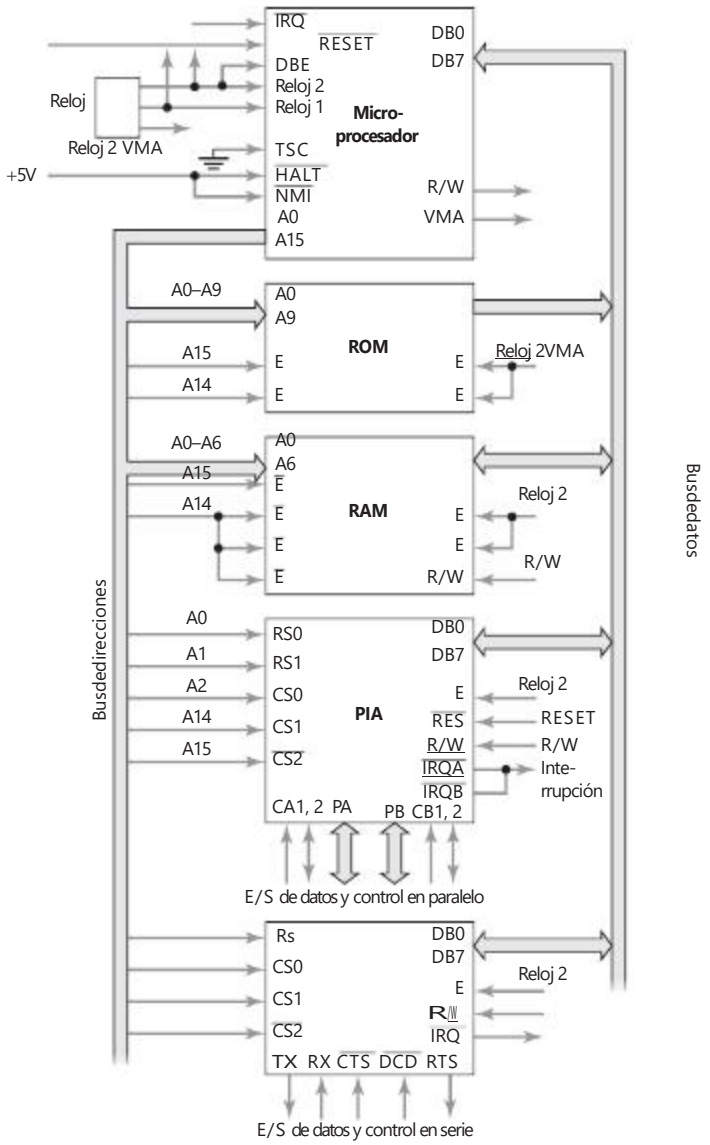
El búfer de entrada 74LS244 pasa el valor binario de las entradas sobre el bus de datos siempre que OE1 y OE2 estén en bajo. Se accede a éste mediante

cualquier dirección entre 2800 y 2FFF, así se podría utilizar 2800. El búfer 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 17.8 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 entradas/salidas no es necesario decodificar la dirección con

Figura 10.8 Sistema M6800.



este sistema, ya que en paralelo se usa un adaptador de interfaz periférico (PIA) (sección 13.4) y para entradas/salidas en serie se utiliza un adaptador de interfaz asíncrono (ACIA) (sección 13.5). Éstos se pueden programar para manejar las entradas y salidas y dar el aislamiento requerido.

1 *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 *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 interfaces 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.

10.3 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 integra en un chip de microprocesador con memoria, interfaces de entrada/salida y otros dispositivos periféricos como temporizadores. La Figura 10.9 muestra un diagrama de bloques general de un microcontrolador.

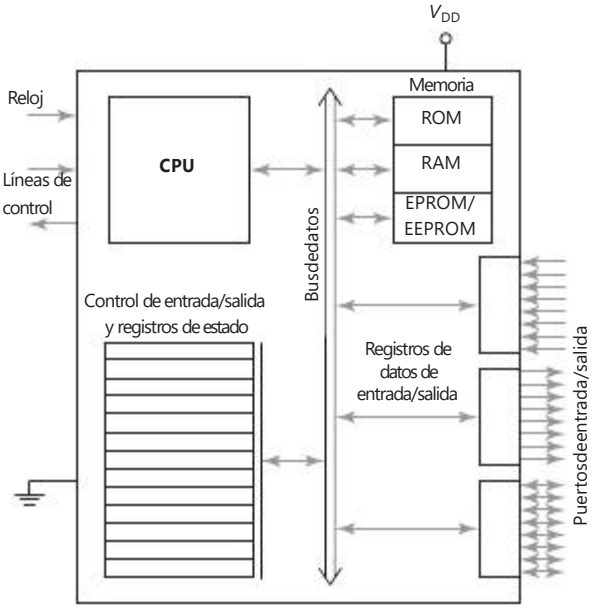


Figura 10.9 Diagrama de bloques de un microcontrolador.

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 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.

10.3.1 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 10.10) se basa en el microprocesador 6800 de Motorola, el cual es muy utilizado para sistemas de control. Observe que este microcontrolador fue introducido por Motorola y ahora lo produce Freescale Semiconductor.

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,

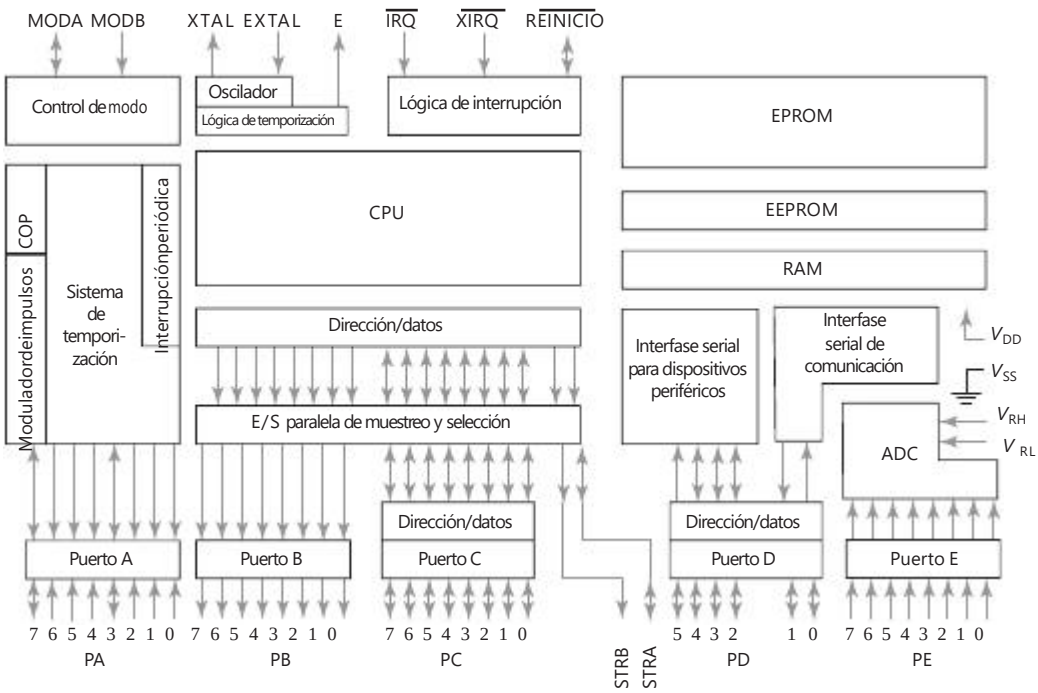


Figura 10.10 Diagrama de bloques del M68HC11.

512 bytes de EEPROM, 256 bytes de RAM, un sistema de temporización de 16 bits, una interfaz serial síncrona, una interfaz 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.

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 10.11), la dirección del registro de control del acumulador de pulsos es $1026 (Figura 10.12); 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 10.11 Registro del puerto A.

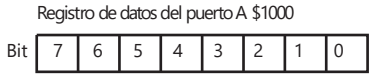
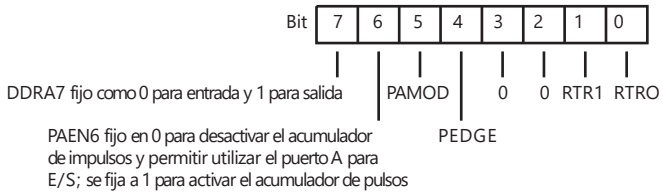


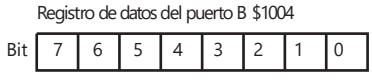
Figura 10.12 Registro del control del acumulador de pulsos.



2 *Puerto B*

El puerto B sólo funciona como salida y tiene ocho líneas (Figura 10.13). 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.

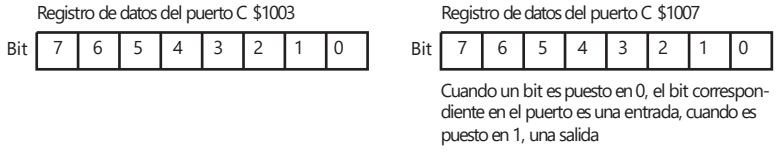
Figura 10.13 Registro del puerto B.



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 10.14). Su dirección se controla mediante el registro de direcciones de datos del puerto en la dirección $1007. Los 8 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 chip sencillo) 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 pa-

Figura 10.14 Registro del puerto C.

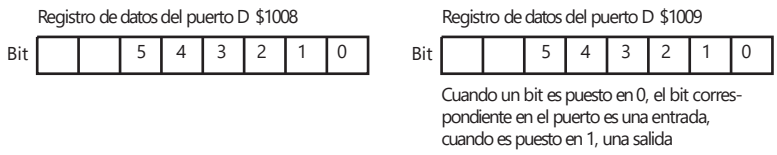


ralelo PIOC, en la dirección $1002 contiene bits para controlar el modo de *handshake*, así como la polaridad y los flancos activos de las señales de *handshake*.

4 *Puerto D*

El puerto D contiene sólo seis líneas, que pueden ser entrada o salida, y su registro de datos está en la dirección $1008 (Figura 10.15); 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 interfaz para comunicación serial es un sistema asíncrono que proporciona una comunicación serial compatible con modems y terminales. La interfaz periférica serial es un sistema síncrono de alta velocidad diseñado para comunicar el microcontrolador y los componentes periféricos compatibles con estas velocidades.

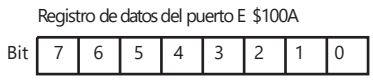
Figura 10.15 Registros del puerto D.



5 *Puerto E*

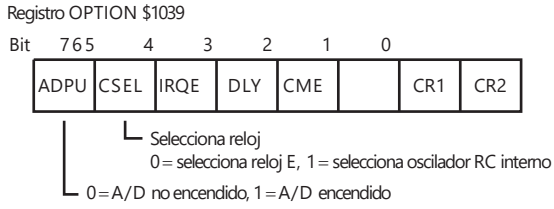
El puerto E es de 8 bits sólo de entrada (Figura 10.16) que se puede utilizar como puerto de entrada de propósito general, o para entradas al convertidor interno analógico-digital. Las dos entradas V_{RH} y V_{RL} proporcionan voltaje de referencia al ADC. El registro de datos del puerto E está en la dirección $1002.

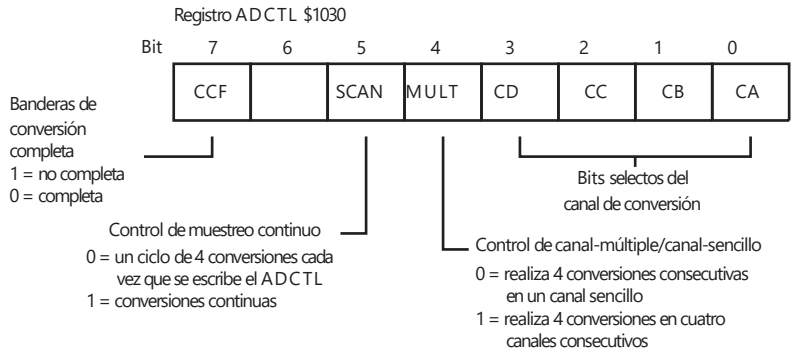
Figura 10.16 Registro del puerto E.



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 10.17), 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.

Figura 10.17 Registro OPTION.





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	-	-	PE0	PE1	PE2	PE3
0	1	-	-	PE4	PE5	PE6	PE7

Figura 10.18 Registro ADCTL.

La conversión analógica a digital se inicia escribiendo al registro ADCTL (registro A/D control/estado) después de encenderlo y del retardo de estabilización (Figura 10.18). 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 C.D.-C.A. El resultado de la conversión se guarda en los registros de resultados A/D ADR1-ADR4.

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

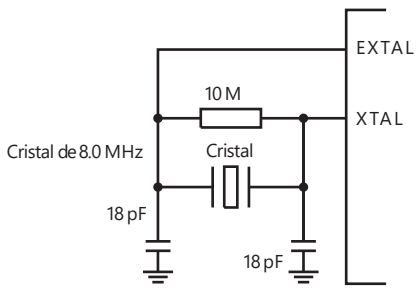
En el modo de un solo chip, el microcontrolador es por completo autosuficiente, excepto por una fuente de reloj externa y un circuito de reinicio. Con este modo, es posible que los recursos propios del microcontrolador no sean suficientes como la memoria; en estos casos se puede usar el modo ampliado 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 especial de prueba 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.

7 *Terminales del oscilador*

Las terminales del sistema oscilador XTAL y EXTAL son conexiones necesarias para accesar al oscilador interno. La Figura 10.19 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.

Figura 10.19 Salida del oscilador.



8 *Controlador de interrupción*

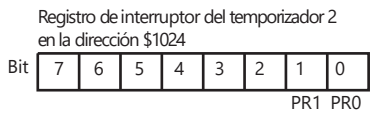
Este controlador permite al microcontrolador interrumpir un programa (sección 13.3.3). Una interrupción es un evento que requiere la CPU para detener la ejecución normal de un programa y para realizar algún servicio relacionado con el evento. Las líneas IRQ y XIRQ están asignadas a las entradas de señales de interrupción externas. RESET es para el restablecimiento del microcontrolador lo que permite un arranque del sistema de manera ordenada. El estado de la terminal se puede configurar ya sea externa o internamente. Cuando una condición de reinicio se detecta, la señal de la terminal se configura baja para cuatro ciclos de reloj. Si después de más de dos ciclos aún sigue baja, entonces se debe considerar que ocurra una configuración externa. Si en la potencia de entrada V_{DD} se detecta una transición positiva, ocurre un restablecimiento en la potencia encendida. Esto equivale a un tiempo de retraso del ciclo 4064. Si la terminal reiniciada está baja al final del tiempo de retraso en la potencia encendida, el microcontrolador permanece en la condición de descanso hasta que suba.

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 registro TMSK2 los bits PR0 y PR1 como bits 0 y 1 en la dirección $1024 (Figura 10.20).

Figura 10.20 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
1	0	8	4 ms	8 ms
1	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 10.12), 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). Consiste en un temporizador que apaga y restablece el sistema si no ha concluido alguna operación dentro de un lapso razonable (sección 23.2). También se le conoce como **temporizador vigilante**.

11 *PWM*

La modulación de ancho de pulso (PWM) controla la velocidad de los motores de c.d. (secciones 3.6 y 9.5.3) 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 microcontrolador, 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.

10.3.2 El 8051 de Intel

Otra familia común de microcontroladores es la 8051 de Intel. La Figura 10.21 muestra sus conexiones y 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.

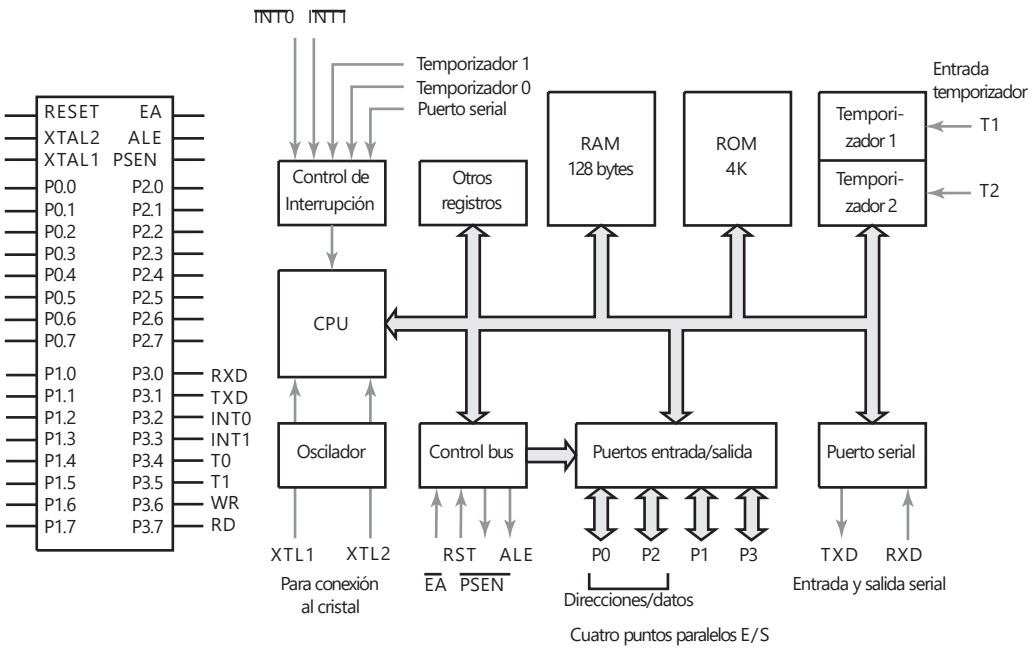


Figura 10.21 Intel 8051.

1 *Puertos de entrada/salida*

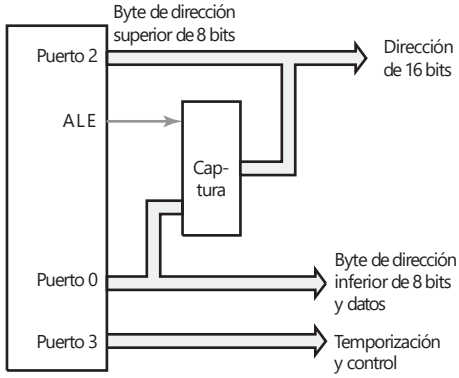
El puerto 0 está en la dirección 80H, el puerto 1 en la dirección 90H, el puerto 2 en la dirección A0H y el puerto 3 en la dirección B0H (Intel utiliza una H (o h) después de la dirección para indicar que es hexadecimal). Cuando un puerto se usa como puerto de salida, los datos se colocan en el registro 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, se puede utilizar el bit 6 del puerto 0 para encender o apagar un motor y quizás el bit 7 para encender o apagar una bomba.

El puerto 0 se utiliza como puerto de entrada o de salida. También se puede emplear para acceder a la 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 como puerto de entrada o de salida. También se puede emplear para acceder a la 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 interfaz 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. El puerto 0 se puede utilizar ya sea como puerto de entrada o como puerto de salida. De manera alternativa, se le puede aprovechar para tener acceso a la memoria externa.

2 *ALE*

La conexión para la habilitación del registro de direcciones (ALE) 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 10.22 ilustra esto.

Figura 10.22 Uso del ALE.



3 *PSEN*

La terminal para la activación del almacenamiento del programa (PSEN) 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) 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 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.

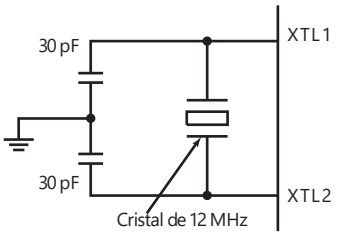


Figura 10.23 Cristal.

5 *XTAL1, XTAL2*
Son las terminales de conexión de un oscilador de cristal o externo. La Figura 10.23 ilustra cómo se usan con un cristal. La frecuencia de cristal más común es 12 MHz.

6 *RESET*
Cuando hay una señal alta en esta conexión al menos en dos ciclos de máquina se reinicia el microcontrolador, esto significa que se pone en una condición que permite un sistema ordenado de inicio.

7 *Entrada/salida serial*
Al escribir en el búfer de datos serial SBUF en la dirección 99H carga los datos para transmisión; al 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.

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 10.24). Se carga como una entidad y no es direccionable por bit. El registro de control del temporizador TCON (Figura 10.25) contiene los bits de estado y control para los temporizadores 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.

Figura 10.24 Registro TMOD.

	7	6	5	4	3	2	1	0
89H	PUERTA	C/T	M1	M0	PUERTA	C/T	M1	M0
	Temporizador 1				Temporizador 0			

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 autorrecargable
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 10.25 Registro TCON.

	7	6	5	4	3	2	1	0
88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF0, TF1	Bandera de saturación del temporizador: establecido por el hardware cuando se alcanza la saturación y limpiado por el hardware cuando el procesador llama la rutina de interrupción
TR0, TR1	Bits de control del temporizador 1: temporizador encendido, 0 = temporizador apagado
IE0, IE1	Borde de interrupción de la bandera establecido por hardware al detectarse el borde o nivel bajo de la interrupción externa y limpiado cuando se procesa la interrupción
IT0, IT1	Software fija tipo de interrupción: 1 = borde descendente dispara la interrupción, 0 = nivel bajo dispara la interrupción

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 la COMPUERTA 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 10.26).

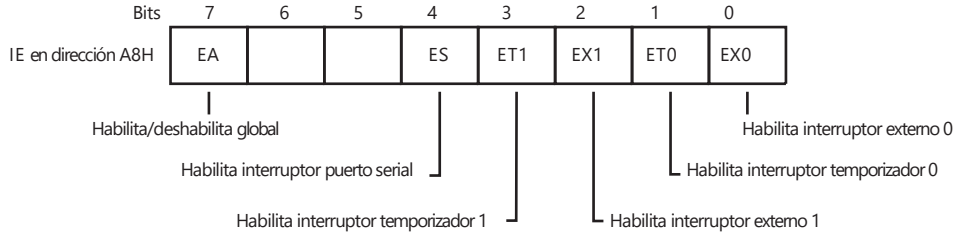


Figura 10.26 Registro IE.

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 10.27 Registros.

El término **registros de función especial** se usa para los registros de control de entrada/salida (Figura 10.27), como el IE descrito antes, éstos 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 registros de captura para los puertos 0, 1, 2 y 3.

10.3.3 Microcontroladores Microchip™

Otra familia de microprocesadores de 8 bits muy empleada es la de Microchip™. Usan el término PIC (Peripheral Interfaz Controller) para designar a sus microcontroladores de un solo chip. Éstos utilizan la **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 10.28). Los microcontroladores tratados en este capítulo no tienen buses separados y los datos del programa deben esperar que las operaciones de lectura/escritura y de entrada/salida se terminen 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

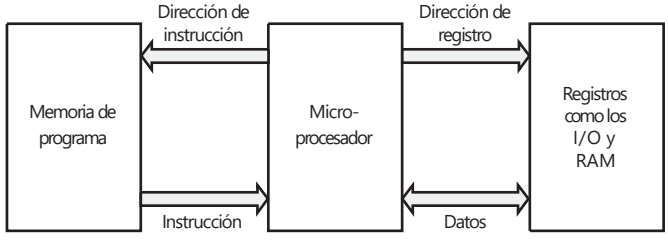
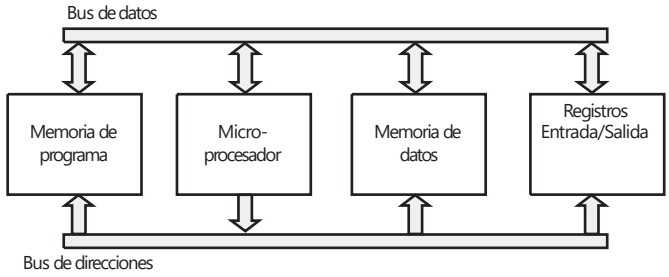


Figura 10.28 Arquitectura Harvard.



Ciclo de instrucción	Operación	
1	Intervalo de búsqueda de instrucción 1	
2	Intervalo de búsqueda de instrucción 2	Ejecución de una instrucción 1
3	Intervalo de búsqueda de instrucción 3	Ejecución de una instrucción 2
4	Intervalo de búsqueda de instrucción 4	Ejecución de una instrucción 3

Arquitectura de Harvard

Ciclo de instrucción	Operación
1	Intervalo de búsqueda de instrucción 1
2	Ejecución de una instrucción 1
3	Intervalo de búsqueda de instrucción 2
4	Ejecución de una instrucción 2

Arquitectura de von Neumann

ciclo después de su envío. La arquitectura Harvard permite una operación más rápida para una frecuencia de reloj dada. La Figura 10.29 muestra las conexiones de una de las versiones de los controladores PICI6C74A y el 16F84; la Figura 10.30 muestra su arquitectura.

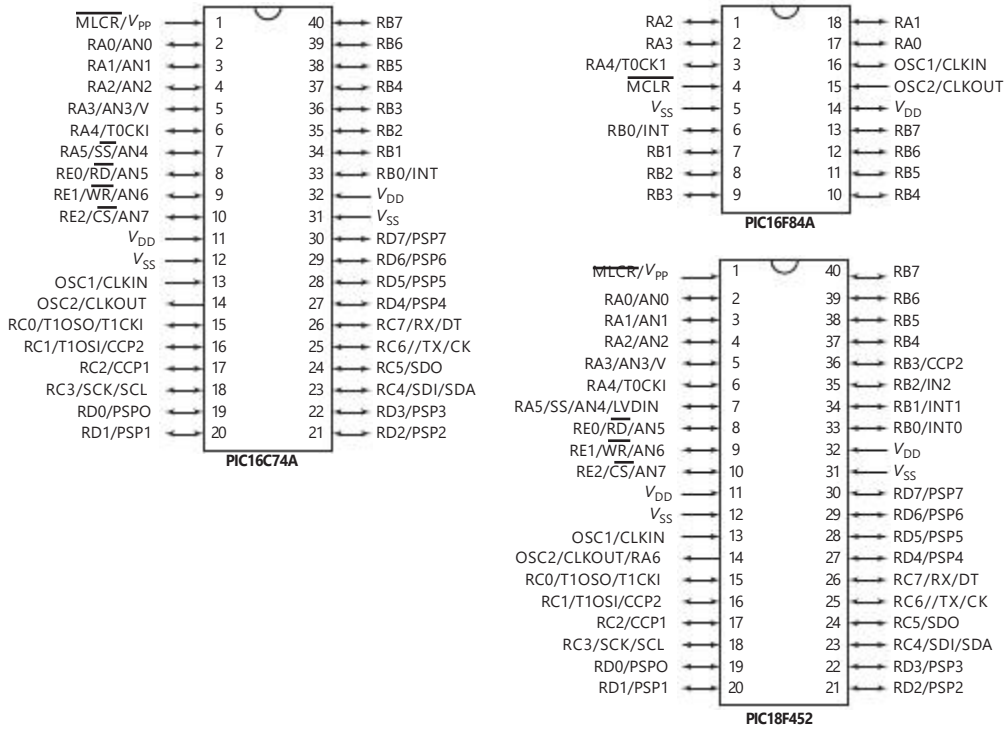


Figura 10.29 Diagramas de PIC.

Las características básicas del microcontrolador 16C74 y otros microcontroladores son:

1 *Puertos de entrada/salida*
Las terminales 2, 3, 4, 5, 6 y 7 corresponden al puerto A de entrada/salida bidireccional. Como en 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 10.31).

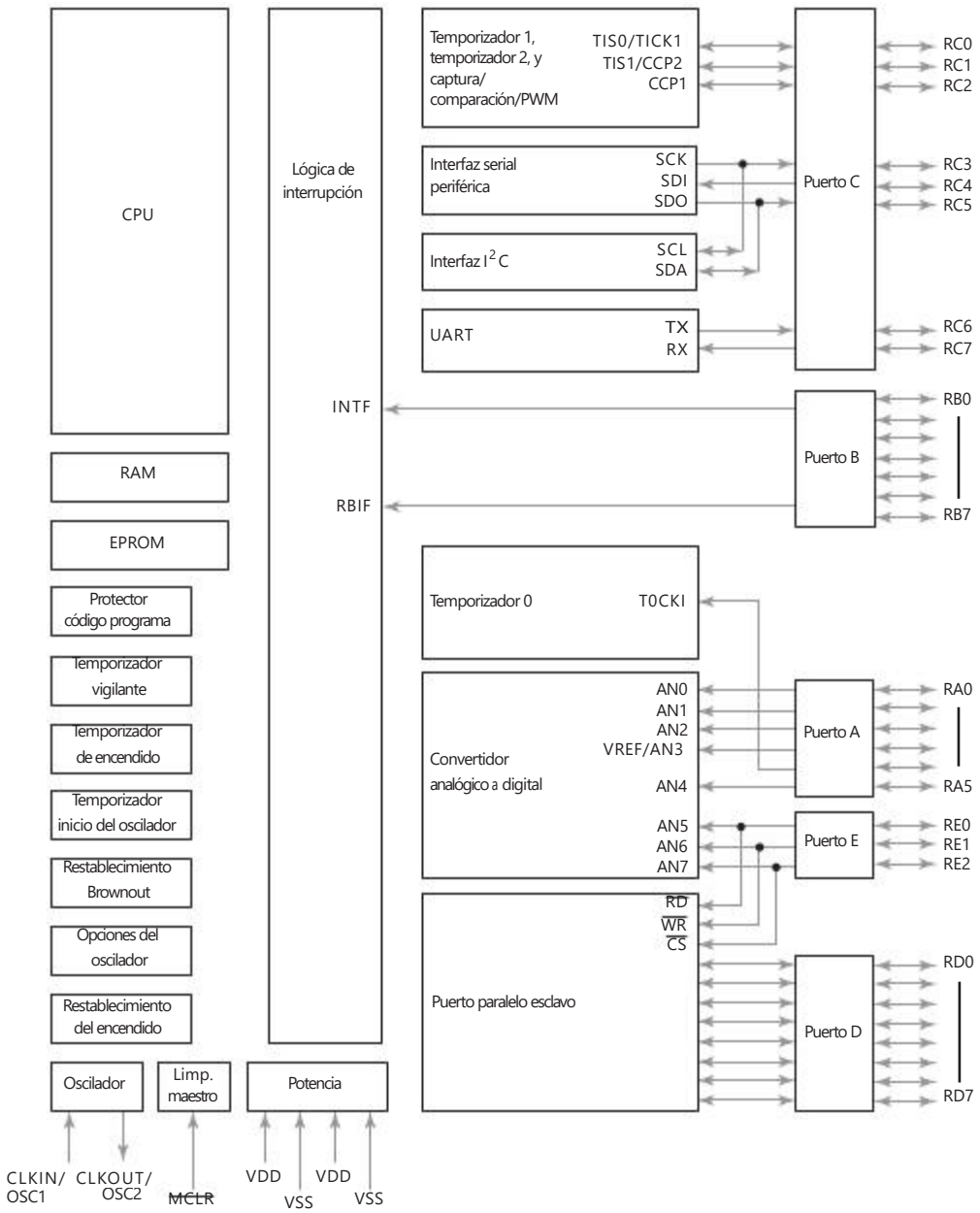


Figura 10.30 PIC16C74/74A.

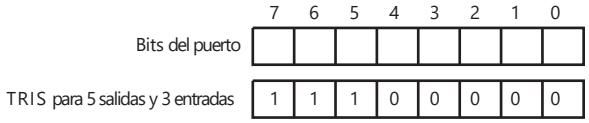


Figura 10.31 Dirección del puerto.

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 círculos de los circuitos de interfaz con computadoras personales, cuando en una aplicación las terminales de los puertos D y E se asignan a esta operación.

2 *Entradas analógicas*

Las terminales 2, 3, 4, 5 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 inicializar como aparece en la tabla anexa:

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

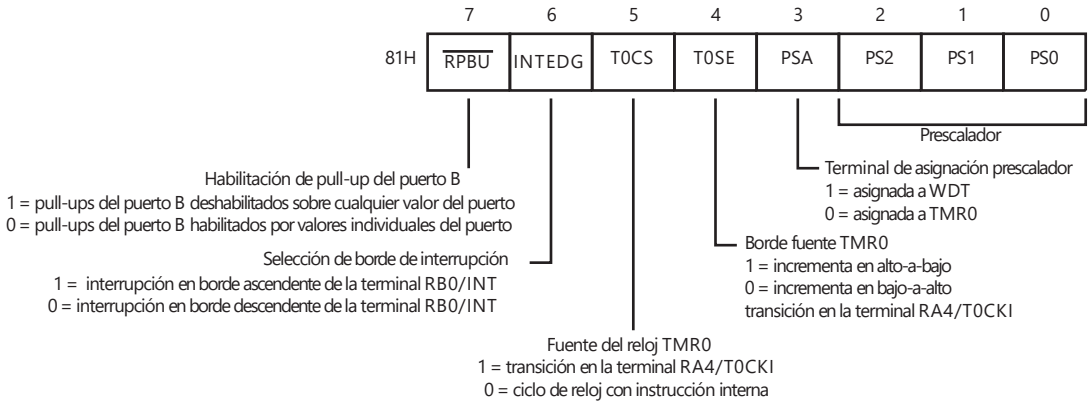


Figura 10.32 Registro OPTION.

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 TOC en el registro OPTION (Figura 10.32).

Si el preescalador no se selecciona, la cuenta se incrementa cada dos ciclos de la fuente de entrada. Se usa el preescalador 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 terminar el conteo y reiniciar el sistema si la operación no concluye en un tiempo razonable; el tiempo es normalmente de 18 ms.

Valores de la terminal preescalar			Relación	Relación
PS2	**PS1**	**PS0**	**TMR0**	**WDT**
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 versátil 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 se puede utilizar para controlar el periodo de una salida PWM. Las salidas PMW se alimentan en las terminales 16 y 17.

4 *Entrada/salida serial*

Los microcontroladores PIC incluyen un módulo de puerto serial sincrónico SSP y un módulo de interfaz 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 interfaz periférica serial (SPI) y el modo I^2C. El bus I^2C proporciona una interfaz 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 interfaz serial con una computadora personal.

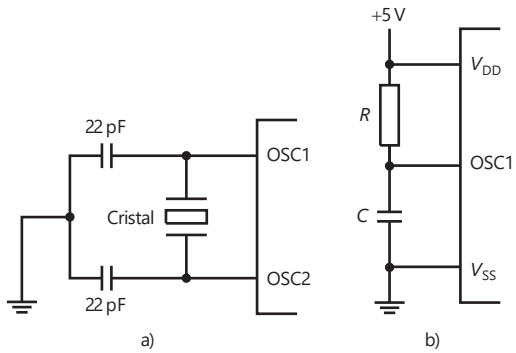
5 *Puerto paralelo esclavo*

El puerto paralelo esclavo usa los puertos D y E y habilita al microcontrolador para proporcionar una interfaz con una PC.

6 *Entrada de 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 10.33a) muestra el arreglo necesario para un control de frecuencia preciso. La Figura 10.33b) muestra una solución para el control de frecuencia de poco costo; para una frecuencia de 4MHz se tendría R = 4.7 *k*æ y C = 33 pF. La relación interna del reloj es la frecuencia del oscilador dividida entre 4.

Figura 10.33 Control de frecuencia.



7 *Borrador maestro*

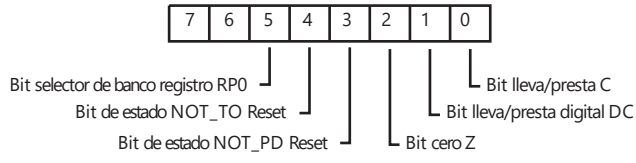
La terminal 1 es el borrador maestro; esto es, la entrada de restablecimiento y requiere un valor bajo para restablecer la unidad. Cuando se detecta una alza V_{DD}, se genera un pulso de energía en reposo (POR) para dar un retraso de tiempo fijo y mantener el procesador en estado de restablecimiento. Si el voltaje V_{DD} baja a un nivel específico para más de una cierta cantidad de tiempo, se activa un restablecimiento *brownout*. El temporizador vigilante es otra forma de restablecimiento, con tiempos fuera y reposos del microcontrolador si una operación no está concluida en un tiempo razonable.

Los **registros de propósito especial** (Figura 10.34) se usan para control de entrada/salida, como se ilustró para algunos de estos registros. Los regis-

Figura 10.34 Registros de propósito especial.

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 7Fh	Registros de propósito general	Registros de propósito general	A0h FFh

Figura 10.35 Registro STATUS.



tros 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 10.35).

10.3.4 Microcontroladores Atmel AVR y Arduino

Atmel AVR tiene un rango de microcontroladores de 8 bits y usa una arquitectura de Harvard modificada con el programa y con los datos almacenados en ubicaciones separadas de memoria. **Arduino** es una pequeña tarjeta microcontroladora con componentes complementarios que se ha diseñado para facilitar el uso del microcontrolador en los proyectos de control. La tarjeta básica, Arduino UNO revisión 3, usa un microcontrolador Atmel de 8 bits, Atmega 328. Éste tiene un sistema de memoria, puertos de entrada/salida, cronómetro/contador, modulación de ancho de pulso, ADC, y un sistema de interrupción y comunicación en serie.

Las tarjetas Arduino pueden comprarse previamente ensambladas con diferentes distribuidores. La tarjeta tiene un contacto bus universal en serie (USB) que le permite conectarse directamente a una computadora y varios enchufes de conexión para conectarse a elementos externos tales como motores, relevadores, etc. La tarjeta se energiza conectándola a una fuente externa de energía, por ejemplo una batería de 9 voltios, o a través de la conexión USB de la computadora. En el chip del microcontrolador de la tarjeta está preprogramado un cargador de arranque que permite directamente la carga de programas en la memoria del microcontrolador. Las tarjetas básicas se complementan con tarjetas accesorias, que se denominan tarjetas de pantalla, por ejemplo una pantalla LCD (pantalla de cristal líquido), una pantalla con motor y una pantalla Ethernet, que pueden enchufarse en la parte superior de la tarjeta básica Arduino y en cabezales con pasadores en la tarjeta. Las pantallas múltiples pueden apilarse.

La tarjeta es de **fuente abierta**. Esto significa que se permite que cualquiera haga tarjetas compatibles con Arduino. Se dispone de kits de arrancador, que comúnmente incluyen elementos tales como la tarjeta Arduino, un cable USB de modo que la tarjeta pueda programarse desde una computadora, un circuito experimental que se usa para el ensamblado de un circuito externo con cables y con componentes de uso común tales como resistores, potenciómetros fotorresistores, condensadores, pulsadores, sensor de temperatura, pantalla alfanumérica LCD, diodos emisores de luz (LED), motor DC (corriente directa), impulsor de motor con puente H, optocoples, transistores y diodos.

La Figura 10.36 muestra las características básicas de la tarjeta Arduino revisión 3. Éstas se describen enseguida.

1 *Pasador de restablecimiento (reseteado) en los conectores de energía de sección*
Esto restablece el microcontrolador de modo que inicie el programa desde el arranque. Para hacer un restablecimiento (reseteado) es necesario que este pasador se sintonice bajo por el momento, es decir que se conecte a 0V. Esta acción también puede iniciarse usando el interruptor de restablecimiento (reseteado).
2 *Otros pasadores en los conectores de energía*
Éstos suministran voltajes diferentes, es decir 3.5 V, 5 V, GND y 9 V.
3 *Entradas analógicas*
Están etiquetadas de A0 a A5 y pueden usarse para detectar las señales de voltaje.
4 *Conexiones digitales*
Etiquetadas como Digitales 0 a 13, pueden usarse tanto para entradas como para salidas. Las dos primeras de estas conexiones también están etiquetadas RX y TX para la recepción y la transmisión en la comunicación.

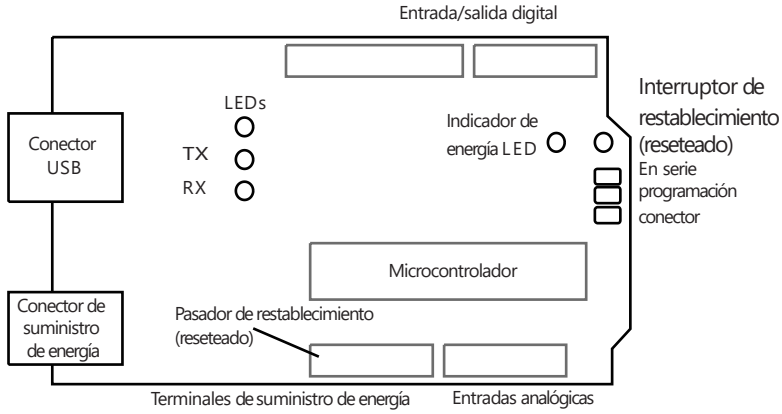


Figura 10.36 Los elementos básicos de una tarjeta Arduino UNO revisión 3.

5 *Conector USB*
Esto se usa para conectar la tarjeta a una computadora.

6 *Conector de programación en serie*
Esto ofrece un medio para la programación del Arduino sin usar el puerto USB.

7 *LED*
La tarjeta está equipada con tres LED, uno para indicar la transmisión en serie (TX), otro para la recepción (RX) y un LED extra para usarse en proyectos.

8 *Conector de suministro de energía*
Una fuente externa de energía puede conectarse a la tarjeta mediante un conector en la esquina inferior izquierda de la tarjeta. Sin embargo, la tarjeta puede energizarse desde el puerto USB durante su conexión a una computadora.

9 *Microcontrolador*
El microcontrolador ATmega328 viene prequemado con un cargador de arranque que permite la carga del código nuevo de programa sin el uso de un programador externo de hardware.

Cuando se compra una tarjeta Arduino generalmente tiene un programa de muestra preinstalado. Este programa es para hacer parpadear un LED en la tarjeta. Para arrancar el programa todo lo que necesita hacer es suministrar energía, lo que puede hacerse conectándose al puerto USB de una computadora. Entonces el LED deberá parpadear, verificándose así que la tarjeta está trabajando. Para instalar un nuevo software, que se denomina bocetos para las tarjetas Arduino, es necesario instalar el software Arduino y cargar los controladores USB en su computadora. Pueden encontrarse las instrucciones para hacer esto así como el software en el sitio de la red Arduino (www.arduino.cc). Una vez que esto se instale puede descargar un programa a la tarjeta Arduino.

10.3.5 Selección de un microcontrolador

Al elegir un microcontrolador se deben considerar los siguientes factores:

1 *Número de terminales de entrada/salida*
¿Cuántas terminales de entrada/salida son necesarias para realizar la tarea respectiva?

2 *Interfaces necesarias*
¿Cuántas 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 (sección 11.2.2), este tiempo está definido por el reloj del procesador.

Como ejemplo de la variación en los microcontroladores disponibles, la Tabla 10.1 muestra detalles de algunos de la familia Intel 8051; la Tabla 10.2 los de la familia PIC16Cxx y la Tabla 10.3 los de la familia M68HC11.

Tabla 10.1 Características de los miembros de la familia Intel 8051.

	ROM	EPROM	RAM	Temporizadores	Puertos E/S	Interrupciones
8031AH	0	0	128	2	4	5
8051AH	4K	0	128	2	4	5
8052AH	8K	0	256	3	4	6
8751H	0	4K	128	2	4	5

Tabla 10.2 Características de los miembros de la familia PIC16C.

	E/S	EPROM	RAM	Canales ADC	USART	Módulos CCP
PIC16C62A	22	2K	128	0	0	1
PIC16C63	22	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

10.4 Aplicaciones

Los siguientes son dos ejemplos de cómo se utilizan los microcontroladores. En el Capítulo 24 se presentan más casos.

10.4.1 Sistemas para medición de temperatura

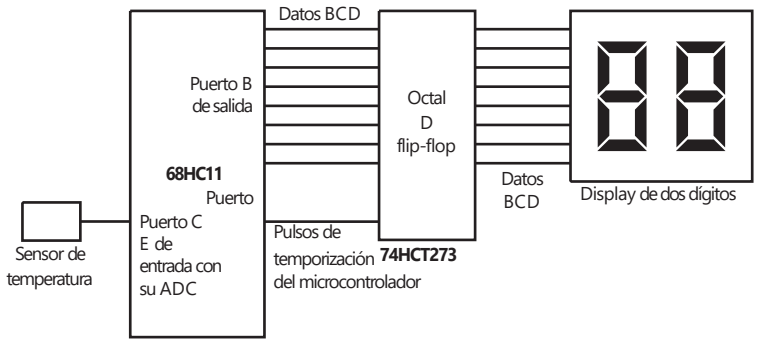
Para ilustrar en forma breve cómo se puede usar un microcontrolador, la Figura 10.37 muestra los principales elementos de un sistema de medición que usa un M68HC11. El sensor de temperatura da un voltaje proporcional a la temperatura (por ejemplo, un termotransistor como el LM35; vea la sección

Tabla 10.3 Características de los miembros de la familia M68HC11.

	ROM	EEPROM	RAM	ADC	Temporizador	PWM	E/S	Serial	Reloj E MHz
68HC11AO	0	0	256	8 c, 8 bits	(1)	0	22	SCI, SPI	2
68HC11A1	0	512	256	8 c, 8 bits	(1)	0	22	SCI, SPI	2
68HC11A7	8K	0	256	8 c, 8 bits	(1)	0	38	SCI, SPI	3
68HC11A8	8K	512	256	8 c, 8 bits	(1)	0	38	SCI, SPI	3
68HC11C0	0	512	256	4 c, 4 bits	(2)	2 c, 8 bits	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 vigilante, acumulador de pulsos; (2) captura de 3 o 4 entradas, comparación de 5 o 4 salidas, interrupción en tiempo real, temporizador vigilante, acumulador de pulsos. En serie: SC1 es una interfaz para comunicaciones en serie asíncrona, SPI es una interfaz para dispositivos periféricos en serie y síncronos.

Figura 10.37 Sistema de medición de temperatura.



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 elementos. 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.

10.4.2 Lavadora doméstica

La Figura 10.38 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 de 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

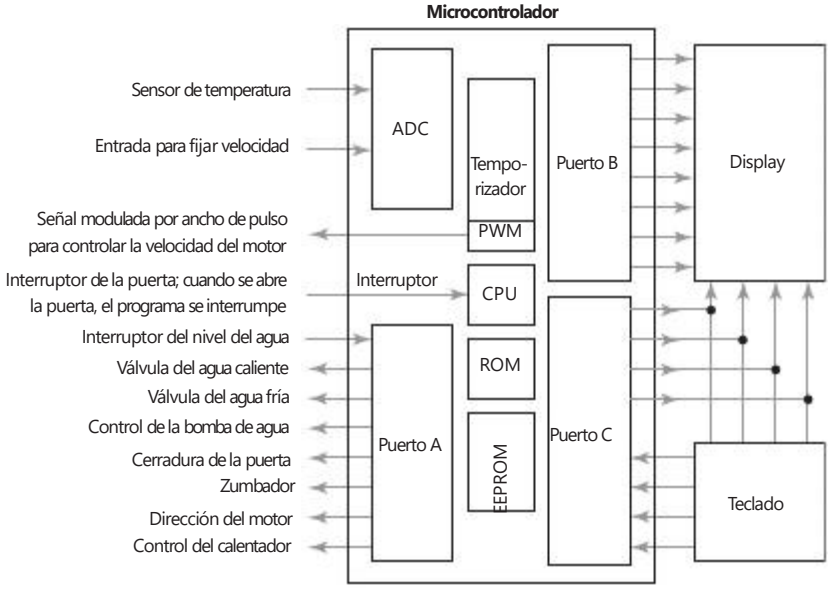


Figura 10.38 Lavadora doméstica.

también recibe las señales de entrada del teclado que se usan 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 se detiene si se abre la puerta de la lavadora.

10.5 Programacion

Un método común para diseñar programas es el siguiente:

1 Definir el problema, indicando con claridad qué función se espera que el programa ejecute, las entradas y salidas requeridas, las restricciones de velocidad de operación, exactitud, capacidad de memoria, etc.
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 10.39a) 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. La Figura 10.39a) muestra parte de un diagrama de flujo donde, después del inicio del programa va la operación A seguida por un ramal, ya sea la operación B o la operación C, dependiendo de si la decisión requerida es sí o no. Otra herramienta de diseño útil es el **seudocódigo**, que es una forma de describir los pasos de un algoritmo de una manera informal, que después se puede traducir en un programa (vea la siguiente sección).
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 lenguaje C, y luego se convierten en forma manual o con un programa ensamblador, en un código aceptable para el microprocesador, esto es código de máquina.

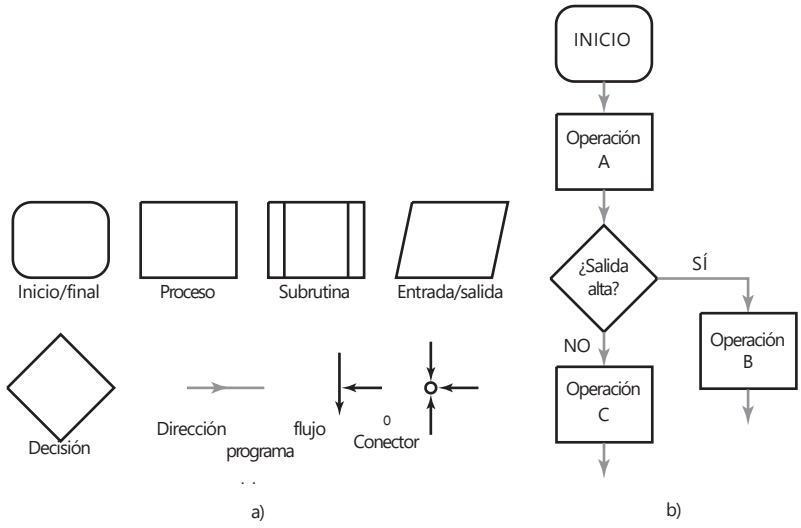


Figura 10.39 Diagrama de flujos: a) símbolos, b) ejemplo.

5 Probar y depurar el programa. Los errores o defectos en los programas se conocen como **bugs**, y el proceso de su rastreo y eliminación se llama **depuración de programas**.

10.5.1 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 de decisión IF-THEN-ELSE y el elemento de repetición WHILE-DO.

Una secuencia se escribiría como (Figura 10.40a):

```
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
```

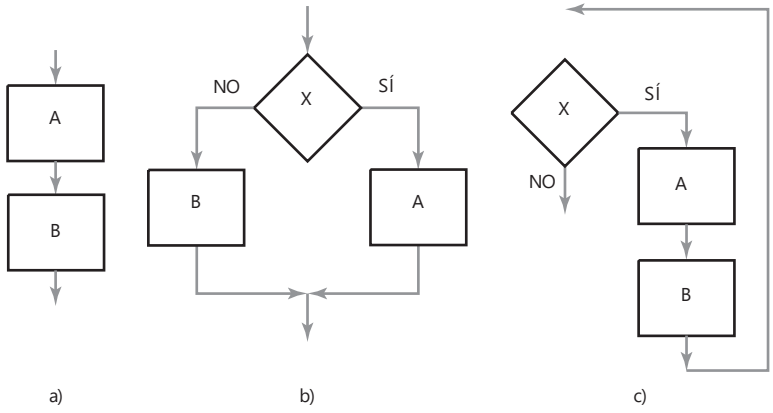


Figura 10.40 a) secuencia, b) IF-THEN-ELSE, c) WHILE-DO.

La Figura 10.40b) 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
```

La Figura 10.40c) muestra WHILE-DO como 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 11 se mostrará cómo elaborar programas en lenguaje ensamblador y en el Capítulo 12 cómo hacerlos en lenguaje C.

Resumen

Básicamente, los sistemas que incluyen **microprocesadores** constan de tres partes: una unidad de procesamiento central (CPU), interfaces de entrada y salida, y memoria. Dentro de un microprocesador, las señales digitales se mueven a lo largo de los **buses** que son trayectorias paralelas para transmisión de datos paralelos en lugar de datos en serie.

Los **microcontroladores** son la integración en un único chip de un microprocesador con memoria, interfaces entrada/salida y otros periféricos como temporizadores.

Un **algoritmo** es la secuencia de pasos que definen el método para resolver un problema. Los **diagramas de flujo** y el **seudocódigo** son dos métodos que sirven para describir estos pasos.

Problemas

10.1 Explique los roles de un microprocesador en a) un acumulador, b) estatus, c) dirección de memoria, d) registros de contadores de programa.

10.2 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?

10.3 Un chip de memoria tiene 8 líneas de datos y 16 líneas de dirección. ¿Cuál es su capacidad?

10.4 ¿Cuál es la diferencia entre un microcontrolador y un microprocesador?

10.5 Dibuje un diagrama de bloques de un microcontrolador básico y explique la función de cada subsistema.

10.6 ¿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?

10.7 ¿Cuántos bytes de memoria tiene el M68HC11A7 para la memoria de datos?

10.8 En el M68HC11 de Motorola el puerto C es bidireccional. ¿Cómo se debe configurar para que funcione como a) entrada, b) salida?

10.9 El M68HC11 de Motorola se puede utilizar en un solo chip y en modo ampliado. ¿Cuál es el propósito de estos modos?

10.10 ¿Para qué se utiliza la conexión ALE del 8051 de Intel?

10.11 ¿Qué entrada se requiere para restablecer el microcontrolador 8051 de Intel?

10.12 Represente en seudocódigo lo siguiente:

a) Si A es sí, entonces B o bien, C.

b) En tanto que A es sí, hacer B.

Capítulo once Lenguaje ensamblador

Objetivos

Después de estudiar este capítulo, el lector debe ser capaz de:

- **Utilizar el lenguaje ensamblador para escribir programas que contengan transferencias de datos, aritmética, lógica, jumps (saltos), branches (ramificaciones o controles de flujo), subrutinas, retrasos y tablas de consulta.**

11.1 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 se puede representar 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 lo pueda utilizar el microprocesador. En este capítulo se presenta un panorama general de cómo elaborar programas utilizando lenguaje ensamblador; en el Capítulo 12 se usa lenguaje C.

11.2 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 el apéndice C se dan conjuntos de instrucciones de los tres tipos más comunes de un microcontrolador. Dichas instrucciones establecen las diferencias entre un microprocesador y otro. En general, las instrucciones se clasifican en:

Transferencia de datos/movimiento

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 y por lo general se utiliza con microprocesadores Motorola, por ejemplo LDAA $0010:

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 A

2 *Almacenar* (*store*)
Esta instrucción copia el contenido de un registro especificado en una localidad de memoria dada y por lo general se usa con microprocesadores Motorola STA $0011:

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

3 *Mover* (*move*)
Esta instrucción se usa para mover datos dentro de un registrador o copiar datos desde un registro a otro y se usa con microprocesadores PIC e Intel, por ejemplo con PIC, MOV R5,A:

Antes de la instrucción	*Después de la instrucción*
Datos en el registrador A	Datos en el registrador A Datos copiados para registrar R5

4 *Limpiar* (*clear*)
Esta instrucción reinicia todos los bits a ceros, por ejemplo con Motorola, CLRA para limpiar el acumulador A; con PIC, CLRF 06 para limpiar el registro 06 del archivo.

Aritméticas

5 *Sumar (add)*
Esta instrucción suma el contenido de una localidad de memoria especificada con los datos de algún registro; por ejemplo, Intel, ADD A, #10h:

Antes de la instrucción	*Después de la instrucción*
Acumulador A con datos	Acumulador A más 10 hex

y con Motorola, ADDD #0020:

Antes de la instrucción	*Después de la instrucción*
Acumulador D con datos	Acumulador D más contenidos de localidad de memoria 0020

o los contenidos de un registro a los datos en un registro; por ejemplo, con Intel, ADD A, @R1:

Antes de la instrucción	*Después de la instrucción*
Acumulador A con datos	Acumulador A más contenidos de ubicación R1

y con PIC, addwf 0C:

Antes de la instrucción	*Después de la instrucción*
Registro 0C con datos	Registro 0C más contenidos de ubicación w

6 *Decrementar (decrement)*
Esta instrucción resta 1 del contenido de una localidad especificada. Por ejemplo, suponga que se tiene un registro 3 como localidad especificada y así con Intel, DEC R3:

Antes de la instrucción	*Después de la instrucción*
Registro R3 con datos 0011	Registro R3 con datos 0010

7 *Incremento* (*increment*)
Esta instrucción añade 1 a los contenidos de una ubicación especificada, por ejemplo INCA con Motorola para aumentar los datos en el acumulador A por 1, incf 06 con PIC para aumentar los datos en un registro 06 por 1.

8 *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.

Lógicas

9 *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 operación bit por bit, por ejemplo, con Motorola, ANDA %1001:

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

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. Con PIC, ANDLW 01 se agrega el número binario 01 al número en W y si el bit menos significativo es, por ejemplo, 0, entonces el resultado es 0.

10 *OR*
Esta instrucción lleva a cabo una operación lógica OR con los contenidos de una ubicación de memoria especificada y los datos en algún registro, bit por bit; por ejemplo, con Intel, ORL A, #3Fh con los contenidos OR del registro A con el número hexadecimal 3F.

11 *EXCLUSIVE-OR*
Esta instrucción aplica la operación lógica EXCLUSIVE-OR al contenido de la localidad de memoria especificada y a los datos en un registro dado;

la operación se realiza bit por bit. Por ejemplo con PIC, xorlw 81h (en binario 10000001):

Antes de la instrucción	*Después de la instrucción*
Registro w con 10001110	Registro w con 00001111

XORing con un 0 que deja bits de datos sin cambiar mientras que con 1 los bits de datos están invertidos.

12 *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. Con Motorola la instrucción podría ser LSRA para cambiar a la derecha y LSLA para cambiar a la izquierda.

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

13 *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 con instrucción de Motorola ASRA:

Antes de la instrucción	*Después de la instrucción*
Acumulador con dato 1011	Acumulador con dato 1001 El registro de estado indica acarreo 1

14 *Rotación* (*a la izquierda o a la derecha*)
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, instrucción Intel RR A:

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

Control del programa

15 *Salto o ramificación o control de flujo*
Esta instrucción modifica la secuencia de ejecución del programa. En general, el contador del programa ocasiona que se ejecute 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 (Figura 11.1a). Los saltos no condicionados ocurren sin que el programa pruebe la aparición de alguna condición. Por lo tanto, con Intel se puede tener LJMP

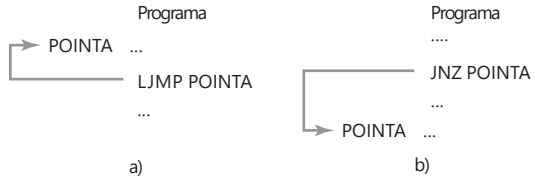


Figura 11.1 a) Salto no condicionado, b) salto condicionado.

POINTA para que el programa salte a la línea en el programa denominado POINTA, con Motorola la instrucción sería JMP POINTA y con PIC sería GOTO POINTA. Los saltos condicionados se dan si se produce alguna condición (Figura 11.1b). Con Intel se puede tener JNZ POINTA para que el programa salte a la línea en el programa denominado POINTA si algún bit en el acumulador no es cero, de cualquier manera éste continúa con la siguiente línea. JZ POINTA es todos los bits en el acumulador que son cero. Con PIC, un salto condicionado puede implicar dos líneas de código: BTFC 05,1 a "bit por bit", por ejemplo, probar si el bit 1 del archivo del registro 5, y si el resultado es 0 entonces éste salta a la siguiente línea del programa, si es que 1 la ejecuta. La siguiente línea es GOTO POINTA. Con la ramificación de Motorola es una instrucción de salto condicionado por el programa para determinar qué ramificación de un programa se seguirá si se reúnen las condiciones específicas. Por ejemplo, Motorola utiliza BEQ para ramificar si es igual a 0, BGE para ramificar si es mayor o igual a, BLE para ramificar si es menor o igual a.

16 *Paro*
Esta instrucción detiene la actividad del microprocesador.

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 está 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 o 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 y precedido por B, y para números en hexadecimal por H.

11.2.1 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 *Inmediato*
El dato que sigue al mnemónico es 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 código 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 mov1w H'25' para cargar el número 25 al registro de trabajo w, la H indica que es un número hexadecimal.

2 *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 MOV A,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 *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 *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 *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 *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 *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 11.1 muestra algunas instrucciones con los modos de direccionamiento utilizados en los sistemas de Motorola.

Tabla 11.1 Ejemplos de indexado.

Modo de dirección	Instrucción	
Immediato	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
Inherente	CLR A	Borrar acumulador A
Extendido	CLR $2020	Borrar dirección 2020, es decir, guardar todos los ceros 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

11.2.2 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					
		IMMED			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 **op-code** o **código de operación (OP)** 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 inherente.

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 ramificación.

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.

11.3 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 seudo operaciones (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 **seudooperaciones**, 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, etc. 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 bytes.
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.

11.3.1. Ejemplos de programas en lenguaje ensamblador

Figura 11.2 Diagrama de flujo para la suma de dos números.

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 transfiera 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 11.2 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     $C000   ; 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.

El mismo programa para un Intel 8051 sería:

Figura 11.3 Un ciclo.

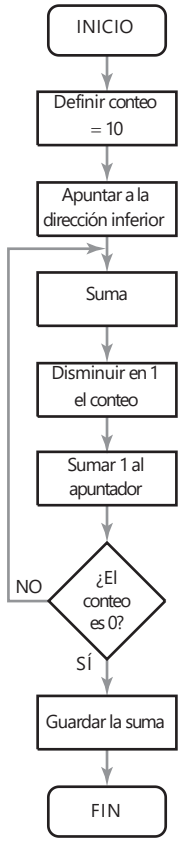


Figura 11.4 Diagrama de flujo de la suma de 10 números.

Programa 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 H'22'
        End
```

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 11.3 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 continúa 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 11.4 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 índices
                         ; 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 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, por ejemplo, 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 11.5 muestra el diagrama de flujo. El programa es:

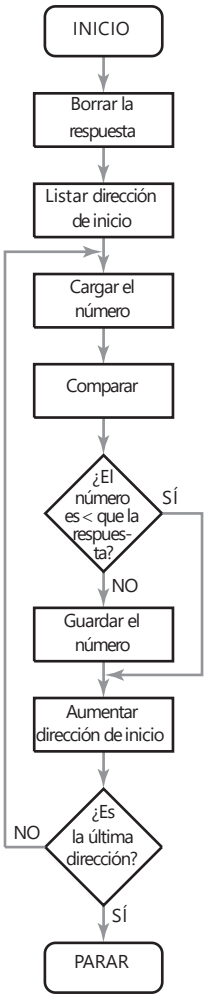


Figura 11.5 Diagrama de flujo para obtener el número mayor.

```
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 ramifica para repetir el ciclo con el siguiente número.

11.4 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. 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 ramificar 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 ramifica 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, 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 use para guardar datos en forma temporal. En este caso, las dos instrucciones son:

1 PUSH. Mediante esta instrucción los datos de los registros especificados se guardan en la siguiente localidad de la pila que esté libre.
2 PULL o POP. 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, quizá 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
```

11.4.1 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   ; Ramificar 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

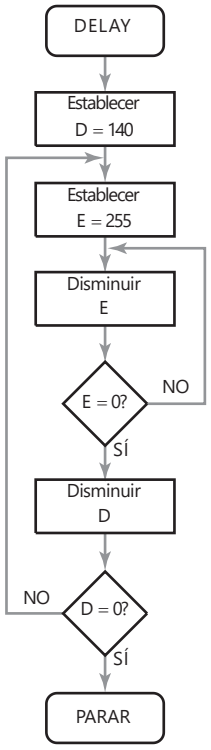


Figura 11.6 Ciclo de retardo anidado.

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 $1/(4 * 10^6)$ 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 11.6 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, por ejemplo, 140, el tiempo de retardo será 140 * 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 OLOOP 140 veces
```

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 al bit 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 LED.

La instrucción rotar se puede usar para encender en forma sucesiva los LED, 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 LED 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
          BNE   LOOP ; ramifica al ciclo ocho veces
DELAY     RTS           ; retraso simple corto
          END
```

11.5 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 1.7.2 donde el tiempo de encendido es una función del ángulo del eje del cigüeñal y de la presión en la entrada del múltiple. 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 cigüeñal.

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
        LDAB   $20          ; carga el acum. 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 Celsius 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+DPTR       ; obtiene el valor de la tabla
        RET                     ; regresa de la subrutina
TMP     DB      32, 34, 36, 37, ; dando valores a la tabla
                39, 41, 43, 45
```

Otro ejemplo del uso de una tabla es dar la secuencia de un número de salidas. Ésta puede ser la secuencia para operar las luces de un semáforo que controla el tráfico, que dé la secuencia rojo, rojo más ámbar, verde, ámbar. La luz roja se ilumina cuando hay una salida de RD0, la ámbar se ilumina con RD1 y la verde con RD2. Los datos de la tabla serían:

	Rojo	Rojo + ámbar	Verde	Ámbar
Índice	0	1	2	3
	0000 0001	0000 0011	0000 0100	0000 0010

11.5.1 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 ¿Éste 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.

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.

	Salidas requeridas desde el puerto B				
Paso	**PB0**	**PB1**	**PB2**	**PB3**	**Código**
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	4

```
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
        LDAA    #$80
        STAA    TFLG1,X         ; Borrar bandera
START   LDY     #STTBL
BEG     LDAA    0,Y             ; Empezar por la primera posición de la
                                ; tabla
        STAA    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, ramificar a BEG
        BRA     START           ; Si es así, ir de nuevo a inicio

DELAY   LDD     TCNT,X
        ADDD    #TEN_MS         ; Aumentar una demora de 10 ms
        STD     TOC2,X
HERE    BRCLR   TFLG1, X, $80,  ; Esperar hasta que haya transcurrido
                HERE            ; la demora
        LDAA    #$80
        STAA    TFLG1,X         ; Borrar bandera
        RTS
```

Observe que en la etiqueta TEN_MS hay un espacio subrayado para indicar que TEN y 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.

11.6 Sistemas embebidos

Los microprocesadores y microcontroladores a menudo son "embebidos" en sistemas donde se pueda ejercer el control. Por ejemplo, una lavadora de ropa moderna cuenta con un microcontrolador embebido que ha sido programado con los diferentes programas de lavado; todo lo que el operador de la máquina debe hacer es seleccionar el programa de lavado requerido por medio de un interruptor y se implementa el programa requerido. El operador no tiene que programar el microcontrolador. El término **sistema embebido** se utiliza para un sistema basado en un microprocesador diseñado para controlar una función o rango de funciones y no está diseñado para ser programado por el usuario del sistema. El fabricante ha hecho el programa y lo ha "quemado" en el sistema de la memoria y no se puede cambiar por el usuario del sistema.

11.6.1 Programas embebidos

En un sistema embebido los fabricantes hacen un ROM que contiene el programa. Esto es sólo económico si hay una necesidad para una gran cantidad de estos chips. De forma alternativa, para el prototipo o aplicaciones de bajo volumen, un programa puede ser cargado dentro del EPROM/EEPROM del hardware. Lo siguiente ilustra cómo se programa el EPROM/EEPROM de los microcontroladores.

Por ejemplo, para programar el EPROM del microcontrolador Intel 8051, se requiere el arreglo que se muestra en la Figura 11.7a). Debe haber una entrada de oscilador de 4-6 MHz. El procedimiento se esboza enseguida:

1 La dirección de una localidad EPROM, para ser programada en el rango de 0000H a 0FFFH, se aplica al puerto 1 y a las terminales P2.0 y P2.1 del puerto 2; al mismo tiempo, el código byte para ser programado dentro de esa dirección se aplica al puerto 0.
2 Las terminales P2.7, RST y ALE deben mantenerse en alta, las terminales P2.6 y PSEN en baja. Para las terminales P2.4 y P2.5 no importa si son altas o bajas.
3 La terminal EA/V_{PP} se mantiene en lógica alta justo antes de que ALE sea pulsado, luego hay una elevación de +21 V, ALE es pulsado hacia abajo para 50 ms para programar el código byte dentro de la ubicación direccionada, y luego EA se regresa a lógica alta.

La verificación del programa, por ejemplo la lectura del programa, se logra mediante el arreglo que se muestra en la Figura 11.7b).

1 La dirección de la ubicación del programa a ser leída se aplica al puerto 1 y las terminales P2.0 a P2.3 del puerto 2.
2 Las terminales EA/V_{PP}, RST y ALE deben mantenerse en alta, las terminales P2.7, P2.6 y PSEN en baja. Para las terminales P2.4 y P2.5 no importa si están en alta o en baja.
3 Los contenidos de la ubicación direccionada salen en puerto 0.

Se puede programar un bit de seguridad para denegar el acceso eléctrico mediante cualquier medio externo a la memoria del programa en el chip. Una vez programado este bit, éste sólo puede limpiarse al borrar por completo la

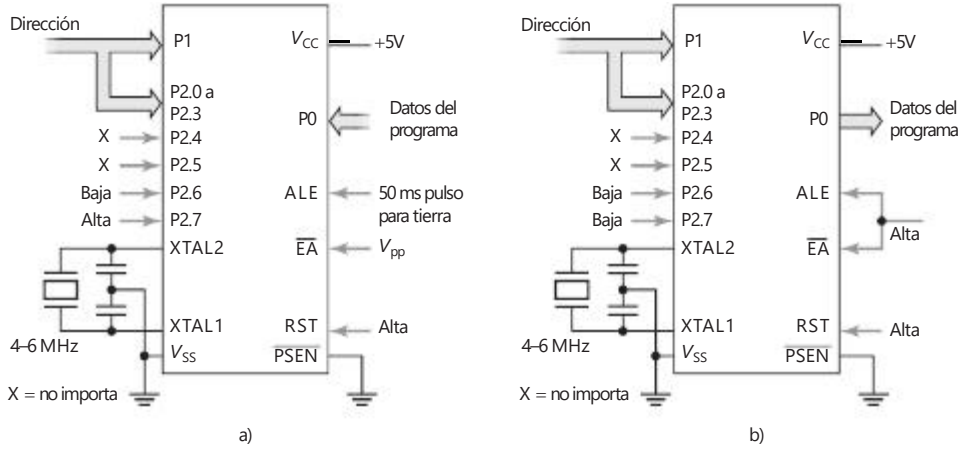
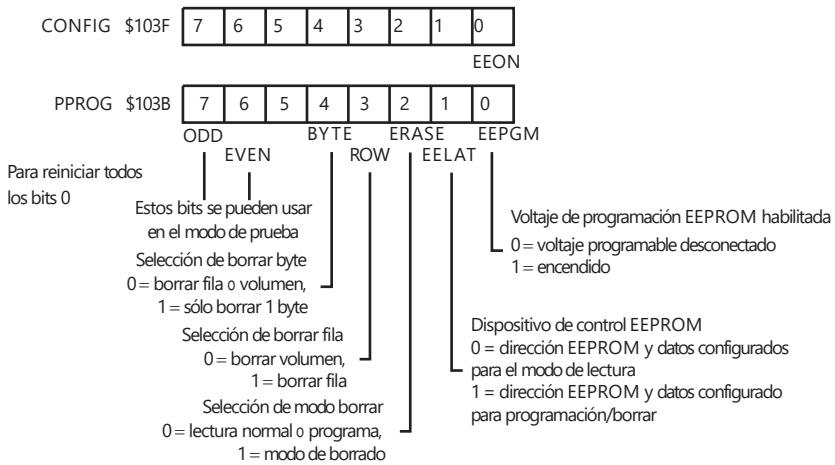


Figura 11.7 Intel 8051: a) programación, b) verificación.

memoria del programa. El mismo arreglo se utiliza como programación (Figura 11.7a) pero P2.6 se mantiene arriba. El borrado se logra por exposición a luz ultravioleta. Puesto que la luz del Sol y la luz fluorescente contienen algún rayo ultravioleta, la exposición prolongada (alrededor de una semana a la luz del Sol o 3 años con luz fluorescente en ambiente cerrado) debe evitarse y la ventana del chip debe protegerse con una etiqueta opaca.

El microcontrolador Motorola 68HC11 está disponible con una memoria interna eléctricamente programable y borrable de sólo lectura (EEPROM). El EEPROM se ubica en las direcciones $B600 a $B7FF. Como un EPROM se borra un byte cuando todos los bits son 1 y la programación implica la fabricación particular de bits 0. El EEPROM está posibilitado al fijar el bit EEON en el registro CONFIG (Figura 11.8) a 1 y deshabilitada al fijarla a 0. La programación se controla por el registro de programación EEPROM (PPROG) (Figura 11.8).

Figura 11.8 CONFIG y PPROG.



El procedimiento para la programación es:

1 Escribe al registro PPROG para fijar el bit EELAT a 1 para programación.
2 Escribe datos de la dirección seleccionada EEPROM. Esto se prende en la dirección y datos a ser programados.
3 Escribe el registro PPROG para fijar el bit EEPGM a 1 para encender en el voltaje de programación.
4 Retraso de 10 ms.
5 Escribe al registro de PPROG para apagar, por ejemplo a 0, todos los bits.

He aquí en lenguaje ensamblador, una subrutina de programación para uso con el MC68HC11:

```
EELAT     EQU   %00000010      ; bit EELAT
EEPGM     EQU   %00000001      ; bit EEPGM
PPROG     EQU   $1028          ; dirección de registro PPROG
EEPROG
          PSHB
          LDAB  #EELAT
          STAB  PPROG          ; filiar EELAT = 1 y EEPGM = 0
          STAA  0,X            ; datos de almacen X para dirección
                                 EEPROM
          LDAB  #%00000011
          STAB  PPROG          ; filiar EELAT = 1 y EEPGM = 1
          JSR   DELAY_10       ; salto para retrasar 10 ms la subrutina
```

```
                CLR     PPROG       ; borrar todos los bits del PPROG y
                                      regresar al modo de lectura
                PULB
                RTS

    ; Subrutina para aproximadamente 10 ms de retraso
DELAY_10
                PSHX
                LDX     #2500       ; contar para 20 000 ciclos
DELAY           DEX
                BNE     DELAY
                PULX
                RTS
```

El procedimiento para borrar es:

1 Escribe al registro PPROG para seleccionar borrar un byte, fila o el EEPROM completo.
2 Escribe a una dirección EEPROM dentro del rango a ser borrado.
3 Escribe a 1 el registro PPROG para encender el bit EEPGM y por lo tanto el voltaje borrador.
4 Retraso por 10 ms.
5 Escribe ceros al registro PPROG para apagar todos los bits.

Con el EEPROM construido con un microcontrolador PIC, un programa para escribir datos dentro es (Figura 11.9):

```
                bcf     STATUS, RP0     ; Cambia a Banco 0 para los datos
                mov.f   Data, w         ; Datos de carga para escribirse
                movwf   EEDATA
                movf    Addr, w         ; Dirección de carga de los datos a
                                          escribir
                movwf   EEADR
                bsf     STATUS, RP0     ; Cambia a Banco 1
                bcf     INTCON, GIE     ; Interrupciones inhabilitadas
                bsf     EECON1, WREN    ; Habilitada para escritura
                movlw   55h             ; Secuencia especial para habilitar
                                        ; escritura
                movwf   EECON2
                movlw   0AAh
                movwf   EECON2
                bsf     EECON1, WR      ; iniciar ciclo de escritura
                bsf     INTCON, GIE     ; Interrupciones rehabilitadas
EE_EXIT         btfsc   EECON, WR       ; Verificar que la escritura esté
                                        ; completa
                goto    EE_EXIT         ; Si no es así, reintentar
                bsf     EECON, WREN     ; Escritura EEPRON completada
```

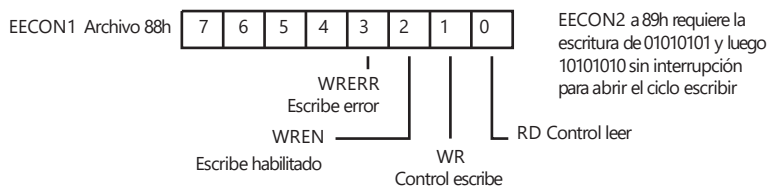


Figura 11.9 Registros EECON.

Resumen

La colección de instrucciones que un microprocesador reconocerá es su **conjunto de instrucciones**. La serie de instrucciones necesarias para llevar a cabo una tarea en particular se denomina **programa**.

Los microprocesadores trabajan en código binario. Las instrucciones escritas en código binario son referidas como **código de máquina**. Un código taquigráfico que utiliza términos sencillos e identificables en lugar del código binario se llama **código mnemónico**, un código mnemónico es un código "auxiliar para la memorización". A este código se le conoce como **lenguaje ensamblador**. Los programas de lenguaje ensamblador consisten en una secuencia de instrucciones, una por línea, cada una con uno o cuatro campos: etiqueta, código de operación, operando y comentario. La **etiqueta** es el nombre que recibe una entrada en particular en la memoria. El **código de operación** especifica cómo manipular los datos. El **operando** contiene la dirección de los datos a operar. El campo de **comentario** es para permitir al programador incluir comentarios que podrían hacer más comprensible el programa al lector.

Problemas

11.1 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 direccionamiento inmediato, b) de direccionamiento directo.

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

11.2 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?

11.3 ¿Cuáles son los mnemónicos 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?

11.4 Escriba una línea de programa ensamblador para a) cargar el acumulador con 20 (hex), b) decrementar el acumulador A, c) borrar la dirección $0020, d) SUMAR al acumulador A el número en la dirección $0020.

11.5 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.

11.6 Escriba los programas en lenguaje ensamblador para:

a) Restar un número hexadecimal en la dirección de la memoria 0050 desde el número hexadecimal en la ubicación de la memoria 0060 y almacene el resultado en la ubicación 0070.

b) Multiplique dos números de 8 bits ubicados en las direcciones 0020 y 0021, y almacene el producto, un número de 8 bits, en la ubicación 0022.
c) Almacene los números hexadecimales de 0 a 10 en las ubicaciones de la memoria empezando en 0020.
d) Mueva el bloque de 32 números empezando en la dirección $2000 para una nueva dirección de inicio de $3000.

11.7 Escriba en lenguaje ensamblador una subrutina que se pueda usar para producir un retraso y que pueda fijarse a cualquier valor.

11.8 Escriba en lenguaje ensamblador una rutina que se pueda usar de manera que si la entrada producida por un sensor en la dirección 2000 si es alta, el programa salta a una rutina que empieza en la dirección 3000; si es baja, el programa continúa.

Capítulo doce Lenguaje C

Objetivos

Después de estudiar este capítulo, el lector debe ser capaz de:

- **Comprender las principales características de los programas en lenguaje C.**
- **Utilizar C para la escritura de programas sencillos para microcontroladores.**

12.1 ¿Por qué el lenguaje C?

Este capítulo intenta dar una introducción al lenguaje C y la escritura de programas. C es un lenguaje de alto nivel que a menudo se utiliza en vez del lenguaje ensamblador (Capítulo 11) 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 (American National Standards Institute).

12.2 Estructura de un programa

La Figura 12.1 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.

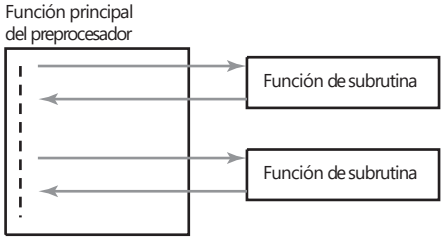


Figura 12.1 Estructura de un programa C.

12.2.1 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 los espacios y los cambios de línea y sólo se usan para comodidad del programador, ya que facilitan la lectura del programa.

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 llaves, **{ }**. 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");
```

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*

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. Las reglas aritméticas funcionan igual para estas operaciones. 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	Símbolo
AND	&&
OR	||
NOT	!

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

14 *Operaciones sobre bits*
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, queden 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 *Cadena o secuencia*
La serie de caracteres comprendida dentro de comillas, `` ´´, se conoce como cadena de secuencia. 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 de secuencia 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
\?	signo de interrogación
\'	apóstrofo

12.2.2 Ejemplo de un programa en C

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

```
/*A simple program in C*/

# include <stdio.h>
void main(void)
{
```

```
    int a, b, c, d; /*a, b, c y d son enteros*/
    a = 4; /*a se le asigna el valor 4*/
    b = 3; /*b se le asigna el valor 3*/
    c = 5; /*c se le asigna el valor 5*/
    d = a * b * c; /*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.

12.3 Control de flujo y ciclos

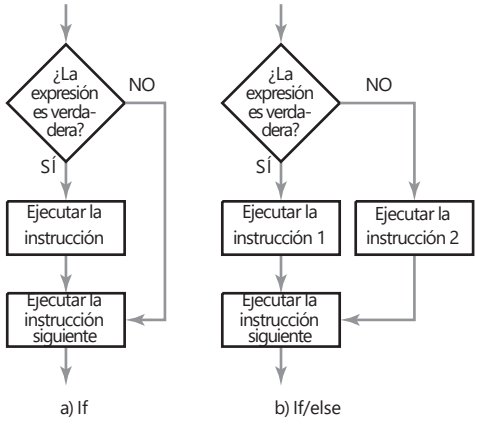
Las instrucciones que permiten el control de flujo y la realización de ciclos en los programas son *if* (si), *if/else* (si/de otra manera), *for* (para), *while* (mientras) y *switch* (conmutar).

1 *If*

La instrucción *if* produce una ramificación (Figura 12.2a). 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 12.2 a) If, b) if/else.



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

```
#include <studio.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;
}
```

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 cuál 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 12.2b). Por ejemplo:

```
#include <studio.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 12.3a) 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;
```

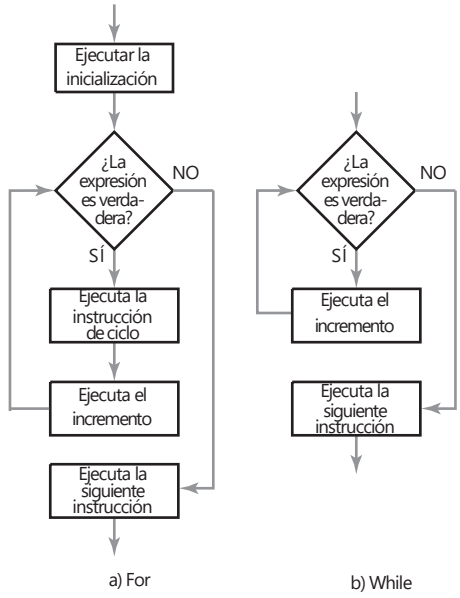
Un ejemplo de cómo se usa es

```
#include <studio.h>

int contador

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

Figura 12.3 a) For, b) while.



El valor inicial de contador es 0, se incrementa en 1, se hace un ciclo y se repite la instrucción for en tanto que el 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.

4 *While*

Con esta instrucción la repetición de un ciclo continúa mientras la expresión sea verdadera (Figura 12.3b). 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 del contador es menor que 7, y despliega los resultados.

```
#include <studio.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.

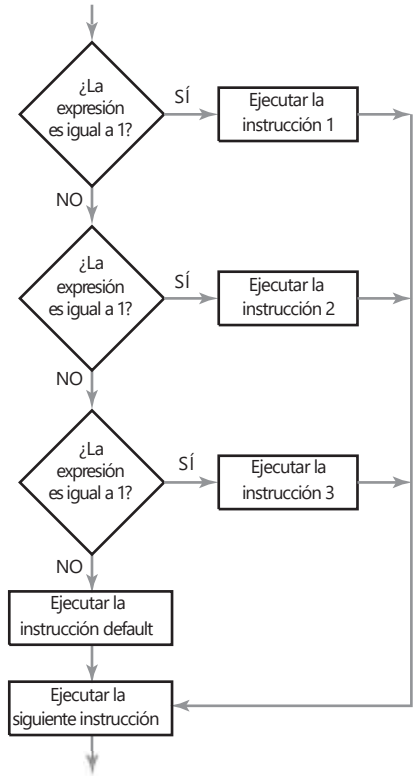
5 *Switch*

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, y así sucesivamente. 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 12.4):

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

Figura 12.4 Switch.



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;
}
```

12.4 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 corchetes [ ], 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 12.5a) muestra la forma de un arreglo secuencial. Para almacenar valores en el arreglo, se puede escribir:

```
temperatura [0] = 22.1;
temperatura [1] = 20.4;
etc.
```

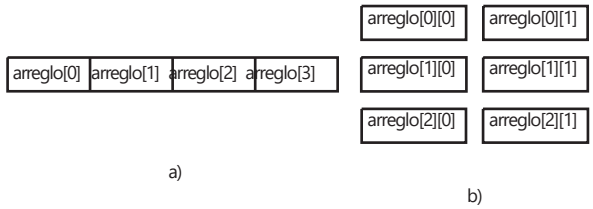


Figura 12.5 a) Arreglo secuencial de cuatro elementos, b) arreglo bidimensional.

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};
```

Existe la posibilidad de emplear **arreglos multidimensionales**. Por ejemplo, una tabla de datos es un arreglo bidimensional (Figura 12.5b), donde x representa la fila en tanto que y es la columna, y se escribe como:

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

12.5 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 pointer*/

   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.

12.5.1 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*/
```

12.5.2 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.

12.6 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 reeditar el código fuente. El compilador almacena el código de máquina en otro archivo.

3 *Vinculación 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.

12.6.1 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 como get (obtener, entradas, es decir lee información de un dispositivo), put (poner, salidas, es decir escribe in-

formación en un dispositivo) y scanf (leer datos); <math.h> contiene funciones matemáticas 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 <reg.51.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 e 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, el encabezado <hc11e9.h> define los registros para un MC68HC11E9 de Motorola, 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 como teclados y pantallas de cristal líquido.

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

12.7 Ejemplos de programas

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

12.7.1 Encendido y apagado de un motor

Suponga que desea programar el microcontrolador M68HC11 para arrancar y detener un motor de c.d. 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 12.6). 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 establece como 0 y el puerto C queda definido para recibir entradas.

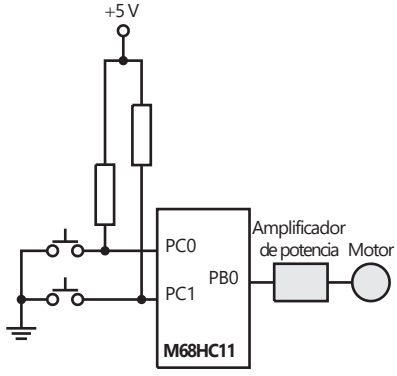


Figura 12.6 Control de un motor.

El programa correspondiente sería:

```
#include <hc11e9.h>    /*incluye el archivo de encabezado*/

void main(void)
{
   PORTB.PB0 &=0; /*al inicio asegura que 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 de 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 apliça 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.

12.7.2 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 12.7). 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.

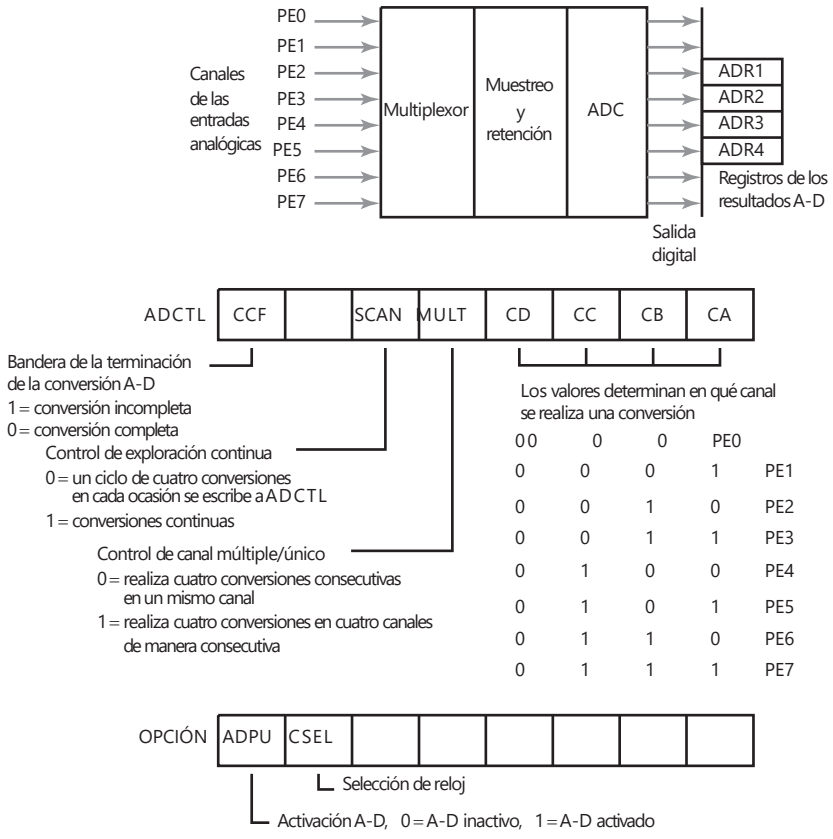
El programa sería el siguiente:

```
#include <hc11e9.h>   /*incluye el archivo de encabezado*/

   void main(void)
   {
      unsigned int k; /*da el número del canal*/

      OPTION=0; /*esta línea y las siguientes encienden el ADC*/
      OPTION.ADPU=1;
```

Figura 12.7 Convertidor ADC.




```
    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*/
}
```

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.

12.8 Programas Arduino

Se usa el término **esbozo** para programas que se utilizan con la tarjeta Arduino. Arduino usa el lenguaje C para sus programas. El formato básico de estos programas consiste de dos funciones, setup (configuración) y loop (ciclo). La función setup se ejecuta al inicio del programa y se usa para configurar pasadores, y para declarar variables, constantes, etc. La función loop se ejecuta paso a paso y, cuando llega al final del ciclo, regresa automáticamente al pri-

mer paso de la función loop y continúa repitiendo el ciclo hasta que el programa se detiene.

```
void setup( )
{
//el código para el set up se coloca aquí
}

void loop ( )
{
// los pasos del código se suministran aquí
}
```

Observe que en los programas Arduino el primer **{** algunas veces se pone en el renglón que está después del comando y entonces el programa anterior se vería como sigue:

```
void setup( ) {
// el código para el set up se coloca aquí
}
void loop ( ) {
// los pasos del código se suministran aquí
}
```

La misma función setup llama a dos funciones incorporadas, pinMode y digitalWrite. La función pinMode hace a un pasador específico igual a una entrada o a una salida, ya que los pasadores digitales Arduino pueden funcionar como entradas o salidas. La función digitalWrite hace que un pasador sea HIGH (ALTO) o LOW (BAJO). Estas dos funciones no devuelven un valor, de modo que el programa tiene que decir que están vacías. Como ejemplo, considere un programa que hace que parpadee un diodo emisor de luz (LED) en la tarjeta, el cual ha sido conectado internamente al pasador 13. Los comentarios en los programas se encierran entre /* y */ cuando están en más de un renglón o simplemente están precedidos por // cuando están en un solo renglón. Los comentarios no se compilan en lenguaje de máquina para cargarse en el microcontrolador.

```
//Encienda el LED interior durante 0.5 s, luego apáguelo durante 0.5 s, en
forma repetida.
void setup ( )
{
        pinMode (13, OUTPUT);
}
void loop ( )
{
        digitalWrite (13, HIGH);
        delay (500);
        digitalWrite (13, LOW);
}
```

La función delay es para crear una demora de 0.500 s entre la escritura del pasador 13 HIGH y luego la escritura de LOW.

Suponga que queremos encender un LED externo durante 0.5 s, luego apagarlo durante 0.5 s, en forma repetida. Necesitamos especificar a cual pa-

sador está conectado el LED y si el pasador debe considerarse como una entrada. Un punto importante cuando se conecta un LED es que comúnmente la caída de voltaje a través de éste se limita a aproximadamente 2 V con una corriente de 20 mA. Como la tarjeta suministra 5 V, es necesario colocar un resistor en serie con el LED de modo que la caída de voltaje a través de él será de 3 V, dejando solamente una caída de 2 V para el LED. Entonces, la resistencia mínima necesaria es V/I = 3/0.020 = 150 Ω. Generalmente se usará un valor más alto, ya que el LED puede resplandecer con bastante brillantez con menos corriente.

```
//Apague un LED externo durante 0.5 s, luego enciéndalo durante 0.5 s, en
forma repetida.

#define ext_LED 12
void setup( )
{
        pinMode (ext_LED, OUTPUT);
}
void loop ( )
{
        digitalWrite (ext_LED,LOW);
        delay (500);
        digitalWrite (ext_LED, HIGH);
        delay (500);
}
```

Ahora considere la operación con un LED externo y uno interno.

```
/*Encienda el LED interno y apague el LED externo durante 0.5 s, luego
apague el LED interno y encienda el externo durante 0.5 s, en forma repetida.

#define int_LED 13
#define ext_LED 12

void setup ( )
{
  pinMode (int_LED, OUTPUT);
  pinMode (ext_LED, OUTPUT);
}

void loop ( )
{
  digitalWrite (int_LED, HIGH);
  digitalWrite (ext_LED, LOW);
  delay (500);
  digitalWrite (int_LED, LOW);
  digitalWrite (ext_LED, HIGH);
  delay (500);
}
```

Entonces el programa anterior hace que los LED interno y externo parpadeen en forma repetida alternativamente. Una posible mejora del programa anterior es que esto suceda sólo si se cierra un interruptor.

```
/*Si un interruptor se cierra, encienda el LED interno y apague el LED externo
durante 0.5 s, luego apague el LED interno y encienda el externo durante 0.5
s, en forma repetida.*/
*/
#define int_LED 13
#define ext_LED 12
#define ext_sw 11
Int switch_value;

void setup ( )
{
   pinMode (int_LED, OUTPUT);
   pinMode (ext_LED, OUTPUT);
   pinMode (ext_sw. INPUT);
}

void loop ( )
{
switch_value = digitalRead(ext_sw);
  if (switch_value ==LOW)
    {
     digitalWrite (int_LED, HIGH);
     digitalWrite (ext_LED, LOW);
     delay (500);
     digitalWrite (int_LED, LOW);
     digitalWrite (ext_LED, HIGH);
     delay (500);
     }
  else
    {
    digitalWrite(int_LED, LOW);
   digitalWrite(ext_LED, LOW)
   }
}
```

Lo anterior da solamente una introducción sencilla de cómo escribir programas para el Arduino. Los procedimientos requeridos son esencialmente aquellos que se esbozaron con anterioridad en este capítulo con C. Muchos programas ya escritos están disponibles gratuitamente en el sitio de la red de Arduino.

Para usar un programa en lenguaje C con Arduino, primero se descarga el programa denominado Arduino Development Environment (Ambiente de desarrollo de Arduino) del sitio de la red de Arduino a la computadora huésped. Este programa permite que se teclee en la computadora código en lenguaje C y luego compila el programa, verificando que se adhiera a las reglas del lenguaje C y lo traduce a lenguaje ensamblador y luego a código de máquina, ya que éste es el lenguaje que entiende la tarjeta Arduino. Cuando arranca la tarjeta Arduino por primera vez, ingresa el cargador de arranque, que es un código que ha sido descargado en su memoria en la fábrica y permite que los programas se carguen mediante su conector USB. Si entonces recibe un comando proveniente de la computadora huésped para cargar un

programa, el programa con el código de máquina se carga en la memoria Arduino de modo que queda disponible para que lo use el microcontrolador Arduino.

Básicamente, la secuencia de operaciones es:

1 Descargar el Arduino Development Environment en la computadora huésped desde el sitio de la red de Arduino.
2 Conectar la tarjeta Arduino a la computadora huésped mediante un cable USB.
3 Arrancar el Arduino Development Environment.
4 Teclear el programa C en la computadora.
5 Seleccionar el botón Upload (descargar) en la pantalla.
6 El programa corre en la tarjeta Arduino.

Resumen

El lenguaje C es de alto nivel el cual tiene ventajas cuando se le compara con el lenguaje ensamblador, su uso es más sencillo y se pueden utilizar diferentes microprocesadores con el mismo programa; todo lo que se necesita para esto es que se utilice un compilador apropiado que traduzca el programa C a un lenguaje de máquina que tenga relevancia con el microprocesador. El lenguaje ensamblador es distinto para los diferentes microprocesadores, en tanto que el lenguaje C está estandarizado.

Los paquetes C están provistos de bibliotecas que cuentan con una gran cantidad de funciones predeterminadas con el código C ya escrito. Para utilizar el contenido de cualquier biblioteca en particular, ésta debe especificarse en un archivo de encabezado. Cada programa C debe tener una función denominada main( ), la cual ejerce control cuando se ejecuta el programa y es la primera función a la que se invoca. Un programa consta de instrucciones, cada una definida con punto y coma. Si se ponen las instrucciones entre llaves { }, se pueden agrupar en bloques.

Problemas

12.1 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>
```

12.2 Para el siguiente programa indique las razones por las que se incluye la línea a) #include <stdio.h>, b) las llaves { }, c) /d y d) ¿qué aparece en la pantalla cuando se ejecuta el programa?

```
#include <stdio.h>

main( )
{
   printf(/d"problema 3");
}
```

12.3 ¿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;
}
```

12.4 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".

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

12.6 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)
      print f("No se puede dividir entre cero")
   else
      printf("El resultado es: %d", num1/num2);

   return 0;
}
```

Capítulo trece Sistemas de entrada/salida

Objetivos

Después de estudiar este capítulo, el lector debe ser capaz de:

- **Identificar los requerimientos de interfaz y cómo se pueden verificar: búfers, handshaking (reconocimiento), poleo e interfaz serial.**
- **Explicar cómo se utilizan las interrupciones con microcontroladores.**
- **Explicar la función de los adaptadores de interfaz periférica y ser capaz de programarlos para situaciones particulares.**
- **Explicar la función de los adaptadores de interfaz de comunicación asíncrona.**

13.1 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 como relevadores y motores. El término **periférico** designa un dispositivo, que puede ser un sensor, un teclado, un actuador, etc., 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 interfaz, que permite el acoplamiento entre los dispositivos periféricos y el microprocesador. La Figura 13.1 ilustra esta configuración. La interfaz es la parte donde se elimina la incompatibilidad.

Figura 13.1 Las interfaces.

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

13.2 Direccionamiento entrada/salida

Existen dos formas en que el microprocesador puede seleccionar los dispositivos de entrada/salida. Algunos microprocesadores, por ejemplo, el Zilog Z80, tiene **entradas y salidas aisladas,** e instrucciones de entrada especiales como IN que se utiliza para leer desde un dispositivo de entrada, e instruccio-

nes 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 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.

13.2.1 Registros de entrada/salida

El microcontrolador 68HC11 de Motorola tiene cinco puertos A, B, C, D y E (sección 10.3.1). Los puertos A, C y D son bidireccionales y se pueden usar para entrada o 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 10.12) 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 es 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 dirección $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 sería:

```
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 serían:

```
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 para 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
                            ; de datos a partir de REGBAS
         CLR    DDRC,X      ; llenar DDRS con 0
```

Para el microcontrolador 8051 de Intel (sección 10.3.2) 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 escribe 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 los registros de dirección TRIS (sección 10.3.3). 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

TRISB hacerlo 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:

```
Output   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
         bcf    STATUS,RP0     ; usa el registro de estado (status) para
                               ; seleccionar el banco 0
                               ; el puerto B es ahora una salida que se hizo 0
```

13.3 Requerimientos de una interfaz

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

1 *Acoplamiento mediante búfer/aislamiento eléctrico*
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 **búfer** 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 búfer 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 (handshake lines)**, y el proceso como **reconocimiento (handshaking)**.

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 quizá requerir una palabra más larga que la del microprocesador.

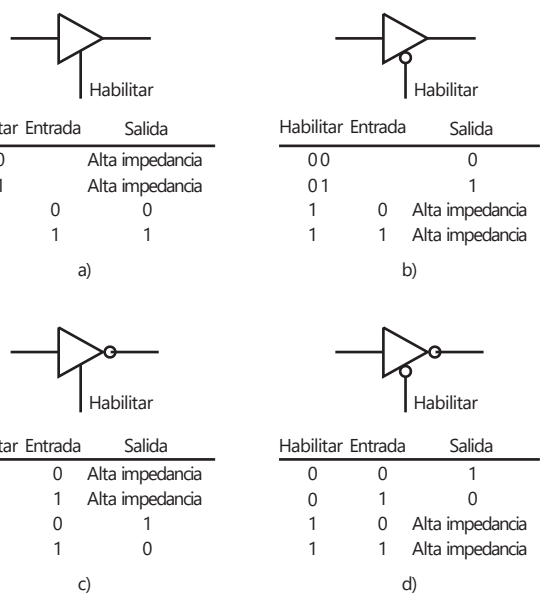
5 *Transferencia de datos en serie a paralelo y viceversa*
En un microprocesador de 8 bits en general los datos se manipulan 8 bits a la vez. Para transferir de manera simultánea 8 bits a un dispositivo periférico se necesitan ocho rutas de datos. Esta forma de transferencia se llama **transferencia 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 analógica para operar un actuador. Muchos microcontroladores se han construido en convertidores análogos a digitales, por ejemplo; PIC 16C74/74A (Figura 13.30) y Motorola M68HC11 (Figura 13.10), así pueden manejar entradas análogas. Sin embargo, cuando se requieren salidas análogas, la salida del microcontrolador por lo general pasa a través de un convertidor externo análogo-digital (como ejemplo vea la sección 13.6.2).

13.3.1 Búfers

Un **búfer** es un dispositivo que se conecta entre dos partes de un sistema para evitar interferencias no deseadas entre las dos partes. Un uso importante de un búfer está en el puerto de entrada del microprocesador para aislar los datos de entrada desde el bus de datos del microprocesador hasta que el microprocesador lo requiera. El búfer de uso más común es un **búfer de tres estados**. El búfer de tres estados está habilitado por una señal de control para proveer salidas lógicas de 0 o 1, cuando no está habilitado tiene una impedancia alta y por lo tanto desconecta los circuitos de manera efectiva. La Figura 13.2 muestra los símbolos para los búfer de tres estados y las condiciones bajo las cuales cada uno está habilitado. La Figura 13.2a) y b) muestran el símbolo para búfer que no cambian la lógica de la entrada y la Figura 13.2c) y d) para los búfer que lo hacen.



Habilitar	Entrada	Salida
0	0	Alta impedancia
0	1	Alta impedancia
1	0	0
1	1	1

a)

Habilitar	Entrada	Salida
0	0	0
0	1	1
1	0	Alta impedancia
1	1	Alta impedancia

b)

Habilitar	Entrada	Salida
0	0	Alta impedancia
0	1	Alta impedancia
1	0	1
1	1	0

c)

Habilitar	Entrada	Salida
0	0	1
0	1	0
1	0	Alta impedancia
1	1	Alta impedancia

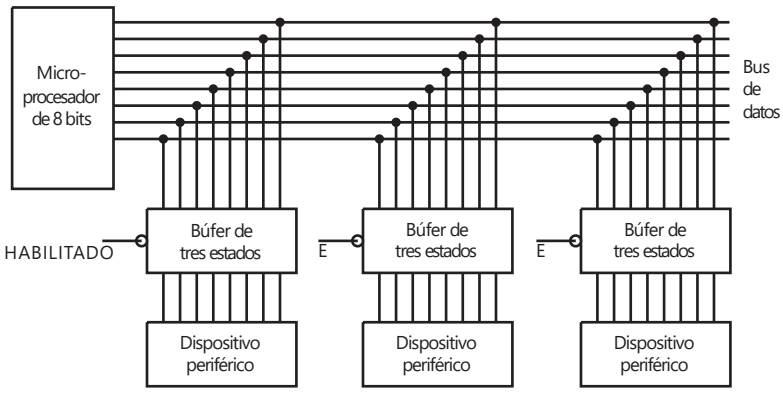
d)

Figura 13.2 Búfers: a) ningún cambio lógico, habilitado por 1, b) ningún cambio lógico, habilitado por 0, c) cambio lógico, habilitado por 1, d) cambio lógico, habilitado por 0.

Con microcontroladores PIC (sección 10.3.3), el bit TRIS está conectado a la entrada habilitada de un búfer de tres estados. Si el bit es 0, el búfer de tres estados está habilitado y sencillamente pasa su valor de entrada a su salida, si es 1 el búfer de tres estados está deshabilitado y la salida se vuelve de alta impedancia (como en la Figura 13.2b).

Estos búfer de tres estados se utilizan cuando una cantidad de dispositivos periféricos tienen que compartir las mismas líneas de datos desde el microprocesador; es decir, están conectados al bus de datos, y así hay una necesidad para que el microprocesador sea capaz para activar sólo uno de los dispositivos cuando los otros se deshabilitan. La Figura 13.3 muestra qué tanto se pueden utilizar los búfer. Estos búfers están disponibles como circuitos integrados; por ejemplo, el 74125 con cuatro búfers no inversores tipo activo bajo y el 74126 con cuatro búfer no inversores tipo activo alto.

Figura 13.3 Búfer de tres estados.



13.3.2 Reconocimiento

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**. El dispositivo periférico manda una señal de DATOS LISTOS a la sección de entrada/salida. La CPU entonces determina que esa señal está activa. Luego la CPU lee los datos desde la sección entrada/salida y envía una señal de RECONOCIMIENTO DE ENTRADA al dispositivo periférico. Esta señal indica que se ha completado la transferencia y de esta manera el dispositivo periférico puede enviar más datos. Para una salida, el periférico envía una señal de REQUERIMIENTO DE SALIDA o PERIFÉRICO LISTO a la sección de entrada/salida. La CPU determina que la señal PERIFÉRICO LISTO está activada y envía los datos al dispositivo periférico. La siguiente señal de PERIFÉRICO LISTO se puede utilizar para informar a la CPU que la transferencia se ha completado.

Con el microcontrolador MC68HC11, la operación básica de entrada/salida muestreada consiste en lo siguiente. Para las señales de control de reconocimiento se usan las terminales STRA y STRB (Figura 13.4a), vea también la Figura 10.10 para 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 lo 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

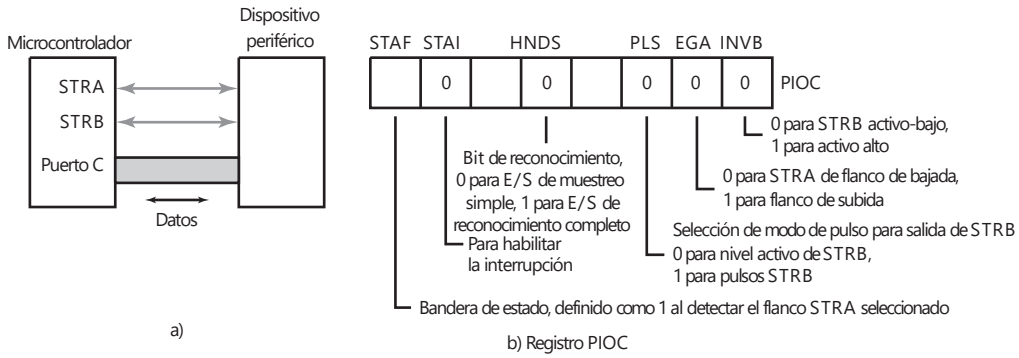
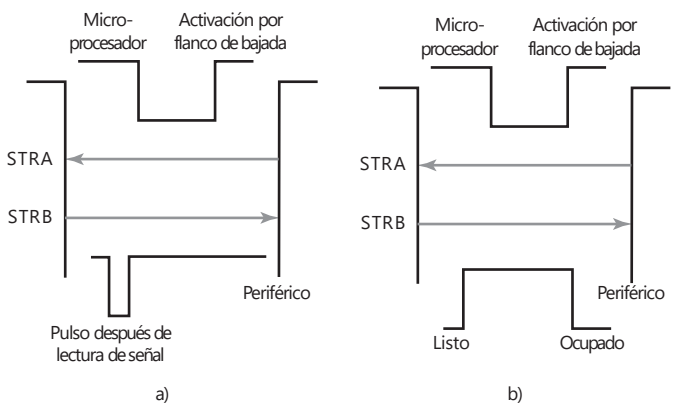


Figura 13.4 Control de reconocimiento: muestreo y reconocimiento.

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 en paralelo PIOC en la dirección $1002 debe ser el primero que se configure. La Figura 13.4b) ilustra los estados necesarios para los bits que están en ese registro.

La **entrada/salida con reconocimiento completo** consiste en el envío 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 13.5).

Figura 13.5 Reconocimiento completo: a) tipo pulsado, b) tipo asegurado.



13.3.3 Poleo e interrupciones

Suponga una situación en la que todas las transferencias de entrada/salida de datos se controlan en un programa. Cuando los 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 de forma continua, con saltos a rutinas de servicio 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 (Figura 13.6).

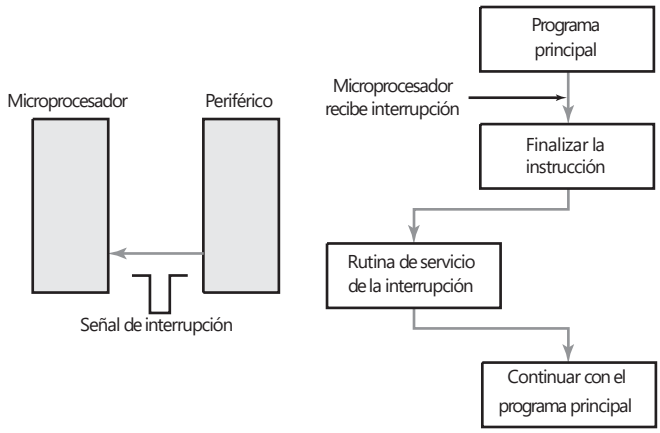


Figura 13.6 Control de interrupciones.

De este modo, cuando ocurre una interrupción:

1 La CPU espera a que termine la instrucción que está ejecutando antes de manejar la interrupción.
2 Todos los registros de la 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 La 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**. 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 de la 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 tedado 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, la 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 (Figura 13.26), 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 activar todas las interrupciones externas o se hace 0 para desactivar. El registro TCON (Figura 13.25) se usa para determinar el tipo de señal entrada de interrupción que inicializará una interrupción.

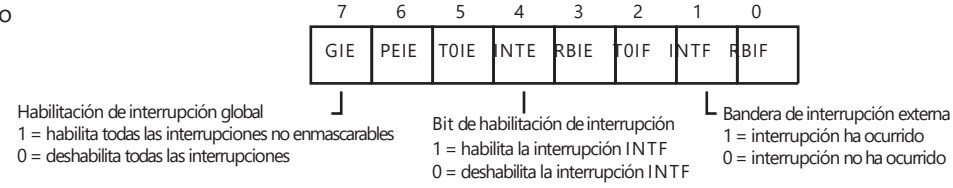


Figura 13.7 Registro INTCON.

Con los microcontroladores PIC, las interrupciones se controlan mediante el registro INTCON (Figura 13.7). 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 (Figura 13.32); 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.

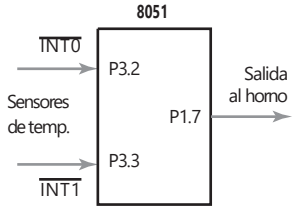


Figura 13.8 Sistema de calefacción central.

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 13.8). 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, por ejemplo, 20.5º C y el otro cuando sube más de 21.0º C. El sensor para la temperatura de 21.0º C se conecta a la interrupción INT0, puerto 3.2, y el sensor para la temperatura de 20.5º 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 21.0º 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 20.5º 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 activar las interrupciones, establecer las condiciones iniciales de que el horno esté encendido si la temperatura es menor que 21.0º C o esté apagado si es mayor, y entonces espera sin hacer 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 activar la interrupción externa 0
        SETB    EX1         ; para activar 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 0
        SETB    P1.7        ; enciende el horno
        JB      P3.2,HERE   ; si la temperatura es mayor que 21.0° C
                            ; 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 M68HC11 tiene un temporizador vigilante sincronizador de controlador de secuencia para el adecuado funcionamiento de la computadora (COP), que detecta errores en el procesamiento del software cuando la CPU no ejecuta ciertas secciones de código dentro del lapso asignado. Si 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.

13.3.4 Interfaz 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** el receptor y 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 13.9). En este modo de transmisión, en general el transmisor y el receptor son remotos (el Capítulo 15 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.

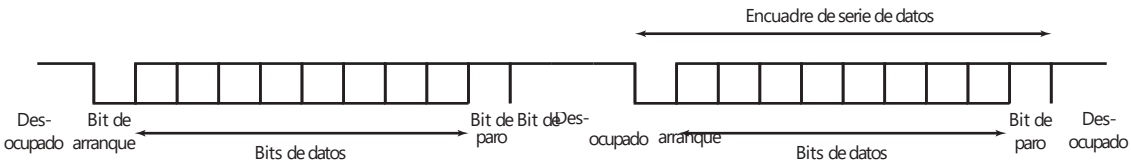


Figura 13.9 Transmisión asíncrona.

El microcontrolador MC68HC11 (Figura 13.10) tiene una interfaz para comunicaciones en serie (SCI) 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 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 interfaz para dispositivo periférico en serie (SPI) para la transmisión síncrona. Se utiliza en comunicaciones en serie locales; locales significa comunicaciones dentro de la máquina en donde se encuentra el chip.

13.4 Adaptadores de interfaz para dispositivos periféricos

Es posible diseñar interfaces para entrada/salida específicas; pero también existen dispositivos para interfaces 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 interfaz para periféricos** (PIA).

Una interfaz 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 esencia de dos puertos de entrada/salida en paralelo, con su lógica de control para conectarse con el microprocesador principal. La Figura 13.10 muestra la configuración básica del PIA MC6821, y las conexiones.

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

1 Un *registro de interfaz para periférico*. El funcionamiento de un puerto de salida difiere del de entrada pues debe guardar los datos para el 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.

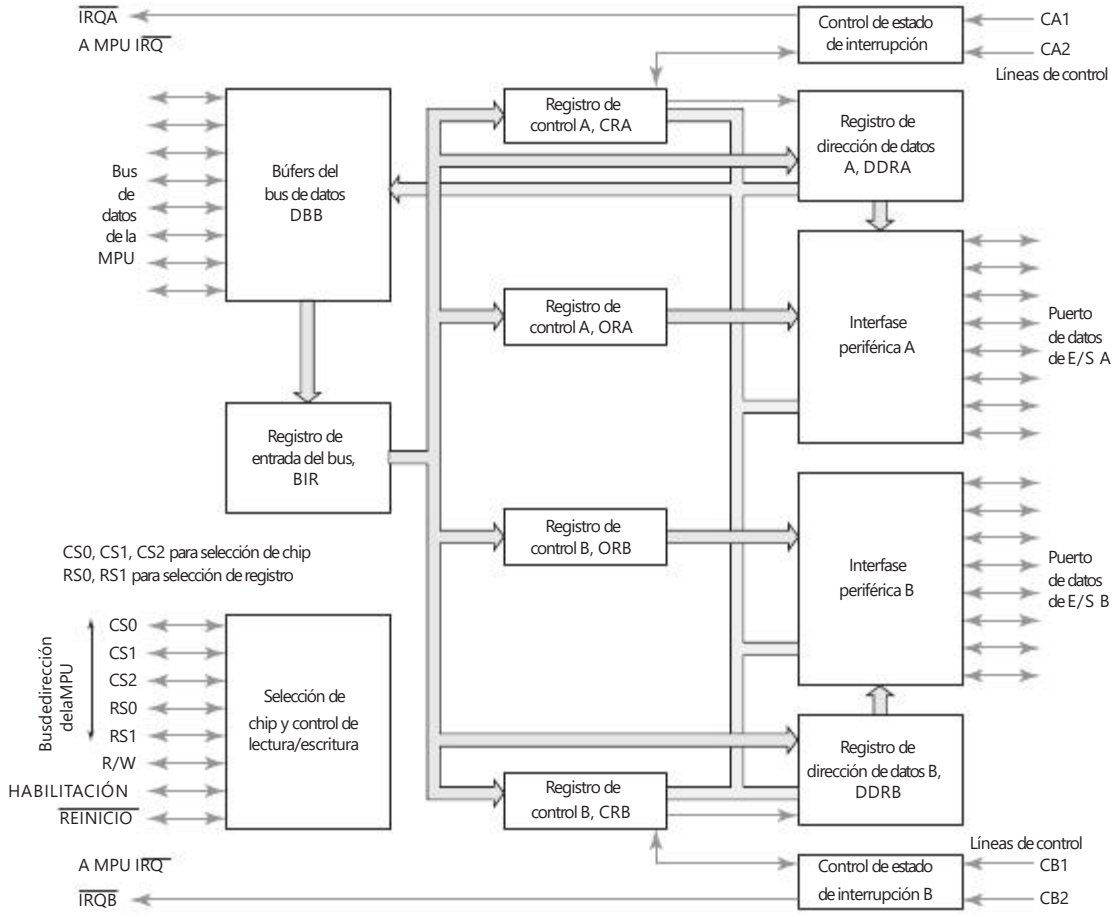


Figura 13.10 PIA MC6821.

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 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 al lado B cuando es alto. RS0 direcciona los registros a un lado en particular, ya sea 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 13.11. En el registro de control B se utiliza una configuración similar.

Figura 13.11 Registro de control.

B7	B6	B5	B4	B3	B2	B1	B0
IRQA1	IRQA2		Control CA2		DDRA Acceso	Control CA1	

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 que el indicador de interrupción IRQA(B), bit B6, se defina 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 selección 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.

13.4.1 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 ceros 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 0 en los demás bits
        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 del dispositivo periférico al puerto de salida.

13.4.2 Conexión de señales de interrupción a través del PIA

El PIA MC6821 de Motorola (Figura 13.12) 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 $04 en el registro de control. La forma de modificación dependerá del tipo de cambio de la entrada requerida para iniciar la interrupción.

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

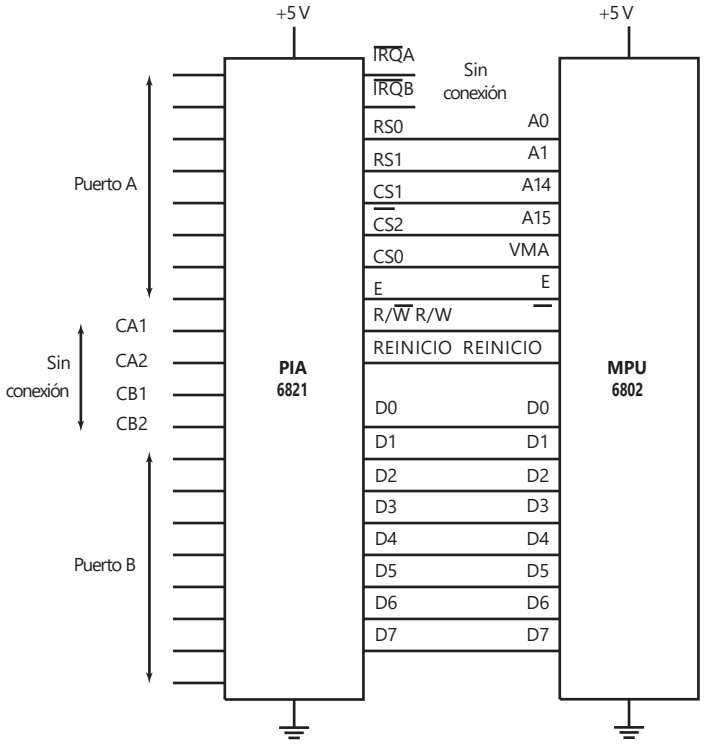


Figura 13.12 Acoplamiento mediante interfaz con un PIA.

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.
B3, B4 y B5 son 0 porque CA2 está desactivado.
B6 y B7 son indicadores sólo de lectura, por lo que se usan 0 o 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.

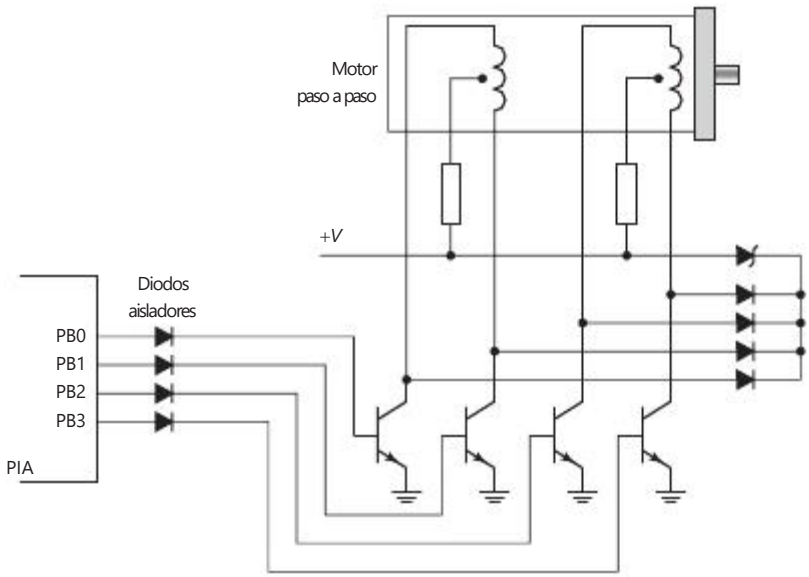
Por lo tanto, el formato para CA1 sería 00110100, es decir 34 en notación hexadecimal. El programa de inicialización sería:

```
INIT    LDAA    #$00    ; Carga 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    #$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
```

13.4.3 Ejemplo de conexión de una interfaz con un PIA

La Figura 13.13 es un ejemplo de conexión de una interfaz con un PIA: en ella se muestra un circuito que se usa para un motor paso a paso unipolar (sección 9.7.2). Al conectar los devanados inductivos se puede generar una fuerza contraelectromotriz 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 interfaz sencilla y barata; en cambio, las resistencias no aíslan por completo el PIA.

Figura 13.13 Acoplamiento mediante interfaz con un motor paso a paso.



13.5 Interfaz para comunicaciones en serie

El **receptor/transmisor asíncrono universal** (UART) 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 interfaz para comunicaciones asíncronas** (ACIA) MC6850 de Motorola. La Figura 13.14 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.

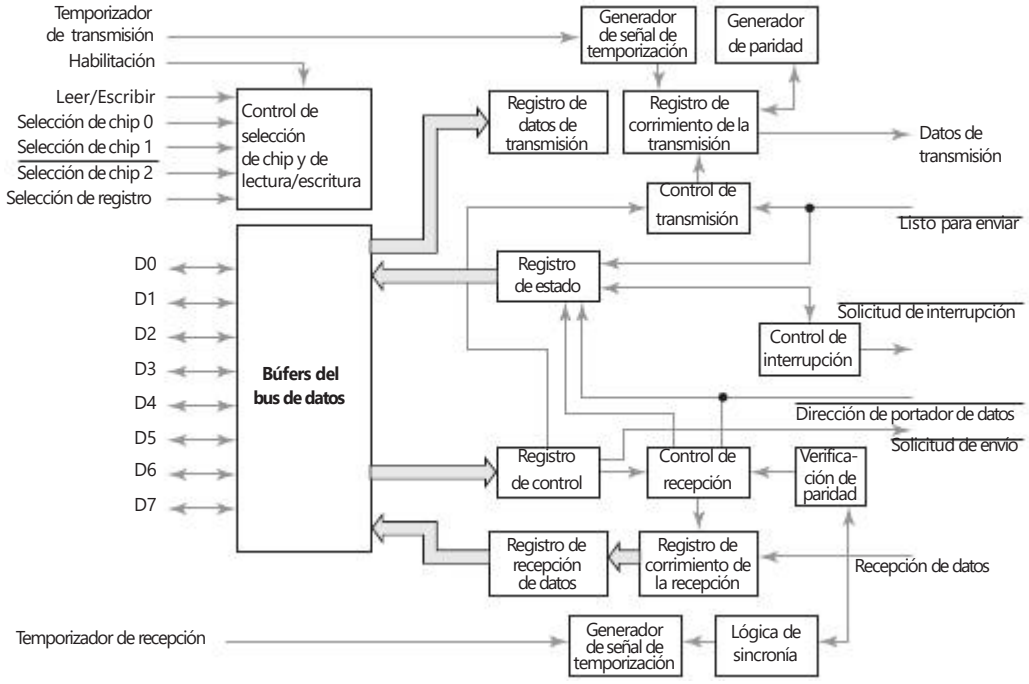


Figura 13.14 ACIA MC6850.

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 13.15 y 13.16 muestran, respectivamente, los formatos de bit de los registros de control y de estado.

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 el Capítulo 15 para más detalles).

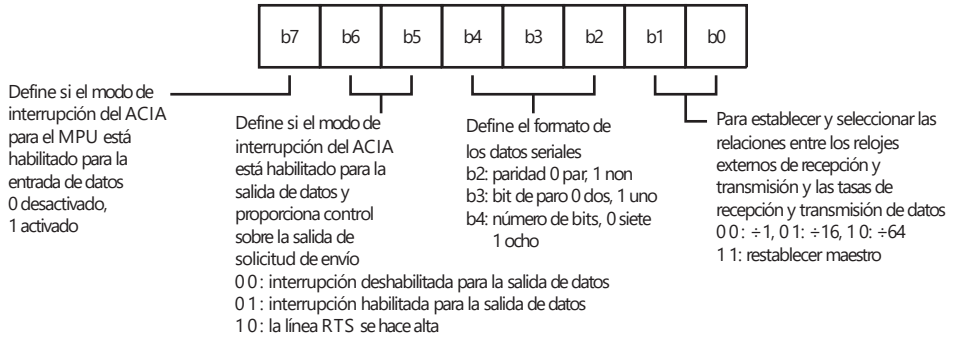
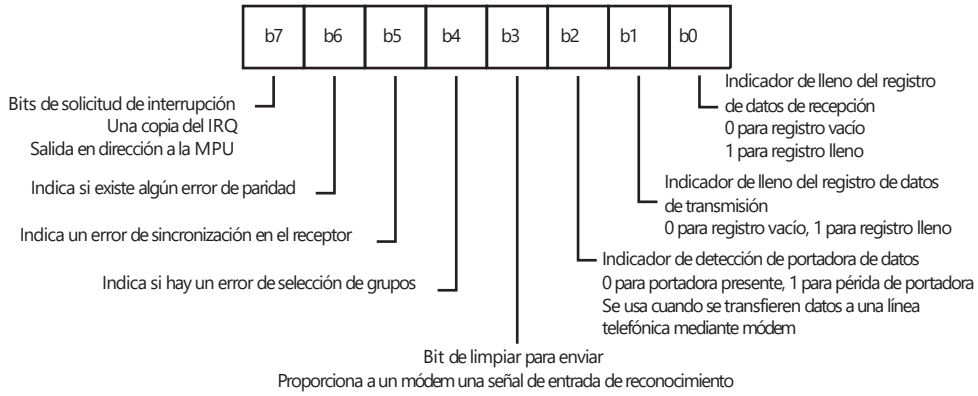


Figura 13.15 Registro de control.

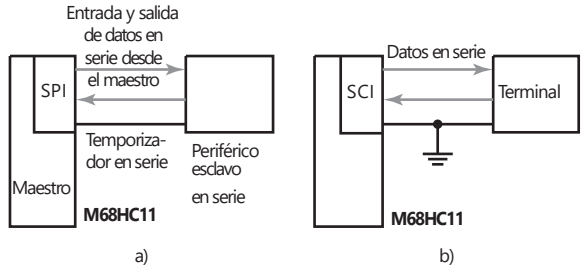
Figura 13.16
Registro de estado.



13.5.1 Interfaces en serie de microcontroladores

Muchos microcontroladores tienen interfaces en serie, es decir UART integrados. Por ejemplo, el M68HC11 tiene una interfaz para periféricos en serie (SPI), una interfaz síncrona y una interfaz para comunicaciones en serie (SCI), que es una interfaz asíncrona (Figura 13.10). La SPI requiere la misma señal de sincronización que usan el microcontrolador y el dispositivo o dispositivos que se conectan en forma externa (Figura 13.17a). Es posible conectar a la SPI varios microcontroladores. La SCI es una interfaz asíncrona, y por ello es posible utilizar diferentes señales de sincronización entre su sistema y el dispositivo que se conecta de manera externa (Figura 13.17b). Los microprocesadores para propósito general no cuentan con interfaz 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 interfaz de comunicaciones en serie, y es necesario complementar el microcontrolador M68HC11 con una UART.

Figura 13.17 a) SPI, b) SC1.



La SPI se inicializa por los bits del registro de control de la SPI (SPCR) y 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. La 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 de la SCI.

El Intel 8051 cuenta con una interfaz serial integrada con cuatro modos de operación, éstos se seleccionan al escribir unos o ceros dentro de los bits SMO y SMI en el registro SCON (control de puerto serial) en la dirección 98H (Figura 13.18 y Tabla 13.1).

En el modo 0, los datos en serie entran y salen por RXD. La terminal TXD sale del reloj de cambio lo que luego se usa para sincronizar la transmi-

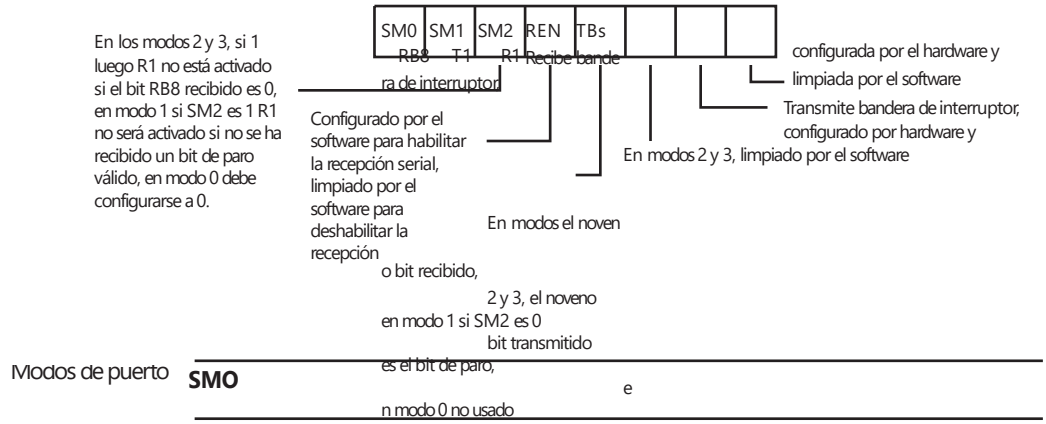


Figura 13.18 Registro SCON.

Tabla 13.1
en serie Intel 8051.

	SM1	Modo	Descripción	Velocidad en baudios
0	0	0	Registro de deslizamiento	Frecuencia de oscilación/12
0	1	1	UART de 8 bits	Variable
1	0	2	UART de 9 bits	Frecuencia de oscilación/12 o 64
1	1	3	UART de 9 bits	Variable

sión y recepción de datos. La recepción de datos se inicia cuando REN está en 1 y R1 es 0. La transmisión se inicia cuando cualquier dato se escribe para el SBUF, esto si es el búfer de puerto serial en la dirección 99H. En el modo 1 se transmiten 10 bits en TXD o se reciben en RXD, éstos son el bit de arranque de 0, los ocho bits de datos y un bit de paro de 1. La transmisión comienza al escribir a SBUF y la recepción con la transición de 1 a 0 en RXD. En modos 2 y 3, 11 bits se transmiten en TXD o se reciben en RXD.

Los microcontroladores PIC tienen un SP1 (Figura 13.30) que se puede utilizar para comunicaciones en serie en sincronía. Cuando los datos se escriben para el registro SSBUF se deslizan afuera de la terminal SDO en sincronía con una señal de reloj en SCK y salen a través de la terminal RC5 como señal de serie con el bit más importante en primer lugar y una señal de reloj a través de RC3. La entrada dentro del registrador SSBUF es vía RC4. Muchos microcontroladores PIC tienen también un UART para crear una interfaz serial para utilizarla con datos seriales transmitidos de manera asincrónica. Al transmitirlos, cada byte de 8 bits es enmarcado por un bit de ARRANQUE y uno de PARO. Cuando se transmite el bit de ARRANQUE, la línea RX cae hacia una transición de bajo y el receptor entonces hace la sincronización en esta transición de alto a bajo. Luego el receptor lee los 8 bits de datos en serie.

Ejemplos de acoplamiento mediante interfaz

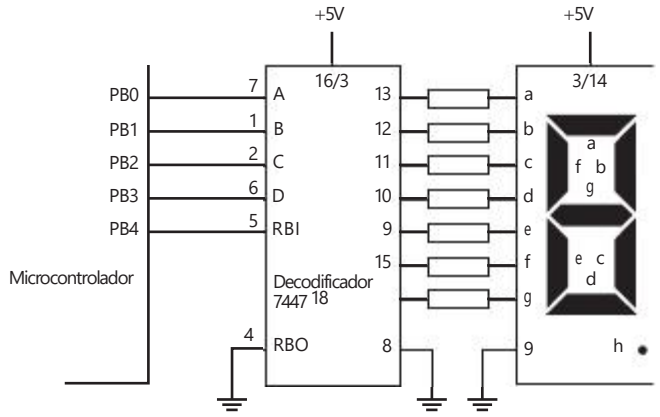
Los siguientes son ejemplos de acoplamientos mediante interfaces.

13.6.1 Acoplamiento mediante interfaz 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 (sección 6.5). Un LED es un indicador de apagado-encendido; el número que aparezca en el visualizador dependerá

de cuáles LED estén encendidos. La Figura 13.19 muestra cómo usar un microcontrolador para activar un visualizador de ánodo común utilizando un

Figura 13.19 Manejo de un visualizador.



controlador de decodificador; 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 13.2 muestra las señales de entrada y salida del decodificador.

Tabla 13.2 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

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 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 13.20).

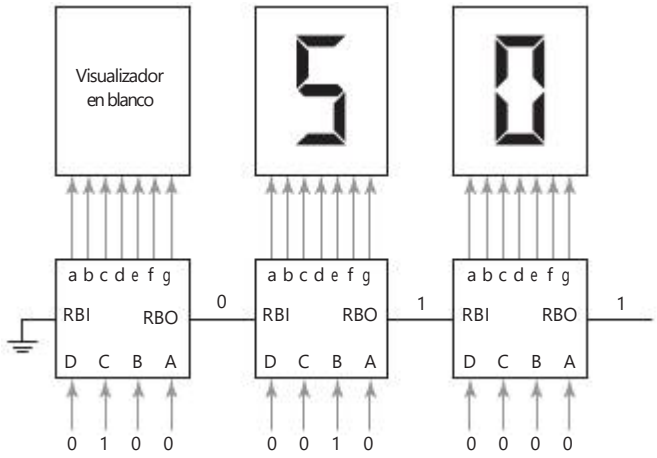


Figura 13.20 Puesta en blanco del acarreo.

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 13.21 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 consecuencia 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.

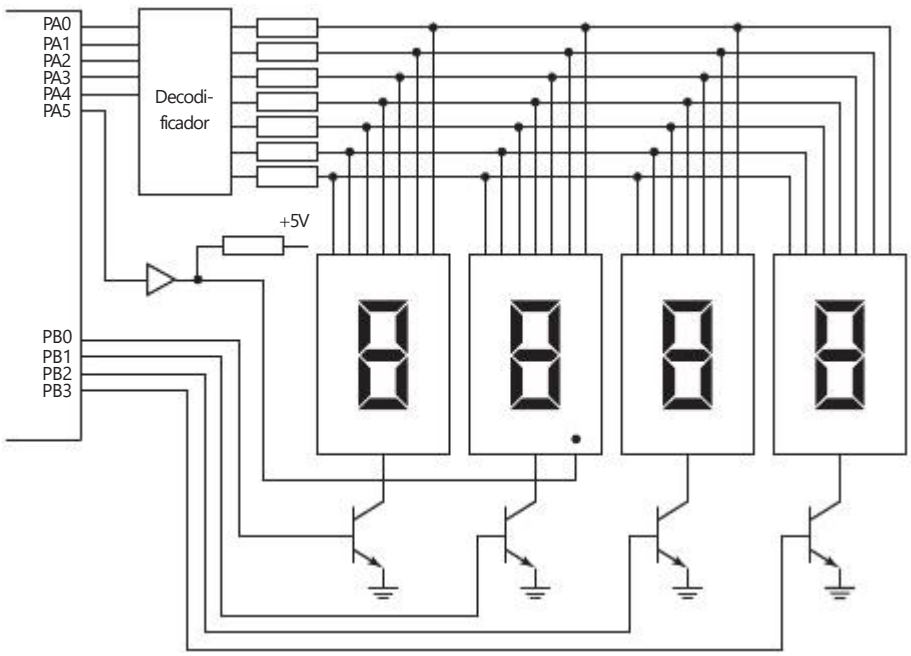


Figura 13.21 Multiplexado de cuatro visualizadores.

13.6.2 Acoplamiento mediante interfaz 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 interfaz 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.

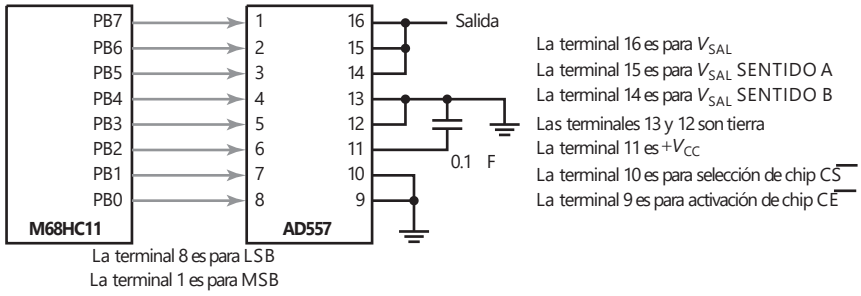


Figura 13.22 Generación de formas de onda.

La Figura 13.22 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 DAC
AGAIN   INC  PORTB,X    ; Incrementar en 1
        BRA  AGAIN      ; Repetir
        END
```

Resumen

Los **requerimientos de interfaz** a menudo significan **acoplamiento mediante búfer/aislamiento eléctrico**, control de temporización, conversión de código, cambio del número de líneas, transferencia de datos en serie a paralelo y viceversa, conversión de análogo a digital y viceversa. No es necesario un **reconocimiento** (handshaking) a menos que dos dispositivos puedan enviar y recibir datos a velocidades idénticas.

El **poleo** es el control del programa de entradas/salidas en el cual se utiliza un ciclo de manera continua para leer las entradas y actualizar las salidas, con saltos para servir a las rutinas como se requiere; es decir, un proceso de revisión repetitiva de cada dispositivo periférico para verificar si está listo para enviar o aceptar un nuevo byte de datos. Una alternativa para el control de

programa es el **control de interrupciones**. Una interrupción implica un dispositivo periférico que activa una línea requerida de interrupción separada. La recepción de una interrupción da como resultado la suspensión de ejecución en el microprocesador de su programa principal y salta a la rutina de servicio para el periférico. Después de la rutina de servicio de interrupción, los contenidos de la memoria son restaurados y el microprocesador puede continuar ejecutando el programa principal desde donde fue interrumpido.

Existen dos tipos básicos de transferir datos en serie: asíncrona y síncrona. Con la **transmisión asíncrona**, el receptor y el transmisor utilizan sus propias señales de reloj de manera que no es posible que un receptor reconozca cuándo empieza o termina una palabra. De esta manera, es necesario que cada palabra de datos transmitido lleve su propio arranque y paro de sus bits de tal forma que sea posible que el receptor indique cuándo se detiene una palabra y empieza otra. Con la **transmisión síncrona**, el transmisor y el receptor tienen una señal de reloj común y así la transmisión y la recepción se pueden sincronizar.

Los **adaptadores de interfaz para periféricos** (PIA) son dispositivos programables de interfaz de entrada/salida que permiten diferentes tipos de opciones de entrada/salida para ser seleccionadas mediante el software.

El **receptor/transmisor asíncrono universal** (UART) es el elemento esencial de un sistema de comunicación en serie, su función es cambiar los datos en serie a paralelos para la entrada y los datos paralelos a serie para la salida. Una forma programable común de un UART es el **adaptador de interfaz para comunicación asíncrona** (ACIA).

Problemas

13.1 Describa las funciones que puede realizar una interfaz.

13.2 Explique la diferencia entre una interfaz en paralelo y una interfaz en serie.

13.3 Explique qué se entiende por un sistema de mapeo de memoria para entradas/salidas.

13.4 ¿Cuál es la función de un adaptador de interfaz periférico (PIA)?

13.5 Describa la arquitectura del PIA MC6821 de Motorola.

13.6 Explique la función del programa de inicialización de un PIA.

13.7 ¿Qué ventajas ofrece utilizar las interrupciones externas en vez del muestreo por software como medio de comunicación con dispositivos periféricos?

13.8 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.

13.9 Escriba un programa en lenguaje ensamblador para inicializar el PIA MC6821 de Motorola, de manera que se cumplan las especificaciones del problema 13.8.

13.10 Escriba un programa en lenguaje ensamblador para inicializar el PIA MC6821 de Motorola, de manera que lea ocho bits de datos del puerto A.

Capítulo catorce Controladores lógicos programables

Objetivos

Después de estudiar este capítulo, el lector debe ser capaz de:

- **Describir la estructura básica de los PLC y su operación.**
- **Desarrollar programas de escalera para un PLC que involucran funciones lógicas, cierre, relevadores internos y secuenciación.**
- **Desarrollar programas que involucran temporizadores, contadores, registros de cambio, relevadores maestros, saltos y manejo de datos.**

14.1 Controladores lógicos programables

Un **controlador lógico programable** (PLC) es un dispositivo electrónico digital 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 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 9.2) 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 interfaz para las entradas y las salidas está dentro del controlador.
3 Es muy fácil programarlos.

14.2 Estructura básica del PLC

La Figura 14.1 muestra la estructura interna básica de un PLC que, en esencia, consiste en una unidad central de procesamiento (CPU), memoria y circuitos de entrada/salida. La CPU controla y procesa todas las operaciones dentro

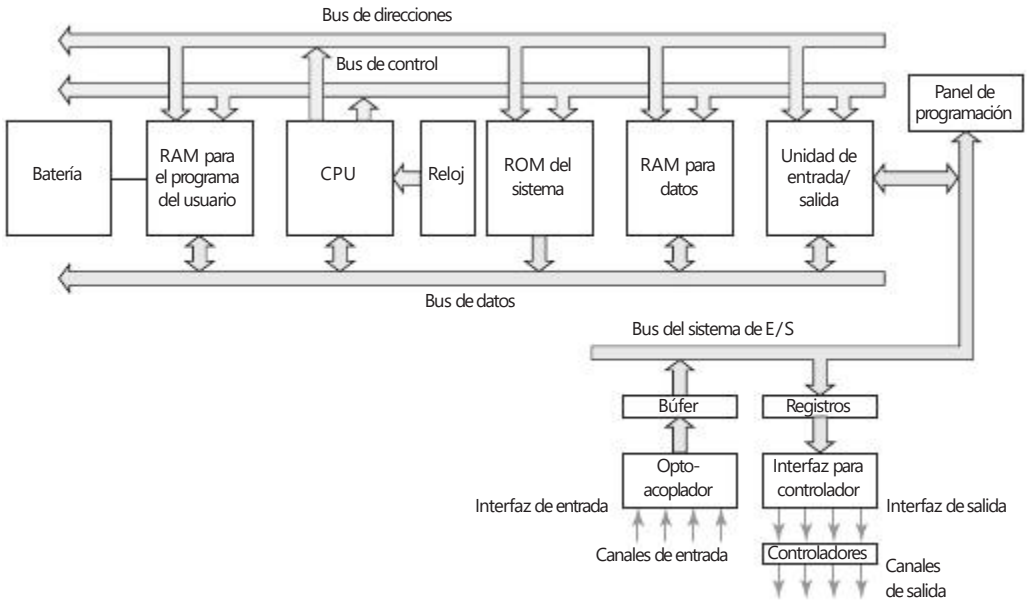


Figura 14.1 Arquitectura de un PLC.

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 búfer temporal para los canales de entrada/salida.

14.2.1 Entrada/salida

La unidad de entrada/salida es la interfaz entre el sistema y el mundo externo y donde el procesador recibe información desde dispositivos externos y comunica información a dispositivos externos. Las interfaces de entrada/salida ofrecen aislamiento y funciones de acondicionamiento de señal de manera que esos sensores y actuadores a menudo pueden conectarse directamente a ellos sin necesitar otro circuito. Las entradas pueden estar desde interruptores límite que se activan al presentarse algún evento, u otros sensores como sensores de temperatura o sensores de flujo. Las salidas pueden servir para activar las bobinas de arranque, válvulas solenoides, etc., de un motor. El aislamiento eléctrico del mundo externo por lo general es por medio de optoaisladores (sección 3.3).

La Figura 14.2 muestra la forma básica de un canal de entrada. La señal digital que por lo general es compatible con el microprocesador en el PLC es de 5 V de c.d. Sin embargo, el acondicionamiento de señal en el canal de entrada, con aislamiento, permite un rango amplio de señales de entrada para suministrarlo. Por lo tanto, con un PLC más grande se podrían tener voltajes de entrada posibles de 5 V, 24 V, 110 V y 240 V. Un PLC pequeño es probable que tenga sólo una forma de entrada, por ejemplo, 24 V.

La salida para la salida de la unidad será digital con un nivel de 5 V. Las salidas se especifican como tipo de relevador, tipo transistor o tipo triac. Con el tipo de