LABORATORIO DE COMPUTADORAS

2019

CAPÍTULOS – HERBERT TAUB

Temas:

- **Computadora sencilla.**
- **Computadora básica o mejorada.**
- **Microprogramación.**
- **Ejemplos de aplicación.**

Fuente:

Circuitos Digitales y Microprocesadores

Editorial: Mc Graw Hill

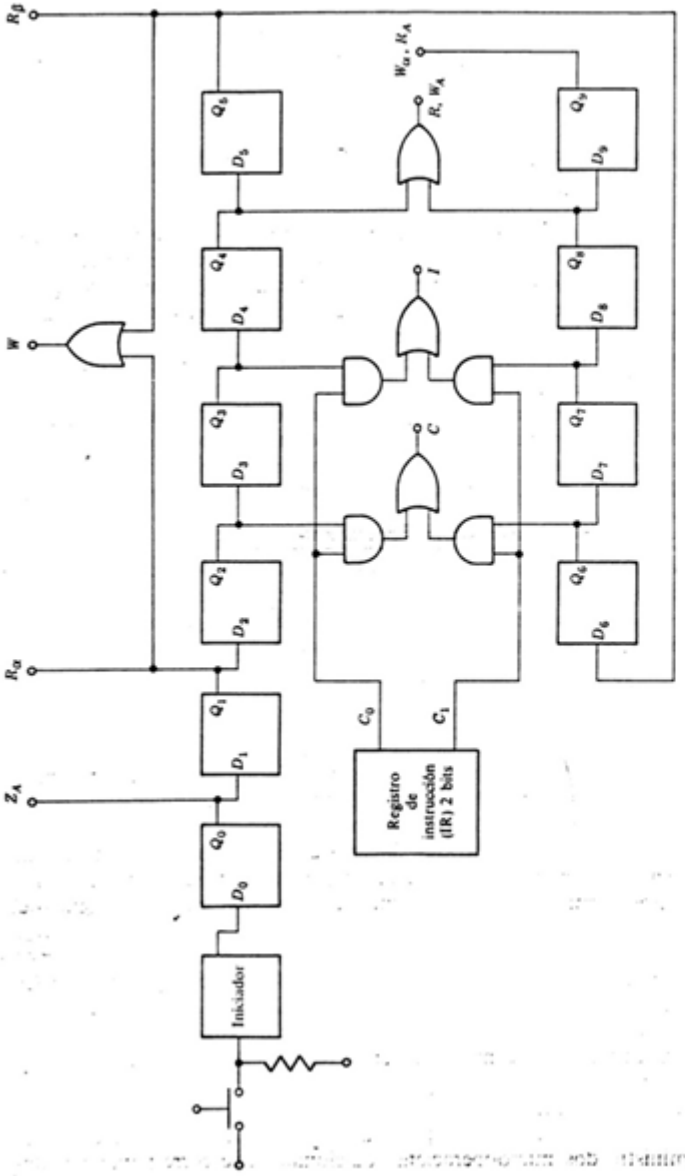


Figura 8.8-1 Controlador de registro de desplazamiento utilizado con la arquitectura de la figura 8.4-1. El controlador colocará en el registro R_x una de las cuatro cantidades $\pm R_\alpha$ y $\pm R_\beta$, dependiendo de la instrucción del registro de instrucciones.

Instrucción		Operación total
C_1	C_0	
0	0	$R_\alpha + R_\beta$
0	1	$-R_\alpha + R_\beta$
1	0	$R_\alpha - R_\beta$
1	1	$-R_\alpha - R_\beta$

Figura 8.8-2 El código de instrucción C_1C_0 para las cuatro instrucciones del controlador de la figura 8.8-1.

estados adicionales. Si diseñamos un controlador de estados mínimo, la tabla de estados tendrá dos estados adicionales; como nuestro controlador adicional tiene siete estados, el nuevo controlador tendrá nueve y se necesitarán cuatro flip-flops. Además, tendremos que cambiar la lógica que lleva a la secuencia a través de sus pasos y tendremos que añadir lógica al decodificador. Si diseñamos un controlador de registro de desplazamiento tendremos que añadir dos flip-flops a la cadena de los de dicho registro. Los detalles de cada uno de estos diseños se dejan como ejercicio al estudiante.

A continuación supongamos que queremos formar $-R_x+R_\beta$. Entonces, comenzando nuevamente con el controlador original de la figura 8.5-2 o de la figura 8.6-2, podríamos añadir dos microoperaciones, esta vez para formar el opuesto del número de R_x. Si quisiésemos $-R_x-R_\beta$, tendríamos que añadir cuatro microoperaciones.

Debíamos también decidir que son realmente innecesarios cuatro controladores diferentes para realizar las cuatro operaciones R_x+R_β, R_x-R_β, $-R_x+R_\beta$ y $-R_x-R_\beta$. En vez de ello, debemos diseñar un controlador sencillo que pueda realizar alguna de las cuatro operaciones dependiendo de la *instrucción* que le demos. Dicho controlador se da en la figura 8.8-1. Con el propósito de mantener la *instrucción* hemos añadido un *registro de instrucción* de 2 bits. Aquí consideramos que la instrucción se coloca en el registro de instrucción manipulando los conmutadores, justo como los números que van a ser combinados; es decir, los operandos en los registros R_x y R_β. También hemos añadido cuatro estados adicionales al controlador original de la figura 8.6-2. En estos estados, los números del registro CI serán complementados e incrementados para cambiar el signo de los operandos. La complementación en el registro CI se realiza cuando el terminal C de la figura 8.8-1 (conectado a C en la figura 8.4-1) adopta el 1 lógico. La incrementación se realiza cuando el terminal I adopta el 1 lógico. Cuando las líneas de los bits de instrucción son $C_0=C_1=0$, se inhabilitan todas las puertas AND de la figura 8.8-1, C e I están siempre en 0 lógico y el resultado neto es que la operación total realizada genera la suma R_x+R_β. La operación total para las cuatro instrucciones posibles se da en la figura 8.8-2.

8.9 UNA COMPUTADORA SENCILLA

El *controlador* de la figura 8.8-1 operando en conjunción con los *registros de almacenamiento y controlables* organizados en la *arquitectura* de la figura 8.4-1

nos permitirá combinar aritméticamente dos números mediante suma y resta. Para utilizar la máquina pondremos los números en R_α y R_β, presumiblemente a mano mediante conmutadores, ya que no hemos hecho otras previsiones. Entonces podríamos poner manualmente una instrucción en el registro de instrucciones (IR). Después de eso, todo se realiza automáticamente. Solamente necesitamos pulsar el botón de comenzar y después de un rato encontraríamos nuestro resultado en R_α.

Supongamos ahora que queremos hacer nuestra máquina más elaborada. Primero queremos poder tratar con más de dos números u operandos. Así podemos realizar una suma de esos operandos pero seleccionándolos entre un gran número. O, disponiendo un gran número de operandos, podemos combinar muchos. Una modificación obvia que se ocurre entonces es reemplazar los registros R_α y R_β por un gran array de registros. Dicho array es, por supuesto, una memoria como la descrita en el Capítulo 6. Si queremos cambiar fácilmente los operandos necesitaremos una RAM.

En el controlador de la figura 8.8-1 una serie de pasos de secuencia tratan a los operandos almacenados en R_α y R_β. Es claro que este modelo nos conduciría a una secuencia muy grande si hubiese muchos operandos. Cada nuevo operando añade pasos a la secuencia. Sin embargo, reconocemos que cada operando está realmente sujeto a las mismas operaciones. El operando se transfiere desde un registro de almacenamiento (o mejor desde una posición de memoria) al registro CI y de ahí a través del sumador al acumulador. Si necesitamos cambiar el signo del operando, complementamos e incrementamos en el registro CI. En otra situación, este registro no hace nada. Esta secuencia de microoperaciones se repite de nuevo para cada operando. Por tanto, parece que realmente nos podemos arreglar con un controlador que tenga una secuencia corta que trate exactamente un operando. Sin embargo, con dicho controlador de secuencia necesitaríamos algún mecanismo para ajustar cada nuevo operando que se vaya a tratar. Una forma de ajustar la instrucción es, realmente, detener la secuencia de control después de tratar cada operando y entonces manualmente cambiar la instrucción antes de permitir a la secuencia tratar al siguiente operando. Pero ya que tenemos memoria, podemos almacenar en ella la información relativa a como se trata cada operando y la operación completa puede hacerse automáticamente. Con estas consideraciones en cuenta volvamos ahora a la figura 8.9-1, que visualiza un sistema que (con alguna intervención manual) nos permitirá combinar aritméticamente un gran número de operandos automáticamente.

El sistema tiene una memoria RAM. Para ser específicos, supongamos una memoria de 64 palabras, cada una de 8 bits. Una posición de memoria se direcciona con 6 bits ($2^6 = 64$). La memoria tiene una entrada de *habilitación* y otra de *lectura/escritura*. Cuando la *habilitación* = 1, la memoria se conecta al bus (de 8 bits) y cuando la *habilitación* = 0, el bus se aisla de la memoria. Cuando la *habilitación* es = 1, la memoria leerá una palabra en el bus o escribirá una palabra en memoria, dependiendo de que *lectura/escritura* sea 1 ó 0. La posición de memoria de la que se lee o en la que se escribe una palabra

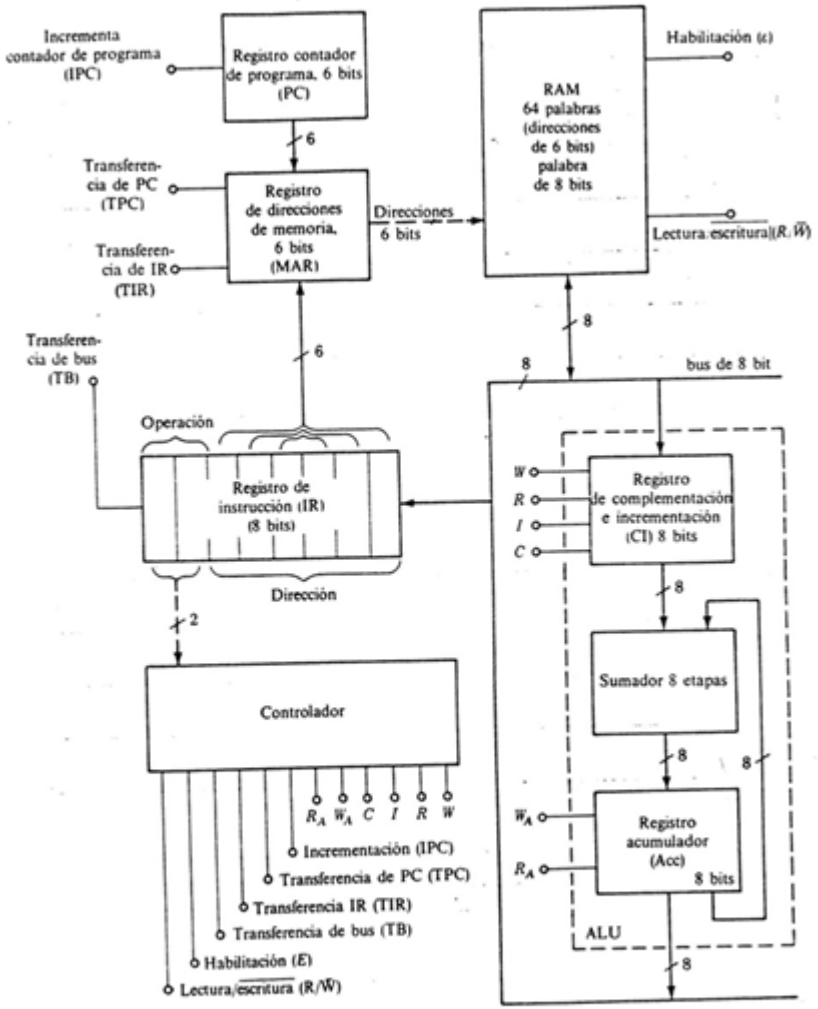
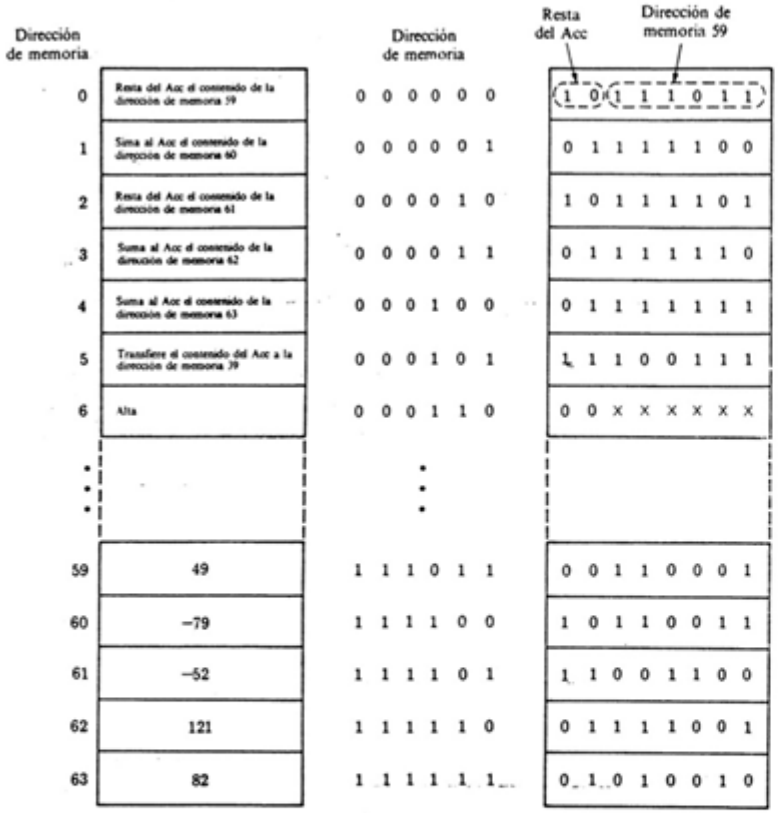


Figura 8.9-1 Arquitectura de una «computadora» que permitirá la combinación aritmética de un gran número de operandos.



Código	Instrucción
00	Alto
01	Suma al Acc
10	Resta del Acc
11	Transfiere el contenido del Acc a

(b)

Figura 8.9-2 (*a*) Posible contenido de la memoria de la figura 8.9-1. (*b*) Código de instrucción. (*c*) Contenido real de bits binarios de las posiciones de memoria.

se determina por los seis bits de dirección. La doble cabeza de flecha que conecta la memoria al bus indica que la transferencia de información entre el bus y memoria es bidireccional.

En la figura 8.9-2 ilustramos lo que típicamente debía estar contenido en la memoria de nuestra máquina sencilla. En la figura 8.9-2*a* hemos escrito en palabras y números decimales el contenido de algunas posiciones de memoria. En las posiciones 0 a 6 hemos escrito un *programa* de instrucciones. En el otro extremo de la memoria hemos almacenado algunos operandos. Si la máquina realizase las instrucciones dadas en el orden listado, entonces calcularía la cantidad $-(49)+(-79)-(-52)+(121)+(82)=+127$, como puede verse observando los contenidos de las posiciones de memoria 59 a 63. La máquina transferirá esta suma acumulada del registro acumulador a la posición 39 de memoria y entonces se parará y esperará a una intervención humana. Como veremos, hay un propósito al colocar las instrucciones en posiciones sucesivas de memoria. No es importante que las instrucciones y los operandos estén colocados en los extremos opuestos de la memoria. Lo único que se requiere es que las instrucciones se coloquen (en posiciones consecutivas) en una parte de la memoria y los operandos en otra parte. El orden indicando en la figura 8.9-2 se sugiere principalmente con el fin de realizar los cálculos ordenadamente. Las posiciones de memoria no indicadas en la figura tienen algún contenido. (Una posición de memória borrada con contenido 00...00 tiene un contenido sin nada.) Pero el contenido de estas posiciones de memoria no es relevante en nuestra discusión actual. Nos proponemos almacenar nuestro resultado en la posición 39 (no mostrada). El contenido de la posición 39 no se conoce y es irrelevante. Cuando la instrucción se ejecuta para que escriba el resultado en la posición 39, el contenido previo de esa posición se perderá.

Señalamos que hemos utilizado cuatro instrucciones: suma, resta, transferencia y parada. Arbitrariamente, utilizamos el código de 2 bits de la figura 8.9-2*b* para representar esas instrucciones. En la figura 8.9-2*c* hemos vuelto a escribir la memoria en forma binaria. En las posiciones que contienen operandos hemos reemplazado sencillamente los números decimales por binarios. En las posiciones de memoria que contienen instrucciones hemos dispuesto arbitrariamente que los dos bits de más a la izquierda representen la instrucción y los 6 bits restantes especifiquen la posición que contiene el operando. Esta asignación de significado a los bits se indica expresamente para la primera posición de memoria. Las posiciones de memoria 0 a 5 contienen instrucciones que realizan una operación y se refieren a una posición específica de memoria donde se almacena el operando. La posición de memoria 6 realiza una operación pero como no hay operando, es irrelevante la dirección de seis bits.

Observemos que las palabras de memoria tienen una longitud de 8 bits. Si nos proponemos representar números negativos en la forma del complemento a dos, el rango de los números puede acomodarse desde $+127$ a -128. Entonces debemos ser cuidadosos de asegurarnos cómo acumula nuestra máquina los números que estamos combinando; nunca se requerirá que la suma acumulada sobrepase el rango permitido. Los números introducidos en

las posiciones de memoria 59 a 63 se han seleccionado arbitrariamente excepto que hemos observado esta ligadura con respecto al rango.

Volviendo a la figura 8.9-1 señalamos que la sección en la caja a trazos es el sistema de la figura 8.4-1 son los registros R_α y R_β. (Estos registros se han sustituido por la parte de memoria que almacena los operandos.) Este pequeño sistema realiza aritmética (suma e incrementa) y lógica (complementación) y razonablemente se denomina unidad aritmética lógica (ALU), aunque difiere en muchos aspectos de la ALU de la figura 5.13-1. Por simplicidad hemos dejado fuera el terminal de *borrador* Z_A, ya que el acumulador puede borrarse empleando las instrucciones de la figura 8.9-2*b* (Problema 8.9-1).

Continuando nuestro examen de la figura 8.9-1 señalamos la presencia de un registro contador de programa (PC), registro de instrucción (IR), registro de direcciones de memoria (MAR) y, finalmente, el controlador que pone a nuestra pequeña computadora sobre sus pasos y que ya hemos diseñado. El propósito detallado de cada registro será discutido brevemente. De momento notemos simplemente los caminos disponibles para las transferencias de palabras (o partes de palabras) y las facultades que han sido incorporadas a cada registro. Señalamos que hay un bus de 8 bits al que la memoria tiene una conexión bidireccional. La ALU puede aceptar una palabra del bus sobre su lado de entrada y suministrar una palabra al bus de su lado de salida. El registro de instrucción puede aceptar una palabra del bus. El registro de direcciones de memoria tiene dos entradas de control que pueden utilizarse para transferir al MAR la palabra de 6 bits del contador de programa o los 6 bits de más a la derecha del registro de instrucción. Los dos bits de más a la izquierda del registro de instrucciones se hacen disponibles al controlador (no transferidos). Por consiguiente, esta conexión de dos bits se indica a trazos en vez de por una línea de salida. La única operación que puede realizar el contador de programa es la de incrementar. Finalmente, el controlador tiene una línea de salida de control correspondiente a cada línea de entrada de control de cada registro y de la memoria. Cualquier microoperación se realiza cuando la línea de control correspondiente adopte el nivel de habilitación. Con la excepción de la memoria, todos los registros y el controlador tienen reloj, y el momento actual en el que un registro acepta una transferencia y se incrementa en el instante de la transición de disparo de reloj. La línea de la señal de reloj que se distribuye a través de todo el sistema de la figura 8.9-1 no está indicada en el dibujo.

8.10 OPERACIÓN DE LA COMPUTADORA

Para ver cómo opera nuestra computadora consideremos que nuestra memoria se carga como en la figura 8.9-2*c*. Podemos imaginar que para efectuar esta carga estaba desconectada temporalmente del sistema y que las entradas (direcciones, datos, *habilitación*, *lectura/escritura*) se aplicaban manualmente. También suponemos que en principio el controlador de programa (PC) y el registro acumulador están borrados. (No debe importar si los

otros registros están borrados inicialmente.) Ahora listaremos en las tablas 8.10-1 y 8.10-2 cada ciclo de reloj, la secuencia de microoperaciones a través de las cuales el controlador debe pasar al sistema para que éste tome nota de la primera instrucción, realice su intento y se prepare para la siguiente instrucción. Cuando es factible realiza más de una microoperación durante el transcurso de un ciclo de reloj, por ello tomaremos nota de esta característica con el fin de ahorrar tiempo. Especialmente se señala que las únicas operaciones especificadas en la tabulación son aquellas que estarán afectadas por el hecho de que una o más salidas de control del controlador van al nivel de habilitación.

Tabla 8.10-1 Ciclo de búsqueda

Ciclo de reloj	Descripción simbólica de la operación	Línea de control a habilitar
1. Transfiere el contenido del contador de programa al registro de direcciones de memoria	PC→MAR	TPC
2. Transfiere la instrucción direccionada (en la posición 000000) al registro de instrucción mediante: 1) habilitación de memoria al conectarla al bus; 2) poniendo *R/W* a 1 al leer memoria, y 3) transfiriendo la palabra del bus al registro de instrucción, incrementando el contador de programa preparándolo para llamar la siguiente instrucción cuando se ha completado la respuesta a la primera	M→IR («M» representa palabra de memoria direccionada) PC+1→PC	*E*, *R/W*, TB IPC

En las operaciones listadas en la tabla 8.10-1 hemos trasladado las primeras instrucciones de memoria al registro de instrucciones. Esta parte del ciclo de operaciones de la máquina se denomina *ciclo de búsqueda*. Ahora que la primera instrucción está disponible, la máquina procederá a responder a las instrucciones. El ciclo de operaciones por el que se realiza esta respuesta se denomina *ciclo de ejecución*. Estas operaciones del ciclo de búsqueda, por supuesto, se realizan bajo la supervisión del controlador, pero la operación de éste durante el ciclo de búsqueda es independiente de los bits de más a la izquierda en el registro de instrucción, disponibles para el controlador. En la práctica, como no borrábamos el registro de instrucción, no sabíamos cuáles eran esos bits. Sin embargo, ahora que hemos buscado y almacenado esta primera instrucción en el registro de instrucción, el tratamiento de este punto seguirá un curso dependiente de la instrucción, convenido por los dos *bits de operación* de la misma. Para la instrucción *llamada sustracción*, su *ejecución* se realiza como muestra la tabla 8.10-2.

Tabla 8.10-2 Ciclo de ejecución

Ciclo de reloj	Descripción simbólica de la operación	Línea de control a habilitar
3. Transfiere la parte dirección del registro (6 bits de la derecha) al registro de direcciones de memoria (dirección 59)	IR (ADD)→MAR	TIR
4. Transfiere la palabra direccionada de memoria al bus y de ahí al registro CI	M→BUS BUS→CI	*E, R/W, W*
5. Complementa CI	$\overline{CI}$→CI	*C*
6. Incrementa CI	CI+1→CI	*I*
7. Salida del registro sumador al registro acumulador	Sumador→Acc	W_A

La máquina ha realizado ahora las primeras instrucciones. Dispondremos ahora que, consiguientemente, el controlador que diseñemos haya completado su secuencia completa y vuelto a su comienzo. Por tanto, la siguiente operación a realizar es transferir, de nuevo, el contador de programa al registro de direcciones de memoria. Pero recalcamos que hemos incrementado el contador de programa. Como consecuencia, la instrucción buscada será la segunda (en la posición 00001). La segunda instrucción se ejecutará como la primera, excepto que como se trata de una suma en lugar de una resta se eliminan las operaciones de complementación e incrementación. Así vemos cómo actúan las secuencias del controlador: busca después ejecuta, busca después ejecuta, etc. La operación de búsqueda es siempre la misma; las operaciones durante la ejecución dependen, por supuesto, de la instrucción.

Con respecto al sencillo sistema de la figura 8.9-1 tomamos una serie de decisiones (algunas bastante arbitrariamente, otras basadas en la experiencia.) Estas decisiones se refieren al número y función de los registros, a los tipos de interconexión entre sí y entre ellos y memoria, al número de palabras de memoria, al número de bits por palabra, al número y tipo de operaciones que la ALU puede realizar, etc. Estas materias constituyen la arquitectura y organización de la computadora. Una vez establecida una arquitectura y organización, queda la tarea de diseño del controlador. (El controlador para nuestra máquina se diseña en la siguiente sección.)

Hay por supuesto muchas arquitecturas y organizaciones que pueden asumirse por una parte de la máquina computadora. Después de muchos años de experimentación con una amplia gama de posibilidades, las máquinas computadoras de la actualidad incorporan generalmente una serie de características comunes, algunas de las cuales se han visto en nuestra computadora sencilla y que apuntamos ahora.

Primero señalamos que la máquina tiene memoria, en la que en principio *almacenamos* todas las instrucciones necesarias en la realización de un cálculo. Por esta razón, la máquina se denomina *computadora de programa almacenado*. Como la máquina está completamente instruida, no necesitaremos interrumpirla para darle direcciones adicionales para sus cálculos. La memoria almacena no sólo las instrucciones, sino también los operandos y los resultados de los cálculos. Por tanto, tendremos que hacer frecuentes referencias a memoria para leer de ella y escribir en ella. La posición de la dirección se encuentra en el registro de *direcciones de memoria* (*memory address register*, MAR) y no es sorprendente que se acceda al MAR desde distintas direcciones. En nuestro caso podemos acceder al MAR desde el contador de programa y desde el registro de instrucciones. (Computadoras más sofisticadas están provistas de medios adicionales indirectos y directos de acceso al MAR.)

A continuación, señalamos que nuestra máquina tiene una unidad aritmético-lógica en la que se realizan todas las operaciones lógicas y aritméticas. El resto de la máquina (además del controlador) no consta de más de un array de registros. Y los registros de almacenamiento no hacen nada. Exactamente constituyen el «papel de escribir» en el que escribimos las palabras digitales que necesitemos encontrar en futuras referencias. Nuestra ALU también es bastante simple; complementa, incrementa y suma. ALU más elaboradas realizan estas funciones y también operaciones lógicas (AND, OR, etc.), desplazamientos a izquierda o derecha, etc.

Observemos que al manipular la máquina se dispone que una serie de materias «se comprendan». Cuando se ha realizado una instrucción, se «comprende» *que la siguiente instrucción esté en la siguiente posición de memoria. Por tanto, no se necesita especificar la posición de dicha instrucción.* Nuevamente, cuando se va a realizar una suma la instrucción especifica solamente uno de los operandos de la suma. Se «comprende» que el otro esté en el registro acumulador. Finalmente, la instrucción no da indicación donde debe almacenarse el resultado de la suma. Se «comprende» que el resultado pueda dejarse en el acumulador. Todas estas compresiones conllevan una economía considerable en la longitud de palabra. Gracias a estas comprensiones que hemos incorporado en la máquina, en una instrucción solamente se necesita especificar una operación y una dirección de operando. Sin estas comprensiones en una instrucción se tendría que indicar la operación, la fuente del primer operando, la del segundo, el sitio donde debe almacenarse el resultado y la fuente de la instrucción siguiente. Por supuesto, esta instrucción así elaborada requerirá muchos más bits que las instrucciones más simples posibles debida a las comprensiones.

8.11 DISEÑO DEL CONTROLADOR DE LA COMPUTADORA

Un diseño para el controlador requerido de la figura 8.9-1 se da en la figura 8.11-1. En este diseño, con el fin de conseguir simplicidad visualizaremos un deliberado y descuidado extremo para economizar hardware. Como es habitual, el *elemento de comienzo* y todos los flip-flops son gobernados por una señal de reloj común (no mostrada explícitamente). Al cerrar el conmutador comienza la secuencia del registro de desplazamiento. Cuando la

siguiente cambio de dirección no ocurrirá hasta que DAC se haga primero DAC = 0 y después DAC = 1. De esta forma el controlador hace cierto que el receptor ha aceptado la palabra transmitida antes que una nueva palabra se coloque en el bus.

El diagrama de flujo para el controlador *B* se indica en la figura 8.13-3*b*. El estado 1*B* corresponde al estado 1*A*. Cuando los controladores *A* y *B* están en estos estados, los contadores de dirección no avanzan, los datos son válidos. E_B está en $E_B = 1$, y está en proceso una transferencia de palabra. Señalar que el controlador *A* no puede dejar el estado 1*A* hasta que el controlador *B* haya dejado el estado 1*B*. Señalar, además, que el controlador no volverá al estado 1*B* hasta que DAV haya ido primero a DAV = 0 y después vuelto a DAV = 1. De esta forma se asegura que la siguiente palabra transferida será una nueva palabra.

El controlador correspondiente al diagrama de flujo de la figura 8.13-3 se indica en la figura 8.13-4.

CAPÍTULO

NUEVE

COMPUTADORAS

El sistema de la figura 8.8-1 puede ser llamado, con cierta benevolencia, computadora. A pesar de su simplicidad, exhibe algunas importantes características que se encuentran en computadoras más sofisticadas. Pero nuestra «computadora» tiene la seria deficiencia de que no es capaz de hacer demasiado. En este capítulo vamos a considerar un diseño más elaborado.

9.1 UNA ARQUITECTURA MEJORADA

Como hemos señalado, el diseño de una computadora u otro procesador digital generalmente comienza considerando la arquitectura. Se han de tomar decisiones acerca del tipo de bloques componentes que se van a usar, por ejemplo, número de registros, tipos de registros, operaciones que tendrán que realizar las ALU con los registros y, más importante aún, los registros y componentes que han de ser interconectados para que los datos puedan ser transmitidos. No existe un procedimiento de diseño que permita seleccionar una «mejor» arquitectura. Se llega a una determinada estructura arquitectónica sobre la base de la experiencia del diseñador y, a menudo, simplemente «mejorando» un diseño previo. Una mejora del sistema de la figura 8.8-1 se ve en la figura 9.1-1. Generalmente las mejoras permiten hacer más cosas a un sistema a costa de aumentar su complejidad.

En nuestra nueva máquina, la memoria, esencial en toda máquina de programa almacenado, almacenará palabras de 12 bits. Por ahora no haremos ningún uso preciso de la longitud de palabra. Nuestra máquina tendrá un repertorio de 16 instrucciones, por lo cual se requieren 4 bits de instrucción. Si tuviésemos palabras de 8 bits nos quedarían 4 bits de dirección, permitiendo direccionar solamente una memoria de 16 palabras.

En ocasiones anteriores hemos indicado que las memorias de acceso aleatorio (RAM) se equipan con dos terminales de control y, posiblemente, otros. Uno de éstos es un terminal de *habilitación* (enable, E) y el otro es un

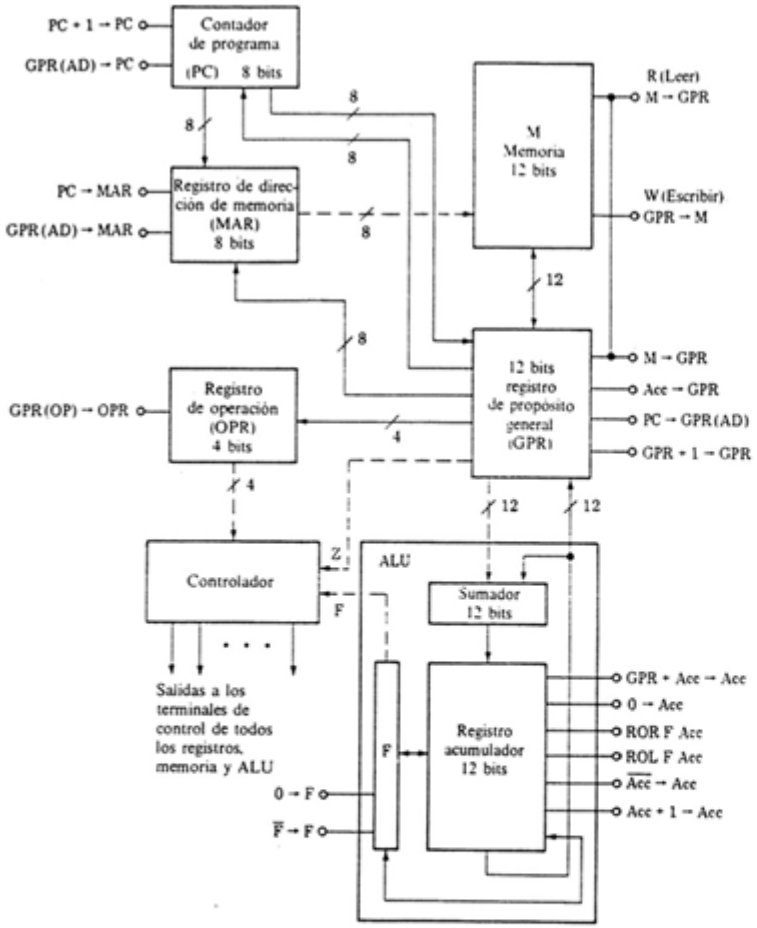


Figura 9.1-1 Arquitectura de una computadora.

terminal lectura-escritura (read/write, R/$\overline{W}$). Una palabra podrá ser escrita en la memoria o leída de la memoria sólo si E = 1. Para escribir hacemos R/$\overline{W}$ = 0 y para leer R/$\overline{W}$ = 1. En nuestro presente diseño es más conveniente tener la memoria equipada con dos terminales de control llamados simplemente leer (R) y escribir (W). Cuando W = 1, la palabra presente en los terminales de datos de memoria se escribe en la memoria y cuando R = 1, la palabra en memoria es leída y colocada en los terminales de datos. Cuando W = R = 0 no tiene lugar ni lectura ni escritura, mientras que W = R = 1 es una configura-

ción no permitida. Para ello podemos introducir la circuitería adecuada entre los terminales W y R, lo mismo que (Prob. 9.9-1) entre los terminales E y R/$\overline{W}$. Se supone que estos circuitos han de ser incorporados a la memoria de la figura 9.1-1 y no se muestran explícitamente. Como antes, tenemos un contador de programa (PC) y un registro de direcciones de memoria (MAR). No tenemos un registro de instrucciones, sino un registro de propósito general (GPR) de 12 bits y un registro de operaciones (OPR) de 4 bits que contendrá la parte operación de la instrucción. Cuando una instrucción (que consta de una parte operación y una parte dirección de operando) se lee de la memoria, va a parar primero al GPR. La parte operación de la instrucción será transmitida al OPR. La parte dirección del operando será transmitida al MAR. Mientras el GPR retiene los 8 bits de dirección, no están usándose 4 bits de la capacidad del GPR. La plena capacidad del GPR será usada cuando esté almacenando la parte dirección de un operando. La disposición presentada nos proporciona la conveniencia de poder separar la parte operación de una instrucción de la parte dirección de operando. Por ello podremos realizar algunas manipulaciones útiles de la parte dirección sin involucrar a la parte operación.

Como antes, hemos incorporado al contador de programa la capacidad de ser incrementado. Pero, además, ahora hemos proporcionado conexión directa entre el contador de programa y el registro de propósito general para que puedan transmitirse sus contenidos. Más aún, hemos proporcionado dos caminos separados, uno desde el PC al GPR y otro desde el GPR al PC, de tal manera que sus contenidos puedan ser intercambiados durante un simple ciclo de reloj. Para ello pueden usarse flip-flops maestro-esclavo así como de otros tipos, como se apuntó anteriormente.

El registro de propósito general es capaz de cuatro microoperaciones:

1. Puede transmitirse asimismo la palabra de memoria direccionada. Para efectuar esta transmisión pondremos 1 lógico en el terminal de control del GPR marcado M→GPR y simultáneamente 1 lógico en el terminal R de la memoria. Este terminal se usa sólo para permitir que una palabra de memoria sea leída en el GPR; así pues, podemos conectar el terminal READ al terminal M→GPR del GPR, como se ve en la figura 9.1-1.
2. Puede transmitir a él mismo el contenido del acumulador.
3. Puede transmitir a él mismo los 8 bits del contador de programa colocándolos en las 8 posiciones reservadas para la dirección.
4. Puede ser incrementado.

Nuestra nueva unidad lógico-aritmética se diseña de manera diferente a la de la ALU de la figura 8.9-1 y, comparada con aquel diseño, tiene un repertorio ampliado. La nueva ALU tiene un único registro de 12 bits más un registro extra F de un bit, o sea, un flip-flop. La ALU, por supuesto, tiene una cantidad sustancial de lógica, esto es, circuitería combinacional, que no se muestra. La ALU puede realizar 8 microoperaciones:

1. Puede sumar a su contenido presente el número de 12 bits que viene por la línea de 12 bits del GPR.
2. Puede limpiar el acumulador.
3. Puede limpiar F.
4. Puede complementar F.
5. Puede complementar el acumulador.
6. Puede incrementar el acumulador.
7. Puede desplazar cíclicamente a la derecha el acumulador incluyendo F [Rotate Right through F (ROR F, Acc)].
8. Igual que 7, pero a la izquierda [Rotate Left through F (ROL F, Acc)].

En la operación ROR F, Acc todos los bits del acumulador son desplazados una posición a la derecha. El bit del flip-flop F se coloca en la posición más a la izquierda del acumulador, y el bit originariamente en la posición más a la derecha del acumulador se coloca en el flip-flop F. Así pues, si vemos el conjunto del acumulador y el flip-flop F como un contador de anillo, la operación ROR F, Acc desplaza los bits una posición a la derecha. Asimismo, ROL F, Acc desplaza los bits una posición a la izquierda. Estas dos operaciones se representan en los diagramas de la figura 9.1-2.

Por simplicidad convenimos en que cada camino de transmisión se reserva para una función de transmisión única; o sea, no hay buses compartidos. Así, los 12 bits del GPR están siempre disponibles para aceptar datos o no, según lo desee la ALU, y los 12 bits del acumulador están siempre disponibles para el GPR. Todos los caminos por los que pueden realizarse transmisiones de palabras solamente cuando se activa una entrada de control están dibujados en línea continua. Los caminos en línea discontinua indican transmisiones que no requieren tal activación.

El controlador suministrará señales de control a todos los terminales de control de los diversos registros y de la memoria; esto es, durante ciclos de reloj adecuados uno o más terminales de control, dependiendo de la

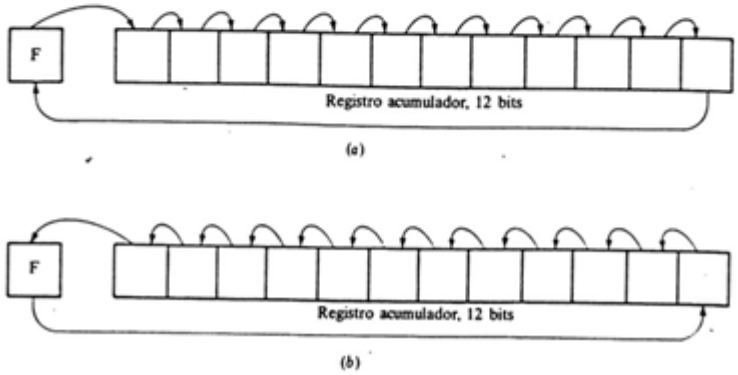


Figura 9.1-2 (*a*) La operación ROR F, Acc, y (*b*) la operación ROL F, Acc.

operación que en ese momento está en el registro de operaciones, serán puestos a nivel lógico 1. Hay ocasiones, sin embargo, en que las órdenes emitidas por el controlador se tienen que hacer depender de algo más que de la parte operación de la instrucción. Para ello a lo largo de la secuencia de microoperaciones que se producen durante la ejecución de una instrucción, una determinada microoperación puede depender de los resultados producidos por las microoperaciones precedentes. Por ello, hemos proporcionado dos entradas adicionales al controlador. Una de ellas viene del GPR y se llama Z. Esta línea Z estará en 1 cuando (y sólo cuando) cada uno de los 12 bits del GPR estén en 0. La otra entrada viene del flip-flop F asociado al registro acumulador. Indicamos ahora estas conexiones para comprender

Tabla 9.1-1 Componentes y operaciones de control del sistema

Componente	Simbolismo de control	Explicación
Memoria	1. GPR→M	Escribe el contenido del GPR en el lugar de memoria direccionado
Contador de programa (PC)	2. PC+1→PC	Incrementa el PC
	3. GPR(AD)→PC	Transmite los bits de dirección del GPR al PC
Registro de direcciones de memoria (MAR)	4. PC→MAR	Transmite desde el PC al MAR
	5. GPR(AD)→MAR	Transmite los bits de dirección del GPR al MAR
Registro de operación (OPR)	6. GPR(OP)→OPR	Transmite los bits de operación del GPR al OPR
Registro de propósito general (PR)	7. M→GPR	Transmite palabra direccionada al GPR
	8. Acc→GPR	Transmite el contenido del Acc al GPR
	9. PC→GPR(AD)	Transmite el contenido del PC a la parte dirección del MAR
	10. GPR+1→GPR	Incrementa el GPR
Unidad aritmética lógica (ALU)	11. GPR+Acc→Acc	Suma el número del GPR al número del Acc y deja el resultado en Acc
	12. 0→Acc	Limpia Acc
	13. ROR F, Acc	Desplaza cíclicamente a la derecha el Acc junto con F
	14. ROL F, Acc	Desplaza cíclicamente a la izquierda el Acc junto con F
	15. 0→F	Pone a «0» el flip-flop F
	16. F→F	Complementa el flip-flop F
	17. $\overline{\text{Acc}}$→Acc	Complementa el Acc
	18. Acc+1→Acc	Incrementa el Acc

más tarde que el controlador seguirá una u otra secuencia de microoperaciones dependiendo del nivel lógico de Z o de F.

La tabla 9.1-1 es una lista de todas las componentes de nuestro sistema, excepto el controlador, y todos los símbolos usados para rotular los terminales de control; por convenio se establece, para cada terminal, la microoperación que se realiza cuando el terminal se activa.

9.2 INSTRUCCIONES

Como antes, cuando el controlador comience a trabajar, lo hará cíclicamente, cogiendo una instrucción, ejecutándola, trayendo otra instrucción, etc. Supongamos que hemos puesto manualmente en el contador de programa (PC) la dirección de la primera instrucción. La secuencia de microoperaciones que hay que seguir ahora para buscar una instrucción es la siguiente:

Ciclo de reloj	Microoperación	Explicación
1	PC→MAR	Transmitir la posición de la instrucción desde el PC al MAR
2	M→GPR PC+1→PC	Transmitir la palabra direccionada al GPR; incrementar el PC
3	GPR(OP)→OPR	Transmitir la parte operación de la instrucción al OPR

Cuando la secuencia de microoperaciones de búsqueda se ha completado, la parte operación de la instrucción ha sido transmitida al registro de operación. Si la instrucción involucra a un operando, la dirección de este operando se deja en la parte dirección del registro de propósito general GPR (AD). Desde luego, no todas las instrucciones involucran un operando; en tales casos el contenido de GPR (AD) es totalmente irrelevante al final de la secuencia de búsqueda.

Vamos a definir ahora una serie de instrucciones de nuestra computadora. Desde luego, una instrucción es factible sólo si puede ser efectuada poniendo en nivel activo, en alguna secuencia, los terminales de órdenes de la figura 9.1-1. Si definimos pocas instrucciones y además simples, el controlador será simple. En tal caso, los programas de instrucciones escritos para alcanzar algún resultado tendrán que ser necesariamente más largos. Si introducimos numerosas y complicadas instrucciones, los programas serán más simples, pero el controlador tendrá que ser más elaborado. Nosotros favoreceremos la alternativa de pocas y simples instrucciones, siendo éstas representativas de las instrucciones usadas en componentes comerciales más sofisticados. Introducimos primero un número de instrucciones que son completamente elementales en el sentido de que requieren un único terminal de control para ser puestas en nivel activo. Así pues, cada una de ellas puede ser ejecutada en un simple ciclo de reloj y constan de una única microoperación. Ninguna d ellas involucra referencias a memoria, ni para leer ni para escribir, y po consiguiente ninguna necesita una dirección. Definimos:

Instrucción (microoperación)	Explicación	Nemotécnico
0→Acc	Limpiar acumulador	CRA
$\overline{\text{Acc}}$→Acc	Complementar acumulador	CTA
Acc+1→Acc	Incrementa acumulador	ITA
0→F	Limpia flip-flop F	CRF
$\overline{\text{F}}$→F	Complementa flip-flop F	CTF
PC+1→PC	Salta a la siguiente instrucción si F es cero	SFZ
Desplazamiento cíclico a la derecha	Desplaza cíclicamente a la derecha el acumulador junto con F	ROR
Desplazamiento cíclico a la izquierda	Desplaza cíclicamente a la izquierda el acumulador junto con F y Acc	ROL

La instrucción SKIP, PC+1→PC merece algún comentario. Supongamos qu esta instrucción está almacenada en la celda de memoria de dirección Durante la secuencia de búsqueda de microoperaciones que leen es instrucción desde la memoria, el contador de programa PC es incrementad y llega a valer $K+1$. Sin embargo, la ejecución de la instrucción, com cualquier otra instrucción, causa un incremento adicional, de tal manera qu al final de la ejecución tendremos PC=$K+2$ y la instrucción en la direcci de memoria $K+1$ será saltada. Hagamos notar además que este salto condicional sobre el valor de F. Si F=0, la microoperación de salto PC+ →PC es ejecutada. Pero si cuando le toca el turno a esta microoperación salto es F=1, la instrucción no ejecuta operación alguna. Al establecer arquitectura de la figura 9.1-1 quisimos que esta operación de salto fue condicional sobre F. Por ello F es allí una de las entradas al controlad Vemos cómo esta instrucción de salto, en cooperación con una instruccci JUMP (presentada más adelante), permitirá a nuestra máquina seleccion un determinado curso de acción u otro, según el valor de F.

Consideremos ahora otras instrucciones que requerirán una secuencia microoperaciones para su ejecución.

ADD, dirección

Suma al contenido actual del acumulador el número (operando) que se encuentra en la dirección de memoria especificada como parte de la instrucción.

Después de la secuencia de microoperaciones de búsqueda de la instrucci la parte operación de la misma queda en el registro de operación (OPR) y

parte dirección de operando de la instrucción queda en la parte dirección del registro de propósito general GPR (AD). La instrucción se ejecuta a partir de ese momento siguiendo la secuencia de microoperaciones:

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→MAR	Transmite la dirección del operando desde GPR(AD) a MAR
2	M→GPR	Lee desde la memoria la palabra de la celda cuya dirección está en el MAR
3	GPR+Acc→Acc	Suma el contenido del GPR al contenido del Acc, dejando la suma en Acc

Frecuentemente, en lugar de sumar al acumulador un operando desde la memoria, simplemente queremos transmitir tal operando al acumulador. Una instrucción de esta clase podría ser descrita como «cargar el acumulador» y tener el nemotécnico LDA. Sin embargo, en aras de la economía, no introduciremos tal instrucción. En lugar de ello, cuando necesitemos una

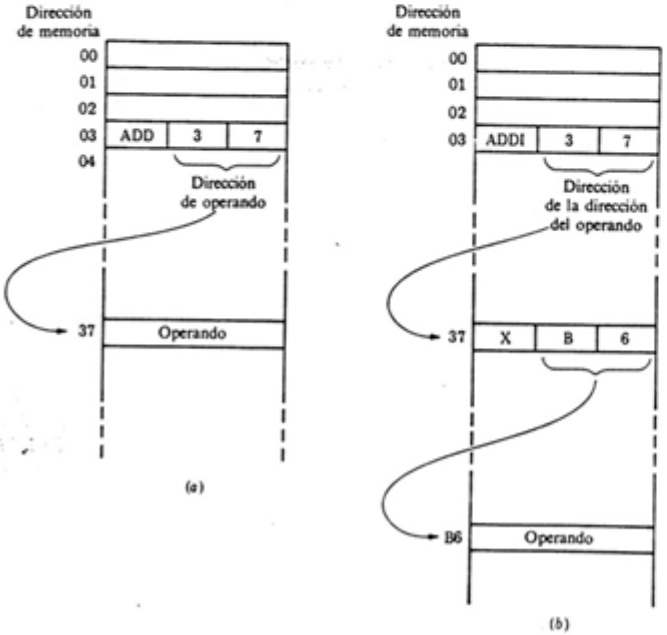


Figura 9.2-1 Diferencia entre (*a*) instrucción de direccionamiento directo y (*b*) instrucción indirecta.

operación de «carga», lo haremos usando en secuencia las dos instruccione
[RA (limpiar el acumulador) y ADD (sumar el acumulador)].

La instrucción ADD proporciona la dirección del operando que ha de se
sumado al acumulador. Como veremos, es a menudo muy conveniente tene
una instrucción que, en lugar de proporcionar la dirección del operand
almacene más bien la dirección del lugar de memoria donde está la direcció
del operando. Cuando la dirección de un operando se proporciona de est
manera, decimos que la dirección es *indirecta*. Para indicar que un
instrucción involucra una dirección indirecta, añadiremos el símbolo I
nemotécnico de la instrucción. La diferencia entre ADD [suma al acumulado
y ADDI (sumar al acumulador, indirecto)] aparece clara en la figura 9.2-
Aquí, para simplificar, hemos escrito palabras binarias en notación hexadec
mal (ver sección 1.13). En la figura 9.2-1*a* consideramos que la celda
memoria 03 contiene la instrucción ADD 37, lo que significa que va a s
sumado al acumulador el operando en la celda de memoria de dirección 3
(La parte operación de la instrucción usa 4 bits, como cada uno de l
números 3=0011 y 7=0111.) Se encuentra directamente al operando en
celda de memoria 37. En la figura 9.2-1*b*, donde se propone direccionamien
indirecto, encontramos en la celda de dirección 37 no el operando, sino
dirección del operando. (Puesto que la celda 37 contiene solamente un
dirección, sólo 8 de los 12 bits son relevantes y los primeros 4 bits pueden s
cualquiera.) En la celda 37 encontramos que el operando está en la celda
memoria B6, y éste es el operando que se suma al acumulador. La secuenc
de microoperaciones desencadenada por la instrucción ADDI es:

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→MAR	Transmite la dirección desde el GPR al MAR
2	M→GPR	Transmite el contenido de la celda de memoria direccionada al GPR (el GPR tendrá entonces la dirección del operando)
3	GPR(AD)→MAR	Transmite la dirección del operando al MAR
4	M→GPR	Transmite el operando direccionado al GPR
5	GPR+Acc→Acc	Suma el contenido del GPR al Acc

Si necesitamos efectuar una carga indirecta del acumulador, podem
hacerlo con la secuencia de instrucciones CRA (limpiar el acumulad
seguida de ADDI.

La secuencia de instrucciones CRA y ADD copia un operando desde
memoria: esto es, se *lee* la memoria y se transmite la palabra así leída
acumulador. La instrucción inversa *escribe* en la memoria el contenido
acumulador. El nemotécnico para esta instrucción es STA [almacenar
acumulador (store the accumulator)]. Para esta instrucción STA hacemos
siguiente definición:

STA, dirección

Almacena el contenido del acumulador en la dirección de memoria especificada.

La secuencia de microoperaciones es:

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→MAR	Transmite la dirección desde el GPR al MAR
2	Acc→GPR	Transmite el contenido del Acc al GPR
3	GPR→M	Escribe el contenido del GPR en la dirección de memoria retenida en el MAR

Hemos apuntado que las instrucciones están listadas en memoria según el orden en que van a ser ejecutadas, y que se accede a una instrucción después de otra incrementando el contador de programa. Pero, como veremos, hay veces en que no es útil seguir ese procedimiento, sino bifurcar a una instrucción que no está en orden secuencial. Las próximas instrucciones proporcionan tal facultad.

JMP, dirección

Bifurca (JuMP) a la instrucción almacenada en la celda de memoria especificada en la instrucción.

Para aclarar el propósito de esta instrucción, supongamos que como el controlador trae y ejecuta una y otra instrucción trae la instrucción JMP 29. Entonces la única ejecución que el controlador realiza es transmitir la dirección, esto es, 29, no al registro de dirección de memoria (MAR), sino al contador de programa (PC). La única microprogramación exigida es:

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→PC	Transmite la dirección de la siguiente instrucción desde el GPR al PC

Así pues, la *siguiente* instrucción que será buscada es la instrucción de la celda de memoria 29; si a continuación no se encuentran otras instrucciones de bifurcación, la máquina seguirá ordenadamente con la instrucción de la celda de memoria 30, 31, etc.

JMPI, dirección

Bifurca a la instrucción cuya dirección está almaecenada en la dirección de memoria especificada en la instrucción.

Esta instrucción es la forma *indirecta* de la instrucción JMP dada antes. La parte dirección de la instrucción JMPI es la dirección de la celda de memoria que retiene la *dirección de la siguiente* instrucción. La secuencia de microoperaciones es:

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→MAR	Transmitir la dirección al MAR
2	M→GPR(AD)	Lee de la memoria la dirección de la instrucción siguiente
3	GPR(AD)→PC	Transmite la dirección de la instrucción siguiente desde el GPR al PC

Antes de presentar la próxima instrucción se precisa alguna explicación. Supongamos que escribimos un programa (una lista de instrucciones para que el ordenador las ejecute con el fin de cumplir algún propósito) para nuestro ordenador. Supongamos, además, que durante la ejecución del programa encontramos a menudo un conjunto particular de instrucciones que vuelven a ocurrir regularmente. Por ejemplo, el programa puede necesitar a menudo la multiplicación de dos números. Como veremos, nuestra máquina no es capaz de realizar tal operación directamente. En lugar de ello para efectuar una multiplicación el ordenador tendrá que ejecutar una lista de instrucciones, o sea, una *rutina* de la multiplicación. Si nos place, podemos incluir esta rutina en nuestro programa cada vez que se requiere una multiplicación. Hay, sin embargo, un procedimiento alternativo que tiene interés obvio. Podemos almacenar una sola vez esta rutina en algún sitio de la memoria. Entonces, cuando se necesite realizar una multiplicación, podemos bifurcar a la rutina almacenada y cuando la rutina de la multiplicación se haya completado, podemos volver al lugar de nuestro programa desde el que bifurcó.

Este proceso de abandonar la secuencia ordenada de instrucciones para ir a una rutina especial es denominado *llamar a subrutina* (calling a subroutine) o *bifurcar a subrutina* (jumping to a subroutine). El final de una llamada a subrutina debe ser siempre una *vuelta* al lugar del programa desde el que se hizo la bifurcación.

Antes de bifurcar a una subrutina, debemos *elegir una celda de memoria* para almacenar la dirección con la que ha de ser recargado el contador de programa cuando la subrutina sea completada. Esta dirección de esta *celda elegida* debe ser incluida en la instrucción de llamada a la subrutina. También debemos incluir en la instrucción la dirección de memoria donde comienza la

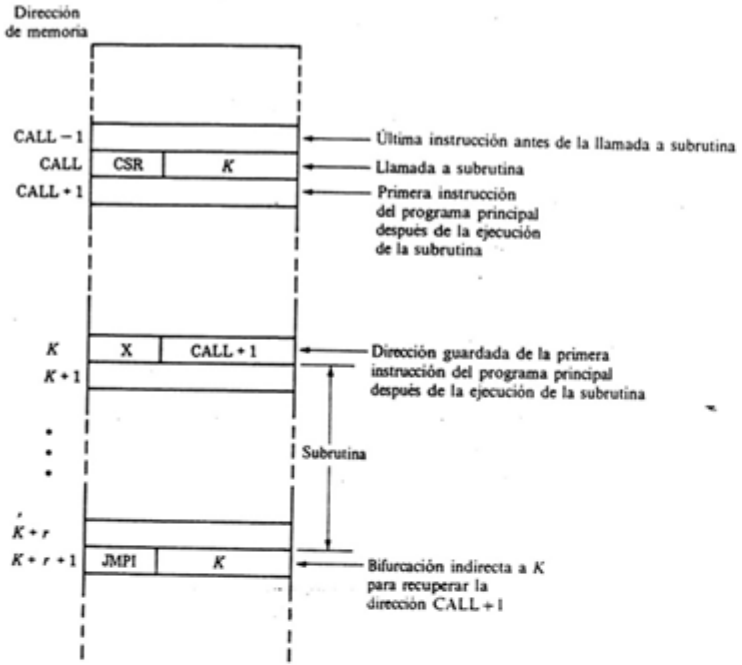


Figura 9.2-2 Manipulaciones involucradas en una llamada a subrutina.

subrutina. Así pues, esta instrucción CSR [llamar a subrutina (call subroutine)] involucra dos direcciones. Sin embargo, como nuestra instrucción CSR, igual que cualquier otra, sólo tiene cabida para una dirección debe haber una cierta disposición o conocimiento previos.

El mecanismo que proponemos para llamar a una subrutina y volver al programa principal interrumpido se muestra en la figura 9.2-2. La instrucción CSR está en la celda de memoria de dirección CALL. La subrutina a la que se va bifurcar está almacenada en r celdas sucesivas de direcciones $K+1$, $K+2$, ..., $K+r$. Igual que para el programa principal, convenimos que las instrucciones de la subrutina están almacenadas en celdas sucesivas de memoria en el mismo orden en que van a ser ejecutadas. Hemos tomado la precaución de dejar las celdas K y $K+r+1$ disponibles para nuestro propósito. Ahora para llamar a la subrutina y después volver desde ella, escribimos en nuestro programa, en los lugares adecuados, dos instrucciones. La primera de ellas es la instrucción CSR, que estamos discutiendo, con la definición:

CSR, dirección

Llamar a subrutina. Almacenar la dirección de vuelta al programa principal (o sea, la dirección CALL+1) en la celda de memoria especificada en esta instrucción (o sea, dirección=K) y bifurcar a la celda de memoria $K+1$.

La segunda instrucción se escribe en la celda de memoria $K+r+1$ y es la instrucción de bifurcación indirecta JMPI K que retorna el programa a la instrucción almacenada en CALL+1. A continuación se detalla la secuencia requerida de microoperaciones para realizar la instrucción CSR.

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→MAR	Transmite al MAR la dirección donde ha de almacenarse la dirección de vuelta; la dirección de almacenamiento es K
2	GPR(AD)→PC PC→GPR(AD)	Intercambia los contenidos del PC y del GPR [después del intercambio el PC contiene la dirección K y GPR(AD) contiene CALL+1 puesto que el PC fue incrementado desde el valor CALL A CALL+1 durante el ciclo de búsqueda
3	GPR(AD)→M	Transmite el contenido de GPR(AD) a la memoria [el resultado es que la dirección CALL+1 será escrita en la celda de memoria K]
4	PC+1→PC	Incrementa el PC [el PC contendrá entonces la dirección $K+1$; la siguiente instrucción será tomada entonces de la celda $K+1$, donde está la primera instrucción de la subrutina]

Se necesita de nuevo una explicación antes de describir la siguiente instrucción. Hay ocasiones en que una secuencia de instrucciones debe ser ejecutada muchas veces seguidas. Tal secuencia repetitiva es realizada ejecutando un *lazo* (*loop*) *de programa*. La idea es que una vez que la secuencia ha sido completada la siguiente instrucción sea una instrucción de bifurcación para que el programa vuelva al comienzo de la secuencia. La instrucción ISZ se usa para terminar este proceso cíclico cuando se ha repetido el número requerido de veces. Supongamos que se necesita repetir la secuencia n veces. Entonces se almacena el número negativo $-n$ en alguna celda apropiada de memoria. En cada ciclo se lee de memoria el número $-n$, se incrementa y se devuelve a la memoria. Cuando la secuencia ha sido ejecutada n veces, el número habrá sido incrementado hasta el valor cero. En este punto la «siguiente instrucción», que es la instrucción para volver al principio del lazo será saltada y el programa avanzará fuera del lazo. La definición precisa es:

ISZ, direccion

Incrementar y saltar si cero (Increment and skip if zero). Leer el número de la celda especificada de memoria, incrementarlo y devolverlo a su celda original. Si después de incrementarlo, el número es cero, saltar a la siguiente instrucción.

Las microoperaciones que requiere esta instrucción ISZ son:

Ciclo de reloj	Microoperación	Explicación
1	GPR(AD)→MAR	Transmitir al MAR la dirección donde está alumbrado el número que ha de ser incrementado
2	M→GPR	Leer de la memoria el número
3	GPR+1→GPR	Incrementar el número
4	GPR→M	Devolver el número a la memoria
5	PC+1→PC (si GPR=0)	Saltar a la siguiente instrucción si GPR=0

Notar que para realizar esta instrucción el controlador tiene que saber cuándo está en cero el registro GPR. Como se indicó en la figura 9.1-1, tal información se tiene disponible sobre la línea marcada Z [cero (Zero)]. Notar además que el contador de programa se incrementa durante el ciclo de traida. Así pues, si se incrementa de nuevo, como en el ciclo de reloj 5, PC habrá sido incrementado dos veces y será saltada la instrucción de bifurcación que vuelve el programa al comienzo del lazo.

9.3 SUMARIO DE INSTRUCCIONES

Para tener una vista global del ordenador que hemos postulado, recopilamos aquí el repertorio completo de instrucciones:

1. CRA borrar el acumulador
2. CTA complementar el acumulador
3. ITA incrementar el acumulador
4. CRF borrar el flip-flop F
5. CTF complementar el flip-flop F
6. SFZ saltar a la siguiente instrucción si F=0
7. ROR desplazar cíclicamente a la derecha
8. ROL desplazar cíclicamente a la izquierda
9. ADD sumar al acumulador
10. ADDI sumar indirecto al acumulador
11. STA almacenar en memoria del acumulador
12. JMP bifurcar
13. JMFI bifurcar indirecto
14. CSR llamada a subrutina
15. ISZ incrementar y saltar si Z=0
16. HLT alto

Notar que, además de las instrucciones que hemos descrito, hemos añadido la esencial instrucción de parada.

En las secciones siguientes se ven algunos ejemplos de cómo puede ser programada nuestra computadora para realizar algunas funciones útiles.

9.4 SUMA Y RESTA

En la figura 9.4-1 hemos dispuesto la memoria de forma que nuestra computadora pueda calcular la suma de tres números y guardarla en la memoria. (Todos los números están expresados en código hexadecimal.) Las primeras seis celdas 00 a 05 de memoria almacenan instrucciones. Los 4 bits más a la izquierda de una instrucción definen la operación. Puesto que no hemos establecido un código para las operaciones, las dejamos escritas en forma nemotécnica. Las primeras cuatro instrucciones tienen direcciones. La instrucción HLT no tiene dirección, así que la parte dirección de esta instrucción es irrelevante. Los operandos que han de ser sumados (017+ +00B+01C=03B) están almacenados en las celdas 06, 07 y 08. La suma será almacenada en la celda 09. La primera instrucción limpia el acumulador y lo carga con el contenido de la celda 06. La segunda suma el contenido de la celda 07, etc. Nosotros hemos almacenado el resultado en la celda 09. Supongamos, sin embargo, que una vez realizados los cálculos, no necesita-

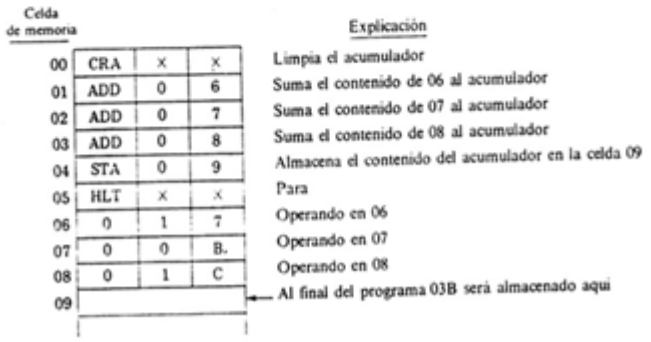


Figura 9.4-1 Memoria con un programa que calcula la suma de tres números.

mos más tiempo ni el programa ni los operandos. En tal caso, podríamos haber almacenado el resultado en cualquiera de las celdas 00 a 08.

En el programa simple de la figura 9.4-1, desde el principio es fácil ver que se requieren 6 celdas, 00 a 05, para escribir la secuencia de instrucciones. Así pues, podíamos colocar el primer operando en la celda 06 y, consecuentemente, escribir las dos primeras instrucciones como CRA y ADD 06. En general será mucho más conveniente no tener que especificar un lugar numérico de memoria preciso para cada operando hasta después de que el programa haya sido totalmente escrito. Es más conveniente especificar direcciones de operandos en forma simbólica, siendo los símbolos eventualmente reemplazados por números cuando el programa esté completo. Tal procedimiento también impedirá tener que cambiar muchos números cuando un paso de programa o un operando sea sumado o suprimido.

El programa de la figura 9.4-1 está duplicado en la tabla 9.4-1 con las direcciones en forma simbólica. Puesto que en el programa no se necesita hacer referencia a ninguna instrucción del mismo, no se necesita asignar dirección simbólica a ninguna de las seis celdas donde está almacenado. Cuando el programa se presenta en la forma de la tabla 9.4-1, tendrán que tomarse un cierto número de decisiones. Así, ¿en qué celda de memoria va a comenzar el programa? En la figura 9.4-1 colocamos arbitrariamente el programa a partir de la celda 00, pero por supuesto se puede comenzar a nuestra conveniencia en cualquier otro lugar. Una vez que se ha decidido la celda inicial del programa, las instrucciones siguientes deben colocarse ordenadamente en los lugares sucesivos. Sin embargo, no es necesario que los operandos sigan inmediatamente después de las instrucciones, ni que los operandos estén colocados consecutivamente entre sí.

A continuación vamos a usar nuestro ordenador para restar un número de otro, digamos 0B7-09C (Hex.). Nuestra máquina más simple de la figura

Tabla 9.4-1 El programa de la figura 9.4-1 escrito asignando direcciones simbólicas a las posiciones de memoria a las que debe hacerse referencia

Celda simbólica	Contenido	Comentario
	CRA	Limpia el acumulador
	ADD W	Suma el contenido de W al acumulador
	ADD X	Suma el contenido de X al acumulador
	ADD Y	Suma el contenido de Y al acumulador
	STA Z	Almacena el contenido del acumulador en la celda Z
	HLT	Para
W	017	Operando en W
X	00B	Operando en X
Y	01C	Operando en Y
Z	XXX	Celda para almacenar el resultado

Tabla 9.4-2 Programa para la resta

Rótulo	Contenido	Comentario
	CRA	Limpia el acumulador
	ADD SUB	Suma el sustraendo al acumulador
	CTA	Complementa el acumulador
	ITA	Incrementa el acumulador
	ADD MIN	Suma el minuendo al acumulador
	STA DIF	Almacena en la celda rotulada «DIF»
	HLT	Para
SUB	09C	Sustraendo en la celda rotulada «SUB»
MIN	0B7	Minuendo en la celda rotulada «MIN»
DIF	XXX	Celda para almacenar la diferencia (la diferencia será 0B7 – 09C = 01B)

8.9-1 tenía una instrucción de resta. Puesto que nuestra máquina actual tiene una organización y arquitectura que no incluye tal instrucción, tendremos que escribir un programa. Para efectuar una resta tendremos que formar el negativo del sustraendo. Este cambio de signo puede ser efectuado en el acumulador ejerciendo la facultad de complementar e incrementar el acumulador. Primero debemos cargar el sustraendo en el acumulador, porque si se cargase primero el minuendo, no se podría más tarde cambiar el signo del acumulador. El programa de la resta se da en la tabla 9.4-2. En ella se ha cambiado el término «dirección simbólica» por *rótulo* (*label*).

La tabla 9.4-3 muestra el programa que calcula 0B7 – 09C – 005. Puesto que ahora hay dos sustraendos, el registro acumulador será usado dos veces para cambiar signos. Cuando haya sido efectuado un cambio de signo, el resultado debe ser almacenado en la memoria para dejar disponible el acumulador para cambiar el signo del segundo sustraendo.

9.5 USO DE JMP E ISZ

Como un ejemplo de la utiliadad de las instrucciones JMP e ISZ y de la facultad de indirección, veamos cómo nuestra máquina puede usarse para formar la suma de un gran número (digamos 100) de sumandos. De algún modo hemos de poner nuestros 100 sumandos en la memoria. Después de ello podemos escribir un programa en el que, simplemente, incluyamos 100 instrucciones sucesivas de suma, que difieren una de otra únicamente en la dirección del operando. En la tabla 9.5-1 se lista un programa más corto y más elegante.

El programa proporciona una celda de memoria (ANA) para retener la

Tabla 9.4-3 Programa para la resta involucrando dos sustraendos

Rótulo	Contenido	Comentario
	CRA	Limpia el acumulador
	ADD SUB(1)	Suma al acumulador el contenido de la celda SUB(1) (sustraendo 1)
	CTA	Complementa el acumulador
	ITA	Incrementa el acumulador
	STA NSUB(1)	Almacena en la celda NSUB(1) (−sustraendo 1)
	CRA	Limpia el acumulador
	ADD SUB(2)	Suma sustraendo (2) al acumulador
	CTA	Complementa el acumulador
	ITC	Incrementa el acumulador
	ADD MIN	Suma al acumulador el contenido de la celda MIN (minuendo)
	ADD NSUB(1)	Suma el contenido de la celda NSUB(1)
	STA RES	Almacena en la celda RES (resultado)
	HLT	Para
MIN	0B7	Minuendo en la celda MIN
SUB(1)	09C	Sustraendo 1 en la celda SUB(1)
SUB(2)	005	Sustraendo 2 en la celda SUB(2)
NSUB(1)	XXX	Celda para almacenar el negativo de SUB(1)
RES	XXX	Celda para almacenar el resultado

dirección del siguiente sumando que ha de ser sumado. Después de cada suma, esta dirección ha de ser incrementada. Una segunda celda actuando como contador (CTR) se usa para guardar la cuenta del número de sumandos sumados. Las sumas repetitivas se programan mediante un ciclo que efectúa una suma cada vez que se recorre. Al comienzo la celda contador guarda el número −100 (decimal) = F9C (hex.). Por cada suma el contador es incrementado, hasta que finalmente alcanza el valor cero.

Refiriéndonos ahora al programa de la tabla 9.5-1, encontramos que, después de que el acumulador ha sido limpiado, la siguiente instrucción efectúa una suma *indirecta*. La celda de memoria ANA guarda no el sumando, sino la dirección del sumando. Al principio la dirección en ANA es la dirección del primer sumando (FAD), y hacemos notar que hemos de tener el primer sumando en la celda de dirección FAD. Habiendo colocado ADD(1) en el acumulador, ahora incrementamos la dirección almacenada en ANA. Para esta operación de incrementar usamos una instrucción ISZ. Aunque sabemos que el contenido de ANA no llegará nunca a ser cero, es conveniente usar esta instrucción para incrementar, a pesar de no aprovechar todas sus posibilidades. A continuación incrementamos el número de la celda CTR.

Tabla 9.5-1 Programa para sumar un gran número de sumandos

Rótulo	Contenido	Comentario
	CRA	Limpia el acumulador
LOOP	ADDI ANA	Suma al acumulador el número cuya dirección está en la celda ANA (dirección del siguiente sumando)
	ISZ ANA	Incrementa la dirección del sumando
	ISZ CTR	Incrementa el número en la celda CTR y se salta la siguiente instrucción si el incremento hace cero el contenido de CTR
	JMP LOOP	Bifurca a la instrucción rotulada LOOP
	STA RES	Almacena el resultado en la celda RES
	HLT	Para
RES	XXX	Aquí va a ser almacenado el resultado
ANA	FAD	Esta celda contiene la dirección del siguiente sumando
CTR	F9C	Esta celda contiene el número de sumas que han de hacerse (representación por complemento a dos de −100)
FAD	ADD(1)	100 sumandos que han de sumarse
	ADD(2)	
	.	
	.	
	.	
	ADD(100)	

Mientras que no haya alcanzado el valor cero, no se salta la siguiente instrucción. Esta siguiente instrucción vuelve el programa a la instrucción ADDI, para sumar el siguiente sumando al acumulador. Puesto que esta instrucción ADDI es referenciada por otra instrucción, es necesario asignar específicamente un rótulo a la celda que la contiene. El programa será recorrido 100 veces, hasta que el contenido de la celda contador sea cero. Cuando esta ocurra, la instrucción JMP será saltada, el resultado quedará almacenado y el ordenador parará.

9.6 PROGRAMA PARA MULTIPLICACIÓN

Una forma de programar la multiplicación es usar una sucesión de sumas simplemente. Así, para multiplicar 18 por 16 sumamos 18 a sí mismo 16 veces o bien 16 a sí mismo 18 veces. Este más bien poco elegante esquema no ha de desaprobarse cuando se usa en computadoras pequeñas y lentas. Aun así han tenido buena acogida miniordenadores como el DEC pdp-8, que no proporciona ningún otro método en su esquema básico de instrucciones.

Por supuesto se puede usar el procedimiento explicado, pero ahora vamos a considerar el método que usa desplazamiento y suma, que es la forma ordinaria de multiplicar con papel y lápiz (figura 5.15-1). Podemos usar este método porque hemos incorporado a nuestra máquina la facultad de desplazar el contenido del acumulador y hemos proporcionado al acumulador una entrada desde el flip-flop F, que es una extensión del acumulador.

El esquema es el siguiente. En la memoria reservamos celdas para almacenar el multiplicando, el multiplicador y la suma de los productos parciales (multiplicando = MD, multiplicador = MR, suma de productos parciales = SP). Transmitimos el multiplicador al registro acumulador y lo desplazamos cíclicamente a la derecha, colocando así el bit menos significativo del multiplicador en el flip-flop F. El multiplicador desplazado se devuelve entonces a su celda MR, quedando así el acumulador disponible para manipular el multiplicando. El multiplicando se transmite entonces al acumulador. Si F = 1, el contenido de la celda SP (inicialmente SP está limpia) se suma al multiplicando en el registro acumulador y el contenido del acumulador se devuelve entonces a la celda SP. Una vez hecho esto, devolvemos el multiplicando al acumulador, lo desplazamos cíclicamente a la izquierda, y devolvemos el multiplicando desplazado a su celda MD. Si F = 0, no se realiza la suma del multiplicando al contenido de SP, pero el desplazamiento del multiplicando sí. El desplazamiento a la derecha del multiplicador se hace de tal manera que podamos consultar cada bit del multiplicador y decidir una u otra acción según sea 1 ó 0. Esta consulta y consiguiente decisión sólo puede hacerse colocando cada bit en el flip-flop F, puesto que sólo este flip-flop de la ALU tiene conexión con el controlador. El desplazamiento a la izquierda del multiplicando se hace para multiplicar el multiplicando por las potencias sucesivas de 2.

Puesto que los datos numéricos tienen 12 bits, este proceso de sumar productos parciales (cuando F = 1) y desplazar debe ser hecho 12 veces. Consiguientemente hemos de incorporar un *lazo* o *ciclo* en nuestro programa y debemos tener una celda *contador* en la memoria que sirva para contar el número de veces que se completa el lazo.

El programa de la multiplicación se da en la tabla 9.6-1. Hemos supuesto que el multiplicando y el multiplicador son de tal forma que el producto puede ser contenido en 12 bits. (El producto de dos números de 12 bits puede extenderse hasta 24 bits como máximo.) Las instrucciones en las celdas de memoria 1 y 2 limpian la celda SP, donde se va a ir fabricando la suma de los productos parciales. No tenemos un medio directo para limpiar una celda de memoria. Así, para limpiar SP, limpiamos primero el acumulador y luego transmitimos a SP el contenido del acumulador. Con las instrucciones de las celdas 3, 4 y 5 colocamos el bit más a la derecha del multiplicador en el flip-flop F, desplazamos el multiplicador y los devolvemos a la memoria. Las instrucciones de las celdas 6, 7 y 8 determinan el curso proseguido por el programa, según sea F = 1 o F = 0. Las instrucciones de las celdas 9, 10, 11 y 12, que se ejecutan si F = 1, suman el multiplicando al contenido de la celda SP. Las instrucciones de las celdas 14, 15, 16 y 17 desplazan el multiplicando y lo devuelven a su celda de memoria. La operación de limpia en la línea 13

Tabla 9.6-1 Programa para la multiplicación

Celda de memoria	Rótulo	Contenido	Comentario
1		CRA	Limpia la celda donde la suma de productos
2		STA SP	SP, esto es, el resultado de la multiplicación, será almacenada
3	LOOP	ADD MR	Carga el contenido de la celda MR en el acumulador
4		ROR	Desplaza cíclicamente de derecha a izquierda el bit más a la derecha del multiplicador para colocarlo en F
5		STA MR	Devuelve el multiplicador a la celda MR
6		SFZ	Se salta la siguiente instrucción si F = 0
7		JMP 1	Bifurca a la celda rotulada «1» (puesto que F ≠ 0)
8		JMP 0	Bifurca a la celda rotulada «0» (puesto que F = 0)
9	1	CRA	Carga el contenido de la celda MD en el
10		ADD MD	acumulador
11		ADD SP	Suma el contenido de la celda SP al acumulador
12		STA SP	Almacena el contenido del acumulador en la celda SP
13		CRF	Limpia el flip-flop F
14	0	CRA	Carga el contenido de la celda MD en el
15		ADD MD	acumulador
16		ROL	Desplaza a la izquierda
17		STA MD	Devuelve el multiplicando desplazado a la celda MD
18		ISZ CTR	Incrementa el contador
19		JMP LOOP	Bifurca a la instrucción de la celda LOOP. El contador no es cero
20		HLT	Para
21	CTR	FF4	Representación hexadecimal de −12
22	MD	Multiplicando	Celda para el multiplicando
23	MR	Multiplicador	Celda para el multiplicador
24	SP	XXX	Celda para el resultado (limpiado por las primeras dos instrucciones)

es necesaria para impedir colocar un 1 en la posición más a la derecha del acumulador cuando el multiplicando es desplazado a la izquierda para multiplicar por 2. La instrucción de la celda 18 incrementa el contador (CTR) y la 19 bifurca el programa a LOOP. El contenido inicial de CTR es FF4, que es la representación hexadecimal en complemento a dos de −12 (decimal).

9.7 PROGRAMA ILUSTRANDO UNA LLAMADA A SUBRUTINA

Para ilustrar una llamada a subrutina consideremos un programa para calcular la cantidad $N_1N_2+N_3N_4+N_5N_6+...$, siendo las N números positivos sin signo. Hay aquí multiplicaciones y sumas repetidas. El programa de suma es tan simple que podemos escribirlo tantas veces como se necesite. Por otra parte el programa de multiplicación es lo suficientemente grande como para instalar la rutina de la multiplicación en una subrutina de tal manera que podamos llamarla repetidamente. El programa de la tabla 9.7-1 calcula la cantidad $N_1N_2+N_3N_4$. El programa está escrito de tal manera que si queremos sumar términos adicionales (N_5N_6, etc.), hemos de proporcionar los números adicionales y los pasos de programa adicionales para hacer la suma.

Las celdas de memoria 1 a 4 son para los datos. Puesto que hemos de tener acceso a estos números, sus celdas han sido identificadas con rótulos. La celda 5, rotulada también para su acceso, simplemente almacena el número −12. Volvemos a recordar que en la rutina de la multiplicación comenzamos con el número −12 almacenando en una celda de memoria rotulada CTR. Cuando la multiplicación se ha completado, esta CTR puede ser limpiada. Así pues, necesitamos tener disponible el número −12 de forma que podamos cargarlo en CTR al comienzo de cada nueva llamada a la subrutina de la multiplicación. Las instrucciones de las celdas 6 a 11 cargan los números N_1 y N_2 en las celdas de memoria asociadas a la subrutina de la multiplicación y reservadas para el multiplicando y el multiplicador.

En la celda 12 hay una llamada a la subrutina de multiplicar. La parte de esta subrutina situada desde la celda 32 a la línea 50 es idéntica al programa de la tabla 9.6-1. La celda 28 contiene la dirección de la celda de memoria donde se encuentra la siguiente instrucción después de que la subrutina ha sido completada. No hay ninguna instrucción en ese lugar de memoria. Esa línea está rotulada (MULT), puesto que tendremos que leer su dirección para volver al programa principal. La primera vez que sea llamada la subrutina de multiplicar, la dirección colocada en esta celda (28) será la dirección 13. Para anticiparnos, debemos notar que hay una segunda llamada a subrutina en la celda 23. Cuando la instrucción de la línea 23 haya sido ejecutada, la dirección de vuelta 24 (nuevo valor de la dirección de vuelta) habrá sido almacenada en la línea 28. La instrucción de la celda 51 es una bifurcación a la celda MULT para conocer la dirección de vuelta, que nos devuelve al programa principal. En el programa de la tabla 9.6-1, donde sólo se contempló una multiplicación, la instrucción correspondiente (línea 18 de la tabla 9.6-1) es HLT.

Tabla 9.7-1 Programa en el que aparece como subrutina el programa de la multiplicación

Celda de memoria	Rótulo	Contenido	Comentario
1	A	N_1	Las celdas *A*, *B*, *C* y *D* contienen los números N_1, N_2, N_3 y N_4 para calcular $N_1N_2+N_3N_4$
2	B	N_2	
3	C	N_3	
4	D	N_4	
5	T	FF4	Representación hexadecimal de −12
6		CRA	Pone N_1 en la celda «MD», reservada para el multiplicando en la subrutina de la multiplicación
7		ADD A	
8		STA MD	
9		CRA	Pone N_2 en la celda MR, reservada para el multiplicador en la subrutina de la multiplicación
10		ADD B	
11		STA MR	
12		CSR MULT	Llama a la subrutina en la dirección «MULT»
13		CRA	Limpia la celda «PR» usada para almacenar el resultado parcial N_1N_2
14		STA PR	
15		ADD SP	Almacena el resultado parcial N_1N_2 en la celda «PR»
16		STA PR	
17		CRA	Para N_3 y N_4 en MD y MR, en donde están accesibles a la subrutina de multiplicar
18		ADD C	
19		STA MD	
20		CRA	
21		ADD B	
22		STA MR	
23		CSR MULT	Llama a la subrutina de multiplicar
24		CRA	Carga el acumulador con N_3N_4 desde la celda «SP»
25		ADD SP	
26		ADD PR	Suma N_1N_2 desde la celda «PR» para formar $N_1N_2+N_3N_4$
27		HLT	Para, ya está $N_1N_2+N_3N_4$ en el acumulador
28	MULT		Contiene la dirección de vuelta (variable)
29		CRA	Pone −12 en la celda de memoria rotulada CTR
30		ADD T	
31		STA CTR	
32		CRA	Subrutina de multiplicar
33		STA SP	
34	LOOP	ADD MR	
35		ROR	
36		STA MR	
37		SFZ	Continuación de la subrutina de multiplicar
38		JMP 1	
39		JMP 0	

Tabla 9.7-1 Programa en el que aparece como subrutina el programa de la multiplicación (*continuación*)

Celda de memoria	Rótulo	Contenido	Comentario
40	1	CRA	Continuación
41		ADD MD	de la subrutina
42		ADD SP	de multiplicar
43		STA SP	
44		CRF	
45	0	CRA	
46		ADD MD	
47		ROL	
48		STA MD	
49		ISZ CTR	
50		JMP LOOP	
51		JMP I MULT	Vuelta al programa principal
52	CTR		Celdas rotuladas de memoria, reservadas para
53	MD		el contador, el multiplicando, el multiplicador,
54	MR		la suma de productos parciales y los resultados
55	SP		parciales
56	PR		

Una vez que hemos vuelto desde la subrutina, las instrucciones de las celdas 9 a 16 limpian la celda de memoria rotulada PR (resultados parciales), donde se almacena el producto N_1N_2. Las celdas 17 a 22 cargan N_3 y N_4 en la rutina de multiplicar, y la celda 22 llama de nuevo a la subrutina. Las celdas 24 y 26 suman N_1N_2 a N_3N_4 y devuelven la suma a la celda PR. Términos adicionales N_5N_6, etc., pueden ser formados y añadidos de la misma manera y, finalmente, tendremos que añadir una instrucción HLT.

9.8 MICROPROGRAMACIÓN

En el Capítulo 8 consideramos un número de formas de diseñar un controlador. Ahora consideramos una técnica adicional de llevar a cabo el control, llamada *microprogramación*. Como veremos, una computadora controlada por microprograma es, en cierto sentido, una computadora dentro de otra. Por esta razón hemos aplazado la descripción del método hasta que hemos tenido alguna experiencia elemental con una estructura del tipo computadora.

Hemos visto que las microoperaciones de nuestra computadora tienen lugar en respuesta a señales aplicadas a los terminales de control de los registros de la ALU y de la memoria. Estas señales así aplicadas se llaman *señales de mando* (*command signals*). Durante algún intervalo de reloj deben ser elevadas a nivel lógico 1 una o más entradas de control por las señales de mando aplicadas.

Refiriéndonos ahora a nuestra máquina de la figura 9.1-1 podemos

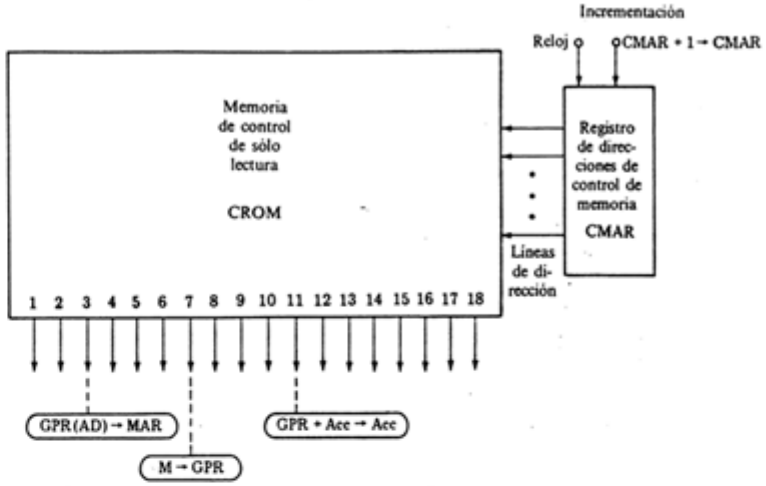


Figura 9.8-1 Las palabras leídas sucesivamente de una ROM de control proporcionan la secuencia de señales de mando requeridas para controlar un procesador.

verificar que hay 18 entradas terminales de control que, en uno u otro momento, requerirán una señal de mando para llegar a estar activas. Imaginemos una memoria de sólo lectura, como la de la figura 9.8-1, en la que las palabras tienen 18 bits de longitud, de forma que podemos establecer una asociación uno a uno entre los bits de la palabra y las entradas de control de los terminales. En la figura 9.8-1 hemos asignado, sobre una base enteramente arbitraria, el bit 3 de la palabra de memoria al terminal de control que, cuando está activado, realiza la microoperación GPR(AD) →MAR. Similarmente hemos asignado el bit 7 a M→MBR y el bit 11 a GPR+Acc→Acc. La salida del bit 3 de la ROM es conectada al terminal de salidas. Veamos ahora, como de ejemplo, cómo podemos arreglárnoslas para usar la ROM para generar en la secuencia adecuada las microoperaciones necesarias para ejecutar una instrucción.

Consideremos específicamente la instrucción ADDI, cuyas microoperaciones, cinco en número, se detallan en la página 387. Consideremos que hemos escrito en la ROM, en cinco celdas sucesivas, la secuencia de palabras listada en la figura 9.8-2. Y finalmente supongamos que el registro de direcciones de la figura 9.8-1 se inicializa con la dirección de la primera palabra y es incrementado para las restantes direcciones en sincronismo con el reloj que conduce el ordenador de la figura 9.1-1. Durante el primer ciclo de reloj sólo el bit 3 está en 1 lógico, así que sólo el terminal GPR(AD)→MAR se hace activo, cuando se necesita. En el segundo intervalo de reloj sólo el bit 7 está en 1 lógico, dando lugar a que se realice la microoperación M→GPR. En el tercer intervalo de reloj de nuevo tiene lugar GPR(AD)→MAR. En el cuarto

Palabra	Número de bit																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figura 9.8-2 Las cinco micropalabras sucesivas de la ROM de la figura 9.8-1 que proporcionan las microinstrucciones para causar la ejecución de las microoperaciones requeridas por la instrucción ADDI.

intervalo el bit 7 está de nuevo en 1 lógico y de nuevo se realizará la microoperación M→GPR. Finalmente en el quinto intervalo de reloj tendrá lugar GPR + Acc→Acc. En este ejemplo, sucede que sólo se necesita que se ejecute una microoperación en cada ciclo de reloj. Así pues, en cada palabra de CROM solamente un bit está en 1 lógico. Cuando se necesitan ejecutar varias microoperaciones en un mismo ciclo de reloj, el correspondiente número de bits estarán en 1 lógico.

Así pues, leyendo conjuntamente una sucesión de palabras llamadas *palabras de control* o *micropalabras*, realizamos una secuencia de *microoperaciones* almacenadas en la memoria de sólo lectura y el resultado final es que se ejecuta la operación que se requiere. Las palabras que identifican las microoperaciones en la ROM se llaman también *microinstrucciones* (en oposición a las *instrucciones*, que se almacenan en la RAM de lectura y escritura de la computadora). Para quitar ambigüedad, las instrucciones de una máquina pueden llamarse *macroinstrucciones* para distinguirlas de las microinstrucciones almacenadas en la ROM de control. Una computadora cuyo controlador opera como aquí se ha descrito se llama *computadora microprogramada*. Notar que una tal computadora tiene dos memorias: 1) como todas las computadoras tiene una RAM, que contiene las instrucciones y los datos, y 2) tiene un ROM en su controlador, que contiene las macroinstrucciones para ejecutar las instrucciones. Esta segunda memoria es generalmente llamada *memoria de sólo lectura de control* (*control read-only memory*, CROM), y consiguientemente su registro de direcciones se llama *registro de direcciones de la memoria de control* (*control memory address register*, CMAR).

9.9. BIFURCACIÓN EN MICROPROGRAMAS

El sencillo esquema de microprogramación de la sección precedente describe el principio de la microprogramación, pero no es lo suficientemente versátil para ser útil en una situación real. En esta sección y la siguiente consideraremos algunas elaboraciones.

En la disposición de la figura 9.8-1 somos capaces de leer microinstrucciones solamente en el orden en que han sido escritas en memoria. Una disposición que permite la bifurcación se muestra en la figura 9.9-1. Aquí suponemos que hay 9 bits de mando para proporcionar N entradas a los terminales de control de los registros del ordenador, y convenimos en que el número de microinstrucciones para ser almacenadas es tal que se requiere una dirección de memoria de control de M bits. Así pues, hemos dispuesto en la ROM que la palabra sea de $N + M + 1$ bits. N bits son bits de mando, y M bits constituyen la dirección a la que hay que bifurcar cuando ocurra una bifurcación. El bit extra se usa para indicar cuándo se invoca una bifurcación o no. Este bit extra es el *bit de control de carga* (*load control bit*). Si este bit está en 0 lógico, la entrada *incremento* está en 1 lógico y el registro de direcciones será incrementado en cada intervalo de reloj. Si, por el contrario, el bit de control de carga está en 1 lógico, no tendrá lugar el incremento. En su lugar tendremos un 1 lógico en el terminal de entrada de dirección de bifurcación y carga del registro de direcciones, y durante el flanco de los impulsos de reloj la dirección de M bits a la que queremos bifurcar será cargada en el registro de direcciones. Por supuesto, el registro de direcciones de la figura 9.9-1 es, por lo que se refiere a su implementación hardware, más complicado que el registro de la figura 9.8-1. El primer registro debe ser capaz solamente de incrementar, mientras que el segundo debe incrementar o aceptar una carga paralela.

En la ROM de control del controlador microprogramado de la figura 9.9-1 hemos proporcionado suficientes bits por palabra para que una palabra contenga una dirección así como una microinstrucción. Está proliferación de bits es algo generalmente prohibitivo cuando se establece la longitud de palabra de una memoria RAM. Hay dos buenas razones a favor de esta extensión en la longitud de palabra: 1) las RAM son más caras que las ROM de igual tamaño, y 2) una palabra más larga en la RAM, que contiene

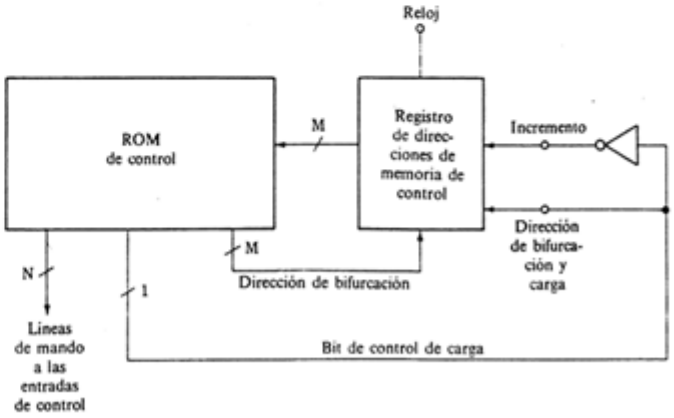


Figura 9.9-1 Una modificación de la disposición de la figura 9.8-1 que permite no sólo incrementar la dirección de la CROM, sino también bifurcar a una dirección especificada por la micropalabra anterior.

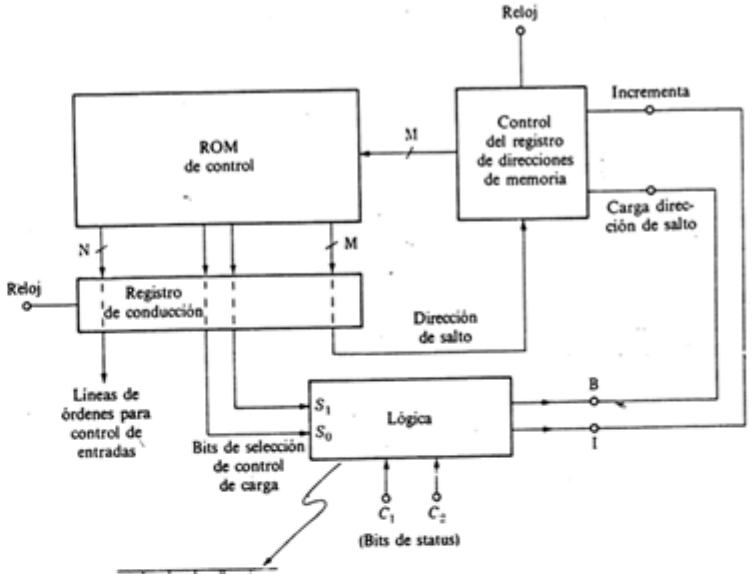
instrucciones (más bien de microinstrucciones) y datos, requeriría consiguientemente una longitud de palabra más larga en casi todos los registros del ordenador. Pero no es tal el caso cuando se aumenta la longitud de palabra de la ROM. Por ello generalmente se observa que mientras los diseñadores añadan longitud al tamaño de la palabra para instrucciones y datos sólo después de una deliberada decisión de construir un ordenador más costoso, los bits para la memoria de control se añaden con bastante libertad. Actualmente el control microprogramado está llegando a ser considerado más favorablemente. Es la manera más sistematizada y ordenada de diseñar un controlador, y las ROM requeridas en su implementación hardware están disponibles fácilmente y son baratas.

9.10 BIFURCACIÓN CONDICIONAL

Consideramos a continuación la disposición de la figura 9.10-1. El *registro de conducción* mostrado no es relevante en nuestra discusión actual y se incluyó en la figura para poder utilizarla nuevamente en la discusión de la siguiente sección. Entonces de momento ignoramos el registro de conducción y consideramos las líneas a trazos a través de este registro simplemente como continuación de las líneas de salida de la ROM de control.

La disposición de la figura 9.10-1 introduce en el controlador flexibilidad adicional. Disponemos de dos bits de selección de control de carga S_1 y S_0. Los bits C_1 y C_2 son *bits de status* que derivan de algún sitio de la computadora y se proponen para indicar si se satisface o no alguna condición. Por ejemplo, el bit C_1 puede ser la entrada del controlador Z de la figura 9.1-1. Si $C_1(=Z)$ es $C_1=1$, el registro de propósito general se borra. Si $C=0$ este registro no se borra. Así el bit C_1 da información acerca del status del GPR. Análogamente, C_2 puede dar información del status de otras componentes de la computadora. Así, refiriéndonos de nuevo a la figura 9.1-1, C_2 puede ser F. Dichos bits de status cuando hacen disponible al controlador, le permiten seguir una secuencia u otra dependiendo del nivel lógico del bit de status.

El bloque de *lógica* de la figura 9.10-1 es un circuito combinacional cuya tabla de verdad se dio (ver Prob. 9.10-1); cuando $S_1S_0=00$, la salida del bloque lógico I = 1 y la salida B = 0. Suponemos que los terminales de control del registro de direcciones de memoria del control se activan en 1 lógico. Por tanto, cuando $S_1S_0=00$, el registro se incrementará. Análogamente, cuando $S_1S_0=01$ el registro bifurcará a la dirección especificada por el campo de dirección de la micropalabra cuya instrucción se está implementando. En estos casos el bit de status se ignora. Cuando $S_1S_0=10$, *el bit de status C* determina si tiene lugar un incremento o una bifurcación, mientras que cuando $S_1S_0=11$ el *bit de status* C_2 determina si tiene lugar el incremento o la bifurcación. (Como I y B son siempre complementarios entre sí, debíamos haber permitido exactamente una línea de salida del bloque lógico y dispuesto la complementariedad como en la figura 9.9-1. No lo hemos hecho así, ya que en el último punto querremos hacer modificaciones que permitan que I y B sean 0 lógico simultáneamente.)



S_1	S_0	C_1	C_2	B	I	
0	0	×	×	0	1	Incrementa: ignora los bits de status
0	1	×	×	1	0	Salta: ignora los bits de status
1	0	0	×	0	1	Responde sólo a C_1: incrementa si $C_1=0$; salta si $C_1=1$
1	0	1	×	1	0	
1	1	×	0	0	1	Responde sólo a C_2: incrementa si $C_2=0$; salta si $C_2=1$
1	1	×	1	1	0	

Figura 9.10-1 Modificación adicional de un controlador microprogramado que permite que la operación del controlador se vea influenciada por el bit de status.

9.11 CONDUCCIÓN (PIPELINING)

En esta sección veremos cómo la adición del *registro de conducción* sirve para incrementar la velocidad a la cual puede operar el controlador microprogramado. Consideremos primero, en *ausencia* de este registro, los retardos asociados al controlador y al sistema que controla. Supongamos que ocurre una transición de disparo en $t=t_0$. El instante t_0 es el comienzo de la operación de establecer una nueva dirección en el registro de direcciones de memoria de control, bien incrementando o bien cargando. ¿Cuánto debemos esperar ahora antes que podamos permitir la siguiente transición de reloj? Primero debemos esperar un tiempo t_A para permitir que se establezca una dirección válida en la salida del registro de direcciones. (La dirección es válida

después que todas las M líneas de salida del registro de direcciones presenten el nuevo bit propio a la ROM y no ocurran más cambios.) A continuación debemos esperar el tiempo adicional t_M del retardo de propagación a través de la memoria de control. Después de un tiempo t_A+t_M la microinstrucción direccionada se establecerá válidamente en la salida de la memoria de control. Ahora debemos esperar para que el sistema controlado responda a las microinstrucciones, complete su respuesta y esté preparado para la siguiente instrucción. Si llamamos t_R al tiempo de respuesta del sistema, el período de reloj mínimo permitido es $t_A+t_M+t_R$. El mayor de estos tiempos componentes es t_R, que debe ser bastante grande para acomodar la microoperación más lenta, generalmente la operación de leer, o escribir en una RAM.

El proceso de establecer una nueva micropalabra de salida de ROM comienza en el instante de la ocurrencia de la transición de disparo del reloj cuando el registro comienza a incrementar o a cargar una dirección de salto. Ahora nos ocurre que debemos poder ahorrar tiempo para *comenzar* a establecer una nueva micropalabra antes que el sistema controlado haya completado su respuesta a la micropalabra previa.

Sea t_X el tiempo de retardo de propagación de la caja lógica de la figura 9.10-1 y sea el período de reloj $t_C=t_A+t_M+t_X$. Por ejemplo, en $t=t_0$ un flanco de reloj inicia al registro de direcciones en el proceso de incrementar y cargar una dirección de bifurcación. En $t=t_0+t_A+t_M$ se establecerá una nueva micropalabra válida en la salida de la ROM y en $t=t_0+t_A+t_M+t_X=$ $=t_0+t_C$ se ha suministrado un bit de control de carga válido al registro de direcciones para que pueda ocurrir el siguiente flanco de disparo de reloj. Con el período de reloj puesto en $t_C=t_A+t_M+t_X$ aparecerá una sucesión de nuevas micropalabras válidas a la salida de la ROM a la frecuencia del reloj. Supongamos ahora que el tiempo de respuesta del sistema t_R es menor que el período de reloj. Entonces el sistema controlado tendrá tiempo adecuado para responder a cada microinstrucción. Por ejemplo, supongamos que el período de reloj es $t_C=t_A+t_M+t_X=2+3+1=6\,\mu$ y que la respuesta del sistema es $t_R=4\,\mu$. Cada $6\,\mu$ se presentarán nuevas microinstrucciones al sistema controlado y éste completará su respuesta en $4\,\mu$. Flancos de reloj en $t=0$, 6, $12\,\mu$ generarán microinstrucciones válidas en $t=6$, 12, $18\,\mu$. El sistema responderá a la microinstrucción que se presente en $t=6$ durante el intervalo de $t=6$ a $t=10$. El sistema entonces espera a recibir su siguiente microinstrucción en $t=12$ y realiza su respuesta correspondiente de $t=12$ a $t=16$, y así sucesivamente. El punto importante es que se ahorra tiempo, ya que el sistema da su respuesta a una microoperación *al mismo tiempo* que la siguiente microoperación se está abriendo camino en la línea de conducción, es decir, a través de la caja lógica combinacional el registro de dirección y la ROM. El período de reloj es $6\,\mu$ menor que $10\,\mu$, como se esperaría que la respuesta del sistema se complete antes que comencemos a generar una nueva microoperación.

El argumento del parágrafo anterior, que indica cómo puede aumentarse la velocidad del reloj, realmente *no* es *válido*. Allí se asume que si en $t=t_0$ ocurre una transición de disparo aparecerá una nueva micropalabra de salida

de la ROM en $t_0+t_A+t_M$, que en el intervalo comprendido entre t_0 y t_0+t_A+ $+t_M$ no ocurrirá *ningún cambio* en la salida de la ROM, y que durante el intervalo la salida de la ROM contendrá la micropalabra previa. Este no es el caso. Es cierto que necesitamos esperar un tiempo t_A+t_M para asegurar que han tenido lugar *todos* los cambios en la salida de la ROM que están ocurriendo. Sin embargo, algunos cambios pueden tener lugar después de un intervalo mucho más corto. Así, mientras podamos requerir $5\,\mu$ para que se establezca completamente válida una nueva salida de ROM, algunas salidas pueden cambiar en tiempos del orden de $0{,}1\,\mu$. Como no debemos permitir cambios en la salida de la ROM hasta que el sistema haya completado su respuesta, debemos esperar que se complete una respuesta del sistema antes que *comencemos* el proceso de cambio de dirección.

Esta dificultad se minimiza añadiendo el *registro de conducción* indicado en la figura 9.10-1, como ahora veremos. Realmente este registro no constituye la conducción, ya que es el retardo de propagación por el bloque lógico, registro de dirección y la ROM, que son análogos a una línea de conducción. El registro sirve como una válvula al fin de la línea permitiéndola operar con efectividad. Este registro está gobernado por la misma señal de reloj que gobierna el registro de dirección. La transición de reloj que incrementa o carga el registro de dirección es la misma transición que carga el registro de conducción con la micropalabra de salida de la ROM. La respuesta del sistema controlado puede ahora comenzar en el instante que se cargue el registro de conducción. *En este mismo momento* podemos comenzar a cambiar la dirección con el fin de llamar la siguiente microinstrucción. Mientras el sistema microcontrolado está completando la respuesta, los bits de la palabra de salida de la ROM pueden estar cambiando, pero estos cambios no provocarán problemas, ya que el sistema está separado de la microinstrucción cambiante debido al aislamiento proporcionado por el registro de conducción.

9.12 CONTROLADOR MICROPROGRAMADO

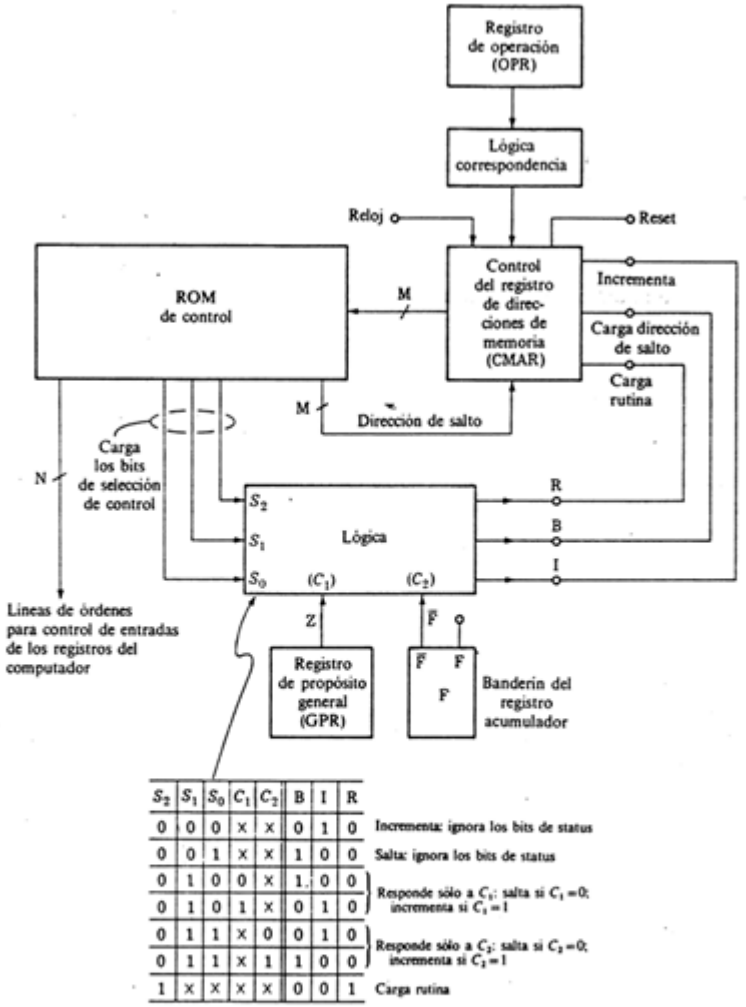
En la computadora de la figura 9.1-1 incluimos un controlador para suministrar órdenes de habilitación a todos los registros componentes para que cada uno de ellos también realice una de las microoperaciones de su repertorio cuando se requiera. Para ilustrar el principio de microprogramación consideraremos ahora cómo puede realizarse un controlador microprogramado. En la figura 9.12-1 se muestra el controlador microprogramado que se ha modelado después del controlador de la figura 9.10-1. Por simplicidad hemos omitido la línea de conducción, ya que ésta no afecta el principio de operación, solamente su velocidad.

En un diagrama del sistema computador completo (figura 9.1-1) señalamos que el controlador tiene entradas del registro de operación (OPR), del registro de propósito general (GPR) y del banderín (flag) del acumulador F. Estos dos registros y el de banderín se incorporan a la figura 9.12-1. El bit de condición de bifurcación C_1 de la figura 9.10-1 es el bit Z suministrado por el GPR. Este bit da información relativa al *status* del GPR. Cuando el registro

en GPR es cero, $Z=1$; de otra forma, $Z=0$. Análogamente la línea lógica F, sustituyendo a C_2, suministra un bit de *status* al controlador para informar si el flip-flop del banderín F está en set o reset. En el controlador de la figura 9.10-1 hacemos previsión bien para incrementar el registro de direcciones de memoria (CMAR) o para cargar en él la dirección de bifurcación. En el caso presente hemos previsto también cargar en el CMAR una dirección determinada por el código de operación registrado en el registro de operación. (La dirección real se determina por el OPR y el diagrama lógico combinacional, como veremos.) La orden para cargar esta dirección-determinada-del registro de operación se denomina «rutina de carga». Notar que el campo de selección de control de carga consta de 3 bits en lugar de dos como en la figura 9.10-1. Ahora discribiremos como opera el controlador microprogramado.

La secuencia de microoperaciones (escritas en la ROM de control como una secuencia de micropalabras) que realizan los pasos necesarios para buscar una instrucción se denomina *rutina de búsqueda*. Análogamente una secuencia que provoque la ejecución de una instrucción se denomina *rutina de ejecución*. En alguna posición en la ROM de control, por ejemplo, comenzando en la posición que llamaremos ADDR(FETCH), escribiremos la rutina de búsqueda. El controlador gobernará la secuencia a través de la rutina de búsqueda, incrementando el CMAR después de cada microoperación. Al fin de la rutina de búsqueda, la operación indicada por la instrucción buscada habrá sido cargada en el registro de operación. Supongamos, por ejemplo, que la instrucción buscada es ADDI. La rutina para ejecutar la instrucción ADDI se cargará en la ROM de control comenzando en la posición ADDR(ADDI). Además es necesario al finalizar la rutina de búsqueda que haya un *salto* a la rutina ADDI que comienza en ADDR(ADDI). Esta dirección ADDR(ADDI) se determina, por supuesto, por el *código de operación* en el registro de operación. Sería muy conveniente si pudiésemos disponer que el código de OP de ADDI fuese el mismo que la dirección ADDR(ADDI). Si éste fuera el caso, podría realizarse una *transferencia* a la rutina ADDI cargando el código en el CMAR. Sin embargo, generalmente, no es factible disponer de esta característica conveniente y además es necesario interponer entre el registro de operación y el CMAR un bloque combinacional apropiado que acepte el código de OP como entrada y suministre como salida la dirección de comienzo de la rutina de instrucción. Esta traducción del código de OP a la dirección de comienzo de una rutina se denomina *correspondencia* (*mapping*) y en la figura 9.12-1 se realiza por el bloque combinacional de lógica-correspondencia (que posiblemente puede ser una ROM). En cualquier situación ahora es necesario suministrar la facultad no sólo de *incrementar* el CMAR y cargarlo con la dirección de *salto*, sino también de cargarlo con la rutina de ejecución cuyo código de OP está en el OPR. Ya hemos incorporado exactamente esta facultad en el controlador de la figura 9.12-1.

La tabla de verdad para el bloque lógico de la figura 9.12-1 es una elaboración de la tabla de verdad de la figura 9.10-1; esto se permite, ya que hemos suministrado un bit adicional de selección del control de carga S_2. Así



S_2	S_1	S_0	C_1	C_2	B	I	R	
0	0	0	×	×	0	1	0	Incrementa: ignora los bits de status
0	0	1	×	×	1	0	0	Salta: ignora los bits de status
0	1	0	0	×	1	0	0	Responde sólo a C_1: salta si $C_1=0$; incrementa si $C_1=1$
0	1	0	1	×	0	1	0	
0	1	1	×	0	0	1	0	Responde sólo a C_2: salta si $C_2=0$; incrementa si $C_1=1$
0	1	1	×	1	1	0	0	
1	×	×	×	×	0	0	1	Carga rutina

Figura 9.12-1 Una elaboración adicional del controlador microprogramado que permite cargar en el CMAR la dirección CROM de comienzo de una subrutina.

durante el tiempo que $S_2=0$, R = 0 y el resto de la tabla de verdad de la figura 9.12-1 es la misma que la de la figura 9.10-1. Cuando $S_2=1$, R = 1 y la dirección de la rutina de ejecución de la instrucción especificada por el registro de operación se cargará en el CMAR.

9.13 CONTROL DEL CONTENIDO DE LA ROM

Consideremos ahora lo que sería aconsejable que contuviera la ROM de control de un controlador microprogramado, de nuestra sencilla computadora de la figura 9.1-1. Estos contenidos se indican en la figura 9.13-1. Los bits de las micropalabras se clasifican en tres clases o campos. El *campo de operación* especifica la microoperación que se realiza. El *campo de selección de dirección* especifica cómo el controlador selecciona la dirección en la ROM de control donde se encuentra la siguiente microoperación, esta dirección siguiente se encontrará en el *campo de la dirección siguiente*. Cuando la siguiente microinstrucción se selecciona incrementando el CMAR o cargando en ese registro una dirección determinada por el registro de operación, el contenido del siguiente campo de dirección será irrelevante.

En las posiciones ADDR(FETCH), ADDR(FETCH)+1 y ADDR (FETCH)+2 de la ROM de control encontramos microoperaciones que realizan la rutina de búsqueda (ver página 384). Después de la primera microoperación y también después de la segunda, el registro de dirección se incrementa; por tanto, encontramos en estos dos primeros casos que el campo de dirección lee $S_2S_1S_0=000$. Después que la tercera microoperación en la posición ADDR(FETCH)+2 se ha realizado, el registro de operación

Dirección de memoria	Campo de operación	Campo de selección de dirección			Campo de la dirección siguiente
		S_2	S_1	S_0	
ADDR(FETCH)	PC → MAR	0	0	0	
ADDR(FETCH) + 1	M → GPR; PC + 1 → PC	0	0	0	
ADDR(FETCH) + 2	GPR(OP) → OPR	1	x	x	
⋮					
ADDR(ADDI)	GPR(AD) → MAR	0	0	0	
ADDR(ADDI) + 1	M → GPR	0	0	0	
ADDR(ADDI) + 2	GPR(AD) → MAR	0	0	0	
ADDR(ADDI) + 3	M → GPR	0	0	0	
ADDR(ADDI) + 4	GPR + Acc → Acc	0	0	1	ADDR (FETCH)
⋮					
ADDR(ISZ)	GPR (AD) → MAR	0	0	0	
ADDR(ISZ) + 1	M → GPR	0	0	0	
ADDR(ISZ) + 2	GPR + 1 → GPR	0	0	0	
ADDR(ISZ) + 3	GPR → M	0	1	0	ADDR(FETCH)
ADDR(ISZ) + 4	PC + 1 → PC	0	0	1	ADDR(FETCH)
⋮					
ADDR(SFZ)	NOP	0	1	1	ADDR(FETCH)
ADDR(SFZ) + 1	PC + 1 → PC	0	0	1	ADDR(FETCH)

Figura 9.13-1 Contenido parcial de la CROM de la figura 9.12-1 si el controlador se utilizase con la computadora de la figura 9.1-1.

(cuyo contenido se decodifica por la correspondencia lógica) contiene la dirección CROM del comienzo de la rutina de instrucción que se va a ejecutar. Por tanto, en este caso se activa la entrada de la rutina de carga del registro de dirección y consecuentemente encontramos que $S_2S_1S_0=1\times\times$.

Por ejemplo, si la instrucción así cargada en el registro de dirección fuese la instrucción ADDI, a continuación iríamos a través de la secuencia ADDR(ADDI), ADDR(ADD)+1, etc., que constituye la rutina para ejecutar la instrucción ADDI. Cuando se ha ejecutado la instrucción, debemos volver a la rutina de búsqueda. Por tanto, la última micropalabra de la rutina ADDI tiene el código de selección de dirección $S_2S_1S_0=001$ (salta a la dirección indicada en el campo de dirección) y el campo de dirección contiene la dirección ADDR(FETCH). Generalmente, de esta misma forma la última micropalabra de cada rutina de ejecución de instrucción provocará un salto atrás de ADDR(FETCH).

A continuación consideremos la rutina de ISZ (incrementa palabra de memoria y salta a la siguiente instrucción si el número incrementado tiene el valor cero). Esta instrucción es especial, ya que involucra el bit de status Z (también llamado C_1). La rutina para esta instrucción está escrita en la figura 9.13-1, comenzando en la posición de la ROM ADDR(ISZ) (ver también página 392). Las 3 primeras micropalabras tienen el campo de selección de dirección $S_2S_1S_0=000$ para incrementar el CMAR. El GPR se incrementa durante una tercera microoperación. Después de que esta operación de incrementación ha sido completada, sabremos si se borra el GPR, en cuyo caso Z=1, o no se borra, en cuyo caso Z=0. La cuarta microoperación tiene el campo de selección de dirección 010, que indica que la dirección de la siguiente microoperación se determina por Z (C_1). Si Z=0, habrá un salto atrás a ADDR(FETCH) como indicaba la tabla de verdad de la figura 9.12-1. Si Z=1, el CMAR no se cargará con la dirección de bifurcación pero en vez de ello se incrementará. Si se produce dicho incremento se ejecutará la quinta microoperación: PC+1→PC y entonces habrá un salto a ADDR(FETCH).

Consideremos a continuación la instrucción SFZ (salta a la siguiente macroinstrucción si F=0). Como puede verse en la figura 9.12-1 hemos empleado $\bar{F}$ como el bit de status C_2, así que $C_2=1$ cuando F=0. Las microoperaciones requeridas para ejecutar SFZ están listadas en la figura 9.13-1 comenzando en ADDR(SFZ). La primera «microoperación» es realmente la no operación (NOP). Se realiza disponiendo que todos los bits de órdenes de la ROM de control sean 0. Durante el intervalo de NOP, el controlador tendrá tiempo para determinar si C_2 ($=\bar{F}$) es 0 ó 1. El campo de selección de dirección en esta micropalabra es $S_2S_1S_0=011$, que como muestra la tabla de verdad de la figura 9.12-1, indica que el controlador está para responder a C_2. Si $C_2=0$ (F=1) habrá un salto a la dirección del campo de dirección. El resultado será que no se habrá realizado operación y el controlador volverá a la rutina de búsqueda. Sin embargo, si $C_2=1$, el CMAR será incrementado a ADDR(SFZ)+1. En esa posición encontramos la microinstrucción PC+1→PC, así que el contador de programa se incrementará y entonces comenzará la rutina de búsqueda. El resultado final es que una microinstrucción habrá sido saltada.

Pueden escribirse otras instrucciones en la memoria de control siguiendo el mismo patrón que el aplicado a las instrucciones de la figura 9.13-1. Los detalles de dejan como ejercicio de estudiantes.

9.14 MÉTODOS DE DIRECCIONAMIENTO

Hemos visto que generalmente (aunque no necesariamente) una instrucción consta de una parte operación y otra de dirección. La parte dirección puede contener la dirección de un operando utilizado en la ejecución de la instrucción. En otras ocasiones la parte dirección de la instrucción puede contener no la dirección del operando, sino la dirección donde se encuentra la dirección del operando. En el primer caso la dirección se describe como *dirección directa;* en el segundo caso es una *dirección indirecta*. En computadoras, minicomputadoras y microcomputadoras se emplea una amplia gama de modos de direccionamiento, de los que consideraremos algunos en esta sección.

Directo. En el *direccionamiento directo*, como ya señalamos, la instrucción contiene la dirección de la posición de memoria donde se encuentra el operando.

Indirecto. En el *direccionamiento indirecto*, señalamos de nuevo, la instrucción contiene no la posición de memoria donde se encuentra el operando, sino la dirección de la posición de memoria donde se encuentra la dirección del operando.

Relativo. En el *direccionamiento relativo* la parte dirección de la instrucción contiene un número N. En memoria la dirección del operando se encuentra sumando el número N al número del contador de programa. Por ejemplo, si el contador de programa (PC) contiene el número 17 y si $N=14$, la dirección del operando es $PC+N=17+14=31$. El número N puede ser positivo, como en nuestro ejemplo, o negativo, así que la dirección efectiva $PC+N$ puede ser mayor o menor que la dirección del contador de programa.

Indexado. En el *direccionamiento indexado* como en el relativo la parte dirección de la instrucción contiene un número N que puede ser positivo o negativo. Sin embargo, para utilizar el direccionamiento indexado el computador debe estar equipado con un registro especial (distinto del contador de programa) empleado para permitir direccionamiento indexado, y denominado naturalmente *registro índice* (I). La posición de memoria donde se localiza el operando se encuentra mediante la suma $I+N$. (En la computadora sencilla de la figura 9.1-1 no tenemos registro índice y no se ha hecho previsión para direccionamiento indexado.)

Registro indirecto. Algunas computadoras que incorporan la facultad del *direccionamiento de registro indirecto* tienen un registro especial, a menudo

llamado *registro puntero* (P) (*pointer register*). Este registro P contiene la dirección del operando; es decir, apunta a la posición de memoria del operando. Una instrucción que invoque realmente direccionamiento de registro indirecto no tiene bits significativos en su parte dirección. En lugar de ello, la instrucción completa se incluye en los bits asignados a la parte de operación de la instrucción. Una instrucción típica que use registro de direccionamiento indirecto debería especificar «cargar el acumulador con el operando localizado en la dirección de memoria dada en el registro P».

Otros esquemas comunes para localizar fuentes de operandos o destinos de operandos, denominados *modos de direccionamiento*, incluyen los siguientes:

Inmediato. En el *direccionamiento inmediato* la parte dirección de la instrucción contiene no la dirección del operando, sino el mismo operando. Así, la instrucción «cargar acumulador *directo* con 37 significa cargar el acumulador con el contenido de la posición de memoria 37. Por otro lado, «cargar el acumulador *inmediato* con 37» significa cargar el acumulador con el número 37.

Inherente. Ordinariamente una dirección que es parte de una instrucción se refiere a una posición de memoria. Cuando una instrucción indica una fuente o un destino de algunos datos y no se direcciona específicamente, ya que no se hace referencia a la posición de memoria, se dice que la instrucción tiene una *dirección inherente*. Por ejemplo, en la instrucción «borrar el acumulador» los «datos» movidos están en una palabra cuyos bits son todos 0 y la «dirección» de destino de estos datos es el registro acumulador. De nuevo en la instrucción mover el contenido del registro R1 al registro R2, R1 y R2 son las «direcciones» de la que se lee una palabra y en la que se escribe la palabra.

9.15 PILAS

En el acceso a la información almacenada en memoria hemos señalado ya, cuando es posible, que es útil tener conocimiento de la dirección de la siguiente palabra que va a ser leída o escrita. Así encontramos muy útil almacenar instrucciones consecutivas en posiciones consecutivas de memoria, que indicaban que sólo infrecuentemente teníamos que confeccionar la dirección de la siguiente instrucción. Excepto cuando se encuentran instrucciones de salto, el suministro de la dirección de la siguiente instrucción requiere solamente que se incremente un contador, el contador de programa.

De forma similar es muy útil (cuando es factible hacerlo así) almacenar los datos en memoria, de forma tal que la nueva dirección pueda establecerse simplemente incrementando (o decrementando) un contador. Dicha memoria se denomina pila. Ya hemos considerado pilas en la sección 6.16, donde señalamos la razón para el uso del nombre de pila y por qué a la escritura de una palabra en una pila se denominaba introducir y a la lectura sacar.

Cualquier array de registros puede ser utilizado como pila. Habitualmen-