

## CAPÍTULO 16

---

# Funcionamiento de la unidad de control

### **16.1. Microoperaciones**

- El ciclo de captación
- El ciclo indirecto
- El ciclo de interrupción
- El ciclo de ejecución
- El ciclo de instrucción

### **16.2. Control del procesador**

- Requisitos funcionales
- Señales de control
- Un ejemplo de señales de control
- Organización interna del procesador
- El Intel 8085

### **16.3. Implementación cableada**

- Entradas de la unidad de control
- Lógica de la unidad de control

### **16.4. Lecturas recomendadas**

### **16.5. Palabras clave, preguntas de repaso y problemas**

- Palabras clave
- Preguntas de repaso
- Problemas

## PUNTOS CLAVE

- La ejecución de una instrucción implica la ejecución de una secuencia de pasos más pequeños, normalmente llamados ciclos. Por ejemplo, una ejecución puede constar de ciclos de captación, acceso indirecto a memoria, ejecución e interrupción. Además, cada ciclo se compone de una serie de operaciones más elementales, llamadas microoperaciones. Una única microoperación por lo general implica una transferencia entre registros, una transferencia entre un registro y un bus externo, o una operación sencilla de la ALU.
- La unidad de control de un procesador realiza dos tareas: (1) hace que el procesador ejecute las microoperaciones en la secuencia correcta, determinada por el programa que se está ejecutando, y (2) genera las señales de control que provocan la ejecución de cada microoperación.
- Las señales de control generadas por la unidad de control causan la apertura y el cierre de ciertas puertas lógicas, lo que da como resultado una transferencia de datos hacia, o desde, los registros, y una operación de la ALU.
- Una técnica para construir la unidad de control es la implementación cableada, en la cual dicha unidad es un circuito combinacional. Sus señales lógicas de entrada, gobernadas por la instrucción máquina en curso, se transforman en un conjunto de señales de control de salida.

**E**n el Capítulo 10, señalamos que el conjunto de instrucciones máquina contribuye en gran medida a definir el procesador. Si conocemos el conjunto de instrucciones máquina, lo que incluye una comprensión del efecto de cada código de operación y de los modos de direccionamiento, y si conocemos el conjunto de registros visibles por el usuario, entonces conocemos las funciones que puede realizar el procesador. Esta no es una descripción completa. Necesitamos conocer las interfaces externas, por lo general accesibles a través de un bus, y cómo se manejan las interrupciones. Siguiendo esta línea de razonamiento, surge la siguiente lista de conceptos necesarios para especificar la funcionalidad de un procesador:

1. Operaciones (códigos de operación).
2. Modos de direccionamiento.
3. Registros.
4. Interfaz con el módulo de E/S.
5. Interfaz con el módulo de memoria.
6. Estructura del procesamiento de interrupciones.

Esta lista, aunque general, es bastante completa. Los puntos del 1 al 3 quedan definidos por el conjunto de instrucciones. Los puntos 4 y 5 vienen determinados típicamente por el bus del sistema. El punto 6 está definido parcialmente por el bus del sistema y parcialmente por el tipo de apoyo que ofrece el procesador al sistema operativo.

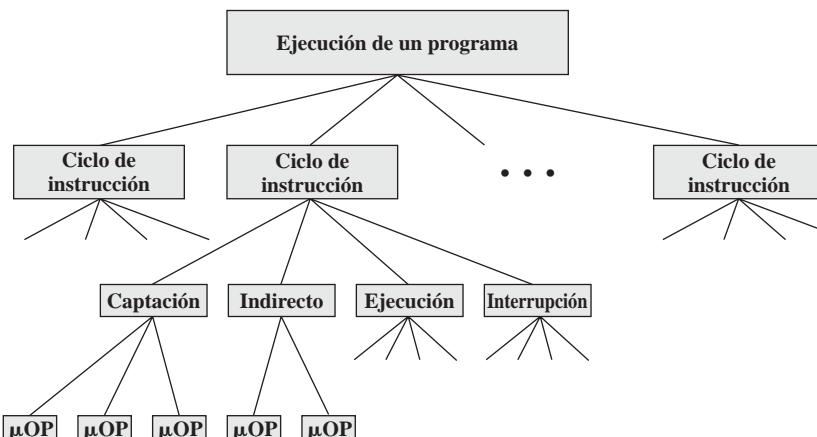
Los seis puntos de esta lista podrían denominarse requisitos funcionales de un procesador. Ellos determinan lo que debe hacer el procesador. Nos ocupamos de esto en las Partes Dos y Tres. Ahora nos vamos a centrar en la cuestión de cómo se realizan esas funciones o, más específicamente, cómo se controlan los diversos elementos del procesador para proporcionar esas funciones. Por tanto, vamos a estudiar la unidad de control, que controla el funcionamiento del procesador.

## 16.1. MICROOPERACIONES

Hemos visto que el funcionamiento de un computador, cuando ejecuta un programa, consiste en una secuencia de ciclos de instrucción, con una instrucción máquina por ciclo. Naturalmente, debemos recordar que esta secuencia de ciclos de instrucción no es necesariamente la misma que la *secuencia escrita* de instrucciones que constituye un programa, debido a la existencia de instrucciones de salto. A lo que nos referimos aquí es a la *secuencia temporal* de ejecución de instrucciones.

Hemos visto además que cada ciclo de instrucción puede considerarse compuesto por varias unidades más pequeñas. Una subdivisión práctica es captación, ciclo indirecto, ejecución e interrupción, si bien solo aparecen siempre los ciclos de captación y de ejecución.

Para diseñar una unidad de control, no obstante, necesitamos desglosar más esta descripción. En nuestra discusión sobre la segmentación en el Capítulo 12, comenzamos a ver que es posible una mayor descomposición. En realidad, veremos que cada uno de los ciclos más pequeños implica una serie de pasos, cada uno de los cuales involucra ciertos registros del procesador. Nos referiremos a estos pasos como *microoperaciones*. El prefijo *micro* alude al hecho de que cada paso es muy sencillo y hace muy poco. La Figura 16.1 representa la relación entre los distintos conceptos de los que hemos hablado. De forma resumida, la ejecución de un programa consiste en la ejecución secuencial de instrucciones. Cada instrucción se ejecuta durante un ciclo de instrucción compuesto por subciclos más cortos (por ejemplo, subciclo de captación, indirecto, de ejecución, de interrupción, etc.). La ejecución de cada subciclo incluye una o más operaciones más breves, es decir, una o más microoperaciones.



**Figura 16.1.** Elementos que constituyen la ejecución de un programa.

Las microoperaciones son las operaciones funcionales, o atómicas, de un procesador. En esta sección examinaremos las microoperaciones para llegar a comprender cómo los eventos de cualquier instrucción se pueden describir como una secuencia de tales microoperaciones. Usaremos un ejemplo sencillo. En el resto de este capítulo, mostraremos cómo el concepto de microoperaciones sirve como guía para el diseño de la unidad de control.

## EL CICLO DE CAPTACIÓN

Comenzamos examinando el ciclo de captación, que tiene lugar al principio de cada ciclo de instrucción y hace que una instrucción sea captada de la memoria. Para el fin de este estudio, suponemos la organización representada en la Figura 12.6. Hay cuatro registros implicados:

- **Registro de dirección de memoria (*Memory Address Register, MAR*):** está conectado a las líneas de dirección del bus del sistema. Especifica la dirección de memoria de una operación de lectura o de escritura.
- **Registro intermedio de memoria (*Memory Buffer Register, MBR*):** está conectado a las líneas de datos del bus del sistema. Contiene el valor a almacenar en memoria o el último valor leído de memoria.
- **Contador de programa (*Program Counter, PC*):** contiene la dirección de la siguiente instrucción a captar.
- **Registro de instrucción (*Instruction Register, IR*):** contiene la última instrucción captada.

Consideremos la secuencia de eventos del ciclo de captación desde el punto de vista de su efecto sobre los registros del procesador. En la Figura 16.2 se muestra un ejemplo. Al comienzo del ciclo de captación, la dirección de la siguiente instrucción a ejecutar está en el contador de programa (PC); en este caso, la dirección es 1100100. El primer paso es llevar esa dirección al registro de dirección de memoria (MAR).

|     |                               |
|-----|-------------------------------|
| MAR |                               |
| MBR |                               |
| PC  | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| IR  |                               |
| AC  |                               |

(a) Comienzo

|     |                               |
|-----|-------------------------------|
| MAR | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| MBR | 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 |
| PC  | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| IR  |                               |
| AC  |                               |

(c) Segundo paso

|     |                               |
|-----|-------------------------------|
| MAR | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| MBR |                               |
| PC  | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| IR  |                               |
| AC  |                               |

(b) Primer paso

|     |                               |
|-----|-------------------------------|
| MAR | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| MBR | 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 |
| PC  | 0 0 0 0 0 0 0 0 1 1 0 0 1 0 0 |
| IR  | 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 |
| AC  |                               |

(d) Tercer paso

**Figura 16.2.** Secuencia de eventos del ciclo de captación.

memoria (MAR), ya que este es el único registro conectado a las líneas de dirección del bus del sistema. El segundo paso es traer la instrucción. La dirección deseada (en MAR) se coloca en el bus de direcciones, la unidad de control emite una orden READ por el bus de control, y el resultado aparece en el bus de datos y se copia en el registro intermedio de memoria (MBR). Es necesario además incrementar PC en  $I$  (longitud de la instrucción) para que esté preparado para la siguiente instrucción. Como estas dos acciones (leer una palabra de memoria, sumar  $I$  a PC) no se interfieren entre sí, podemos hacerlas simultáneamente para ahorrar tiempo. El tercer paso es transferir el contenido de MBR al registro de instrucción (IR). Esto libera MBR para su uso durante un posible ciclo indirecto.

De este modo, el sencillo ciclo de captación consta realmente de tres pasos y cuatro microoperaciones. Cada microoperación implica la transferencia de datos hacia dentro o hacia fuera de un registro. Con tal de que estas transferencias no se interfieran entre sí, varias de ellas pueden tener lugar durante un paso, ahorrando tiempo. Simbólicamente, podemos escribir esta secuencia de eventos como sigue:

```

 $t_1: \text{MAR} \leftarrow (\text{PC})$ 
 $t_2: \text{MBR} \leftarrow \text{Memoria}$ 
 $\quad \text{PC} \leftarrow (\text{PC}) + I$ 
 $t_3: \text{IR} \leftarrow (\text{MBR})$ 

```

donde  $I$  es la longitud de la instrucción. Tenemos que hacer varios comentarios sobre esta secuencia. Suponemos que se dispone de un reloj a efectos de temporización, y que este emite pulsos de reloj espaciados regularmente. Cada pulso de reloj define una unidad de tiempo. Así, todas las unidades de tiempo tienen la misma duración. Cada microoperación puede llevarse a cabo dentro de una única unidad de tiempo. La notación  $(t_1, t_2, t_3)$  representa las sucesivas unidades de tiempo. En palabras, tenemos:

- **Primera unidad de tiempo:** transferir el contenido de PC a MAR.
- **Segunda unidad de tiempo:** transferir el contenido de la posición de memoria especificada por MAR a MBR. Incrementar en  $I$  el contenido de PC.
- **Tercera unidad de tiempo:** transferir el contenido de MBR a IR.

Observe que las microoperaciones segunda y tercera tienen lugar durante la segunda unidad de tiempo. La tercera microoperación podría haberse agrupado con la cuarta sin afectar al funcionamiento de la captación:

```

 $t_1: \text{MAR} \leftarrow (\text{PC})$ 
 $t_2: \text{MBR} \leftarrow \text{Memoria}$ 
 $t_3: \text{PC} \leftarrow (\text{PC}) + I$ 
 $\quad \text{IR} \leftarrow (\text{MBR})$ 

```

Los agrupamientos de microoperaciones deben cumplir dos sencillas reglas:

1. Debe seguirse la secuencia correcta de eventos. Así,  $(\text{MAR} \leftarrow (\text{PC}))$  debe preceder a  $(\text{MBR} \leftarrow \text{Memoria})$ , ya que la operación de lectura de memoria hace uso de la dirección almacenada en MAR.

2. Deben evitarse los conflictos. No se debe intentar leer y escribir en el mismo registro en una unidad de tiempo, ya que los resultados serían imprevisibles. Por ejemplo, las microoperaciones ( $MBR \leftarrow$  Memoria) e ( $IR \leftarrow MBR$ ) no deberían tener lugar en la misma unidad de tiempo.

Un punto final digno de atención es que una de las microoperaciones incluye una suma. Para evitar la duplicación de circuitería, la suma podría realizarse en la ALU. El uso de la ALU puede implicar microoperaciones adicionales, dependiendo de la funcionalidad de la ALU y de la organización del procesador. Aplazamos la discusión de este punto hasta más adelante en este capítulo.

Es útil comparar los eventos descritos en esta y en las siguientes subsecciones con la Figura 3.5. Mientras que las microoperaciones se ignoraban en aquella figura, la presente discusión muestra las microoperaciones necesarias para llevar a cabo los subciclos del ciclo de instrucción.

## EL CICLO INDIRECTO

Una vez que se capta una instrucción, el siguiente paso es captar los operandos fuente. Siguiendo con nuestro ejemplo sencillo, supongamos un formato de instrucción de una dirección, que permite direccionamiento directo e indirecto. Si la instrucción especifica una dirección indirecta, un ciclo indirecto ha de preceder al ciclo de ejecución. El flujo de datos difiere un poco del que se indicó en la Figura 12.27, e incluye las siguientes microoperaciones:

```

 $t_1: MAR \leftarrow (IR \text{ (dirección)})$ 
 $t_2: MBR \leftarrow \text{Memoria}$ 
 $t_3: IR \text{ (dirección)} \leftarrow (MBR \text{ (dirección)})$ 

```

El campo de dirección en la instrucción se transfiere a MAR. Este se usa después para captar la dirección del operando. Por último, el campo de dirección de IR se actualiza con el contenido de MBR, de modo que contenga una dirección directa en lugar de una indirecta.

IR tiene ahora el mismo estado que si no se hubiera usado direccionamiento indirecto, y está listo para el ciclo de ejecución. De momento saltamos ese ciclo, para considerar el ciclo de interrupción.

## EL CICLO DE INTERRUPCIÓN

Cuando termina el ciclo de ejecución, se realiza una comprobación para determinar si ha ocurrido alguna interrupción habilitada. Si es así, tiene lugar un ciclo de interrupción. La naturaleza de este ciclo varía mucho de una máquina a otra. Aquí presentamos una secuencia muy simple de eventos, ilustrados en la Figura 12.8. Tenemos:

```

 $t_1: MBR \leftarrow (PC)$ 
 $t_2: MAR \leftarrow \text{Dirección de salvaguardia}$ 
 $PC \leftarrow \text{Dirección de rutina}$ 
 $t_3: \text{Memoria} \leftarrow (MBR)$ 

```

En el primer paso, el contenido de PC se transfiere a MBR, de modo que pueda guardarse para el retorno de la interrupción. Entonces MAR se carga con la dirección en la cual va a guardarse el

contenido de PC, y PC se carga con la dirección de comienzo de la rutina de procesamiento de la interrupción. Cada una de estas dos acciones puede ser una única microoperación. Sin embargo, ya que la mayoría de los procesadores tienen múltiples tipos y/o niveles de interrupciones, podrían hacer falta una o más microoperaciones adicionales para obtener la dirección de salvaguardia y la dirección de la rutina antes de que puedan transferirse a MAR y a PC, respectivamente. En todo caso, una vez hecho esto, el paso final es almacenar MBR, que contiene el antiguo valor de PC, en la memoria. El procesador queda entonces preparado para iniciar el siguiente ciclo de instrucción.

## EL CICLO DE EJECUCIÓN

Los ciclos de captación, indirecto y de interrupción son sencillos y predecibles. Cada uno implica una secuencia pequeña y fija de microoperaciones y, en todos los casos, se repiten las mismas microoperaciones para cada ejecución de una instrucción.

Esto no ocurre así en el ciclo de ejecución. En una máquina con  $N$  códigos de operación diferentes, pueden ocurrir  $N$  secuencias diferentes de microoperaciones. Consideremos varios ejemplos hipotéticos.

En primer lugar, consideremos una instrucción de suma:

ADD R1, X

que suma el contenido de la posición X al registro R1. Puede suceder la siguiente secuencia de microoperaciones:

```
t1: MAR ← (IR (dirección))
t2: MBR ← Memoria
t3: R1 ← (R1) + (MBR)
```

En un principio IR contiene la instrucción ADD. En el primer paso, la parte de dirección de IR se carga en MAR. Después se lee la posición de memoria referenciada. Por último, la ALU suma los contenidos de R1 y MBR. De nuevo, este es un ejemplo simplificado. Pueden necesitarse microoperaciones adicionales para extraer la referencia a registro desde IR y tal vez para poner las entradas o salidas de la ALU en algunos registros intermedios.

Consideremos dos ejemplos más complejos. Una instrucción frecuente es «incrementar y saltar si cero» (*Increment and Skip if Zero*):

ISZ X

El contenido de la posición X se incrementa en 1. Si el resultado es 0, la siguiente instrucción se salta. Una posible secuencia de microoperaciones es

```
t1: MAR ← (IR (dirección))
t2: MBR ← Memoria
t3: MBR ← (MBR) + 1
t4: Memoria ← (MBR)
If ((MBR) = 0) then (PC ← (PC) + I)
```

La nueva característica introducida aquí es la actuación condicional. PC se incrementa si  $(MBR) = 0$ . Esta comprobación y actuación puede implementarse como una microoperación. Observe que esta microoperación puede ejecutarse durante la misma unidad de tiempo en la cual el valor actualizado de MBR se almacena en memoria.

Por último, examinemos una instrucción de llamada a subrutina. Como ejemplo, consideremos la instrucción «saltar y guardar la dirección» (*Branch and Save Address*):

BSA X

La dirección de la instrucción que viene a continuación de la instrucción BSA se guarda en la posición X, y la ejecución continúa en la posición  $X + I$ . La dirección guardada se usará más adelante en el retorno. Esta es una técnica sencilla para proporcionar llamadas a subrutinas. Son suficientes las siguientes microoperaciones:

```

 $t_1: \text{MAR} \leftarrow (\text{IR} \text{ (dirección)})$ 
 $\text{MBR} \leftarrow (\text{PC})$ 
 $t_2: \text{PC} \leftarrow (\text{IR} \text{ (dirección)})$ 
 $\text{Memoria} \leftarrow (\text{MBR})$ 
 $t_3: \text{PC} \leftarrow (\text{PC}) + I$ 

```

La dirección que hay en PC al comienzo de la instrucción es la dirección de la siguiente instrucción secuencial. Ésta se guarda en la dirección señalada por IR. Esta última dirección también se incrementa para obtener la dirección de la instrucción correspondiente al siguiente ciclo de instrucción.

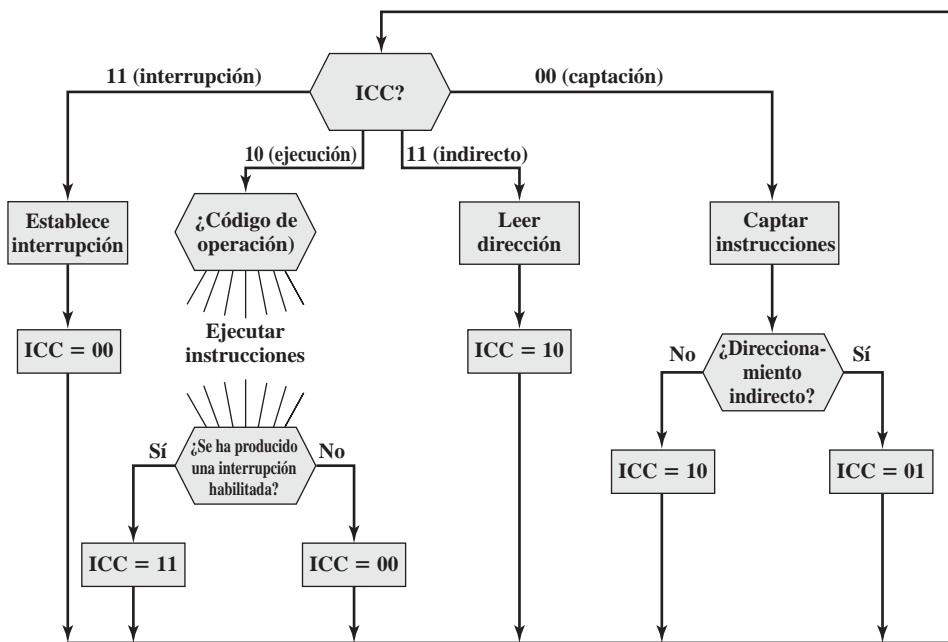
## EL CICLO DE INSTRUCCIÓN

Hemos visto que cada fase del ciclo de instrucción puede descomponerse en una secuencia de microoperaciones elementales. En nuestro ejemplo, hay una secuencia para cada uno de los ciclos de captación, indirecto y de interrupción, y para el ciclo de ejecución existe una secuencia de microoperaciones para cada código de operación.

Para completar la descripción, tenemos que unir las secuencias de microoperaciones, como se ha hecho en la Figura 16.3. Suponemos que hay un nuevo registro de dos bits llamado *código de ciclo de instrucción* (*instruction cycle code*, ICC). El ICC designa el estado del procesador en términos de en qué parte del ciclo se encuentra este:

- 00: Captación
- 01: Indirecto
- 10: Ejecución
- 11: Interrupción

Al final de cada uno de los cuatro ciclos, el ICC se actualiza convenientemente. El ciclo indirecto siempre viene seguido del ciclo de ejecución. El ciclo de interrupción siempre es seguido por el



**Figura 16.3.** Diagrama de flujo del ciclo de instrucción.

ciclo de captación (ver Figura 12.4). En el caso de los ciclos de ejecución y de captación, el siguiente ciclo depende del estado del sistema.

De este modo, el diagrama de flujo de la Figura 16.3 define la secuencia completa de microoperaciones, que dependen solo de la secuencia de instrucciones y del patrón de interrupciones. Naturalmente, este es un ejemplo simplificado. El diagrama de flujo de un procesador real sería más complejo. De todas formas, hemos llegado al punto de nuestra discusión en el que el funcionamiento del procesador se define como la ejecución de una secuencia de microoperaciones. Podemos considerar ahora cómo consigue la unidad de control que ocurra esta secuencia.

## 16.2. CONTROL DEL PROCESADOR

### REQUISITOS FUNCIONALES

Como consecuencia de nuestro análisis de la sección precedente, hemos descompuesto el comportamiento o funcionamiento del procesador en operaciones elementales, llamadas microoperaciones. Reduciendo el funcionamiento del procesador a su nivel más básico, podemos definir exactamente qué es lo que la unidad de control tiene que hacer que ocurra. Así, podemos definir los *requisitos funcionales* de la unidad de control como aquellas funciones que debe llevar a cabo. Una definición de estos requisitos funcionales es la base del diseño e implementación de la unidad de control.

Con la información a mano, el siguiente proceso de tres pasos lleva a la caracterización de la unidad de control:

1. Definir los elementos básicos del procesador.
2. Describir las microoperaciones que ejecuta el procesador.
3. Determinar las funciones que debe realizar la unidad de control para hacer que se ejecuten las microoperaciones.

Ya hemos presentado los pasos 1 y 2. Resumamos los resultados. En primer lugar, los elementos funcionales básicos del procesador son los siguientes:

- ALU
- Registros
- Caminos de datos internos
- Caminos de datos externos
- Unidad de control

Algunas consideraciones deberían convencerle de que esta lista está completa. La ALU es la esencia funcional del computador. Los registros se usan para almacenar datos internos del procesador. Algunos registros contienen información de estado necesaria para gestionar el secuenciamiento de las instrucciones (por ejemplo, la palabra de estado del programa). Otros contienen datos que van hacia, o vienen desde, la ALU, la memoria y los módulos de E/S. Los caminos de datos internos se usan para transferir datos entre los registros y entre estos y la ALU. Los caminos de datos externos unen los registros a la memoria y a los módulos de E/S, a menudo por medio de un bus del sistema. La unidad de control hace que se produzcan operaciones dentro del procesador.

La ejecución de un programa consta de operaciones que involucran estos elementos del procesador. Como hemos visto, estas operaciones consisten en una secuencia de microoperaciones. Tras la revisión de la Sección 16.1, el lector debería observar que todas las microoperaciones se pueden clasificar en una de las siguientes categorías:

- Transferir datos de un registro a otro.
- Transferir datos de un registro a una interfaz externa (por ejemplo, al bus del sistema).
- Transferir datos de una interfaz externa a un registro.
- Realizar una operación aritmética o lógica, usando registros para entrada y salida.

Todas las microoperaciones necesarias para realizar un ciclo de instrucción, incluyendo todas las microoperaciones necesarias para ejecutar cada instrucción del repertorio, están incluidas en una de estas categorías.

Podemos ser ahora algo más explícitos acerca de la forma en que funciona la unidad de control. La unidad de control realiza dos tareas básicas:

- **Secuenciamiento:** la unidad de control hace que el procesador avance a través de una serie de microoperaciones en la secuencia oportuna, basada en el programa que se está ejecutando.
- **Ejecución:** la unidad de control hace que se ejecute cada microoperación.

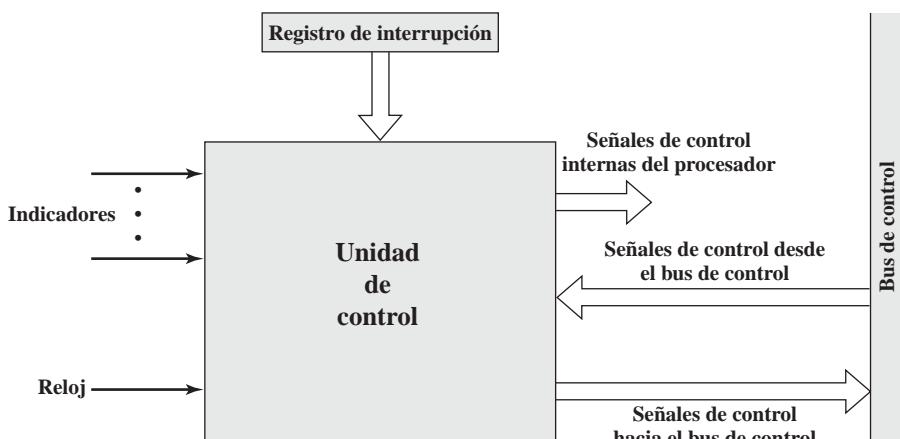
Lo que precede es una descripción funcional de lo que hace la unidad de control. La clave de cómo funciona la unidad de control es la utilización de señales de control.

## SEÑALES DE CONTROL

Hemos definido los elementos que componen el procesador (ALU, registros, caminos de datos) y las microoperaciones que se llevan a cabo. Para que la unidad de control realice su función, debe tener entradas que le permitan determinar el estado del sistema y salidas que le permitan controlar el comportamiento del mismo. Estas son las especificaciones externas de la unidad de control. Internamente, la unidad de control ha de tener la lógica necesaria para realizar sus funciones de secuenciamiento y ejecución. Aplazamos el estudio del funcionamiento interno de la unidad de control hasta la Sección 16.3 y el Capítulo 17. El resto de esta sección se ocupa de la interacción entre la unidad de control y otros elementos del procesador.

La Figura 16.4 es un modelo general de la unidad de control, que muestra todas sus entradas y salidas. Las entradas son las siguientes:

- **Reloj:** es el encargado de «mantener la hora exacta». La unidad de control hace que se ejecute una microoperación (o un conjunto de microoperaciones simultáneas) en cada pulso de reloj. Este a menudo es referenciado como tiempo de ciclo del procesador, o período de reloj.
- **Registro de instrucción:** el código de operación de la instrucción en curso se usa para determinar qué microoperaciones hay que realizar durante el ciclo de ejecución.
- **Indicadores:** los necesita la unidad de control para determinar el estado del procesador y el resultado de anteriores operaciones de la ALU. Por ejemplo, para la instrucción incrementar y saltar si cero (ISZ), la unidad de control incrementará PC si el indicador de cero está a uno.
- **Señales de control del bus de control:** la parte de control del bus del sistema suministra señales a la unidad de control, tales como señales de interrupción y de reconocimiento.



**Figura 16.4.** Diagrama de bloques de la unidad de control.

Las salidas son las siguientes:

- **Señales de control internas al procesador:** son de dos tipos: las que hacen que los datos se transfieran de un registro a otro, y las que activan funciones específicas de la ALU.
- **Señales de control hacia el bus de control:** también las hay de dos tipos: señales de control de la memoria, y señales de control de los módulos de E/S.

El nuevo elemento introducido en esta figura es el de señal de control. Se usan tres tipos de señales de control: las que activan una función de la ALU, las que activan un camino de datos, y las que son señales del bus del sistema externo u otra interfaz externa. Todas estas señales acaban aplicándose directamente como entradas binarias a puertas lógicas individuales.

Consideremos nuevamente el ciclo de captación para entender cómo mantiene el control la unidad de control. La unidad de control se mantiene al tanto de dónde está dentro del ciclo de instrucción. En un punto determinado, sabe que inmediatamente después se va a realizar un ciclo de captación. El primer paso es transferir el contenido de PC a MAR. La unidad de control hace esto activando la señal de control que abre las puertas entre los bits de PC y los bits de MAR. El siguiente paso es leer una palabra desde memoria a MBR e incrementar PC. La unidad de control hace esto enviando las siguientes señales de control simultáneamente:

- Una señal de control que abre las puertas que permiten que el contenido de MAR aparezca en el bus de direcciones.
- Una señal de control de lectura de memoria, en el bus de control.
- Una señal de control que abre las puertas que permiten almacenar el contenido del bus de datos en MBR.
- Señales de control de la lógica que suma  $I$  al contenido de PC y almacena el resultado de nuevo en PC.

Después de esto, la unidad de control envía una señal de control que abre las puertas adecuadas entre MBR e IR.

Esto completa el ciclo de captación exceptuando un detalle: la unidad de control debe decidir si ejecuta a continuación un ciclo indirecto o un ciclo de ejecución. Para decidir esto, examina IR viendo si se hace una referencia indirecta a memoria.

Los ciclos indirecto y de interrupción funcionan de un modo parecido. En el caso del ciclo de ejecución, la unidad de control comienza examinando el código de operación y, en función de él, decide qué secuencia de microoperaciones deben realizarse para ejecutar el ciclo.

## UN EJEMPLO DE SEÑALES DE CONTROL

Para ilustrar el funcionamiento de la unidad de control, examinemos un ejemplo sencillo. La Figura 16.5 ilustra el ejemplo. Se trata de un procesador sencillo con un único acumulador. Se indican los caminos de datos entre los distintos elementos. Los caminos de control de las señales que proceden de la unidad de control no se muestran, pero las terminaciones de las señales de control están

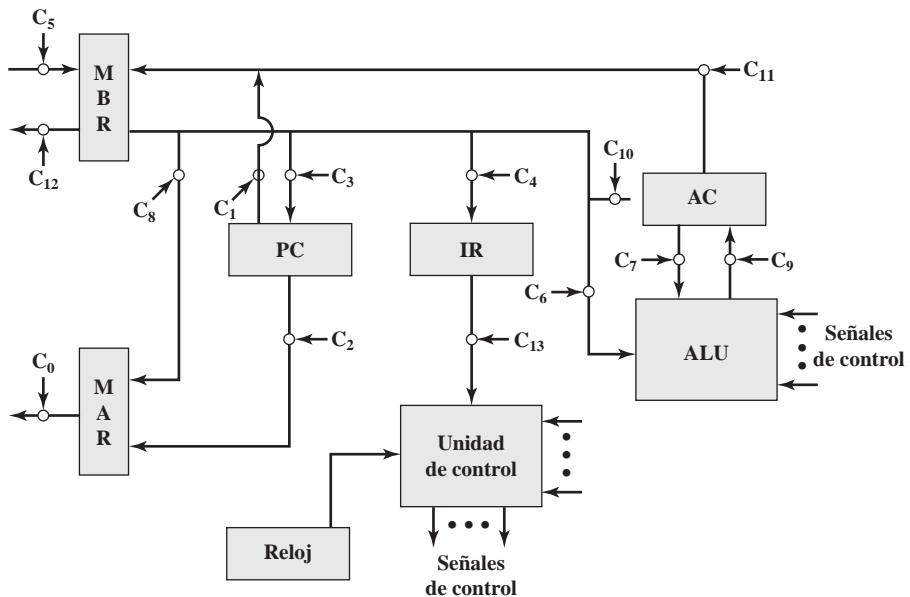


Figura 16.5. Caminos de datos y señales de control.

designadas como  $C_i$  y se indican mediante un círculo. La unidad de control recibe entradas del reloj, del registro de instrucción, y de los indicadores. En cada ciclo de reloj, la unidad de control lee todas sus entradas y emite un conjunto de señales de control. Las señales de control van hacia tres destinos distintos:

- **Caminos de datos:** la unidad de control dirige el flujo interno de datos. Por ejemplo, en la captación de instrucción, el contenido del registro intermedio de memoria se transfiere al registro de instrucción. Por cada camino a controlar hay una puerta (indicada mediante un círculo en la figura). Una señal de control de la unidad de control abre temporalmente la puerta para dejar pasar los datos.
- **ALU:** la unidad de control gobierna el funcionamiento de la ALU mediante un conjunto de señales de control. Estas señales activan diversos dispositivos y puertas dentro de la ALU.
- **Bus del sistema:** la unidad de control envía señales de control a las líneas de control del bus del sistema (por ejemplo, lectura de la memoria).

La unidad de control debe conocer en todo momento dónde está dentro del ciclo de instrucción. Usando ese conocimiento, y leyendo todas sus entradas, la unidad de control emite una serie de señales de control que hacen que se efectúen las microoperaciones. La unidad de control usa los pulsos de reloj para temporizar la secuencia de eventos, dejando tiempo entre eventos para que los niveles de las señales se estabilicen. La Tabla 16.1 indica las señales de control necesarias para realizar algunas de las secuencias de microoperaciones descritas con anterioridad. Por simplicidad, no se han mostrado los caminos de datos y de control necesarios para incrementar PC y para cargar direcciones fijas en PC y MAR.

El carácter mínimo de la unidad de control merece ser considerado. La unidad de control es el motor que mueve todo el computador. Lo hace basándose solo en el conocimiento de las instrucciones

**Tabla 16.1.** Microoperaciones y señales de control.

| Microoperaciones | Temporización  | Señales de control activas |
|------------------|--|----------------------------|
| Captación:       | $t_1: MAR \leftarrow (PC)$   | $C_2$                      |
|                  | $t_2: MBR \leftarrow \text{Memoria}$<br>$PC \leftarrow (PC) + l$                                     | $C_5, C_R$                 |
|                  | $t_3: IR \leftarrow (MBR)$   | $C_4$                      |
| Indirecto:       | $t_1: MAR \leftarrow (IR \text{ (dirección)})$   | $C_8$                      |
|                  | $t_2: MBR \leftarrow \text{Memoria}$   | $C_5, C_R$                 |
|                  | $t_3: IR \text{ (dirección)} \leftarrow (MBR \text{ (dirección)})$                                   | $C_4$                      |
| Interrupción:    | $t_1: MBR \leftarrow (PC)$   | $C_1$                      |
|                  | $t_2: MAR \leftarrow \text{Dirección de salvaguardia}$<br>$PC \leftarrow \text{Dirección de rutina}$ |                            |
|                  | $t_3: \text{Memoria} \leftarrow (MBR)$   | $C_{12}, C_W$              |

$C_R$  = Señal de control de lectura (*read*) hacia el bus del sistema.

$C_W$  = Señal de control de escritura (*write*) hacia el bus del sistema.

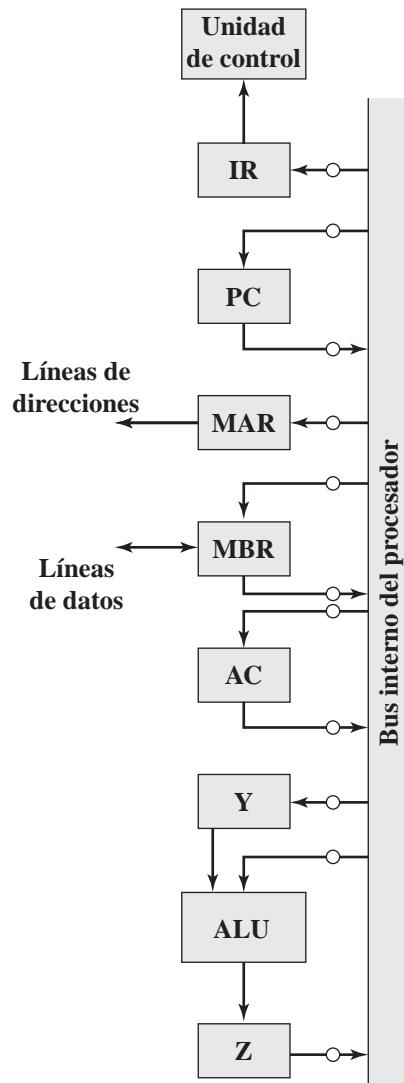
que tiene que ejecutar y de la naturaleza de los resultados de las operaciones aritméticas y lógicas (por ejemplo, resultado positivo, desbordamiento, etc.). Nunca llega a ver los datos que se procesan o los resultados reales producidos. Y controla todo con unas pocas señales de control que van a ciertos puntos dentro del procesador y unas pocas señales que van hacia el bus del sistema.

## ORGANIZACIÓN INTERNA DEL PROCESADOR

La Figura 16.5 indica el uso de diversos caminos de datos. La complejidad de este tipo de organización debería estar clara. Es más normal usar algún tipo de configuración de bus interno, como se sugirió en la Figura 12.2.

Usando un bus interno al procesador, la Figura 16.5 puede disponerse del modo que muestra la Figura 16.6. Un único bus interno conecta la ALU y todos los registros del procesador. Hay puertas y señales de control encargadas de realizar transferencias de datos entre el bus y cada registro. Otras señales de control dirigen las transferencias de datos hacia y desde el bus (externo) del sistema y el funcionamiento de la ALU.

Se han añadido dos nuevos registros, rotulados como Y y Z, a la organización. Son necesarios para el correcto funcionamiento de la ALU. Cuando se realiza una operación que incluye dos operandos, uno se puede obtener desde el bus interno, pero el otro ha de obtenerse de otra fuente. El AC podría usarse para este propósito, pero ello limita la flexibilidad del sistema y no funcionaría en un procesador con múltiples registros de uso general. El registro Y proporciona un almacenamiento temporal de la otra entrada. La ALU es un circuito combinacional (*ver* Apéndice A) sin almacenamiento interno. De este modo, cuando las señales de control activan una función de la ALU, la entrada de esta se transforma en una salida. Debido a ello, la salida de la ALU no se puede conectar directamente al bus, ya



**Figura 16.6.** Procesador con bus interno.

que realimentaría la entrada. El registro Z proporciona el almacenamiento temporal de salida. Con esta configuración, una operación de suma de un valor de la memoria al AC tendría los siguientes pasos:

- $t_1: \text{MAR} \leftarrow (\text{IR} \text{ (dirección)})$
- $t_2: \text{MBR} \leftarrow \text{Memoria}$
- $t_3: Y \leftarrow (\text{MBR})$
- $t_4: Z \leftarrow (\text{AC}) + (Y)$
- $t_5: R1 \leftarrow (R1) + (\text{MBR})$

Son posibles otras organizaciones, pero, en general, se usa algún tipo de bus interno o conjunto de buses internos. El uso de caminos de datos comunes simplifica el trazado de las interconexiones y

el control del procesador. Otra razón práctica para usar un bus interno es ahorrar espacio. El espacio ocupado por las conexiones entre registros tiene que minimizarse especialmente en los microprocesadores, que sólo pueden ocupar un trozo cuadrado de silicio de un cuarto de pulgada.

### EL INTEL 8085

Para ilustrar algunos de los conceptos introducidos hasta aquí en este capítulo, consideremos el Intel 8085. Su organización se muestra en la Figura 16.7. Hay varios componentes clave que pueden no explicarse por sí mismos:

- **Latch (cerrojo) incrementador/decrementador de direcciones:** lógica que puede sumar 1 o restar 1 al contenido del puntero de pila o al contador de programa. Ahorra tiempo ya que evita usar la ALU para este fin.
- **Control de interrupciones:** este módulo maneja múltiples niveles de señales de interrupción.

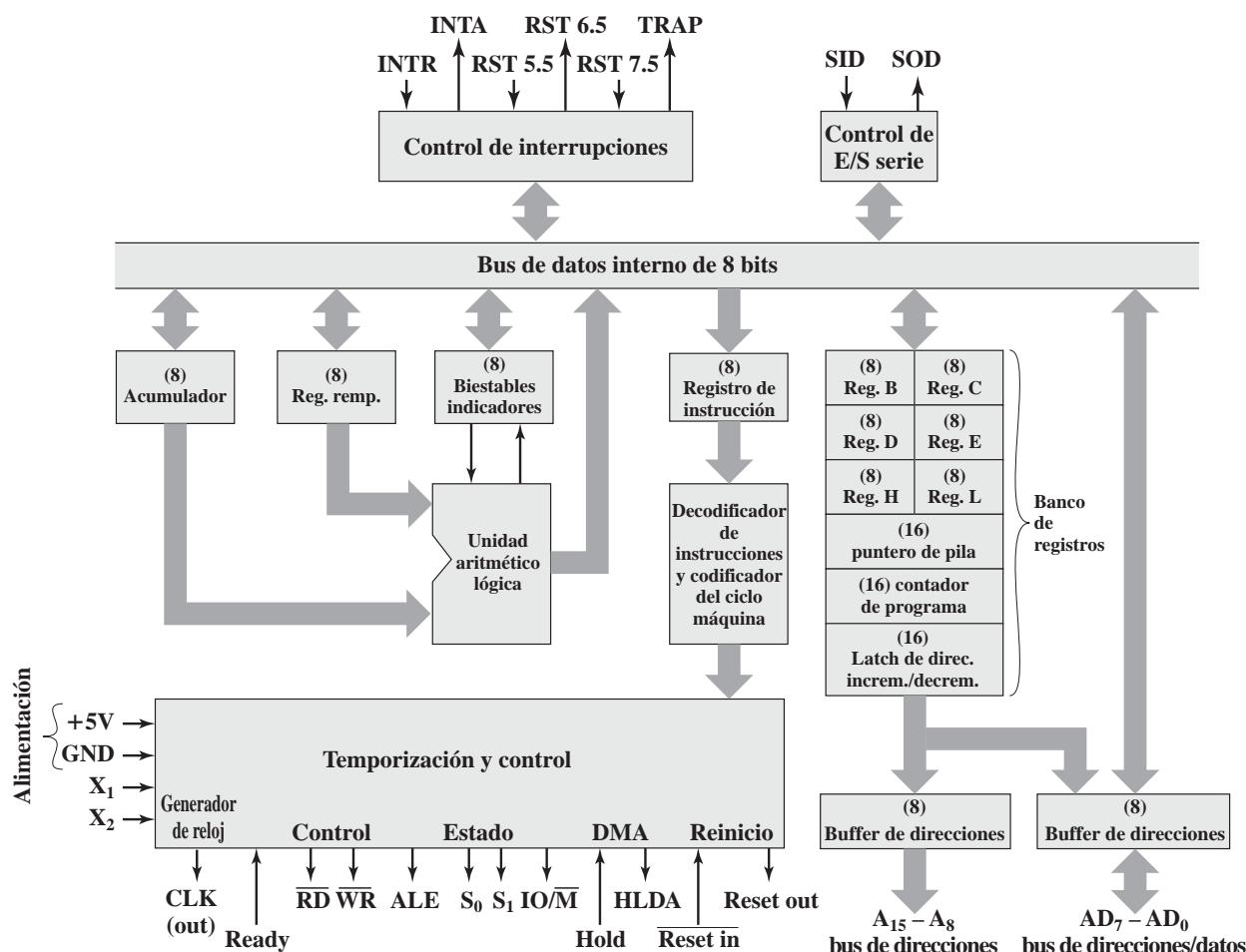


Figura 16.7. Diagrama de bloques del procesador Intel 8085.

- **Control de E/S serie:** este módulo se conecta a dispositivos capaces de transferir 1 bit cada vez.

La Tabla 16.2 describe las señales externas que entran y salen del 8085. Estas se unen al bus externo del sistema y son la interfaz entre el procesador 8085 y el resto del sistema (Figura 16.8).

**Tabla 16.2.** Señales externas del Intel 8085.

#### *Señales de datos y direcciones*

**Direcciones altas (A<sub>15</sub> – A<sub>8</sub>)**

Los ocho bits más significativos de una dirección de 16 bits.

**Direcciones/datos (AD7 – AD0)**

Los ocho bits menos significativos de una dirección de 16 bits o bien ocho bits de datos. Esta multiplexación ahorra pines.

**Dato de entrada serie (Serial Input Data, SID)**

Una entrada de un único bit para adaptarse a dispositivos que transmitan de forma serie (un bit cada vez).

**Dato de salida serie (Serial Output Data, SOD)**

Una salida de un único bit para adaptarse a dispositivos que reciben de forma serie.

#### *Señales de temporización y control*

**CLK (Out)**

El reloj del sistema. Cada ciclo representa un estado T. La señal CLK va hacia circuitos periféricos y sincroniza su temporización.

**X<sub>1</sub>, X<sub>2</sub>**

Estas señales provienen de un cristal externo u otro dispositivo para controlar el generador de reloj interno.

**Habilitación del latch de direcciones (Address Latch Enable, ALE)**

Tiene lugar durante el primer estado de un ciclo máquina y hace que los circuitos periféricos almacenen las líneas de dirección. Esto permite a un módulo (por ejemplo, memoria, E/S) reconocer que está siendo direccionado.

**Estado (S<sub>0</sub>, S<sub>1</sub>)**

Señales de control usadas para indicar si está teniendo lugar una operación de lectura o escritura.

**IO/M**

Usada para controlar operaciones de lectura o escritura de los módulos de E/S o de memoria.

**Control de lectura (RD)**

Indica que el módulo de memoria o de E/S seleccionado va a leerse y que el bus de datos está disponible para una transferencia de datos.

**Control de escritura (WR)**

Indica que el dato que hay en el bus de datos va a escribirse en la posición de memoria o E/S seleccionada.

#### *Señales originadas en memoria y E/S*

**Hold**

Pide al procesador que abandone el control y el uso del bus del sistema externo. El procesador completará la ejecución de la instrucción que hay en el momento presente en el IR y entrará en un estado de desconexión, durante el cual no pone señales en los buses de control, de direcciones y de datos. Durante el estado de desconexión, el bus puede usarse para operaciones de DMA.

**Reconocimiento de Hold (Hold Acknowledge, HLDA)**

Esta señal de salida de la unidad de control reconoce la señal HOLD e indica que el bus queda disponible.

**Ready**

Usado para sincronizar el procesador con una memoria más lenta o con dispositivos de E/S. Cuando un dispositivo seleccionado mantiene Ready a uno, el procesador puede continuar con una operación de entrada (DBIN) o salida (WR). En caso contrario, el procesador entra en un estado de espera hasta que el dispositivo esté listo.

(Continúa)

**Tabla 16.2.** Señales externas del Intel 8085 (*continuación*).

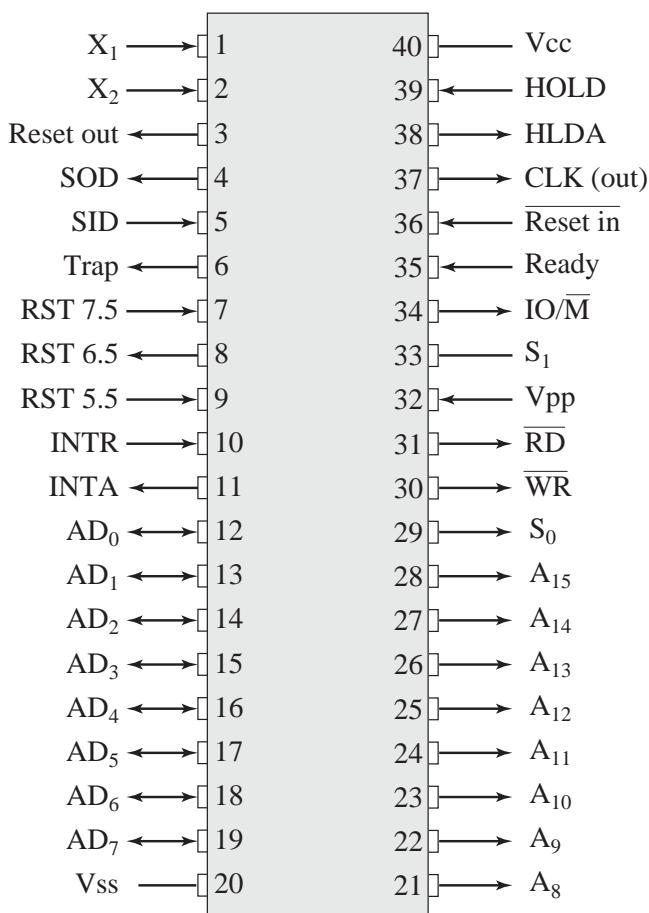
| <b>Señales relacionadas con interrupciones</b>           |   |
|--|---|
| <b>Trap.</b>   |   |
| <b>Interrupciones de reanudación (RST 7.5, 6.5, 5.5)</b> |   |
| <b>Petición de interrupción (INTR)</b>                   | Un dispositivo externo usa estas cinco líneas para interrumpir al procesador. Este no aceptará una petición si se encuentra en el estado de desconexión o si la interrupción está inhabilitada. Una interrupción se acepta solo al final de una instrucción. Estas interrupciones se indican en orden de prioridad descendente. |
| <b>Reconocimiento de interrupción</b>                    | Reconoce una interrupción.  |
|  | <b>Reinicio del procesador</b>  |
| <b>Reset In</b>  | Provoca la puesta a cero del contenido de PC. El procesador reanuda la ejecución en la posición cero.   |
| <b>Reset Out</b>   | Reconoce que el procesador ha sido reiniciado. Esta señal puede usarse para reiniciar el resto del sistema.   |
|  | <b>Alimentación y tierra</b>  |
| <b>V<sub>cc</sub></b>                                    | Alimentación a +5 voltios.  |
| <b>V<sub>ss</sub></b>                                    | Tierra.   |

La unidad de control se identifica como los dos componentes rotulados (1) decodificador de instrucciones y codificación del ciclo máquina y (2) temporización y control. Se aplaza la discusión del primer componente hasta la siguiente sección. La parte fundamental de la unidad de control es el módulo de temporización y control. Este módulo incluye un reloj y acepta como entradas la instrucción en curso y algunas señales de control externas. Su salida consiste en señales de control hacia los otros componentes del procesador más señales de control hacia el bus externo del sistema.

La temporización de las operaciones del procesador está sincronizada por el reloj y está controlada por la unidad de control por medio de señales de control. Cada ciclo de instrucción se divide en *ciclos máquina*, de uno a cinco; cada ciclo máquina se divide a su vez en *estados*, de tres a cinco. Cada estado dura un ciclo de reloj. Durante un estado, el procesador ejecuta una o un conjunto de microoperaciones simultáneas determinadas por las señales de control.

El número de ciclos máquina es fijo para cada instrucción pero varía de una instrucción a otra. Los ciclos máquina se definen como equivalentes a los accesos al bus. De este modo, el número de ciclos máquina de una instrucción depende del número de veces que el procesador debe comunicarse con los dispositivos externos. Por ejemplo, si una instrucción se compone de dos partes de ocho bits, se necesitan dos ciclos máquina para captar dicha instrucción. Si la instrucción implica una operación de un byte con memoria o E/S, será necesario un tercer ciclo máquina para su ejecución.

La Figura 16.9 ofrece un ejemplo de temporización del 8085 que muestra el valor de las señales de control externas. Naturalmente, al mismo tiempo, la unidad de control está generando señales de control internas para controlar transferencias de datos en el interior del procesador. El diagrama muestra el ciclo de instrucción de la instrucción OUT. Se necesitan tres ciclos máquina ( $M_1, M_2, M_3$ ).



**Figura 16.8.** Configuración de pines del Intel 8085.

Durante el primero se capta la instrucción OUT. El segundo ciclo máquina capta la segunda mitad de la instrucción, que contiene el número del dispositivo de E/S seleccionado para salida. Durante el tercer ciclo, el contenido del AC se escribe a través del bus de datos en el dispositivo seleccionado.

El comienzo de cada ciclo máquina viene determinado por el pulso de habilitación del *latch* de direcciones (*Address Latch Enable*, ALE) emitido por la unidad de control. Durante el estado de temporización T<sub>1</sub> del ciclo máquina M<sub>1</sub>, la unidad de control ajusta la señal IO/M para indicar una operación con memoria. Además, la unidad de control hace que el contenido de PC se sitúe en el bus de direcciones (A<sub>15</sub> a A<sub>8</sub>) y el bus de direcciones/datos (AD<sub>7</sub> a AD<sub>0</sub>). En el flanco de bajada del pulso ALE, los otros módulos conectados al bus almacenan la dirección.

Durante el estado de temporización T<sub>2</sub> el módulo de memoria direccionado pone el contenido de la posición de memoria en el bus de direcciones/datos. La unidad de control activa la señal de control de lectura (RD) para indicar una lectura, pero espera hasta T<sub>3</sub> para captar los datos del bus. Esto deja tiempo al módulo de memoria para poner los datos en el bus y para que se establezcan los niveles de las señales. El estado final, T<sub>4</sub>, es un estado de *bus desocupado* durante el cual el procesador decodifica la instrucción. Los restantes ciclos máquina actúan de forma parecida.

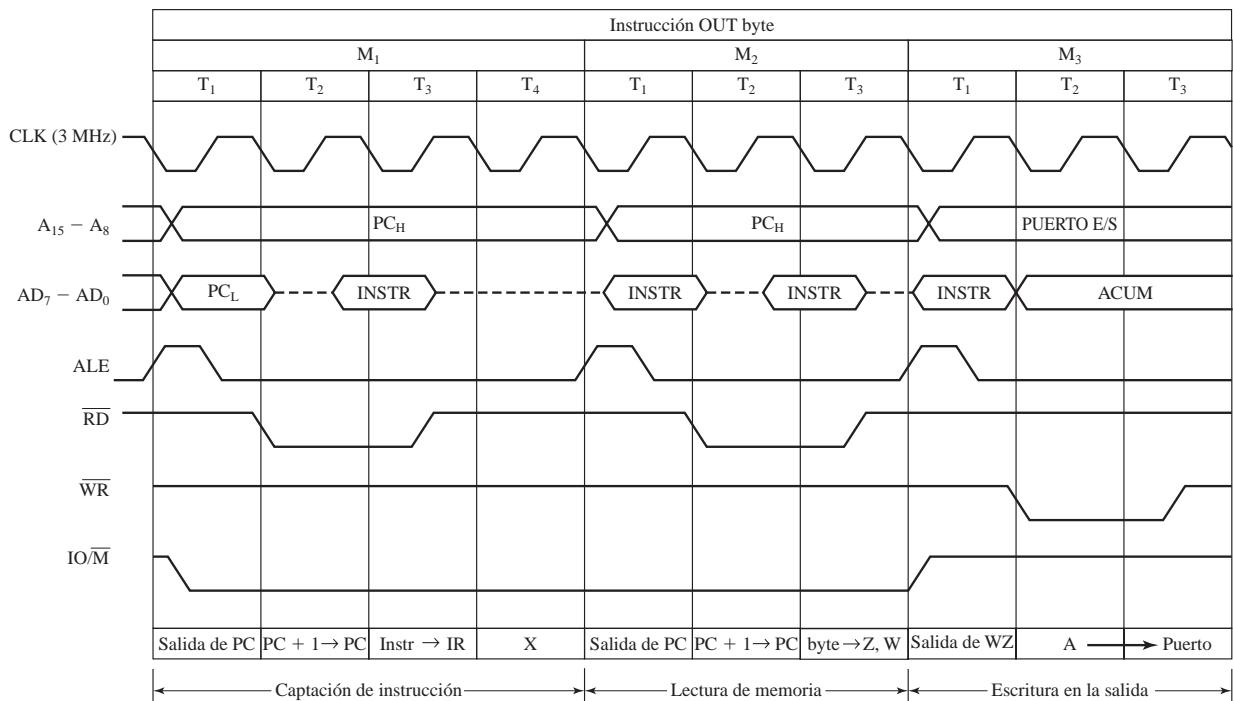


Figura 16.9. Diagrama de tiempos de la instrucción OUT del Intel 8085.

### 16.3. IMPLEMENTACIÓN CABLEADA

Hemos estudiado la unidad de control en lo referente a entradas, salidas y funciones. Ahora es el momento de volver al tema de la implementación de la unidad de control. Se ha usado una gran variedad de técnicas. La mayoría de ellas se pueden clasificar en dos categorías:

- Implementación cableada
- Implementación micropogramada

En una implementación cableada, la unidad de control es esencialmente un circuito combinacional. Sus señales lógicas de entrada se transforman en un conjunto de señales lógicas de salida, que son las señales de control. Este enfoque se examina en esta sección. La implementación microprogramada es el tema del que trata el Capítulo 17.

#### ENTRADAS DE LA UNIDAD DE CONTROL

La Figura 16.4 representa la unidad de control como la hemos estudiado hasta aquí. Las entradas principales son el registro de instrucción, el reloj, los indicadores y las señales de control del bus. En el caso de los indicadores y de las señales de control del bus, cada bit individual tiene normalmente

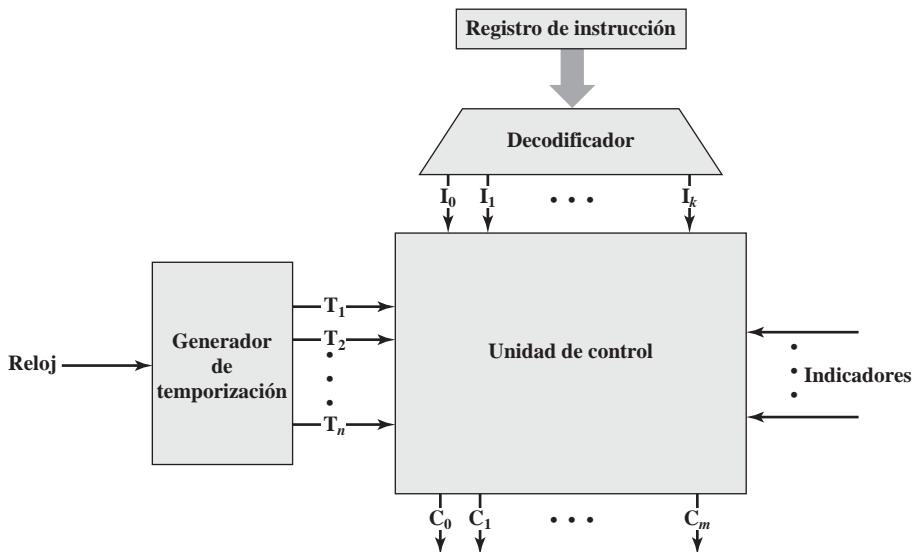
un significado determinado (por ejemplo, desbordamiento). Las otras dos entradas, sin embargo, no son útiles a la unidad de control tal como entran.

Consideremos en primer lugar el registro de instrucción. La unidad de control hace uso del código de operación y realiza acciones diferentes (emite una combinación diferente de señales de control) para cada instrucción. Para simplificar la lógica de la unidad de control, debería existir una entrada lógica única para cada código de operación. Esta función la puede realizar un *decodificador*, que toma una entrada codificada y produce una salida única. En general, un decodificador tendrá  $n$  entradas binarias y  $2^n$  salidas binarias. Cada uno de los  $2^n$  patrones de entrada distintos activará una única salida. La Tabla 16.3 es un ejemplo. El decodificador de una unidad de control normalmente tendrá que ser más complejo que el de la tabla, para representar códigos de operación de longitud variable. En el Apéndice A se presenta un ejemplo de la lógica digital usada para realizar un decodificador.

El reloj de la unidad de control emite una secuencia repetitiva de pulsos. Esto es útil para delimitar la duración de las microoperaciones. Esencialmente, el periodo de los pulsos de reloj ha de ser suficientemente largo para permitir la propagación de las señales a lo largo de los caminos de datos y a través de la circuitería del procesador. Sin embargo, como hemos visto, la unidad de control emite señales de control diferentes en unidades de tiempo diferentes dentro de un único ciclo de instrucción. Por tanto, podríamos tener un contador como entrada a la unidad de control, con una señal de control diferente para  $T_1$ ,  $T_2$ , etc. Al final de un ciclo de instrucción, la unidad de control deberá realimentar el contador para reiniciarlo a  $T_1$ .

Con estos dos refinamientos, la unidad de control se puede representar como en la Figura 16.10.

**Tabla 16.3.** Un decodificador con cuatro entradas y dieciséis salidas.



**Figura 16.10.** Unidad de control con entradas decodificadas.

## LÓGICA DE LA UNIDAD DE CONTROL

Para definir la implementación cableada de una unidad de control, todo lo que queda es estudiar su lógica interna, que produce señales de control de salida a partir de las señales de entrada.

Básicamente, lo que tenemos que hacer es, para cada señal de control, obtener su expresión booleana como una función de las entradas. Esto se explica mejor con un ejemplo. Consideraremos otra vez nuestro ejemplo sencillo ilustrado en la Figura 16.5. Vimos en la Tabla 16.1 las secuencias de microoperaciones y de señales de control necesarias para controlar tres de las cuatro fases del ciclo de instrucción.

Consideremos una única señal de control,  $C_5$ . Esta señal hace que se lean datos del bus de datos externo en MBR. Podemos ver que se usa dos veces en la Tabla 16.1. Definamos dos nuevas señales de control, P y Q, que tengan la siguiente interpretación:

|           |                       |
|-----------|-----------------------|
| $PQ = 00$ | Ciclo de captación    |
| $PQ = 01$ | Ciclo indirecto       |
| $PQ = 10$ | Ciclo de ejecución    |
| $PQ = 11$ | Ciclo de interrupción |

La siguiente expresión booleana define a  $C_5$ :

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2$$

Es decir, la señal de control  $C_5$  se pondrá a uno durante la segunda unidad de tiempo de los ciclos de captación e indirecto.

Esta expresión no está completa.  $C_5$  se necesita también durante el ciclo de ejecución. Para nuestro sencillo ejemplo, supongamos que hay solo tres instrucciones que leen de la memoria: LDA, ADD, y AND. Ahora podemos definir  $C_5$  como

$$C_5 = \bar{P} \cdot \bar{Q} \cdot T_2 + \bar{P} \cdot Q \cdot T_2 + P \cdot \bar{Q} \cdot (LDA + ADD + AND) \cdot T_2$$

Este mismo proceso podría repetirse para cada señal de control generada por el procesador. El resultado sería un conjunto de ecuaciones booleanas que definiría el comportamiento de la unidad de control y por tanto del procesador.

Juntando todo, la unidad de control debe controlar el estado del ciclo de instrucción. Como se mencionó, al final de cada subciclo (captación, indirecto, ejecución, interrupción), la unidad de control emite una señal que hace que el generador de temporización se reinicie y emita  $T_1$ . Además, la unidad de control ha de establecer los valores adecuados de  $P$  y  $Q$  para definir el siguiente subciclo a ejecutar.

El lector debe comprender que en un complejo procesador moderno, el número de ecuaciones booleanas necesarias para definir la unidad de control es muy grande. La tarea de implementar un circuito combinacional que satisfaga todas esas ecuaciones llega a ser sumamente difícil. El resultado es que, por regla general, se usa una aproximación mucho más sencilla, conocida como *microprogramación*. De este tema se ocupa el próximo capítulo.

## 16.4. LECTURAS RECOMENDADAS

Varios libros de texto tratan los principios básicos del funcionamiento de la unidad de control, entre los que se incluyen [FARH04] y [MANO04].

**FARH04** FARHAT, H.: *Digital Design and Computer Organization*. Boca Raton, FL. CRC Press, 2004.

**MANO04** MANO, M.: *Logic and Computer Design Fundamentals*. Upper Saddle River, NJ. Prentice Hall, 2004.

## 16.5. PALABRAS CLAVE, PREGUNTAS DE REPASO Y PROBLEMAS

### PALABRAS CLAVE

bus de control  
camino de control

implementación cableada  
microoperaciones

señal de control  
unidad de control

### PREGUNTAS DE REPASO

- 16.1. Explique la diferencia entre la secuencia escrita y la secuencia de tiempo de una instrucción.
- 16.2. ¿Cuál es la relación entre instrucciones y microoperaciones?

## CAPÍTULO 17

---

# Control micropogramado

### **17.1. Conceptos básicos**

Microinstrucciones  
Unidad de control microprogramada  
Control de Wilkes  
Ventajas e inconvenientes

### **17.2. Secuenciamiento de microinstrucciones**

Consideraciones respecto al diseño  
Técnicas de secuenciamiento  
Generación de direcciones  
Secuenciamiento de microinstrucciones en el LSI-11

### **17.3. Ejecución de microinstrucciones**

Una taxonomía de las microinstrucciones  
Codificación de las microinstrucciones  
Ejecución de microinstrucciones en el LSI-11  
Ejecución de microinstrucciones en el IBM 3033

### **17.4. TI 8800**

Formato de microinstrucción  
Microsecuenciador  
ALU con registros

### **17.5. Lecturas recomendadas**

### **17.5. Palabras clave, preguntas de repaso y problemas**

Palabras clave  
Preguntas de repaso  
Problemas

## PUNTOS CLAVE

- Una alternativa a la unidad de control cableada es la unidad de control microprogramada, en la cual la lógica de la unidad de control se especifica mediante un microprograma. Un microprograma consiste en una secuencia de instrucciones en un lenguaje de microprogramación. Se trata de instrucciones muy elementales que especifican microoperaciones.
- Una unidad de control microprogramada es un circuito lógico relativamente sencillo que es capaz de (1) realizar el secuenciamiento de las microinstrucciones y (2) generar las señales de control para ejecutar cada microinstrucción.
- Como en una unidad de control cableada, las señales de control generadas por una microinstrucción se usan para producir transferencias entre registros y operaciones de la ALU.

**E**l término *microprograma* lo acuñó M. V. Wilkes a principios de los años cincuenta [WILK51]. Wilkes propuso una aproximación al diseño de la unidad de control que era ordenada y sistemática, y evitaba la complejidad de la implementación cableada. La idea despertó la curiosidad de muchos investigadores pero se mostraba irrealizable porque necesitaba una memoria de control que fuera rápida y relativamente barata.

El número de febrero de 1964 de *Datamation* revisaba el estado del arte de la microprogramación. No había ningún sistema microprogramado de uso generalizado en aquella época, y uno de los artículos [HILL64] resumía la opinión popular del momento de que el futuro de la microprogramación «es un tanto turbio. Ninguno de los grandes fabricantes ha mostrado interés en la técnica, a pesar de que probablemente todos la hayan analizado».

La situación cambió completamente en muy pocos meses. El System/360 de IBM se anunció en abril, y todos excepto los modelos más grandes eran microprogramados. Aunque la serie 360 precedió a la disponibilidad de ROM de semiconductores, las ventajas de la microprogramación fueron lo bastante convincentes para IBM como para introducir este cambio. La microprogramación se convirtió en una técnica popular para implementar la unidad de control de los procesadores CISC. En los últimos años, la microprogramación se está usando menos, pero sigue siendo una herramienta disponible para los diseñadores de computadores. Por ejemplo, tal y como hemos visto, en el Pentium 4, las instrucciones máquina se convierten a un formato tipo RISC y la mayoría de estas nuevas instrucciones se ejecutan sin el uso de microprogramación. No obstante, algunas de las instrucciones se ejecutan usando microprogramación.

### 17.1. CONCEPTOS BÁSICOS

#### MICROINSTRUCCIONES

La unidad de control parece un dispositivo bastante sencillo. Sin embargo, implementar una unidad de control como una interconexión de elementos lógicos básicos no es una tarea fácil. El diseño debe

incluir lógica para realizar el secuenciamiento de las microoperaciones, para ejecutar las microoperaciones, para interpretar los códigos de operación, y para tomar decisiones basadas en los indicadores de la ALU. Todo este hardware es difícil de diseñar y de verificar. Además, el diseño es relativamente inflexible. Por ejemplo, es difícil cambiar el diseño si se desea añadir una nueva instrucción máquina.

Una alternativa, que se ha usado en muchos procesadores CISC, es realizar una unidad de control micropogramada.

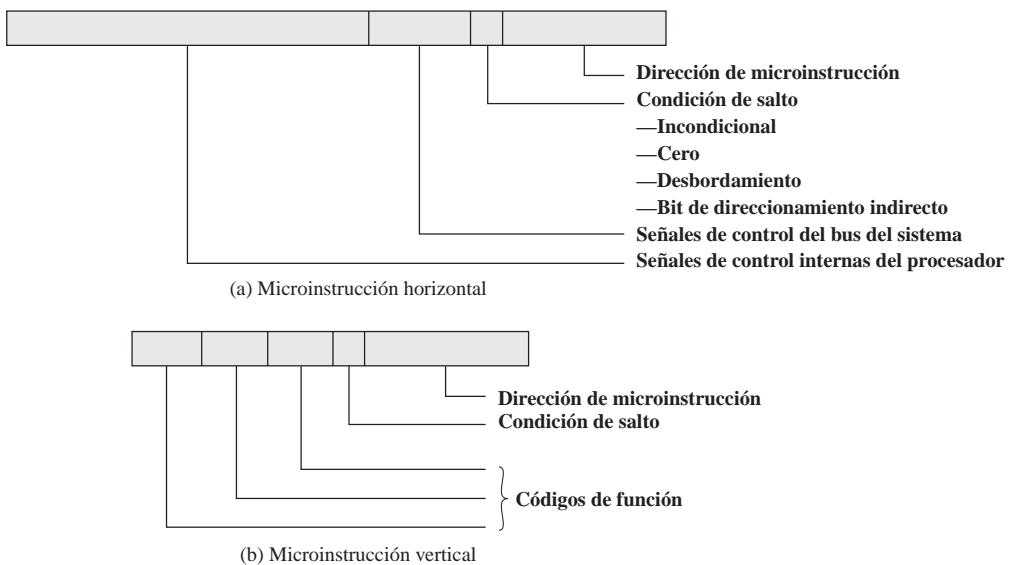
Consideremos de nuevo la Tabla 16.1. Además de indicar las señales de control, cada microoperación está descrita en notación simbólica. Esta notación tiene toda la apariencia de un lenguaje de programación. De hecho es un lenguaje, conocido como **lenguaje de micropogramación**. Cada línea describe un conjunto de microoperaciones que suceden a la vez y que se conoce como **microinstrucción**. Una secuencia de instrucciones se conoce como **micropograma** o *firmware*. Este último término refleja el hecho de que un micropograma está a mitad de camino entre hardware y software. Es más fácil diseñar en firmware que en hardware, pero es más difícil escribir un programa firmware que un programa software.

¿Cómo se puede usar el concepto de micropogramación para implementar una unidad de control? Consideremos que para cada microoperación, todo lo que la unidad de control puede hacer es generar un conjunto de señales de control. Por consiguiente, para cualquier microoperación, cada línea de control procedente de la unidad de control estará activa o inactiva. Esta condición puede, naturalmente, representarse con un dígito binario para cada línea de control. De este modo, podríamos construir una *palabra de control* en la que cada bit representara una línea de control. Entonces, cada microoperación se representaría mediante un patrón diferente de unos y ceros en la palabra de control.

Supongamos que se emplea una secuencia de palabras de control para representar la secuencia de microoperaciones ejecutadas por la unidad de control. A continuación, hemos de admitir que la secuencia de microoperaciones no es fija. Algunas veces tenemos un ciclo indirecto; otras no. Por tanto, coloquemos nuestras palabras de control en una memoria, cada palabra en una dirección única. Añadimos ahora un campo de dirección a cada palabra de control, indicando la posición de la siguiente palabra de control a ejecutar si una determinada condición es cierta (por ejemplo, que el bit de direccionamiento indirecto de una instrucción que referencia la memoria sea uno). Y añadimos además algunos bits para especificar la condición.

El resultado se conoce como **microinstrucción horizontal**; se muestra un ejemplo en la Figura 17.1a. El formato de la microinstrucción o palabra de control es el siguiente. Hay un bit para cada línea de control interna al procesador y un bit para cada línea de control del bus del sistema. Hay un campo de condición que indica la condición bajo la cual debe producirse un salto, y un campo con la dirección de la microinstrucción a ejecutar cuando el salto se produzca. Esta microinstrucción se interpreta como sigue:

1. Para ejecutar la microinstrucción, se activan todas las líneas de control cuyos bits estén a 1; y se dejan inactivas todas las líneas de control indicadas con un bit a 0. Las señales de control resultantes harán que se ejecuten una o más microoperaciones.
2. Si la condición indicada por los bits de condición es falsa, se ejecuta la siguiente microinstrucción secuencial.
3. Si la condición indicada por los bits de condición es cierta, la siguiente microinstrucción a ejecutar se indica en el campo de dirección.



**Figura 17.1.** Formatos de microinstrucción típicos.

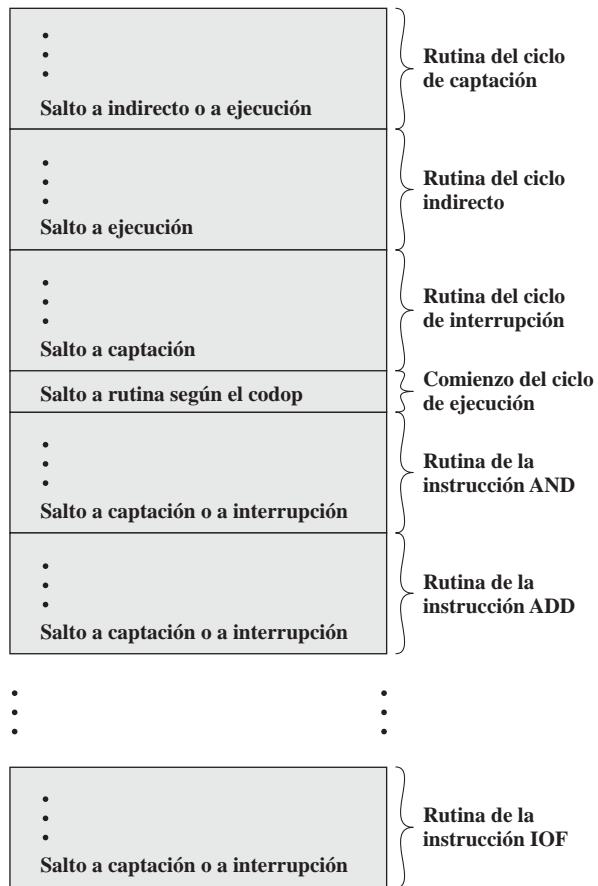
La Figura 17.2 muestra cómo se pueden organizar estas palabras de control o microinstrucciones en una **memoria de control**. Las microinstrucciones de cada rutina se ejecutarán secuencialmente. Cada rutina termina con una instrucción de bifurcación o salto indicando a dónde ir a continuación. Hay una sección especial del ciclo de ejecución cuyo único objetivo es indicar cuál de las rutinas de las instrucciones máquina (AND, ADD, etc.) se va a ejecutar a continuación, en función del código de operación actual.

La memoria de control de la Figura 17.2 es una descripción concisa de todo lo que hace la unidad de control. Define la secuencia de microoperaciones a realizar en cada ciclo (captación, indirecto, ejecución, interrupción), y especifica el secuenciamiento de estos ciclos. Si solo fuera eso, esta notación sería un recurso útil para documentar el funcionamiento de una unidad de control para un computador particular. Pero es más que eso. Es también una forma de implementar la unidad de control.

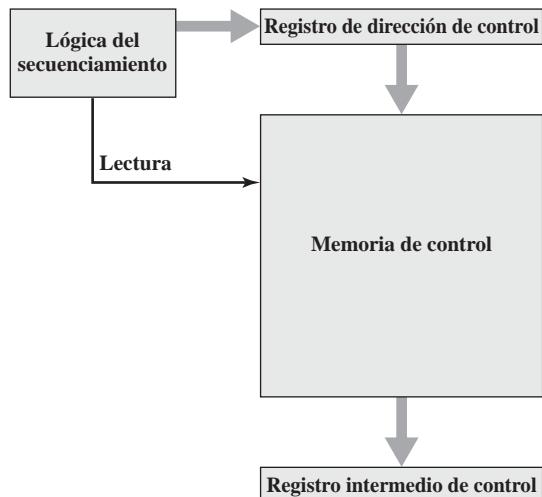
## UNIDAD DE CONTROL MICROPROGRAMADA

La memoria de control de la Figura 17.2 contiene un programa que describe el funcionamiento de la unidad de control. Resulta que podríamos implementar la unidad de control sencillamente ejecutando ese programa.

La Figura 17.3 muestra los elementos más importantes de esta implementación. El conjunto de microinstrucciones se almacena en la *memoria de control*. El *registro de dirección de control* contiene la dirección de la siguiente microinstrucción a leer. Cuando se lee una microinstrucción de la memoria de control, se transfiere al *registro intermedio de control*. La parte izquierda de ese registro (ver Figura 17.1a) se conecta a las líneas de control que salen de la unidad de control. De este modo, *leer* una microinstrucción de la memoria de control es lo mismo que *ejecutar* la microinstrucción. El tercer elemento que muestra la figura es una unidad de secuenciamiento que carga el registro de dirección de control y emite una orden de lectura.



**Figura 17.2.** Organización de la memoria de control.



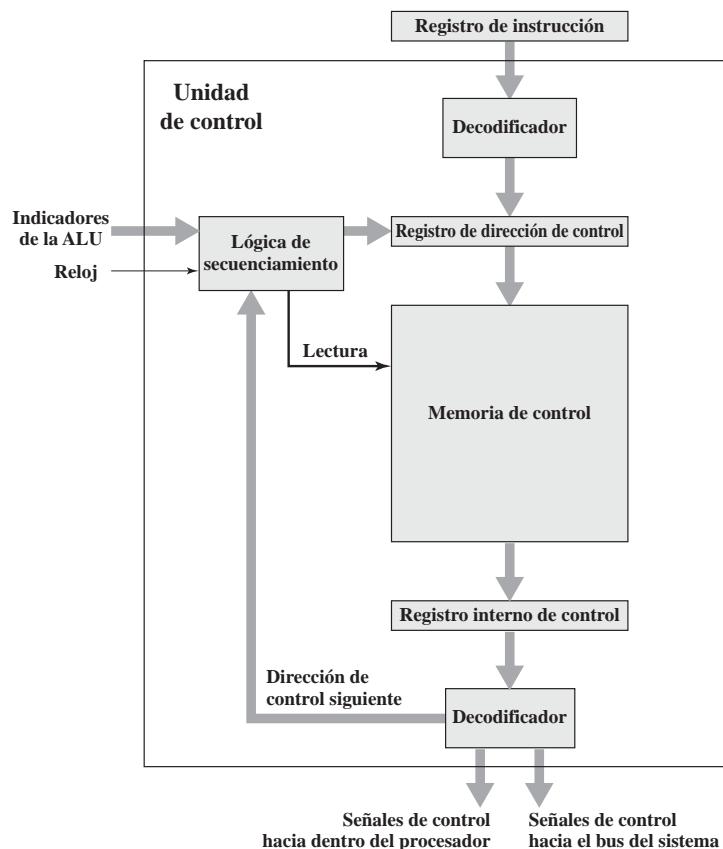
**Figura 17.3.** Microarquitectura de una unidad de control.

Examinemos esta estructura con mayor detalle, como representa la Figura 17.4. Comparándola con la Figura 16.4, vemos que la unidad de control sigue teniendo las mismas entradas (IR, indicadores de la ALU, reloj) y salidas (señales de control). La unidad de control funciona como sigue:

1. Para ejecutar una instrucción, la unidad lógica de secuenciamiento emite una orden de lectura a la memoria de control.
2. La palabra cuya dirección se especifica en el registro de dirección de control se lee en el registro intermedio de control.
3. El registro intermedio de control genera las señales de control y la información de dirección siguiente para la unidad lógica de secuenciamiento.
4. La unidad lógica de secuenciamiento carga en el registro de dirección de control una nueva dirección, basada en la información de dirección siguiente del registro intermedio de control y en los indicadores de la ALU.

Todo esto sucede durante un pulso de reloj.

El último paso recién mencionado requiere cierta elaboración. Al final de la ejecución de cada microinstrucción, la unidad lógica de secuenciamiento carga una nueva dirección en el registro de



**Figura 17.4.** Funcionamiento de una unidad de control microprogramada.

dirección de control. Dependiendo del valor de los indicadores de la ALU y del registro intermedio de control, se toma una de las tres siguientes decisiones:

- **Captar la microinstrucción siguiente:** se suma 1 al registro de dirección de control.
- **Saltar a una nueva rutina según indica una microinstrucción de salto:** el campo de dirección del registro intermedio de control se carga en el registro de dirección de control.
- **Saltar a la rutina de una instrucción máquina:** se carga el registro de dirección de control en función del código de operación almacenado en IR.

La Figura 17.4 muestra dos módulos designados como *decodificador*. El decodificador de arriba traduce el código de operación de IR en una dirección de memoria de control. El decodificador de abajo no se usa con microinstrucciones horizontales pero sí con **microinstrucciones verticales** (Figura 17.1b). Como se mencionó, en una microinstrucción horizontal cada bit del campo de control corresponde a una línea de control. En una microinstrucción vertical se usa un código para cada acción a realizar —por ejemplo, MAR ((PC)—, y el decodificador traduce este código a señales de control individuales. La ventaja de las microinstrucciones verticales es que son más compactas (ocupan menos bits) que las microinstrucciones horizontales, a costa de añadir una pequeña lógica y cierto retardo temporal.

## CONTROL DE WILKES

Como se ha mencionado, Wilkes fue el primero que propuso la utilización de una unidad de control micropogramada en 1951 [WILK51]. Más tarde elaboró su propuesta en un diseño más detallado [WILK53]. Es instructivo examinar esta propuesta inicial.

La configuración propuesta por Wilkes se representa en la Figura 17.5. El núcleo del sistema es una matriz parcialmente llena de diodos. Durante un ciclo máquina, se activa una fila de la matriz mediante un pulso. Esto produce señales en aquellos puntos en los que un diodo está presente (indicados mediante un punto en el diagrama). La primera parte de la fila genera las señales de control que gobiernan el funcionamiento del procesador. La segunda parte genera la dirección de la fila que será seleccionada mediante un pulso en el siguiente ciclo máquina. Por tanto, cada fila de la matriz es una microinstrucción y el trazado de la matriz es la memoria de control.

Al comienzo de un ciclo, la dirección de la fila a seleccionar está almacenada en el registro I. Esta dirección es la entrada del decodificador, el cual, cuando se activa mediante un pulso de reloj, selecciona una fila de la matriz. Dependiendo de las señales de control, durante el ciclo se pasa al registro II bien el código de operación almacenado en el registro de instrucción, o bien la segunda parte de la fila activada. El contenido del registro II se lleva entonces al registro I mediante un pulso de reloj. Se usan pulsos de reloj alternos para activar una fila de la matriz y para transferir el contenido del registro II al registro I. La configuración con dos registros es necesaria ya que el decodificador es sencillamente un circuito combinacional; con un único registro, la salida se convertiría en entrada dentro del mismo ciclo, originando un estado inestable.

Este esquema es muy similar al enfoque de micropogramación horizontal descrito anteriormente (Figura 17.1a). La principal diferencia es esta: en aquella descripción, el registro de dirección de control podía incrementarse en uno para acceder a la siguiente dirección. En el esquema de Wilkes, la dirección siguiente está contenida en la microinstrucción. Para permitir bifurcaciones, una fila debe

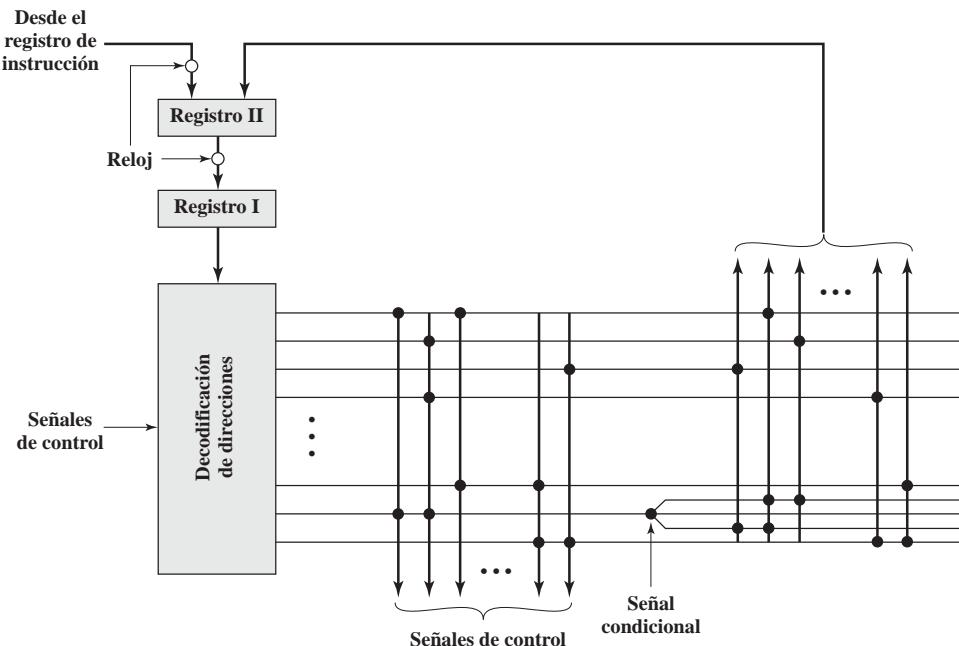


Figura 17.5. Unidad de control micropogramada de Wilkes.

contener dos partes de direcciones, controladas por una señal condicional (por ejemplo, un indicador), como muestra la figura.

Después de proponer este esquema, Wilkes proporciona un ejemplo de su utilización para implementar la unidad de control de una máquina sencilla. Este ejemplo, el primer diseño conocido de un procesador microprogramado, merece repetirse aquí porque ilustra muchos de los principios contemporáneos de la microprogramación.

El procesador de la máquina hipotética incluye los siguientes registros:

- A multiplicando
- B acumulador (mitad menos significativa)
- C acumulador (mitad más significativa)
- D registro de desplazamiento

Además, hay tres registros y dos indicadores de un bit accesibles solo por la unidad de control. Los registros son los siguientes:

- E sirve como registro de dirección de memoria (MAR) y como almacenamiento temporal
- F contador de programa
- G otro registro temporal, usado en cálculos

La Tabla 17.1 enumera el conjunto de instrucciones máquina para este ejemplo. La Tabla 17.2 contiene el conjunto completo de microinstrucciones, expresadas de forma simbólica, que implementa

**Tabla 17.1.** Conjunto de instrucciones máquina del ejemplo de Wilkes.

| Orden     | Efecto de la orden   |
|-----------|--|
| <i>An</i> | $C(Ac) + C(n)$ al $Ac_1$   |
| <i>Sn</i> | $C(Ac) - C(n)$ al $Ac_1$   |
| <i>Hn</i> | $C(n)$ al $Ac_2$   |
| <i>Vn</i> | $C(Ac_2) \times C(n)$ al $Ac_1$ , donde $C(n) \geq 0$  |
| <i>Tn</i> | $C(Ac_1)$ a $n$ , 0 al $Ac$  |
| <i>Un</i> | $C(Ac_1)$ a $n$  |
| <i>Rn</i> | $C(Ac) \times 2^{-(n+1)}$ al $Ac$  |
| <i>Ln</i> | $C(Ac) \times 2^{n+1}$ al $Ac$   |
| <i>Gn</i> | Si $C(Ac) < 0$ , transferir el control a $n$ ; si $C(Ac) \geq 0$ , ignorar (es decir, continuar secuencialmente) |
| <i>In</i> | Leer el siguiente carácter del mecanismo de entrada en $n$   |
| <i>On</i> | Enviar $C(n)$ al mecanismo de salida   |

Notación:

 $Ac$  = acumulador $Ac_1$  = mitad más significativa del acumulador $Ac_2$  = mitad menos significativa del acumulador $n$  = posición de memoria  $n$  $C(X)$  = contenido de  $X$  ( $X$  = registro o posición de almacenamiento)

la unidad de control. Solo es necesario un total de 38 microinstrucciones para definir el sistema completamente.

La primera columna contiene la dirección (número de fila) de cada microinstrucción. Las direcciones referentes a códigos de operación están etiquetadas. De este modo, cuando se encuentra el código de operación de la instrucción de suma (A), se ejecuta la microinstrucción de la posición 5. Las columnas 2 y 3 expresan las acciones a realizar por la ALU y por la unidad de control, respectivamente. Cada expresión simbólica ha de traducirse en un conjunto de señales de control (bits de la microinstrucción). Las columnas 4 y 5 tienen que ver con la modificación y el uso de los dos indicadores (biestables). La columna 4 especifica la señal que ajusta el indicador. Por ejemplo,  $(1)C_s$  significa que el indicador número 1 se ajusta según el bit de signo del número contenido en el registro  $C$ . Si la columna 5 contiene un identificador de indicador, las columnas 6 y 7 contienen las dos direcciones de microinstrucción alternativas. Si no, la columna 6 especifica la dirección de la siguiente microinstrucción a captar.

Las instrucciones de la 0 a la 4 constituyen el ciclo de captación. La microinstrucción 4 presenta el código de operación a un decodificador, que genera la dirección de la microinstrucción a captar correspondiente a la instrucción máquina en curso. El lector debería ser capaz de deducir el funcionamiento completo de la unidad de control a partir de un cuidadoso estudio de la Tabla 17.2.

## VENTAJAS E INCONVENIENTES

La ventaja principal que aporta el uso de la micropogramación para implementar una unidad de control es que simplifica su diseño. Por consiguiente, su implementación resulta más barata y menos

**Tabla 17.2.** Microinstrucciones del ejemplo de Wilkes.

Notación:  $A, B, C\dots$  representan los diversos registros de las unidades aritmética y de registros de control.  $C a D$  indica que los circuitos de conmutación conectan la salida del circuito  $C$  a la entrada del registro  $D$ ;  $(D + A)$  a  $C$  indica que la salida del registro  $A$  se conecta a una entrada de la unidad de suma (la salida de  $D$  está conectada permanentemente a la otra entrada), y la salida del sumador al registro  $C$ . Un símbolo numérico  $n$  entre comillas (es decir, 'n') representa la fuente cuya salida es el número  $n$  en unidades del dígito menos significativo.

|      | Unidad aritmética       | Unidad de registros de control | Biestable condicional |     | Microinstrucción siguiente |   |
|------|-------------------------|--------------------------------|-----------------------|-----|----------------------------|---|
|      |                         |                                | Ajuste                | Uso | 0                          | 1 |
| 0    |                         | $F a G y E$                    |                       |     | 1                          |   |
| 1    |                         | $(G + '1') a F$                |                       |     | 2                          |   |
| 2    |                         | Mem. a $G$                     |                       |     | 3                          |   |
| 3    |                         | $G a E$                        |                       |     | 4                          |   |
| 4    |                         | $E a$ decodif.                 |                       |     | —                          |   |
| A 5  | $C a D$                 |                                |                       |     | 16                         |   |
| S 6  | $C a D$                 |                                |                       |     | 17                         |   |
| H 7  | Mem. a $B$              |                                |                       |     | 0                          |   |
| V 8  | Mem. a $A$              |                                |                       |     | 27                         |   |
| T 9  | $C a$ Mem.              |                                |                       |     | 25                         |   |
| U 10 | $C a$ Mem.              |                                |                       |     | 0                          |   |
| R 11 | $B a D$                 | $E a G$                        |                       |     | 19                         |   |
| L 12 | $C a D$                 | $E a G$                        |                       |     | 22                         |   |
| G 13 |                         | $E a G$                        | $(1)C_s$              |     | 18                         |   |
| I 14 | Entrada a Mem.          |                                |                       |     | 0                          |   |
| O 15 | Mem. a salida           |                                |                       |     | 0                          |   |
| 16   | $(D + \text{Mem.}) a C$ |                                |                       |     | 0                          |   |
| 17   | $(D - \text{Mem.}) a C$ |                                |                       |     | 0                          |   |
| 18   |                         |                                |                       | 1   | 0                          | 1 |
| 19   | $D a B (R)^*$           | $(G - '1') a E$                |                       |     | 20                         |   |
| 20   | $C a D$                 |                                | $(1)E_s$              |     | 21                         |   |
| 21   | $D a C (R)$             |                                |                       | 1   | 11                         | 0 |
| 22   | $D a C (L)^{\dagger}$   | $(G - '1') a E$                |                       |     | 23                         |   |
| 23   | $B a D$                 |                                | $(1)E_s$              |     | 24                         |   |
| 24   | $D a B$                 |                                |                       |     | 12                         | 0 |
| 25   | '0' a $B$               |                                |                       |     | 26                         |   |
| 26   | $B a C$                 |                                |                       |     | 0                          |   |
| 27   | '0' a $C$               | $'18' a E$                     |                       |     | 28                         |   |
| 28   | $B a D$                 | $E a G$                        | $(1)B_1$              |     | 29                         |   |

(Continúa)

**Tabla 17.2.** Microinstrucciones del ejemplo de Wilkes (*continuación*).

|    | Unidad aritmética      | Unidad de registros de control | Biestable condicional |     | Microinstrucción siguiente |    |
|----|------------------------|--------------------------------|-----------------------|-----|----------------------------|----|
|    |                        |                                | Ajuste                | Uso | 0                          | 1  |
| 29 | $D \text{ a } B (R)$   | $(G - '1') \text{ a } E$       |                       |     | 30                         |    |
| 30 | $C \text{ a } D (R)$   |                                | $(2)E_s$              | 1   | 31                         | 32 |
| 31 | $D \text{ a } C$       |                                |                       | 2   | 28                         | 33 |
| 32 | $(D + A) \text{ a } C$ |                                |                       | 2   | 28                         | 33 |
| 33 | $B \text{ a } D$       |                                | $(1)B_1$              |     | 34                         |    |
| 34 | $D \text{ a } B (R)$   |                                |                       |     | 35                         |    |
| 35 | $C \text{ a } D (R)$   |                                |                       | 1   | 36                         | 37 |
| 36 | $D \text{ a } C$       |                                |                       |     | 0                          |    |
| 37 | $(D - A) \text{ a } C$ |                                |                       |     | 0                          |    |

\* Desplazamiento a la derecha (*Right shift*). Los circuitos de conmutación de la unidad aritmética están organizados de tal forma que el dígito menos significativo del registro  $C$  se lleva a la posición más significativa del registro  $B$  durante las microoperaciones de desplazamiento a la derecha, y el dígito más significativo del registro  $C$  (dígito de signo) se repite (realizándose, por tanto, la corrección para números negativos).

† Desplazamiento a la izquierda (*Left shift*). Los circuitos de conmutación están dispuestos de manera similar para pasar el dígito más significativo del registro  $B$  a la posición menos significativa del registro  $C$  durante las microoperaciones de desplazamiento a la izquierda.

propensa a errores. Una unidad de control *cableada* contendrá lógica compleja para hacer el secuenciamiento a través de las muchas microoperaciones del ciclo de instrucción. Por otra parte, los decodificadores y la unidad lógica de secuenciamiento de una unidad de control micropogramada son elementos lógicos muy sencillos.

El principal inconveniente de una unidad micropogramada es que será algo más lenta que una unidad cableada de tecnología comparable. A pesar de ello, la micropogramación es la técnica dominante para implementar unidades de control en las arquitecturas CISC puras, debido a su facilidad de implementación. Los procesadores RISC, dado que tienen un formato de instrucción más sencillo, emplean normalmente unidades de control cableadas. Examinaremos ahora con más detalle la solución micropogramada.

## 17.2. SECUENCIAMIENTO DE MICROINSTRUCCIONES

Las dos tareas básicas realizadas por una unidad de control micropogramada son las siguientes:

- **Secuenciamiento de microinstrucciones:** obtener la siguiente microinstrucción de la memoria de control.
- **Ejecución de microinstrucciones:** generar las señales de control necesarias para ejecutar la microinstrucción.

Al diseñar una unidad de control, las dos tareas deben considerarse conjuntamente, ya que ambas afectan al formato de la microinstrucción y a la temporización de la unidad de control. En esta

sección, nos centramos en el secuenciamiento y hablaremos lo mínimo posible sobre los temas de formato y temporización. Estos temas se examinarán más detalladamente en la sección siguiente.

## CONSIDERACIONES RESPECTO AL DISEÑO

Hay dos cuestiones involucradas en el diseño de una técnica de secuenciamiento de microinstrucciones: el tamaño de la microinstrucción y el tiempo de generación de la dirección. El primer asunto es evidente: minimizar el tamaño de la memoria de control reduce el coste de este componente. El segundo asunto es sencillamente un deseo de ejecutar las microinstrucciones tan rápido como sea posible.

Cuando se ejecuta un microprograma, la dirección de la siguiente microinstrucción a ejecutar está en una de estas situaciones:

- Viene determinada por el registro de instrucción.
- Es la siguiente dirección secuencial.
- Es el destino de un salto.

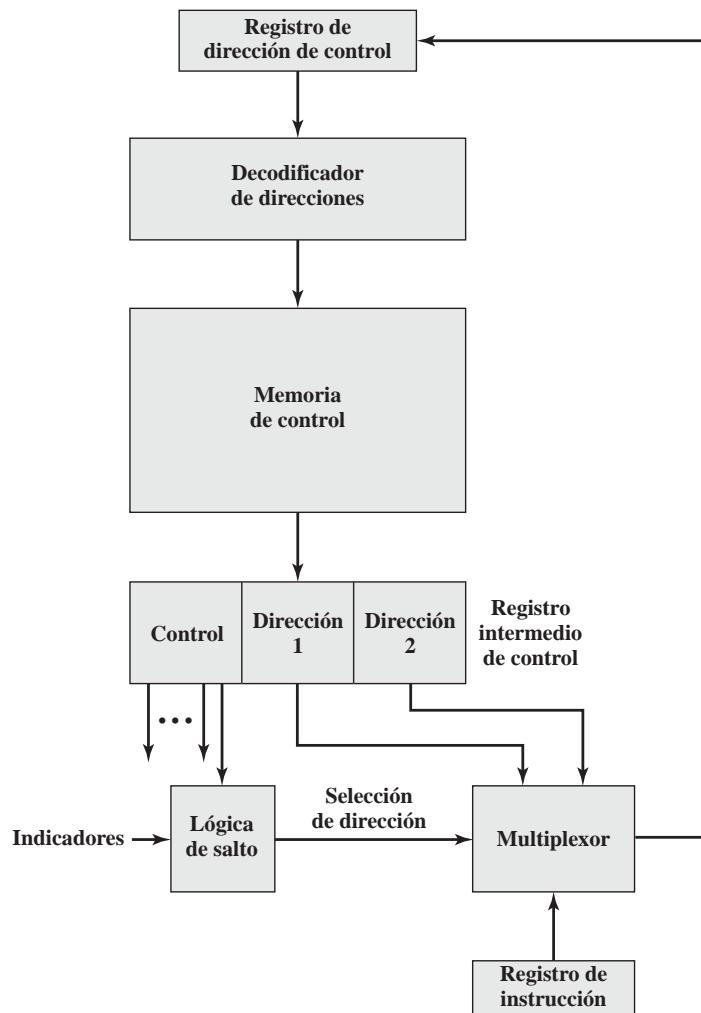
La primera situación tiene lugar solo una vez por ciclo de instrucción, justo tras la captación de la instrucción. La segunda situación es la más común en la mayoría de los diseños. No obstante, el diseño no se puede optimizar solo para los accesos secuenciales. Los saltos, tanto condicionales como incondicionales, son una parte necesaria del microprograma. Además, las secuencias de microinstrucciones tienden a ser cortas; una de cada tres o cuatro microinstrucciones podría ser un salto [SIEW82]. Por consiguiente, es importante diseñar técnicas compactas y eficientes en cuanto al tiempo para los saltos a microinstrucciones.

## TÉCNICAS DE SECUENCIAMIENTO

A partir de la microinstrucción en curso, de los indicadores de condición, y del contenido del registro de instrucción, hay que generar una dirección de la memoria de control para la siguiente microinstrucción. Se han usado numerosas técnicas. Podemos agruparlas en tres categorías, como ilustran las Figuras 17.6 a 17.8. Estas categorías se basan en el formato de la información de dirección de la microinstrucción:

- Dos campos de dirección.
- Un único campo de dirección.
- Formato variable.

La técnica más sencilla es tener dos campos de dirección en cada microinstrucción. La Figura 17.6 indica cómo se va a usar esta información. Se tiene un multiplexor que sirve de destino de los dos campos de dirección y del registro de instrucción. Basándose en la entrada de selección de dirección, el multiplexor transmite el código de operación o una de las dos direcciones al registro de dirección de control (*control address register*, CAR). El CAR se decodifica a continuación para producir la dirección de la siguiente microinstrucción. Las señales de selección de dirección son suministradas por un módulo de lógica de salto, cuyas entradas son los indicadores de la unidad de control y ciertos bits de la parte de control de la microinstrucción.

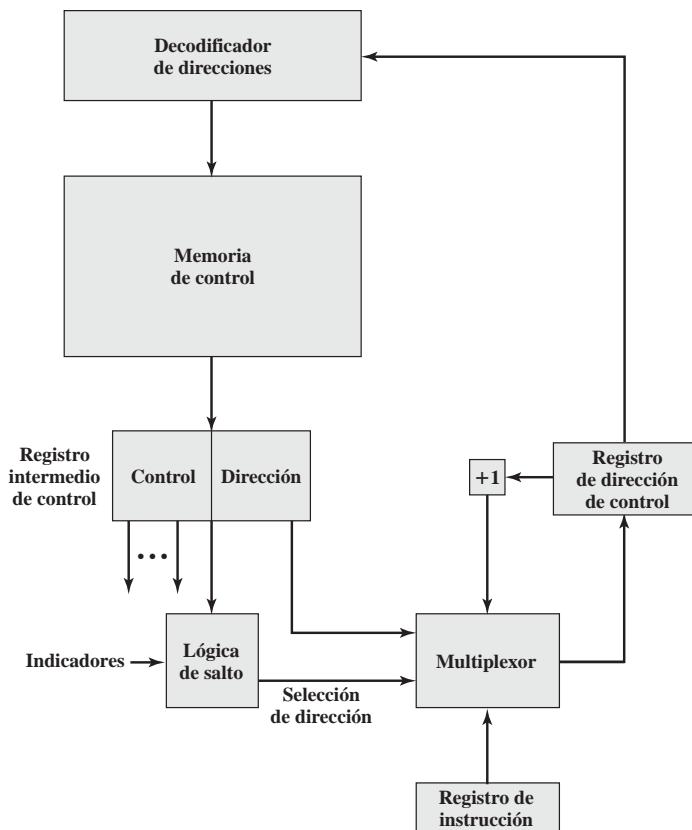


**Figura 17.6.** Lógica de control de salto con dos campos de dirección.

Aunque el método de dos direcciones es sencillo, necesita más bits por microinstrucción que las otras técnicas. Con alguna lógica adicional, se puede conseguir cierto ahorro. Una aproximación frecuente es tener un único campo de dirección (Figura 17.7). Con este enfoque, las opciones para la dirección siguiente son:

- Campo de dirección.
- Código del registro de instrucción.
- Siguiente dirección secuencial.

Las señales de selección de dirección determinan qué opción se escoge. Esta técnica reduce el número de campos de dirección a uno. Observe, sin embargo, que el campo de dirección a menudo no se usa. Por tanto, hay cierta ineficiencia en este esquema de codificación.



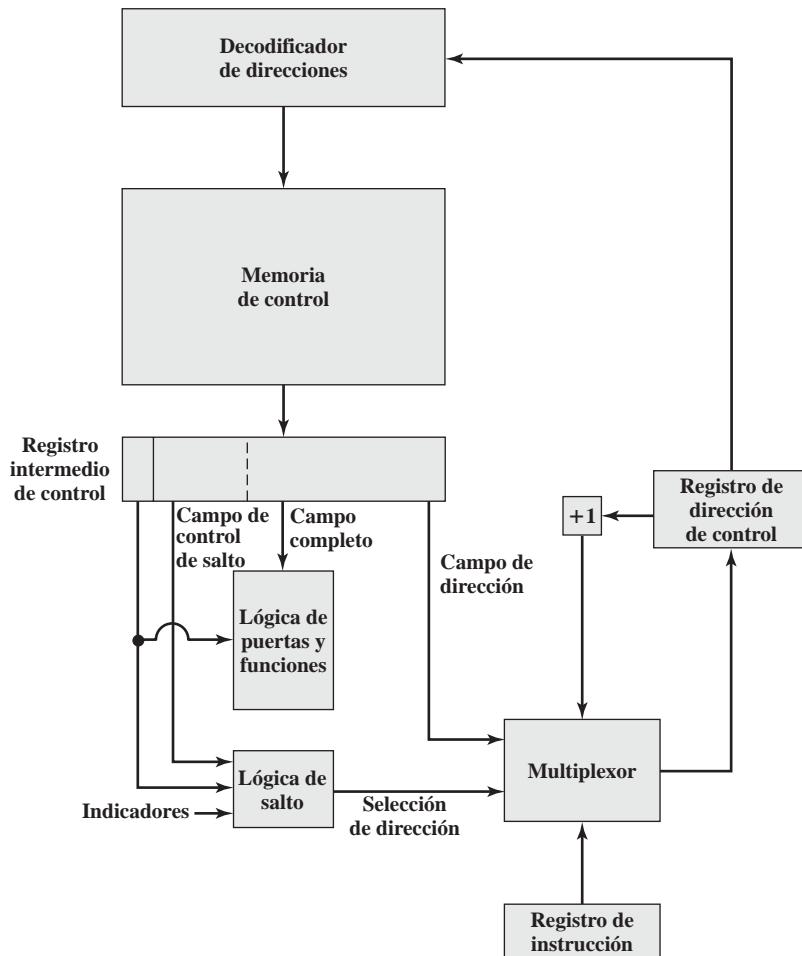
**Figura 17.7.** Lógica de control de salto con un único campo de dirección.

Otro método es proporcionar dos formatos de microinstrucción totalmente diferentes (Figura 17.8). Un bit designa qué formato se utilizará. En uno de los dos formatos, los demás bits se usan para activar señales de control. En el otro formato, algunos bits controlan el módulo de lógica de salto y los bits restantes suministran la dirección. En el primer formato, la dirección siguiente es la siguiente dirección secuencial o una dirección derivada del registro de instrucción. En el segundo formato, se especifica un salto condicional o incondicional. Un inconveniente de esta aproximación, tal como se ha descrito, es que se consume un ciclo completo por cada microinstrucción de salto. Con las otras técnicas, la generación de la dirección sucede como parte del mismo ciclo en el que se generan las señales de control, lo cual minimiza los accesos a la memoria de control.

Las aproximaciones que se acaban de describir son generales. Las implementaciones específicas con frecuencia usarán una variación o una combinación de estas técnicas.

## GENERACIÓN DE DIRECCIONES

Hemos enfocado el problema del secuenciamiento desde el punto de vista de las consideraciones sobre el formato y de los requisitos de lógica en general. Otro punto de vista es considerar las diversas formas de obtener o calcular la siguiente dirección.



**Figura 17.8.** Lógica de control de salto con formato variable.

La Tabla 17.3 relaciona las diversas técnicas de generación de la dirección. Estas se pueden dividir en técnicas explícitas, en las que la dirección aparece explícitamente en la microinstrucción, y técnicas implícitas, que requieren lógica adicional para generar la dirección.

Nos hemos ocupado esencialmente de las técnicas explícitas. Con un enfoque de dos campos, hay dos direcciones alternativas disponibles en cada microinstrucción. Usando un único campo de

**Tabla 17.3.** Técnicas de generación de direcciones de microinstrucción.

| Explícitas          | Implícitas       |
|---------------------|------------------|
| Dos campos          | Traducción       |
| Salto incondicional | Adición          |
| Salto condicional   | Control residual |

dirección o un formato variable, se pueden implementar varias instrucciones de salto. Una instrucción de salto condicional depende de los siguientes tipos de información:

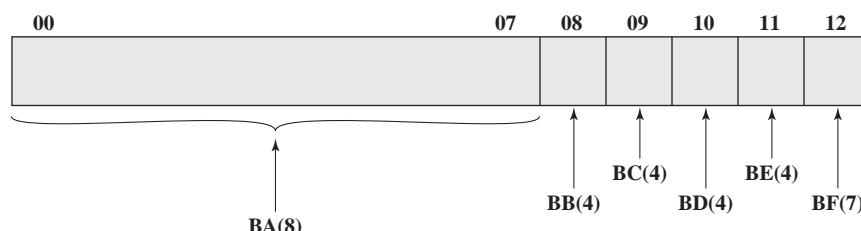
- Indicadores de la ALU.
- Parte del código de operación o campos de modo de direccionamiento de la instrucción máquina.
- Partes de un registro seleccionado, tales como el bit de signo.
- Bits de estado dentro de la unidad de control.

También se usan frecuentemente algunas técnicas implícitas. Una de ellas, la traducción, se necesita en casi todos los diseños. La parte de una instrucción máquina que contiene el código de operación se traduce a una dirección de microinstrucción. Esto ocurre solo una vez por ciclo de instrucción.

Una técnica implícita habitual consiste en combinar o sumar dos partes de una dirección para formar la dirección completa. Este procedimiento fue adoptado por la familia IBM S/360 [TUCK67] y usado por muchos de los modelos S/370. Usaremos el IBM 3033 como ejemplo.

El registro de dirección de control del IBM 3033, de trece bits, se ilustra en la Figura 17.9. Se pueden distinguir dos partes en la dirección. Los ocho bits más significativos (00-07) no cambian normalmente de un ciclo de microinstrucción al siguiente. Durante la ejecución de una microinstrucción, estos ocho bits se copian directamente desde un campo de ocho bits de la microinstrucción (el campo BA) en los ocho bits más significativos del registro de dirección de control. Ello define un bloque de 32 microinstrucciones en la memoria de control. Los otros cinco bits del registro de dirección de control se ajustan para especificar la dirección concreta de la microinstrucción a captar a continuación. Cada uno de estos bits viene determinado por un campo de cuatro bits (excepto uno por un campo de siete bits) de la microinstrucción en curso; cada campo especifica la condición para ajustar el bit correspondiente. Por ejemplo, un bit del registro de dirección de control puede ponerse a 1 o a 0 dependiendo de si se produjo acarreo en la última operación de la ALU.

La última técnica de la lista de la Tabla 17.3 se denomina *control residual*. Esta aproximación implica el uso de una dirección de microinstrucción guardada previamente en un almacenamiento temporal dentro de la unidad de control. Por ejemplo, algunos conjuntos de microinstrucciones están dotados de la posibilidad de hacer llamadas a subrutinas. Un registro interno o una pila de registros se usan para guardar las direcciones de retorno. Un ejemplo de esta técnica aparece en el LSI-11, que examinamos ahora.



**Figura 17.9.** Registro de dirección de control del IBM 3033.

## SECUENCIAMIENTO DE MICROINSTRUCCIONES EN EL LSI-11

El microcomputador LSI-11 es una versión del PDP-11 con los componentes principales del sistema en una misma tarjeta. El LSI-11 está implementado usando una unidad de control micropogramada [SEBE76].

El LSI-11 utiliza microinstrucciones de 22 bits y una memoria de control de 2Kpalabras de 22 bits. La dirección de la siguiente microinstrucción se determina de una de estas cinco formas:

- **Dirección secuencial siguiente:** en ausencia de otras instrucciones, el registro de dirección de control de la unidad de control se incrementa en 1.
- **Traducción del código de operación:** al comienzo de cada ciclo de instrucción, la dirección de la siguiente microinstrucción viene determinada por el código de operación.
- **Llamada/retorno de subrutina:** explicada más abajo.
- **Comprobación de interrupciones:** ciertas microinstrucciones especifican una comprobación de interrupciones. Si ha ocurrido una interrupción, ello determina la dirección de la siguiente microinstrucción.
- **Salto:** se usan microinstrucciones de salto condicional e incondicional.

Se proporciona la posibilidad de subrutinas de un nivel. Un bit de cada microinstrucción se dedica a esta tarea. Cuando el bit está a uno, un registro de retorno de once bits se carga con el contenido actualizado del registro de dirección de control. Una instrucción posterior que especifique un retorno hará que se cargue el registro de dirección de control con el contenido del registro de retorno.

El retorno es una forma de microinstrucción de salto incondicional. Otra forma de salto incondicional hace que se carguen los bits del registro de dirección de control con once bits de la microinstrucción. La microinstrucción de salto condicional hace uso de un código de comprobación de cuatro bits dentro de la microinstrucción. Este código especifica la comprobación de diversos códigos de condición de la ALU para determinar la decisión de salto. Si la condición no es cierta, se escoge la siguiente dirección secuencial. Si es cierta, los ocho bits menos significativos del registro de dirección de control se cargan con ocho bits de la microinstrucción. Esto permite el salto dentro de una página de memoria de 256 palabras.

Como puede observarse, el LSI-11 incluye un potente secuenciamiento de direcciones dentro de la unidad de control. Ello da al micropogramador una flexibilidad apreciable y puede facilitar la tarea de la micropogramación. Por otra parte, esta aproximación requiere más lógica en la unidad de control que otra con menores capacidades.

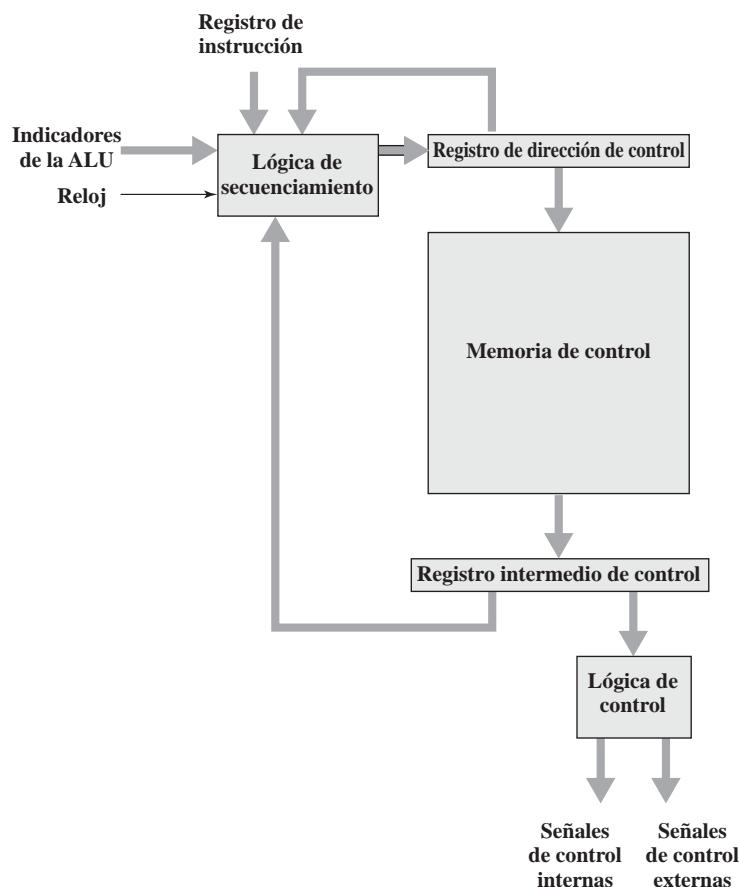
### 17.3. EJECUCIÓN DE MICROINSTRUCCIONES

El ciclo de microinstrucción es el evento básico de un procesador micropogramado. Cada ciclo de compone de dos partes: captación y ejecución. La parte de captación depende de la generación de una dirección de microinstrucción, que fue tratada en la sección precedente. Esta sección se ocupa de la ejecución de una microinstrucción.

Recordemos que el resultado de la ejecución de una microinstrucción es la generación de señales de control. Algunas de estas señales controlan puntos internos del procesador. Las demás señales van al bus de control externo o a otras interfaces externas. Como una función accesoria, se determina la dirección de la siguiente microinstrucción.

La descripción precedente sugiere la organización de la unidad de control que se muestra en la Figura 17.10. Esta versión ligeramente revisada de la Figura 17.4 subraya el centro de atención de esta sección. Los principales módulos de este diagrama ya deben estar claros. El módulo de lógica de secuenciamiento contiene la lógica que realiza las funciones estudiadas en la sección anterior. Genera la dirección de la siguiente microinstrucción, usando como entradas el registro de instrucción, los indicadores de la ALU, el registro de dirección de control (para incrementarlo) y el registro intermedio de control. El último puede proporcionar una dirección real, bits de control o ambos. Este módulo está controlado por un reloj que determina la temporización del ciclo de microinstrucción.

El módulo de lógica de control genera las señales de control en función de algunos de los bits de la microinstrucción. Debería quedar claro que el formato y el contenido de la microinstrucción determinarán la complejidad del módulo de lógica de control.



**Figura 17.10.** Organización de la unidad de control.

## UNA TAXONOMÍA DE LAS MICROINSTRUCCIONES

Las microinstrucciones se pueden clasificar de varias formas. Las distinciones que generalmente se hacen en la bibliografía incluyen

- Vertical-horizontal.
- Empaquetada/no empaquetada.
- Microprogramación *hard/soft*.
- Codificación directa/indirecta.

Todas ellas se refieren al formato de la microinstrucción. Ninguno de estos términos se ha usado de una manera coherente y precisa en la bibliografía. No obstante, un examen de estas parejas de cualidades sirve para aclarar las diferentes alternativas en el diseño de las microinstrucciones. En los siguientes párrafos consideraremos primero el principal asunto, referente al diseño, que subyace en todas estas parejas de características alternativas, y después consideraremos los conceptos que cada pareja de alternativas sugiere.

En la propuesta original de Wilkes [WILK51], cada bit de una microinstrucción producía directamente una señal de control o un bit de la dirección siguiente. Hemos visto, en la sección precedente, que son posibles esquemas de secuenciamiento de direcciones más complejos que usan menos bits por microinstrucción. Tales esquemas requieren un módulo de lógica de secuenciamiento más complejo. Existe un tipo de compromiso semejante para la parte de la microinstrucción que ataña a las señales de control. Se pueden ahorrar bits de la palabra de control codificando la información de control, y decodificándola más tarde para producir las señales de control.

¿Cómo puede hacerse esta codificación? Para responder a esta pregunta consideremos que hay un total de  $K$  señales de control internas y externas diferentes que tiene que generar la unidad de control. En el esquema de Wilkes se dedicarían a este propósito  $K$  bits de la microinstrucción. Esto permite que se puedan generar las  $2^K$  combinaciones posibles de señales de control durante cualquier ciclo de instrucción. Pero somos capaces de hacerlo mejor si observamos que no todas las combinaciones posibles se usarán. Veamos algunos ejemplos:

- Dos fuentes no se pueden llevar al mismo destino (por ejemplo,  $C_2$  y  $C_8$  en la Figura 16.5).
- Un registro no puede ser a la vez fuente y destino (por ejemplo,  $C_5$  y  $C_{12}$  en la Figura 16.5).
- Solo un patrón de señales de control se puede presentar a la ALU cada vez.
- Solo un patrón de señales de control se puede presentar al bus de control externo cada vez.

De este modo, para un procesador dado, puede hacerse una lista con todas las posibles combinaciones de señales de control admisibles, obteniendo un número de posibilidades  $Q < 2^K$  que podrían codificarse con  $\log_2 Q$  bits, siendo  $(\log_2 Q) < K$ . Esta sería la forma más estricta posible de codificación que preserva todas las combinaciones permisibles de señales de control. En la práctica, este sistema de codificación no se usa, por dos razones:

- Es tan difícil de programar como un esquema decodificado puro (como el de Wilkes). Este punto se discutirá más a fondo dentro de poco.
- Requiere un módulo de lógica de control complejo y, por consiguiente, lento.

En lugar de eso, se adoptan algunos compromisos. Los hay de dos tipos:

- Se usan más bits de los estrictamente necesarios para codificar las posibles combinaciones.
- Algunas combinaciones que son físicamente permisibles no se pueden codificar.

El último tipo de compromiso tiene el efecto de reducir el número de bits. El resultado neto, sin embargo, es que se usan más bits que  $\log_2 Q$ .

En la siguiente subsección discutiremos técnicas de codificación específicas. El resto de esta subsección se ocupa de los efectos de la codificación y de los diversos términos usados para describirla.

Basándonos en lo anterior, podemos ver que el campo de señales de control del formato de microinstrucción se encuadra dentro de un espectro. En un extremo, hay un bit para cada señal de control; en el otro extremo, se usa un formato muy codificado. La Tabla 17.4 muestra otras características de una unidad de control microprogramada que también se encuadran dentro de espectros de posibilidades y que, por lo general, dependen del espectro del grado de codificación.

La segunda pareja de características de la tabla es bastante evidente. El esquema puro de Wilkes es el que requiere más bits. También debería estar claro que este extremo ofrece la visión más detallada del hardware. El microprogramador maneja individualmente cada señal de control. La codificación se hace de tal modo que se agrupan funciones o recursos, y debido a ello el microprogramador ve el procesador a un nivel más alto, menos detallado. Además, la codificación se diseña para facilitar la microprogramación. De nuevo, debería quedar claro que la labor de comprender y orquestar el uso de todas las señales de control es difícil. Como se mencionó, una de las consecuencias típicas de la codificación es que impide el uso de ciertas combinaciones que serían permisibles si no existiera esta.

El párrafo precedente discute el diseño de la microinstrucción desde el punto de vista del microprogramador. Pero el grado de codificación también puede considerarse viendo sus efectos sobre el

**Tabla 17.4.** El espectro de las microinstrucciones.

| <b>Características</b>   |  |
|--|--|
| Microinstrucción no codificada<br>Muchos bits<br>Visión detallada del hardware<br>Difícil de programar<br>Concurrencia explotada completamente<br>Poca o ninguna lógica de control<br>Ejecución rápida<br>Optimización de las prestaciones | Microinstrucción muy codificada<br>Pocos bits<br>Visión global del hardware<br>Fácil de programar<br>Concurrencia no explotada completamente<br>Lógica de control compleja<br>Ejecución lenta<br>Optimización de la programación |
| <b>Terminología</b>  |  |
| No empaquetada<br>Horizontal<br><i>Hard</i>  | Empaquetada<br>Vertical<br><i>Soft</i>   |

hardware. Con un formato puro no codificado, se necesita poca o ninguna lógica de decodificación; cada bit genera una señal de control individual. Conforme se usan esquemas de codificación más compactos y globales, se necesita una lógica de decodificación más compleja. Esto puede afectar proporcionalmente a las prestaciones. Se necesita más tiempo para propagar las señales a través de las puertas de una lógica de control más compleja. Por tanto, la ejecución de microinstrucciones codificadas tarda más tiempo que la ejecución de las no codificadas.

De este modo, todas las características relacionadas en la Tabla 17.4 se distribuyen a lo largo de un espectro de estrategias de diseño. En general, un diseño que cae hacia el extremo izquierdo del espectro se propone optimizar las prestaciones de la unidad de control. Los diseños del extremo derecho están más interesados en optimizar el proceso de micropogramación. En efecto, los conjuntos de microinstrucciones cercanos al extremo derecho del espectro se parecen mucho a los conjuntos de instrucciones máquina. Un buen ejemplo de ello es el diseño del LSI-11, descrito más adelante en esta sección. Típicamente, cuando el objetivo es sencillamente implementar una unidad de control, el diseño estará cerca del extremo izquierdo del espectro. El diseño del IBM 3033, que se estudiará luego, está en esta categoría. Como veremos más adelante, algunos sistemas permiten que diferentes usuarios construyan micropogramas diferentes usando el mismo tipo de microinstrucción. En este segundo caso, el diseño caerá probablemente cerca del extremo derecho del espectro.

Podemos ocuparnos ahora de alguna terminología introducida anteriormente. La Tabla 17.4 indica cómo tres de estas parejas de términos se relacionan con el espectro de las microinstrucciones. Fundamentalmente, cualquiera de estas parejas describe la misma cosa pero da importancia a diferentes características de diseño.

El grado de empaquetamiento se relaciona con el grado de identificación entre una tarea de control determinada y algunos bits específicos de la microinstrucción. Cuanto más *empaquetados* están los bits, un número dado de bits contiene más información. Por lo tanto, el empaquetamiento se relaciona con la codificación. Los términos *horizontal* y *vertical* se relacionan con el ancho relativo de las microinstrucciones. [SIEW82] indica a modo de regla empírica que las microinstrucciones verticales tienen longitudes en el rango de 16 a cuarenta bits, y las microinstrucciones horizontales longitudes de cuarenta a cien bits. Los términos micropogramación *hard* y *soft* se utilizan para indicar el grado de proximidad a las señales de control subyacentes y a la configuración del hardware. Los micropogramas *hard* generalmente son fijos y se sitúan en memoria de sólo lectura. Los micropogramas *soft* son más cambiables y sugieren una micropogramación por parte del usuario.

La otra pareja de términos mencionados al comienzo de esta subsección se refiere a codificación directa frente a indirecta, un asunto al que volvemos ahora.

## CODIFICACIÓN DE LAS MICROINSTRUCCIONES

En la práctica, las unidades de control micropogramadas no se diseñan utilizando un formato de microinstrucción puro no codificado u horizontal. Se hace uso, al menos, de algún grado de codificación para reducir el ancho de la memoria de control y simplificar la tarea de la micropogramación.

La técnica básica de codificación se ilustra en la Figura 17.11a. La microinstrucción se organiza como un conjunto de campos. Cada campo contiene un código que, tras la decodificación, activa una o más señales de control.

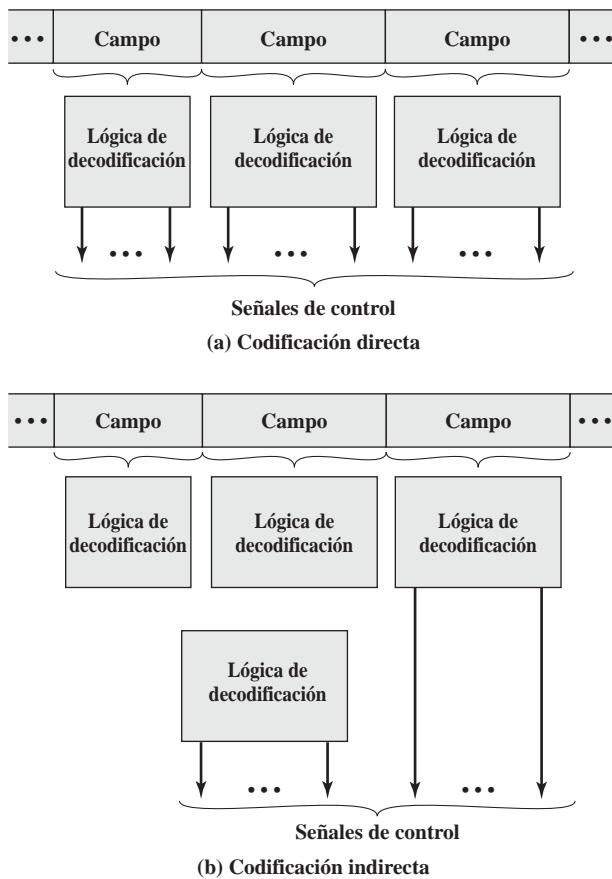


Figura 17.11. Codificación de la microinstrucción.

Consideremos las implicaciones de este esquema. Cuando la microinstrucción se ejecuta, cada campo se decodifica y genera señales de control. Así, con  $N$  campos, se especifican  $N$  acciones simultáneas. Cada acción se traduce en la activación de una o más señales de control. Generalmente, aunque no siempre, querremos diseñar el formato de modo que cada señal de control no pueda ser activada por más de un campo. Claramente, no obstante, debe ser posible activar cada señal de control al menos por un campo.

Consideremos ahora un campo individual. Un campo que conste de  $L$  bits puede contener uno de los  $2^L$  códigos posibles, cada uno de los cuales puede codificar un patrón diferente de señales de control. Como solo puede aparecer un código en un campo en un momento dado, los códigos son mutuamente exclusivos y, por lo tanto, las acciones que ellos producen también lo son.

El diseño de un formato de microinstrucción codificado puede plantearse ahora en términos muy simples:

- Organizar el formato en campos independientes. Es decir, cada campo representa un conjunto de acciones (patrón de señales de control) de tal forma que pueden ocurrir simultáneamente acciones de diferentes campos.

- Definir cada campo de modo que las acciones alternativas que puede especificar ese campo sean mutuamente exclusivas. Es decir, que solo una de las acciones especificadas por un determinado campo pueda ocurrir en un momento dado.

Se pueden usar dos planteamientos para la organización de la microinstrucción codificada en campos: funcional y por recursos. El método de *codificación funcional* identifica funciones dentro de la máquina y designa los campos según el tipo de función. Por ejemplo, si se pueden utilizar varias fuentes para la transferencia de datos al acumulador, se puede utilizar un campo para este propósito, especificando cada código una fuente diferente. La *codificación por recursos* ve a la máquina como un conjunto de recursos independientes y le dedica un campo a cada uno de ellos (por ejemplo, E/S, memoria, ALU).

Otro aspecto de la codificación es si esta es directa o indirecta (Figura 17.11b). En la codificación indirecta, se utiliza un campo para determinar la interpretación de otro campo. Por ejemplo, consideremos una ALU capaz de realizar ocho operaciones aritméticas y ocho operaciones de desplazamiento diferentes. Se podría usar un campo de un bit para indicar si se trata de una operación aritmética o de desplazamiento, y un campo de tres bits podría indicar la operación. Generalmente esta técnica implica dos niveles de decodificación, incrementando el retardo de propagación de las señales.

La Figura 17.12 es un ejemplo sencillo de estos conceptos. Se supone un procesador con un único acumulador y varios registros internos, tales como un contador de programa y un registro temporal para la entrada de la ALU. La Figura 17.12a muestra un formato muy vertical. Los tres primeros bits indican el tipo de operación, los tres siguientes codifican la operación, y los dos últimos seleccionan un registro interno. La Figura 17.12b es una solución más horizontal, aunque todavía usa cierta codificación. En este caso, las funciones diferentes aparecen en campos diferentes.

## EJECUCIÓN DE MICROINSTRUCCIONES EN EL LSI-11

El LSI-11 [SEBE76] es un buen ejemplo del planteamiento de microinstrucciones verticales. Veamos en primer lugar la organización de la unidad de control y después el formato de la microinstrucción.

**Organización de la unidad de control del LSI-11.** El LSI-11 es el primer miembro de la familia PDP-11 que se ofreció como procesador en una sola tarjeta. La tarjeta contiene tres circuitos LSI, un bus interno conocido como *bus de microinstrucciones* (*microinstruction bus*, MIB) y alguna lógica de interfaz adicional.

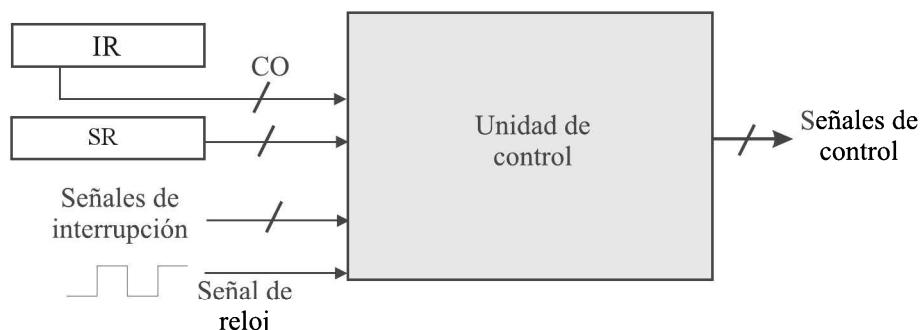
La Figura 17.13 representa, de forma simplificada, la organización del procesador LSI-11. Los tres circuitos son el de datos, el de control y el de memoria de control. El circuito de datos contiene una ALU de ocho bits, 26 registros de ocho bits y almacenamiento para varios códigos de condición. Dieciséis de esos registros se usan para implementar los ocho registros de uso general de 16 bits del PDP-11. Otros incluyen una palabra de estado del programa, un registro de dirección de memoria (MAR), y un registro intermedio de memoria. Como la ALU trata solo ocho bits a la vez, se requieren dos pasos a través de ella para implementar una operación aritmética de 16 bits del PDP-11. Esto lo controla el micropograma.

El circuito o circuitos de memoria de control contienen la memoria de control, de 22 bits de ancho. El circuito de control contiene la lógica de secuenciamiento y ejecución de las microinstrucciones. Contiene también el registro de dirección de control, el registro de datos de control y una copia del registro de instrucción máquina.

de estado podrán ser atendidas, mientras que las que se encuentren en nivel igual o inferior quedarán inhabilitadas. Así, cuando llega una interrupción de un determinado nivel, la unidad de control fija la máscara de interrupción a ese nivel, impidiendo de esta forma atender a otras interrupciones de menor prioridad mientras se está atendiendo a la interrupción actual. Cuando termina el tratamiento de la interrupción se vuelve a fijar el nivel de interrupción permitido para poder atender a otras interrupciones de menor prioridad. Con este mecanismo se impide que interrupciones de menor prioridad puedan interrumpir la ejecución de la rutina de tratamiento de aquellas con mayor prioridad, y solo aquellas de mayor prioridad podrán interrumpir a ejecución de las de menor prioridad.

## 5.6. Diseño de la unidad de control

Como se ha indicado en secciones anteriores, la función de la unidad de control es ejecutar instrucciones máquina, para lo cual lleva a cabo de forma sincronizada operaciones elementales que se realizan mediante la activación de las señales de control correspondientes. Como se puede ver en la Figura 5.2 y 5.11, la unidad de control toma como entradas el código de operación (CO) que reside en el registro de instrucción, el contenido del registro de estado, información sobre las señales de interrupción y la señal de temporización del reloj. A partir de esta información se debe encargar de generar las señales de control necesarias para la correcta ejecución de cada instrucción.



**Figura 5.11:** Entradas y salidas de la unidad de control

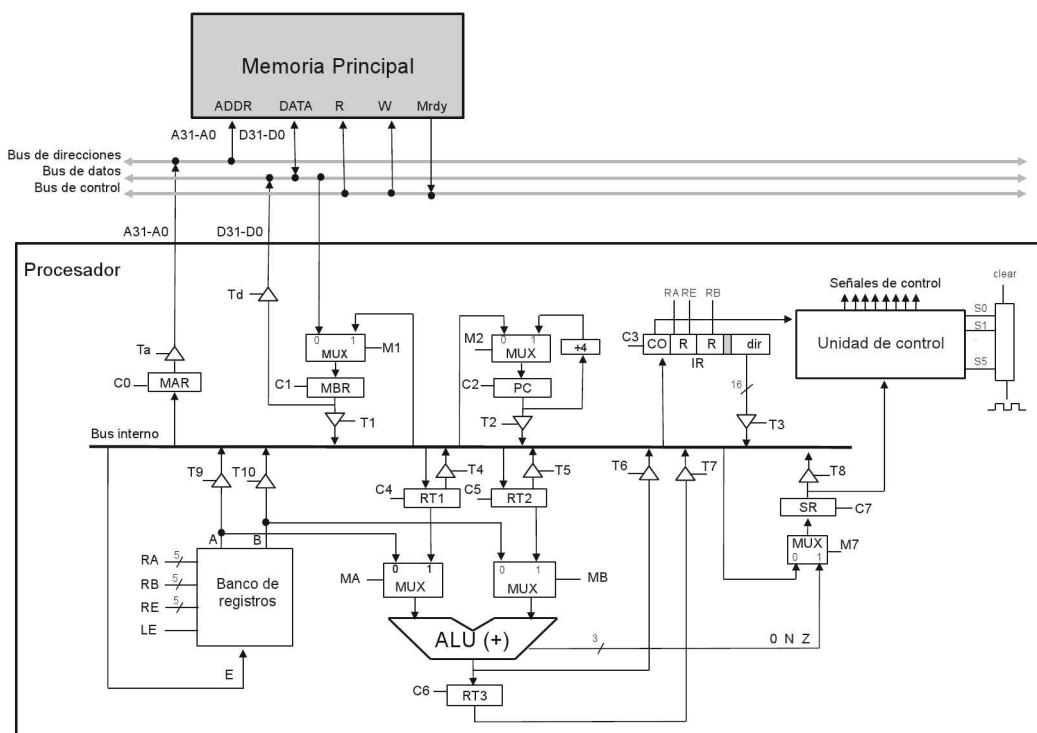
El paso previo para el diseño de la unidad de control es obtener la máquina de estados que define el funcionamiento de la unidad de control, donde cada estado se corresponde con un ciclo de reloj. A modo de ejemplo, considere un computador que dispone de cuatro instrucciones máquina:

- add Rd, Rf, que realiza la suma  $Rd = Rd + Rf$ . Código de operación:  $I_1 I_0 = 00$ .
- lw Rd, dirección, que carga en el registro  $Rd$  el contenido de la posición de memoria *dirección*. Código de operación:  $I_1 I_0 = 01$ .
- sw Rf, dirección que almacena en una posición de memoria el contenido del registro  $Rf$ . Código de operación:  $I_1 I_0 = 10$ .
- bz R, dirección, que bifurca a una dirección de memoria dada si el contenido del registro  $R$  es 0. Código de operación:  $I_1 I_0 = 11$ .

Se va a ilustrar el diseño de una unidad de control para un computador con una estructura como la mostrada en la Figura 5.12 (se ha modificado ligeramente la estructura descrita al inicio del capítulo por simplicidad)

y para estas cuatro instrucciones. En esta estructura se ha considerado que la instrucción almacenada en el registro de instrucción tiene un formato fijo y que las señales *RA*, *RB* y *RE* se obtienen directamente del registro de instrucción, tal y como se muestra en la figura. La ALU de este procesador realiza solo la operación de suma, y se considera que el registro 0 del banco de registros, es un registro de solo lectura que tiene cableado su valor a 0.

La máquina de estados que describe el funcionamiento de la unidad de control es la que se muestra en la Figura 5.13. Cada estado de esta máquina se corresponde con un ciclo de reloj, durante el cual la unidad de control debe mantener activadas las señales de control necesarias para ejecutar la operación elemental correspondiente. Así, por ejemplo, en el estado 3, la unidad de control debe activar la señal *T1* y la señal del carga en registro de instrucción *C3*.

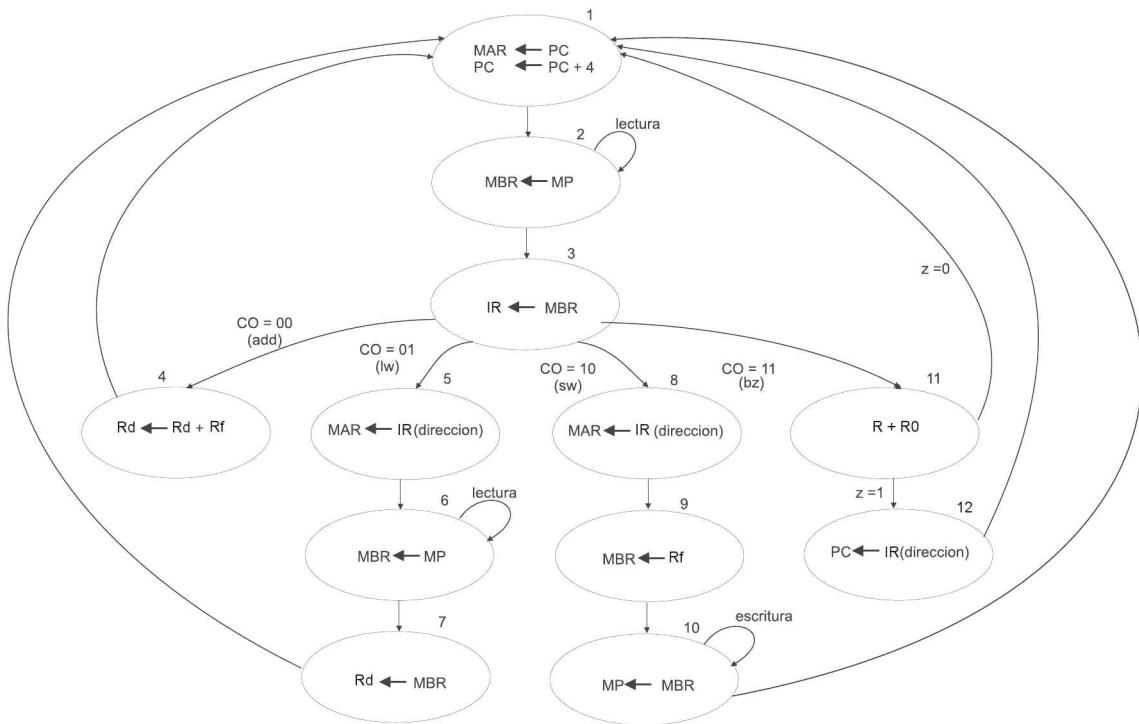


**Figura 5.12:** Estructura del computador que ejecuta las instrucciones *add*, *lw*, *sw* y *bz*

A partir de la máquina de estados que describe el funcionamiento del computador, existen dos formas de abordar el diseño de la unidad de control: unidad de control cableada y unidad de control micropogramada.

### 5.6.1. Unidad de control cableada

La unidad de control cableada se construye mediante puertas lógicas utilizando alguno de los métodos clásicos de diseño de circuitos combinacionales y secuenciales. La Figura 5.14 muestra una posible implementación de la unidad de control para el procesador de la Figura 5.12, que ejecuta las cuatro instrucciones máquina *add*, *lw*, *sw* y *bz*, descritas anteriormente.



**Figura 5.13:** Máquina de estados correspondiente al procesador de la Figura 5.12, que ejecuta las instrucciones *add*, *lw*, *sw* y *bz*

Para obtener este diseño, tal y como se puede observar en la Figura 5.12, se ha incluido un circuito contador que genera los valores  $0, 1, \dots, 5$  en las señales  $S_0, S_1, \dots, S_5$ , de forma que en cada pulso de reloj se incrementa el valor, de acuerdo a la siguiente tabla:

| valor | <i>S0</i> | <i>S1</i> | <i>S2</i> | <i>S3</i> | <i>S4</i> | <i>S5</i> |
|-------|-----------|-----------|-----------|-----------|-----------|-----------|
| 0     | 1         | 0         | 0         | 0         | 0         | 0         |
| 1     | 0         | 1         | 0         | 0         | 0         | 0         |
| 2     | 0         | 0         | 1         | 0         | 0         | 0         |
| 3     | 0         | 0         | 0         | 1         | 0         | 0         |
| 4     | 0         | 0         | 0         | 0         | 1         | 0         |
| 5     | 0         | 0         | 0         | 0         | 0         | 1         |

El registro contador dispone de una señal *clear* que permite volver al valor 0, es decir, aquel en el que  $S_0$  es 1 y el resto 0. A partir de este registro contador y del valor del código de operación de la instrucción se pueden definir cada uno de los 12 estados de la máquina presentada en la Figura 5.13. Así, el estado 1 de la máquina mostrada en la Figura 5.13 se corresponde con estado  $S_0$  del registro contador. El estado 4 se corresponde con el estado  $S_3$  cuando el código de operación de la instrucción a ejecutar es  $I_1 I_0 = 00$  (*add*). El estado 9 de la máquina de estados se ha asociado al estado  $S_4$  cuando el código de operación es  $I_1 I_0 = 10$  (*sw*). A continuación se muestran las señales de control que se activan para cada uno de los 12 estados de acuerdo a los valores de las señales  $S_0, S_1, \dots, S_5$ , del código de operación de la instrucción a ejecutar ( $I_1 I_0$ ) y del valor del biestable  $Z$ .

Señales de control a activar durante el ciclo de *fetch*:

Estado 1 ( $S_0$ ):  $C_0, T_2, M_2, C_2$

Estado 2 ( $S_1$ ):  $T_a, R, C_1$

Estado 3 ( $S_2$ ):  $T_1, C_3$

Señales de control a activar para la instrucción *add* (código de operación  $I_1 I_0 = 00$ ):

Estado 4 ( $S_3 \wedge I_1 I_0 = 00$ ):  $T_6, LE, C_7, M_7, clear$

Señales de control a activar para la instrucción *lw* (código de operación  $I_1 I_0 = 01$ ):

Estado 5 ( $S_3 \wedge I_1 I_0 = 01$ ):  $T_3, C_0$

Estado 6 ( $S_4 \wedge I_1 I_0 = 01$ ):  $T_a, R, C_1$

Estado 7 ( $S_5 \wedge I_1 I_0 = 01$ ):  $T_1, LE, clear$

Señales de control a activar para la instrucción *sw* (código de operación  $I_1 I_0 = 10$ ):

Estado 8 ( $S_3 \wedge I_1 I_0 = 10$ ):  $T_3, C_0$

Estado 9 ( $S_4 \wedge I_1 I_0 = 10$ ):  $T_9, M_1, C_1$

Estado 10 ( $S_5 \wedge I_1 I_0 = 10$ ):  $T_a, T_d, W, clear$

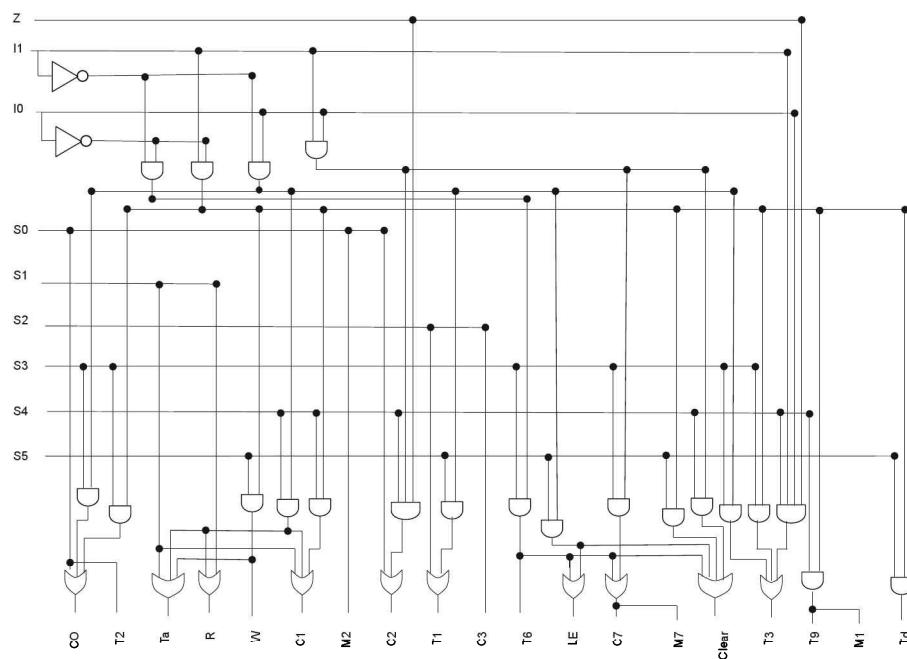
Señales de control a activar para la instrucción *bz* (código de operación  $I_1 I_0 = 11$ ):

Estado 11 ( $S_3 \wedge I_1 I_0 = 11$ ):  $M_7, C_7$

Estado 12 ( $S_4 \wedge I_1 I_0 = 11 \wedge Z == 1$ ):  $T_3, C_2, clear$

Estado 12 ( $S_4 \wedge I_1 I_0 = 11 \wedge Z == 0$ ):  $clear$

Para la ejecución de esta última instrucción se ha asumido que esta instrucción se codifica de la siguiente forma: el código de operación en los 2 primeros bits, el registro *R* en los 5 siguientes bits, el registro con



**Figura 5.14:** Ejemplo de circuito que implementa la unidad de control para las instrucciones *add*, *lw*, *sw* y *bz*

identificador 0 en los 5 siguientes y por último la dirección de salto en los últimos 16 bits de la instrucción. De esta forma, durante el estado 11 lo que se hace es sumar el contenido del registro  $R$  (la señal  $RA$  tendrá el valor que tenga el registro  $R$  en la instrucción) con el registro con identificador 0, que se encuentra en el tercer campo de la instrucción y hace que  $RB$  valga 0. Si el resultado de la operación es cero, cosa que solo puede ocurrir cuando  $R$  es 0, se actualizará el biestable  $Z$  en el registro de estado. Así, durante el ciclo 12 se consulta el valor del biestable  $Z$ , si está activado ( $R$  es cero) se modifica el contador de programa con la dirección almacenada en los últimos 16 bits de la instrucción que está en  $IR$ . En caso contrario, simplemente se activa la señal *clear* para volver al estado 1. Observe que en todos los estados finales de cada instrucción se activa la señal *clear* para volver de nuevo al estado 1 de la máquina de estados finito, que se corresponde con el primer estado del ciclo de *fetch*.

Se ha considerado que la memoria funciona de forma síncrona y necesita un ciclo para las operaciones de lectura y escritura. En caso de disponer de una memoria asíncrona, los estados 2, 6 y 10 de la Figura 5.13 deberían mantenerse activos mientras la memoria no hubiera completado la operación.

A partir de las señales de control que han de activarse en cada uno de los estados se pueden obtener las expresiones lógicas asociadas a cada una de las señales de control y a partir de ellas obtener un posible circuito como el que se muestra en la Figura 5.14. Estas expresiones lógicas se presentan a continuación.

$$\begin{aligned}
 C0 &= S0 + S3 \cdot \overline{I_1} \cdot I_0 + S3 \cdot I_1 \cdot \overline{I_0} \\
 T2 &= S0 \\
 Ta &= S1 + S4 \cdot \overline{I_1} \cdot I_0 + S5 \cdot I_1 \cdot \overline{I_0} \\
 R &= S1 + S4 \cdot \overline{I_1} \cdot I_0 \\
 W &= S5 \cdot I_1 \cdot \overline{I_0} \\
 C1 &= S1 + S4 \cdot \overline{I_1} \cdot I_0 + S4 \cdot I_1 \cdot \overline{I_0} \\
 M2 &= S0 \\
 C2 &= S0 + S4 \cdot I_1 \cdot I_0 \cdot Z \\
 T1 &= S2 + S5 \cdot \overline{I_1} \cdot I_0 \\
 C3 &= S2 \\
 T6 &= S3 \cdot \overline{I_1} \cdot \overline{I_0} \\
 LE &= S3 \cdot \overline{I_1} \cdot \overline{I_0} + S5 \cdot \overline{I_1} \cdot I_0 \\
 C7 &= S3 \cdot \overline{I_1} \cdot \overline{I_0} + S3 \cdot I_1 \cdot I_0 \\
 M7 &= S3 \cdot \overline{I_1} \cdot \overline{I_0} + S3 \cdot I_1 \cdot I_0 \\
 clear &= S3 \cdot \overline{I_1} \cdot \overline{I_0} + S5 \cdot \overline{I_1} \cdot I_0 + S5 \cdot I_1 \cdot \overline{I_0} + S4 \cdot I_1 \cdot I_0 \\
 T3 &= S3 \cdot \overline{I_1} \cdot I_0 + S3 \cdot I_1 \cdot \overline{I_0} + S4 \cdot I_1 \cdot I_0 \cdot Z \\
 T9 &= S4 \cdot I_1 \cdot \overline{I_0} \\
 M1 &= S4 \cdot I_1 \cdot \overline{I_0} \\
 Td &= S5 \cdot I_1 \cdot \overline{I_0}
 \end{aligned}$$

### 5.6.2. Unidad de control microprogramada

En una unidad de control microprogramada se utiliza una memoria de control para almacenar el estado de las diferentes señales de control durante cada uno de los ciclos de cada instrucción. En este tipo de unidades de control, se denomina **microinstrucción** al conjunto de bits que identifican las señales de control que debe generar la unidad de control en cada ciclo de reloj. Un **microprograma** constituye la secuencia ordenada de microinstrucciones que debe generar la unidad de control para ejecutar cada instrucción máquina. Finalmente, se denomina **firmware** o **microcódigo** al conjunto de todas los microprogramas de un computador.

Para el ejemplo que se está utilizando en esta sección y la máquina de estados de la Figura 5.13 se muestra en la Figura 5.15 el formato de la microinstrucción así como el valor que tienen todas las microinstrucciones

correspondientes a los microprogramas asociados al *fetch* y a las instrucciones *add*, *lw*, *sw* y *bz*. Como se puede observar se ha considerado al ciclo de *fetch* como un microprograma más a considerar.

La Figura 5.16 muestra la estructura de una unidad de control microprogramada. Una unidad de control microprogramada debe incluir una memoria de control donde almacenar todas las microinstrucciones correspondientes al juego de instrucciones del computador y un secuenciador encargado de ir generando las direcciones de la memoria de control. En cada ciclo de reloj, el secuenciador debe generar la dirección de la memoria de control donde se encuentra almacenada la microinstrucción a ejecutar en ese ciclo de reloj. Esta microinstrucción contiene los valores de las señales de control que deben estar activos durante ese ciclo de reloj. Existen dos formas de realizar el secuenciamiento de la memoria de control: secuenciamiento explícito y secuenciamiento implícito.

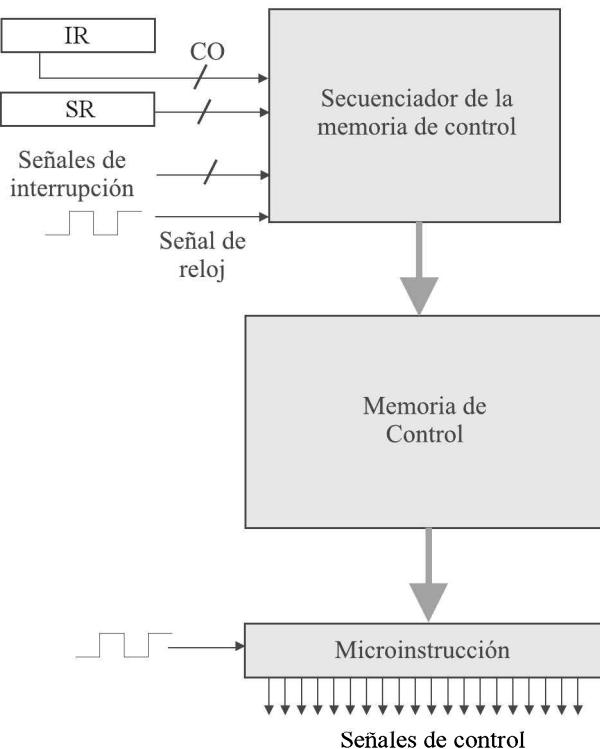
En una memoria de control con **secuenciamiento explícito** cada microinstrucción incluye la dirección de la microinstrucción siguiente. En una unidad de control con **secuenciamiento implícito** la microinstrucción no incluye la dirección de la siguiente microinstrucción a ejecutar. La siguiente microinstrucción a ejecutar, siempre que sea del mismo microprograma, es la siguiente, es decir, la que ocupa la posición de memoria consecutiva. En este tipo de unidades de control, la memoria de control debe almacenar de forma consecutiva todas las microinstrucciones asociadas a un mismo microprograma.

A continuación se va a presentar un posible esquema de unidad de control microprogramada con secuenciamiento implícito para el procesador de la Figura 5.12, que ejecuta las instrucciones *add*, *lw*, *sw* y *bz*. Para este ejemplo se necesitan 12 microinstrucciones ta y como se muestra la Figura 5.15. Como se puede apreciar en la Figura 5.12, la unidad de control debe poder elegir como siguiente microdirección:

- La que se encuentra en la dirección 0, es decir, donde se almacena la primera microinstrucción correspondiente al *fetch*.
- La siguiente, que se obtiene a partir del sumador que se muestra en la Figura 5.17.
- La microdirección donde se almacena la primera microinstrucción correspondiente a una instrucción máquina. Se obtiene a partir de la memoria ROM que se observa en la Figura 5.17. Esta memoria se direcciona con el código de operación y en cada dirección almacena la microdirección de la primera microinstrucción asociada a un microprograma. Así, en el esquema mostrado como ejemplo en la Figura 5.17, la posición 10 de la memoria ROM almacena la microdirección de la primera microinstrucción de la instrucción con código de operación 10, es decir, la instrucción *sw*.

|                | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | LE | MA | MB0 | MB1 | M1 | M2 | M7 | R | W | Ta | Td |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|-----|-----|----|----|----|---|---|----|----|
| <i>μfetch0</i> | 1  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 1  | 0 | 0 | 0  | 0  |
| <i>μfetch1</i> | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 1 | 0  | 1  |
| <i>μfetch2</i> | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 0 | 0  | 0  |
| <i>μadd0</i>   | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0   | 1  | 0  | 0   | 0   | 0  | 0  | 0  | 1 | 0 | 0  | 0  |
| <i>μlw0</i>    | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 0 | 0  | 0  |
| <i>μw1</i>     | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 1  | 0 | 1 | 0  | 0  |
| <i>μlw2</i>    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 1  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 0 | 0  | 0  |
| <i>μsw0</i>    | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 0 | 0  | 0  |
| <i>μsw1</i>    | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0   | 0  | 1  | 0   | 0   | 0  | 0  | 0  | 0 | 0 | 0  | 0  |
| <i>μsw3</i>    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 1 | 1  | 1  |
| <i>μbz0</i>    | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 1  | 0 | 0 | 0  | 0  |
| <i>μbz1</i>    | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 1  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0   | 0  | 0  | 0   | 0   | 0  | 0  | 0  | 0 | 0 | 0  | 0  |

**Figura 5.15:** Formato de microinstrucción y microprogramas para el procesador de la Figura 5.12 que ejecuta las instrucciones *add*, *lw*, *sw* y *bz*

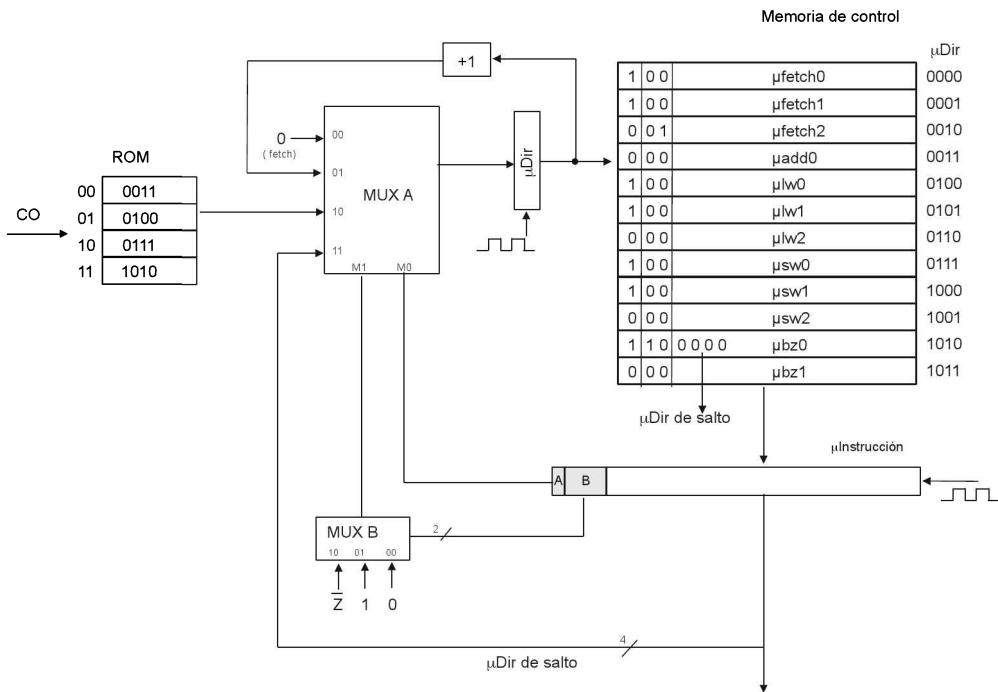


**Figura 5.16:** Esquema de una unidad de control micropogramada

- Otra microdirección que se almacena en la propia microinstrucción, que se utiliza en las instrucciones de salto condicional. Para el ejemplo mostrado se va a asumir que solo existe un único bit de estado asociado a la ALU. En este caso el bit  $Z$ . En caso de haber más, lo único que habría que hacer es ampliar el multiplexor B utilizado como comparador.

Como se observa en la Figura 5.17, la última microinstrucción asociada al ciclo de *fetch* obtiene para  $A$  y  $B$  los valores 0 y 01, que seleccionan en el multiplexor A la entrada 10 a partir de la cual se obtiene la microdirección de la primera microinstrucción asociada a la instrucción leída de memoria durante este ciclo. Al final, por ejemplo, del microprograma asociado a la instrucción *sw*,  $A$  y  $B$  toman los valores 0 y 00 respectivamente, que permiten seleccionar en el multiplexor A la entrada 00 de la que se obtiene como siguiente microdirección la 0, que es donde comienza el microprograma asociado al ciclo de *fetch*. Para la instrucción de salto condicional, la microinstrucción almacenada en la dirección 1010 asigna a  $A$  y  $B$  los valores 1 y 10 respectivamente. Con estos valores se selecciona en el multiplexor A la entrada  $\bar{Z}$ , donde  $Z$  puede valer 0 y 1. En caso de que el valor de  $Z$  sea 1, se selecciona la entrada 01 del multiplexor<sup>4</sup>, y la siguiente microinstrucción a ejecutar será la que se encuentra a continuación, es decir, en la dirección 1011. En esta microinstrucción se deberá modificar el contenido del contador de programa con la dirección de salto incluida en la instrucción *bz*. En caso de que  $Z$  sea 0, no se ha de modificar el contador de programa. En este caso se selecciona como entrada del multiplexor A, el valor 11, con lo que la siguiente microinstrucción a ejecutar es aquella que se encuentra en la microdirección de salto incluida en la propia microinstrucción. En este caso la microdirección 0000, que se corresponde con el inicio del microprograma asociado al *fetch*. Para codificar las microdirecciones de salto, se pueden utilizar dentro de la propia microinstrucción algunos bits que no vayan a ser utilizados durante la misma.

<sup>4</sup>Nótese que la entrada 10 del multiplexor B es  $\bar{Z}$ .



**Figura 5.17:** Ejemplo de unidad de control microprogramada con secuenciamiento implícito para el procesador de la Figura 5.12, que ejecuta las instrucciones *add*, *lw*, *sw* y *bz*

## 5.7. Arranque del computador

Cuando se arranca el computador o se pulsa el botón *reset* se genera una señal eléctrica que carga unos valores predefinidos en los registros del computador. Uno de los valores importantes es el que se carga en el contador de programa. Este valor corresponde a la primera instrucción máquina que ejecutará el computador. Esta instrucción corresponde a un programa denominado *iniciador* que se almacena en la memoria ROM del computador. El iniciador ROM es un programa que se ejecuta en modo núcleo con las interrupciones inhabilitadas. Este programa se encarga de realizar las siguientes funciones:

- Realiza una comprobación general del sistema para detectar las características del hardware instalado y comprobar el funcionamiento del computador.
- Una vez finalizada la comprobación inicial se pasa a leer y almacenar en memoria el programa cargador del sistema operativo también denominado *boot*. Éste es un programa independiente del hardware y por tanto debe existir un convenio entre el computador y el sistema operativo sobre la ubicación y el tamaño del programa cargador. Normalmente este programa ocupa un sector (512 bytes) y se incluye en el primer sector (*sector de arranque*) del dispositivo utilizado para arrancar el sistema operativo (disco duro, unidad de disco, unidad de CD, etc.).

- Se almacena en el PC la dirección del programa cargador del sistema operativo. De esta forma se cede el control a este programa.

Una vez cedido el control al programa cargador del sistema operativo, este programa se encargará de realizar la carga en memoria del sistema operativo y cederle el control posterior al mismo.

## 5.8. Tiempo de ejecución de un programa

El tiempo de ejecución de un programa es uno de los principales elementos que suele utilizarse para analizar las prestaciones de un computador. Este tiempo viene dado por la siguiente expresión:

$$T_{ejecucion} = N \cdot CPI \cdot t_{ciclo\_UCP} + N \cdot AMI \cdot t_{ciclo\_mem} \quad (5.1)$$

donde:

- $N$  es el número de instrucciones máquina del programa. Este número depende del juego de instrucciones del computador y del compilador utilizado.
- $CPI$  es el número medio de ciclos de reloj que la unidad de control necesita para ejecutar una instrucción. Este parámetro depende del juego de instrucciones y de la estructura y organización interna del procesador.
- $t_{ciclo\_UCP}$  es el tiempo que dura el ciclo de reloj del procesador. La duración del ciclo depende la tecnología empleada para la construcción del procesador y de la estructura del mismo.
- $AMI$  es el número medio de accesos a memoria por instrucción. Este valor depende del juego de instrucciones del procesador.
- $t_{ciclo\_mem}$  es el número medio de ciclos necesarios para realizar un acceso a memoria. Este número depende de la tecnología de memoria empleada.

Analizando la expresión anterior, se observa que para reducir el tiempo de ejecución de un programa es necesario reducir tanto el tiempo debido al procesador como el tiempo debido a la memoria. El tiempo del procesador se puede reducir con mejoras en la tecnología, que permiten reducir la duración del ciclo de reloj de la CPU, mejoras en la organización del procesador y mejoras en la tecnología de compiladores. El tiempo de acceso a memoria se puede mejorar con memorias más rápidas y organizaciones que emplean diferentes niveles de memoria caché.

## 5.9. Problemas resueltos

---

**Problema 5.1** Dado el procesador de la figura 5.2, indique las operaciones elementales y señales de control necesarias para ejecutar la instrucción  $sw\ Rd,\ n(Rf)$ , que al igual que en el MIPS almacena en el registro  $Rd$  el contenido de la dirección de memoria que se obtiene de sumar  $n$  al contenido de  $Rf$ .

#### 4.1. INTRODUCCIÓN. CONTROL CABLEADO VERSUS MICROPROGRAMADO

Como se ha visto en el capítulo 3, la función de una sección de control es generar las señales necesarias para el funcionamiento del sistema en el estado de control actual y para determinar cuál es el siguiente estado de control. Estas operaciones las realiza en función del código de operación (C.O.) y el modo de direccionamiento (MD) de la instrucción que hay en el registro de instrucción (RI), cargada previamente desde la memoria principal.

Normalmente, la ejecución de una instrucción simple causa más de un cambio de estado en el sistema computador, enviándose, para cada uno de ellos, un grupo de señales de control a las unidades del sistema. Cada grupo de señales causa la ejecución de unas operaciones básicas específicas (denominadas *microoperaciones*), tal como la transferencia entre dos registros, el desplazamiento de los contenidos de un registro, o la selección de la función a realizar por la ALU. El siguiente grupo de señales de control puede o no depender de los resultados de la presente microoperación, o del estado de ciertos flags del sistema<sup>6</sup>. De esta manera, la interpretación y ejecución de una (macro)instrucción da lugar a una secuencia de operaciones máquina básicas (microoperaciones), cada una controlada por un grupo específico de señales de control (o microinstrucción).

La tarea principal de la unidad de control será secuenciar las microoperaciones, estableciendo en cada ciclo el estado de control actual (y siguiente) y generar las microinstrucciones precisas para la ejecución de la macroinstrucción. Por consiguiente, la sección de control debe contener la lógica necesaria para el almacenamiento de la información que identifica al estado de control actual y la lógica de decisión para la generación del identificador del estado de control siguiente. El identificador de estado actual puede estar conectado directamente a los puntos de control de la sección de procesamiento, o, más habitualmente, ser transformado mediante una lógica de decodificación en una plantilla que activa las señales de control.

La unidad de control cableada, cuyo diagrama básico de bloques se muestra en la figura 4.1, está compuesta por un conjunto de dispositivos de almacenamiento MSI (registros), para almacenar el estado de control actual y un conjunto de bloques combinacionales MSI (decodificadores) y SSI (puertas discretas), para generar el identificador de estado siguiente y las señales de

---

<sup>6</sup>En otras palabras, la siguiente microoperación será determinada en función de la microoperación actual y en algunos casos del estado de ciertos flags.

control. Típicamente, el diseño de este dispositivo se hace mediante la aplicación de heurísticos, de forma que, en principio, no existe una metodología precisa y concreta para su implementación. Una sección de control de este tipo suele ser (salvo para sistemas muy simples) poco flexible y estructurada, de forma que una modificación de su función requiere normalmente el rediseño parcial o total de la unidad.

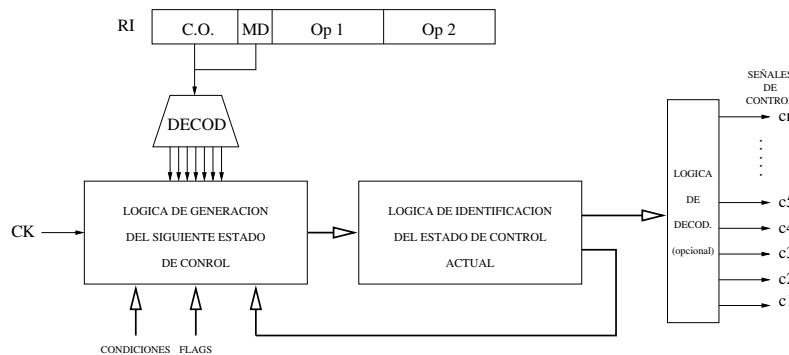


Figura 4.1: Unidad de control cableada

Frente a esta alternativa clásica, la unidad de control microprogramada, cuyo diagrama de bloques se muestra en la figura 4.2, utiliza un bloque de memoria LSI (ROM, PROM, PLA), llamado memoria microprograma.

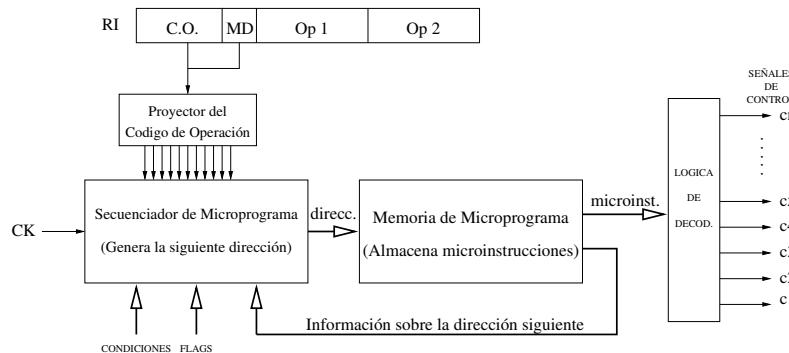


Figura 4.2: Unidad de control microprogramada

Esta memoria almacena la información del estado de control actual, que estará identificado por la dirección seleccionada en dicha memoria. El contenido de esta posición será la microinstrucción a ejecutar y suministra la informa-

ción necesaria para generar las señales de control y para que el secuenciador de microprograma determine la dirección de la siguiente microinstrucción a ejecutar.

El concepto de microprogramación fue propuesto en 1951 por Maurice Wilkes. Aunque conceptualmente la unidad de control microprogramada era mucho más flexible y estructurada que la cableada, la microprogramación no se empezó a usar comercialmente hasta 1964, con la serie System/360 de IBM. La razón es que los tiempos de acceso en las memorias ROM disponibles hasta entonces eran tan altos que la pérdida de prestaciones en los sistemas microprogramados era inaceptable para usos prácticos.

A partir de finales de los 60 y principios de los 70, el desarrollo de memorias de semiconductor rápidas y baratas permitió la expansión de la microprogramación a los minicomputadores, generalizándose su uso. Adicionalmente, el uso de memorias RAM para contener microprogramas posibilitó el diseño de sistemas microprogramables dinámicamente, en los que el programa de control puede ser cambiado durante el funcionamiento simplemente cambiando los contenidos de la memoria microprograma.

En la actualidad, el uso del control microprogramado es generalizado en los microprocesadores de juego de instrucción complejo (CISC). Sin embargo, en el diseño de procesadores de juego de instrucción reducido (RISC), en los que el control es relativamente simple, se prefiere el uso de unidades de control cableadas, con el fin de optimizar al máximo las prestaciones del hardware y reducir en lo posible el tiempo medio de ejecución de una instrucción (es decir, bajar el CPI – *ciclos de reloj por instrucción*–).

Puede resultar confuso y quizás poco didáctico intentar definir desde el primer momento los conceptos relativos a la microprogramación. Por ello, intentaremos hacer una primera aproximación a estos conceptos mediante un ejemplo para posteriormente adentrarnos más en detalle en temas avanzados. De acuerdo con esta filosofía, comenzaremos describiendo el flujo de datos de un hipotético procesador simple. A continuación, ilustraremos la técnica de control microprogramada mediante el desarrollo paso a paso de la sección de control microprogramada que controle a la sección de procesamiento inicial. Finalmente, se introducirá un conjunto de conceptos más complejos, relacionados con la temporización, optimización del diseño, y generación de microsubrutinas asociadas a instrucciones máquina.

#### 4.2. DISEÑO DE UNA UNIDAD DE CONTROL MICROPROGRAMADA

Se desea diseñar una unidad de control microprogramada para la sección de procesamiento elemental de la figura 4.3. Esta sección está compuesta por registros de 8 bits (DR, AC, IR, PC, MAR), una ALU con operandos y resultado de 8 bits, multiplexores y buses de 8 bits para interconectar estos elementos.

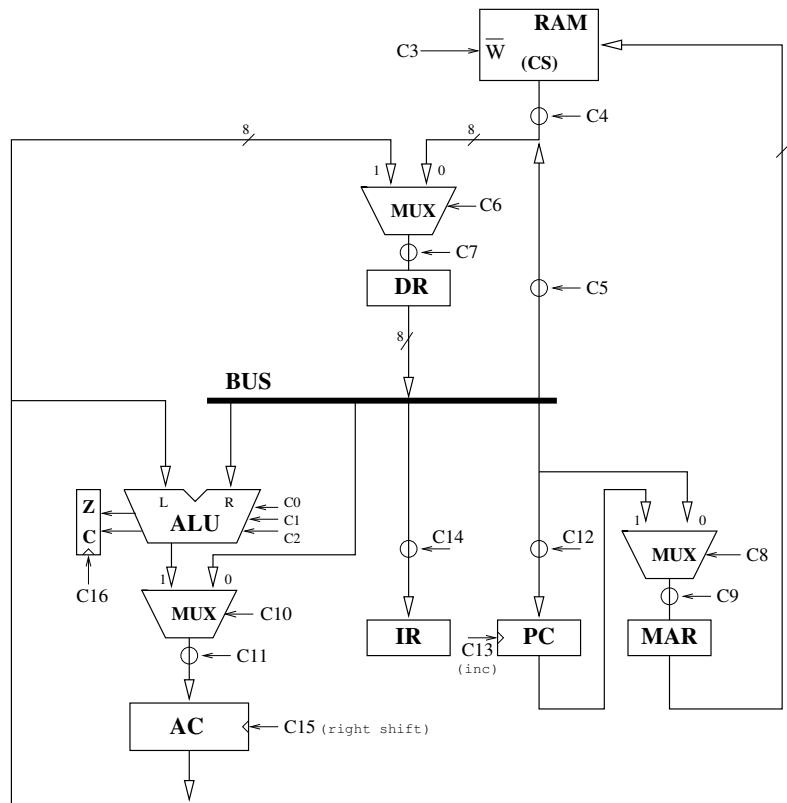


Figura 4.3: Una hipotética sección de procesamiento elemental

Es importante aclarar que aunque existe un bloque de memoria RAM en la figura, éste no pertenece a la sección de procesamiento (sino a la unidad de memoria del computador). En dicha memoria principal se almacenarán las instrucciones y los datos que procesará la sección de procesamiento.

Para simplificar el proceso de diseño, las entradas de control de los diversos componentes han sido abstraídas por comutadores conceptuales, numerados

del 0 al 16 (c0-c16), y situados en las puertas de entrada y salida de los registros, estando cada conmutador controlado por una única línea de control. Un conmutador a la entrada de un registro la habilita cuando su línea de control pasa de 0 a 1 (es decir, es activo en flanco de subida) y permanece activado sólo un breve intervalo de tiempo, suficiente para que el dato sea almacenado en el registro. Un conmutador a la salida de un componente se activa cuando su línea de control pasa a valer 1 y permanece activado hasta que la línea vuelve a cero (es decir, es activo por nivel).

Aquellos registros que posean una salida conectada a un bus compartido poseerán capacidad tri-estado en dicha salida; si por el contrario el bus le está enteramente dedicado, la salida no poseerá esta capacidad.

El direccionamiento y acceso a memoria se realiza a través de los registros MAR y DR. El registro MAR posee una salida sin capacidad tri-estado, a través de la cual el contenido del registro MAR se usará como dirección a acceder en la memoria. Por su parte, la entrada del registro DR está controlada por un multiplexor: si la línea de control c6 está a 0, la entrada al registro DR será el contenido del bus de datos de la memoria; si, por el contrario, la línea c6 está a 1, el registro DR recibirá la información contenida en el registro AC. El registro MAR es denominado **registro de dirección de memoria**, mientras el registro DR se denomina **registro de dato de memoria**.

El punto de control c4 habilita el acceso a memoria. En lectura ( $c_3=1$ ), deja el dato en el bus de datos mientras c4 esta a uno (nivel). En escritura ( $c_3=0$ ), es en el flanco de bajada de c4 cuando se escribe el dato en la posición de memoria contenida en el registro MAR.

En la tabla 4.1 se sumariza la función de cada uno de los conmutadores del diseño.

Los diseños como los de la figura 4.3 son más un producto de la intuición y la experiencia que de la aplicación de un conjunto preestablecido de principios de diseño. Esto es, no existe una metodología exacta que nos diga que, para un conjunto de instrucciones dados y unos objetivos de rendimiento establecidos, el diseño de la sección de procesamiento es la elección óptima.

El siguiente paso del diseño consiste en conseguir que esta sección de procesamiento haga algo útil. Para ello, crearemos una sección de control microprogramada que, para cada instrucción del conjunto de instrucciones del procesador que estamos construyendo, genere una secuencia de conjuntos de señales que, ordenadamente, vayan activando los puntos de control pertinentes.

Comenzaremos por introducir un **registro de control** (o registro de microinstrucción –**MicroRI**–), de 17 bits de anchura, en el que cada bit esté co-

| C   | Significado                                      | C   | Significado                |
|-----|--|-----|----------------------------|
| c0  | Selección ADD en ALU                             | c1  | Selección AND en ALU       |
| c2  | Selección COMP de la entrada L de la ALU (C1)    | c3  | Read( $R$ ) / Write( $W$ ) |
| c4  | Habilitación memoria                             | c5  | MEM $\leftarrow$ DR        |
| c6  | 0:(DR $\leftarrow$ MEM), 1:(DR $\leftarrow$ AC)  | c7  | Escritura en DR            |
| c8  | 0:(MAR $\leftarrow$ DR), 1:(MAR $\leftarrow$ PC) | c9  | Escritura en MAR           |
| c10 | 0:(AC $\leftarrow$ DR), 1:(AC $\leftarrow$ ALU)  | c11 | Escritura en AC            |
| c12 | PC $\leftarrow$ DR                               | c13 | INC PC                     |
| c14 | IR $\leftarrow$ DR                               | c15 | Desplaza a la derecha AC   |
| c16 | Carga registro de estado (C,Z)                   |     |                            |

Tabla 4.1: Funciones de los commutadores de control

nectado a una única línea de control, tal como se muestra en la figura 4.4.

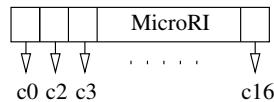


Figura 4.4: Registro de control

El problema ahora es el siguiente: colocar un valor en el registro de control, ¿desde dónde? Una segunda consideración a tener en cuenta es que el poner un único valor no será suficiente; realmente necesitaremos un conjunto de valores que varíe en el tiempo, de manera que una secuencia preestablecida de valores de control vaya apareciendo en el registro de control en intervalos periódicos.

Para conseguir esto, ya imaginamos las dos alternativas básicas:

- Diseño de un dispositivo secuencial (máquina de estados) formado por puertas lógicas, biestables, contadores, etc..., que dilucide en cada momento (en función del estado en el que se encuentre) qué valor debe almacenarse en el registro de control y a qué próximo estado debe cambiar. Esta solución es la que ya habíamos identificado en la sección anterior con el **control cableado**.
- Ya que tenemos que definir una secuencia de valores de 17 bits a almacenar en el MicroRI, ¿por qué no almacenamos estos valores en otra memoria (distinta de la memoria principal de la figura 4.3) y cada vez que necesitemos una palabra de control (microinstrucción) la leemos de dicha memoria? Esta solución conduce al diseño de una sección de **con-**

trol microprogramada como la de la figura 4.5.

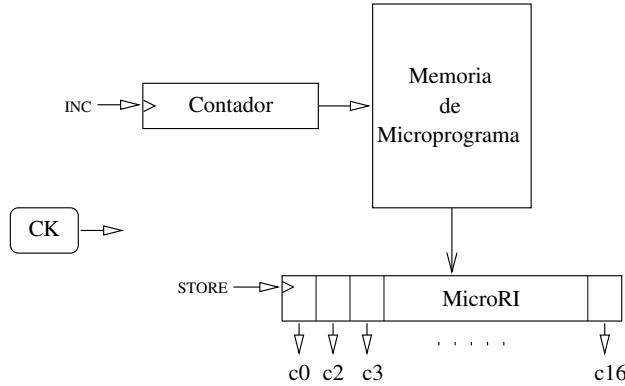


Figura 4.5: Sección de control microprogramado inicial

En la figura 4.6 vemos como el registro MicroRI se conecta a una **Memoria de Micropograma** (también llamada memoria de control, micromemoria o  $\mu$ Mem.), direccionada por medio de un contador. El registro MicroRI tiene una señal *Store* (activa por flanco de subida). Por su parte, el contador tiene una entrada INC, activa en una transición de 0 a 1 (flanco de subida), que le hará incrementar en uno su valor actual.

Evidentemente, será también necesario un reloj, CK, que permita realizar la sincronización necesaria para ir leyendo periódicamente las microinstrucciones de la Memoria de Micropograma y almacenándolas en el MicroRI. Inicialmente se considerará un esquema de control muy simple, en el cual la señal de reloj se conectaría directamente a la entrada *Store* del registro MicroRI y la misma señal invertida atacaría el punto de control del registro Contador, como vemos en la figura 4.6. Por tanto, en cada flanco de subida del ciclo de reloj, el contenido de la memoria apuntado por el registro Contador se cargaría en MicroRI y en el flanco de bajada se incrementaría el Contador.

Llegados a este punto daremos unas cuantas definiciones:

- A este periodo de reloj se le denominará **tiempo del ciclo máquina** y será el intervalo entre conjuntos sucesivos de señales de control suministradas a la sección de procesamiento.
- Cada palabra de 17 bits de la memoria de control recibe el nombre de **microinstrucción**. Por consiguiente, una microinstrucción es un conjunto de señales de control o especificaciones que controla el flujo de datos y las funciones del dispositivo en la sección de procesamiento durante un

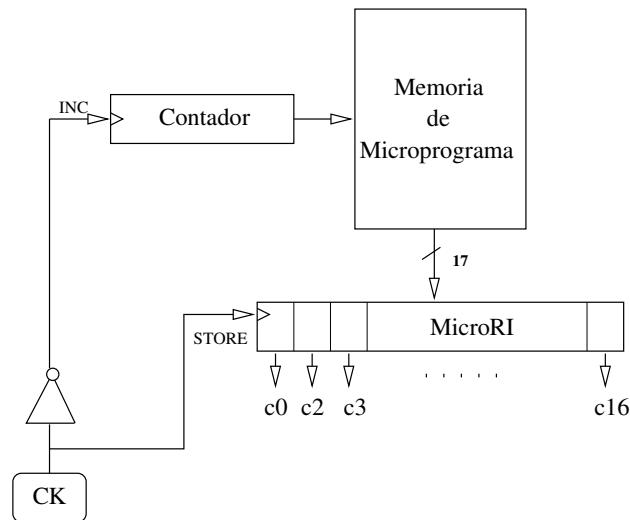


Figura 4.6: Sección de control microprogramado más detallada

ciclo máquina.

- La colección de microinstrucciones almacenadas en la memoria se denomina **micropograma**. La memoria que contiene el micropograma se denomina **memoria de micropograma**, mientras el contador que la direcciona actúa como un secuenciador de micropograma muy primitivo que también llamaremos **contador de micropograma** o **MicroPC**.

#### 4.2.1. Características temporales

Un primer examen de la unidad de control diseñada, rápidamente revela que el sistema no funciona, debido a que no se ha tenido en cuenta la temporización en el uso de los recursos del procesador.

Un problema que se plantea es el hecho de que no hay ningún mecanismo que garantice que las señales de control asociadas a interruptores activos por flanco pasen a 0 tras haber tomado el valor 1 y, por tanto, que garantice la aparición del flanco de activación. Por ejemplo, si dos microinstrucciones sucesivas son del tipo:

```
XXXXXX XXXX1 XXXXXX XX
XXXXXX XXXX1 XXXXXX XX
```

el 1 de la segunda microinstrucción no tendrá ningún efecto. Esto es debido

a que ese punto de control,  $c_9$ , se activa por flanco de subida, de forma que fuerza la carga del registro MAR cuando pasa de 0 a 1. Por tanto, la primera microinstrucción si tendrá el efecto esperado<sup>7</sup>, pero la segunda no, ya que en principio no existe ninguna transición baja-alta.

Es evidente, pues, que es necesario secuenciar la generación de diferentes conjuntos de señales de control dentro del ciclo máquina. Una solución simple que adoptaremos para el ejemplo que estamos desarrollando, en el que disponemos de una única señal de reloj, es emplear el flanco de bajada de ésta para temporizar las señales activas por flanco, de manera que si un bit del MicroRI asociado a un punto de control por flanco esté activo, se produzca una transición baja-alta con el flanco de bajada del reloj. Los cronogramas mostrados en 4.7 muestran la situación que deseamos implementar.

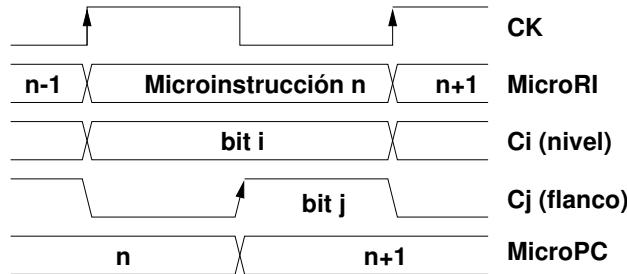


Figura 4.7: Formas de onda de las señales en el tiempo

Escribiendo ciertas ecuaciones lógicas para los puntos de control del sistema se puede diseñar un circuito que permita crear la secuencia descrita anteriormente. Ello se consigue fácilmente realizando una operación AND de los bits que corresponden a señales por flanco en MicroRI con  $\overline{CK}$  tal como se esquematiza en la figura 4.8.

#### 4.2.2. Ejemplos de microprograma

Una vez solucionada la temporización del ciclo máquina, ya es posible ejecutar en el procesador un pequeño microprograma. Veamos algunos ejemplos que nos servirán más adelante en la microprogramación del conjunto de instrucciones del procesador.

- $AC \leftarrow Mem(DR)$  Carga de AC con el contenido de la posición apuntada por DR. Para ello tenemos que llevar la dirección al registro MAR,

<sup>7</sup>siempre que la microinstrucción anterior tenga  $c_9=0$

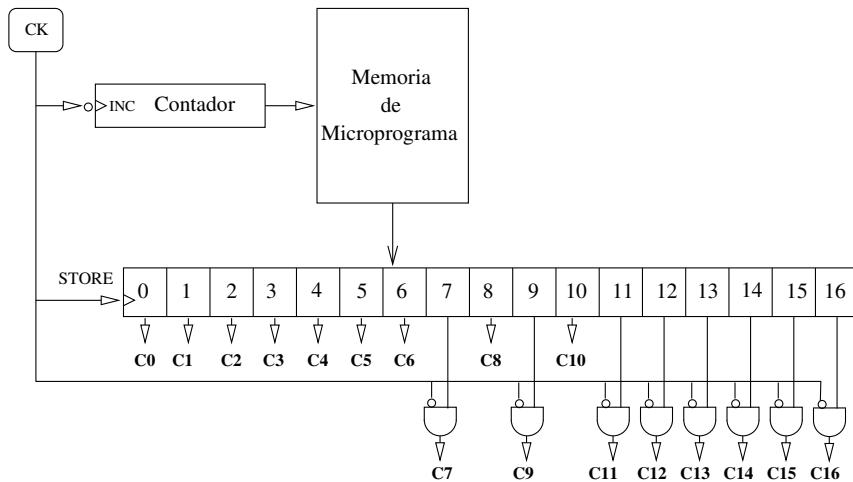


Figura 4.8: Esquema parcial de las conexiones de temporización

leer desde memoria en DR y de ahí al registro AC.

|                   | c0    c4    c8    c12    c16              |
|-------------------|---|
| 1. MAR <- DR      | 0000 0000 0100 0000 0 (activos: c9)       |
| 2. DR <- Mem(MAR) | 0001 1001 0000 0000 0 (activos: c3,c4,c7) |
| 3. AC <- DR       | 0000 0000 0001 0000 0 (activos: c11)      |

- $\boxed{Mem(DR) \leftarrow AC}$  Cargar en la dirección de memoria apuntada por DR el contenido del acumulador AC. Como en el caso anterior la dirección destino se guardará en MAR, el acumulador se guarda en DR y DR se escribe en memoria.

|                   | c0    c4    c8    c12    c16           |
|-------------------|--|
| 1. MAR <- DR      | 0000 0000 0100 0000 0 (activos: c9)    |
| 2. DR <- AC       | 0000 0011 0000 0000 0 (activos: c6,c7) |
| 3. Mem(MAR) <- DR | 0000 1100 0000 0000 0 (activos: c4,c5) |

- $\boxed{DR \leftarrow DR/2}$  Dividir el contenido de DR por dos. Para ello trasladamos el contenido de DR a AC, desplazamos a la derecha AC y el contenido de AC se volcará en DR.

|                    | c0    c4    c8    c12    c16           |
|--------------------|--|
| 1. AC <- DR        | 0000 0000 0001 0000 0 (activos: c11)   |
| 2. Shift_Right(AC) | 0000 0000 0000 0001 0 (activos: c15)   |
| 3. DR <- AC        | 0000 0011 0000 0000 0 (activos: c6,c7) |

### 4.2.3. Capacidad de salto

Otro importante defecto del diseño, relacionado con el secuenciamiento de la unidad de control, es la falta de capacidad de ramificación en el microprograma, que obliga a pasar secuencialmente por todas y cada una de las microinstrucciones contenidas en la memoria de control. Un segundo problema de la unidad de control, es la ausencia de un mecanismo que permita tomar decisiones en el microprograma, es decir, dar saltos condicionados al valor de una o más variables de estado de la sección de procesamiento.

Ambos problemas expresan la necesidad de incluir algún procedimiento que permita la modificación, condicional o incondicional, del contenido del contador de microprograma. En una aproximación simple, suponiendo que la memoria de control tiene 64 palabras, se ampliará la microinstrucción con dos campos más. Uno de estos campos, al que se llamará **dirección de salto**, tiene 6 bits, y su función es contener la dirección de una palabra de la memoria de control. El otro campo, que llamaremos de **Condición de Salto** (o **CS**) tiene dos bits, con el siguiente significado:

- 00 Ejecución secuencial** normal. El contador de secuencia se incrementa en 1 y la próxima microinstrucción a ejecutar es la inmediatamente siguiente a la actual.
- 11 Salto incondicional.** La dirección de la siguiente microinstrucción a ejecutar está contenida en el campo dirección de salto de la microinstrucción en ejecución. El contador se cargará con el valor contenido en los 6 bits de este campo.
- 01 Salto condicional** asociado al acarreo de salida **C** de la ALU. Si este acarreo de salida es 1, la microinstrucción siguiente viene definida por el campo de dirección de salto como en el salto incondicional. En caso contrario, la ejecución será secuencial (normal).
- 10 Salto condicional** asociado a la salida **Z** de la ALU. Si el resultado a la salida de la ALU es 0 (línea Z a 1), la microinstrucción siguiente viene definida por el campo de dirección de salto de la actual microinstrucción como en el salto incondicional. En caso contrario, la ejecución será secuencial (normal).

Para implementar esta capacidad de salto (condicional o incondicional) de la sección de control podemos utilizar el hardware de la figura 4.9. Los bits 17 y 18 de la microinstrucción forman la microfunción de control de salto, mientras los bits 19 a 24 constituyen el campo de la dirección de salto.

Vemos como hemos ampliado la funcionalidad del Contador de microprograma (MicroPC a partir de ahora), de forma que además de ser incrementado,

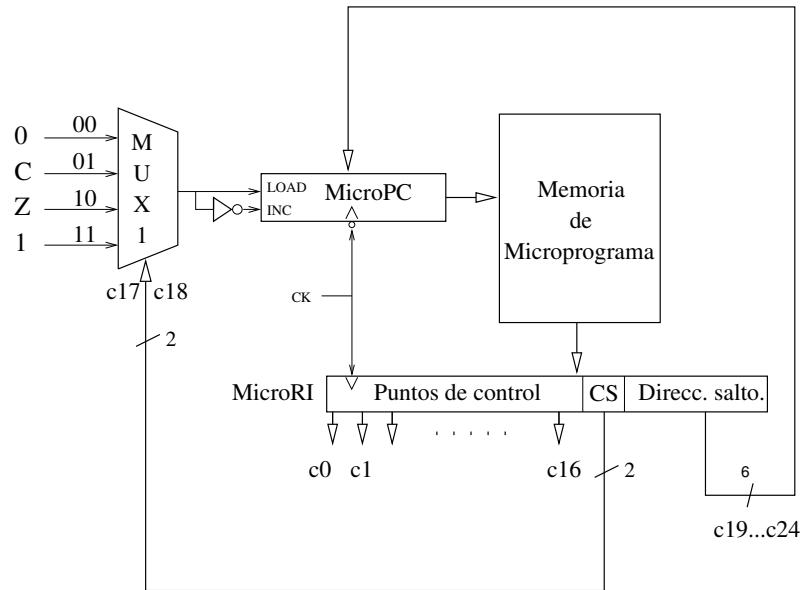


Figura 4.9: Sección de control con capacidad de salto

también puede ser cargado con un nuevo valor que llega desde el campo *Dirección* del MicroRI. El incremento o la carga del MicroPC estará determinado por las entradas de control *INC* o *LOAD* respectivamente, siendo estas entradas activas a nivel alto. El incremento del MicroPC está asociado a la ejecución secuencial, mientras que la carga implicará un *salto* a nivel de micropograma. Los dos bits del campo *Condición de Salto*, **CS**, seleccionan una de las entradas del **MUX1** de forma que:

**CS=00** La entrada 00, alimentada con un 0 lógico, provocará el incremento del MicroPC. Es decir, ejecución secuencial.

**CS=11** La entrada 11, alimentada con un 1 lógico, provocará la carga del MicroPC. Es decir, salto incondicional.

**CS=01** La entrada 01, alimentada con el valor del flag C determinará la carga o el incremento del MicroPC según C=1 o C=0, respectivamente. Es decir: salto condicional al flag C.

**CS=10** La entrada 10, alimentada con el valor del flag Z determinará la carga o el incremento del MicroPC según Z=1 o Z=0, respectivamente. Es decir: salto condicional al flag Z.

Este simple mecanismo de salto permite incrementar notablemente las po-

sibilidades del sistema. A cambio, se incrementa sustancialmente el tamaño de la microinstrucción. En este sentido, el campo de dirección de salto resulta especialmente costoso, pues sus 6 bits suponen un incremento de un 35 % en la longitud de la palabra de control, y sólo se emplea en unas pocas microinstrucciones.

Como ejemplo microprogramemos la operación JZ DR: saltar a la dirección apuntada por DR cuando Z=1, en caso contrario incrementar PC, esto es:

*Si Z=1 PC ← DR si no PC ← PC + 1*

| micropograma    |                        |      |      |      |     |     |     |     |
|-----------------|------------------------|------|------|------|-----|-----|-----|-----|
|                 | c0                     | c4   | c8   | c12  | c16 | c17 | c19 | c24 |
| 1.Si Z=1 goto 4 | 0000                   | 0000 | 0000 | 0000 | 0   | 10  | 000 | 100 |
| 2.PC <- PC+1    | 0000                   | 0000 | 0000 | 0100 | 0   | 00  | 000 | 000 |
| 3.goto 5        | 0000                   | 0000 | 0000 | 0000 | 0   | 11  | 000 | 101 |
| 4.PC <- DR      | 0000                   | 0000 | 0000 | 1000 | 0   | 00  | 000 | 000 |
| 5.siguiente     | .....                  |      |      |      |     |     |     |     |
|                 | microinstrucción ..... |      |      |      |     |     |     |     |

En este ejemplo las operaciones de las microinstrucciones 2 y 3 se podían haber realizado en una sola microinstrucción pero se ha hecho así por claridad.

#### 4.2.4. Microprogramación del conjunto de instrucciones

Una vez diseñados el *datapath* y el secuenciador de control del procesador elemental, se podría proceder a la microprogramación de las diversas (macro)instrucciones de que se quiera dotar a su juego de instrucciones. Es decir, tenemos que completar el diseño teniendo en cuenta que al fin y al cabo el procesador que hemos diseñado tiene que ser capaz de ejecutar cualquier programa que tenga en su memoria principal. Por tanto debe ejecutar cualquier instrucción del conjunto de instrucciones de este procesador, a través de las etapas de *búsqueda de instrucción, decodificación, cálculo de la dirección efectiva, búsqueda de operandos y ejecución* como se explicó en el tema anterior. Más concretamente, la ejecución de cada instrucción se llevará a cabo mediante la ejecución de una **microsubrutina** (o **microrrutina**) asociada a cada instrucción máquina.

Se supone que antes de ejecutar el programa, el sistema operativo, habrá colocado el programa en memoria y habrá inicializado el contador de programa, PC, con la dirección de la primera instrucción máquina a ejecutar. A partir de ahí, la sección de control será la encargada de ir leyendo las instrucciones que sean apuntadas por el PC, cargándolas en el IR y ejecutándolas, hasta que el programa termine y se devuelva el control al S.O.

En cuanto al registro IR, vemos que no es más que un registro que se de-

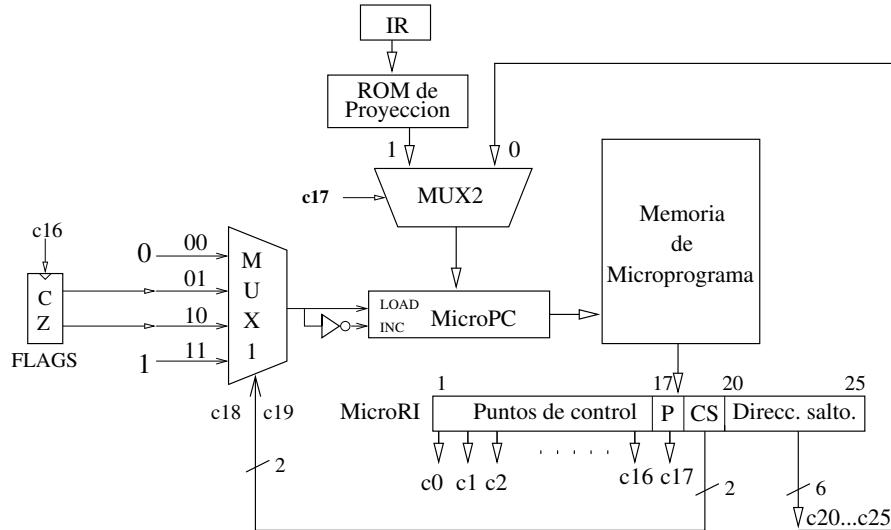


Figura 4.10: Sección de control completa

be cargar desde el BUS MEM, con la instrucción que ha seleccionado el PC en memoria. El código de operación de la instrucción cargada en el IR debe inicializar el MicroPC de forma que apunte a la primera microinstrucción de la microsubrutina asociada a dicho C.O. Para encontrar la dirección de una microrrutina a partir de su C.O. asociado, utilizaremos una tabla implementada mediante una ROM, (**ROM de Proyección**). Esta ROM contendrá en la dirección indicada por el C.O. la dirección de la microrrutina asociada a dicha macroinstrucción. Por ejemplo, si el código de operación de la instrucción ADD es 25 y tiene asociada una microrrutina que está en la dirección 97 de la Memoria de Micropograma, la posición 25 de la ROM de proyección debe contener un 97.

Dado que el MicroPC es cargado algunas veces con la dirección que viene de la ROM de Proyección y otras veces con la que viene del campo Dirección del MicroRI, utilizaremos el multiplexor MUX2 para seleccionar cuál de las direcciones será la que actualice el MicroPC. Ello implica añadir un nuevo punto de control que denominaremos **bit de proyección**. Por cuestiones de significado, añadimos este bit inmediatamente antes del campo de condición de salto, de manera que será c17, pasando c18, c19 a ser los puntos de control de salto y c20 a c25 los bits de dirección de salto. En la figura 4.10 se muestra la unidad de control microprogramada completa. En esta figura no mostramos,

por simplificación, la lógica adicional necesaria de temporización, pero no hay que olvidar que las microordenes son secuenciadas ordenadamente, tal y como se explicó en la sección 4.2.1.

Retomaremos ahora, una vez completado el diseño del procesador, el problema de escribir las microrrutinas asociadas a cada una de las instrucciones máquina. Cualquier microrutina debe contener las microinstrucciones necesarias para realizar las siguientes funciones:

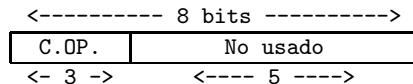
1. El contenido de la memoria apuntado por el PC debe ser cargado en el registro RI (**búsqueda de instrucción**). A continuación y a partir del C.O. recién cargado en el RI, se debe cargar la dirección inicial de la microrutina asociada en el registro MicroPC (**Decodificación**). Además, se incrementará el registro PC para que apunte a la siguiente palabra de memoria.
2. Es ejecutado el núcleo de la microrutina: **cálculo de dirección efectiva, búsqueda de operando(s) y ejecución**.
3. La última microinstrucción de la microrutina deberá saltar de nuevo al conjunto de microinstrucciones encargadas de la búsqueda y decodificación de instrucciones, que procederá a leer de la memoria principal la siguiente instrucción a ejecutar, volviendo así a comenzar el ciclo.

Dado que la fase de búsqueda y decodificación de instrucciones es común e independiente de la instrucción en particular que vaya a ser ejecutada, escribiremos las microinstrucciones encargadas de dicha misión una única vez en la memoria de microprograma, por ejemplo, a partir de la posición 0 de esta memoria. Llamaremos a ese conjunto de microinstrucciones, **microrutina de búsqueda y decodificación**, o, **microrutina de fetching**. Al resto de las fases (cálculo de dirección efectiva, búsqueda de operando y ejecución) lo llamaremos el **núcleo de la microrutina** asociada a una instrucción dada.

Por tanto, cuando se va a ejecutar un programa y el S.O. lo carga en memoria e inicializa el PC, apuntando a la primera instrucción, también se debe resetear el MicroPC al valor cero. Una vez completada la inicialización, se ejecuta la microrutina de *fetching* (recordemos que está almacenada a partir de la posición 0). Así, se carga la primera instrucción en el RI y en la decodificación, la ROM de Proyección proporciona la dirección del núcleo de la microrutina asociada al código de operación que se encuentre en RI. Esa dirección se cargará en el MicroPC, o dicho de otra forma, estaremos saltando a la posición de la Memoria de Microprograma que contiene dicho núcleo de la microrutina. Una vez completada ésta, la última microinstrucción debe contener un salto a la posición 0 para volver a buscar y ejecutar la siguiente

instrucción máquina.

Supondremos que en esta máquina hipotética, las instrucciones tiene un tamaño fijo de 8 bits, interpretándose de la siguiente manera: los 3 bits más significativos corresponden con el código de operación y los 5 menos significativos no se usan. Es decir:



Siguiendo las indicaciones marcadas en el apartado anterior, vamos a elaborar el microprograma que se almacenará en la micromemoria y que implementa el conjunto de instrucciones mostrado en la tabla 4.2.

| Mnemónico | Descripción  | C.OP. |
|-----------|--|-------|
| LOAD [X]  | $AC \leftarrow Mem(X)$                               | 000   |
| STORE [X] | $Mem(X) \leftarrow AC$                               | 001   |
| ADD [X]   | Suma: $AC \leftarrow AC + Mem(X)$                    | 010   |
| AND [X]   | AND lógico: $AC \leftarrow AC \wedge Mem(X)$         | 011   |
| JMP       | Salto incondicional: $PC \leftarrow AC$              | 100   |
| JZ        | Salta si cero: si $Z=1 \Rightarrow PC \leftarrow AC$ | 101   |
| NOT       | $AC \leftarrow C1(AC)$                               | 110   |
| SHR       | $AC \leftarrow 0, AC[7:1]$                           | 111   |

Tabla 4.2: Conjunto de instrucciones de la CPU

Vemos como en total tendremos 8 instrucciones distintas, donde las 4 primeras utilizan direccionamiento directo a memoria, siendo X una palabra de 8 bits que debe estar a continuación del C.O. en la memoria principal. Sin embargo las últimas cuatro instrucciones son de 0 direcciones y toman implícitamente el operando del registro AC.

El contenido de la micromemoria resultante para estas instrucciones se muestra en la tabla 4.3.

Sobre este microprograma cabe hacer los siguientes comentarios:

- La búsqueda de instrucciones se realiza en las microinstrucciones 0 a 3.
- Las instrucciones se hayan en: LOAD: líneas 4 a 8; STORE: líneas 9 a 13; ADD: líneas 14 a 18; AND: líneas 19 a 23; JMP: líneas 24 y 25; JZ: líneas 26 a 29; COMP: línea 30; SHIFT: línea 31.
- Cada instrucción acaba con la microinstrucción `goto 0`, que retorna a la búsqueda de instrucción.
- La decodificación se realiza en la línea 3, siendo ésta la única microinstrucción que activa el bit 17 (proyección).

| Operación                           | N  | Microprograma |      |      |      |      |    |    |         |
|-------------------------------------|----|---------------|------|------|------|------|----|----|---------|
|                                     | c0 | 1             | 5    | 9    | 13   | 17   | 18 | 20 | 25      |
| <b>----- búsquedas y dec. -----</b> |    |               |      |      |      |      |    |    |         |
| MAR <- PC                           | 0  | 0             | 0000 | 0001 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR)                      | 1  | 0             | 0011 | 0010 | 0000 | 0000 | 0  | 00 | 000 000 |
| IR <- DR                            | 2  | 0             | 0000 | 0000 | 0000 | 0100 | 0  | 00 | 000 000 |
| PC <- PC+1                          | 3  | 0             | 0000 | 0000 | 0000 | 1000 | 1  | 11 | 000 000 |
| <b>----- load X -----</b>           |    |               |      |      |      |      |    |    |         |
| MAR <- PC                           | 4  | 0             | 0000 | 0001 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR), PC++                | 5  | 0             | 0011 | 0010 | 0000 | 1000 | 0  | 00 | 000 000 |
| MAR <- DR                           | 6  | 0             | 0000 | 0000 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR)                      | 7  | 0             | 0011 | 0010 | 0000 | 0000 | 0  | 00 | 000 000 |
| AC <- DR; goto 0                    | 8  | 0             | 0000 | 0000 | 0010 | 0000 | 0  | 11 | 000 000 |
| <b>----- store X -----</b>          |    |               |      |      |      |      |    |    |         |
| MAR <- PC                           | 9  | 0             | 0000 | 0001 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR), PC++                | 10 | 0             | 0011 | 0010 | 0000 | 1000 | 0  | 00 | 000 000 |
| MAR <- DR                           | 11 | 0             | 0000 | 0000 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- AC                            | 12 | 0             | 0000 | 0110 | 0000 | 0000 | 0  | 00 | 000 000 |
| Mem(MAR) <- DR; goto 0              | 13 | 0             | 0001 | 1000 | 0000 | 0000 | 0  | 11 | 000 000 |
| <b>----- add X -----</b>            |    |               |      |      |      |      |    |    |         |
| MAR <- PC                           | 14 | 0             | 0000 | 0001 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR), PC++                | 15 | 0             | 0011 | 0010 | 0000 | 1000 | 0  | 00 | 000 000 |
| MAR <- DR                           | 16 | 0             | 0000 | 0000 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR)                      | 17 | 0             | 0011 | 0010 | 0000 | 0000 | 0  | 00 | 000 000 |
| AC <- AC+DR; goto 0                 | 18 | 1             | 0000 | 0000 | 0110 | 0001 | 0  | 11 | 000 000 |
| <b>----- and X -----</b>            |    |               |      |      |      |      |    |    |         |
| MAR <- PC                           | 19 | 0             | 0000 | 0001 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR), PC++                | 20 | 0             | 0011 | 0010 | 0000 | 1000 | 0  | 00 | 000 000 |
| MAR <- DR                           | 21 | 0             | 0000 | 0000 | 1000 | 0000 | 0  | 00 | 000 000 |
| DR <- Mem(MAR)                      | 22 | 0             | 0011 | 0010 | 0000 | 0000 | 0  | 00 | 000 000 |
| AC <- AC & DR; goto 0               | 23 | 0             | 1000 | 0000 | 0110 | 0001 | 0  | 11 | 000 000 |
| <b>----- jmp -----</b>              |    |               |      |      |      |      |    |    |         |
| DR <- AC                            | 24 | 0             | 0000 | 0110 | 0000 | 0000 | 0  | 00 | 000 000 |
| PC <- DR; goto 0                    | 25 | 0             | 0000 | 0000 | 0001 | 0000 | 0  | 11 | 000 000 |
| <b>----- jz -----</b>               |    |               |      |      |      |      |    |    |         |
| Si Z=1 goto 28                      | 26 | 0             | 0000 | 0000 | 0000 | 0000 | 0  | 10 | 011 100 |
| goto 0                              | 27 | 0             | 0000 | 0000 | 0000 | 0000 | 0  | 11 | 000 000 |
| DR <- AC                            | 28 | 0             | 0000 | 0110 | 0000 | 0000 | 0  | 00 | 000 000 |
| PC <- DR; goto 0                    | 29 | 0             | 0000 | 0000 | 0001 | 0000 | 0  | 11 | 000 000 |
| <b>----- not -----</b>              |    |               |      |      |      |      |    |    |         |
| AC <- NOT(AC) ; goto 0              | 30 | 0             | 0100 | 0000 | 0110 | 0001 | 0  | 11 | 000 000 |
| <b>----- shr -----</b>              |    |               |      |      |      |      |    |    |         |
| SH_RIGHT(AC); goto 0                | 31 | 0             | 0000 | 0000 | 0000 | 0010 | 0  | 11 | 000 000 |

Tabla 4.3: Contenido de la memoria de micropograma

Así pues, propuesto el contenido de la micromemoria, el contenido del sistema de proyección (la dirección está expresada en binario y la salida en decimal), que asocia a cada instrucción el comienzo de la microrutina que la ejecuta, viene dado por:

| C.O.P. | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|
| Línea  | 4   | 9   | 14  | 19  | 24  | 26  | 30  | 31  |

### 4.3. DISEÑO AVANZADO

En esta sección se propone un diseño avanzado para una sección de control asociada a la sección de datos que se muestra en la figura 4.11. Esta sección está compuesta por cuatro registros (I, A, B y D), un sumador binario completo, un desplazador y un multiplexor de dos entradas; adicionalmente, existen dos registros cuyo contenido es siempre cero. Todos estos componentes están conectados mediante un conjunto de buses. En particular el **BUS X** proporciona la entrada a los registros, el **BUS MEM.** es el bus de datos de la memoria RAM, el **BUS L** alimenta la entrada izquierda (*Left*) de la ALU, así como el **BUS R** hace lo propio con la derecha (*Right*). Aunque la anchura de los componentes y las vías de datos no es importante para el diseño de la unidad de control, supondremos una anchura de 16 bits. Las entradas de control de los dispositivos están summarizadas en la tabla 4.4.

| C   | Significado                 | C   | Significado                         |
|-----|-----------------------------|-----|-------------------------------------|
| c1  | I → BUS L del Sumador       | c2  | A → BUS L del Sumador               |
| c3  | B → BUS L del Sumador       | c4  | 0 → BUS L del Sumador               |
| c5  | B → BUS R del Sumador       | c6  | D → BUS R del Sumador               |
| c7  | 0 → BUS R del Sumador       | c8  | Sumador → Desplazador               |
| c9  | BUS X → I                   | c10 | BUS X → A                           |
| c11 | BUS X → B                   | c12 | D-Mpx → D                           |
| c13 | D → BUS MEM                 | c14 | 0 → pasa BUS X; 1 → pasa BUS MEM    |
| c15 | Carry de entrada al Sumador | c16 | 1 → Desplazamiento; 0 → No desplaza |
| c17 | 1 → Derecha; 0 → Izquierda  | c18 | Habilitación de Memoria             |
| c19 | 1 → Lectura; 0 → Escritura  |     |                                     |

Tabla 4.4: Funciones de los commutadores de control

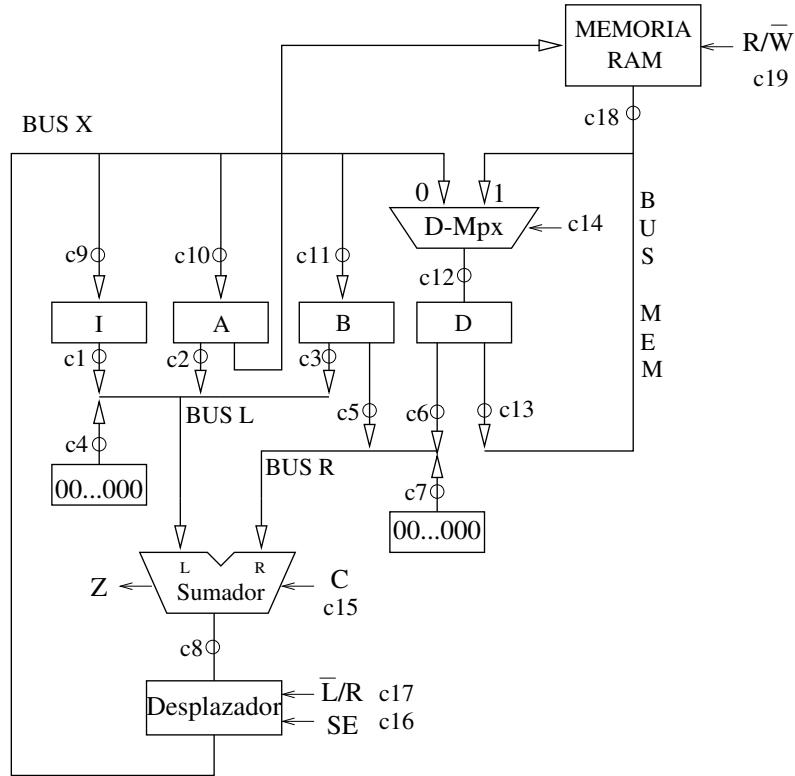


Figura 4.11: Sección de procesamiento elemental

#### 4.3.1. Características temporales

En el diseño de la unidad de control propuesta en la sección 4.2 se temporizó las señales de control mediante un único reloj. Ello impone una gran limitación en cuanto a la ejecución de una microinstrucción, ya que el número de registros que puede “atravesar” un dato por cada periodo de reloj es uno, es decir, en cada periodo de reloj un dato puede ser transferido de un registro a otro nada más. Sin embargo, nuestro objetivo puede ser más ambicioso y podemos desear que en un periodo de reloj, esto es, en una sola microinstrucción, se pueda realizar la transferencia de un dato por un camino que implique más de un registro.

Podemos, para ello, dividir el ciclo máquina en una secuencia de subciclos, en los que se carguen una serie de registros, con lo que en un sólo ciclo podamos trasladar un dato, si quisieramos, desde la memoria hasta la salida de la ALU.

Esta división podría ser:

- Fase 1.** Habilitación de la memoria de microprograma y lectura de la microinstrucción actual.
- Fase 2.** Carga del MicroRI con la microinstrucción actual.
- Fase 3.** Habilitación de la salida del MicroRI. Deshabilitación de la memoria de microprograma.
- Fase 4.** Si la microinstrucción actual lo requiere, habilitación de los controles de salida c1 a c7 y c13 y de la selección de memoria principal, c18.
- Fase 5.** Si la microinstrucción actual lo requiere, habilitación del commutador c8 para pasar la salida del sumador al desplazador. Generación de una transición 0-1 en la entrada INC del contador de microprograma.
- Fase 6.** Deshabilitación de los commutadores c1 a c7, c13 y c18. Si la microinstrucción lo requiere, habilitación del commutador c16 para realizar un desplazamiento.
- Fase 7.** Si la microinstrucción actual lo requiere, habilitación de los commutadores c9 a c12 para almacenar la salida del desplazador a través del bus X.

Las señales de control c14, c15, c17 y c19 son señales asíncronas (activas por nivel), pues no inician ninguna acción. En consecuencia, no requieren un control temporal, bastando con garantizar que tomen el valor deseado desde el momento en que se habilita la salida del registro MicroRI.

Una posible implementación de esta secuencia podría ser la generación de 7 pulsos de reloj dentro de cada periodo de 200 ns, para distinguir cada fase, pero eso implica subir la frecuencia del reloj. Sin embargo, es posible simplificar el circuito de reloj necesario usando una única señal de reloj para crear cuatro pulsos temporales (denominados T1 a T4), también de periodo 200 ns, pero desfasados del reloj principal en tal forma que se solapen.

Ésto se puede conseguir mediante una cadena de elementos de retardo (que pueden consistir simplemente en dos inversores en secuencia). Dado que la fase de acceso a la memoria de control es, presumiblemente, la más lenta, se tomará un retardo de 50 ns entre T1 y T2 y 23 ns de retardo entre T2 y T3 y entre T3 y T4. En la figura 4.12 se muestran las formas de onda resultantes. El solapamiento de las señales permite discriminar un considerable número de intervalos temporales, definidos simplemente mediante el producto (AND) y suma (OR) lógicos de las señales T1 a T4.

Las ecuaciones lógicas para los puntos de control del sistema deben implicar la secuencia de siete pasos, como se describirá más adelante. Así, según se vió en dicho esquema de fases, las acciones iniciales deben ser habilitar la

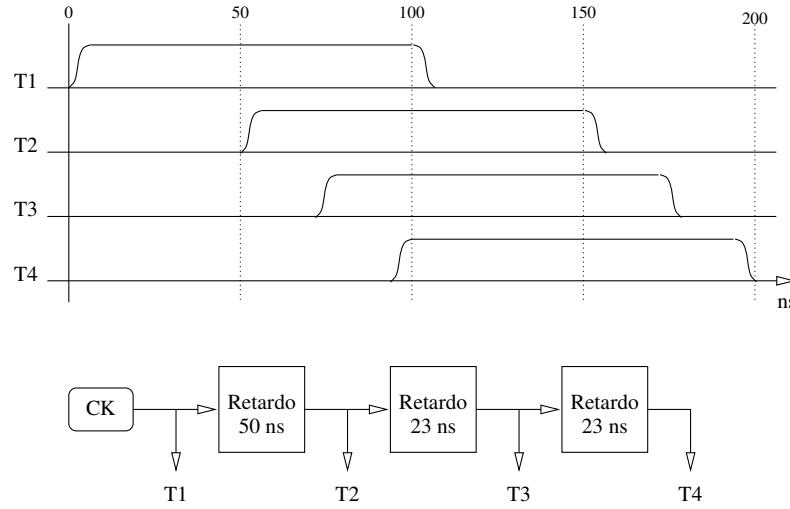


Figura 4.12: Formas de onda de las señales de tiempo

memoria de microprograma, cargar el MicroRI con la salida de esta memoria de microprograma, tras un tiempo igual al tiempo de acceso a dicha memoria y, por último, habilitar la salida del MicroRI y deshabilitar la lectura de la memoria de microprograma. La gestión de la memoria de microprograma puede hacerse simplemente conectando la señal  $T1 \text{ AND } \overline{T2}$  como señal de control *Enable* de la memoria de microprograma.

Por su parte, si se supone que el tiempo de acceso de la memoria de microprograma es de 40 ns, la señal de habilitación de escritura en el MicroRI, *Store*, deberá conectarse a T2 y, por tanto, el registro MicroRI se cargará en el flanco de subida de T2. Por último, puesto que el MicroRI debe proporcionar las señales de control a la unidad de procesamiento desde ese instante hasta el final del ciclo máquina, se conectarán la habilitación de lectura del MicroRI, *Enable*, a la señal T2 OR T4. En la figura 4.13 se representa parte del esquema de conexionado que permite el secuenciamiento descrito.

El control de los commutadores c1 a c7, c13 y c18, necesario en las fases 4 y 6, se logra simplemente realizando el AND de las señales de control correspondientes con la señal T3. El commutador c8, que debe abrirse en un instante posterior, se controla mediante la señal resultante del AND de la línea de control c8 y de la señal T4.

Por su parte, la operación de desplazamiento debe retardarse hasta que el registro de desplazamiento haya recibido la información procedente del su-

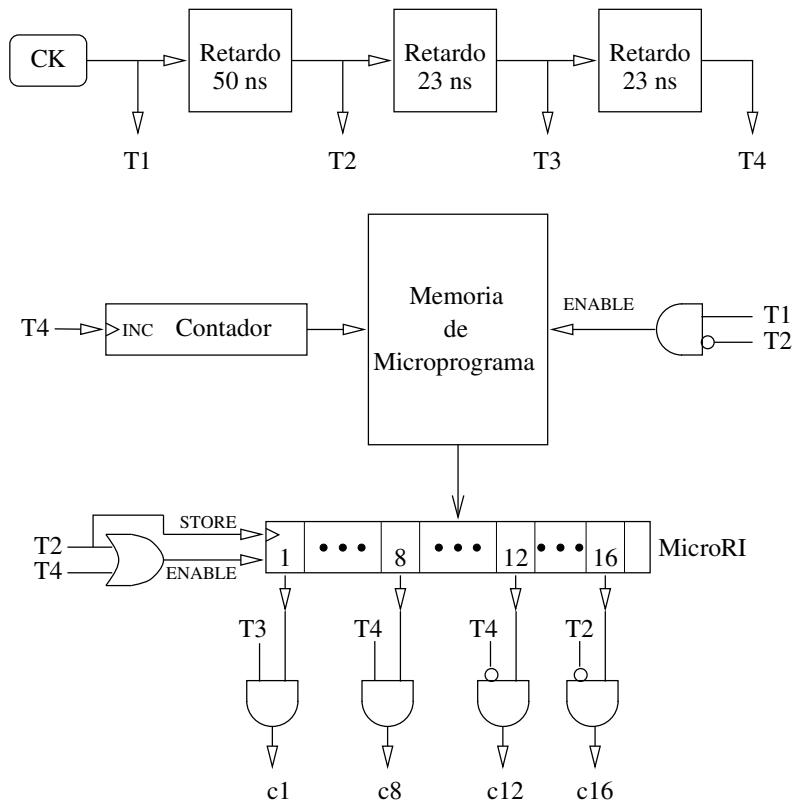


Figura 4.13: Esquema parcial de las conexiones de temporización

mador; en este caso, se conectará al conmutador c16 el AND de la línea de control c16 y de  $\overline{T2}$ , de forma que el desplazamiento esté controlado por el flanco de bajada de T2 y, por tanto, ocurrirá (si procede) unos 50 ns después de la habilitación del conmutador c8.

Finalmente, los conmutadores c9 a c12 deben habilitarse para terminar el ciclo (fase 7); en consecuencia, para cada una de sus respectivas señales de control deberá realizarse el AND con  $\overline{T4}$ , de forma que los registros I, A, B y/o D almacenarán la información desde el bus X en el flanco de bajada de la señal T4. La figura 4.14 muestra el diagrama de tiempos de las distintas fases durante un ciclo máquina del procesador.

En dicha figura, las líneas discontinuas indican que el punto de control asociado se activa sólo si el bit correspondiente vale 1.

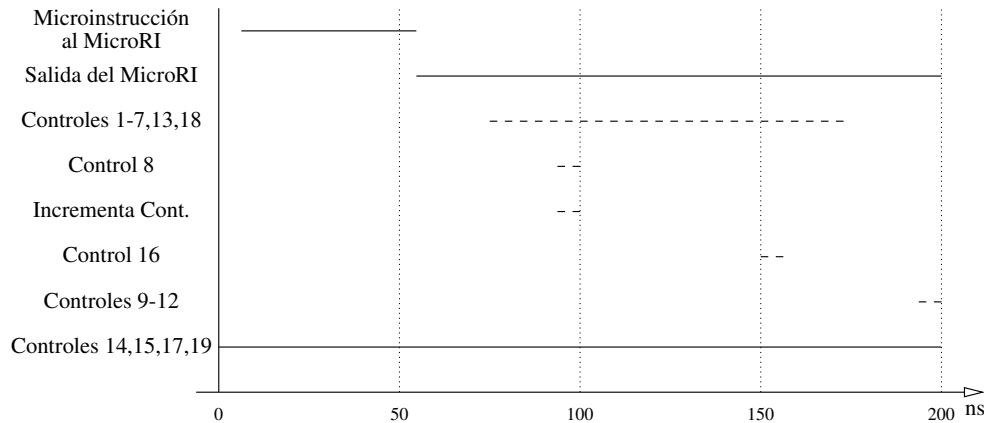


Figura 4.14: Diagrama de tiempos del ciclo máquina

#### 4.3.2. Codificación de las microinstrucciones

Hasta este punto, la filosofía seguida en el diseño de la microinstrucción ha sido una filosofía que llamaremos **horizontal** o sin codificación, en la que a cada bit corresponde una señal de control. Sin embargo, los 19 bits empleados dan lugar a 524.288 combinaciones diferentes, cantidad mucho mayor que el número máximo de microinstrucciones distintas que se emplearán previsiblemente. Esto significa que la palabra de control contiene una redundancia considerable. Si el tamaño de la memoria de control es un factor importante en el coste final del diseño, será necesario reducir el tamaño de la microinstrucción introduciendo codificación, a costa de reducir la velocidad de ejecución.

La metodología básica consiste en localizar en la microinstrucción grupos de bits de control que sean mutuamente exclusivos o redundantes y codificarlos. El ejemplo más sencillo son los bits 1 al 4, que controlan cuál de los registros (I, A, B o cero) depositará su contenido en la entrada izquierda del sumador, L. Dado que sólo uno de estos cuatro registros puede ser seleccionado simultáneamente, es posible codificarlo usando tan sólo dos bits, de la siguiente manera:

| Código | Registro |
|--------|----------|
| 00     | ceros    |
| 01     | I        |
| 10     | A        |
| 11     | B        |

Para que esta representación funcione, es necesario incluir una lógica de decodificación entre el registro de microinstrucción y los conmutadores del *datapath*. Así, por ejemplo, si los bits empleados para codificar el registro seleccionado son el 1 y el 2, la señal de habilitación del conmutador c1 deberá cumplir la ecuación:

$$c1 = \overline{b_1} \text{ AND } b_2 \text{ AND } T_3$$

En la figura 4.15 se representa la lógica necesaria para la generación de esta señal. Una lógica de decodificación análoga deberá ser incluida para los conmutadores c2 a c4.

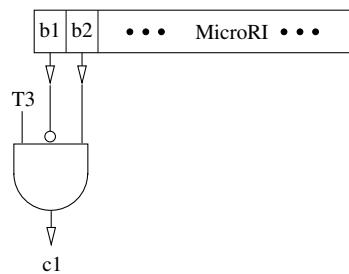


Figura 4.15: Decodificador de microfunción

Como se observa, al codificar la información contenida en los bits b1 y b2 del registro de microinstrucción, ahora ninguno de ellos tiene sentido por separado, sino que significan algo sólo cuando son analizados juntos. A este conjunto de uno o más bits en una microinstrucción que controlan un fragmento del *datapath* y que sólo tienen sentido al ser analizados juntos, es lo que habíamos denominado **microfunción** o **microorden**.

Examinando el *datapath*, es posible reducir aún más el número de bits de control. Así, los conmutadores c5 a c7, que controlan el registro origen en la entrada derecha del sumador, pueden codificarse con dos bits, quedando una combinación sin usar. Por otra parte, los conmutadores c9 a c12 controlan el destino del contenido del bus X. Teniendo en cuenta que no es necesario almacenar este contenido en más de un registro distinto y que es preciso incluir un caso en que el bus X no sea almacenado en ningún registro, deberán usarse tres bits, en los que se puede usar la siguiente codificación de destino:

| Código  | Destino          |
|---------|------------------|
| 000     | no operación     |
| 001     | I                |
| 010     | A                |
| 011     | B                |
| 100     | D (hace c14 = 0) |
| 101-111 | no usados        |

La inclusión de un código que no vuelca el contenido del bus X en ningún registro permite eliminar la necesidad de generar la señal de control para el conmutador c8, asociado al registro de desplazamiento. En efecto, cuando una microinstrucción usa el sumador esta señal debe ser 1; sin embargo, cuando la microinstrucción no usa el sumador, puede también dejarse esta señal a 1 y elegir el código 000 como destino del bus X. En consecuencia, se elimina el bit de control del conmutador c8, que estará controlado directamente por la señal de reloj T4.

Existe también redundancia entre los bits 12 a 14 (control de la entrada y salida del registro D) y los bits 18 y 19 (acceso a memoria y lectura/escritura, respectivamente). En efecto, en una operación de lectura de memoria (c18 y c19 a 1), debe habilitarse el conmutador c12, deshabilitarse el conmutador c13 y ponerse a 1 la línea c14 de selección en el multiplexor D-Mpx. Por el contrario, en una operación de escritura a memoria (bit 18 a 1, bit 19 a 0) el conmutador c12 debe permanecer deshabilitado, mientras el c13 deberá habilitarse. Teniendo en cuenta que el control del registro D en operaciones que no involucren a la memoria es efectuado por la microorden destino del bus X, resulta evidente que los bits 12 a 14 son superfluos, pues el estado de las señales de control c12 a c14 puede decodificarse de los bits 18 y 19 y de la microorden del bus X.

Como resumen de las simplificaciones sugeridas, en la tabla 4.5 se muestra la nueva definición de las microordenes, en la que se han reducido las necesidades de almacenamiento de control en un 37%; adicionalmente, se ha reducido de forma significativa las posibilidades de codificar microinstrucciones erróneas o sin significado. El precio a pagar por ello es un aumento de la lógica de control necesaria y una pérdida de velocidad y posibilidad de funcionamiento concurrente. A esta filosofía de representación de microinstrucciones la hemos llamamos **vertical** o con codificación.

#### 4.3.3. Expansión funcional del “datapath”: ALU

La funcionalidad del *datapath* empleado hasta ahora como ejemplo es muy limitada debido al hecho de que la única operación que puede realizar es la

| Microorden                    | Bits  | Caso                                       | Función   |
|-------------------------------|-------|--|---|
| Entrada izquierda del Sumador | 1-2   | 00<br>01<br>10<br>11                       | 0<br>I<br>A<br>B  |
| Entrada derecha del Sumador   | 3-4   | 00<br>01<br>10<br>11                       | 0<br>B<br>D<br>No usada   |
| Carry de entrada al Sumador   | 5     | 0<br>1                                     | Carry=0<br>Carry=1  |
| Desplazamiento                | 6-7   | 00<br>01<br>10<br>11                       | No desplazamiento<br>Desplazamiento Izq.<br>Desplazamiento Der.<br>No usada |
| Destino del BUS X             | 8-10  | 000<br>001<br>010<br>011<br>100<br>101-111 | Ninguno<br>I<br>A<br>B<br>D<br>No usada                                     |
| Operación de Memoria          | 11-12 | 00<br>01<br>10<br>11                       | No operación<br>Leer a D<br>Escribir de D<br>No usada                       |

Tabla 4.5: Definición de microinstrucción codificada

suma. Este defecto es fácilmente solventable sustituyendo el sumador binario completo por una ALU capaz de realizar las siguientes funciones:  $L + R + C$ ;  $L + \bar{R} + C$ ;  $L \text{ OR } R$  y  $L \text{ AND } R$ .

Con cuatro posibles operaciones para ejecutar, se necesitarán dos bits para su selección, c20 y c21, por lo que deberá incrementarse en dos bits el tamaño de la microinstrucción. En la figura 4.16 vemos que esta ALU también proporciona dos *flags* de estado, C (carry) y Z (cero), que son almacenados en un registro de **FLAGS**. Un bit de control adicional, c22, señalará la carga de dicho registro.

#### 4.3.4. Capacidad de salto y constantes en la microinstrucción

El campo “dirección de salto” en las microinstrucciones sólo es útil en aquellas que supongan un salto en el microprograma, siendo un conjunto de bits desaprovechados en el resto. No obstante, podemos ampliar la funcionalidad

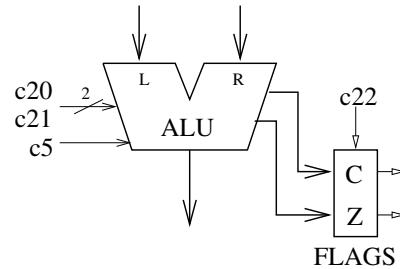


Figura 4.16: Ampliación de la ALU

de las microinstrucciones, permitiendo que, en las instrucciones de no salto, el contenido de este campo pueda ser usado como una constante (valor) en la microinstrucción. Para ello, abrimos un camino de datos que permita cargar en la ALU el contenido del campo “dirección de salto” del registro MicroRI. Este camino estará controlado con un nuevo punto de control, c23, que se muestra en la figura 4.17. A este campo le llamaremos desde ahora “dirección/constante”.

Respecto a los esquemas de temporización necesarios para el buen funcionamiento de estas unidades de procesamiento y control mejoradas, es necesario realizar un cambio importante relativo al momento en que se debe cargar el MicroPC. Es decir, si queremos que en una misma microinstrucción se pueda realizar una operación aritmética y un salto condicional al resultado de ésta, tendremos que permitir la carga o el incremento del MicroPC una vez haya concluido la operación en la ALU y se haya actualizado convenientemente el registro de FLAGS. Por ejemplo, en la fase 5 de la página 92, en que se habilita c8 para cargar el registro de desplazamiento con el resultado de la ALU, también podemos activar c22 para actualizar el registro de FLAGS si es necesario, es decir, con el flanco de subida de T4. De esta forma, el incremento o carga del MicroPC, condicionada al valor de los flags, se puede realizar con el flanco de bajada de T3 o T4.

#### 4.3.5. Microprogramación del conjunto de instrucciones

Para que el conjunto datapath y sección de control pueda ser usado como un procesador de propósito general, añadiremos un sistema de proyección a continuación de IR y un contador de programa PC. De esta forma, el procesador final, incluida la sección de control, sería el mostrado en la figura 4.17, en el que se muestran todos los puntos de control incluidos los adicionales (carga de constantes,...). Puesto que hemos empleado codificación en la microinstruc-

ción, es necesario usar un decodificador a la salida de MicroRI. El significado de los bits que componen la microinstrucción se muestra en la tabla 4.6.

| Microorden                    | Bits  | Caso  | Función   |
|-------------------------------|-------|---|---|
| Entrada izquierda del Sumador | 1-2   | 00<br>01<br>10<br>11                              | 0<br>I<br>A<br>B  |
| Entrada derecha del Sumador   | 3-5   | 000<br>001<br>010<br>011<br>100<br>101-111        | 0<br>B<br>D<br>PC<br>Constante (bits 21-30)<br>No usado   |
| Carry de entrada al Sumador   | 6     | 0<br>1  | Carry=0<br>Carry=1  |
| Desplazamiento                | 7-8   | 00<br>01<br>10<br>11                              | No desplazamiento<br>Desplazamiento Izq.<br>Desplazamiento Der.<br>No usada   |
| Destino del BUS X             | 9-11  | 000<br>001<br>010<br>011<br>100<br>101<br>110-111 | Ninguno<br>I<br>A<br>B<br>D<br>PC<br>No usada   |
| Operación de Memoria          | 12-14 | 000<br>001<br>010<br>011<br>100<br>101-111        | No operación<br>$(A) \rightarrow D$<br>$D \rightarrow (A)$<br>$(PC) \rightarrow RI$<br>$(PC) \rightarrow D$<br>No usada |
| Función de la ALU             | 15-16 | 00<br>01<br>10<br>11                              | Suma ( $L + R + C$ )<br>Resta ( $L + \bar{R} + C$ )<br>OR ( $L \text{ OR } R$ )<br>AND ( $L \text{ AND } R$ )           |
| Carga del Registro de FLAGS   | 17    | 0<br>1  | No modifica el registro<br>Actualiza el registro  |
| Control de MUX2               | 18    | 0<br>1  | Pasa Dirección de MicroRI<br>Pasa Dirección de ROM  |
| Control de Salto              | 19-20 | 00<br>01<br>10<br>11                              | $\mu PC = \mu PC + 1$<br>Salto Condicional si C<br>Salto Condicional si Z<br>$\mu PC = \text{Dirección}$                |
| Dir. Salto/Constante          | 21-30 |   |   |

Tabla 4.6: Definición final de microinstrucción

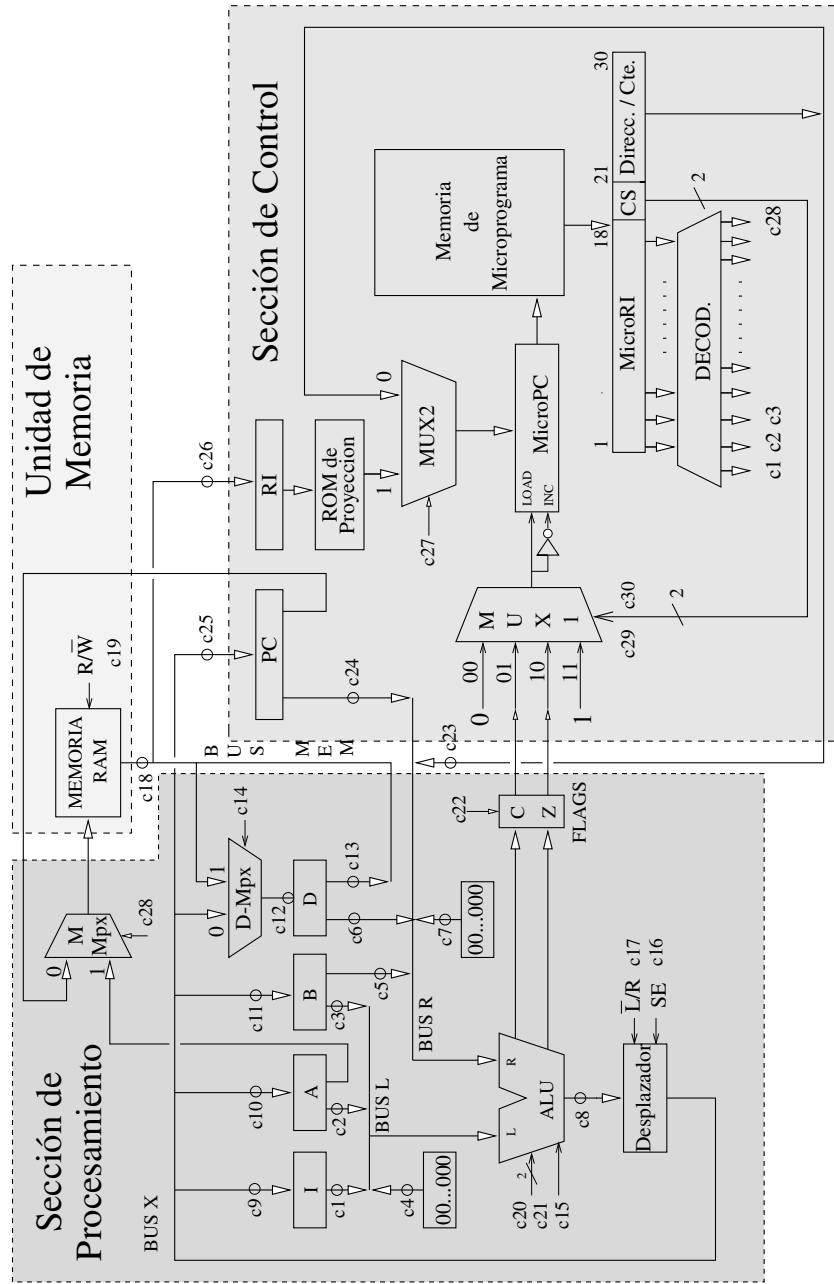


Figura 4.17: Procesador hipotético completo. Secciones de control y procesamiento con codificación horizontal

Aclararemos con un ejemplo el problema de escribir una microrutina asociada a una instrucción máquina dada para este procesador. Consideremos la instrucción ejemplo:

### AND 25(A),B

Esta instrucción debe almacenar en el registro B el resultado de hacer el AND lógico entre el contenido de este mismo registro y el contenido de la posición de memoria apuntada por el registro A más 25. Es decir:

$$(25 + A) \text{ AND } B \rightarrow B$$

Dado que el C.O. ocupa la palabra completa de memoria, el desplazamiento, 25, del direccionamiento relativo, se almacena en la siguiente palabra. Por tanto, las microinstrucciones que se deben ejecutar son las siguientes:

1.  $RI \leftarrow (PC)$ : Cargar el contenido de memoria apuntado por el PC en RI, es decir, **búsqueda de instrucción**.
2.  $c27 \leftarrow 1; CS = 11; PC \leftarrow PC + 1$ : **Decodificación** (carga del MicroPC con la dirección que viene de la ROM de Proyección) e incremento del PC.
3.  $D \leftarrow (PC); PC \leftarrow PC + 1$ : Extraemos el desplazamiento de la memoria (el “25”) e incrementamos el PC para que apunte a la siguiente instrucción.
4.  $A \leftarrow A + D$ : Completamos el **cálculo de la dirección efectiva** sumando al registro A el desplazamiento previamente almacenado en D.
5.  $D \leftarrow (A)$ : **Búsqueda del operando** en memoria a partir de su dirección efectiva almacenada en A.
6.  $B \leftarrow (B \text{ AND } D); CS = 11; Dir = 0$ : **Ejecución**, realizando el AND lógico, y salto final a la posición 0, donde se encuentra la rutina de fetching.

Siguiendo la codificación de las microordenes presentada en la tabla 4.6 y suponiendo que el núcleo de la microrutina se almacena a partir de la posición 12 de la memoria de microprograma, tendremos que el contenido parcial de esta memoria es el siguiente:

| Dirección<br>~~~~~ | Contenido<br>~~~~~                       |
|--------------------|--|
| 00000 00000        | 00 000 0 00 000 011 00 0 0 00 0000000000 |
| 00000 00001        | 00 011 1 00 101 000 00 0 1 11 0000000000 |
| .                  | .  |
| 00000 01100        | 00 011 1 00 101 100 00 0 0 00 0000000000 |
| 00000 01101        | 10 010 0 00 010 000 00 0 0 00 0000000000 |
| 00000 01110        | 00 000 0 00 000 001 00 0 0 00 0000000000 |
| 00000 01111        | 11 010 0 00 011 000 11 1 0 11 0000000000 |

La ROM de proyección debe contener la traducción entre códigos de operación y dirección de microrutina asociada a la instrucción.

Como vemos en este ejemplo, el código de operación de cada instrucción contiene en este caso información, no sólo de la operación a realizar y del modo de direccionamiento, sino también de los operandos. Esto implica que, según nuestro ejemplo, la instrucción **AND 10(A), I** debe tener una microrrutina distinta a la descrita anteriormente, con el consiguiente consumo de Memoria de Microprograma. En diseños más elaborados, el C.O. no ocupará todo el RI, dejando espacio para especificar los registros que se vayan a acceder, los cuales serán seleccionados directamente desde el propio RI.

#### 4.4. DISEÑO DE UNA UNIDAD DE CONTROL CABLEADA

El diseño de unidades de control cableadas implica un compromiso entre la cantidad de hardware usado, su velocidad de operación y el coste de su proceso de diseño. Los métodos de diseño utilizados en la práctica son, a menudo, de naturaleza heurística y ad hoc (no generales, sino orientados al problema concreto a resolver). Además, estos métodos de diseño no pueden formalizarse fácilmente, debido al gran número de señales de control necesarias en una unidad de control y a su dependencia con el conjunto de instrucciones particular que se implementa.

Para ilustrar el problema, consideramos dos aproximaciones simplificadas y sistemáticas para el diseño de unidades de control cableadas. Estos métodos son representativos de los que se usan en la práctica, pero son adecuados únicamente para unidades de control sencillas, tales como las que se pueden encontrar en controladores no programables o en procesadores RISC. Estas dos aproximaciones se denominan respectivamente **método de los elementos de retardo** y **método de los contadores de secuencia**.

Vamos a explicar estos métodos sobre un ejemplo, diseñando una unidad

de control para la sección de procesamiento que se presentó anteriormente en la figura 4.3. Asociada a esta sección se detallaron los puntos de control en la tabla 4.1 .

Desde el punto de vista del diseño consideraremos que los señales de control del procesador son activas a nivel alto. No obstante aquellas que actúen por flanco de subida o bajada sobre el commutador, (carga, desplazamiento, incremento de registros) tendrán que ser combinada con la señal de reloj.

Consideraremos un repertorio de instrucciones reducido, que se muestra en la tabla 4.7<sup>8</sup>.

| Mnemónico | Descripción   |
|-----------|---|
| LOAD A    | Transfiere el contenido de la posición A al acumulador: $(A) \rightarrow ACC$ |
| STORE A   | Transfiere el contenido del acumulador a la posición A: $ACC \rightarrow (A)$ |
| ADD A     | Suma al acumulador el contenido de la posición A: $ACC + (A) \rightarrow ACC$ |
| JMPZ A    | Salta a la dirección A si el resultado fue cero: <i>si Z : A → PC</i>         |

Tabla 4.7: Repertorio de instrucciones

Vamos a aplicar los métodos antes citados para la construcción de este procesador simplificado. En la figura 4.18 se representa el diagrama de flujo del funcionamiento del procesador. BEGIN representa una señal de inicio. Como se puede observar la tarea del procesador se divide en dos fases básicas: búsqueda de instrucción y ejecución. Durante la búsqueda se accede a la posición de memoria apuntada por el PC y se transfiere ese contenido al registro de instrucción IR. El PC es incrementado apuntando ahora a la siguiente dirección de memoria. El contenido de IR se decodifica (identificación de a qué instrucción pertenece el código de operación) y según la operación a realizar se ejecuta una u otra cosa. En la fase de ejecución se busca el operando, se ejecuta la operación de ALU y/o se almacena datos en memoria según proceda.

#### 4.4.1. Método de los elementos de retardo

En este método se parte del diagrama de flujo de control para crear el circuito encargado de producir las señales de control. Para ello lo que vamos a hacer es generar el conjunto de señales necesarias para cada operación elemental (transferencia de registros) que hemos escrito en el diagrama de flujo, en el orden adecuado. Cada etapa del diagrama de flujo lo denominaremos microoperación.

---

<sup>8</sup>Con (A) nos estamos refiriendo al contenido de la posición A de memoria

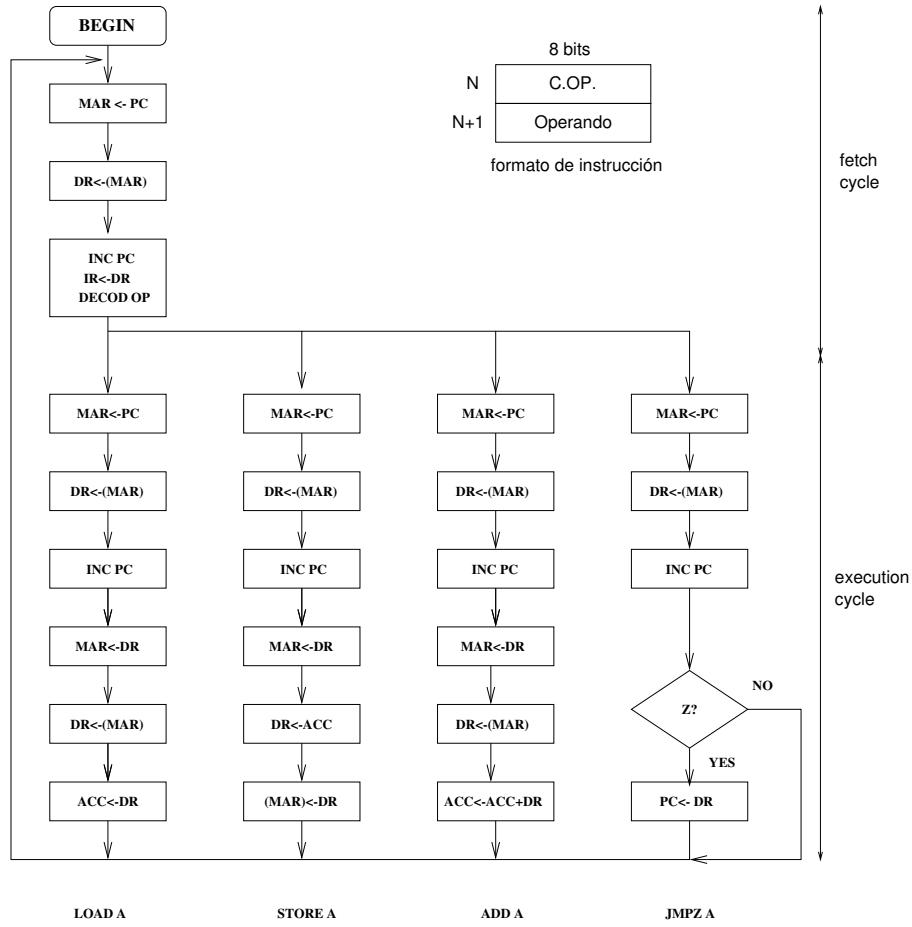


Figura 4.18: Diagrama de flujo a nivel RTL para el procesador de la fig. 4.3

Vamos a introducir un conjunto de señales auxiliares  $C_{ji}$  cuyo significado es: “ $C_{ji}$  se activa en un intervalo  $t_i$  si la señal de control  $C_j$  debe estar activa en ese intervalo”.

Consideremos el problema de generar la siguiente secuencia de señales de control en los instantes  $t_1, t_2, \dots, t_n$ :

```

t1: Activar Cj1;
t2: Activar Cj2;
...
tn: Activar Cjn;

```

Aquí con  $C_{ji}$  representamos la activación del punto de control  $C_j$  en el

instante  $t_i$ . ¡Cuidado! no todas las señales se activan en todos los instantes; en cada instante se activarán aquel subconjunto de señales que intervenga en la transferencia de registros que queramos realizar). En cada ciclo o intervalo  $t_i$  se ejecuta una microoperación.

Podemos producir la cadencia de señales de control introduciendo elementos de retardo (“delay element”) y propagando la señal de activación. Con este método la translación entre el diagrama de flujo y la sección de control es directa ya que el diagrama de flujo expresa la secuencia de activación de los puntos de control. El circuito diseñado tiene esencialmente la misma estructura que el diagrama de flujo. La forma de obtener el circuito a partir del diagrama de flujo nos la indica unas reglas simples, que se ven en la figura 4.19. A continuación se explican dichas reglas.

1. Cada secuencia de dos microoperaciones sucesivas requiere un elemento de retardo. Las señales que activan las líneas de control se toman directamente de las líneas de entrada y salida del circuito de retardo. Las señales que activan una misma línea de control  $C_i$  se unen con una puerta OR cuya salida es  $C_i$ . Esta línea puede conectarse al punto de control que activa (introduciendo la temporización adecuada si fuera preciso).
2.  $k$  líneas en el diagrama de flujo que se unen a una línea común se transforma en una puerta OR de  $k$  entradas, como muestra la figura 4.19
3. Un bloque de decisión (operación de salto condicional) o bifurcación se implementa con puertas AND, como aparece en la figura 4.19.

No olvidemos que las señales de control así generadas  $C_j$  pueden necesitar ser combinadas con la señal de reloj para generar los flancos adecuados cuando conectemos las señales con la sección de procesamiento.

Aplicando esta filosofía el resultado para el procesador que estamos construyendo es el que se muestra en la figura 4.20.

En este esquema hay que tener en cuenta:

- Como se puede observar el código de operación contenido en IR, es decodificado para producir 4 señales (LOAD, STORE, ADD, JMPZ) que nos sirven para escoger qué rama del diagrama de flujo seguir según la instrucción que represente el código de operación.
- Es necesario introducir una señal BEGIN (no aparece en el esquema) que ponga a cero (CLEAR) todos los elementos de retardo (y así todas las señales de control) salvo el primer elemento (señalado con ‘\*’) de retardo que se pondrá a uno (PRESET) cuando comience la ejecución. Esta señal así mismo deberá poner el PC a un valor conocido apuntando a la primera instrucción a ejecutar (lo más sencillo sería iniciarla a cero).

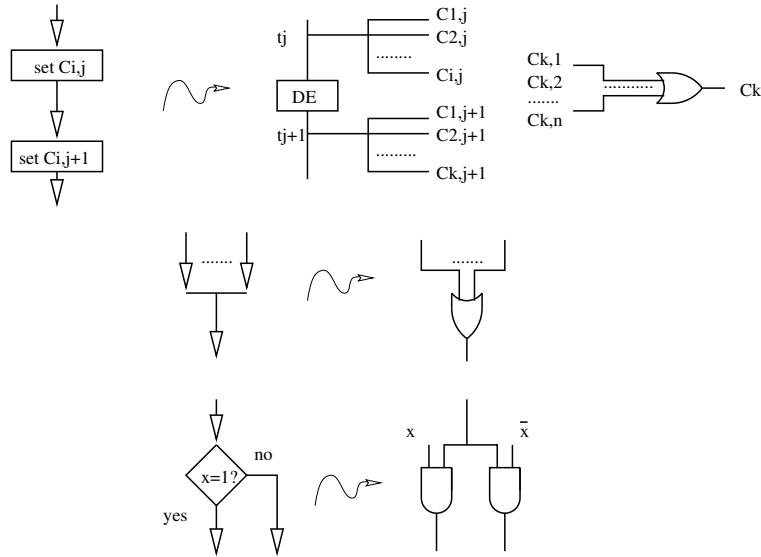


Figura 4.19: Transformación entre diagrama de flujo y circuito con elementos de retardo

- Existen señales como  $C_{8,4}$ ,  $C_{8,10}$ ,  $C_{8,16}$ ,  $C_{8,22}$  que no se pueden activar nunca a la vez ya que pertenecen a “ramas” excluyentes en el flujo de ejecución. Es por esta razón por lo que se ha usado una numeración para el segundo subíndice diferente, aunque las 4 señales, cuando se activan, lo harían en el intervalo ficticio  $t_4$ . (Es decir dentro del ciclo de ejecución de una instrucción  $t_4$ ,  $t_{10}$ ,  $t_{16}$ ,  $t_{22}$  representan el mismo intervalo temporal real).

Una vez que se posee el circuito que en cada instante genera los  $C_{ji}$  adecuadamente, hay que generar las señales de control  $C_j$ . Como el significado de  $C_{ji}$  es la activación de la señal  $C_j$  en el instante  $t_i$ , los  $C_j$  los determinamos con el OR de todos los  $C_{ji}$  que se activan para algún  $t_i$ . Las ecuaciones que resultan son las siguientes:

$$C_j = \sum_i C_{ji}$$

El elemento de retardo (DE), se construye simplemente con un biestable tipo D activo por flanco (por ejemplo de subida). Construido de esta manera la señal de reloj de los elementos de retardo debe ser aquella que sincronice también los puntos de control activos por flanco en la sección de datos. Así por

ejemplo la señal  $C_7$  que controla la carga del registro DR debe ser combinada con la señal de reloj cuando es conectada en el punto de control  $c7$ :  $c7 \equiv C_7 \cdot \overline{CK}$ ; o para el acceso a memoria  $c3 \equiv C_3 + \overline{CK}$  ya que cuando se escribe el dato se almacena en el flanco de subida de  $c3$ .

#### 4.4.2. Método del contador de secuencias

Consideremos el circuito de la figura 4.21, que consiste basicamente en un contador módulo  $k$  cuya salida esta conectada a un decodificador  $\frac{1}{k}$ . Si la entrada ENABLE del contador se conecta a un reloj fuente, el contador ira pasando continuamente por sus  $k$  estados. El decodificador genera  $k$  señales pulso  $S_j$  en su salida y, además, pulsos consecutivos están separados por un periodo de reloj, como muestra la figura. Las señales  $S_j$  dividen el tiempo requerido para que un contador recorra su ciclo completo en  $k$  partes iguales. A estas señales  $S_j$  se les llama señales de fase. Se necesita un flip-flop y dos líneas de entrada para poner en marcha o parar el contador. Un pulso en la línea BEGIN hace que el contador comience a recorrer sus estados cíclicamente, conectando la línea ENABLE del contador al reloj ( $CK$ ) fuente. Un pulso en la línea END desconecta el reloj y resetea el contador. Supondremos que la señal de “reset” del contador es síncrona con la entrada de reloj. El circuito de la figura 4.21 se llama contador de secuencia y lo representaremos tal como se indica en dicha figura.

La utilidad de los contadores de control de este tipo radica en el hecho de que muchos circuitos digitales se diseñan para realizar un número relativamente pequeño de acciones repetidamente.

Cada paso a través del lazo constituye un ciclo de microoperación. Observando el diagrama de flujo de la figura 4.18 vemos que desde que una instrucción comienza su ejecución, en el caso peor se consumen 9 etapas (longitud del camino mas largo). Si suponemos que cada etapa puede realizarse en un periodo de reloj elegido, entonces podremos construir una unidad de control para esta CPU, partiendo de un simple contador de secuencia módulo 9. Cada señal  $S_i$  activara las líneas de control que procedan, en la etapa  $i$ -ésima.

El diseño del circuito que genera las señales de control se reduce a un combinacional que activará cada control  $C_j$  en función del código de operación, del contenido del registro de estados y de la etapa ( $S_i$ ) en la que se encuentre la ejecución. El combinacional, que representamos en la figura 4.22, debe implementar las ecuaciones lógicas que se muestran:

$$\begin{aligned} C_0 &= ADD \cdot S_9 \\ C_1 &= 0 \end{aligned}$$

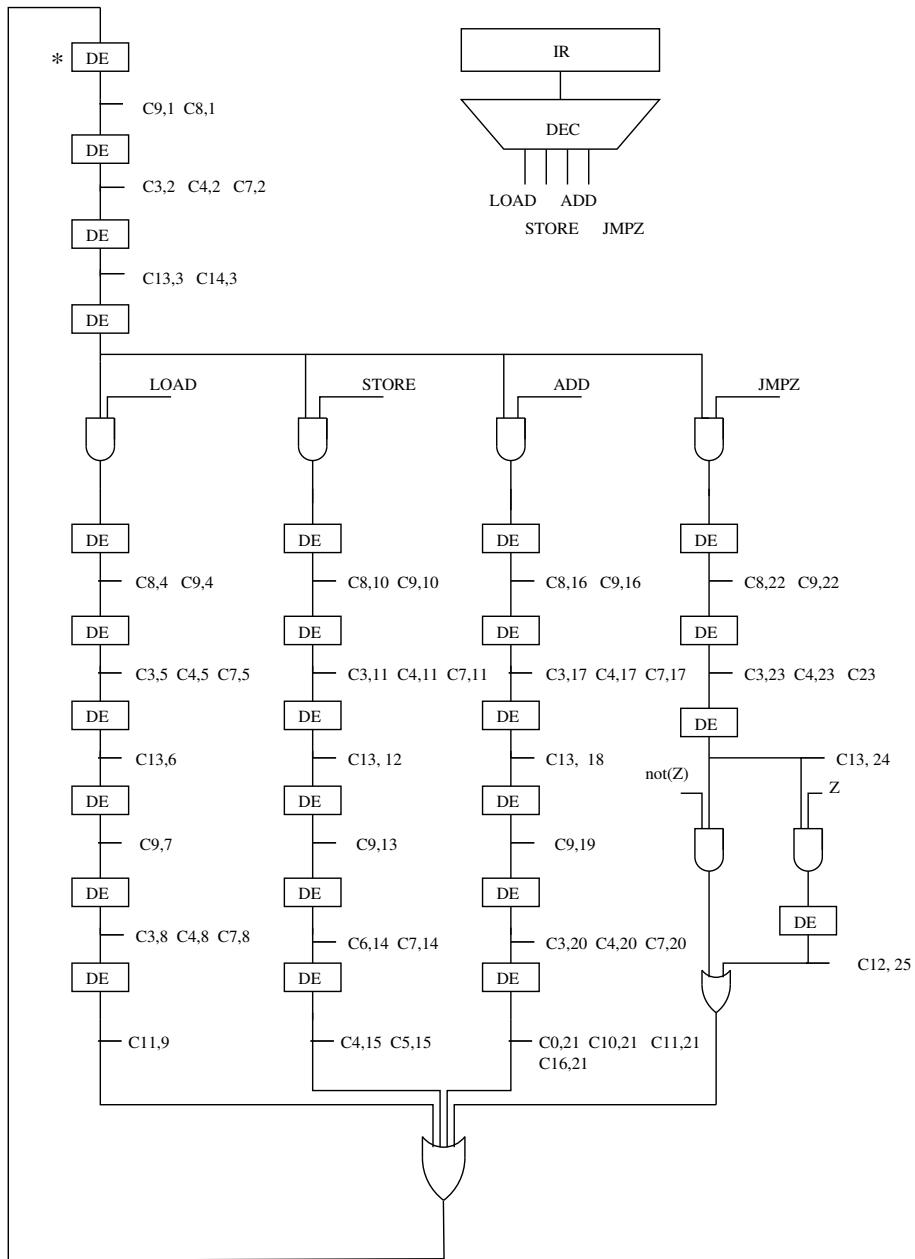


Figura 4.20: Señales de control con elementos de retardo

$$\begin{aligned}
 C_2 &= 0 \\
 C_3 &= S_2 + LOAD \cdot S_5 + STORE \cdot S_5 + ADD \cdot S_5 + JMPZ \cdot \\
 S_5 + LOAD \cdot S_8 + ADD \cdot S_8 & \\
 C_4 &= S_2 + LOAD \cdot S_5 + STORE \cdot S_5 + ADD \cdot S_5 + JMPZ \cdot \\
 S_5 + LOAD \cdot S_8 + ADD \cdot S_8 & \\
 C_5 &= STORE \cdot S_9 \\
 C_6 &= STORE \cdot S_8 \\
 C_7 &= S_2 + LOAD \cdot S_5 + STORE \cdot S_5 + ADD \cdot S_5 + JMPZ \cdot \\
 S_5 + LOAD \cdot S_8 + STORE \cdot S_8 + ADD \cdot S_8 & \\
 C_8 &= LOAD \cdot S_5 + STORE \cdot S_5 + ADD \cdot S_5 + JMPZ \cdot S_5 + S_1 \\
 C_9 &= S_1 + LOAD \cdot S_7 + STORE \cdot S_7 + ADD \cdot S_7 + S_4 \\
 C_{10} &= ADD \cdot S_9 \\
 C_{11} &= LOAD \cdot S_9 + ADD \cdot S_9 \\
 C_{12} &= JMPZ \cdot S_7 \\
 C_{13} &= S_3 + S_6 \\
 C_{14} &= S_3 \\
 C_{15} &= 0 \\
 C_{16} &= ADD \cdot S_9 \\
 RESET &= JMPZ \cdot S_8 \cdot Z + JMPZ \cdot S_7 \cdot \bar{Z}
 \end{aligned}$$

Sobre estas ecuaciones podemos comentar:

- Su obtención es inmediata a partir del significado de las señales. Así por ejemplo  $C_3$  nos indica que se activa siempre en la segunda etapa, y en la etapa 5 cuando la instrucción es un LOAD, STORE, ADD o JMPZ, y en la etapa 8 cuando se trata de un LOAD o un ADD.
- Aunque se han expresado de forma que se entienda como se obtienen, las expresiones son susceptible de simplificación, más aún teniendo en cuenta que expresiones mutuamente exclusivas siempre ocurren, por ejemplo  $LOAD + STORE + ADD + JMPZ = 1$ .
- Hay señales que nunca se activan en este conjunto de instrucciones ( $C_1, C_2, C_{15}$ ).
- No todas las instrucciones consumen la 9 etapas. La instrucción JMPZ consume menos. Por eso hay que introducir la señal RESET (no olvidar que es síncrona con el reloj). Esta señal se activa sólo para JMPZ, cuya ejecución finaliza en la etapa 8 si Z está activo (salto) o en la etapa 7 si Z está inactivo (no salto).

Como se indicó para el método anterior, las señales de control deben ser combinadas con el reloj en aquellos casos en el que el conmutador de control funcione por flanco.

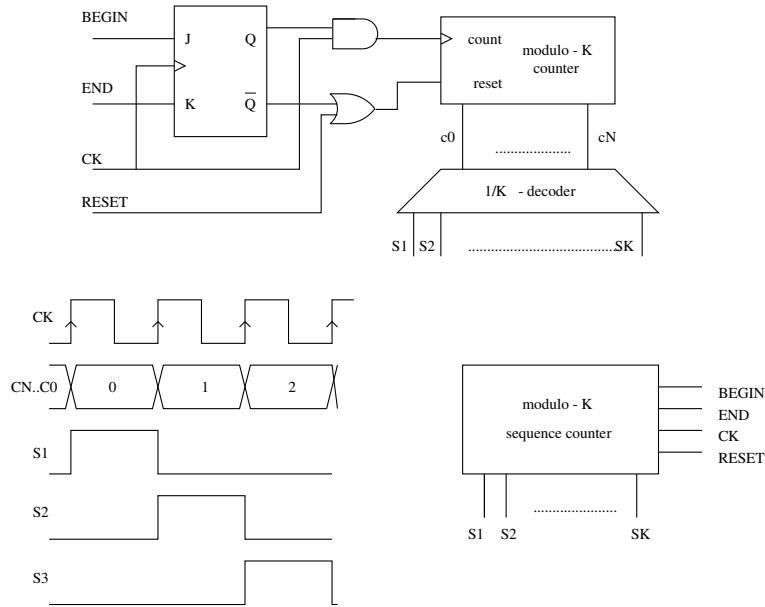


Figura 4.21: Contador de secuencias módulo k

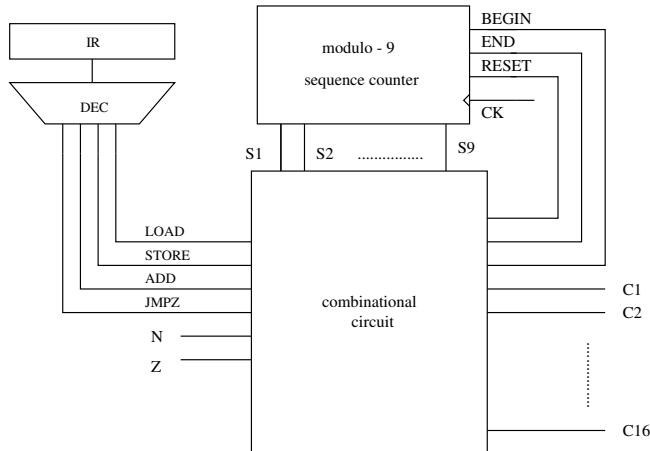


Figura 4.22: Unidad de control utilizando un contador de secuencias

## SINOPSIS

La técnica de control microprogramado permite el diseño de una unidad de control de forma sistemática y estructurada. En resumen, hemos visto que

básicamente necesitamos incluir dentro de la sección de control un contador de microprograma, **MicroPC**, un registro de microinstrucción, **MicroRI** y una **Memoria de Microprograma**, donde almacenamos cada una de las microrrutinas asociadas a cada instrucción máquina. Además, hemos introducido la metodología de diseño de unidades de control cableada, usadas en procesadores RISC.