


## 4.1

## Introducción

En el capítulo 1 vimos que las prestaciones de una máquina están determinadas por tres factores clave: el número de instrucciones, el tiempo del ciclo de reloj y los ciclos de reloj por instrucción (*cycles per instruction*, CPI). El compilador y la arquitectura del repertorio de instrucciones, que hemos examinado en el capítulo 2, determinan el número de instrucciones requerido por un cierto programa. Sin embargo, tanto el tiempo de ciclo del reloj como el número de ciclos por instrucción vienen dados por la implementación del procesador. En este capítulo se construye el camino de datos y la unidad de control para dos realizaciones diferentes del repertorio de instrucciones MIPS.

Este capítulo incluye una explicación de los principios y técnicas utilizadas en la implementación de un procesador, comenzando con un resumen altamente abstracto y simplificado en esta sección, seguido por una sección que construye un camino de datos y una versión simple de un procesador suficiente para realizar repertorios de instrucciones como MIPS. El núcleo del capítulo describe una implementación segmentada más realista del MIPS, seguido de una sección que desarrolla los conceptos necesarios para la implementación de un repertorio de instrucciones más complejo, como el x86.

Para el lector interesado en comprender la interpretación a alto nivel de las instrucciones y su impacto en las prestaciones del programa, esta sección inicial y la sección 4.5 proporcionan los conceptos básicos de la segmentación. Las tendencias actuales se indican en la sección 4.10, y la sección 4.11 describe el microprocesador AMD Opteron X4 (Barcelona). Estas secciones proporcionan suficiente bagaje para comprender los conceptos de segmentación a alto nivel.

Las secciones 4.3, 4.4 y 4.6 son útiles para los lectores interesados en entender el procesador y sus prestaciones con mayor profundidad y las secciones 4.2, 4.7, 4.8 y 4.9 para los interesados en como construir un procesador. Para los lectores interesados en diseño hardware moderno, la  [sección 4.12](#) en el CD describe como se utilizan los lenguajes de descripción hardware y las herramientas de CAD para la implementación de hardware y como utilizar un lenguaje de descripción hardware para describir una implementación segmentada. También se incluyen más figuras sobre la ejecución en un hardware segmentado.

### Una implementación básica MIPS

Examinaremos una implementación que incluirá un subconjunto básico del repertorio de instrucciones del MIPS formado por:

- Las instrucciones de referencia a memoria cargar palabra (*load word lw*) y almacenar palabra (*store word sw*).
- Las instrucciones aritmético-lógicas *add*, *sub*, *and*, *or* y *sllt*.
- Las instrucciones de saltar si igual (*branch on equal beq*) y salto incondicional (*jump j*), que se añadirán en último lugar.

Este subconjunto no incluye todas las instrucciones con enteros (por ejemplo, se han omitido las de desplazamiento, multiplicación y división), ni ninguna de las instrucciones de punto flotante. Sin embargo, servirá para ilustrar los principios básicos que se utilizan en la construcción del camino de datos y el diseño de la unidad de control. La implementación de las instrucciones restantes es similar.

Al examinar la implementación, tendremos la oportunidad de ver cómo la arquitectura del repertorio de instrucciones determina muchos aspectos de la implementación y cómo la elección de varias estrategias de implementación afecta a la frecuencia del reloj y al CPI de la máquina. Muchos de los principios básicos de diseño introducidos en el capítulo 1, como las directrices «hacer rápido el caso común» y «la simplicidad favorece la regularidad», pueden observarse en la implementación. Además, la mayoría de los conceptos utilizados para realizar el subconjunto MIPS en este capítulo y en el siguiente son las ideas básicas que se utilizan para construir un amplio espectro de computadores, desde servidores de altas prestaciones hasta microprocesadores de propósito general o procesadores empujados.

### Una visión general de la implementación

En el capítulo 2 analizamos el núcleo básico de instrucciones MIPS, incluidas las instrucciones aritmético-lógicas sobre enteros, las instrucciones de referencia a memoria y las instrucciones de salto. La mayor parte de lo necesario para implementar estas instrucciones es común para todas ellas, independientemente de su tipo concreto. Para cada instrucción, los dos primeros pasos son idénticos:

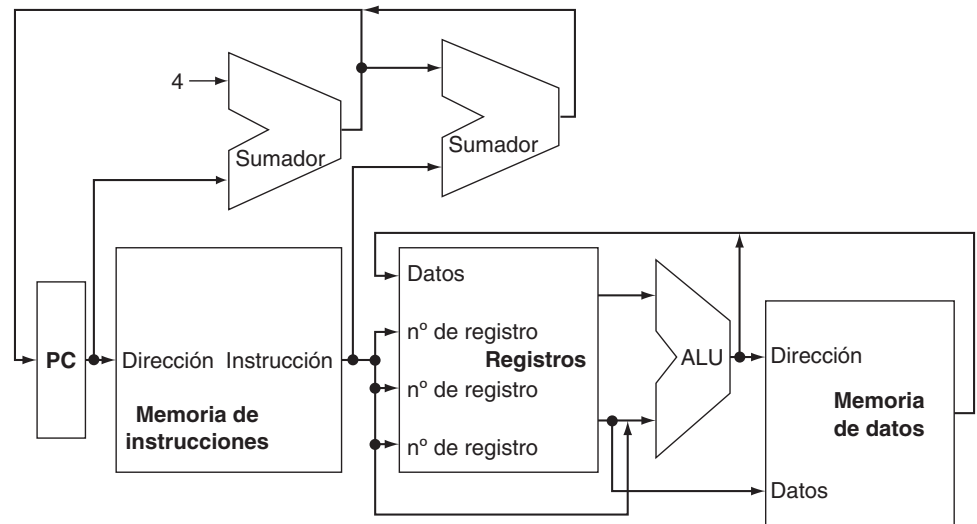
1. Enviar el contador de programa (PC) a la memoria que contiene el código y cargar la instrucción desde esa memoria.
2. Leer uno o dos registros, utilizando para ello los campos específicos de la instrucción para seleccionar los registros a leer. Para la instrucción de cargar palabra es necesario un solo registro, pero la mayoría del resto de las instrucciones requiere la lectura de dos registros.

Después de estos dos pasos, las acciones necesarias para completar la instrucción dependen del tipo de la misma. Afortunadamente, para los tres tipos de instrucciones (referencia a memoria, aritmético-lógicas y saltos) las acciones son generalmente las mismas, independientemente del código de operación exacto. La sencillez y regularidad del repertorio de instrucciones MIPS simplifica la implementación porque la ejecución de muchas clases de instrucciones es similar.

Por ejemplo, todos los tipos de instrucciones, excepto las de salto incondicional, utilizan la unidad aritmético-lógica (ALU) después de la lectura de los registros. Las instrucciones de referencia a memoria utilizan la ALU para el cálculo de la dirección, las instrucciones aritmético-lógicas para ejecutar la operación, y las de salto condicional para comparar. Después de utilizar la ALU, las acciones requeridas para completar la ejecución de los diferentes tipos de instrucciones son distintas. Una instrucción de referencia a memoria necesitará acceder a memoria, ya sea para escribir el dato en una operación de almacenamiento o para leer, en caso de una carga. Una instrucción aritmético-lógica o una de carga debe escribir el resultado calculado por la ALU o el leído de la memoria en un registro. Finalmente, en una instrucción de salto condicional es necesario modificar la dirección de la siguiente


instrucción según el resultado de la comparación; en caso contrario el PC debe incrementarse en 4 para obtener la dirección de la siguiente instrucción.


La figura 4.1 muestra un esquema de alto nivel de una implementación MIPS, en el que se muestran las diferentes unidades funcionales y su interconexión. A pesar de que esta figura muestra la mayor parte del flujo de datos en el procesador, omite dos aspectos importantes de la ejecución de las instrucciones.



**FIGURA 4.1 Una visión abstracta de la implementación del subconjunto MIPS en la que se muestra la mayor parte de las unidades funcionales y las conexiones entre ellas.** Todas las instrucciones comienzan utilizando el contador de programa (PC) para proporcionar la dirección de la instrucción en la memoria de instrucciones. Tras la captación de la instrucción, ciertos campos de ésta especifican los registros que se utilizan como operandos fuente. Una vez que éstos han sido leídos, puede operarse con ellos, ya sea para calcular una dirección de memoria (en una carga o un almacenamiento), para calcular un resultado aritmético (para una instrucción aritmético-lógica entera), o bien para realizar una comparación (en un salto condicional). Si la instrucción es una instrucción aritmético-lógica, el resultado de la ALU debe almacenarse en un registro. Si la operación es una carga o un almacenamiento, el resultado de la ALU se utiliza como la dirección donde almacenar un valor en memoria o cargar un valor en los registros. El resultado de la ALU o memoria se escribe en el banco de registros. Los saltos condicionales requieren el uso de la salida de la ALU para determinar la dirección de la siguiente instrucción, que proviene de la ALU (donde se suma el PC y el desplazamiento) o desde un sumador que incrementa el valor actual del PC en 4. Las líneas gruesas que interconectan las unidades funcionales representan los buses, que consisten en múltiples señales. Las flechas se utilizan para indicar al lector cómo fluye la información. Ya que las líneas de señal pueden cruzarse, se muestra explícitamente la conexión con la presencia de un punto donde se cruzan las líneas.

En primer lugar, en varios puntos, la figura muestra datos dirigidos a una unidad particular provenientes de dos orígenes diferentes. Por ejemplo, el valor escrito en el PC puede provenir de cualquiera de los dos sumadores, el dato escrito en el banco de registros puede provenir de la ALU o de la memoria de datos, y la segunda entrada de la ALU proviene de un registro o del campo inmediato de la instrucción. En la práctica, estas líneas de datos no pueden conectarse directamente; debe añadirse un elemento que seleccione entre los múltiples orígenes y dirija una de estas fuentes al destino. Esta selección se realiza comúnmente mediante un dispositivo denominado *multiplexor*, aunque sería más adecuado

denominarlo *selector de datos*. El multiplexor, que se describe en detalle en el  **apéndice C**, selecciona una de entre varias entradas según la configuración de sus líneas de control. Las líneas de control se configuran principalmente a partir de información tomada de la instrucción en ejecución.

En segundo lugar, varias de las unidades deben controlarse dependiendo del tipo de instrucción. Por ejemplo, la memoria de datos debe leer en una carga y escribir en un almacenamiento. Debe escribirse en el banco de registros en una instrucción de carga y en una instrucción aritmético-lógica. Y, por supuesto, la ALU debe realizar una entre varias operaciones, tal como se mostró en el capítulo 2. (El  **apéndice C** describe el diseño lógico detallado de la ALU.) Al igual que los multiplexores, estas operaciones son dirigidas por las líneas de control que se establecen según los diversos campos de la instrucción.

La figura 4.2 muestra el camino de datos de la figura 4.1 con los tres multiplexores necesarios añadidos, así como las líneas de control para las principales unidades funcionales. Para determinar la activación de las líneas de control de las unidades funcionales y de los multiplexores se utiliza una unidad de control que toma la instrucción como entrada. El tercer multiplexor, que determina si  $PC + 4$  o la dirección destino del salto se escribe en el PC, se activa según el valor de la salida Cero de la ALU, que se utiliza para realizar la comparación de la instrucción *beq*. La regularidad y simplicidad del repertorio de instrucciones MIPS implica que un simple proceso de descodificación puede ser utilizado para determinar cómo activar las líneas de control.

En el resto de este capítulo, se refina este esquema para añadir los detalles, lo que va a requerir que se incluyan unidades funcionales adicionales, incrementar el número de conexiones entre las unidades y, por supuesto, añadir una unidad de control que determine las acciones que deben realizarse para cada uno de los distintos tipos de instrucciones. Las secciones 4.3 y 4.4 describen una realización simple que utiliza un único ciclo de reloj largo para cada instrucción y sigue la forma general de las figuras 4.1 y 4.2. En este primer diseño, cada instrucción inicia su ejecución en un flanco de reloj y completa la ejecución en el siguiente flanco de reloj.

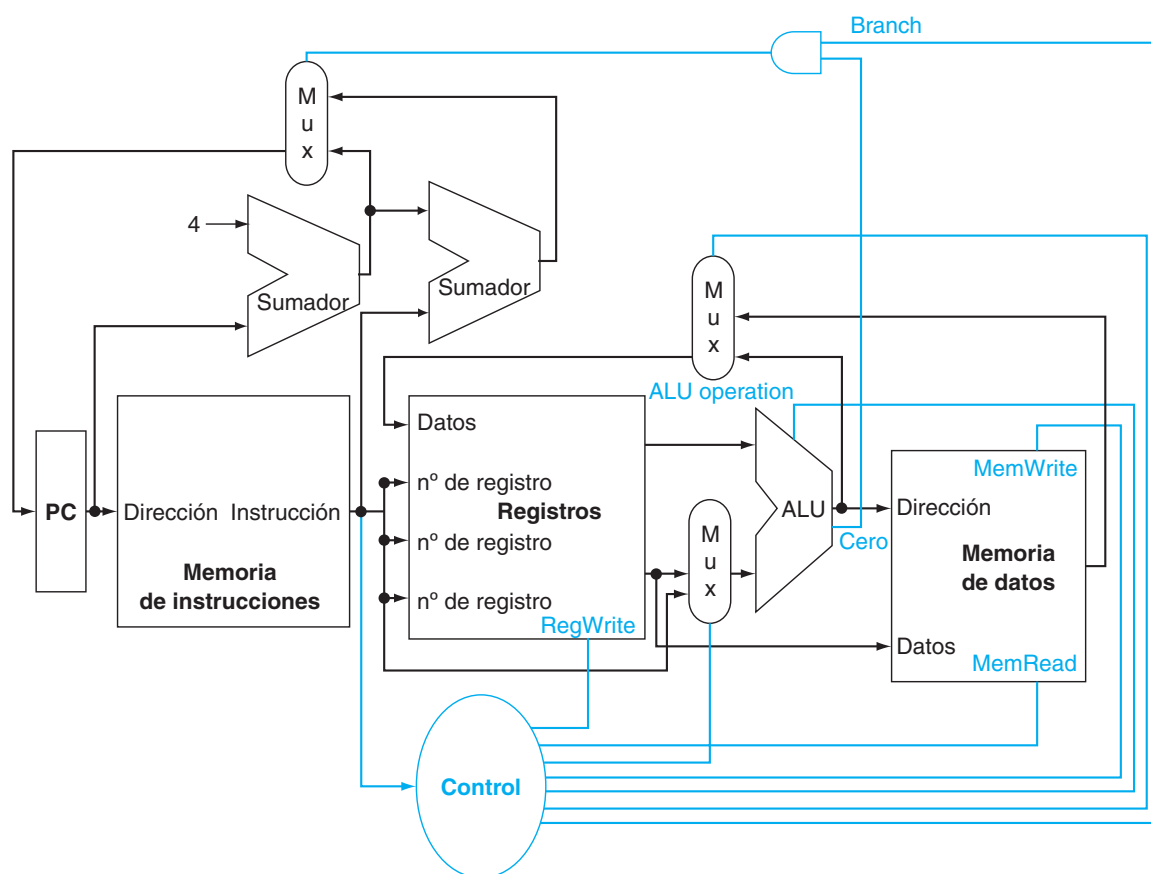
Aunque es más fácil de comprender, este enfoque no es práctico, puesto que hay que alargar el ciclo de la señal de reloj para permitir la ejecución de la instrucción más lenta. Una vez diseñado el control para este computador sencillo, se analizará la implementación segmentada en toda su complejidad, incluyendo el tratamiento de las excepciones.

¿Cuántos de los cinco componentes clásicos de un computador, mostrados en la página 299, se incluyen en la figuras 4.1 y 4.2?

**Autoevaluación**

## 4.2 Convenios de diseño lógico

Para tratar el diseño de la máquina, se debe decidir cómo operará su implementación lógica y cómo será su sincronización. Esta sección repasa unas cuantas ideas clave de diseño lógico que se utilizarán ampliamente en este capítulo. Si se tiene




**FIGURA 4.2 Implementación básica del subconjunto del MIPS incluyendo los multiplexores y líneas de control necesarias.** El multiplexor de la parte superior controla el valor que se va a cargar en el PC ( $PC + 4$  o la dirección destino del salto condicional); el multiplexor es controlado por una puerta que realiza una función «and» entre la salida Cero de la ALU y una señal de control que indica que la instrucción es de salto condicional. El multiplexor cuya salida está conectada al banco de registros se utiliza para seleccionar la salida de la ALU (en el caso de una instrucción aritmético-lógica) o la salida de la memoria de datos (en el caso de una carga desde memoria) para escribir en el banco de registros. Por último, el multiplexor de la parte inferior determina si la segunda entrada de la ALU procede de los registros (en una instrucción aritmético-lógica no inmediata) o del campo de desplazamiento de la instrucción (en una operación inmediata, una carga o almacenamiento, o un salto condicional). Las líneas de control añadidas son directas y determinan la operación realizada por la ALU, si se debe leer o escribir en la memoria de datos, y si los registros deben realizar una operación de escritura. Las líneas de control se muestran en color para facilitar su identificación.


escaso o ningún conocimiento sobre diseño lógico, el **apéndice C** resultará muy útil antes de abordar esta sección.

Las unidades funcionales de la implementación MIPS constan de dos tipos de elementos lógicos diferentes: elementos que operan con datos y elementos que contienen el estado. Los elementos que operan con datos son todos **combinacionales**, lo que significa que sus salidas dependen únicamente de los valores actuales de las entradas. Para una misma entrada, un elemento combinacional siempre produce la misma salida. La ALU mostrada en la figura 4.1 y analizada en detalle en el **apéndice C** es un elemento combinacional. Para un conjunto de entradas, siempre produce la misma salida porque no tiene almacenamiento interno.

**Elemento combinacional:** un elemento operacional tal como una puerta AND o una ALU.

Otros elementos en el diseño no son combinacionales, sino que contienen *estado*. Un elemento contiene estado si tiene almacenamiento interno. Estos elementos se denominan **elementos de estado** porque, si se apaga la máquina, se puede reiniciar cargando dichos elementos con los valores que contenían antes de apagarla. Además, si se guardan y se restauran, es como si la máquina no se hubiera apagado nunca. Así, los elementos de estado caracterizan completamente la máquina. En la figura 4.1, las memorias de instrucciones y datos, así como los registros, son ejemplos de elementos de estado.

Un elemento de estado tiene al menos dos entradas y una salida. Las entradas necesarias son el valor del dato a almacenar en el elemento de estado y el reloj, que determina cuándo se almacena el dato. La salida del elemento de estado proporciona el valor que se almacenó en un ciclo de reloj anterior. Por ejemplo, uno de los elementos de estado más simple es un biestable de tipo D (véase el  **apéndice C**), el cual tiene exactamente estas dos entradas (un dato y un reloj) y una salida. Además de los biestables, la implementación MIPS también utiliza otros dos tipos de elementos de estado: memorias y registros, que aparecen en la figura 4.1. El reloj se utiliza para determinar cuándo debería escribirse el elemento de estado. Un elemento de estado se puede leer en cualquier momento.

Los componentes lógicos que contienen estado también se denominan *secuenciales* porque sus salidas dependen tanto de sus entradas como del contenido de su estado interno. Por ejemplo, la salida de la unidad funcional que representa a los registros depende del identificador del registro y de lo que se haya almacenado en los registros anteriormente. La operación tanto de los elementos combinacionales como secuenciales, así como su construcción, se analiza con más detalle en el  **apéndice C**.

Se utiliza la palabra **activado** para indicar que una señal se encuentra lógicamente a alta y *activar* para especificar que una señal debe ser puesta a alta, mientras que *desactivar* o **desactivado** representan un valor lógico a baja.

### Metodología de sincronización

Una **metodología de sincronización** define cuándo pueden leerse y escribirse las diferentes señales. Es importante especificar la temporización de las lecturas y las escrituras porque, si una señal se escribe en el mismo instante en que es leída, el valor leído puede corresponder al valor antiguo, al valor nuevo o ¡incluso a una combinación de ambos! No es necesario decir que los diseños de computadores no pueden tolerar tal imprevisibilidad. La metodología de sincronización se diseña para prevenir esta circunstancia.

Por simplicidad, supondremos una metodología de **sincronización por flanco**. Esta metodología de sincronización implica que cualquier valor almacenado en un elemento lógico secuencial se actualiza únicamente en un flanco de reloj. Debido a que sólo los elementos de estado pueden almacenar datos, las entradas de cualquier lógica combinacional deben proceder de elementos de este tipo y las salidas también deben dirigirse hacia elementos de este tipo. Las entradas serán valores escritos en un ciclo anterior de reloj, mientras que las salidas son valores que pueden utilizarse en el ciclo siguiente.

**Elemento de estado:**  
elemento de memoria.

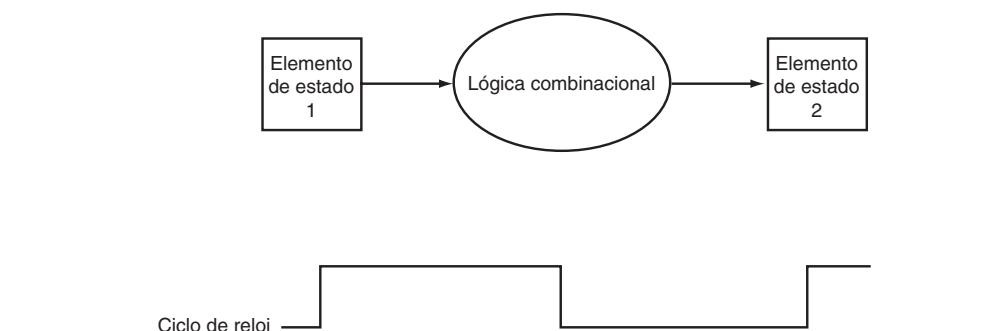
**Activado:** la señal está al nivel lógico alto o verdadero.

**Desactivado:** la señal está al nivel lógico bajo o falso.

**Metodología de sincronización:** aproximación que determina cuándo los datos son válidos y estables utilizando como referencia el reloj.

**Sincronización por flanco:** esquema de sincronización en el cual todos los cambios de estado se producen en los flancos de reloj.

La figura 4.3 muestra dos elementos de estado que rodean a un bloque de lógica combinacional, que opera con un único ciclo de reloj. Todas las señales deben propagarse desde el elemento de estado 1, a través de la lógica combinacional hasta el elemento de estado 2 en un único ciclo de reloj. El tiempo necesario para que las señales alcancen el elemento de estado 2 define la duración del ciclo de reloj.

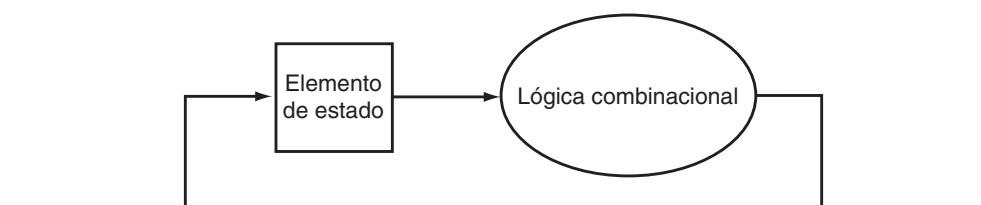


**FIGURA 4.3 La lógica combinacional, los elementos de estado y el reloj están estrechamente relacionados.** En un sistema digital síncrono, el reloj determina cuándo los elementos con estado van a escribir valores en el almacenamiento interno. Cualquiera de las entradas de un elemento de estado debe alcanzar un valor estable (es decir, alcanzar un valor que no va a cambiar hasta el siguiente flanco del reloj) antes de que el flanco de reloj activo cause la actualización del estado. Se supone que todos estos elementos, incluida la memoria, están sincronizados por flanco.

**Señal de control:** señal utilizada como selección en un multiplexor o para dirigir la operación de una unidad funcional; contrasta con una **señal de datos**, que contiene información sobre la que opera una unidad funcional.

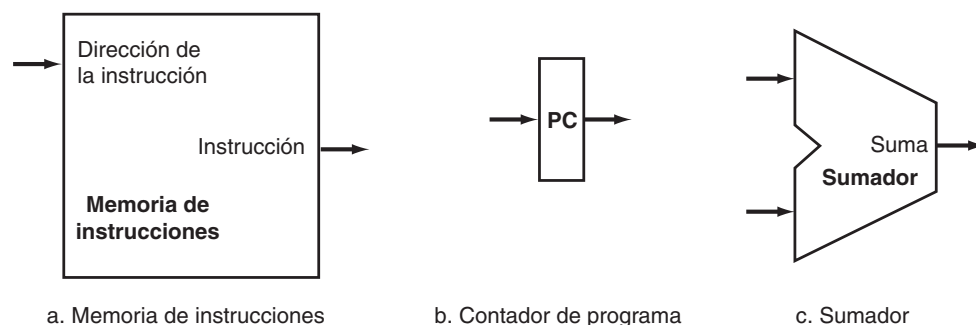
Por simplicidad, no se muestra una **señal de control** de escritura cuando se escribe en un elemento de estado en cada flanco activo de reloj. En cambio, si el elemento de estado no se actualiza en cada ciclo de reloj, es necesario incluir una señal de control de escritura (*write*). La señal de reloj y la señal de control de escritura son entradas, y el elemento de estado se actualiza únicamente cuando la señal de control de escritura se activa y ocurre un flanco de reloj.

La metodología por flanco permite leer el contenido de un registro, enviar el valor a través de alguna lógica combinacional, y escribir en ese mismo registro en un mismo ciclo de reloj, tal como se muestra en la figura 4.4. Es indiferente donde se asuma la implementación de las escrituras, ya sea en el flanco ascendente o en el descendente, ya que las entradas del bloque combinacional sólo pueden modifi-



**FIGURA 4.4 La metodología de sincronización por flanco permite leer y escribir en un elemento de estado en un mismo ciclo de reloj sin crear una condición de carrera que pueda conducir a valores indeterminados de los datos.** Por supuesto, el ciclo de reloj debe ser suficientemente largo para que los valores de entrada sean estables cuando ocurra el flanco de reloj activo. El baipás no puede darse dentro de un mismo ciclo debido a la actualización por flanco del elemento de estado. Si fuera posible el baipás, puede que este diseño no funcione correctamente. Los diseños de este capítulo como los del próximo se basan en este tipo de metodología de sincronización y en estructuras como la mostrada en la figura.





**FIGURA 4.5 Se necesitan dos elementos de estado para almacenar y acceder a las instrucciones, y un sumador para calcular la dirección de la instrucción siguiente.** Los elementos de estado son la memoria de instrucciones y el contador de programa. La memoria de instrucciones es solo de lectura, ya que el camino de datos nunca escribe instrucciones, y se trata como un elemento de lógica combinacional. La salida en cualquier instante refleja el contenido de la localización especificada por la dirección de entrada, y no se necesita ninguna señal de control de lectura (sólo se necesitará escribir en la memoria de las instrucciones cuando se cargue el programa; esto no es difícil de añadir, por lo que se ignora para simplificar). El contador de programa es un registro de 32 bits que se modifica al final de cada ciclo de reloj y, de esta manera, no necesita ninguna señal de control de escritura. El sumador es una ALU cableada para que sume siempre dos entradas de 32 bits y dé el resultado en su salida.

queta sumador como en la figura 4.5 para indicar que es un sumador permanente y que no puede realizar ninguna otra función propia de una ALU.

Para ejecutar cualquier instrucción se debe empezar por cargar la instrucción desde memoria. Para poder ejecutar la siguiente instrucción se debe incrementar el contador de programa para que apunte hacia ella 4 bytes más allá. La figura 4.6 muestra cómo se combinan los tres elementos de la figura 4.5 para formar un camino de datos que busca instrucciones e incrementa el PC para obtener la dirección de la siguiente instrucción secuencial.

Ahora considérense las instrucciones tipo R (véase la figura 2.20 de la página 136). Todas ellas leen dos registros, operan con la ALU los contenidos de dichos registros como operandos, y escriben el resultado. Estas instrucciones se denominan de tipo R o aritmético-lógicas (ya que realizan operaciones aritméticas o lógicas). En este tipo de instrucciones se incluyen las instrucciones `add`, `sub`, `and`, `or` y `sllt`, que fueron introducidas en el capítulo 2. Recuérdese también que el ejemplo típico de este tipo de instrucciones es `add $t1, $t2, $t3`, que lee `$t2` y `$t3` y escribe en `$t1`.


Los registros de 32 bits del procesador se agrupan en una estructura denominada **banco de registros**. Un banco de registros es una colección de registros donde cualquier registro puede leerse o escribirse especificando su número. El banco de registros contiene el estado de los registros de la máquina. Además, se necesita una ALU para poder operar con los valores leídos de los registros.

Debido a que las instrucciones tipo R tienen tres operandos registro, por cada instrucción se necesita leer dos datos del banco y escribir uno en él. Por cada registro que se lee, se necesita una entrada en el banco donde se especifique el número de registro que se quiere leer, así como una salida del banco donde se

#### Banco de registros:

elemento de estado que consiste en un conjunto de registros que pueden ser leídos y escritos proporcionando el identificador del registro al que se desea acceder.



carse en el flanco elegido. Con este tipo de metodología de sincronización, no existe baipás dentro de un mismo ciclo, y la lógica de la figura 4.4 funciona correctamente. En el  **apéndice C** se analizan brevemente otras limitaciones de la sincronización —como los tiempos de preestabilización (*setup times*) y de mantenimiento (*hold times*)— así como otras metodologías.

En la arquitectura MIPS de 32 bits, casi todos estos elementos lógicos y de estado tendrán entradas y salidas de 32 bits, ya que ésta es la anchura de muchos de los datos tratados por el procesador. Se marcará de alguna forma cuando una unidad tenga una entrada o una salida con una anchura diferente. Las figuras mostrarán los buses (señales con anchura superior a un bit) mediante líneas más gruesas. Cuando se quiera combinar varios buses para formar uno de anchura superior, por ejemplo, si se quiere tener un bus de 32 bits combinando dos de 16, las etiquetas de dichos buses especificarán que hay varios buses agrupados. También se añaden como ayuda flechas para destacar la dirección del flujo de datos entre los elementos. Finalmente, el **color** indicará si se trata de una señal de control o de datos. Esta distinción se especificará mejor más adelante en este capítulo.


Verdadero o falso: ya que el banco de registros es leído y escrito en el mismo ciclo de reloj, cualquier camino de datos MIPS que utilice escrituras por flanco debe tener más de una copia del banco de registros.

**Extensión:** Existe una versión de 64 bits de la arquitectura MIPS y, naturalmente, la mayor parte de los componentes tienen un ancho de 64 bits. Por otra parte, se usan los términos activado y desactivado porque a veces el 1 representa el estado lógico alto y otras veces el estado lógico bajo.

## 4.3

### Construcción de un camino de datos

Una forma razonable de empezar el diseño de un camino de datos es examinar los componentes principales necesarios para ejecutar cada tipo de instrucción del MIPS. Primero se consideran los **elementos del camino de datos** que necesita cada instrucción. Cuando se muestran los elementos del camino de datos, también se muestran sus señales de control.

La figura 4.5a muestra el primer elemento que se necesita: una unidad de memoria donde almacenar y suministrar las instrucciones a partir de una dirección. La figura 4.5b también muestra un registro, denominado **contador de programa (PC)**, que hemos visto en el capítulo 2 y se utiliza para almacenar la dirección de la instrucción actual. Finalmente se necesita un sumador encargado de incrementar el PC para que apunte a la dirección de la siguiente instrucción. Este sumador, que es combinacional, se puede construir a partir de la ALU descrita en detalle en el  **apéndice C**, haciendo simplemente que las líneas de control siempre especifiquen una operación de suma. Este sumador se representa como una ALU con la eti-

### Autoevaluación

**Elemento del camino de datos:** unidad funcional utilizada para operar o mantener un dato dentro de un procesador. En la implementación MIPS, los elementos del camino de datos incluyen las memorias de instrucciones y datos, el banco de registros, la unidad aritmético-lógica (ALU), y sumadores.

**Contador de programa (PC):** registro que contiene la dirección de la instrucción del programa que está siendo ejecutada.