4.5 Microprogramación para simplificar el diseño del control

Para el control de la implementación multiciclos del subconjunto MIPS considerado, una máquina de estados como la mostrada en la figura 4.28 es adecuada. Por que el subconjunto es pequeño y la cantidad de estados que se requiere puede representarse sin problema en una hoja de papel. Sin embargo, el conjunto completo de instrucciones MIPS comprende al rededor de 100 instrucciones y algunas de ellas requieren hasta de 20 ciclos de reloj para su ejecución, por lo que construir una máquina de estados para el conjunto completo resultaría bastante complejo. La unidad de control podría requerir fácilmente de mil estados con cientos de diferentes secuencias. Y la representación gráfica ya no sería fácil de construir y manejar.

Una situación similar ocurre a los programadores cuando alguna aplicación requiere de muchas líneas de código. Si todo el programa se concentra en la función principal, se tendrá un programa mal estructurado, difícil de entender y difícil de actualizar. En ese caso, los programadores organizan al código en módulos mas simples (funciones) y el módulo principal se encarga de coordinar a los demás módulos.

¿Será posible aplicar alguna técnica similar para el desarrollo del hardware? Podríamos tratar a las señales de control que serán acertadas en un estado como una instrucción que será ejecutada por el camino de los datos. Para evitar confusión entre las instrucciones MIPS y estas instrucciones de control a bajo nivel, en lo sucesivo las referiremos como *microinstrucciones*. Cada microinstrucción define el conjunto de señales de control que serán acertadas en el camino de los datos para un estado dado. Ejecutar una microinstrucción tiene el efecto de acertar las señales de control especificadas por la microinstrucción.

Además de definir a las señales que serán acertadas, se debe definir la secuencia, es decir, cual será la microinstrucción siguiente. En una máquina de estados finitos, el diagrama indica cual será la estado siguiente, en algunos casos la transición es incondicional, mientras que en otros depende del valor de las entradas. Al escribir programas, también ocurren situaciones análogas, en algunos casos las instrucciones se realizan secuencialmente mientras que en otros, se incluyen algunos brincos a otras instrucciones de acuerdo al valor de una variable. La ejecución por default es secuencial y los brincos deben ser indicados explícitamente.

La *microprogramación* consiste en definir el control como un programa que implementa las instrucciones de la máquina en términos de microinstrucciones mas simples. La idea clave es representar los valores acertados en las líneas de control simbólicamente, de manera que el microprograma es una representación de las microinstrucciones, así como el lenguaje ensamblador es una representación del lenguaje máquina. Al definir la sintaxis de un lenguaje ensamblador, se eligen los diferentes campos que formarán cada instrucción (opcode, registros, y desplazamientos o datos inmediatos); de la misma manera, la representación de las microinstrucciones incluirá un conjunto de campos cuya función este relacionada.

Definición del formato de una microinstrucción

El microprograma es una representación simbólica del control que será trasladado por un programa a la lógica de control. De esta manera, es posible elegir cuantos campos debería contener una microinstrucción y las señales de control que serán afectadas por cada campo. El formato de la microinstrucción debe elegirse buscando simplicidad en la representación y que facilite la escritura y comprensión de un microprograma. Por ejemplo, es útil contar con un campo que controle la ALU y un conjunto de tres campos que determinen las dos entradas de la ALU así como la ubicación del resultado de la operación. Además por legibilidad, se debe verificar que el formato de un microprograma haga difícil o imposible escribir microinstrucciones inconsistentes. Una microinstrucción es inconsistente si requiere que una señal de control dada sea ajustada a dos valores diferentes.

Para evitar formatos que permitan inconsistencias, los campos serás responsables de especificar conjuntos de señales de control que no se traslapen entre sí. Para elegir la partición de las señales de control es conveniente revisar la figura 4.21 en donde se mostró al camino de datos completo y al control como un bloque en el que se generan todas las señales, y la tabla 4.7 en la que se mostró el efecto para los diferentes valores de cada una de las señales.

Las señales que nunca son acertadas simultáneamente pueden compartir el mismo campo. En la tabla 4.9 se muestra como la microinstrucción puede dividirse en siete campos y define la función general de cada campo. Los primeros seis campos controlan al camino de datos, mientras que el campo de Secuencia (séptimo campo) indica como se seleccionará la siguiente microinstrucción.

Nombre	Función del campo
ALU Control	Especifica la operación que hará la ALU durante el ciclo de reloj actual, el resultado será escrito siempre en ALUOut.
SCR1	Especifica el origen del primer operando de la ALU.
SCR2	Especifica el origen del segundo operando de la ALU.
Register Control	Especifica la lectura o escritura para el archivo de registros, y el origen del dato para una escritura.
Memory	Especifica la lectura o escritura, y el origen para la memoria. Para una lectura, especifica el registro destino.
PCWrite Control	Especifica la escritura del PC.
Sequencing	Especifica la siguiente microinstrucción a ser ejecutada.

Tabla 4.9 Los siete campos que contendrá cada microinstrucción.

Las microinstrucciones serán puestas en una ROM o en un PLA, de manera que podemos asignarles direcciones a cada una de ellas. Las direcciones se elegirán secuencialmente, de la misma manera en que se eligen los números para los estados en una máquina de estados finitos. Se tiene tres posibilidades para elegir la siguiente microinstrucción a ejecutarse:

1. Incrementar la dirección de la microinstrucción actual para obtener la dirección de la siguiente microinstrucción. Este comportamiento secuencial será indicado en el microprograma con la etiqueta *Seq* en el campo de secuencia (*sequencing*). Dado que la ejecución secuencial se encuentra frecuentemente, muchos sistemas de microprogramación la tendrán por default.

2. Brincar a la microinstrucción que inicia la ejecución de la siguiente instrucción MIPS. A esta microinstrucción inicial la etiquetaremos como *Fetch* (corresponde al estado 0) y colocaremos el indicador *Fetch* en el campo de secuencia para indicar esta acción.

3. Elegir la siguiente microinstrucción con base en la entrada a la unidad de control. La elección de la siguiente instrucción se conoce como un *despacho (dispatch)*. Los despachos usualmente son implementados creando una tabla que contenga la dirección de la microinstrucción destino. Esta tabla es indexada por la entrada a la unidad de control y puede ser implementada en una ROM o en un PLA. Frecuentemente hay múltiples tablas de despacho, en esta implementación se emplearán dos tablas, una para despachar desde el estado 1 (cuando se determina el tipo de instrucción) y otra para despachar desde el estado 2 (para los accesos a memoria, cuando se determina si se trata de una carga o almacenamiento). Indicaremos que la siguiente microinstrucción se debe seleccionar por una operación de despacho colocando *Dispatch i*, donde *i* es el número de tabla de despacho en el campo de Secuencia.

La tabla 4.10 da una descripción de los valores permitidos para cada campo de la microinstrucción y el efecto de los diferentes valores. Recordemos que el microprograma es una representación simbólica. Este formato de microinstrucciones es solo un ejemplo de muchos formatos poderosos.

Creación del microprograma

Ahora crearemos el programa para la unidad de control. Etiquetaremos las instrucciones en el microprograma con etiquetas simbólicas, las cuales pueden ser usadas para especificar el contenido de las tablas de despacho. En la escritura del microprograma, hay dos situaciones en las cuales requeriríamos dejar un campo de la microinstrucción vacío. Cuando un campo que controla una unidad funcional o que produce la escritura de un elemento de estado esta vacío, las señales de control no deberán ser acertadas. Cuando un campo sólo especifica el control de un multiplexor que determina la entrada a una unidad funcional es dejado en blanco, significa que no importa el dato que llegará a la unidad funcional (o la salida del multiplexor).

La forma mas fácil de entender el microprograma es partirlo en piezas, buscando que cada componente trate con la ejecución de una instrucción, así como cuando se diseño la máquina de estados finitos.

Nombre	Valores para el campo	Función del campo con valores específicos
Etiqueta	Alguna Cadena	Usado para especificar etiquetas para controlar la secuencia de micro código. Las etiquetas que terminan con 1 o 2 son destinos de los despachos obtenidos de una tabla de símbolos que es indexada con el opcode. Otras etiquetas son usadas como destino directo en la secuencia de microinstrucciones. Las etiquetas no generan señales de control, son usadas para representar el contenido de una tabla de despachos y determinar el comportamiento del campo de secuencias.
ALU Control	Add	Produce una suma en la ALU
	Subt	Produce una resta en la ALU
	Func code	Usa el campo de función de la instrucción para determinar el control de la ALU
SCR1	PC	Usa al PC como primera entrada de la ALU
	A	El registro A es la primera entrada de la ALU
SCR2	B	El registro B es la segunda entrada de la ALU
	4	El 4 es la segunda entrada de la ALU
	Extend	La salida de la Unidad de Extensión de signo es la segunda entrada de la ALU
	Extrshft	Después del desplazamiento en 2 se obtiene la segunda entrada de la ALU
Register Control	Read	Lee dos registros usando los campos Rs y Rt de IR como los números de registros, poniendo los datos leídos en los registros A y B
	Write ALU	Escribe el archivo de registros usando el campo Rd como el número de registro y el contenido de ALUOut como el dato
	Write MDR	Escribe el archivo de registros usando el campo Rt como el número de registro y el contenido de MDR como el dato
Memory	Read PC	Lee la memoria usando al PC como dirección, escribe el resultado en IR (y en MDR)
	Read ALU	Lee la memoria usando ALUOut como dirección, escribe el resultado en MDR
	Write ALU	Escribe la memoria usando ALUOut como dirección y el contenido de B como el dato
PCWrite Control	ALU	Escribe la salida de la ALU en el PC
	ALUOut-cond	Si la salida Zero de la ALU es activa, escribe el PC con el contenido del registro ALUOut
	Jump address	Escribe el PC con la dirección de salto desde la instrucción
Sequencing	Seq	Elige la siguiente microinstrucción secuencialmente
	Fetch	Va a la primera microinstrucción a iniciar una nueva instrucción
	Dispatch i	Despacha usando la ROM especificada en i (1 o 2)

Tabla 4.10 Valores permitidos para cada campo de la microinstrucción y su efecto.

La primer componente en la ejecución de todas las instrucciones consiste en la captura y decodificación de la instrucción, así como el cálculo secuencial del PC y el destino de un brinco. Estas acciones corresponden directamente a los dos pasos de ejecución descritos con anterioridad. Las dos microinstrucciones necesarias para estos pasos son:

Etiqueta	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1

- Con la primera microinstrucción, la ALU suma el valor de PC con 4 y la salida de la ALU se escribe en el mismo PC. Se lee la memoria con la dirección proporcionada por el PC atrapando la instrucción en el registro IR y se continua con la siguiente microinstrucción.

- En la segunda microinstrucción, la ALU suma al PC con el dato inmediato tomado del registro IR, extendido en signo y desplazado a la izquierda en dos. Se leen los registros indicados en los campos Rs y Rt del registro IR y sus valores se escriben en los registros A y B. Se continua con la instrucción especificada en la ROM de despacho 1.

Podemos pensar en la ROM de despacho como una expresión *switch-case* con el opcode como llave y los datos de la tabla usados para seleccionar una de las cuatro diferentes secuencias de microinstrucciones, las cuales etiquetaremos como:

- *Mem1* para instrucciones de accesos a memoria
- *Rformat1* para instrucciones tipo R
- *BEQ1* para brincos sobre igual
- *JUMP1* para la instrucción de salto

Todas las etiquetas finalizan en 1 para indicar que corresponden a la tabla de despacho 1.

El microcódigo para los accesos a memoria tiene 4 microinstrucciones, éstas se muestran a continuación:

Etiqueta	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch

- La primera microinstrucción calcula la dirección del dato que será accesado sumando el registro A con la constante de 16 bits tomada del registro IR extendida en signo. Para seleccionar la siguiente microinstrucción se utiliza la ROM de despacho 2, la cual

contiene solo dos elementos, uno para cuando se tratan de cargas o el otro para desplazamientos.

- La segunda microinstrucción es propia de las cargas, se lee de la memoria un dato diseccionándolo con el registro ALUOut y escribiendo el dato en el registro MDR. Se continua con la siguiente microinstrucción.

- La tercer microinstrucción completa una carga, se escribe en el archivo de registro el dato que está en el registro MDR y en el registro indicado en el campo Rt. Se continua con la microinstrucción etiquetada con Fetch, para iniciar con una nueva instrucción.

- La cuarta microinstrucción culmina con un almacenamiento, escribe la memoria usando ALUOut como dirección y el contenido de B como dato. Y también continua en la microinstrucción etiquetada como Fetch.

Para las instrucciones tipo R se realizan las dos microinstrucciones siguientes:

Etiqueta	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch

- La primera microinstrucción hace que la ALU aplique sobre los registros A y B la operación especificada en el campo de función. Continúa con la siguiente microinstrucción.

- Con la segunda microinstrucción se escribe en el archivo de registros el resultado que está en ALUOut, en el registro indicado en el campo Rd del registro Rs. Y se continua en la microinstrucción etiquetada como Fetch.

Para los brincos (BEQ), puesto que ya se calculó la dirección destino, para culminar con un brinco solo queda pendiente una microinstrucción, la cual es:

Etiqueta	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
BEQ1	Subt	A	B			ALUOut-cond	Fetch

- Con esta microinstrucción la ALU resta los operandos A y B para generar la bandera Zero. En el PC se escribirá el contenido de ALUOut si la bandera Zero fue generada, y continua en la microinstrucción etiquetada como Fetch.

Para los saltos incondicionales (J) queda pendiente una microinstrucción:

Etiqueta	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
JUMP1						Jump addres	Fetch

- Con esta microinstrucción el PC es sustituido con la dirección destino del salto, y se continúa con la microinstrucción etiquetada como Fetch.

En la tabla 4.11 se muestra el microprograma completo con las 10 microinstrucciones descritas anteriormente. El microprograma coincide con la máquina de estados finitos diseñada previamente, puesto que ambas se implementaron con base en los cinco pasos en los que se dividió la ejecución de las instrucciones.

Etiqueta	ALU Control	SRC1	SRC2	Register Control	Memory	PCWrite Control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump addres	Fetch

Tabla 4.11 El microprograma para la unidad de control.

Implementación del microprograma

Trasladar el microprograma en hardware involucra dos aspectos: Decidir como implementar la función de secuencia y elegir un método de almacenamiento de la función principal del control. El microprograma puede ser pensado como una representación textual de una máquina de estados finitos para implementarse en un PLA en el que se codifique tanto la secuencia como el control principal. Sin embargo, para microprogramas grandes, la función de secuencia y el control principal se deben implementar de manera diferente.

La forma alternativa de implementación involucra almacenar la función del control principal en una ROM e implementar la función de secuencia separadamente. En la figura 4.29 (a) se muestra esta forma diferente para implementar la función de secuencia. Se usa un mecanismo de incremento para elegir la siguiente instrucción del control. De manera que el microcódigo almacenado determina el valor de las líneas de control en el camino de datos, así como la forma de seleccionar la siguiente microinstrucción. La lógica de selección de direcciones deberá contener las tablas de despacho en memorias ROM, y dependiendo del control de secuencia determinará si la siguiente microinstrucción a

ejecutarse es la inmediata al contador del microprograma, o bien si debe accesar a alguna tabla para tomar la dirección de la siguiente microinstrucción. En caso de que se accese a una tabla, se deberá considerar al opcode para saber que dato tomar de la tabla, estos detalles se muestran en la figura 4.29 (b).

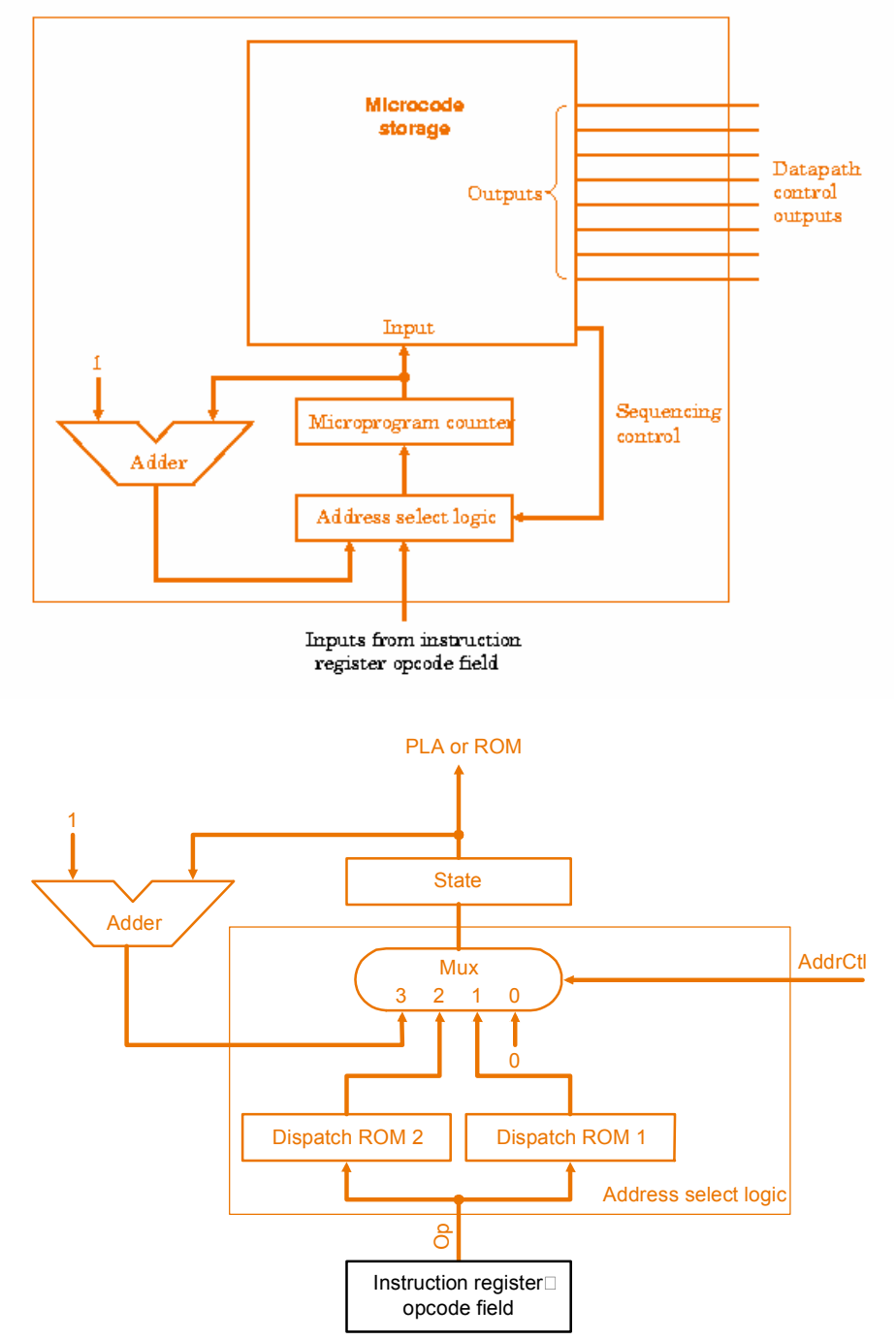

Figura 4.29 (a) Una forma típica de implementación de un microprograma.
(b) Lógica para la selección de direcciones.

4.6 Excepciones

Una de las partes mas duras del control es la implementación de *excepciones* e *interrupciones* – eventos que cambian el flujo normal de la ejecución de instrucciones. Una excepción es un evento inesperado desde dentro del procesador, un sobreflujo aritmético es un ejemplo de una excepción. Una interrupción es un evento que también cambia el flujo normal de la ejecución pero viene desde fuera del procesador. Las interrupciones son usadas por los dispositivos de entrada y salida para comunicarse con el procesador.

Muchas arquitecturas y autores no distinguen entre excepciones e interrupciones, frecuentemente con el término interrupción se refieren a ambos tipos de eventos. La arquitectura MIPS utiliza al término excepción para referir a cualquier cambio inesperado en el flujo de la ejecución de un programa, sin distinguir si la causa es interna o externa. La arquitectura Intel 80x86 usa la palabra interrupción para todos estos eventos, mientras que la arquitectura PowerPC usa la palabra excepción para un evento inusual e interrupción para indicar un cambio en el control de flujo.

En este documento solo consideraremos dos posibles excepciones, aquellas que ocurren cuando se presenta un sobreflujo y las que se originan por una instrucción desconocida. La acción básica que la máquina debe realizar cuando ocurre una excepción es respaldar la dirección de la instrucción afectada en el contador del programa de excepciones (EPC) y después transferir el control del sistema operativo a alguna dirección específica.

El sistema operativo puede entonces tomar la acción apropiada, la cual puede involucrar proporcionar algún servicio al usuario del programa, tomar una acción preestablecida ante la presencia de un sobreflujo o detener la ejecución del programa y reportar un error. Después de realizar cualquier acción requerida por la excepción, el sistema operativo puede terminar con la ejecución del programa o continuar su ejecución utilizando al registro EPC para determinar en donde reiniciar.

Para que el sistema operativo pueda manejar la excepción, debe de conocer la causa de su origen, hay dos métodos para conocer la causa de una excepción; el primero es utilizado por MIPS y consiste en la inclusión de un registro en el que se escriba la razón de la excepción (llamado *Cause register*).

El segundo método consiste en el uso de un *vector de interrupciones*. El vector de interrupciones contiene un conjunto predefinido de direcciones a las cuales será transferido el control dependiendo de la causa de la interrupción. Por ejemplo, si sólo se tiene dos causas posibles, el vector contendrá dos direcciones para atender a cada una de ellas.

En el caso de MIPS, se le agregaron dos registros al camino de datos para el manejo de excepciones:

- EPC : Un registro de 32 bits para mantener la dirección de la instrucción afectada (tal registro es necesario aún con interrupciones vectorizadas).

- Cause : Un registro usado para registrar la causa de la excepción. También es de 32 bits, aunque actualmente no todas las combinaciones son usadas. En nuestro caso, como ya se había mencionado, sólo se considerarán dos posibles excepciones: por una instrucción indefinida (causa = 0) y por sobreflujo aritmético (causa = 1).

Además de estos registros, es necesario agregar un multiplexor con las diferentes causas (0 o 1), así como extender el multiplexor cuya salida va hacía el PC, para darle al PC la posibilidad de tomar el valor de la dirección en la que se atenderá a las excepciones, supondremos que esta dirección corresponde a la $C0000000_{hex}$. También es necesario agregar algunas señales de control, las señales de escritura de los registro EPC y Cause (*EPCWrite* y *CauseWrite,* respectivamente), y la señal para la selección de la causa (*IntCause*).

Dado que el PC es incrementado durante el primer ciclo de cada instrucción, no es posible simplemente escribir el valor del PC en EPC por que se hará referencia a la instrucción siguiente. Sin embargo es posible usar la ALU para restar 4 al PC y escribir el resultado en el registro EPC. Esto no requiere hardware adicional, ya que es posible seleccionar como primer operando de la ALU al PC, como segundo a la constante 4 y hacer una resta con la ALU. En la figura 4.30 se muestra el camino de datos con el hardware agregado para el manejo de excepciones.

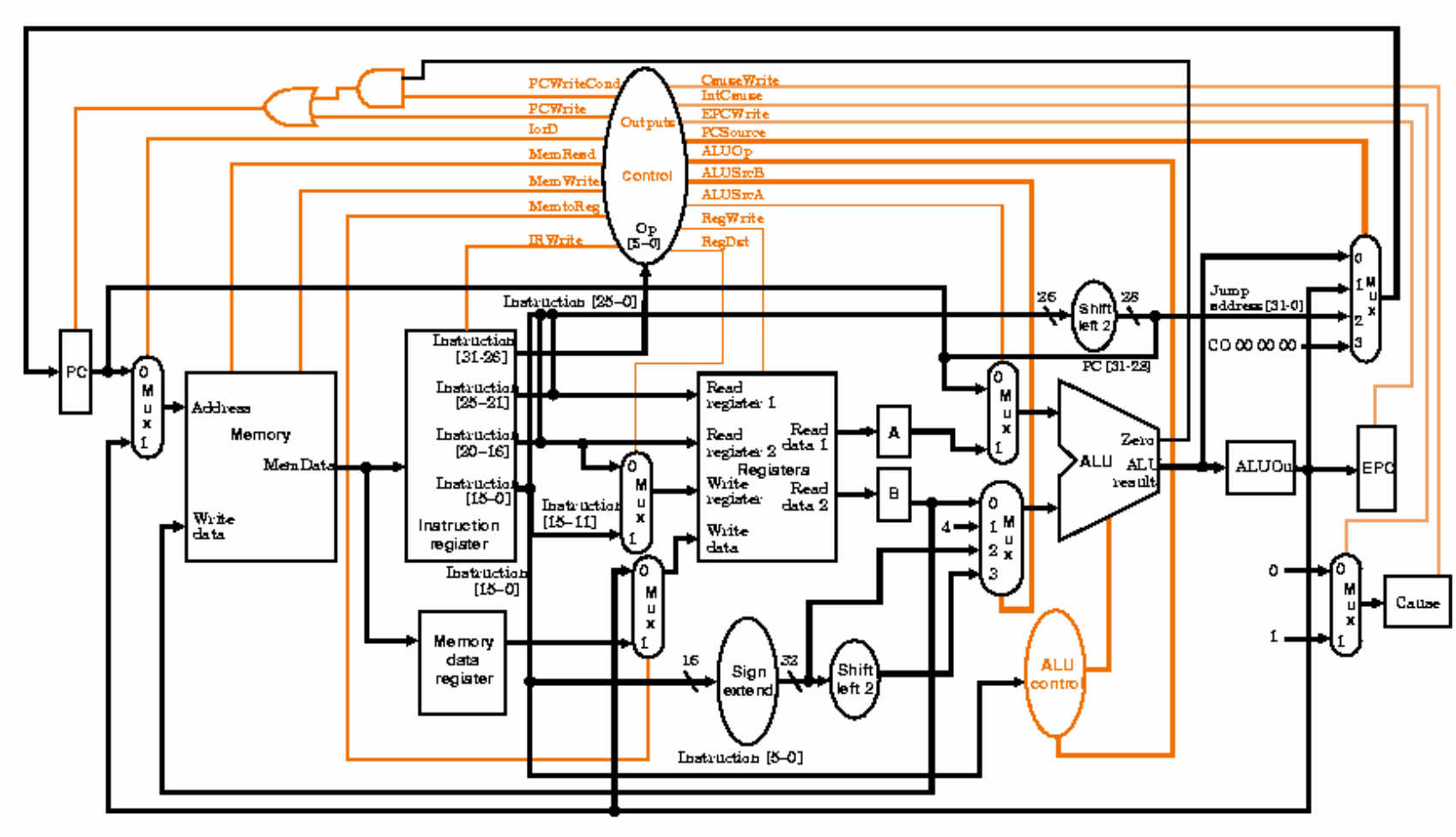

Figura 4.30 El camino de datos multiciclo con los elementos agregados para el manejo de excepciones.

Para el manejo de las excepciones será necesario agregar algunos estados al control, sin embargo, para las dos excepciones bajo consideración, debe restaurarse el valor del PC para escribirse en EPC.

En la figura 4.31 se muestran los estados que se agregarán, el estado 10 corresponde a una instrucción desconocida y en él se escribe el valor 0 en el registro *cause*, se calcula la dirección de la instrucción que produjo la excepción (restándole 4 al PC) para escribirse en EPC y se selecciona el valor de $C0000000_{hex}$ para el PC. El estado 11 corresponde a la generación de un sobreflujo y las acciones que se toman son similares al estado 10, excepto por que ahora se trata de la causa 1.

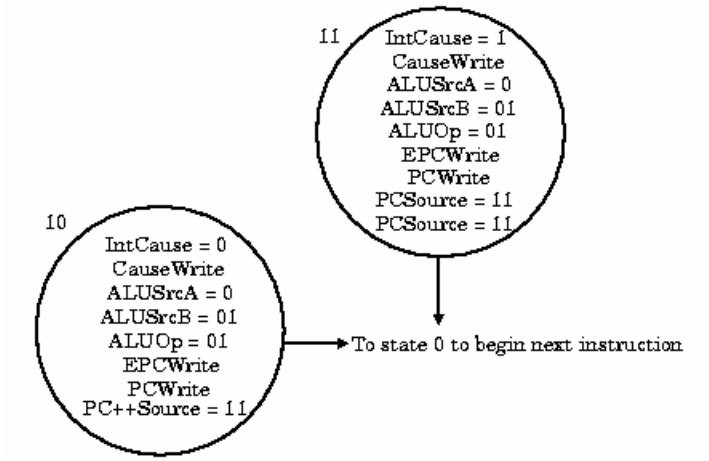

Figura 4.31 Par de estados que maneja las dos diferentes excepciones bajo consideración.

Cuando se presenta una instrucción desconocida, si al realizar la decodificación de la instrucción en el estado 1, resulta que el opcode no coincide con las que instrucciones implementadas, el estado siguiente será el 10.

Después del estado 10 el siguiente estado será el 0, para iniciar con la nueva instrucción; aunque debe aclararse que ésta instrucción ya es parte del código dedicado a la atención de interrupciones.

Los sobreflujos se presentan en operaciones aritméticas, por lo que después del estado 7, que corresponde al estado en que se completa una instrucción tipo-R, si se detecta la presencia de un sobreflujo, el estado siguiente será el estado 11. Y similar al estado 10, el estado que sigue al estado 11 es el estado 0.

En la figura 4.32 se muestra la máquina de estados con la que se implemetará el control de una implementación multiciclos con la inclusión de algunos estados para el manejo de excepciones.

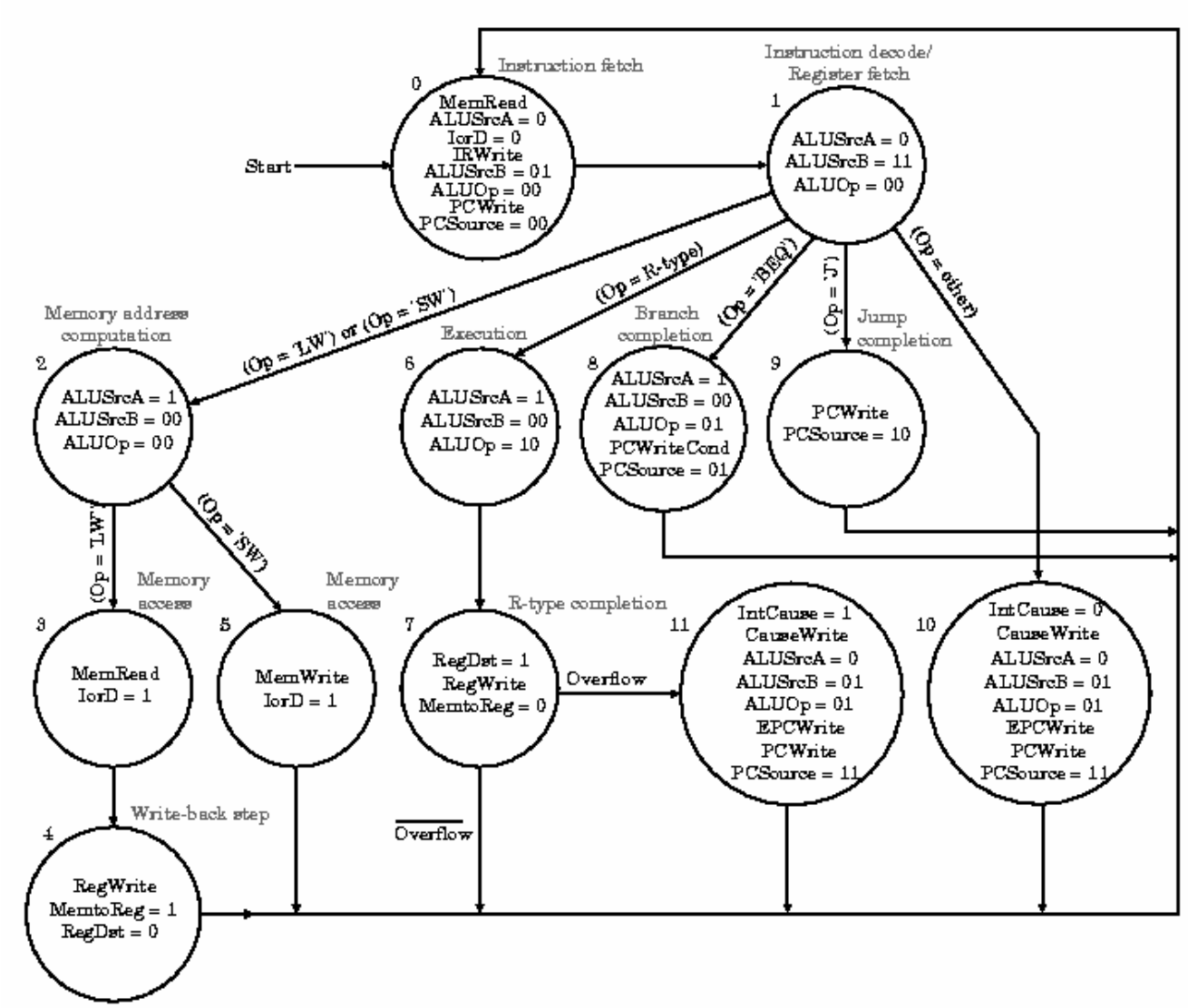

Figura 4.32 Máquina de estados con los nuevos estados para el manejo de excepciones.

Tarea 10

1. ¿Qué ventajas tendría el uso de microprogramación para el diseño del control con respecto a una máquina de estados finitos?

2. ¿Se tendrían desventajas? Explique.

3. ¿Cómo se modifica la tabla 4.11 al considerar la ejecución de las instrucciones **ADDI** y **JAL** de acuerdo con sus resultados de la tarea anterior?

4. ¿Por qué es importante que los procesadores soporten el manejo de excepciones?