Tema 10: Diseño de un CPU multiciclo y Microprogramación

Arquitectura de Computadoras

Ing. Nicolás Majorel Padilla (npadilla@herrera.unt.edu.ar)

http://microprocesadores.unt.edu.ar/arqcom/

Temas que veremos

- Ideas básicas de diseño multiciclo.
- Camino de datos.
- Unidad de Control.
  - Diseño mediante una MEF.
  - Implementación mediante ROM.
  - Implementación de MEF con secuenciador.
  - ROM de microcódigo (firmware).
- Implicancias y conclusiones.
- Manejo de interrupciones/excepciones.

Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021):
  - Sección 4.5: *A Multicycle Implementation*
  - Sección C.3: *Implementing Finite-State Machine Control*
  - Sección C.4: *Implementing the Next-State Function with a Sequencer*
  - Sección C.5: *Translating a Microprogram to Hardware*
  - Sección C.6: *Concluding Remarks*

Recapitulación

- Empezamos diseñando un procesador de ciclo único.
  - Control simple, combinacional.
  - Camino de datos sin escatimar recursos. Alto costo.
  - Duración del ciclo larga, determinado por la instrucción más larga.
    - Castigamos a todas por culpa de la más lenta.
- Luego hicimos un diseño en pipeline.
  - Para mejorar la performance.
  - Conocimos los riesgos, que nos limitan esa mejora.
- Después vimos muchas técnicas para aumentar al máximo posible la performance.
  - Siempre agregando complejidad y aumentando el costo.

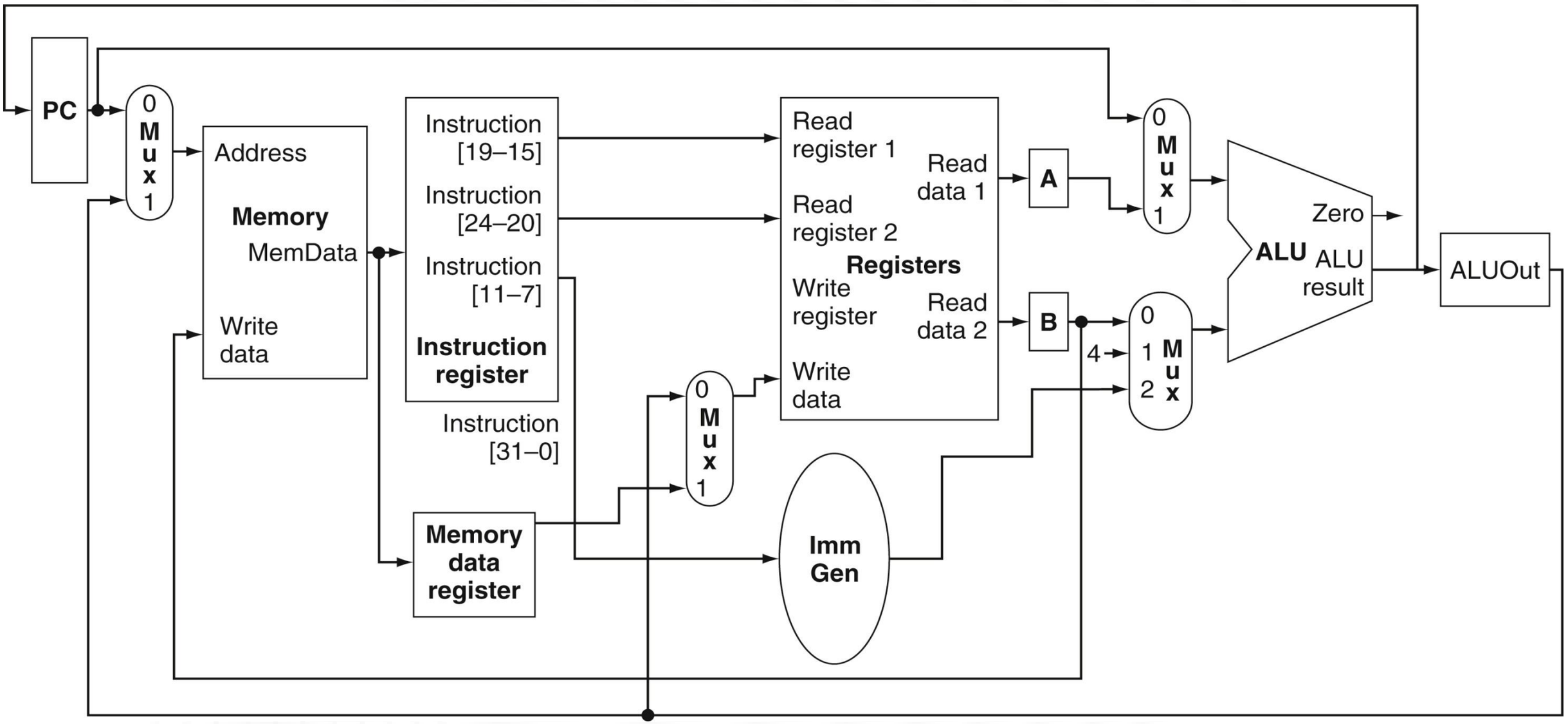
Recapitulación

- *¿Y si no todo fuera una cuestión de performance?*
- Algunos diseños de procesadores no buscan maximizar la performance.
  - Pueden buscar minimizar el costo, o el consumo de energía.
    - Por ejemplo, en sistemas embebidos.
  - Objetivos distintos necesitan criterios de diseño distintos.
    - Y soluciones diferentes.

Multiciclo – Idea básica

- Dividir las instrucciones en múltiples “pasos” modulares.
  - Igual que hicimos con el diseño en pipeline.
- Cualquier instrucción se forma mediante la **combinación** de estos pasos.
  - En **distinta cantidad de ciclos**, según la instrucción.
- Cada “paso” se ejecuta en un ciclo T pequeño.
  - Balanceado, similar al diseño en pipeline.
  - En cada “paso” se utiliza sólo una unidad funcional.
  - La duración del ciclo T dependerá de la duración del “paso” más lento.
- **Reutilización de recursos** para minimizar costo.
  - Una única ALU para operaciones Tipo R, para cálculo de la dirección, y para actualizar PC.
  - Una única unidad de memoria para datos e instrucciones.
- Necesita de registros adicionales.
  - Para almacenar valores intermedios a utilizarse en el ciclo siguiente.

Camino de datos



- Se agregan 5 registros (IR, MDR, A, B, ALUOut) y 2 mux y se expande otro mux.
- Se quitan 2 sumadores y una memoria.
- No contempla los saltos (todavía).

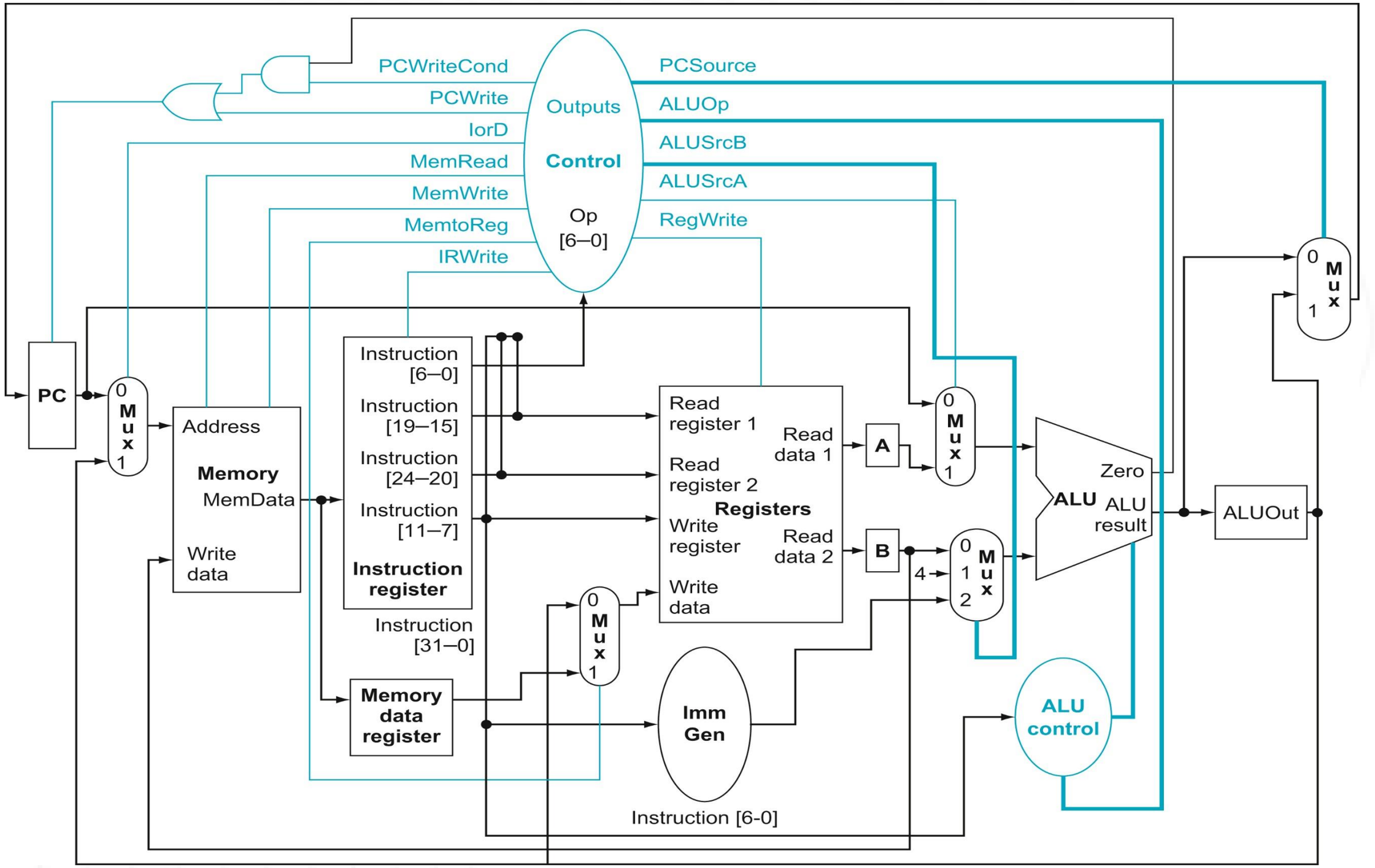
Pasos de ejecución de las instrucciones

- ¡Los mismos que vimos en pipeline! IF, ID, EX, ME y WB.
- Pero ahora cada instrucción **utiliza los que necesita**.
  - Demorarán entre 3 y 5 ciclos en ejecutarse.
  - Los 2 primeros pasos son comunes a todas las instrucciones.
- Describimos los pasos utilizando RTL:

Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	IR <= Memory[PC] PC <= PC + 4		
Instruction decode/register fetch	A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate		
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	
Memory read completion		Load: Reg[IR[11:7]] <= MDR	

- Algunos pasos pueden utilizar dos unidades funcionales, pero lo hacen en paralelo y no secuencialmente.

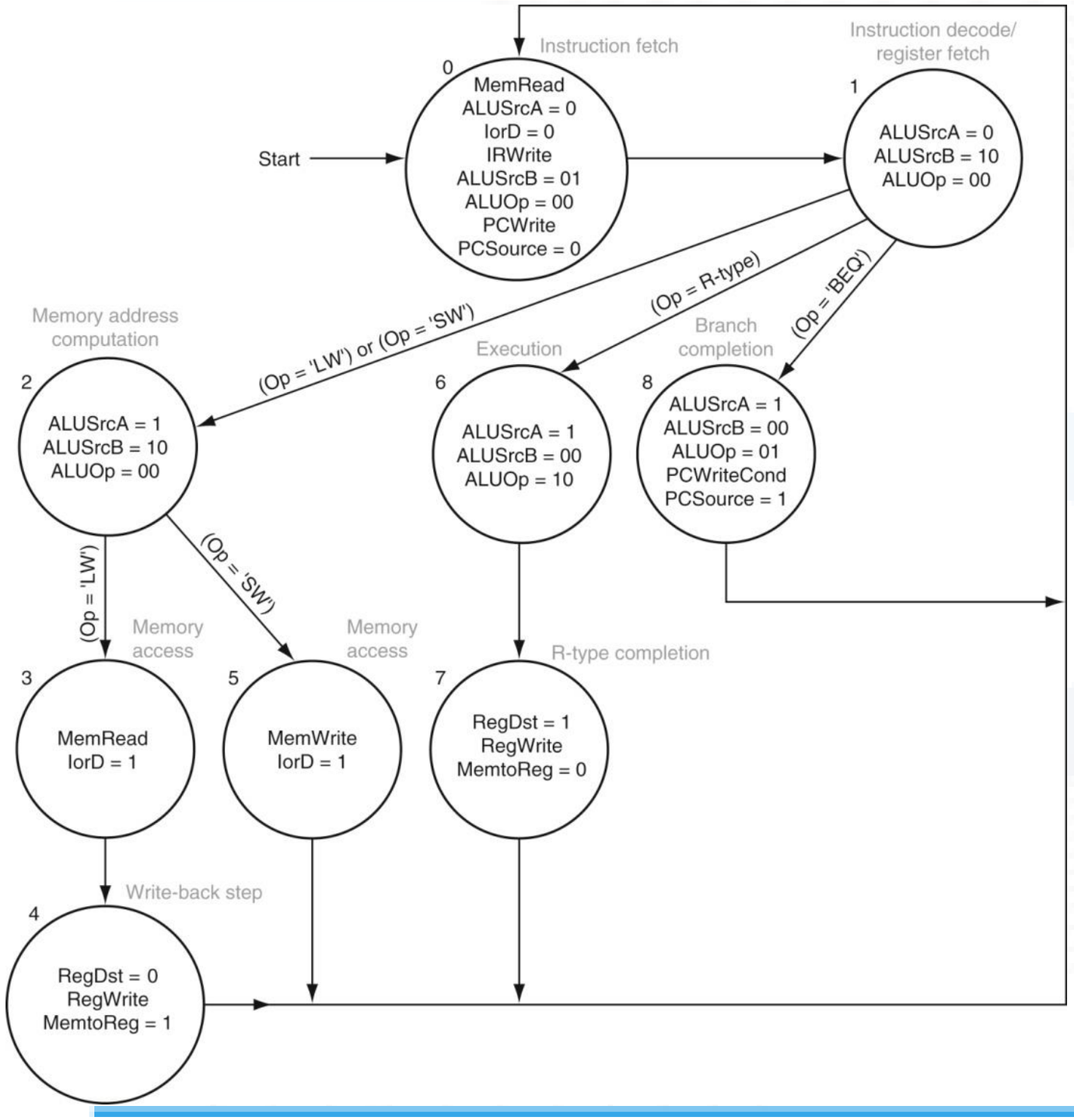
Camino de datos multiciclo completo



Diseño del Control

- Se agregaron nuevos multiplexores, que requieren nuevas señales de control.
- En este caso, las señales de control no dependen únicamente de la instrucción.
  - Por ejemplo, *¿qué debe hacer la ALU para una instrucción ADD?*
    - La respuesta sería otra pregunta: *¿en qué ciclo?*
- Por lo tanto, **el control es secuencial** y no combinacional.
  - Suelen implementarse mediante una Máquina de Estados Finitos (MEF).

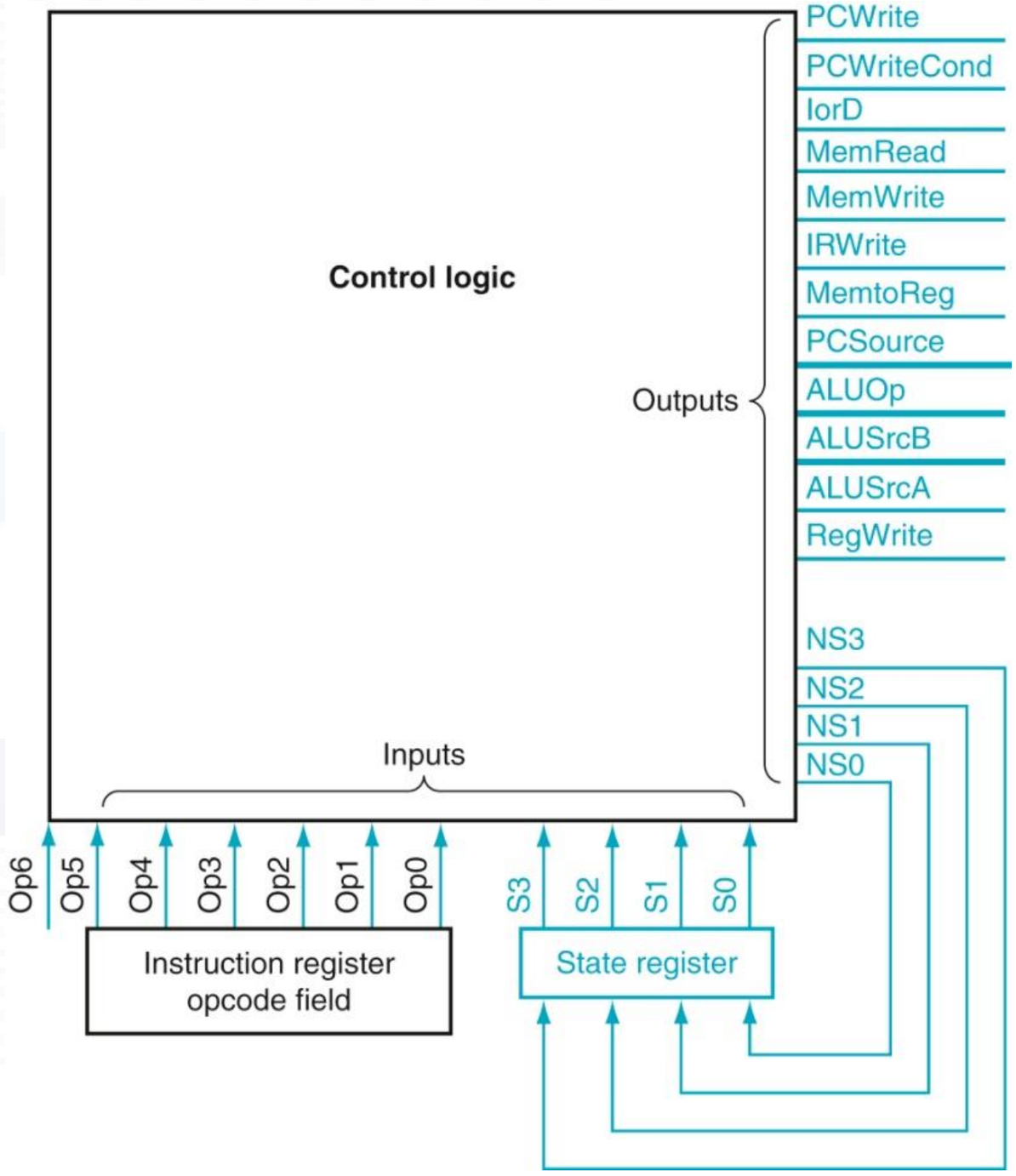
Diagrama MEF de Control



- Para nuestro ejemplo, la MEF tiene 9 estados.
  - Uno por cada “paso” diferente del RTL.
- En cada estado, **las salidas dependen únicamente del mismo estado**.
  - MEF estilo Moore.
- Al terminar la ejecución de una instrucción, se vuelve al estado inicial (IF).

Implementación tradicional de una MEF

- Recibe como entradas los bits del opcode y los del estado actual.
  - Para codificar 9 estados se necesitan 4 bits.
  - Y un registro que indique el estado actual.
- La lógica de control determina las señales para el camino de datos y para el próximo estado.



Transición de estados

- Observando la MEF, notamos que tres estados pasan **siempre** al siguiente.
- Cuatro estados vuelven **siempre** al 0.
- Y solamente dos estados tienen bifurcaciones que dependen de la instrucción.
- Podemos construir una tabla como la siguiente:

Output	Current states	Op
NextState0	state4 + state5 + state7 + state8	
NextState1	state0	
NextState2	state1	(Op = 'lw') + (Op = 'sw')
NextState3	state2	(Op = 'lw')
NextState4	state3	
NextState5	state2	(Op = 'sw')
NextState6	state1	(Op = 'R-type')
NextState7	state6	
NextState8	state1	(Op = 'beq')

Implementación con una ROM

- Posiblemente sea la implementación más simple.
- Entradas: 7 bits de opcode + 4 para el estado = 11 en total.
  - Permite hasta 2^{11} = 2048 direcciones diferentes.
  - Cada dirección representa un estado de la MEF.
- Salidas: 16 para el camino de datos + 4 para el estado = 20 en total.
- Tamaño total: $2^{11} * 20$ = 40Kbits
  - Tamaño poco usual.
  - No tiene en cuenta las condiciones de indiferencia.
  - Si tuviésemos más instrucciones, aumentan las salidas.

Implementación con ROM mejorada

- Como se dijo anteriormente, la MEF es de tipo Moore.
  - Las señales de salida dependen únicamente del estado actual.
- Es posible dividir la ROM anterior en dos:
  - ROM1: 4 bits de entrada producen 16 bits de salida.
    - El estado actual genera las señales de control.
  - ROM2: 11 bits de entrada producen 4 bits de salida.
    - El estado actual + el opcode determinan el estado siguiente.
  - Tamaño total = $2^4*16+2^{11}*4$ = 8,25 Kbits.
  - Mucho más eficiente.

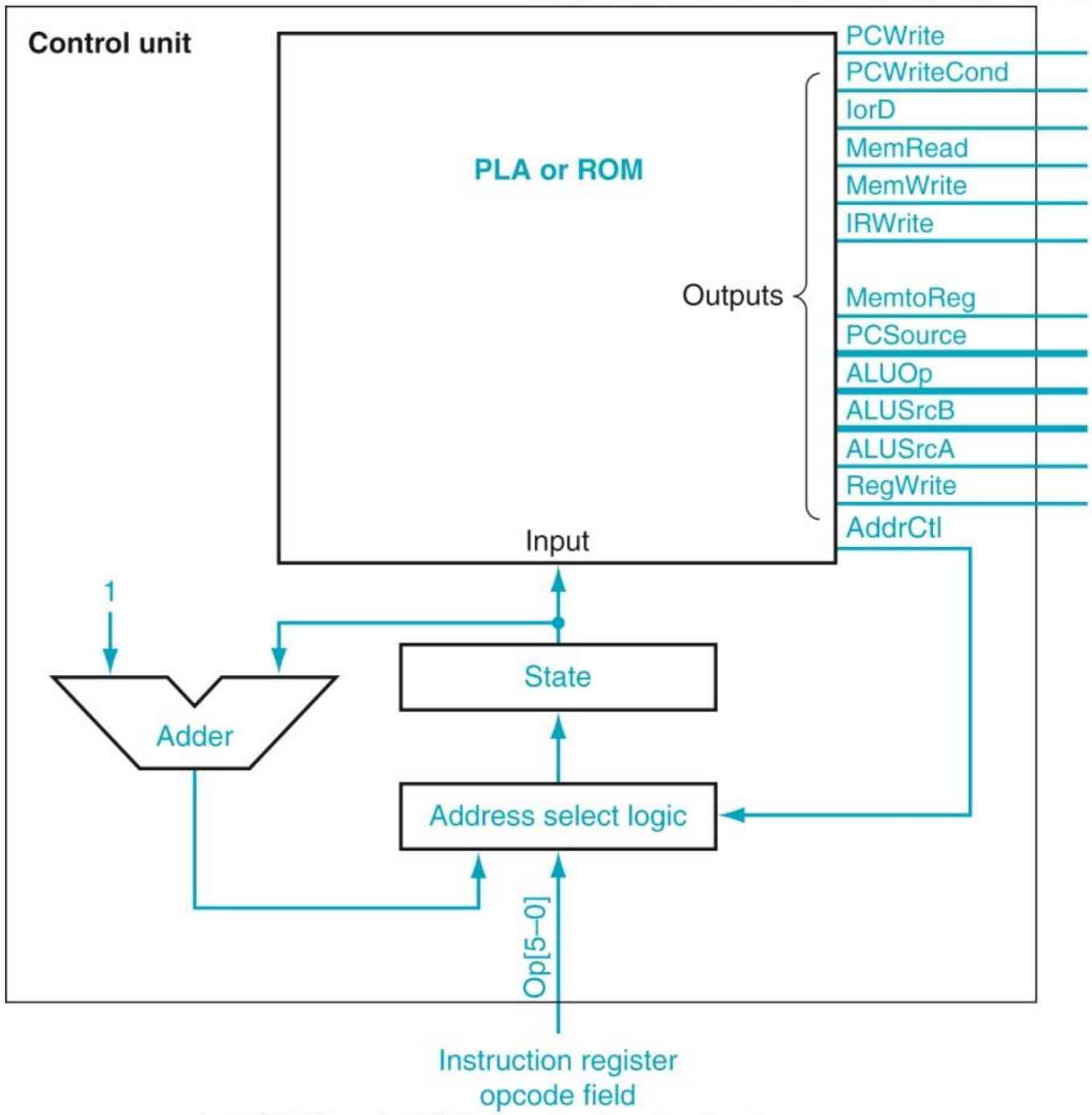
Implementación con ROM mejorada

- Sin embargo, sigue teniendo múltiples direcciones que no se necesitan, o redundantes.
  - Por ejemplo, habrá 128 direcciones que indiquen que del estado 0 se pasa al estado 1.
- ROM2 contiene las señales para determinar el próximo estado.
  - Consume la mayor cantidad de celdas (casi 97% del total).
  - Sin embargo, en general, las transiciones de estado son simples:
    - Mayoría de veces se pasa al siguiente, o al inicial.

Microprogramación – Idea principal

- A las salidas de la ROM1 las llamamos **microinstrucciones**.
  - Son las señales de control para el camino de datos.
  - Cada estado de la MEF representa una microinstrucción.
- A la ROM2 la reemplazamos por un circuito denominado **secuenciador**:
  - Un registro que mantiene el estado actual.
  - Un sumador que incremente en 1 el estado.
  - Una lógica para determinar el próximo estado.
    - Las señales de control para esta lógica deben estar en la microinstrucción.

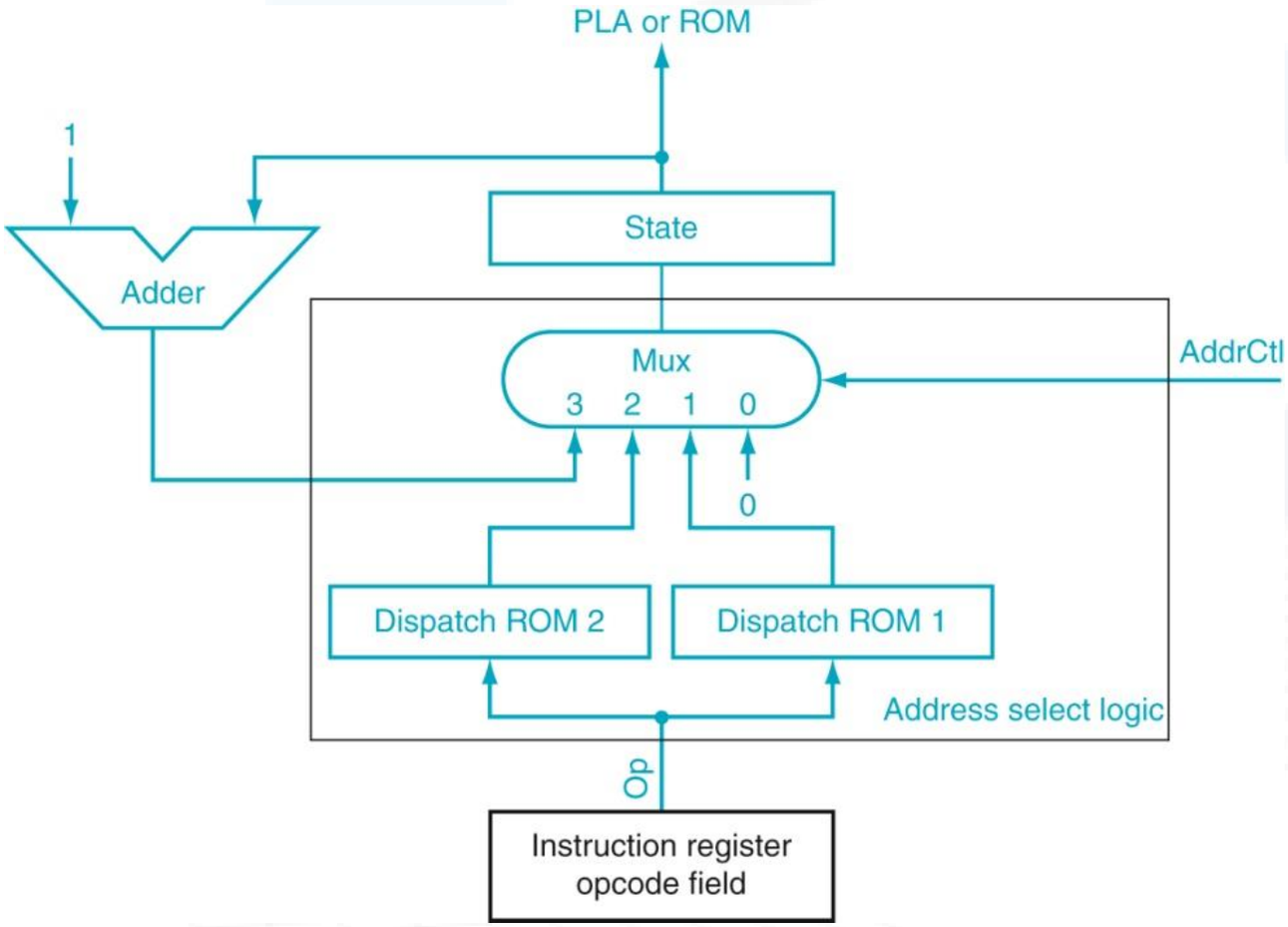
Implementación de MEF con Secuenciador



- La lógica para determinar el próximo estado:
  - Los estados 0, 3 y 6 pasan al siguiente (+1).
  - Los estados 4, 5, 7 y 8 vuelven al estado 0.
  - Del estado 1 se puede pasar al 2, 6 u 8 (dependiendo del opcode).
  - Del estado 2 se puede pasar al 3 o 5 (dependiendo del opcode).

Detalle del Secuenciador

- La lógica para determinar el próximo estado se implementa simplemente con un multiplexor.
- Agregamos dos ROM muy pequeñas para los casos en los que haya bifurcaciones en la MEF.



Dispatch ROM 1		
Op	**Opcode name**	**Value**
000000	R-format	0110
000010	jmp	1001
000100	beq	1000
100011	lw	0010
101011	sw	0010

Dispatch ROM 2		
Op	**Opcode name**	**Value**
100011	lw	0011
101011	sw	0101

Contenido de la ROM1

- A las salidas de control para el camino de datos se le agregan las salidas de control para el secuenciador (2 bits).
- Cada fila representa un estado.

State number	Control word bits 17–2	Control word bits 1–0
0	1001010000001000	11
1	0000000000011000	01
2	0000000000010100	10
3	0011000000000000	11
4	0000001000000010	00
5	0010100000000000	00
6	0000000001000100	11
7	0000000000000011	00
8	0100000010100100	00

- Tamaño total = 2^4*18 (ROM1)+ 2^7*4*2 (ROMs despacho) = 1,28 Kbits
- Versus 8,25 Kbit de la versión anterior, y 40 Kbit de la primera.

Microprogramación – Definición

- Un programa es una secuencia de instrucciones, que indica qué hacer con datos y cómo sigue la secuencia.
- En la ROM1 las salidas indican al camino de datos qué hacer, y cómo sigue la secuencia.
  - Entonces cada línea de la ROM1 se denomina microinstrucción.
- El contenido de la ROM1 se denomina **microprograma** o **microcódigo**. También denominado **firmware**.
- El registro de estado que apunta a la próxima microinstrucción se le llama microPC.
- Se puede definir un microensamblador, un formato de las microinstrucciones, etc, etc.

Microprogramación – Implicancias

- Cada instrucción del programa se mapea en una secuencia de microinstrucciones.
- Las microinstrucciones manejan señales de control, mientras que las instrucciones manejan datos.
- Las microinstrucciones son fijas.
  - Las instrucciones pueden cambiar entre cada programa.
  - Cambiar las microinstrucciones implica cambiar la ROM de microcódigo, cambiar el firmware.
    - Se hace para corregir bugs, o ampliar el ISA.
    - O para hacer que una máquina se comporte como otra (emulación).
  - Es muy fácil agregar nuevas instrucciones, más complejas, como combinación de microinstrucciones.
    - ISA incremental, tipo CISC.

Manejo de Interrupciones/Excepciones

- Al igual que con el diseño en pipeline, también se debe proveer soporte para el manejo de interrupciones y excepciones.
- En la MEF de control se agregan estados.
  - Bifurcaciones en los estados que puedan generar excepciones, hacia nuevos estados que guarden el PC y la causa.
  - Y luego otro estado que cargue PC con la dirección del manejador de excepciones.
- Siempre hay que tener en cuenta que las instrucciones con problemas no modifiquen los registros ni memoria.

Microprogramación – Conclusiones

- Es un diseño de control relativamente sencillo.
- Es muy flexible.
  - Permite hacer modificaciones con el tiempo.
- Permite implementar instrucciones muy poderosas.
  - Pero vimos que no se las prefiere.
- Facilita tener distintos ISA en una misma máquina.
- Facilita la compatibilidad.
  - Distintas máquinas con el mismo firmware.

Microprogramación – Conclusiones

- Tienen peor performance que máquinas RISC, pero…
  - Si se pudieran ejecutar instrucciones en un único ciclo de reloj, como las microinstrucciones.
  - Si se pudieran escribir programas que produzcan microinstrucciones.
  - Si la mayoría de los programas usara instrucciones simples, como las microinstrucciones.
  - Si el microcódigo pudiera guardarse en RAM y no en ROM.
- *¿Por qué no se hace que directamente esté compuesto por microinstrucciones?*
  - Sí se hace, y es una inspiración para el diseño RISC que vimos en los temas anteriores.

Resumen final

- Nuevo estilo de diseño: reutilizando recursos.
  - Se agregan registros de almacenamiento temporal entre ciclos (IR, MDR, etc.).
- Instrucciones de duración variable, en ciclos.
  - Se dividen en “pasos”, y se usan los que se necesitan.
  - Duración del ciclo determinado por el “paso” más lento.
- Control secuencial, no combinacional.
  - Especificado mediante una MEF de 9 estados.
  - Las señales de control dependen únicamente del estado.
  - Implementado con dos ROM. Poco eficiente.

Resumen final

- El control es la parte más difícil del diseño de un procesador multiciclo.
- Microprogramación es otra manera de implementar una MEF de control, para un diseño de procesador multiciclo.
  - Cada estado de la MEF es visto como una microinstrucción.
  - Basada en una ROM que contiene el microprograma y un secuenciador.
  - Mucho más simple, eficiente y flexible que otras implementaciones basadas en tablas de verdad.
  - Aún usada actualmente en procesadores cuyo objetivo primordial no es performance.

Agradecimientos

- Las diapositivas de este tema fueron basadas en las realizadas por el Ing. Daniel Cohen.
- A su vez inspiradas en las clases del curso CS152 de la Universidad de Berkeley, California, USA.
  - Realizadas por los Prof. D. A. Patterson, John Lazzaro, Krste Asanovic.