Tema 07: Diseño de un CPU de ciclo único

Arquitectura de Computadoras

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

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

Temas que veremos

- Componentes combinacionales y secuenciales.
- Metodología de clocking.
- Pasos para diseñar un procesador.
- Construcción paso a paso de un camino de datos.
- Diseño del control principal.
- Consideraciones de timing.

Lectura recomendada

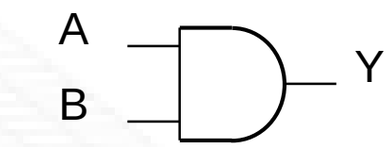
- Computer Organization and Design, RISC-V Edition (2da ed, 2021)
  - Sección 4.1: *Introduction*
  - Sección 4.2: *Logic Design Conventions*
  - Sección A.3: *Combinational Logic*
  - Sección 4.3: *Building a Datapath*
  - Sección 4.4: *A Simple Implementation Scheme*
  - Sección C.2: *Implementing Combinational Control Units*
  - Sección A.7: *Clocks*
  - Sección A.11: *Timing Methodologies*

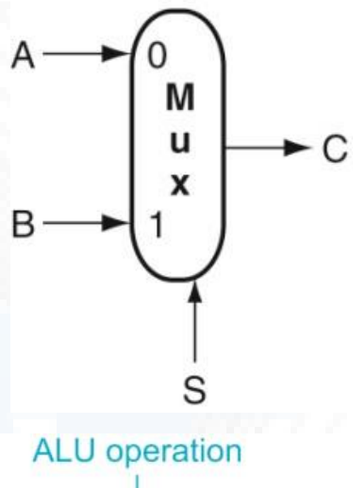
Repaso general y Objetivo

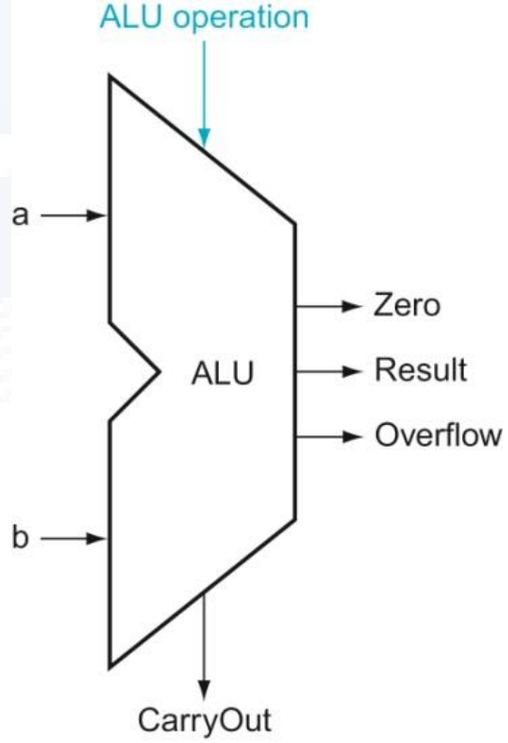
- Componentes básicas de una computadora:
  - Procesador (**Camino de datos** + **Control**), Memoria, Sistema de Entrada/Salida.
- Performance: **t = CI * CPI * T**
  - CI determinada por ISA y compilador
  - CPI y T influenciados por el camino de datos.
- Objetivo: diseñar un camino de datos simplificado que ejecute todas las instrucciones **en un ciclo de reloj**.
  - **CPI = 1**
  - ¿Desventajas? Las veremos al final.

Componentes combinacionales

- Sus salidas son función únicamente de sus entradas.
  - Las entradas y salidas pueden ser de múltiples bits.
- Compuertas AND
- Multiplexor
  - Usado para seleccionar un destino entre dos o más posibles fuentes.
    - Porque no se pueden unir cables así nomás.
- ALU y Sumador

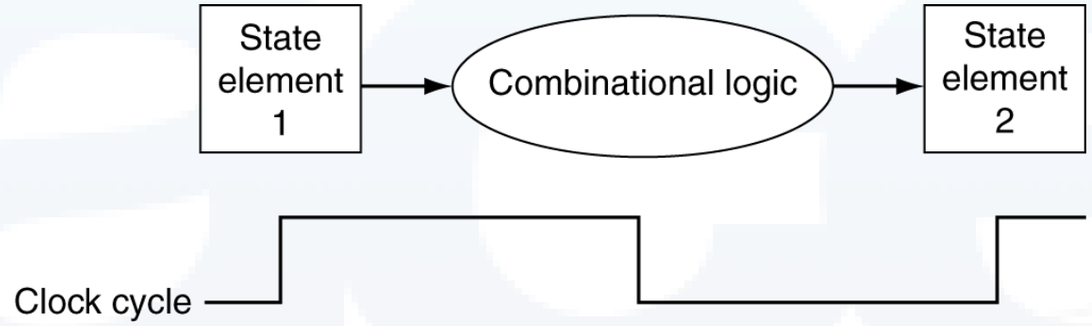






Metodología de Clocking

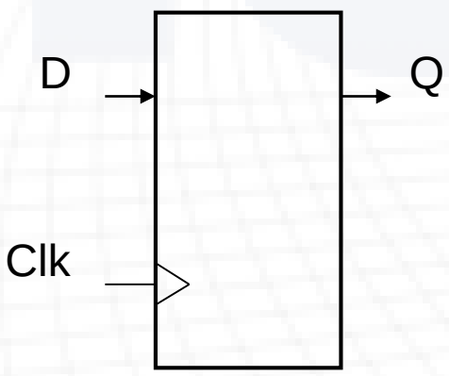
- Define cuándo las señales pueden ser leídas y cuándo escritas.
- La lógica combinacional modifica los datos en un ciclo de reloj (**entre dos flancos iguales**).
  - Siempre toma los datos de un elemento de estado, y guarda los resultados en un elemento de estado.

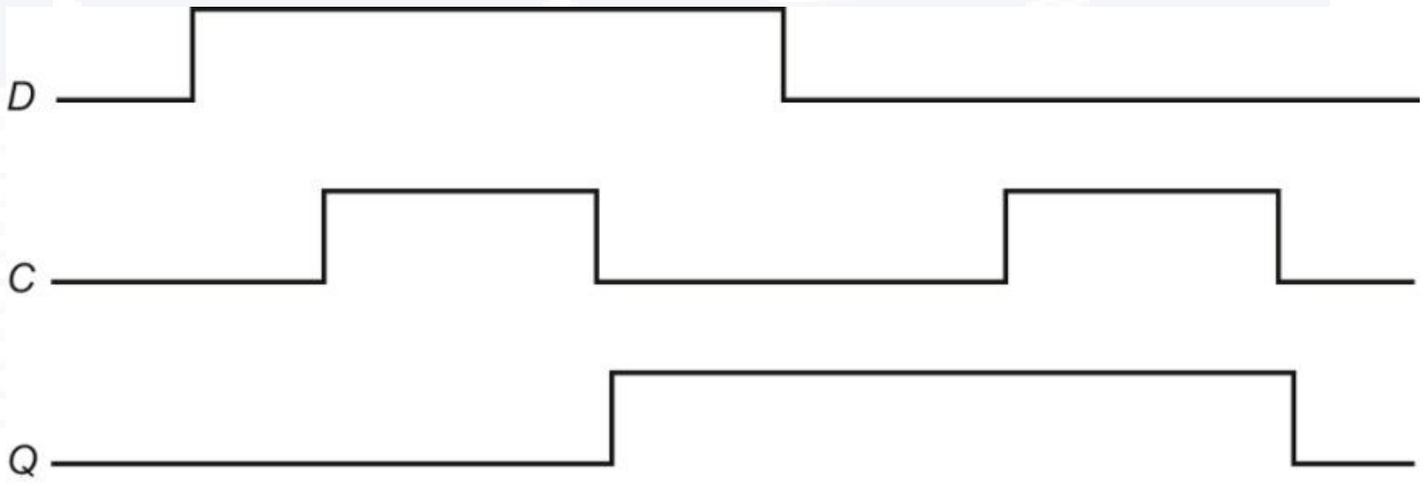


- **El mayor retardo combinacional determina la duración del ciclo de reloj.**
- Es posible que un elemento de estado sea leído y escrito en el mismo ciclo de reloj, sin ambigüedades.

Componentes secuenciales

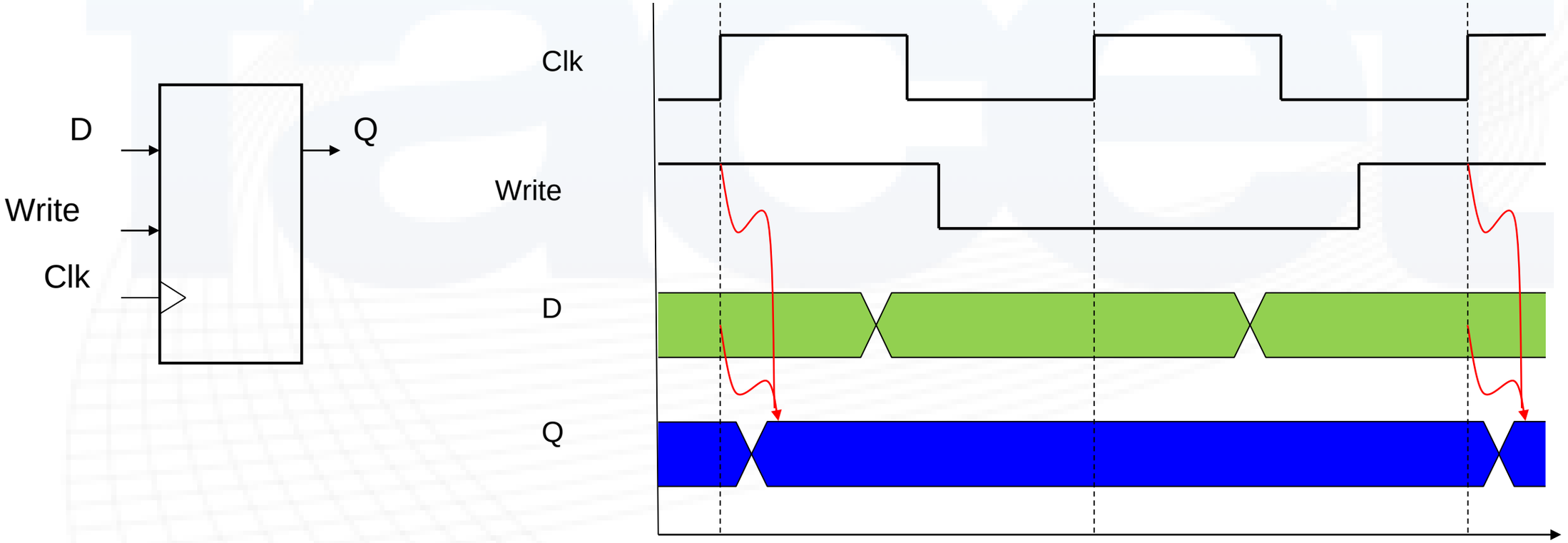
- Su salida depende de sus entradas y de su estado actual.
- Almacenan información sobre su estado.
- Registros
  - Usualmente un conjunto de Flip-Flop tipo D.
  - Usan una señal de reloj para determinar cuándo actualizar el valor almacenado.
  - Disparados por flanco de reloj.





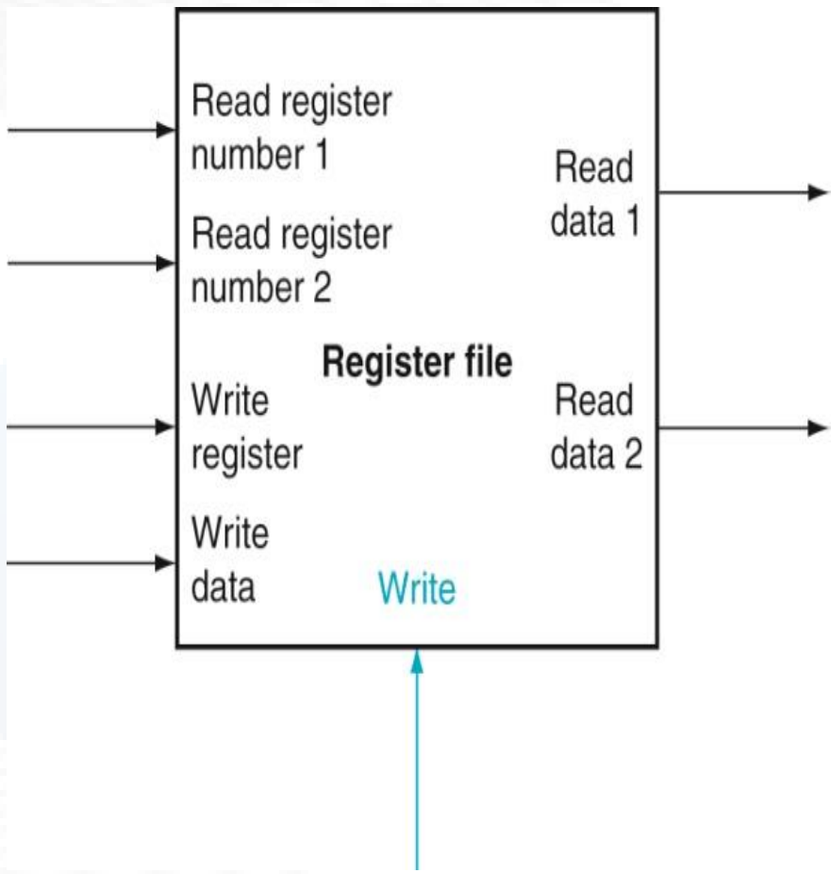
Componentes secuenciales

- Registros con control de escritura.
  - Iguales a los anteriores, pero sólo actualizan el valor almacenado al llegar el flanco de reloj si la señal de control está activa.



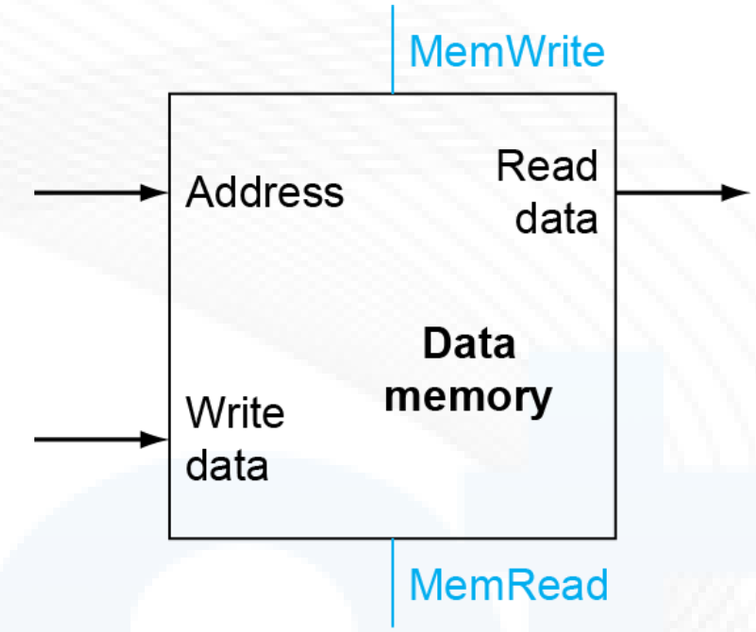
Banco de Registros

- 32 registros agrupados.
- Dos puertos de lectura de salida, con dos entradas de selección correspondientes.
  - Se comporta como un bloque combinacional.
  - El dato a la salida es válido luego de esperar un tiempo de acceso.
  - *¿Cuántos bits tienen las salidas?*
  - *¿Y las entradas?*
- Un puerto de escritura, con su entrada de selección correspondiente.
- Una señal de control de escritura.
- *¿Por qué dos puertos de lectura y uno de escritura?*



Memoria “Ideal”

- Una entrada de direcciones.
- Una entrada de datos
- Una salida de datos.
- Dos señales de control
  - Para Lectura y para Escritura.
  - No siempre es así.



- Cuando la señal de control de lectura está activa, se muestra en la salida el contenido apuntado por la dirección de entrada, después de un cierto tiempo de acceso.
- Cuando la señal de control de escritura está activa, se guarda en el contenido apuntado por la dirección el valor en la entrada de datos, después de un cierto tiempo de acceso.

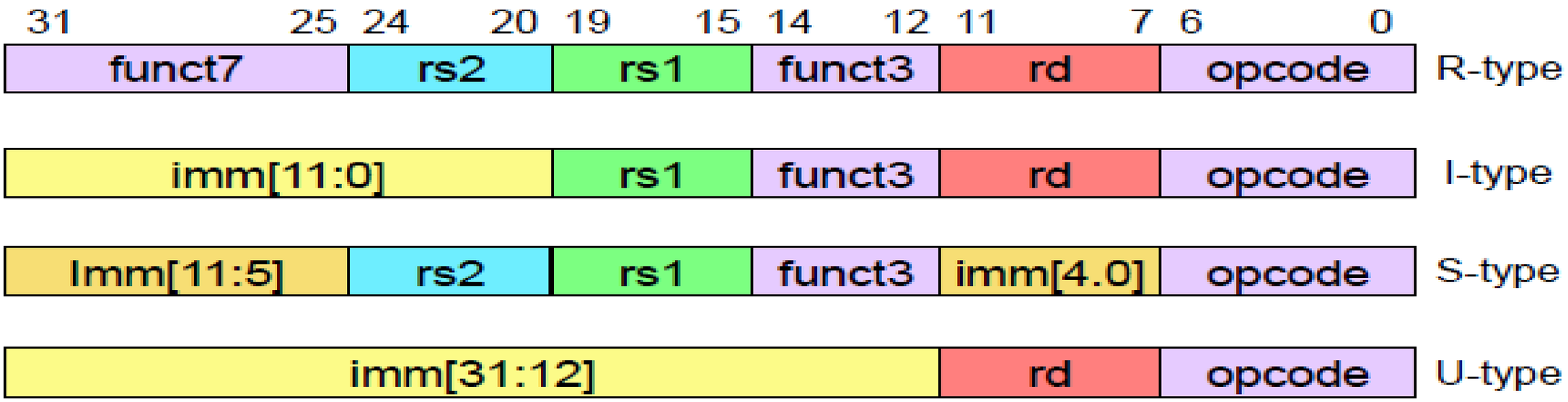
Pasos para Diseñar un Procesador

- ¿Por dónde empezar?
  - **Por el ISA**.
  - El objetivo es ejecutar correctamente las instrucciones especificadas.
- Describimos las instrucciones en RTL abstracto.
  - Eso nos va a dar una pauta de los componentes necesarios para el camino de datos.
  - Y de cómo interconectarlos.
  - *¿Recuerdan SMM y el SRC-1?*
- Una vez que tengamos los componentes, tendremos sus señales de control.
  - Y con ellas diseñaremos la unidad de control que se encargue de generarlas.

Subconjunto del ISA de RISC-V

- Nuestro camino de datos no implementará todas las instrucciones.
  - Pero la técnica es la misma al momento de agregar las demás.
  - Referencias a memoria: LW, SW.
  - Procesamiento: ADD, SUB, AND, OR.
  - Saltos: BEQ
  - *¿Qué formatos de instrucción tienen?*

Repaso RV32I – Formatos de Instrucciones



- Todos los formatos manejan un Opcode de 7 bits.
- Dos formatos adicionales (usados para saltos)
  - B (casi igual que S) y J (casi igual que U).
  - La diferencia es que el campo inmediato está desplazado un bit.
- Los campos para los registros que van a ser leídos o escritos están siempre en la misma ubicación para todas las instrucciones.
- Los campos inmediatos siempre son extendidos en signo (bit 31).

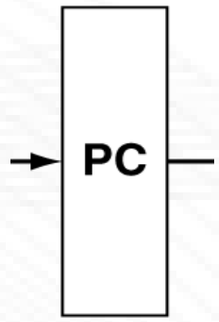
Algunos RTL de ejemplo

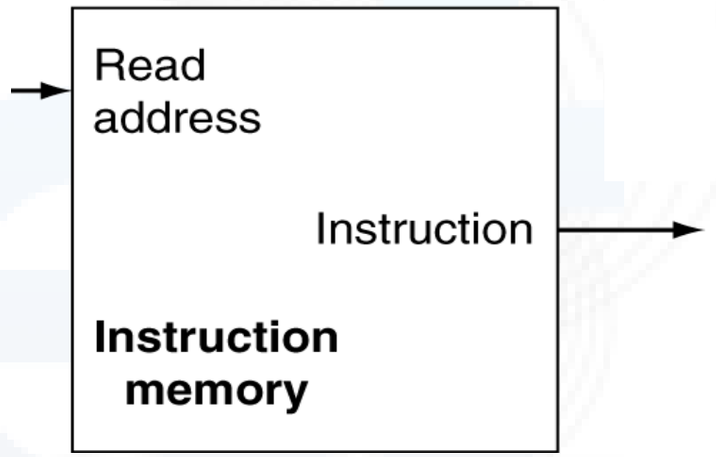
Instrucción	RTL Abstracto
add rd, rs1, rs2	M[PC]; PC ← PC + 4 R[rd] ← R[rs1]+R[rs2]
lw rd, rs1, imm	M[PC]; PC ← PC + 4 R[rd] ← M[R[rs1] + SignExt(imm)]
sw rs1, rs2, imm	M[PC]; PC ← PC + 4 M[R[rs1] + SignExt(imm)] ← R[rs2]
beq rs1, rs2, imm	M[PC]; Si (R[rs1] == R[rs2]) PC ← PC + SignExt(imm) Si no PC ← PC + 4

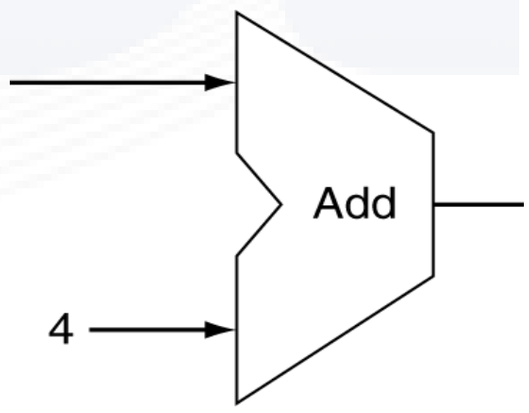
- Todos los Fetch son idénticos, excepto para el salto.
  - *¿Por qué PC se incrementa en 4?*
- Lo que está en rojo es el cálculo de la dirección de acceso a memoria.
  - *¿Cómo funciona la extensión de signo?*
- El salto requiere una comparación.

Fetch de Instrucciones

- Común a todas las instrucciones secuenciales.
  - Dejamos los saltos en suspenso momentáneamente.
- ¿Qué componentes se necesitan?
  - Un registro PC
    - Se escribe al final de cada ciclo.
  - Una memoria para instrucciones
    - No necesita señales de control, ni entrada de datos.
    - La salida de la memoria contiene la instrucción a ejecutar.
  - Un sumador que sume 4.

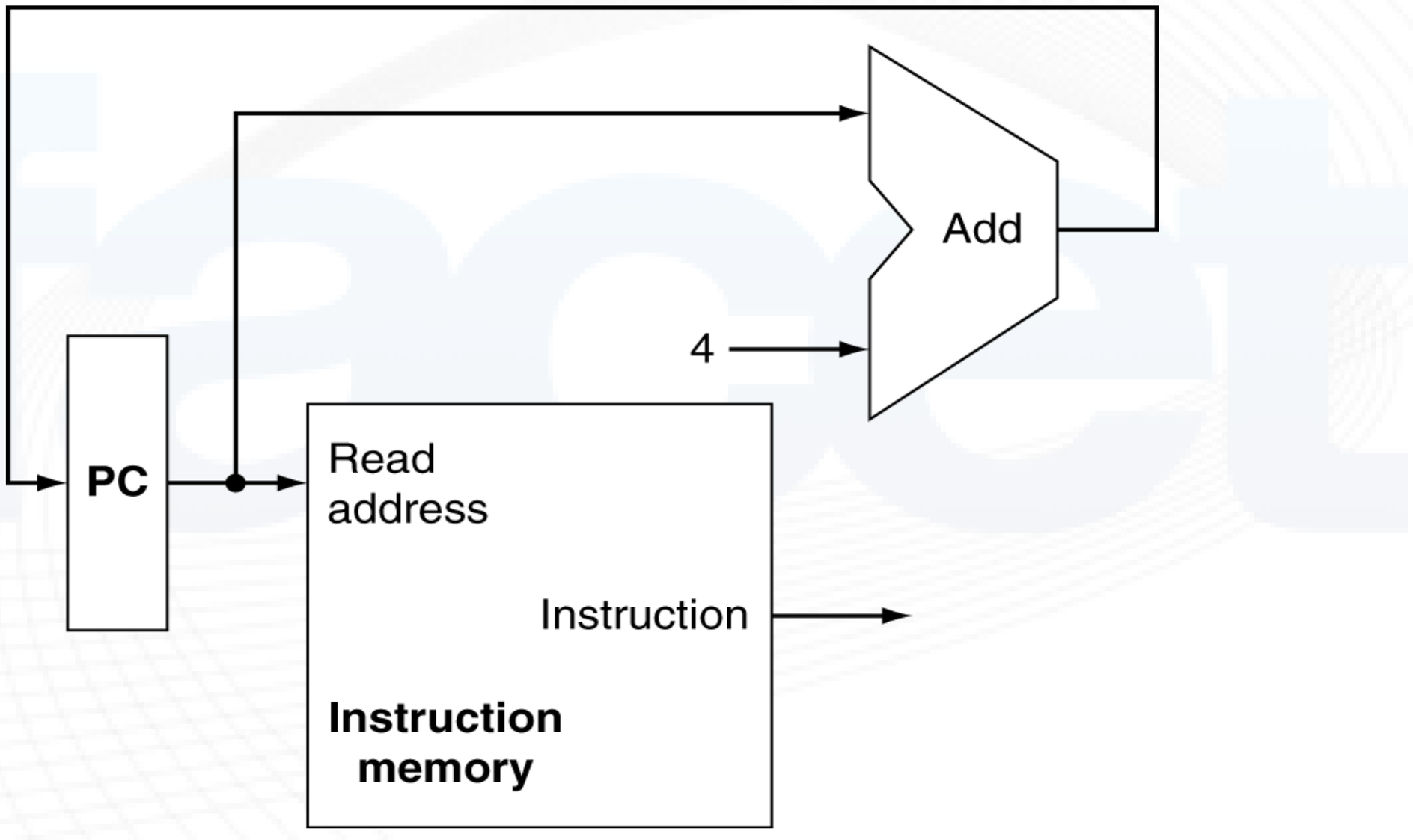






Camino de datos para el Fetch

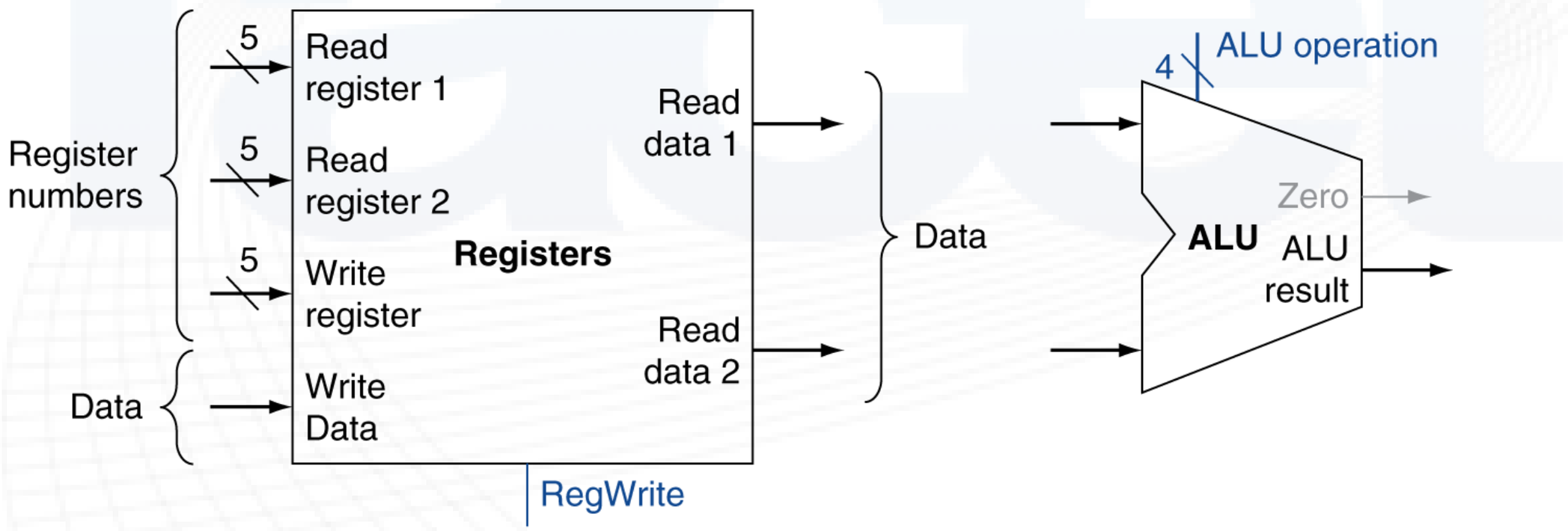
- Conectamos todos los componentes de la mejor manera posible, buscando performance.



Instrucciones de Formato Tipo R

Instrucción	RTL Abstracto
add rd, rs1, rs2	R[rd] ← R[rs1]+R[rs2]

- Necesitan leer dos registros del banco de registros.
  - Campos disponibles en la instrucción.
- Realizar la operación aritmética/lógica en una ALU.
- Escribir el resultado nuevamente en el banco de registros.



Instrucciones Load/Store

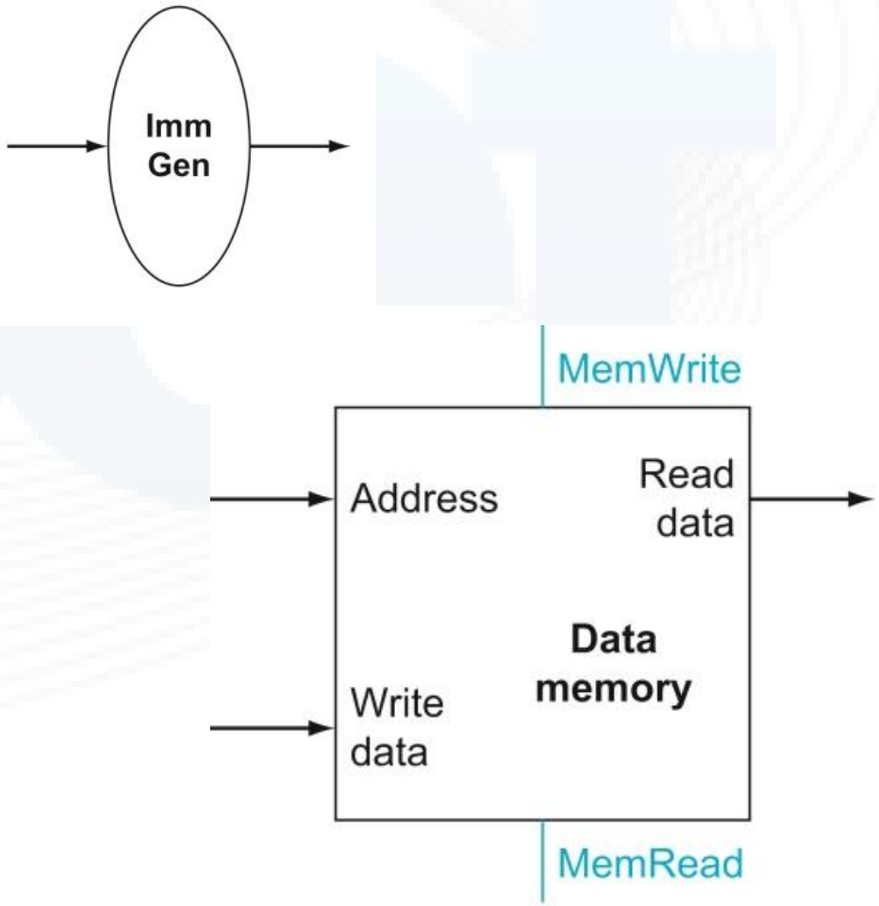
Instrucción	RTL Abstracto
lw rd, rs1, imm	R[rd] ← M[R[rs1] + SignExt(imm)]
sw rs1, rs2, imm	M[R[rs1] + SignExt(imm)] ← R[rs2]

- También necesitan leer registros del banco de registros.
- Calculan la dirección de memoria en la ALU usando la constante inmediata que viene con la instrucción.
  - Hay que extenderla antes que ingrese a la ALU.
- Load accede a memoria, y luego guarda el resultado en el banco de registros.
- Store guarda el dato del banco de registros en memoria.
- Esta memoria, *¿puede ser la misma que para las instrucciones?*
  - Se necesita una memoria separada para datos.

Instrucciones Load/Store

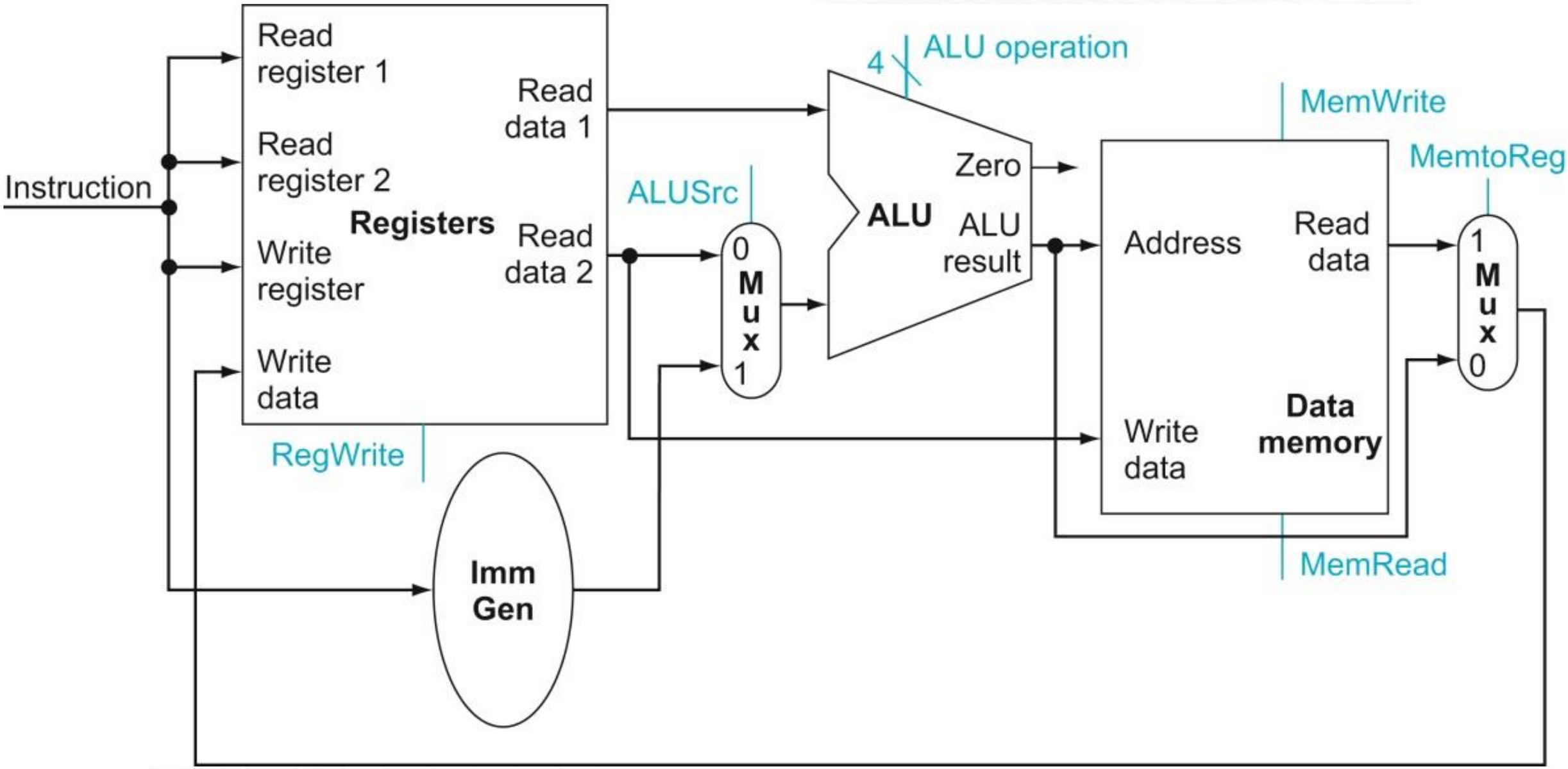
Instrucción	RTL Abstracto
lw rd, rs1, imm	R[rd] ← M[R[rs1] + SignExt(imm)]
sw rs1, rs2, imm	M[R[rs1] + SignExt(imm)] ← R[rs2]

- Se necesitan dos nuevos componentes:
  - Un extensor de signo para la constante inmediata.
  - Una memoria de datos.
    - Entrada de direcciones
    - Entrada de datos
    - Salida de datos
    - Señales de control de lectura y escritura



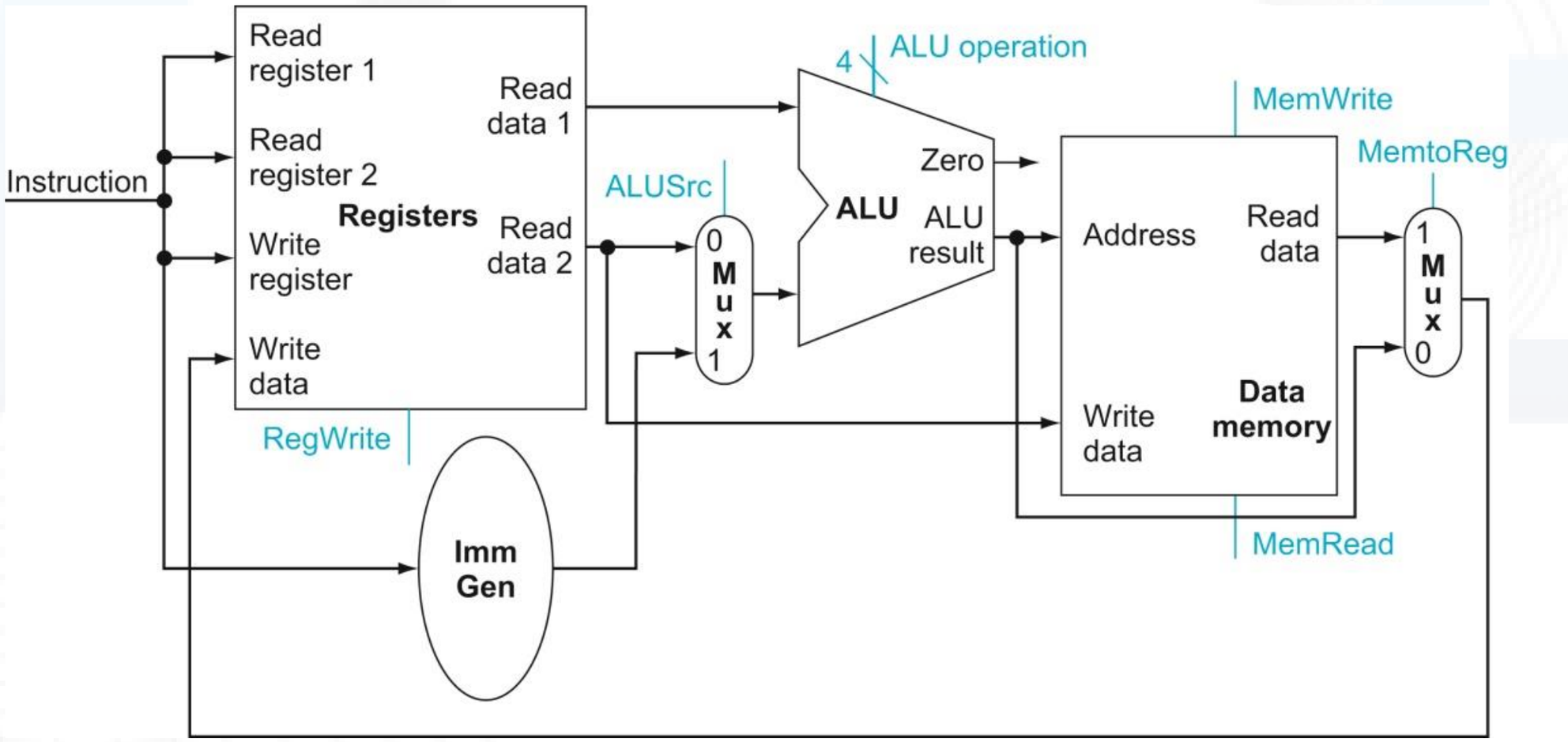
Camino de Datos Tipo R/Load/Store

- Nuevamente, conectamos todos los componentes de la mejor manera posible, buscando performance.



Camino de Datos Tipo R/Load/Store

- Surge la necesidad de agregar dos multiplexores.
  - El segundo operando de la ALU puede venir del banco de registros o de la constante inmediata.
  - El dato a escribir en el banco de registros puede venir de la ALU o de la memoria.

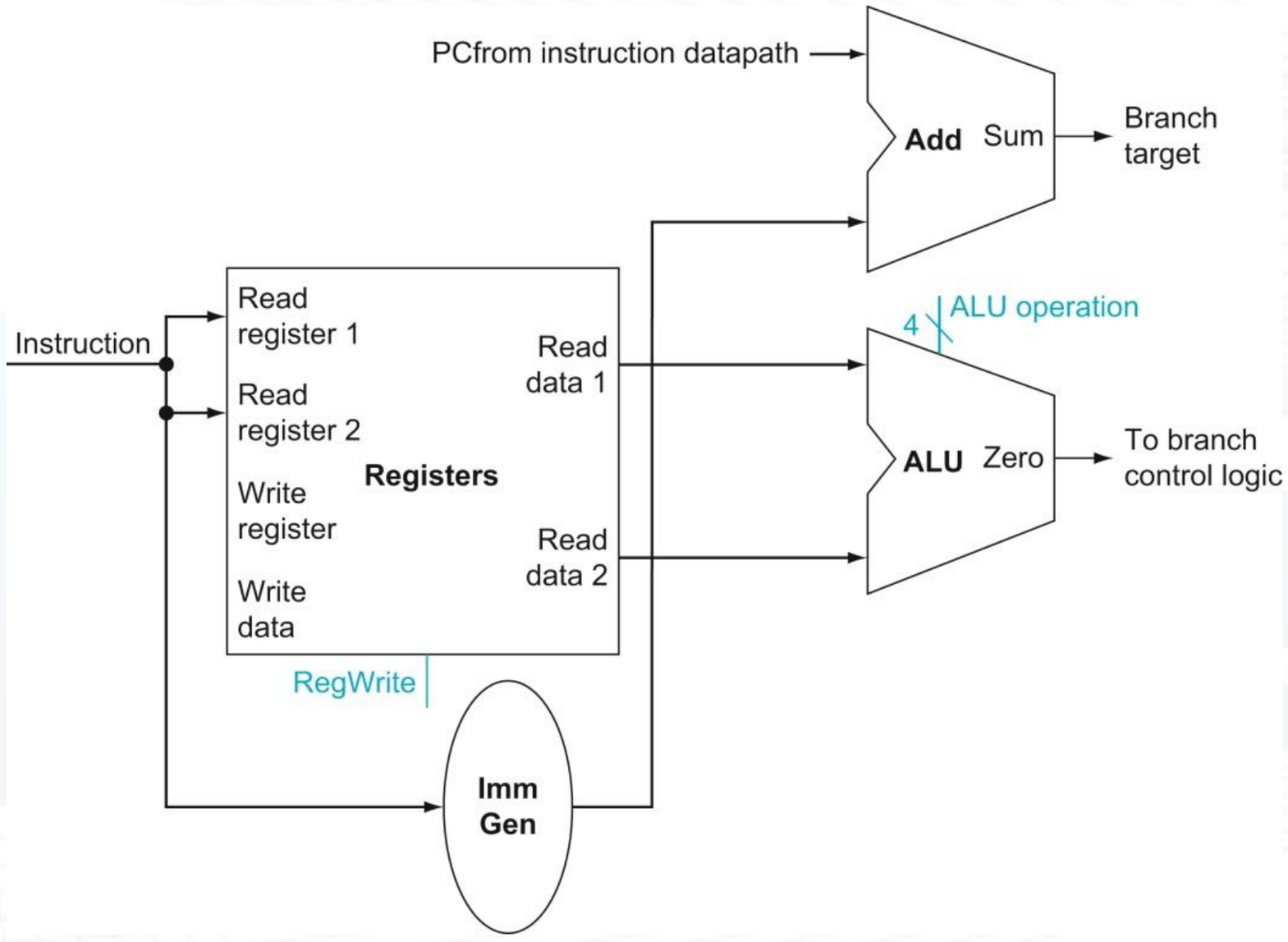


Instrucciones Branch

Instrucción	RTL Abstracto
beq rs1, rs2, imm	Si (R[rs1] == R[rs2]) PC ← PC + SignExt(imm) Si no PC ← PC + 4

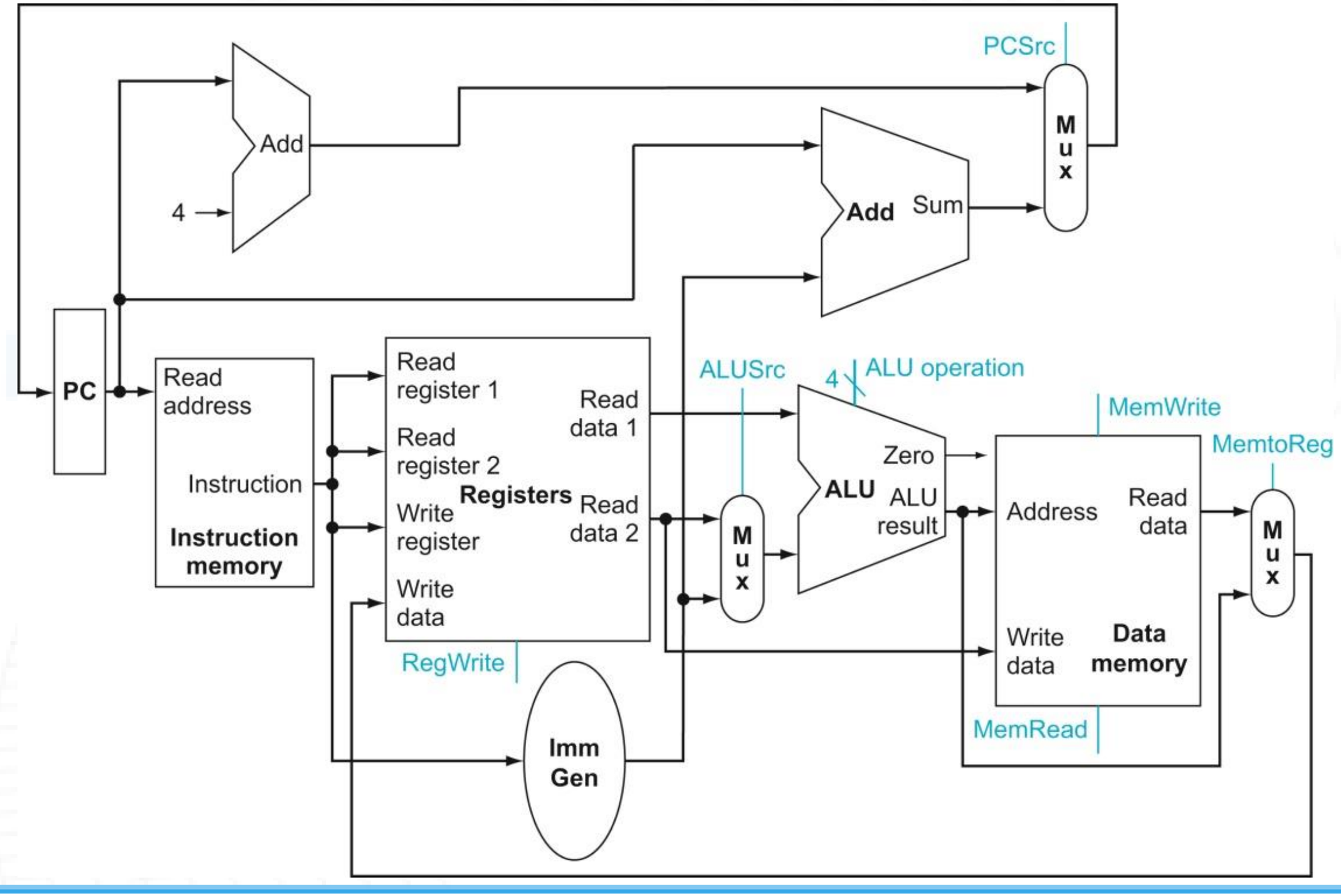
- También leen dos registros del banco de registros.
- Usan la ALU para compararlos.
  - Los restan, y verifican si la salida es igual a 0.
- Calcula la dirección destino:
  - Extienden la constante inmediata.
  - La desplazan un lugar hacia la izquierda.
    - *¿Por qué?*
  - Suman este desplazamiento a PC.
- Genera la necesidad de un nuevo sumador.
- Y de un nuevo multiplexor, porque ahora PC puede tomar dos valores diferentes.

Camino de datos Branch



- El desplazamiento hacia la izquierda es simplemente mover los cables de lugar.

Camino de datos completo (1)



Señales de control

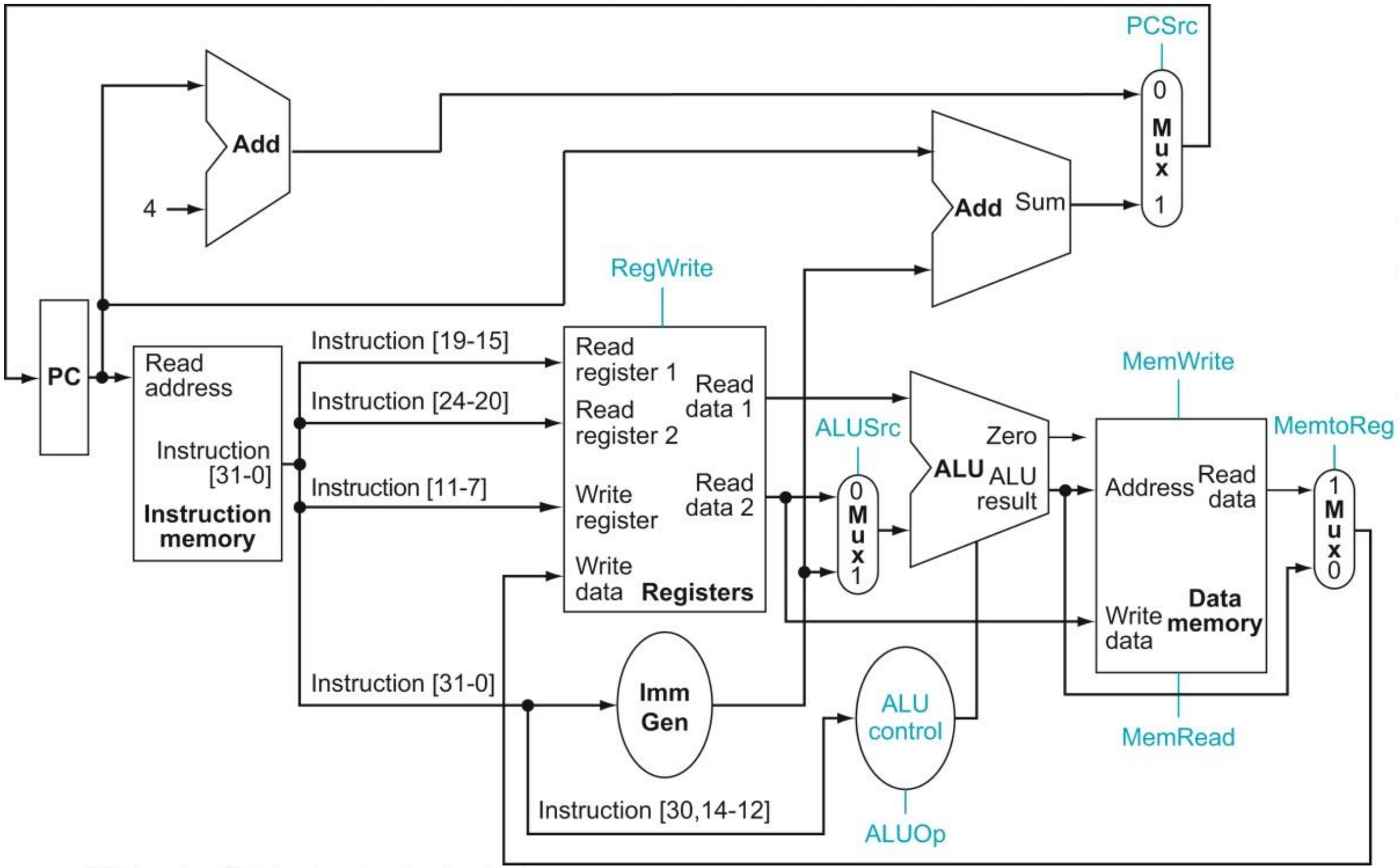
- En el diagrama anterior, quedan 7 señales de control por especificar.
- Una de ellas es la función de la ALU, típicamente de 4 bits.
  - Aunque nosotros no necesitamos todas las funciones.
    - Para LW y SW, necesitamos que sume.
    - Para BEQ, que reste.
    - Para las Tipo R, que haga lo que corresponda.
  - Y además, esta señal de control de la ALU se usa “un poco después” que las demás.
- Decisión de compromiso: **separar el control de la ALU del control Principal**.
  - Reduce el tamaño y el retardo del control principal.

Control de la ALU

- Es un **control local**, **completamente combinacional**.
- Posee como entrada un campo de 2 bits, ALUOp, que permite codificar las tres funciones principales mencionadas.
  - 00 para que sume, 01 para que reste, 10 para que haga lo determinado por los campos función.
  - Este campo depende de la instrucción, y vendrá del control principal.
- Además recibe como entrada los campos función de las instrucciones de formato R.
- Genera como salida los 4 bits para controlar la ALU.

Instruction opcode	ALUOp	Operation	Funct7 field	Funct3 field	Desired ALU action	ALU control input
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
R-type	10	sub	0100000	000	subtract	0110
R-type	10	and	0000000	111	AND	0000
R-type	10	or	0000000	110	OR	0001

Camino de datos completo (2)



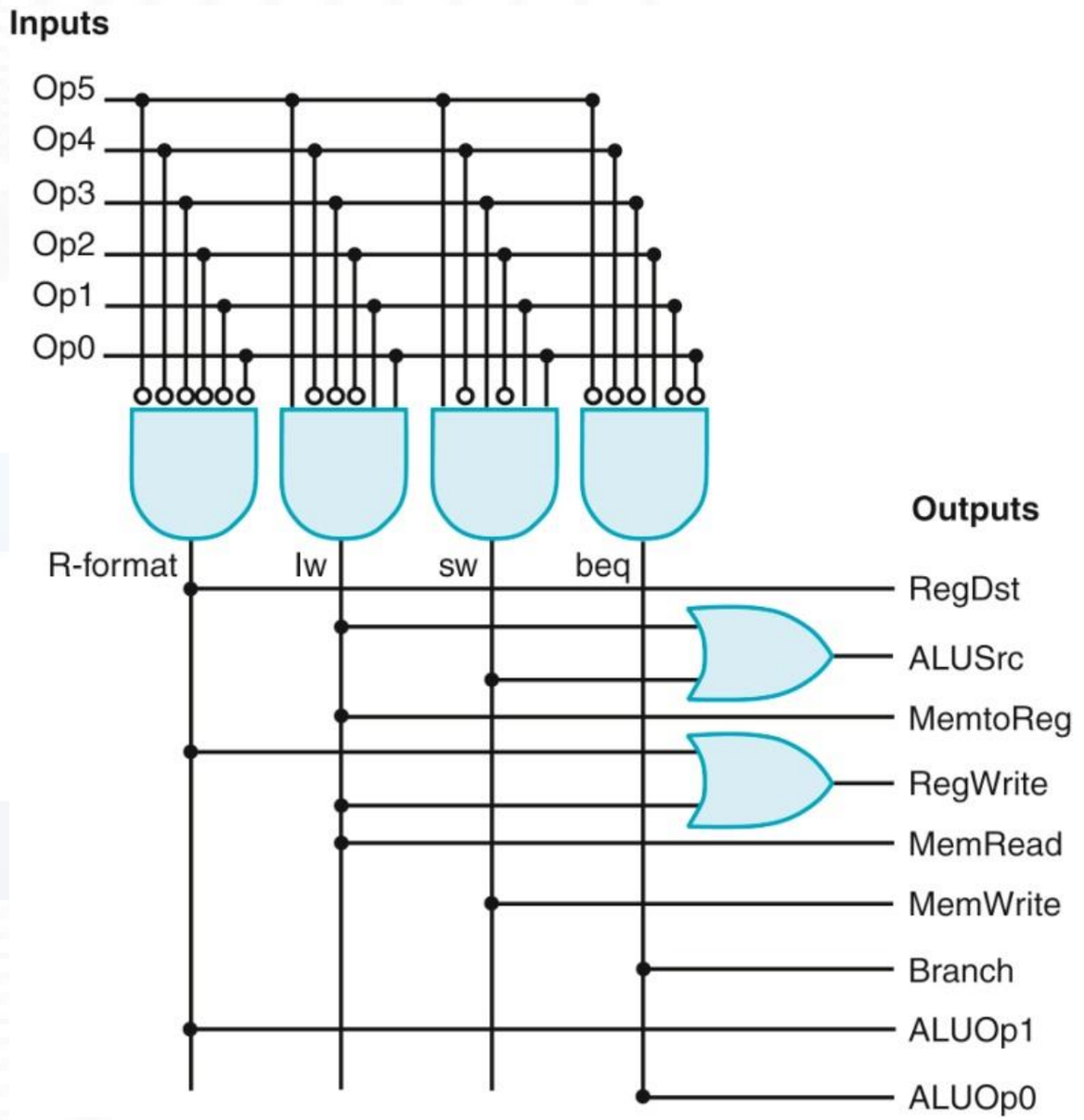
Diseño del Control principal

- Es necesario establecer 6 señales de control de 1 bit, más ALUOp de 2 bits.
  - Los valores que tomarán estas señales dependen de la instrucción a ejecutar (o sea, del opcode).
  - Nuevamente, **totalmente combinacional**.
- La señal PCSrc depende del resultado de la ALU.
  - Por lo tanto, sólo generaremos una señal Branch que indique que es un salto.
  - PCSrc = Branch & Zero

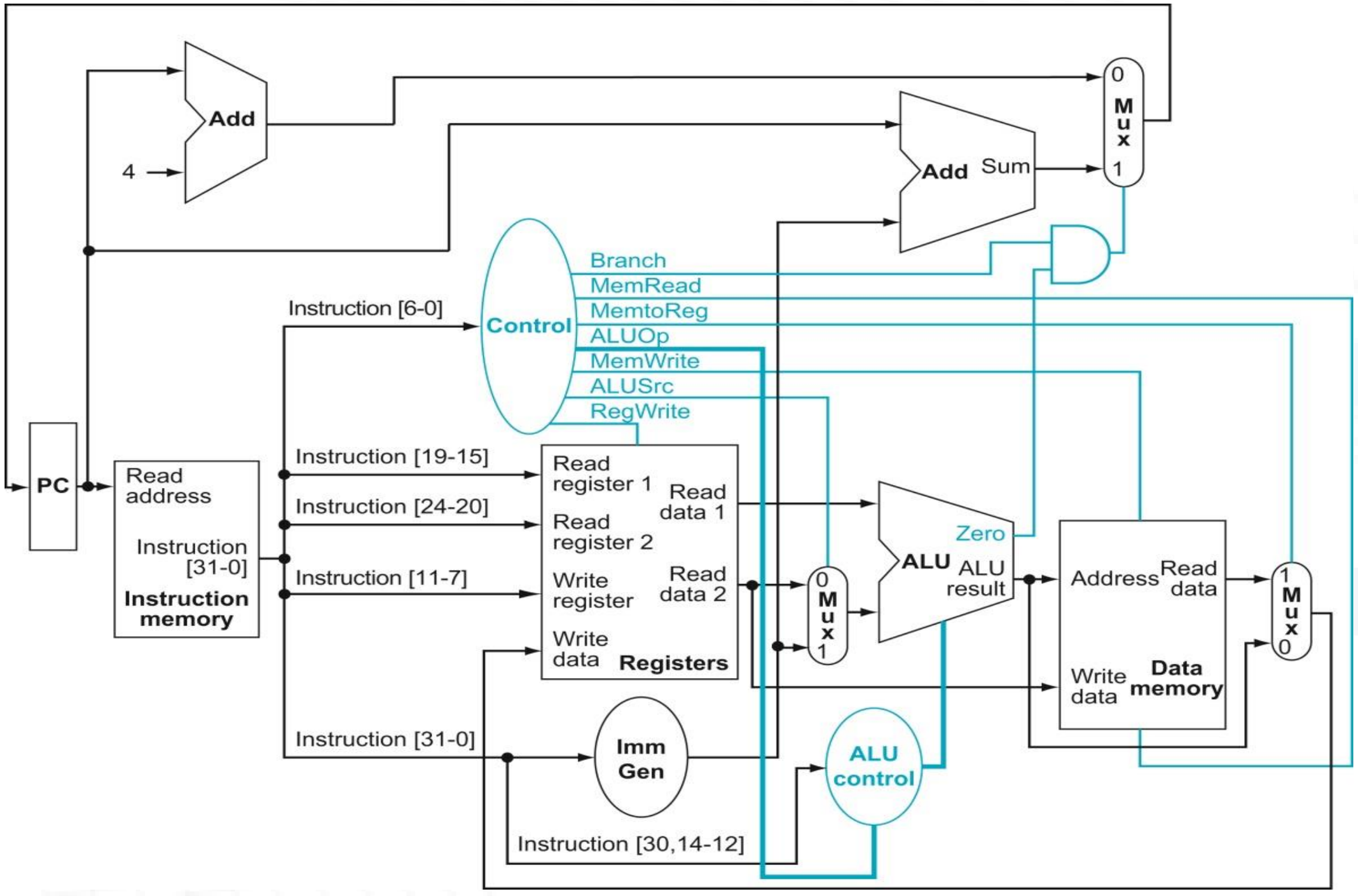
Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
lw	1	1	1	1	0	0	0	0
sw	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Diseño del Control principal

- Al ser totalmente combinacional, se implementa mediante una tabla de verdad.
  - Se busca simplificar señales, mediante condiciones de indiferencia.
  - Se escriben las ecuaciones para cada señal de salida **en una lógica de dos niveles**, como suma de productos.
  - Y se las implementa en un arreglo tipo PLA, **similar** al de la imagen.



Camino de datos ciclo único completo



Timing del camino completo

- La duración del ciclo de reloj está determinada por el mayor retardo.
  - Tenemos que determinar el **camino crítico**.
    - Instrucción que use más unidades funcionales.
    - Memoria de instrucciones + banco de registros + operación en ALU + memoria de datos + escribir resultado en banco de registros.
  - **Nuestro peor caso es la instrucción LW**.
  - Suponiendo que las memorias tienen un retardo de 200 ps, la ALU y los sumadores de 200 ps y el banco de registros de 100 ps (demás retardos despreciables):
    - *¿Cuánto demora la instrucción LW?*
    - *¿Y las demás?*

Conclusiones diseño ciclo único

- ¡Estamos violando uno de nuestros Principios de Diseño! *¿Cuál?*
  - **GRAN** problema de performance.
- Tampoco podemos hacer un reloj de período variable para cada instrucción.
- Además estamos duplicando recursos.
  - Más área, mayor costo.
- Sin embargo…
  - ¡Diseñamos nuestro primer camino de datos!
  - Con CPI = 1, que es bastante bueno.
  - Tiene mucho potencial para mejorar

Resumen final

- Para diseñar un procesador se parte desde el ISA, haciendo el RTL de las instrucciones.
  - Usamos un subconjunto, pero la técnica es general.
- Fuimos construyendo paso a paso el camino de datos, con las unidades funcionales necesarias.
  - Definimos los valores de las señales de control.
- Diseñamos el control principal.
  - Y un control local para la ALU.
  - Ambos combinacionales. Implementados en lógica de dos niveles, mediante PLA.
- Logramos el objetivo de CPI = 1.
- Pero T muy grande, determinado por la instrucción más larga.
  - No cumple con un Principio de Diseño.

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.