Tema 05: 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

- Repaso diseño microprocesador multiciclo.
- Construcción paso a paso del camino de datos.
- Diseño del control principal.
- Consideraciones de timing.
- Comparación entre ambos diseños.

Lectura recomendada

- Computer Organization and Design, RISC-V Edition (2da ed, 2021)
  - Sección 4.3: *Building a Datapath*
  - Sección 4.4: *A Simple Implementation Scheme*

- Opcionales:
  - Sección 4.1: *Introduction*
  - Sección 4.2: *Logic Design Conventions*
  - Sección A.3: *Combinational Logic*
  - Sección A.7: *Clocks*
  - Sección A.11: *Timing Methodologies*

Repaso general

- Partes 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.
- Ya vieron el diseño de un camino de datos para RV32I.
  - **Repasar Temas 05 y 06 de Sistemas con micros**

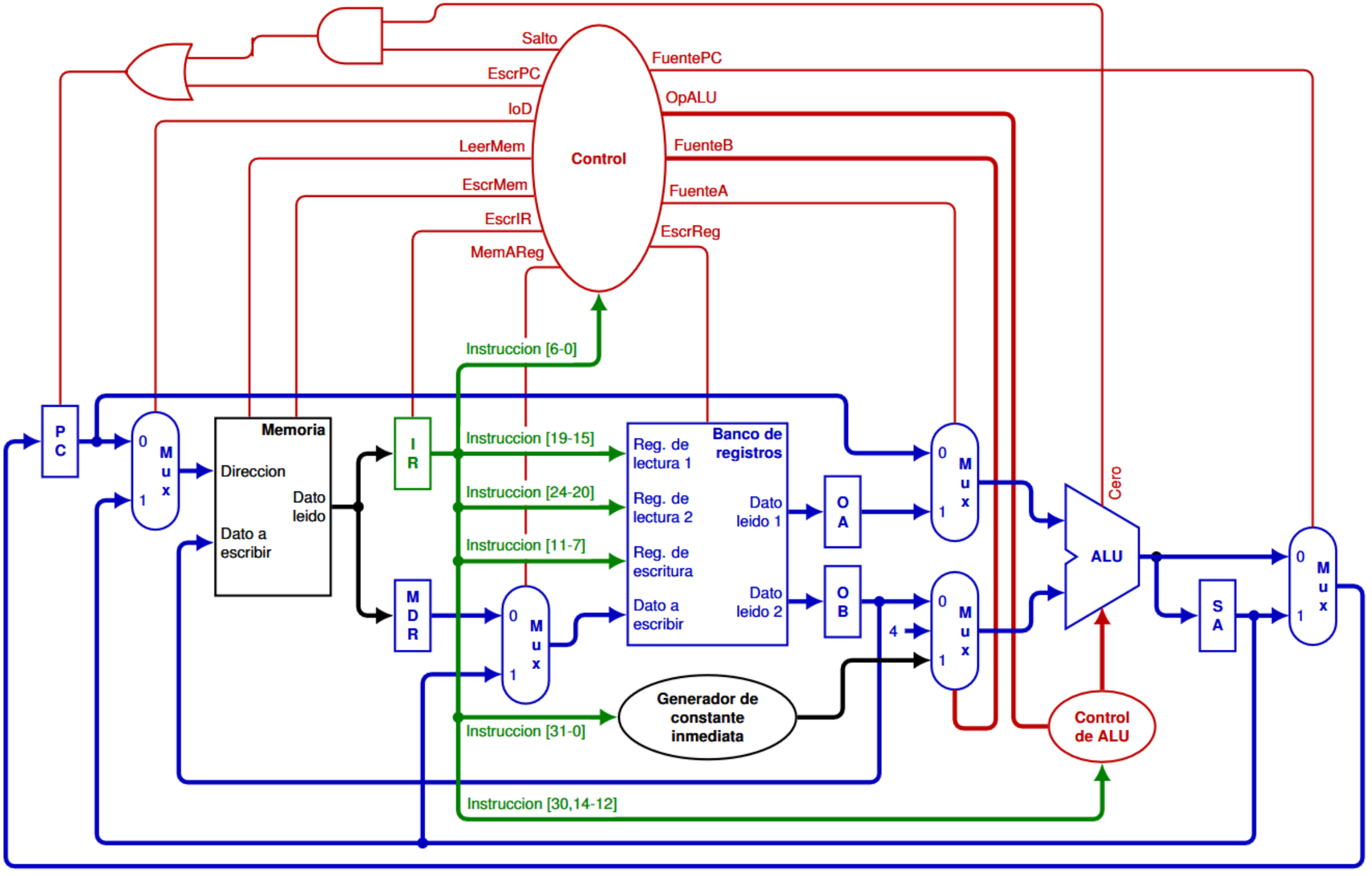
Repaso Procesador Multiciclo

- Objetivos:
  - Ejecutar las instrucciones en múltiples ciclos de reloj.
  - Minimizar la duración del ciclo de reloj.
  - Sin duplicar recursos innecesarios, para minimizar costos.

Repaso Procesador Multiciclo

- Proceso de diseño:
  - Partimos desde el ISA, porque es la interfaz entre el hardware y el software.
    - El camino de datos debe ejecutar correctamente las instrucciones.
  - Definimos un RTN abstracto para un subconjunto de instrucciones.
    - **lw**, **sw**, **add**, **sub**, **and**, **or** y **beq**.
  - Se fueron agregando componentes combinacionales y secuenciales a medida que se fueron necesitando e interconectándolos entre sí.
    - Con multiplexores en donde hacía falta.

Repaso Procesador Multiciclo



Repaso Procesador Multiciclo

- Que los formatos de las instrucciones sean pocos y con sus campos regulares ayudó mucho al diseño del camino de datos.
- Consideraciones importantes:
  - Una única memoria para datos e instrucciones.
  - Una única ALU.
  - Se agregaron registros adicionales, para almacenar valores temporales.
    - IR, MDR, OA, OB y SA.
    - Estos registros no forman parte del ISA, y son una decisión de diseño del camino de datos.
  - Retardos: Memoria y ALU (200 ps), banco de registros (100 ps), generador de constantes (30 ps), registros y multiplexores (despreciables).

Repaso Procesador Multiciclo

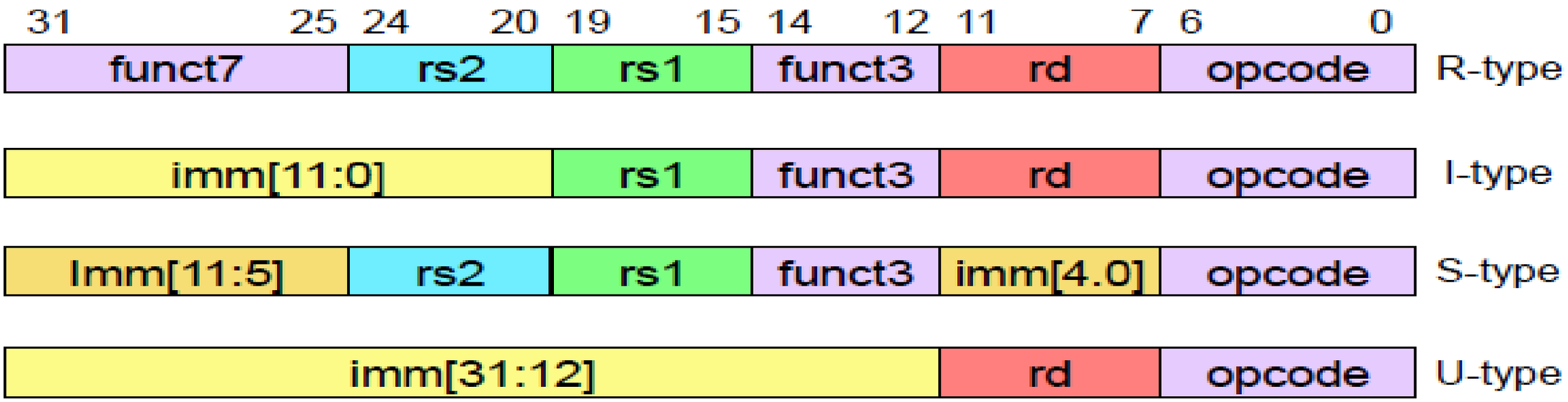
- Consideraciones importantes:
  - Control de la ALU independiente del Control principal.
    - Combinacional, implementado con lógica de dos niveles.
  - Control principal es secuencial.
    - MEF estilo Moore de 10 estados.
    - Implementado mediante microprogramación.
    - Gran flexibilidad y modularidad.
  - Distintas instrucciones necesitan distintas cantidad de ciclos para ejecutarse.
  - Ciclo de reloj determinado por la duración de la microinstrucción más lenta: 200 ps.

Nuevo Objetivo

- Diseñar un camino de datos que ejecute todas las instrucciones **en un ciclo de reloj**.
  - **CPI = 1**
  - *¿Desventajas?* Las veremos al final.

- *¿Cómo empezamos?*
  - Con el mismo proceso de diseño ya visto.
  - Pero tomando algunas decisiones diferentes.

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).

Repaso RTN Abstractos

Instrucción	RTN 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]`

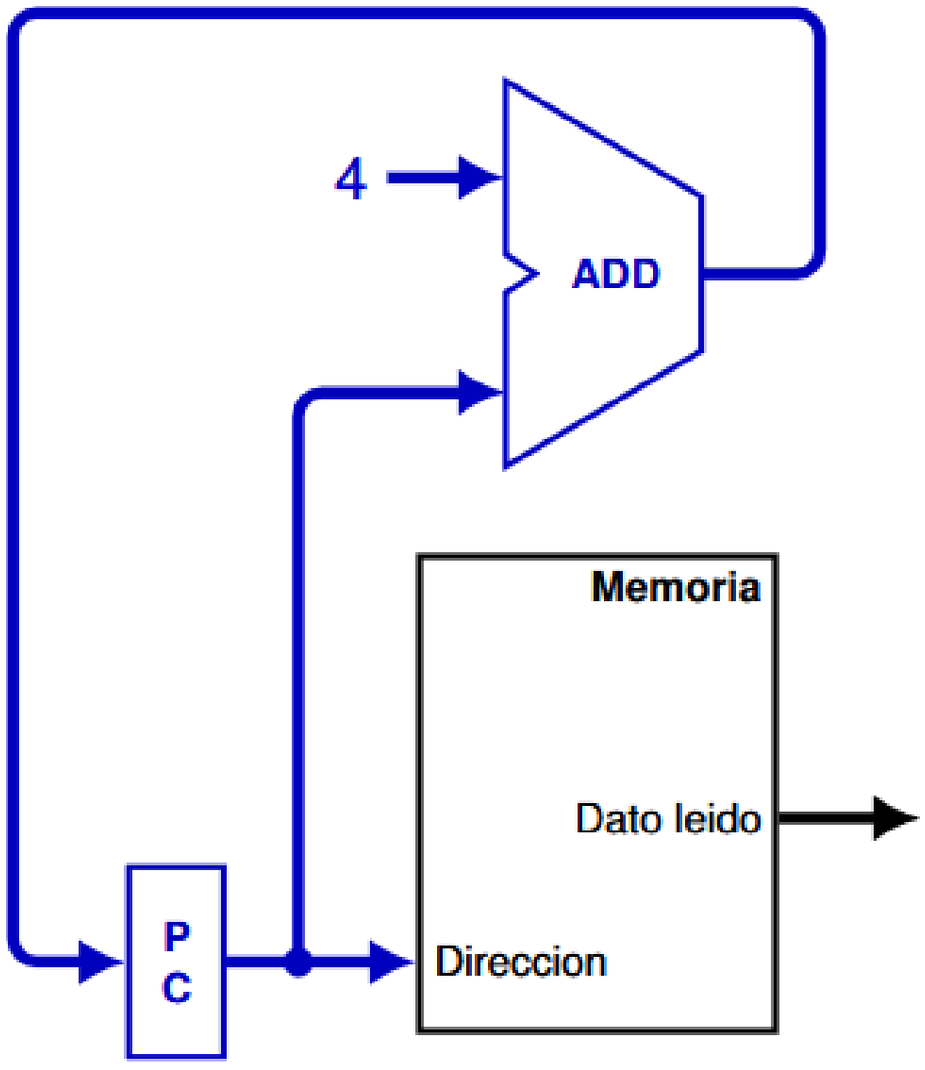
`beq rs1,rs2,imm`	`M[PC]`
	`PC ← PC + 4`
	`Si (R[rs1] == R[rs2]) PC ← PC + SignExt(imm)`

Fetch de Instrucciones

- Común a todas las instrucciones.
- ¿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.



- *¿Dónde guardamos la instrucción leída de memoria?*

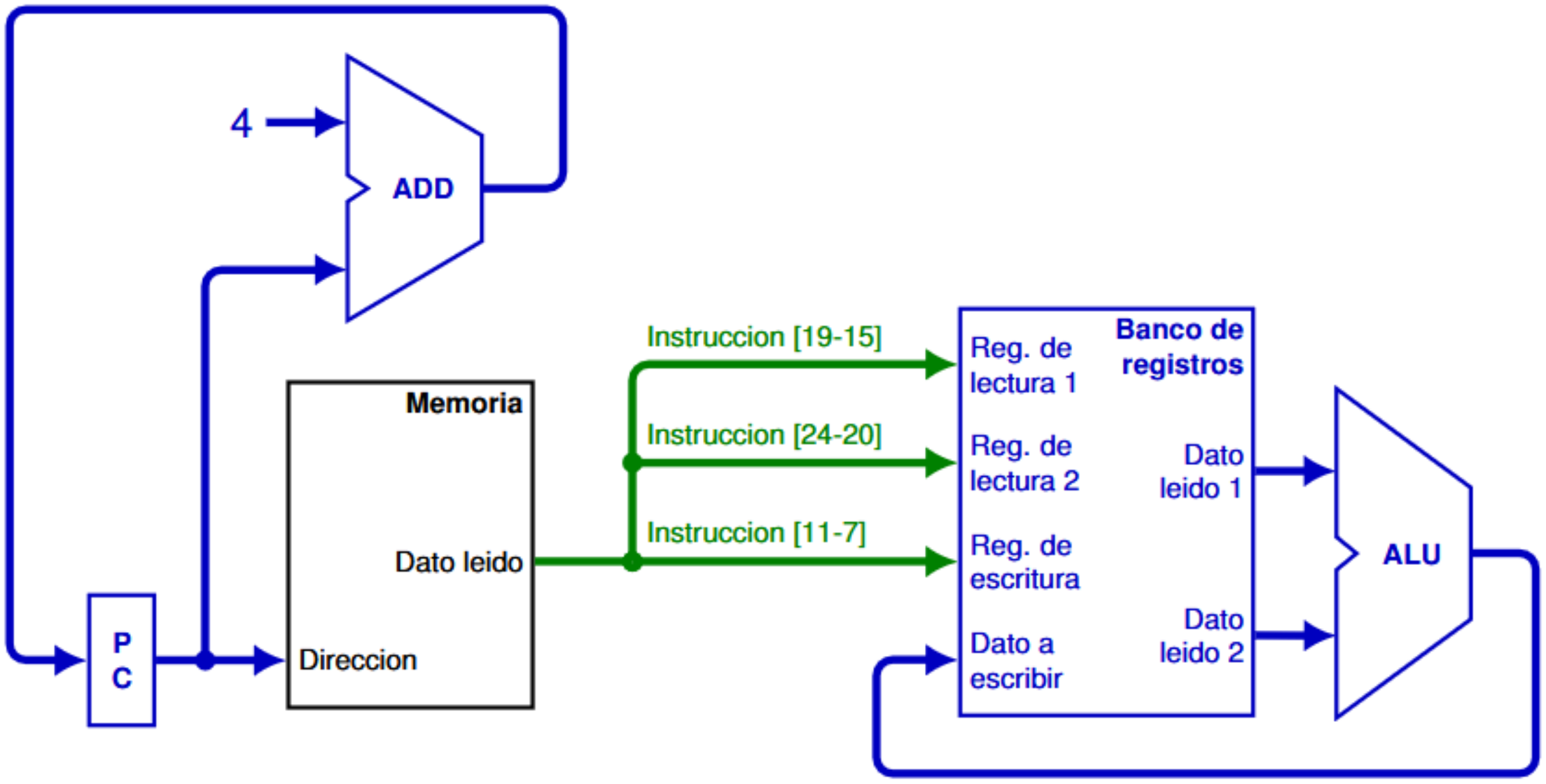
Instrucciones de Formato Tipo R

Instrucción	RTN 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.
  - *¿Podemos reutilizar el sumador que pusimos para el fetch?*
- Escribir el resultado nuevamente en el banco de registros.

Camino de datos Fetch + Tipo R

- Agregamos estos dos nuevos componentes:



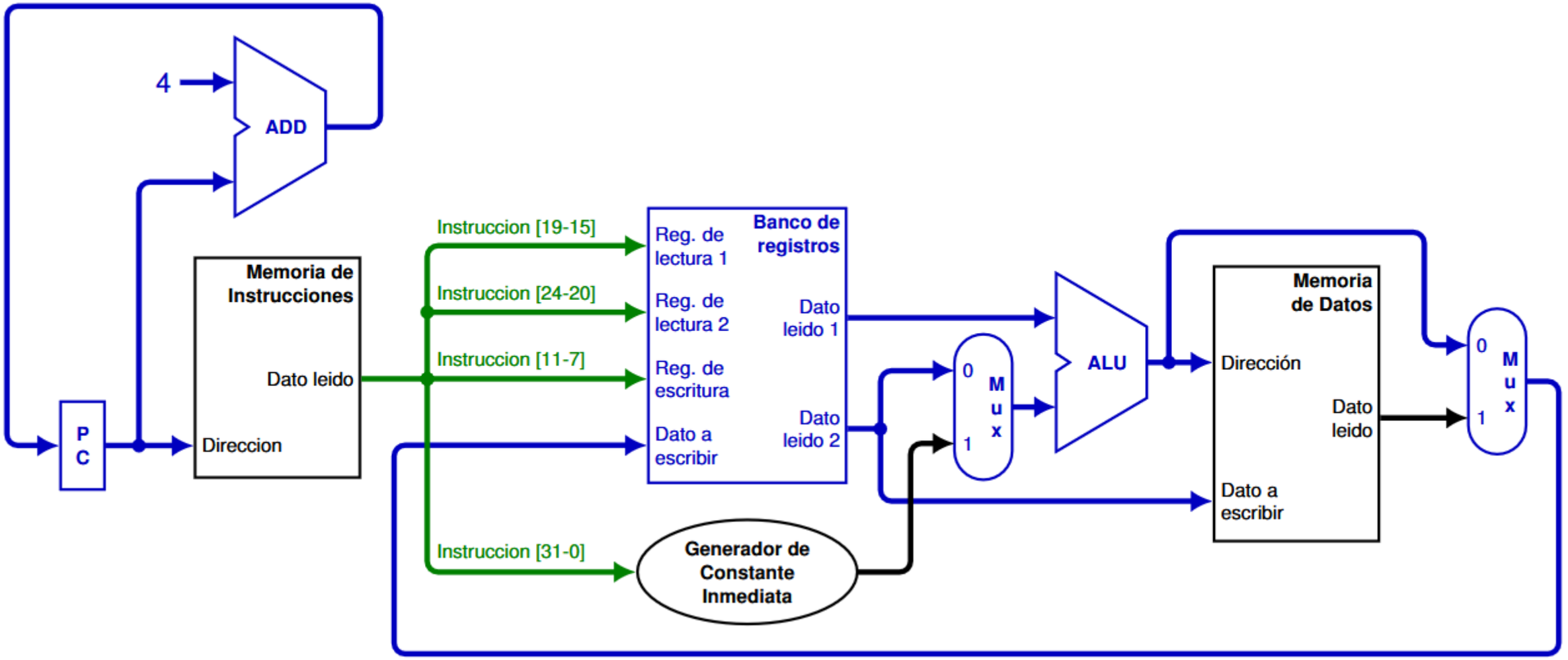
Instrucciones Load/Store

Instrucción	RTN 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 en signo antes que ingrese a la ALU, por lo que agregamos el generador de constantes.
- Load accede a memoria con la dirección calculada en la ALU, y luego guarda el resultado en el banco de registros.
- Store guarda el dato del banco de registros en memoria, en la dirección calculada en la ALU.
- Esta memoria, *¿puede ser la misma que para las instrucciones?*
  - Se necesita una memoria separada para datos.

Camino de Datos Tipo R/Load/Store

- Al conectar los dos nuevos componentes, 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.

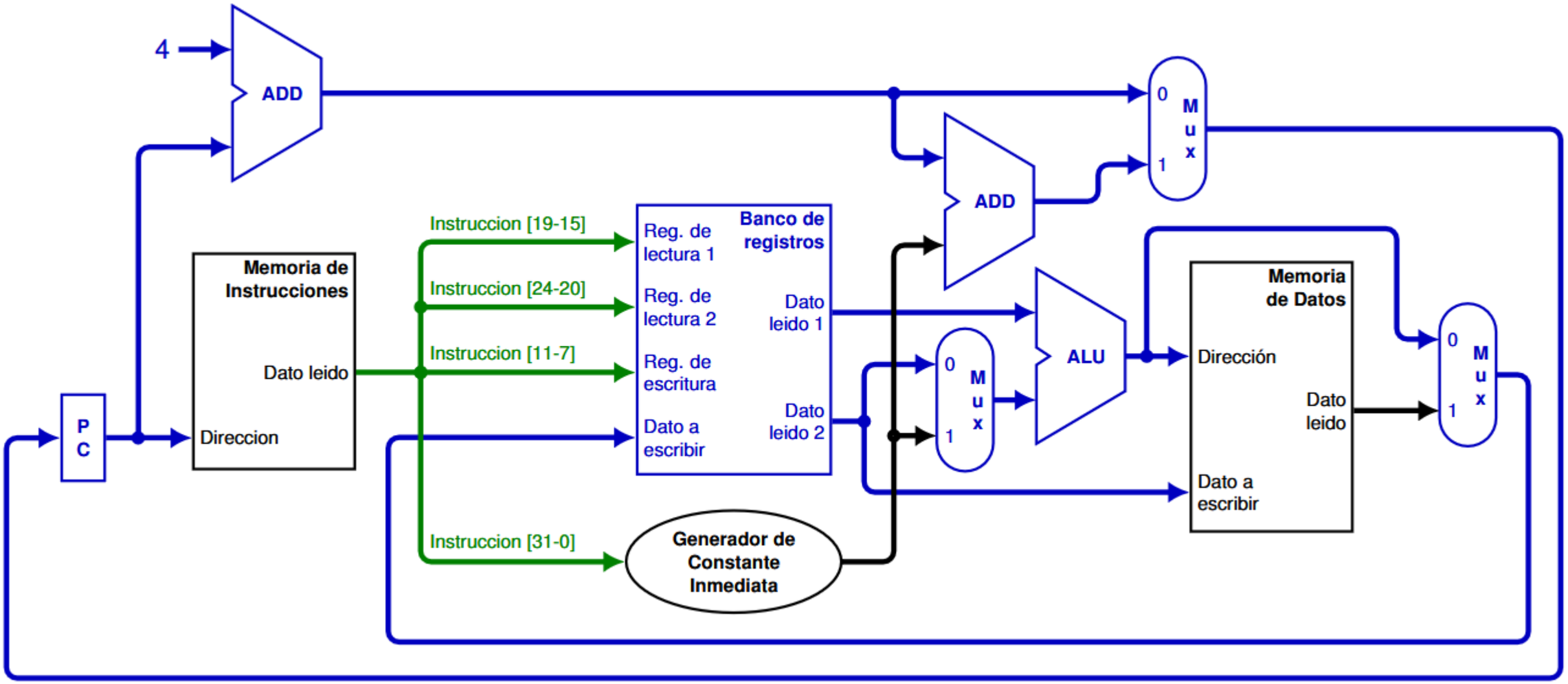


Instrucciones Branch

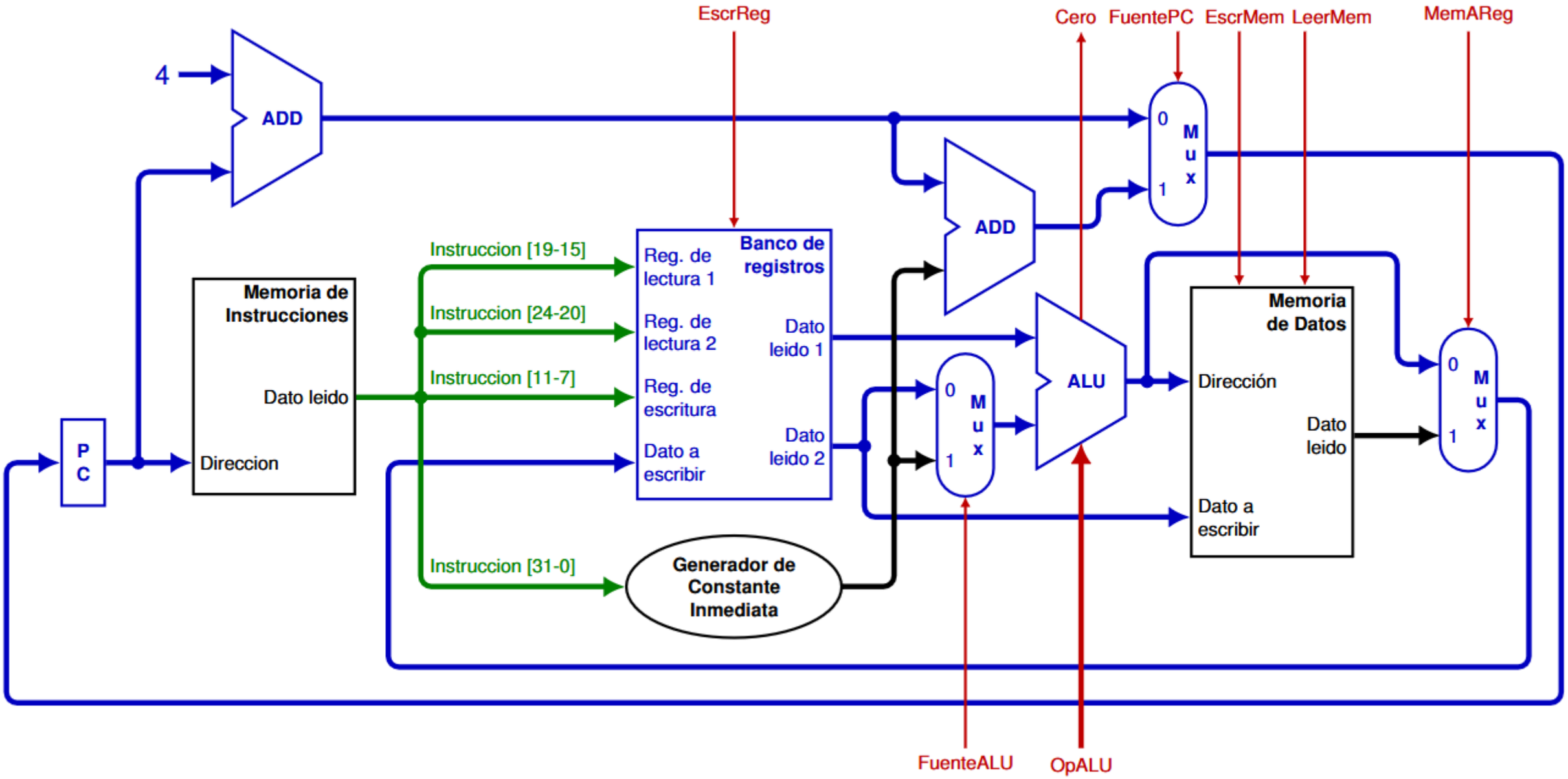
Instrucción	RTN Abstracto
beq rs1, rs2, imm	Si (R[rs1] == R[rs2]) PC ← PC + SignExt(imm)

- 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 y 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 completo (1)



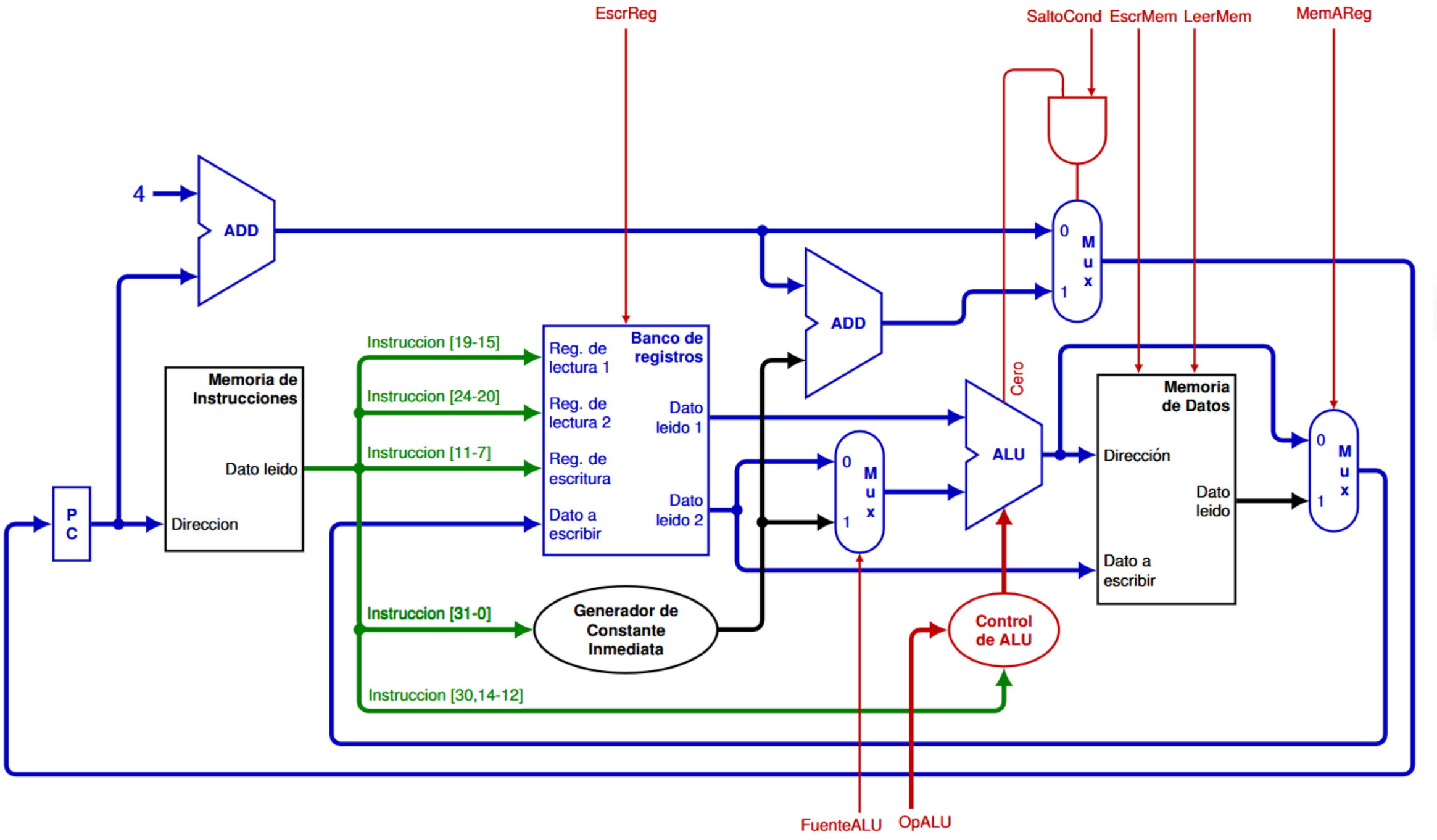
Agregando las señales de control



Control de la ALU y Saltos

- Tomamos la misma decisión que en el procesador multiciclo: separar el control de la ALU del control principal.
  - Para reducir tamaño y retardo del control principal.
- La señal de control FuentePC depende del resultado de la ALU.
  - Lo que haremos será generar una señal de control SaltoCond que indique que es un salto condicional.
  - FuentePC = SaltoCond & Cero

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 únicamente de la instrucción a ejecutar.
  - O sea, del opcode.
- Esta vez, el control principal es **totalmente combinacional**.
  - Se lo implementa con una lógica de dos niveles.
  - La tabla de verdad sería la siguiente:

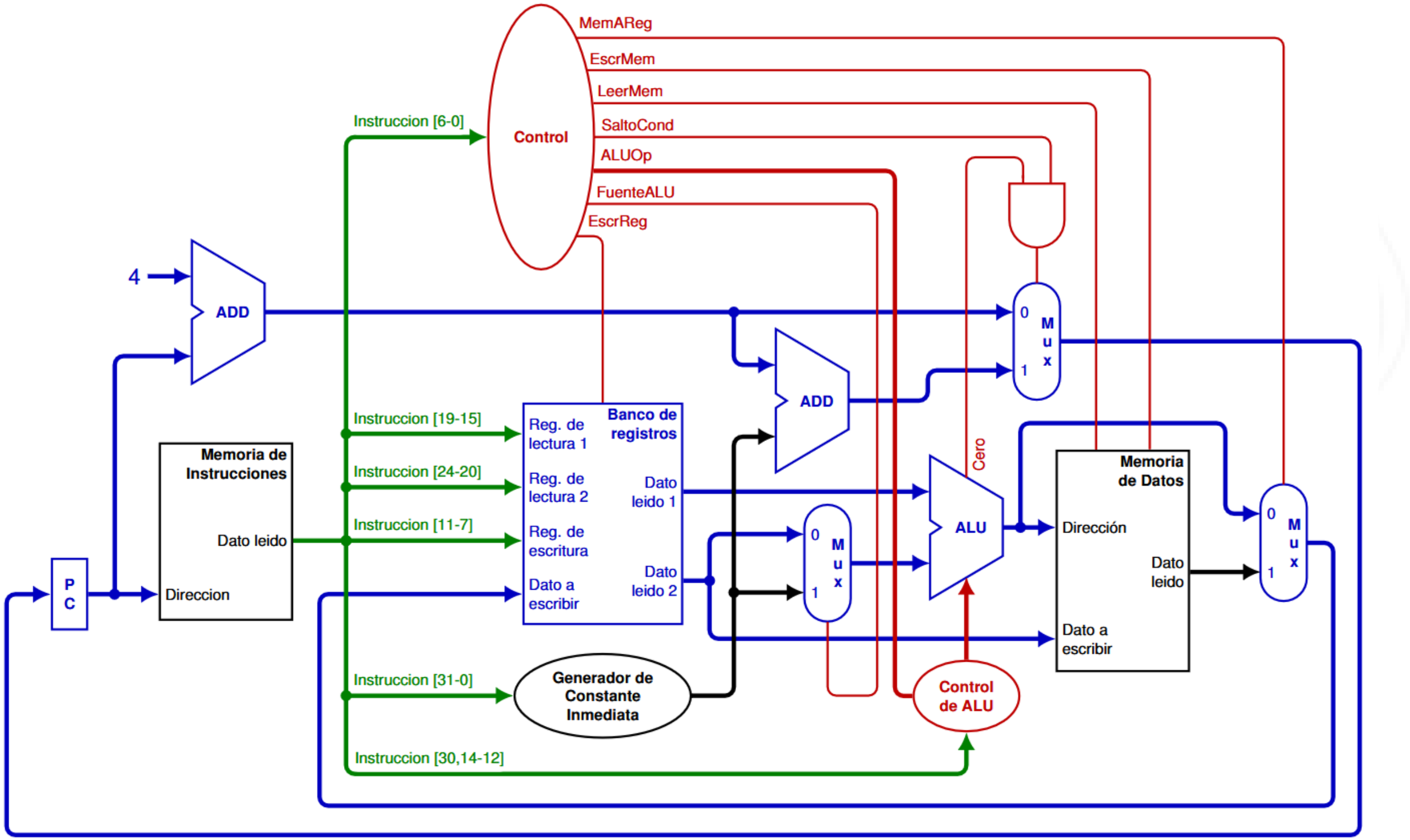
Instrucción	FuenteALU	ALUOp	LeerMem	EscrMem	MemAReg	EscrReg	SaltoCond
`Tipo-R`	`0`	`10`	`0`	`0`	`0`	`1`	`0`
`lw`	`1`	`00`	`1`	`0`	`1`	`1`	`0`
`sw`	`1`	`00`	`0`	`1`	`X`	`0`	`0`
`beq`	`0`	`01`	`0`	`0`	`X`	`0`	`1`

Diseño del Control principal

Instrucción	FuenteALU	ALUOp	LeerMem	EscrMem	MemAReg	EscrReg	SaltoCond
Tipo-R	0	10	0	0	0	1	0
lw	1	00	1	0	1	1	0
sw	1	00	0	1	X	0	0
beq	0	01	0	0	X	0	1

- Agregar nuevas instrucciones implica agregar una nueva fila en la tabla.
  - Y especificar los valores de todas las señales de control para la misma.
  - Con lógica de dos niveles, equivale a agregar una compuerta AND.
- Si alguna nueva instrucción necesita el agregado de un nuevo componente que a su vez necesita una nueva señal de control, implica agregar una nueva columna a la tabla.
  - Y especificar los valores de esa nueva señal de control para todas las instrucciones.
  - Con lógica de dos niveles, equivale a agregar una compuerta OR.

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?*

Comparando ambos diseños

- Consideremos la mezcla de instrucciones de gcc: 22% lw, 11% sw, 48% aritméticas, 19% saltos.
- *¿Cuál de las dos implementaciones tiene mejor performance y por cuánto?*
- *¿Cuál tiene mayor costo?*
- *¿En cuál es más fácil agregar instrucciones?*

- Vemos cómo un mismo ISA puede tener distintas implementaciones.
- Y cómo las decisiones de camino de datos y control (microarquitectura) impactan en CPI y en T.

Conclusiones diseño ciclo único

- ¡Estamos violando uno de nuestros Principios de Diseño! *¿Cuál?*
- Tampoco podemos hacer un reloj de duración variable para cada instrucción.
- Sin embargo…
  - Logramos CPI = 1, que es bastante bueno.
  - El control es mucho más simple.
  - Tiene potencial para mejorar ☺

Resumen final

- Para diseñar un procesador se parte desde el ISA, haciendo el RTN abstracto 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, que es combinacional.
    - Con un control local para la ALU.
    - Ambos implementados en lógica de dos niveles.
- Logramos el objetivo de CPI = 1.
- Pero T muy grande, determinado por la instrucción más larga.
  - No cumple con un Principio de Diseño.