Fundamentos de Computadores

2º Cuatrimestre

2012-2013

fc²

Módulo 10: Diseño del Procesador

- ✓Introducción al diseño de un procesador
- ✓Procesador multiciclo

Bibliografía recomendada:

Estructura y Diseño de Computadores: Volumen 1

D. Patterson y J. Hennessy, ed. Reverté, 2000

Capítulo 5: "El procesador: camino de datos y control"

Introducción al diseño de un procesador

- **Paso 1**: **Analizar el repertorio de instrucciones** para obtener los requisitos de la ruta de datos
  - La ruta de datos debe incluir tantos **elementos de almacenamiento** como registros sean visibles por el programador. Además puede tener otros elementos de almacenamiento transparentes.
  - La ruta de datos debe incluir tantos tipos de **elementos operativos** como tipos de operaciones de cálculo se indiquen en el repertorio de instrucciones
  - El significado de cada instrucción vendrá dado por un conjunto de transferencias entre registros. La ruta de datos debe ser capaz de soportar dichas transferencias.
- **Paso 2: Establecer la temporización**
  - **Multiciclo**: cada instrucción se ejecuta en varios ciclos de reloj.

Introducción al diseño de un procesador

- **Paso 3**: **Seleccionar el conjunto de módulos** (de almacenamiento, operativos e interconexión) que forman la ruta de datos.
- **Paso 4**: **Ensamblar la ruta de datos** de modo que se cumplan los requisitos impuestos por el repertorio, **localizando los puntos de control.**
- **Paso 5**: **Determinar los valores de los puntos de control** analizando las transferencias entre registros incluidas en cada instrucción.
- **Paso 6: Diseñar la lógica de control.**

Introducción al diseño de un procesador

- Analizar el repertorio de instrucciones
  - 68000 - CISC
  - Intel (X86) - CISC
  - MIPS – RISC
  - ARM – RISC

Introducción al diseño de un procesador

- Arquitectura MIPS
  - **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
  - Creado en 1981 por Hennessy
  - Producido por Silicon Graphics (SGI)
  - Utilizado por:
    - TiVo
    - Windows CE
    - Cisco routers
    - Nintendo 64
    - Sony PlayStation, PlayStation 2, PlayStation Portable.

Introducción al diseño de un procesador

- **R2000** (1985): procesador con operaciones multiciclo de multiplicación y división, dentro de un coprocesador matemático, los resultados de estas operaciones eran conseguidos a través de instrucciones particulares. Además, aunque los registros eran de 32 bits podían utilizarse como de 64 para doble precisión.
- **R3000** (1988): añade una cache de 32KB (más tarde 64KB), una MMU para gestionar la memoria virtual, fue el primer procesador MIPS que tuvo éxito en el mercado (se vendieron 1 millón). Por ejemplo, fue utilizado en la **Play Station**, además en los primeros portátiles que utilizaban **Windows CE**.
- **R4000** (1991): aumenta el conjunto de instrucciones a un procesador de 64 bits añadiendo la FPU dentro del chip. Todo esto permite un alto ciclo de reloj (a cambio se reducen las caches a 8KB) y una super-segmentación. **Nintendo64** utiliza una CPU basada en este diseño, así como los primeros routers de **Cisco (36x0 y 7x00)**
- **R8000** (1994): fue el primer procesador super-escalar que podía trabajar a la vez con dos ALUs y dos memorias. Su unidad FPU era perfecta para cálculos científicos, pero duró en el mercado sólo un año.
- **R10000** (1995): tenía caches de 32 KB y su mayor innovación fue la utilización de la ejecución fuera de orden.
- **R20000** (2006): 2 cores … no ha llegado a producirse

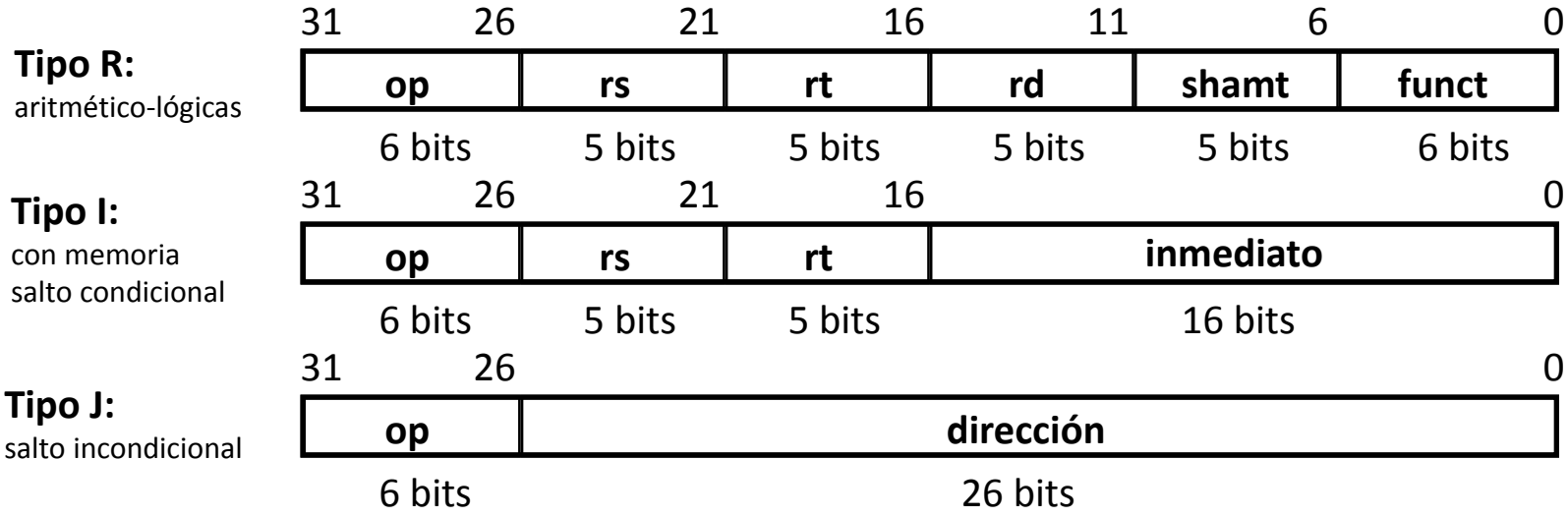
Cómo creamos un repertorio de instrucciones

- ¿Qué necesitamos saber?
  - Ancho de la instrucción: número máximo de bits que podemos utilizar para codificar la instrucción (32 bits en el caso del MIPS)
  - Tipos de instrucciones
  - Número de instrucciones de cada tipo
  - Cómo vamos a conseguir los datos con los que tienen que trabajar las instrucciones: modos de direccionamiento
  - Requisitos específicos

Introducción al diseño de un procesador

- Arquitectura MIPS
  - Todas las instrucciones tienen el mismo tamaño (32 bits)
  - Todas las instrucciones tienen formato similar
    - Código de operación 6 bits
    - Sólo existen 3 formatos de instrucciones
  - Accesos a memoria sencillos
    - Indirecto registro con desplazamiento

Introducción al diseño de un procesador



El significado de los campos es:

- **op**: identificador de instrucción
- **rs**, **rt**, **rd**: identificadores de los registros fuentes y destino
- **shamt**: cantidad a desplazar (en operaciones de desplazamiento)
- **funct**: selecciona la operación aritmética a realizar
- **inmediato**: operando inmediato o desplazamiento en direccionamiento a registro-base
- **dirección**: dirección destino del salto

Instrucciones formato tipo-R

- Los 32 bits que describen la instrucción se dividen en los campos:

op	rs	rt	rd	No Usado	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Se destinan los primeros 6 bits de la instrucción para indicar el tipo de operación que hace ésta → Campo OP, llamado Código de Operación
- En las instrucciones **A-L** siempre se pone el OP a 0 (6 bits a 0) → El computador entenderá que se trata de una instrucción que realiza una operación aritmética o lógica (suma, resta, or, etc.)

Instrucciones formato tipo-R

- Tras los 6 bits de OP, se destinan 3x5 bits para indicar el identificador de los registros sobre los que se operará:
  - rs → Primer registro operando fuente
  - rt → Segundo registro operando fuente
  - rd → Registro operando destino
- 5 bits para cada uno son suficientes, pues como hemos visto el Banco de Registros del MIPS tiene 32 registros.

Instrucciones formato tipo-R

- Los 5 bits siguientes no se utilizan
- Los 6 últimos bits de la instrucción (campo funct) se destinan a indicar la variante específica de operación aritmética o lógica:
  - add → funct=32 (100000)
  - sub → funct=34 (100010)
  - and → funct=36 (100100)
  - or → funct=37 (100101)
  - slt → func=42 (101010)

Instrucciones formato tipo-I

- Los 32 bits que describen la instrucción se dividen en los campos:

op	rs	rt	Desplazamiento/Inmediato
6 bits	5 bits	5 bits	16 bits

- Se destinan los primeros 6 bits de la instrucción para indicar el tipo de operación → Campo OP, llamado Código de Operación
- Valor de OP en instrucciones tipo I:
  - lw → OP=35 (100011)
  - sw → OP= 43 (101011)
  - beq → OP= 4 (000100)

Instrucciones formato tipo-I

- Tras los 6 bits de OP, se destinan 2x5 bits para indicar el identificador de los registros sobre los que se operará:
  - rs → Registro fuente/base
    - registro base (a sumar al desplazamiento) en lw,sw
  - rt → Registro fuente/destino
    - CUIDADO: es fuente para sw, beq
- 5 bits para cada uno son suficientes, pues como hemos visto el Banco de Registros del MIPS tiene 32 registros

Instrucciones formato tipo-I

- El resto de la instrucción (16 bits), lo utilizamos para
  - el desplazamiento a partir de la dirección base
    - rs en lw/sw
    - PC en saltos (expresado en número de palabras)
- En ensamblador sólo etiquetas → En lenguaje máquina se sustituyen por números

Procesador multiciclo

- **Instrucciones con referencia a memoria**:
  - lw rt, inmed(rs) rt ← Memoria(rs + SignExt(inmed)) , PC ← PC + 4
  - sw rt, inmed(rs) Memoria(rs + SignExt(inmed)) ← rt , PC ← PC + 4
- **Instrucciones aritmético-lógicas con operandos en registros**:
  - add rd, rs, rt rd ← rs + rt, PC ← PC + 4
  - sub rd, rs, rt rd ← rs - rt , PC ← PC + 4
  - and rd, rs, rt rd ← rs and rt , PC ← PC + 4
  - or rd, rs, rt rd ← rs or rt , PC ← PC + 4
  - slt rd, rs, rt (si (rs < rt) entonces (rd ← 1)
    en otro caso (rd ← 0)), PC ← PC+4
- **Instrucciones de salto condicional**:
  - beq rs, rt, inmed si (rs = rt) entonces (PC ← PC + 4 + 4·SignExp(inmed))
    en otro caso PC ← PC + 4

Procesador multiciclo

- La ejecución de cada instrucción se divide en varias fases:

1. Búsqueda de la instrucción (*fetch*)
2. Decodificación (*deco*)
3. Ejecución (*ex*)
4. Acceso a memoria (*mem*)
5. Almacenamiento del resultado (*write back*)

- El tiempo de ejecución de cada fase es 1 ciclo de reloj.
- Cada instrucción tarda un número distinto de ciclos, según las fases que deba ejecutar

Procesador multiciclo

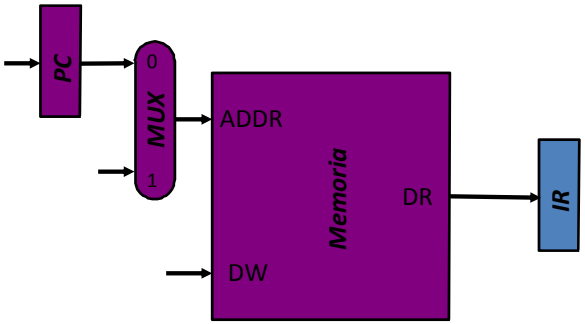
- Las fases de búsqueda de instrucción y decodificación son comunes a todas las operaciones
- Las fases de ejecución y almacenamiento del resultado son distintas para las operaciones aritmético-lógicas, lw, sw y salto.
- La fase de acceso a memoria sólo la realizan las operaciones lw.
- La operación de load tarda 5 ciclos de reloj y el resto de operaciones 4.

Diseño de la ruta de datos multiciclo

- ¿Qué HW necesitamos para implementar cada una de las fases de ejecución de una operación?
  - Nótese que las distintas fases de la ejecución de una instrucción pueden compartir el HW de la ruta de datos ya que se ejecutan en ciclos distintos

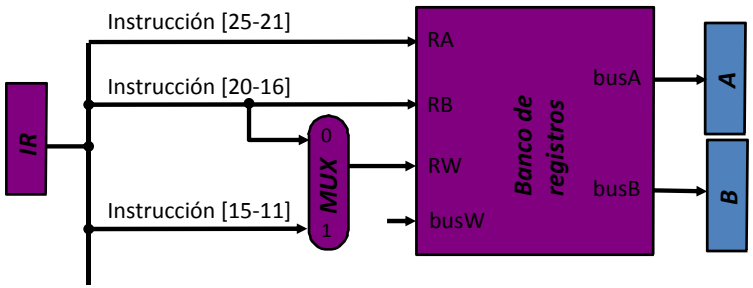
Diseño de la ruta de datos multiciclo

- Fase de **búsqueda de la instrucción**:
  - Lectura de la instrucción ubicada en la dirección de la **memoria** indicada por el **contador de programa**.
  - Hardware necesario:
    - **Contador de programa (*PC*)**
    - **Memoria**
    - **Registro de instrucción (*IR*)**



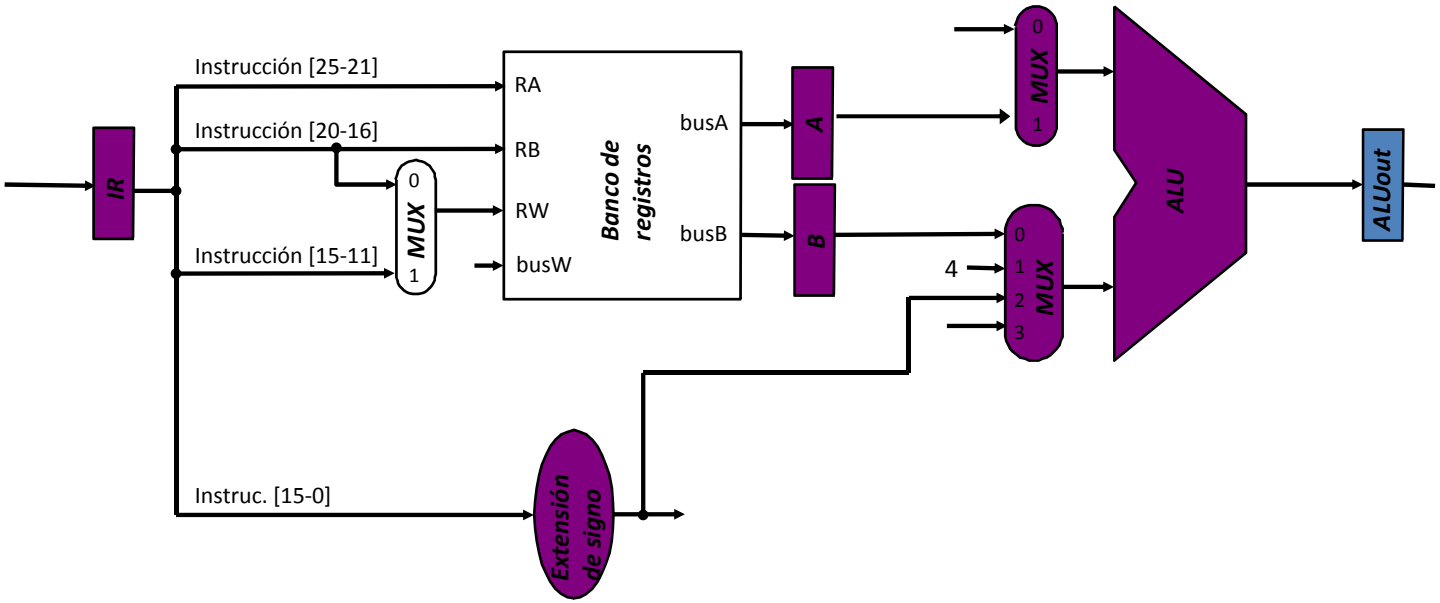
Diseño de la ruta de datos multiciclo

- Fase de **decodificación de la instrucción**:
  - Lectura de los **operandos** necesarios del **banco de registros**.
  - Hardware necesario:
    - **Banco de registros**
    - **2 registros de operandos *A* y *B***



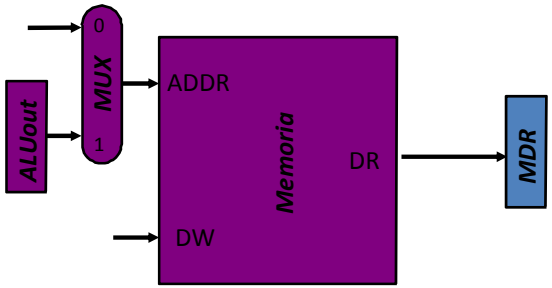
Diseño de la ruta de datos multiciclo

- Fase de **ejecución de la instrucción**:
  - Ejecución en la ALU de la operación necesaria.
  - Hardware necesario:
    - **ALU**
    - **Registro para salvar el resultado de la ALU (*ALUOut*)**
    - **Extensor de signo** (únicamente para lw y sw)



Diseño de la ruta de datos multiciclo

- Fase de **acceso a memoria** (sólo instrucción *lw*):
  - Lectura de un dato de la memoria.
  - Hardware necesario:
    - **Registro para almacenar el dato leído (*MDR*)**

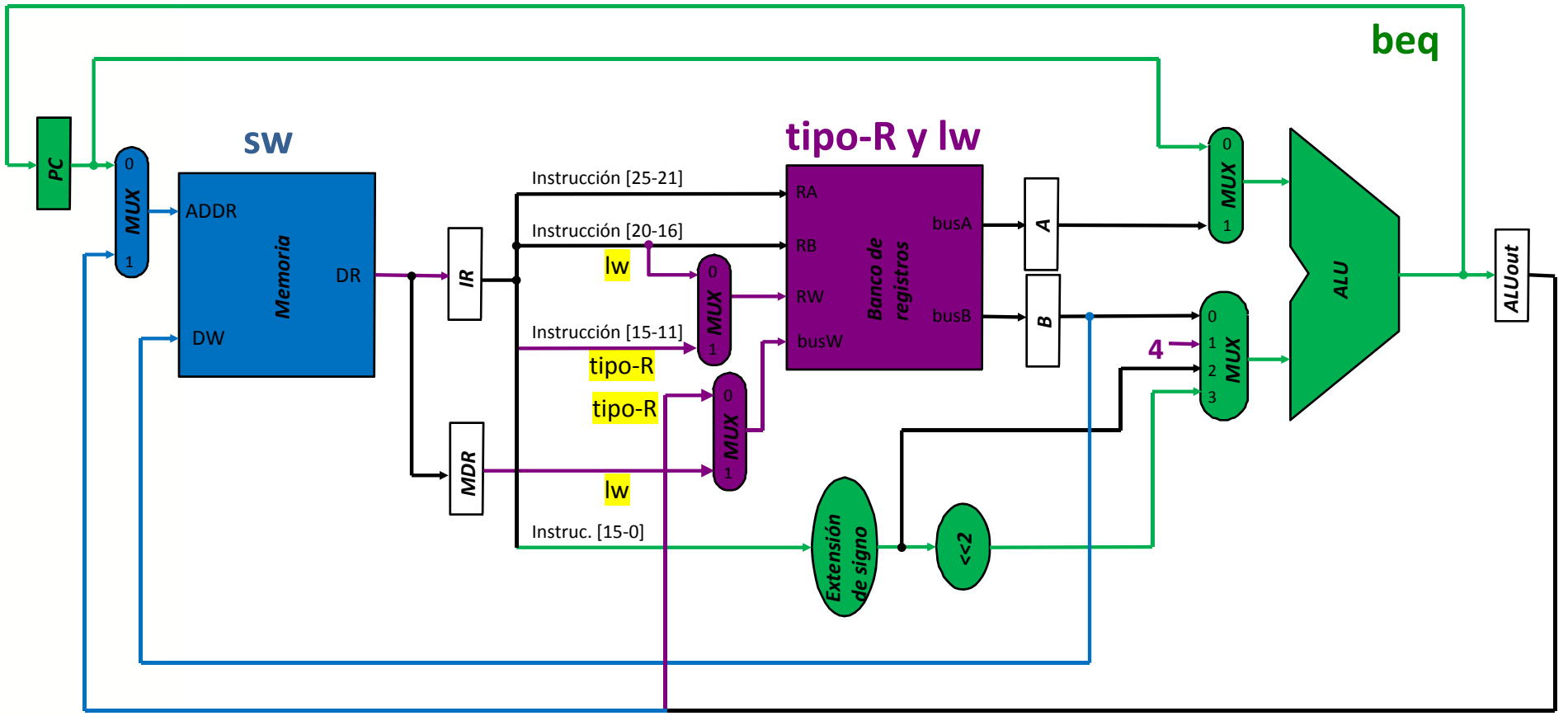


Diseño de la ruta de datos multiciclo

- Fase de **almacenamiento del resultado**:
  - Escritura del resultado en el banco de registros (instrucciones aritmético-lógicas y *lw*) o en la memoria (*sw*).
  - La instrucción de salto actualiza el valor de *PC* si se cumple la condición del salto.
  - Hardware necesario:
    - **Desplazador a la izquierda:** para implementar la multiplicación por 4 (necesaria en el salto).

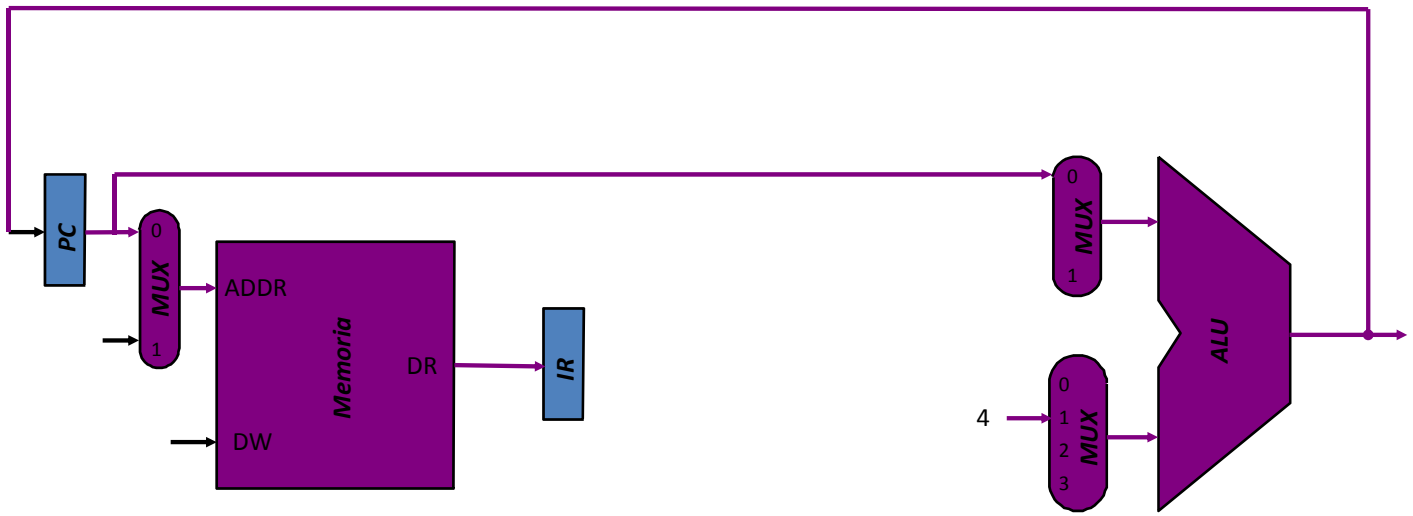
Diseño de la ruta de datos multiciclo

- Fase de **almacenamiento del resultado**:



Diseño de la ruta de datos multiciclo

- ¿Y el cálculo de la siguiente instrucción?
  - Consiste en la actualización del puntero a la siguiente instrucción en secuencia. Todos los programas siguen una **ejecución secuencial** (excepto cuando se ejecutan saltos) :
    - Actualizar el **contador de programa** sumando 4 por ser una memoria direccionable por bytes y una arquitectura con tamaño de palabra de 32 bits
  - Se realiza durante la fase de búsqueda de instrucción y se utiliza la ALU para hacer la suma

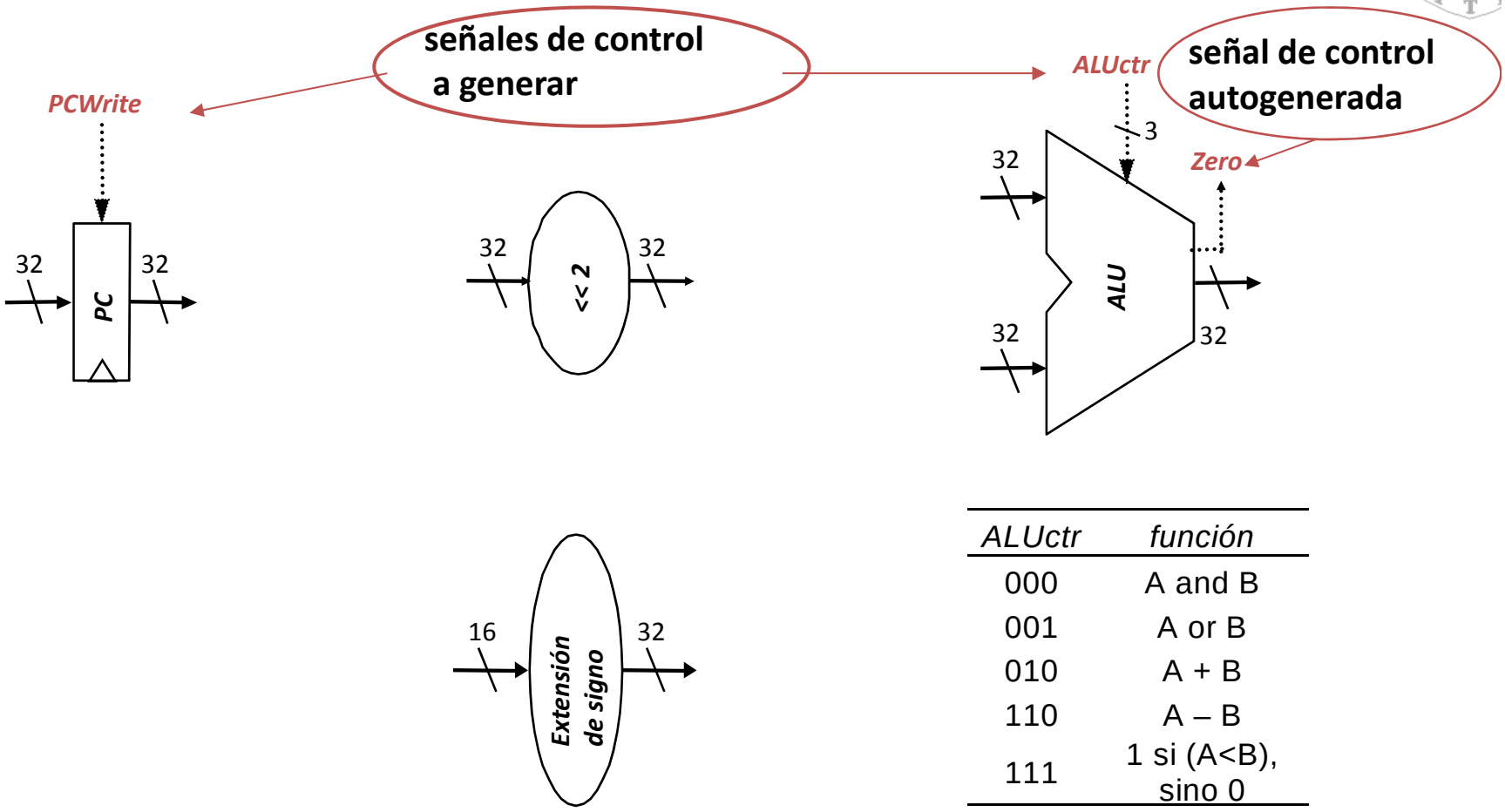


Fase de búsqueda de instrucción

Diseño de la ruta de datos multiciclo

- Para implementar el subconjunto del repertorio del MIPS en una implementación multiciclo se requieren (el tamaño de palabra es de 32 bits):
  - **Memoria de instrucciones y datos**
  - **32 registros de datos:** visibles por el programador.
  - **Contador de programa**
  - **ALU:** capaz de realizar suma, resta, and, or, comparación de mayoría e indicación de que el resultado es cero (para realizar la comparación de igualdad mediante resta)
  - **Extensor de signo:** para adaptar el operando inmediato de 16 bits al tamaño de palabra.
  - **Desplazador a la izquierda:** para implementar la multiplicación por 4 (necesaria en el salto).

Diseño de la ruta de datos multiciclo

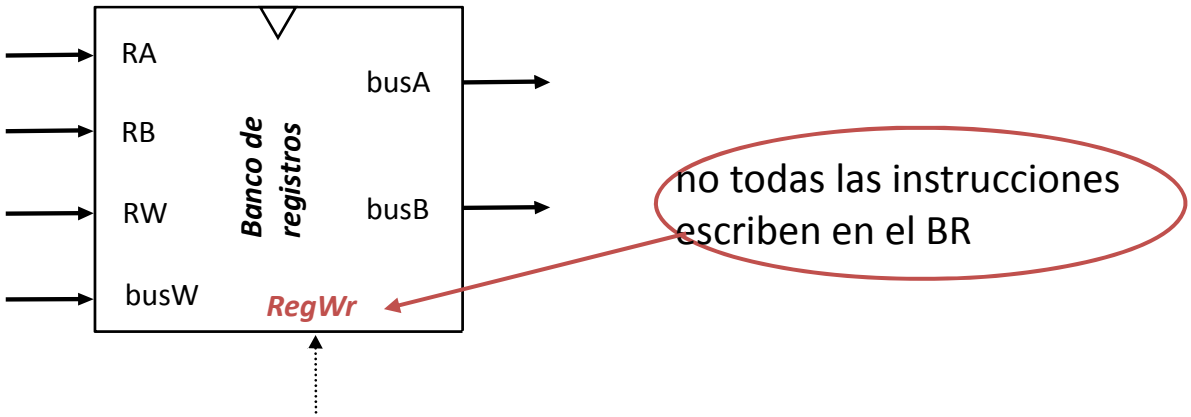


ALUctr	*función*
000	A and B
001	A or B
010	A + B
110	A – B
111	1 si (A<B), sino 0

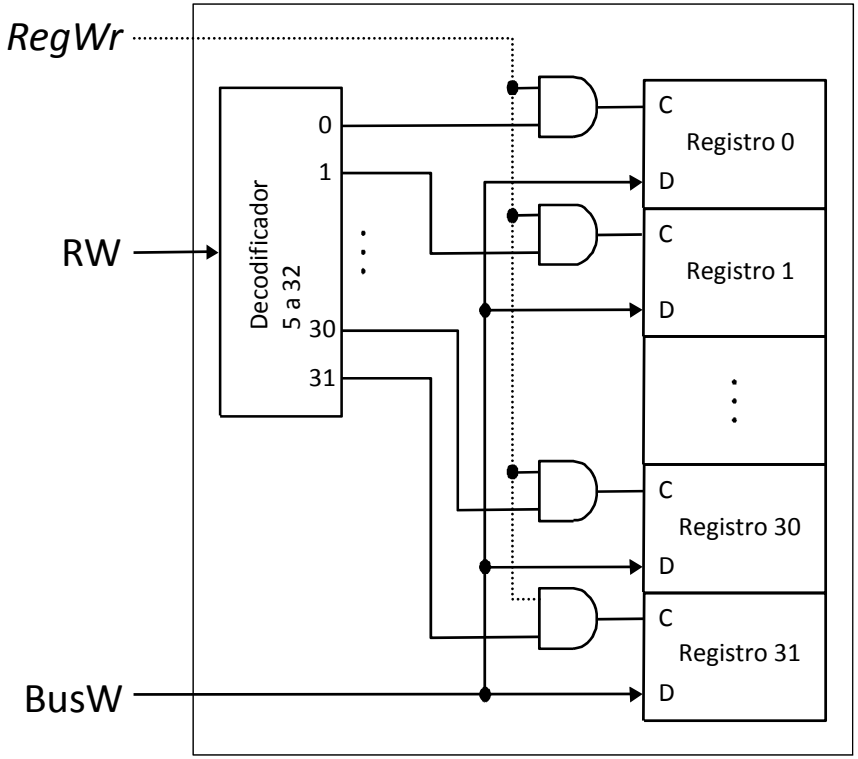
¿Cómo es una ALU por dentro?

Diseño de la ruta de datos multiciclo

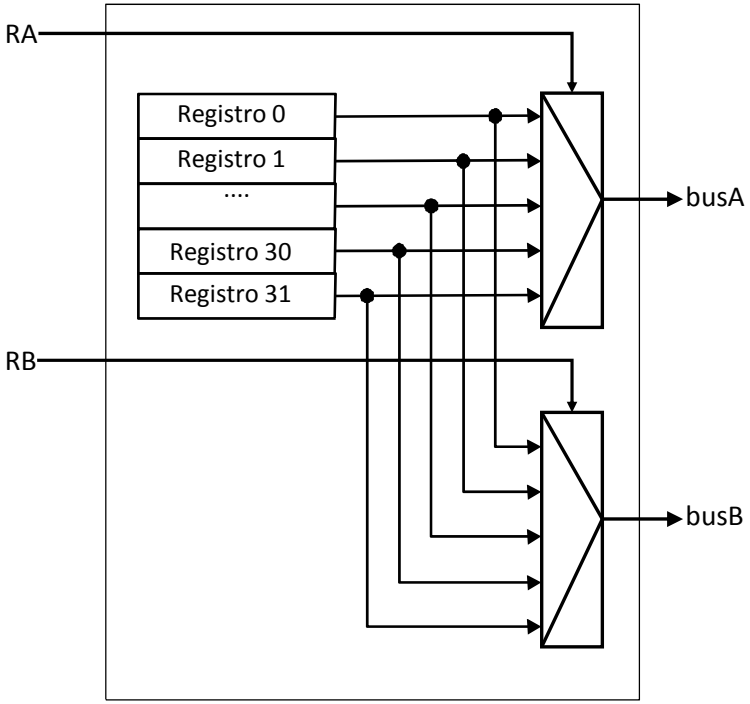
- Los 32 registros se almacenan en un **banco de registros**. Dado que en las instrucciones de tipo R, se requiere un acceso simultáneo a 3 registros:
  - 2 salidas de datos de 32 bits
  - 1 entradas de datos de 32 bits
  - 3 entradas de 5 bits para la identificación de los registros
  - 1 entrada de control para habilitar la escritura sobre uno de los registros
  - 1 puerto de reloj (sólo determinante durante las operaciones de escritura, las de lectura son combinacionales)



Diseño de la ruta de datos multiciclo



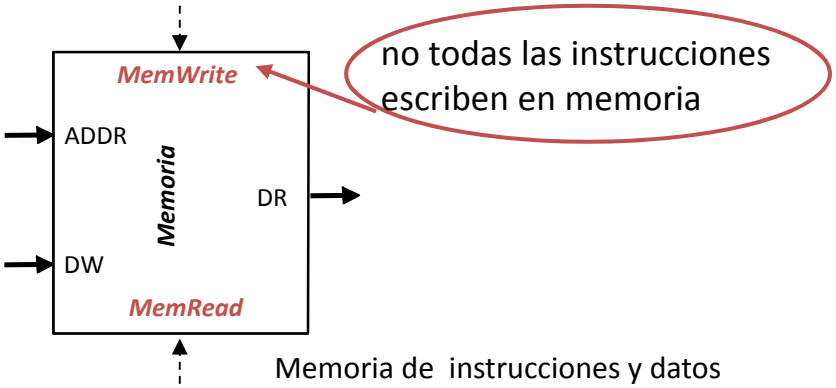
Mecanismo de Escritura



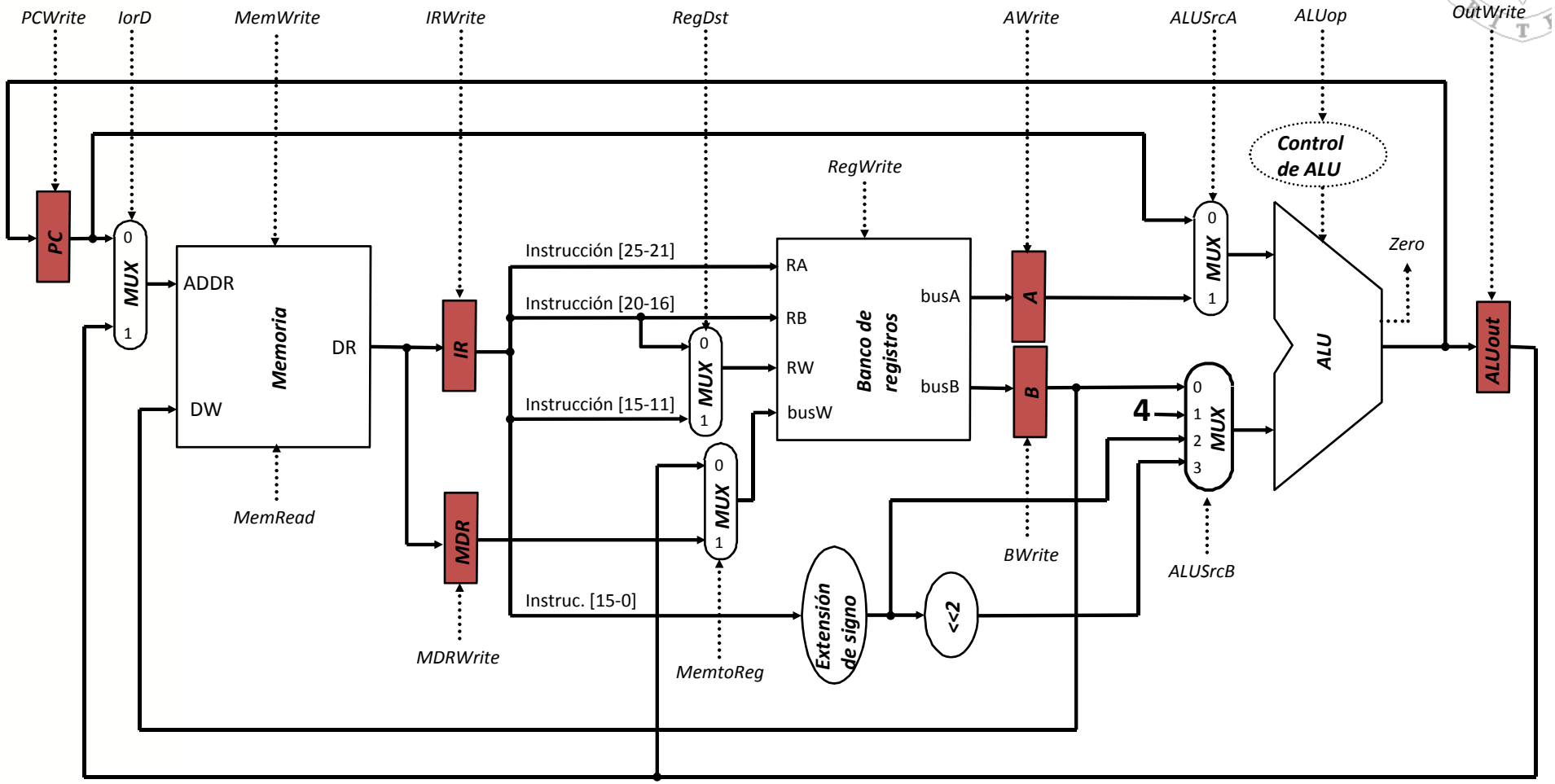
Mecanismo de Lectura

Diseño de la ruta de datos multiciclo

- La memoria tendrá un comportamiento idealizado.
  - "Integrada" dentro de la CPU.
  - Direccionable por bytes, pero capaz de aceptar/ofrecer 4 bytes por acceso
    - 1 entrada de dirección
    - 1 salida de datos de 32 bits
    - 1 entrada de datos de 32 bits
  - 1 entrada de control para seleccionar el tipo de operación (lectura/escritura)
  - Se supondrá que se comporta temporalmente como el banco de registros (síncronamente) y que tienen un tiempo de acceso menor que el tiempo de ciclo

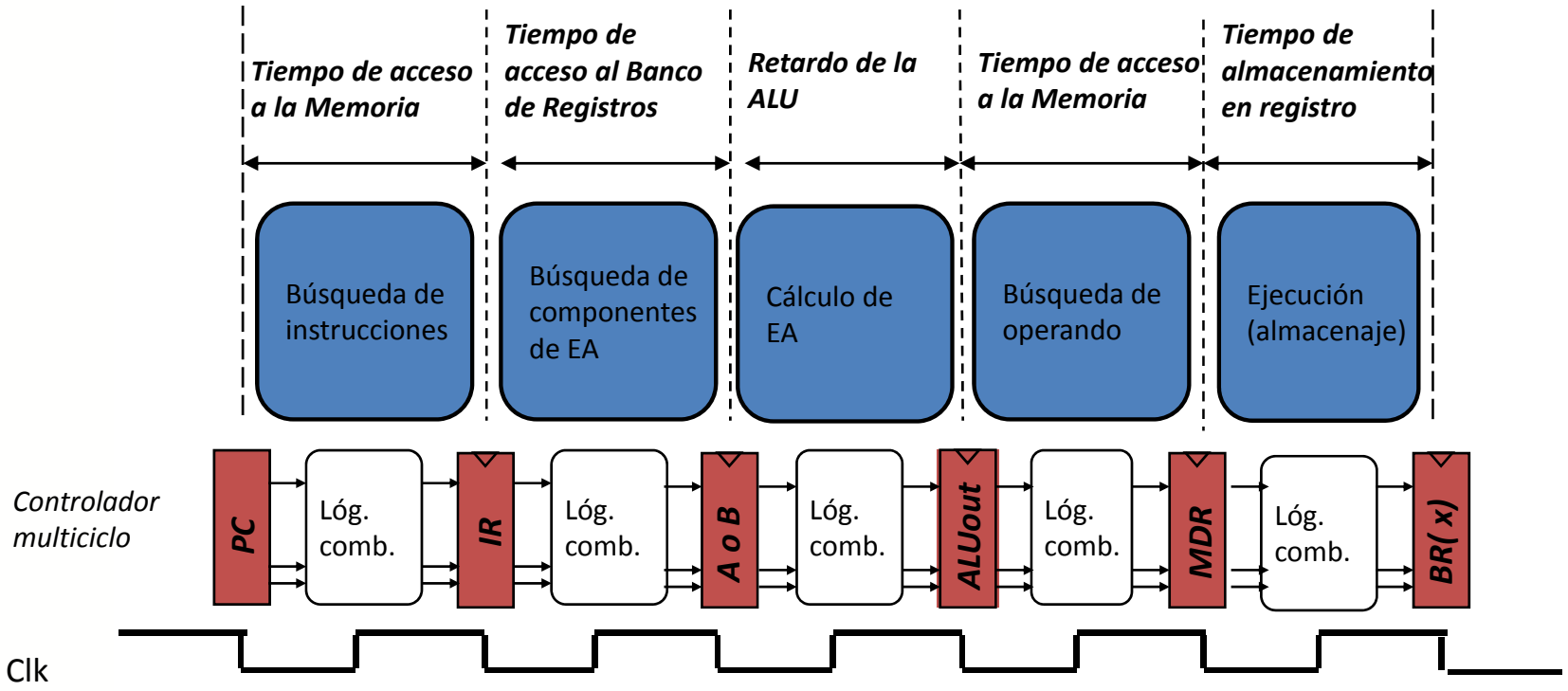


Diseño de la ruta de datos multiciclo

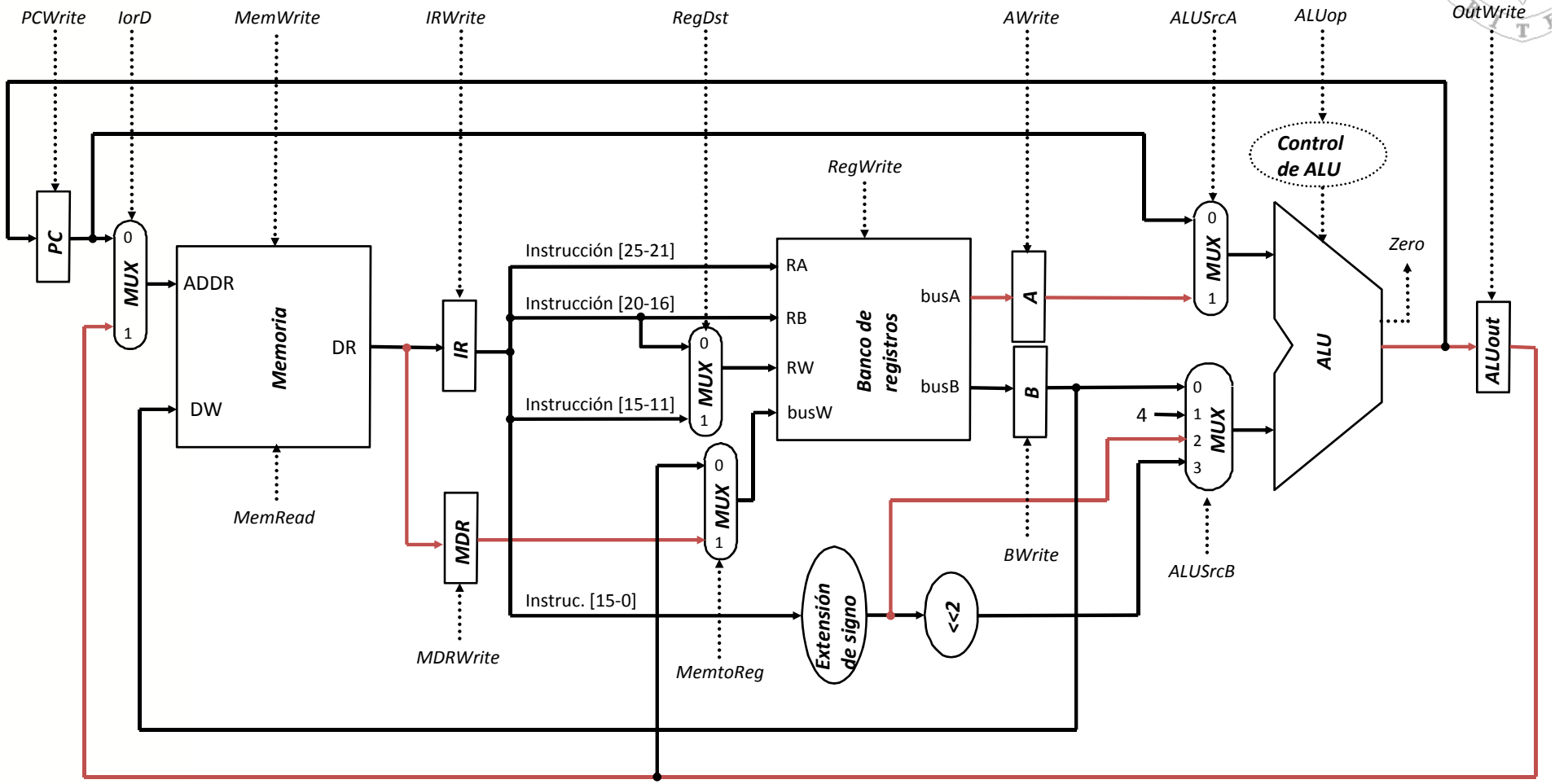


Diseño de la ruta de datos multiciclo

- Temporización de la instrucción *lw*:

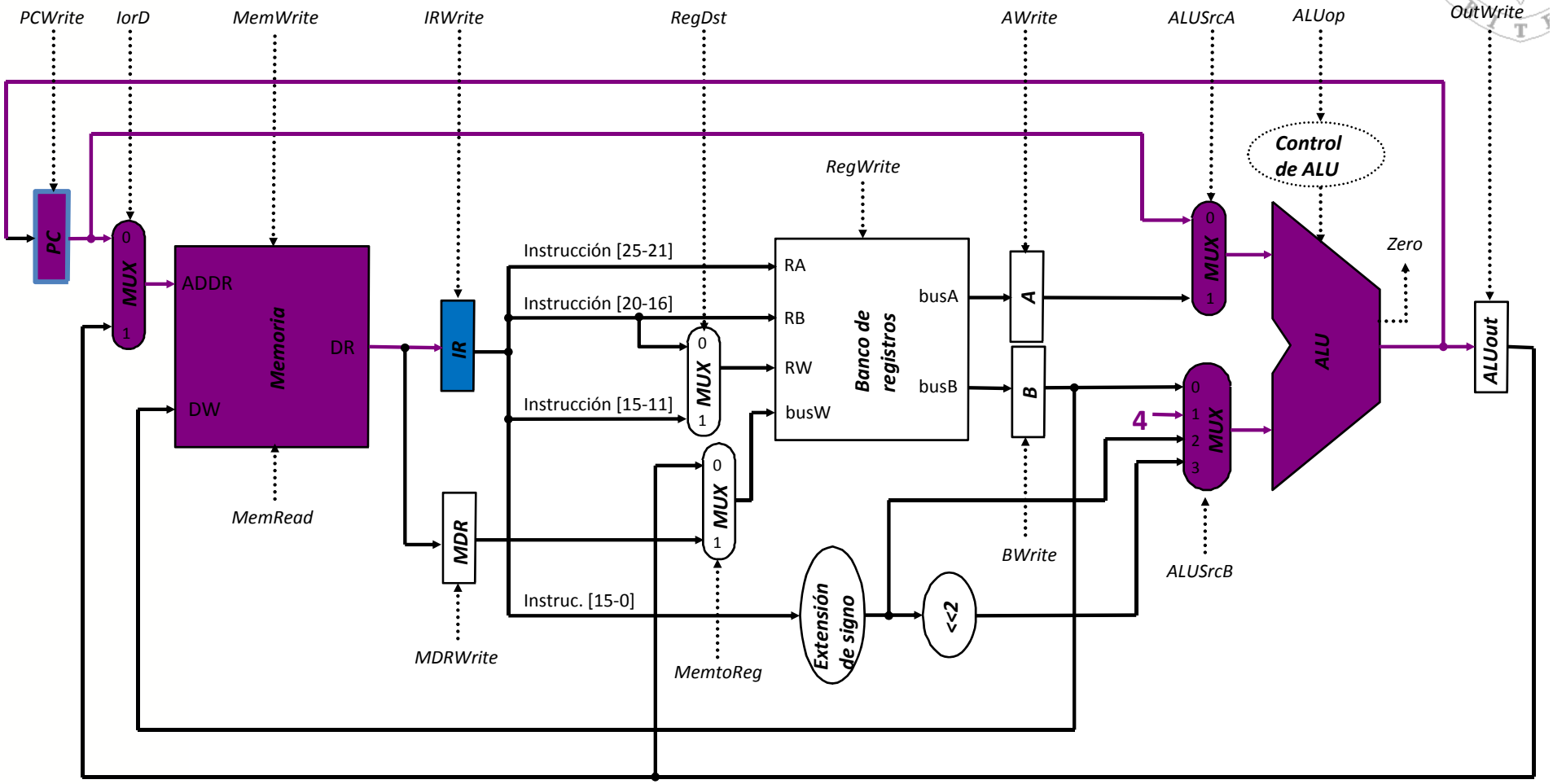


Diseño de la ruta de datos multiciclo



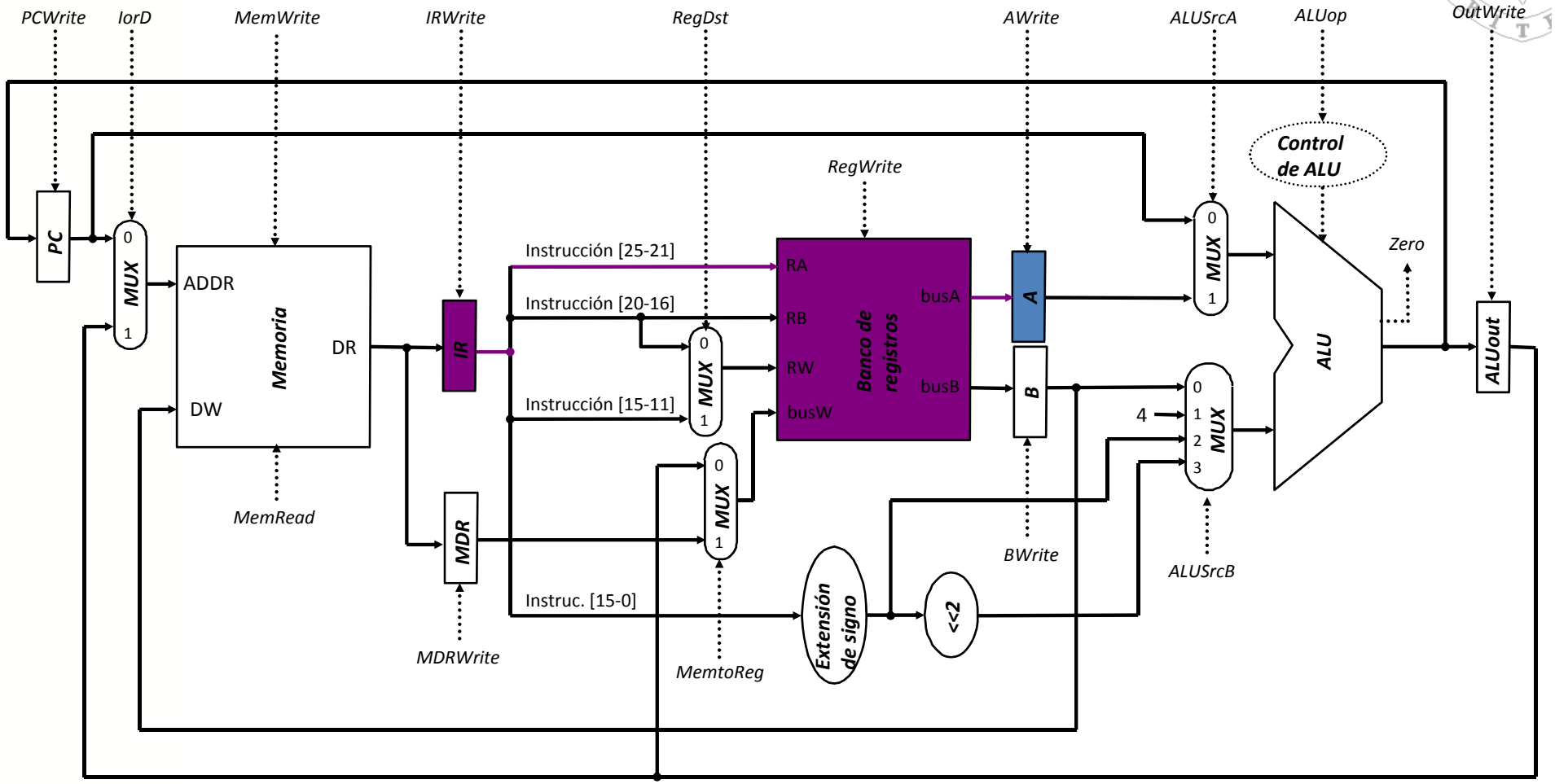
Instrucción lw

Diseño de la ruta de datos multiciclo



Instrucción lw
Etapa de carga de la instrucción – instruction fetch

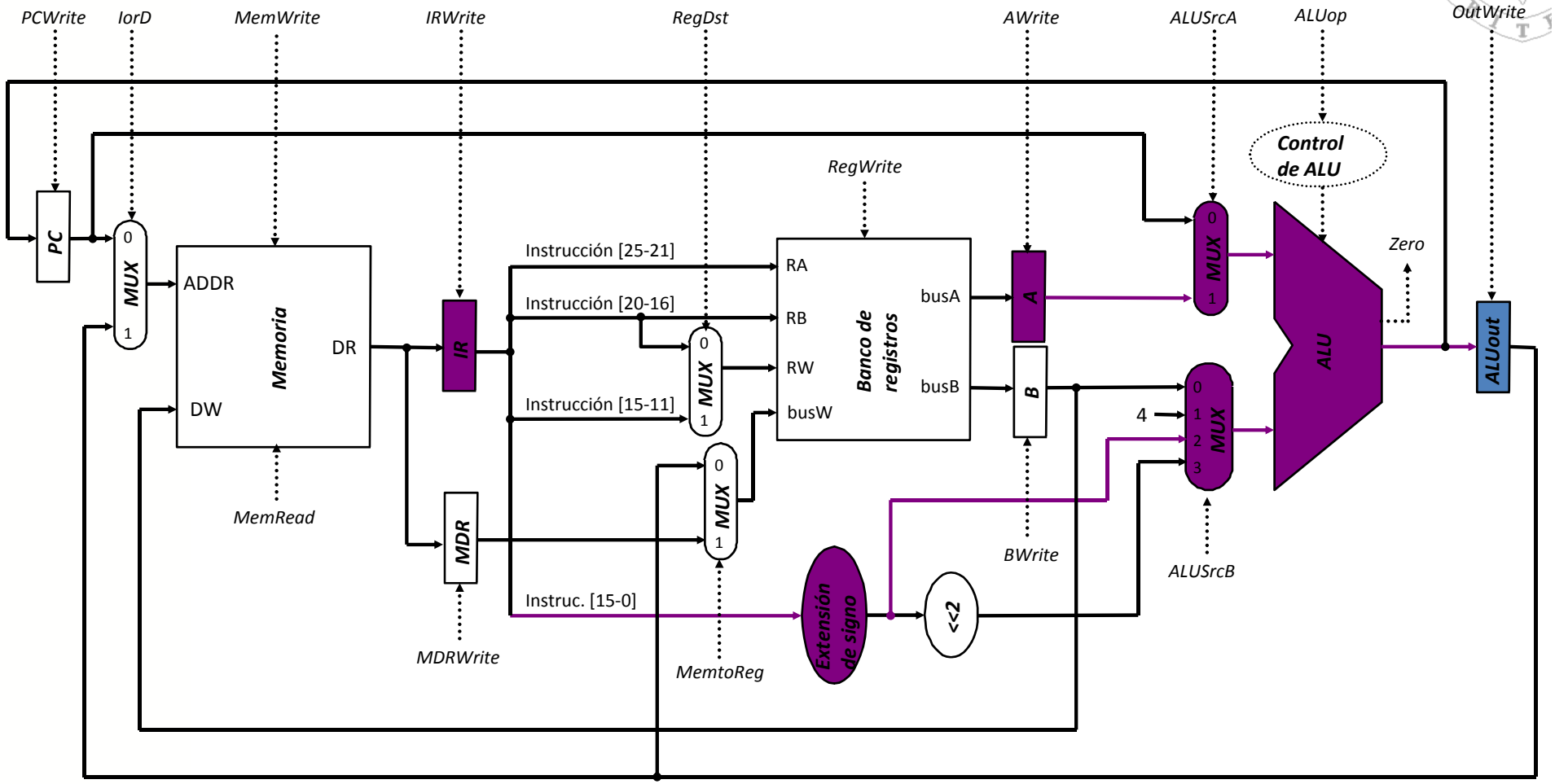
Diseño de la ruta de datos multiciclo



Instrucción lw

Etapa de decodificación de la instrucción y búsqueda de los registros – instruction deco

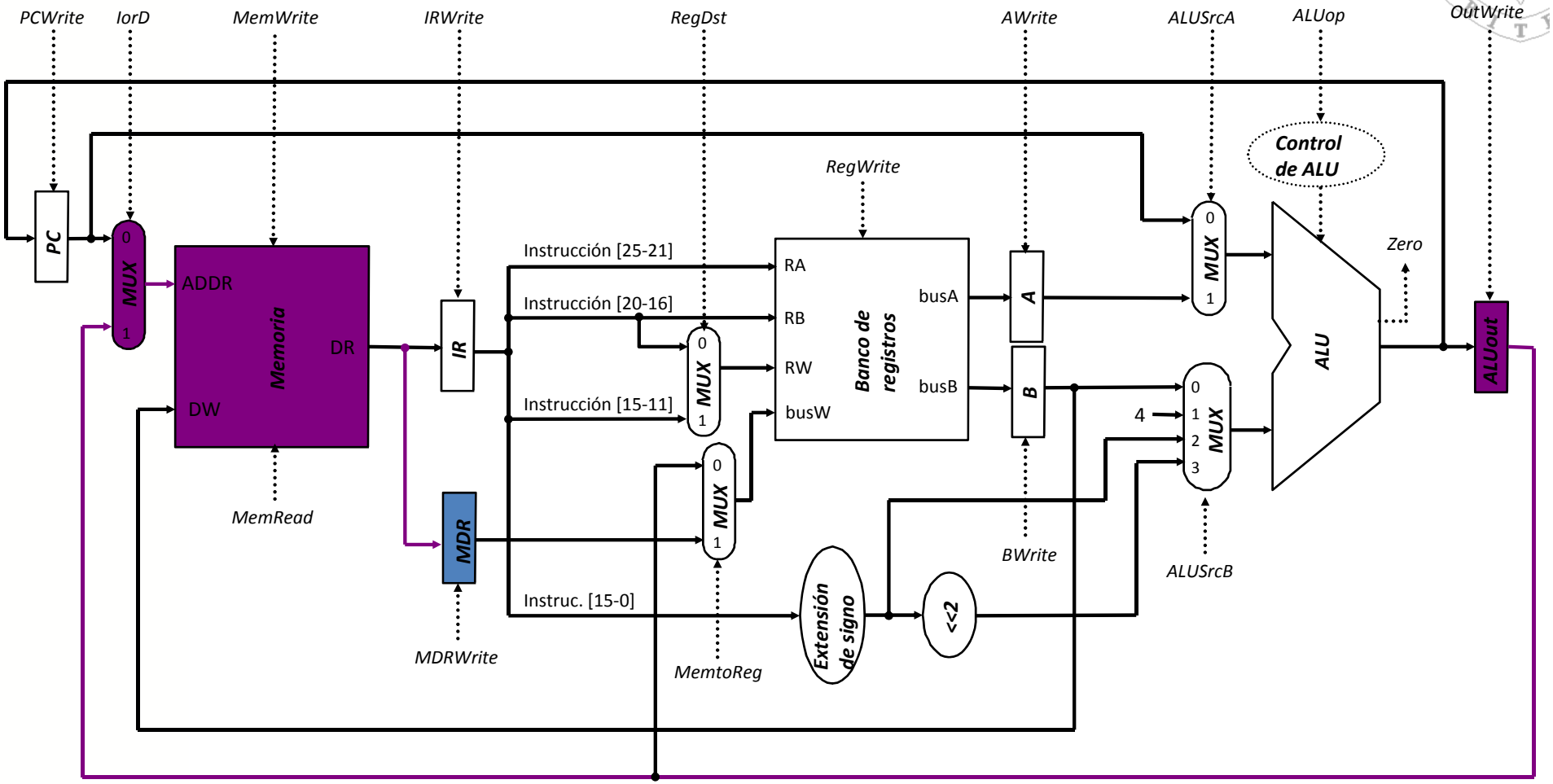
Diseño de la ruta de datos multiciclo



Instrucción lw

Cálculo de la dirección – execution

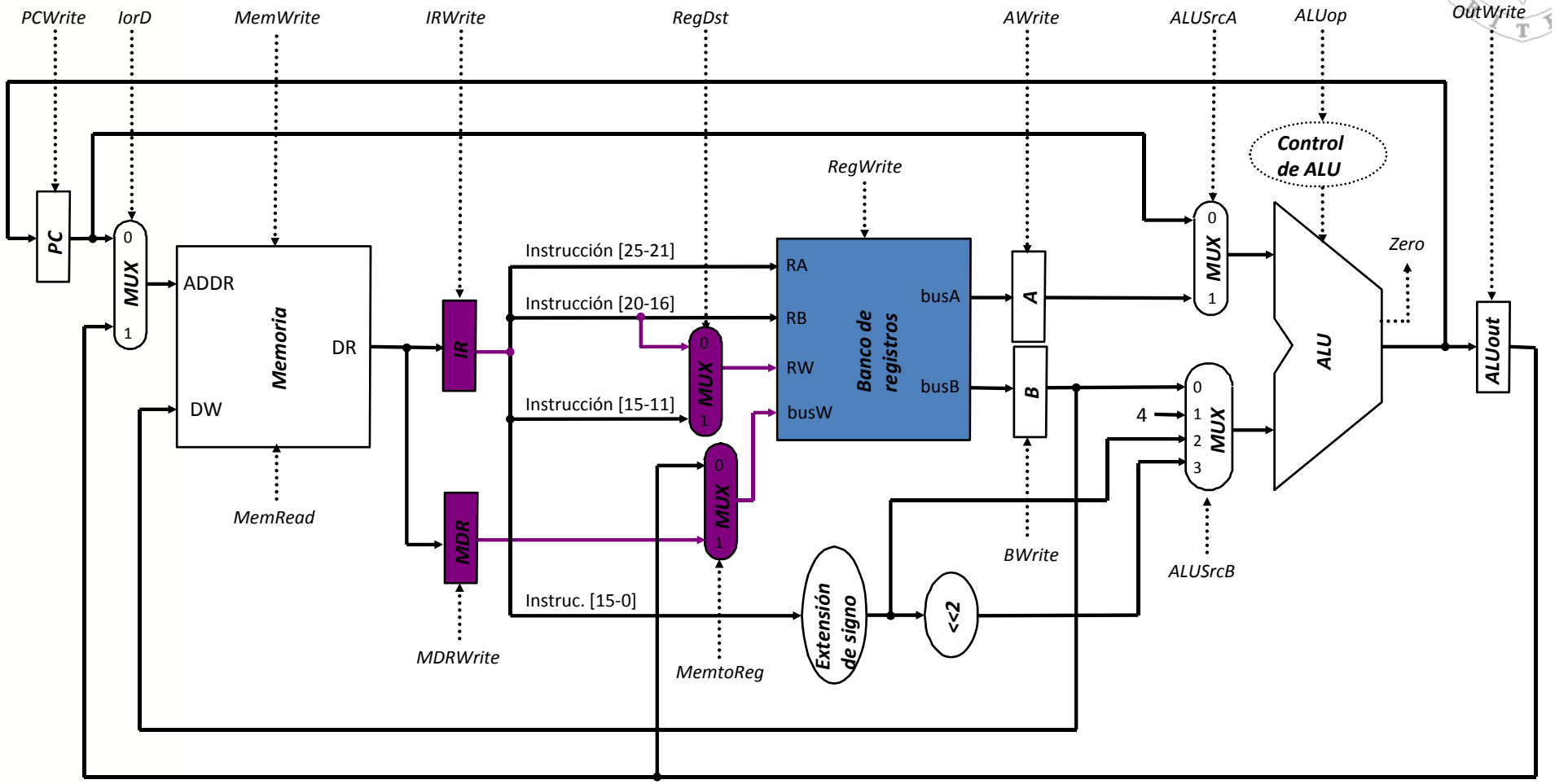
Diseño de la ruta de datos multiciclo



Instrucción lw

Etapa de acceso a memoria – memory access

Diseño de la ruta de datos multiciclo

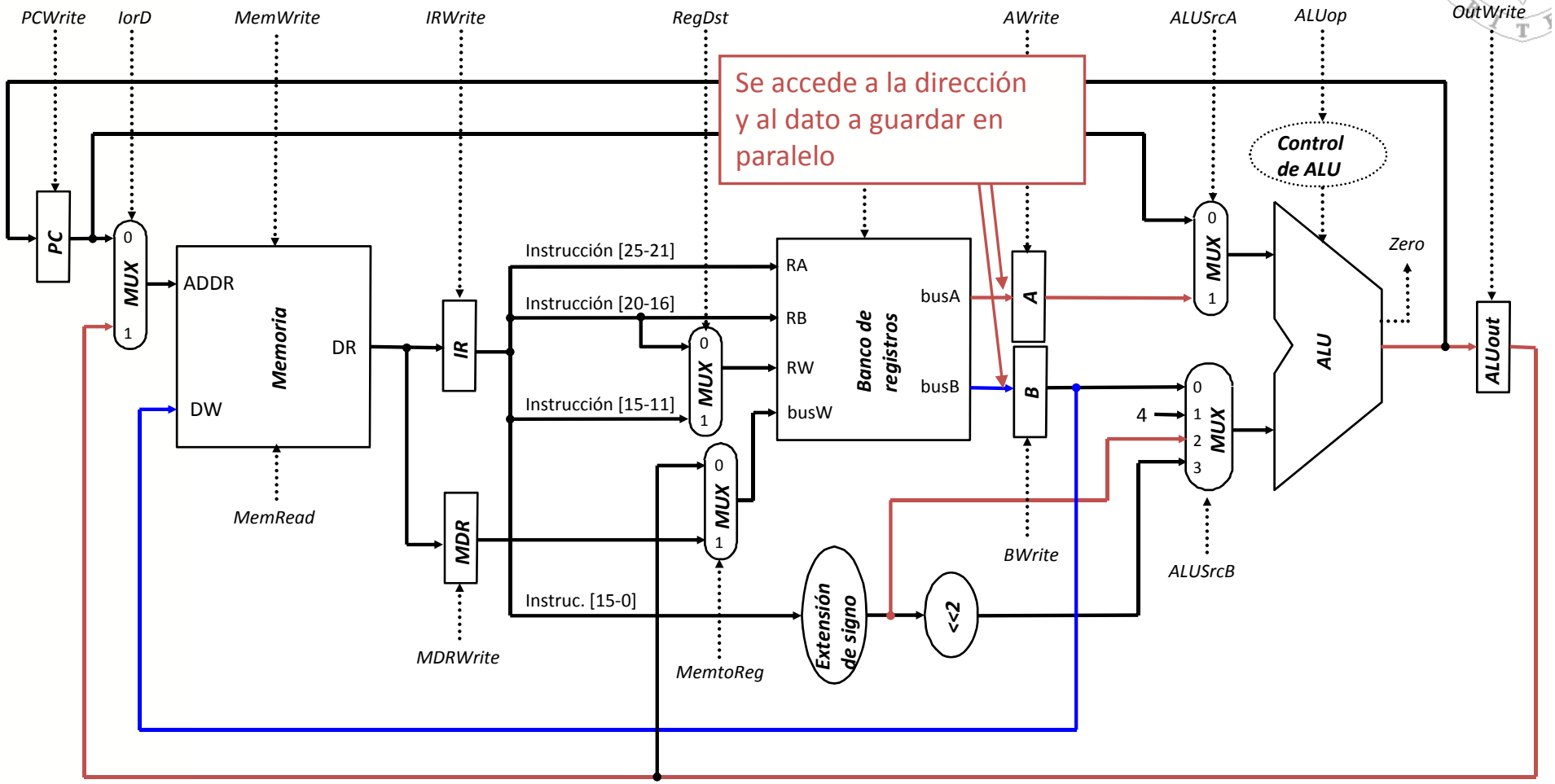


Instrucción lw

Etapa de finalización de la lectura de memoria

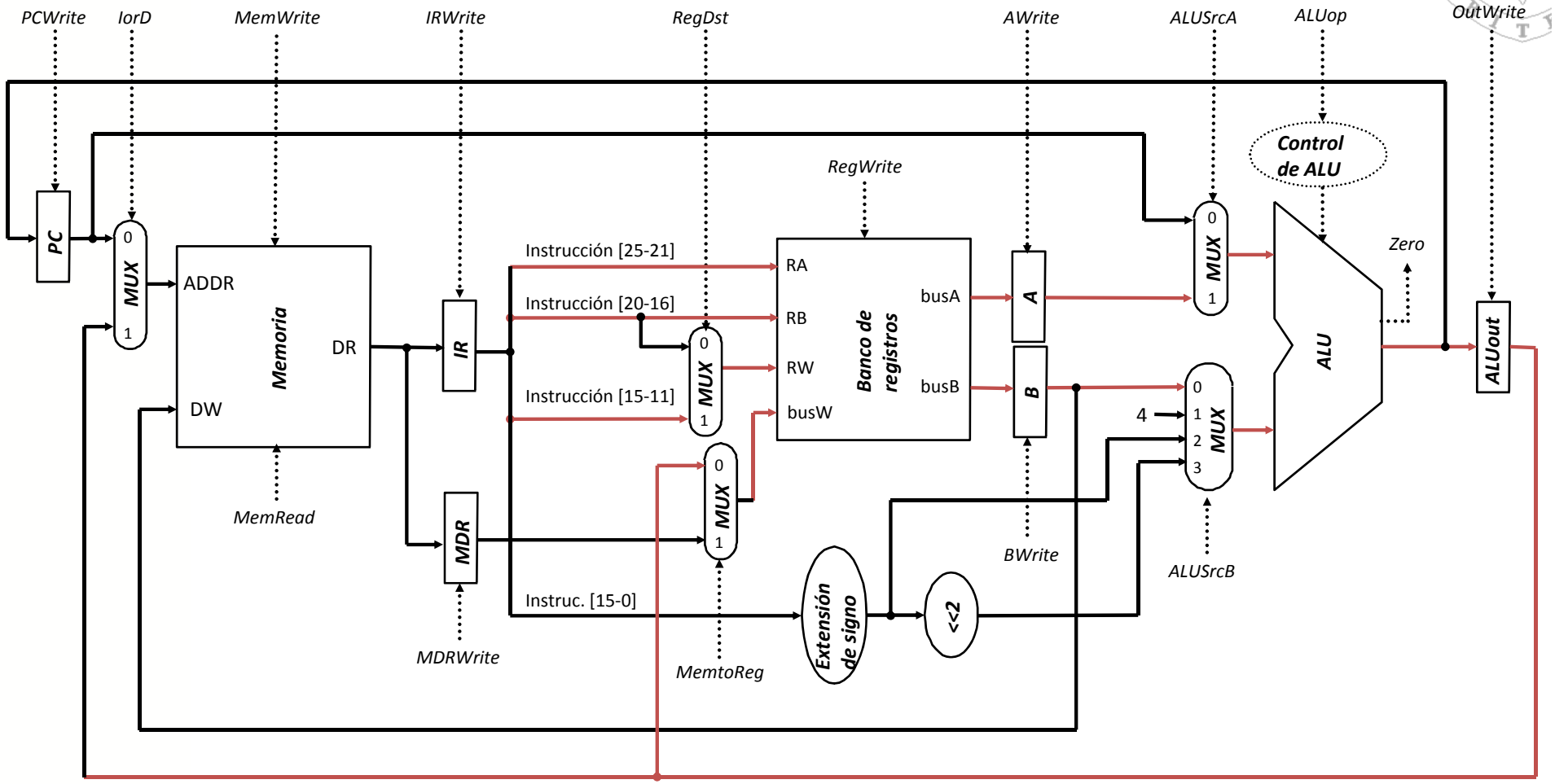
- write back

Diseño de la ruta de datos multiciclo



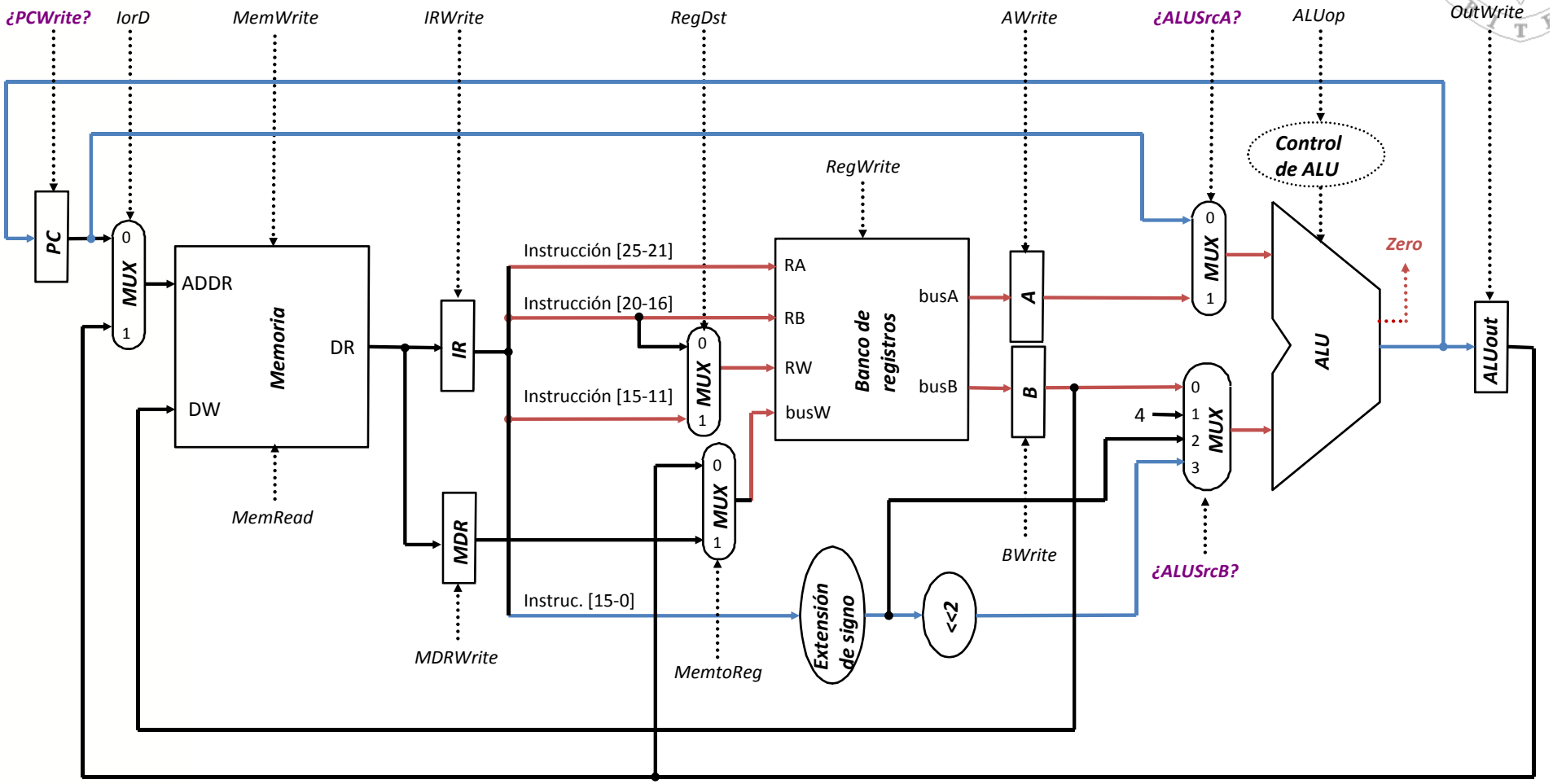
Instrucción sw

Diseño de la ruta de datos multiciclo



Instrucción aritmético-lógica

Diseño de la ruta de datos multiciclo

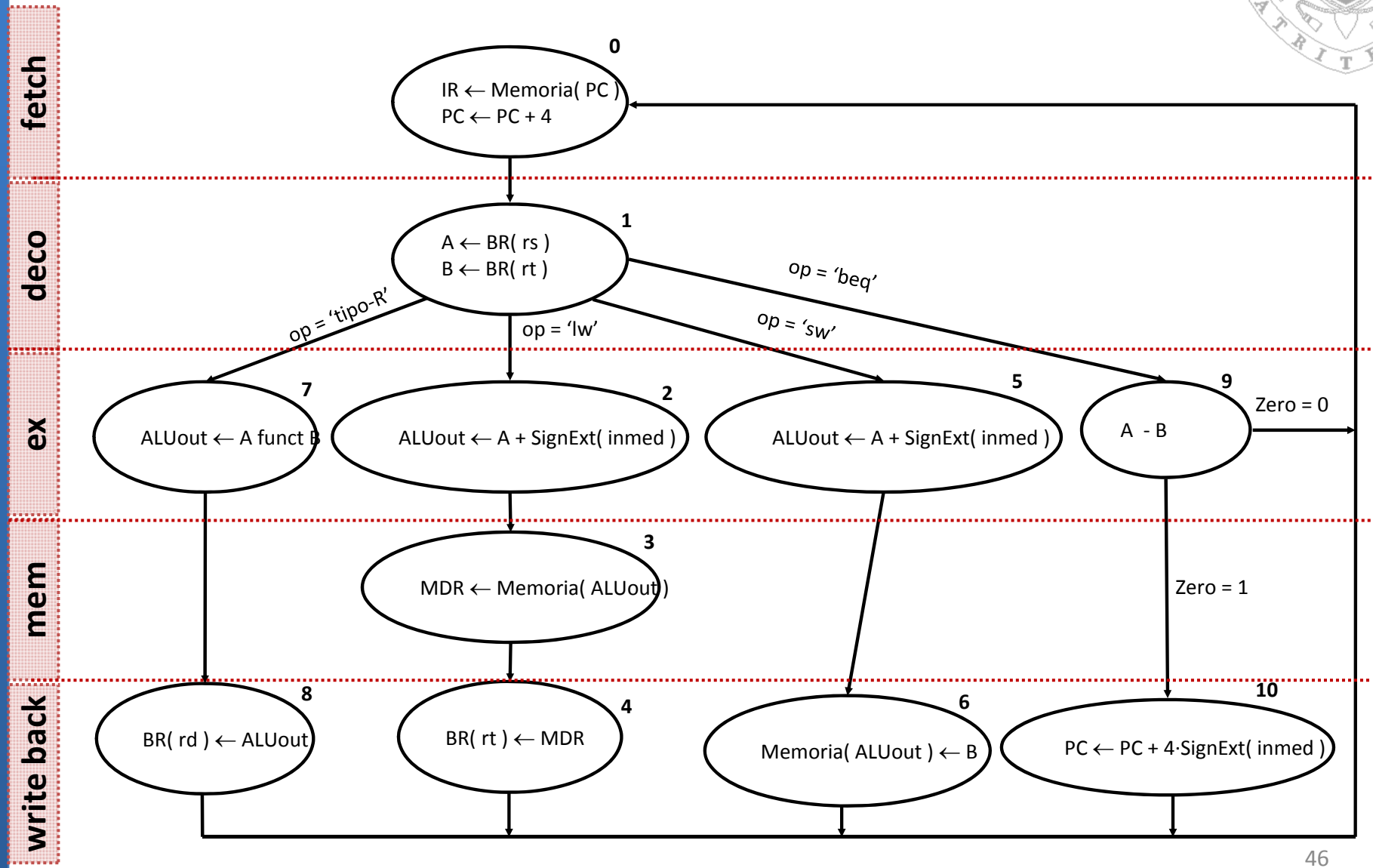


Instrucción de salto

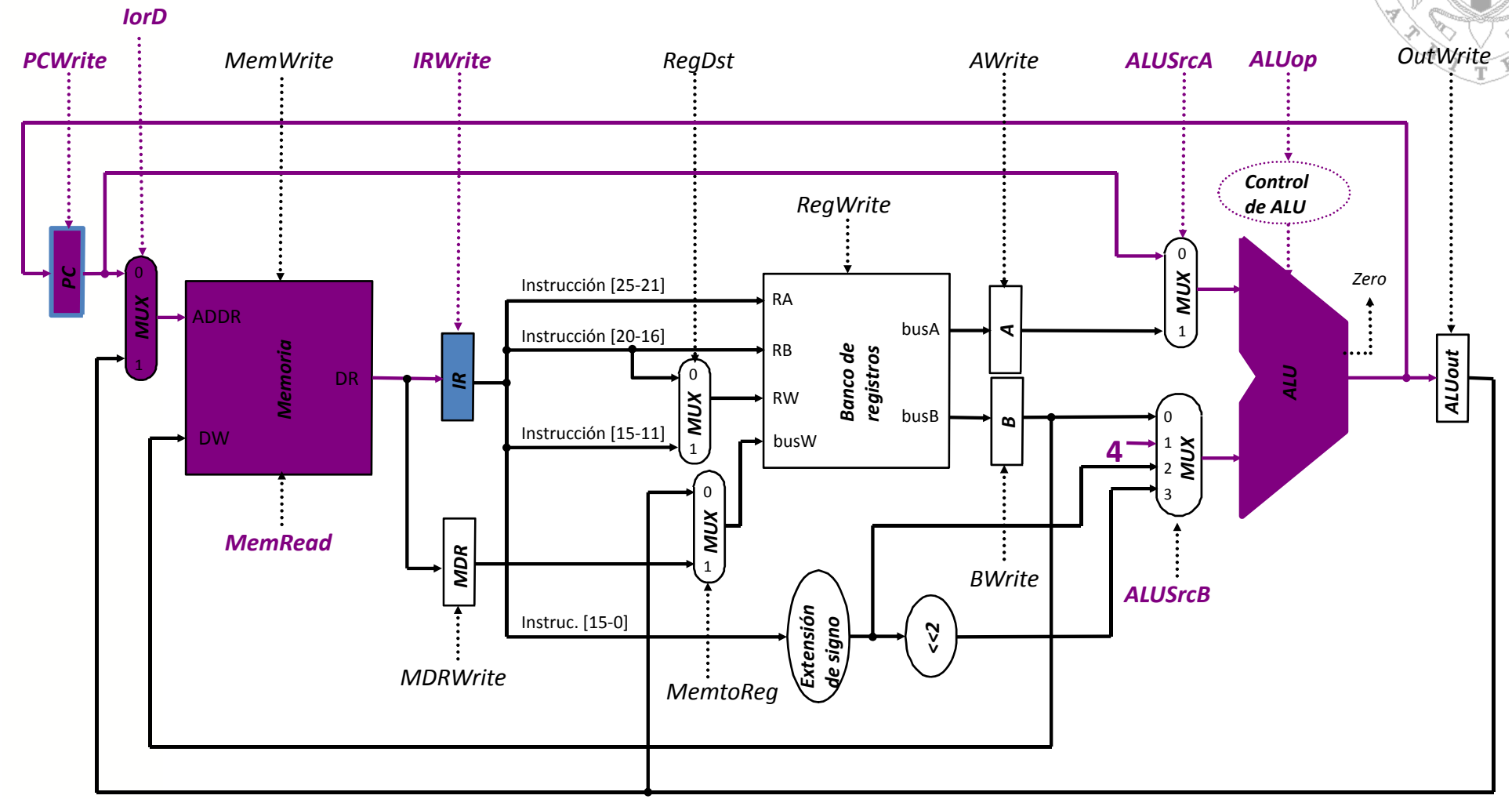
Diseño del controlador multiciclo

- La tarea del **controlador** es:
  - Seleccionar las operaciones a realizar por los módulos multifunción (ALU, read/write, ...)
  - Controlar el flujo de datos, controlando la entrada de selección de los multiplexores y la señal de carga de los registros
- Los valores de las **señales de control** dependen de la instrucción que se está ejecutando y la etapa en la que se encuentre

Diseño del controlador multiciclo

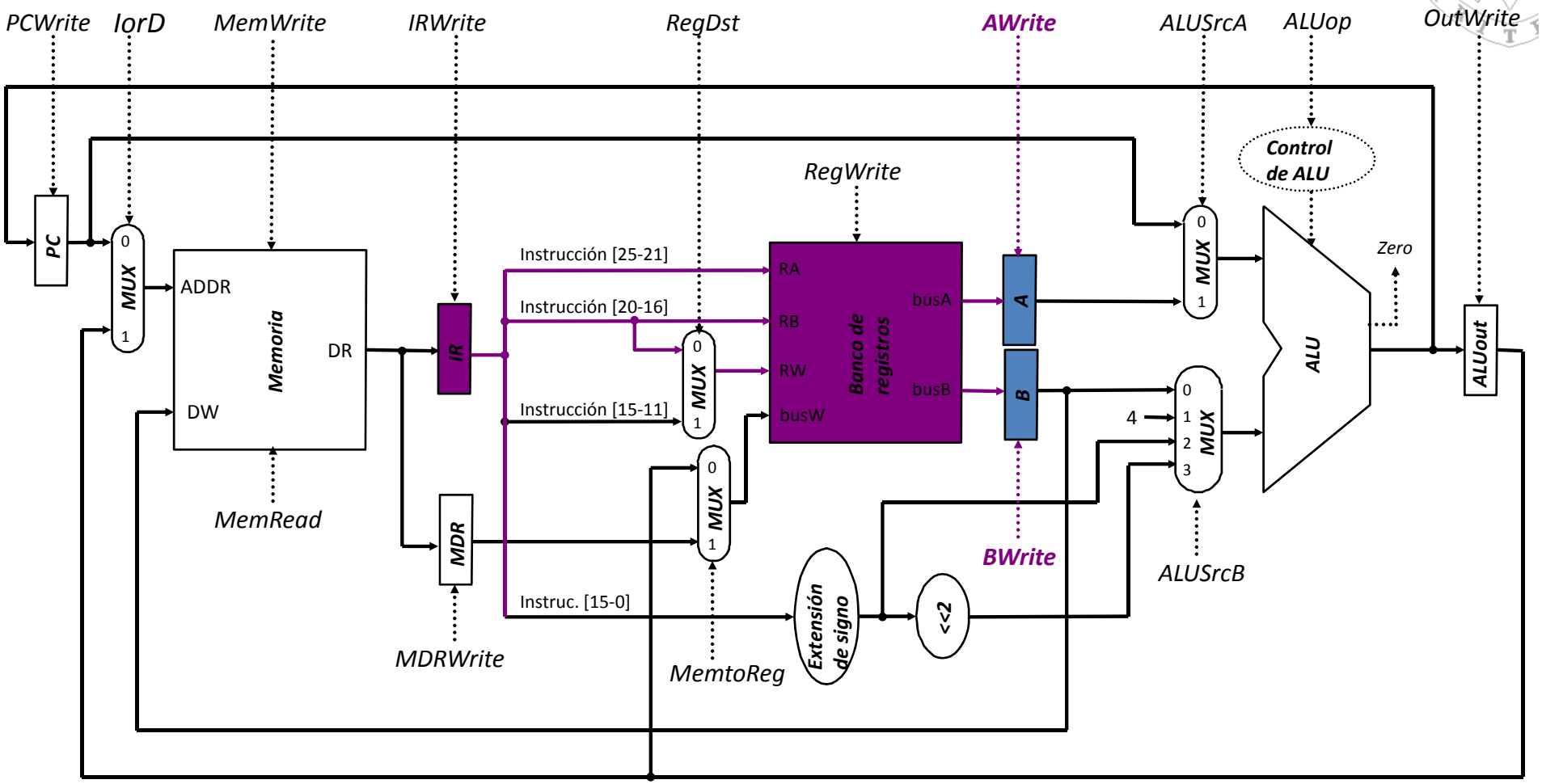


Diseño del controlador multiciclo



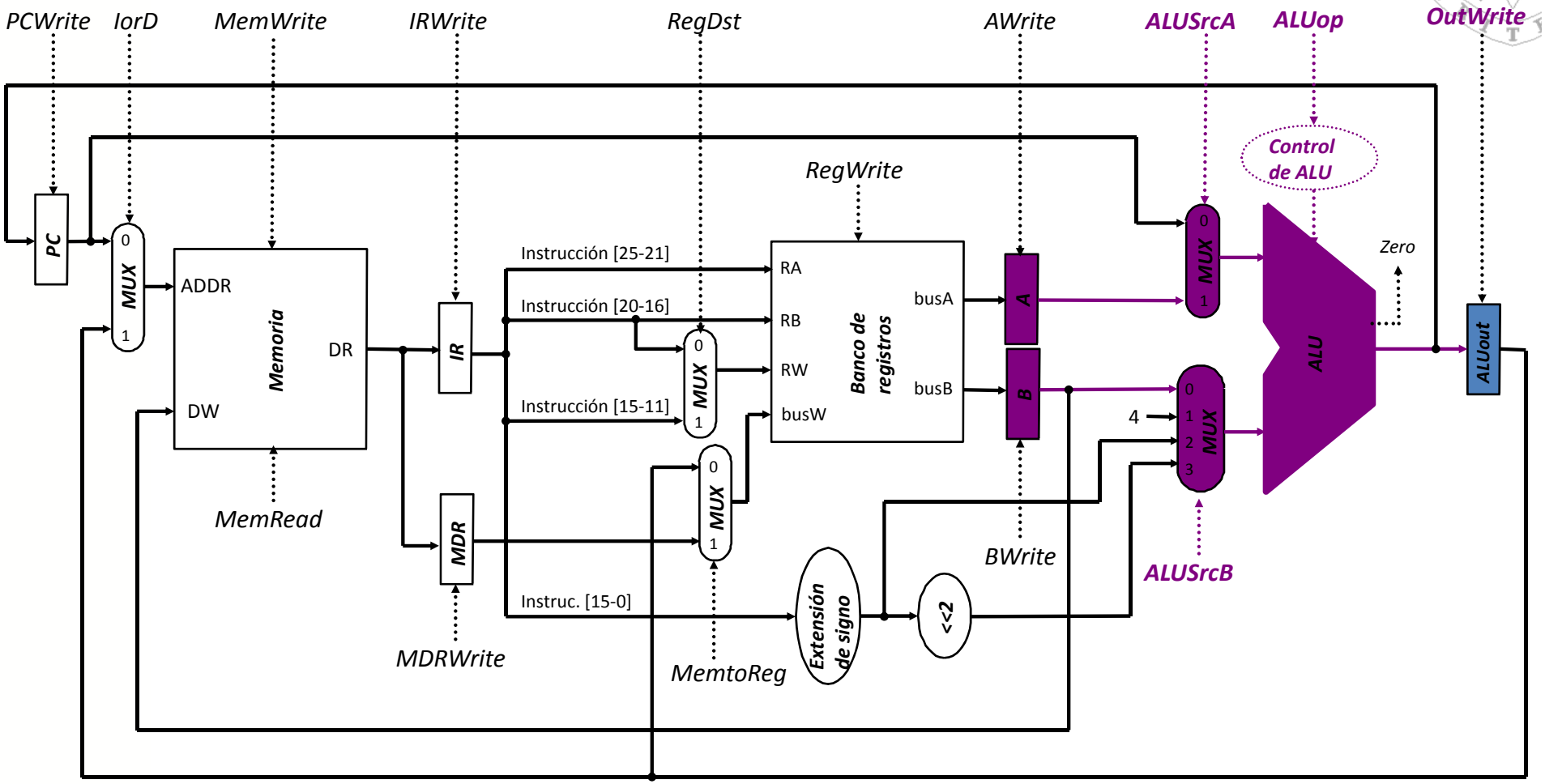
Ejemplo completo para i. aritmético-lógicas

Diseño del controlador multiciclo



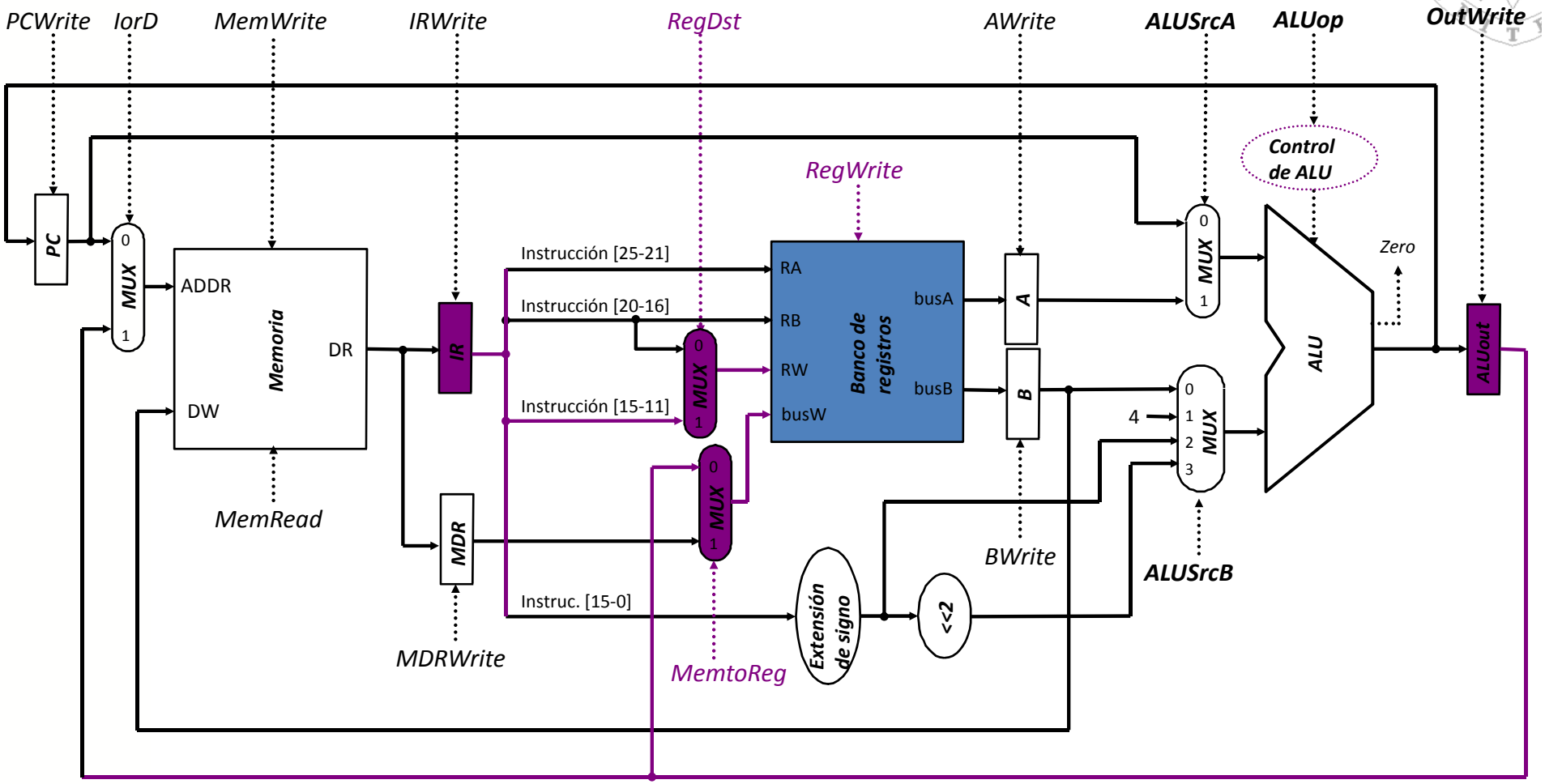
Ejemplo completo para i. aritmético-lógicas

Diseño del controlador multiciclo



Ejemplo completo para i. aritmético-lógicas

Diseño del controlador multiciclo



Ejemplo completo para i. aritmético-lógicas

Diseño del controlador multiciclo

- I. Aritmético-lógicas
  - Todas las señales de carga de registros deberían estar a **cero** cuando no se estén cargando dichos registros.
  - La señal de **MemWrite** debería estar a **cero** siempre
  - La señal **RegWrite** debería estar a **uno** en el estado **writeback** y a cero en el resto de estados
  - La señal **MemRead** debería estar a **uno** en el estado **fetch** y a cero en el resto de estados

Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

PCWrite = 1
IorD = 0
MemWrite = 0
MemRead = 1
IRWrite = 1

Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

AWrite = 1
BWrite = 1
RegWrite = 0

Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

ALUSrcA = 0
ALUSrcB = 0
ALUOp = *"function"*
OutWrite = 1

Diseño del controlador multiciclo

Transferencias entre registros "lógicas"

BR(rd) ← BR(rs) funct BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción Aritmético-lógica

MemtoReg = 0
RegDst = 1
RegWrite = 1

Diseño del controlador multiciclo

Instrucción de carga (lw)

Transferencias entre registros "lógicas"

BR(rt) ← Memoria(BR(rs) + SignExt(inmed)), PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A + SignExt(inmed)
4. MDR ← Memoria(ALUout)
5. BR(rt) ← MDR

Instrucción de almacenaje (sw)

Transferencias entre registros "lógicas"

Memoria(BR(rs) + SignExt(inmed)) ← BR(rt), PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A + SignExt(inmed)
4. Memoria(ALUout) ← B

Diseño del controlador multiciclo

Instrucción de salto condicional (beq)

Transferencias entre registros "lógicas"

si (BR(rs) = BR(rt))

entonces PC ← PC + 4 + 4·SignExt(inmed)

sino PC ← PC + 4

Transferencias entre registros "físicas"

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt),
3. A - B
4. si Zero entonces PC ← PC + 4·SignExt(inmed)

Observaciones: en todas las instrucciones las acciones 1. y 2. son iguales

Diseño del controlador multiciclo


```
lw r1, 0(r0)
lw r2, 4(r0)
add r3, r1, r2
beq r3, r5, 1
sub r3, r3, r5
sw r3, 8(r0)
```

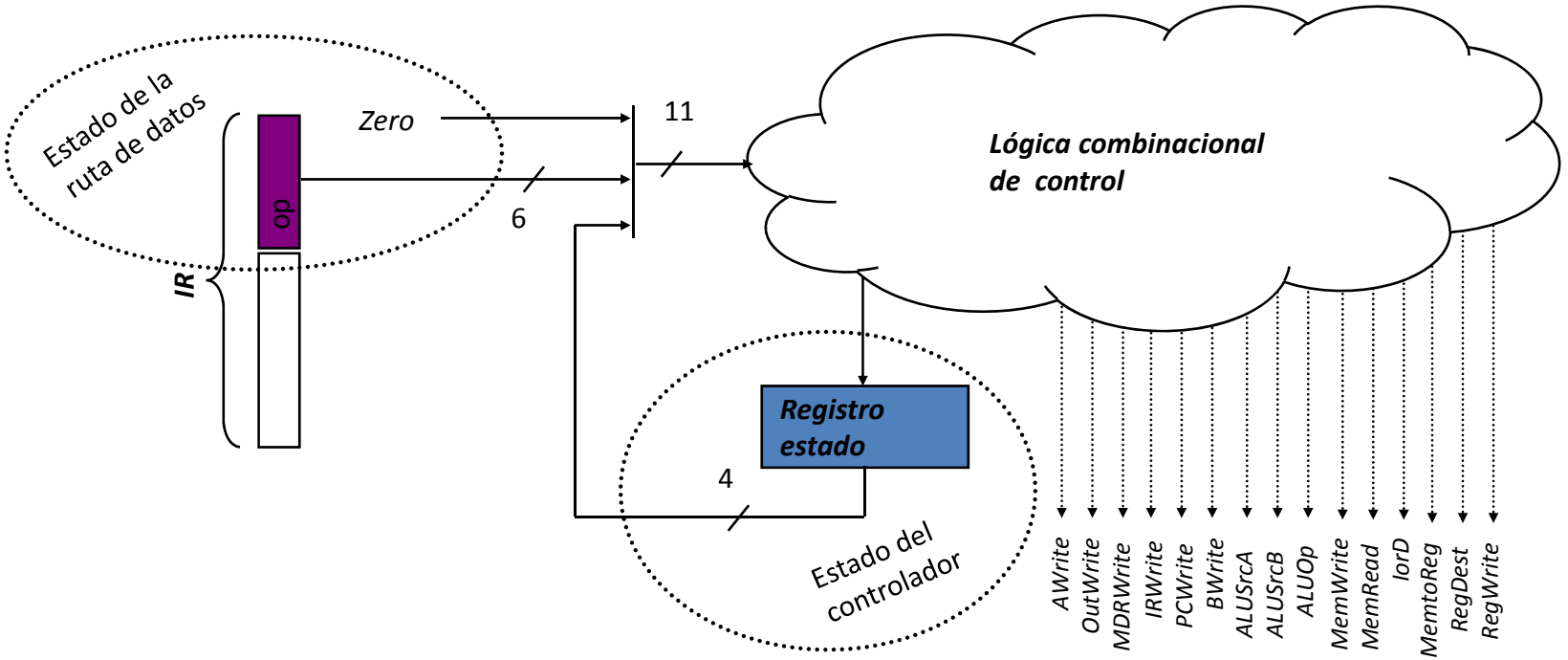
¿Tiempo de ejecución (ciclos)?

Diseño del controlador multiciclo

El controlador como FSM (finite state machine)

- 1. Se **traducen** las transferencias entre registros como **conjuntos de activaciones** de los puntos de control de la ruta de datos
- 2. Se **codifican** los estados
- 3. Mediante **tablas de verdad** se describen:
  - las **transiciones de estado** en función del código de operación y del estado de la ruta de datos
  - el **valor de las señales** de control en función del estado (controlador tipo Moore) y adicionalmente en función de algunas señales de la ruta de datos (controlador tipo Mealy).
- 4. La **estructura del controlador** estará formada por:
  - **Registro de estado**
  - Conjunto de **lógica combinacional de control** que implementa las anteriores tablas

Diseño del controlador multiciclo



Diseño del controlador multiciclo

Tabla de verdad del controlador

Estado actual	op	Zero	Estado siguiente	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
0000	XXXXXX	X	0001	1	1			0	01	00 (add)		0	1	0				0
0001	100011 (lw)	X	0010															
0001	101011 (sw)	X	0101	0	0	1	1					0	0					0
0001	000000 (tipo-R)	X	0111															
0001	000100 (beq)	X	1001															
0010	XXXXXX	X	0011	0	0			1	10	00 (add)	1	0	0					0
0011	XXXXXX	X	0100	0	0							0	1	1	1			0
0100	XXXXXX	X	0000	0	0							0	0			1	0	1
0101	XXXXXX	X	0110	0	0		0	1	10	00 (add)	1	0	0					0
0110	XXXXXX	X	0000	0	0							1	0	1				0
0111	XXXXXX	X	1000	0	0			1	00	10 (funct)	1	0	0					0
1000	XXXXXX	X	0000	0	0							0	0			0	1	1
1001	XXXXXX	0	0000	0	0			1	00	01 (sub)		0	0					0
1001	XXXXXX	1	1010															
1010	XXXXXX	X	0000	0	1			0	11	00 (add)		0	0					0

Diseño del controlador multiciclo

Estado actual	op	Zero	Estado siguiente	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
s_3 s_2 s_1 s_0	op_5 op_4 op_3 op_2 op_1 op_0		**ns_3** ns_2 ns_1 ns_0															
0 0 0 0	X X X X X X	X	**0** 0 0 1	1	1			0	01	00 (add)		0	1	0				0
0 0 0 1	1 0 0 0 1 1	X	**0** 0 1 0															
0 0 0 1	1 0 1 0 1 1	X	**0** 1 0 1	0	0	1	1					0	0					0
0 0 0 1	0 0 0 0 0 0	X	**0** 1 1 1															
0 0 0 1	0 0 0 1 0 0	X	**1** 0 0 1															
0 0 1 0	X X X X X X	X	**0** 0 1 1	0	0			1	10	00 (add)	1	0	0					0
0 0 1 1	X X X X X X	X	**0** 1 0 0	0	0							0	1	1	1			0
0 1 0 0	X X X X X X	X	**0** 0 0 0	0	0							0	0			1	0	1
0 1 0 1	X X X X X X	X	**0** 1 1 0	0	0		0	1	10	00 (add)	1	0	0					0
0 1 1 0	X X X X X X	X	**0** 0 0 0	0	0							1	0	1				0
0 1 1 1	X X X X X X	X	**1** 0 0 0	0	0			1	00	10 (funct)	1	0	0					0
1 0 0 0	X X X X X X	X	**0** 0 0 0	0	0							0	0			0	1	1
1 0 0 1	X X X X X X	0	**0** 0 0 0															
1 0 0 1	X X X X X X	1	**1** 0 1 0	0	0			1	00	01 (sub)		0	0					0
1 0 1 0	X X X X X X	X	**0** 0 0 0	0	1			0	11	00 (add)		0	0					0

Diseño del controlador multiciclo

- Lógica discreta:
  - 21 funciones de conmutación
  - 11 variables diferentes
- 1 PLA
  - 11 entradas
  - 21 salidas
  - 15 términos producto
- 1 ROM (~42 Kbits):
  - 11 bits de dirección (2^{11} palabras)
  - palabras de 21 bits
- 2 ROM (~10 Kbits)
  - ROM de control:
    - 4 bits de dirección (2^4 palabras)
    - palabra de 17 bits
  - ROM de siguiente estado:
    - 11 bits de dirección (2^{11} palabras)
    - palabras de 4 bits

- Ventajas de la lógica discreta:
  - velocidad y coste
- Ventajas de la lógica almacenada:
  - facilidad de diseño
  - adaptabilidad