Estructura de computadores

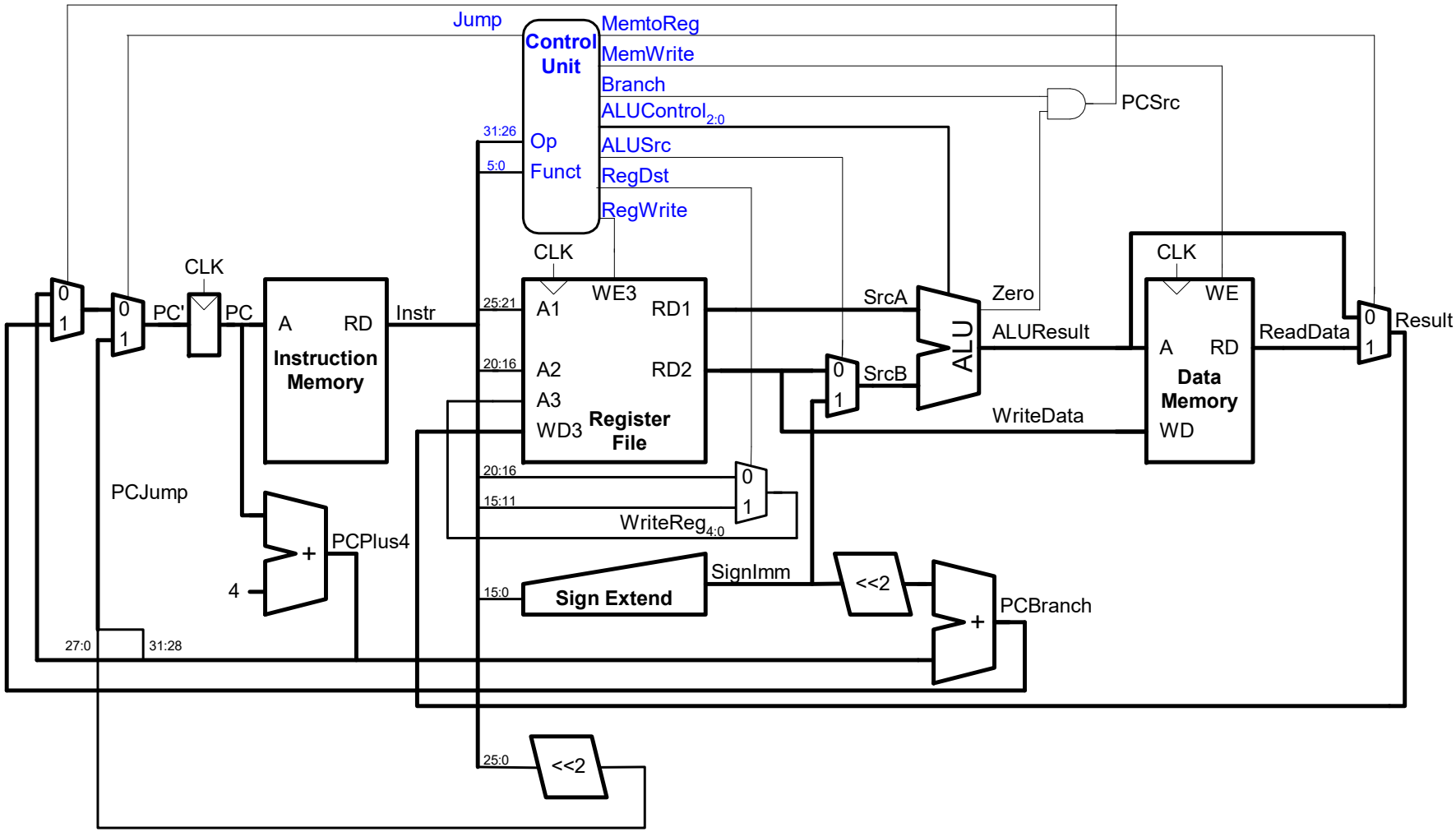
Unidad 5. El Procesador III: Diseño y control de la ruta de datos: Arquitectura multiciclo

Escuela Politécnica Superior - UAM

Índice

- **Resumen arquitectura MIPS uniciclo**
- Ruta de datos multiciclo
- Control multiciclo
- Añadir más instrucciones

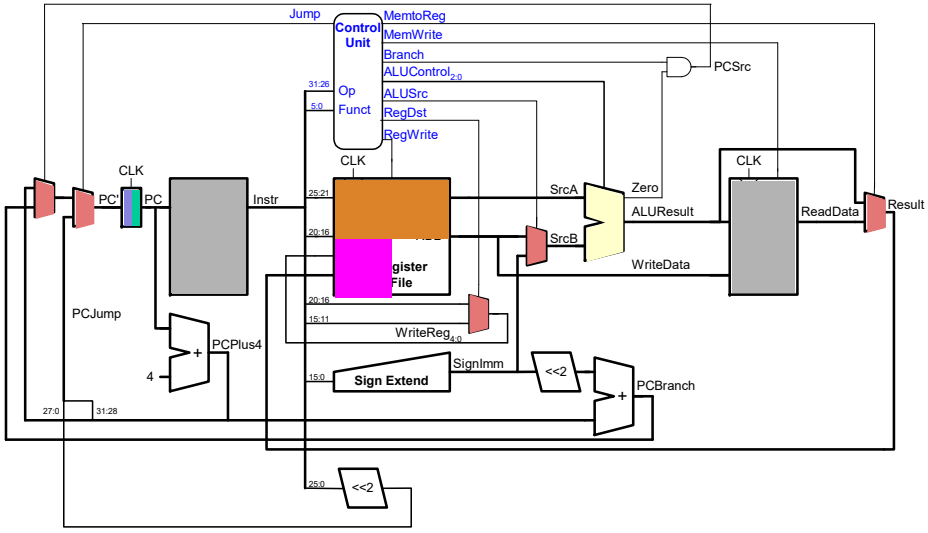
Resumen: MIPS uniciclo



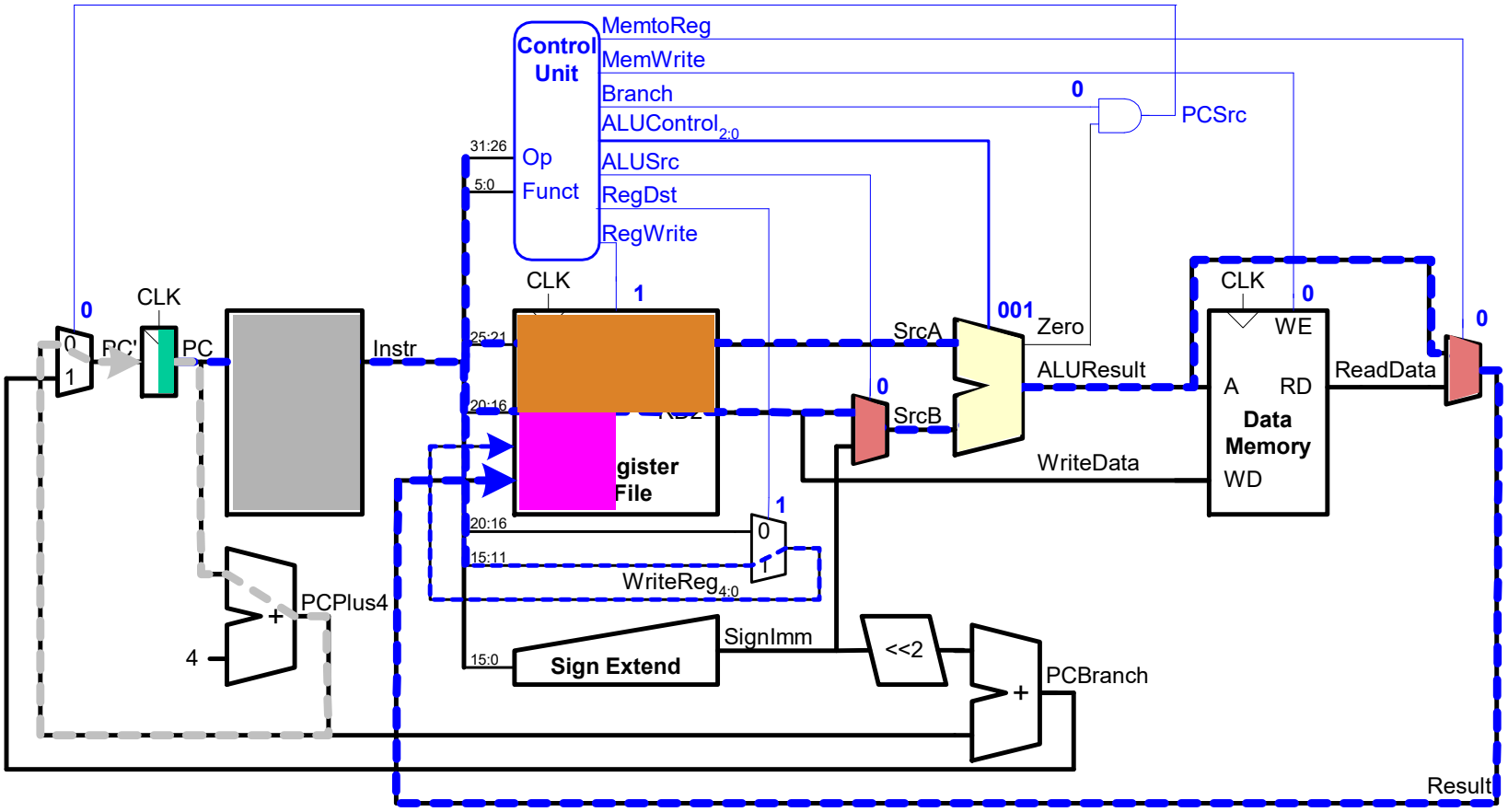
Parámetros temporales: Ciclo de reloj en uniciclo

- Todas las instrucciones utilizan el mismo ciclo de reloj, que tendrá que ser suficiente para la más lenta de todas.

Elemento	Parámetro	Retardo (ps)
$T_{CLK \rightarrow Q}$ Registro	t_{pcq_PC}	30
Leer de Memoria	t_{mem}	250
Leer del BancoReg	t_{RFread}	150
Multiplexor	t_{mux}	25
ALU	t_{ALU}	200
T_{SETUP} BancoReg	$t_{RFsetup}$	20
T_{SETUP} Registro	t_{setup}	20



Parámetros temporales: Ciclo de reloj en uniciclo. Ejemplo: camino crítico para ***add***



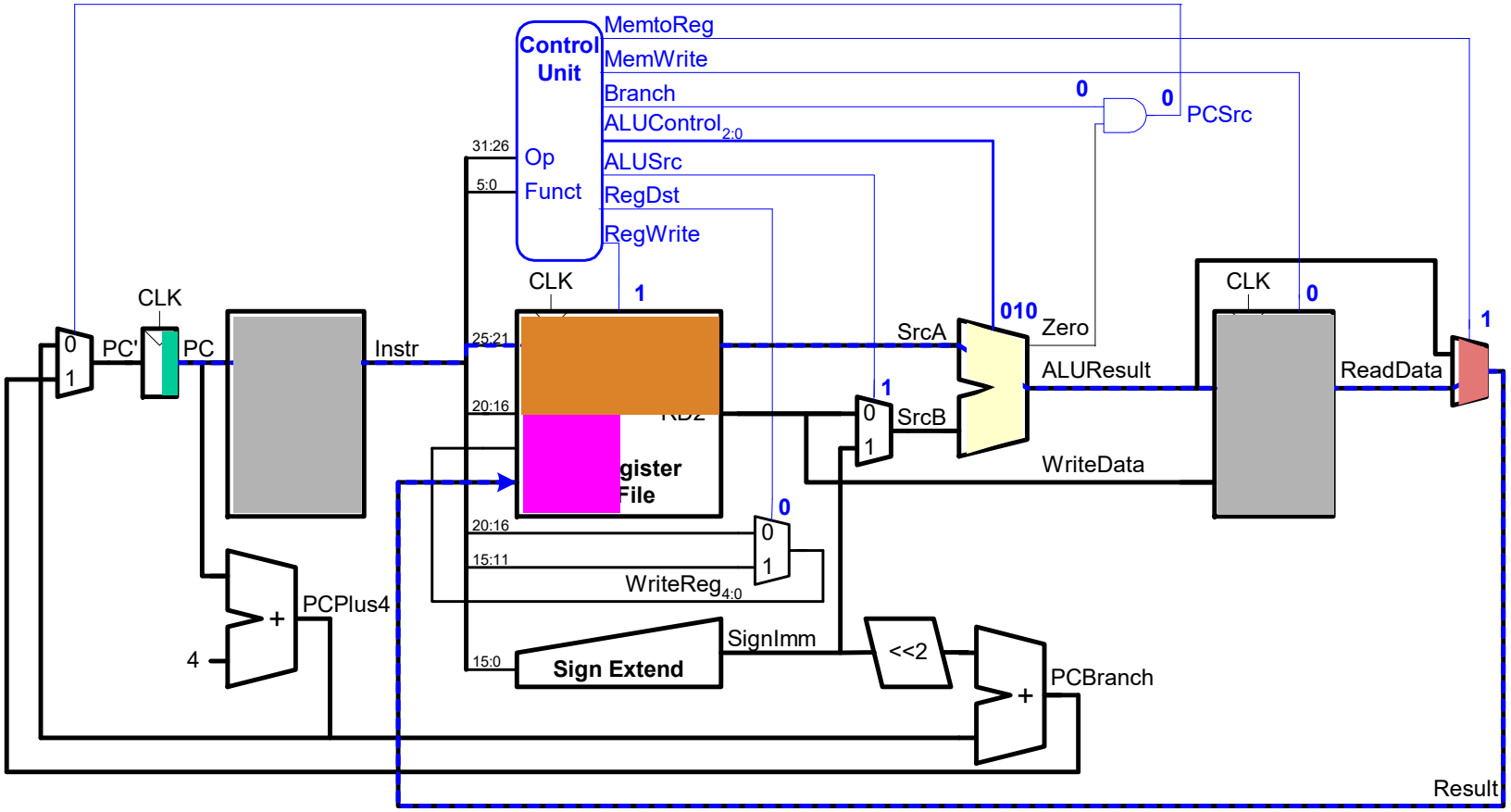
Retraso de la instrucción add:

$T_c = t_{pcq\_PC} + t_{mem} + t_{RFread} + t_{mux} + t_{ALU} + t_{mux} + t_{RFsetup} =$

$= [30 + 250 + 150 + 25 + 200 + 25 + 20]$ ps = 700 ps (1,4 GHz)

"La escritura en el banco de registros se realiza en el flanco siguiente"

Parámetros temporales: Ciclo de reloj en uniciclo. Ejemplo: camino crítico para *lw*



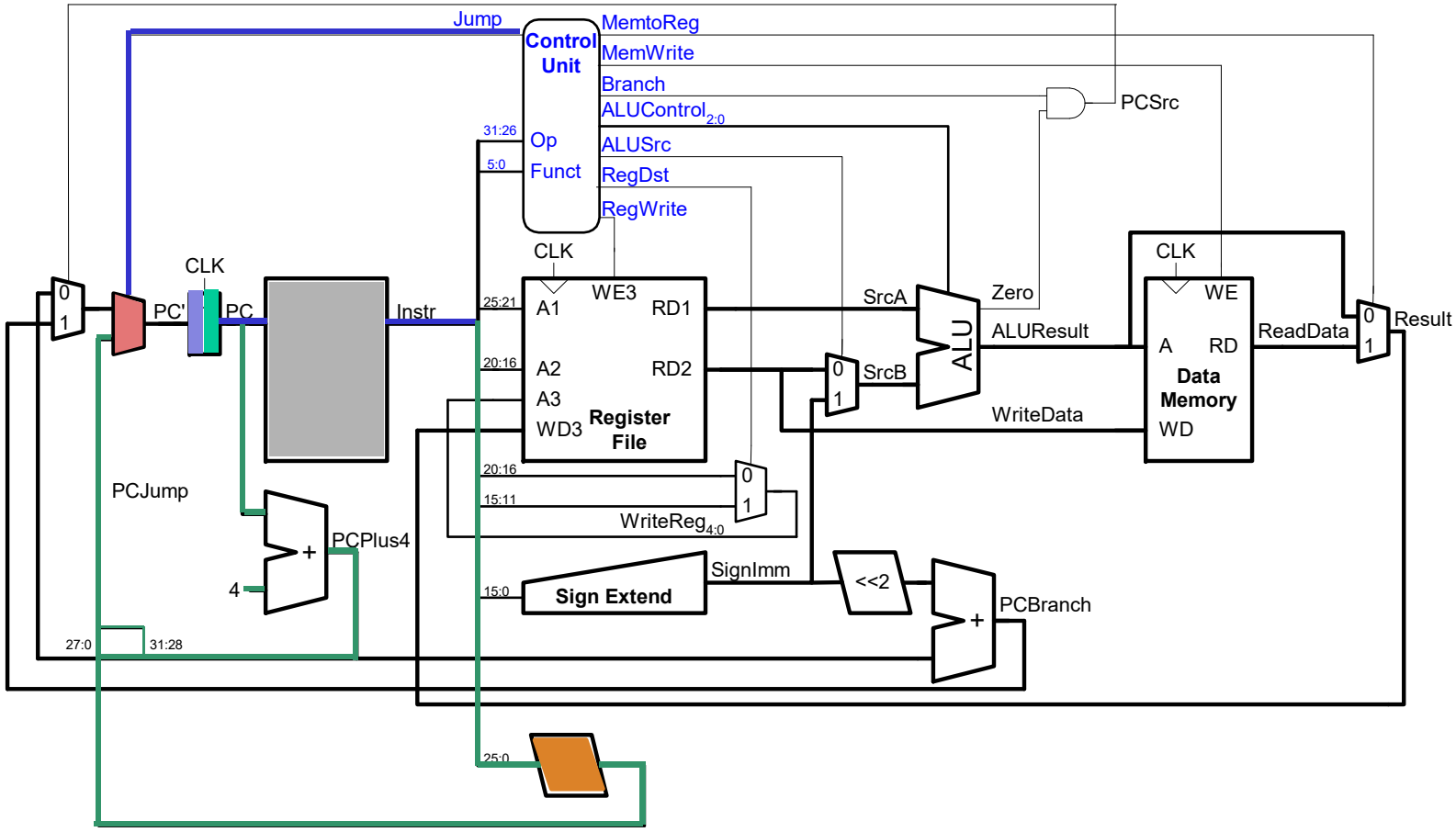
Retraso de la instrucción lw:

$T_c = t_{pcq\_PC} + t_{mem} + t_{RFread} + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup} =$

$= [30 + 250 + 150 + 200 + 250 + 25 + 20]$ ps = 925 ps (1,1 GHz)

"La escritura en el banco de registros se realiza en el flanco siguiente"

Parámetros temporales: Ciclo de reloj en uniciclo. Ejemplo: camino crítico para ***jump(j)***



Camino crítico para la instrucción *jump* (j):

$T_c = t_{pcq\_PC} + t_{mem} + t_{<<} + t_{mux} + t_{PCsetup} =$

$= [30 + 250 + 150 + 25 + 20]$ ps = 475 ps (~2,1 GHz)

MIPS uniciclo vs multiciclo

	t_{pcq_PC}	t_{mem}	$T_{<<}$	t_{RFread}	t_{mux}	t_{ALU}	t_{mem}	t_{mux}	$t_{RFsetup}/t_{PCsetup}$	T_{total} (ps)
add	30	250	--	150	25	200	--	25	20/20	700 ps
lw	30	250	--	150	--	200	250	25	20/20	**925 ps**
j	30	250	150	--	--	--	--	25	20	475 ps

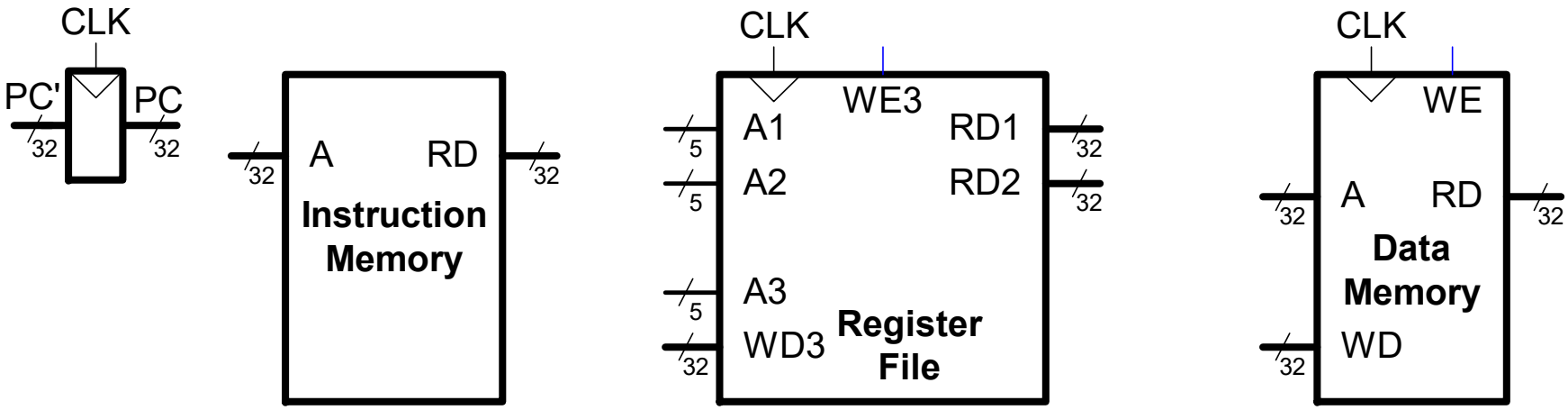
- Microarquitectura uniciclo:
  - + Simple
  - - Ciclo de reloj limitado por instrucción más lenta (**lw**)
  - - 3 sumadores/ALUs y 2 memorias (código y datos)
- Microarquitectura multiciclo:
  - + Reloj más rápido
  - + Instrucciones sencillas van más rápido (menos ciclos)
  - + Reutilizar hardware de aquellos elementos que antes se utilizaban en el mismo ciclo y ahora se utilizarán en distintos ciclos de reloj)
  - - Si no se divide en etapas del mismo retraso se pierden prestaciones

Índice

- Resumen arquitectura MIPS uniciclo
- **Ruta de datos multiciclo**
- Control multiciclo
- Añadir más instrucciones

Repaso uniciclo.
Estado de la arquitectura

- Se puede conocer en qué situación se encuentra el micro conociendo los valores de:
  - PC
  - Banco de registros (los 32 registros)
  - Memoria (de código y de datos)
- Primeros elementos a considerar en la ruta de datos:



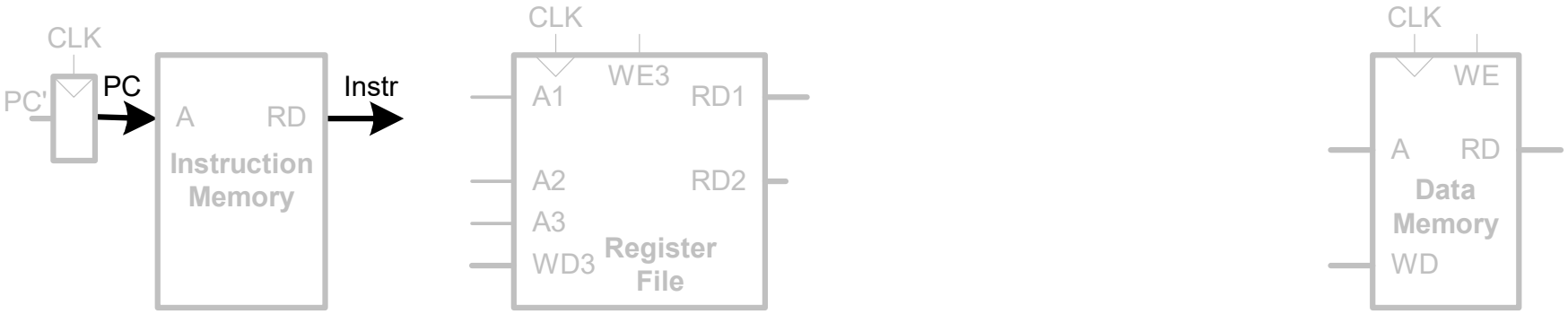
Repaso uniciclo.
Ruta de datos: captura lw

El análisis de la ruta de datos comienza con la instrucción:

(**0x8C112000**) **`lw $s1,0x2000($0)`**

y los pasos para ejecutarla:

- **PASO 1:** captura de instrucción (*fetch*)

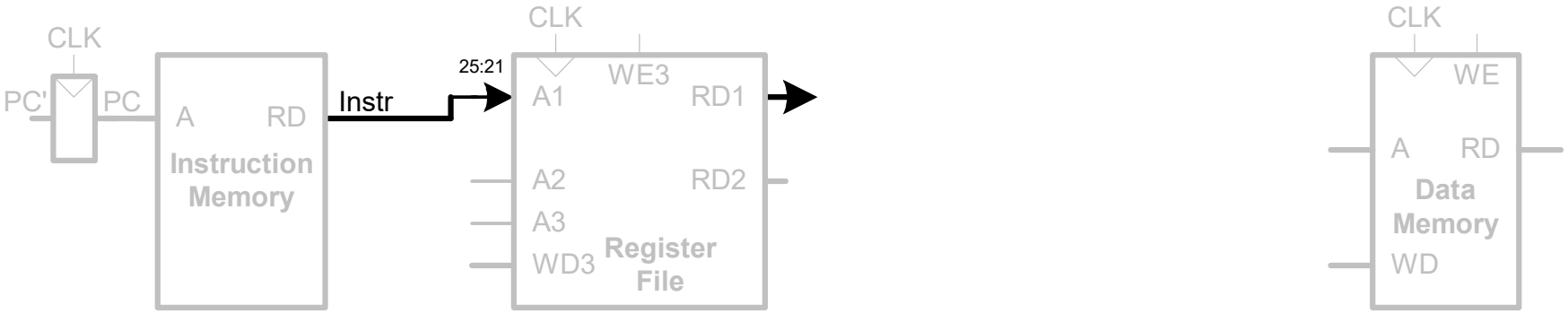


Instr <= "100011 00000 10001 0010000000000000"

op: Instr [31:26] =	"100011"	=>	**`lw`**
rs: Instr [25:21] =	"00000"	=>	**`$0`**
rt: Instr [20:16] =	"10001"	=>	**`$s1`**
imm: Instr [15:0] =	"0010000000000000"	=>	**`0x2000`**

Repaso uniciclo.
Ruta de datos: lectura de registros lw

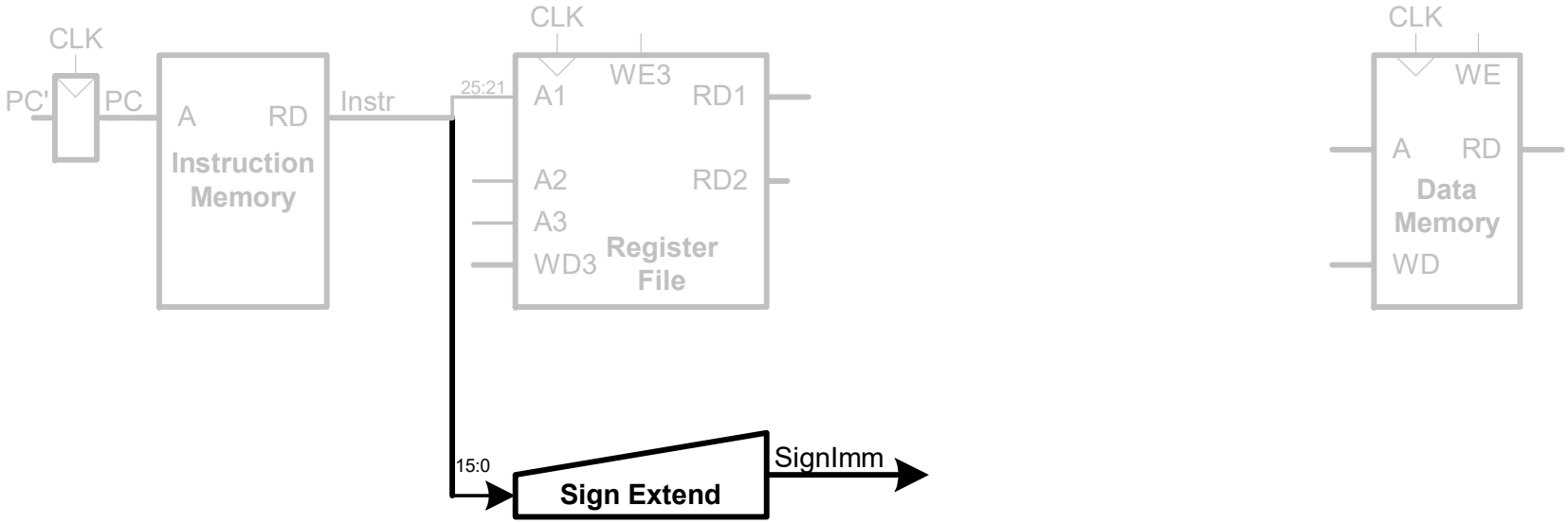
- **PASO 2:** lectura de los operandos fuente del banco de registros



A1: Instr [25:21] = “00000”; RD1 <= “0x00000000” = ($0)

Repaso uniciclo.
Ruta de datos: dato inmediato lw

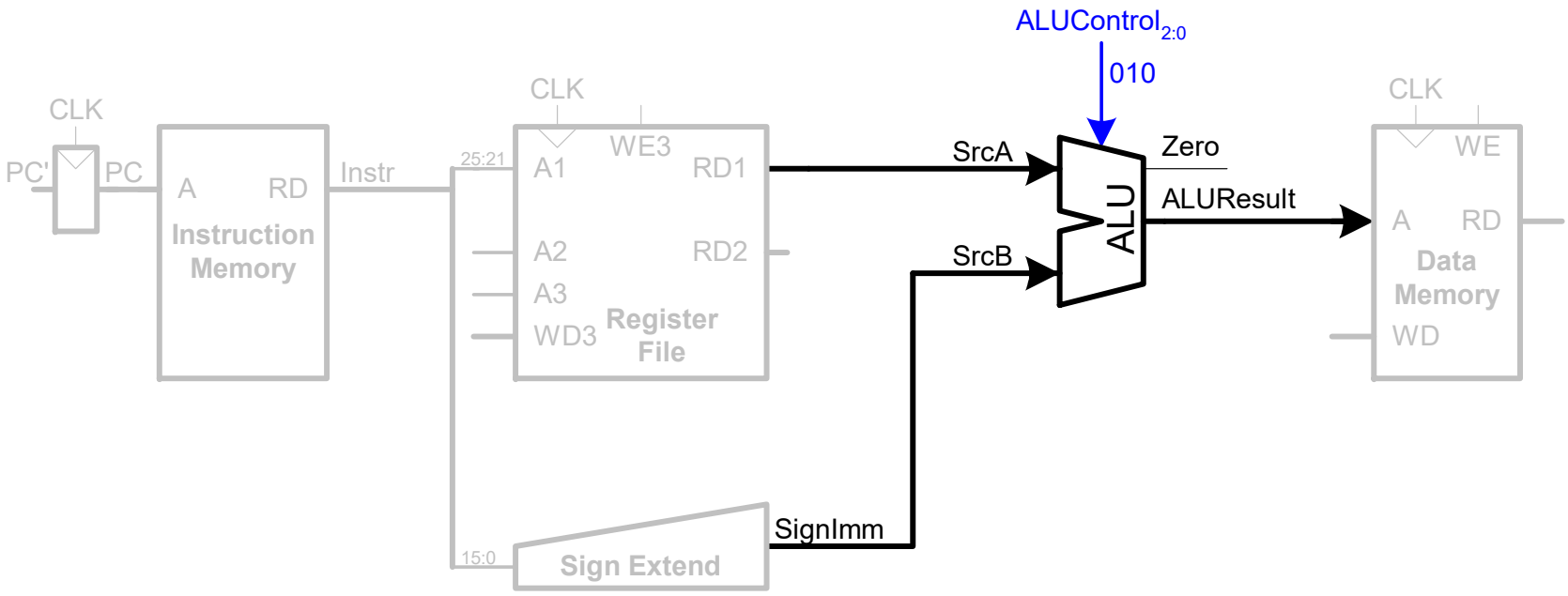
- **PASO 3:** extensión en signo del dato inmediato



SignImm <= 0x00002000 = Sign Extend (0x2000)

Repaso uniciclo.
Ruta de datos: dirección lw

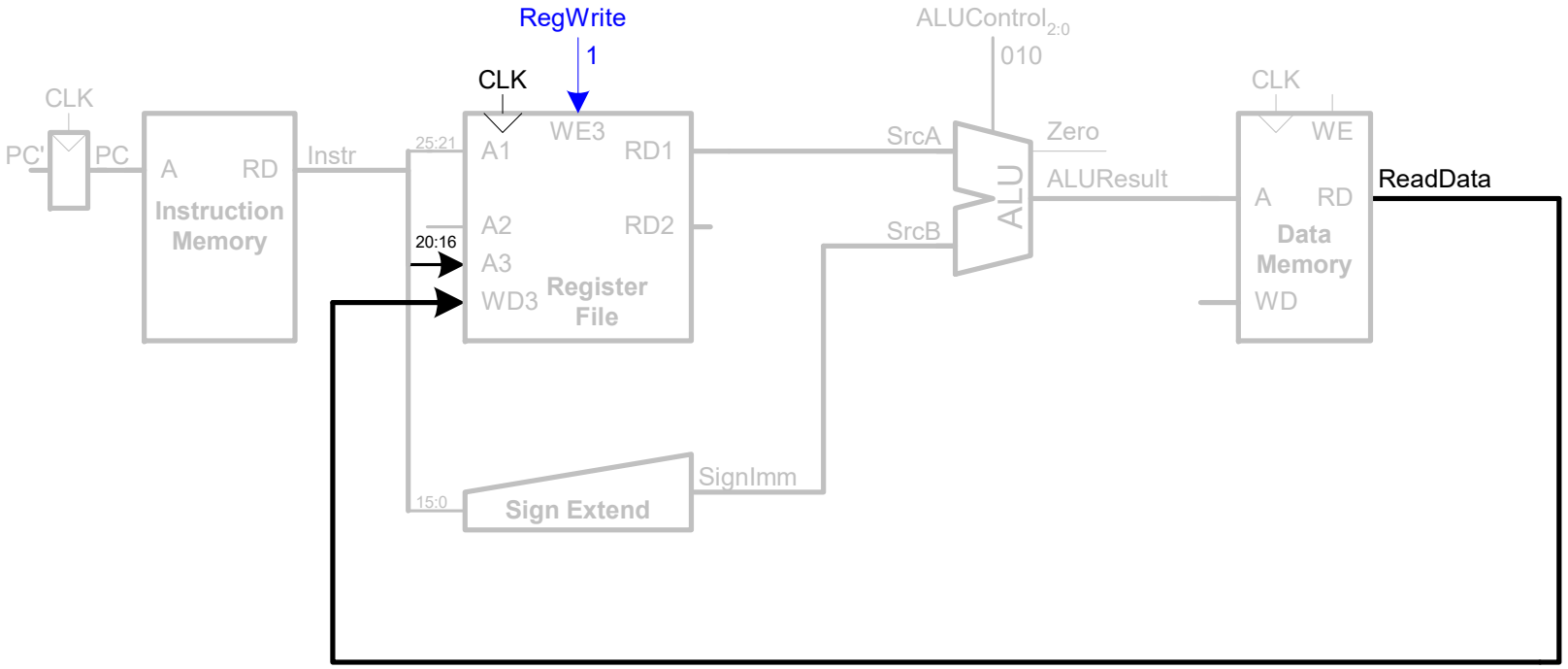
- **PASO 4:** calcular la dirección para acceso a memoria de datos sumando ([rs] + SignImm) en la ALU



ALUResult <= 0x00002000 = 0x00000000 + 0x00002000

Repaso uniciclo.
Ruta de datos: leer memoria lw

- **PASO 5:** leer el dato buscado de memoria y escribirlo en el registro destino, rt



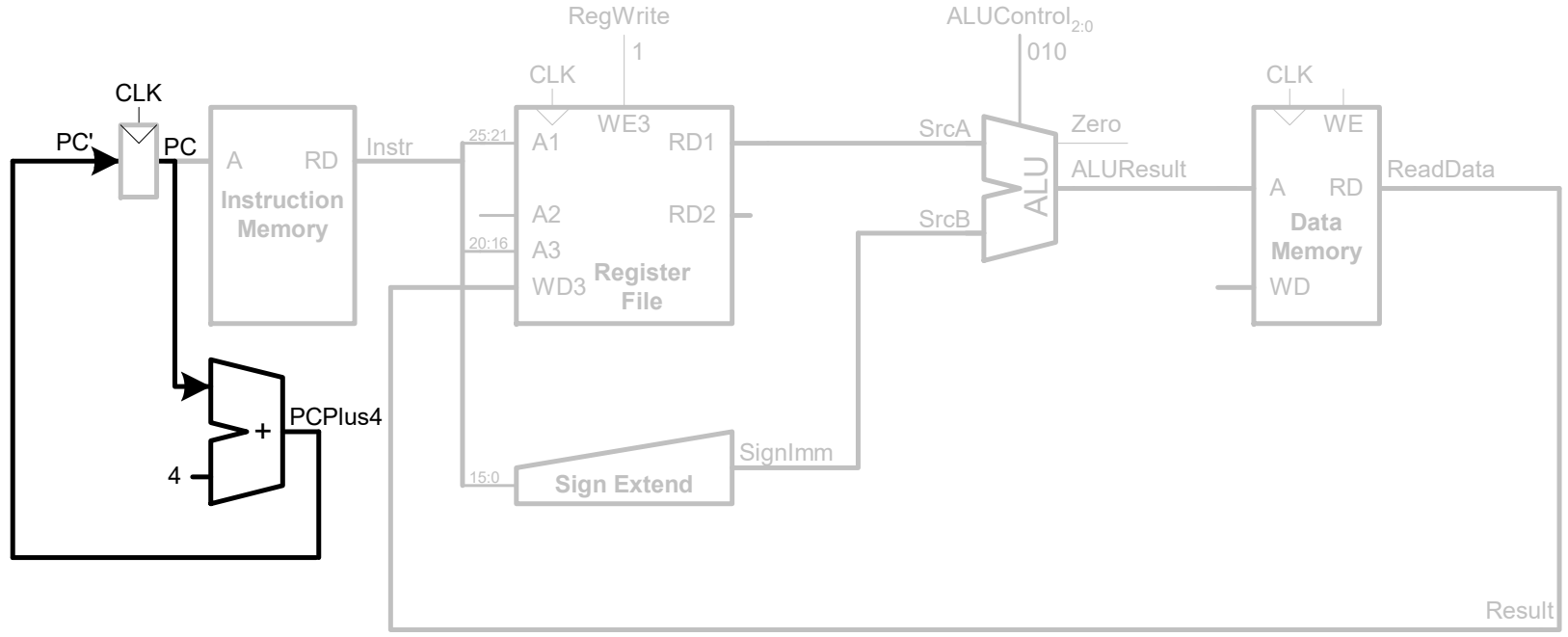
A3: Instr [20:16] = “10001” ($s1)

$s1 <= WD3 = MEM[0x00002000]

Repaso uniciclo.
Ruta de datos: incrementar PC

P

- **PASO 6:** incrementar el PC en 4 para tener la dirección de la próxima instrucción

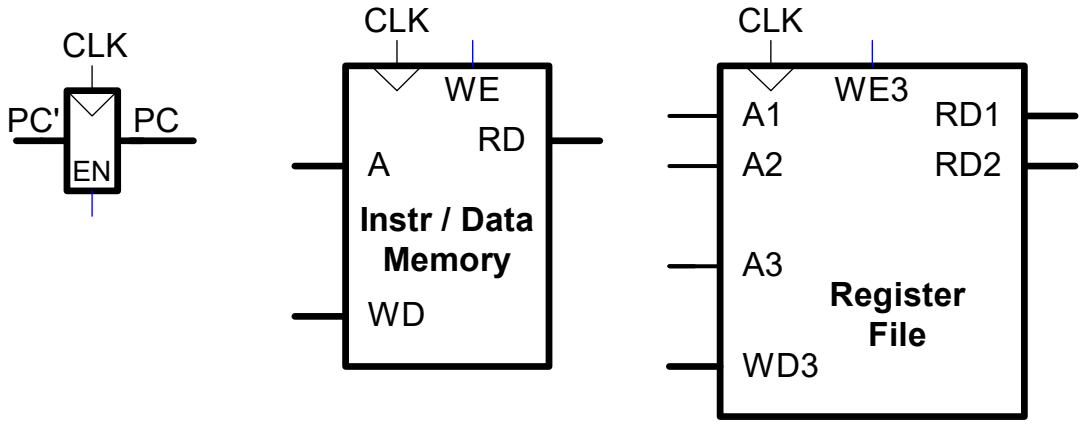


$PC <= $PC + 4

Elementos de estado (memorias)

Las memorias de instrucciones y de datos se juntan en una sola memoria.

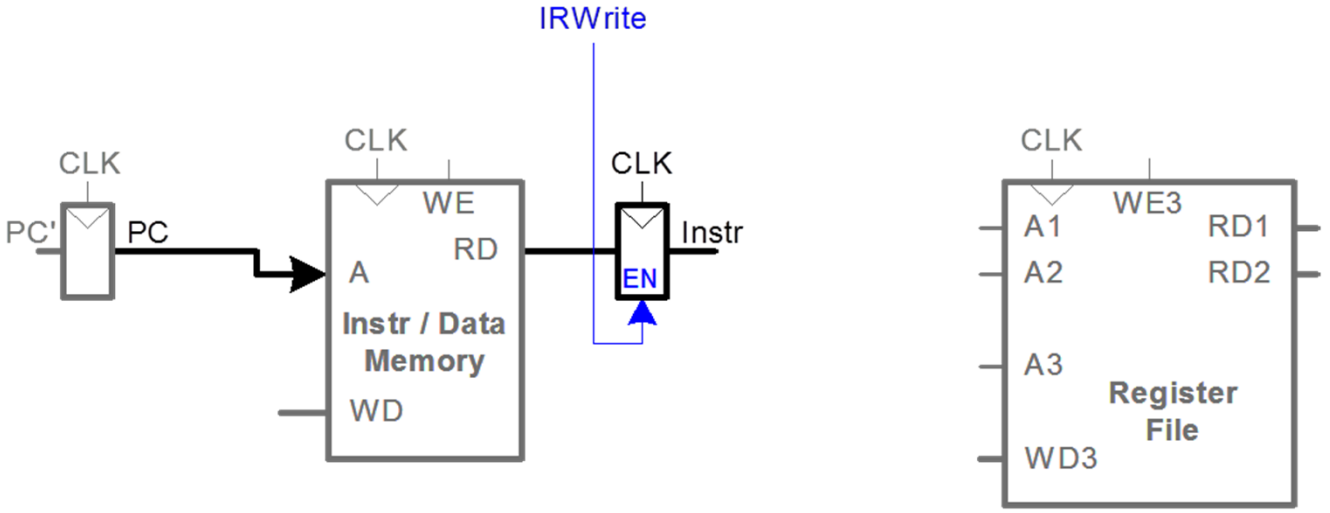
- Se accede a cada elemento del sistema en ciclos distintos del reloj.



Ruta de datos: captura lw

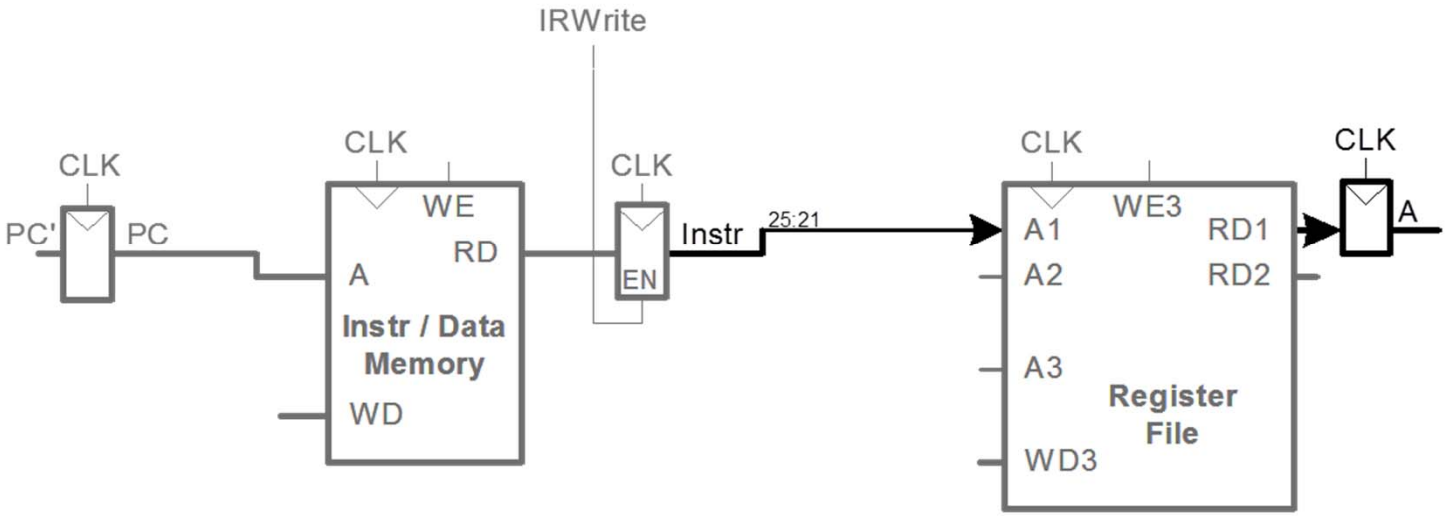
De nuevo empezamos el análisis de la ruta de datos con la instrucción lw

- **CICLO 1:** captura de instrucción (*fetch*).
  - Cada paso es un ciclo de reloj, pero mucho más corto que en el caso uniciclo (en un ciclo se ejecutaba todo).
  - Ahora el resultado de cada paso se registra (registro **Instr**) habilitando la escritura (IRWrite, *enable*) en el ciclo de reloj en que se dispone de la información.



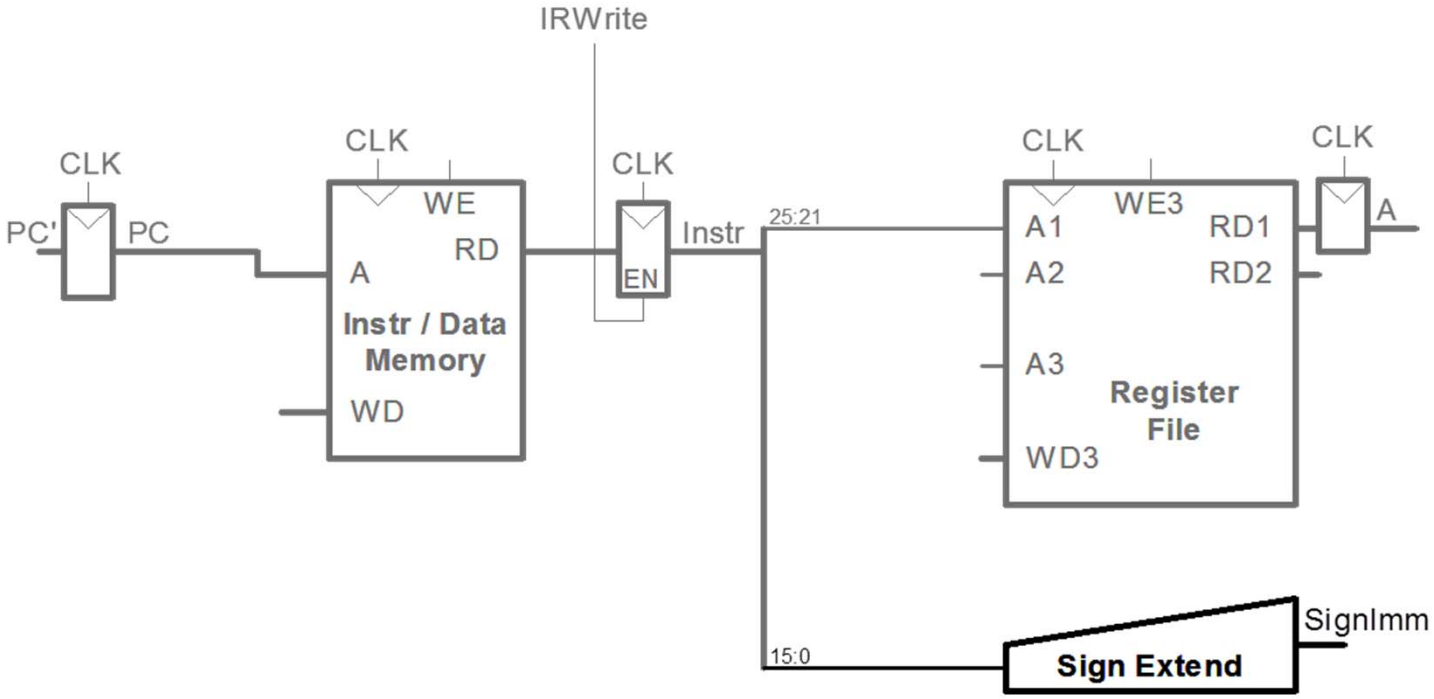
Ruta de datos: lectura de registros lw

- **CICLO 2:** leer los operandos fuente del banco de registros.
  - El resultado (operando 1) se guarda en un nuevo registro (**A**). No tiene habilitación de escritura, así que se actualiza todos los ciclos (rs no cambia durante la instrucción).



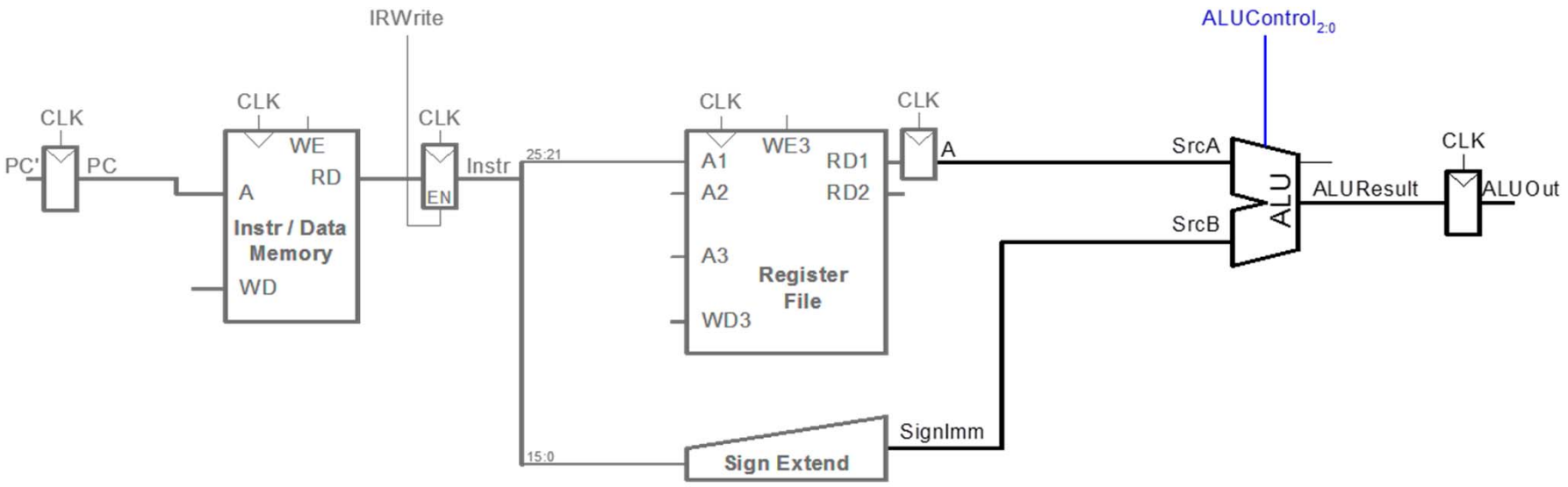
Ruta de datos: dato inmediato lw

- **CICLO 2:** se hace en paralelo a la lectura del banco de registros, durante el ciclo 2.
  - No hace falta guardarlo en registro porque el dato inmediato ya está registrado en Instr.



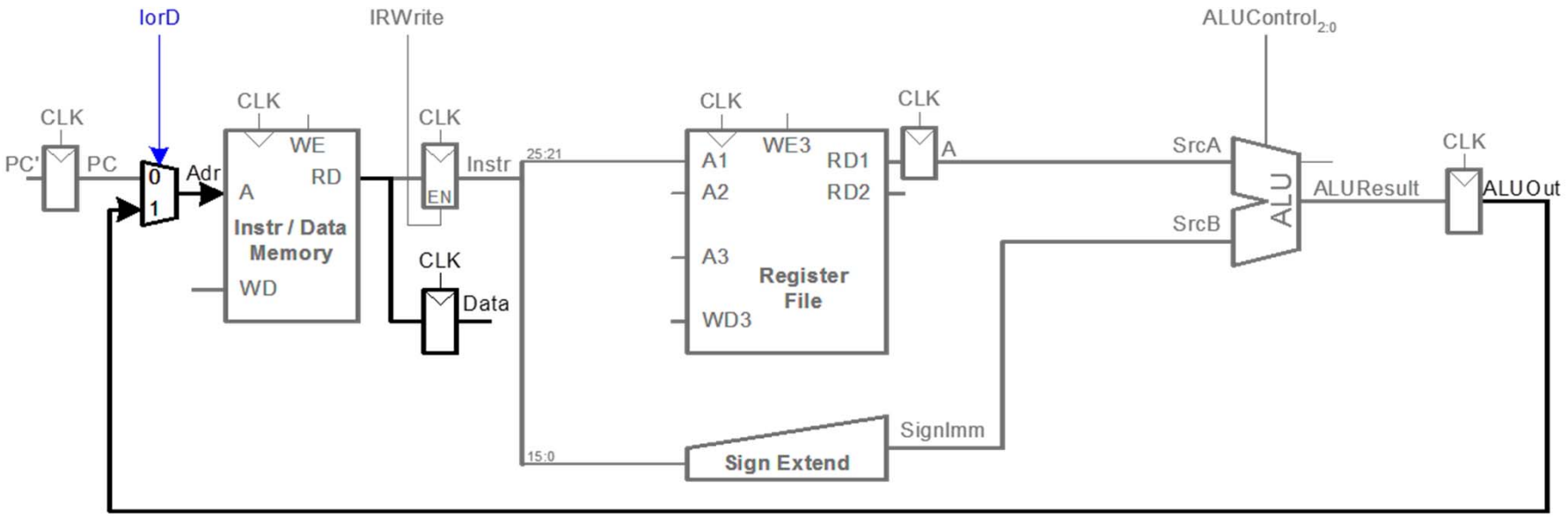
Ruta de datos: dirección lw

- **CICLO 3:** se calcula la dirección del dato sumando en la ALU registro y dato inmediato.
  - Se guarda el resultado en un nuevo registro (**ALUOut**), antes de cambiar de ciclo de reloj.



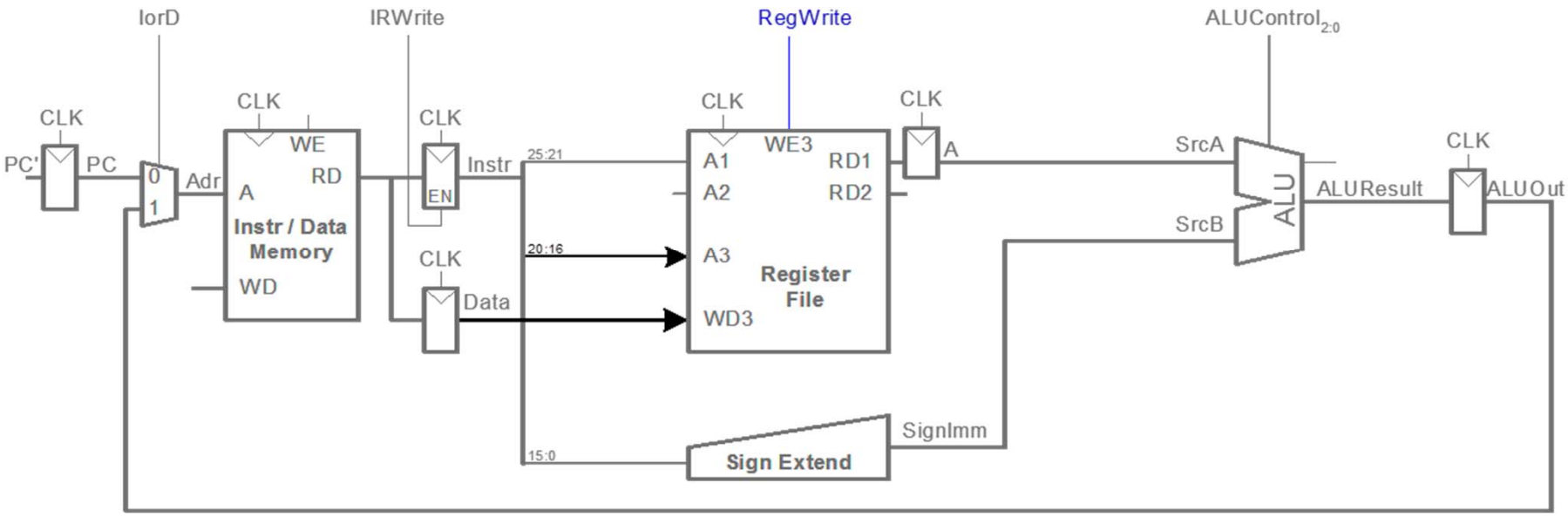
Ruta de datos: leer memoria lw

- **CICLO 4:** se usa la dirección calculada para leer el dato de memoria: nueva señal de control IorD (instrucción o dato).
  - Puede ser la misma memoria que la de código porque durante este ciclo de reloj no hay captura de instrucción.
  - El resultado hay que guardarlo en un nuevo registro (**Data**).



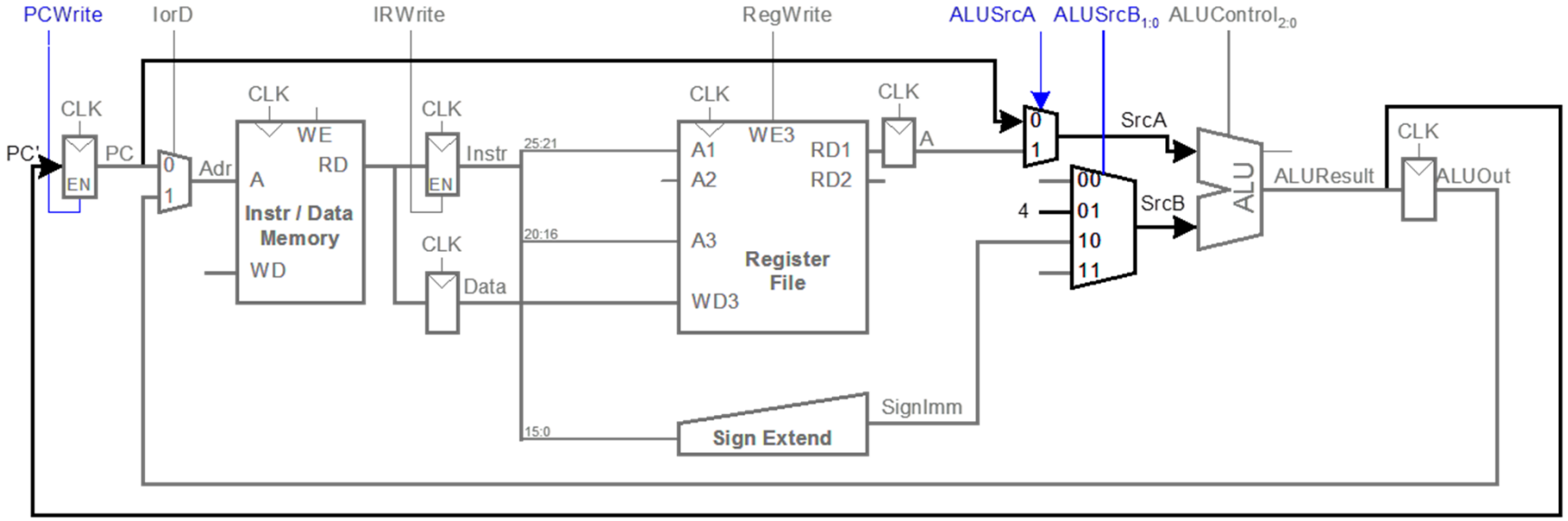
Ruta de datos: guardar registro lw

- **CICLO 5:** el dato leído se escribe en el registro destino.
  - No hay registro adicional, porque ya se guarda el resultado en el propio banco de registros.



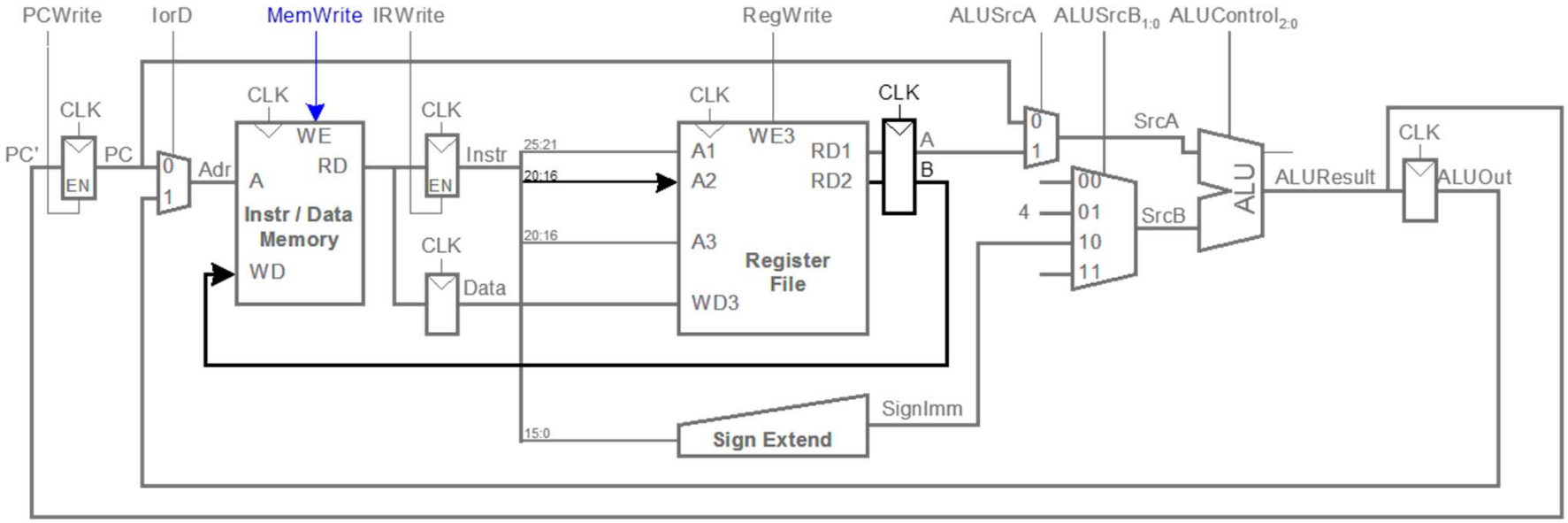
Ruta de datos: incrementar PC lw

- **CICLO 1:** durante el primer ciclo de cada instrucción se guarda el nuevo PC (provisional en los saltos) para la próxima instrucción, que es PC + 4.
  - En vez de utilizar un sumador específico para sumar 4 se usa la ALU, que no hace nada más durante este ciclo de reloj.
  - Requiere que PC y 4 lleguen a los dos operandos de la ALU (más multiplexores).



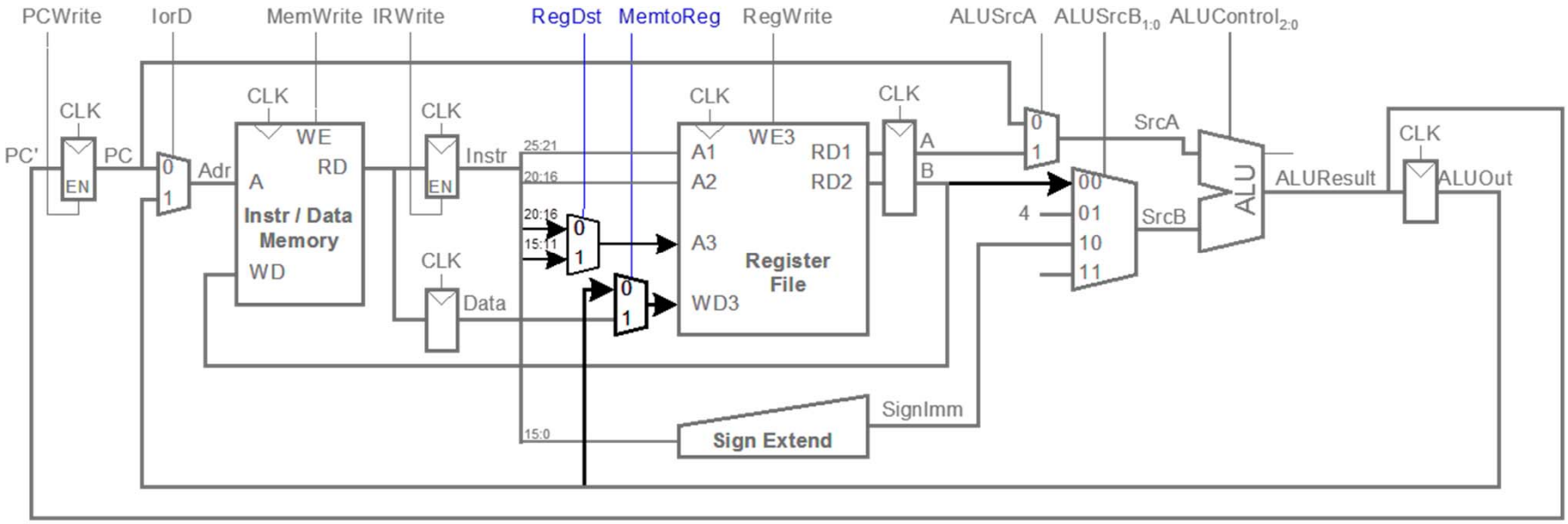
Ruta de datos: sw

- **Ciclo 2:** El operando tiene que ser registrado (registro **B**) porque la lectura de registro y la escritura en memoria se ejecutan en ciclos distintos.
- **Ciclo 3:** La dirección de memoria se guarda en (**ALUOut**).
- **Ciclo 4:** El registro rt (ahora en B) se escribe en memoria (MemWrite), así que debe llegar a la entrada WD de memoria.



Ruta de datos: Tipo-R

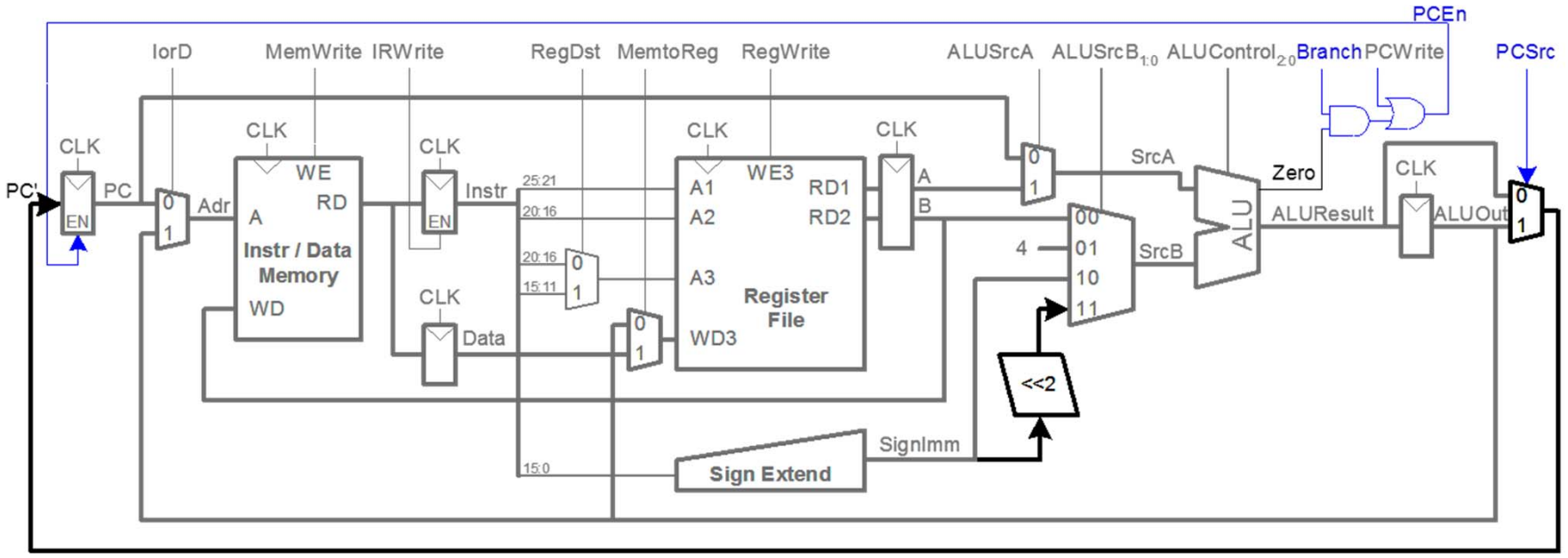
- **Ciclo 2:** Leer y registrar rs y rt, los dos operandos de entrada de la ALU.
- **Ciclo 3:** El resultado ALUResult, se registra en *ALUOut.*
- **Ciclo 4:** Ahora el contenido de ALUOut se escribe en el banco de registros (MemtoReg=‘0’). Se escribe en rd en lugar de rt (RegDst=‘1’).



Ruta de datos: beq

Cálculo de la dirección de salto: BTA = (PC+4) + (Sign Imm << 2)

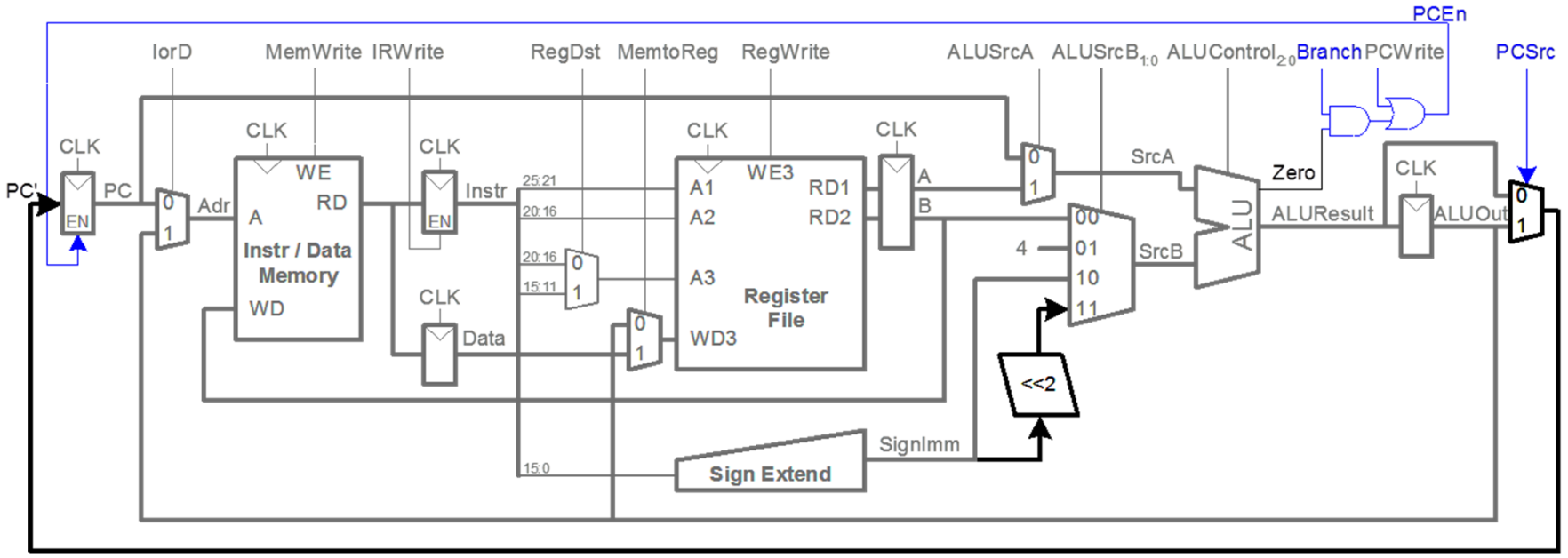
- **Ciclo 1:** PC <= PC+4 (en la ALU), habilita PC (PCWrite='1') .
- **Ciclo 2:** ALUOut <= (PC+4) + (Sign Imm << 2) (en la ALU).
- **Ciclo 3:** Se restan los operandos para determinar Z. ALUOut se envía a PC (PCSrc='1') por si se produce el salto (Branch AND Z = '1').



Ruta de datos: beq (decisión)

- Se decide si se salta o no con la bandera Z (Zero).
- Se actualiza el PC en dos casos:
  - ✓ PCWrite='1', cada ciclo 1 para conseguir PC+4 (**siempre**)
  - ✓ Zero and Branch (Z='1' y es un salto condicional).

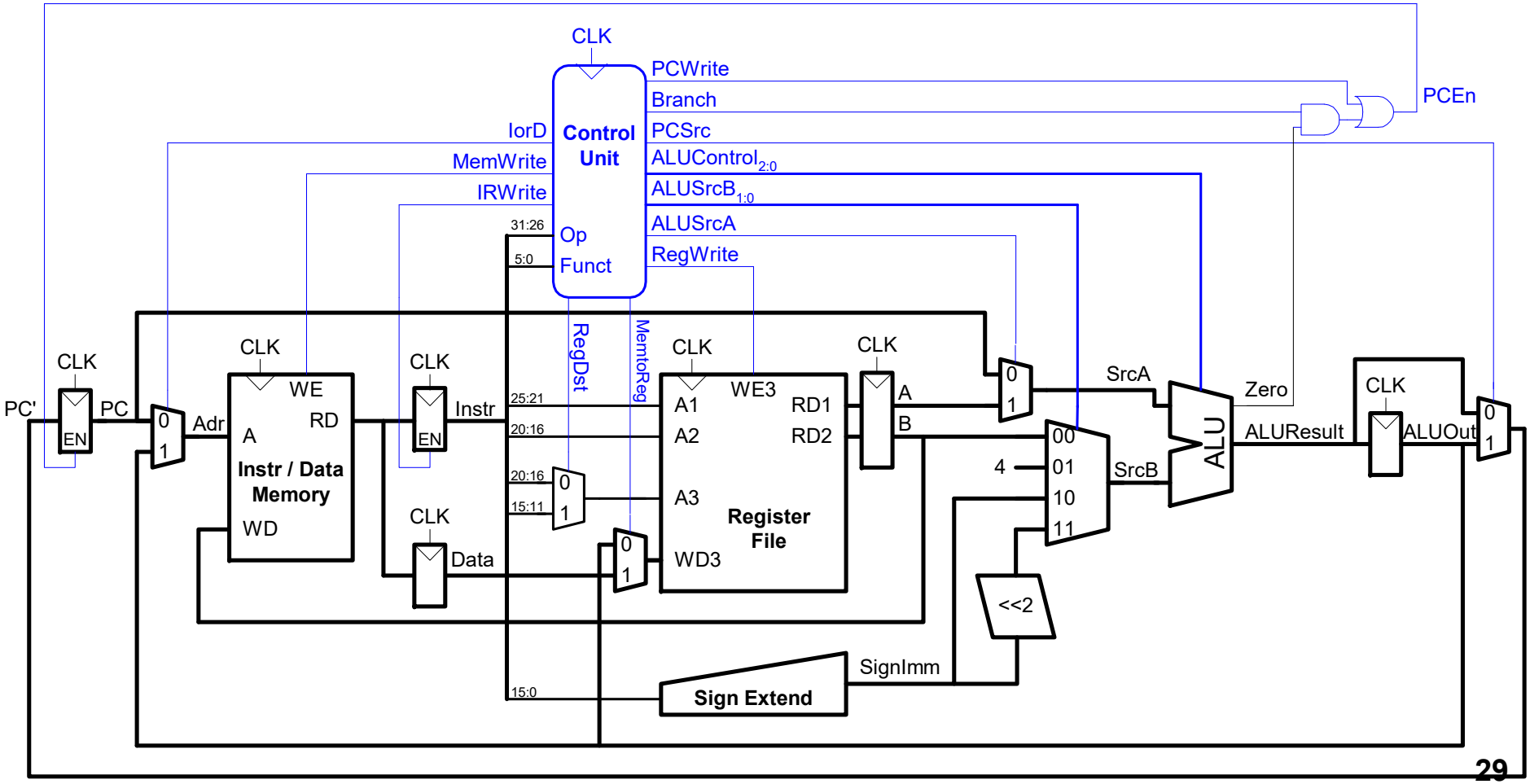
    Branch se debe generar en el ciclo 3 de beq, y si finalmente se salta o no depende de Zero.



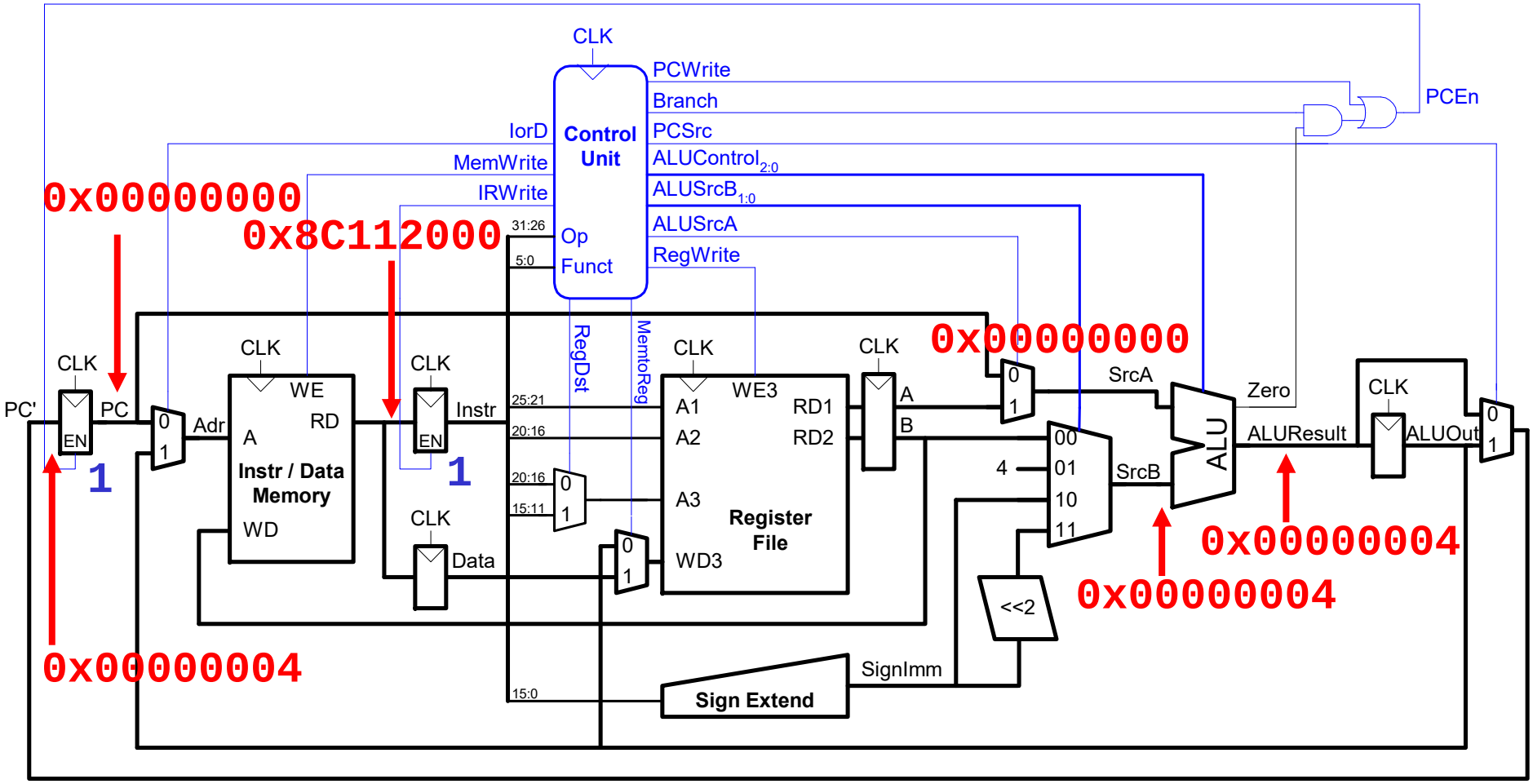
Ruta de datos y de control multiciclo

P

En la arquitectura multiciclo, las señales de control se generan según la instrucción (hay que decodificar opcode y funct) **y varían según el ciclo de reloj**.

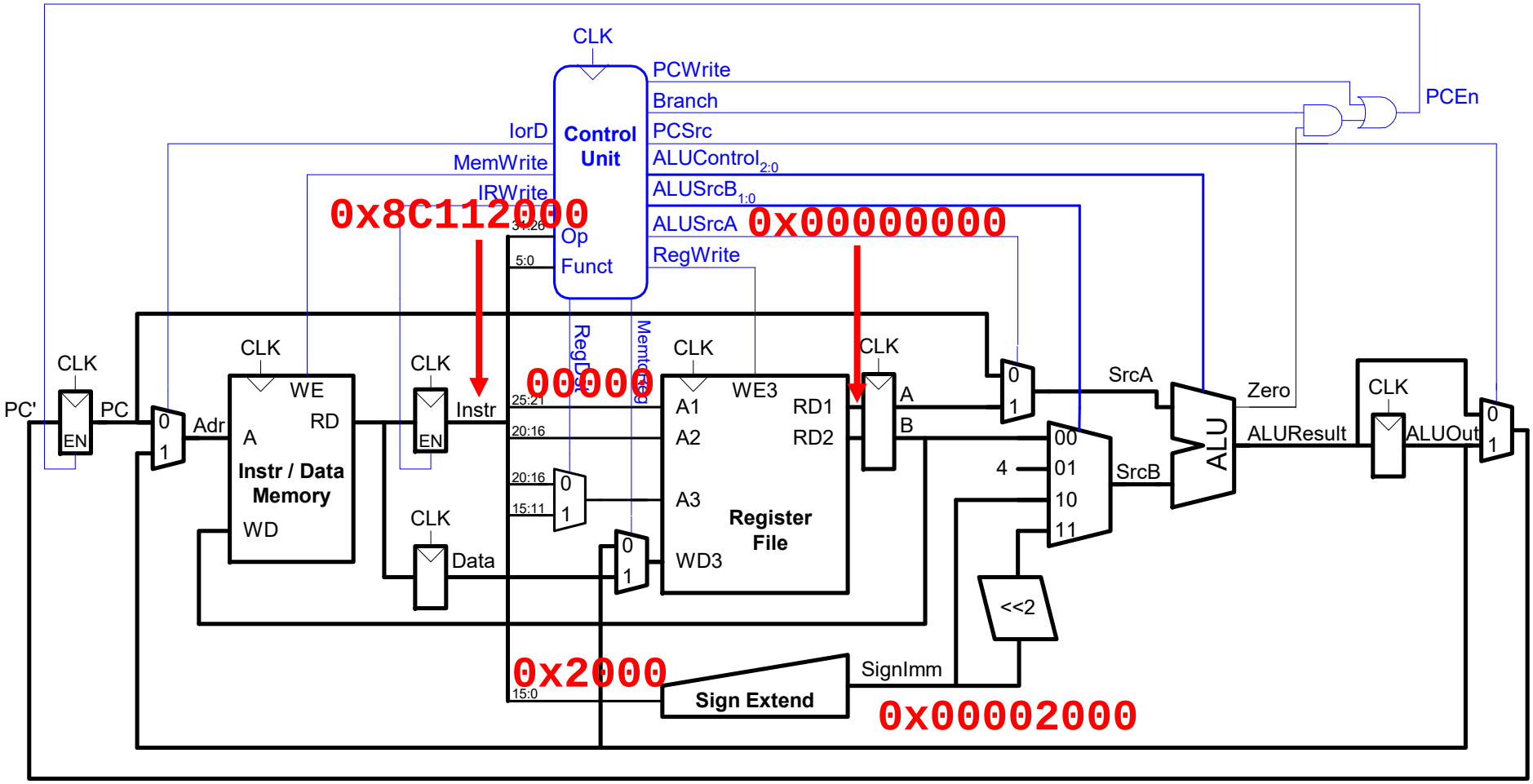


Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 1



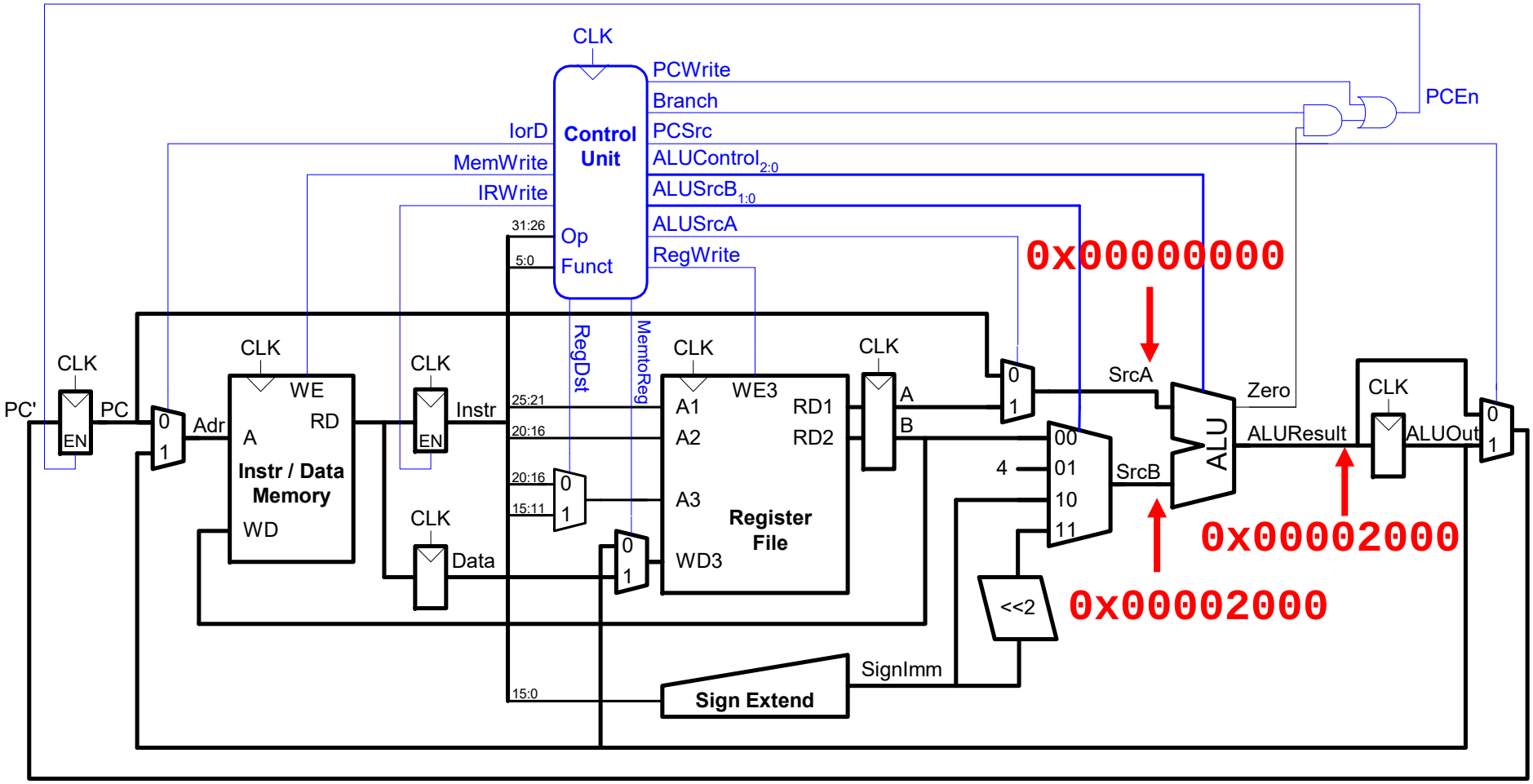
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 2



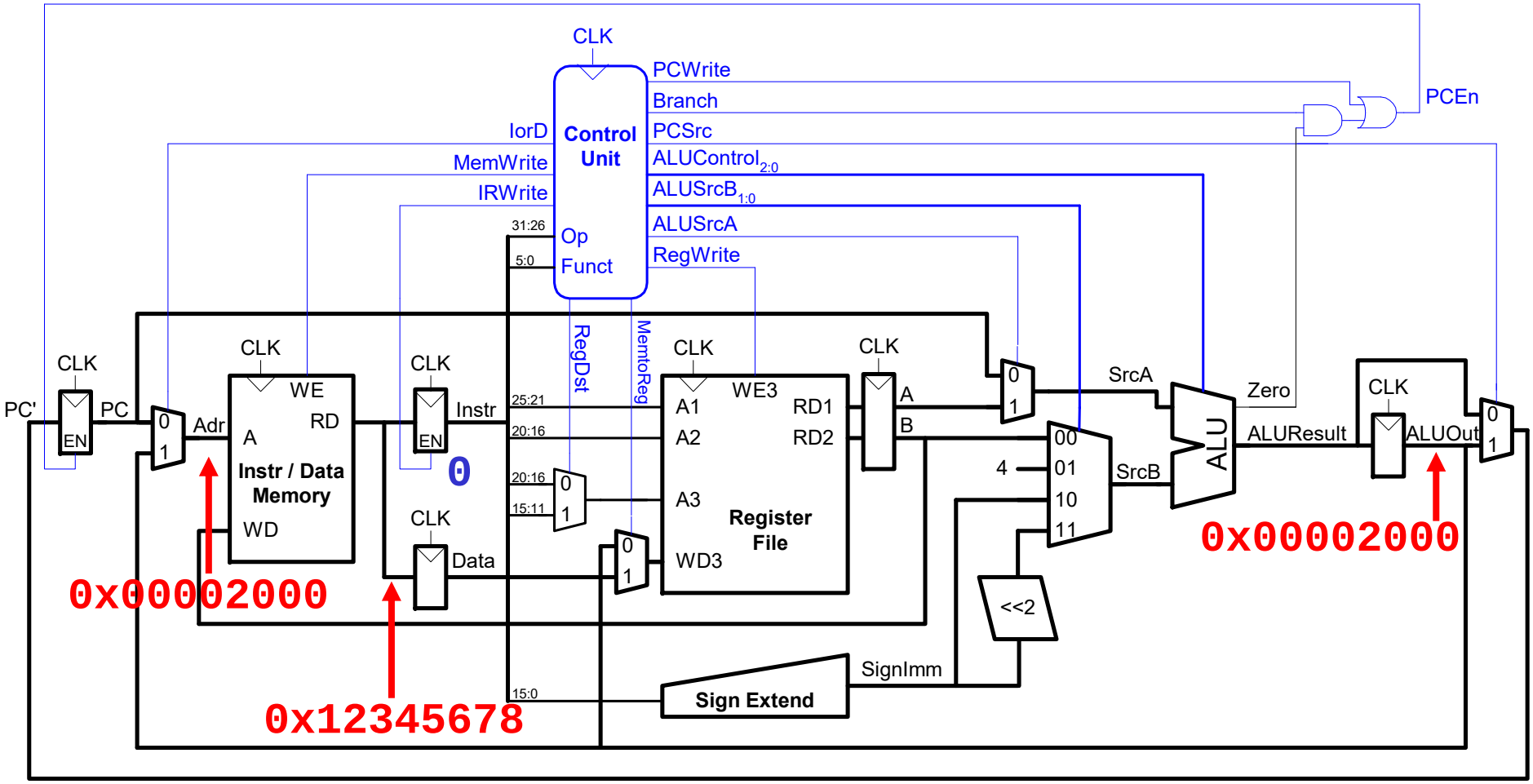
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 3



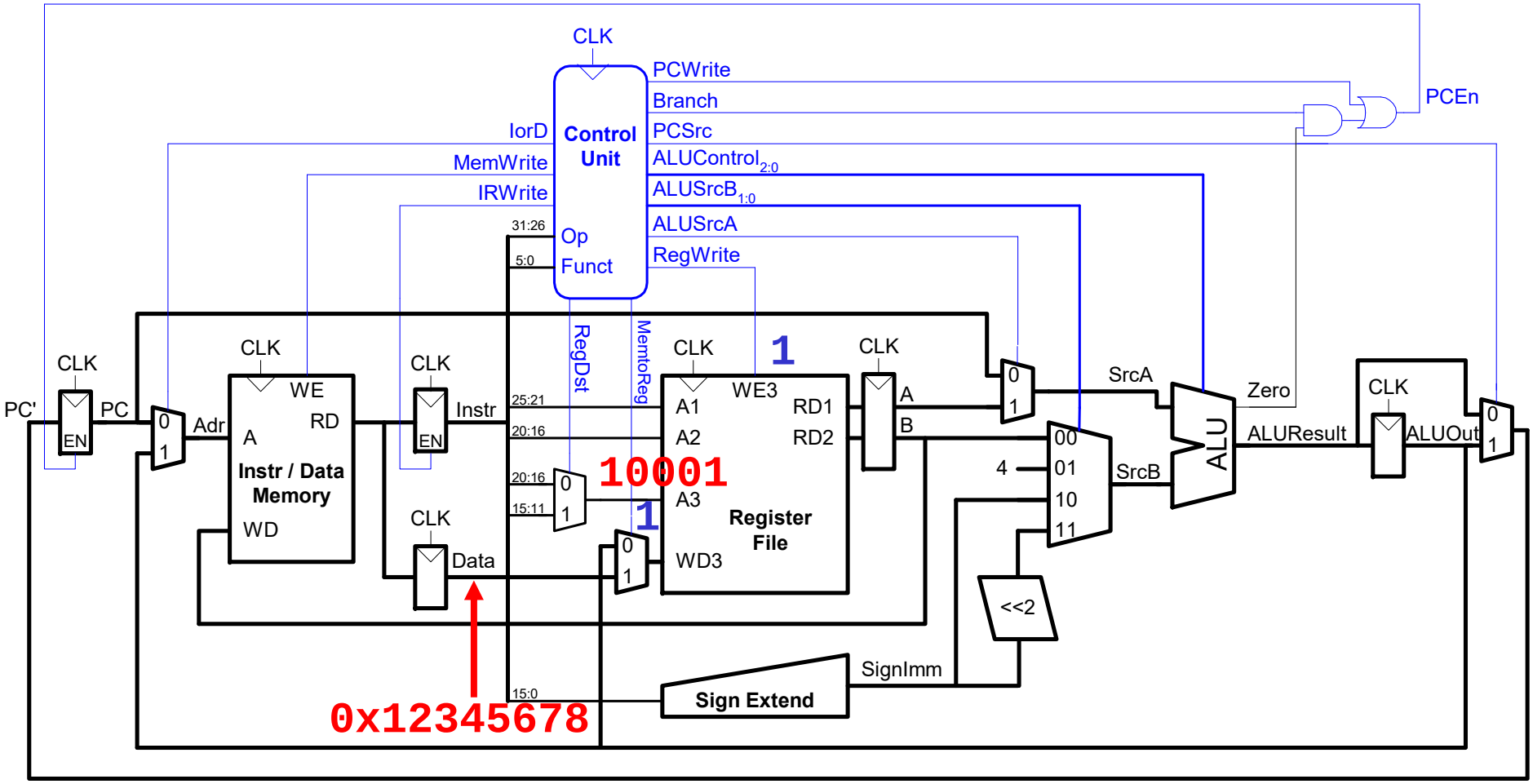
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 4



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 5



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

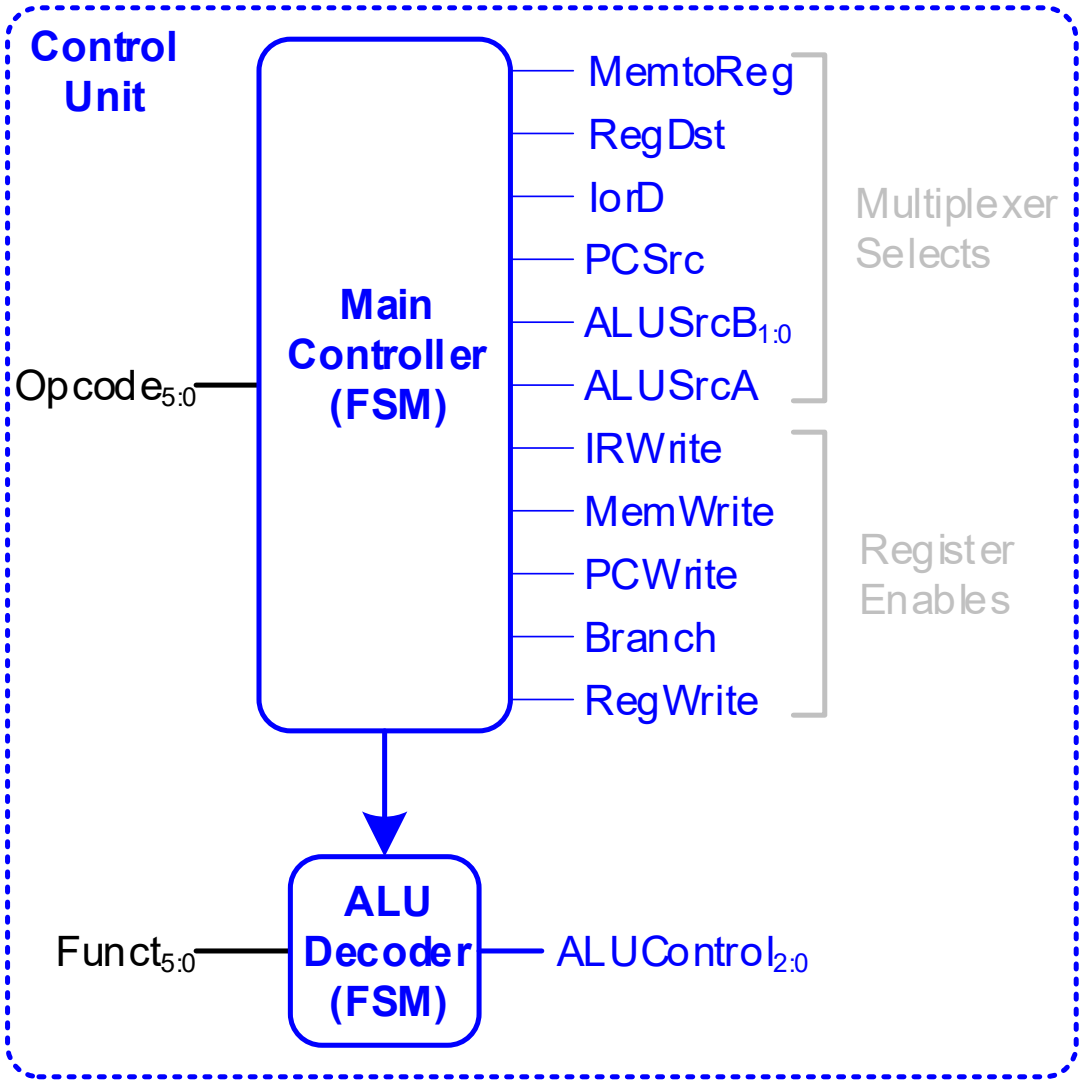
Índice

- Resumen arquitectura MIPS uniciclo
- Ruta de datos multiciclo
- **Control multiciclo**
- Añadir más instrucciones

Planificación de instrucciones

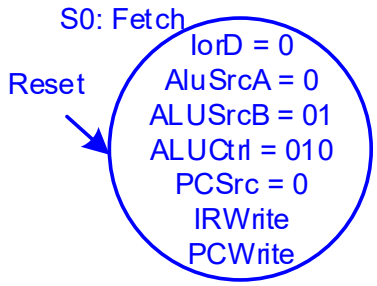
	lw	sw	R-type	beq
Ciclo 1	Captura de la instrucción. PC <= PC + 4			
Ciclo 2	Lectura operandos.			**a.** Lectura operandos. **b.** Cálculo BTA. (no escribe en PC)
Ciclo 3	Cálculo de dirección de Mem. Datos		Cálculo ALU	**a.** Resta en ALU **b.** Si Z=1, actualiza PC.
Ciclo 4	Lectura Mem. Datos	Escritura Mem. Datos	Escritura Reg.	
Ciclo 5	Escritura Reg.			

Unidad de control

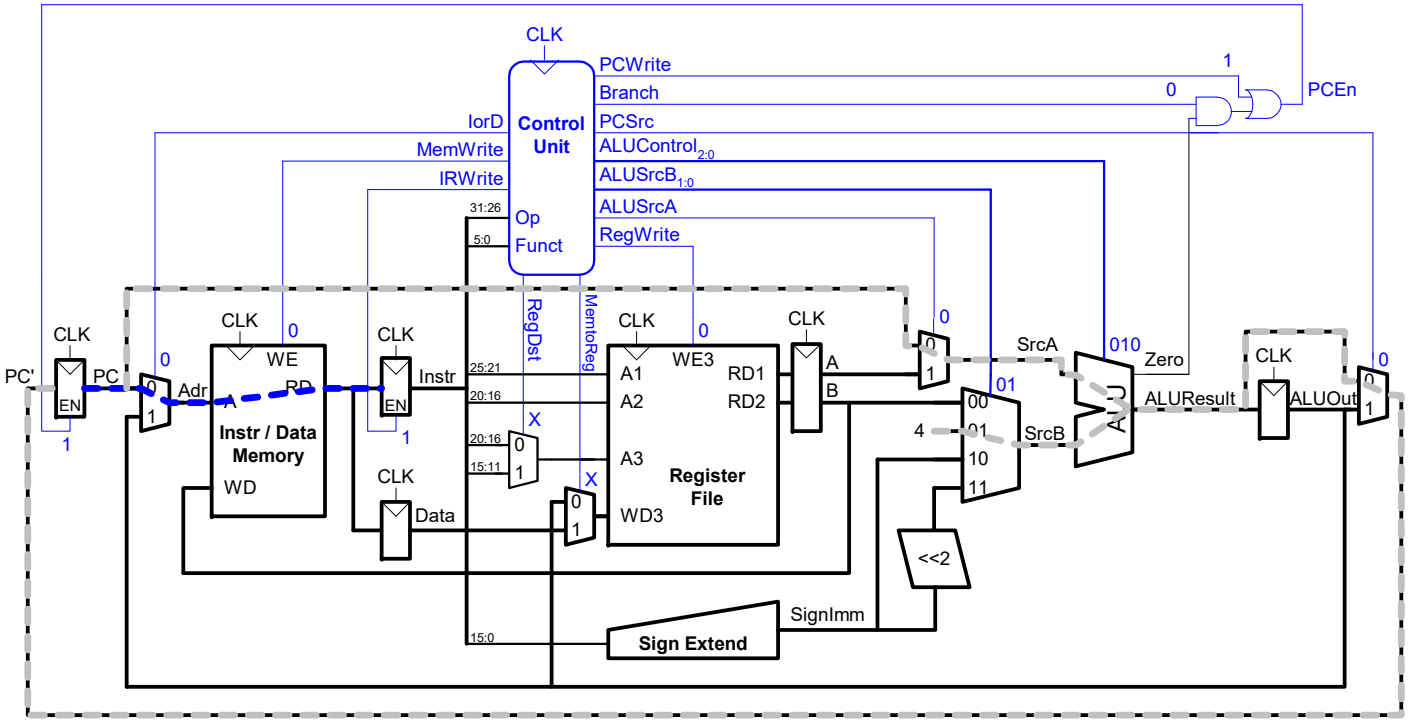


- Como hay que ir variando las señales de control según el ciclo de reloj, ya no puede ser combinacional.
- La parte de ALU Decoder sigue igual. Si se quiere que la ALU haga varias operaciones durante una instrucción se cambia ALUControl en el ciclo que corresponda.
- ***Main Controller*** es una máquina de estados finita (FSM).

Máquina de estados del control (ciclo 1)

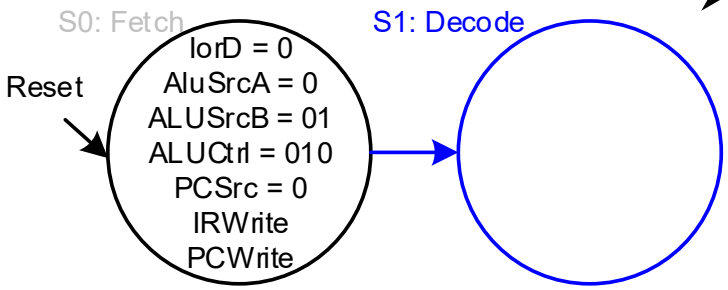


- **Ciclo-1:** es igual para todas las instrucciones.
  - ✓ Captura de la instrucción desde memoria.
  - ✓ Actualización de PC con PC+4 (ALU suma).

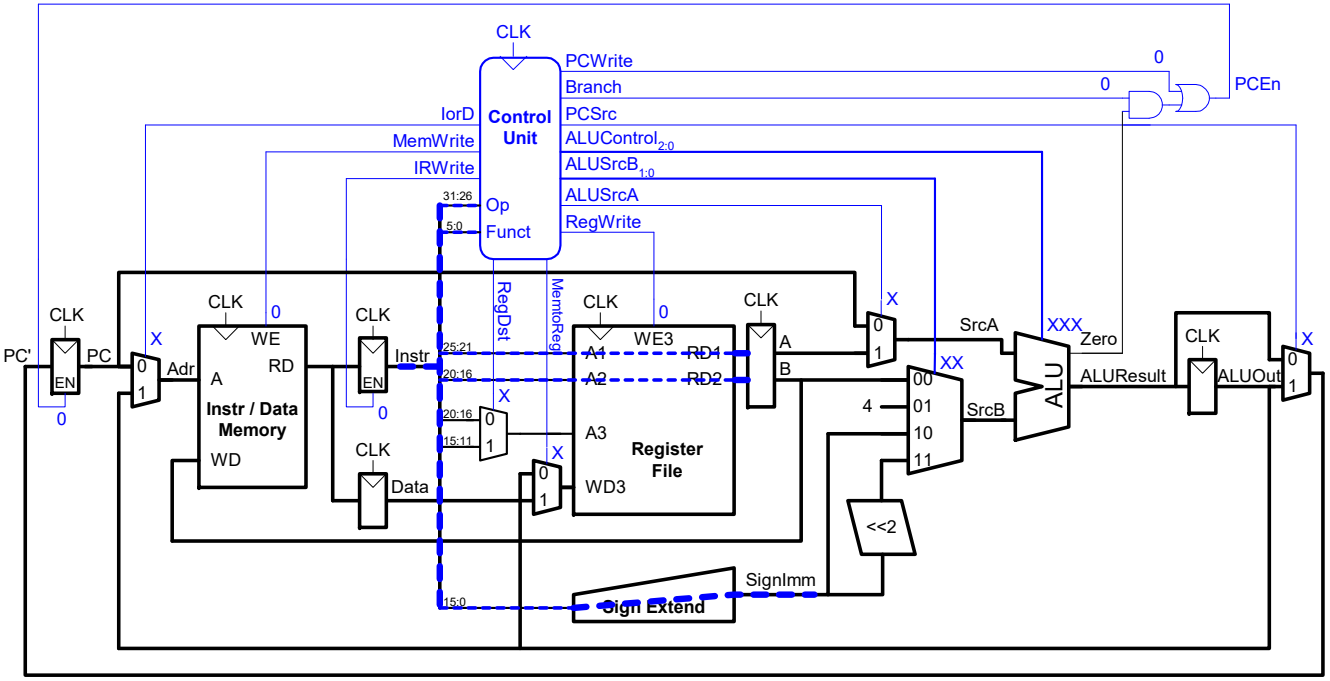


Nota: sólo se destacan las señales de selección de mux relevantes y los enables de registros si se activan (se ponen a '1').

Control: ciclo 2. Todas

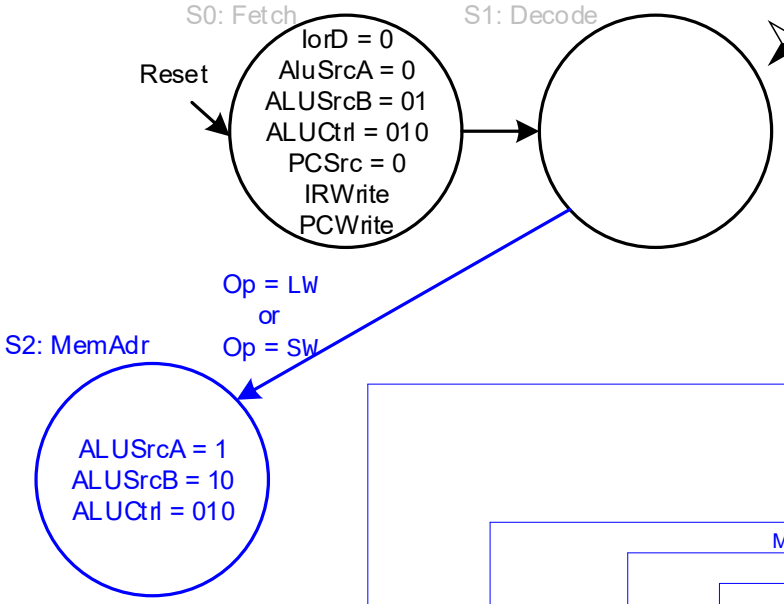


- **Ciclo-2:** Se decodifica la instrucción y se capturan operandos.
  - ✓ Como el primero, este ciclo es igual para todas las instrucciones. Se registran A y B (aunque no se vayan a usar). No tienen enable, así que siempre se registran.

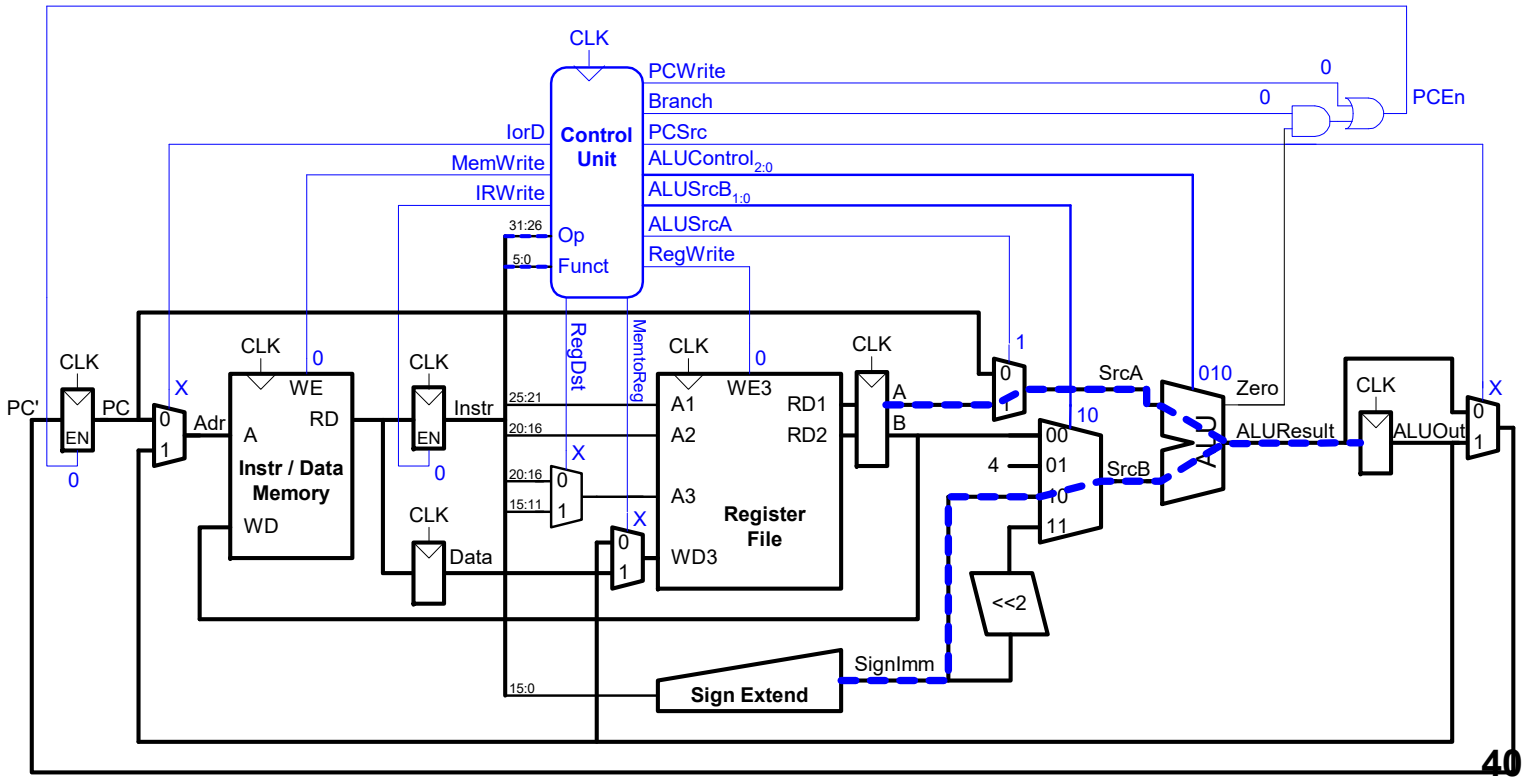


✓A partir de aquí, la evolución de la máquina es distinta para cada instrucción.

Control: ciclo 3, instrucciones lw/sw



Ciclo-3 (para lw o sw): se calcula la dirección de memoria de datos, que se guarda en ALUOut (no tiene enable, siempre se guarda).

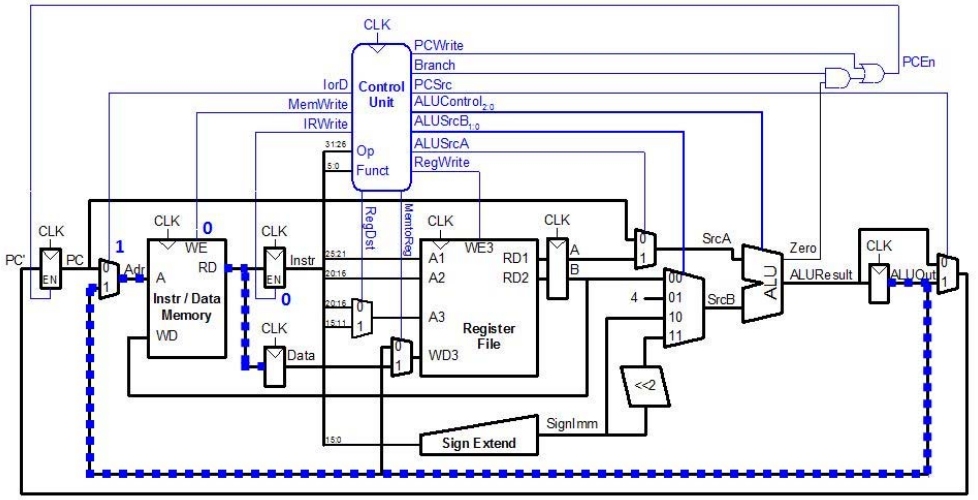


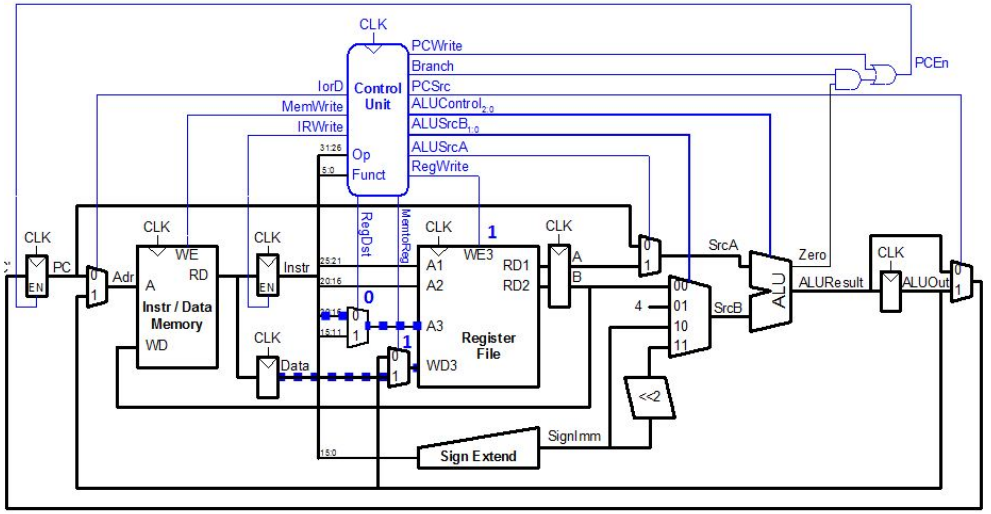
Control: ciclos 4 y 5, instrucción lw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria para leer el dato (IorD='1'), y se guarda el dato leído en Data (sin enable, siempre guarda).

- **Ciclo-5:** el dato (Data) se guarda en registro (registro rt es el destino).

Final de instrucción (vuelta a S0).

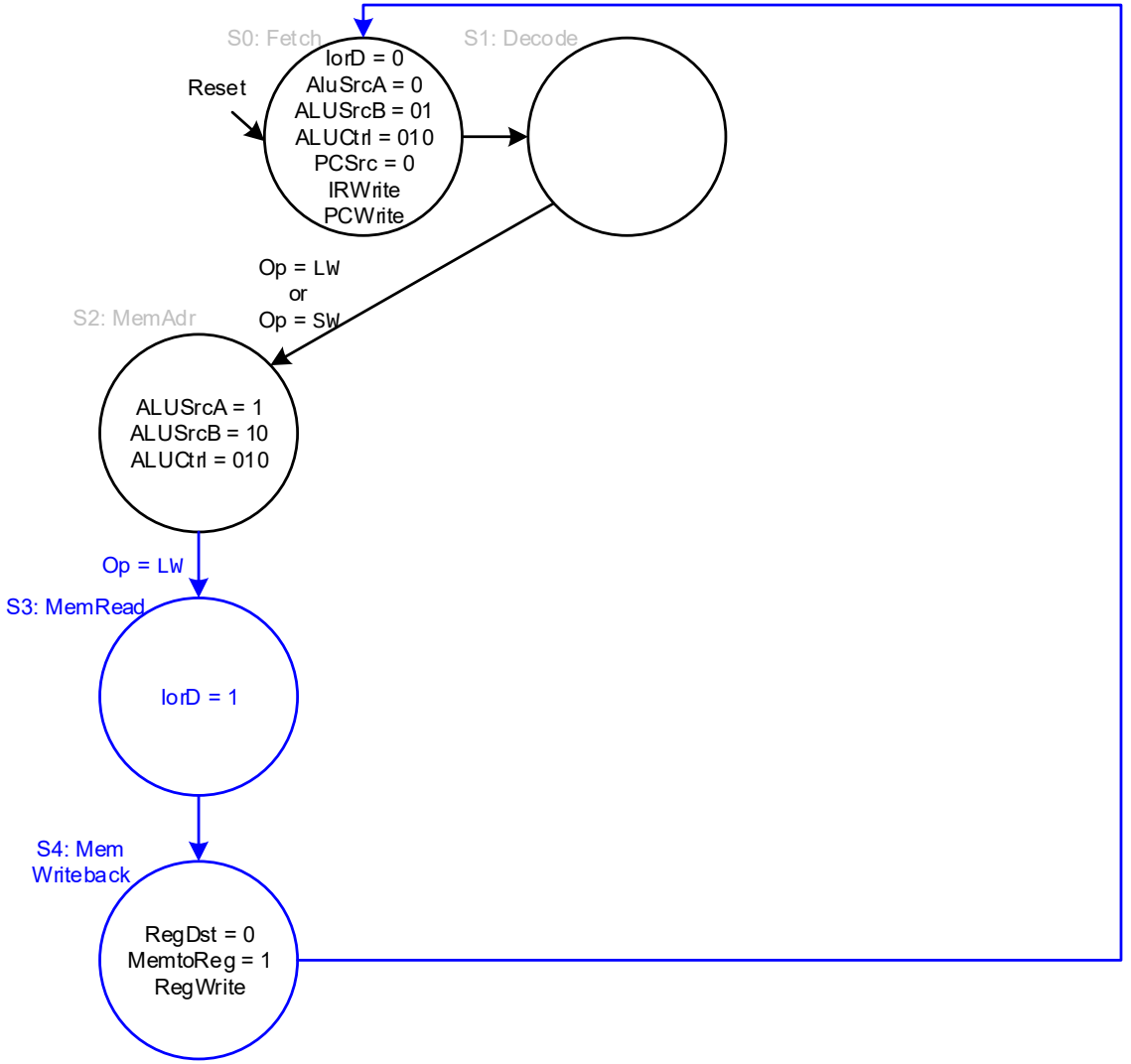




Control: ciclos 4 y 5, instrucción lw

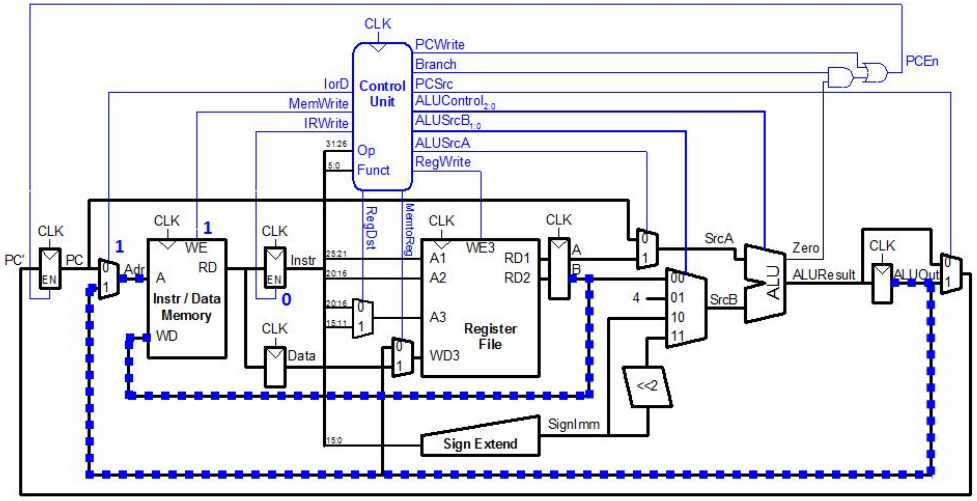
- **Ciclo-4:** la dirección (ALUOut) se manda a memoria para leer el dato (IorD='1'), y se guarda el dato leído en Data (sin enable, siempre guarda).
- **Ciclo-5:** el dato (Data) se guarda en registro (registro rt es el destino).

Final de instrucción (vuelta a S0).



Control: ciclo 4, instrucción sw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria (IorD='1') para escribir el dato. La memoria se activa para escritura. El dato escrito es rt, guardado en B.



Final de instrucción (vuelta a S0).

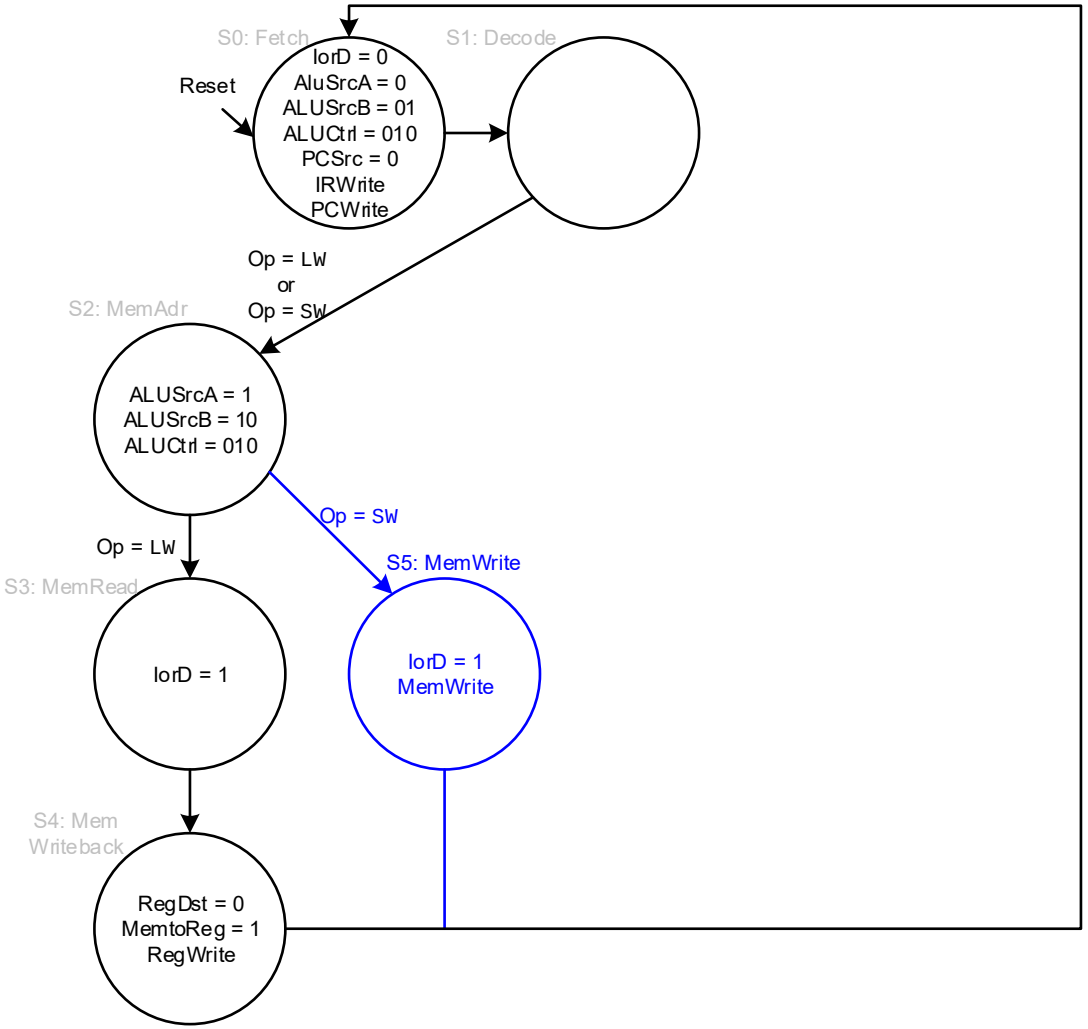
La instrucción sw sólo necesita 4 ciclos, no 5.

Control: ciclo 4, instrucción sw

- **Ciclo-4:** la dirección (ALUOut) se manda a memoria (IorD='1') para escribir el dato. La memoria se activa para escritura. El dato escrito es rt, guardado en B.

Final de instrucción (vuelta a S0).

La instrucción sw sólo necesita 4 ciclos, no 5.



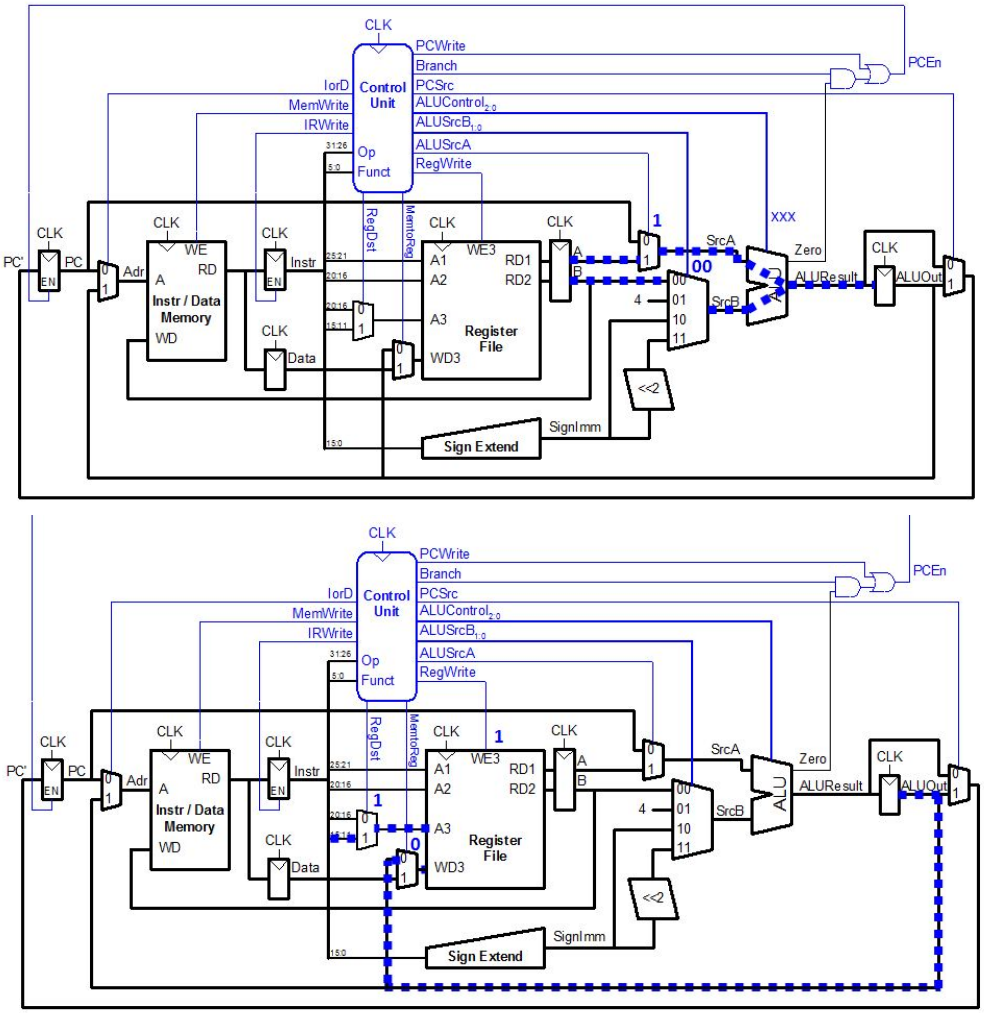
Control: ciclos 3 y 4, instrucciones Tipo-R

- **Ciclo-3:** se obtiene el resultado en la ALU. Se usan A y B (datos de registros) y la operación de la ALU depende de funct.

- **Ciclo-4:** el resultado, que ha quedado en ALUOut, se manda al registro rd.

  Final de instrucción (vuelta a S0).

  Se ejecutan en 4 ciclos.



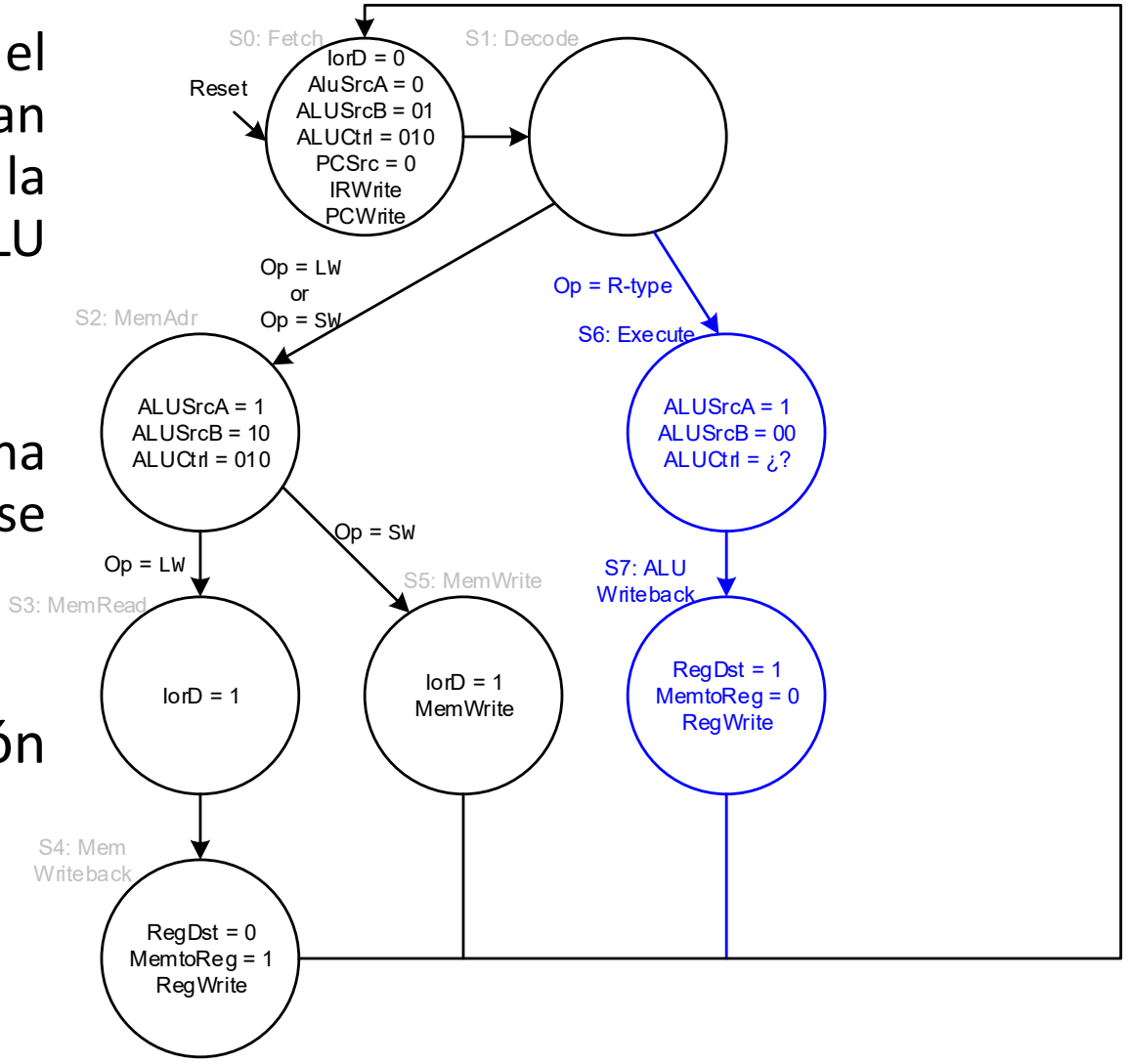
Control: ciclos 3 y 4, instrucciones Tipo-R

- **Ciclo-3:** se obtiene el resultado en la ALU. Se usan A y B (datos de registros) y la operación de la ALU depende de funct.

- **Ciclo-4:** el resultado, que ha quedado en ALUOut, se manda al registro rd.

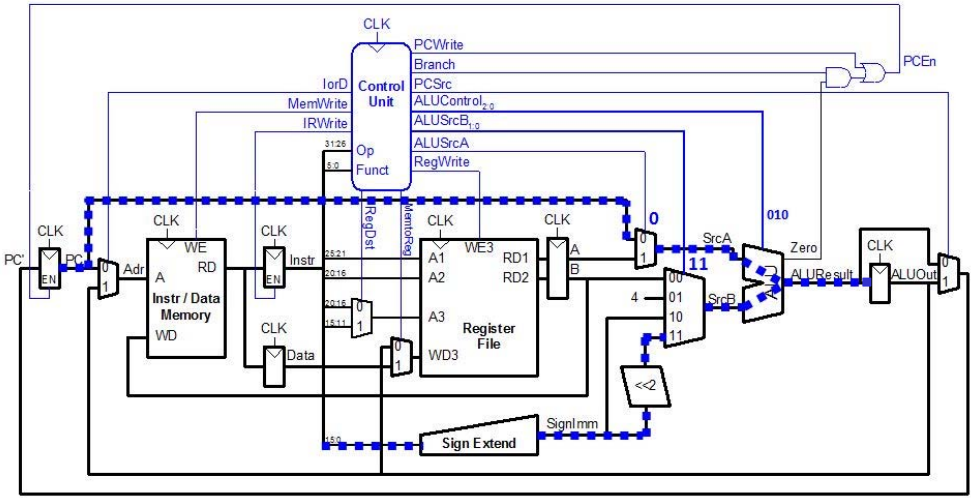
Final de instrucción (vuelta a S0).

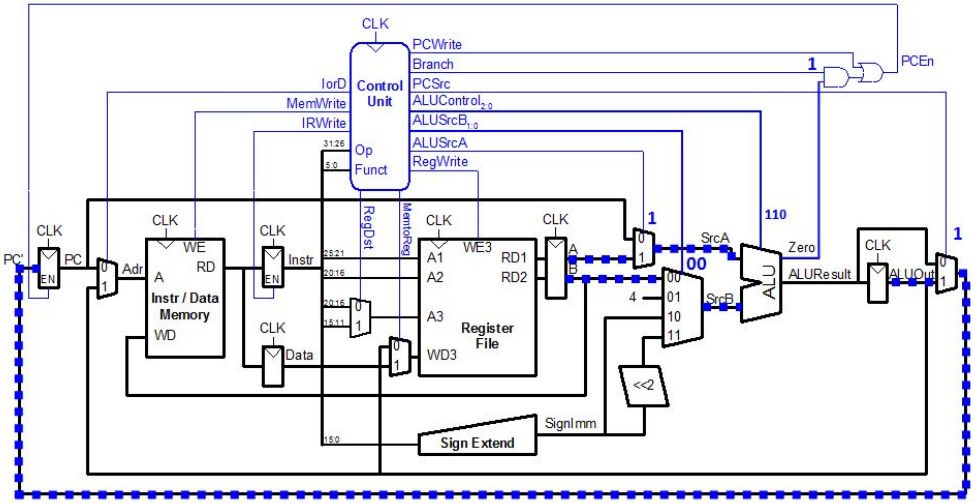
Se ejecutan en 4 ciclos.



Control beq, ciclo 2

- **Ciclo 1:** la ALU calcula PC+4
- **Ciclo-2:**, se calcula la dirección de salto: SrcA=0 y SrcB=11. Se hace en **todas** las instrucciones, y si no es un salto, simplemente se descarta ALUOut, producido en ciclo 2.
- **Ciclo 3:** se activa Branch. En la ALU se restan los registros a comparar. De esta forma se activa o no la bandera Z, y por tanto se activará o no PCEn.

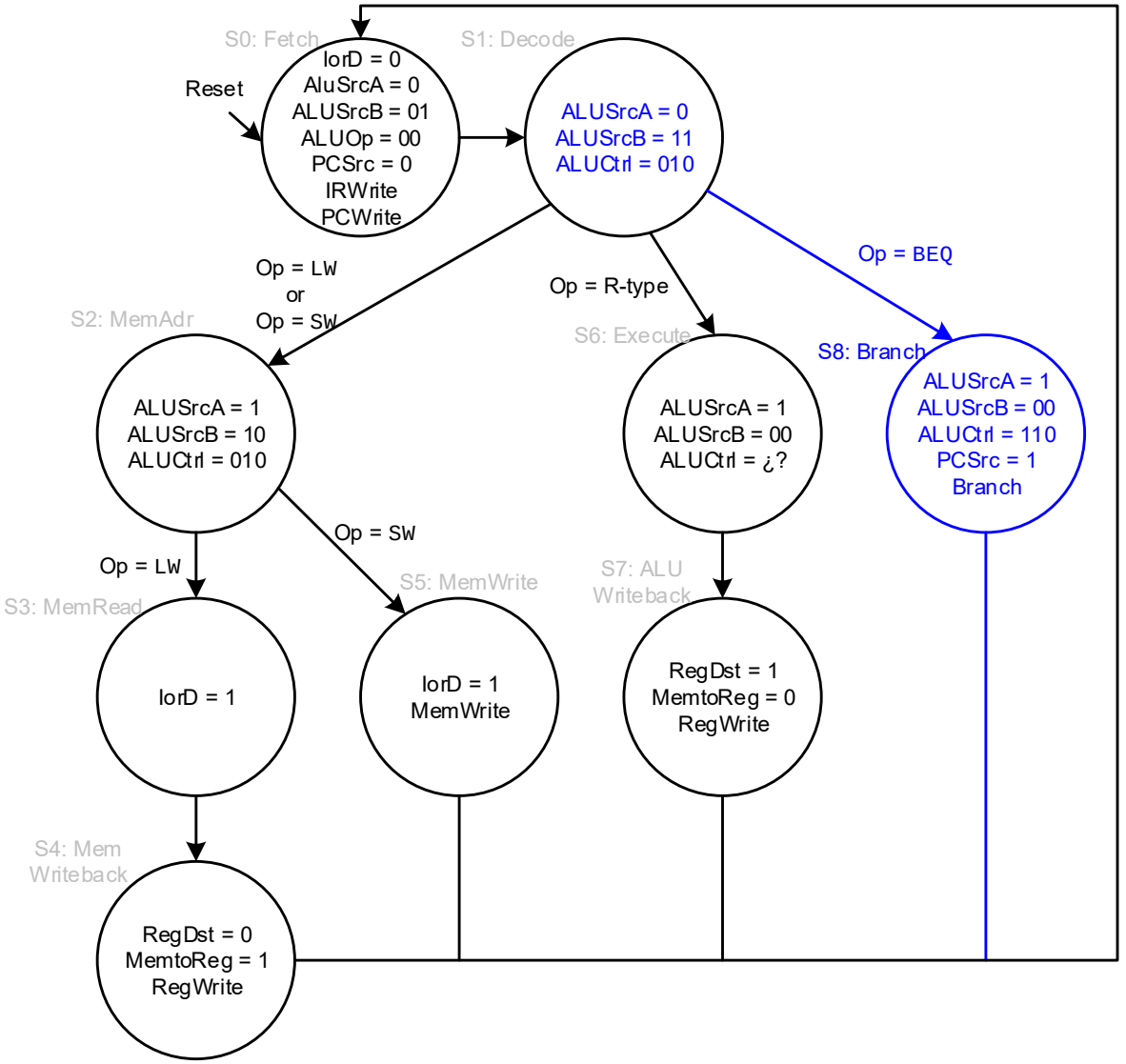




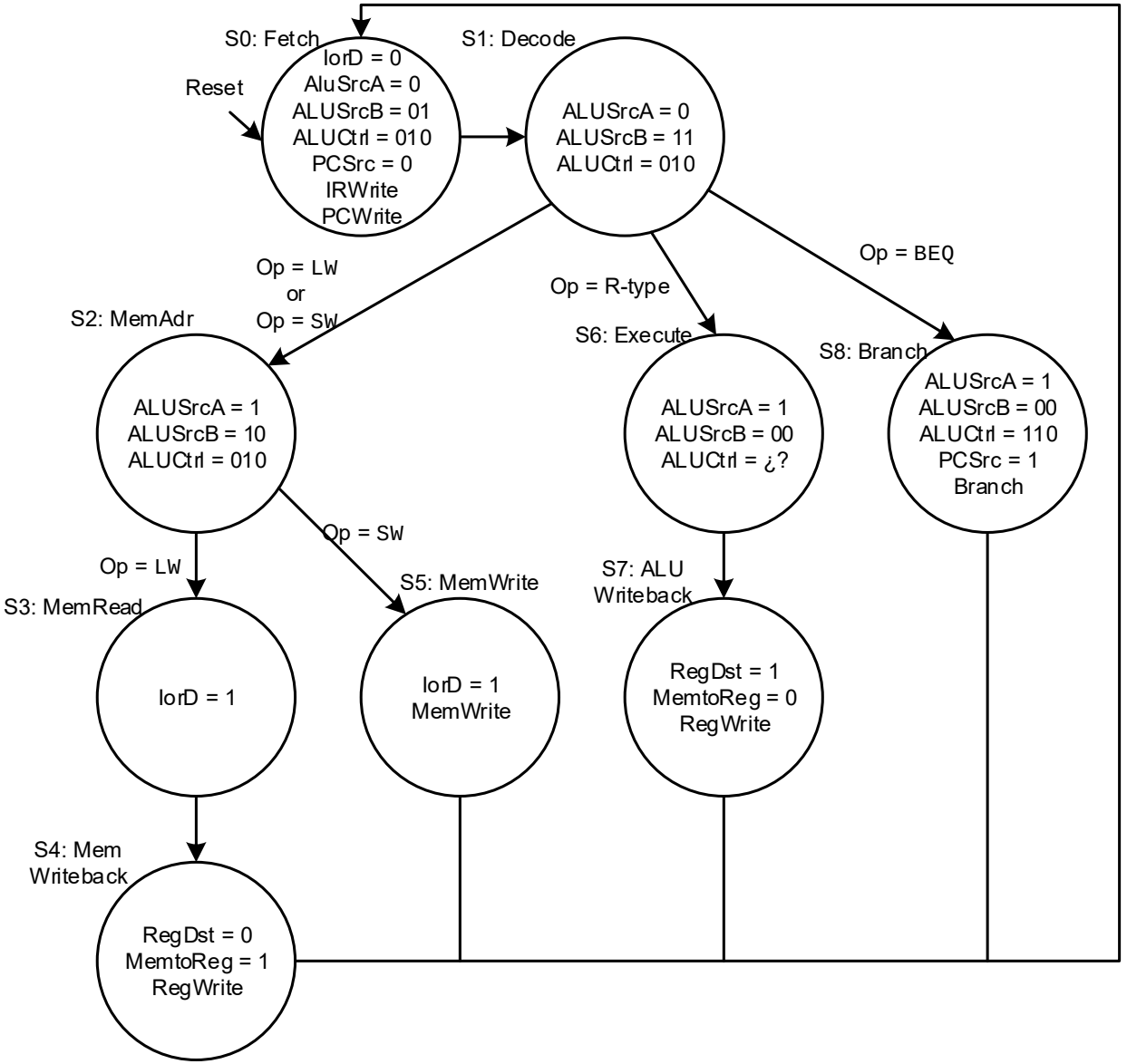
Control beq, ciclo 3

✓Sólo se usan 3 ciclos, a pesar de haber 3 usos de la ALU.

✓Si finalmente hay salto, PCEn estará activa. Hay que actualizar PC con ALUOut, dirección del salto calculada en ciclo 2.



Resumen: control multiciclo (sin instrucciones añadidas)



Planificación de instrucciones

P

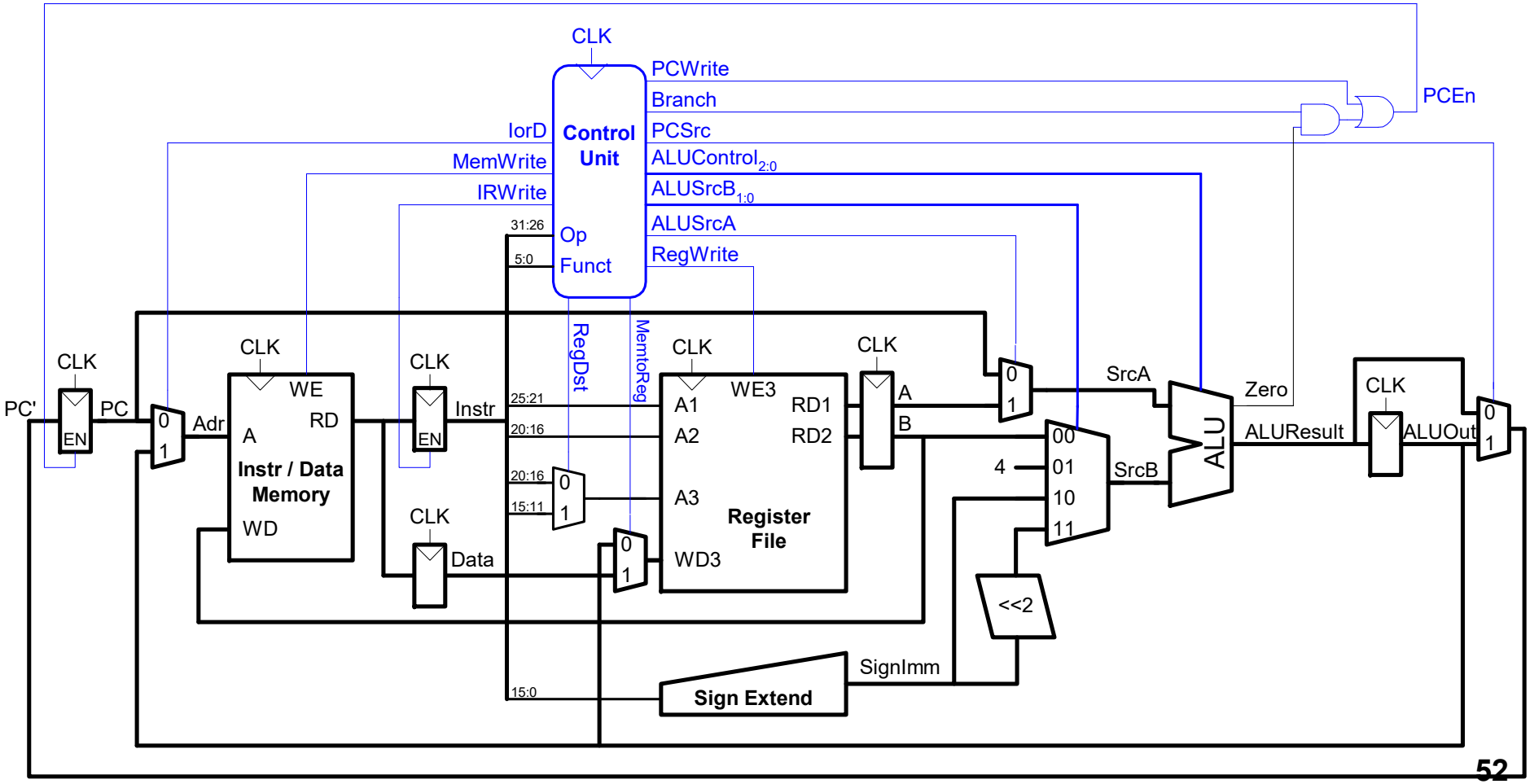
	lw	sw	R-type	beq
Ciclo 1	Captura de instrucción. PC <= PC + 4			
Ciclo 2	**a.** Lectura operandos. **b.** Cálculo BTA (no se guarda en PC)			
Ciclo 3	Cálculo de dirección de Mem. Datos		Cálculo ALU	**a.** Resta en ALU **b.** Si Z=1, actualiza PC.
Ciclo 4	Lectura Mem. Datos	Escritura Mem. Datos	Escritura Reg.	
Ciclo 5	Escritura Reg.			

Índice

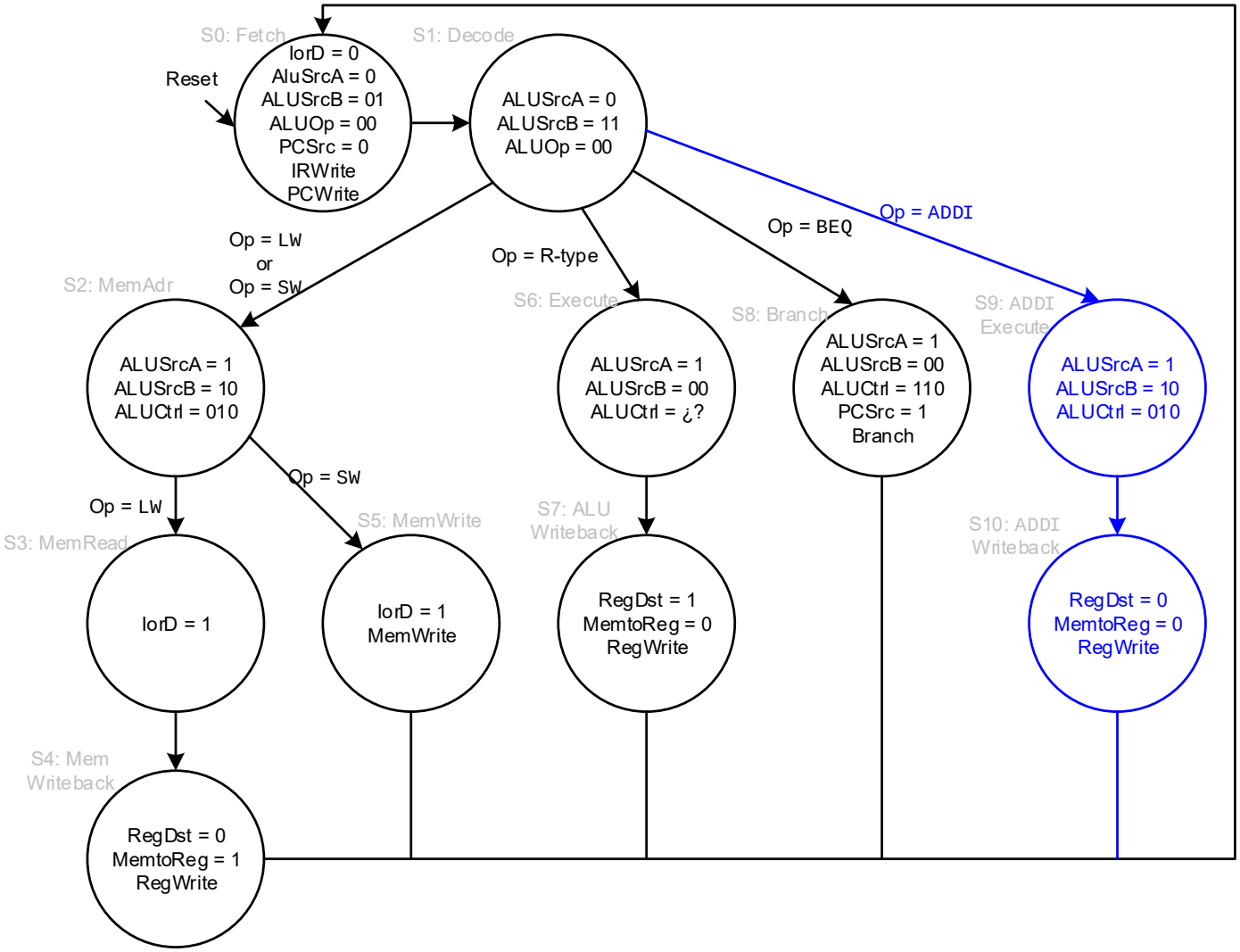
- Resumen arquitectura MIPS uniciclo
- Ruta de datos multiciclo
- Control multiciclo
- **Añadir más instrucciones**

Añadimos addi

- La ruta de datos no necesita cambios (se puede sumar un registro y un dato inmediato como en lw, sw). Sólo hay cambios en el control.



Control: cambios para addi

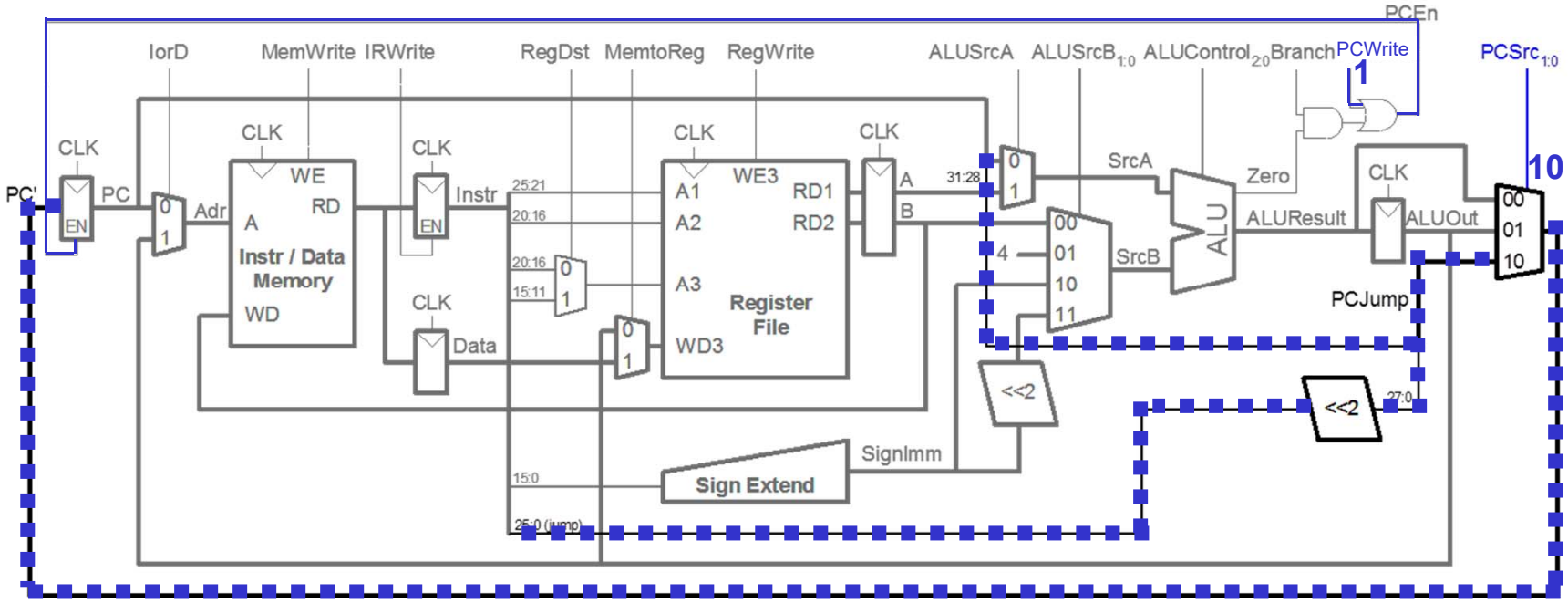


Añadimos j

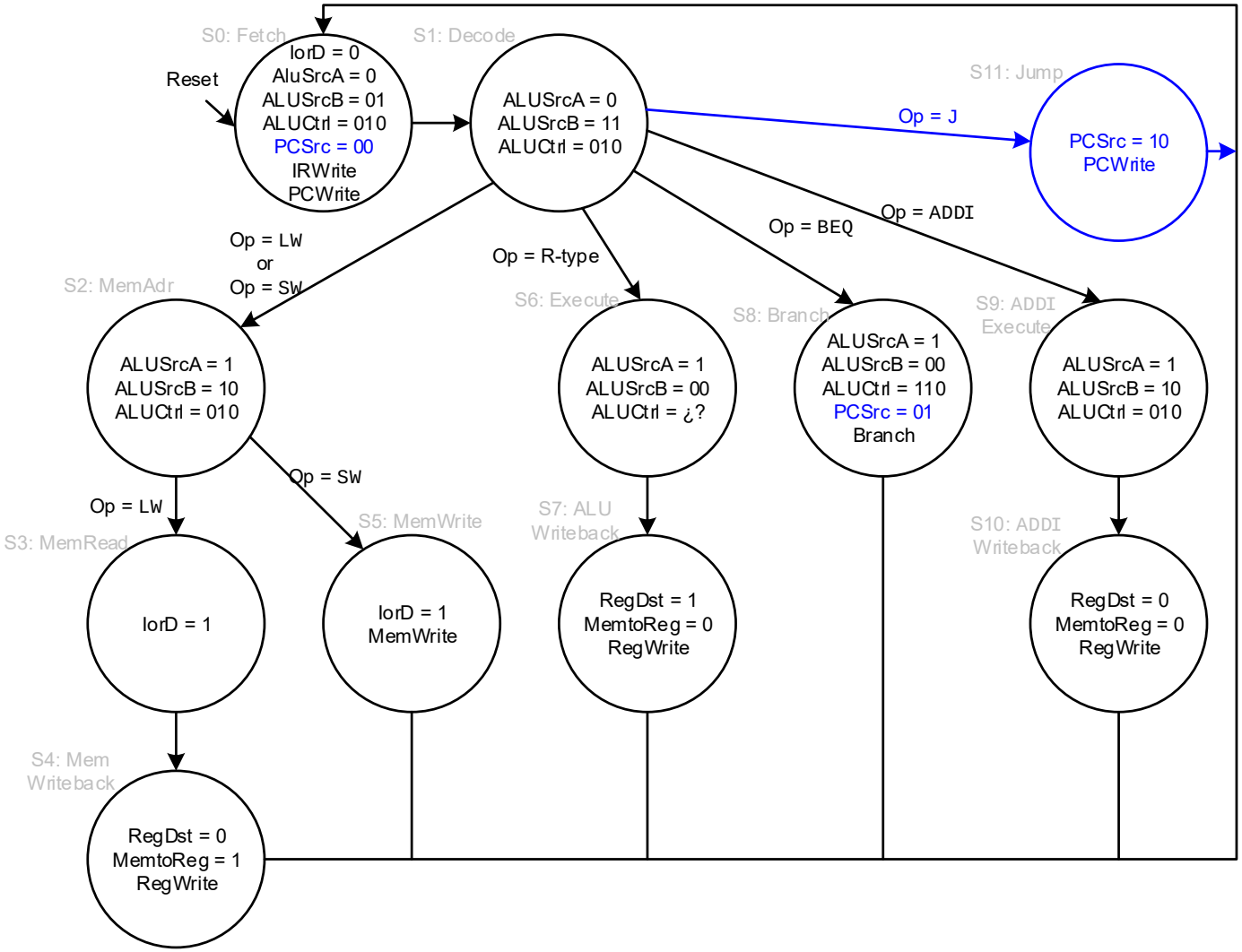
- La ruta de datos sí necesita cambios: hay que poder generar JTA.
  JTA = (PC+4)[31:28] & addr & “00”.

  En **Ciclo-1** PC <= (PC+4)

➢ **Ciclo-3:** Se concatena addr con dos ceros. PCSrc se convierte en una señal de dos bits, porque hay tres señales a elegir.



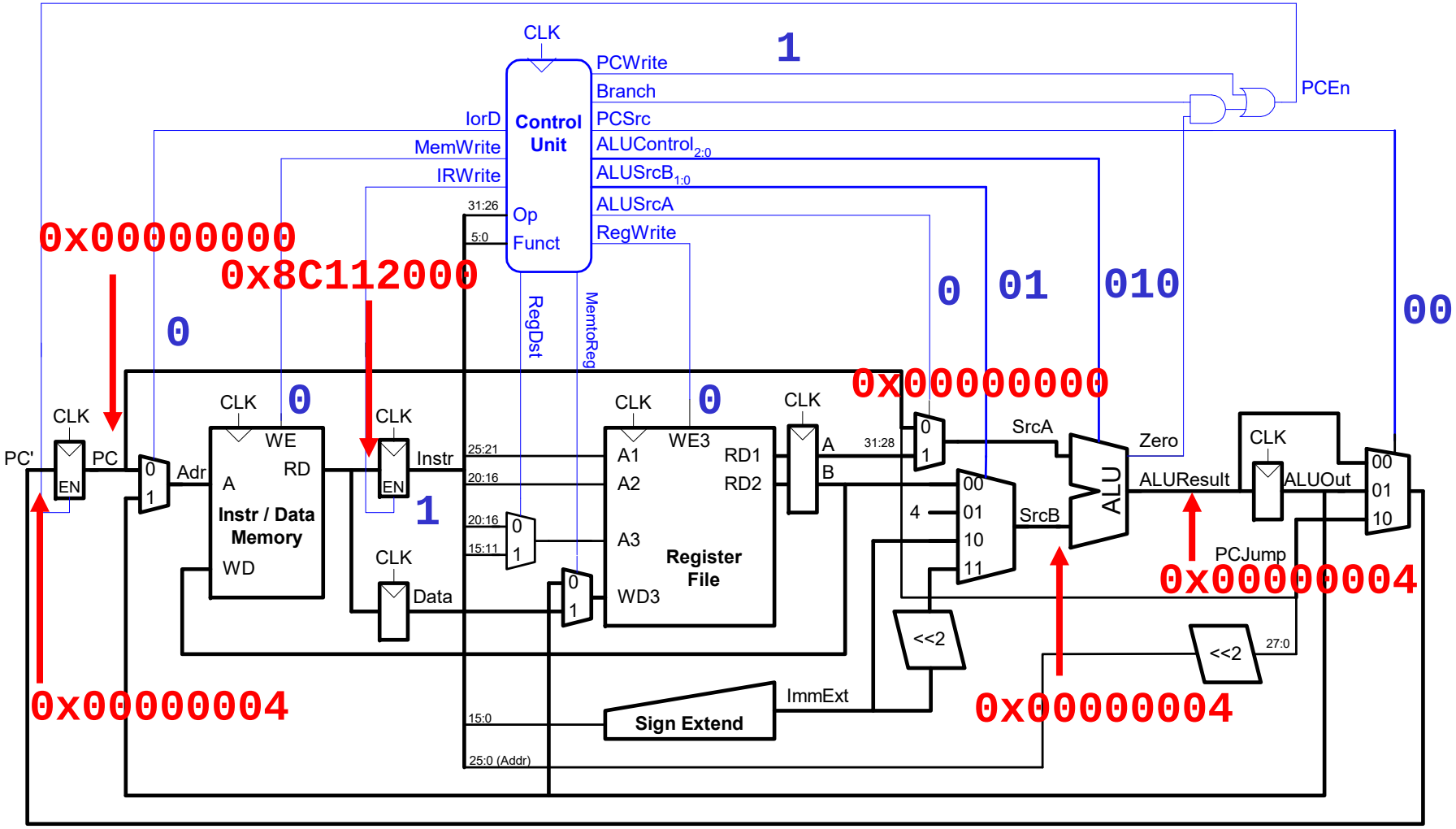
Control: cambios para j



Planificación de instrucciones

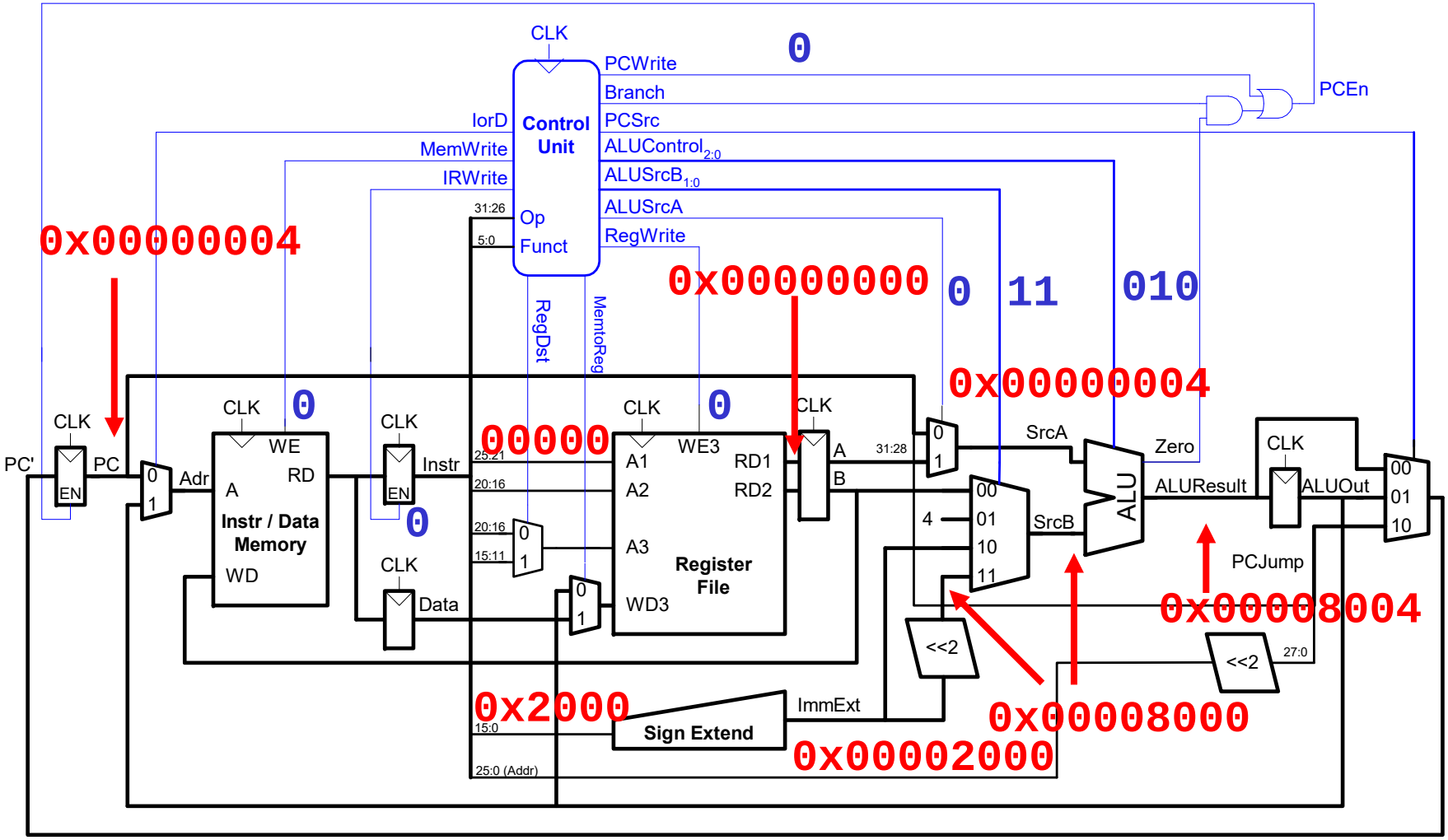
	lw	**sw**	**R-type**	**beq**	**addi**	**j**
Ciclo 1	Captura de instrucción. PC <= PC + 4					
Ciclo 2	Lectura operandos. Cálculo BTA (no se guarda en PC)					
Ciclo 3	Cálculo de dirección de Mem. Datos		Cálculo ALU	Resta en ALU. Salta si Z=1.	Cálculo ALU	Salta a JTA
Ciclo 4	Lectura Mem. Datos	Escritura Mem. Datos	Escritura Reg.		Escritura Reg.	
Ciclo 5	Escritura Reg.					

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 1



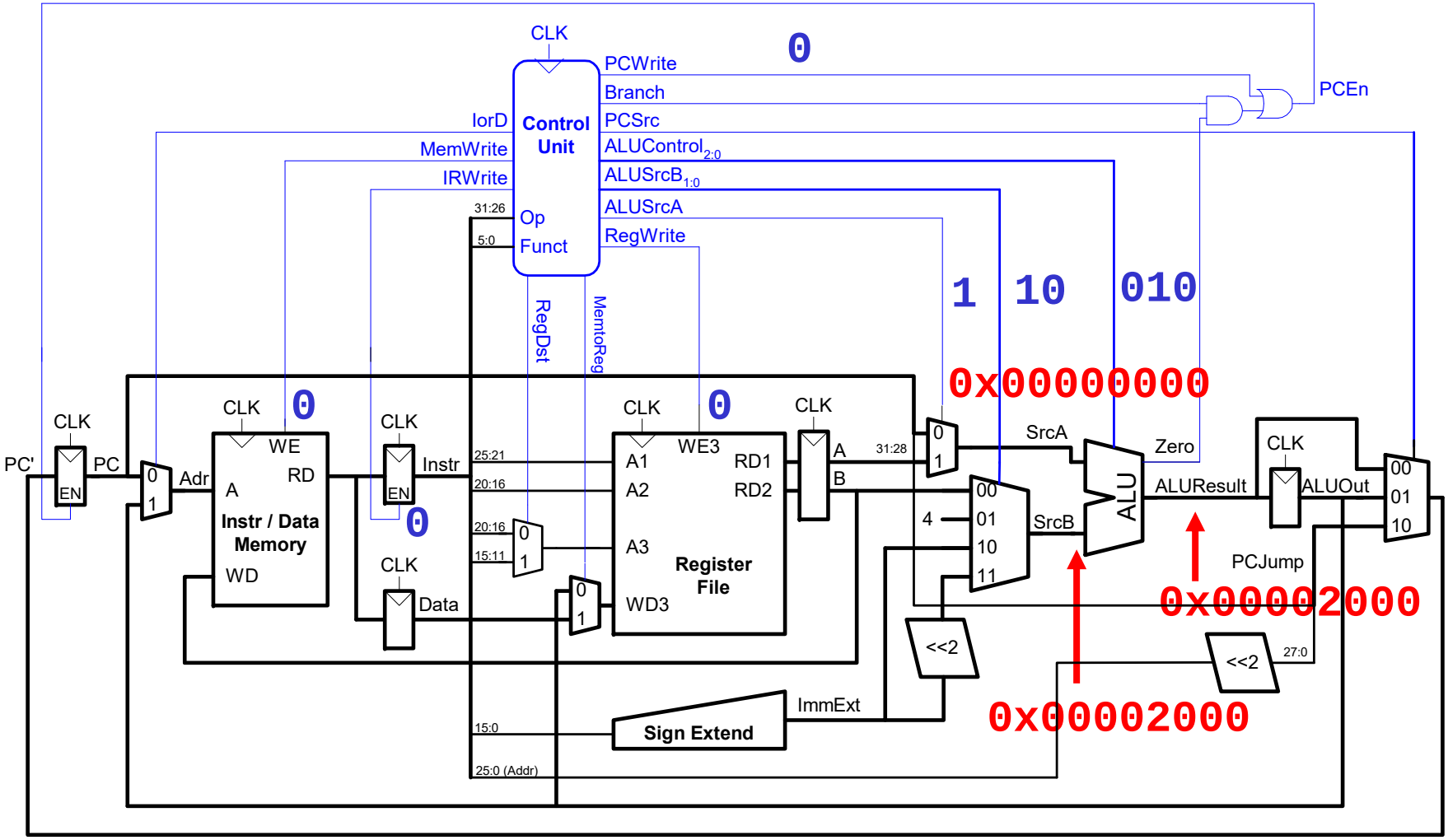
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 2



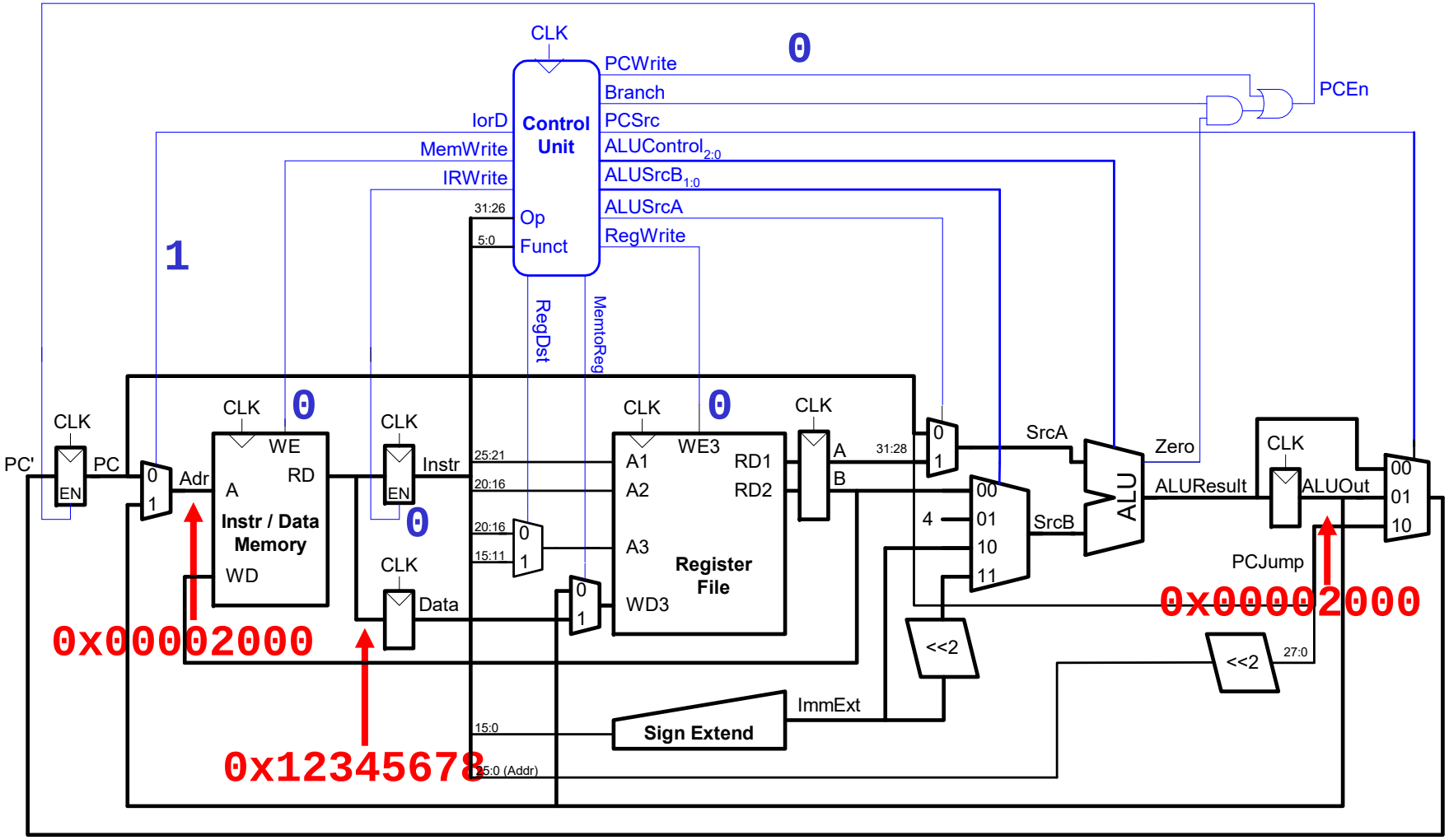
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 3



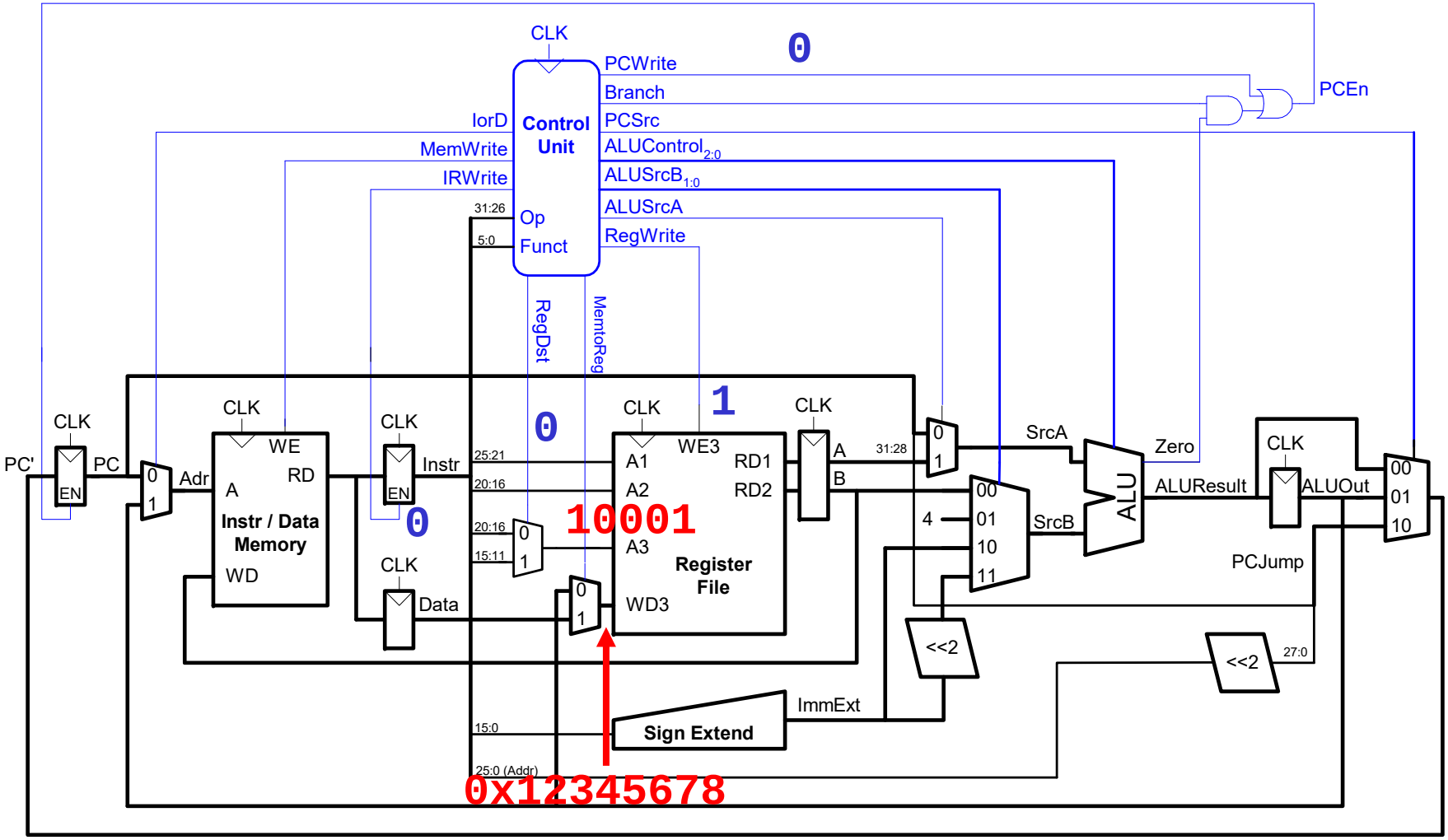
MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 4



MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ruta de datos y de control multiciclo Ejemplo lw. Ciclo 5

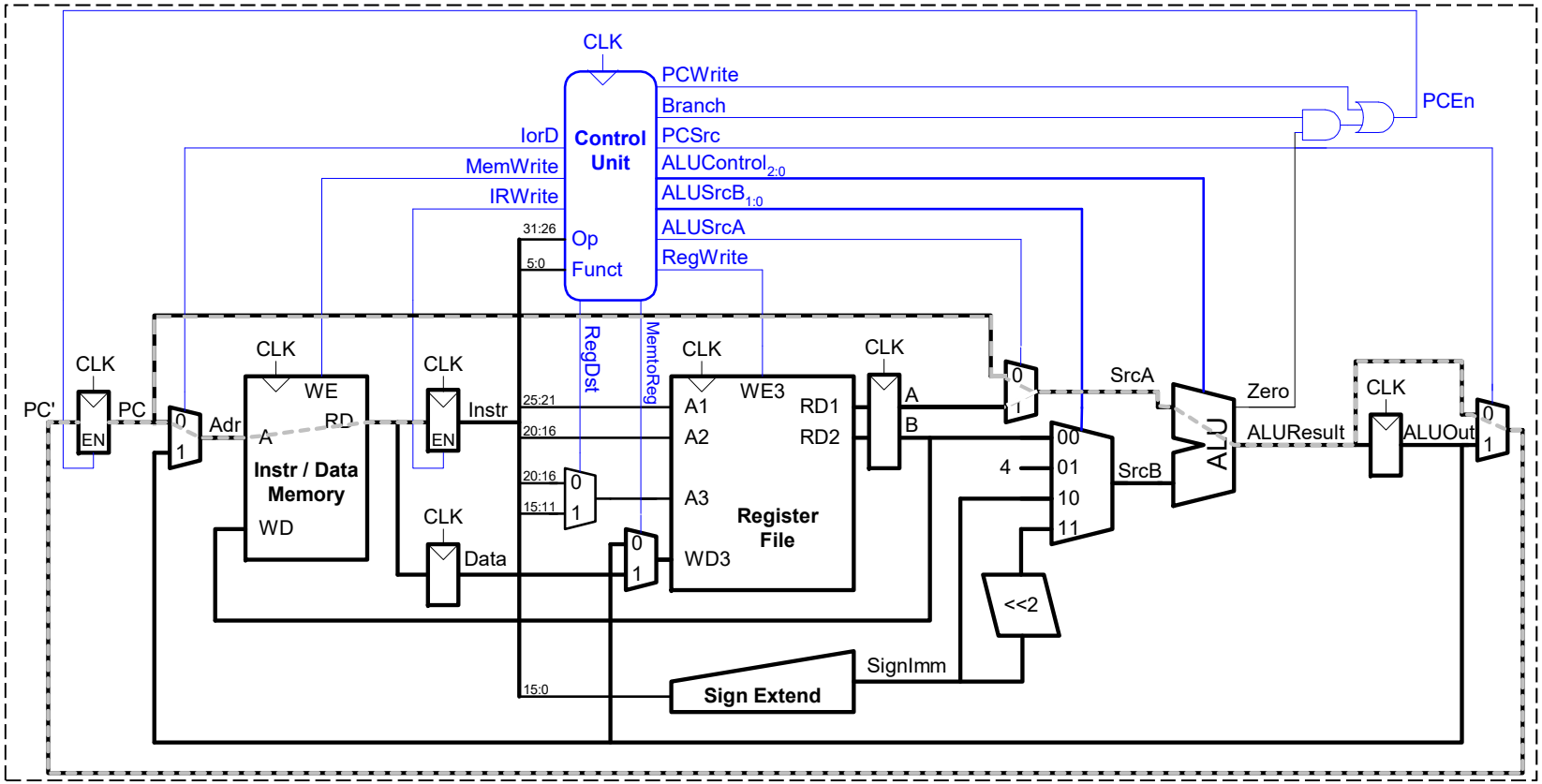


MEM[0x00000000]=0x8C112000 => lw $s1,0x2000($0).
Suponemos que MEM[0x00002000]=0x12345678

Ciclo de reloj en multiciclo

- Lo marca el ciclo de reloj más lento (ciclo 1):

$$T_c = t_{pcq} + t_{mux} + t_{mem} + t_{setup}$$



Ciclo de reloj en multiciclo

Elemento	Parámetro	Retardo (ps)
Register clock-to-Q	t_{pcq_PC}	30
Register setup	t_{setup}	20
Multiplexer	t_{mux}	25
ALU	t_{ALU}	200
Memory read	t_{mem}	250
Register file read	t_{RFread}	150
Register file setup	$t_{RFsetup}$	20

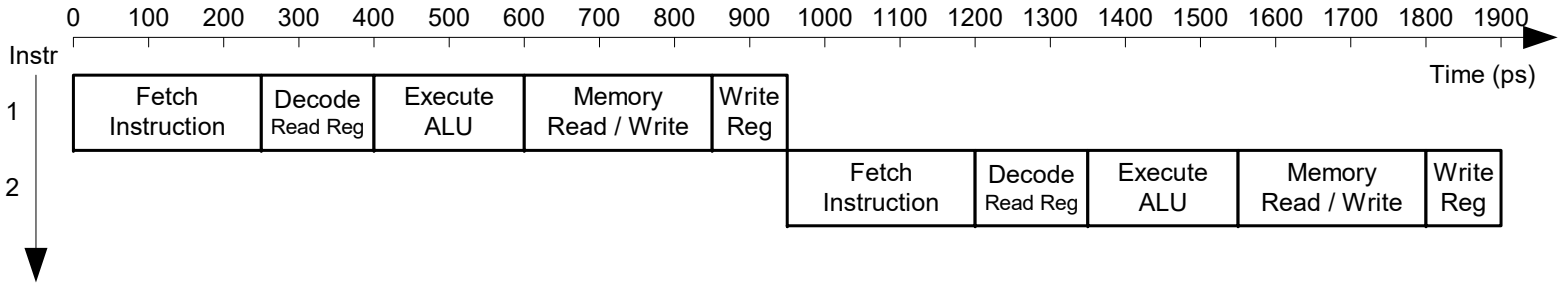
$$T_c = t_{pcq\_PC} + t_{mux} + t_{mem} + t_{setup} =$$
$$= [30 + 25 + 250 + 20] \text{ ps} = 325 \text{ ps}$$

Análisis: uniciclo vs multiciclo

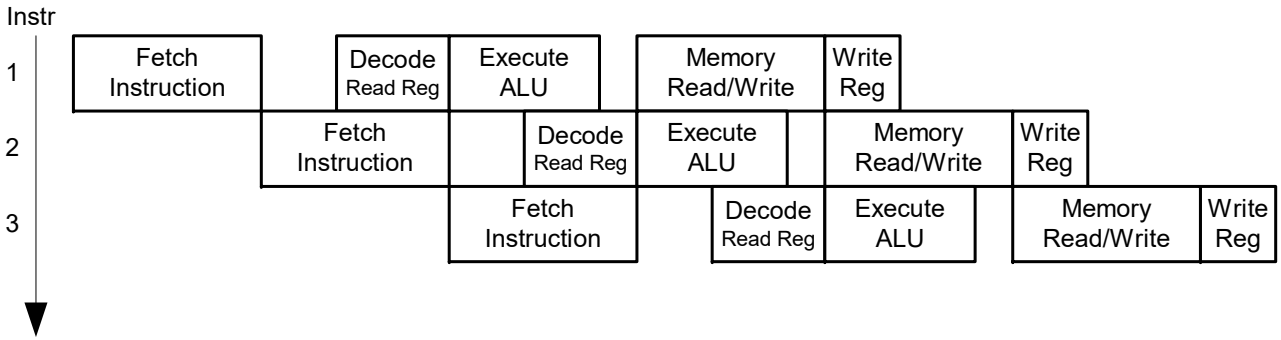
- En uniciclo, todas las instrucciones tardan lo mismo que la más lenta (lw), es decir, 925 ps.
- En multiciclo, todos los ciclos de reloj van al ritmo del ciclo más lento (ciclo 1), es decir, 325 ps.
- En multiciclo, cada instrucción tarda un número distinto de ciclos, entre 3 y 5. El tiempo de ejecución está entre 3·325 y 5·325, es decir, entre 975 y 1625 ps.
- Multiciclo sale más lento que uniciclo (en este ejemplo): la razón es que los ciclos de reloj no se han dividido uniformemente (lo uniforme habría sido 925 ps/5 = 185 ps).
- Sin embargo, multiciclo es la base de segmentación (*pipeline*), en donde cada ciclo "corto" de reloj se empieza una nueva instrucción, ejecutándose varias en paralelo.

Uniciclo vs segmentado

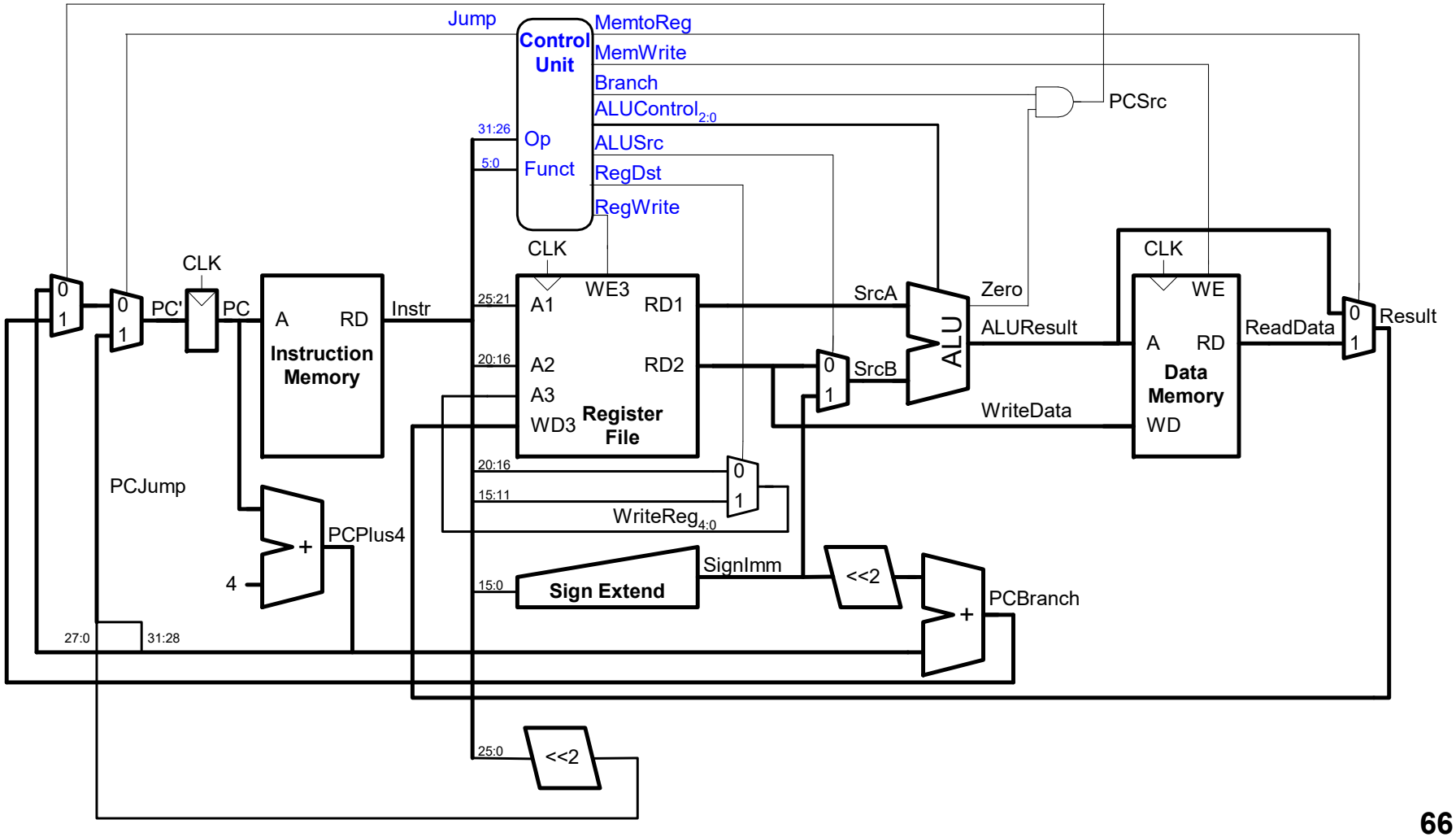
Single-Cycle



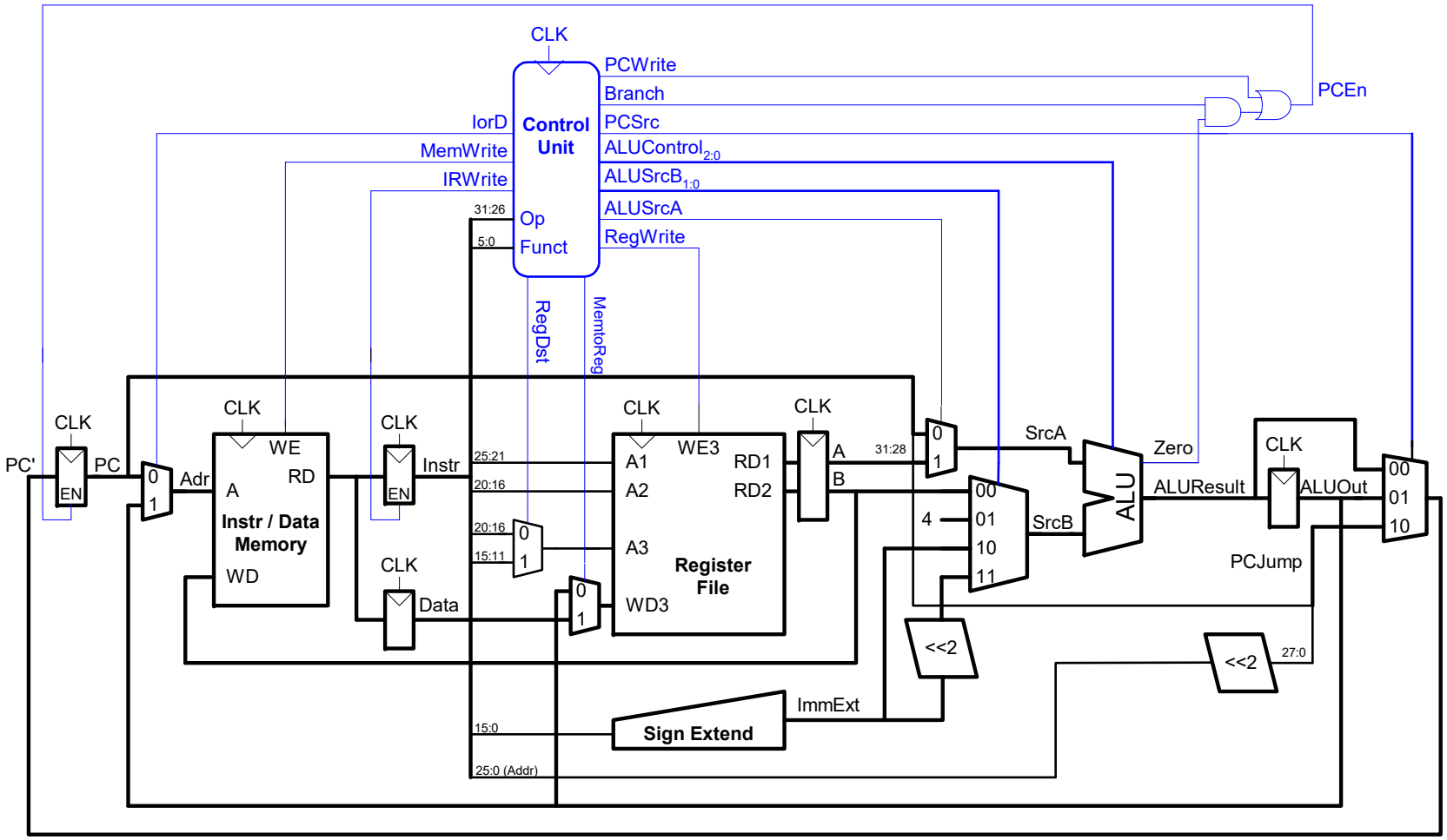
Pipelined



Resumen: MIPS uniciclo



Resumen: MIPS multiciclo



Unidad 5. El Procesador III: Diseño y control de la ruta de datos: Arquitectura multiciclo

Escuela Politécnica Superior - UAM

Problemas U5

T

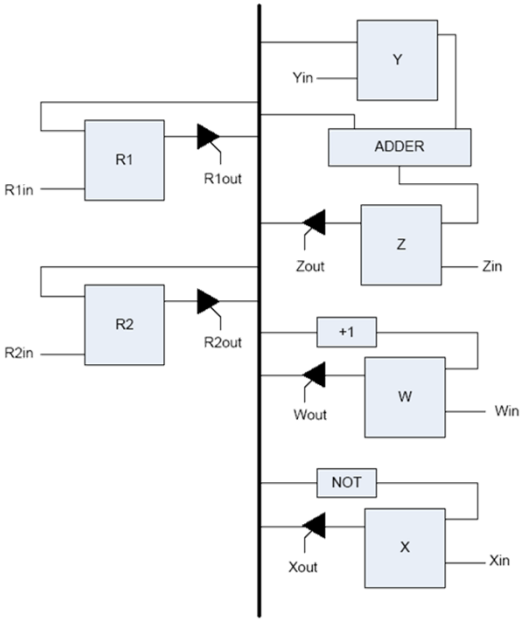
5.1. Utilizando la arquitectura facilitada en la figura, se pide indicar ciclo a ciclo la ruta de datos así como en cada caso, las microinstrucciones necesarias para ejecutar la instrucción:

a) R1 ← R1 - R2 + 1

b) R2 ← 2*R2 + 2*R1 + 2

Describir ciclo a ciclo cada operación necesaria señalando el movimiento de datos entre registros (descripción RTL, *Register Transfer Level*).

Nota: para cada uno de los registros del sistema las variables: Yin, Zin, Win, Xin, R1in y R2in, habilitan en cada caso al registro para la escritura. Por su parte, las señales XXout habilitan el triestado correspondiente al ponerse a '1'.



Problemas U5

T

5.3. Sea un sistema procesador basado en MIPS con arquitectura multiciclo igual que el estudiado en clase. Se recuerda que las direcciones para el acceso a memoria son de 32 bits. Se quiere ejecutar el código que se adjunta, en donde se señala el inicio de la memoria de código y el de la memoria de datos.

a) Con la ayuda de la tabla de códigos para MIPS, encuentre el código máquina para la instrucción del programa anterior "**bne $s1, $0, lab"**

b) Señale el código ensamblador que corresponde al código dado en hexadecimal. Teniendo en cuenta los operandos, señale brevemente la función que esta instrucción realiza.

c) Para cada una de las líneas de control señaladas, señale los valores que en cada ciclo de reloj correspondan, para ejecutar completamente las dos primeras instrucciones del programa dado lw $t2, X($0) y sllv $t3, $t2, $t2

RAM CÓDIGO / DATOS	
.text 0	.data 0x2000
lab: lw $t2, X($0)	.space 8
sllv $t3, $t2, $t2	X: 0x0002
addi $t1, $t3, 0x200C	Y: 0x00FF
and $t4, $t1, $t2	Z:
xor $s1, $t4, $t4	
bne $s1,$0, lab	
0xAC112010	
fin: j fin	

Señales de Control										
IorD	IRWrite	RegDst	MemWrite	MentoReg	RegWrite	PC_Write	Branch	ALUScrA	$ALUScrB_{(1:0)}$	$PCSrc_{(1:0)}$

Problemas U5

5.20. Sea un sistema procesador basado en MIPS con arquitectura multiciclo igual que el estudiado en la asignatura. Se quiere ejecutar el código que se adjunta.
Se pide completar la siguiente tabla con la evolución a lo largo de los nueve primeros ciclos de reloj de diversas señales y registros, sabiendo que en el ciclo 1 de reloj se está ejecutando el primer ciclo de la primera instrucción del programa.

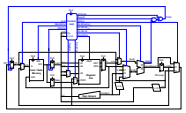
Datos iniciales: $s1 = 0x04; $s2 = 0x08; $s3 = 0x0C

RAM CÓDIGO/DATOS	

.text 0x0000
lw $s2, 0x2000($s1)
beq $s2, $s3, fin
sw $s2, 0x2000($0)
fin: j fin
	.data 0x2000
	X: 0x0A
	Y: 0x0C

Registros						
$pc:	$s1:	$s2:	$s3:	A:	B:	ALUOut:

Señales de Control		
RegWrite	ALUScrA	$ALUScrB_{(1:0)}$



Problemas U5

5.22. En el esquema adjunto se muestra la ruta de datos para la arquitectura multiciclo del procesador ARC (A Risc Computer), incluyendo la ruta para el acceso a la memoria única de instrucciones y datos. En ARC el tamaño para las instrucciones y datos es de 32 bits. En el esquema se identifican, de izquierda a derecha de la imagen, el registro de instrucciones ir, el banco de registros GPR, los registros temporales rtA y rtB a la entrada de la ALU y el registro acumulador rtAcc a la salida de esta esta unidad. Las señales de control son las que aparecen enmarcadas en el esquema.

La siguiente instrucción de ARC hace la suma de dos operandos fuente y escribe el resultado en un registro: **addcc %r10, - 100, %r15**, (en MIPS esta instrucción es equivalente a: **addi $15, $10, - 100**). Con la información de la que se dispone, se pide:

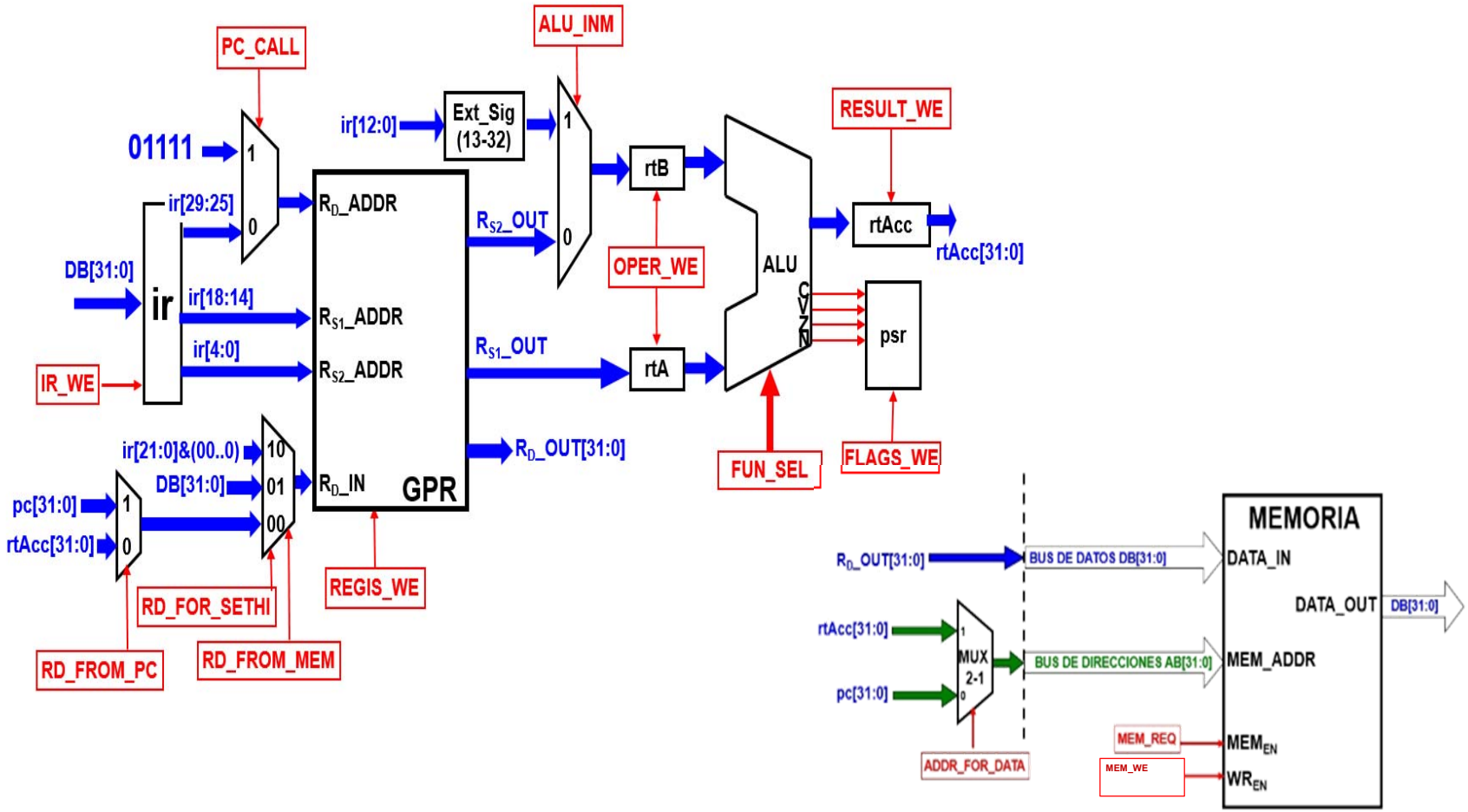
La instrucción máquina equivalente. Seleccione agrupando en bits los campos identificados y marque con ‘x’ los bits para los que no se dispone de información y que corresponden con el campo o con los campos destinados al código de operación.

La ruta de datos multiciclo de ARC utiliza 4 ciclos para ejecutar esta instrucción, complete la tabla adjunta indicando el valor para las señales de control señaladas. Las señales terminadas en _WE, habilitan la escritura en el circuito síncrono correspondiente. El resto son señales de control cuyo valor debe saber identificar para ejecutar la instrucción demandada. Complete la tabla con los valores ‘0’, ‘1’ ó ‘X’ según corresponda a cada ciclo. **Nota**: Señalar que en el esquema se ha omitido la ruta de actualización para el PC y por tanto no hay que actualizarlo.

Señales	MEM_WE	ADDR_FOR_DATA	IR_WE	PC_CALL	ALU_INM	OPER_WE	RESULT_WE
	RD_FROM_PC	RD_FOR_SETHI	RD_FROM_MEM	REGIS_WE			

Problemas U5

5.22.



Problemas U5

5.5. Se quiere añadir en la ruta de datos multiciclo de MIPS estudiada en clase, los elementos de hardware necesarios para ejecutar una instrucción que permita intercambiar el contenido de dos registros de propósito general cualesquiera. La sintaxis **e**nsamblador sería: **swap $0, $X, $Y** y también **swap $X, $Y**; donde $X y $Y representan los registros a intercambiar. Esta nueva instrucción de formato R-Type e identificada por el código funct = “110000”, debe ejecutarse en 4 ciclos incluyendo la etapa de captura. Para facilitar el diseño se adjunta la descripción funcional en base al movimiento de datos entre registros (RTL) para la nueva instrucción.

①	**[Instr] <= MEM [pc] ; [pc] <= [pc] + 4**
②	**[A] <= [rs]; [B] <= [rt]**
③	**[rt] <= [A]**
④	**[rs] <= [B]**

Se pide:

a) Escribir el código máquina para esta instrucción tomando para X e Y dos valores cualesquiera.

b) Señalar de forma esquemática los cambios necesarios en la ruta de datos que se proponen para ejecutar la instrucción (nuevos dispositivos y/o nuevos caminos).

c) Señalar, indicando cómo y cuándo se utilizan, las nuevas variables definidas para el control de la ruta de datos una vez incorporada la nueva instrucción.

Problemas U5

5.17. Sea un sistema procesador basado en MIPS con arquitectura multiciclo igual que el estudiado en la asignatura. El sistema se encuentra en el periodo T ejecutando el primer ciclo (captura) de una determinada instrucción, instrucción actual, del código que se adjunta. En este código, para el paso de parámetros entre rutinas se utilizan los registros específicos de MIPS ($ax y $vx).
Con la información facilitada de la tabla de control, debe poder identificar la instrucción actual y las dos instrucciones ejecutadas en los ciclos precedentes.

a. Indique el valor en hexadecimal del registro **pc** en los ciclos T-1, T y T+1.

b. En la tabla adjunta, complete con los valores adecuados las señales de control necesarias para ejecutar completamente la instrucción actual (utilice las columnas que considere necesarias).

c. Complete el valor en hexadecimal del registro señalado **%ra** y de la posición de memoria dada por la etiqueta **R**, una vez ejecutada completamente la instrucción actual.

d. En la ruta de datos de MIPS, los registros intermedios A y B, se utilizan para registrar los operandos de la ALU en el caso que se necesiten y puede cambiar cada ciclo su valor. Sabiendo que la codificación de la instrucción incluida en el código **jr $ra, al final de la rutina mulx4**, es **0x03E00008**, se pide señalar el valor en hexadecimal de los registros A y B después de completar los dos últimos ciclos necesarios para ejecutar la citada instrucción, es decir un poco después del flanco de reloj correspondiente a los citados ciclos.

Problemas U5

5.17.

Ciclo	T-7	T-6	T-5	T-4	T-3	T-2	T-1	T
PCWrite	1	0	0	0	1	0	1	1
IorD	0	X	X	1	0	X	X	0
IRWrite	1	0	0	0	1	0	0	1
MemWrite	0	0	0	1	0	0	0	0
RegWrite	0	0	0	0	0	0	0	0
MemtoReg	X	X	X	X	X	X	X	X
RegDst	X	X	X	X	X	X	X	X
ALUSrcA	0	0	1	X	0	0	1	0
ALUSrcB$_{(1:0)}$	01	11	10	XX	01	11	XX	01

RAM CÓDIGO / DATOS

```
.text 0x0000
main:   lw $s1,
   Num($0)
        add $a0, $s1,
   $0
        jal sub1
        lw $s1, R($0)
fin:    j fin
----------------------------
----------------------------
.data 0x2000
Num:    0x00000064
R:      0x00000000
```

```
.text 0x0100
sub1:   addi $sp, $sp, -4
        sw $ra, 0($sp)
        addi $t1, $0, 100
        beq $a0, $t1, etiq
        jal mulx8
        j ret
etiq:   jal mulx4
ret:    lw $ra, 0($sp)
        addi $sp, $sp, 4
        sw $v0, R($0)
        jr $ra

.text 0x0200
mulx4:      sll $v0, $a0, 2
            jr $ra

.text 0x0300
mulx8:      sll $v0, $a0, 3
            jr $ra
```

Problemas U5

5.12. Sea un sistema procesador basado en MIPS con arquitectura multiciclo igual que el estudiado en clase. Se quiere ejecutar el código que se adjunta:
Se pide completar la tabla facilitada con los valores de las señales y de los registros indicados, desde el comienzo (primer ciclo) de la ejecución de la instrucción etiquetada con “salto”, hasta que se ejecute completamente el programa. Señale el contenido final de las posiciones de memoria indicadas (A, B y C).

RAM CÓDIGO/DATOS	

.text 0x0000
lw $s1, 5($s2)
and $s2, $s1, $s3
beq $s1,$s2,salto
sw $s2, -4($s3)
j fin
.text 0x001C
salto: jal fin
or $s3,$s1,$s2
fin: and $s2,$s1,$s3
sw $s3,B($s2)
	.data 0x2200
	A: 0x00000002
	B: 0x00002200
	C: 0X0000EA17

Datos iniciales: $s1 = 0xA4; $s2 = 0x02; $s3 = 0x4C y $ra = 0x00

Registros					**Memoria**		
$pc:	$s1:	$s2:	$s3:	$ra:	A:	B:	C:

Señales de Control										
IorD	IRWrite	RegDst	MemWrite	MentoReg	RegWrite	PC_Write	Branch	ALUScrA	$ALUScrB_{(1:0)}$	$PCSrc_{(1:0)}$

Resumen: MIPS multiciclo

T