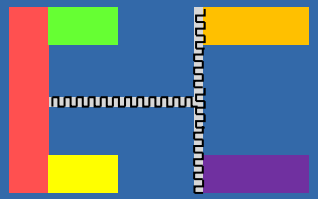
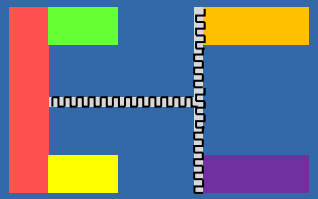


TEMA 4.
UNIDAD CENTRAL DE PROCESAMIENTO

dtic



EJERCICIOS UCP MONOCICLO

EJERCICIO 1

- Calcular el tiempo de ciclo suponiendo retardos despreciables para todos los elementos de la ruta de datos monociclo excepto para:
  - Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns)

TEMA 4. UNIDAD CENTRAL DE PROCESAMIENTO

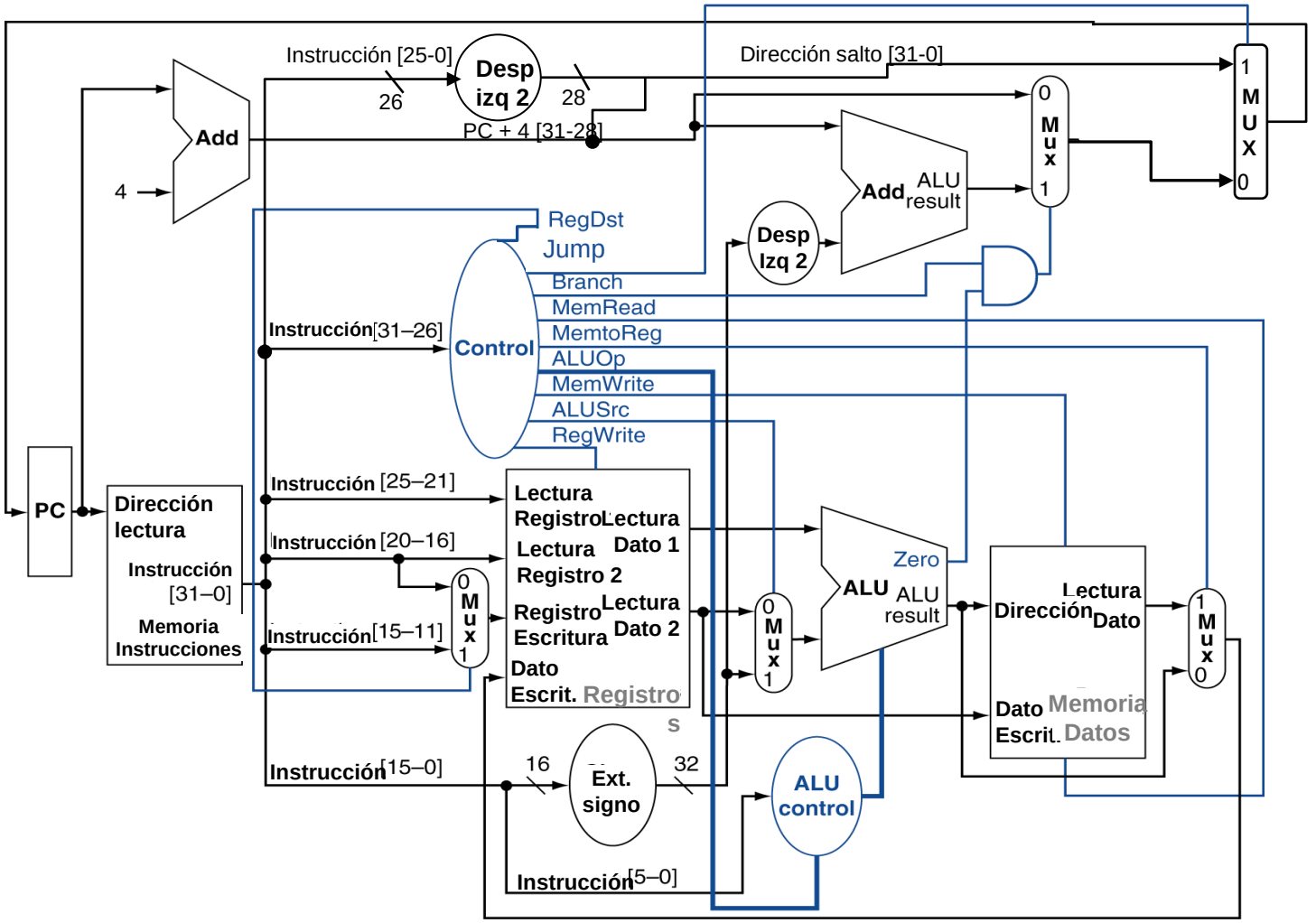
EJERCICIO 1

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.

  - Antes de empezar tenemos que saber que en una arquitectura monociclo, el ciclo de reloj viene determinado por la instrucción de mayor duración. Por lo tanto, tenemos que analizar los distintos casos del repertorio de instrucciones.

EJERCICIO 1

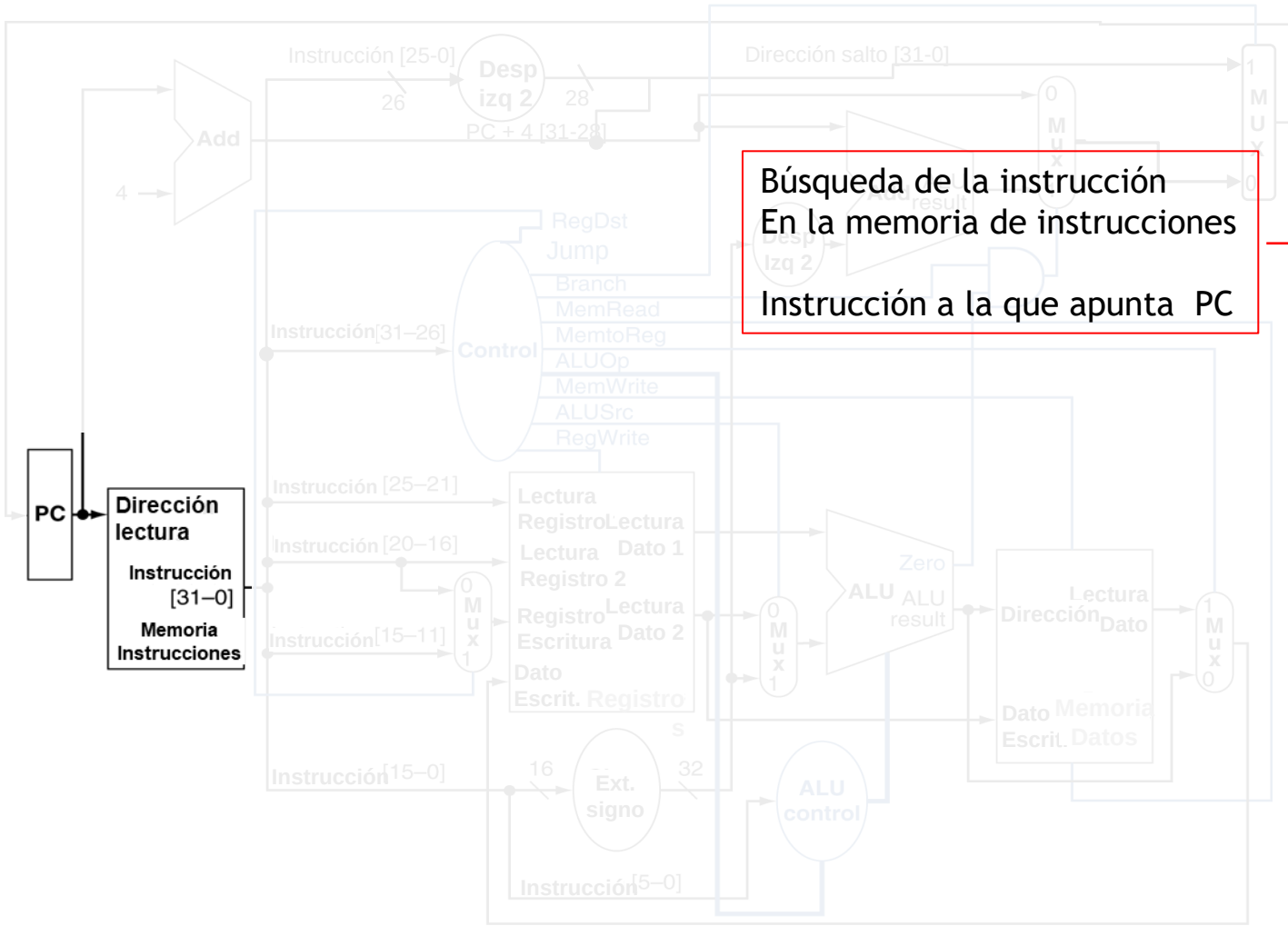
- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.



Instrucción				
R	Lw	Sw	beq	j

EJERCICIO 1

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.



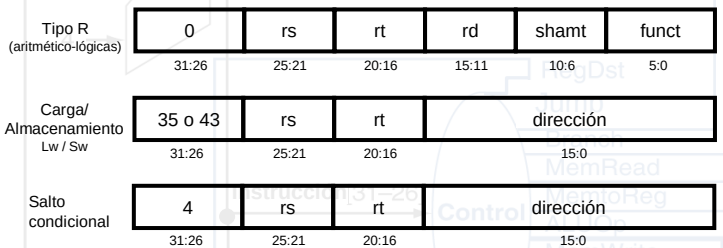
Búsqueda de la instrucción
En la memoria de instrucciones

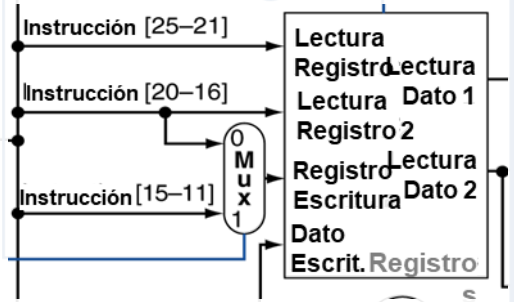
Instrucción a la que apunta PC

Instrucción				
R	Lw	Sw	beq	j
2	2	2	2	2

EJERCICIO 1

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.





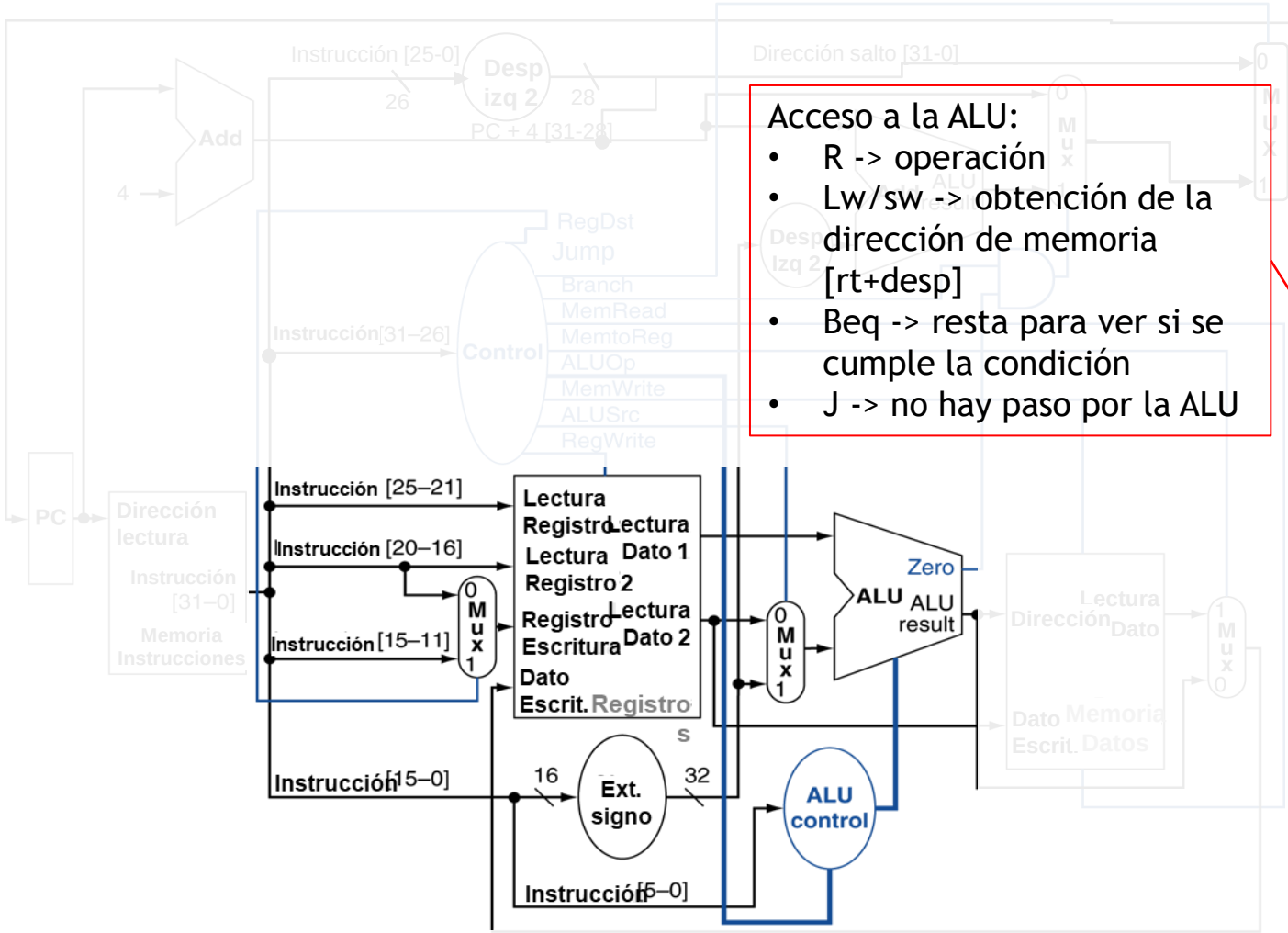
Acceso a registro para leer el Registro Rs

La instrucción "j" no hace esta lectura ya que simplemente salta a una nueva instrucción

Instrucción				
R	Lw	Sw	beq	j
2	2	2	2	2
0.5	0.5	0.5	0.5	

EJERCICIO 1

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.

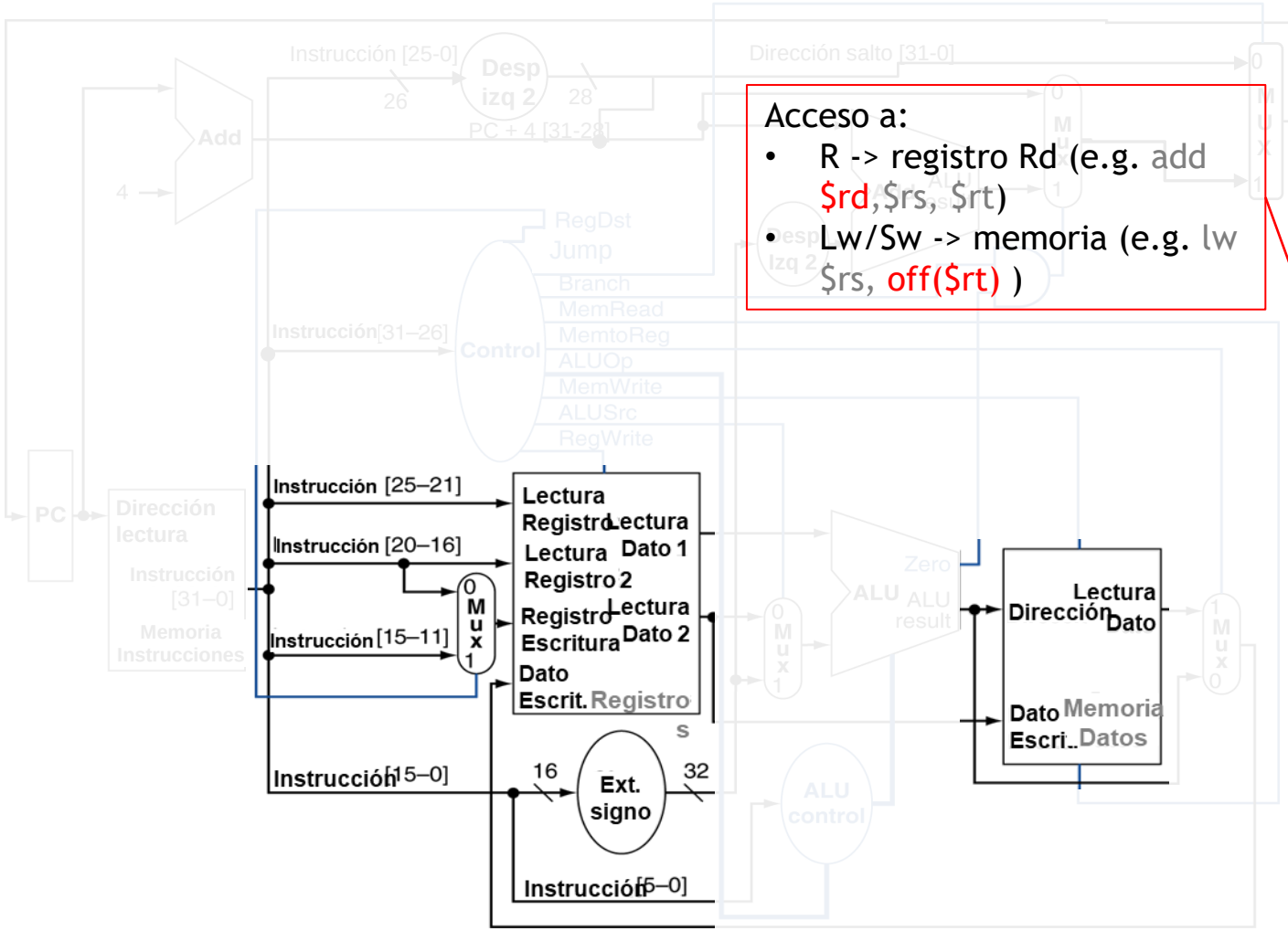


Acceso a la ALU:
- R -> operación
- Lw/sw -> obtención de la dirección de memoria [rt+desp]
- Beq -> resta para ver si se cumple la condición
- J -> no hay paso por la ALU

Instrucción				
R	Lw	Sw	beq	j
2	2	2	2	2
0.5	0.5	0.5	0.5	
1	1	1	1	

EJERCICIO 1

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.

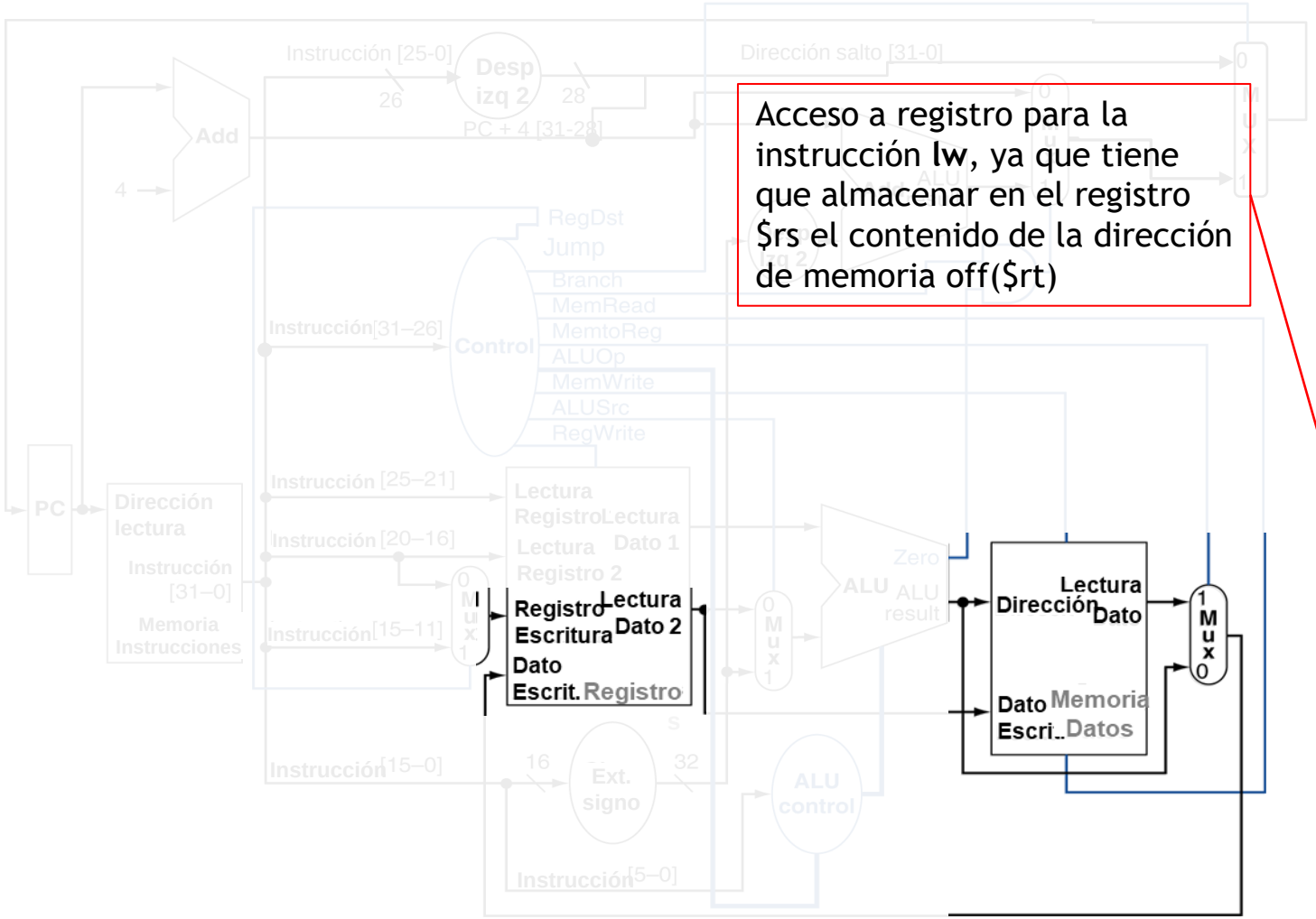


Acceso a:
- R -> registro Rd (e.g. add $rd,$rs, $rt)
- Lw/Sw -> memoria (e.g. lw $rs, off($rt))

Instrucción				
R	Lw	Sw	beq	j
2	2	2	2	2
0.5	0.5	0.5	0.5	
1	1	1	1	
0.5	2	2		

EJERCICIO 1

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.



Acceso a registro para la instrucción **lw**, ya que tiene que almacenar en el registro $rs el contenido de la dirección de memoria off($rt)

Instrucción				
R	Lw	Sw	beq	j
2	2	2	2	2
0.5	0.5	0.5	0.5	
1	1	1	1	
0.5	2	2		
	0.5			
TOTAL				
4	6	5.5	3.5	2

- Se toman como datos: Acceso a memoria (2 ns), ALU y sumadores (1 ns), acceso al banco de registro (0.5 ns), el resto de aspectos se supone que no cuentan.
- ***El ciclo de reloj deberá ser de 6 ns.***

El ciclo de reloj en un monociclo será el tiempo de la instrucción de mayor duración. En este caso es la instrucción lw, con 6ns el ciclo.

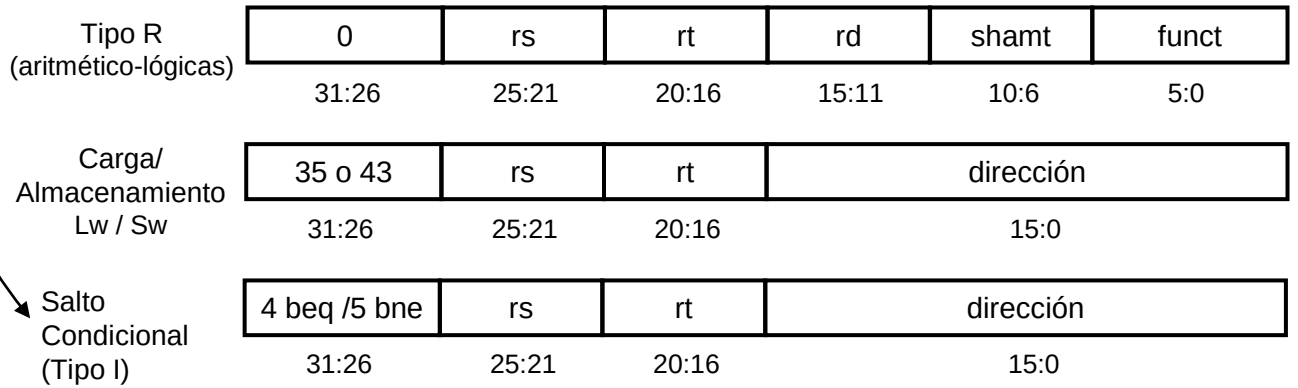
Tipo Instr.	Unidades funcionales utilizadas					Total
Tipo - R	Buscar Instr. (2 ns)	Acceso Reg. (0.5 ns)	ALU (1 ns)	Acceso Reg. (0.5 ns)		4 ns
Lw	Buscar Instr. (2 ns)	Acceso Reg. (0.5 ns)	ALU (1 ns)	Acceso mem. (2 ns)	Acceso Reg. (0.5 ns)	6 ns
Sw	Buscar Instr. (2 ns)	Acceso Reg. (0.5 ns)	ALU (1 ns)	Acceso mem. (2 ns)		5.5 ns
Salto Cond. (beq)	Buscar Instr. (2 ns)	Acceso Reg. (0.5 ns)	ALU (1 ns)			3.5 ns
Salto Incond. (j)	Buscar Instr. (2 ns)					2 ns

Ejercicio 2

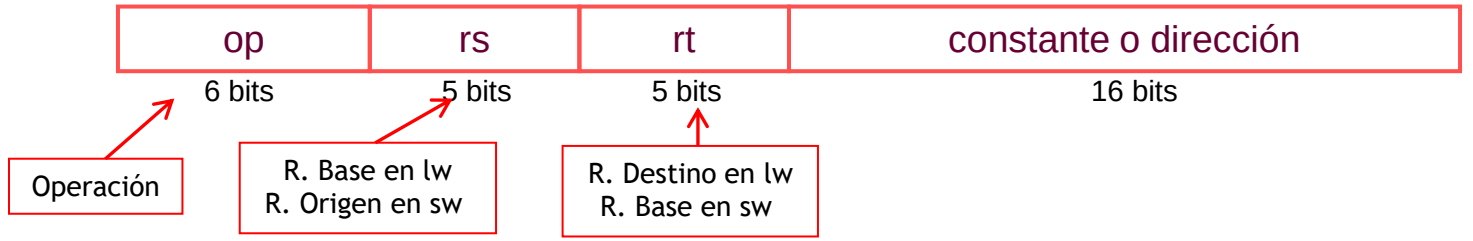
- Dadas las siguientes instrucciones, determinar codificación hexadecimal:
  - lw $1, 40 ($6) #Op = 35
  - bne $1, $2, 100 #Op = 5

Ejercicio 2

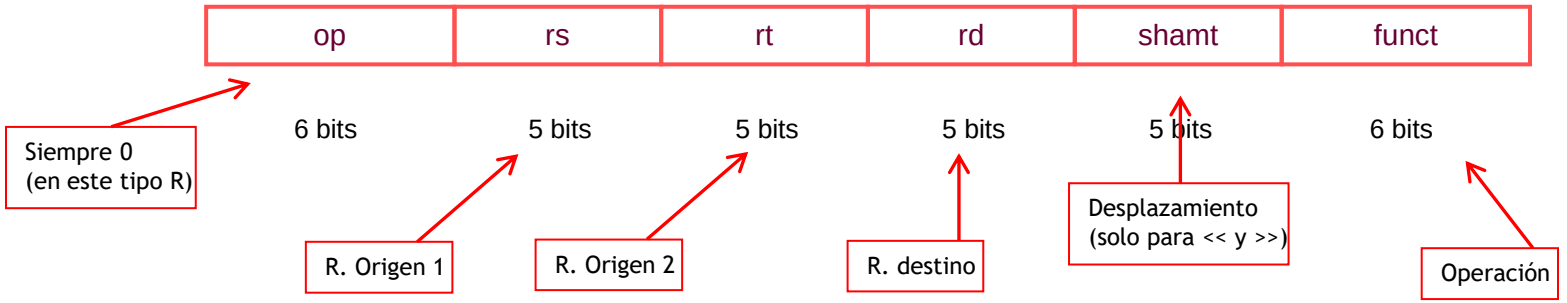
- Dadas las siguientes instrucciones, determinar codificación hexadecimal:
  - lw $1, 40 ($6) #Op = 35
  - bne $1, $2, 100 #Op = 5
- En primer lugar, hay que conocer qué tipo de instrucción es cada una



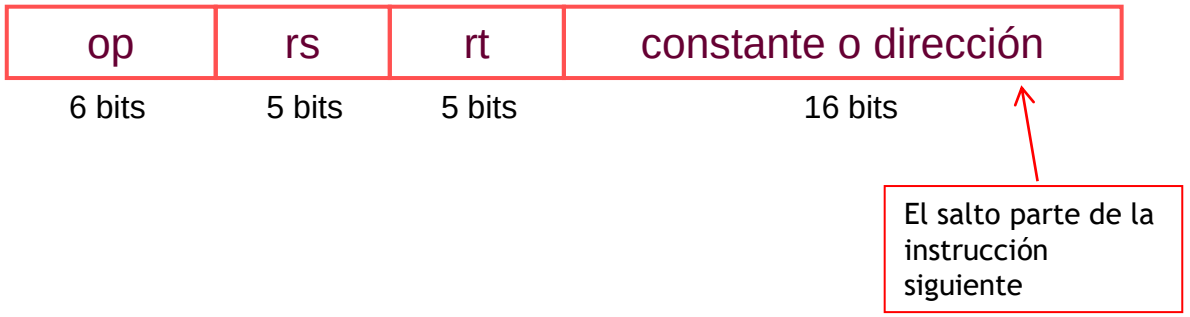
- Formato tipo - I **lw** (carga de memoria), **sw** (almacena en memoria)



- Formato tipo - R **add, …**

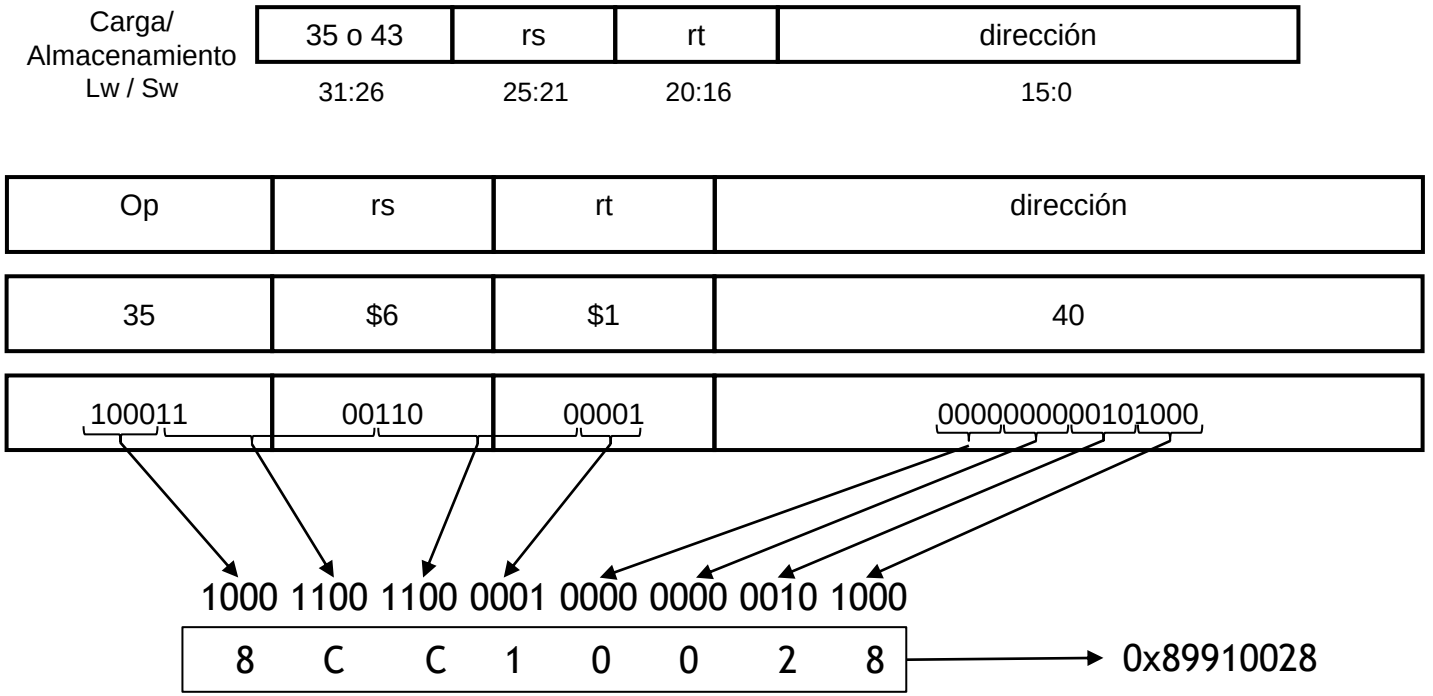


- Formato tipo - I (**beq**)



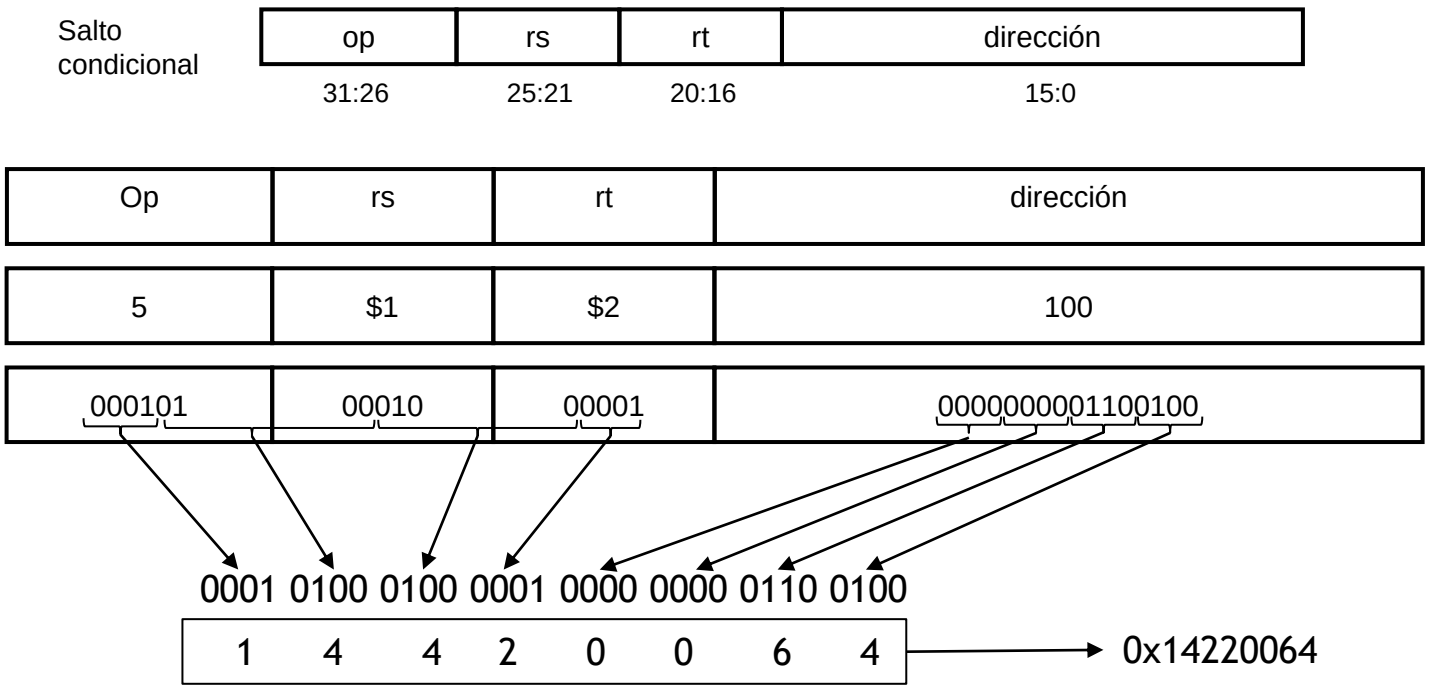
Ejercicio 2

- lw $1, 40 ($6) #Op = 35
  - Lw $rt, off($rs)



Ejercicio 2

- bne $1, $2, 100 #Op = 5
  - Bne $rs, $rt, imm

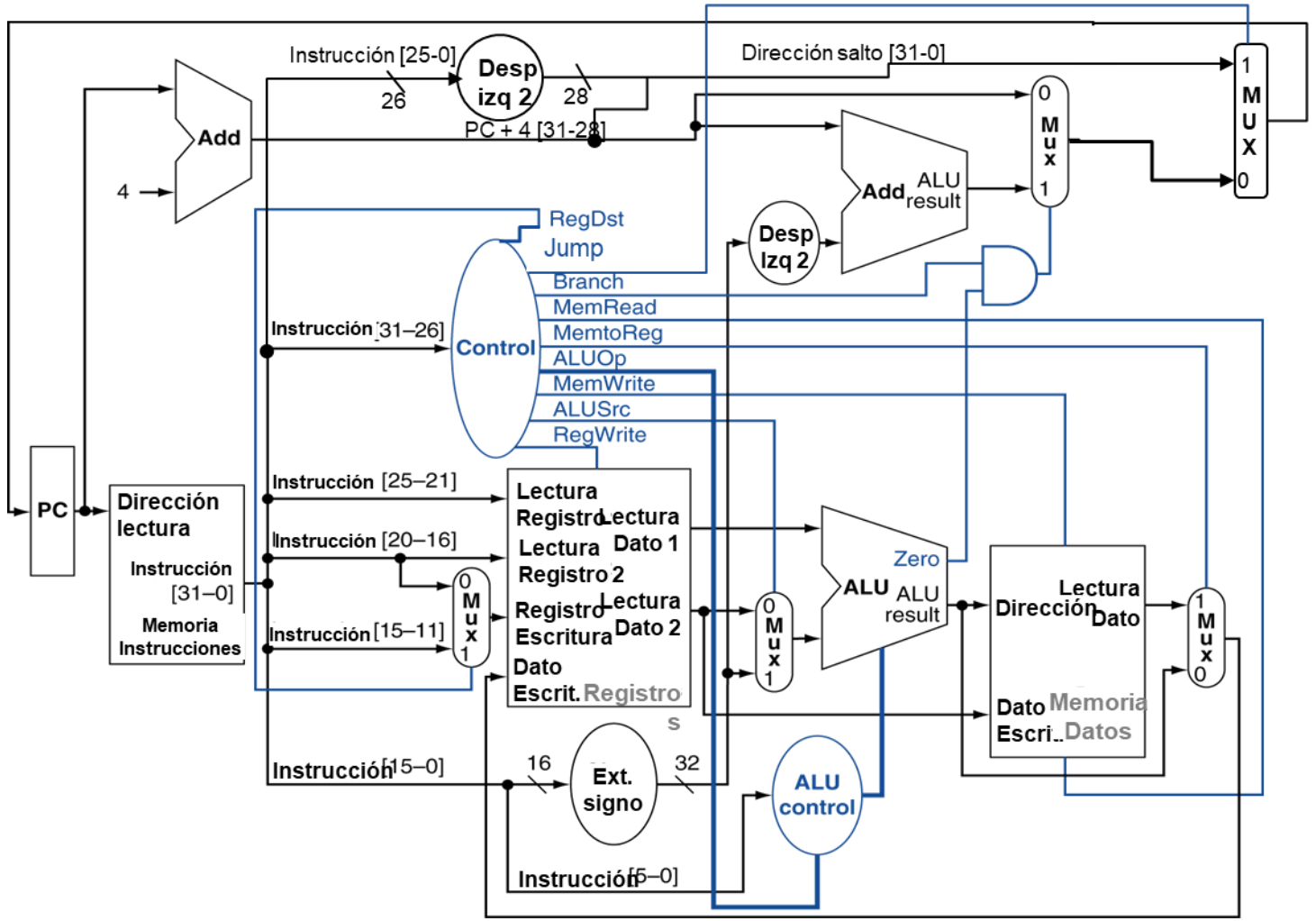


Ejercicio 3

- Dadas las siguientes instrucciones y la figura de pág. 31, determinar los valores de las señales de control de la UC:

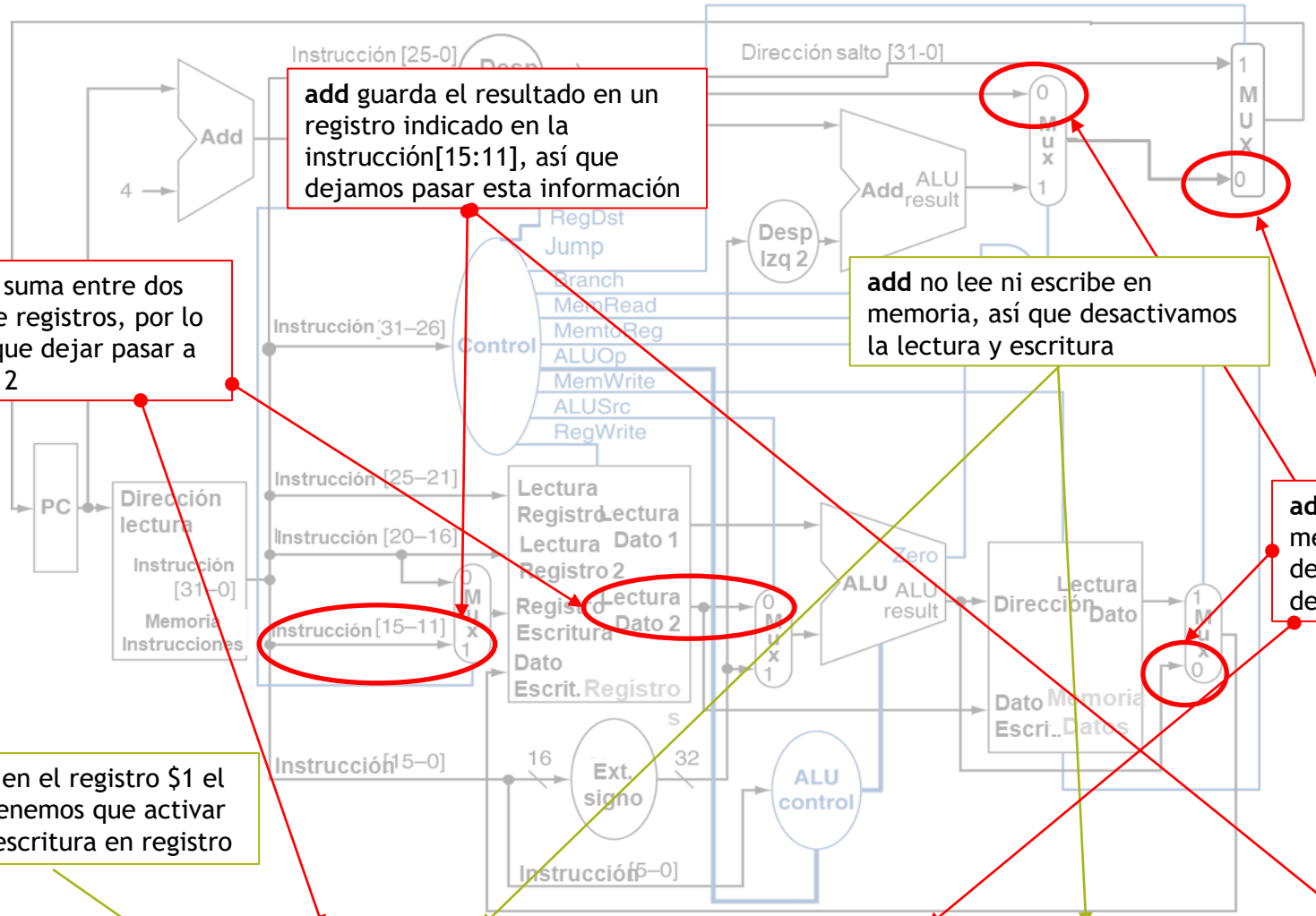
1) add $1, $2, $25
2) addi $1, $2, 10 (Instrucción Tipo I)
3) lw $4, 100($1)
4) sw $1, 35($5)
5) beq $3, $4, 1000

Ejercicio 3



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
add $1, $2, $25	1	0	0	1	0	0	0	0	0	1

Ejercicio 3



add guarda el resultado en un registro indicado en la instrucción[15:11], así que dejamos pasar esta información

add calcula la suma entre dos datos leídos de registros, por lo que tenemos que dejar pasar a la ALU el dato 2

add no lee ni escribe en memoria, así que desactivamos la lectura y escritura

add no utiliza la memoria, así que dejamos pasar el dato de la salida de la ALU

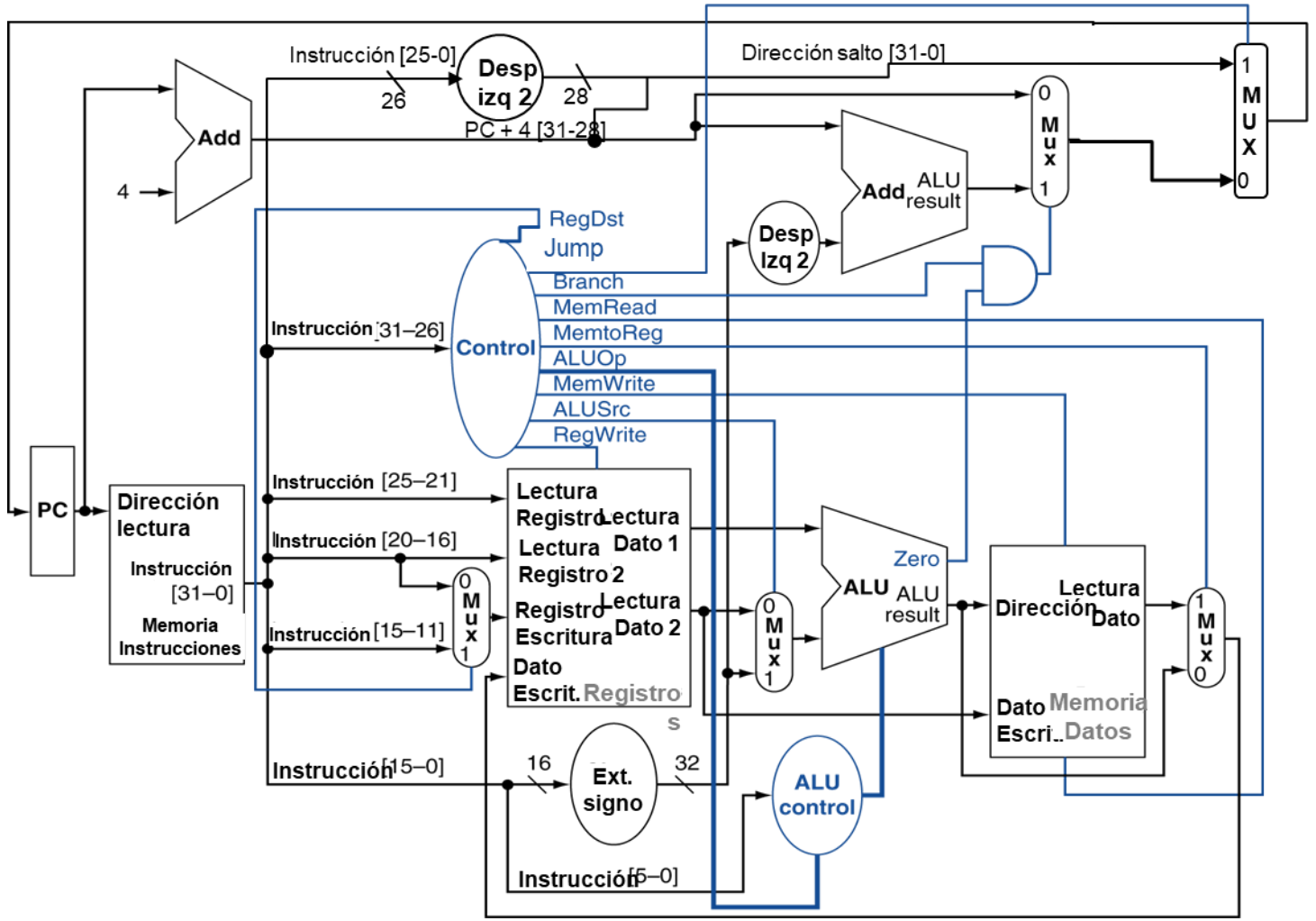
add escribe en el registro $1 el resultado, tenemos que activar la señal de escritura en registro

add no realiza saltos, así que desactivamos los saltos y dejamos pasar PC+4

	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
add $1, $2, $25	1	0	0	1	0	0	0	0	0	1

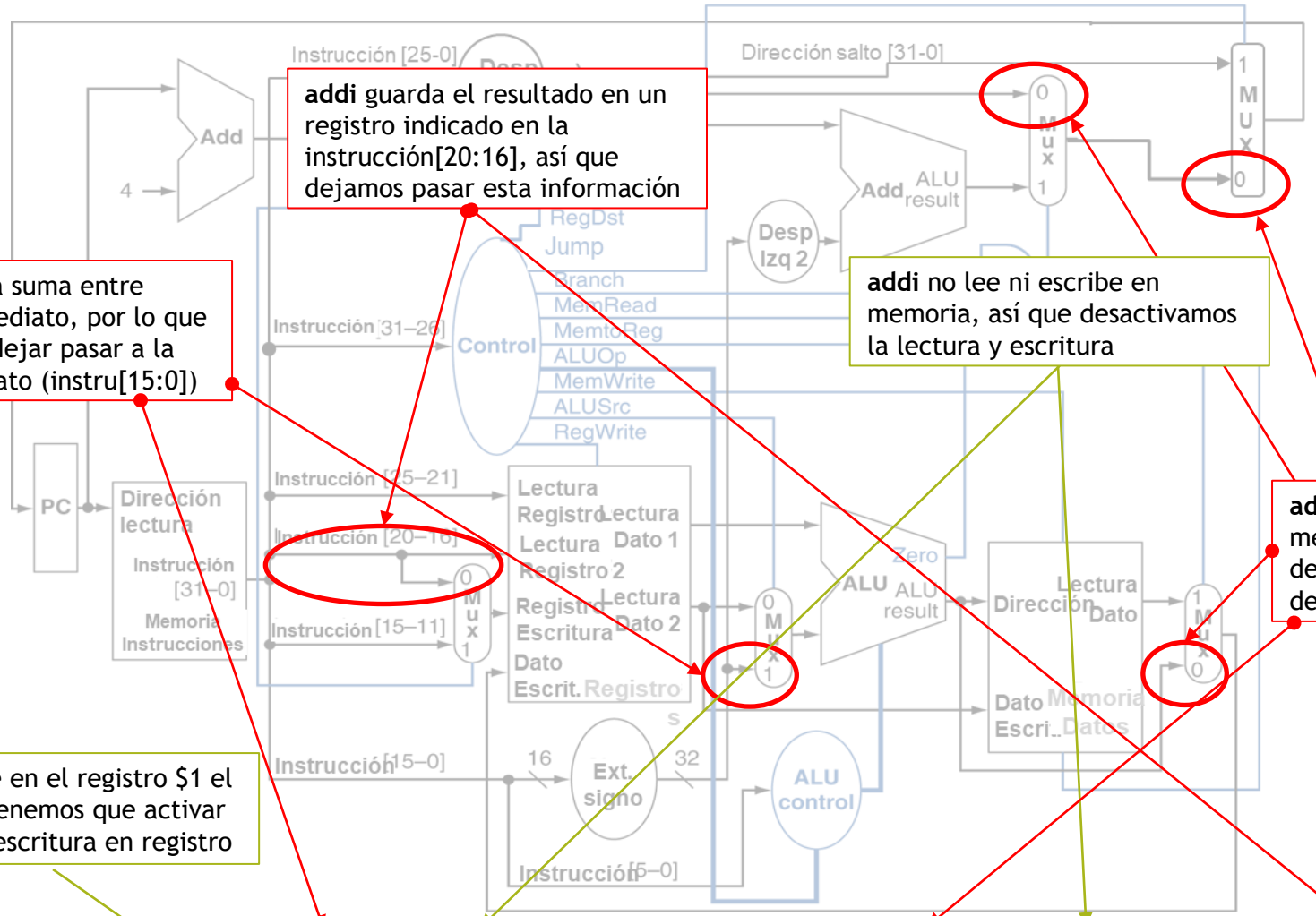
add es instrucción de tipo R, cuya entrada a la ALU es [1 0]

Ejercicio 3



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
add $1, $2, 10	1	1	0	0	0	0	0	0	0	0

Ejercicio 3



addi guarda el resultado en un registro indicado en la instrucción[20:16], así que dejamos pasar esta información

addi calcula la suma entre registro e inmediato, por lo que tenemos que dejar pasar a la ALU el inmediato (instru[15:0])

addi no lee ni escribe en memoria, así que desactivamos la lectura y escritura

addi no utiliza la memoria, así que dejamos pasar el dato de la salida de la ALU

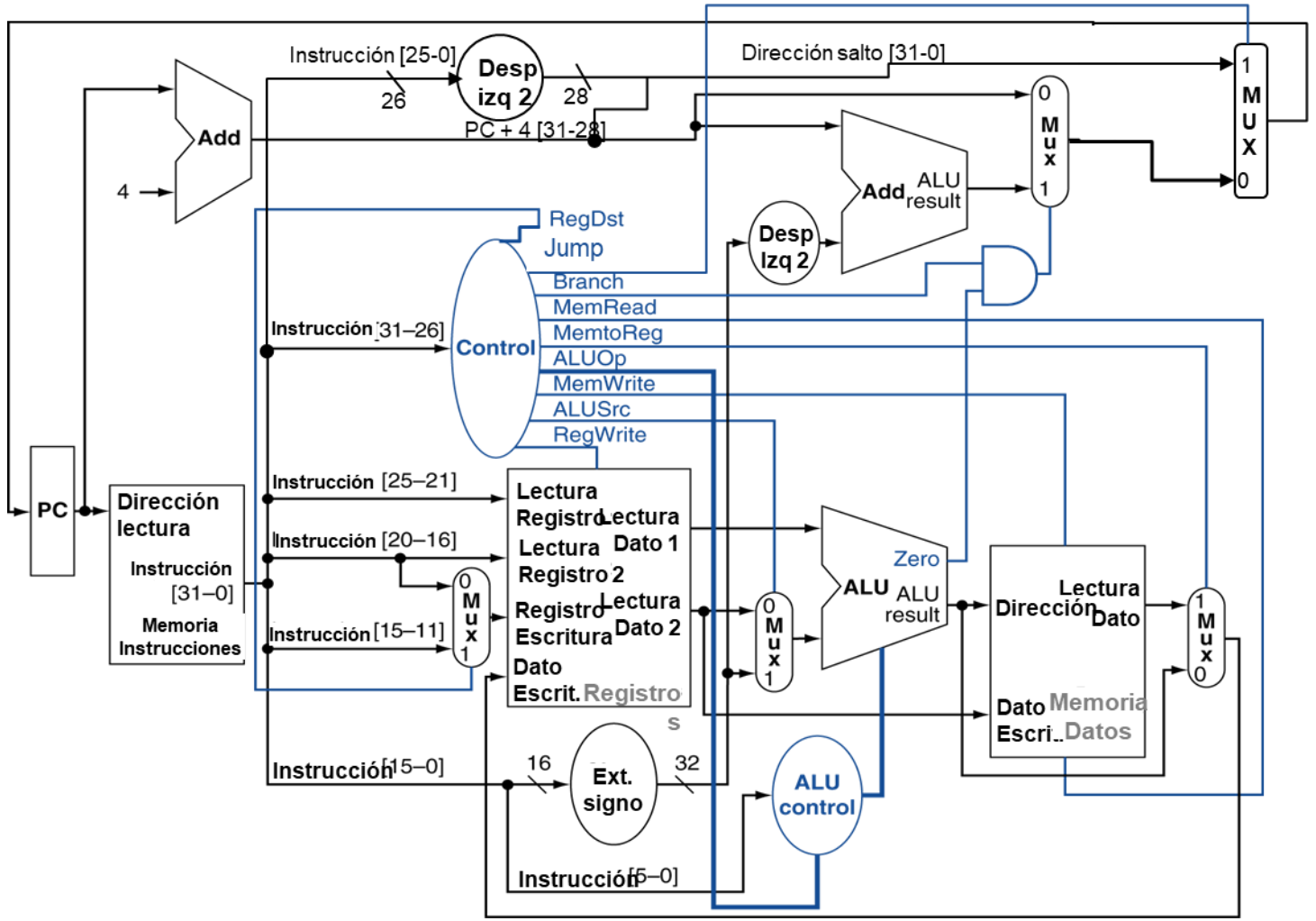
addi escribe en el registro $1 el resultado, tenemos que activar la señal de escritura en registro

addi no realiza saltos, así que desactivamos los saltos y dejamos pasar PC+4

	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
addi $1, $2, 10	1	1	0	0	0	0	0	0	0	0

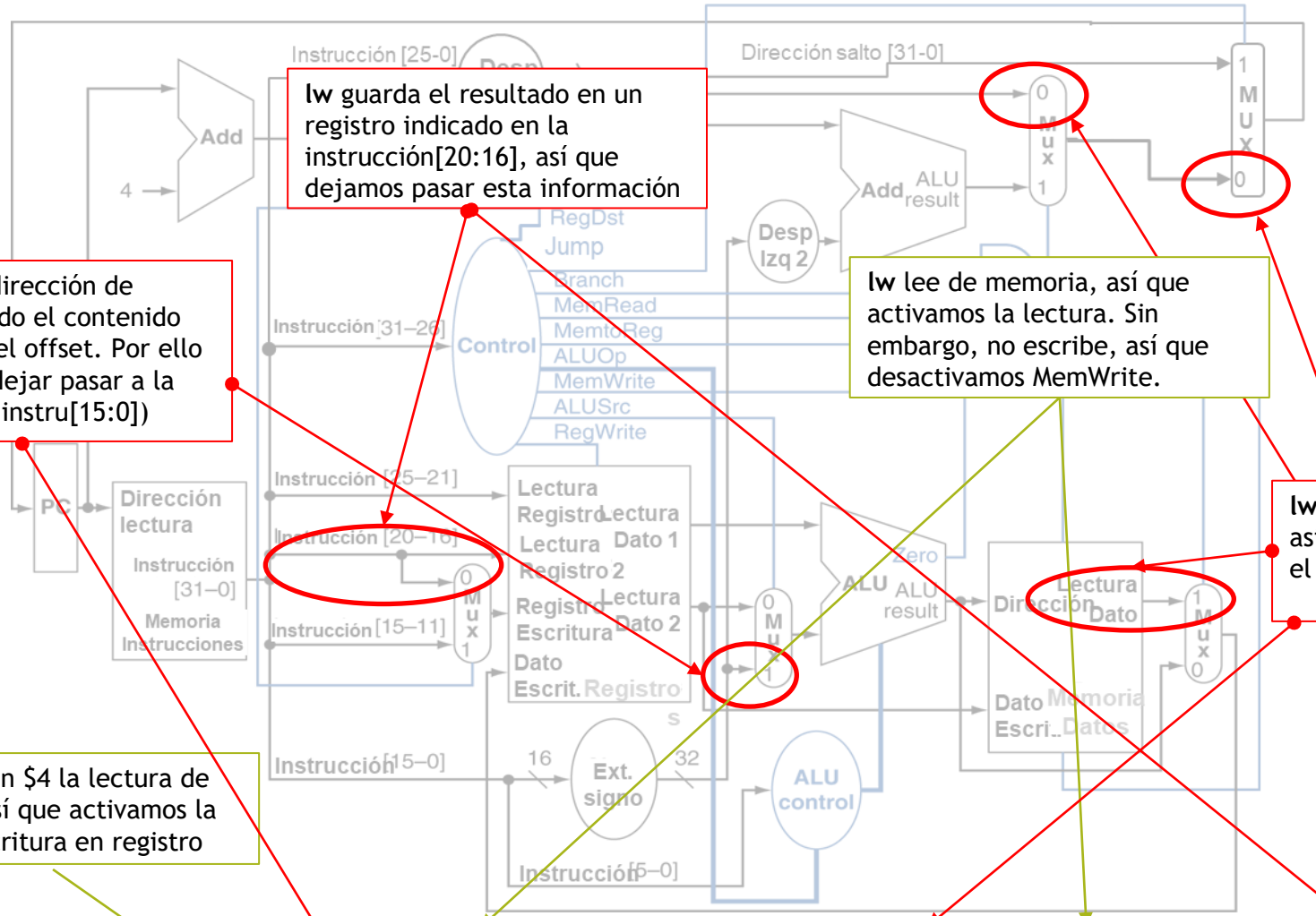
addi es instrucción de tipo I, cuya entrada a la ALU es [0 0]

Ejercicio 3



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
lw $4, 100($1)	1	1	0	0	0	1	1	0	0	0

Ejercicio 3



lw guarda el resultado en un registro indicado en la instrucción[20:16], así que dejamos pasar esta información

lw calcula la dirección de lectura sumando el contenido del registro y el offset. Por ello tenemos que dejar pasar a la ALU el offset (instru[15:0])

lw lee de memoria, así que activamos la lectura. Sin embargo, no escribe, así que desactivamos MemWrite.

lw sí utiliza la memoria, así que dejamos pasar el dato de la misma.

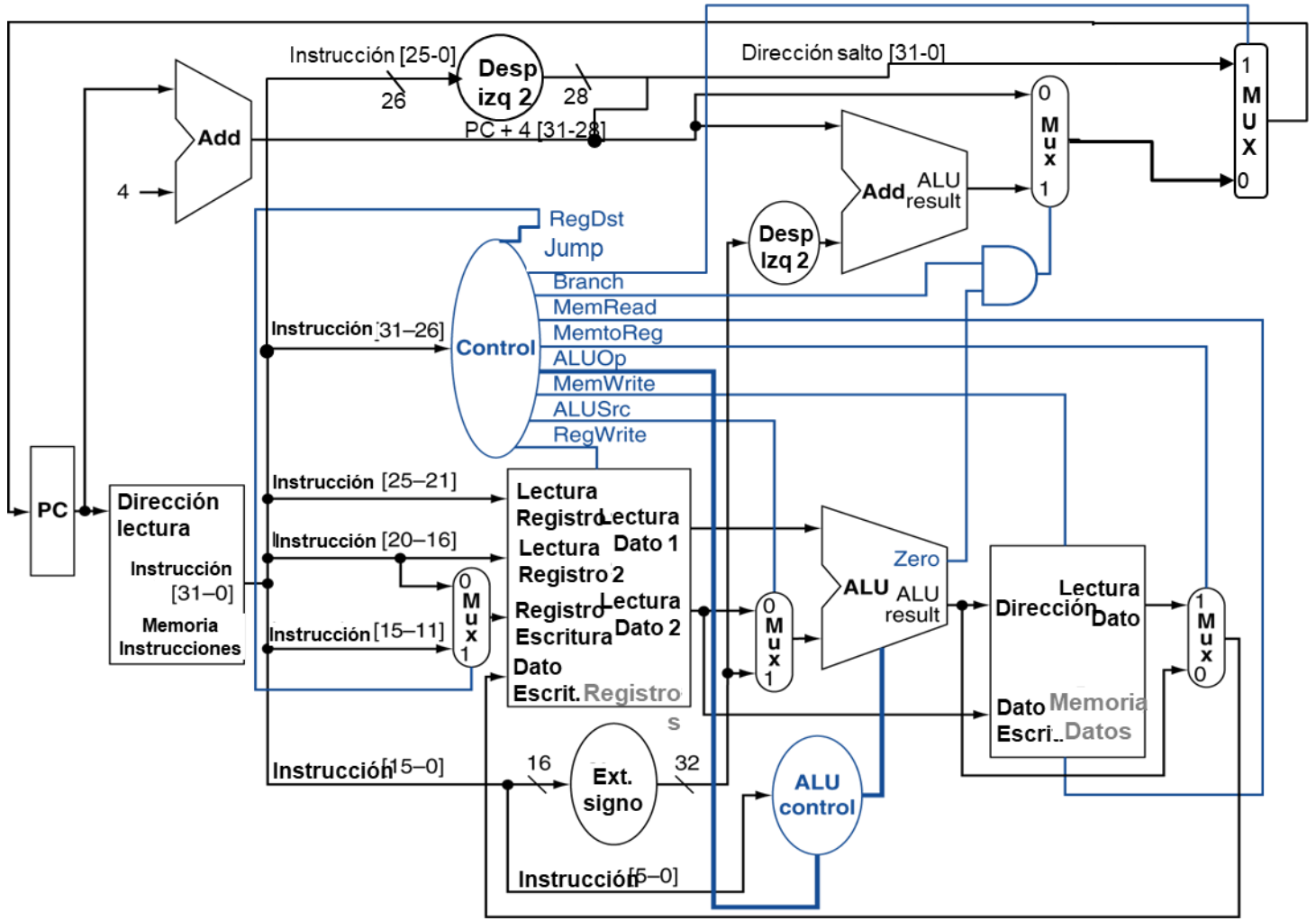
lw escribe en $4 la lectura de memoria, así que activamos la señal de escritura en registro

lw no realiza saltos, así que desactivamos los saltos y dejamos pasar PC+4

	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
lw $4, 100($1)	1	1	0	0	0	1	1	0	0	0

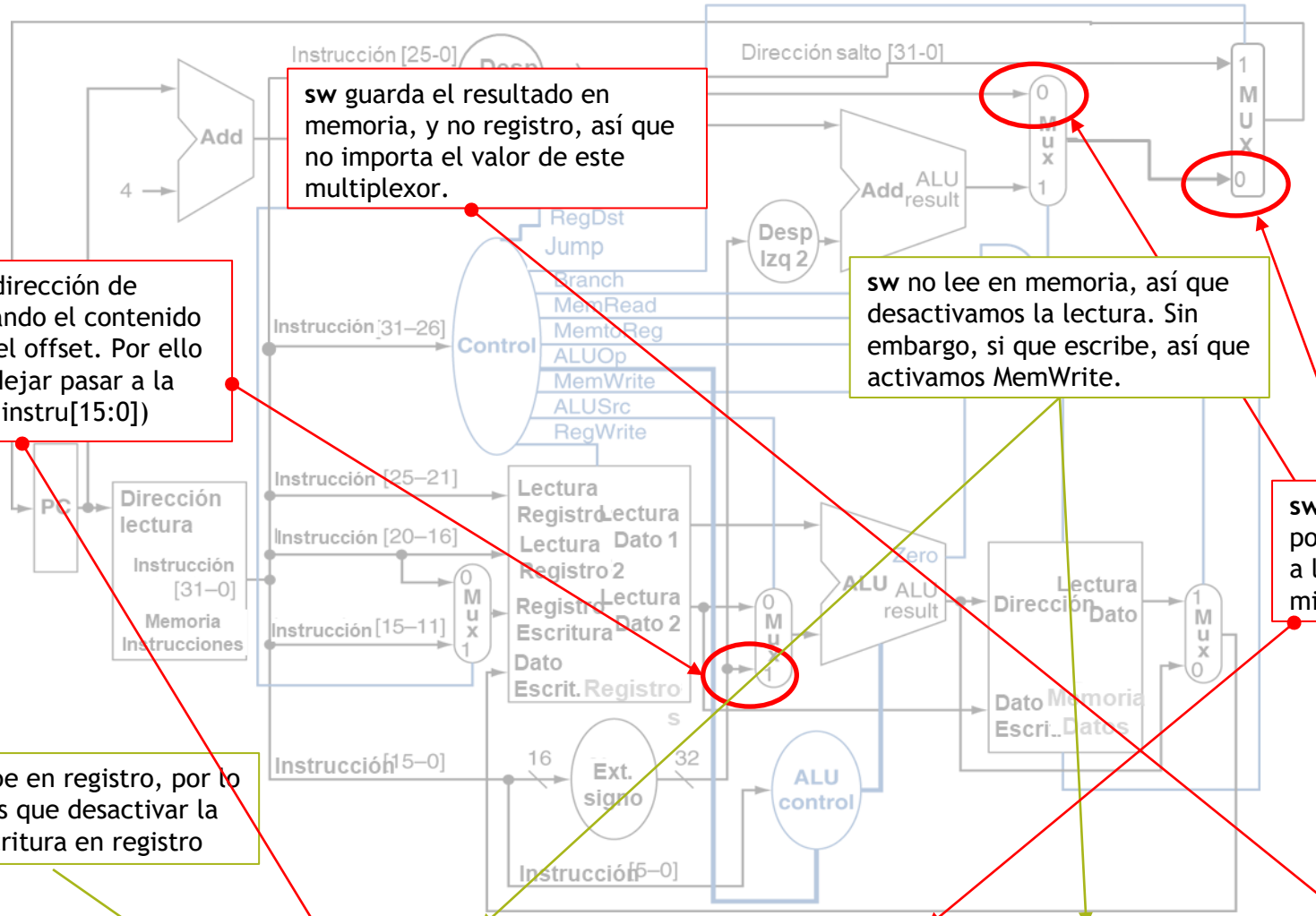
lw es instrucción de tipo I, cuya entrada a la ALU es [0 0]

Ejercicio 3



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
sw $1, 35($5)	0	1	1	0	0	X	0	0	0	X

Ejercicio 3



sw guarda el resultado en memoria, y no registro, así que no importa el valor de este multiplexor.

sw calcula la dirección de escritura sumando el contenido del registro y el offset. Por ello tenemos que dejar pasar a la ALU el offset (instru[15:0])

sw no lee en memoria, así que desactivamos la lectura. Sin embargo, si que escribe, así que activamos MemWrite.

sw escribe en memoria, por lo que no hay salida a los registros, dando lo mismo esta señal

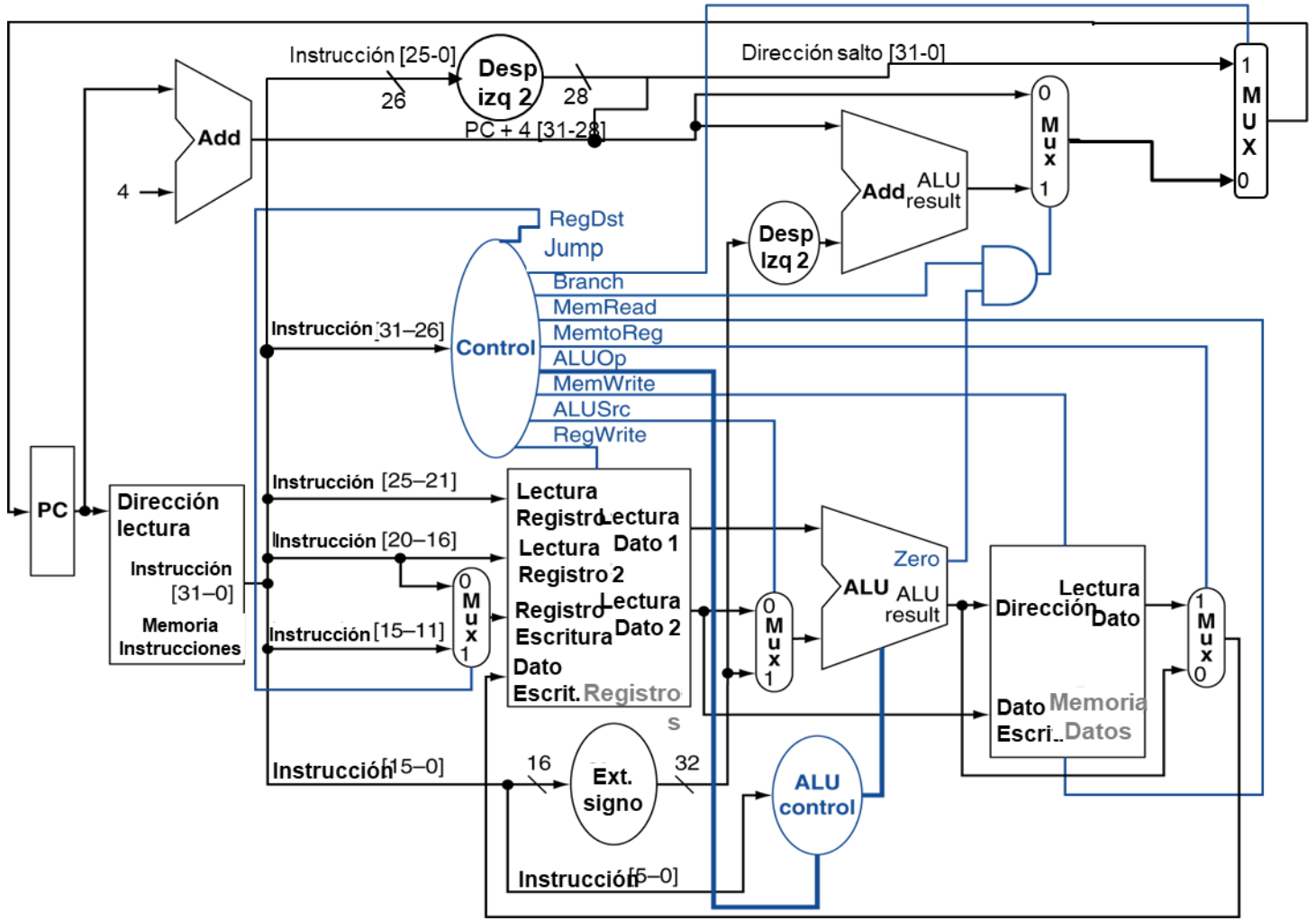
sw no realiza saltos, así que desactivamos los saltos y dejamos pasar PC+4

sw no escribe en registro, por lo que tenemos que desactivar la señal de escritura en registro

	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
sw $1, 35($5)	0	1	1	0	0	X	0	0	0	X

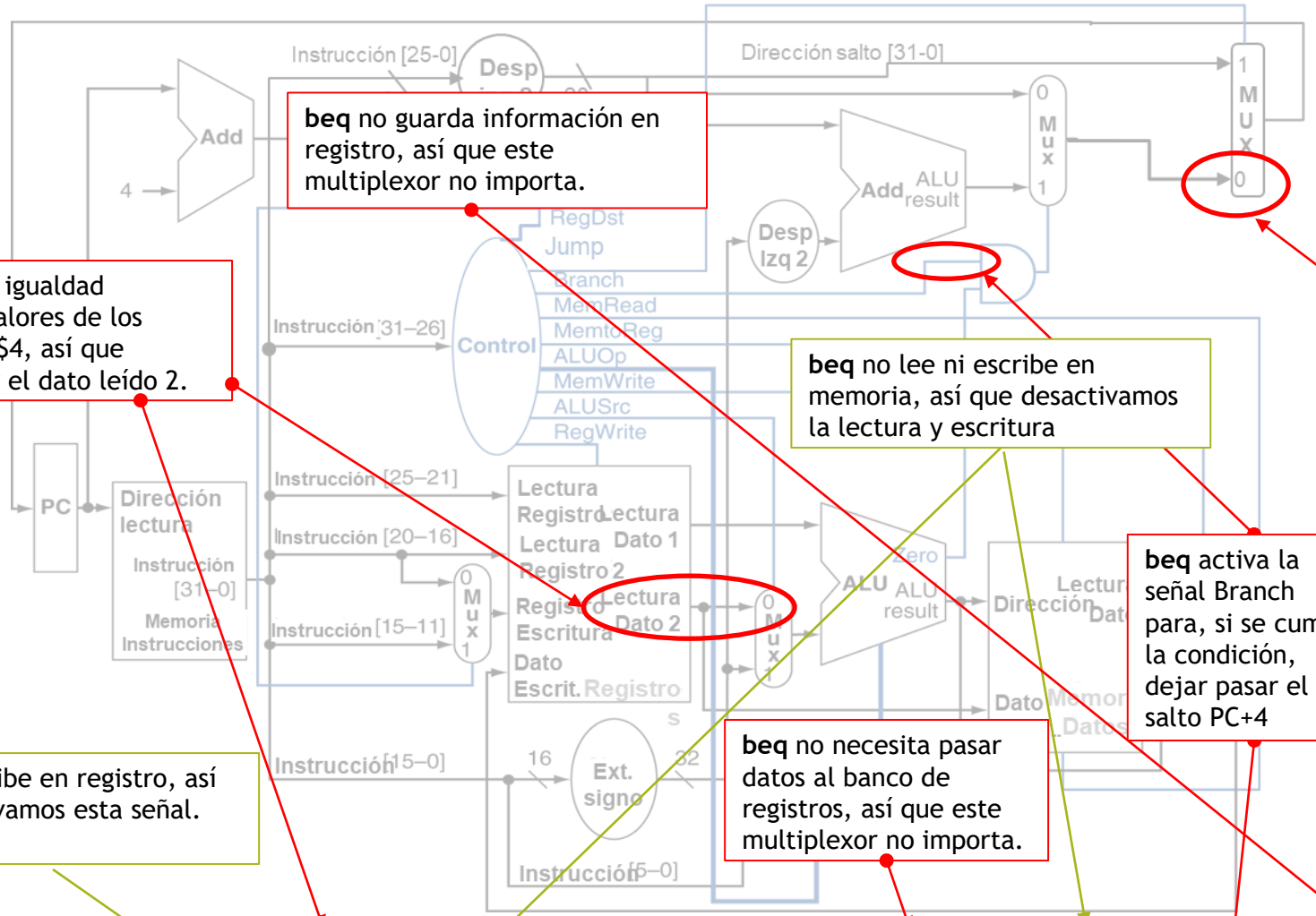
sw es instrucción de tipo I, cuya entrada a la ALU es [0 0]

Ejercicio 3



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
beq $3, $4, 1000	0	0	0	0	1	X	0	1	0	X

Ejercicio 3



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
beq $3, $4, 1000	0	0	0	0	1	X	0	1	0	X

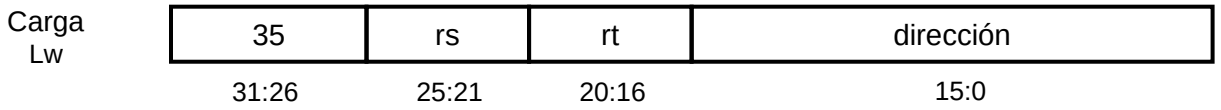
beq es instrucción de tipo salto, cuya entrada a la ALU es [0 1]

Ejercicio 4

- ¿Qué valor toman las señales de control de la UC al ejecutar la instrucción: 0x8D0B0008 ?

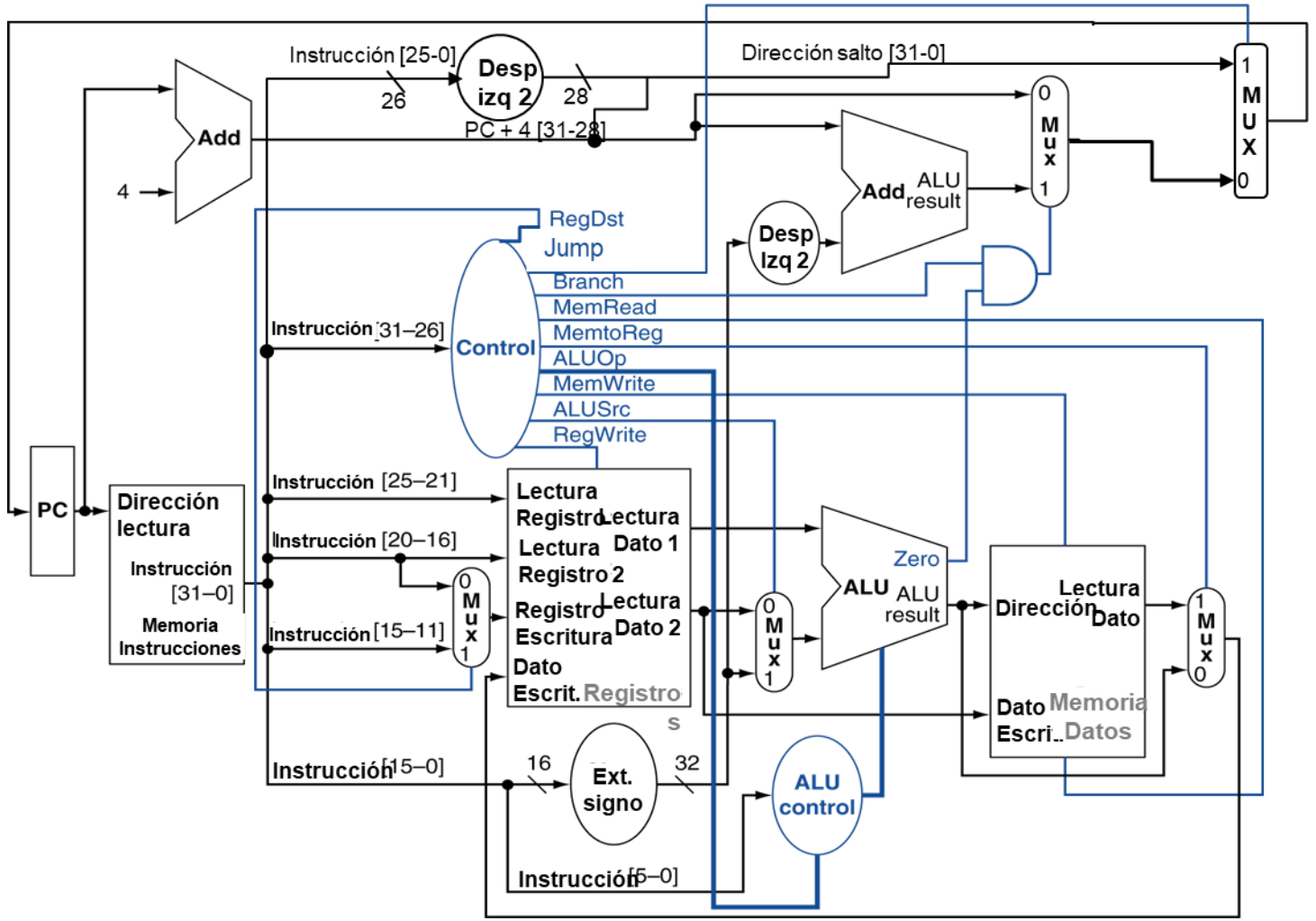
Ejercicio 4

- ¿Qué valor toman las señales de control de la UC al ejecutar la instrucción: 0x8D0B0008 ?
- Sacamos el binario: 1000 1101 0000 1011 0000 0000 0000 1000
- Obtenemos el código de operación que son los 6 primeros bits:
  - 100011 → 35 (lw) → tipo I de instrucción



- Extraemos el resto de los campos de la instrucción
- 100011 01000 01011 0000000000001000
- Lw $11, 8($8)

Ejercicio 4



	RegWrite	ALUSrc	MemWrite	ALUOp1	ALUOp0	Mem2Reg	MemRead	Branch	Jump	RegDst
Lw $11, 8($8)	1	1	0	0	0	1	1	0	0	0

Ejercicio 5

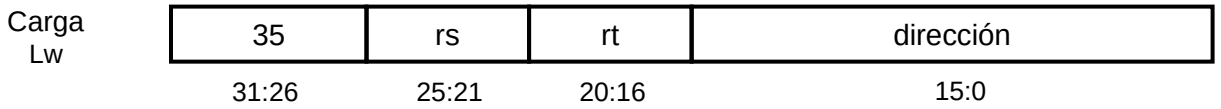
- Dadas las siguientes codificaciones de instrucciones:
  - 0x8C430010
  - 0x1023000C

Determinar:

1. Salida de la extensión de signo.
2. Entradas de la Unidad de Control de la ALU
3. Nueva dirección del PC después de ejecutar la instrucción e indicar el camino

Ejercicio 5

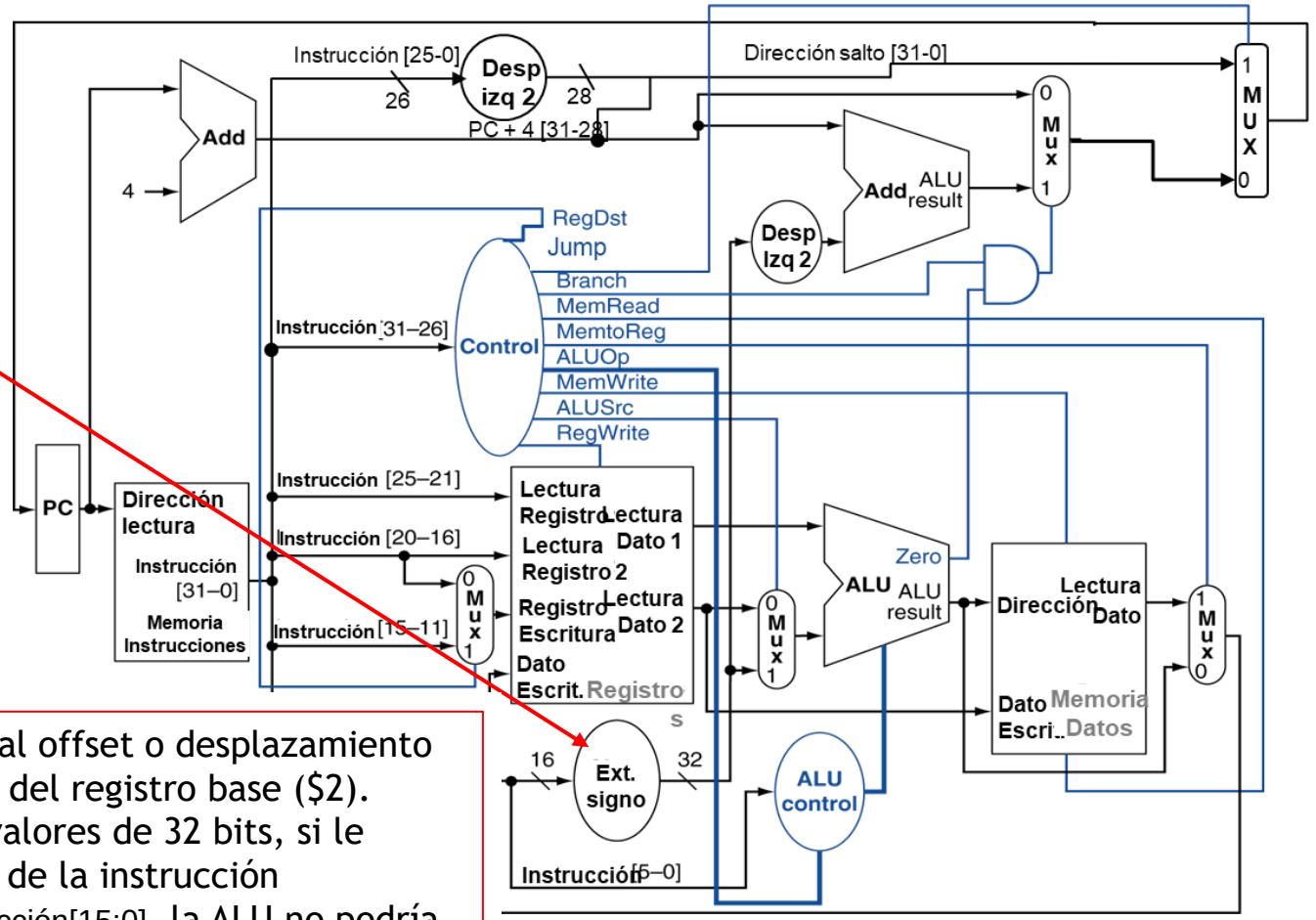
- 0x8C430010
- Sacamos el binario: 1000 1100 0100 0011 0000 0000 0001 0000
- Obtenemos el código de operación que son los 6 primeros bits:
  - 100011 → 35 (lw) → tipo I de instrucción



- Extraemos el resto de los campos de la instrucción
- 100011 00010 00011 0000000000010000
- Lw $3, 16($2)

Ejercicio 5

- Lw $3, 16($2)
- Determinar:
  1. Salida de la extensión de signo.



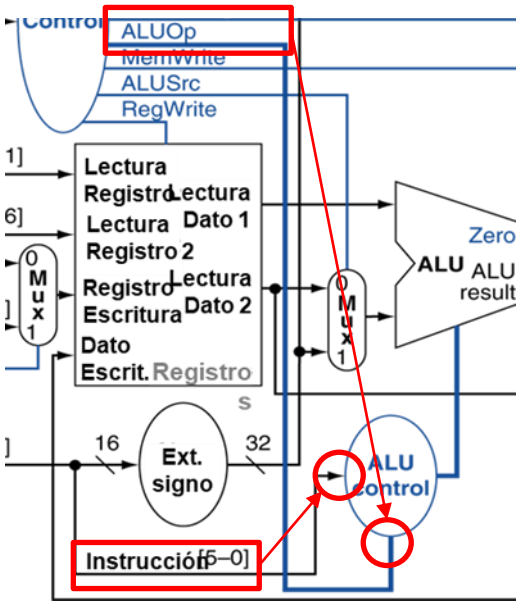
La extension de signo se aplica al offset o desplazamiento que se aplica sobre la dirección del registro base ($2). Puesto que la ALU trabaja con valores de 32 bits, si le Pasamos únicamente los 16 bits de la instrucción correspondientes al offset instrucción[15:0], la ALU no podría funcionar. Así que se extiende el valor repicando el bit de signo del dato original hasta obtener el tamaño deseado.

Ejercicio 5

- Lw $3, 16($2) → 100011 00010 00011 0000000000010000
- Determinar:
  1. Salida de la extensión de signo.
     1. Si la entrada es 0000000000001000, la salida será
     2. 00000000000000000000000000010000

Ejercicio 5

- Lw $3, 16($2) → 100011 00010 00011 0000000000010000
- Determinar:
  1. Salida de la extensión de signo.
  2. Entradas de la Unidad de Control de la ALU



Ejercicio 5

- Lw $3, 16($2) → 100011 00010 00011 0000000000010000
- Determinar:
  1. Salida de la extensión de signo.
  2. Entradas de la Unidad de Control de la ALU
     1. ALUOp → [0 0] ; Instrucción[5-0] → [010000]

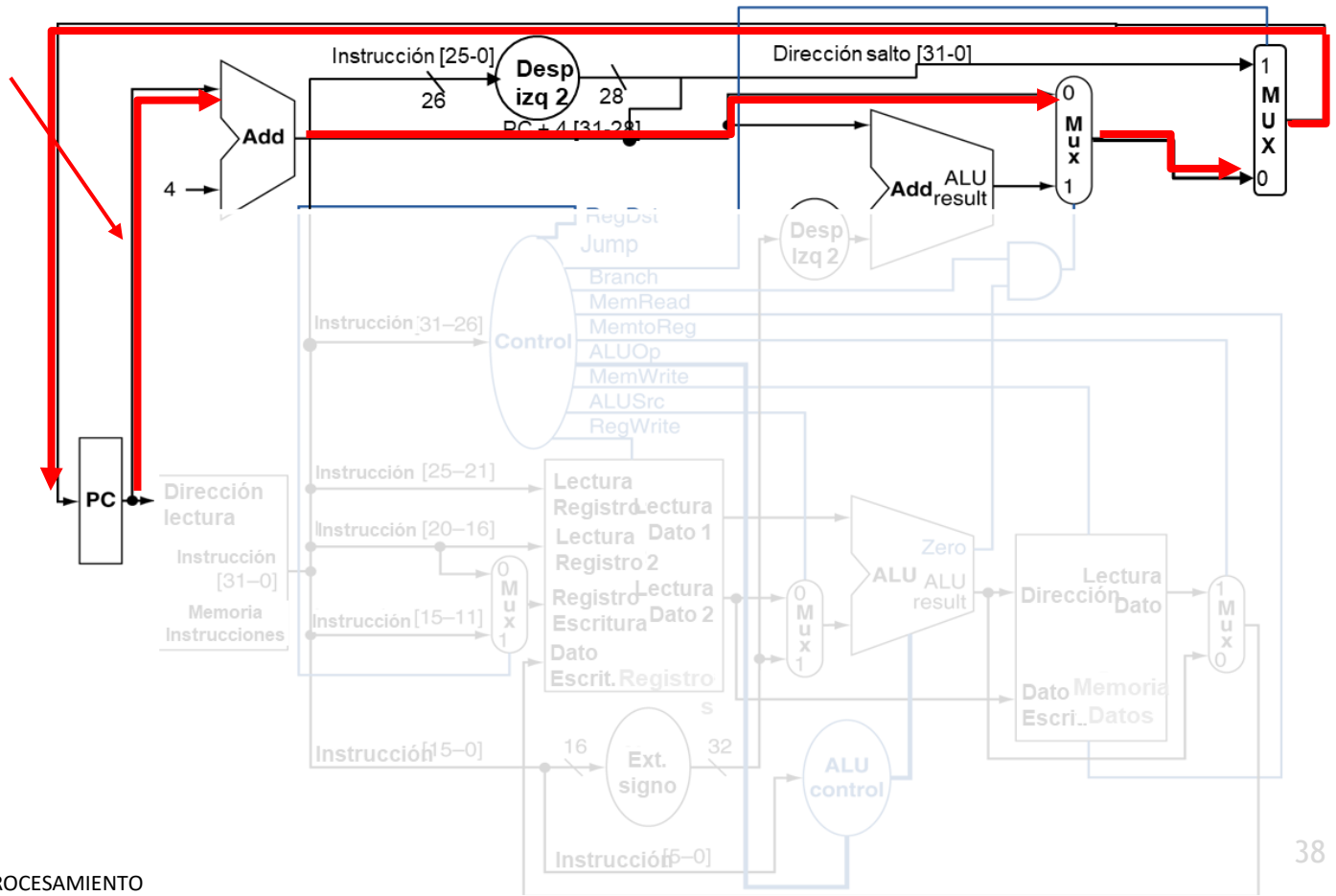
Instrucción	ALUOp	Operación	Campo funct Inst[5-0]	Acción de la ALU	Entrada de control a la ALU
lw	00	Cargar palabra	010000	Suma	0010
sw	00	Almacenar palabra	XXXXXX	Suma	0010
beq	01	Saltar si igual	XXXXXX	Resta	0110
R-type	10	Suma	100000	Suma	0010
		Resta	100010	Resta	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		Activar si menor que	101010	Activar si menor que	0111

Ejercicio 5

- Lw $3, 16($2) → 100011 00010 00011 0000000000010000
- Determinar:
  1. Salida de la extensión de signo.
  2. Entradas de la Unidad de Control de la ALU
  3. Nueva dirección del PC después de ejecutar la instrucción e indicar el camino

Ejercicio 5

- Lw $3, 16($2) → 100011 00010 00011 0000000000010000
- Nueva dirección del PC después de ejecutar la instrucción y camino
  - **PC = PC+4**
  - Camino



Ejercicio 5

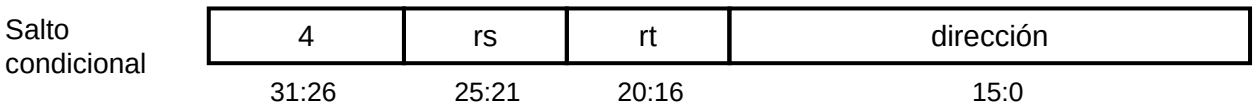
- Dadas las siguientes codificaciones de instrucciones:
  - 0x8C430010
  - 0x1023000C

Determinar:

1. Salida de la extensión de signo.
2. Entradas de la Unidad de Control de la ALU
3. Nueva dirección del PC después de ejecutar la instrucción e indicar el camino

Ejercicio 5

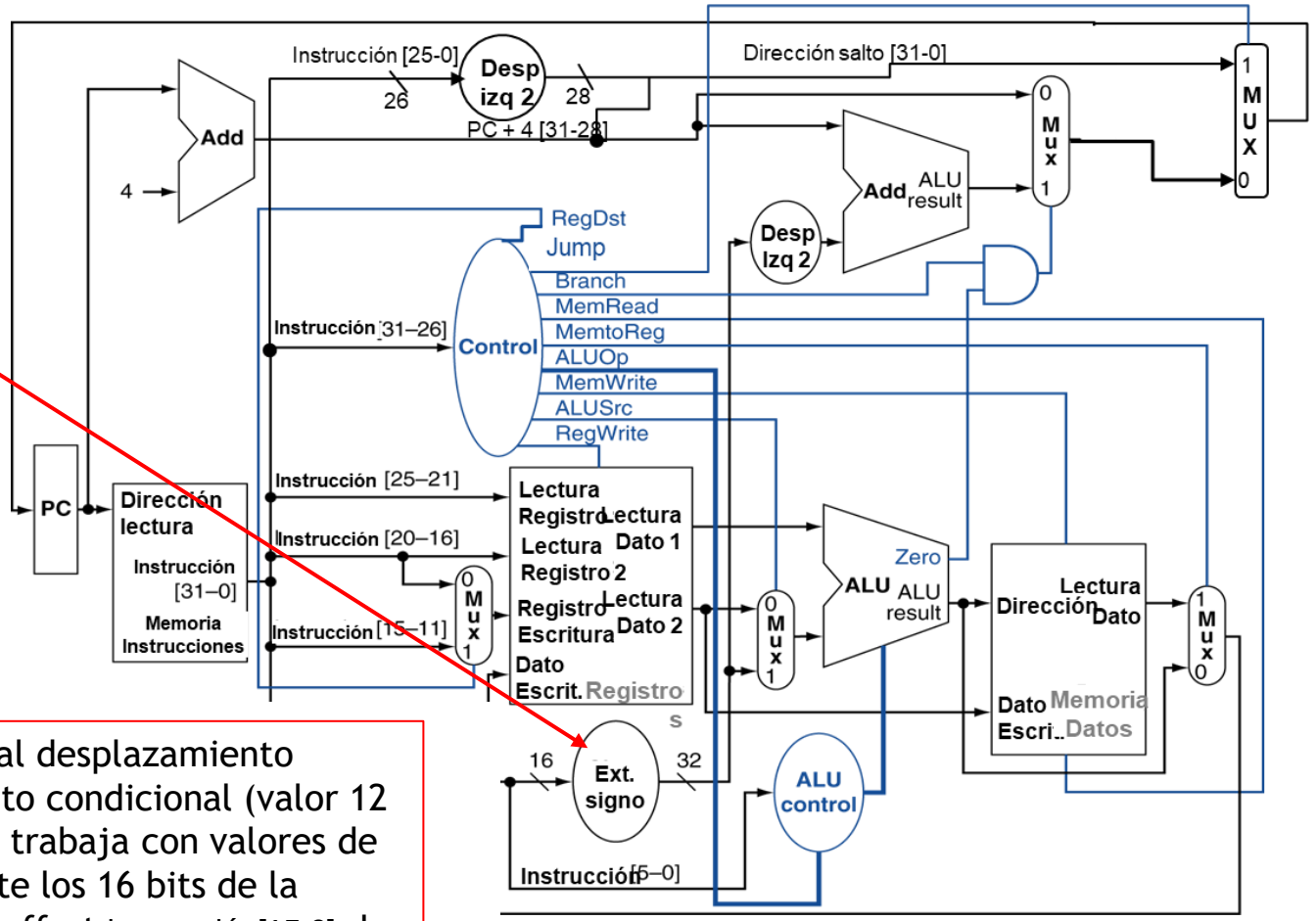
- 0x1023000C
- Sacamos el binario: 0001 0000 0010 0011 0000 0000 0000 1100
- Obtenemos el código de operación que son los 6 primeros bits:
  - 000100 → 4 (salto condicional, **beq**) → tipo I de instrucción



- Extraemos el resto de los campos de la instrucción
- 000100 00001 00011 0000000000001100
- beq $1, $3, 12

Ejercicio 5

- beq $1, $3, 12
- Determinar:
  1. Salida de la extensión de signo.



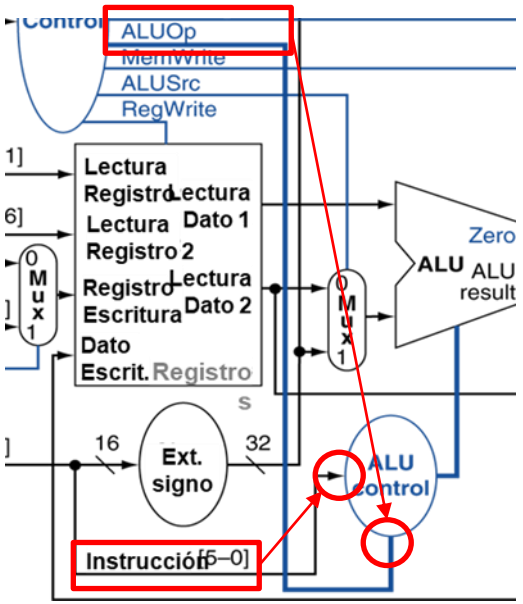
La extensión de signo se aplica al desplazamiento que se usará para realizar el salto condicional (valor 12 en este caso. Puesto que la ALU trabaja con valores de 32 bits, si le Pasamos únicamente los 16 bits de la instrucción correspondientes al offset instrucción[15:0], la ALU no podría funcionar. Así que se extiende el valor repicando el bit de signo del dato original hasta obtener el tamaño deseado.

Ejercicio 5

- beq $1, $3, 12 → 000100 00001 00011 0000000000001100
- Determinar:
  1. Salida de la extensión de signo.
     1. Si la entrada es 0000000000001100, la salida será
     2. 0000000000000000 0000000000001100

Ejercicio 5

- beq $1, $3, 12 → 000100 00001 00011 0000000000001100
- Determinar:
  1. Salida de la extensión de signo.
  2. Entradas de la Unidad de Control de la ALU



Ejercicio 5

- beq $1, $3, 12 → 000100 00001 00011 0000000000001100
- Determinar:
  1. Salida de la extensión de signo.
  2. Entradas de la Unidad de Control de la ALU
     1. ALUOp → [0 1] ; Instrucción[5-0] → [001100]

Instrucción	ALUOp	Operación	Campo funct Inst[5-0]	Acción de la ALU	Entrada de control a la ALU
lw	00	Cargar palabra	XXXXXX	Suma	0010
Sw	00	Almacenar palabra	XXXXXX	Suma	0010
Beq	01	Saltar si igual	001100	Resta	0110
R-type	10	Suma	100000	Suma	0010
		Resta	100010	Resta	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		Activar si menor que	101010	Activar si menor que	0111

Ejercicio 5

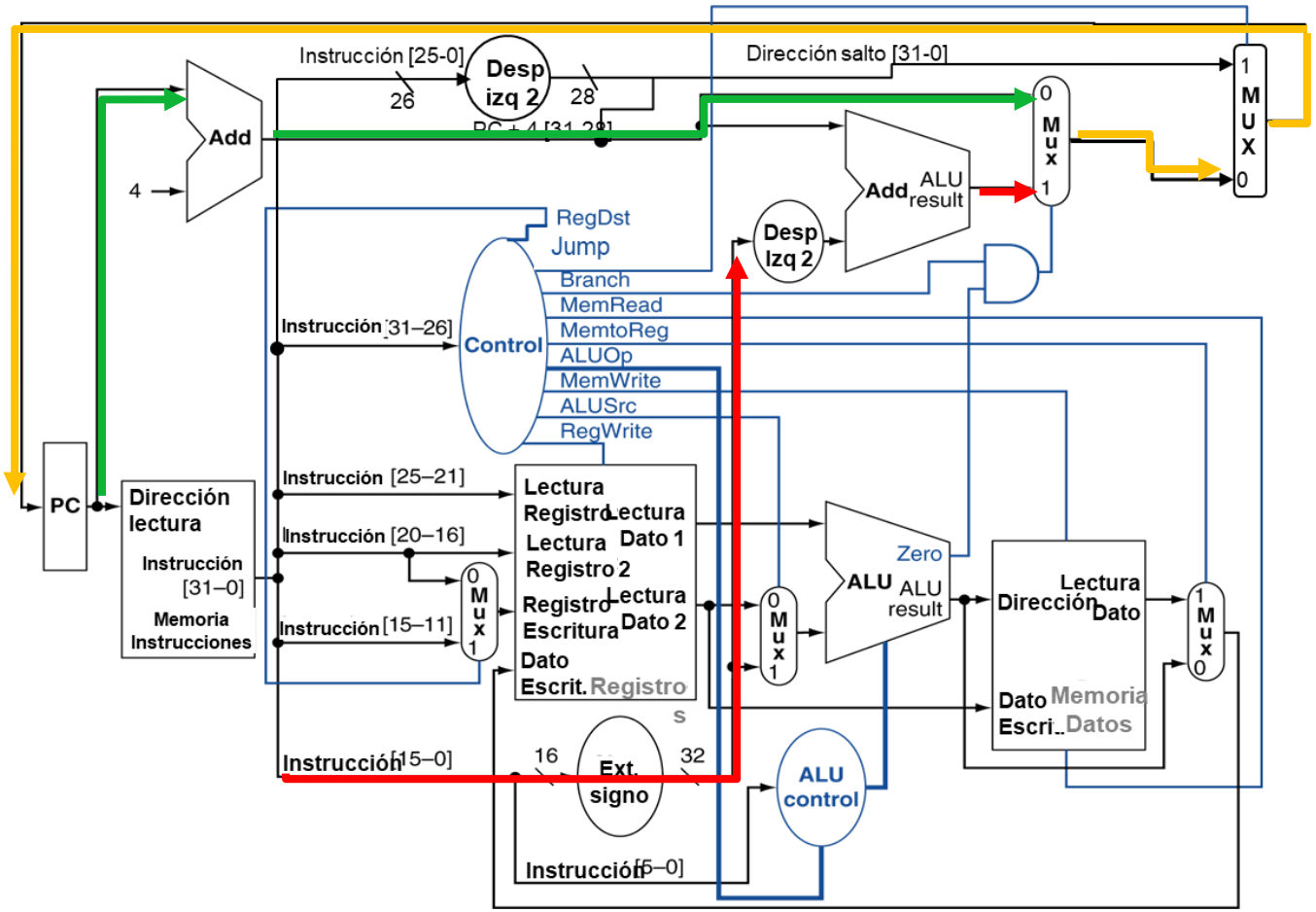
- beq $3, $1, 12 → 000100 00001 00011 0000000000001100
- Determinar:
  1. Salida de la extensión de signo.
  2. Entradas de la Unidad de Control de la ALU
  3. Nueva dirección del PC después de ejecutar la instrucción e indicar el camino

Ejercicio 5

- beq $1, $3, 12 → 000100 00001 00011 0000000000001100
- Nueva dirección del PC después de ejecutar la instrucción y camino

Hay dos posibles caminos:

- Si se cumple la igualdad entre $1 y $3 se toma el camino rojo, ya que la ALU pondrá a 1 el valor “Zero” y la señal “Branch” está a 1 por el Código de instrucción. En este case se incrementa el PC en 12*4 por el Desp Izq de 2 bits.
- Si no se cumple la igualdad entre $1 y $3 se toma el camino verde, que es in incremento normal del contardor +4.
- El camino amarillo es común, ya que es el que reescribe el valor antiguo de PC por el nuevo.



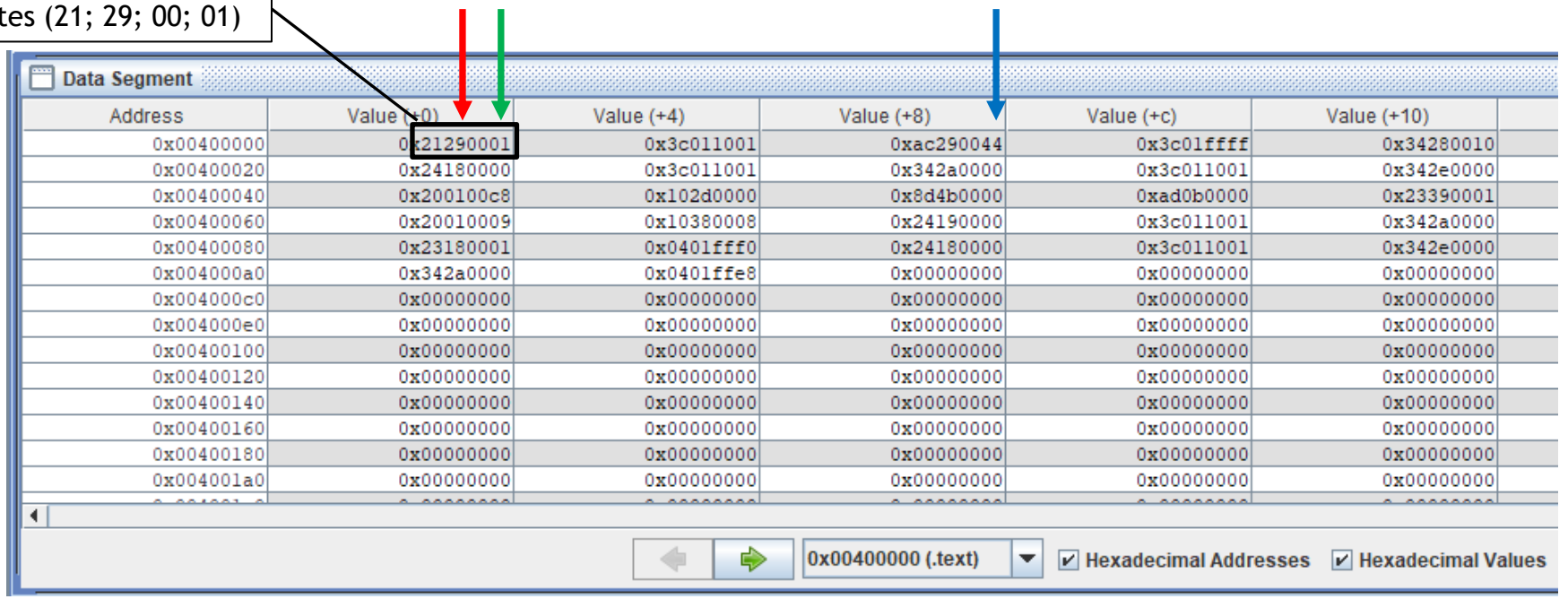
Ejercicio 5

- Recordad que el Desplazamiento a la Izquierda de 2 bits cuando se hace un salto condicional es para que el incremento esté alineado con la memoria.

- Si saltamos el valor que indica la instrucción sin deplazar, y esta fuese **beq $1, $2, 2**, tendríamos el PC en la dirección 0x00400000 (flecha verde), y pasaríamos a la 0x00400002 (flecha roja). Esto no sería correcto ya que una instrucción no puede empezar a mitad de una palabra.

- Si saltamos con el valor indicado en la instrucción con el desplazamiento a la izquierda de 2 bits, pasaríamos a la dirección 0x00400008 (2*4 = 8), correspondiente a la flecha azul, que es una dirección de memoria 2 veces posterior.

Cada dirección son
4 bytes (21; 29; 00; 01)



Ejercicio 6

- Dadas las siguientes codificaciones de instrucciones:

  (a) 0x8C430010

  (b) 0x1023000C

Suponiendo que la memoria de datos está TODA a 0 y los registros contienen la siguiente información:

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(a)	0	1	2	3	-4	5	6	8	1	-32
(b)	0	-16	-2	-3	4	-10	-6	-1	8	-4

1. Mostrar los valores de salida de cada Mx (considerar ruta con *jump*)
2. Valores de entrada de la ALU y de las dos unidades de SUMA
3. Valores de todas las entradas del BANCO de REGISTROS

Ejercicio 6

- Dadas las siguientes codificaciones de instrucciones:

  (a) 0x8C430010

  (b) 0x1023000C

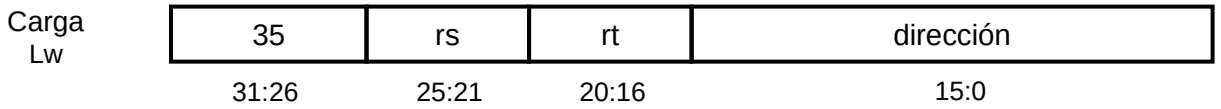
Suponiendo que la memoria de datos está TODA a 0 y los registros contienen la siguiente información:

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(a)	0	1	2	3	-4	5	6	8	1	-32
(b)	0	-16	-2	-3	4	-10	-6	-1	8	-4

1. Mostrar los valores de salida de cada Mx (considerar ruta con *jump*)
2. Valores de entrada de la ALU y de las dos unidades de SUMA
3. Valores de todas las entradas del BANCO de REGISTROS

Ejercicio 6

- 0x8C430010
- Sacamos el binario: 1000 1100 0100 0011 0000 0000 0001 0000
- Obtenemos el código de operación que son los 6 primeros bits:
  - 100011 → 35 (lw) → tipo I de instrucción



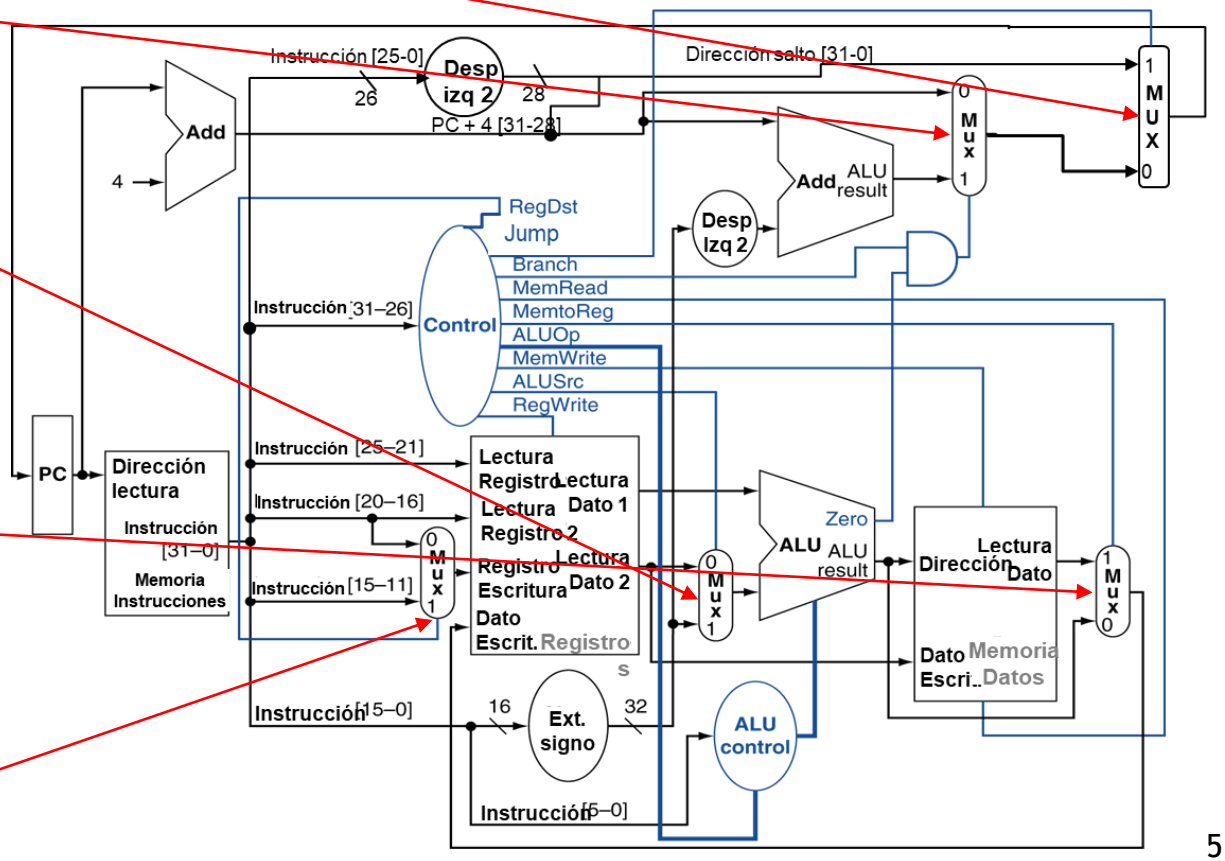
- Extraemos el resto de los campos de la instrucción
- 100011 00010 00011 0000000000010000
- Lw $3, 16($2)

Ejercicio 6

- 0x8C430010 --> Lw $3, 16($2)
  - Mostrar los valores de salida de cada Mx (considerar ruta con *jump*)

Jump: al no ser una instrucción de salto incondicional, el Mx deja pasar el PC+4.	PC +4
Branch: al no ser una instrucción de salto condicional, el Mx deja pasar el PC+4.	PC +4
ALUSrc: La instrucción lw pasa el offset por la extensión de signo y se suma al registro $2, por lo que el Mx deja pasar el dato que viene por la entrada 1, que es el offset.	16
Mem2Reg: la instrucción lw lee de memoria de datos la dirección [2+16] ($2 tiene un 2 almacenado) y la guarda en el registro $3. Por lo tanto, tras la lectura en memoria se tiene que enviar ese dato y no la salida de la ALU. El Mx deja pasar la entrada 1 (dato leído que es 0 ya que se dice en el enunciado que la memoria está vacía).	0
RegDst: la instrucción lw guarda en registro el dato leído de memoria. El registro donde se ha almacenado es el $3.	3

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(a)	0	1	2	3	-4	5	6	8	1	-32

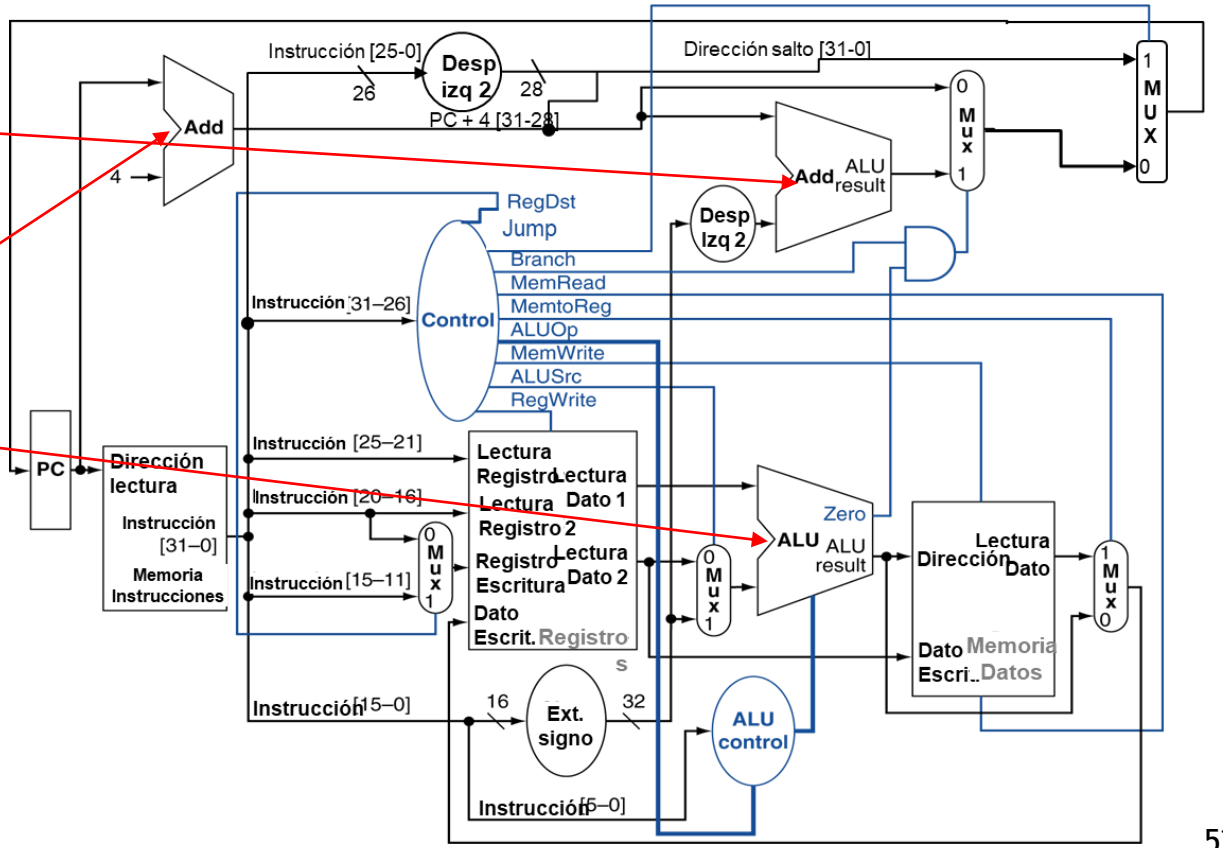


Ejercicio 6

- 0x8C430010 --> Lw $3, 16($2)
  - Mostra valores de entrada de la ALU y de las dos unidades de SUMA

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(a)	0	1	2	3	-4	5	6	8	1	-32

ALU Add Salto condicional. Esta unidad de suma recibe el PC+4 por un lado. Por el otro, recibe los bits [15:0] de la instrucción extendidos y desplazados.	PC+4; 16*4
ALU Add +4. Esta unidad de control recibe el PC y un 4	PC; 4
ALU: La ALU recibe los valores a sumar para calcular la dirección de la memoria que queremos leer [16+$2].	2; 16

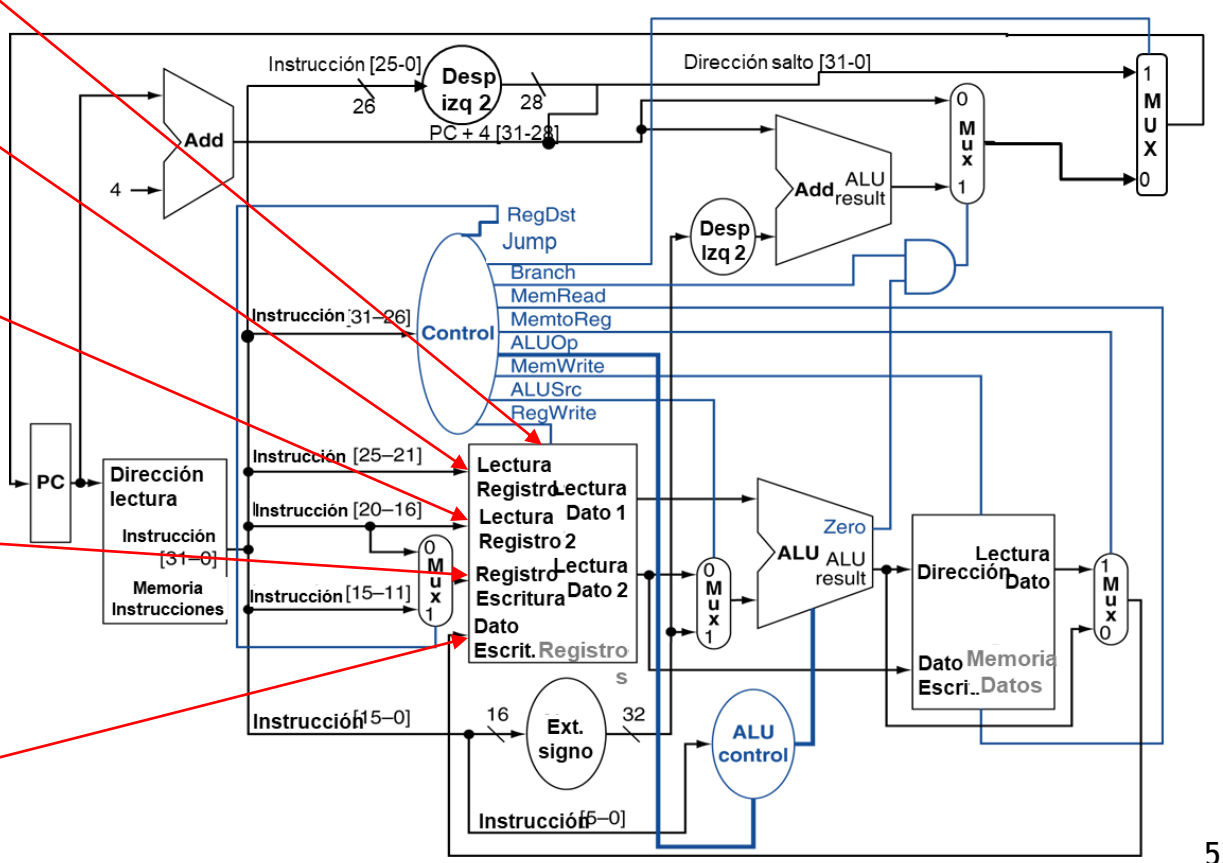


Ejercicio 6

- 0x8C430010 --> Lw $3, 16($2)
  - Mostrar valores de todas las entradas del BANCO de REGISTROS

RegWrite: como la instrucción lw escribe en el registro $3 el dato leído de memoria, esta señal se activa.	1
Registro 1: registro base del direccionamiento a memoria que se encuentra en el registro $2. Por lo tanto, la entrada aquí es un 2 para indicar la lectura del segundo registro.	2
Registro 2: recibe el valor de la instrucción[20:16], que es un 3, aunque no se utiliza como lectura.	3
Registro escritura: la instrucción lw escribe en registro lo leído en memoria. Por lo tanto, este registro recibe como entrada el nombre del registro sobre el que queremos escribir, definido en instrucción[20:16]. Esta entrada indica que se escribirá en el registro $3, por lo que es un 3.	3
Dato escritura: la instrucción lw guarda en el banco de registros el dato leído en memoria. Por lo tanto, aquí se almacena el dato de memoria que el enunciado decía que estaba vacía.	0

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(a)	0	1	2	3	-4	5	6	8	1	-32



Ejercicio 6

- Dadas las siguientes codificaciones de instrucciones:

  (a) 0x8C430010

  (b) 0x1023000C

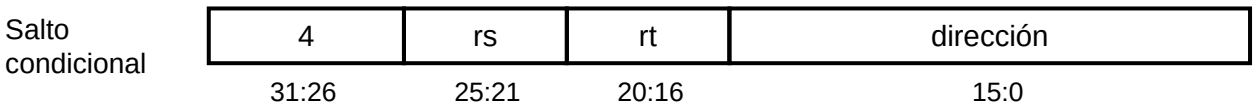
Suponiendo que la memoria de datos está TODA a 0 y los registros contienen la siguiente información:

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(a)	0	1	2	3	-4	5	6	8	1	-32
(b)	0	-16	-2	-3	4	-10	-6	-1	8	-4

1. Mostrar los valores de salida de cada Mx (considerar ruta con *jump*)
2. Valores de entrada de la ALU y de las dos unidades de SUMA
3. Valores de todas las entradas del BANCO de REGISTROS

Ejercicio 6

- 0x1023000C
- Sacamos el binario: 0001 0000 0010 0011 0000 0000 0000 1100
- Obtenemos el código de operación que son los 6 primeros bits:
  - 000100 → 4 (salto condicional, **beq**) → tipo I de instrucción



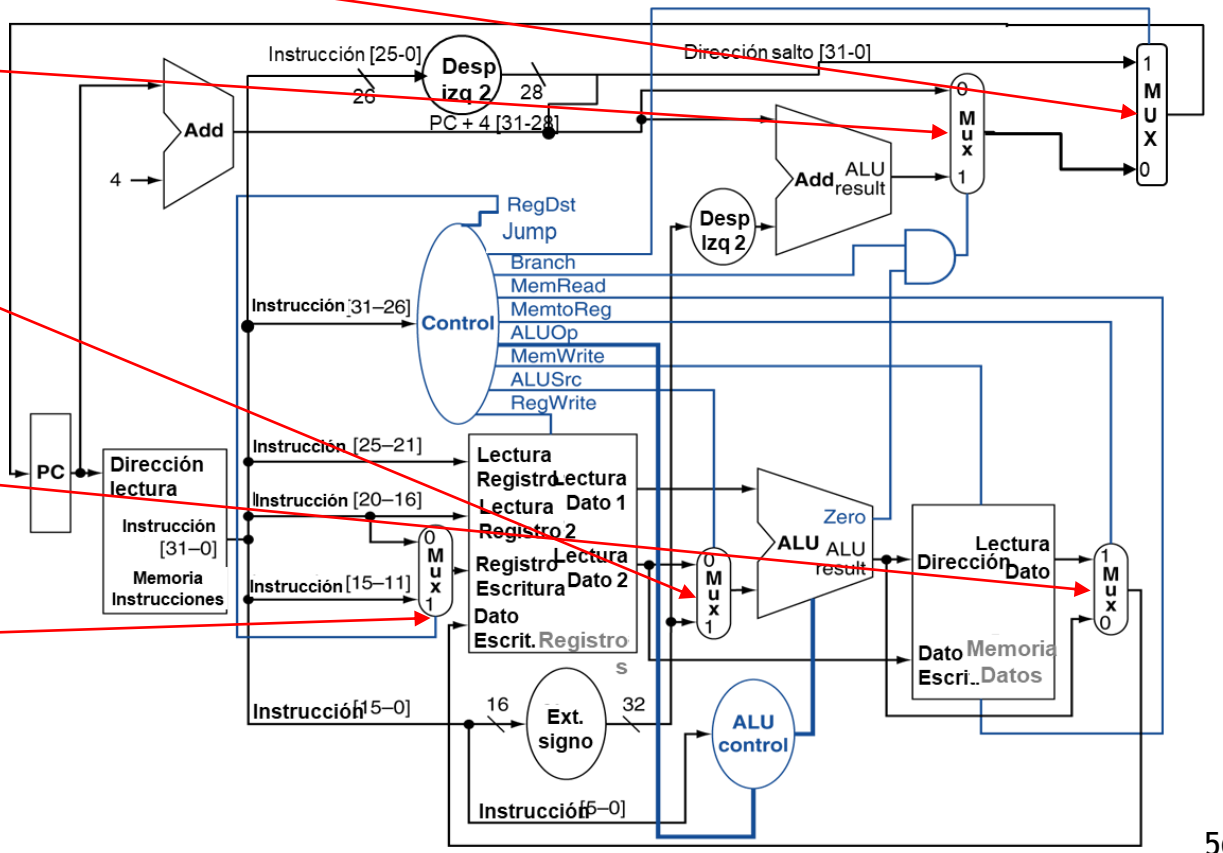
- Extraemos el resto de los campos de la instrucción
- 000100 00001 00011 0000000000001100
- beq $1, $3, 12

Ejercicio 6

- 0x1023000C --> beq $1, $3, 12
  - Mostrar los valores de salida de cada Mx (considerar ruta con *jump*)

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(b)	0	-16	-2	-3	4	-10	-6	-1	8	-4

Jump: al no ser una instrucción de salto incondicional, el Mx deja pasar el PC+4.	PC +4
Branch: la instrucción es de salto condicional, pero $1≠$3 ya que -16≠-3. Así que no hay salto y se pasa el PC+4.	PC +4
ALUSrc: La instrucción beq pasa el inmediato extendido a la **ALU Add,** y los registros a comparar a la ALU. Por lo tanto, el Mx deja pasar el dato del registro $3, es decir, un -3.	-3
Mem2Reg: la instrucción beq no lee de memoria ni requiere pasar un dato a registro para su escritura, así que este Mx no importa en su ruta de datos.	X
RegDst: la instrucción beq no escribe en el banco de registros, por lo que este Mx no importa en su ruta de datos.	X

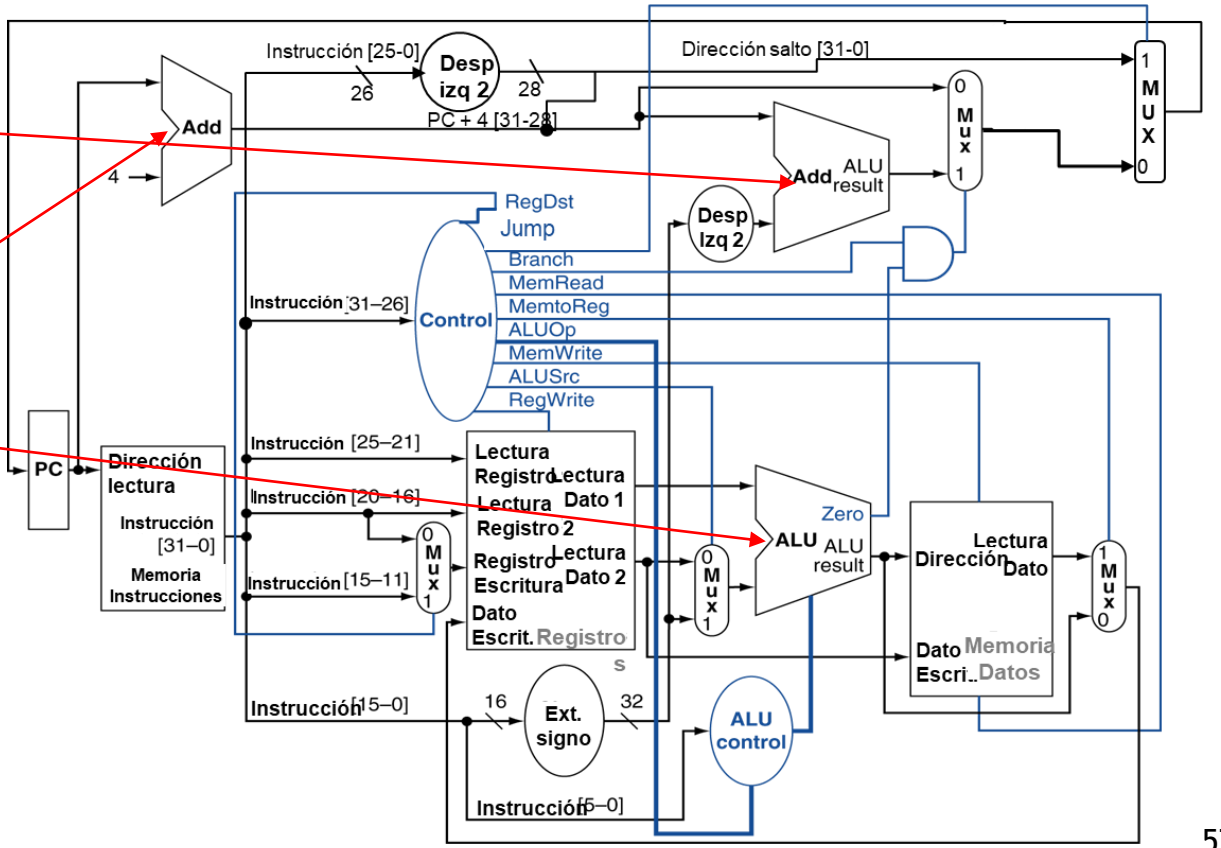


Ejercicio 6

- 0x1023000C --> beq $1, $3, 12
  - Mostra valores de entrada de la ALU y de las dos unidades de SUMA

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(b)	0	-16	-2	-3	4	-10	-6	-1	8	-4

ALU Add Salto condicional. Esta unidad de suma recibe el PC+4 por un lado. Por el otro, recibe los bits [15:0] de la instrucción extendidos y desplazados.	PC+4; 12*4
ALU Add +4. Esta unidad de control recibe el PC y un 4	PC; 4
ALU: La ALU de $1 y $3 para restarlos y así compararlos.	-16; -3



Ejercicio 6

- 0x1023000C --> beq $1, $3, 12
  - Mostrar valores de todas las entradas del BANCO de REGISTROS

	$0	$1	$2	$3	$4	$5	$6	$8	$12	$31
(b)	0	-16	-2	-3	4	-10	-6	-1	8	-4

RegWrite: como la instrucción beq no escribe en ningún registro, se desactiva.	0
Registro 1: uno de los registros a leer para la comparación, concretamente el $1, indicado en la instrucción[25:21] como un 1.	1
Registro 2: el otro de los registros para la comparación, concretamente el $3, indicado en la instrucción[20:16] como un 3.	3
Registro escritura: la instrucción beq no escribe sobre ningún registro, por lo que aquí no importa la entrada.	X
Dato escritura: la instrucción beq no escribe en los registros por lo que el dato no importa.	X