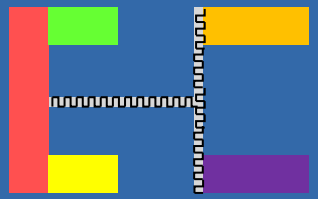


TEMA 4.
UNIDAD CENTRAL DE PROCESAMIENTO

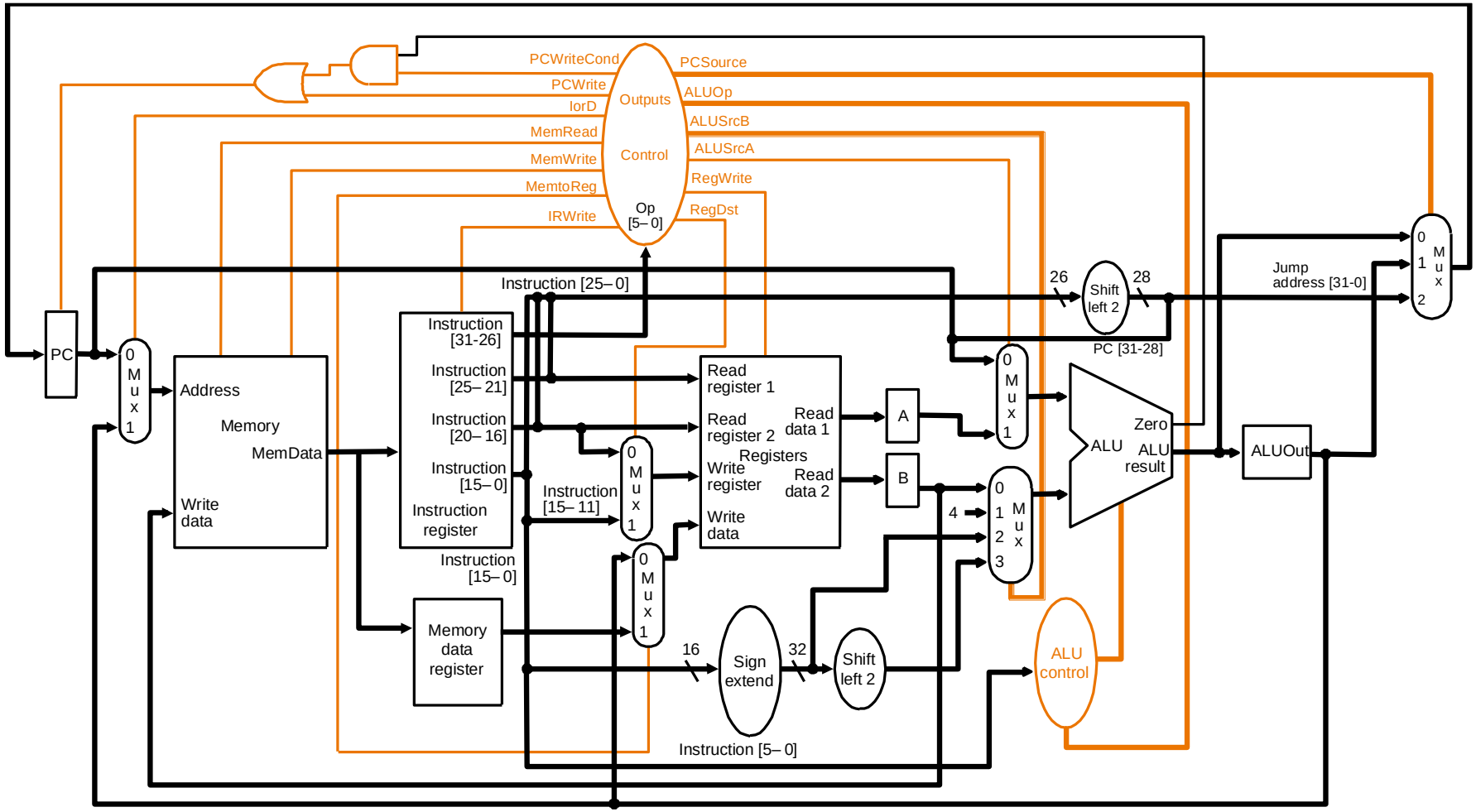
dtic

RESUMEN DE LAS FASES DE EJECUCIÓN

Nombre de la etapa o fase	Acción para instrucciones tipo R	Acción para instrucciones de referencia a memoria	Acción para saltos condicionales	Acción para saltos incondicionales
Búsqueda instrucción	RI ← Mem[PC] PC ← PC + 4			
Decodificación/ búsqueda de registros	A ← Reg [IR[25-21]] B ← Reg [IR[20-16]] SalidaALU ← PC + (extensión-signo (IR[15-0]) << 2)			
Ejecución, cálculo de la dirección/ finalización del salto	SalidaALU ← A op B	SalidaALU ← A + extensión-signo (RI[15-0])	Si (A ==B) entonces PC ← SalidaALU	PC ← PC [31-28] & (RI[25-0]<<2)
Acceso a memoria o Finalización tipo R	Reg [RI[15-11]] ← SalidaALU	**Lw:** MDR ←Mem [SalidaALU] o **Sw:** Mem [SalidaALU] ← B		
Finalización de la Lectura de memoria		**Lw:** Reg[RI[20-16]]= ←MDR		

RUTA DE DATOS COMPLETA Y CONTROL

Esquema de implementación multiciclo



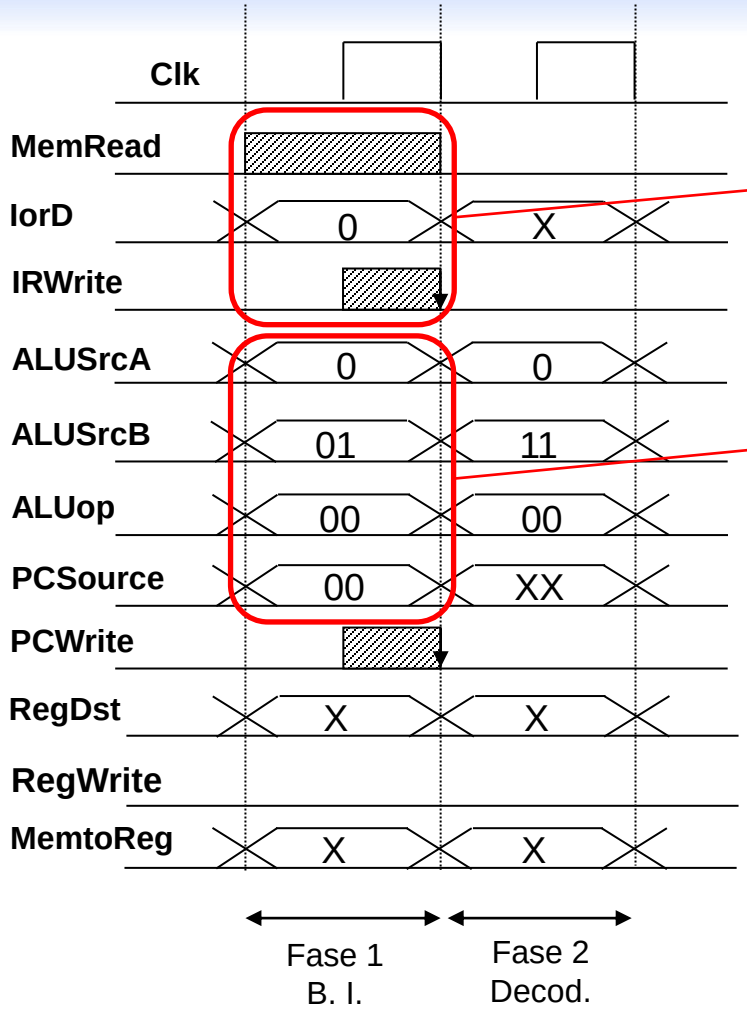
ACTIVACIÓN DE LAS SEÑALES DE CONTROL

Fases	Operación	Activación de señales
FASE 1		
	RI ← M[PC] PC ← PC + 4	MemRead, IorD=0, IRWrite ALUSrcA=0, ALUSrcB=01, ALUOp=00, PCSource=00, PCWrite
FASE 2		
	A ← Reg [IR[25-21]] B ← Reg [IR[20-16]] SalidaALU ← PC + (extensión-signo (IR[15-0]) << 2)	ALUSrcA=0, ALUSrcB=11, ALUOp=00
FASE 3		
LW, SW	SalidaALU ← A + extensión-signo (RI[15-0])	ALUSrcA=1, ALUSrcB=10, ALUOp=00
Tipo R	SalidaALU ← A op B	ALUSrcA=1, ALUSrcB=00, ALUOp=10
BEQ	Si (A ==B) entonces PC ← SalidaALU	ALUSrcA=1, ALUSrcB=00, ALUOp=01, PCWriteCond=1, PCSource=01
J	PC ← PC [31-28] & (RI[25-0]<<2)	PCWrite, PCSource=10
FASE 4		
LW	MDR ←Mem [SalidaALU]	MemRead, IorD=1
SW	Mem [SalidaALU] ← B	MemWrite, IorD=1
Tipo R	Reg [RI[15-11]] ← SalidaALU	RegDst=1, RegWrite, MemtoReg=0
FASE 5		
LW	Reg[RI[20-16]]= ←MDR	RegDst=0, RegWrite, MemtoReg=1

CRONOGRAMA INSTRUCCIÓN
EJERCICIO 1

- Hacer el cronograma de los distintos tipos de instrucciones **lw**, **sw**, **tipo-R**, **beq**, **j**, y **slt**.

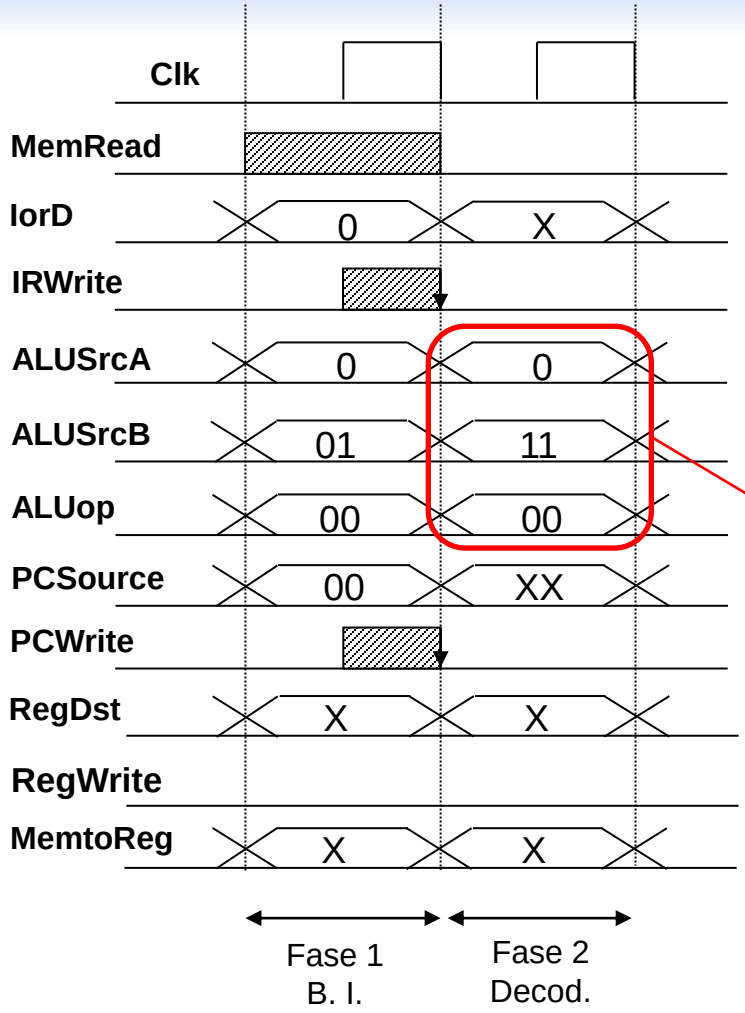
CRONOGRAMA INSTRUCCIÓN
Fases comunes



La fase 1:

- busqueda de la instrucción. Se lee de la memoria la dejando pasar la dirección de la memoria a través de IorD y se almacenan los datos en los registros.
- preparación de la siguiente. Para "adelantar trabajo" aumentamos el PC a PC+4 dejando pasar el valor actual de PC con ALUsrcA, el valor 4 por ALUsrcB , la operación suma con ALUop y el resultado se devuelve a través del Mx con PCSource. En caso de que la instrucción sea un salto, este valor se sobreescribirá posteriormente.

CRONOGRAMA INSTRUCCIÓN
Fases comunes

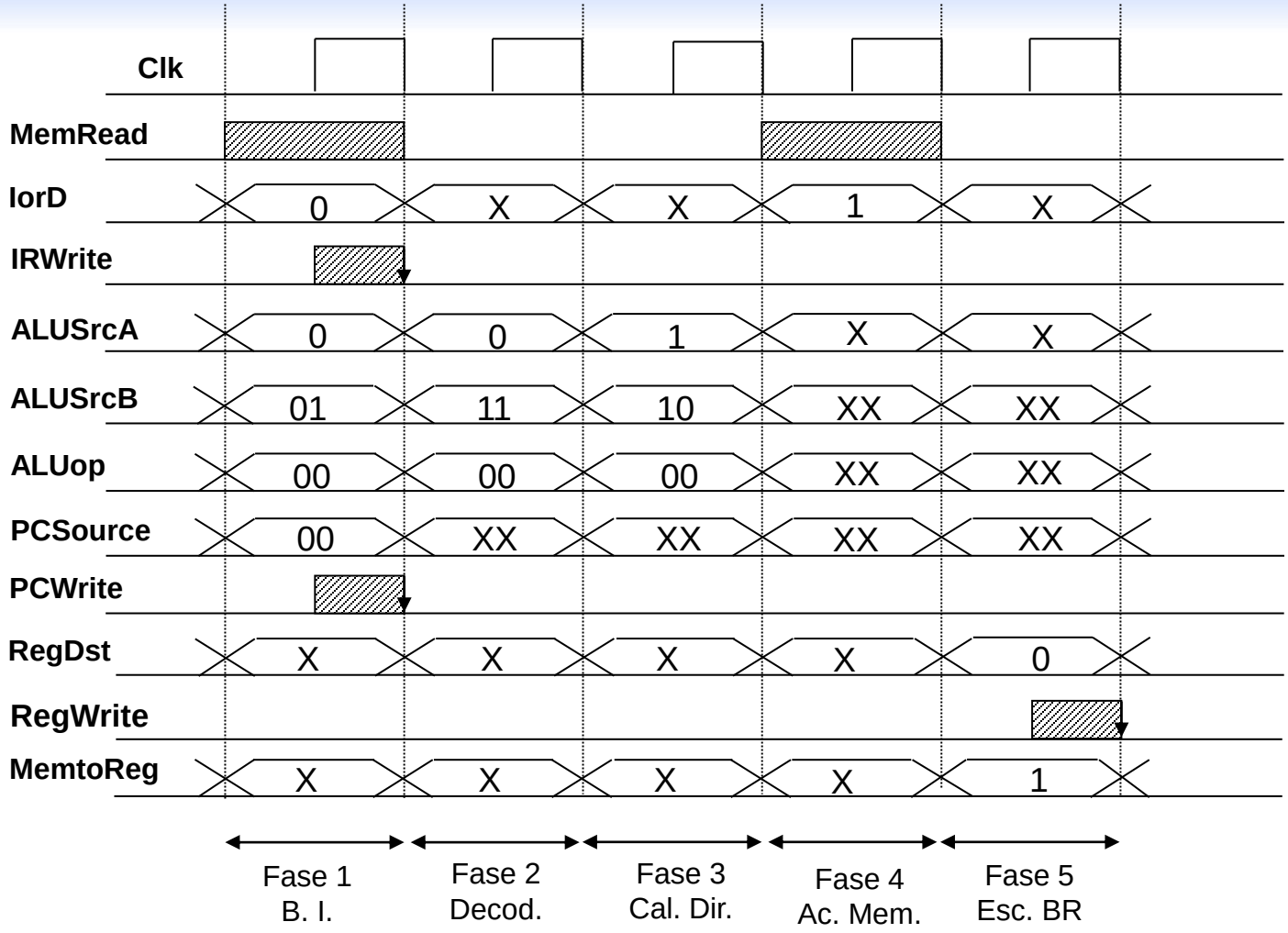


La fase 2:

- Decodificación: esta tarea ocurre internamente en la Unidad de Control, donde decodifica la instrucción para saber qué señales se han de activar.
- Búsqueda de registros: se obtienen los valores de los registros que necesita la instrucción para su ejecución.
- Calculo de la dirección de memoria: Esta es una optimización de MIPS que permite calcula la dirección de salto por si la instrucción fuse de salto condicional.

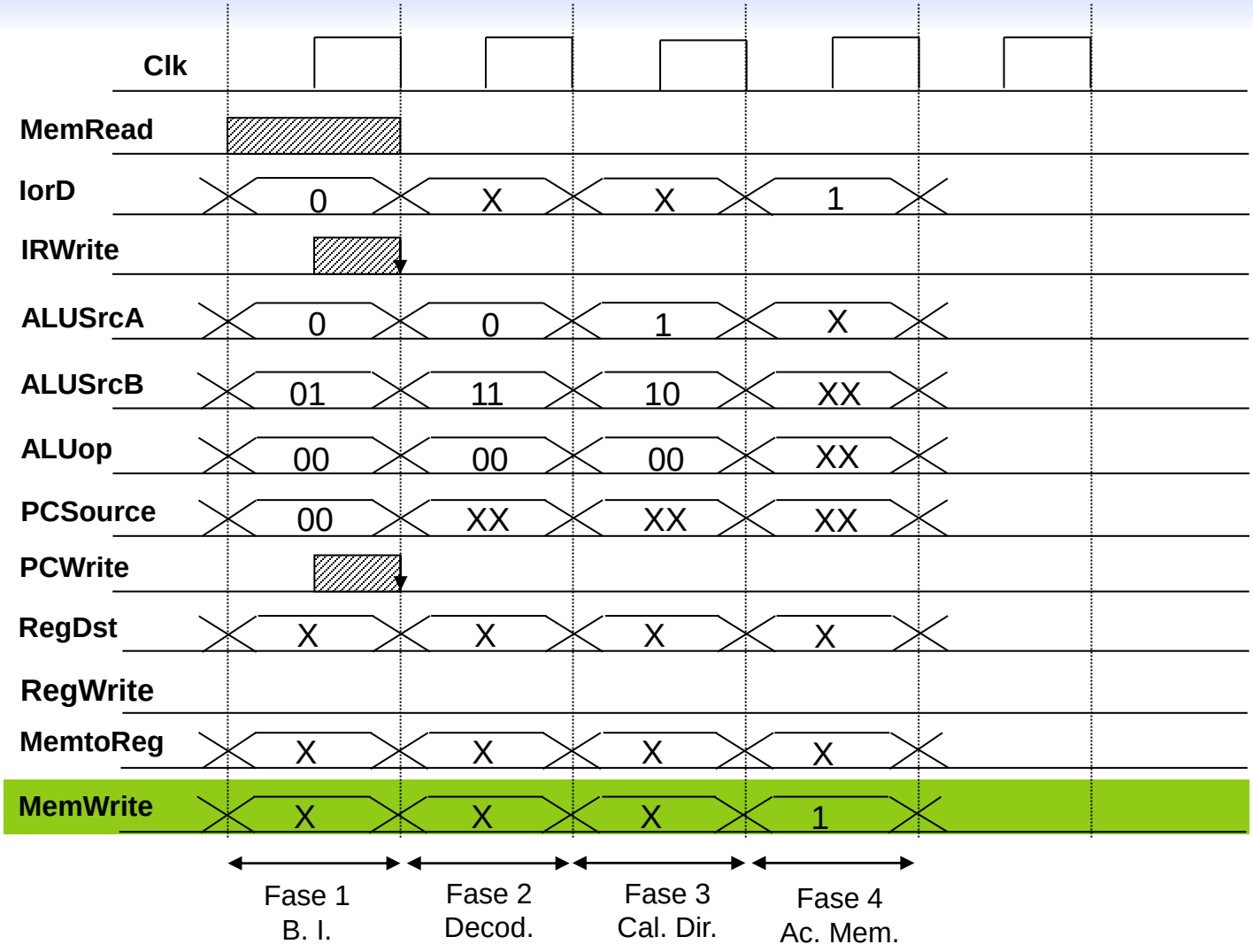
CRONOGRAMA INSTRUCCIÓN Lw

Ejercicios



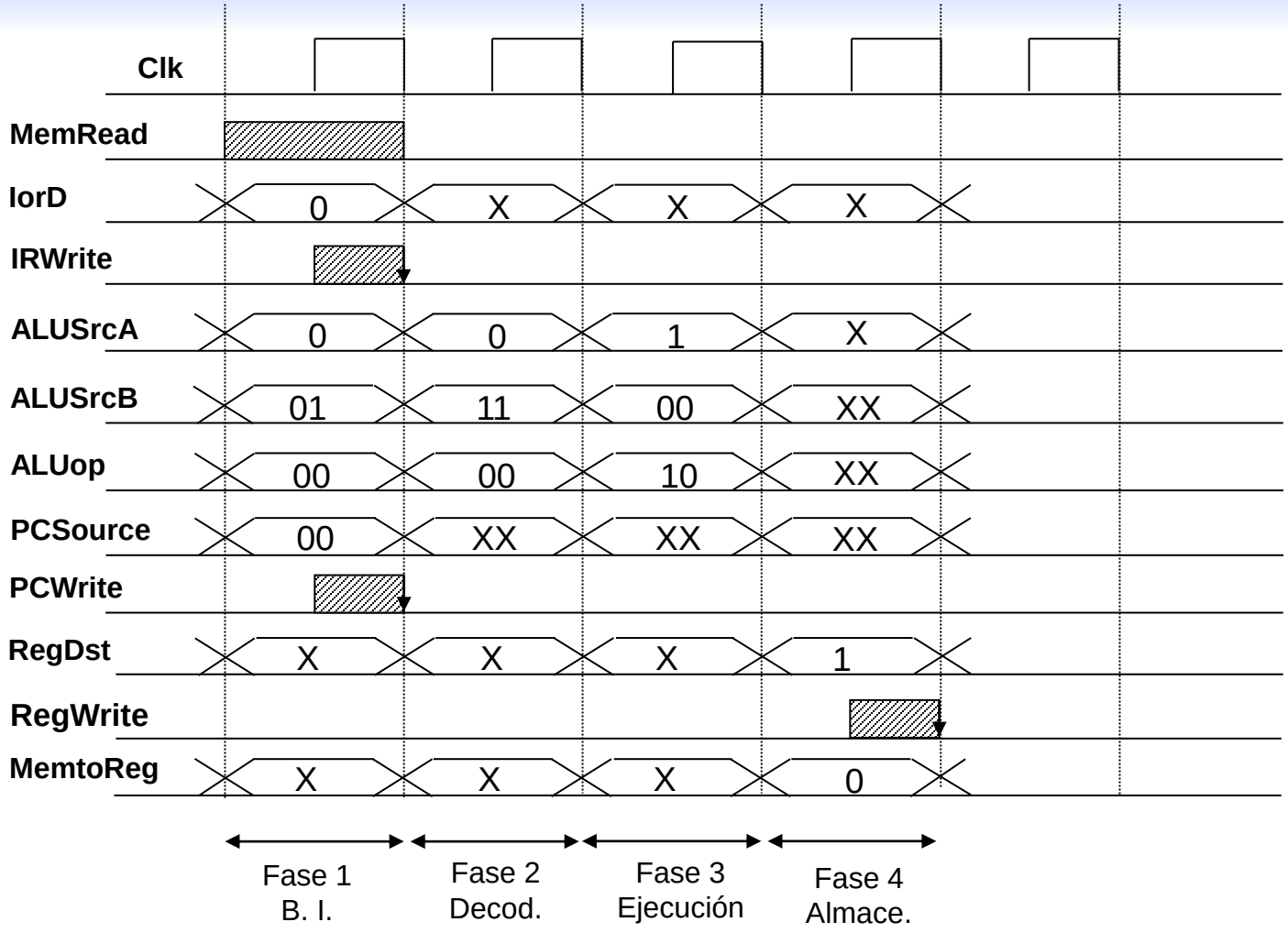
CRONOGRAMA INSTRUCCIÓN sw

Ejercicios



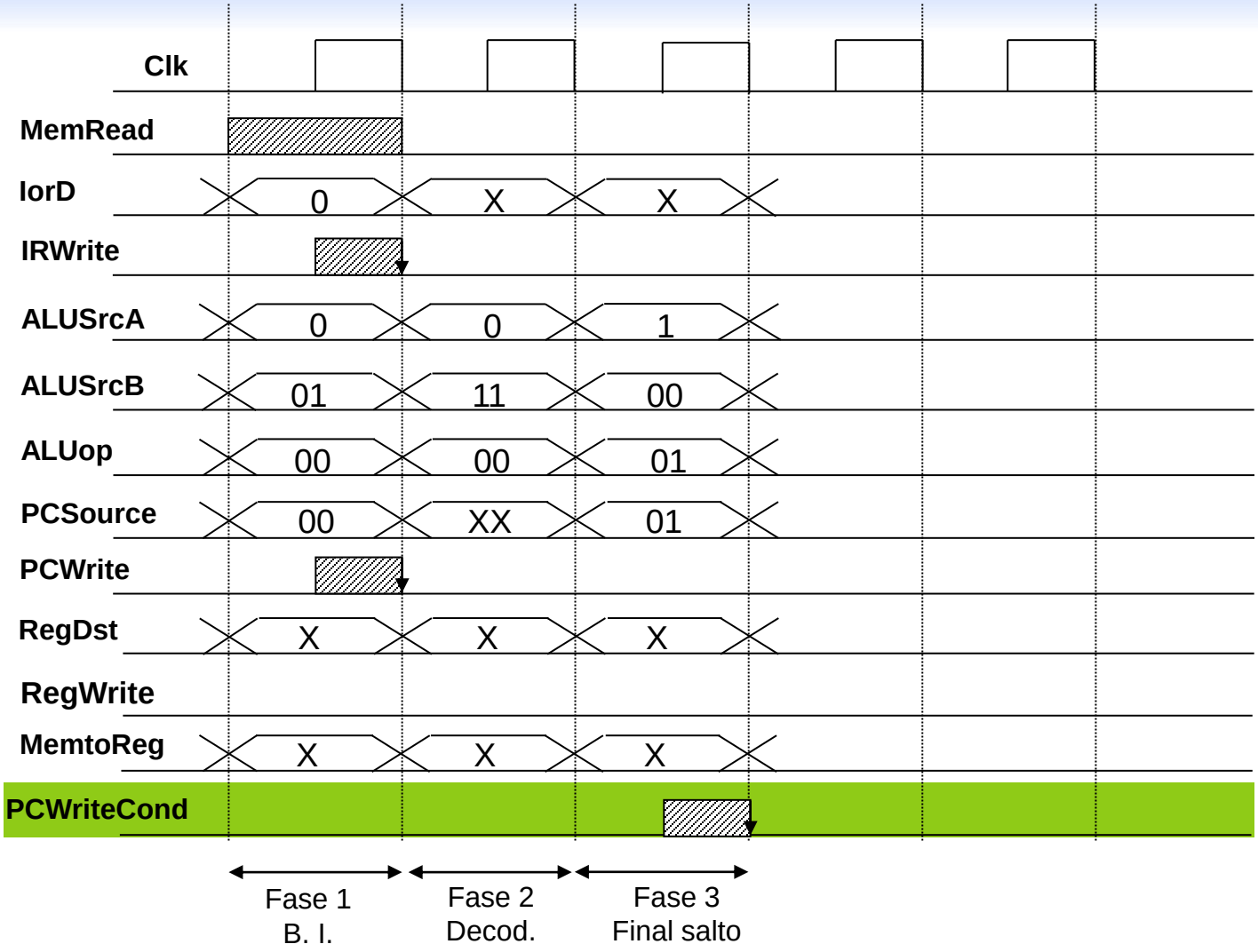
CRONOGRAMA INSTRUCCIÓN Tipo-R

Ejercicios



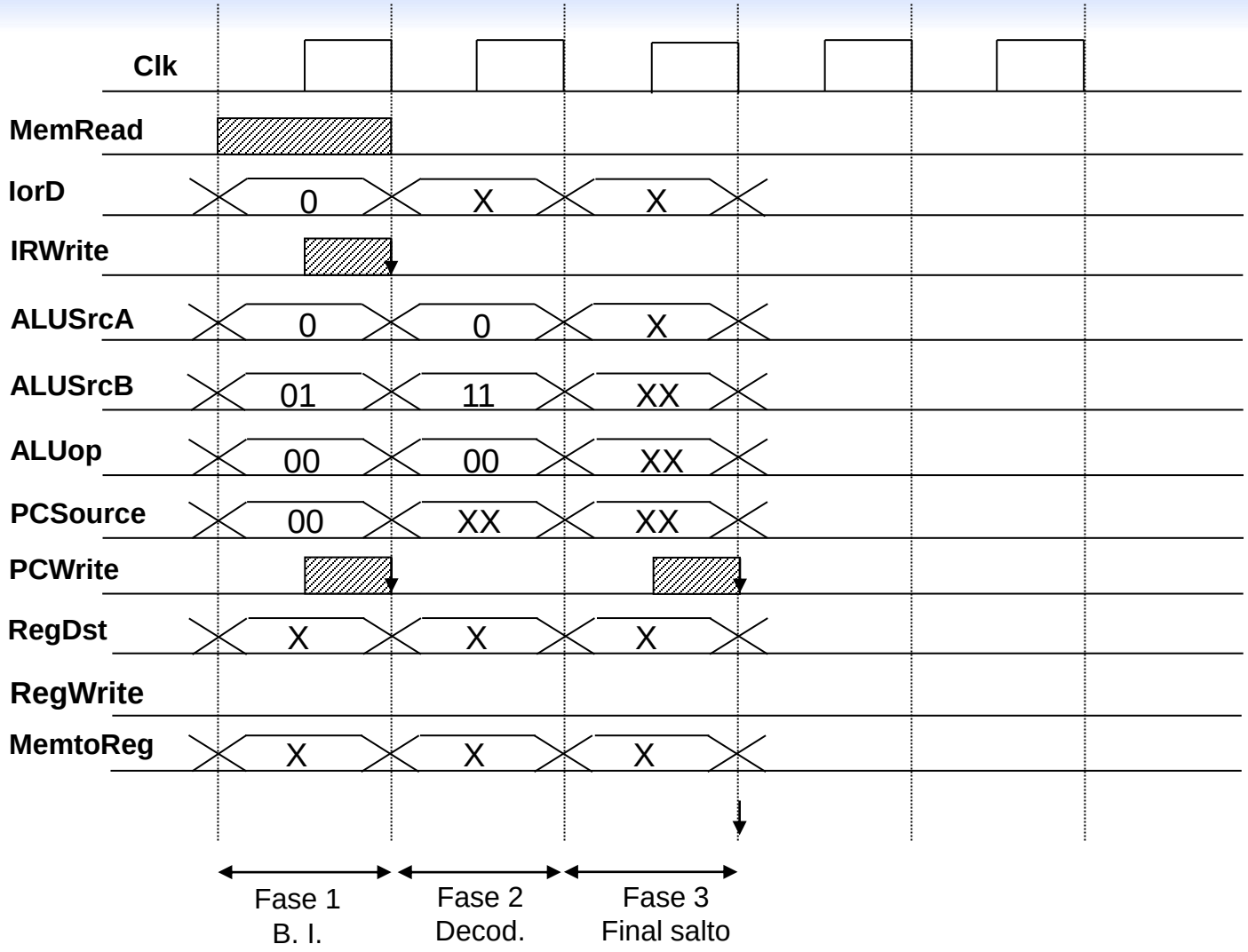
CRONOGRAMA INSTRUCCIÓN salto condicional (beq)

Ejercicios

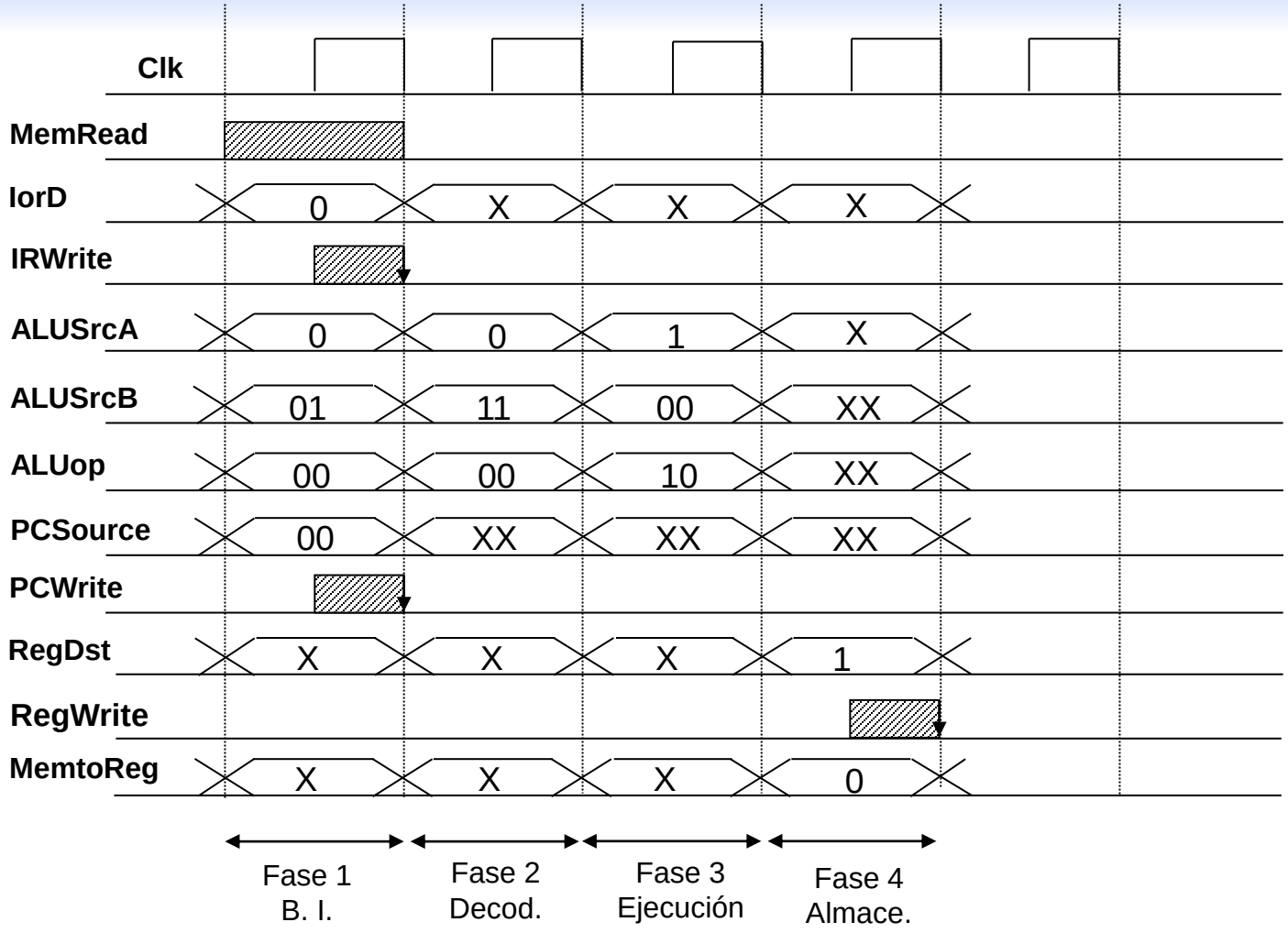


CRONOGRAMA INSTRUCCIÓN salto incondicional (j)

Ejercicios



CRONOGRAMA INSTRUCCIÓN
slt $1,$2,$3(si $2<$3->$1=1; sino $1=0)



EJERCICIO 2

- Cuántos ciclos se necesitan para ejecutar este código?

```
        lw $t2, 0($t3)
        lw $t3, 4($t3)
        beq $t2, $t3, Label  <-- #suponer no
        add $t5, $t2, $t3
        sw $t5, 8($t3)
Label:  ...
```

- ¿Qué está ocurriendo durante el octavo ciclo de ejecución?
- ¿En qué ciclo ocurre la suma de `$t2 y $t3`?
- ¿Cuánto tarda el programa si la CPU es de 100Mhz?

EJERCICIO 2

- Cuántos ciclos se necesitan para ejecutar este código?

```
    (5)  lw $t2, 0($t3)
    (5)  lw $t3, 4($t3)
    (3)  beq $t2, $t3, Label  <-- #suponer no
    (4)  add $t5, $t2, $t3
    (4)  sw $t5, 8($t3)
Label:   ...
```

- Como sabemos, las instrucciones tienen dos fases en común, es decir, dos ciclos. Las instrucciones **beq** y **j** finalizan en la tercera fase con el salto.

  Las instrucciones **sw** y las de **tipo R** finalizan en la fase 4 al escribir en memoria (**sw**) o registro (**tipo R**).

  Por último, en la instrucción **lw** tiene 5 fases.

Fases	Operación
FASE 1	
	RI ← M[PC] PC ← PC + 4
FASE 2	
	A ← Reg [IR[25-21]] B ← Reg [IR[20-16]] SalidaALU ← PC + (extensión-signo (IR[15-0]) << 2)
FASE 3	
LW, SW	SalidaALU ← A + extensión-signo (RI[15-0])
Tipo R	SalidaALU ← A op B
BEQ	Si (A ==B) entonces PC ← SalidaALU
J	PC ← PC [31-28] & (RI[25-0]<<2)
FASE 4	
LW	MDR ←Mem [SalidaALU]
SW	Mem [SalidaALU] ← B
Tipo R	Reg [RI[15-11]] ← SalidaALU
FASE 5	
LW	Reg[RI[20-16]]= ←MDR

EJERCICIO 2

- Cuántos ciclos se necesitan para ejecutar este código?

```
(5)  lw $t2, 0($t3)
(5)  lw $t3, 4($t3)
(3)  beq $t2, $t3, Label  <-- #suponer no
(4)  add $t5, $t2, $t3
(4)  sw $t5, 8($t3)
Label:  ...
```

- El total de ciclos necesarios para ejecutar el código es **21 ciclos**. Si la instrucción **beq** hubiera sido efectiva, el salto hubiera hecho que el total de ciclos fuse 13.

Fases	Operación
FASE 1	
	RI ← M[PC] PC ← PC + 4
FASE 2	
	A ← Reg [IR[25-21]] B ← Reg [IR[20-16]] SalidaALU ← PC + (extensión-signo (IR[15-0]) << 2)
FASE 3	
LW, SW	SalidaALU ← A + extensión-signo (RI[15-0])
Tipo R	SalidaALU ← A op B
BEQ	Si (A ==B) entonces PC ← SalidaALU
J	PC ← PC [31-28] & (RI[25-0]<<2)
FASE 4	
LW	MDR ←Mem [SalidaALU]
SW	Mem [SalidaALU] ← B
Tipo R	Reg [RI[15-11]] ← SalidaALU
FASE 5	
LW	Reg[RI[20-16]]= ←MDR

EJERCICIO 2

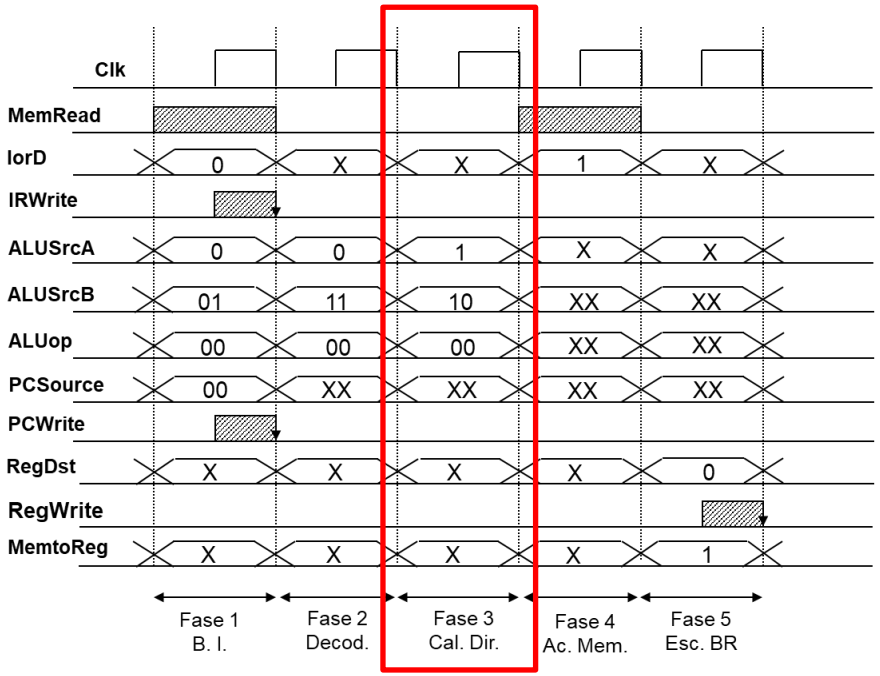
- Cuántos ciclos se necesitan para ejecutar este código?

```
        lw $t2, 0($t3)
        lw $t3, 4($t3)
        beq $t2, $t3, Label  <-- #suponer no
        add $t5, $t2, $t3
        sw $t5, 8($t3)
Label:  ...
```

- ¿Qué está ocurriendo durante el octavo ciclo de ejecución?

EJERCICIO 2

- ¿Qué está ocurriendo durante el octavo ciclo de ejecución?
  - El octavo ciclo pertenece a la segunda instrucción de carga (**lw**). Concretamente al tercer ciclo de la segunda instrucción, ya que la primera instrucción ocupa 5 ciclos, más 3 de la segunda.



Fases	Operación
FASE 1	
	RI ← M[PC] PC ← PC + 4
FASE 2	
	A ← Reg [IR[25-21]] B ← Reg [IR[20-16]] SalidaALU ← PC + (extensión-signo (IR[15-0]) << 2)
FASE 3	
LW, SW	SalidaALU ← A + extensión-signo (RI[15-0])
Tipo R	SalidaALU ← A op B
BEQ	Si (A ==B) entonces PC ← SalidaALU
J	PC ← PC [31-28] & (RI[25-0]<<2)
FASE 4	
LW	MDR ←Mem [SalidaALU]
SW	Mem [SalidaALU] ← B
Tipo R	Reg [RI[15-11]] ← SalidaALU
FASE 5	
LW	Reg[RI[20-16]]= ←MDR

EJERCICIO 2

- Cuántos ciclos se necesitan para ejecutar este código?

```
        lw $t2, 0($t3)
        lw $t3, 4($t3)
        beq $t2, $t3, Label  <-- #suponer no
        add $t5, $t2, $t3
        sw $t5, 8($t3)
Label:  ...
```

- ¿Qué está ocurriendo durante el octavo ciclo de ejecución?
- ¿En qué ciclo ocurre la suma de `$t2` y `$t3`?

EJERCICIO 2

- ¿En qué ciclo ocurre la suma de `$t2 y $t3`?
  - La instrucción **add** es la cuarta. Antes ha habido dos **lw** y una **beq**, lo que suman 5(lw) + 5(lw) + 3(beq) , 13 ciclos.
  - En las instrucciones de tipo R la operación se realiza en la fase 3.
  - Por lo tanto, la suma se realiza en el ciclo 16.
  - La operación completa finaliza en el ciclo 17.

Fases	Operación
FASE 1	
	RI ← M[PC] PC ← PC + 4
FASE 2	
	A ← Reg [IR[25-21]] B ← Reg [IR[20-16]] SalidaALU ← PC + (extensión-signo (IR[15-0]) << 2)
FASE 3	
LW, SW	SalidaALU ← A + extensión-signo (RI[15-0])
Tipo R	SalidaALU ← A op B
BEQ	Si (A ==B) entonces PC ← SalidaALU
J	PC ← PC [31-28] & (RI[25-0]<<2)
FASE 4	
LW	MDR ←Mem [SalidaALU]
SW	Mem [SalidaALU] ← B
Tipo R	Reg [RI[15-11]] ← SalidaALU
FASE 5	
LW	Reg[RI[20-16]]= ←MDR

EJERCICIO 2

- Cuántos ciclos se necesitan para ejecutar este código?

```
        lw $t2, 0($t3)
        lw $t3, 4($t3)
        beq $t2, $t3, Label  <-- #suponer no
        add $t5, $t2, $t3
        sw $t5, 8($t3)
Label:  ...
```

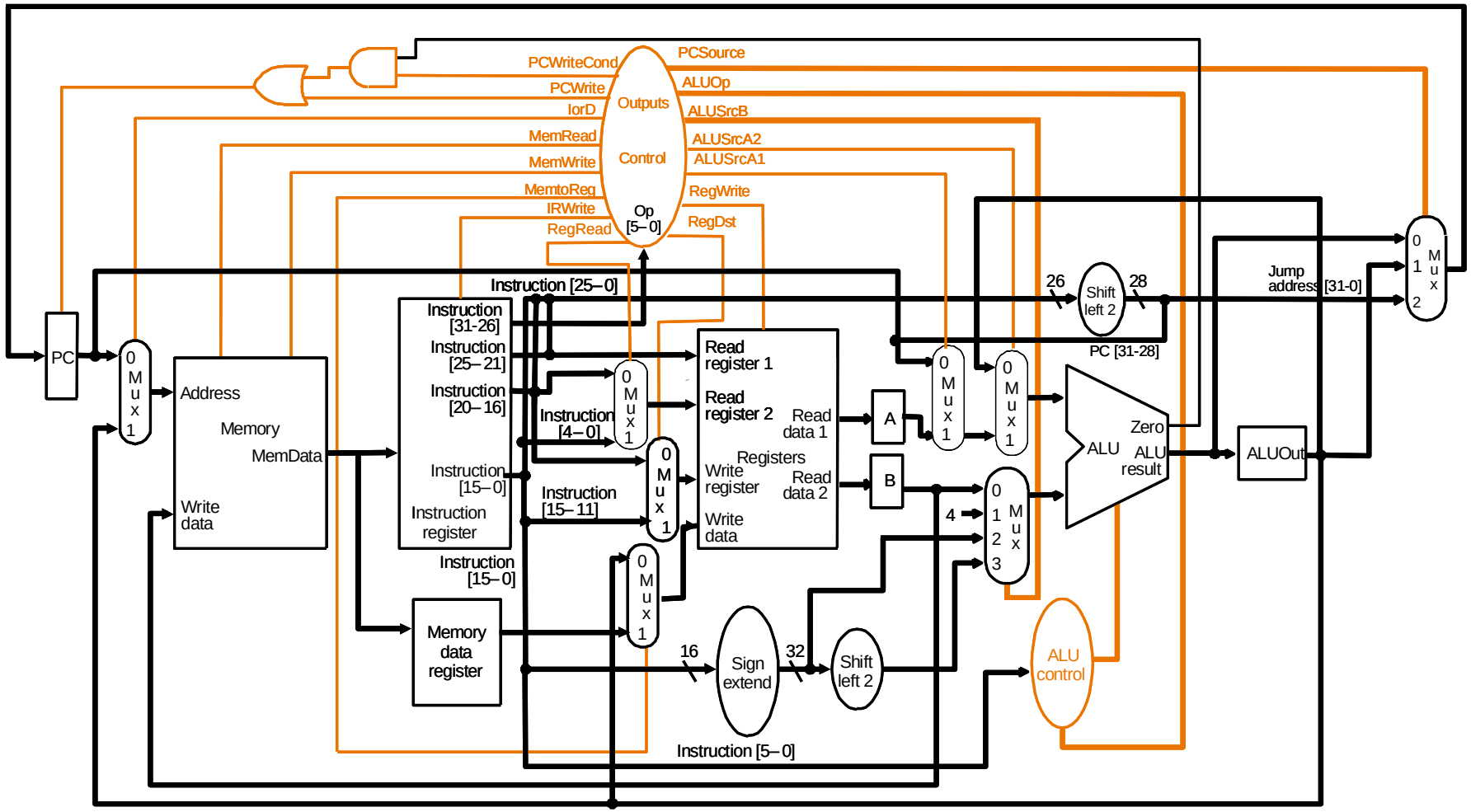
- ¿Qué está ocurriendo durante el octavo ciclo de ejecución?
- ¿En qué ciclo ocurre la suma de `$t2` y `$t3`?
- ¿Cuánto tarda el programa si la CPU es de 100Mhz?

EJERCICIO 2

- ¿Cuánto tarda el programa si la CPU es de 100Mhz?
  - $f = 100MHz \rightarrow T = \frac{1}{100 \times 10^6} seg = 0.1ns$
  - Si 1 ciclo tarda 0.1ns, 21 ciclos tardarán **2.1ns**

EJERCICIO 3

Ejercicios



EJERCICIO 3

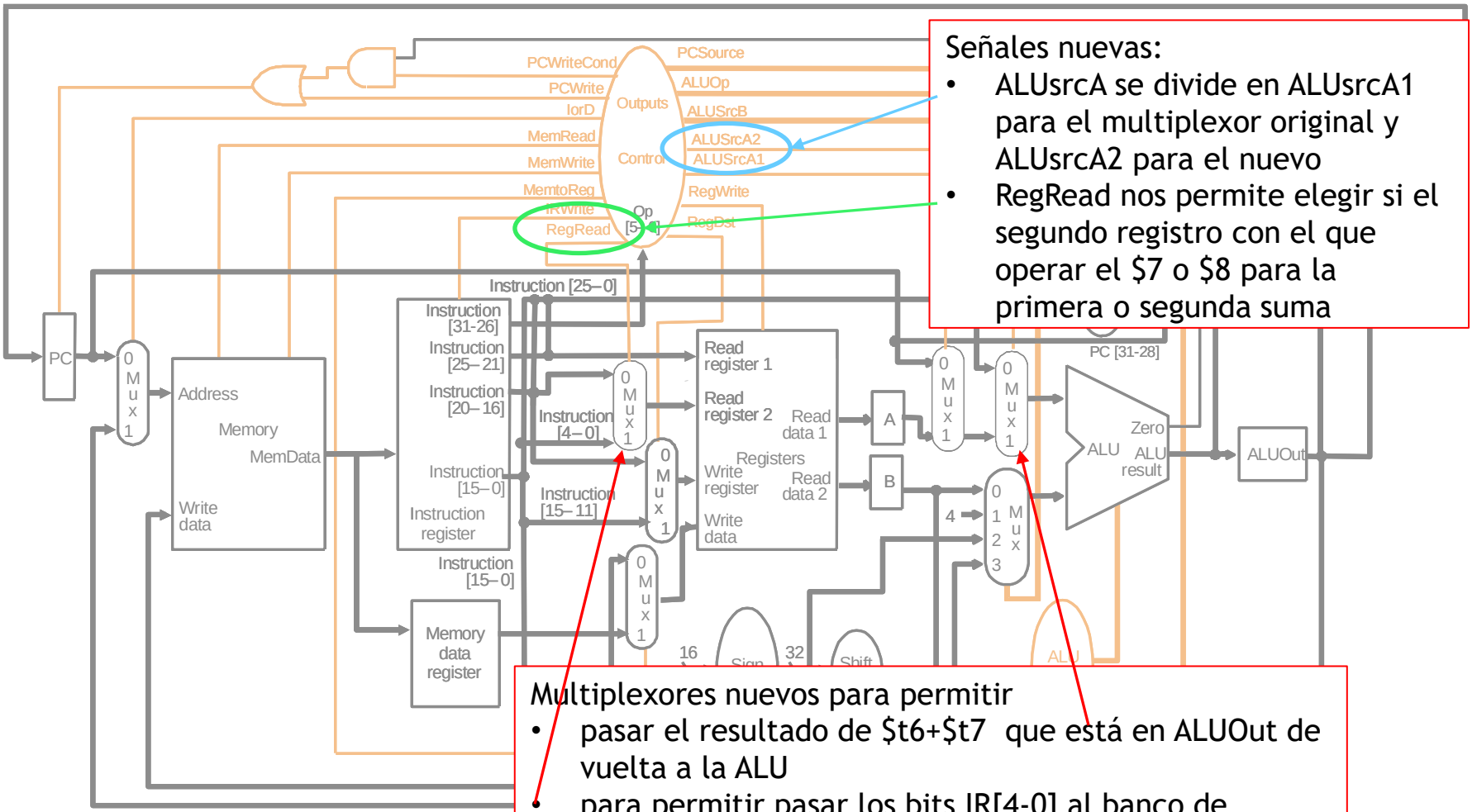
- Se ha modificado ligeramente la ruta de datos del MIPS como se muestra en la figura para que se pueda ejecutar la siguiente instrucción nueva:

  add3 $t5, $t6, $t7, $t8 # $t5←$t6+$t7+$t8

- Suponer que se añade un nuevo formato de instrucción similar al R con la diferencia que los bits [0-4] se utilizan para especificar el registro fuente adicional, además se añade un nuevo código de operación para esta instrucción.

- Suponed que en esta nueva ruta de datos se ejecutan también las instrucciones aritmético-lógicas (**add, sub, and, or y slt**) con formato tipo R y las instrucciones **lw, sw y beq** con formato tipo I.

EJERCICIO 3



Señales nuevas:

- ALUsrcA se divide en ALUsrcA1 para el multiplexor original y ALUsrcA2 para el nuevo
- RegRead nos permite elegir si el segundo registro con el que operar el $7 o $8 para la primera o segunda suma

Multiplexores nuevos para permitir

- pasar el resultado de $t6+$t7 que está en ALUOut de vuelta a la ALU
- para permitir pasar los bits IR[4-0] al banco de registros para obtener el valor de $8 para la segunda suma

EJERCICIO 3

- Se pide:

a) Obtened las acciones a realizar en cada ciclo de reloj mediante lenguaje de transferencia de registros (por ejemplo: PC ← PC +4) de las instrucciones **add3**, **add, lw y beq**. Las instrucciones deben ejecutarse en el **menor número posible** de ciclos de reloj.

b) Obtened el valor de las señales de control que se activan en cada ciclo de reloj para las instrucciones **add3, add, lw y beq**. Suponed que al inicio de cada ciclo de reloj todas señales de control tienen el valor 0 (están desactivadas).

c) Si la frecuencia de reloj del procesador es de 100GHz, ¿Cuánto tarda en ejecutarse cada instrucción?

EJERCICIO 3 - SOLUCIÓN

- Apartados a) y b)
  - Como la instrucción **add3** requiere utilizar la ALU 2 veces primero sumando $t6+$t7 y luego sumando $8 al resultado de lo anterior, se ha añadido una nueva entrada al multiplexor de entrada de la ALU para que pueda sumarse el valor almacenado en ALUout.
  - Se necesita también volver a leer el banco de registros, por lo que se requiere un multiplexor y una nueva señal de control para la entrada de uno de los puertos de lectura del registro 3.
  - La lectura del tercer registro fuente se ha implementado durante el ciclo de reloj en el que se realiza la primera suma.

EJERCICIO 3 - SOLUCIÓN

Ejercicios

Ciclos	Operación (apartado a)	Activación de señales (apartado b)
Ciclo 1		
	RI ← Memoria[PC] PC ← PC + 4	MemRead, IorD=0, IRWrite, ALUSrcA1=0, ALUSrcA2=1, ALUSrcB=01, ALUOp=00, PCSource=00, PCWrite=1
Ciclo 2		
	Decodificación A ← Reg[RI[25-21]] B ← Reg[RI[20-16]] ALUOut ←PC+(extensión-signo(RI[15-0])<< 2)	RegRead=0, ALUSrcA1=0, ALUSrcA2=1 ALUSrcB=11, ALUOp=00
Ciclo 3		
Add3	ALUOut ← A + B B ←Reg[RI[4-0]]	RegRead=1, ALUSrcA1=1, ALUSrcA2=1 ALUSrcB=00, ALUOp=10
Add	ALUOut ← A + B	ALUSrcA1=1, ALUSrcA2=1 ALUSrcB=00, ALUOp=10
Lw	ALUOut ← A + extensión-signo (IR[15-0]);	ALUSrcA1=1, ALUSrcA2=1 ALUSrcB=11, ALUOp=00
Beq	Si (A==B) PC ← SalidaALU	ALUSrcA1=1, ALUSrcA2=1, ALUSrcB=00, ALUOp=10, PCWriteCond=1, PCSource=01
Ciclo 4		
Add3	ALUOut ← ALUOut + B	ALUSrcA2=0, ALUSrcB=00, ALUOp=10
Add	Reg[RI[15-11]] ← ALUOut	RegDst=1, RegWrite=1, MemtoReg=0
Lw	MDR ← Mem [ALUOut]	MemRead=1, IorD=1
Ciclo 5		
Add3	Reg[RI[15-11]] ← ALUOut	RegDst=1, RegWrite=1, MemtoReg=0
Lw	Reg[IR[20-16]]← MDR	RegDst=0, RegWrite=1, MemtoReg=1

EJERCICIO 3 - SOLUCIÓN

- Apartados c)

$$f = 100GHz \rightarrow T = \frac{1}{100 \times 10^{+9}} seg = 10ps$$

- Las instrucciones **add3** y **lw** tardan duran 5 ciclos

  5x10ps=50ps en ejecutarse

- La instrucción **add** tarda 4 ciclos

  4x10ps=40 ps en ejecutarse

- La instrucción **beq** tarda 3 ciclos

  3x10ps=30ps en ejecutarse

EJERCICIO 4

- Se desea que la ruta de datos multiciclo del MIPS ejecute una nueva instrucción de tipo I denominada Sm (saltar a memoria):

$$PC \leftarrow M[Rf1 + Rf2 + Desplazamiento]$$

- Se pide:

a) Modificad la ruta de datos multiciclo de la figura de tal manera que se pueda ejecutar esta nueva instrucción. Suponed que en esta nueva ruta de datos se ejecutan también las instrucciones aritmético-lógicas (**add**, **sub**, **and**, **or** y **slt)** con formato tipo R y las intrucciones **lw**, **sw** y **beq** con formato tipo I.

b) Obtened las acciones a realizar en cada ciclo de reloj mediante lenguaje de transferencia de registros (por ejemplo: PC ← PC +4) de la instrucción **Sm** y de las instrucciones **add, lw y beq**. Las instrucciones deben ejecutarse en el menor número posible de ciclos de reloj.

c) Obtened el valor de las señales de control que se activan en cada ciclo de reloj para la instrucción **Sm** y para las instrucciones **add, lw y beq**. Suponed que al inicio de cada ciclo de reloj todas señales de control tienen el valor 0 (están desactivadas).

d) Rellena la tabla de salida donde se muestren el valor de las señales de control en cada ciclo de reloj para la instrucción Sm.

EJERCICIO 4

PC ← M[Rf1 + Rf2 + Desplazamiento]

- Primero debemos analizar la instrucción. Esta instrucción tiene dos registros y un desplazamiento, clásico esquema de instrucción tipo I.

- Necesitamos realizar las acciones siguientes:
  - Sumar los dos registros Rf1 y Rf2
  - Sumar el resultado al valor inmediato que viene en los últimos 16 bits
  - Buscar en memoria el valor
  - Pasar la nueva dirección leída de memoria al PC

EJERCICIO 4

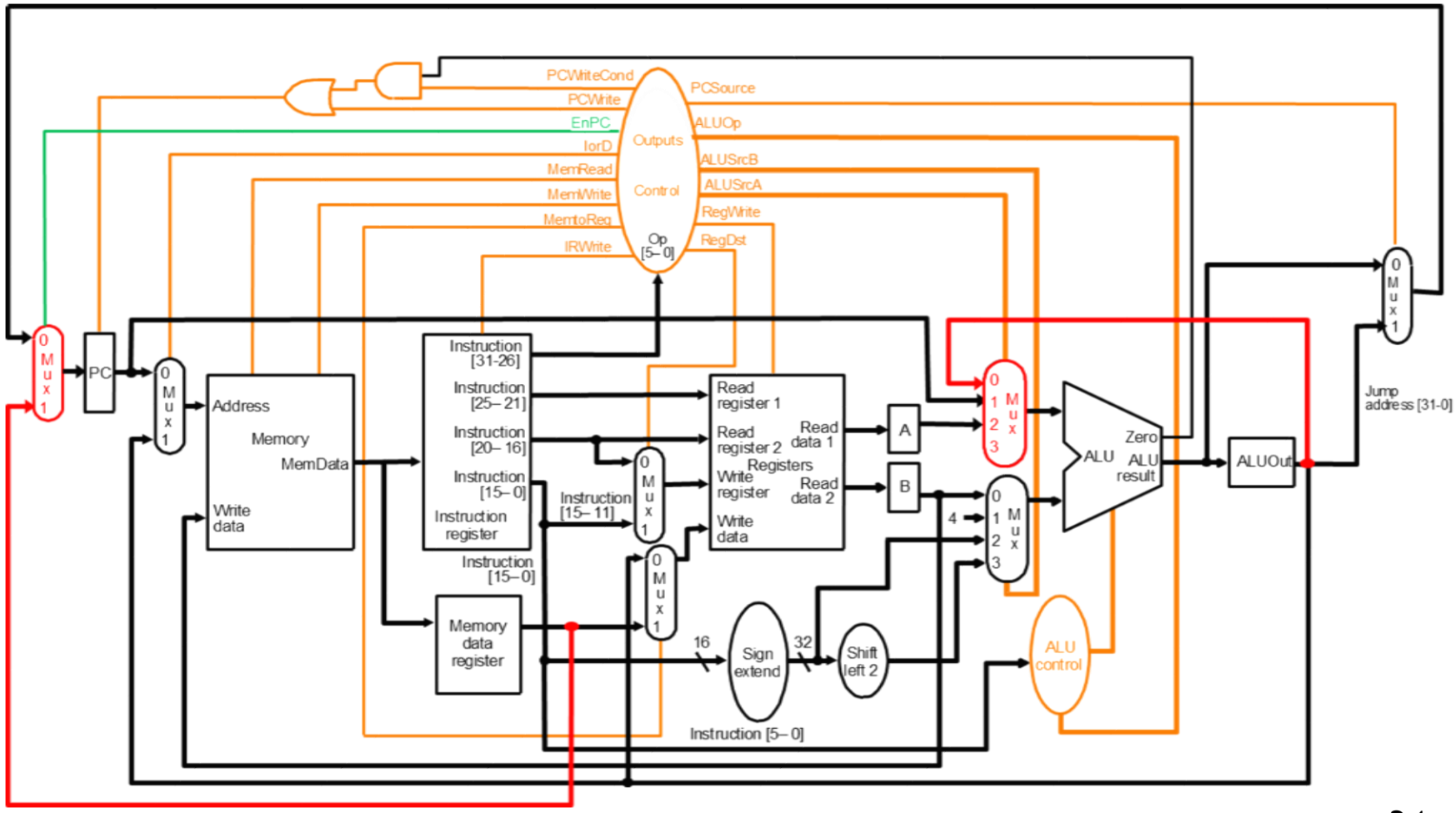
- Necesitamos realizar las acciones siguientes:
  - Sumar los dos registros Rf1 y Rf2. Esta es una suma normal que se puede realizar con el esquema multiciclo.
  - Sumar el resultado al valor inmediato que viene en los últimos 16 bits. Para esta segunda suma necesitamos recuperar el resultado de la suma anterior y pasarla a la ALU. Por lo tanto, la ALU debe tener como entradas de este paso el valor inmediato con el signo extendido y el resultado de la anterior suma que se encuentra en ALUOut. Para poder elegir la entrada a la ALU entre el registro leído (operación de suma anterior) o el resultado encontrado en ALUout (para esta segunda suma) necesitamos que el multiplexor ALUsrcA se amplíe con más entradas o utilizar dos multiplexores en serie (como en el ejercicio 3).
  - Buscar en memoria el valor. Una vez se tenga la dirección de memoria a buscar hay que acceder a la memoria como se hace en el cuarto ciclo de una instrucción **lw**. Este valor se queda almacenado en MDR (Memory Data Register).
  - Pasar la nueva dirección leída de memoria al PC. Para esto tenemos que dejar pasar al PC o la dirección calculada de forma normal o la de esta nueva instrucción **sm**. Por lo tanto, debemos añadir un multiplexor y su señal para poder elegir.

EJERCICIO 4 - SOLUCIÓN

- Apartado a)
  - Para que se pueda ejecutar la nueva instrucción **Sm** se propone incluir un multiplexor a la entrada del PC para que una de las entradas sea el dato procedente de MDR
  - Hay que ampliar el primer multiplexor de la ALU para que acepte el dato de salida de la ALU tal como muestra la figura.
  - Finalmente, se ha añadido una nueva señal de control llamada EnPC.

EJERCICIO 4 - SOLUCIÓN

- Apartado a)



Apartados b) y c)

Ciclos	Operación (apartado b)	Activación de señales (apartado c)
Ciclo 1		
	RI ← Memoria[PC] PC ← PC + 4	MemRead, IorD=0, IRWrite, **ALUSrcA=01**, ALUSrcB=01, ALUOp=00, PCSource=0, EnPC=0, PCWrite=1
Ciclo 2		
	Decodificación A ← Reg[RI[25-21]] B ← Reg[RI[20-16]] ALUOut ←PC+(extensión-signo(RI[15-0])<< 2)	RegRead=0, **ALUSrcA=01**, ALUSrcB=11, ALUOp=00
Ciclo 3		
Add/Sm	ALUOut ← A + B	**ALUSrcA=10,** ALUSrcB=00, ALUOp=10
Lw/Sw	ALUOut ← A + extensión-signo (IR[15-0])	**ALUSrcA=10**, ALUSrcB=10, ALUOp=00
Beq	Si (A==B) PC ← SalidaALU	**ALUSrcA=10**, ALUSrcB=00, ALUOp=01, EnPC=0, PCWriteCond=1, PCSource=01
Ciclo 4		
Sm	ALUOut ← ALUOut + extensión-signo (IR[15-0])	**ALUSrcA=00**, ALUSrcB=10, ALUOp=00
Add	Reg[RI[15-11]] ← ALUOut	RegDst=1, RegWrite=1, MemtoReg=0
Lw	MDR ← Mem [ALUOut]	MemRead=1, IorD=1
Sw	Mem [ALUOut] ←B	MemWrite=1, IorD=1
Ciclo 5		
Sm	MDR ← Mem [ALUOut]	MemRead=1, IorD=1
Lw	Reg[IR[20-16]]← MDR	RegDst=0, RegWrite=1, MemtoReg=1
Ciclo 6		
Sm	PC ← MDR	EnPC=1, PCWrite

Señales nuevas

- ALUscrA de dos bits
- EnPC

EJERCICIO 4 - SOLUCIÓN

- Apartado d)

Señales de control	Ciclos de reloj					
	1	2	3	4	5	6
PCWriteCond	0	0	0	0	0	0
PCWrite	1	0	0	0	0	1
EnPC	0	X	X	X	X	1
IorD	0	X	X	X	1	X
MemRead	1	0	0	0	1	0
MemWrite	0	0	0	0	0	0
MemtoReg	X	X	X	X	X	X
IRWrite	1	0	0	0	0	0
PCSource	0	X	X	X	X	X
ALUop	00	00	00	00	XX	XX
ALUSrcA	01	01	10	00	XX	XX
ALUSrcB	01	11	00	10	XX	XX
RegWrite	0	0	0	0	0	0
RegDst	X	X	X	X	X	X