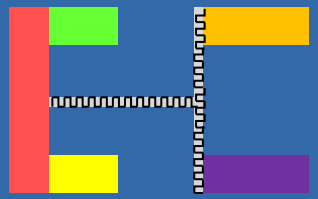
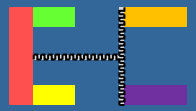


TEMA 4.
UNIDAD CENTRAL DE PROCESAMIENTO

dtic



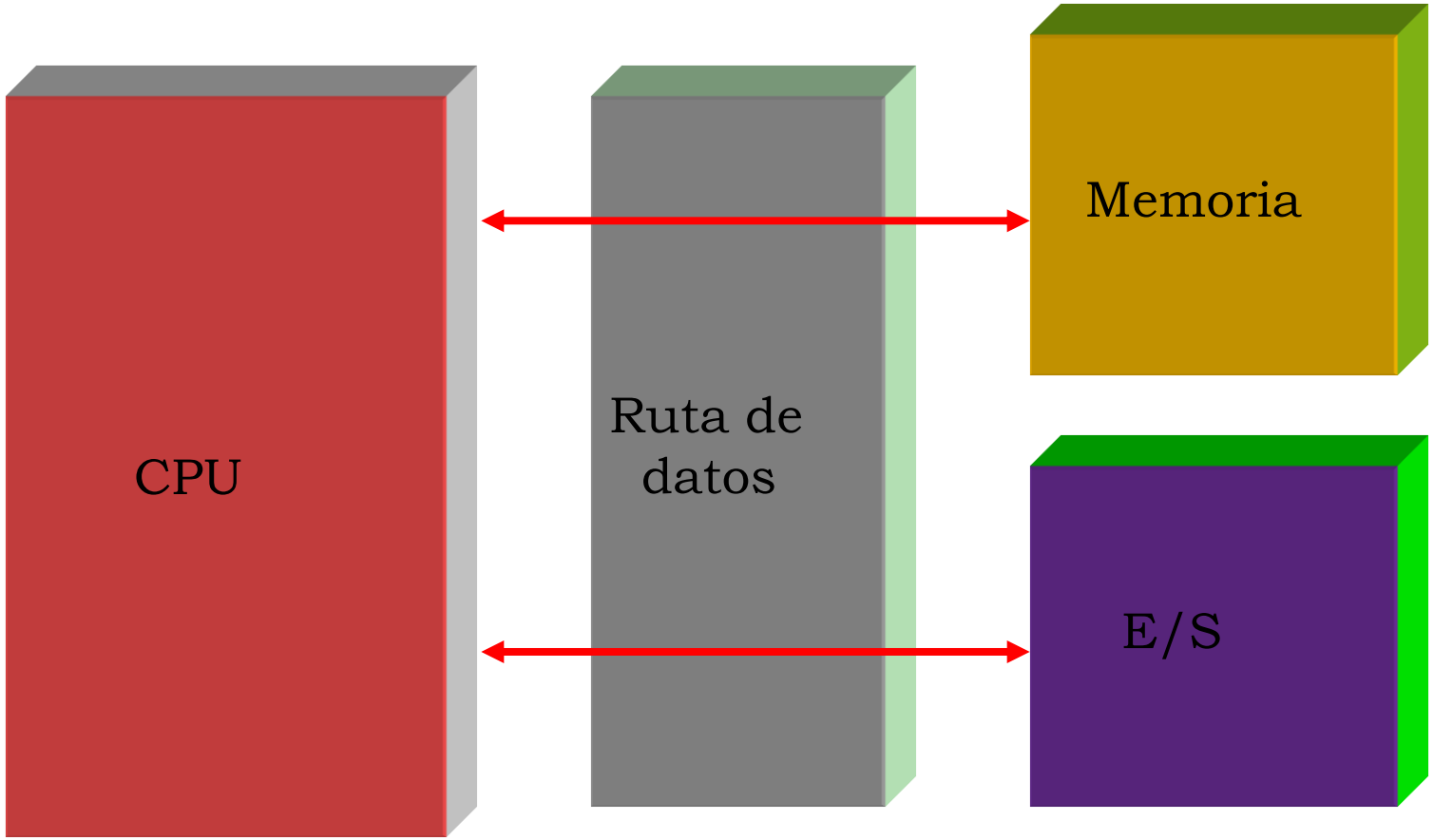
UNIDAD CENTRAL DE PROCESAMIENTO

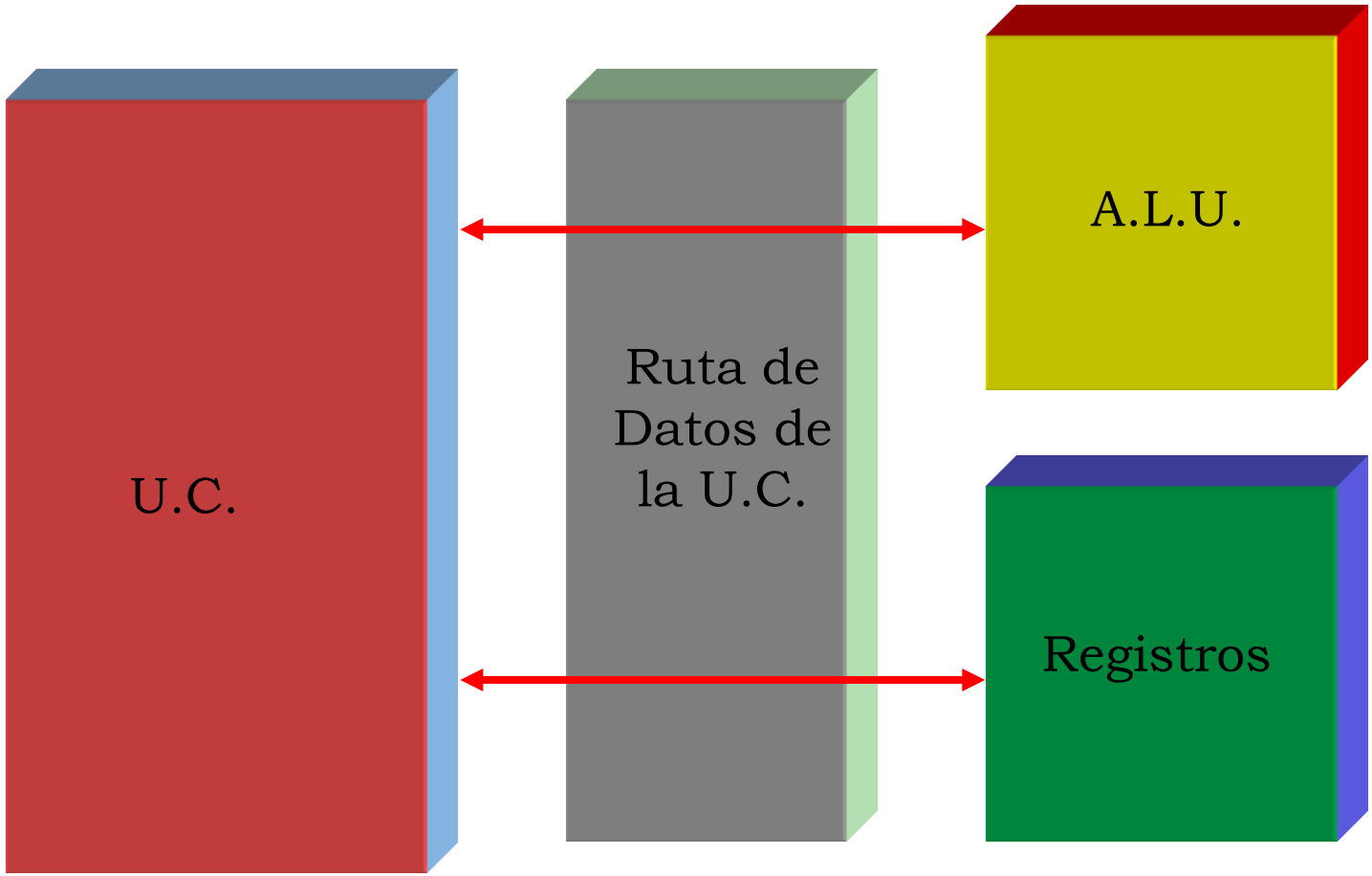
Índice

UNIDAD CENTRAL DE PROCESAMIENTO

- Las estructuras básicas del computador:
  - Unidad Central de Procesamiento
    - Unidad Aritmético-Lógica
    - Unidad de Control
  - Memoria Principal
  - Unidad de E/S
  - Ruta de datos

COMPUTADOR




U.C.
Ruta de Datos de la U.C.
A.L.U.
Registros

PROCESADOR EN ESTUDIO: MIPS

- Máquina RISC (computador con repertorio de instrucciones reducido)
- Tamaño buses y ancho de palabra: 32 bits
- Banco de registros
  - 32 registros de propósito general
  - Ancho de los registros: 32 bits
- Memoria:
  - 4G x 32
  - Acceso por byte (B), media palabra (16 bits, H), palabra (32 bits, W)
- Ancho registros de direcciones (PC) y datos (RI, MDR): 32 bits

- Estudio simplificado: subconjunto de instrucciones del MIPS:
  - Instrucciones de referencia a memoria: **lw, sw**

```
lw $t3, 4($t1);    $t3 ← M[$t1+4]
sw $t0, 48($s3);   M[48+$s3] ← $t0
```

  - Instrucciones aritmético-lógicas : **add, sub, and, or, slt**

```
add $t0, $s2, $t0 ; $t0 ← $s2 + $t0
slt $t0, $s3, Ss4 ; Si ($s3<$s4) entonces $t0=1 sino $t0=0
```

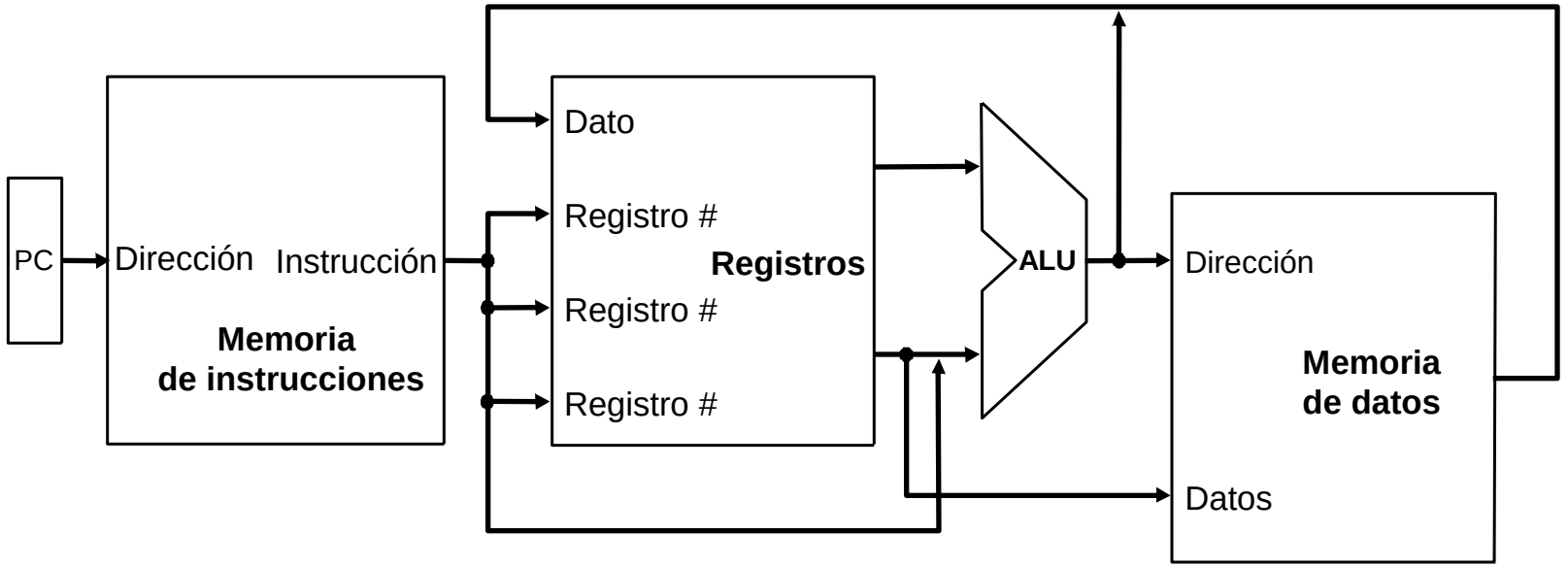
  - Instrucciones de control : **beq, j**

```
beq $t0, $zero, Salto ; Si $t0=$zero entonces ir a Salto
j Bucle ; Ir a Bucle
```

- Implementación genérica:
  - Uso del contador de programa (PC) para proporcionar la dirección de la instrucción que se encuentra en memoria.
  - Leeremos registros
  - Utilizaremos la instrucción para decidir exactamente que hacer

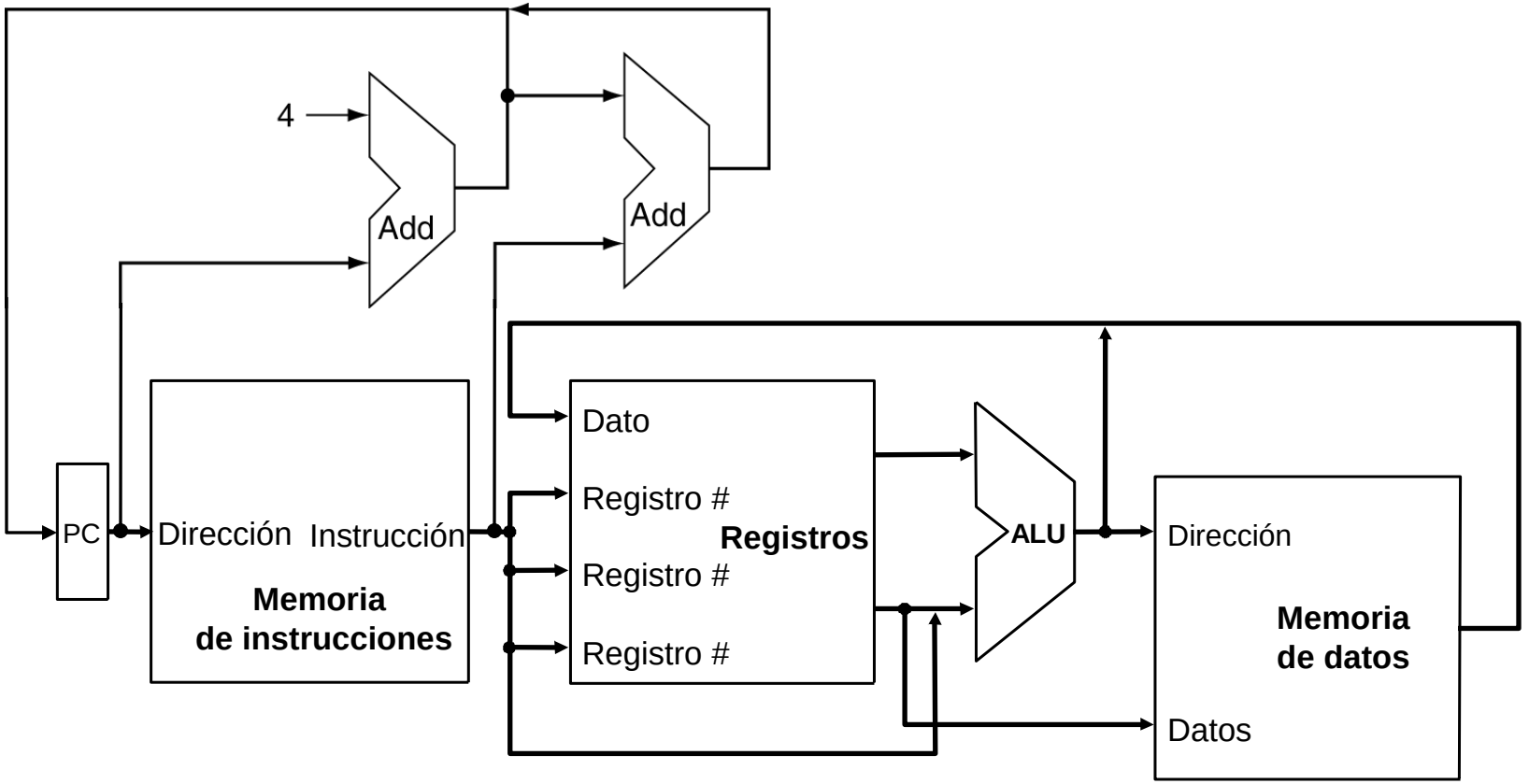
- Todas las instrucciones usarán la ALU después de leer los registros
  ¿porqué? ¿Referencia a memoria? ¿Aritméticas? ¿Control?

VISIÓN GENERAL DEL PROCESADOR



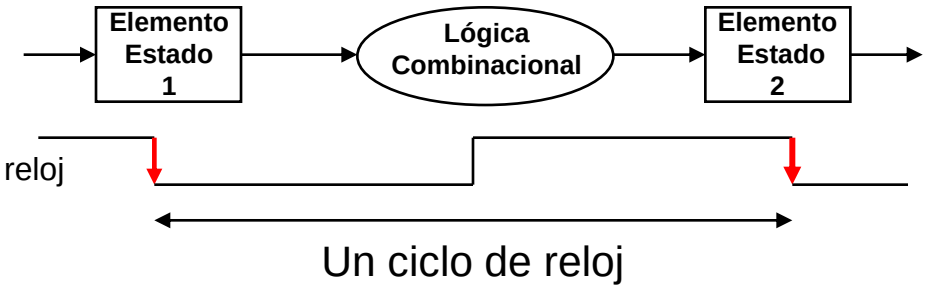
- Dos tipos de unidades funcionales :
  - elementos que operan con datos (combinacionales)
  - elementos que contienen estado (secuenciales)

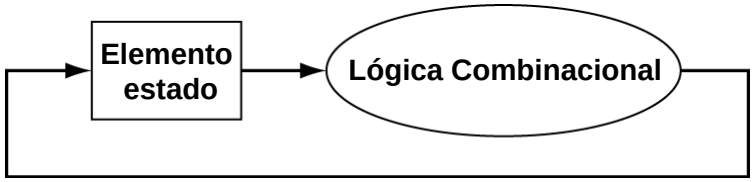
- Visión más refinada del procesador del esquema de implementación simple (monociclo).



METODOLOGÍA DE SINCRONIZACIÓN

- Metodología disparada por flanco
- Ejecución típica:
  - Leer contenido de algún elemento de estado
  - Enviar valores a través de la lógica combinacional
  - Escribir resultado en uno o más elementos de estado

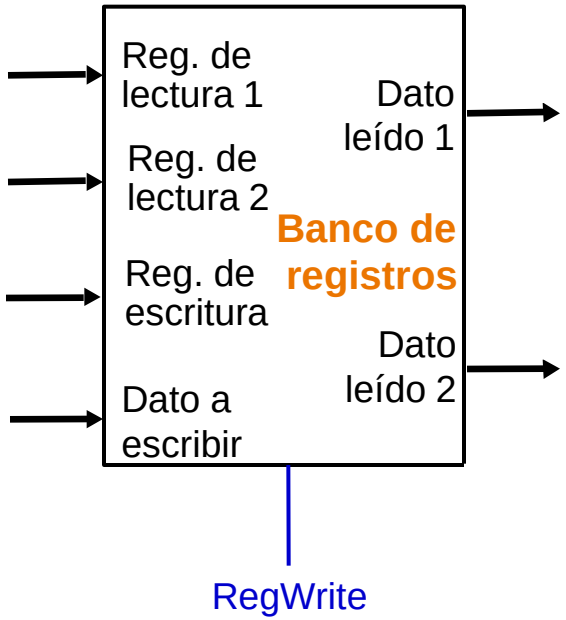


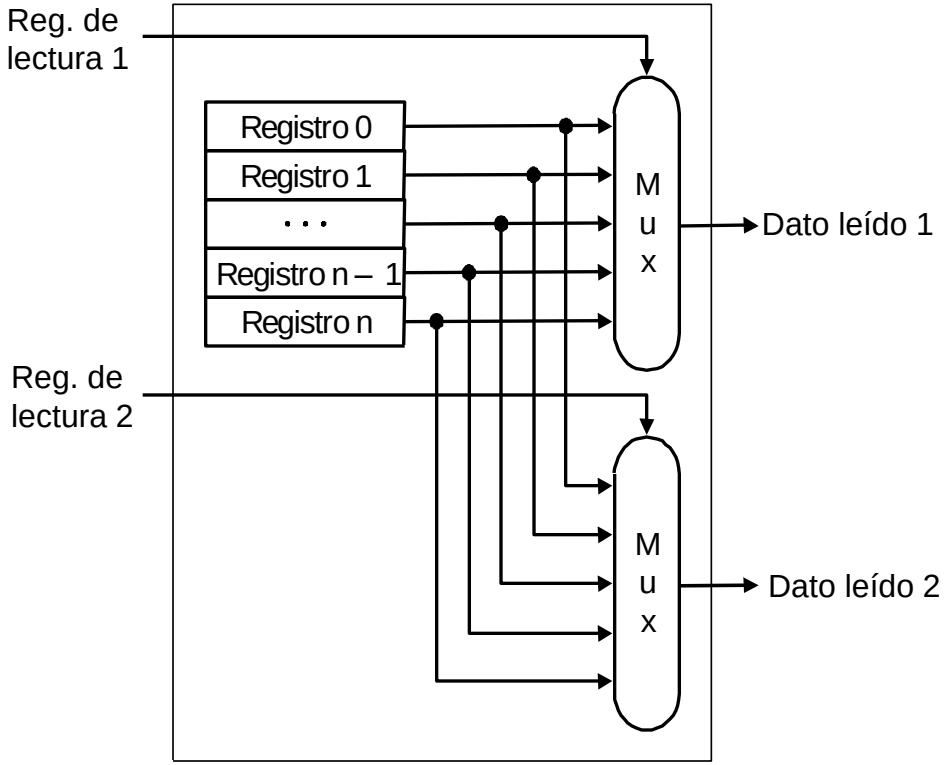


CONSTRUCCIÓN DE LA RUTA DE DATOS

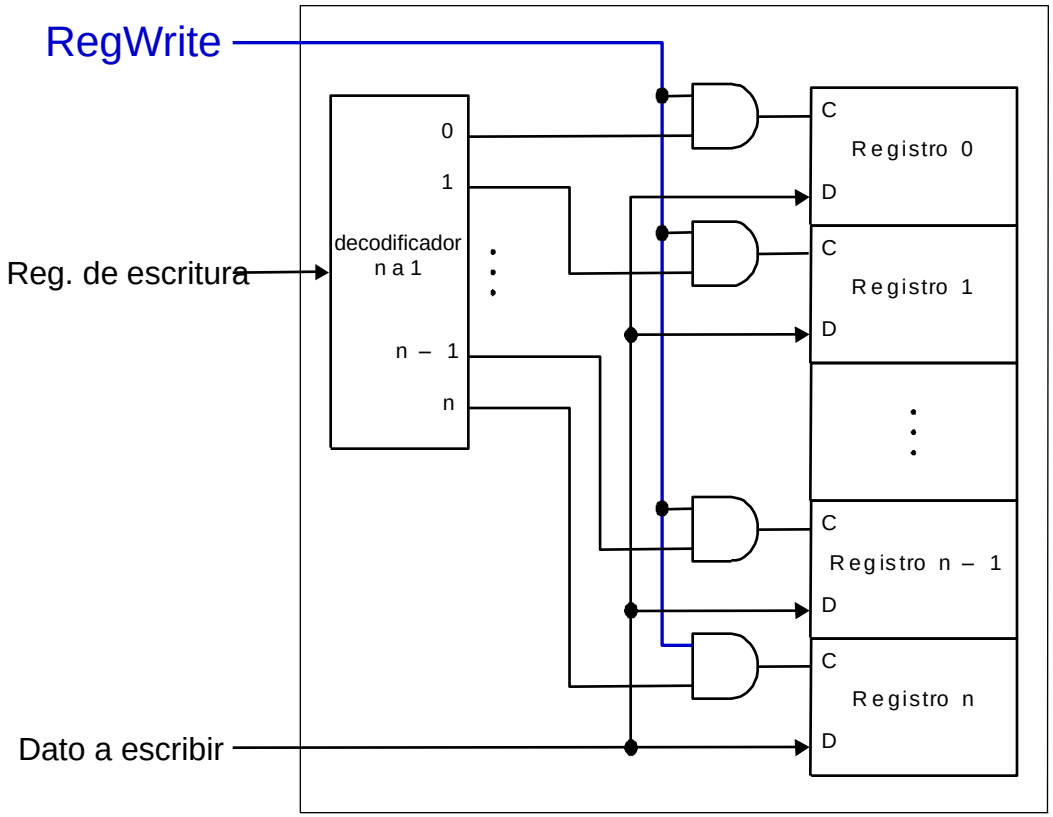
- Ruta de datos:
  - Elementos que procesan los datos y las direcciones en la CPU
    - Registros,
    - ALUs,
    - multiplexores,
    - memorias,
    - Buses
    - …

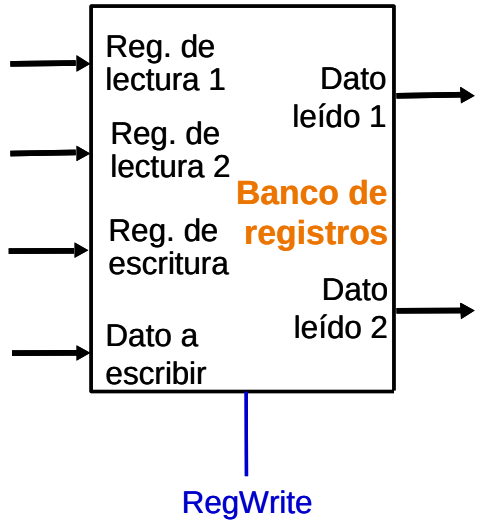
- Construido con flips-flops D.
  - **Lectura del Banco de registros.**





Escritura en el Banco de registros

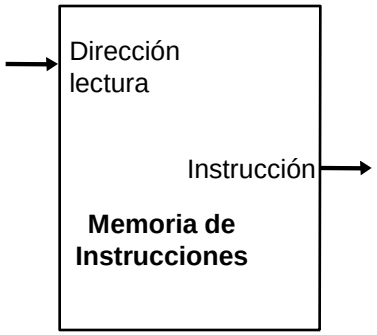




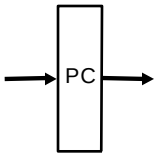
BÚSQUEDA DE LA INSTRUCCIÓN

- Acciones que involucra:
  - Leer instrucción de la memoria de instrucciones
  - Actualizar el valor del PC para que apunte a la siguiente instrucción

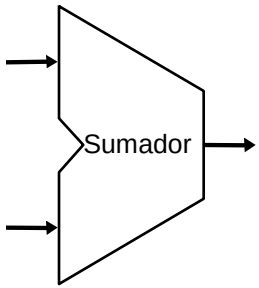
Elementos de la ruta de datos:



a. Memoria de Instrucciones

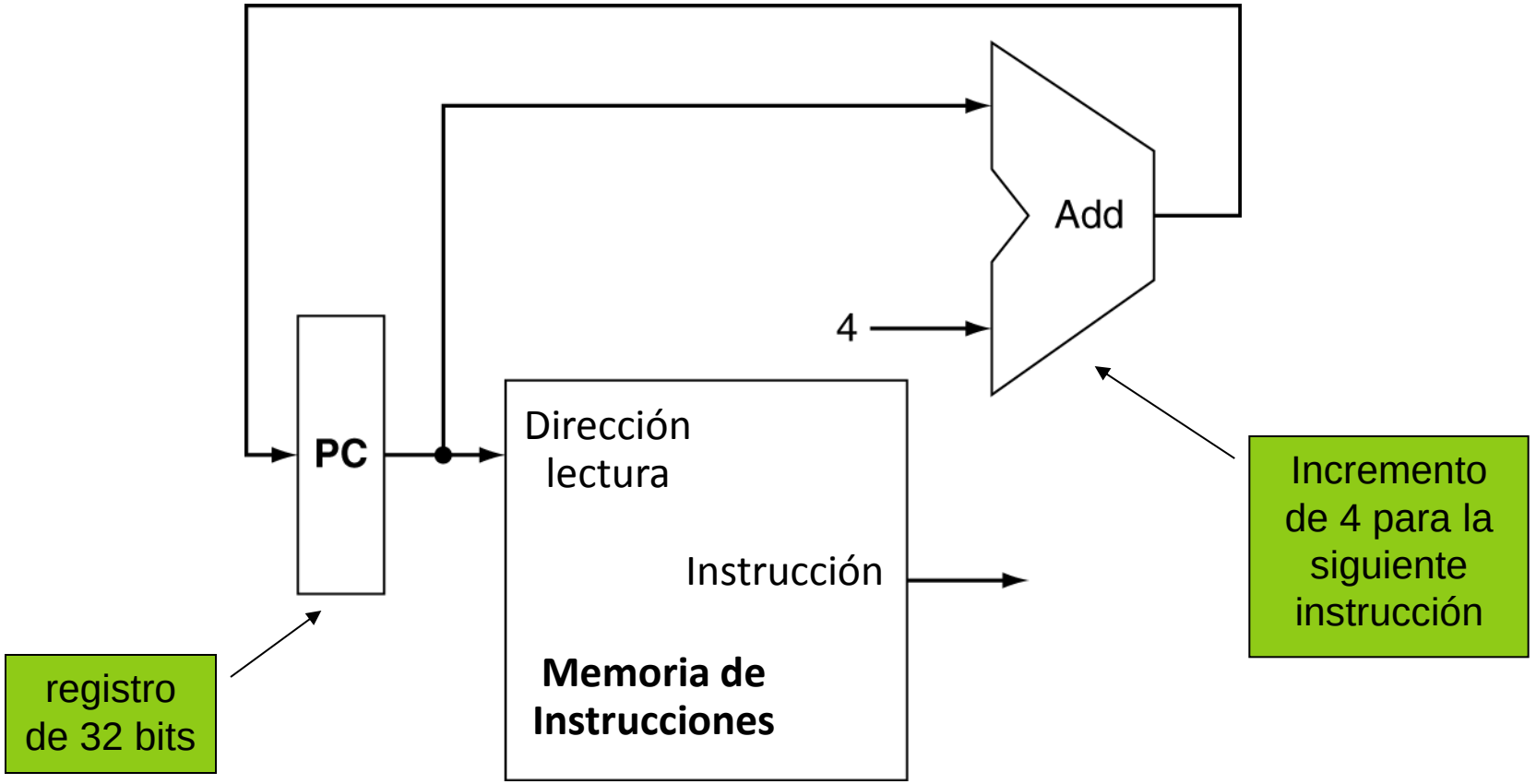


b. Contador del programa



c. Sumador

BÚSQUEDA DE LA INSTRUCCIÓN



INSTRUCCIONES TIPO R

- Formato tipo R (**add, sub, and, or, slt**)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Ejemplo: `add $t0, $s1, $s2` (`$8← $17+$18`)

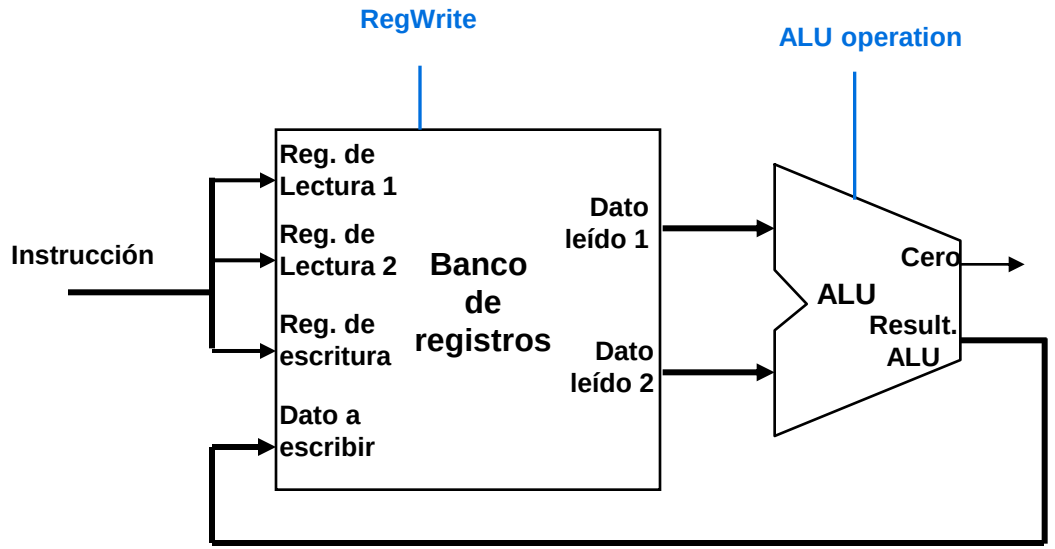
Cod. Op.	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

INSTRUCCIONES TIPO R

- Acciones:

```
add $t0, $s1, $s2
($8← $17+$18)
```

  - Leer dos registros operandos
  - Realizar la operación aritmética o lógica
  - Escribir en el registro resultado



INSTRUCCIONES DE CARGA Y ALMACENAMIENTO

- Formato tipo I (**lw, sw**)

op	rs	rt	constante o dirección
6 bits	5 bits	5 bits	16 bits

- Ejemplo: `lw $t1, 100($t2)` (`$9← M[$10+100]`)

op	$t2	$t1	Desplazamiento 16 bits
35	10	9	100
100011	01010	01001	0000 0000 0110 0100

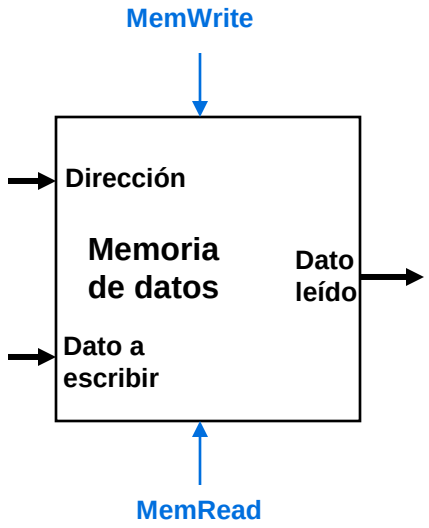
INSTRUCCIONES DE CARGA Y ALMACENAMIENTO

lw $t1, 100($t2) ($t1← M[$t2+100])

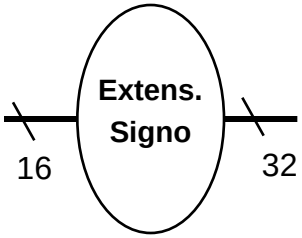
sw $t0, 48($s3); M[48+$s3] ← $t0

- Acciones:
  - Leer registros operando del Banco de Registros
  - Calcular dirección de memoria utilizando el desplazamiento de 16 bits (utilizaremos la ALU y la unidad de extensión de signo)
  - Carga (lw): Escribir en el Banco de Registros el dato leído de la memoria de de datos.
  - Almacenamiento (sw): Escribir en la memoria de datos el registro leído del Banco de Registros.

Elementos nuevos:

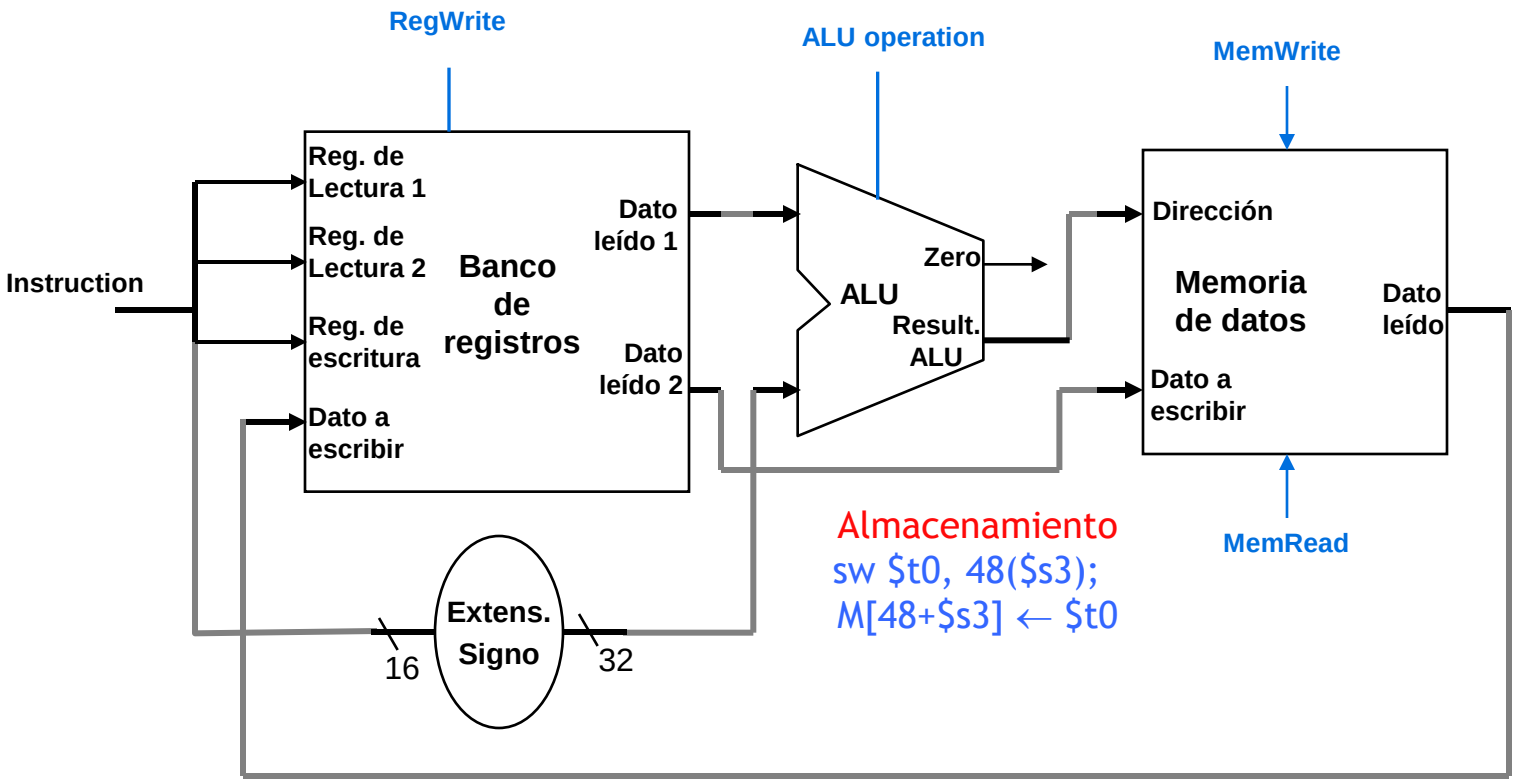


a) Memoria de datos



b) Unidad de extensión de signo

- Ruta de datos:



INSTRUCCIÓN DE SALTO CONDICIONAL

- Formato tipo I (**beq**)

op	rs	rt	constante o dirección
6 bits	5 bits	5 bits	16 bits

- Ejemplo: `beq $1, $2, 100` (Si $1=$2 entonces PC← PC+4+100

op	$2	$1	Desplazamiento 16 bits
4	2	1	100
000100	00010	00001	0000 0000 0110 0100

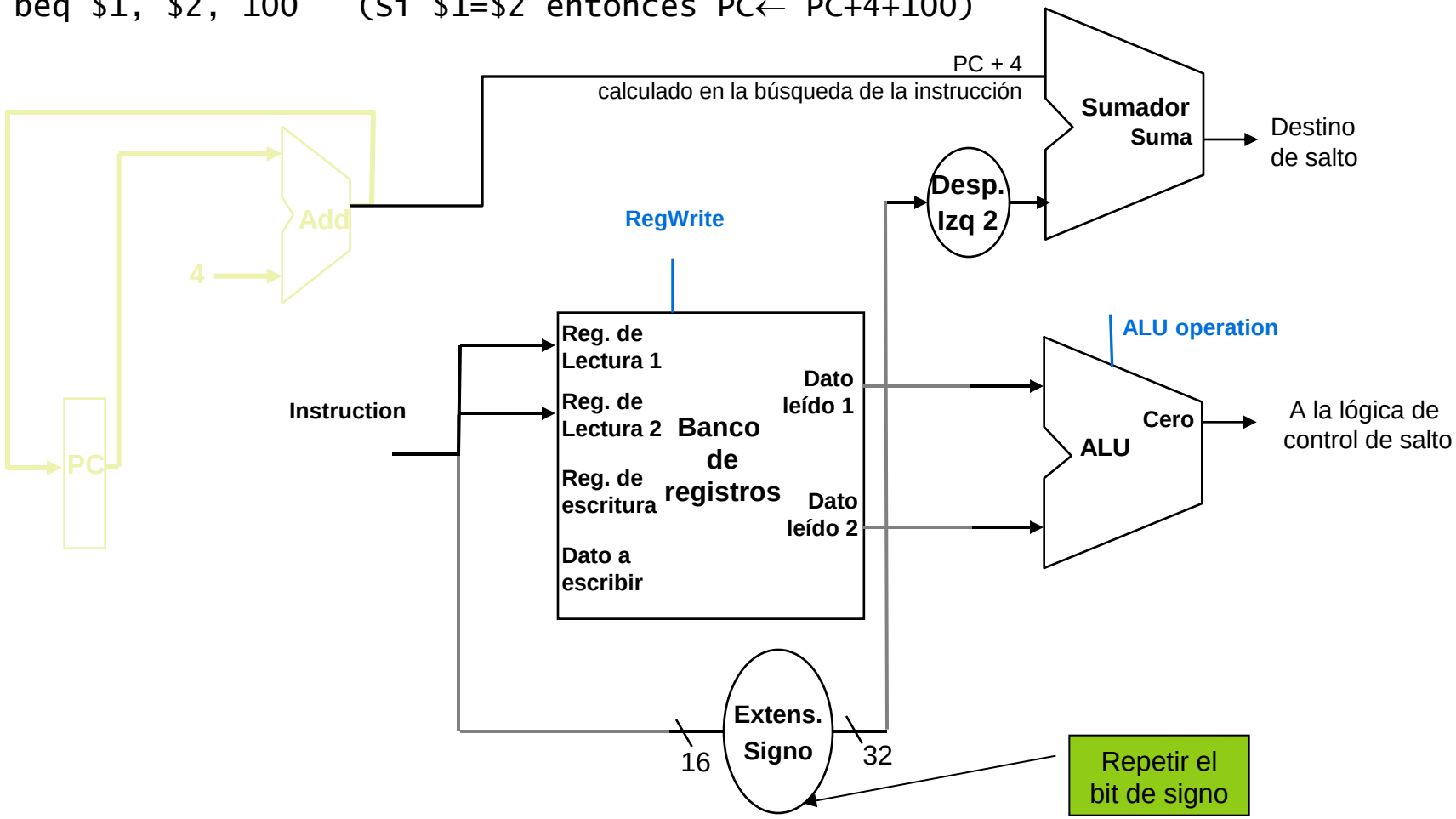
INSTRUCCIÓN DE SALTO CONDICIONAL

- Acciones:
  - Leer registros operandos.
  - Comparar los registros
    - (Utilizar la ALU para restar y chequear el indicador de resultado “cero”)
  - Calcular la dirección de salto
    - Extender el signo del campo desplazamiento
    - Desplazar dos bits a la izquierda
    - Utilizar un sumador para añadir el desplazamiento al nuevo PC (PC + 4 calculado en la búsqueda de la instrucción)

INSTRUCCIÓN DE SALTO CONDICIONAL

- Ruta de datos

```
beq $1, $2, 100   (Si $1=$2 entonces PC← PC+4+100)
```



INSTRUCCIÓN DE SALTO INCONDICIONAL

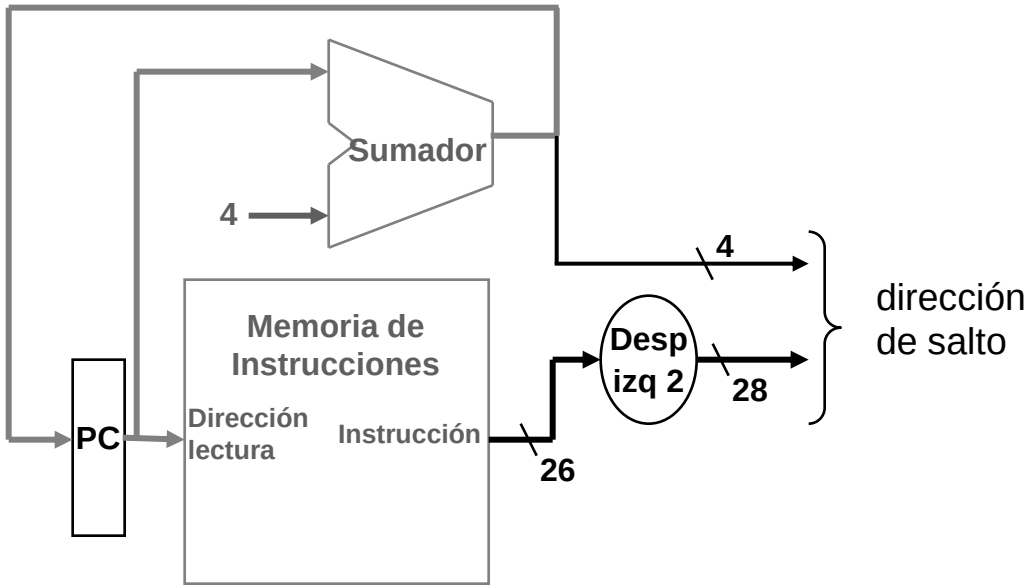
- Formato tipo J (**j**)

- Ejemplo: `j 2500` (PC← 10000)

op	Dirección 26 bits
2	2500
000010	00 0000 0000 0000 1001 1100 0100

INSTRUCCIÓN DE SALTO INCONDICIONAL

- Acciones:
  - Reemplazar los 28 bits de menor peso del PC actualizado (PC+4 calculado en la búsqueda de la instrucción) por los 26 bits de menor peso de la instrucción desplazados 2 bits a la izquierda.



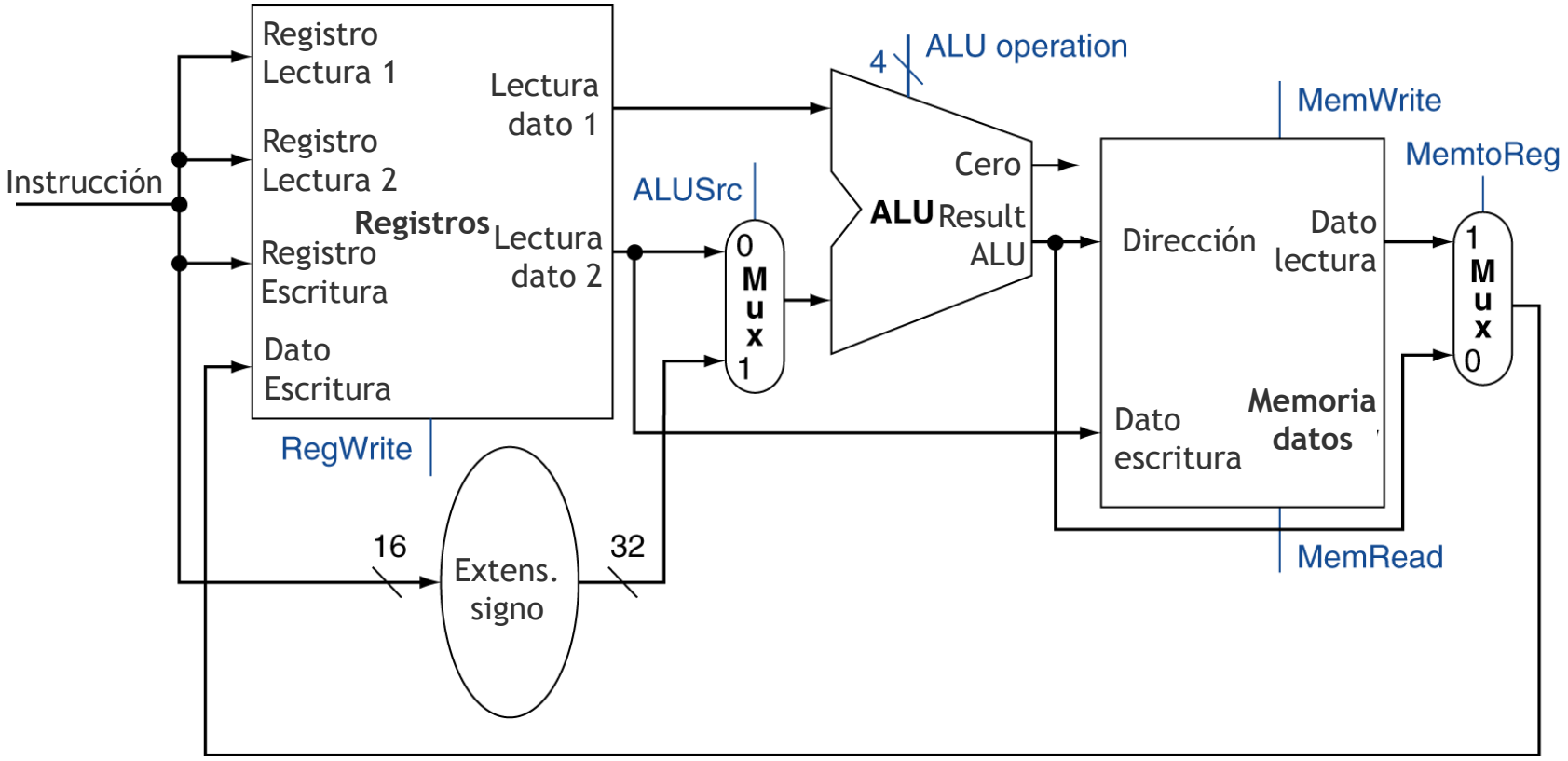
IMPLEMENTACIÓN MONOCICLO

- Combinar segmentos de la ruta de datos y añadir las líneas de control y los multiplexores necesarios.
- Diseño monociclo: las instrucciones se ejecutan en un único ciclo de reloj.
  - Ningún elemento de la ruta de datos pude utilizarse más de una vez por instrucción (duplicación de elementos):
    - Memoria separadas para instrucciones y datos, varios sumadores…
  - Utilizar multiplexores para compartir elementos con entradas distintas para instrucciones diferentes.
  - La duración del ciclo de reloj estará determinado por la duración de la instrucción más lenta.

EJEMPLO DE RUTA DE DATOS

Esquema de implementación simple

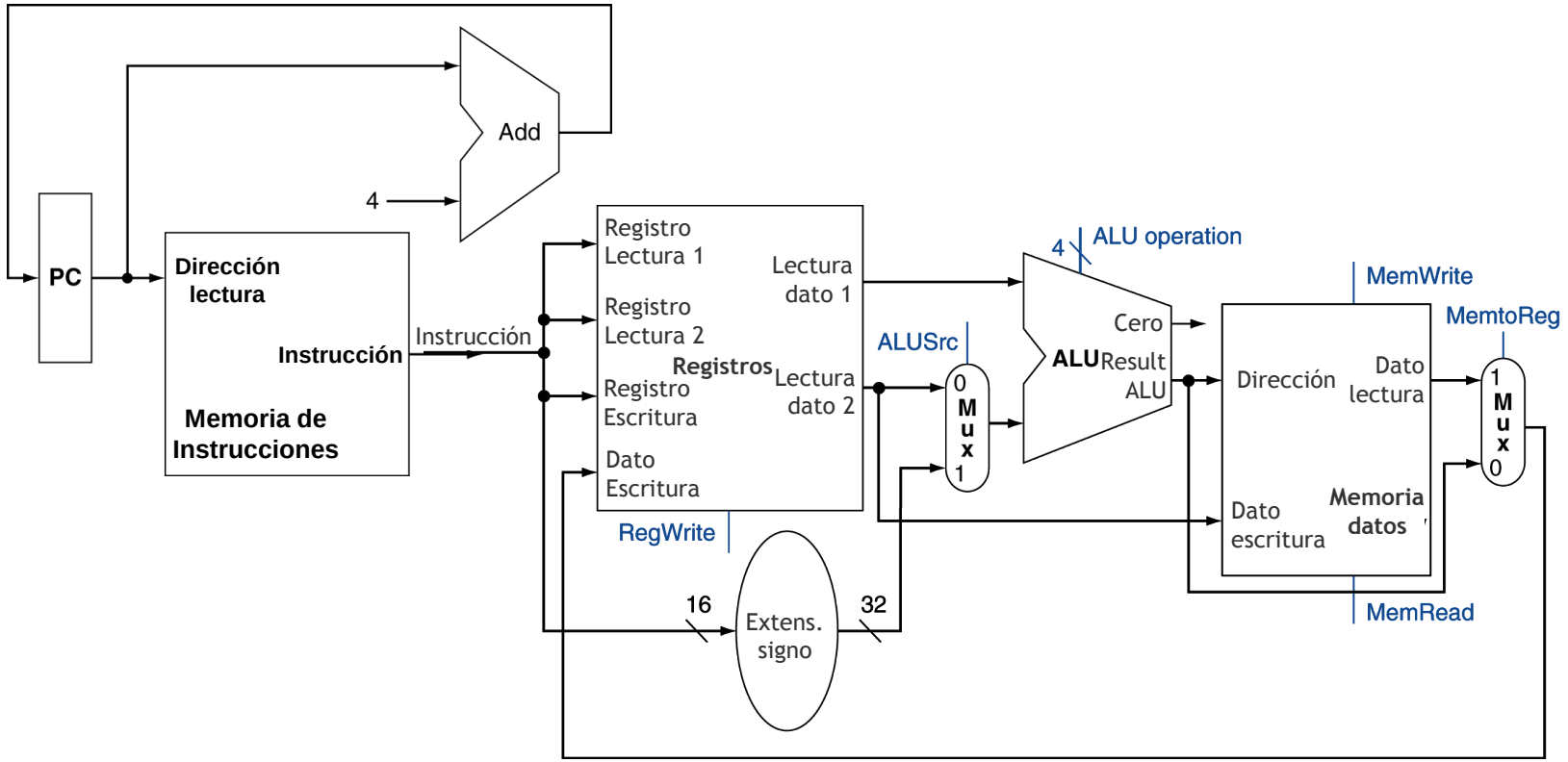
Parte operacional para instrucciones con referencia a memoria y aritmético-lógicas.



EJEMPLO DE RUTA DE DATOS

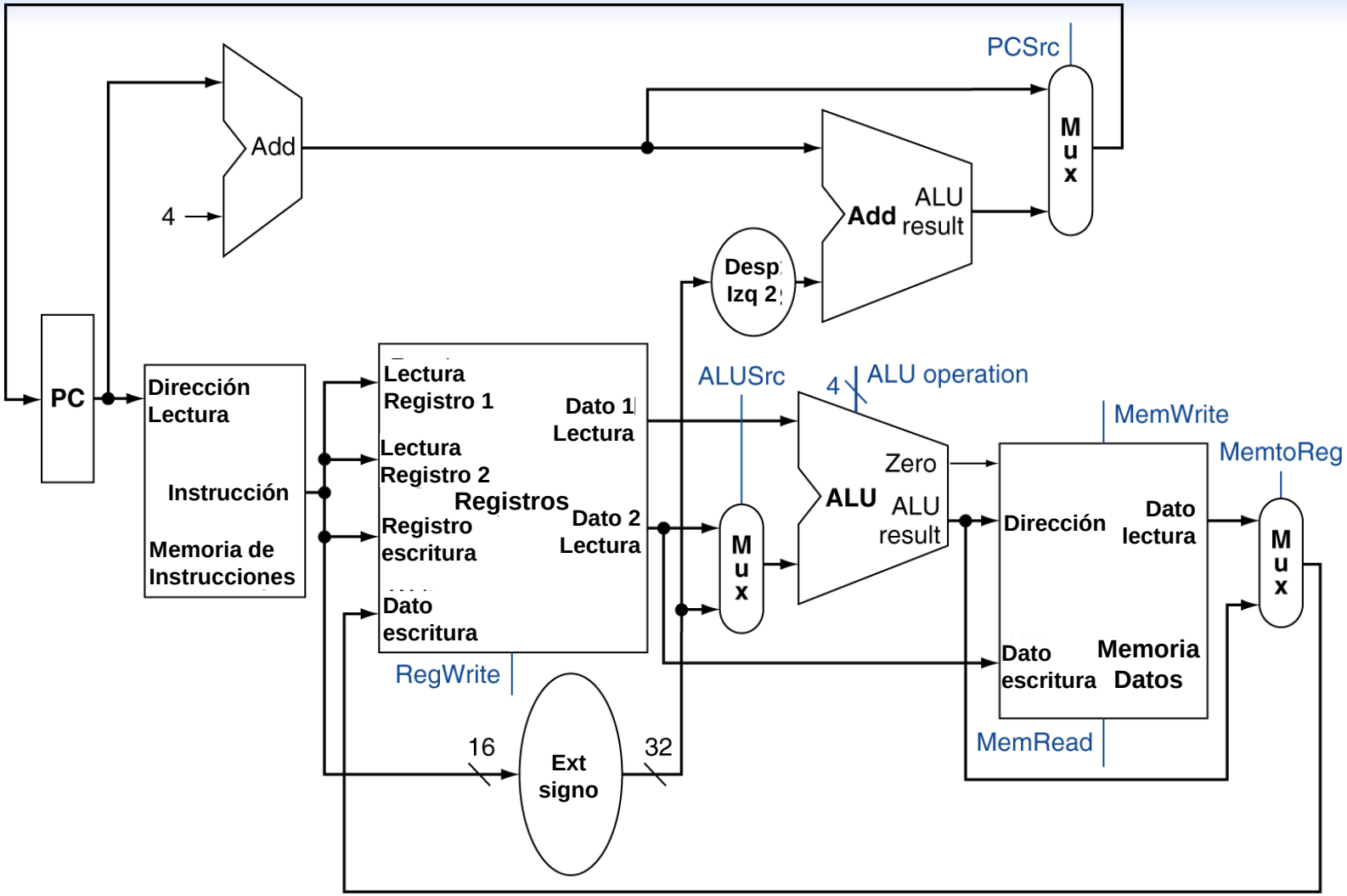
Esquema de implementación simple

Incorporamos la búsqueda de la instrucción.



RUTA DE DATOS COMPLETA

Esquema de implementación simple



- Selecciona las operaciones a realizar (operación ALU, leer/escribir, etc.)
- Controla el flujo de los datos (entradas de los multiplexores)
- La información proporcionada por los 32 bits de la instrucción
- Ejemplo: add $8, $17, $18

op	rs	rt	rd	shamt	funct
000000	10001	10010	01000	00000	100000

- La operación de la ALU está determinada por el tipo de instrucción y el campo funct.

- Ejemplo: ¿Qué debería hacer la ALU con esta instrucción?
- lw $1, 100($2)

op	rs	rt	Desplazamiento 16 bits
35	2	1	100

- Señales de control a la ALU

ALU operation (4 bits)	Operación
0000	AND
0001	OR
0010	suma
0110	resta
0111	Activar si menor que
1100	NOR

CONTROL DE LA ALU

- Instrucciones lw/sw
  - Operación de la ALU: SUMA → **ALU operation =0010**
- Instrucciones de salto
  - Operación de la ALU: RESTA → **ALU operation =0110**
- Instrucciones Tipo R
  - Operación de la ALU: DEPENDE DEL CAMPO FUNCT
- Suponer señal **ALUOp** que identifique el tipo de instrucción:

$$\textbf{ALUOp} = \begin{cases} \textbf{00} \rightarrow \text{lw/sw} \\ \textbf{01} \rightarrow \text{beq} \\ \textbf{11} \rightarrow \text{Tipo R (aritmético-lógicas)} \end{cases}$$

CONTROL DE LA ALU

- Dos niveles de Control: control principal y control ALU
- El Control Principal genera señal **ALUop** según el código de operación

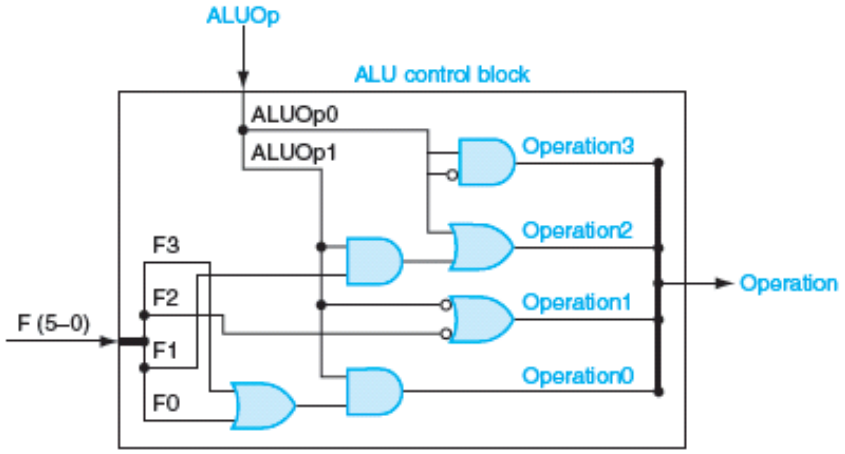
Instrucción	ALUOp	Operación	Campo funct	Acción de la ALU	Entrada de control a la ALU
lw	00	Cargar palabra	XXXXXX	Suma	0010
sw	00	Almacenar palabra	XXXXXX	Suma	0010
beq	01	Saltar si igual	XXXXXX	Resta	0110
R-type	10	Suma	100000	Suma	0010
		Resta	100010	Resta	0110
		AND	100100	AND	0000
		OR	100101	OR	0001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

CONTROL DE LA ALU

- Tabla de verdad:

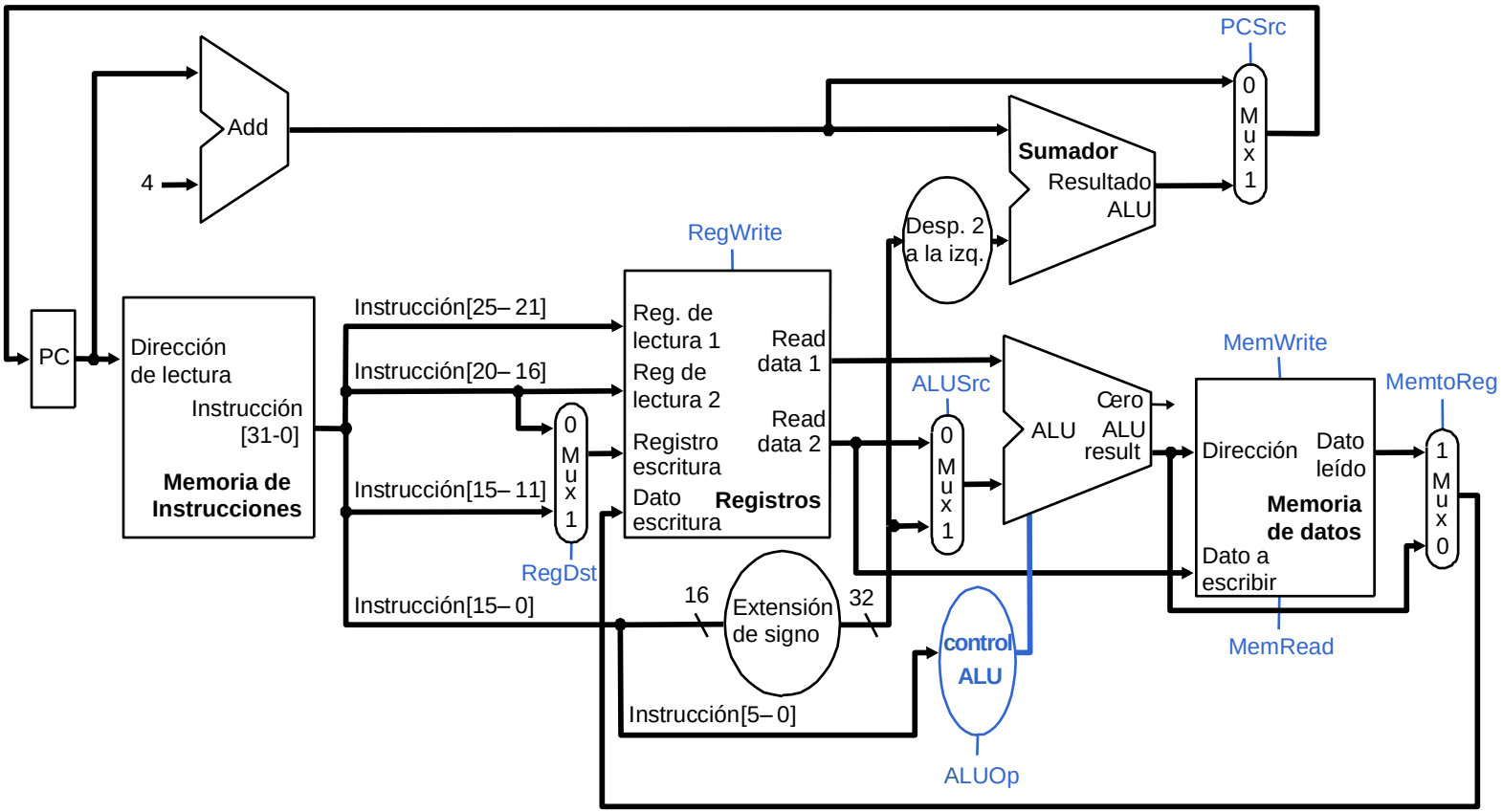
Bits ALUOp		Bits del campo funct						Entrada de control a la ALU Bits de operación			
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Op3	Op2	Op1	Op0
0	0	X	X	X	X	X	X	0	0	**1**	0
0	1	X	X	X	X	X	X	0	**1**	**1**	0
1	0	X	X	0	0	0	0	0	0	**1**	0
1	X	X	X	0	0	1	0	0	**1**	**1**	0
1	0	X	X	0	1	0	0	0	0	0	0
1	0	X	X	0	1	0	1	0	0	0	**1**
1	0	X	X	1	0	1	0	0	**1**	**1**	**1**

- Lógica Combinacional:

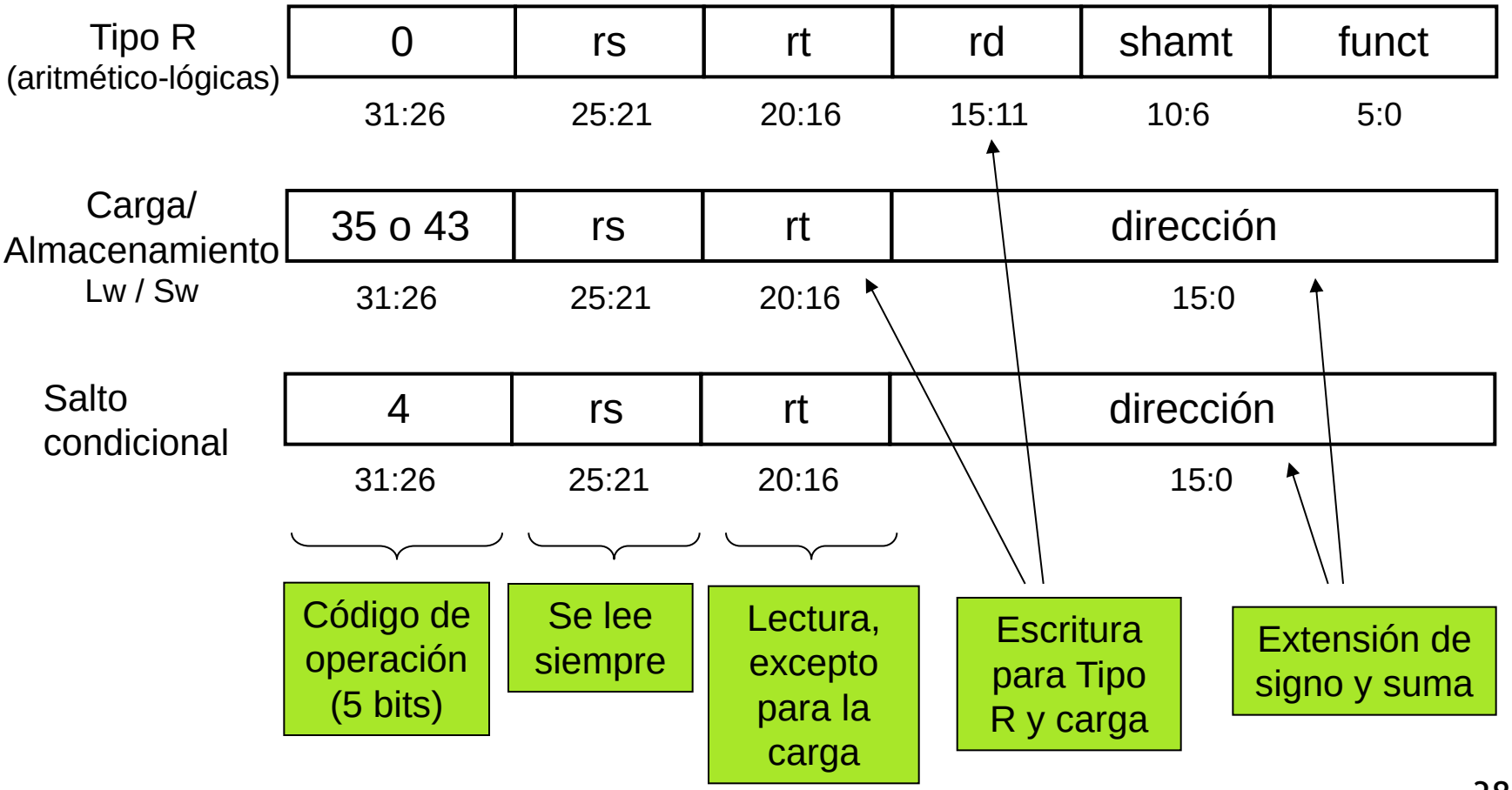


RUTA DE DATOS Y CONTROL ALU

Esquema de implementación simple



- Las señales de control se obtienen de las instrucciones:

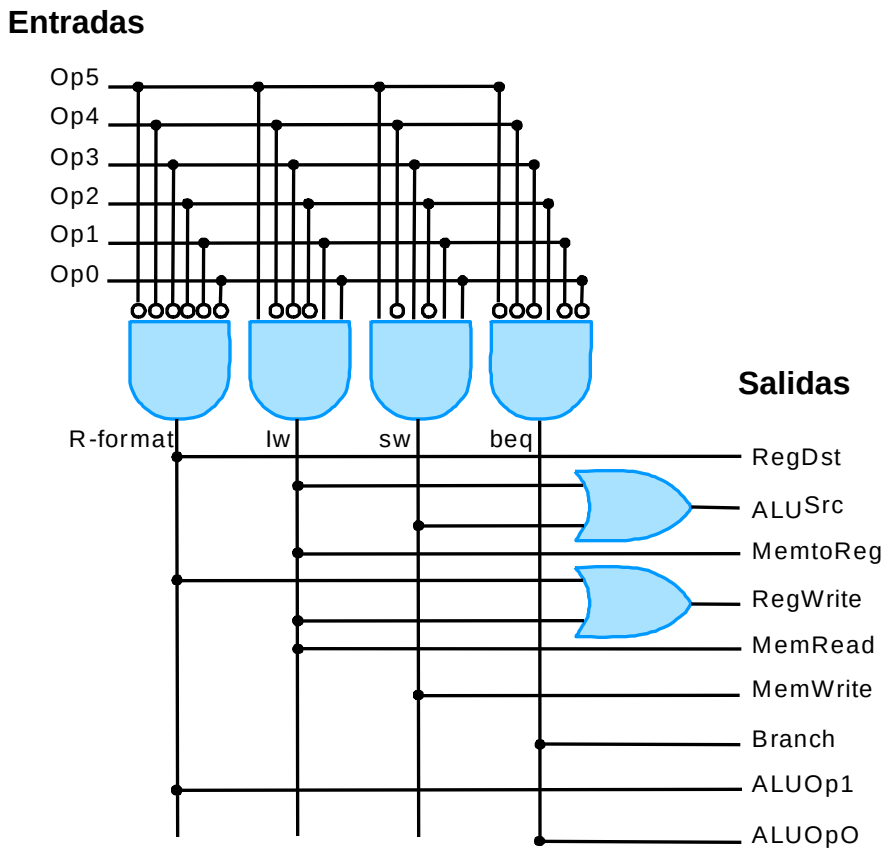


- Tabla de verdad:

	Entradas						Salidas								
Instrucción	Código de Operación						RegDst	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
	Op5	Op4	Op3	Op2	Op1	Op0									
Formato R	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	X	1	X	0	0	1	0	0	0
beq	0	0	0	1	0	0	X	0	X	0	0	0	1	0	1

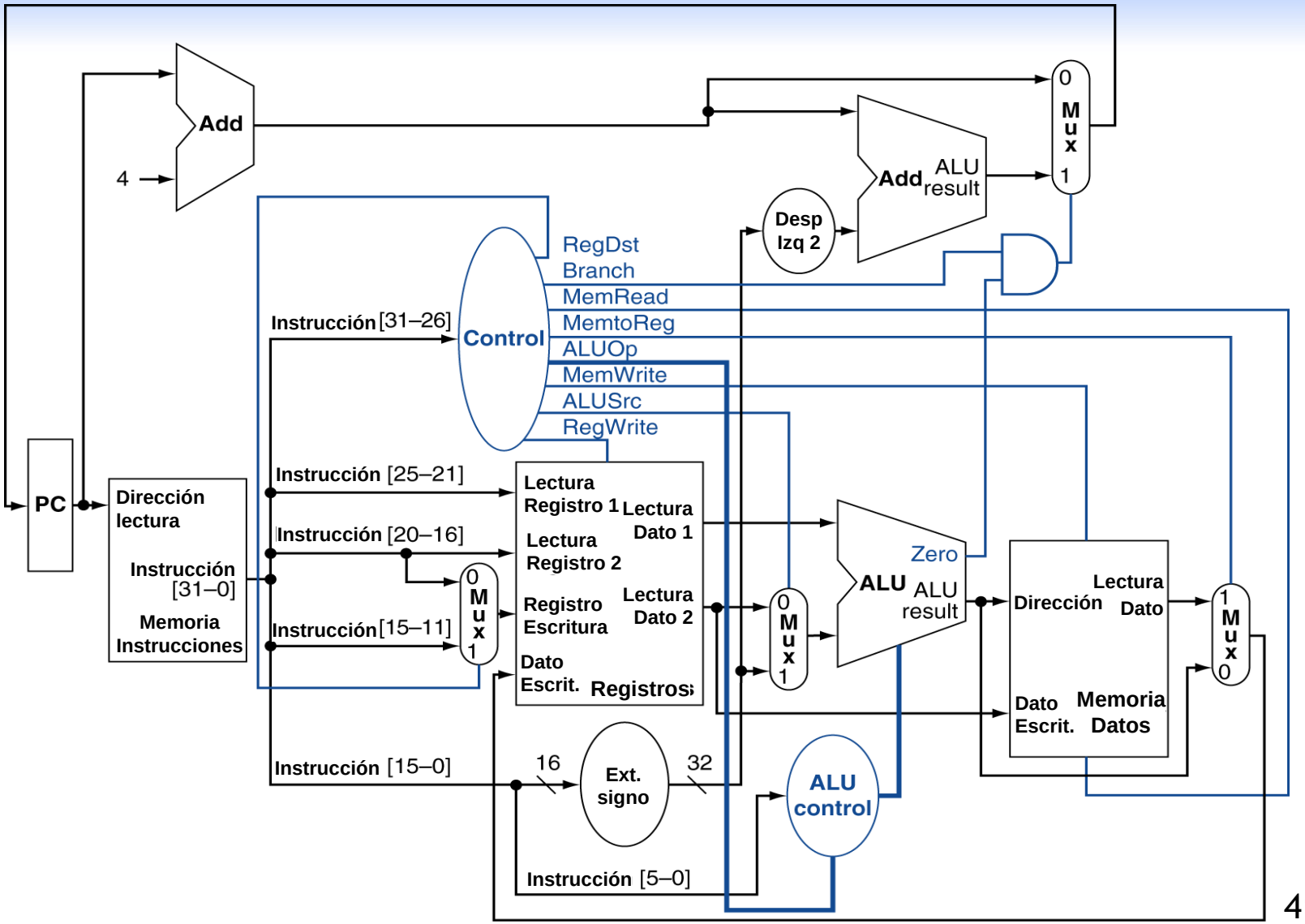
Cuando la tabla de verdad es grande conviene obtener la expresión algebraica como suma de productos que puede ser fácilmente implementada en un circuito lógico programable (PLD).

- Circuito lógico:



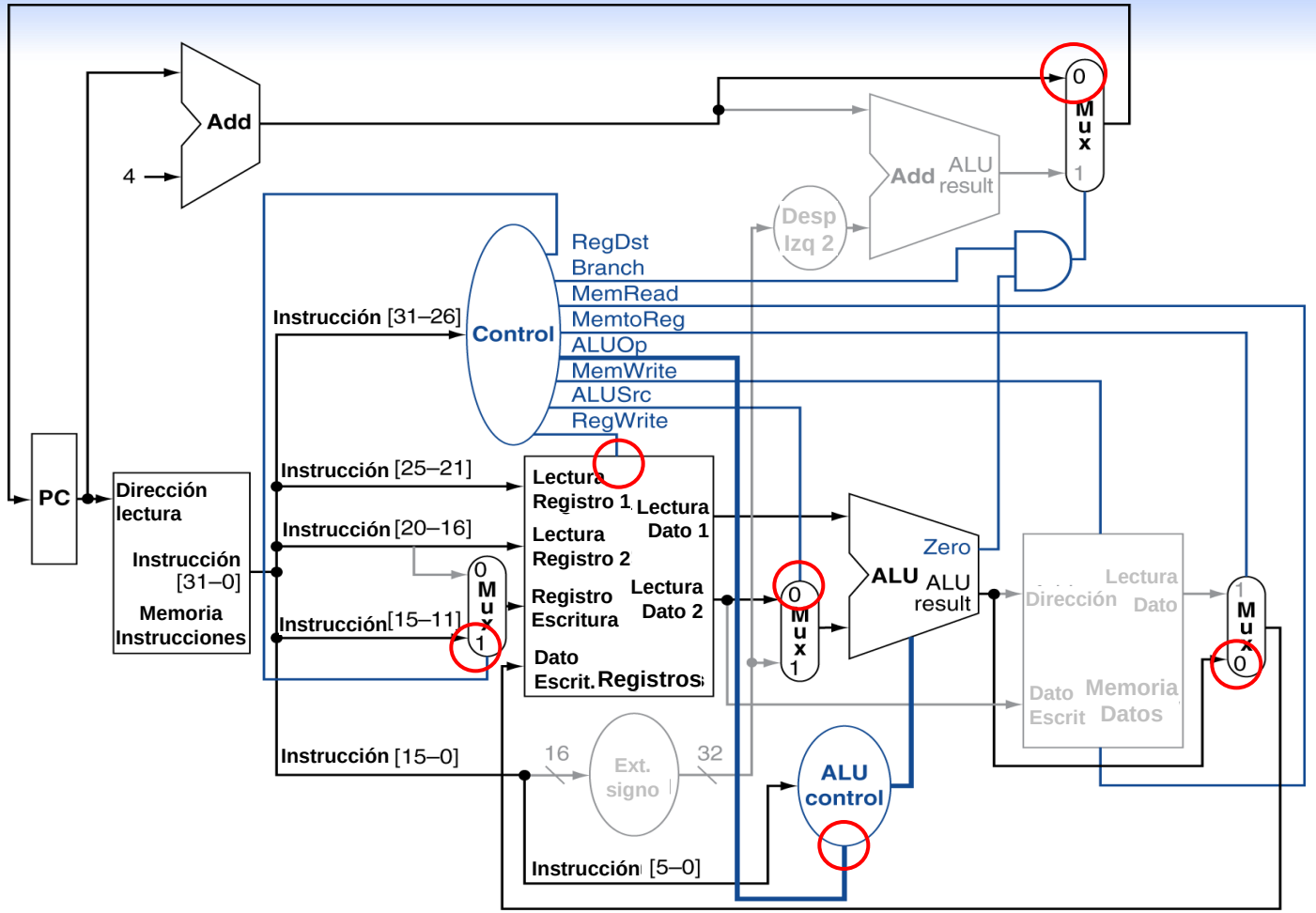
RUTA DE DATOS Y CONTROL

Esquema de implementación simple



INSTRUCCIONES FORMATO TIPO R

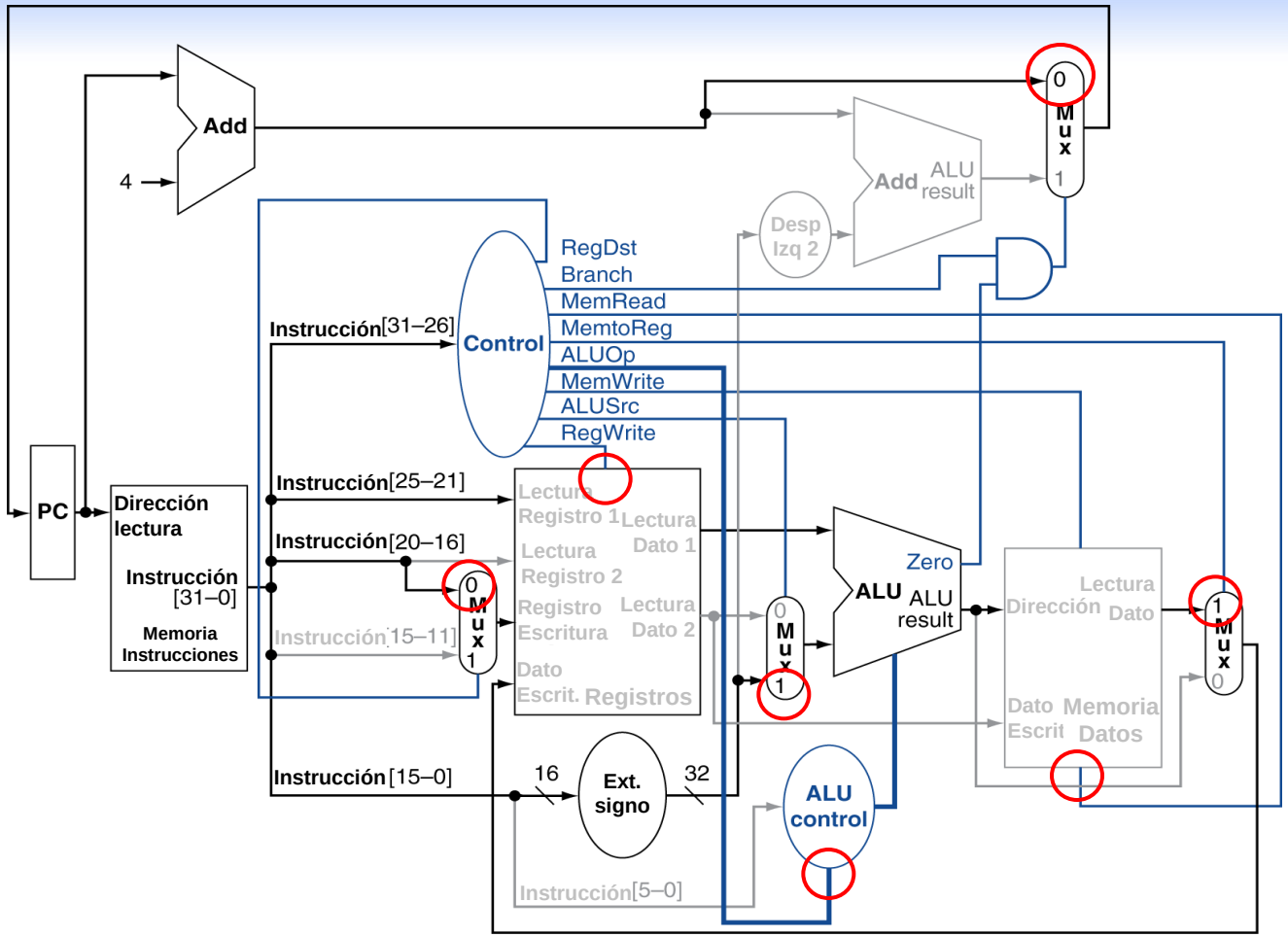
Esquema de implementación simple



Instrucción	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUp0
Formato R	1	0	0	1	0	0	0	1	0

INSTRUCCIÓN DE CARGA (Lw)

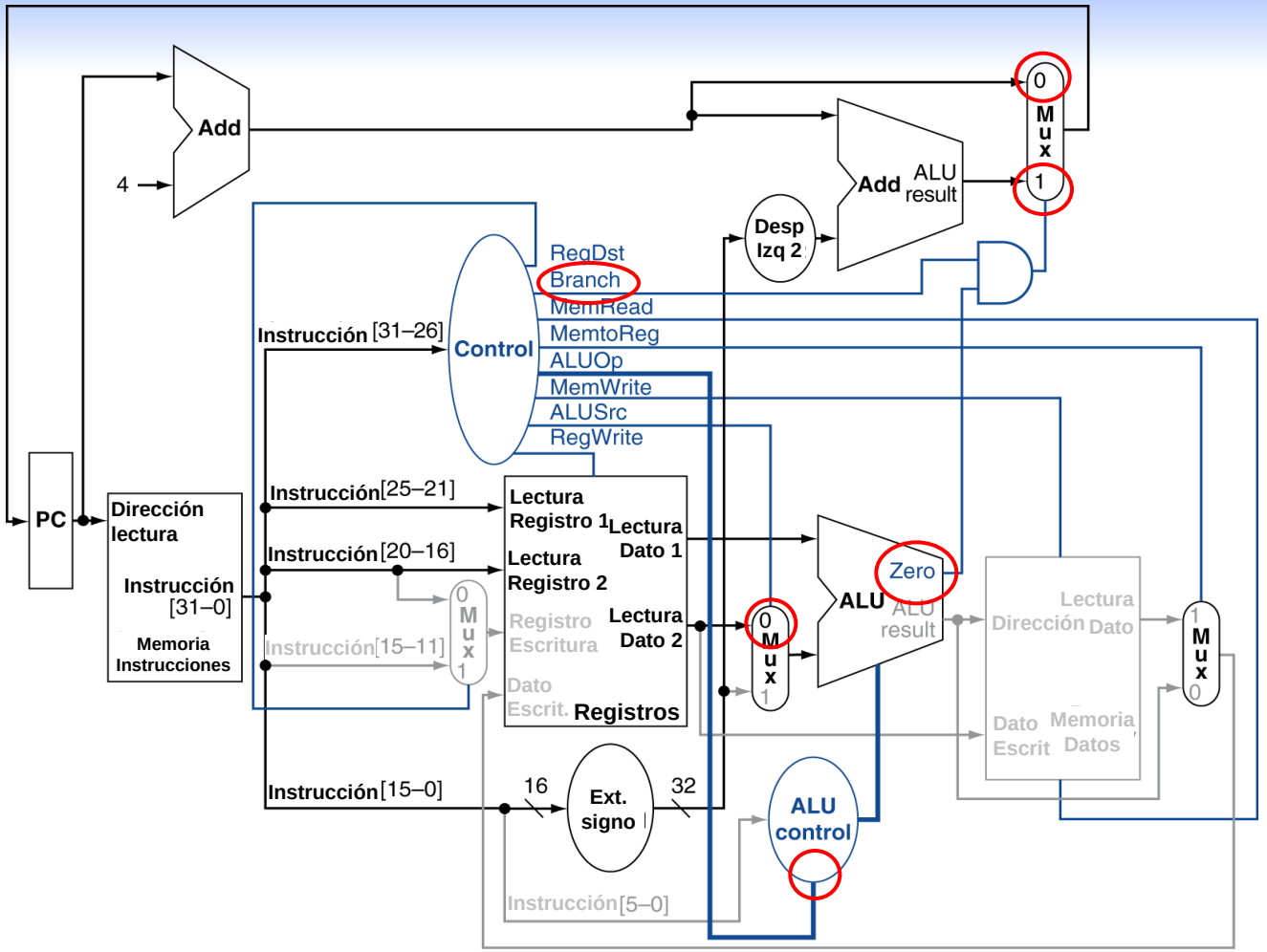
Esquema de implementación simple



Instrucción	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUp0
Lw	0	1	1	1	1	0	0	0	0

INSTRUCCIÓN SALTAR SI IGUAL (Beq)

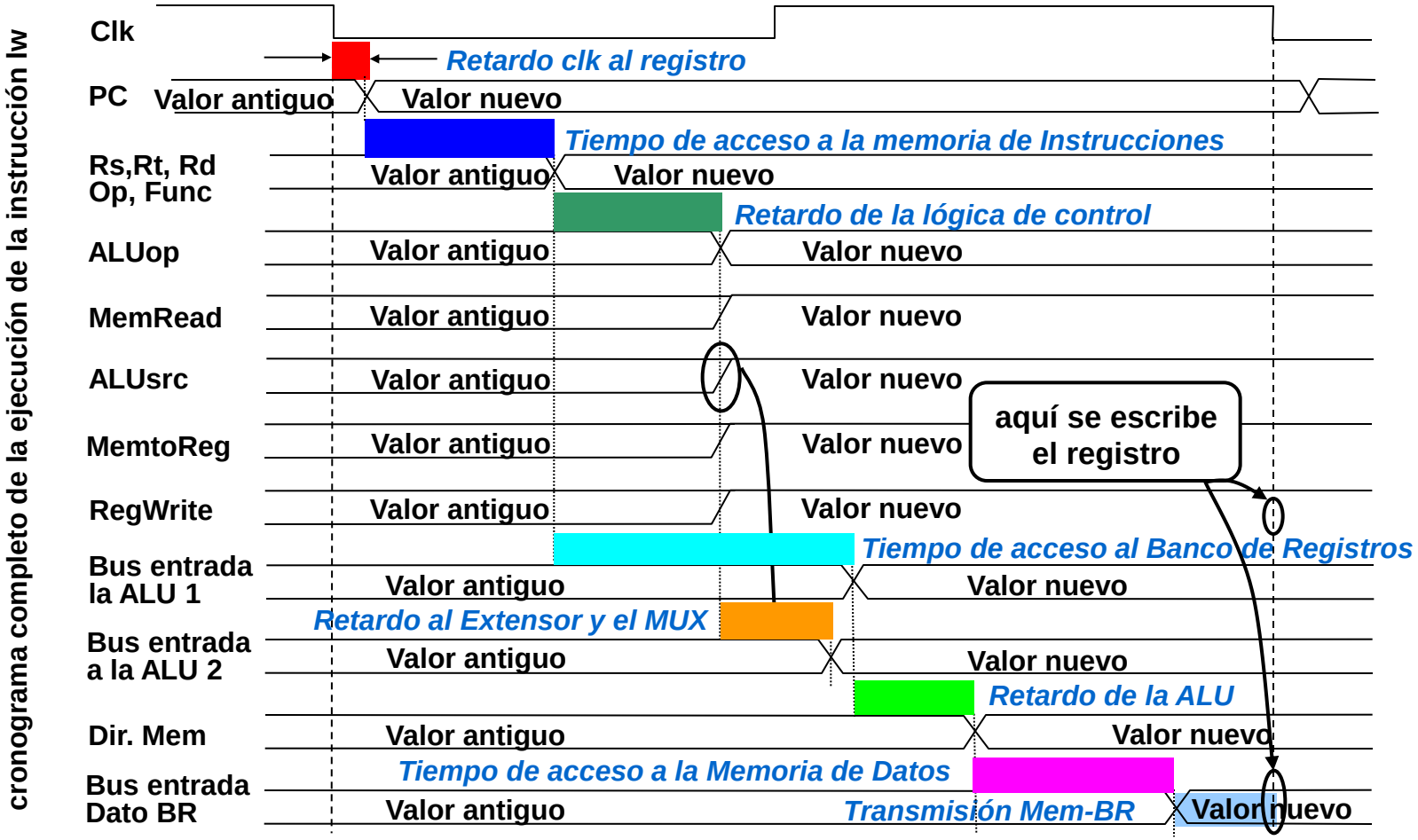
Esquema de implementación simple



Instrucción	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUp0
Beq	X	0	X	0	0	0	1	0	1

CRONOGRAMA INSTRUCCIÓN LW

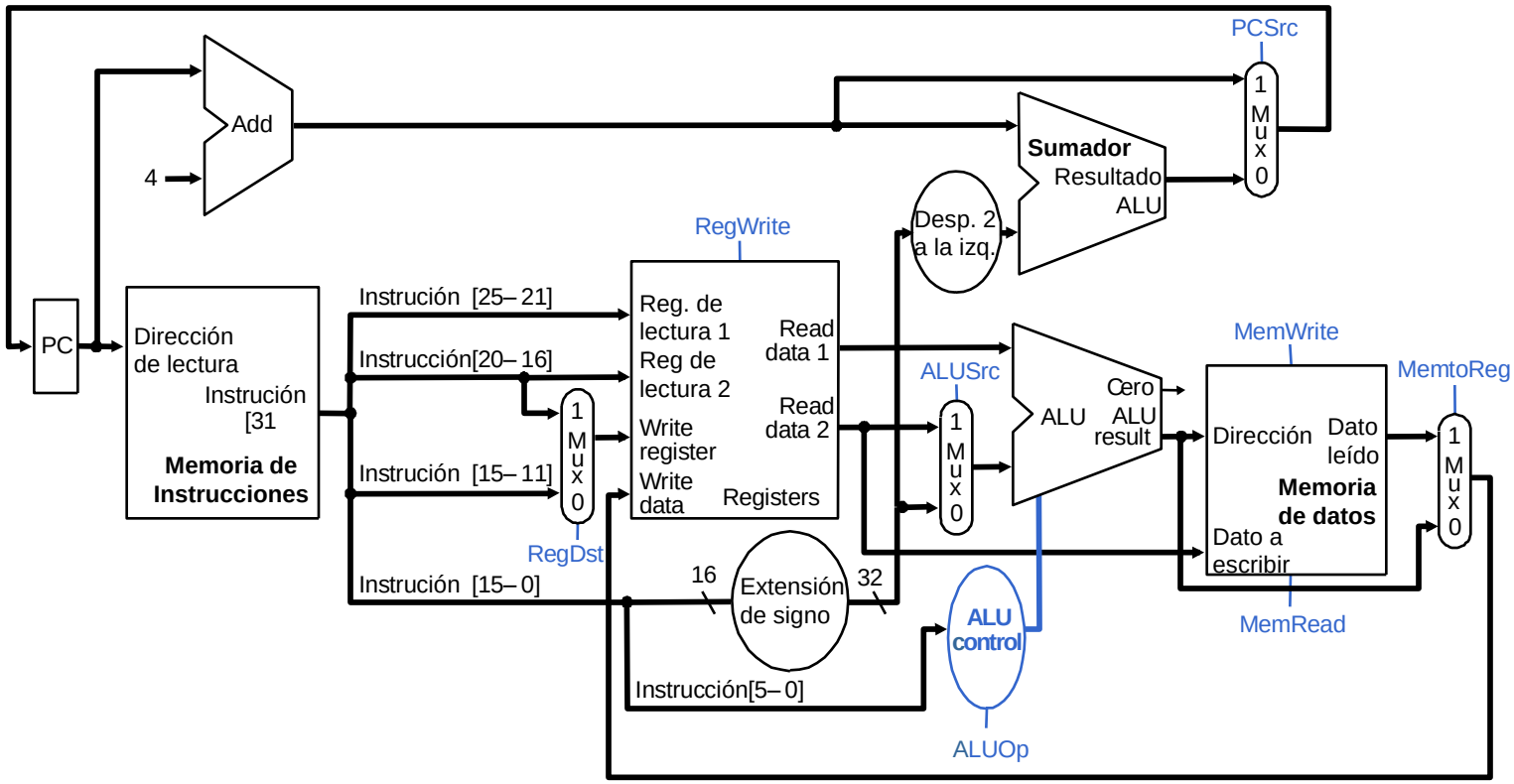
lw $t1, 100($t2) ($t1← M[$t2+100])



EJERCICIO

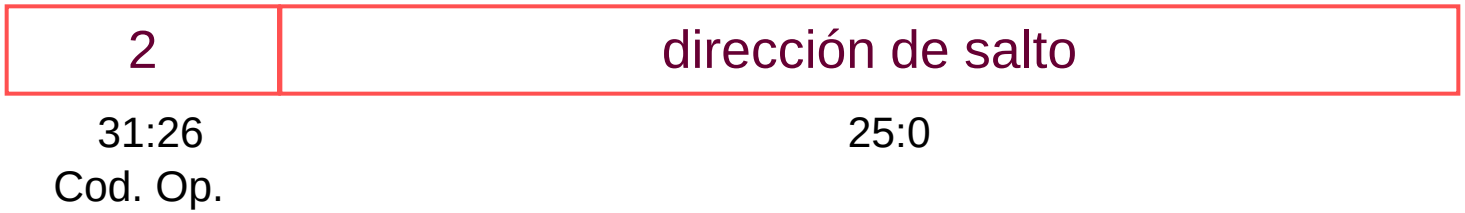
- Calcular el tiempo de ciclo suponiendo retardos despreciables para todos los elementos de la ruta de datos monociclo excepto para:

Acceso a memoria (2ns), ALU y sumadores (2ns), acceso al banco de registro (1ns)



AÑADIR INSTRUCCIÓN DE SALTO INCONDICIONAL

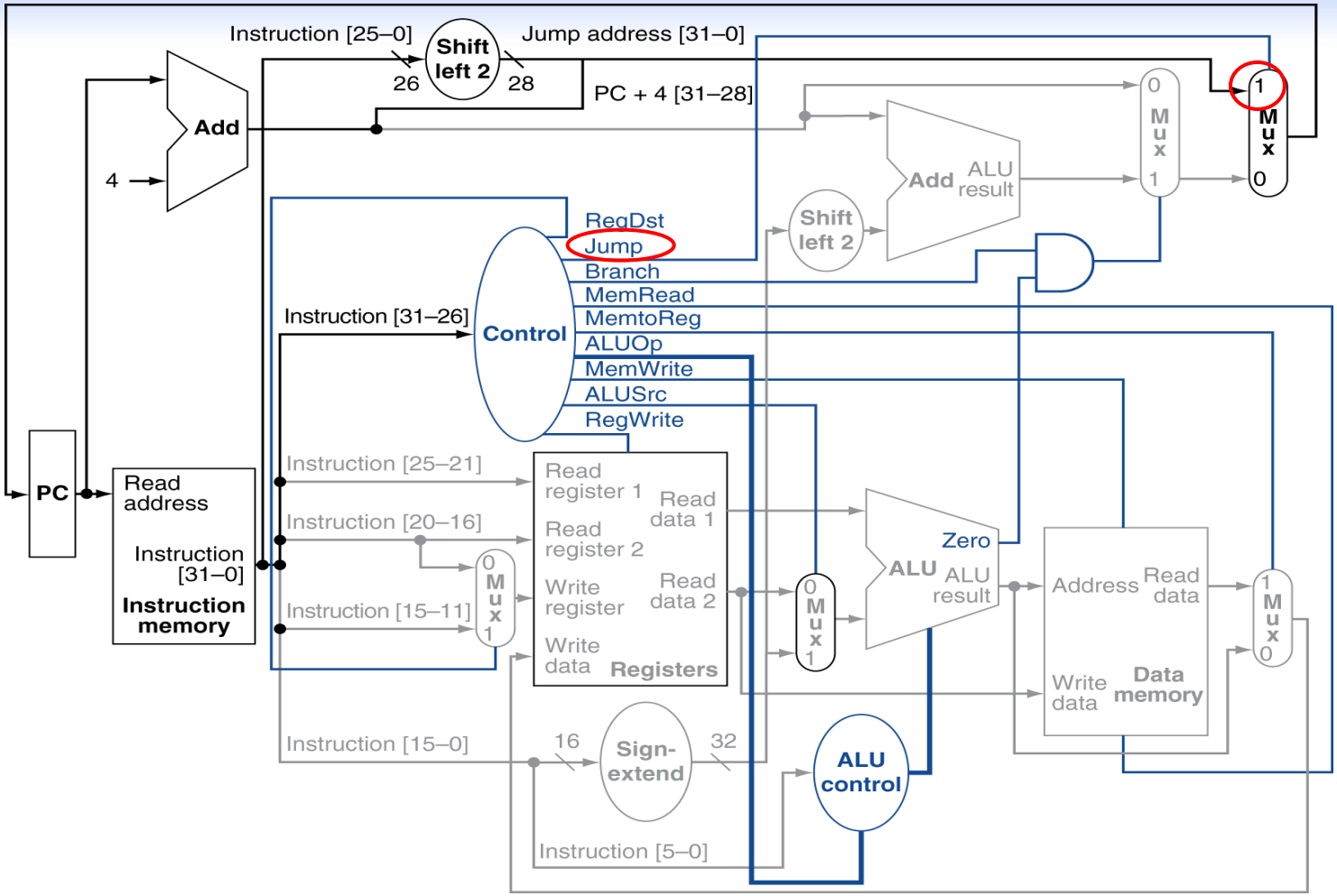
- Formato de la instrucción:



- La dirección de salto se forma con la concatenación de:
  - 4 bits superiores del PC actualizado (PC+4 calculado en la búsqueda de la instrucción)
  - los 26 bits de menor peso de la instrucción
  - 00
- Habrá que añadir una nueva señal de control que se active con la instrucción: ***Jump***

RUTA DE DATOS CON SALTO INCONDICIONAL

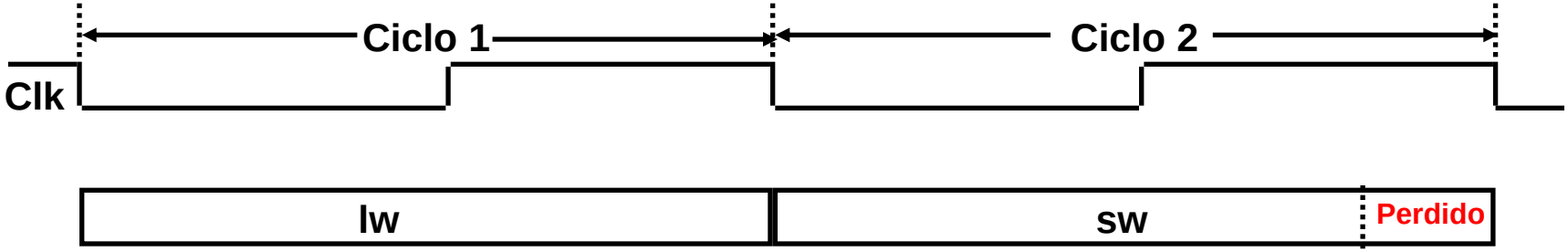
Esquema de implementación simple



Instrucción	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Jump	Branch	ALUOp1	ALUp0
J	X	X	X	0	0	0	1	0	X	X

PROBLEMAS IMPLEMENTACIÓN MONOCICLO

- La duración del ciclo de reloj viene determinado por la instrucción más lenta.



  - ¿Qué ocurriría si tuviéramos una instrucción muy complicada, por ejemplo una operación en coma flotante?

- No es viable duraciones del ciclo de reloj distintas para distintas instrucciones.

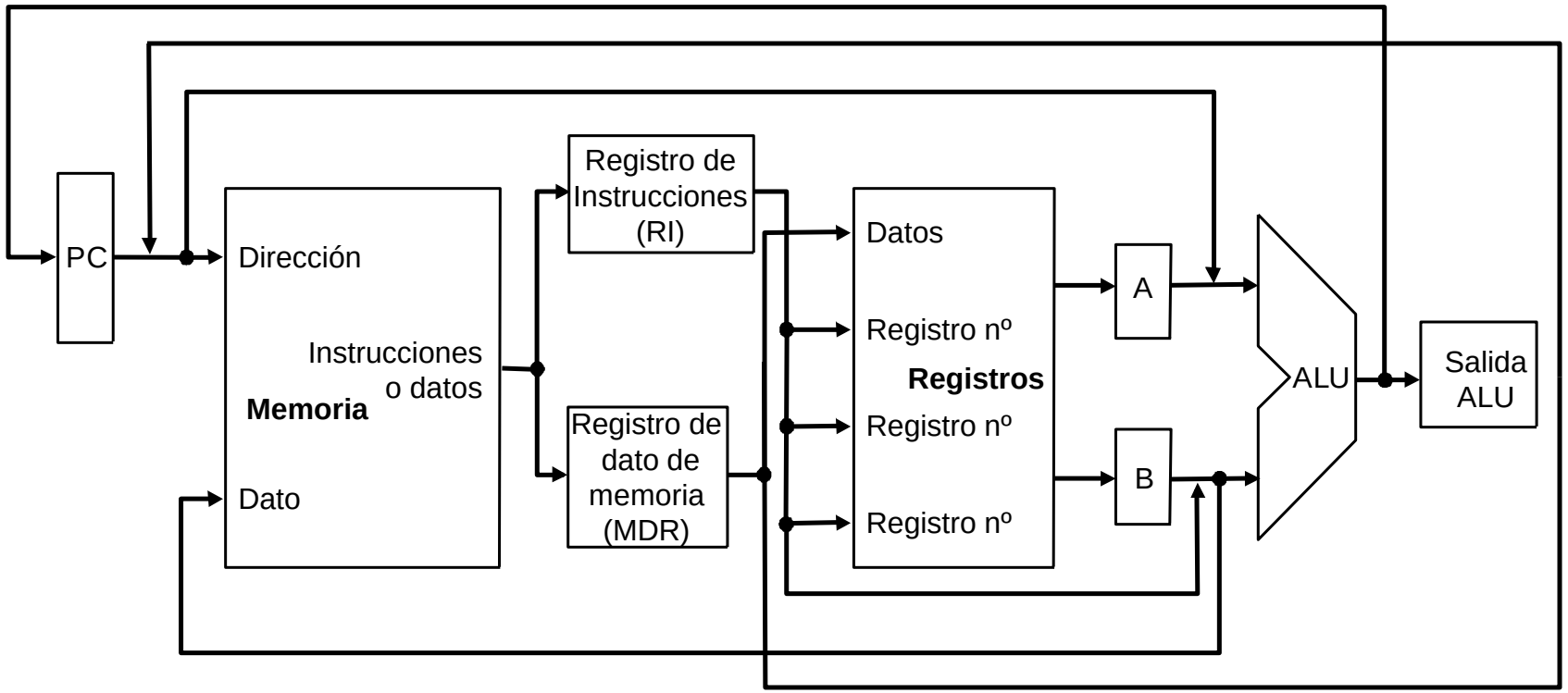
- Se utilizan mucho elementos en la ruta de datos y algunos de ellos se encuentran replicados.

SOLUCIÓN A LA IMPLEMENTACIÓN MONOCICLO

- Ruta de datos multiciclo
  - Utiliza un ciclo de reloj más corto.
  - Cada instrucción puede durar distintos ciclos de reloj
  - Se comparten los elementos entre las instrucciones
- Segmentación
  - La búsqueda y ejecución de la siguiente instrucción comienza antes de terminar la instrucción en curso.
- Procesamiento superescalar
  - Búsqueda y ejecución de varias instrucciones a la vez.
- Estas últimas técnicas se estudiarán el próximo curso.

VISIÓN GENERAL IMPLEMENTACIÓN MULTICICLO

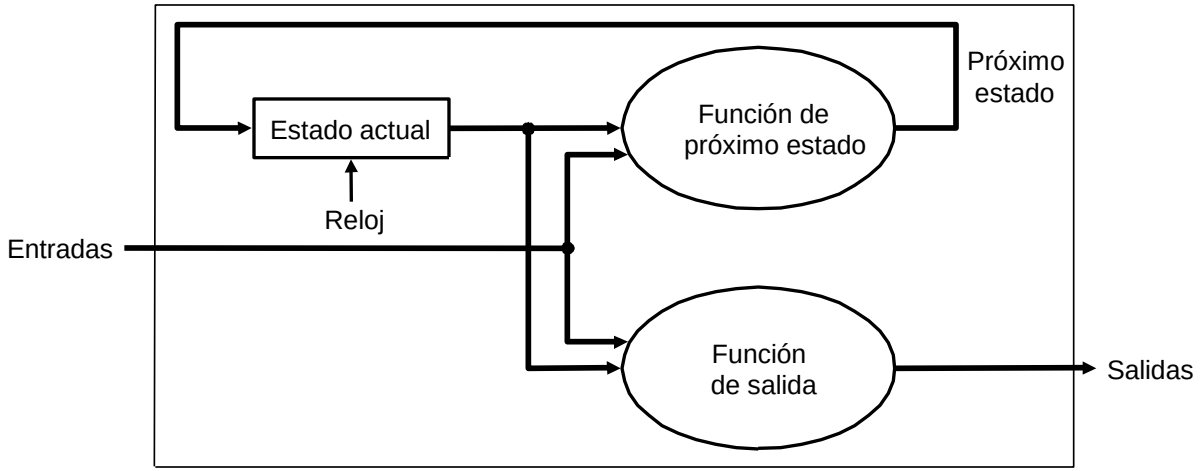
Esquema de implementación multiciclo



APROXIMACIÓN MULTICICLO

- Reutilizaremos las unidades funcionales
  - ALU utilizada para calcular la dirección y para incrementar PC
  - Memoria utilizada para albergar las instrucciones y datos
- Las señales de control no estarán únicamente determinadas por la instrucción
  - e.j.,¿Qué debería hacer la ALU para una instrucción de resta?
- Utilizaremos una máquina de estados finitos para especificar el control

- Formada por:
  - Un conjunto de estados
  - Función de próximo estado (determinado por el estado actual y la entrada)
  - Función de salida (determinado por el estado actual y la posible entrada)

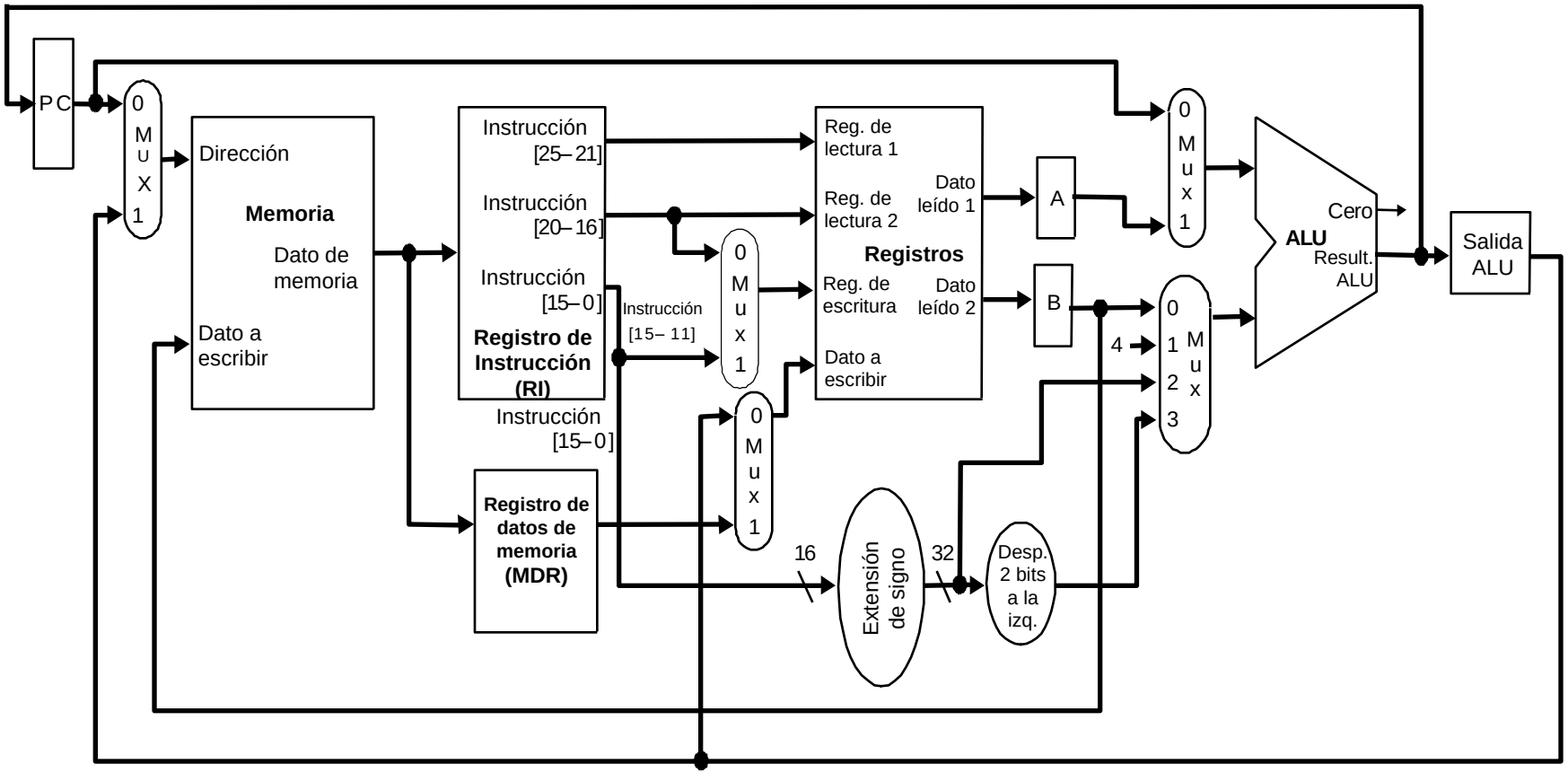


RUTA DE DATOS MULTICICLO

- Romper las instrucciones en pasos (o etapas), cada uno con una duración de un ciclo de reloj
  - Distribuir la cantidad de trabajo a realizar
  - Restricción: en cada ciclo sólo pueda utilizarse una unidad funcional
- Al final del ciclo
  - Almacenar valores para usarlos en ciclos posteriores (facilidad)
  - Introducir registros “internos” adicionales

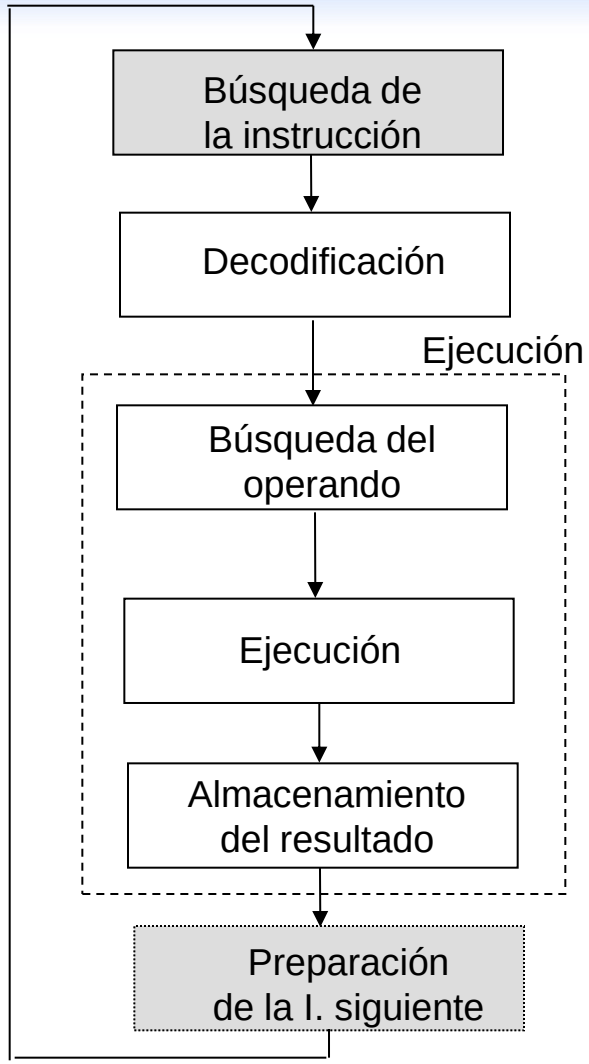
RUTA DE DATOS MULTICICLO

Esquema de implementación multiciclo

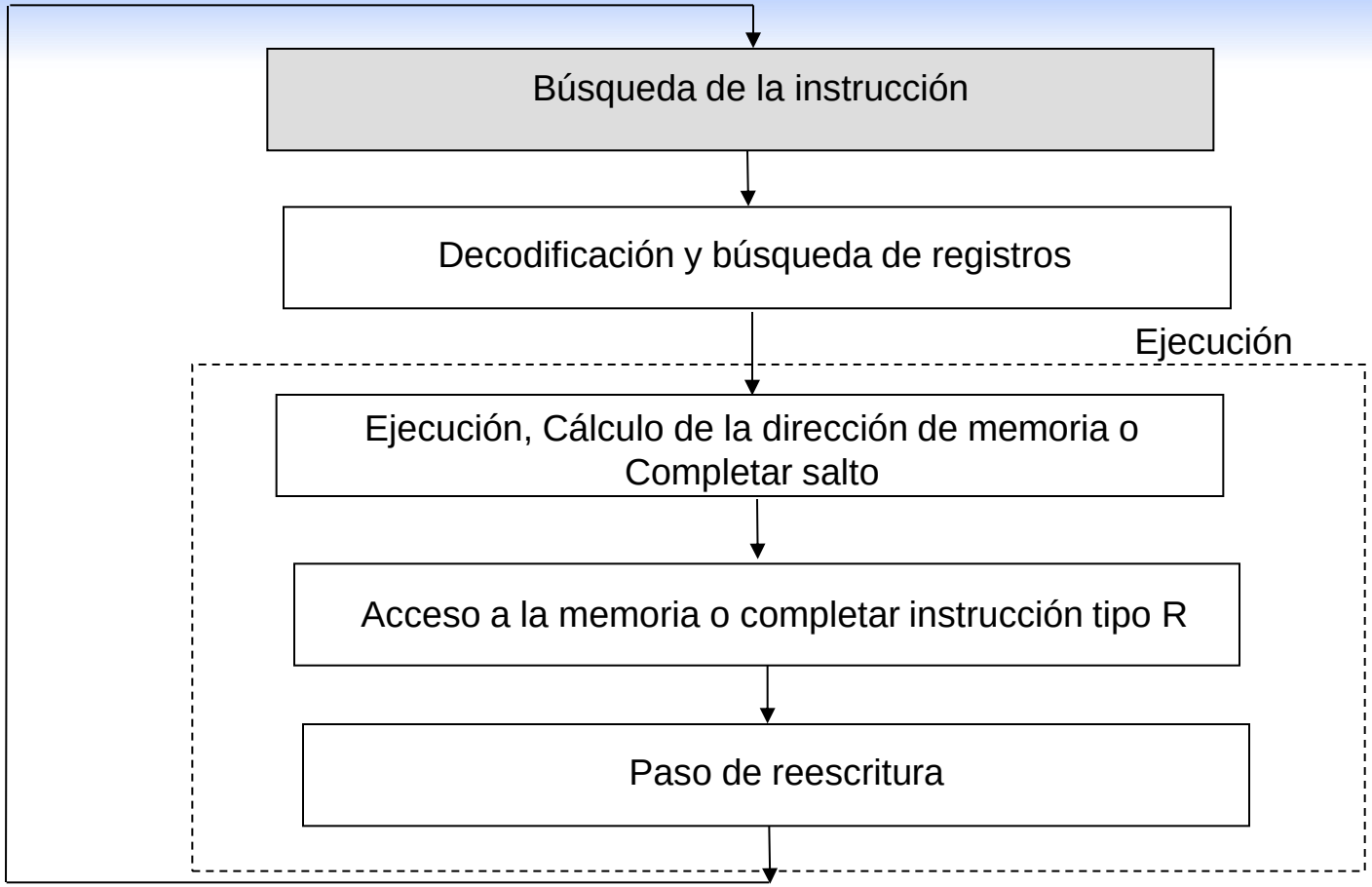


FASES DE EJECUCIÓN EN GENERAL

Esquema de implementación multiciclo



FASES DE EJECUCIÓN



LAS INSTRUCCIONES TARDARÁN DE 3 - 5 CICLOS!

ESTABLECIMIENTO DE LAS FASES (1)

- Fase 1: Búsqueda de la instrucción
  - Utilizamos el PC para buscar la instrucción y la guardamos en el registro de instrucción (RI).
  - Incrementados el PC en 4 y guardamos el resultado en el PC.

Se puede describir de forma sucinta utilizando RTL "Register-Transfer Language"

```
RI ← Memoria[PC];
PC ← PC + 4;
```

¿Podrías establecer el valor de las señales de control que se deben activar?

¿Qué ventaja tiene actualizar ahora el PC?

ESTABLECIMIENTO DE LAS FASES (2)

- Fase 2: Decodificación y búsqueda de registros
  - Leer los registros indicados por rs y rt y almacenarlos en A y B.
  - Calcular la dirección de salto (se ignorará si la instrucción no es de salto)

RTL:

```
        A ← Reg[RI[25-21]];
        B ← Reg[RI[20-16]];
ALUOut ← PC+(extensión-signo(RI[15-0])<< 2);
```

Estas dos fases son comunes a todas las instrucciones

ESTABLECIMIENTO DE LAS FASES (3)

- Fase 3: Ejecución, cálculo de la dirección de memoria o finalización del salto.

  La ALU está realizando uno de las tres siguientes funciones dependiendo de que instrucción se trate

  - Referencia a memoria:

    ```
    SalidaALU ← A + extensión-signo (IR[15-0]);
    ```

  - Tipo R:

    ```
    SalidaALU ← A op B;
    ```

  - Salto condicional:

    ```
    Si (A==B) PC ← SalidaALU;
    ```

ESTABLECIMIENTO DE LAS FASES (4)

- Fase 4: Acceso a memoria o finalización instrucciones tipo R.
  - Accesos a memoria de cargas y almacenamientos

```
MDR ← Memoria[SalidaALU];
```

o

```
Memoria[SalidaALU] ← B;
```

  - Finalización de la instrucción tipo R

```
Reg[RI[15-11]] ← SalidaALU;
```

ESTABLECIMIENTO DE LAS FASES (5)

- Fase 5: Finalización de lectura en memoria

```
Reg[IR[20-16]]← MDR;
```

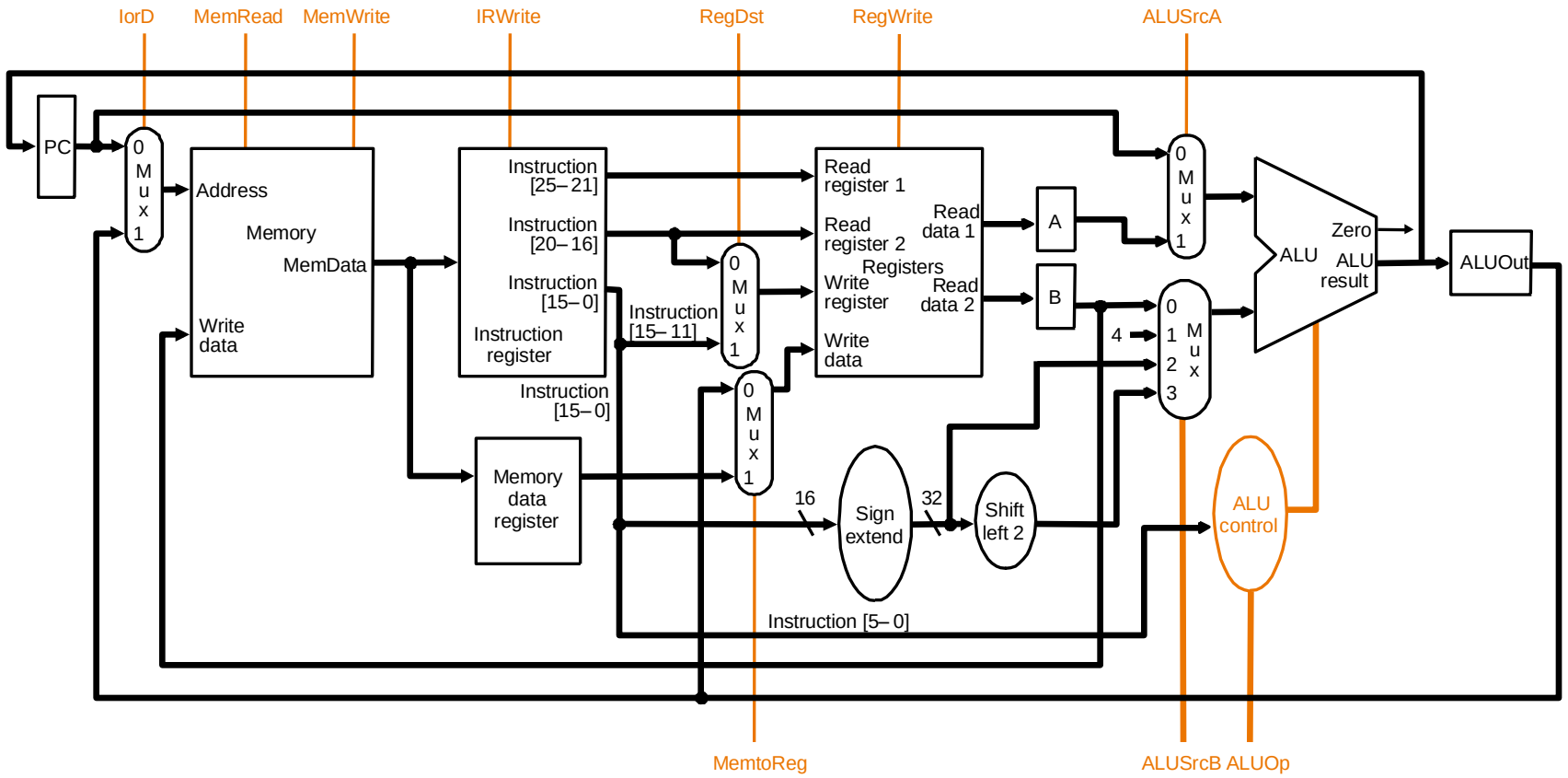
¿ Qué ocurre con el resto de las instrucciones?

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 Y SEÑALES DE CONTROL

Esquema de implementación multiciclo

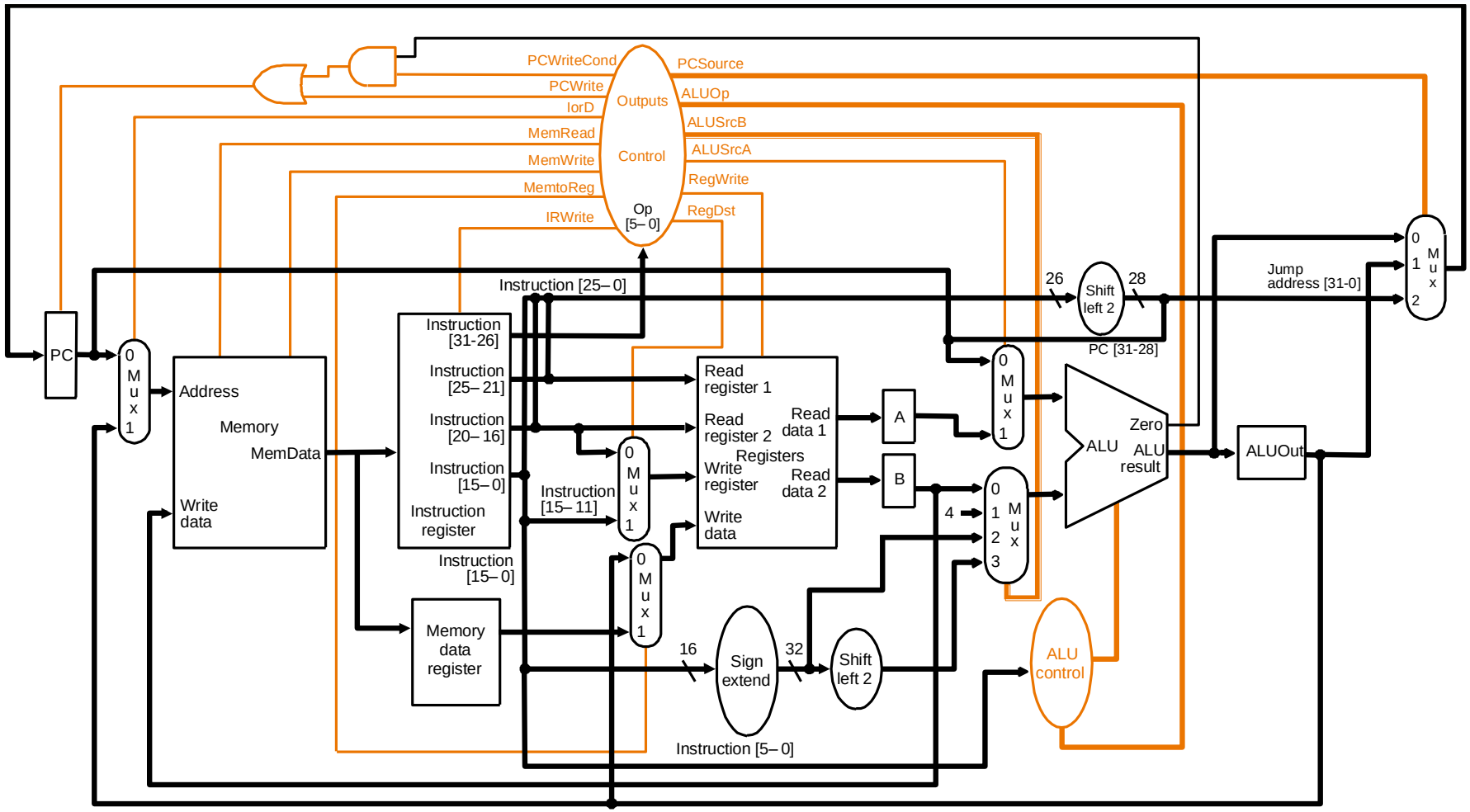


ACLARACIONES

- Las señales ALUOp y ALUSrcB son de dos bits, el resto de 1bit.
- Los registros A, B, MDR y SalidaALU se escriben en cada ciclo de reloj (no necesitan señal de control).
- En la ruta de datos anterior no se pueden ejecutar las instrucciones de salto condicional (**beq**) e incondicional (**j**ump), falta completarla:
  - En el registro PC se puede escribir el PC incrementado, o las direcciones de salto obtenidas para las instrucciones de salto (hay que añadir un multiplexor a su entrada)
  - La escritura del PC se puede hacer combinando dos señales de control:
    - una que escribe incondicionalmente en el PC (PCWrite) que se activa en la fase 1 o si es un salto incondicional.
    - Y otra que escribe en el PC si el salto es efectivo (PCWriteCond) que se activa si la instrucción es Beq.

RUTA DE DATOS COMPLETA Y CONTROL

Esquema de implementación multiciclo

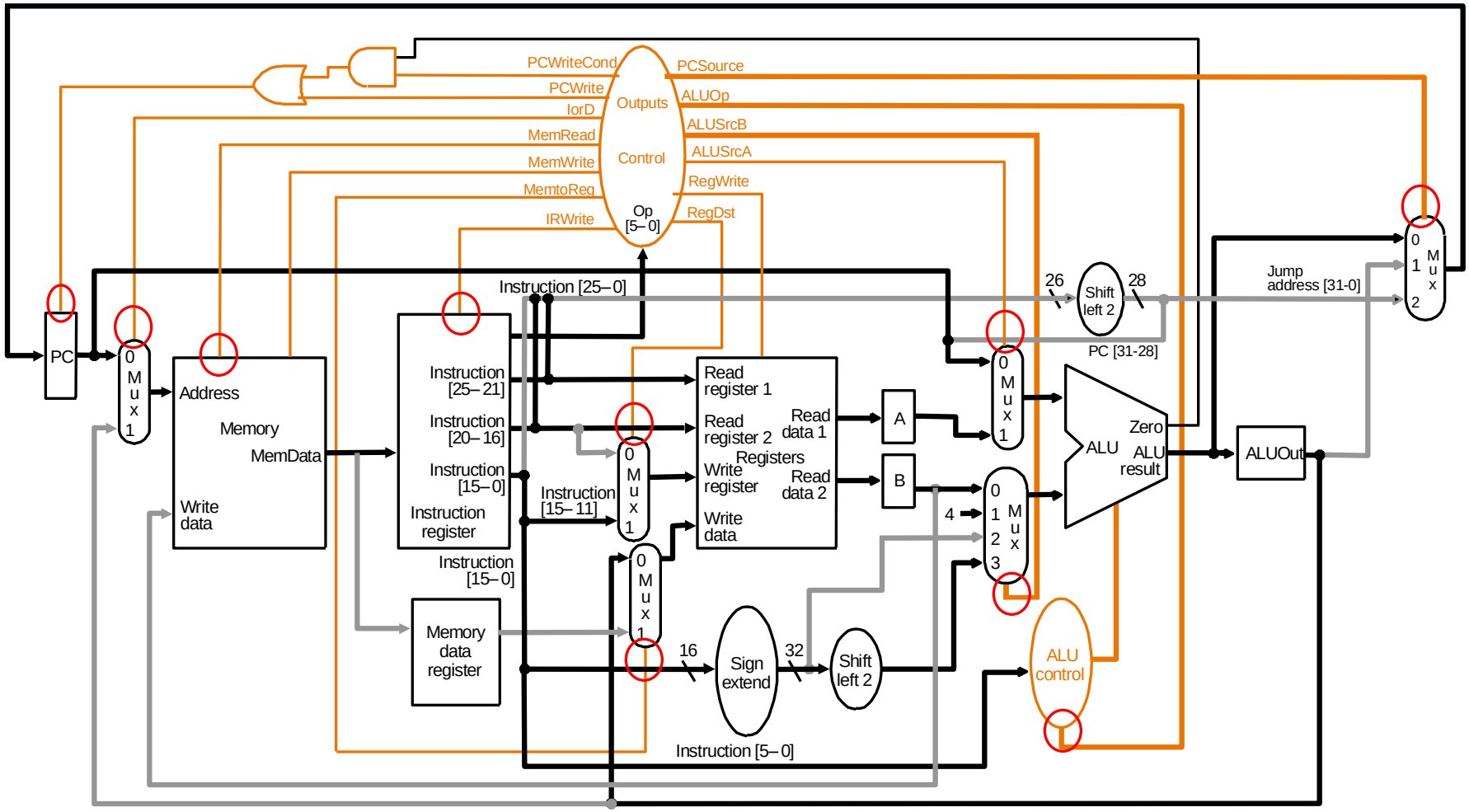


ACTIVACIÓN DE LAS SEÑALES DE CONTROL

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, PVWriteCond, 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

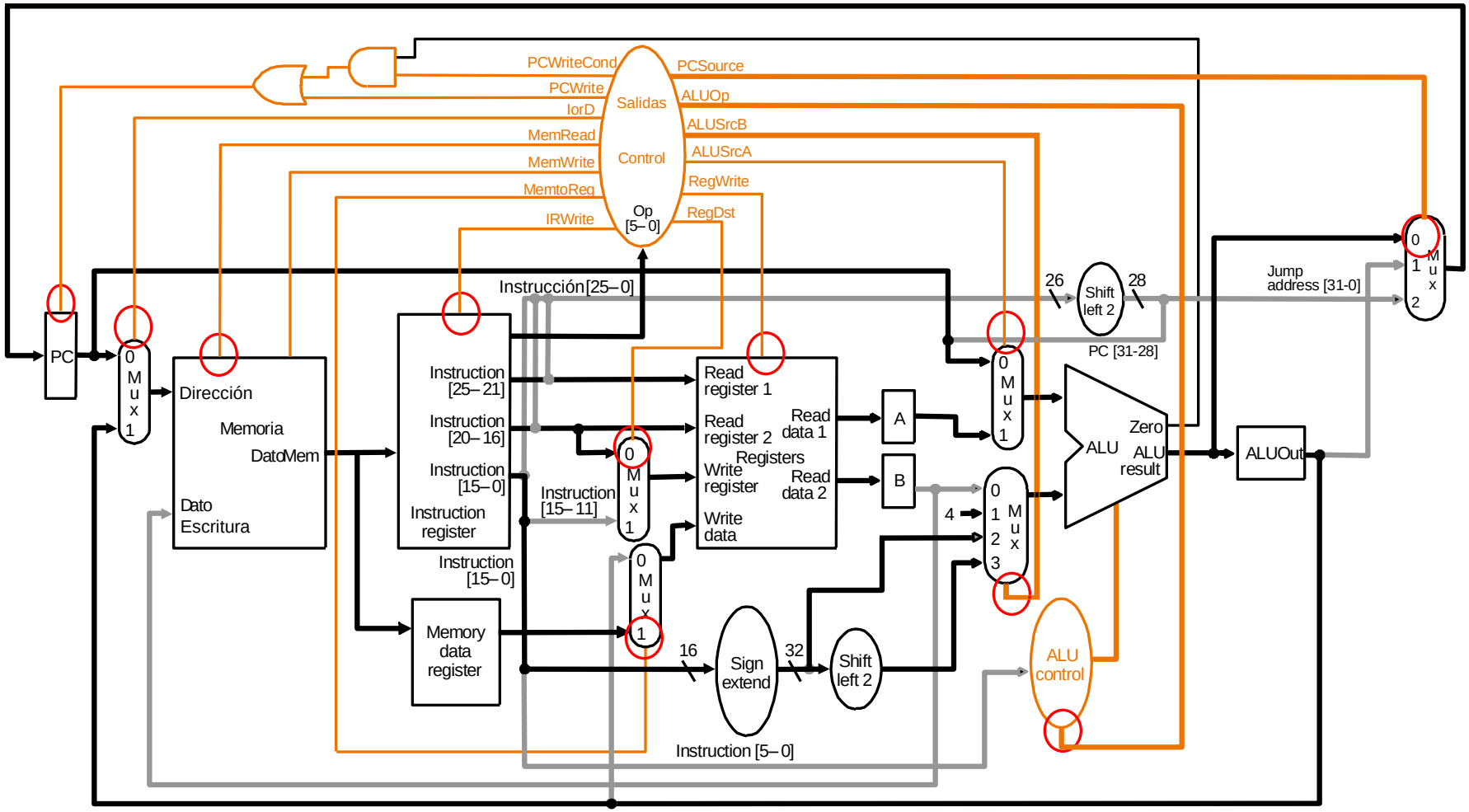
EJEMPLO: EJECUCIÓN INSTRUCCIONES TIPO R

Esquema de implementación multiciclo

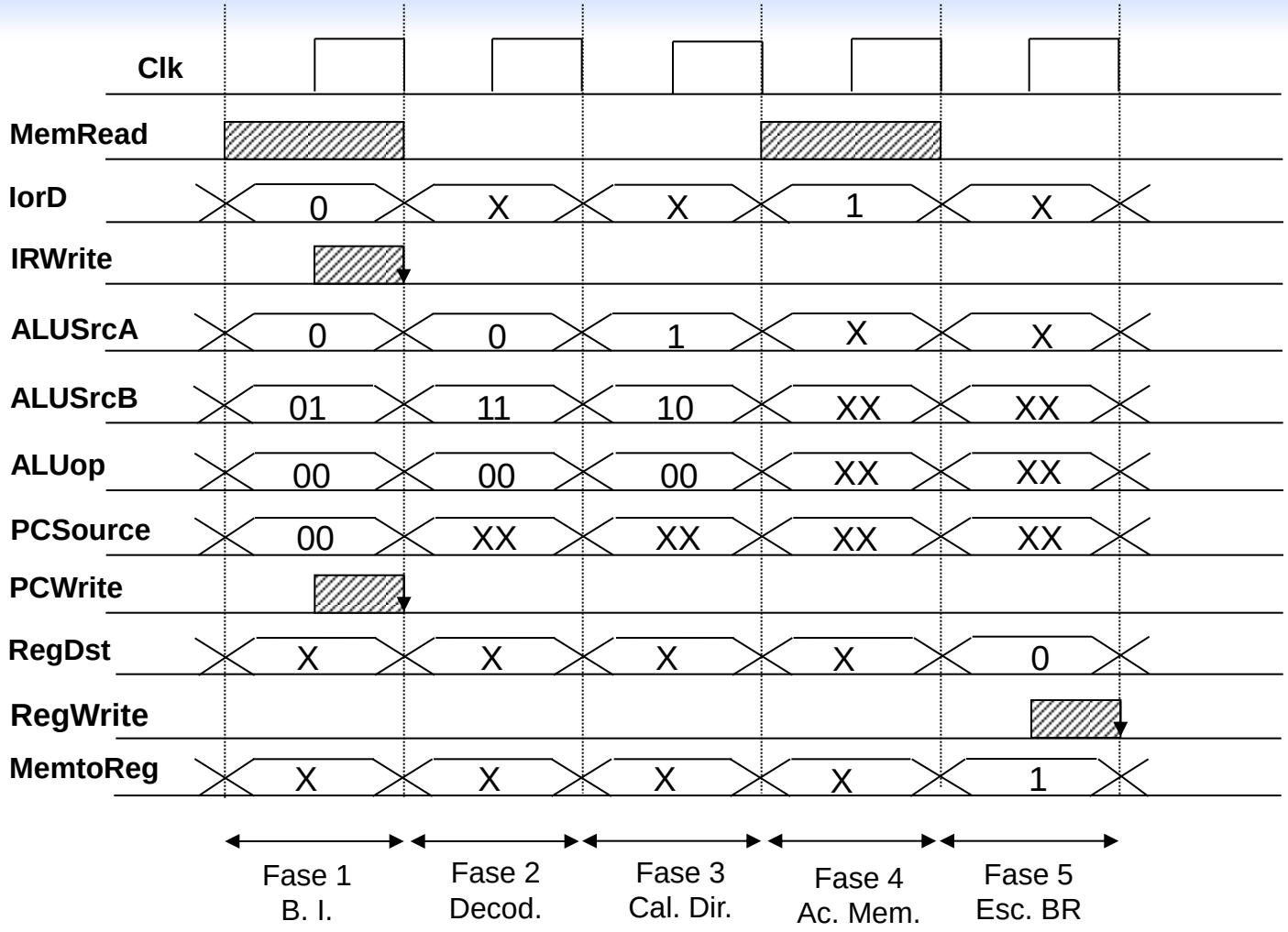


EJEMPLO: EJECUCIÓN INSTRUCCIÓN Lw

Esquema de implementación multiciclo



CRONGRAMA INSTRUCCIÓN LW



EJERCICIOS

- 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` ?

Método de la tabla de estados

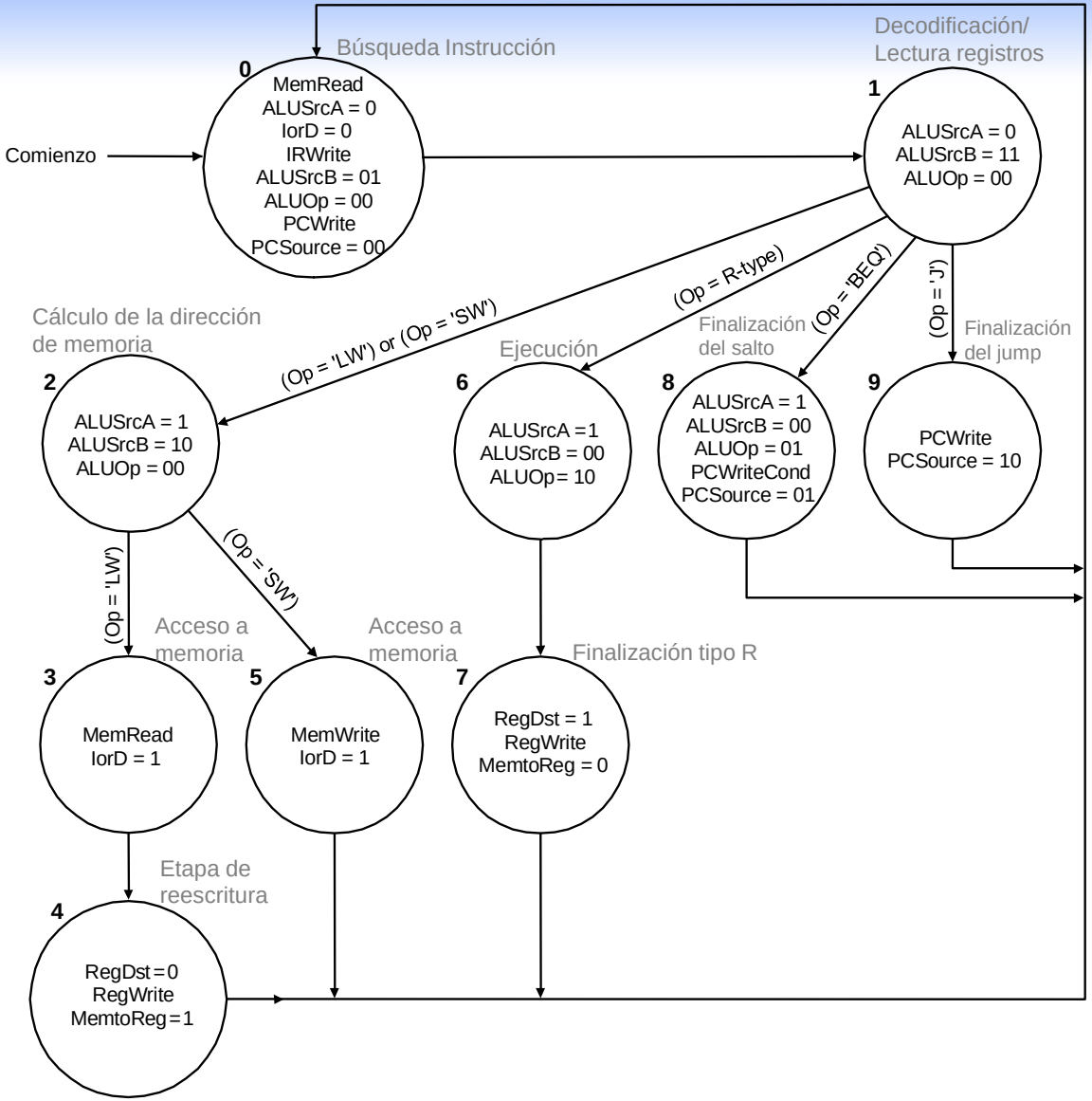
- Basada en una máquina de estados finitos
- Una máquina de estados finitos consta:
  - Memoria Interna que contiene el estado
    - Cada estado corresponde a un ciclo de reloj y contiene las operaciones a realizar en ese ciclo
  - Dos funciones combinacionales:
    - La función de estado siguiente
      - La función de estado siguiente es una función combinacional que a partir de las entradas y el estado actual determina el estado siguiente.
    - La función de salida
      - La función de salida produce el conjunto de señales de control a partir de sus entradas y el estado actual.

IMPLEMENTACIÓN DEL CONTROL

- El valor de las señales de control depende de:
  - Que instrucción se esté ejecutando
  - Que paso (etapa o fase) se está realizando
- Utilizaremos la información que hemos recogido hasta el momento para especificar la máquina de estados finitos
  - Especificar la máquina de estados finitos gráficamente, o
  - Utilizar microprogramación
- La implementación derivará de la especificación

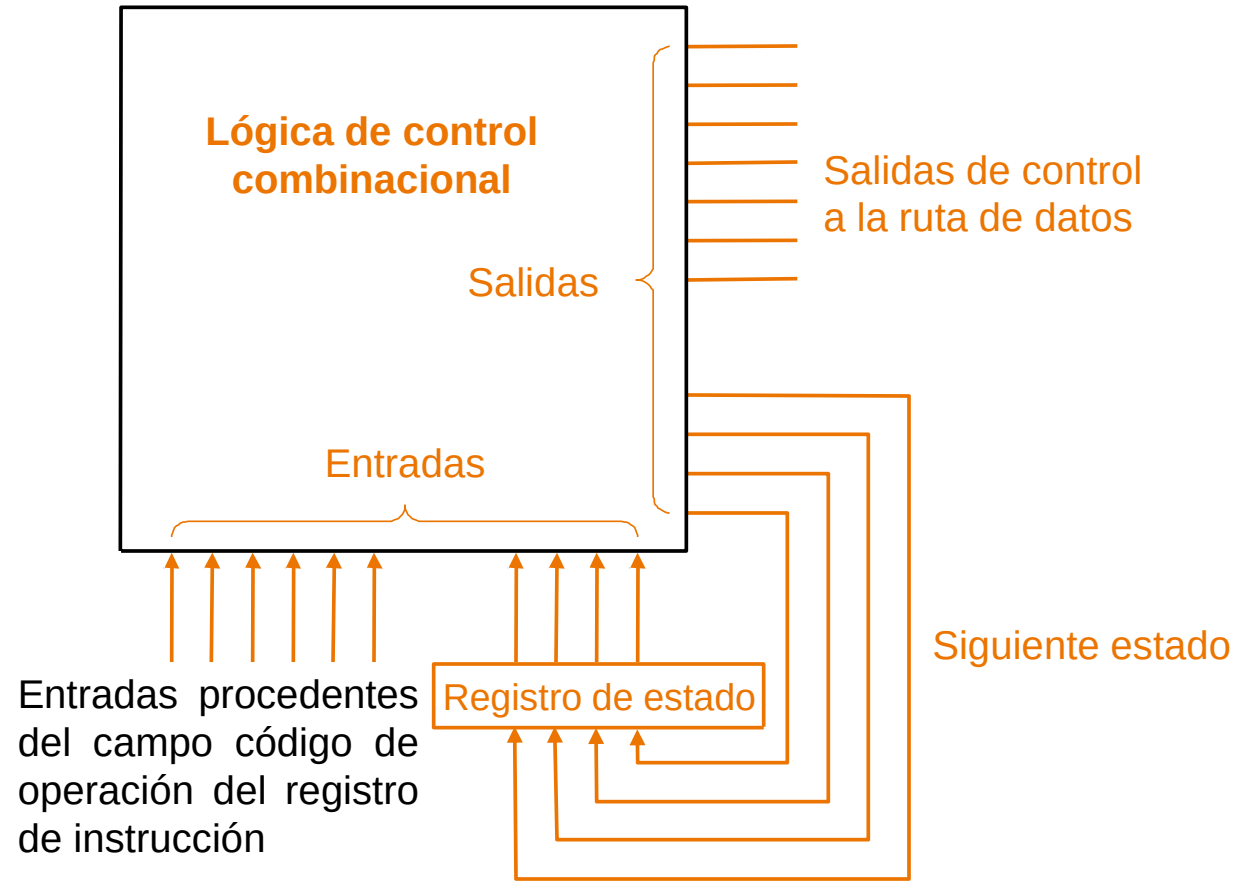
GRAFO DE ESTADOS

¿Cuántos bits necesitamos para el estado?



IMPLEMENTACIÓN DE LA MAQUINA DE ESTADOS FINITOS

Esquema de implementación multiciclo



ECUACIONES LÓGICAS PARA EL CONTROL

- Para las señales de control

Salidas	Estado Actual
PCWrite	estado0 + estado9
PCWriteCond	estado8
IorD	estado3 + estado5
MemRead	estado0 + estado3
MemWrite	estado5
IRWrite	estado0
MemtoReg	estado4
PCSource1	estado9
PCSource0	estado8
ALUOp1	estado6
ALUOp0	estado8
ALUSrcB1	estado1 + estado2
ALUSrcB0	estado0 + estado1
ALUSrcA	estado2 + estado6 + estado8
RegWrite	estado4 + estado7
RegDst	estado7

- Para el siguiente estado de forma comprimida

Salidas	Estado Actual	Código de Operación
SiguienteEstado0	estado4 + estado5 + estado7 +estado8 + estado9	
SiguienteEstado1	estado0	
SiguienteEstado2	estado1	(Op = 'lw') + (Op = 'sw')
SiguienteEstado3	estado2	(Op = 'lw')
SiguienteEstado4	estado3	
SiguienteEstado5	estado2	(Op = 'sw')
SiguienteEstado6	estado1	(Op = ‘Tipo R')
SiguienteEstado7	estado6	
SiguienteEstado8	estado1	(Op = 'beq')
SiguienteEstado9	estado1	(Op = 'jump')

- Obtener la ecuación lógica para el bit de menor peso de SiguienteEstadoN

  - Solución

Utilizaremos 4 bits para codificar tanto el estado actual (bits E3, E2, E1,y E0) como el siguiente estado (bits SE3, SE2, SE1 y SE0). Por ejemplo, SiguienteEstado6 se codificará con SiguienteEstado=0110= $\overline{SE3}\cdot SE2\cdot SE1\cdot\overline{SE0}$

El bit de menor peso estará a 1 en SiguienteEstado1, SiguienteEstado3, SiguienteEstado5, SiguienteEstado7 y SiguienteEstado9. Las ecuaciones para cada uno estos estados se obtiene de la tabla anterior:

$$SiguienteEstado1 = Estado0 = \overline{E3}\cdot\overline{E2}\cdot\overline{E1}\cdot\overline{E0}$$

$$SiguienteEstado3 = Estado2\cdot(Op[5..0] = Lw) = \overline{E3}\cdot\overline{E2}\cdot E1\cdot\overline{E0}\cdot Op5\cdot\overline{Op4}\cdot\overline{Op3}\cdot\overline{Op2}\cdot Op1\cdot Op0$$

$$SiguienteEstado5 = Estado2\cdot(Op[5..0] = sw) = \overline{E3}\cdot\overline{E2}\cdot E1\cdot\overline{E0}\cdot Op5\cdot\overline{Op4}\cdot Op3\cdot\overline{Op2}\cdot Op1\cdot Op0$$

$$SiguienteEstado7 = Estado6 = \overline{E3}\cdot E2\cdot E1\cdot\overline{E0}$$

$$SiguienteEstado9 = Estado1\cdot(Op[5..0] = jump) = \overline{E3}\cdot\overline{E2}\cdot\overline{E1}\cdot E0\cdot\overline{Op5}\cdot\overline{Op4}\cdot\overline{Op3}\cdot\overline{Op2}\cdot Op1\cdot\overline{Op0}$$

El bit SE0 es la suma lógica de todos los términos.

TABLA DE SALIDA

Tabla de verdad para las señales de control que dependen sólo de los estados

Señales Control	Estado (E[3..0])									
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
PCWrite	1	0	0	0	0	0	0	0	0	1
PCWriteCond	0	0	0	0	0	0	0	0	1	0
IorD	0	0	0	1	0	1	0	0	0	0
MemRead	1	0	0	1	0	0	0	0	0	0
MemWrite	0	0	0	0	0	1	0	0	0	0
IRWrite	1	0	0	0	0	0	0	0	0	0
MemtoReg	0	0	0	0	1	0	0	0	0	0
PCSource1	0	0	0	0	0	0	0	0	0	1
PCSource0	0	0	0	0	0	0	0	0	1	0
ALUOp1	0	0	0	0	0	0	1	0	0	0
ALUOp0	0	0	0	0	0	0	0	0	1	0
ALUSrcB1	0	1	1	0	0	0	0	0	0	0
ALUSrcB0	1	1	0	0	0	0	0	0	0	0
ALUSrcA	0	0	1	0	0	0	1	0	1	0
RegWrite	0	0	0	0	1	0	0	1	0	0
RegDst	0	0	0	0	0	0	0	1	0	0

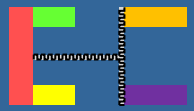


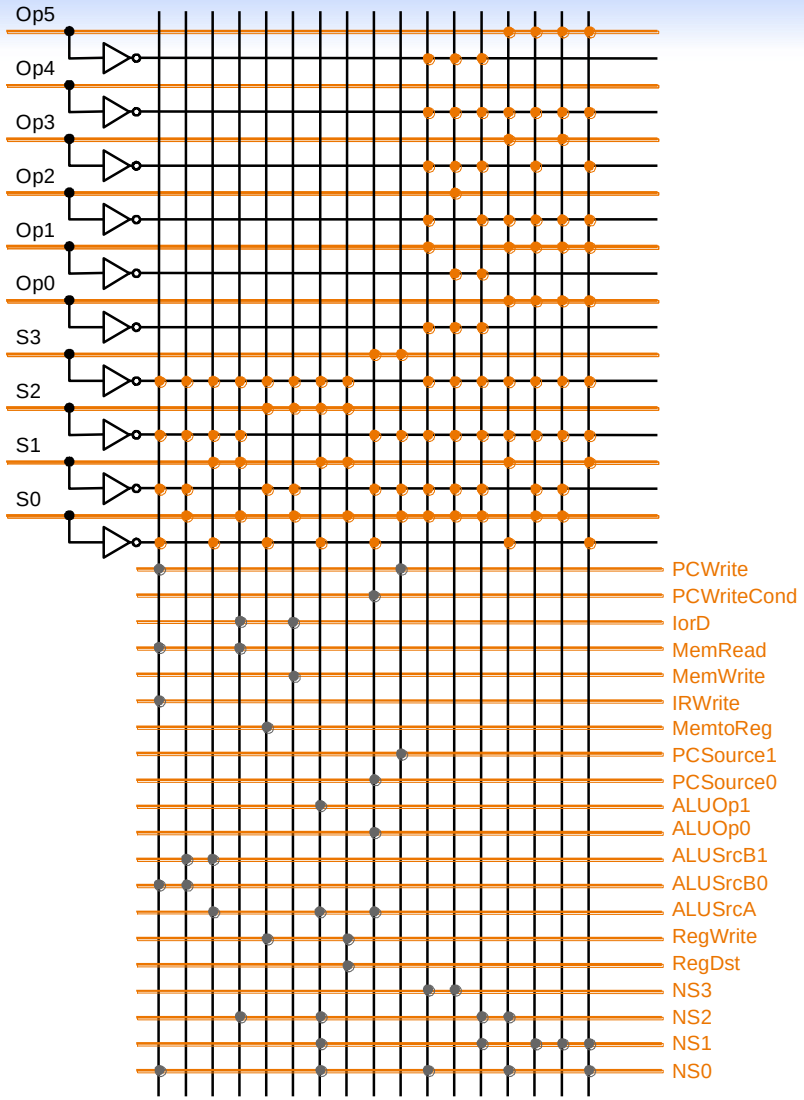
TABLA DE SIGUIENTE ESTADO

Tabla de verdad para el siguiente estado, depende de la instrucción.

Estado Actual E[3..0]	Op[5..0]				
	000000 (Formato R)	000010 (jump)	000100 (beq)	100011 (lw)	101011 (sw)
0000	0001	0001	0001	0001	0001
0001	0110	1001	1000	0010	0010
0010	XXXX	XXXX	XXXX	0011	0101
0011	0100	0100	0100	0100	0100
0100	0000	0000	0000	0000	0000
0101	0000	0000	0000	0000	0000
0110	0111	0111	0111	0111	0111
0111	0000	0000	0000	0000	0000
1000	0000	0000	0000	0000	0000
1001	0000	0000	0000	0000	0000

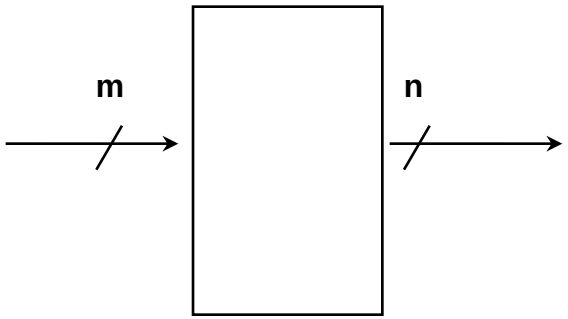
IMPLEMENTACIÓN CON UN PLD

Esquema de implementación multiciclo



IMPLEMENTACIÓN CON ROM

- ROM = "Read Only Memory"
  - Los valores de las localizaciones de la memoria están fijados previamente
- Se puede utilizar una ROM para implementar una tabla de verdad
  - Si la dirección tiene m-bits, podemos direccionar 2^m entradas en la ROM
  - Las salidas serán los bits de datos apuntados por la dirección.

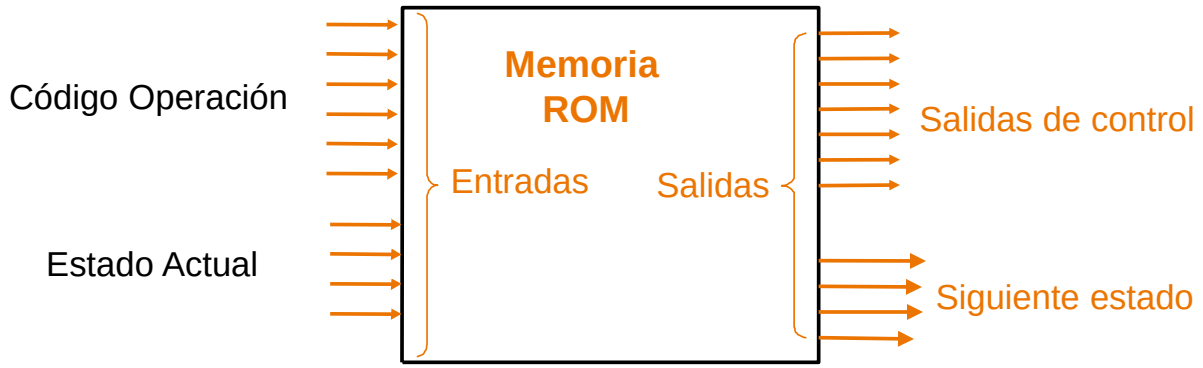


m es la “altura”, y n es la “anchura”

0	0	0	0	0	1	1
0	0	1	1	1	0	0
0	1	0	1	1	0	0
0	1	1	1	0	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	1
1	1	0	0	1	1	0
1	1	1	0	1	1	1

IMPLEMENTACIÓN CON ROM

- Las entradas a la unidad de control son las líneas de dirección de la ROM
- El ancho de la palabra de la memoria corresponde al número de salidas de la Unidad de Control.



IMPLEMENTACIÓN CON ROM

- ¿Cuántas entradas hay?

  6 bits para el código de operación, 4 bits para el estado = 10 líneas de dirección
  (es decir, 2^{10} = 1024 direcciones diferentes)

- ¿Cuántas salidas hay?

  16 salidas de control a la ruta de datos, 4 bits para el estado = 20 salidas

La ROM es 2^{10} x 20 = 20K bits (un tamaño bastante inusual)

Bastante grande, ya que para muchas entradas las salidas son las mismas

ya que el código de operación es a menudo ignorado

EJEMPLO

- ¿Contenido de las memoria ROM para los 16 bits más altos de la palabra de memoria?
  - Solución

PCWrite está a nivel alto en los estados 0 y 9; el estado actual corresponde a los 4 bits más bajos de la dirección de memoria (los bits más altos de la dirección de la ROM corresponden al código de operación),
las direcciones de la memoria ROM que tengan el bit más alto de la palabra en memoria serán:

0000000000, 0000001001, 0000010000, 0000011001, …, 1111110000 y 1111111001.

Es decir las posiciones de memoria XXXXXX0000 y XXXXXX1001.

EJEMPLO

- ¿Para qué direcciones de la ROM el bit correspondiente a la señal de control PCWrite (que será el bit de mayor peso de la palabra de control) estará a 1?
  - Solución

Se obtiene directamente de la tabla de salida. Este conjunto de palabras estará repetido 64 veces en toda la ROM para las distintas combinaciones de la parte alta de la dirección.

Dirección de memoria	Bits 19..4 de la palabra
XXXXXX 0000	1001010000001000
XXXXXX 0001	0000000000011000
XXXXXX 0010	0000000000010100
XXXXXX 0011	0011000000000000
XXXXXX 0100	0000001000000010
XXXXXX 0101	0010100000000000
XXXXXX 0110	0000000001000100
XXXXXX 0111	0000000000000011
XXXXXX 1000	0100000010100100
XXXXXX 1001	1000000100000000

ROM vs CIRCUITO LÓGICO

- Se pueden utilizar dos ROMs para el control
  - 4 bits del estado nos dan 16 salidas, 2^4 x 16 bits de ROM
  - 10 bits para los 4 bits del siguiente estado, 2^{10} x 4 bits de ROM
  - Total: 4.3K bits de ROM
- PLD es mucho más pequeño
  - puede compartir términos productos
  - solo necesita entradas que produzcan una salida activa
  - se pueden tener en cuenta los términos X
- Tamaño es (entradas × términos producto) + (salidas × términos producto)
  - Para el ejemplo = (10x17)+(20x17) = 460 celdas PLD