Diseño de Procesadores: Camino de datos y control

Procesador MIPS

Un microprocesador MIPS R4400 fabricado por Toshiba

Con el nombre de **MIPS** siglas de (**M***icroprocessor without* **I***nterlocked* **P***ipeline* **S***tages)* se conoce a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies.

Los diseños del MIPS son utilizados en la línea de productos informáticos de SGI (Silicon Graphics); en muchos sistemas embebidos; en dispositivos para Windows CE (S.O. para sistemas embebidos); routers Cisco; y videoconsolas como la Nintendo 64 o las PlayStation de Sony.

Procesador MIPS

Debido a que los diseñadores crearon un conjunto de instrucciones muy claro, los cursos sobre arquitectura de computadores en universidades y escuelas técnicas a menudo se basan en la arquitectura MIPS. El diseño de la familia de CPU's MIPS influiría de manera importante en otras arquitecturas RISC posteriores como los DEC Alpha.

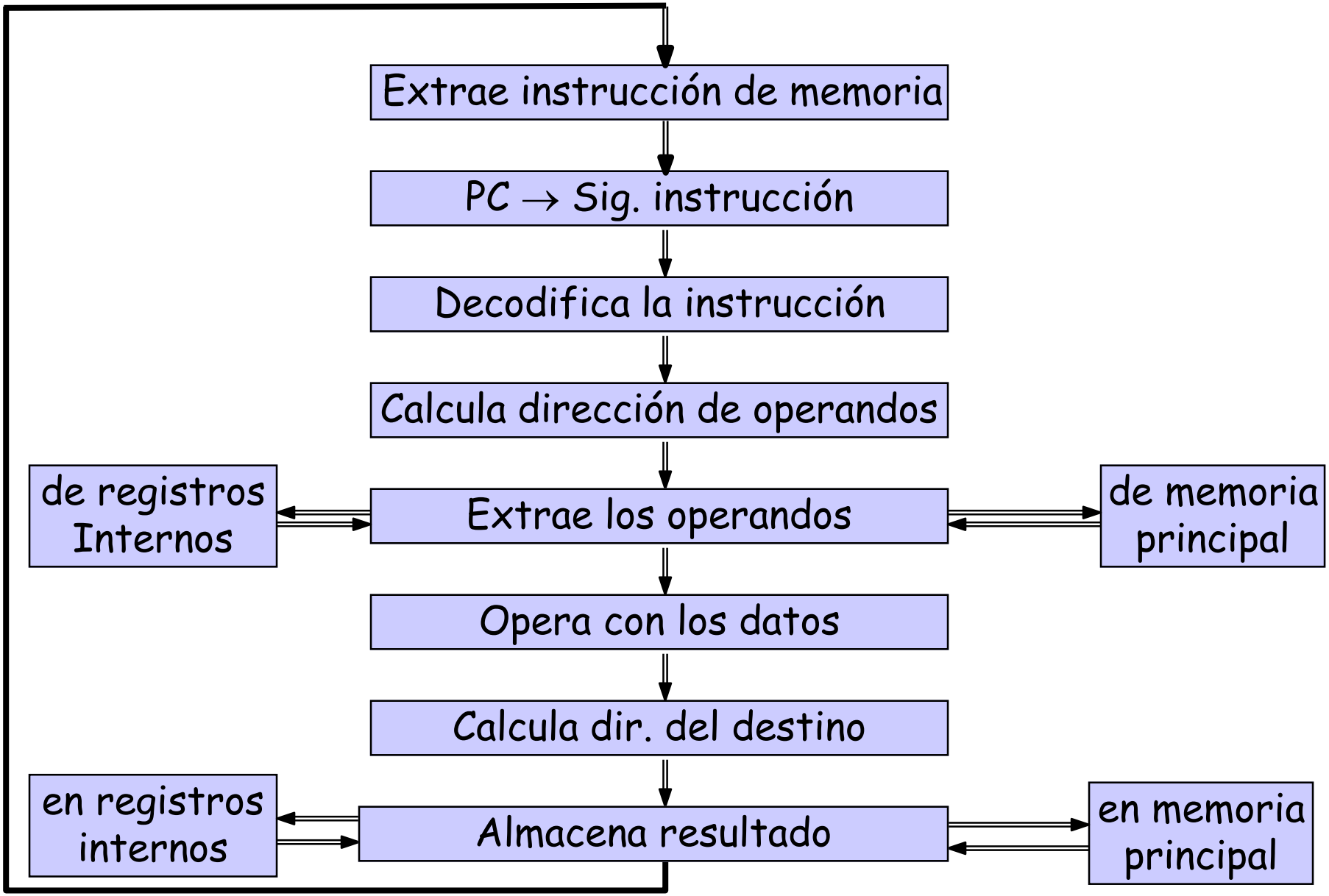
¿Qué vemos?

- Principales Unidades Funcionales.
- Camino de datos de los distintos tipos de instrucciones.
- Estado y cambio de estado.
- Sincronización mediante reloj.
- Camino de datos y señales de control.

Implementación en un solo ciclo

- Existe un único ciclo y es igual a la demora del camino más largo
- Un load usa cinco unidades.
- Las instrucciones más cortas no se benefician.
- El costo a pagar es mayor con las instrucciones de punto flotante
- Cada unidad funcional puede ser usada una sola vez (x inst.)

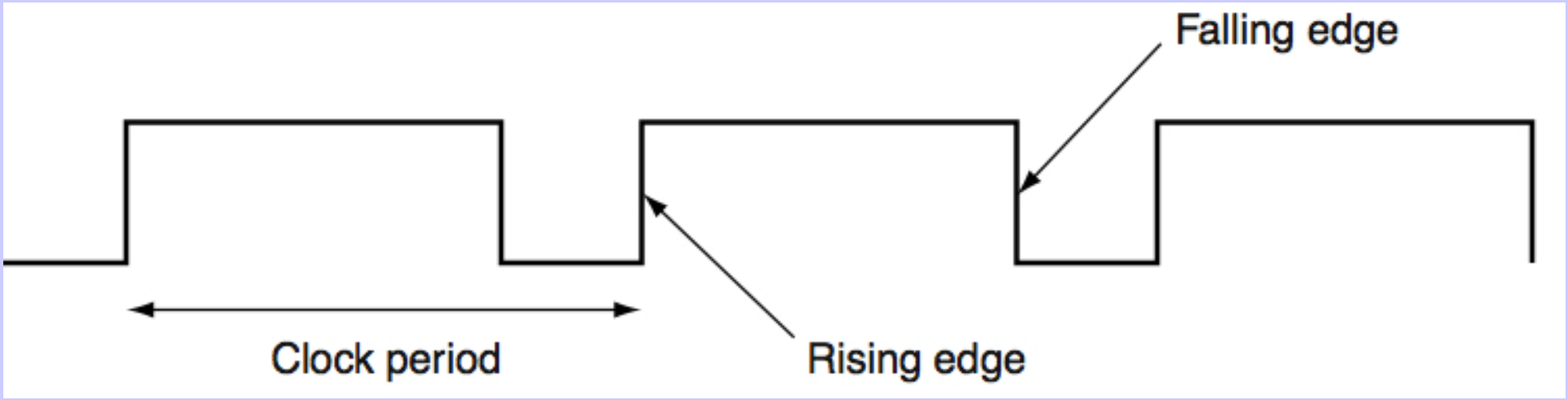
La CPU Ciclo de Instrucción



Diseño del procesador uniciclo

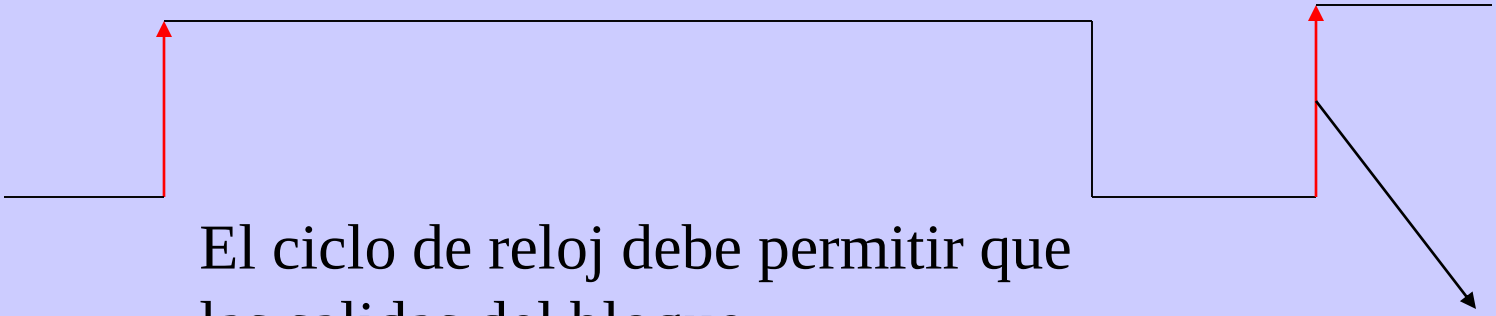
- Cada instrucción se ejecuta en un único ciclo de reloj
- CPI=1 para todas las instrucciones
- El período de reloj debe ser el de la instrucción más costosa: camino crítico
- Diseño poco flexible, pero fácil de comprender Y de implementar!!

Señal del reloj



Disparo por flancos – (edge-triggered)

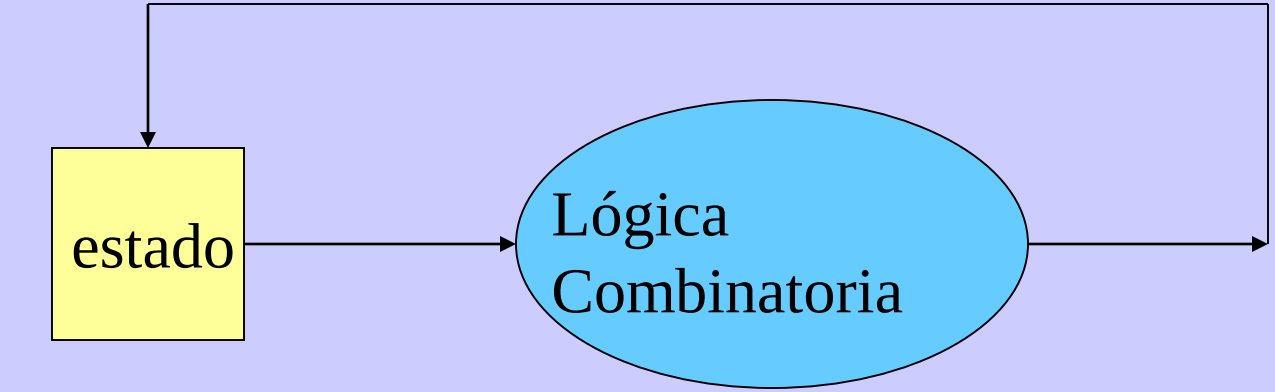
Metodología de sincronización
Sincronización por reloj

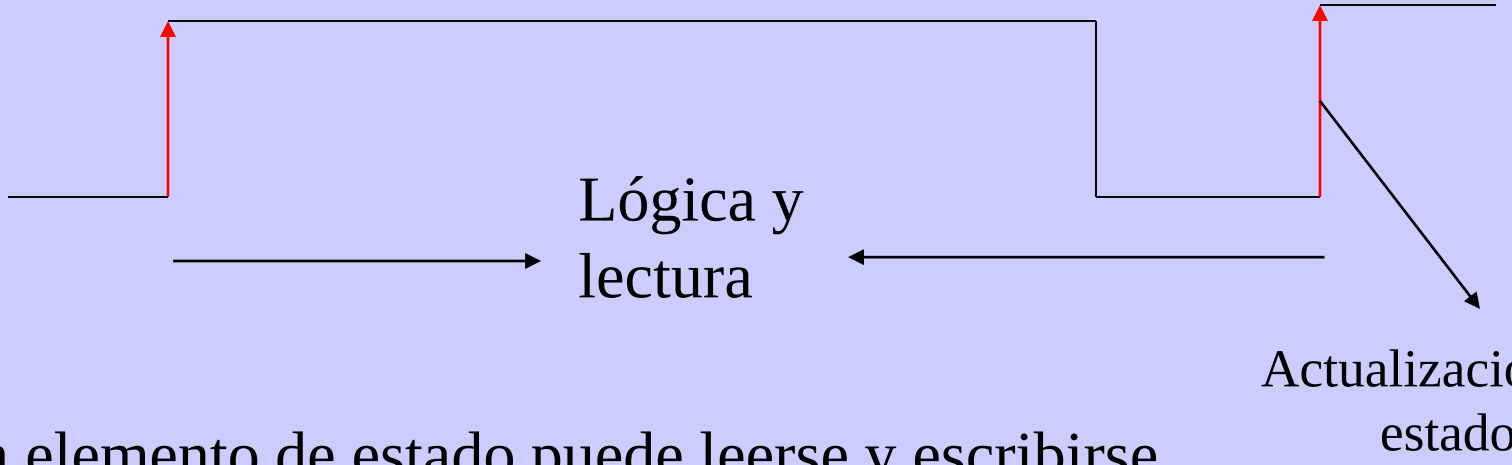


El ciclo de reloj debe permitir que las salidas del bloque combinacional se estabilicen

Actualización del estado

Metodología de sincronización
Cambio de estado





Un elemento de estado puede leerse y escribirse en el mismo ciclo.

Diseño: Pasos necesarios

1er Paso: Analizar el conjunto de instrucciones para determinar los requerimientos del Camino de Datos

2º Paso: Seleccionar los componentes

3er Paso: Construir el Camino de Datos según los requerimientos

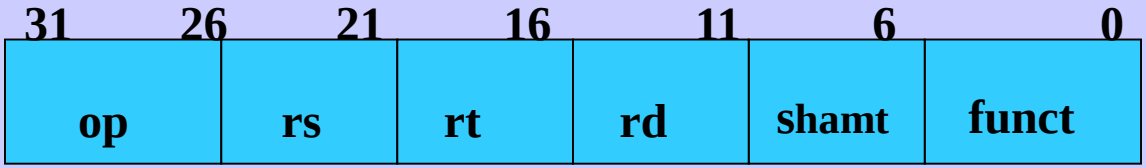
4º Paso: Analizar la implementación de cada instrucción para determinar las señales de control necesarias

5º Paso: Construir el Control.

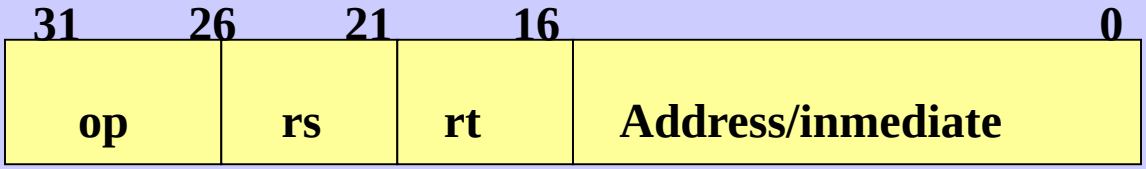
1er Paso: Análisis del formato de Instrucción MIPS

Son todas de 32 bits y las hay en Tres formatos:

- Tipo R



- Tipo I



- Tipo J

op: identificador de instrucción

rs, rt, rd: identificadores de los registros fuentes y destino

shamt: cantidad a desplazar (en operaciones de desplazamiento)

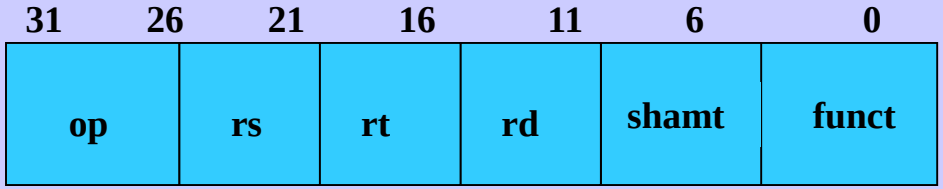
funct: selecciona la operación aritmética a realizar

inmediato: operando inmediato o desplazamiento en direccionamiento con base en registro

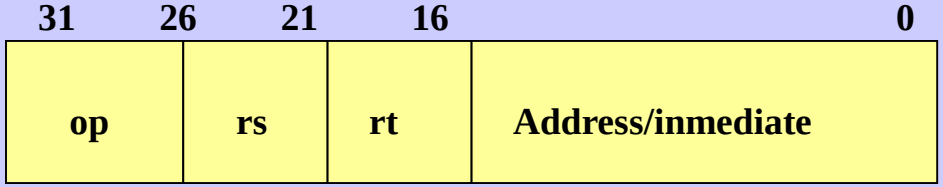
dirección: dirección destino del salto

1er Paso: Subconjunto MIPS

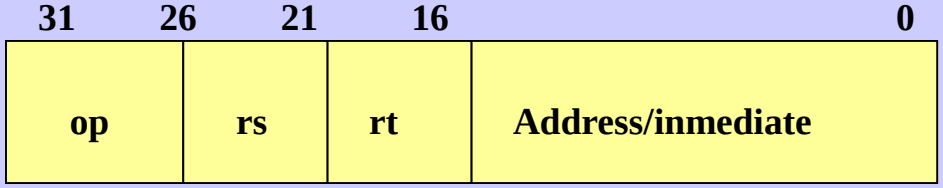
- ADD y SUB (tipo R)
  - addu rd ,rs ,rt
  - subu rd, rs ,rt



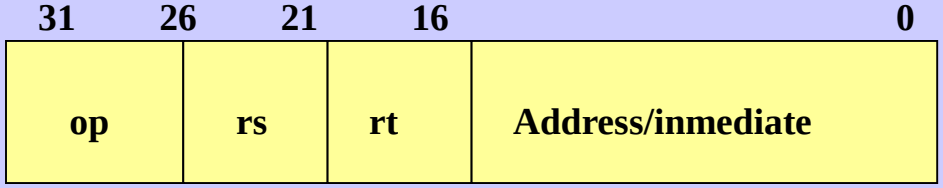
- OR inmediato
  - ori rt, rs, inm16



- LOAD and STORE
  - lw rt, rs, inm16
  - sw rt, rs, inm16



- BRANCH
  - beq rs, rt, inm16



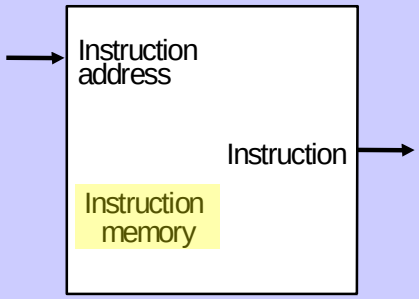
Todas instrucción comienza con un “fetch”

ADDU	**R[rd]←R[rs]+R[rt];**	**PC←PC+4**
SUBU	**R[rd]←R[rs]-R[rt];**	**PC←PC+4**
ORI	**R[rt]←R[rs] | zero_ext(Inm16);**	**PC←PC+4**
LOAD	**R[rt]←Mem[R[rs]+sign_ext(Inm16)];**	**PC←PC+4**
STORE	**Mem[R[rs]+sign_ext(Inm16)]← R[rt];**	**PC←PC+4**
BEQ	**if (R[rs]==R[rt]) then PC←PC+4+(sign_ext(Inm16)*4)** **else PC←PC+4**	

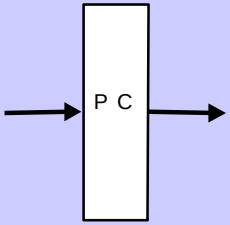
1er Paso: Análisis de requerimientos del Conjunto de Instrucciones

- Memoria
  - Separadas para Instrucciones y Datos (Arquitectura Harvard)
- Registros (32x32)
  - Leer rs
  - Leer rt
  - Escribir rt o rd
- PC (Program Counter)
- Extensor de signo
- Sumar y Restar registros y/o valores inmediatos
- Operaciones lógicas (and/or) entre registros y/o valores inmediatos
- Sumar al PC 4 ó 4+inmediato extendido *4

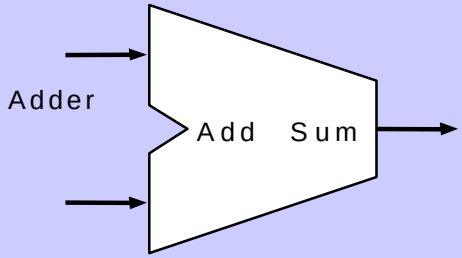
2º Paso: Componentes del Camino de Datos

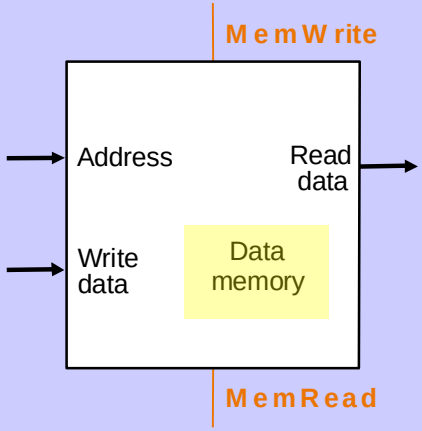


Instruction memory

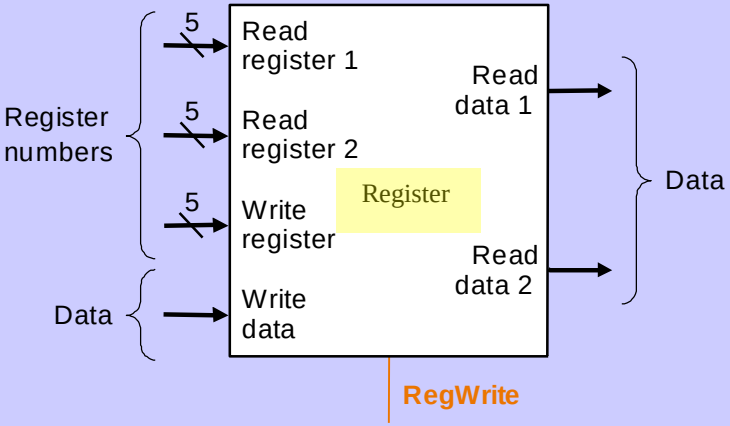


Program counter

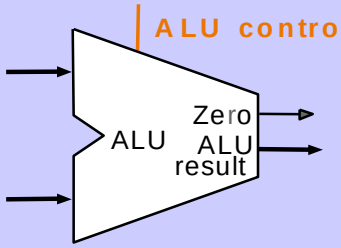




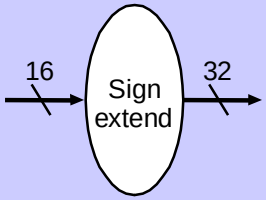
Data memory unit



Registers



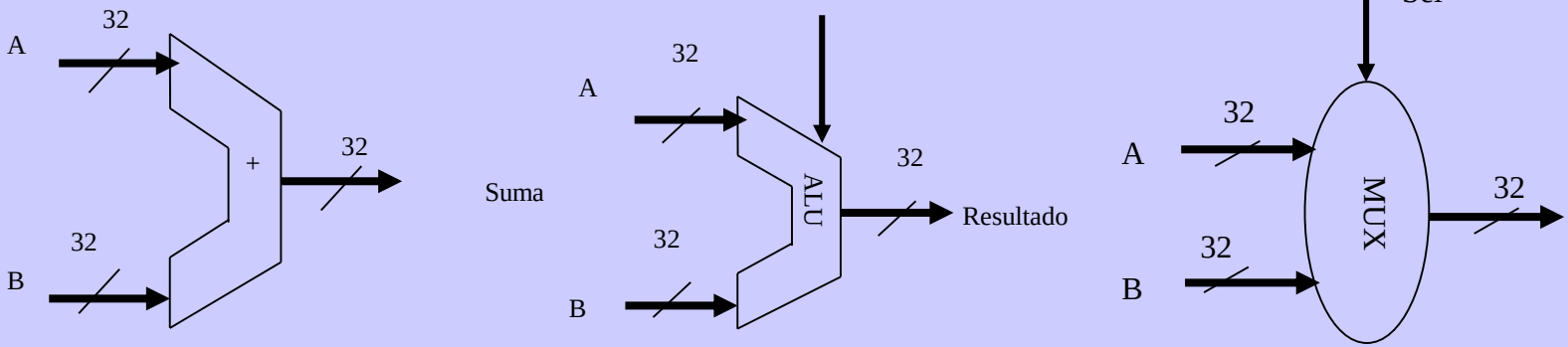
ALU



Sign-extension unit

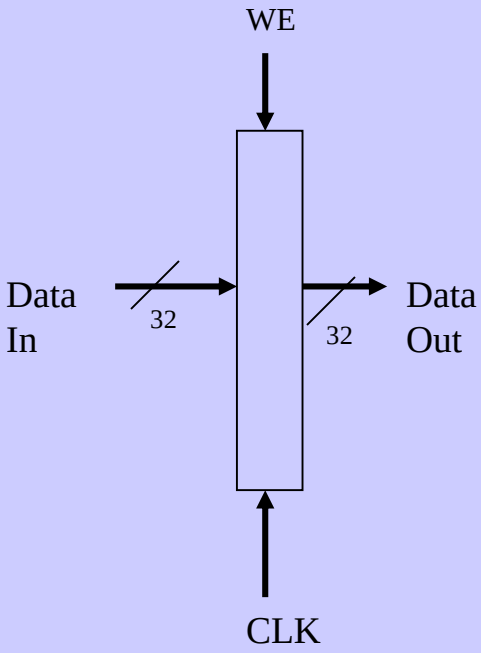
2º Paso: Componentes del Camino de Datos

- Elementos Combinacionales
  - Sumador , ALU y Multiplexor

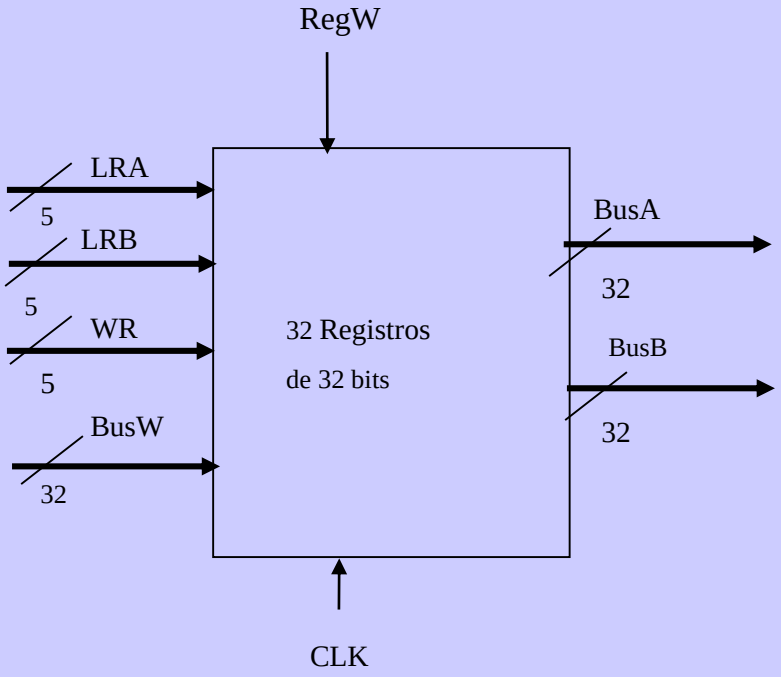


2º Paso: Componentes del Camino de Datos

- Elementos de Almacenamiento: Banco de Registros

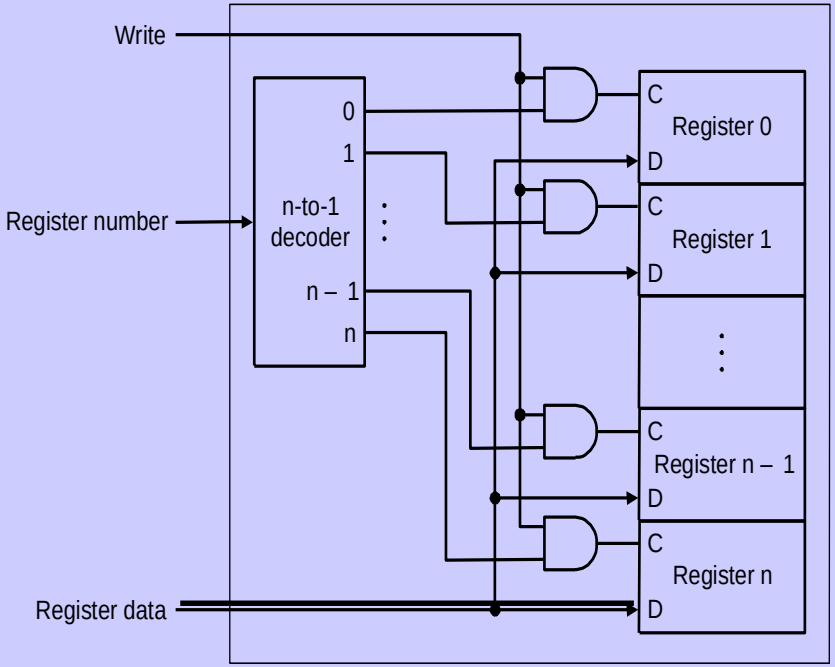


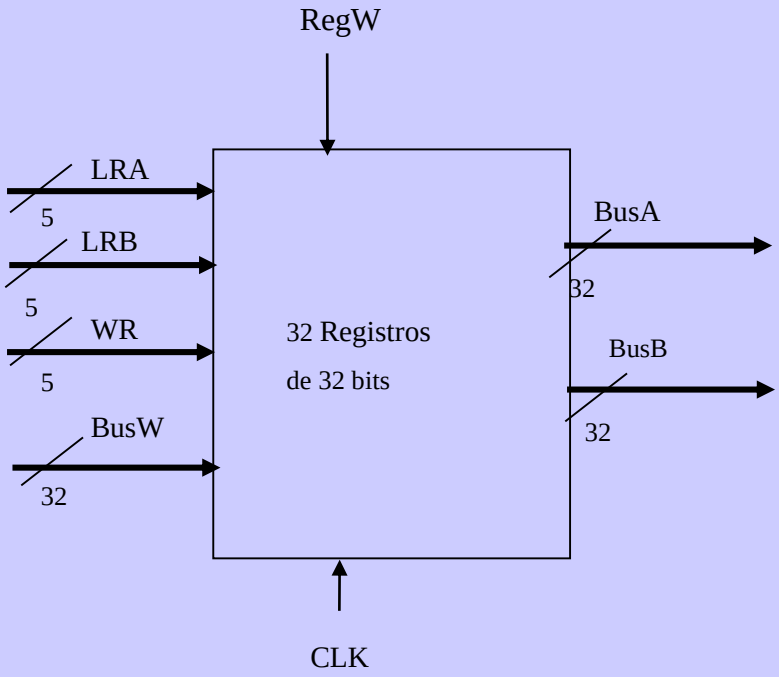
registro de 32 bits



2º Paso: Componentes del Camino de Datos

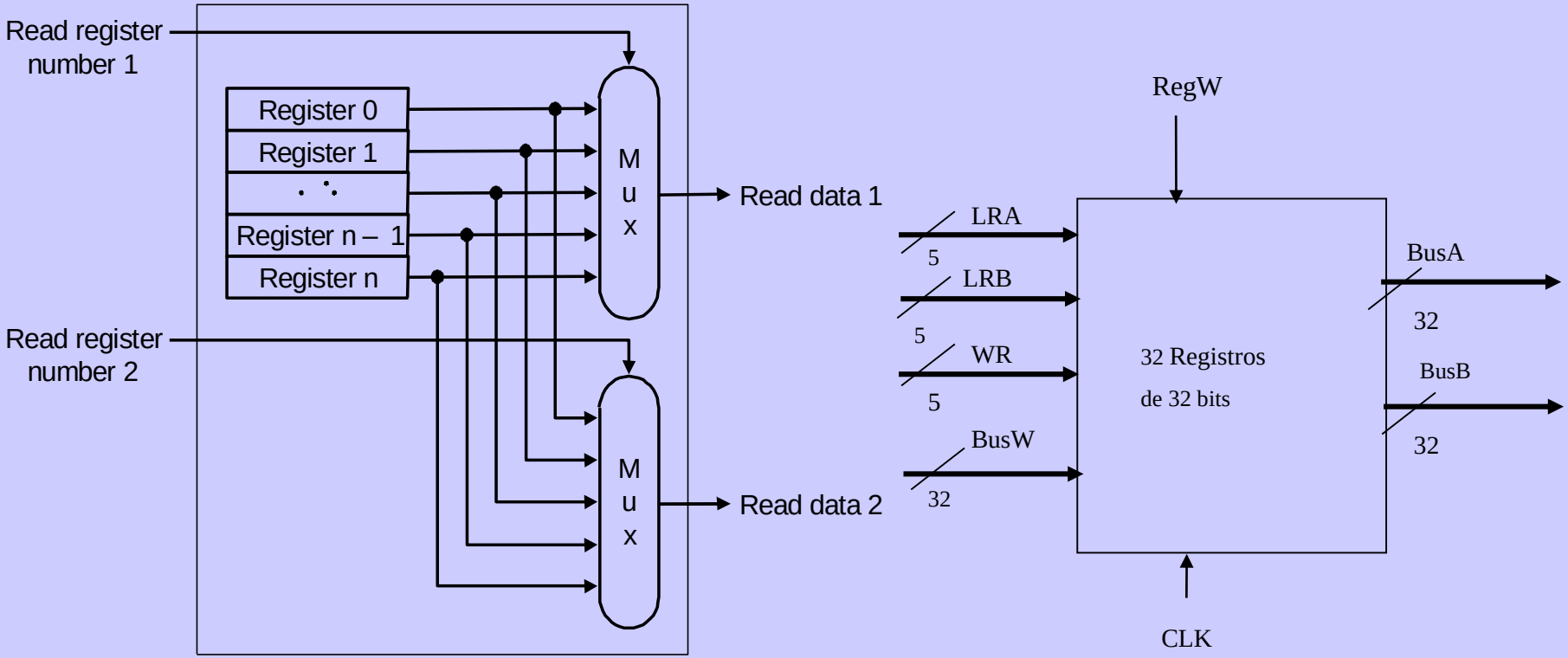
- Elementos de Almacenamiento: Banco de Registros: un puerto de escritura





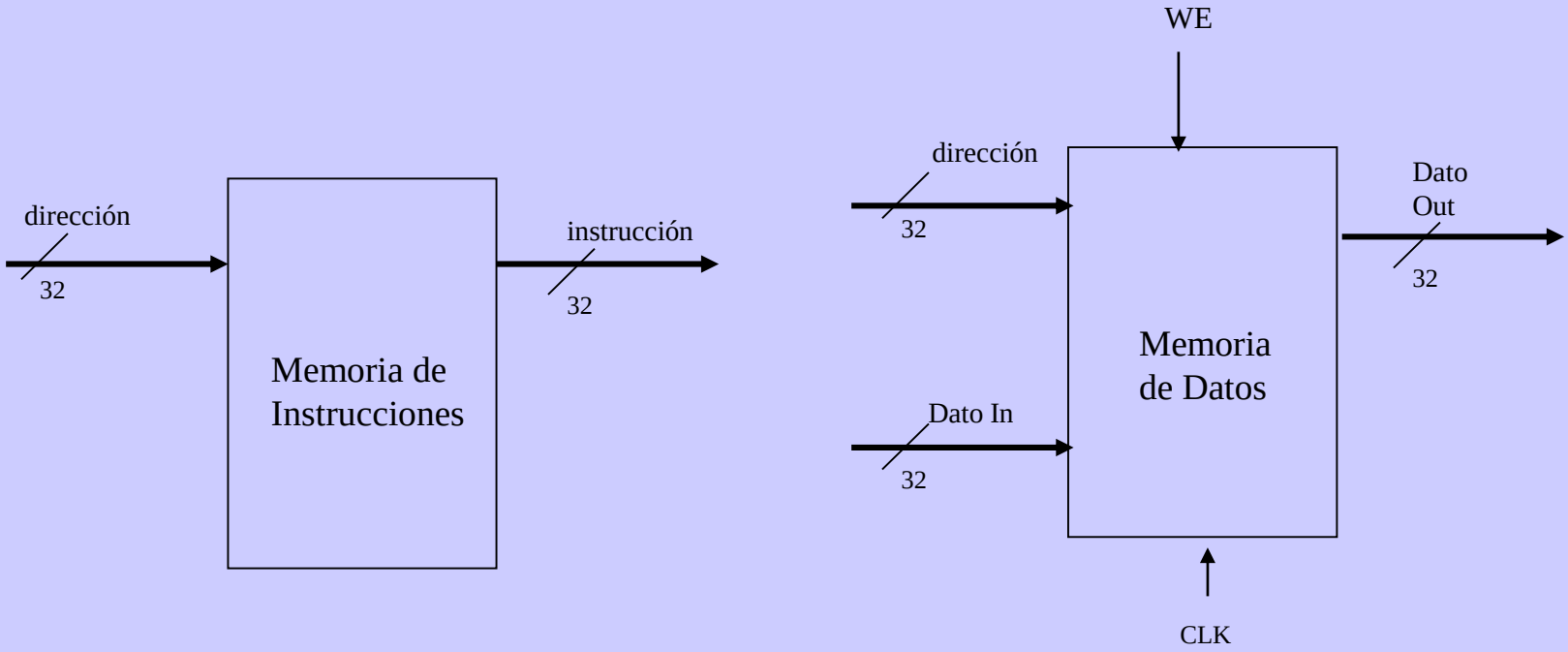
2º Paso: Componentes del Camino de Datos

- Elementos de Almacenamiento: Banco de Registros: dos puertos de lectura



2º Paso: Componentes del Camino de Datos

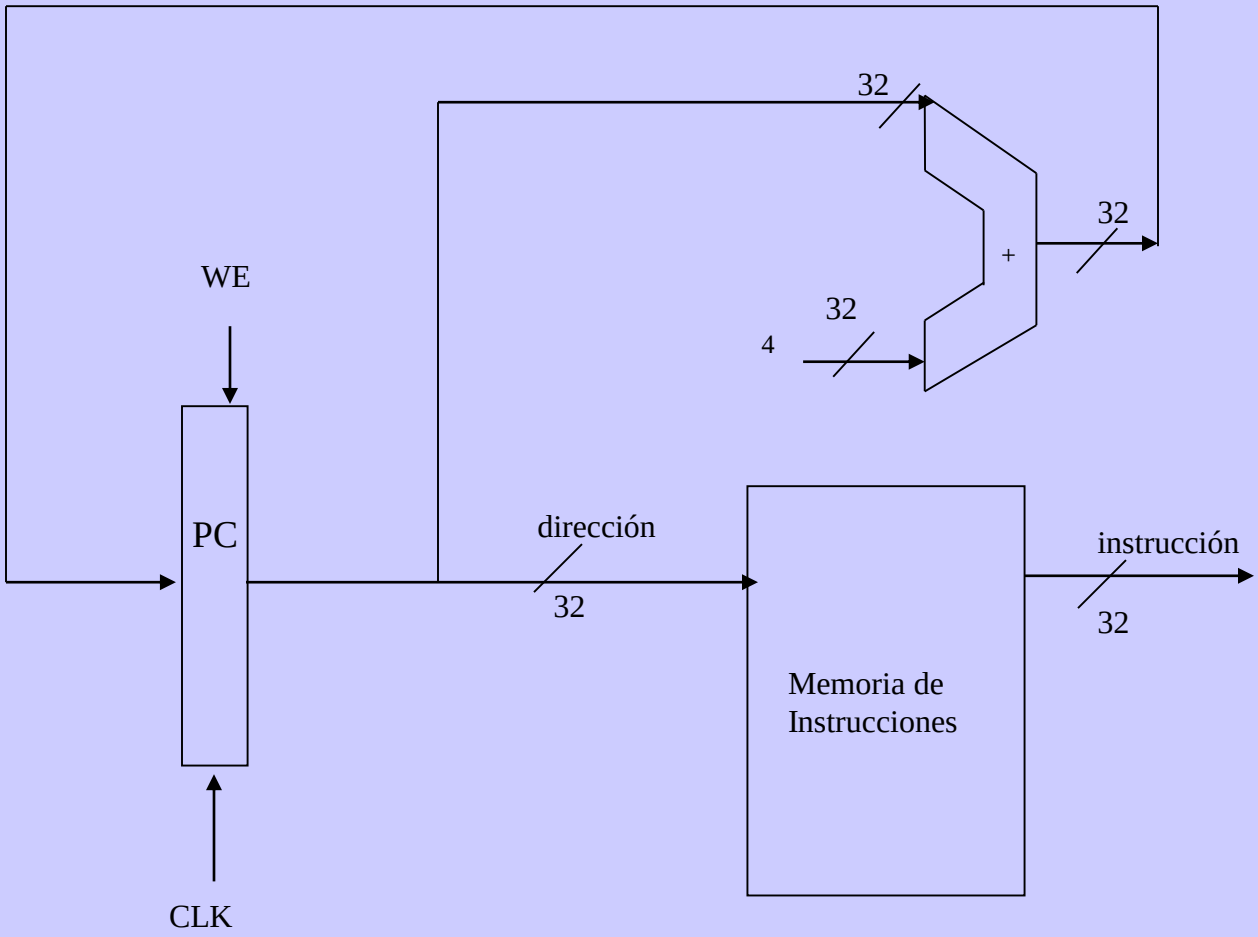
- Elementos de Almacenamiento:
  - Memorias



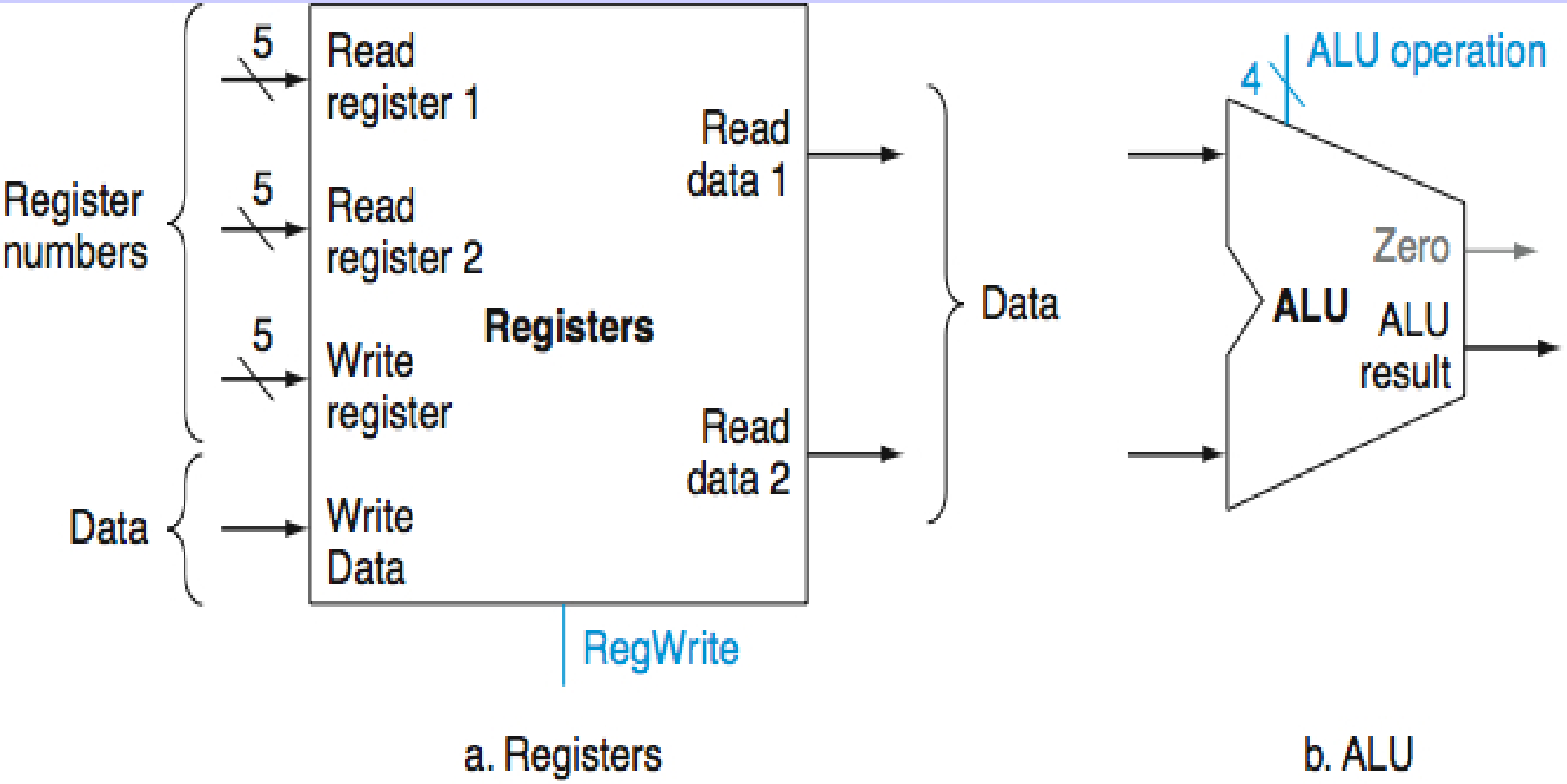
3er Paso: Construcción del Camino de Datos

Búsqueda (Fetch) de Instruccion y actualización del PC

- Mem[PC]; PC← PC+4 (código secuencial)



Utilización de la ALU



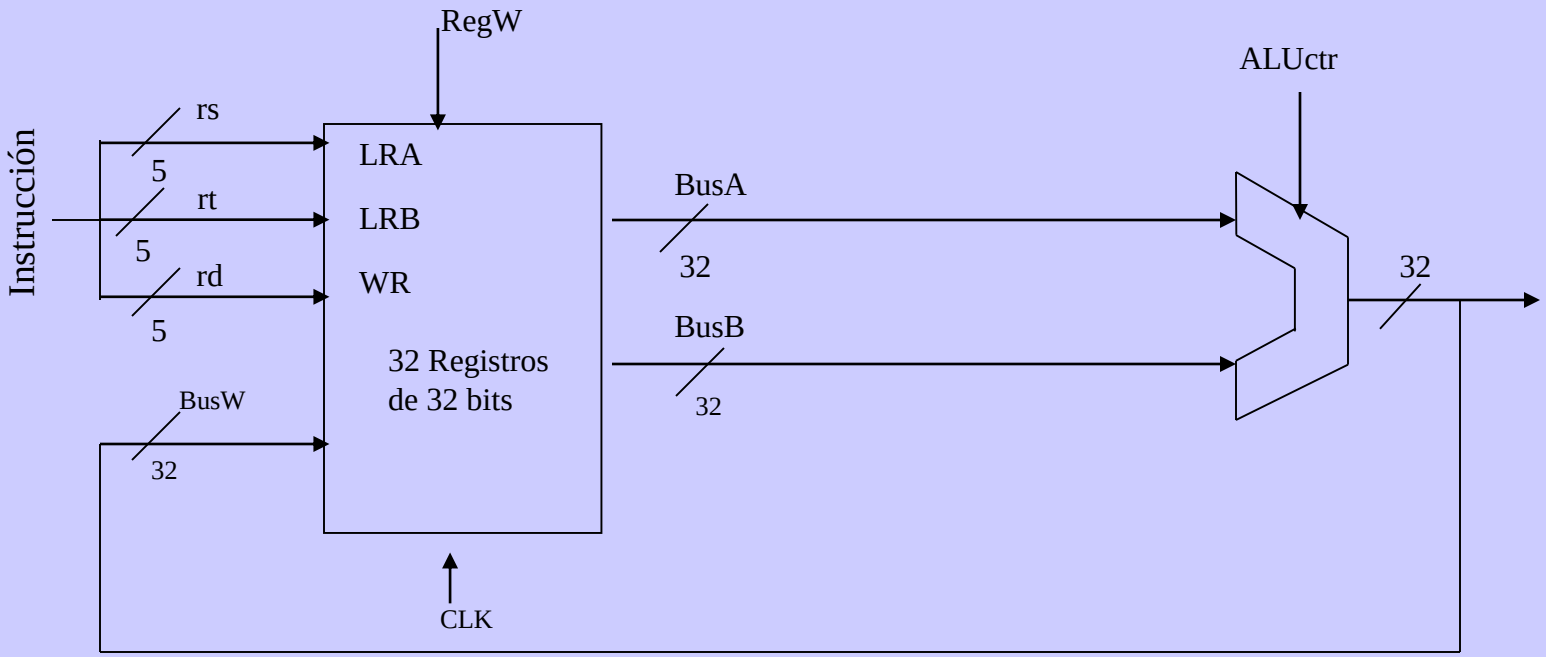
Los registros son entrada y salida de la ALU

3er Paso: Construcción del Camino de Datos

Formato R: Addu y Subu

Suma sin signo

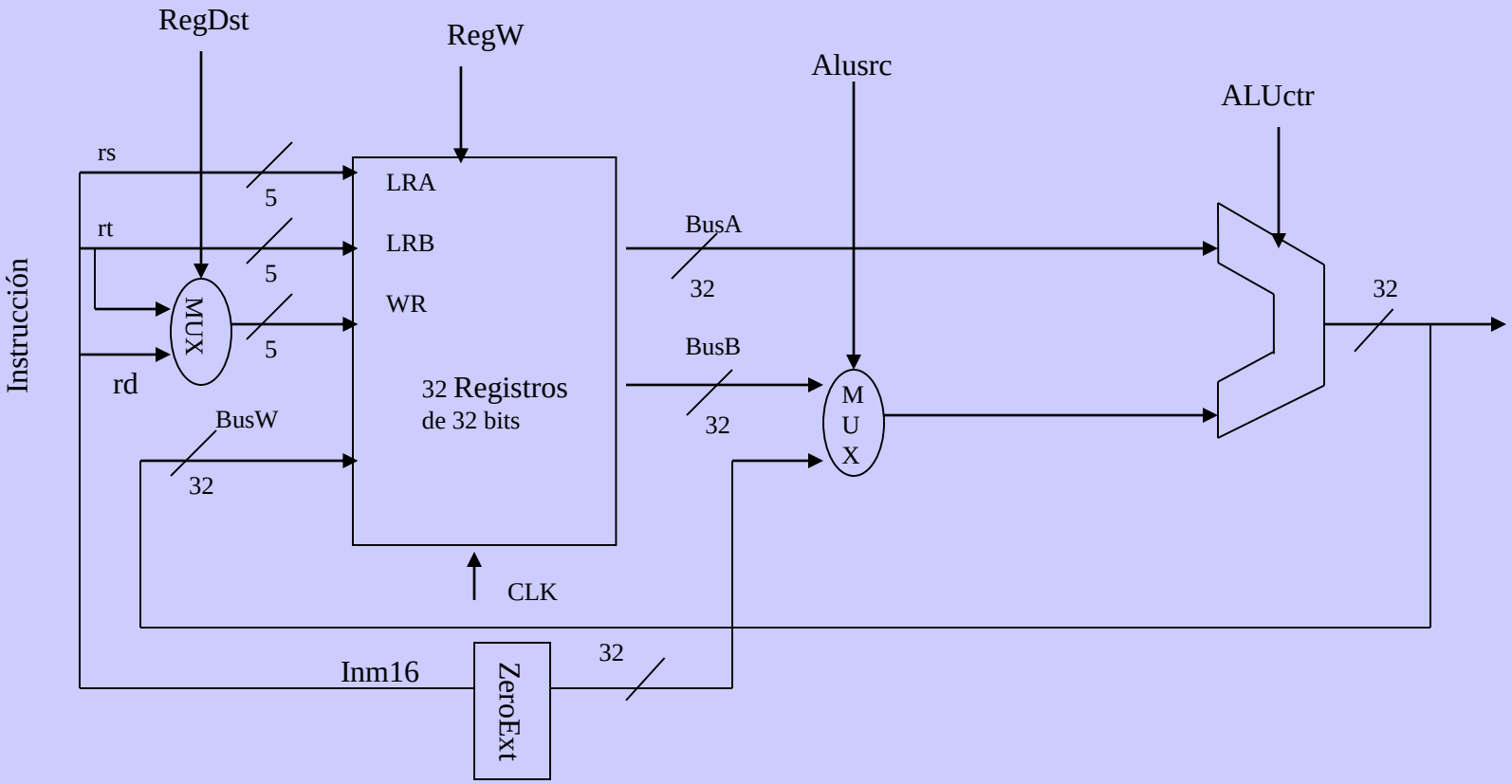
ADDU	R[rd]←R[rs]+R[rt];	PC←PC+4
SUBU	R[rd]←R[rs]-R[rt];	PC←PC+4



3er Paso: Construcción del Camino de Datos

Formato I: lógicas con Inmediato

ORi R[rt]←R[rs] op ZeroExtR[inm16]; PC←PC+4

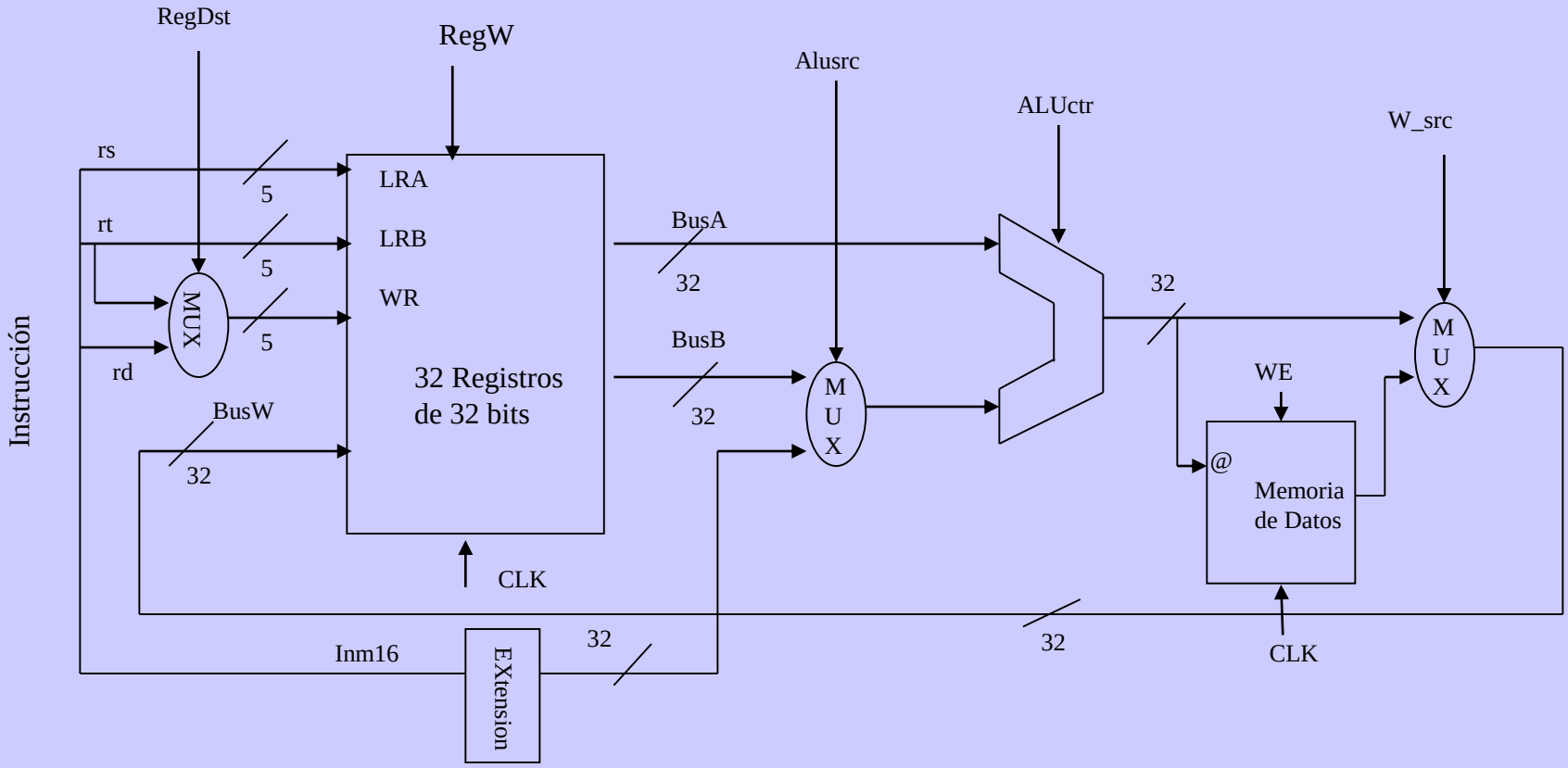


3er Paso: Construcción del Camino de Datos

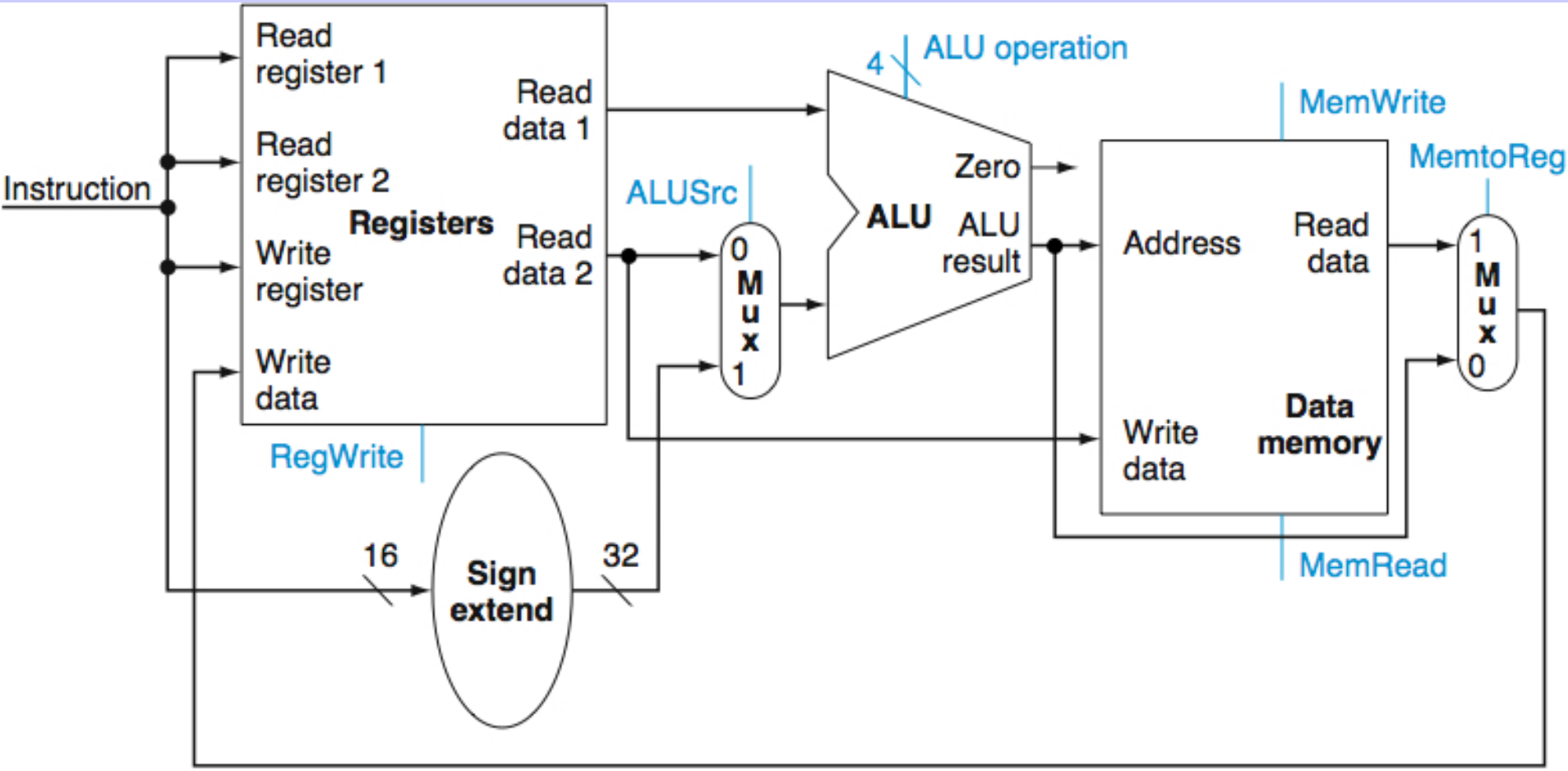
Formato I: LOAD

Lw rt,rs,inmm16

R[rt]←Mem[R[rs]+ SignExt[inm16]]; PC←PC+4



Acceso a memoria



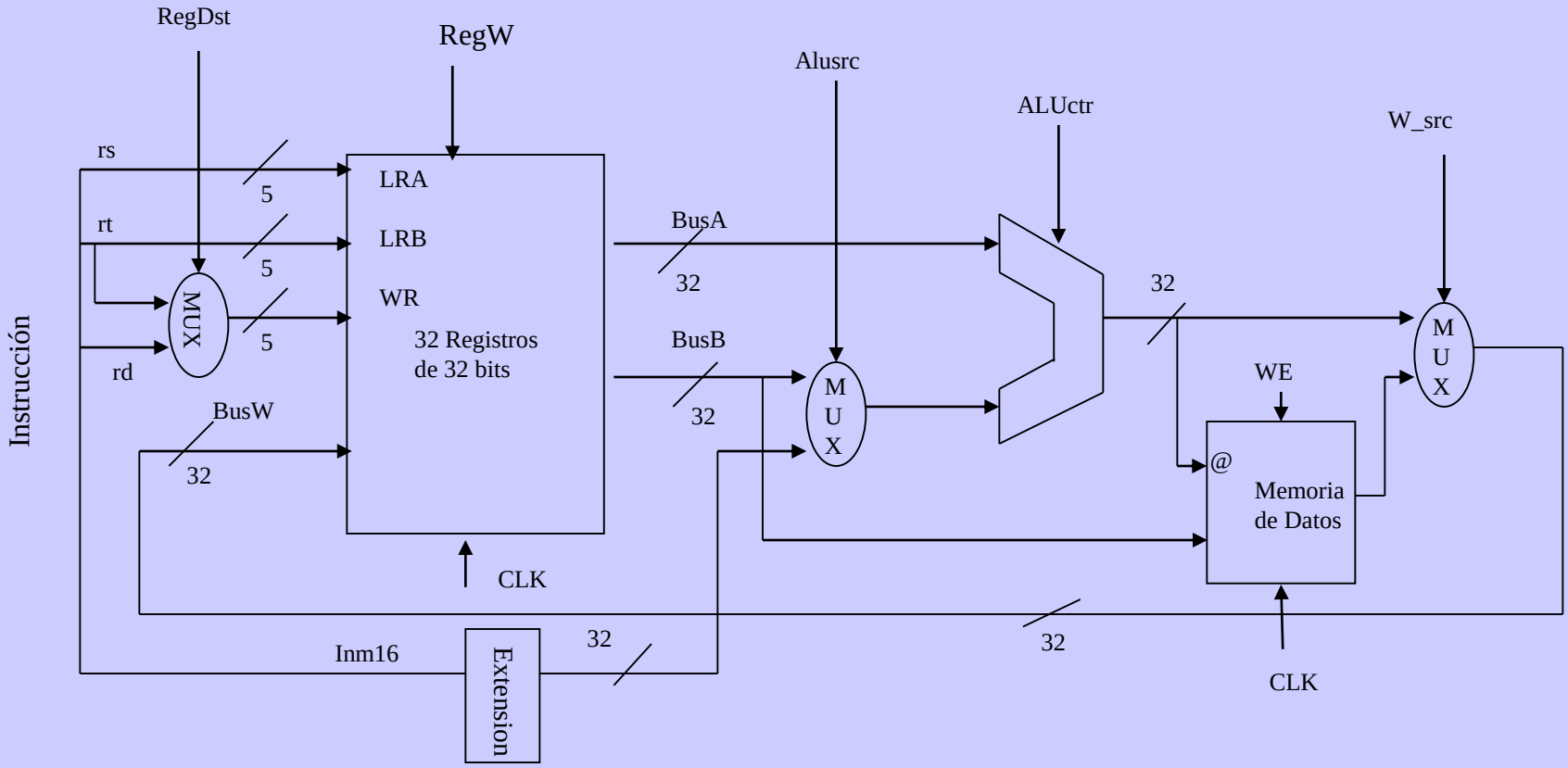
Extensión de signo y cálculo de dirección del operando

3er Paso: Construcción del Camino de Datos

Formato I: STORE

Sw rt,rs,inmm 16

Mem[R[rs]+ SignExt[inm16]]←R[rt]; PC←PC+4



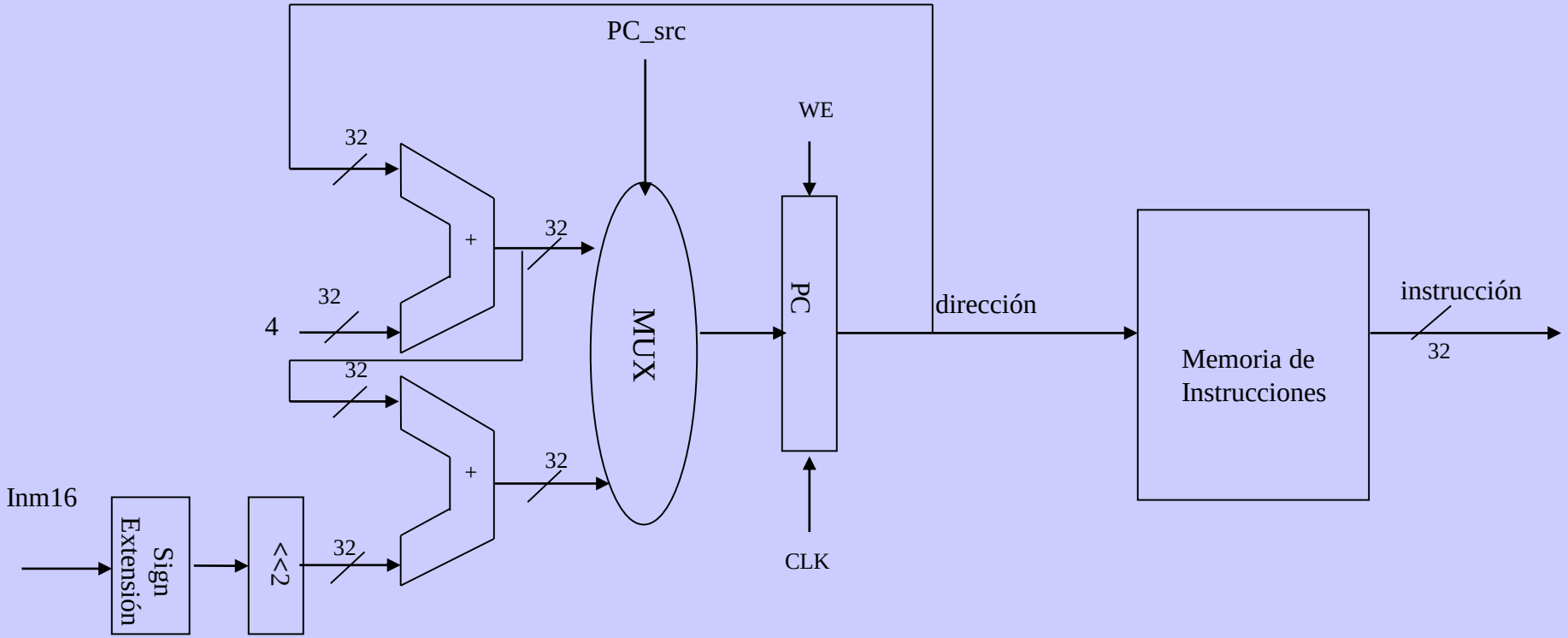
3er Paso: Construcción del Camino de Datos

Formato I: Branch

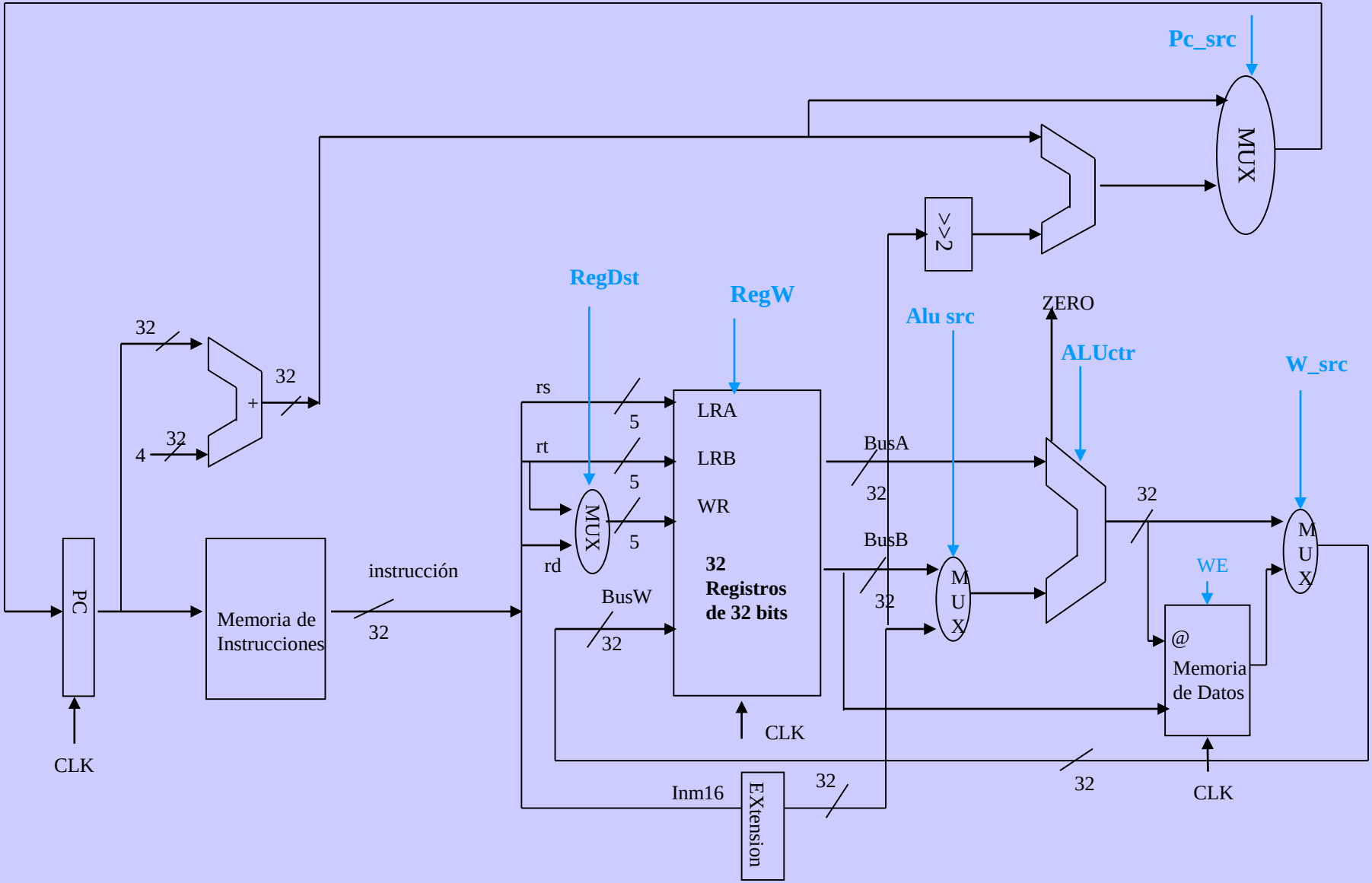
Beq rs, rt, inm16

if (R[rs]==R[rt]) then PC←PC+4+(sign_ext(Inm16)*4

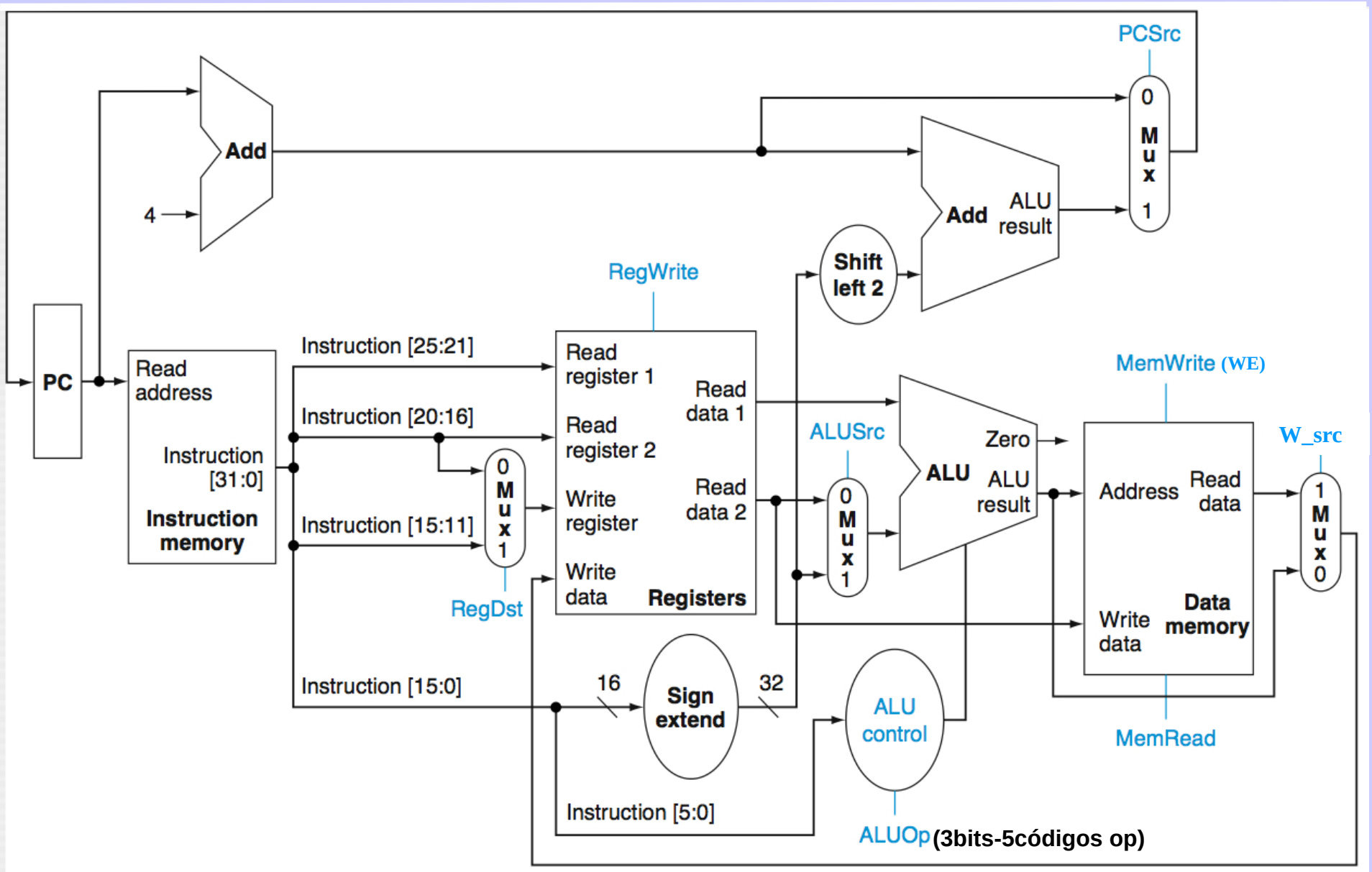
else PC←PC+4



3er Paso: Juntando Todo



Datos, multiplexores y líneas de control

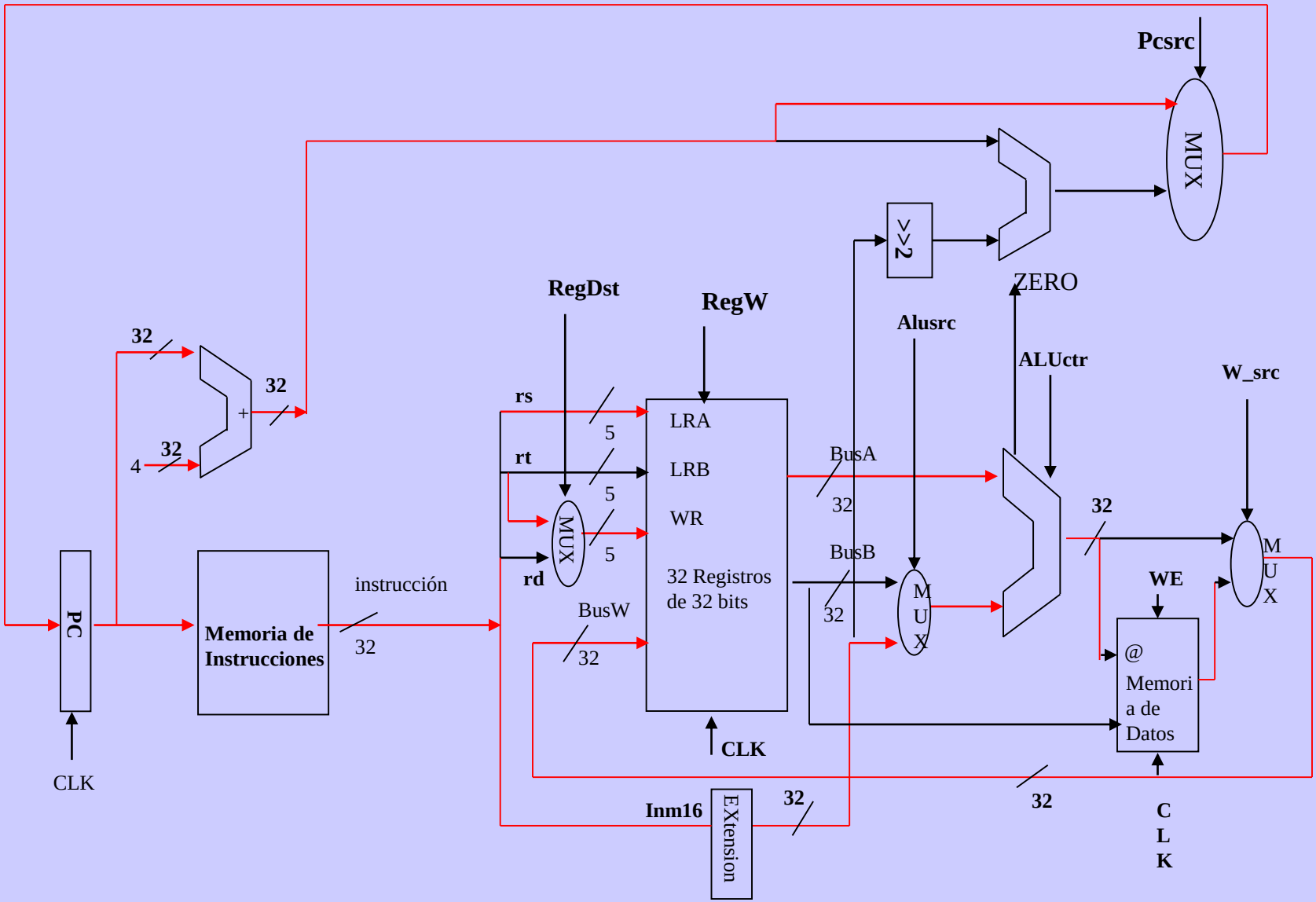


Señales de Control

	No Activa	Activa
RegDst	destino dado por bits de rt	destino dado por bits de rd
ALUsrc	operando proviene del Bco. De Reg	operando son los 16 bits de la inst. con el signo extendido
W_src	El valor que se escribe en el Bco. de Reg proviene de la ALU	El valor que se escribe en el B. De Reg proviene de Memoria
RegW	Nada	Se escribe el valor en el registro especificado
WE	Nada	Se escribe en la memoria el valor que está en su entrada
PC_src	PC=PC+4	PC=destino de salto

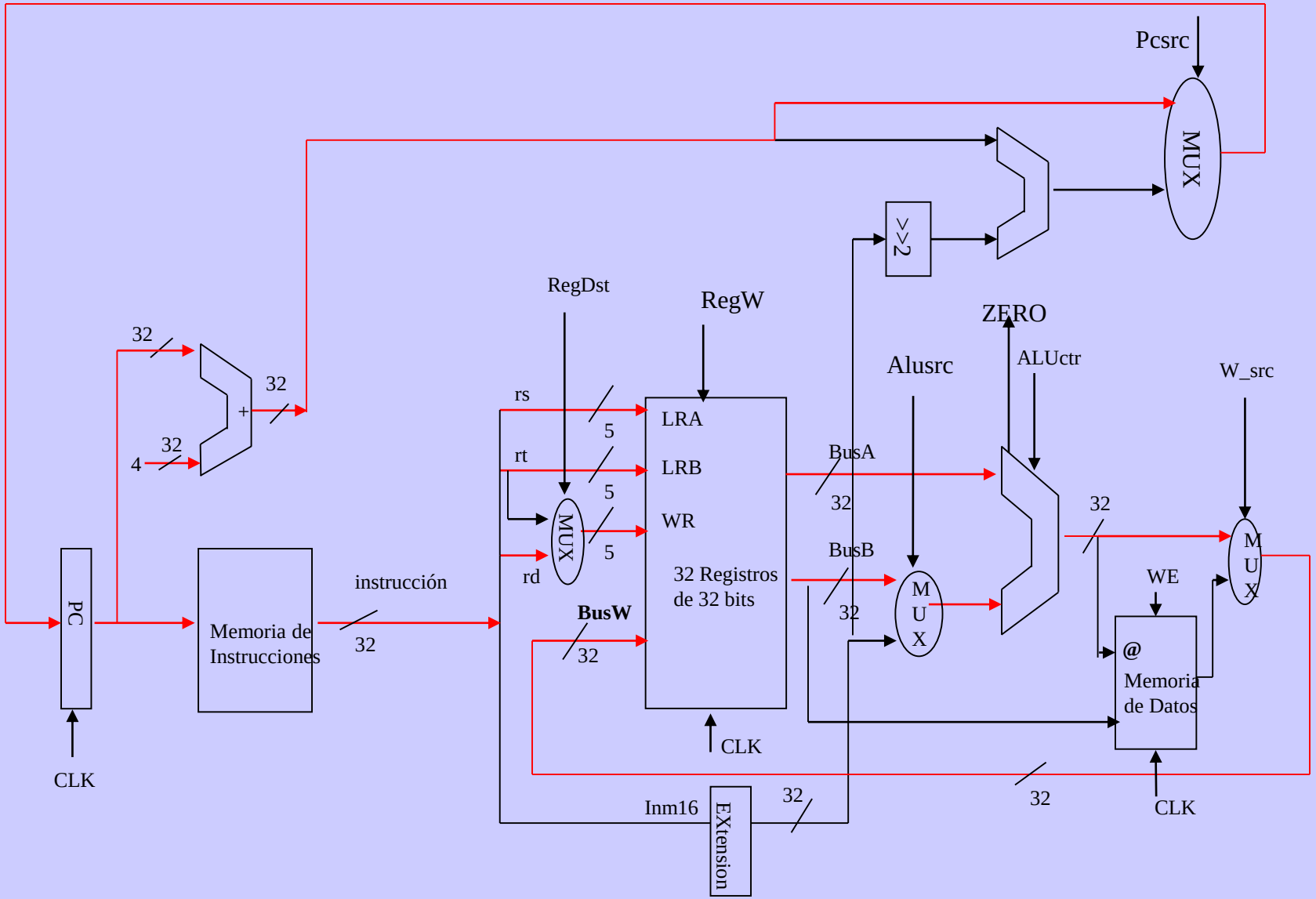
Ejemplo: Load

R[rt]←Mem[R[rs]+ SignExt[inm16]]; PC←PC+4



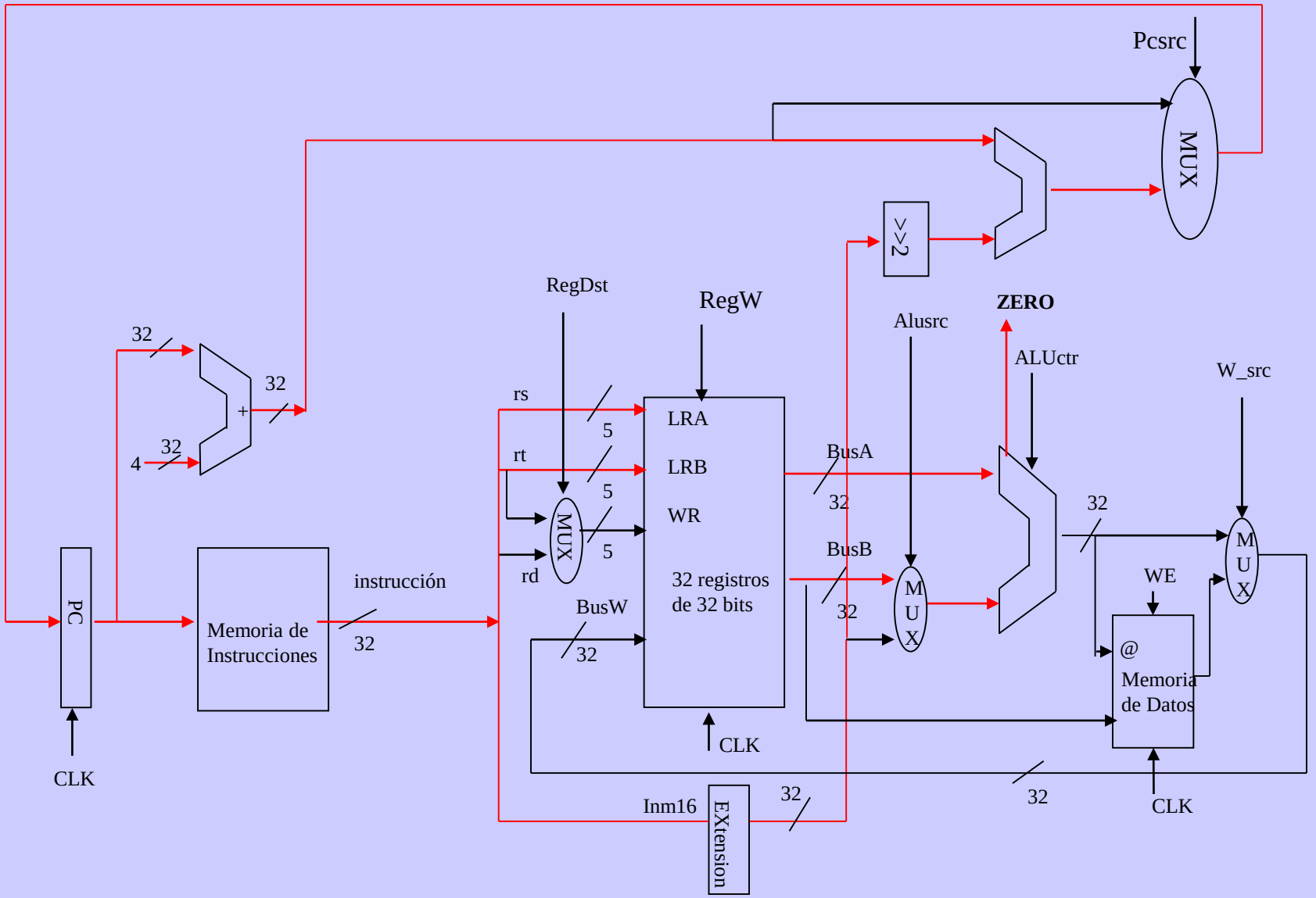
Ejemplo: ADDU

R[rd]←R[rs]+R[rt]; PC←PC+4

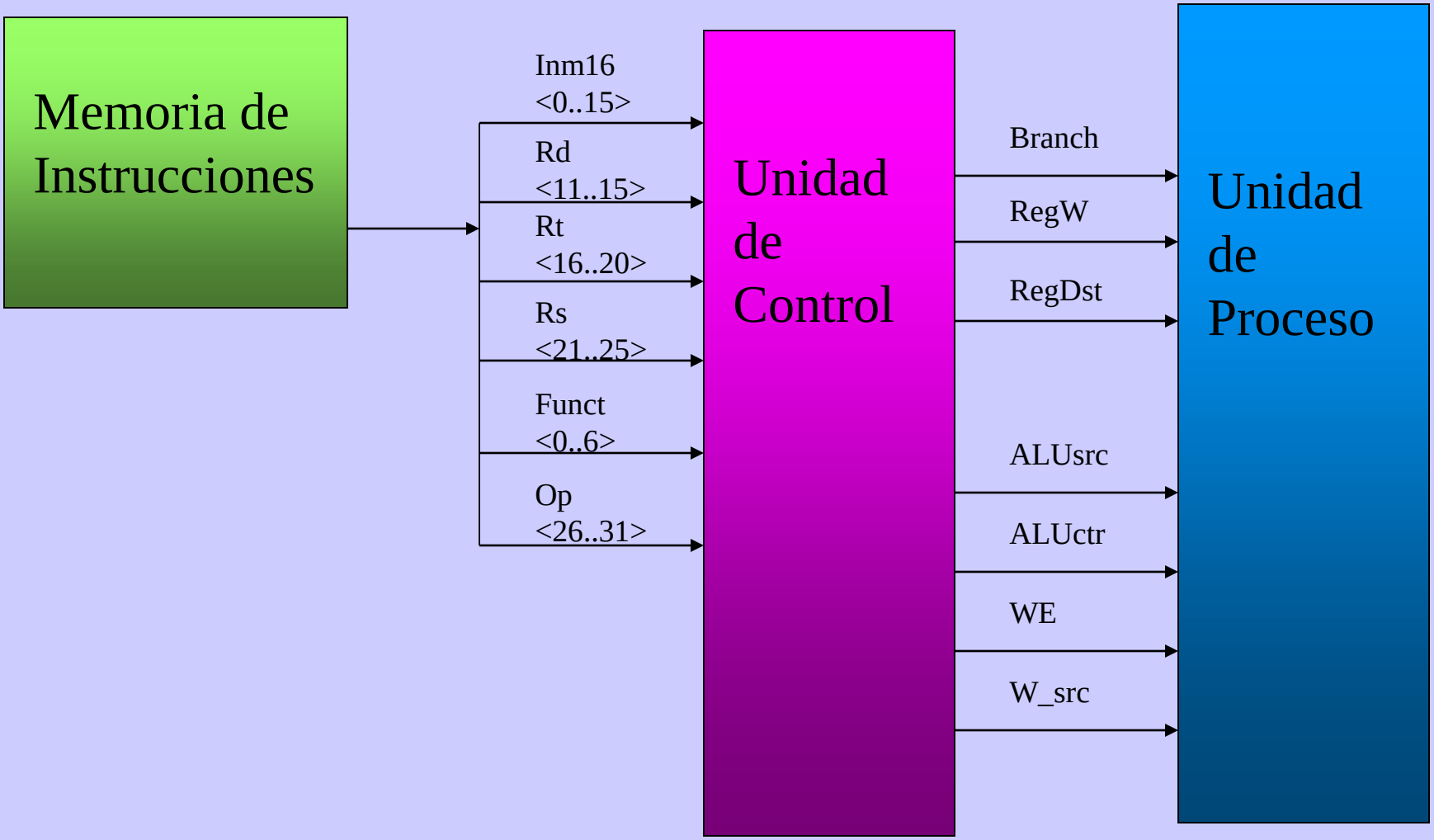


Ejemplo: BEQ (realizado)

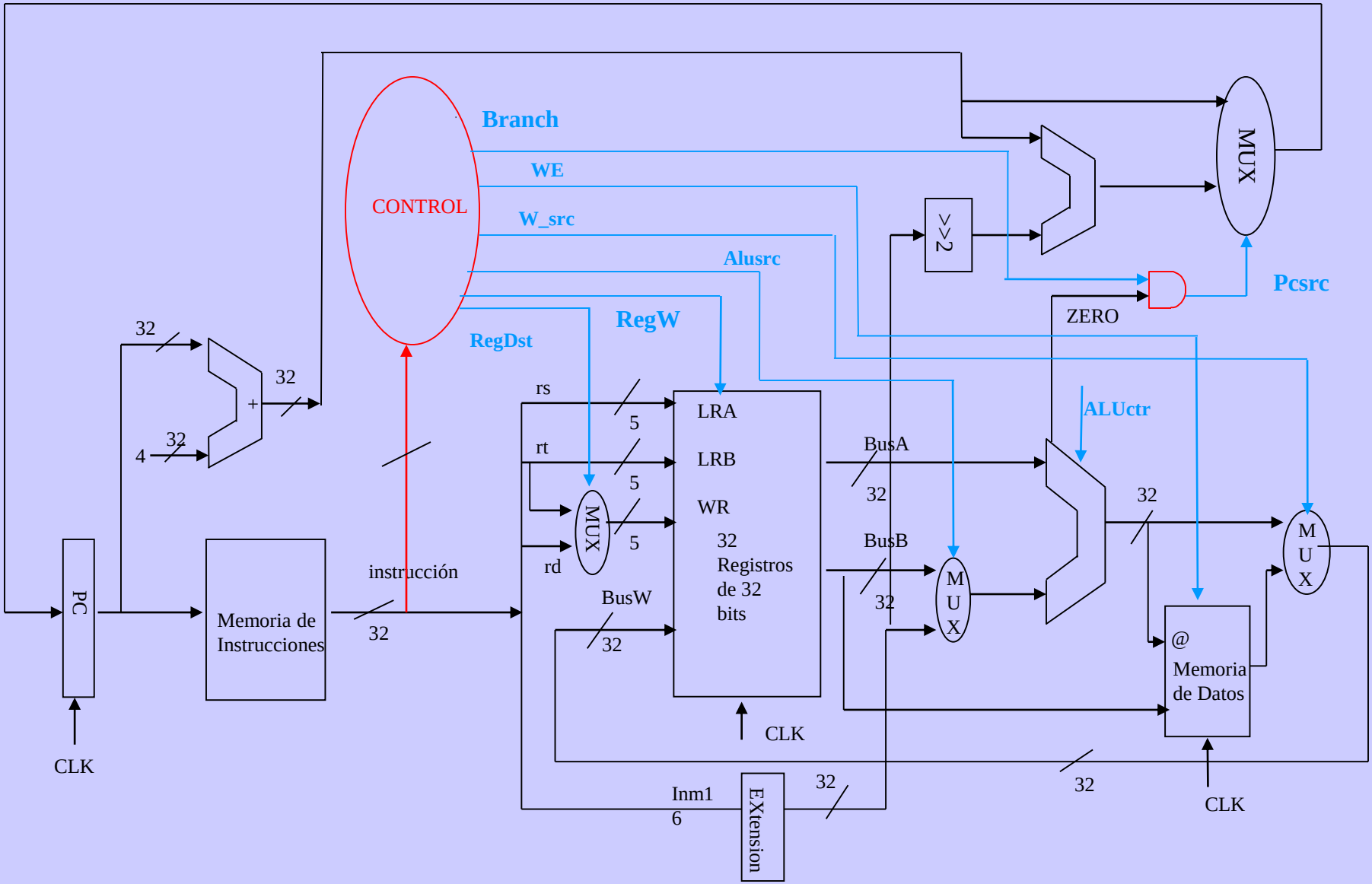
if (R[rs]==R[rt]) then PC←PC+4+(sign_ext(Inm16)*4)



4^0 Paso: Control



4º Paso: Control

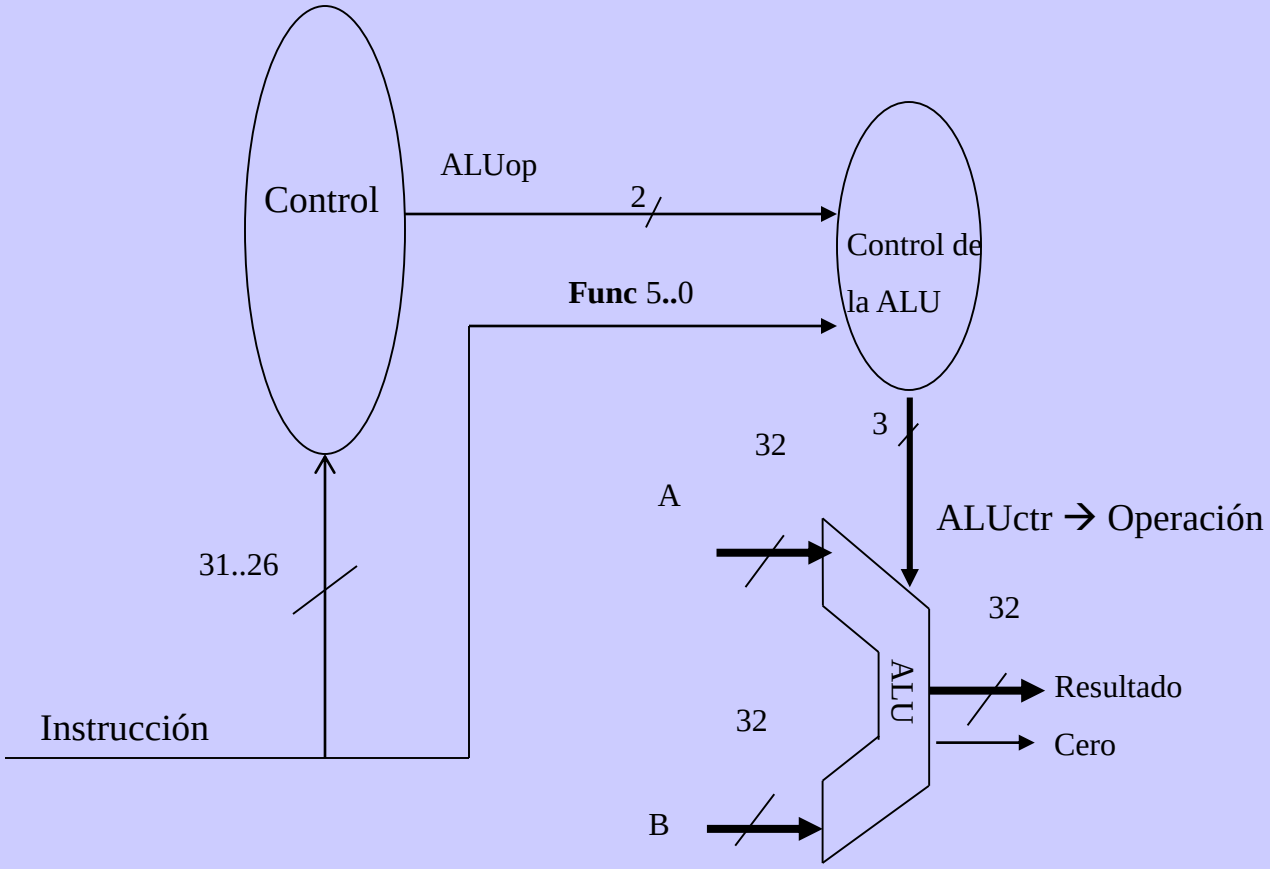


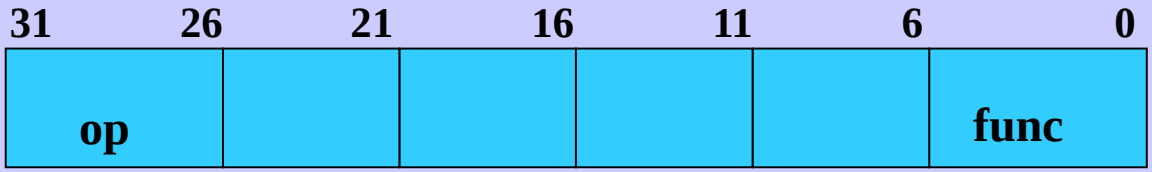
	No Activa	Activa
RegDst	destino dado por los bits de rt	destino dado por bits de rd
ALUsrc	operando proviene del B.de REG	operando son los 16 bits de la inst. con el signo ext
W_src	El valor proviene de la ALU	El valor proviene de MEM
RegW	Nada	Se escribe el valor en el registro especificado
WE	Nada	Se escribe en Mem. el valor que está en su entrada
PC_src	PC=PC+4	PC=destino de salto

4º Paso: Control Principal

	Addu	Subu	Lw	Sw	Ori	Beq
RegDst	1	1	0	x	0	x
ALUsrc	0	0	1	1	1	0
W_src	0	0	1	x	0	x
RegW	1	1	1	0	1	0
WE	0	0	0	1	0	0
Branch	0	0	0	0	0	1
ALUctr	suma	resta	suma	suma	or	resta

4º Paso: Control de la ALU





4º Paso: Control de la ALU

- La ALU necesita 3 bits de control que serán generados con:

  - el tipo de instrucción

    00 = lw, sw
    01 = beq,
    10 = arithmetic

    } ALU Op

  Observar que los bits de "Funct field" sólo se usan cdo ALUop=10

  - y el código de función ("funct field") de las tipo R

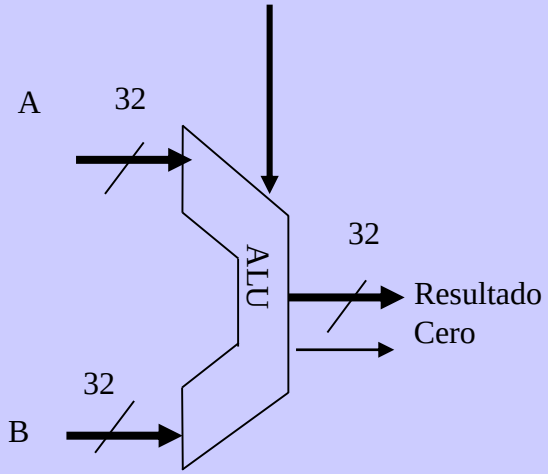
ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	0	1	0	0	0	0	0	010
1	0	1	0	0	0	1	0	110
1	0	1	0	0	1	0	0	000
1	0	1	0	0	1	0	1	001
1	0	1	0	1	0	1	0	111

Add (lw, sw)
Sub (beq)
Add (R-type)
sub (R-type)
and (R-type)
or (R-type)
set-on-less-than

4º Paso: Control de la ALU

operation ----------> Operation

```
000    AND
001    OR
010    add
110    subtract
111    set-on-less-than (slt)
```



Función Set Less Than (slt)

- **slt** se define como:

A **slt** B =
- 000 … 001 if A < B, i.e. if A – B < 0
- 000 … 000 if A >= B, i.e. if A – B >= 0

4º Paso: Control de la ALU

- La ALU necesita 3 bits de control que serán generados con:

  - el tipo de instrucción

    00 = lw, sw
    01 = beq,
    10 = arithmetic

    } ALU OP

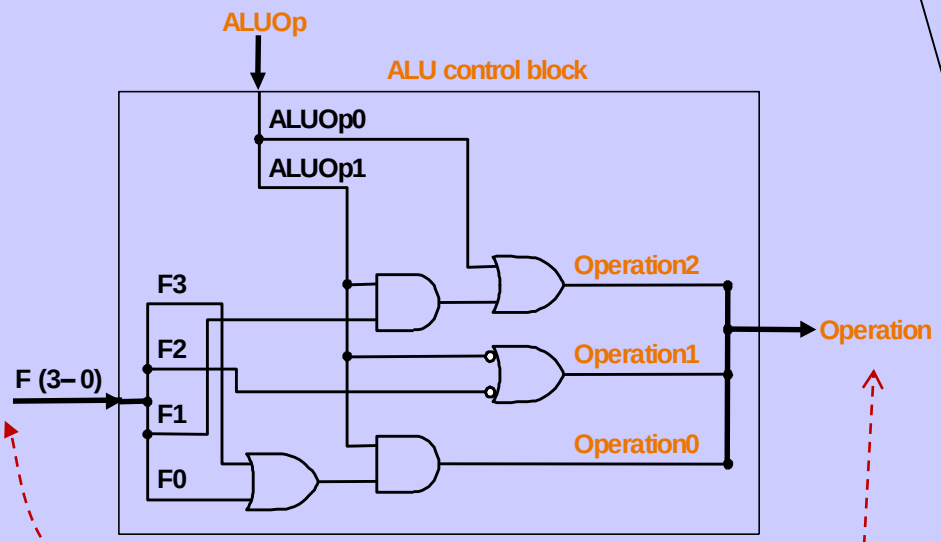
  - y el código de función de las tipo R

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111

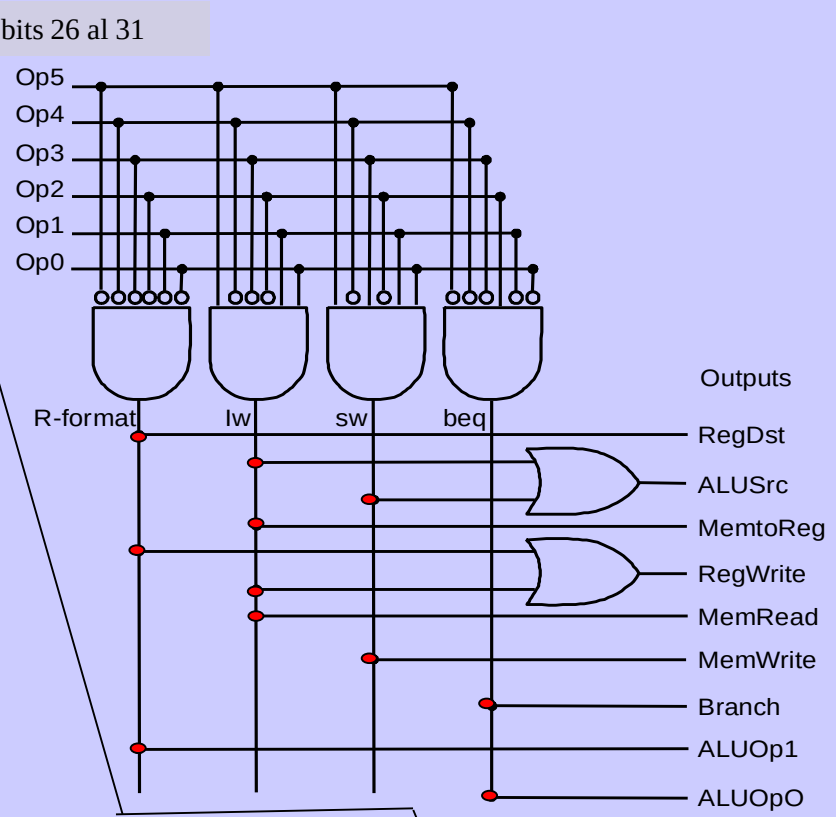
Add (lw, sw)
Sub (beq)
Add (R-type)
sub (R-type)
and (R-type)
or (R-type)
set-on-less-than

Ver cambios X

5º Paso: Implementación

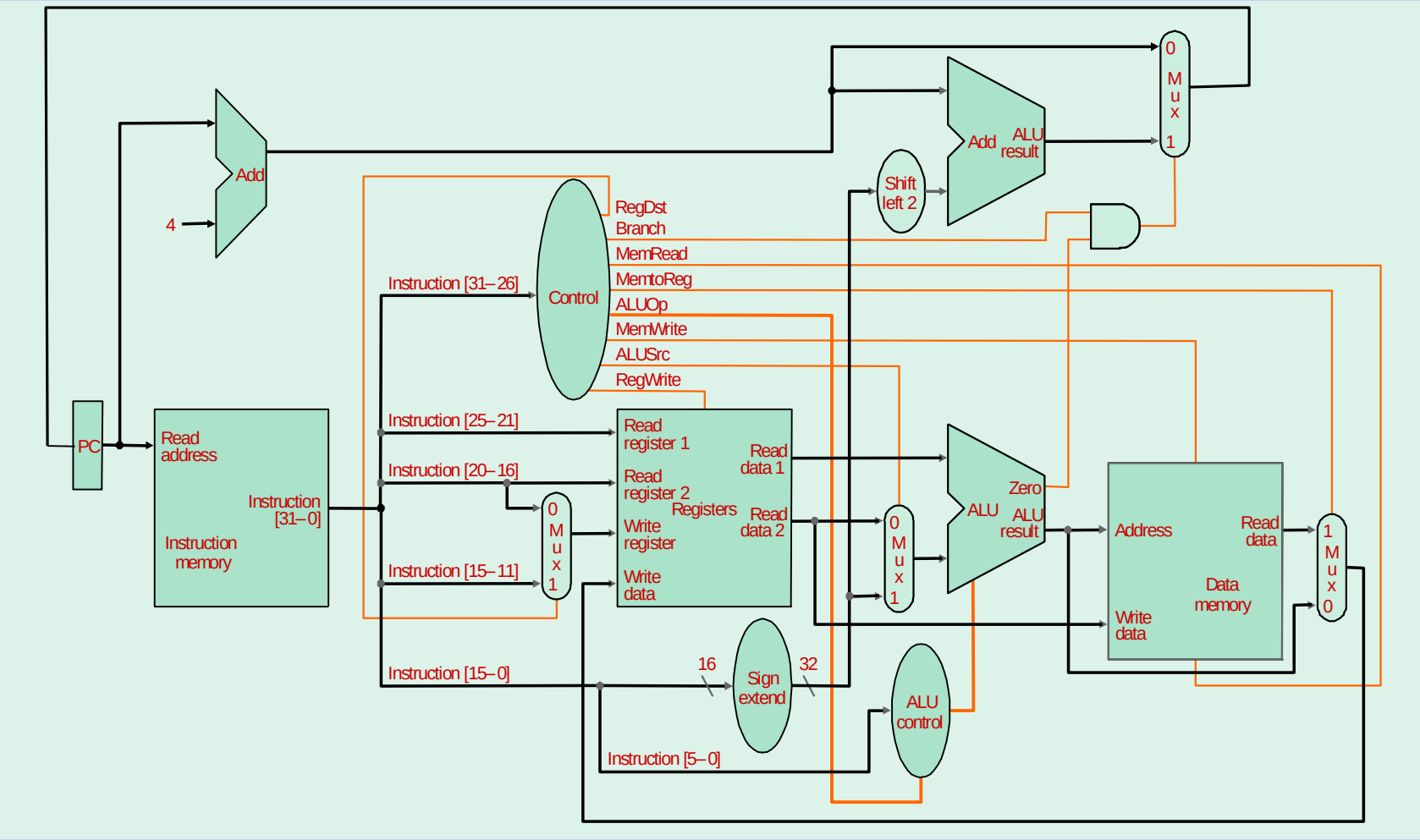


ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	010
0	1	X	X	X	X	X	X	110
1	X	X	X	0	0	0	0	010
1	X	X	X	0	0	1	0	110
1	X	X	X	0	1	0	0	000
1	X	X	X	0	1	0	1	001
1	X	X	X	1	0	1	0	111



Add (lw, sw)
Sub (beq)
Add (R-type)
sub (R-type)
and (R-type)
or (R-type)
set-on-less-than

Control



Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUp0
R-format	1	0	0	1	0	0	0	1	0
`lw`	0	1	1	1	1	0	0	0	0
`sw`	X	1	X	0	0	1	0	0	0
`beq`	X	0	X	0	0	0	1	0	1

Simple Estructura de Control

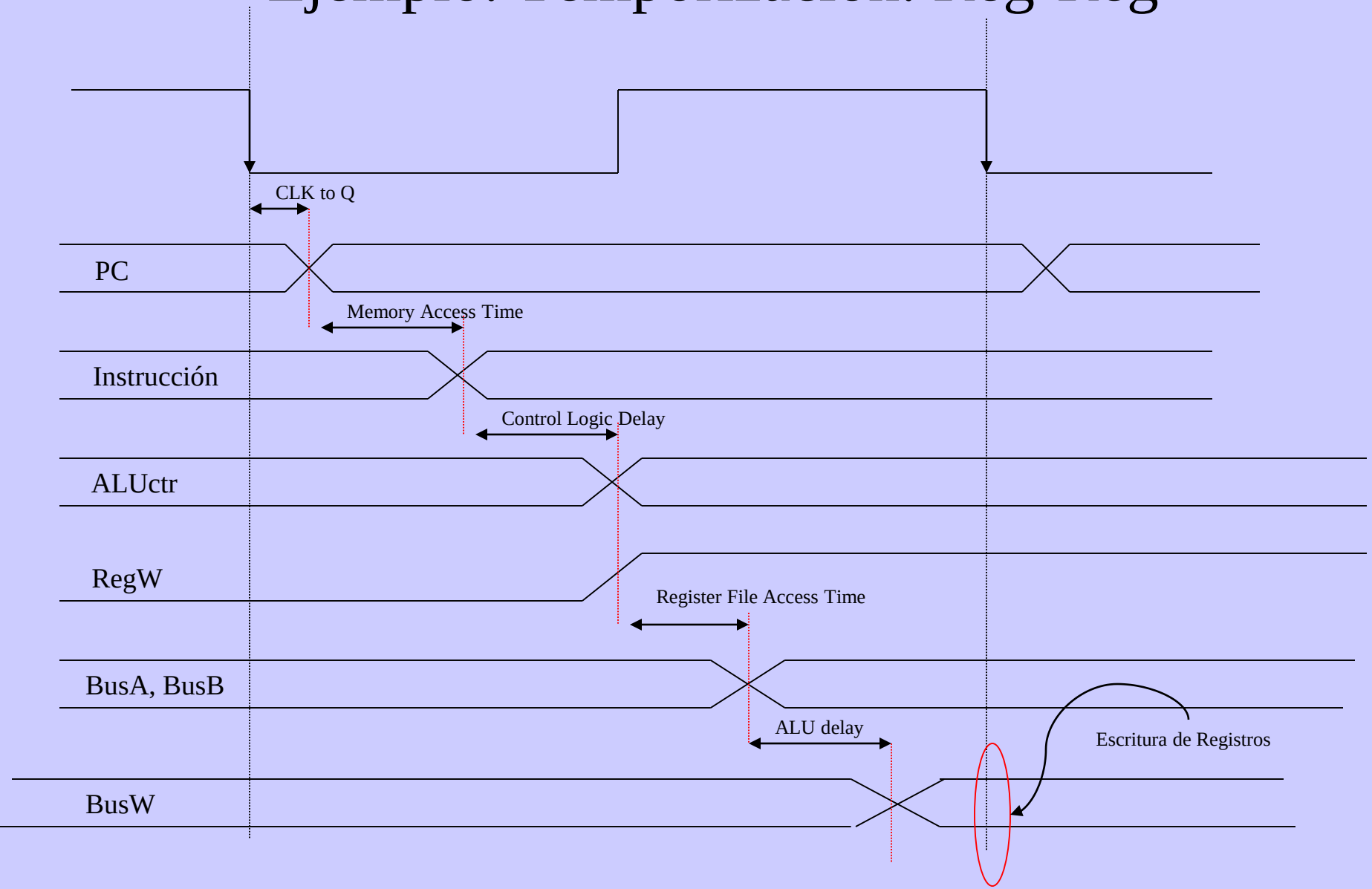
- Toda la lógica es combinacional
- Hay que esperar que las señales se estabilicen
- La duración del ciclo está determinada por la longitud del camino crítico

Camino Crítico: Load

Tiempos a tener en cuenta para calcular el periodo del reloj

- PC: propagación (hold)+
- Acceso a Memoria de Instrucciones +
- Acceso al Banco de Registros +
- ALU, suma de 32 bits +
- Acceso a Memoria de Datos +
- Tiempo de setup para escribir en registros +
- Sesgo (skew) del reloj

Ejemplo: Temporización: Reg-Reg



Duración del ciclo de reloj

CLK to Q ..... **despreciable**

Memory Access Time ..... **10 nseg**

Decodificación ..... **despreciable**

Register File Access Time ..... **5 nseg**

Alu **10nseg**

Ìnstrucción	Memoria instrucc.	Lectura Registros	ALU	Memoria de Datos	Escritura Registros	Total
Formato R	10	5	10		5	30
Load	10	5	10	10	5	40
Store	10	5	10	10		35
Beq	10	5	10			25

FIN Uniciclo

Camino de datos y control:

implementación multiciclo

Recordemos Implementación en un solo ciclo

- Existe un único ciclo y es igual a la demora del camino más largo
- Un load usa cinco unidades.
- Las instrucciones más cortas no se benefician.
- El precio es mayor con instrucciones de punto flotante.
- Cada unidad puede ser usada una sola vez

Dividiendo la ejecución en ciclos

- Búsqueda (Fetch) de instrucción.
- Decodificación y lectura de registros.
- Ejecución, o cálculo de una dirección, o fin de branch.
- Acceso a memoria o fin de instrucciones R.
- Fin de lectura de memoria.

Camino de datos y control: Implementación Multiciclo

1. Introducción.
2. Construcción del camino de datos multiciclo.
3. Ejecución de las instrucciones en varios ciclos de reloj.
4. Diseño de la unidad de control para el camino de datos multiciclo.
5. Implementación de la unidad de control con una ROM.
6. Implementación de la unidad de control con una PLA.
7. Implementación de la unidad de control con un secuenciador.
8. Tratamiento de excepciones.
9. Conclusiones e implicaciones del diseño multiciclo.

1. Introducción

En temas previos:

•Se han presentado varias instrucciones del repertorio del MIPS.

•Se han estudiado los elementos básicos que forman parte de un camino de datos en un computador sencillo.

•Se han presentado algunos caminos de datos sencillos para ciertas operaciones.

•Se ha construido un camino de datos uniciclo para todas las instrucciones presentadas, se ha analizado su funcionamiento y se ha diseñado el control.

En esta parte:

•Se modificará el camino de datos anterior para que la ejecución de las instrucciones dure varios ciclos de reloj.

•Se diseñará una nueva unidad de control como una máquina de estados.

•Se estudiarán otras alternativas de diseño para la unidad de control.

2. Camino de datos multiciclo

Las instrucciones pueden durar un número variable de ciclos en función de la tarea que realicen.

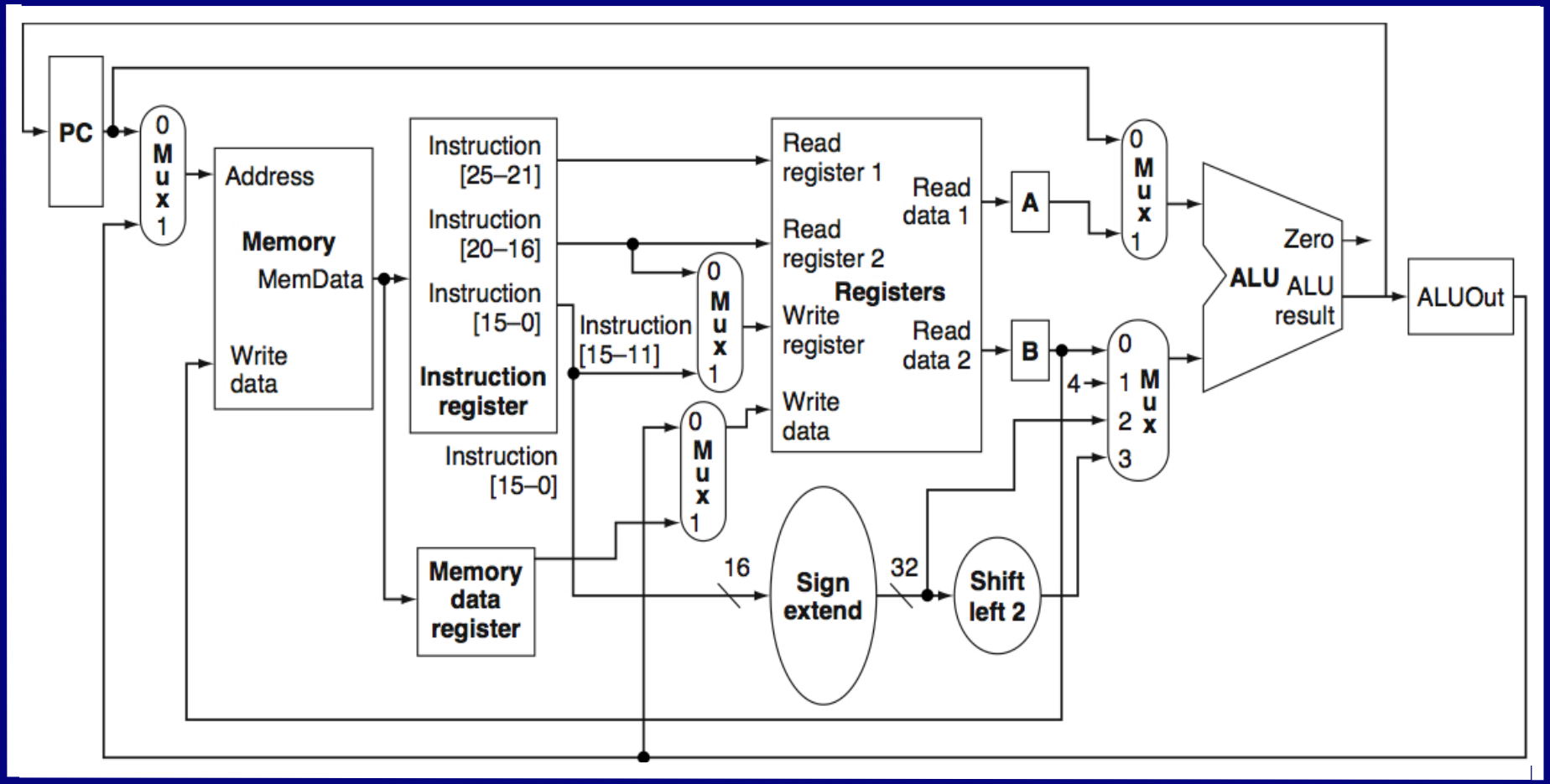
Es preciso añadir varios registros temporales de propósito específico, que almacenen las salidas de las distintas unidades funcionales, para que así dichas salidas estén disponibles en ciclos posteriores.

- Registro de instrucción (instruction register, **IR**): para almacenar la última instrucción leída en memoria.
- Registro de datos de memoria (memory data register, **MDR**): para almacenar el último dato leído de memoria.
- Registros para operandos ubicados en banco de registros: un registro para cada salida del banco de registros (**A** y **B**).
- Registro para la salida de la UAL (**ALUOut**): para guardar el resultado de la UAL (también se le llama en ocasiones registro acumulador).

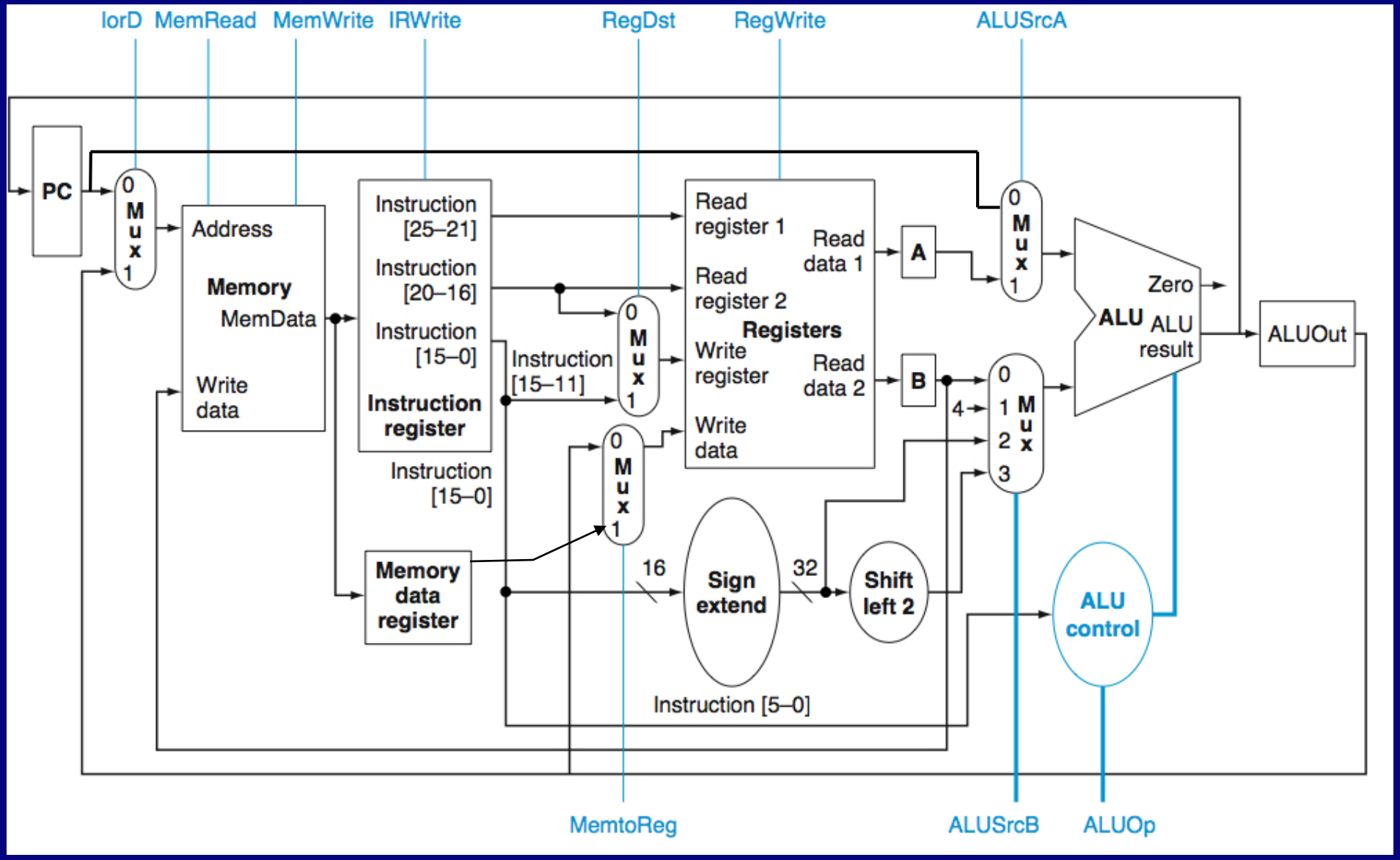
Si la instrucción dura varios ciclos de reloj, una misma unidad funcional puede utilizarse varias veces a lo largo de la ejecución de la instrucción siempre que se haga en ciclos diferentes, y no hace falta replicarla.

- Una única memoria para instrucciones y datos.
- Una sola UAL (se eliminan los dos sumadores adicionales).

Esquema del camino de datos Multiciclo

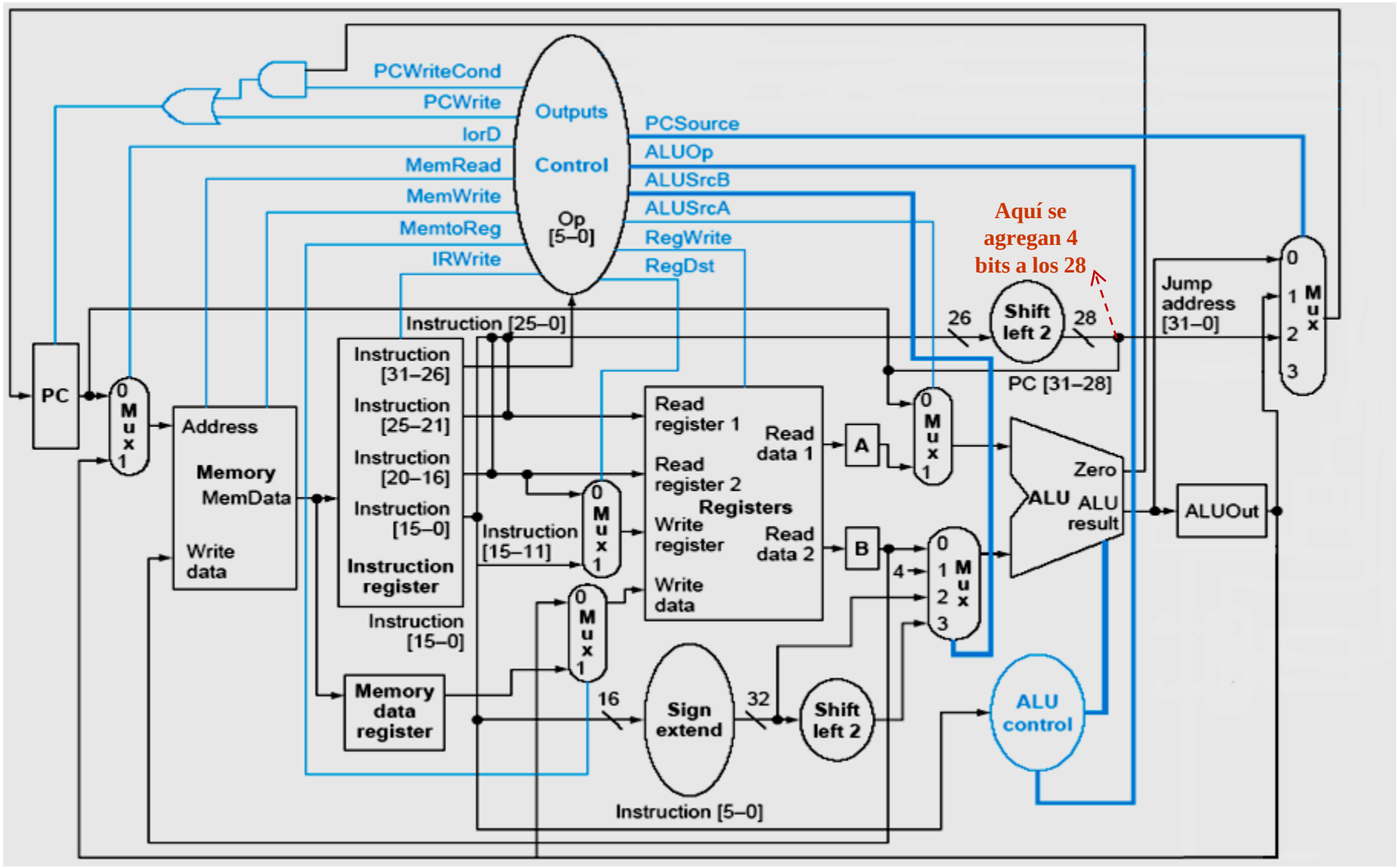


Señales de control en Multiciclo



Camino de datos multiciclo

Camino de datos incluyendo el control para instrucciones de salto (cond. e incond.)



Señales de control

Señal de control	Efecto cuando está inactiva	Efecto cuando está activa
RegDest **RegDst**	El identificador de registro destino está en **rt** (bits 20-16)	El identificador de registro destino está en **rd** (bits 15-11)
EscrReg **RegWrite**	Ninguno	El registro destino se escribe con el valor correspondiente
SelALUA **ALUSrcA**	El primer operando de la UAL proviene del PC	El primer operando de la UAL proviene del registro A
LeerMem **MemRead**	Ninguno	Se lee una posición de memoria y su contenido se coloca a la salida de datos
EscrMem **MemWrite**	Ninguno	Se escribe una posición de memoria con el valor dado en la entrada de datos
MemaReg **MemtoReg**	El valor en la entrada del banco de registros procede de la UAL	El valor de la entrada del banco de registros procede del registro MDR
IoD **IorD**	El PC suministra la dirección para acceder a memoria	ALUOut suministra la dirección para acceder a memoria
EscrIR **IRWrite**	Ninguno	La salida de memoria se escribe en el registro IR
EscrPC **PCWrite**	Ninguno	Escribir el PC un valor que depende de la señal FuentePC (**PCSource**)
EscrPCCond **PCWriteCond**	Ninguno	Escribir el PC cuando la señal de resultado nulo de la UAL está también activa

Señales de control

Señal de control	Valor	Efecto
ALUOp	00	La UAL realiza una operación de suma
	01	La UAL realiza una operación de resta
	10	La operación realizada por la UAL viene dada por el campo **Funct** de la instrucción
SelALUB **ALUSrcB**	00	El segundo operando de la UAL proviene del registro B
	01	El segundo operando de la UAL es la constante 4
	10	El segundo operando de la UAL son los 16 bits menos significativos del IR extendidos en signo a 32 bits
	11	El segundo operando de la UAL son los 16 bits menos significativos del IR extendidos en signo a 32 bits y desplazados dos lugares hacia la izquierda
FuentePC **PCSource**	00	La salida de la UAL (PC+4) se envía para ser escrita en el PC
	01	El contenido del registro ALUOut (destino de la bifurcación condicional) se envía para ser escrito en el PC
	10	La dirección de destino de salto en una instrucción **J** desplazada dos bits hacia la izquierda y concatenada con los 4 bits más significativos del PC se envía para ser escrita en el PC

El control de la UAL se realizará igual que en el camino de datos uniciclo.

3. Ejecución de instrucciones en varios ciclos

En la realización multiciclo, la unidad de control genera y activa todas las señales pertinentes para una instrucción en sucesivos periodos de reloj.

En su ejecución, las instrucciones atraviesan una serie de fases o etapas.

- El número y la secuencia de etapas depende de la propia instrucción.
- La unidad de control se encarga de generar y secuenciar las señales de control.

La información necesaria en un ciclo habrá sido almacenada antes en un registro.

- La lectura o escritura en un registro de propósito específico es muy rápida.

El control del banco de registros es complejo y tiene una sobrecarga mayor (en Tiempo) que el acceso a un registro de propósito específico.

- Para mantener el ciclo de reloj corto, se utiliza un ciclo completo para acceder al banco de registros (en lectura o en escritura).

Objetivo: equilibrar al máximo la duración de las etapas para ajustar la duración del ciclo de reloj de forma que se desperdicie el menor tiempo posible.

- En cada ciclo habrá un acceso a memoria o al banco de registros o una operación UAL.
- Un mismo ciclo puede incluir varias de estas operaciones, con la condición de que se realicen todas ellas en paralelo de forma simultánea, y no en secuencia.

Etapas o ciclos en una instrucción

1. Carga de la instrucción (*instruction fetch*)

Se lee una instrucción de memoria y se carga en el IR, y además se incrementa el PC.

2. Decodificación de la instrucción (*instruction decoding*)

A veces la circuitería de control necesaria para averiguar cuál es la instrucción leída es compleja, de ahí que se dedique a la decodificación un ciclo de reloj completo.

Al mismo tiempo pueden realizarse también:

•Operaciones comunes a todas las instrucciones y que puedan adelantar su ejecución.

•Operaciones que puedan adelantar la ejecución de ciertas instrucciones y que no resulten perjudiciales en caso de que la instrucción leída sea distinta a la esperada.

Son:

Lectura de registros en el banco.

Cálculo de direcciones de salto.

Las siguientes etapas dependen de cuál sea la instrucción leída.

...Etapas o ciclos en una instrucción

3. Ejecución UAL, cálculo de la dirección de memoria, o finalización de salto

La UAL realiza una operación

- En instrucciones R se ejecuta la operación pedida.
- En instrucciones **Lw** o **Sw** se calcula la dirección del operando de memoria.
- En bifurcaciones se comparan los registros para determinar el valor de la condición, y en su caso se carga el nuevo valor en el PC.
- En saltos incondicionales se escribe el nuevo valor en el PC.

4. Acceso a memoria o finalización de instrucciones R

Se efectúa el acceso a memoria o se escribe el resultado en instrucciones R.

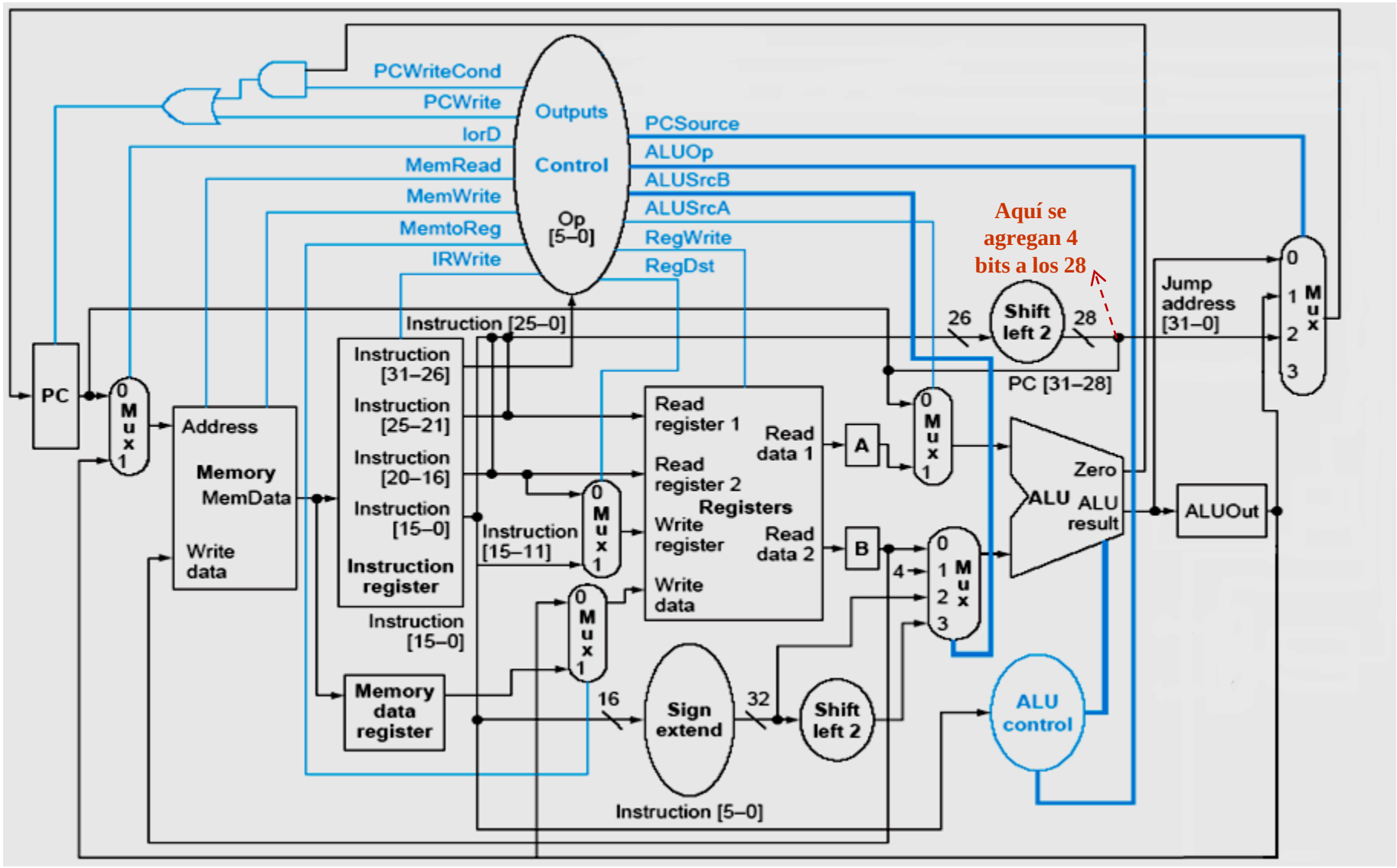
- En instrucciones **R** se escribe en el banco de registros el resultado de la operación de la etapa anterior.
- En instrucciones **Lw** se lee el dato de memoria.
- En instrucciones **Sw** se escribe el dato en memoria.

5. Finalización de lectura en memoria

En instrucciones **Lw** se escribe el dato leído en el banco de registros.

Camino de datos multiciclo

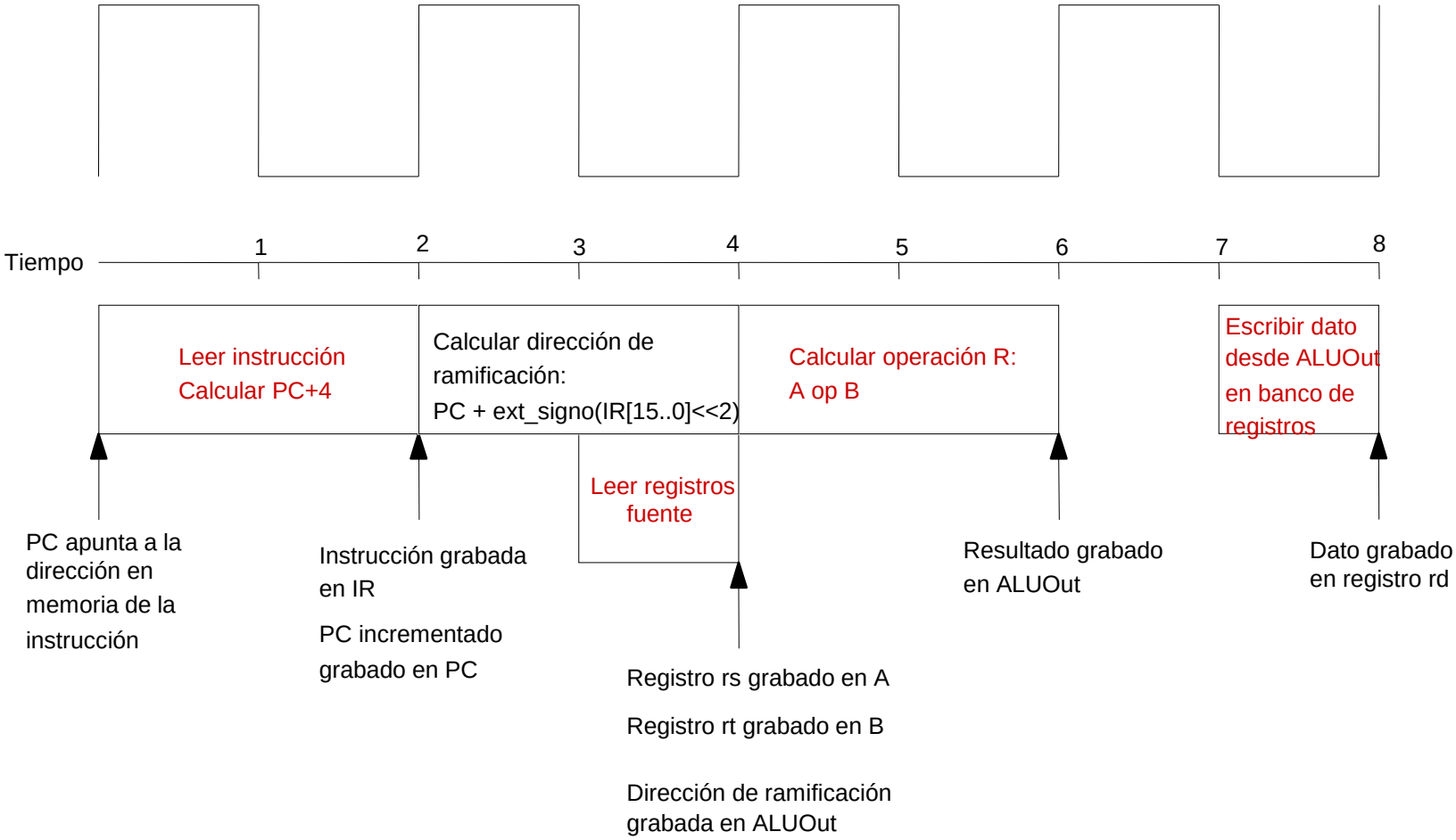
Camino de datos incluyendo el control para instrucciones de salto (cond. e incond.)



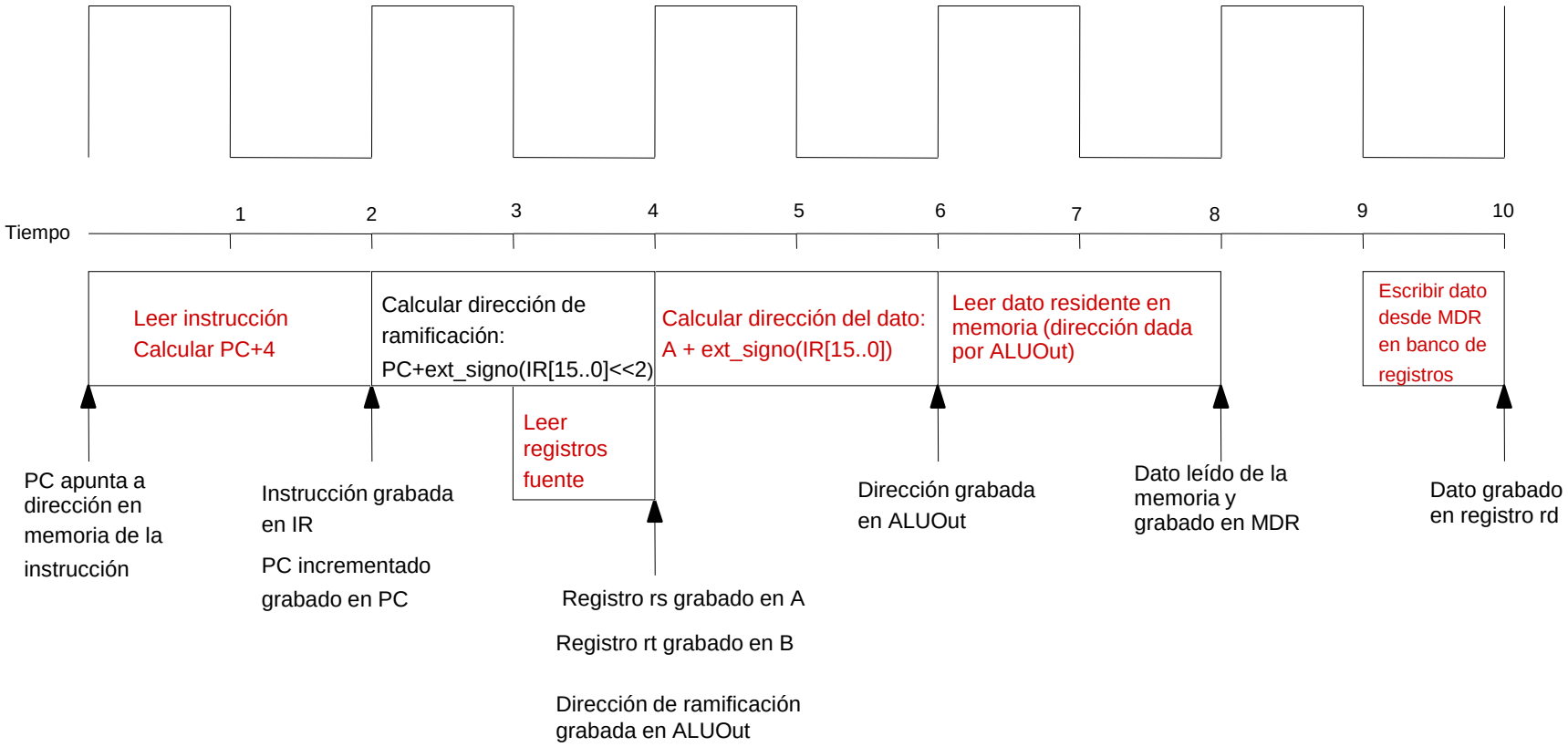
Etapas o ciclos en una instrucción: resumen

Nombre de la etapa	Acción en instrucción R	Acción en instrucción de memoria	Acción en instrucción de ramificación condicional	Acción en instrucción de salto
Carga de instrucción	**IR = Memoria[PC]** **PC = PC+4**			
Decodificación de instrucción / carga de registros	**A = Reg[IR[25-21]]** **B = Reg[IR[20-16]]** **ALUOut = PC + (extensión_signo(IR[15-0]) << 2)**			
Ejecución / cálculo de dirección / finalización de salto y ramificación	**ALUOut = A op B**	**ALUOut = A + extensión_signo (IR[15-0])**	**Si (A==B) entonces PC = ALUOut**	**PC = PC[31-28] || (IR[25-0] << 2)**
Acceso a memoria / finalización de instrucción R	**Reg [IR[15-11]] = ALUOut**	**LW: MDR = Memoria[ALUOut]** **SW: Memoria[ALUOut] = B**		
Finalización de instrucción de lectura en memoria		**LW: Reg[IR[20-16]] = MDR**		

Ejecución de instrucciones multiciclo: tipo R



Ejecución de instrucciones multiciclo: Lw



Ejecución de instrucciones multiciclo: beq

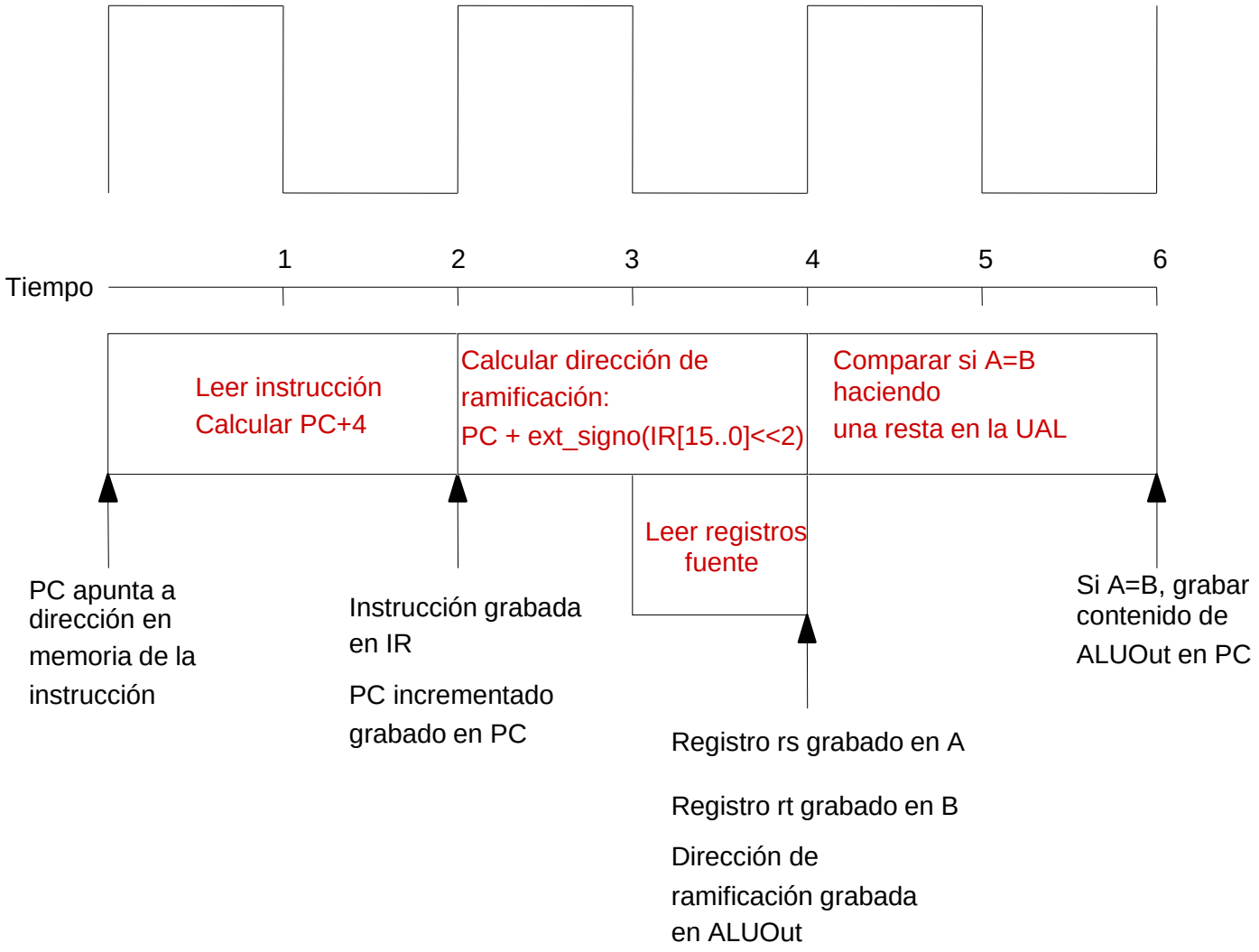
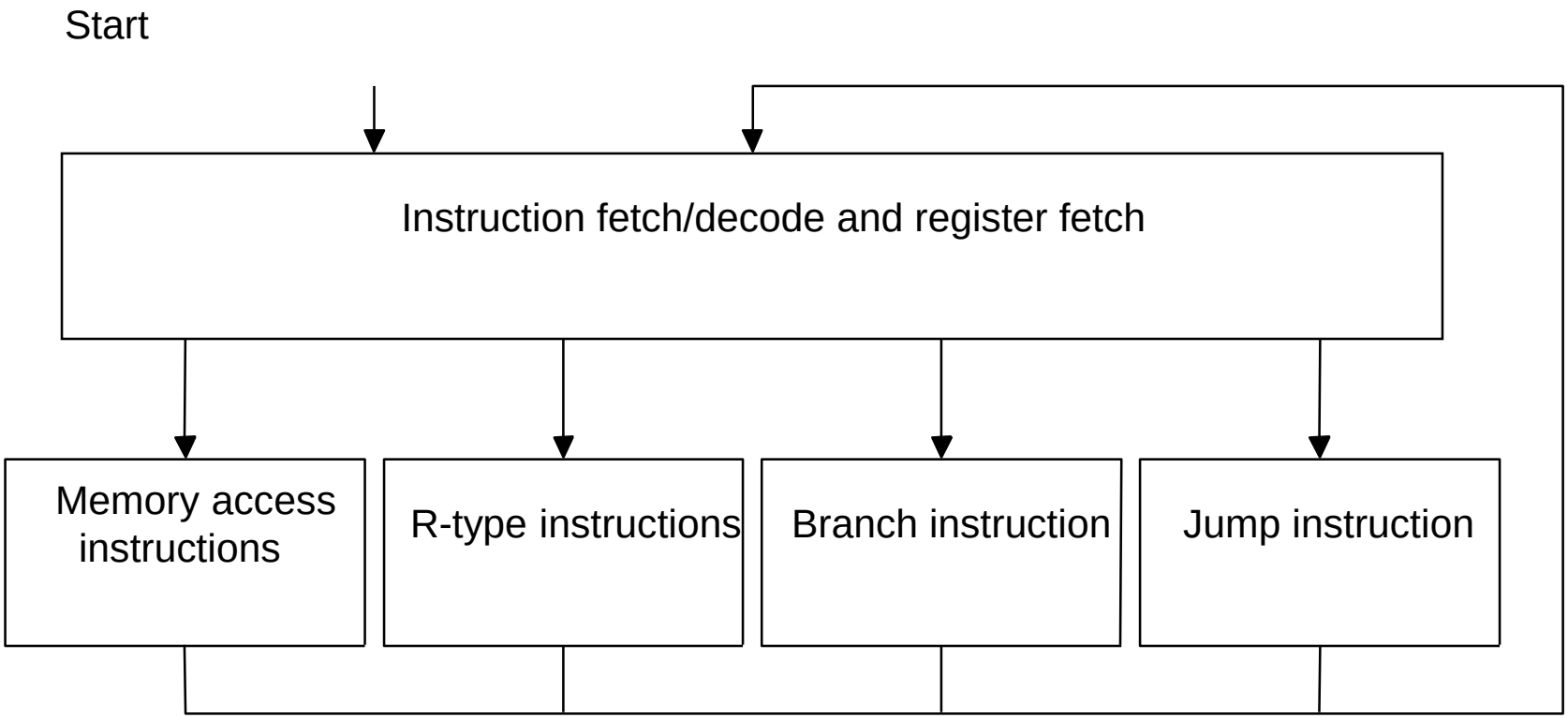


Diagrama de estados de la unidad de control



Camino de datos multiciclo

Camino de datos incluyendo el control para instrucciones de salto (cond. e incond.)

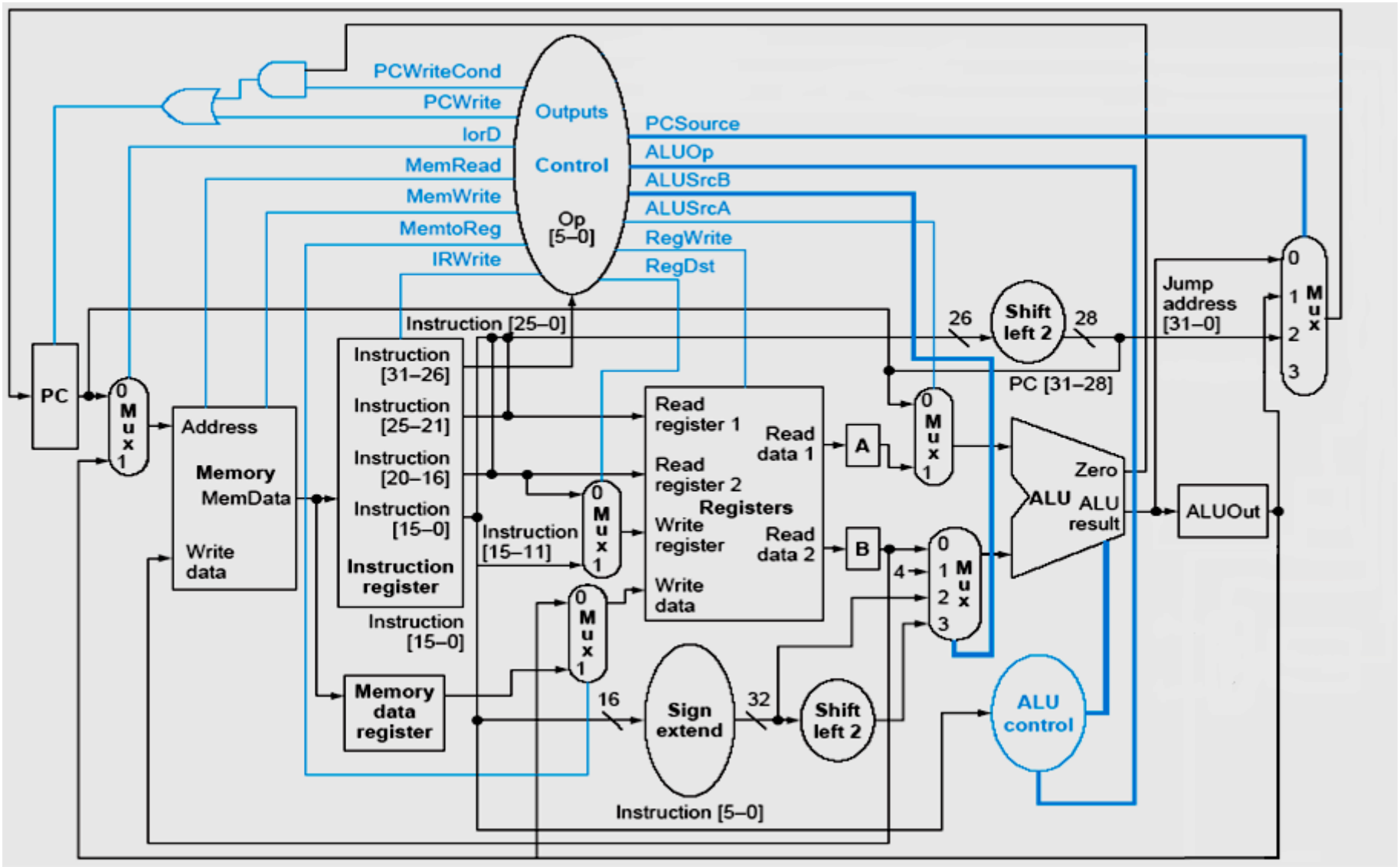
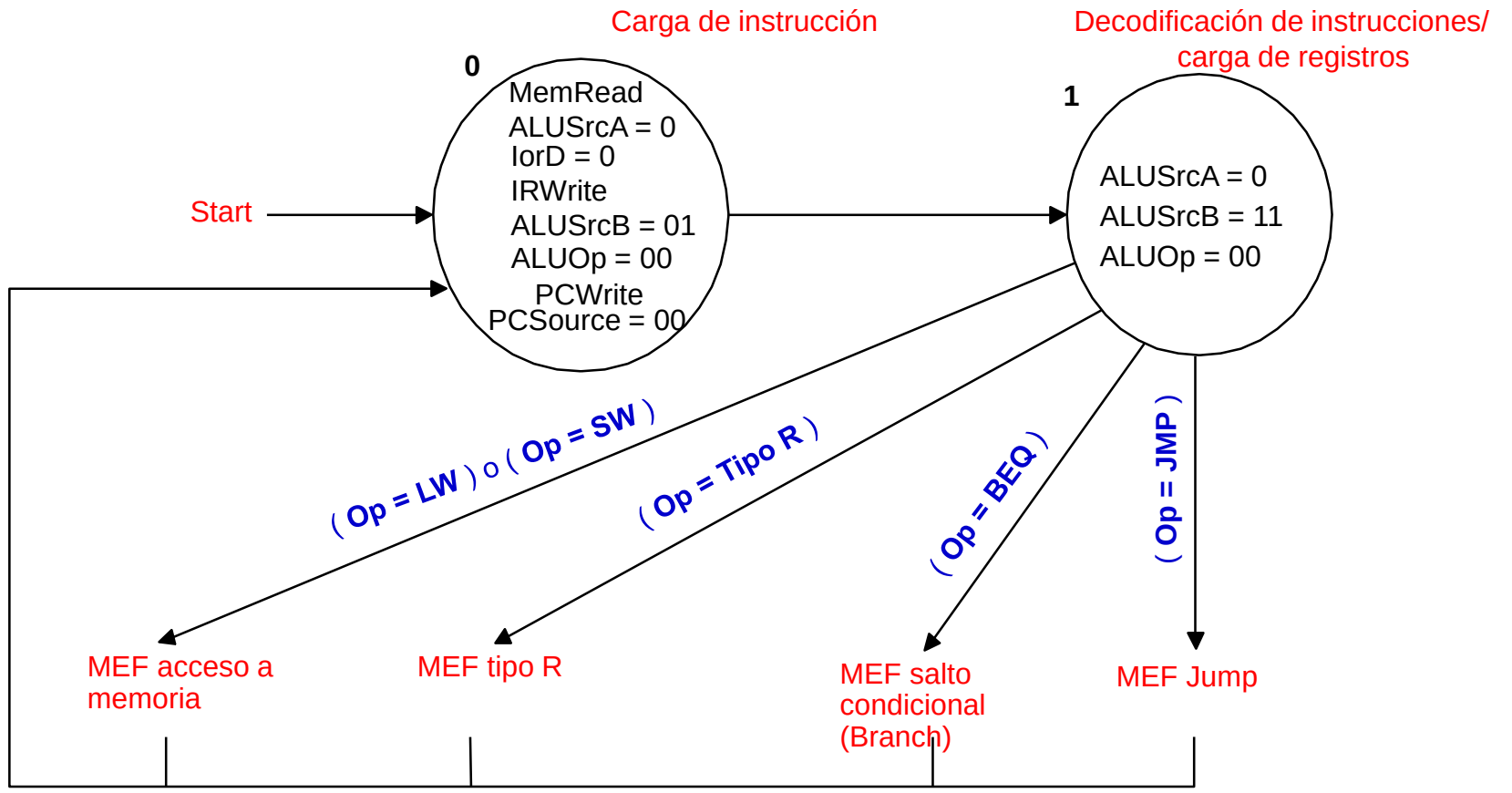


Diagrama de estados: lectura y decodificación

En cada estado se especifican sólo las señales activas.

•Las señales no especificadas están inactivas.



MEF=Maquina de Estados Finitos

Diagrama de estados: ejecución de Instrucciones de acceso a memoria

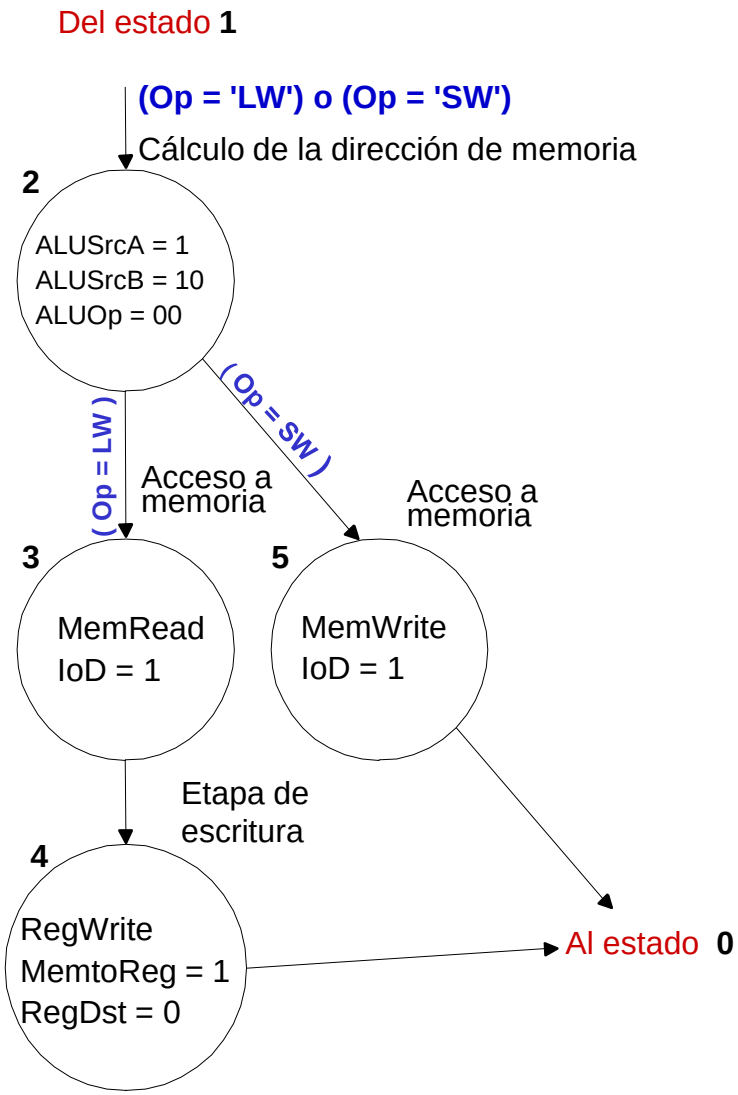
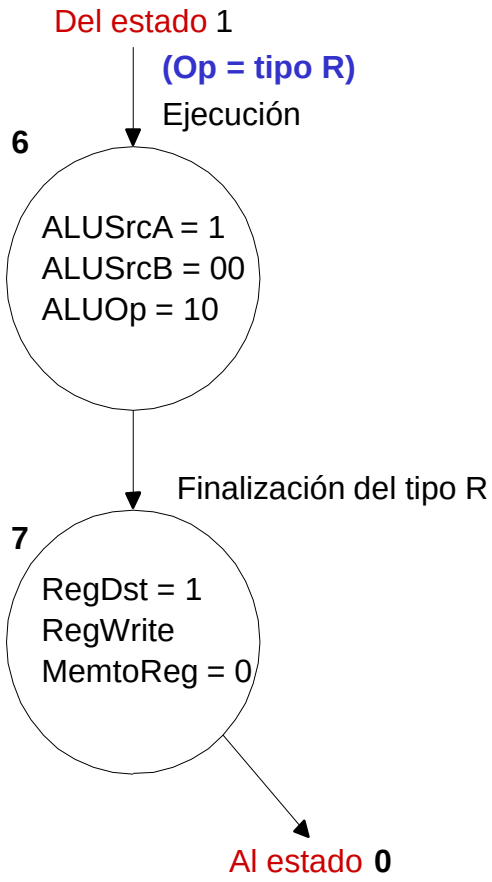


Diagrama de estados: ejecución de Instrucciones de tipo R



Camino de datos multiciclo

Camino de datos incluyendo el control para instrucciones de salto (cond. e incond.)

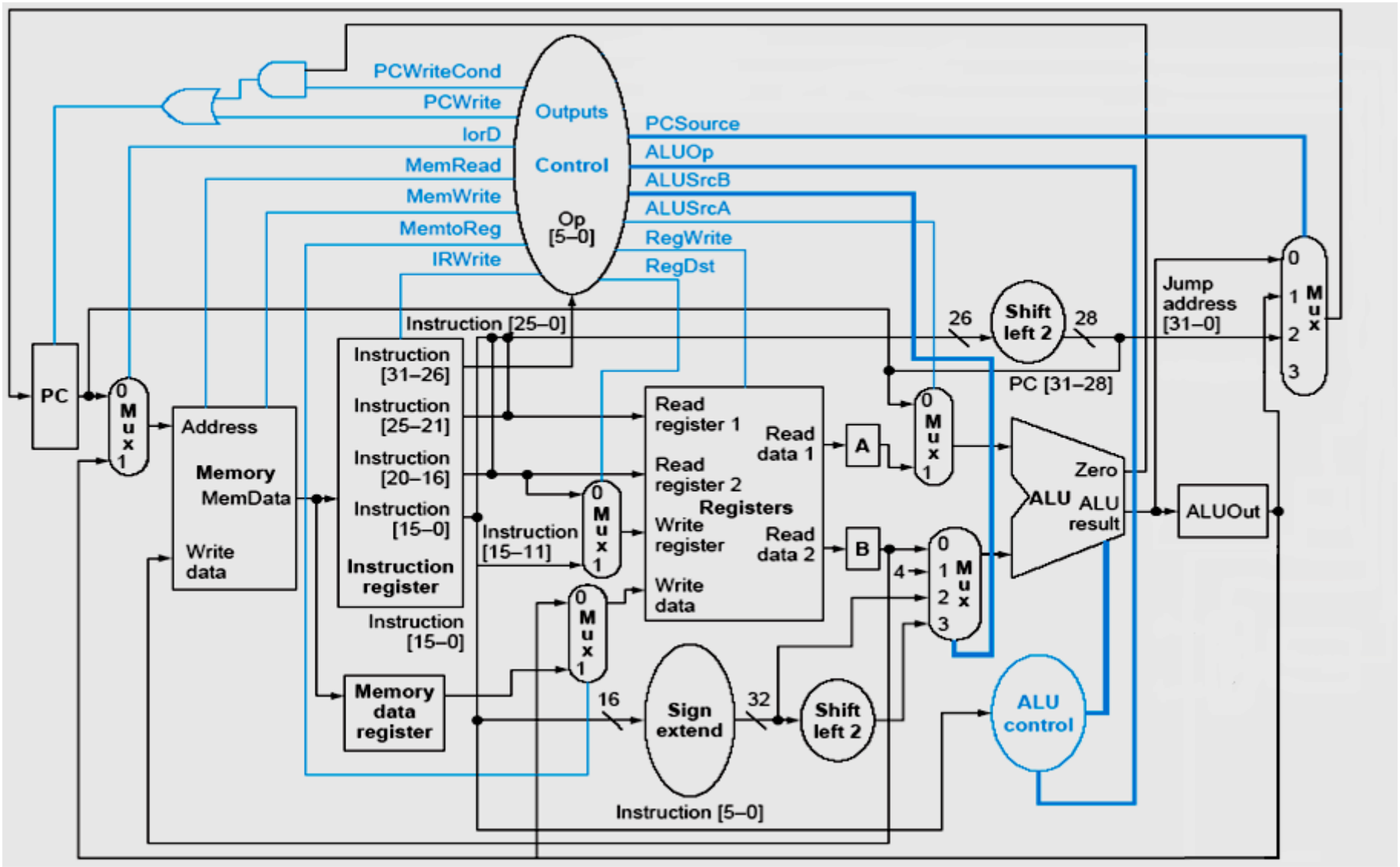
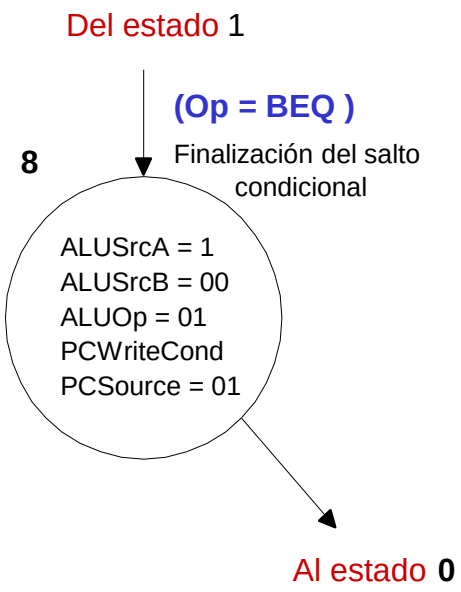
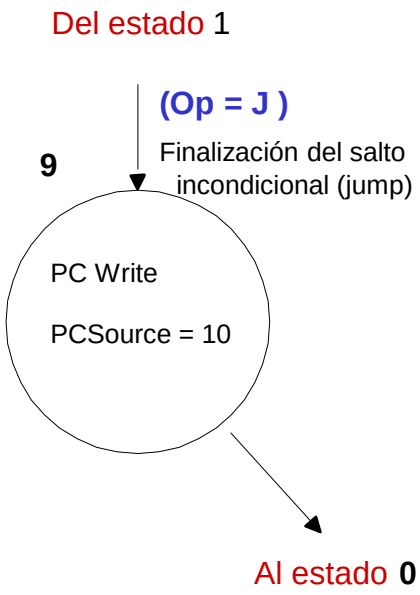


Diagrama de estados: ejecución

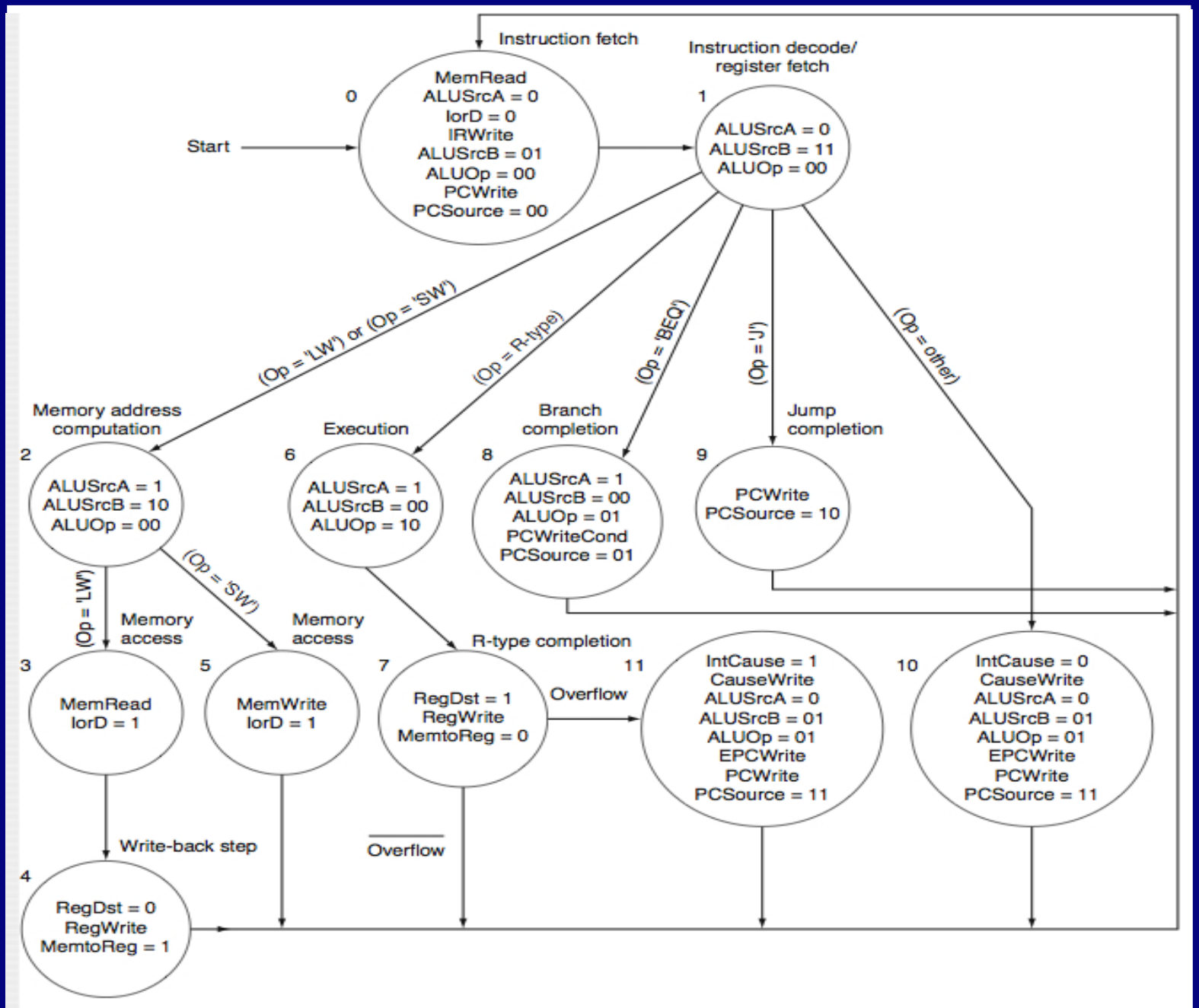
Instrucción de salto condicional



Instrucción de salto incondicional

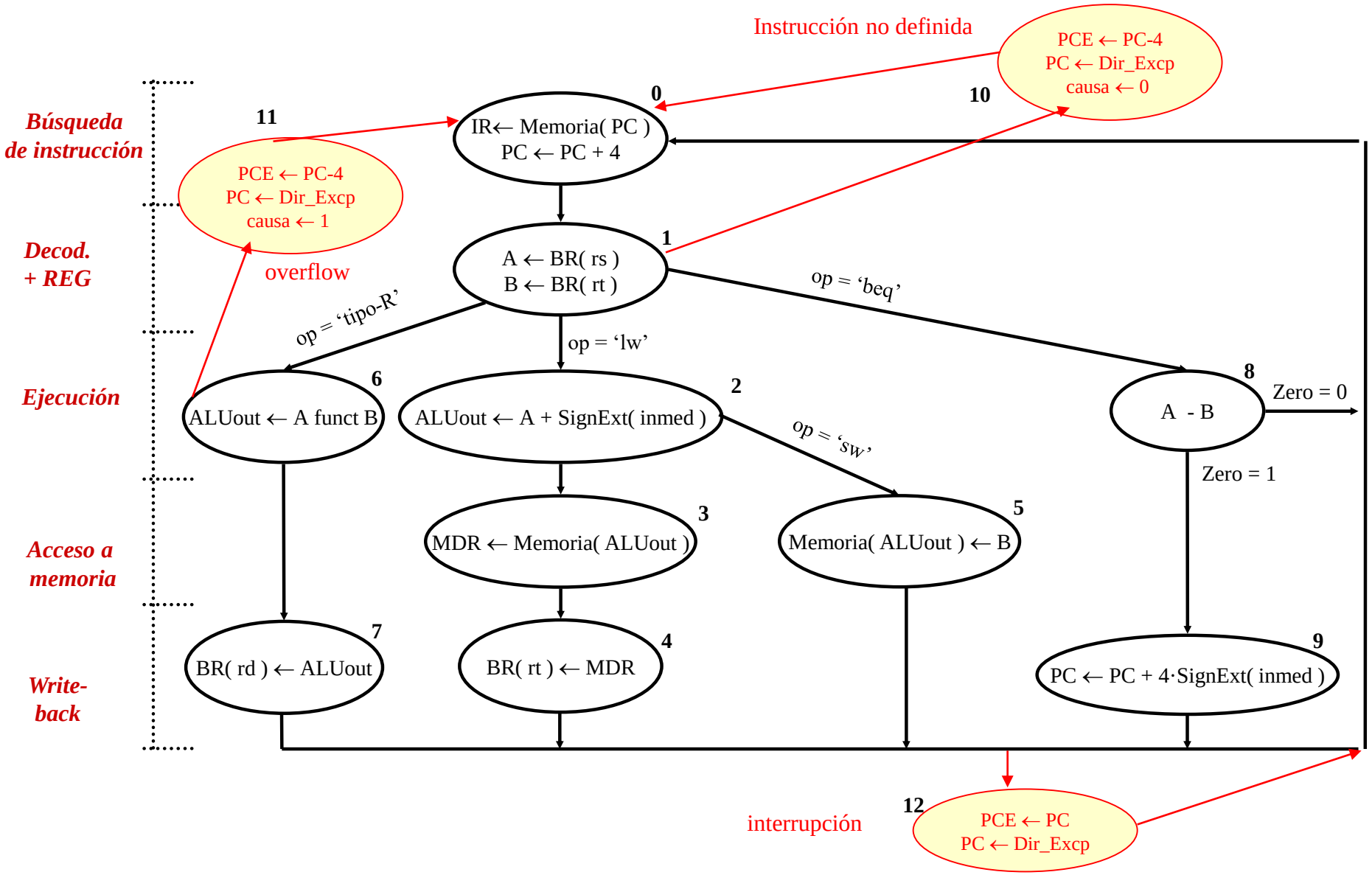


Autómata con las señales de control en multiciclo



Recordatorio

- **Diagrama de estados del controlador multiciclo** (inf. de otra fuente)

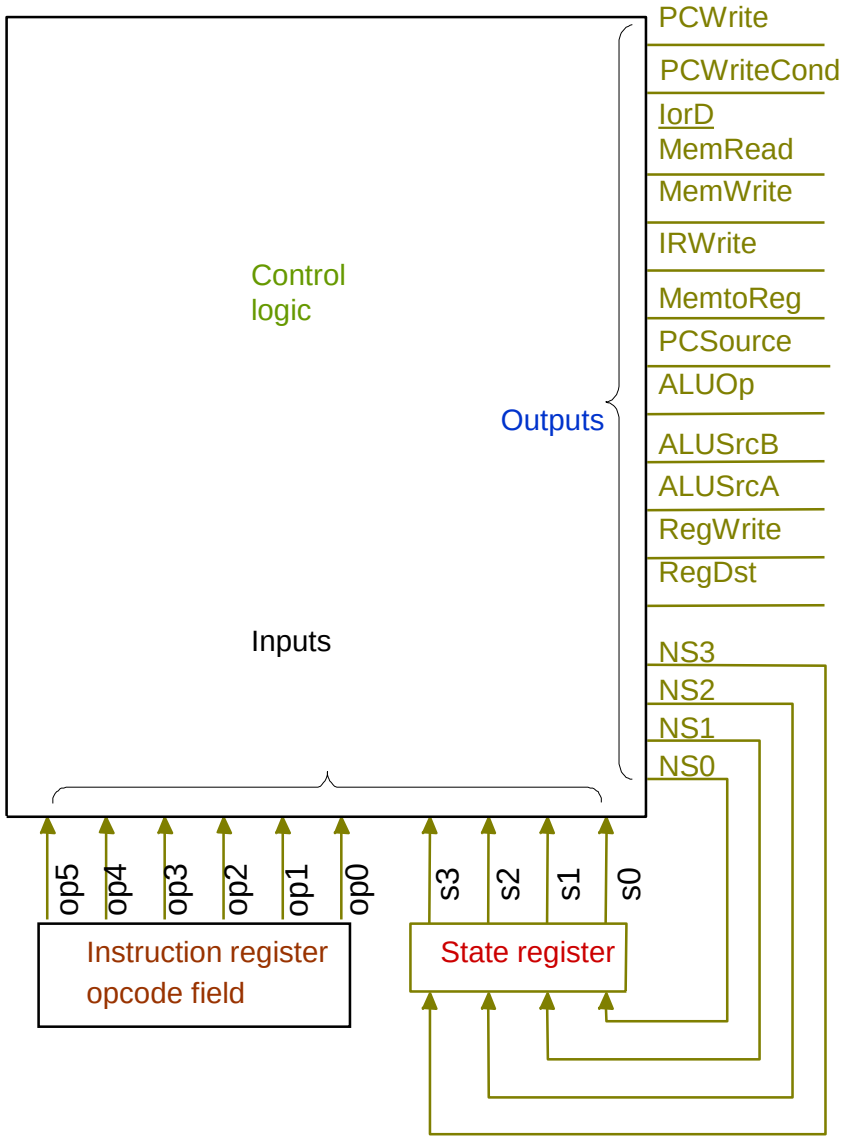


Esquema de la unidad de control

La unidad de control constará de:

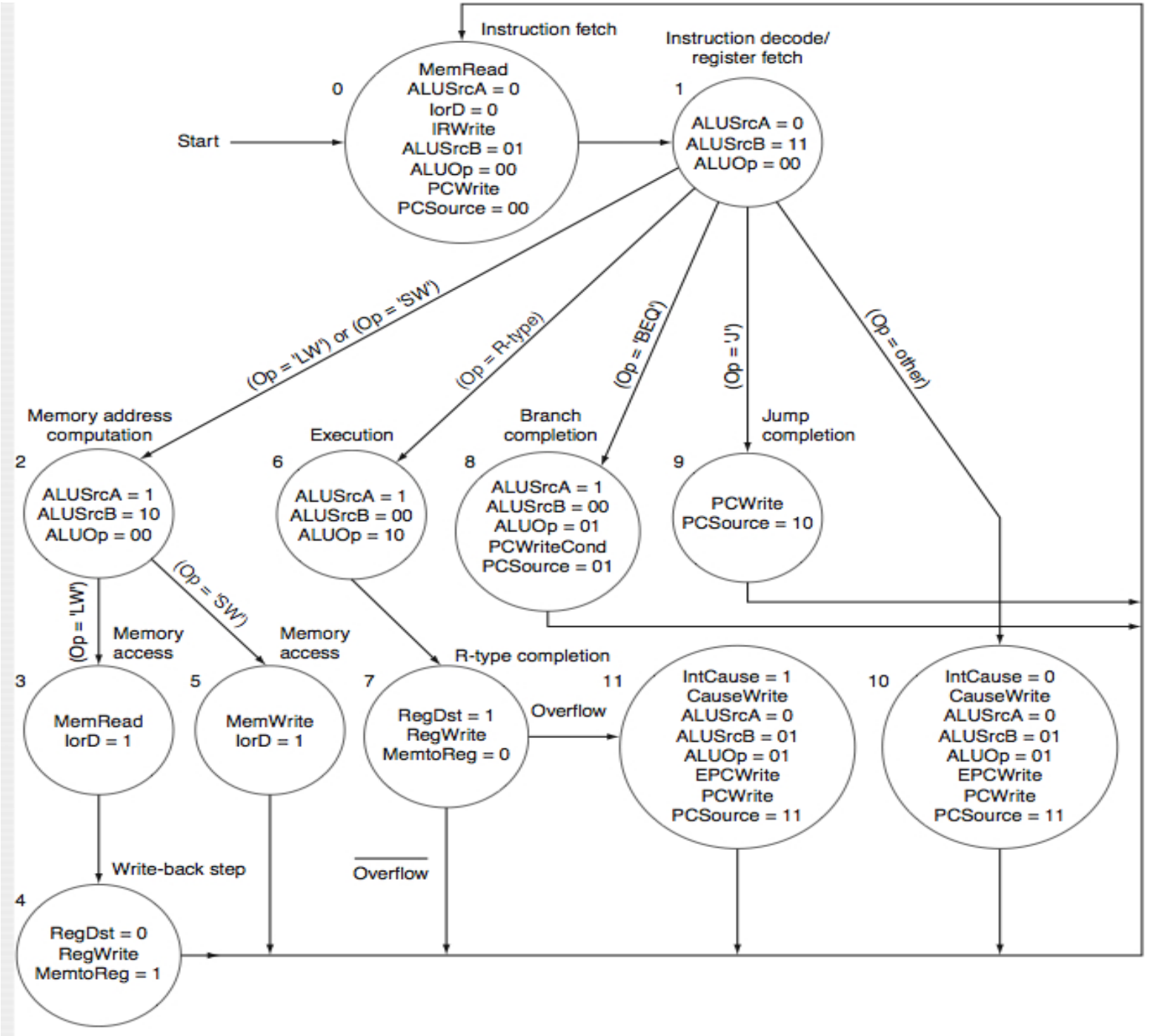
- Un registro para almacenar el estado de la MEF (state register).
- La circuitería necesaria para realizar la transición entre estados (lógica de control para las transiciones).
- La circuitería necesaria para generar las salidas o señales de control (lógica de control para las salidas).

La unidad de control utilizará como entrada el contenido del registro de instrucción (IR).



Función de control: definiciones

Control	Señal	Ecuación
Entradas	(Op = ‘lw’)	$Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$
	(Op = ‘sw’)	$Op5\ \overline{Op4}\ Op3\ \overline{Op2}\ Op1\ Op0$
	(Op = ‘FormatoR’)	$\overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}$
	(Op = ‘beq’)	$\overline{Op5}\ \overline{Op4}\ \overline{Op3}\ Op2\ \overline{Op1}\ \overline{Op0}$
	(Op = ‘j’)	$\overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ \overline{Op0}$
Estados		S3 S2 S1 S0
	Estado0	0 0 0 0
	Estado1	0 0 0 1
	Estado2	0 0 1 0
	Estado3	0 0 1 1
	Estado4	0 1 0 0
	Estado5	0 1 0 1
	Estado6	0 1 1 0
	Estado7	0 1 1 1
	Estado8	1 0 0 0
	Estado9	1 0 0 1


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
Overflow
4
RegDst = 0
RegWrite
MemtoReg = 1

Función de control: transiciones

El estado siguiente depende del estado actual y del contenido de IR.

En varios casos el estado siguiente sólo depende del estado actual, ya que la máquina de estados se ha construido del siguiente modo:

- Una secuencia de dos estados (0 y 1) independiente de IR.
- Una transición que depende de IR.
- Una secuencia independiente de IR en cada rama (salvo en **lw** y **sw**).
- Al final de cada rama se vuelve al estado 0.

Estado siguiente	Ecuación booleana
NuevoEstado0	Estado4 + Estado5 + Estado7 + Estado8 + Estado9
NuevoEstado1	Estado0
NuevoEstado2	Estado1 · [(Op = ‘lw’) + (Op = ‘sw’)]
NuevoEstado3	Estado2 · (Op = ‘lw’)
NuevoEstado4	Estado3
NuevoEstado5	Estado2 · (Op = ‘sw’)
NuevoEstado6	Estado1 · (Op = ‘FormatoR’)
NuevoEstado7	Estado6
NuevoEstado8	Estado1 · (Op = ‘beq’)
NuevoEstado9	Estado1 · (Op = ‘j’)

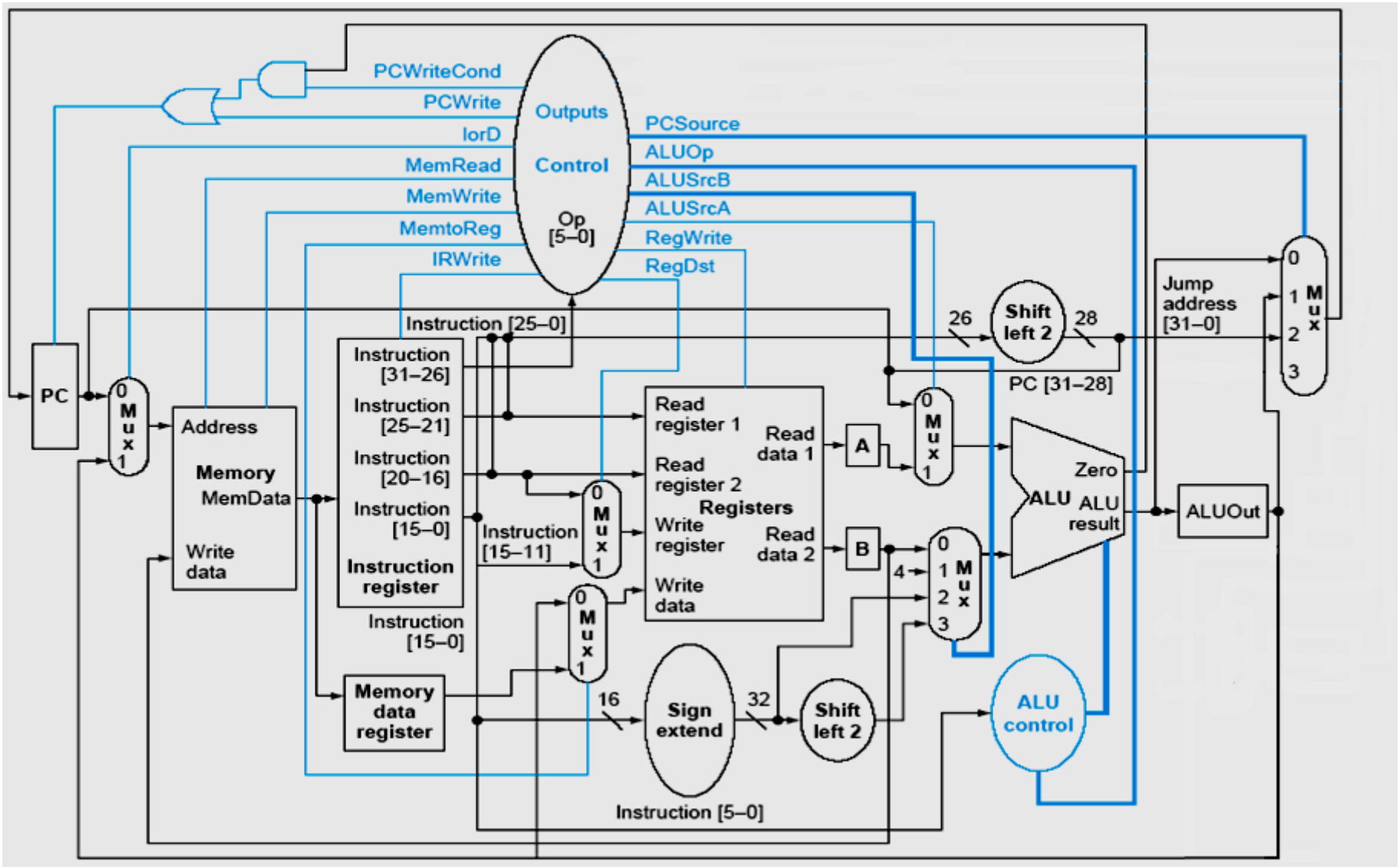
Función de control: salidas

En una máquina de Moore, las salidas sólo dependen del estado actual.

Salida	Ecuación booleana
EscrPC (**PCWrite**)	Estado0 + Estado9
EscrPCCond (**PCWriteCond**)	Estado8
IoD (**IorD**)	Estado3 + Estado5
LeerMem (**MemRead**)	Estado0 + Estado3
EscrMem (**MemWrite**)	Estado5
EscrIR (**IRWrite**)	Estado0
MemaReg (**MemtoReg**)	Estado4
PCSource1 (bit1)	Estado9
PCSource0 (bit0)	Estado8
ALUOp1	Estado6
ALUOp0	Estado8
SelALUB1(**ALUSrcB1**)	Estado1 + Estado2
SelALUB0(**ALUSrcB0**)	Estado0 + Estado1
SelALUA(**ALUSrcA**)	Estado2 + Estado6 + Estado8
EscrReg (**RegWrite**)	Estado4 + Estado7
RegDest (**RegDst**)	Estado7

Camino de datos multiciclo

Camino de datos incluyendo el control para instrucciones de salto (cond. e incond.)



5. Implementación del control con una ROM

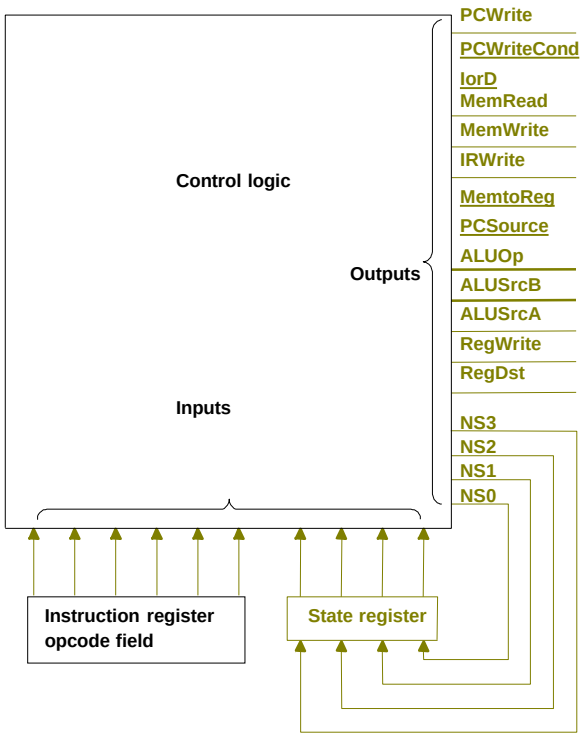
La lógica de control se puede materializar mediante una ROM que contenga las salidas y las transiciones de estado para todas las posibles combinaciones de los contenidos del IR y del registro de estado.

Número de entradas de dirección= 10 =Número de bits del registro de estado (4) + número de bits del código de operación (6)

Número de palabras de la memoria: 2^{10}= 1024.

Ancho de cada palabra: 20 bits= Número de salidas de control (16) + número de bits del registro de estado (4)

Tamaño total de la memoria: 20 Kbits.



Comentarios:

- Implementar el control mediante una ROM es la opción más sencilla y flexible.
- El control normalmente se realizará mediante una máquina de Moore.

Control con una ROM: salidas

Estado actual ⇓	Bits 19-4 de las filas de la ROM															
	PCWrite	PCWriteCond	IorD	MemRead	MemWrite	IRWrite	MemtoReg	PCSource 1	PCSource 0	ALUop 1	ALUop 0	ALUSrcB1	ALUSrcB0	ALUSrcA	RegWrite	RegDst
0000	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0
0001	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
0010	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
0011	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0100	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
0101	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
0110	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
0111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
1000	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0
1001	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
Resto (estados 1010 hasta 1111)																

Cada fila (estado) se repite 64 veces en la ROM (hay 2^6= 64 códigos de operación), variando los bits de la función de transición (mostrados en la tabla de la transparencia siguiente). La fila etiquetada “Resto” representa 64 x 6 = 384 filas en la ROM que nunca se usan.

Control con una ROM: transiciones de estado

Estado actual ⇓	Códigos de operación (Op[5-0]) 000000 (Instrucción R)	000010 (Instrucción J)	000100 (Instrucción BEQ)	100011 (Instrucción LW)	101011 (Instrucción SW)	Cq otro valor
0000	**0001**	**0001**	**0001**	**0001**	**0001**	**0001**
0001	**0110**	**1001**	**1000**	**0010**	**0010**	ilegal
0010	**xxxx**	**xxxx**	**xxxx**	**0011**	**0101**	ilegal
0011	**0100**	**0100**	**0100**	**0100**	**0100**	ilegal
0100	**0000**	**0000**	**0000**	**0000**	**0000**	ilegal
0101	**0000**	**0000**	**0000**	**0000**	**0000**	ilegal
0110	**0111**	**0111**	**0111**	**0111**	**0111**	ilegal
0111	**0000**	**0000**	**0000**	**0000**	**0000**	ilegal
1000	**0000**	**0000**	**0000**	**0000**	**0000**	ilegal
1001	**0000**	**0000**	**0000**	**0000**	**0000**	ilegal
Resto (estados 1010 hasta 1111)	**?**	**?**	**?**	**?**	**?**	

Cada casilla muestra los 4 bits de la palabra de control cuya dirección viene dada por los bits del estado actual y los bits Op. Por ej. Si el estado es 0000 la salida es siempre 0001. Notar que en este caso hay 64 posiciones en la ROM con el mismo valor=0001 ya que el código de operación puede tomar cq. Valor.

Hay muchas casillas (filas de la ROM) no accesibles:

- "?">> estados no existentes en la máquina.
- XXXX: siguiente estado indiferente (combinaciones imposibles).
- Ilegal: códigos de operación no especificados (cada casilla son 64 filas de la ROM).

Control con una ROM

Hay muchas posiciones sobrantes en la ROM.

•Hay combinaciones (estado,código de operación) imposibles (por ejemplo, estado 2 con instrucción R): hay posiciones a las que nunca se accederá (y por tanto sobran).

•Si el número de estados **s** no es potencia de 2, sobran las posiciones correspondientes a estados inexistentes (estados 10 al 15).

Hay mucha información repetida en la ROM.

•Si el tamaño del código de operación es **k**, a cada estado le corresponden 2^{k} posiciones de la ROM, y todas ellas contienen los mismos valores para los bits de salida (en nuestro caso, las salidas se repiten 2^{6}= 64 veces por cada estado).

•Hay secuencias que no dependen del código de operación, por tanto la información de transición en ciertos estados se ha rellenado repitiendola 2^{k} veces (estados 0 y 3-9).

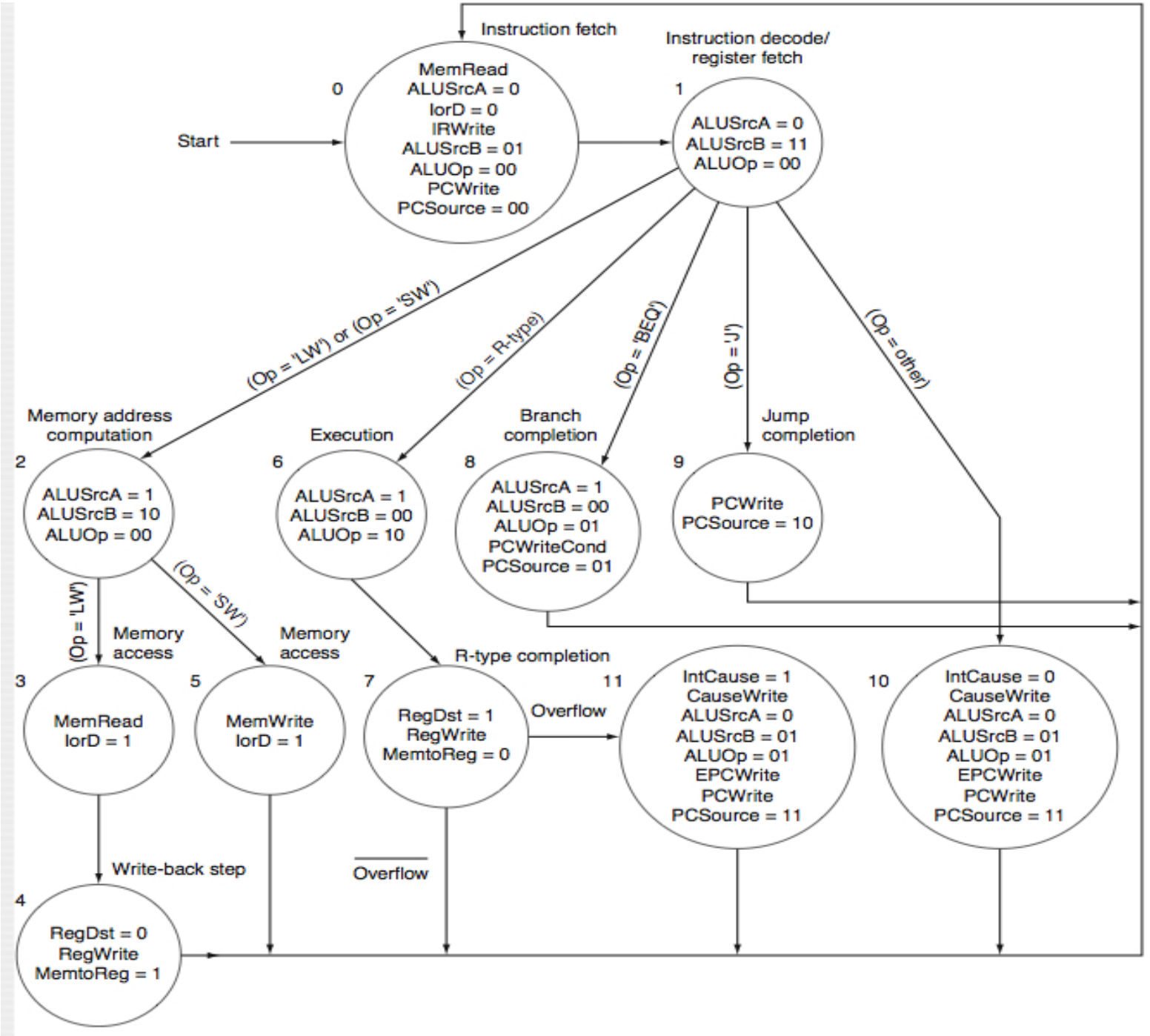
Conclusión: usar una ROM en estos casos puede ser un despilfarro.

Una solución posible es usar una ROM para las transiciones de estado y otra para las salidas (¿cuáles serían sus tamaños?).

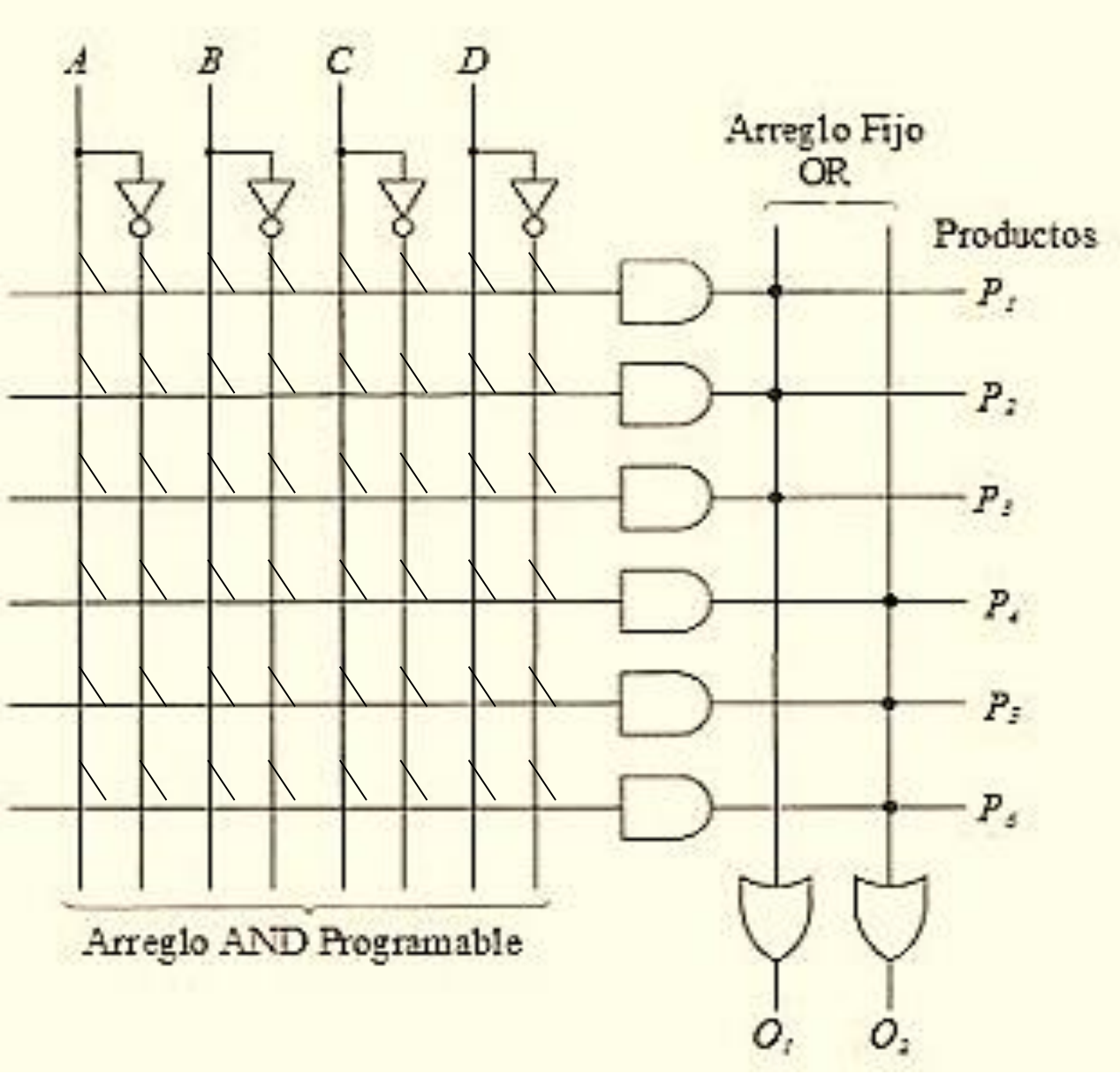
•Aún sobraría información (estados inexistentes, transiciones independientes de IR, etc).

Función de control: definiciones

Control	Señal	Ecuación
Entradas	(Op = ‘lw’)	$Op5\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ Op0$
	(Op = ‘sw’)	$Op5\ \overline{Op4}\ Op3\ \overline{Op2}\ Op1\ Op0$
	(Op = ‘FormatoR’)	$\overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ \overline{Op1}\ \overline{Op0}$
	(Op = ‘beq’)	$\overline{Op5}\ \overline{Op4}\ \overline{Op3}\ Op2\ \overline{Op1}\ \overline{Op0}$
	(Op = ‘j’)	$\overline{Op5}\ \overline{Op4}\ \overline{Op3}\ \overline{Op2}\ Op1\ \overline{Op0}$
Estados		S3 S2 S1 S0
	Estado0	0 0 0 0
	Estado1	0 0 0 1
	Estado2	0 0 1 0
	Estado3	0 0 1 1
	Estado4	0 1 0 0
	Estado5	0 1 0 1
	Estado6	0 1 1 0
	Estado7	0 1 1 1
	Estado8	1 0 0 0
	Estado9	1 0 0 1


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
Overflow
4
RegDst = 0
RegWrite
MemtoReg = 1

Arreglo Lógico Programable (PLA)



6. Implementación del control con una PLA

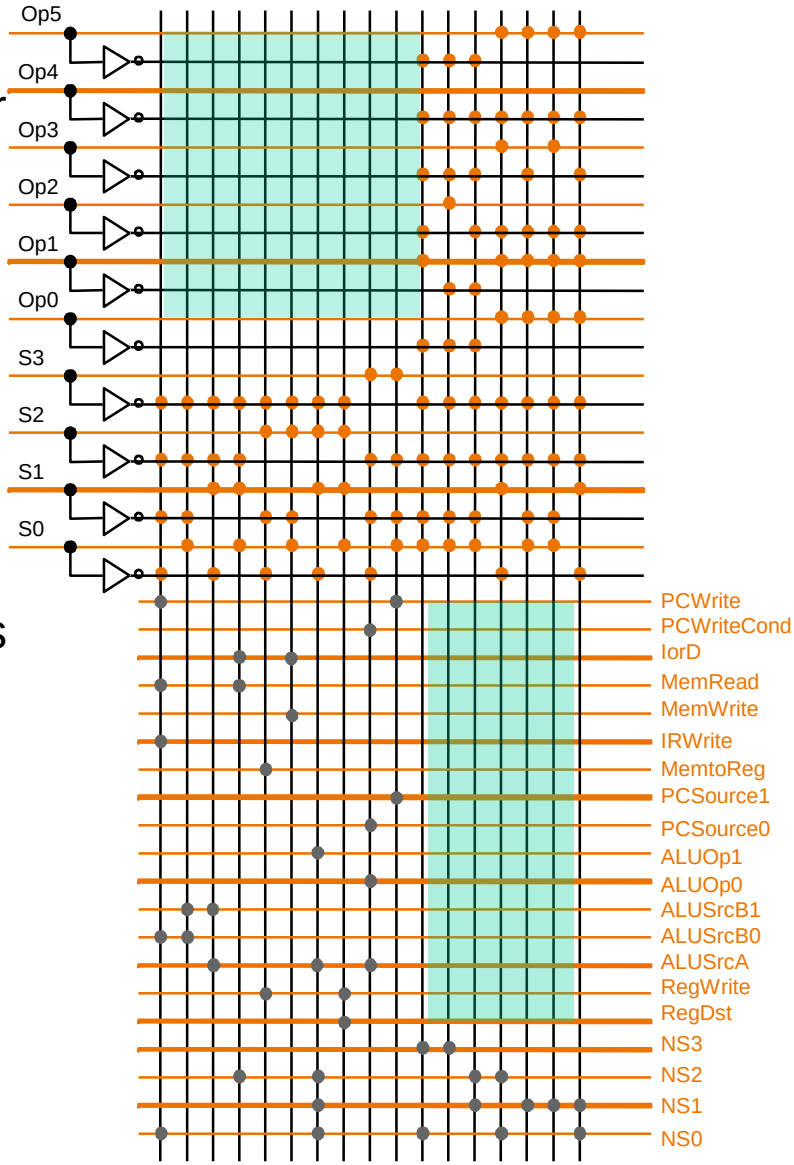
Esta opción es más sensata que utilizar una ROM, ya que en la PLA se pueden simplificar los términos redundantes y sobrantes.

El tamaño de una PLA es proporcional al término (número de entradas x número de términos producto + número de salidas x número de términos producto).

En nuestro caso, la PLA tendría 17 minterms únicos de 10 variables y 20 salidas.

- El tamaño de la PLA de la figura es proporcional a (10 x 17 + 20 x 17) = 510.

Para reducir el tamaño de la PLA, podría dividirse en dos: una PLA para la función de transición y otra PLA para la de salida



7. Implementación del control con un secuenciador

En la realización multiciclo mediante dos ROM, la mayor parte de la lógica se emplea para materializar la función de transición.

Si crece el repertorio de instrucciones, también crece el número de estados, y con él la lógica de control, especialmente la dedicada a la función de transición.

Si hay instrucciones que duran muchos ciclos de reloj, en su ejecución se producen largas secuencias de estados con un solo camino que los une.

- Se producen muchísimas combinaciones estado+IR imposibles.
- Sin embargo, gran parte de la lógica se dedica a la función de transición.

Por todo ello, la función de transición se puede codificar más eficientemente utilizando un contador que se incremente automáticamente en cada ciclo de reloj.

- Así se evita codificar explícitamente la función de transición en la unidad de control, ahorrando gran cantidad de lógica.

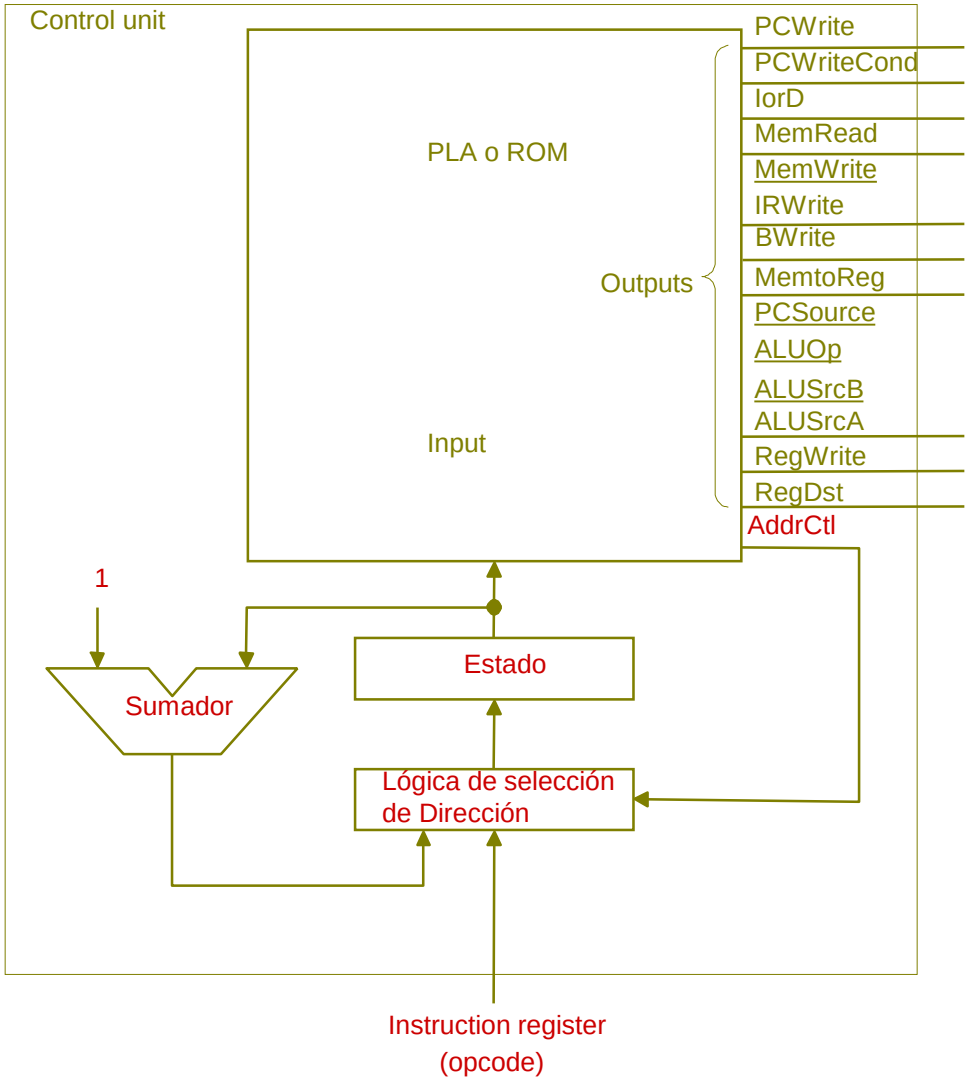
- Es preciso resolver situaciones en las que el estado siguiente no sea el estado anterior incrementado (“saltos” o “envíos”).

Control con un secuenciador

El contador se puede construir mediante un registro más un incrementador (sumador que suma 1 al estado anterior).

Para tener en cuenta los “saltos” o “envíos” se añadirá:

- Una lógica (tabla) para calcular el estado (dirección) siguiente de la MEF si hay un “salto”.
- Una señal de control que permita seleccionar entre la dirección (estado) incrementada y la dirección de “salto” (**CtrlDir** o **AddrCtl**).



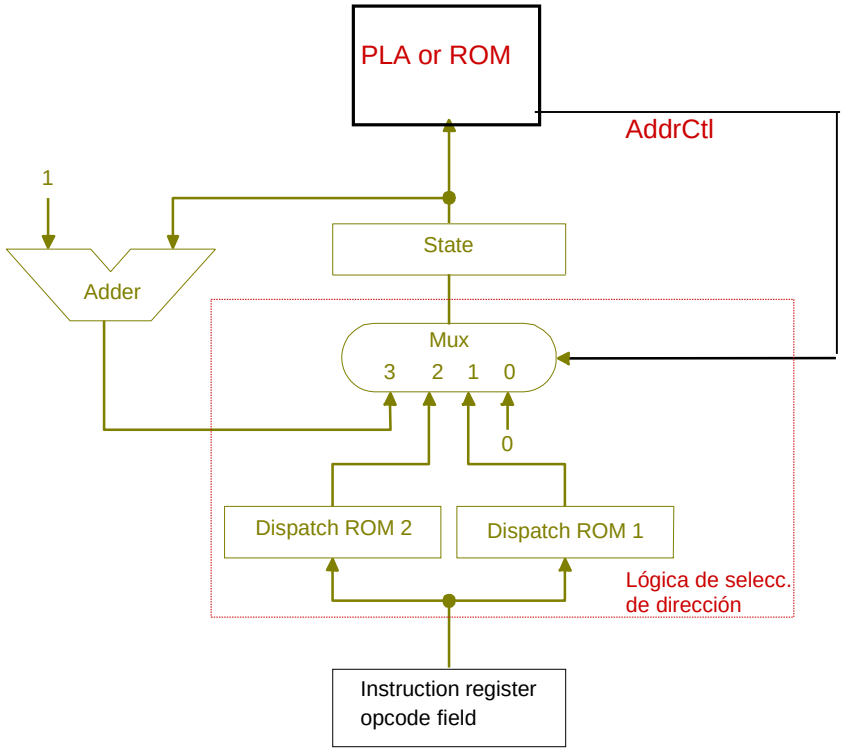
Control con un secuenciador: transiciones

Envío (*dispatch*): salto en función del código de operación.

•En nuestro diagrama de estados hay dos situaciones de envío (estados 1 y 2).

Puede utilizarse una tabla (ROM o PLA) para cada estado en el que sea preciso realizar un envío (bits de dirección: código de operación, quizá compactado).

Valor de AddrCtl	Acción
0 (00_2)	Pasar al estado 0
1 (01_2)	Envío con ROM 1
2 (10_2)	Envío con ROM 2
3 (11_2)	Usar estado incrementado



Podría usarse una única tabla para todos los envíos (bits de dirección: estado+código de operación).

Control con un secuenciador: transiciones

Tabla de envío 1

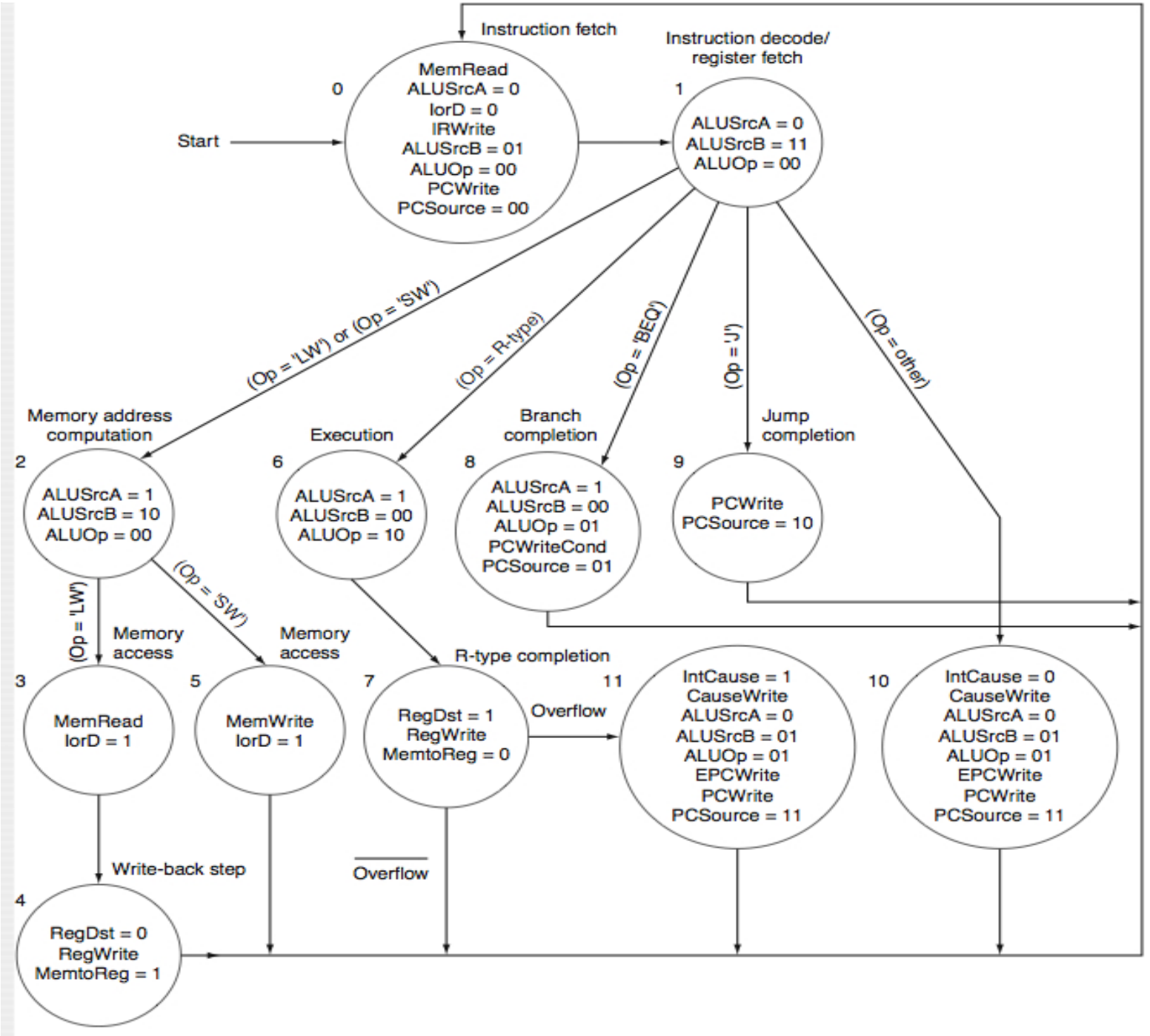
Op[5-0]	Instrucción	Valor
000000	Instrucción R	0110
000010	Instrucción J	1001
000100	Instrucción BEQ	1000
100011	Instrucción LW	0010
101011	Instrucción SW	0010

Tabla de envío 2

Op[5-0]	Instrucción	Valor
100011	Instrucción LW	0011
101011	Instrucción SW	0101

Valor de AddrCtl en cada estado

Estado	Acción del control de dirección	Valor de AddrCtl
0	Usar estado incrementado	3 (11_2)
1	Envío con ROM 1	1 (01_2)
2	Envío con ROM 2	2 (10_2)
3	Usar estado incrementado	3 (11_2)
4	Pasar al estado 0	0 (00_2)
5	Pasar al estado 0	0 (00_2)
6	Usar estado incrementado	3 (11_2)
7	Pasar al estado 0	0 (00_2)
8	Pasar al estado 0	0 (00_2)
9	Pasar al estado 0	0 (00_2)


Instruction fetch
Instruction decode/
register fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Start
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
(Op = other)
Memory address
computation
Execution
Branch
completion
Jump
completion
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
6
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Overflow
11
IntCause = 1
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
10
IntCause = 0
CauseWrite
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 01
EPCWrite
PCWrite
PCSource = 11
Write-back step
Overflow
4
RegDst = 0
RegWrite
MemtoReg = 1

Control con un secuenciador: salidas

Estado actual ⇓	Bits 17-0																	
	PCWrite	PCWriteCond	IorD	MemRead	MemWrite	IRWrite	MemtoReg	PCSource 1	PCSource 0	ALUOp 1	ALUOp 0	ALUSrcB 1	ALUSrcB 0	ALUSrcA	RegWrite	RegDst	AddrCtr 1	AddrCtl 0
0000	1		**0**	1	0	1	0	0	0	0	0	0	1	0	0	0	1	1
0001	0			**0**	0	0	0	0	0	0	0	1	1	0	0	0	0	1
0010	0			**0**	0	0	0	0	0	0	0	1	0	1	0	0	1	0
0011		**0**		**1**											**0**	0	1	1
0100						**0**	1							**0**	1	0	0	0
0101		**0**	1	0	1										**0**	0	0	0
0110									**0**	1			**0**	1	0	0	1	1
0111														**0**	1	1	0	0
1000	0	1						**0**	1	0	1		**0**	1	0	0	0	0
1001	1						**0**	1							**0**	0	0	0
Resto (estados 1010 hasta 1111)															**?**	?	?	?

8. Excepciones e interrupciones

Las excepciones y las interrupciones son situaciones que requieren un tratamiento especial por parte del procesador.

Excepción: suceso inesperado del procesador acaecido por causas internas propias de la ejecución de las instrucciones:

- Instrucción indefinida.
- Desbordamiento aritmético.
- Error de dirección.
- Llamada a servicio del sistema operativo.
- Otras causas.

Interrupción: suceso inesperado producido fuera del procesador motivado por la demanda de un dispositivo de entrada/salida que pretende comunicarse.

Excepciones e interrupciones

La terminología difiere según las arquitecturas:

•Intel 80x86: se denomina interrupción a ambos tipos de procesos.

•Familia 68K de Motorola:
Excepción: operación del procesador fuera del proceso normal de ejecución de instrucciones. Puede ser externa o interna.
Excepción interna: generada internamente por una instrucción o por alguna condición inusual asociada a la ejecución de instrucciones.
Excepción externa: interrupción debida a algún periférico, excepción por error de bus o excepción de reset.

•PowerPC:
Excepción: suceso inusual.
Interrupción: cambio en el flujo de control de la ejecución de instrucciones.

Excepciones e interrupciones

Terminología MIPS:

- Excepción: cualquier cambio inesperado en el flujo de control, sin distinguir si se debe a causas internas o externas.
- Interrupción: excepción debido a causas externas.

Tipos de excepciones:

- Excepción por eventos externos: se deben a eventos externos al procesador como son las interrupciones o los errores de bus.
- Excepciones por traducción de direcciones de memoria.
- Excepciones por errores en la ejecución de instrucciones (desbordamiento, error de alineamiento de dirección, etc).
- Problemas de integridad en los datos: errores de paridad.
- Llamadas a sistema.

Tratamiento de excepciones e interrupciones

Cada excepción concreta requiere un tratamiento individualizado, dependiendo de las causas que la produzcan.

Cuando se produce una excepción:

•El procesador realiza una serie de acciones automáticamente (*procesamiento de la excepción*).

•El procesador llama a una rutina de servicio de la excepción (*manejador de la excepción*).

• Cuando una interrupción es reconocida, se desencadena en el procesador una secuencia de acciones que culmina con la ejecución de una *rutina de manejo de interrupciones* que lleva a cabo las acciones necesarias para atender al periférico demandante.

Procesamiento de la excepción consiste en:

•Identificar la causa que ha producido la excepción.

•Guardar la dirección de la instrucción culpable (*offending*) de la excepción.

•Transferir el control al sistema operativo, que comenzará a ejecutar una **rutina de tratamiento de la excepción**.

Excepciones en MIPS

•Las rutinas de manejo de las excepciones forman parte del sistema operativo de los computadores.

Por tanto, el código de dichas rutinas reside en memoria principal.

•*Vector de excepción*: posición de memoria que contiene la dirección de comienzo del código correspondiente a una rutina de tratamiento de una excepción.

Es el punto de entrada a la rutina de tratamiento de la excepción.

•En MIPS la dirección de comienzo de la rutina de tratamiento es única:

La dirección es 80000180_{16}.

Esta dirección se carga automáticamente en el PC al producirse una excepción.

•Algunas rutinas de servicio de excepciones tienen su propio punto de entrada:

Excepciones de memoria virtual relacionadas con la TLB (*translation lookahead buffer*).

Excepciones que deben estar en zonas de memoria no ubicables en memoria caché.

Errores de paridad en caché.

Excepción de reinicio (*reset*).

Tratamiento de excepciones e interrupciones

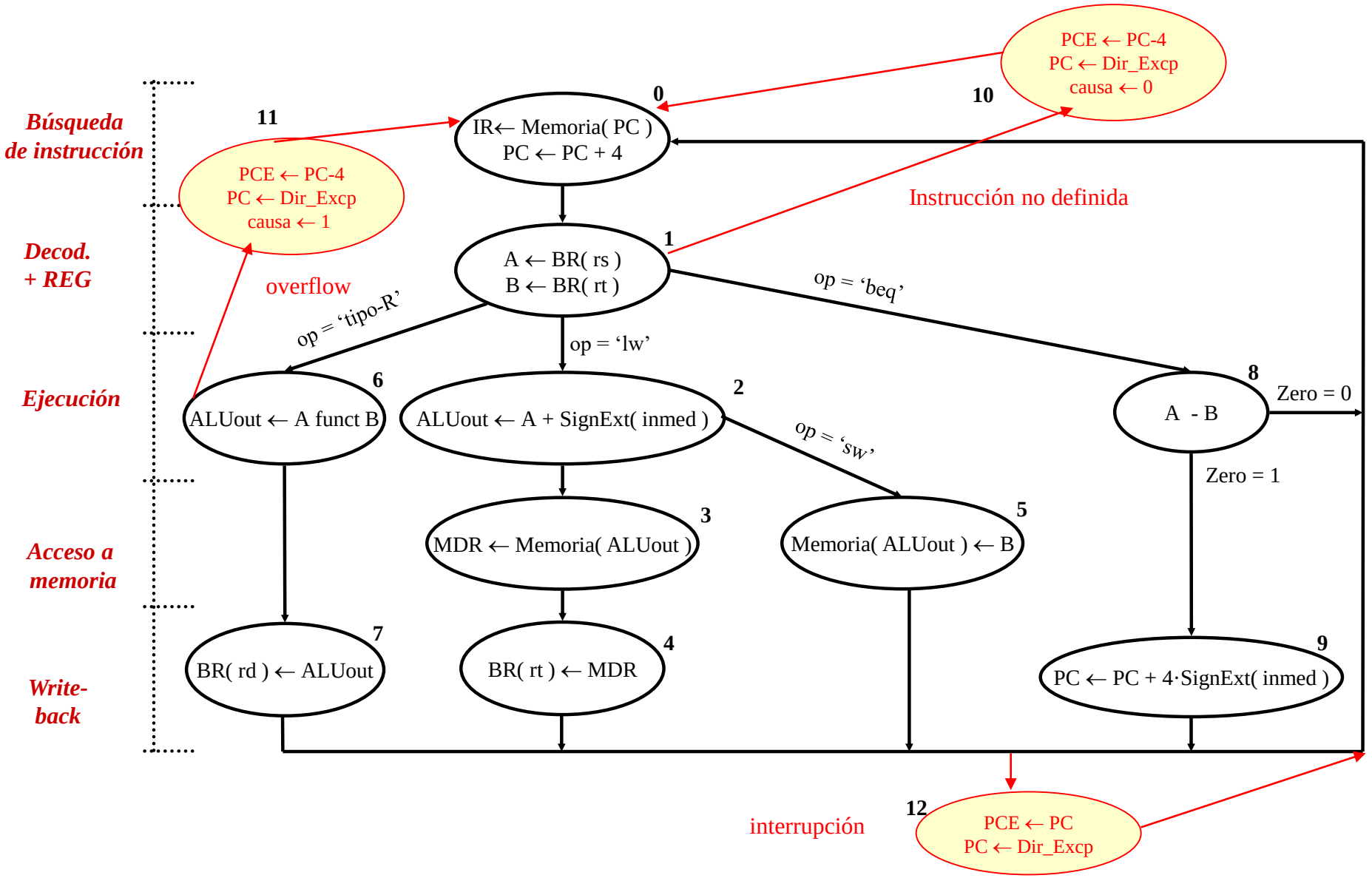
En el MIPS simplificado se añadirá la siguiente circuitería:

- Un registro de 32 bits para guardar la dirección de la instrucción causante de la excepción: **EPC**.
- Un registro de 32 bits que contiene un campo que indicará la causa que ha producido la excepción: Causa (**Cause**).
- Una señal de control para escribir en el EPC: EscrEPC (**EPCWrite**).
- Una señal de control para escribir en el registro de causa: EscrCausa (**CauseWrite**).
- Una señal de control que determinará la causa de la excepción: CausaInt (**IntCause**).
- Un multiplexor para seleccionar el valor escrito en el registro **Cause**.
- Al multiplexor para cargar el valor siguiente al PC se le añadirá una cuarta posibilidad para cargar el valor 80000180_{16} de la rutina de tratamiento de excepciones.

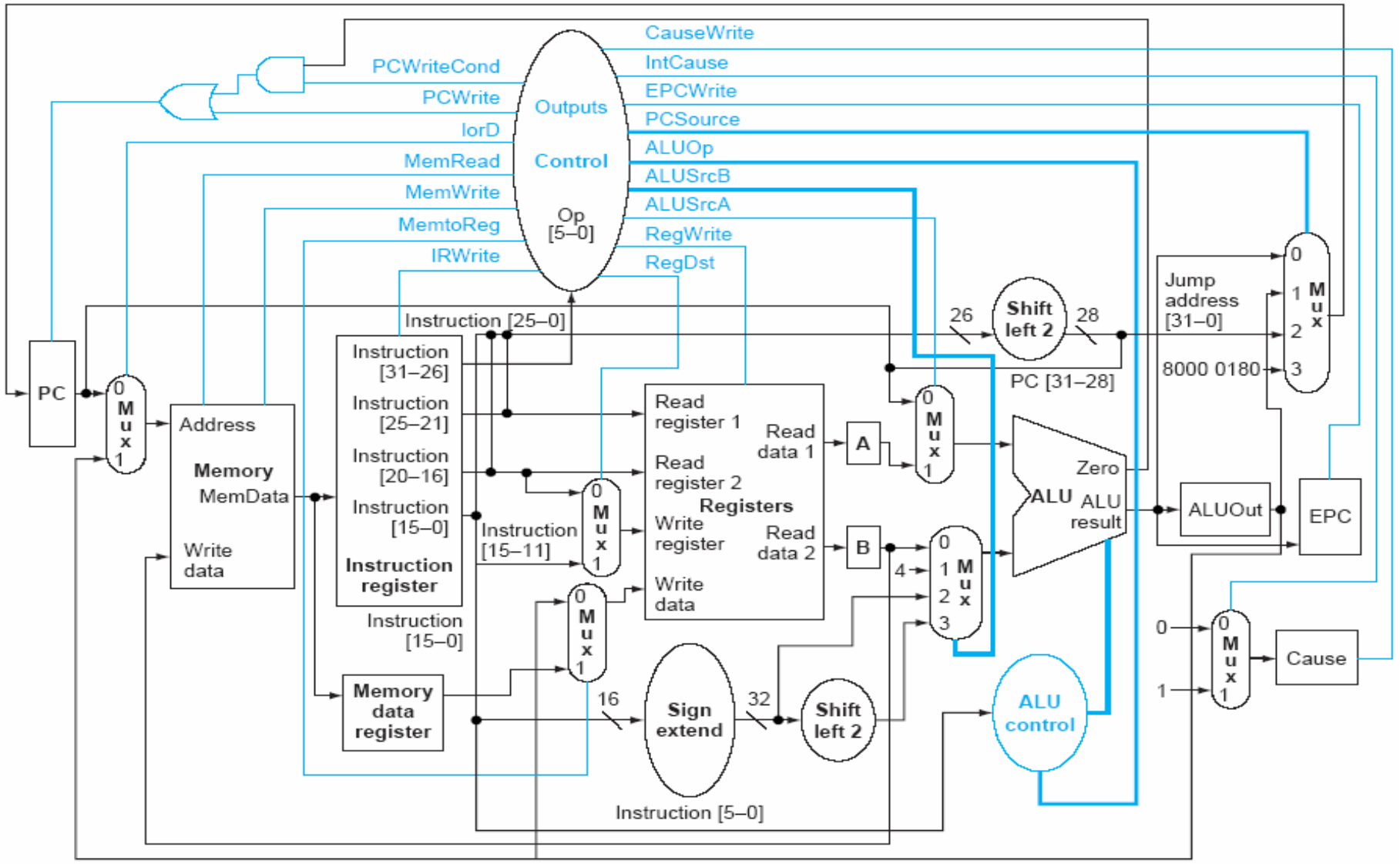
Posibles causas de excepción en nuestro camino de datos:

- Instrucción indefinida (código: 0).
- Desbordamiento aritmético (código: 1).

- **Diagrama de estados del controlador multiciclo** (repetición…)



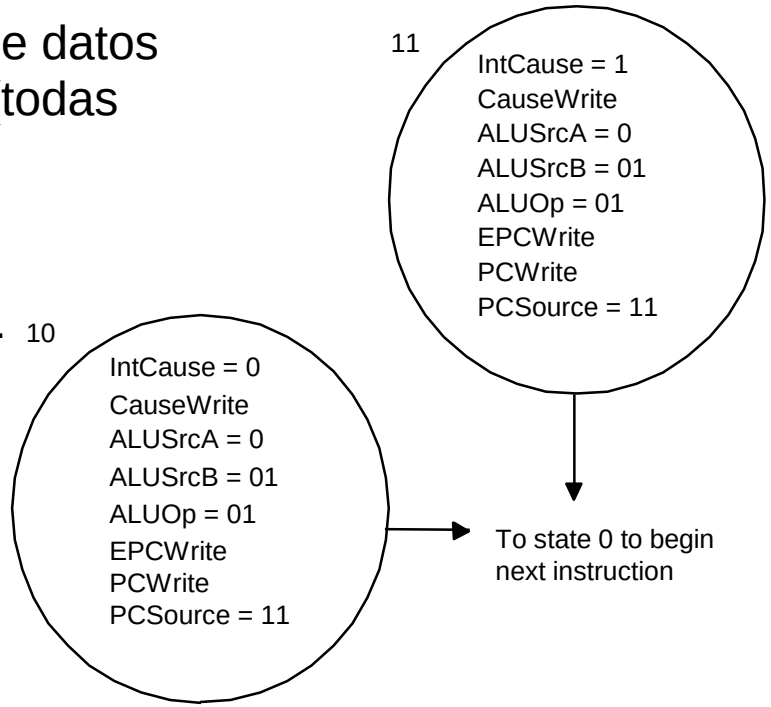
Camino de datos multiciclo con excepciones



Tratamiento de excepciones e interrupciones

Ante una excepción, en nuestro camino de datos se realizarán las siguientes operaciones (todas ellas en un único ciclo de reloj):

- Se detecta la causa de la excepción y se pone el valor adecuado en la señal **IntCause**.
- Se carga el registro **Cause**.
- Decrementa el PC (ya está apuntando a la instrucción siguiente) y guarda el valor decrementado en **EPC**.
- Escribe en el PC el valor 80000180_{16}.



Detección de las causas de excepción en nuestro camino de datos:

- Instrucción indefinida: el código de operación no está contemplado en el estado 1 del diagrama de estados.
- Desbordamiento aritmético: la señal de desbordamiento de la UAL está activa tras realizar la operación aritmética en las instrucciones R.

Tratamiento de excepciones e interrupciones

Con este esquema, en las instrucciones de tipo R se escribe el resultado en el banco de registros aun cuando se haya producido desbordamiento.

Hay otras posibles situaciones de excepción que de momento no se han Contemplado en el camino de datos:

- Error de acceso a memoria.
- Interrupción solicitada por un dispositivo de entrada/salida.

El tratamiento de excepciones en el MIPS y la circuitería asociada es más complejo que lo que se ha visto.

La detección y el tratamiento de excepciones están en el camino crítico del tiempo de la máquina e influyen en la duración del ciclo del reloj.

9. Conclusiones e implicaciones del diseño

La duración de las instrucciones se ajusta al número de etapas necesarias para su lectura y ejecución (tantos ciclos de reloj como estados le correspondan en la MEF).

- Camino crítico de la ruta de datos --> retardo de la etapa más larga.

Supongamos que las unidades funcionales tienen los siguientes retardos:

- Unidad de memoria: 2 ns.
- UAL: 2 ns.
- Acceso de cualquier tipo al banco de registros: 1 ns.
- Multiplexores, unidad de control, cables, unidad de extensión de signo, lectura o escritura de PC y demás registros de propósito específico: despreciable.

En tal caso, el ciclo de reloj duraría 2 ns.

- Habría que añadir un margen para evitar problemas debidos al sesgo del reloj.

El objetivo es alcanzar unas prestaciones elevadas haciendo sencillo el control.

El diseño del camino de datos y el control depende en gran medida de la naturaleza y características del repertorio de instrucciones del computador.

Conclusiones e implicaciones del diseño

Factores que permiten mantener sencillo el control:

- Instrucciones sencillas, homogéneas y poco potentes.
- Pocos modos de direccionamiento.
- Pocos formatos de instrucción.
- Codificación uniforme, con códigos de operación de tamaño y posición fijas.

Factores que complican el control:

- Existencia de instrucciones potentes que hagan muchas cosas y que duren muchos ciclos de reloj.
- Muchos modos de direccionamiento.
- Muchos formatos de instrucción.
- Codificación no uniforme, con códigos de operación de longitudes y posiciones variables.
- Tratamiento de excepciones.

Tendencias:

- RISC: Reduced Instruction Set Computer (computador con repertorio de instrucciones reducido). Ejemplo: MIPS.
- CISC: Complex Instruction Set Computer (computador con repertorio de instrucciones complejo). Ejemplo: Motorola 68000.