

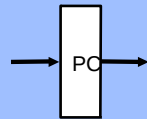
Organización del Computador

Diseño de una computadora

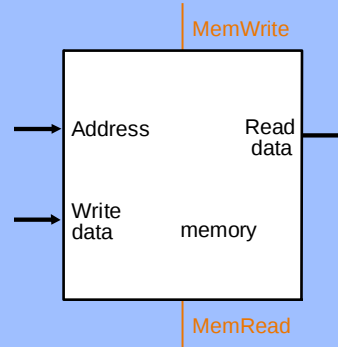
Consideraciones

- Una única ALU, una única Memoria, un único Banco de Registros.
- Problemas con el uso de los recursos??
- Dos tipos de problemas:
 - Una instrucción utiliza un mismo recurso varias veces y se pierden los valores anteriores.
 - Una instrucción utiliza un mismo recurso en la misma etapa para dos o mas cosas distintas.

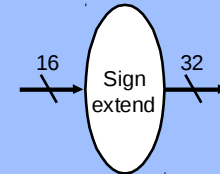
Algunos componentes disponibles



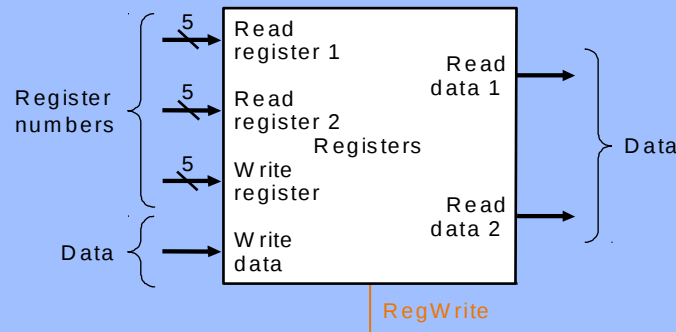
b. Program counter



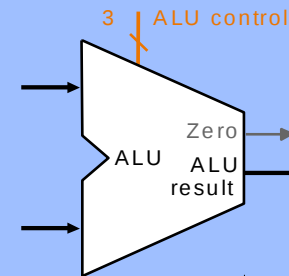
a. memory unit



b. Sign-extension unit



a. Registers



b. ALU

Diseño: Pasos necesarios

- **1^{er} Paso:** Analizar el conjunto de instrucciones para determinar los requerimientos del Camino de Datos.
- **2^o Paso:** Seleccionar los componentes.
- **3^{er} Paso:** Construir el Camino de Datos según los requerimientos.
- **4^o Paso:** Analizar la implementación de cada instrucción para determinar las señales de control necesarias.
- **5^o Paso:** Construir el Control.

Cinco Etapas de Ejecución

- 1) Fetch de Instrucción (Fetch/IF).
- 2) Decodificación (y lectura de registros) (Decode/ID).
- 3) Ejecución (o Cálculo de Dirección de memoria) (Execution/EX).
- 4) Acceso a datos en memoria (Mem).
- 5) Escritura en registros (Write Back/WB).

Formato MIPS de Instrucción

Son todas de 32 bits. Tres formatos:

Tipo R

- Aritméticas



Tipo I

- Transferencia, salto
- Operaciones con operando inmediato



Tipo J

- Saltos



Formato MIPS

• ADD y SUB

- $(R[rd] = R[rs] \text{ op } R[rt])$
- `addu rd,rs,rt`
- `subu rd,rs,rt`

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

• LOAD and STORE

- `lw rt,rs,inm16`
 - $R[rt] = \text{Mem}[R[rs] + \text{sign_ext}(\text{Inm16})];$
- `sw rt,rs,inm16`
 - $\text{Mem}[R[rs] + \text{sign_ext}(\text{Inm16})] = R[rt];$

op	rs	rt	Address/immediate
----	----	----	-------------------

• BRANCH

- `beq rs,rt,inm16`
- if $(R[rs] == R[rt])$ then
 $\text{PC} = \text{PC} + (\text{sign_ext}(\text{Inm16}) * 4)$

op	rs	rt	Address/immediate
----	----	----	-------------------

• JUMP

- `J target`
- $\text{PC} \langle 31:2 \rangle = \text{PC} \langle 31:28 \rangle, (\text{target} \langle 25:0 \rangle \ll 2)$

op	Target Address
----	----------------

RTL

- Cada instrucción está formada por un conjunto de microoperaciones.
- RTL (Register Transfer Language): se utiliza para determinar la secuencia exacta de microoperaciones.
- Ejemplo (Fetch en Marie):
 - t1: $MAR \leftarrow (PC)$
 - t2: $MBR \leftarrow mem[MAR], PC \leftarrow (PC) + 1$
 - t3: $IR \leftarrow (MBR)$

1º Paso: Tipo R (add, sub..)

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

- $R[rd] = R[rs] \text{ op } R[rt] \text{ y } PC = PC + 4$

- RTL

T1: $IR \leftarrow \text{mem}[PC]; PC \leftarrow PC + 4$

T2: $A \leftarrow R[rs]; B \leftarrow R[rt]$

T3: $ALUOut \leftarrow A \text{ op } B$

T4: $R[rd] \leftarrow ALUOut$

Branch

op	rs	rt	Address/immediate
----	----	----	-------------------

BEQ if ($R[rs] == R[rt]$) then $PC = PC + (\text{sign_ext}(\text{Imm16}) * 4)$

● RTL

T1: $IR \leftarrow \text{mem}[PC];$
 $PC \leftarrow PC + 4$

T2: $A \leftarrow R[rs]$ (Se guardan para el ciclo siguiente)
 $B \leftarrow R[rt]$
 $ALUOut \leftarrow PC + \text{signextend}(\text{imm16}) \ll 2$ (calcula la dir. Del salto)

T3: Comparar A y B
 $PC \leftarrow ALUOut$ Si el flag Zero esta activo

LOAD



- LOAD $R[rt] = \text{Mem}[R[rs] + \text{sign_ext}(\text{Imm16})];$

- RTL

T1: $IR \leftarrow \text{mem}[PC]$
 $PC \leftarrow PC + 4$

T2: $A \leftarrow R[rs]$
 $B \leftarrow R[rt]$ (B no se usa)

T3: $ALUOut \leftarrow A + \text{signextend}(\text{imm16})$ (Calcula la dir.)

T4: $MBR \leftarrow \text{Mem}[ALUOut]$

T5: $R[rt] \leftarrow MDR$

STORE



- STORE $\text{Mem}[R[\text{rs}] + \text{sign_ext}(\text{Imm16})] \leftarrow R[\text{rt}];$

- RTL

T1: $\text{IR} \leftarrow \text{mem}[\text{PC}]$

$\text{PC} \leftarrow \text{PC} + 4$

T2: $A \leftarrow R[\text{rs}]$

$B \leftarrow R[\text{rt}]$ (valor a escribir)

T3: $\text{ALUOut} \leftarrow A + \text{signextend}(\text{imm16})$

T4: $\text{Mem}[\text{ALUOut}] \leftarrow B$

JUMP



- Jump: $PC\langle 31:2 \rangle \leftarrow PC\langle 31:28 \rangle, (\text{target} \langle 25:0 \rangle \ll 2)$
 - Calcula la dirección concatenando los 26 bits del operando
- RTL
 - T1: $IR \leftarrow \text{mem}[PC]$
 $PC \leftarrow PC + 4$
 - T2: NADA!
 - T3 $PC\langle 31:2 \rangle \leftarrow PC\langle 31:28 \rangle, (IR\langle 25:0 \rangle \ll 2)$

Resumen de las etapas

Cycle	Instruction type	action
IF	all	$IR \leftarrow \text{Memory}[PC]$ $PC \leftarrow PC + 4$
ID	all	$A \leftarrow \text{Reg}[rs]$ $B \leftarrow \text{Reg}[rt]$ $ALUOut \leftarrow PC + (\text{imm16} \ll 2)$
EX	R-type Load/Store Branch Jump	$ALUOut \leftarrow A \text{ op } B$ $ALUOut \leftarrow A + \text{sign-extend}(\text{imm16})$ if $(A == B)$ then $PC \leftarrow ALUOut$ $PC \leftarrow PC[31:28] \parallel (IR[25:0] \ll 2)$
MEM	Load Store	$MDR \leftarrow \text{Memory}[ALUOut]$ $\text{Memory}[ALUOut] \leftarrow B$
WB	R-type Load	$\text{Reg}[rd] \leftarrow ALUOut$ $\text{Reg}[rt] \leftarrow MDR$

Número de Ciclos

- Branch y Jump: 3 ciclos (IF,ID,EX)
- Las tipo R: 4 ciclos (IF,ID,EX,WB)
- STORE: 4 ciclos (IF,ID,EX,MEM)
- LOAD: 5 (IF,ID,EX,MEM,WB)

1- Fetch (IF)

- RTL

$IR \leftarrow \text{Memory}[PC]$

$PC \leftarrow PC + 4$

2 - Decode (ID)

- RTL

- Opción lectura de registros

$A \leftarrow \text{reg}[\text{IR}[25:21]]$

$B \leftarrow \text{reg}[\text{IR}[20:16]]$

- Opción cálculo de salto

$\text{ALUOut} \leftarrow \text{PC} + \text{sign-extend}(\text{IR}[15:0]) \ll 2$

3 - Execute (Ex)

- RTL

- Opción aritmética/lógica
 - $ALUOut \leftarrow A \text{ op } B, \text{ o}$
 - $ALUOut \leftarrow A \text{ op sign-extend}(IR[15:0])$
- Cálculo de dirección (Load o Store)
 - $ALUOut \leftarrow A + \text{sign-extend}(IR[15:0])$
- Salto condicional
 - Si $A = B$, $PC \leftarrow ALUOut$
- Jump
 - $PC[31:28] \leftarrow PC \parallel IR[25:0] \ll 2$

4 – Memoria (Mem)

- Sólo para Load o Store
- RTL
 - Opción Lectura
 - $MBR \leftarrow \text{Memory}[\text{ALUOut}]$
 - Opción Grabación
 - $\text{Memory}[\text{ALUOut}] \leftarrow B$

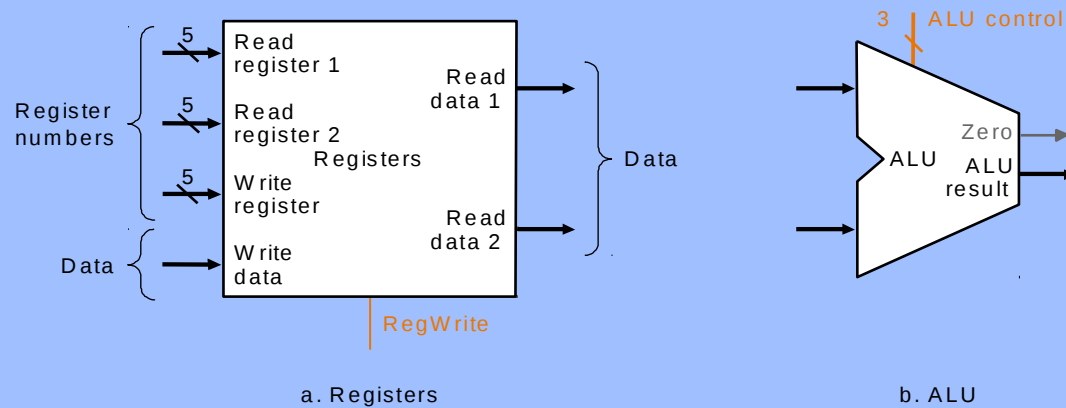
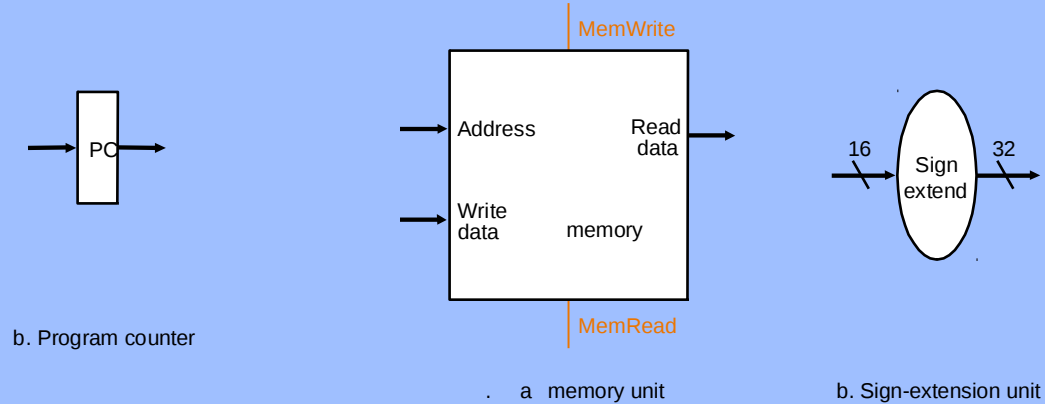
5 - Escritura

- RTL
 - Opción instrucción tipo R o aritmética inmediata
 - $\text{reg[IR[15:11]]} \leftarrow \text{ALUOut}$
 - Opción Load (celda de memoria leída en el ciclo anterior)
 - $\text{reg[IR[20:16]]} \leftarrow \text{MBR}$

1^{er} Paso: Requerimientos del Conjunto de Instrucciones

- Memoria
 - Para Instrucciones y Datos
- Registros (32x32)
 - Leer rs, Leer rt
 - Escribir rt o rd
- PC, MBR
- A, B para datos intermedios, ALUOut (retener salida ALU)
- Extensor de signo (16 a 32)
- Sumar y Restar registros y/o valores inmediatos
- Operaciones lógicas (and/or) registros y/o valores inmediatos
- Sumar 4 al PC o 4+inmediato extendido *4

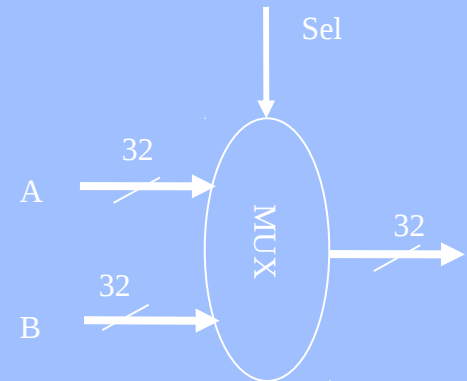
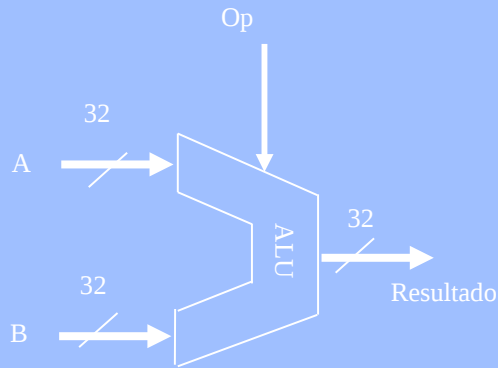
2º Paso: Componentes del Camino de Datos



2º Paso: Componentes del Camino de Datos

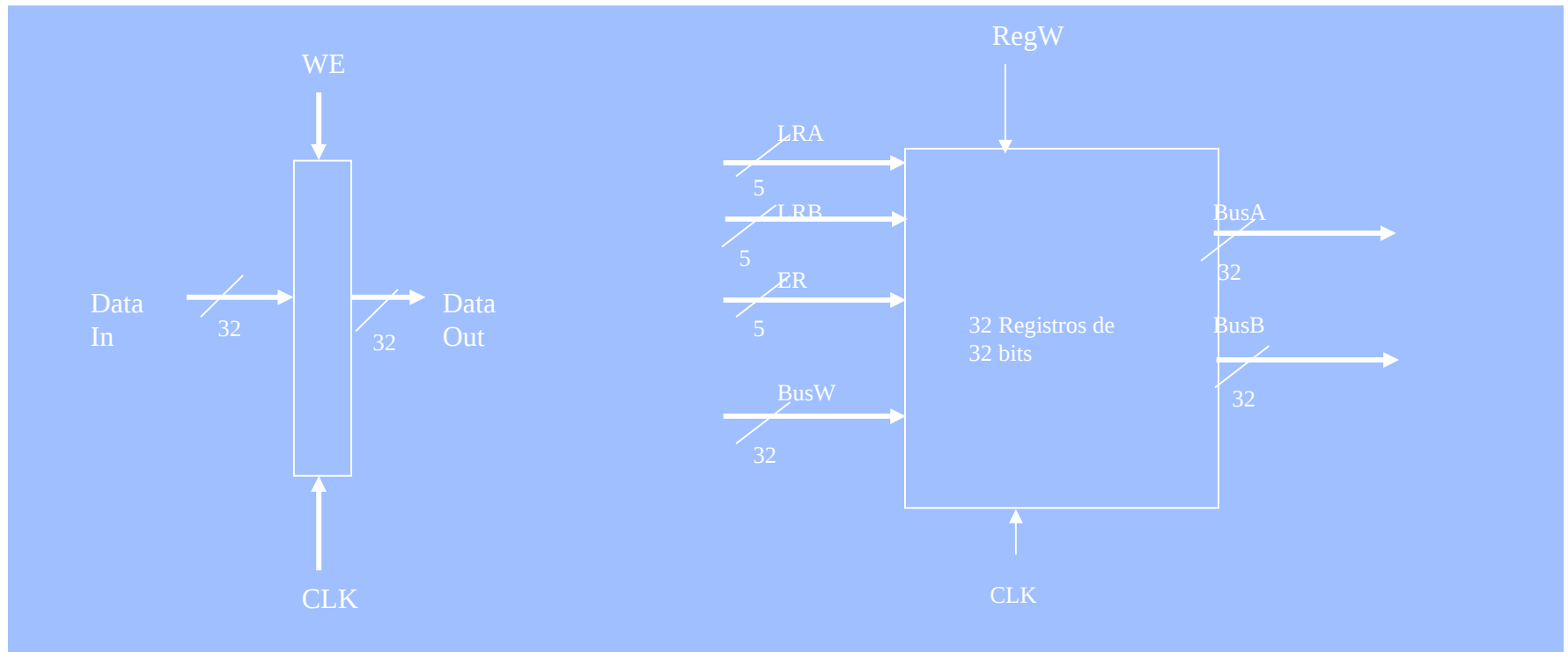
- Elementos Combinacionales

- ALU y Multiplexor



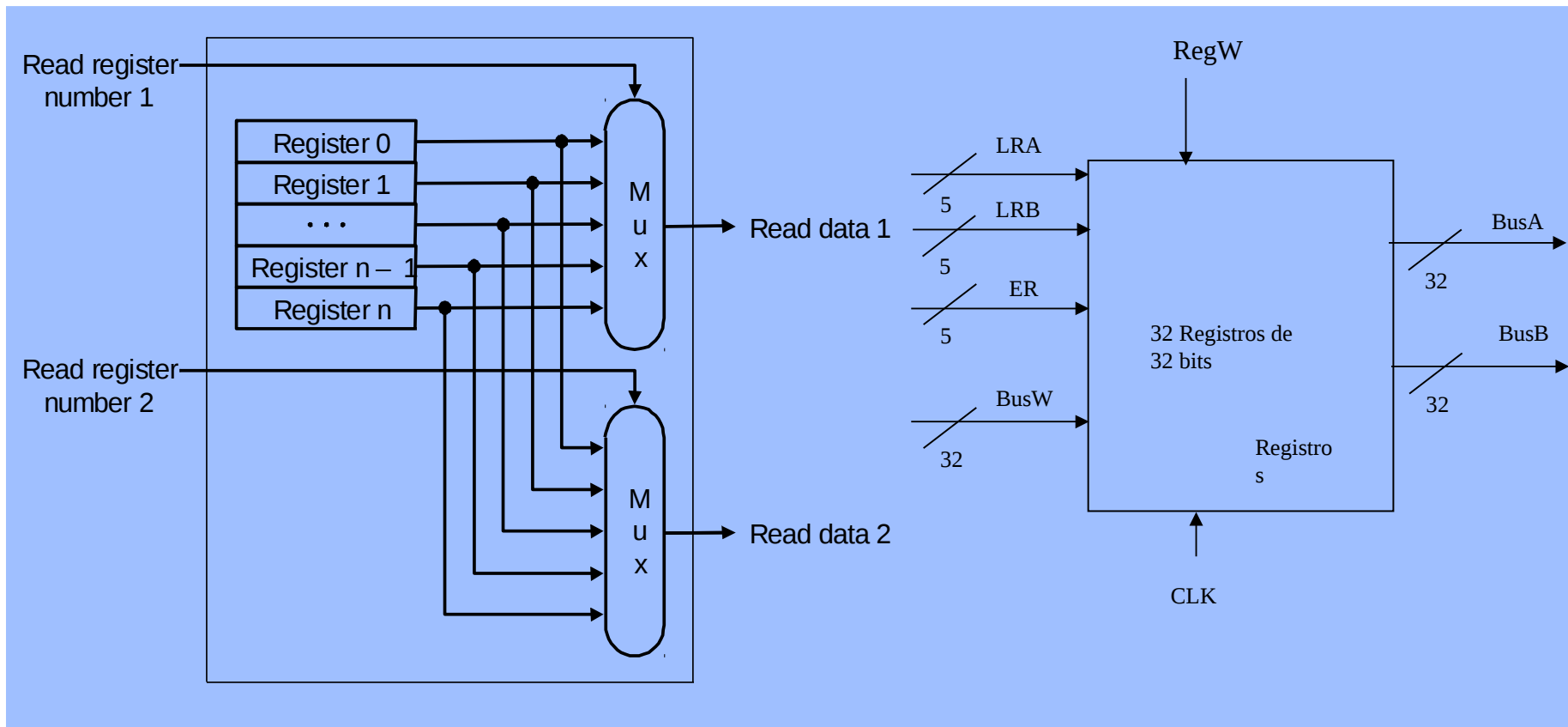
2º Paso: Componentes del Camino de Datos

- Elementos de Almacenamiento: Banco de Registros



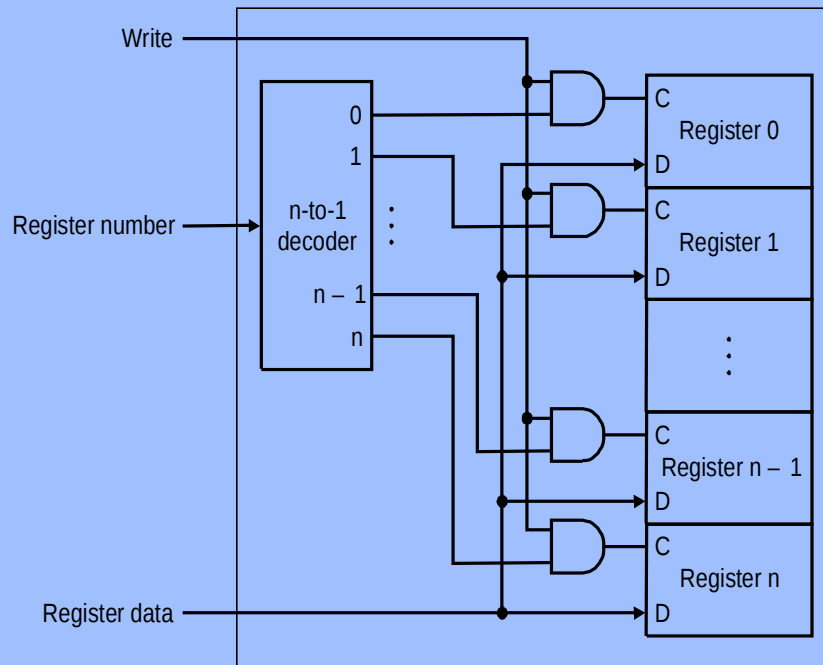
2º Paso: Componentes del Camino de Datos

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



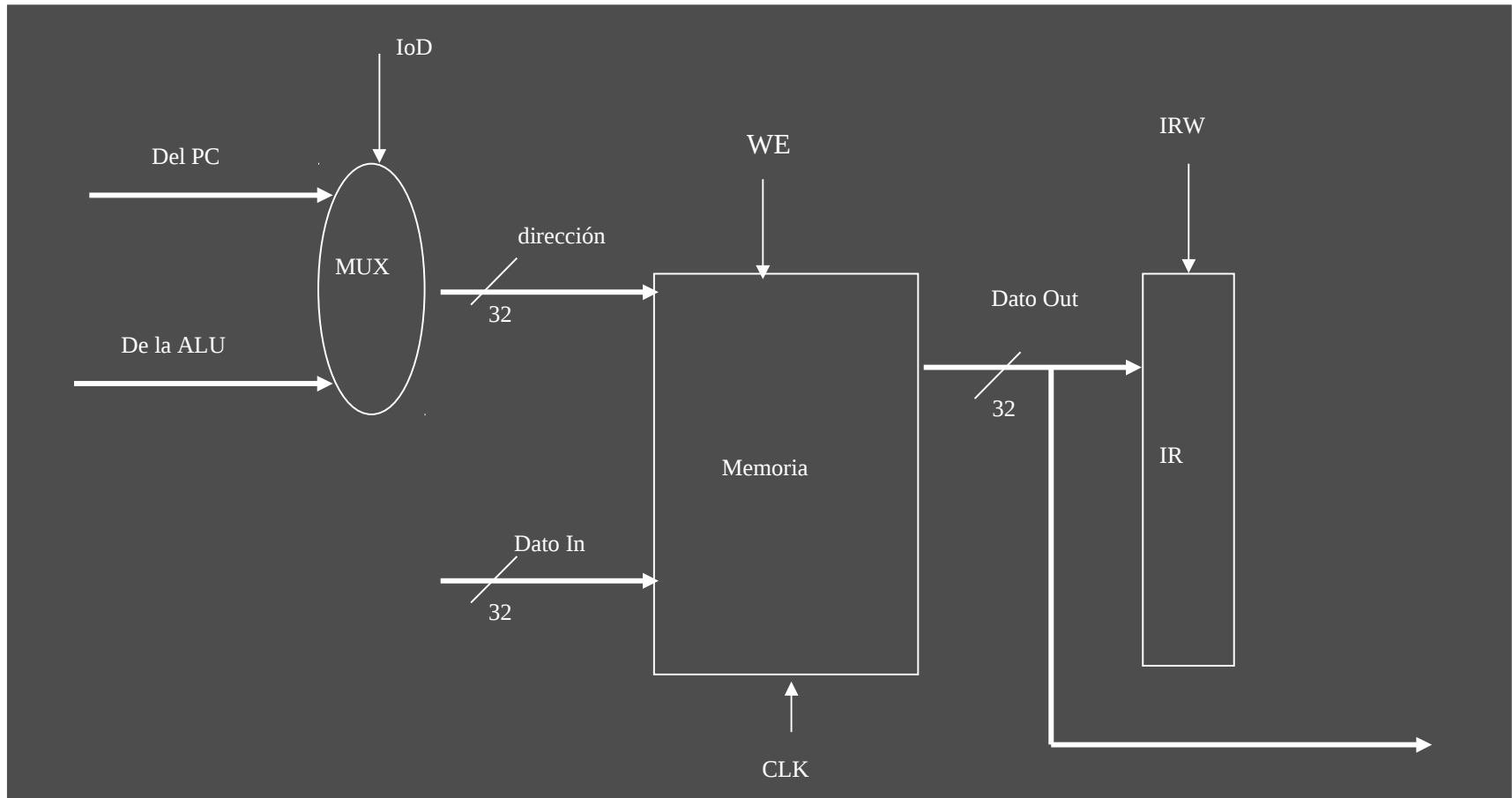
2º Paso: Componentes del Camino de Datos

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



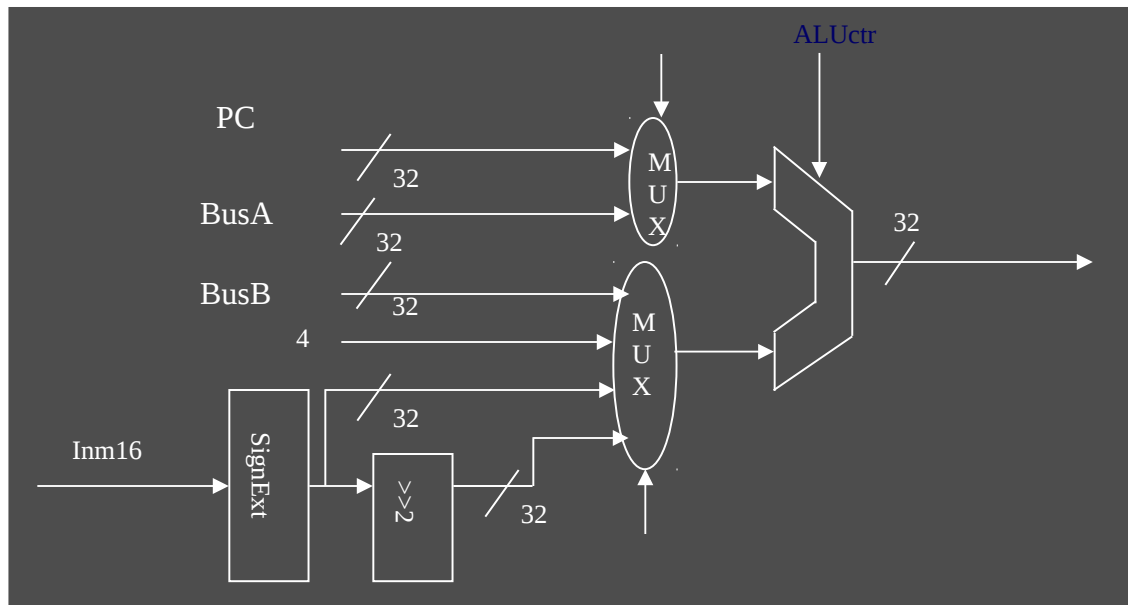
2º Paso: Componentes del Camino de Datos

- Una sola memoria para instrucciones y datos

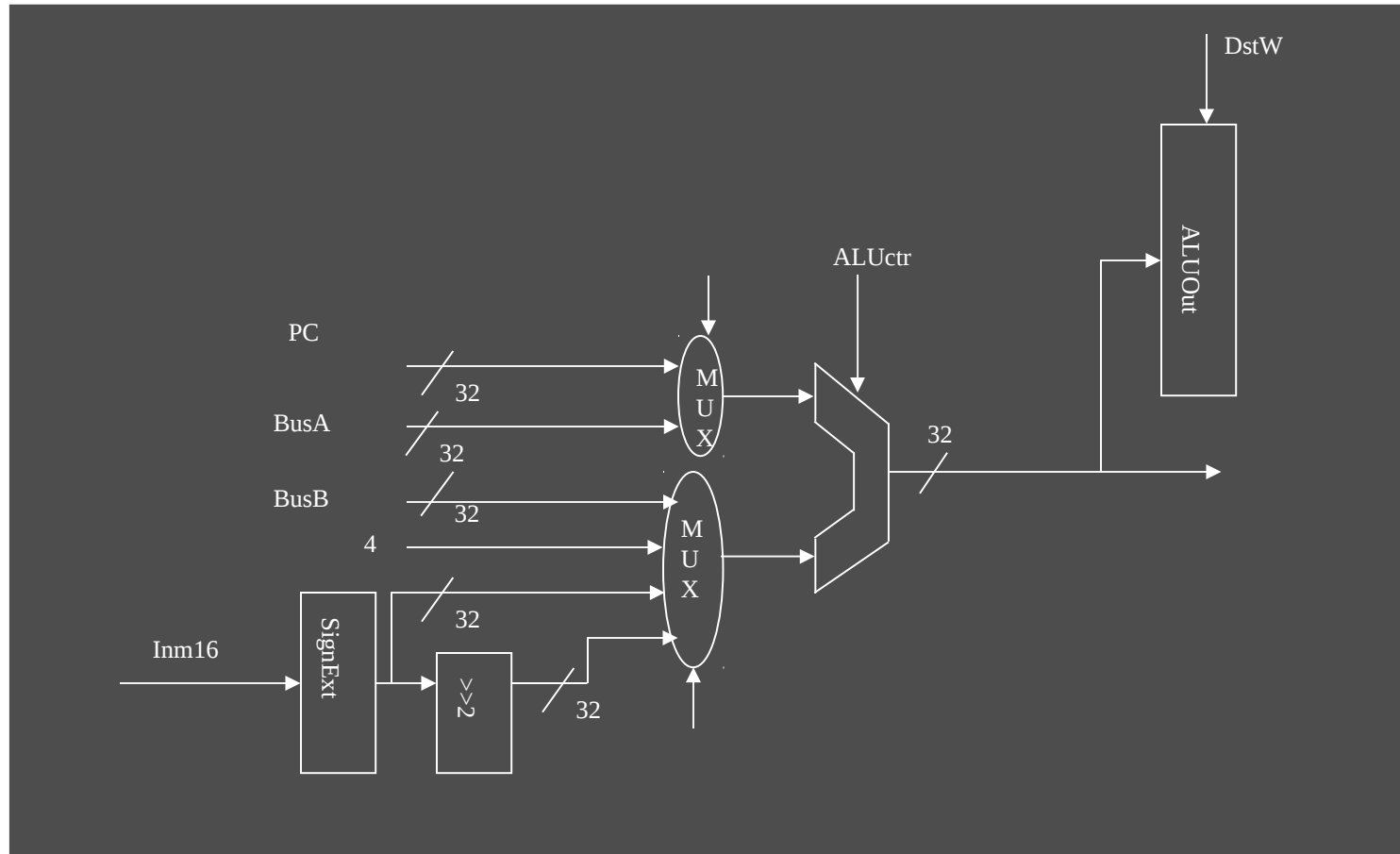


3^{er} Paso: Reuso de Unidades Funcionales: ALU única

- ALU debe realizar
 - operaciones sobre registros
 - base + desplazamiento para loads y stores
 - dirección destino de salto: $\text{registro} + \text{signo_ext}(\text{inmm16}) * 4$
 - $\text{PC} = \text{PC} + 4$

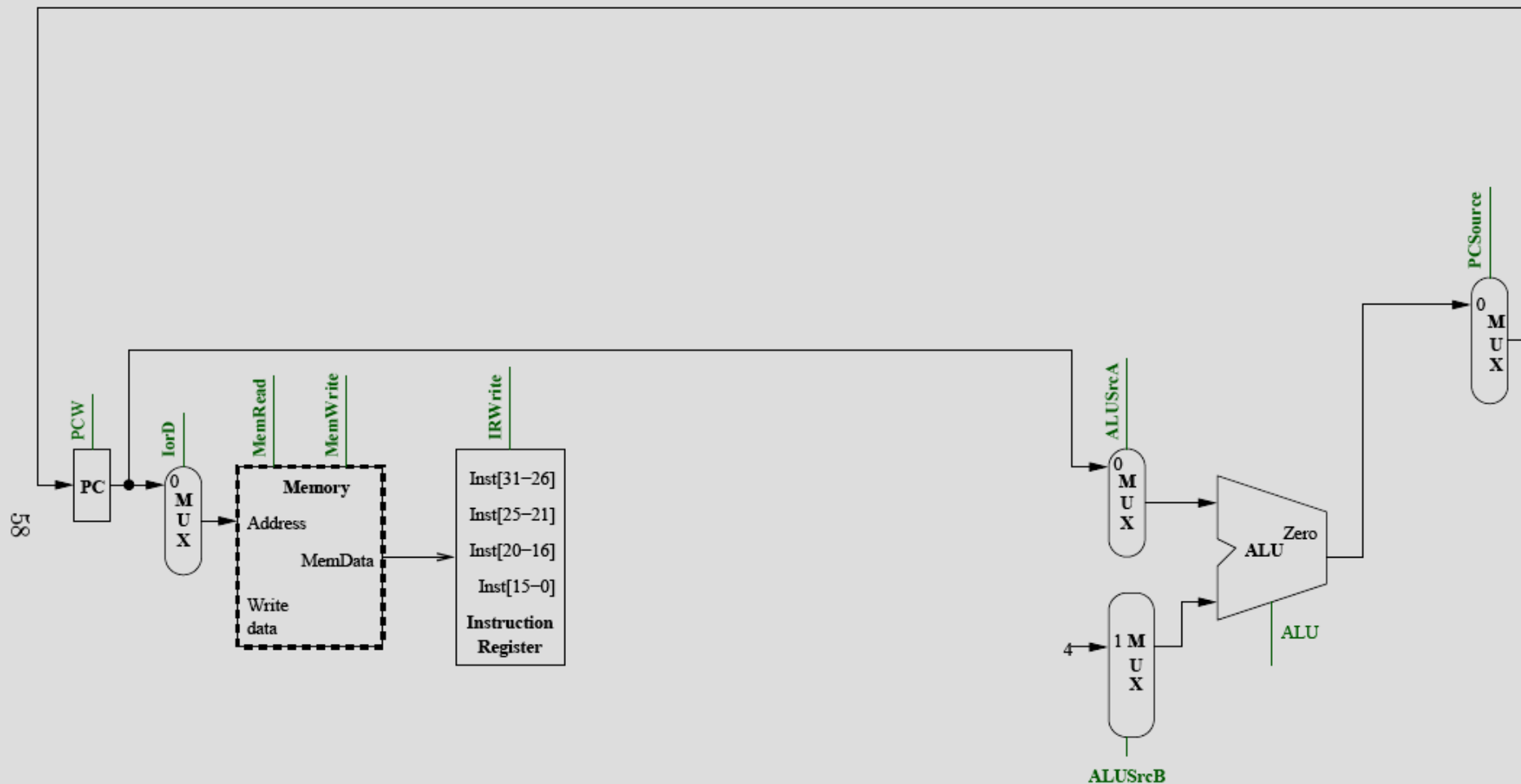


3^{er} Paso: Registro ALUout

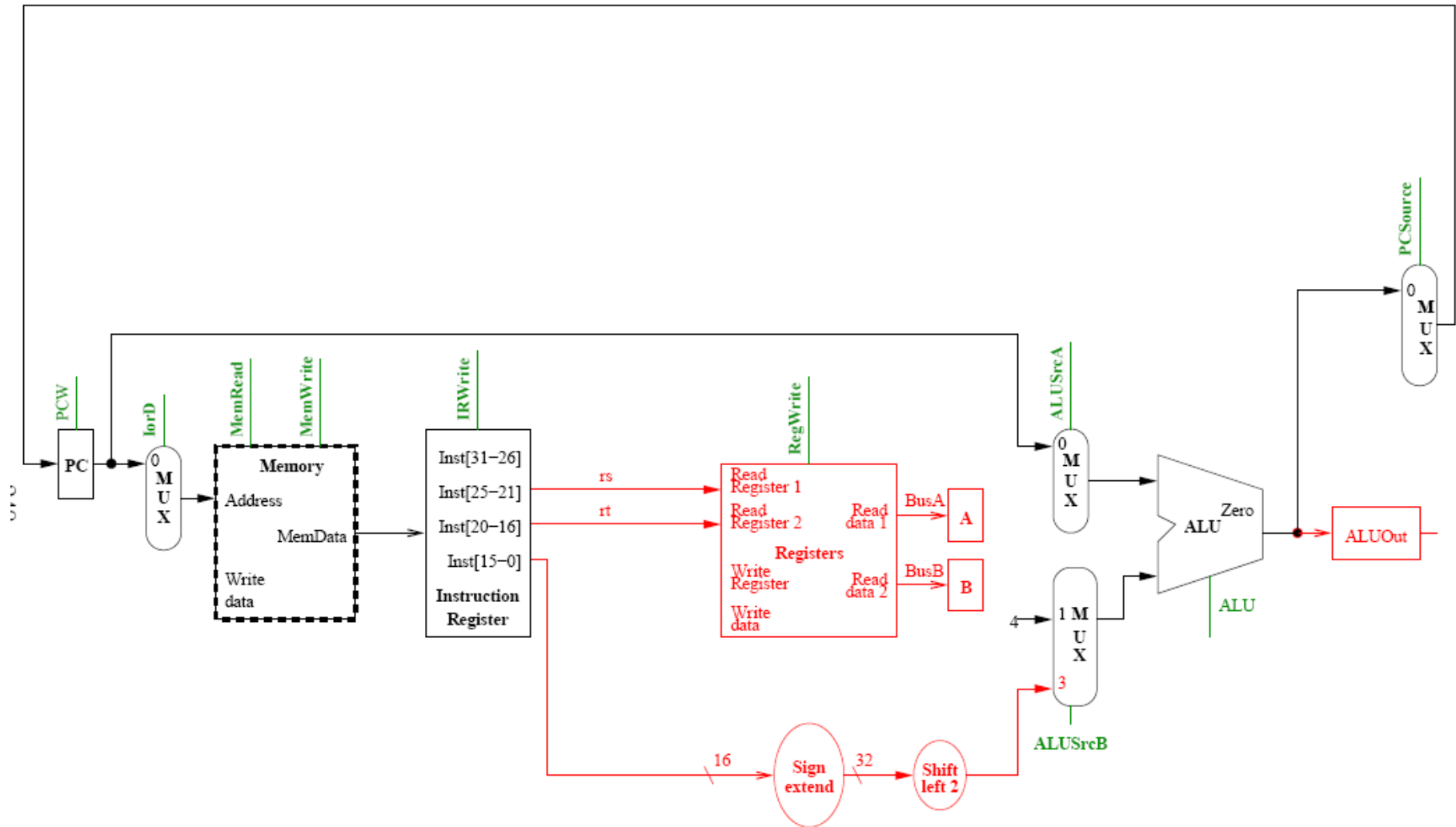


3^{er} Paso: Fetch - IF

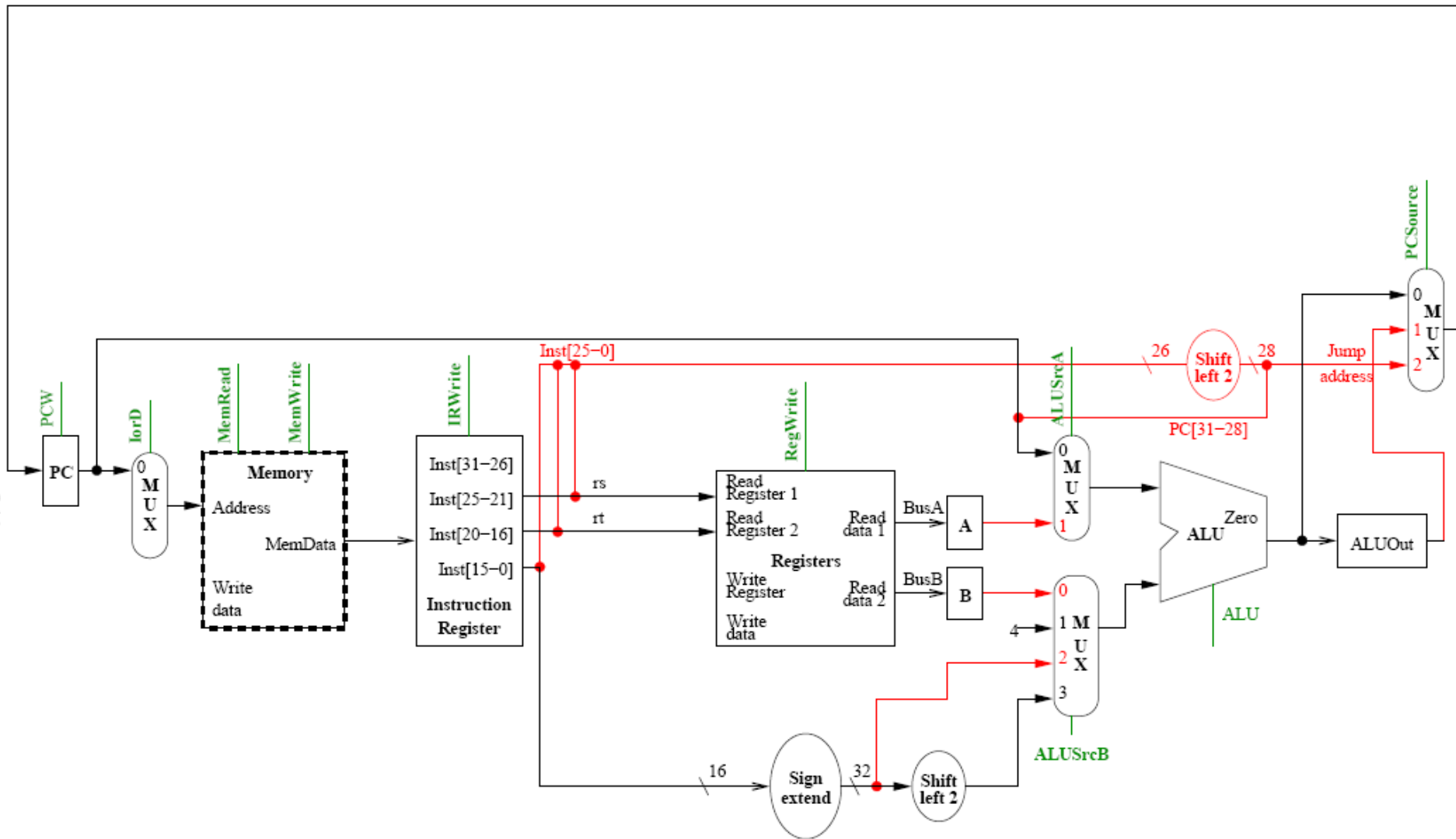
- Mem[PC]; PC \leftarrow PC+4 (código secuencial)



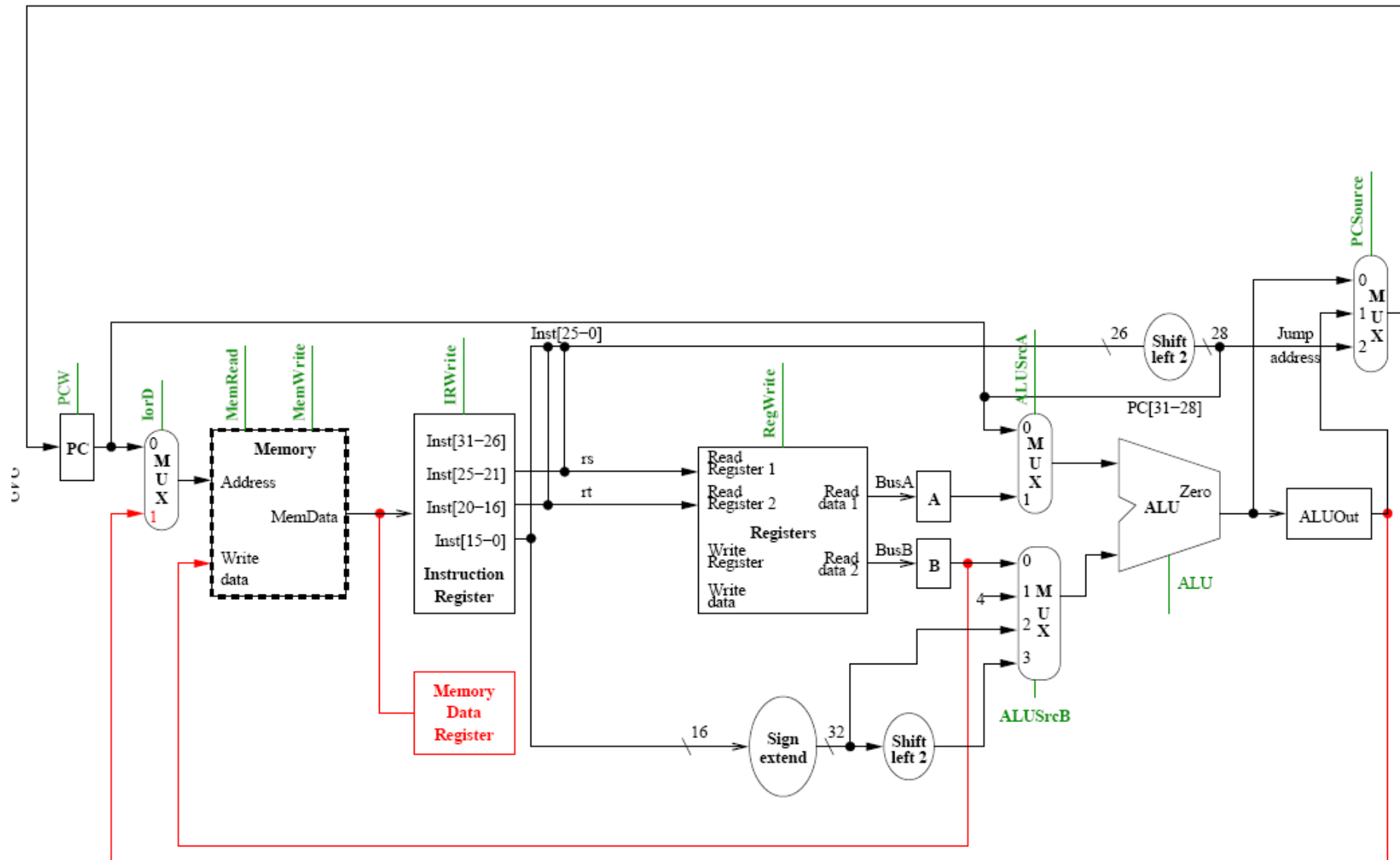
3^{er} Paso: Decode - ID



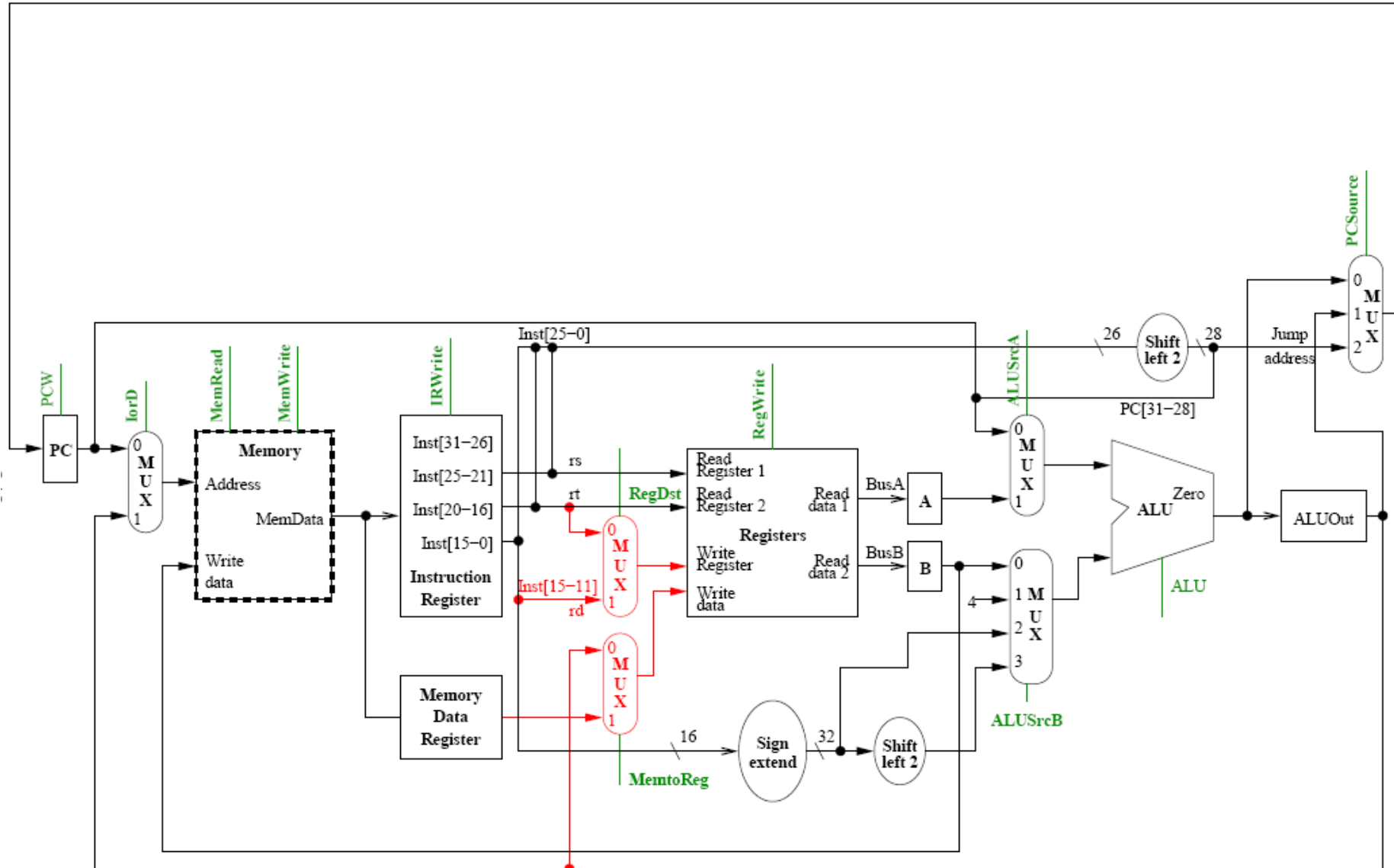
3^{er} Paso: DataPath - Ex



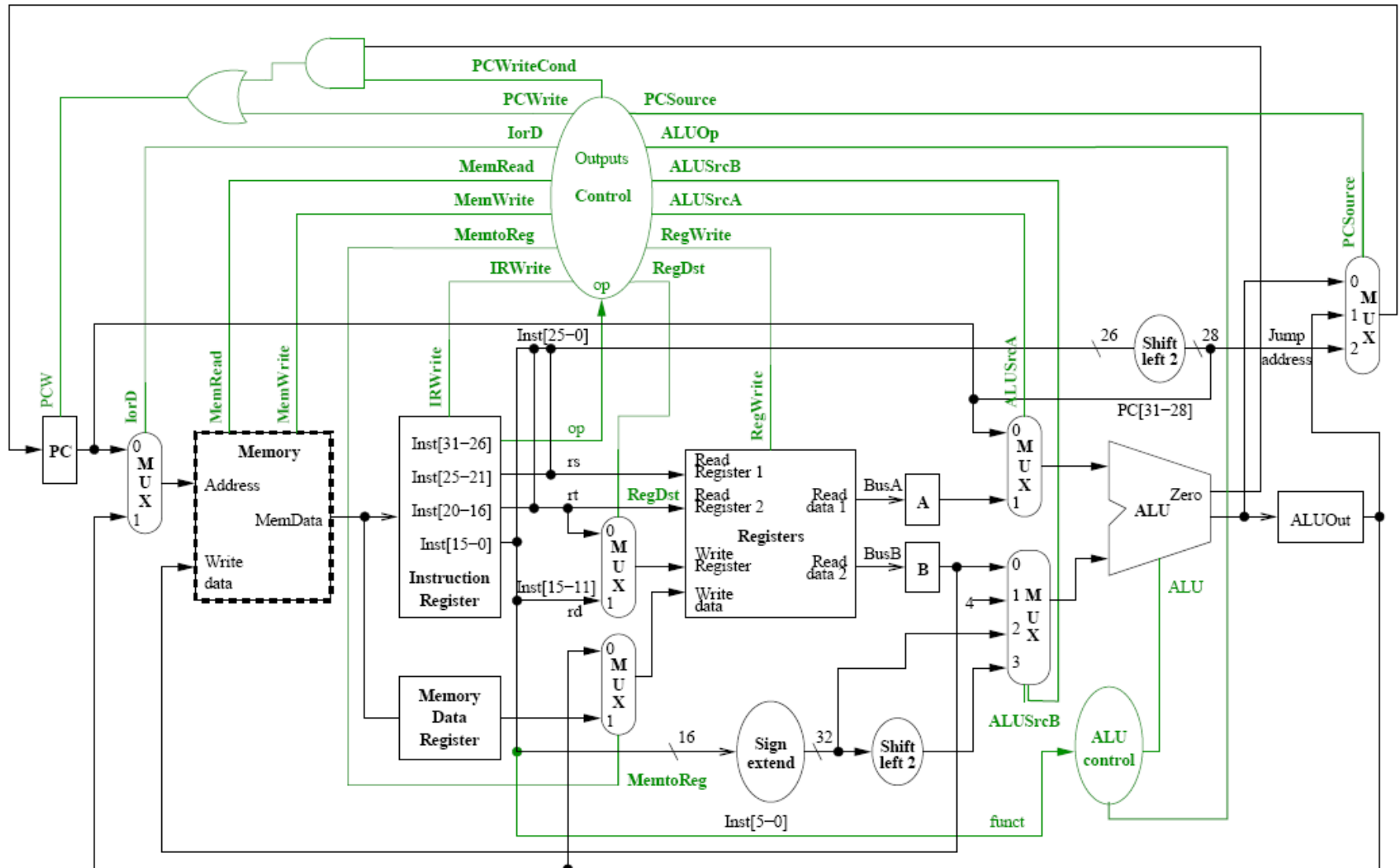
3^{er} Paso: DataPath - MEM



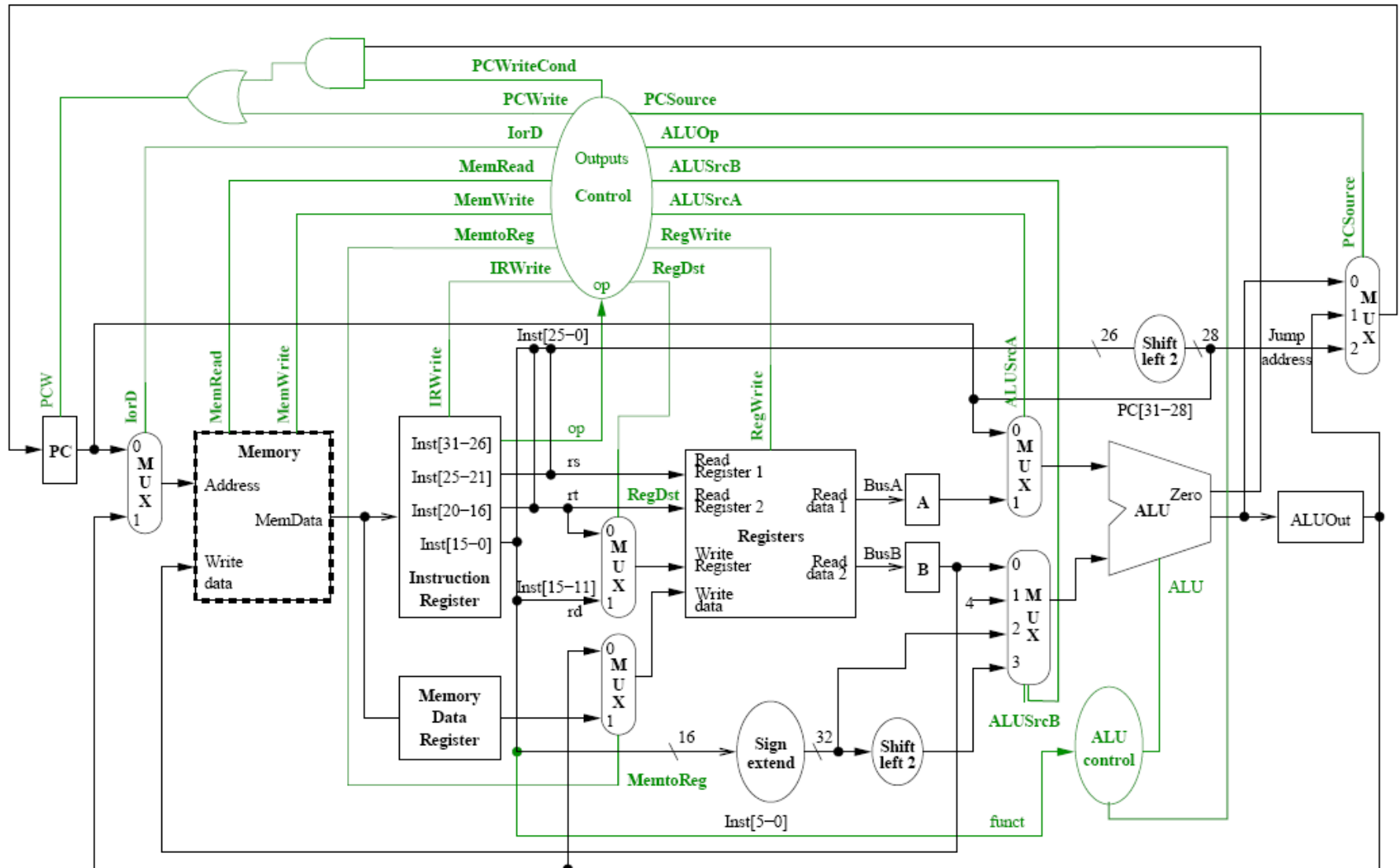
3^{er} Paso: DataPath - WB



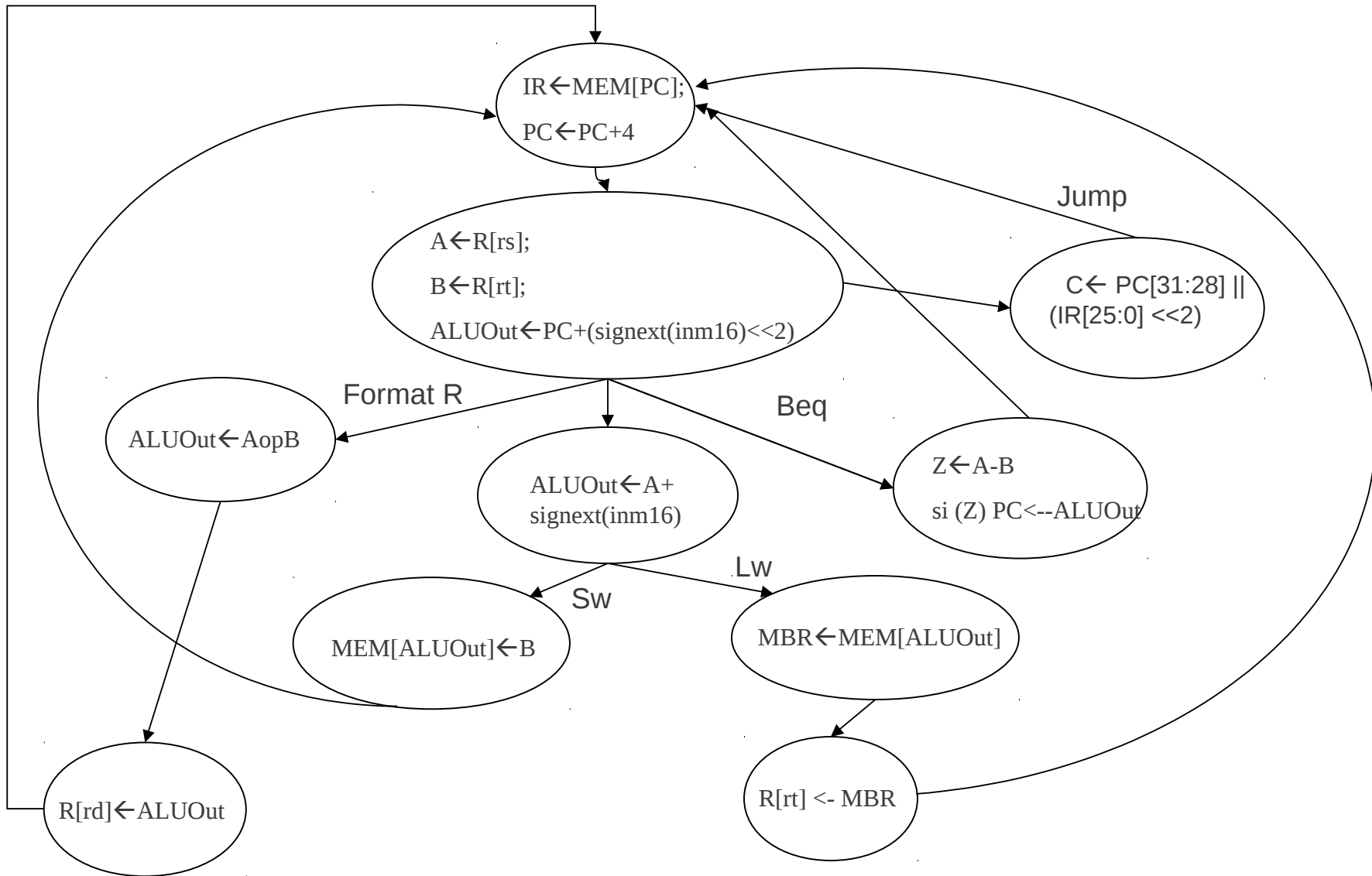
Señales de control



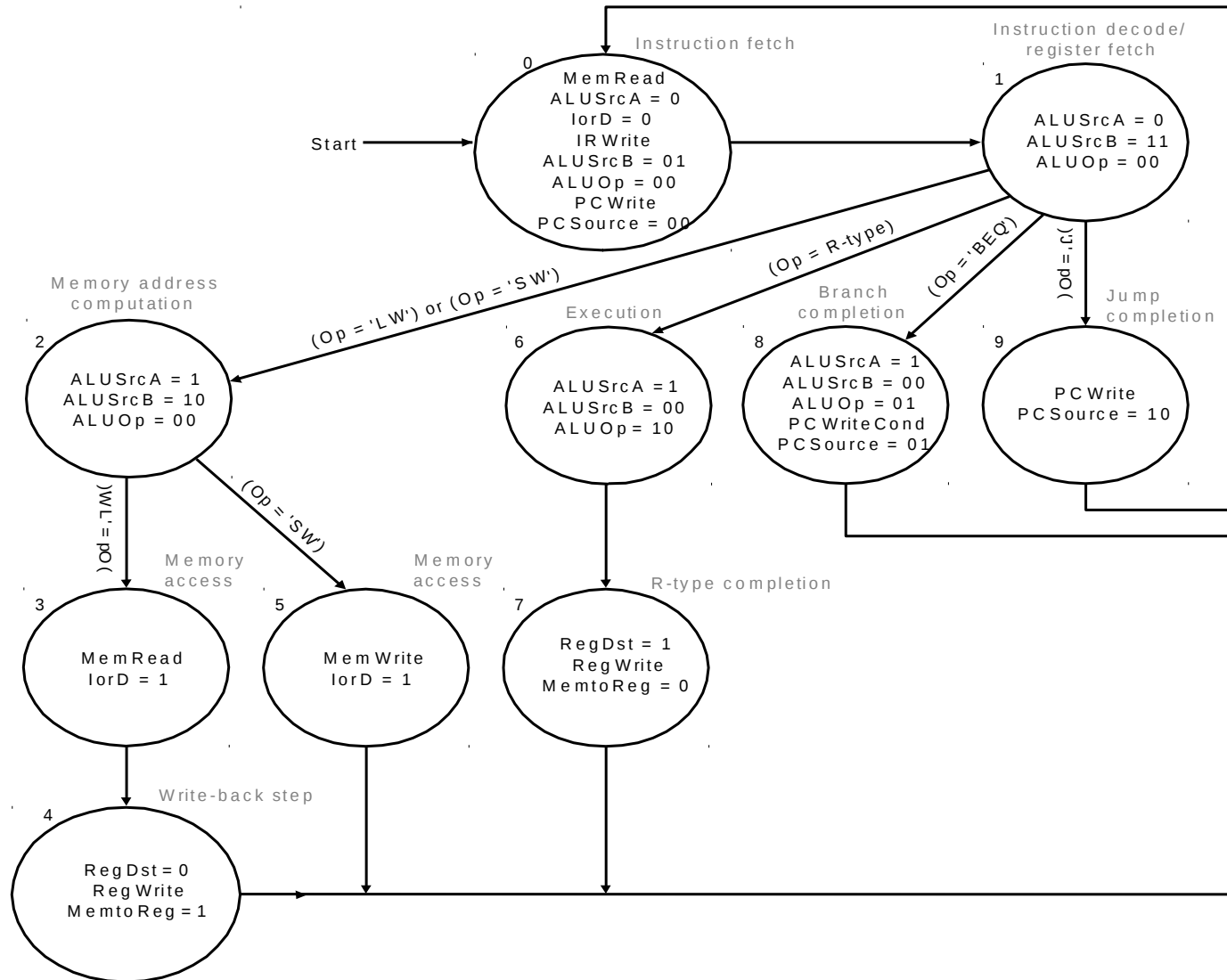
Señales de control



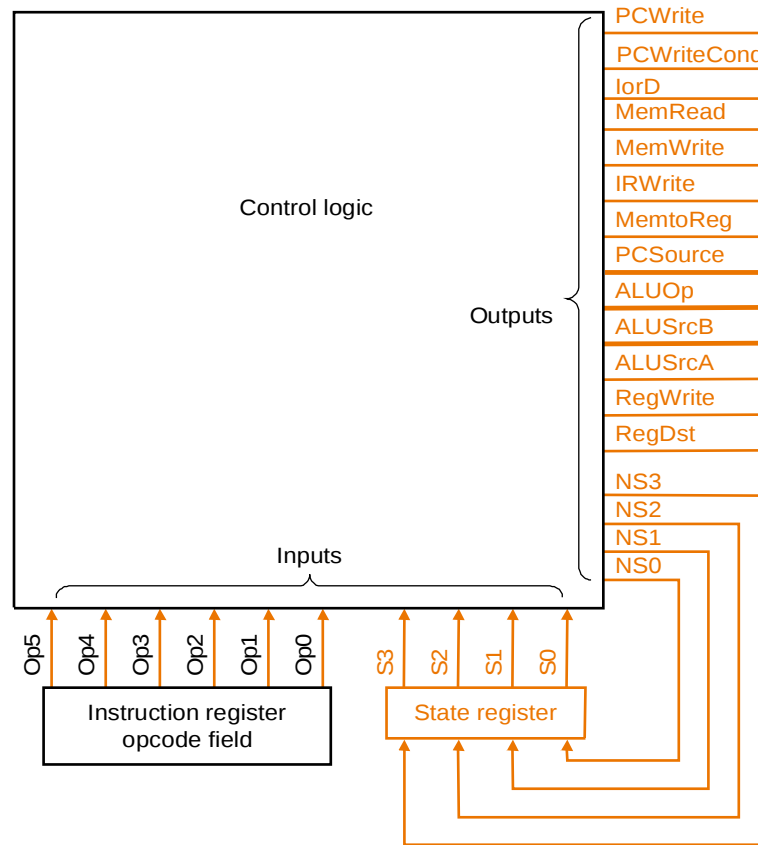
Grafo de Estados



Control de Señales



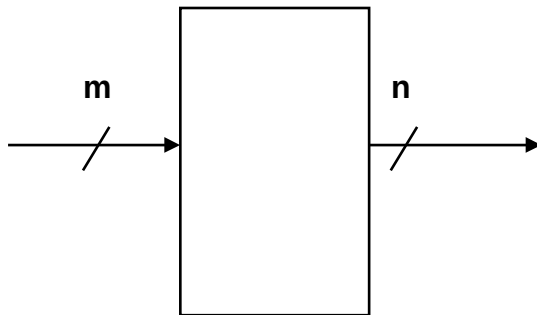
Máquina de Estados Finitos



- Hardwired → Circuito Combinacional (Tabla de verdad!)

Implementación con ROM

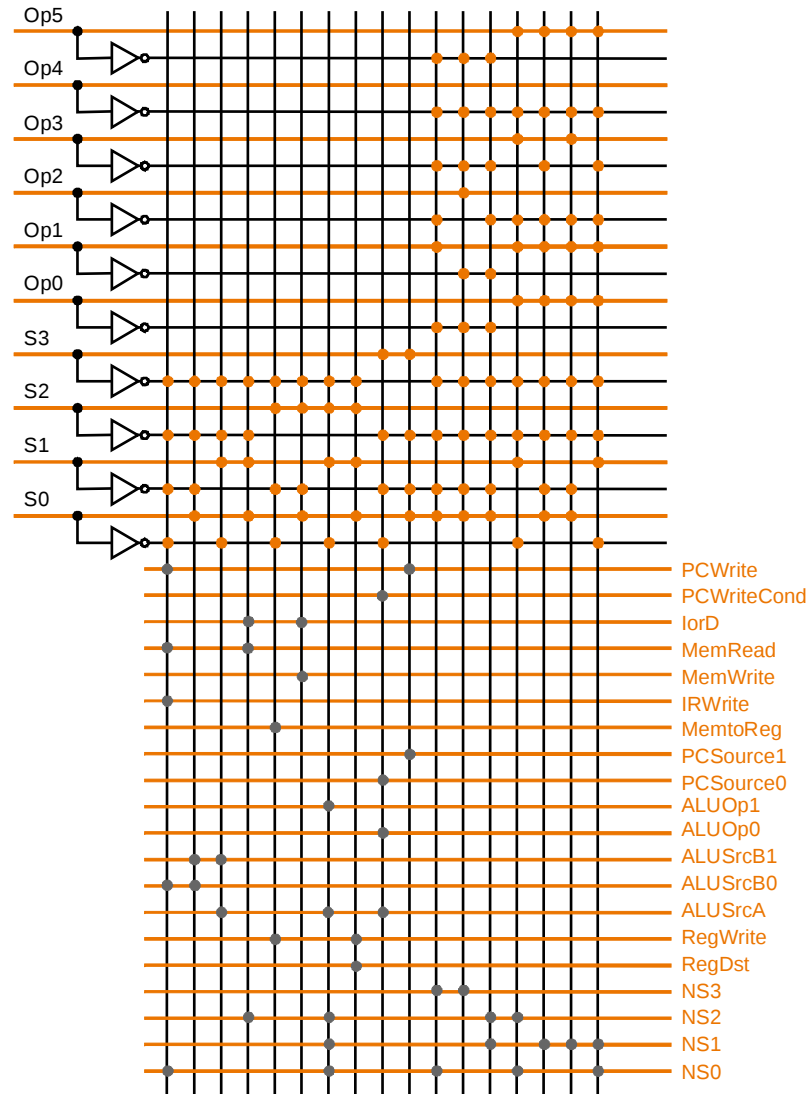
- ROM = "Read Only Memory"
 - Se graba la memoria con valores fijos
- Se usa la ROM para implementar la Tabla de Verdad
 - Con direcciones de m -bits, podemos direccionar 2^m entradas en la ROM.
 - Las salidas son los datos (palabras) direccionadas



Implementación con ROM

- ¿Cuántas entradas tenemos?
 - 6 bits para el opcode
 - 4 bits para el estado
 - = 10 líneas de direcciones ($2^{10} = 1024$ posibles direcciones)
- ¿Cuántas salidas?
 - 16 señales de control del camino de datos
 - 4 bits de estado
 - = 20 líneas de salida
- ➔ ROM de $2^{10} \times 20\text{bits} = 20\text{Kbits}$
- Problema: mucho desperdicio, ya que para muchísimas entradas, las salidas son idénticas.
 - Por ejemplo, el código de operación se ignora muchas veces

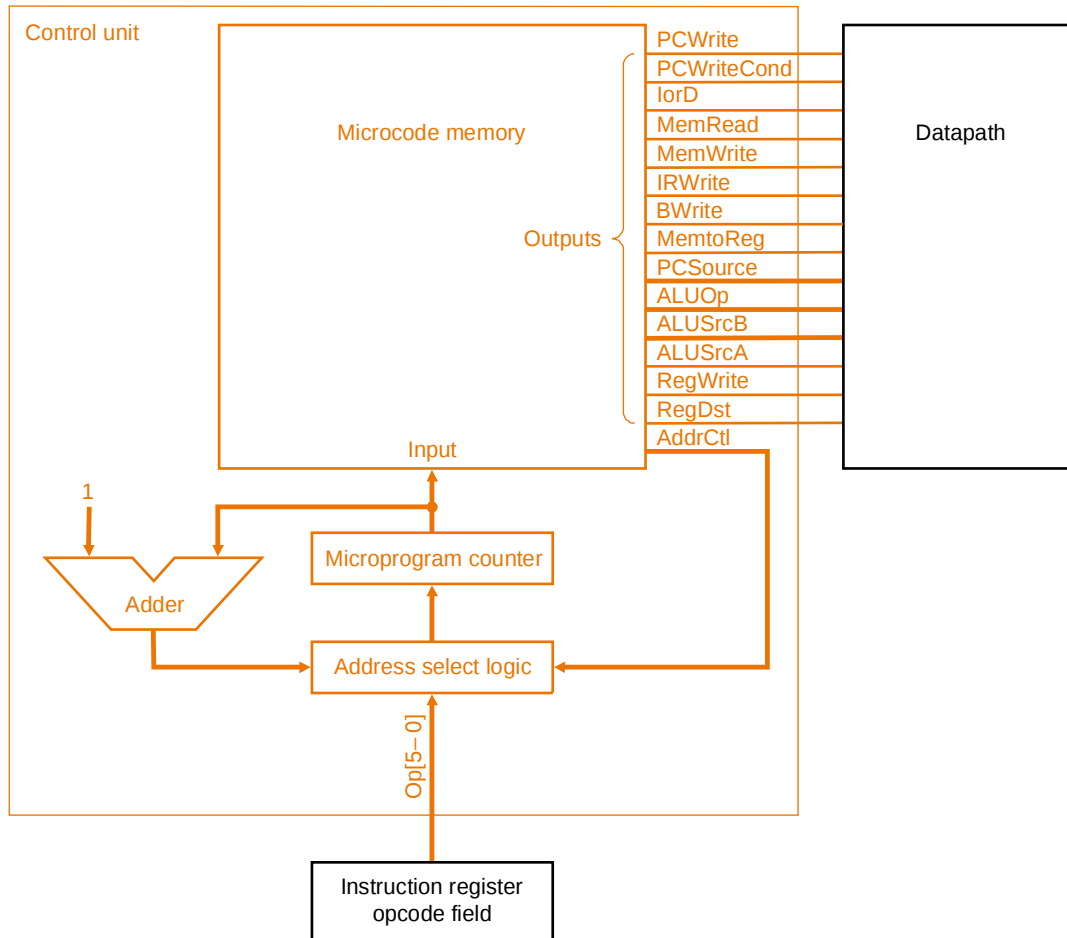
Implementación con PLA



ROM vs PLA

- Se podrían hacer dos ROM:
 - 4 bits de estado se usan como dirección de las palabras de salida: $2^4 \times 16 \text{ bits} = 256 \text{ bits}$ de ROM
 - 10 bits (6 opcode, 4 estado) se usan como dirección para la función de transición (nuevo estado): $2^{10} \times 4 \text{ bits}$ de ROM
 - Total: 4K bits de ROM
- PLA es mas pequeña
 - puede compartir términos producto
 - sólo utiliza las entradas que producen valores
 - puede considerar los “no importa”
- Tamaño $(\#inputs \times \#product\text{-}terms) + (\#outputs \times \#product\text{-}terms)$
 - En el ejemplo = $(10 \times 17) + (20 \times 17) = 460$ PLA cells
 - Una celda de PLA es un poco mas grande que una de ROM

Microprogramación



La ROM es la memoria donde se guardan las instrucciones para el camino de datos (microinstrucciones). La dirección de la ROM (microPC) es el estado.

Microprogramación

- **Es una metodología de especificación**
 - **Apropiada para arquitecturas con cientos de instrucciones, modos, alto CPI, etc.**
 - **Las señales se especifican simbólicamente usando microinstrucciones**
 - **Se define el formato de la microinstrucción, estructurado en campos.**
 - **Luego cada campo se asocia a un conjunto de señales**

Diseño del Microcódigo

- Función básica: Proveer señales para el datapath
- Dos enfoques:
 - Horizontal:
 - La microinstrucción provee todas las señales de control necesarias para un ciclo
 - Paralelismo
 - Vertical
 - Más compacta
 - Las señales están codificadas para que ocupen menos bits
 - Menos paralelismo

Diseño de microinstrucciones

- Diferentes señales agrupadas por campos

Campo	Función
Alu Control	Que operación debe hacer la ALU en este ciclo
SRC1	Especifica el 1º operando de la ALU
SRC2	Especifica el 2º operando de la ALU
Register Ctrl	Especifica Lectura/Grabación de Registros, y la fuente para la grabación
Memoria	Especifica Lectura/Grabación. En lectura el registro de destino
PCWriteCtrl	Especifica la grabación del PC
Secuencia	Determina como elegir la proxima microinstrucción

Formato de Microinstrucción

Field name	Value	Signals active	Comment
ALU control	Add	ALUOp = 00	Cause the ALU to add.
	Subt	ALUOp = 01	Cause the ALU to subtract; this implements the compare for branches.
	Func code	ALUOp = 10	Use the instruction's function code to determine ALU control.
SRC1	PC	ALUSrcA = 0	Use the PC as the first ALU input.
	A	ALUSrcA = 1	Register A is the first ALU input.
SRC2	B	ALUSrcB = 00	Register B is the second ALU input.
	4	ALUSrcB = 01	Use 4 as the second ALU input.
	Extend	ALUSrcB = 10	Use output of the sign extension unit as the second ALU input.
	Extshft	ALUSrcB = 11	Use the output of the shift-by-two unit as the second ALU input.
Register control	Read		Read two registers using the rs and rt fields of the IR as the register numbers and putting the data into registers A and B.
	Write ALU	RegWrite, RegDst = 1, MemtoReg = 0	Write a register using the rd field of the IR as the register number and the contents of the ALUOut as the data.
	Write MDR	RegWrite, RegDst = 0, MemtoReg = 1	Write a register using the rt field of the IR as the register number and the contents of the MDR as the data.
Memory	Read PC	MemRead, lorD = 0	Read memory using the PC as address; write result into IR (and the MDR).
	Read ALU	MemRead, lorD = 1	Read memory using the ALUOut as address; write result into MDR.
	Write ALU	MemWrite, lorD = 1	Write memory using the ALUOut as address, contents of B as the data.
PC write control	ALU	PCSource = 00 PCWrite	Write the output of the ALU into the PC.
	ALUOut-cond	PCSource = 01, PCWriteCond	If the Zero output of the ALU is active, write the PC with the contents of the register ALUOut.
	jump address	PCSource = 10, PCWrite	Write the PC with the jump address from the instruction.
Sequencing	Seq	AddrCtl = 11	Choose the next microinstruction sequentially.
	Fetch	AddrCtl = 00	Go to the first microinstruction to begin a new instruction.
	Dispatch 1	AddrCtl = 01	Dispatch using the ROM 1.
	Dispatch 2	AddrCtl = 10	Dispatch using the ROM 2.

Microprogramación

- Microprogramando!

Label	ALU control	SRC1	SRC2	Register control	Memory	PCWrite control	Sequencing
Fetch	Add	PC	4		Read PC	ALU	Seq
	Add	PC	Extshft	Read			Dispatch 1
Mem1	Add	A	Extend				Dispatch 2
LW2					Read ALU		Seq
				Write MDR			Fetch
SW2					Write ALU		Fetch
Rformat1	Func code	A	B				Seq
				Write ALU			Fetch
BEQ1	Subt	A	B			ALUOut-cond	Fetch
JUMP1						Jump address	Fetch

Microcódigo: Ventajas-Desventajas

- Ventajas en la especificación:
 - Fácil de diseñar: se escribe el microprograma
- Implementación en ROM (off-chip)
 - Fácil de cambiar
 - Puede emular otras arquitecturas
 - Puede usar registros internos
- Desventajas de la implementación
 - Control se implementa (hoy) en el mismo chip que el camino de datos
 - La ROM no es mas rápida que la RAM (CISC vs RISC)

Referencias

- Capitulo 5 - Patterson
 - Ver resumen en secciónDownload
- Capitulo 4 - Tanembaun
- Capitulo 16 y 17 - Stallings
- Capitulo 4 - Null