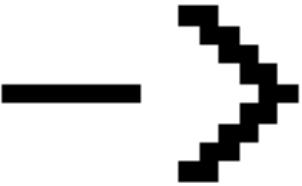


# Organización del computador

Unidad de Control y  
Microprogramación

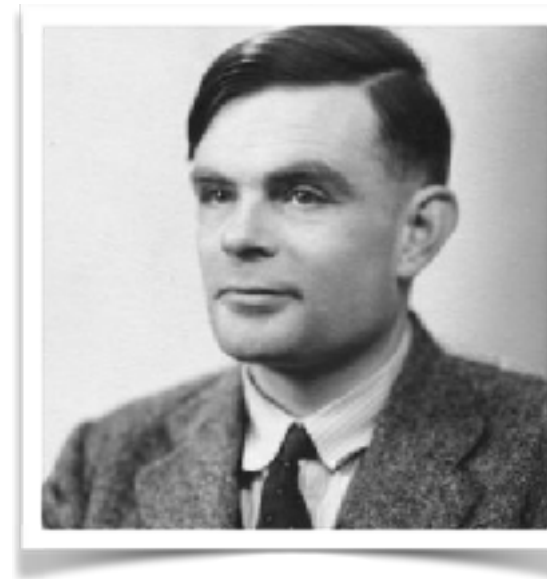
# Jerarquía de máquina



Nivel 6	Usuario	Programa ejecutables
Nivel 5	Lenguaje de alto nivel	C++, Java, Python, etc.
Nivel 4	Lenguaje ensamblador	Assembly code
Nivel 3	Software del sistema	Sistema operativo, bibliotecas, etc.
Nivel 2	Lenguaje de máquina	Instruction Set Architecture (ISA)
Nivel 1	Unidad de control	Microcódigo / hardware
Nivel 0	Lógica digital	Circuitos, compuertas, memorias

- ➤ Cada nivel funciona como una máquina abstracta que oculta la capa anterior
- ➤ Cada nivel es capaz de resolver determinado tipo de problemas a partir de comprender un tipo de instrucciones específico
- ➤ La capa inferior es utilizada como servicio

# Von Newman / Turing



- \* Los programas y los datos se almacenan en la misma memoria sobre la que se puede leer y escribir
- \* La operación de la máquina depende del estado de la memoria
- \* El contenido de la memoria es accedido a partir de su posición
- \* La ejecución es secuencial (a menos que se indique lo contrario)

# Arquitectura de von Neumann

—> 3 componentes principales:

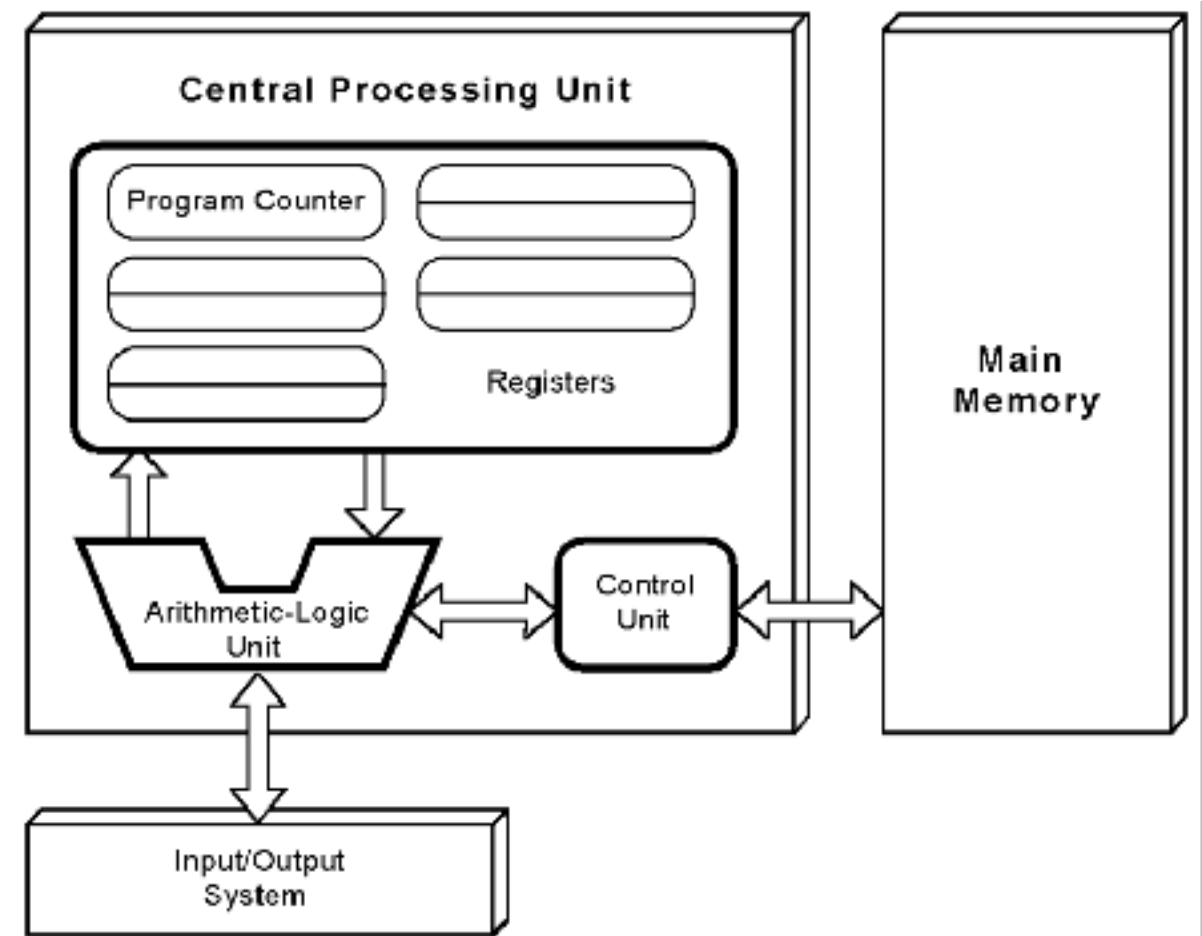
- **CPU:** Unidad de control, Unidad Aritmética lógica, Registros
- **Memoria:** Almacenamiento de programas y datos
- **Sistema** de Entrada y Salida

—> Procesamiento secuencial de instrucciones

—> Datos almacenados en sistema binario

—> Sistema de interconexión de componentes:

- Conecta la Unidad de Control con la Memoria mediante un camino único
- La unicidad del camino fuerza la alternación entre ciclos de lectura / escritura y ejecución
- Esta alternación se llama cuello de botella de von Neumann (von Neumann bottleneck)<sup>[\*]</sup>

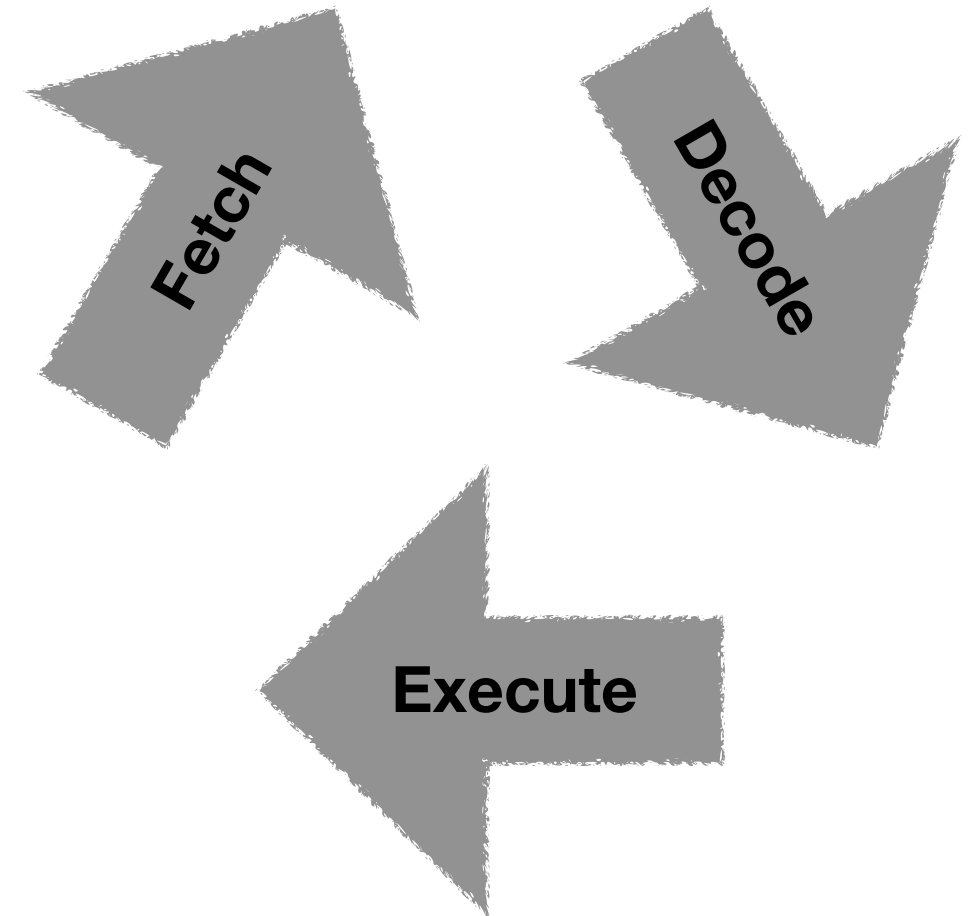


[\*] El término “cuello de botella de von Neumann” fue acuñado por John Backus en su conferencia de la concesión del Premio Turing ACM de 1977.

# Arquitectura de von Neumann

## —> Ciclo de instrucción (UC):

Fetch	<ul style="list-style-type: none"><li>- Se obtiene la instrucción apuntada por el <b>PC</b> de la <b>Memoria</b></li><li>- La <b>ALU</b> incrementa el <b>PC</b></li></ul>
Decode	<ul style="list-style-type: none"><li>- Se decodifica la instrucción</li><li>- Se obtienen los operandos de la <b>Memoria</b> y se los coloca en <b>Registros</b></li></ul>
Execute	<ul style="list-style-type: none"><li>- La <b>ALU</b> realiza la operación</li><li>- Se coloca el resultado en la <b>Memoria</b></li></ul>



# Diseño de una computadora

- ➤ Una única ALU
- ➤ Una única memoria
- ➤ Un único banco de registros
- ➤ **Problemas a resolver:**
  - ➤ Una instrucción puede utilizar un mismo recurso más de una vez lo que hace que se pierdan los valores anteriores (por ejemplo, el bus de datos o un registro)
  - ➤ Una instrucción puede utilizar el mismo recurso durante la misma etapa para más de una cosa diferente (por ejemplo, un registro para calcular una dirección y realizar una operación)

# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**
2. Selección de **componentes electrónicos**
3. Construcción del camino de datos según los requerimientos con las componentes seleccionadas
4. Analizar la implementación de cada instrucción para determinar las **señales de control** necesarias
5. Construir la **unidad de control** que implemente el comportamiento necesario

# Etapas de la ejecución de una instrucción

1. Fetch de instrucción (**Fetch / IF**)
2. Decodificación de instrucción (y lectura de registros) (**Decode / ID**)
3. Lectura a datos de la memoria previo cálculo de la dirección (**Mem**)
4. Ejecución de la instrucción (**Execution / Ex**)
5. Escritura de resultados en la memoria o registros (**Write back / WB**)



# MIPS - Microprocessor without Interlocked Pipeline Stages

- Procesador RISC desarrollado por MIPS Computer Systems
- 32 registros de propósito general (salvo excepciones)
- 3 tipos de instrucciones de 32 bits con el siguiente formato:

Type	format (bits)					
	-31-					-0-
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

- El PC es de 32 bits: los 4 más significativos (31..28) dividen la memoria en (Reserved, Text, Data y Stack), los 26 siguientes son la dirección relativa en dicho espacio y los 2 menos significativos son 00 para alinear a palabra.
- Las instrucciones se dividen en: **1- CPU instructions** (Loads and Stores, ALU, Shifts, Multiplication and division, Jump and Branch y Exception) y **2- Floating-Point Unit instructions** (Arithmetic, Data transfer y Branch)

---

La arquitectura MIPS posee un coprocesador dedicado a operaciones sobre números de punto flotante.

# Formato de instrucción MIPS

## Loads and Stores



Instruction name	Mnemonic	Format	Encoding			
Load Byte	LB	I	32	rs	rt	offset
Load Halfword	LH	I	33	rs	rt	offset
Load Word Left	LWL	I	34	rs	rt	offset
Load Word	LW	I	35	rs	rt	offset
Load Byte Unsigned	LBU	I	36	rs	rt	offset
Load Halfword Unsigned	LHU	I	37	rs	rt	offset
Load Word Right	LWR	I	38	rs	rt	offset
Store Byte	SB	I	40	rs	rt	offset
Store Halfword	SH	I	41	rs	rt	offset
Store Word Left	SWL	I	42	rs	rt	offset
Store Word	SW	I	43	rs	rt	offset
Store Word Right	SWR	I	46	rs	rt	offset



LW \$1, 100(\$2)    I    \$1 = M[\$2+100]

SW \$1, 100(\$2)    I    M[\$2+100] = \$1

# Formato de instrucción MIPS

## ALU

ADD \$1, \$2, \$3    I    \$1 = \$2 + \$3

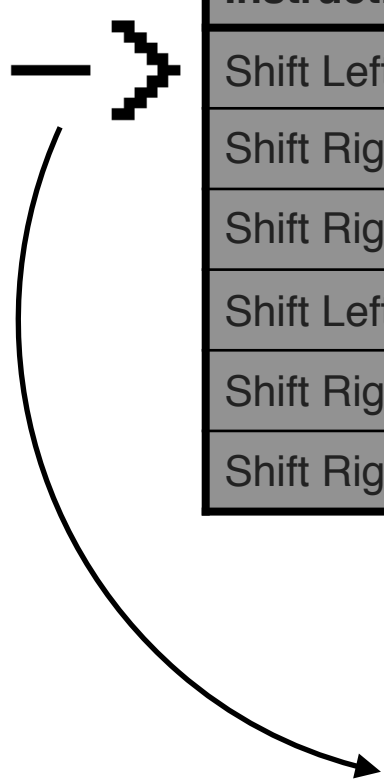
Instruction name	Mnemonic	Format	Encoding					
Add	ADD	R	0	rs	rt	rd	0	32
Add Unsigned	ADDU	R	0	rs	rt	rd	0	33
Subtract	SUB	R	0	rs	rt	rd	0	34
Subtract Unsigned	SUBU	R	0	rs	rt	rd	0	35
And	AND	R	0	rs	rt	rd	0	36
Or	OR	R	0	rs	rt	rd	0	37
Exclusive Or	XOR	R	0	rs	rt	rd	0	38
Nor	NOR	R	0	rs	rt	rd	0	39
Set on Less Than	SLT	R	0	rs	rt	rd	0	42
Set on Less Than Unsigned	SLTU	R	0	rs	rt	rd	0	43
Add Immediate	ADDI	I	8	rs	rd	immediate		
Add Immediate Unsigned	ADDIU	I	9	\$s	\$d	immediate		
Set on Less Than Immediate	SLTI	I	10	\$s	\$d	immediate		
Set on Less Than Immediate Unsigned	SLTIU	I	11	\$s	\$d	immediate		
And Immediate	ANDI	I	12	\$s	\$d	immediate		
Or Immediate	ORI	I	13	\$s	\$d	immediate		
Exclusive Or Immediate	XORI	I	14	\$s	\$d	immediate		
Load Upper Immediate	LUI	I	15	010	\$d	immediate		

SLT \$1, \$2, \$3    I    Si \$2 < \$3 entonces \$1 = 1, si no \$1 = 0

ADDI \$1, \$2, 100    I    \$1 = \$2 + 100

# Formato de instrucción MIPS

## Shifts



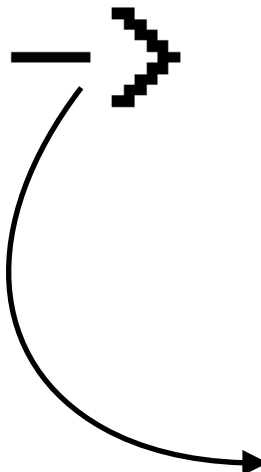
Instruction name	Mnemonic	Format	Encoding					
Shift Left Logical	SLL	R	0	0	rt	rd	ra	0
Shift Right Logical	SRL	R	0	0	rt	rd	sa	2
Shift Right Arithmetic	SRA	R	0	0	rt	rd	sa	3
Shift Left Logical Variable	SLLV	R	0	rs	rt	rd	0	4
Shift Right Logical Variable	SRLV	R	0	rs	rt	rd	0	6
Shift Right Arithmetic Variable	SRAV	R	0	rs	rt	rd	0	7

SLL \$1, \$2, 4    I    \$1 = \$2 << 4

# Formato de instrucción MIPS

## Multiplication and Division

Instruction name	Mnemonic	Format	Encoding					
Move from HI	MFHI	R	0	0	0	rd	0	16
Move to HI	MTHI	R	0	rs	0	0	0	17
Move from LO	MFLO	R	0	0	0	rd	0	18
Move to LO	MTLO	R	0	rs	0	0	0	19
Multiply	MULT	R	0	rs	rt	0	0	24
Multiply Unsigned	MULTU	R	0	rs	rt	0	0	25
Divide	DIV	R	0	rs	rt	0	0	26
Divide Unsigned	DIVU	R	0	rs	rt	0	0	27

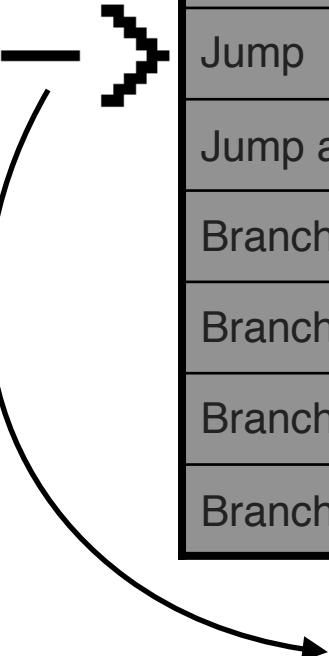


MULT \$1, \$2    I    \$t0 = \$1 \* \$2[31..0] y \$t1 = \$1 \* \$2[63..32]

# Formato de instrucción MIPS

## Jump and Branch

Instruction name	Mnemonic	Format	Encoding					
Jump Register	JR	R	0	rs	0	0	0	8
Jump and Link Register	JALR	R	0	rs	0	rd	0	9
Branch on Less Than Zero	BLTZ	I	1	rs	0	offset		
Branch on Greater Than or Equal to Zero	BGEZ	I	1	rs	1	offset		
Branch on Less Than Zero and Link	BLTZAL	I	1	rs	16	offset		
Branch on Greater Than or Equal to Zero and Link	BGEZAL	I	1	rs	17	offset		
Jump	J	J	2	instr_index				
Jump and Link	JAL	J	3	instr_index				
Branch on Equal	BEQ	I	4	rs	rt	offset		
Branch on Not Equal	BNE	I	5	rs	rt	offset		
Branch on Less Than or Equal to Zero	BLEZ	I	6	rs	0	offset		
Branch on Greater Than Zero	BGTZ	I	7	rs	0	offset		



J 10000    I    PC[27..2] = 10000 y PC[1..0] = 00

# Formato de instrucción MIPS

**Exception**

**FPU instructions - Arithmetic**

**FPU instructions - Data transfer**

**FPU instructions - Branch**

# RTL - Register Transfer Language

- ➤ Cada instrucción es implementada como un **microprograma**
- ➤ Los microprogramas son secuencias de **microoperaciones**
- ➤ Las microoperaciones permiten mover datos entre registros y memoria
- ➤ La ejecución de las microoperaciones presenta una precedencia temporal derivada de la utilización múltiple de recursos
- ➤ **Ejemplo:** el Fetch de una instrucción (en Marie) se escribe:  
     $IR \leftarrow Mem[PC]$   
     $PC \leftarrow PC + 4$

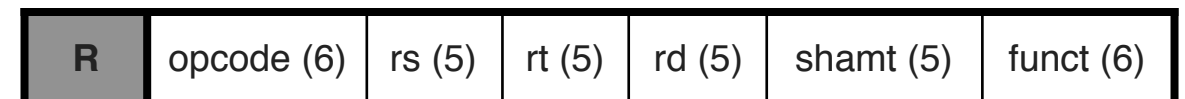


# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

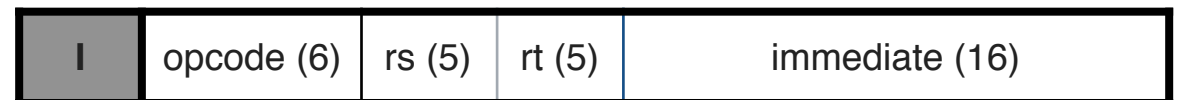
## Add y Sub:

ADDU rd, rs, rt |  $R[rd] \leftarrow R[rs] + R[rt]$   
SUBU rd, rs, rt |  $R[rd] \leftarrow R[rs] - R[rt]$



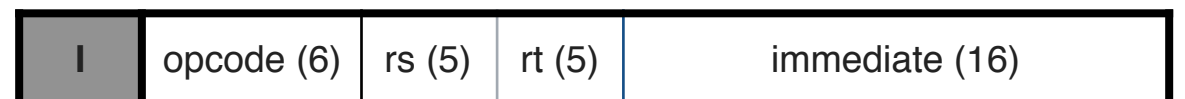
## Load y Store:

LW rt, rs, imm16 |  $R[rt] \leftarrow \text{Mem}[R[rs] + \text{sign\_ext}(\text{imm16})]$   
SW rt, rs, imm16 |  $\text{Mem}[R[rs] + \text{sign\_ext}(\text{imm16})] \leftarrow R[rt]$



## Branch:

BEQ rt, rs, imm16 |  
if ( $R[rt] == R[rs]$ ) then  $PC \leftarrow PC + \text{sign\_ext}(\text{imm16}) * 4$



## Jump:

J imm26 |  
 $PC[31..2] \leftarrow PC[31..28] ++ (\text{imm26} \ll 2)$



# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

## Add y Sub:

ADDU rd, rs, rt |  $R[rd] \leftarrow R[rs] + R[rt]$   
SUBU rd, rs, rt |  $R[rd] \leftarrow R[rs] - R[rt]$

R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
---	------------	--------	--------	--------	-----------	-----------

## Implementación en RTL:

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

$PC \leftarrow PC + 4$

**T2:**  $A \leftarrow R[rs]$

$B \leftarrow R[rt]$

**T3:**  $\text{ALUOut} \leftarrow A +|- B$

**T4:**  $R[rd] \leftarrow \text{ALUOut}$

# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

## Load y Store:



LW rt, rs, imm16 |  $R[rt] \leftarrow \text{Mem}[R[rs] + \text{sign\_ext}(\text{imm16})]$

## Implementación en RTL:

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

$PC \leftarrow PC + 4$

**T2:**  $A \leftarrow R[rs]$

$B \leftarrow R[rt]$

// B no va a ser utilizado.

**T3:**  $\text{ALUOut} \leftarrow A + \text{sign\_ext}(IR[15..0])$

// Cálculo de la dirección.

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

**T5:**  $R[rt] \leftarrow \text{MDR}$

# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

## Load y Store:



SW rt, rs, imm16 |  $\text{Mem}[\text{R}[\text{rs}] + \text{sign\_ext}(\text{imm16})] \leftarrow \text{R}[\text{rt}]$

## Implementación en RTL:

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

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

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

$\text{B} \leftarrow \text{R}[\text{rt}]$

// Valor a escribir.

**T3:**  $\text{ALUOut} \leftarrow \text{A} + \text{sign\_ext}(\text{IR}[15..0])$

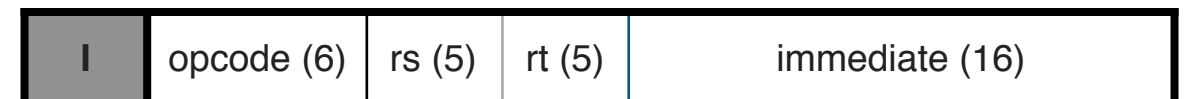
// Cálculo de la dirección.

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

# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

## Branch:



BEQ rt, rs, imm16 |

if (R[rt] == R[rs]) then PC  $\leftarrow$  PC + sign\_ext(imm16)\*4

## Implementación en RTL:

**T1:** IR  $\leftarrow$  Mem[PC]

PC  $\leftarrow$  PC + 4

**T2:** A  $\leftarrow$  R[rs]

B  $\leftarrow$  R[rt]

ALUOut  $\leftarrow$  PC + sign\_ext(IR[15..0]) << 2 // Cálculo de la dirección de salto.

**T3:** Comp A B

if zero then PC  $\leftarrow$  ALUOut // Observa si el flag correspondiente a zero está en 1.

# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

## Jump:

J imm26 |  
 $PC[31..2] \leftarrow PC[31..28]++(imm26 \ll 2)$



## Implementación en RTL:

**T1:** IR  $\leftarrow$  Mem[PC]  
PC  $\leftarrow$  PC + 4

**T2:** — — — — —

**T3:**  $PC[31..2] \leftarrow PC[31..28]++(IR[25..0] \ll 2)$

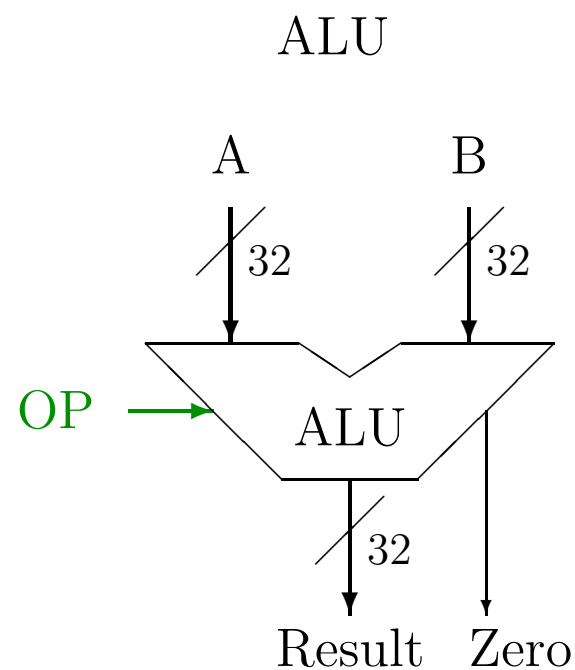
# Diseño de una computadora

1. Analizar el conjunto de instrucciones para determinar los requerimientos del **camino de datos**

Cycle	Instruction type	Action
IF	Todas	$IR \leftarrow \text{Mem}[PC]$ $PC \leftarrow PC + 4$
	Todas salvo J	$A \leftarrow R[rs]$ $B \leftarrow R[rt]$
ID	BEQ	$ALUOut \leftarrow PC + \text{sign\_ext}(IR[15..0]) \ll 2$
	ADDU/SUBU LW/SW BEQ J	$ALUOut \leftarrow A + - B$ $ALUOut \leftarrow A + \text{sign\_ext}(IR[15..0])$ Comp A B, if zero then $PC \leftarrow ALUOut$ $PC[31..2] \leftarrow PC[31..28] ++ (IR[25..0] \ll 2)$
EX	LW SW	$MDR \leftarrow \text{Mem}[ALUOut]$ $\text{Mem}[ALUOut] \leftarrow B$
	ADDU/SUBU LW	$R[rd] \leftarrow ALUOut$ $R[rt] \leftarrow MDR$
Mem		
WB		

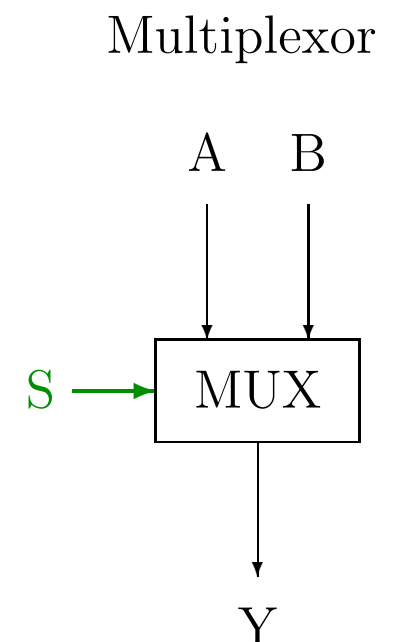
# Diseño de una computadora

## 2. Selección de componentes electrónicos



— ➤ Todas las operaciones aritmético-lógicas serán realizadas por una **única** unidad aritmética lógica

— ➤ Varios componentes como la ALU tienen entradas variadas dependiendo del código de la operación que se está ejecutando; se utilizarán multiplexores para permitir la selección usando líneas de control



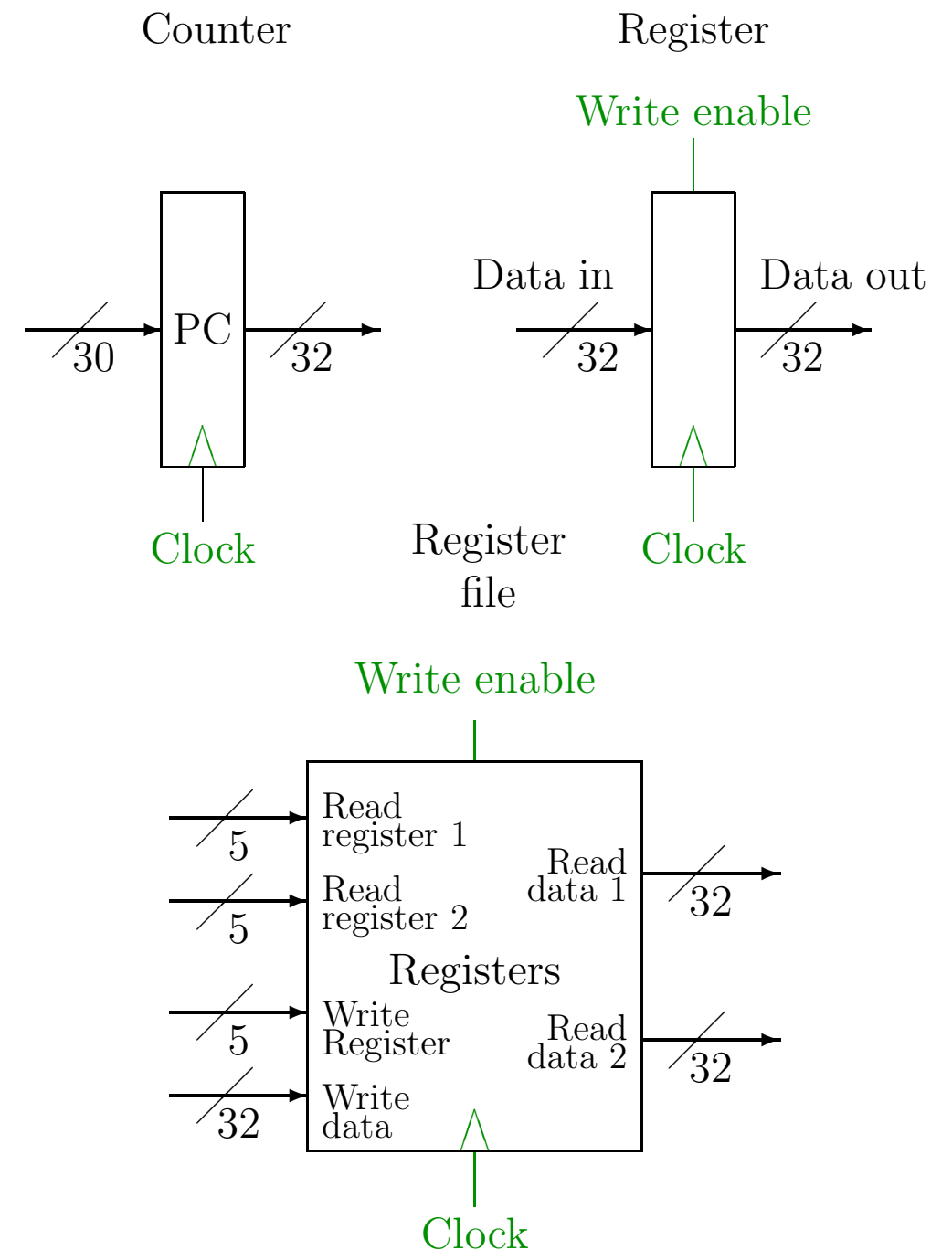


# Diseño de una computadora

## 2. Selección de **componentes electrónicos**

— ➤ Por un lado utilizaremos registros ad-hoc de 32 bits con el objetivo de resolver tareas específicas como ser la de almacenar el program counter o las entradas y/o salidas de la ALU

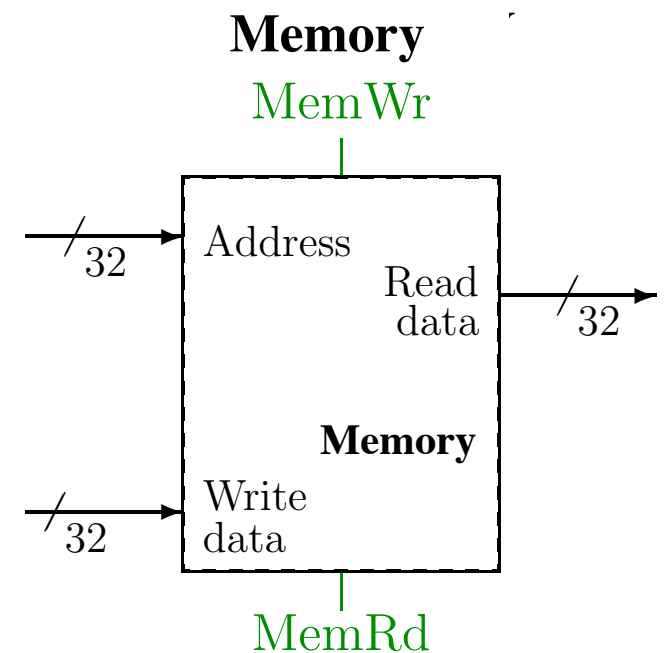
— ➤ Por el otro contaremos con un banco de 32 registros de 32 bits de propósito general que permitirá la realización de dos lecturas simultáneas



# Diseño de una computadora

## 2. Selección de **componentes electrónicos**

- ➤ Una **única** unidad de memoria donde se encuentran almacenados tanto los programas como los datos sobre los que estos ejecutan



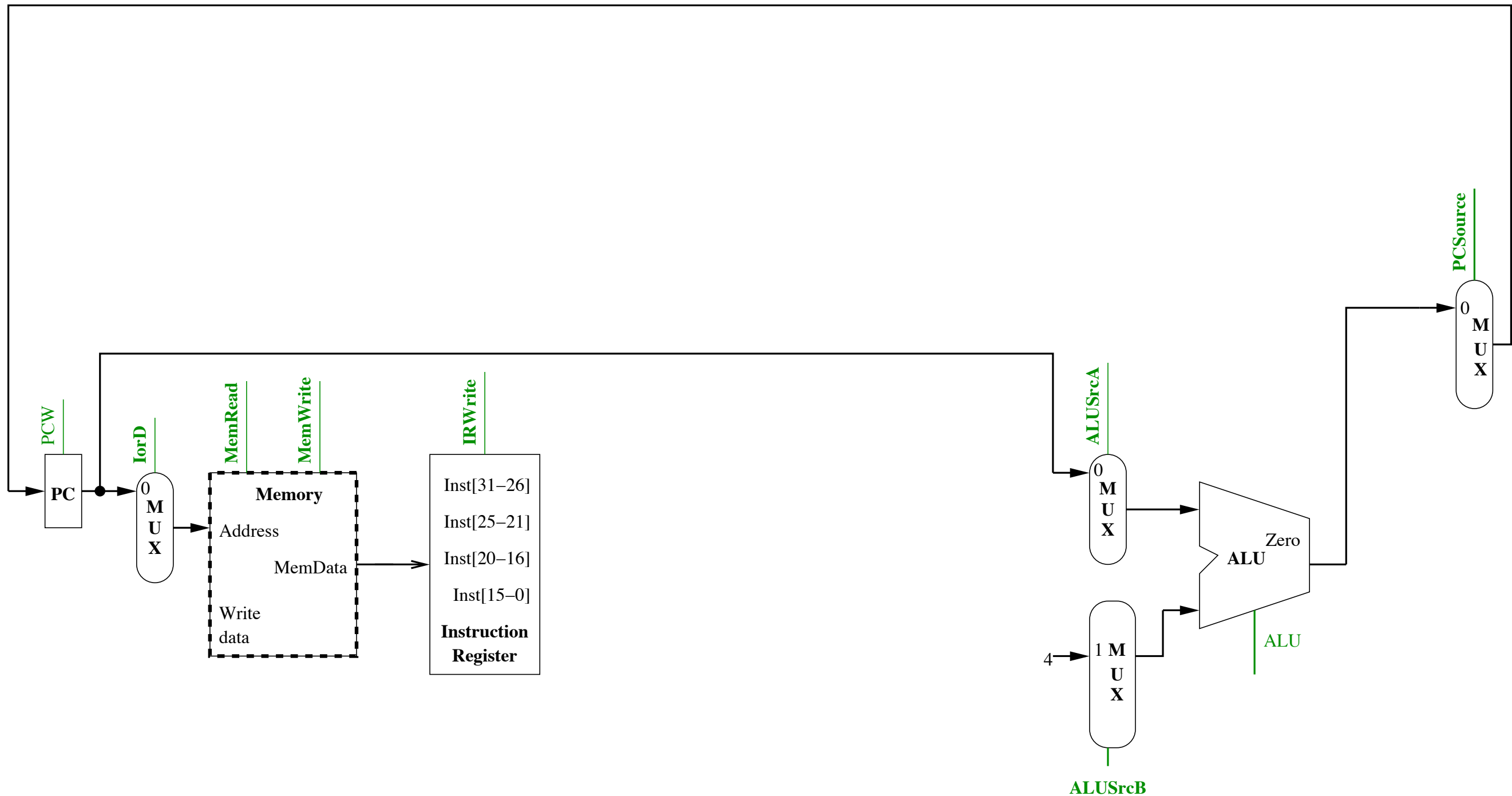
# Diseño de una computadora

## 3. Construcción del camino de datos según los requerimientos con las componentes seleccionadas

Cycle	Instruction type	Action
IF	Todas	$IR \leftarrow \text{Mem}[PC]$ $PC \leftarrow PC + 4$
ID	Todas salvo J	$A \leftarrow R[rs]$ $B \leftarrow R[rt]$
	BEQ	$ALUOut \leftarrow PC + \text{sign\_ext}(IR[15..0]) \ll 2$
EX	ADDU/SUBU LW/SW BEQ J	$ALUOut \leftarrow A + - B$ $ALUOut \leftarrow A + \text{sign\_ext}(IR[15..0])$ Comp A B, if zero then $PC \leftarrow ALUOut$ $PC[31..2] \leftarrow PC[31..28] ++ (IR[25..0] \ll 2)$
Mem	LW SW	$MDR \leftarrow \text{Mem}[ALUOut]$ $\text{Mem}[ALUOut] \leftarrow B$
WB	ADDU/SUBU LW	$R[rd] \leftarrow ALUOut$ $R[rt] \leftarrow MDR$

# Diseño de una computadora

## Instruction Fetch



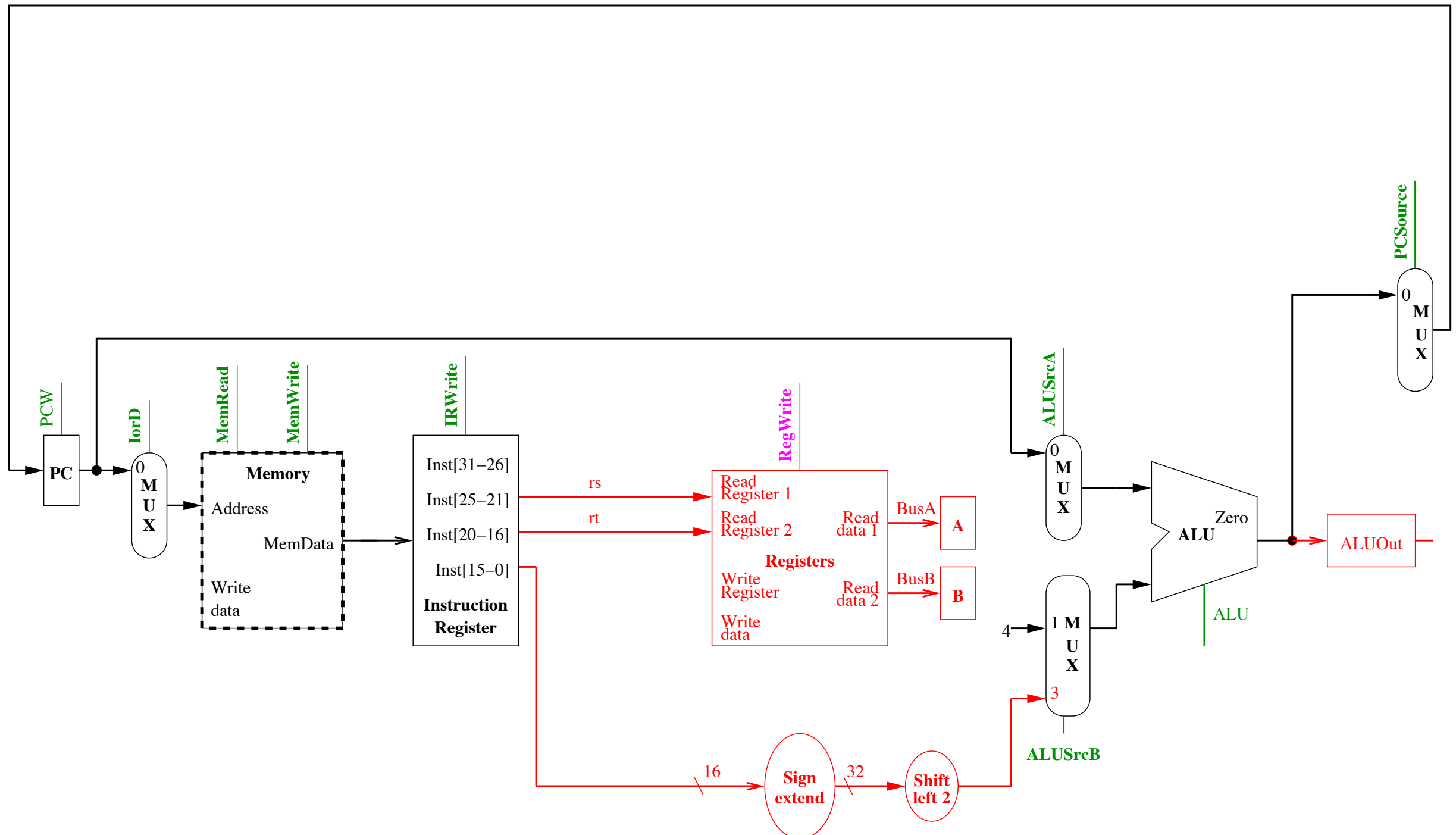
# Diseño de una computadora

## 3. Construcción del camino de datos según los requerimientos con las componentes seleccionadas

Cycle	Instruction type	Action
IF	Todas	$IR \leftarrow \text{Mem}[PC]$ $PC \leftarrow PC + 4$
	Todas salvo J	$A \leftarrow R[rs]$ $B \leftarrow R[rt]$
ID	BEQ	$ALUOut \leftarrow PC + \text{sign\_ext}(IR[15..0]) \ll 2$
	ADDU/SUBU LW/SW BEQ J	$ALUOut \leftarrow A + - B$ $ALUOut \leftarrow A + \text{sign\_ext}(IR[15..0])$ Comp A B, if zero then $PC \leftarrow ALUOut$ $PC[31..2] \leftarrow PC[31..28] ++ (IR[25..0] \ll 2)$
EX	LW SW	$MDR \leftarrow \text{Mem}[ALUOut]$ $\text{Mem}[ALUOut] \leftarrow B$
	Mem	
WB	ADDU/SUBU LW	$R[rd] \leftarrow ALUOut$ $R[rt] \leftarrow MDR$
	WB	

# Diseño de una computadora

## Instruction Decode



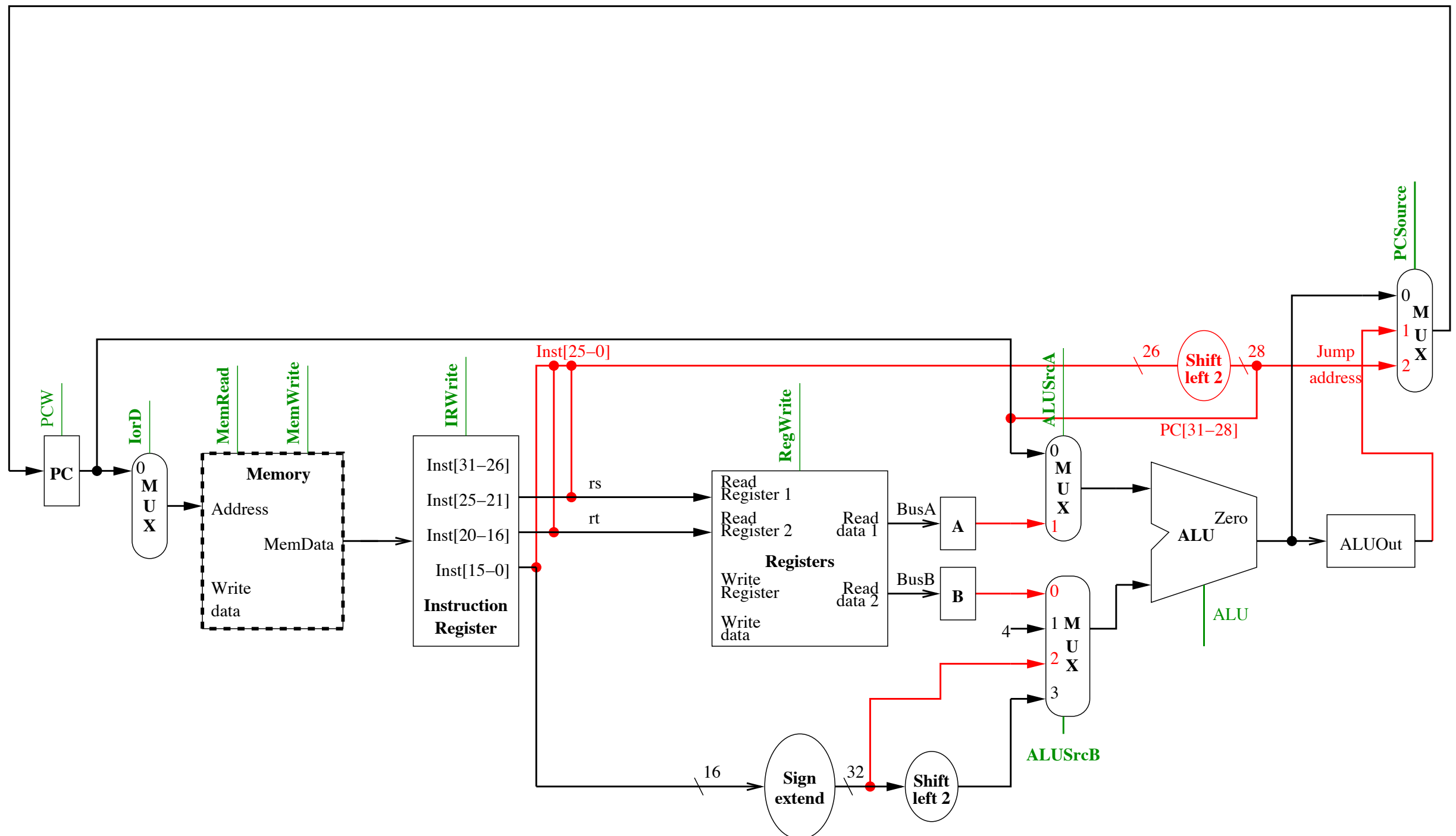
# Diseño de una computadora

## 3. Construcción del camino de datos según los requerimientos con las componentes seleccionadas

Cycle	Instruction type	Action
IF	Todas	$IR \leftarrow \text{Mem}[PC]$ $PC \leftarrow PC + 4$
	Todas salvo J	$A \leftarrow R[rs]$ $B \leftarrow R[rt]$
ID	BEQ	$ALUOut \leftarrow PC + \text{sign\_ext}(IR[15..0]) \ll 2$
	ADDU/SUBU LW/SW BEQ J	$ALUOut \leftarrow A + - B$ $ALUOut \leftarrow A + \text{sign\_ext}(IR[15..0])$ Comp A B, if zero then $PC \leftarrow ALUOut$ $PC[31..2] \leftarrow PC[31..28] ++ (IR[25..0] \ll 2)$
EX	LW SW	$MDR \leftarrow \text{Mem}[ALUOut]$ $\text{Mem}[ALUOut] \leftarrow B$
	ADDU/SUBU LW	$R[rd] \leftarrow ALUOut$ $R[rt] \leftarrow MDR$
Mem		
WB		

# Diseño de una computadora

## Instruction Execute





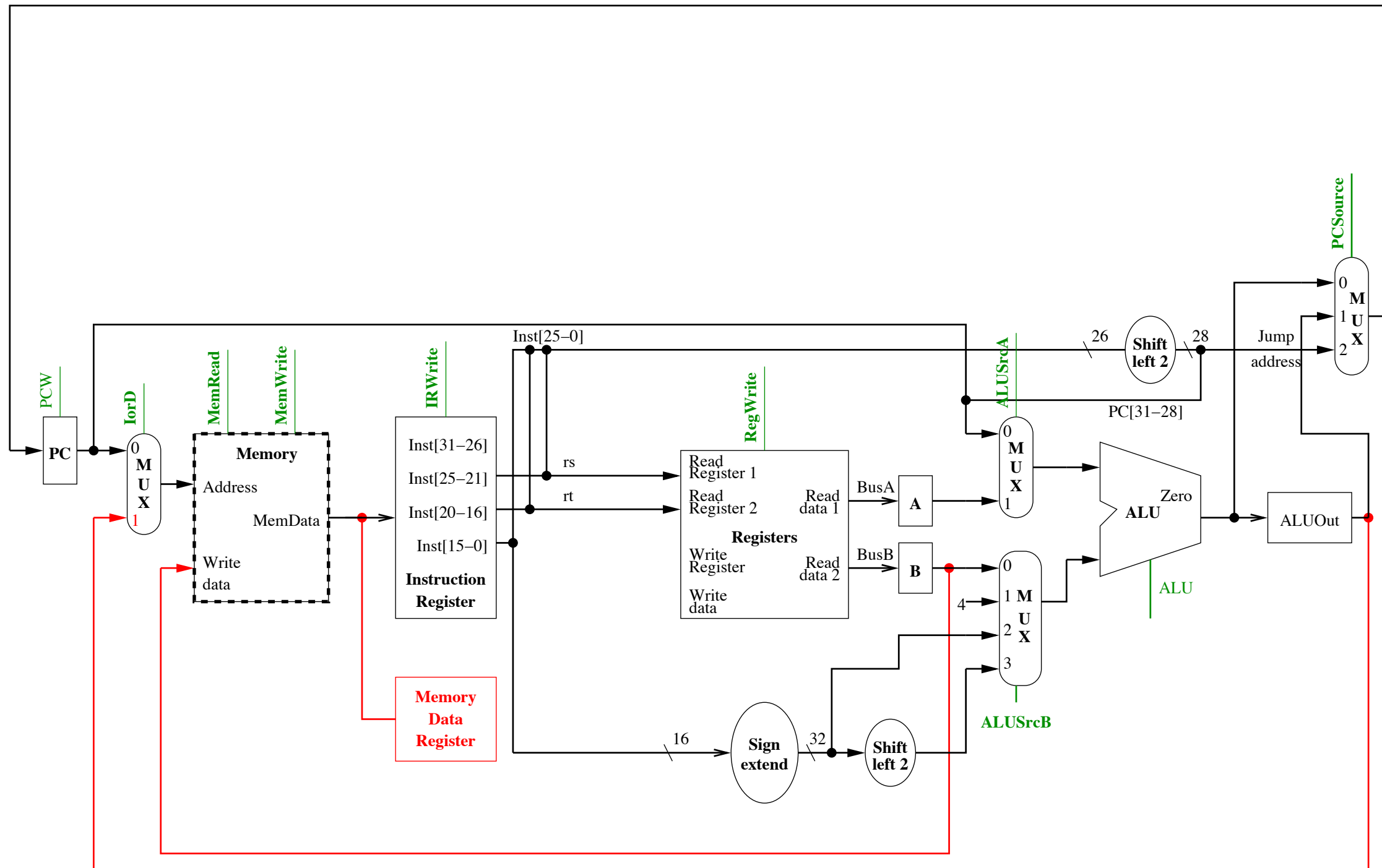
# Diseño de una computadora

## 3. Construcción del camino de datos según los requerimientos con las componentes seleccionadas

Cycle	Instruction type	Action
IF	Todas	$IR \leftarrow \text{Mem}[PC]$ $PC \leftarrow PC + 4$
	Todas salvo J	$A \leftarrow R[rs]$ $B \leftarrow R[rt]$
ID	BEQ	$ALUOut \leftarrow PC + \text{sign\_ext}(IR[15..0]) \ll 2$
	ADDU/SUBU LW/SW BEQ J	$ALUOut \leftarrow A + - B$ $ALUOut \leftarrow A + \text{sign\_ext}(IR[15..0])$ Comp A B, if zero then $PC \leftarrow ALUOut$ $PC[31..2] \leftarrow PC[31..28] ++ (IR[25..0] \ll 2)$
EX	LW SW	$MDR \leftarrow \text{Mem}[ALUOut]$ $\text{Mem}[ALUOut] \leftarrow B$
	ADDU/SUBU LW	$R[rd] \leftarrow ALUOut$ $R[rt] \leftarrow MDR$
Mem		
WB		

# Diseño de una computadora

Mem

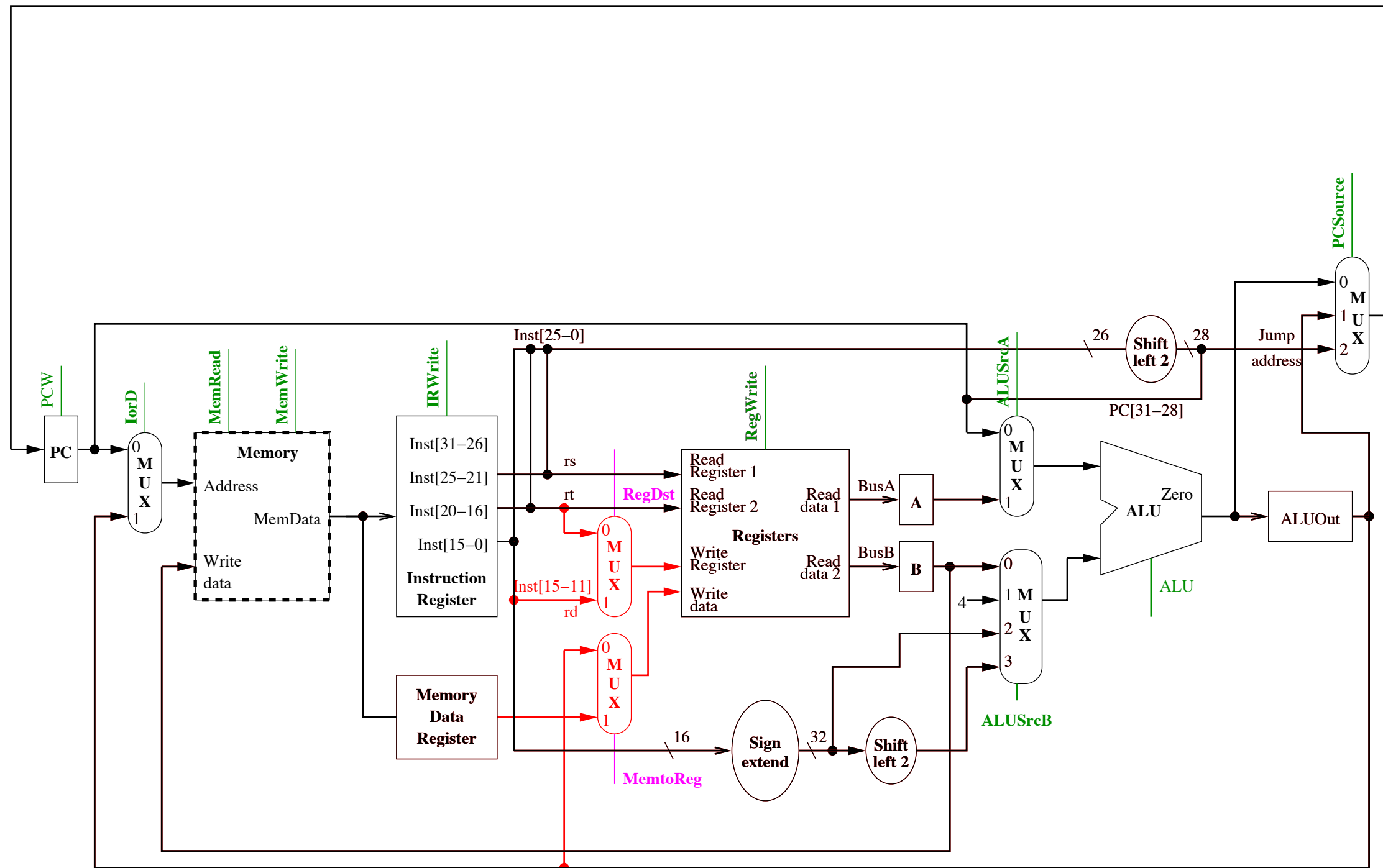


# Diseño de una computadora

## 3. Construcción del camino de datos según los requerimientos con las componentes seleccionadas

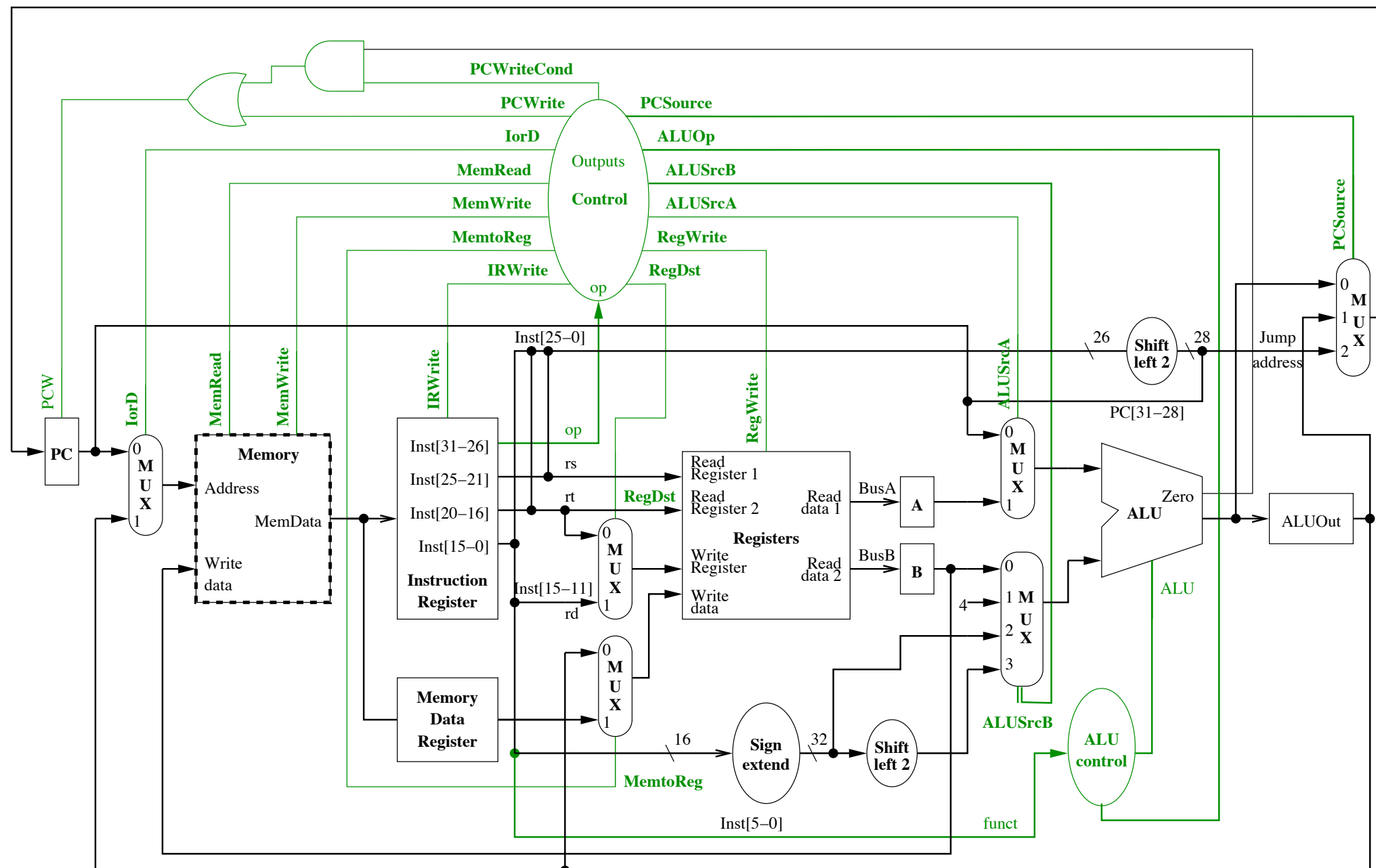
Cycle	Instruction type	Action
IF	Todas	$IR \leftarrow \text{Mem}[PC]$ $PC \leftarrow PC + 4$
	Todas salvo J	$A \leftarrow R[rs]$ $B \leftarrow R[rt]$
ID	BEQ	$ALUOut \leftarrow PC + \text{sign\_ext}(IR[15..0]) \ll 2$
	ADDU/SUBU LW/SW BEQ J	$ALUOut \leftarrow A + - B$ $ALUOut \leftarrow A + \text{sign\_ext}(IR[15..0])$ Comp A B, if zero then $PC \leftarrow ALUOut$ $PC[31..2] \leftarrow PC[31..28] ++ (IR[25..0] \ll 2)$
EX	LW SW	$MDR \leftarrow \text{Mem}[ALUOut]$ $\text{Mem}[ALUOut] \leftarrow B$
	Mem	
WB	ADDU/SUBU LW	$R[rd] \leftarrow ALUOut$ $R[rt] \leftarrow MDR$

## Write Back



# Diseño de una computadora

4. Analizar la implementación de cada instrucción para determinar las **señales de control** necesarias



# Diseño de una computadora

4. Analizar la implementación de cada instrucción para determinar las **señales de control** necesarias

Señal	Acción	
	0	1
<b>RegDst</b>	El registro destino es <b>rt</b>	El registro destino es <b>rd</b>
<b>RegWrite</b>	No se escribe en el banco de registros	Si se escribe en el banco de registros
<b>ALUSrcA</b>	El primer operando de la ALU es el <b>PC</b>	El primer operando de la ALU es el registro <b>A</b>
<b>MemRead</b>	No se lee la memoria	El contenido de la memoria en la dirección especificada es colocado en el bus de datos
<b>MemWrite</b>	No se escribe la memoria	El contenido del registro <b>B</b> es escrito en la memoria en la dirección especificada
<b>MemtoReg</b>	El valor escrito en el banco de registros proviene de <b>ALUOut</b>	El valor escrito en el banco de registros proviene de <b>MDR</b>
<b>lorD</b>	La dirección proviene del <b>PC</b>	La dirección proviene del <b>ALUOut</b>
<b>IRWrite</b>	No se escribirá en el <b>IR</b>	Se escribirá en <b>IR</b>
<b>PCWrite</b>	— — —	Se escribe en el <b>PC</b> el valor de salida del MUX controlado por la señal <b>PCSource</b>
<b>PCWriteCond</b>	Si junto con <b>PCWrite</b> , no se escribe en <b>PC</b>	Se escribe el <b>PC</b> si el flag <b>zero</b> de la ALU está en 1

# Diseño de una computadora

4. Analizar la implementación de cada instrucción para determinar las **señales de control** necesarias

Señal	Valor	Acción
ALUOp	00	La ALU realizará una suma
	01	La ALU realizará una resta
	10	La ALU realizará la operación declarada en <b>funct</b>
ALUSrcB	00	El segundo operando de la ALU es el registro <b>B</b>
	01	El segundo operando de la ALU es 4
	10	El segundo operando de la ALU es la extensión con signo de los 16 bits menos significativos del <b>IR</b>
	11	El segundo operando de la ALU es la extensión con signo de los 16 bits menos significativos del <b>IR</b> decalado dos bits
PCSource	00	El <b>PC</b> se actualiza con <b>PC + 4</b>
	01	El <b>PC</b> se actualiza con el valor de <b>ALUOut</b> (el destino la operación BEQ)
	10	El <b>PC</b> se actualiza con el destino de la operación J

# Diseño de una computadora

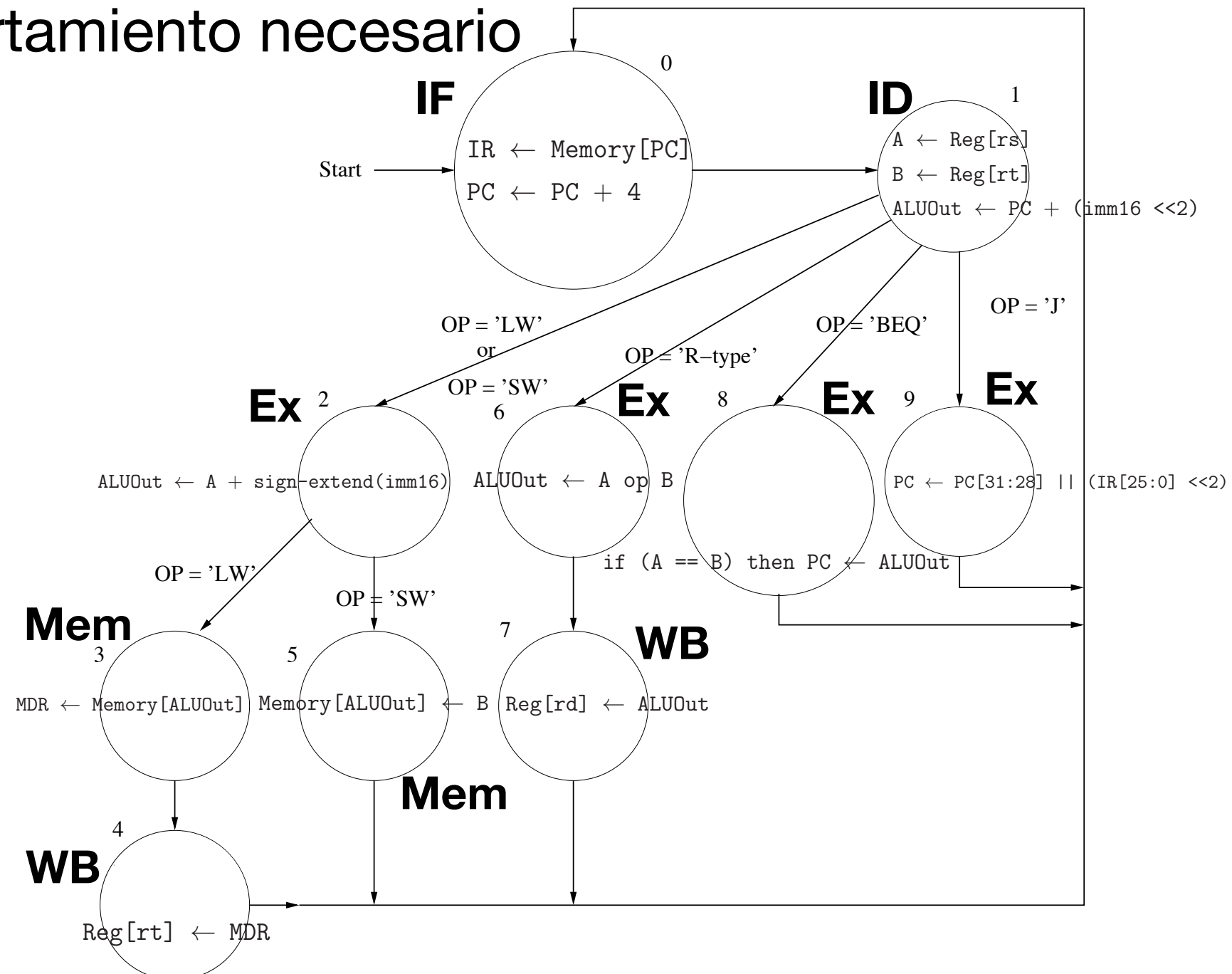
## 5. Construir la **unidad de control** que implemente el comportamiento necesario

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	$\text{Reg}[rd] \leftarrow ALUOut$ $\text{Reg}[rt] \leftarrow MDR$



# Diseño de una computadora

5. Construir la **unidad de control** que implemente el comportamiento necesario





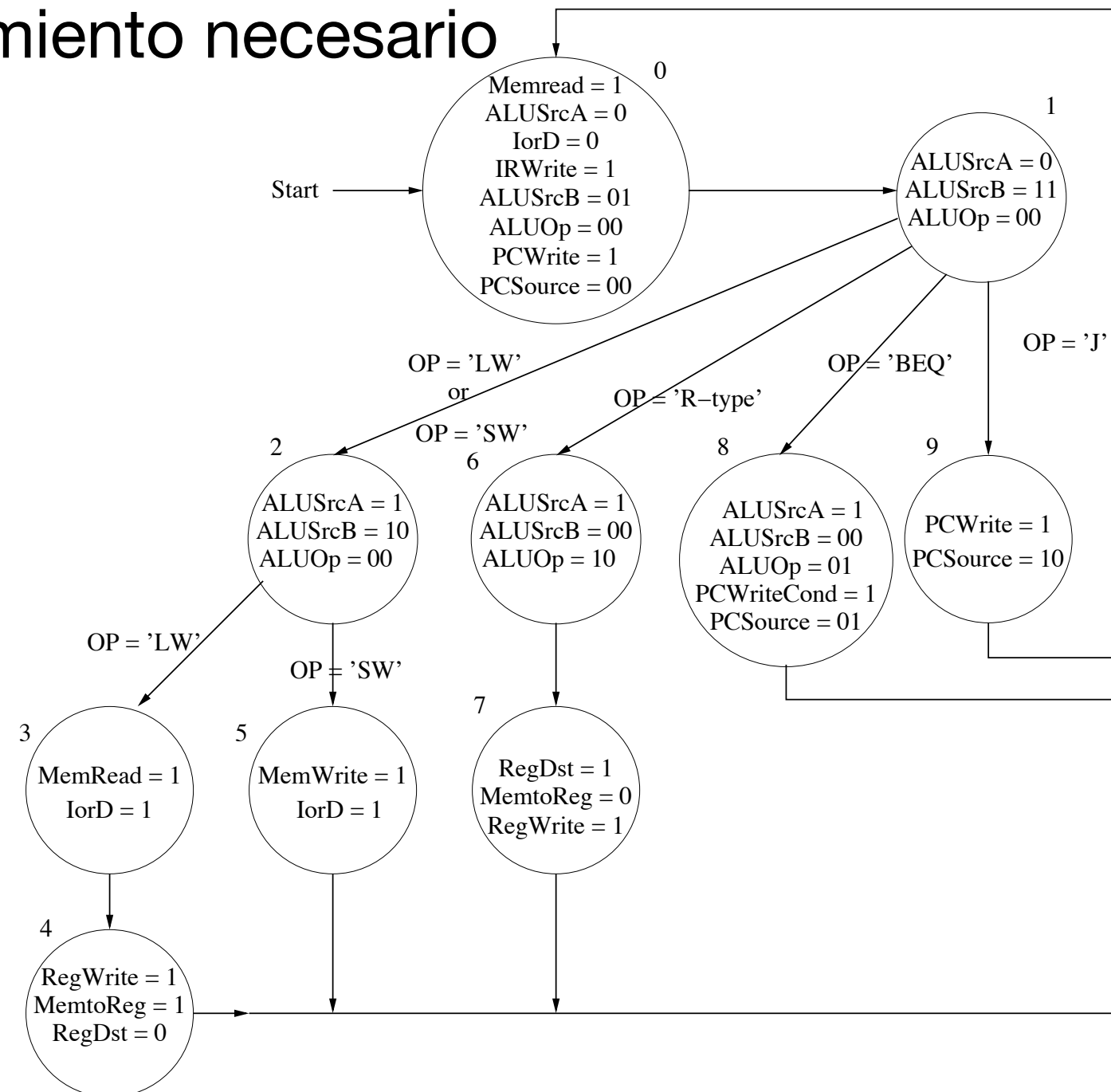
# Diseño de una computadora

5. Construir la **unidad de control** que implemente el comportamiento necesario

	0	1	2	3	4	5	6	7	8	9
RegWrite	0	0	0	0	1	0	0	1	0	0
IRWrite	1	0	0	0	0	0	0	0	0	0
MemRead	1			1						
MemWrite	0	0	0	0	0	1	0	0	0	0
PCWrite	1	0	0	0	0	0	0	0	0	1
PCWCond									1	
PCSource	00								01	10
ALUsrcA	0	0	1	1	1	1	1	1	1	
ALUsrcB	01	11	10	10	10	10	00	00	00	00
MemToReg					1			0		
RegDst					0			1		
IoD	0			1	1	1				
ALUop	00	00	00				10		01	

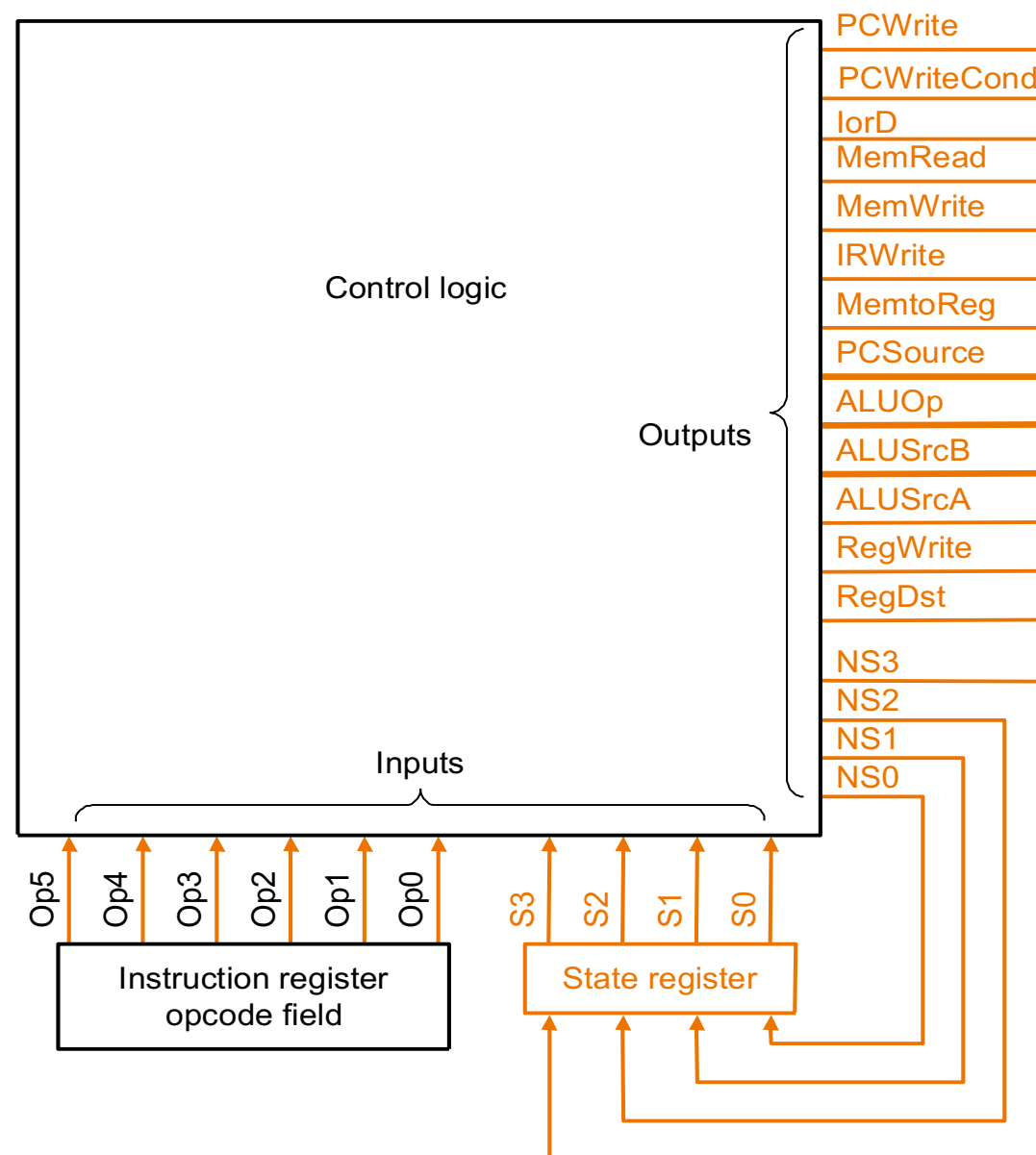
# Diseño de una computadora

5. Construir la **unidad de control** que implemente el comportamiento necesario



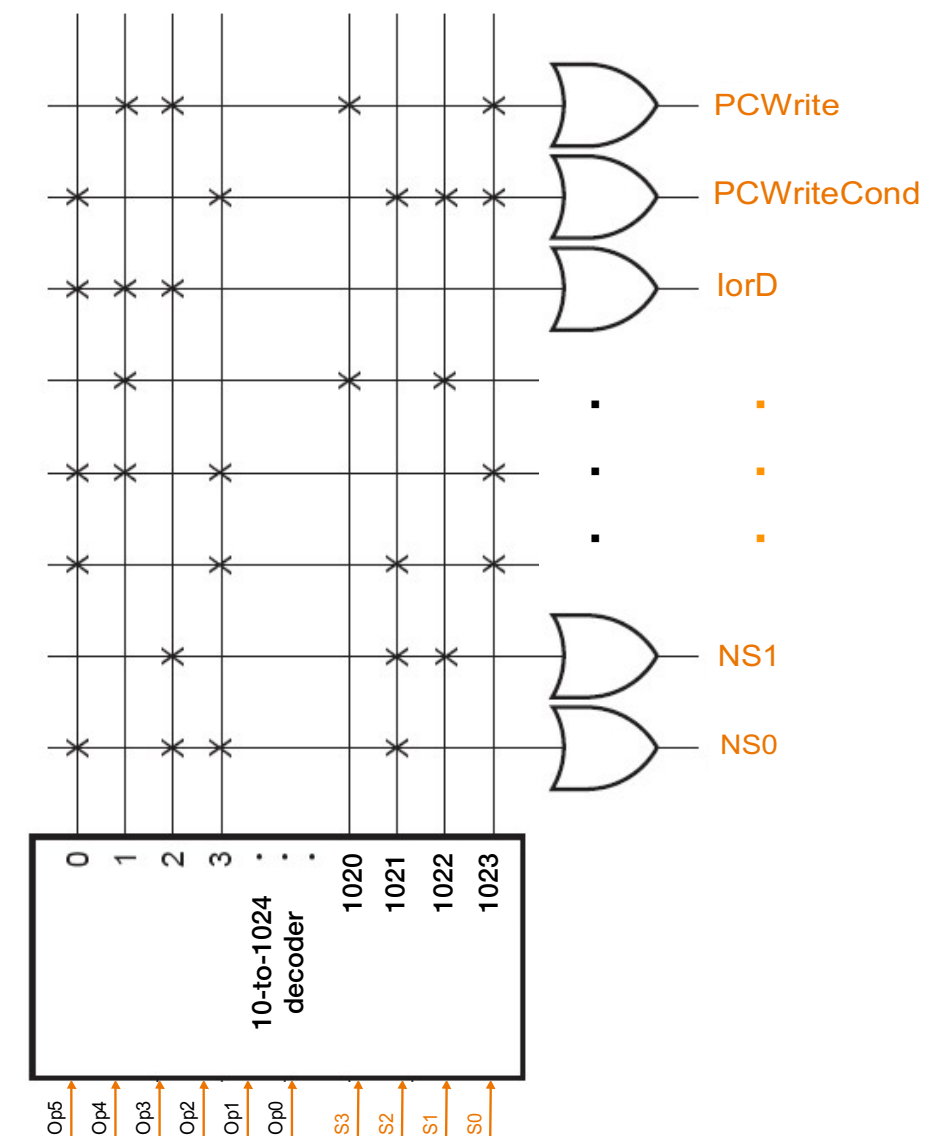
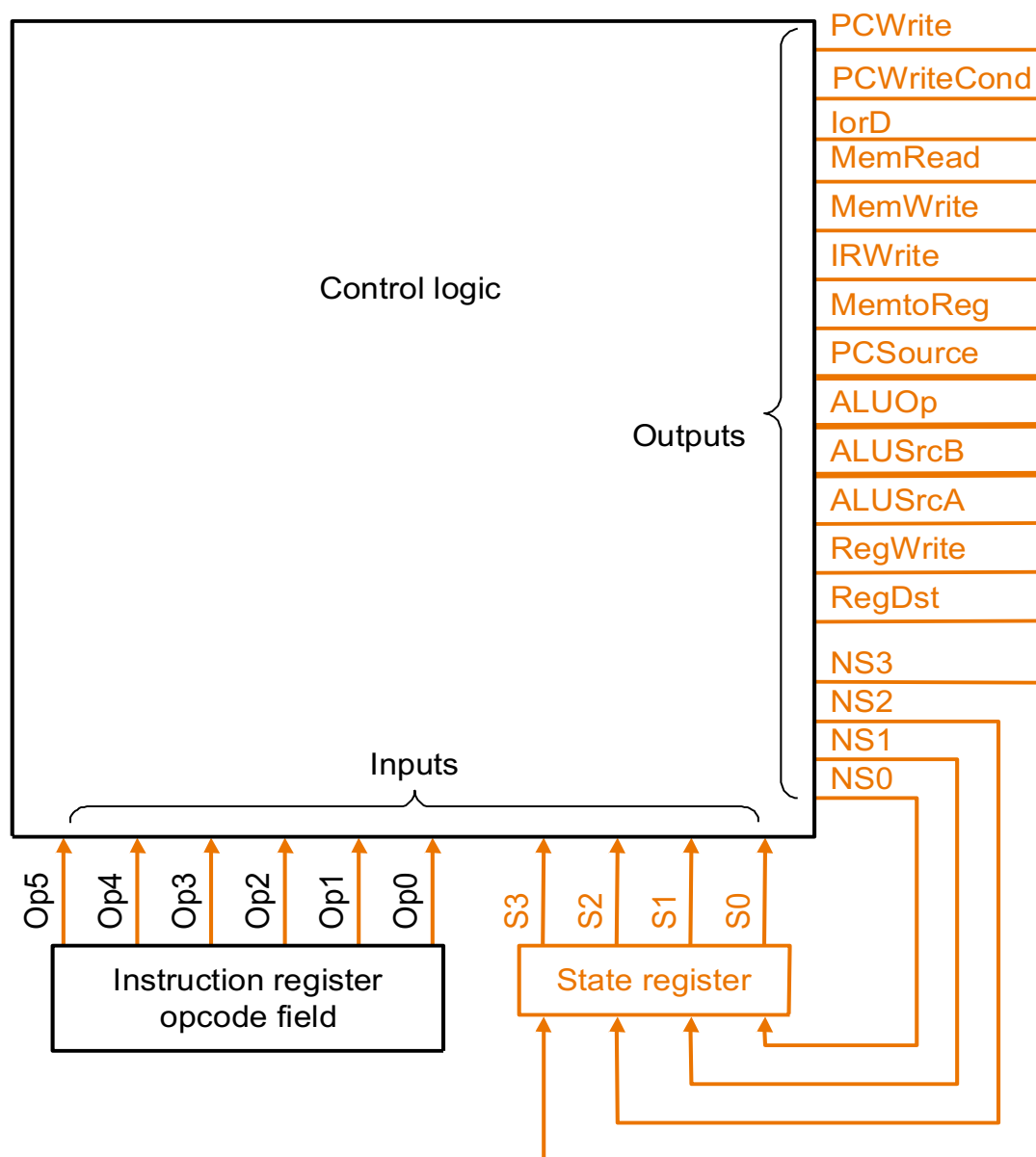
# Diseño de una computadora

5. Construir la **unidad de control** que implemente el comportamiento necesario



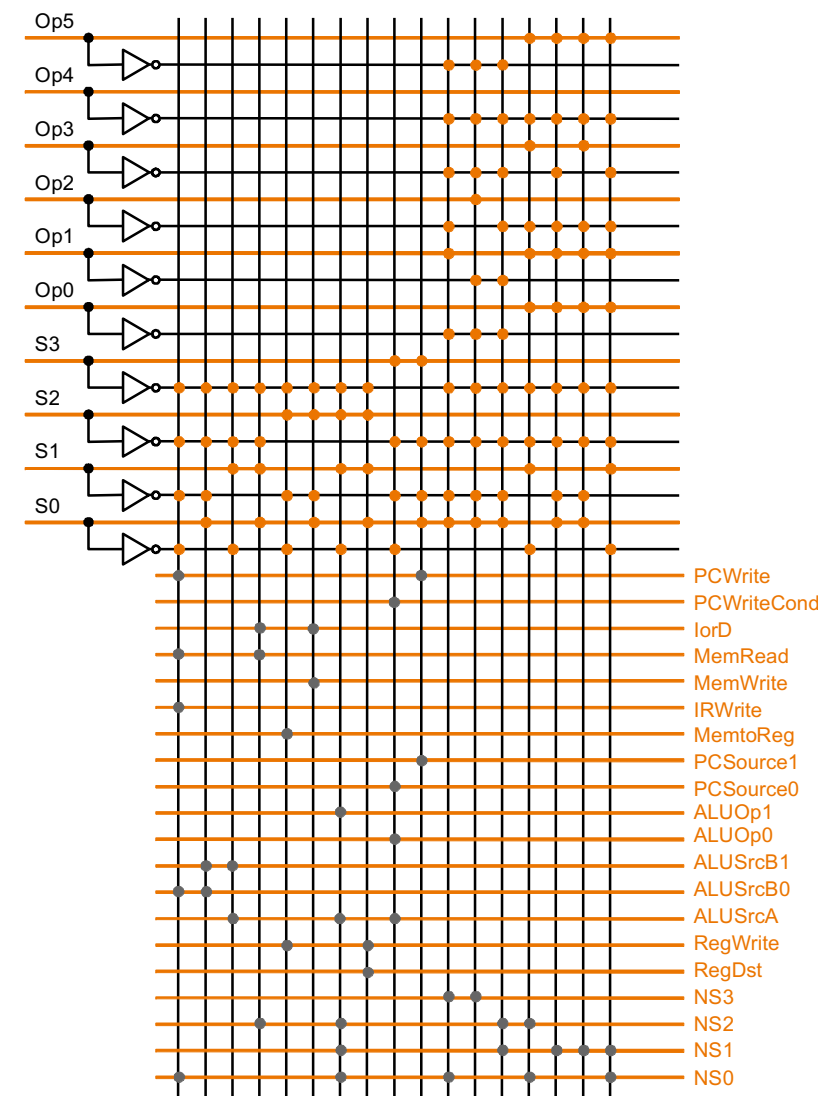
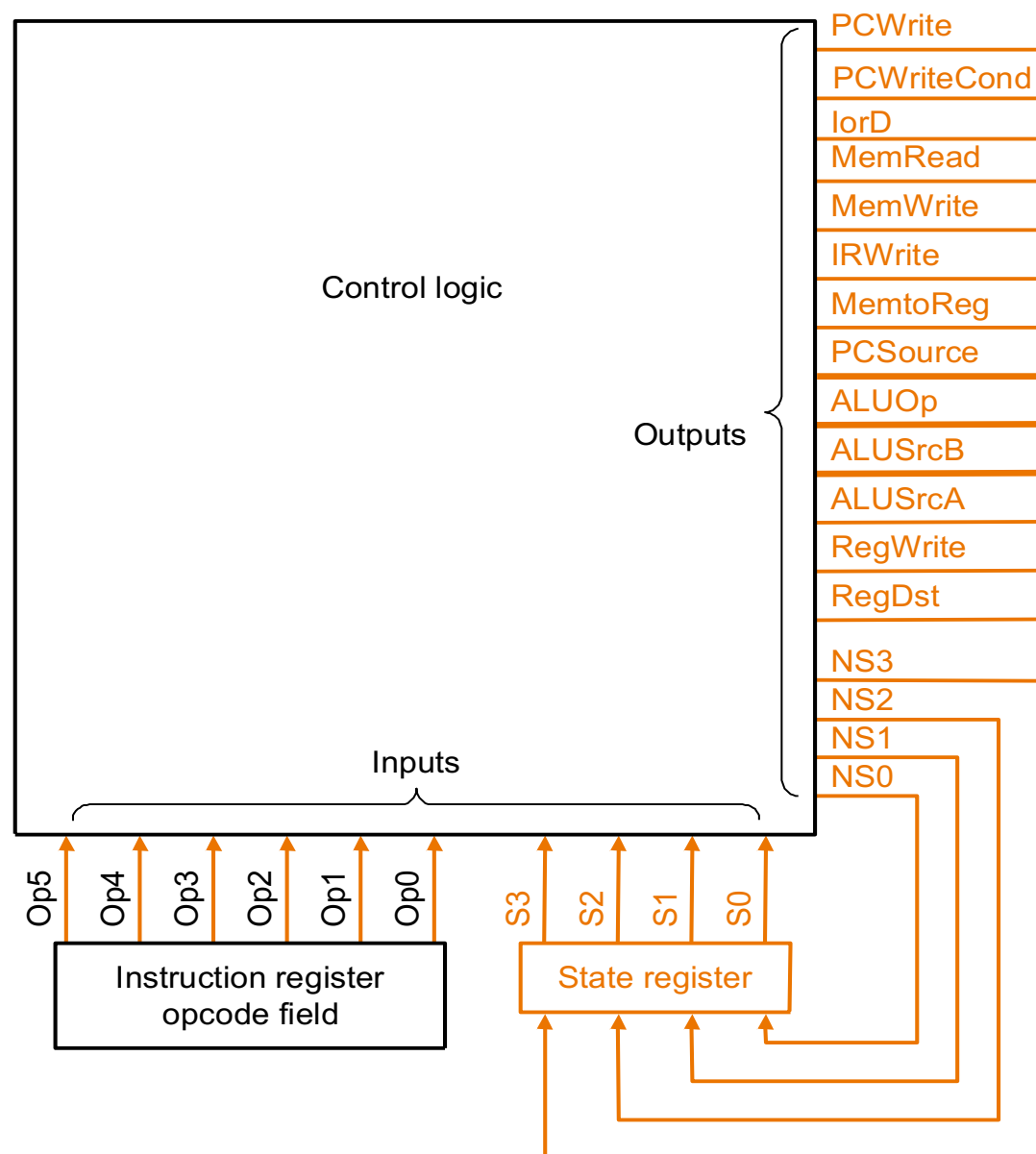
# Diseño de una computadora

## Implementación con ROM



# Diseño de una computadora

## Implementación con PLA



# Diseño de una computadora

## Comparación en tamaño de las implementaciones

### ROM

- ➤ 10 bits de entrada
- ➤ 20 bits de salida
- ➤ 1024 palabras de 20 bits
- ➤  $1024 \times 20 = 20$  Kbits ROM
  
- ➤ La enorme mayoría de las combinaciones de entrada no son utilizadas.

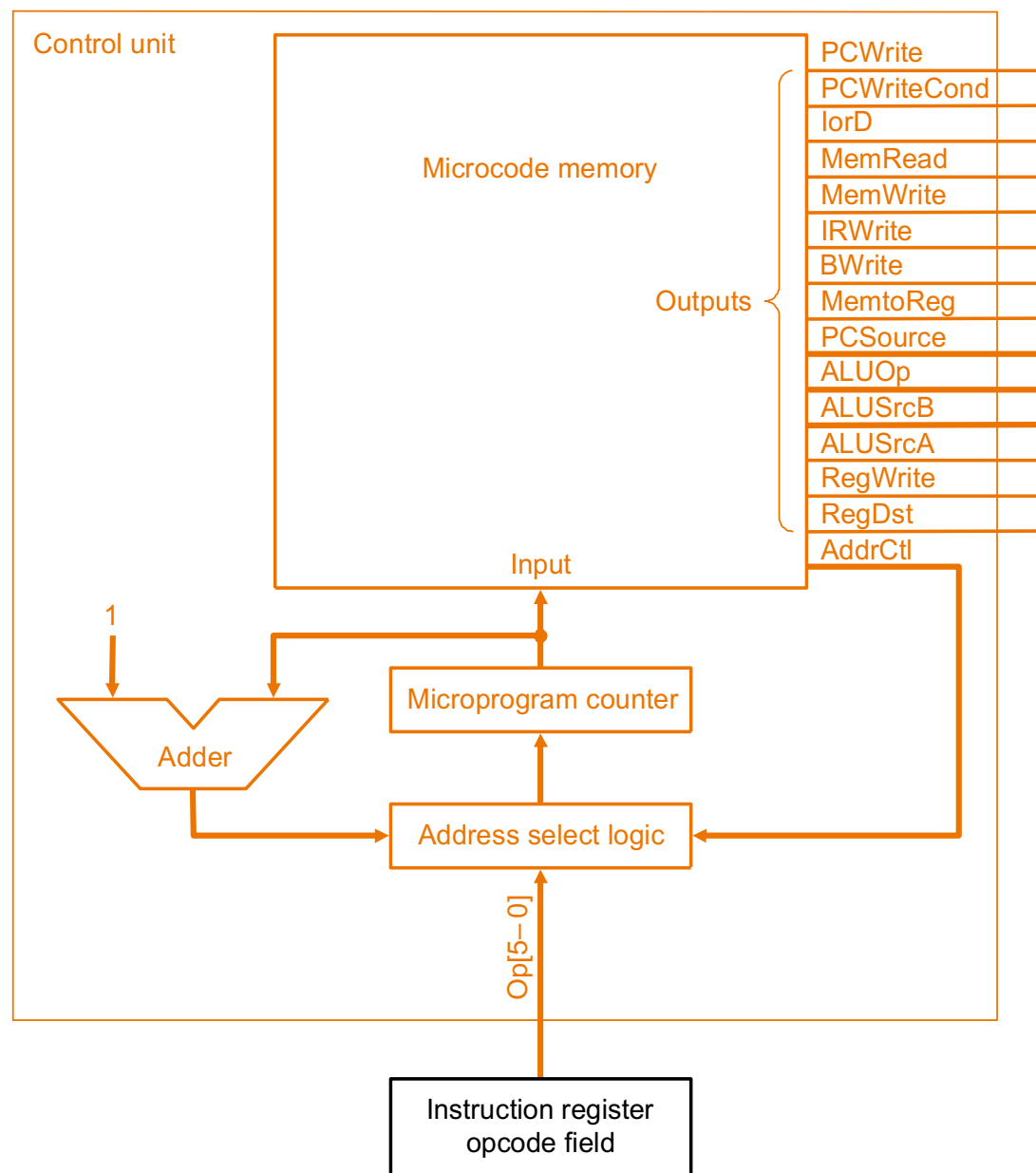
### PLA

- ➤ 10 bits de entrada
- ➤ 17 bits de salida
- ➤  $10 \times 17 + 20 \times 17 = 460$  PLA cells
  
- ➤ 17 son las combinaciones utilizadas realmente.



# Diseño de una computadora

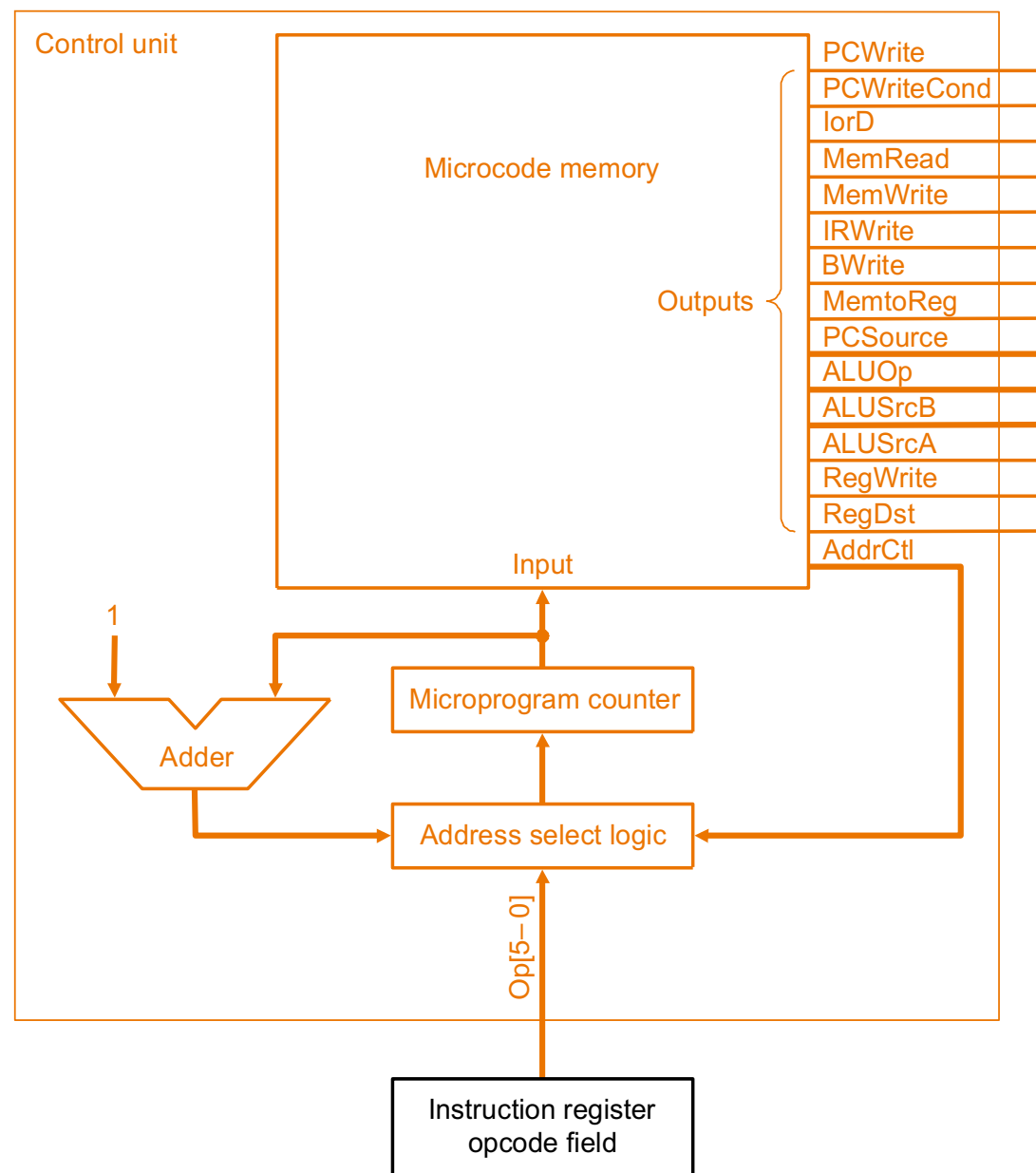
## Microprogramación



- ➤ Metodología apropiada para ISA CISC que tienen cientos de instrucciones y modos
- ➤ Las señales se especifican como instrucciones
- ➤ Las instrucciones tienen un formato determinado por campos
- ➤ Cada campo tiene asociado un conjunto de señales
- ➤ Cada valor del campo implica valores para dichas señales

# Diseño de una computadora

## Microprogramación



- AddrCtl abstrae las 4 señales (NS0 — NS3) que determinan el número de estado de llegada
- Los estados se utilizan para seleccionar la dirección de una ROM en la que se encuentra la microinstrucción

State number	Address-control action	Value of AddrCtl
0	Use incremented state	3
1	Use dispatch ROM 1	1
2	Use dispatch ROM 2	2
3	Use incremented state	3
4	Replace state number by 0	0
5	Replace state number by 0	0
6	Use incremented state	3
7	Replace state number by 0	0
8	Replace state number by 0	0
9	Replace state number by 0	0

Dispatch ROM 1			
OP	Name	Value	state
000000	R-type	Rformat1	0110
000010	j	JUMP1	1001
000100	beq	BEQ1	1000
100011	lw	Mem1	0010
101011	sw	Mem1	0010

Dispatch ROM 2			
OP	Name	Value	state
100011	lw	LW2	0011
101011	sw	SW2	0101

# Diseño de una computadora

## Microprogramación

- ➤ Las señales son agrupadas por su rol en la determinación del camino de datos
- ➤ La agrupación se establece a partir de ponerle nombre a los 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
Memoria	Especifica Lectura/Grabación. En lectura el registro
PCWriteCtrl	Especifica la grabación del PC
Secuencia	Determina como elegir la proxima microinstrucción

# Diseño de una computadora

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.
	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.
Register control	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.

# Diseño de una computadora

## Microprogramación

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