



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Organización del Computador 1

Microprogramación

Dr. Marcelo Risk

4 de octubre de 2022

Índice

Introducción

Elementos para el diseño de una CPU

Pasos del diseño de una CPU

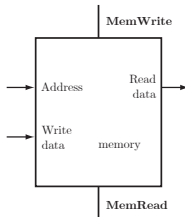
Consideraciones en el diseño de una CPU

- ▶ Una única ALU, una única Memoria, un único Banco de Registros.
- ▶ ¿Problemas con el uso de recursos?
 - ▶ 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 más tareas diferentes.

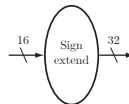
Recursos disponibles



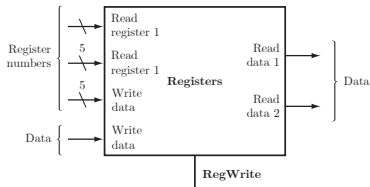
b. Program counter



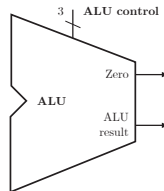
a. memory unit



b. Sign-extension unit



a. Registers



b. ALU

Diseño: pasos necesarios

1. Analizar el conjunto de instrucciones para determinar los requerimientos del camino de datos.
2. Seleccionar los componentes.
3. Construir el camino de datos según los requerimientos.
4. Analizar la implementación de cada instrucción para determinar las señales de control necesarias.
5. Construir el control.

Etapas de ejecución de una instrucción

1. Fetch de la instrucción: **IF** (Fetch)
2. Decodificación (y lectura de registros: **ID** (Decode)
3. Ejecución ó cálculo de dirección de memoria: **EX** (Execution)
4. Acceso a datos en memoria: **MEM**
5. Escritura en registros: **WB** (Write Back)

Formato de instrucción MIPS

Tipo R

- Aritméticas



Tipo I

- Transferencia, salto
- Operaciones con operando inmediato



Tipo J

- Saltos



Formato de instrucción MIPS

- ADD y SUB

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



- LOAD and STORE

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



- BRANCH

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



- JUMP

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



Register Transfer Language

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

Ejemplo Paso 1: Tipo R (add, sub...)

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

- $R[rd] = R[rs] \text{ op } R[rt]$ 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: $R[RD] \leftarrow ALUout$

Ejemplo Paso 1: Branch

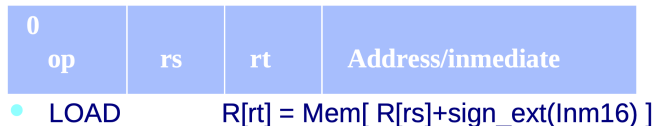
op	rs	rt	Address/immediate
----	----	----	-------------------

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

RTL

- ▶ t1: IR <- mem[PC], PC <- PC + 4
- ▶ t2: A <- R[RS], B <- R[RT]
- ▶ t3: ALUout <- PC + signextend(imm16) << 2
- ▶ t4: Comparar A y B, PC <- ALUout (si Z=1)

Ejemplo Paso 1: Load



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: $ALUout \leftarrow PC + \text{signextend}(\text{imm16}) \ll 2$
- ▶ t4: $MBR \leftarrow \text{mem}[ALUout]$
- ▶ t5: $R[RT] \leftarrow MBR$

Ejemplo Paso 1: Store

0			
op	rs	rt	Address/immediate

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

RTL

- ▶ $\text{t1: IR} \leftarrow \text{mem}[\text{PC}], \text{PC} \leftarrow \text{PC} + 4$
- ▶ $\text{t2: A} \leftarrow R[\text{RS}], \text{B} \leftarrow R[\text{RT}]$ (dato a escribir)
- ▶ $\text{t3: ALUout} \leftarrow \text{PC} + \text{signextend}(\text{imm16}) \ll 2$
- ▶ $\text{t4: mem}[\text{ALUout}] \leftarrow \text{B}$

Ejemplo Paso 1: Jump

0	
op	Target Address

- Jump: $PC<31:2> \leftarrow PC<31:28>, (target <25:0> \ll 2)$
 - Calcula la dirección concatenando los 26 bits del operando

RTL

- ▶ $t1: IR \leftarrow mem[PC], PC \leftarrow PC + 4$
- ▶ $t2: NADA!$
- ▶ $t3: PC<31:2> \leftarrow PC<31:28>, IR<25:0> \ll 2$

Resumen para cada etapa del ciclo de instrucción

Etapa	Tipo de instrucción	acciones
IF	todas	$IR \leftarrow \text{mem}[PC]$ $PC \leftarrow PC + 4$
ID	todas	$A \leftarrow R[RS]$ $B \leftarrow R[RT]$ $ALUout \leftarrow PC + (\text{inm16} \ll 2)$
EX	tipo R Load/Store Branch Jump	$ALUout \leftarrow A \text{ op } B$ $ALUout \leftarrow A + \text{signextend}(\text{inm16})$ if(A==B) then $PC \leftarrow ALUout$ $PC\langle 31:2 \rangle \leftarrow PC\langle 31:28 \rangle, IR\langle 25:0 \rangle \ll 2$
MEM	Load Store	$MBR \leftarrow \text{mem}[ALUout]$ $\text{mem}[ALUout] \leftarrow B$
WB	tipo R Load	$R[RD] \leftarrow ALUout$ $R[RT] \leftarrow MBR$

Etapas según el tipo de instrucción

- ▶ Tipo R: 4 etapas (IF, ID, EX y WB)
- ▶ Branch y Jump: 3 etapas (IF, ID y EX)
- ▶ Store: 4 etapas (IF, ID, EX y MEM)
- ▶ Load: 5 etapas (IF, ID, EX, MEM y WB)

Etapa 1: Fetch (IF)

RTL

► $IR \leftarrow \text{mem}[PC]$

► $PC \leftarrow PC + 4$

Etapa 2: Decode (ID)

RTL

- ▶ Opción lectura de registros:
 - ▶ $A \leftarrow R[IR[25:21]]$
 - ▶ $B \leftarrow R[IR[20:16]]$

- ▶ Opción cálculo de saltos:
 - ▶ $ALUout \leftarrow PC + \text{signextend}(IR[15:0]) \ll 2$

Etapa 3: Execute (EX)

RTL

- ▶ Opción aritmética/lógica:
 - ▶ $ALUout \leftarrow A \text{ op } B$
 - ▶ $ALUout \leftarrow A \text{ op signextend}(IR[15:0])$
- ▶ Opción dirección (Load ó Store):
 - ▶ $ALUout \leftarrow A + \text{signextend}(IR[15:0])$
- ▶ Salto condicional:
 - ▶ Si $A = B$, $PC \leftarrow ALUout$
- ▶ Jump:
 - ▶ $PC[31:28] \leftarrow PC \parallel IR[25:0] \ll 2$

Etapa 3: Memoria (MEM)

RTL sólo para Load y Store

- ▶ Opción lectura:
 - ▶ `MBR <- mem[ALUout]`

- ▶ Opción escritura:
 - ▶ `mem[ALUout] <- B`

Etapa 3: Escritura (WB)

RTL

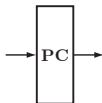
- ▶ Opción tipo R ó aritmética inmediata:
 - ▶ $R[IR[15:11]] \leftarrow ALUout$

- ▶ Opción escritura:
 - ▶ $R[IR[20:16]] \leftarrow MBR$

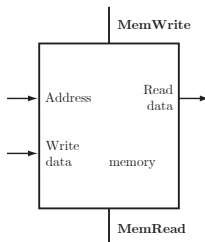
Paso 1: requerimientos de la ISA

- ▶ Memoria
 - ▶ Para instrucciones y datos
- ▶ Registros 32x32
 - ▶ Leer RS, leer RT
 - ▶ Escribir RT ó RD
- ▶ PC, MDR
- ▶ A, B para datos intermedios, ALUout (retiene salida de la ALU)
- ▶ Extensor de signo de 16 a 32 bits
- ▶ Sumar y restar registros y/o valores inmediatos
- ▶ Operaciones lógicas (and/or) con registros y/o valores inmediatos
- ▶ Sumar 4 al PC ó 4+inmediato extendido * 4

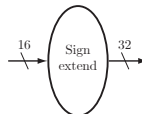
Paso 2: componentes del camino de datos



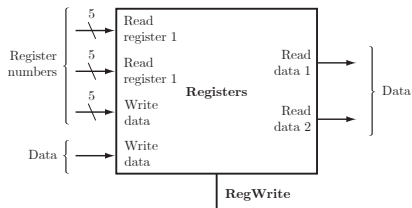
b. Program counter



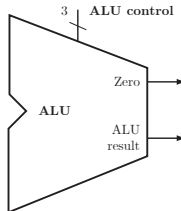
a. memory unit



b. Sign-extension unit



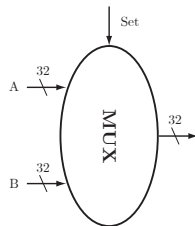
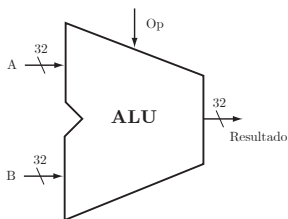
a. Registers



b. ALU

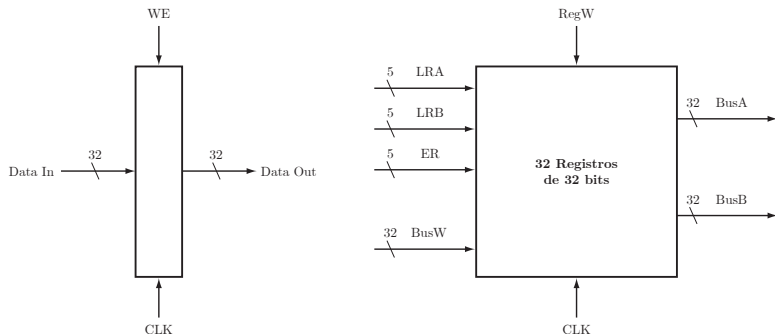
Paso 2: componentes del camino de datos

Elementos de lógica combinacional: ALU y multiplexor



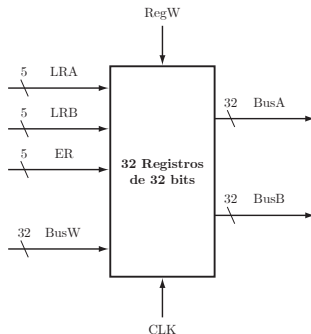
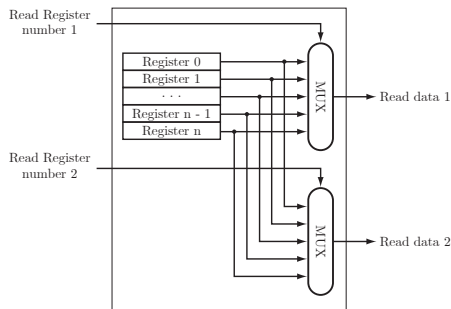
Paso 2: componentes del camino de datos

Elementos de almacenamiento: banco de registros



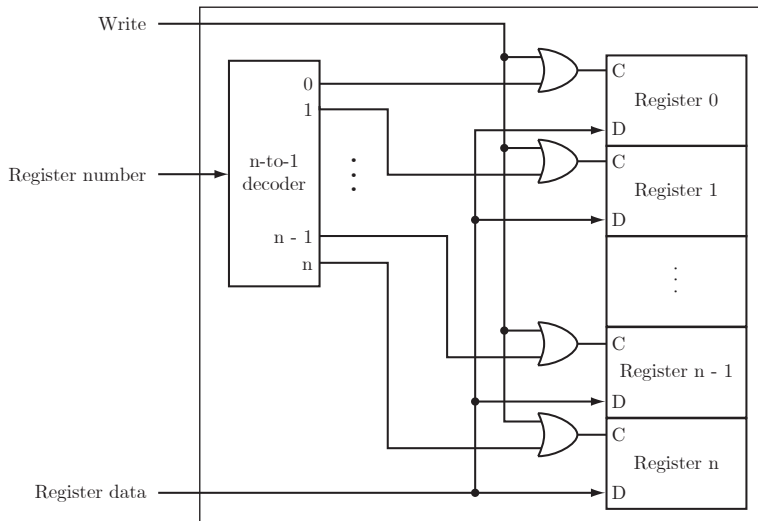
Paso 2: componentes del camino de datos

Elementos de almacenamiento: banco de registros con dos puertos de lectura



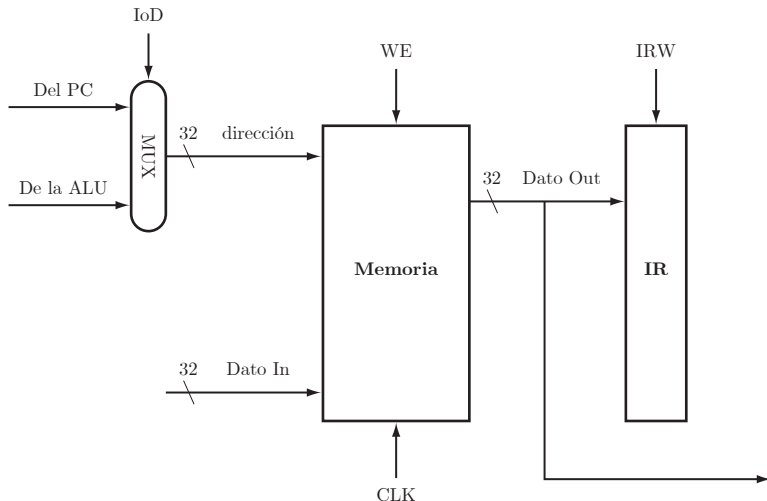
Paso 2: componentes del camino de datos

Elementos de almacenamiento: banco de registros con un puerto de escritura

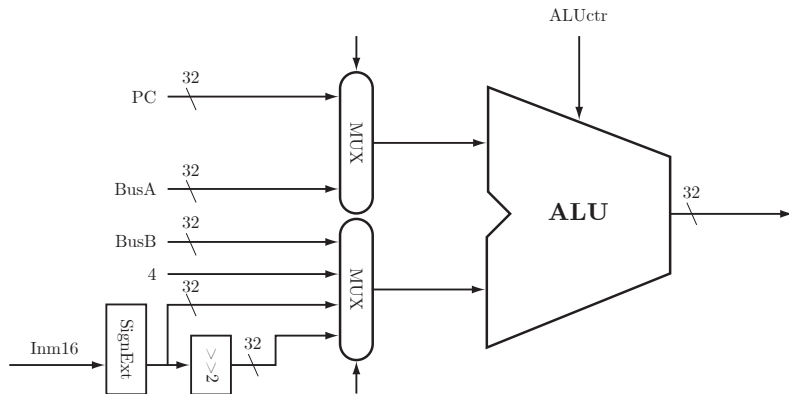


Paso 2: componentes del camino de datos

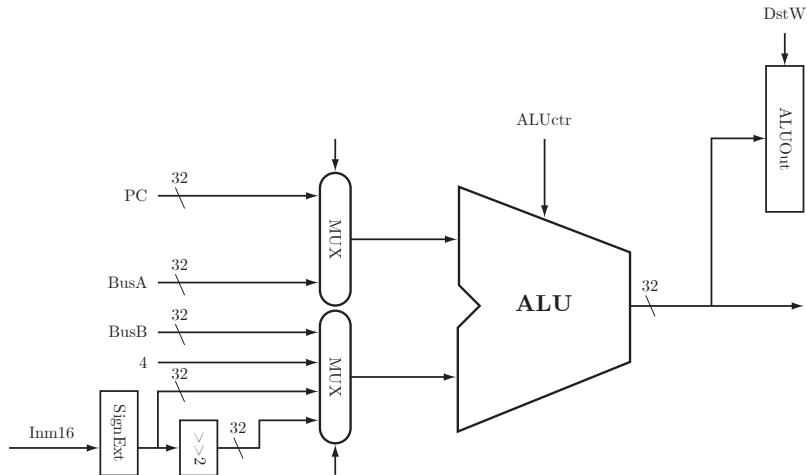
Una sola memoria para instrucciones y datos



Paso 3: reutilización de una sola ALU



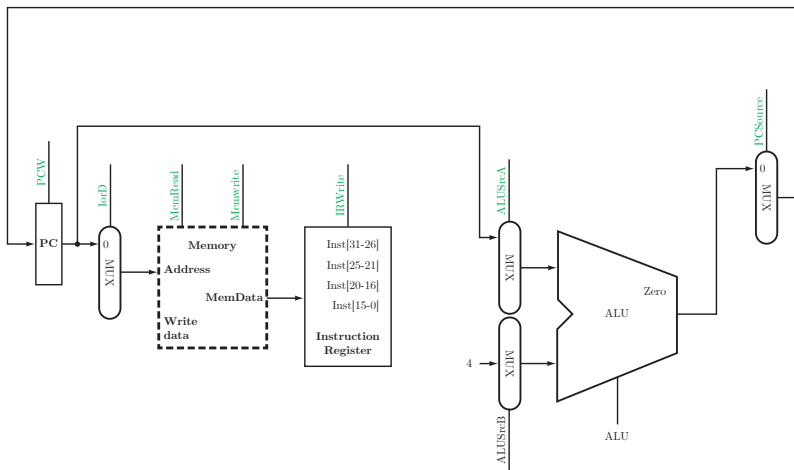
Paso 3: Registro ALUout



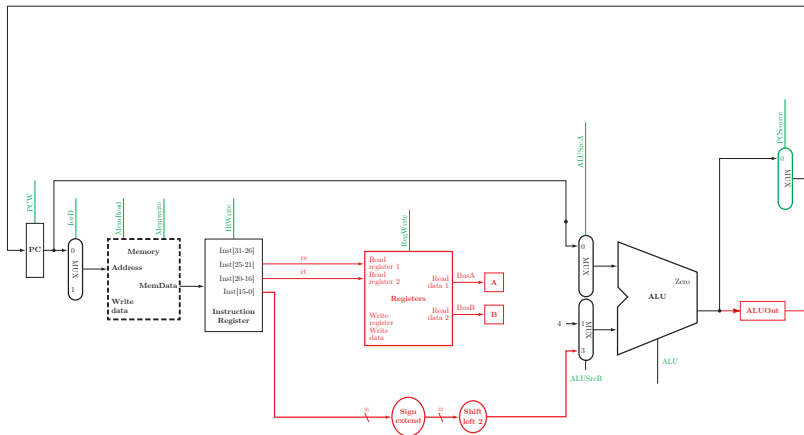
Paso 3: Fetch (IF)

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

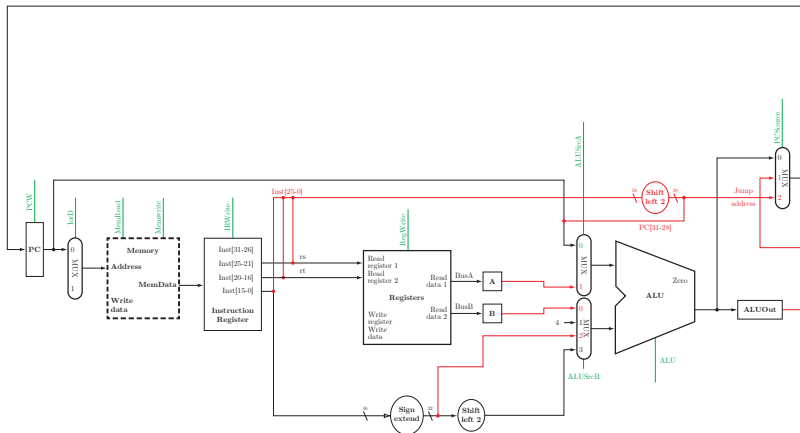
$\text{PC} \leftarrow \text{PC} + 4$ (código secuencial)



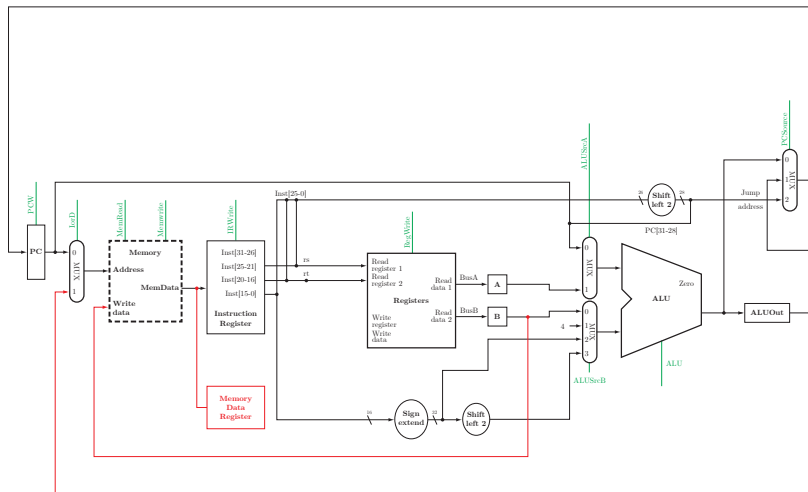
Paso 3: Decode (ID)



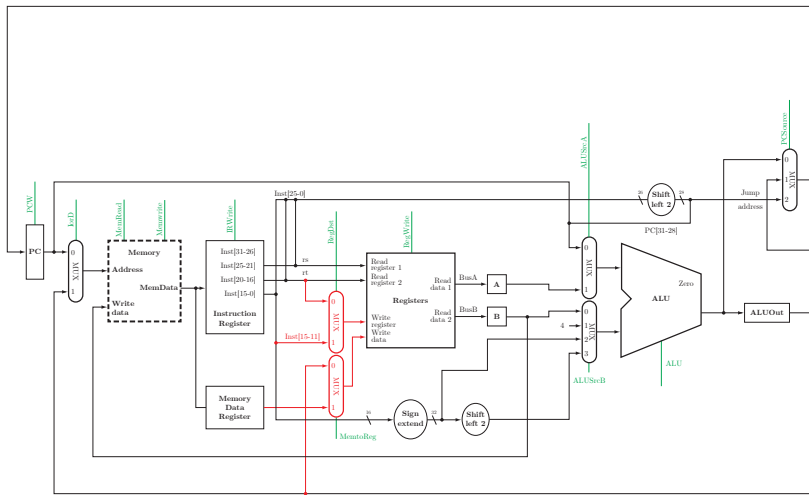
Paso 3: Data path (EX)



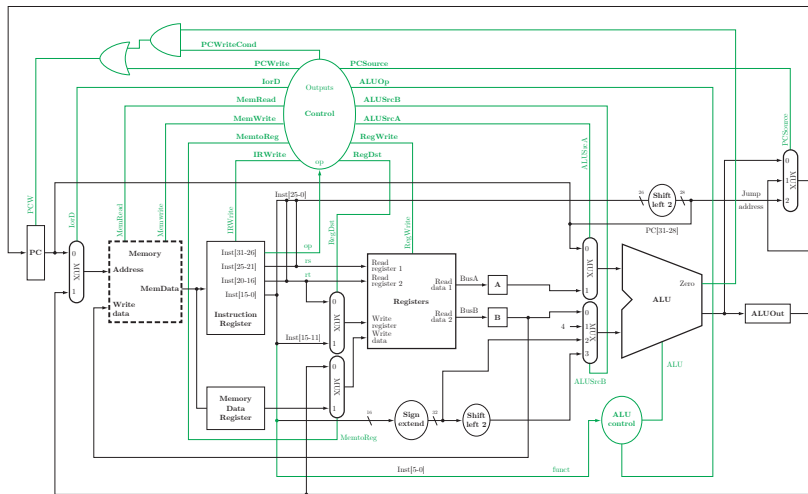
Paso 3: Data path (MEM)



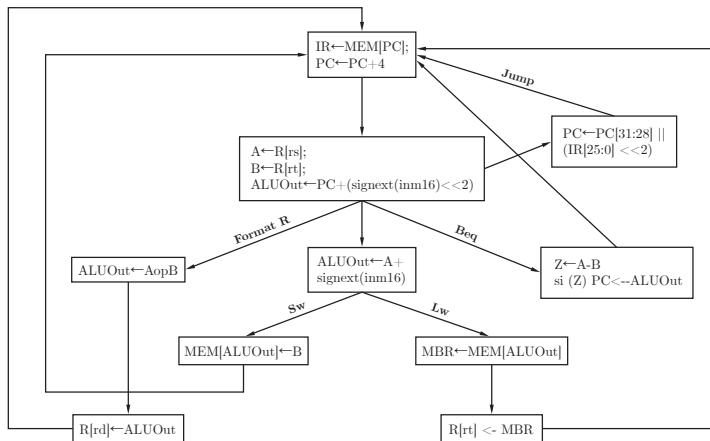
Paso 3: Data path (WB)



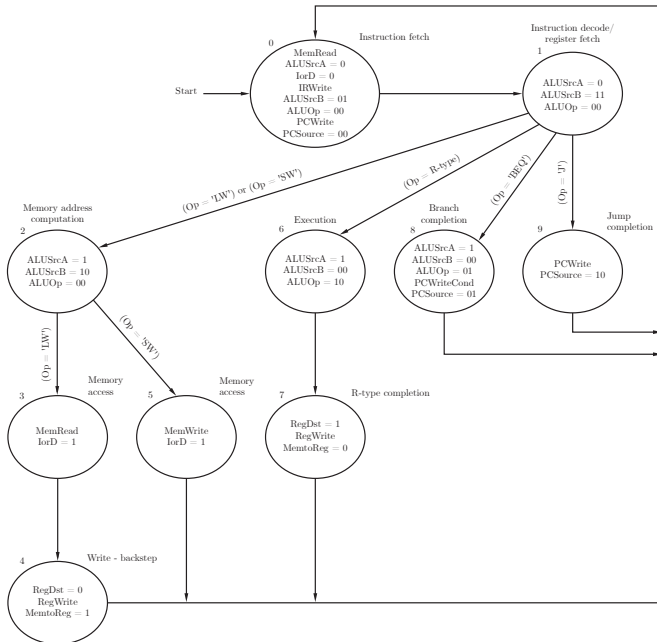
Señales de control



Grafo de estados

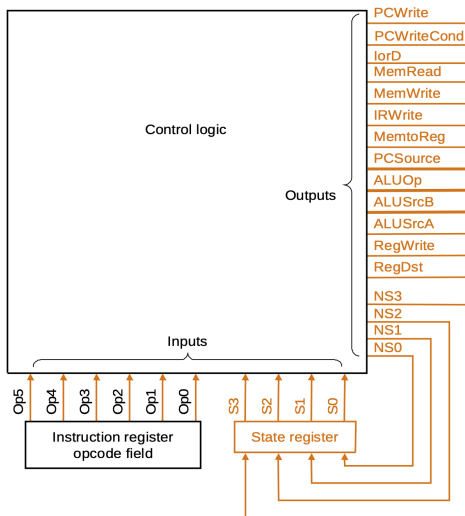


Control de señales



Máquina de estados finitos

Hardwired: con circuito combinacional, y tabla de verdad



Microprogramada con una memoria ROM

