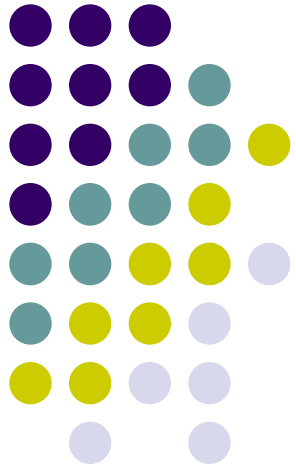
Arquitectura del MIPS

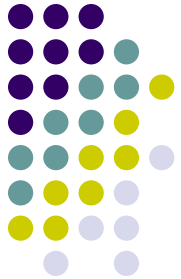
Arquitectura de Computadoras I

Prof. Dr. Martín Vázquez



LabSET
Laboratorio de Sistemas Embebidos Tandil

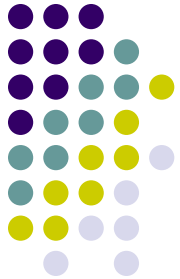
Antes de esta clase



Repasaron…

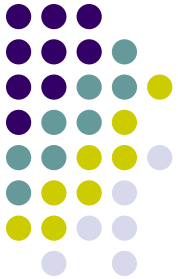
- Clase: “Introducción a Arquitectura de MIPS”
  - cátedra “Introducción a la Arquitectura de Sistemas”, 2do de Ing. Sistemas
- Conceptos requeridos
  - principios de diseño
  - juego de instrucciones
  - formato de memoria y registros internos
  - modos de direccionamiento

Procesador MIPS (repaso)

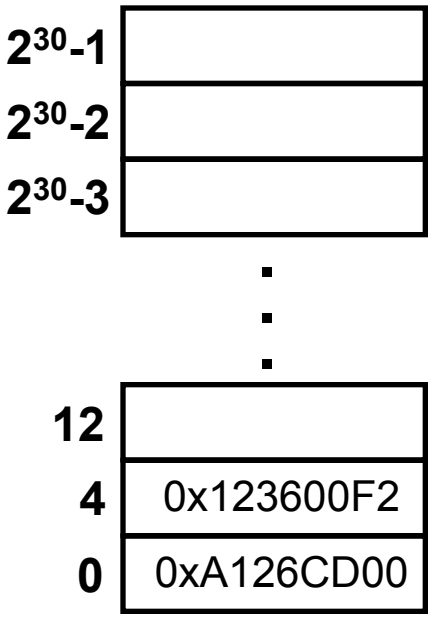


- Procesador MIPS (*Microprocessor without Interlocked Pipeline Stages*)
- 32 registros de uso general: $0..$31 (excepto el $0=0)
- datos, memoria e instrucciones de 32 bits
- memorias de datos y programa separadas (Harvard)
- memoria con 2^{30} palabras de 32 bits
- dos instrucciones para accesos a memorias
  - load (carga una palabra de memoria en registro)
  - store (carga el contenido de un registro a la memoria)

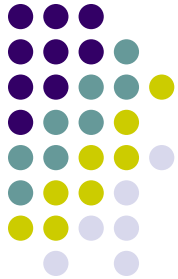
Acerca de Memoria (repaso)



- Existen instrucciones que transfieren datos desde memoria de datos y registros internos: load y store
- Cada byte posee su dirección
- Posee una memoria de 2^{30} palabras de 32 bits (4 bytes)

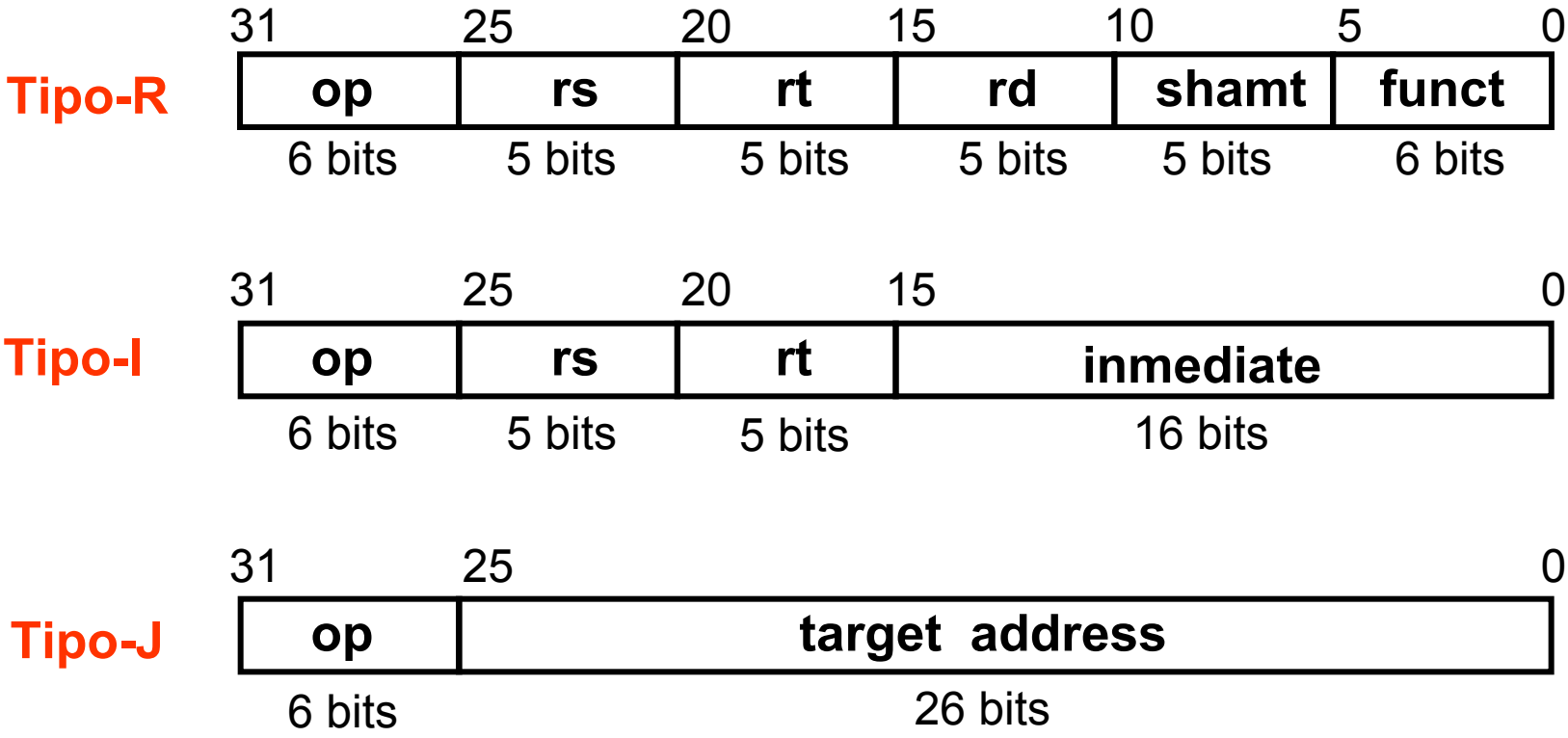


Repertorio de Instrucciones

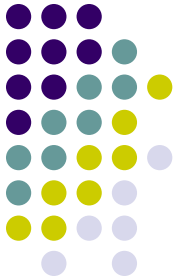


Conjunto de Instrucciones del MIPS (32 bits)

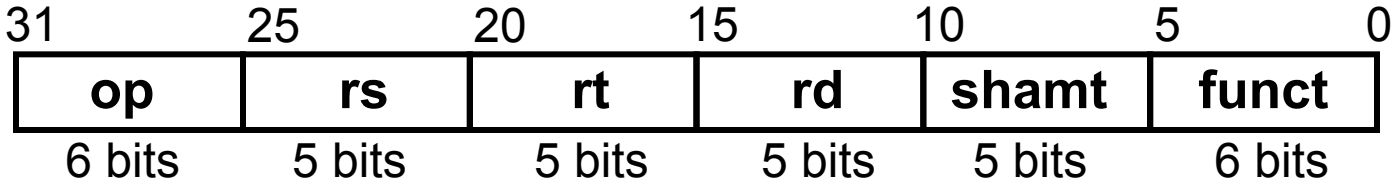
- Tres formatos de instrucciones



Repertorio de Instrucciones



- Tipo-R



op: código de operación

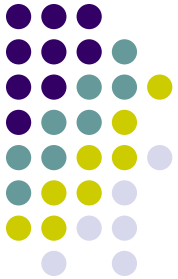
rd: identificador de registro destino

rt y **rs**: identificador de registros fuentes

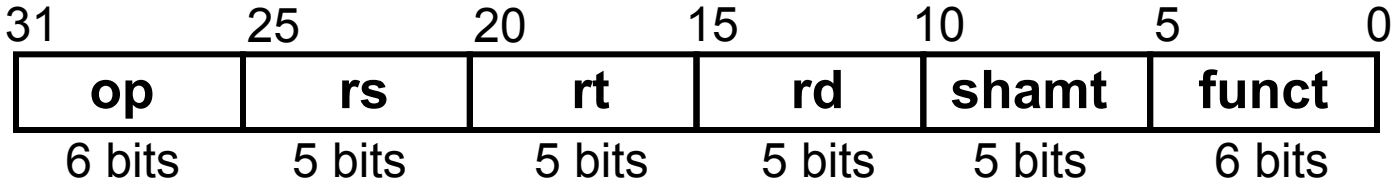
shamt: desplazamiento deseado

funct: selección de función asociada

Repertorio de Instrucciones



- Tipo-R



op: código de operación

rd: identificador de registro destino

rt y **rs**: identificador de registros fuentes

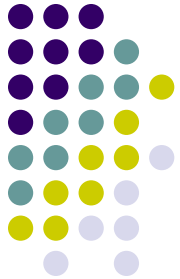
shamt: desplazamiento deseado

funct: selección de función asociada

Ejemplos: add, sub

```
add rd, rs, rt    # rd = rs+rt
sub rd, rs, rt    # rd = rs-rt
```

Repertorio de Instrucciones



- Tipo R - Ejemplos

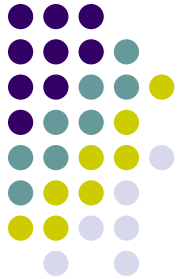
add $17, $12, $23 # $17= $12+$23

0	12	23	17	0	32
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

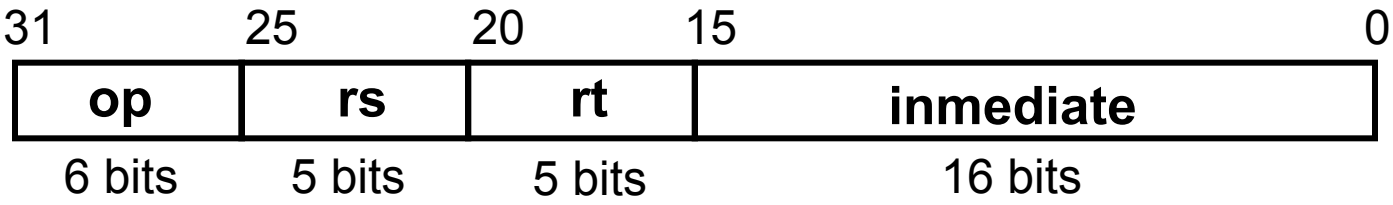
slt $1, $2, $3 # *if* $2<$3 *then* $1=1 *else* $1=0

0	2	3	1	0	42
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Repertorio de Instrucciones



- Tipo-I



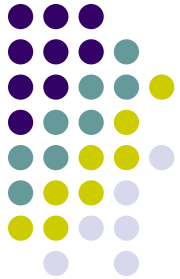
op: código de operación

rt: identificador de registro destino

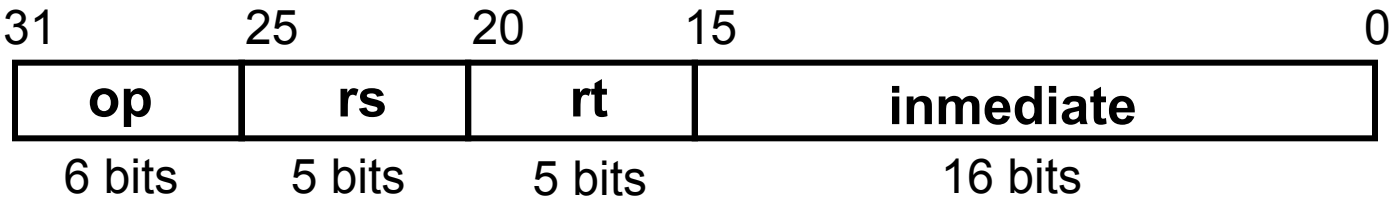
rs: identificador de registros fuente

inmediate: dato inmediato en 16 bits

Repertorio de Instrucciones



- Tipo-I



op: código de operación

rt: identificador de registro destino

rs: identificador de registros fuente

inmediate: dato inmediato en 16 bits

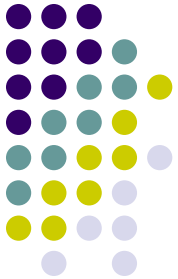
Ejemplos: addi, beq, lw

addi rt, rs, inm # rt = rs+inm

beq rt, rs, L # *if* rt==rs *then goto* L

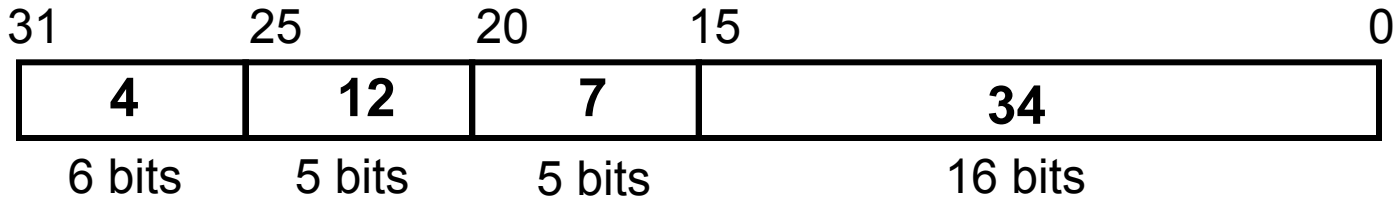
lw rt, inm(rs) # rt= mem[rs+inm]

Repertorio de Instrucciones

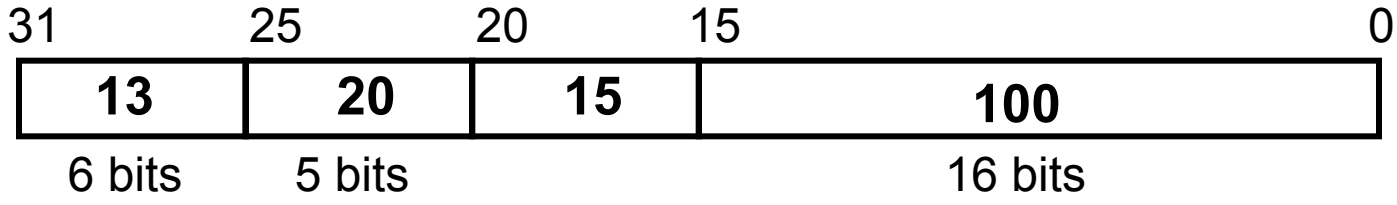


- Tipo I – Ejemplos (operaciones aritméticas-lógicas)

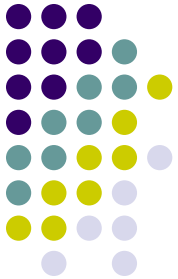
addi $7, $12, 34 # $7 = $12+34



ori $15, $20, 100 # $15 = $20|100

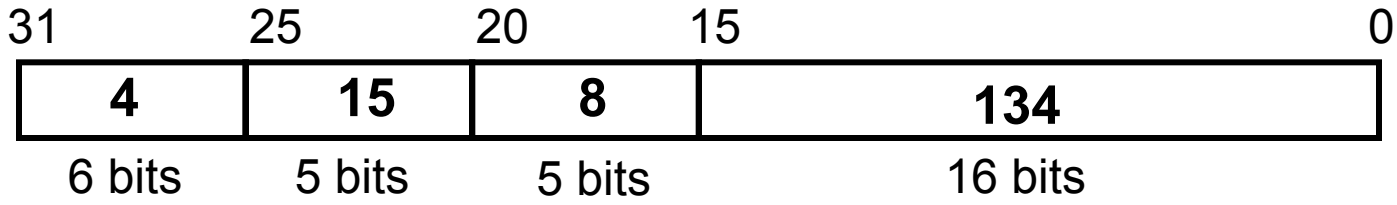


Repertorio de Instrucciones

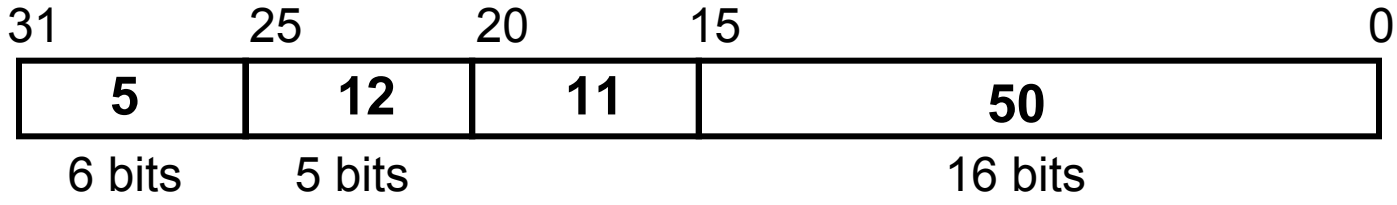


- Tipo I – Ejemplos (saltos condicionales)

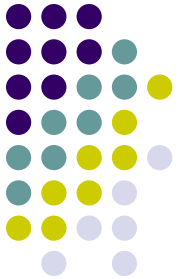
beq $8, $15, 134 # *if* $15==$8 *then* PC= PC+4+134*4



bne $11, $12, 50 # if $12<>$11 *then* PC= PC+4+50*4

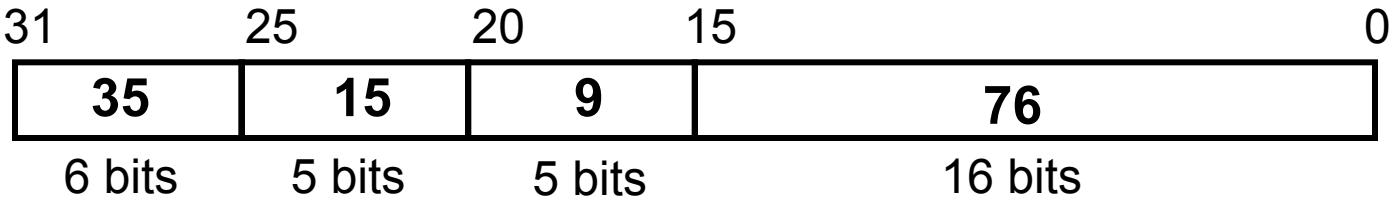


Repertorio de Instrucciones

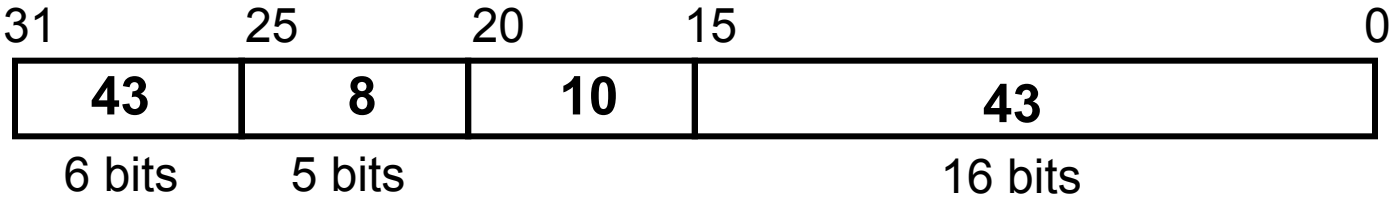


- Tipo I – Ejemplos (acceso a memoria)

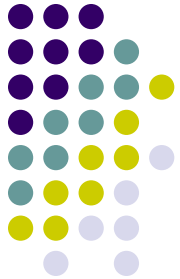
lw $9, 76($15) # $9 = memory[$15+76]



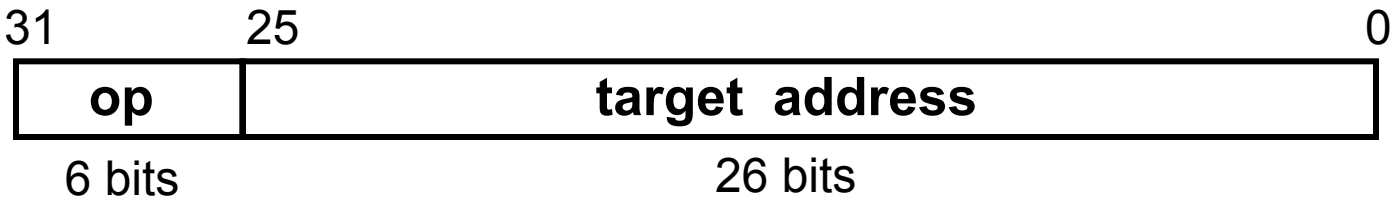
sw $10, 43($8) # memory[$8+43] = $10



Repertorio de Instrucciones



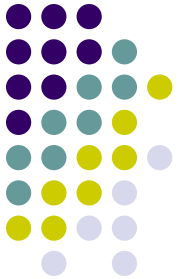
- Tipo-J



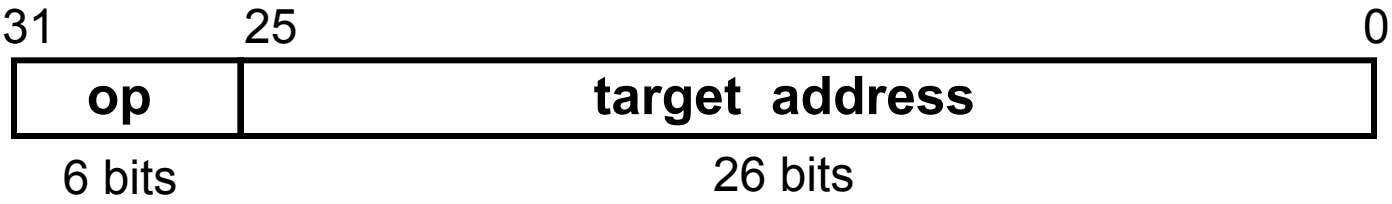
op: código de operación

target address: dirección destino de salto

Repertorio de Instrucciones



- Tipo-J



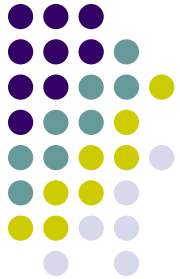
op: código de operación

target address: dirección destino de salto

Ejemplo: j

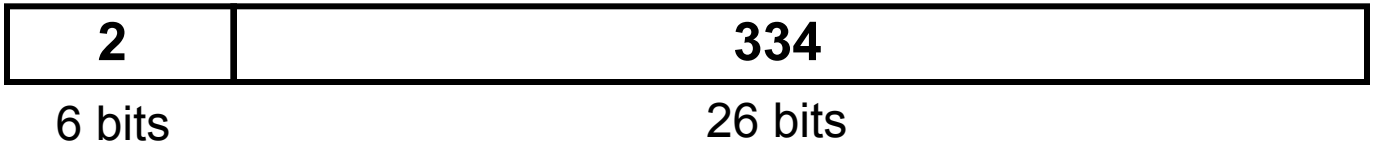
```
j L    # goto L
```

Repertorio de Instrucciones

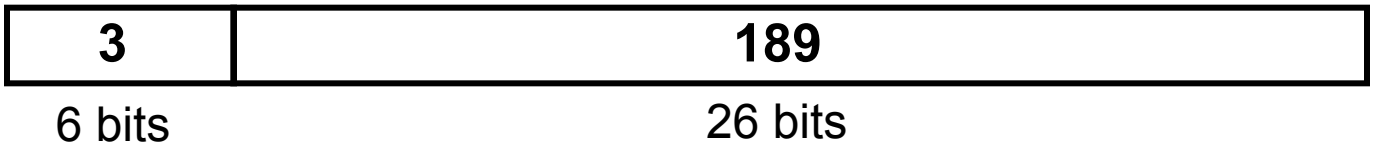


- Tipo J – Ejemplos (saltos incondicionales)

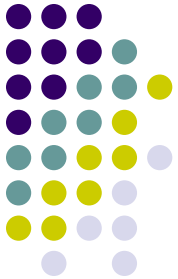
j 334 # PC = (PC+4)[31..28] & 334*4



jal 189 # PC = (PC+4)[31..28] & 189*4
$31=PC+4

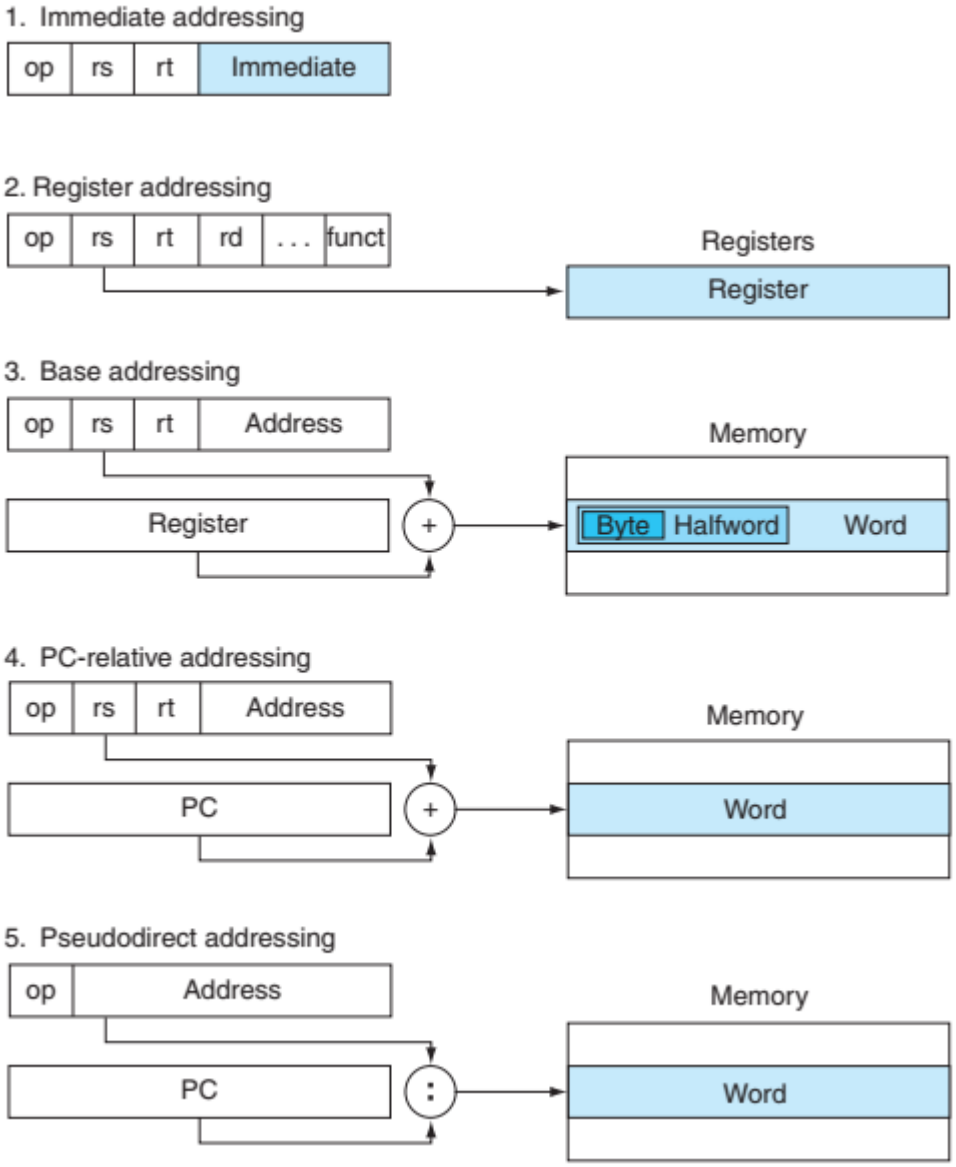


Modos de direccionamiento (repaso)

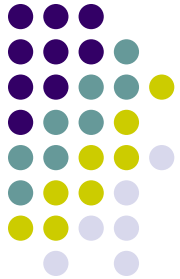


- 5 modos de direccionamiento
  - Mediante registros
    - Tipo-R. ej: add, sub, and…
  - Inmediato
    - Tipo-I. ej: addi, subi, andi…
  - Mediante registro base
    - Tipo-I. ej: lw y sw
  - Relativo a PC
    - Tipo-I. ej: beq, bne
  - Pseudo directo:
    - Tipo-J. ej: j, jal

Modos de direccionamiento (repaso)

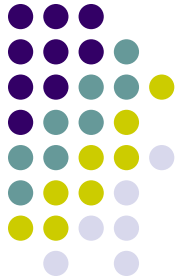


Diseño del microprocesador MIPS



- Microprocesador de 32 bits de un ciclo
  - También llamado *uniciclo* o combinacional
  - Cada instrucción se ejecuta en un ciclo de reloj
- Se diseñará para las siguientes instrucciones
  - Referencia a memorias: lw y sw
  - Aritméticas-lógicas de tipo-R: add, sub, and, slt, ...
  - Saltos: beq

Diseño del microprocesador MIPS



- Ejecución RTL (*Register Transfer Level*) de una instrucción

Ejemplos:

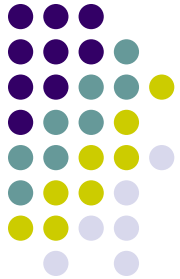
add $8, $7, $9

IR ← Mem[PC];
PC ← PC + 4
Reg[8] = Reg[7]+Reg[9]

lw $12, 132($19)

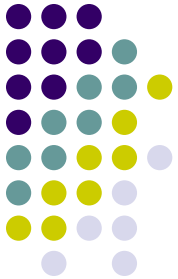
IR ← Mem[PC];
PC ← PC + 4
Reg[12] = Mem[Reg[19]+132]

Diseño del microprocesador MIPS

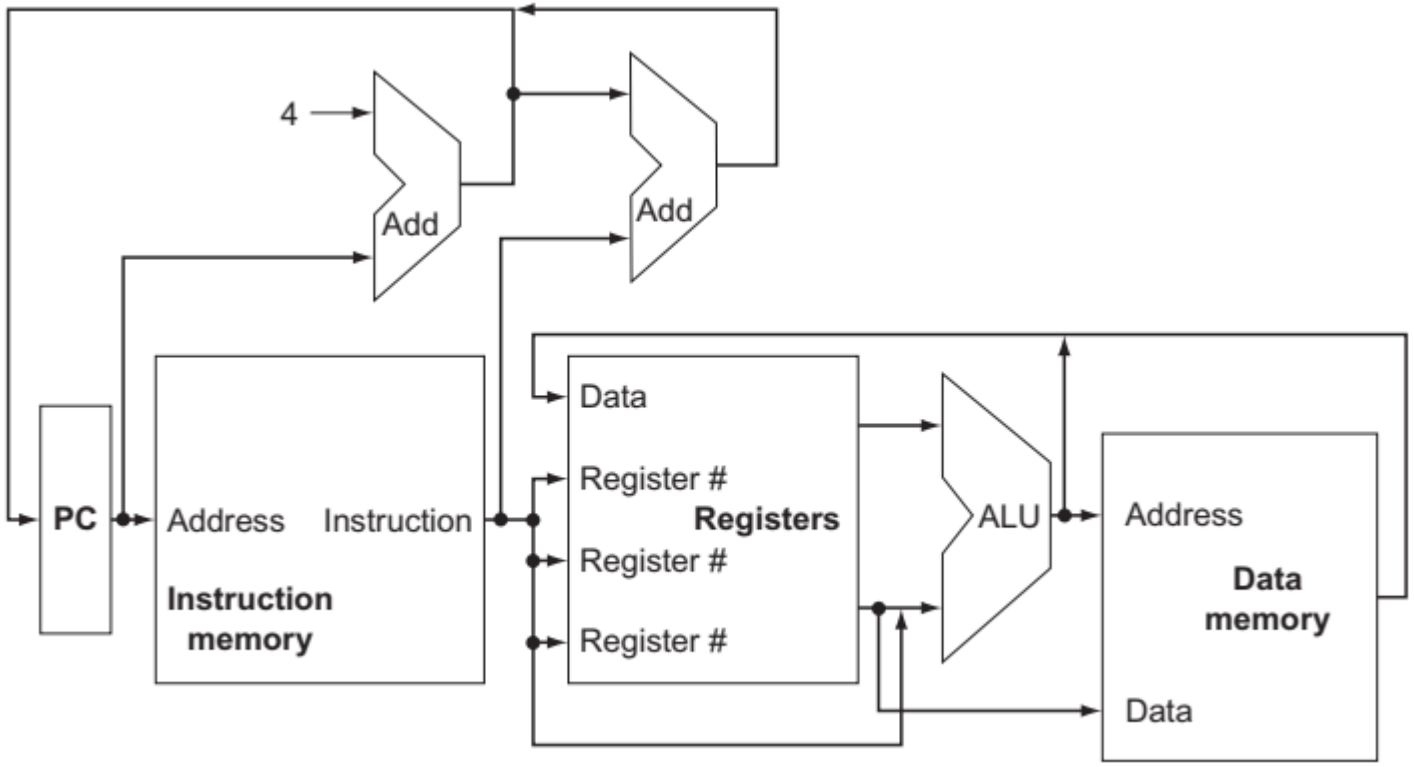


- Para todas las instrucciones los dos primeros pasos son los mismos:
  - Obtener la instrucción desde la memoria y actualizar el PC
  - Leer uno o dos registros
- El resto de los pasos depende de la instrucción a ejecutar
- Todas usan de alguna manera la ALU
  - Referencia a memoria: cálculo de dirección de acceso
  - Aritméticas-lógicas: según campo *funct* de la instrucción
  - Saltos: para determinar condición

Diseño del microprocesador MIPS

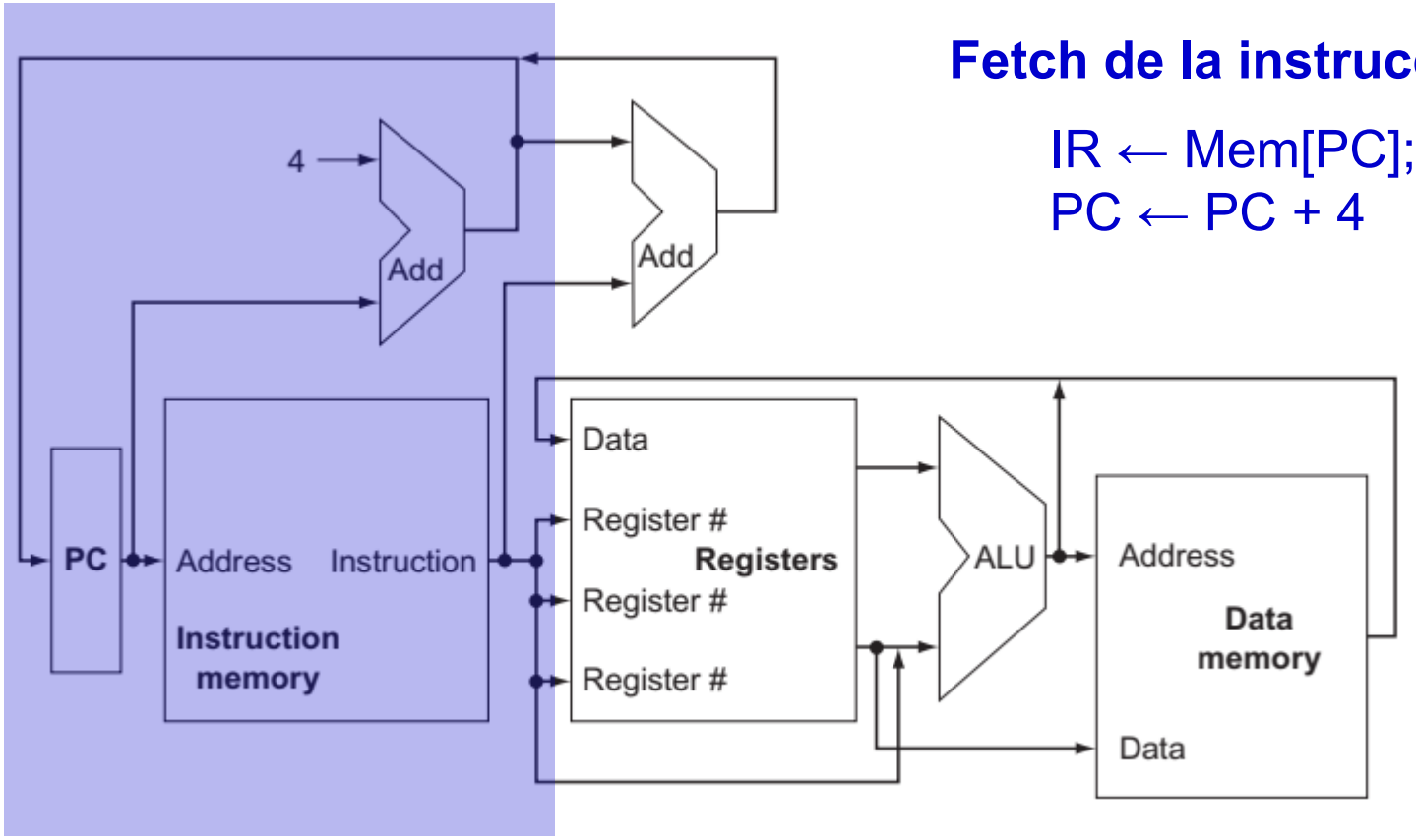


- Vista a muy alto nivel



Diseño del microprocesador MIPS

- Vista a muy alto nivel



Fetch de la instrucción

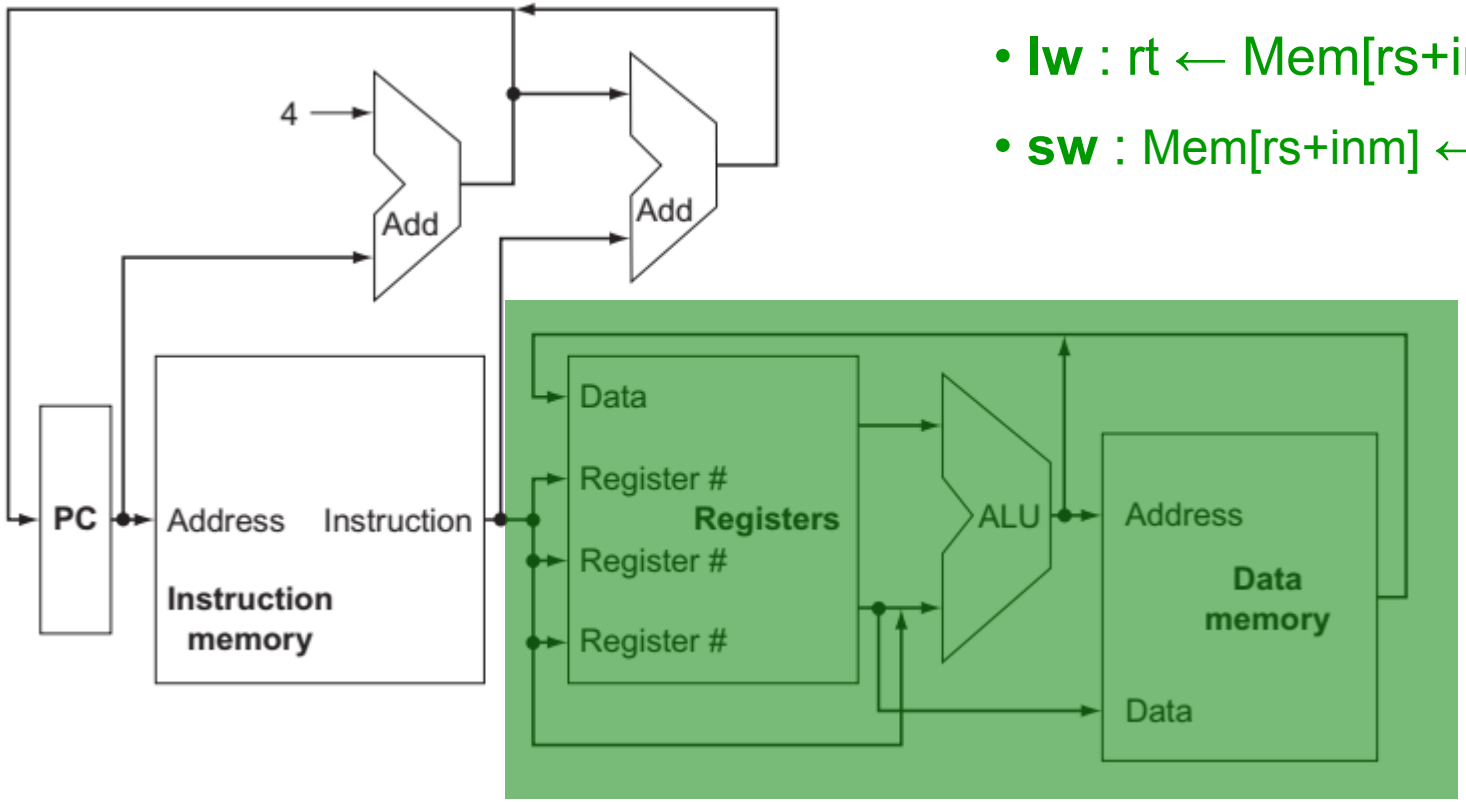
IR ← Mem[PC];
PC ← PC + 4

Diseño del microprocesador MIPS

- Vista a muy alto nivel

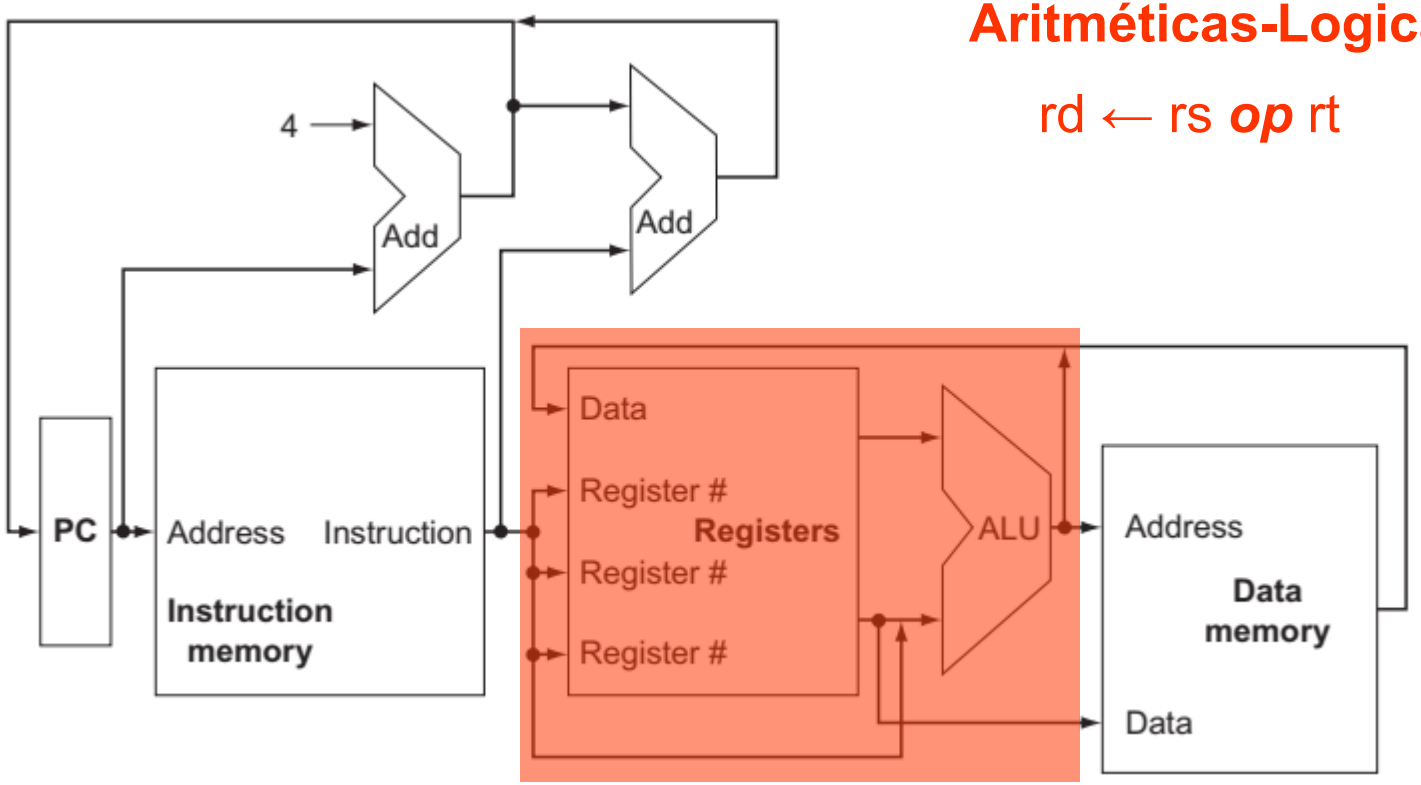
Referencia a Memoria

- **lw** : rt ← Mem[rs+inm]
- **sw** : Mem[rs+inm] ← rt



Diseño del microprocesador MIPS

- Vista a muy alto nivel



Aritméticas-Logicas

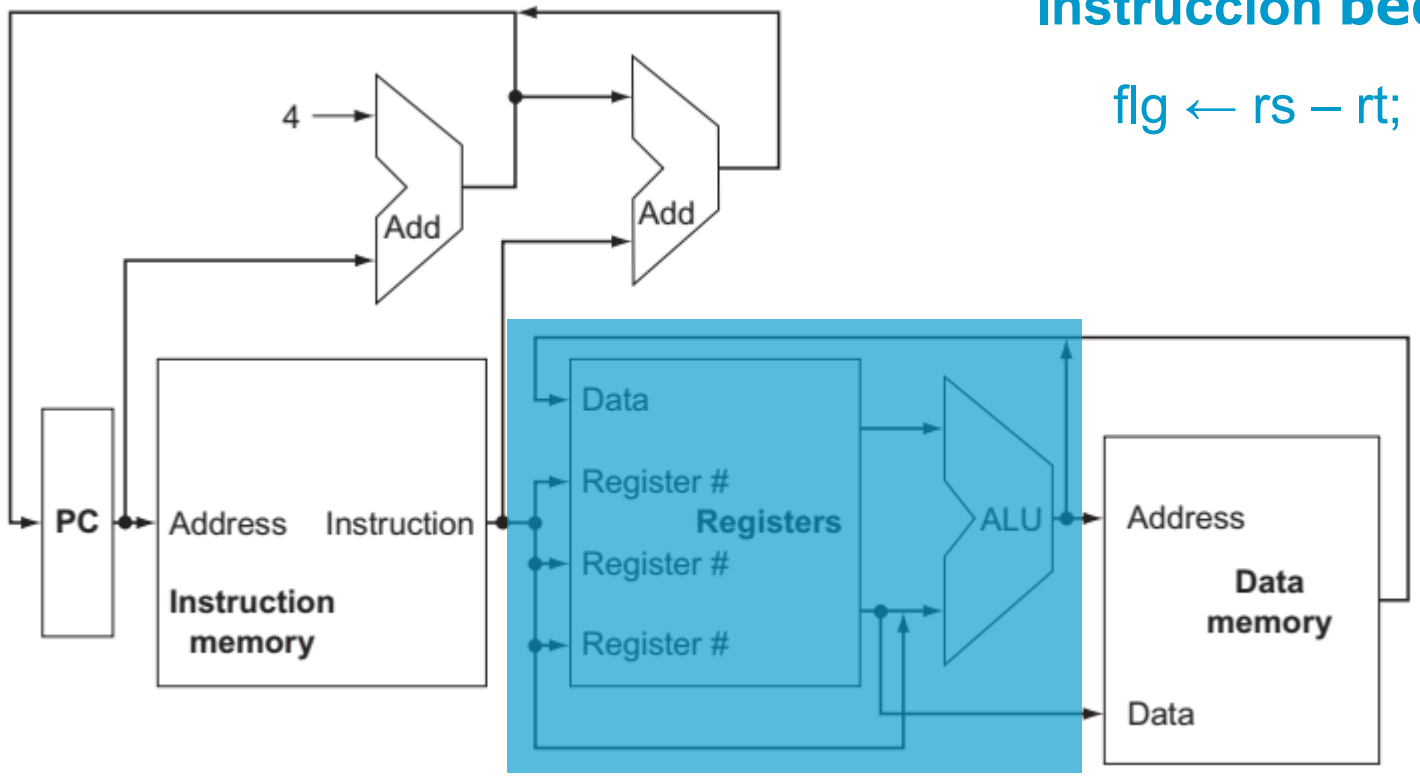
rd ← rs ***op*** rt

Diseño del microprocesador MIPS

- Vista a muy alto nivel

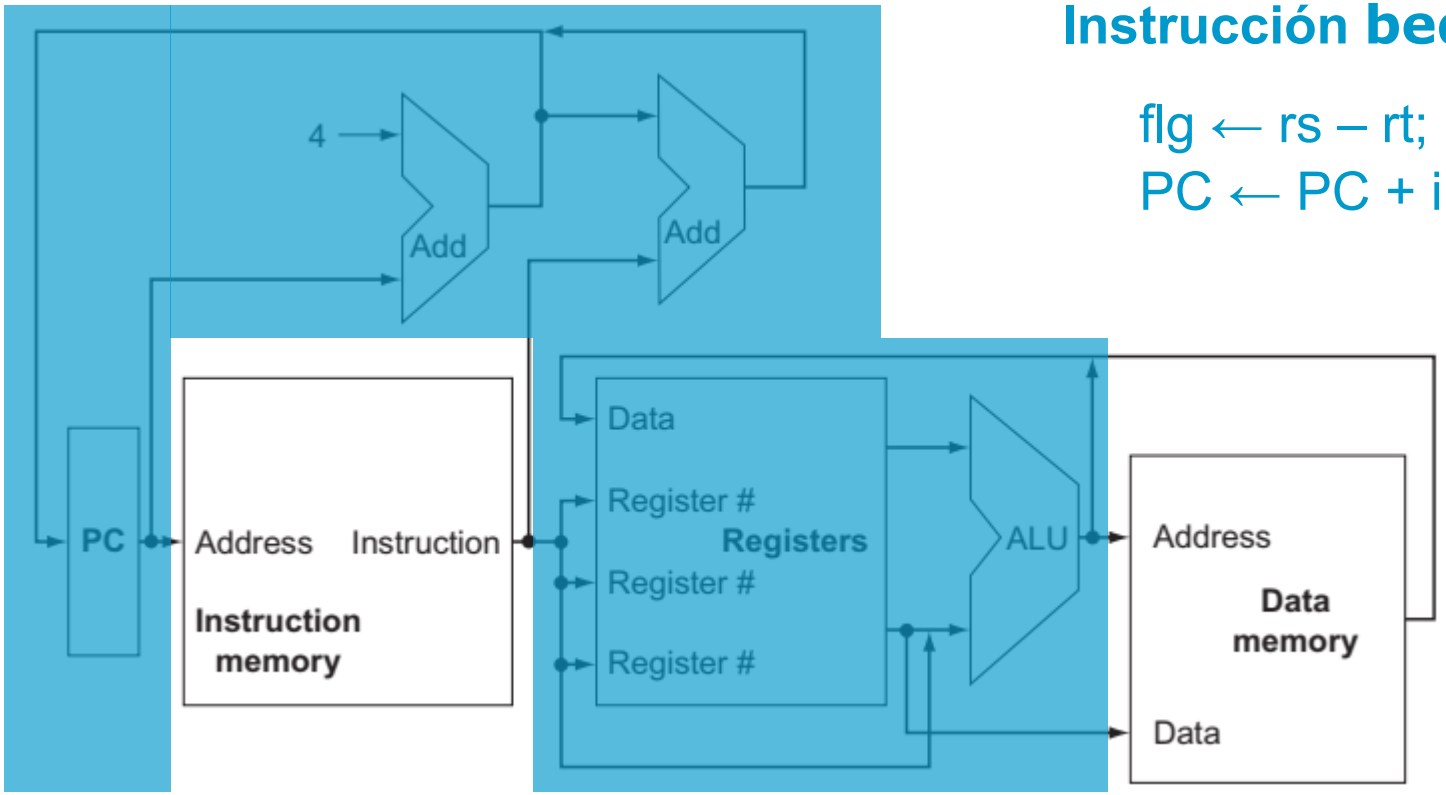
Instrucción beq

flg ← rs – rt;



Diseño del microprocesador MIPS

- Vista a muy alto nivel

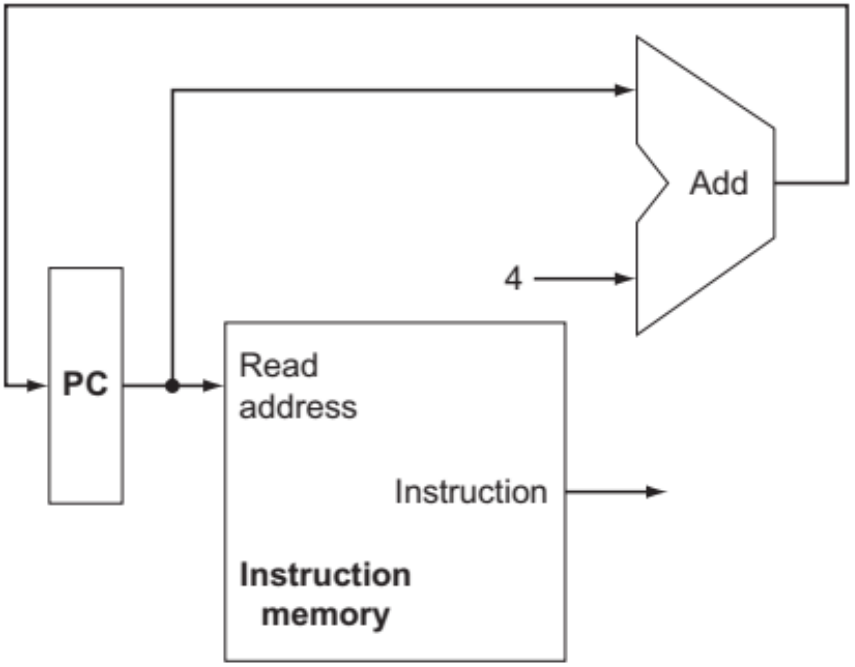


Instrucción beq

flg ← rs – rt;
PC ← PC + inm

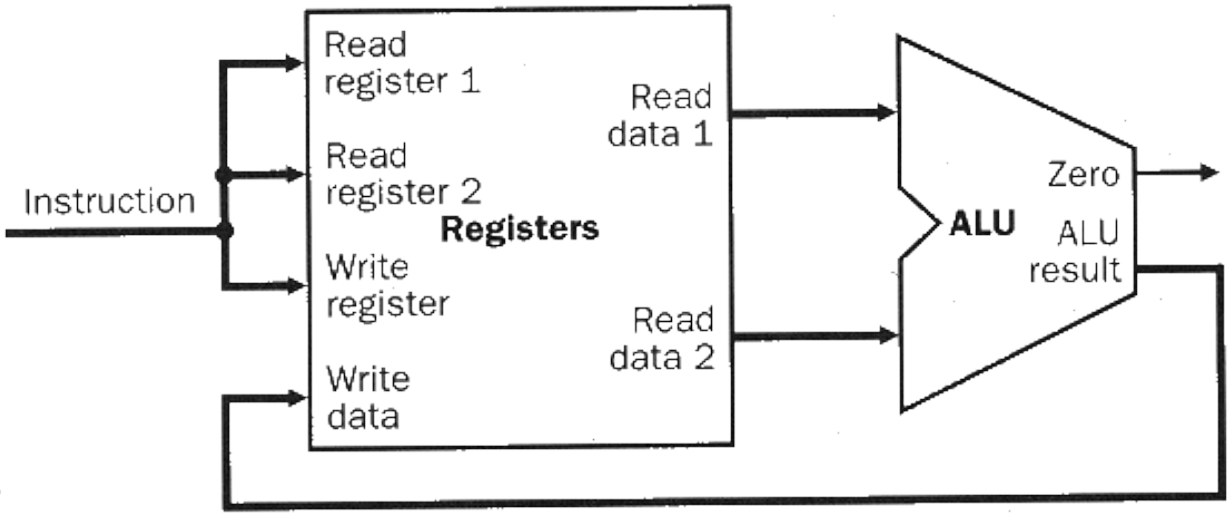
Diseño del microprocesador MIPS

- Parte del circuito para extracción de instrucción (Fetch)



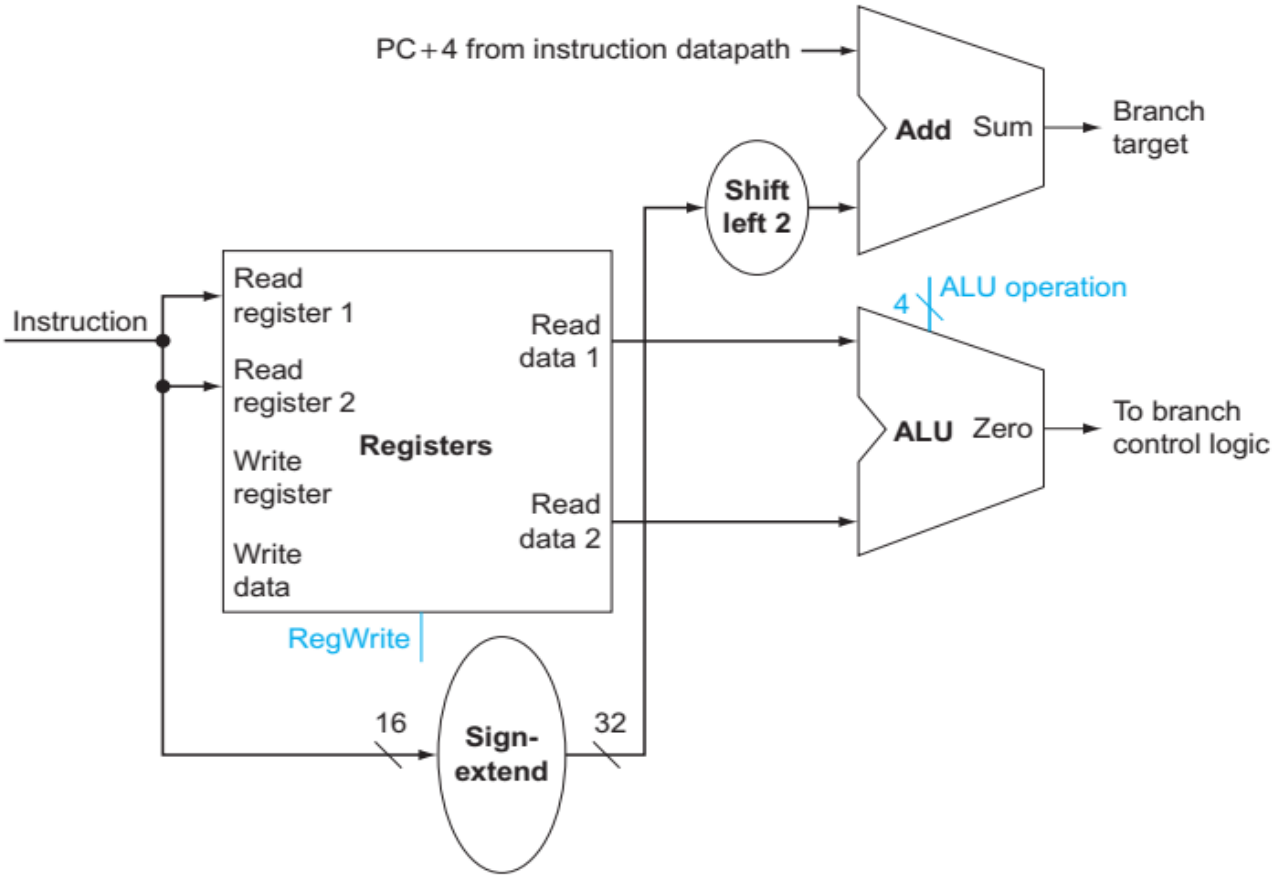
Diseño del microprocesador MIPS

- Parte del circuito para instrucción Aritméticas-lógicas (Tipo-R)
  - extracción de operandos
  - ejecución
  - almacenamiento de resultado



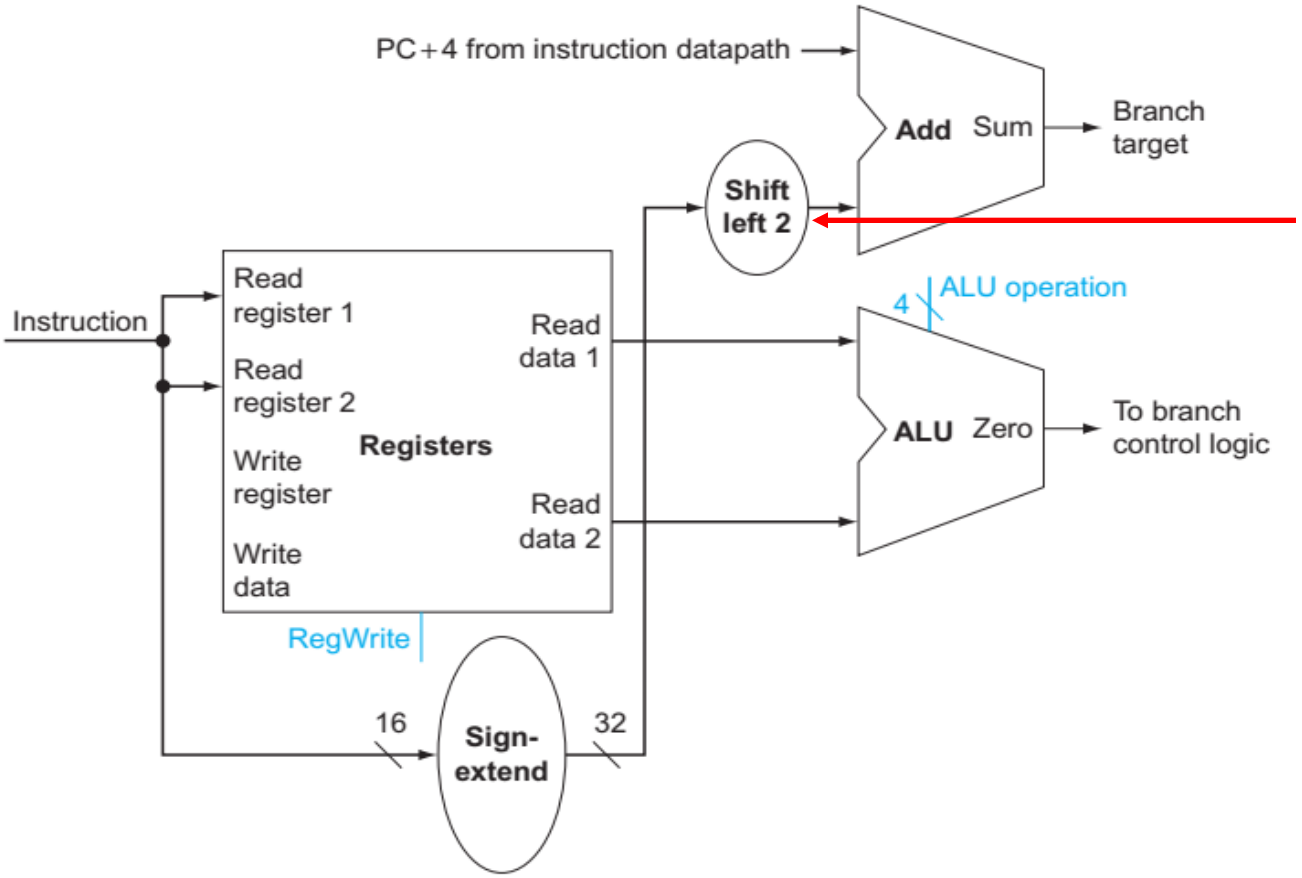
Diseño del microprocesador MIPS

- Parte del circuito para salto condicional (beq)
  - ALU para evaluar la condición de salto
  - Sumador para determinar la dirección de salto



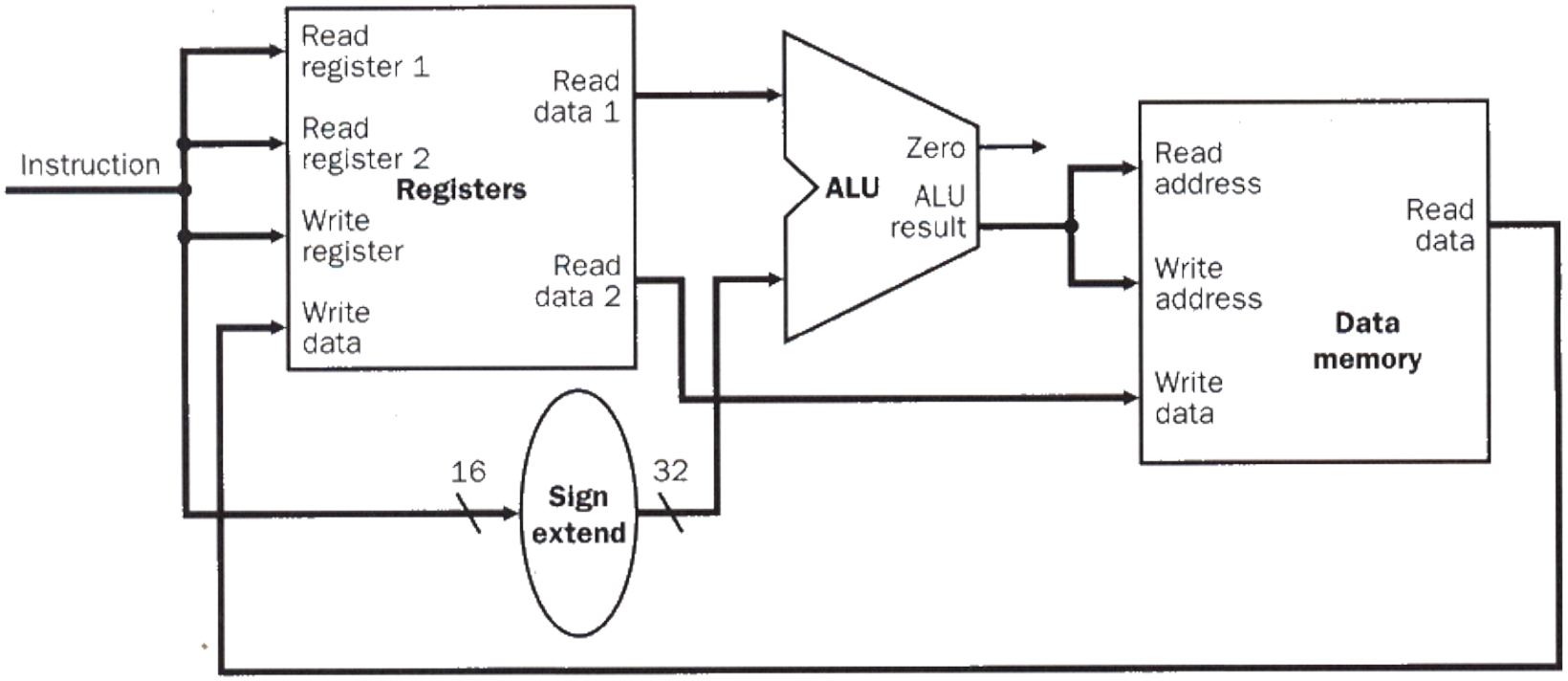
Diseño del microprocesador MIPS

- Parte del circuito para salto condicional (beq)
  - ALU para evaluar la condición de salto
  - Sumador para determinar la dirección de salto



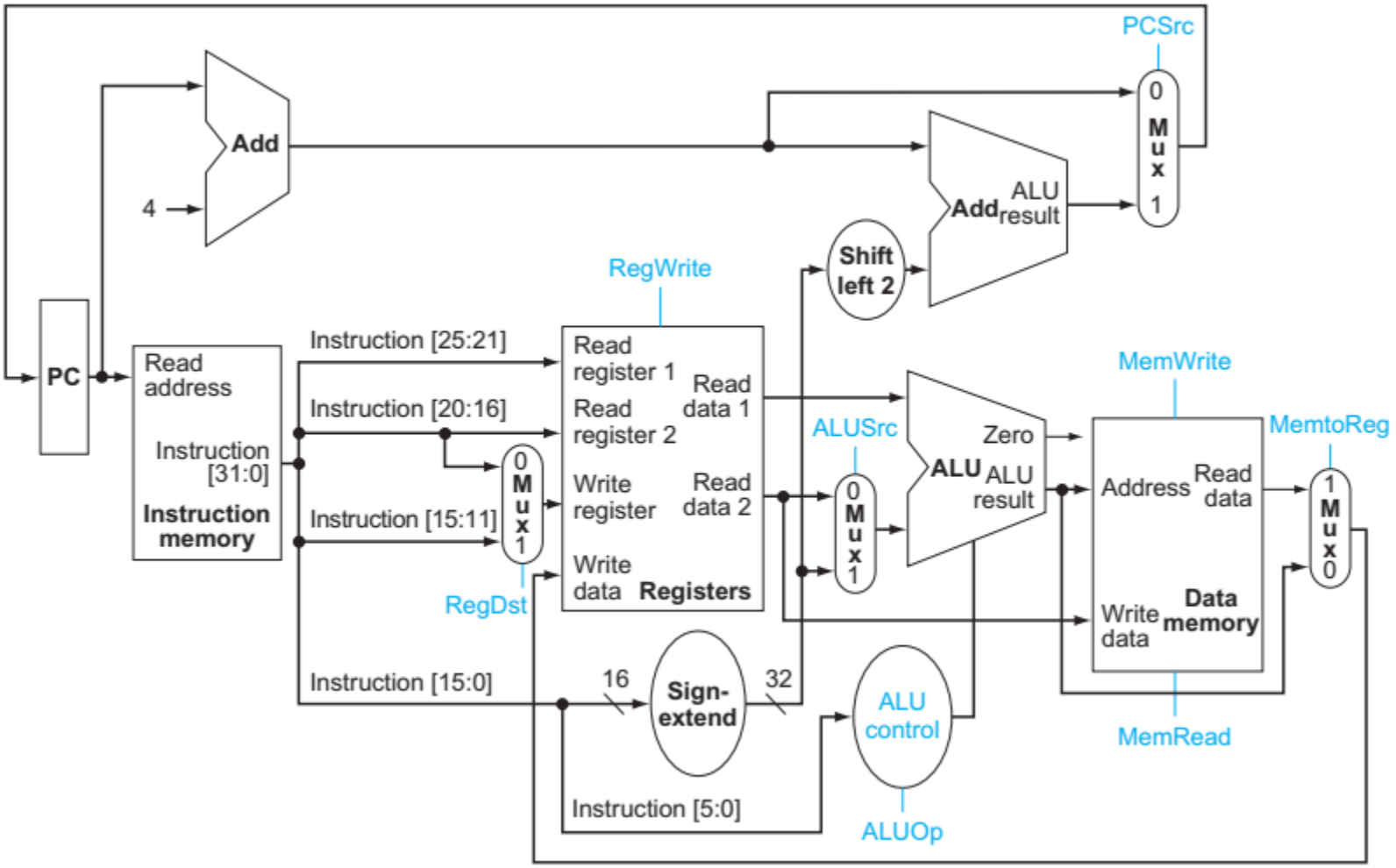
Diseño del microprocesador MIPS

- Parte del circuito para instrucciones lw y sw
  - ALU determina la dirección de acceso a memoria



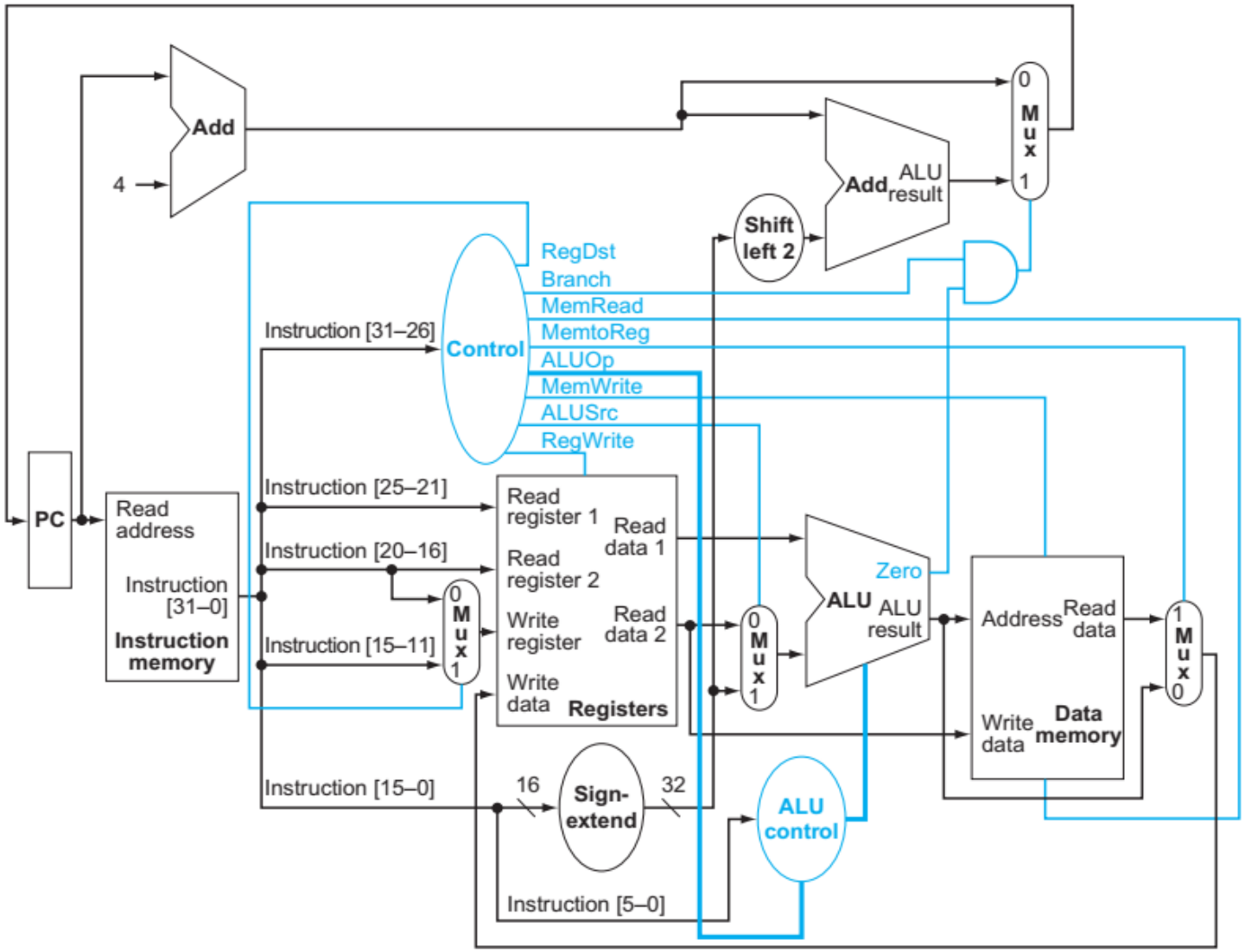
Diseño del microprocesador MIPS

- Combinación de todos los circuitos…



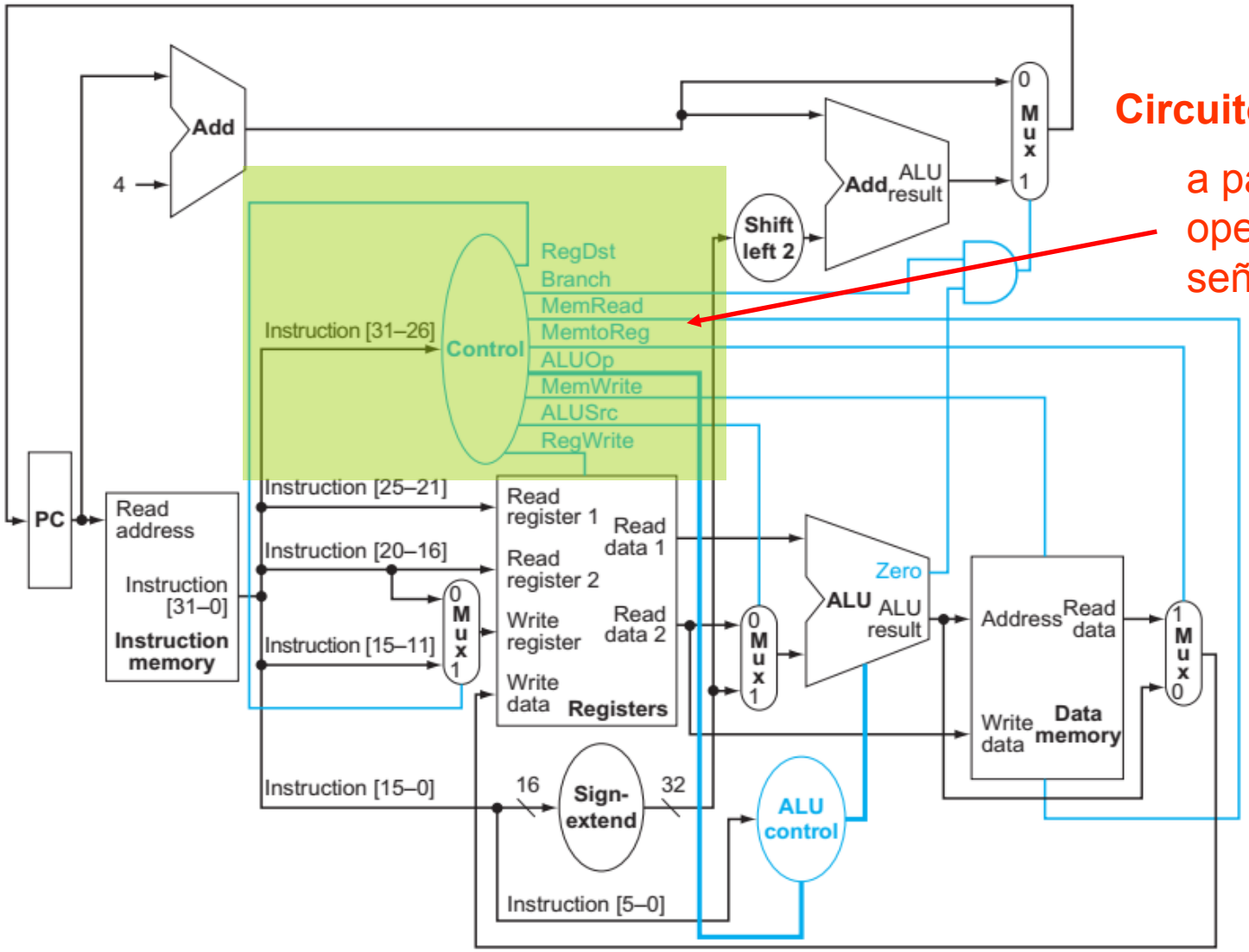
Diseño del microprocesador MIPS

- Completo con Unidad de Control



Diseño del microprocesador MIPS

- Completo con Unidad de Control

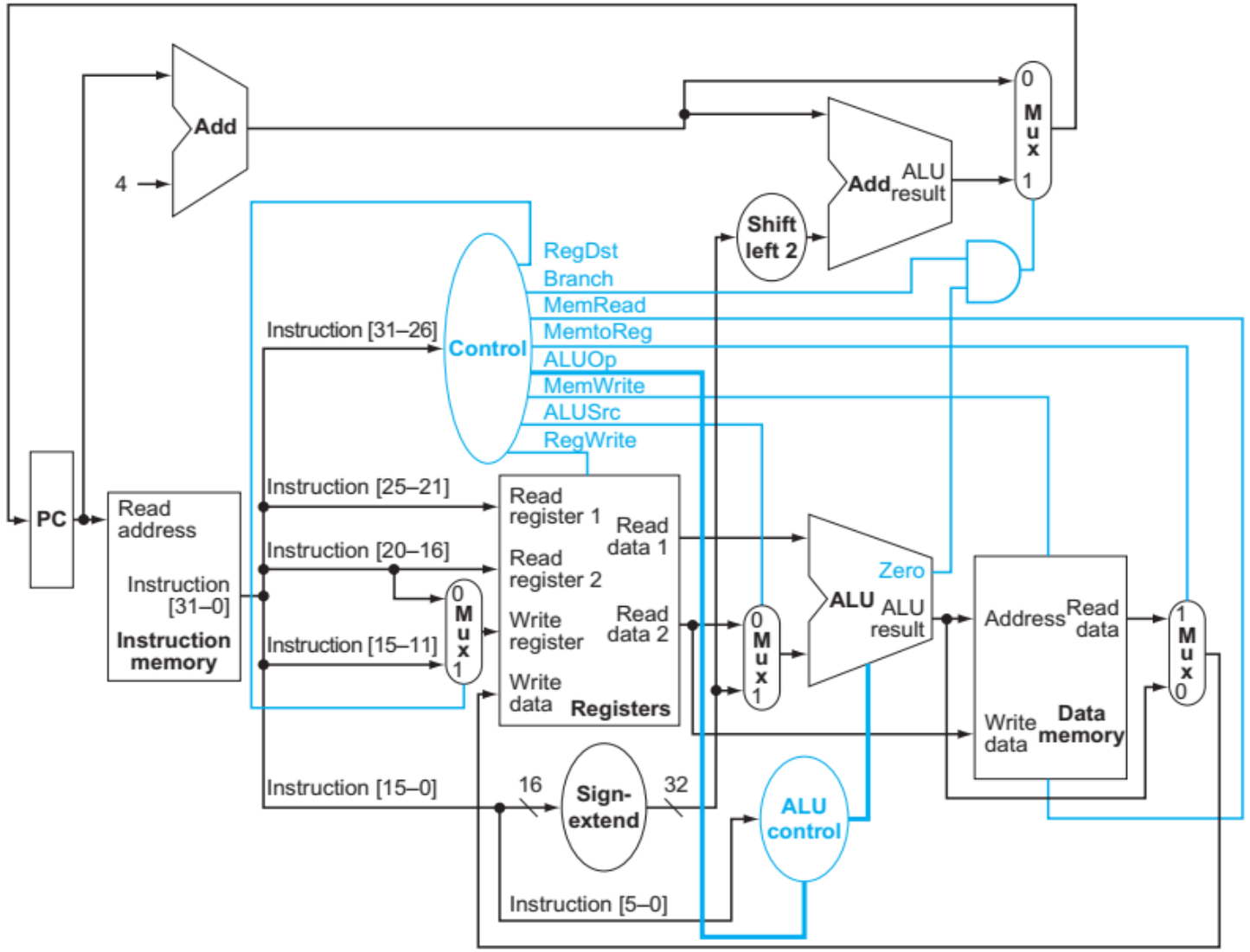


Circuito combinacional

a partir del código de operación genera señales de control

Diseño del microprocesador MIPS

- Señales de control para Instrucción tipo-R



RegDst = 1

ALUSrc = 0

MemtoReg = 0

RegWrite = 1

MemRead = 0

MemWrite = 0

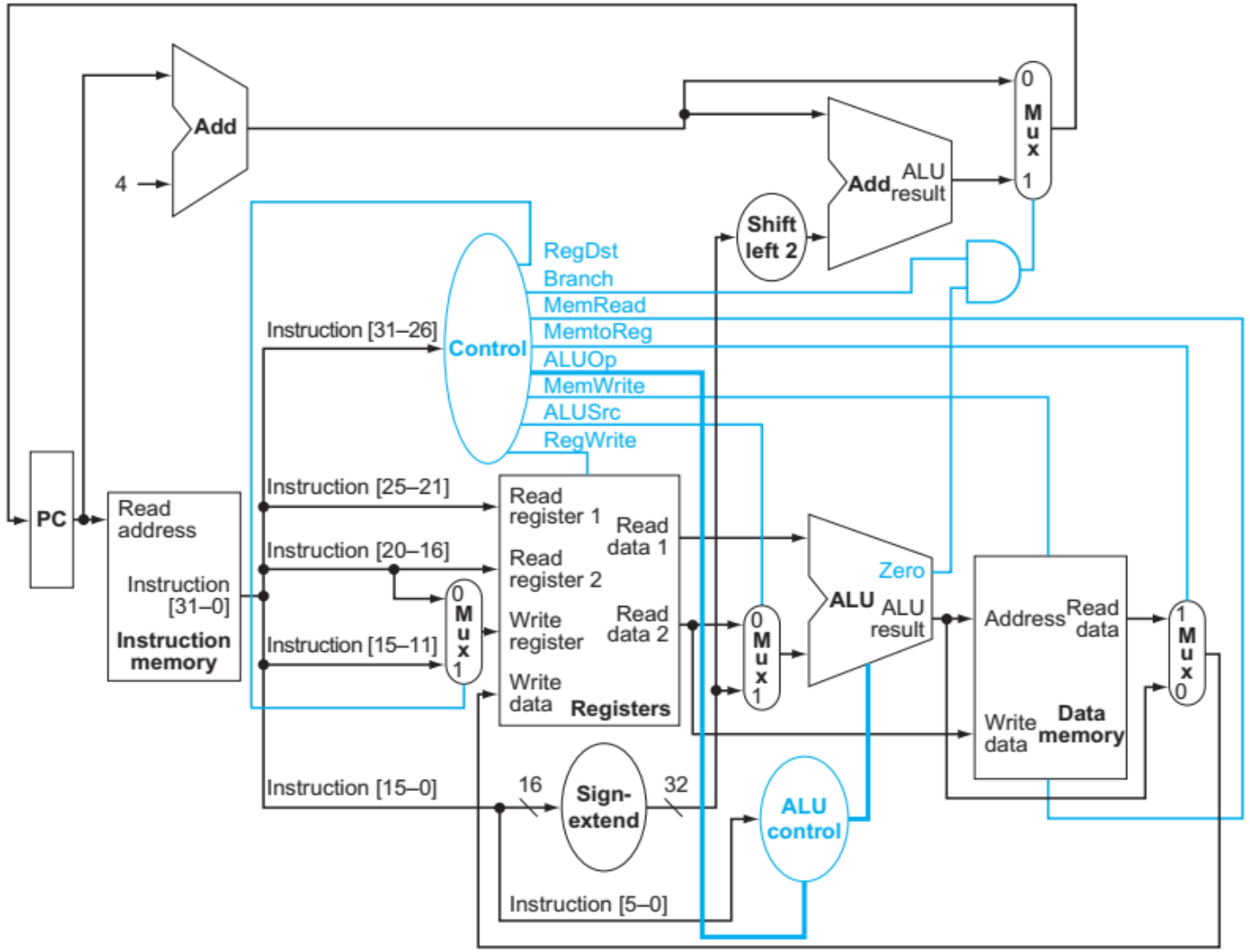
Branch = 0

ALUOp1 = 1

ALUOp0 = 0

Diseño del microprocesador MIPS

- Señales de control para instrucción lw



RegDst = 0

ALUSrc = 1

MemtoReg = 1

RegWrite = 1

MemRead = 1

MemWrite = 0

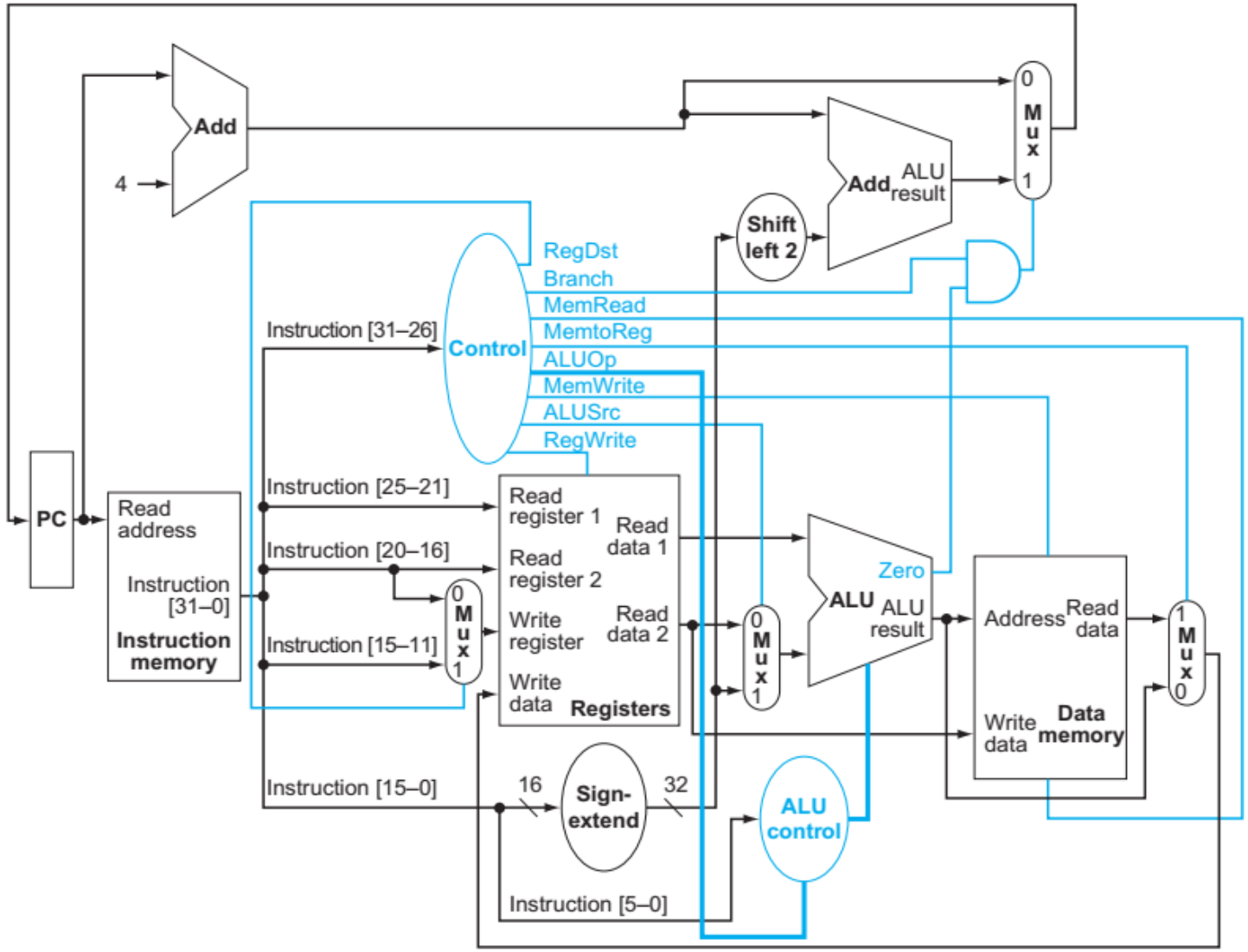
Branch = 0

ALUOp1 = 0

ALUOp0 = 0

Diseño del microprocesador MIPS

- Señales de control para instrucción sw



RegDst = X

ALUSrc = 1

MemtoReg = X

RegWrite = 0

MemRead = 0

MemWrite = 1

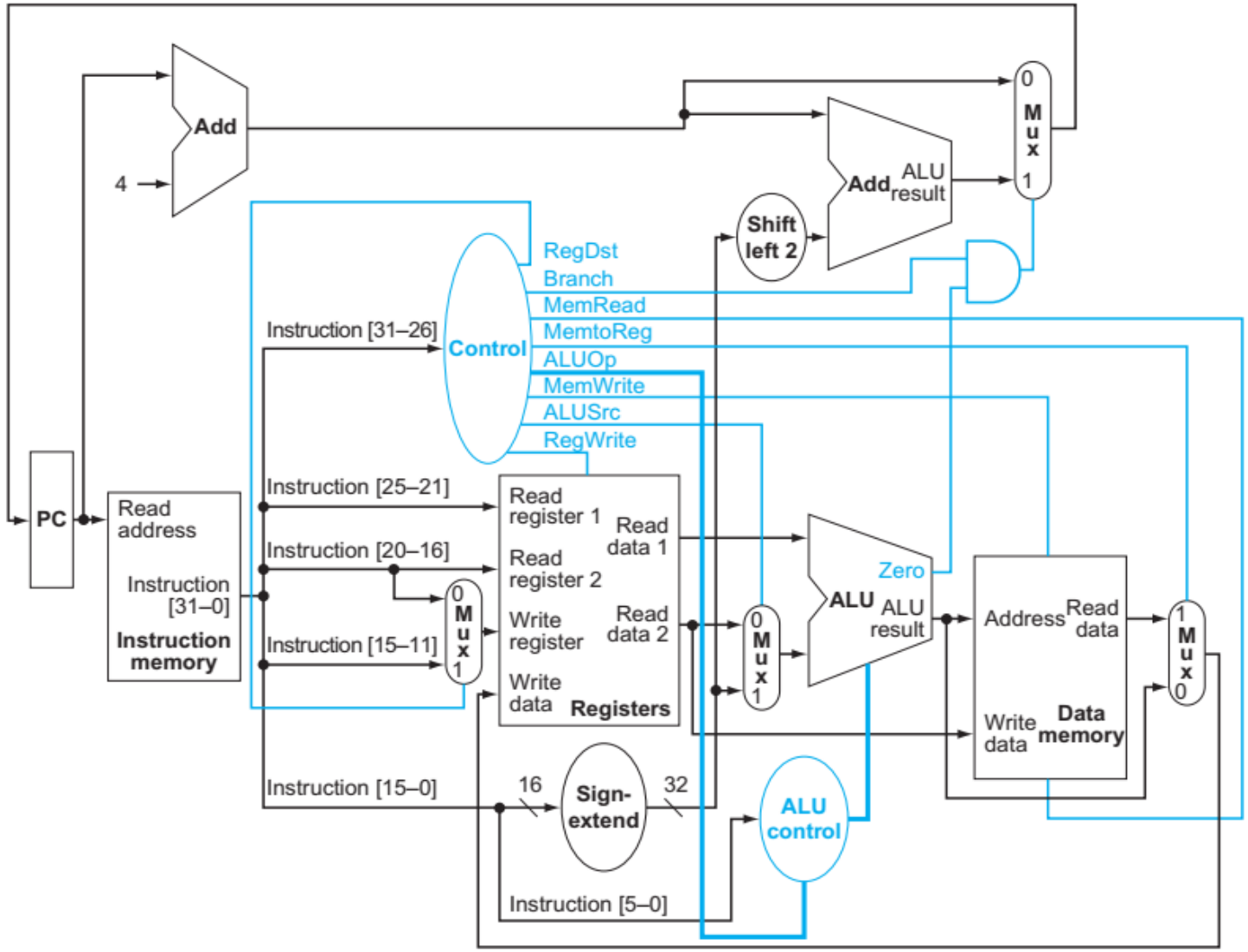
Branch = 0

ALUOp1 = 0

ALUOp0 = 0

Diseño del microprocesador MIPS

- Señales de control para instrucción beq



RegDst = X

ALUSrc = 0

MemtoReg = X

RegWrite = 0

MemRead = 0

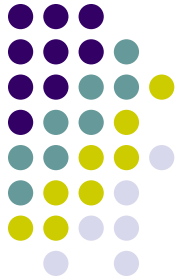
MemWrite = 0

Branch = 1

ALUOp1 = 0

ALUOp0 = 1

Diseño del microprocesador MIPS

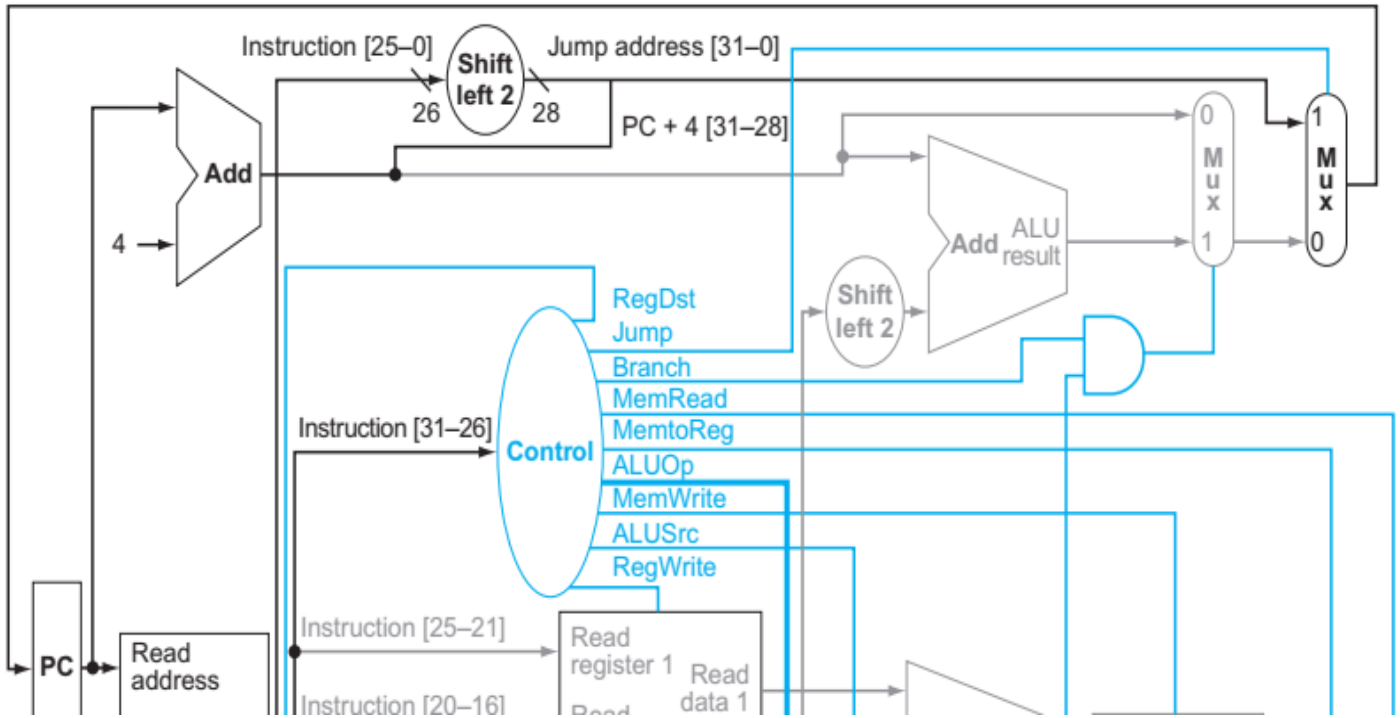


- Las señales generadas por la Unidad de Control actúan sobre:
  - mutiplexores,
  - habilitación de escritura en banco de registros y memorias
  - operaciones de la ALU

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
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

Diseño del microprocesador MIPS

- Si agregamos la implementación de la instrucción j al diseño



RegDst = X

ALUSrc = X

MemtoReg = X

RegWrite = 0

MemRead = 0

MemWrite = 0

Branch = 0

ALUOp1 = X

ALUOp0 = X

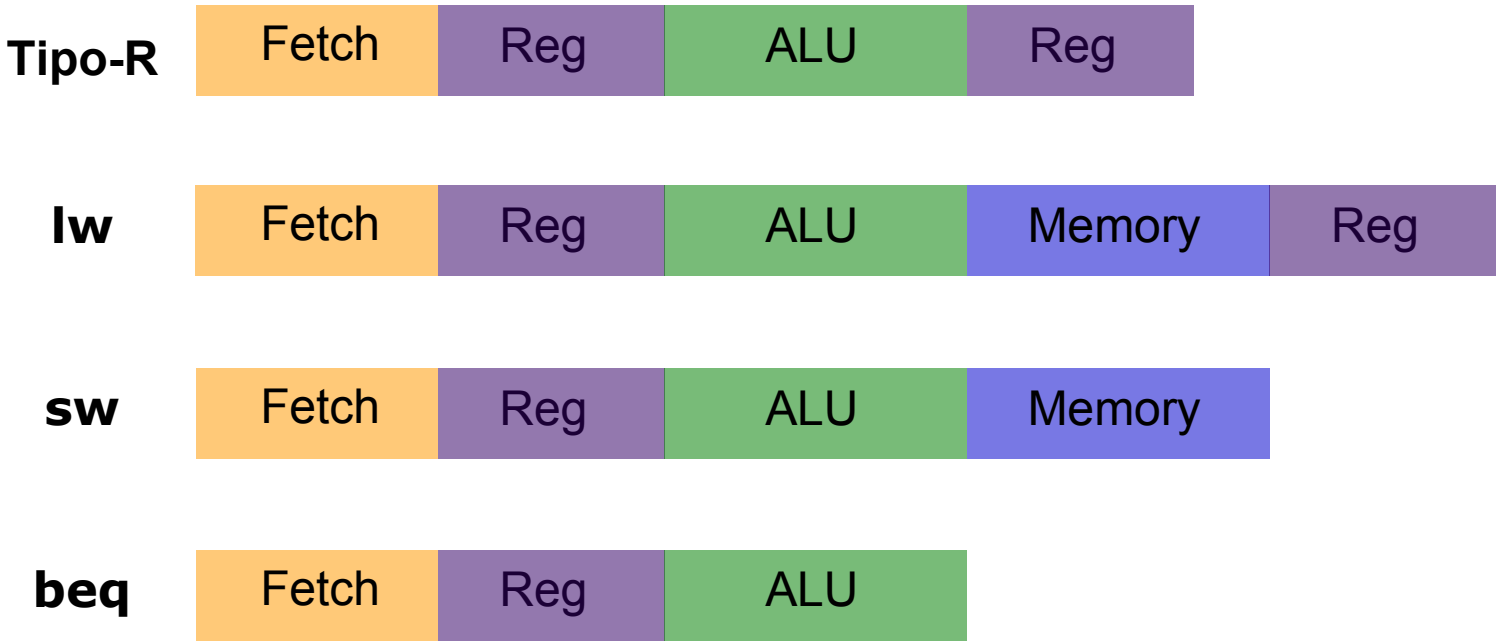
Jump = 1

Operación

PC = (PC+4)[31..28] & target_address<<2

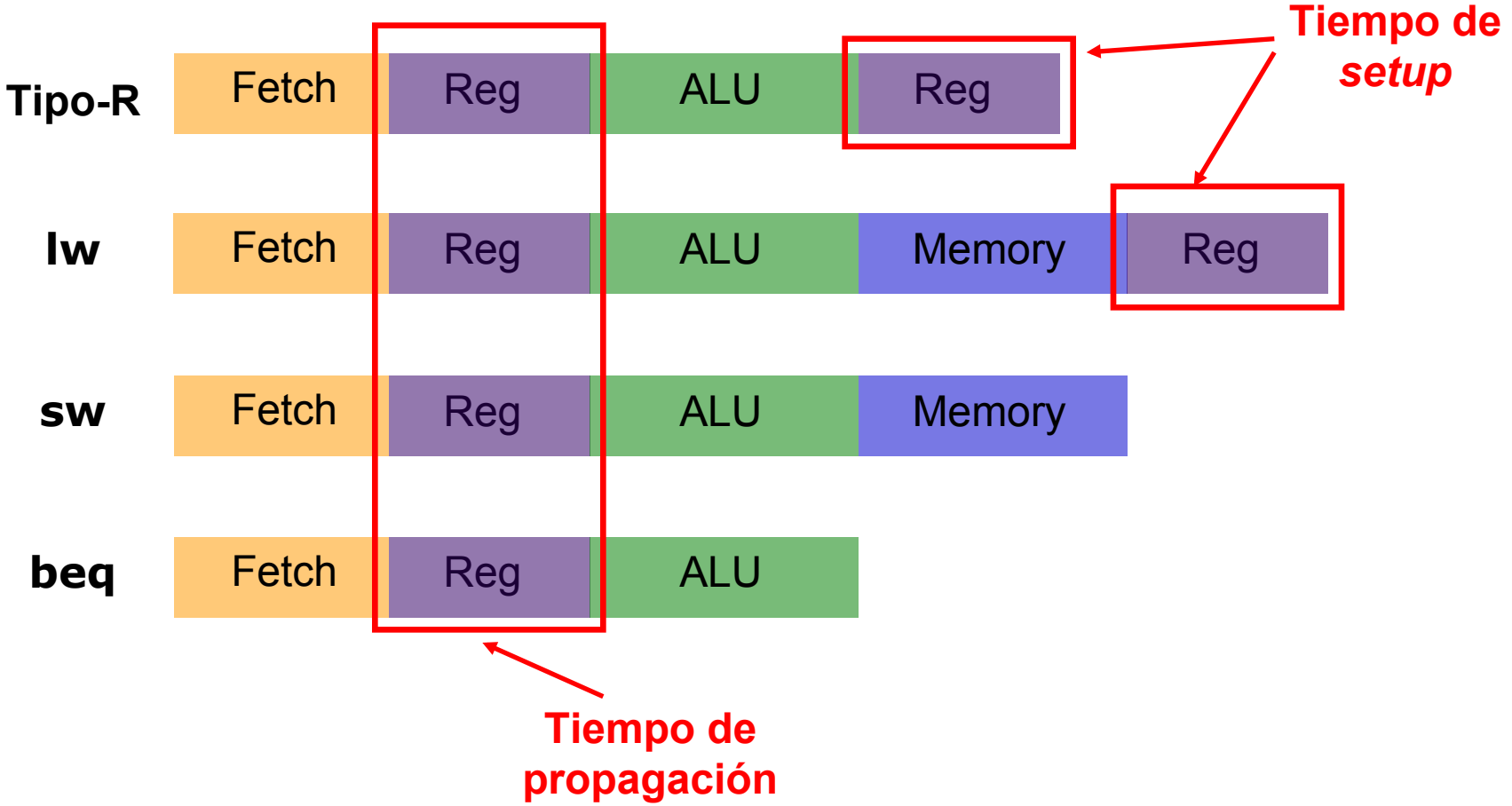
Microprocesador uniciclo

- Análisis **simplificado** de tiempo desde que se extrae la instrucción (fetch) hasta que se deposita resultado o determina salto



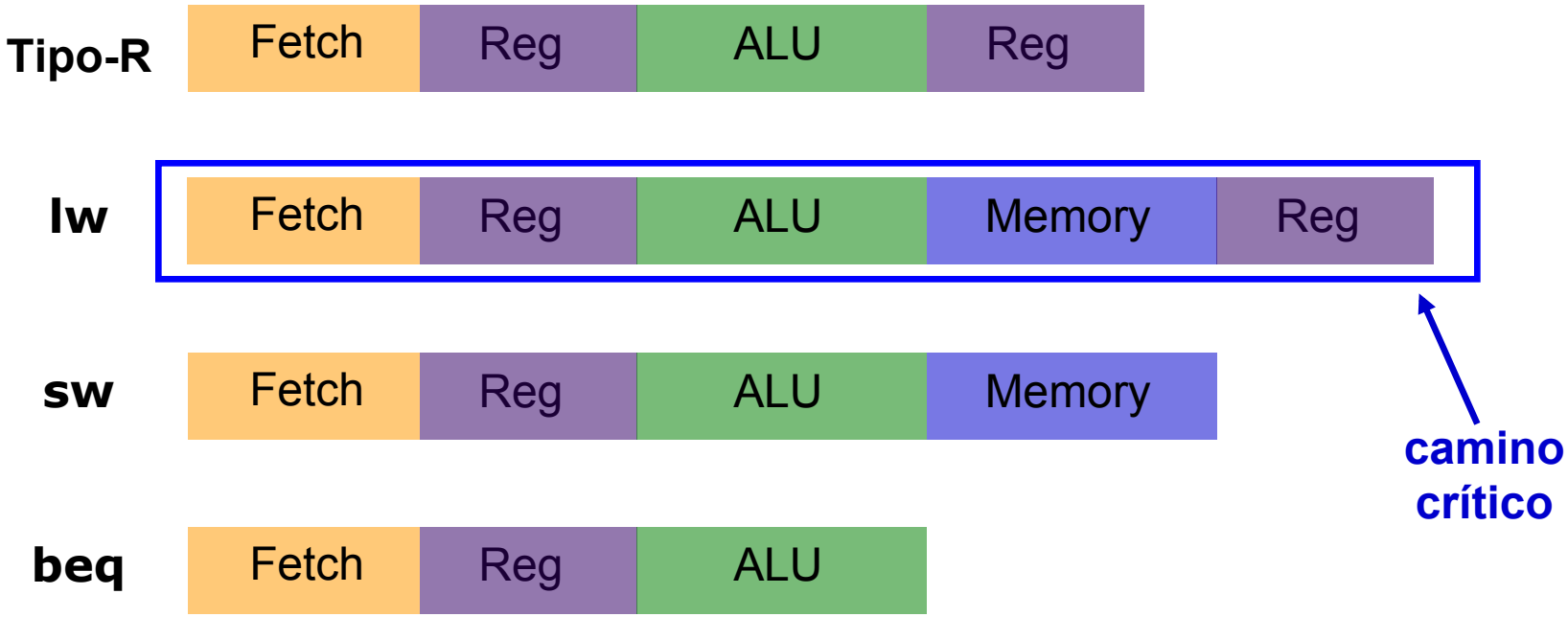
Microprocesador uniciclo

- Análisis **simplificado** de tiempo desde que se extrae la instrucción (fetch) hasta que se deposita resultado o determina salto



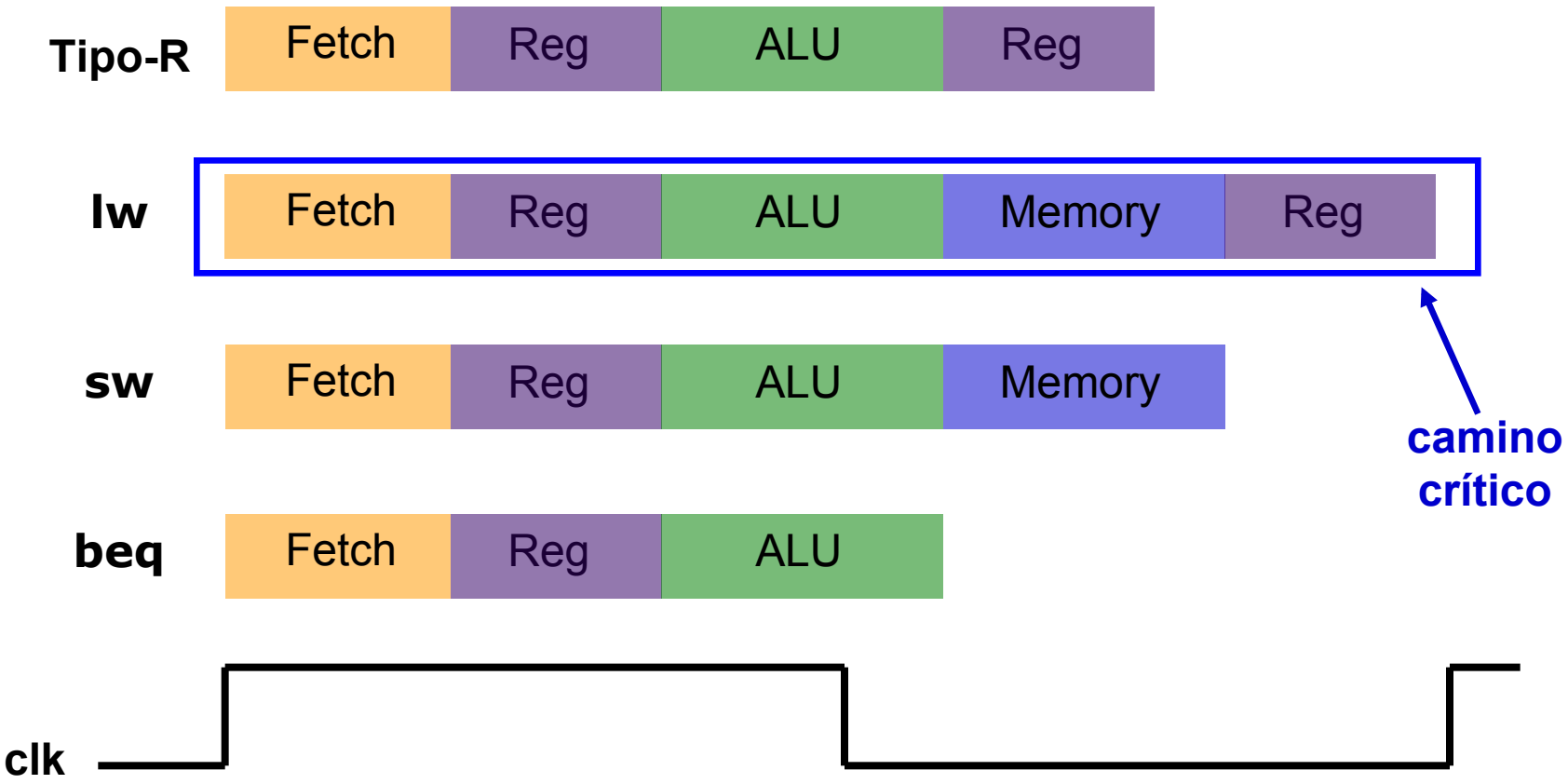
Microprocesador uniciclo

- Análisis **simplificado** de tiempo desde que se extrae la instrucción (fetch) hasta que se deposita resultado o determina salto

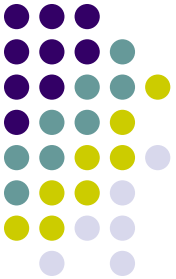


Microprocesador uniciclo

- Análisis **simplificado** de tiempo desde que se extrae la instrucción (fetch) hasta que se deposita resultado o determina salto



Microprocesador uniciclo
Inconvenientes



- Se debe respetar la instrucción que más tarda para determinar el tiempo de ciclo de reloj
- Todas las instrucciones utilizan la misma cantidad de tiempo que la instrucción más lenta
- Mucho tiempo de cómputo para programas con muchas instrucciones
- Duplicación de componentes de similar funcionalidad
- Imposibilidad de reutilizar componentes inactivos durante la ejecución de una instrucción