# Diseño de un procesador secuencial

# Introducción

- El procesador o CPU (unidad central de procesamiento) sigue las instrucciones del programa al pie de la letra.
- Un procesador consta de dos componentes:

1. El datapath. Ejecuta operaciones aritméticas y lógicas.
2. El control. Ordena al datapath, memoria y dispositivos de I/O lo que hay que hacer de acuerdo al programa.

# MIPS simplificado

- Se examinará una implementación de un subconjunto de MIPS:

1. Instrucciones de referencia a memoria: load word (lw) y store word (sw).
2. Instrucciones aritmético-lógicas: suma (add), resta (sub), and, or y set on less than (slt).
3. Instrucciones de brincos: brinca si igual (beq) y brinco incondicional (j).

- Las instrucciones se ejecutan de forma secuencial en un ciclo de reloj.

# Implementación

- Leer una instrucción.
- Leer registros.
- Ejecutar la instrucción.
- Guardar resultados.

# Primeros pasos

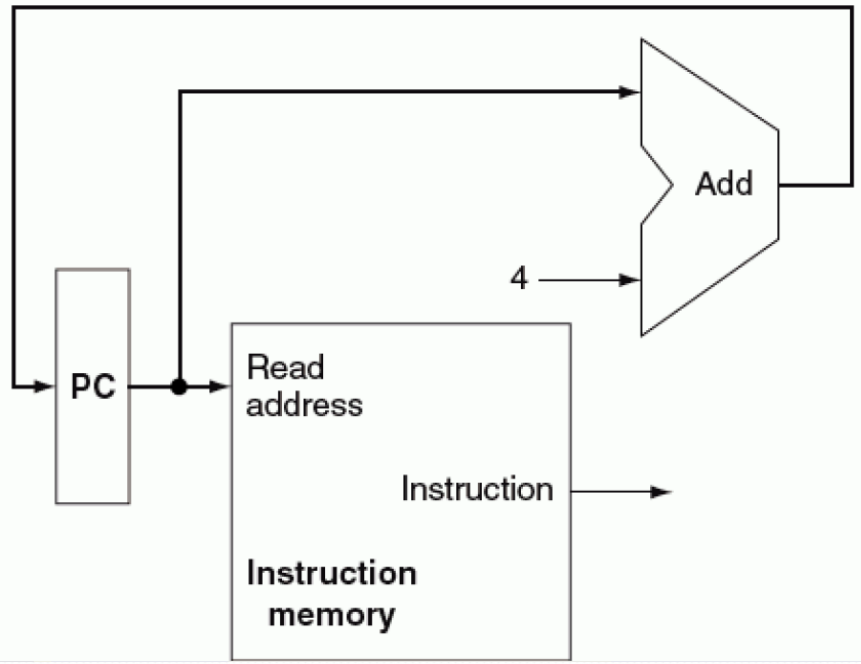
- La implementación de las distintas instrucciones tiene varias acciones en común.
- Los primeros dos pasos son iguales:

1. Enviar el PC (contador de programa) a la memoria y sacar la siguiente instrucción (ciclo de fetch).
2. Leer uno o dos registros del banco de registros.

- Lo siguiente depende de la clase de instrucción, pero es parecido.

# Ciclo de fetch

- Lee una instrucción e incrementa el PC.




**Fuente: COD-HSI 5, p. 253**

# Ejecutar la instrucción

- Todas las instrucciones, excepto el brinco incondicional, usan la ALU (unidad aritmético-lógica).
- Las instrucciones de referencia a memoria para calcular direcciones.
- Las instrucciones aritmético-lógicas para su operación.
- Los brincos para evaluar la condición.

# Resultado

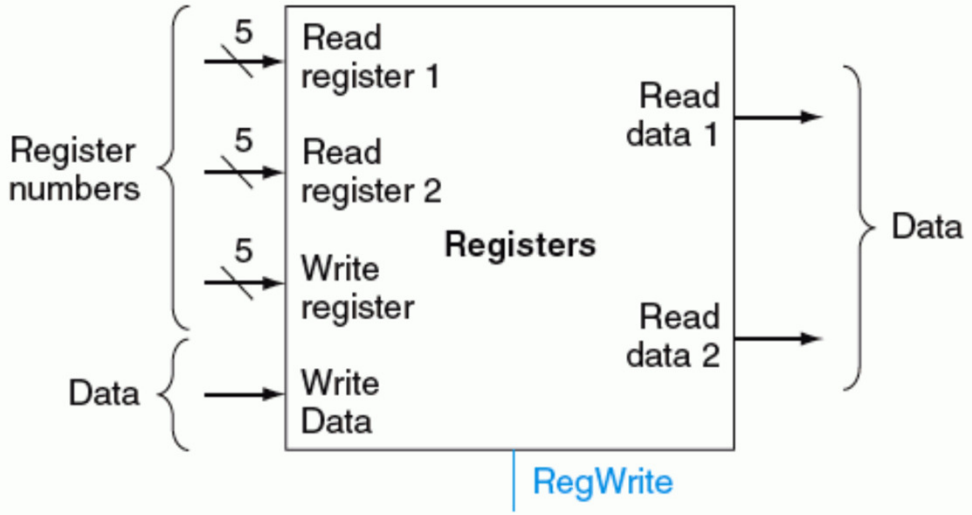
- Las instrucciones aritmético-lógicas guardan la salida de la ALU en el registro destino.
- La carga lee un dato de la memoria y lo guarda en un registro.
- El store guarda el valor del registro fuente en la memoria.
- Los brincos condicionales, dependiendo de la condición, actualizan el PC (contador de programa).
- Los brincos incondicionales actualizan el PC.

# Implementación

- Se utilizan los siguientes componentes:
- Componentes principales: banco de registros, memoria de instrucciones, memoria de datos y ALU.
- Componentes extras: sumadores, unidades de corrimiento, multiplexores y extensor de signo.

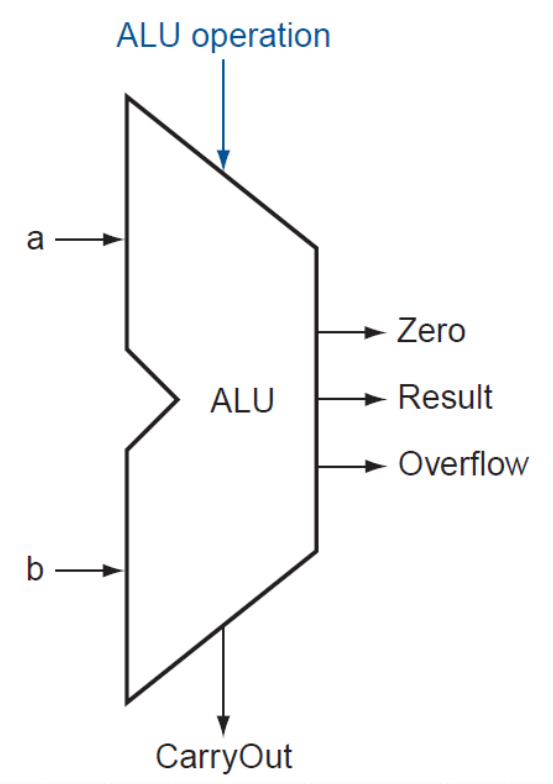
# Banco de registros

- Dos puertos de lectura y uno de escritura.




**Fuente: COD-HSI 5, p. 253**

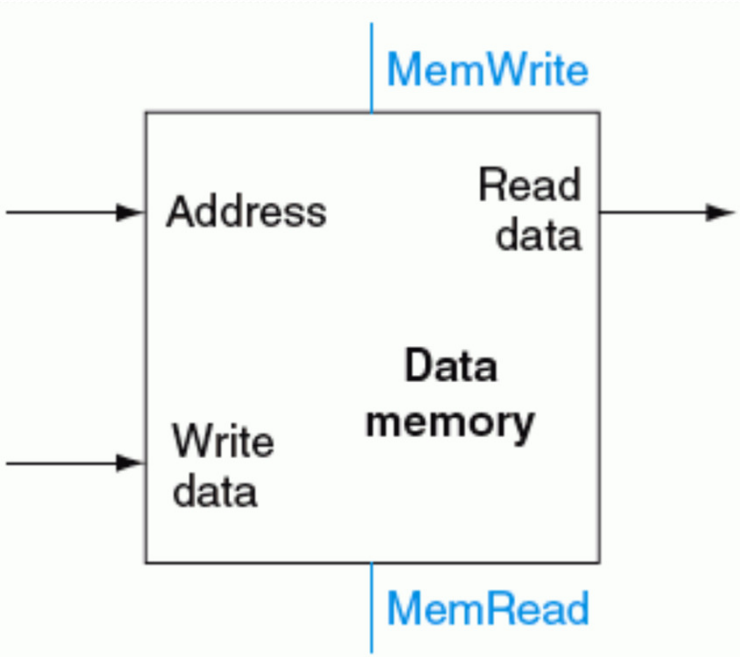
# ALU




| ALU operation | | | | Result |
|---|---|---|---|---|
| $C_3$ | $C_2$ | $C_1$ | $C_0$ | |
| 0 | 0 | 0 | 0 | AND |
| 0 | 0 | 0 | 1 | OR |
| 0 | 0 | 1 | 0 | suma |
| 0 | 1 | 1 | 0 | resta |
| 0 | 1 | 1 | 1 | set on less than |
| 1 | 1 | 0 | 0 | NOR |

**Fuente: COD-HSI 5, p. B-37**

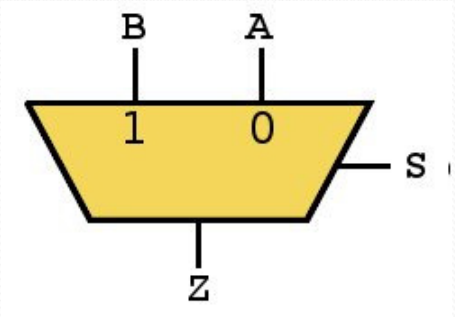
# Memoria de datos




**Fuente: COD-HSI 5, p. 255**

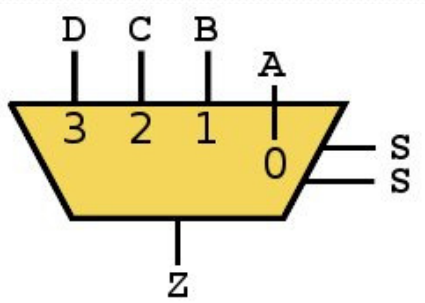
# Multiplexor (mux)

- El multiplexor tiene $2^n$ entradas de datos, una salida y n bits de selección.
- Los bits de selección se usan para decidir cuál entrada pasa a la salida.
- Mux 2 a 1

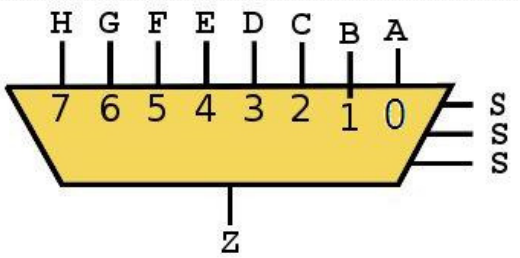


# Multiplexor (mux)

- Mux 4 a 1.

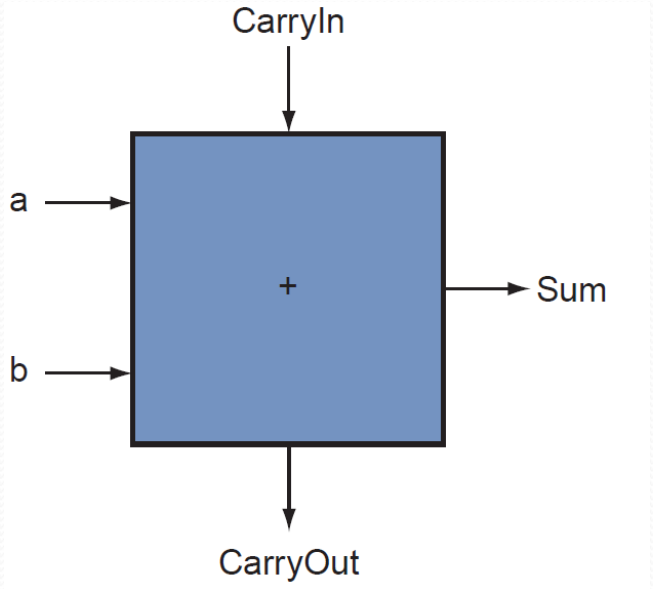



- Mux 8 a 1.



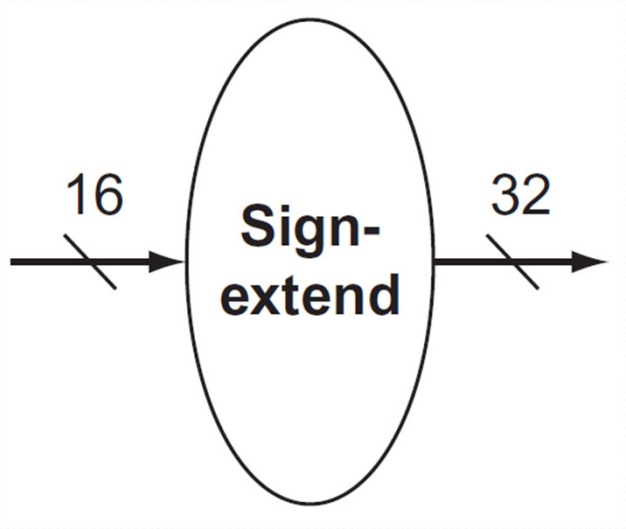
# Full adder / sumador completo

- Entrada: 2 números de 32 bits y 1 bit de carry.
- Salida: 1 número de 32 bits y 1 bit de carry.



# Extensor de signo

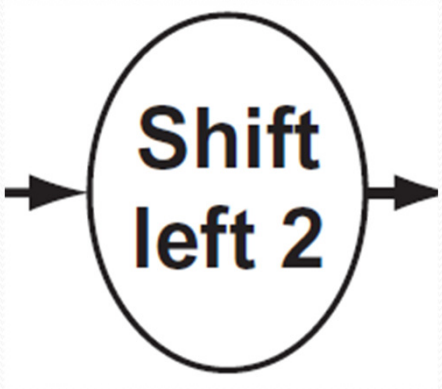
- Entrada: un número de 2 bytes.
- Salida: el mismo número extendido a 4 bytes.




**Fuente: COD-HSI 5, p. 255**

# Unidad de shift

- Entrada: un número de 4 bytes.
- Salida: el mismo número recorrido 2 bits a la izquierda.



# Instrucciones aritmético-lógicas

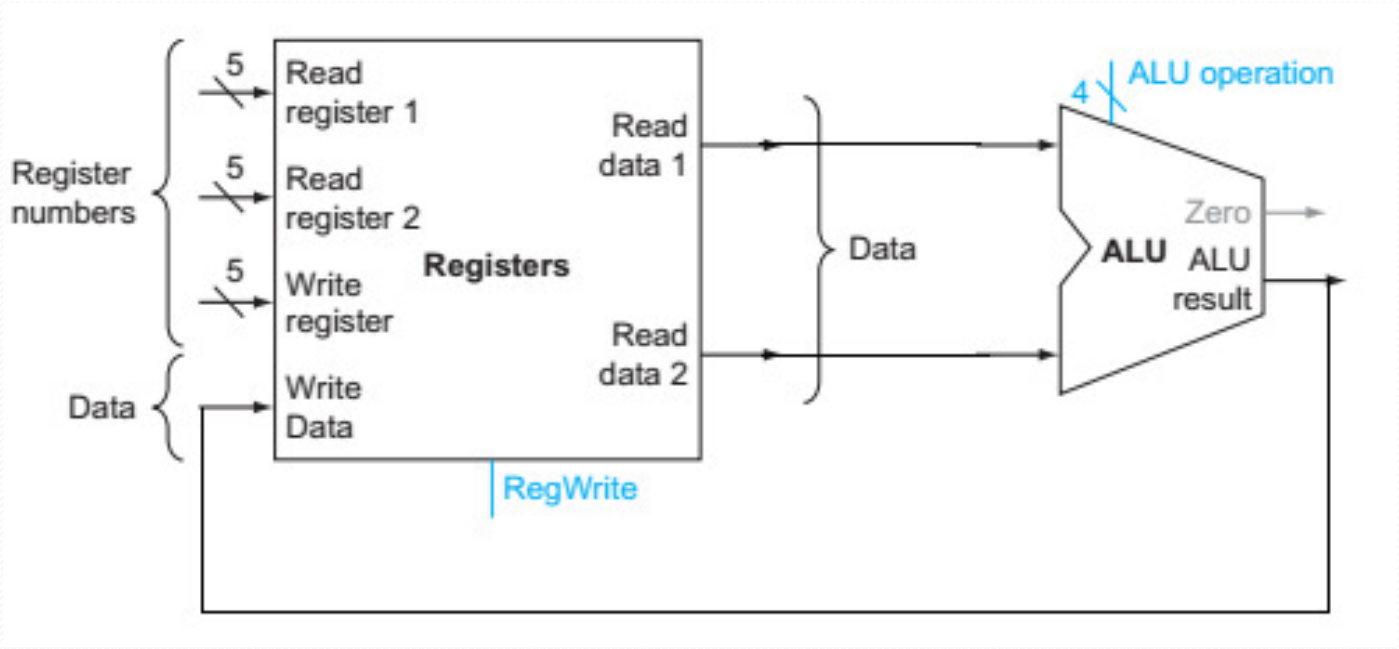
- Sus tres operandos son registros.
- Instrucciones add, sub, and, or, slt.
- Leen dos registros fuente, realizan una operación aritmética o lógica y escriben el resultado en el registro destino.
- Los registros están guardados en el banco de registros.
- La ALU se usa para las operaciones.

# Instrucciones aritmético-lógicas

- El datapath para las instrucciones aritmético-lógicas utiliza:

1. El banco de registros.
2. ALU.

# Datapath




**Fuente: COD-HSI 5, p. 247**

# Instrucciones de acceso a memoria

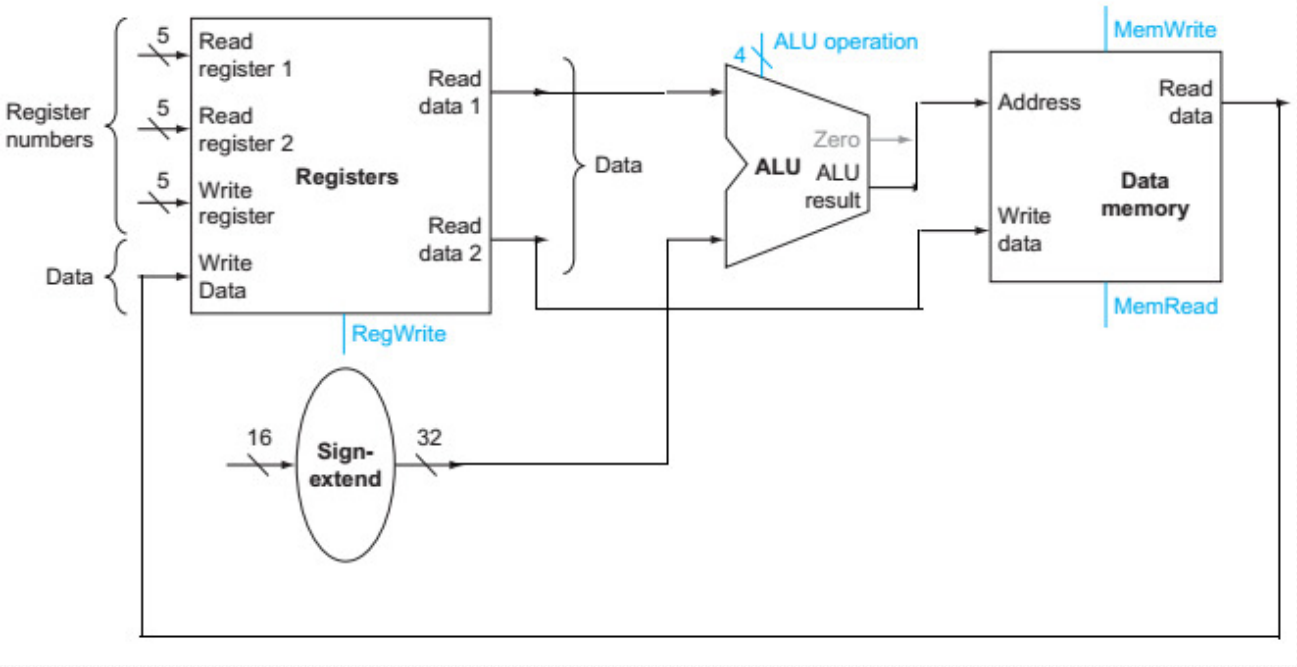
- lw $r1, offset ($r2) **# r1 ← Memoria[r2 + offset]**
- sw $r1, offset ($r2) **# Memoria[r2 + offset] ← r1**
- Hay que sumar el offset (16 bits con signo) al registro base $r2.
- lw tiene que escribir el valor que se obtuvo de la memoria en $r1.
- sw tiene que leer el valor de $r1 y guardar su valor en la memoria

# Instrucciones de acceso a memoria

- El datapath para las instrucciones de acceso a memoria utiliza:

1. Banco de registros.
2. ALU.
3. Extensor de signo.
4. Memoria de datos.

# Datapath




**Fuente: COD-HSI 5, p. 247**

# Instrucciones de brinco

- Dos instrucciones de brinco:
- Brinco condicional:

```
beq $t0, $t1, Etiqueta          # if t0 == t1 goto Etiqueta
```

- Brinco incondicional:

```
j Etiqueta                      # goto Etiqueta
```

# Brinco condicional

- En MIPS, los brincos condicionales son relativos al PC (contador del programa).
- Ejemplo:

```
1000 Etiqueta:      …
                    …
1040                beq $s0, $s1, Etiqueta
```

- El brinco beq se traduce en lenguaje máquina :

```
000100 10000 10001 1111111111110101
 beq     $s0    $s1          -11
```

# Brinco condicional

PC += 4

if $s0 == $s1

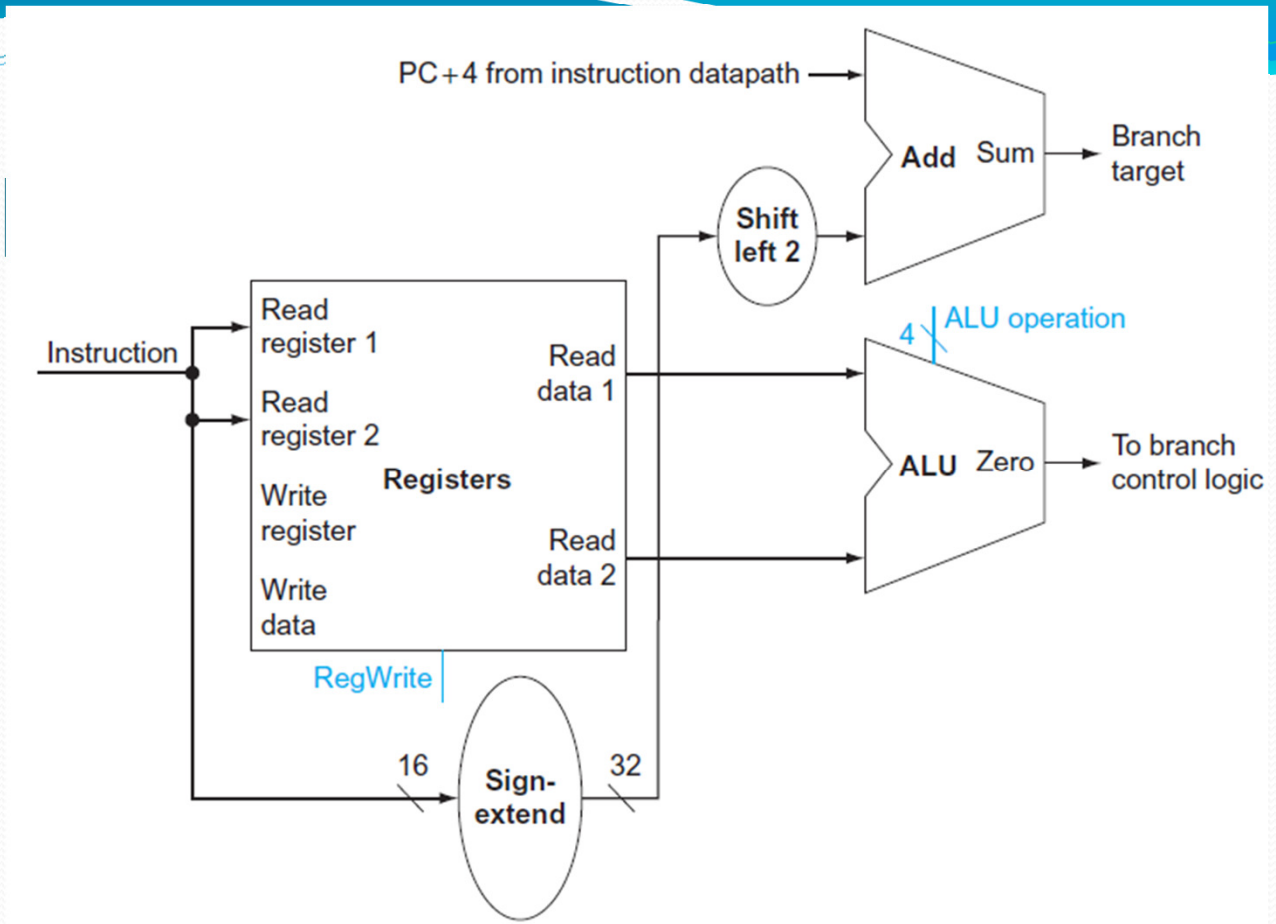
PC = PC + offset * 4

# Brinco condicional

- El datapath para la instrucción de brinco condicional utiliza:

1. Banco de registros.
2. Extensor de signo.
3. Sumador.
4. ALU.
5. Unidad de shift.

# Datapat




**Fuente: COD-HSI 5, p. 256**

# Brinco incondicional

- En MIPS los brincos incondicionales son absolutos.

```
1000 Etiqueta:        …
                      …
1040                  j Etiqueta
```

- La instrucción j se traduce en lenguaje máquina:

```
000010 00000000000000000011111010
   j                    250
```

# Brinco incondicional

```
PC += 4                    # no tiene efecto
PC = (offset << 2)         # multiplica por 4
```

- La instrucción j necesita hacer un shift a la izquierda de 2 bits.
- Usa una unidad de shift.

# Uniendo datapaths

- Objetivo: crear un solo datapath a partir de los datapaths individuales:

a) Instrucciones aritméticas y lógicas.
b) Instrucciones de acceso a memoria.
c) Brincos.

- Agregar el control.

# Paso 1

- Unir los datapaths de las instrucciones aritméticas y lógicas y el de las instrucciones de acceso a memoria.
- Las diferencias son en las entradas de la ALU y en el registro destino.

# ALU

a) Instrucciones aritméticas y lógicas: las entradas son dos registros.
b) Instrucciones de acceso a la memoria: una entrada es un registro y la otra es una constante.

- Ejemplo:

```
add $t0, $t1, $t2   # usa la ALU para sumar t1 + t2
sw $t0, 50($a0)     # usa la ALU para sumar a0 + 50
```

# Registro destino

a) Instrucciones aritméticas y lógicas: el valor que se guarda es la salida de la ALU.

b) Instrucción de carga: el valor que se guarda viene de la memoria.

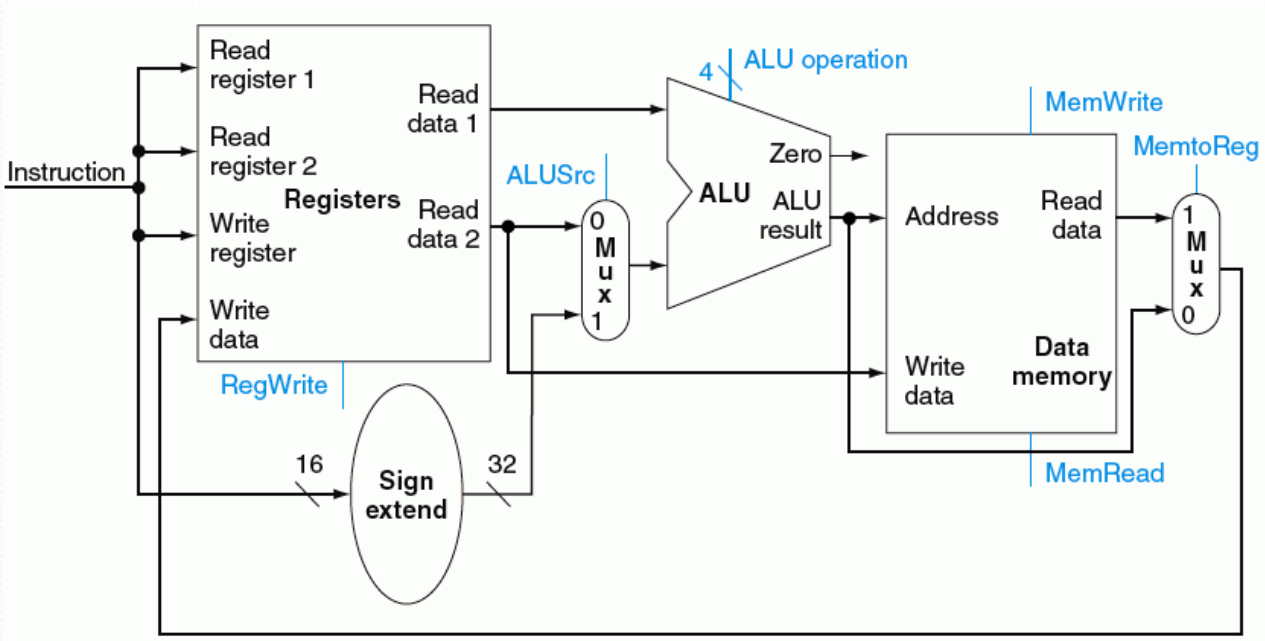
- Ejemplo:

```
add $t0, $t1, $t2    # t0 ← t1 + t2
lw $t0, 50($a0)      # t0 ← Memoria[a0 + 50]
```

# Uniendo datapaths

- Para usar la misma ALU: agregar un mux en la segunda entrada.
- Para usar el mismo banco de registros: agregar un mux en la entrada de datos.

# Datapath unificado paso 1




**Fuente: COD-HSI 5, p. 258**

# Paso 2

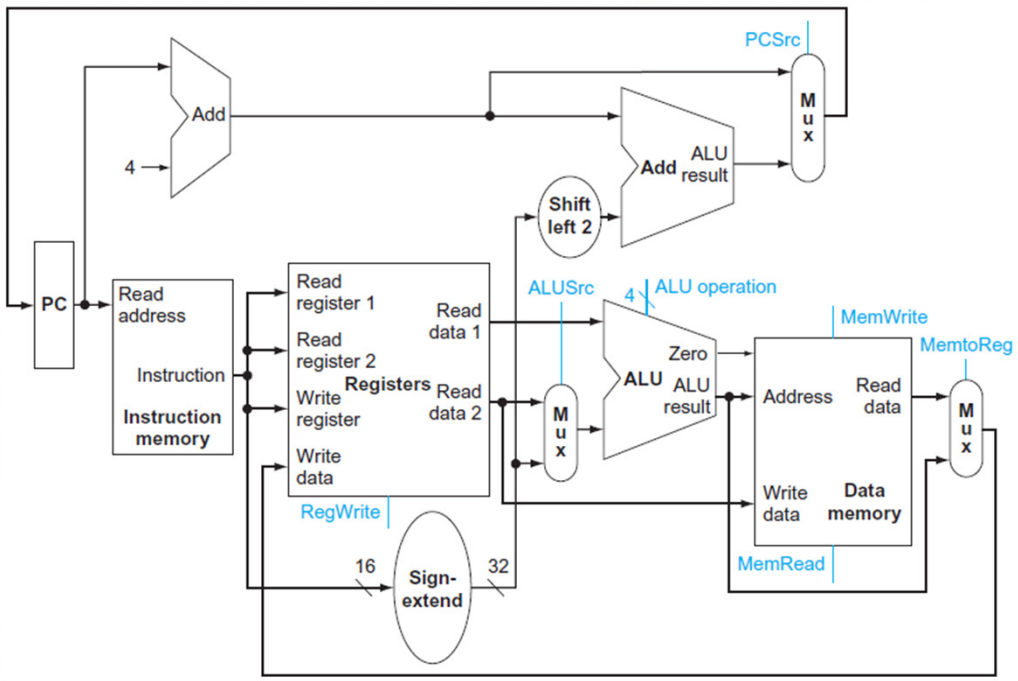
- Combinar los datapaths de:

a) Ciclo de fetch.
b) Datapath del paso 1.
c) Instrucciones de brinco.

# Datapath del brinco condicional

- Se agrega un multiplexor a la entrada del PC (contador de programa) controlado por una señal PCSrc:
- 0 – No hay brinco (PC ← PC + 4).
- 1 – Si hay brinco (PC ← salida del sumador).

# Datapath unificado paso 2

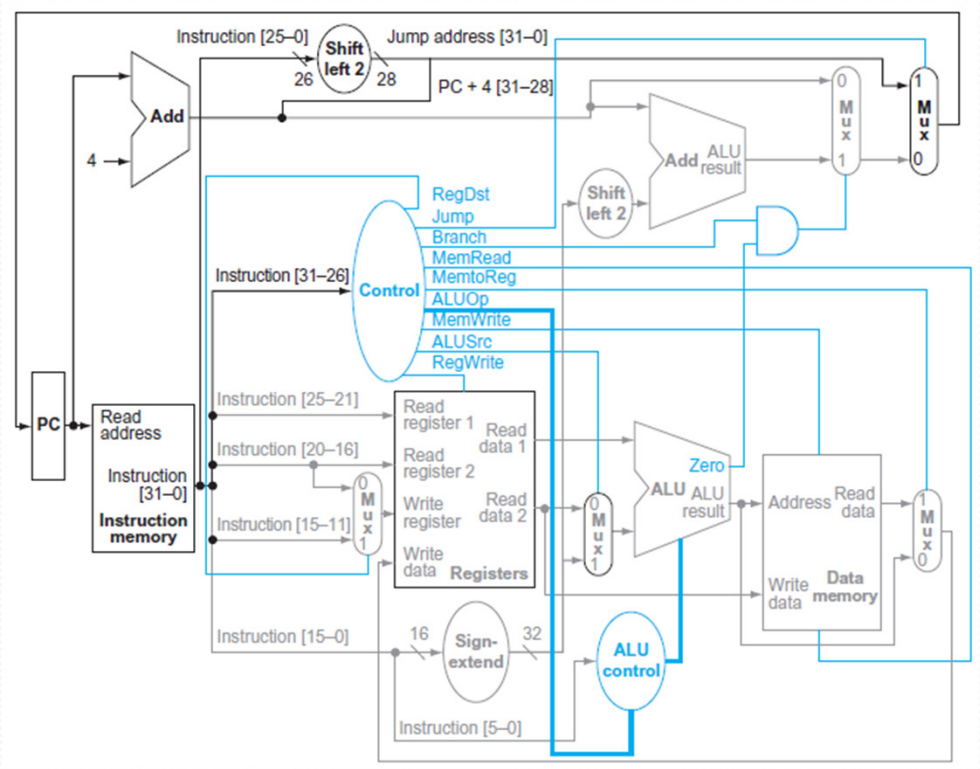



**Fuente: COD-HSI 5, p. 258**

# Agregar el control

- Los detalles están en el libro en la sección 4.4 (no es parte del curso).

# Procesador secuencial de un ciclo




**Fuente: COD-HSI 5, p. 271**

# Señales de control

| 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 |