¿Qué es Pipelining?

Es una técnica de implementación en la que se superpone la ejecución de varias instrucciones.

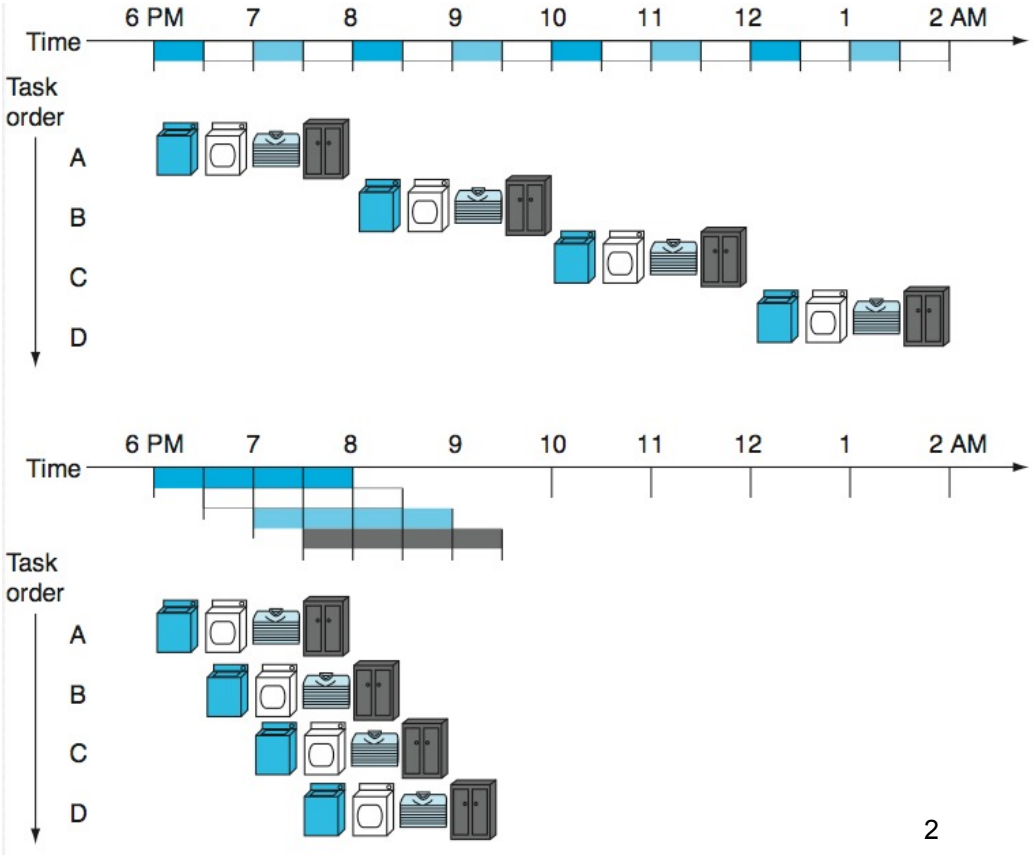
Aprovecha el paralelismo entre instrucciones en una corriente secuenial de instrucciones.

Se busca balancear la demora de cada etapa.

El caso ideal sería que el tiempo entre instrucciones sea =

$$\frac{\text{Tiempo entre instrucciones en la máquina sin pipe}}{\text{Número de etapas del pipe}}$$

¿Qué es Pipelining?



Etapas en las instrucciones MIPS

IF: Recuperación de instrucción desde memoria.

ID: Lectura de registros mientras se decodifica la instrucción. Debido al formato regular.

EX: Ejecución de la operación o cálculo de la dirección.

MEM: Acceso a operandos en memoria de datos.

WB: Escritura del resultado en los registros

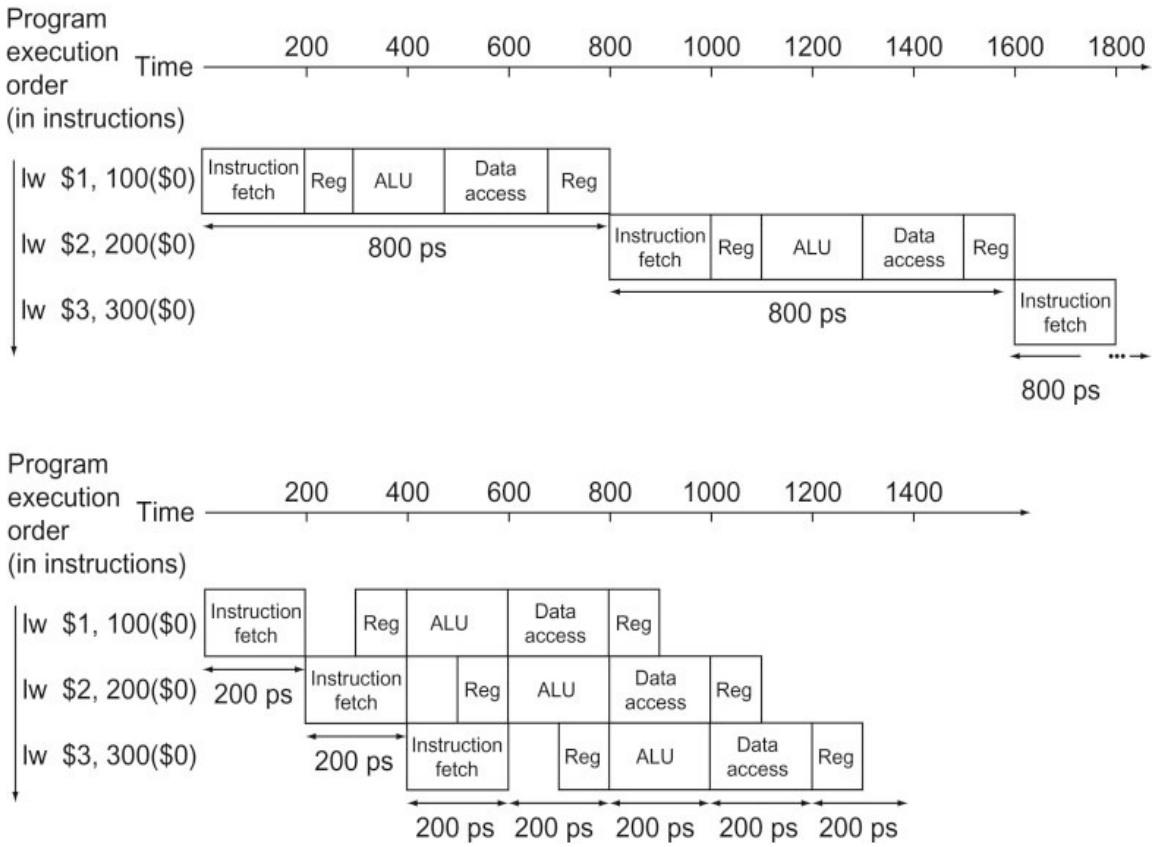
Uni-ciclo vs segmentación. Calcular el tiempo entre instrucciones

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

Tiempo total de cada instrucción se calcula a partir de la demora de cada componente

Se considera que los multiplexores, la unidad de control, el acceso al PC y la extensión de signo no tienen demora.

Uni-ciclo vs segmentación



Aceleración de la segmentación

Para una aceleración máxima se requieren condiciones ideales.

Las etapas deben estar balanceadas.

La máxima aceleración es cercana al número de etapas.

No se refleja con pocas instrucciones.

Considerando condiciones ideales, se cumple que:

$$\text{Tiempo entre instrucciones}_{\text{con pipe}} = \frac{\text{Tiempo entre instrucciones}_{\text{sin pipe}}}{\text{Número de etapas del pipe}}$$

Se mejora el rendimiento (throughput) de instrucciones y no se reduce la ejecución individual de instrucciones.

Conjunto de instrucciones para pipeline

- Igual longitud.
  - Facilita la recuperación.
- Pocos formatos.
  - Se leen registros y se decodifica en etapa 2
- Operandos en memoria solo en load y store.
  - Las direcciones se calculan en la etapa 3
- Operandos alineados en memoria.
  - Cada transferencia requiere un acceso.

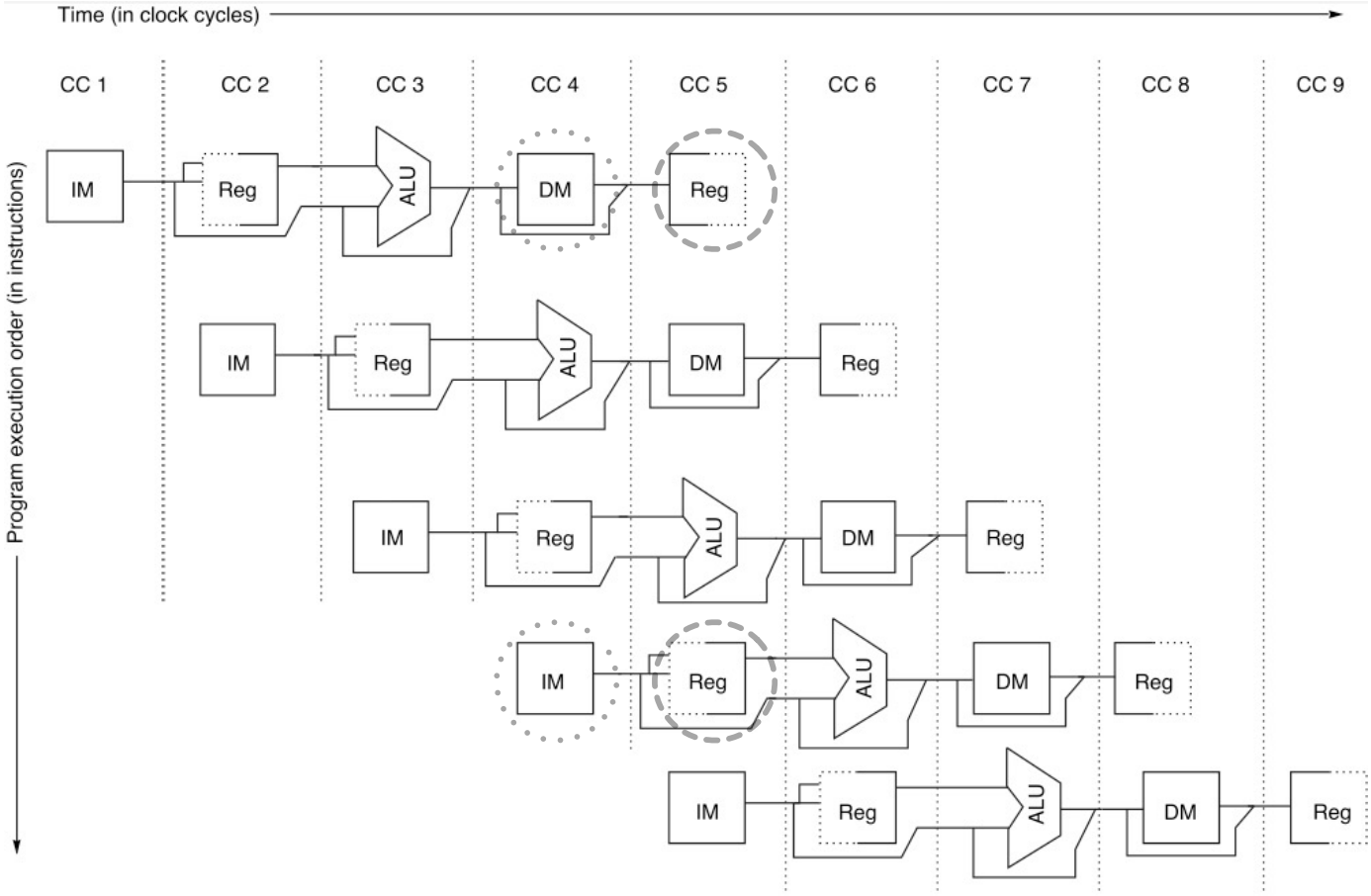
Problemas del pipe: los riesgos(hazards)

Riesgos estructurales. Una instrucción no puede ejecutar en el ciclo previsto porque el hardware no soporta la combinación de instrucciones dispuestas para ejecutar.

Riesgos de datos. Cuando una instrucción planificada no puede ejecutar en el ciclo previsto porque los datos que necesita aun no están disponibles.

Riesgos de control. Cuando la instrucción planificada no puede ejecutar en el ciclo previsto porque la instrucción recuperada no es la que se necesita.

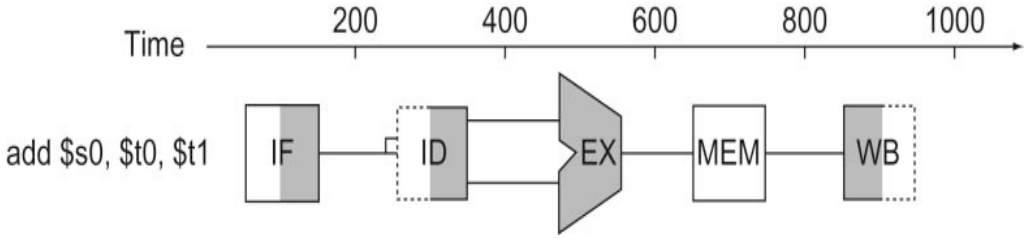
Riesgos estructurales



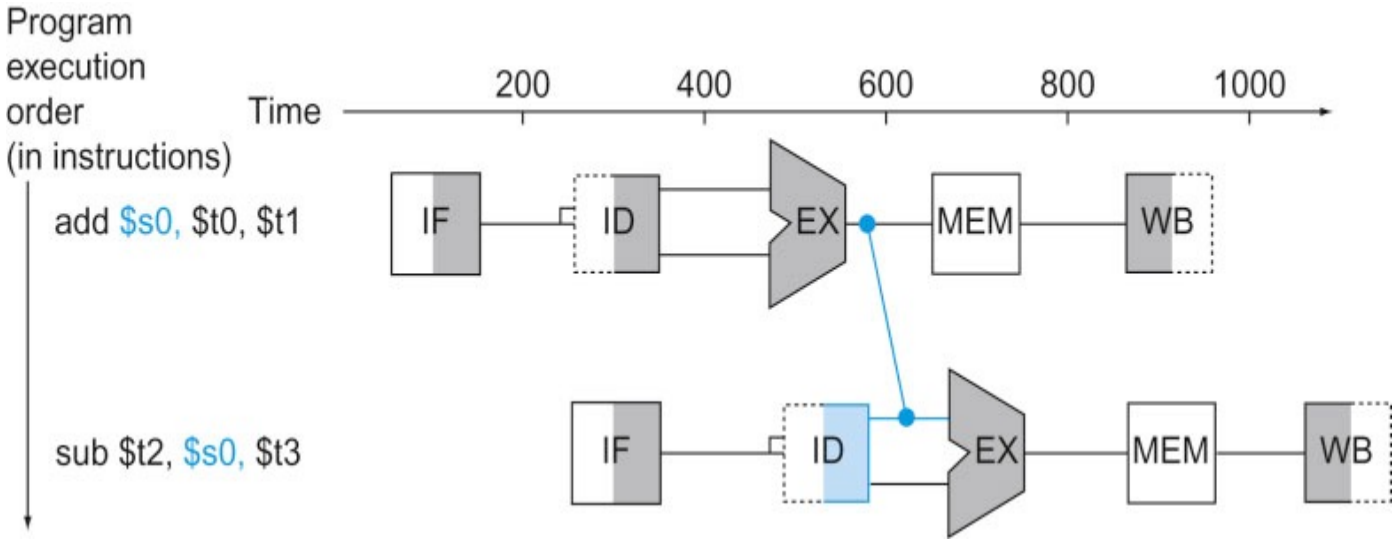
Los riesgos de datos

add $s0, $t0, $t1

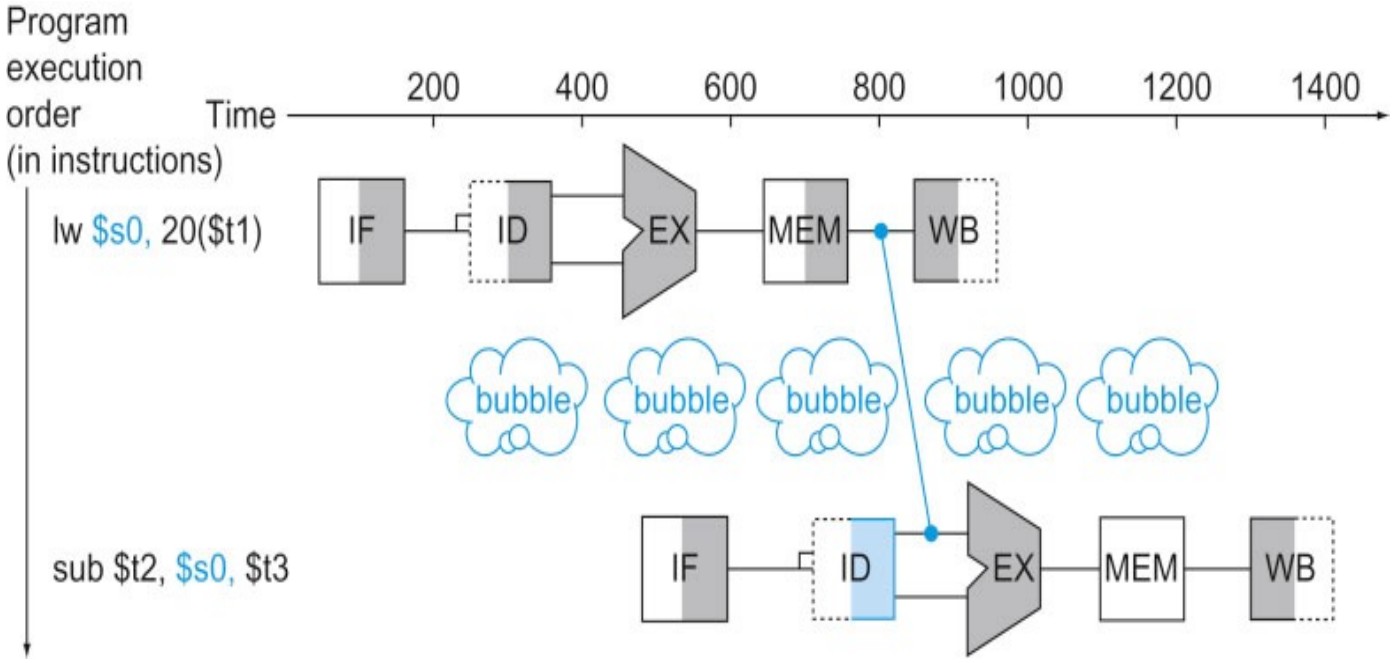
sub $t2, $s0, $t3



Adelantamiento



Riesgo load-use



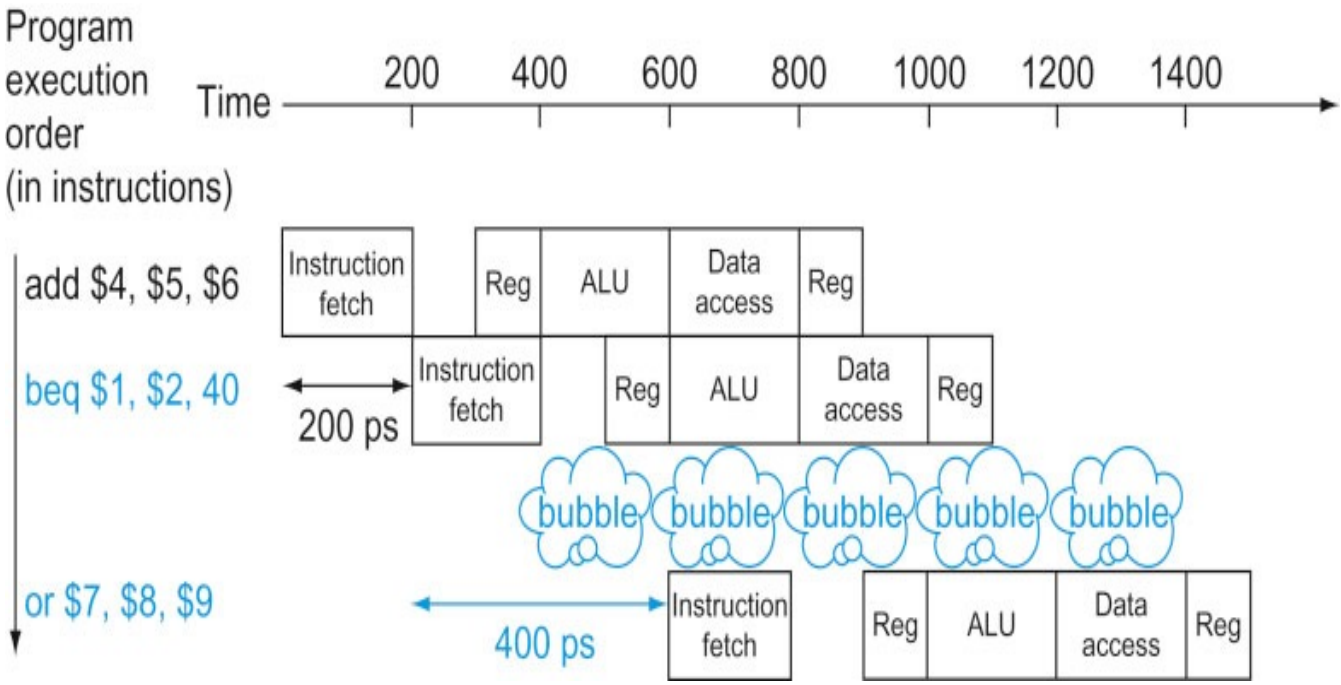
Reordenamiento de código

a = b + e;
c = b + f;

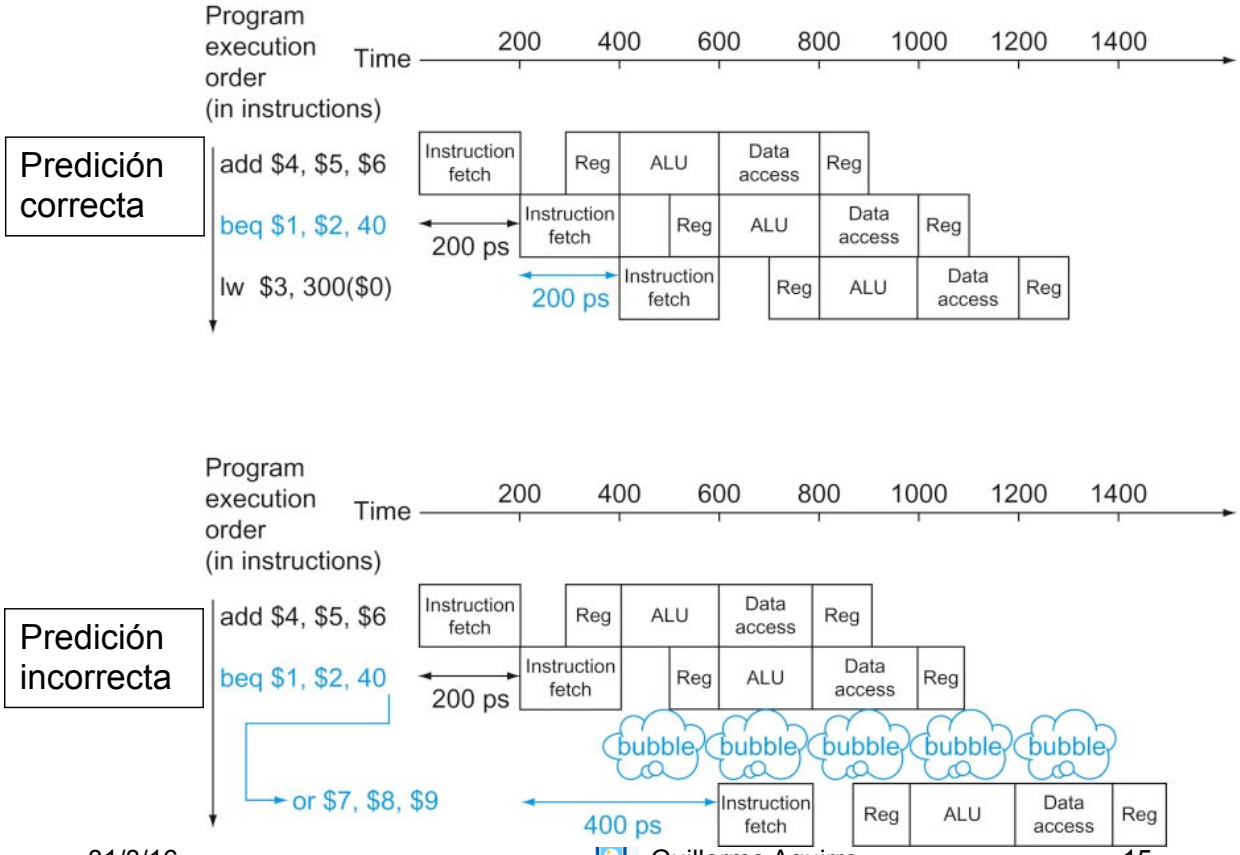
```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```


Riesgos de control (solución 1)



Riesgos de control-Salto no tomado



Predicción de saltos

Estática

En base al comportamiento típico del salto.

Ejemplo: branch en loops y sentencias if

Predecir los saltos hacia atrás como tomados.

Predecir los saltos hacia adelante como no tomados

Dinámica

Se registra el comportamiento de cada branch

La tendencia se mantendrá en el futuro

Salto Demorado - MIPS

Siempre se ejecuta la instrucción en secuencia

El salto se hace después de la demora.

En la demora van instrucciones neutrales.

La demora es un ciclo.

Nociones de segmentación: Resumen

Con la segmentación se incrementa el rendimiento de las instrucciones lo cual mejora el desempeño.

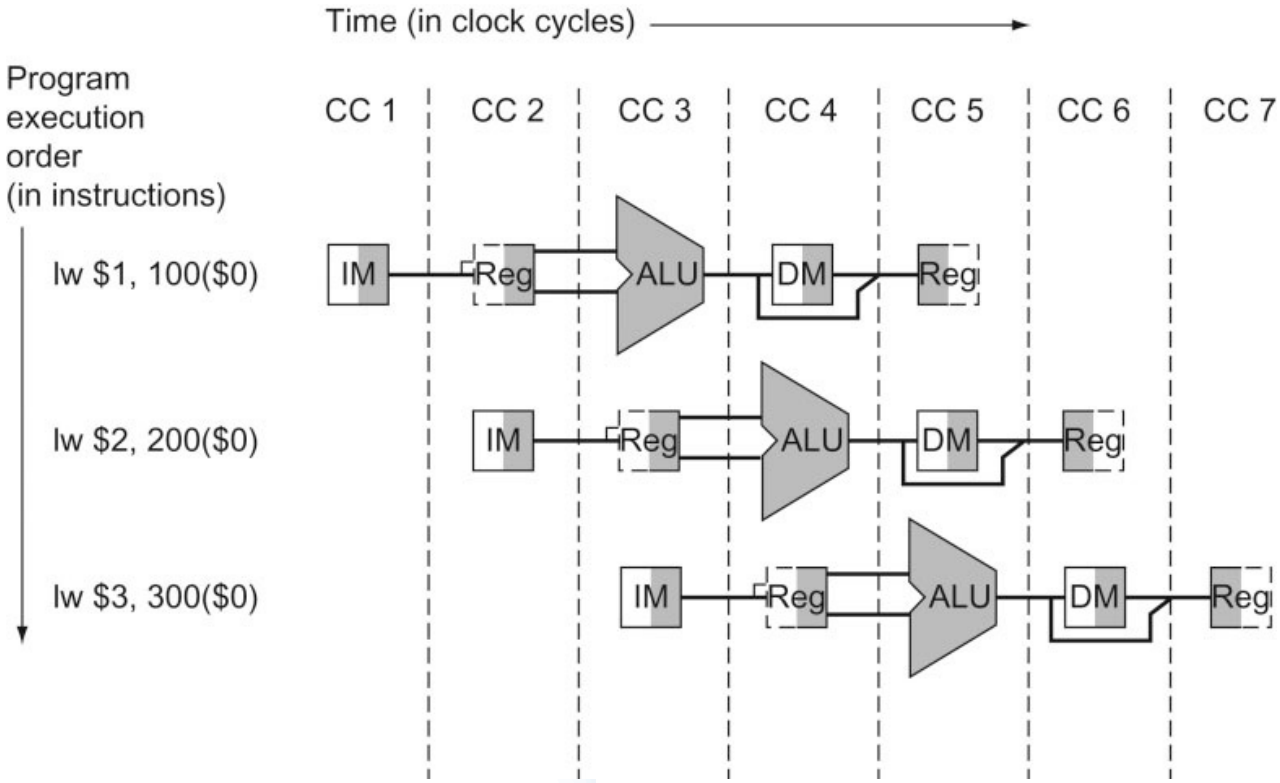
- Explota el paralelismo a nivel de instrucción.
- Cada instrucción tiene la misma latencia.

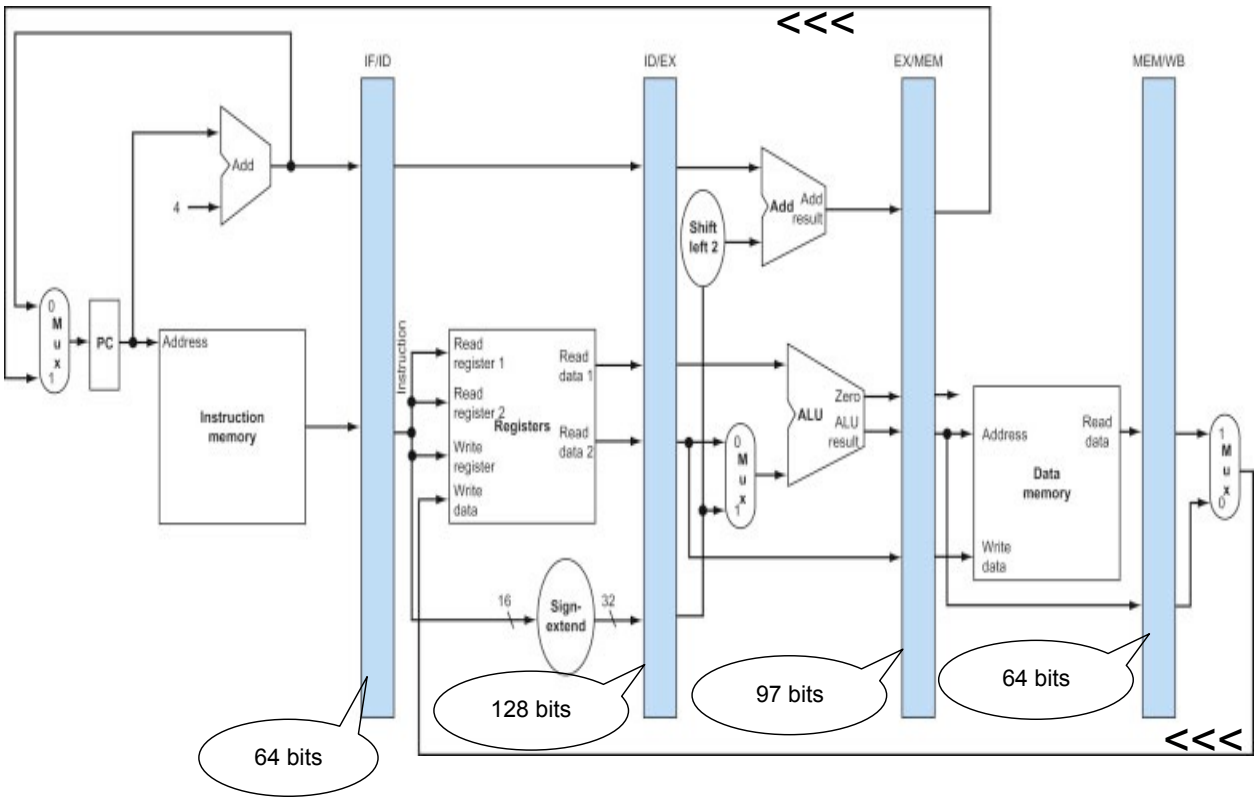
Aparecen los riesgos:

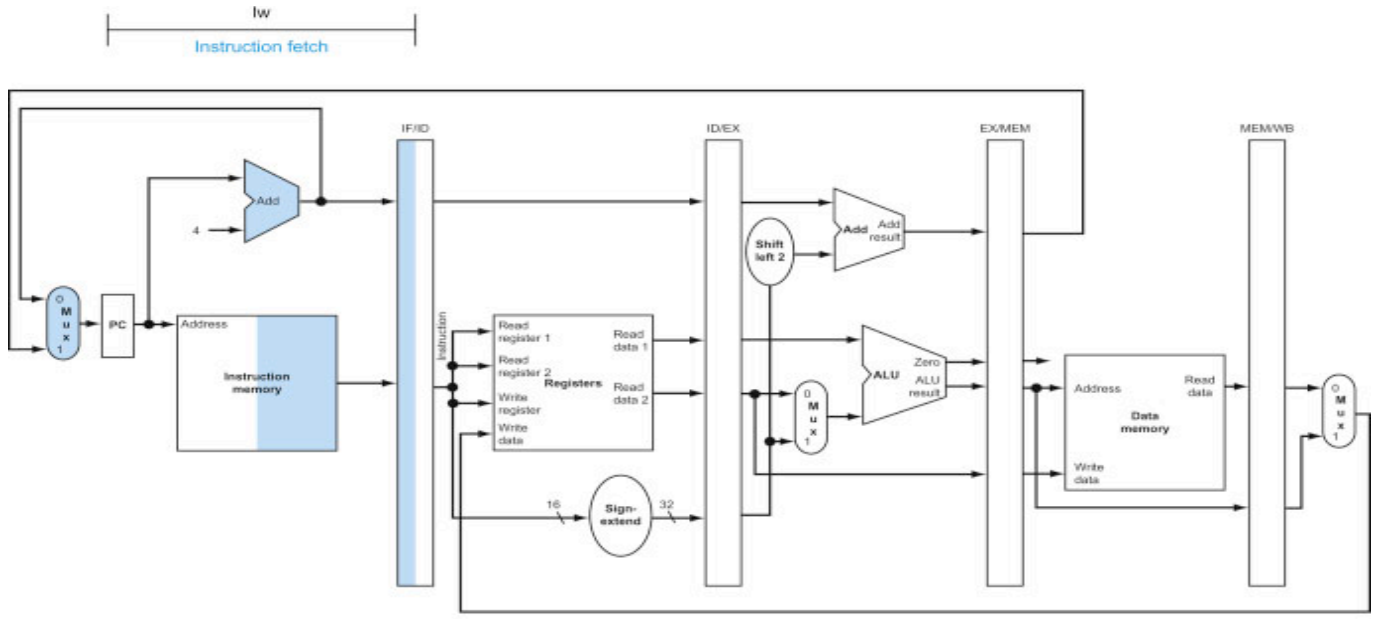
- Estructurales, de datos y de control

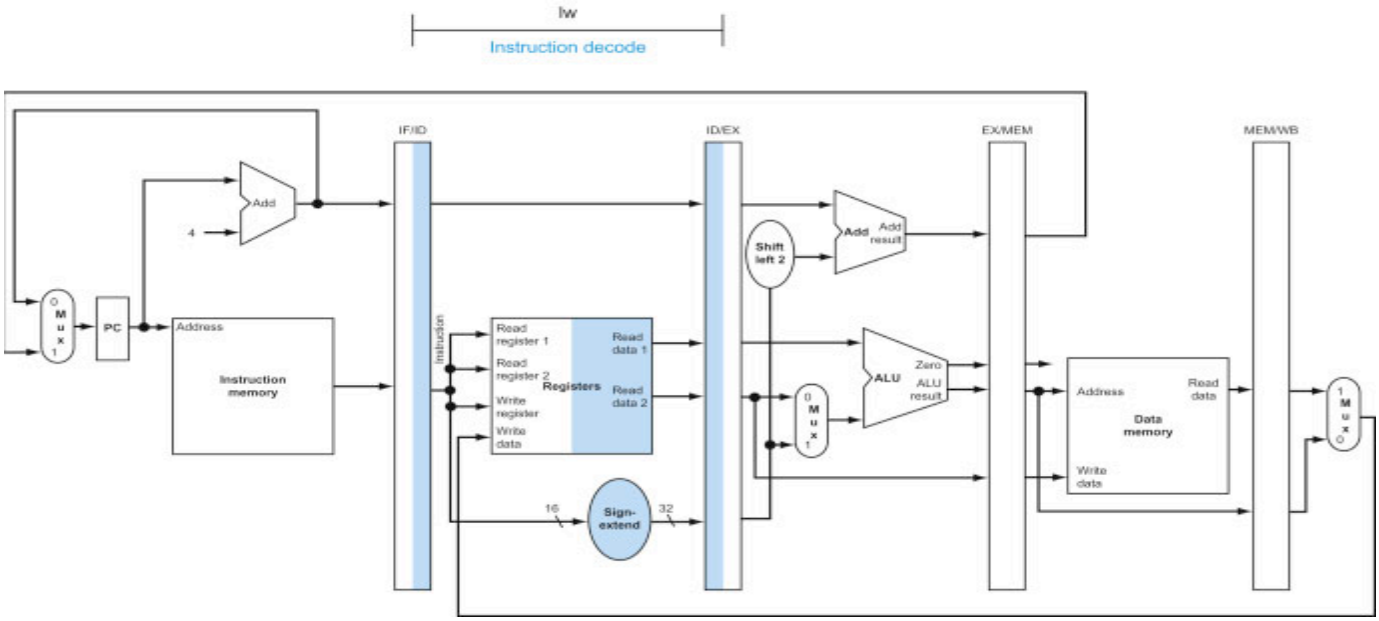
El diseño del conjunto de instrucciones afecta la complejidad para implementar la segmentación.

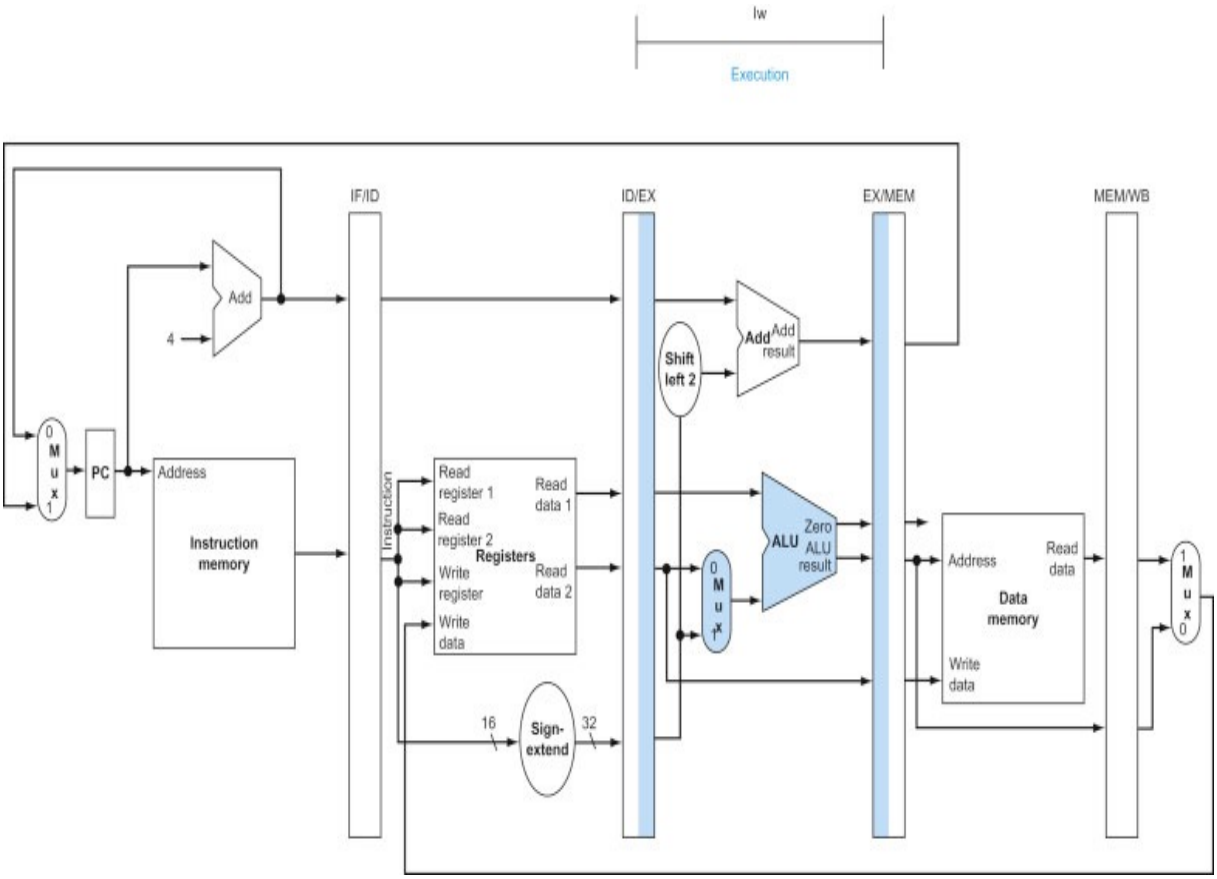
Ejecución segmentada

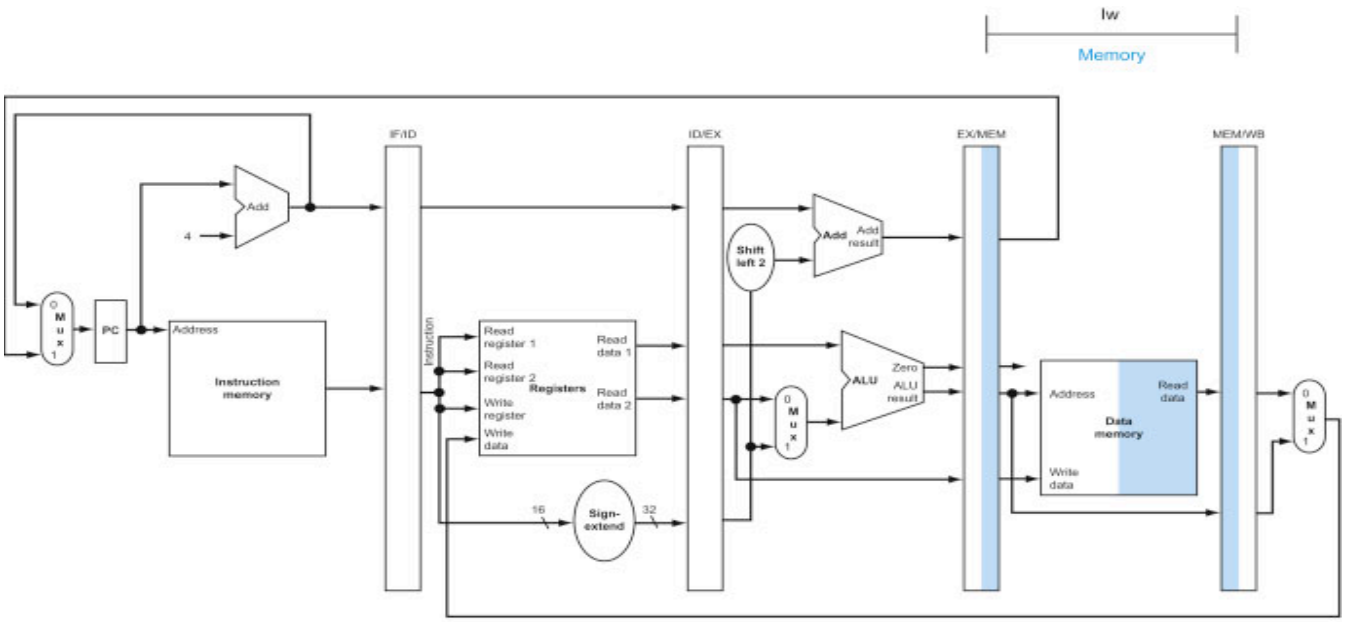


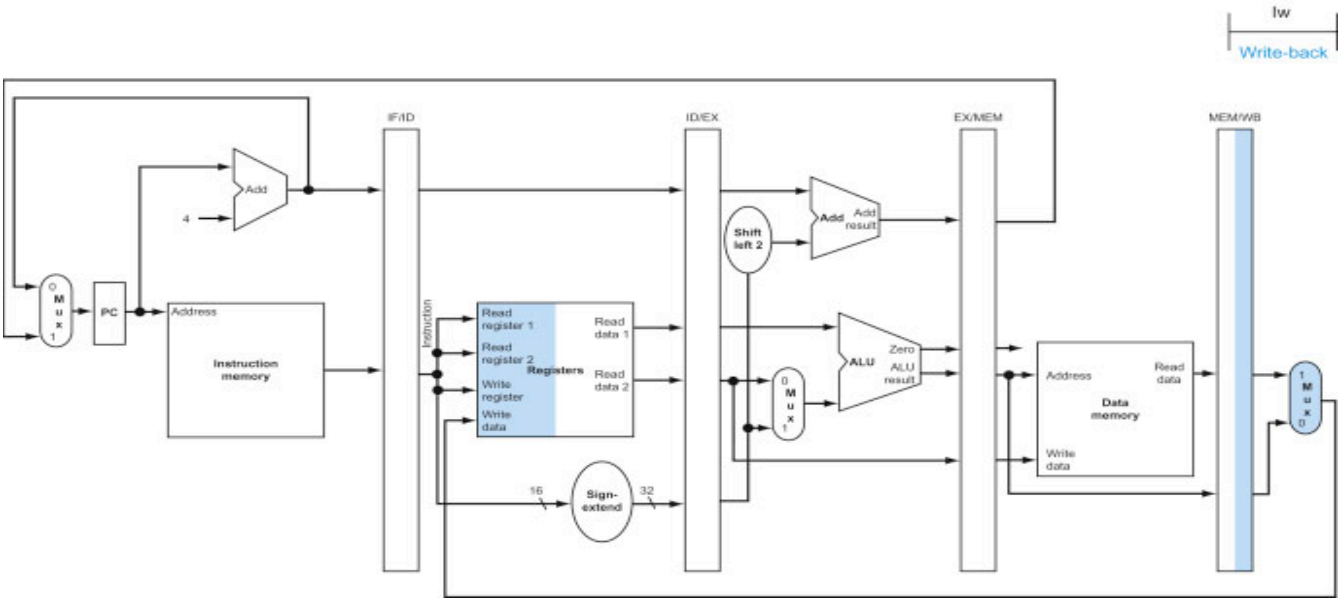


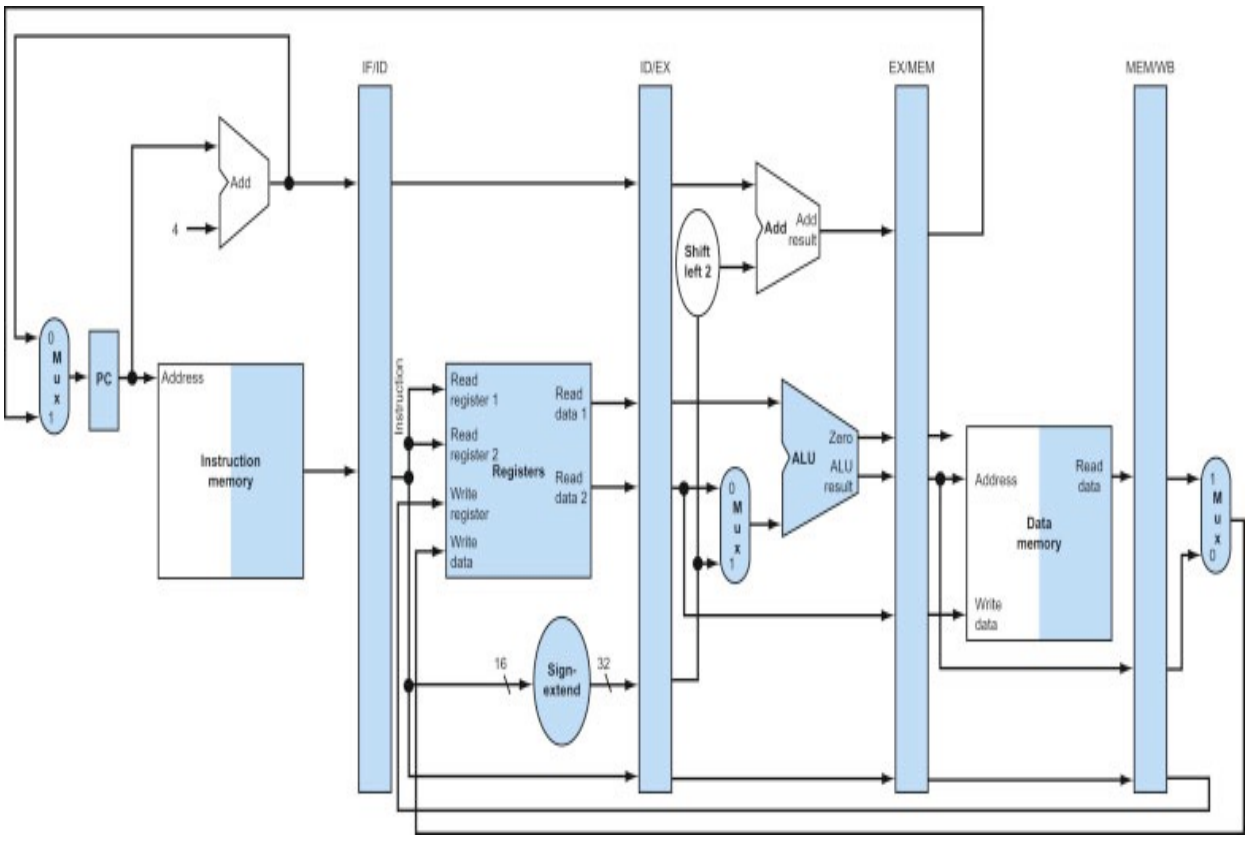

lw
Instruction fetch
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

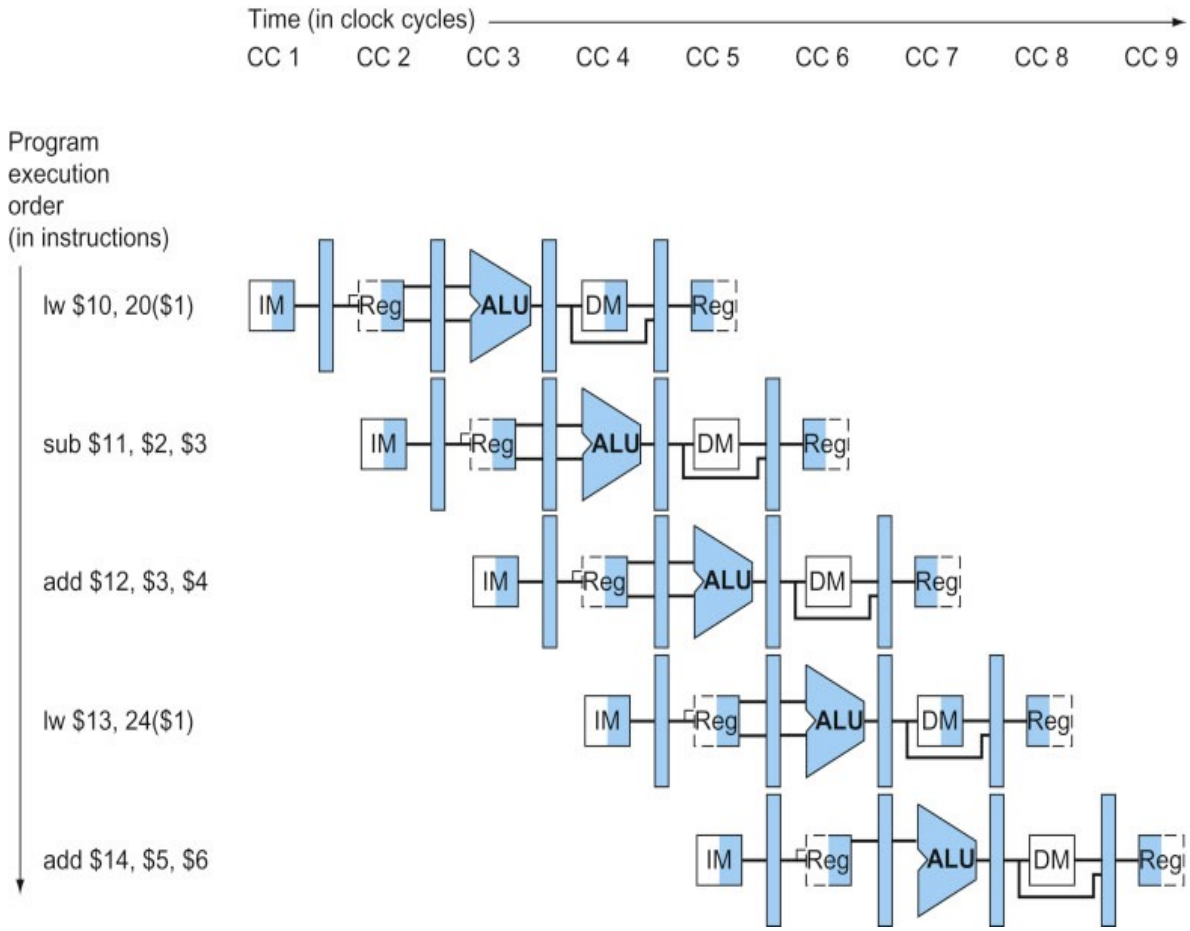

lw
Instruction decode
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

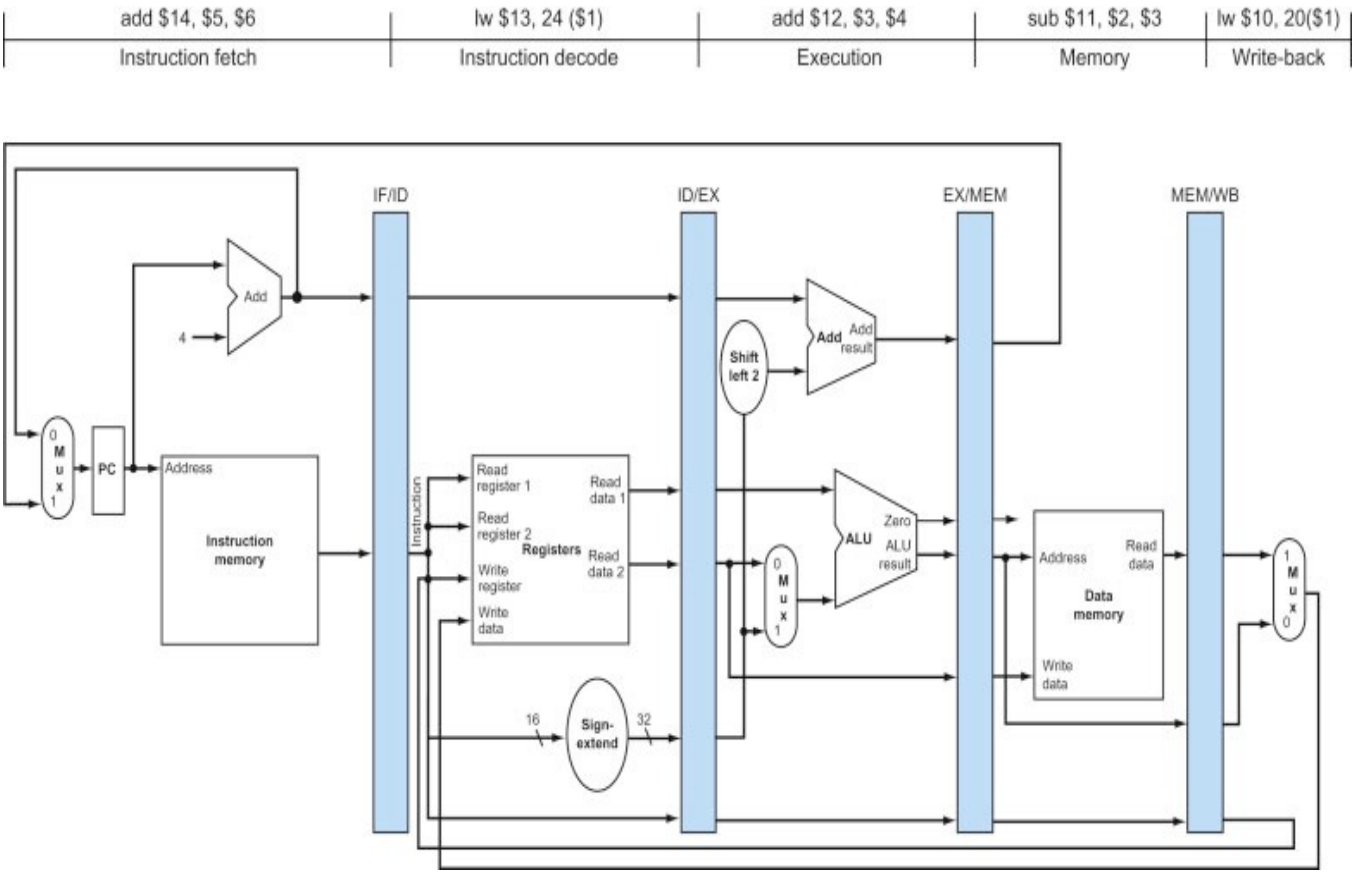

lw
Execution
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Data memory
Read data
Write data


lw
Memory
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
16
Sign-extend
32
Data memory
Read data
Write data

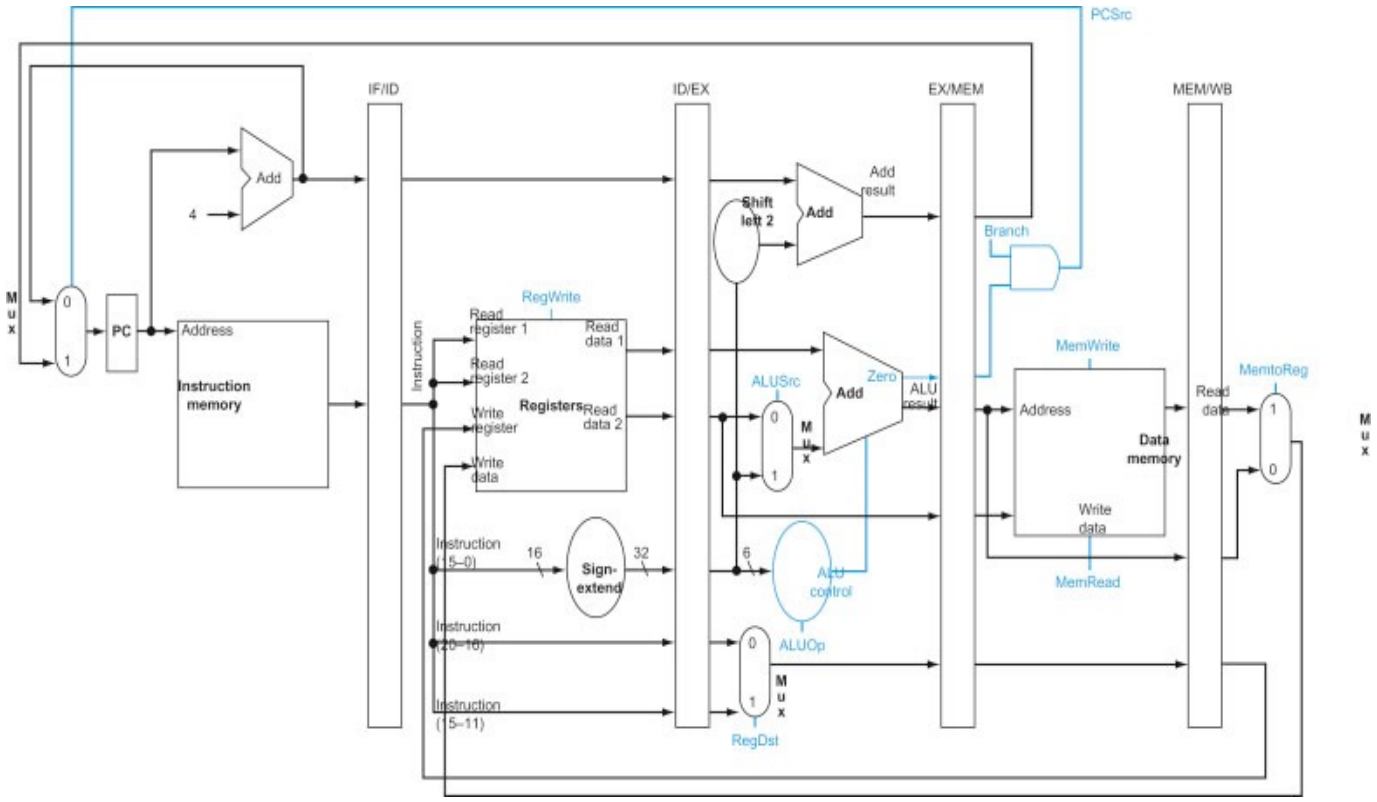

lw
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Data memory
Read data
Sign-extend
16
32


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Data memory
Read data
Sign-extend
16
32


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
lw $10, 20($1)
sub $11, $2, $3
add $12, $3, $4
lw $13, 24($1)
add $14, $5, $6
IM
Reg
ALU
DM
Reg


add $14, $5, $6
lw $13, 24 ($1)
add $12, $3, $4
sub $11, $2, $3
lw $10, 20($1)
Instruction fetch
Instruction decode
Execution
Memory
Write-back
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign-extend
32

Señales de control del camino de datos segmentado



Bits de control de la ALU

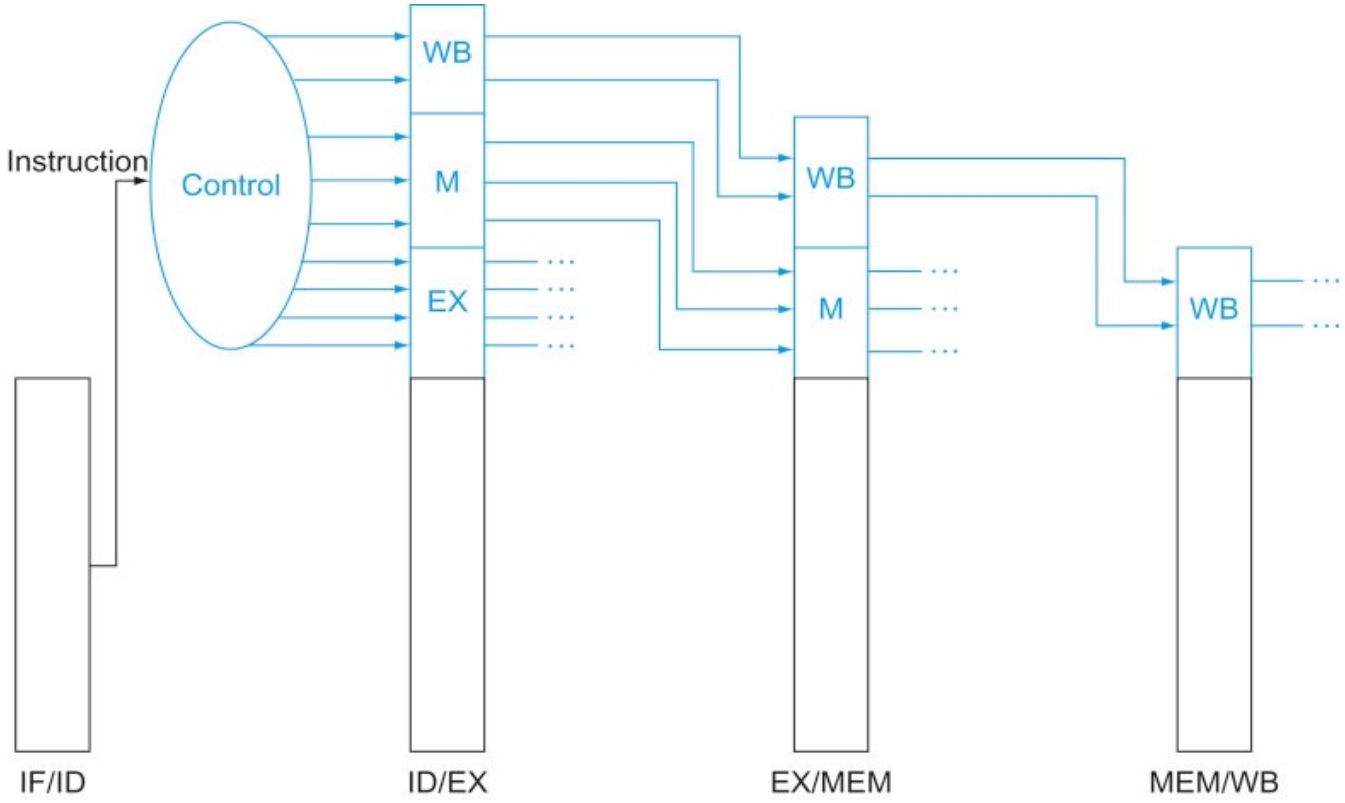
Instruction opcode	ALUOp	Instruction operation	Function code	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

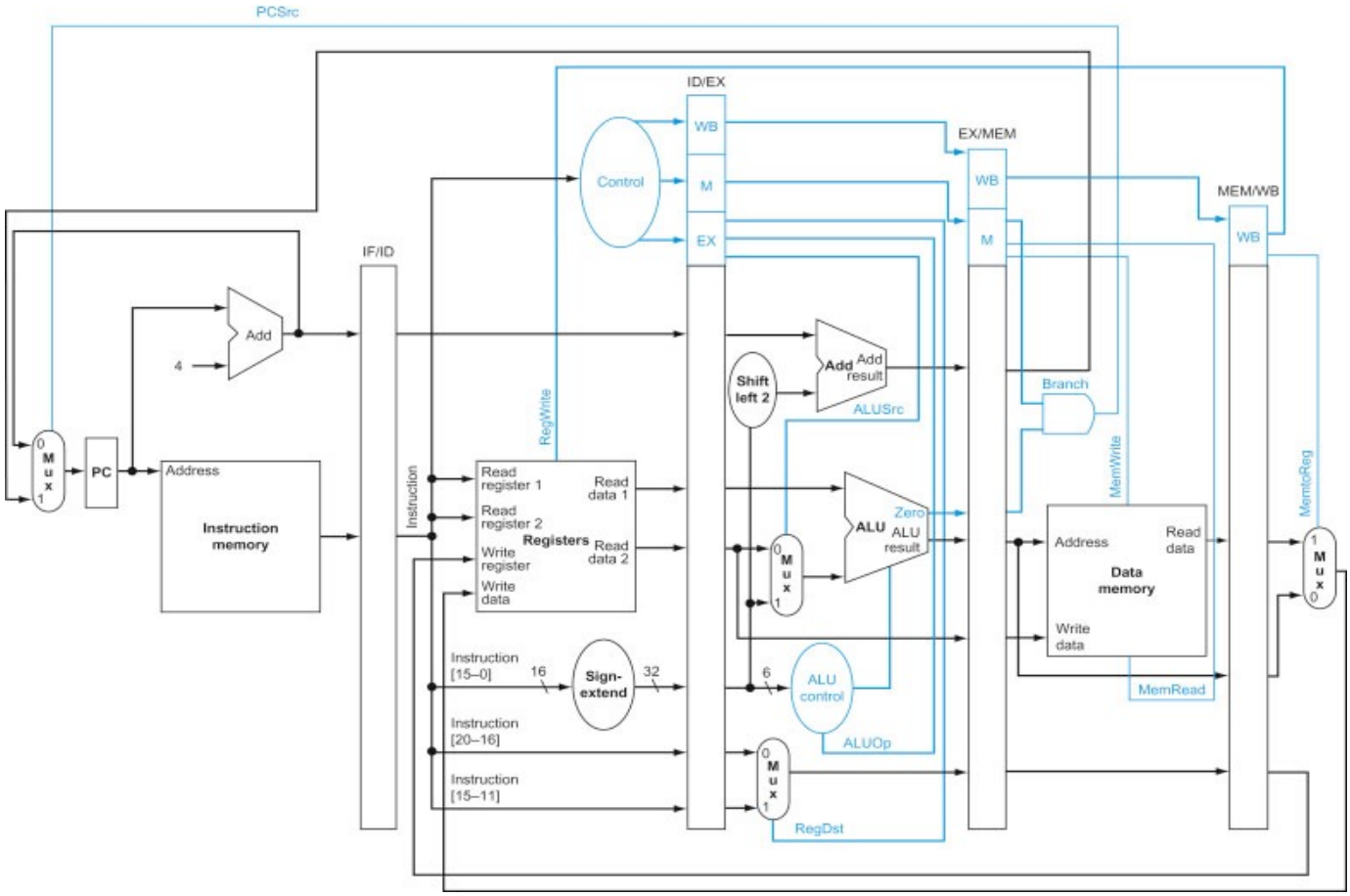
Señales de control

Signal name	Effect when deasserted (0)	Effect when asserted (1)
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Señales de control de las tres etapas finales

Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	RegDst	ALUOp1	ALUOp0	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X


Instruction
Control
WB
M
EX
WB
M
WB
IF/ID
ID/EX
EX/MEM
MEM/WB


PCSrc
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
IF/ID
Add
4
M u x
PC
Address
Instruction memory
Instruction
RegWrite
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Instruction [15–0]
16
Sign-extend
32
Instruction [20–16]
Instruction [15–11]
Shift left 2
Add
Add result
ALUSrc
ALU
Zero
ALU result
6
ALU control
ALUOp
RegDst
Branch
MemWrite
Address
Read data
Data memory
Write data
MemRead
MemtoReg