



Tema 6:
Diseño del procesador

Fundamentos de computadores

Daniel Mozos Muñoz
*Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid*



Contenidos

- Rendimiento de los procesadores
- Formato de las instrucciones RISC-V
- Diseño de la ruta de datos
- Procesador monociclo
 - Ruta de datos
 - Unidad de control
- Procesador multiciclo
 - Ruta de datos
 - Unidad de control
- **Bibliografía:**
 - Computer Organization and Design. The Hardware/Software Interface. RISC-V edition. David A. Patterson & John L. Hennessy, Morgan Kaufmann 2018.
 - Digital design and computer architecture. RISC-V edition. S.L. Harris y D.M. Harris. Morgan Kaufmann, 2021



Rendimiento de los procesadores

versión 2021

tema6:
Sistemas combinacionales

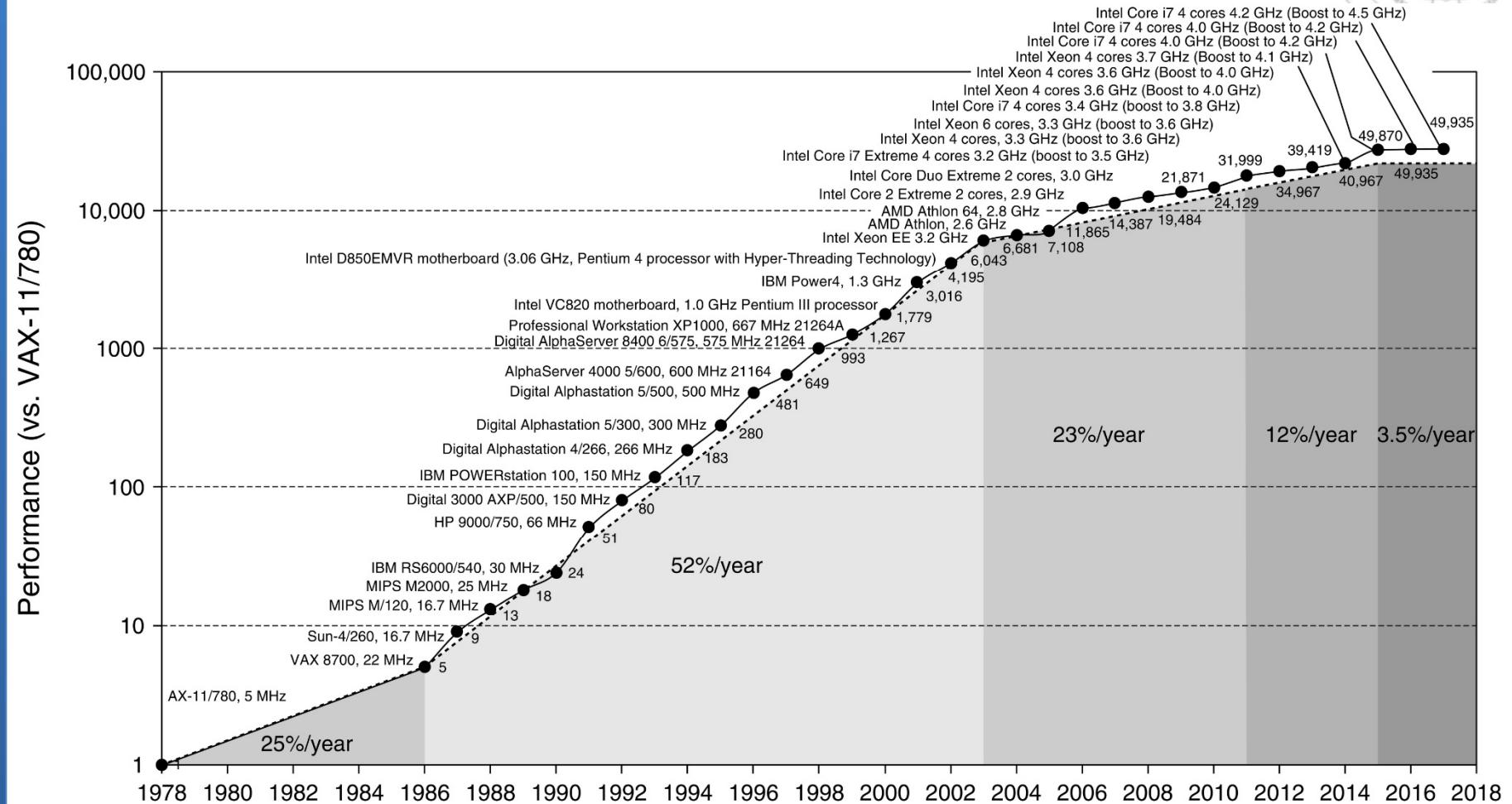
FC

3

“Los buenos programadores se han preocupado siempre por el rendimiento de sus programas porque la rápida obtención de resultados es crucial para crear programas de éxito”

D. A. Patterson y J. L. Hennessy

Rendimiento de los procesadores



- J.L. Hennessy y D.A. Patterson. *Computer Architecture 6th ed.*



TOP 500 20/11/2020

versión 2021

tema6:
Sistemas combinacionales

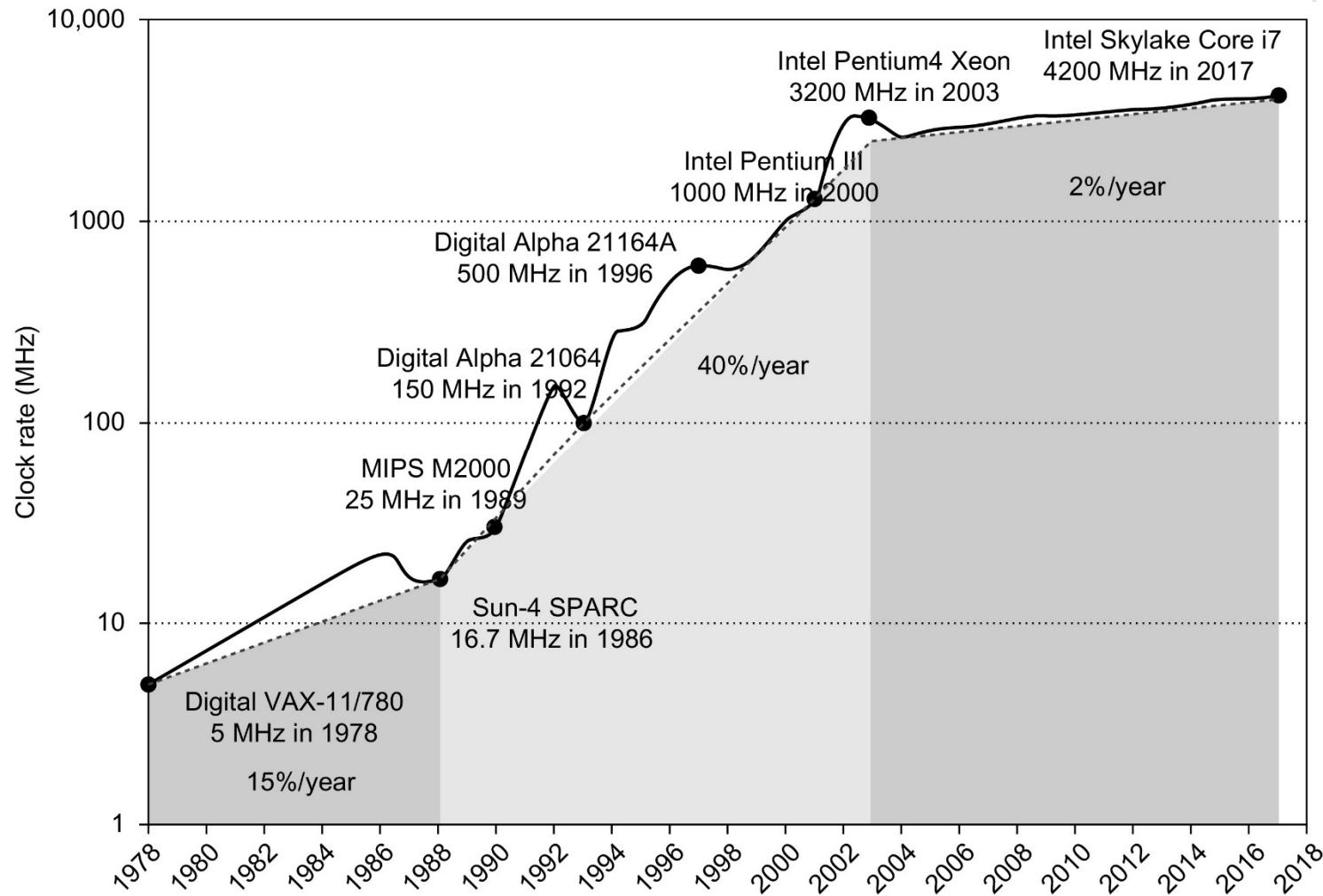
FC

5

Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu <u>RIKEN Center for Computational Science</u> Japan	7,299,072	415,530.0	513,854.7	28,335
2	<u>Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory</u> United States	2,414,592	148,600.0	200,794.9	10,096
3	<u>Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL</u> United States	1,572,480	94,640.0	125,712.0	7,438
4	<u>Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC</u> <u>National Supercomputing Center in Wuxi</u> China	10,649,600	93,014.6	125,435.9	15,371
5	<u>Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia</u> <u>NVIDIA Corporation</u> United States	555,520	63,460.0	79,215.0	2,646



Rendimiento de los procesadores



- J.L. Hennessy y D.A. Patterson. *Computer Architecture 6th ed.*



Rendimiento de los procesadores

- ¿Cuántos ciclos tarda en ejecutarse este programa?

```
lw x1, 24(x10)
lw x2, 12(x3)
add x0, x2, x7
beq x1, x2, loop
sub x0, x2, x7
sw x3, 3(x5)
```

- Depende del procesador: por ejemplo en el RISC-V multiciclo

lw x1, 24(x10)	→ 5 ciclos
lw x2, 12(x3)	→ 5 ciclos
add x0, x2, x7	→ 4 ciclos
beq x1, x2, loop	→ 4 ciclos
sub x0, x2, x7	→ 4 ciclos
sw x3, 3(x5)	→ 4 ciclos

- ¿Y cuánto Tiempo?

- Depende de la frecuencia del reloj y de si hay segmentación o no.



Rendimiento de los procesadores

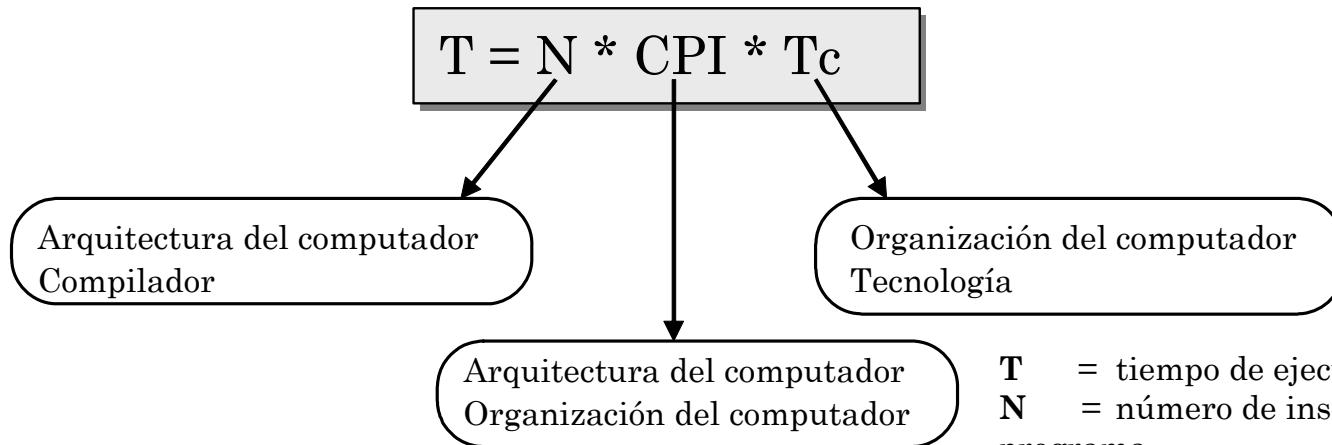
versión 2021

tema 6:
Diseño del procesador

FC

8

■ Tiempo de ejecución de un programa



T = tiempo de ejecución del programa

N = número de instrucciones del programa

CPI = número medio de ciclos por

■ CPI = Ciclos por instrucción

- Una instrucción necesita varios ciclos de reloj para su ejecución
- Diferentes instrucciones tardan diferente número de ciclos
- El **CPI** es una suma ponderada del número de ciclos que tarda por separado cada tipo de instrucción



Formato de las instrucciones

Principios de diseño

1. La regularidad facilita la simplicidad de diseño
 2. Hacer rápido el caso común
 3. Más pequeño es más rápido
 4. Buen diseño exige buenos compromisos
- **Principio de diseño 1: La regularidad facilita la simplicidad de diseño**
 - Datos de 32-bits, instrucciones de 32-bits
 - Por simplicidad de diseño, se preferiría un sólo formato pero las instrucciones tienen diferentes necesidades
 - **Principio de diseño 4: Buen diseño exige buenos compromisos**
 - Tener varios formatos da flexibilidad
 - add, sub: usan 3 registros como operandos
 - lw, sw: usan 2 registros y una constante como operandos
 - El número de formatos debe mantenerse reducido para cumplir con los principios 1 y 4



Formato de las instrucciones

- Instrucciones de 32 bits
- **4 tipos de formatos de instrucción:**
 - Tipo R
 - Tipo I
 - Tipo S/B
 - Tipo U/J



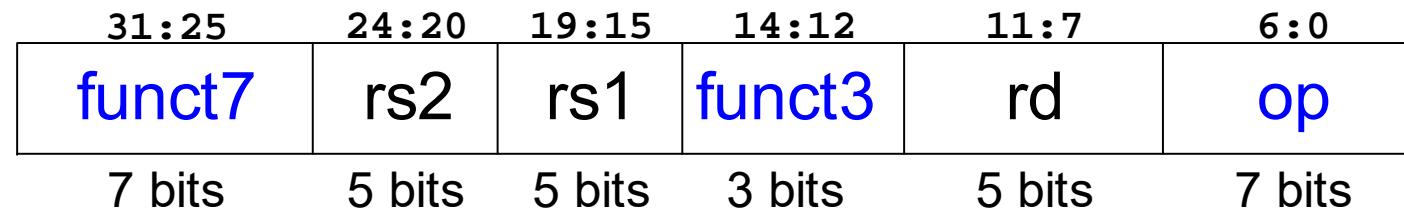
Formato de las instrucciones: Tipo R

versión 2021

tema 6:
Diseño del procesador

- 3 operandos en registros:
 - rs1, rs2: registros fuente
 - rd: registro destino
- Otros campos:
 - op: Código de operación
 - funct7, funct3: la *función* (7 bits y 3-bits, respectivamente) junto a *opcode*, indica al computador qué operación realizar

R-Type





Formato de las instrucciones: Tipo R

- Ejemplos de codificaciones Tipo-R

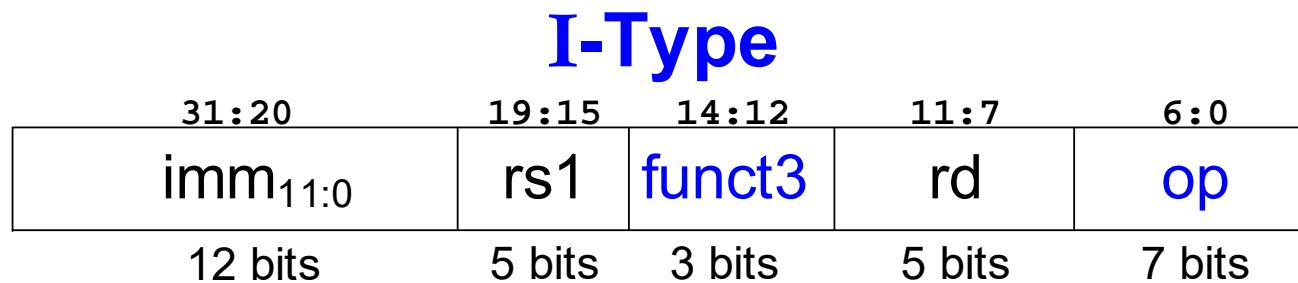
Assembly						Field Values						Machine Code					
	funct7	rs2	rs1	funct3	rd	op		funct7	rs2	rs1	funct3	rd	op				
add s2, s3, s4	0	20	19	0	18	51		0000 000	10100	10011	000	10010	011 0011				(0x01498933)
add x18,x19,x20								0100 000	00111	00110	000	00101	011 0011				
sub t0, t1, t2	32	7	6	0	5	51											(0x407302B3)
sub x5, x6, x7																	

Assembly						Field Values						Machine Code					
	funct7	rs2	rs1	funct3	rd	op		funct7	rs2	rs1	funct3	rd	op				
sll s7, t0, s1	0	9	5	1	23	51		0000 000	01001	00101	001	10111	011 0011				(0x00929BB3)
sll x23,x5, x9								0000 000	11010	11001	100	11000	011 0011				
xor s8, s9, s10	0	26	25	4	24	51											(0x01ACCC33)
xor x24,x25,x26								0100 000	11101	00111	101	00110	001 0011				
srai t1, t2, 29	32	29	7	5	6	19											(0x41D3D313)
srai x6, x7, 29																	



Formato de las instrucciones: Tipo I

- 3 operandos:
 - rs1: registro fuente
 - rd: registro destino
 - inm: Valor inmediato de 12 bits en C2
- Otros campos:
 - op: Código de operación
 - funct3: la *función* (3-bits) junto a *opcode*, indica al computador qué operación realizar





Formato de las instrucciones: Tipo I

- Ejemplos de codificaciones Tipo-I

Assembly

```
addi s0, s1, 12  
addi x8, x9, 12  
addi s2, t1, -14  
addi x18,x6, -14  
lw t2, -6($3)  
lw x7, -6(x19)  
lh s1, 27(zero)  
lh x9, 27(x0)  
lb s4, 0x1F(s4)  
lb x20,0x1F(x20)
```

Field Values

imm _{11:0}	rs1	funct3	rd	op
12	9	0	8	19
-14	6	0	18	19
-6	19	2	7	3
27	0	1	9	3
0x1F	20	0	20	3

Machine Code

imm _{11:0}	rs1	funct3	rd	op	
0000 0000 1100	01001	000	01000	001 0011	(0x00C48413)
1111 1111 0010	00110	000	10010	001 0011	(0xFF230913)
1111 1111 1010	10011	010	00111	000 0011	(0xFFA9A383)
0000 0001 1011	00000	001	01001	000 0011	(0x01B01483)
0000 0001 1111	10100	000	10100	000 0011	(0x01FA0A03)



Formato de las instrucciones: Tipo S y B

- Se diferencian solo en la codificación inmediata

31:25	24:20	19:15	14:12	11:7	6:0
$\text{imm}_{11:5}$	rs2	rs1	funct3	$\text{imm}_{4:0}$	op
$\text{imm}_{12,10:5}$	rs2	rs1	funct3	$\text{imm}_{4:1,11}$	op

S-Type

B-Type

- Tipo store:
 - 3 operandos:
 - rs1 : registro base
 - rs2 : Valor a guardar en memoria
 - inm : Valor inmediato de 12 bits en C2
 - Otros campos:
 - op : Código de operación
 - funct3 : la *función* (3-bits) junto a *opcode*, indica al computador qué operación realizar
- Ejemplos de codificaciones Tipo-S

Assembly

```
sw t2, -6($s3)
sw x7, -6(x19)
sh s4, 23(t0)
sh x20,23(x5)
sb t5, 0x2D(zero)
sb x30,0x2D(x0)
```

Field Values

$\text{imm}_{11:5}$	rs2	rs1	funct3	$\text{imm}_{4:0}$	op
1111 111	7	19	2	11010	35
0000 000	20	5	1	10111	35
0000 001	30	0	0	01101	35

Machine Code

$\text{imm}_{11:5}$	rs2	rs1	funct3	$\text{imm}_{4:0}$	op	
1111 111	00111	10011	010	11010	010 0011	(0xFE79AD23)
0000 000	10100	00101	001	10111	010 0011	(0x01429BA3)
0000 001	11110	00000	000	01101	010 0011	(0x03E006A3)



Formato de las instrucciones: Tipo S y B

- Tipo branch:
 - 3 operandos:
 - **rs1:** registro Fuente 1
 - **rs2:** registro Fuente 2
 - **inm:** Valor inmediato de 12 bits en C2, desplazamiento
 - Otros campos:
 - **op:** Código de operación
 - **funct3:** la *función* (3-bits) junto a *opcode*, indica al computador qué operación realizar

B-Type

31:25	24:20	19:15	14:12	11:7	6:0
$\text{imm}_{12,10:5}$	rs2	rs1	funct3	$\text{imm}_{4:1,11}$	op

7 bits 5 bits 5 bits 3 bits 5 bits 7 bits



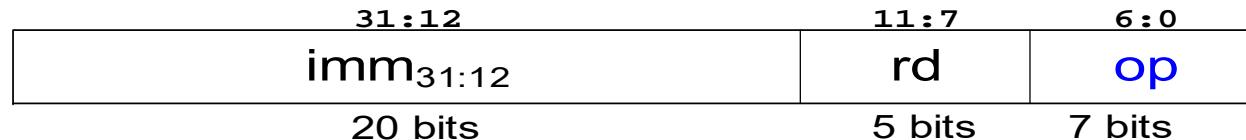
Formato de las instrucciones: Tipo U

■ Tipo-U. Upper immediate

Usado por lui

- 2 operandos:
 - rd : registro destino
 - $\text{imm}_{20,10:1,11,19:12}$: 20 bits (20:1) of a 21-bit immediate
- Otros campos:
 - op: Código de operación

U-Type



■ Ejemplos de codificaciones Tipo-U

Assembly

lui s5, 0x8CDEF
lui x21, 0x8CDEF

Field Values

$\text{imm}_{31:12}$	rd	op
0x8CDEF	21	55

20 bits 5 bits 7 bits

Machine Code

$\text{imm}_{31:12}$	rd	op	
1000 1100 1101 1110 1111	10101	011 0111	(0x8CDEFAB7)

20 bits 5 bits 7 bits

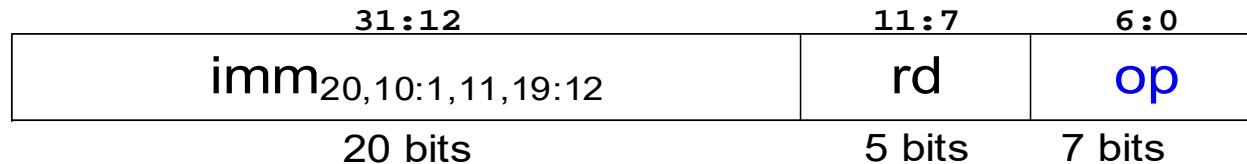


Formato de las instrucciones: Tipo J

Usado por la instrucción jump and link

- 2 operandos:
 - rd: registro destino
 - imm_{31:12}: 20 bits más significativos del inmediato de 32 bits
- Otros campos:
 - op: Código de operación

J-Type



- Nota: jalr es I-type, no j-type, para especificar rs



Formato de las instrucciones

versión 2021

tema 6:
Diseño del procesador

FC

19

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
funct7	rs2	rs1	funct3	rd	op
imm _{11:0}		rs1	funct3	rd	op
imm _{11:5}	rs2	rs1	funct3	imm _{4:0}	op
imm _{12,10:5}	rs2	rs1	funct3	imm _{4:1,11}	op
imm _{31:12}				rd	op
imm _{20,10:1,11,19:12}				rd	op
20 bits			5 bits		7 bits

R-Type

I-Type

S-Type

B-Type

U-Type

J-Type



Diseño de la ruta de datos

Metodología de diseño

- **Paso 1:** Analizar el repertorio de instrucciones para obtener los requisitos de la ruta de datos
 - La ruta de datos debe incluir tantos **elementos de almacenamiento** como registros sean visibles por el programador. Además puede tener otros elementos de almacenamiento transparentes.
 - La ruta de datos debe incluir tantos tipos de **elementos operativos** como tipos de operaciones de cálculo se indiquen en el repertorio de instrucciones
 - El significado de cada instrucción vendrá dado por un conjunto de transferencias entre registros. La ruta de datos debe ser capaz de soportar dichas transferencias.
- **Paso 2:** Establecer la metodología de temporización
 - **Monociclo (CPI = 1):** todas las transferencias entre registros implicadas en una instrucción se realizan en un único ciclo de reloj.
 - **Multiciclo (CPI > 1):** las transferencias entre registros implicadas en una instrucción se reparten entre varios ciclos de reloj.
 - **Segmentada. ?**
- **Paso 3:** Seleccionar el conjunto de módulos (de almacenamiento, operativos e interconexión) que forman la ruta de datos.
- **Paso 4:** Ensamblar la ruta de datos de modo que se cumplan los requisitos impuestos por el repertorio, localizando los puntos de control.
- **Paso 5:** Determinar los valores de los puntos de control analizando las transferencias entre registros incluidas en cada instrucción.
- **Paso 6:** Diseñar la lógica de control.



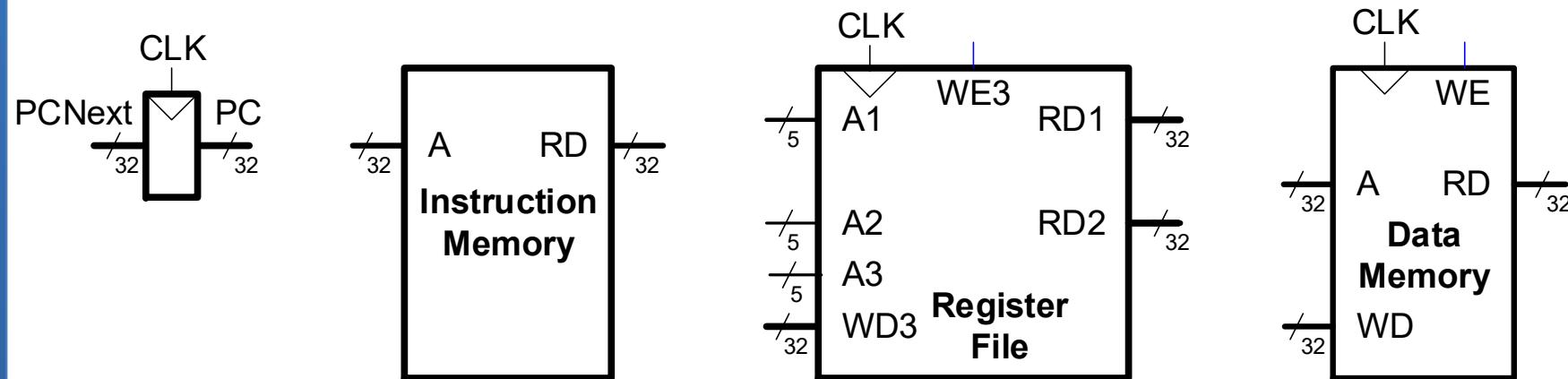
Diseño de la ruta de datos

- Vamos a implementar un subconjunto de todo el repertorio:
 - Instrucciones aritmético lógicas tipo-R
 - **add, sub, and, or, slt**
 - Instrucciones de memoria:
 - **lw, sw**
 - Instrucciones de salto:
 - **beq**



Diseño de la ruta de datos

- Elementos de almacenamiento:
 - Registro Contador de Programa PC
 - BR: 32 registros
 - Memorias: Separadas la de instrucciones y la de datos





Diseño de la ruta de datos

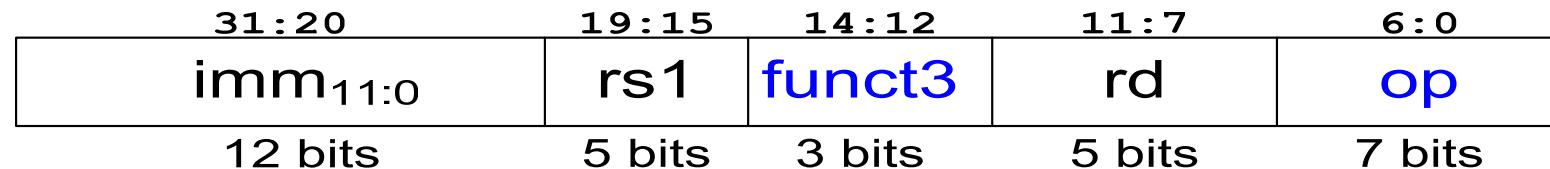
- Características de los elementos de almacenamiento:
 - La memoria de instrucciones sólo lee instrucciones, no nos preocupamos de cómo se escribe.
 - La memoria de datos lee en todos los ciclos y escribe cuando WE (write enable) esté a 1.
 - El Banco de registros permite leer dos registros y escribir un registro en el mismo ciclo:
 - Se leen los registros cuyas direcciones aparecen en A1 y A2 sobre los puertos RD1 y RD2.
 - Se escribe en el registro con dirección A3 lo que viene por el puerto WD3 cuando WE3 (write enable) esté a 1.
 - Todas las lecturas son combinacionales, en el sentido de que si se cambia la dirección aparece el nuevo dato a la salida con un retardo.



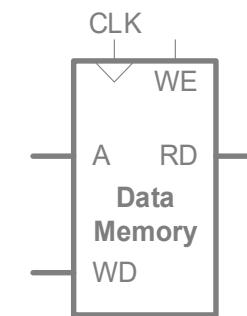
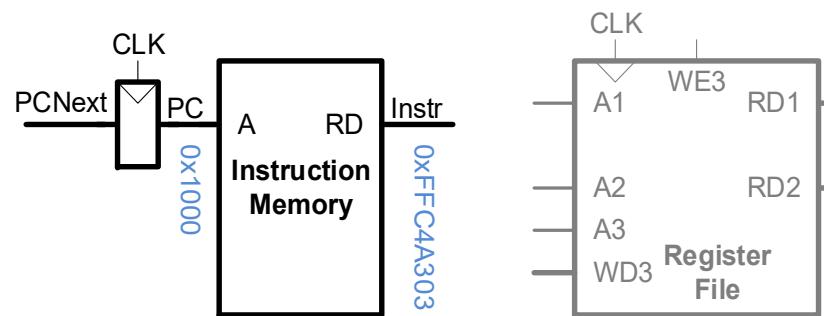
Ruta de datos monociclo: lw

- lw x6, -4(x9)

I-Type



Fase 1: Búsqueda de la instrucción

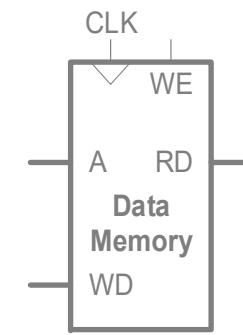
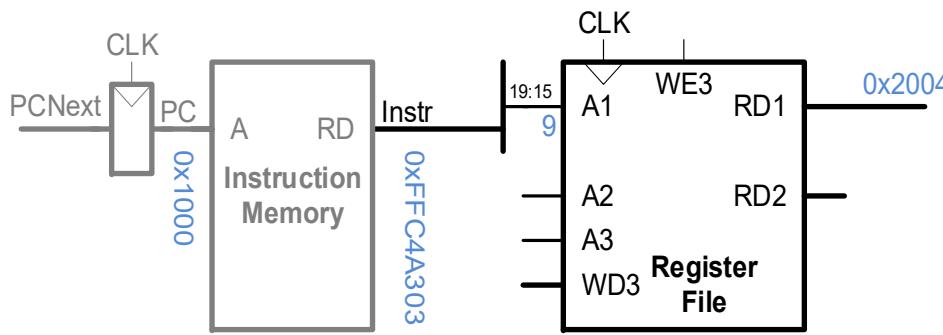


Address	Instruction	Type	Fields	Machine Language
0x1000	l7: lw x6, -4(x9)	I	imm _{11:0} rs1 f3 rd op	FFC4A303



Ruta de datos monociclo: lw

- `lw x6, -4(x9)`
- **Fase 2:** Leer operando fuente del Banco de registros

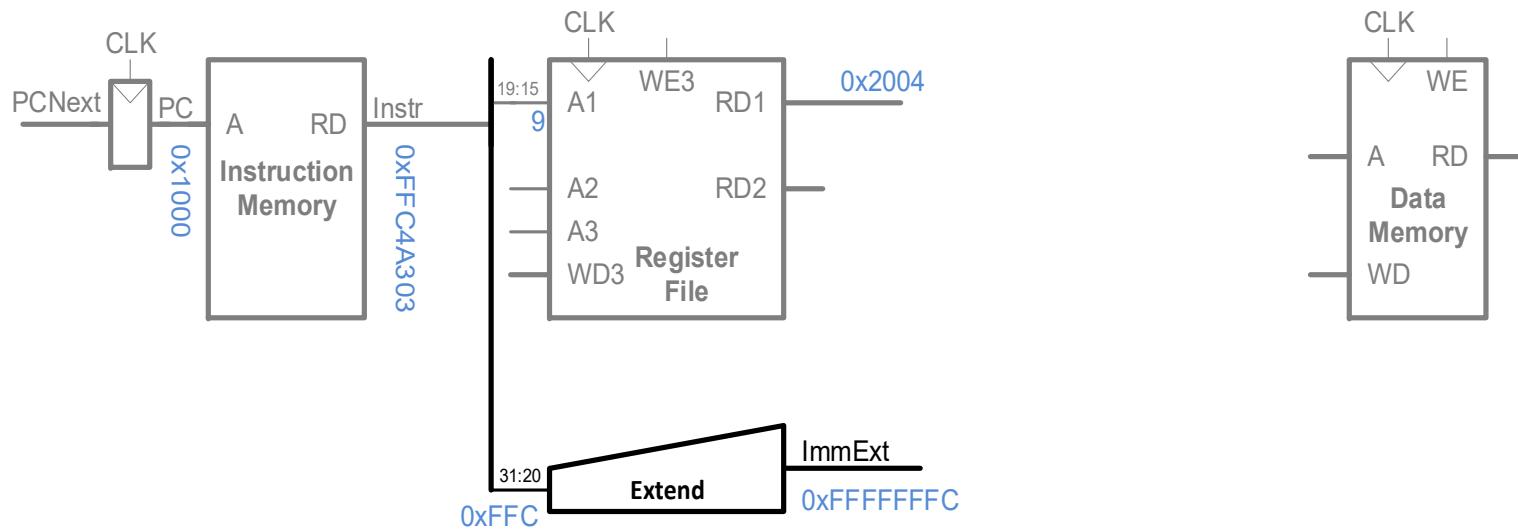


Address	Instruction	Type	Fields	Machine Language
0x1000	<code>lw x6, -4(x9)</code>	I	$\text{imm}_{11:0}$ rs1 f_3 rd	$op_{0000011}$ FFC4A303



Ruta de datos monociclo: lw

- lw x6, -4(x9)
- Fase 3:** Extender el campo inmediato



Address	Instruction	Type	Fields	Machine Language
<code>0x1000</code>	<code>L7: lw x6, -4(x9)</code>	I	<code>imm_{11:0}</code> <code>rs1</code> <code>f3</code> <code>rd</code>	<code>op</code> <code>0000011 FFC4A303</code>



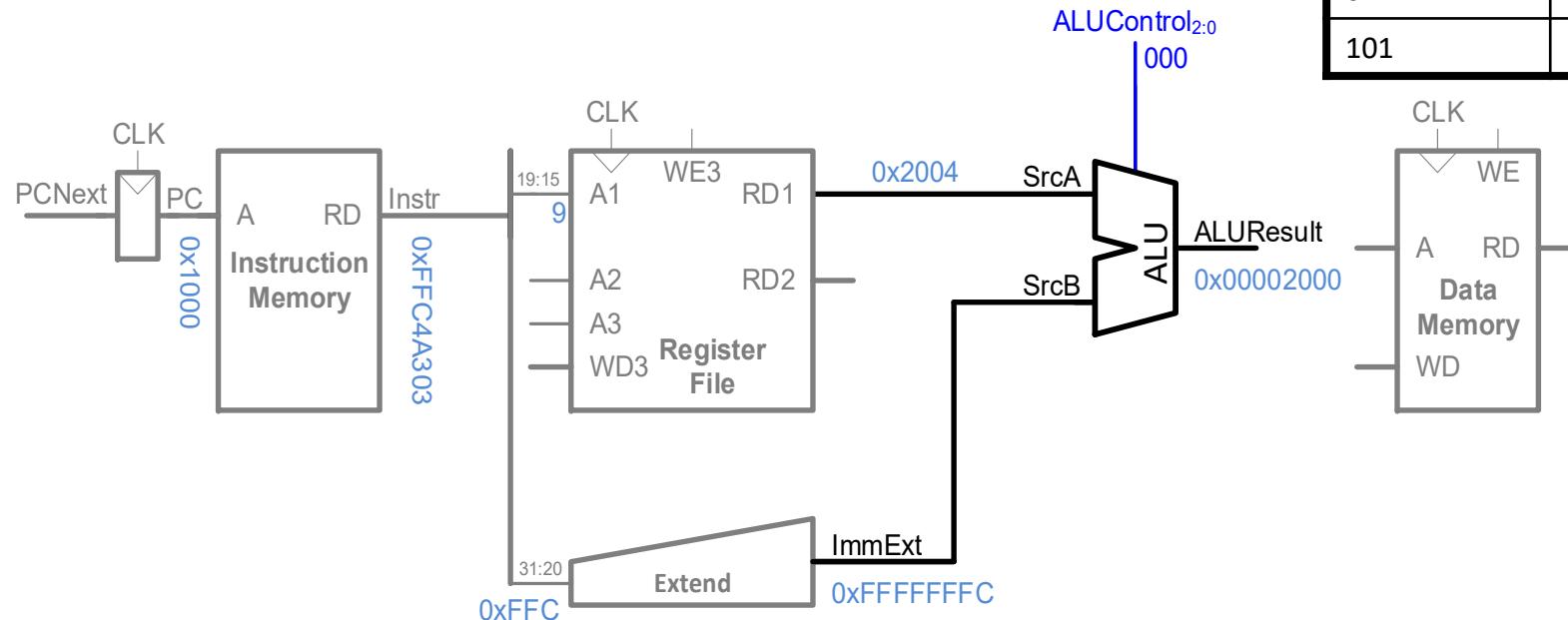
Ruta de datos monociclo: lw

versión 2021

tema 6:
Diseño del procesador

- lw x6, -4(x9)
- Fase 4:** Calcula la dirección de memoria

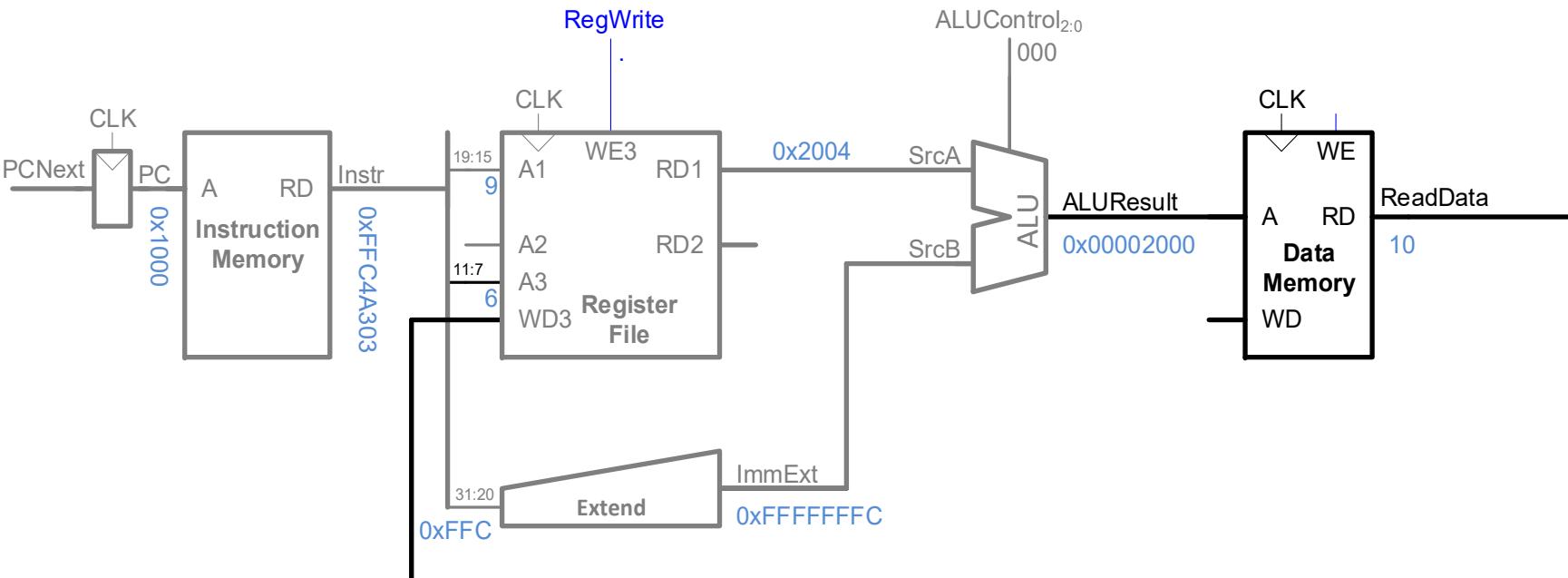
$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



Address	Instruction	Type	Fields	Machine Language
FC 0x1000	L7: lw x6, -4(x9)	I	<code>imm_{11:0}</code> <code>rs1</code> <code>f3</code> <code>rd</code> <code>op</code>	<code>FFC4A303</code>



Ruta de datos monociclo: lw



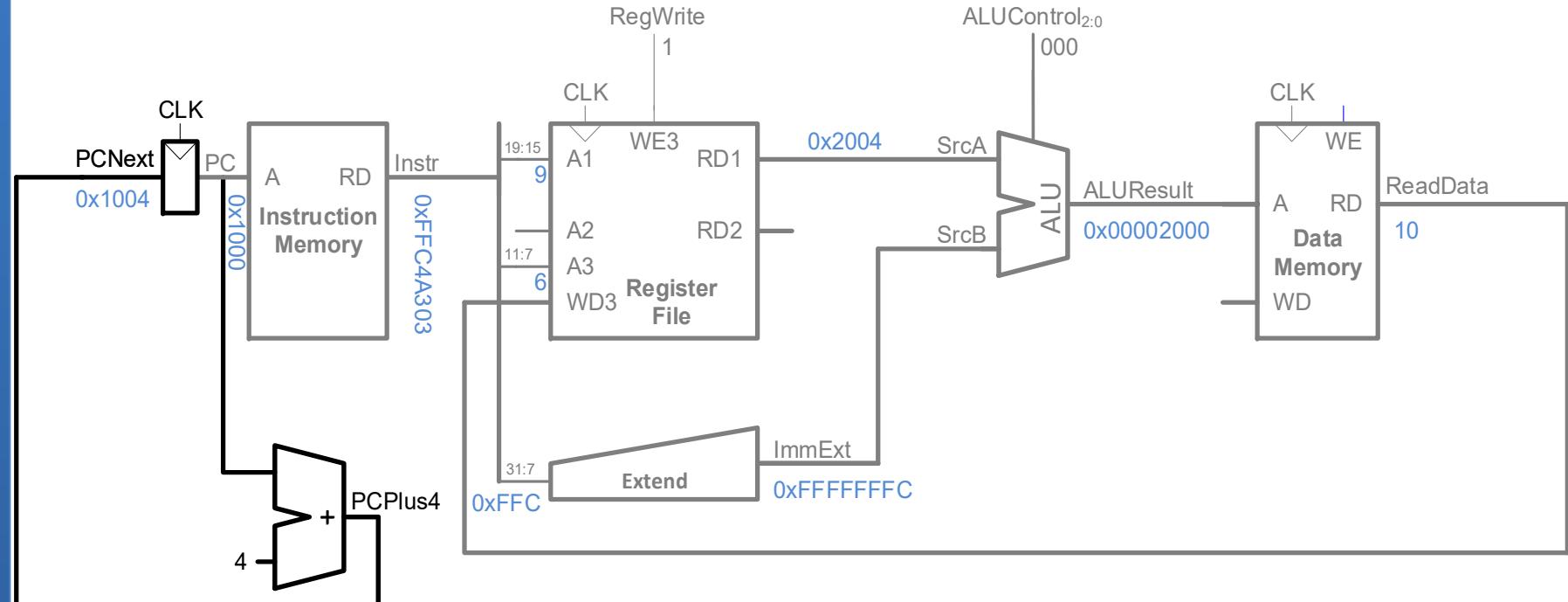
Address	Instruction	Type	Fields	Machine Language
0x1000	I7: lw x6, -4(x9)	I	imm _{11:0} rs1 f3 rd op	1111111111100 01001 010 00110 0000011 FFC4A303



Ruta de datos monociclo: lw

versión 2021

- `lw x6, -4(x9)`
- **Fase 6:** Calcula la dirección de la siguiente instrucción



tema 6:
Diseño del procesador

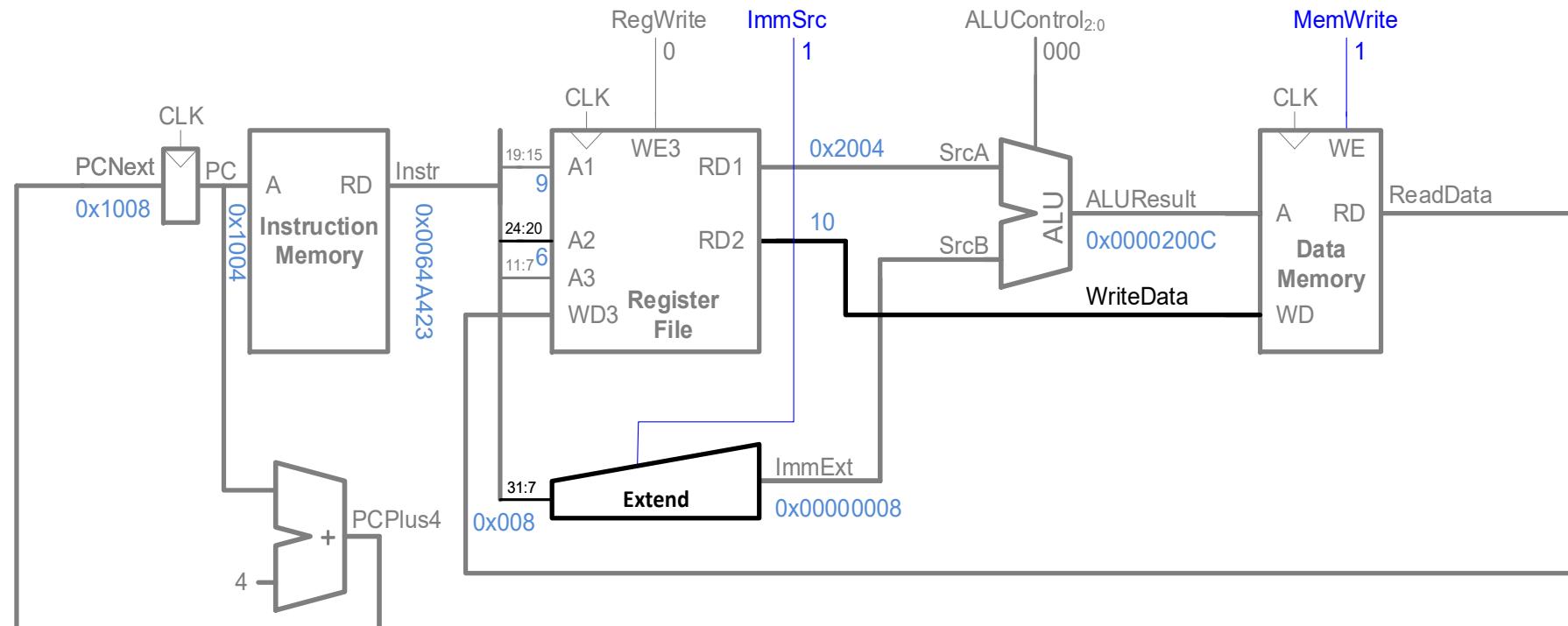
FC

Address	Instruction	Type	Fields	Machine Language
0x1000	L7: <code>lw x6, -4(x9)</code>	I	$\text{imm}_{11:0}$ <code>rs1</code> <code>f3</code> <code>rd</code> <code>op</code>	<code>FFC4A303</code>



Ruta de datos monociclo: sw

- **Inmediato:** Ahora en instr[31-25] y instr[11:7]
- Se añaden las señales de control ImmSrc, MemWrite

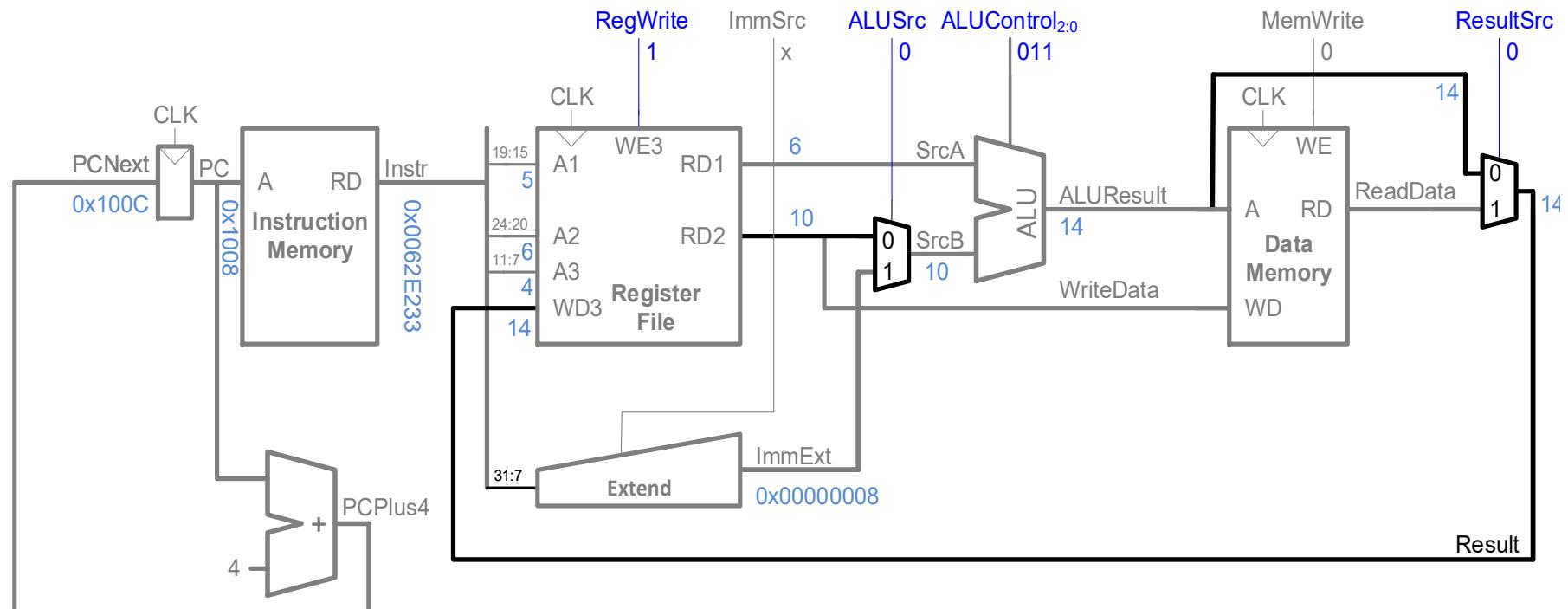


Address	Instruction	Type	Fields						Machine Language	
<code>0x1004</code>	<code>sw x6, 8(x9)</code>	S	<code>imm_{11:5}</code> <code>0000000</code>	<code>rs2</code> <code>00110</code>	<code>rs1</code> <code>01001</code>	<code>f3</code> <code>010</code>	<code>imm_{4:0}</code> <code>01000</code>	<code>op</code> <code>0100011</code>	<code>0064A423</code>	



Ruta de datos monociclo: Tipo-R

- Lee de **rs1** y **rs2** en lugar del inmmedio
- Escribe **ALUResult** a **rd**



Tema 6:
Diseño del procesador

FC

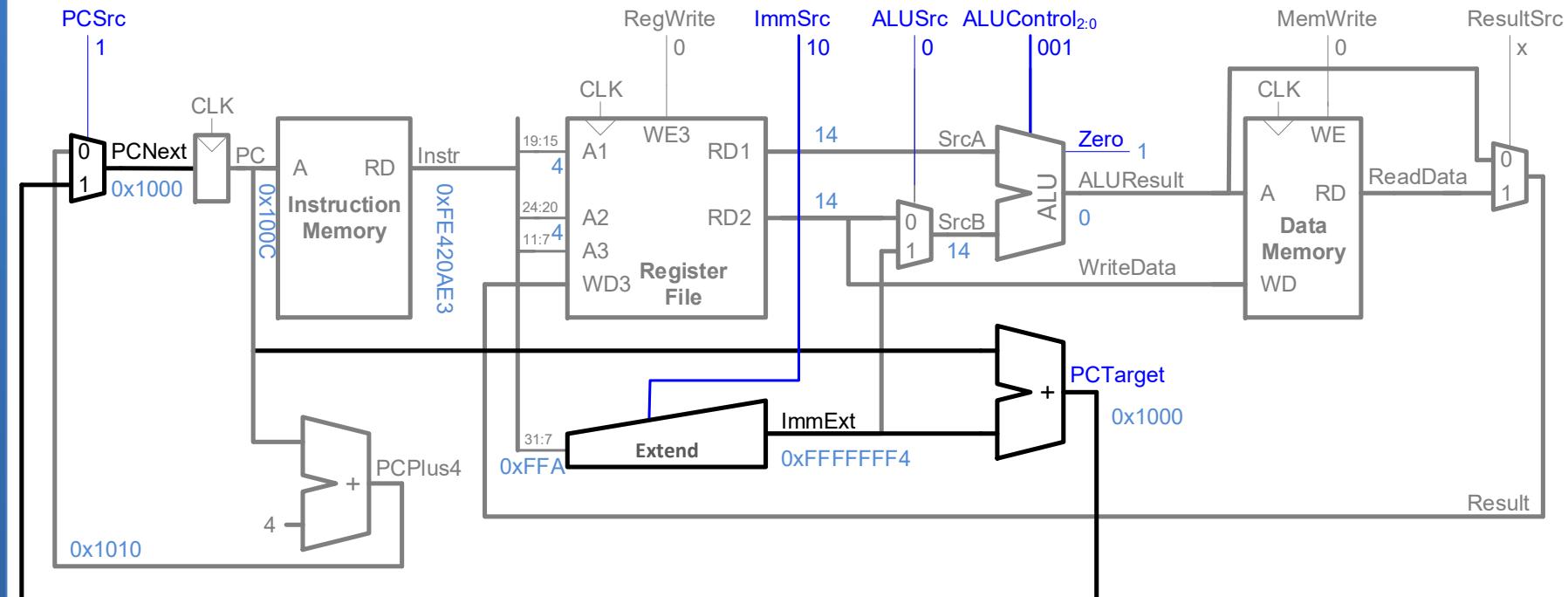
31

Address	Instruction	Type	Fields							Machine Language		
0x1008	or x4, x5, x6	R	funct7	0000000	rs2	00110	rs1	00101	f3	110	rd	op 0110011 0062E233



Ruta de datos monociclo: BEQ

- Calcula dirección de salto: $PC = PC + imm$





Ruta de datos monociclo: ImmExt

versión 2021

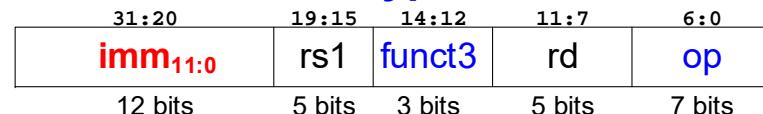
tema 6:
Diseño del procesador

FC

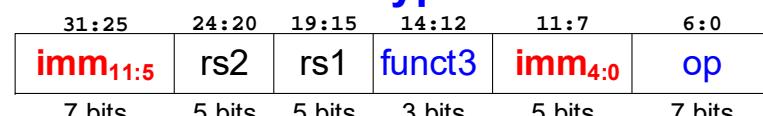
33

ImmSrc _{1:0}	ImmExt	Tipo instrucción
00	$\{{20\{instr[31]\}}, \text{instr}[31:20]\}$	I-Type
01	$\{{20\{instr[31]\}}, \text{instr}[31:25], \text{instr}[11:7]\}$	S-Type
10	$\{{19\{instr[31]\}}, \text{instr}[31], \text{instr}[7], \text{instr}[30:25], \text{instr}[11:8], 1'b0\}$	B-Type

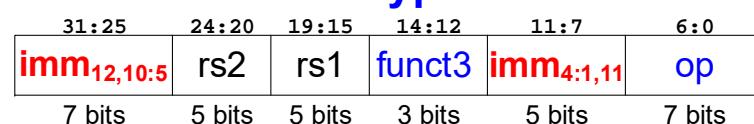
I-Type



S-Type



B-Type





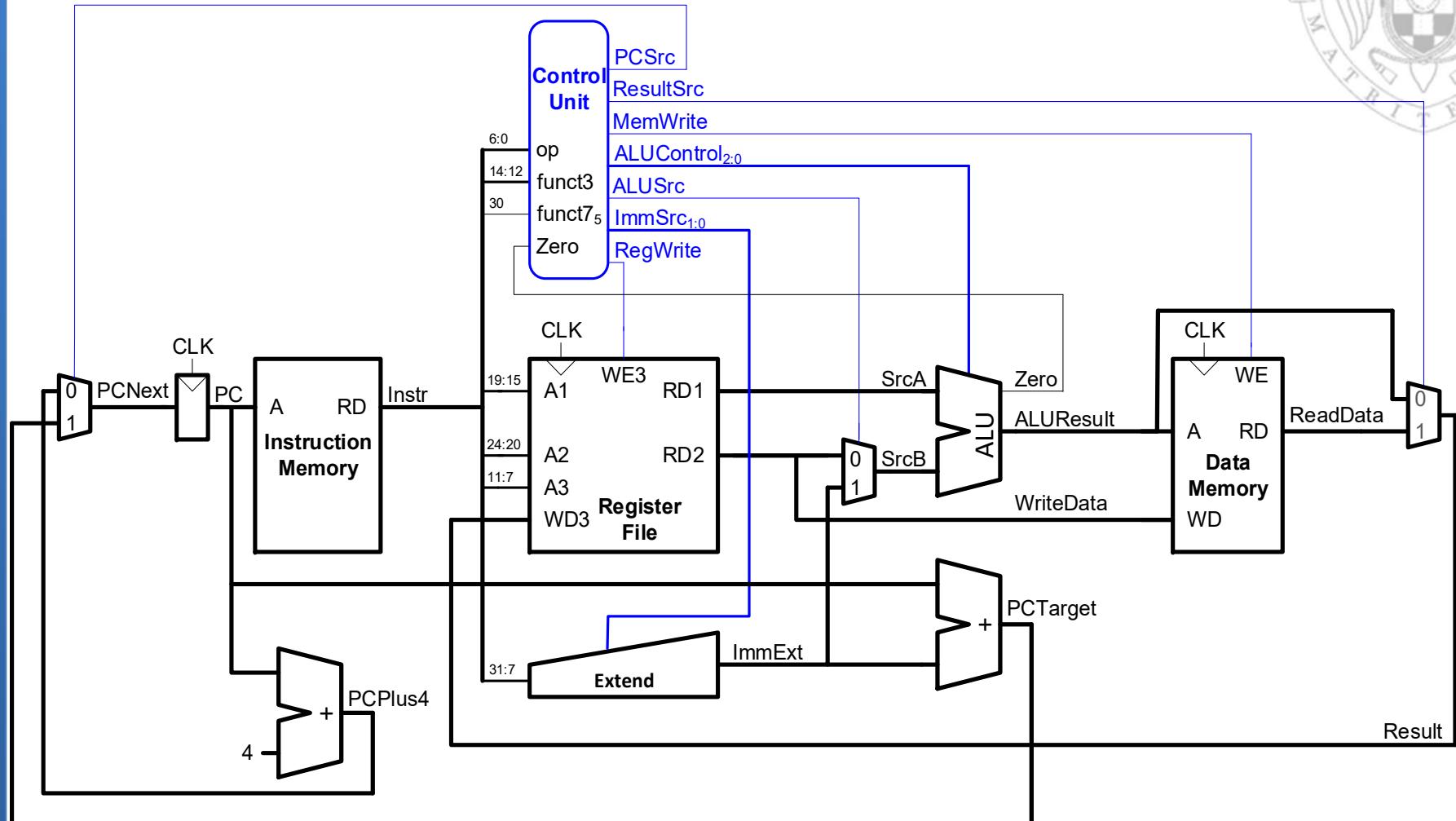
Ruta de datos monociclo RISC-V

versión 2021

tema 6:
Diseño del procesador

FC

34





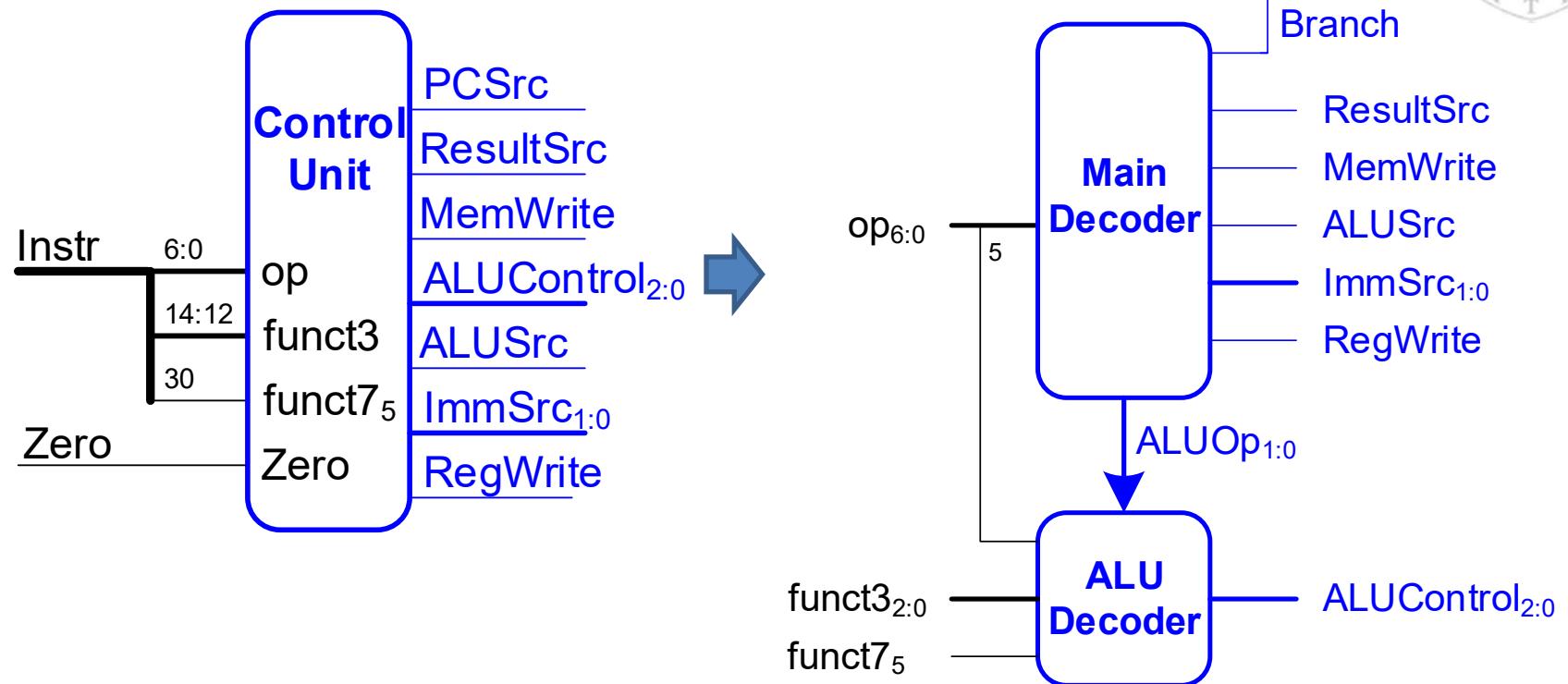
Unidad de control monociclo

versión 2021

tema 6:
Diseño del procesador

FC

35





Unidad de control monociclo

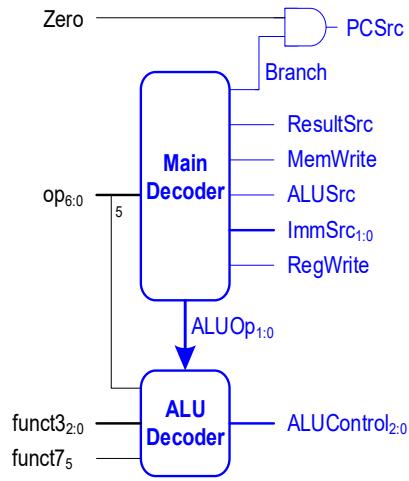
versión 2021

op	Instr.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	lw	1	00	1	0	1	0	00
35	sw	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	beq	0	10	0	0	X	1	01

tema 6:
Diseño del procesador

FC

36





Unidad de control monociclo

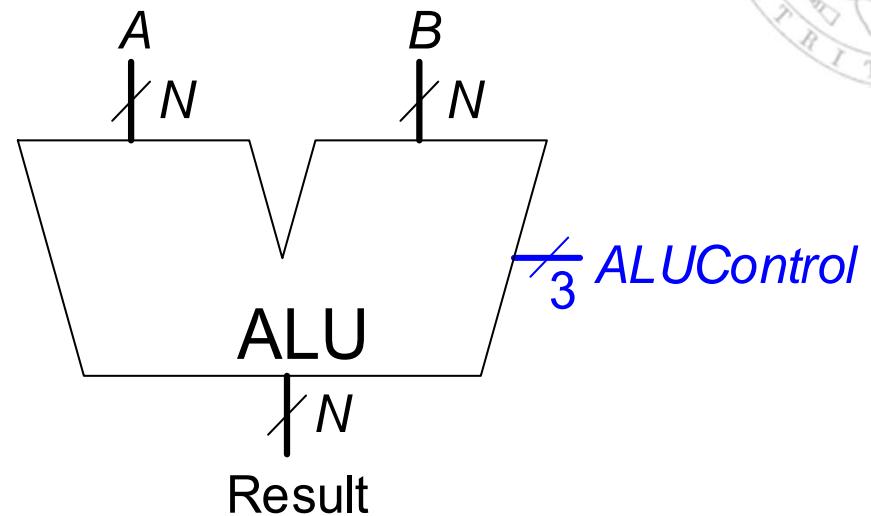
versión 2021

tema 6:
Diseño del procesador

FC

37

ALUControl _{2:0}	Función
000	add
001	subtract
010	and
011	or
101	SLT





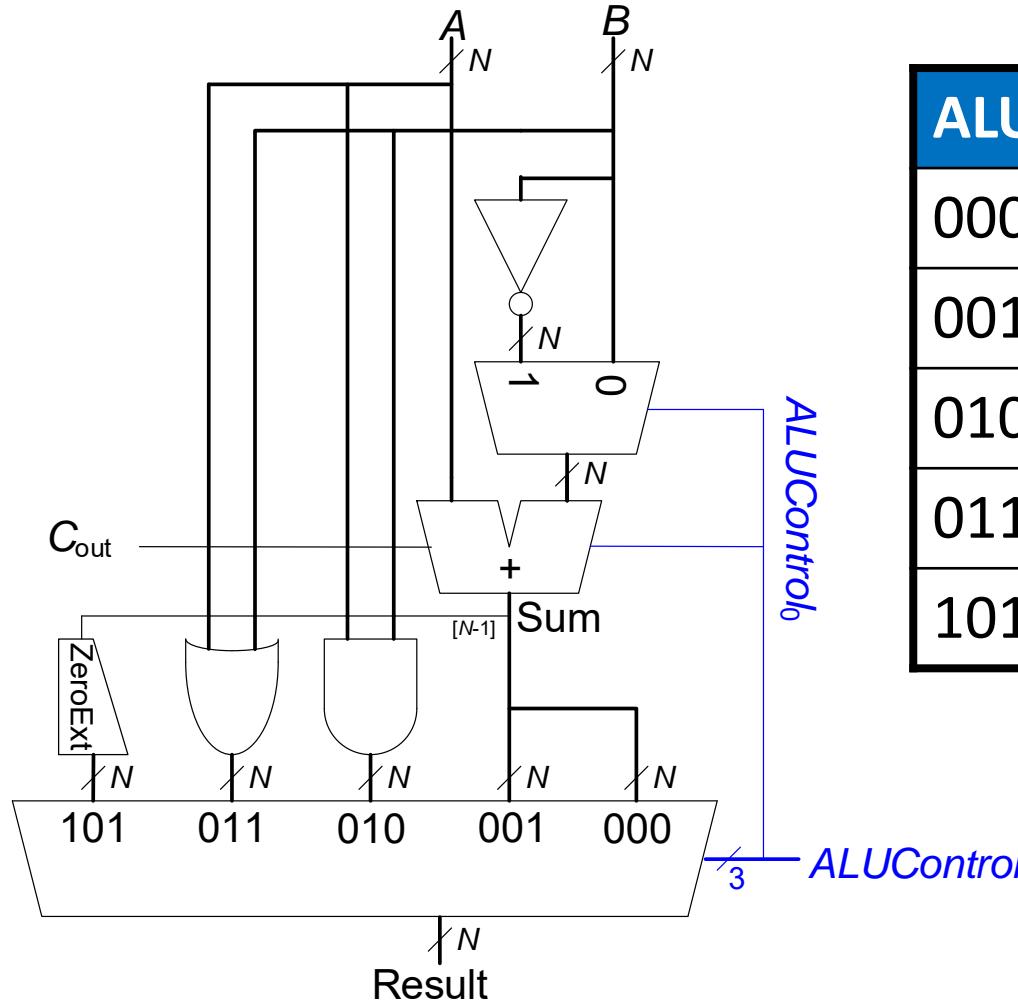
Unidad de control monociclo

versión 2021

tema 6:
Diseño del procesador

FC

38



$ALUControl_{2:0}$	Function
000	add
001	subtract
010	and
011	or
101	SLT



Unidad de control monociclo

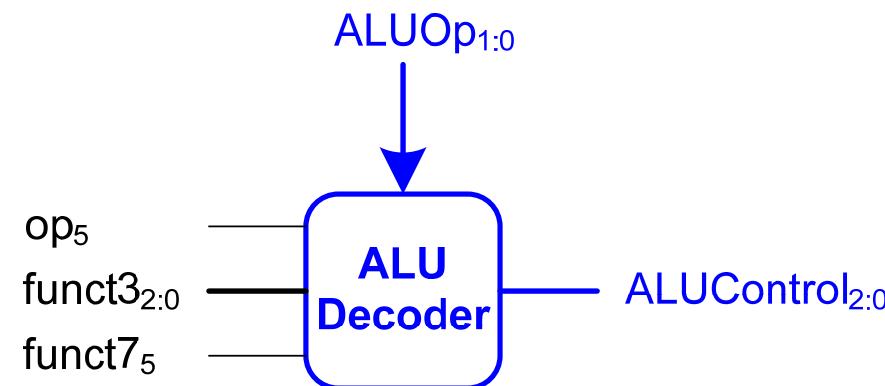
versión 2021

tema 6:
Diseño del procesador

FC

39

$ALUOp$	$funct3$	$op_5, funct7_5$	Instrucción	$ALUControl_{2:0}$
00	x	x	lw, sw	000 (add)
01	x	x	beq	001 (subtract)
10	000	00, 01, 10	add	000 (add)
	000	11	sub	001 (subtract)
	010	x	slt	101 (set less than)
	110	x	or	011 (or)
	111	x	and	010 (and)





Unidad de control monociclo

- Ejemplo: and

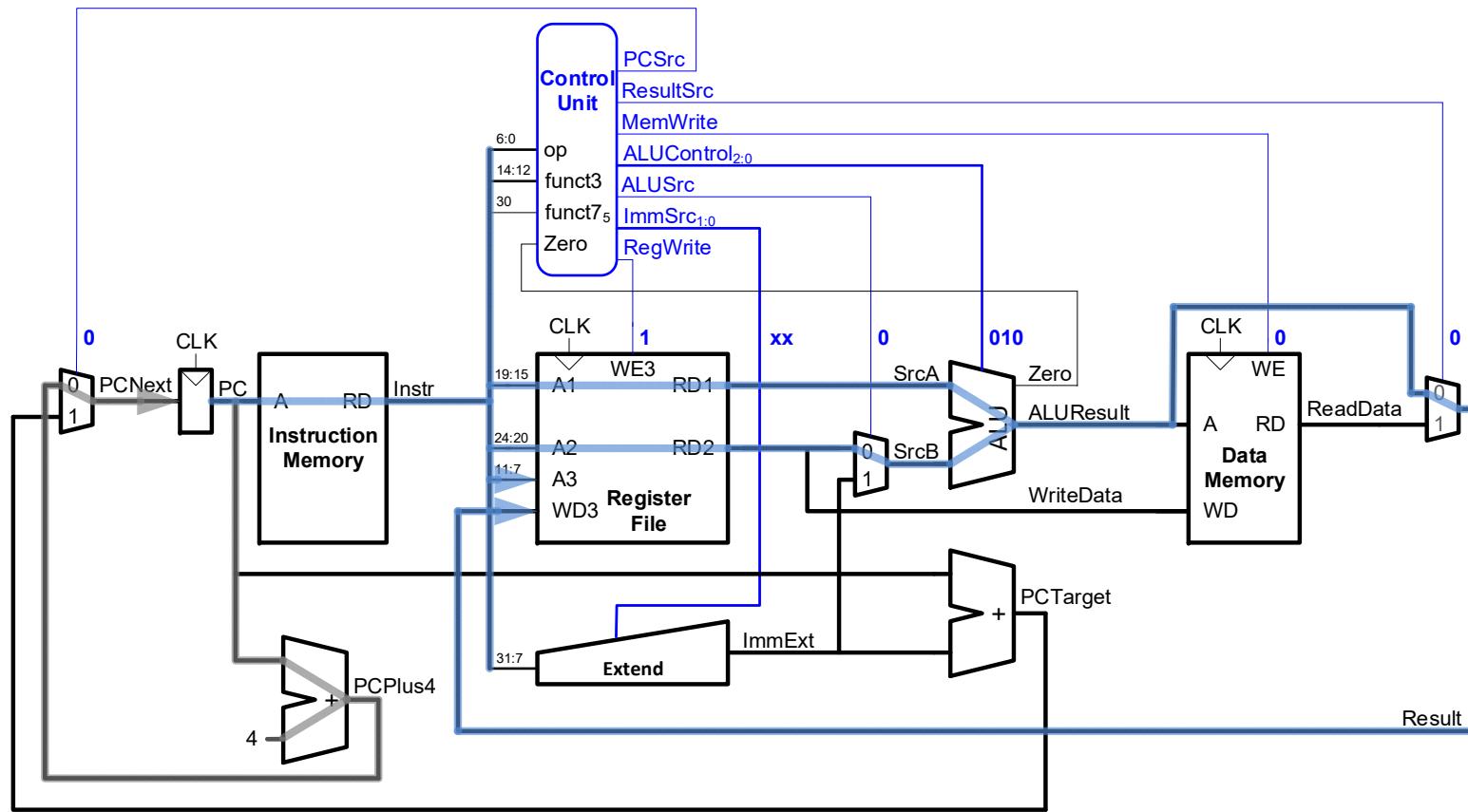
op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
51	R-type	1	XX	0	0	0	0	10

versión 2021

tema 6:
Diseño del procesador

FC

40





Unidad de control monociclo: Ampliar el repertorio

- 1.- Añadir instrucciones aritmético-lógicas con segundo operando inmediato: addi, andi, ori y slti
 - Similar a instrucciones R-type
 - Cambiar **ALUSrc** para seleccionar el valor inmediato
 - Cambiar **ImmSrc** para elegir el inmediato adecuado



Unidad de control monociclo: Ampliar el repertorio

versión 2021

tema 6:
Diseño del procesador

FC

42

op	Instruc	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	lw	1	00	1	0	1	0	00
35	sw	0	01	1	1	X	0	00
51	R-type	1	XX	0	0	0	0	10
99	beq	0	10	0	0	X	1	01
19	I-type	1	00	1	0	0	0	10

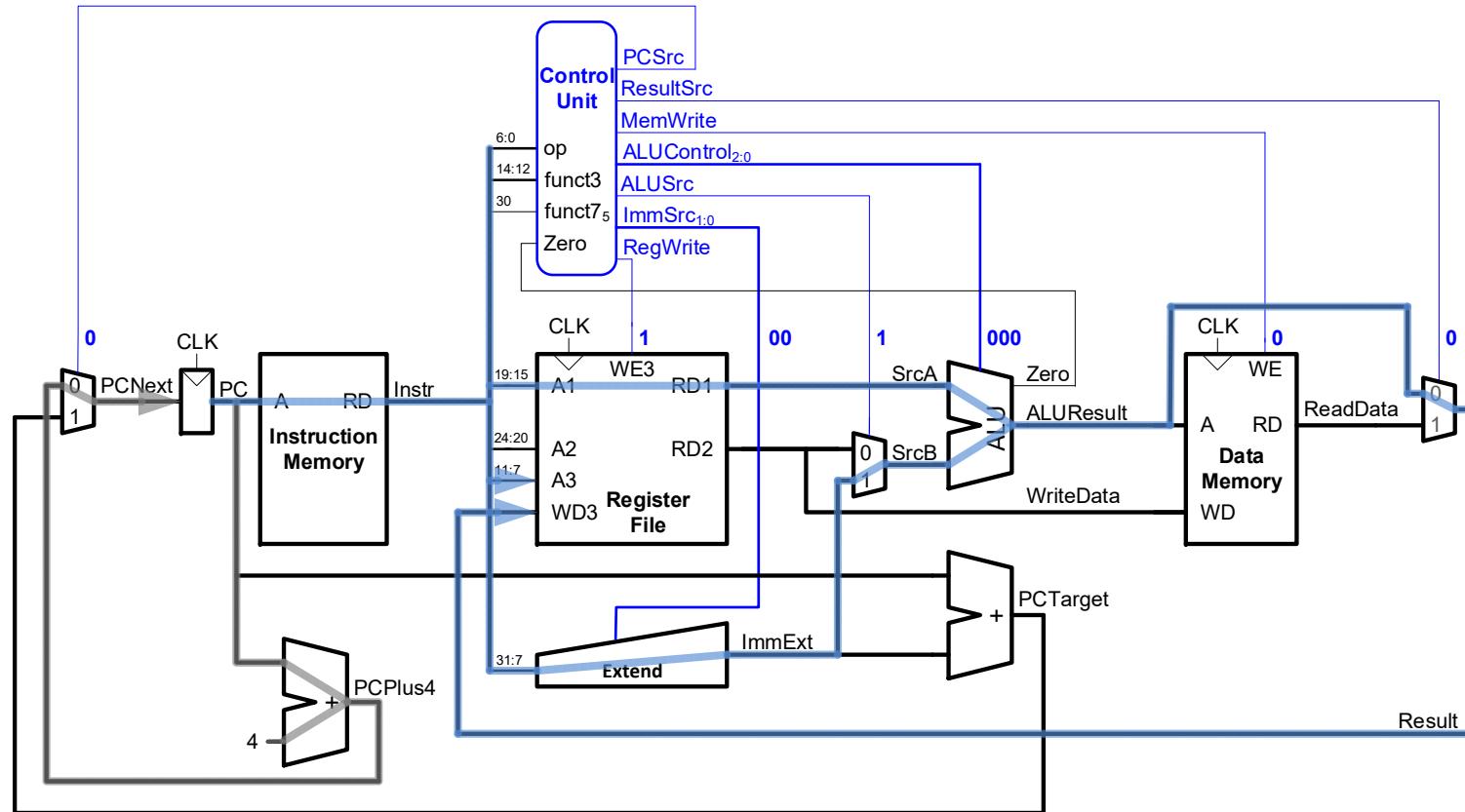


Unidad de control monociclo

- Ejemplo: addi

versión 2021

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
19	I-type	1	00	1	0	0	0	10



tema 6:
Diseño del procesador

FC

43



Unidad de control monociclo: Ampliar el repertorio

- 2.- Añadir instrucciones de jump and link: **jal**
 - Similar a **beq** pero el salto siempre se realiza:
 - $PCSrc$ deberá valer 1
 - El formato Immediato es diferente
 - Se necesita un nuevo $ImmSrc$ con valor 11
 - **jal** debe calcular $PC+4$ y almacenarlo en rd
 - $PC+4$ se obtiene del sumador a través de $ResultSrc$, al que se añade una entrada más.



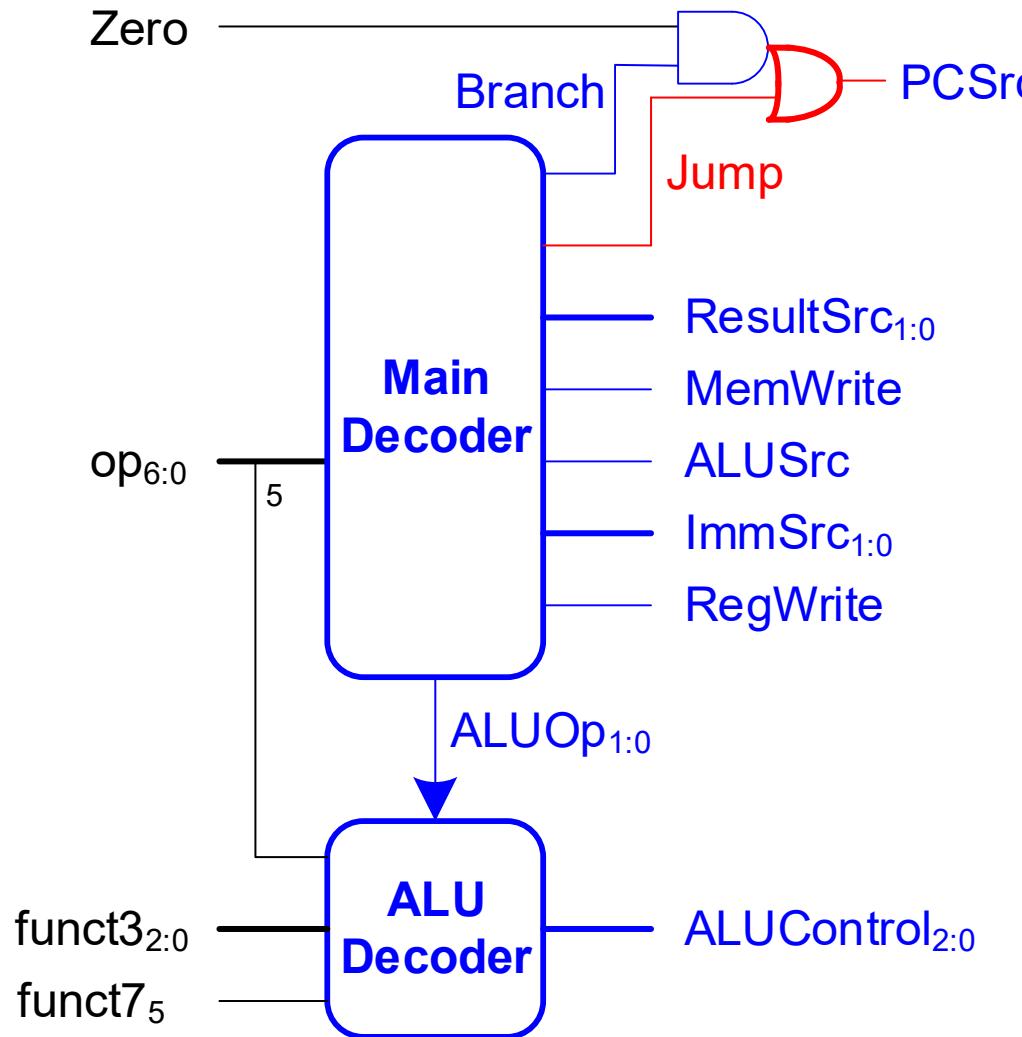
Unidad de control monociclo: Ampliar el repertorio

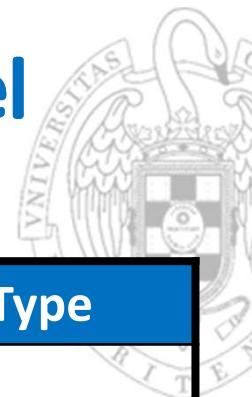
versión 2021

tema 6:
Diseño del procesador

FC

45



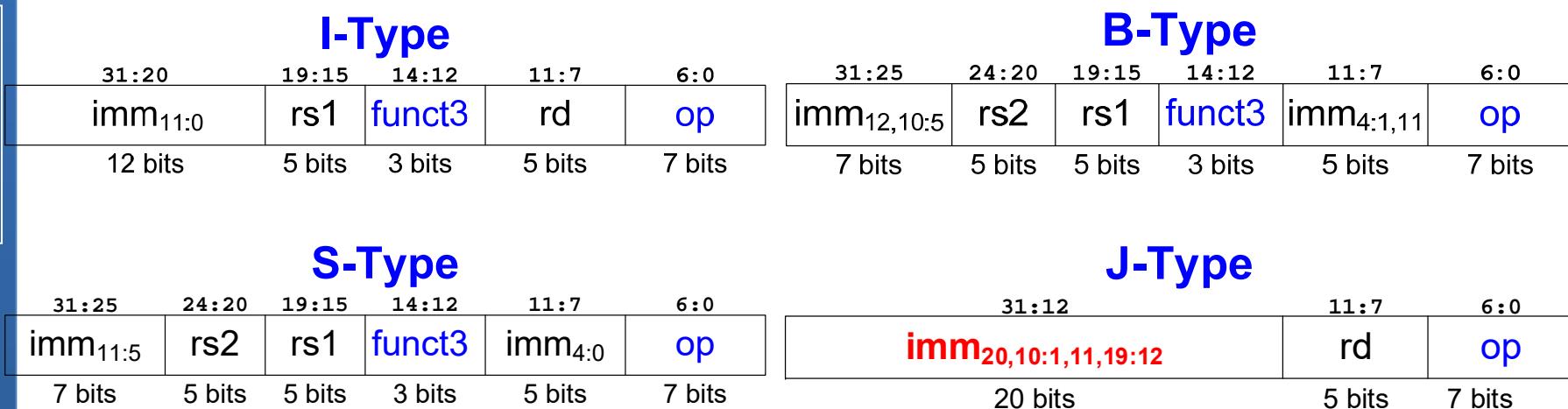


Unidad de control monociclo: Ampliar el repertorio

versión 2021

ImmSrc _{1:0}	ImmExt	Instruction Type
00	$\{\{20\{\text{instr}[31]\}\}, \text{instr}[31:20]\}$	I-Type
01	$\{\{20\{\text{instr}[31]\}\}, \text{instr}[31:25], \text{instr}[11:7]\}$	S-Type
10	$\{\{19\{\text{instr}[31]\}\}, \text{instr}[31], \text{instr}[7], \text{instr}[30:25], \text{instr}[11:8], 1'b0\}$	B-Type
11	$\{\{12\{\text{instr}[31]\}\}, \text{instr}[19:12], \text{instr}[20], \text{instr}[30:21], 1'b0\}$	J-Type

tema 6:
Diseño del procesador



FC

46



Unidad de control monociclo: Ampliar el repertorio

versión 2021

op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
3	<code>lw</code>	1	00	1	0	01	0	00	0
35	<code>sw</code>	0	01	1	1	XX	0	00	0
51	R-type	1	XX	0	0	00	0	10	0
99	<code>beq</code>	0	10	0	0	XX	1	01	0
19	I-type	1	00	1	0	00	0	10	0
111	<code>jal</code>	1	11	X	0	10	0	XX	1

tema 6:
Diseño del procesador

FC

47



Unidad de control monociclo: Ampliar el repertorio

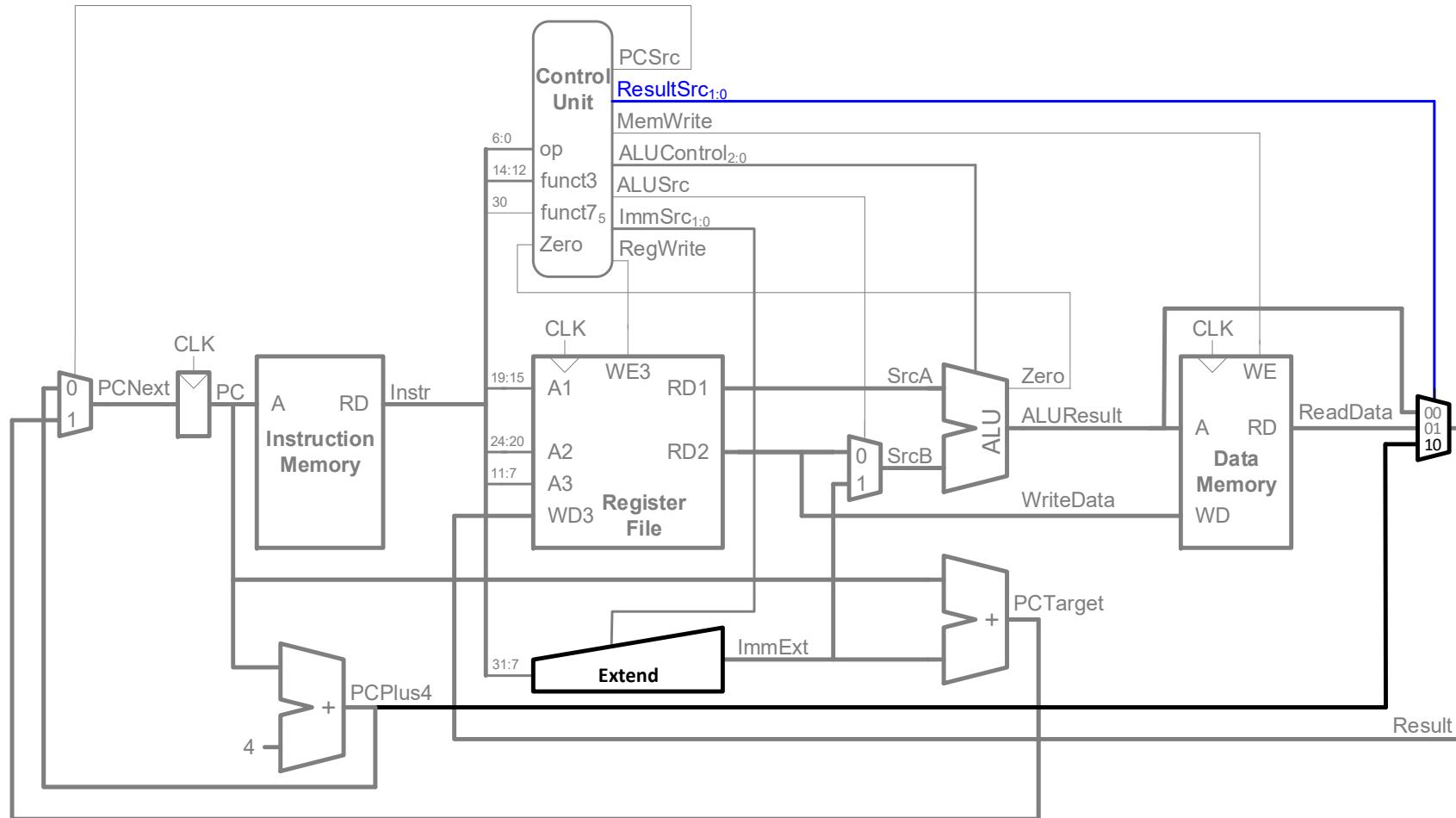
versión 2021

tema 6:
Diseño del procesador

FC

48

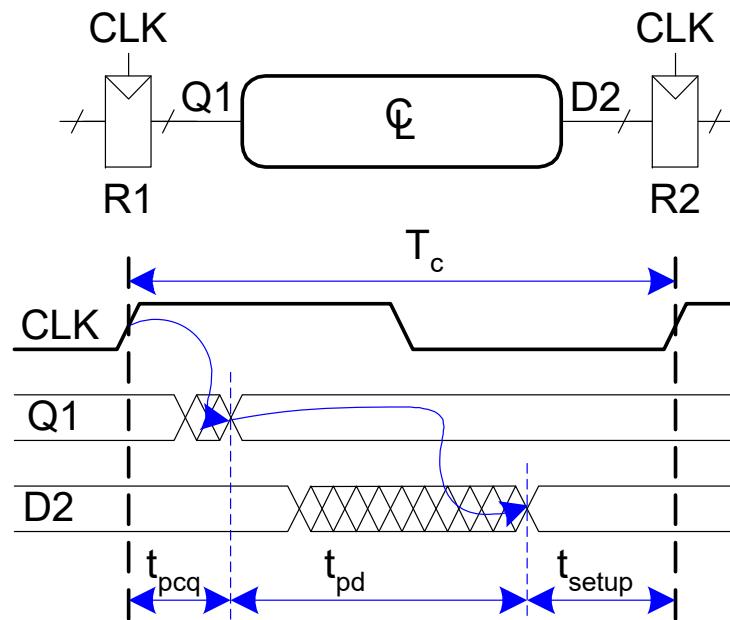
op	Instruct.	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
111	jal	1	11	X	0	10	0	XX	1



Timing

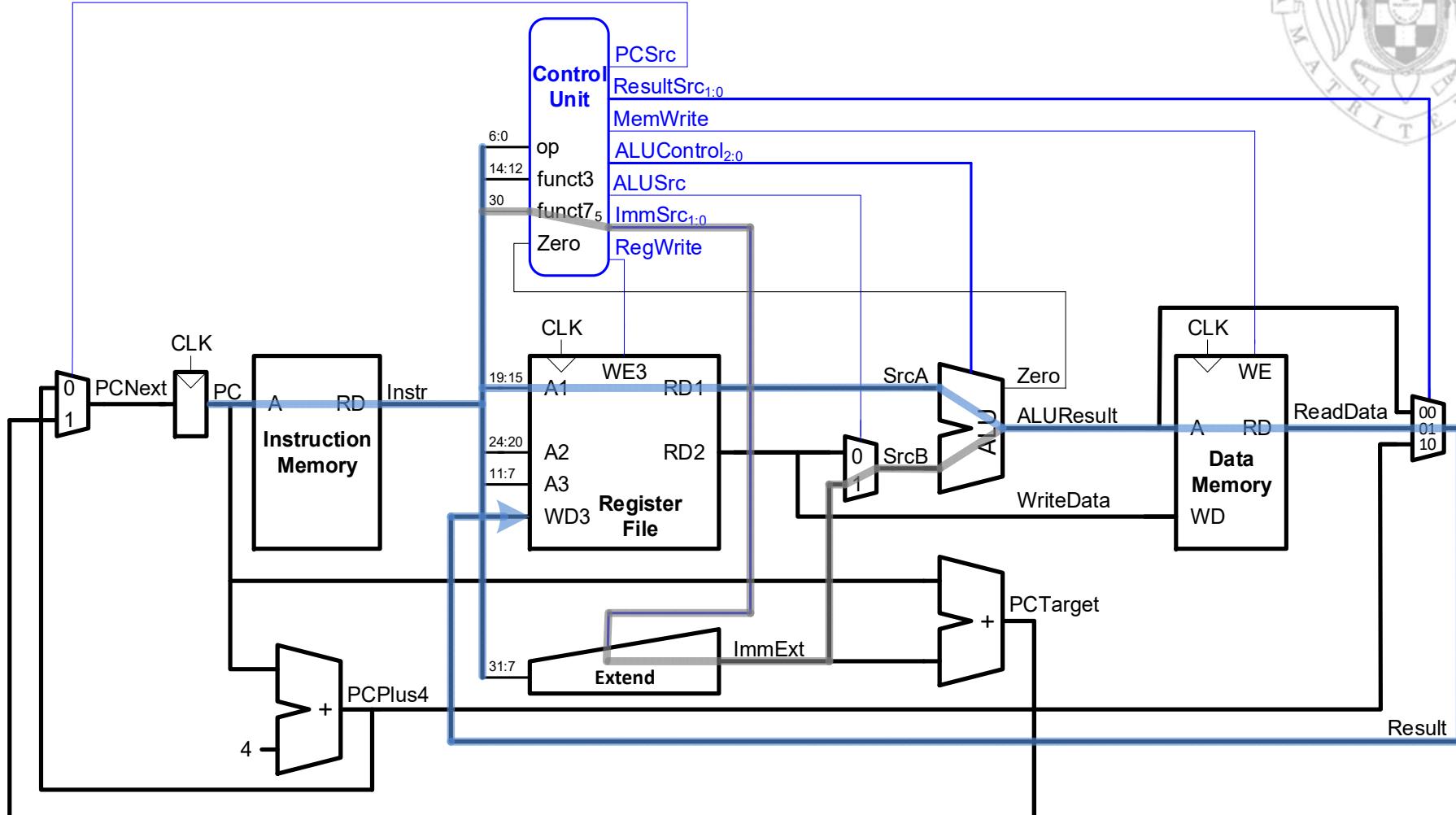


- **Tiempo de setup:** t_{setup} = tiempo que los datos deben estar estables a la entrada de un registro antes de que llegue el flanco de subida del reloj para garantizar el almacenamiento correcto.
- **Retardo de propagación del registro:** t_{pcq} = tiempo que debe transcurrir después del flanco de reloj para que la salida Q esté disponible.
- t_{pd} es el tiempo de propagación a través del elemento combinacional
- **Tiempo de ciclo:** T_c , es la suma de los tres tiempos anteriores





Rendimiento monociclo



T_c limitado por el camino critico (**lw**)



Rendimiento: Monociclo

- Camino crítico monociclo :

$$T_c = t_{pcq_PC} + t_{\text{mem}} + \max[t_{BR\text{read}}, t_{\text{dec}} + t_{ext} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{BR\text{setup}}$$

- Normalmente los caminos más lentos pasan por la memoria, la ALU y el banco de registros

$$T_c = t_{pcq_PC} + 2t_{\text{mem}} + t_{dec} + t_{BR\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{BR\text{setup}}$$



Rendimiento: Monociclo

versión 2021

tema 6:
Diseño del procesador

FC

52

Elemento	Parámetro	Retardo (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexor	t_{mux}	30
Puertas AND-OR	$t_{\text{AND-OR}}$	20
ALU	t_{ALU}	120
Decodificador	t_{dec}	25
Unidad de extensión	t_{ext}	35
Read memoria	t_{mem}	200
Read Banco de registros	$t_{BR\text{read}}$	100
Setup Banco de registros	$t_{BR\text{setup}}$	60

$$\begin{aligned}T_{cl} &= t_{pcq_PC} + 2t_{\text{mem}} + t_{BR\text{read}} + t_{\text{ALU}} + t_{\text{mux}} + t_{BR\text{setup}} \\&= [40 + 2*(200) + 100 + 120 + 30 + 60] \text{ ps} = \mathbf{750 \text{ ps}}\end{aligned}$$



Rendimiento: Monociclo

Programa con 100 mil millones de instrucciones:

$$\begin{aligned}\text{Tiempo de Ejecución} &= \text{N_instrucciones} * \text{CPI} * T_C \\ &= (100 * 10^9) * (1) * (750 * 10^{-12} \text{ s}) \\ &= \mathbf{75 \text{ segundos}}\end{aligned}$$



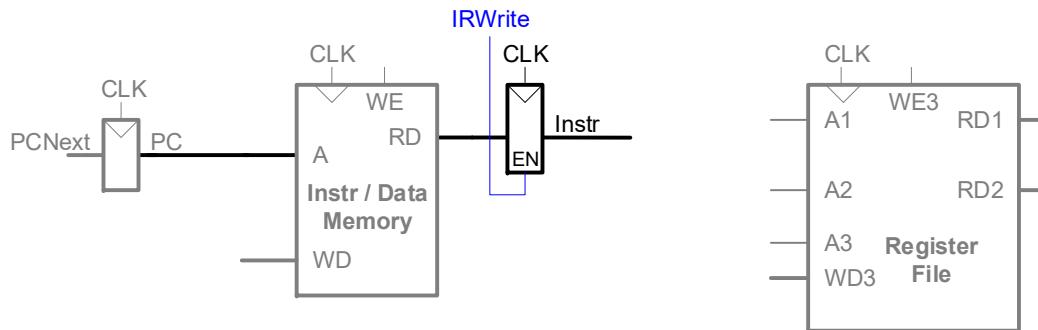
Ruta de datos multiciclo

- Problemas de la implementación monociclo:
 - El ciclo de reloj viene dado por la instrucción más lenta.
 - Memoria de instrucciones y datos separada.
 - No hay reuso de hardware: por ejemplo se requieren tres sumadores
- Implementación multiciclo:
 - Se divide la ejecución de una instrucción en varias etapas. ($CPI > 1$)
 - Cada instrucción requerirá un número diferente de etapas.
 - El tiempo de ciclo será muy inferior.
 - Se puede reusar hardware entre diferentes etapas.
 - Se puede tener una sola memoria para instrucciones y datos

Ruta de datos multiciclo



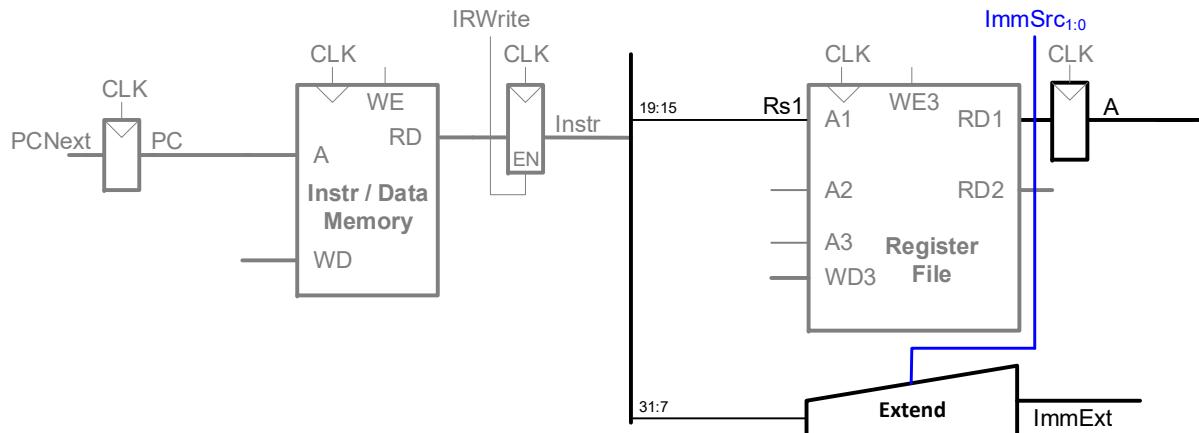
- Aparecen elementos de almacenamiento que no pertenecen a la arquitectura. Transparentes al usuario (no pueden accederse desde el ensamblador).
 - Registros donde almacenar los resultados intermedios
- **Fase 1:** Búsqueda de la instrucción
 - Nuevo Registro *Instruction Register* para guardar el código de la instrucción





Ruta de datos multiciclo: lw

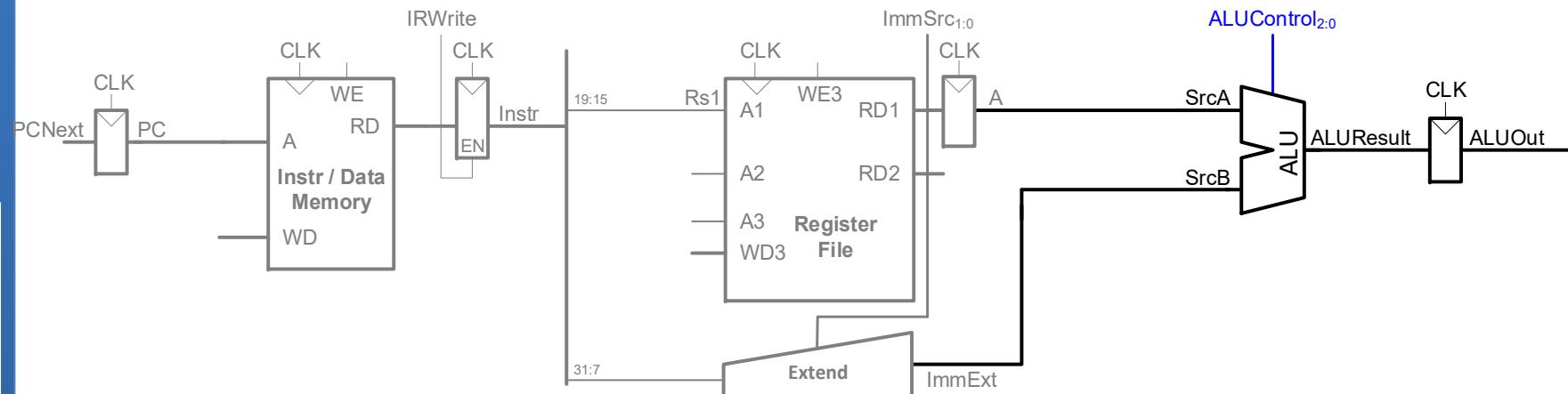
- **Fase 2:** Búsqueda de los operandos en el banco de registro y extensión de signo
 - Nuevo Registro *A* para guardar el contenido del primer operando





Ruta de datos multiciclo: lw

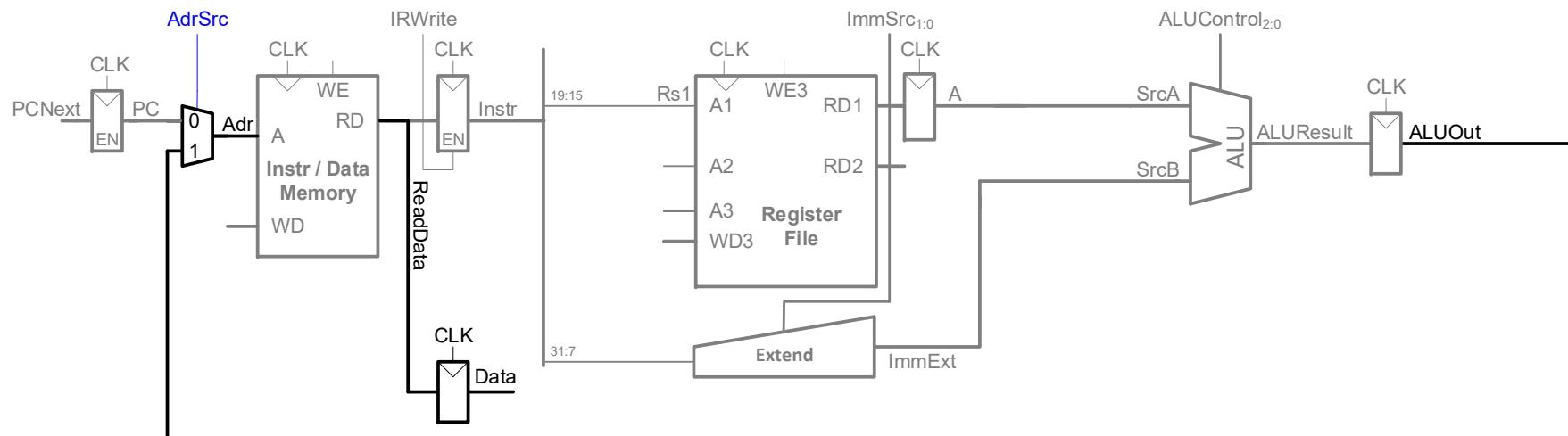
- **Fase 3:** Calcular la dirección de memoria
 - Nuevo Registro *ALUout* para almacenar el valor calculado en la ALU



Ruta de datos multiciclo: lw



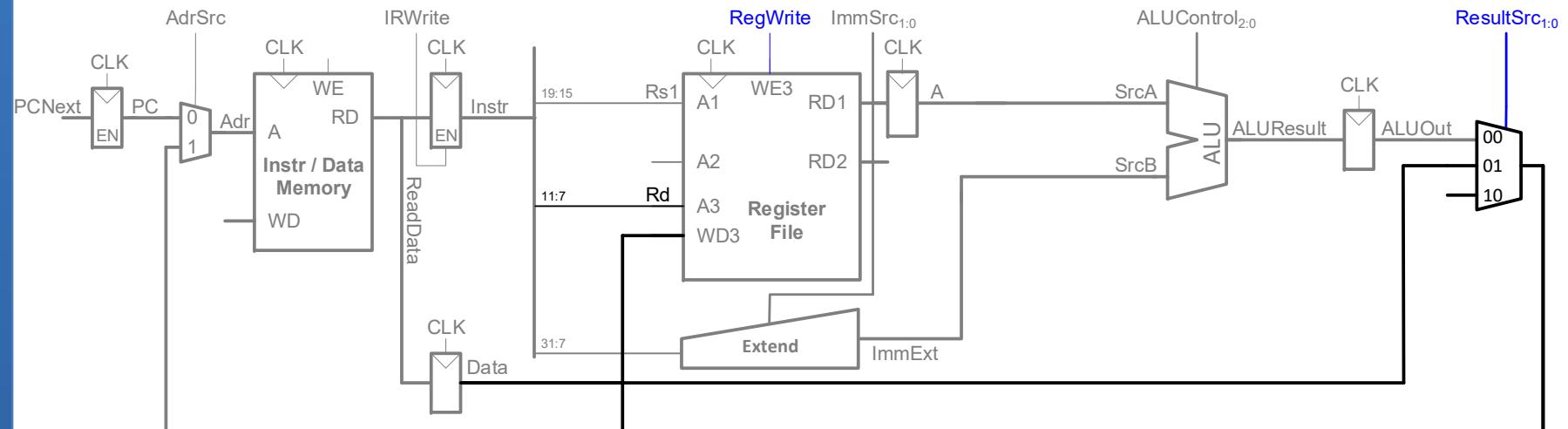
- **Fase 4:** Leer datos de memoria
 - Nuevo Registro *Data* para guardar lo leído de memoria





Ruta de datos multiciclo: lw

- **Fase 5:** Escribir datos en el Banco de registros

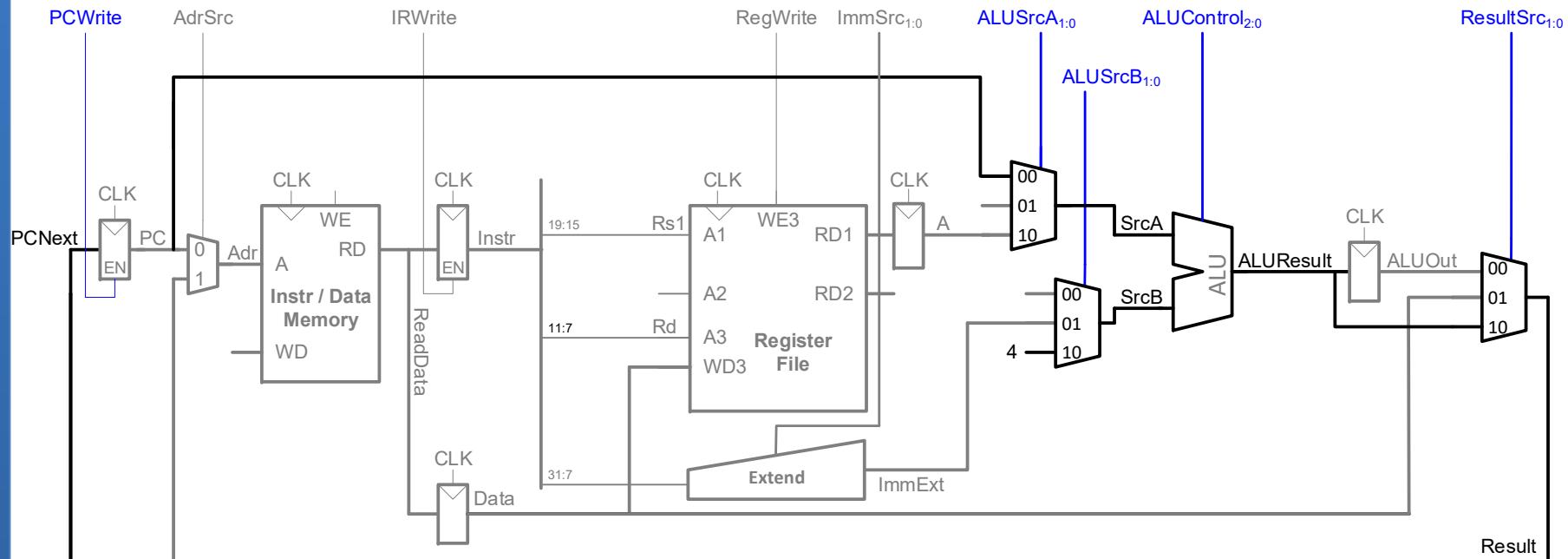




Ruta de datos multiciclo: lw

- **Fase 6:** Incrementa el PC

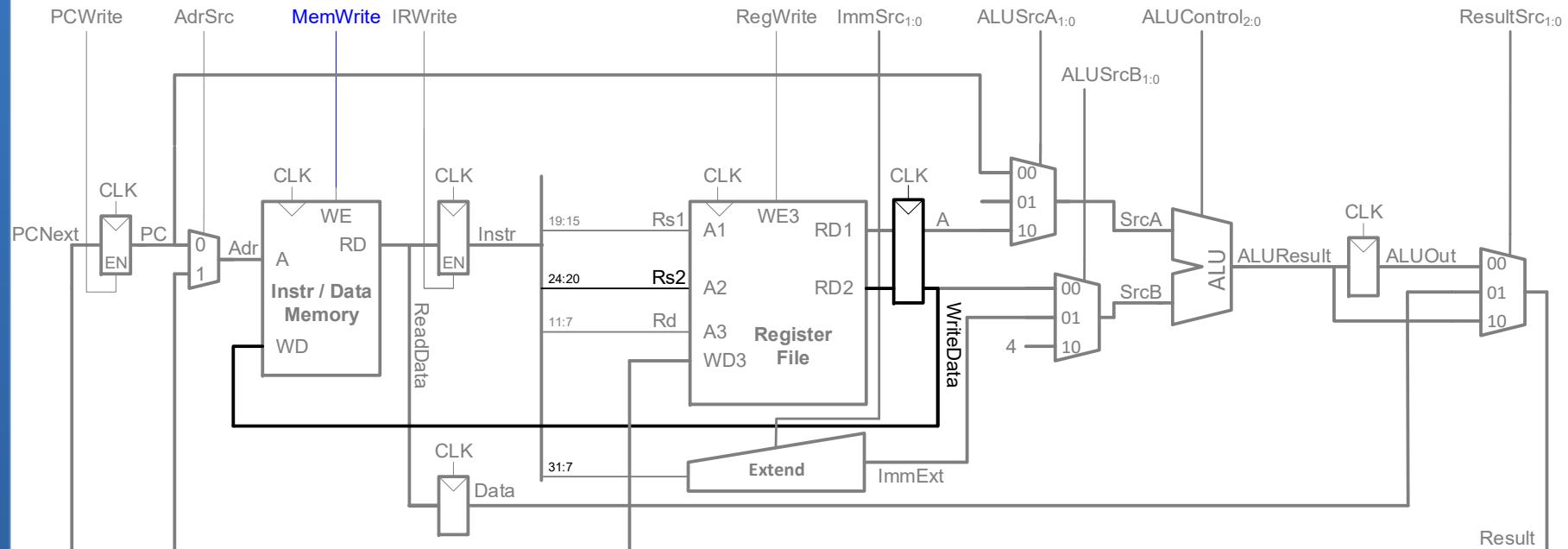
- Añadir dos **Mux**
- Reusamos la ALU





Ruta de datos multiciclo: sw

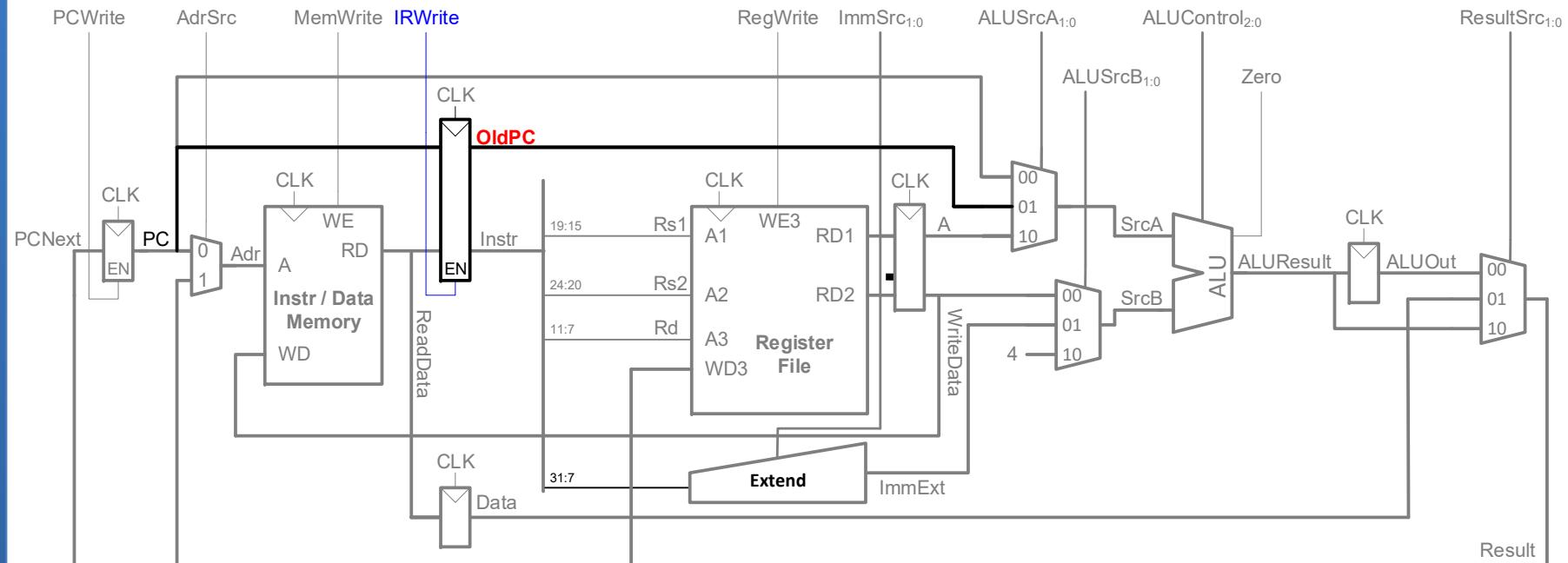
- Escribe el dato contenido en Rn en la memoria:
 - Nuevo registro *WriteData* que almacena el dato a escribir en memoria





Ruta de datos multiciclo: beq

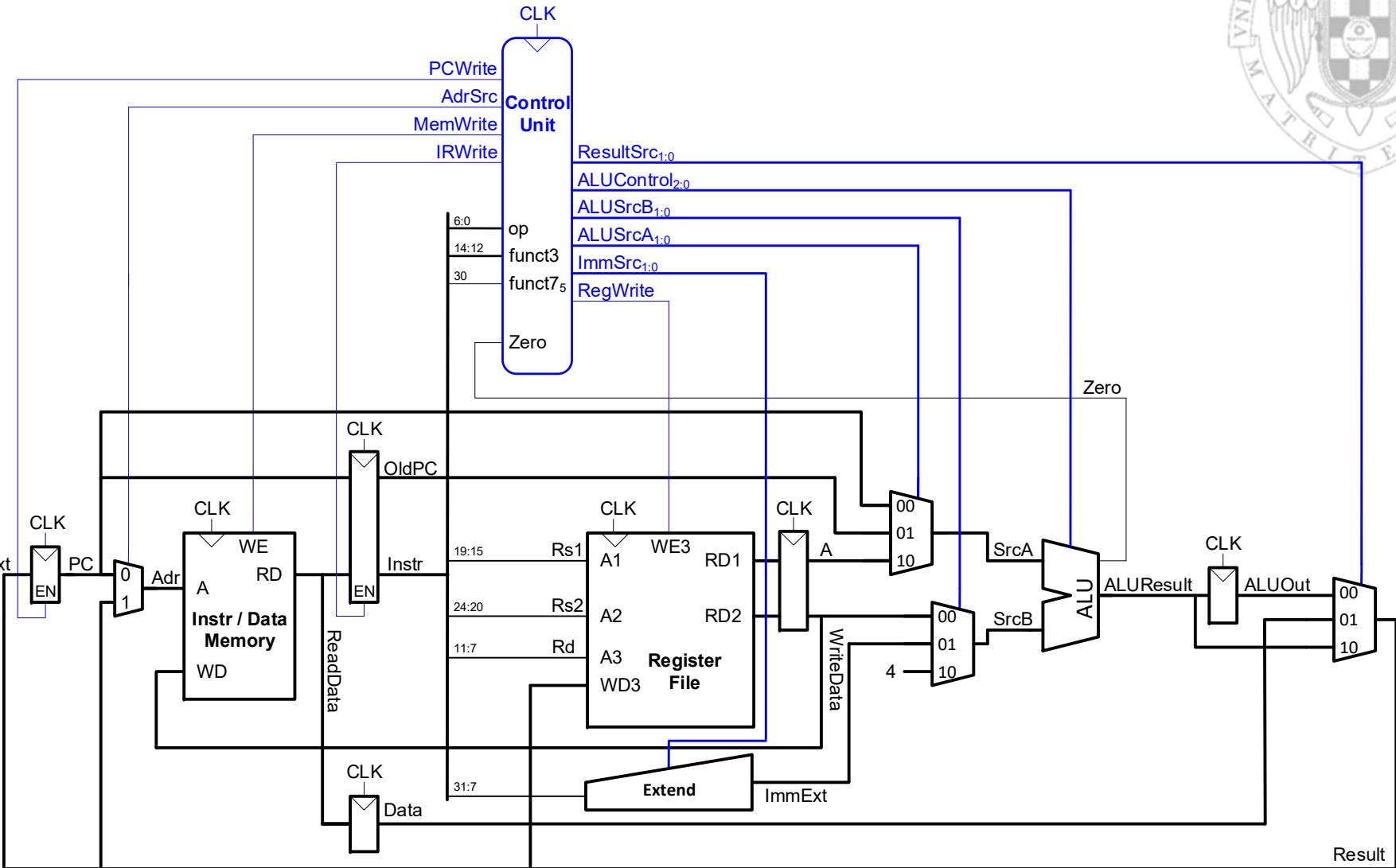
- Branch Target Address (BTA)
 - $BTA = PC + Imm$



PC se actualiza en la fase de búsqueda (IF), se necesita salvar el **old (actual) PC**



Unidad de control multiciclo

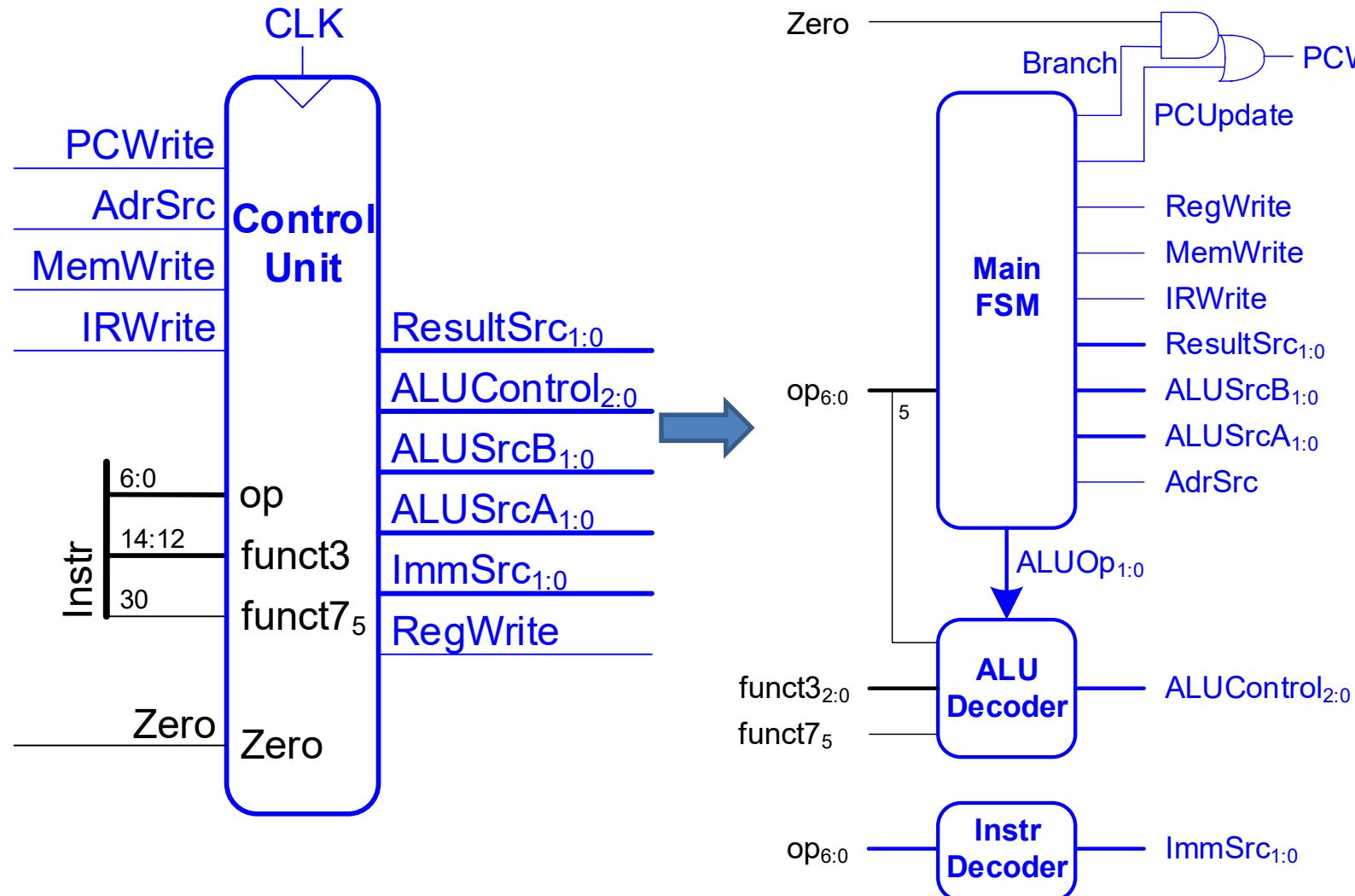


¿Por qué algunos registros no tienen señal de escritura?



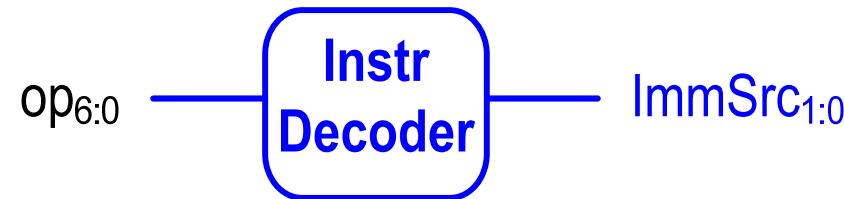
Unidad de control multiciclo

- Los módulos Decodificador ALU y Lógica del PC son iguales a la implementación monociclo





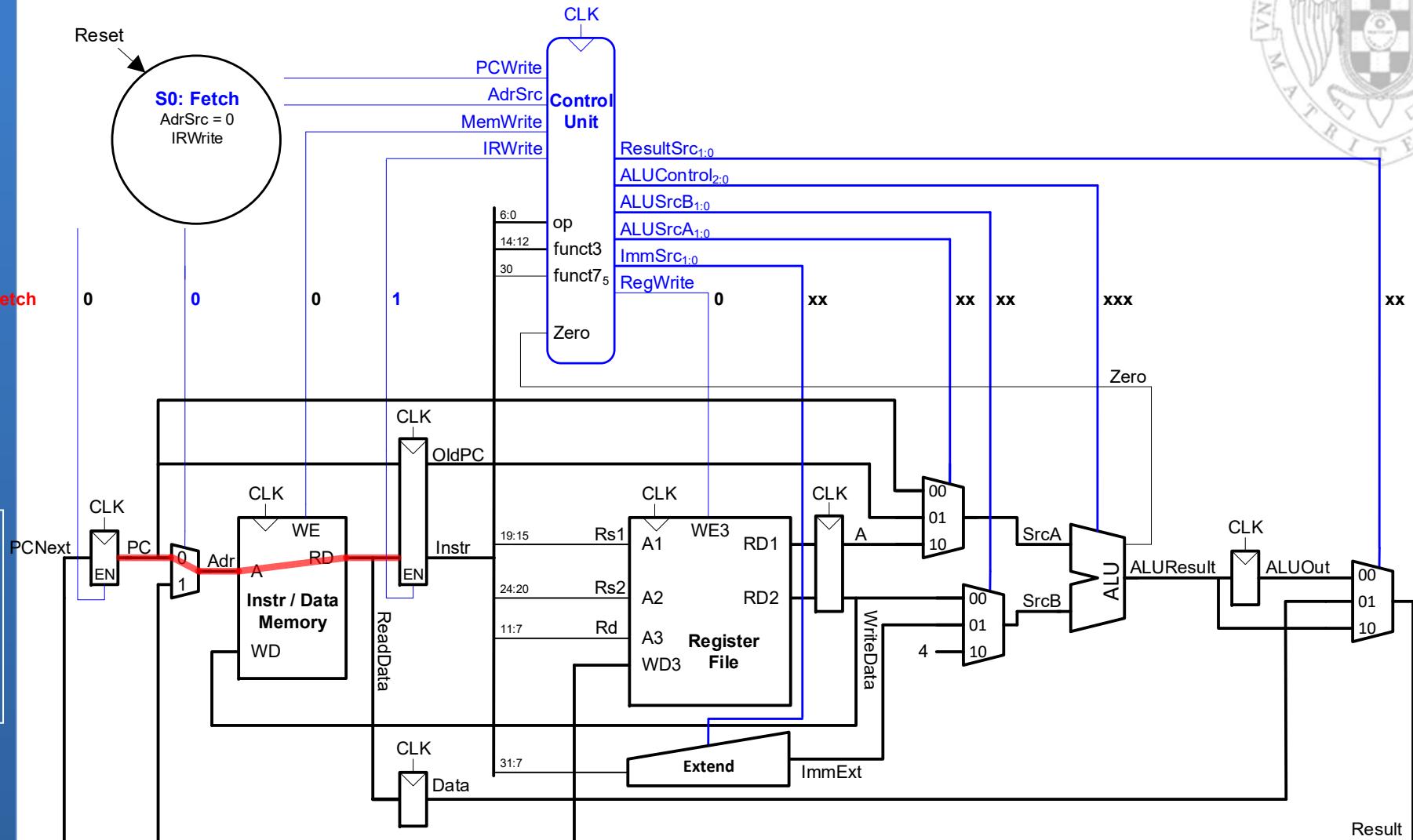
Unidad de control multiciclo



op	Instruction	ImmSrc
3	lw	00
35	sw	01
51	R-type	XX
99	beq	10



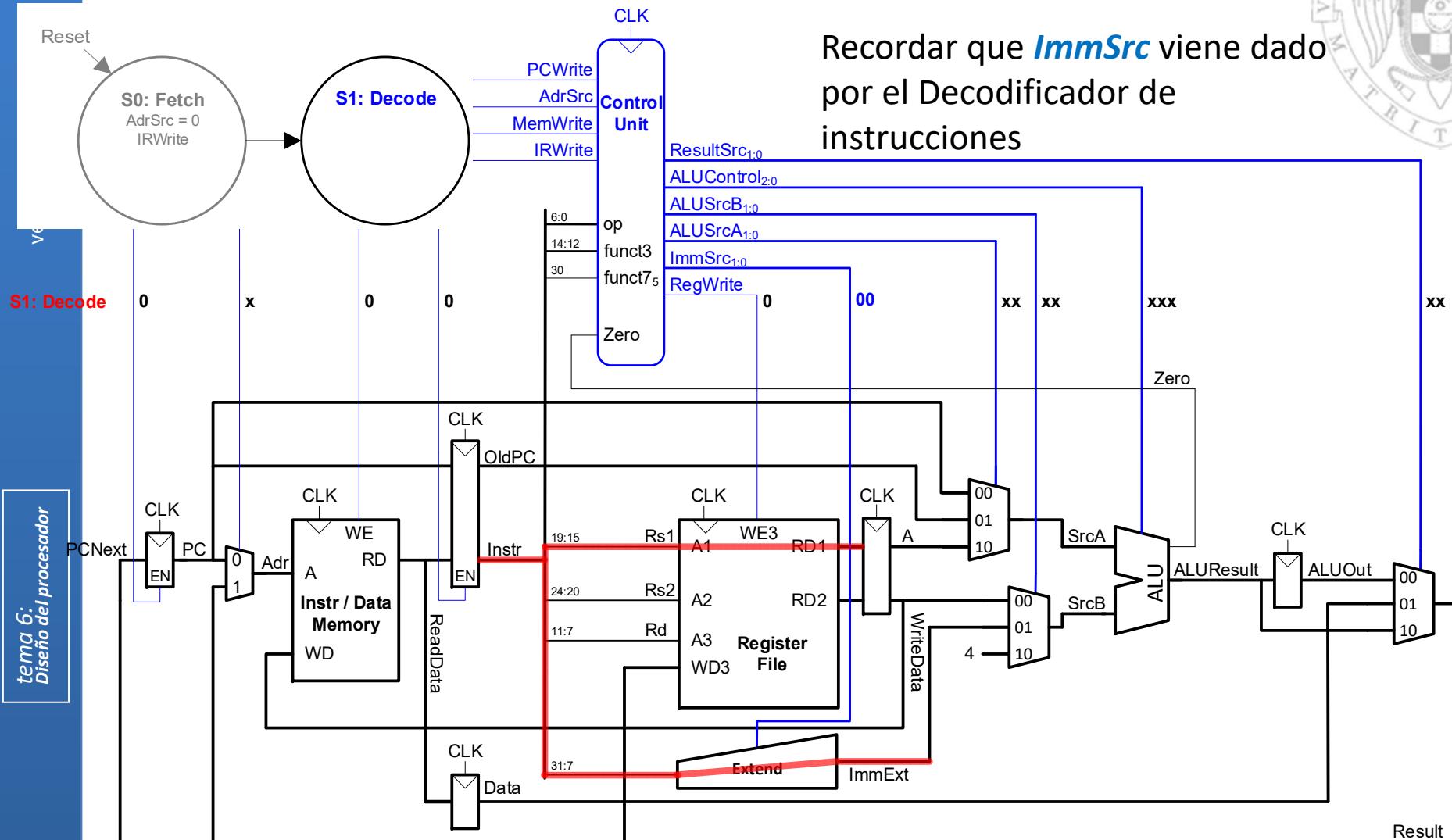
Unidad de control multiciclo: Fetch



- Las señales de escritura (RegWrite, MemWrite, IRWrite, PCUpdate, and Branch) son **0** si no aparecen en un estado.
- El resto de las señales que no aparecen son don't cares



Unidad de control multiciclo: Decode





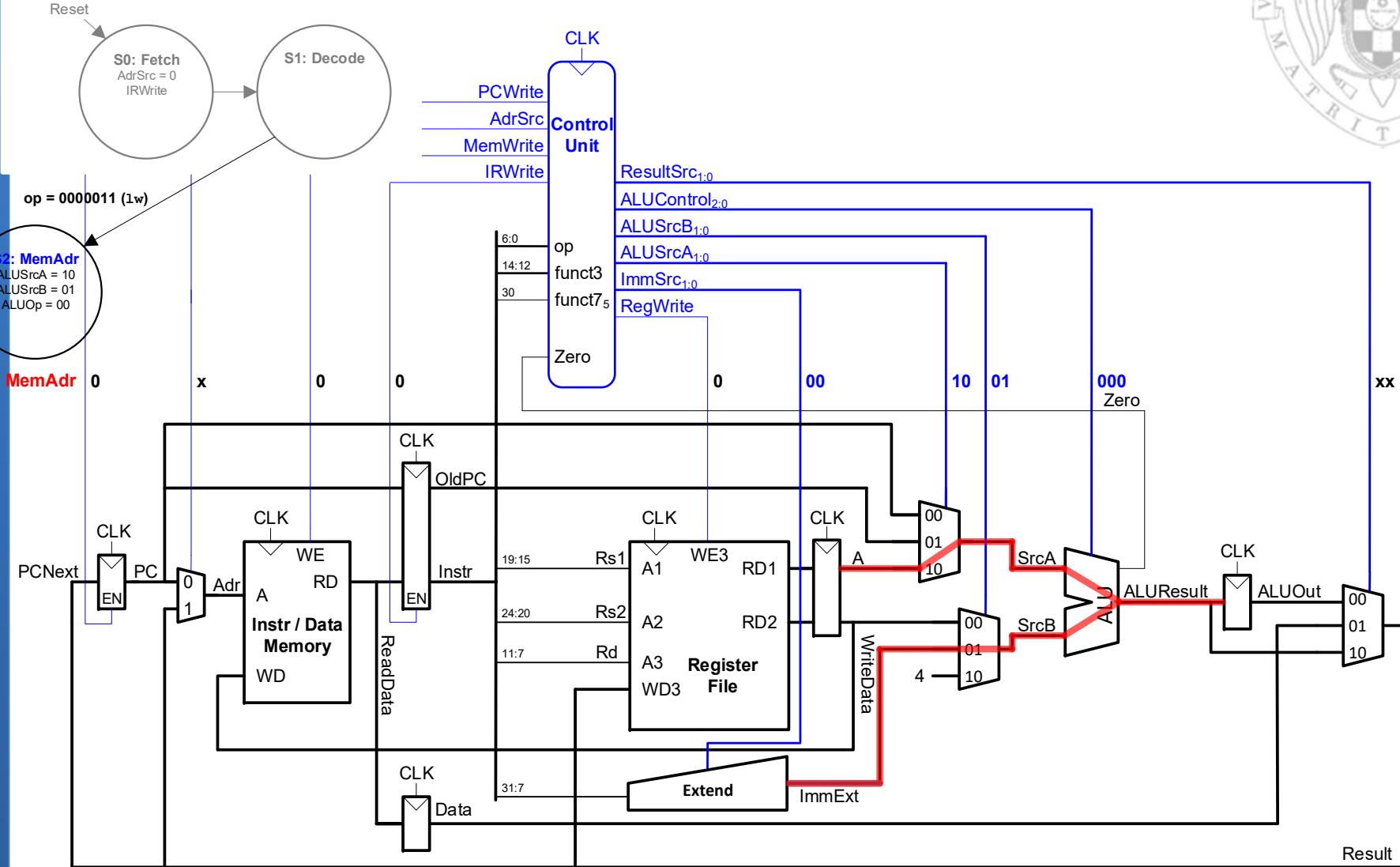
Unidad de control multiciclo: dirección

versión 2021

tema 6:
Diseño del procesador

FC

68





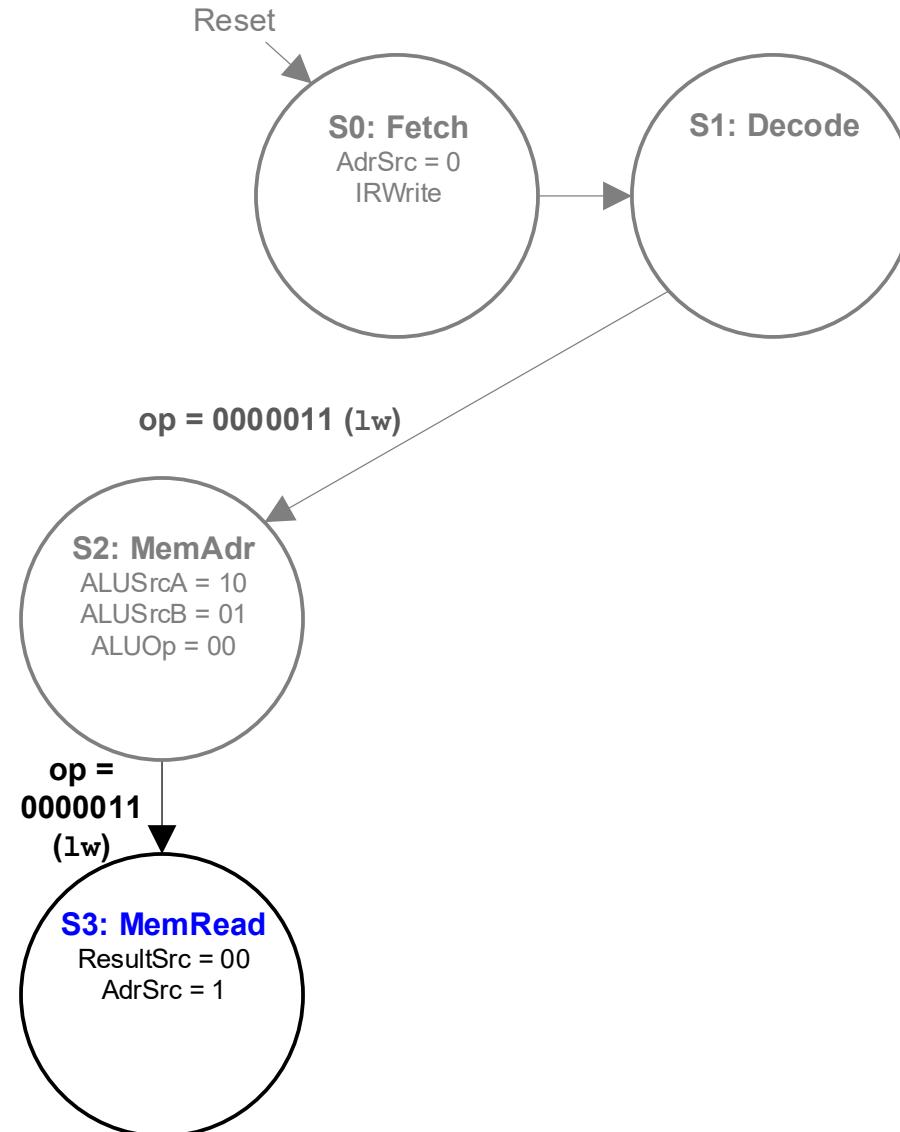
Control multiciclo: leer memoria

versión 2021

tema 6:
Diseño del procesador

FC

69





Control multiciclo: leer memoria

Versión 2020

S3: MemRead

ResultSrc = 00
AdrSrc = 1

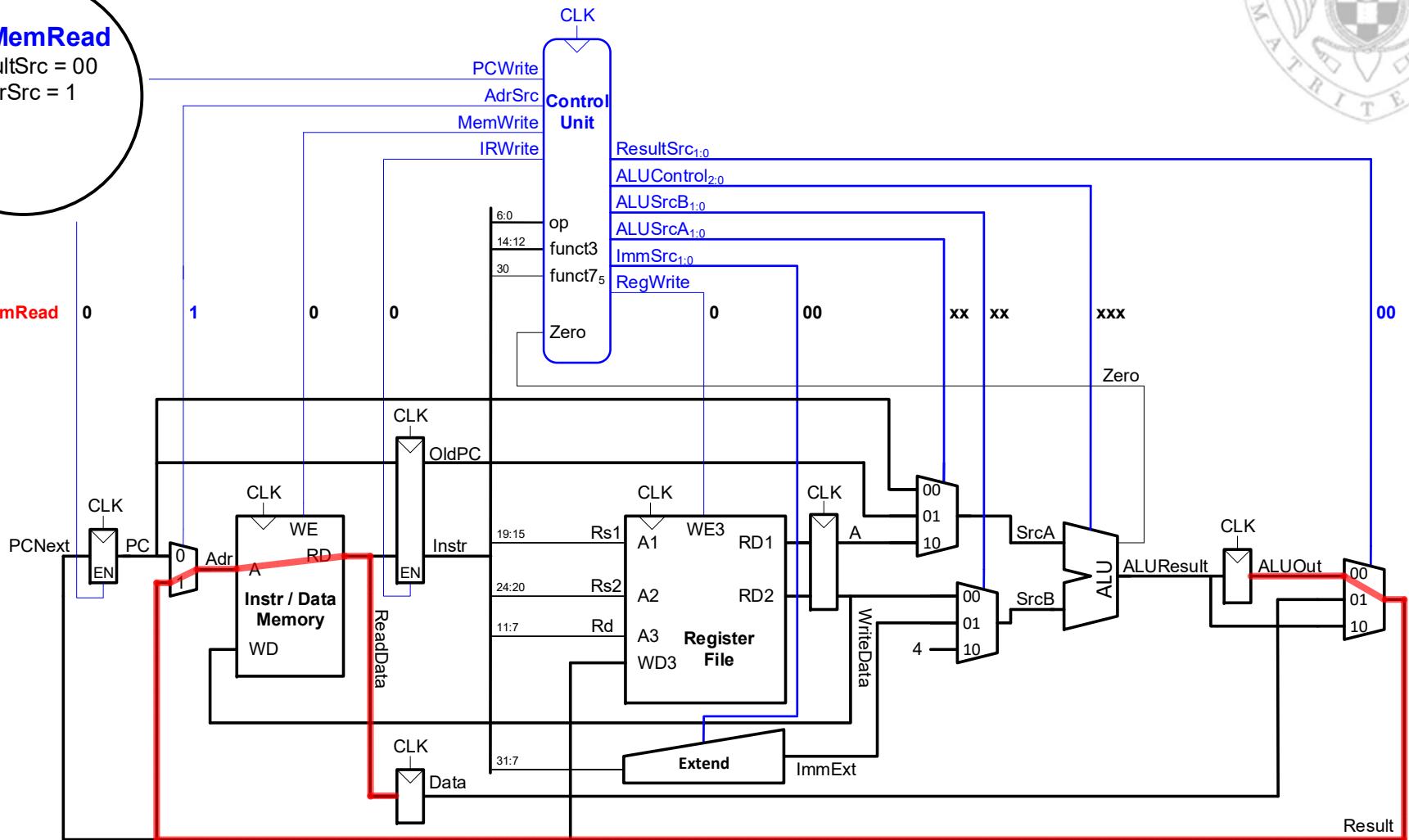
S3: MemRead

PCNext

Tema 6:
Diseño del procesador

FC

70





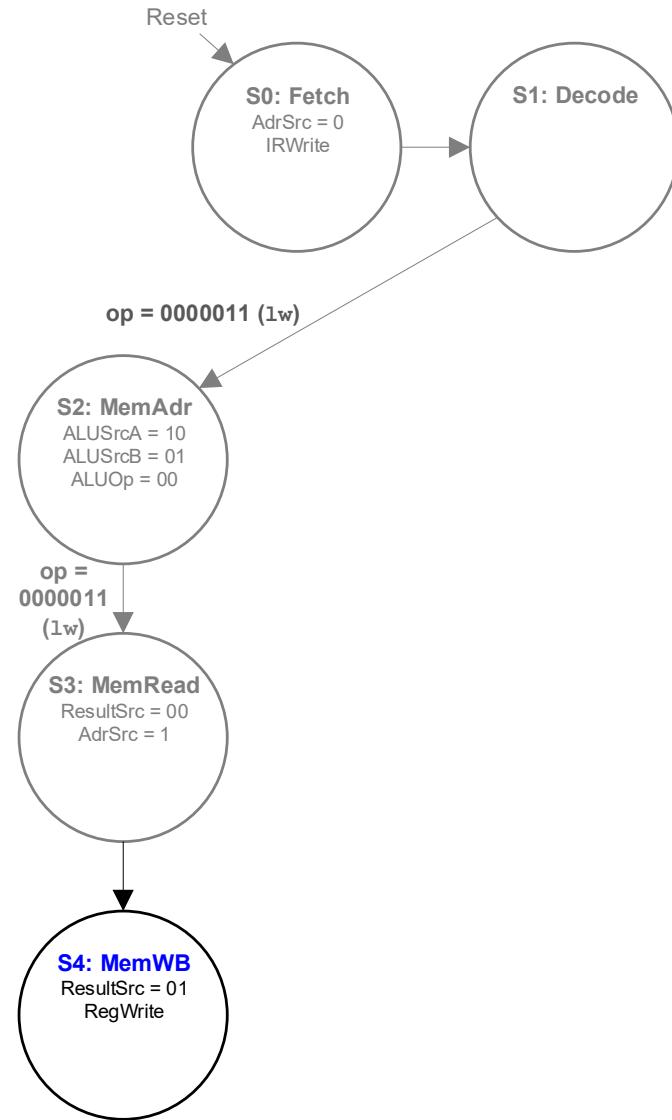
Control multiciclo: escribir BR

versión 2021

tema 6:
Diseño del procesador

FC

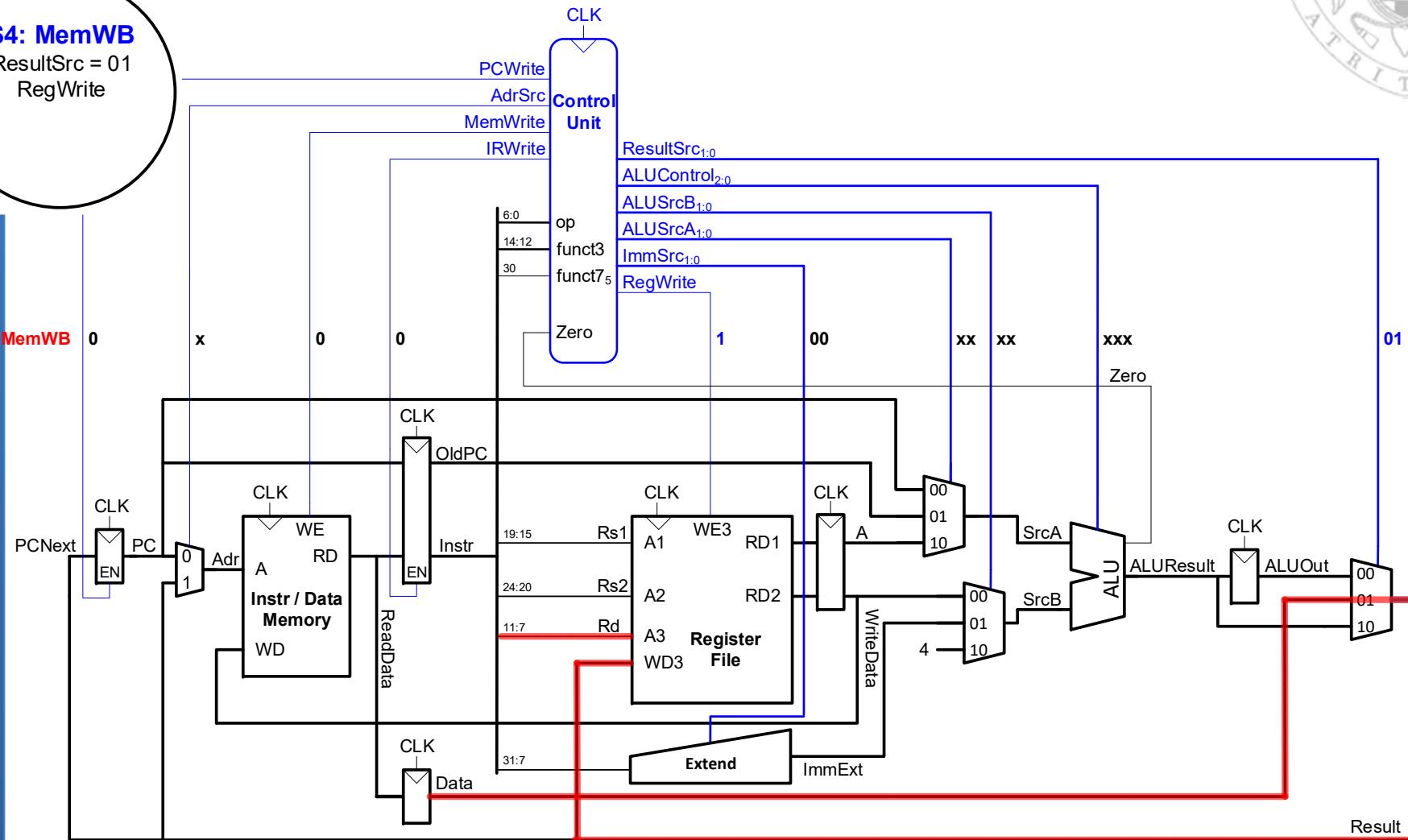
71





Control multiciclo: escribir BR

S4: MemWB
ResultSrc = 01
RegWrite





Control multiciclo: Fetch

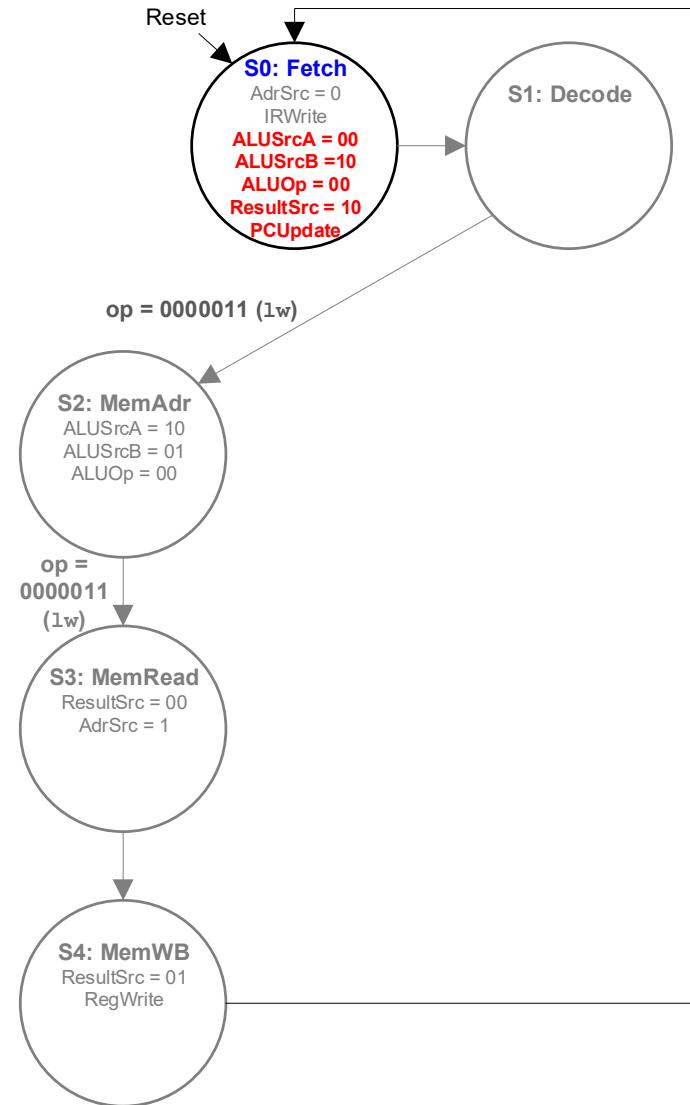
versión 2021

tema 6:
Diseño del procesador

FC

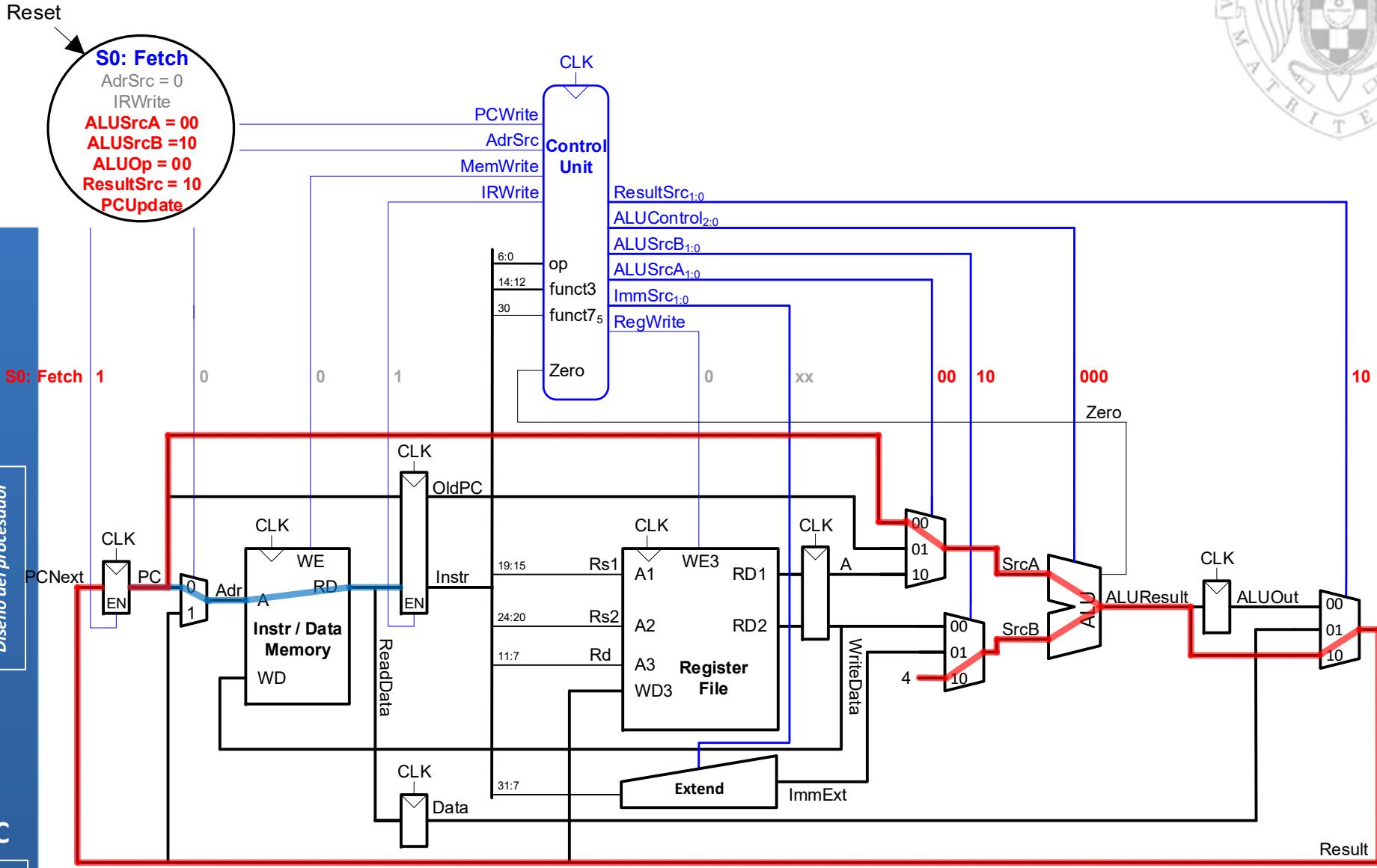
73

Calcular **PC+4** en la fase Fetch dado que la ALU no se está usando



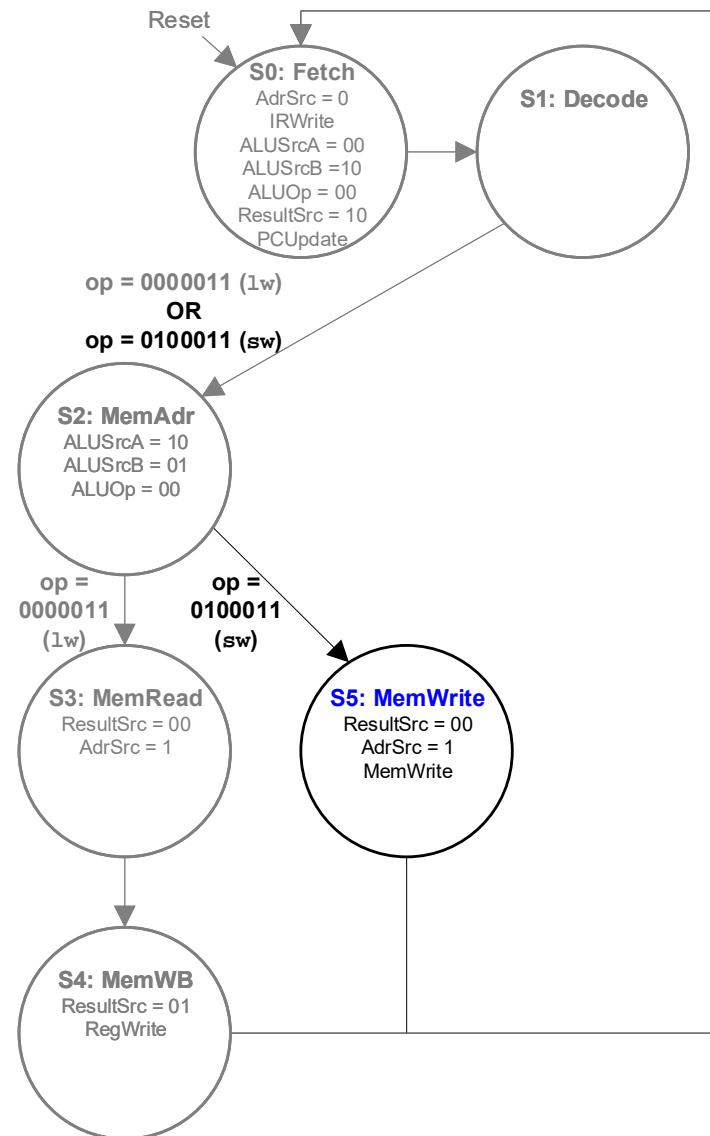


Control multiciclo: Fetch





Control multiciclo: sw

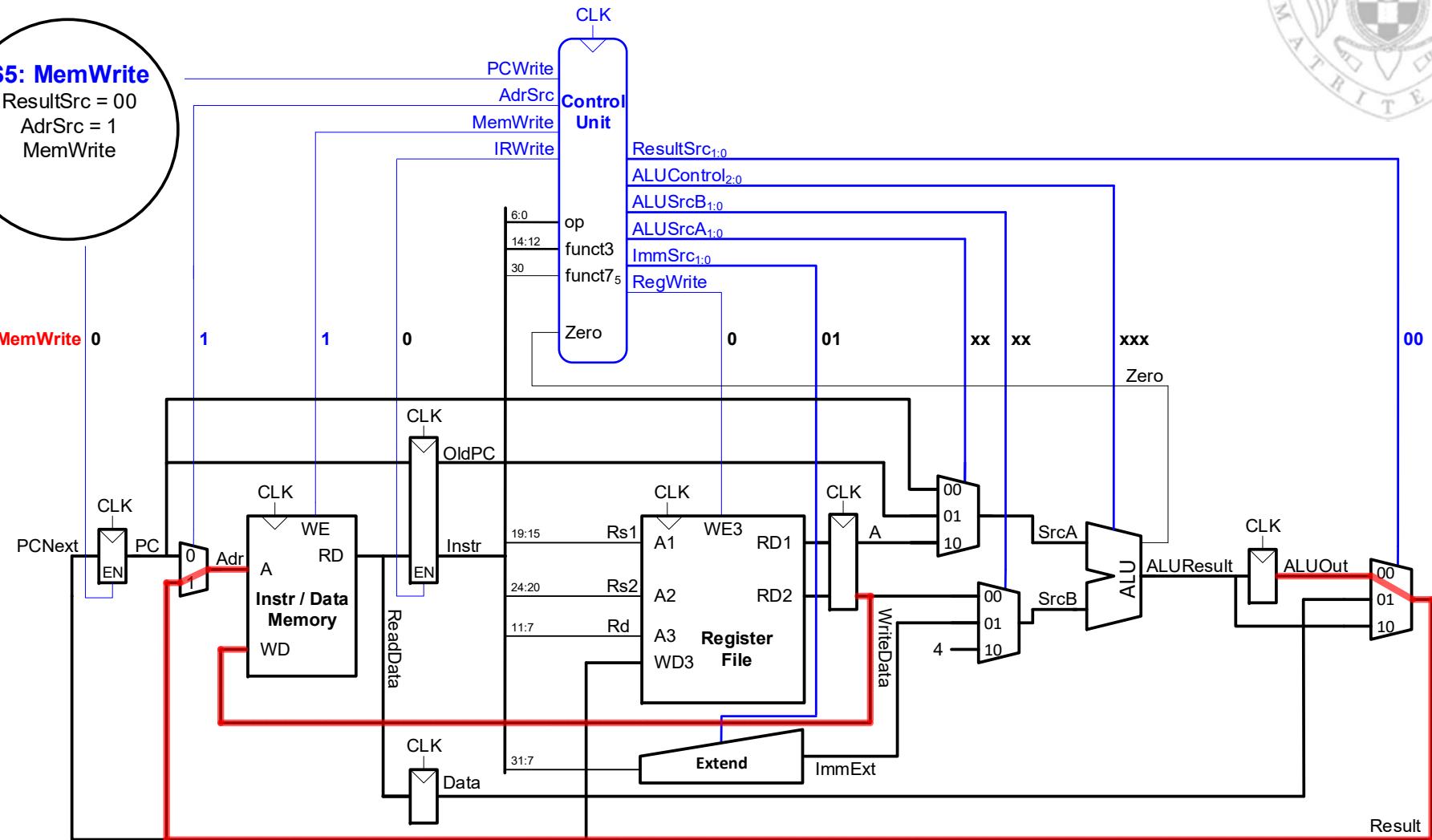




Control multiciclo: sw

Versión 2021

S5: MemWrite
 ResultSrc = 00
 AdrSrc = 1
 MemWrite



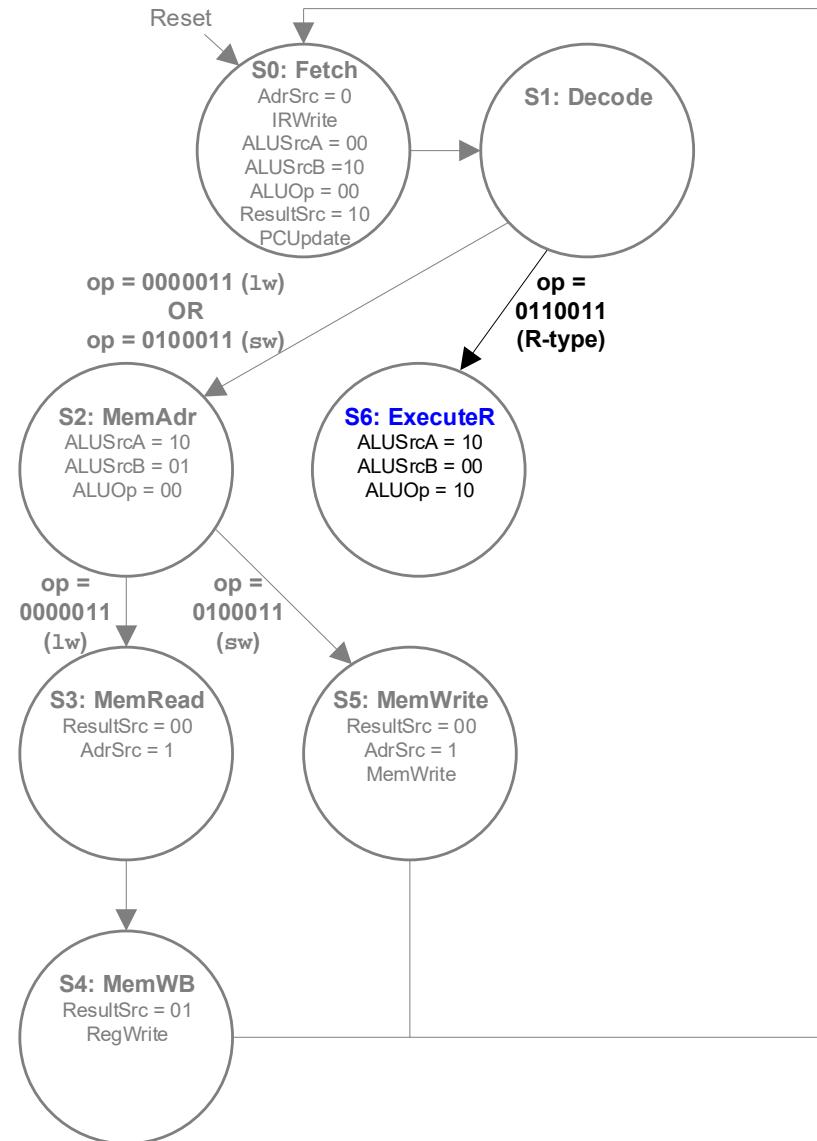
tema 6:
 Diseño del procesador

FC

76

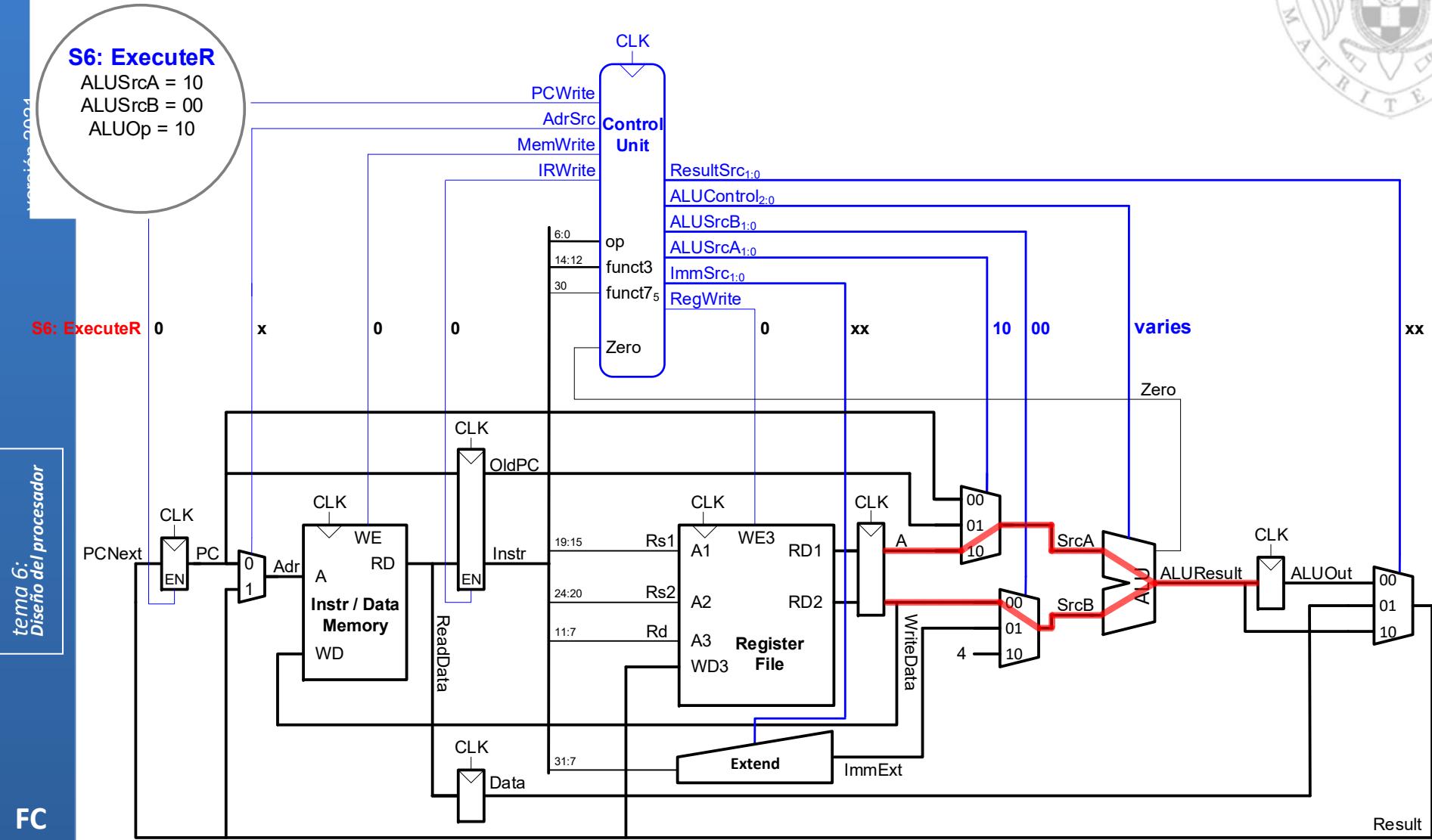


Control multiciclo: Tipo R (ex)



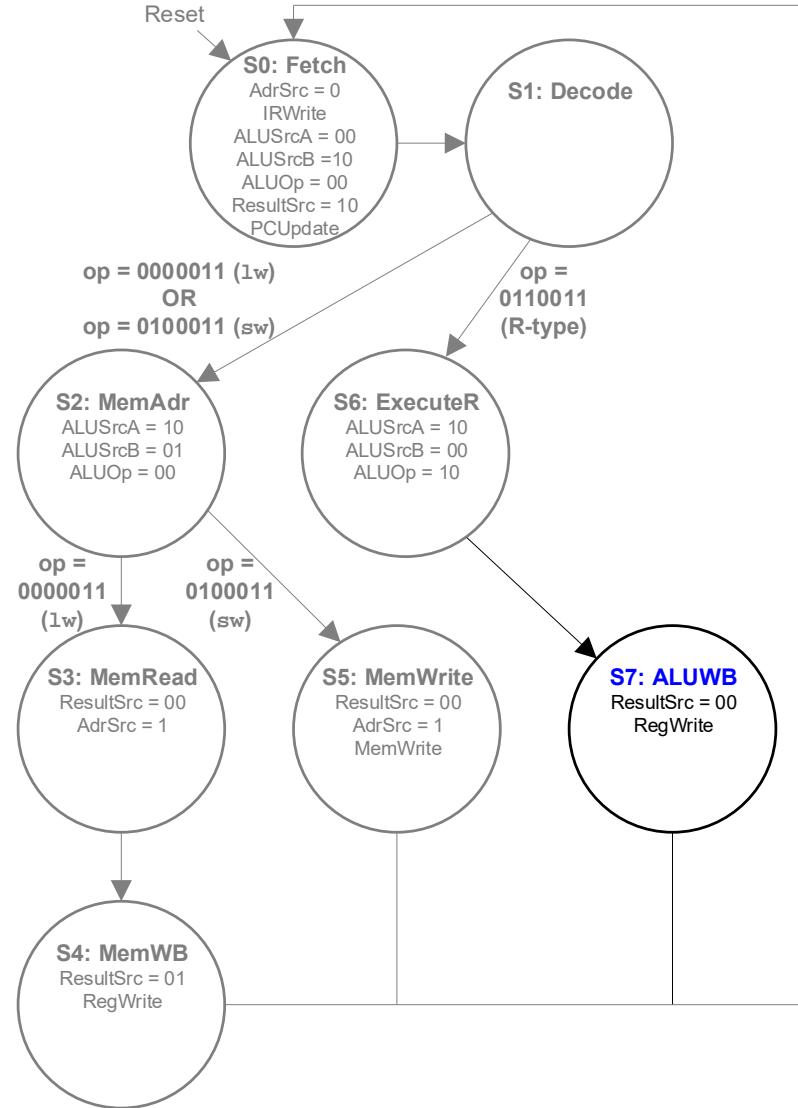


Control multiciclo: Tipo R (ex)





Control multiciclo: Tipo R (WB)





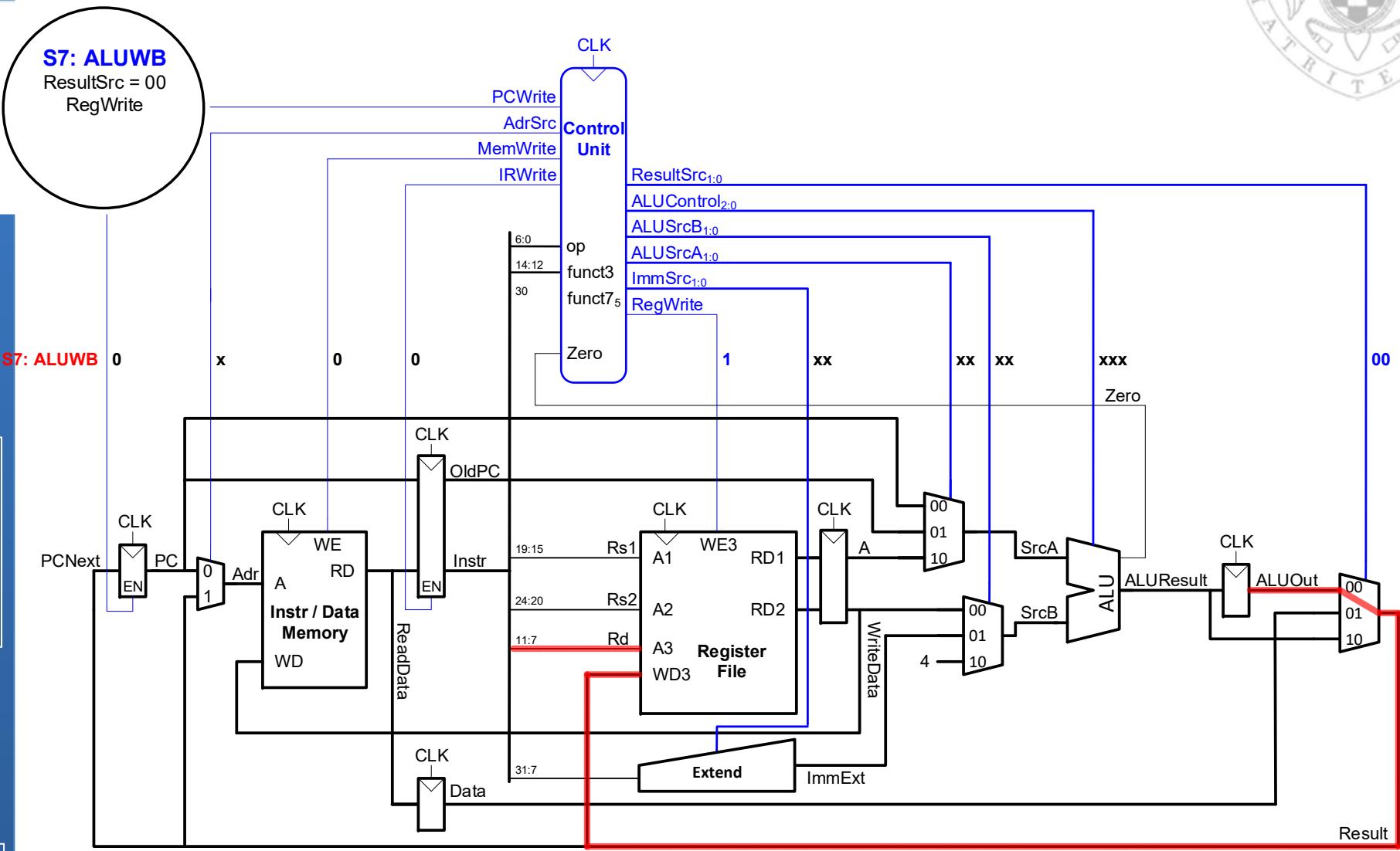
Control multiciclo: Tipo R (WB)

Versión 2021

tema 6:
Diseño del procesador

FC

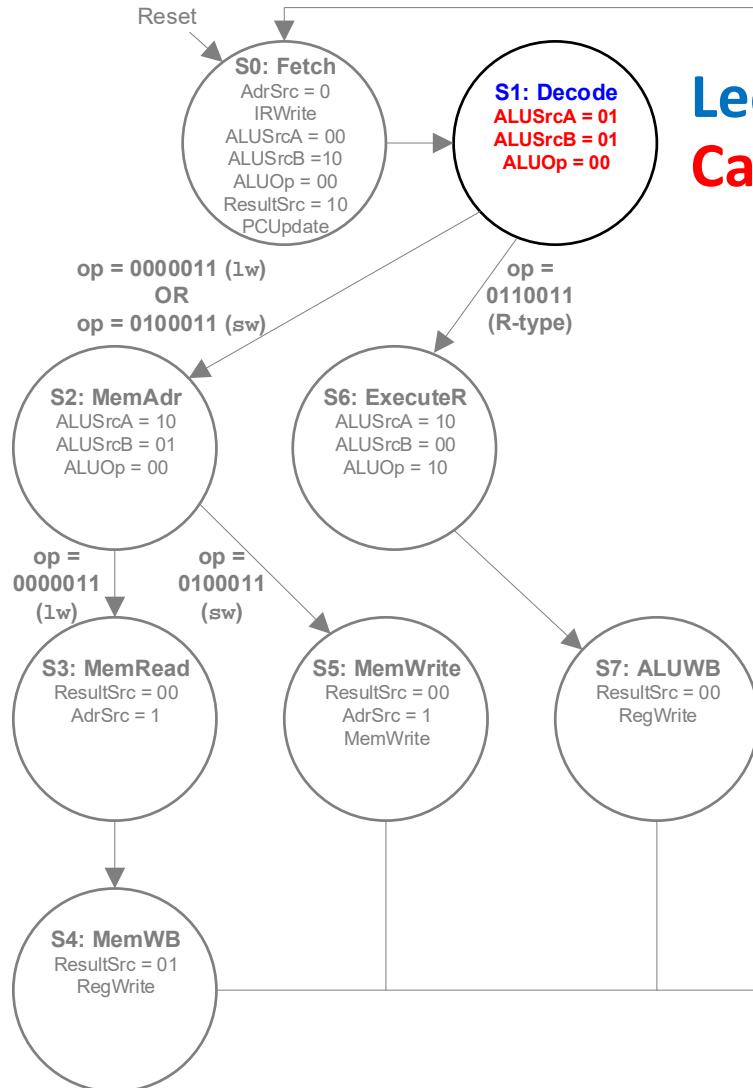
80





Control multiciclo: Tipo beq

Lee Registros y
Calcula Target Address (PC+imm)



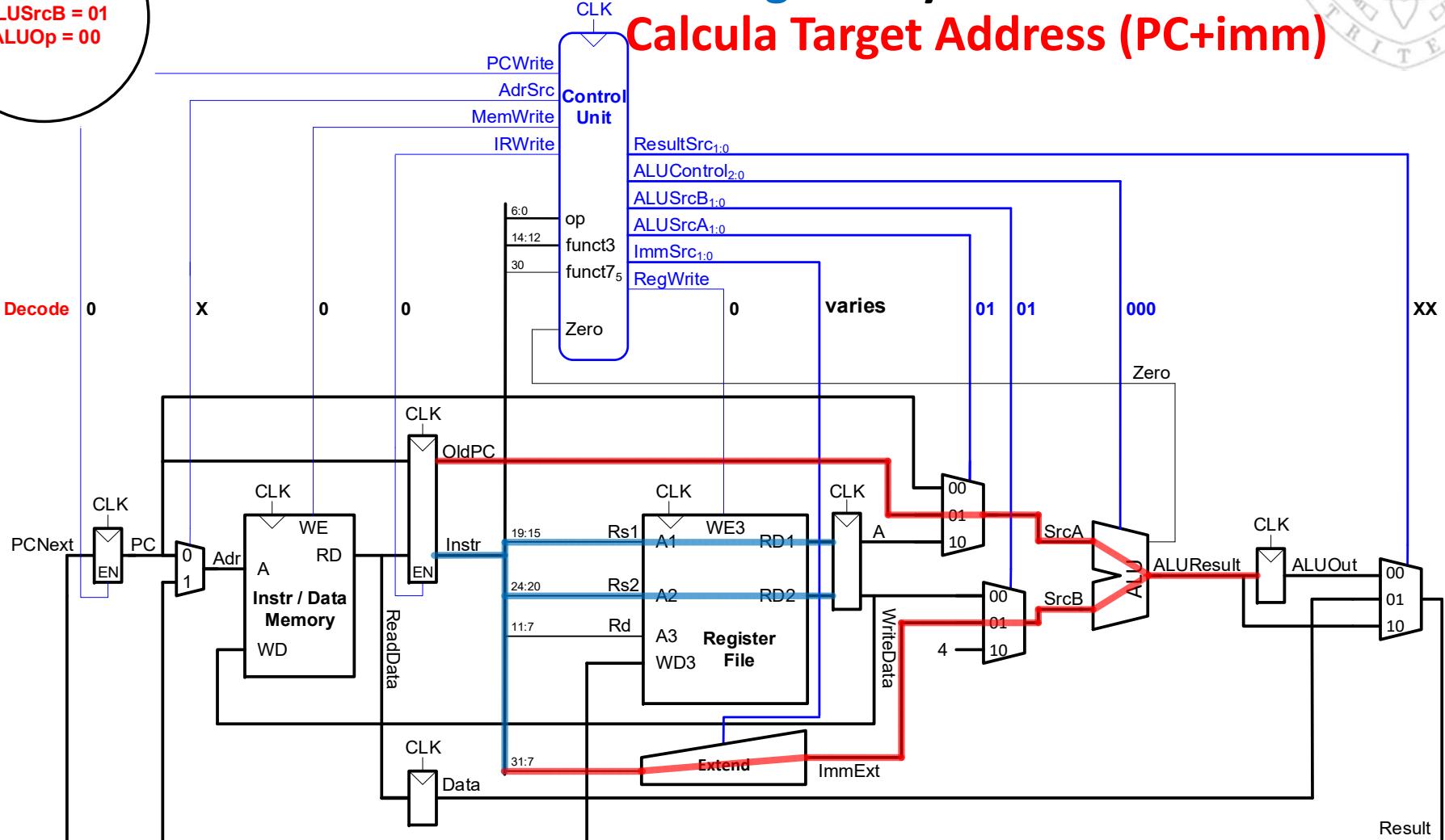
- Se necesita calcular:
 - El destino del salto
 - rs1- rs2 para ver si son iguales
- Como la ALU no se usa en la fase Decode la usamos para calcular PC + Imm



Control multiciclo: beq (Decode)

S1: Decode
ALUSrcA = 01
ALUSrcB = 01
ALUOp = 00

Lee Registros y
Calcula Target Address (PC+imm)





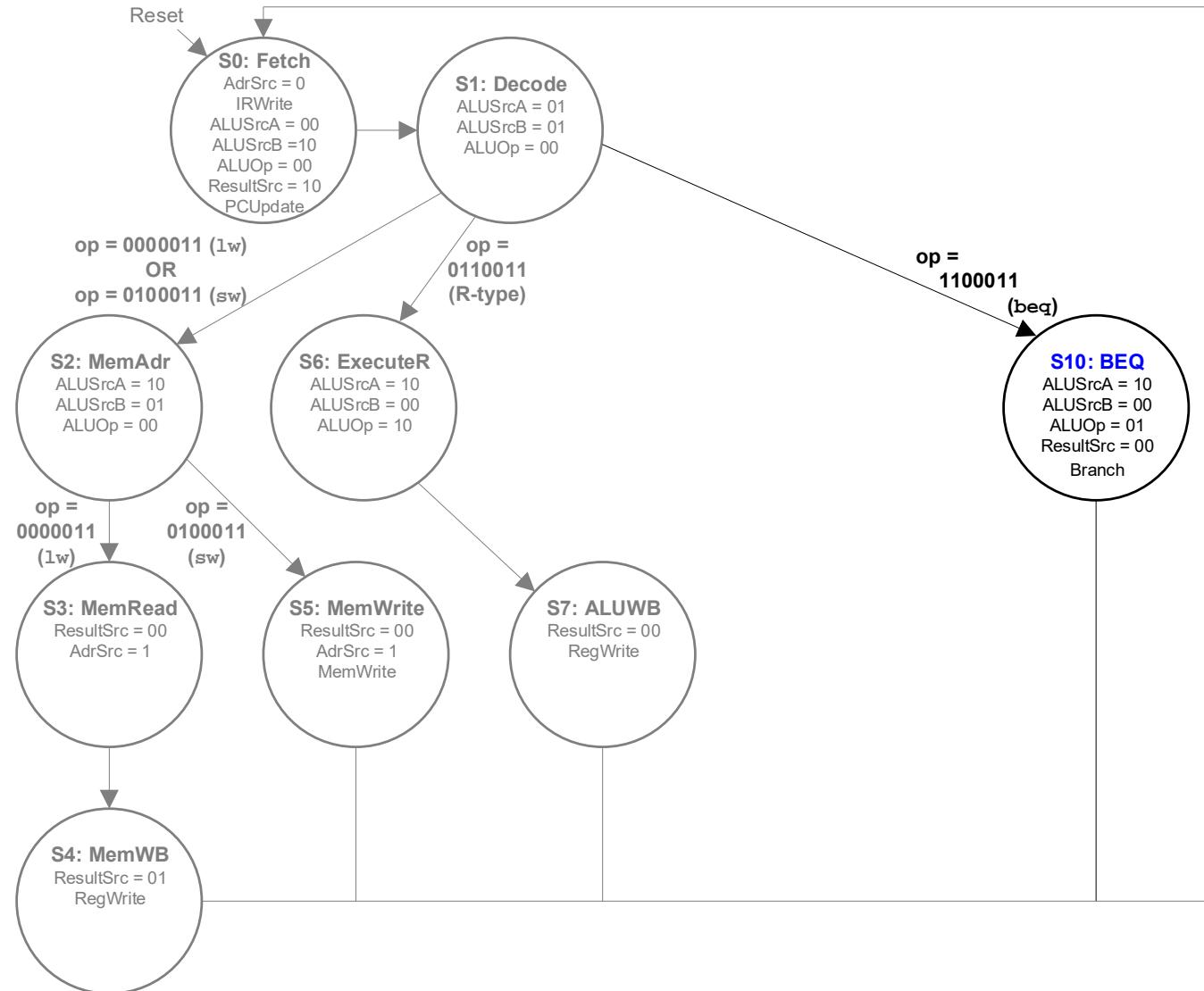
Control multiciclo: beq

versión 2021

tema 6:
Diseño del procesador

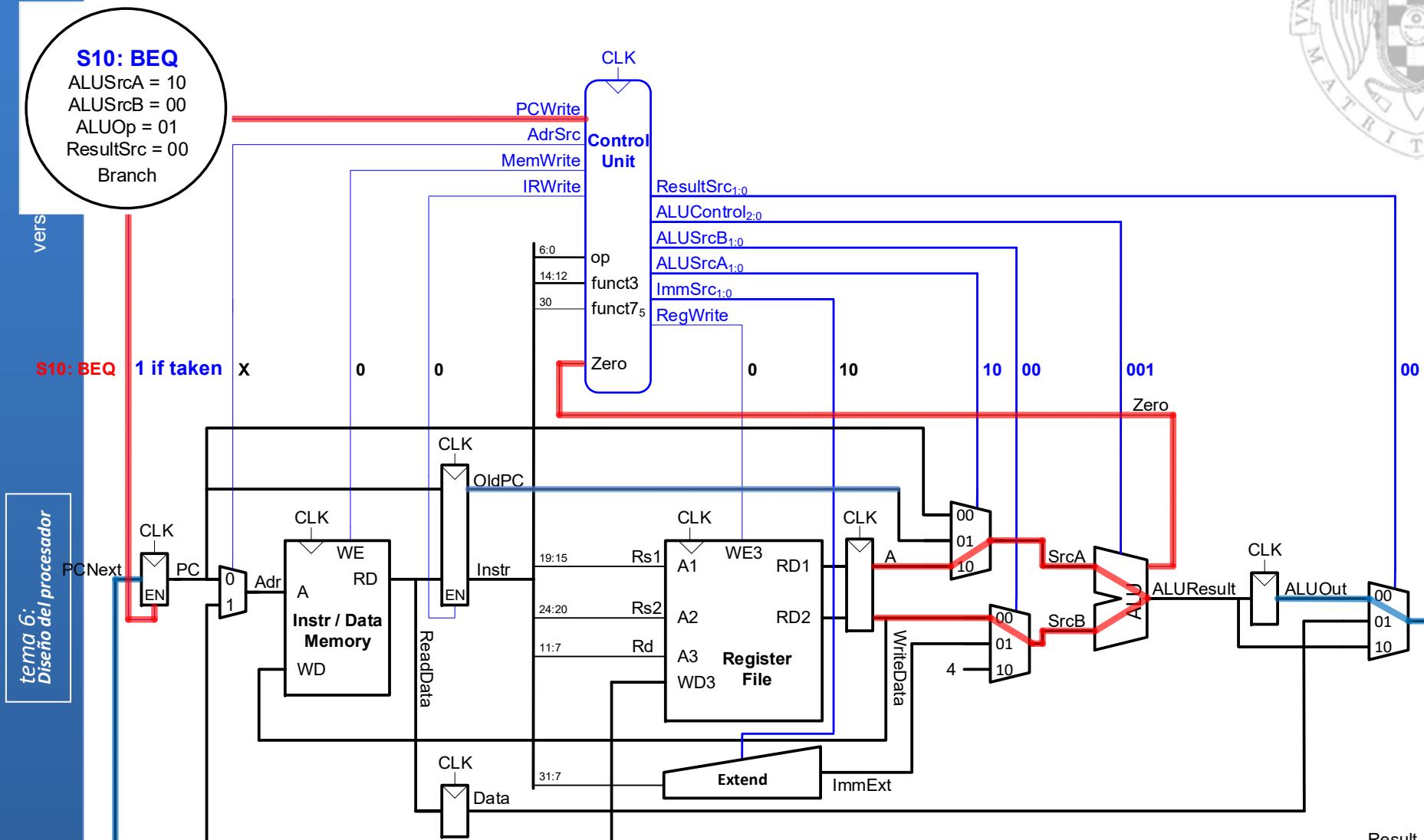
FC

83



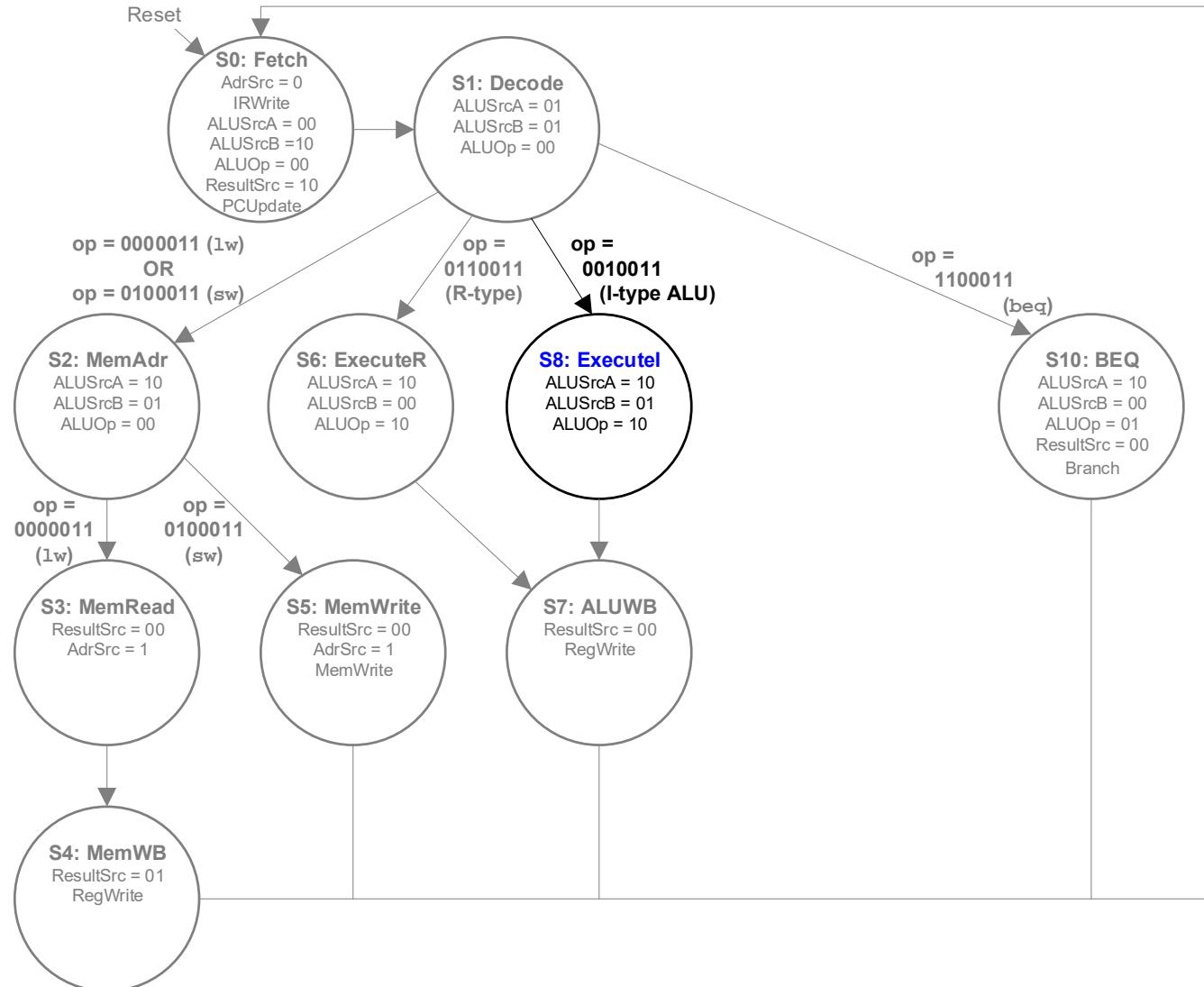


Control multiciclo: beq





Control multiciclo: Tipo I





Control multiciclo: Tipo I

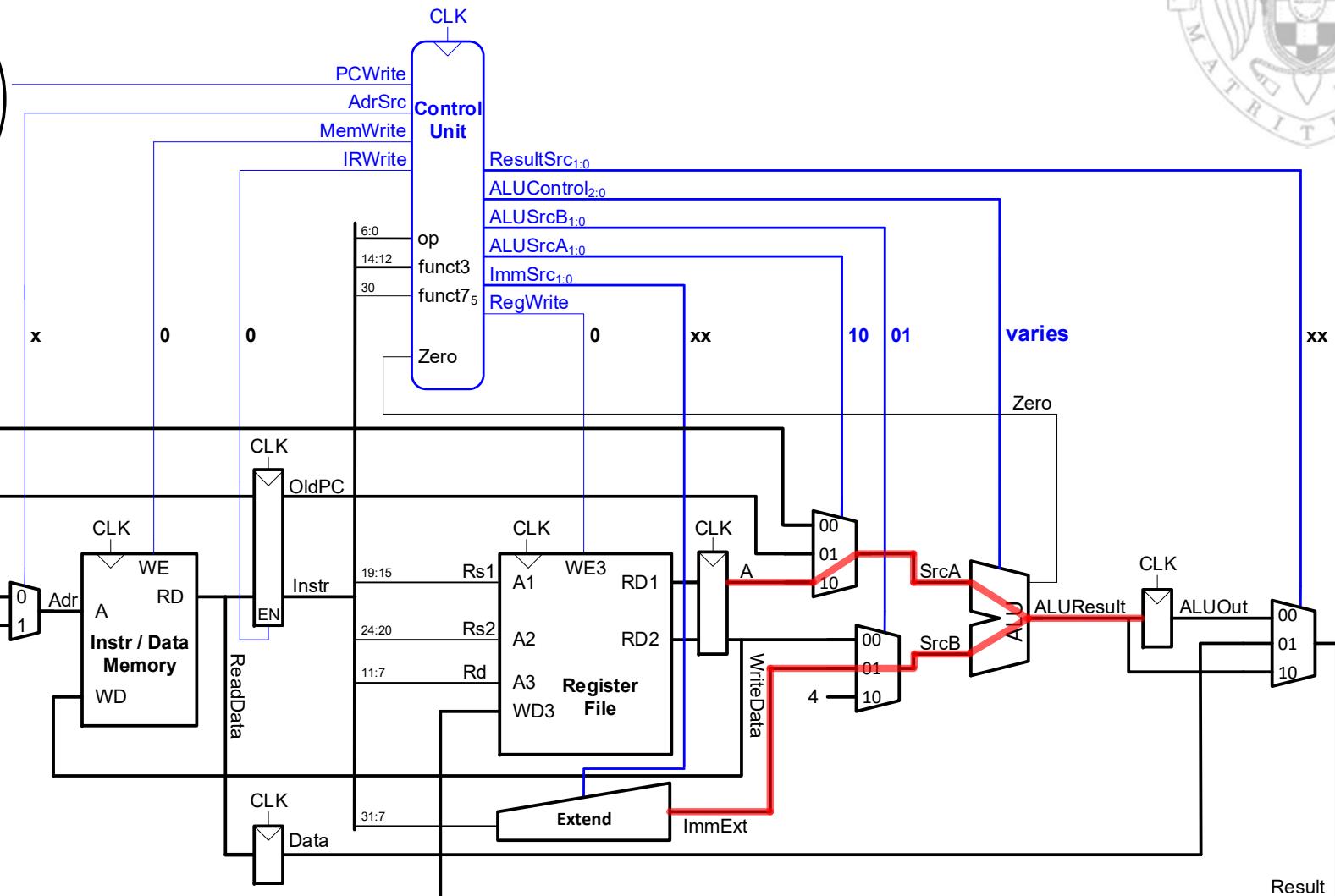
S8: Executel

ALUSrcA = 10
ALUSrcB = 01
ALUOp = 10

ver

S8: Executel

0

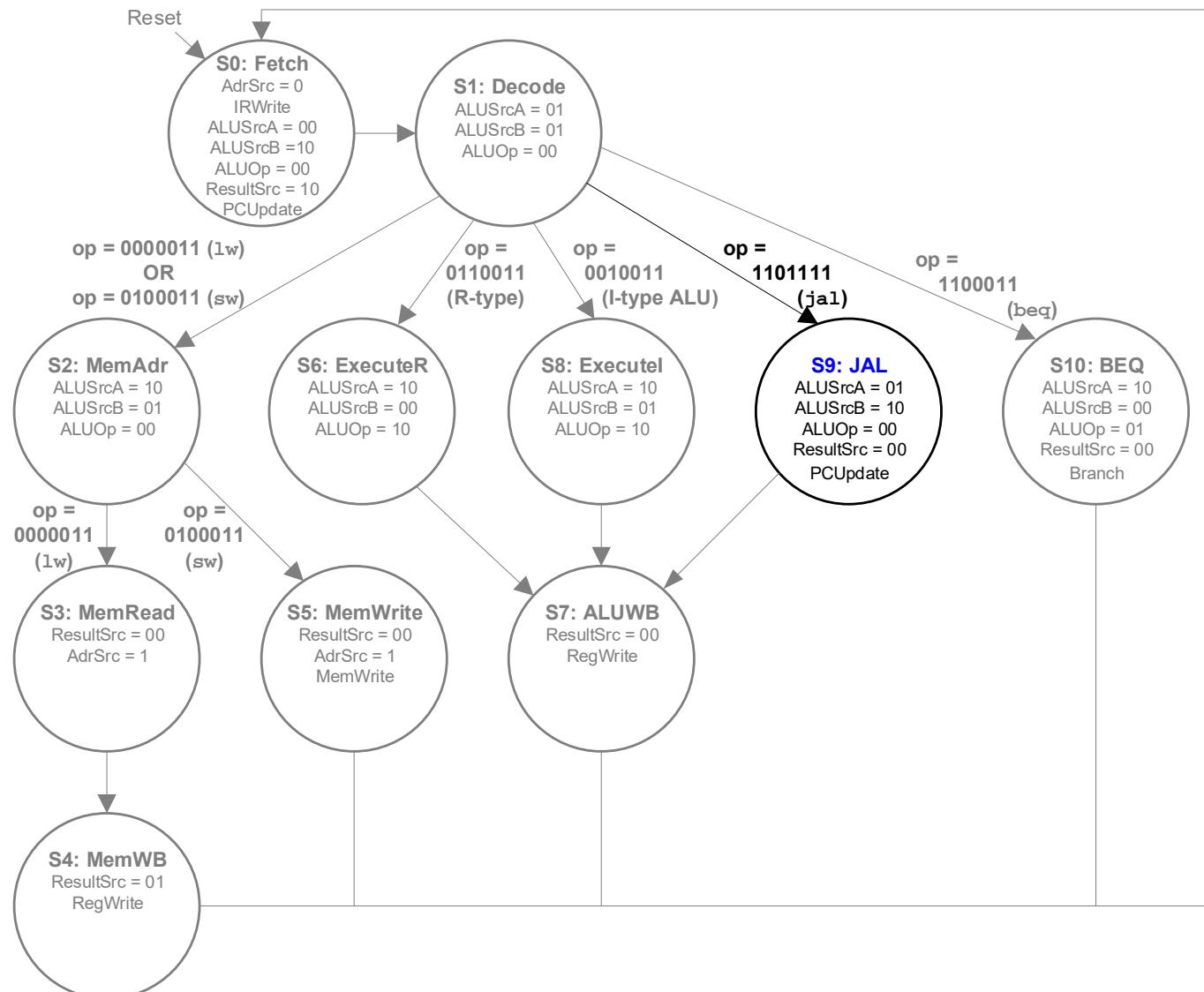


Tema 6:
Diseño del procesador

FC

86

Control multiciclo: jal

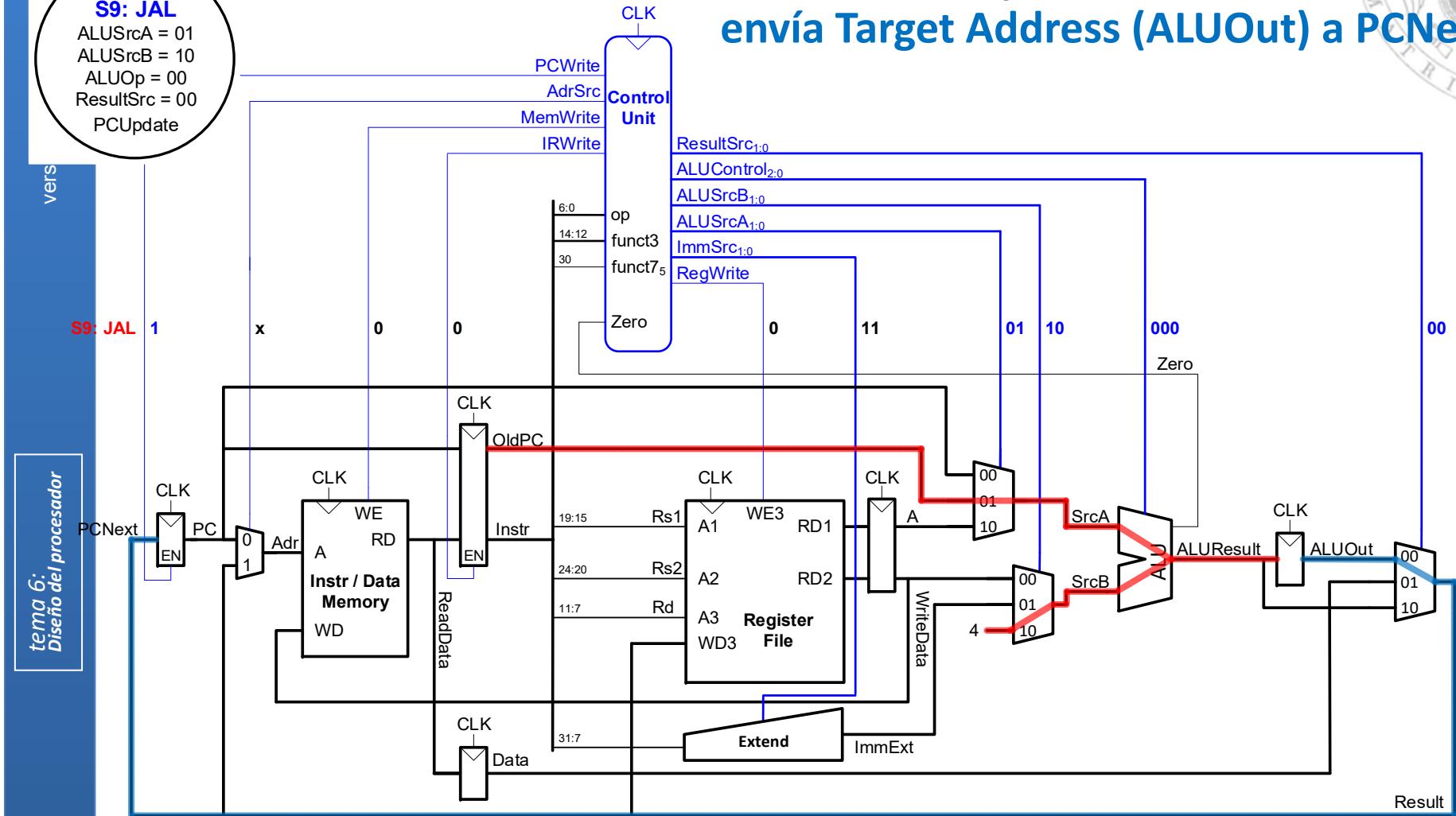




Control multiciclo: jal

Calcula PC + 4 y
envía Target Address (ALUOut) a PCNext

S9: JAL
 ALUSrcA = 01
 ALUSrcB = 10
 ALUOp = 00
 ResultSrc = 00
 PCUpdate





Control multiciclo: jal

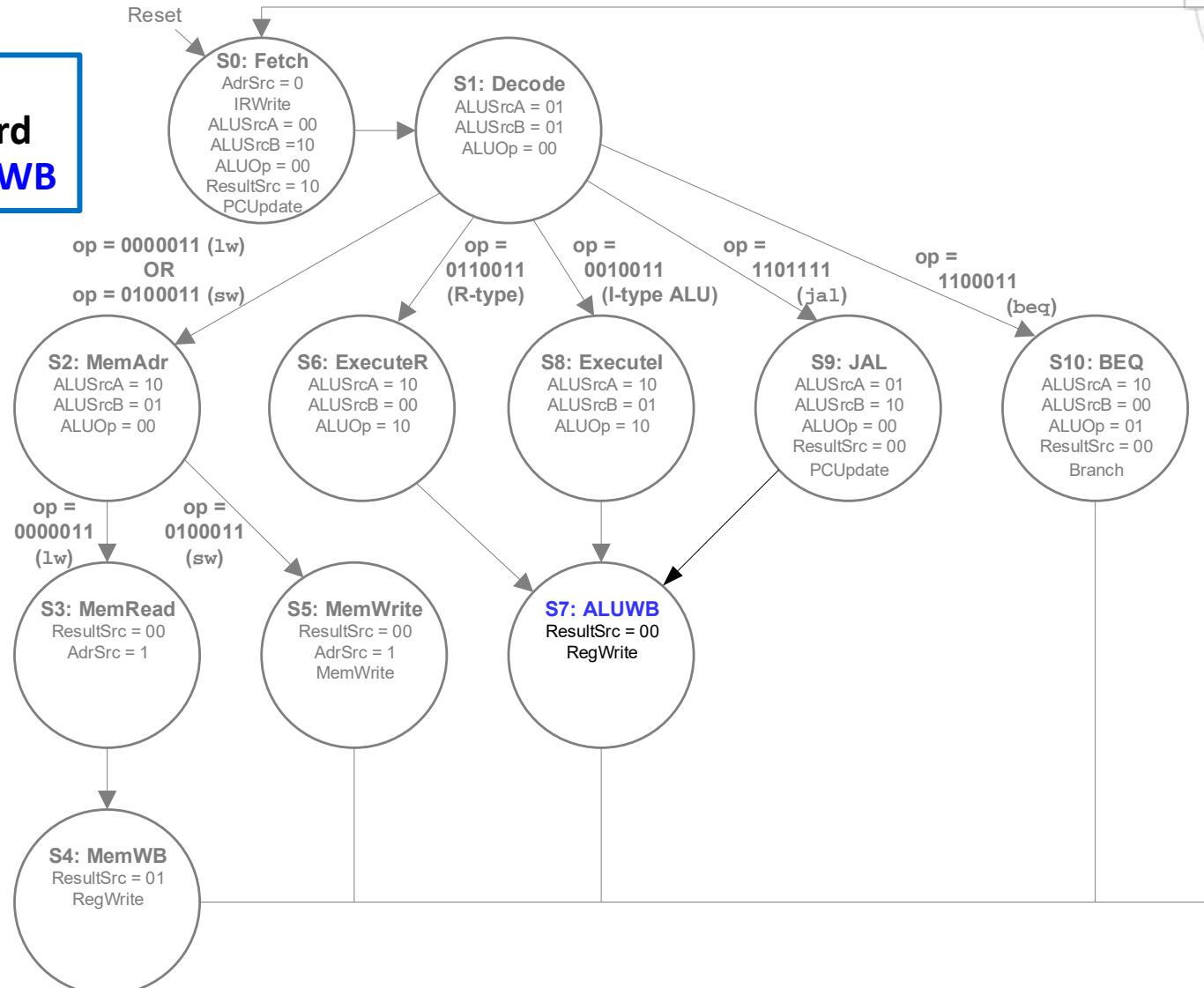
versión 2021

tema 6:
Diseño del procesador

FC

89

**PC + 4 se
escribe en rd
en S7: ALUWB**





Control multiciclo

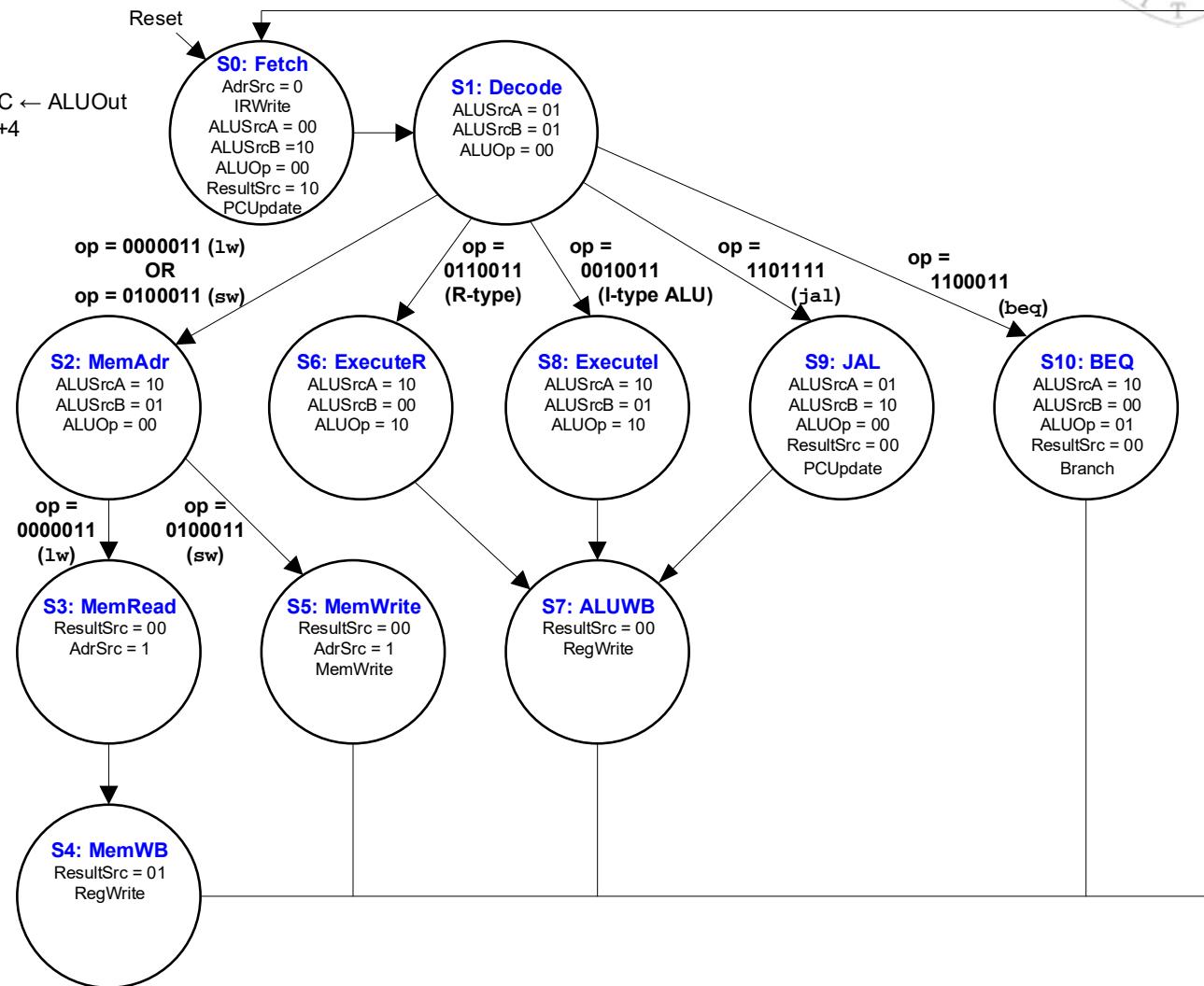
versión 2021

tema 6:
Diseño del procesador

FC

90

State	Datapath μOp
Fetch	Instr ← Mem[PC]; PC ← PC+4
Decode	ALUOut ← PCTarget
MemAdr	ALUOut ← rs1 + imm
MemRead	Data ← Mem[ALUOut]
MemWB	rd ← Data
MemWrite	Mem[ALUOut] ← rd
ExecuteR	ALUOut ← rs1 op rs2
Executel	ALUOut ← rs1 op imm
ALUWB	rd ← ALUOut
BEQ	ALUResult = rs1-rs2; if Zero, PC ← ALUOut
JAL	PC ← ALUOut; ALUOut ← PC+4

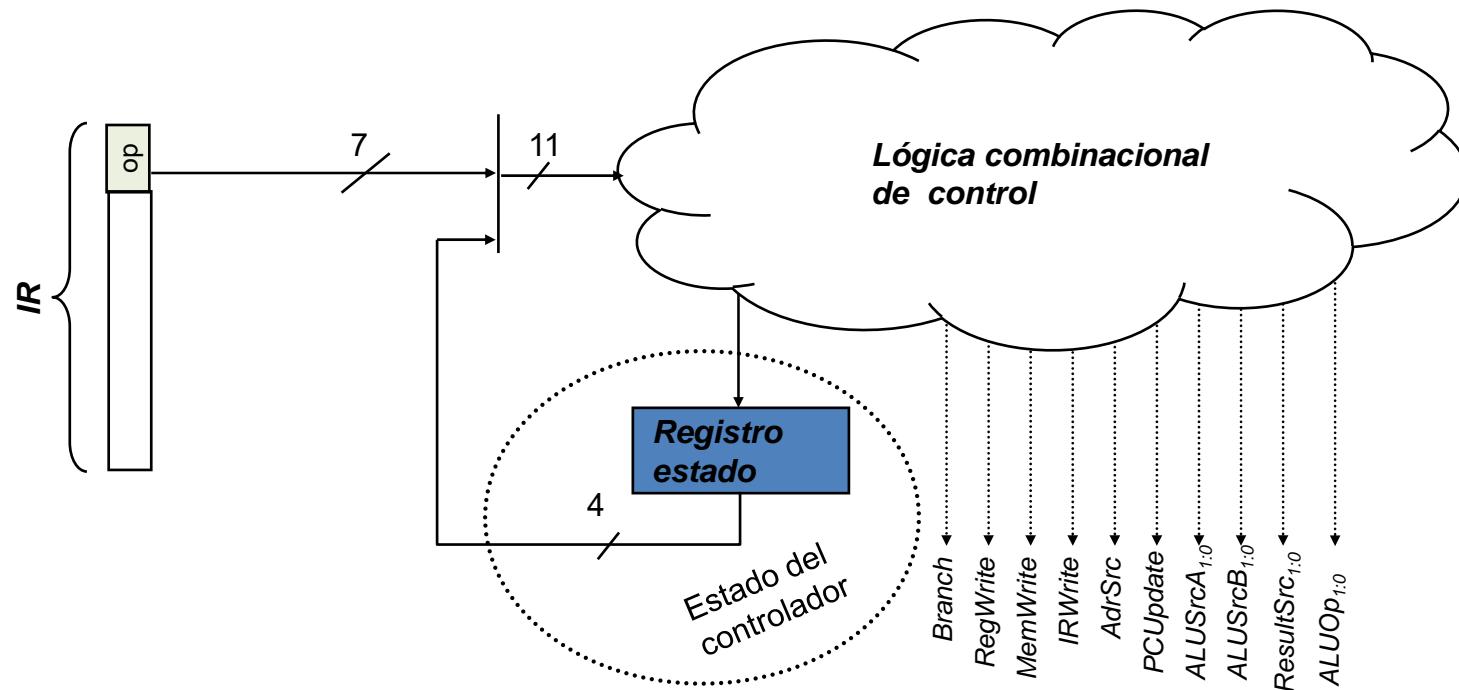




Diseño del controlador multiciclo

Sistemas combinacionales

versión 2021



Diseño del controlador multiciclo



Estado actual	op	Instrucción	Estado siguiente	Branch	RegWrite	MemWrite	IRWrite	PCUpdate	AdrSrc	ALUSrcA _{1:0}	ALUSrcB _{1:0}	ResultSrc _{1:0}	ALUOp _{1:0}	
0000	X		0001	0	0	0	1	1	0	00	10	10	00	
0001	0000011	lw	0010	0	0	0	0	0	X	01	01	X	00	
0001	0100011	sw	0010	0	0	0	0	0	X	01	01	X	00	
0001	0110011	Tipo R	0110	0	0	0	0	0	X	01	01	X	00	
0001	0010011	Tipo I ALU	1000	0	0	0	0	0	X	01	01	X	00	
0001	1101111	jal	1001	0	0	0	0	0	X	01	01	X	00	
0001	1100011	beq	1010	0	0	0	0	0	X	01	01	X	00	
0010	0000011	lw	0011	0	0	0	0	0	X	10	01	X	00	
0010	0100011	sw	0101	0	0	0	0	0	X	10	01	X	00	
0011	X		0100	0	0	0	0	0	1	X	X	00	X	
0100	X		0000	0	1	0	0	0	X	X	X	01	X	
0101	X		0000	0	0	1	0	0	1	X	X	00	X	
0110	X		0111	0	0	0	0	0	X	10	00	X	10	
0111	X		0000	0	0	0	1	0	X	X	X	00	X	
1000	X		0111	0	0	0	0	0	X	10	01	X	10	
1001	X		0011	0	0	0	0	1	X	01	10	00	00	
1010	X		0000	1	0	0	0	0	X	10	00	00	01	



Diseño del controlador multiciclo

- 1 ROM
 - 11 bits de dirección (2^{11} palabras)
 - Palabras de 18 bits
- 2 ROM
 - ROM de control:
 - 4 bits de dirección (2^4 palabras)
 - palabra de 14 bits
 - ROM de siguiente estado:
 - 11 bits de dirección (2^7 palabras)
 - palabras de 4 bits



Rendimiento multiciclo

- Las instrucciones se ejecutan en diferente número de ciclos:
 - 3 ciclos : beq
 - 4 ciclos : Procesamiento de datos, sw
 - 5 ciclos : lw
- CPI se calcula con el porcentaje medio de ejecución de cada instrucción y el número de ciclos
- SPECINT2000 benchmark:
 - 25% loads
 - 10% stores
 - 13% branches
 - 52% procesamiento de datos
- **CPI medio**= $(0.13)(3) + (0.52 + 0.10)(4) + (0.25)(5) = 4.12$



Rendimiento multiciclo

- Camino crítico multiciclo:
- Asumiendo que:
 - El Banco de Registros es más rápido que la memoria
 - Escribir en memoria es más rápido que leer de memoria
- $T_{c_multi} = t_{pcq} + t_{dec} + 2t_{mux} + \max(t_{ALU}, t_{mem}) + t_{setup}$



Rendimiento multiciclo

Elemento	Parámetro	Retardo (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexor	t_{mux}	30
Puertas AND-OR	$t_{\text{AND-OR}}$	20
ALU	t_{ALU}	120
Decodificador	t_{dec}	25
Unidad de extensión	t_{ext}	35
Read memoria	t_{mem}	200
Read Banco de registros	$t_{BR\text{read}}$	100
Setup Banco de registros	$t_{BR\text{setup}}$	60

$$\begin{aligned} T_{c_multi} &= t_{pcq} + t_{\text{dec}} + 2t_{\text{mux}} + \max(t_{\text{ALU}}, t_{\text{mem}}) + t_{\text{setup}} = \\ &= 40 + 25 + 60 + 200 + 60 = 385 \text{ ns} \end{aligned}$$



Rendimiento multiciclo

- Para un programa con 100 mil millones de instrucciones ejecutándose en un ARM multiciclo
 - **CPI** = 4.12 ciclos/instrucción
 - **Tiempo de ciclo:** $T_{c2} = 385 \text{ ps}$

$$\begin{aligned}\text{Tiempo de ejecución} &= (\text{N instrucciones}) \times \text{CPI} \times T_c \\ &= (100 \times 10^9)(4.12)(385 \times 10^{-12}) \\ &= 158,6 \text{ segundos}\end{aligned}$$

- Esto es más lento que el procesador monociclo (80.5 seg.)