CE3201 – Taller de Diseño Digital

Diseño de la organización de un procesador: Caso ARMv4

PROFESOR: ING. LUIS BARBOZA ARTAVIA

Agenda

- Arquitectura ARMv4
- Organización: procesador uniciclo

ARM

- Arquitectura RISC (Reduced Instruction Set Computer):
- Conjunto de instrucciones reducido.
- Simple y eficiente a nivel de hardware.
- Instrucciones del mismo tamaño.
- Mayor cantidad de registros.

Registros ARMv4

Name	Use
R0	Argument / return value / temporary variable
R1–R3	Argument / temporary variables
R4–R11	Saved variables
R12	Temporary variable
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

Registros ARMv4

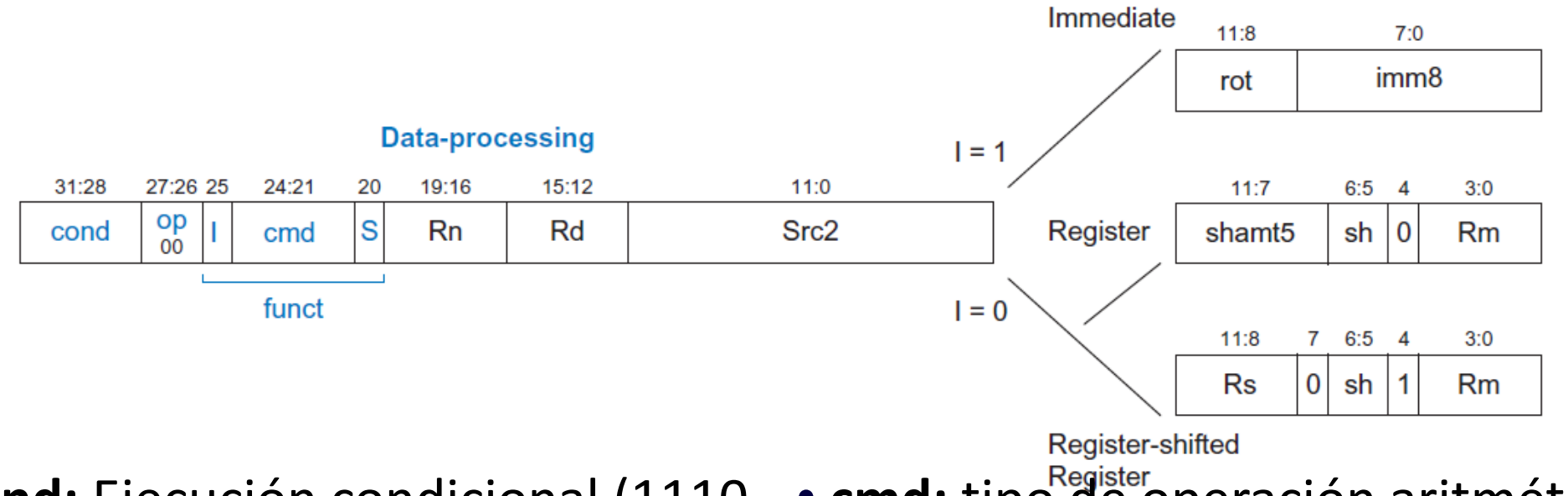
- Instrucciones pueden usar 1, 2 o 3 registros. Los registros pueden ser leídos o escritos.
- `ADD R1, R1, R2`
- `STR R1, [R2, #4]`
- `MOV R0, #1000`

Tipos de instrucciones

- Procesamiento de datos (operaciones lógico-aritméticas)
- Memoria (cargas y escrituras desde y hacia memoria).
- Saltos (cambiar el flujo de ejecución de un programa).

Cada instrucción tiene un formato (encodificación) diferente).

Tipo: Procesamiento de datos



- **cond:** Ejecución condicional (1110 por defecto).
- **op:** Tipo instrucción (00 procesamiento datos)
- **I:** inmediato
- **cmd:** tipo de operación aritmético-lógica.
- **Rn:** registro operando 1.
- **Rm:** registro operando 2.
- **S:** banderas de condición

Tipo: Procesamiento de datos

Assembly Code

```
ADD R5, R6, R7
    (0xE0865007)
SUB R8, R9, R10
    (0xE049800A)
```

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
1110_2	00_2	0	0010_2	0	9	8	0	0	0	10
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm

Tipo: Procesamiento de datos

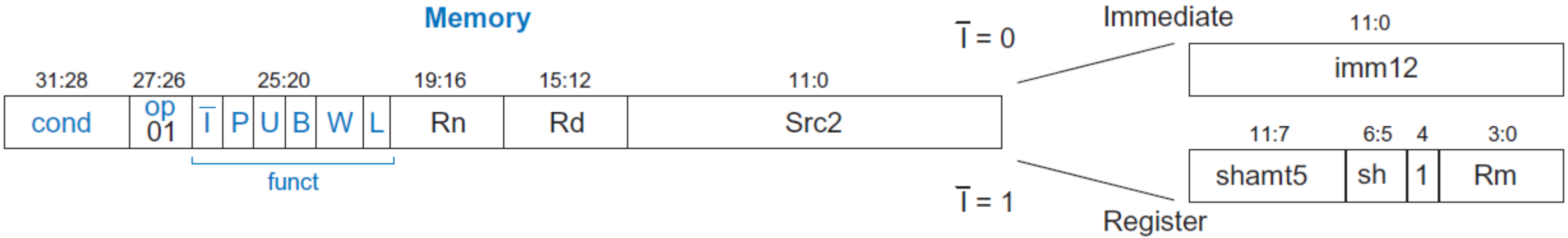
Assembly Code

```
ADD R0, R1, #42
    (0xE281002A)
SUB R2, R3, #0xFF0
    (0xE2432EFF)
```

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
1110_2	00_2	1	0010_2	0	3	2	14	255
cond	op	I	cmd	S	Rn	Rd	rot	imm8

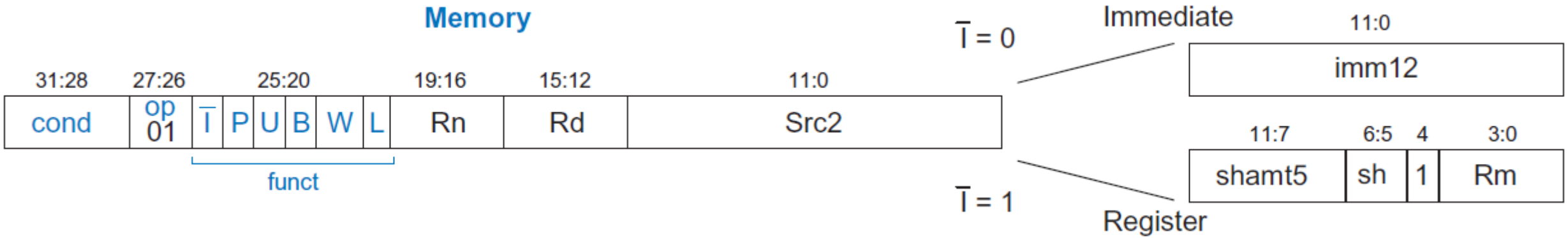
Tipo: Memoria



P	W	Index Mode
0	0	Post-index
0	1	Not supported
1	0	Offset
1	1	Pre-index

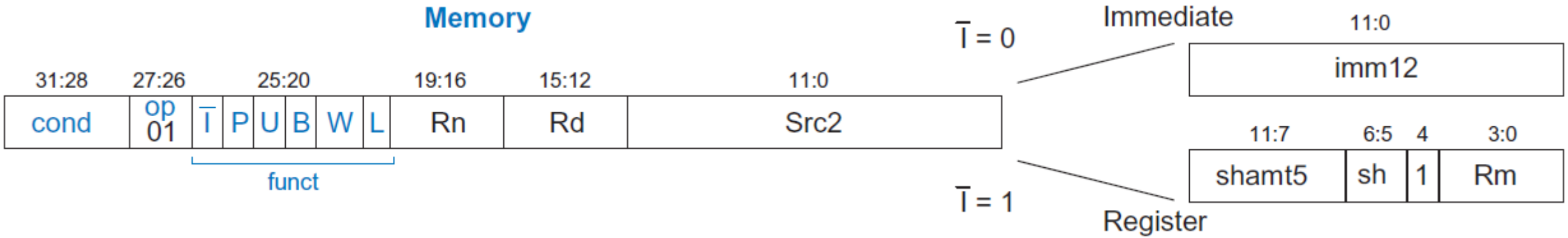
Mode	ARM Assembly	Address	Base Register
Offset	`LDR R0, [R1, R2]`	R1 + R2	Unchanged
Pre-index	`LDR R0, [R1, R2]!`	R1 + R2	R1 = R1 + R2
Post-index	`LDR R0, [R1], R2`	R1	R1 = R1 + R2

Tipo: Memoria



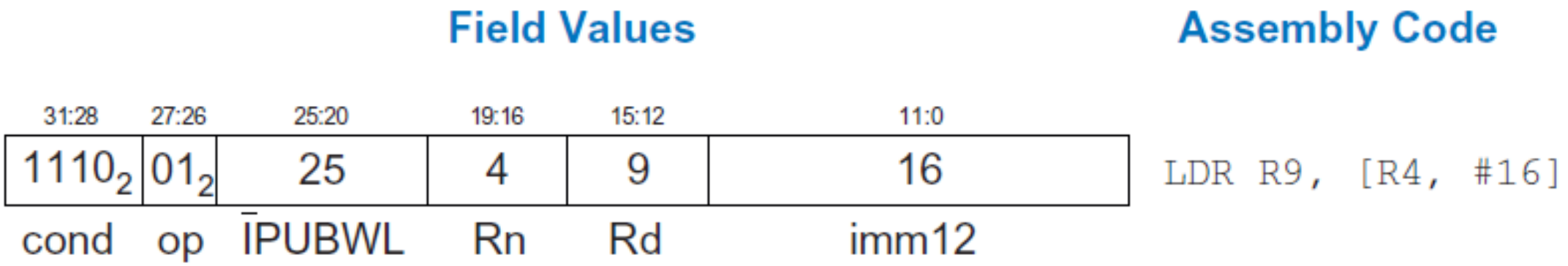
L	B	Instruction
0	0	STR
0	1	STRB
1	0	LDR
1	1	LDRB

Tipo: Memoria

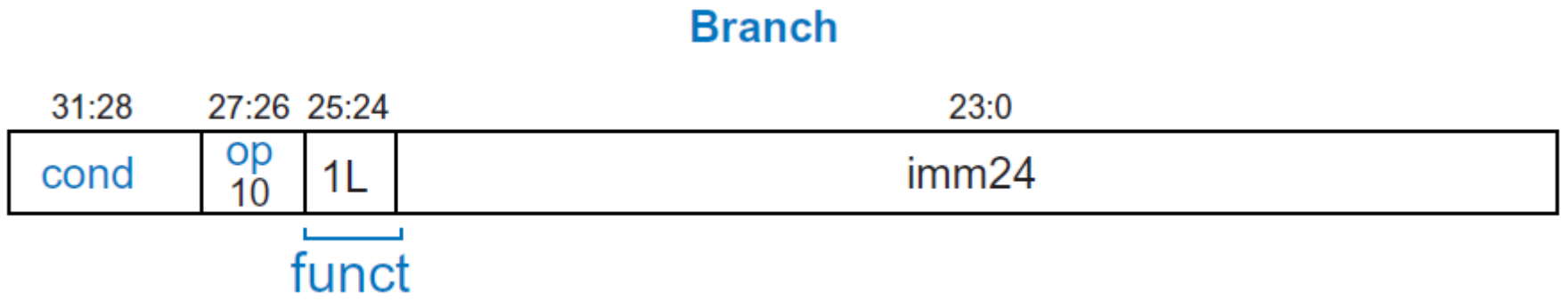


Bit	Meaning Ī	Meaning U
0	Immediate offset in Src2	Subtract offset from base
1	Register offset in Src2	Add offset to base

Tipo: Memoria

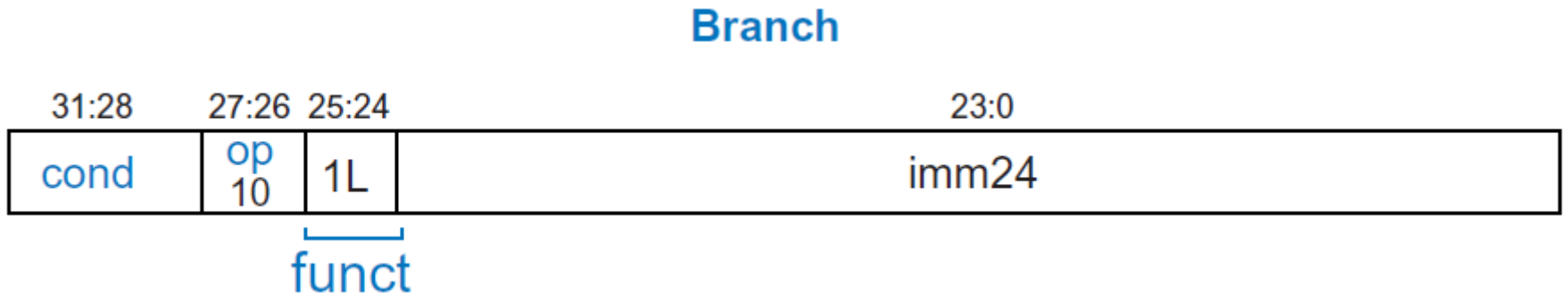


Tipo: Saltos



L	Name	Description	Operation
0	B label	Branch	PC ← (PC+8)+imm24 << 2
1	BL label	Branch with Link	LR ← (PC+8) - 4; PC ← (PC+8)+imm24 << 2

Tipo: Saltos



- **imm24:** cantidad de palabras después de PC+8 para el salto. El corrimiento se realiza para convertir cantidad de palabras a bytes.

Tipo: Saltos

ARM Assembly Code

```
0x80A0          BLT THERE
0x80A4          ADD R0, R1, R2
0x80A8          SUB R0, R0, R9
0x80AC          ADD SP, SP, #8
0x80B0          MOV PC, LR
0x80B4 THERE    SUB R0, R0, #1
0x80B8          ADD R3, R3, #0x5
```

Ciclo de ejecución de instrucciones

Cualquier procesador moderno sigue al menos las siguientes etapas:

1. Búsqueda de instrucción (fetch).
2. Decodificación / Lectura operandos.
3. Ejecución de instrucción.
4. Escritura en registros (cuando aplique).
5. Escritura en memoria (cuando aplique).

Organización de alto nivel

Cualquier procesador puede ser visto internamente como la unión de dos partes:

1. Ruta de datos (Datapath).
2. Unidad de control.

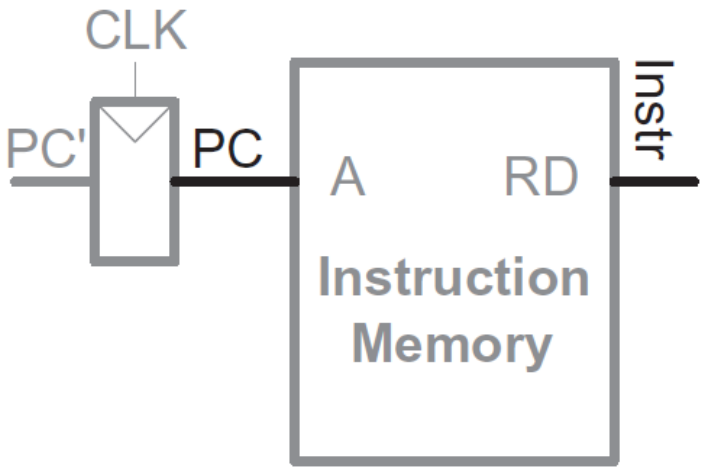
Bloques básicos

La ruta de datos se compone a su vez de varios elementos básicos de procesamiento y almacenamiento:

1. Banco de registros.
2. Unidad lógico-aritmética (ALU).
3. Registro de contador de programa (PC).
4. Multiplexores.
5. Otros.

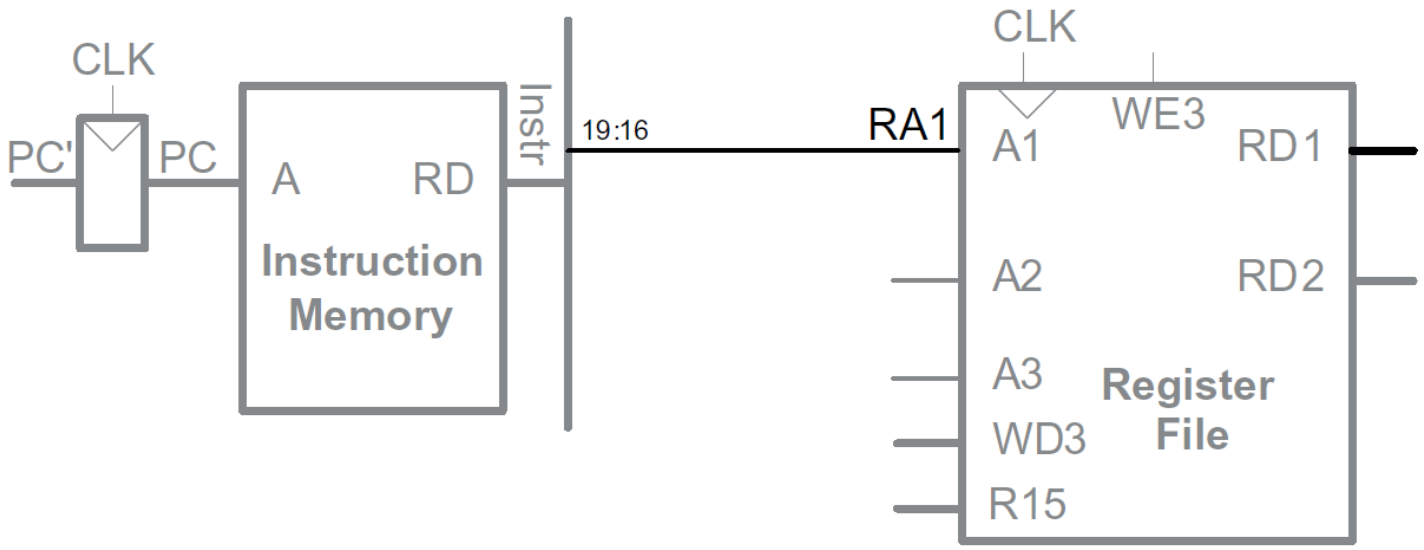
Búsqueda de instrucción

La búsqueda de la instrucción se realiza por medio del contador de programa (PC). La salida de este registro se utiliza para direccionar la memoria de instrucciones, que contiene todo el programa a ejecutar.



Lectura de operandos / Deco

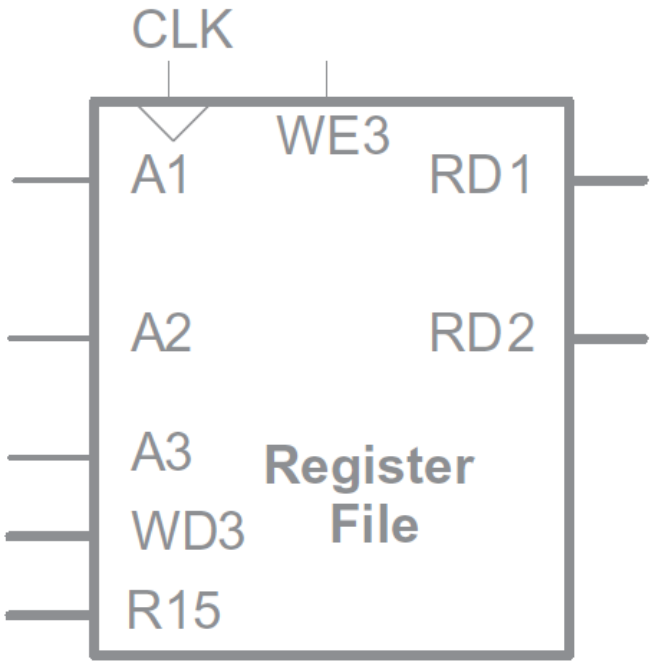
- Con la instrucción, se extraen los operandos a leer por medio del banco de registros.
- **Ejemplo LDR:** operando Rn se extrae de $Instr_{19:16}$



Banco de registros

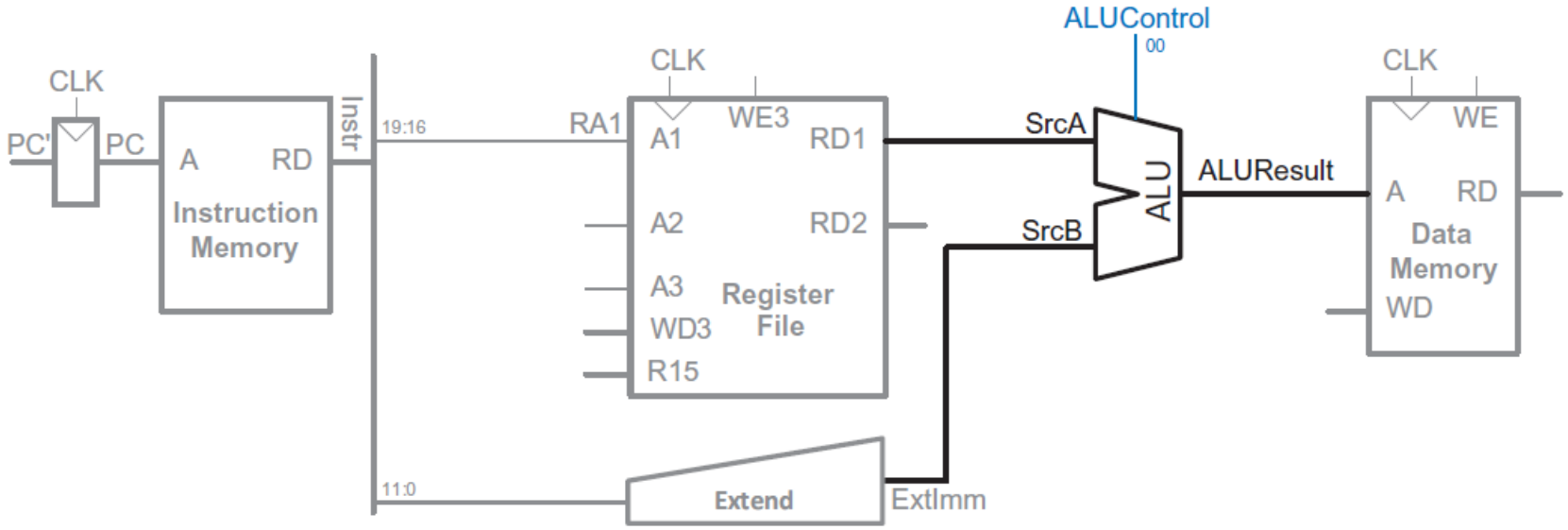
Las señales del banco de registros son:

- **A1, A2:** dirección de registros fuentes.
- **A3:** dirección de registro destino.
- **WD3:** datos de escritura en registro destino.
- **R15:** escritura a registro PC.
- **RD1, RD2:** datos de registros fuente.
- **WE3:** enable de escritura a registro destino.



Ejecución

- Se realiza la operación aritmético-lógica definida por la instrucción.
- Para instrucciones de memoria o saltos, se calcula la dirección.

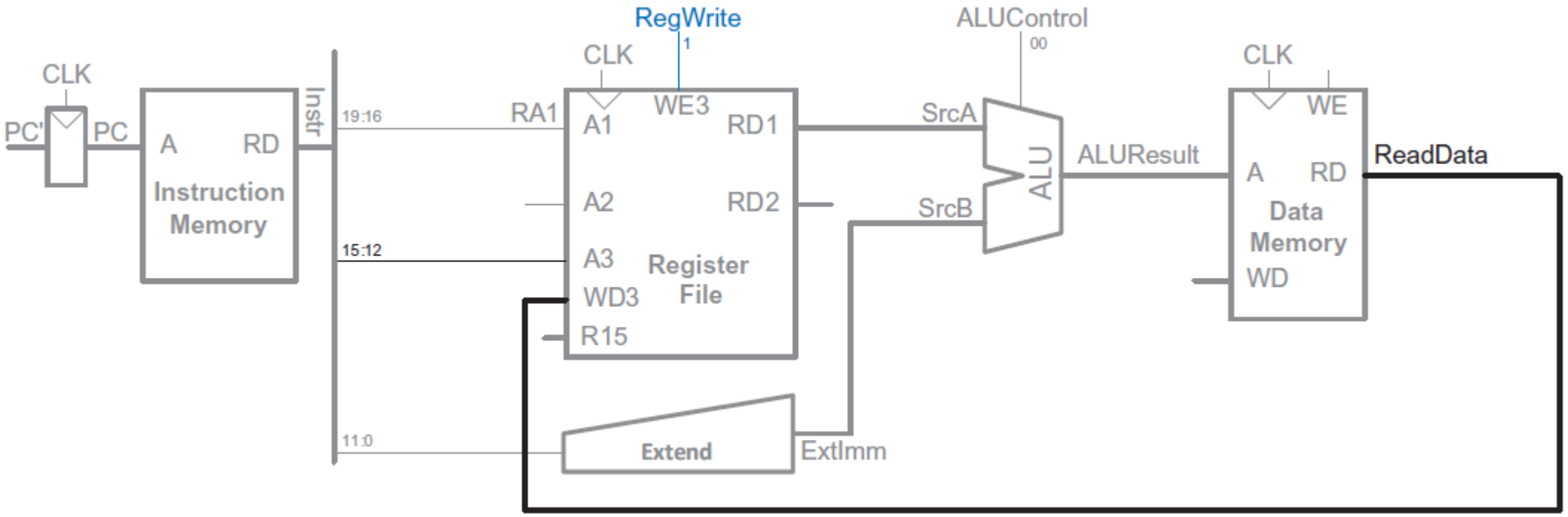


Acceso a memoria

- Se realiza direccionando con el resultado de la ALU (cálculo de dirección anterior).

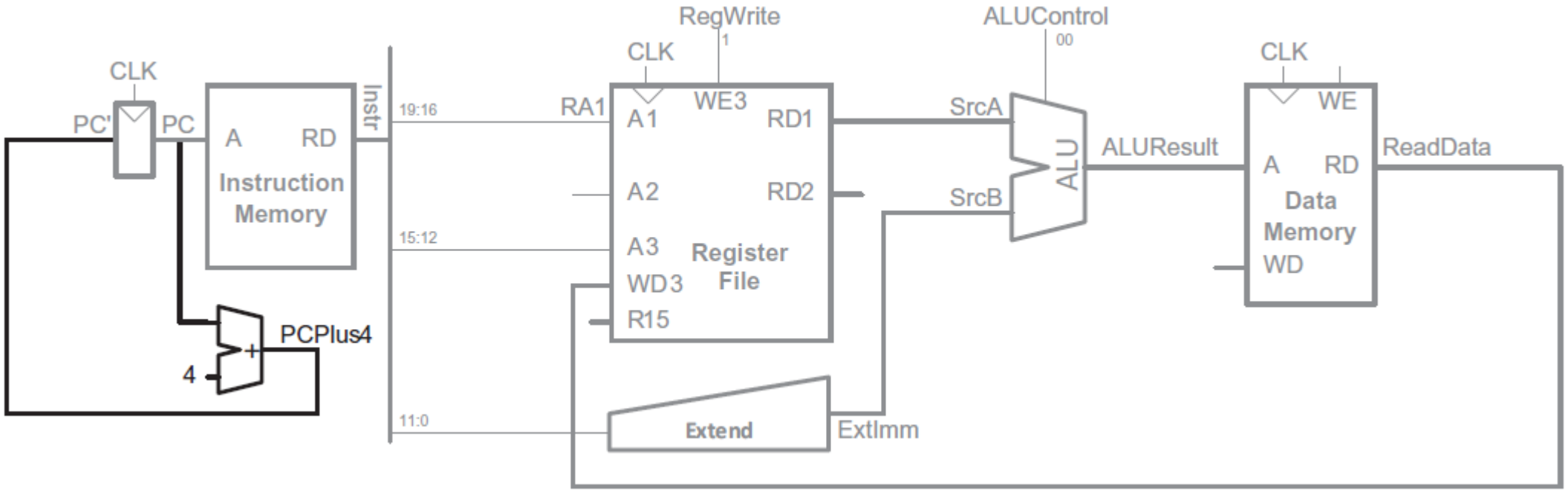
Escritura de registros - WB

- Se realiza al final (flanco negativo de reloj).



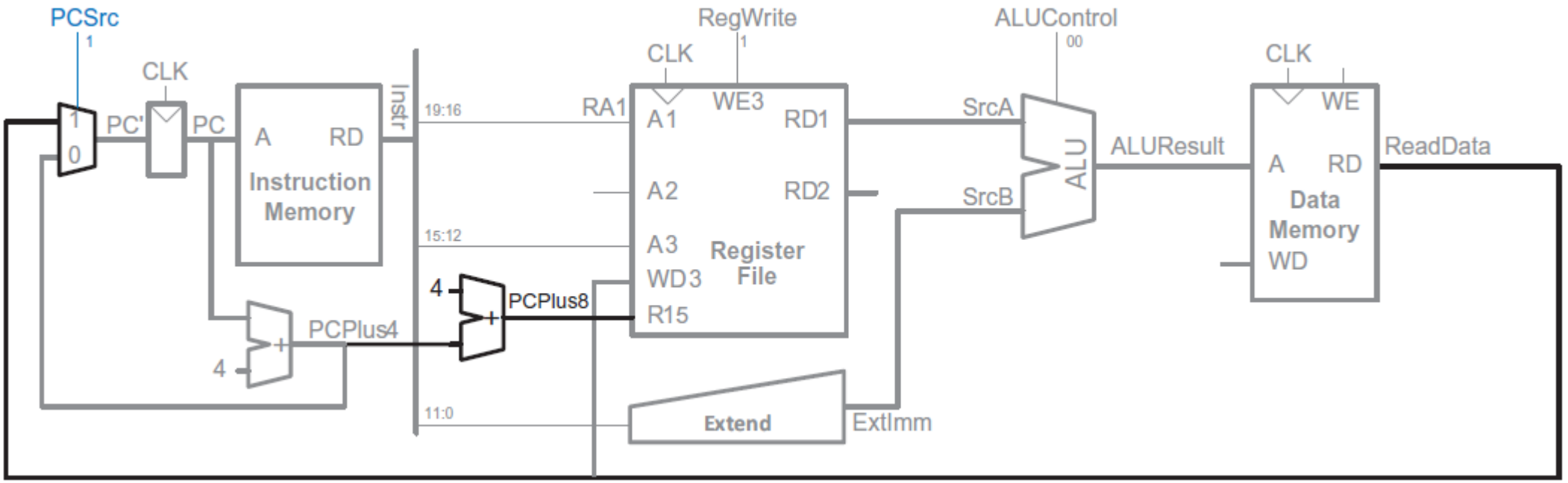
Actualización PC

- Se debe ajustar el PC (flanco positivo) para realizar la búsqueda de la siguiente instrucción.



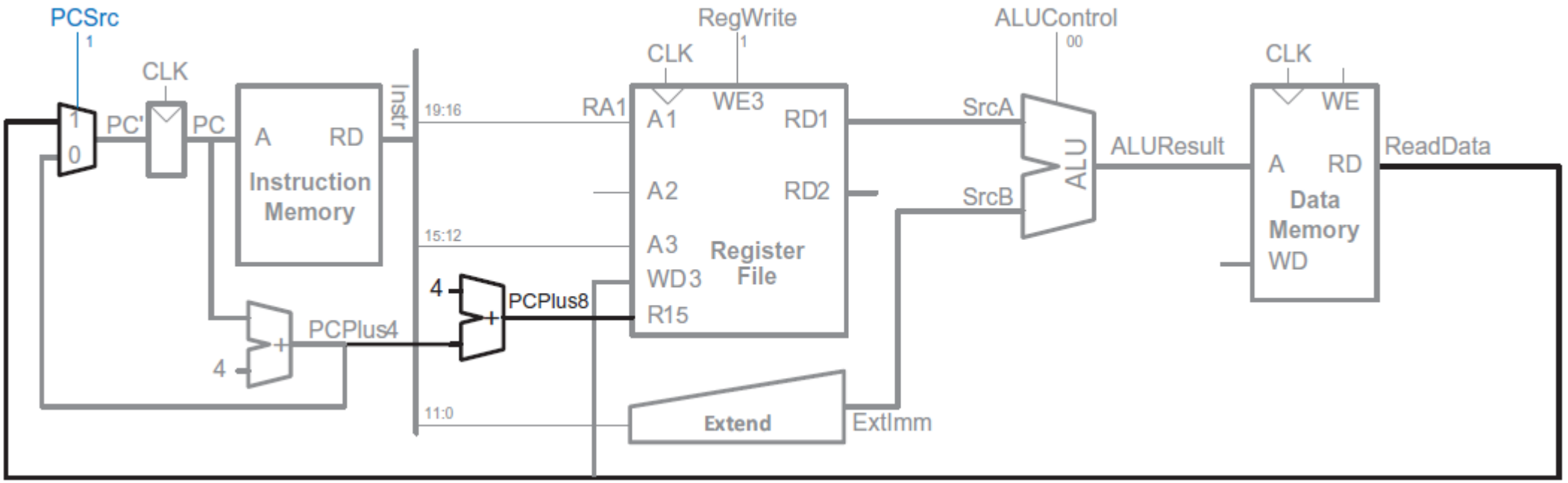
Actualización R15

- Caso especial donde se debe almacenar PC+8.



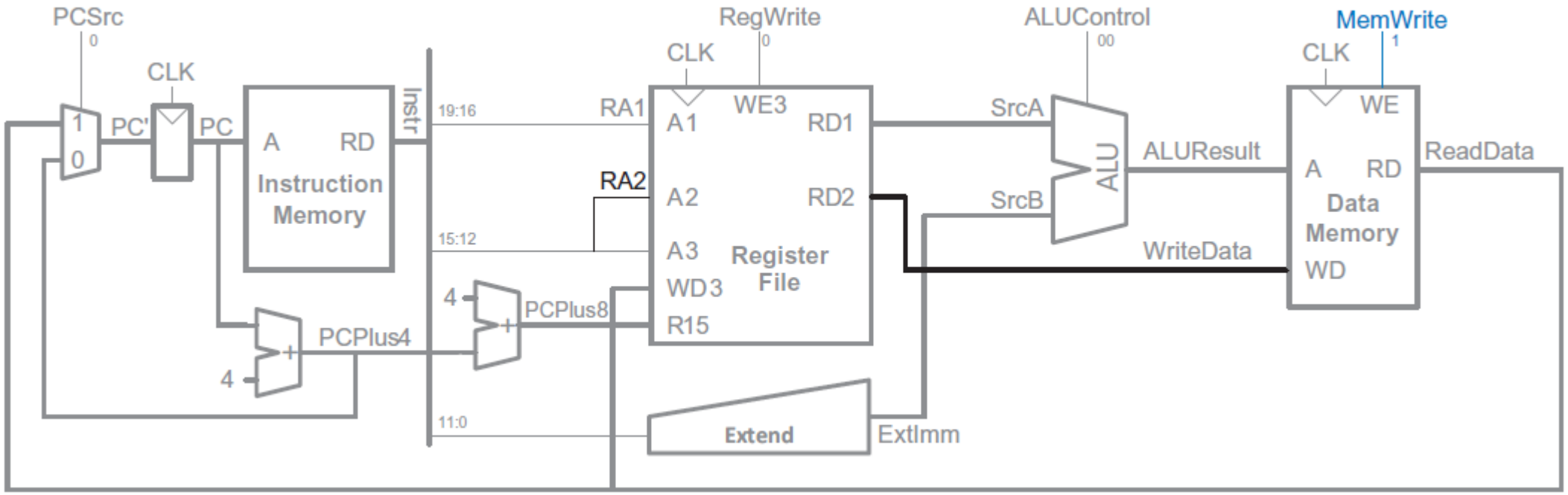
Añadiendo STR

- ¿Qué modificaciones deben realizarse para soportar STR?



Añadiendo STR

- ¿Qué modificaciones deben realizarse para soportar STR?

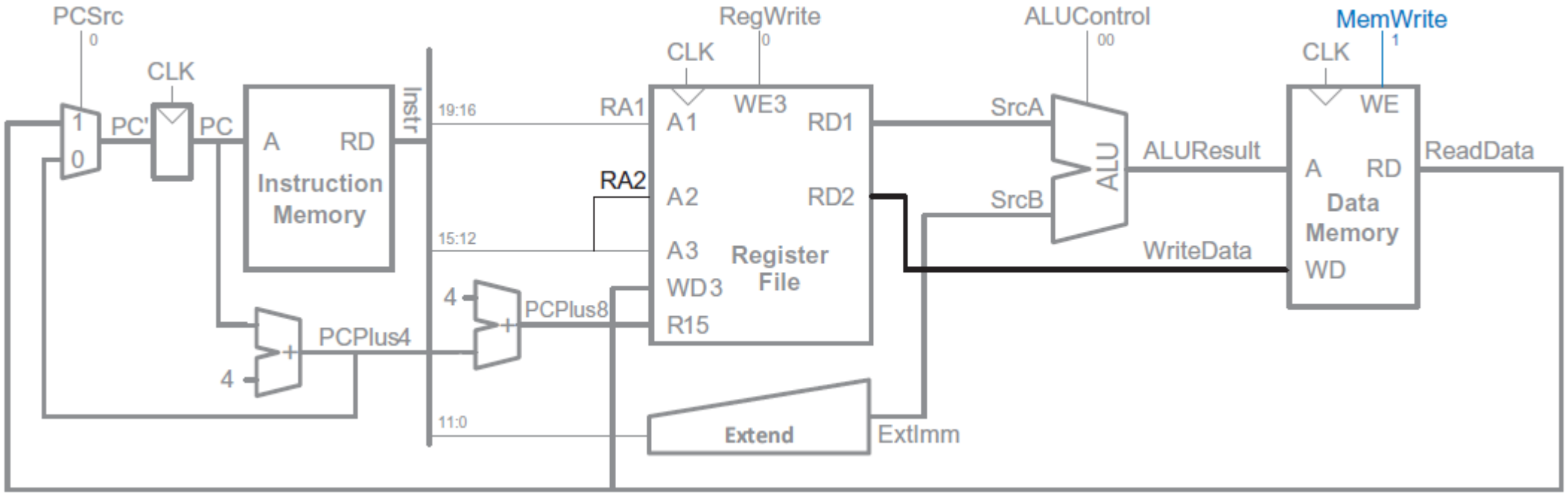


Instrucciones procesamiento con inmediatos

- Instrucciones de procesamiento de datos usan la ALU para realizar operaciones.
- La entrada de control selecciona la operación a realizar.
- La salida debe ser escrita al banco de registros en la posición A3.

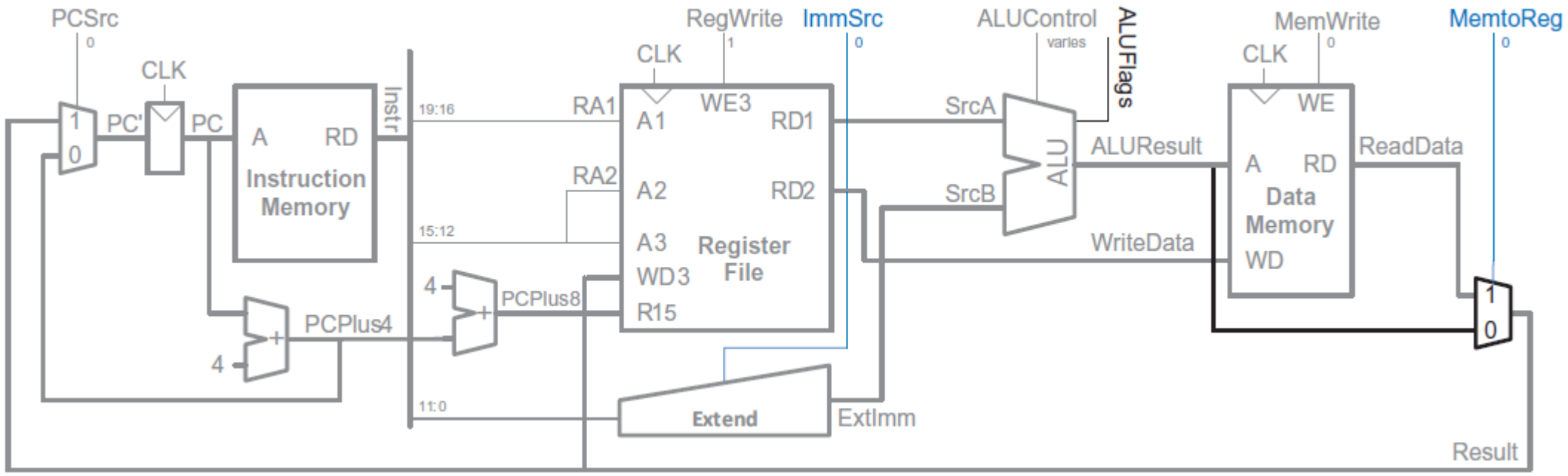
Instrucciones procesamiento con inmediatos

- ¿Qué modificaciones deben realizarse para soportar instrucciones como ADD R0, R1, #90?



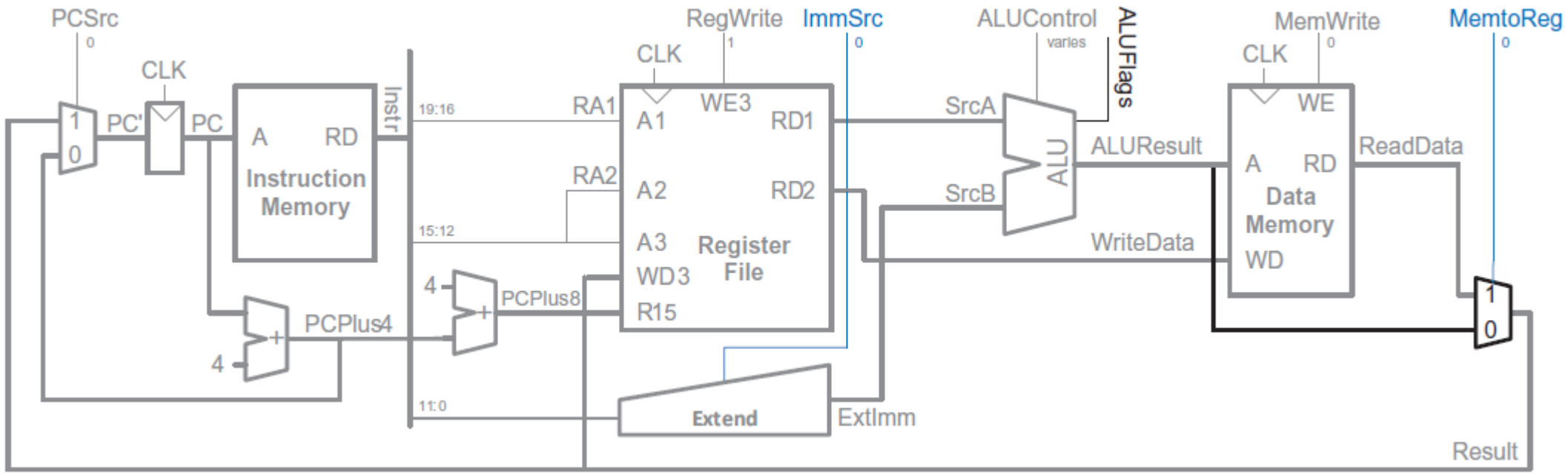
Instrucciones procesamiento con inmediatos

- ¿Qué modificaciones deben realizarse para soportar instrucciones como ADD R0, R1, #90?



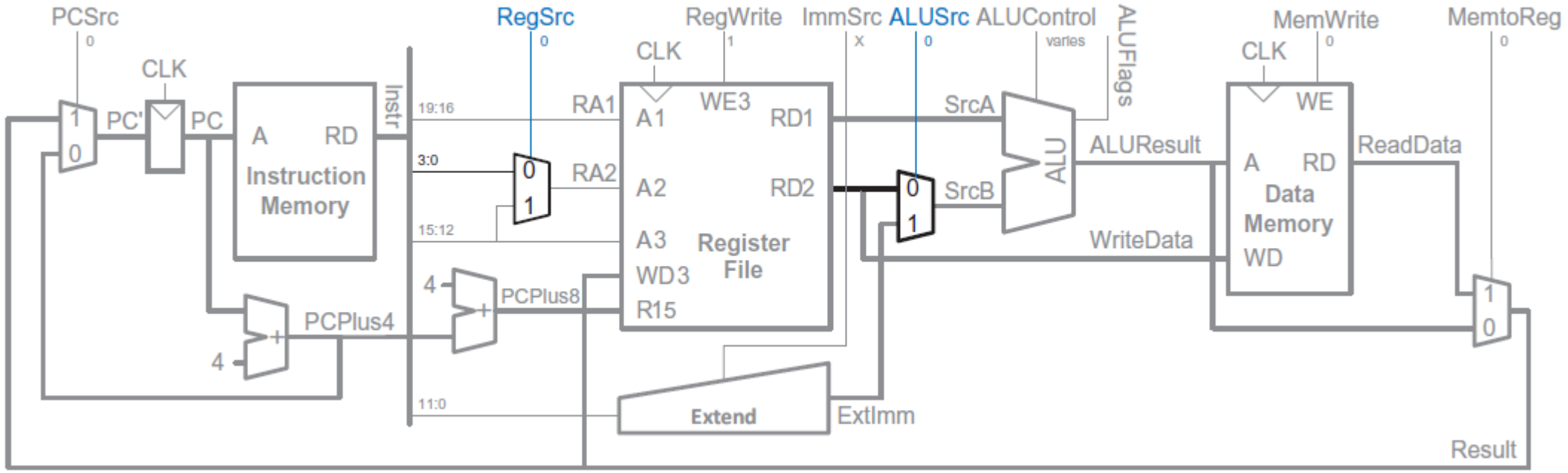
Instrucciones procesamiento con registros

- ¿Qué modificaciones deben realizarse para soportar instrucciones como ADD R0, R1, R2?



Instrucciones procesamiento con registros

- ¿Qué modificaciones deben realizarse para soportar instrucciones como ADD R0, R1, R2?

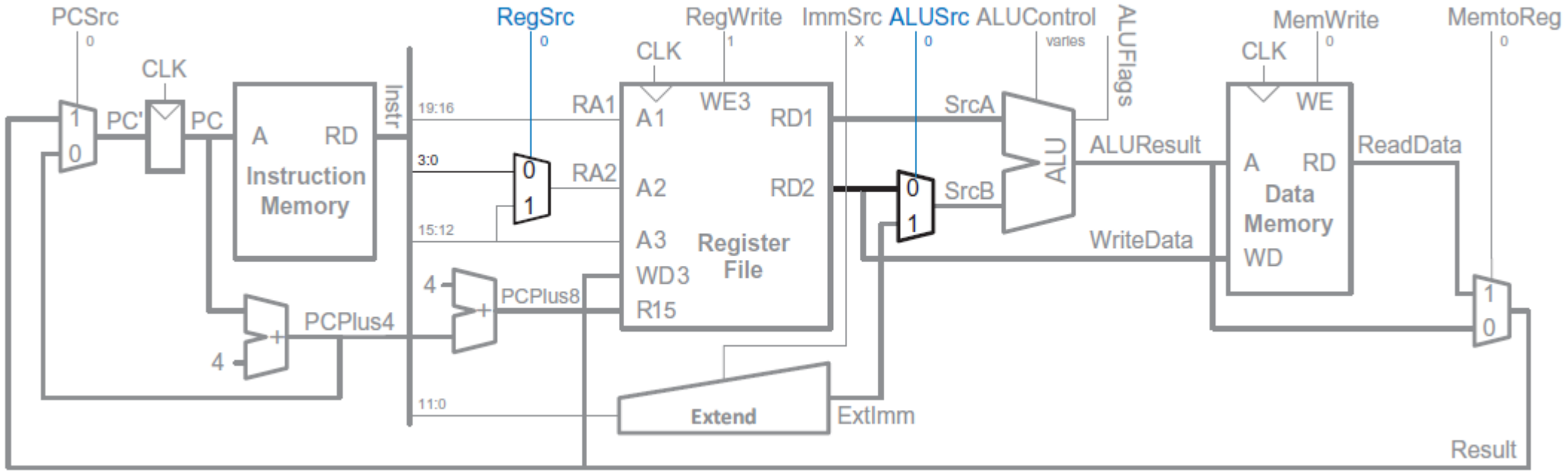


Instrucciones con saltos

- ARM utiliza el valor de PC+8, lo suma con un inmediato de 24 bits y escribe el resultado de vuelta a PC.
- El inmediato se multiplica por 4 y se extiende el signo, por lo que el módulo de extensión debe modificarse.

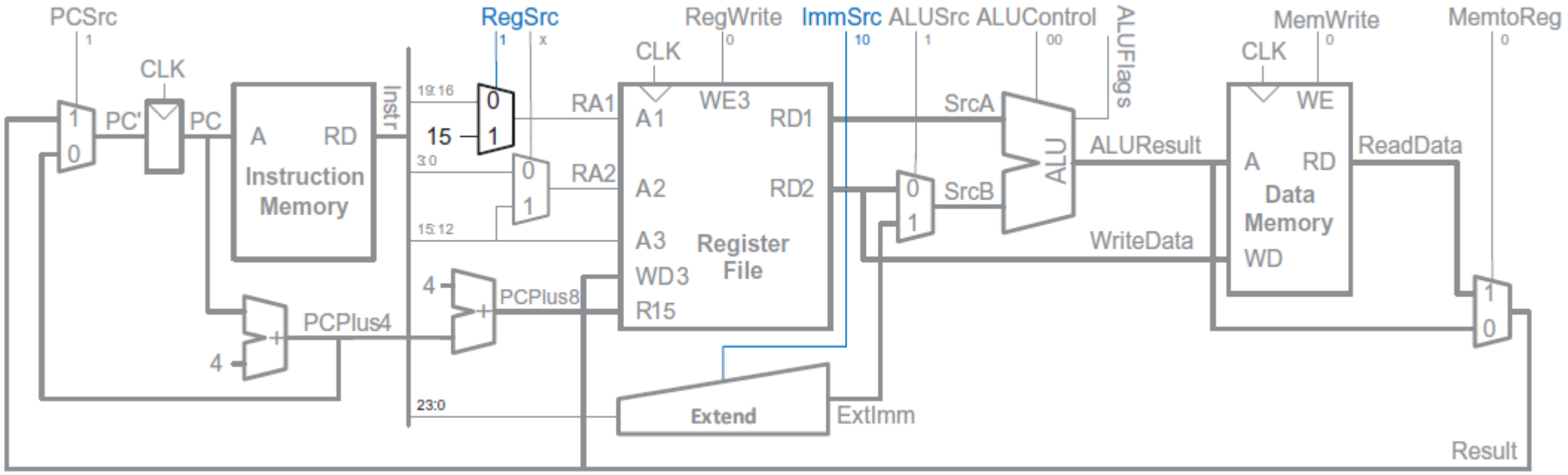
Instrucciones con saltos

- ¿Qué modificaciones deben realizarse para soportar instrucciones con saltos?



Instrucciones con saltos

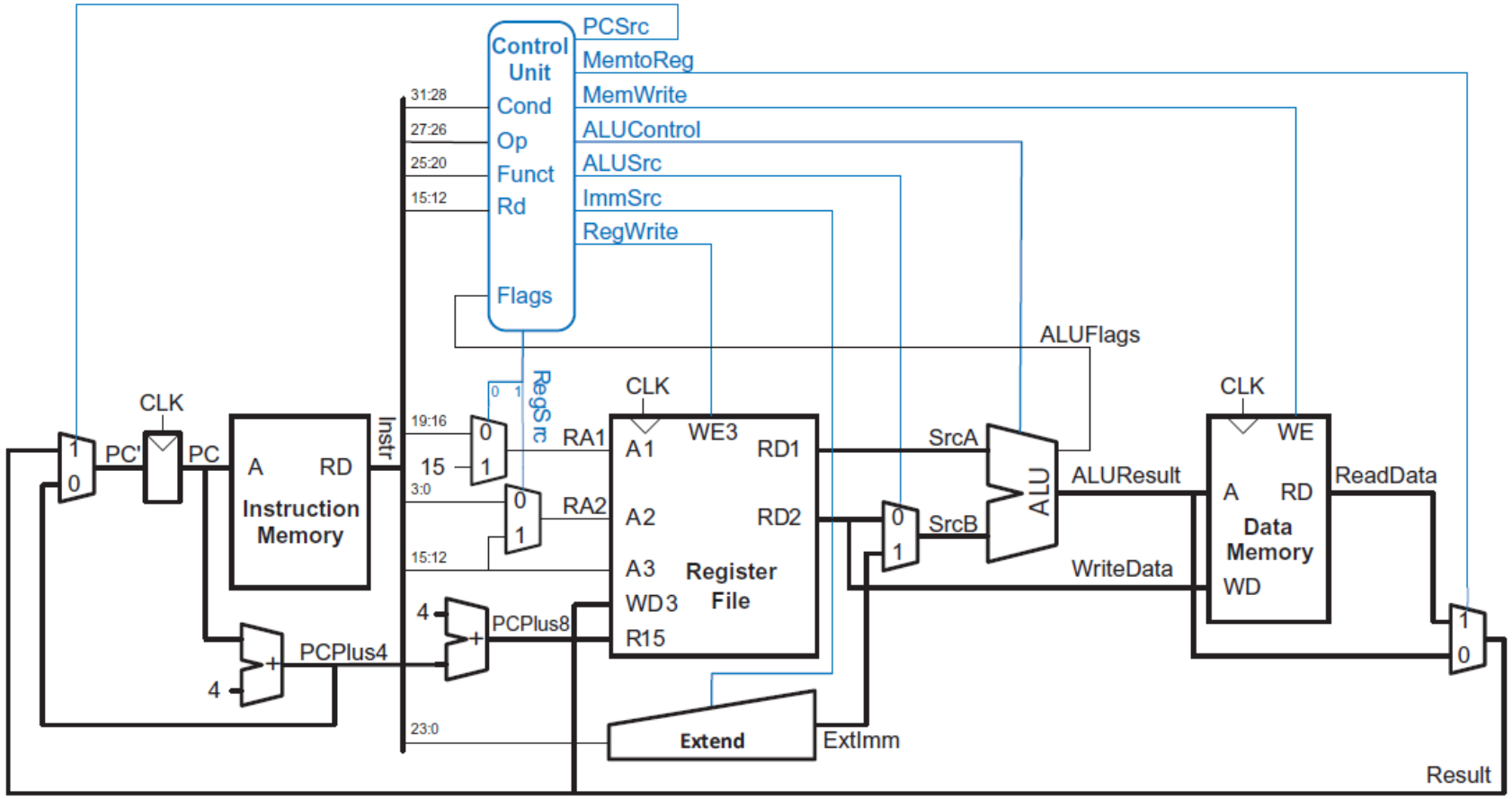
- ¿Qué modificaciones deben realizarse para soportar instrucciones con saltos?



Unidad de control

- La unidad de control debe generar las señales de control requeridas por la ruta de datos. Las entradas son:
- cond: ejecución condicional ($Instr_{31:28}$).
- op: tipo de instrucción ($Instr_{27:26}$).
- funct: identificador de instrucción ($Instr_{25:20}$).
- Rd: en caso de escrituras en el registro ($Instr_{15:12}$).

Unidad de control



Señales de control - MemtoReg

- Señal de escritura a registro desde memoria (load).
- 0: caso no LDR
- 1: caso LDR ($op = 01$ y $funct_0 = 1$).

Señales de control – MemWrite*

- Señal de escritura de memoria (store).
- 0: caso no STR
- 1: caso STR ($op = 01$ y $funct_0 = 0$).
- * Afectado por ejecución condicional.

Señales de control – ALUControl

- Selecciona el tipo de operación en la ALU.
- Depende de cmd ($funct_{4:1}$).
- 0000: AND, ALUControl - 10
- 0010: SUB, ALUControl – 01
- 0100: ADD, ALUControl – 00
- 1100: ORR, ALUControl – 11

Señales de control – ALUSrc

- Señal de selección de operando B.
- 0: caso R-R ($op = 00$ y $funct_5 = 0$ [bit I]).
- 1: caso R-I.

Señales de control – ImmSrc

- Señal de selección de tipo de inmediato.
- 00: inmediato de proc. De datos, 8 bits ($op = 00$).
- 01: inmediato de memoria, 12 bits ($op = 01$).
- 10: inmediato de salto, 24 bits ($op = 10$).

Señales de control – RegSrc

- Señal de dos bits de selección de muxes de operandos.
- 00: caso proc. datos ($op = 00$).
- X1: caso Branch ($op = 10$).
- 10: caso STR ($op = 01$ y $funct_0 = 0$)

Señales de control – RegWrite*

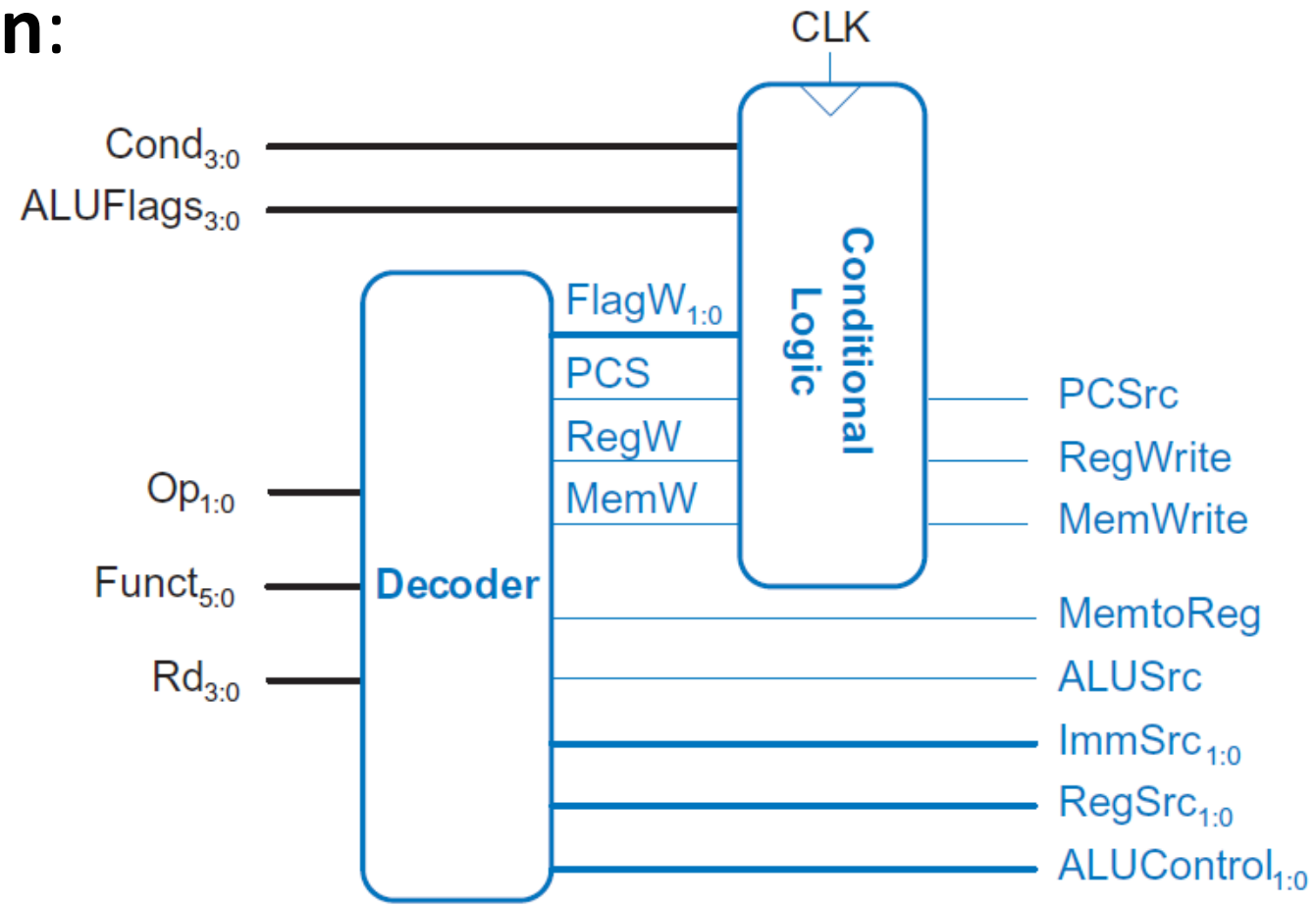
- Señal de escritura al banco de registros.
- 0: caso no escritura a registros.
- 1: caso a registros por proc. o memoria ($op = 00$ o $op = 01$ y $funct_0 = 1$).
- * Afectado por ejecución condicional.

Señales de control – PCSrc*

- Señal fuente de PC, saltos y escrituras a PC.
- 0: caso no saltos (ni escrituras a PC).
- 1: caso salto ($op = 10$) o escritura a PC ($rd = 15$ y RegWrite).
- *Afectado por ejecución condicional.

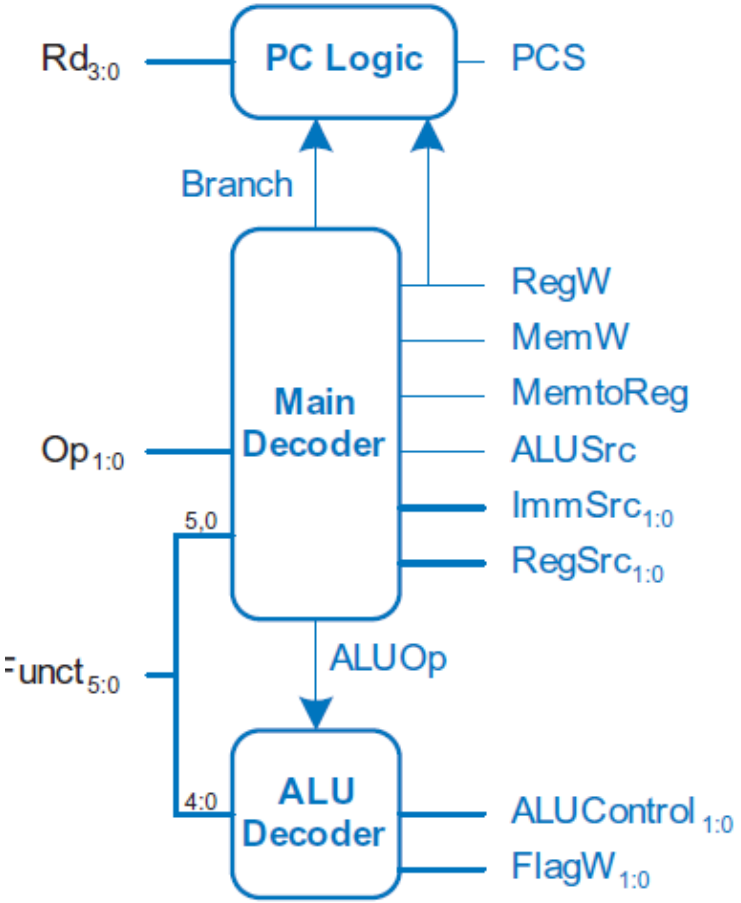
Unidad de control

- **Módulo de decodificación**: genera la mayoría de señales de control.
- **Módulo de ejecución condicional**: señales que dependen de las condicionales.



Unidad de decodificación

- Genera la mayoría de señales de control.
- Se divide en tres módulos.



Decodificador principal

- Implementa la lógica de las señales de control.

Op	$Funct_5$	$Funct_0$	Type	Branch	MemtoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1
00	1	X	DP Imm	0	0	0	1	00	1	X0	1
01	X	0	STR	0	X	1	1	01	0	10	0
01	X	1	LDR	0	1	0	1	01	1	X0	0
10	X	X	B	1	0	0	1	10	0	X1	0

Decodificador ALU

- Toma como entrada la señal de operación de la ALU y $funct_{4:1}$.

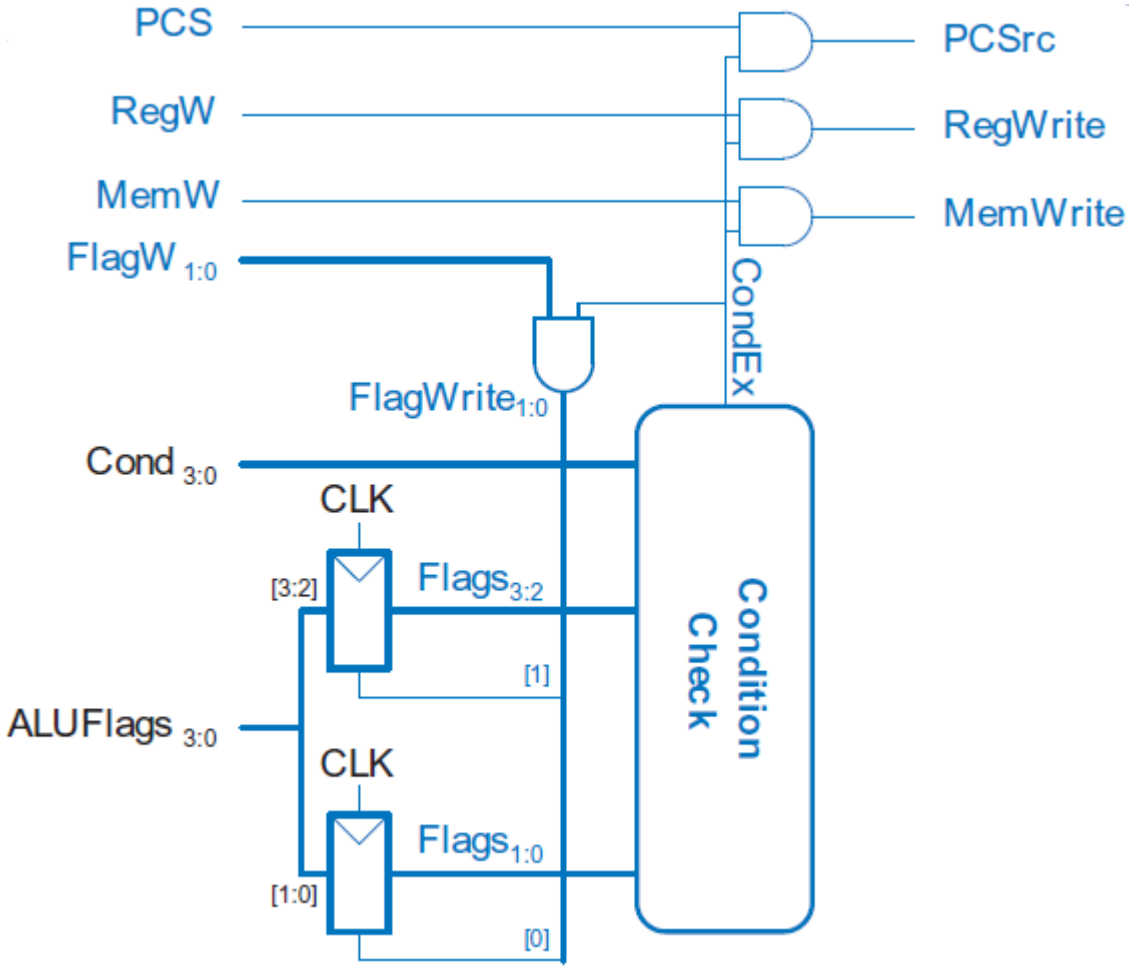
ALUOp	$Funct_{4:1}$ (*cmd*)	$Funct_0$ (*S*)	Type	$ALUControl_{1:0}$	$FlagW_{1:0}$
0	X	X	Not DP	00 (Add)	00
1	0100	0	ADD	00 (Add)	00
		1			11
	0010	0	SUB	01 (Sub)	00
		1			11
	0000	0	AND	10 (And)	00
		1			10
	1100	0	ORR	11 (Or)	00
		1			10

Lógica de PC

- Debe dar la salida PCS, si se escribe en PC o se está haciendo un salto.
- $PCS = \big((Rd == 15)\&RegW\big) | \, Branch$

Lógica condicional

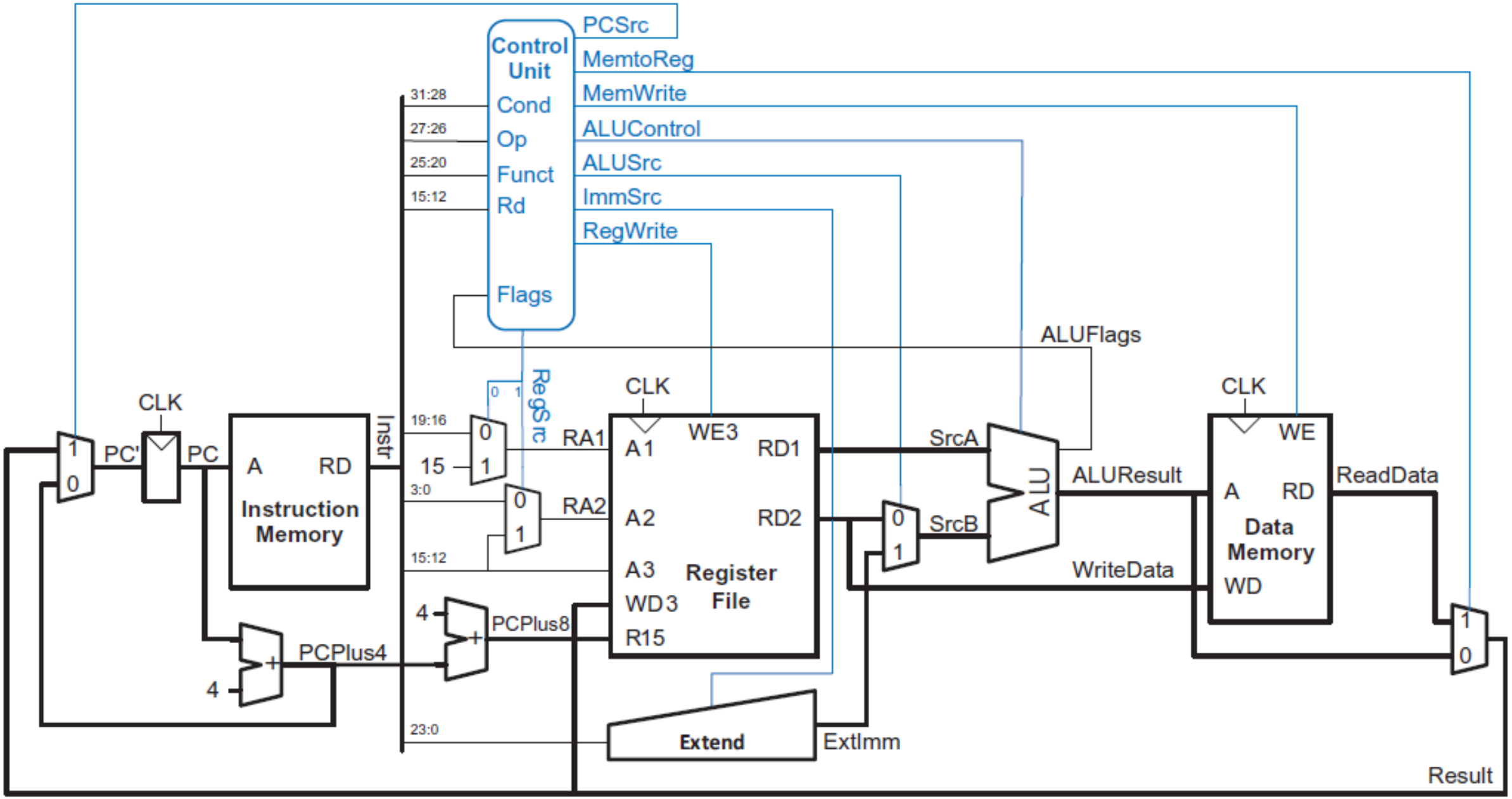
- Genera habilitación de señales (escritura y salto) que dependen de la lógica condicional.
- Posee los siguientes módulos.



Chequeo de condición

- Debe tomar en cuenta la condición a evaluar en la instrucción.

cond	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	$\overline{Z}$
0010	CS/HS	Carry set / unsigned higher or same	C
0011	CC/LO	Carry clear / unsigned lower	$\overline{C}$
0100	MI	Minus / negative	N
0101	PL	Plus / positive or zero	$\overline{N}$
0110	VS	Overflow / overflow set	V
0111	VC	No overflow / overflow clear	$\overline{V}$
1000	HI	Unsigned higher	$\overline{Z}C$
1001	LS	Unsigned lower or same	Z OR $\overline{C}$
1010	GE	Signed greater than or equal	$\overline{N \oplus V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\overline{Z}(\overline{N \oplus V})$
1101	LE	Signed less than or equal	Z OR $(N \oplus V)$
1110	AL (or none)	Always / unconditional	Ignored


Control Unit
PCSrc
MemtoReg
MemWrite
ALUControl
ALUSrc
ImmSrc
RegWrite
Cond
Op
Funct
Rd
Flags
31:28
27:26
25:20
15:12
ALUFlags
CLK
PC'
PC
Instruction Memory
A
RD
Instr
19:16
15
3:0
15:12
RegSrc
RA1
RA2
A1
A2
A3
WD3
R15
WE3
RD1
RD2
Register File
SrcA
SrcB
ALU
ALUResult
WriteData
Data Memory
WE
WD
ReadData
4
PCPlus4
PCPlus8
Extend
ExtImm
23:0
Result

Referencias

- Harris, S., & Harris, D. (2015). Digital design and computer architecture: arm edition. Morgan Kaufmann.
- González, J. (2020) Diseño de la organización de un procesador: Caso ARMv4

CE3201 – Taller de Diseño Digital

Diseño de la organización de un procesador: Caso ARMv4

PROFESOR: ING. LUIS BARBOZA ARTAVIA