

Chapter 7

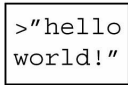


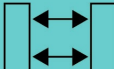
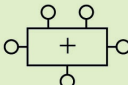

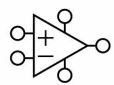


Digital Design and Computer Architecture: ARM® **Edition**

Sarah L. Harris and David Money
Harris



Chapter 7 :: Topics

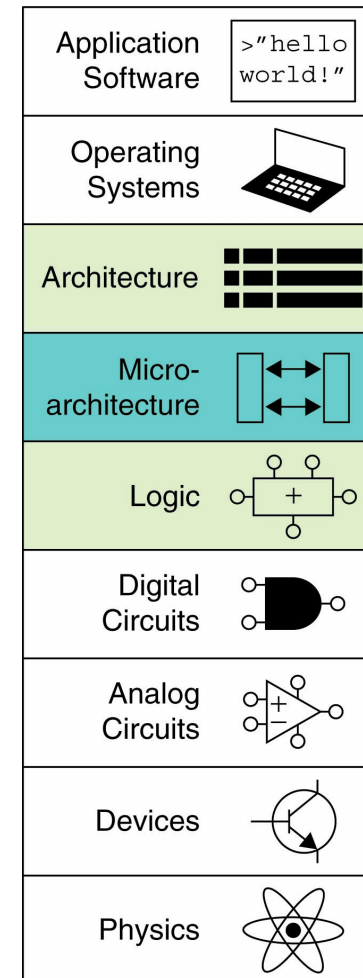
- **Introduction**
- **Performance Analysis**
- **Single-Cycle Processor**
- **Multicycle Processor**
- **Pipelined Processor**
- **Advanced Microarchitecture**

Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	



Introduction

- **Microarchitecture:** how to implement an architecture in hardware
- **Processor:**
 - **Datapath:** functional blocks
 - **Control:** control signals



Microarchitecture

- Multiple implementations for a single architecture:
 - **Single-cycle:** Each instruction executes in a single cycle
 - **Multicycle:** Each instruction is broken up into series of shorter steps
 - **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once



Processor Performance

- **Program execution time**

Execution Time = (#instructions)(cycles/instruction)(seconds/cycle)

- **Definitions:**

- CPI: Cycles/instruction
- clock period: seconds/cycle
- IPC: instructions/cycle = IPC

- **Challenge is to satisfy constraints of:**

- Cost
- Power
- Performance



ARM Processor

- Consider **subset** of ARM instructions:
 - **Data-processing instructions:**
 - **ADD, SUB, AND, ORR**
 - with register and immediate Src2, but **no shifts**
 - **Memory instructions:**
 - **LDR, STR**
 - with **positive immediate offset**
 - **Branch instructions:**
 - **B**



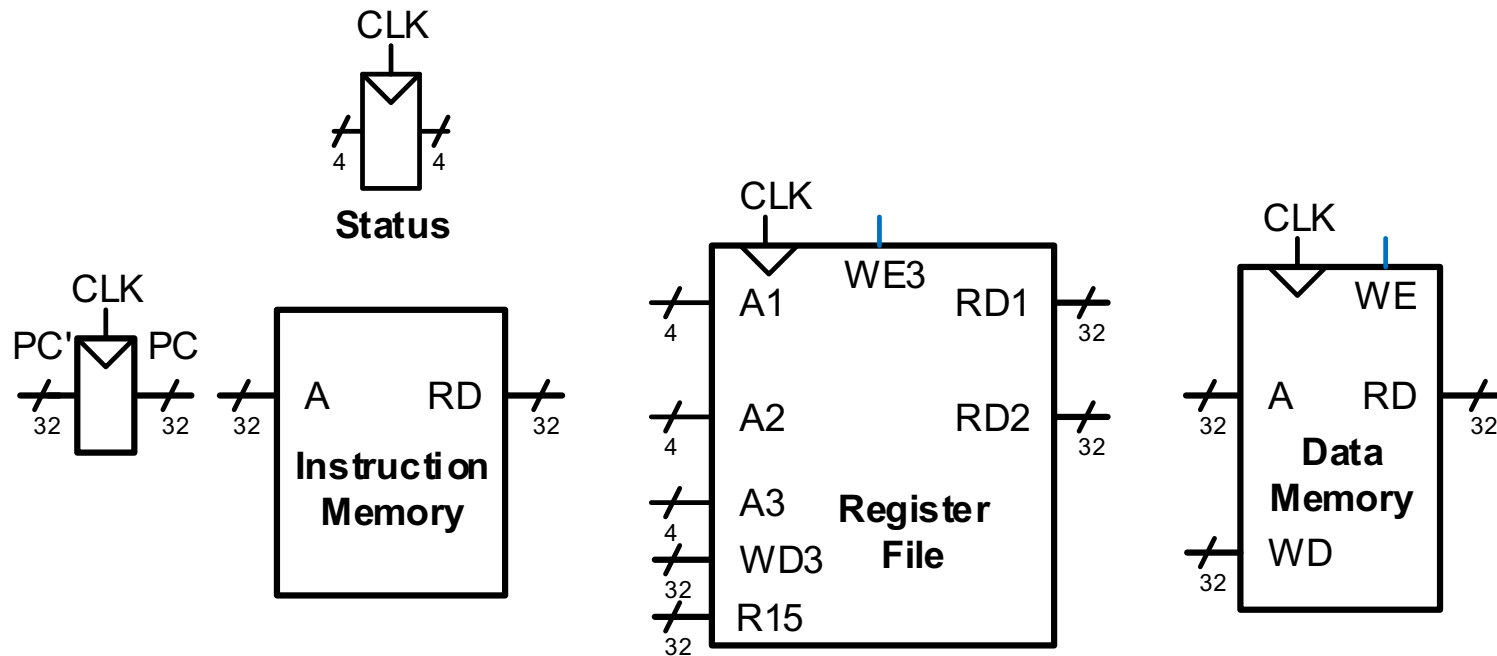
Architectural State Elements

Determines everything about a processor:

- Architectural state:
 - 16 registers (including PC)
 - Status register
- Memory



ARM Architectural State Elements



Single-Cycle ARM Processor

- Datapath
- Control



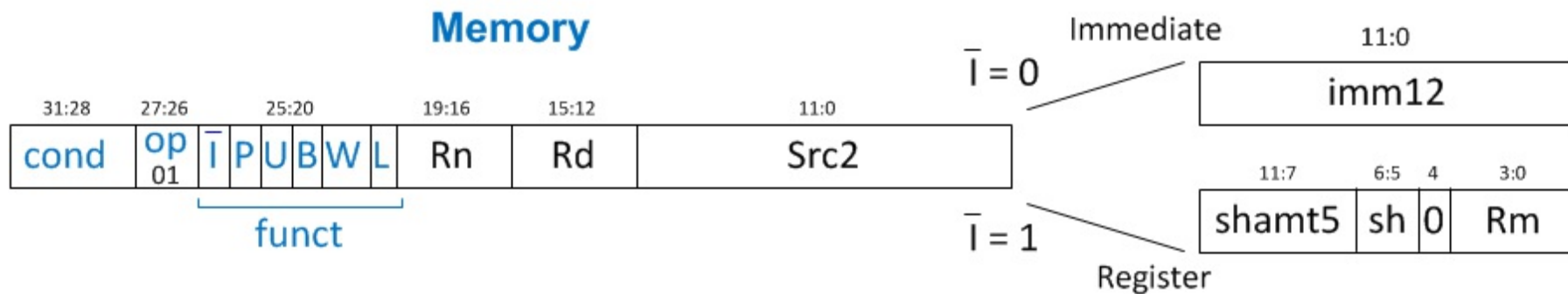
Single-Cycle ARM Processor

- **Datapath**
- Control



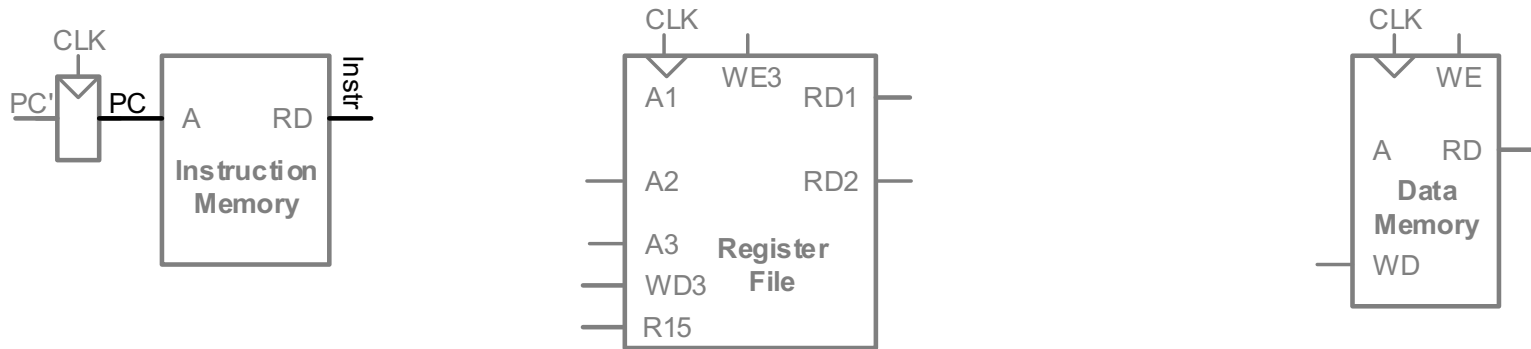
Single-Cycle ARM Processor

- **Datapath:** start with LDR instruction
- **Example:** LDR R1, [R2, #5]
LDR Rd, [Rn, imm12]



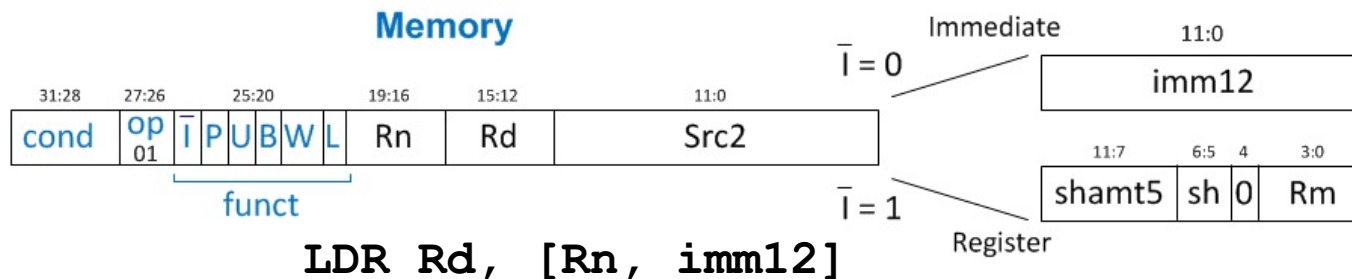
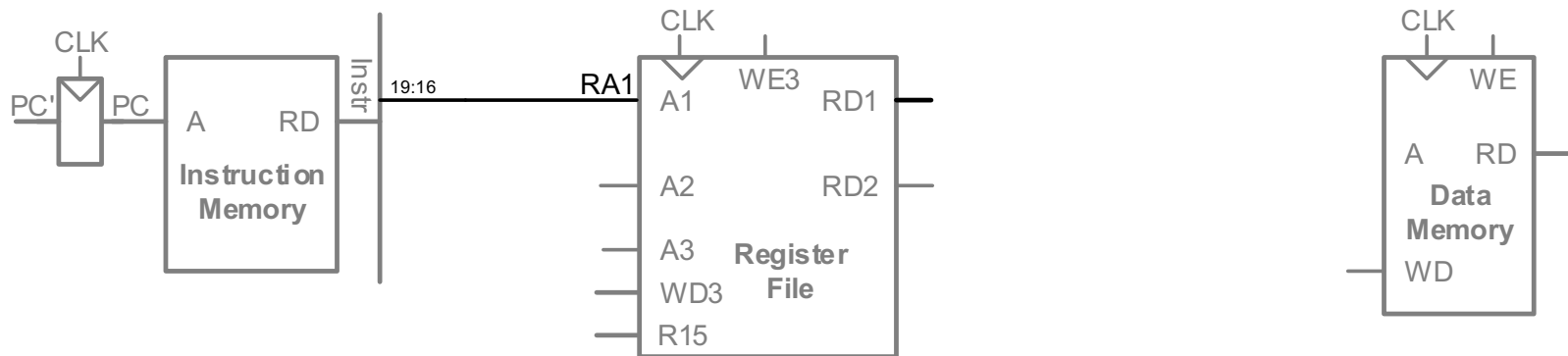
Single-Cycle Datapath: LDR fetch

STEP 1: Fetch instruction



Single-Cycle Datapath: LDR Reg Read

STEP 2: Read source operands from RF

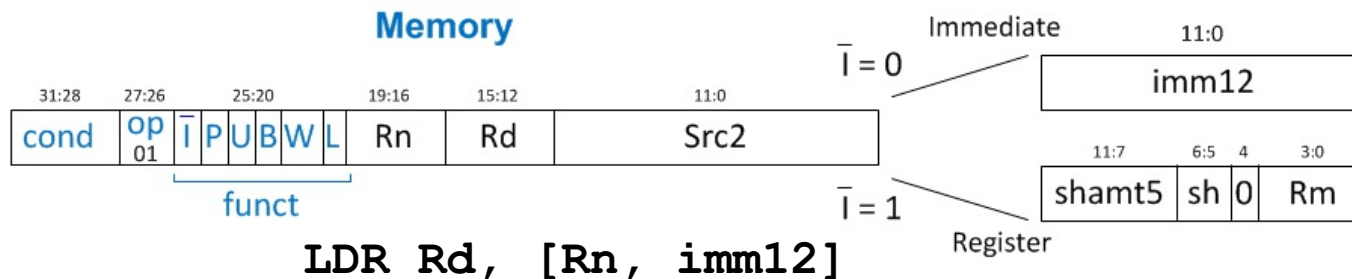
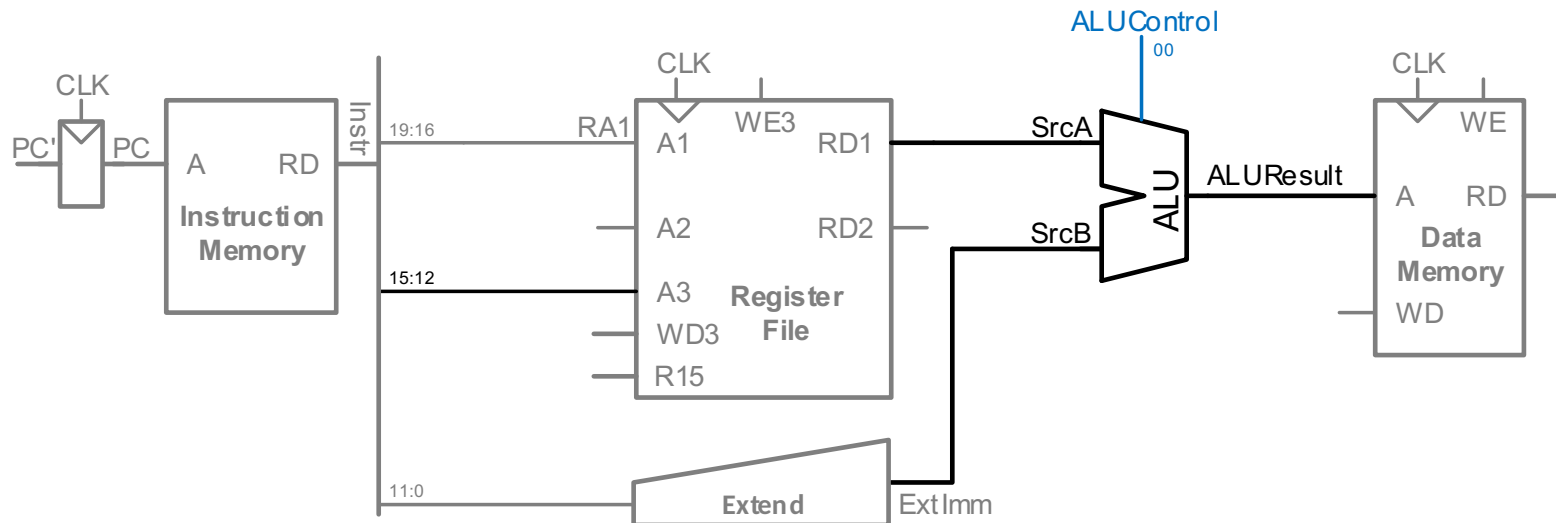


STEP 3: Extend the immediate



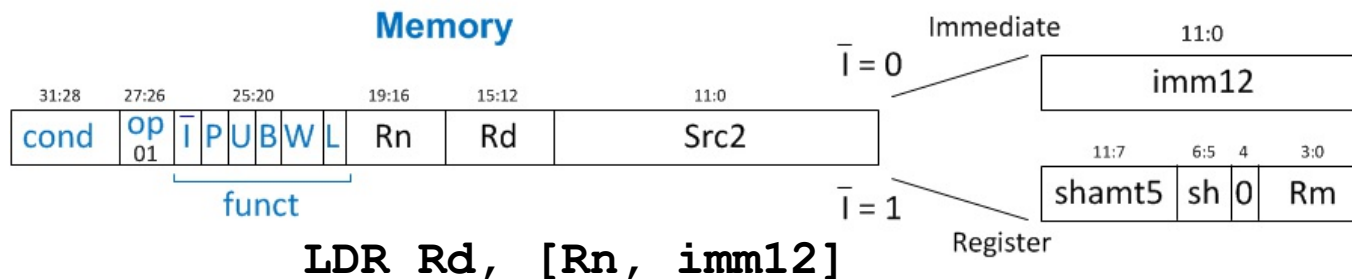
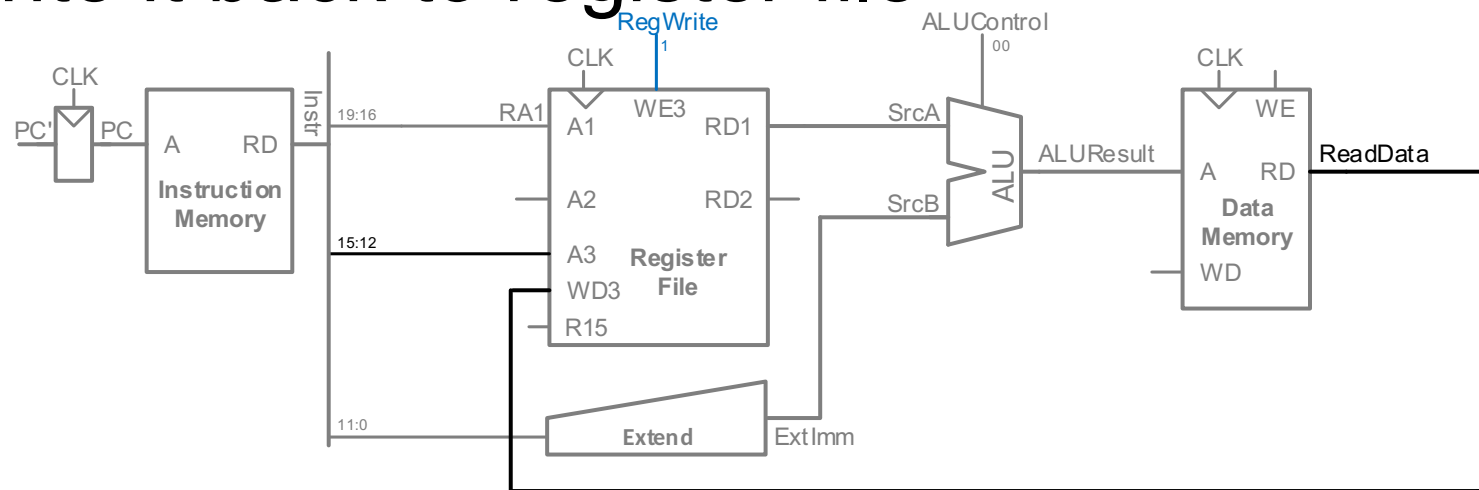
Single-Cycle Datapath: LDR Address

STEP 4: Compute the memory address



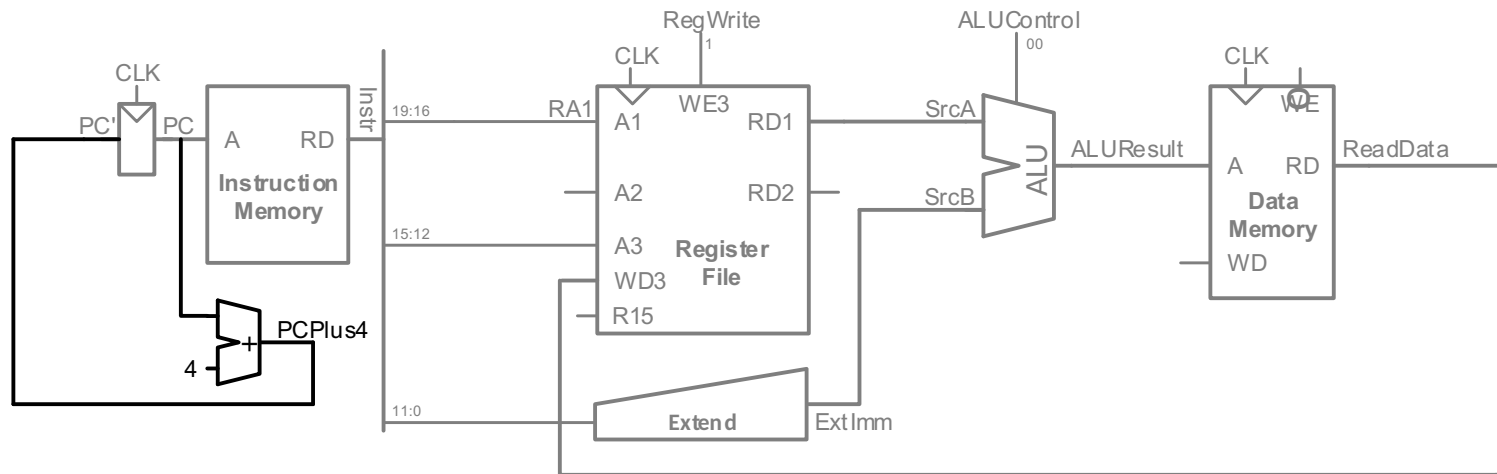
Single-Cycle Datapath: LDR Mem Read

STEP 5: Read data from memory and write it back to register file



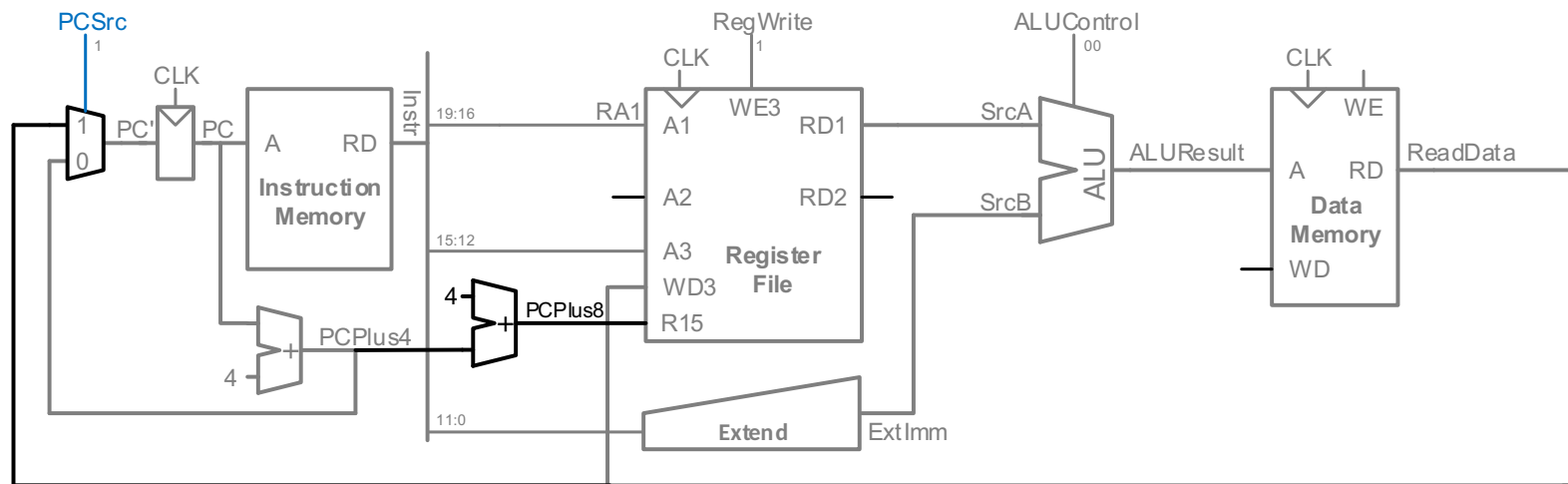
Single-Cycle Datapath: PC Increment

STEP 6: Determine address of next instruction



Single-Cycle Datapath: Access to PC

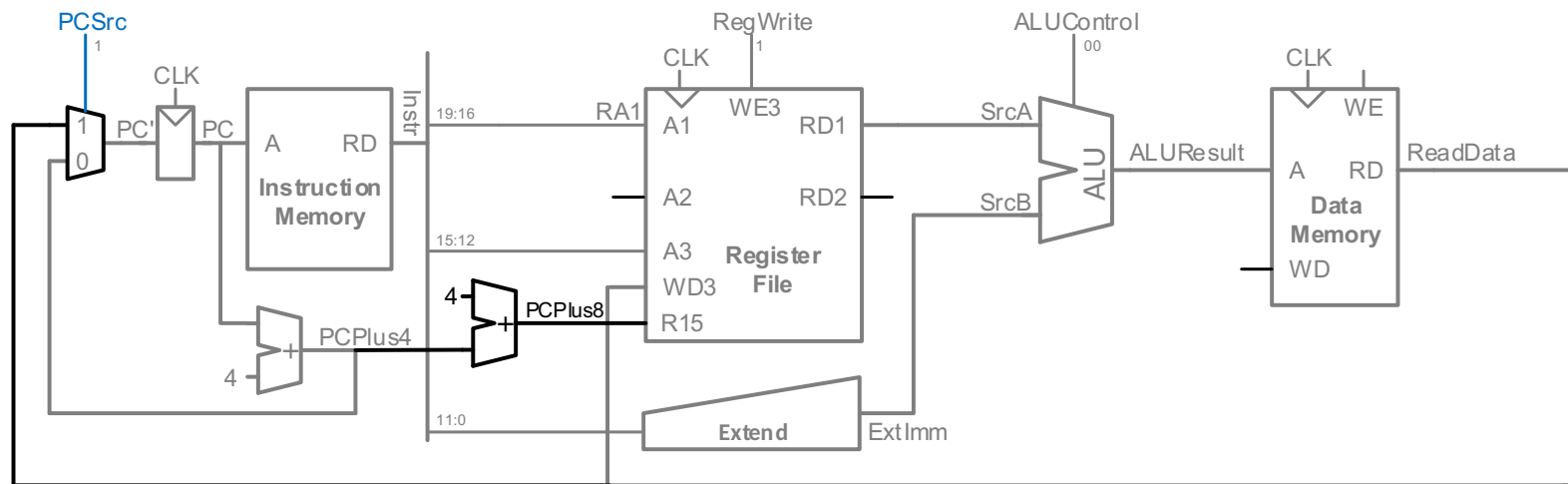
PC can be source/destination of instruction



Single-Cycle Datapath: Access to PC

PC can be source/destination of instruction

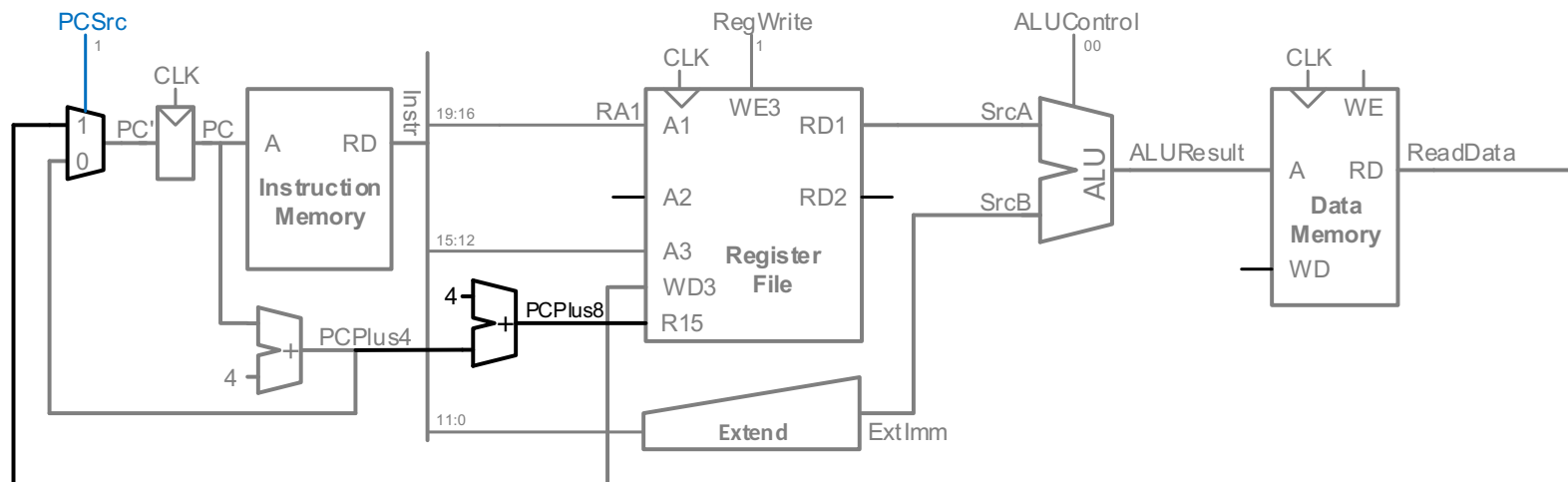
- **Source:** R15 must be available in Register File
 - **PC** is read as the current **PC plus 8**



Single-Cycle Datapath: Access to PC

PC can be source/destination of instruction

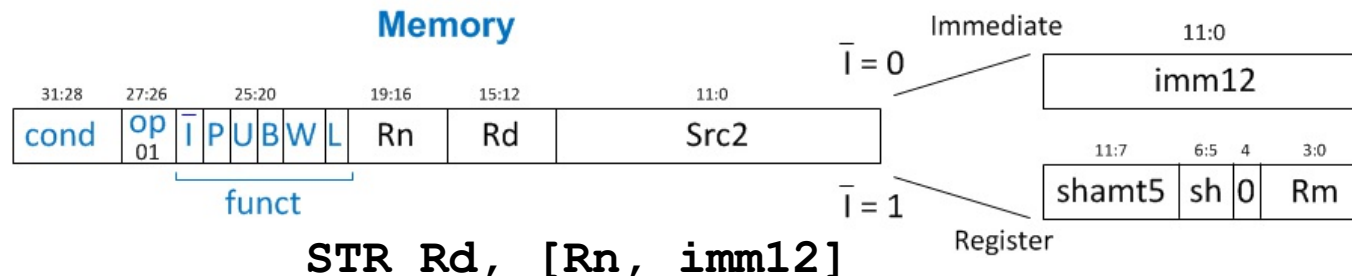
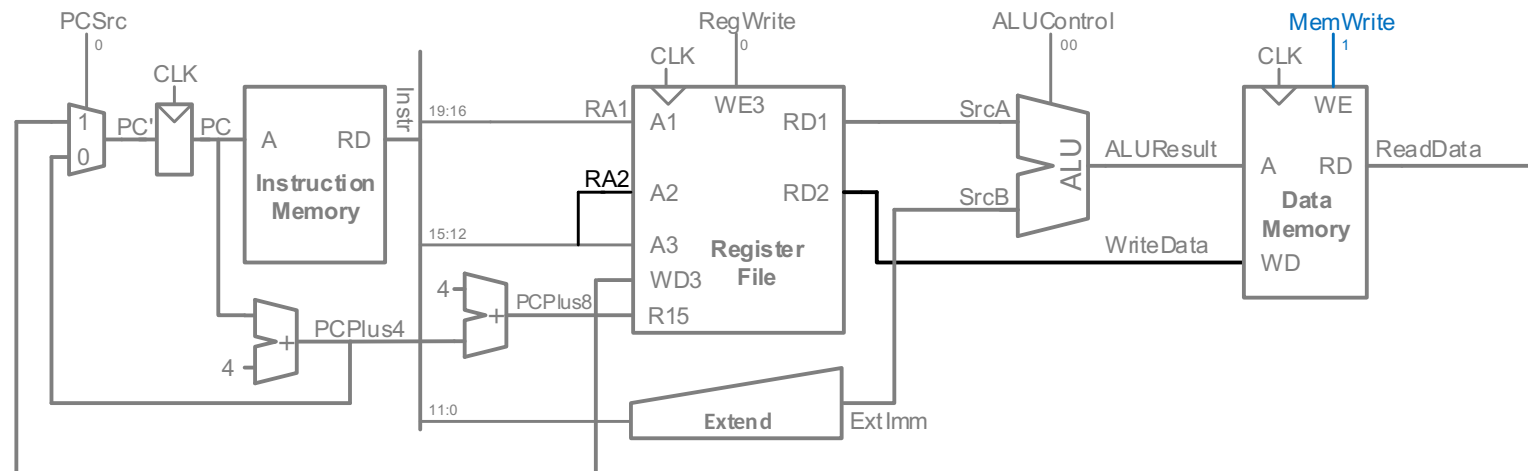
- **Source:** R15 must be available in Register File
 - **PC** is read as the current **PC plus 8**
- **Destination:** Be able to write result to PC



Single-Cycle Datapath: STR

Expand datapath to handle STR:

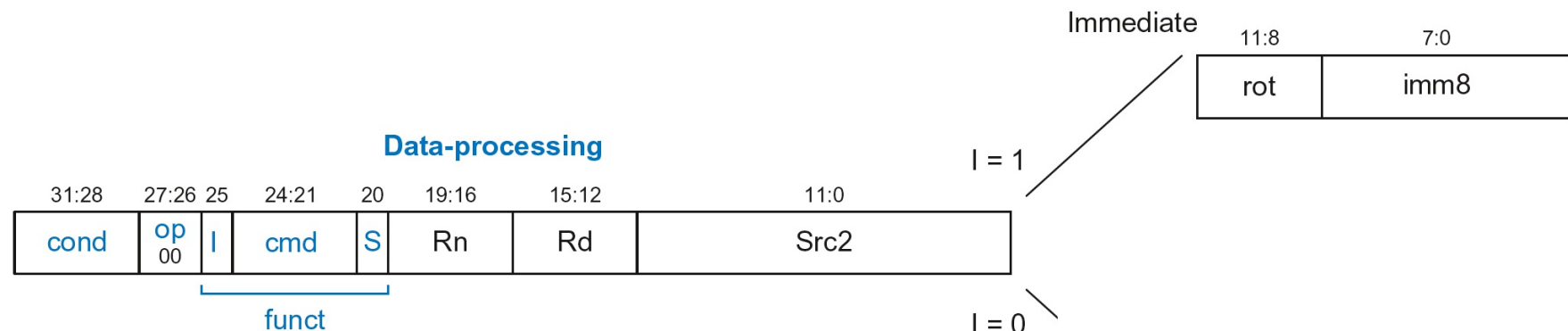
- Write data in R_d to memory



Single-Cycle Datapath: Data-processing

With **immediate** Src2:

- Read from R_n and $Imm8$ ($ImmSrc$ chooses the zero-extended $Imm8$ instead of $Imm12$)
- Write $ALUResult$ to register file
- Write to R_d



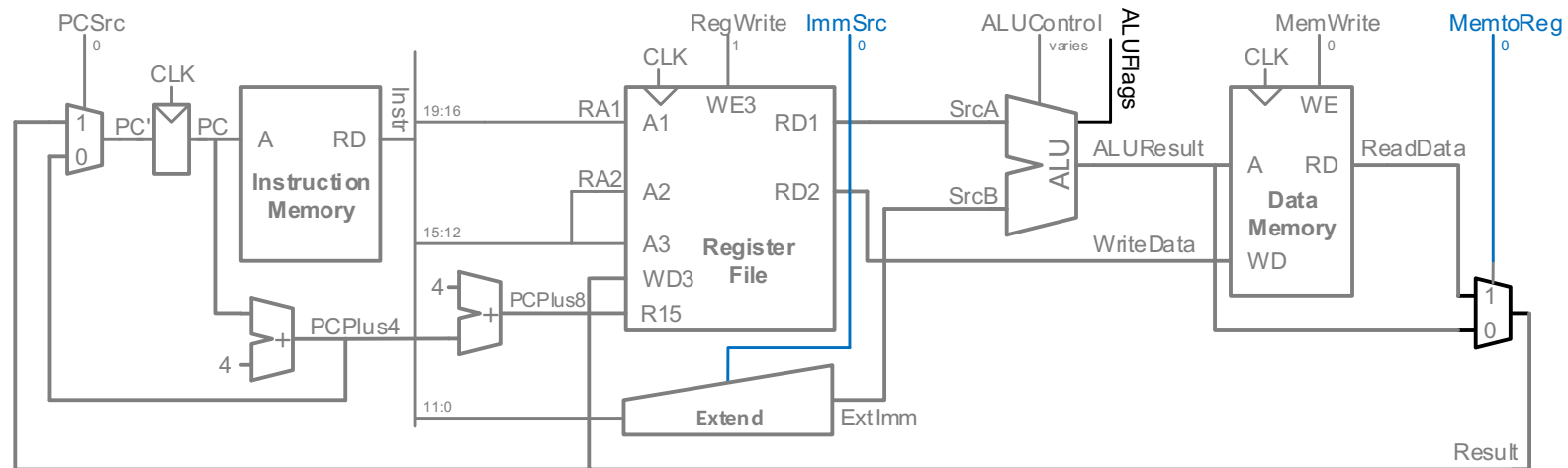
ADD R_d , R_n , imm8



Single-Cycle Datapath: Data-processing

With **immediate Src2**:

- Read from R_n and $Imm8$ ($ImmSrc$ chooses the zero-extended $Imm8$ instead of $Imm12$)
- Write $ALUResult$ to register file
- Write to R_d



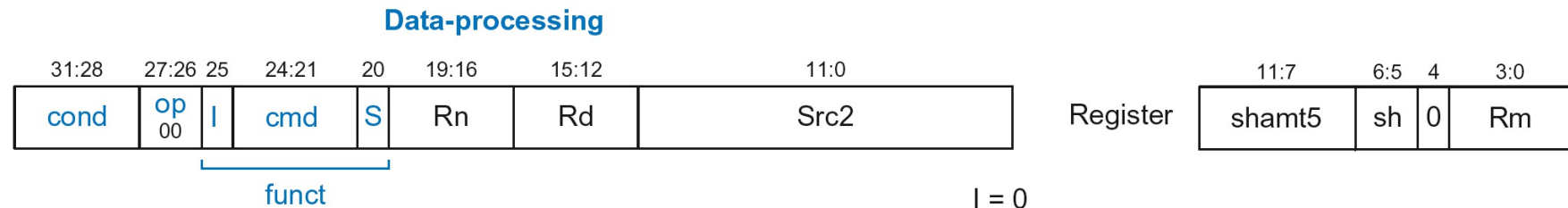
ADD R_d , R_n , $imm8$



Single-Cycle Datapath: Data-processing

With **register Src2**:

- Read from R_n and R_m (instead of $Imm8$)
- Write $ALUResult$ to register file
- Write to R_d

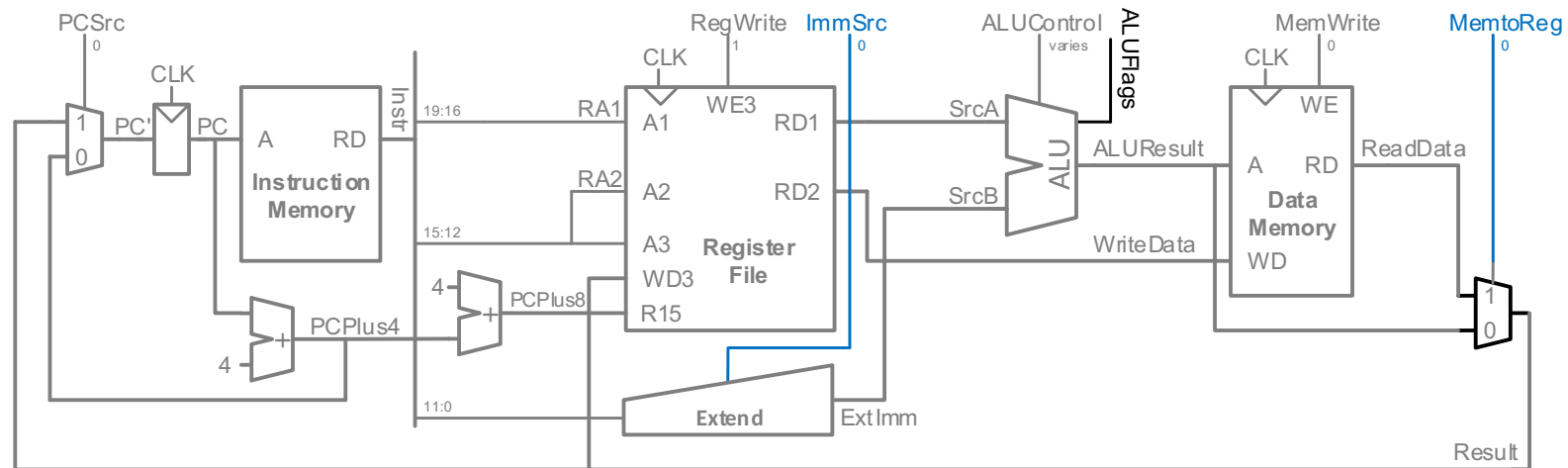


ADD R_d , R_n , R_m



Single-Cycle Datapath: Data-processing

With **immediate Src2**:



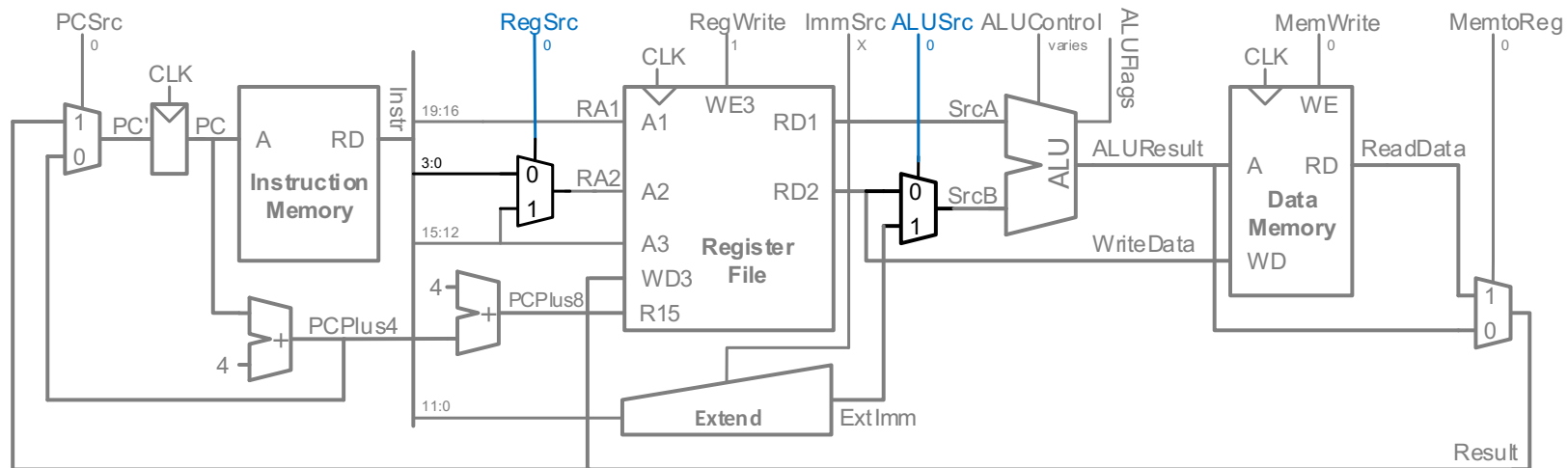
ADD Rd, Rn, imm8



Single-Cycle Datapath: Data-processing

With **register Src2**:

- Read from R_n and R_m (instead of $Imm8$)
- Write $ALUResult$ to register file
- Write to R_d



ADD R_d, R_n, R_m

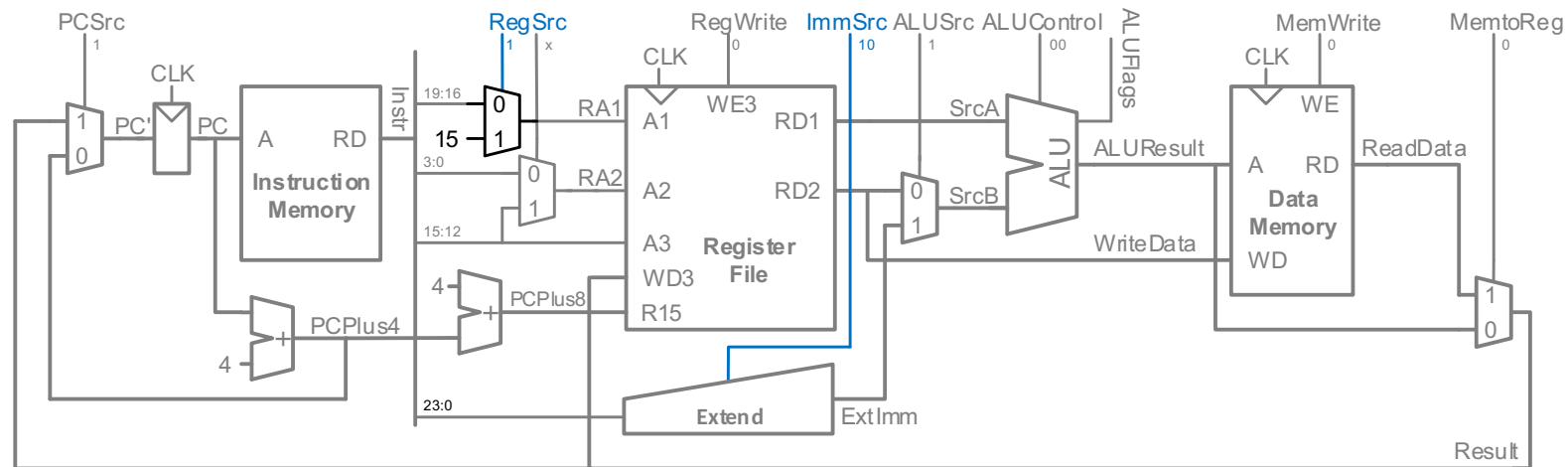


Single-Cycle Datapath: B

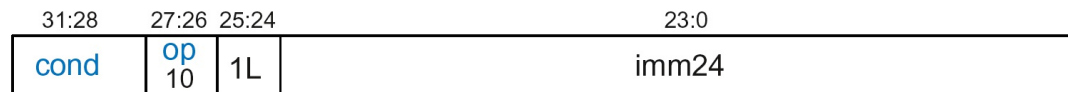
Calculate branch target address:

$$\text{BTA} = (\text{ExtImm}) + (\text{PC} + 8)$$

$$\text{ExtImm} = \text{Imm24} \ll 2 \text{ and sign-extended}$$



Branch

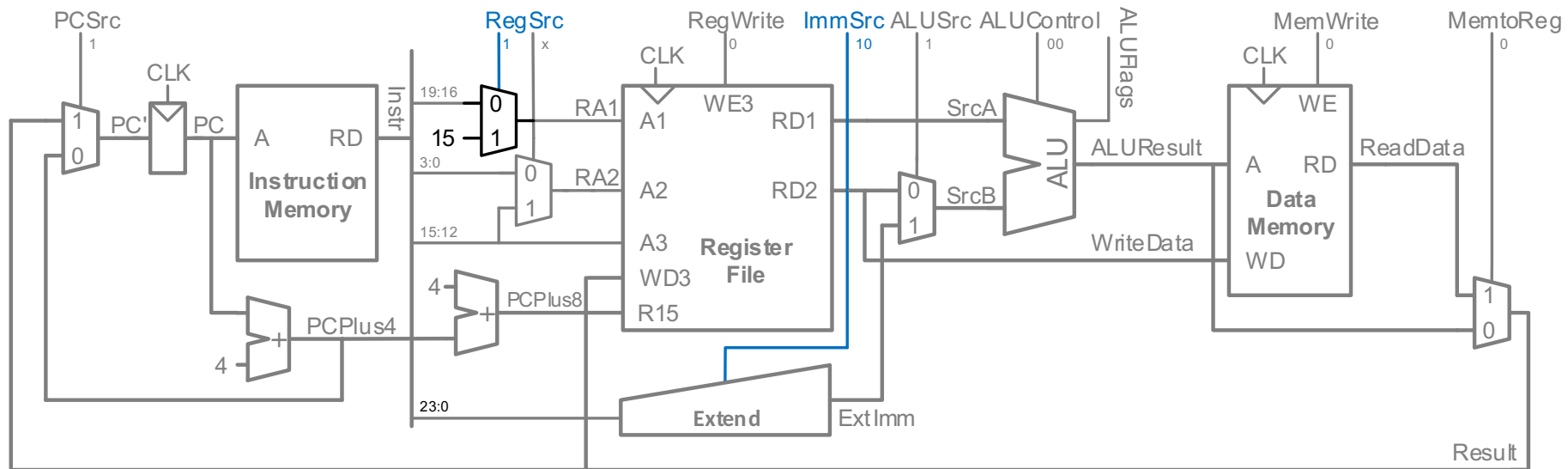


funct

B Label



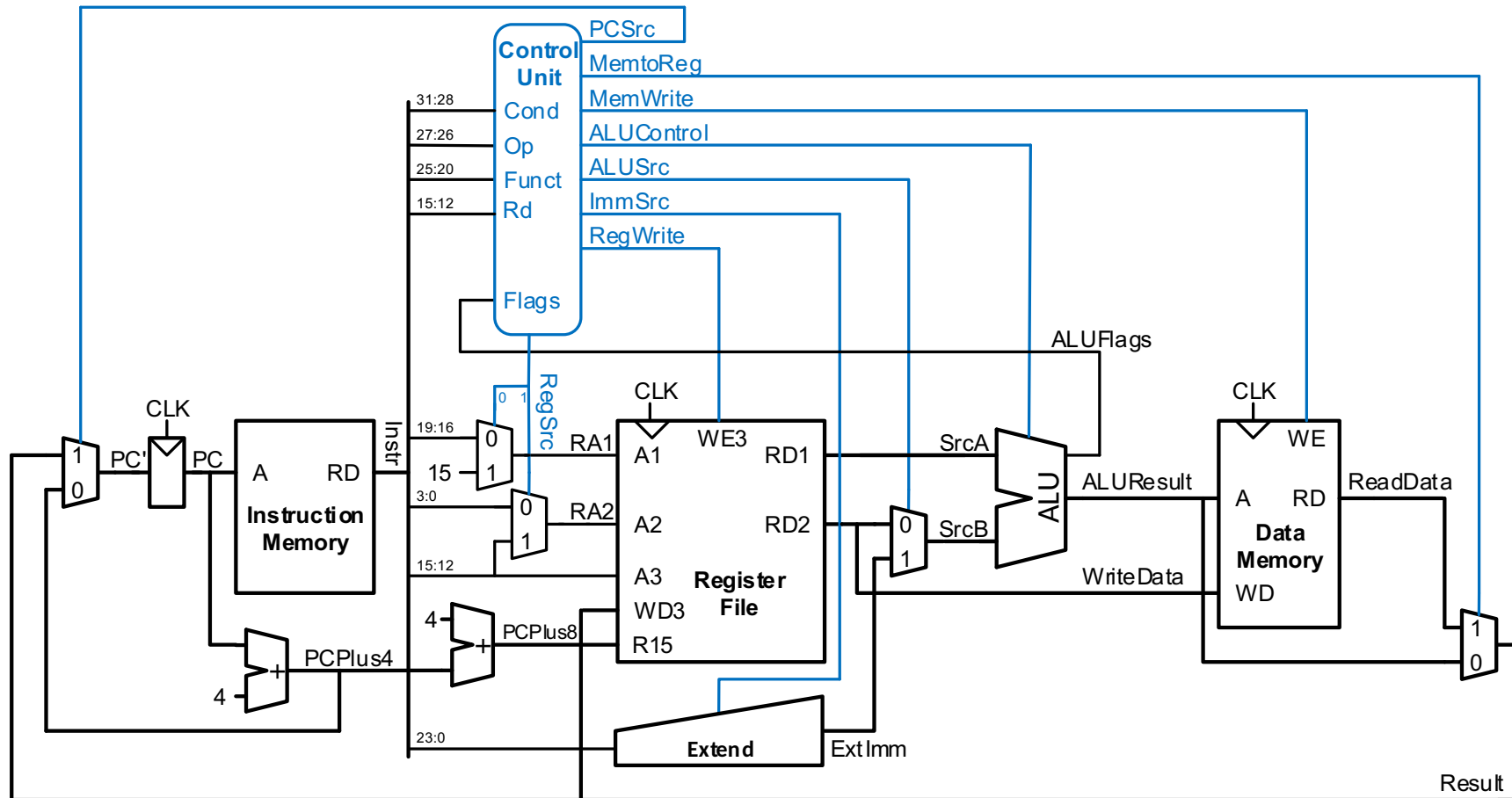
Single-Cycle Datapath: ExtImm



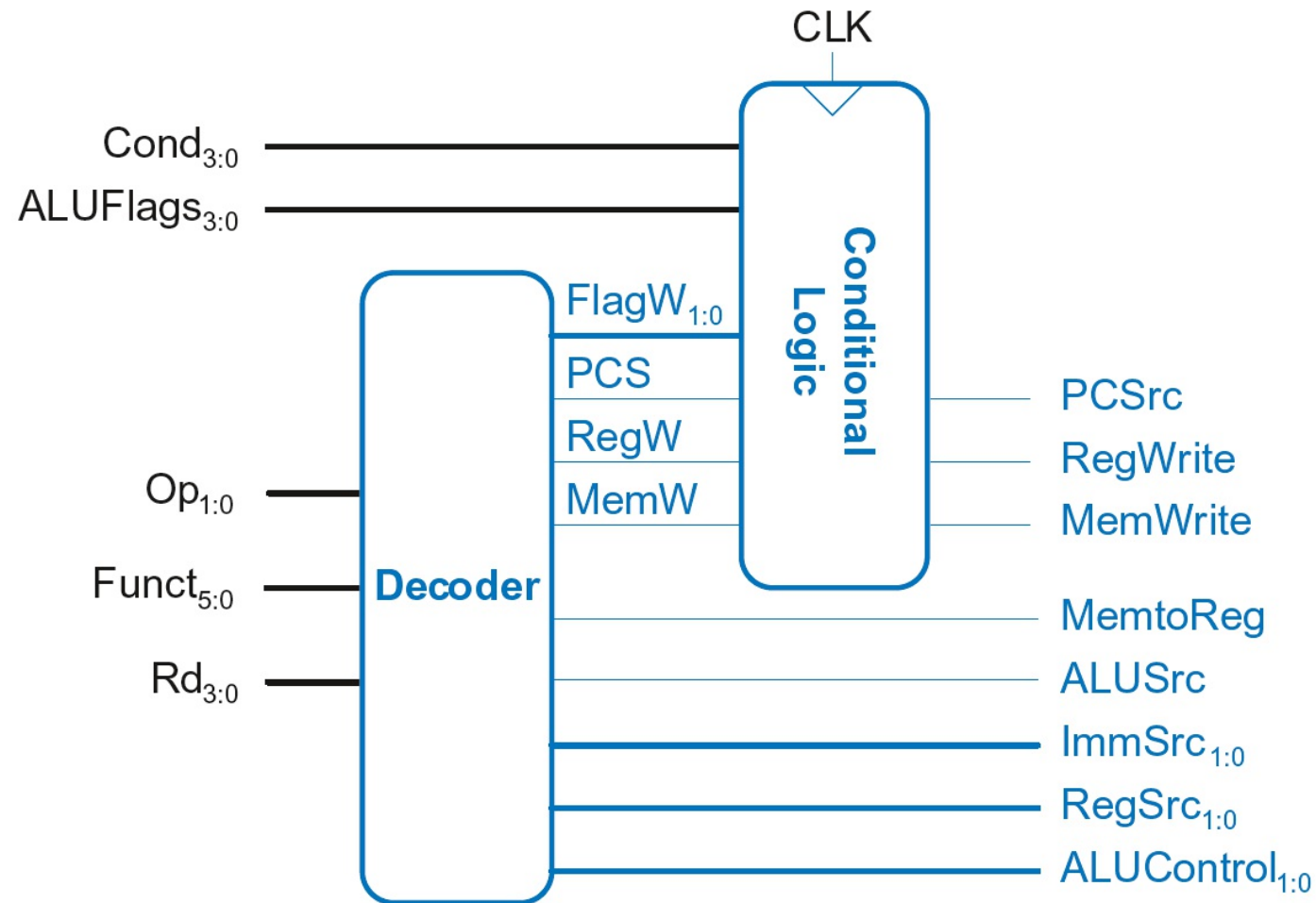
ImmSrc _{1:0}	ExtImm	Description
00	{24'b0, Instr _{7:0} }	Zero-extended <i>imm8</i>
01	{20'b0, Instr _{11:0} }	Zero-extended <i>imm12</i>
10	{6{Instr ₂₃ }, Instr _{23:0} }	Sign-extended <i>imm24</i>



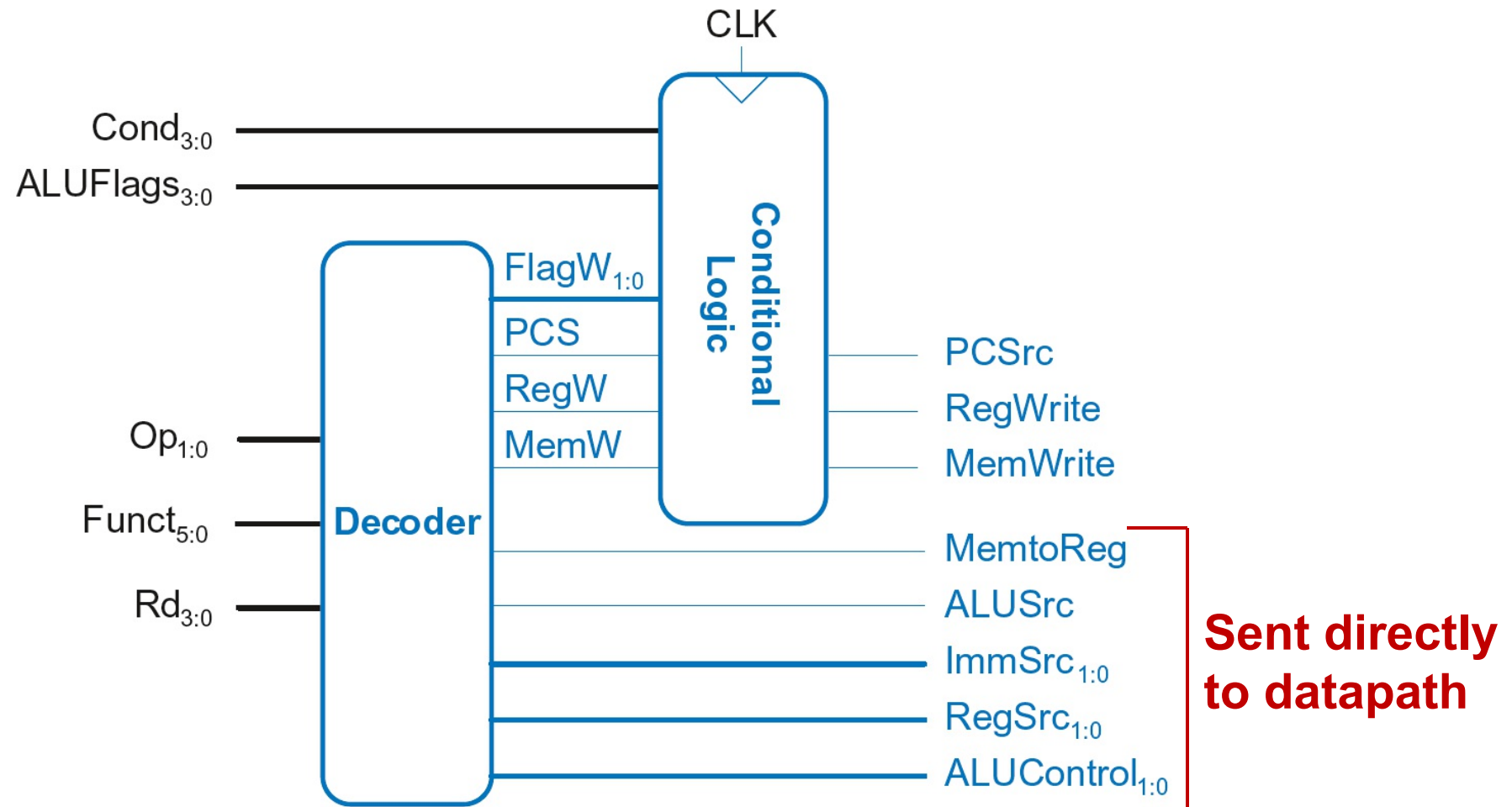
Single-Cycle ARM Processor



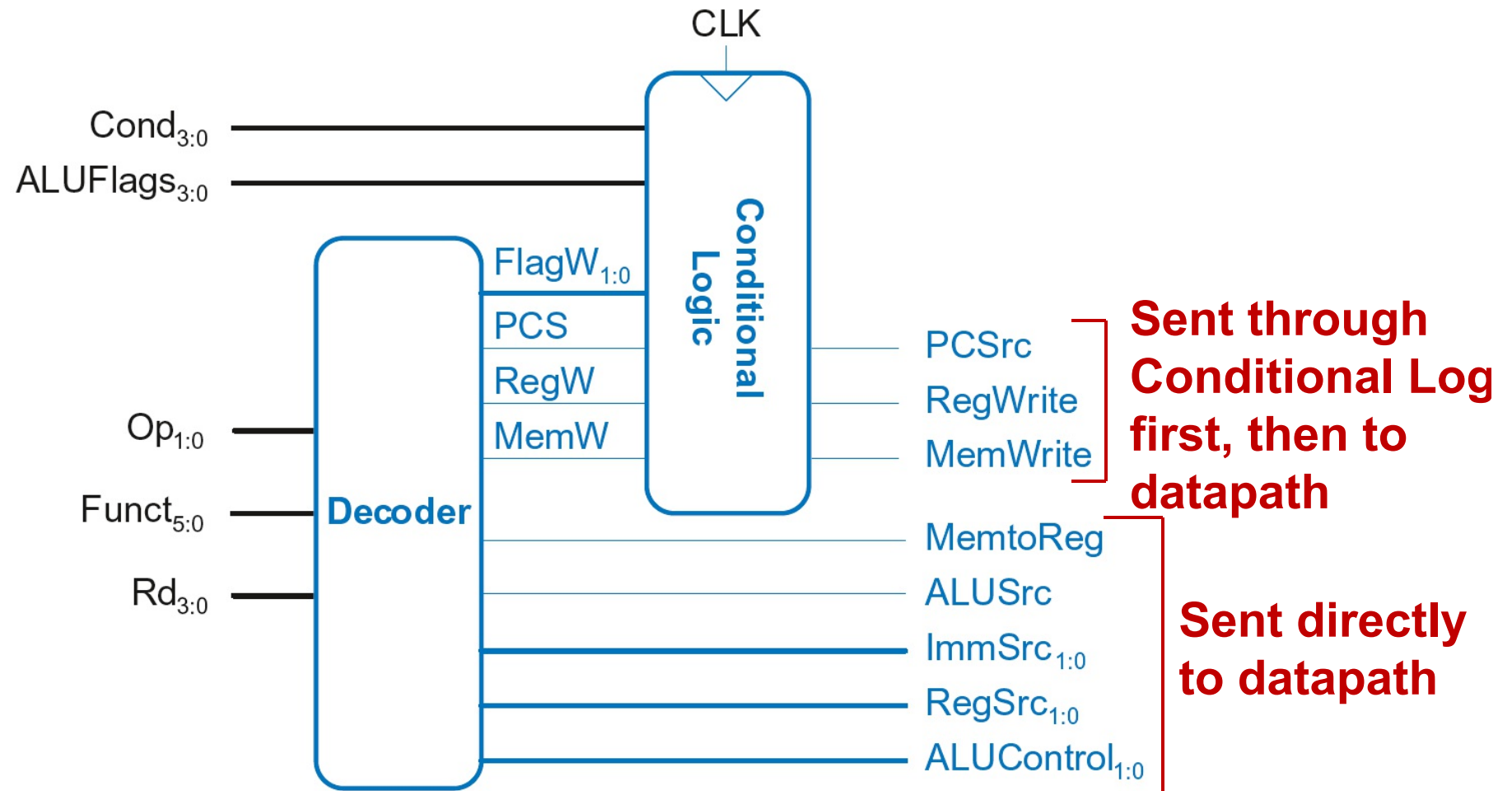
Single-Cycle Control



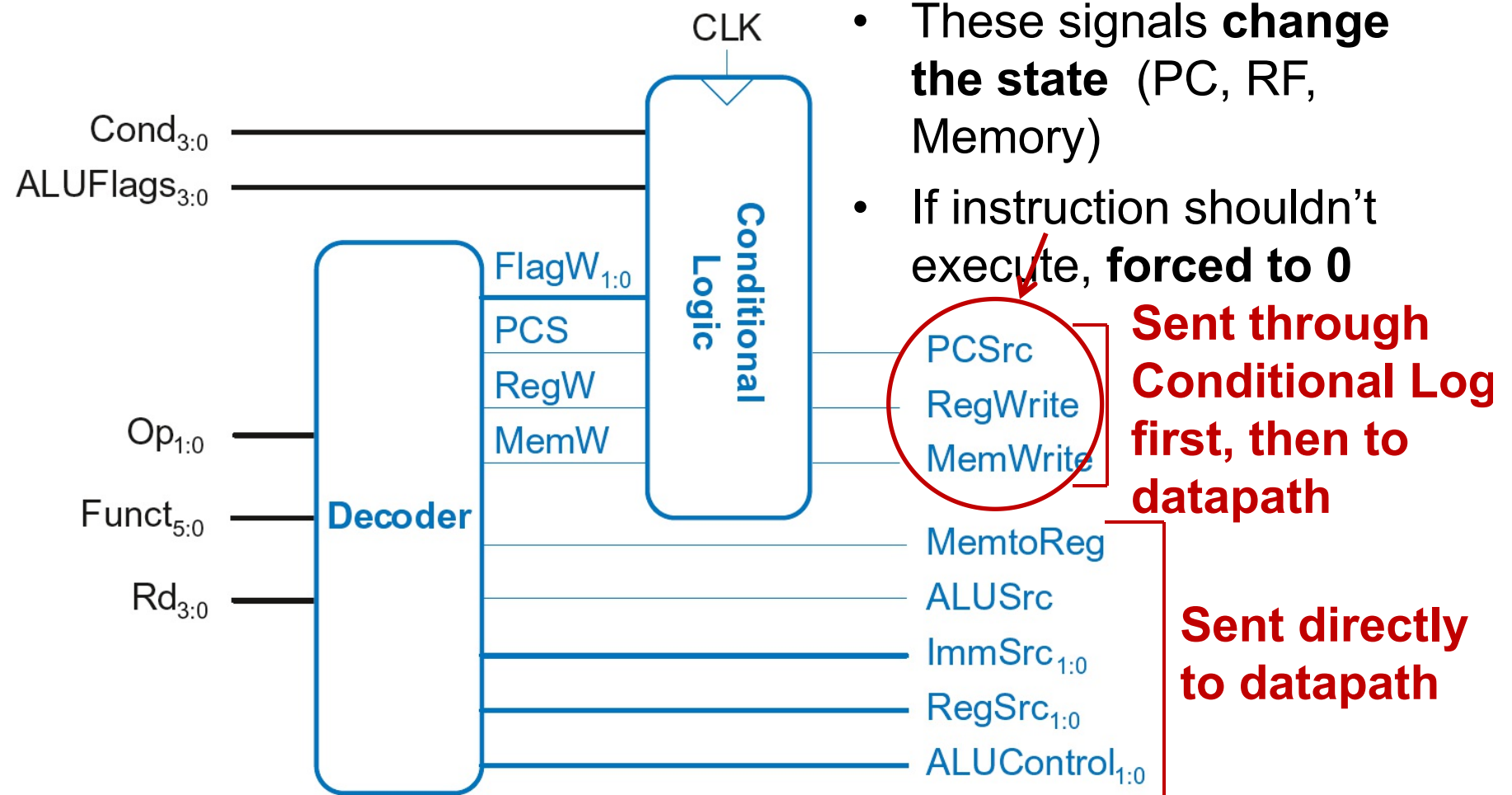
Single-Cycle Control



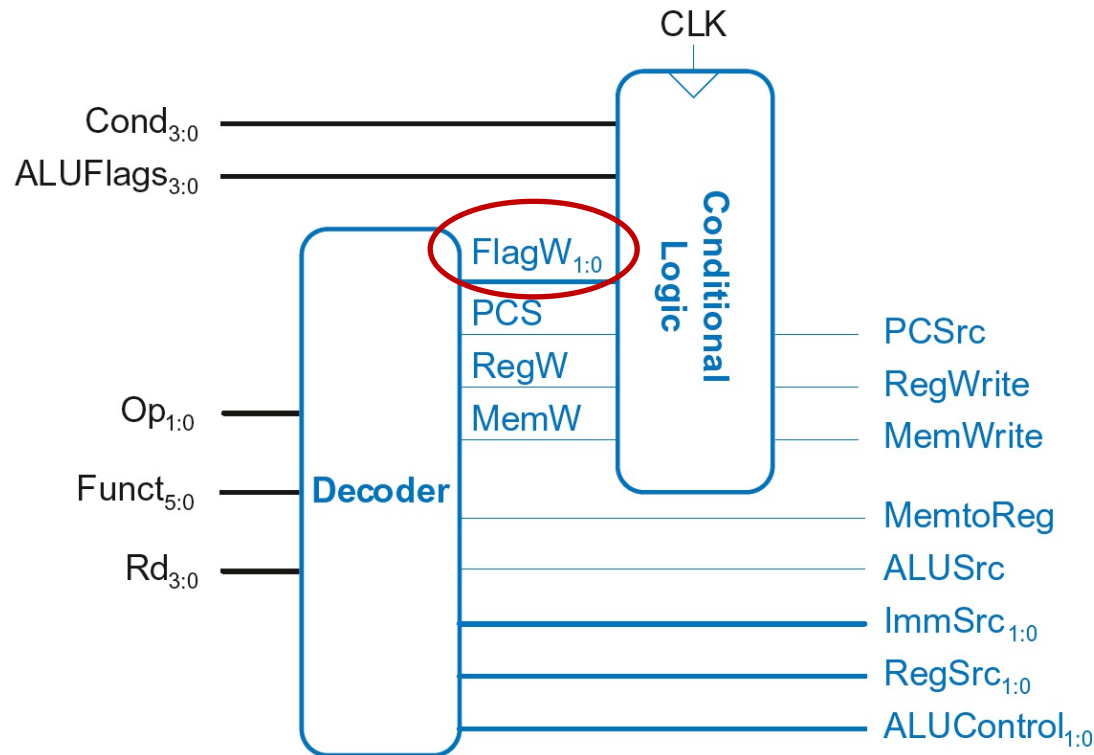
Single-Cycle Control



Single-Cycle Control



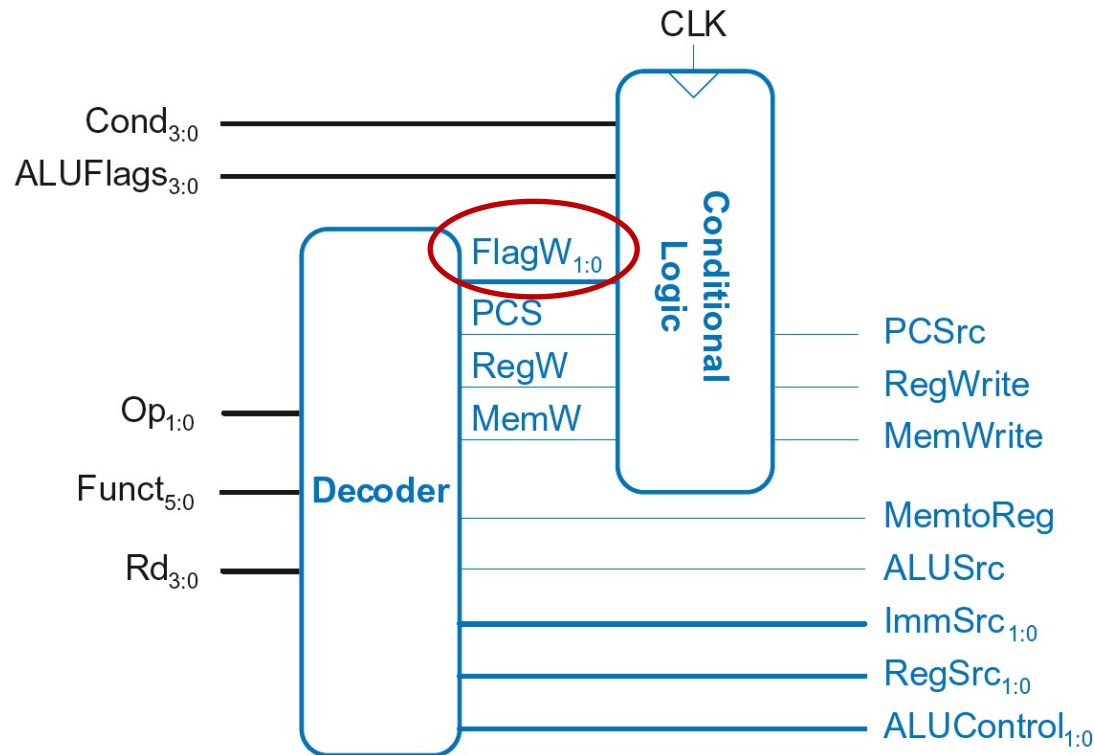
Single-Cycle Control



- **$FlagW_{1:0}$:** Flag Write signal, asserted when $ALUFlags$ should be saved (i.e., on instruction with $S=1$)



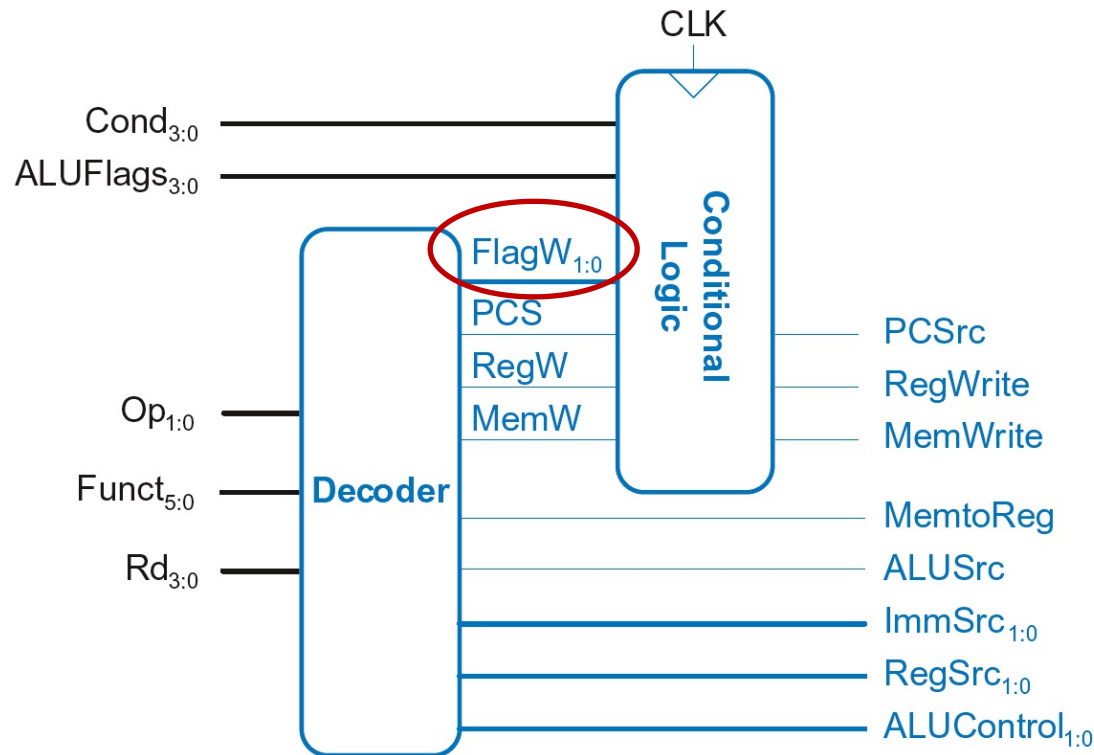
Single-Cycle Control



- **$FlagW_{1:0}$** : Flag Write signal, asserted when $ALUFlags$ should be saved (i.e., on instruction with $S=1$)
- ADD, SUB update all flags (**NZCV**)
- AND, ORR only update **NZ** flags



Single-Cycle Control



- **$FlagW_{1:0}$:** Flag Write signal, asserted when $ALUFlags$ should be saved (i.e., on instruction with $S=1$)
- ADD, SUB update all flags (**NZCV**)
- AND, ORR only update **NZ** flags
- So, two bits needed:

$FlagW_1 = 1$: NZ

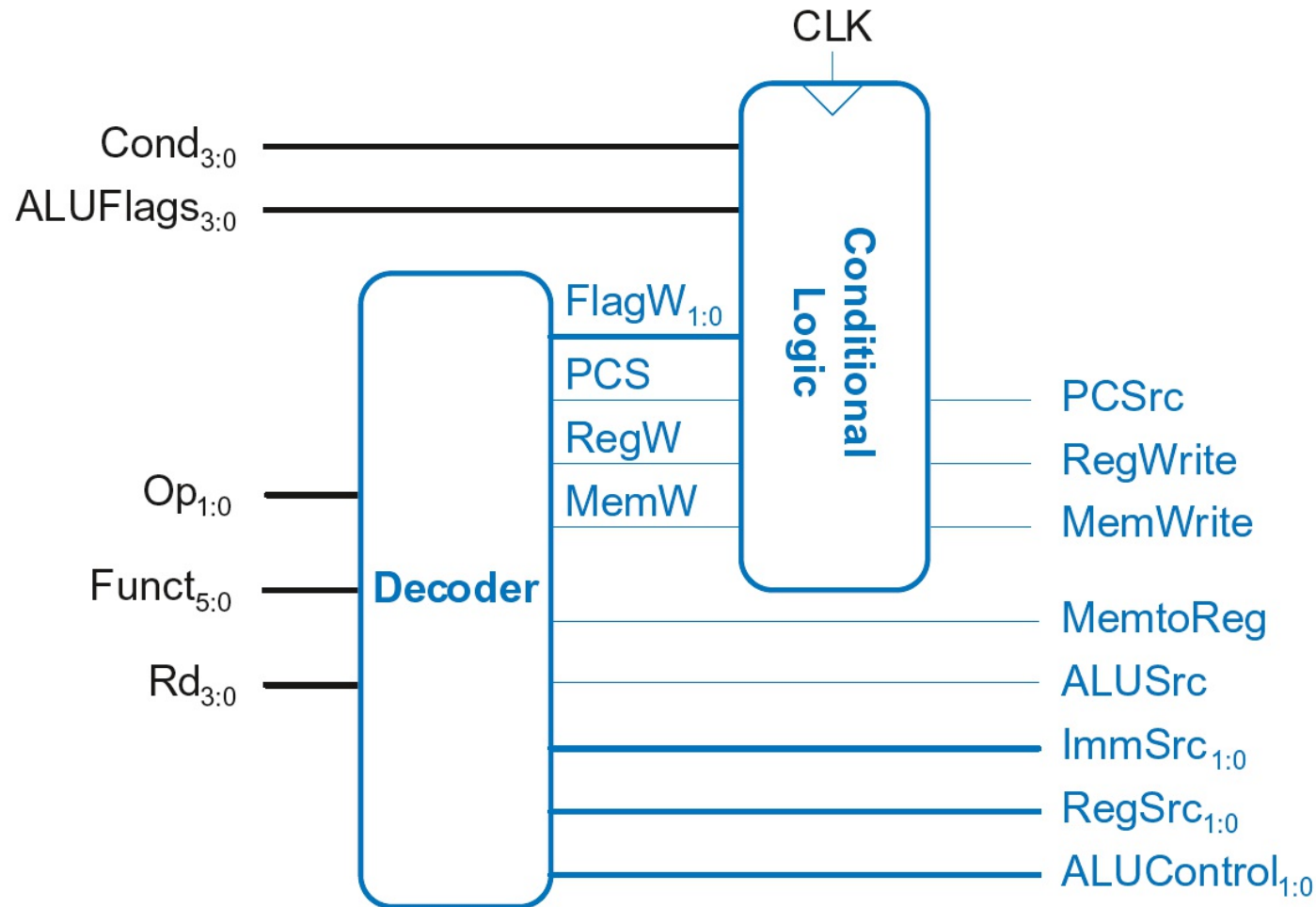
saved (ALUFlags_{3:2} saved)

$FlagW_0 = 1$: CV

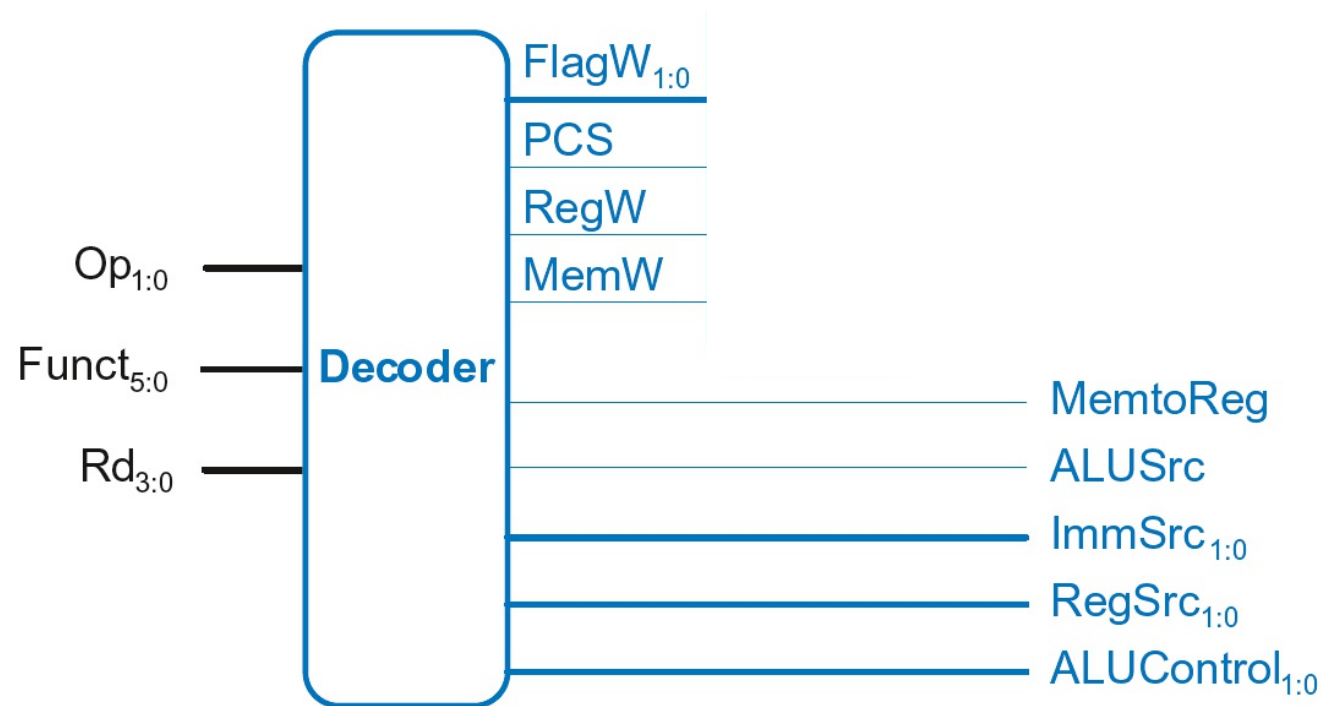
saved (ALUFlags_{1:0} saved)



Single-Cycle Control



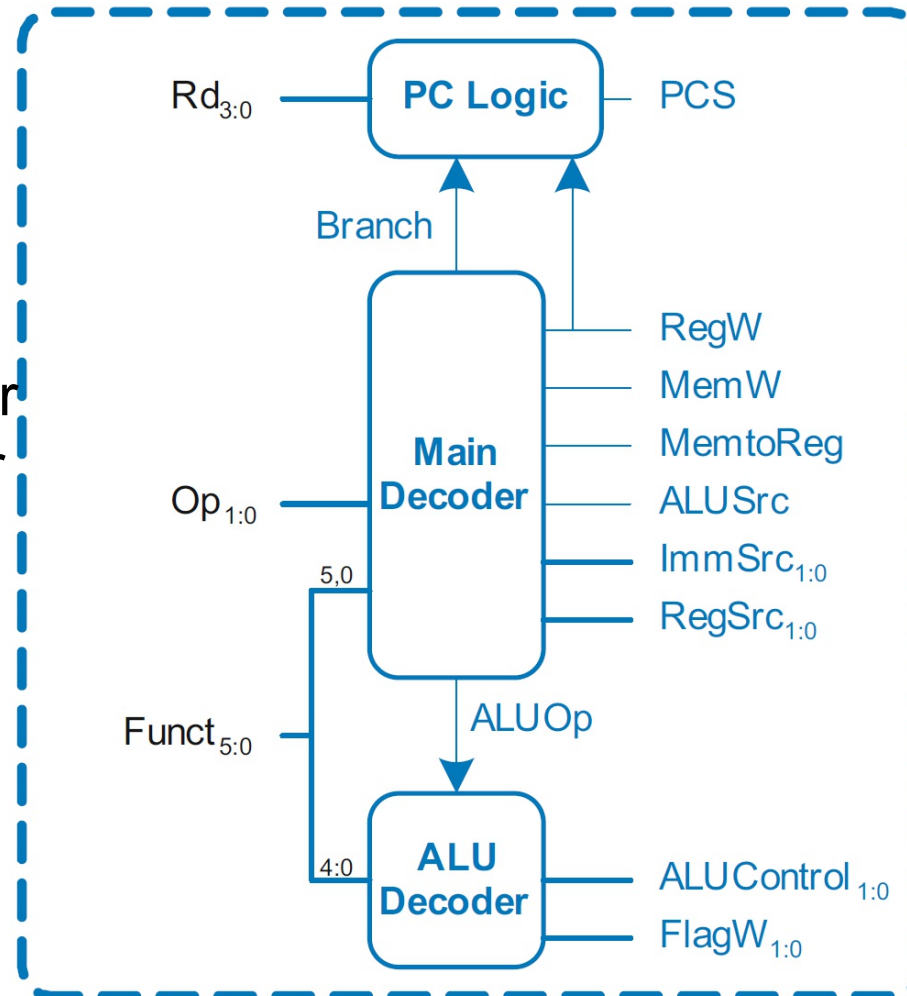
Single-Cycle Control: Decoder



Single-Cycle Control: Decoder

Submodules:

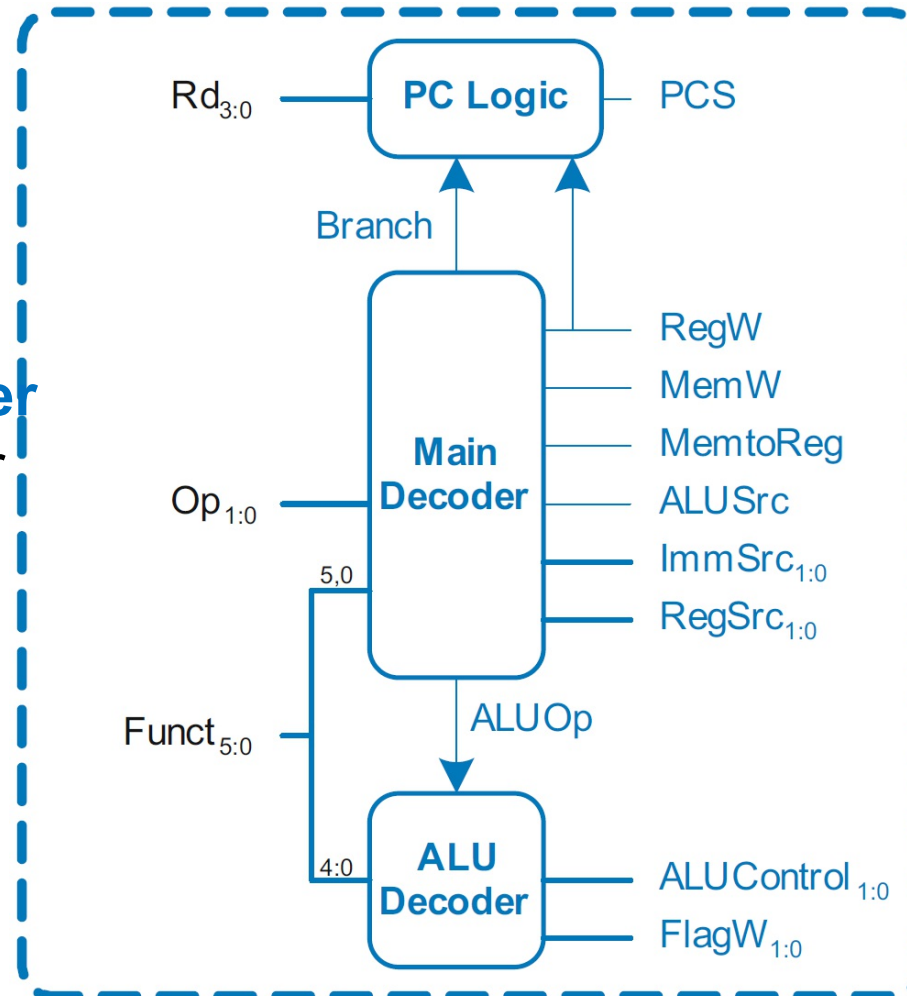
- Main Decoder
- ALU Decoder
- PC Logic



Single-Cycle Control: Decoder

Submodules:

- **Main Decoder**
- ALU Decoder
- PC Logic



Control Unit: Main Decoder

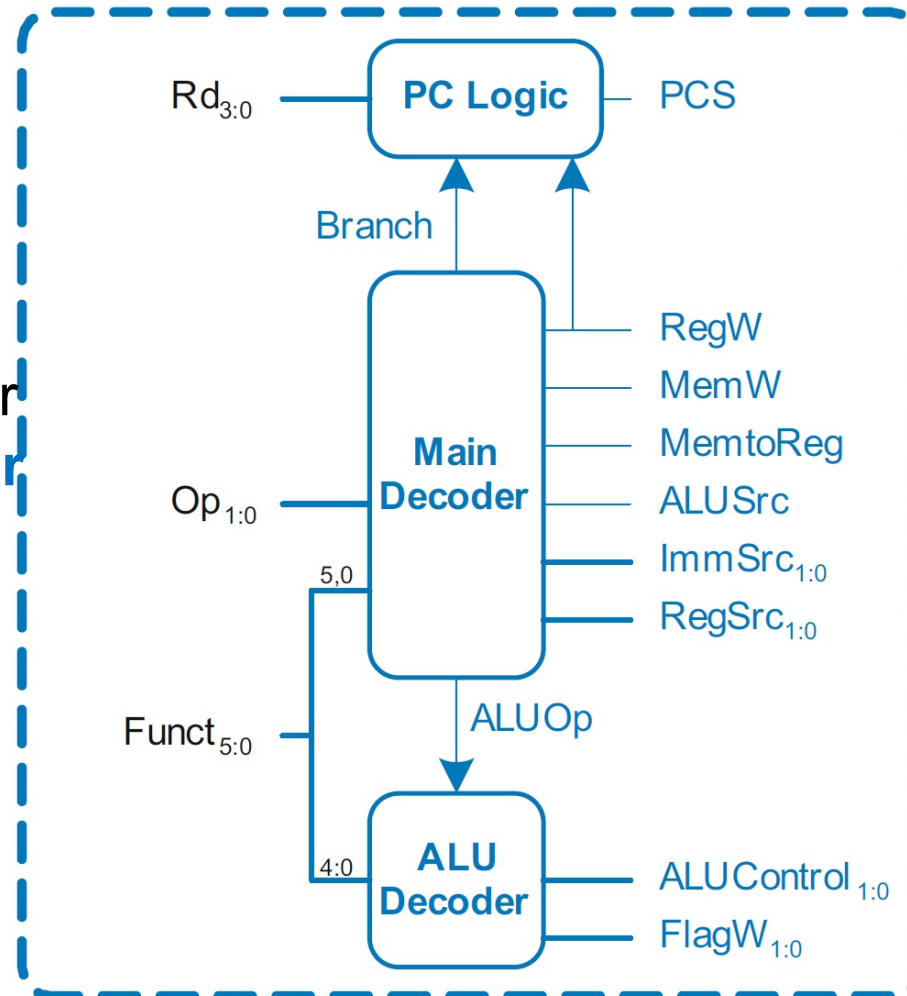
Op	Funct ₅	Funct ₀	Type	Branch	MemtoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1
00	1	X	DP Imm	0	0	0	1	00	1	X0	1
01	X	0	STR	0	X	1	1	01	0	10	0
01	X	1	LDR	0	1	0	1	01	1	X0	0
11	X	X	B	1	0	0	1	10	0	X1	0



Single-Cycle Control: Decoder

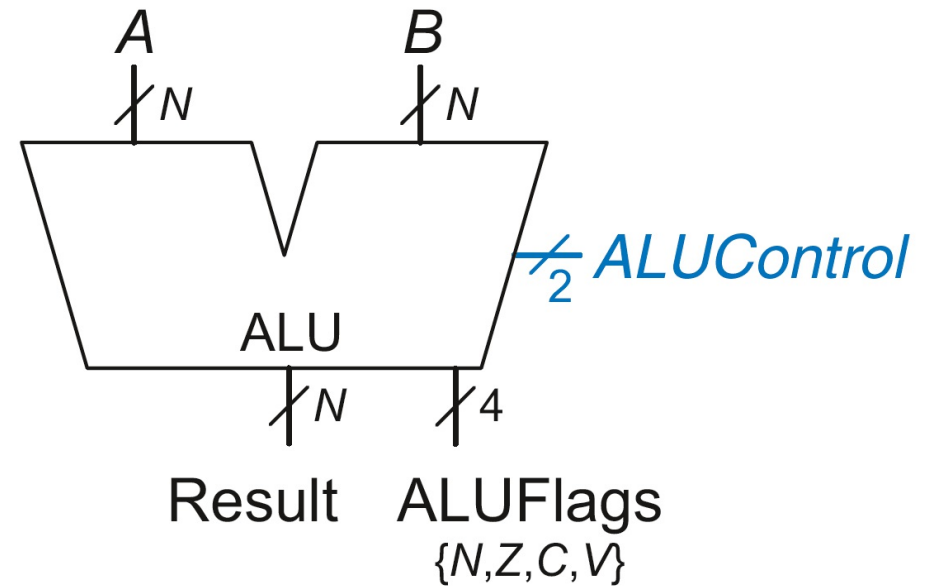
Submodules:

- Main Decoder
- **ALU Decoder**
- PC Logic

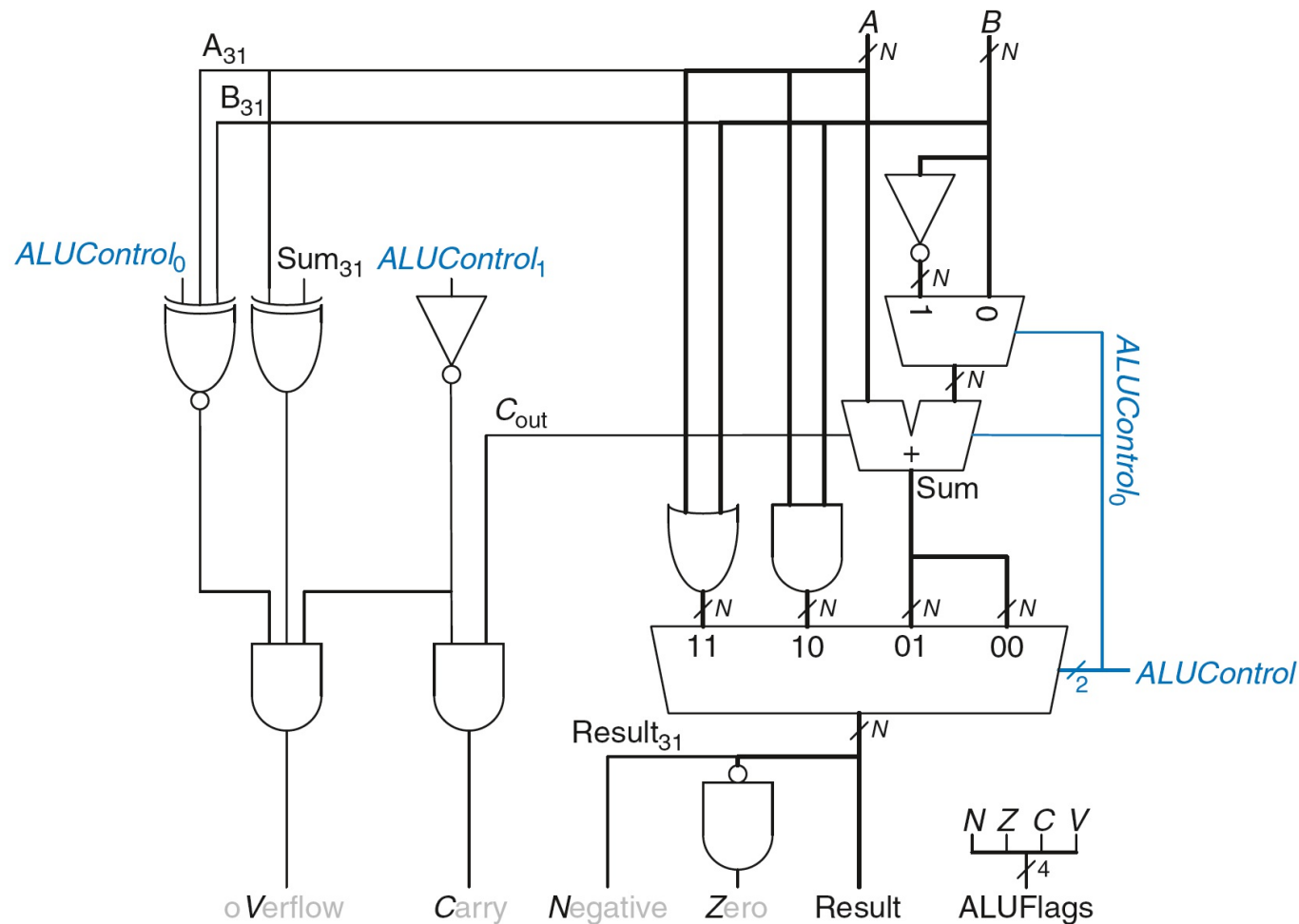


Review: ALU

ALUControl _{1:0}	Function
00	Add
01	Subtract
10	AND
11	OR



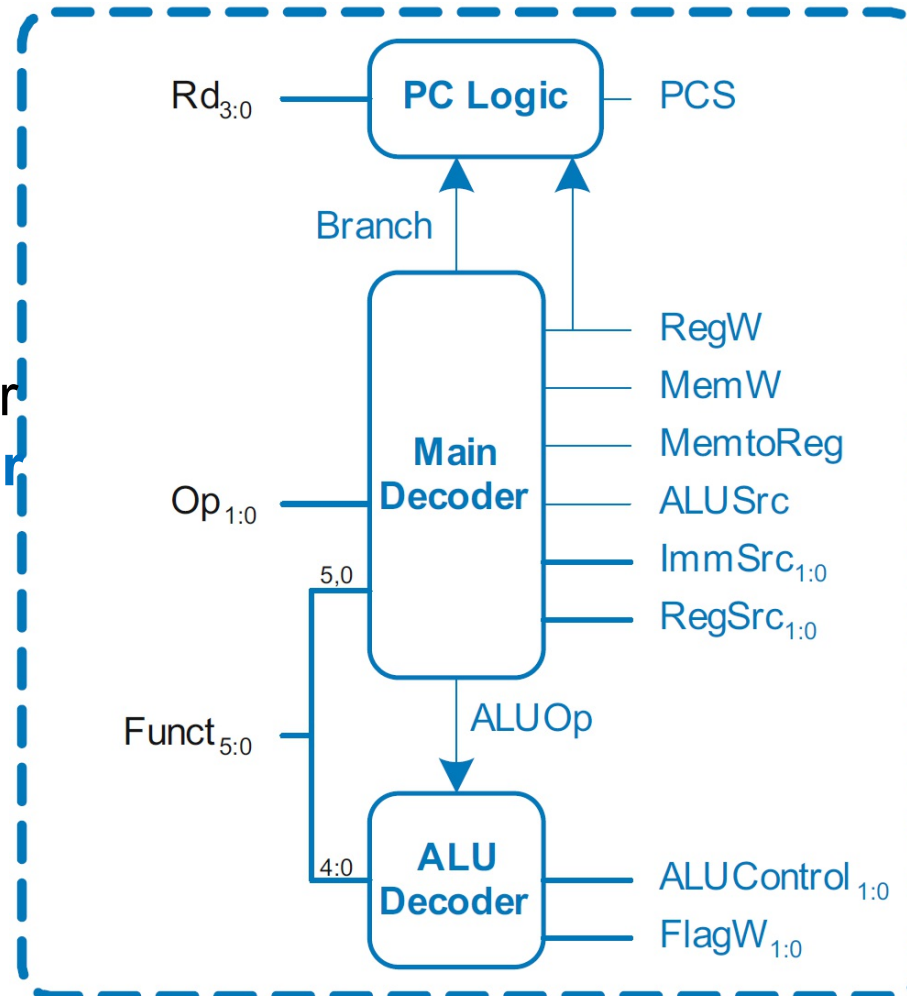
Review: ALU



Single-Cycle Control: Decoder

Submodules:

- Main Decoder
- **ALU Decoder**
- PC Logic



Control Unit: ALU Decoder

ALUOp	Func _{4:1} (cmd)	Func ₀ (S)	Type	ALUControl _{1:0}	FlagW _{1:0}
0	X	X	Not DP	00	00
1	0100	0	ADD	00	00
		1			11
	0010	0	SUB	01	00
		1			11
	0000	0	AND	10	00
		1			10
	1100	0	ORR	11	00
		1			10

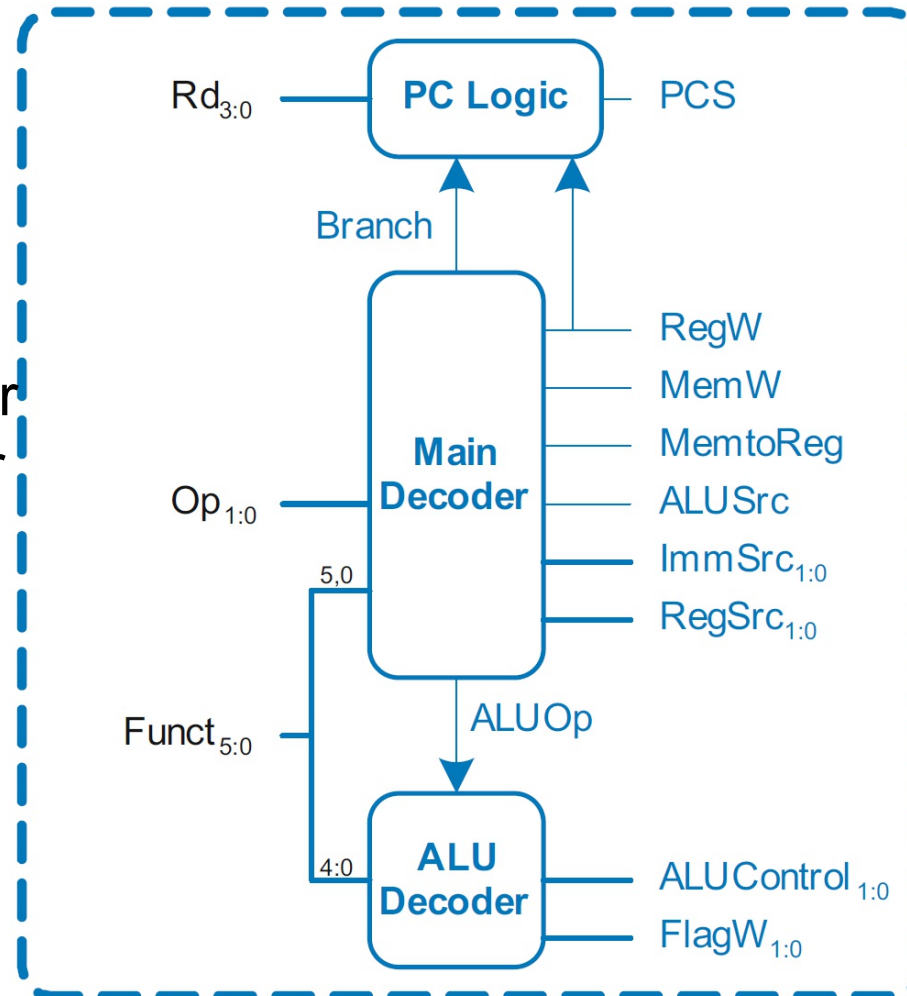
- **FlagW₁** = 1: NZ (*Flags*_{3:2}) should be saved
- **FlagW₀** = 1: CV (*Flags*_{1:0}) should be saved



Single-Cycle Control: Decoder

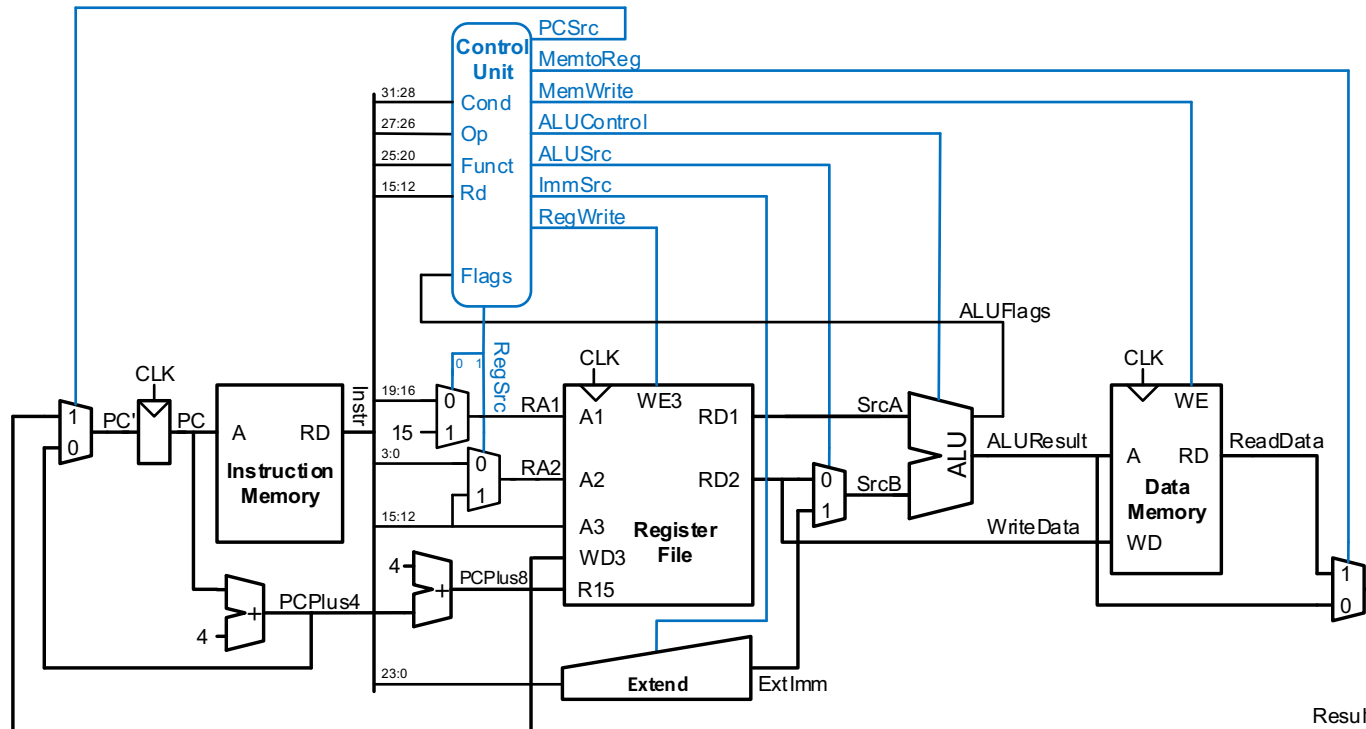
Submodules:

- Main Decoder
- ALU Decoder
- **PC Logic**



Single-Cycle Control: PC Logic

$PCS = 1$ if PC is written by an instruction or branch (B): $PCS = ((Rd == 15) \& RegW) \mid Branch$

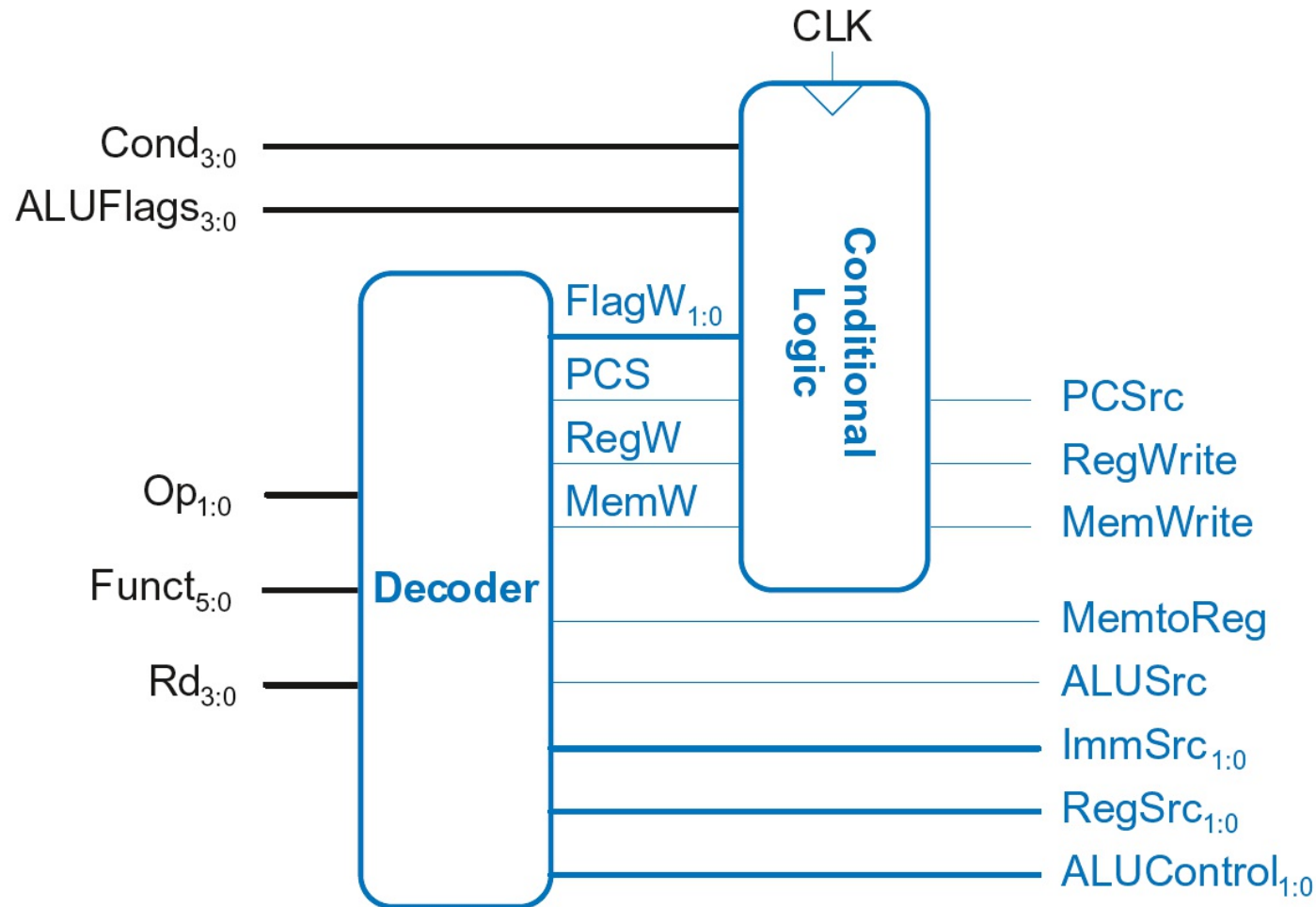


If instruction is executed: $PCSrc = PCS$

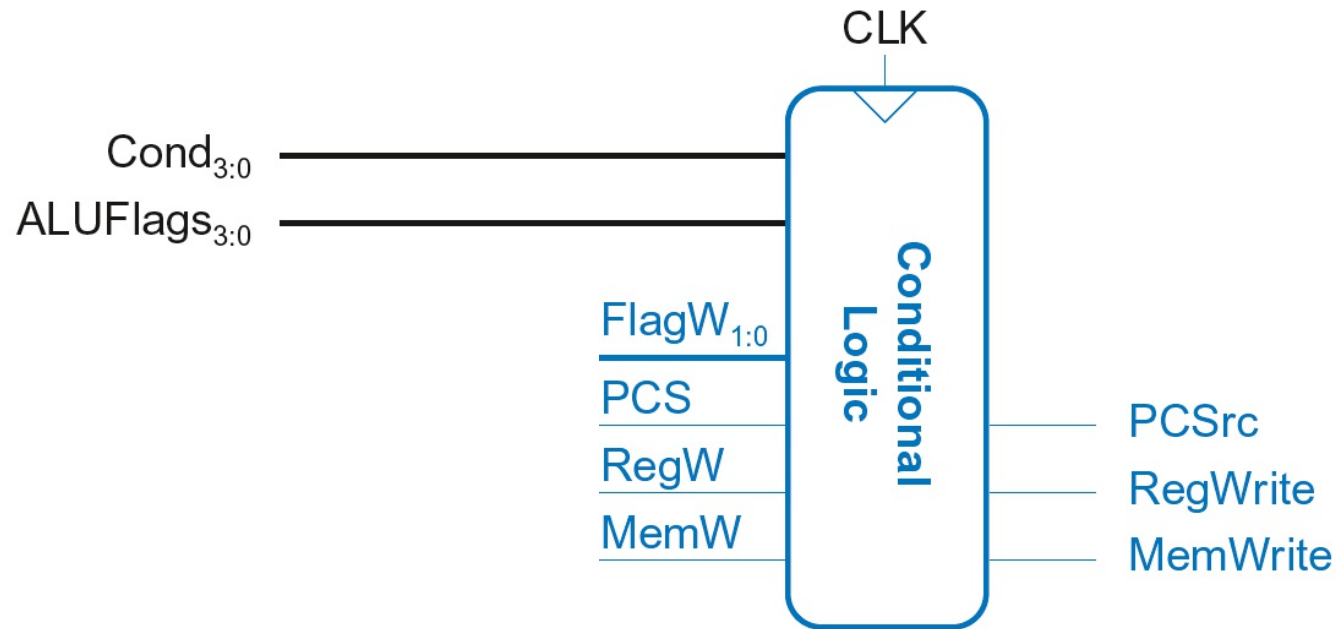
Else $PCSrc = 0$ (i.e., $PC = PC + 4$)



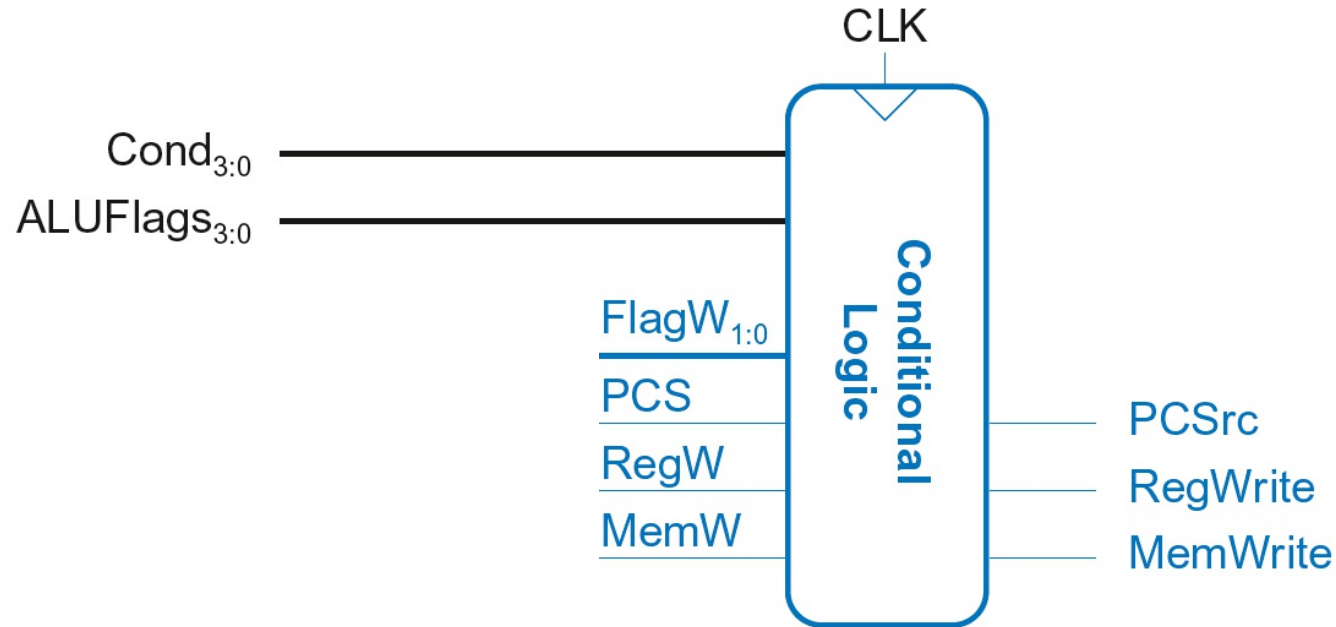
Single-Cycle Control



Single-Cycle Control: Cond. Logic



Conditional Logic

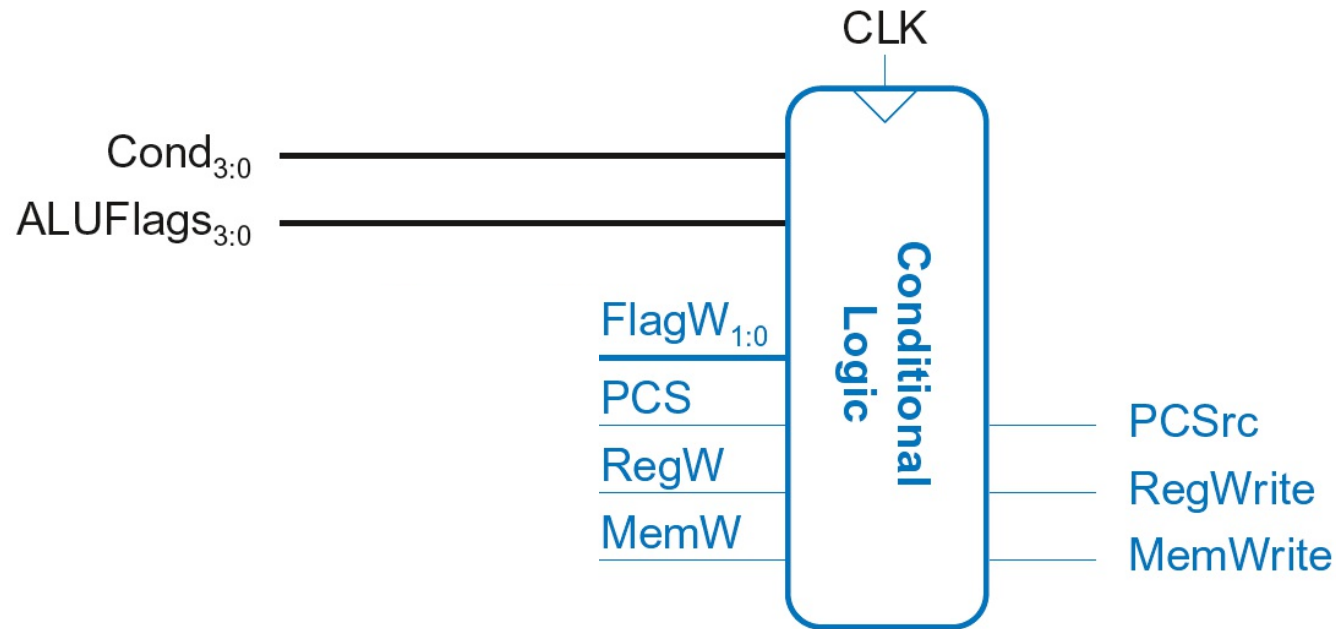


Function:

1. Check if instruction should execute (if not, force PCSrc, RegWrite, and MemWrite to 0)
2. Possibly update Status Register (Flags_{3:0})



Conditional Logic

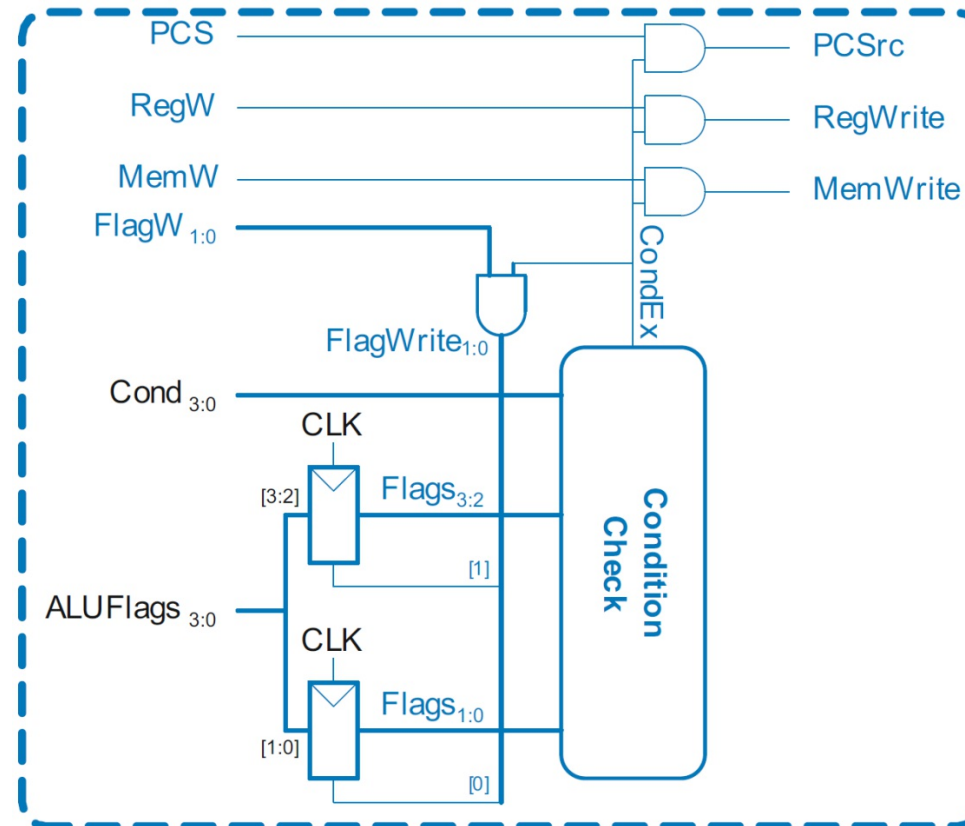


Function:

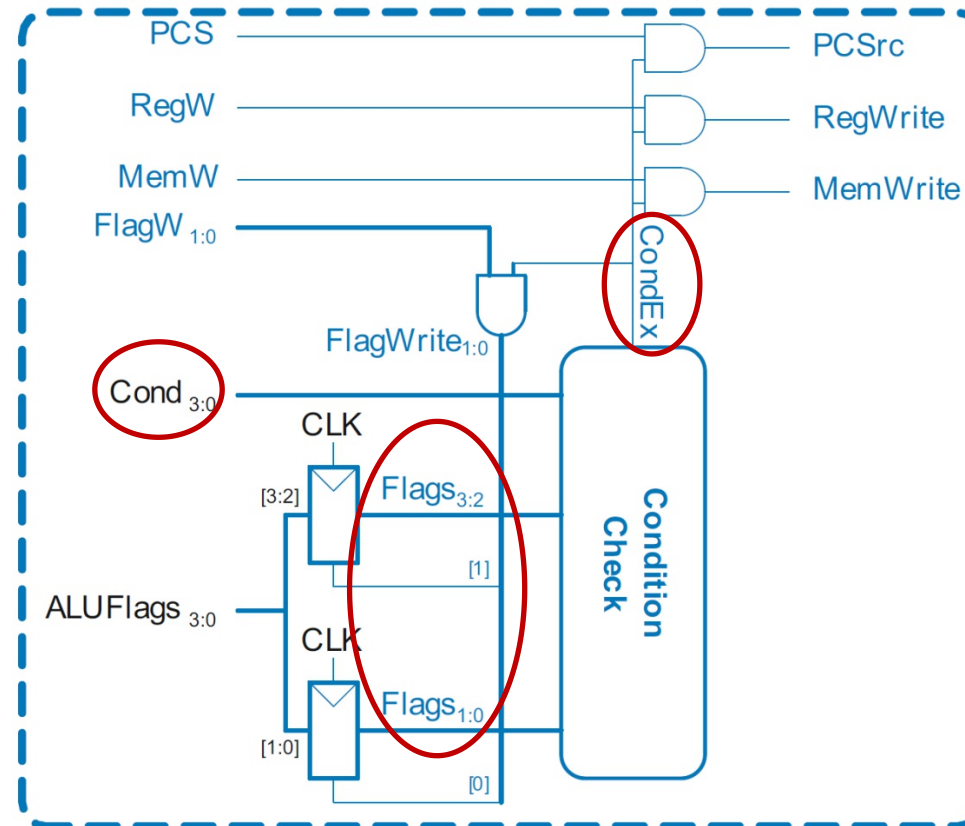
1. Check if instruction should execute (if not, force PCSrc, RegWrite, and MemWrite to 0)
2. Possibly update Status Register (Flags_{3:0})



Single-Cycle Control: Conditional Logic



Conditional Logic: Conditional Execution

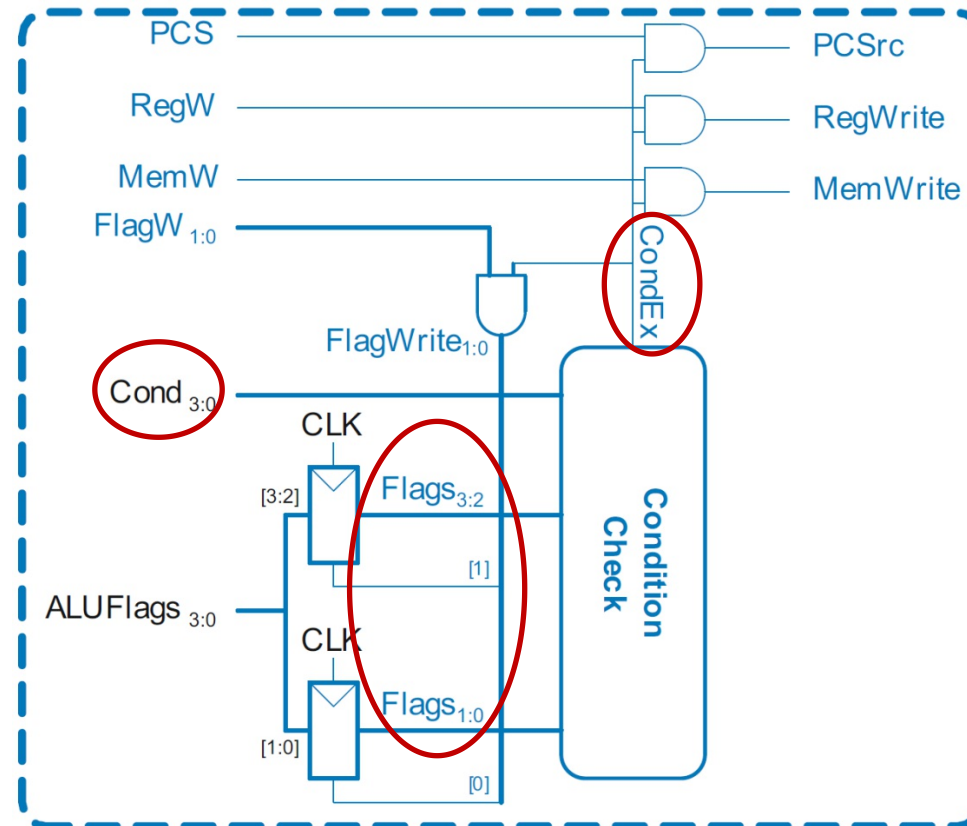


Depending on condition mnemonic (**Cond**_{3:0}) and condition flags (**Flags**_{3:0}) the instruction is executed (**CondEx** = 1)



Conditional Logic: Conditional Execution

Flags_{3:0} is the **status register**

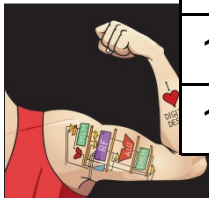


Depending on condition mnemonic (**Cond_{3:0}**) and condition flags (**Flags_{3:0}**) the instruction is executed (**CondEx = 1**)



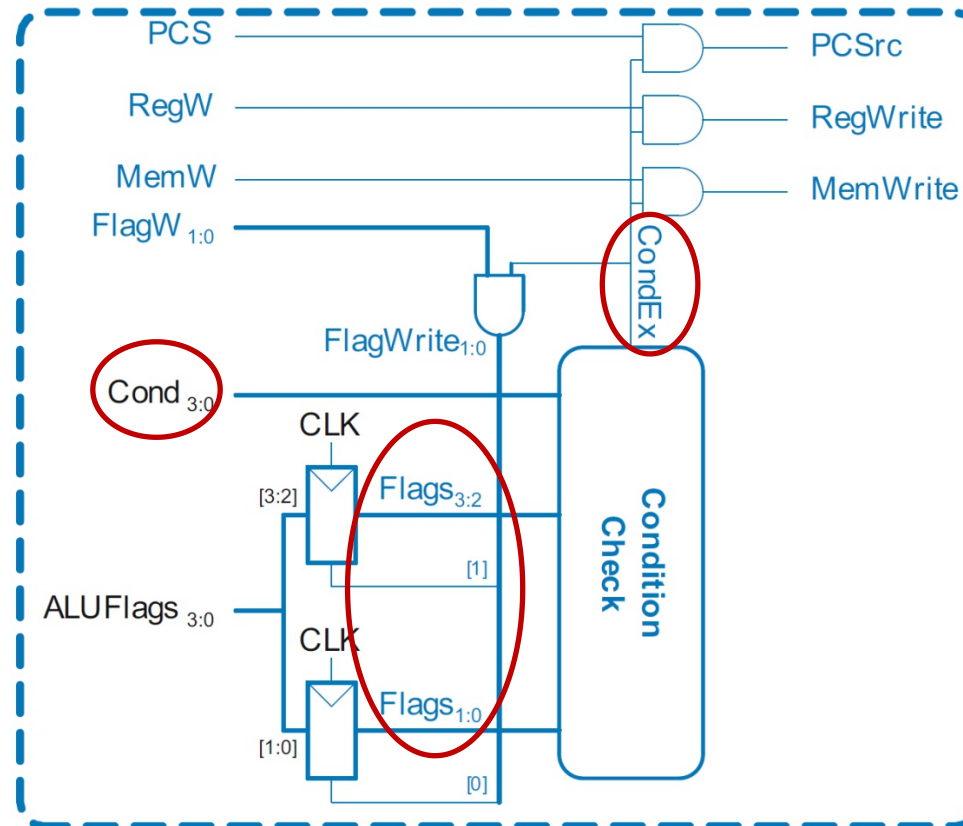
Review: Condition Mnemonics

Cond _{3:} 0	Mnemonic	Name	CondEx
0000	EQ	Equal	Z
0001	NE	Not equal	\bar{Z}
0010	CS / HS	Carry set / Unsigned higher or same	C
0011	CC / LO	Carry clear / Unsigned lower	\bar{C}
0100	MI	Minus / Negative	N
0101	PL	Plus / Positive of zero	\bar{N}
0110	VS	Overflow / Overflow set	V
0111	VC	No overflow / Overflow clear	\bar{V}
1000	HI	Unsigned higher	$\bar{Z}C$
1001	LS	Unsigned lower or same	$Z OR \bar{C}$
1010	GE	Signed greater than or equal	$\bar{N} \oplus \bar{V}$
1011	LT	Signed less than	$N \oplus V$
1100	GT	Signed greater than	$\bar{Z}(\bar{N} \oplus \bar{V})$
1101	LE	Signed less than or equal	$Z OR (N \oplus V)$
1110	AL (or none)	Always / unconditional	ignored



Conditional Logic: Conditional Execution

Flags_{3:0} =
NZCV



Example:

AND R1, R2, R3

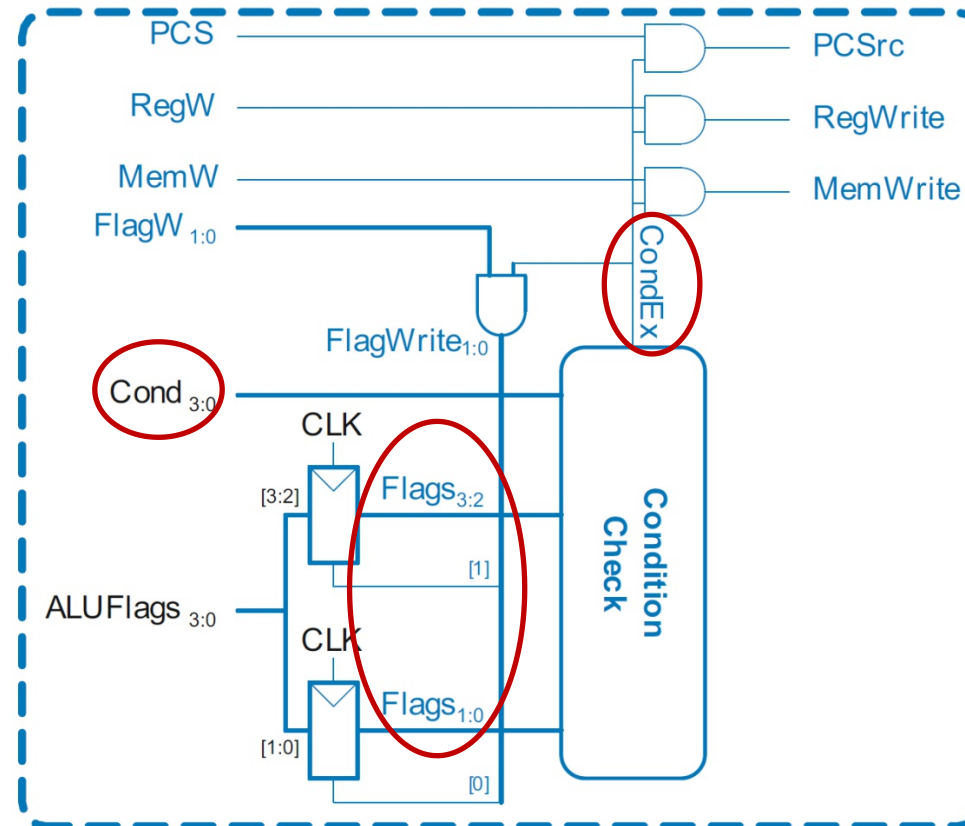
Cond_{3:0} = 1110 (unconditional)

=> **CondEx = 1**



Conditional Logic: Conditional Execution

Flags_{3:0} =
NZCV



Example:

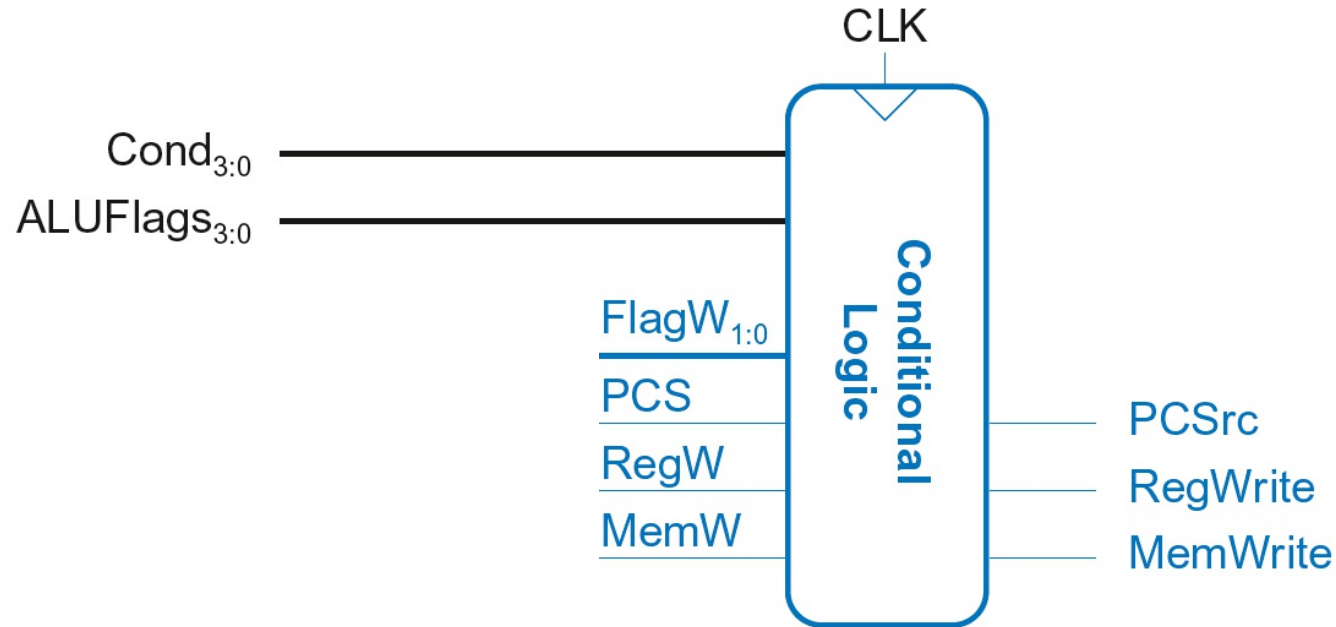
EOREQ R5, R6, R7

Cond_{3:0} = 0000 (EQ): if **Flags_{3:2} = 0100** =>

CondEx = 1



Conditional Logic



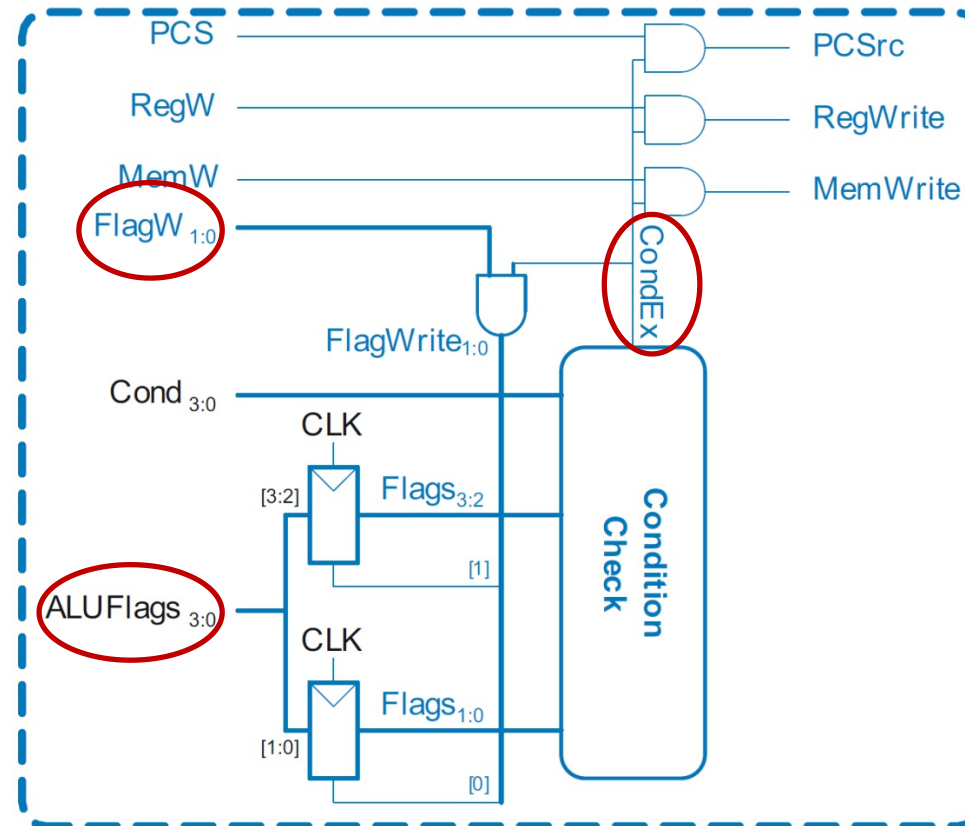
Function:

1. Check if instruction should execute (if not, force $PCSrc$, $RegWrite$, and $MemWrite$ to 0)
2. **Possibly update Status Register ($Flags_{3:0}$)**



Conditional Logic: Update (Set) Flags

Flags_{3:0} =
NZCV

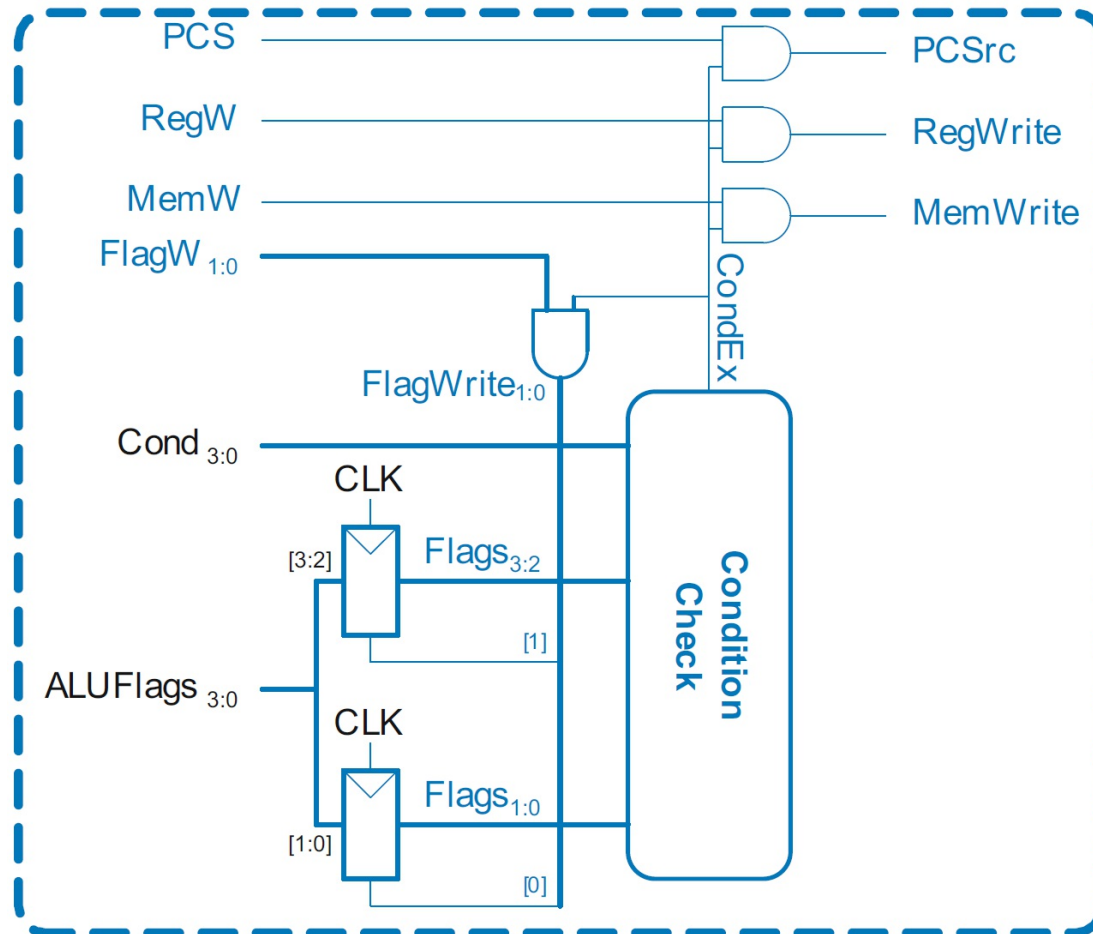


Flags_{3:0} **updated** (with ALUFlags_{3:0}) if:

- **FlagW** is 1 (i.e., the instruction's S-bit is 1)
AND
- **CondEx** is 1 (the instruction should be executed)



Conditional Logic: Update (Set) Flags



Recall:

- ADD, SUB
update **all** Flags
- AND, OR update
NZ only
- So Flags status
register has two
write enables:
FlagW_{1:0}



Review: ALU Decoder

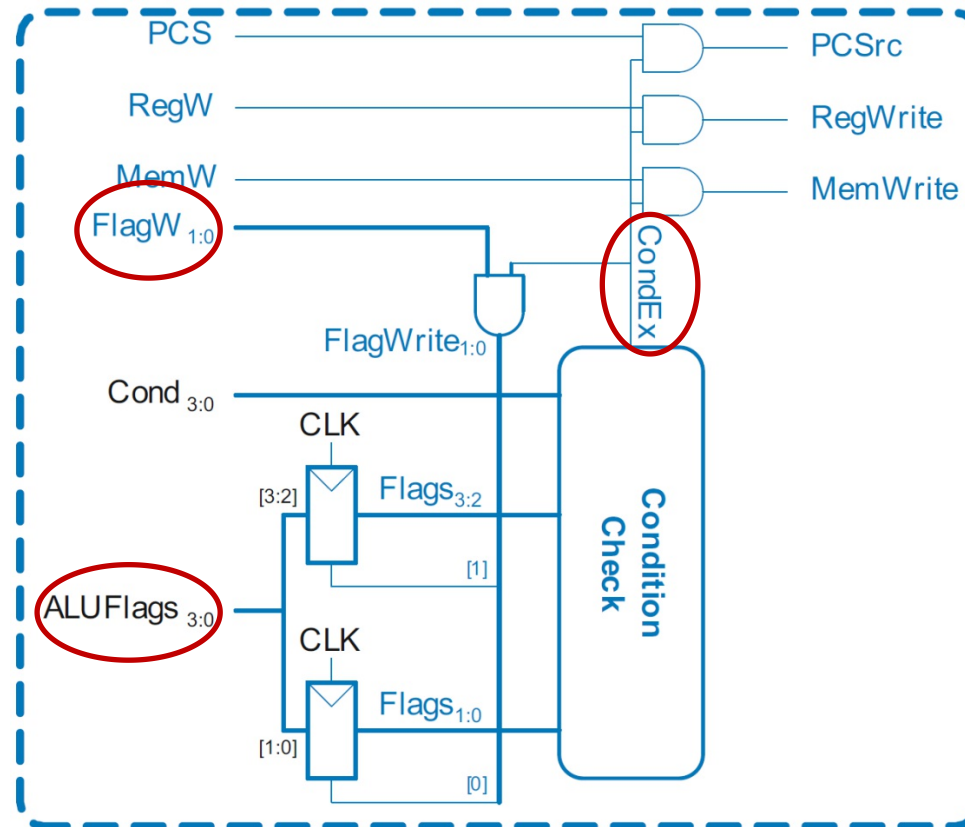
ALUOp	Func _{4:1} (cmd)	Func ₀ (S)	Type	ALUControl _{1:0}	FlagW _{1:0}
0	X	X	Not DP	00	00
1	0100	0	ADD	00	00
		1			11
	0010	0	SUB	01	00
		1			11
	0000	0	AND	10	00
		1			10
	1100	0	ORR	11	00
		1			10

- **FlagW₁** = 1: NZ (*Flags*_{3:2}) should be saved
- **FlagW₀** = 1: CV (*Flags*_{1:0}) should be saved



Conditional Logic: Update (Set) Flags

All Flags
updated



Example: SUBS R5, R6, R7

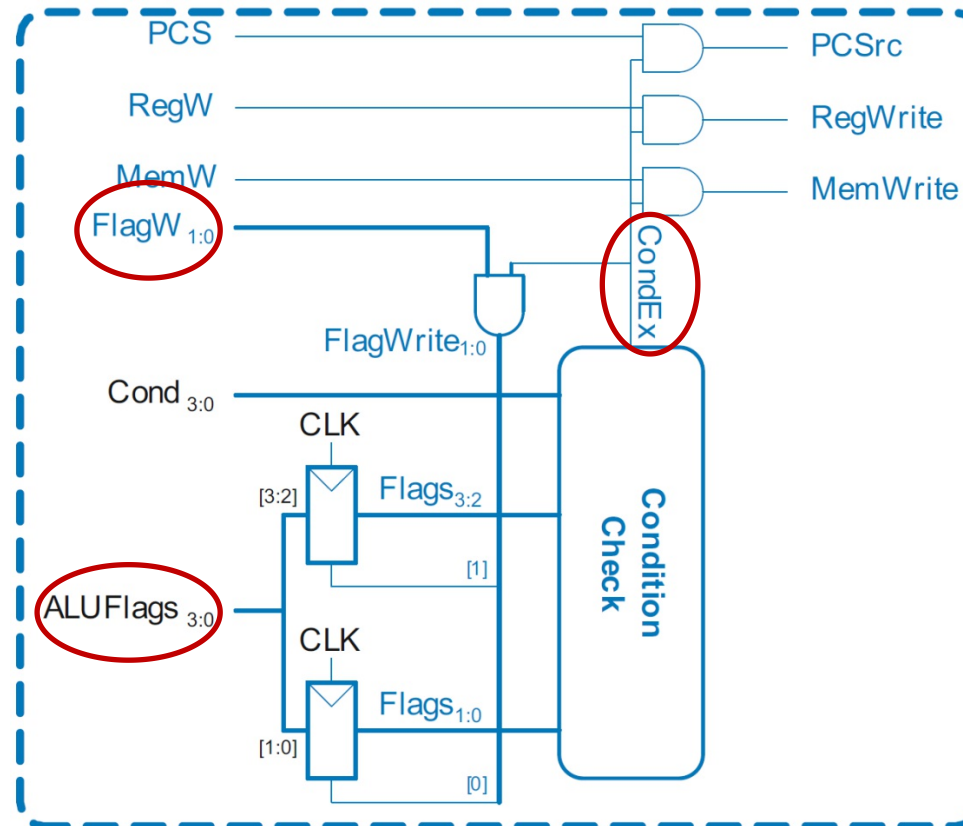
FlagW_{1:0} = 11 AND CondEx = 1 (unconditional) => FlagWrite_{1:0} = 11



Conditional Logic: Update (Set) Flags

Flags_{3:0} =
NZCV

- Only **Flags_{3:2}** updated
- i.e., only **NZ** Flags updated



Example: ANDS R7, R1, R3

FlagW_{1:0} = 10 AND **CondEx** = 1 (unconditional) => **FlagWrite_{1:0}** = 10



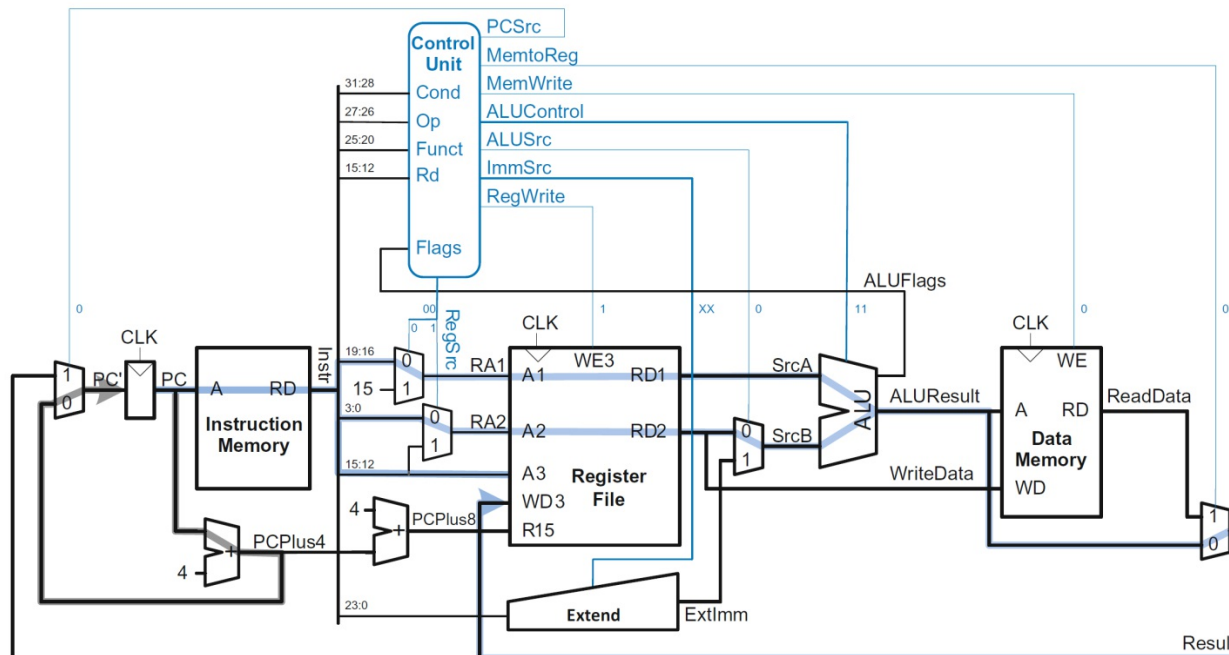
Example: ORR

ALUOp	
RegSrc	
RegW	
ImmSrc	
ALUSrc	
MemW	
MemtoReg	
Branch	
Type	
Funct ₀	
Funct _s	
Op	

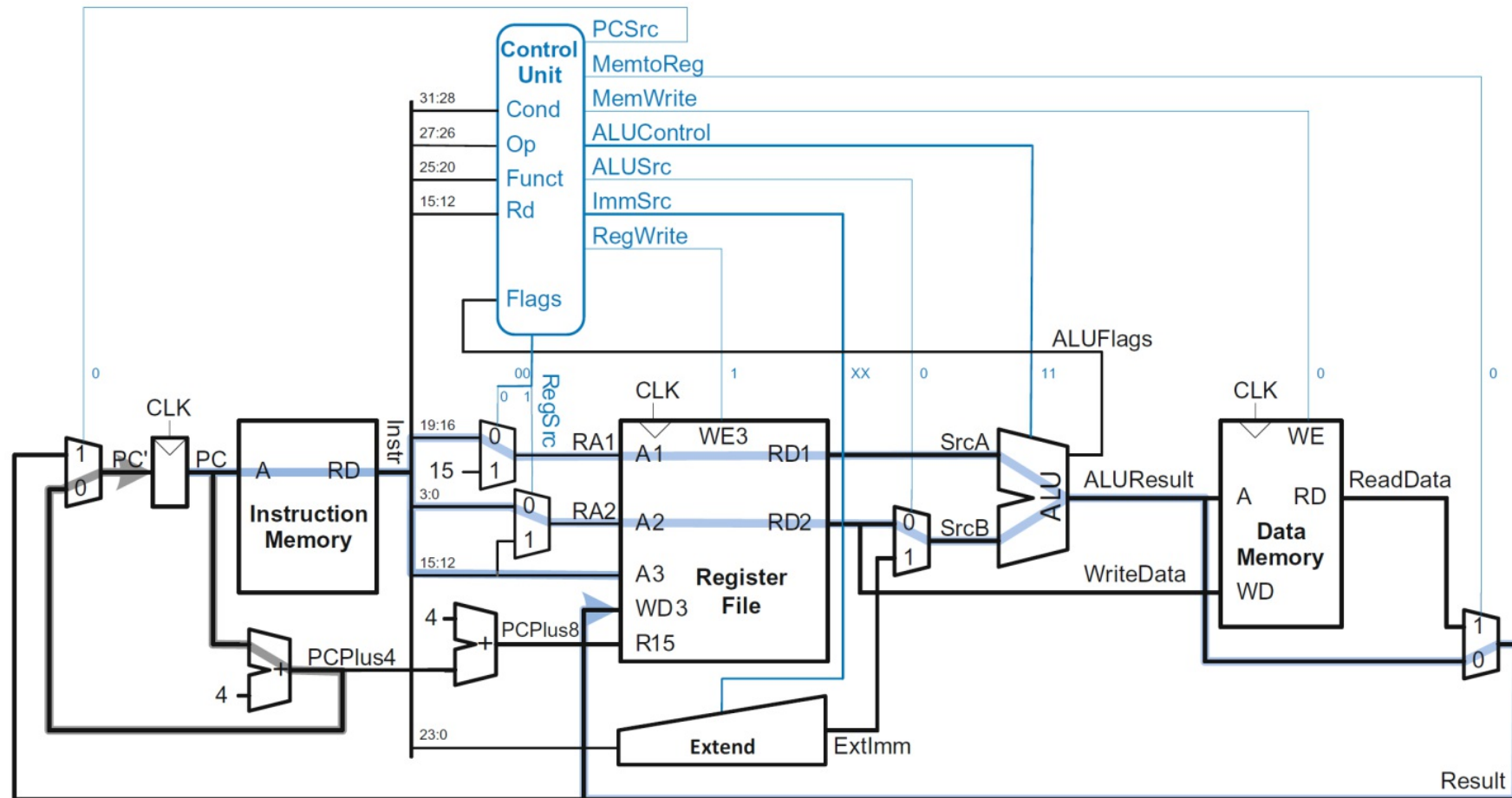


Example: ORR

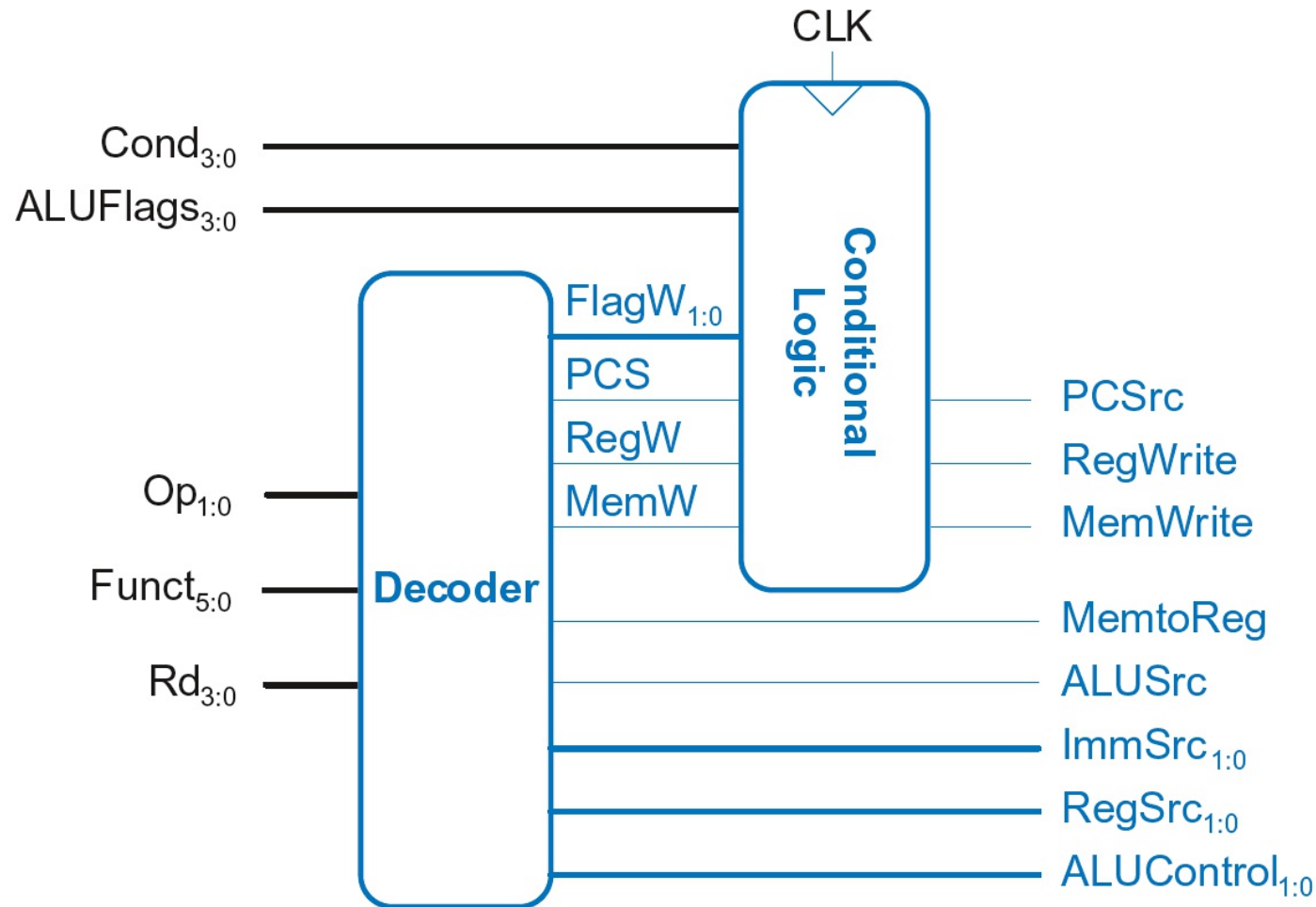
Op	Funct _s	Funct ₀	Type	Branch	MementoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1



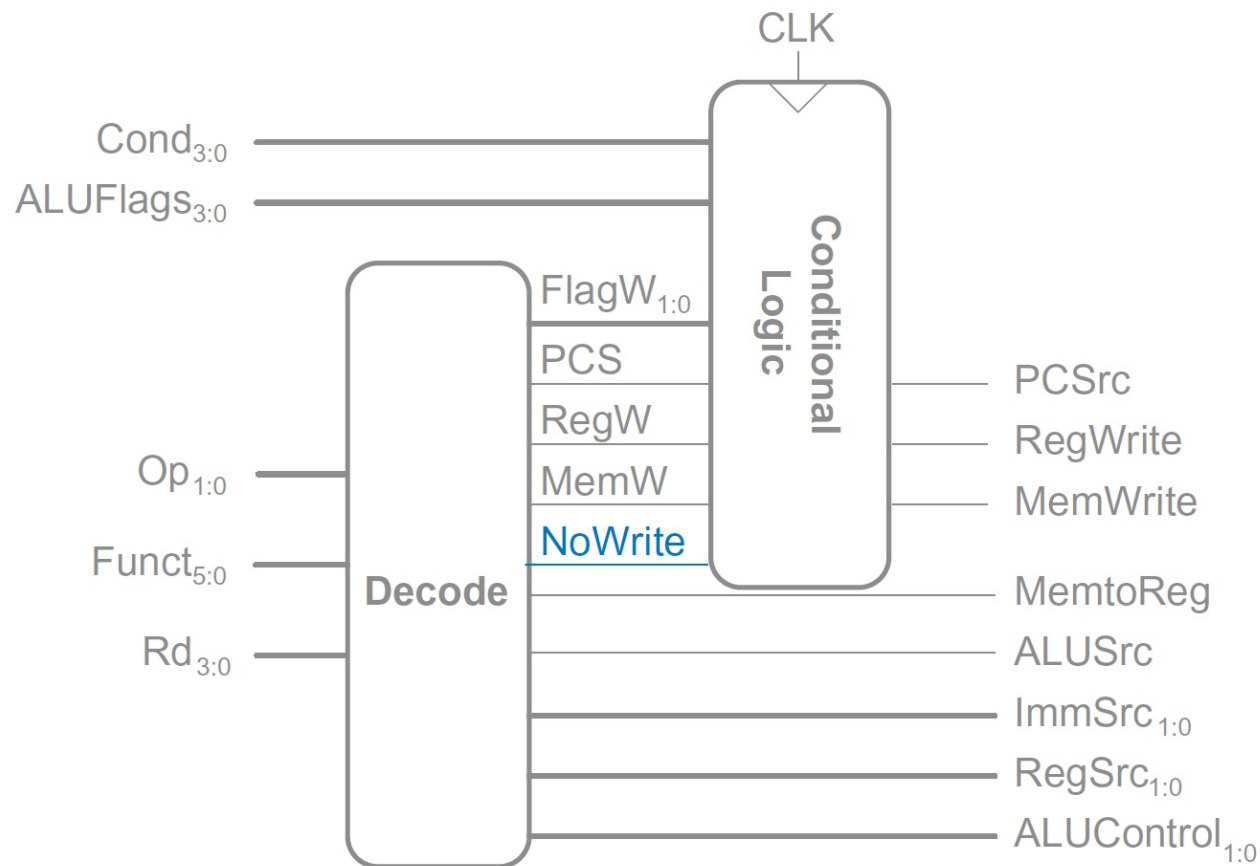
Example: ORR



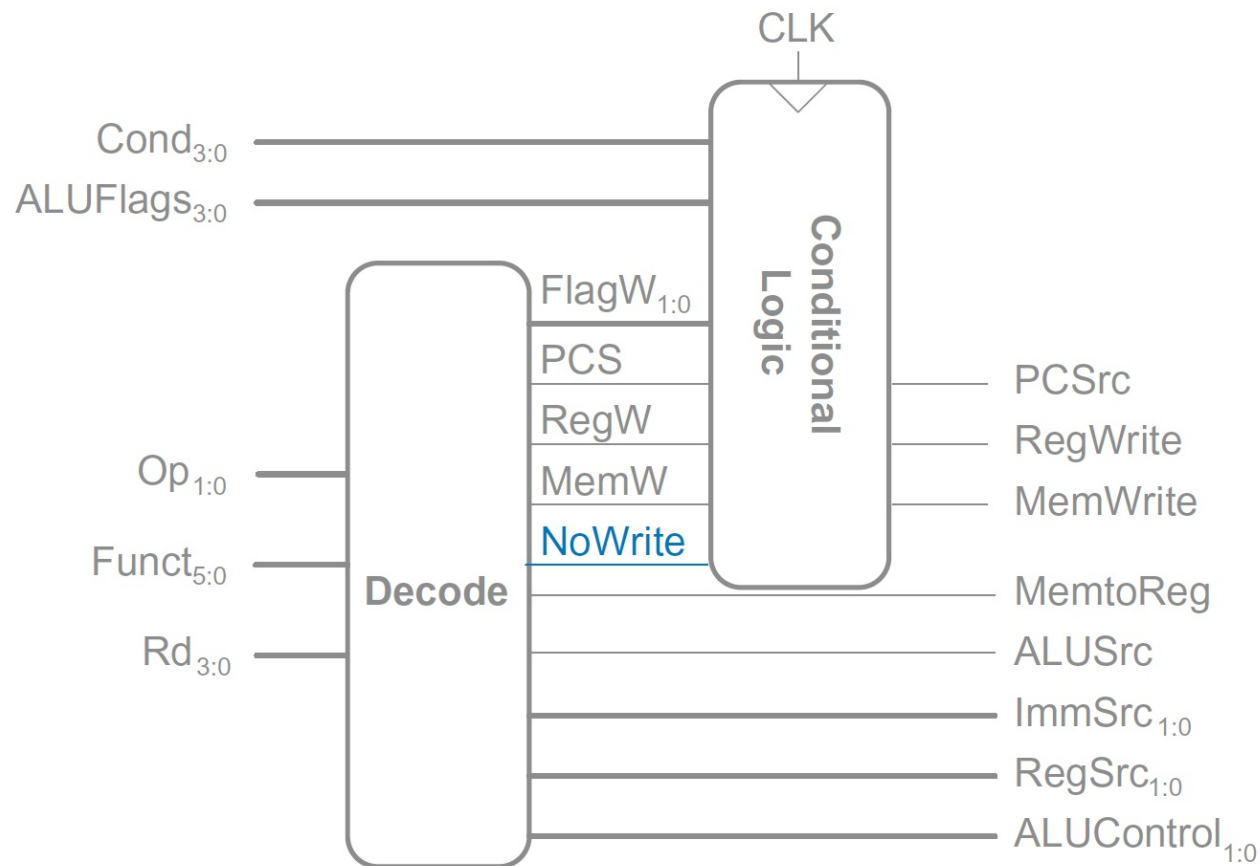
Extended Functionality: CMP



Extended Functionality: CMP



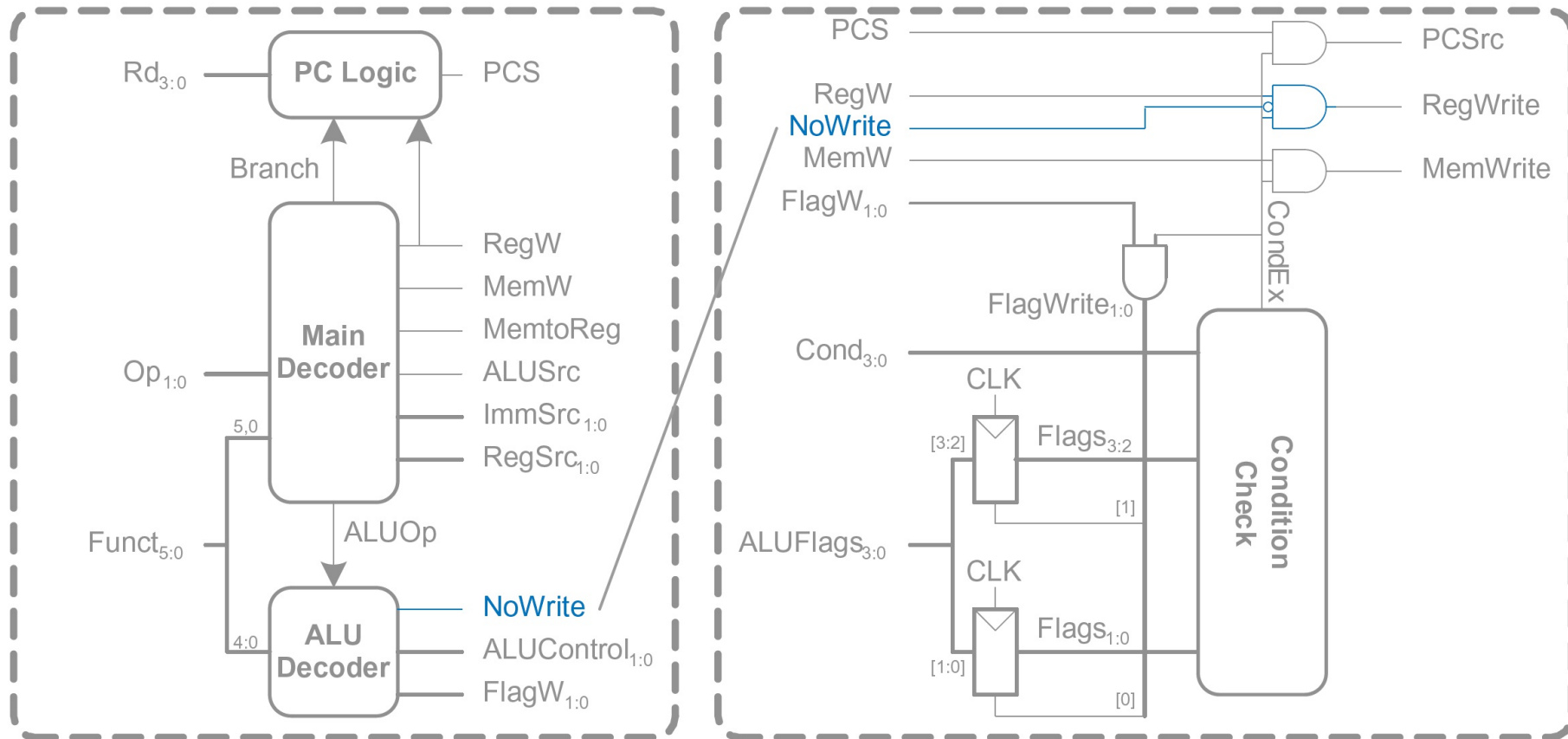
Extended Functionality: CMP



No change to datapath



Extended Functionality: CMP

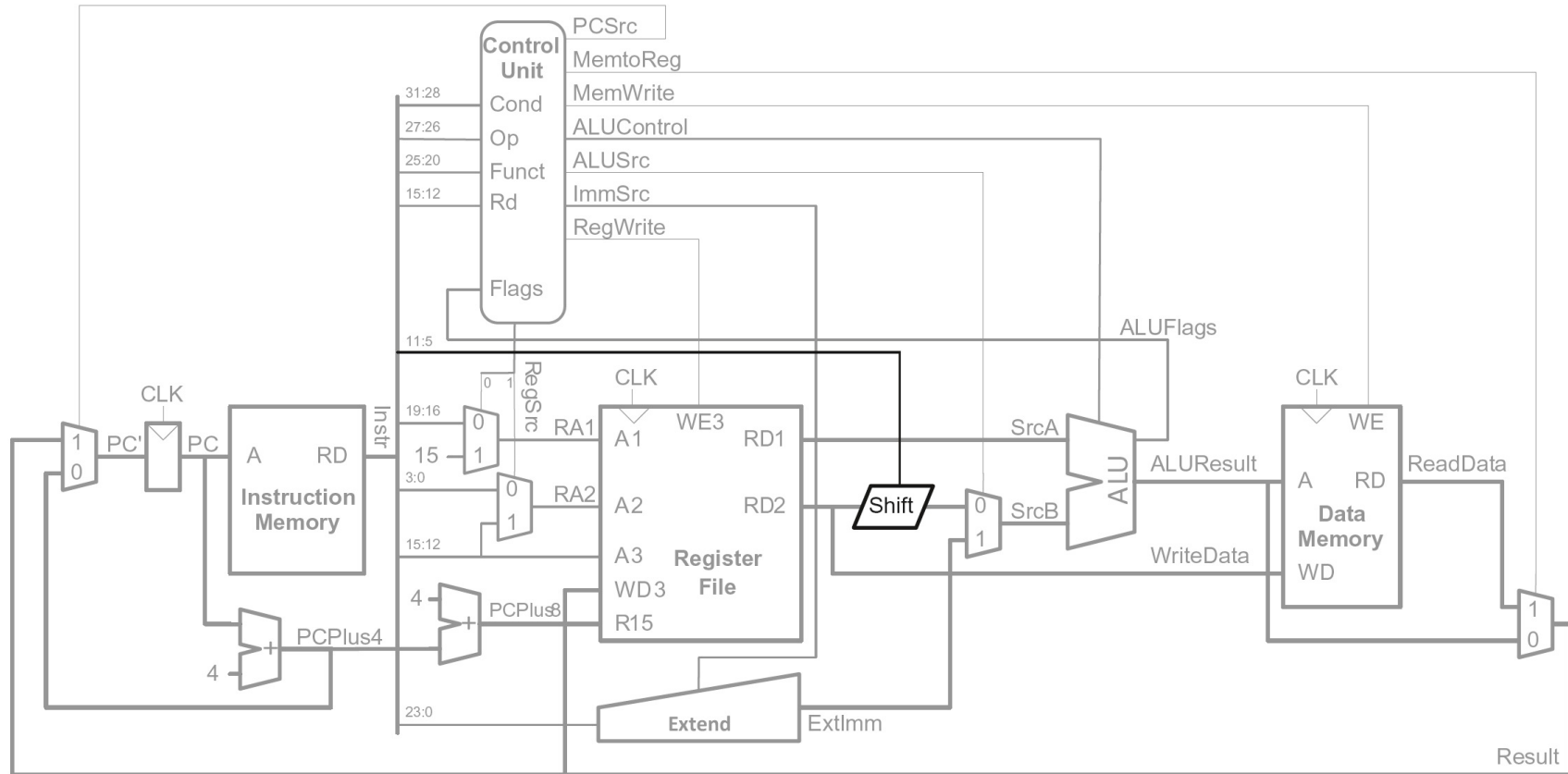


Extended Functionality: CMP

ALUOp	Funct _{4:1} (cmd)	Funct ₀ (S)	Type	ALUControl _{1:0}	FlagW _{1:0}	NoWrite
0	X	X	Not DP	00	00	0
1	0100	0	ADD	00	00	0
		1			11	0
	0010	0	SUB	01	00	0
		1			11	0
	0000	0	AND	10	00	0
		1			10	0
	1100	0	ORR	11	00	0
		1			10	0
	1010	1	CMP	01	11	1



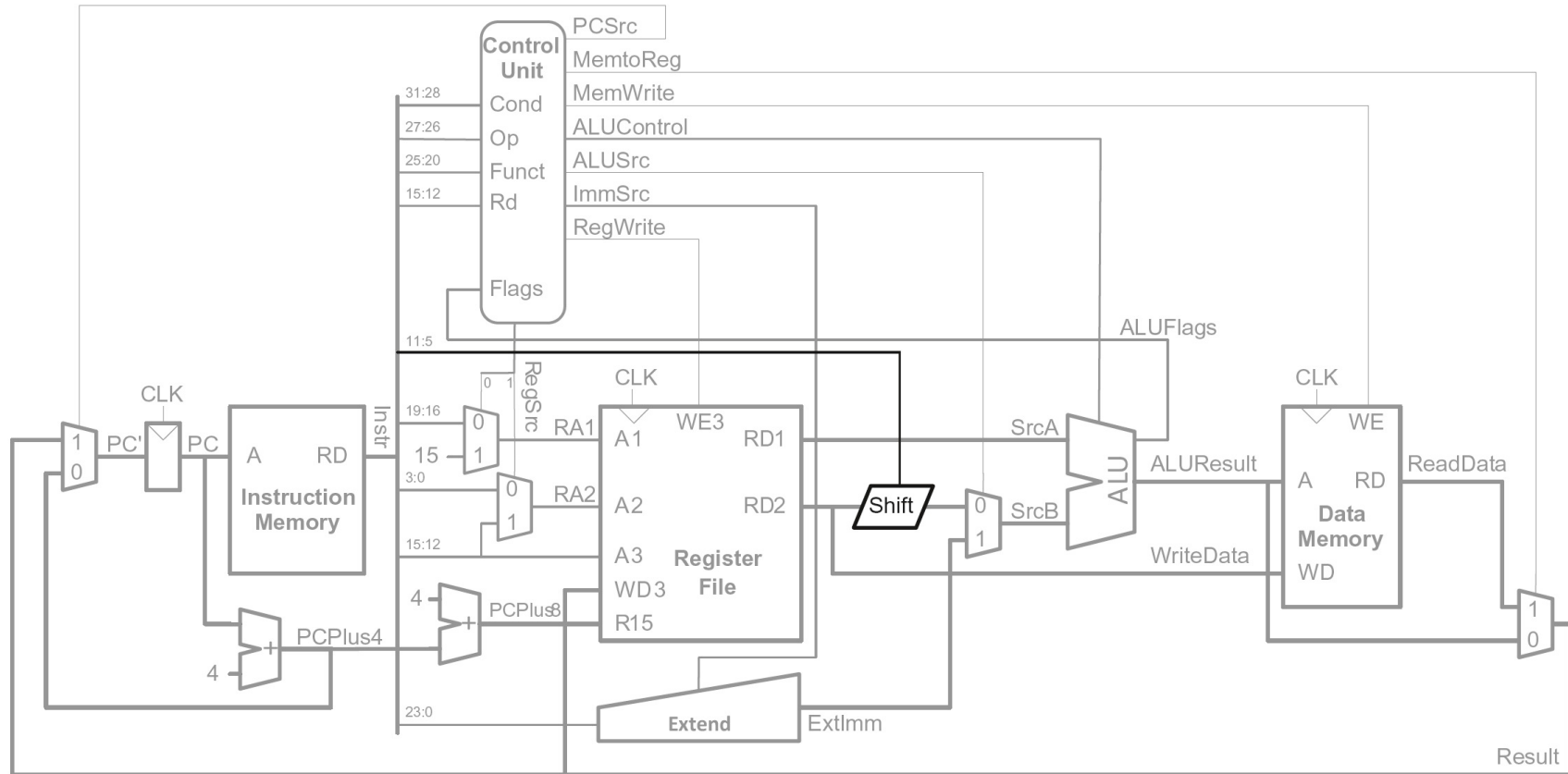
Extended Functionality: Shifted Register



	31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
ADD R7, R2, R12, LSR #5	14	0	0	4	0	2	7	5	0	1	12
	cond	op	l	cmd	S	rn	rd	shamt5	sh		rm



Extended Functionality: Shifted Register



No change to controller



Review: Processor Performance

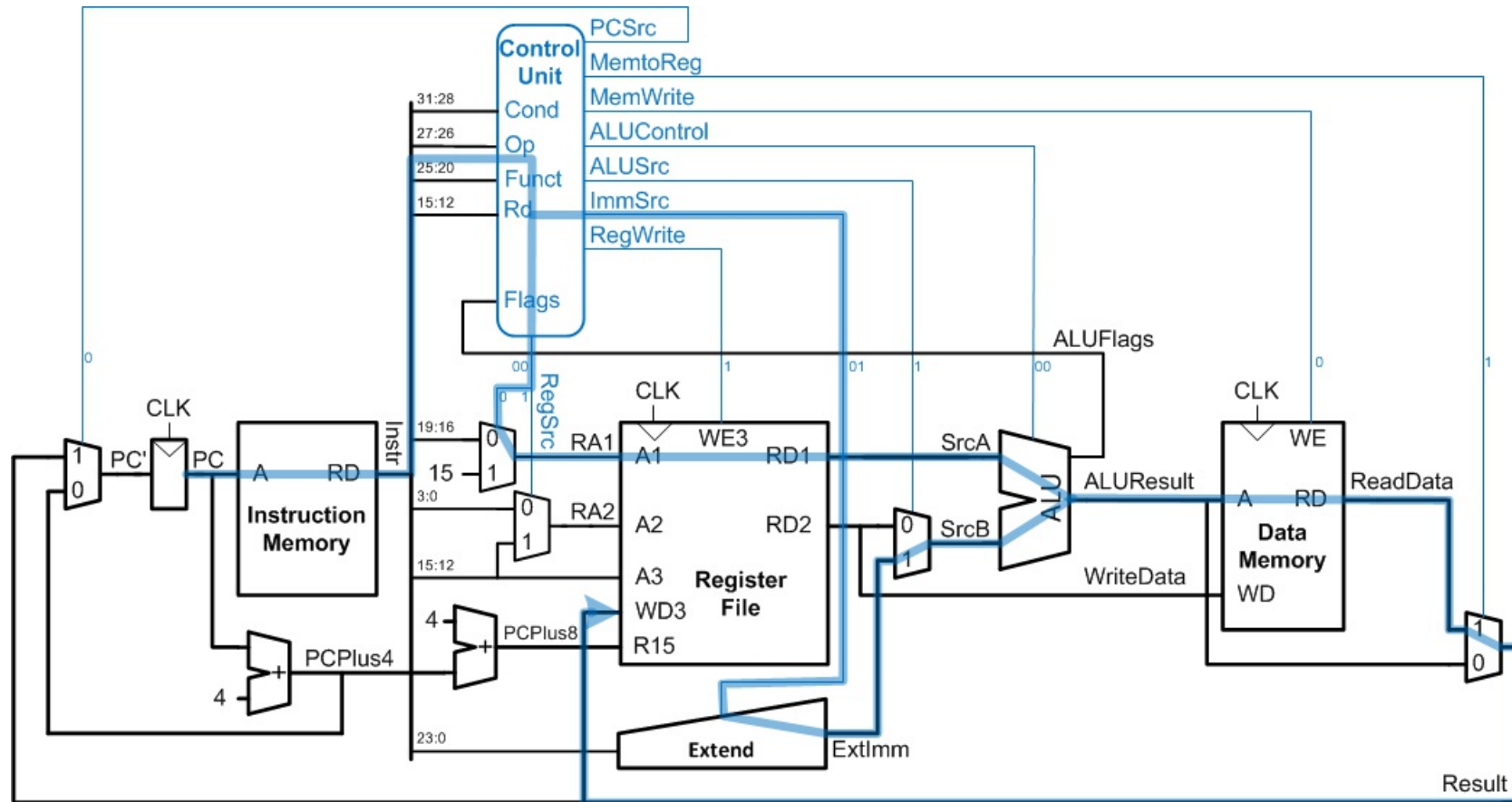
Program Execution Time

$$= (\# \text{instructions})(\text{cycles/instruction})(\text{seconds/cycle})$$

$$= \# \text{ instructions} \times \text{CPI} \times T_C$$



Single-Cycle Performance



T_C limited by critical path (LDR)



Single-Cycle Performance

- **Single-cycle critical path:**

$$T_{c1} = t_{pcq_PC} + t_{mem} + t_{dec} + \max[t_{mux} + t_{RFread}, t_{sext} + t_{mux}] + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$$

- **Typically, limiting paths are:**

- memory, ALU, register file

- $T_{c1} = t_{pcq_PC} + 2t_{mem} + t_{dec} + t_{RFread} + t_{ALU} + 2t_{mux} + t_{RFsetup}$



Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Decoder	t_{dec}	70
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

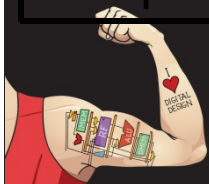
$T_{cl} = ?$



Single-Cycle Performance Example

Element	Parameter	Delay (ps)
Register clock-to-Q	t_{pcq_PC}	40
Register setup	t_{setup}	50
Multiplexer	t_{mux}	25
ALU	t_{ALU}	120
Decoder	t_{dec}	70
Memory read	t_{mem}	200
Register file read	t_{RFread}	100
Register file setup	$t_{RFsetup}$	60

$$\begin{aligned}
 T_{cl} &= t_{pcq_PC} + 2t_{mem} + t_{dec} + \\
 &\quad t_{RFread} + t_{ALU} + 2t_{mux} + t_{RFsetup} \\
 &= [50 + 2(200) + 70 + 100 + \\
 &\quad 120 + 2(25) + 60] \text{ ps} \\
 &= \mathbf{840 \text{ ps}}
 \end{aligned}$$



Single-Cycle Performance Example

Program with 100 billion instructions:

$$\begin{aligned}\text{Execution Time} &= \# \text{ instructions} \times \text{CPI} \times T_C \\ &= (100 \times 10^9)(1)(840 \times 10^{-12} \text{ s}) \\ &= \mathbf{84 \text{ seconds}}\end{aligned}$$

