CPE 221: Computer Organization

14 Single Cycle ARM Processor

rahul.bhadani@uah.edu

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

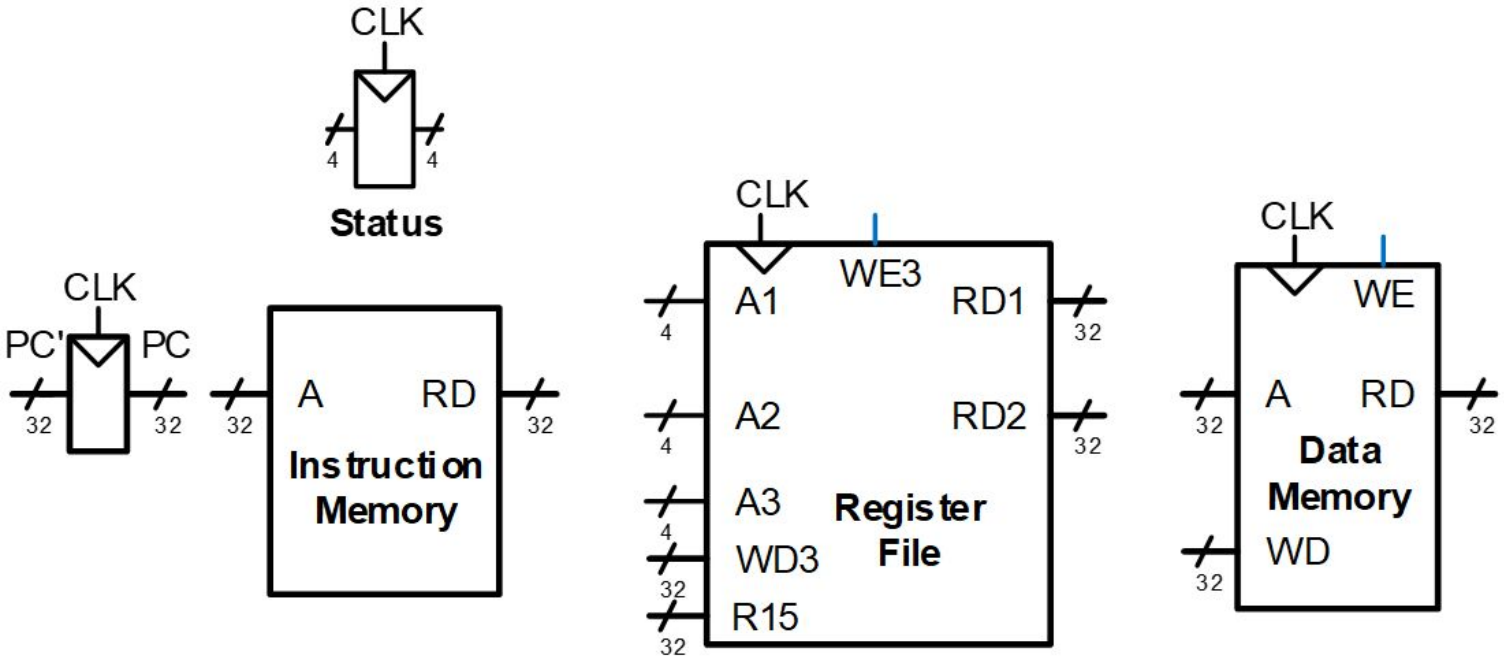
- Consider **subset** of ARM instructions:
  - **Data-processing instructions:**
    - ADD, SUB, AND, ORR
    - with register and immediate Src2, but **no shifts**
  - **Memory instructions:**
    - LDR, STR
    - with **positive immediate offset**
  - **Branch instructions:**
    - B

Architectural State Elements

Determines everything about a processor:

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

ARM Architectural State Elements



Rahul Bhadani

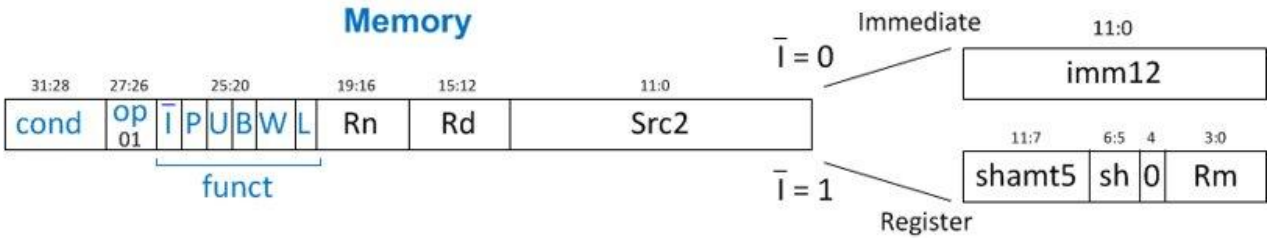
Single-Cycle ARM Processor

- Datapath
- Control

Single-Cycle ARM Processor

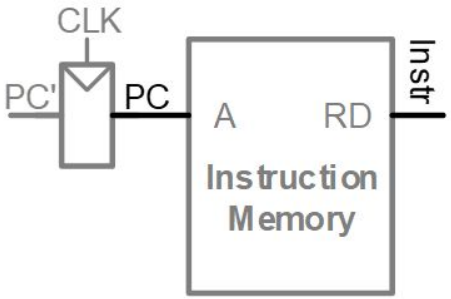
- **Datapath:** start with LDR instruction
- **Example:**

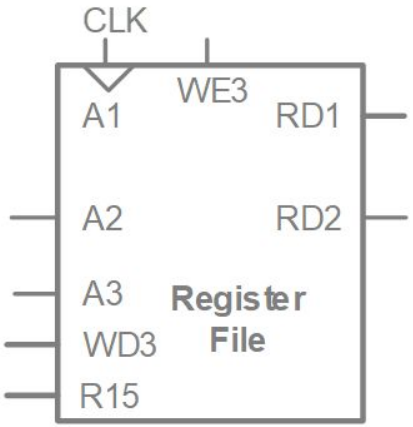
```
LDR R1, [R2, #5]
LDR Rd, [Rn, imm12]
```

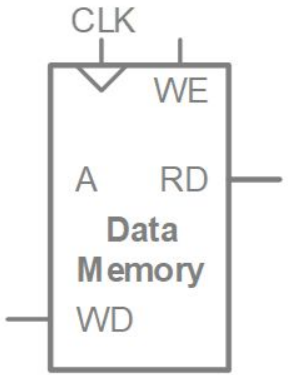


Single-Cycle Datapath: LDR fetch

STEP 1: Fetch instruction

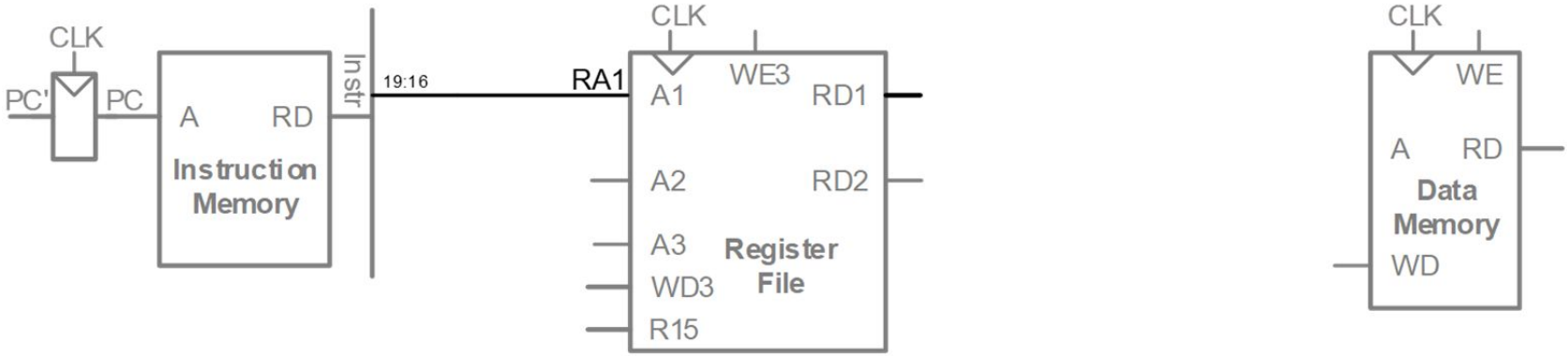




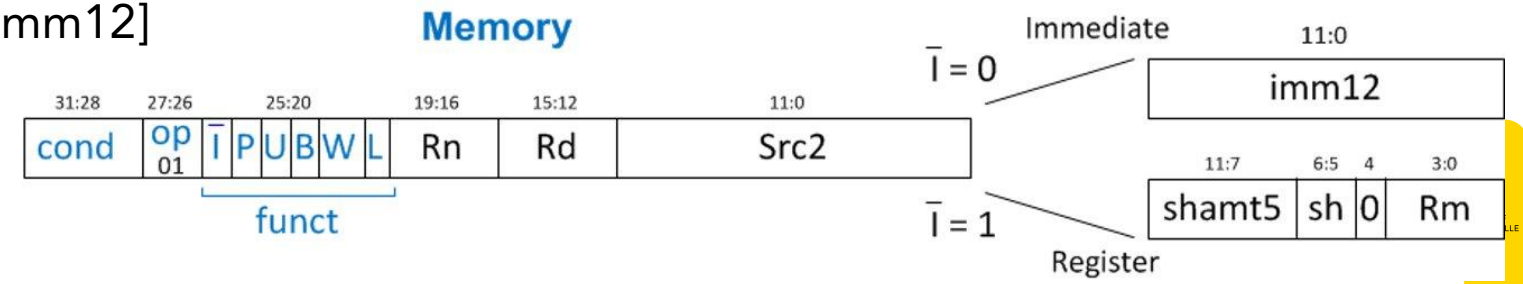


Single-Cycle Datapath: LDR Reg Read

STEP 2: Read source operands from Rn

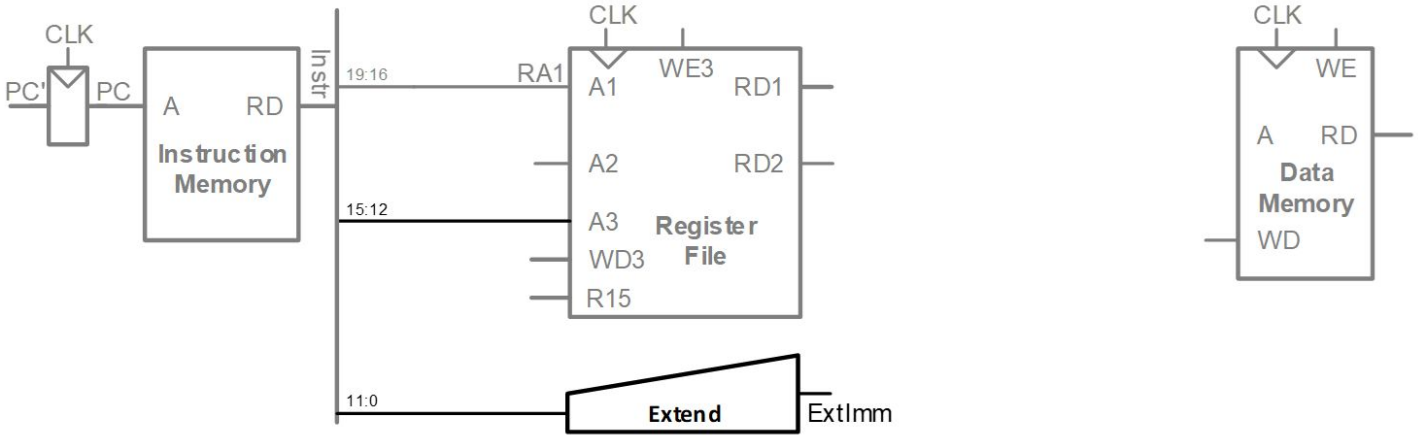


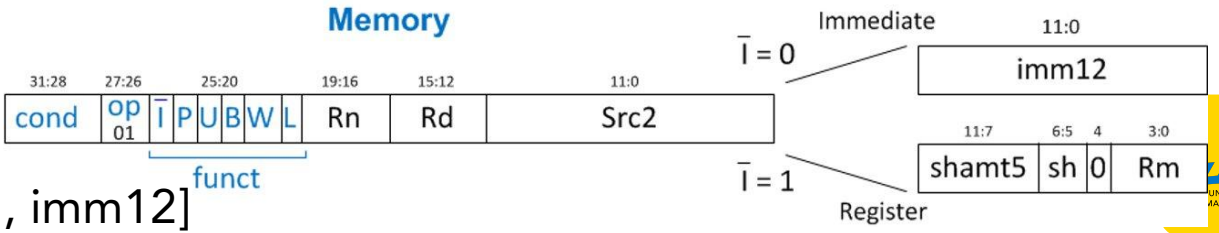
LDR Rd, [Rn, imm12]



Single-Cycle Datapath: LDR Immed.

STEP 3: Extend the immediate

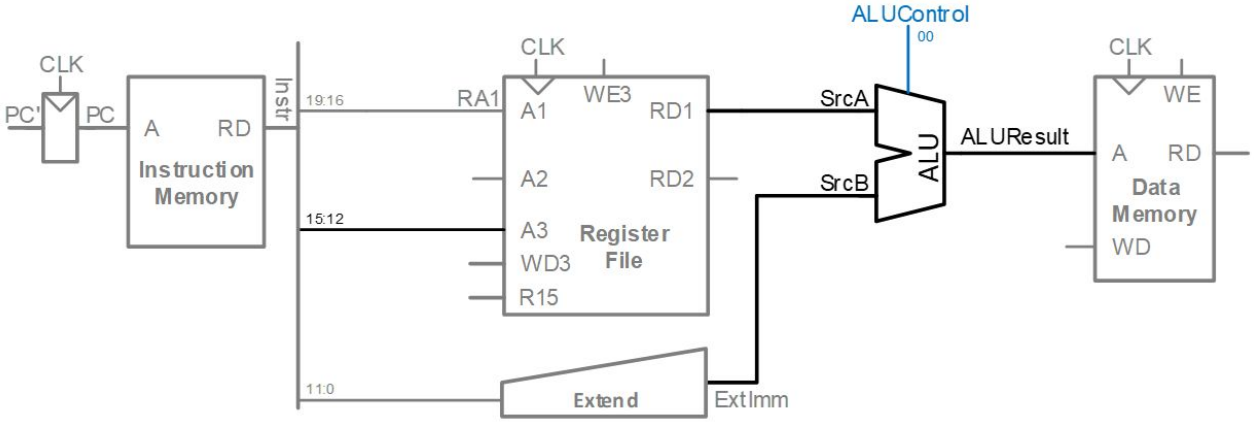




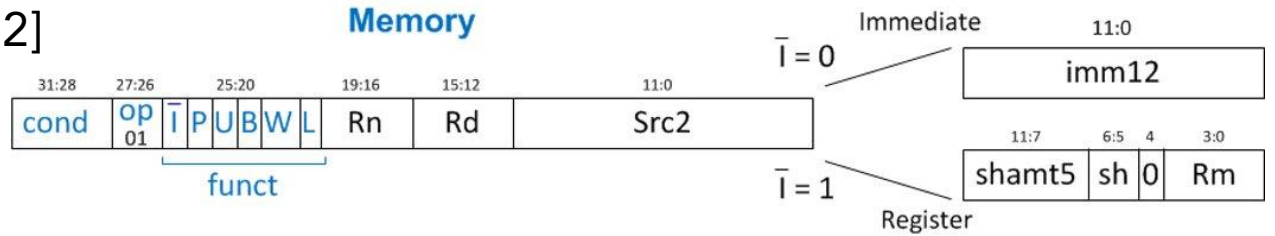
LDR Rd, [Rn, imm12]

Single-Cycle Datapath: LDR Address

STEP 4: Compute the memory address

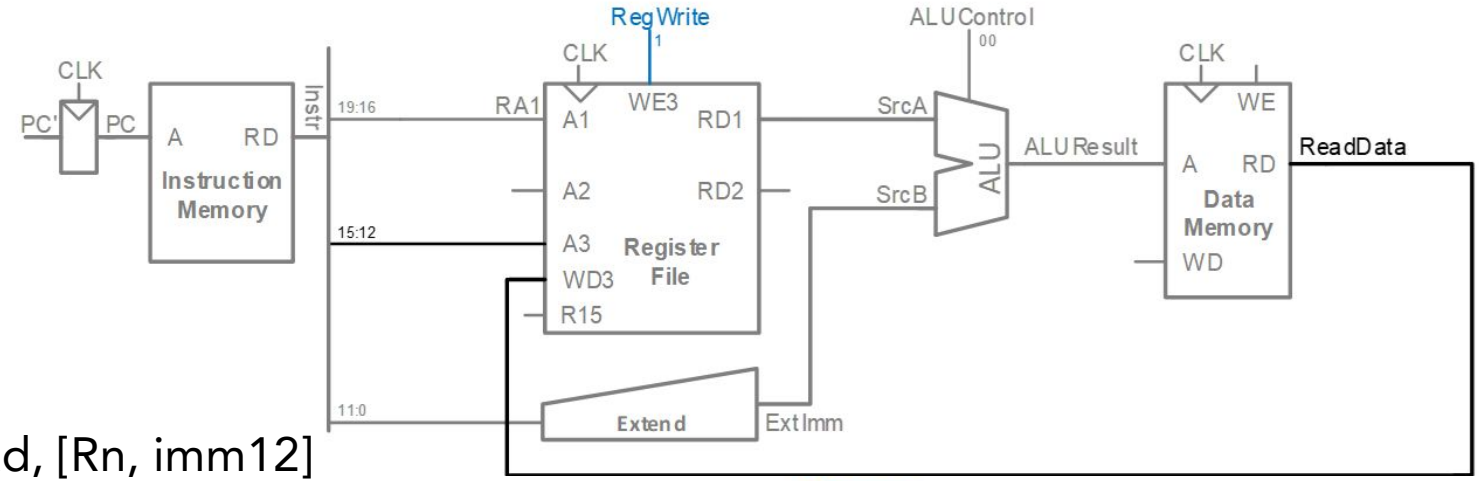


LDR Rd, [Rn, imm12]

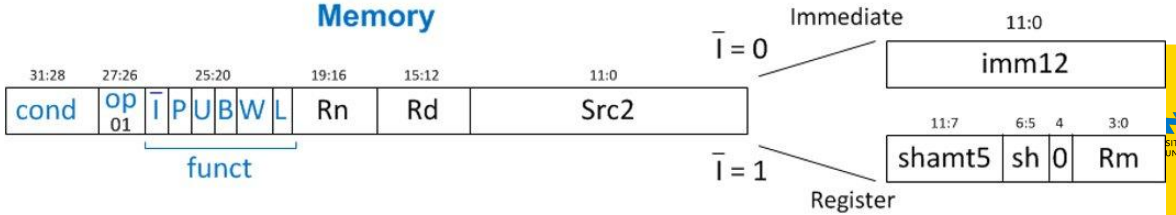


Single-Cycle Datapath: LDR Mem Read

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

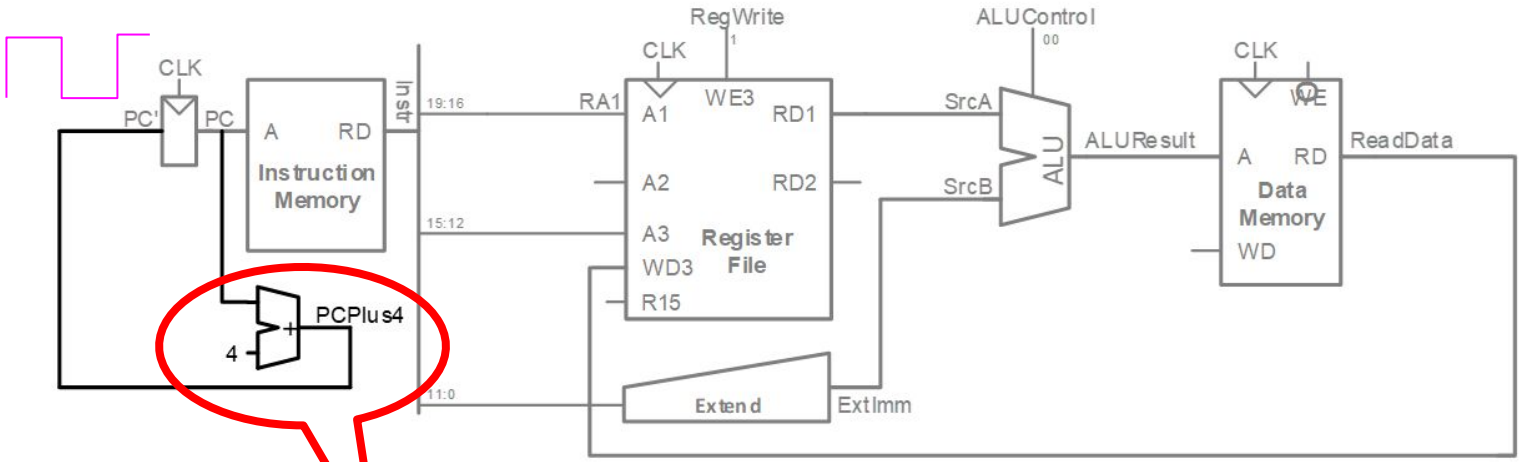


LDR Rd, [Rn, imm12]



Single-Cycle Datapath: PC Increment

STEP 6: Determine address of next instruction



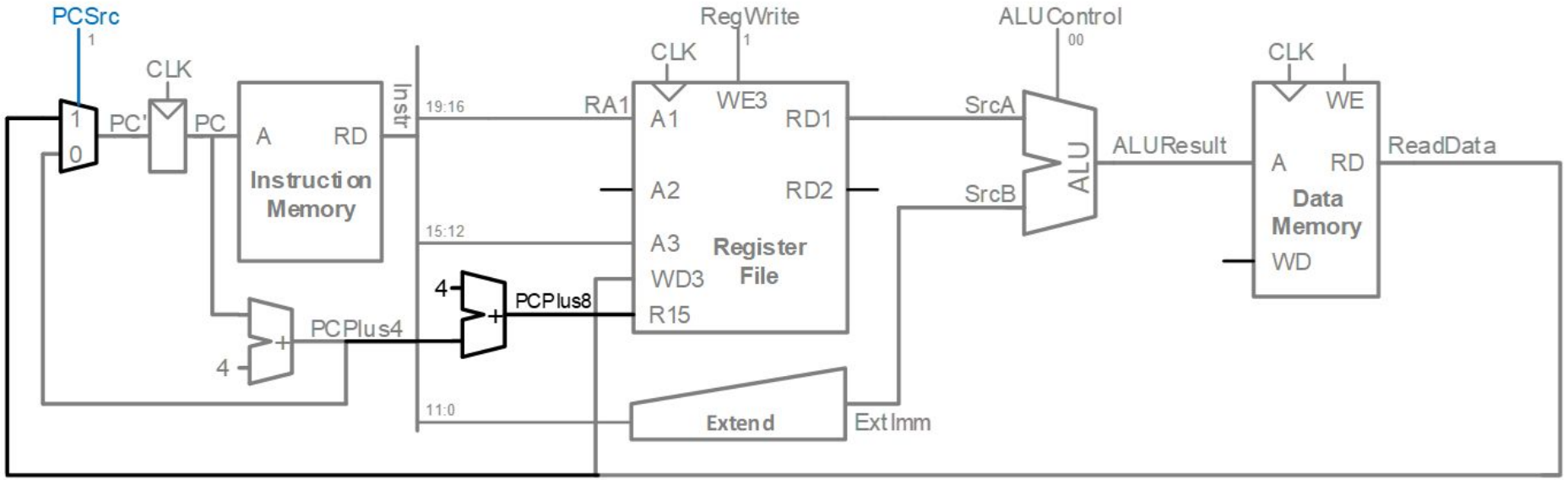
Calculate the address of the next instruction

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Single-Cycle Datapath: Access to PC <Case of Destination being R15/PC>

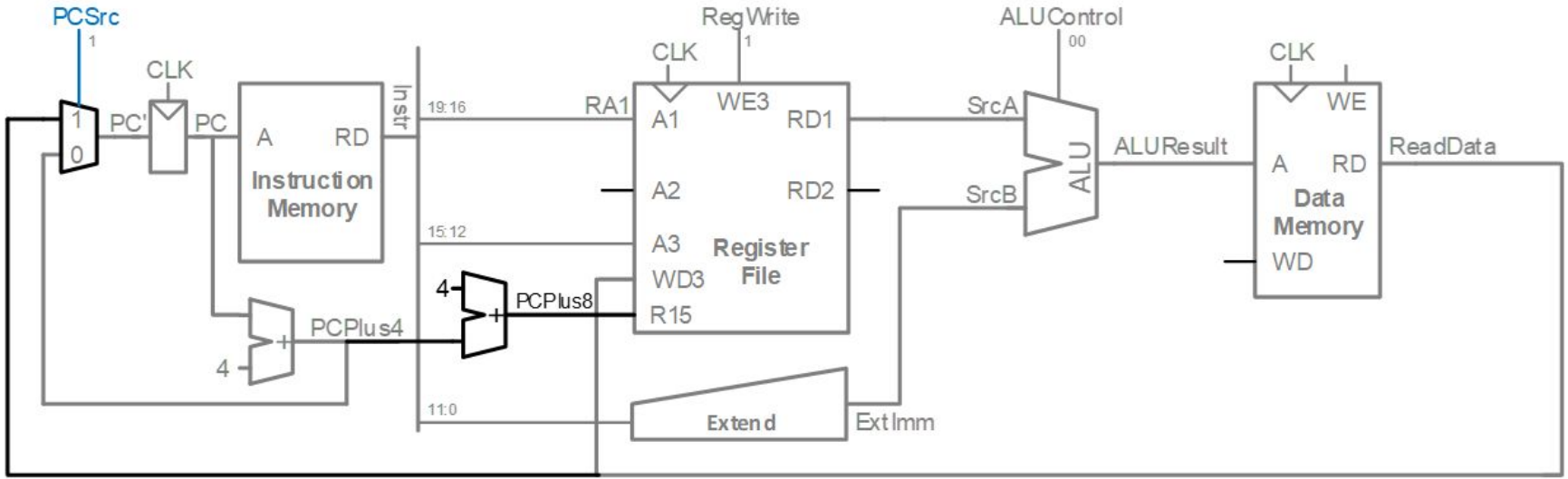
PC (R15) can be source/destination of instruction



Single-Cycle Datapath: Access to PC <Case of Destination being R15/PC>

PC can be source/destination of instruction

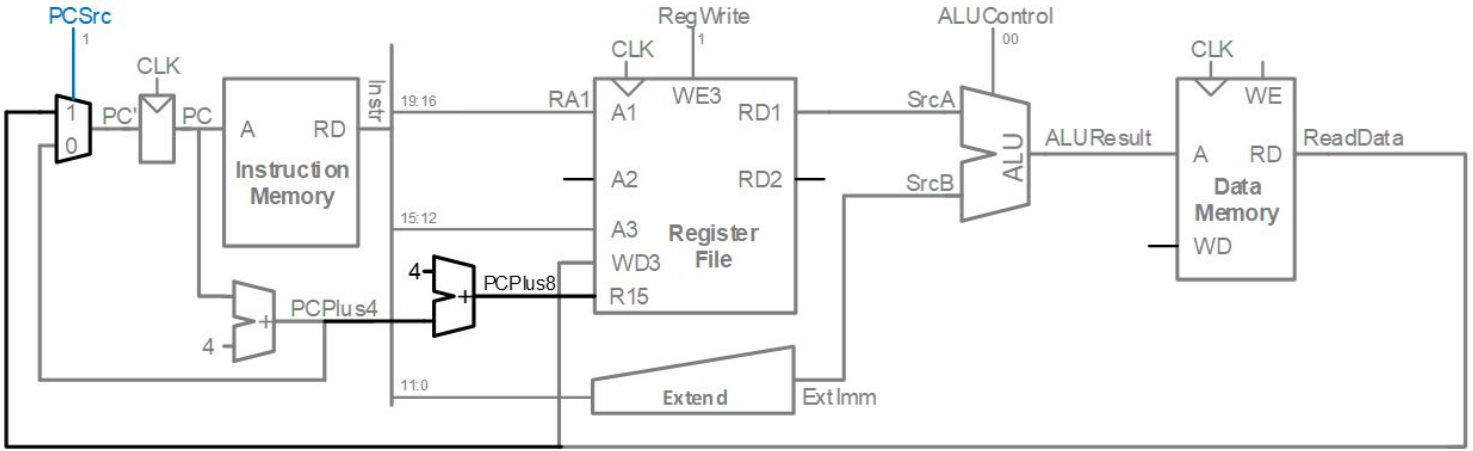
- **Source:** R15 must be available in Register File
  - **PC** is read as the current **PC plus 8**



Single-Cycle Datapath: Access to PC <Case of Destination being R15/PC>

PC can be source/destination of instruction

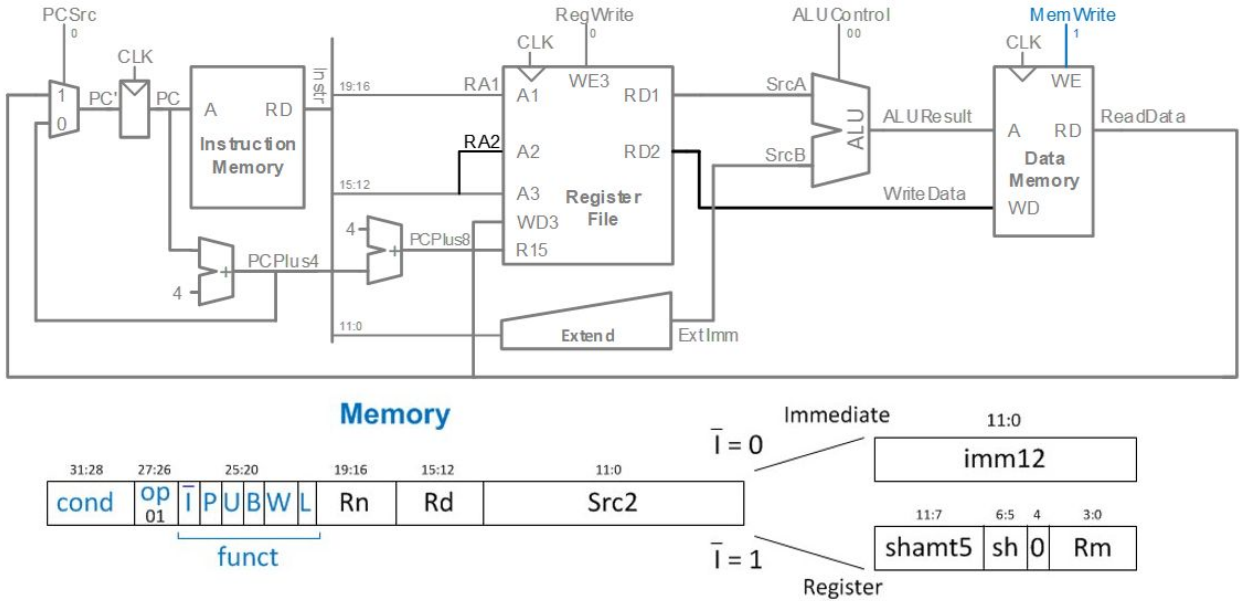
- **Source:** R15 must be available in Register File
  - **PC** is read as the current **PC plus 8**
- **Destination:** Be able to write result to PC



Single-Cycle Datapath: STR

Expand datapath to handle STR: STR Rd, [Rn, imm12]

- Write data in Rd to memory

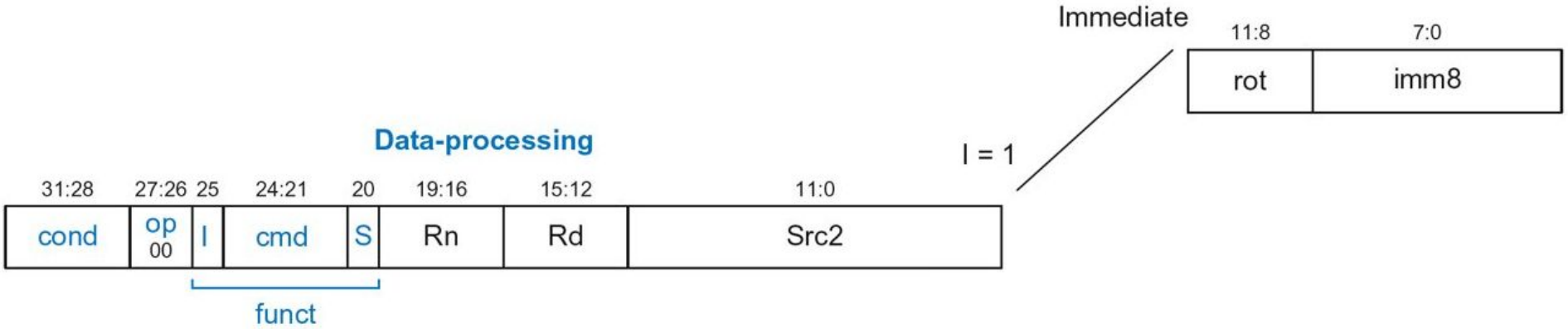


Single-Cycle Datapath: Data-processing

With immediate Src2:

- Read from Rn and Imm8 (*ImmSrc* chooses the zero-extended Imm8 instead of Imm12)
- Write *ALUResult* to register file
- Write to Rd

ADD Rd, Rn, imm8

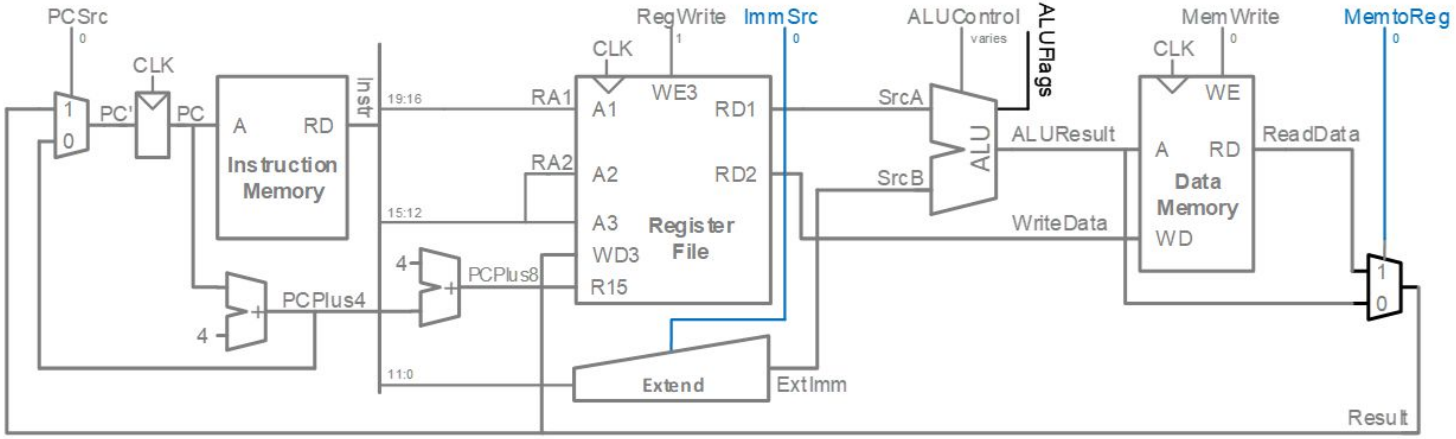


Single-Cycle Datapath: Data-processing

With immediate Src2:

- Read from `Rn` and `Imm8` (*ImmSrc* chooses the zero-extended `Imm8` instead of `Imm12`)
- Write *ALUResult* to register file
- Write to `Rd`

ADD Rd, Rn, imm8

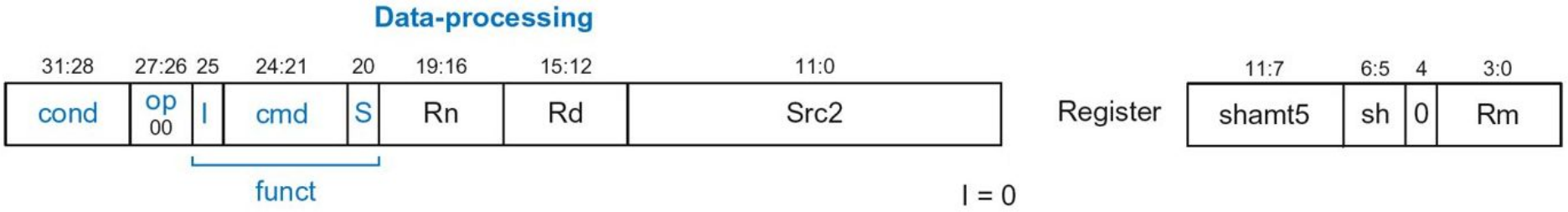


Single-Cycle Datapath: Data-processing

With register Src2:

- Read from Rn and Rm (instead of Imm8)
- Write *ALUResult* to register file
- Write to Rd

ADD Rd, Rn, Rm



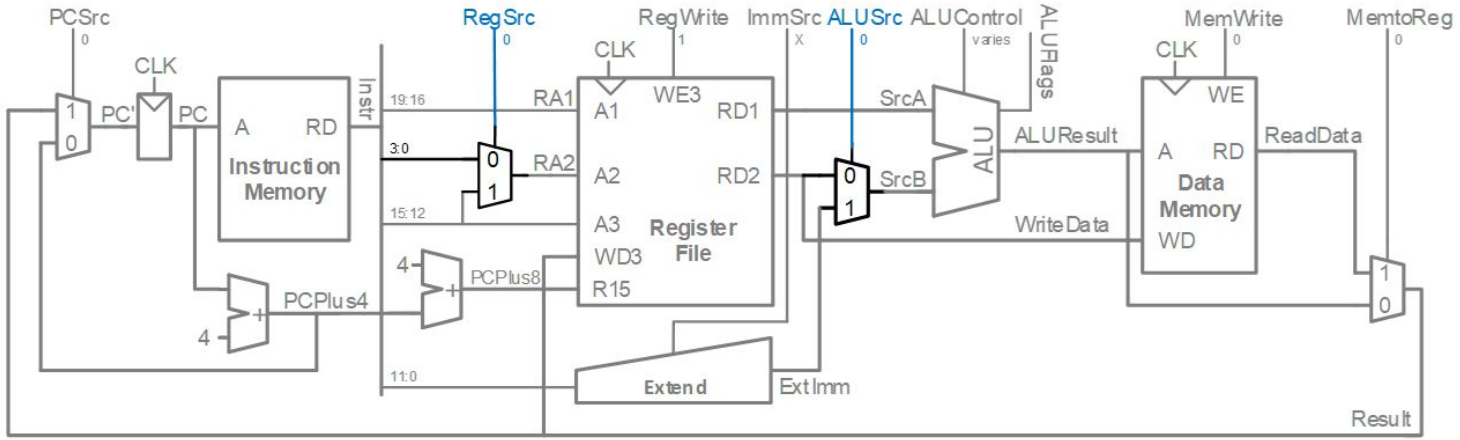
Single-Cycle Datapath: Data-processing

With register Src2:

- Read from Rn and Rm (instead of Imm8)
- Write *ALUResult* to register file
- Write to Rd

Same ALU handles, ADD (00), SUB(01), AND (10), ORR (11)

ADD Rd, Rn, Rm

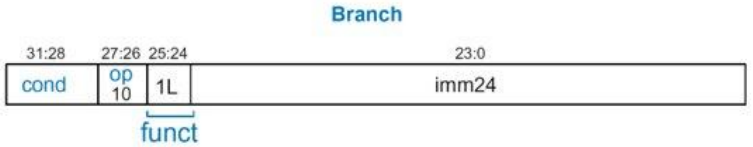


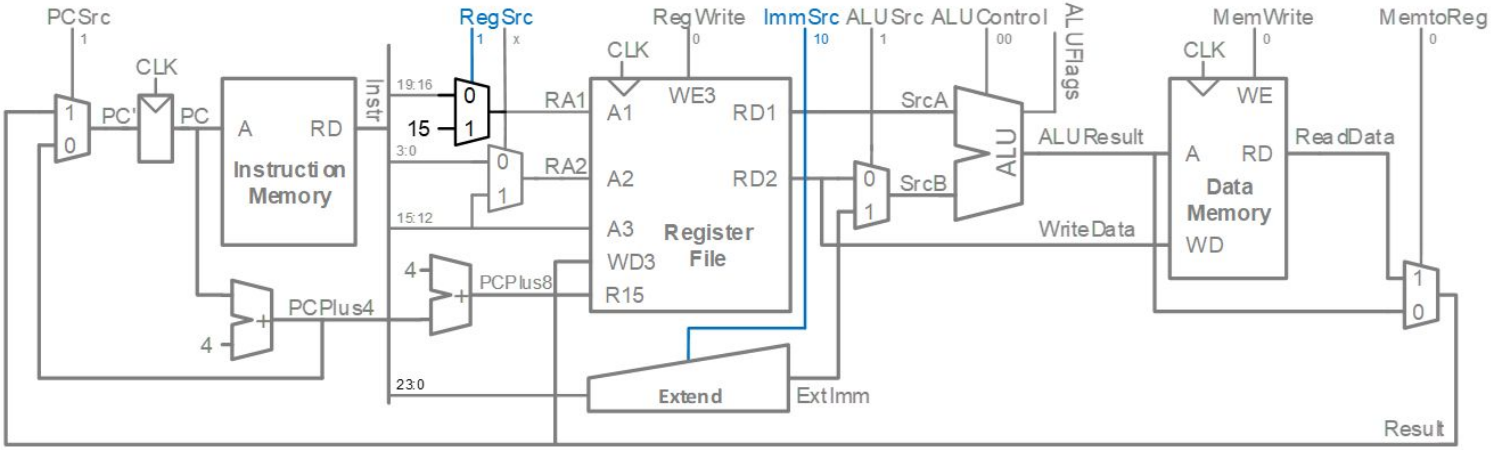
Single-Cycle Datapath: B

Calculate branch target address:

BTA = (*ExtImm*) + (PC + 8)

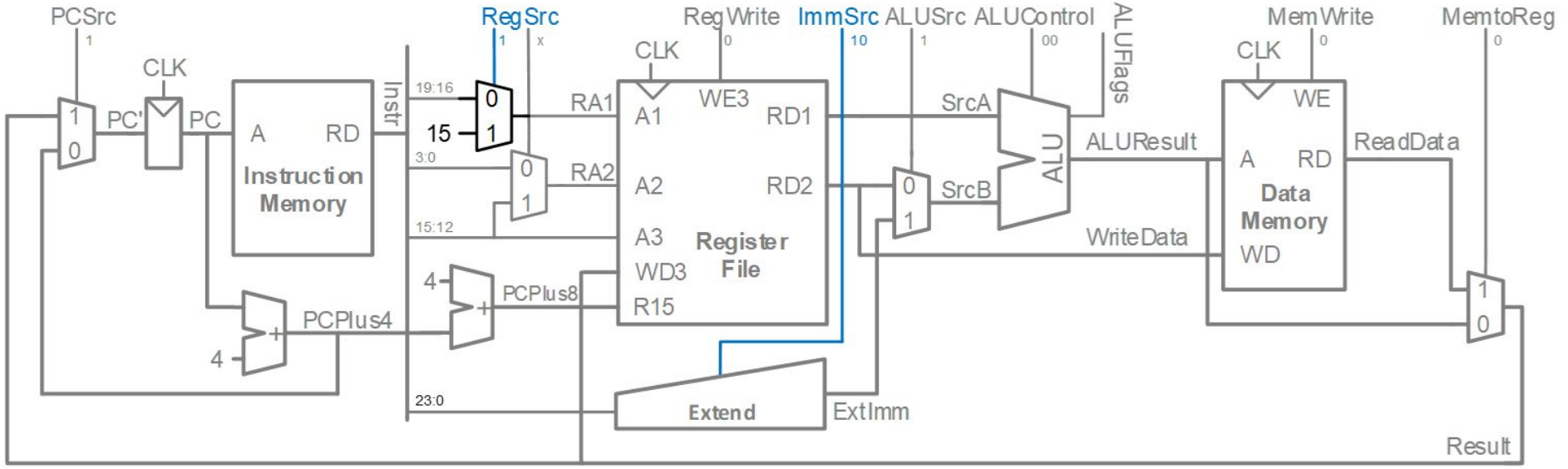
ExtImm = Imm24 << *2* and sign-extended



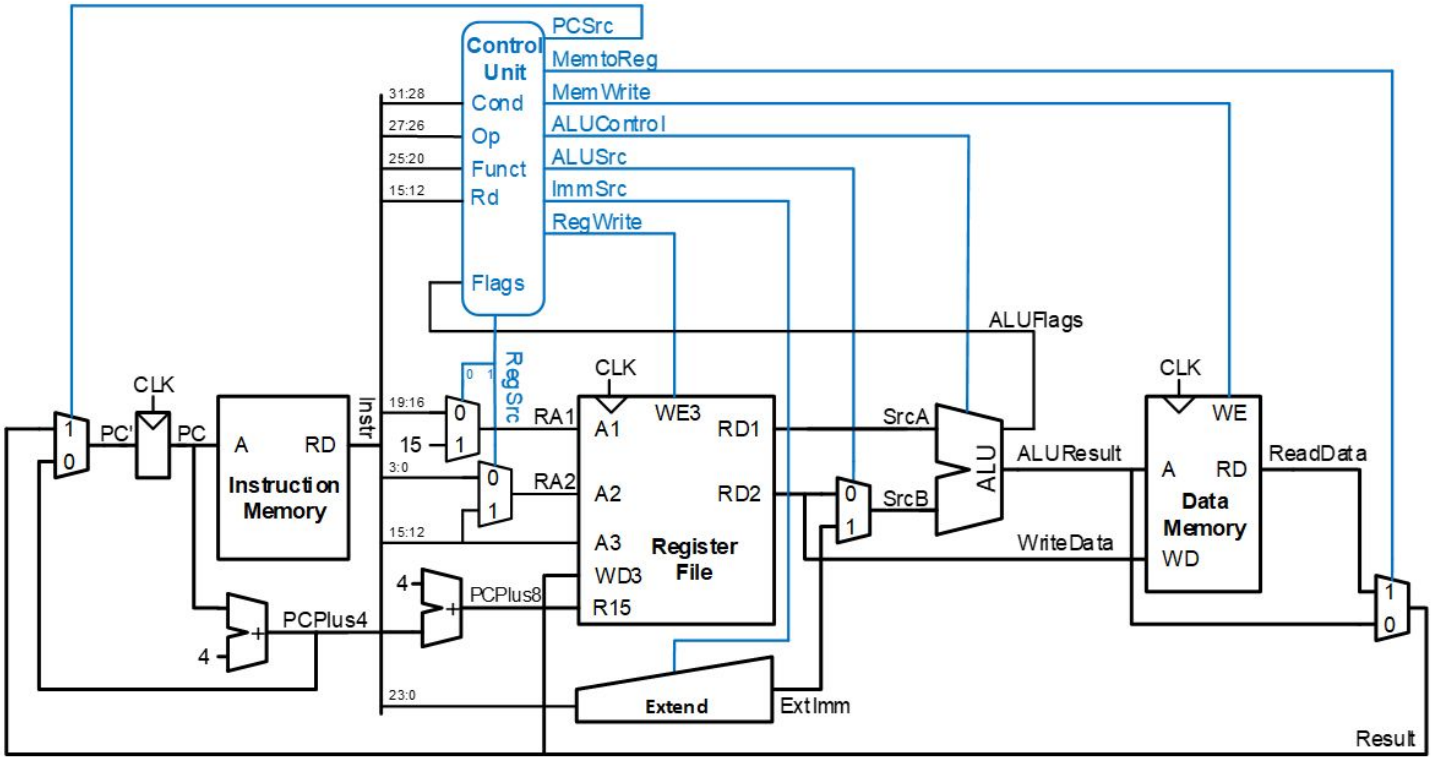


Single-Cycle Datapath: ExtImm

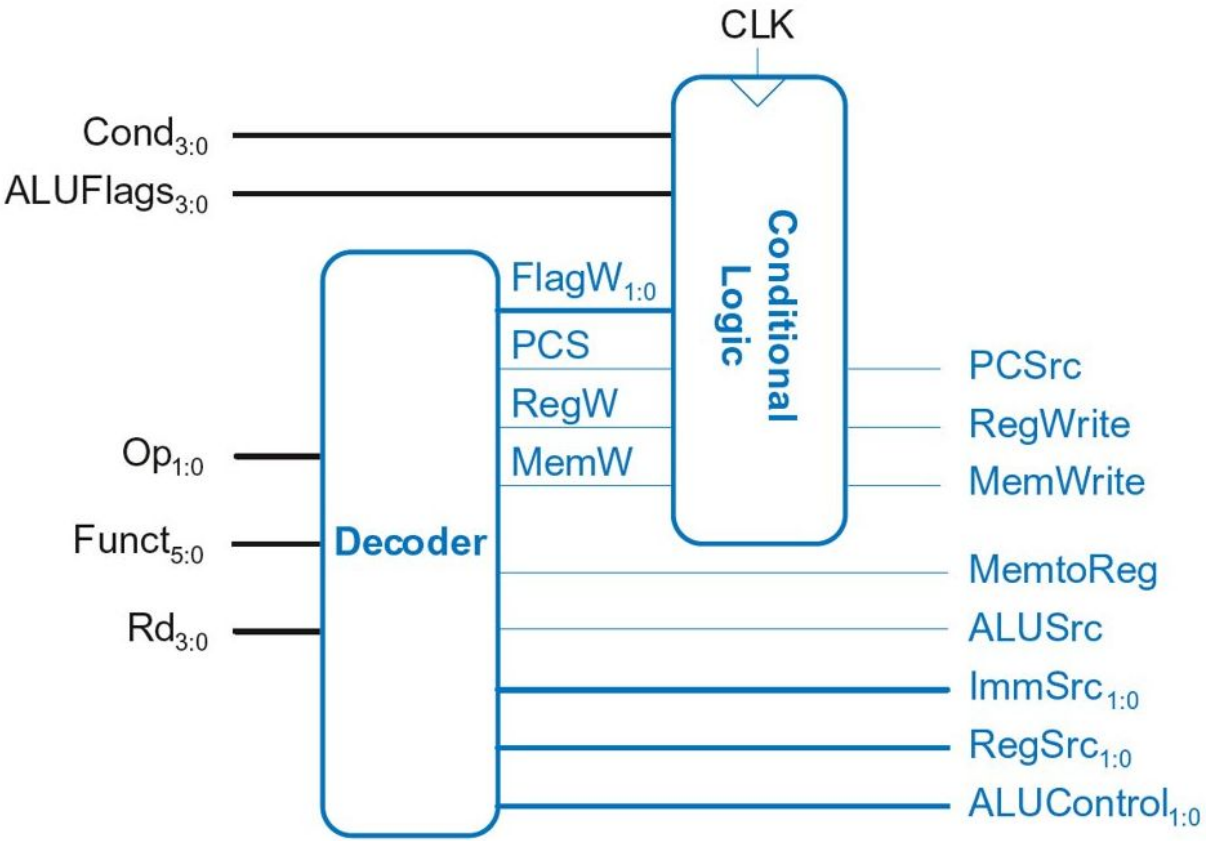
$ImmSrc_{1:0}$	ExtImm	Description
00	{24 `0s`, $Instr_{7:0}$}	Zero-extended *imm8*
01	{20 `0s`, $Instr_{11:0}$}	Zero-extended *imm12*
10	{6 $Instr_{23}$}, $Instr_{23:0}$}	Sign-extended *imm24*



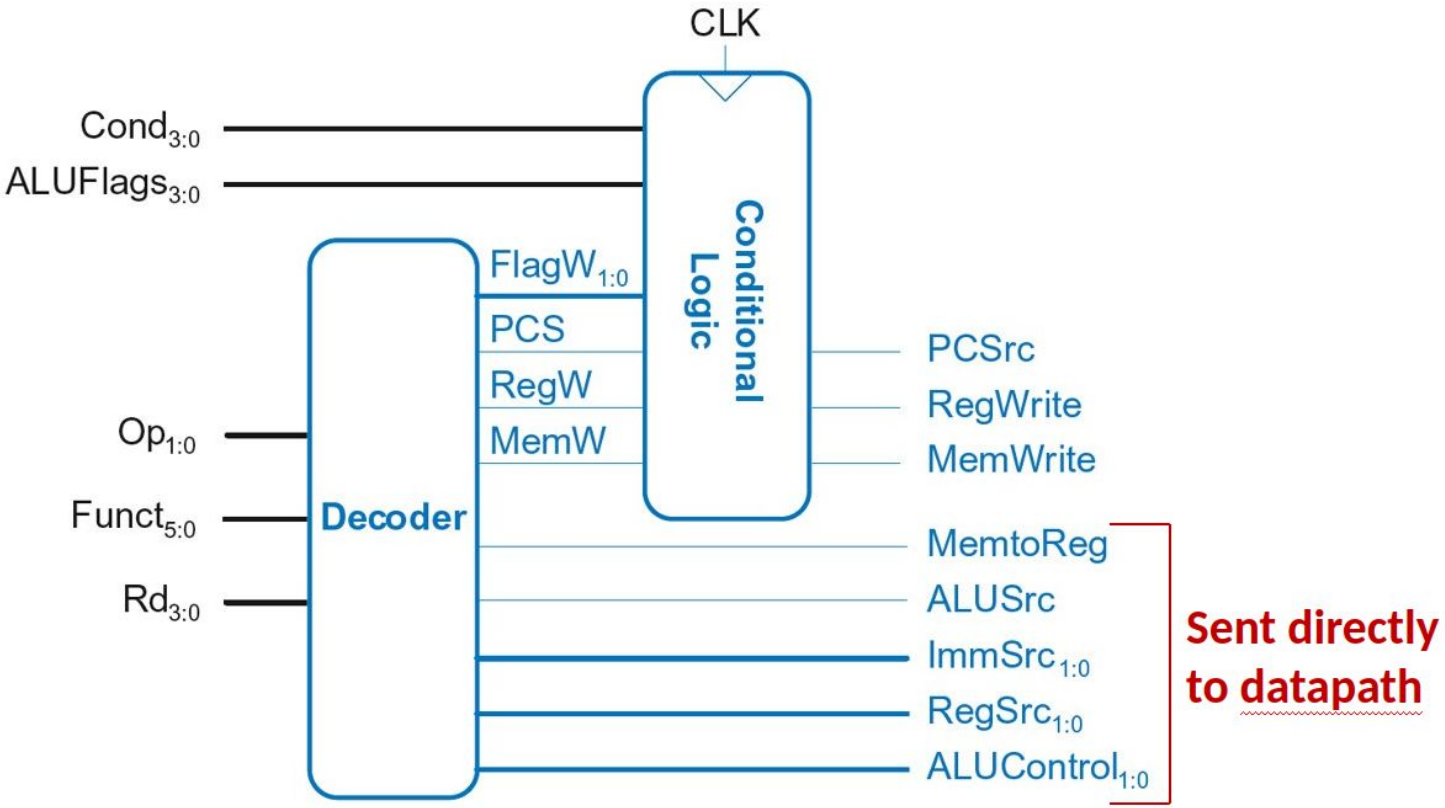
Single-Cycle ARM Processor



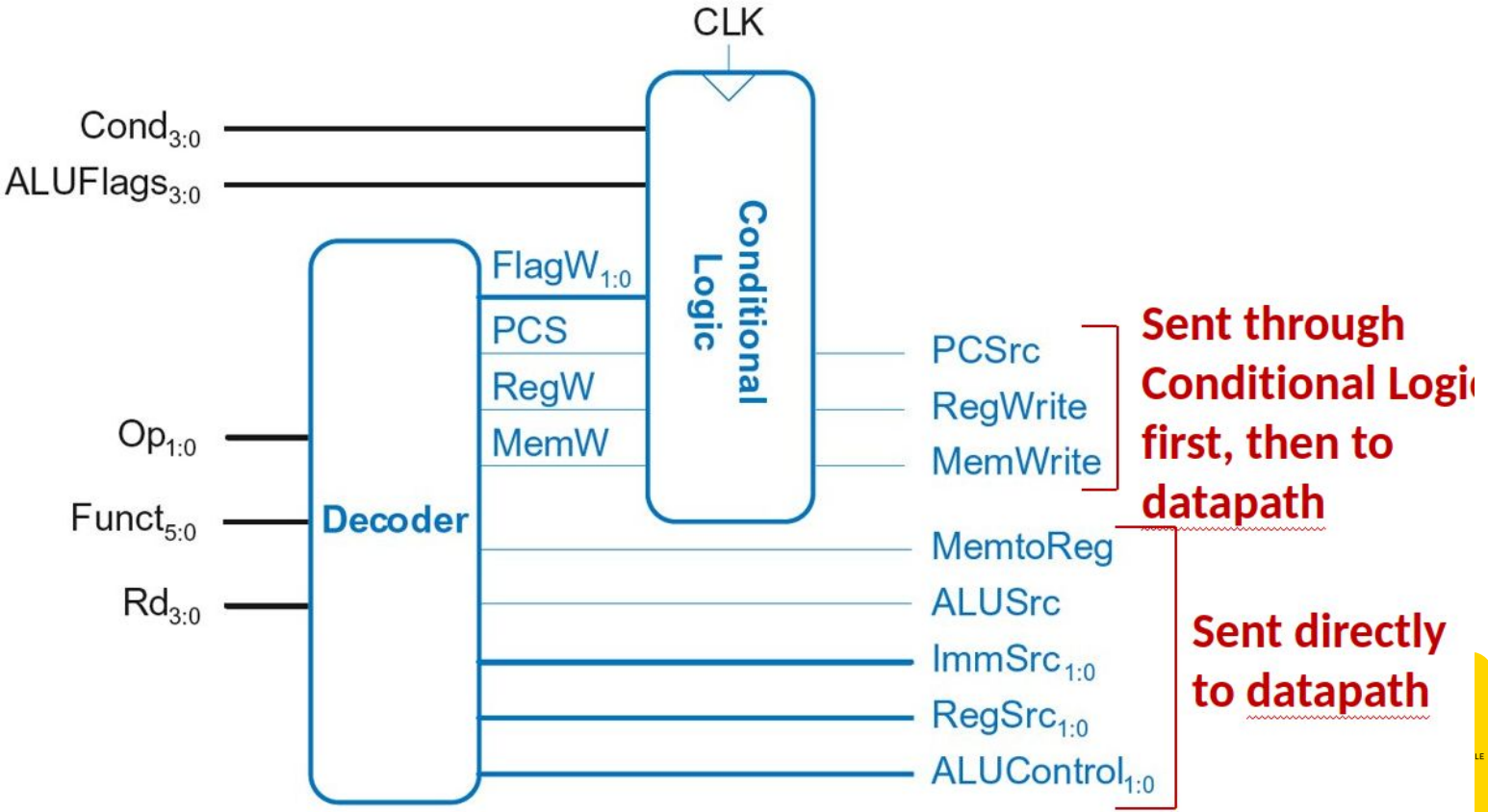
Single-Cycle Control



Single-Cycle Control

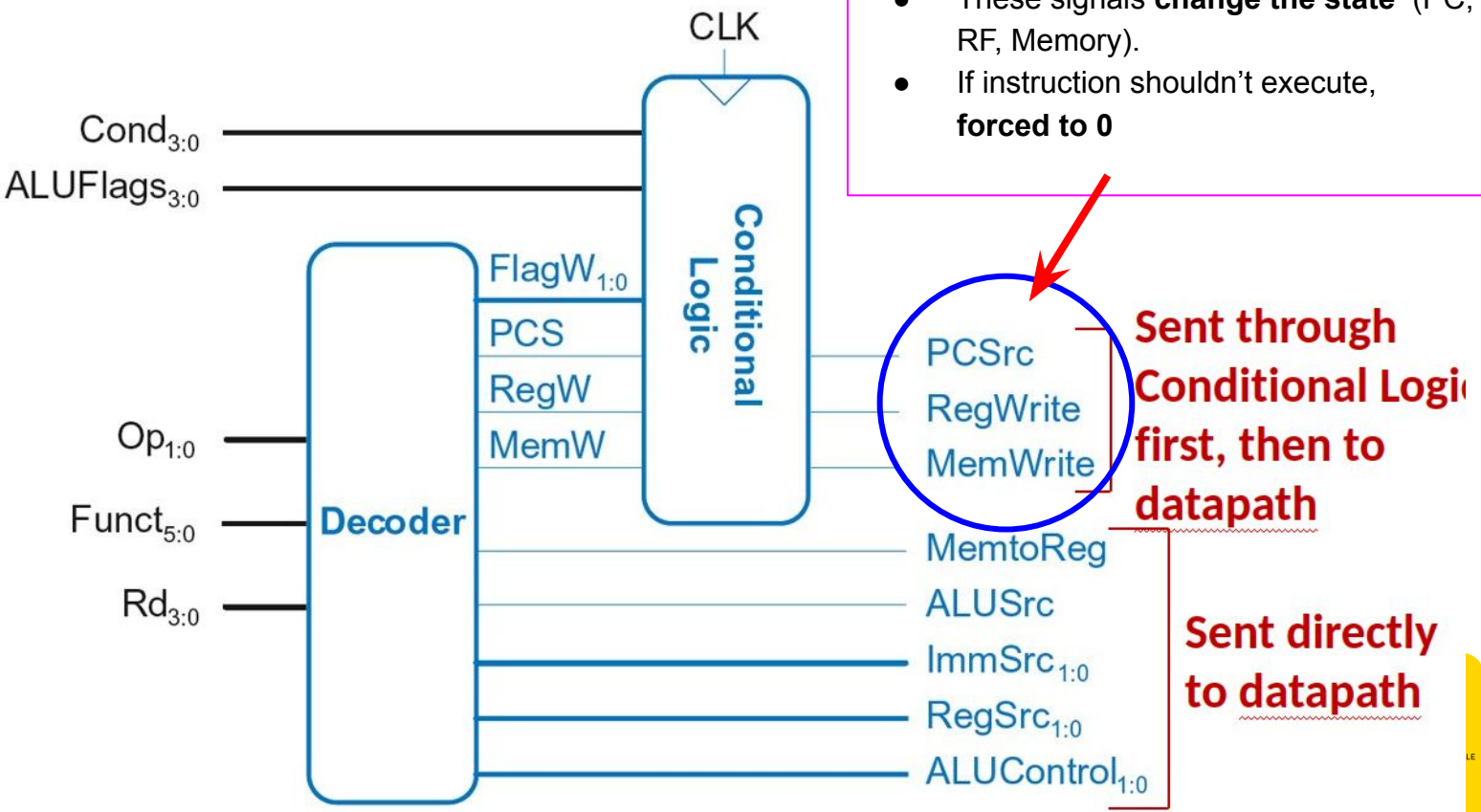


Single-Cycle Control

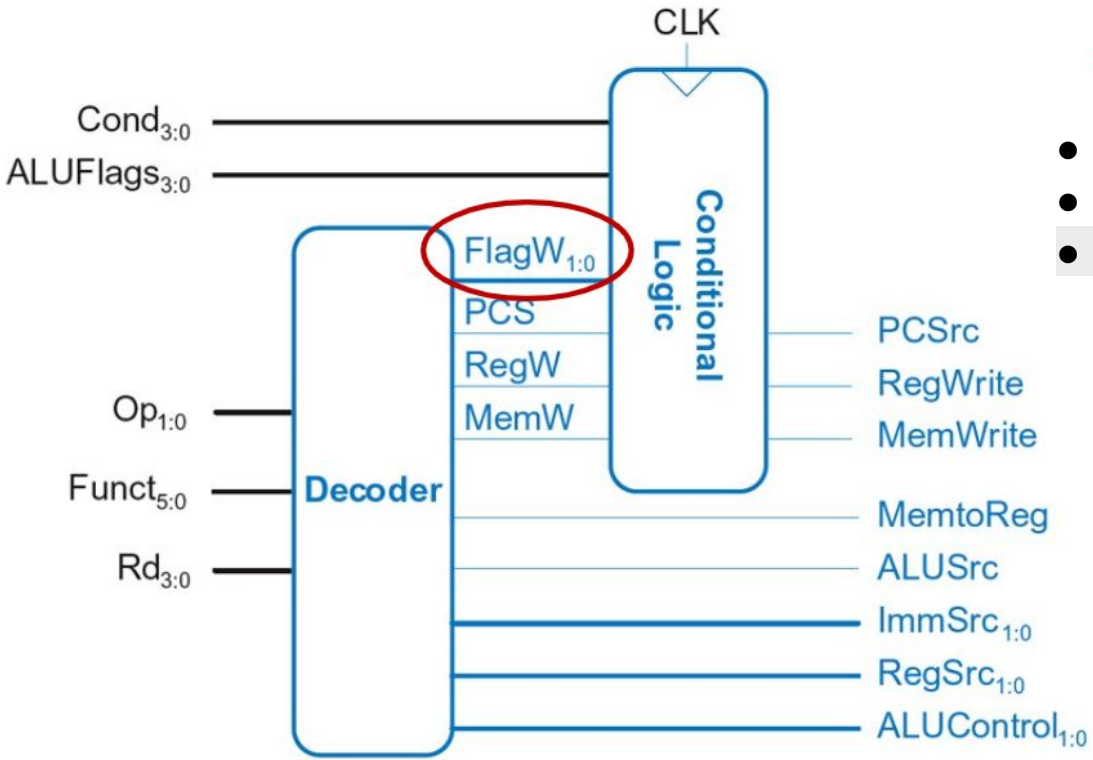


Single-Cycle Control

- These signals **change the state** (PC, RF, Memory).
- If instruction shouldn't execute, **forced to 0**

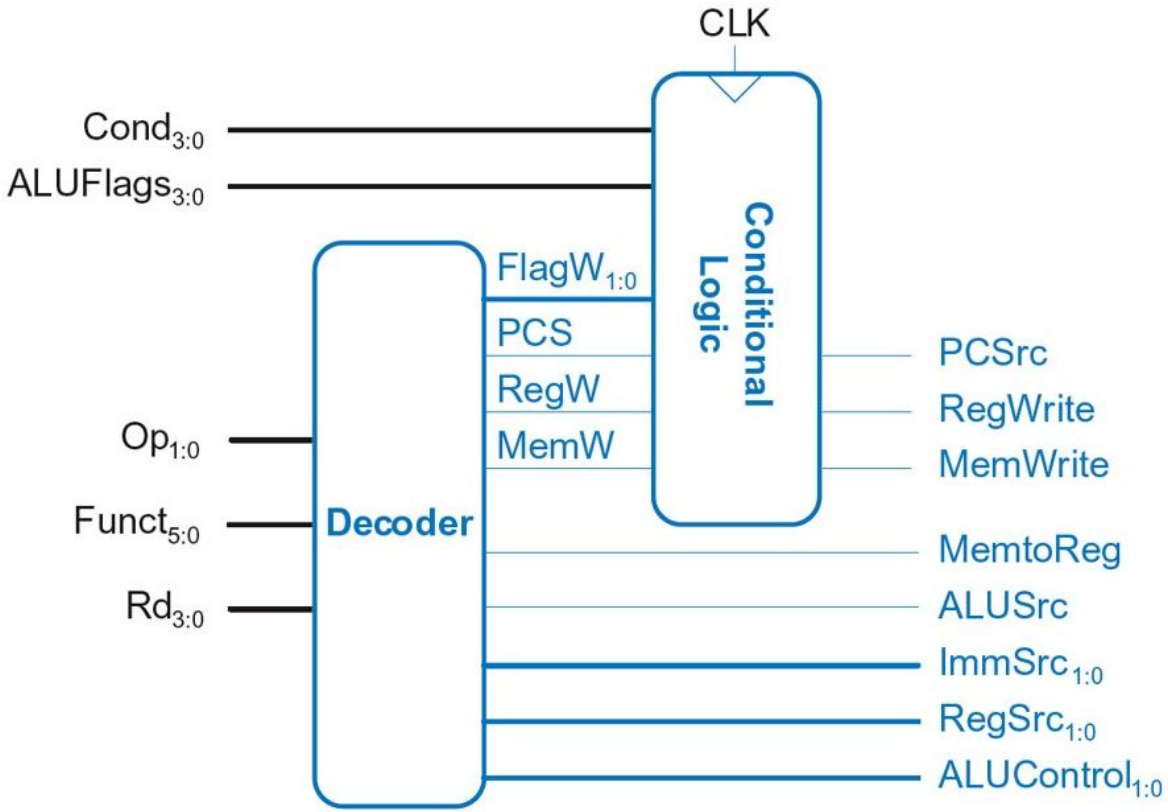


Single-Cycle Control

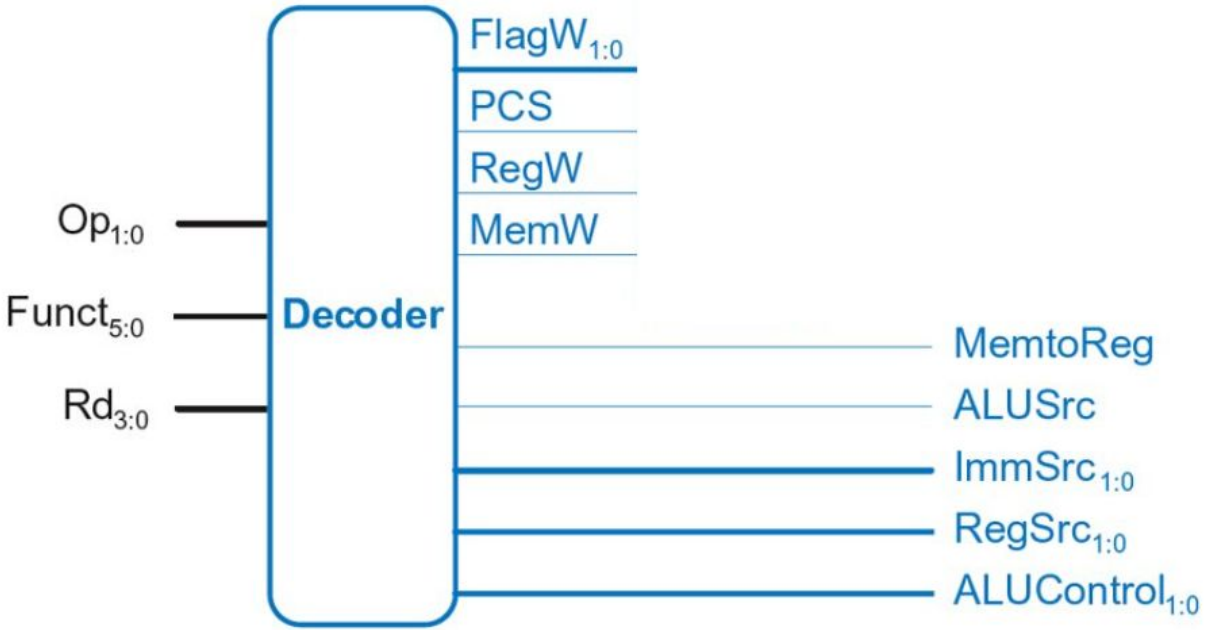


- ***FlagW*$_{1:0}$**: Flag Write signal, asserted when *ALUFlags* should be saved (i.e., on instruction with S=1)
- ADD, SUB update all flags (**NZCV**)
- AND, ORR only update **NZ** flags
- So, two bits needed:
  ***FlagW*$_1$** = 1: NZ saved (ALUFlags3:2 saved)
  ***FlagW*$_0$** = 1: CV saved (ALUFlags1:0 saved)

Single-Cycle Control



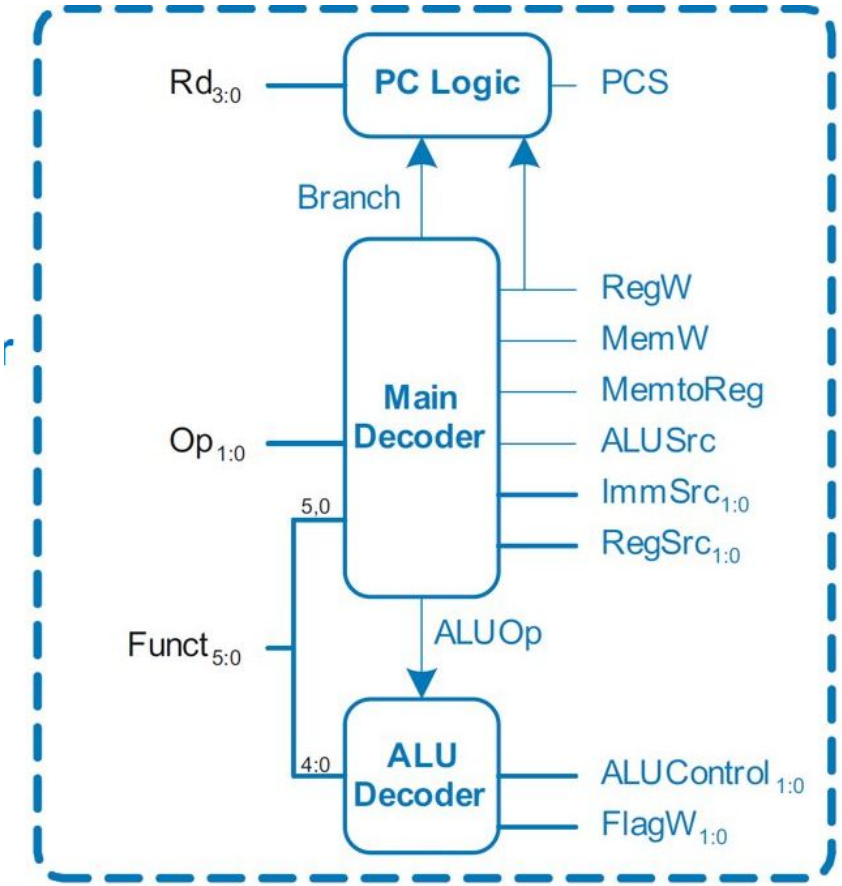
Single-Cycle Control: Decoder



Single-Cycle Control: Decoder

Submodules:

- Main Decoder
- ALU Decoder
- PC Logic



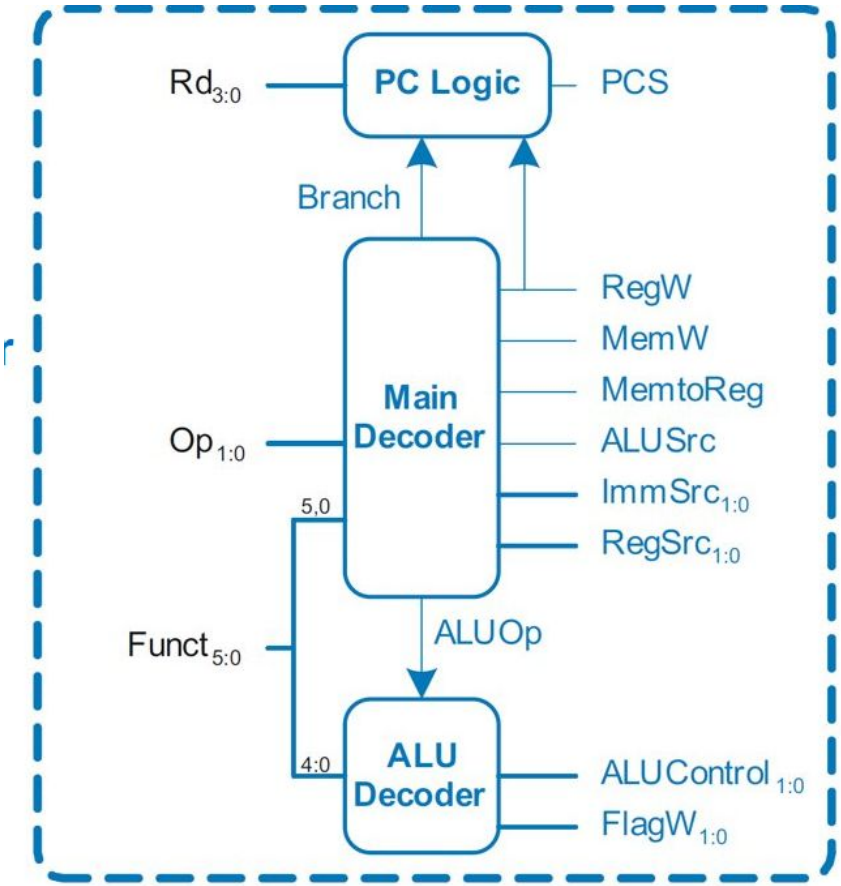
Control Unit: Main Decoder

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:

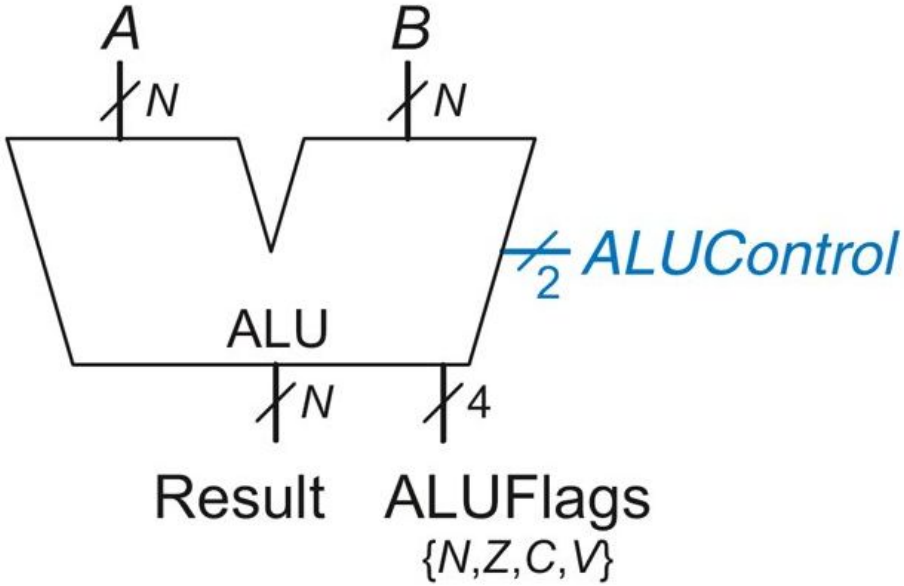
- Main Decoder
- ALU Decoder
- PC Logic



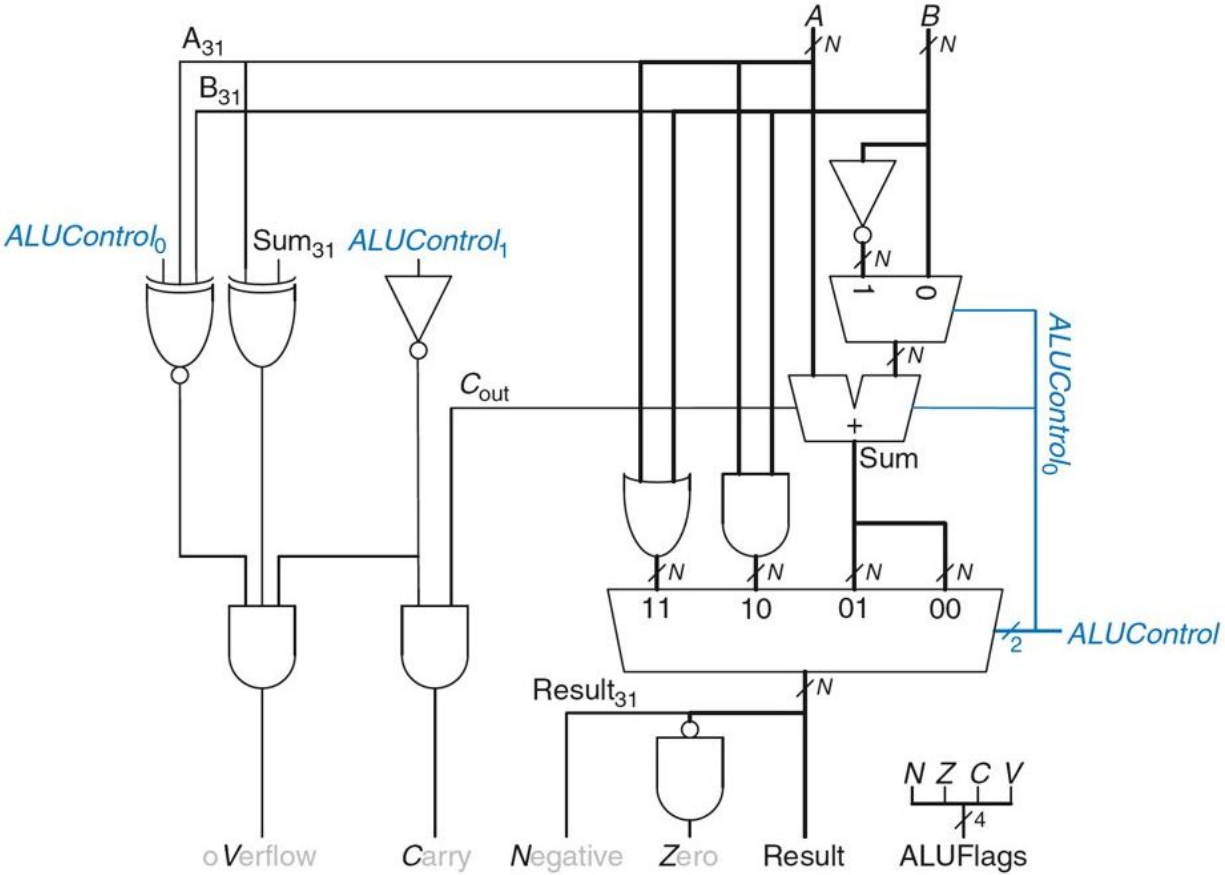
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Review: ALU

$ALUControl_{1:0}$	Function
00	Add
01	Subtract
10	AND
11	OR



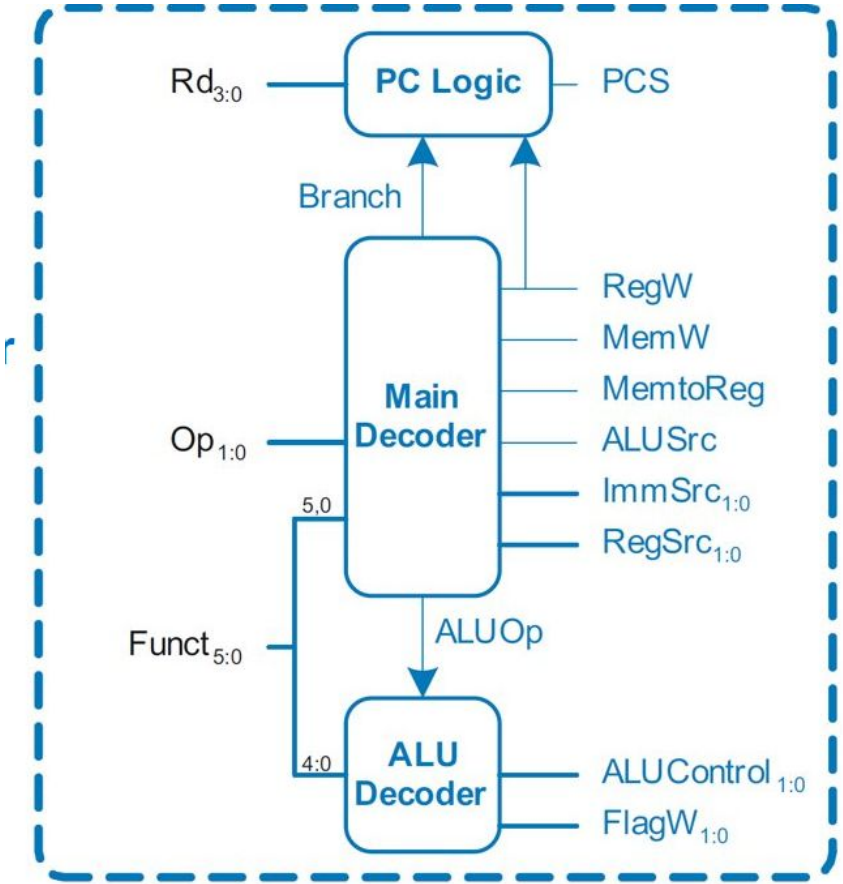
ALU


A
B
N
A31
B31
ALUControl0
Sum31
ALUControl1
Cout
1
0
+
Sum
ALUControl0
11
10
01
00
2
ALUControl
Result31
oVerflow
Carry
Negative
Zero
Result
N Z C V
4
ALUFlags

Single-Cycle Control: Decoder

Submodules:

- Main Decoder
- ALU Decoder
- PC Logic



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Control Unit: ALU Decoder

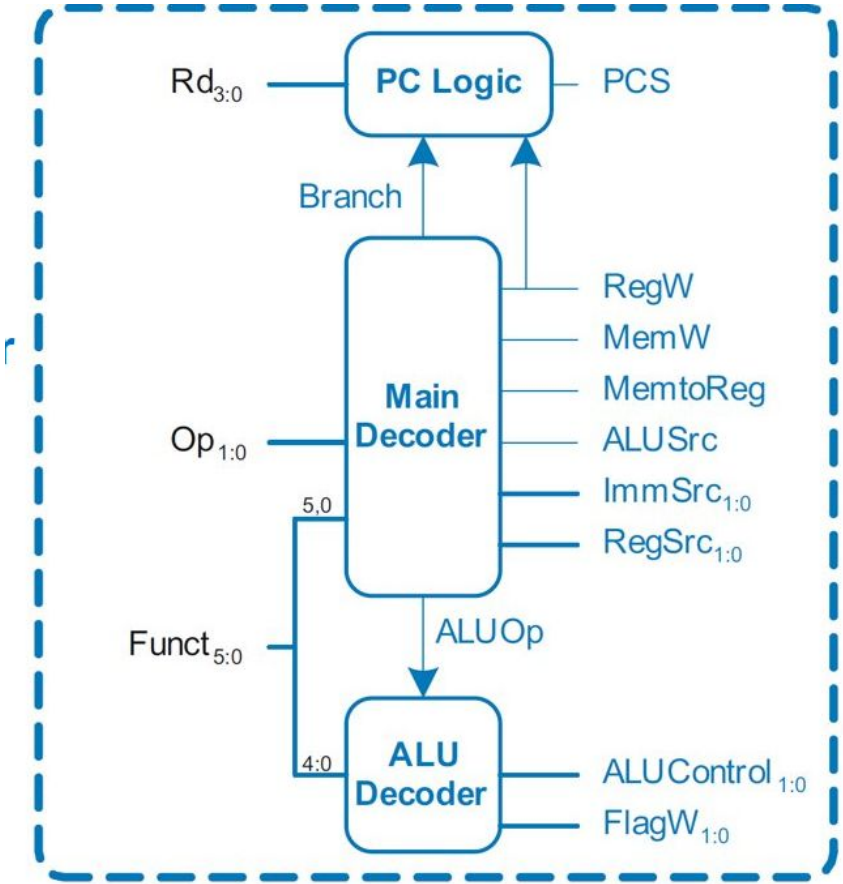
- **FlagW$_1$** = 1: *NZ* (*Flags*$_{3:2}$) should be saved
- **FlagW$_0$** = 1: *CV* (*Flags*$_{1:0}$) should be saved

ALUOp	Funct$_{4:1}$ (*cmd*)	Funct$_0$ (*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

Single-Cycle Control: Decoder

Submodules:

- Main Decoder
- ALU Decoder
- PC Logic



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

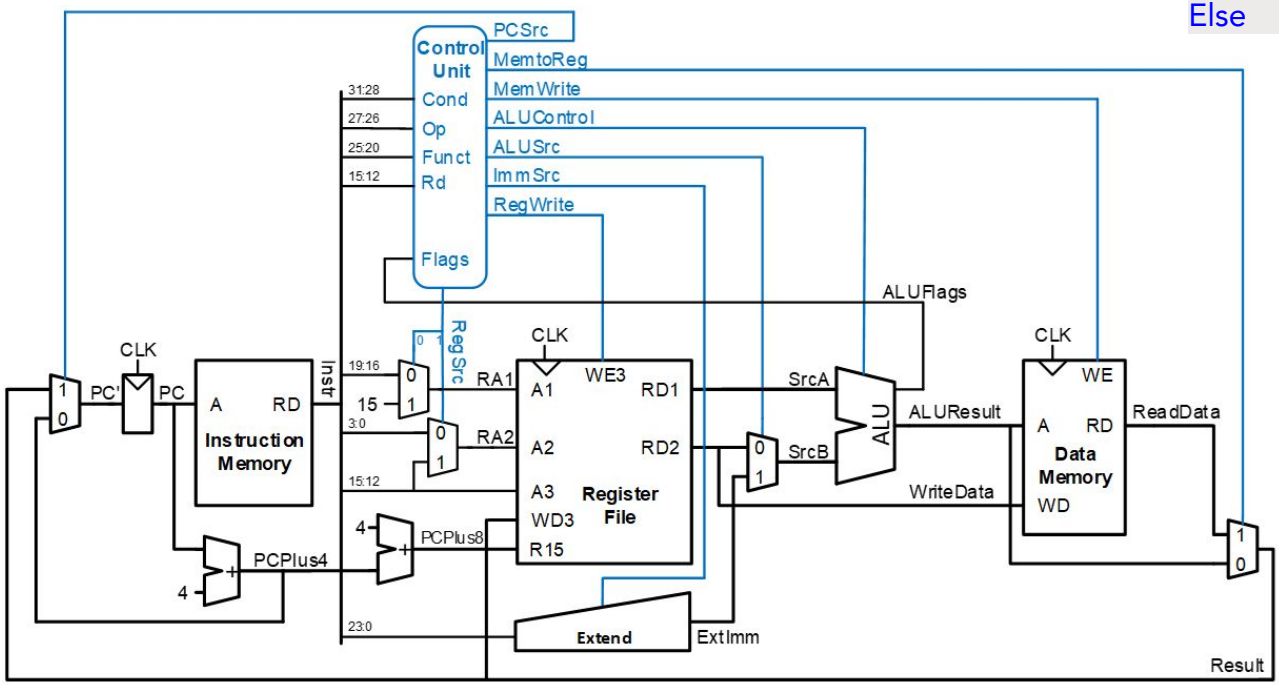
Single-Cycle Control: PC Logic

***PCS* = 1 if PC is written by an instruction or branch (B):**

$$PCS = ((Rd == 15)\ \&\ RegW)\ |\ Branch$$

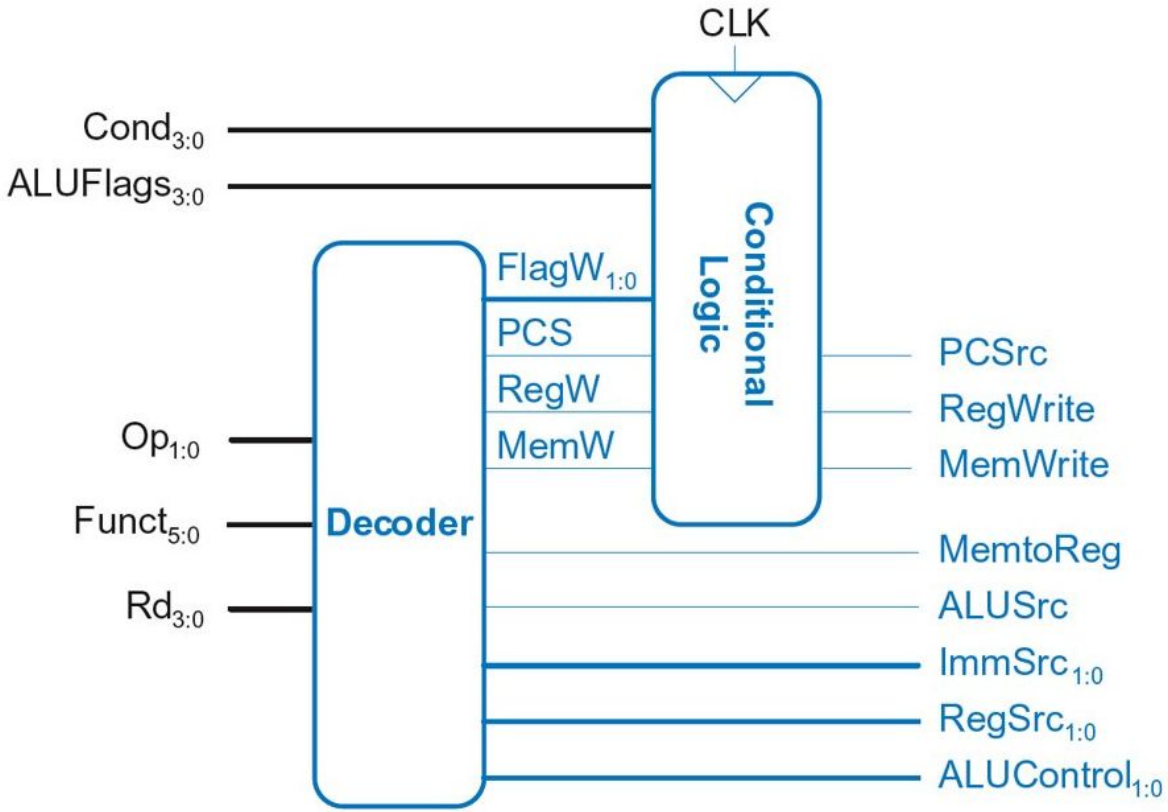
If instruction is executed: *PCSrc* = *PCS*

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



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

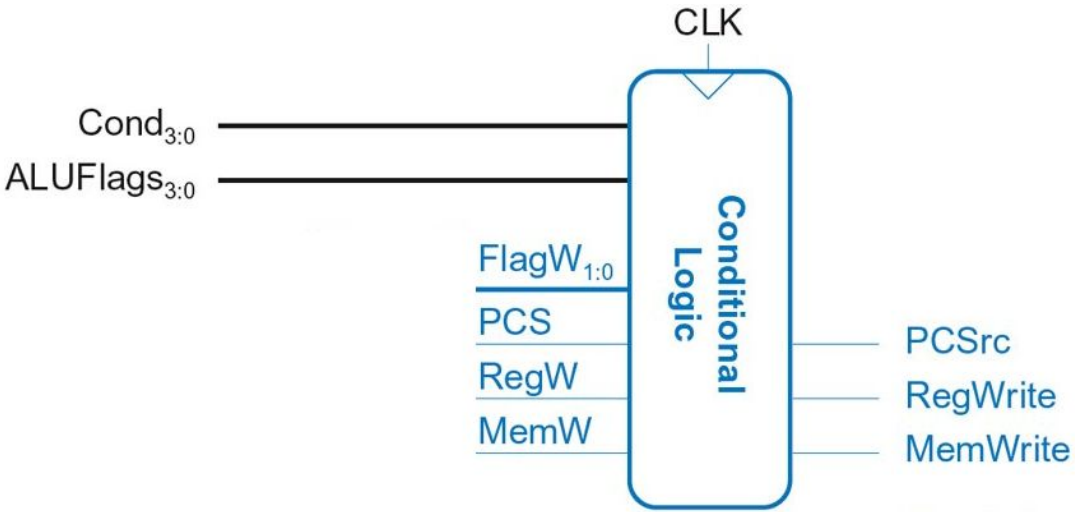
Single-Cycle Control: Cond. Logic



Single-Cycle Control: Cond. Logic

Function:

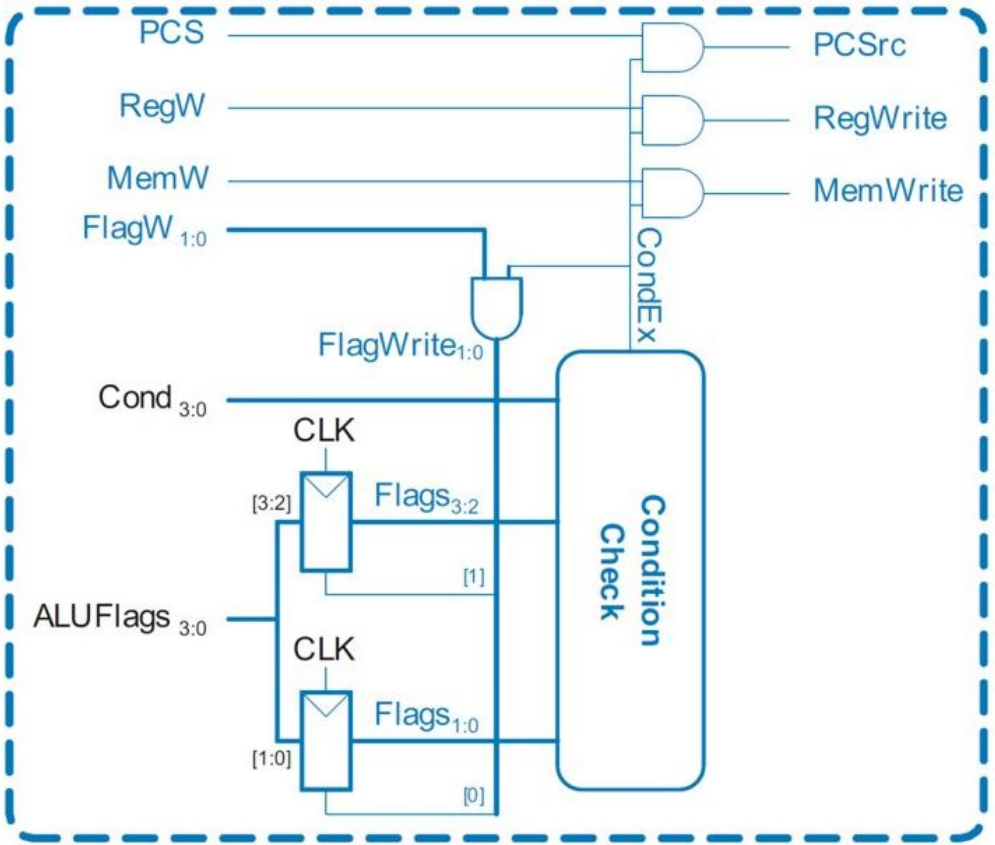
1. Check if instruction should execute (if not, force PCSrc, RegWrite, and MemWrite to 0)
2. Possibly update Status Register ($Flags_{3:0}$)



Single-Cycle Control: Cond. Logic

Function:

1. Check if instruction should execute (if not, force PCSrc, RegWrite, and MemWrite to 0)



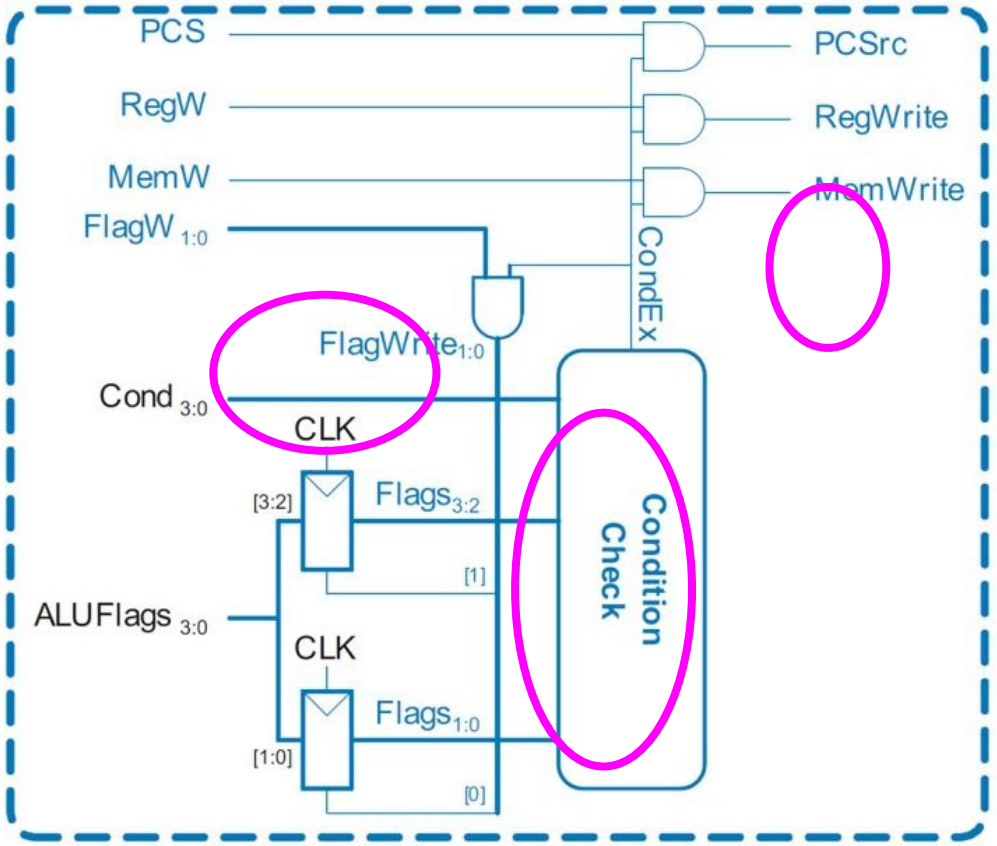
Single-Cycle Control: Cond. Logic

Flags3:0 is the status register

Function:

1. **Check if instruction should execute (if not, force PCSrc, RegWrite, and MemWrite to 0)**

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	
0001	NE	Not equal	
0010	CS / HS	Carry set / Unsigned higher or same	
0011	CC / LO	Carry clear / Unsigned lower	
0100	MI	Minus / Negative	
0101	PL	Plus / Positive of zero	
0110	VS	Overflow / Overflow set	
0111	VC	No overflow / Overflow clear	
1000	HI	Unsigned higher	
1001	LS	Unsigned lower or same	
1010	GE	Signed greater than or equal	
1011	LT	Signed less than	
1100	GT	Signed greater than	
1101	LE	Signed less than or equal	
1110	AL (or none)	Always / unconditional	ignored

Conditional Logic: Conditional Execution

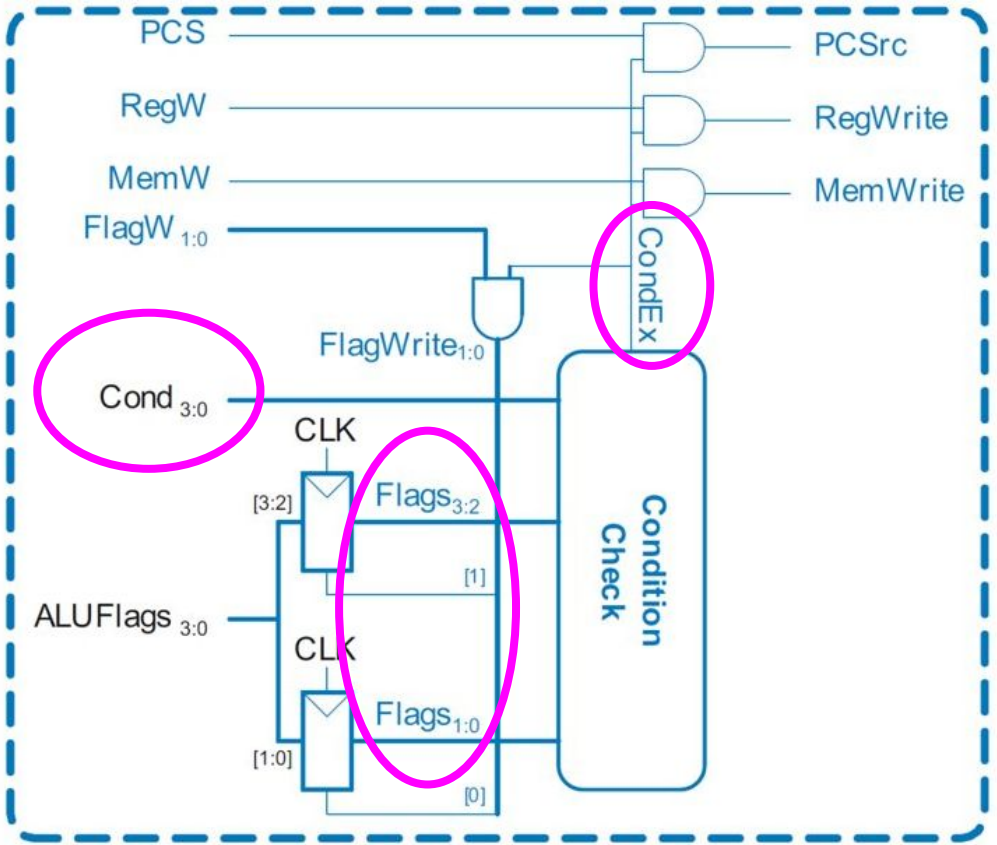
Flags$_{3:0}$ is the status register

Example:

```
AND R1, R2, R3
```

Cond$_{3:0}$=1110 (unconditional) =>

CondEx = 1



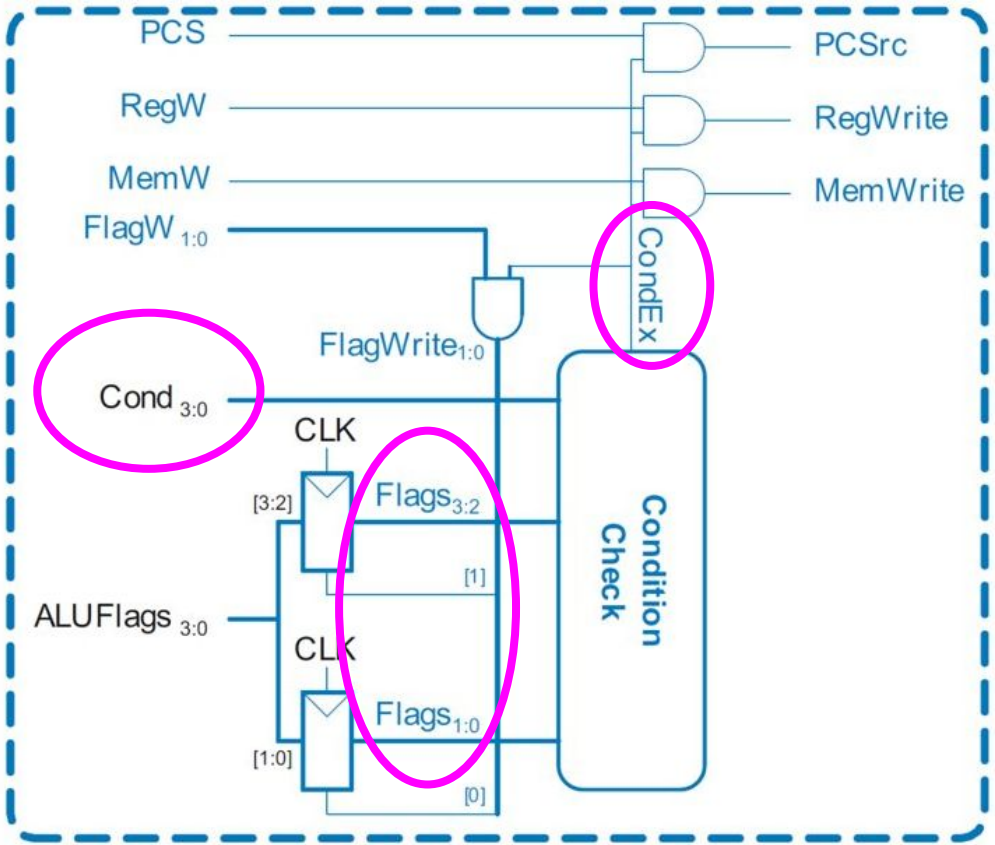
Conditional Logic: Conditional Execution

Flags$_{3:0}$ is the status register

Example:

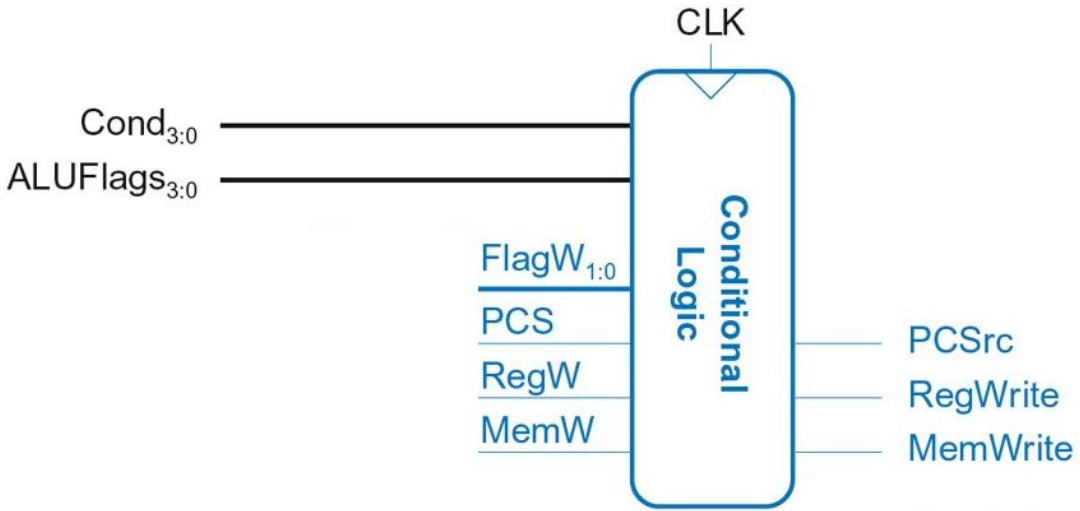
```
EOREQ R5, R6, R7
```

Cond$_{3:0}$=0000 (EQ): if **Flags$_{3:2}$**=0100 => **CondEx** = 1



Function:

2. Possibly update Status Register ($Flags_{3:0}$)



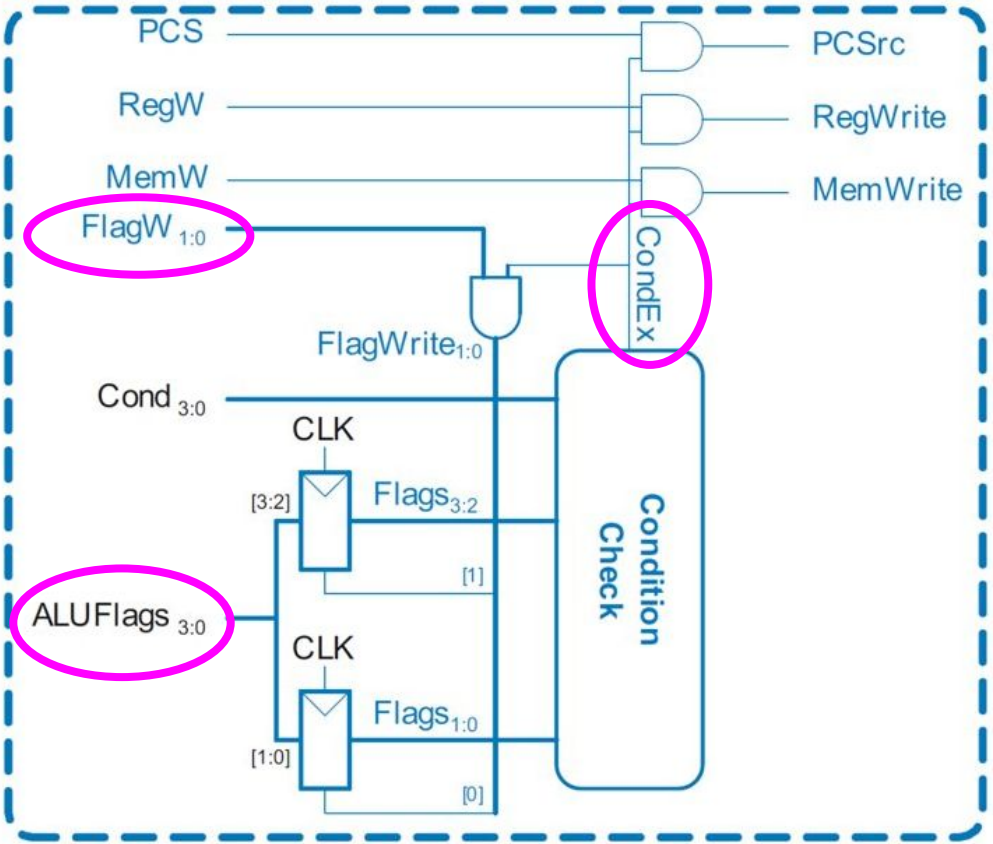
Single-Cycle Control: Cond. Logic

Flags3:0 is the status register

$Flags_{3:0}$ = NZCV

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

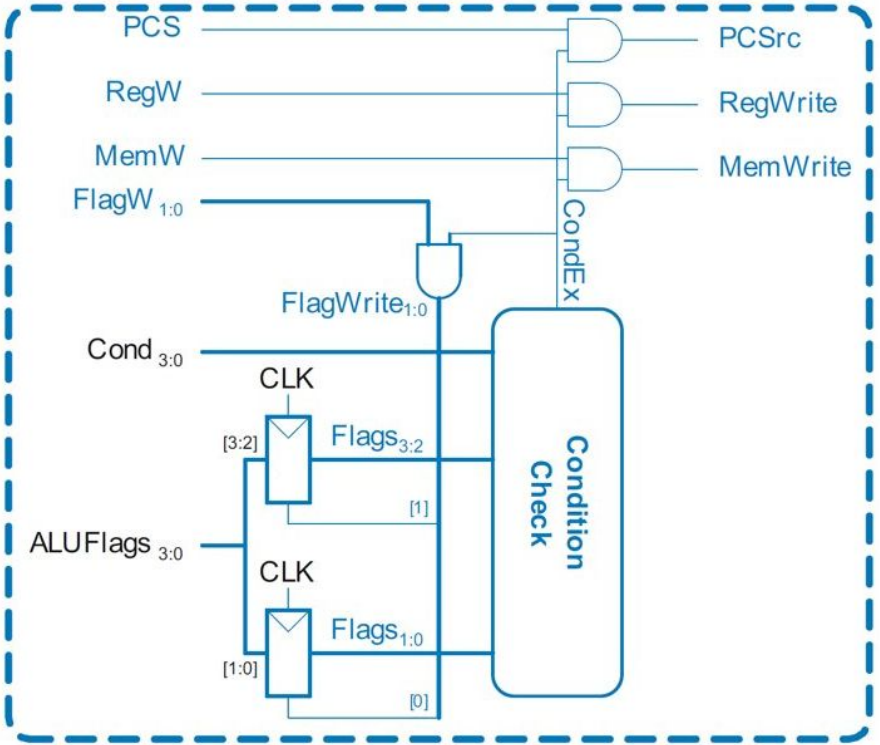
- **FlagW** is 1 (i.e., the instruction's S-bit is 1) AND
- **CondEx** is 1 (the instruction should be executed)



Conditional Logic: Update (Set) Flags

Recall:

- ADD, SUB update **all** Flags
- AND, OR update **NZ only**
- So Flags status register has two write enables: **FlagW1:0**



Review: ALU Decoder

- $FlagW_1 = 1$: NZ ($Flags_{3:2}$) should be saved
- $FlagW_0 = 1$: CV ($Flags_{1:0}$) should be saved

ALUOp	$Funct_{4:1}$ (*cmd*)	$Funct_0$ (*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

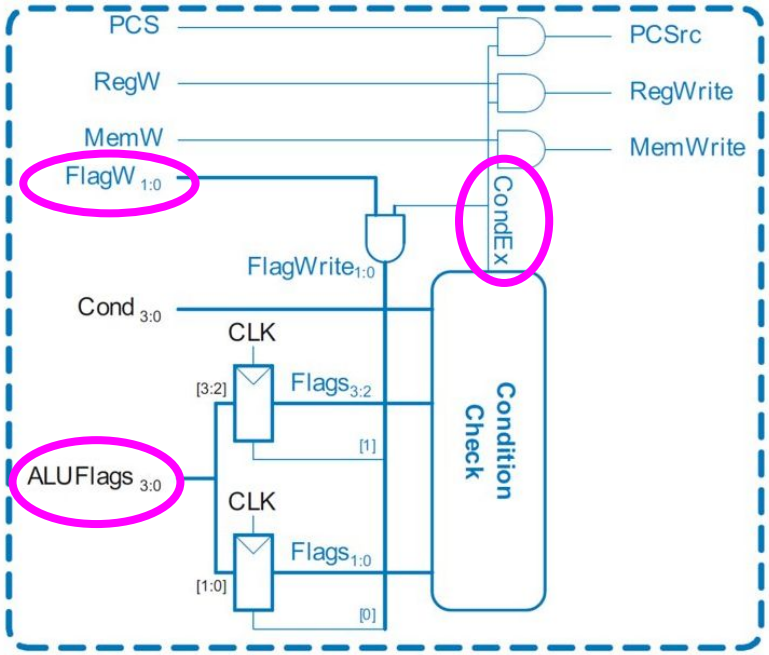
Single-Cycle Control: Cond. Logic

Flags3:0 is the status register

Example: SUBS R5, R6, R7

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

All Flags updated



Single-Cycle Control: Cond. Logic

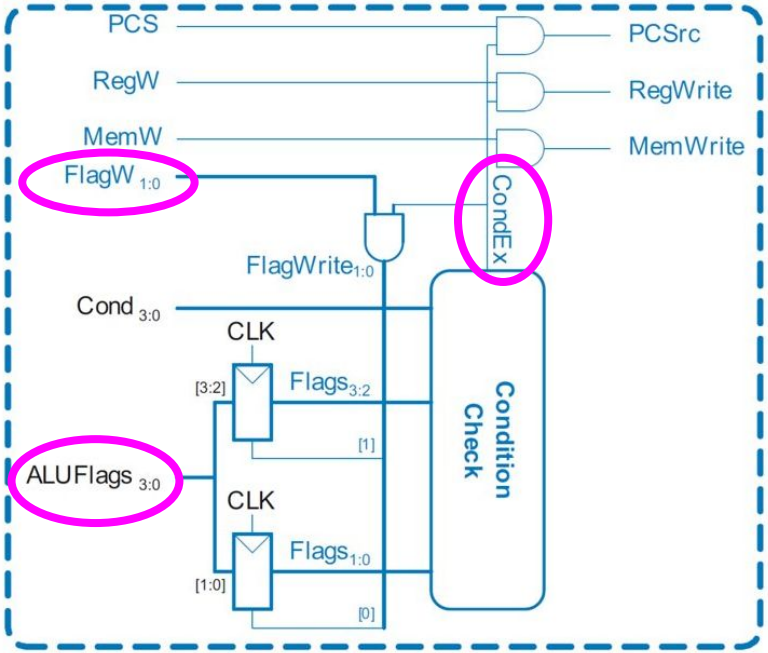
$Flags_{3:0}$ is the status register

Example: ANDS R7, R1, R3

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

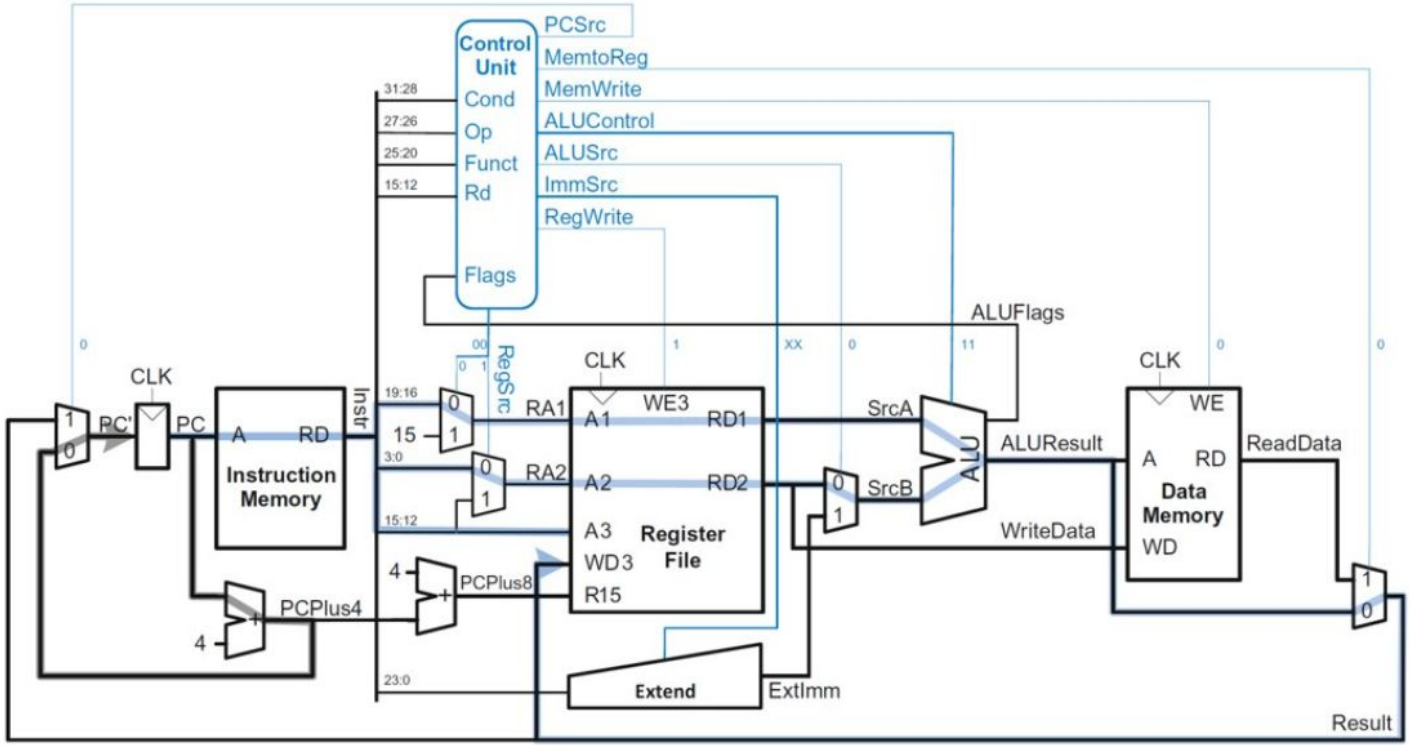
$Flags_{3:0}$ = NZCV

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



Example: ORR

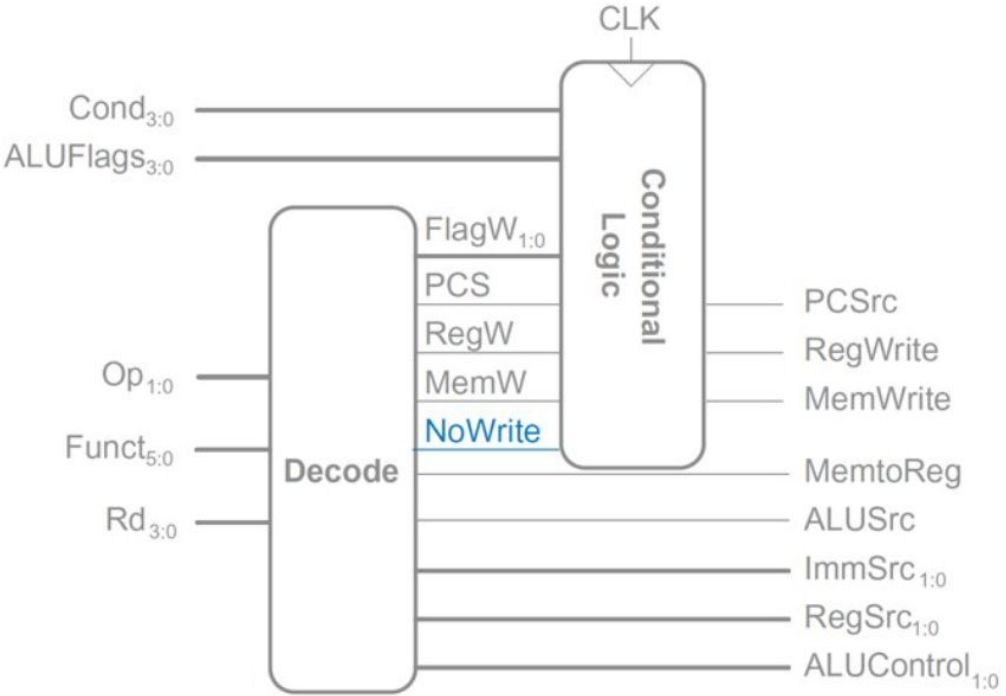
Op	$Funct_5$	$Funct_0$	Type	Branch	MemtoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1



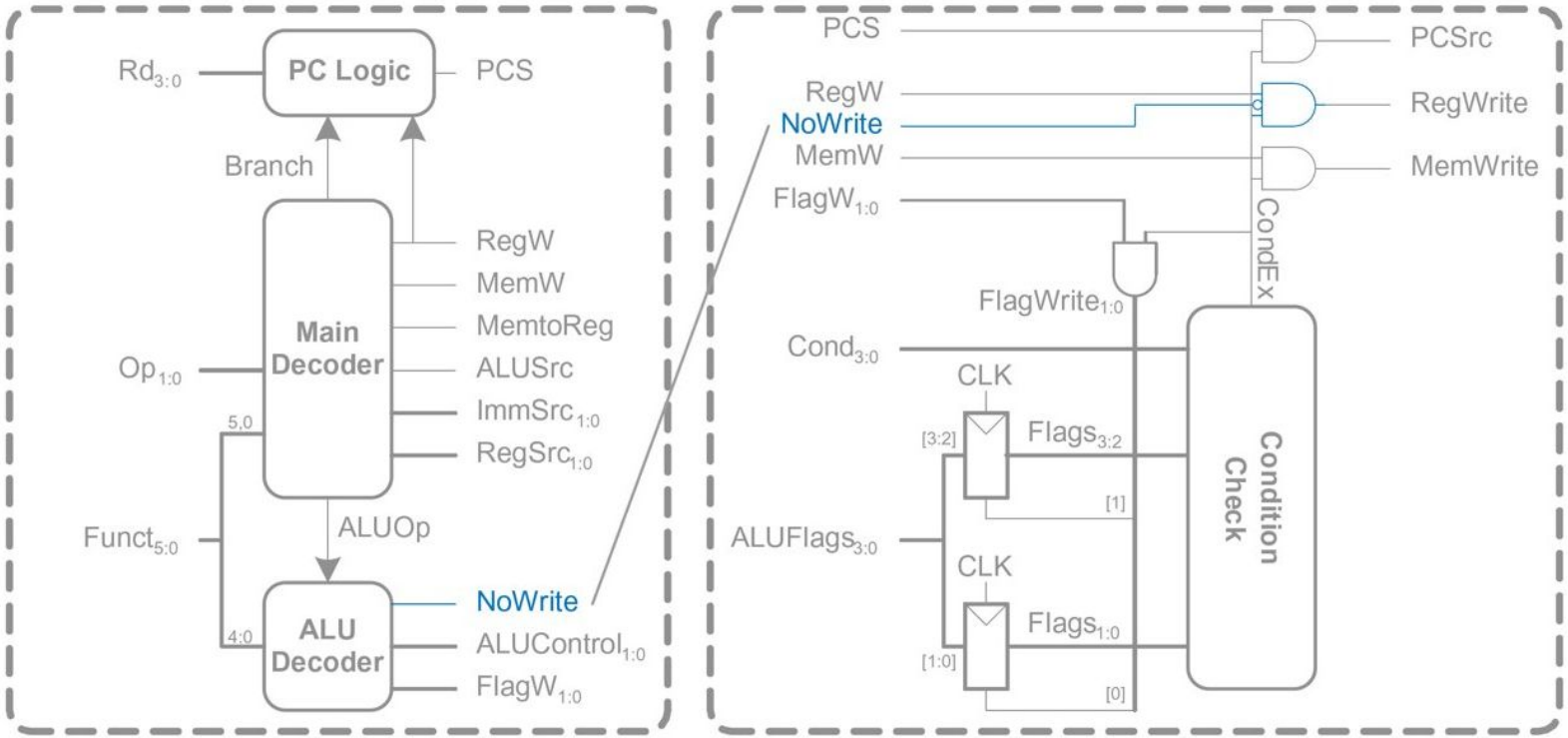
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Extended Functionality: CMP

No change to datapath



Extended Functionality: CMP (look inside)

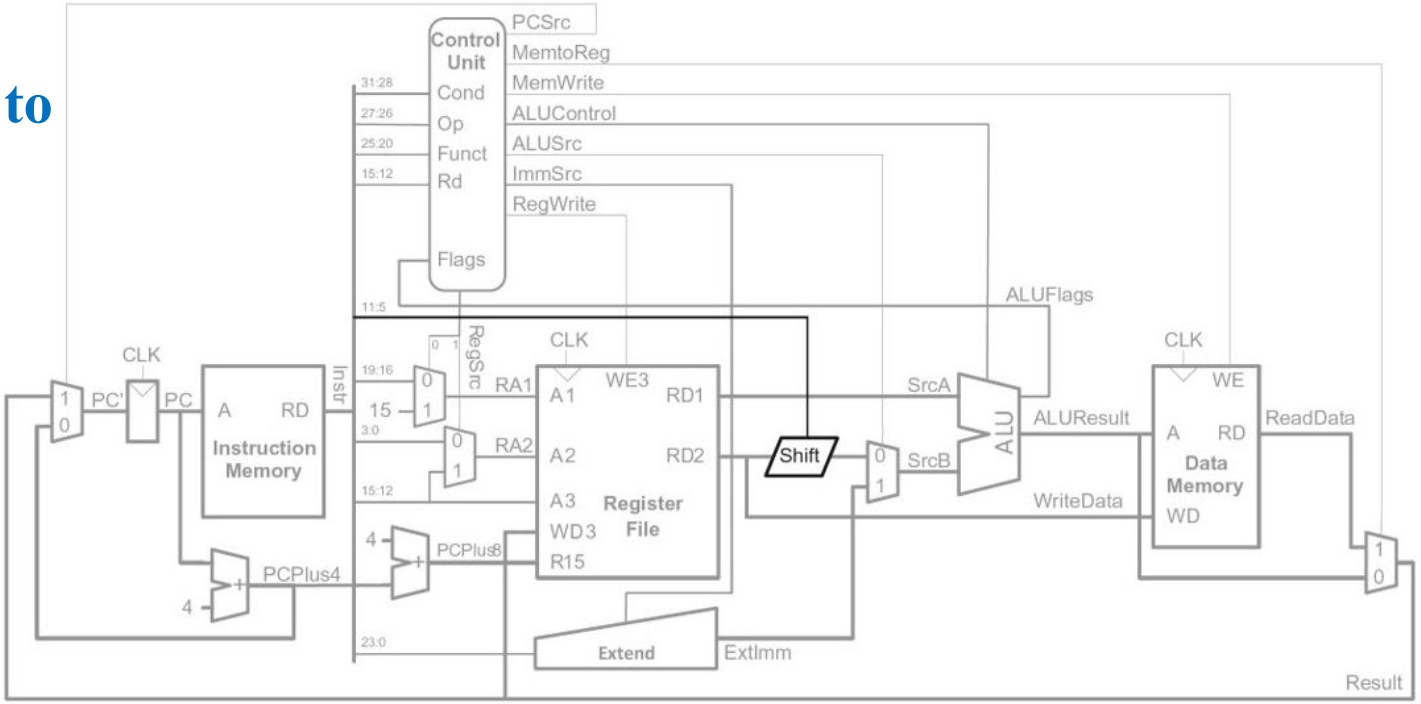


Extended Functionality: CMP

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

No change to controller



ADD R7, R2, R12, LSR #5

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
14	0	0	4	0	2	7	5	01_2	0	12
cond	op	I	cmd	S	rn	rd	shamt5	sh		rm

Processor Performance

Processor Performance

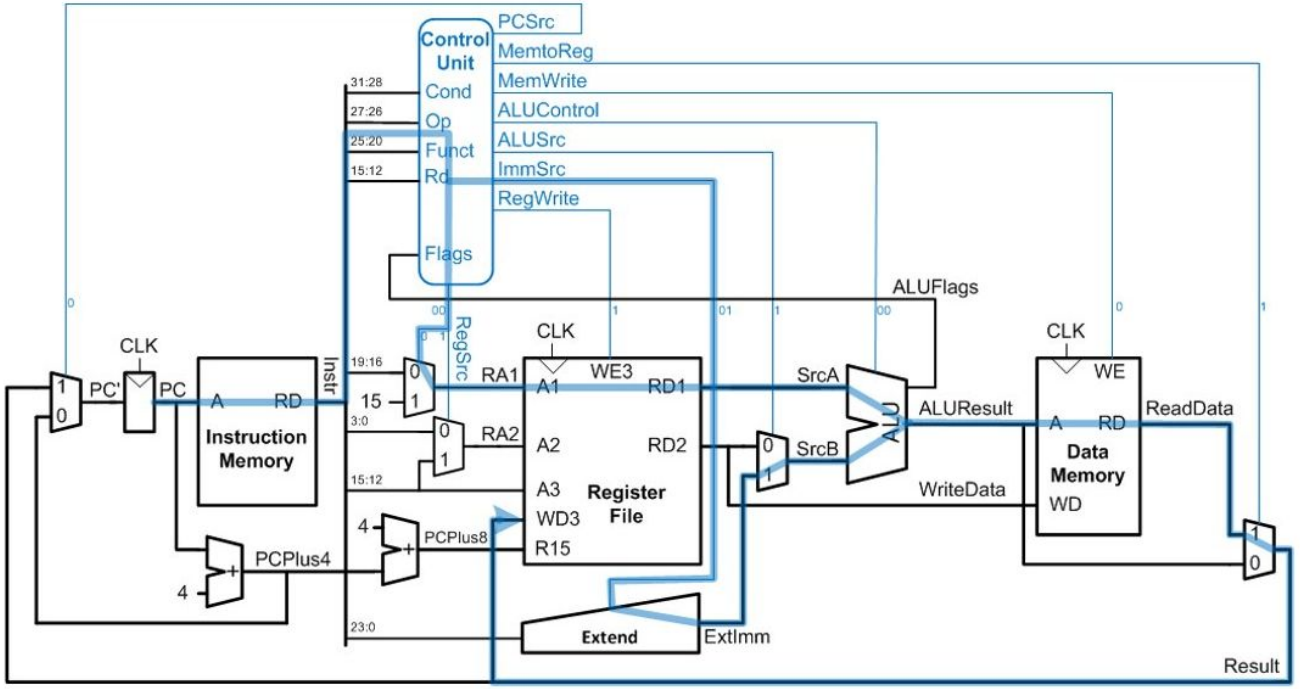
Program Execution Time

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

= # instructions x CPI x TC

Single-Cycle Performance

TC limited by critical path (LDR)



Single-Cycle Performance

- **Single-cycle critical path:**

$$T_{c1} = t_{pcq_PC} + t_{\text{mem}} + t_{\text{dec}} + \max[t_{mux} + t_{RF\text{read}},\ t_{setup} + t_{\text{mux}}] + t_{\text{ALU}} + t_{\text{mem}} + t_{\text{mux}} + t_{RF\text{setup}}$$

- **Typically, limiting paths are:**
  - memory, ALU, register file
  - $T_{c1} = t_{pcq_PC} + 2t_{\text{mem}} + t_{dec} + t_{RF\text{read}} + t_{\text{ALU}} + 2t_{\text{mux}} + t_{RF\text{setup}}$

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	$t_{RF\text{read}}$	100
Register file setup	$t_{RF\text{setup}}$	60

Single-Cycle Performance Example

$T_{c1} = ?$

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

Single-Cycle Performance Example

$$T_{c1} = t_{pcq_PC} + 2t_{\text{mem}} + t_{dec} + t_{RF\text{read}} + t_{\text{ALU}} + 2t_{\text{mux}} + t_{RF\text{setup}}$$
$$= [50 + 2(200) + 70 + 100 + 120 + 2(25) + 60]\ \text{ps}$$
$$= \mathbf{840\ ps}$$

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	$t_{RF\text{read}}$	100
Register file setup	$t_{RF\text{setup}}$	60

Single-Cycle Performance Example

Program with 100 billion instructions:

Execution Time = # instructions x CPI x T_C

$= (100 \times 10^9)(1)(840 \times 10^{-12}$ s)

= 84 seconds

Upnext

Multicycle ARM Processor

Spring 2025

Rahul Bhadani

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE