E85 Digital Design & Computer Engineering

Lecture 21:

Multicycle Processor

HARVEY MUDD COLLEGE

Lecture 21

- **Multicycle Processor**

Multicycle ARM Processor

- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (`LDR`)
  - - separate memories for instruction and data
  - - 3 adders/ALUs
- **Multicycle processor addresses these issues by breaking instruction into shorter steps**
  - o shorter instructions take fewer steps
  - o can re-use hardware
  - o cycle time is faster

Multicycle ARM Processor

- **Single-cycle:**

  + simple

  - cycle time limited by longest instruction (`LDR`)

  - separate memories for instruction and data

  - 3 adders/ALUs

- **Multicycle:**

  + higher clock speed

  + simpler instructions run faster

  + reuse expensive hardware on multiple cycles

  - sequencing overhead paid many times

Multicycle ARM Processor

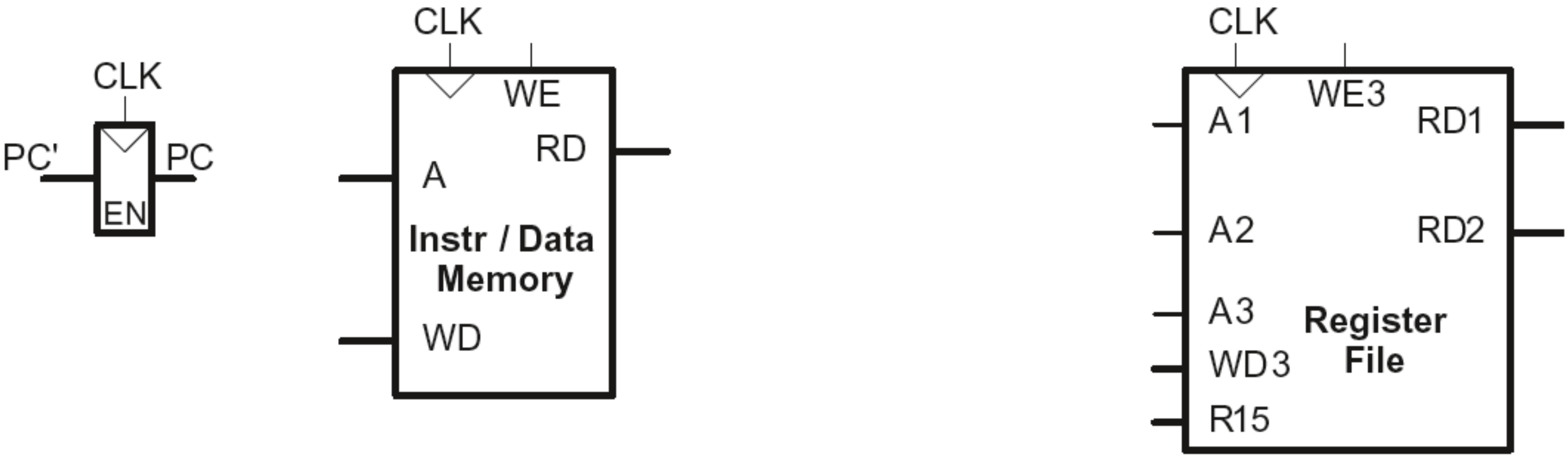
- **Single-cycle:**
  - + simple
  - - cycle time limited by longest instruction (`LDR`)
  - - separate memories for instruction and data
  - - 3 adders/ALUs
- **Multicycle:**
  - + higher clock speed
  - + simpler instructions run faster
  - + reuse expensive hardware on multiple cycles
  - - sequencing overhead paid many times

Same design steps as single-cycle:

- **first datapath**
- **then control**

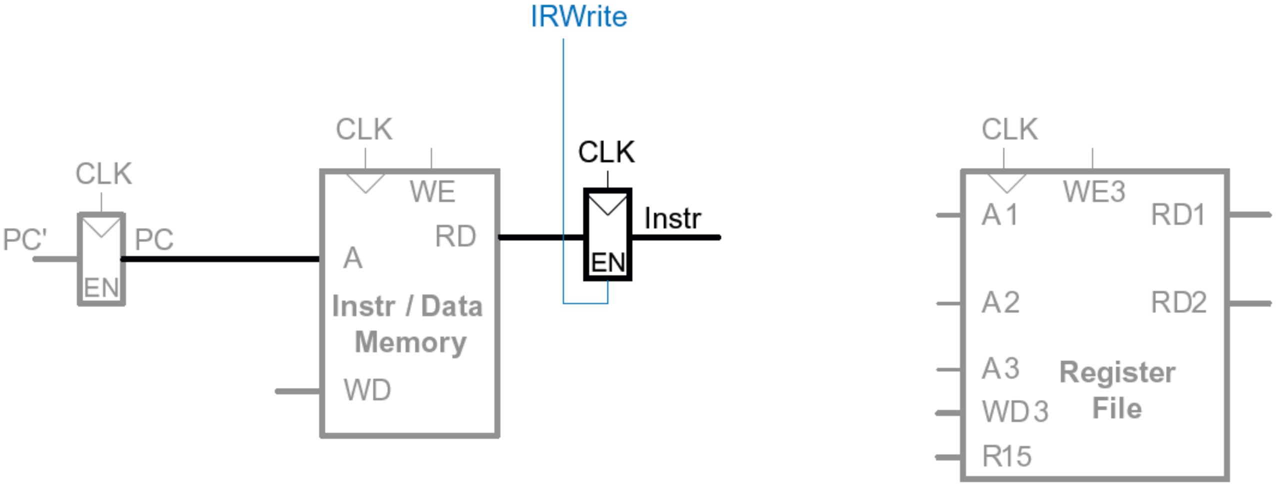
Multicycle State Elements

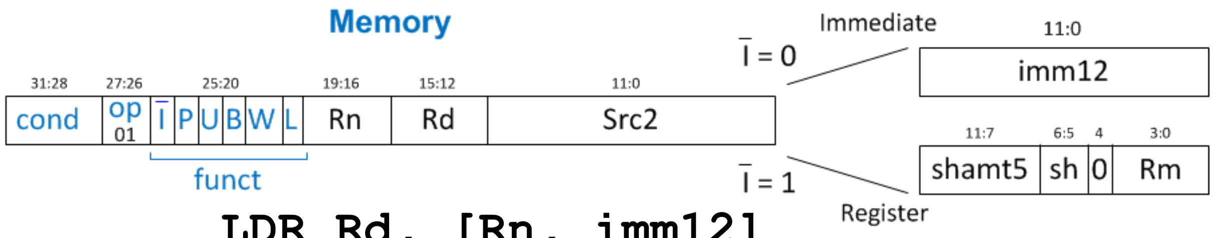
Replace Instruction and Data memories with a single unified memory – more realistic



Multicycle Datapath: Instruction Fetch

STEP 1: Fetch instruction

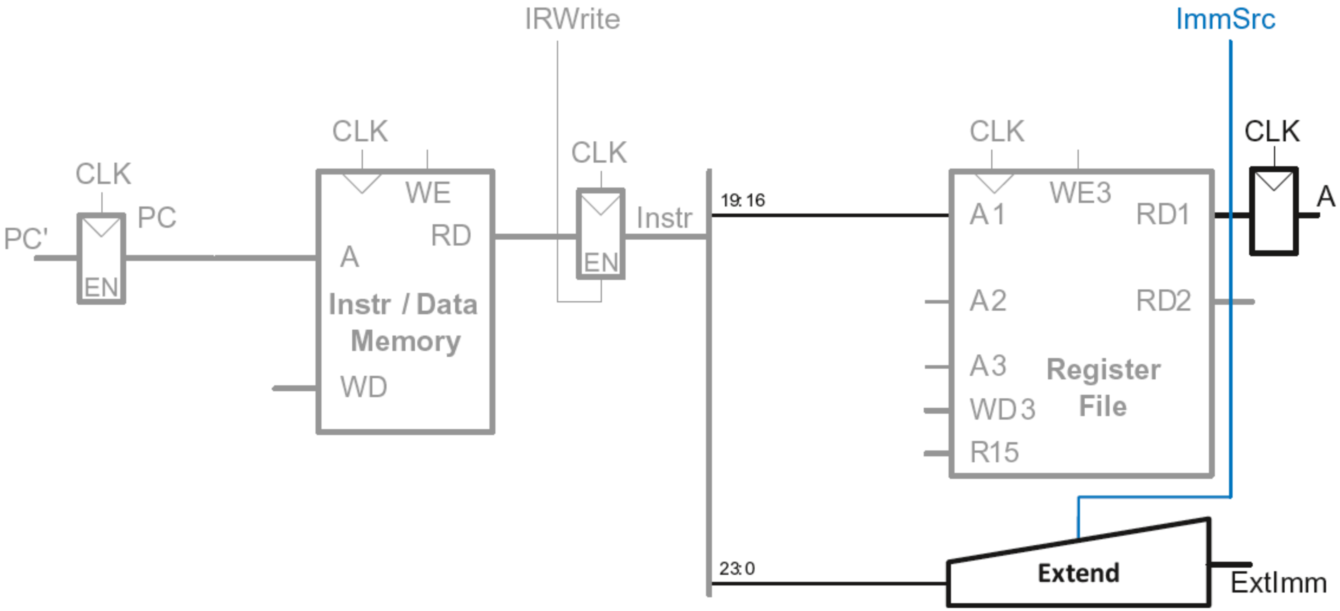




`LDR Rd, [Rn, imm12]`

Multicycle Datapath: LDR Register Read

STEP 2: Read source operands from RF



Memory

31:28	27:26	25:20						19:16	15:12	11:0
cond	op 01	Ī	P	U	B	W	L	Rn	Rd	Src2
		funct								

Ī = 0 Immediate

11:0
imm12

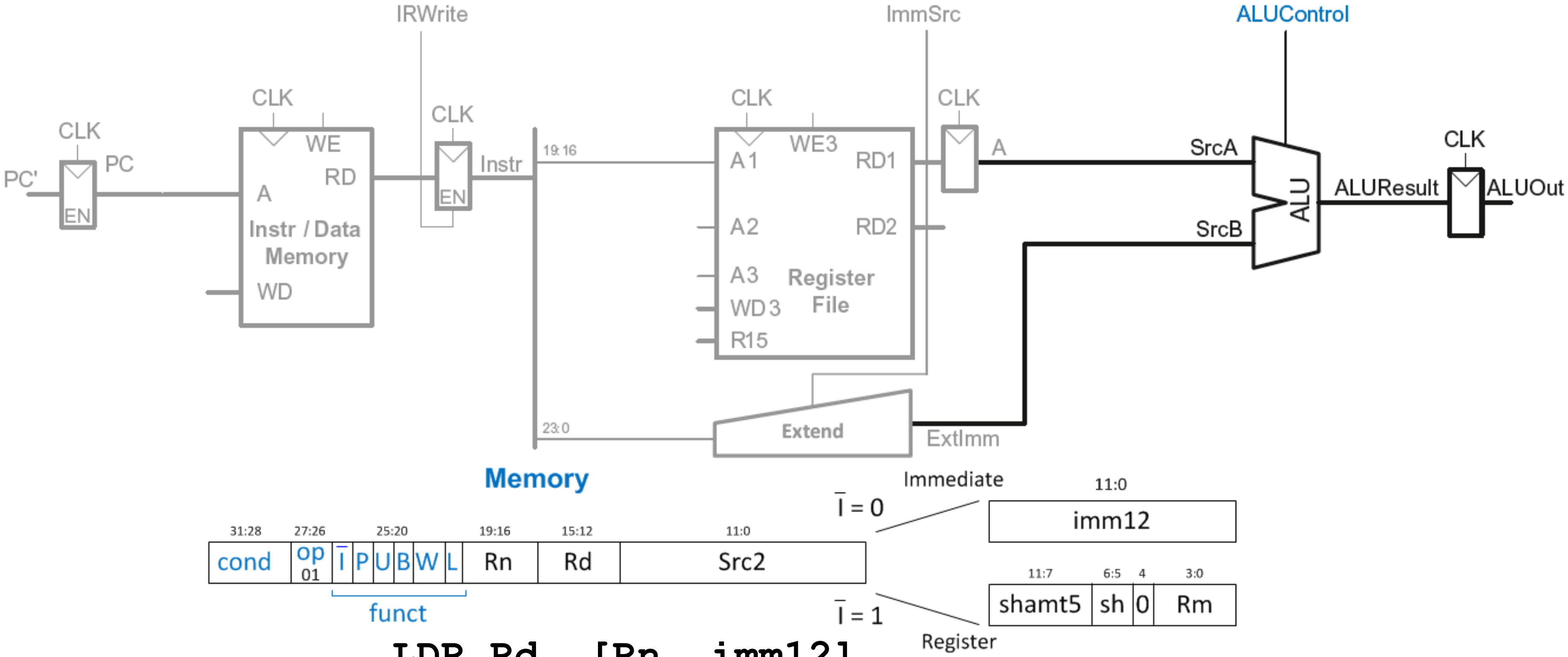
Ī = 1 Register

11:7	6:5	4	3:0
shamt5	sh	0	Rm

LDR Rd, [Rn, imm12]

Multicycle Datapath: LDR Address

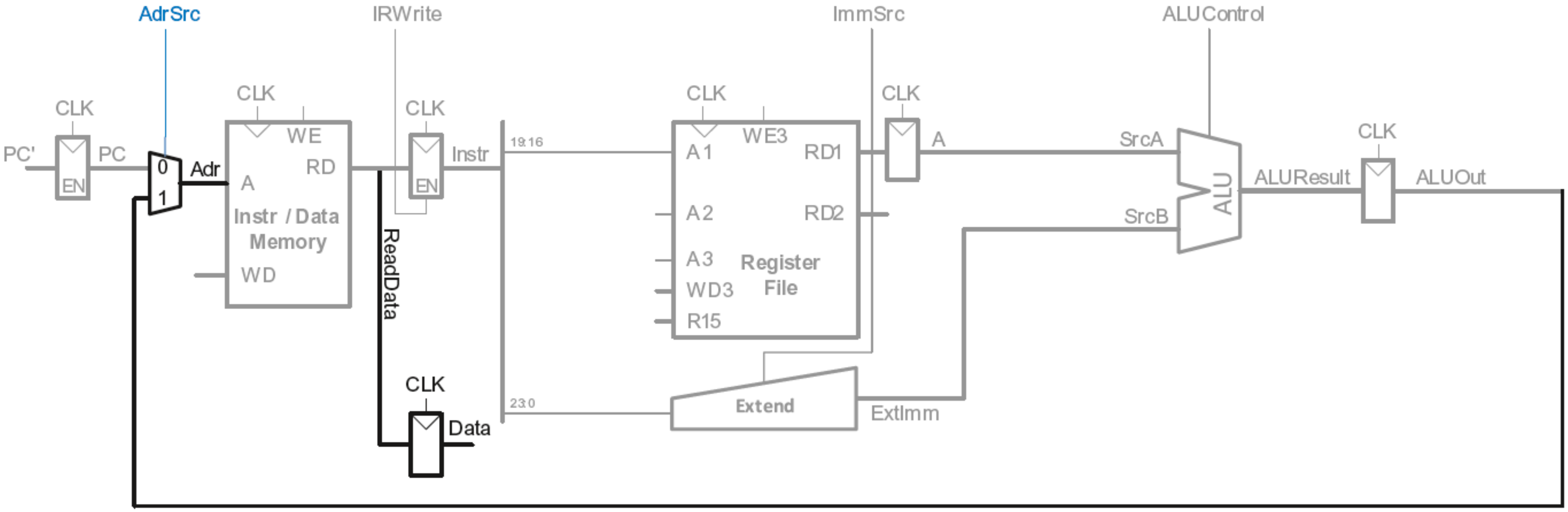
STEP 3: Compute the memory address

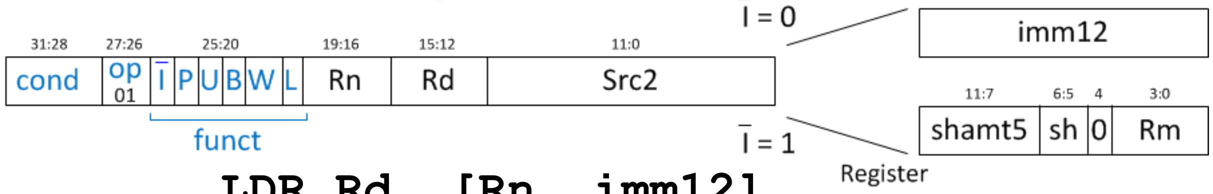


LDR Rd, [Rn, imm12]

Multicycle Datapath: `LDR` Memory Read

STEP 4: Read data from memory

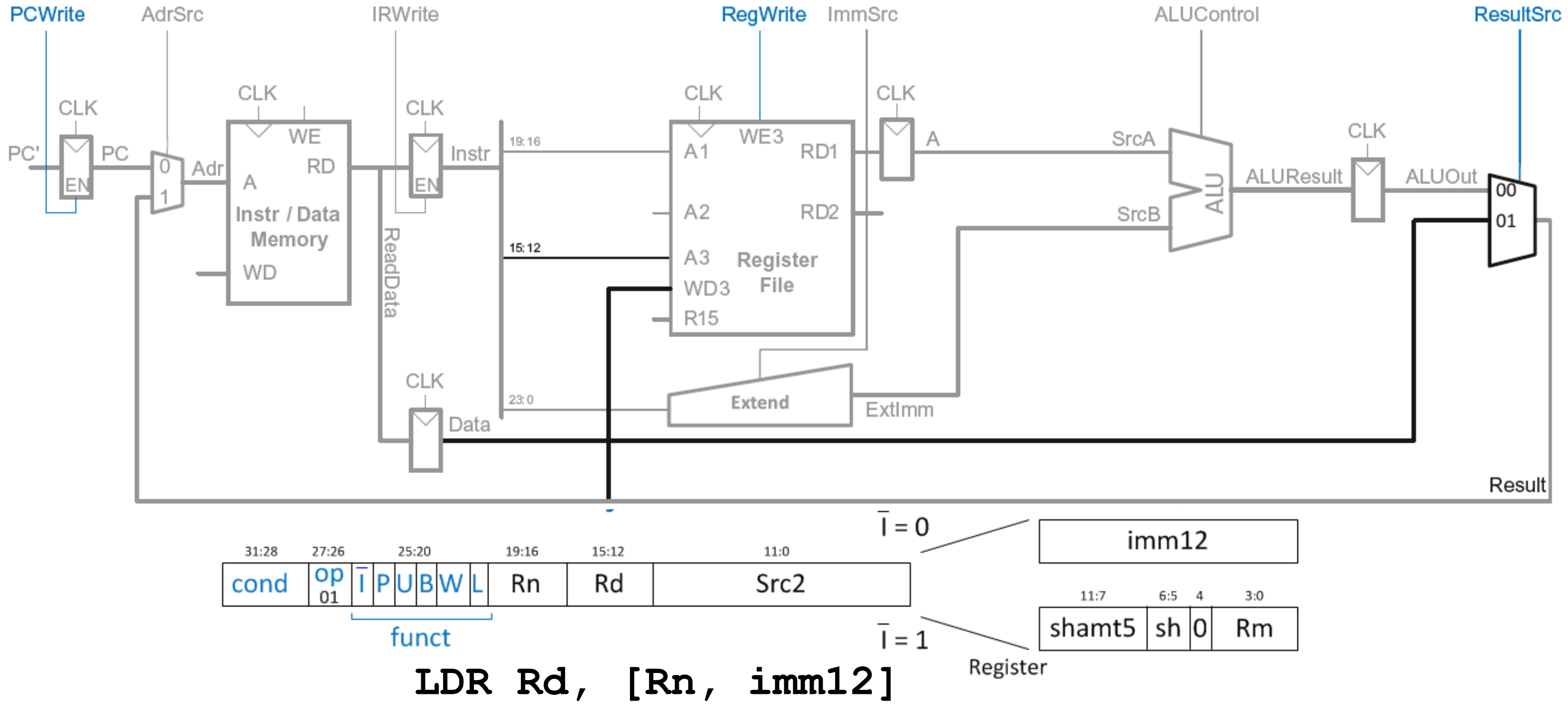




`LDR Rd, [Rn, imm12]`

Multicycle Datapath: LDR Write Register

STEP 5: Write data back to register file

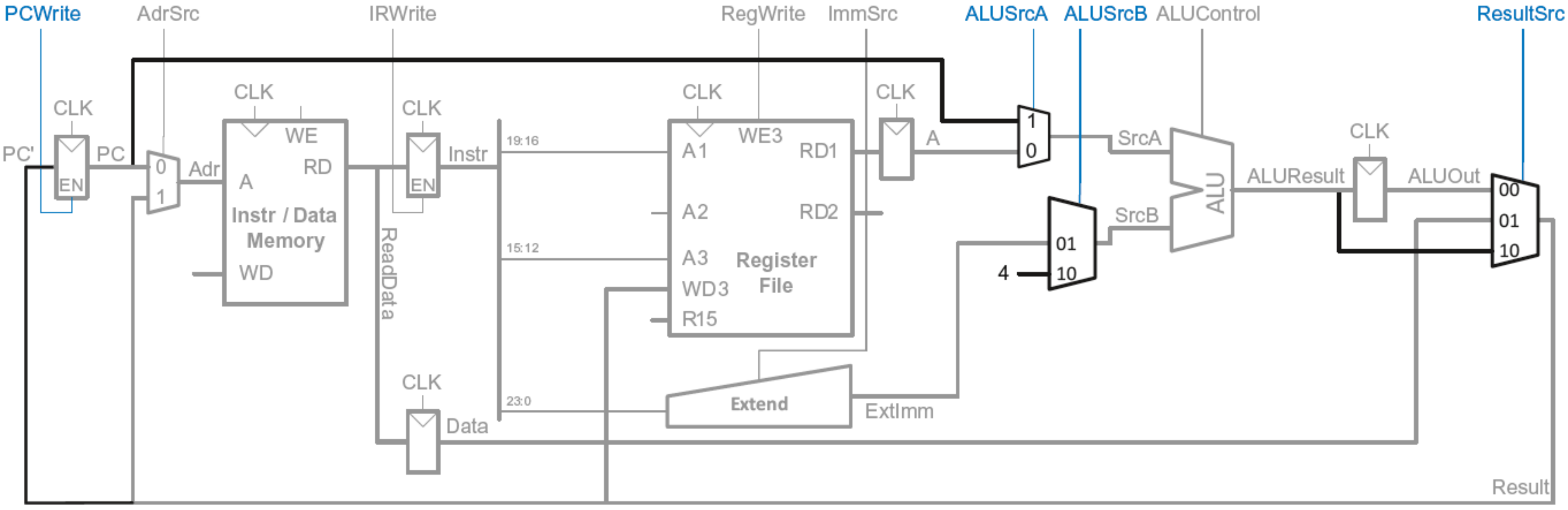


`LDR Rd, [Rn, imm12]`

Multicycle Datapath: Increment PC

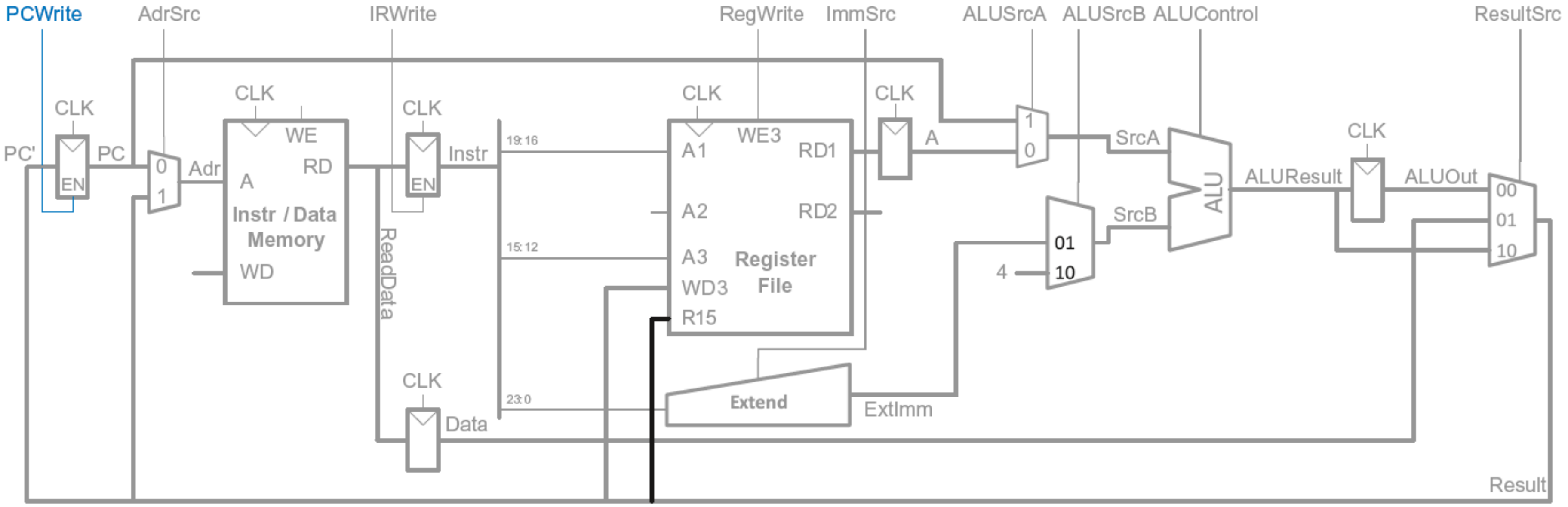
Meanwhile: Increment PC

Concurrent with fetching instruction



Multicycle Datapath: Access to PC

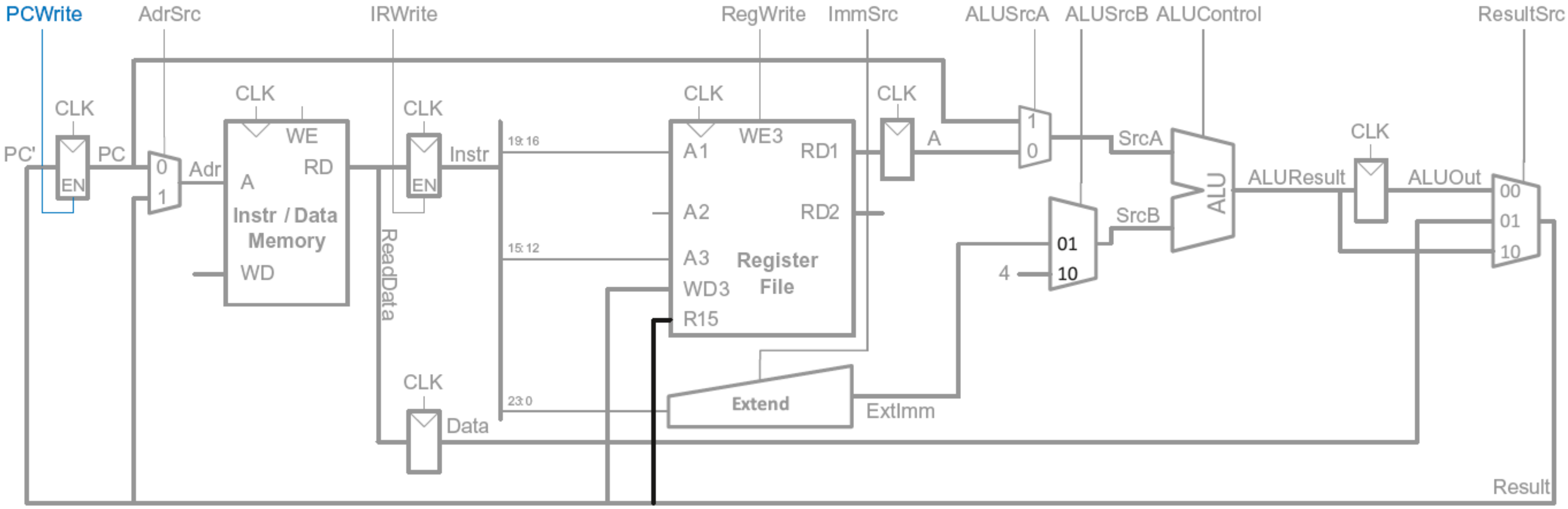
PC can be read/written by instruction



Multicycle Datapath: Access to PC

PC can be read/written by instruction

- **Read:** R15 (PC+8) available in Register File



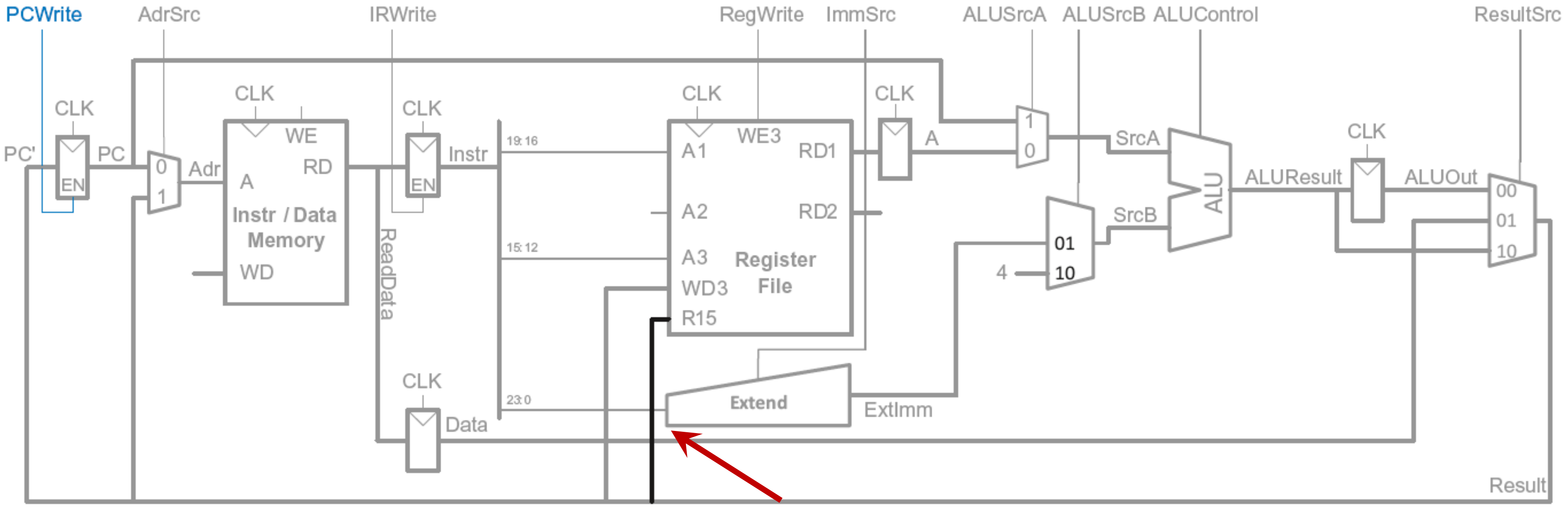
Multicycle Datapath: Read to PC (R15)

Example: `ADD R1, **R15**, R2`

Multicycle Datapath: Read to PC (R15)

Example: `ADD R1, **R15**, R2`

- R15 needs to be read as PC+8 from Register File (RF) in 2nd step
- PC+4 was computed in 1st step
- So (also in 2nd step) ALU computes (PC+4) + 4 for R15 input



Multicycle Datapath: Read to PC (R15)

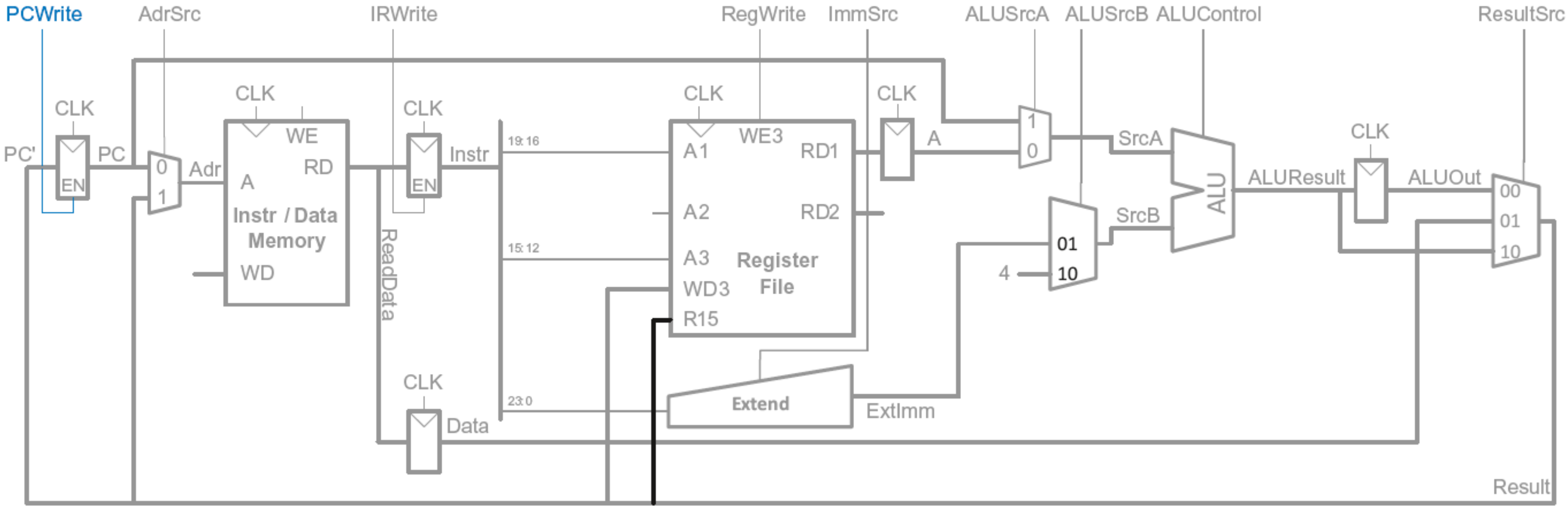
Example: `ADD R1, R15, R2`

- R15 needs to be read as PC+8 from Register File (RF) in 2nd step
- PC+4 was computed in 1st step
- So (also in 2nd step) ALU computes (PC+4) + 4 for R15 input
  - *SrcA* = *PC* (which was already updated in step 1 to PC+4)
  - *SrcB* = 4
  - *ALUResult* = *PC* + 8
- ALUResult is fed to R15 input port of RF in 2nd step (which is then routed to RD1 output of RF)

Multicycle Datapath: Access to PC

PC can be read/written by instruction

- **Read:** R15 (PC+8) available in Register File
- **Write:** Be able to write result of instruction to PC



Multicycle Datapath: Write to PC (R15)

Example: `SUB` **`R15`**`, R8, R3`

Multicycle Datapath: Write to PC (R15)

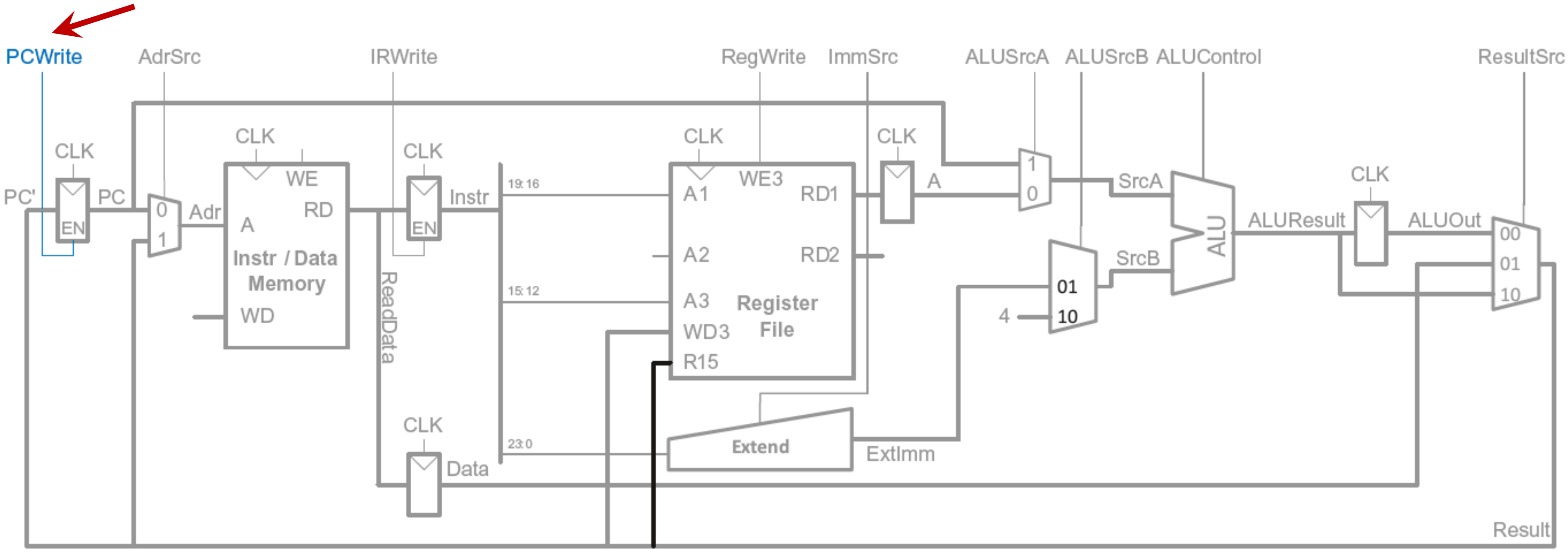
Example: SUB **R15**, R8, R3

- Result of instruction needs to be written to the PC register
- ALUResult already routed to the PC register, just assert PCWrite

Multicycle Datapath: Write to PC (R15)

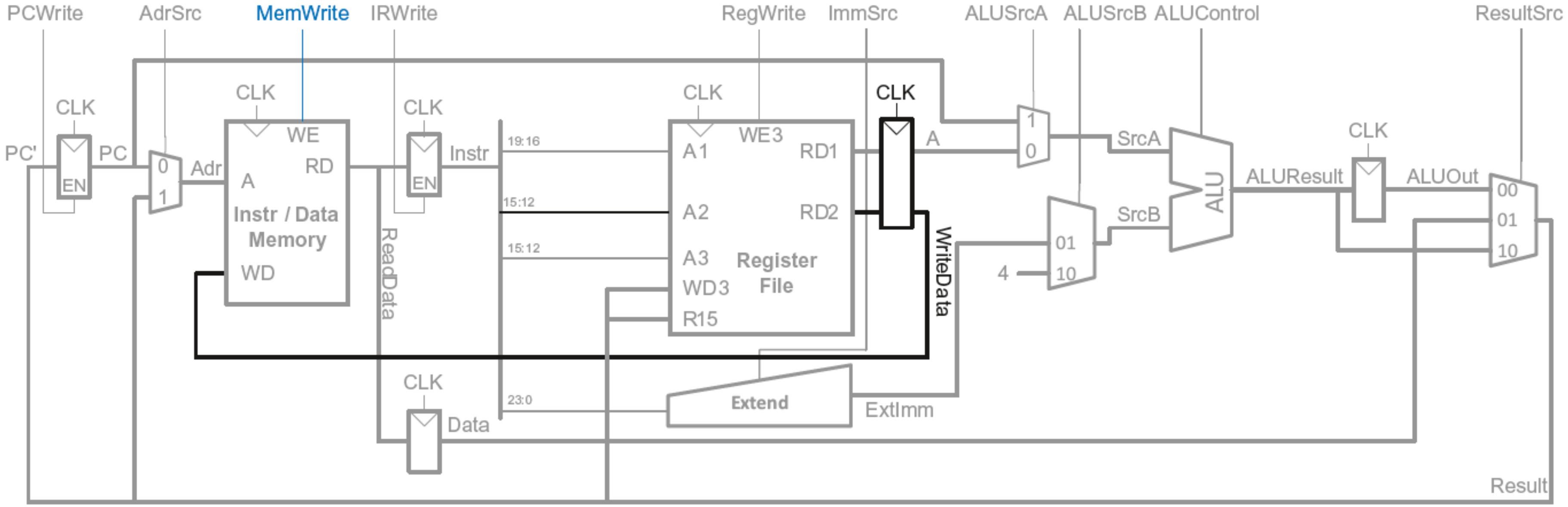
Example: `SUB **R15**, R8, R3`

- Result of instruction needs to be written to the PC register
- ALUResult already routed to the PC register, just assert PCWrite



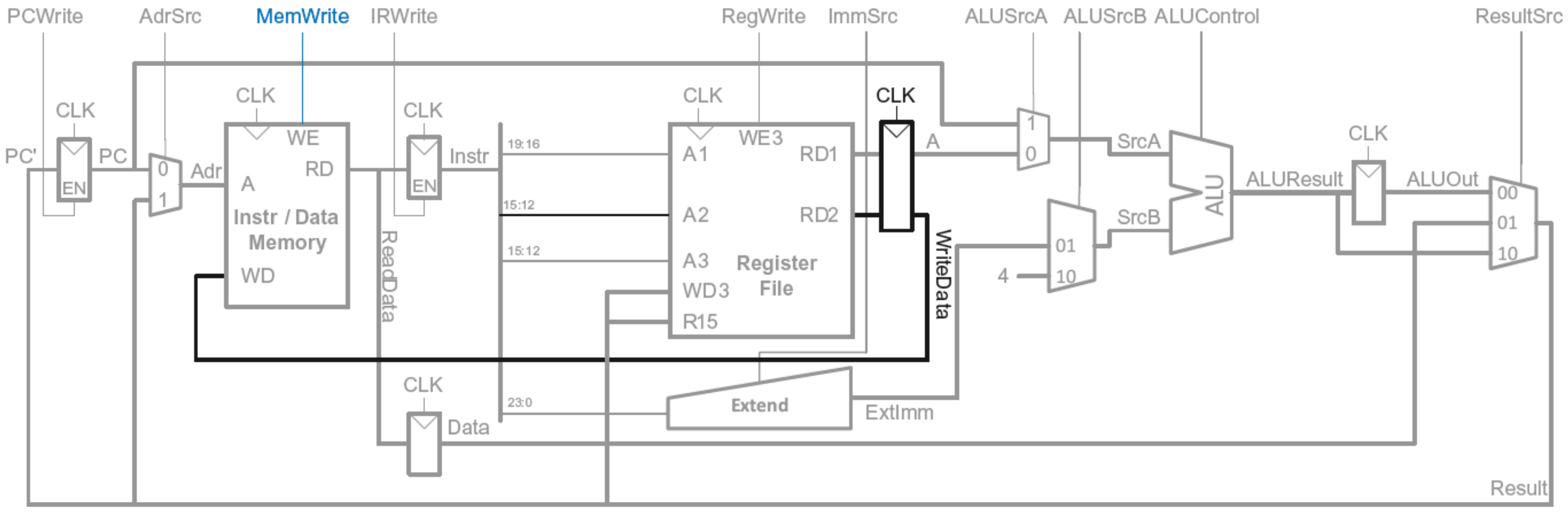
Multicycle Datapath: STR

Write data in Rn to memory



Multicycle Datapath: Data-processing

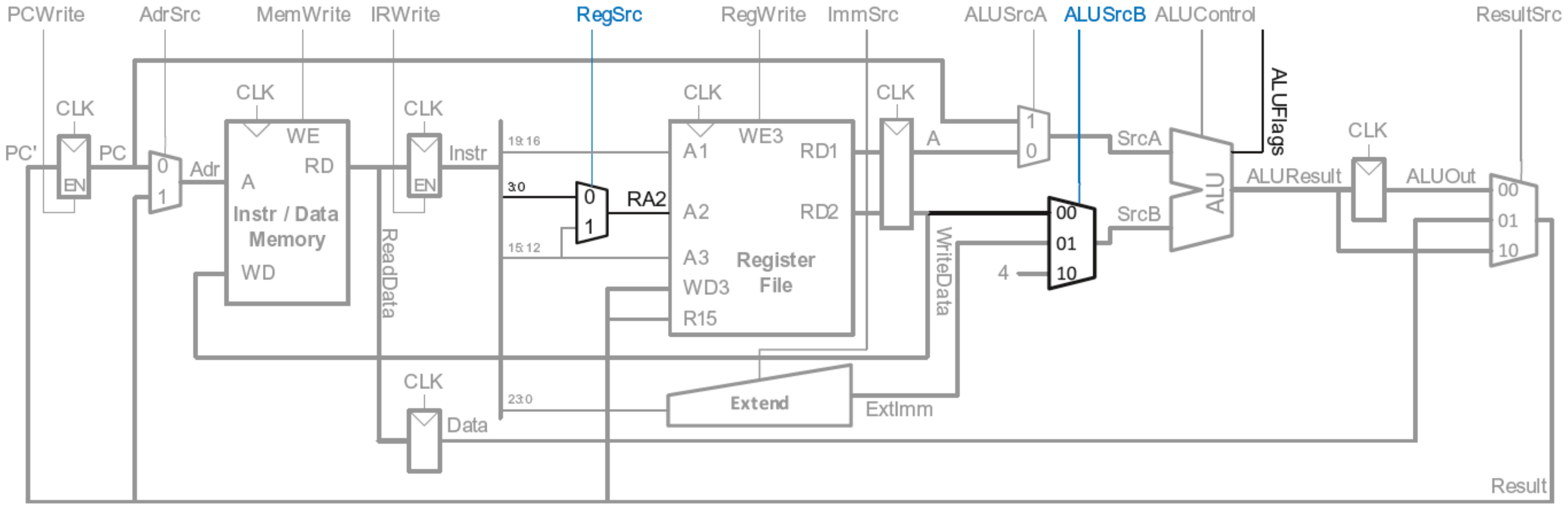
With immediate addressing (i.e., an immediate *Src2*), no additional changes needed for datapath



Multicycle Datapath: Data-processing

With register addressing (register *Src2*):

Read from Rn and Rm

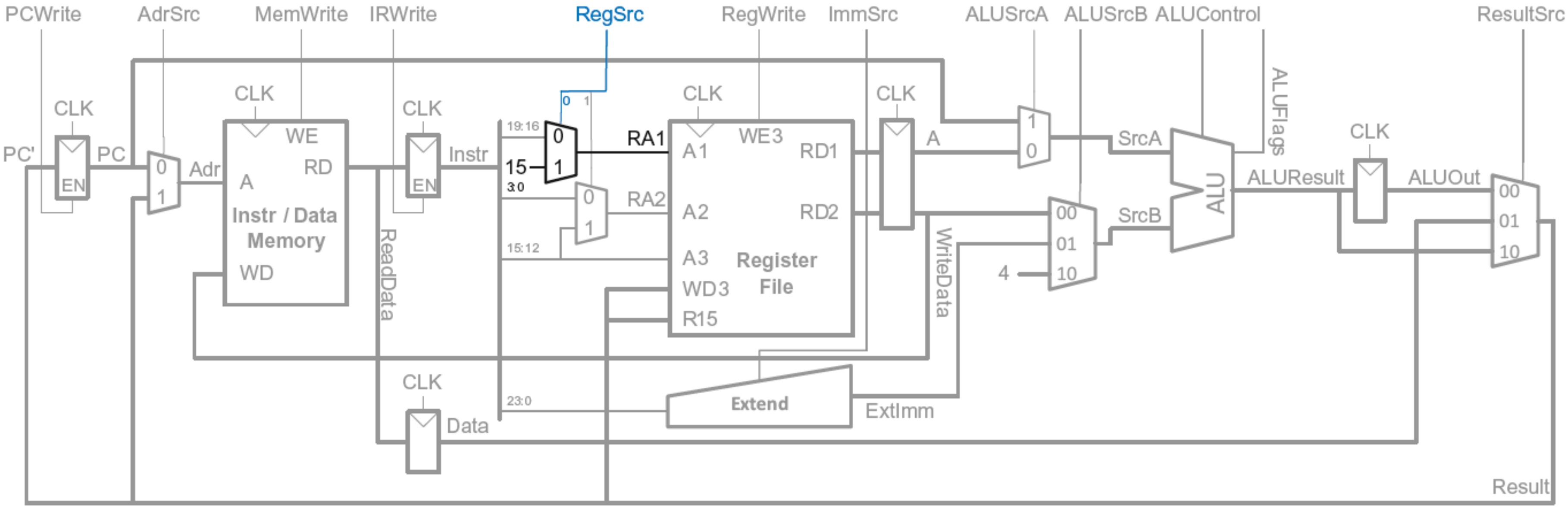


Multicycle Datapath: B

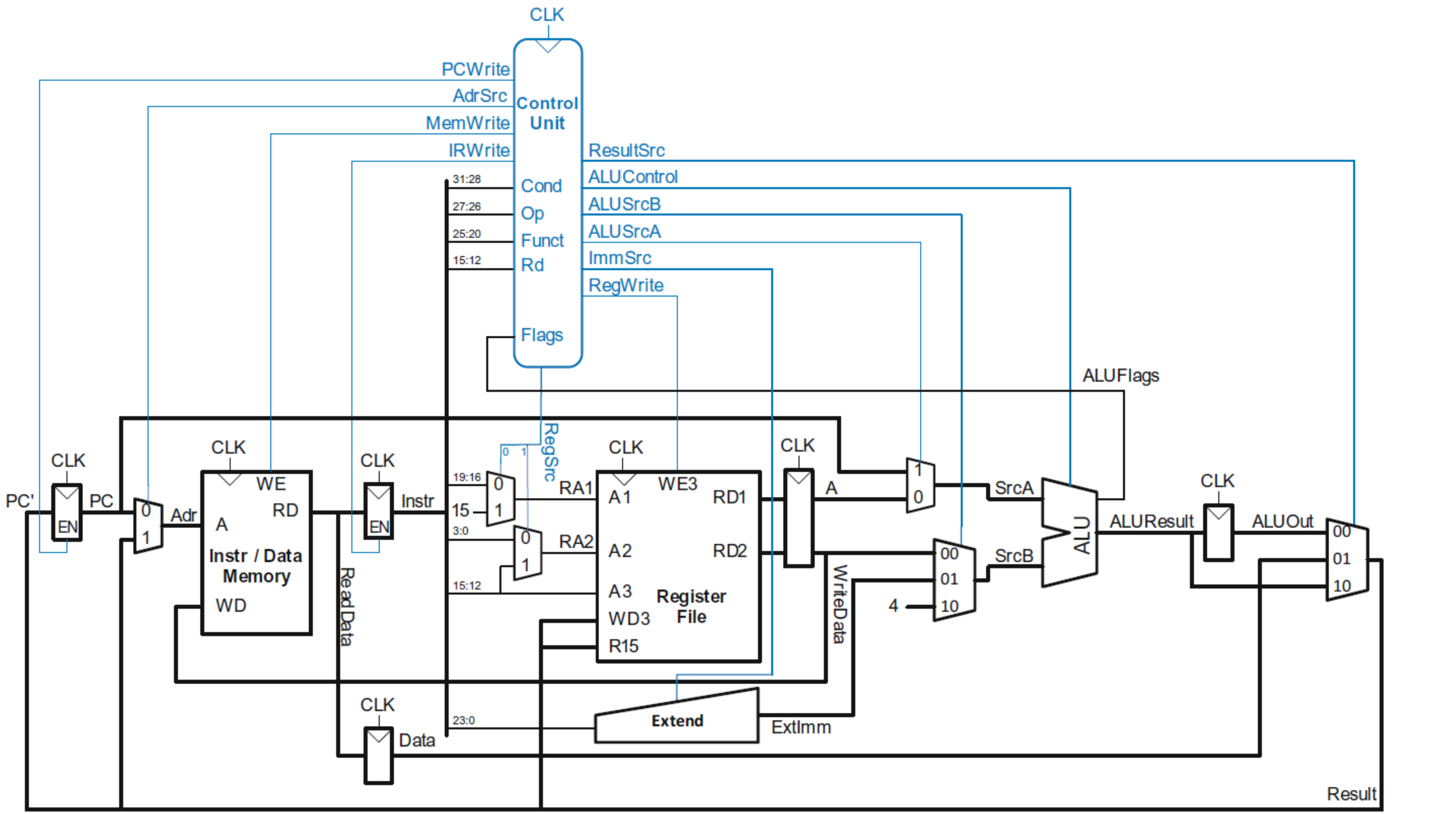
Calculate branch target address:

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

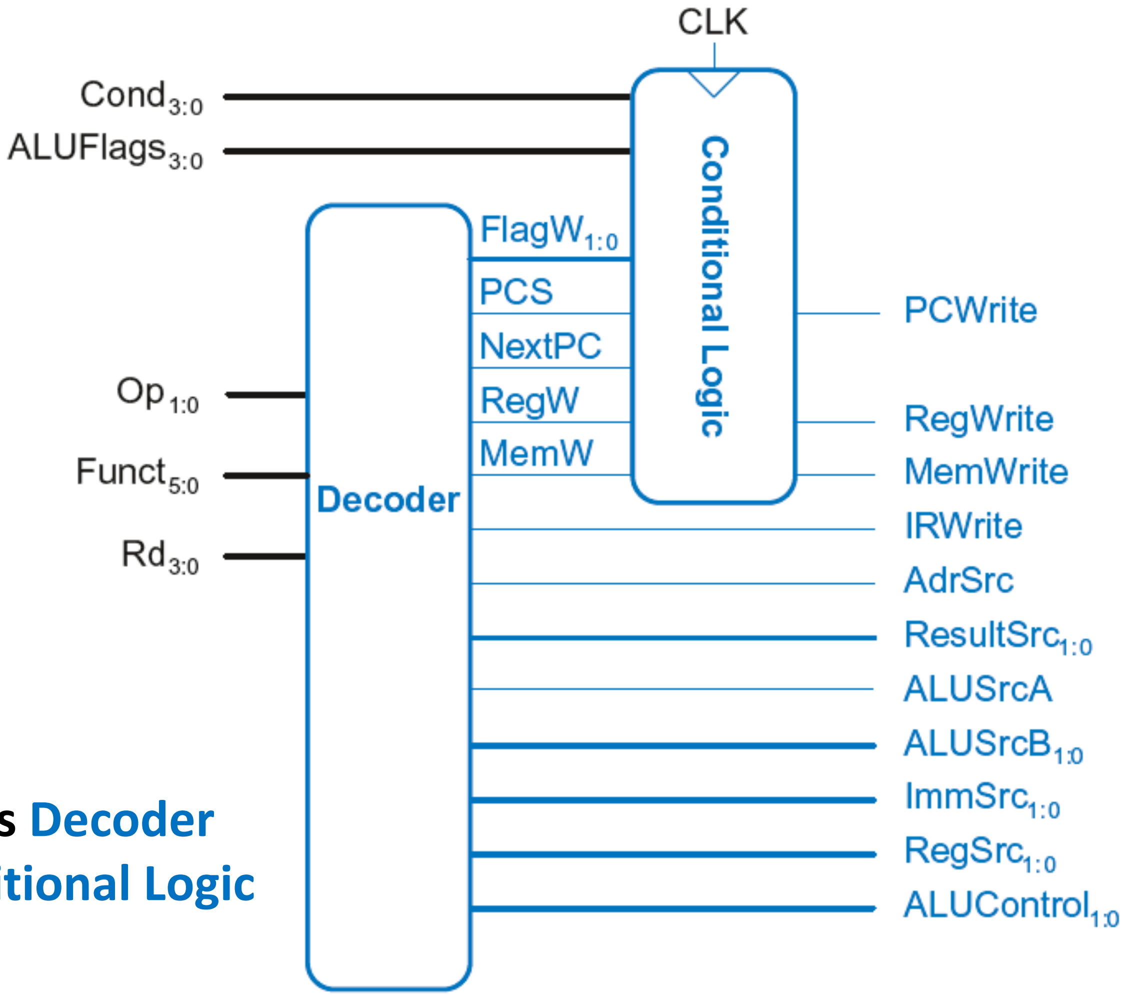
ExtImm = *Imm24* << *2* and sign-extended



Multicycle ARM Processor

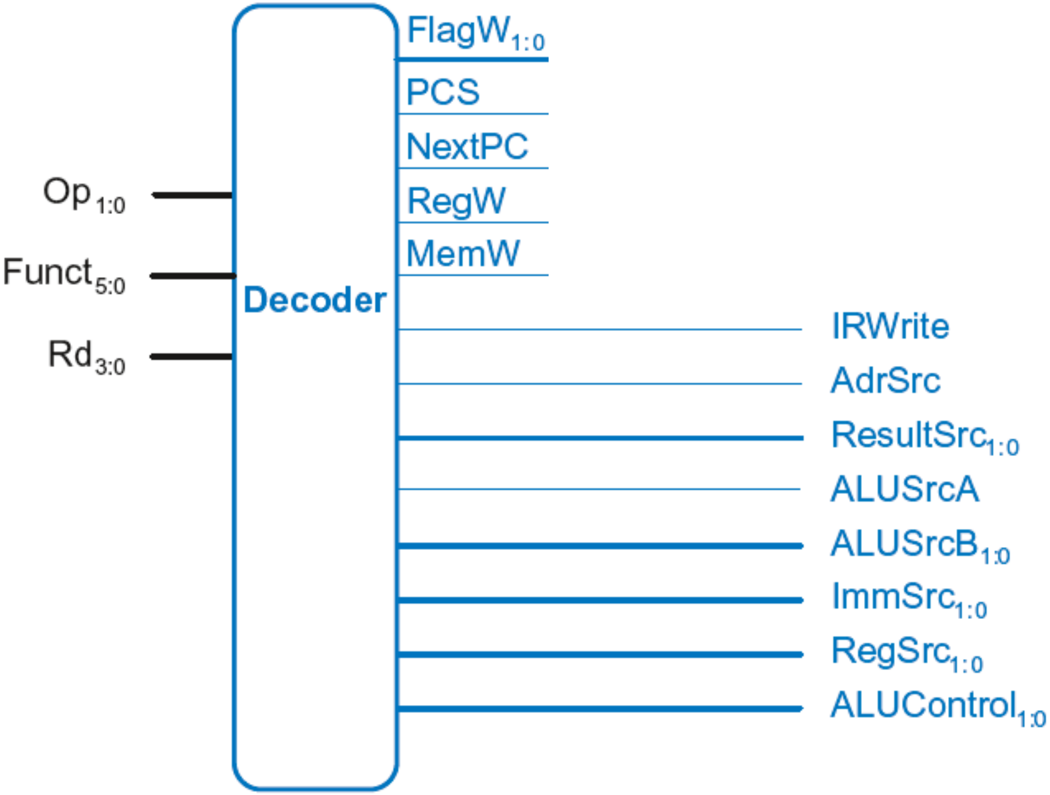


Multicycle Control

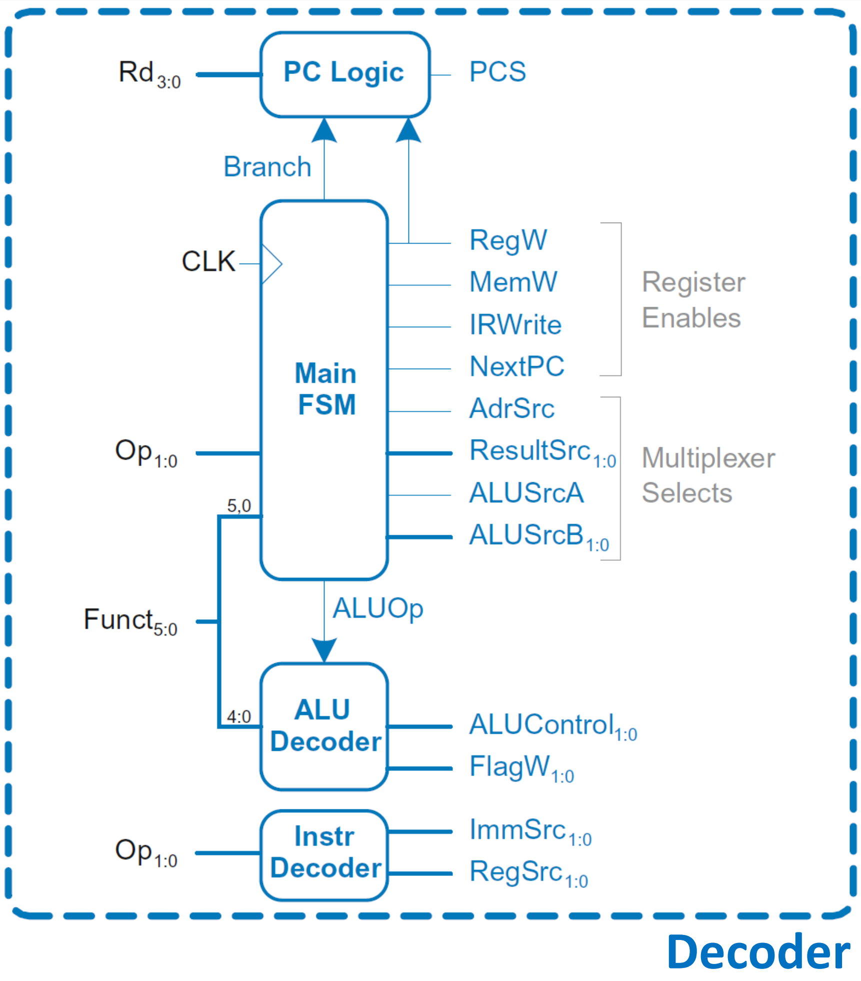


- **First, discuss Decoder**
- **Then, Conditional Logic**

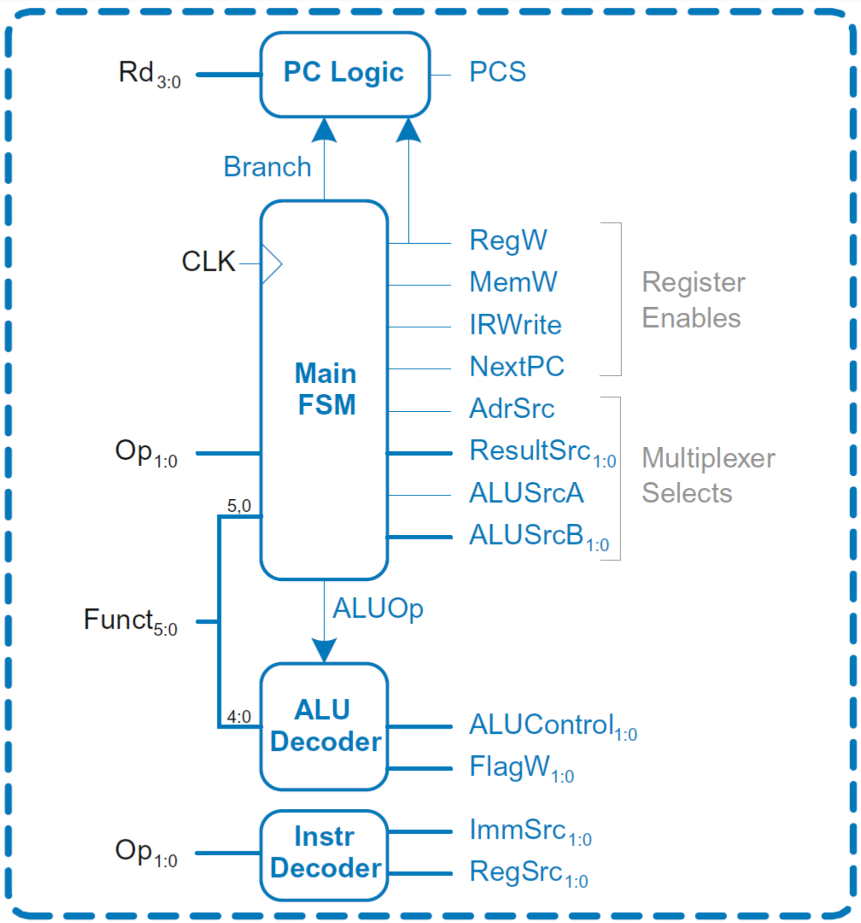
Multicycle Control: Decoder



Multicycle Control: Decoder

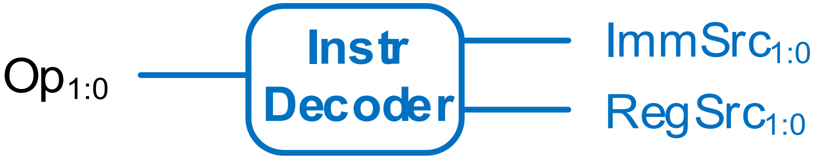


Multicycle Control: Decoder



ALU Decoder and PC Logic same as single-cycle

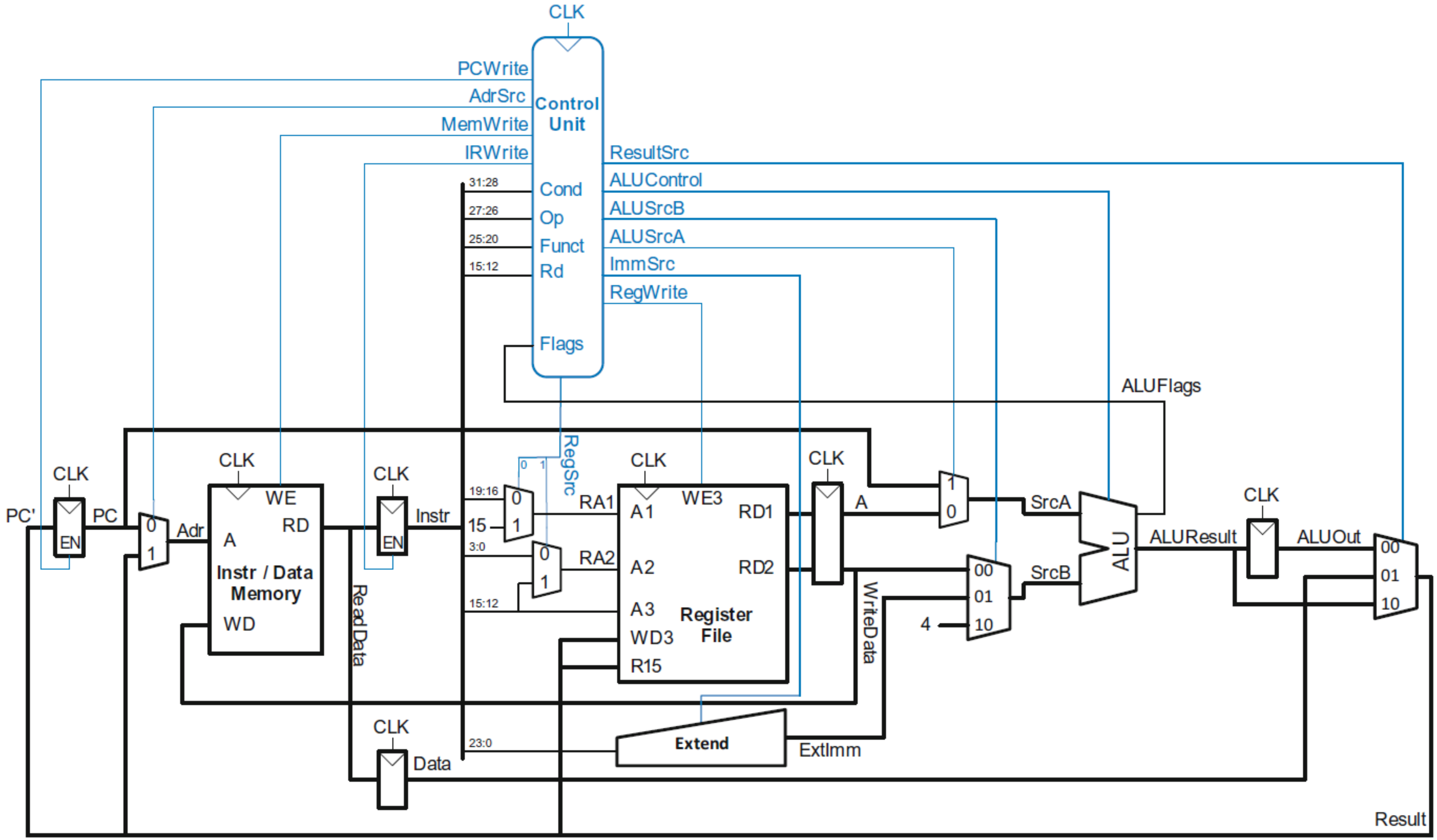
Multicycle Control: Instr Decoder



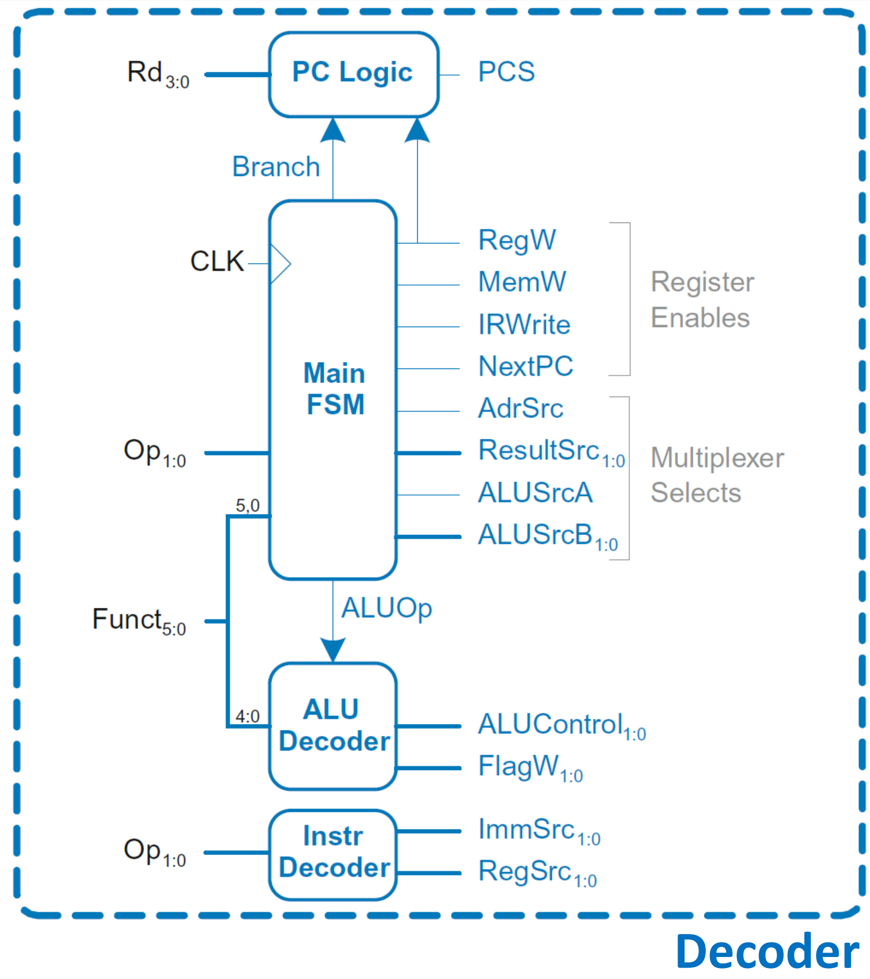
$$RegSrc_0 = (Op == 10_2)$$
$$RegSrc_1 = (Op == 01_2)$$
$$ImmSrc_{1:0} = Op$$

Instruction	Op	$Funct_5$	$Funct_0$	$RegSrc_0$	$RegSrc_1$	$ImmSrc_{1:0}$
LDR	01	X	1	0	X	01
STR	01	X	0	0	1	01
DP immediate	00	1	X	0	X	00
DP register	00	0	X	0	0	00
B	10	X	X	1	X	10

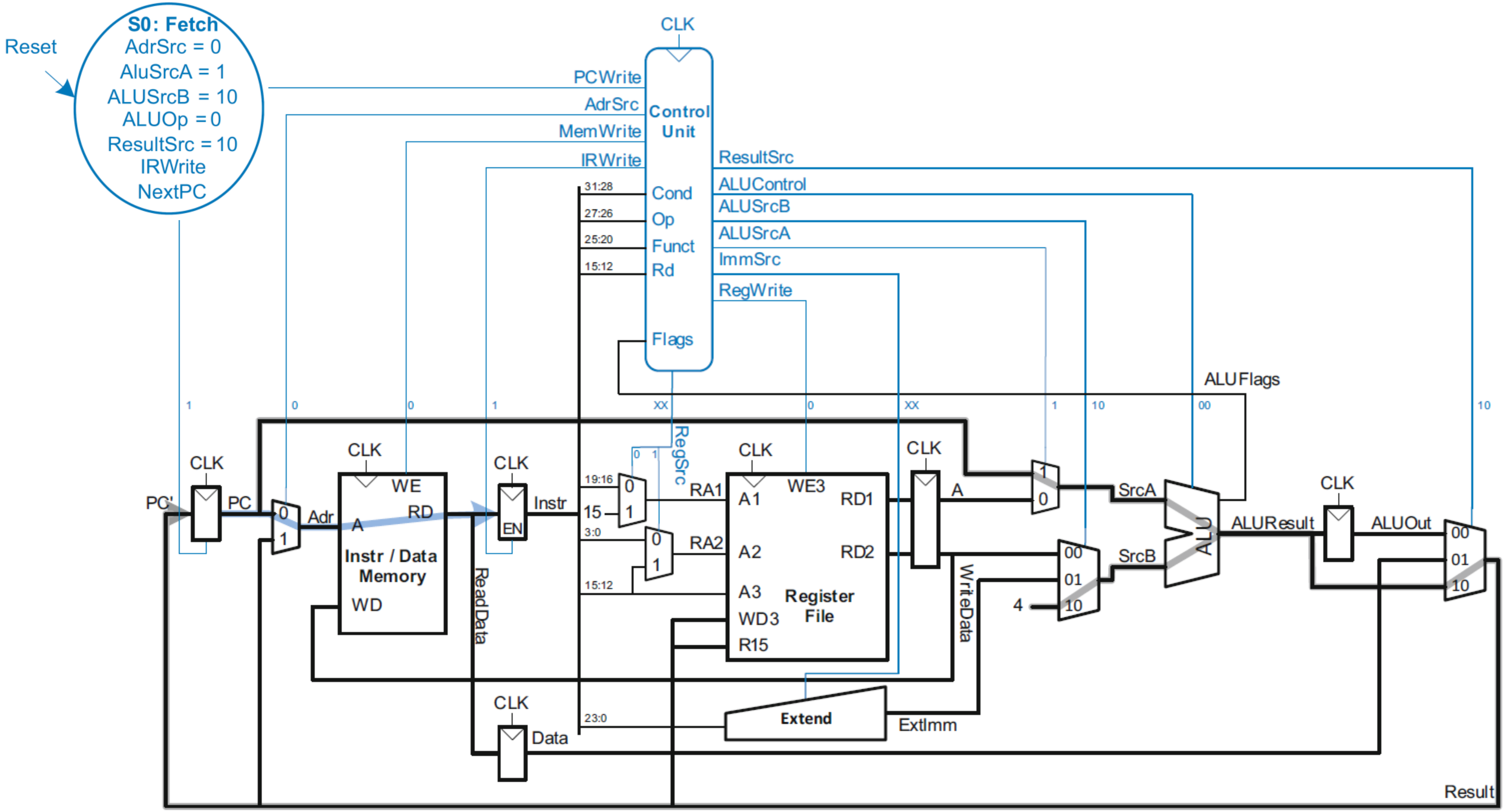
Multicycle ARM Processor



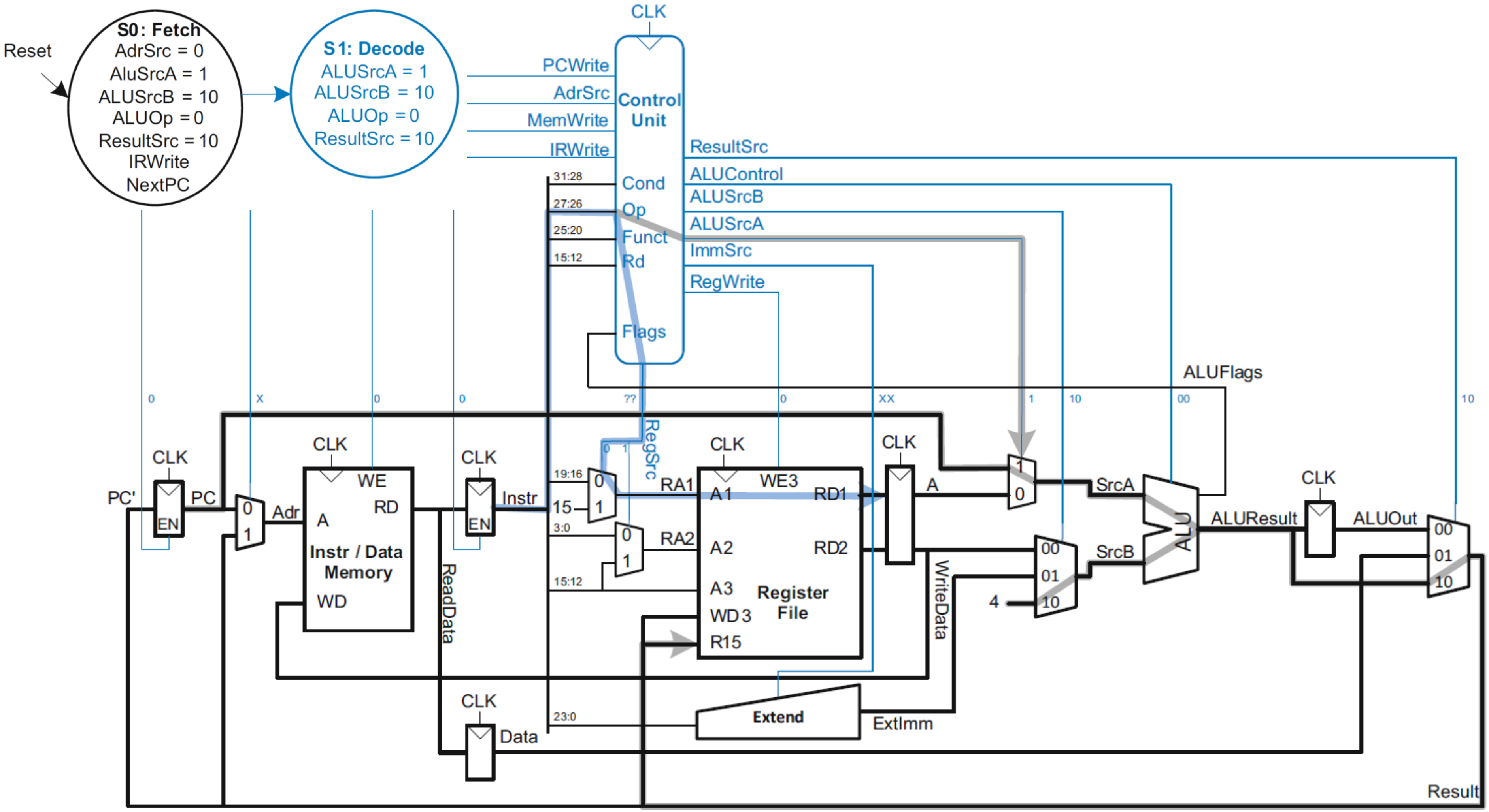
Multicycle Control: Main FSM



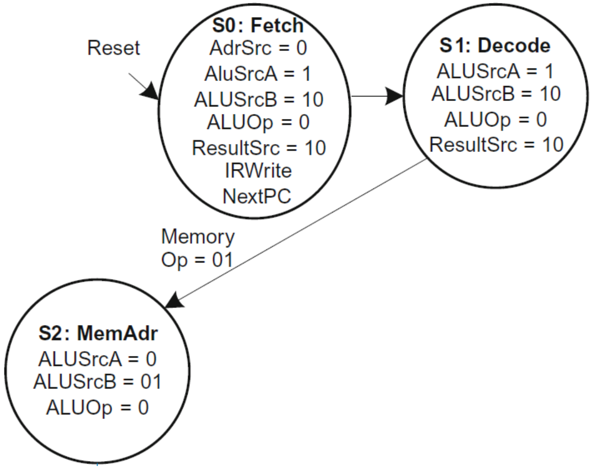
Main Controller FSM: Fetch

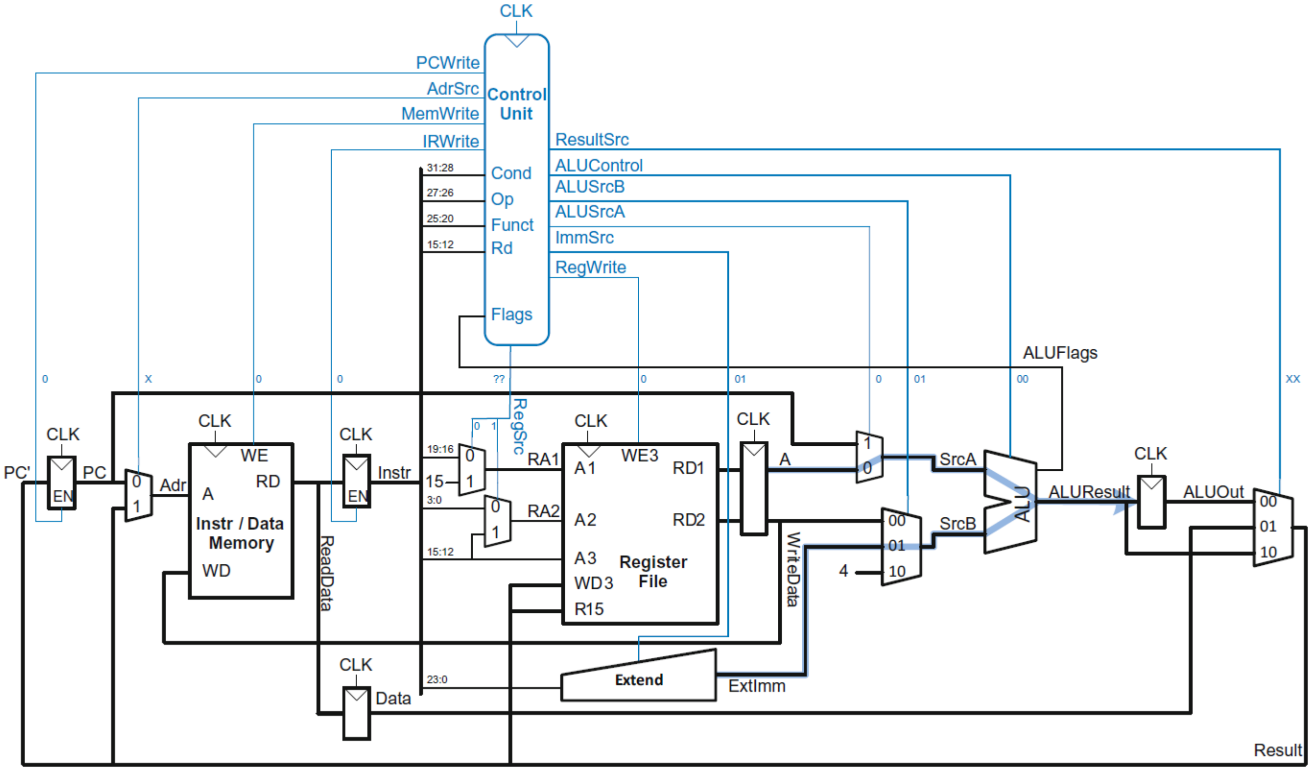


Main Controller FSM: Decode

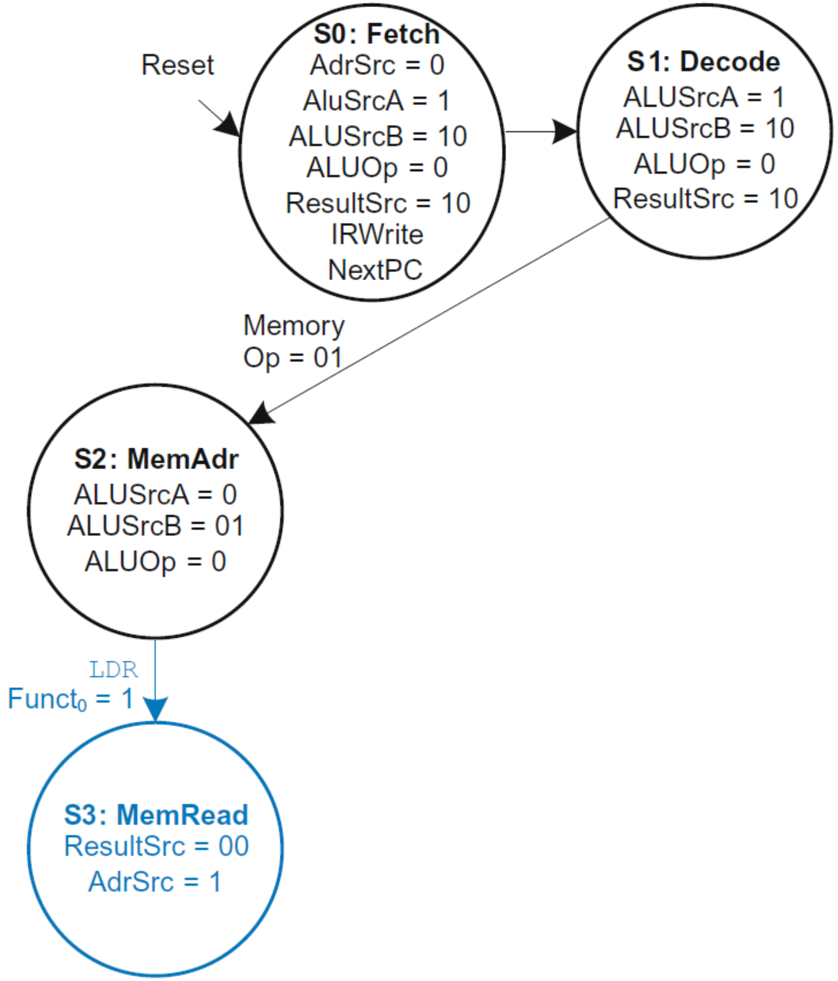


Main Controller FSM: Address

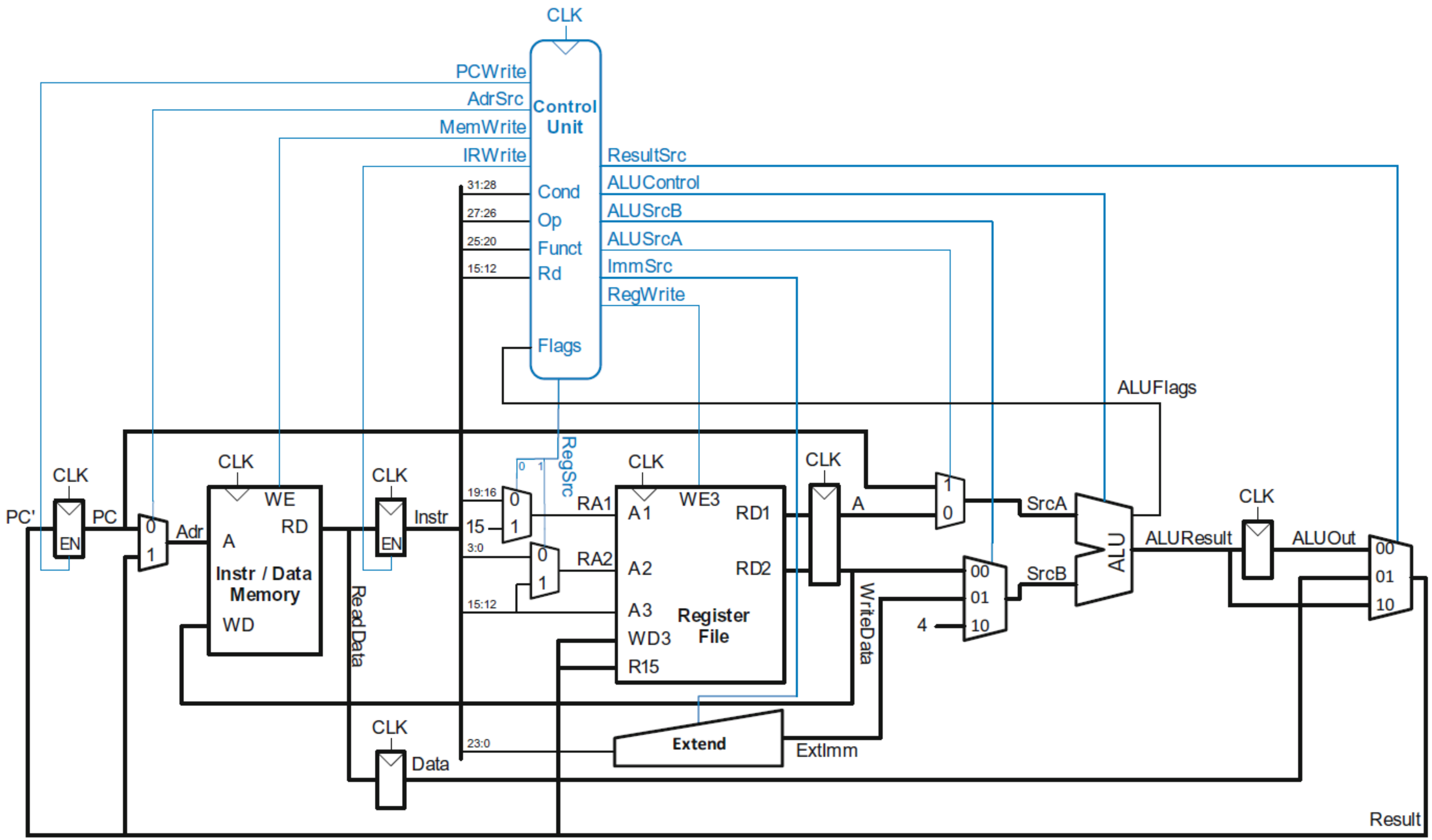




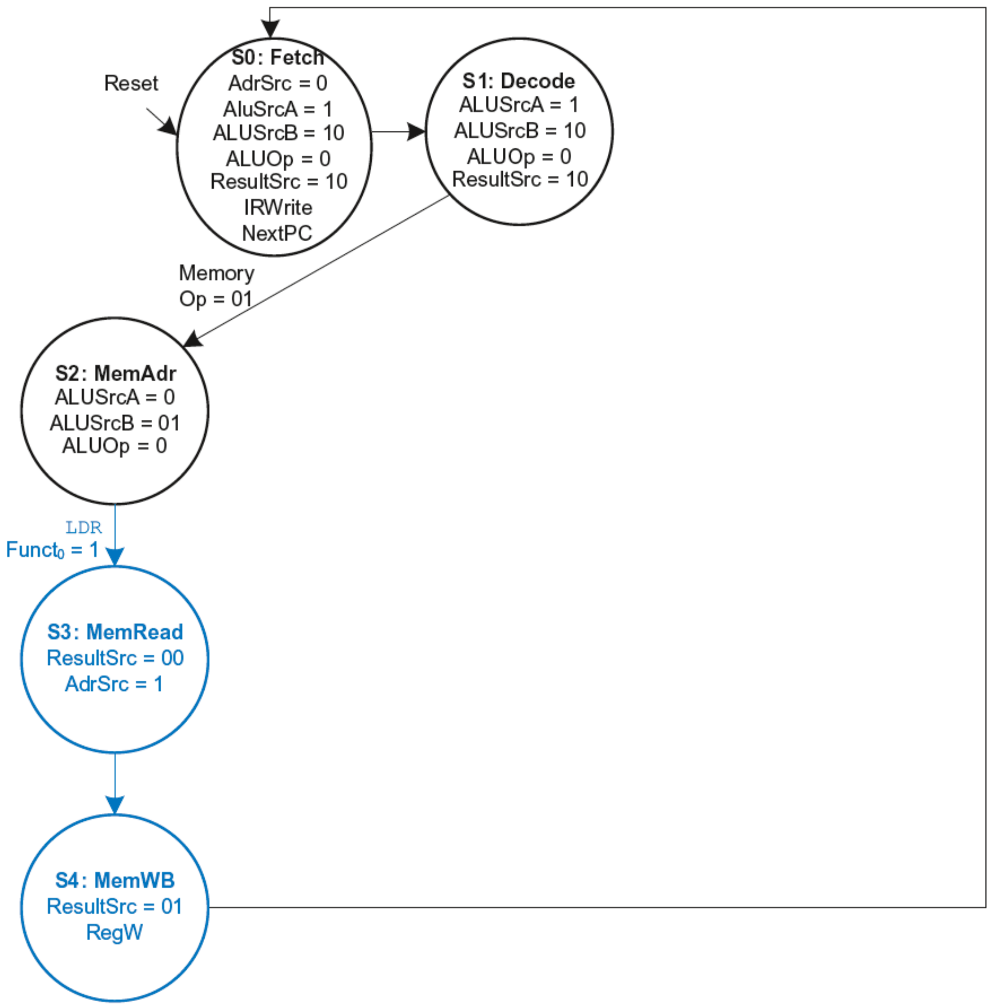
Main Controller FSM: Read Memory



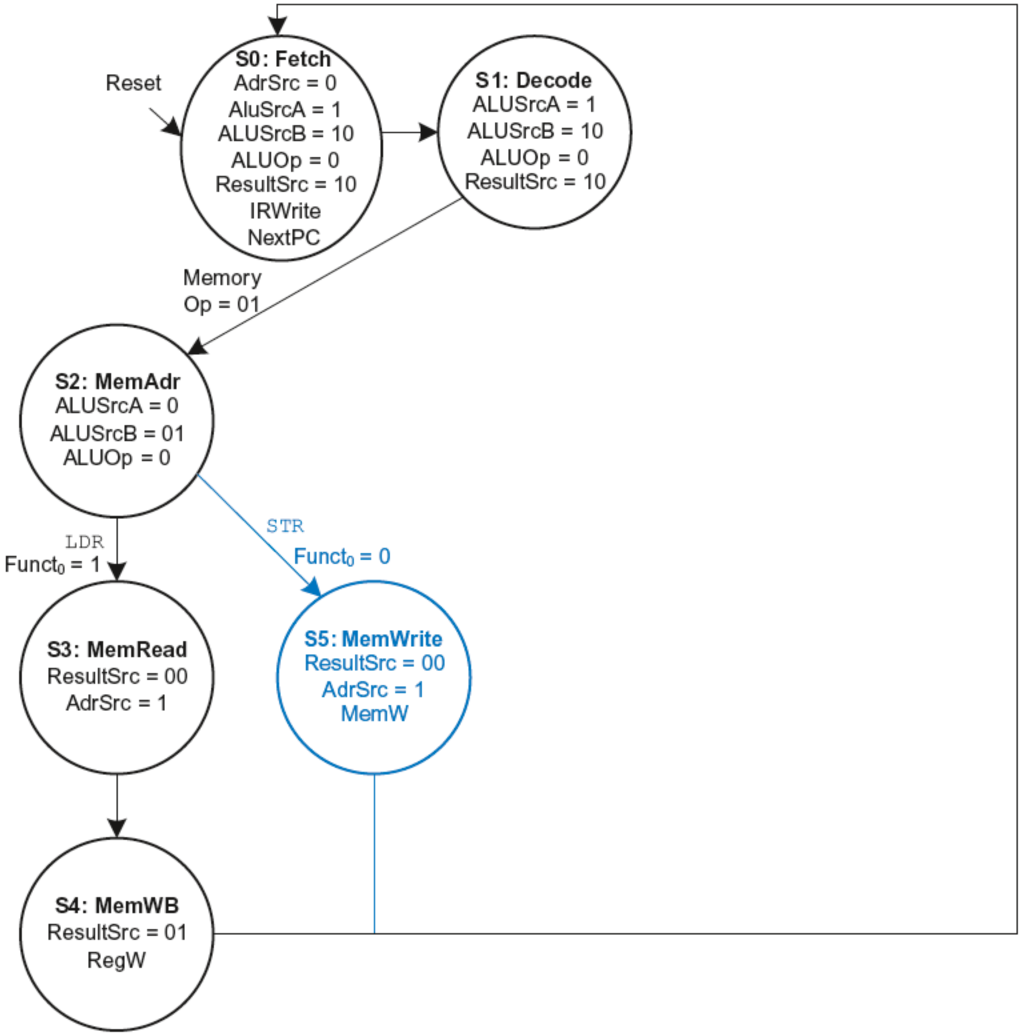
Multicycle ARM Processor



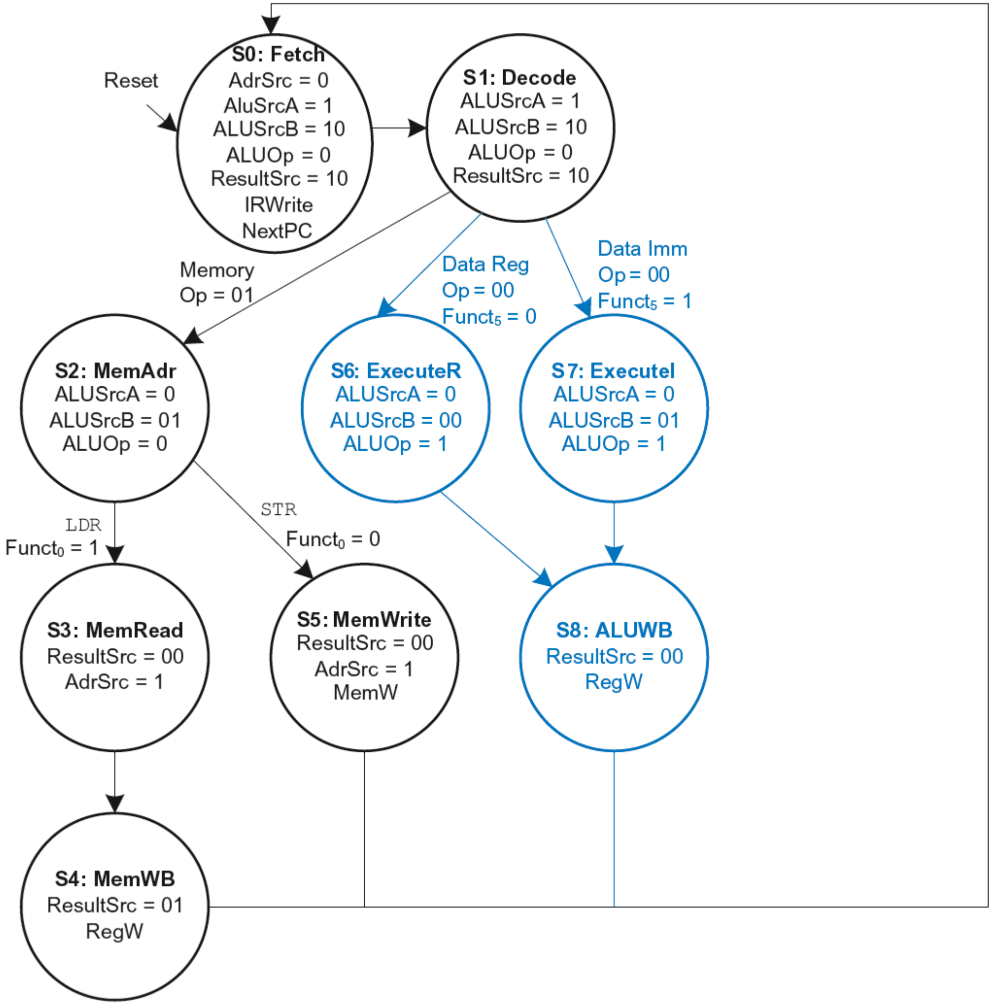
Main Controller FSM: LDR



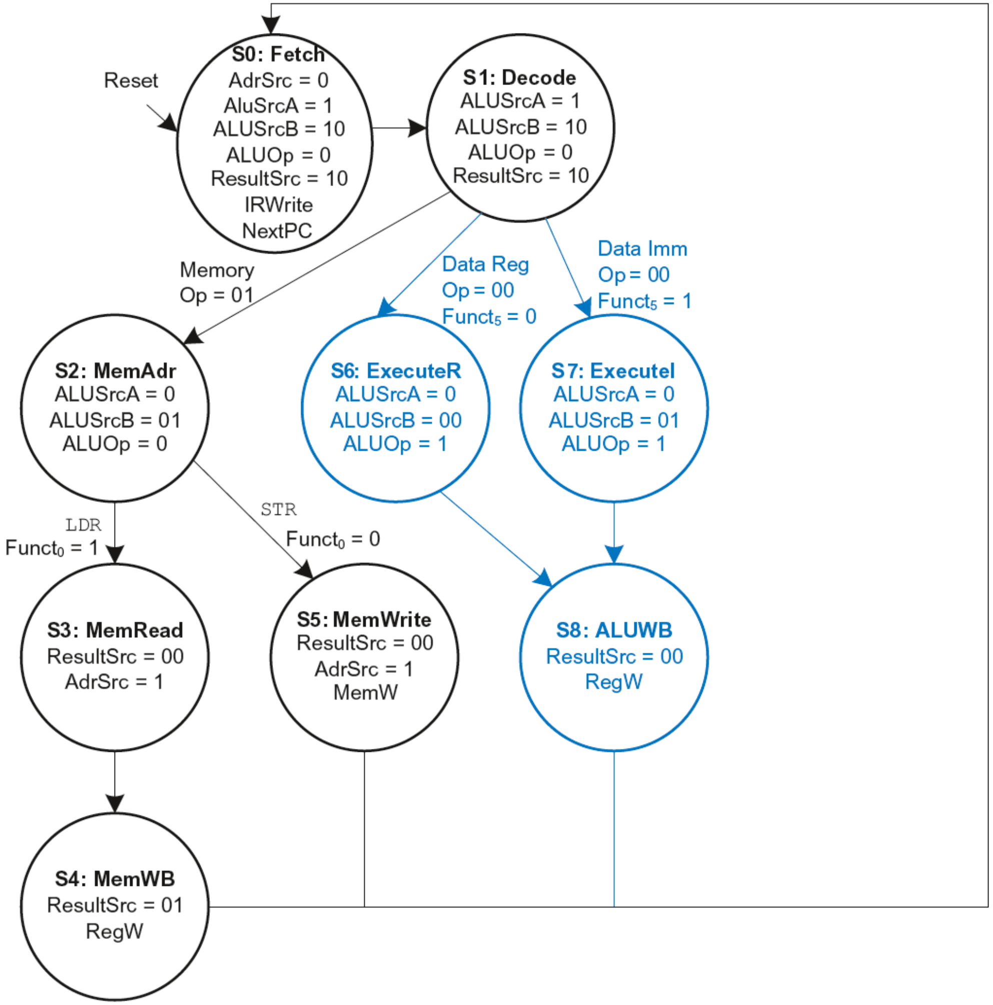
Main Controller FSM: STR



Main Controller FSM: Data-processing

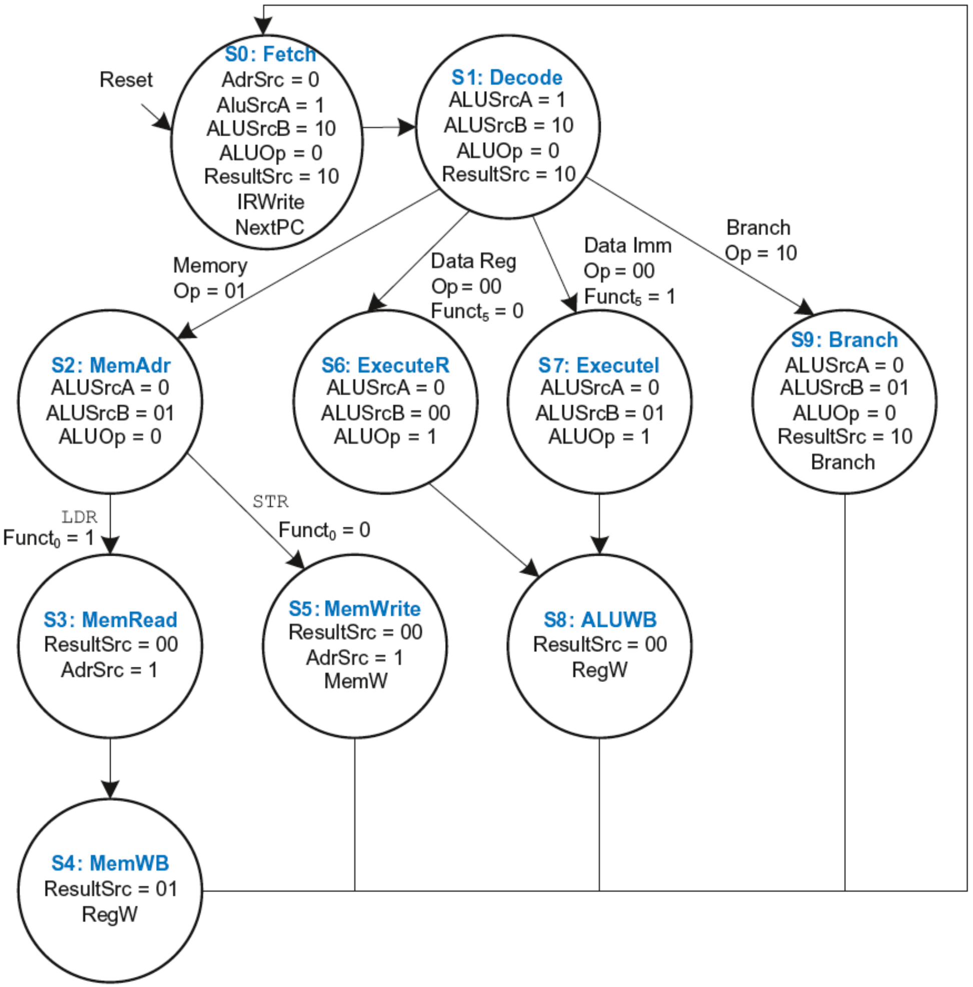


Main Controller FSM: Data-processing

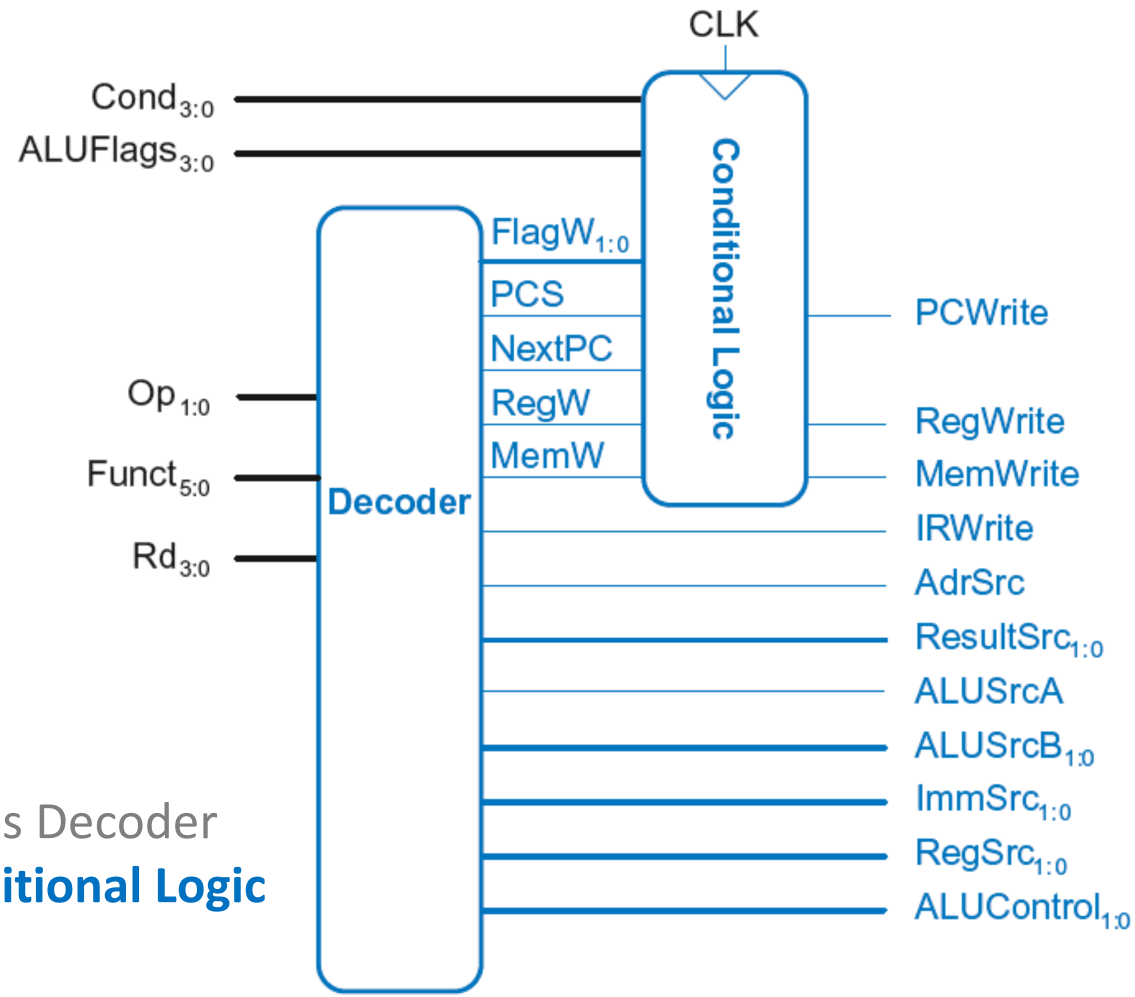


Multicycle Controller FSM

State	Datapath µOp
Fetch	Instr ←Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset

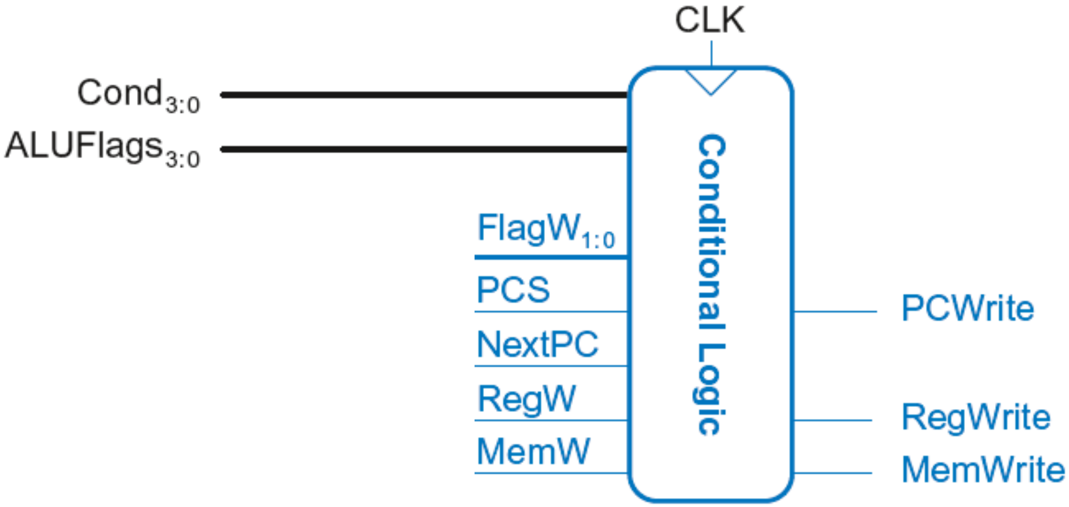


Multicycle Control

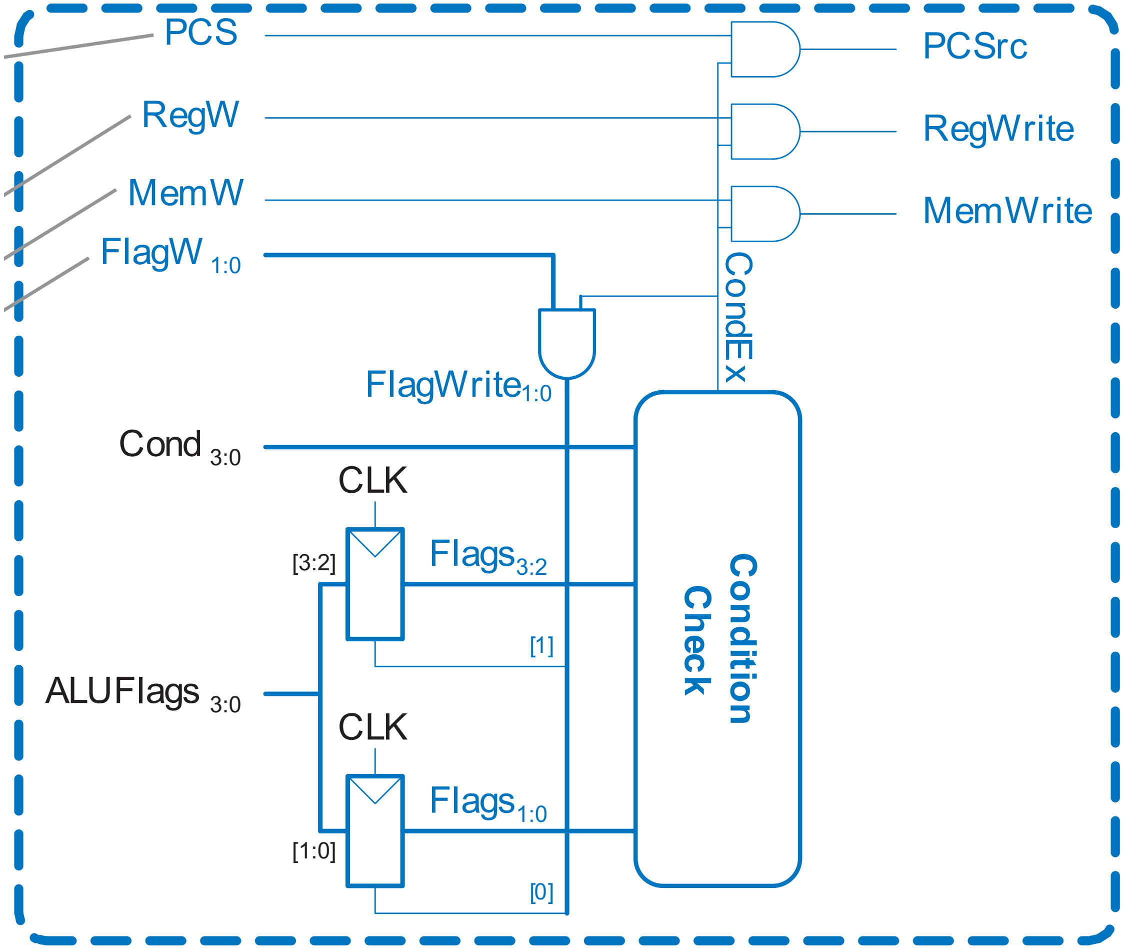


- First, discuss Decoder
- **Then, Conditional Logic**

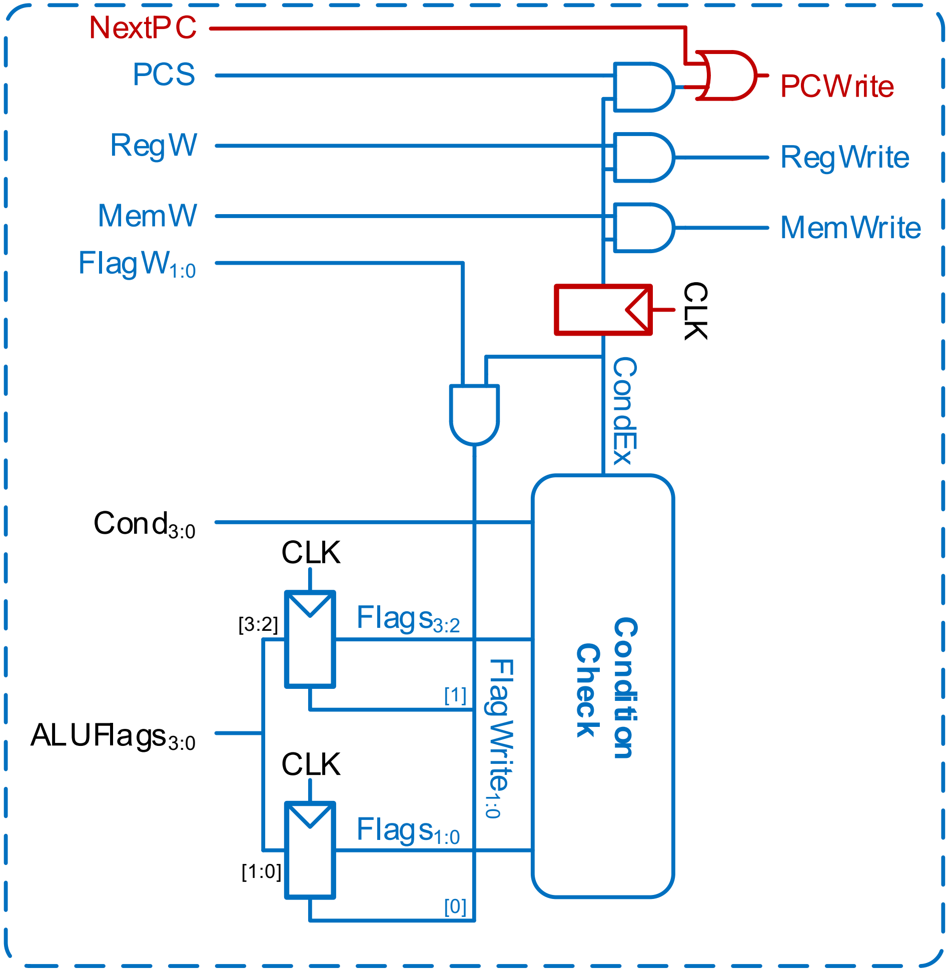
Multicycle Control: Cond. Logic



Single-Cycle Conditional Logic



Multicycle Conditional Logic



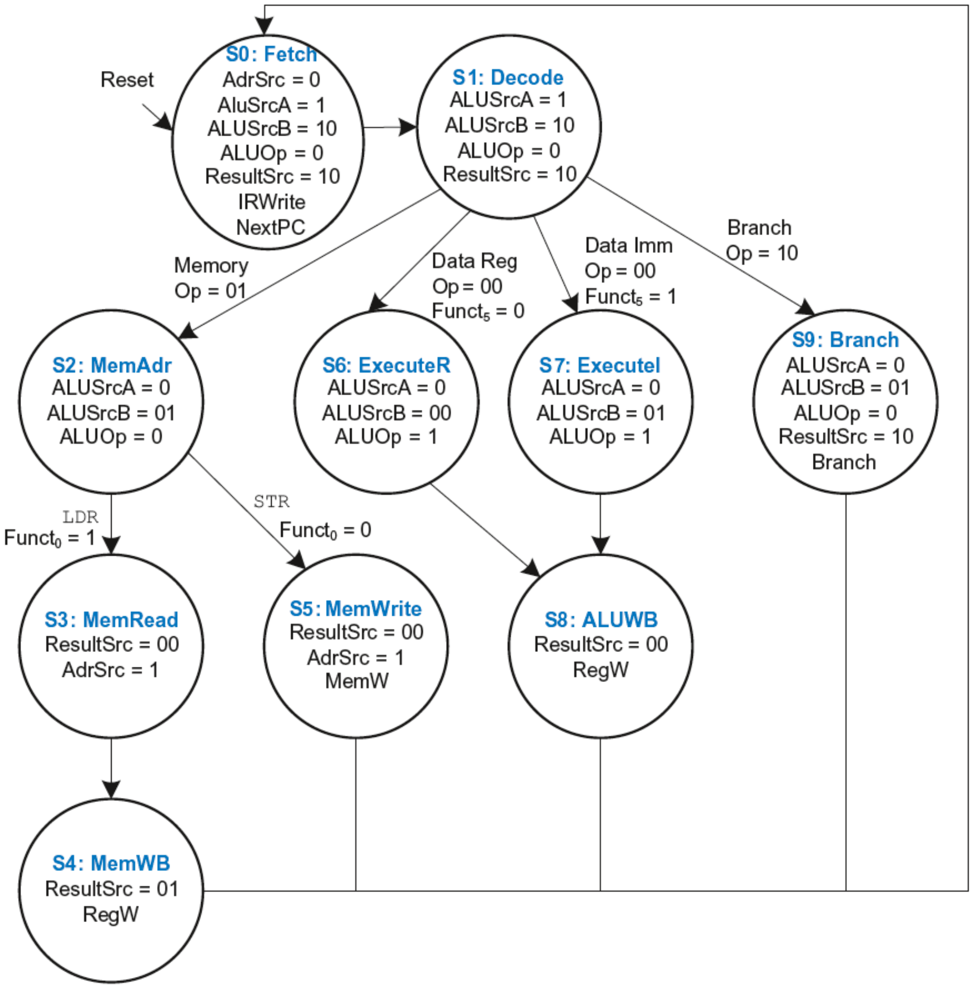
- **PCWrite** asserted in Fetch state
- **ExecuteI/ExecuteR** state:
  CondEx asserts
  ALUFlags generated
- **ALUWB** state:
  Flags updated
  CondEx changes
  PCWrite, ***RegWrite***, and ***MemWrite*** don't see change till new instruction (Fetch state)

Multicycle Processor Performance

- Instructions take different number of cycles.

Multicycle Controller FSM

State	**Datapath μOp**
Fetch	Instr ←Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset



Multicycle Processor Performance

- Instructions take different number of cycles:
  - 3 cycles:
  - 4 cycles:
  - 5 cycles:

Multicycle Processor Performance

- Instructions take different number of cycles:
  - 3 cycles: `B`
  - 4 cycles: DP, `STR`
  - 5 cycles: `LDR`

Multicycle Processor Performance

- Instructions take different number of cycles:
  - 3 cycles: `B`
  - 4 cycles: DP, `STR`
  - 5 cycles: `LDR`
- CPI is weighted average
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% data processing

Multicycle Processor Performance

- Instructions take different number of cycles:
  - 3 cycles: `B`
  - 4 cycles: DP, `STR`
  - 5 cycles: `LDR`
- CPI is weighted average
- SPECINT2000 benchmark:
  - **25%** loads
  - **10%** stores
  - **13%** branches
  - **52%** data processing

$$\textbf{Average CPI} = (\mathbf{0.13})(3) + (\mathbf{0.52 + 0.10})(4) + (\mathbf{0.25})(5) = \mathbf{4.12}$$

Multicycle Processor Performance

Multicycle critical path:

- Assumptions:
  - RF is faster than memory
  - writing memory is faster than reading memory

$$T_{c2} = t_{pcq} + 2t_{\text{mux}} + \max(t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}) + t_{\text{setup}}$$

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

$T_{c2} = ?$

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

$$\begin{aligned} \boldsymbol{T_{c2}} &= t_{pcq} + 2t_{\mathrm{mux}} + \max[t_{\mathrm{ALU}} + t_{\mathrm{mux}},\ t_{\mathrm{mem}}] + t_{\mathrm{setup}} \\ &= [40 + 2(25) + 200 + 50]\ \mathrm{ps} = \mathbf{340\ ps} \end{aligned}$$

Multicycle Performance Example

For a program with **100 billion** instructions executing on a **multicycle** ARM processor

- **CPI** = 4.12 cycles/instruction
- **Clock cycle time:** T_{c2} = 340 ps

Execution Time = ?

Multicycle Performance Example

For a program with **100 billion** instructions executing on a **multicycle** ARM processor

- **CPI** = 4.12 cycles/instruction
- **Clock cycle time:** T_{c2} = 340 ps

Execution Time = (# instructions) × CPI × T_c

$= (100 \times 10^{9})(4.12)(340 \times 10^{-12})$

= **140 seconds**

Multicycle Performance Example

For a program with **100 billion** instructions executing on a **multicycle** ARM processor

- **CPI** = 4.12 cycles/instruction
- **Clock cycle time:** T_{c2} = 340 ps

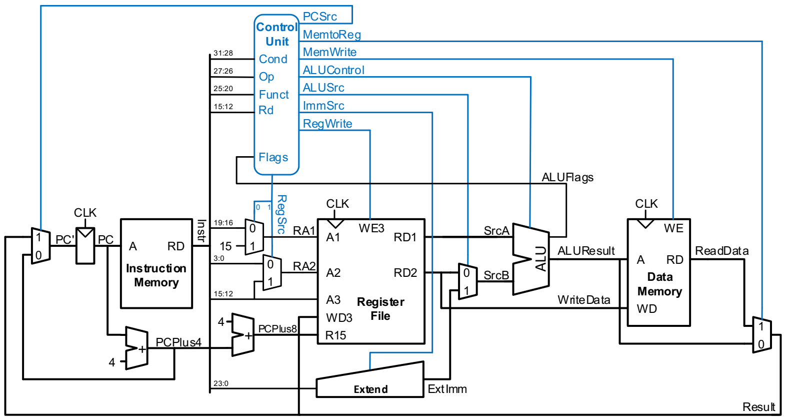
Execution Time = (# instructions) × CPI × T_c

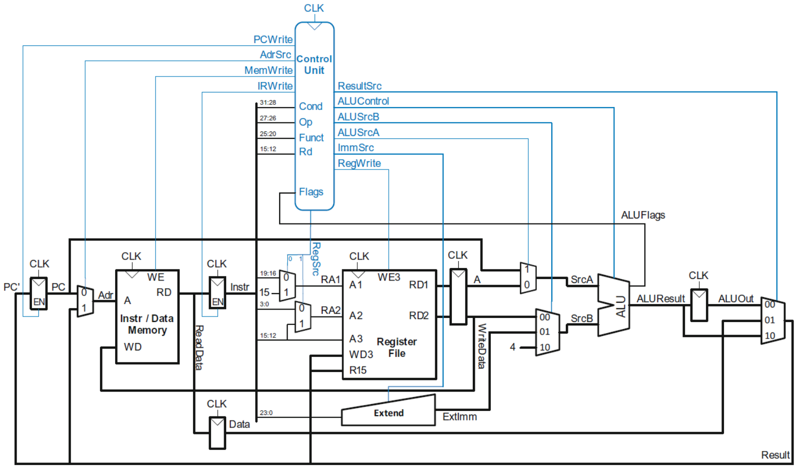
$= (100 \times 10^{9})(4.12)(340 \times 10^{-12})$

= **140 seconds**

This is slower than the single-cycle processor (84 sec.)

Processor Comparisons





Single Cycle

One cycle/instruction

Long clock period

Separate I and D Mem

Combinational controller

Architectural State Only

Multicycle

3-5 cycles/instruction

Shorter clock period

Unified Memory

FSM controller

Extra state