THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

CPE 221: Computer Organization

15 Multicycle ARM Processor

rahul.bhadani@uah.edu

Rahul Bhadani

Multicycle ARM Processor

- **Single-cycle:**

+ simple
- cycle time limited by longest instruction (LDR)
- separate memories for instruction and data
- 3 adders/ALUs – ADDERS are expensive circuits

- **Multicycle processor addresses these issues by breaking instruction into shorter steps**

a. shorter instructions take fewer steps
b. can re-use hardware
c. cycle time is faster
d. read/write the memory/register or use the ALU
e. instruction read in one step, data written in a later step – thus need only single memory for instruction and data.

Multicycle ARM Processor

- **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:

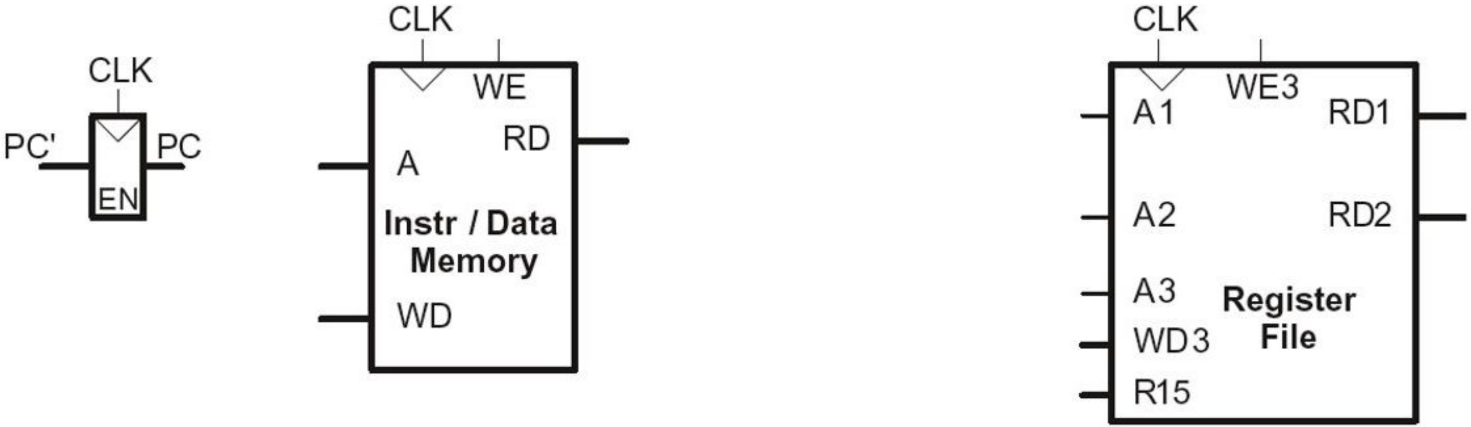
- first datapath
- then control

In addition:

- circuitry to store the intermediate results

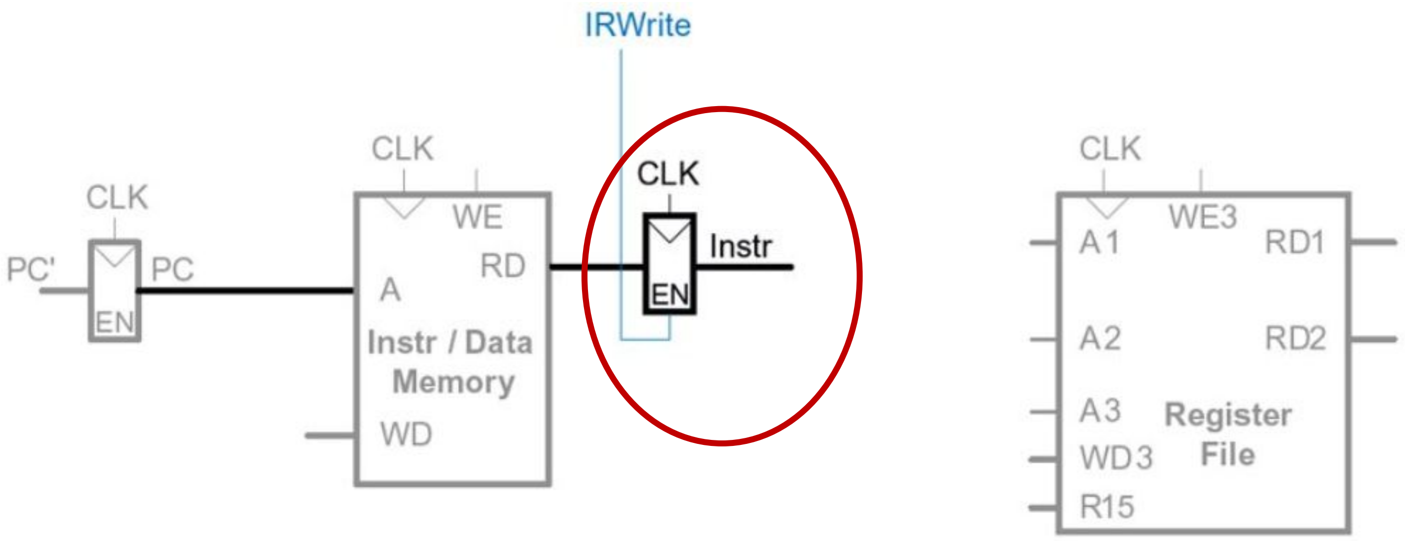
Multicycle State Elements

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



Instruction Register

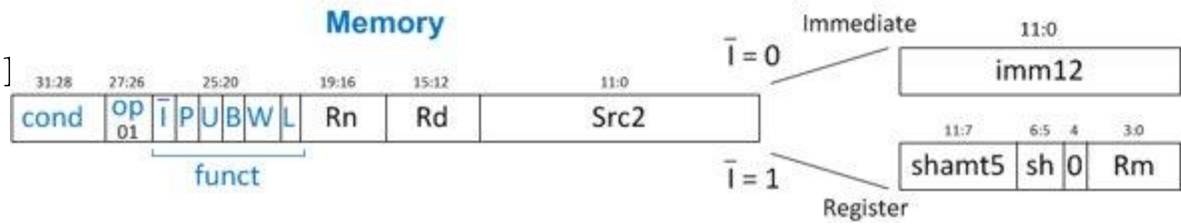
The instruction is read and stored in a new non-architectural instruction register (IR) so that it is available for future cycles. The IR receives an enable signal, called IRWrite, which is asserted when the IR should be loaded with a new instruction.

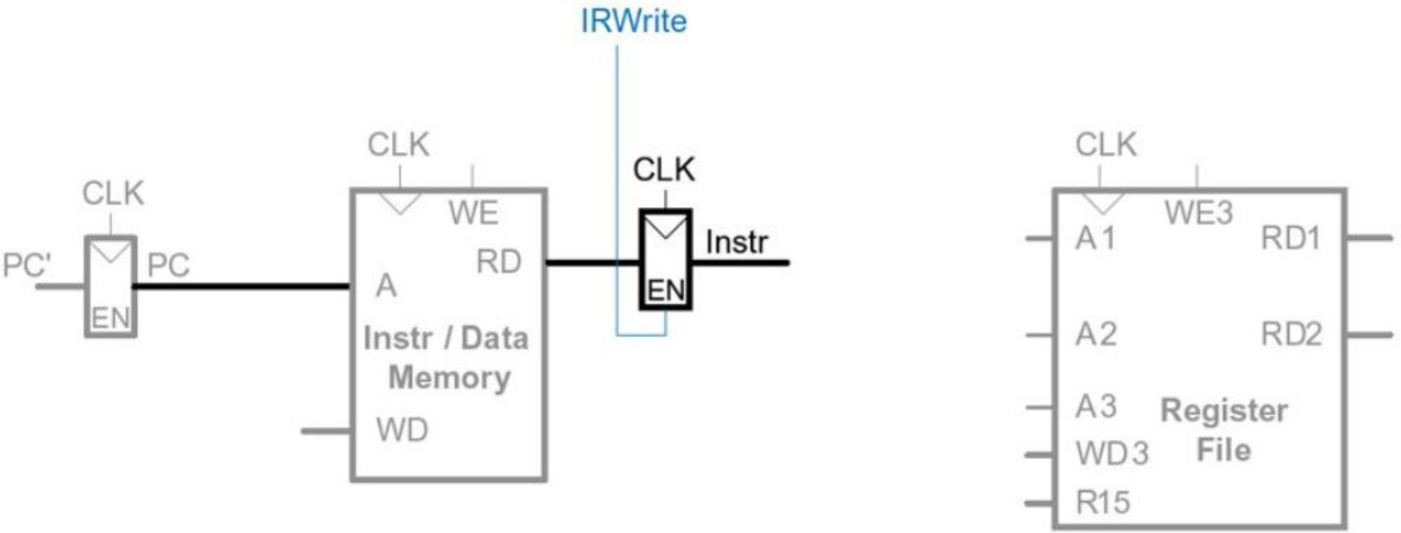


Multicycle Datapath: Instruction Fetch

STEP 1: Fetch instructio

LDR Rd, [Rn, imm12]

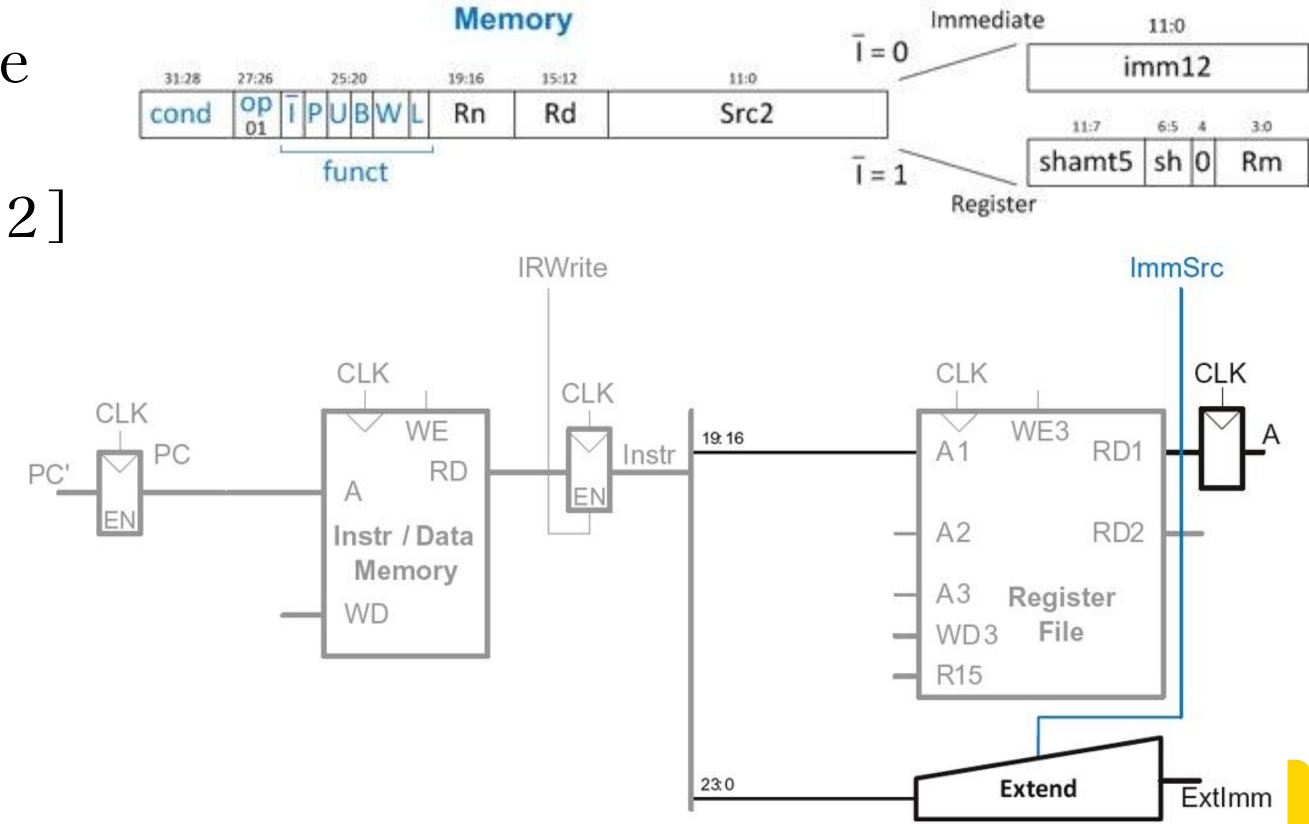




Multicycle Datapath: LDR Register Read

STEP 2: Read source operands from RF

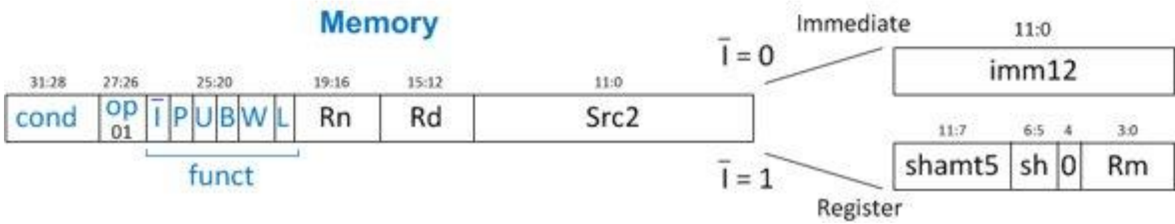
LDR Rd, [Rn, imm12]

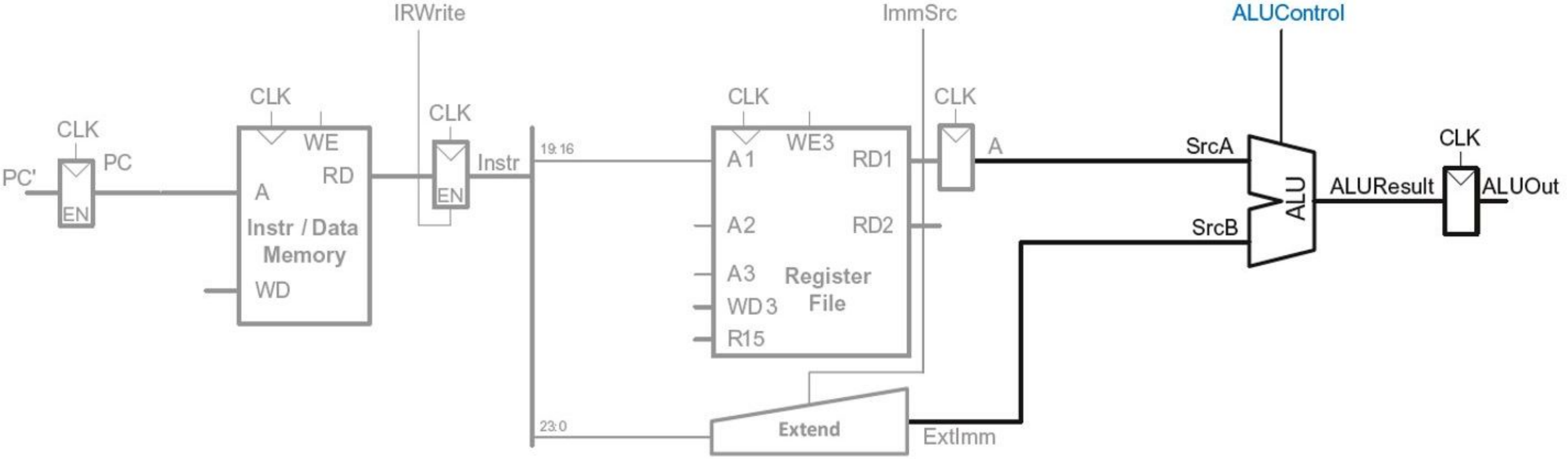


Multicycle Datapath: LDR Address

STEP 3: Compute the memory address

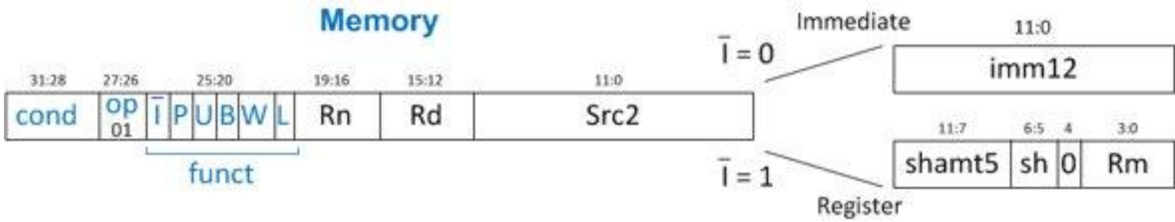
LDR Rd, [Rn, imm12]

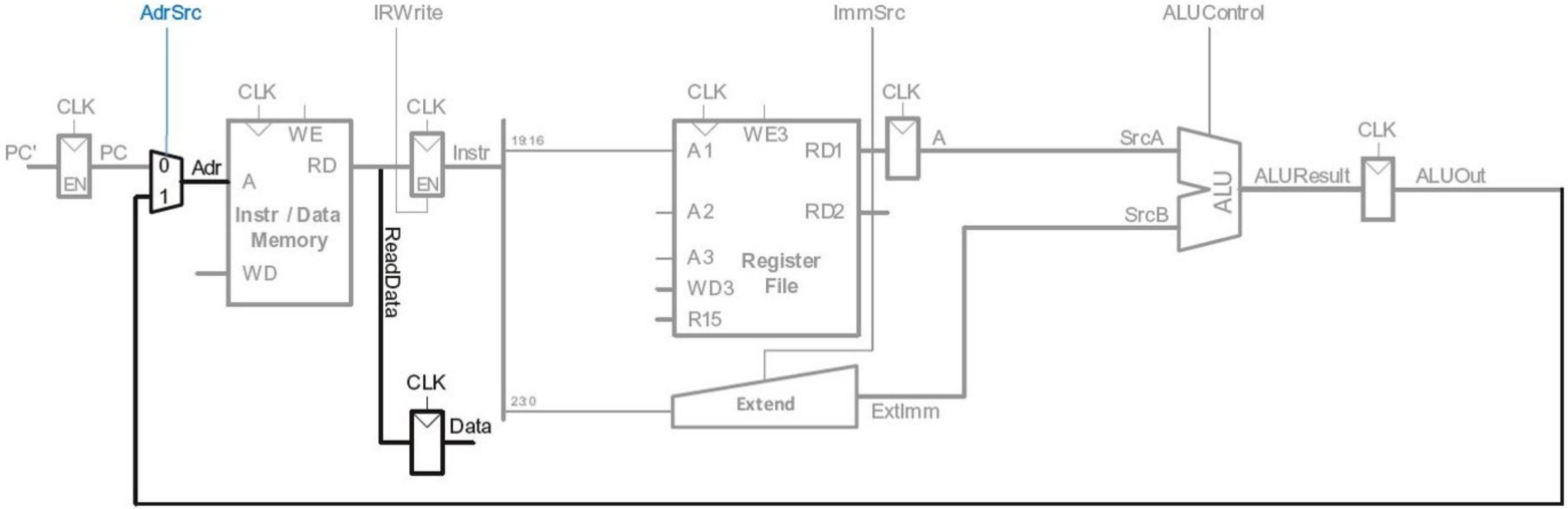




Multicycle Datapath: LDR Memory Read

STEP 4: Read data from memory
LDR Rd, [Rn, imm12]

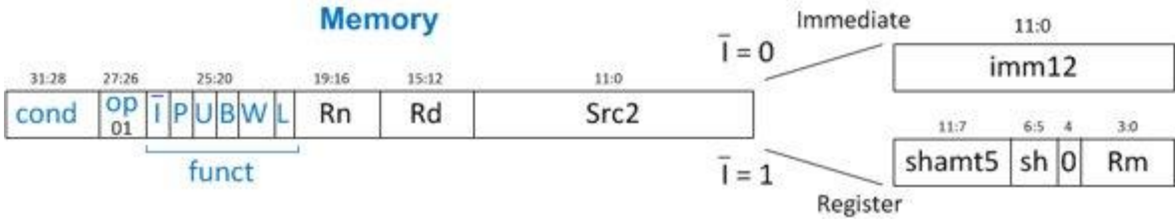


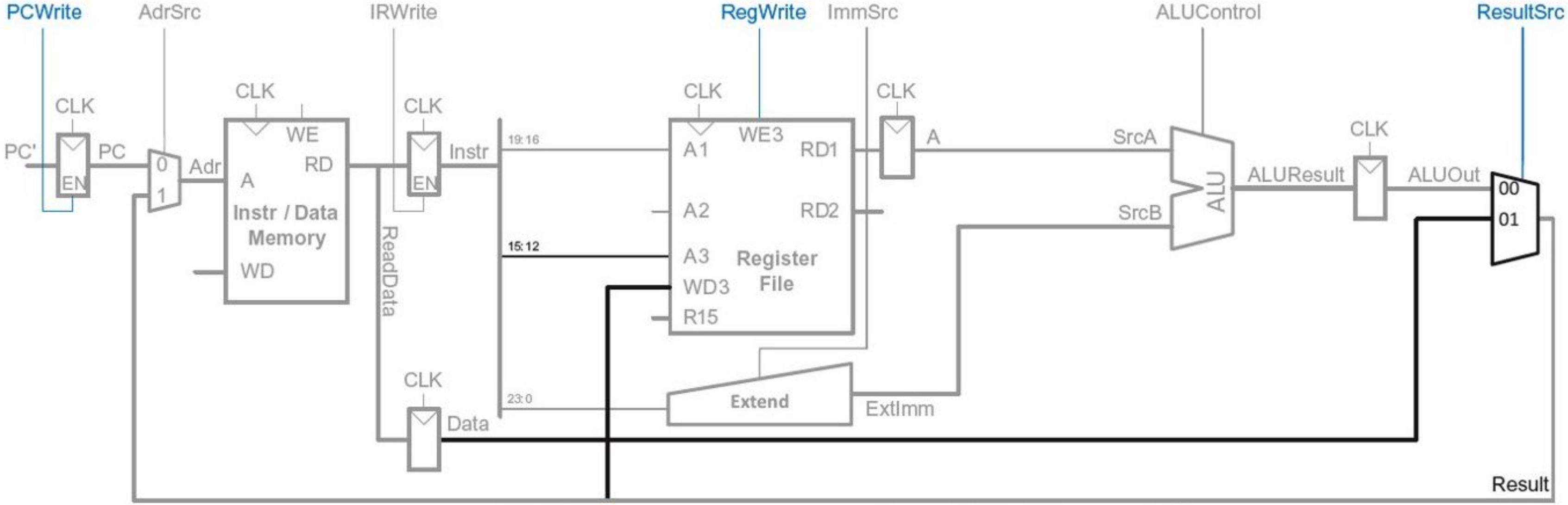


Multicycle Datapath: LDR Write Register

STEP 5: Write data back to register file

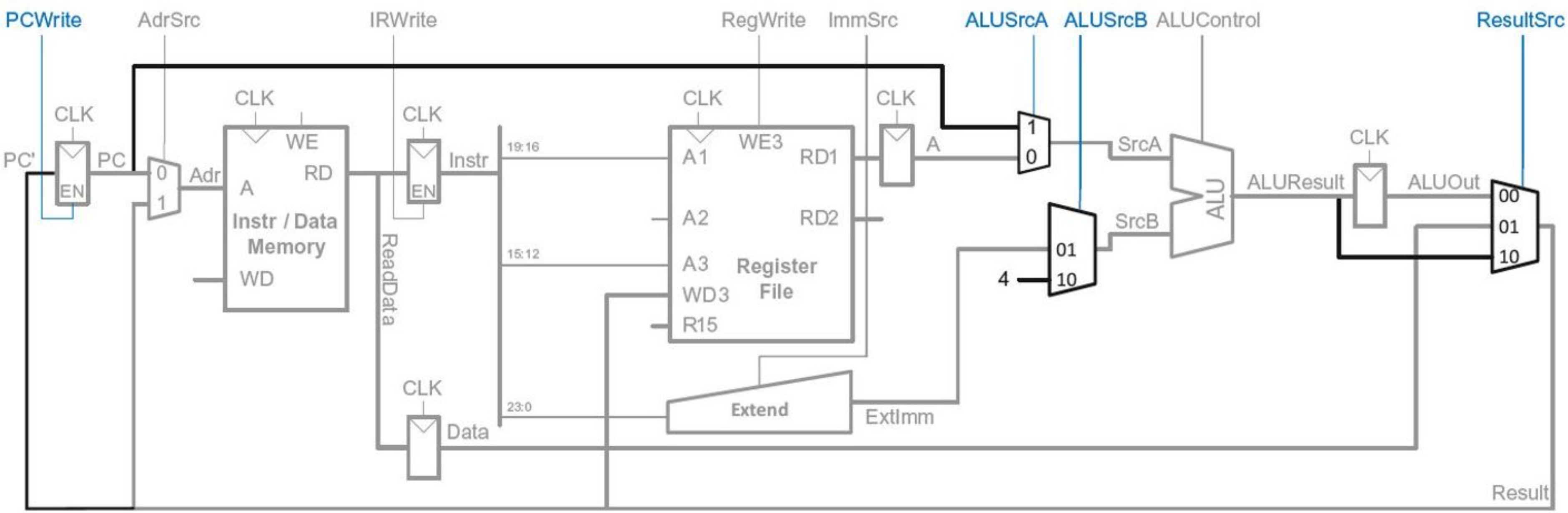
LDR Rd, [Rn, imm12]





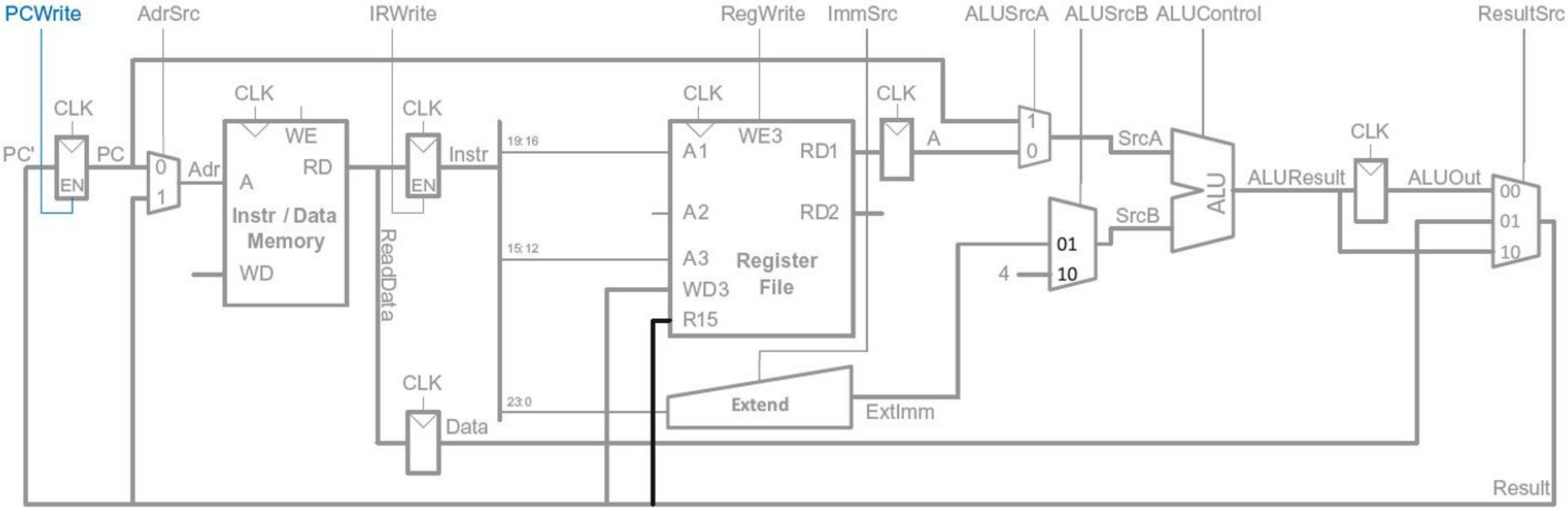
Multicycle Datapath: Increment PC

STEP 6: Increment PC



Multicycle Datapath: Access to PC

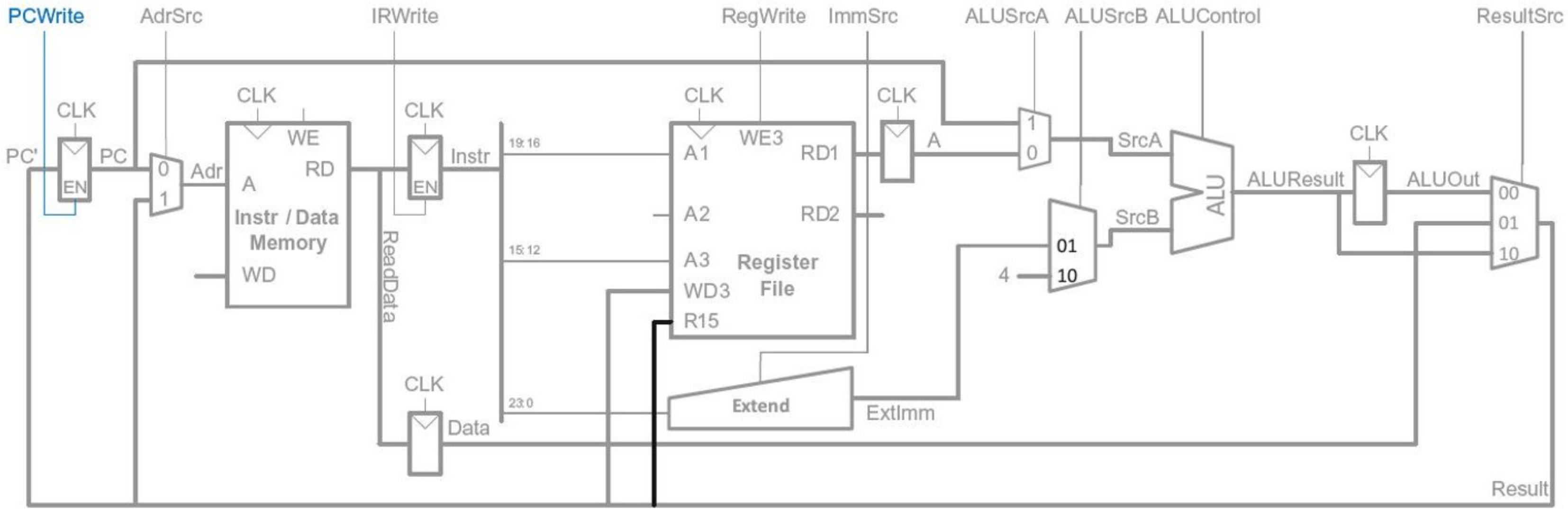
PC can be read/written by instruction



Multicycle Datapath: Access to PC

PC can be read/written by instruction

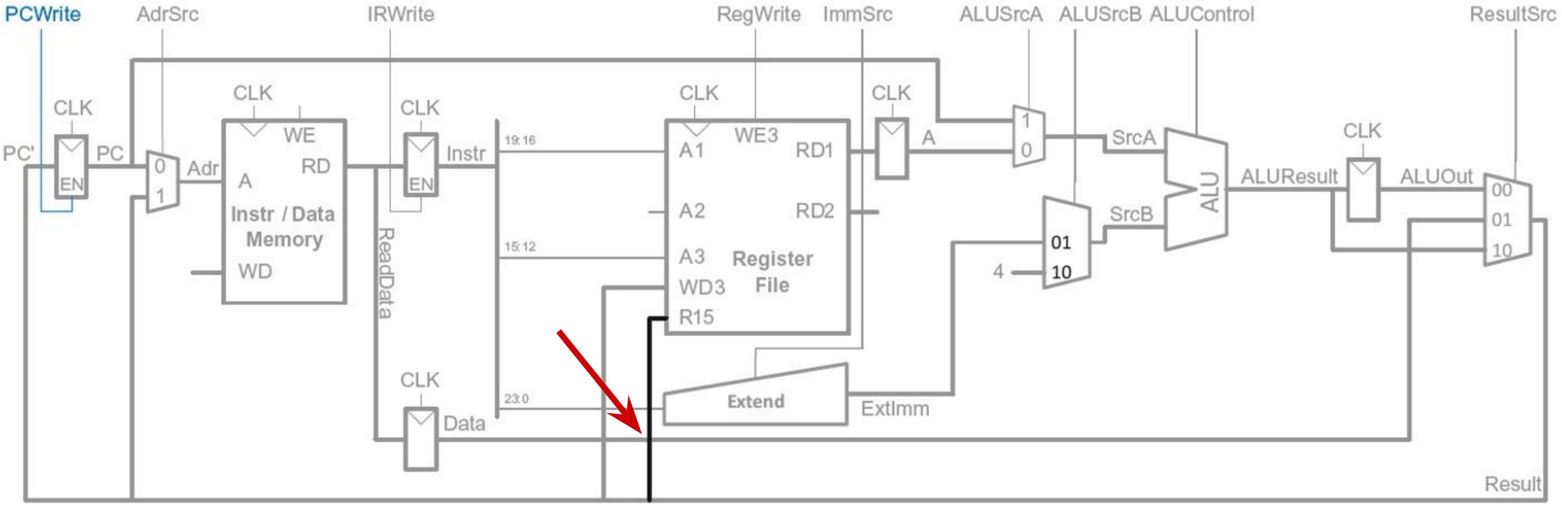
- **Read:** R15 (PC+8) available in Register File



Multicycle Datapath: Read to PC (R15)

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

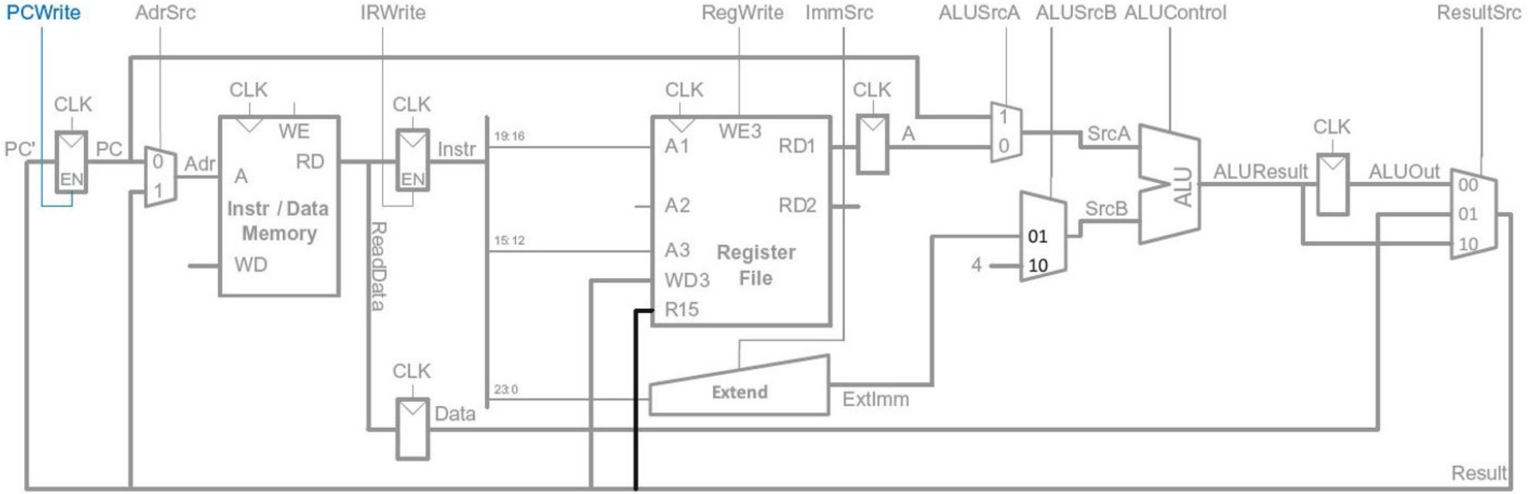
1. R15 needs to be read as PC+8 from Register File (RF) in 2nd step
2. So (also in 2nd step) PC + 8 is produced by ALU and routed to R15 input of RF



Multicycle Datapath: Access to PC

PC can be read/written by instruction

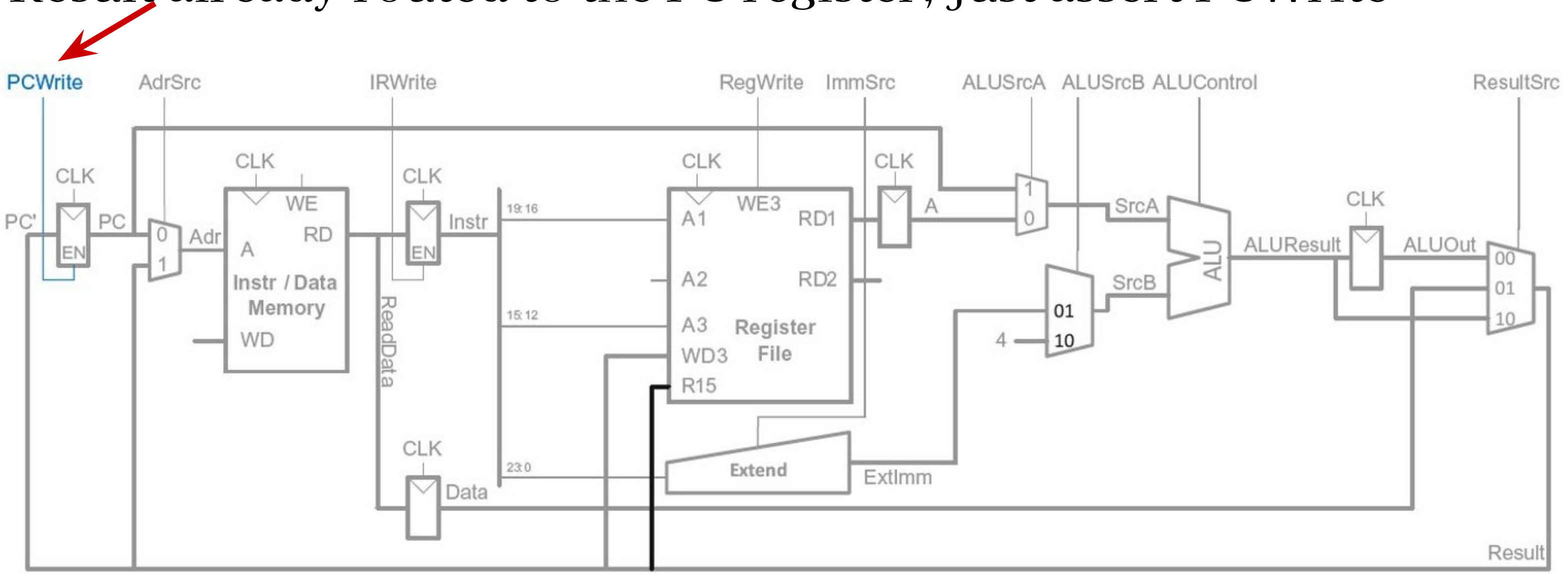
- **Read:** R15 (PC+8) available in Register File
- **Write:** Be able to write result of instruction to PC



Multicycle Datapath: Write to PC (R15)

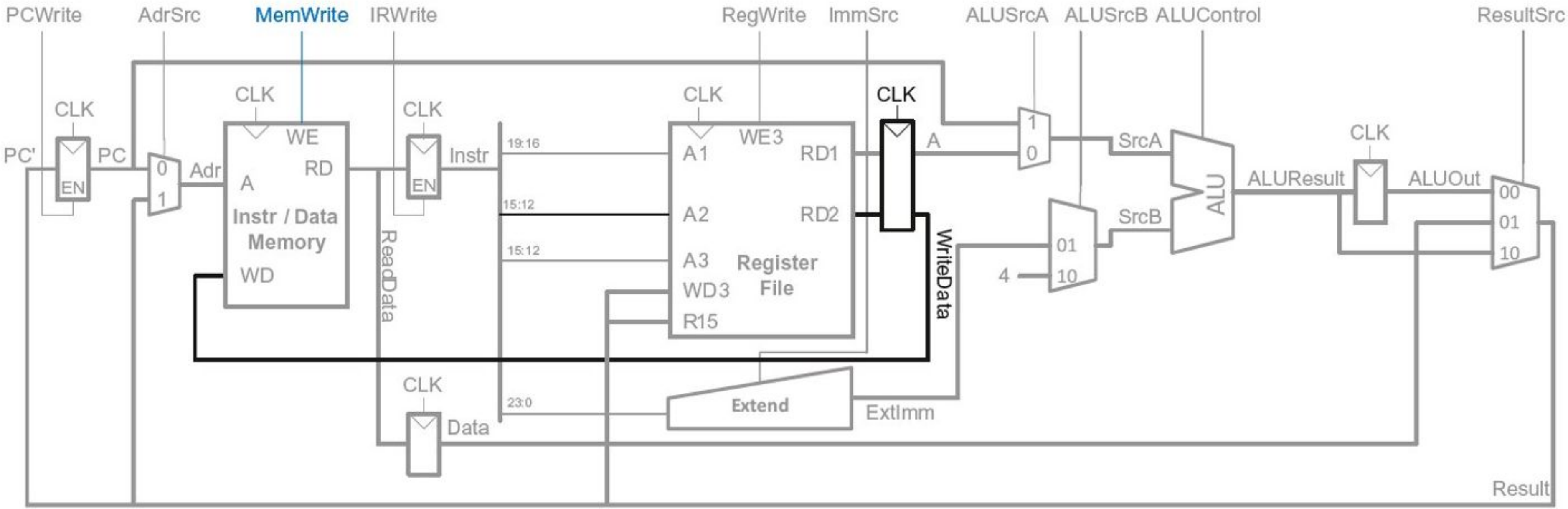
Example: SUB **R15**, R8, R3

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



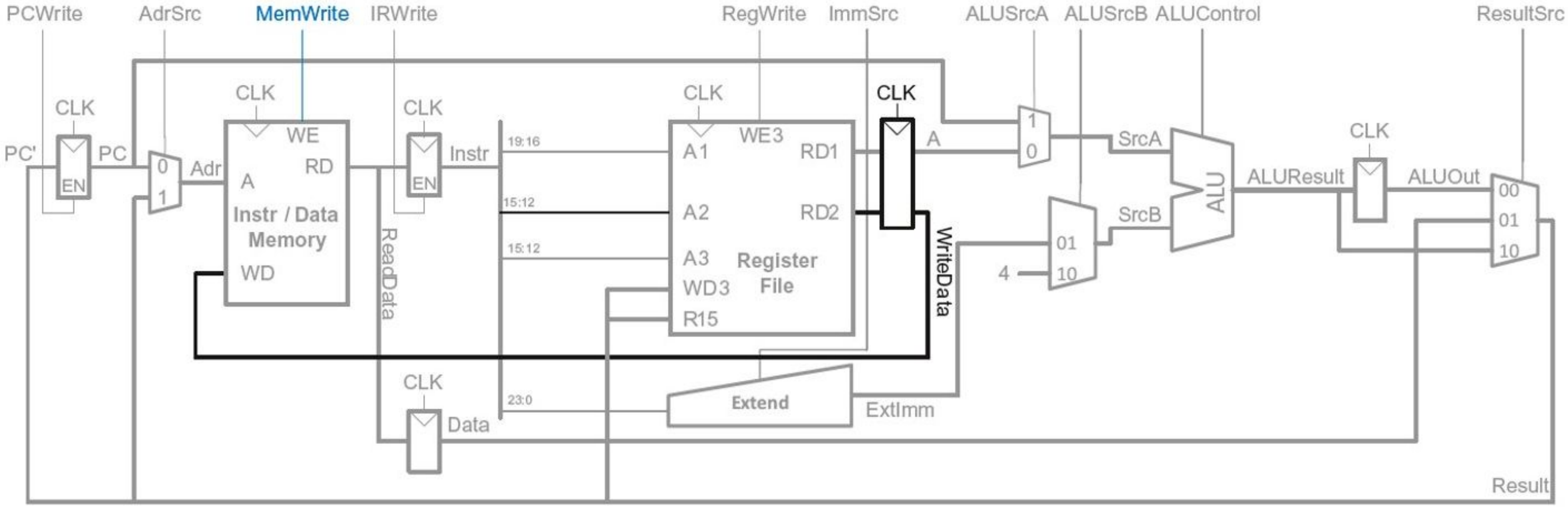
Multicycle Datapath: STR

Write data in Rn to memory



Multicycle Datapath: Data-processing

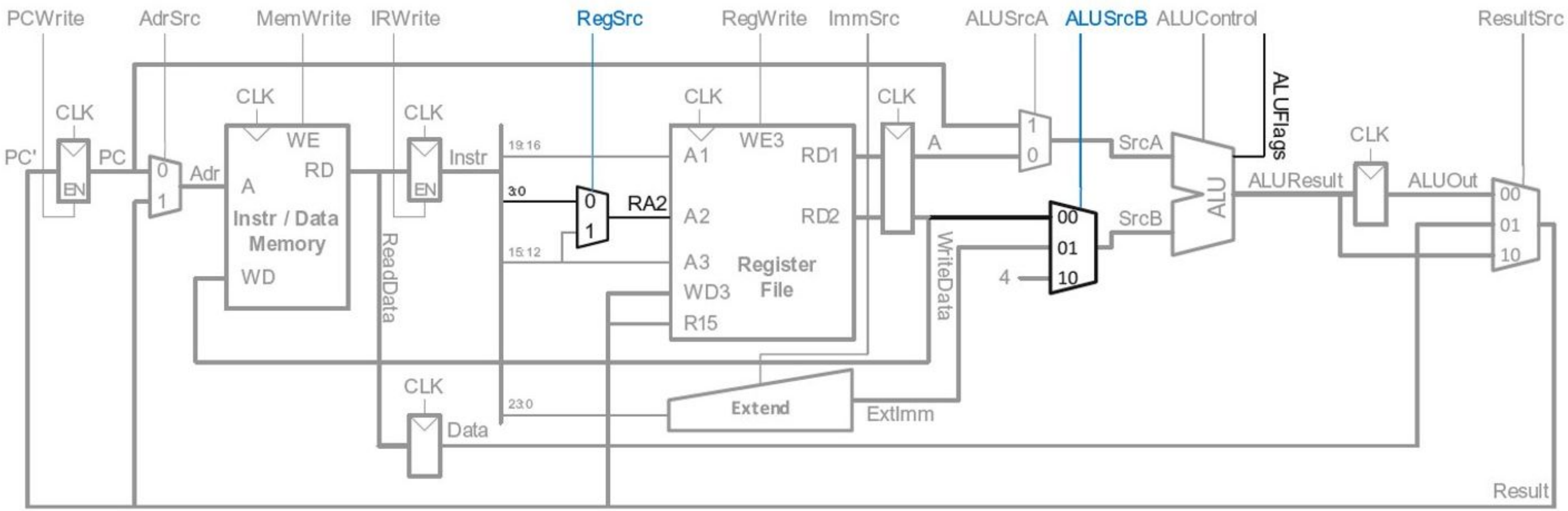
With immediate addressing (i.e., an immediate *Src2*), no additional changes needed for datapath



Multicycle Datapath: Data-processing

With register addressing (register *Src2*):

Read from Rn and Rm

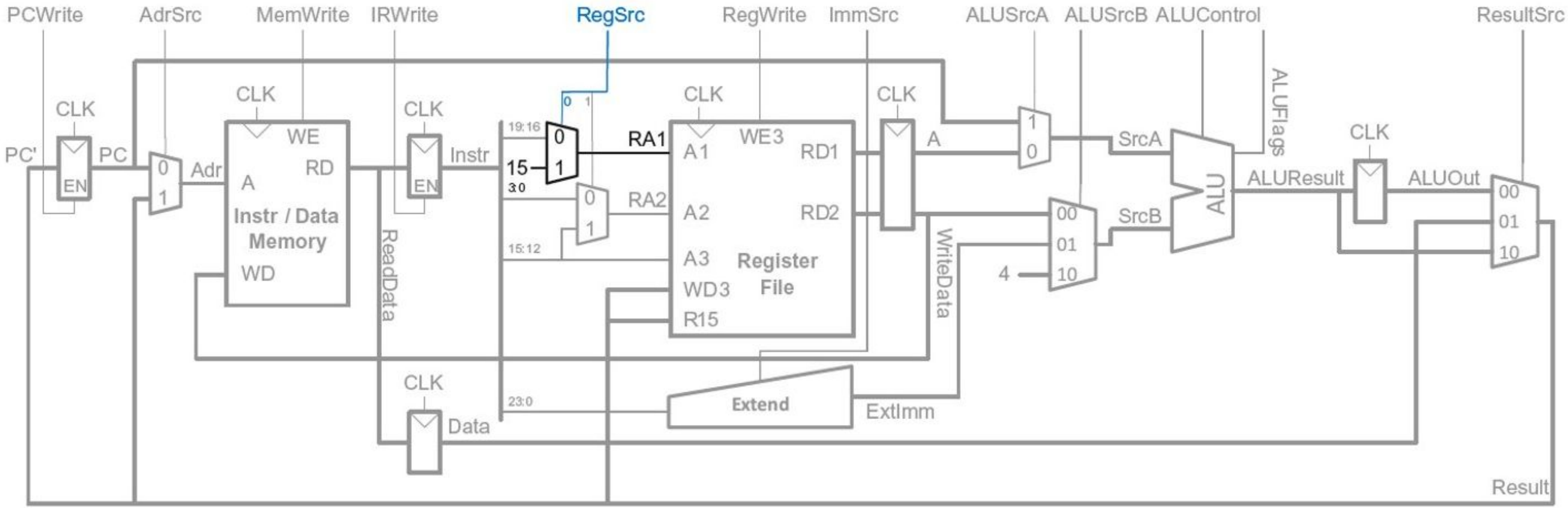


Multicycle Datapath: B

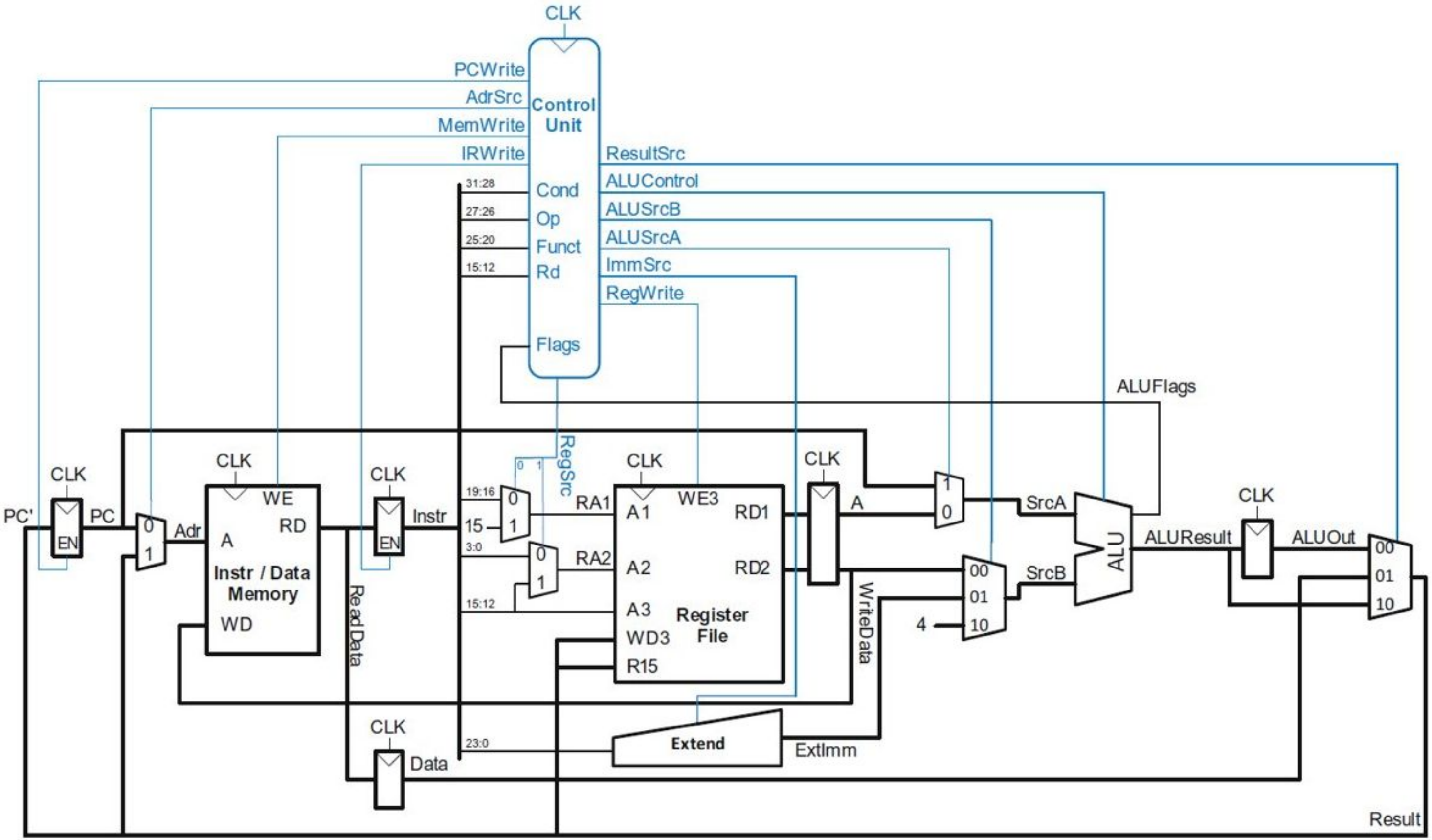
Calculate branch target address:

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

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

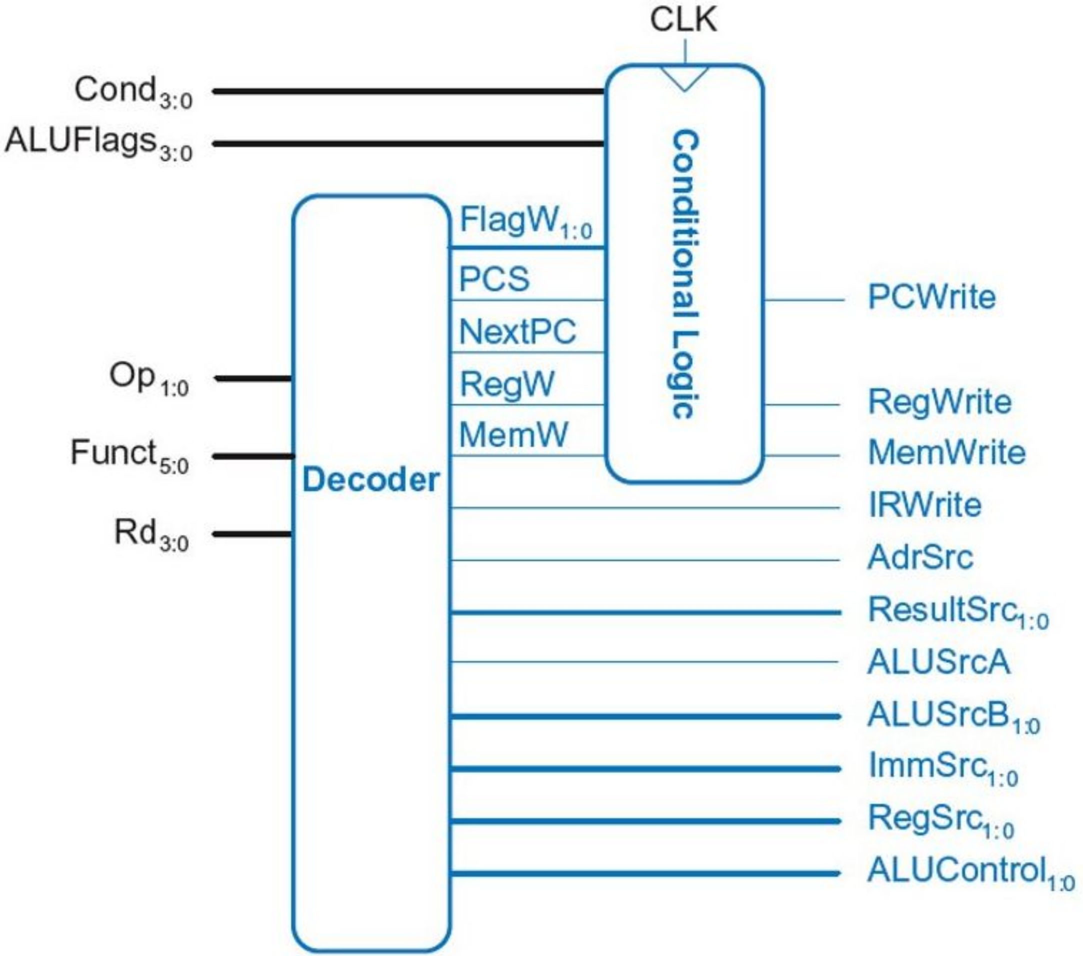


Multicycle ARM Processor (Control Unit)

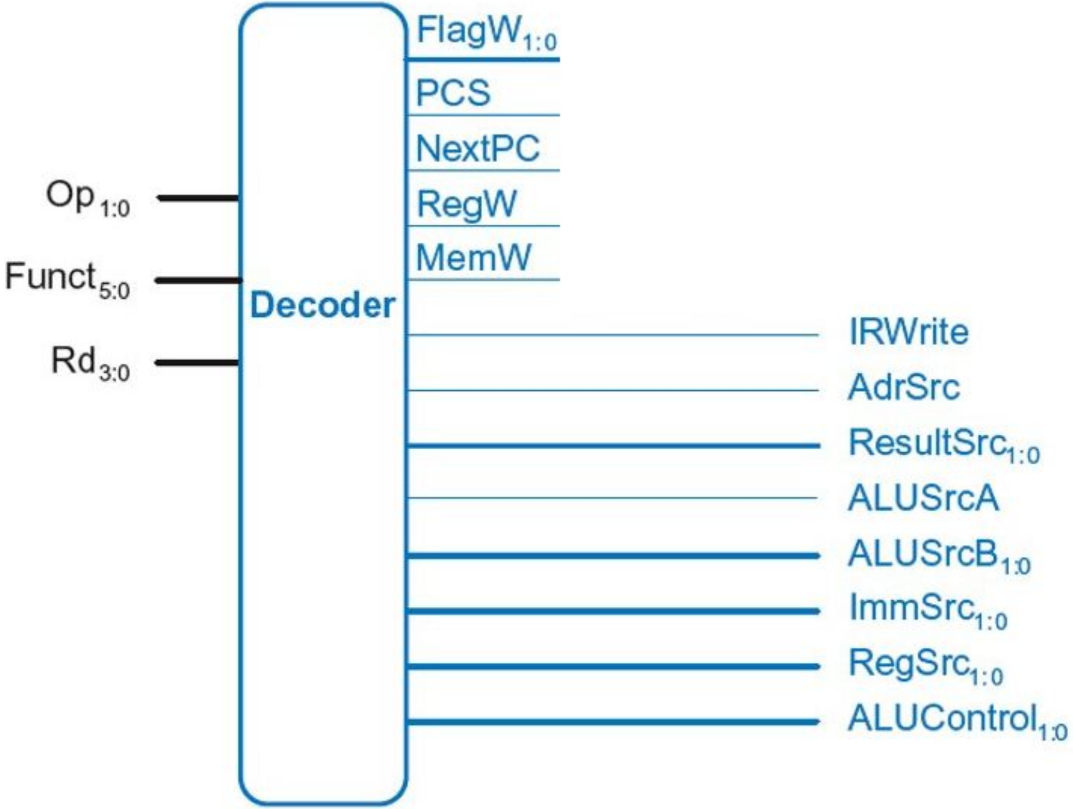


Multicycle Control

- First, discuss Decoder
- Then, Conditional Logic

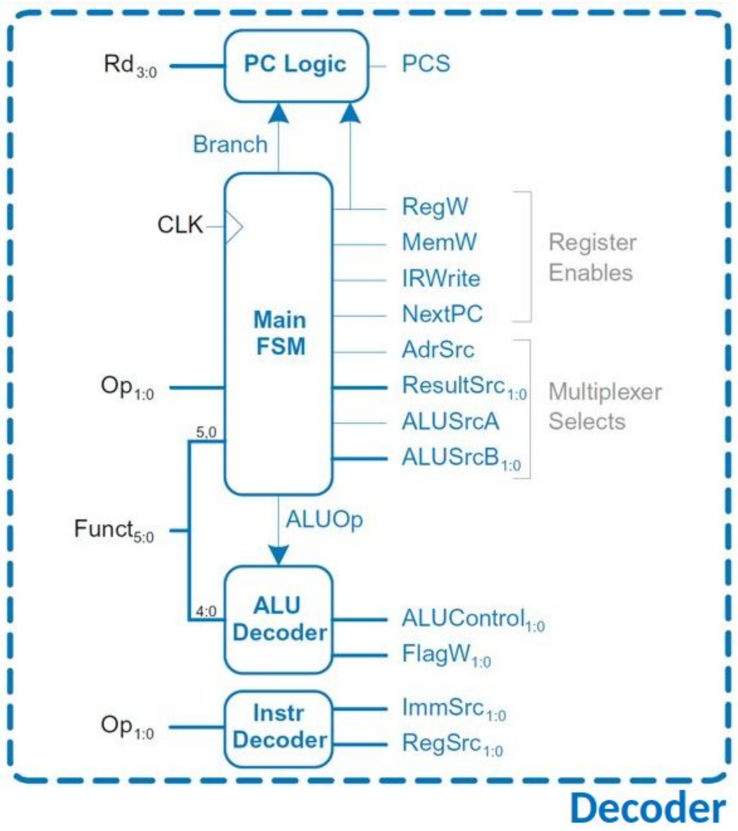


Multicycle Control: Decoder



Multicycle Control: Unwrapping Decoder

ALU Decoder and **PC Logic** same as single-cycle

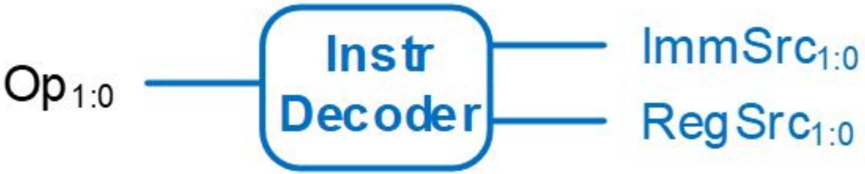


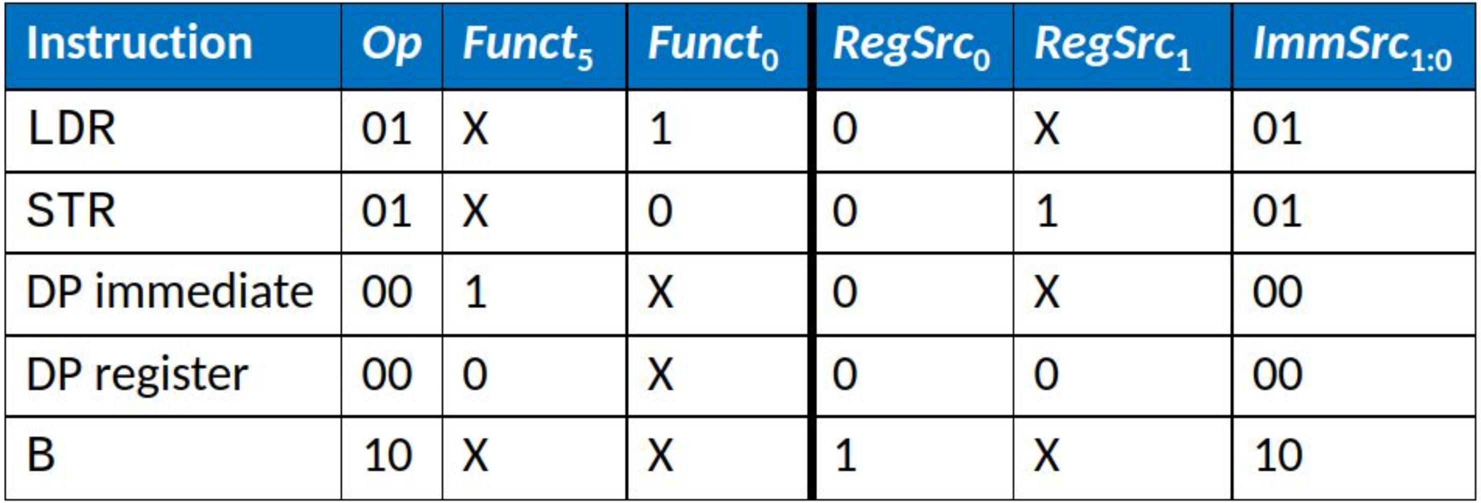
Multicycle Control: Instr Decoder

$$RegSrc_0 = (Op == 10_2)$$

$$RegSrc_1 = (Op == 01_2)$$

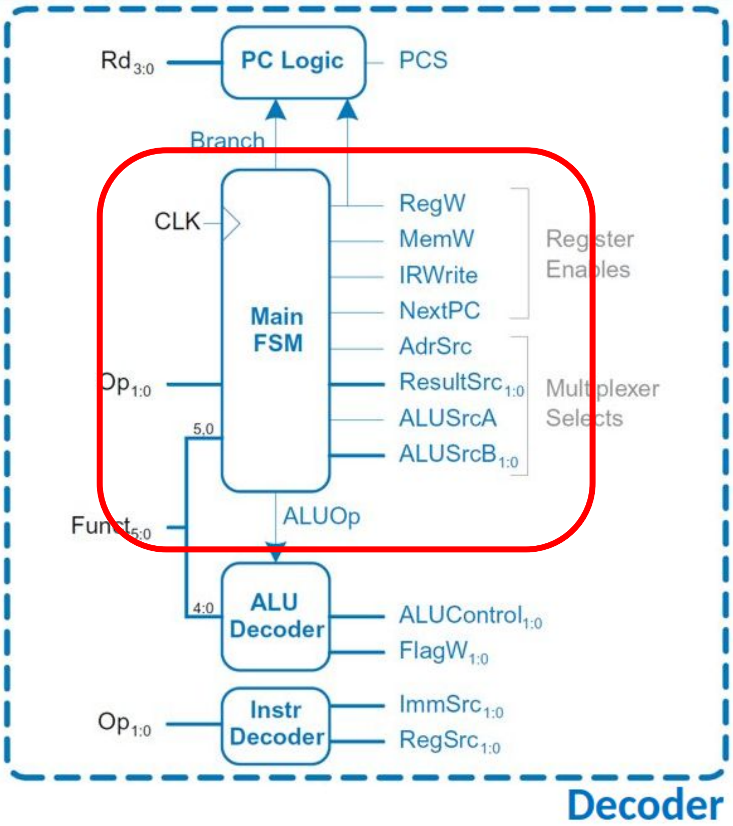
$$ImmSrc_{1:0} = Op$$



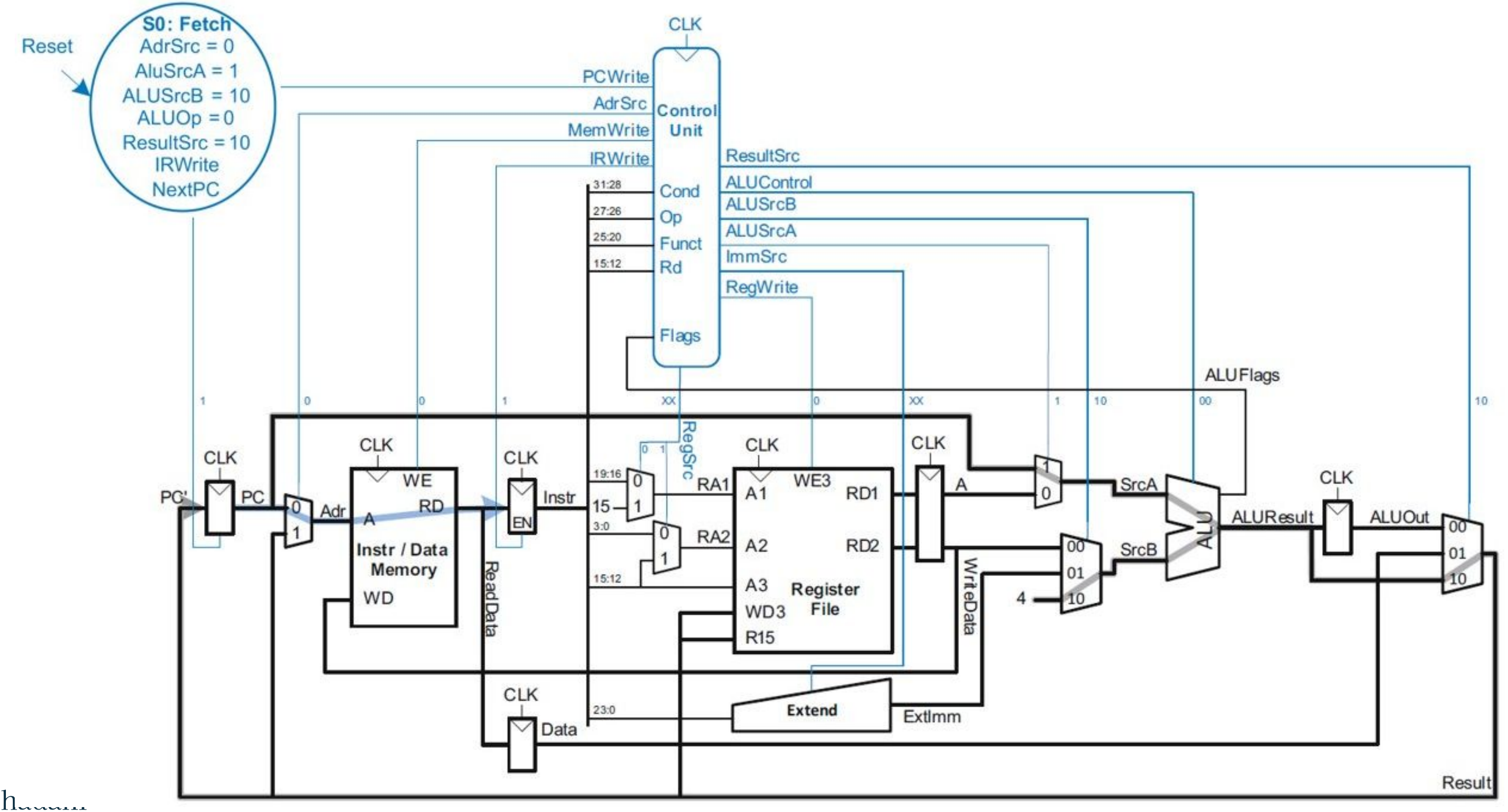


Instruction	Op	$Funct_5$	$Funct_0$	$RegSrc_0$	$RegSrc_1$	$ImmSrc_{1:0}$
LDR	01	X	1	0	X	01
STR	01	X	0	0	1	01
DP immediate	00	1	X	0	X	00
DP register	00	0	X	0	0	00
B	10	X	X	1	X	10

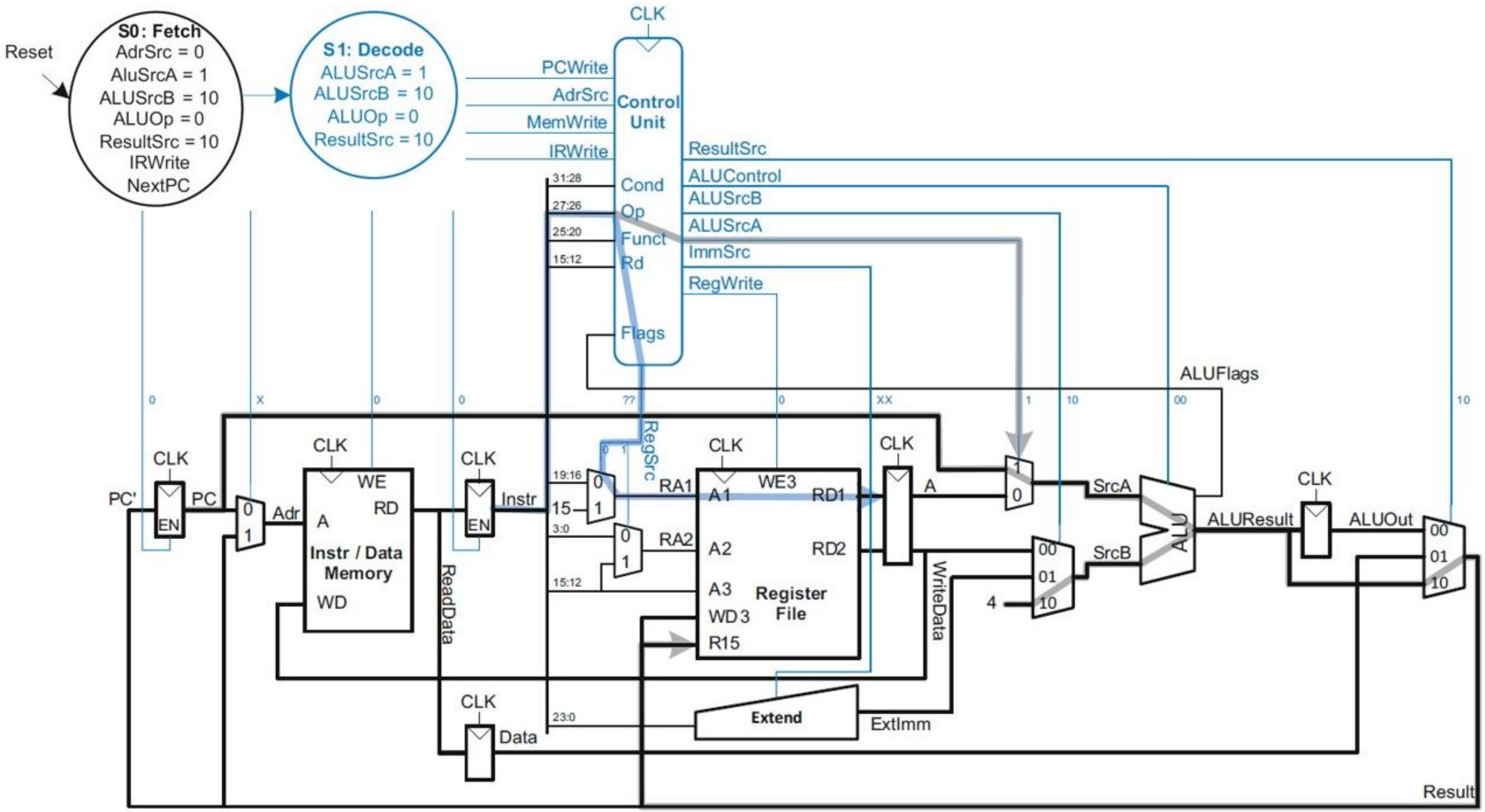
Multicycle Control: Main FSM



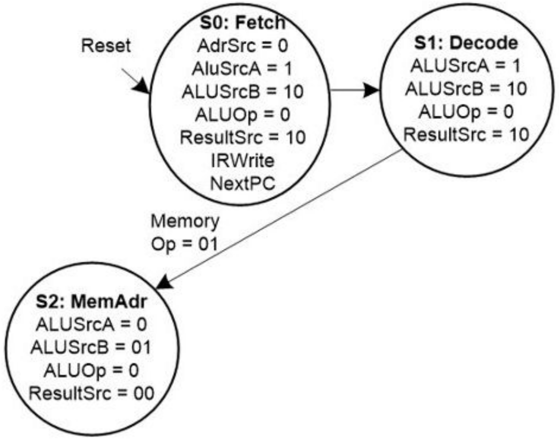
Main Controller FSM: Fetch



Main Controller FSM: Decode



Main Controller FSM: Address



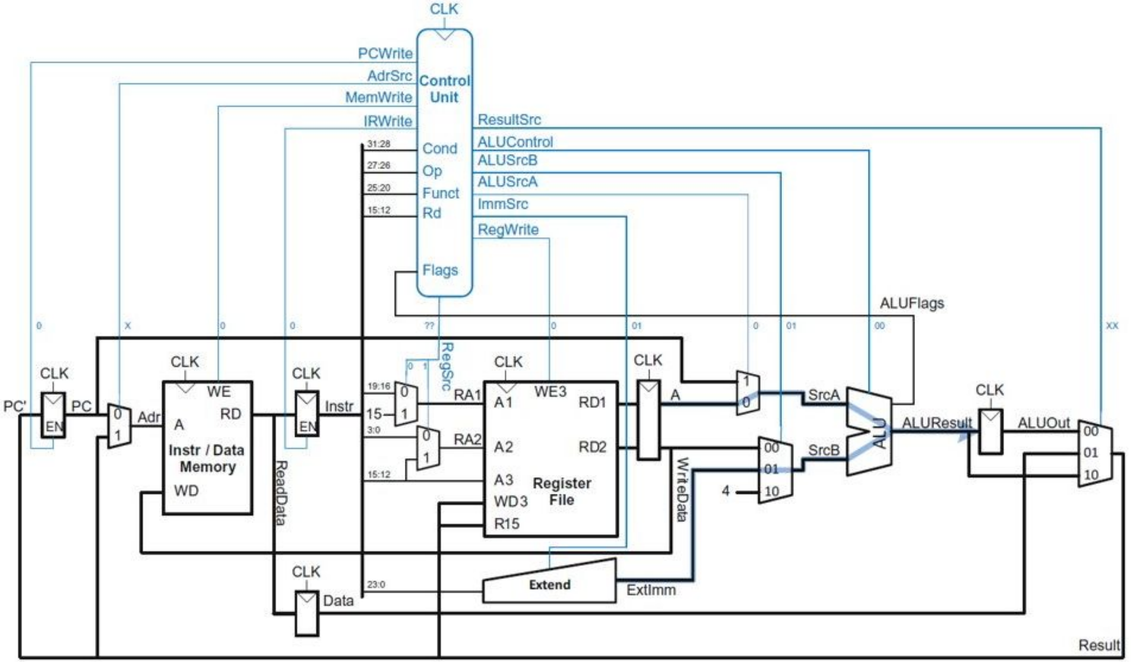
Reset

S0: Fetch
AdrSrc = 0
AluSrcA = 1
ALUSrcB = 10
ALUOp = 0
ResultSrc = 10
IRWrite
NextPC

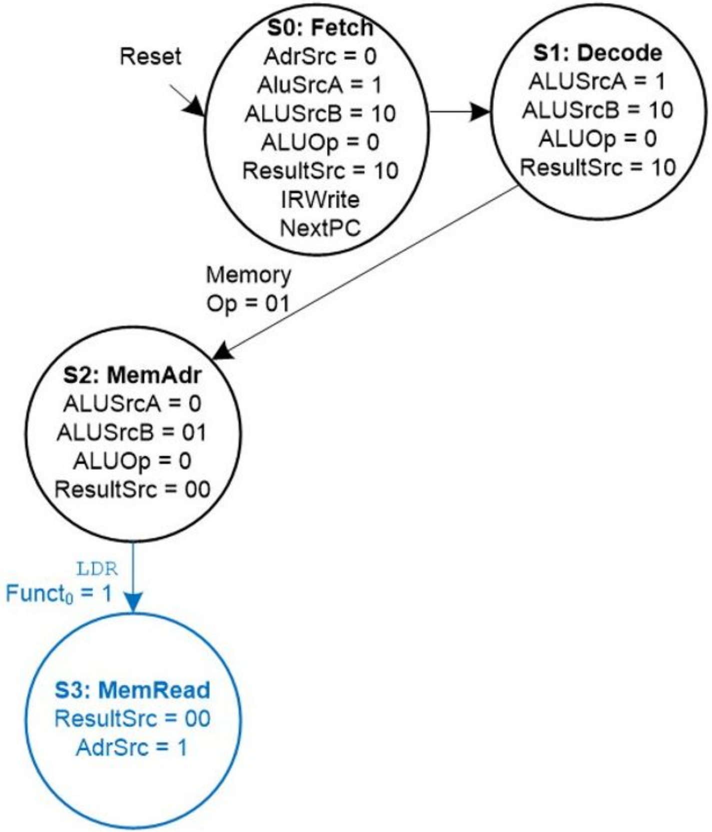
S1: Decode
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 0
ResultSrc = 10

Memory
Op = 01

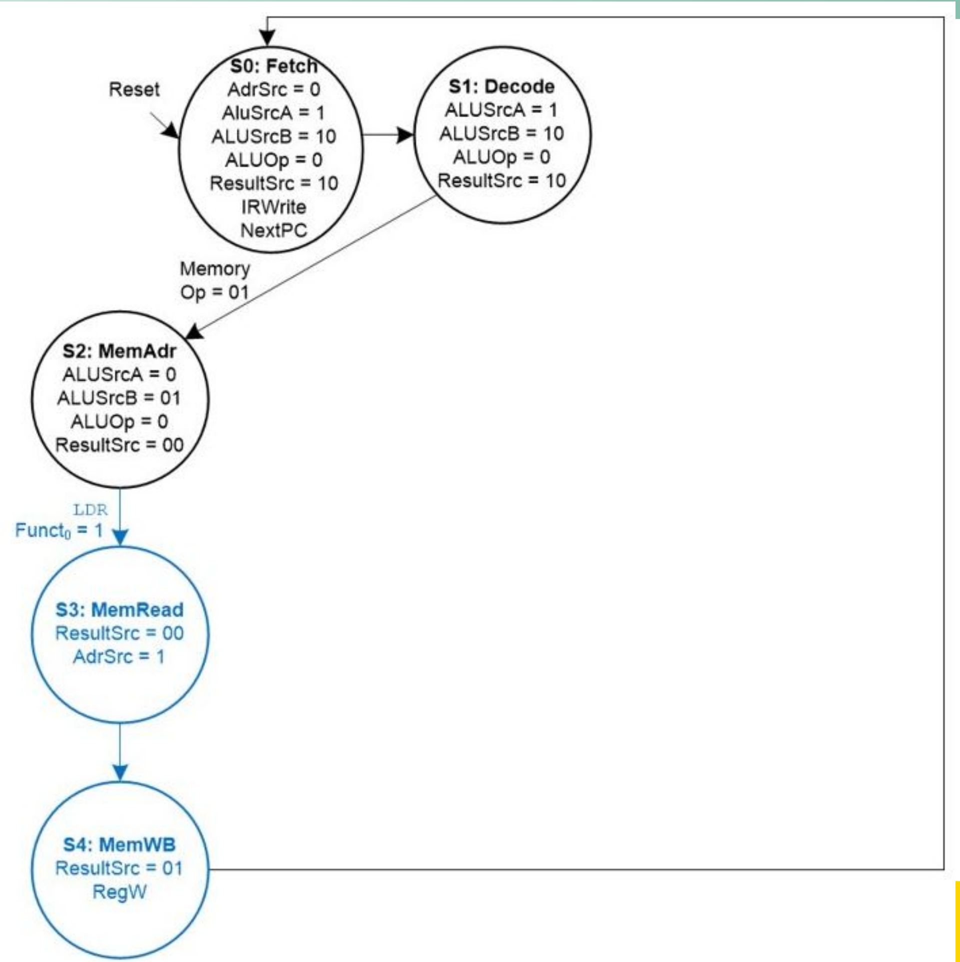
S2: MemAdr
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 0
ResultSrc = 00



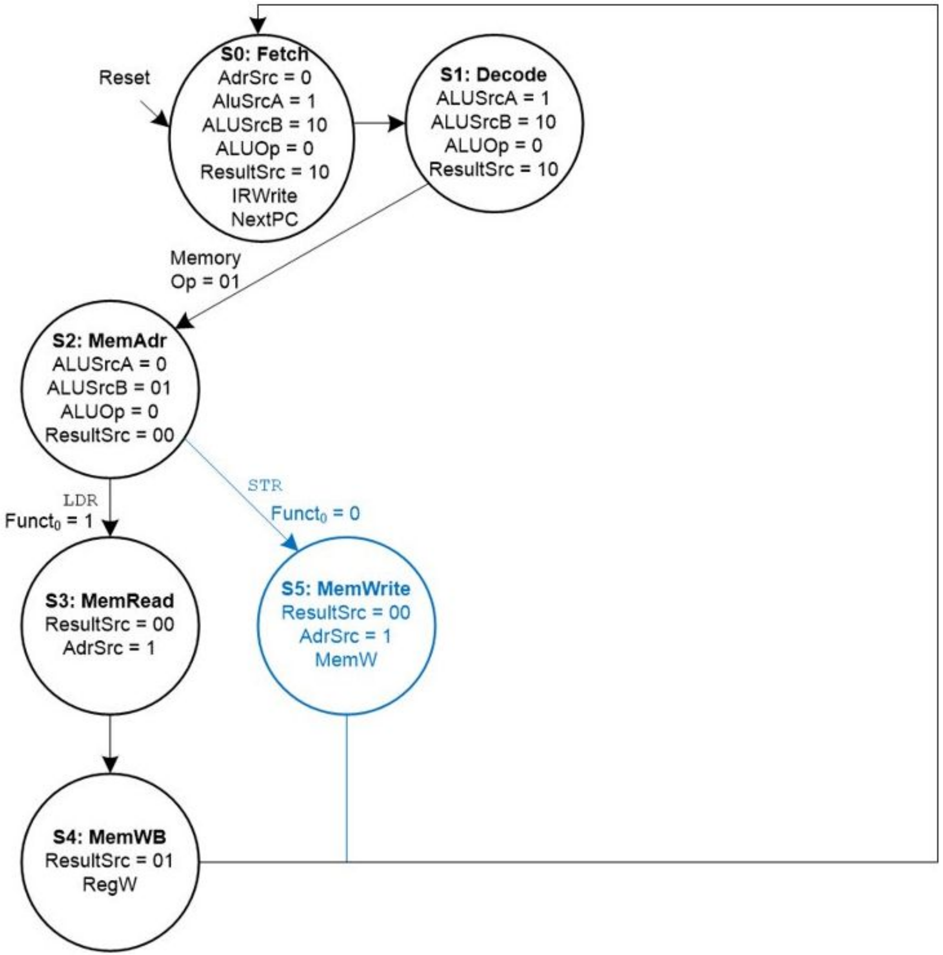
Main Controller FSM: Read Memory



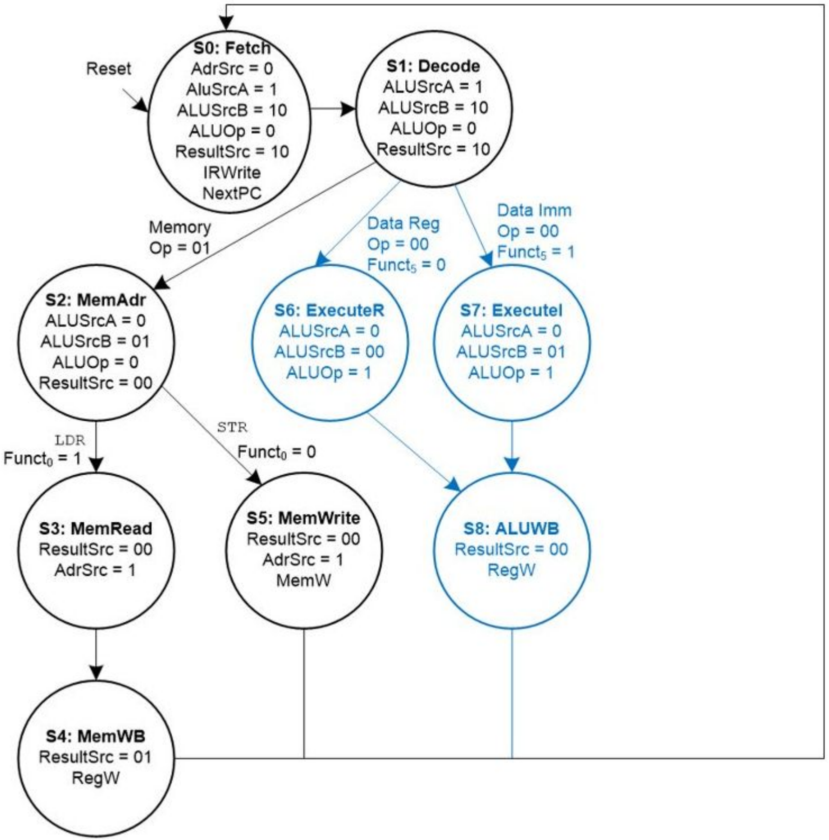
Main Controller FSM: LDR



Main Controller FSM: STR

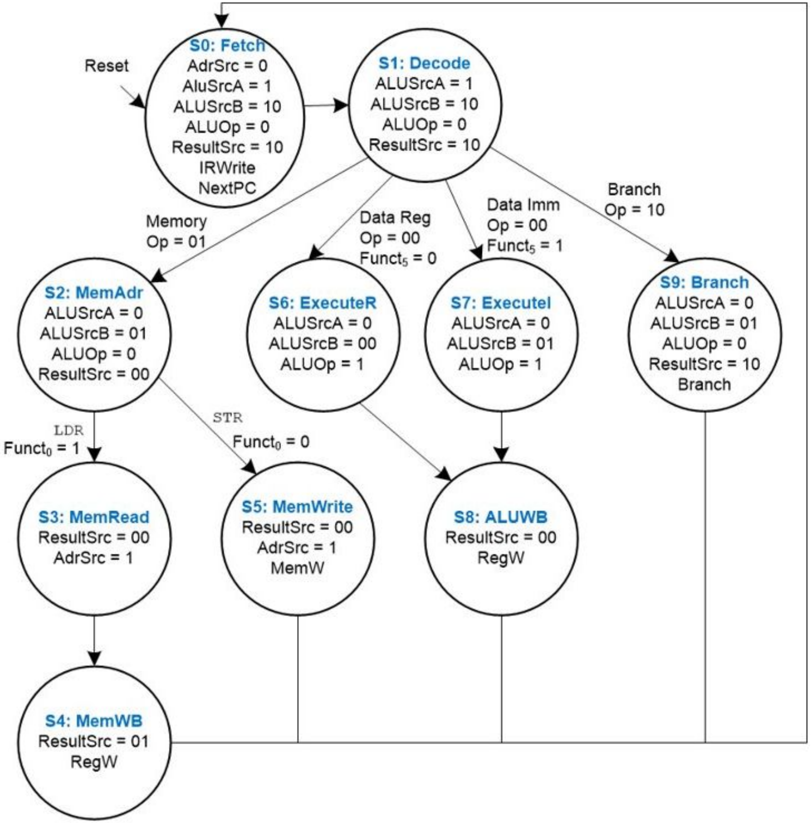


Main Controller FSM: Data-processing

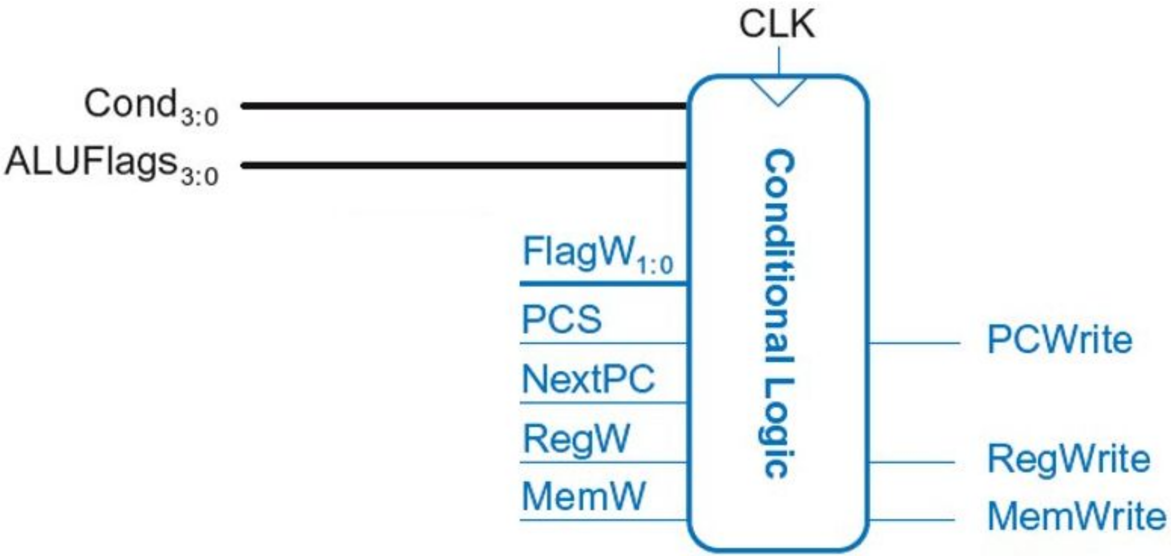


Multicycle Controller FSM

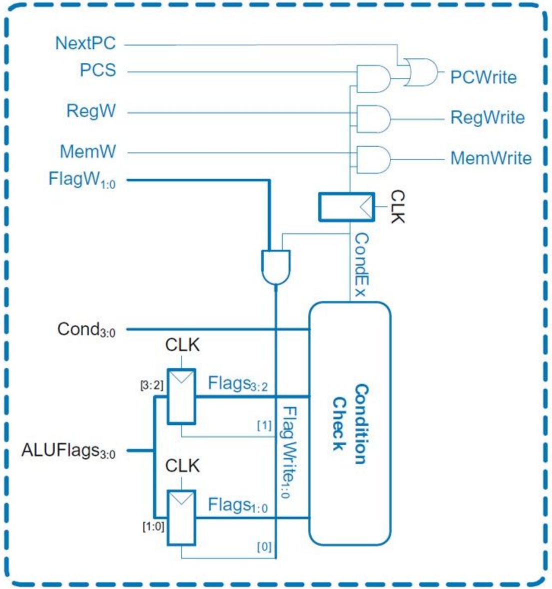
State	Datapath µOp
Fetch	Instr ←Mem[PC]; PC ← PC+4
Decode	ALUOut ← PC +4
MemAdr	ALUOut ← Rn + Imm
MemRead	Data ← Mem[ALUOut]
MemWB	Rd ← Data
MemWrite	Mem[ALUOut] ← Rd
ExecuteR	ALUOut ← Rn op Rm
ExecuteI	ALUOut ← Rn op Imm
ALUWB	Rd ← ALUOut
Branch	PC ← R15 + offset



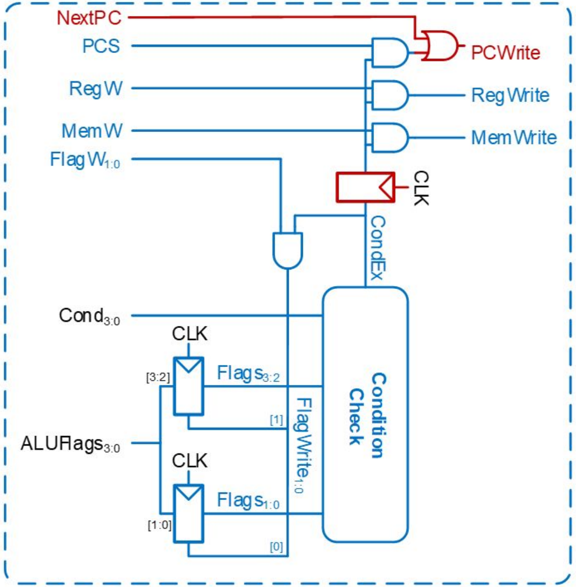
Multicycle Control: Cond. Logic



Single-Cycle vs Multi-cycle Conditional Logic



Single-Cycle



Multi-Cycle

- **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:
  - 3 cycles: B
  - 4 cycles: DP, STR
  - 5 cycles: LDR
- CPI is weighted average
- SPECINT2000 benchmark:
  - 25% loads
  - 10% stores
  - 13% branches
  - 52% R-type

https://www.spec.org/cpu2000/CINT2000/

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

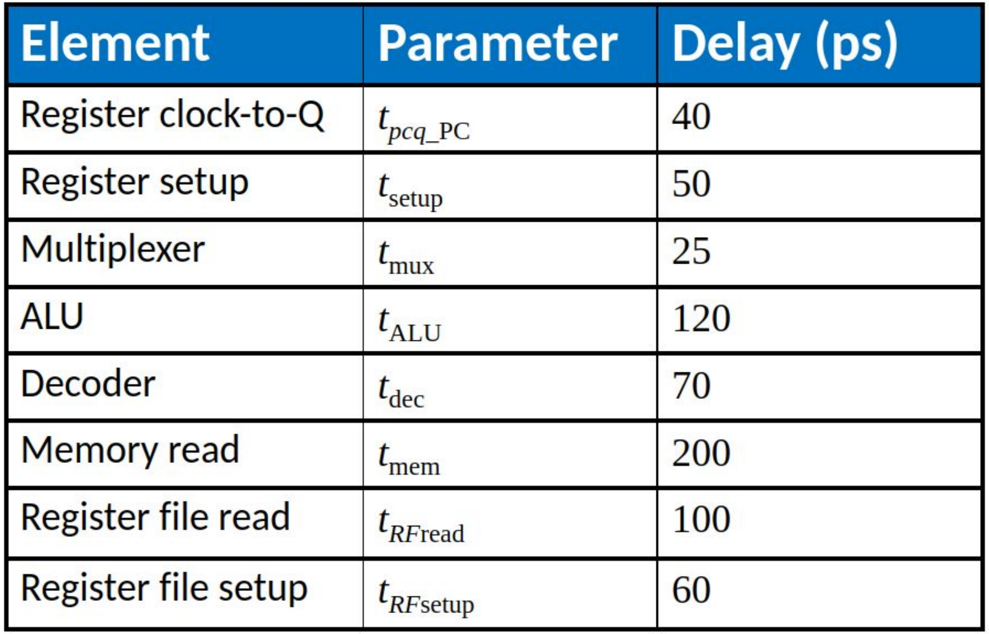
Multicycle Processor Performance

Multicycle critical path:

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

$Tc_2 = tpcq + 2tmux + \max(tALU + tmux,\ tmem) + tsetup$

$Tc_2 = \ ?$



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

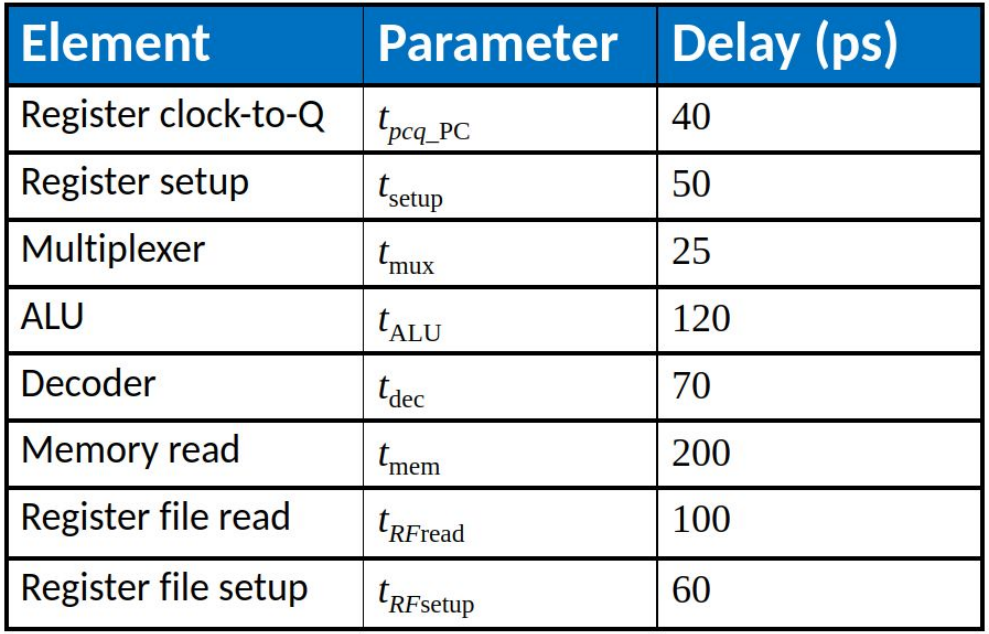
Multicycle Processor Performance

Multicycle critical path:

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

$$\mathbf{T c_2 = t pcq + 2 t mux + \max(t ALU + t mux, t mem) + t setup}$$

$$\begin{aligned} Tc_2 &= t_{pcq} + 2t_{\text{mux}} + \max[t_{\text{ALU}} + t_{\text{mux}}, t_{\text{mem}}] + t_{\text{setup}} \\ &= [40 + 2(25) + 200 + 50]\ \text{ps} = \mathbf{340\ ps} \end{aligned}$$



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

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

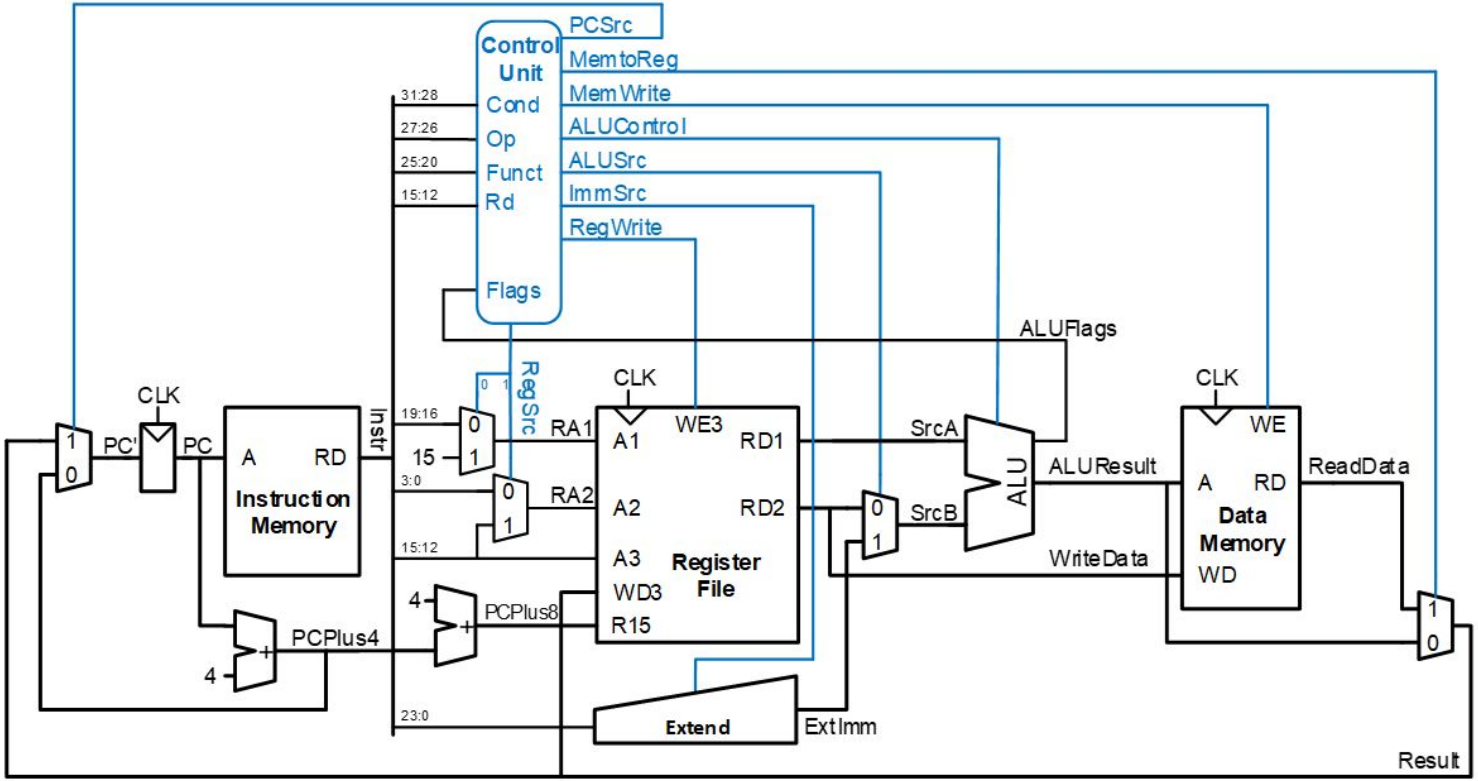
Execution Time = (# instructions) × CPI × T_c

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

= 140 seconds

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

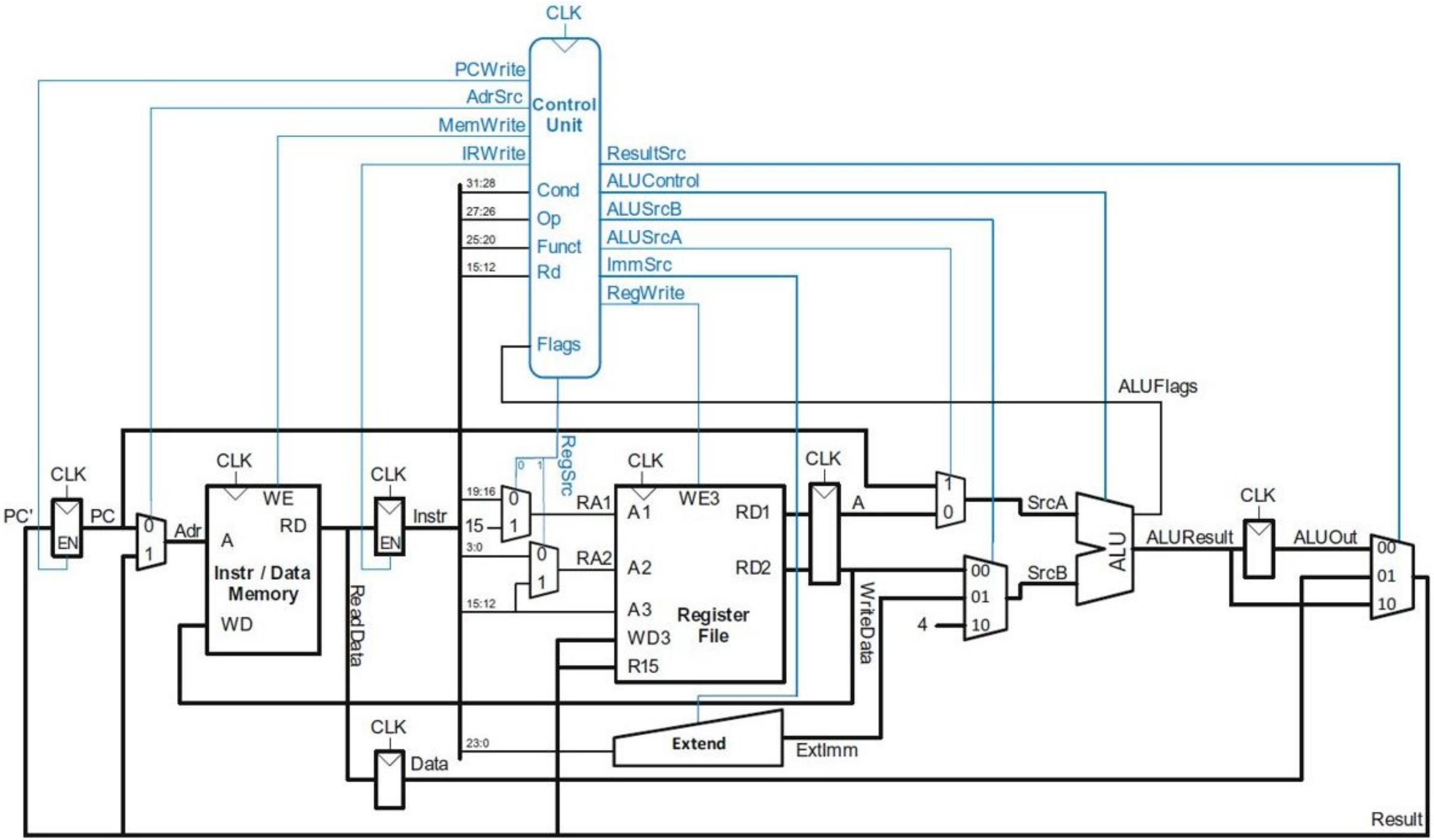
Recap Single-Cycle ARM Processor



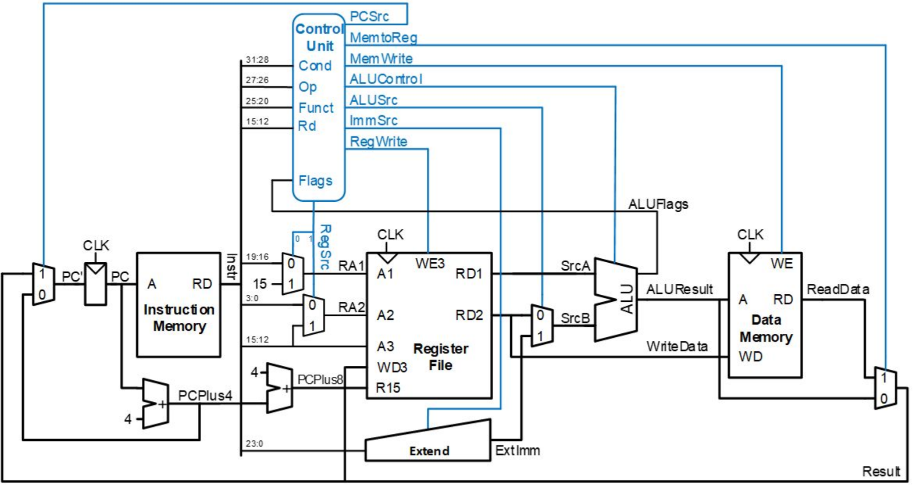
Spring 2025

Rahul Bhadani

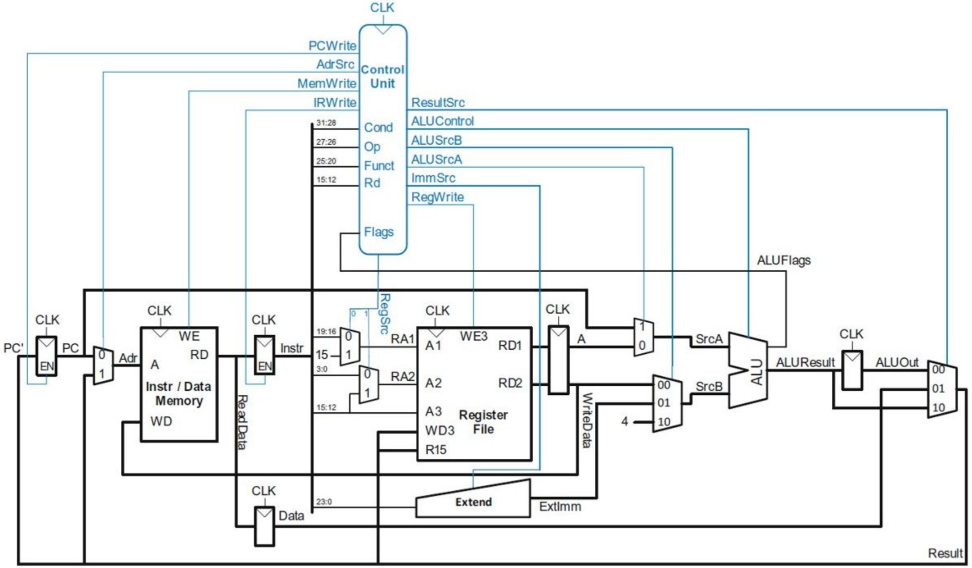
Recap Multicycle ARM Processor



Side by Side



Single-Cycle



Multi-Cycle