Microarchitecture Multi-cycle

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 - 2025I
PROF.: JGONZALEZ@UTEC.EDU.PE
SRC: HARRIS, HARRIS - DIGITAL DESIGN AND COMPUTER ARCHITECTURE

Executive Summary

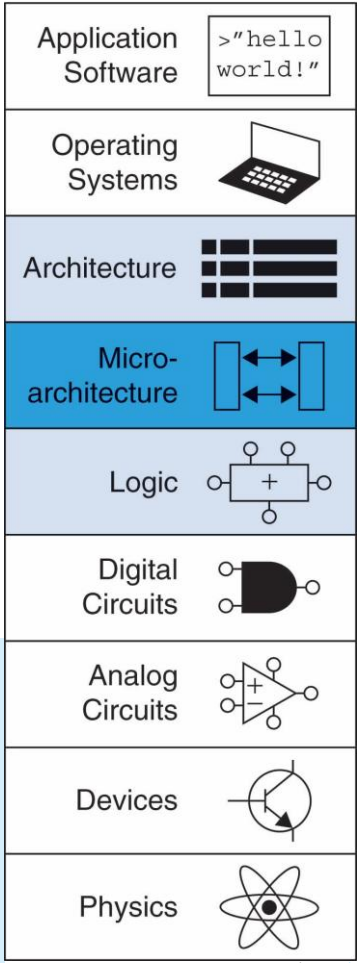
- **Motivation: Computer Architecture studies the ISA and Microarchitecture design.**
- **Problem: We need to detail an efficient and simple implementation of the ISA that enables the processor, known as Microarchitecture.**
- **Overview:**
  - **Define the ARM multicycle microarchitecture.**
  - **Details of the processor datapath and control.**
- **Conclusion: We can build a processor using building blocks to implement an ISA, this defines the microarchitecture.**

Outline

- Introduction
- Single-cycle
- Multicycle Processor
- Multicycle performance
- Conclusions

Recall: Microarchitecture Definition

- **Microarchitecture: how to implement an architecture in hardware.**
  - How the **underlying implementation (invisible to software) actually executes** instructions
    - Microarchitecture can execute instructions in any order **as long as it obeys the semantics specified by the ISA** when making the instruction results **visible to software** (to the programmer).
  - **Two main parts in the processor:**
    - **Datapath:** functional blocks
    - **Control:** control signals
  - **Multiple implementations** for a single architecture:
    1. **Single-cycle:** Each instruction executes in a single cycle
    2. **Multicycle:** Each instruction is broken up into series of shorter steps
    3. **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once



Outline

- Introduction
- Single-cycle
- Multicycle Processor
- Multicycle performance
- Conclusions

Single-cycle Operation

- **All six phases** of the instruction processing cycle **take a *single machine clock cycle*** to complete:
  - Fetch, Decode, Evaluate Address,Fetch Operands, Execute, Store Result
  - Do each phase take the same time (latency) for all instructions?

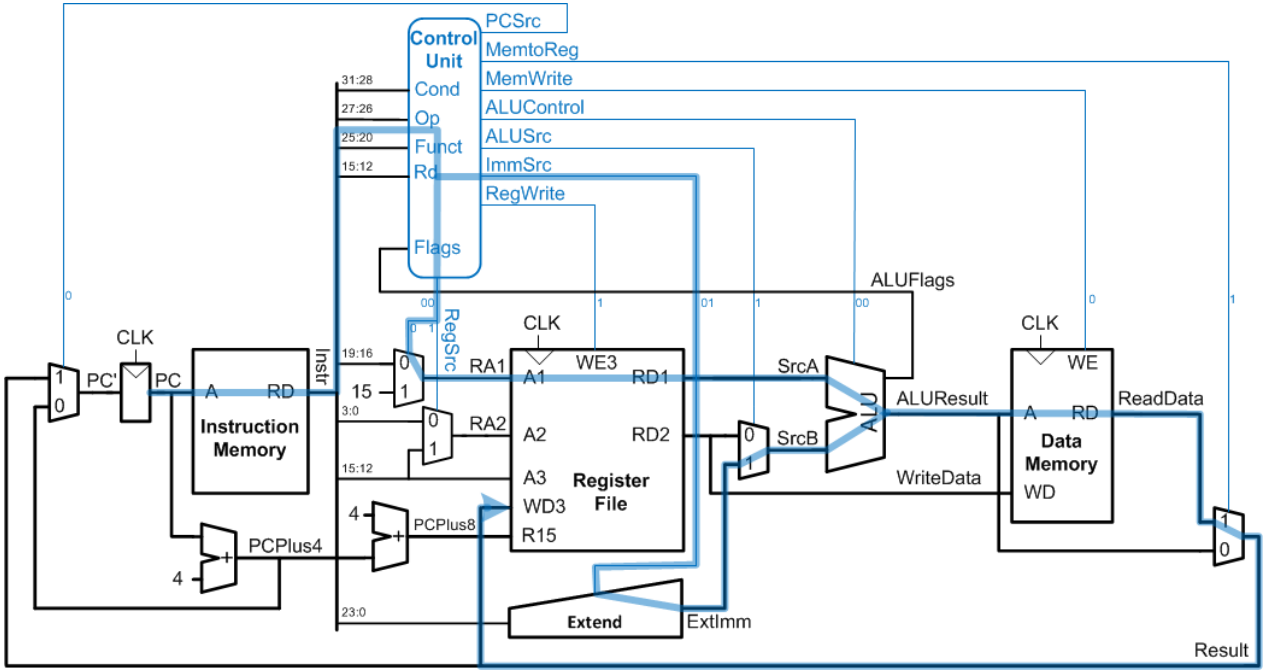
- **Every instruction takes 1 cycle to execute**
  - **CPI (Cycles per instruction) is strictly 1**

- **How long each instruction takes is determined by how long the slowest instruction takes to complete**
  - Even though many instructions do not need that long to execute
  - **Clock cycle time of the microarchitecture =** how long it takes to complete the **slowest instruction**

Instruction phases can be organized as 5 stages:

1. Instruction fetch **(IF)**
2. Instruction decode and register operand fetch **(ID/RF)**
3. Execute/Evaluate memory address **(EX/AG)**
4. Memory operand fetch **(MEM)**
5. Store/writeback result **(WB)**

Recall Single-Cycle Architecture



T_C limited by critical path, defined by instruction LDR

Outline

- Introduction
- Single Cycle
- Multicycle Processor
- Multicycle performance
- Conclusions

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:**
  - **Processor addresses these issues by breaking instruction into shorter steps**
  - shorter instructions take fewer steps
  - can re-use hardware
  - cycle time is faster

Why hardware overhead?

Multi-Cycle Microarchitectures

- Determine **clock cycle time independently of instruction processing time**
- **Each instruction takes as many clock cycles as it needs to take**
  - Multiple state transitions per instruction
  - The states followed by each instruction is different

AS = Architectural (programmer visible) state **at the beginning** of an instruction

Step 1: Process part of instruction **in one clock cycle**

Step 2: Process part of **instruction in the next clock cycle**

...

AS' = Architectural (programmer visible) state **at the end of a clock cycle**

Hardware overhead for registers:

- Need to store the intermediate results at the end of each clock cycle
- **Register setup/hold overhead** paid multiple times for an instruction

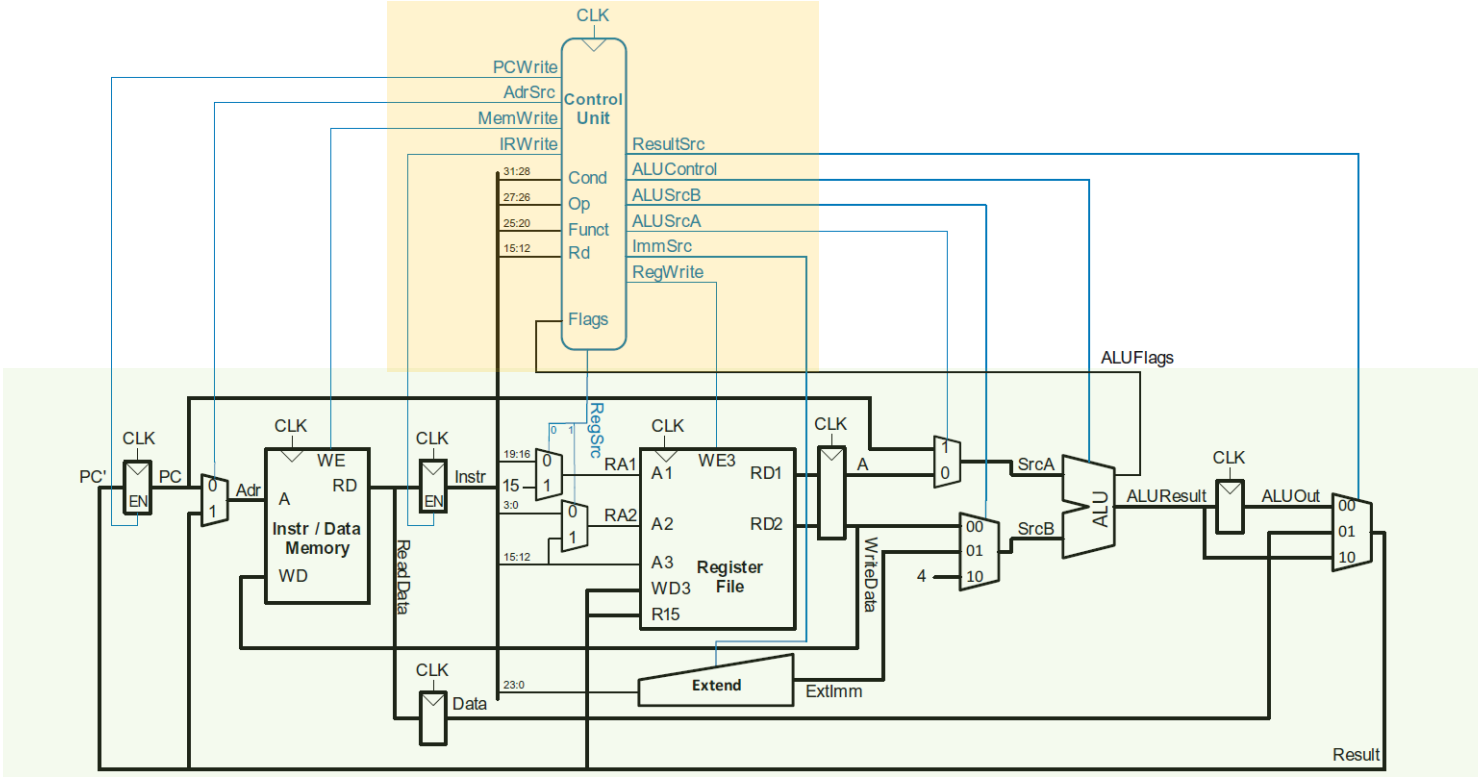
How Do We Implement This?

- **Instruction processing cycle divided into "states"**
  - **A stage in the instruction processing cycle can take multiple states**
  - **Implementation of the "process instruction" step as a finite state machine** that sequences between states and eventually returns back to the "fetch instruction" state
  - **A state is defined by the control signals asserted in it**
  - Control signals for the next state are determined in current state

- **The behavior of the entire processor is specified fully by a *finite state machine***
- In a state (transitions on clock cycle), **control signals for two things:**
  1. How the datapath should process the data
  2. How to generate the control signals for the (next) clock cycle

Maurice Wilkes, **"The Best Way to Design an Automatic Calculating Machine,"** Manchester Univ. Computer Inaugural Conf., 1951. Introduced **the concept of microcoded/microprogrammed machines.**

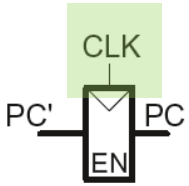
Multicycle ARM Processor

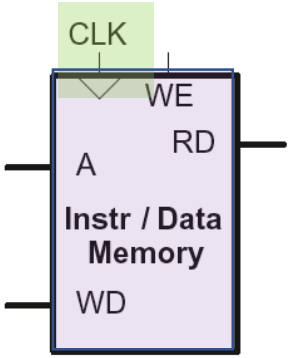


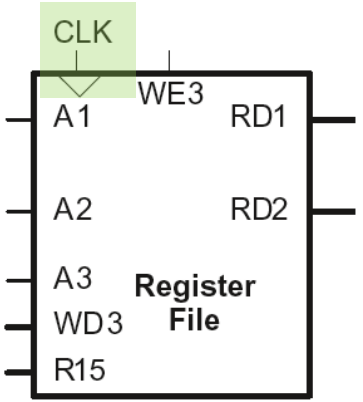
Same design steps as single-cycle: first datapath, then control.

Multicycle State Elements

Instruction and Data memories with a single unified memory

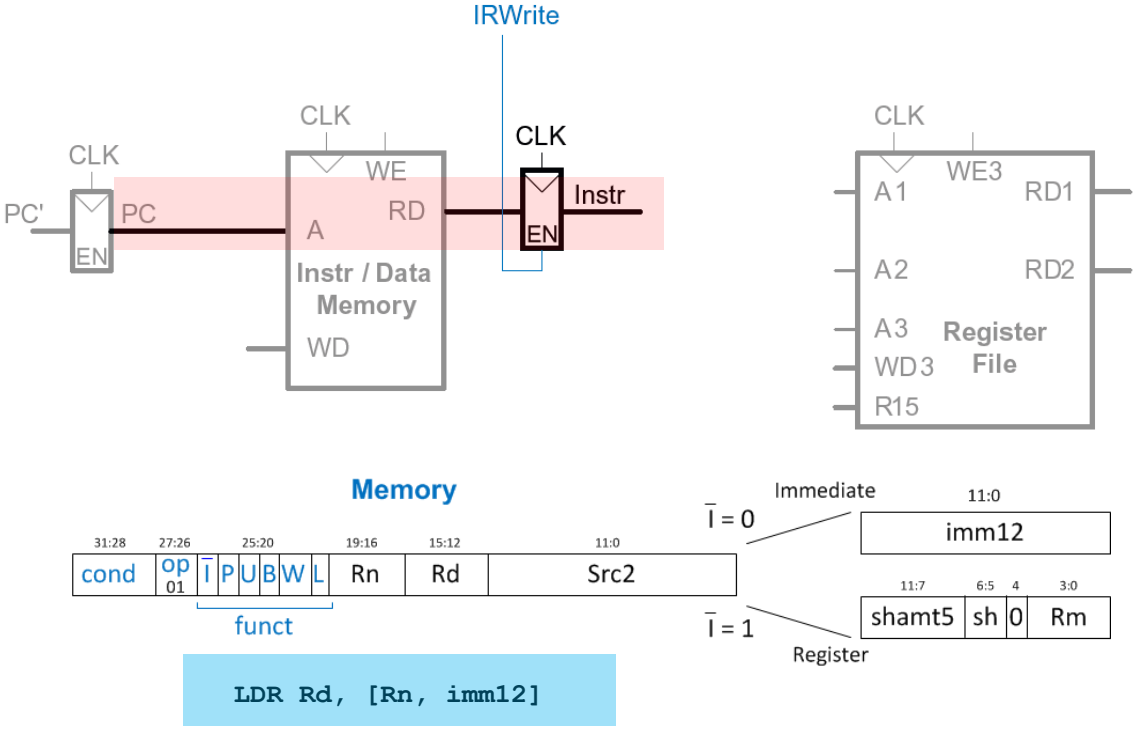






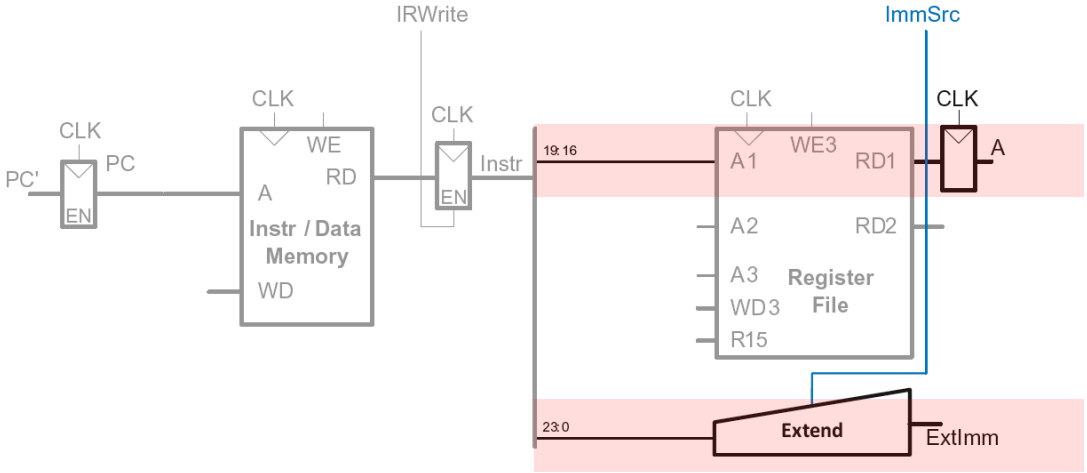
Multicycle Datapath: Instruction Fetch

STEP 1: Fetch instruction



LDR Register Read

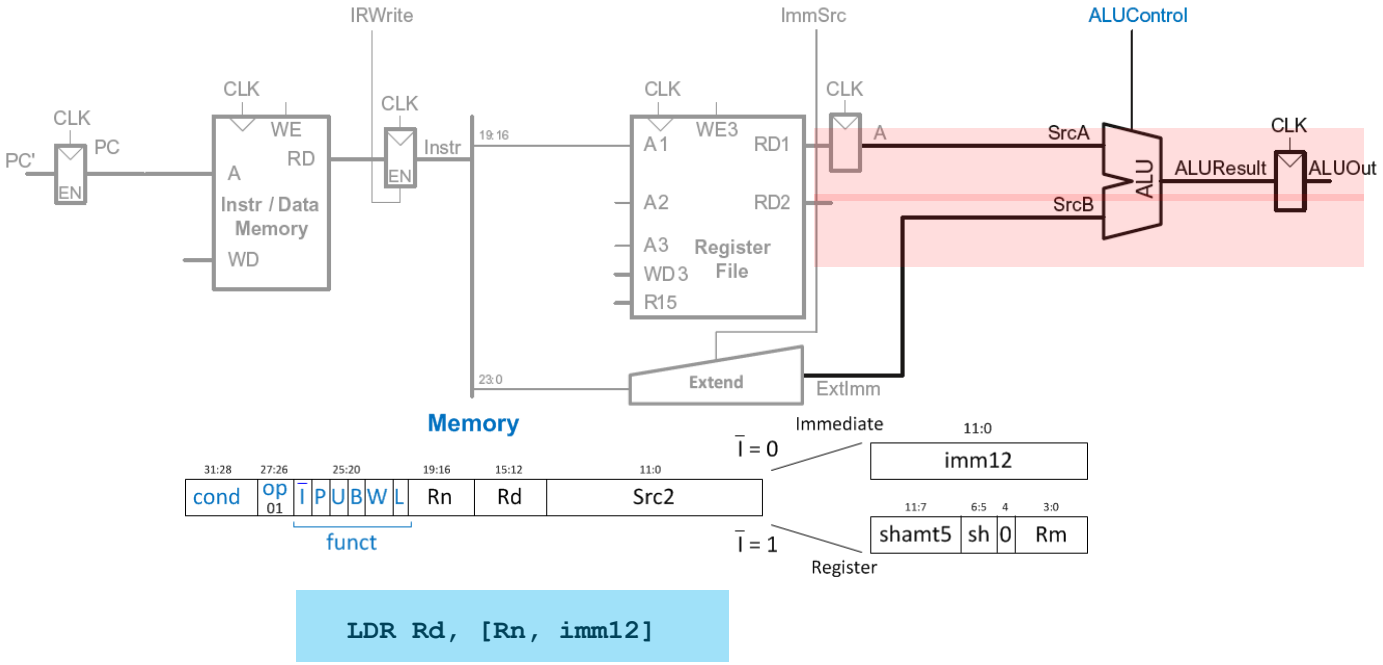
STEP 2: Read source operands from RF




```
LDR Rd, [Rn, imm12]
```

LDR Address

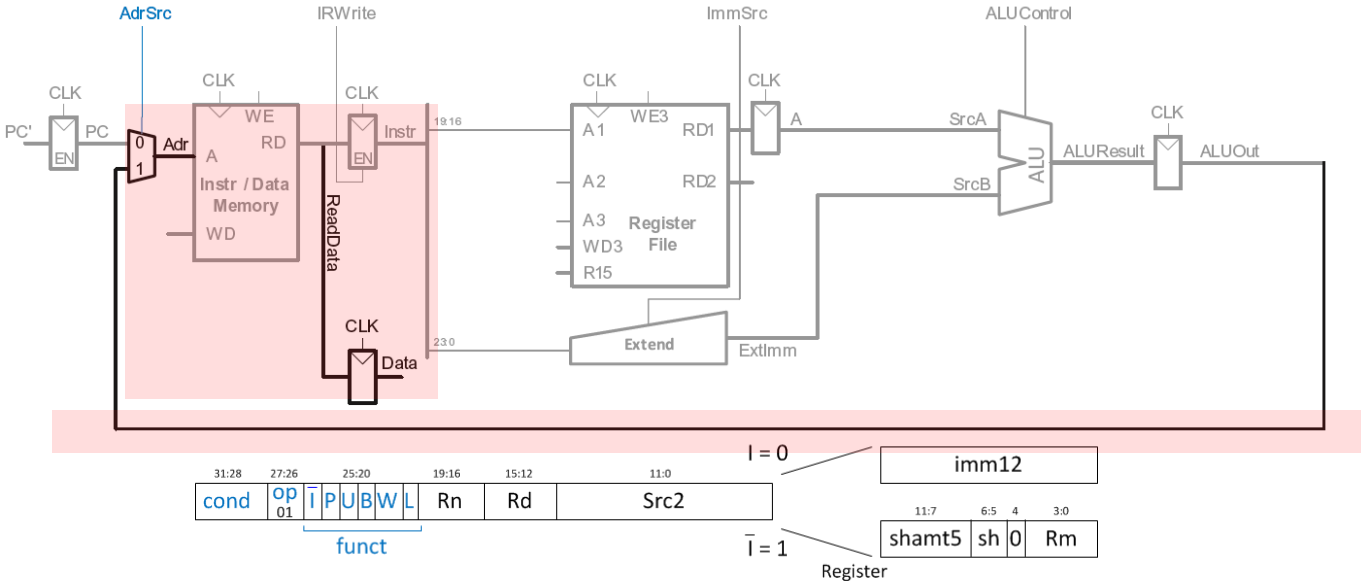
STEP 3: Compute the memory address




```
LDR Rd, [Rn, imm12]
```

LDR Memory Read

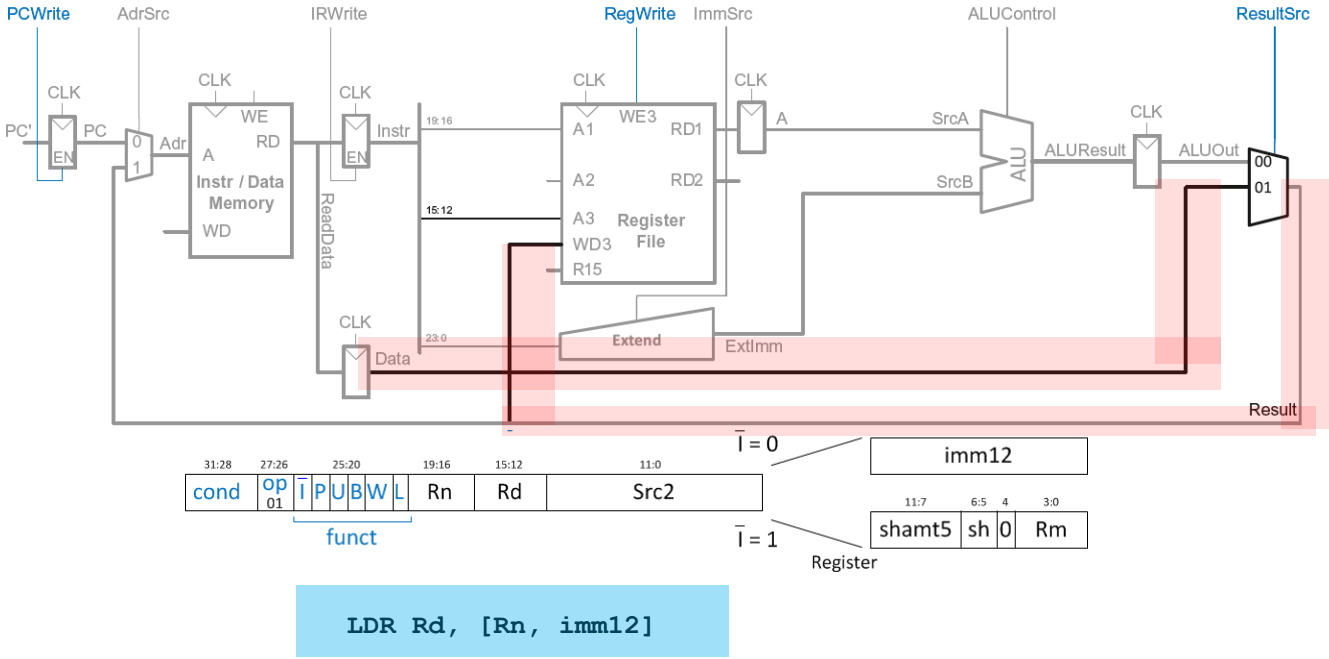
STEP 4: Read data from memory




```
LDR Rd, [Rn, imm12]
```

LDR Write Register

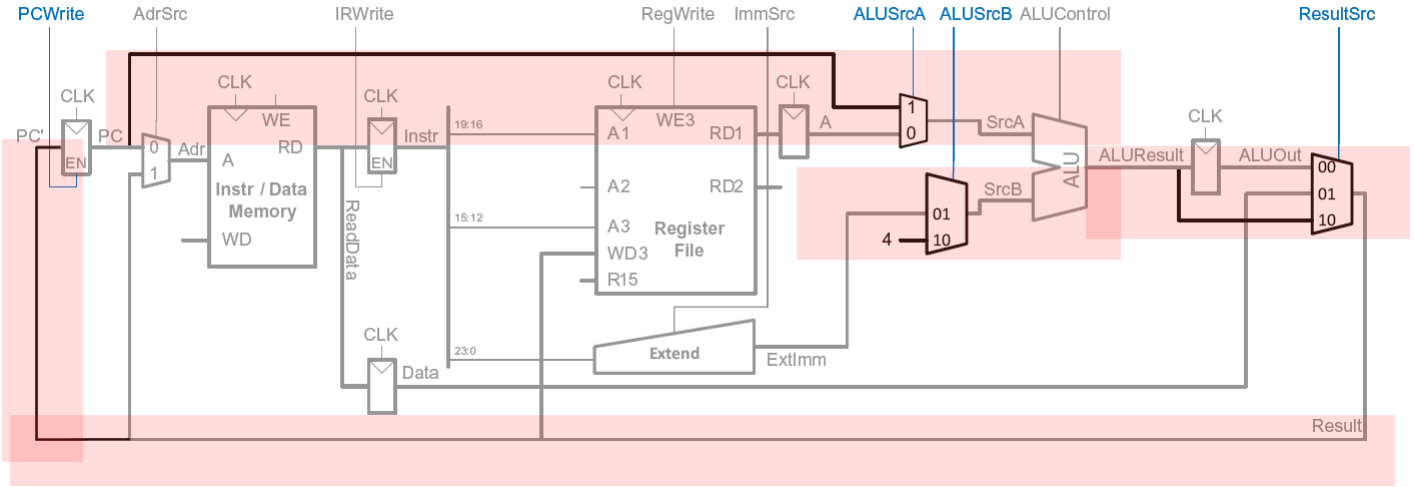
STEP 5: Write data back to register file




```
LDR Rd, [Rn, imm12]
```

Increment PC

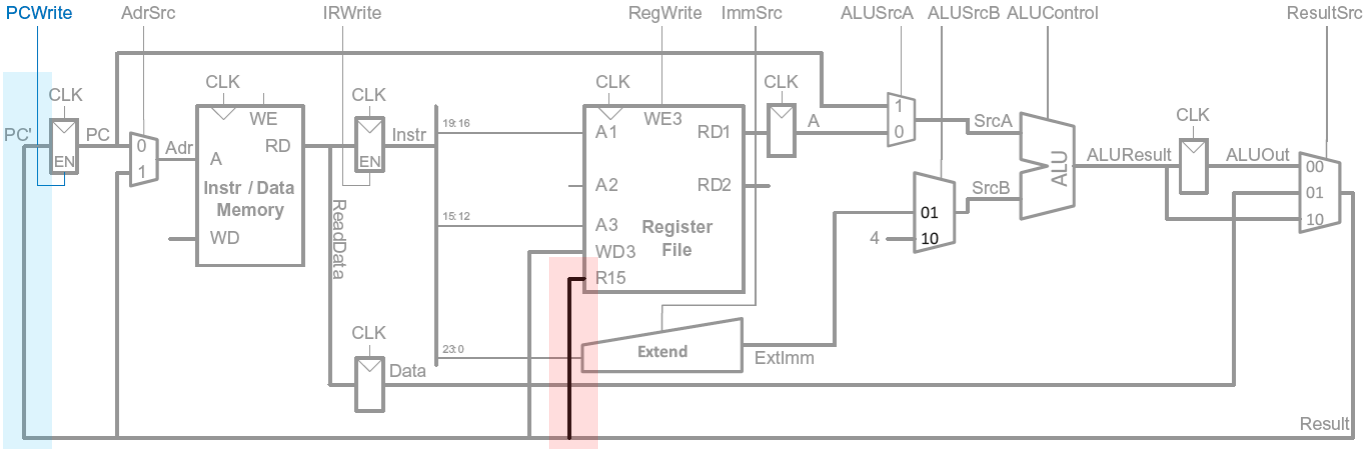
STEP 6: Increment PC



Access to PC

PC can be read/written by instruction

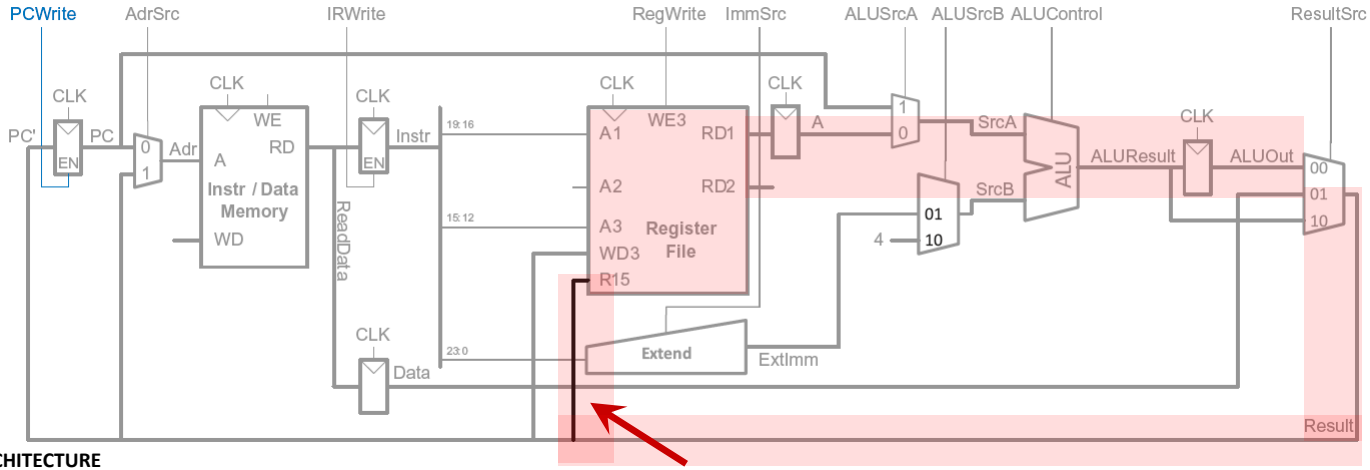
- **Read:** R15 (PC+8) available in Register File
- **Write:** Be able to write result of instruction to PC



Read to PC (R15)

Example: `ADD R1, R15, R2`

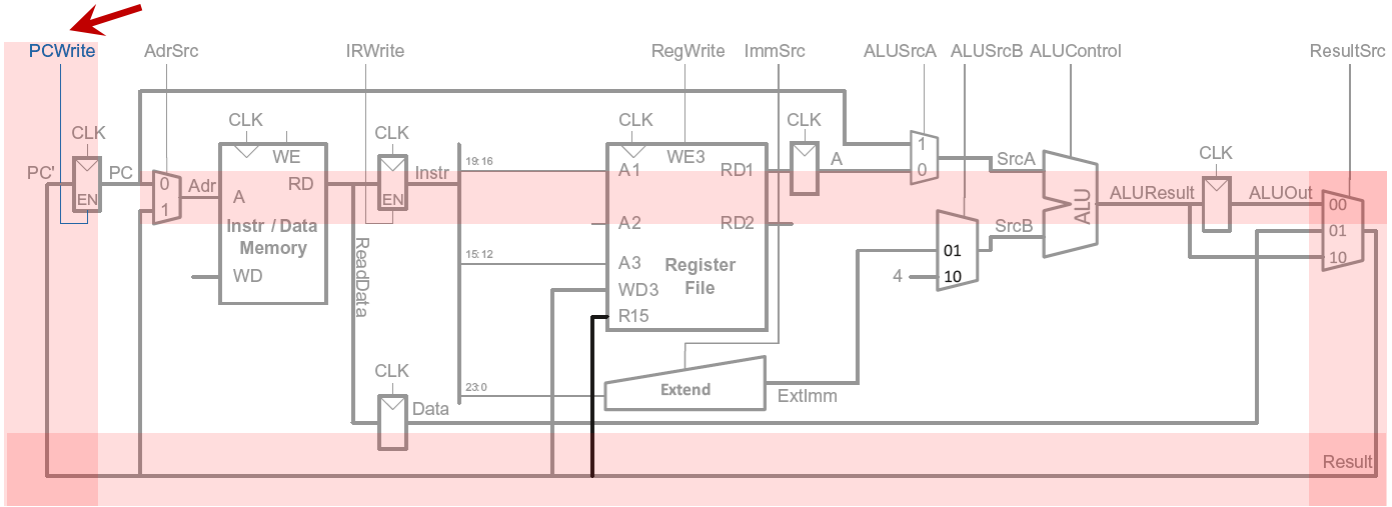
- R15 needs to be read as PC+8 from Register File (RF) in **2nd step**
- So **(also in 2nd step)** PC + 8 is produced by ALU and routed to R15 input of RF
  - *SrcA* = *PC* (which was already updated in step 1 to PC+4)
  - *SrcB* = 4
  - *ALUResult* = *PC* + 8
- ALUResult is fed to R15 input port of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



Write to PC (R15)

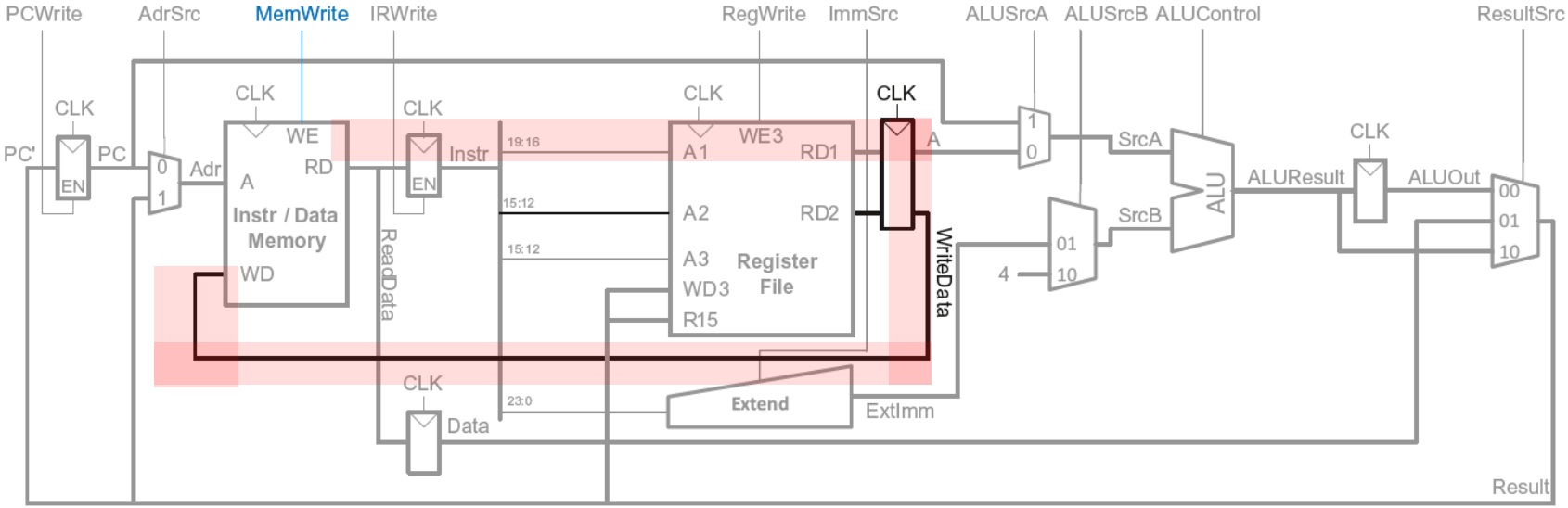
Example: `SUB R15, R8, R3`

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



Multicycle Datapath: STR

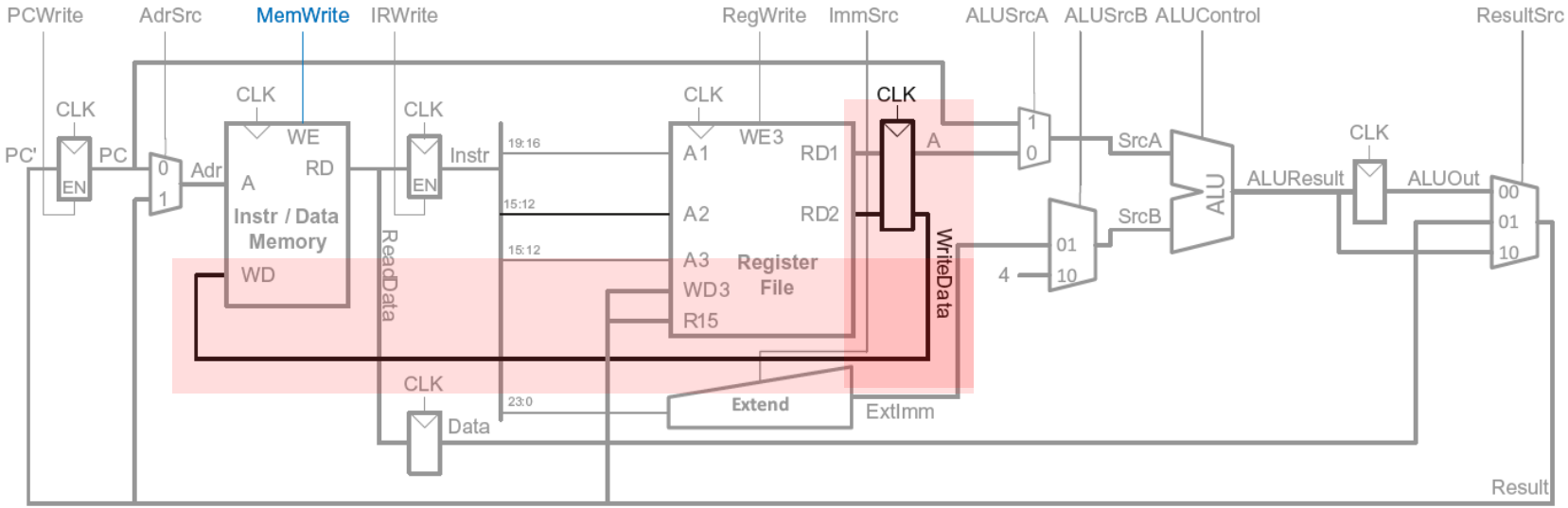
Write data in Rn to memory



Multicycle Datapath: Data-processing

With immediate addressing (i.e., an immediate *Src2*):

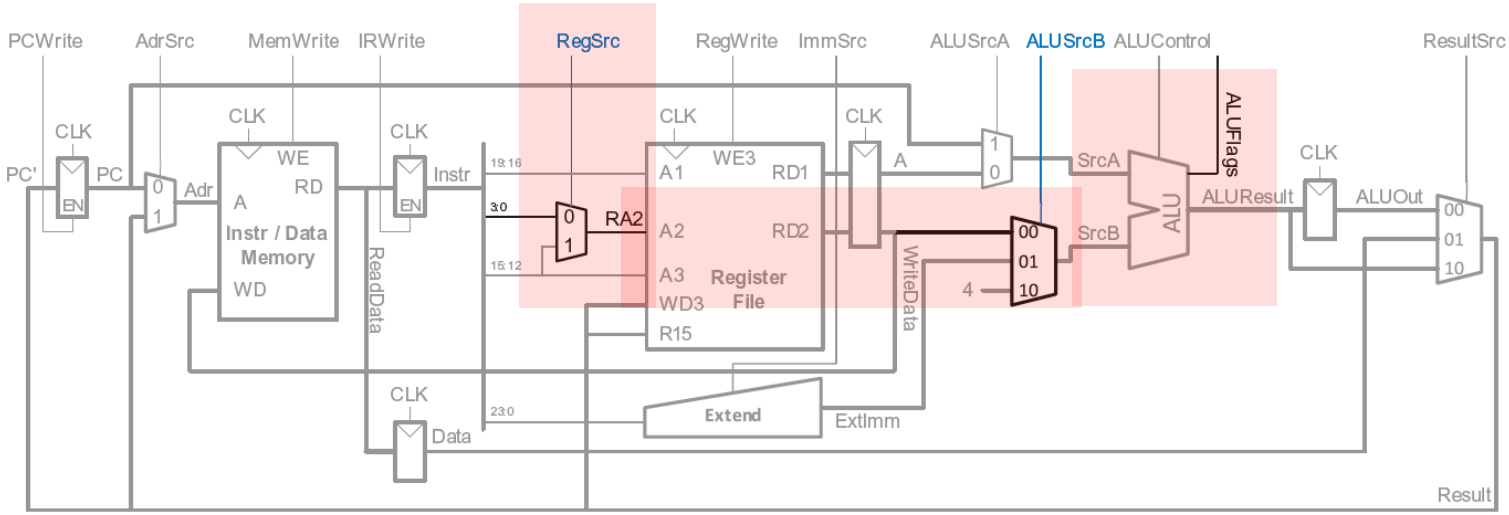
- **No additional changes needed** for datapath



Multicycle Datapath: Data-processing

With register addressing (register *Src2*):

- Read from Rn and Rm

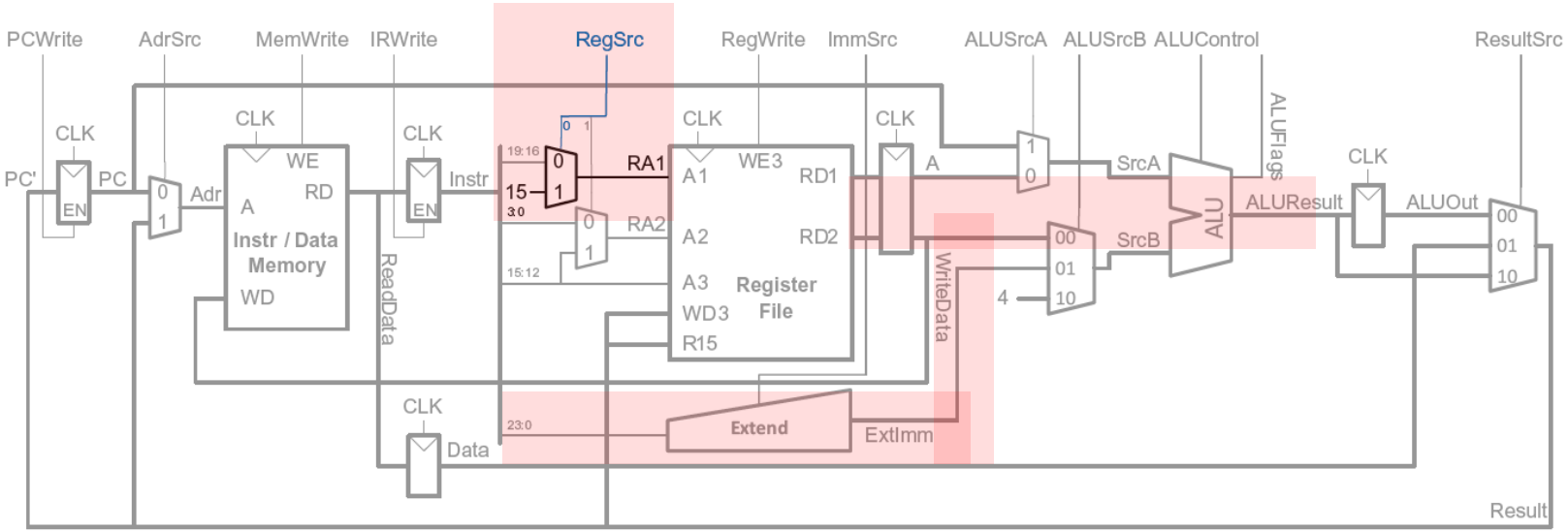


Multicycle Datapath: B

Calculate branch target address:

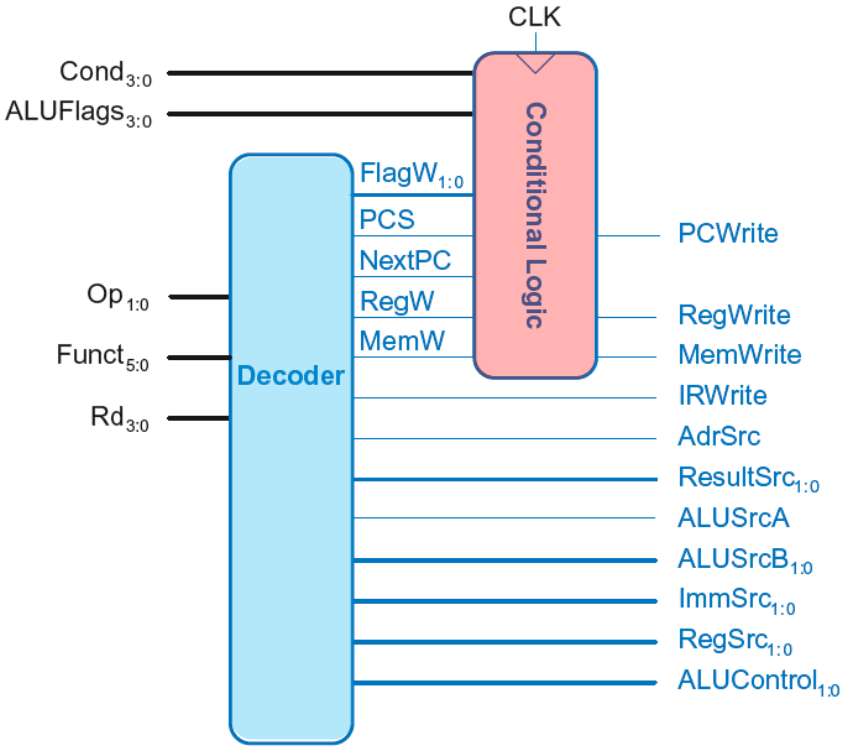
BTA = (*ExtImm*) + (PC+8)

ExtImm = Imm24 << 2 and sign-extended

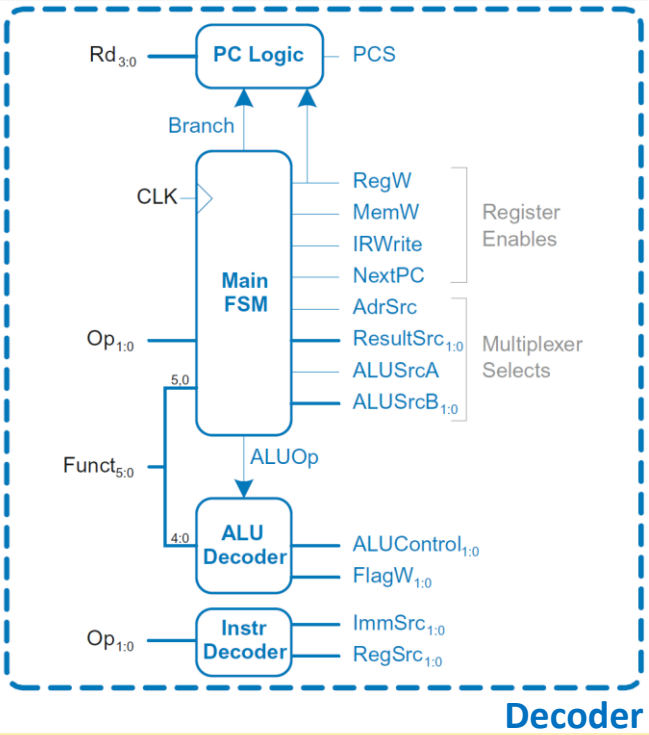


Multicycle Control

- **Two main parts:**
  1. **Decoder**
  2. **Conditional Logic**

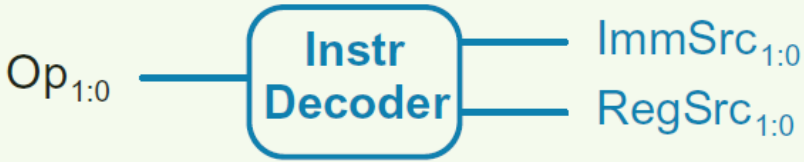


Decoder



ALU Decoder and PC Logic same as single-cycle

Instr Decoder



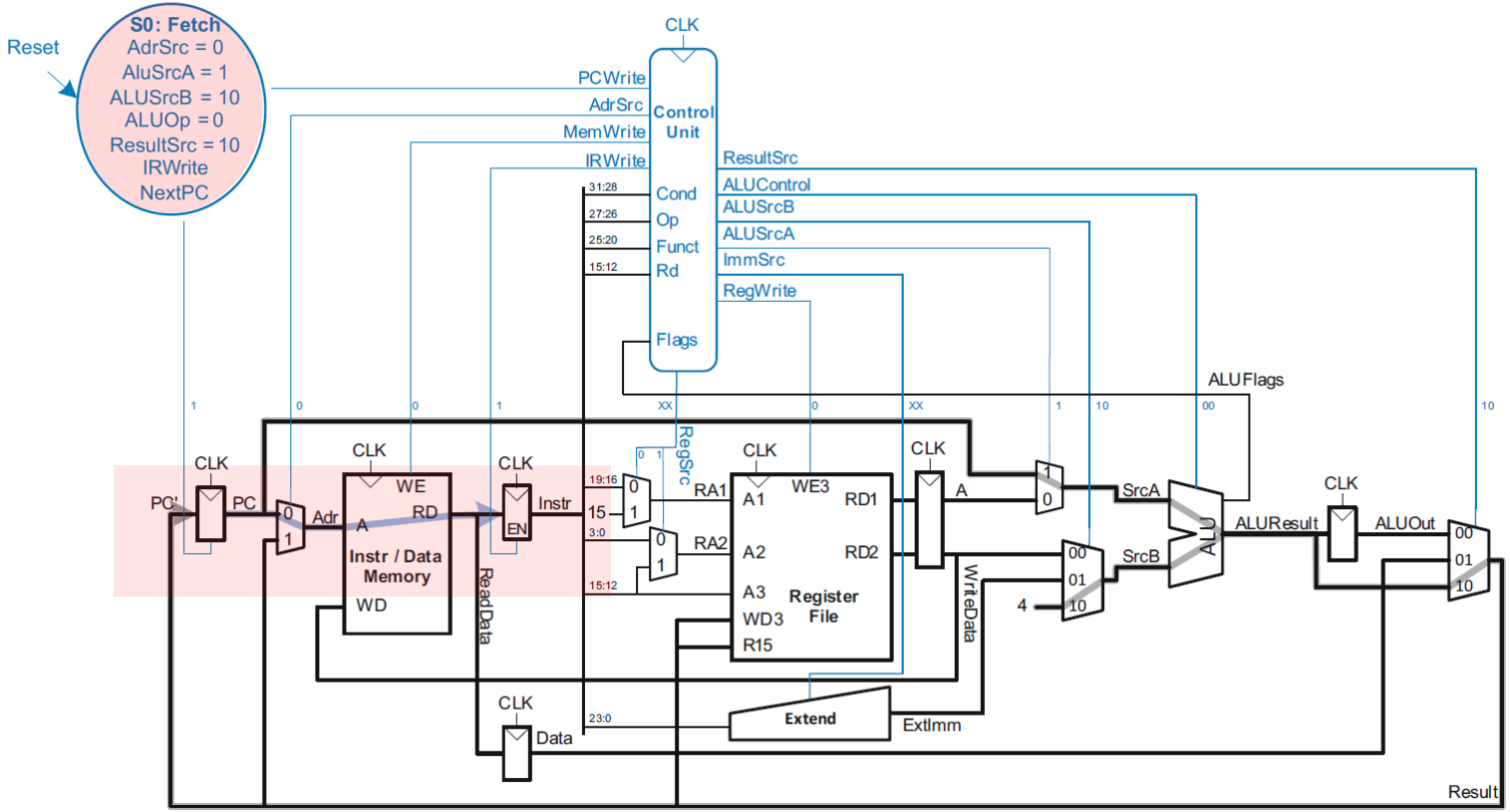
$RegSrc_0 = (Op == 10_2)$

$RegSrc_1 = (Op == 01_2)$

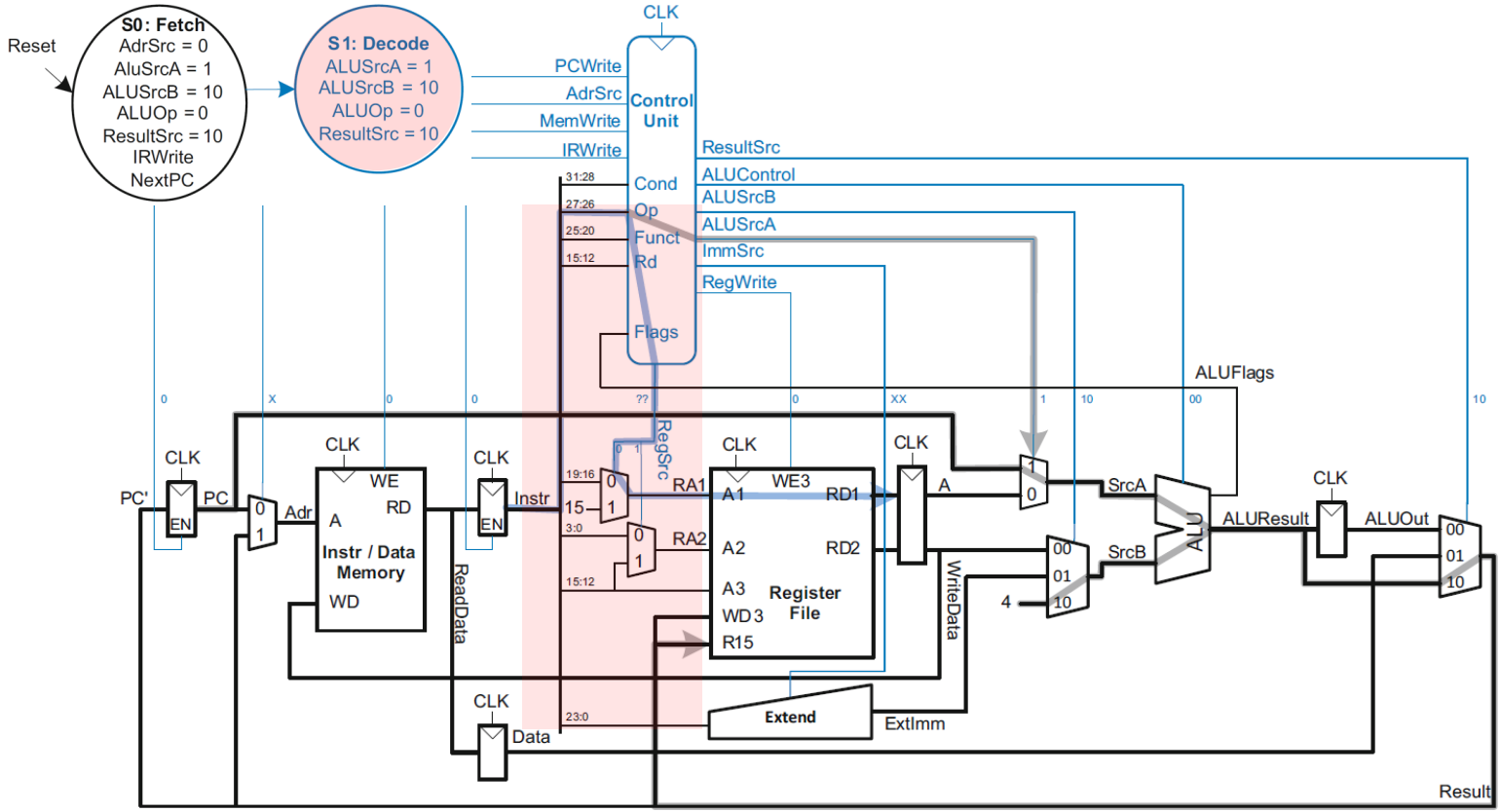
$ImmSrc_{1:0} = Op$

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

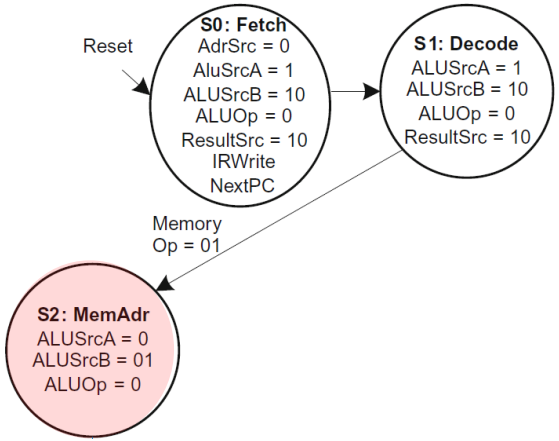
Main Controller FSM: Fetch

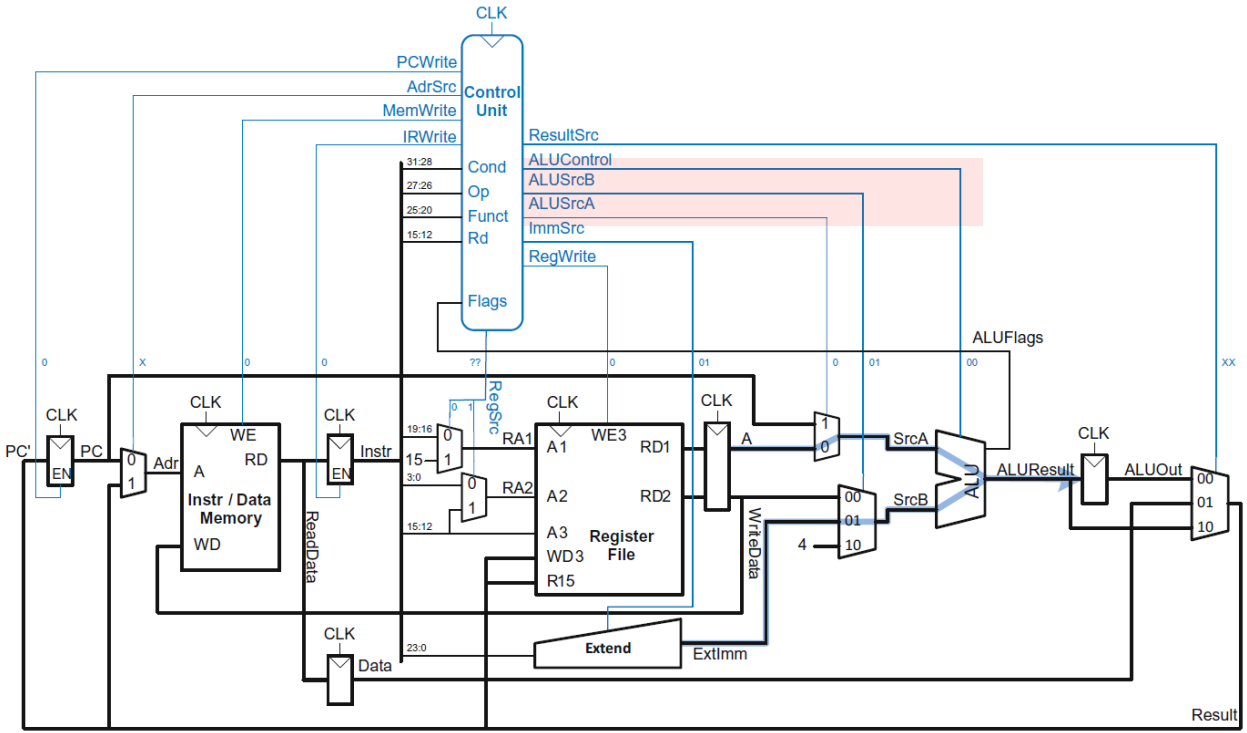


FSM: Decode

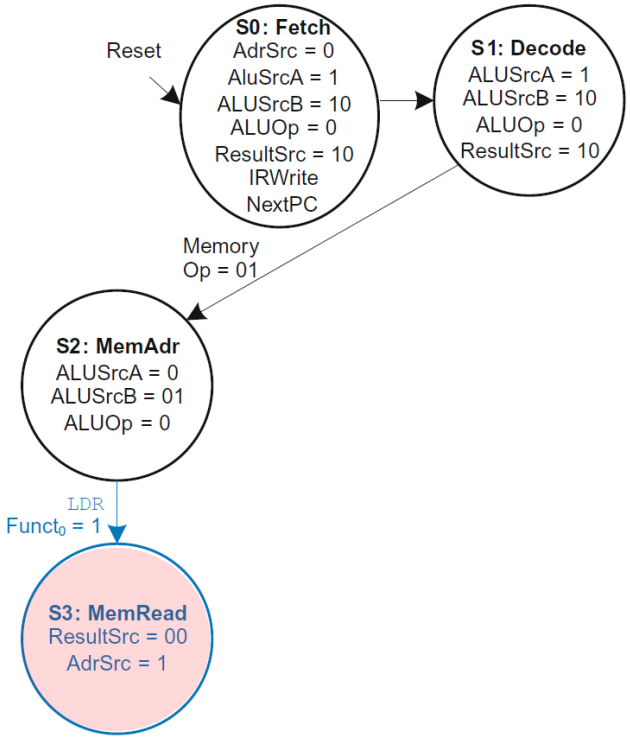

Reset
S0: Fetch
AdrSrc = 0
AluSrcA = 1
ALUSrcB = 10
ALUOp = 0
ResultSrc = 10
IRWrite
NextPC
S1: Decode
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 0
ResultSrc = 10
CLK
Control Unit
PCWrite
AdrSrc
MemWrite
IRWrite
Cond
Op
Funct
Rd
Flags
ResultSrc
ALUControl
ALUSrcB
ALUSrcA
ImmSrc
RegWrite
ALUFlags
31:28
27:26
25:20
15:12
PC'
PC
EN
Adr
WE
RD
A
Instr / Data Memory
WD
ReadData
Instr
19:16
15
3:0
15:12
RegSrc
RA1
RA2
A1
A2
A3
WD3
R15
WE3
RD1
RD2
Register File
A
SrcA
SrcB
ALU
ALUResult
ALUOut
WriteData
4
Data
Extend
ExtImm
23:0
Result

FSM: Address

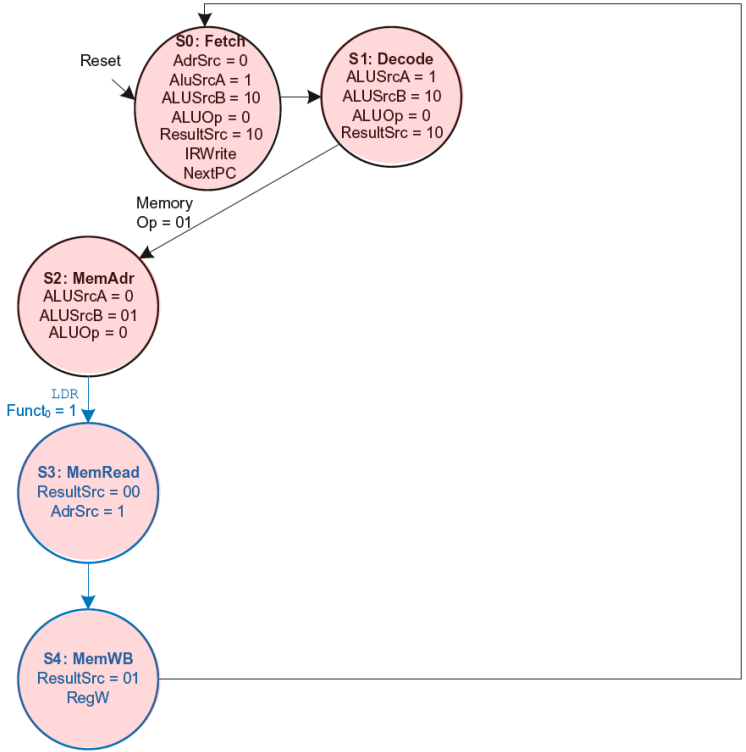

Reset
S0: Fetch
AdrSrc = 0
AluSrcA = 1
ALUSrcB = 10
ALUOp = 0
ResultSrc = 10
IRWrite
NextPC
S1: Decode
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 0
ResultSrc = 10
Memory
Op = 01
S2: MemAdr
ALUSrcA = 0
ALUSrcB = 01
ALUOp = 0


CLK
PCWrite
AdrSrc
MemWrite
IRWrite
Control Unit
Cond
Op
Funct
Rd
Flags
ResultSrc
ALUControl
ALUSrcB
ALUSrcA
ImmSrc
RegWrite
ALUFlags
PC'
PC
Adr
Instr / Data Memory
WE
RD
A
WD
ReadData
Instr
RegSrc
RA1
RA2
Register File
A1
A2
A3
WD3
R15
WE3
RD1
RD2
WriteData
A
SrcA
SrcB
ALU
ALUResult
ALUOut
Extend
ExtImm
Data
Result

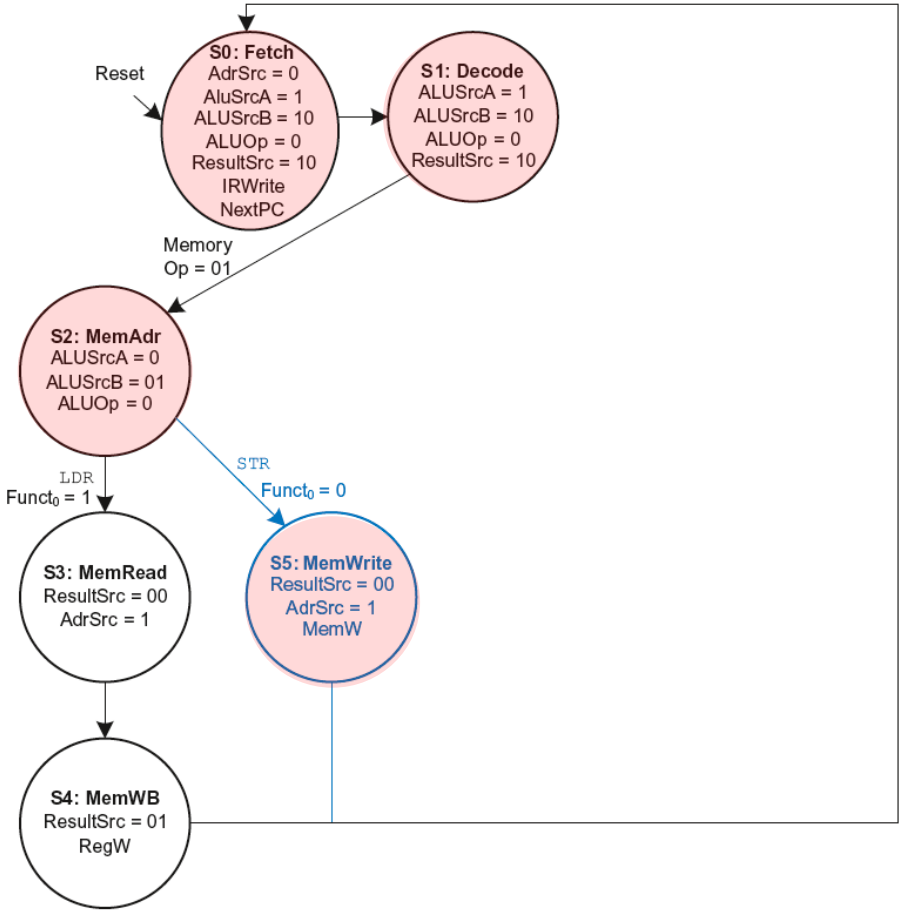
FSM: Read Memory



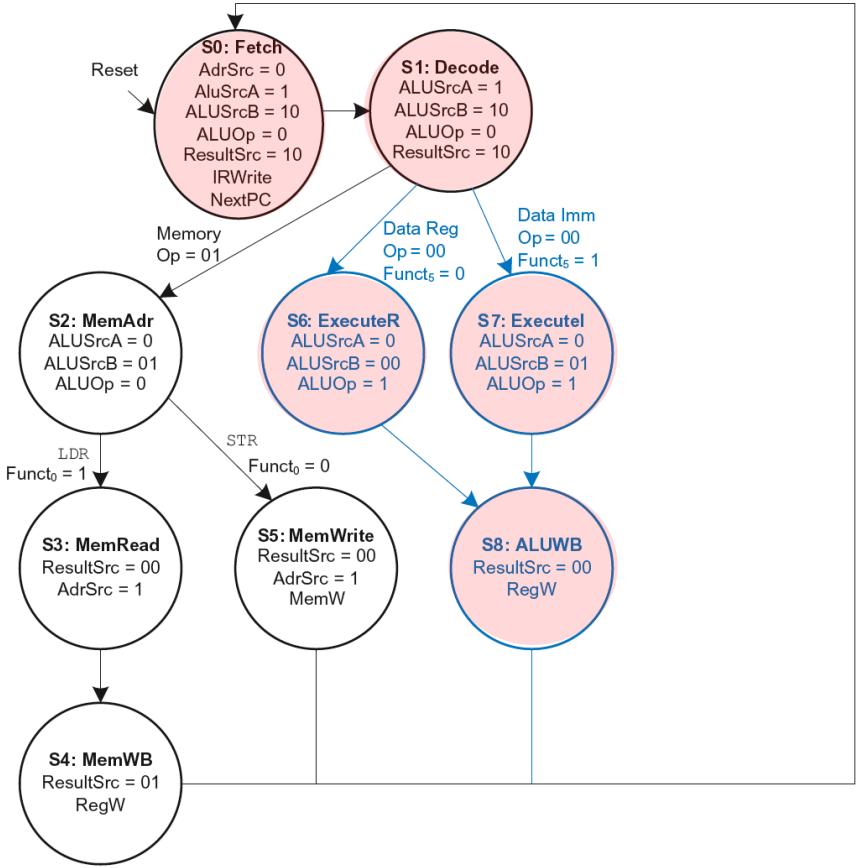
Example FSM: LDR



Example FSM: STR

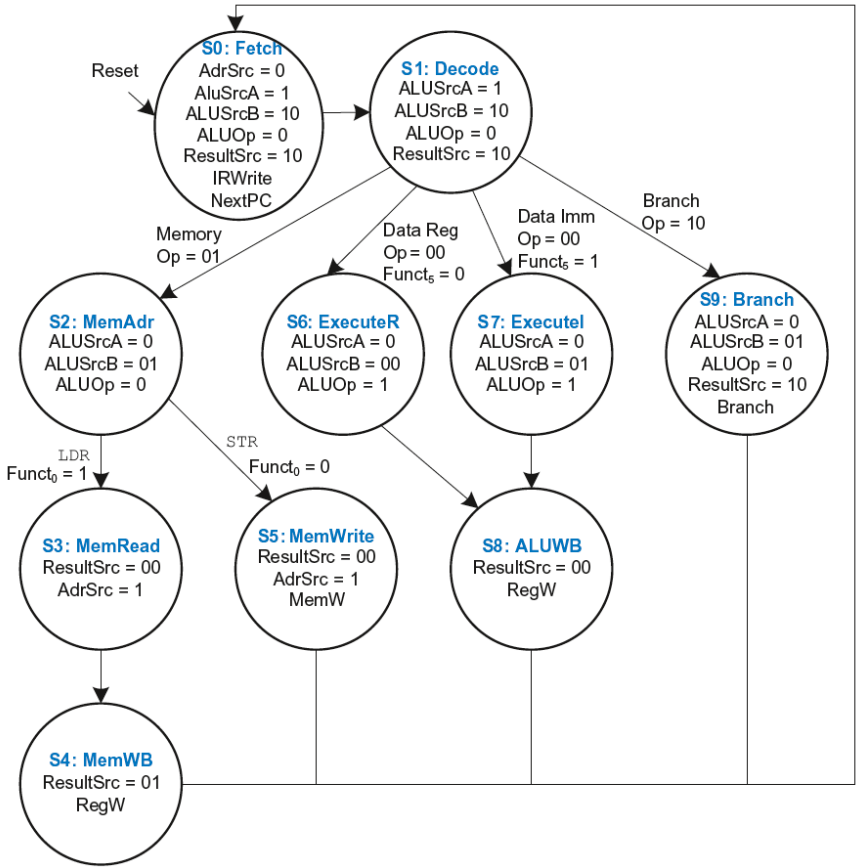


Example FSM: Data-processing



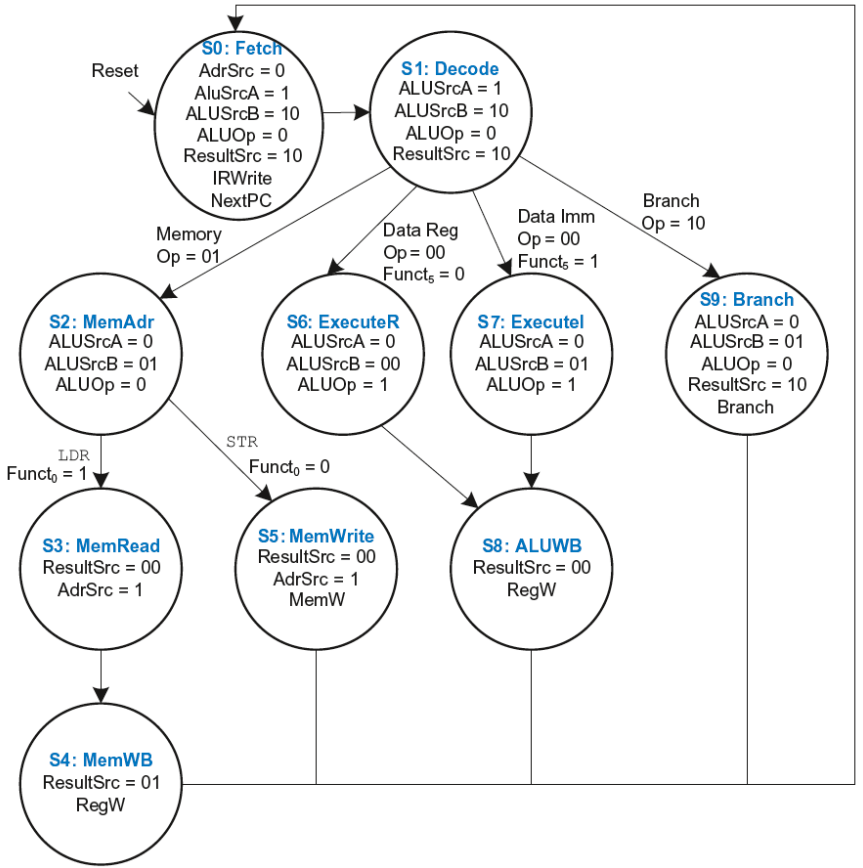
Multicycle Controller FSM

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



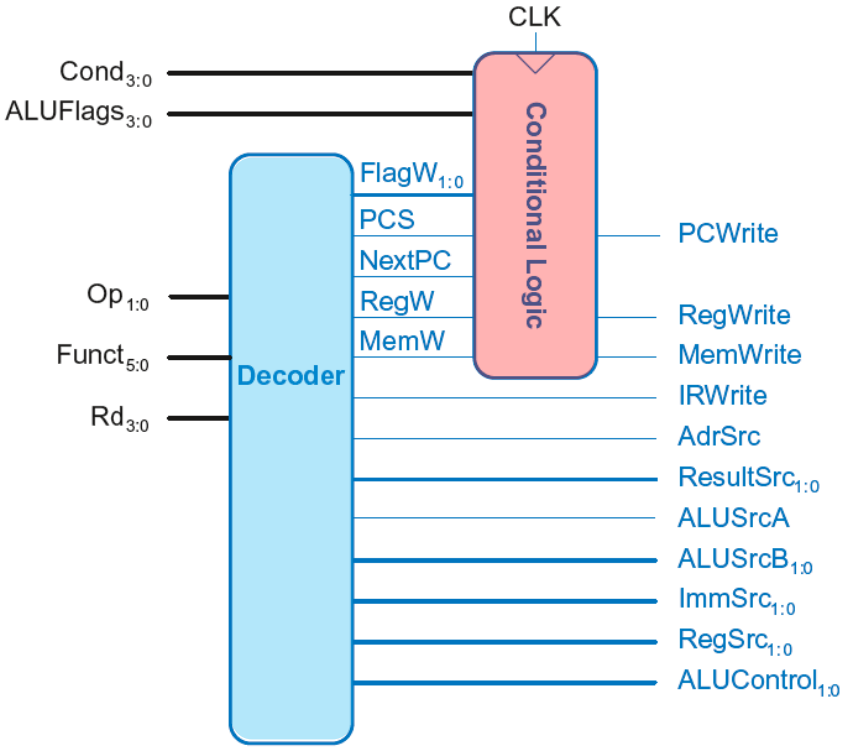
Important

What happens with signals in the next state?
Should they change?
Why?

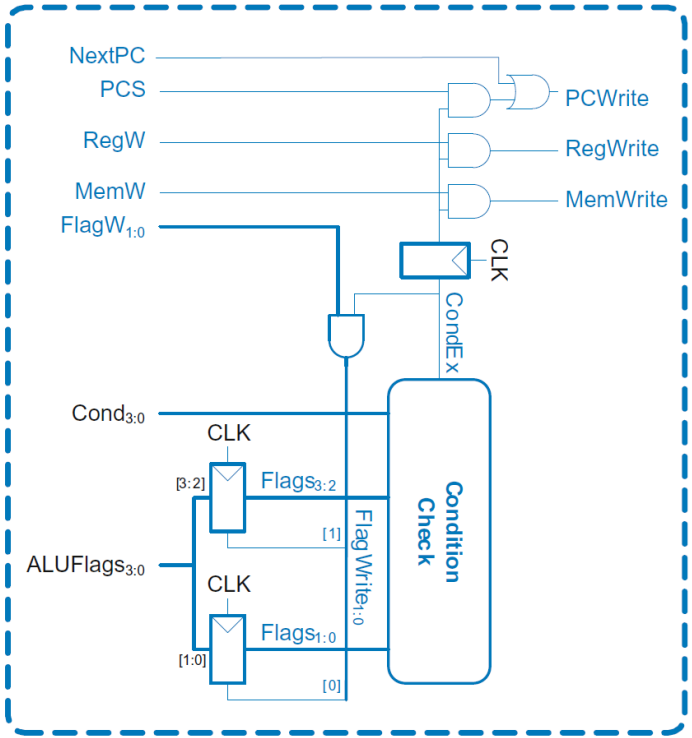


Multicycle Control

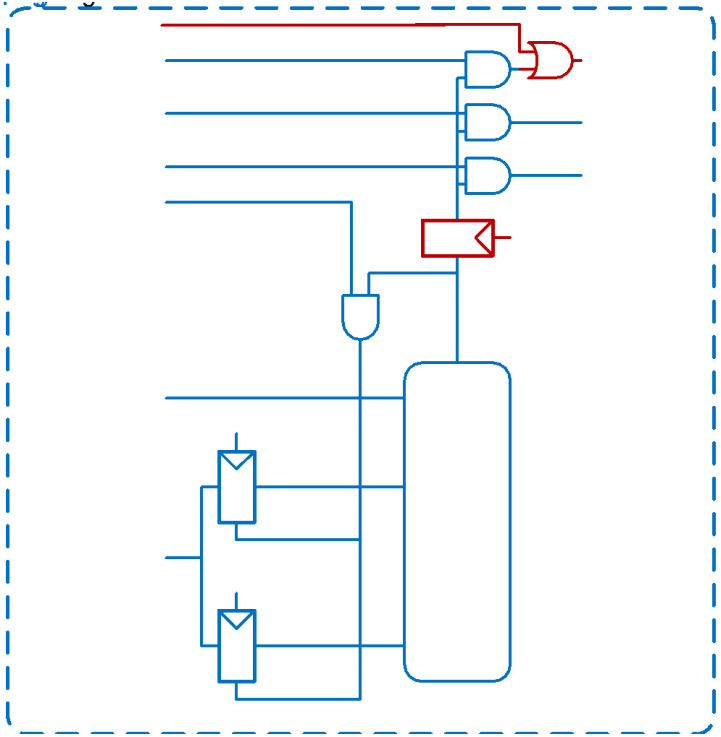
- **Two main parts:**
  1. **Decoder**
  2. **Conditional Logic**



Single-Cycle Conditional Logic



Multicycle Conditional Logic



- **PCWrite** asserted in Fetch state
- **Executel/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)

Outline

Introduction

Single Cycle Performance

Multicycle Processor

Multicycle performance

Conclusions

Multicycle Processor Performance

- **Instructions take different number of cycles. CPI is weighted average.**
- **Multicycle critical path.** Assumptions:
  - **RF is faster than memory**
  - **Writing memory is faster than reading memory**

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

Example: Multicycle Processor Performance

- **Processor instruction characteristics:**
  - B: 3 cycles.
  - DP, STR: 4 cycles.
  - LDR: 5 cycles.
- **Program characteristics (SPECINT2000 benchmark):**
  - **25%** loads
  - **10%** stores
  - **13%** branches
  - **52%** R-type

Average CPI = (**0.13**)(3) + (**0.52 + 0.10**)(4) + (**0.25**)(5) = **4.12**

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

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

Recall: 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

LDR critical path:

$$T_{c1} = t_{pcq\_PC} + 2t_{mem} + t_{dec} + t_{RFread} + t_{ALU} + 2t_{mux} + t_{RFsetup}$$
$$= [50 + 2(200) + 70 + 100 + 120 + 2(25) + 60]\ ps$$
$$= \mathbf{840\ ps}$$

A 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**

other combinational logic: 0 ps

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

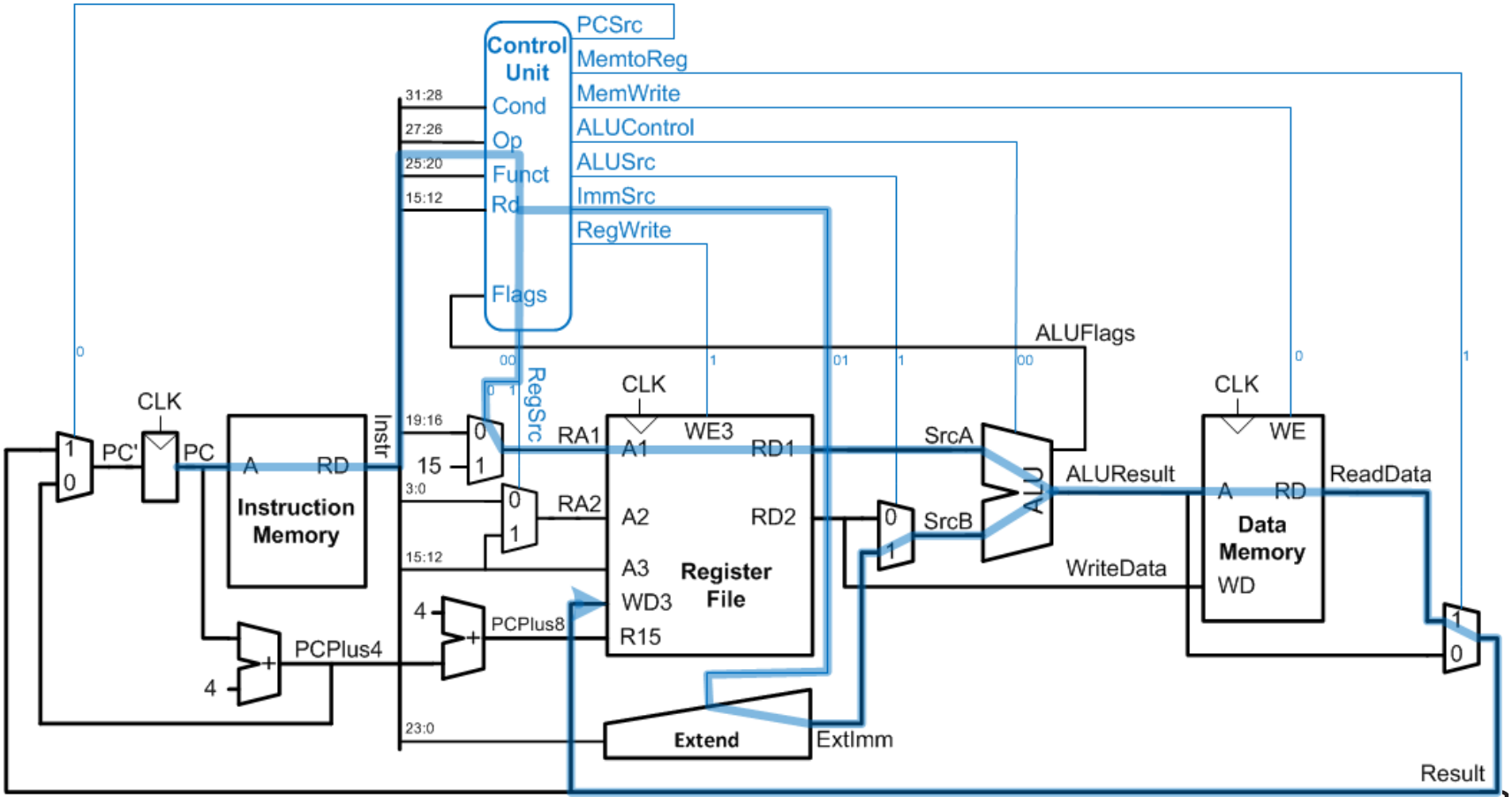
Execution Time = (# instructions) × CPI × T_c

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

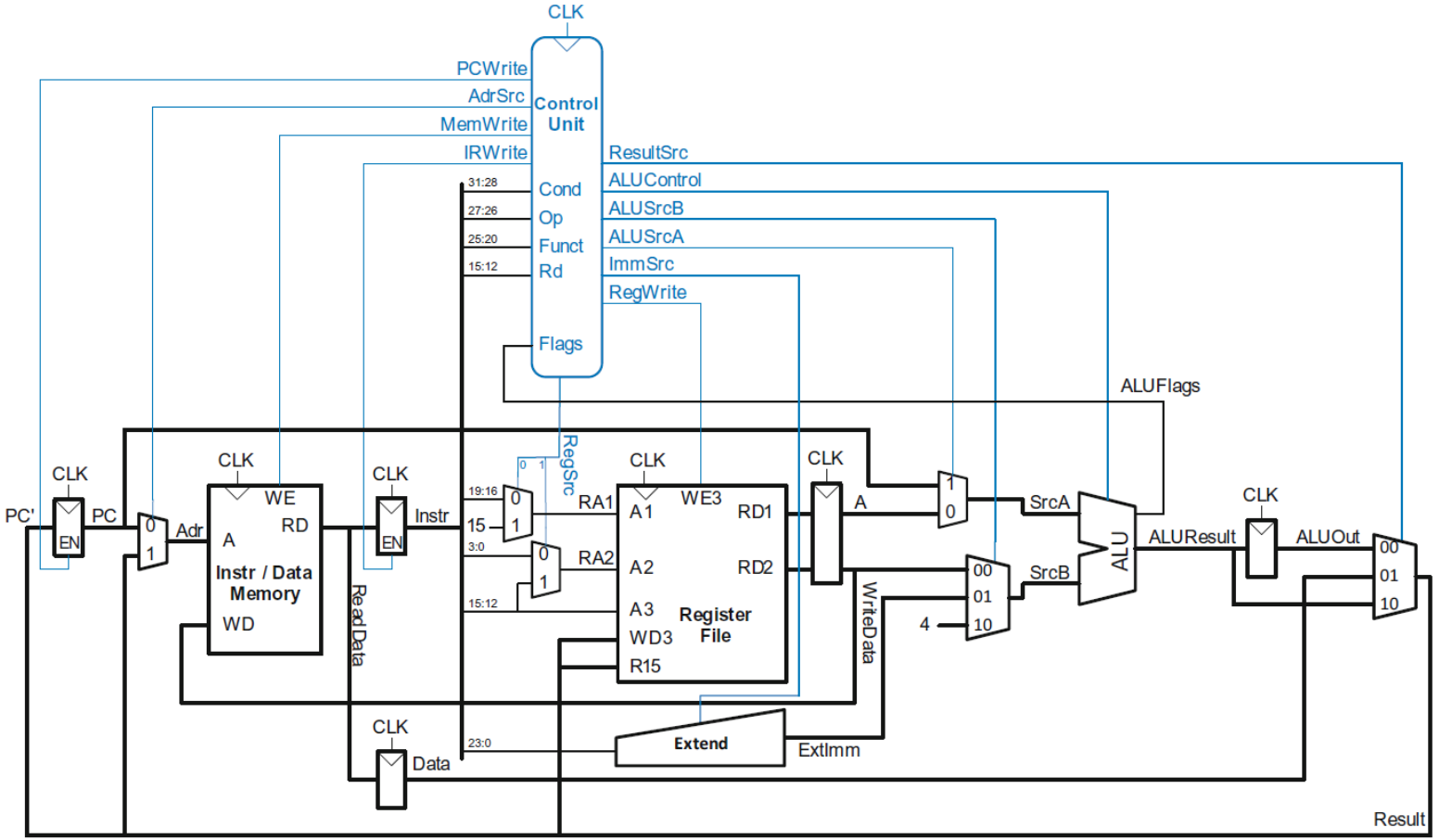
= **140 seconds**

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

Review: Single-Cycle ARM Processor



Review: Multicycle ARM Processor



Outline

Introduction

Single-cycle Performance

Multicycle Processor

Multicycle performance

Conclusions

Conclusions

- **We detailed the processor microarchitecture.**
- **We analyzed the instruction operation and interaction with the multicycle processor datapath and control units.**
- We conclude that a **processor can have different implementations of the ISA leading to different performance related to the executed program.**

Microarchitecture

Computer Architecture

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

CS3501 - 2025I
PROF.: JGONZALEZ@UTEC.EDU.PE