Computer Architecture

Microarchitecture

Debadatta Mishra, CSE, IITK

Microarchitecture

Instruction Set Architecture

ISA is finalized, How to implement?

Architect

Even before implementing… what do we mean by “implementing the ISA”?

Combinatorial and Sequential Logic

Hardware primitives

Microarchitecture

Instruction Set Architecture

Architect

Even before implementing … what do we mean by "implementing the ISA"?

Software visible state of the system (S)

Execute a given instruction

Software visible state of the system (Q)

Combinatorial and Sequential Logic

Hardware primitives

Microarchitecture

Instruction Set Architecture

- Express the state (can use hidden intermediate states)
- Define how state changes for different instructions (S → Q)
- Add logic to implement the state change

Architect

Even before implementing… what do we mean by "implementing the ISA"?

Software visible state of the system (S)

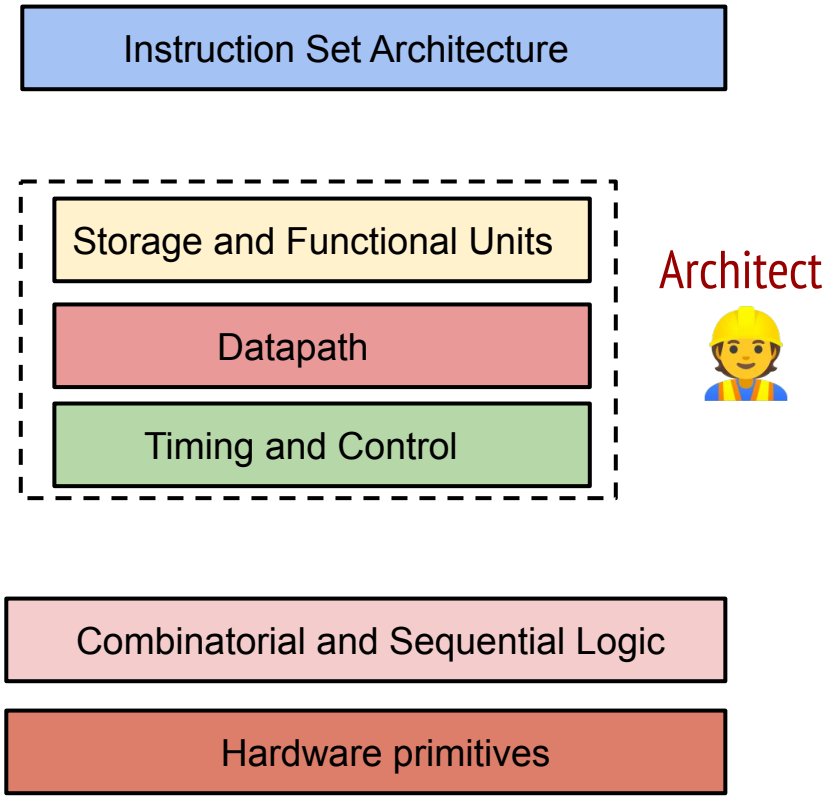
Execute a given instruction

Software visible state of the system (Q)

Combinatorial and Sequential Logic

Hardware primitives

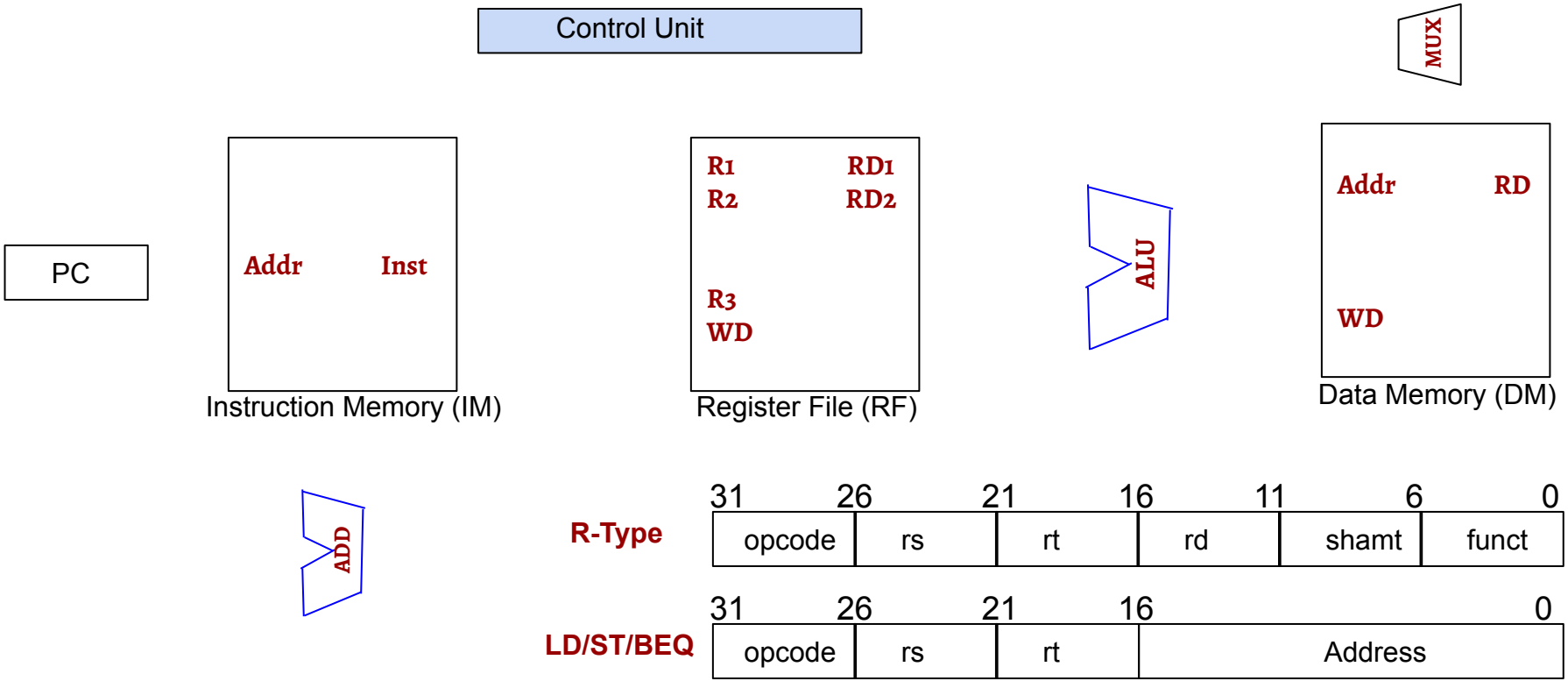
Microarchitecture



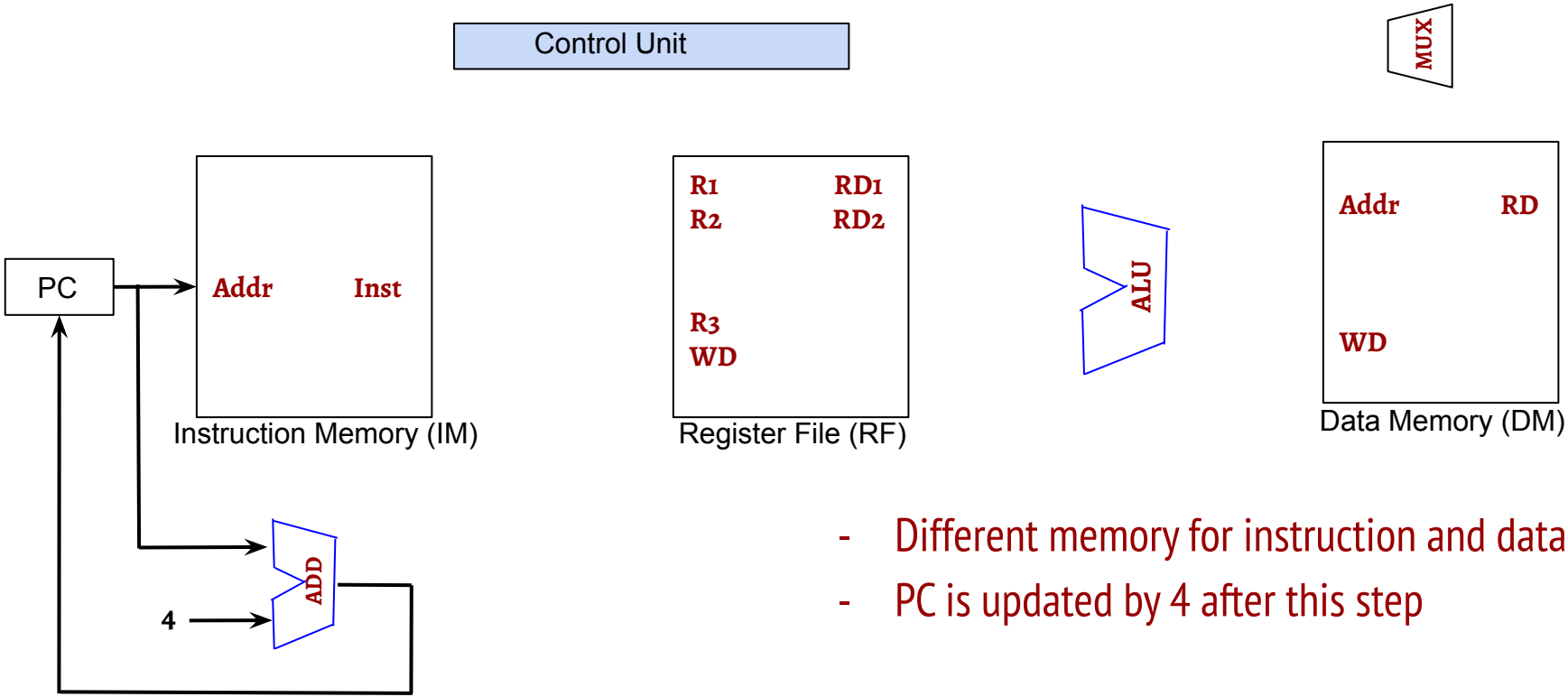
Architect

- Functional units comprise of the user visible state (and some hidden states) along with computation units (e.g, ALUs)
- Datapath defines the data flow between different functional units
- Timing and control determines
  - Which data path should be enabled and when they should be enabled?

Architectural State and Functional Units

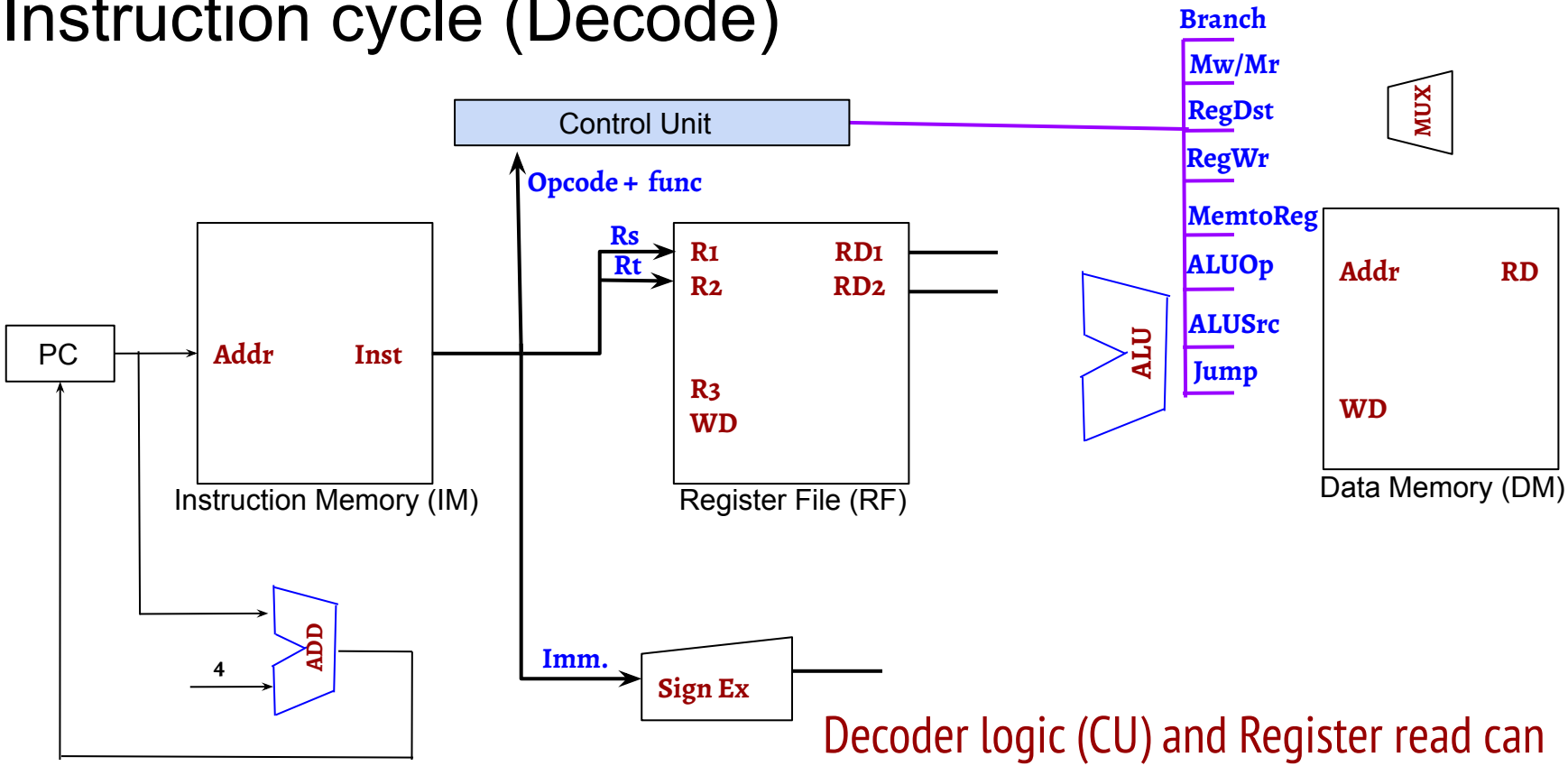


Instruction cycle (Fetch)



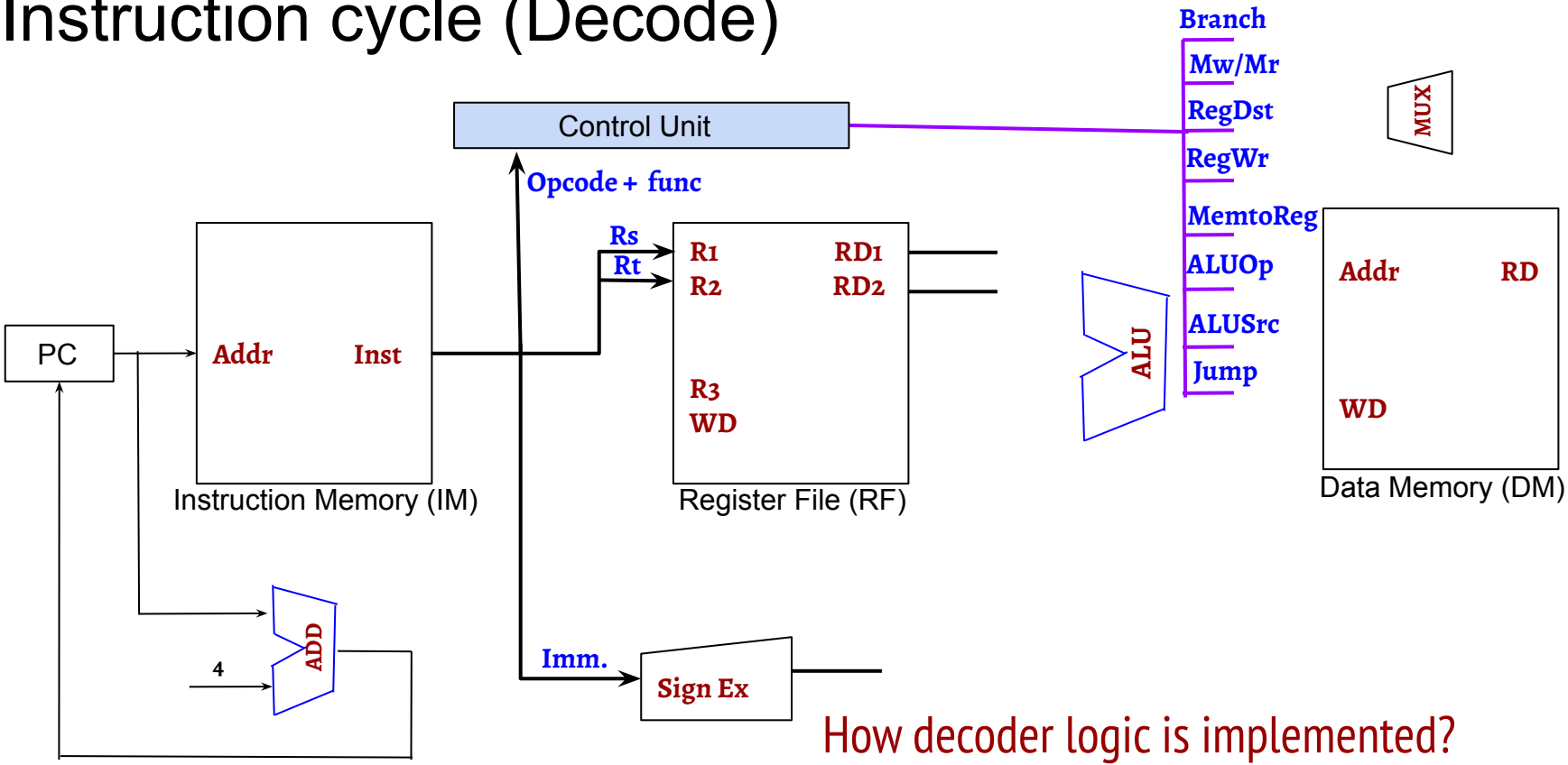
- Different memory for instruction and data
- PC is updated by 4 after this step

Instruction cycle (Decode)



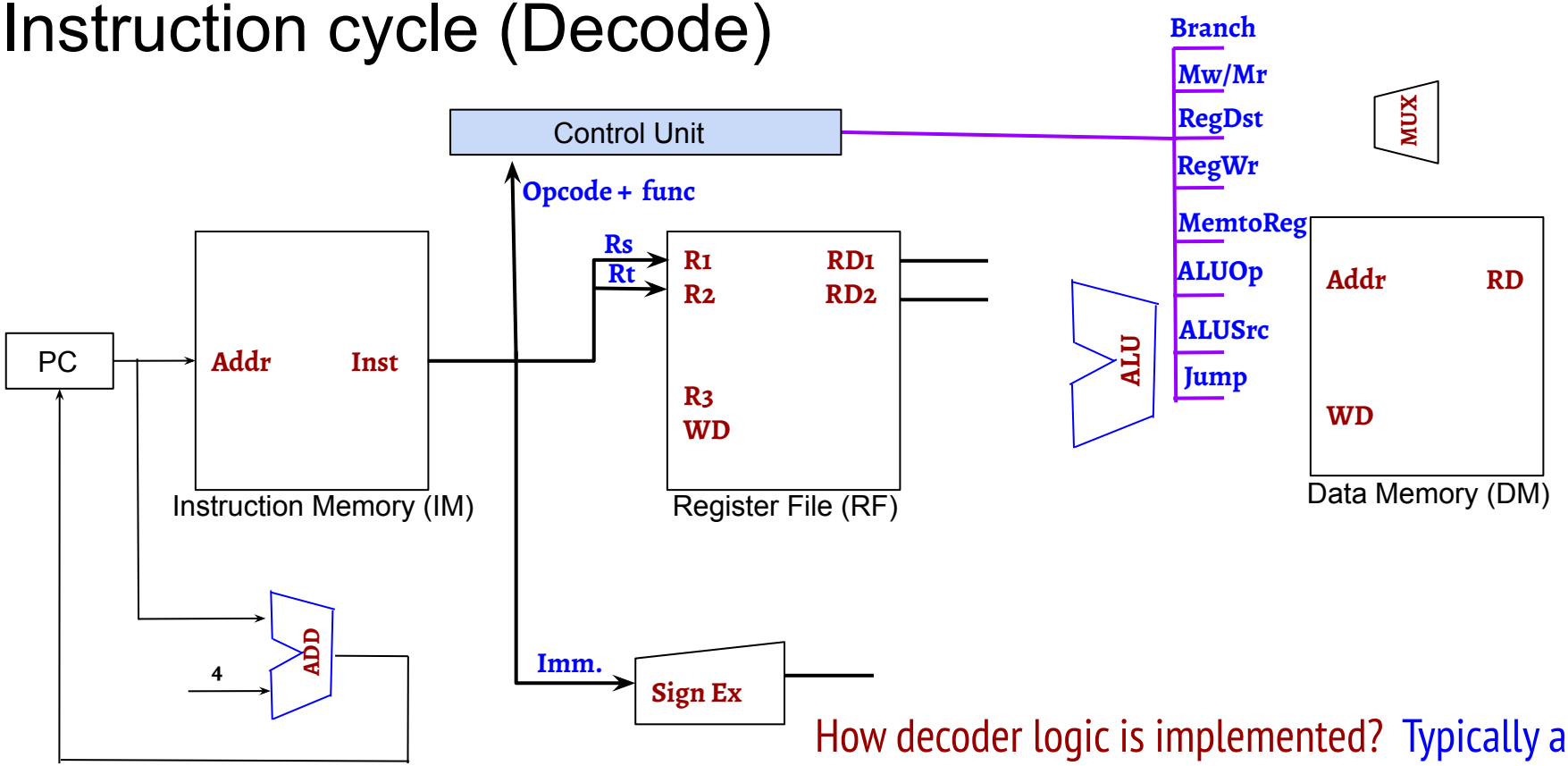
Decoder logic (CU) and Register read can happen concurrently. All control signals must be ready before the next step

Instruction cycle (Decode)



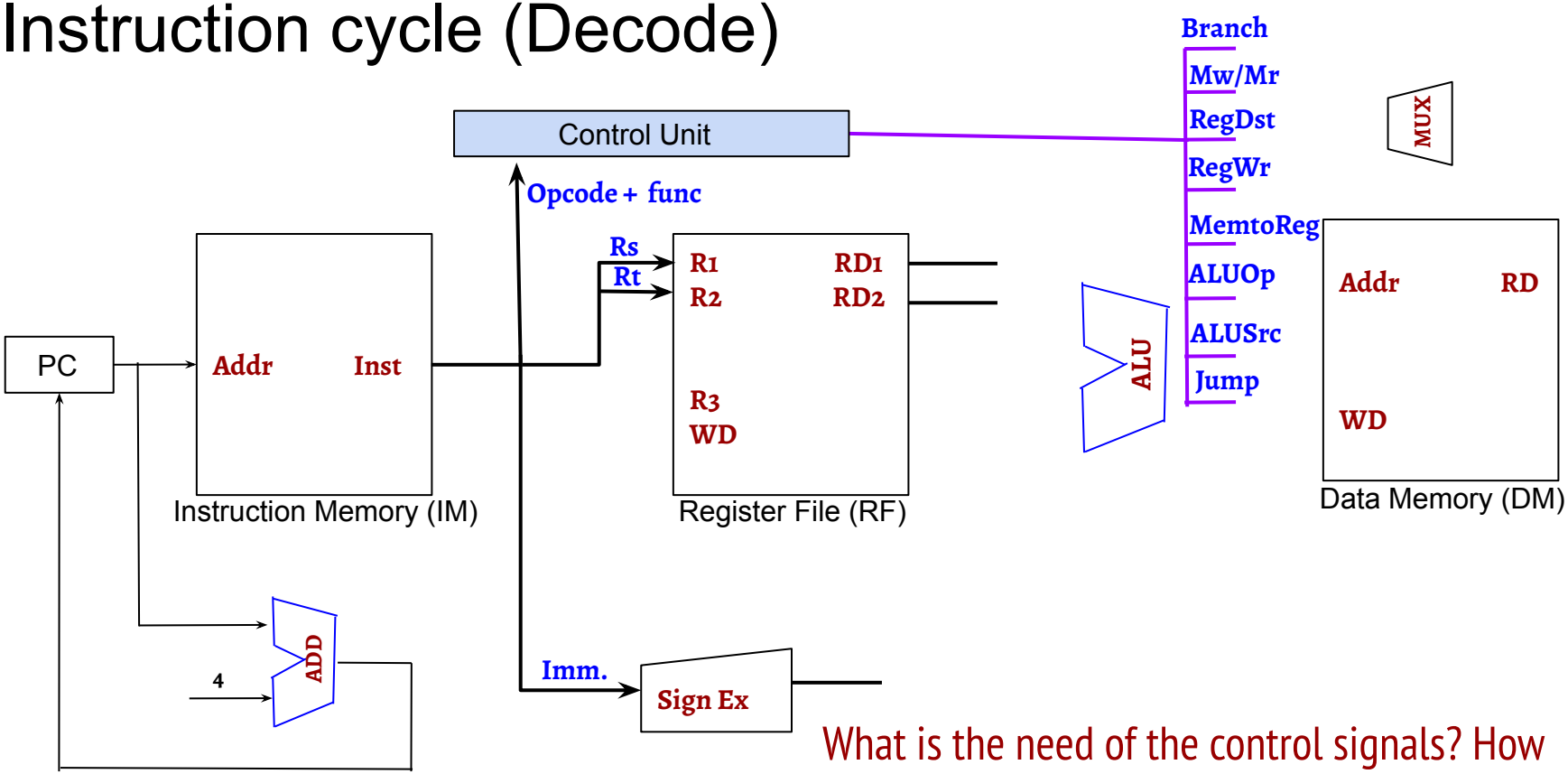
How decoder logic is implemented?

Instruction cycle (Decode)



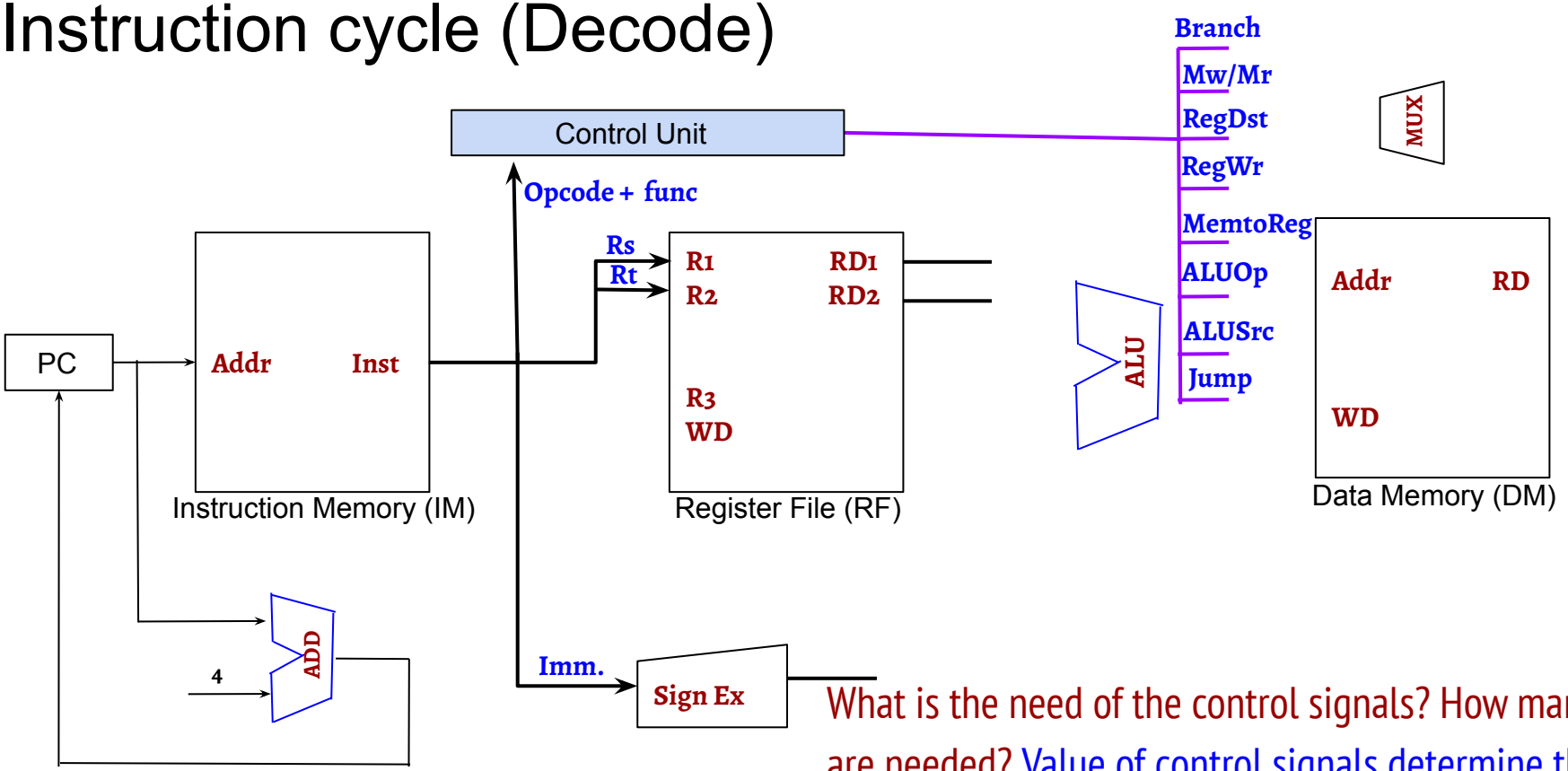
How decoder logic is implemented? Typically a combinatorial circuit mapping the opcode and func bits of the Instruction into a set of outputs

Instruction cycle (Decode)



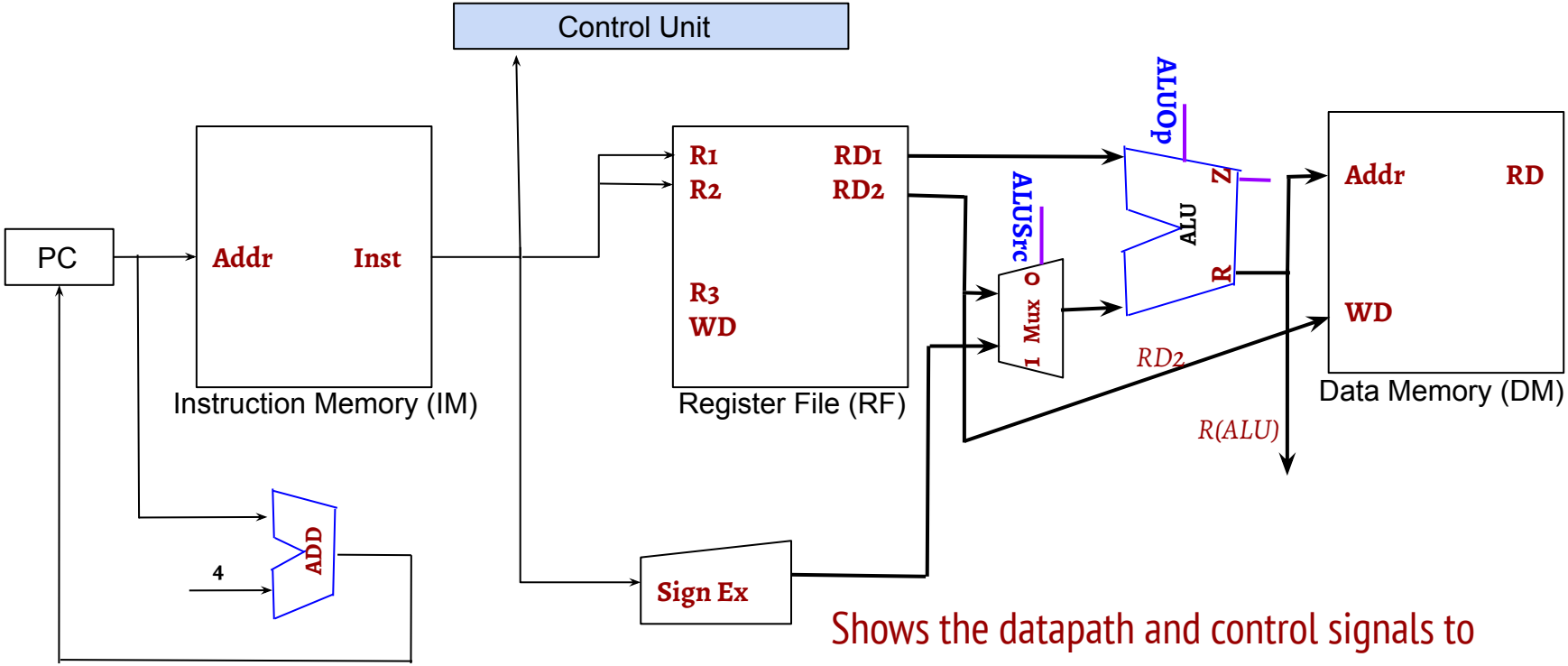
What is the need of the control signals? How many are needed?

Instruction cycle (Decode)



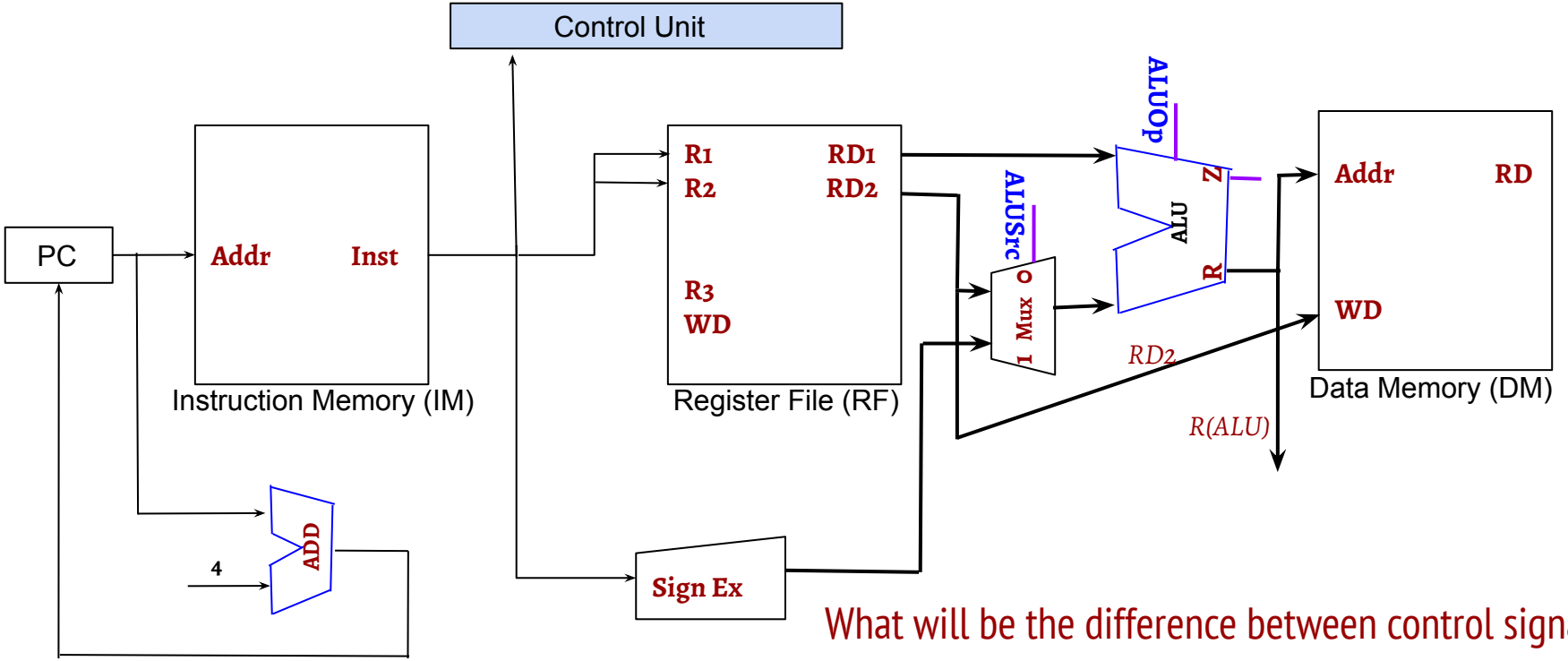
What is the need of the control signals? How many are needed? Value of control signals determine the data flow of subsequent steps. # of control signals depend on the ISA and Uarch implementation.

Instruction cycle (Execute or Address Calculate)



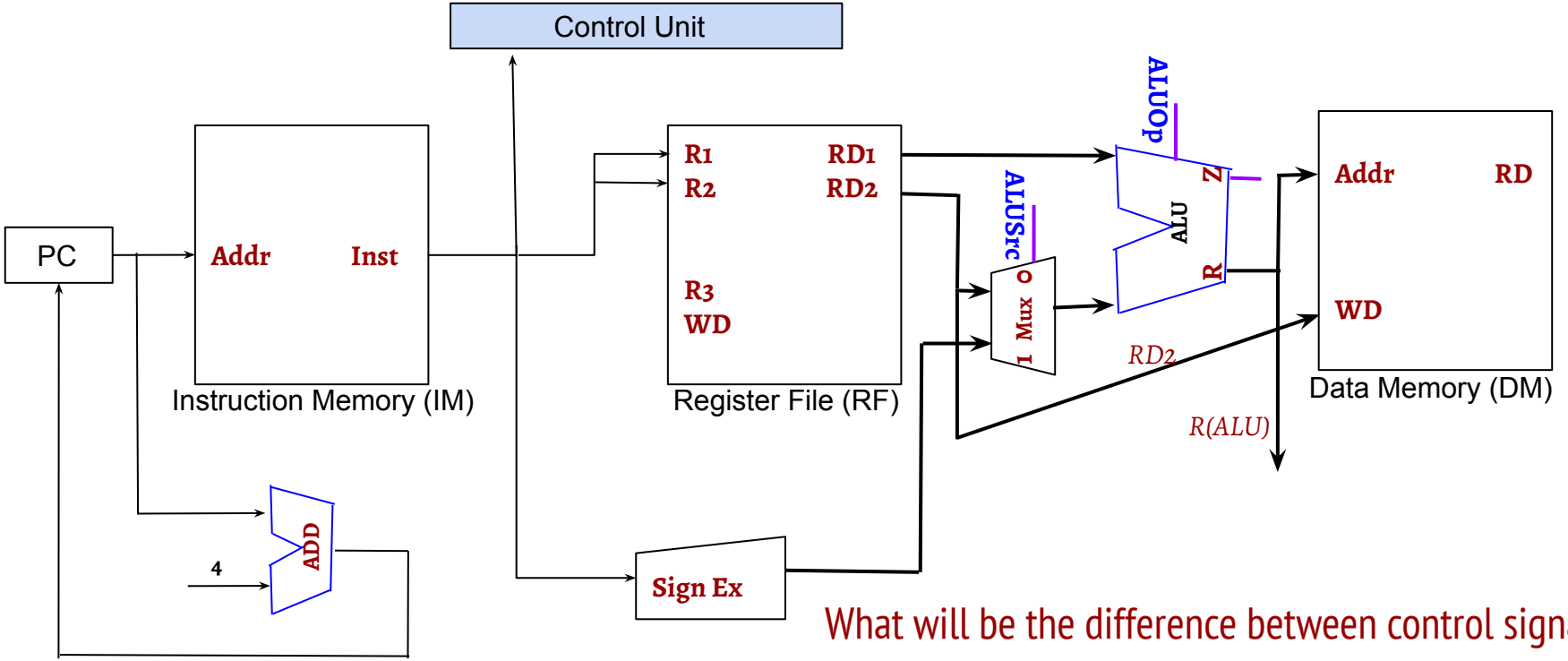
Shows the datapath and control signals to implement the *execute step* of `lw`, `sw` and register instructions (e.g., `add`, `or`, `addi`)

Instruction cycle (Execute or Address Calculate)



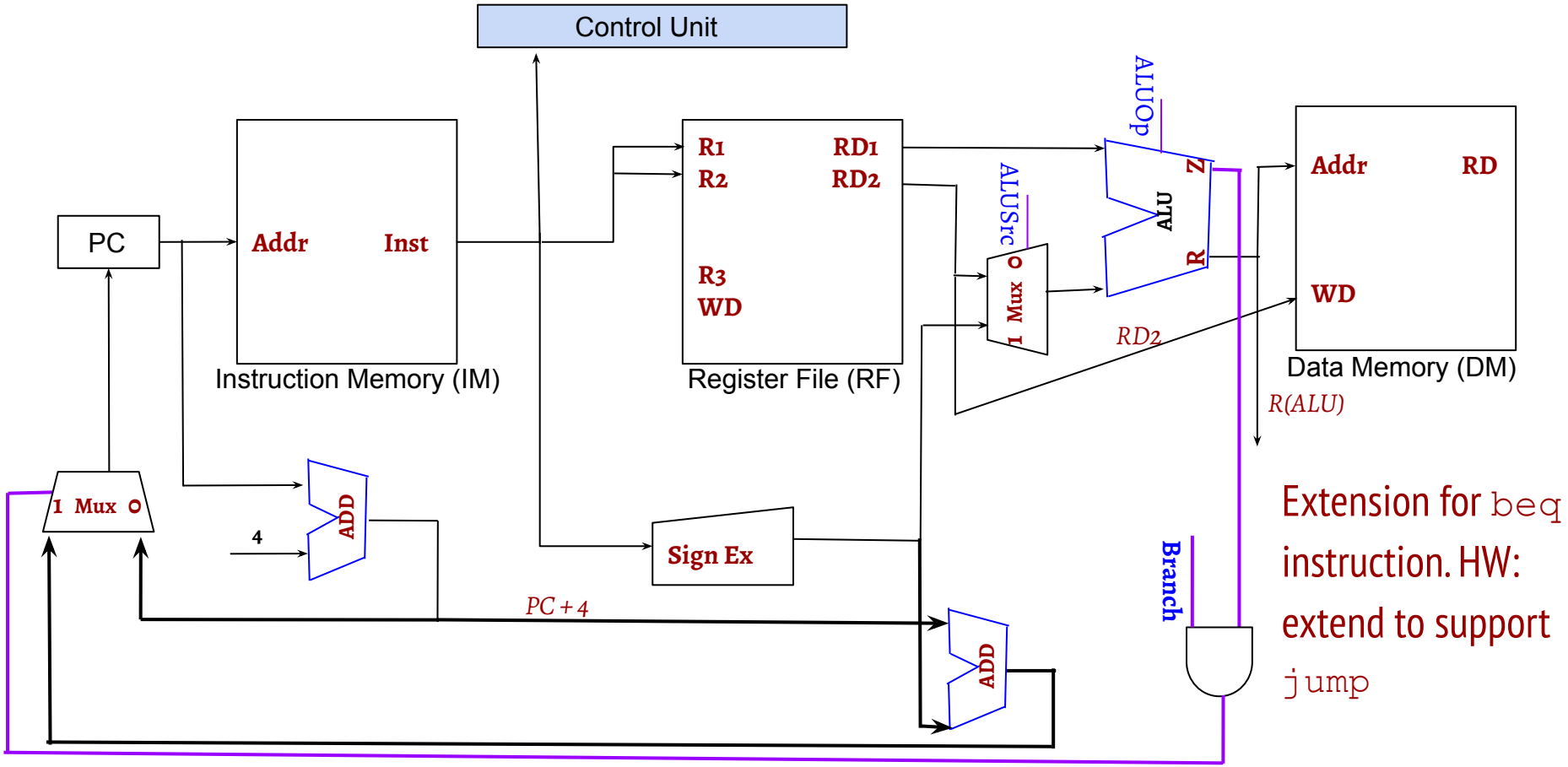
What will be the difference between control signal values between `add` and `addi` instructions?

Instruction cycle (Execute or Address Calculate)



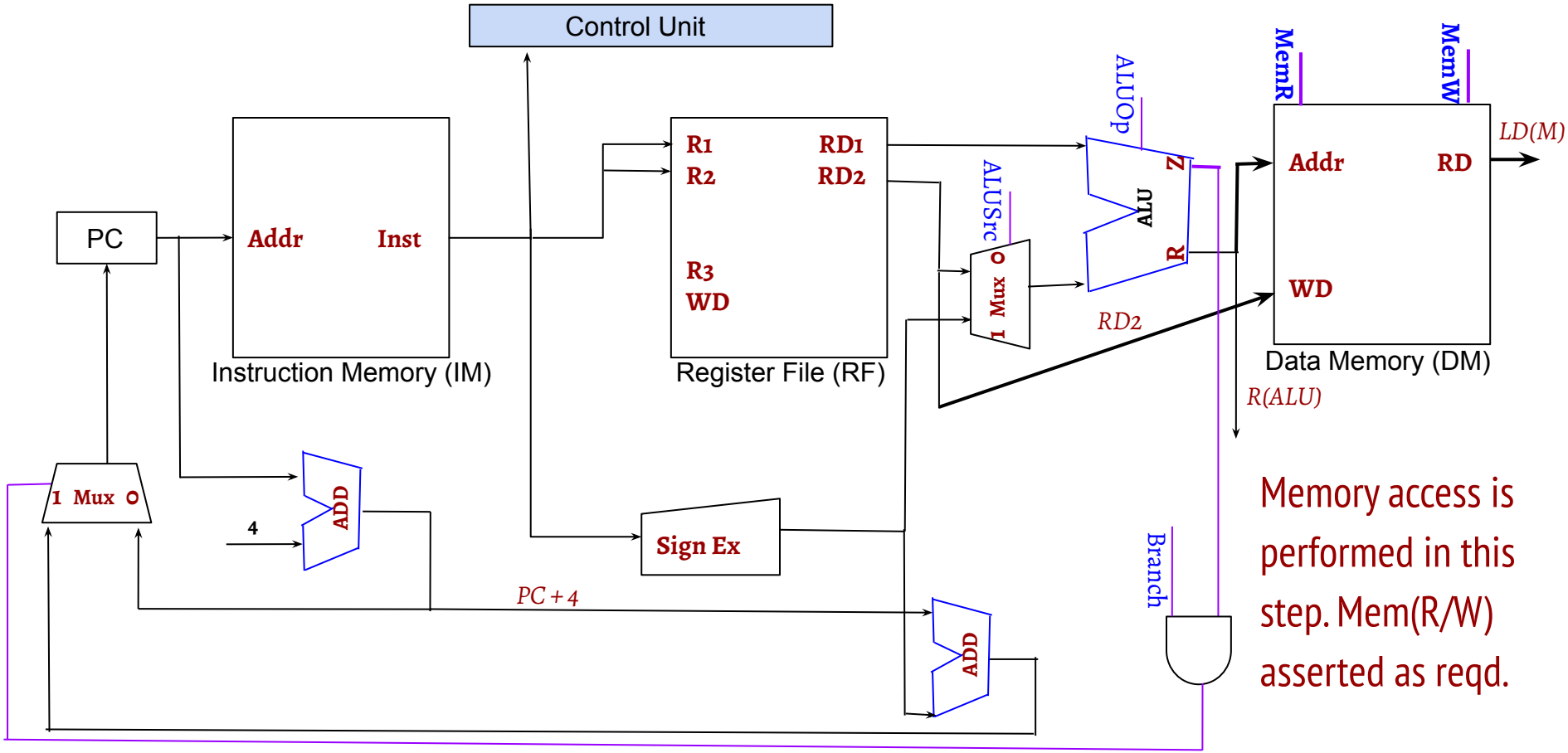
What will be the difference between control signal values between `add` and `addi` instructions? Only ALUSrc input will be different (for this step)

Instruction cycle (Execute or Address Calculate)

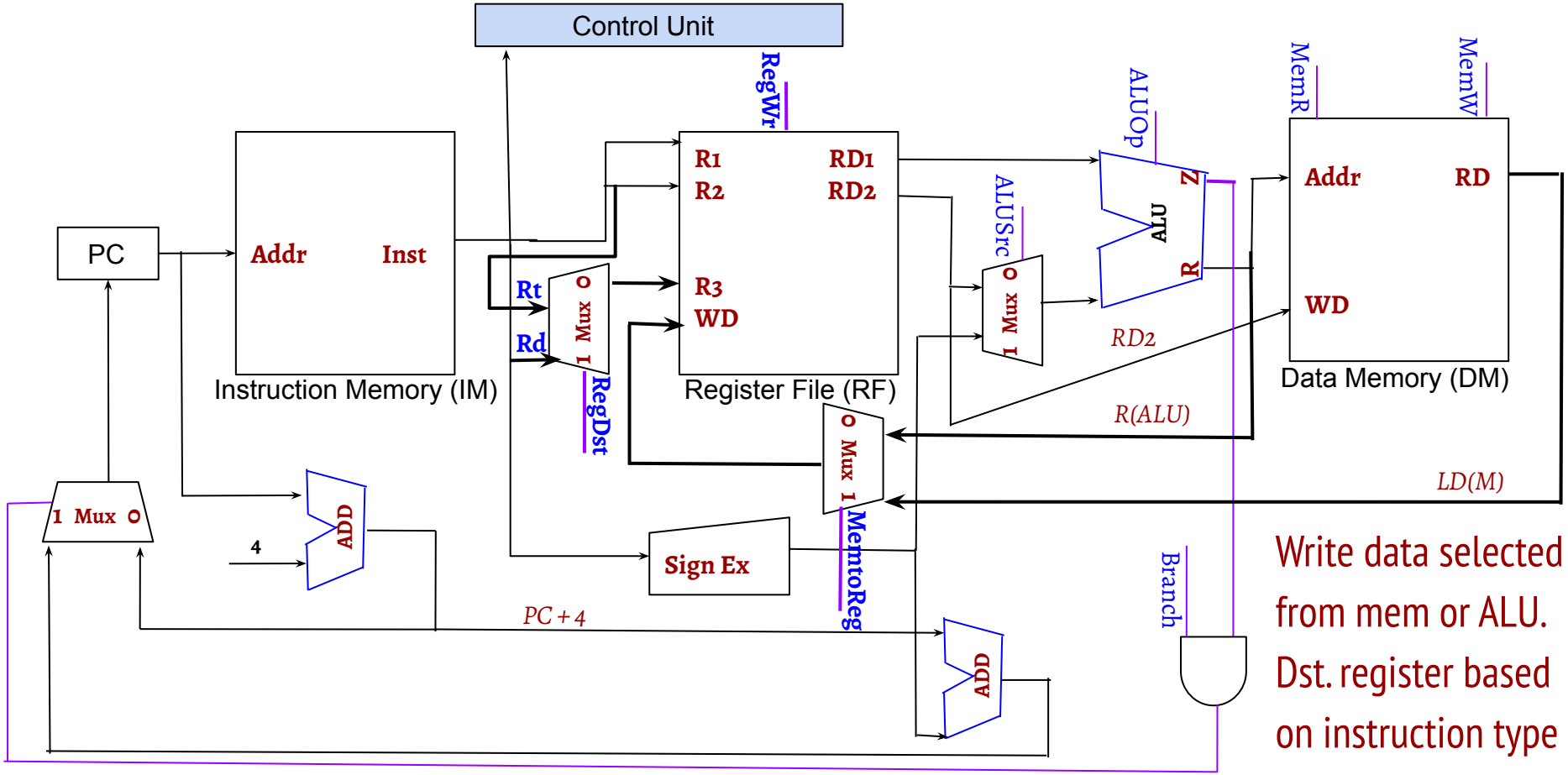


Extension for `beq` instruction. HW: extend to support `jump`

Instruction cycle (Memory Data Access)



Instruction cycle (Register Write)



Instruction Cycle

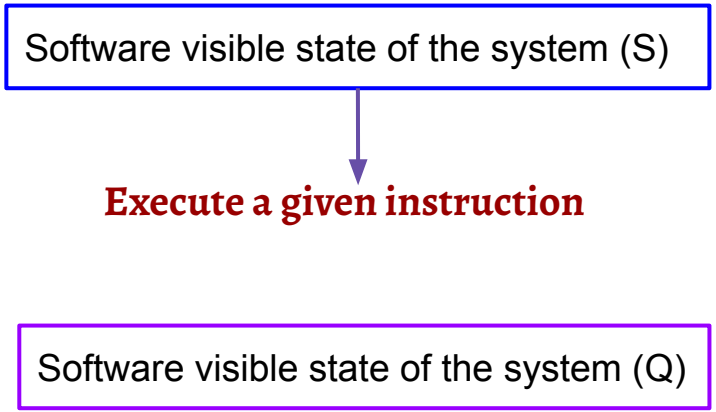
- What will be the control signal values
    - For R-type
    - For Load
    - For BEQ
- How to tackle increased complexity in decoding?

Instruction Cycle

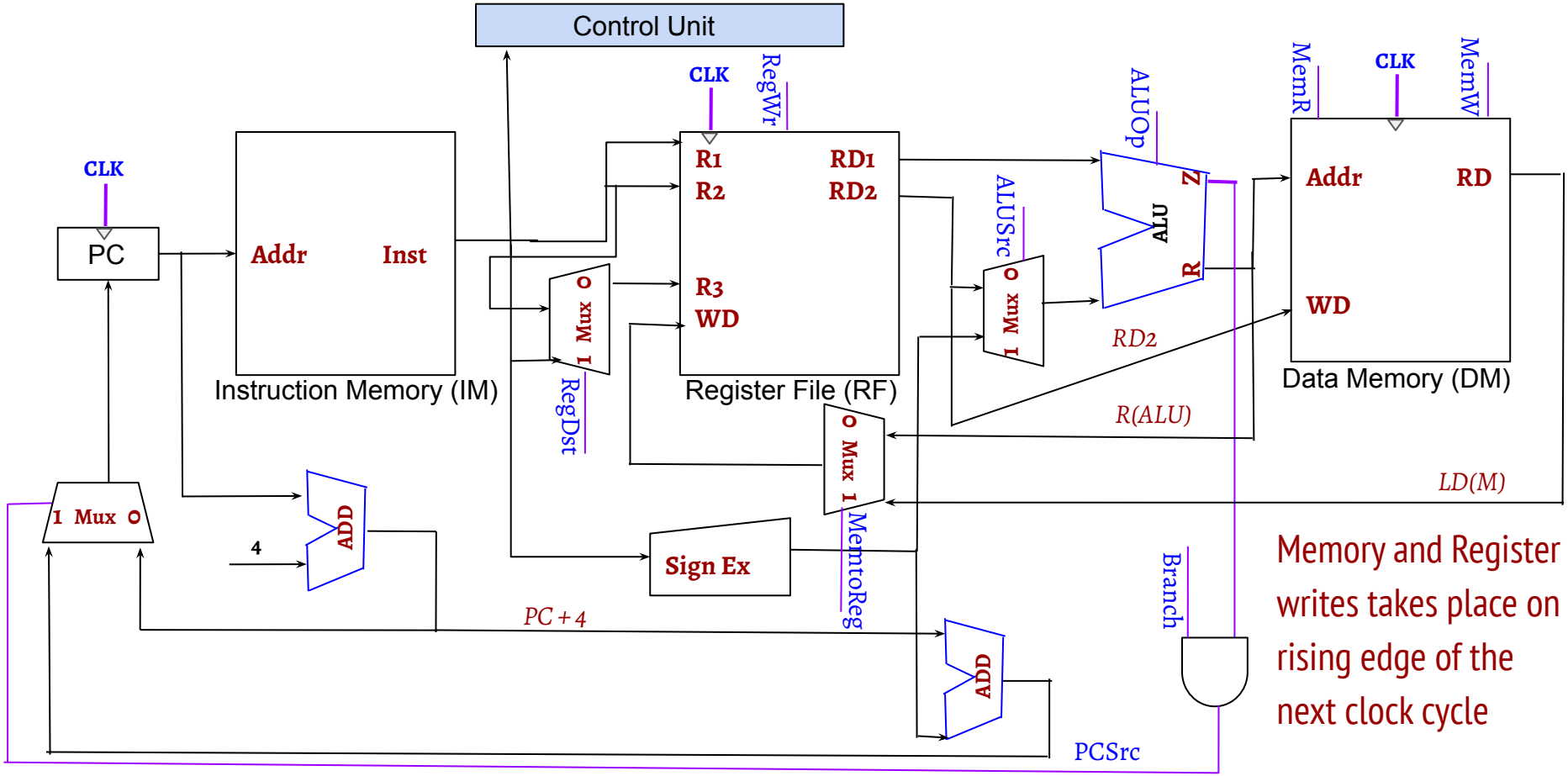
- What will be the control signal values
  - For R-type: RegDst = 1, RegWr = 1, everything is zero. ALUOp = `ALUFunc(funct)`, where `ALUFunc` is a combinatorial logic function
  - For Load: ALUSrc=1, MemR=1, MemtoReg=1, RegWr=1, everything else is zero. ALUOp = `AluFunc`(Opcode, ADD)
  - For BEQ: Branch=1, everything else is zero. ALUOp = `AluFunc`(OpCode, SUB)
- How to tackle increased complexity in decoding?
  - Decoding can be multi-staged combinatorial logic. Example: ALU operation determined by combining input from decoder and *funct* bits

Single cycle implementation

- All steps of instruction execution are performed in a single clock cycle
- Software visible state transition (S ⟶ Q) in a single clock cycle
- Advantages?
- Disadvantages?



Single cycle implementation



Single cycle implementation

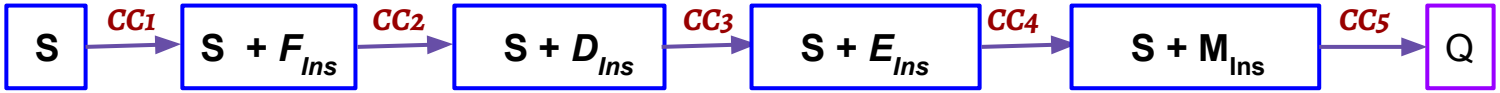
- All steps of instruction execution are performed in a single clock cycle
- Software visible state transition (S ⟶ Q) in a single clock cycle
- Advantages
    - Do not require any additional micro-architectural state
    - Simple control logic
- Disadvantages
    - Clock cycle determined by the slowest datapath (for `lw` in the example)
    - No state element can be used more than once during the instruction execution

Multi-cycle implementation

- All steps of instruction execution are performed in a multiple (**K**) clock cycles
- Software visible state transition (**S** $\longrightarrow$ **Q**) in **K** clock cycles
- Intermediate (software invisible) state between different steps

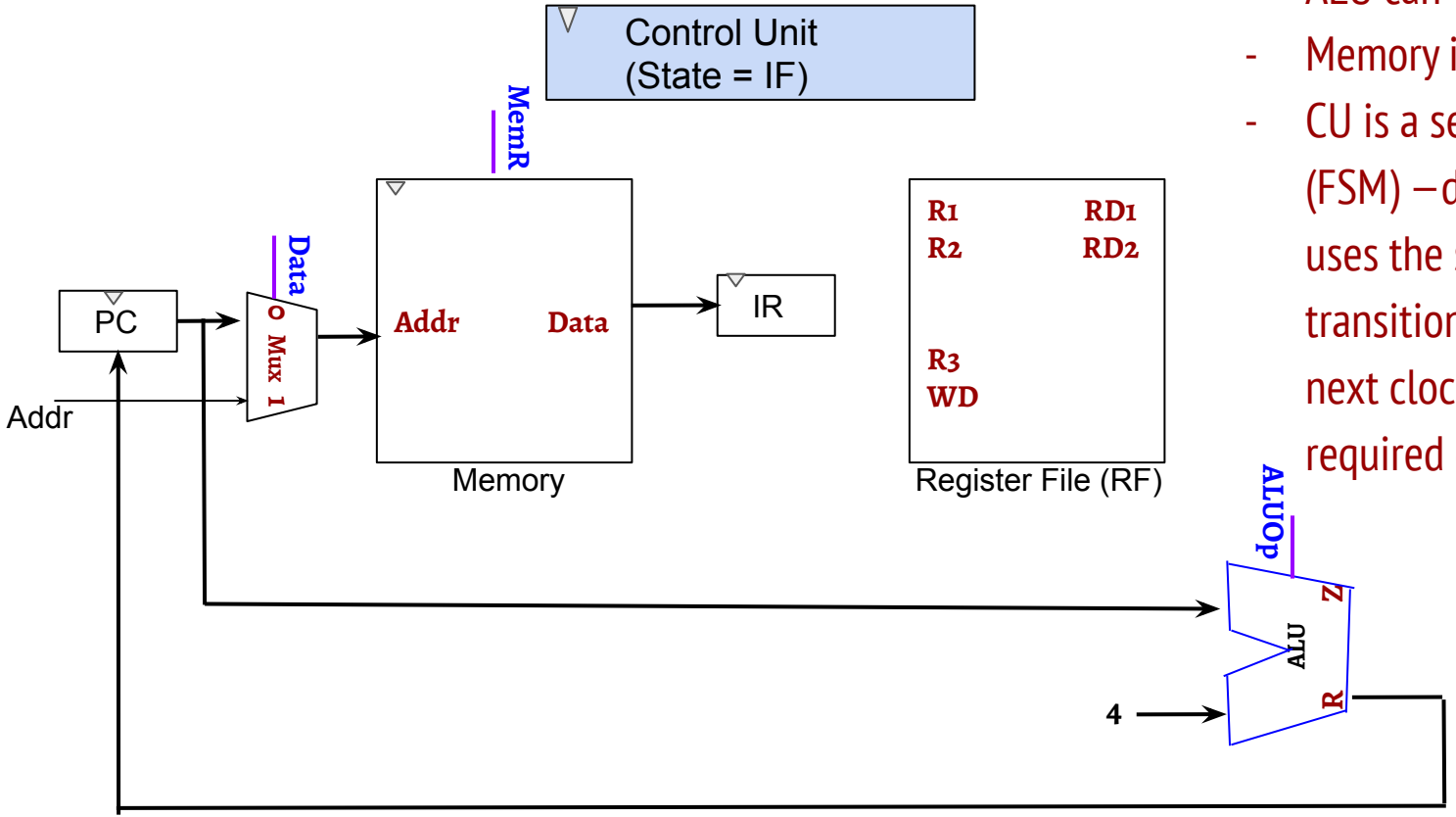
Multi-cycle implementation

- All steps of instruction execution are performed in a multiple (**K**) clock cycles
- Software visible state transition (**S** → **Q**) in **K** clock cycles
- Intermediate (software invisible) state between different steps.
- Example (with 5 steps/cycles)



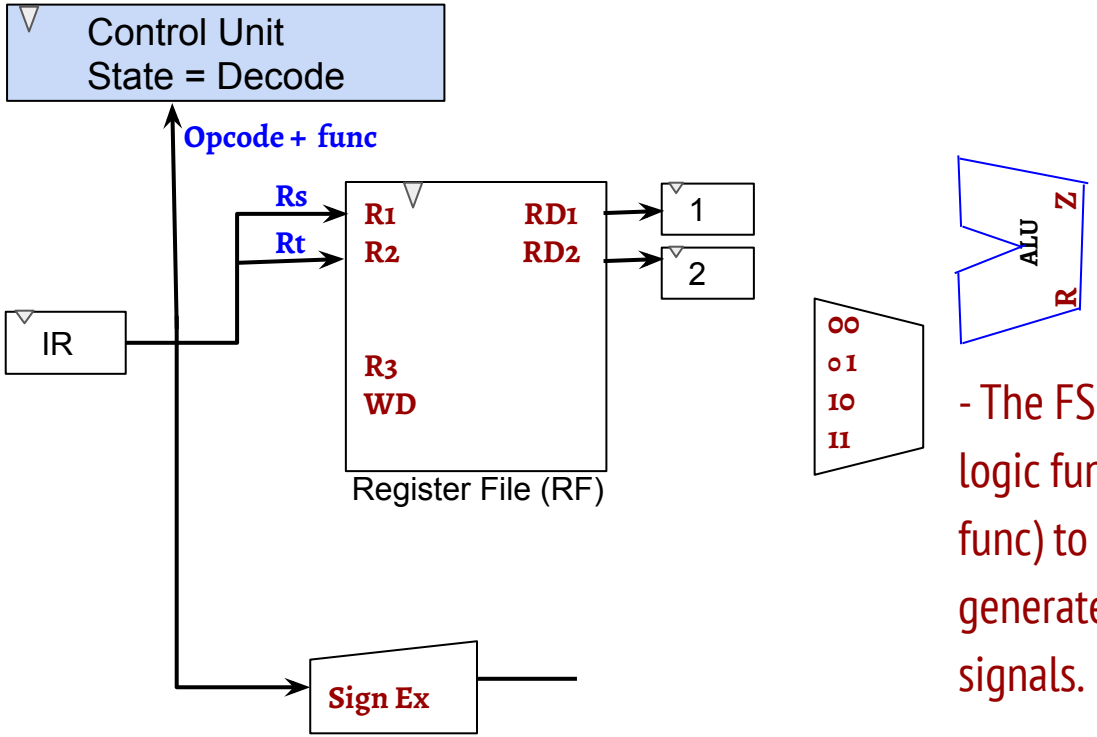
- Next state logic may be implemented as a state machine
- In addition to state change, the controller can generate required control signals

Multi-cycle Fetch (simplified)



- ALU can increment the PC
- Memory is unified
- CU is a sequential circuit (FSM) – during a clock cycle, it uses the state elements to transition to the next state (on next clock) and generate required control signals

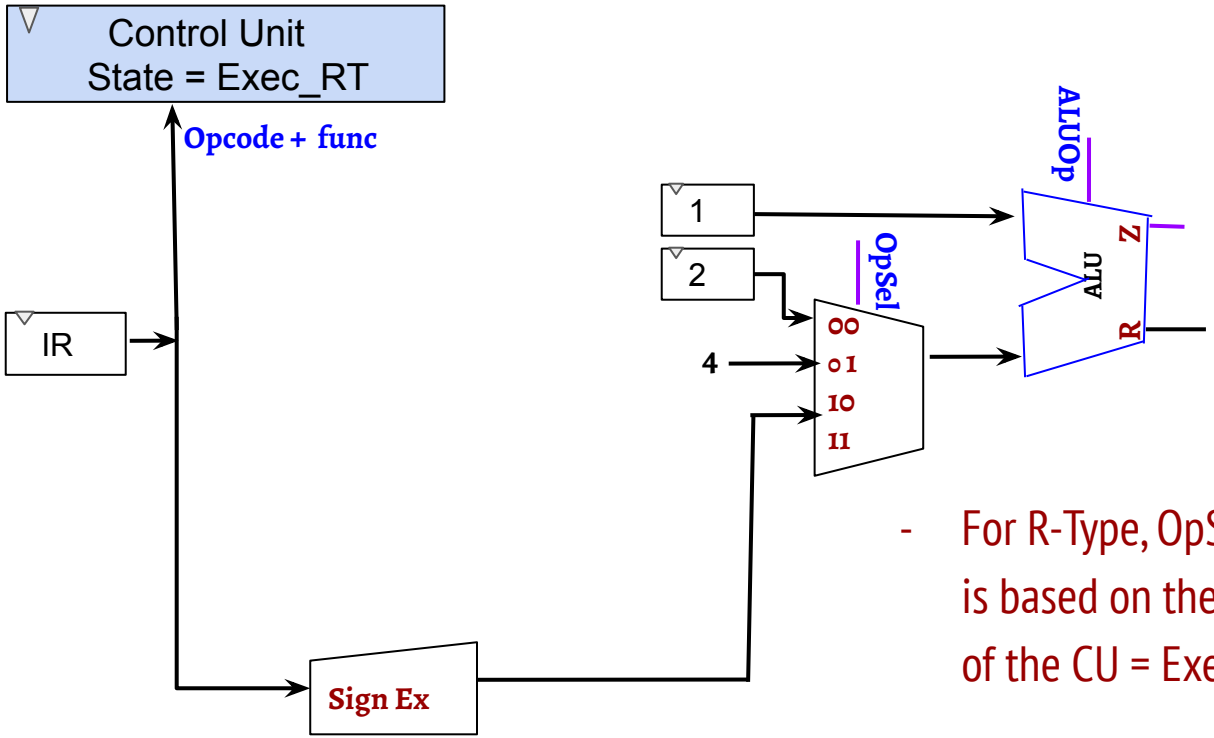
Multi-cycle Decode/RA



- The FSM applies combinatorial logic functions (input = opcode and func) to decide the next state and generate the required control signals.

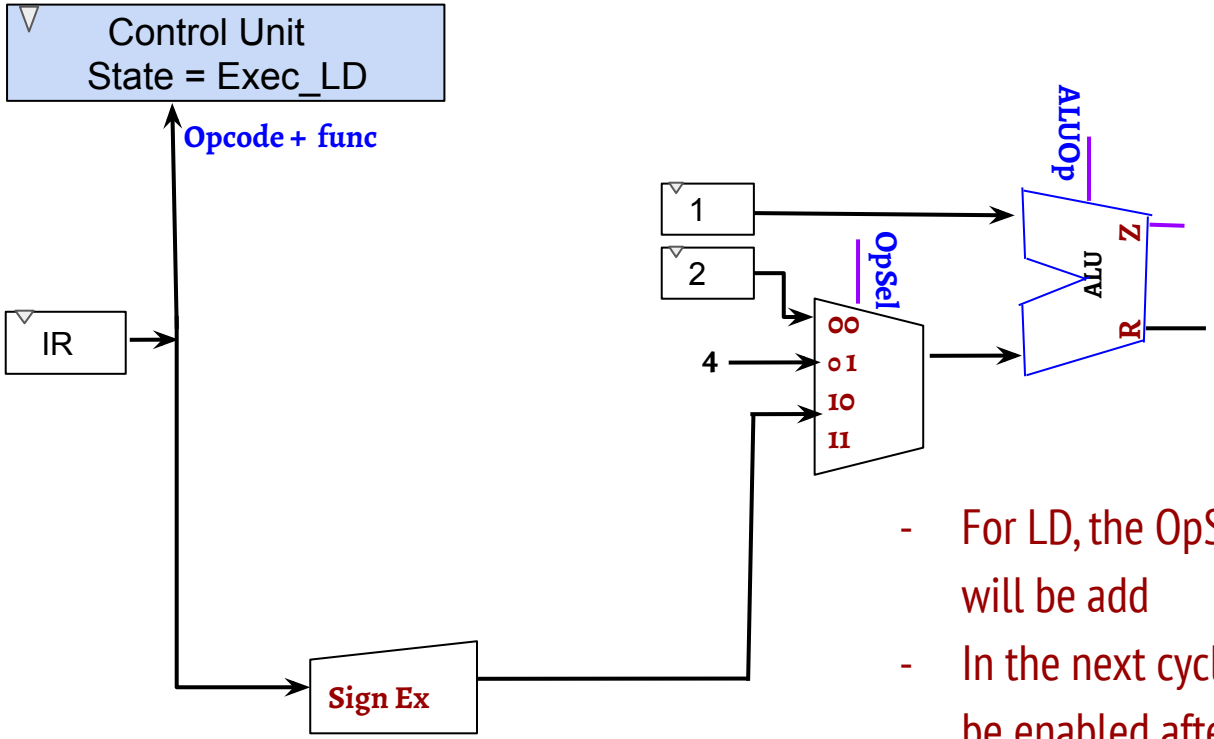
- Operand registers (1 and 2) can be read in next cycle (part of the state)

Multi-cycle Execute (R-Type, LD/ST)



- For R-Type, OpSel = 00 and ALUOp is based on the value of funct (State of the CU = Exec_RT)

Multi-cycle Execute (R-Type, LD/ST)



- For LD, the OpSel = 10 and ALUOp will be add
- In the next cycle (MA), MemR will be enabled after passing ALU result as *Address* with Data = 1

Multi-cycle implementation

- How the clock cycle time is determined?
- How many states in the FSM?
- Advantages?
- Disadvantages?

Multi-cycle implementation

- How the clock cycle time is determined?
    - Slowest stage ⇒ worst case execution time can be more than single cycle
- How many states in the FSM?
    - Depends on complexity of ISA and implementation
- Advantages?
    - Reduction of functional units (because of reuse)
    - Instructions can finish in less number of clock cycles (R-Type in 4 cycles)
- Disadvantages?
    - Even with reuse, functional units remain under-utilized
    - #of states increase with complicated data path

Performance: single-cycle vs. multicycle

Operation	Time
Instruction Fetch	200 ps
Register Read/Write	150 ps
ALU Operation	150 ps
Data Access	200 ps

- Clock cycle time: Single-cycle = ?, Multi-cycle = ?
- Consider a workload with 10 billion instructions where 40% are R-Type, 10% are Load, 20% are Store and 30% are branch instructions. What will be the execution time in single-cycle and multi-cycle implementations?

Performance: single-cycle vs. multicycle

Operation	Time
Instruction Fetch	200 ps
Register Read/Write	150 ps
ALU Operation	150 ps
Data Access	200 ps

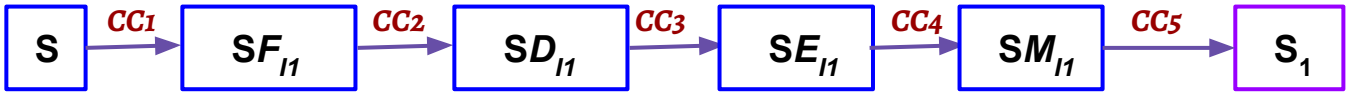
- Clock cycle time: Single-cycle = 850 ps, Multi-cycle = 200 ps
- Consider a workload with 10 billion instructions where 40% are R-Type, 10% are Load, 20% are Store and 30% are branch instructions. What will be the execution time in single-cycle and multi-cycle implementations? Single-cycle = 8.5 sec Multi-cycle = 7.6 sec

Pipelining overview

- Software visible state transition (**S** → **Q**) in **K** clock cycles for a given instruction
- While **I**th instruction is in 2nd step (stage), **(I+1)**th instruction performs step 1
- Non-overlapping use of functional units/storage elements
- Implementation must address the issues due to changes to state by different instructions in the same clock cycle

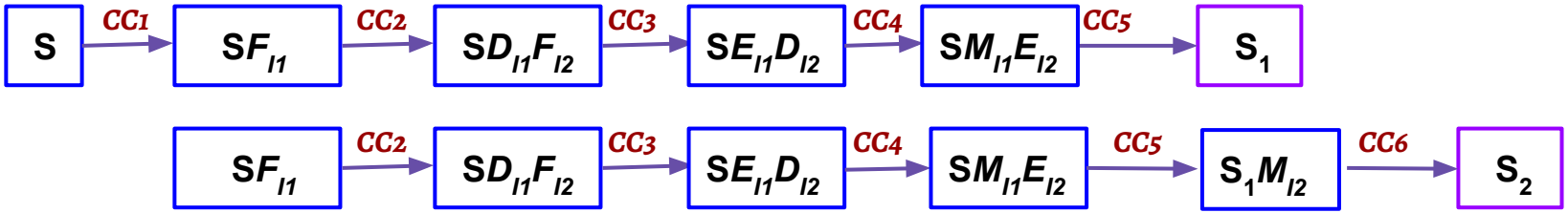
Pipelining overview

- Software visible state transition (**S** → **Q**) in **K** clock cycles for a given instruction
- While **I**th instruction is in 2nd step (stage), **(I+1)**th instruction performs step 1
- Non-overlapping use of functional units/storage elements
- Implementation must address the issues due to changes to state by different instructions in the same clock cycle



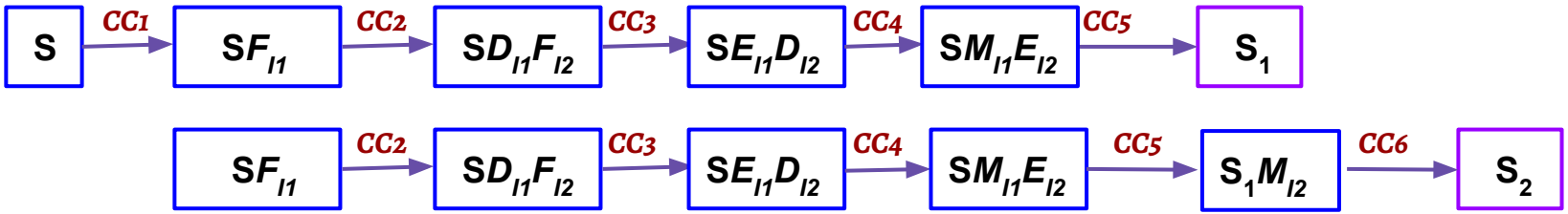
Pipelining overview

- Software visible state transition (**S** → **Q**) in **K** clock cycles for a given instruction
- While **I**th instruction is in 2nd step (stage), **(I+1)**th instruction performs step 1
- Non-overlapping use of functional units/storage elements
- Implementation must address the issues due to changes to state by different instructions in the same clock cycle



Pipelining overview

- Software visible state transition (**S** → **Q**) in **K** clock cycles for a given instruction
- While **I**th instruction is in 2nd step (stage), **(I+1)**th instruction performs step 1
- Non-overlapping use of functional units/storage elements
- Implementation must address the issues due to changes to state by different instructions in the same clock cycle



- State overlap: can be useful and create problems at the same time. Example: PC
- Typical implementations copy and carry the required state along the pipeline

Ideal pipeline performance

Operation	Time
Instruction Fetch	200 ps
Register Read/Write	150 ps
ALU Operation	150 ps
Data Access	200 ps

What will be the time taken to execute the workload with a 5-stage pipeline?

- Clock cycle time: Single-cycle = 850 ps, Multi-cycle = 200 ps
- Consider a workload with 10 billion instructions where 40% are R-Type, 10% are Load, 20% are Store and 30% are branch instructions. What will be the execution time in single-cycle and multi-cycle implementations? Single-cycle = 8.5 sec Multi-cycle = 7.6 sec

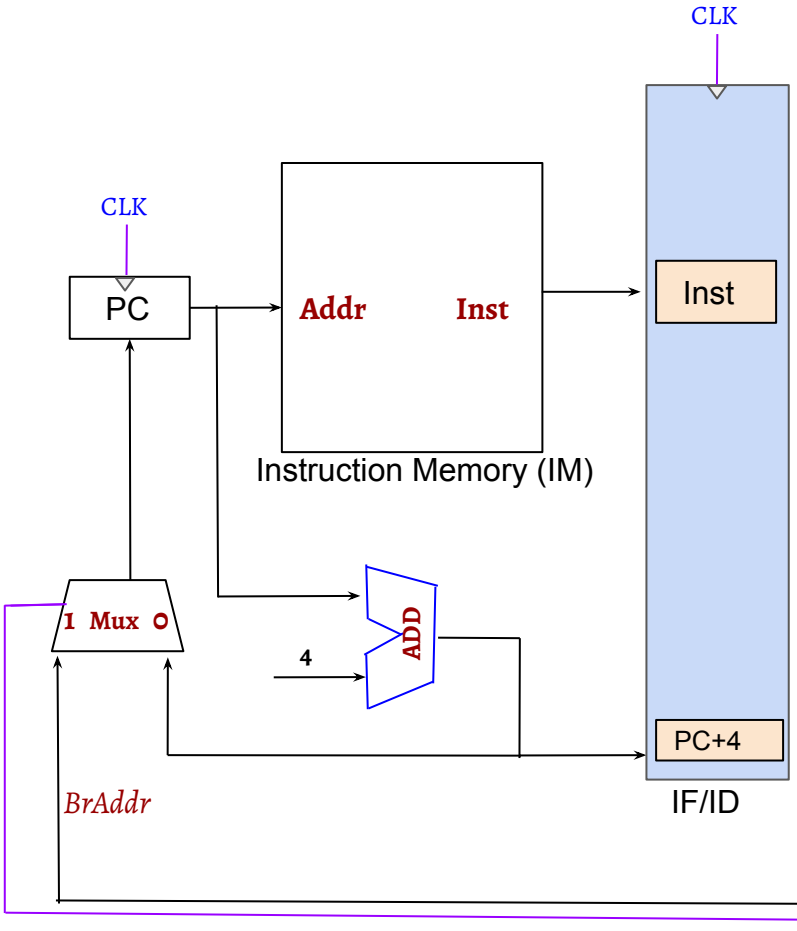
Ideal pipeline performance

Operation	Time
Instruction Fetch	200 ps
Register Read/Write	150 ps
ALU Operation	150 ps
Data Access	200 ps

What will be the time taken to execute the workload with a 5-stage pipeline? 2sec

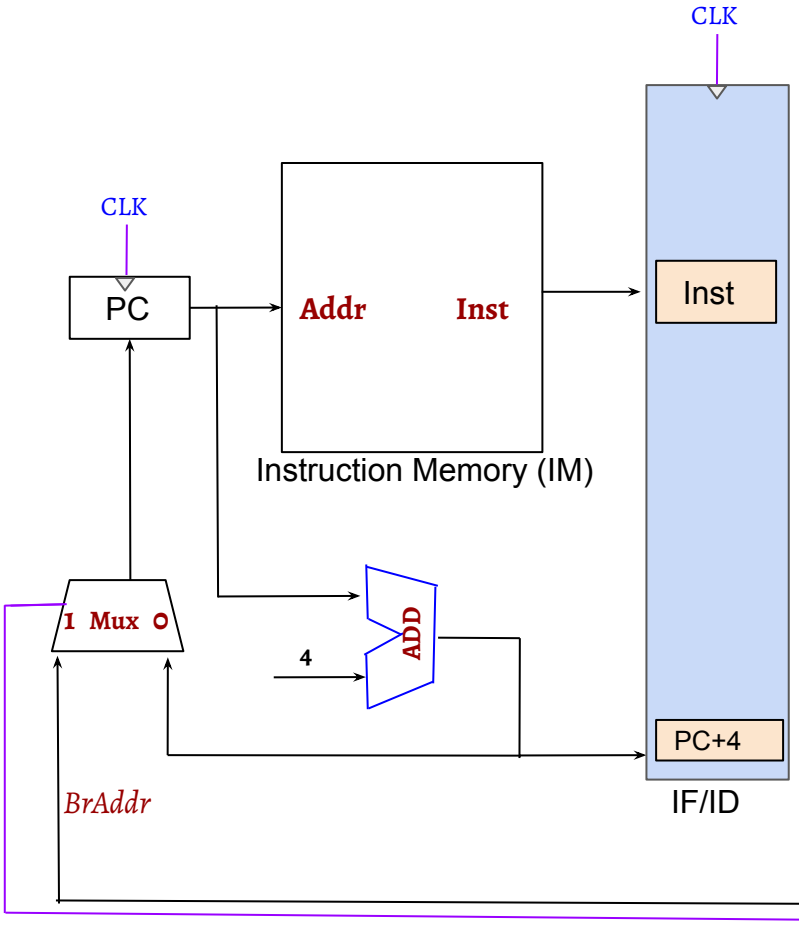
- Clock cycle time: Single-cycle = 850 ps, Multi-cycle = 200 ps
- Consider a workload with 10 billion instructions where 40% are R-Type, 10% are Load, 20% are Store and 30% are branch instructions. What will be the execution time in single-cycle and multi-cycle implementations? Single-cycle = 8.5 sec Multi-cycle = 7.6 sec

Pipelining: Fetch



- Why ALU can not be used to increment PC similar to multi-cycle?
- Why IR and PC are required to be carried along?
- How many pipeline registers (to maintain state) are required?

Pipelining: Fetch



- Why ALU can not be used to increment PC similar to multi-cycle?
  - Can not use ALU because a prev. instruction in Ex/Ad stage may be using
- Why instruction and PC are required to be carried along?
  - Future instructions will modify the PC or the instruction register
- How many pipeline registers (to maintain state) are required?
  - Four assuming write in the rising edge, after which read is performed