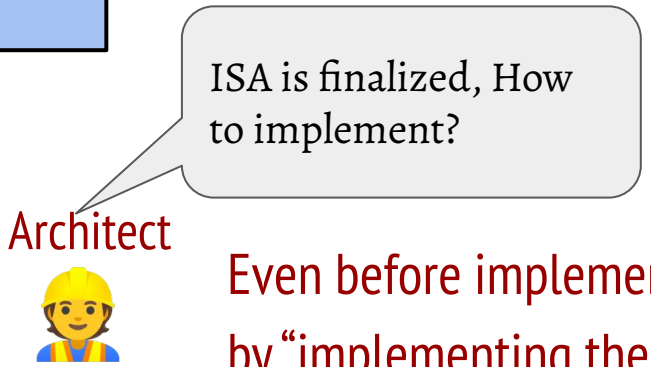
Computer Architecture

Microarchitecture

Debadatta Mishra, CSE, IITK

Microarchitecture (recap)

Instruction Set Architecture



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

Combinatorial and Sequential Logic

Hardware primitives

Microarchitecture (recap)

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)

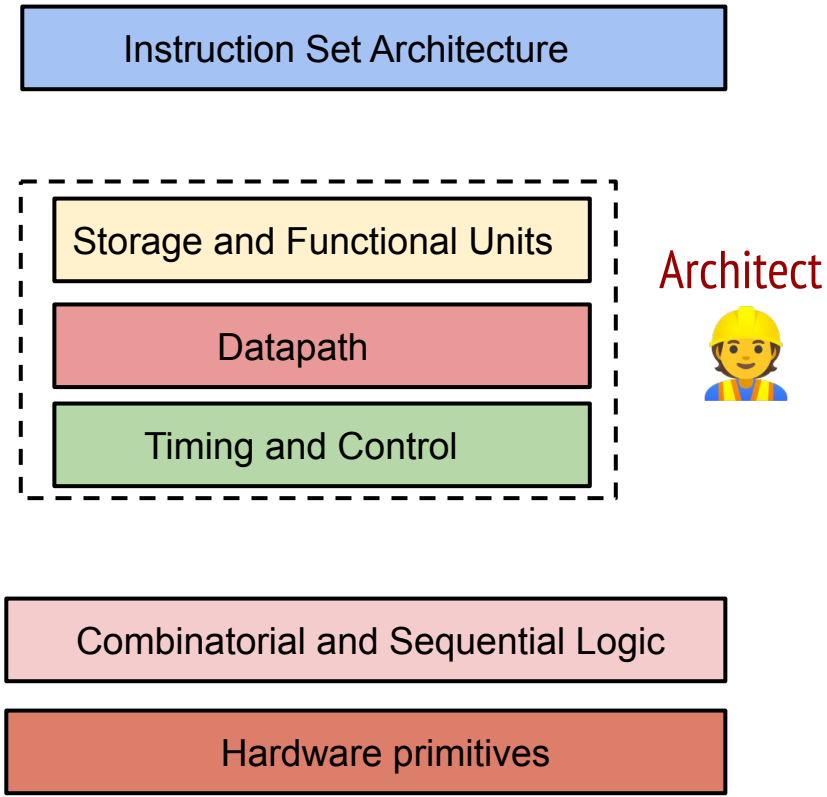
Execute a given instruction

Software visible state of the system (Q)

Combinatorial and Sequential Logic

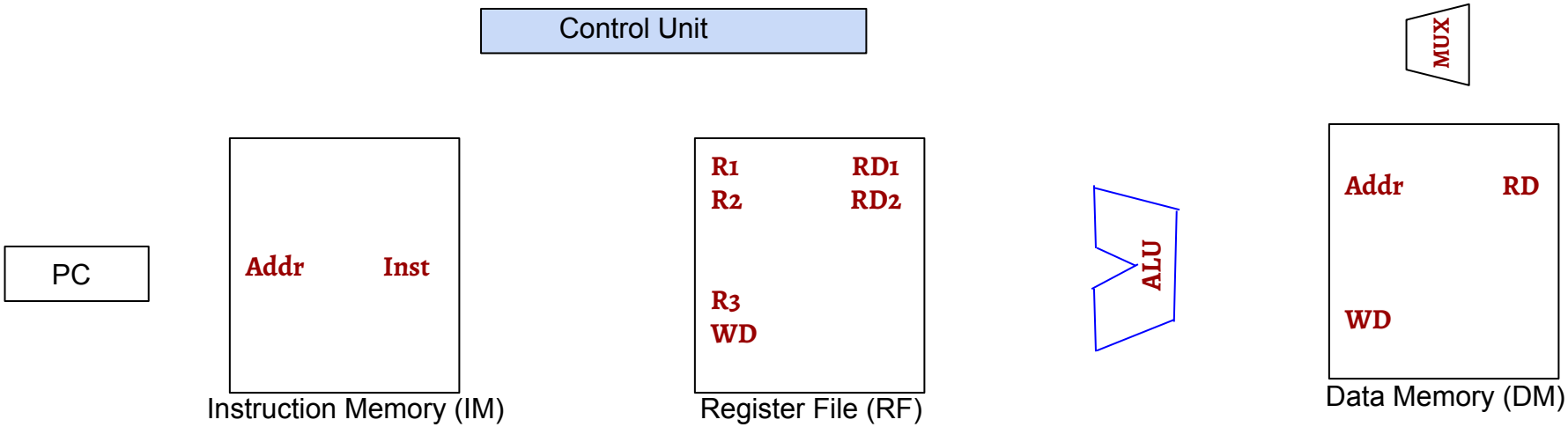
Hardware primitives

Microarchitecture



- 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?

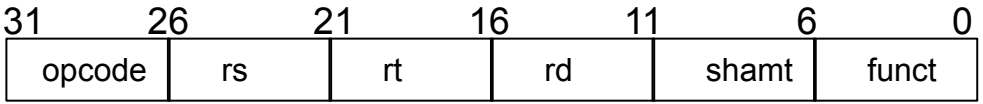
Architectural State and Functional Units



- Register file can be addressed using the register number (0-31) with two read ports and one write port (read and write in the same cycle)
- Instruction memory can be read and data memory can be read/written in the same cycle
- For write into register file or data memory write should be enabled

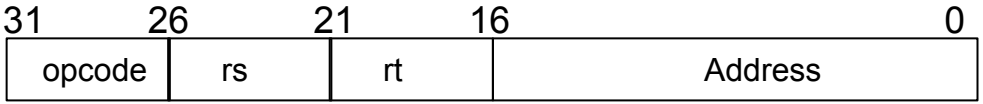
Instruction Semantics

R-Type



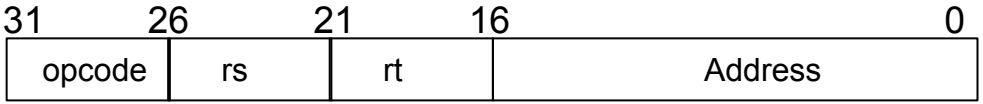
- Source regs: **rs** and **rt** Destination register: **rd** (example: add s1, s2, s3)
- The 'funct' bits are used to decode the the ALU operation (add, sub, and, or, slt)

LD/ST



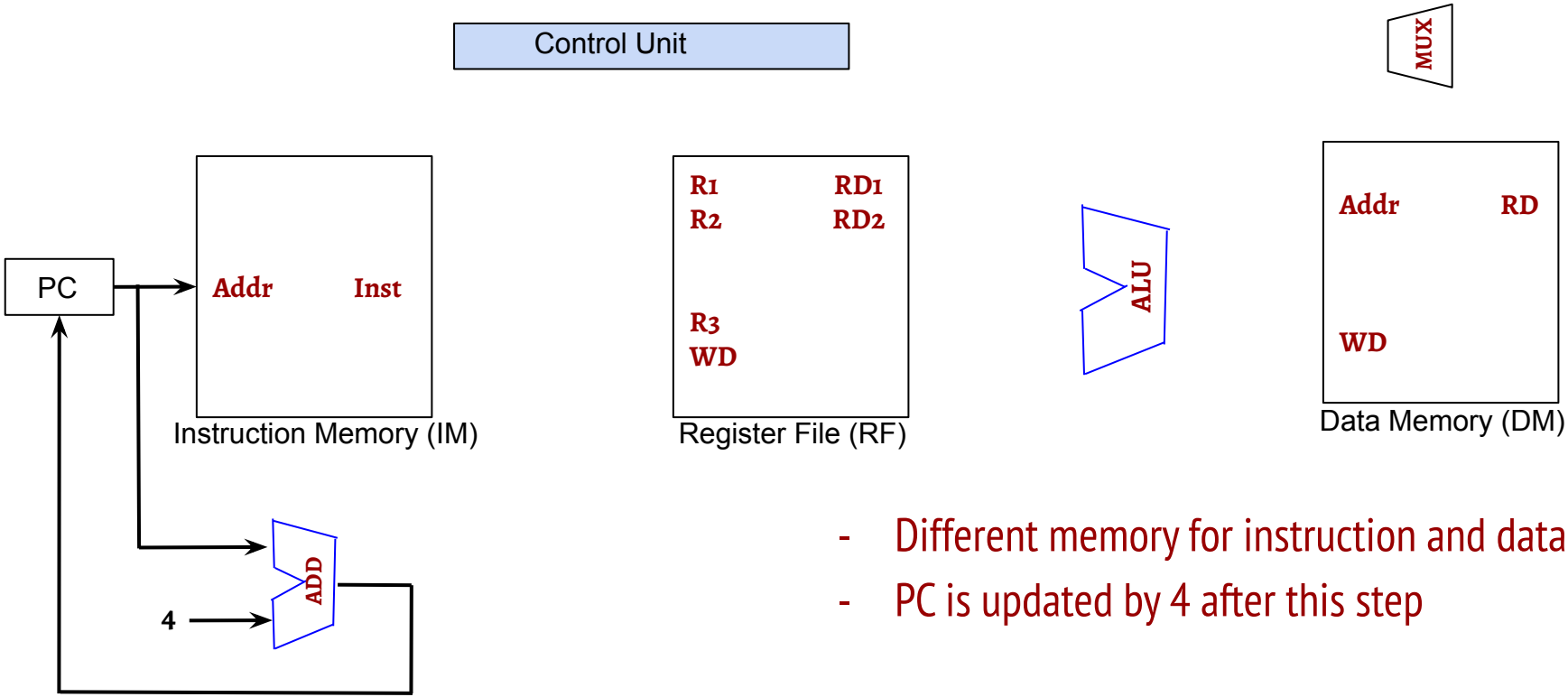
- Memory address = **rs** + **address** (operation in ALU) LD destination or ST data: **rt**
- Depending on Load/Store, the usage semantics of **rt** changes

BEQ



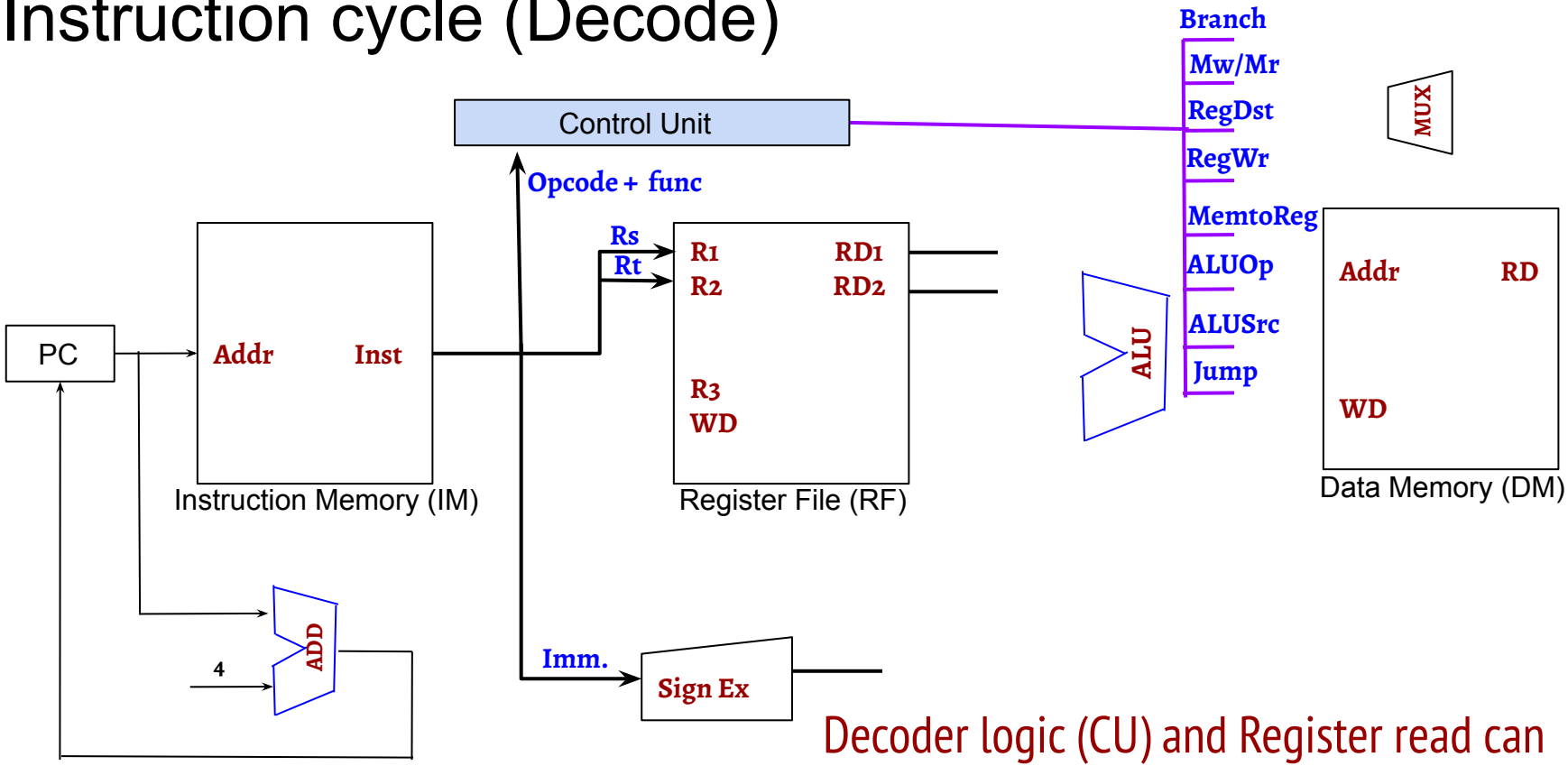
- Registers **rs** and **rt** are compared for equality (comparison in ALU)
- 16-bit address is added to PC if **rs = rt**

Instruction cycle (Fetch)



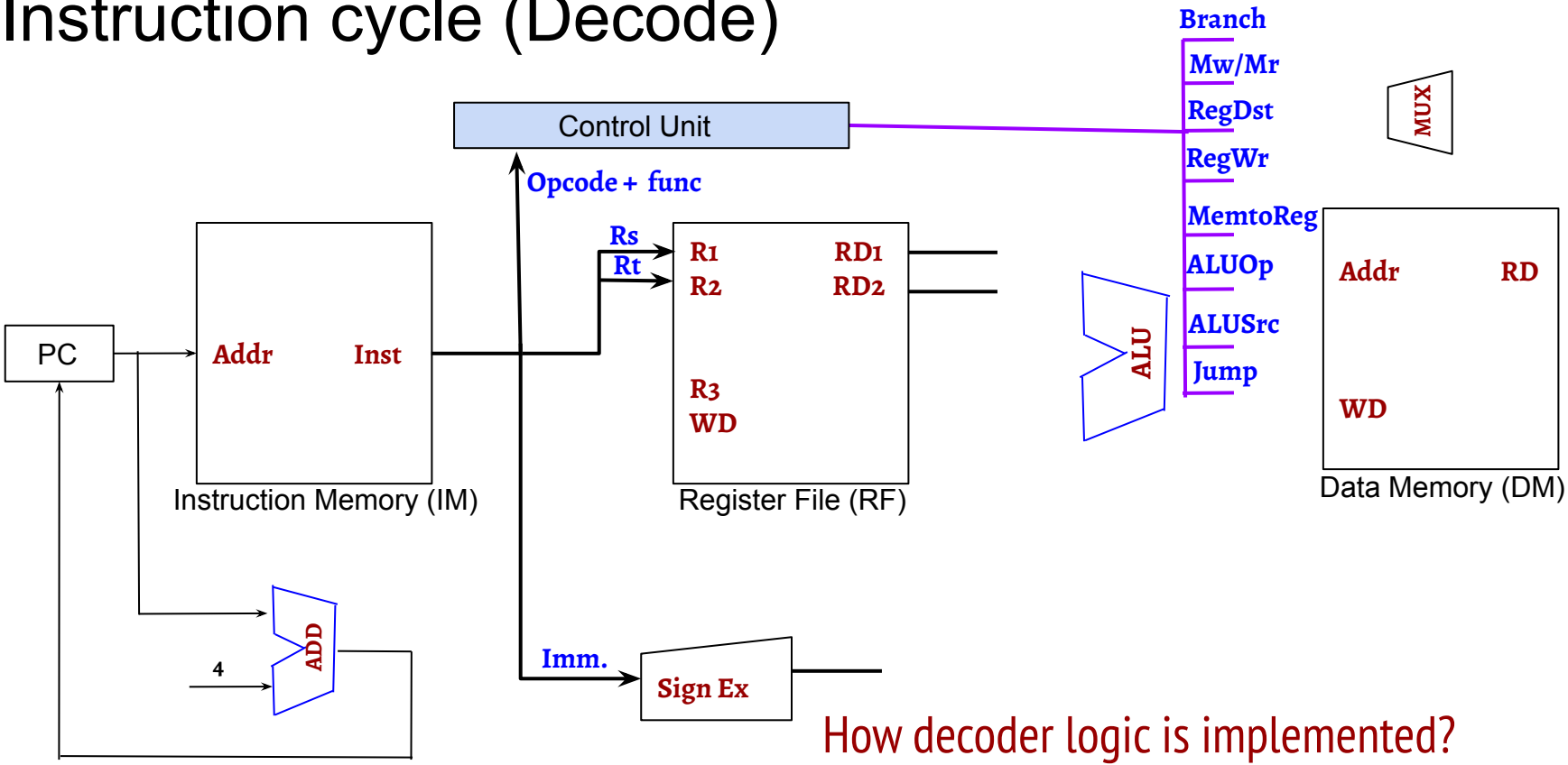
- Different memory for instruction and data
- PC is updated by 4 after this step

Instruction cycle (Decode)



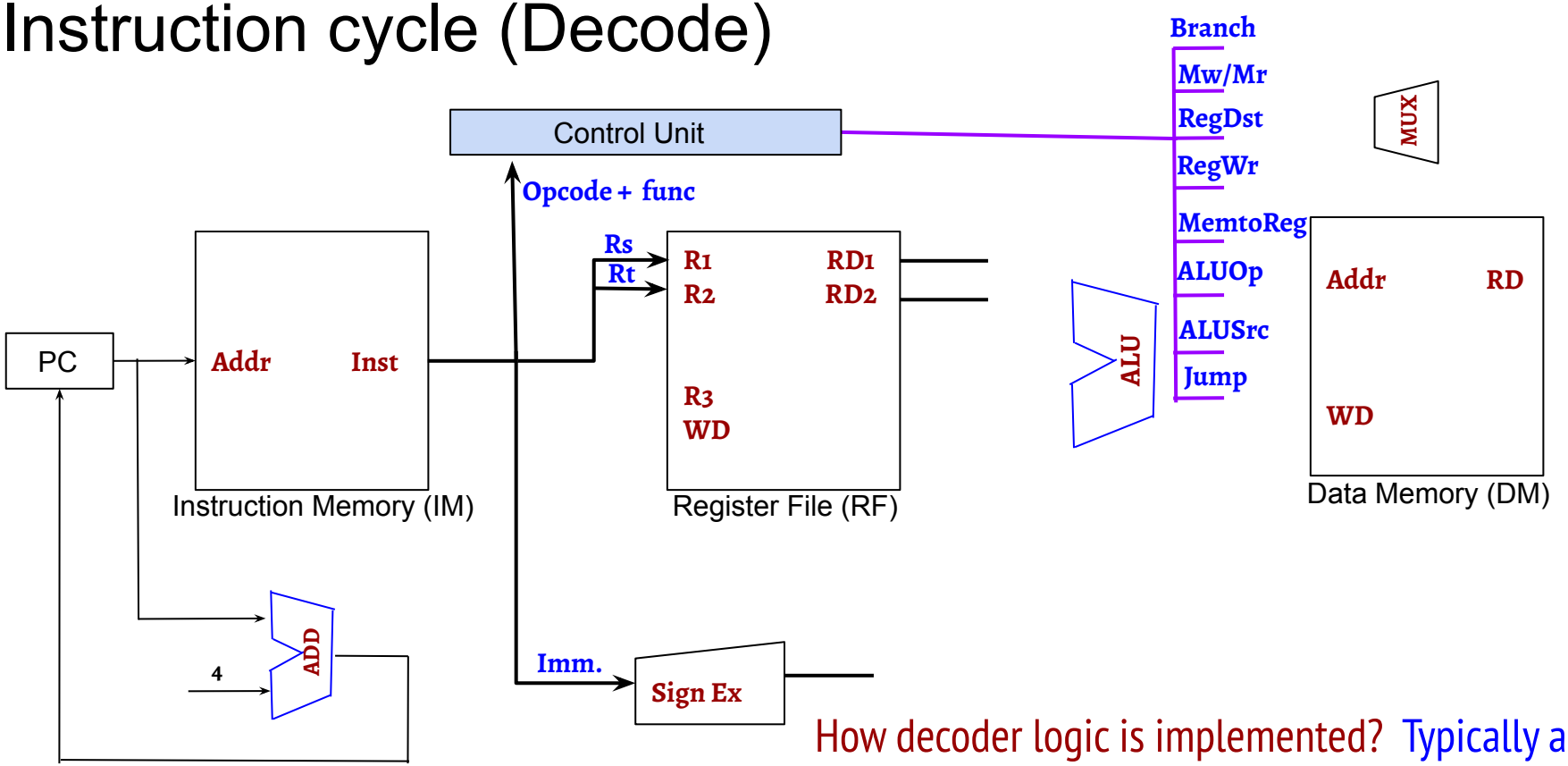
Decoder logic (CU) and Register read can happen concurrently. All control signals must be ready before the next step

Instruction cycle (Decode)



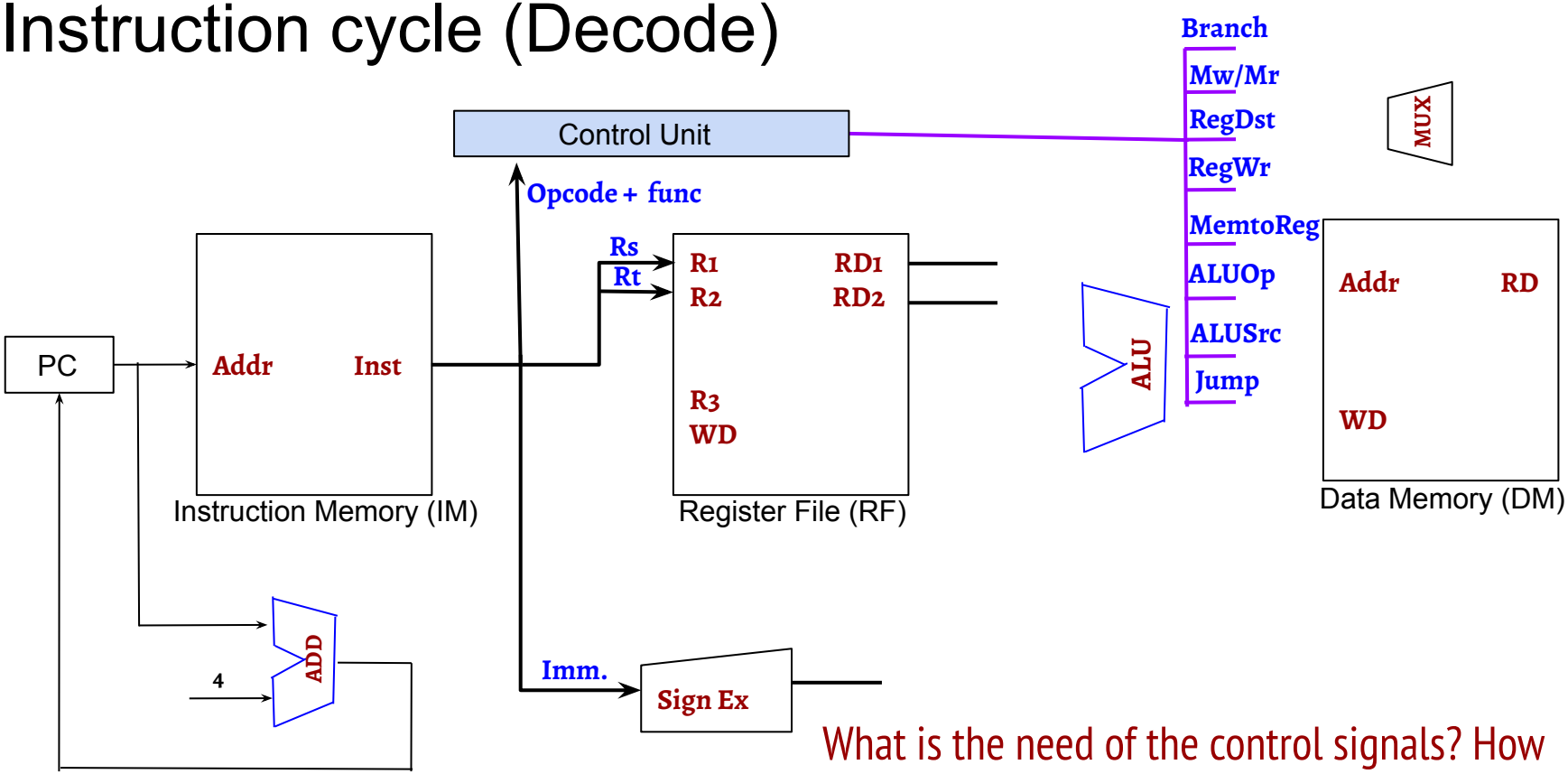
How decoder logic is implemented?

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)

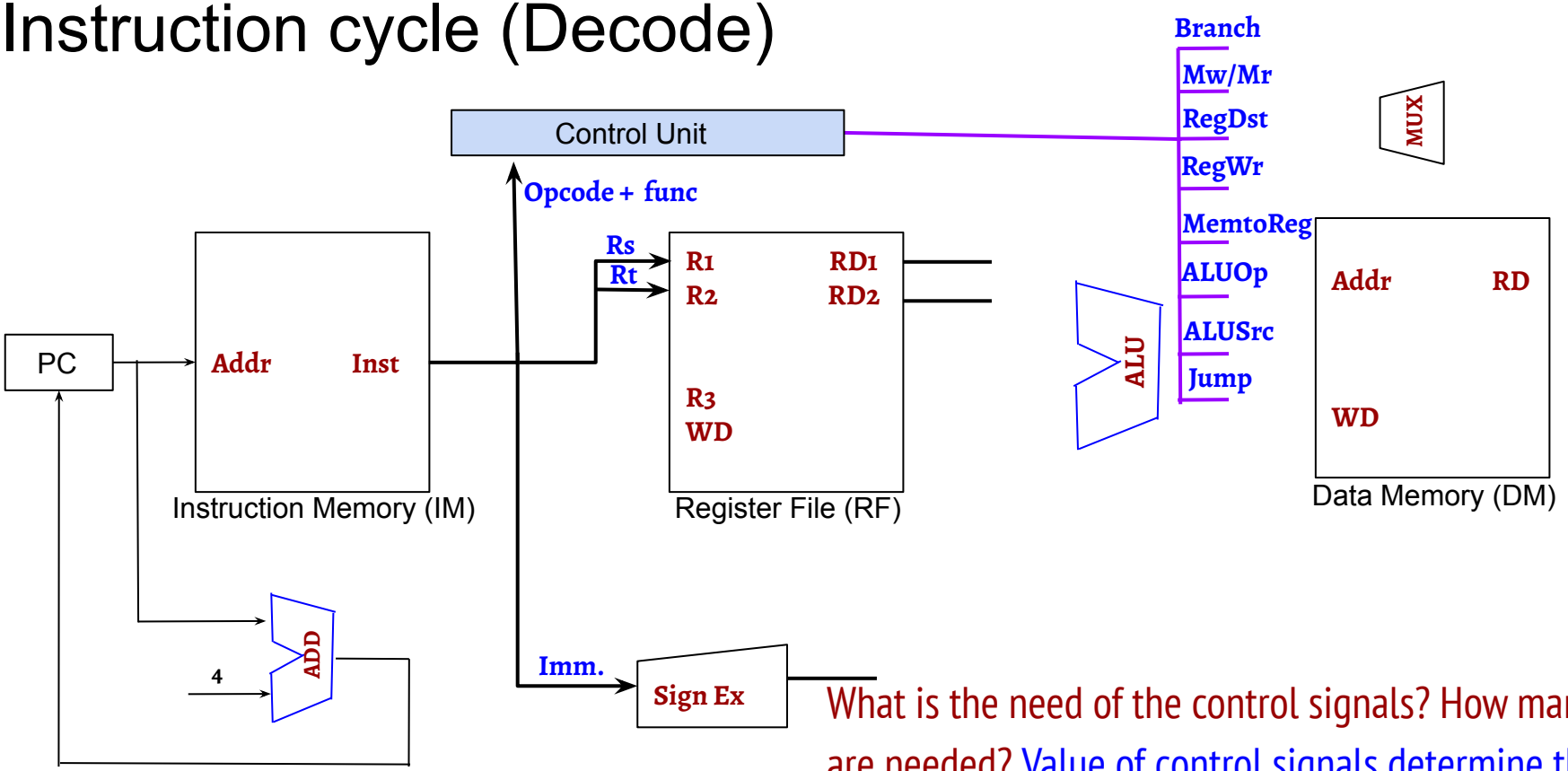


What is the need of the control signals? How many are needed?

Control Signals

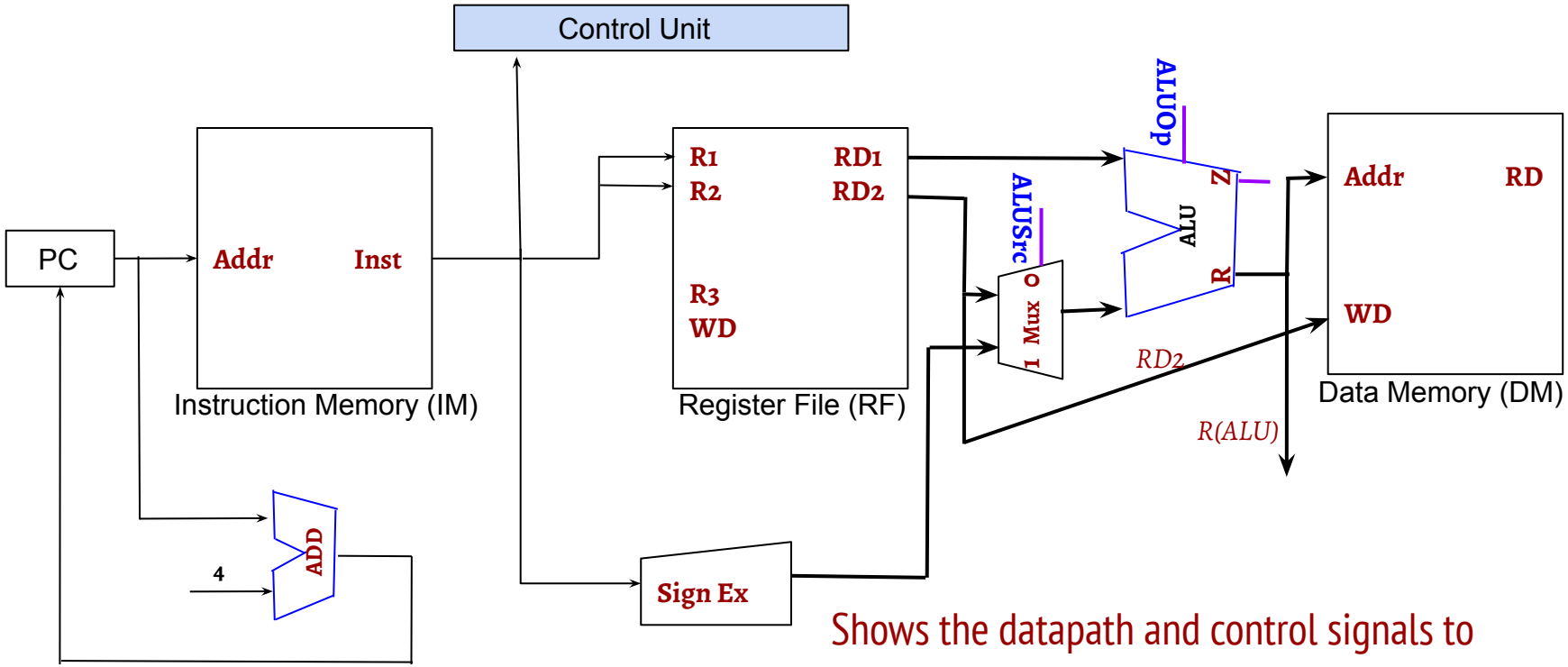
Branch	Set if Branch(opcode) == TRUE
RegDst	Set if R-TYPE (opcode) == TRUE
RegWrite	Set if (R-TYPE (opcode) == TRUE OR IsLoad(opcode) == TRUE)
MemRead	Set if IsLoad(opcode) == TRUE
MemWrite	Set if IsStore(opcode) == TRUE
ALUSrc	Set if the second ALU input is from address or Imm (BEQ, LD/ST)
ALUOp	Determined by the 6bits funct (AND, OR, ADD, SUB etc,)
MemtoReg	Set if IsLoad(opcode) == TRUE (Write to register is from memory)

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.

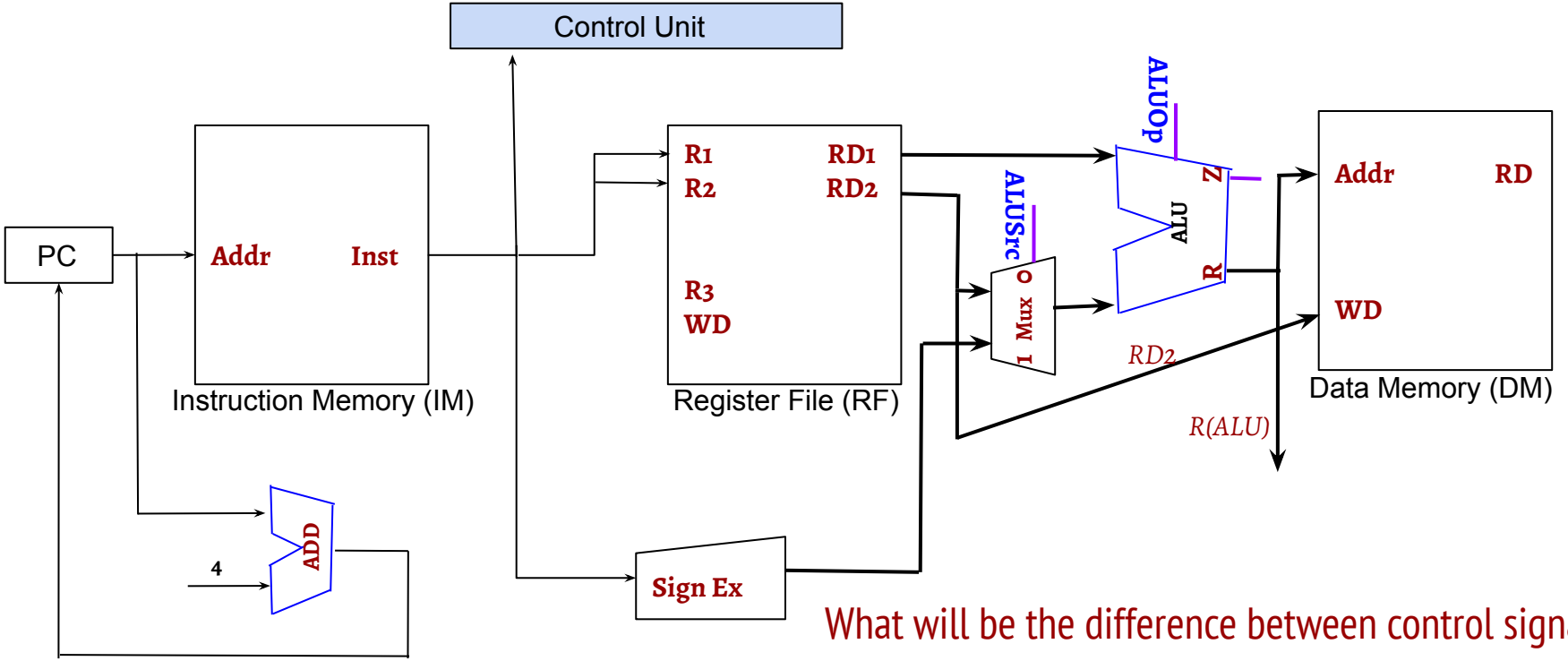
Instruction cycle (Execute or Address Calculate)



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

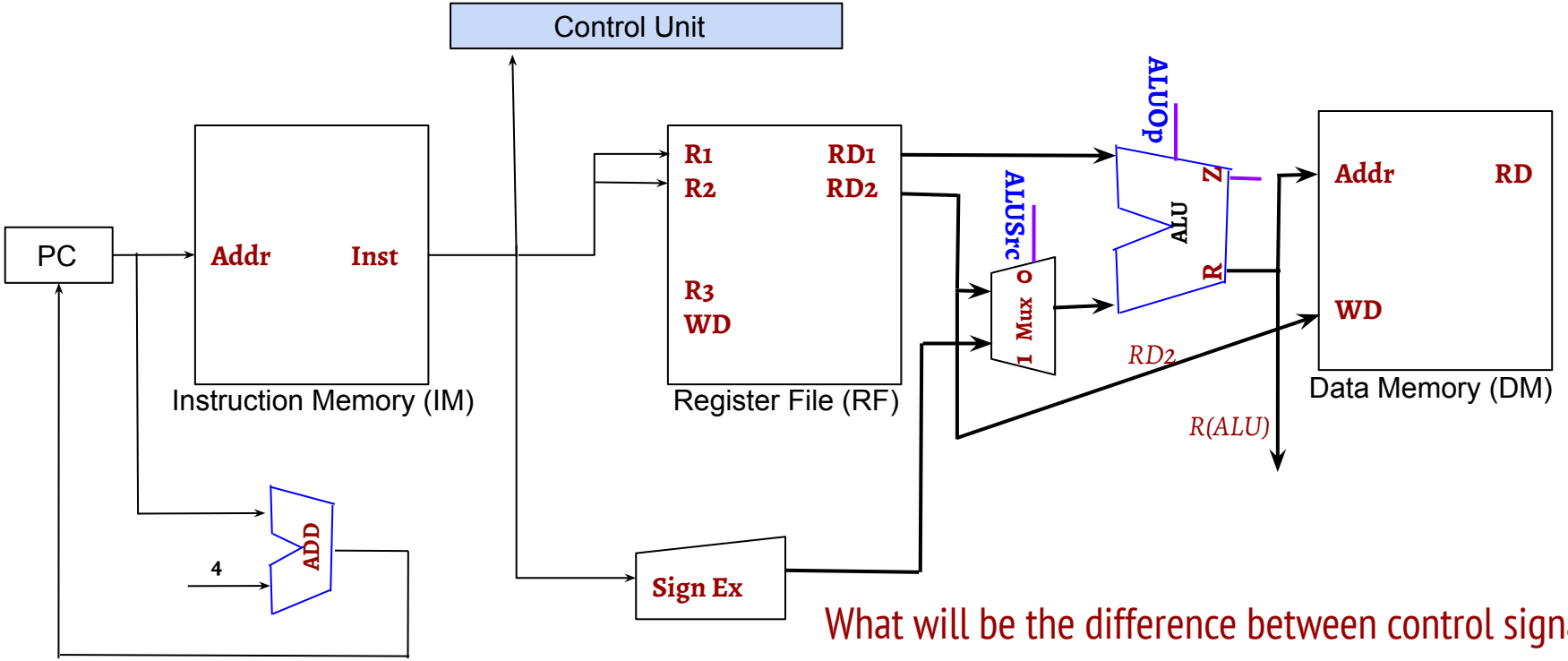
RD_2 contains the data for store instruction

Instruction cycle (Execute or Address Calculate)



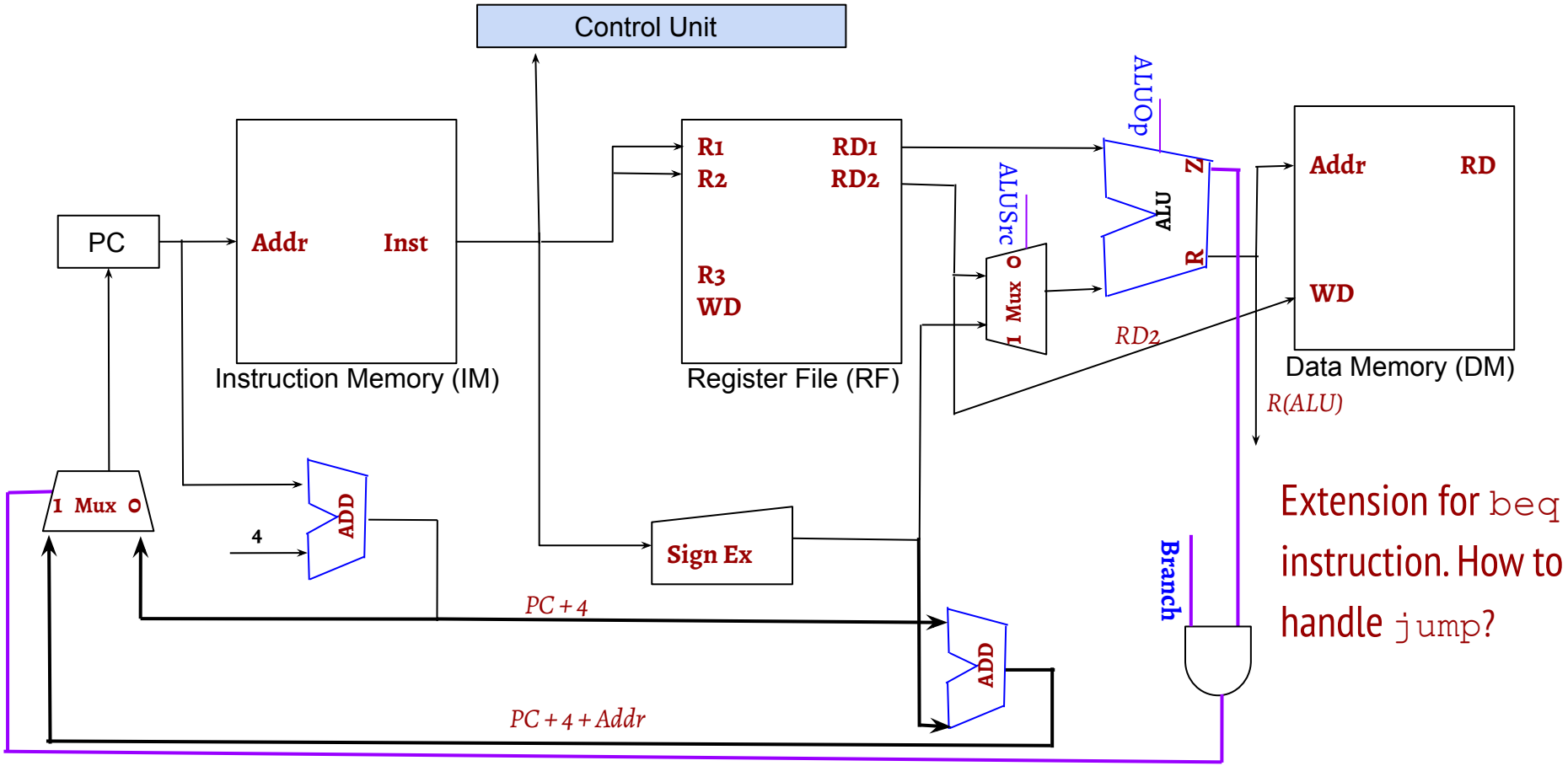
What will be the difference between control signal values between `add` and `addi` instructions?

Instruction cycle (Execute or Address Calculate)



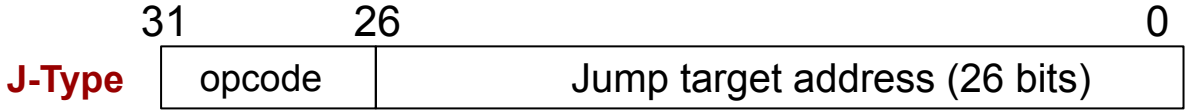
What will be the difference between control signal values between `add` and `addi` instructions? Only ALUSrc input will be different (for this step)

Instruction cycle (Execute or Address Calculate)



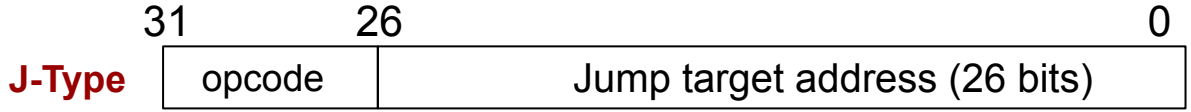
Extension for `beq` instruction. How to handle `jump`?

Jump instruction in MIPS

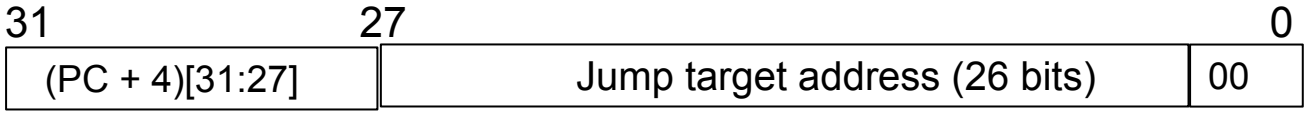


- Jump instructions provide the target address in direct address form (absolute address)
- It has 26 bits to specify the address; with fixed instruction size (4-bytes) it can cover 2^{28} bytes. How to express 32-bit address?

Jump instruction in MIPS

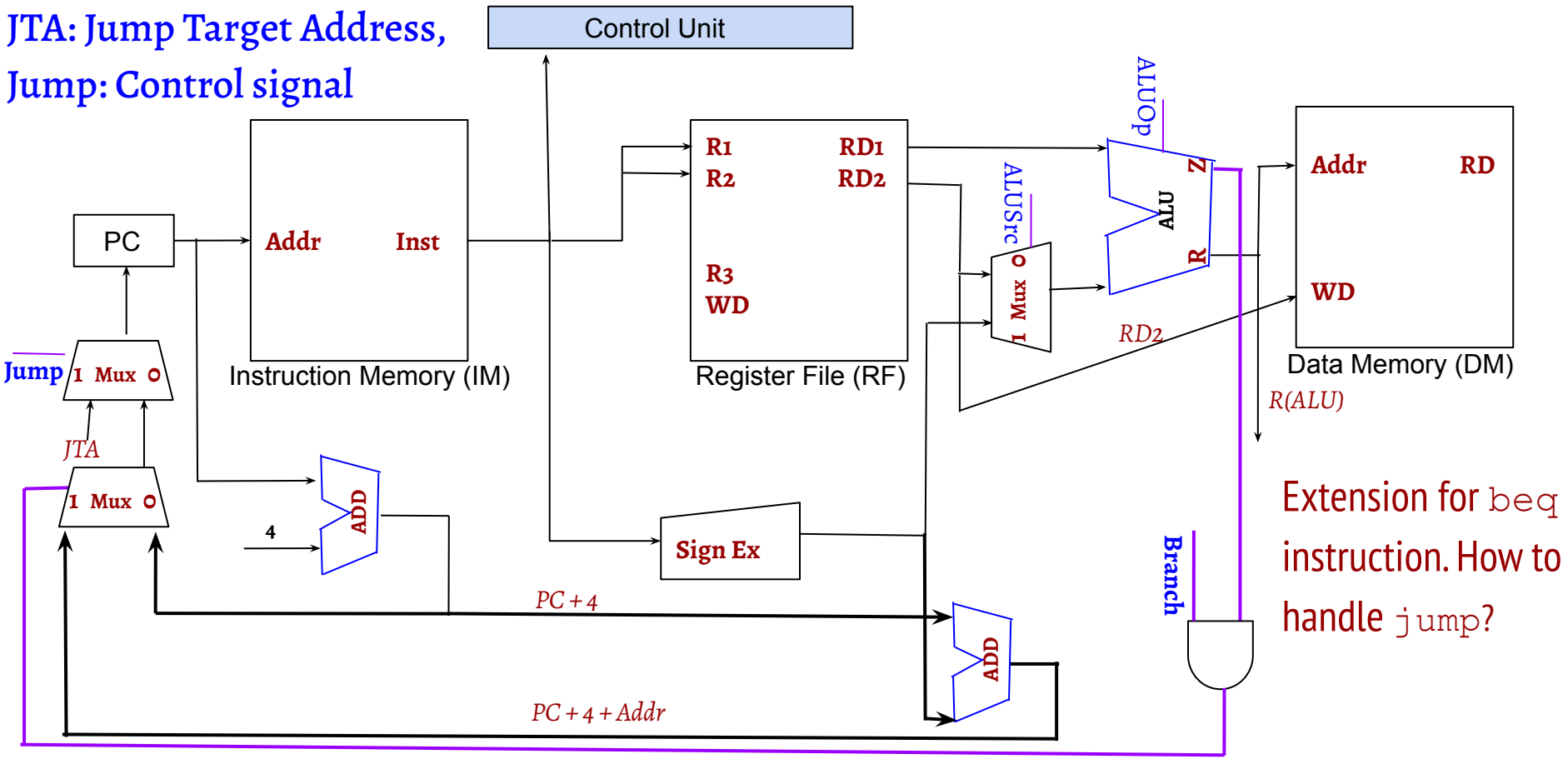


- Jump instructions provide the target address in direct address form (absolute address)
- It has 26 bits to specify the address; with fixed instruction size (4-bytes) it can cover 2^{28} bytes. How to express 32-bit address?

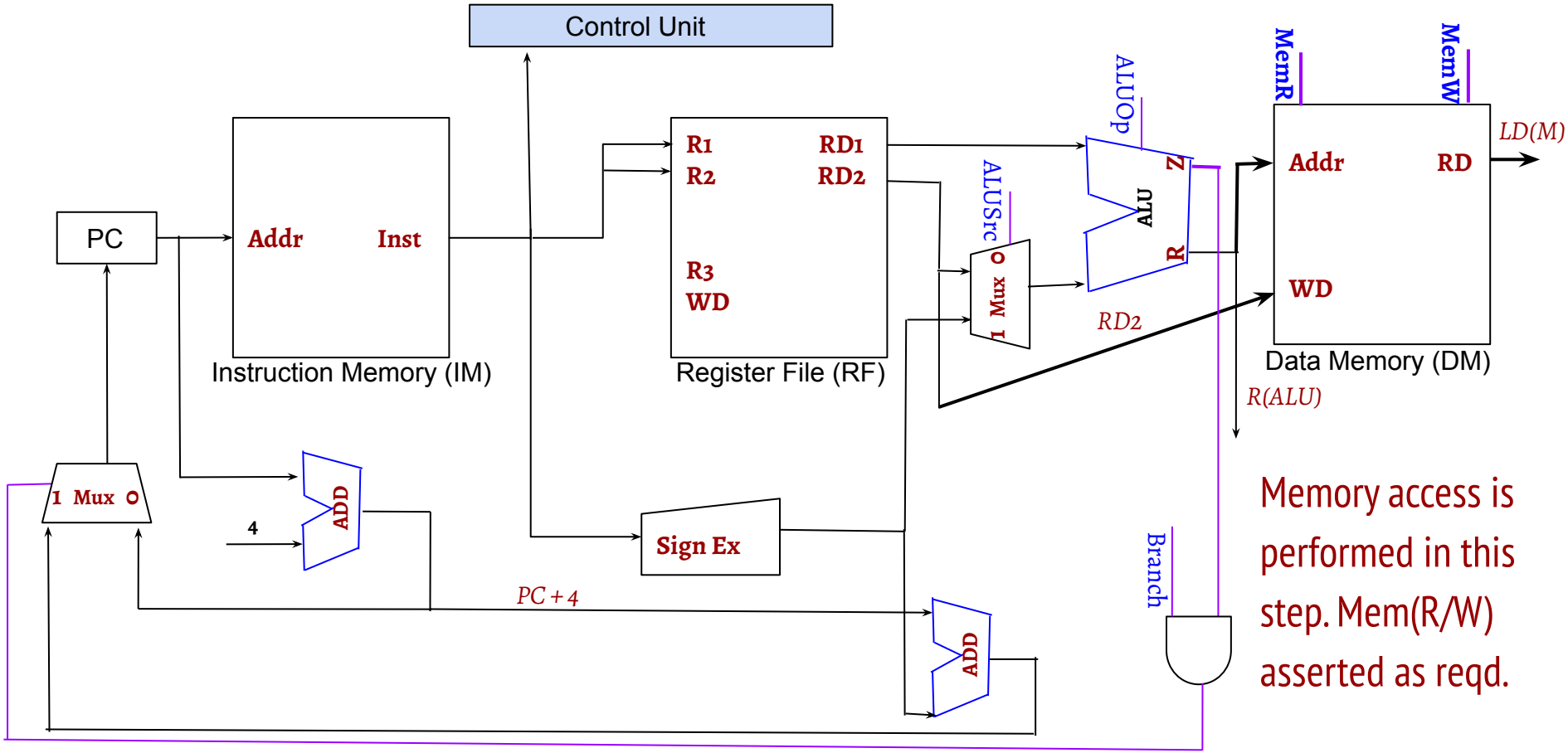


- Can use additional combinatorial logic to calculate the new value of PC
- Use and additional selector (mux) using `jump` control signal

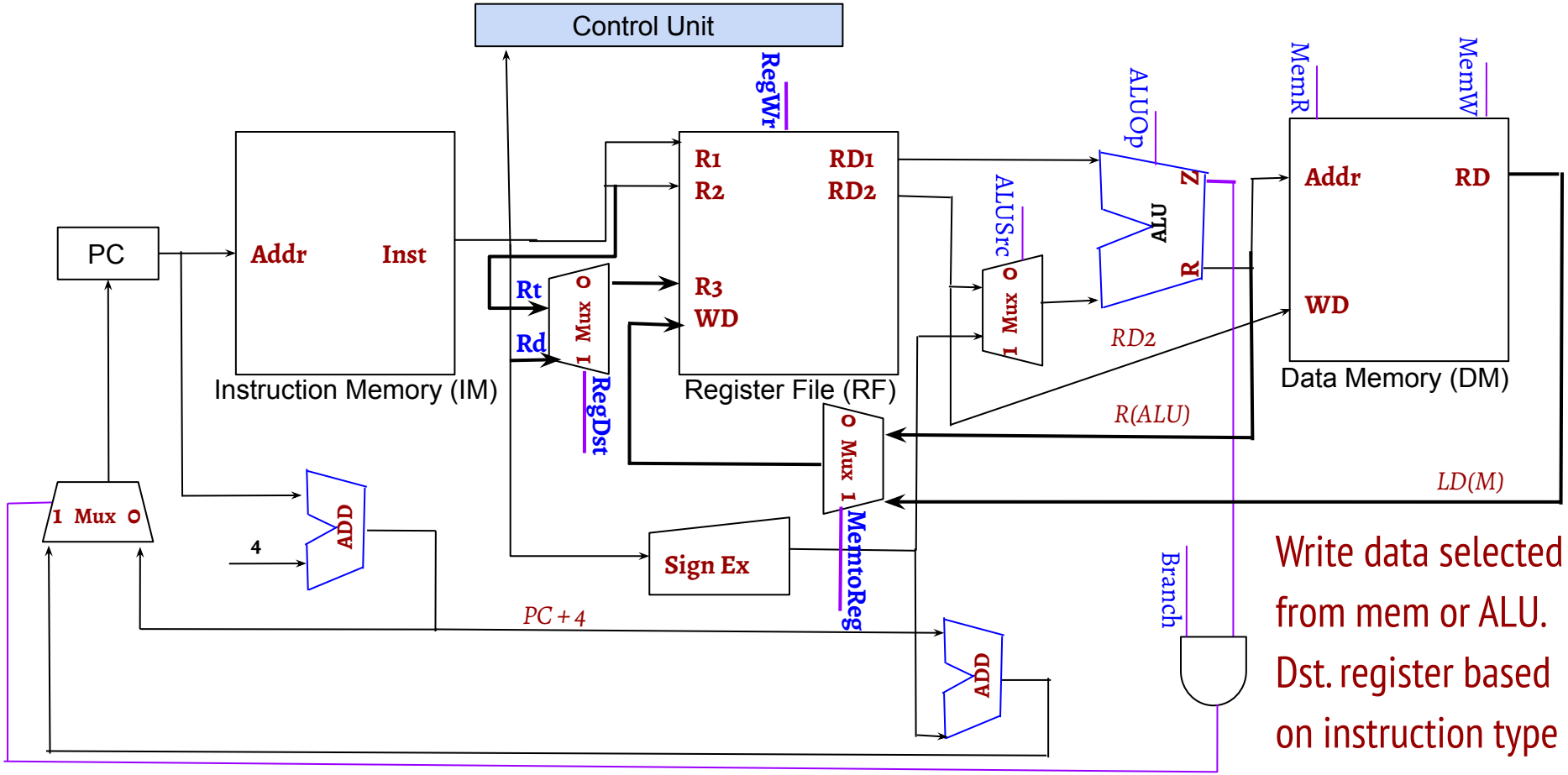
Instruction cycle (Execute or Address Calculate)



Instruction cycle (Memory Data Access)



Instruction cycle (Register Write)



Instruction Cycle

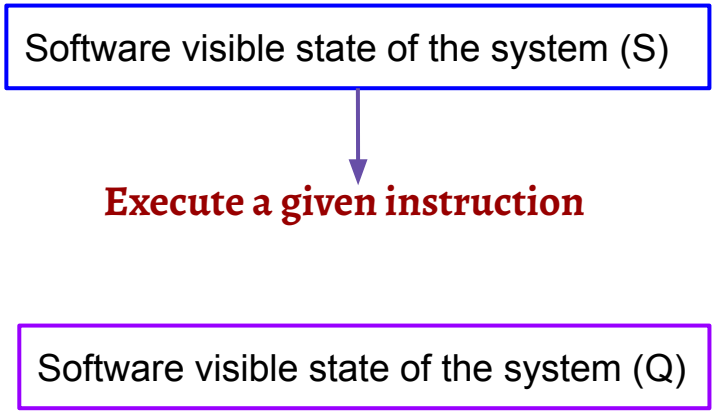
- What will be the control signal values
    - For R-type
    - For Load
    - For BEQ
- How to tackle increased complexity in decoding?

Instruction Cycle

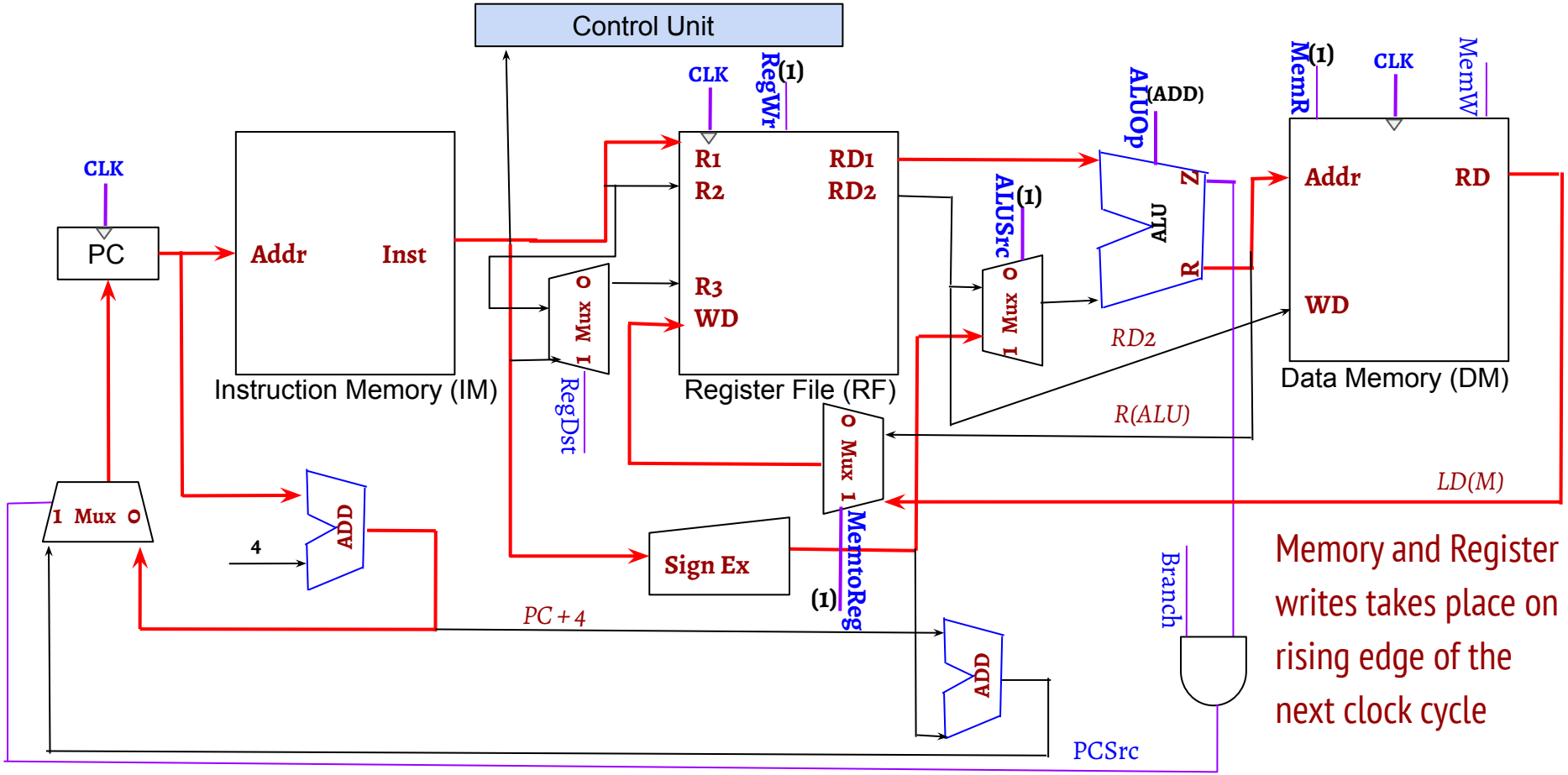
- What will be the control signal values
  - For R-type: RegDst = 1, RegWr = 1, everything is zero. ALUOp = `ALUFunc(funct)`, where `ALUFunc` is a combinatorial logic function
  - For Load: ALUSrc=1, MemR=1, MemtoReg=1, RegWr=1, everything else is zero. ALUOp = `AluFunc`(Opcode, ADD)
  - For BEQ: Branch=1, 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 datapath and control (Load)



Single cycle implementation

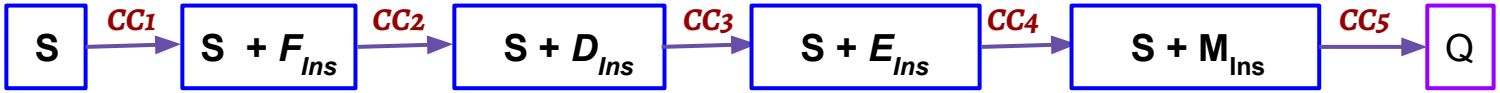
- All steps of instruction execution are performed in a single clock cycle
- Software visible state transition (S ⟶ Q) in a single clock cycle
- Advantages
    - Do not require any additional micro-architectural state
    - Simple control logic
- Disadvantages
    - Clock cycle determined by the slowest 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

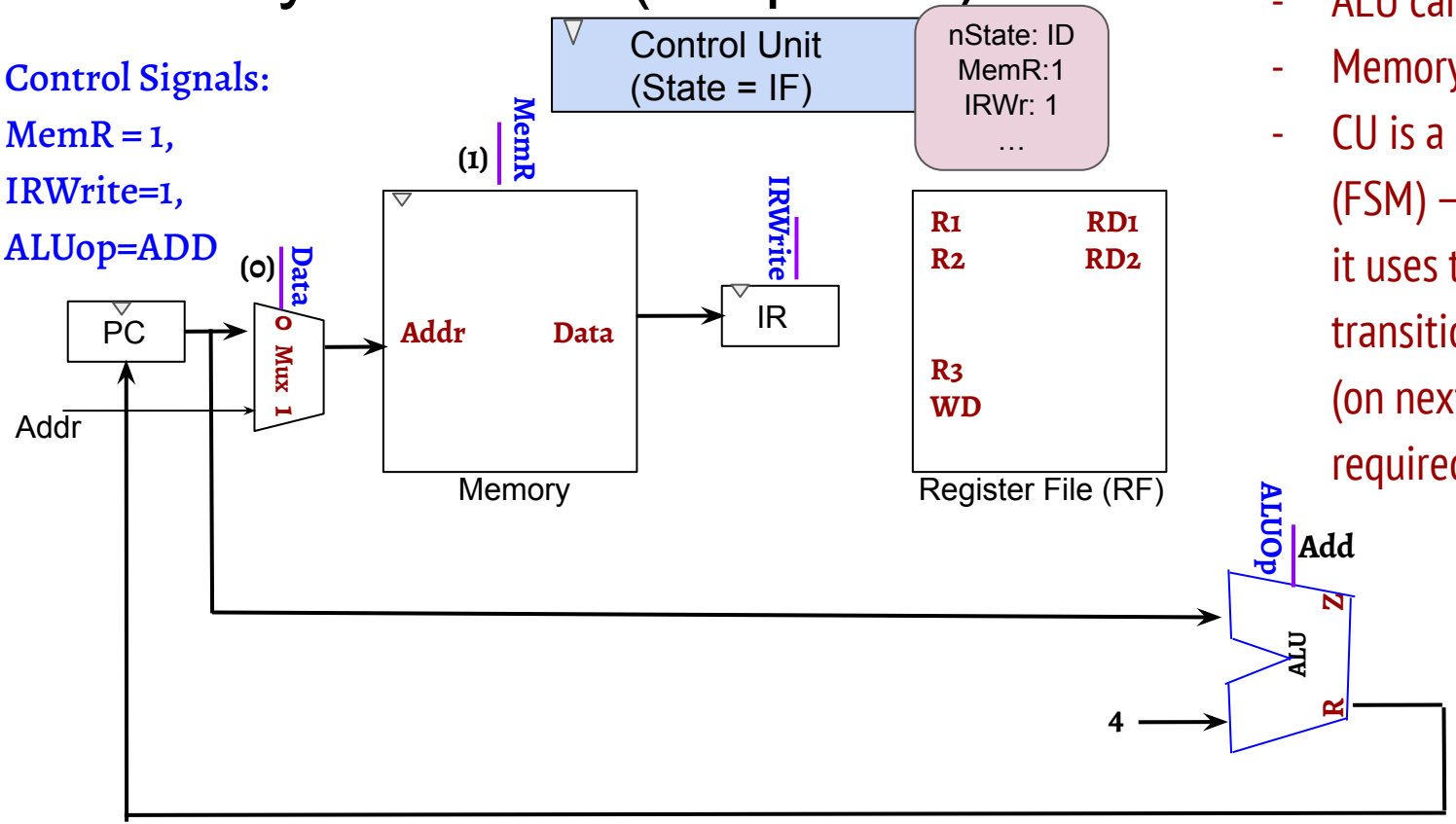
- All steps of instruction execution are performed in a multiple (**K**) clock cycles
- Software visible state transition (**S** → **Q**) in **K** clock cycles
- Intermediate (software invisible) state between different steps.
- Example (with 5 steps/cycles)



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

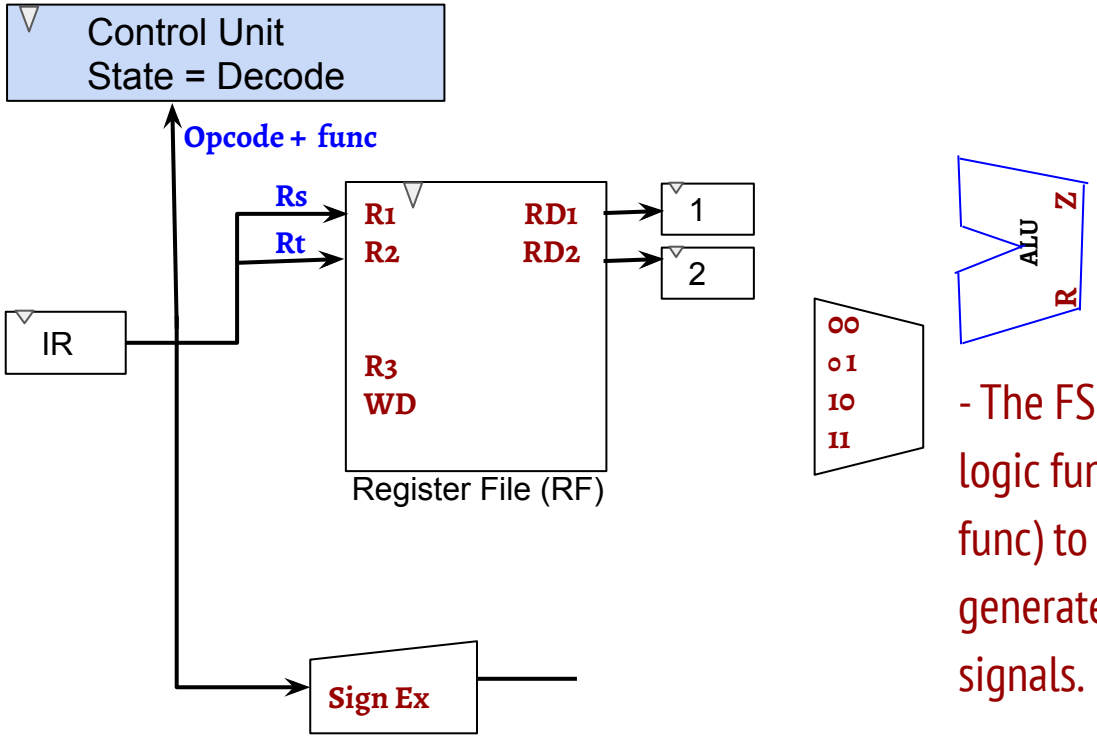
Multi-cycle Fetch (simplified)

Control Signals:
MemR = 1,
IRWrite=1,
ALUop=ADD



- 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

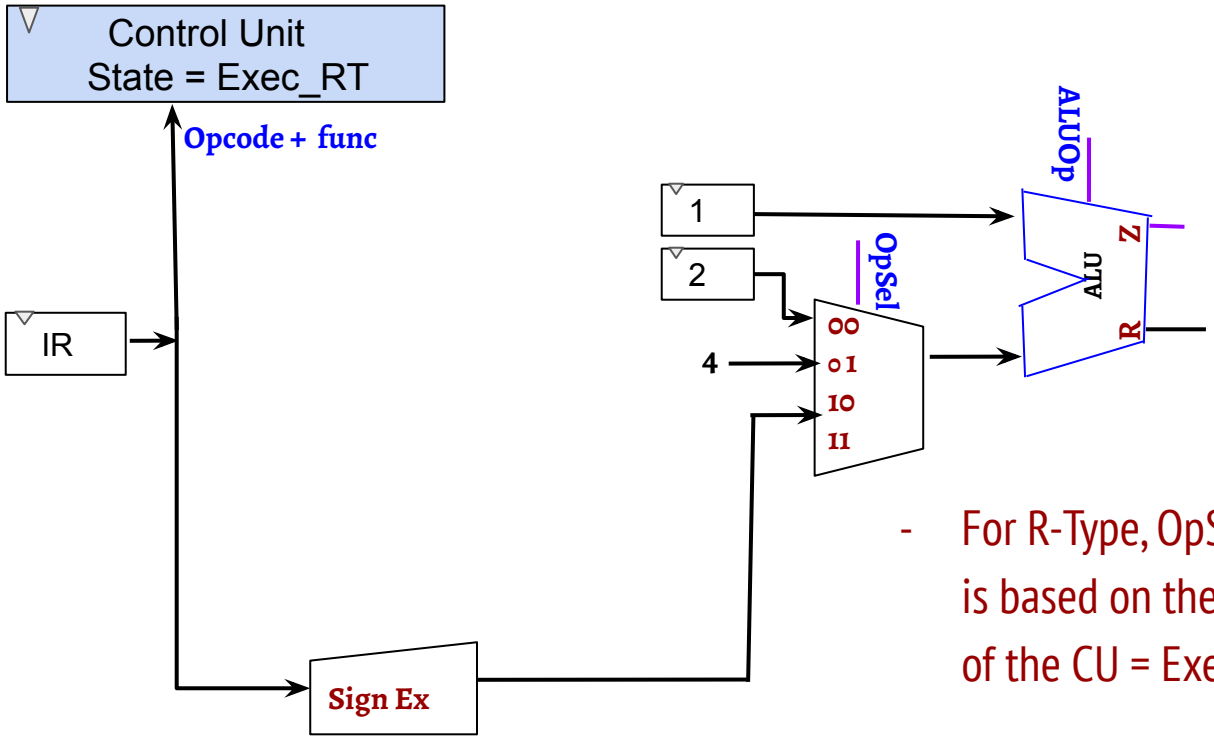
Multi-cycle Decode/RA



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

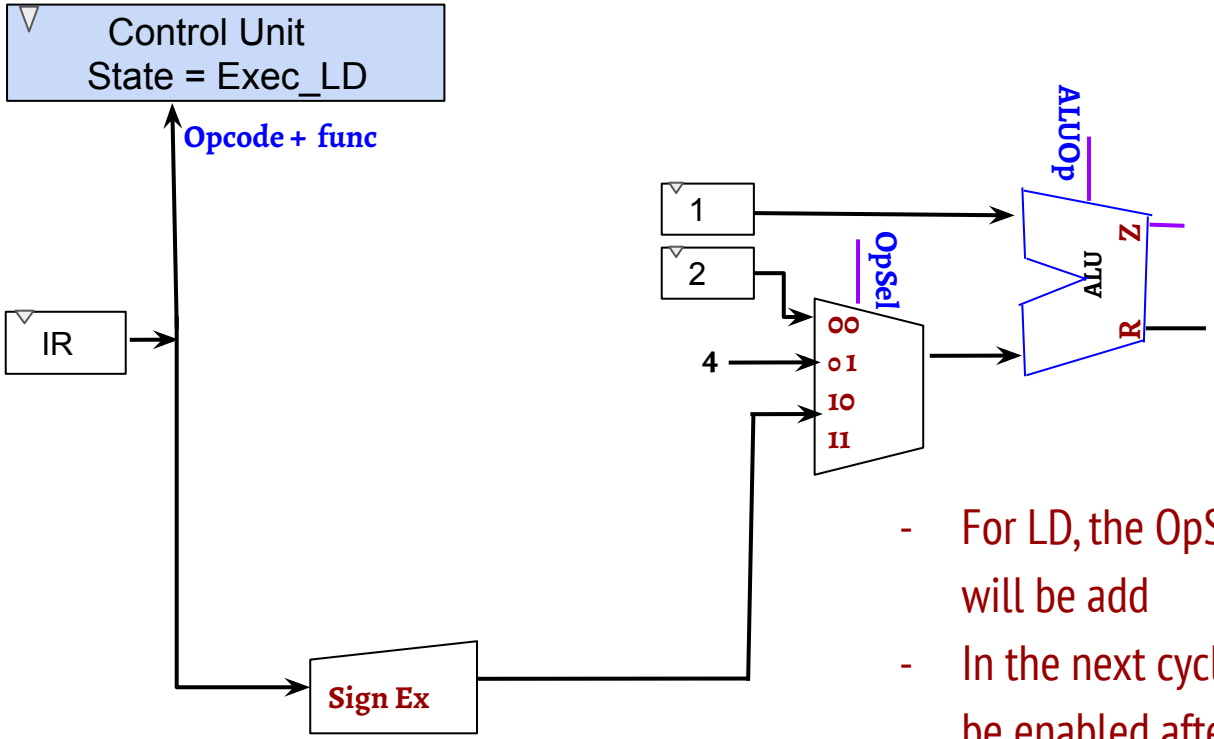
- Operand registers (1 and 2) can be read in next cycle (part of the state)

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 need not use MAX cycles: R-Type and ST: 4 cycles, Branch: 3 cycles, LD: 5 cycles
- Disadvantages?
    - Even with reuse, functional units remain under-utilized

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