Lecture 10

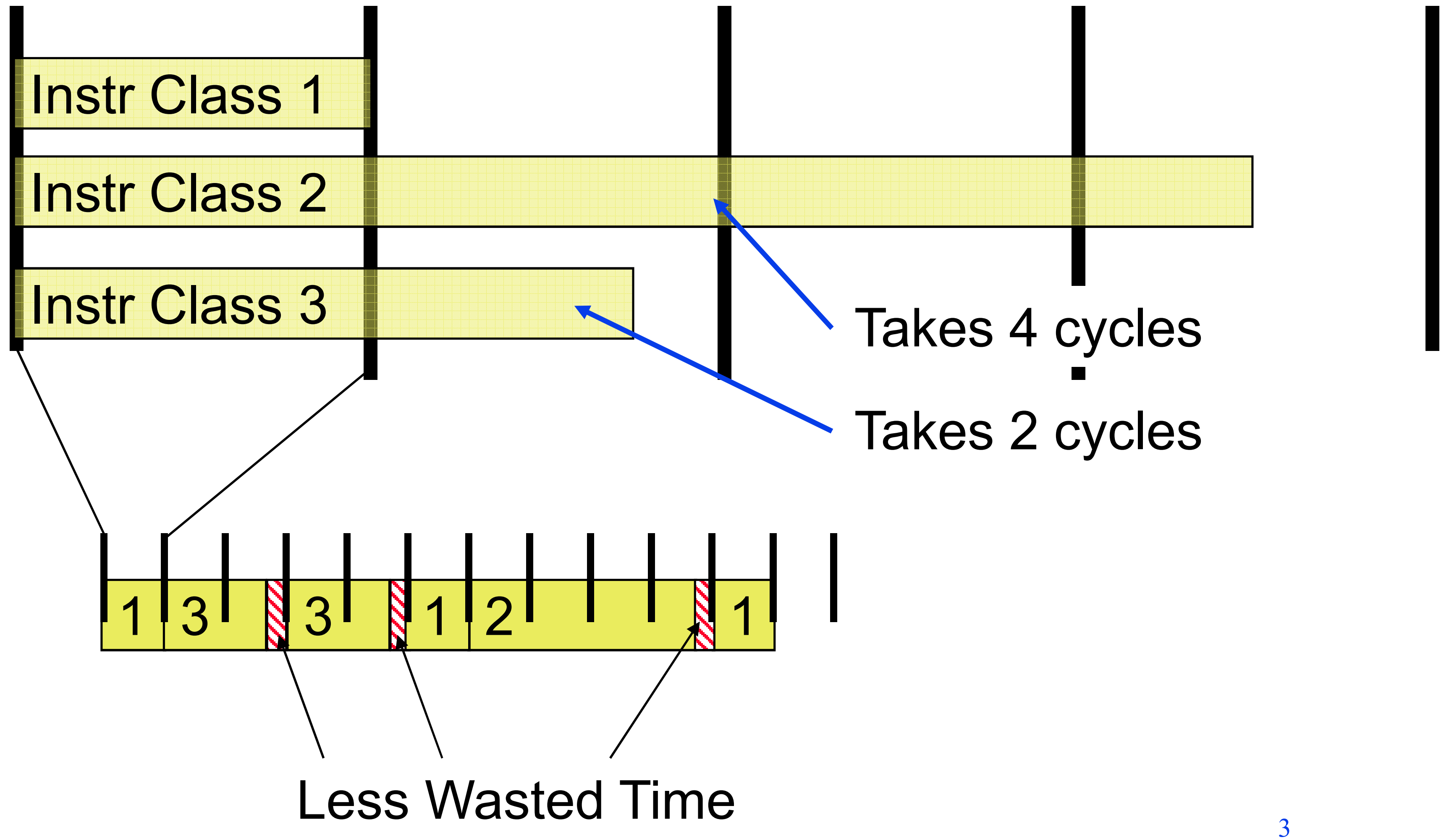
Multi-Cycle Implementation

Today's Menu

- **Multi-Cycle machines**
  - Why multi-cycle?
  - Comparative performance
  - Physical and Logical Design of Datapath and Control
  - Microprogramming

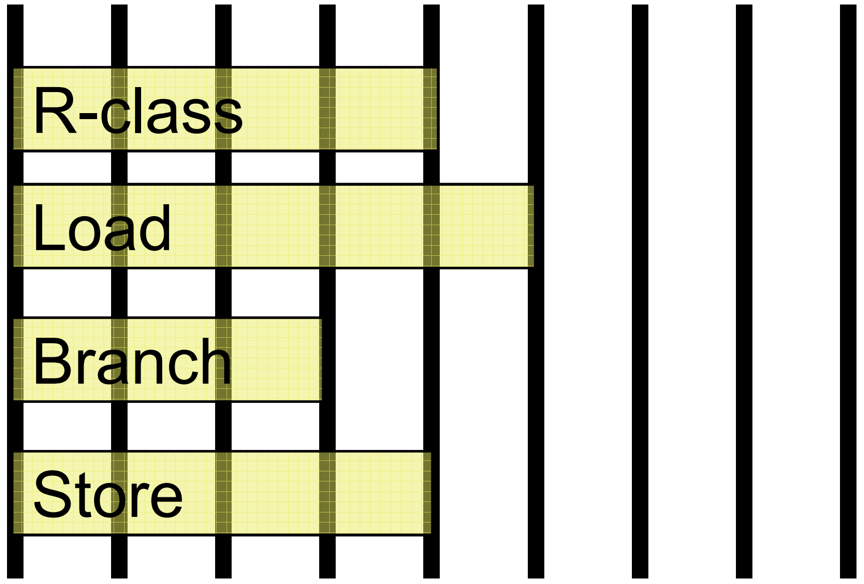
Multi-cycle Solution

Idea: Let the FASTEST instruction determine clock period

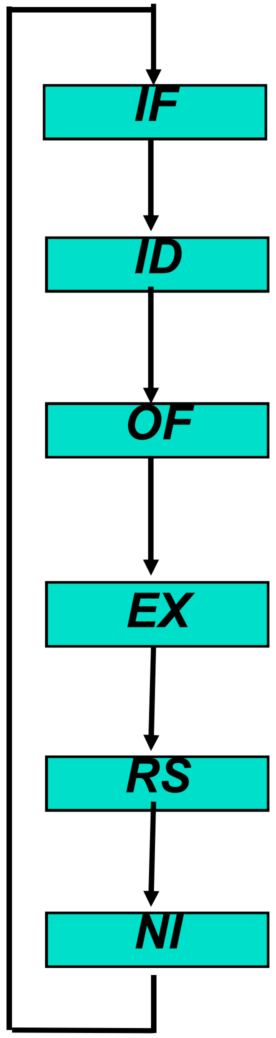


Multi-cycle Reality

- We are going to go further than allowing the fastest instruction to determine rate
- We are going to break EVERY instruction up into phases



MIPS Architecture Instruction Classes



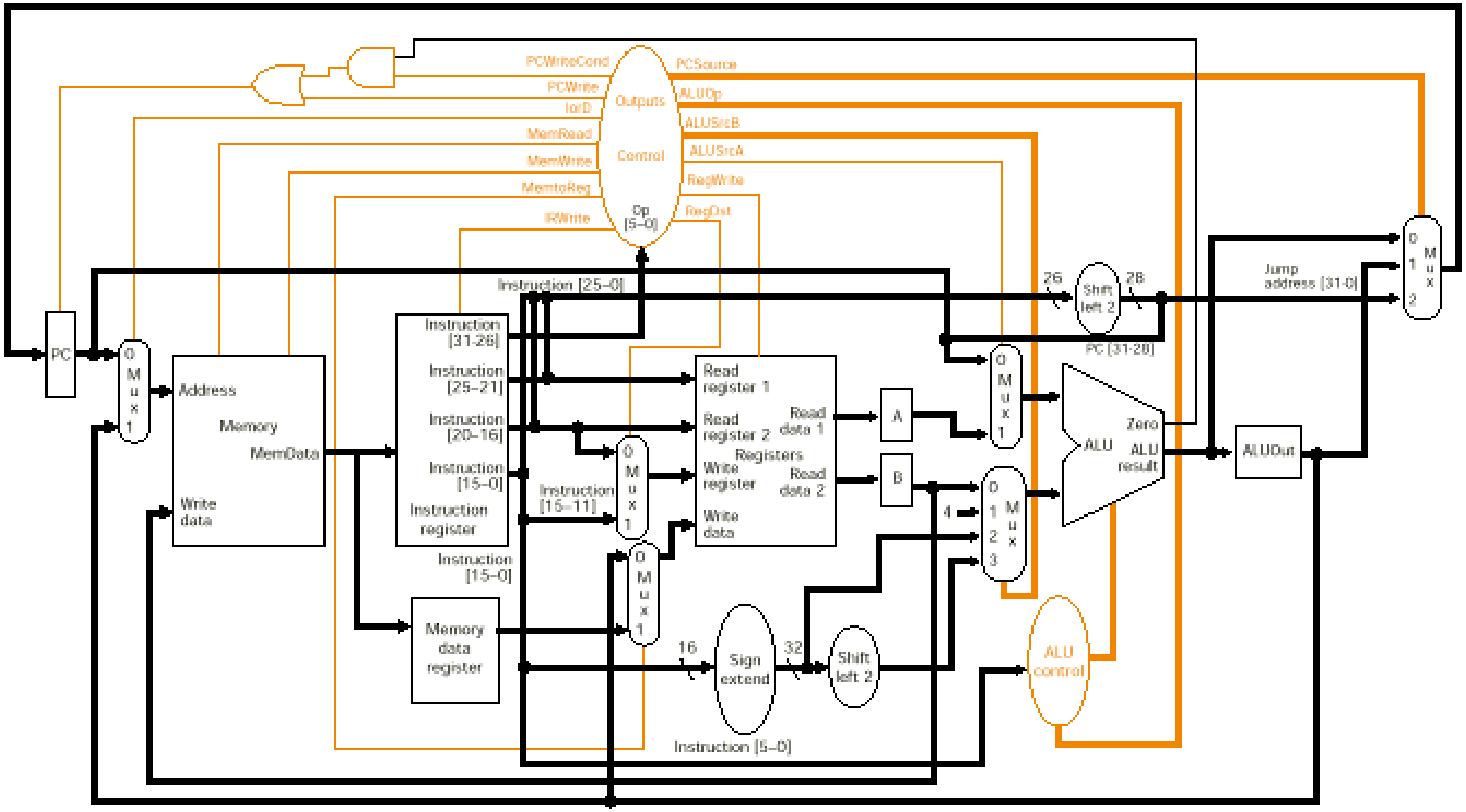
	ALU	Load	Store	Branch	Jump
IF	mem rd	mem rd	mem rd	mem rd	mem rd
ID	op code operands	op code operands	op code operands	op code operands	op code operands
OF	1 or 2 reg rd (immd)	2 reg rd immd	2 reg rd immd	2 regs rd PC, immd	PC, immd
EX	op	addr gen mem rd	addr gen mem wr	cond gen addr gen	addr gen
RS	reg wr	reg wr			
NI	PC+4	PC+4	PC+4	if cond; update PC	update PC

Note That…

- **Instruction decode and register read must be done for all classes of instructions**
- **PC+4 can be done right after fetch**
- **Address generation for Jump can be performed during the decode step**
- **The same adder can be shared among:**
  - Instruction fetch logic (PC = PC + 4)
  - Address generation logic (Address = A + IR[15:0])
  - Branch target calculation (Next PC = (PC + 4) + IR[15:0])
  - ALU operations (ALUOutput = A + B)
- **Same memory port can be used to access instructions and data**

Full Diagram of Multi-cycle Machine

- Figure 5.33



Recall MIPS Instruction Formats…

R-type Instructions

31 - 26	25-21	20-16	15-11	10-6	5-0
op	rs	rt	rd	shamt	funct

Load or Store

31 - 26	25-21	20-16	15-0
35 or 43	rs	rt	immediate (address)

Branch

31 - 26	25-21	20-16	15-0
4	rs	rt	immediate (address)

Jump

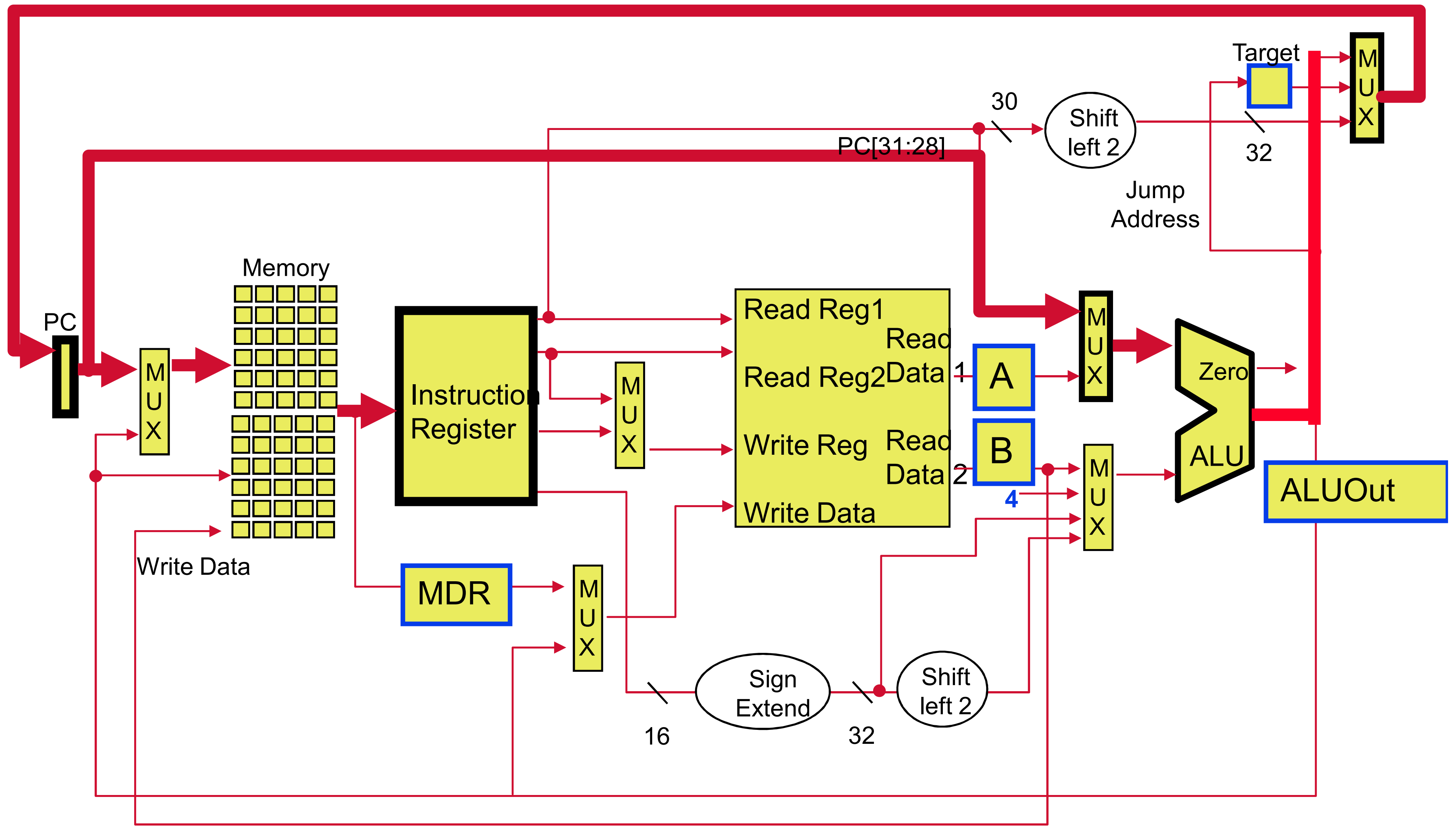
31 - 26	25-0
2	immediate (address)

Execution Steps (1)

- **Instruction Fetch**

```
IR = Memory[PC];
PC = PC + 4;
```

Fetch



Target
M U X
30
Shift left 2
PC[31:28]
32
Jump Address
Memory
PC
M U X
Instruction Register
M U X
Read Reg1
Read Data 1
Read Reg2
Write Reg
Read Data 2
Write Data
A
B
4
M U X
Zero
ALU
M U X
ALUOut
Write Data
MDR
M U X
Sign Extend
16
32
Shift left 2

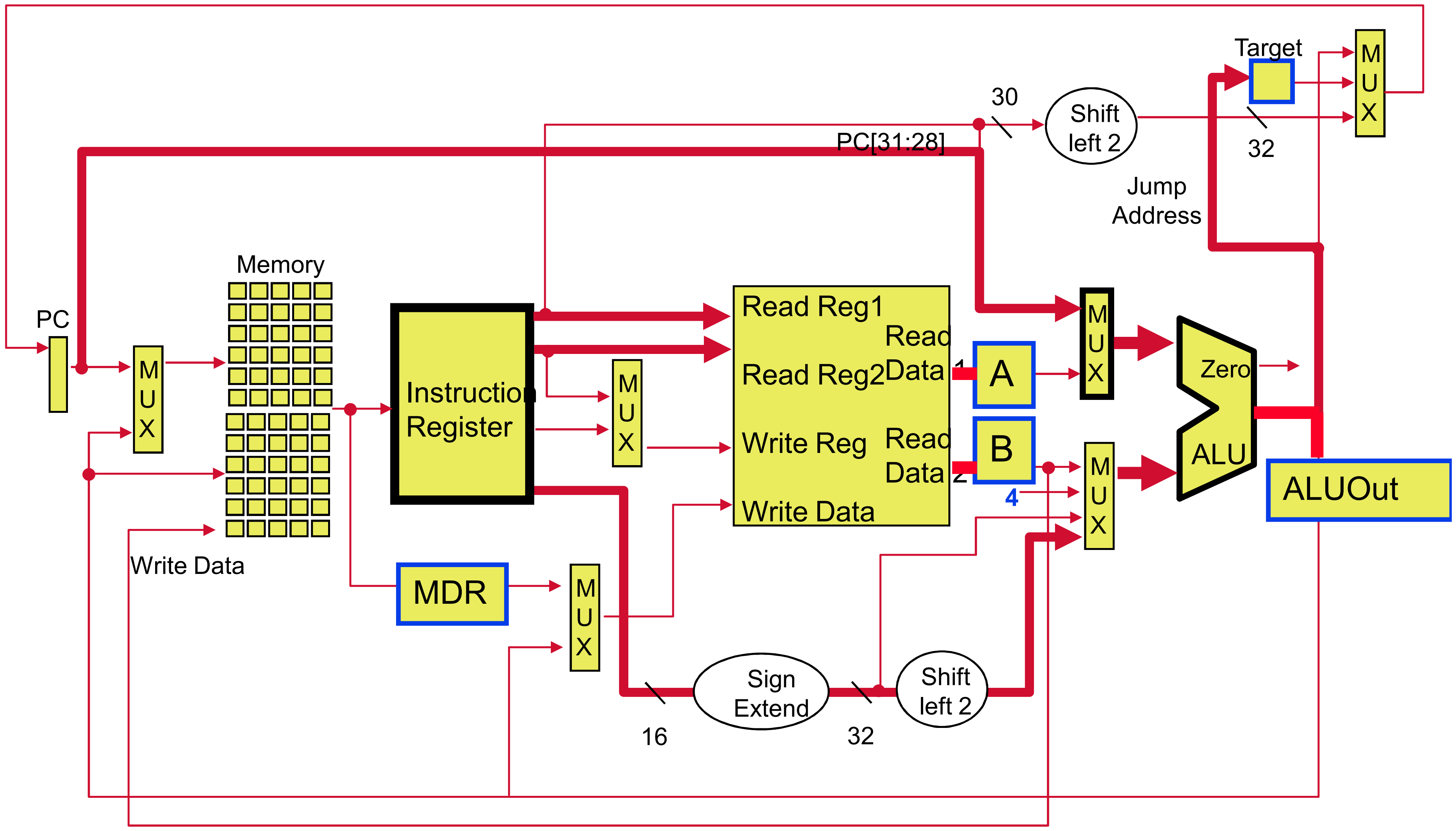
Execution Steps (2)

- Instruction Decode and Register Fetch

```
A = Reg[IR[25..21]];
B = Reg[IR[20..16]];

Target = PC + 4 + (signExtend(IR[15..0])  << 2);
```

Decode and Register Op Fetch



Execution Step (3)

- **Execution, memory address computation or branch completion**
  - Memory Reference

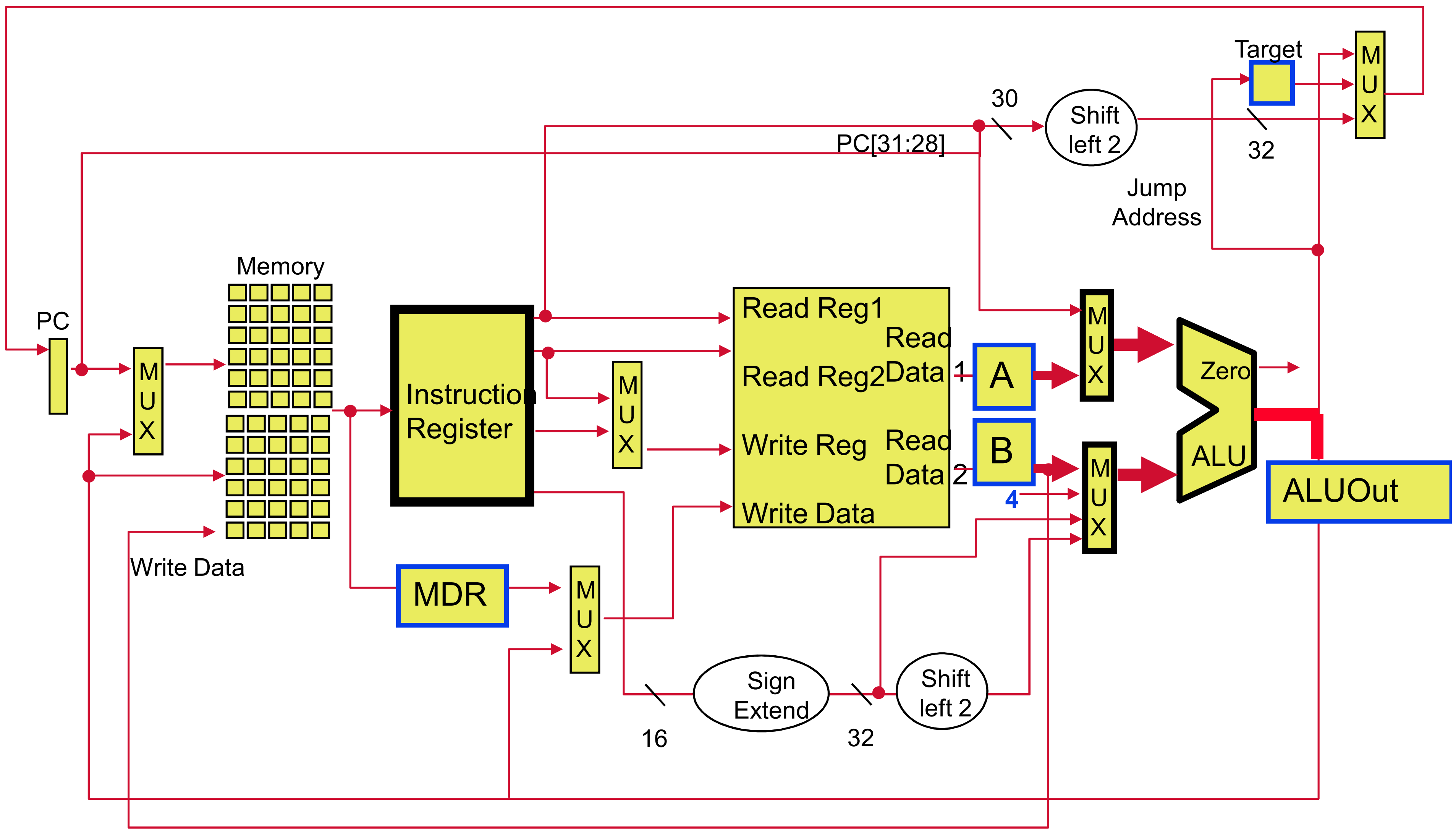
    ALUOut = A + signExtend(IR[15..0]);
  - Arithmetic/Logical Operation

    ALUOut = A + B
  - Branch

    If (A == B) PC = Target;
  - Jump

    PC = PC[31 ..28] || (IR[25..0) << 2);

Execute (R-type)



Execution Step (4)

- **Memory access or R-type instruction completion**
  - Memory Reference

    MDR = Memory[ALUOut];

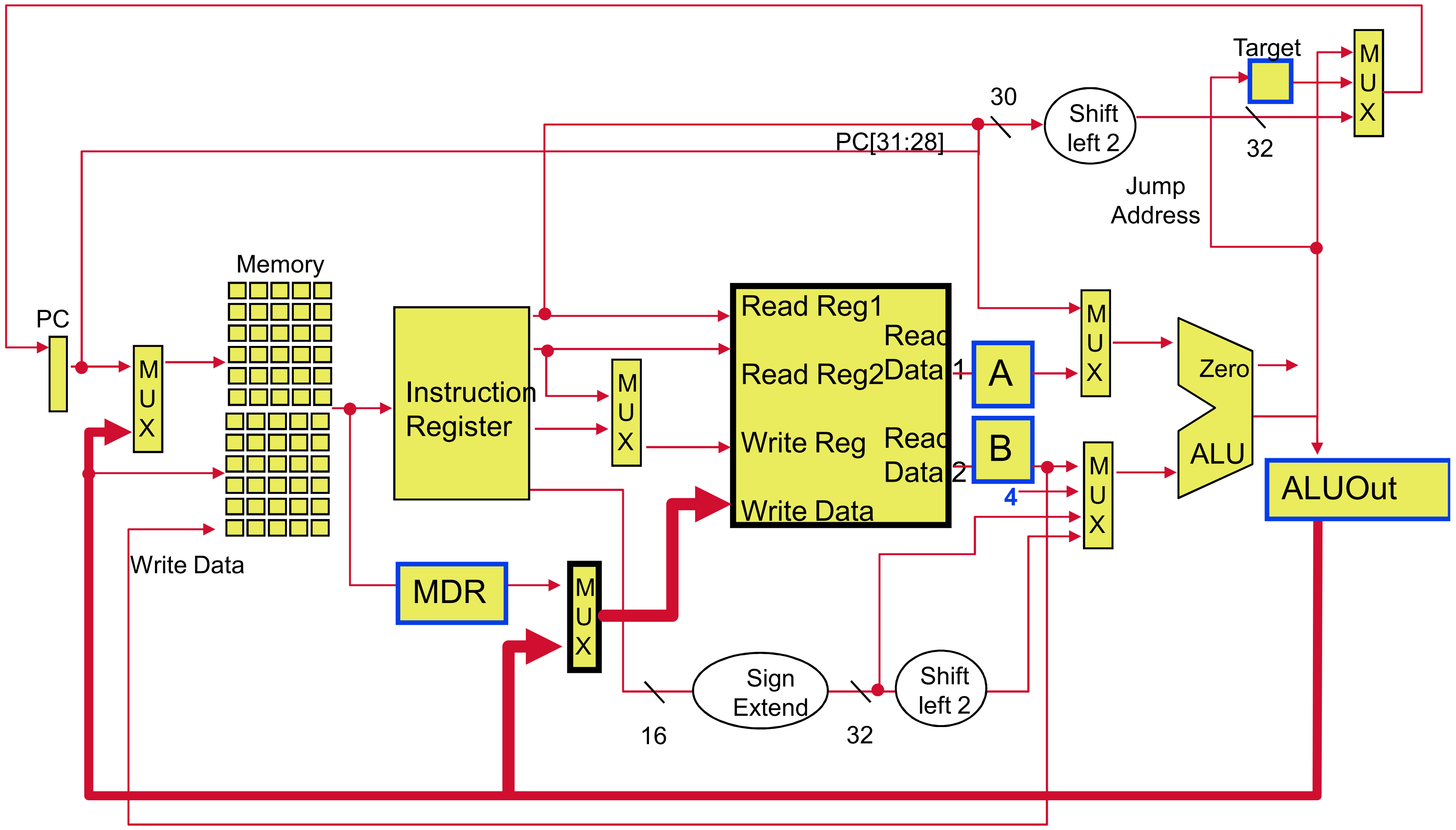
    or

    Memory[ALUOut] = B;
  - Arithmetic/Logical Instructions (R-type)

    Reg[IR[15..11]] = ALUOut;
  - Branch, Jump

    Nothing

Execute (R-type)

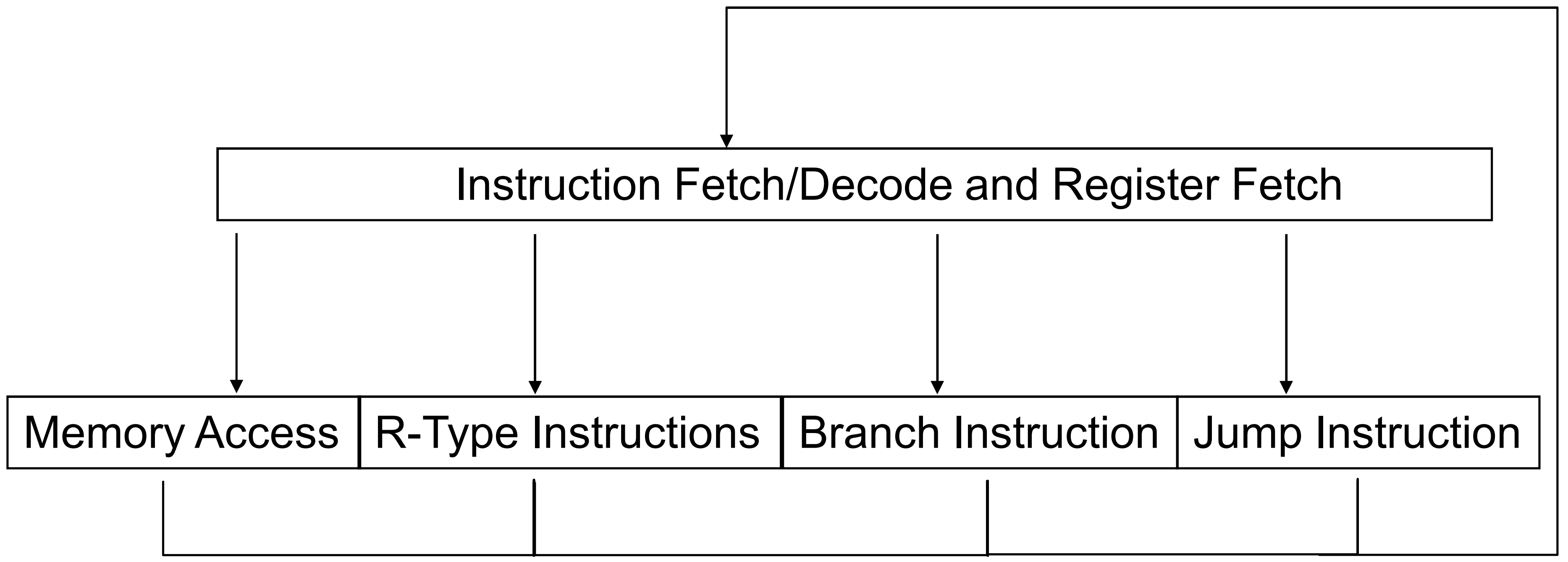


Execution Step (5)

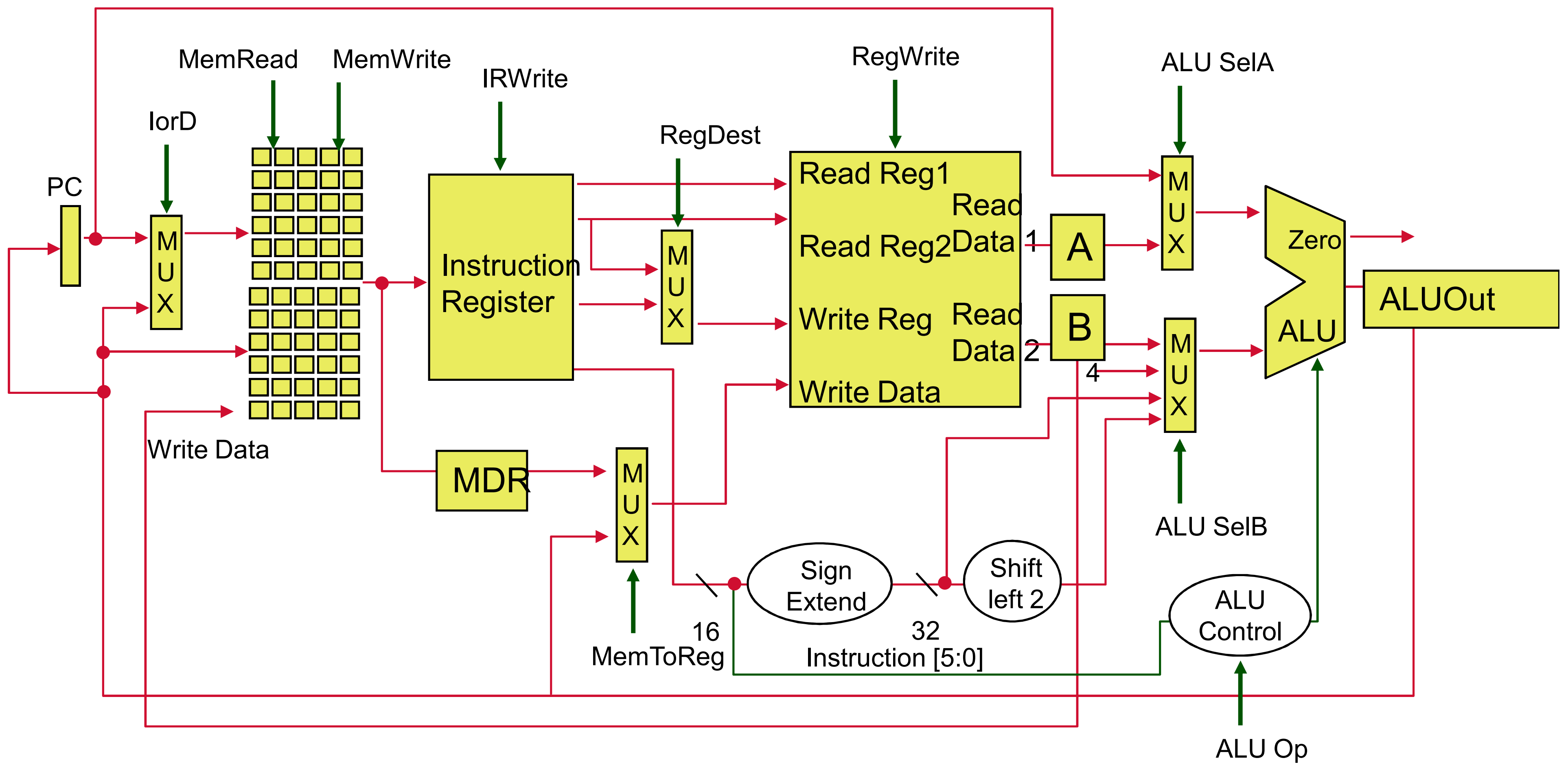
- **Memory Read completion (Load only)**

  Reg[IR[20..16]] = MDR;

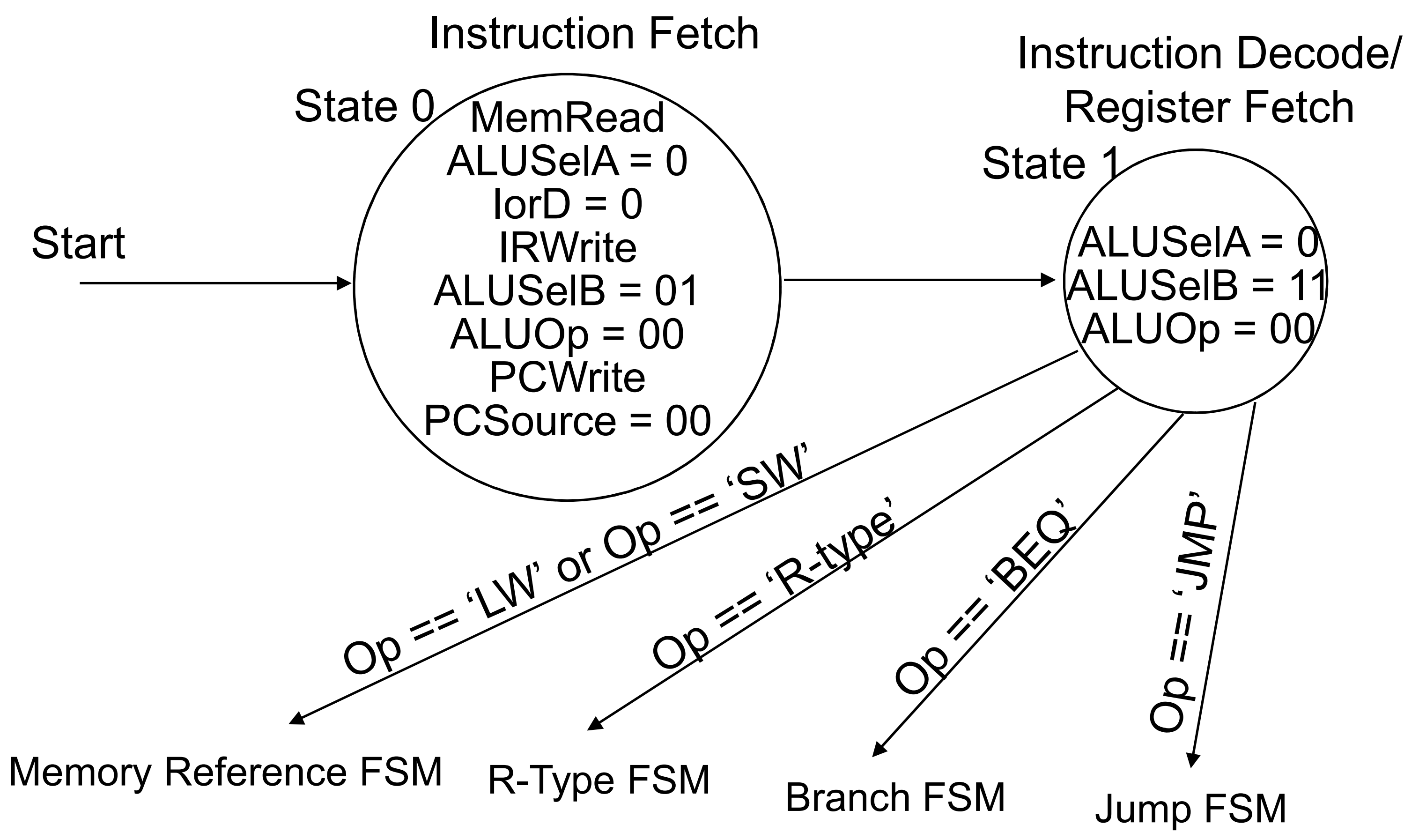
Finite State Machine Control



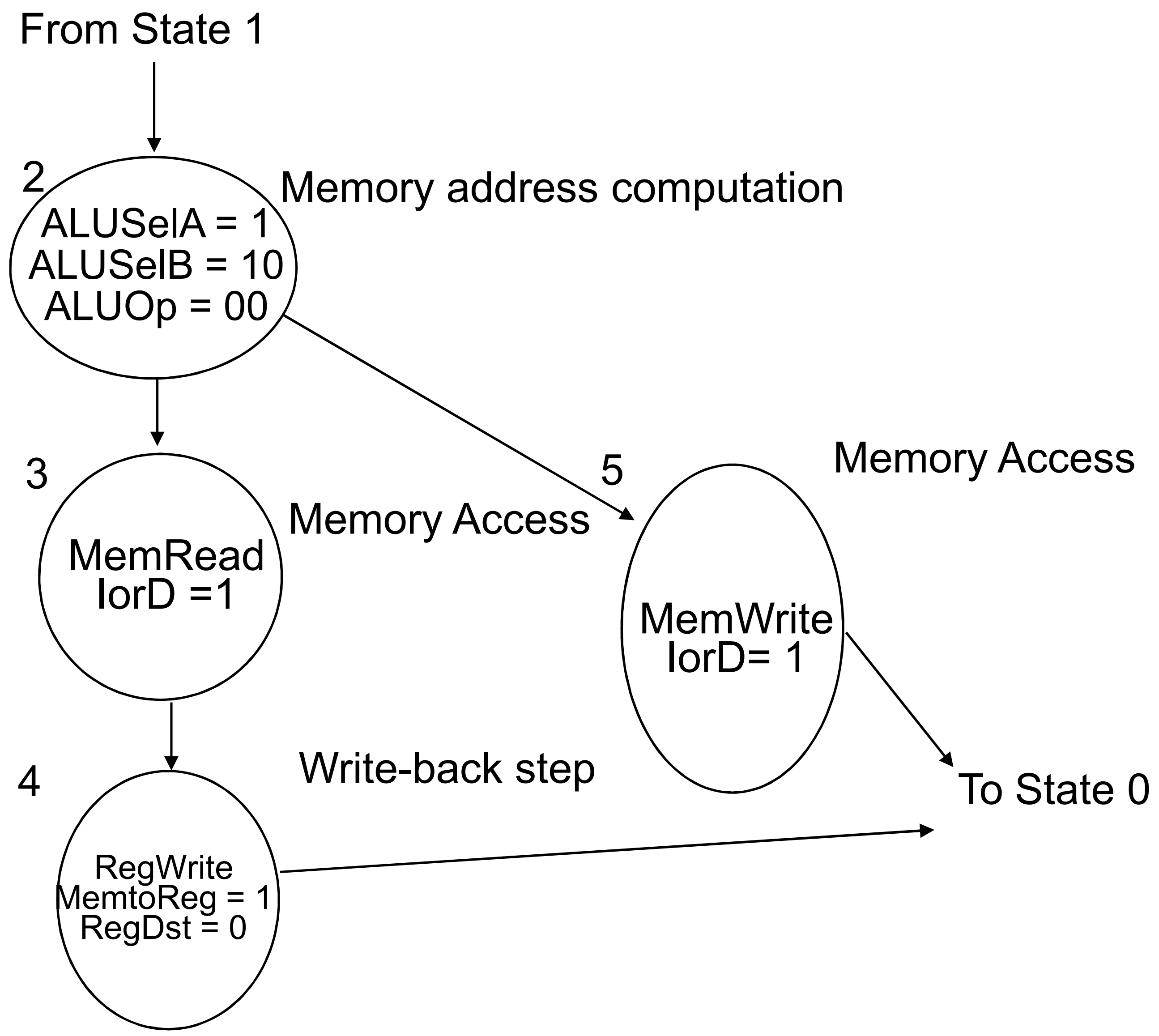
Multicycle Control



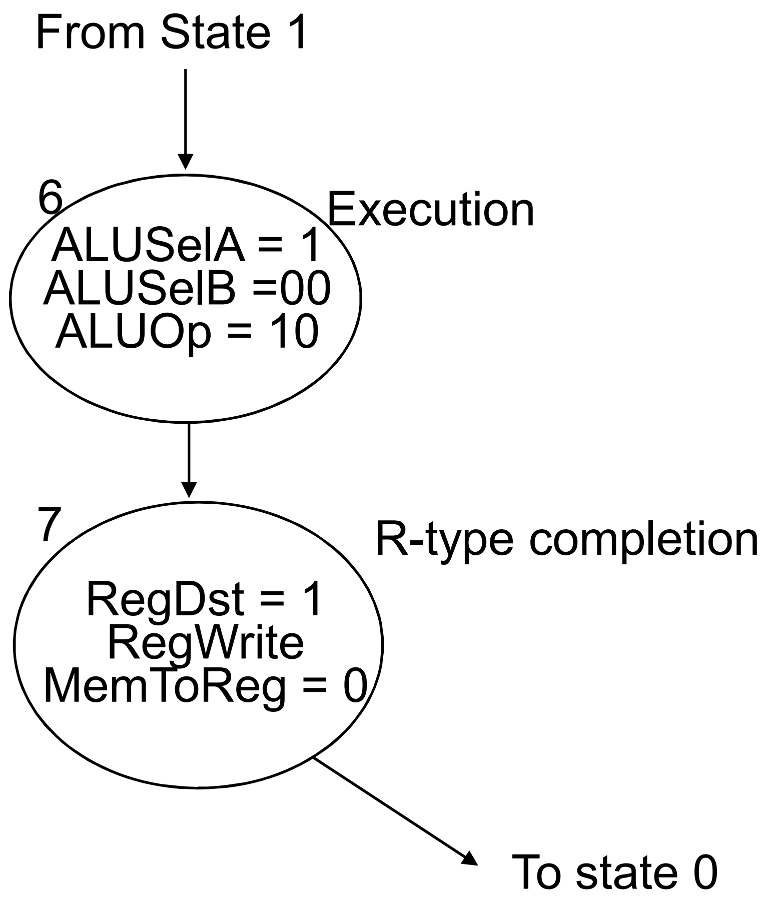
Instruction Fetch and Decode



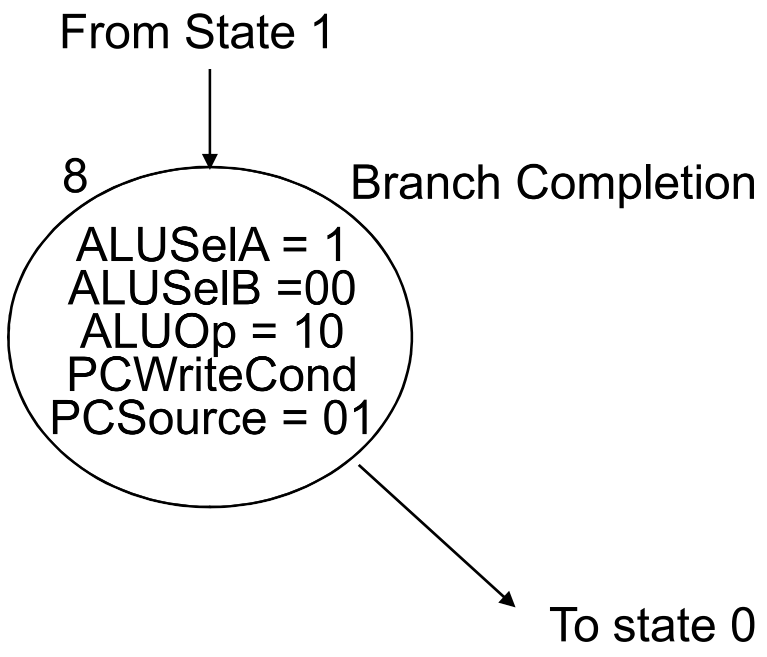
Memory-Reference FSM



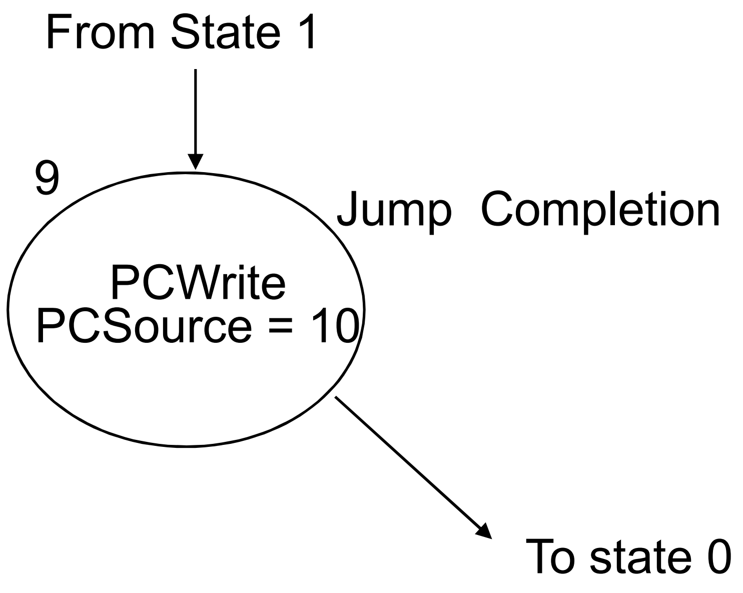
R-type FSM


From State 1
6
ALUSelA = 1
ALUSelB =00
ALUOp = 10
Execution
7
R-type completion
RegDst = 1
RegWrite
MemToReg = 0
To state 0

Branch FSM

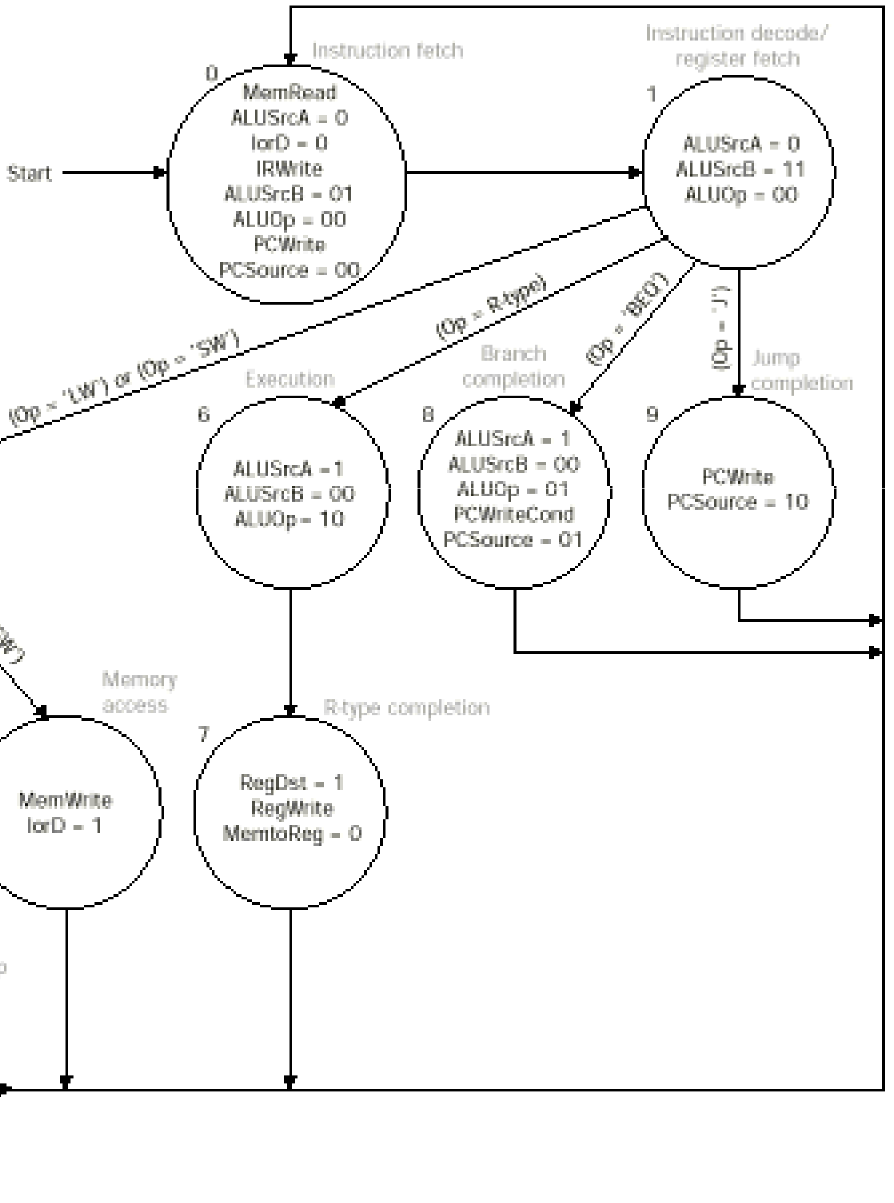

From State 1
8
Branch Completion
ALUSelA = 1
ALUSelB =00
ALUOp = 10
PCWriteCond
PCSource = 01
To state 0

Jump FSM

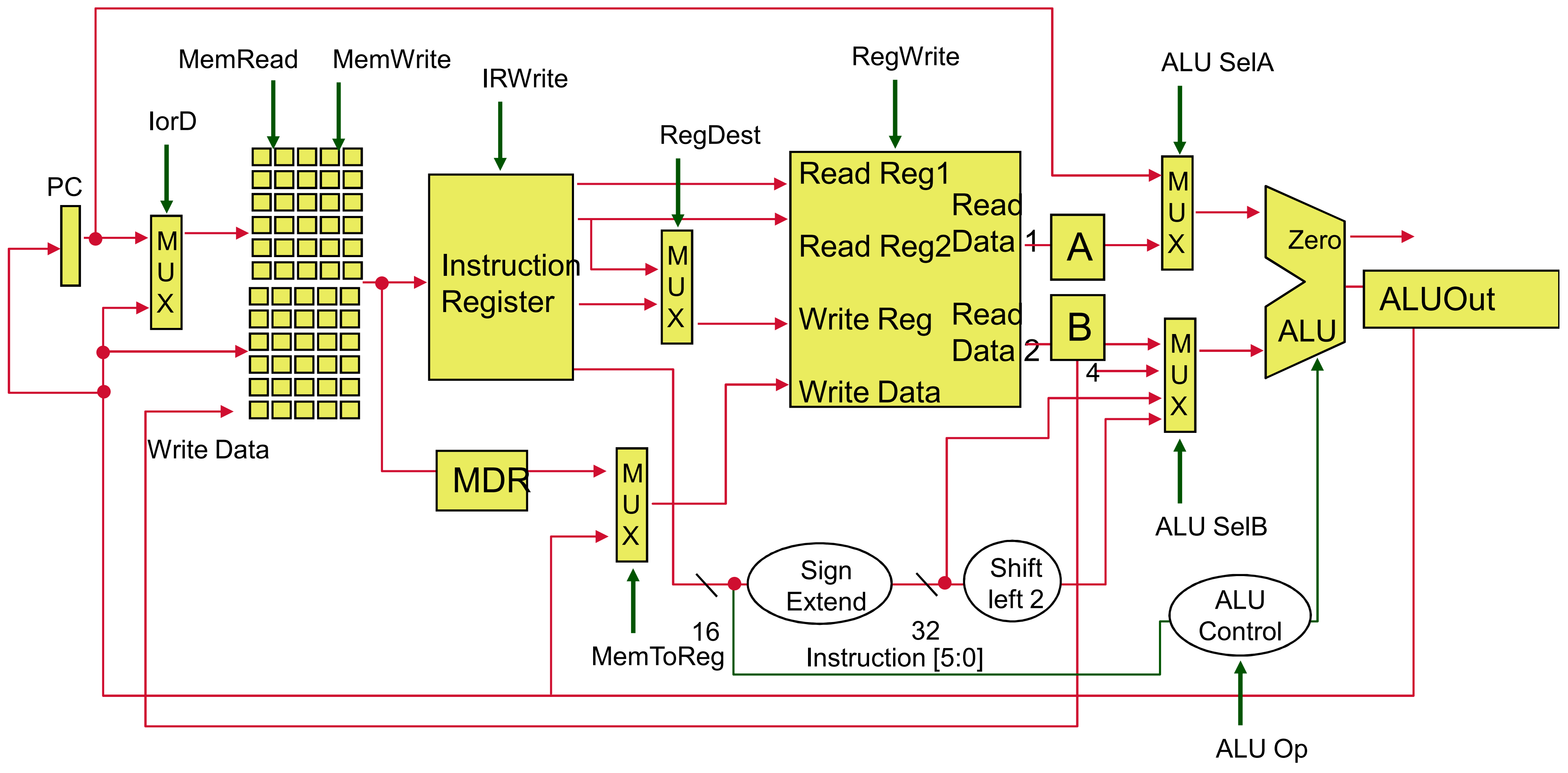

From State 1
9
Jump Completion
PCWrite
PCSource = 10
To state 0

Complete State Diagram

- **Figure 5.42 from book**



Multicycle Control



Performance of Multicycle Implementation

- **Each type of instruction can take a variable # of cycles**
- **Example**
  - Assume the following instruction distributions:

loads	5 cycles	22%
stores	4 cycles	11%
R-type	4 cycles	49%
branches	3 cycles	16%
jump	3 cycles	2%

  - What's the average Cycles Per Instruction (CPI)

    CPI = (CPU clock cycles/Instruction Count)

    CPI = (5 cycles * 0.22) + (4 cycles * 0.11) + (4 cycles * 0.49)
    + (3 cycles * 0.16) + (3 cycles * 0.02)

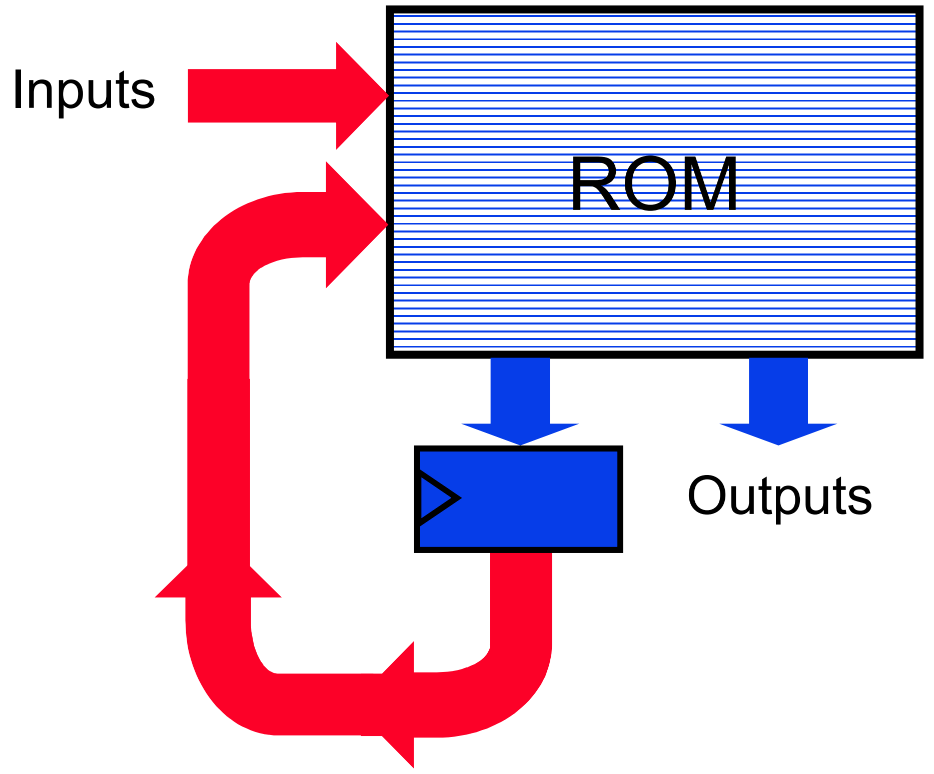
    CPI = 4.04 cycles per instruction
  - What was the CPI for the single-cycle machine?
    - Single cycle implies 1 clock cycle per instruction --> CPI = 1.0
    - So isn't the single-cycle machine about **4** times faster?

Performance of Multicycle Implementation

- **The correct answer should consider the clock cycle time as well:**
  - For the single cycle implementation, the cycle time is given by the worst case delay: T_{cycle} = 40ns (for load instructions, see slide 8)
  - For the multicycle implementation, the cycle time is given by the worst case delay over all execution steps: T_{cycle} = 10 ns (for each of the steps 1, 2, 3, or 4).
- **The execution time per instruction is:**
  - CPI * T_{cycle} = 40 * 1 = 40 ns per instruction for the single cycle machine
  - CPI * T_{cycle} = 10 * 4.04 = 40.4 ns per instruction for the multicycle machine
  - Thus, the single cycle machine is **only 1% faster**
- **When considering other types of units (e.g., FP), the single cycle implementation can be** very **inefficient.**

Microcode: Another Approach

- Another way to implement a Mealy machine:



N: Inputs

X: Outputs

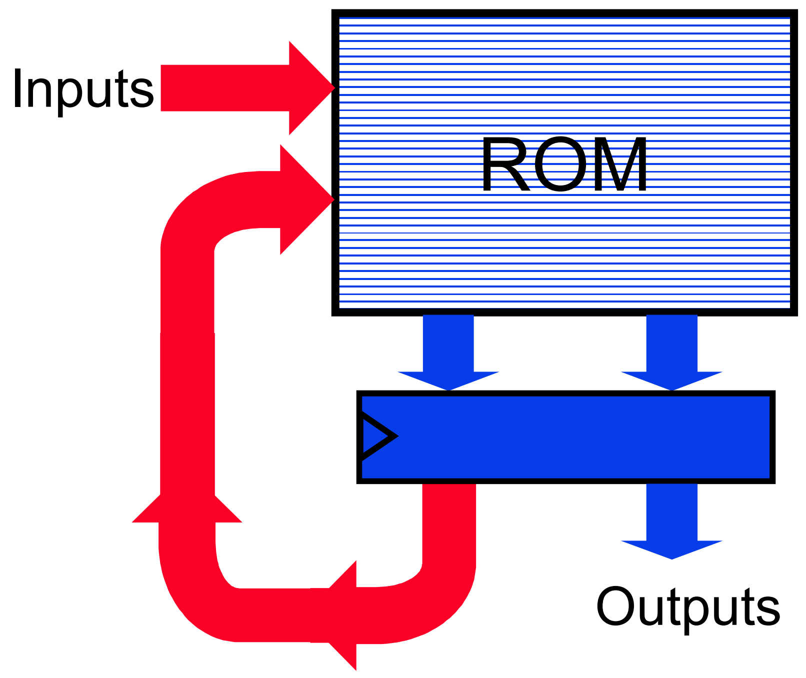
S: State Bits

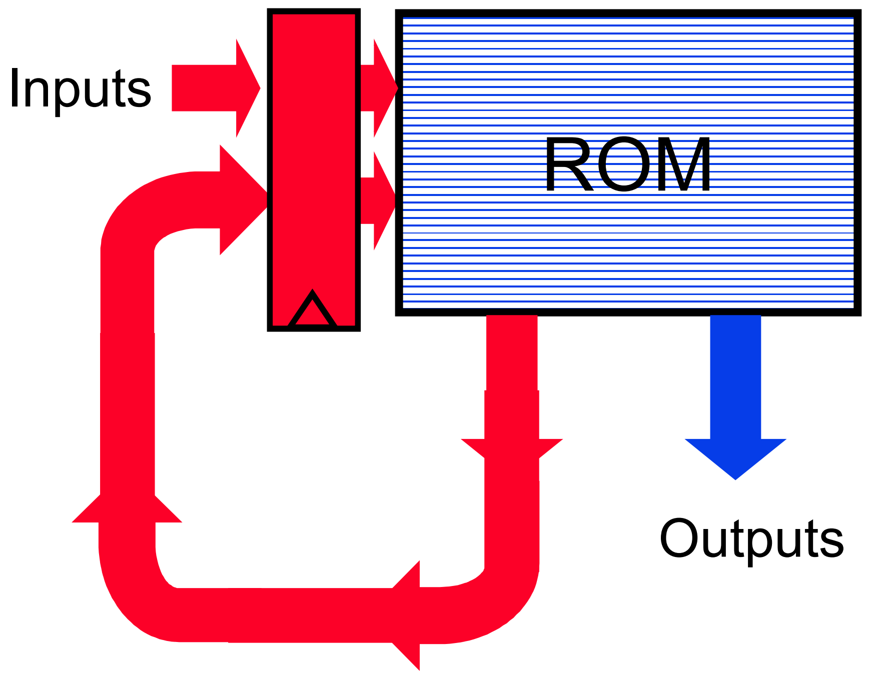
Storage:

$X + S$ (bits/word)

2^{N+S} (words)

Microcode II: Moore Machines





So who cares?

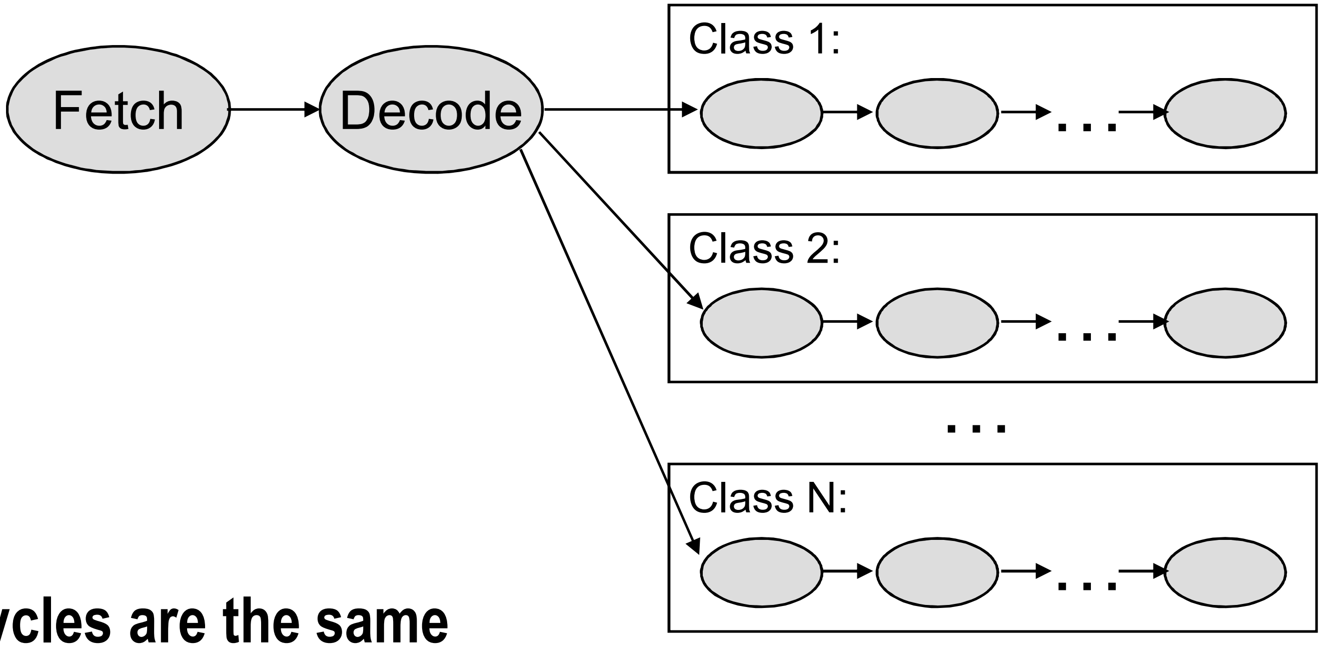
- **Imagine:**
  - Hundreds of instructions…
  - Tens of different instruction classes (we’ve seen four)
  - Instructions that take anywhere from 1 to 100 cycles to complete

  - That’s what any real ISA has

- **Now imagine drawing the FSM diagram for that!**

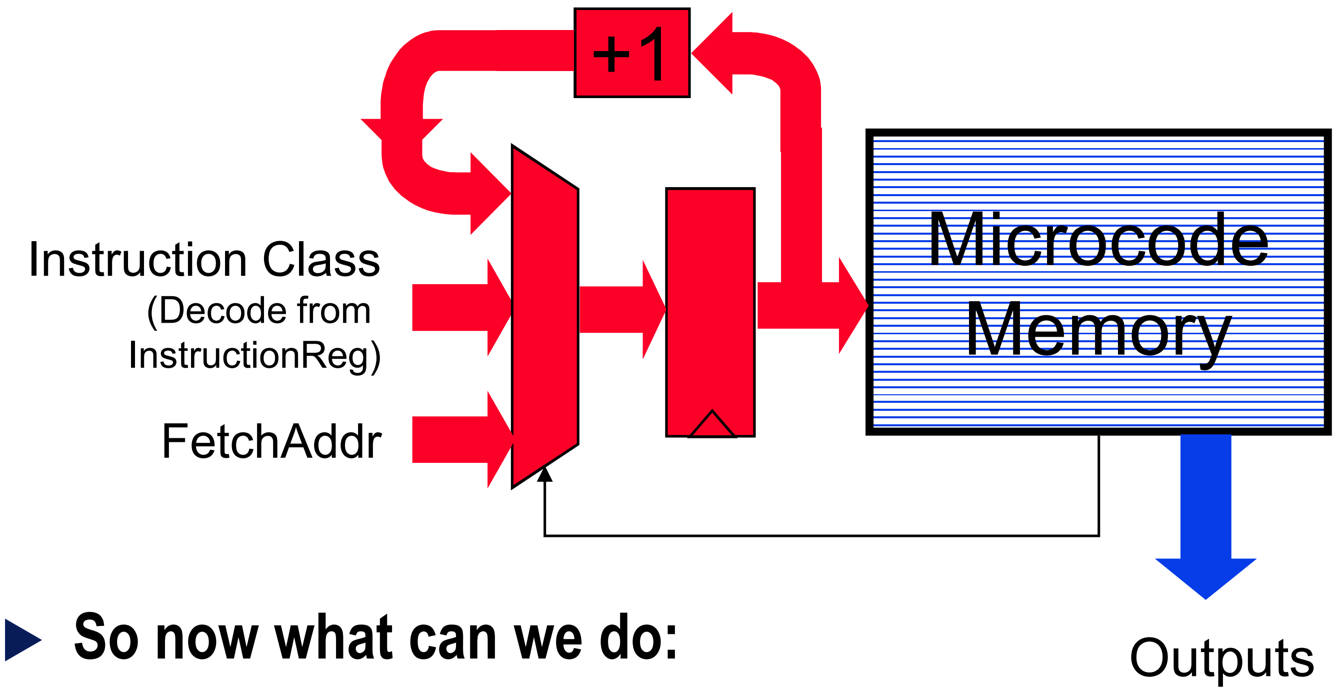
- **Solution 1: Write Verilog and use synthesis (today)**
- **Solution 2: Use some programming lessons**

FSM structure



- First cycles are the same
- No reconvergence after decode
- Limited Branching

Exploiting the structure for microcode



- **So now what can we do:**
  - Go to instruction class address
  - Go to the next word in the memory
  - Go back to the fetch address

  - Control everything by assigning bits in the word to control signals

Microcode Word Definition

ALU Control:	Add, Subt, Func code
SRC1:	PC, A
SRC2:	B, 4, Extend, Extshift
Register Control:	Read, Write ALU, Write MDR
Memory:	Read PC, Read ALU, Write ALU
PCWrite control:	ALU, ALUOut-cond, JumpAddress
Sequencing:	Seq, Fetch, Dispatch

- Total Word Size >= 13

Why is this nice?

- **Reduce complexity of control design**
  - Only way to do it before synthesis tools
  - Only way to encode Complex Instruction Sets
- **Allows bug fixes, optimizations after real hardware**
- **Now how do people do this today?**
- **Specify Style for:**
  - Registers
  - Latches
  - Combinational Logic
  - Finite State Machines

And Never Forget Marketing

- It is easy to sell a 1GHz processor
- It is harder to sell “Yes our processor is only 500MHz, but it actually achieves a lower CPI, and therefore really is higher performance.”

Other Multi-cycle Data Paths

- **Single cycle datapaths are strongly implied by ISA**

- **Is this true of multi-cycle implementations?**
  - NO!
  - There are a lot of possible data paths

Exceptions and Interrupts

- Exceptions are 'exceptional events' that disrupt the normal flow of a program
- Terminology varies between different machines
- Examples of Interrupts
  - User hitting the keyboard
  - Disk drive asking for attention
  - Arrival of a network packet
- Examples of Exceptions
  - Divide by zero
  - Overflow
  - Page fault

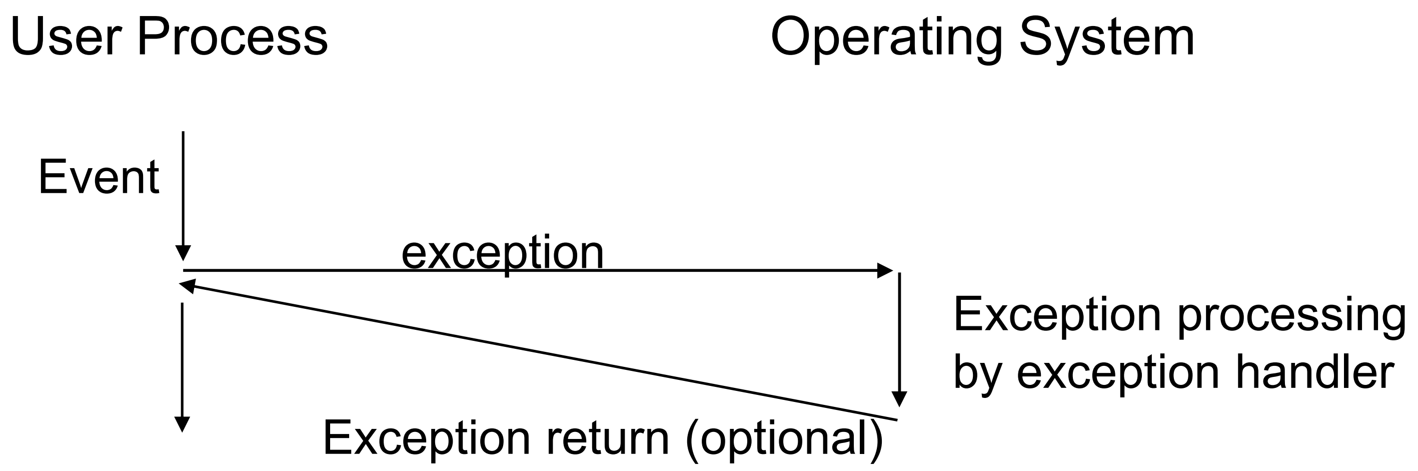
Handling Exceptions and Interrupts

- **When do we jump to an exception?**

- **Upon detection, invoke the OS to "service the event"**
  - Right when it occurs?
  - What about in the middle of executing a multi-cycle instruction
    - Difficult to abort the middle of an instruction
  - Processor checks for event at the end of every instruction
  - Processor provides EPC & Cause registers to inform OS of cause
    - EPC - Exception Program Counter
      - Holds PC that the OS should jump to when resuming execution
    - Cause Register
      - Holds bit-encoded cause of the exception

Exception Flow

- **When an exception (or interrupt) occurs, control is transferred to the OS**


User Process
Operating System
Event
exception
Exception processing by exception handler
Exception return (optional)

Summary

- Single cycle implementations have to consider the worst case delay through the datapath to come-up with the cycle time.
- Multicycle implementations have the advantage of using a different number of cycles for executing each instruction.
- In general, the multicycle machine is better than the single cycle machine, but the actual execution time strongly depends on the workload.
- The most widely used machine implementation is neither single cycle, nor multicycle – it's the pipelined implementation.