UMASS AMHERST

ECE 331

Hardware Organization and Design

Lesson 16

Chapter 4 – The Processor

Sections: 4.4, 4.5

Announcement 1

- **My office hours:**
  - ➢Tu-Th: 11:00am – 12:00Noon
  - ➢KEB 3rd floor conference room (KEB 309)

- **TAs' office hours:**
  - ➢Mon, Tue, Wed, Thu: 5:30pm – 7:00pm
  - ➢Location: HAS0136 (Hasbrouck Lab)

- **Week 9 HW assignment on Moodle – Due Sunday 11/05/2023 – 11:59pm**

Last Lessons Summary

- **Understanding Processor design steps**
  - ➢ Datapath
  - ➢ Control

Lesson Objectives

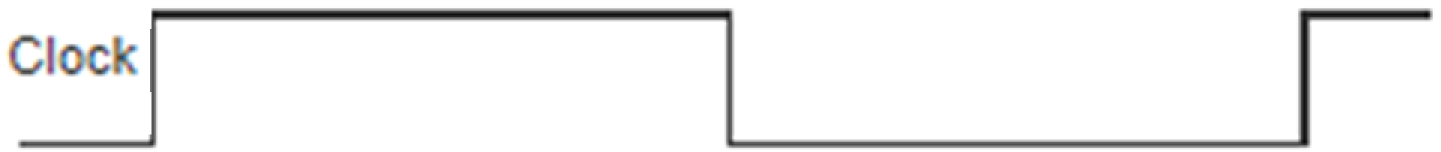
- **Understanding Processor Implementation**
  - A simple implementation scheme:
    - ➢Single Cycle Implementation
  - More complex Implementation:
    - ➢A multicycle implementation
- **With Datapath in place, let’s focus on Control design**

Logic Design Basics

- **Information encoded in binary**
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- **Combinational element**
  - Operate on data
  - Output is a function of current input only
- **State (sequential) elements**
  - Store information
  - Output is a function of current and past inputs (state)

Simple RISC-V Processor Implementation

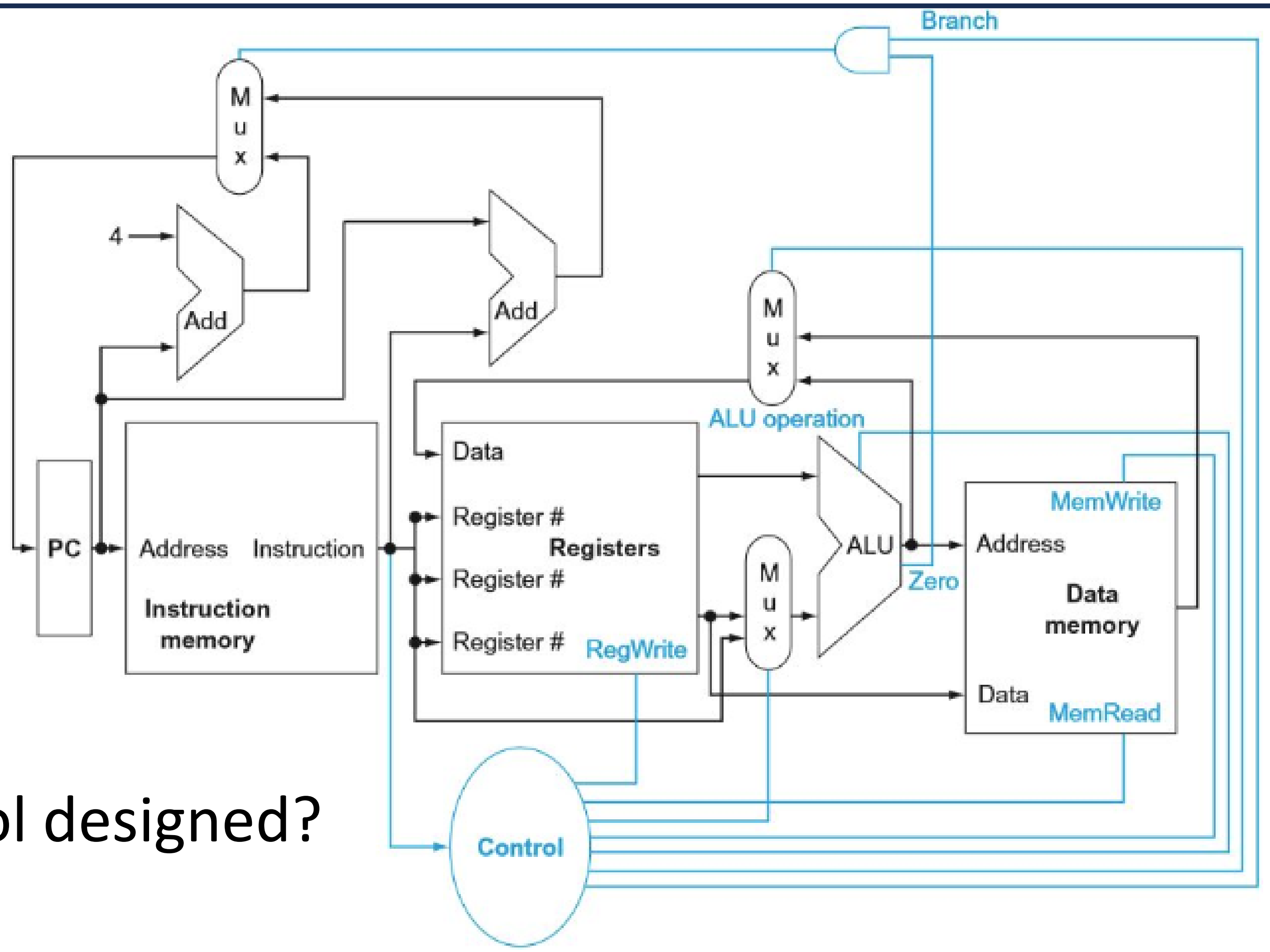
- **Single-Cycle Implementation:**



- **Datapath processes an instruction in one clock cycle**
  - ➢Each Datapath element can only do one function at a time
  - ➢Hence, we need separate instruction and data memories

- **Use multiplexers where alternate data sources are used for different instructions**

Simple RISC-V Processor Implementation



How is the control designed?

ALU Control

- **Let's start with the most important component in Datapath**
- **ALU used for:**
  - ➢R-type: Function depends on opcode (AND, OR, add, sub)
  - ➢Load/Store: Function = add
  - ➢Branch: Function = subtract

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract

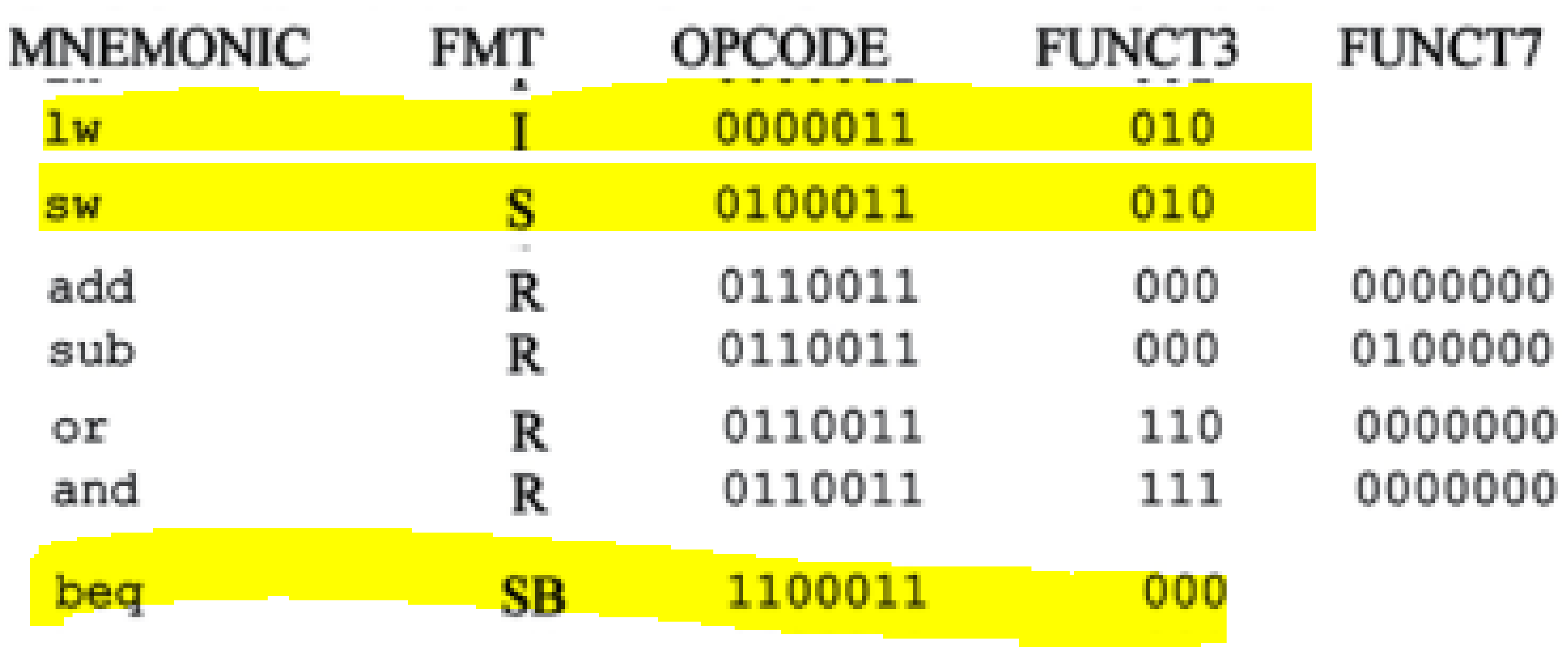
Points to Ponder

- **Show the circuit diagram inside the ALU with the following control function:**

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract

Instruction Codes

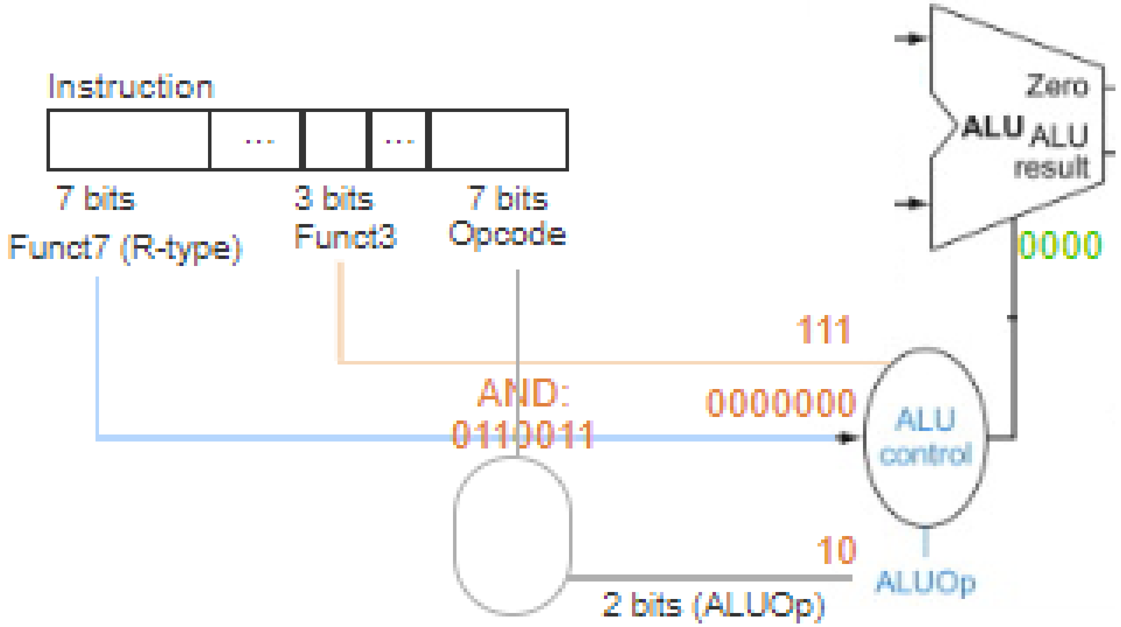
- Control signals: Generated from Instruction codes of the instructions

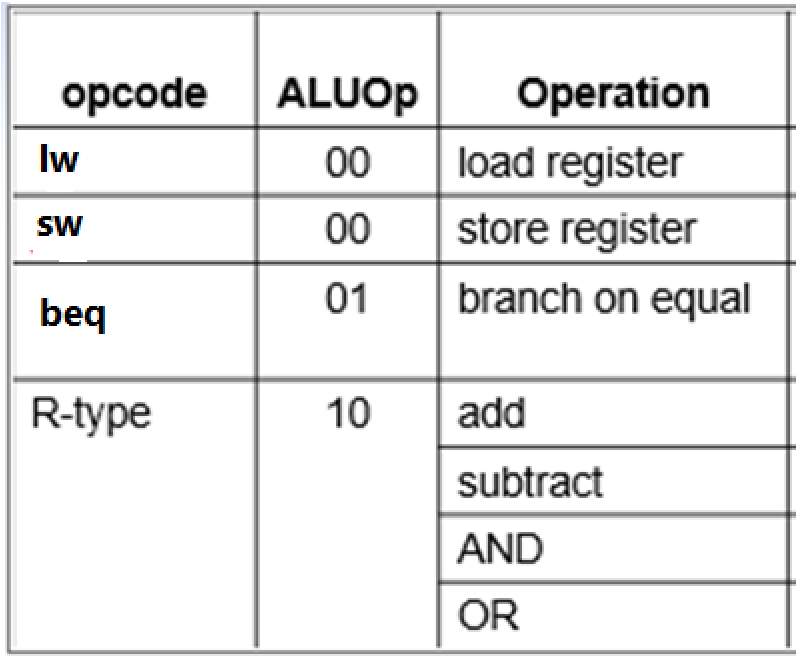


MNEMONIC	FMT	OPCODE	FUNCT3	FUNCT7
lw	I	0000011	010	
sw	S	0100011	010	
add	R	0110011	000	0000000
sub	R	0110011	000	0100000
or	R	0110011	110	0000000
and	R	0110011	111	0000000
beq	SB	1100011	000	

ALU Control (Cont.)

- **Decode is done in two steps to simplify the design:**





opcode	ALUOp	Operation
lw	00	load register
sw	00	store register
beq	01	branch on equal
R-type	10	add
		subtract
		AND
		OR

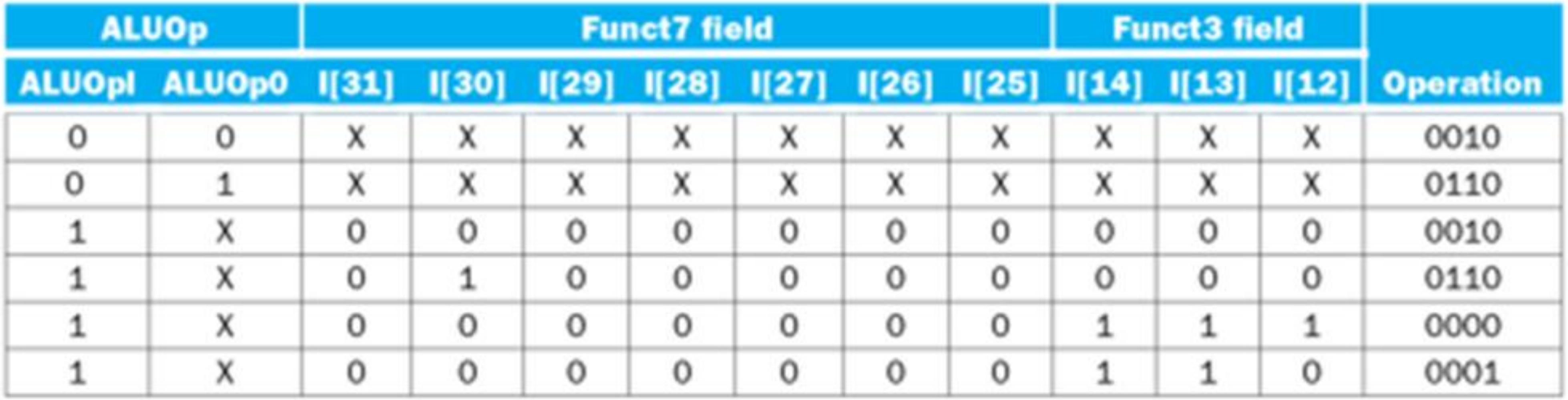
- **ALU Control Looks at ALUOp first**
- **For 10_2 case will look at Funct3 bits too! (values for “and”)**

ALU Control (Cont.)

- Complete Truth Table for ALU Control:

<u>instruction</u>

- lw, sw
- beq
- add
- sub
- and
- or

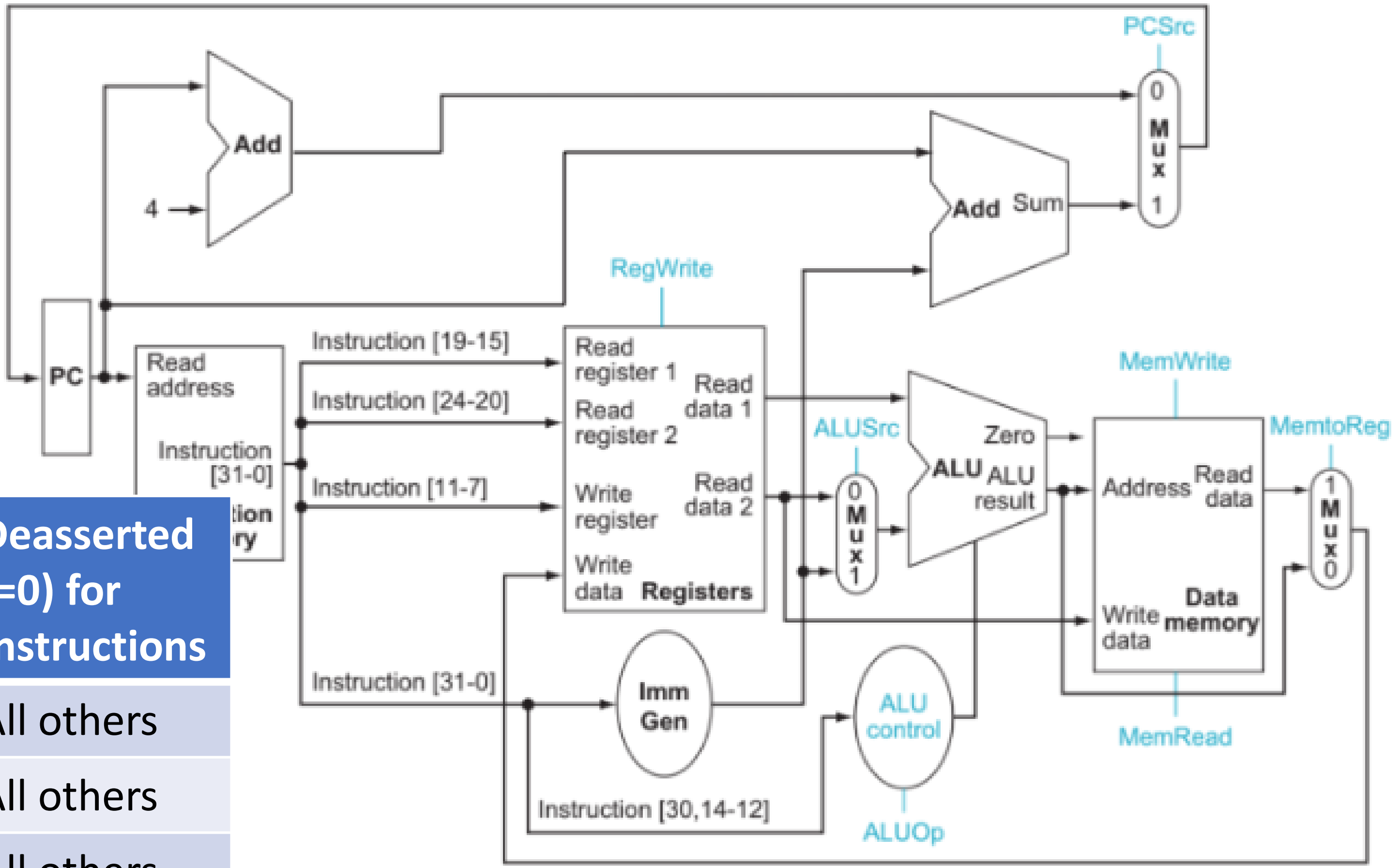


ALUOp		Funct7 field							Funct3 field			
ALUOpl	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

- Note: Use as many X's (don't care cases) to simplify the combinational circuit
- X's (don't care cases)?
- Operation(0) = ?
- Operation(1) = ?

Mux Control Signals

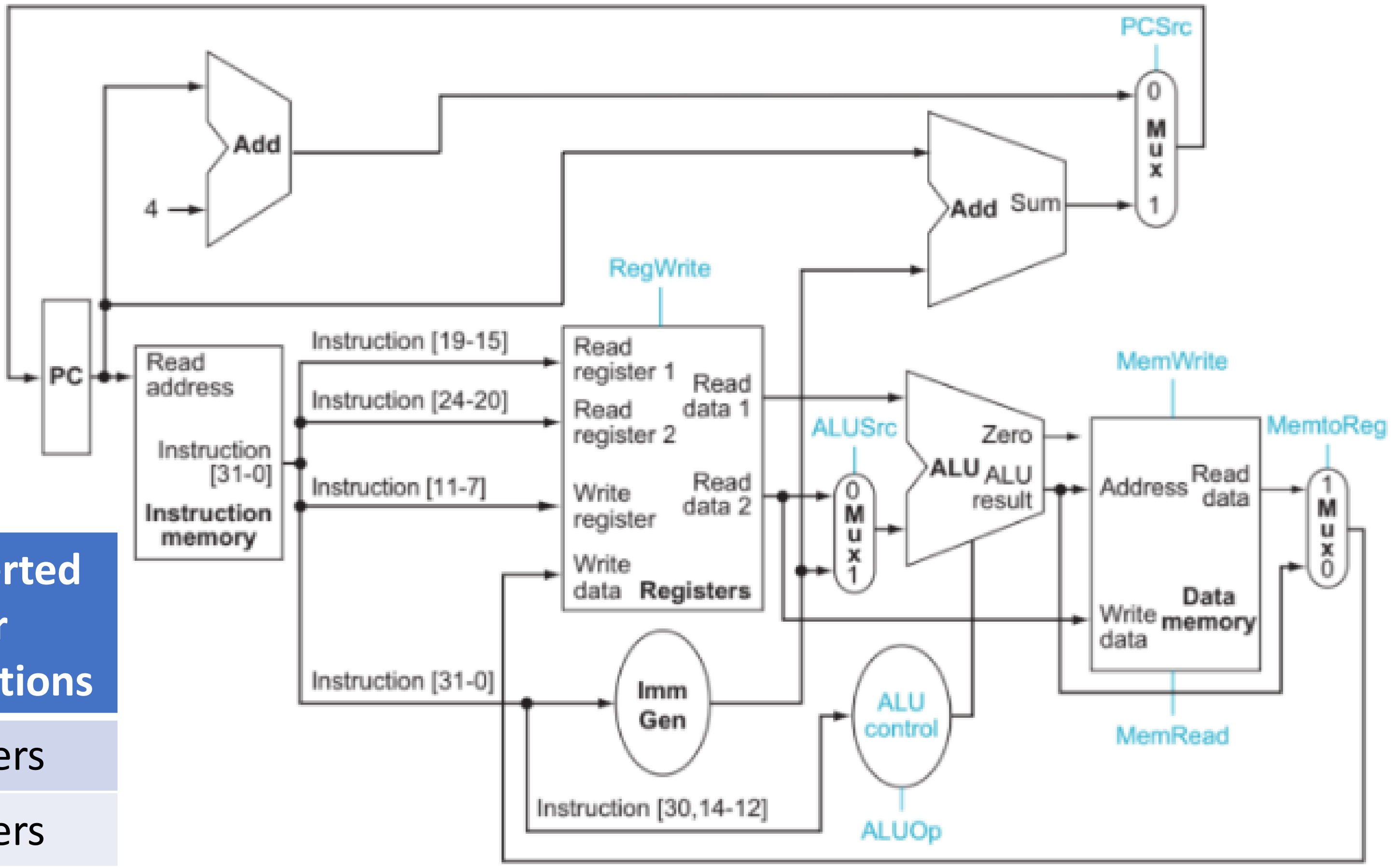
- Mux's select one out of the two inputs
- Instruction dependent



Signal Name	Asserted (=1) for Instructions	Deasserted (=0) for Instructions
MemtoReg	lw	All others
PCSrc	beq (and condition True)	All others
ALUSrc	Sw,lw	All others

Other Control Signals

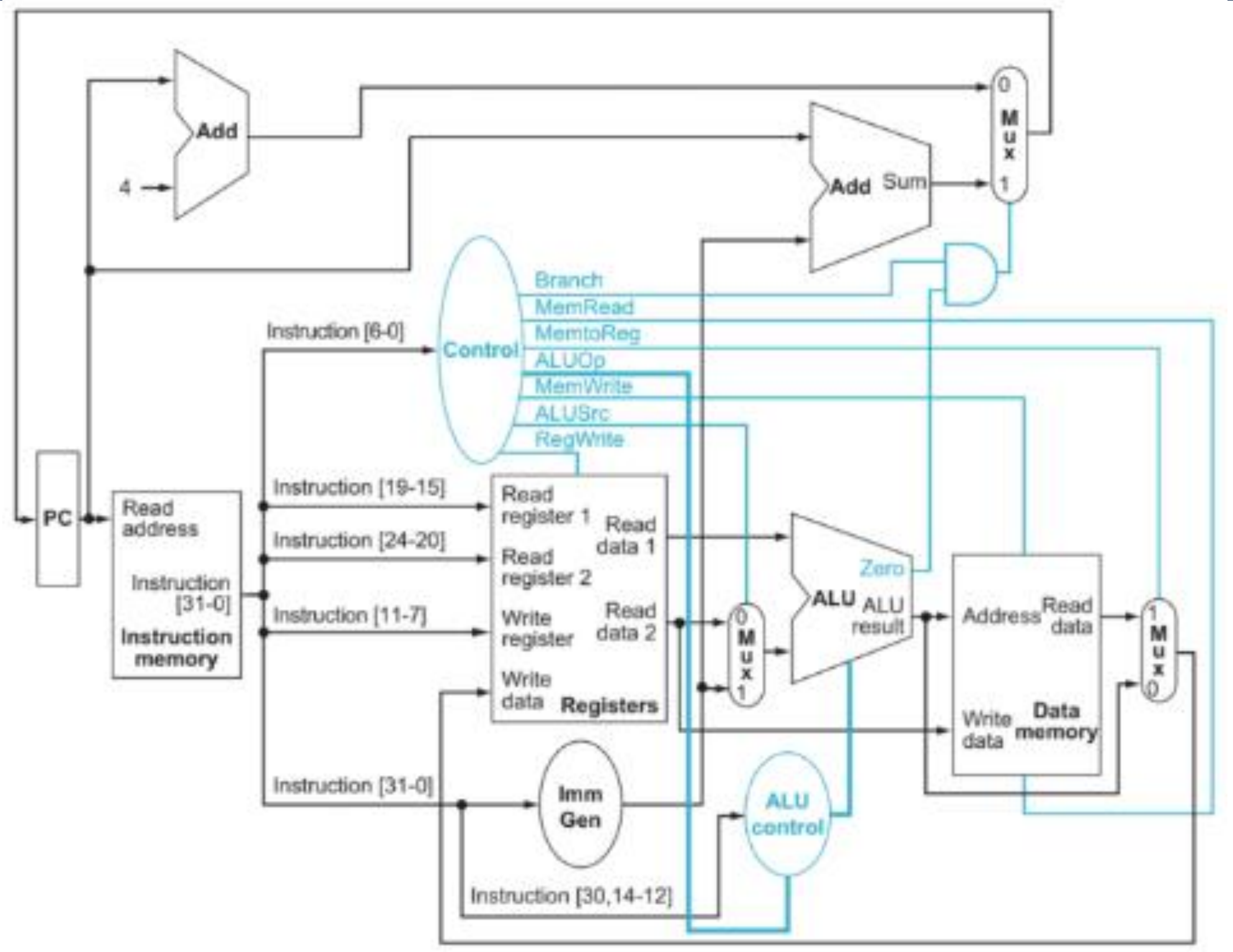
- **MemRead, MemWrite, and RegWrite**
- **Instruction dependent**



Signal Name	Asserted (=1) for Instructions	Deasserted (=0) for Instructions
MemRead	lw	All others
MemWrite	sw	All others
RegWrite	add, sub, and, or, lw	All others

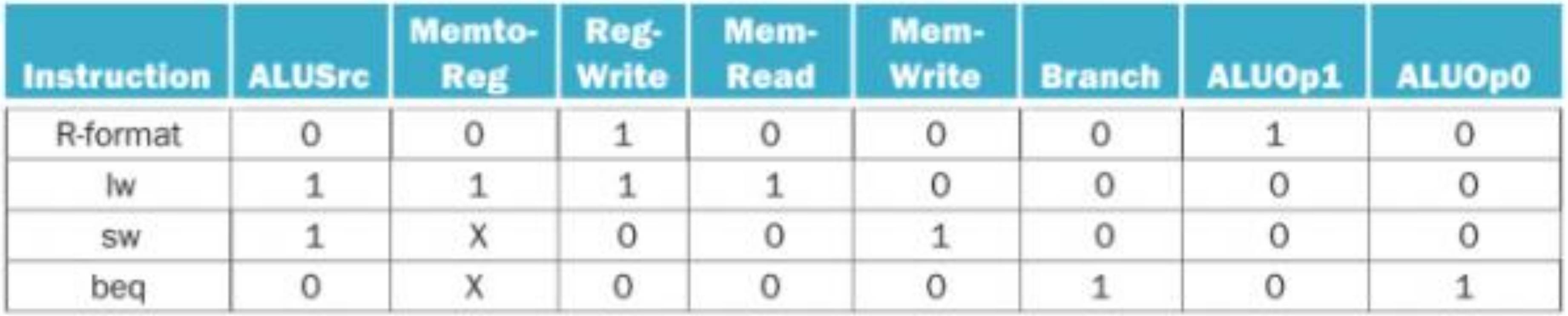
Control Unit Signals

- **A total of 8 signals:**
  - ➢MemRead
  - ➢MemWrite
  - ➢RegWrite
  - ➢MemtoReg
  - ➢ALUOp[1:0]
  - ➢ALUSrc
  - ➢Branch (.and. Zero) = PCSrc



Control Unit Design

- **Simple combinational circuit to design the 8 signals:**
  - ➢MemRead
  - ➢MemWrite
  - ➢RegWrite
  - ➢MemtoReg
  - ➢ALUOp[1:0]
  - ➢ALUSrc
  - ➢Branch (.and. Zero) = PCSrc

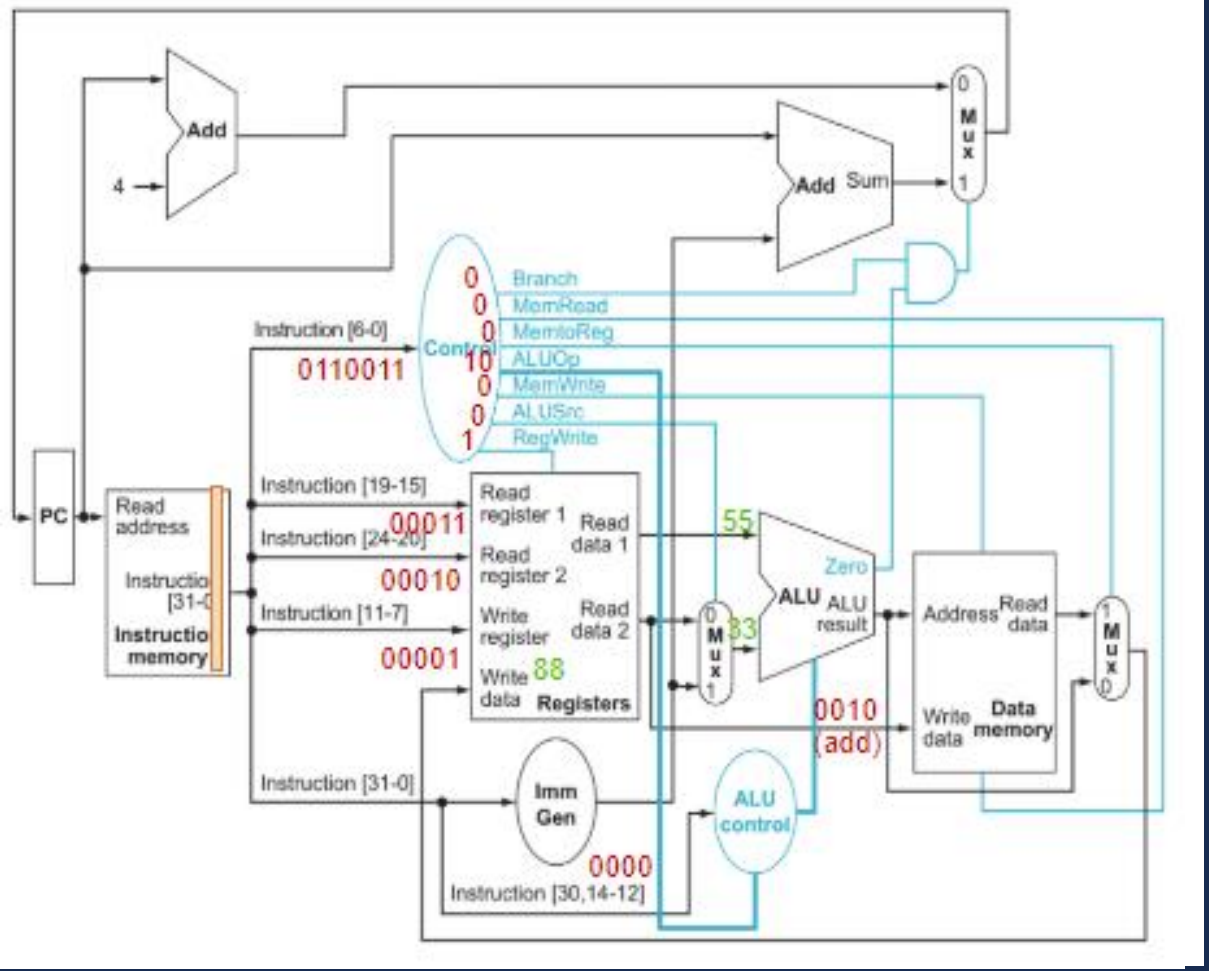


Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
lw	1	1	1	1	0	0	0	0
sw	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

Datapath Operation

- All control signals for instruction: add x1, x2, x3:
- Instruction code:

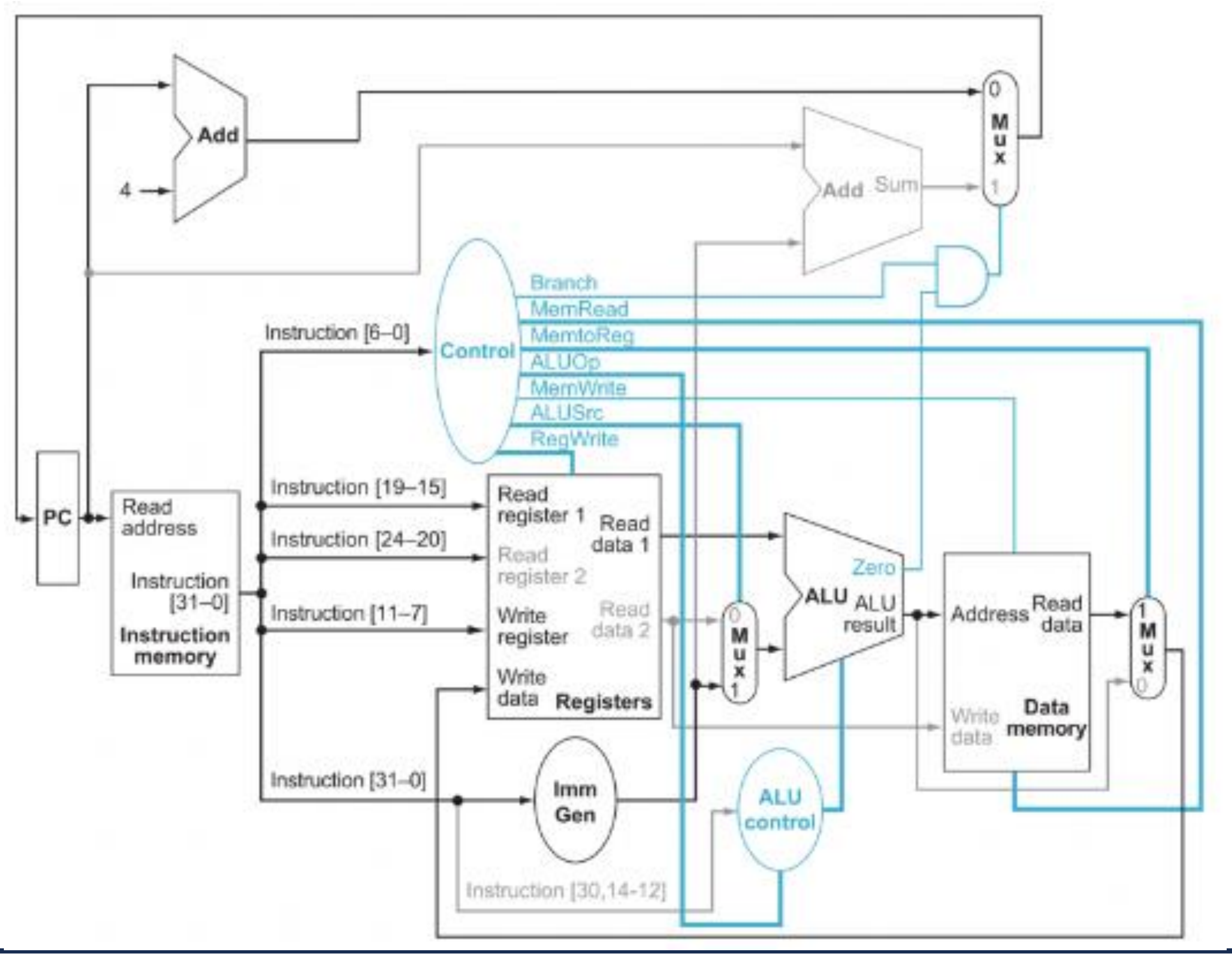
0000000 00011 00010 000 00001 0110011



Quick Check

- **Provide all control signals for instruction: lw x11, 16(x12) on the following figure:**
- **Instruction code:**

?

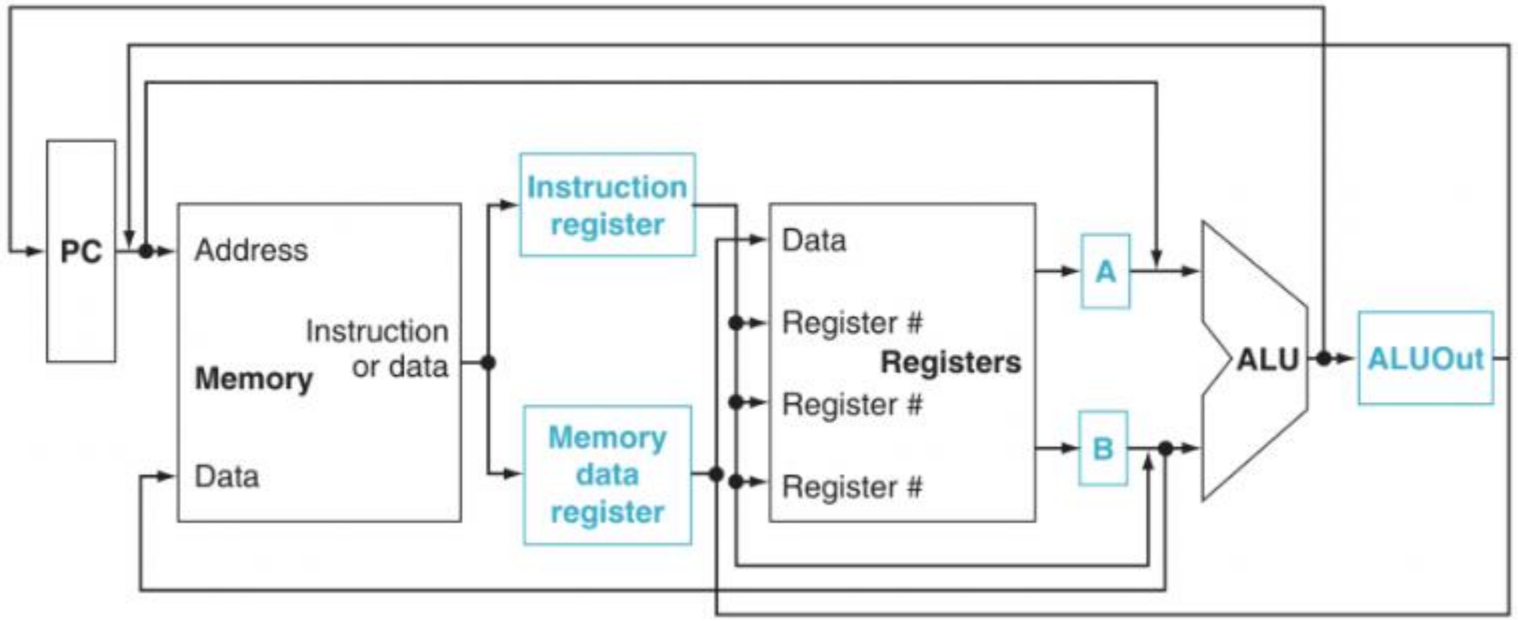


Single-Cycle Implementation Features

- **Easy to understand**
- **Easy to design**
- **Easy to build**
- **Easy to debug**
- **Very low performance!**
  - ➢Why?
  - ➢The clock period (frequency) is set by the slowest instruction
- **Might be fine for simple processors with simple ISA**
- **Gets much worse when more complex (time consuming) instructions exist in ISA**

A Multicycle Implementation

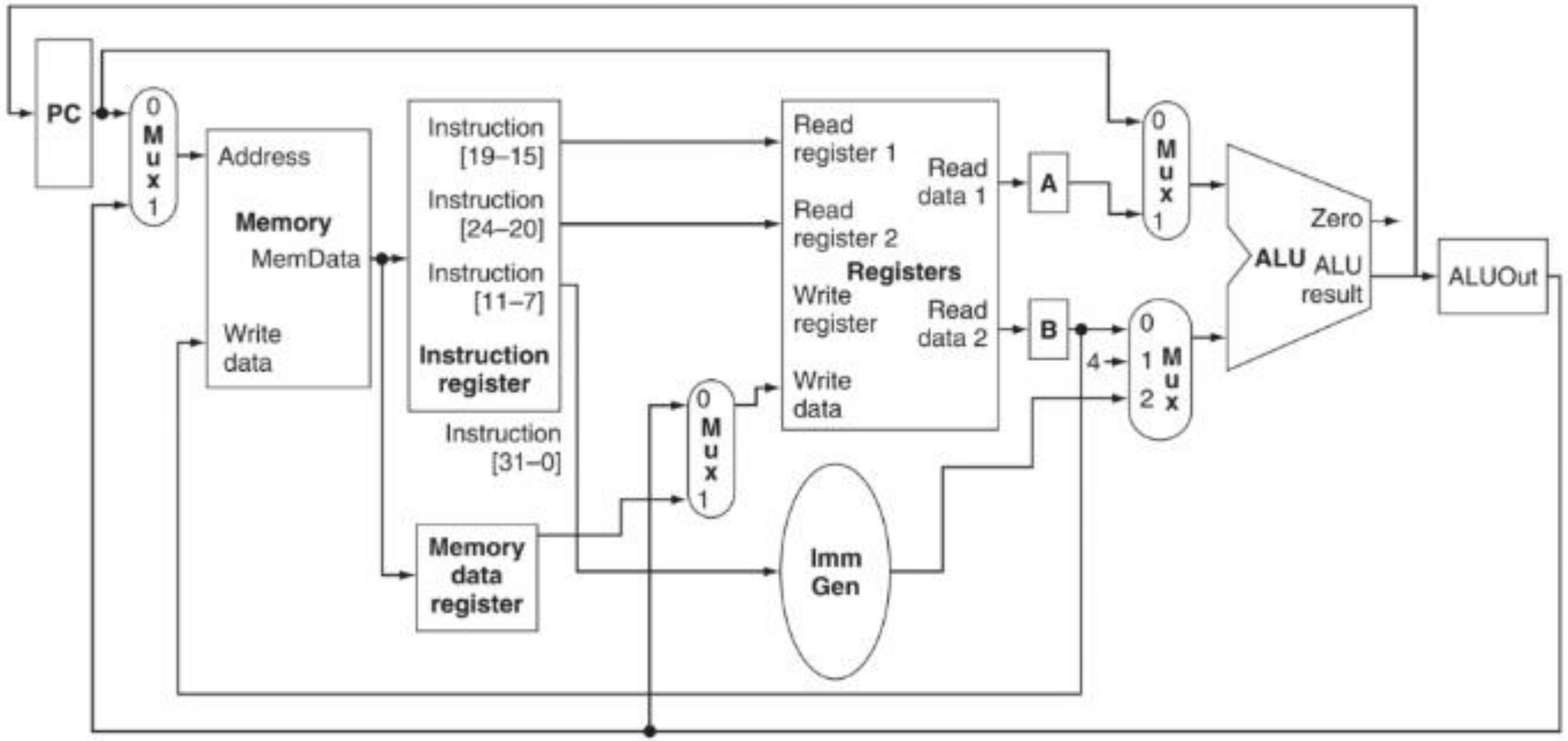
- **A possible Datapath:**



- **Major features:**
  - ➢Blocks are reused
  - ➢Some new registers:
    - IR
    - MDR
    - A
    - B
    - ALUOut
  - ➢Shared memory for data and instructions
  - ➢Instructions can have variable number of clock cycles to execute

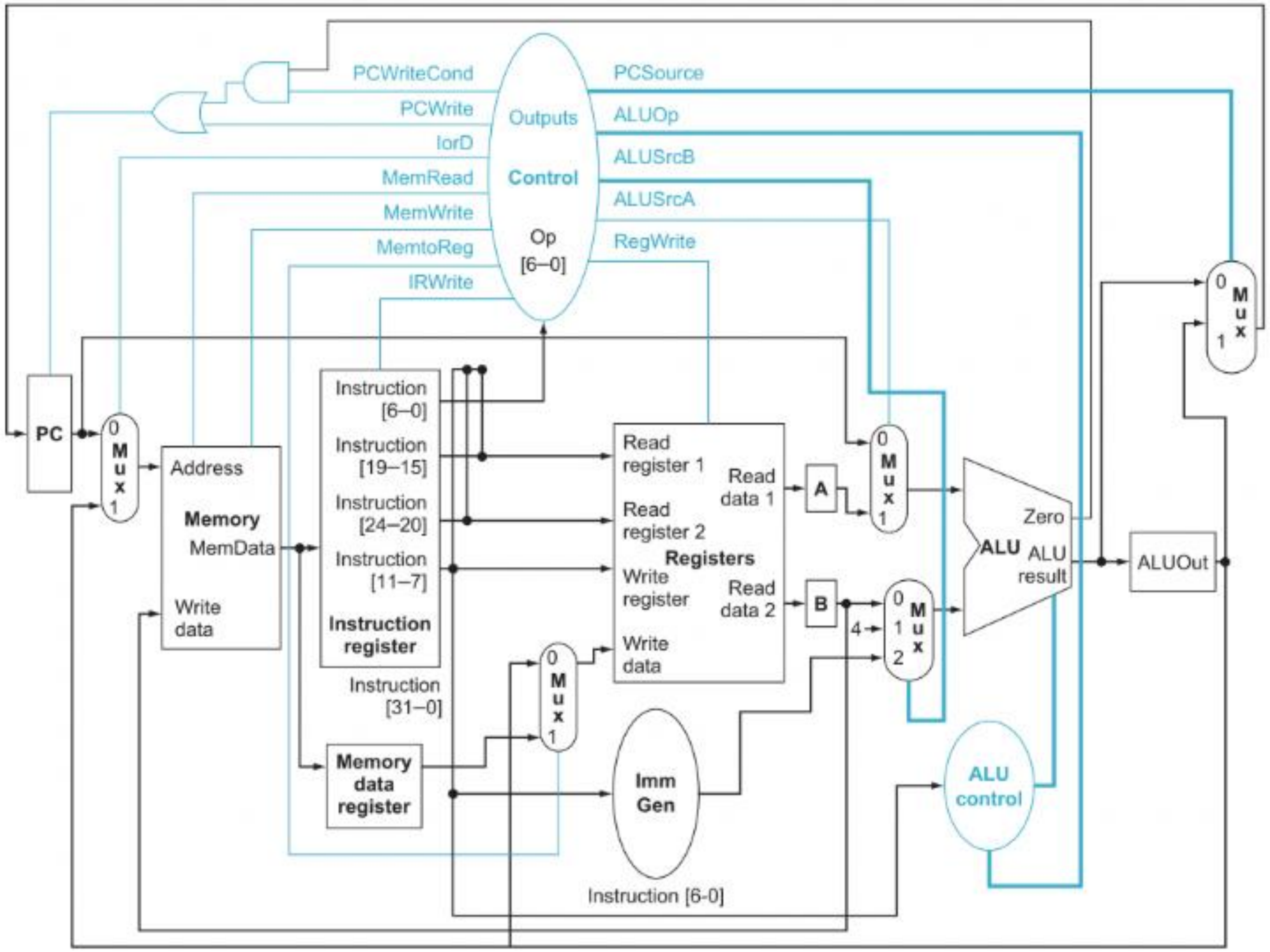
A Multicycle Implementation

- **A more detailed Datapath:**



A Multicycle Implementation

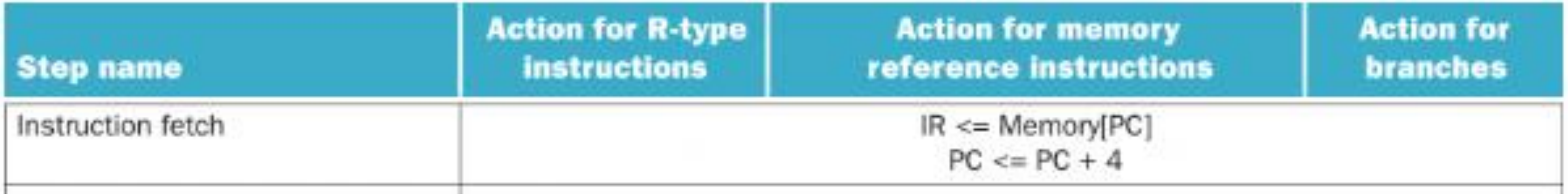
- **The CPU:**



A Multicycle Implementation

- **Instructions are executed in several steps (stages)**
- **Some steps (stages) are the same for all Instructions**
- **Each step (stage) takes one clock cycle to finish**

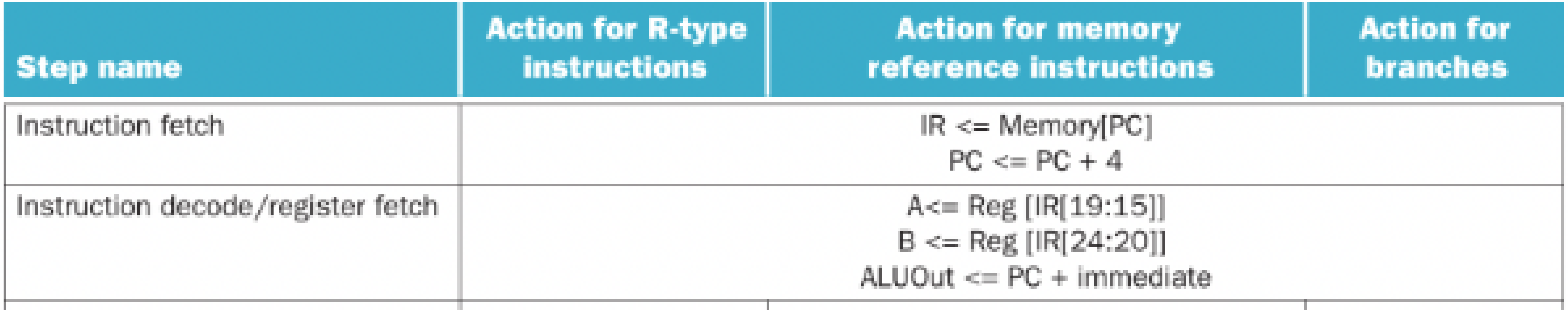
- **Instruction Fetch:**
  - ➢The same for all instructions



Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch		IR <= Memory[PC] PC <= PC + 4	

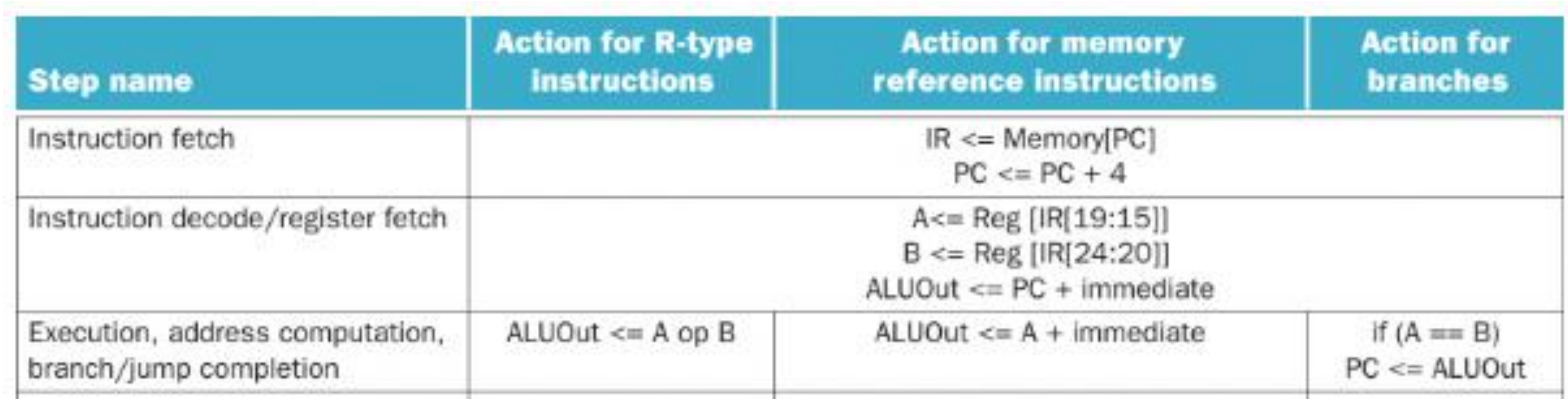
A Multicycle Implementation

- **Instruction Fetch:**
  - ➢The same for all instructions
- **Instruction decode/register fetch:**
  - ➢The same for all instructions



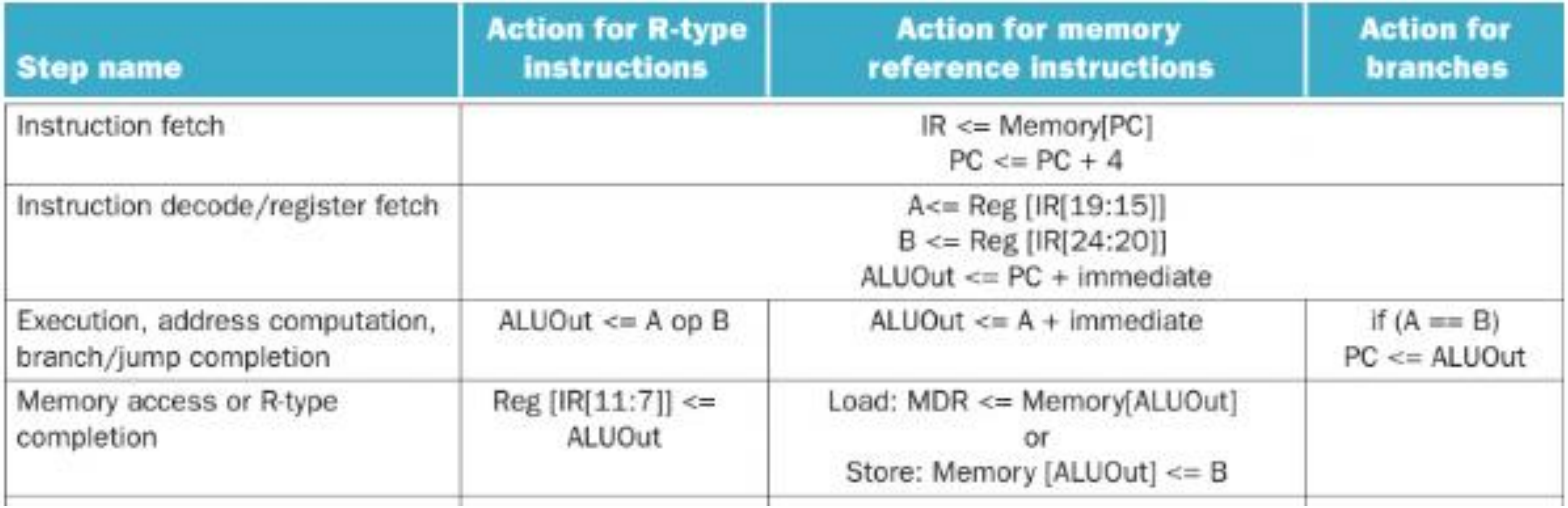
Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch		IR <= Memory[PC] PC <= PC + 4	
Instruction decode/register fetch		A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate	

A Multicycle Implementation



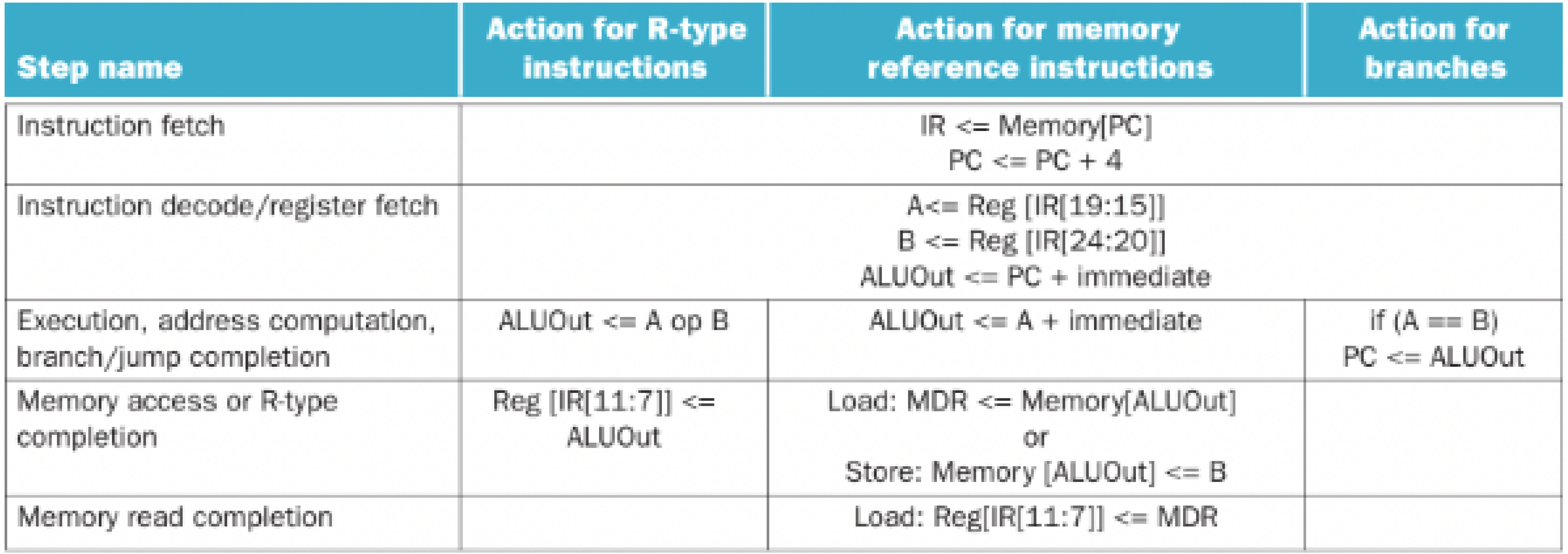
Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch		IR <= Memory[PC] PC <= PC + 4	
Instruction decode/register fetch		A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate	
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut

A Multicycle Implementation



Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch		IR <= Memory[PC] PC <= PC + 4	
Instruction decode/register fetch		A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate	
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	

A Multicycle Implementation



Step name	Action for R-type instructions	Action for memory reference instructions	Action for branches
Instruction fetch	IR <= Memory[PC] PC <= PC + 4		
Instruction decode/register fetch	A<= Reg [IR[19:15]] B <= Reg [IR[24:20]] ALUOut <= PC + immediate		
Execution, address computation, branch/jump completion	ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut
Memory access or R-type completion	Reg [IR[11:7]] <= ALUOut	Load: MDR <= Memory[ALUOut] or Store: Memory [ALUOut] <= B	
Memory read completion		Load: Reg[IR[11:7]] <= MDR	

Control Unit Implementation

- Design of the control unit is harder (than the single-cycle case)
- Usually, an FSM implementation helps
- We will not cover the whole process
- See book for more details if interested

Multicycle Implementation Features

- Less hardware in Datapath as blocks can be reused
- Instructions can skip some stages and run faster
- More complex Control Unit
- Still simple enough to understand and debug easily
- Better performance compared to single-cycle implementation

Performance Analysis

- Assume a program instruction mix is 25% Loads, 10% Stores, 55% ALU, and the rest Branch instructions. What is the CPI of this program on this implementation?

Solution:

Load needs 5 Clock Cycles

Store needs 4 Clock Cycles

ALU instruction needs 4 Clock Cycles

Branch instruction needs 3 Clock Cycles

CPI = 25% * 5 + 10% * 4 + 55% * 4 + 10% * 3 = 1.25 + 0.40 + 2.20 + 0.30 = 4.15

Performance Comparison

- Assume a program instruction mix is 25% Loads, 10% Stores, 55% ALU, and the rest Branch instructions. What is the CPI of this program on the single-cycle implementation? Which implementation has higher performance?

Solution:

Load needs ? Clock Cycles

Store needs ? Clock Cycles

ALU instruction needs ? Clock Cycles

Branch instruction needs ? Clock Cycles

CPI = 25% * ? + 10% * ? + 55% * ? + 10% * ? = 0.25 + 0.10 + 0.55 + 0.10 = 1

Points to Ponder

Assume:

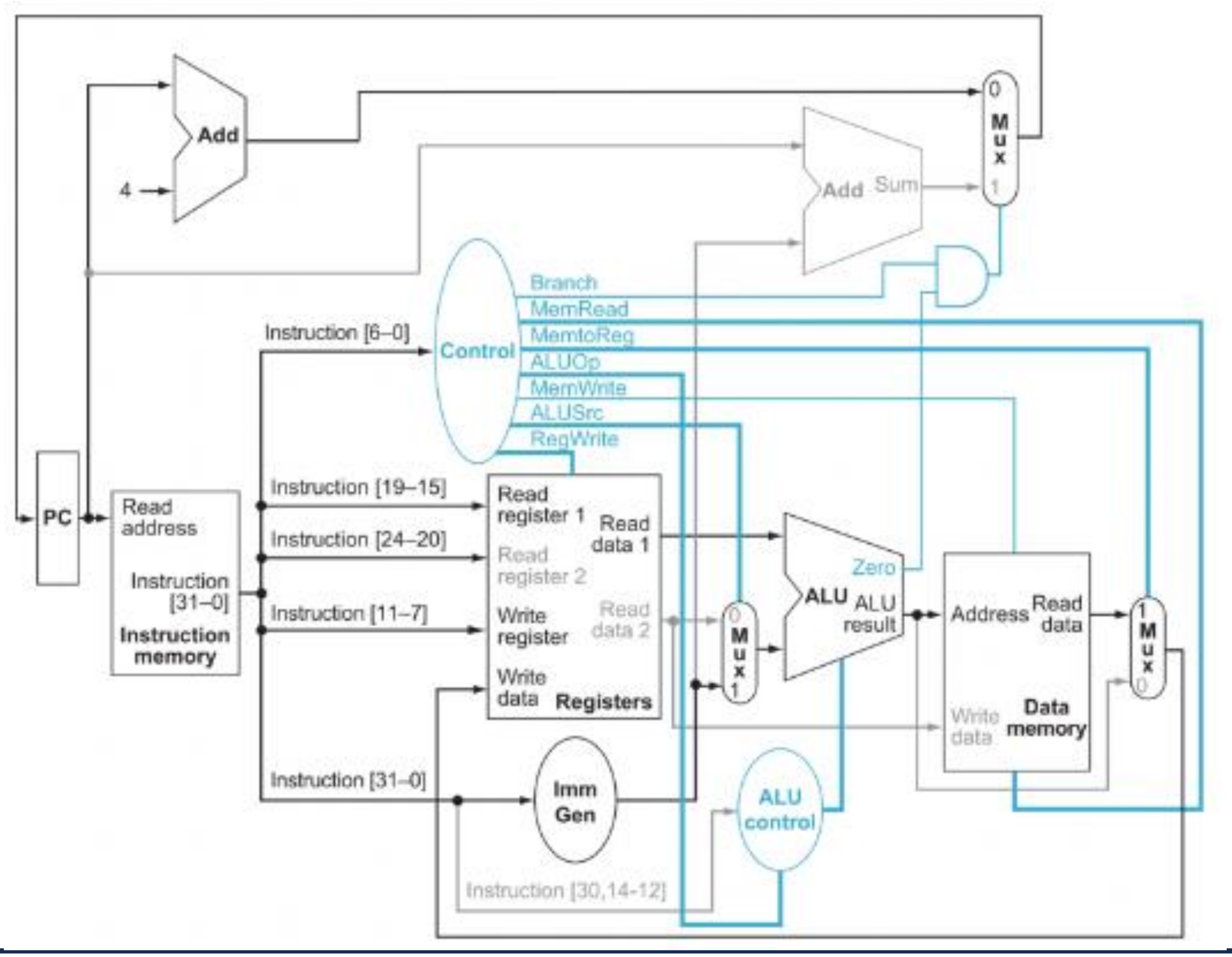
- A memory unit access (read/write) needs 50ps,
- A register file access (read/write) needs 10ps,
- Each ALU operation (including ADDERs) needs 10ps,
- Each MUX has 5ps delay,
- Each control register has 2ps propagation delay time and needs 2ps set up time,
- and every other operation in the Datapath needs 0ps,
- and all control signals are produced instantaneously (0ps delay).

Find the minimum Clock Period for the two implementations (single-Cycle and Multicycle) and from there compare the performance of the implementations for the program in the previous slide.

UMASS AMHERST

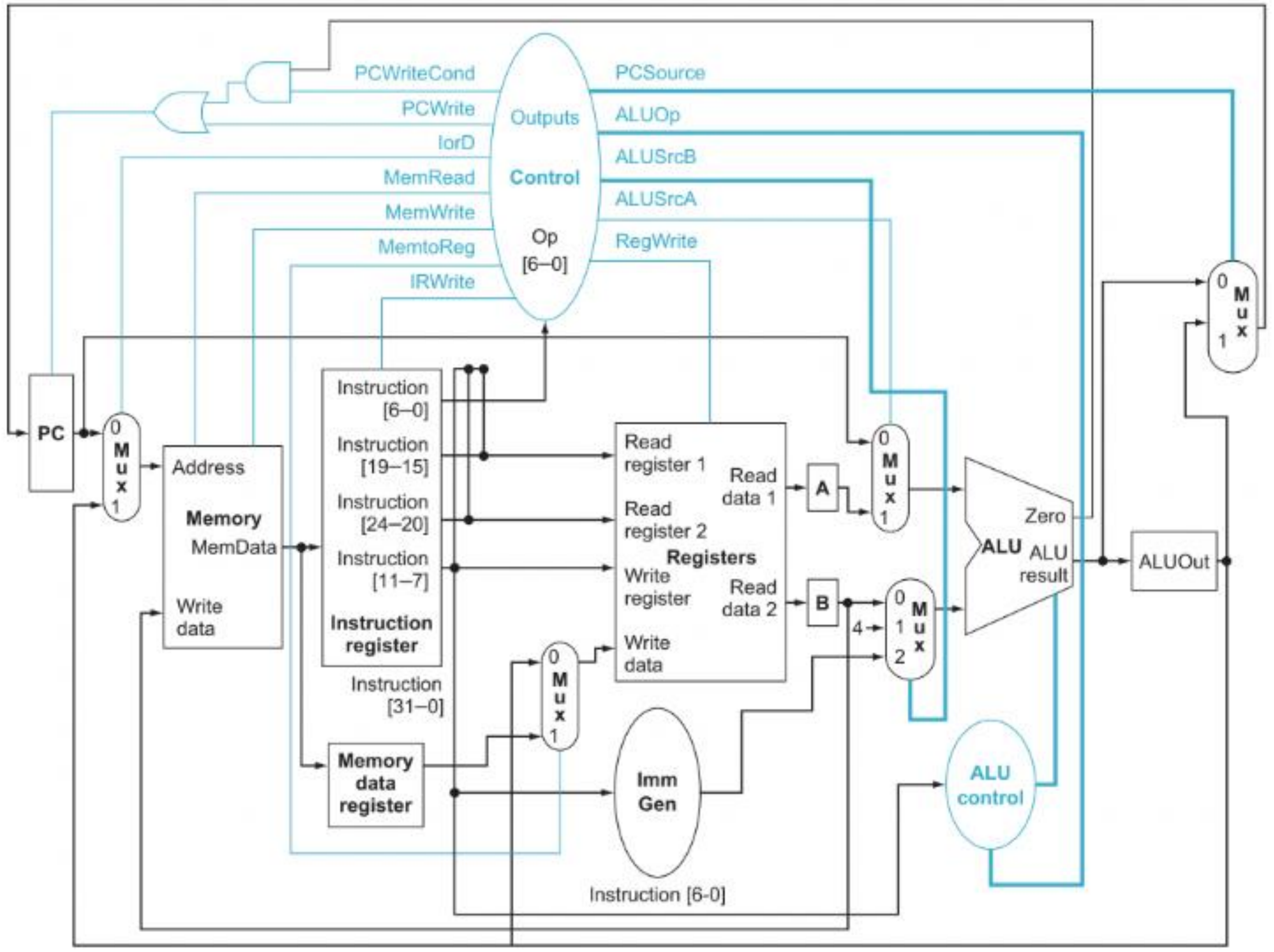
Use to answer Slide 32 for single-cycle case

?



Use to answer Slide 32 for Multicycle case

- ?


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op [6–0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
PC
Mux
Address
Memory
MemData
Write data
Instruction [6–0]
Instruction [19–15]
Instruction [24–20]
Instruction [11–7]
Instruction register
Instruction [31–0]
Memory data register
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
A
B
4
Imm Gen
Instruction [6-0]
Zero
ALU
ALU result
ALUOut
ALU control