# CSE 2021
# Computer Organization

Hugh Chesser, CSEB
1012U


W9-W

# Agenda

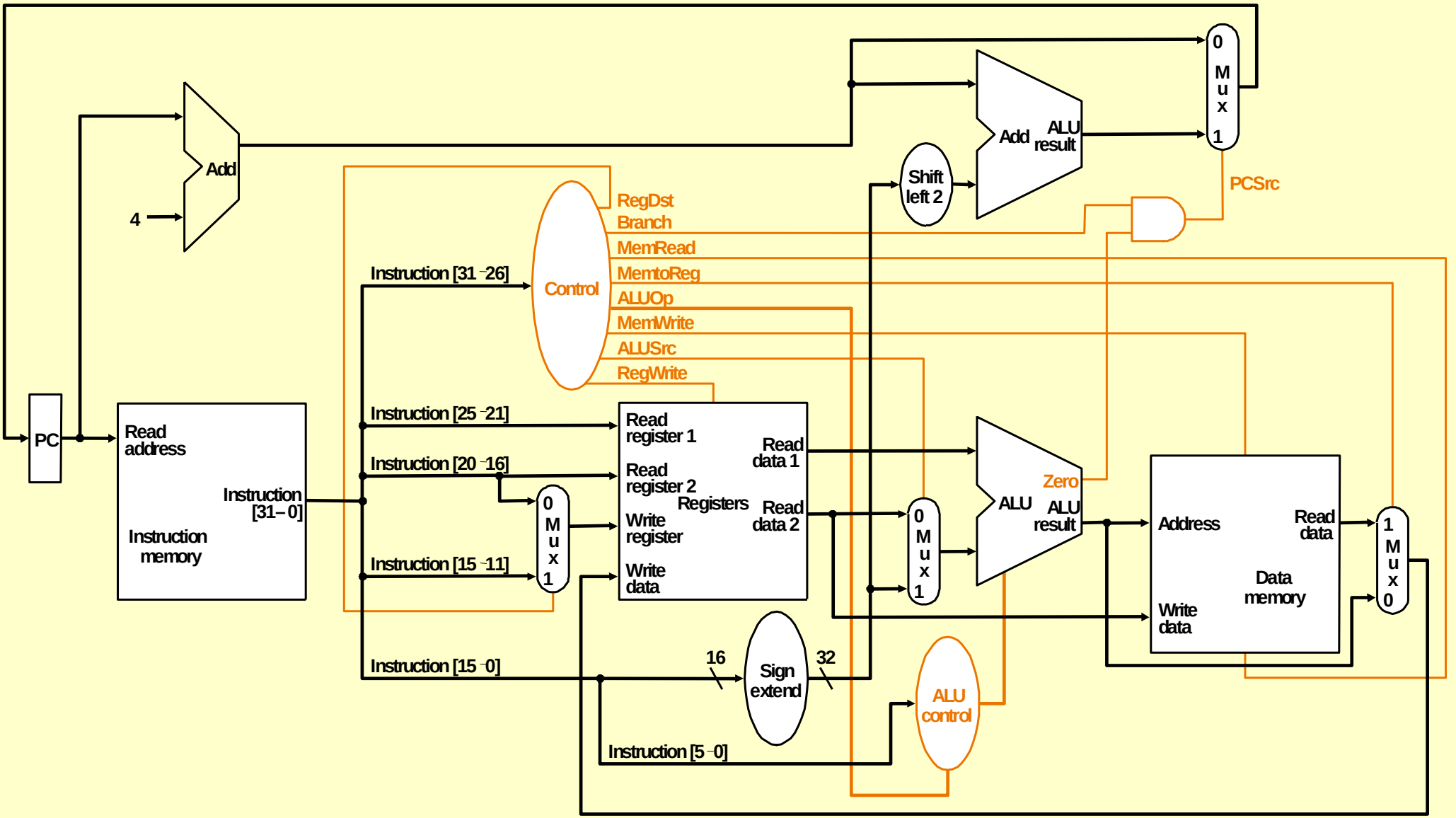

Topics:

1. Single Cycle Review (Sample Exam/Quiz Q)
2. Multiple cycle implementation

Patterson: Section 4.5

Reminder: Quiz #2 – Next Wednesday (November 11)

# Main Control (4)



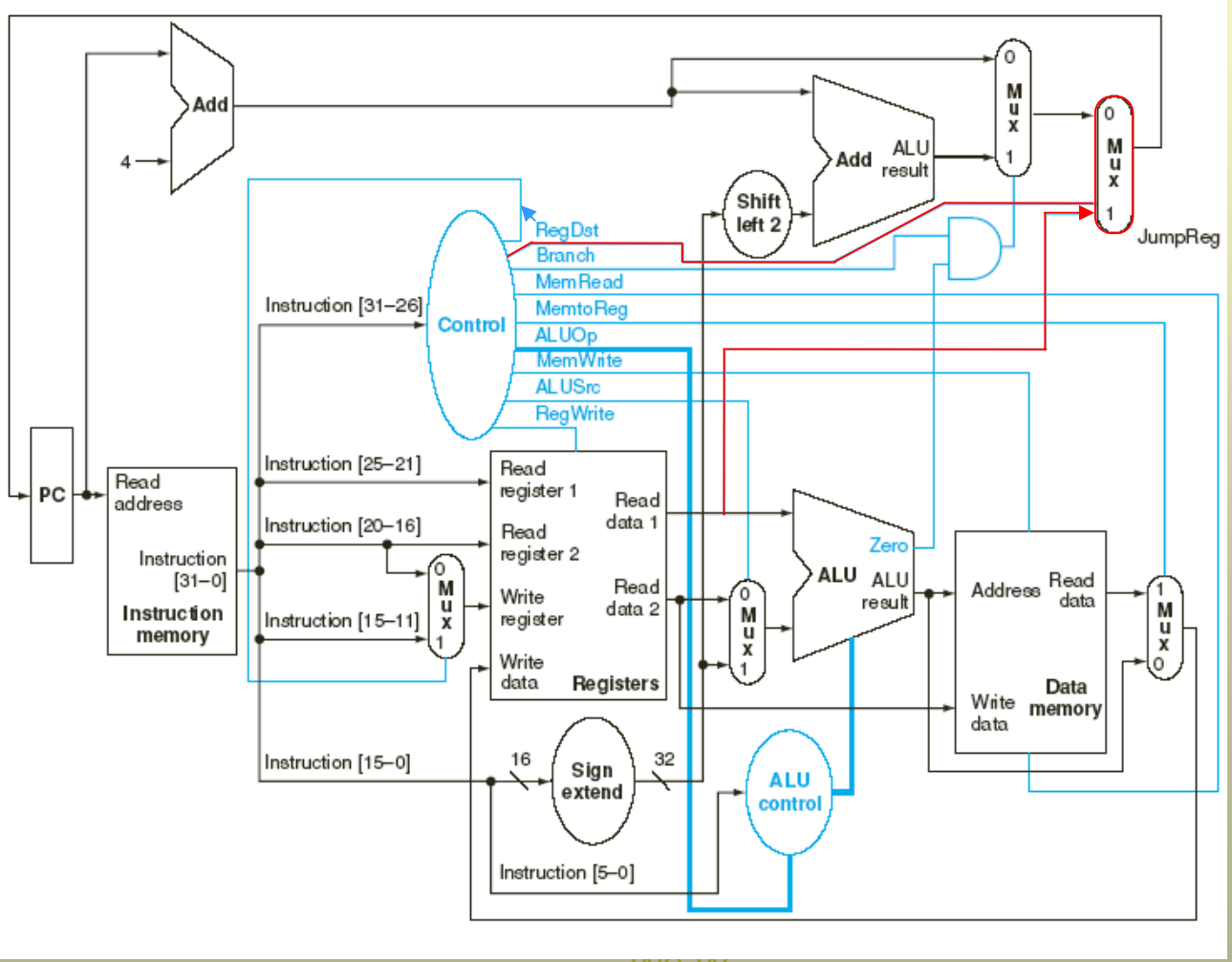
# Activity (Sample Quiz, Exam Q)


*We wish to add jr (jump register) to the single cycle datapath from the previous slide. Add the necessary connections to the single cycle datapath block diagram to implement the jr instruction. Also, append the table below to add the necessary control signals needed for the jr instruction.*

| Instruction | RegDst | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | ALUOp1 | ALUOp0 |
|---|---|---|---|---|---|---|---|---|---|
| **R-format** | **1** | **0** | **0** | **1** | **0** | **0** | **0** | **1** | **0** |
| `lw` | **0** | **1** | **1** | **1** | **1** | **0** | **0** | **0** | **0** |
| `sw` | **X** | **1** | **X** | **0** | **0** | **1** | **0** | **0** | **0** |
| `beq` | **X** | **0** | **X** | **0** | **0** | **0** | **1** | **0** | **1** |

# Answer (Part 1):

*Modify the datapath as shown*



# Answer (Part 2):


*… append the table below to add the necessary control signals needed for the jr instruction.*

| Instruction | RegDst | ALUSrc | MemtoReg | RegWrite | MemRead | MemWrite | Branch | JumpReg | ALUOp1 | ALUOp0 |
|---|---|---|---|---|---|---|---|---|---|---|
| **R-format** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| `lw` | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| `sw` | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| `beq` | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| `jr` | X | 0 | X | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

# Why single-cycle implementation is not used?


Assuming no delay at adder, sign extension unit, shift left unit, PC, control unit, and MUX:

- Load cycle requires 5 functional units:
  instruction fetch, register access, ALU, data memory access, register access
- Store cycle requires 4 functional units:
  instruction fetch, register access, ALU, data memory access
- R-type instruction cycle requires 4 functional units:
  instruction fetch, register access, ALU, register access
- Path for a branch instruction requires 3 functional units:
  instruction fetch, register access, ALU
- Path for a jump instruction requires 1 functional unit:
  instruction fetch

Using a clock cycle of equal duration for each instruction is a waste of resources.

# Why Multicycle?


Example: Assume that the operation times for major functional unit in a microprocessor are:

Memory unit ~ 2ns, ALU and adders ~ 2ns, Register file ~ 1ns

Compare the performance of the following instruction mix

Loads: 24%; Stores: 12%; ALU instructions: 44%; Branches: 18%; Jumps: 2%

on the two implementations

Implementation I: All instructions operate in 1 clock cycle

Implementation II: Each instruction is as long as it needs to be.

| Instruction Class | Functional units used (Steps involved) | | | | | |
|---|---|---|---|---|---|---|
| ALU type | Instruction fetch | Register Access | ALU | Register Access | | 6ns |
| Load word | Instruction fetch | Register Access | ALU | Memory Access | Register Access | 8ns |
| Store word | Instruction fetch | Register Access | ALU | Memory Access | | 7ns |
| Branch | Instruction fetch | Register Access | ALU | | | 5ns |
| Branch | Instruction fetch | | | | | 2ns |

Average time per instruction:

Implementation 1: ~ 8ns

Implementation 2: ~0.24(8)+0.12(7)+0.44(6)+0.18(5)+0.02(2) = 6.34ns

# Multicycle Implementation


Instruction:

— Execution of each instruction is broken into different steps

— Each step requires 1 clock cycle

— Each instruction takes multiple clock cycles

Functional Unit:

— Can be used more than once in an instruction (but still only once in a clock cycle)

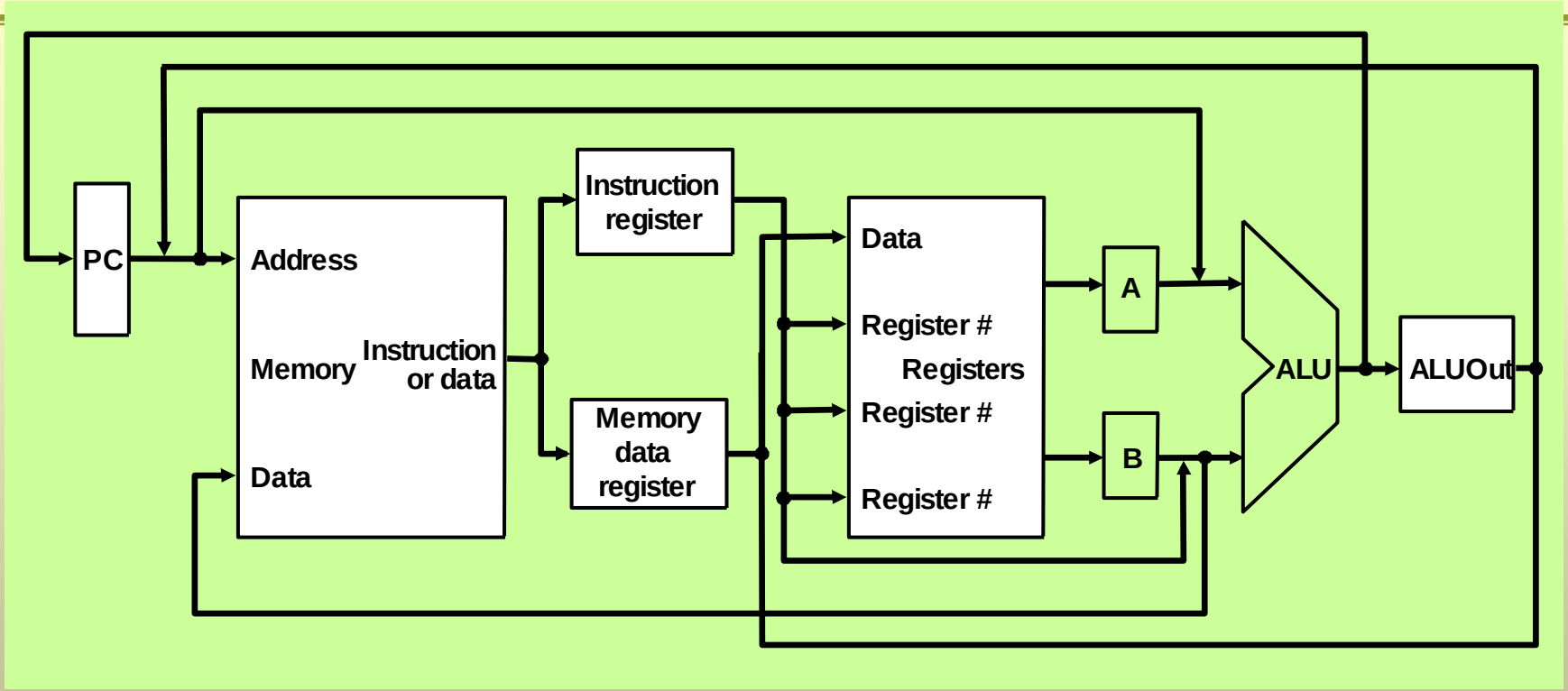
Advantages:

— Functional units can be shared

— ALU and adder is combined

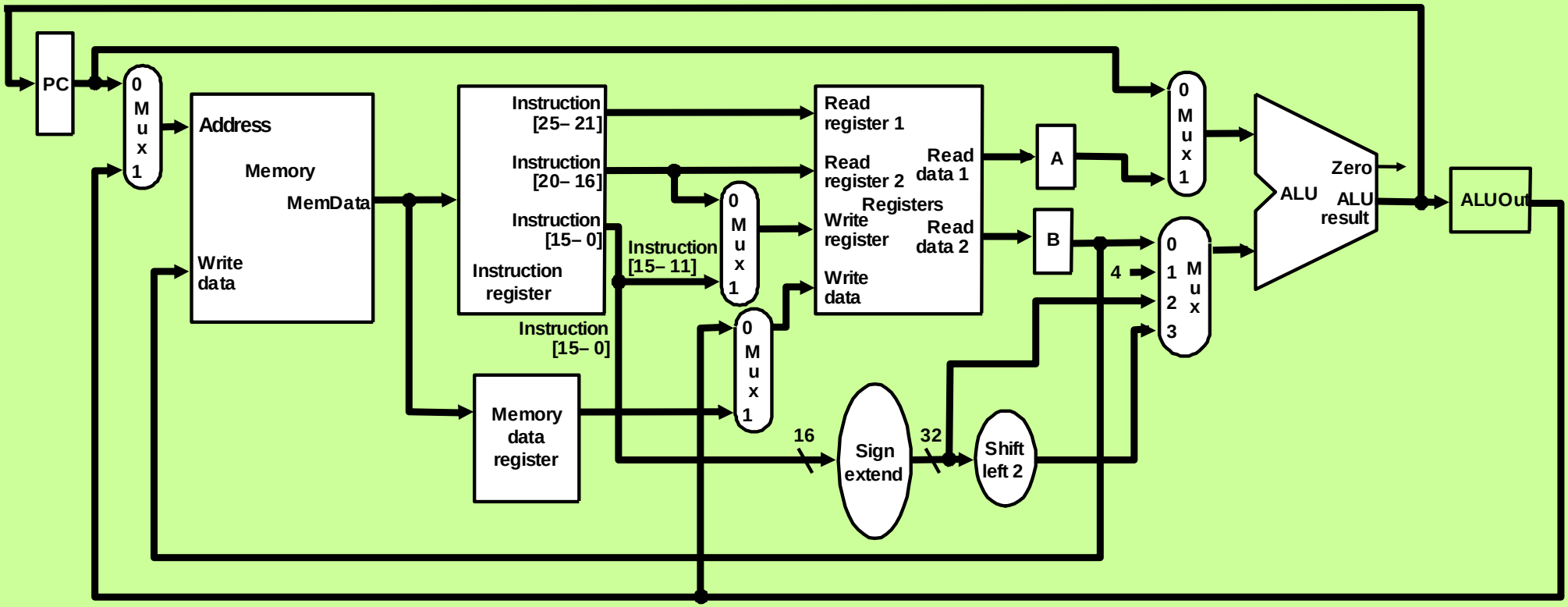
— Single memory is used for instructions and data

# Multicycle Implementation: Abstract Diagram




- One ALU is used for incrementing PC and for arithmetic operations
- Data memory and Instruction memory are combined
- 5 additional registers are added
  1. An instruction register (IR) to hold instructions before distributing data to register file or ALU
  2. A memory data register (MDR) to hold data before distributing to register file or ALU
  3. Registers A and B that hold data before the ALU
  4. Register ALUout that hold data computed by ALU

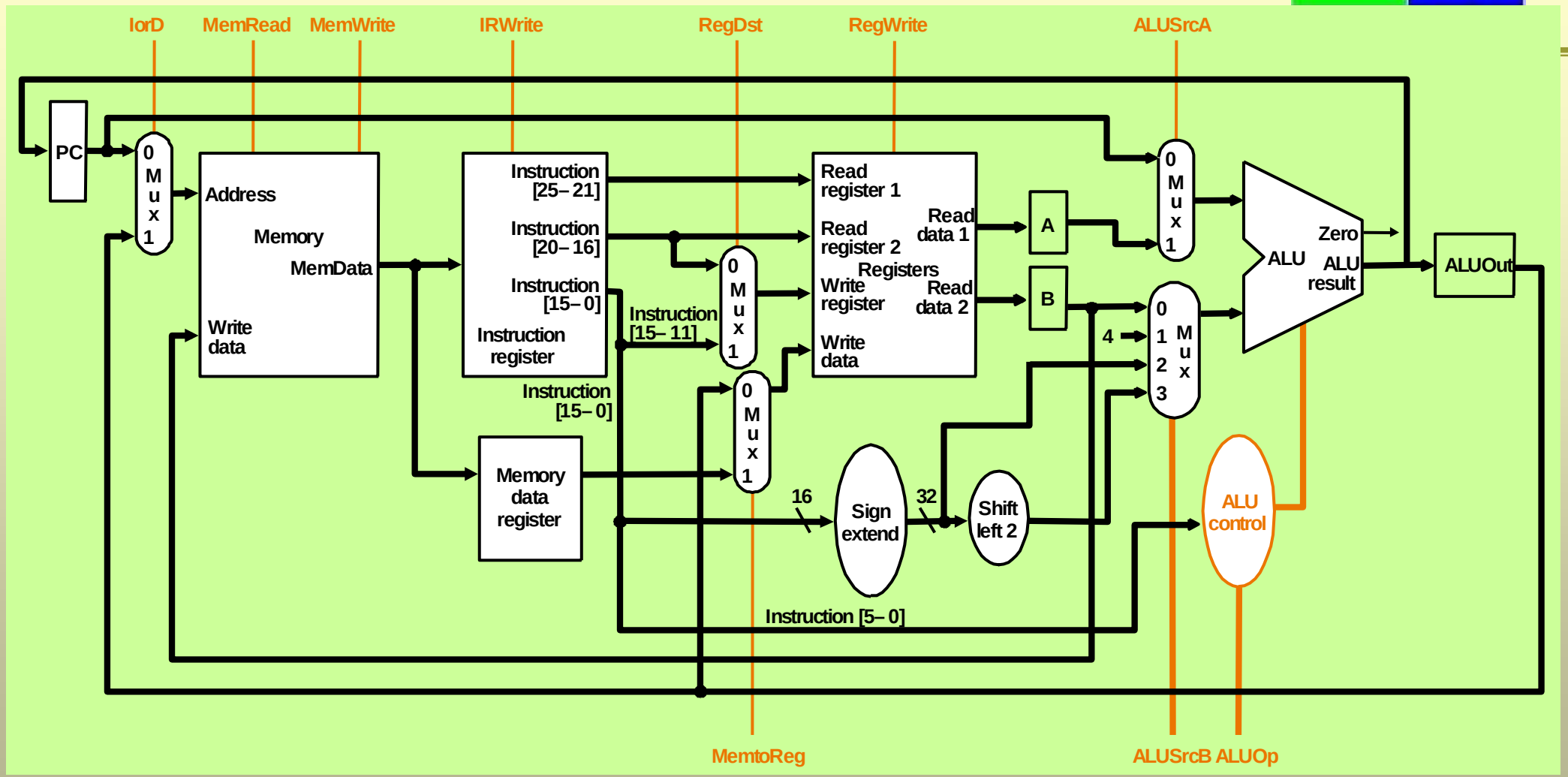
# Multicycle Implementation: Multiplexers added




Because functional units are shared, multiplexers are added to select data between different devices

1. MUX before memory selects either the PC output (fetch instruction) or ALU output (storing data)
2. MUX before "write register" selects write-register number (instruction [15-11] or instruction[20-16])
3. MUX before "write data" selects data from "ALUOut" (R-type instruction) or "MemData" (lw instruction)
4. Upper MUX before ALU selects PC output (increment PC) or "Read data 1" (R-type instruction)
5. Lower MUX before ALU selects "Read data 2", or "sign extended instruction[15-0]" or shift left sign extended instruction[15-0], or 4

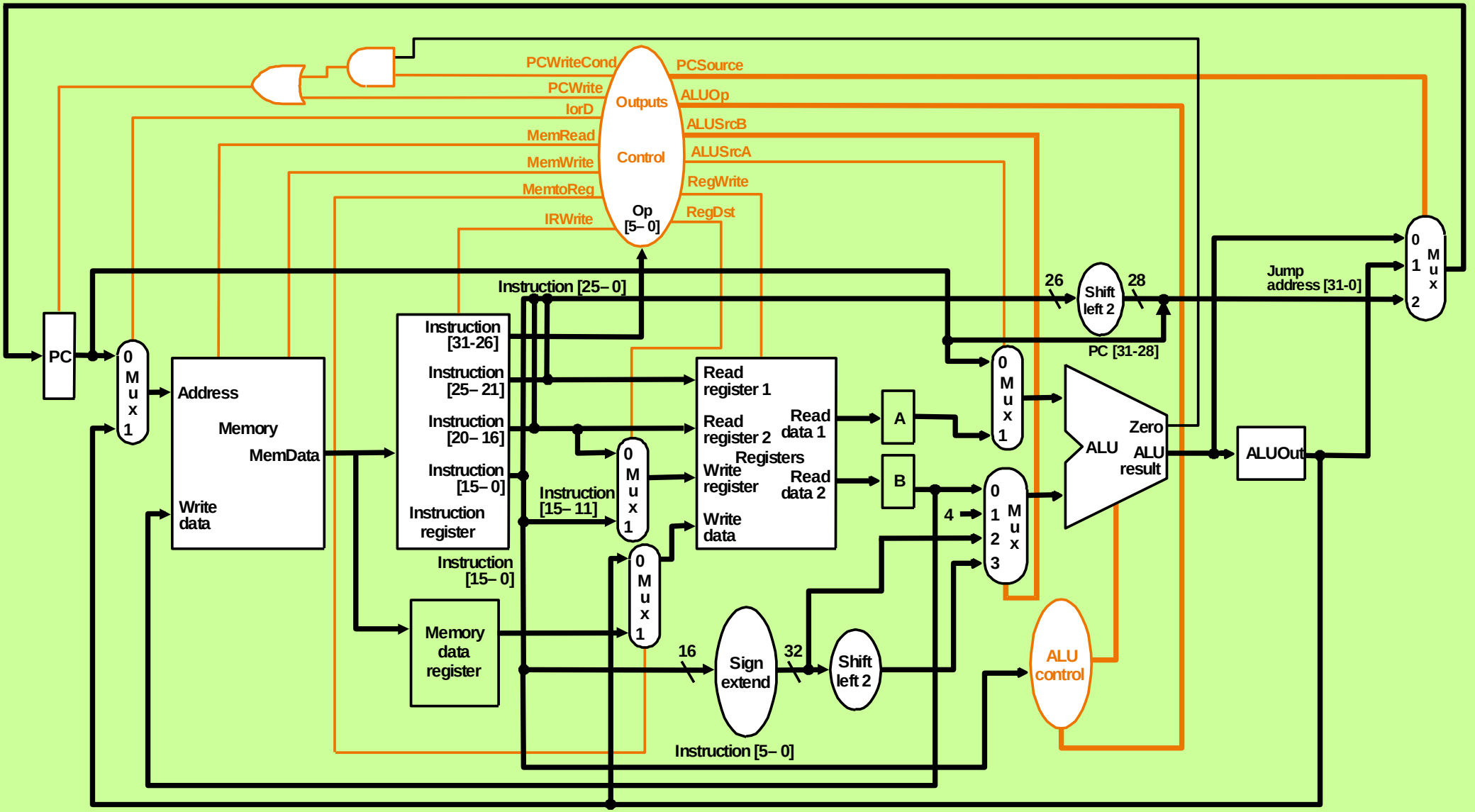
# Multicycle Implementation: Controls added




Because functional units are shared, multiplexers are added to select data between different devices

1. MUX before memory selects either the PC output (fetch instruction) or ALU output (storing data)
2. MUX before "write register" selects write-register number (instruction [15-11] or instruction[20-16])

# Multicycle Implementation: Control Units added



# Action of 1-bit Control Signals


| Control Input | Effect when Deasserted (0) | Effect when asserted (1) |
|---|---|---|
| IorD | PC supplies address to memory (instruction fetch) | ALUout supplies address to memory (lw/sw) |
| MemRead | None | Memory content specified by address is placed on "Memdata" o/p (lw/any instruction) |
| MemWrite | None | I/p "Write data" is stored at specified address (sw) |
| IRWrite | None | "MemData" o/p is written on IR (instruction fetch) |
| RegDst | "Write Register" specified by Instruction[20-16] (lw) | "WriteRegister" specified by Instruction[15-11] (R-type) |
| RegWrite | None | Data from "WriteData" i/p is written on the register specified by "WriteRegister" number |
| ALUSrcA | PC is the first operand in ALU (increment PC) | Register A is the first operand in ALU |
| MemtoReg | "WriteData" of the register file comes from ALUOut | "WriteData" of the register file comes from MDR |
| PCWrite | Operation at PC depends on PCWriteCond and zero output of ALU | PC is written; Source is determined by PCSource |
| PCWriteCond | Operation at PC depends on PCWrite | PC is written if zero o/p of ALU = 1; Source is determined by PCSource |

# Action of 2-bit Control Signals


| Control Input | Value | Effect |
|---|---|---|
| ALUOp | 00 | ALU performs an add operation |
| | 01 | ALU performs a subtract operation |
| | 10 | The function field of Instruction defines the operation of ALU |
| ALUSrcB | 00 | The second operand of ALU comes from Register B |
| | 01 | The second operand of ALU = 4 |
| | 10 | The second operand of ALU is sign extended Instruction[15-0] |
| | 11 | The second operand of ALU is sign extended, 2-bit left shifted Instruction[15-0] |
| PCSource | 00 | Output of ALU (PC + 4) is sent to PC |
| | 01 | Contents of ALUOut (branch target address = PC + 4 + 4 x offset) is sent to PC |
| | 10 | Contents of Instruction[25-0], shift left by 2, and concatenated with the MSB 4-bits of PC is sent to PC (jump instruction) |