# CENG 3420
# Computer Organization & Design
# Lecture 08: Datapath

Textbook: Chapter 4.1-4.4

Zhengrong Wang

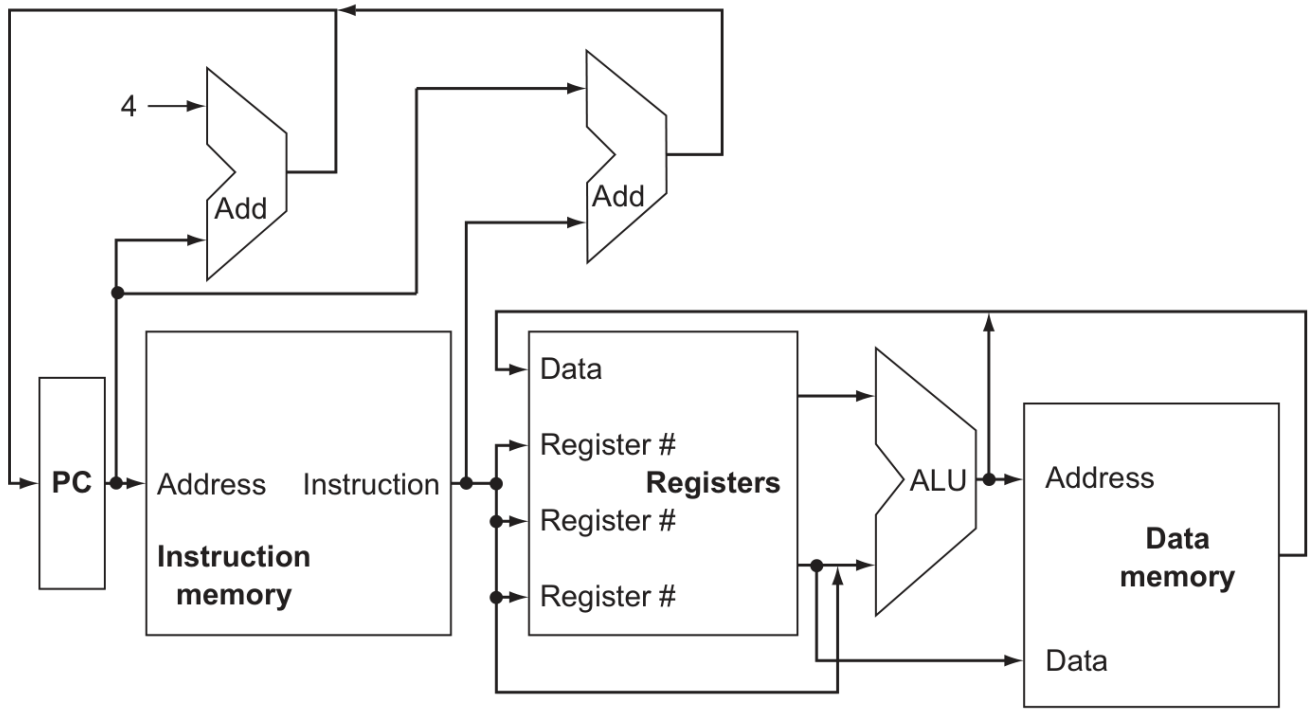
CSE Department, CUHK

zhengrongwang@cuhk.edu.hk

# Overview

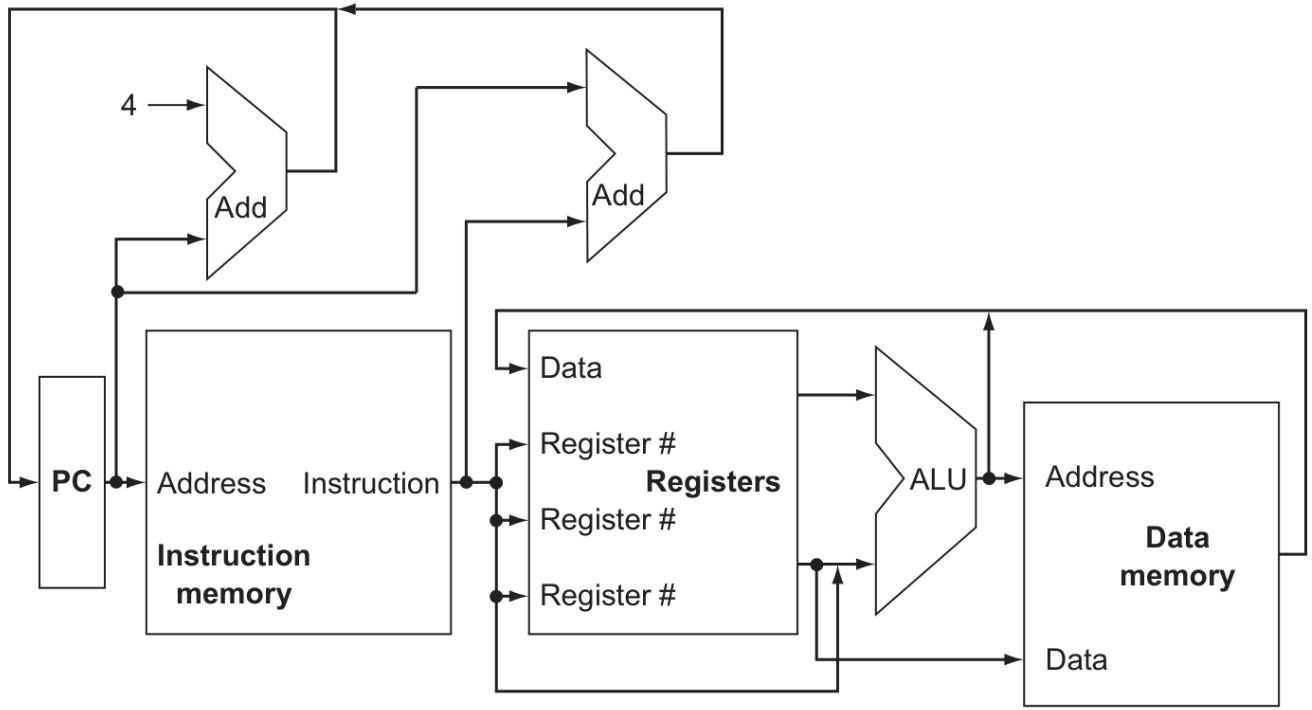
# Processor Overview

- Get instruction from instruction memory at PC (program counter).
- Get source operands from register or immediate number in the instruction.
- Perform the operation, e.g. add, load from memory, store to memory, take a branch.
- Update PC to next instruction (+4 if no branch), and repeat from Step 1.



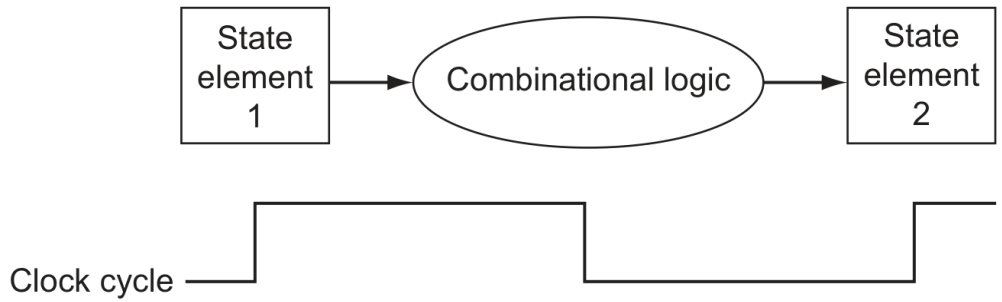
# Implement Minimal RISC-V

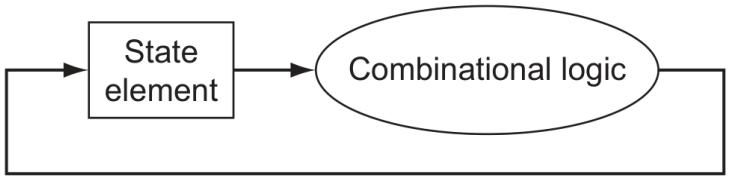
- Arithmetic-logic instructions: add, sub, and, or.
- Memory reference instructions: lw (load word), sw (store word).
- Conditional branch instruction: beq (branch if equal).
- We will see how to add control signals to build datapath for these instructions!



# Logic Design Conventions

- Combinational elements operates on data value:
  - Outputs only depends on the current inputs.
  - E.g., AND gate.
- State elements contains the state:
  - At least two inputs (new state and clock) and one output (old state).
  - At each clock edge (posedge or negedge), the state is updated.
- All elements have some latency – limits the frequency of the processor.
  - Otherwise, state element 2 may not yet get the correct value.

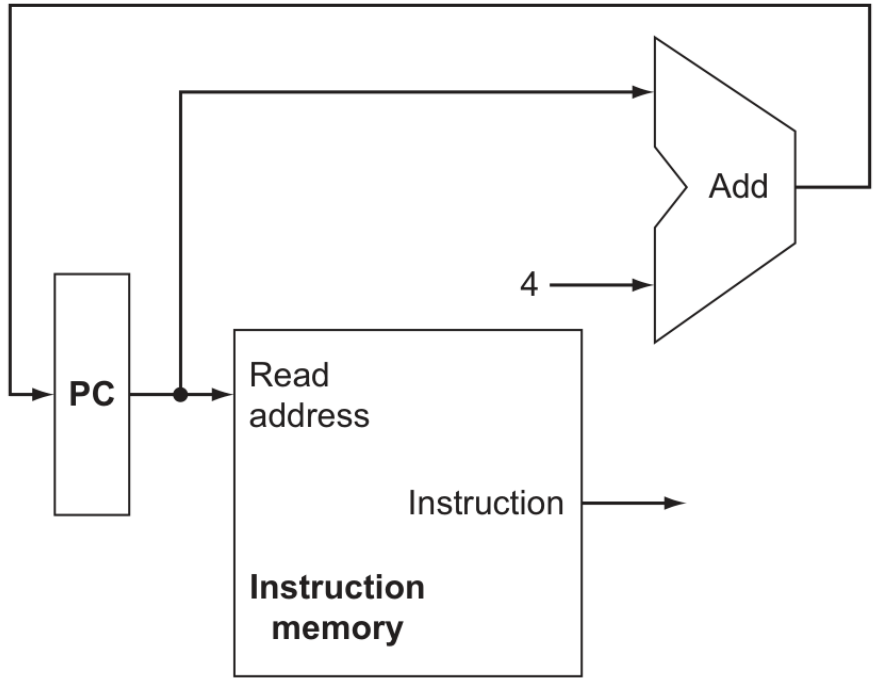


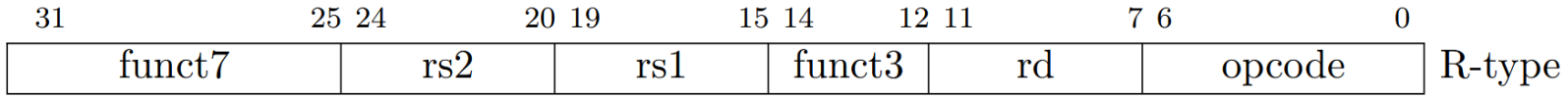
# Simple Datapath

# Instruction Fetch

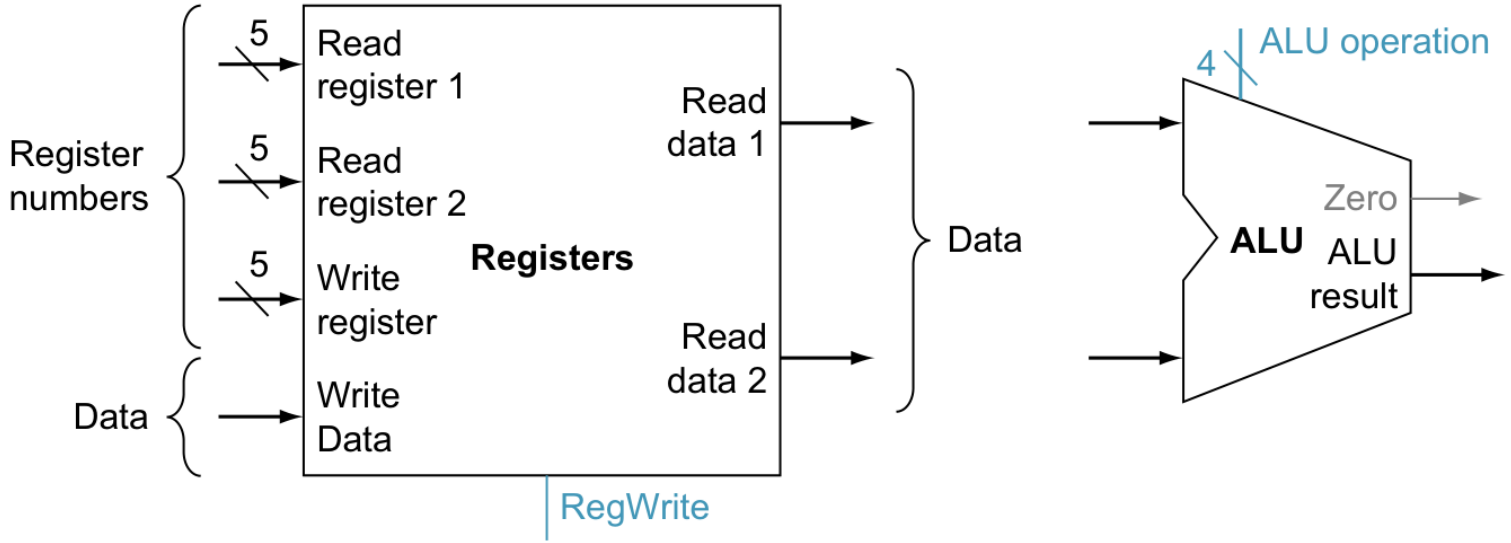
- Three components: PC (state), instruction memory (state), adder.
  - Read one instruction at PC from the instruction memory.
  - Increment PC by 4 bytes (32-bit) for next instruction.
- Shared among all instructions.



# R-Type Instructions




- Always read two registers (rs1 and rs2) and write one register (rd).
- Besides fetching instructions, we need two more components.
  - Register file: Holds 32 32-bit value, with RegWrite controlling whether to write.
  - ALU: 4-bit ALU operation to control the behavior (add, sub, etc., see Lecture 06).

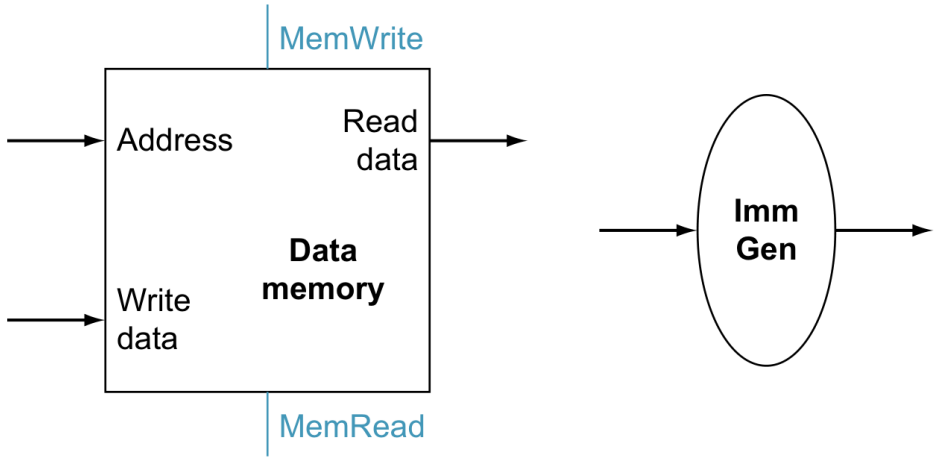


# Memory Load and Store Instructions

| imm[11:0] | rs1 | funct3 | rd | opcode | I-type |
|---|---|---|---|---|---|

| imm[11:5] | rs2 | rs1 | funct3 | imm[4:0] | opcode | S-type |
|---|---|---|---|---|---|---|

- Load and store operations
  - Sign-extend 12-bit imm to 32-bit offset, compute address = base (in rs1) + offset.
  - For load, rd = memory[address]; For store, memory[address] = rs2.
- Besides ALU for address generation, we need two more components:
  - Immediate generation unit (Imm Gen) for sign-extension.
  - Data memory unit with read and write support.

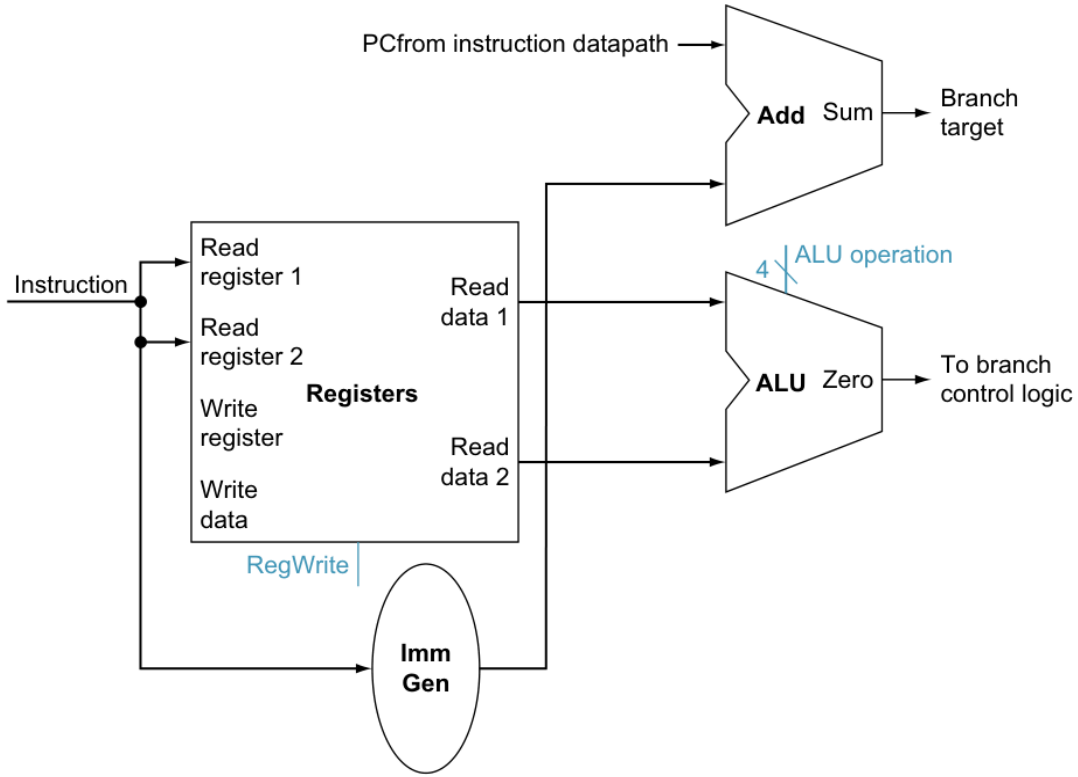


# Branch Instruction

| imm[12] | imm[10:5] | rs2 | rs1 | funct3 | imm[4:1] | imm[11] | opcode | B-type |
|---|---|---|---|---|---|---|---|---|

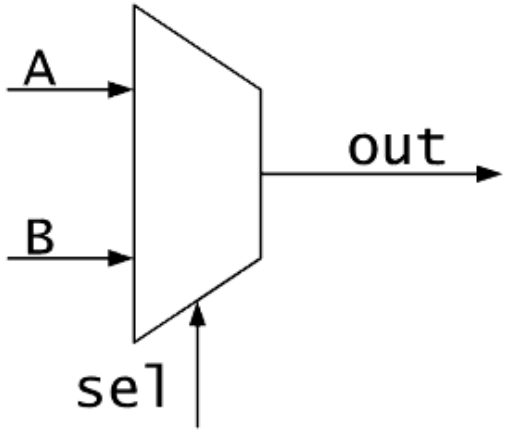
- Branch if equal (beq) :
  - Jump if rs1 equals rs2.
  - Target address = PC + offset.
  - Offset from sign-extension of imm to 32-bit.
  - Note: offset is encoded in factor of 2,

  i.e., imm[0] = 0 → PC always aligns to 2-byte.
- Portion of the datapath for beq instruction
  - Imm Gen for sign-extension.
  - Register file for two compared operands.
  - ALU to test equality.
  - Separate adder to compute target address.

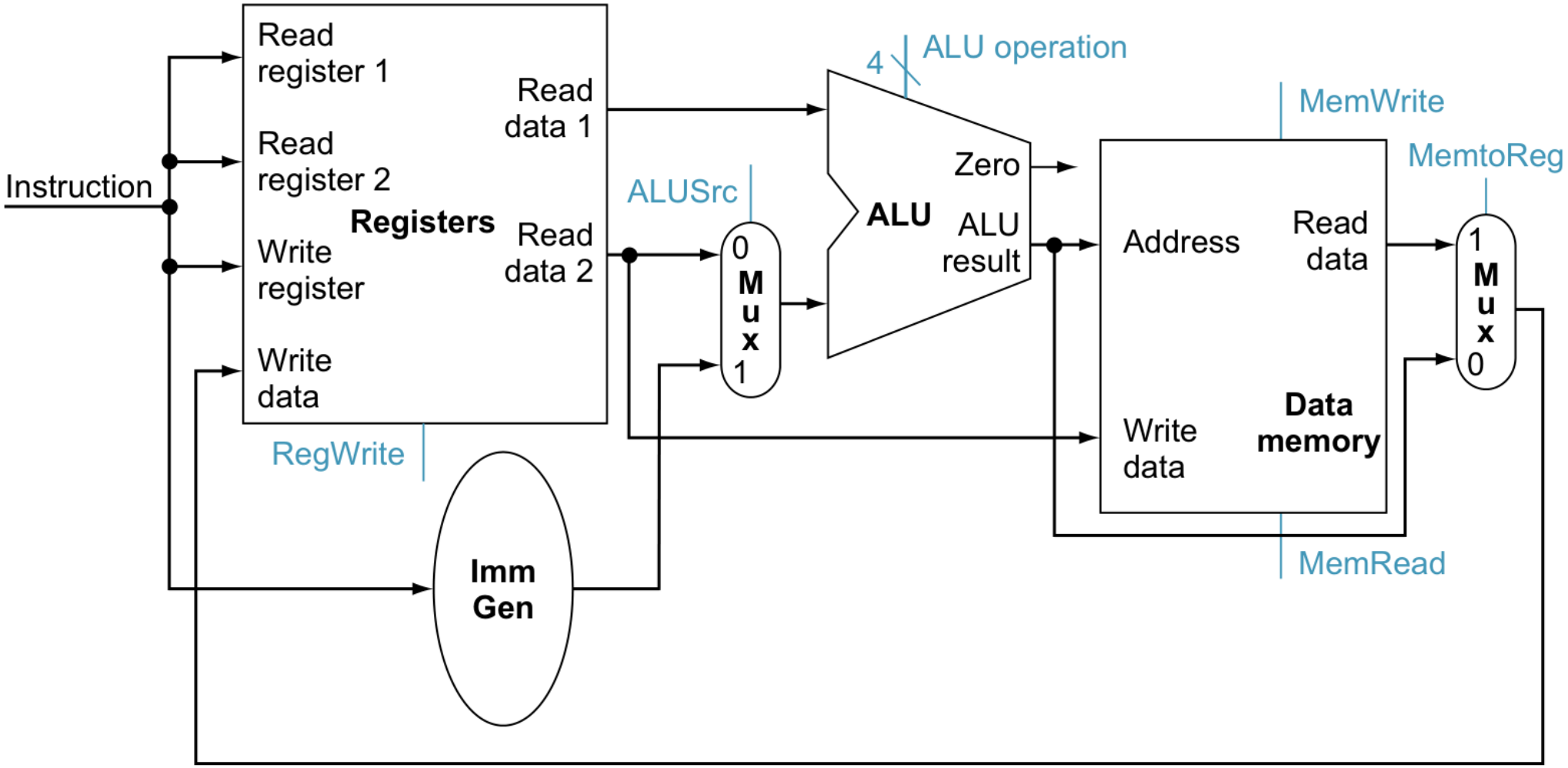


# Shared Datapath

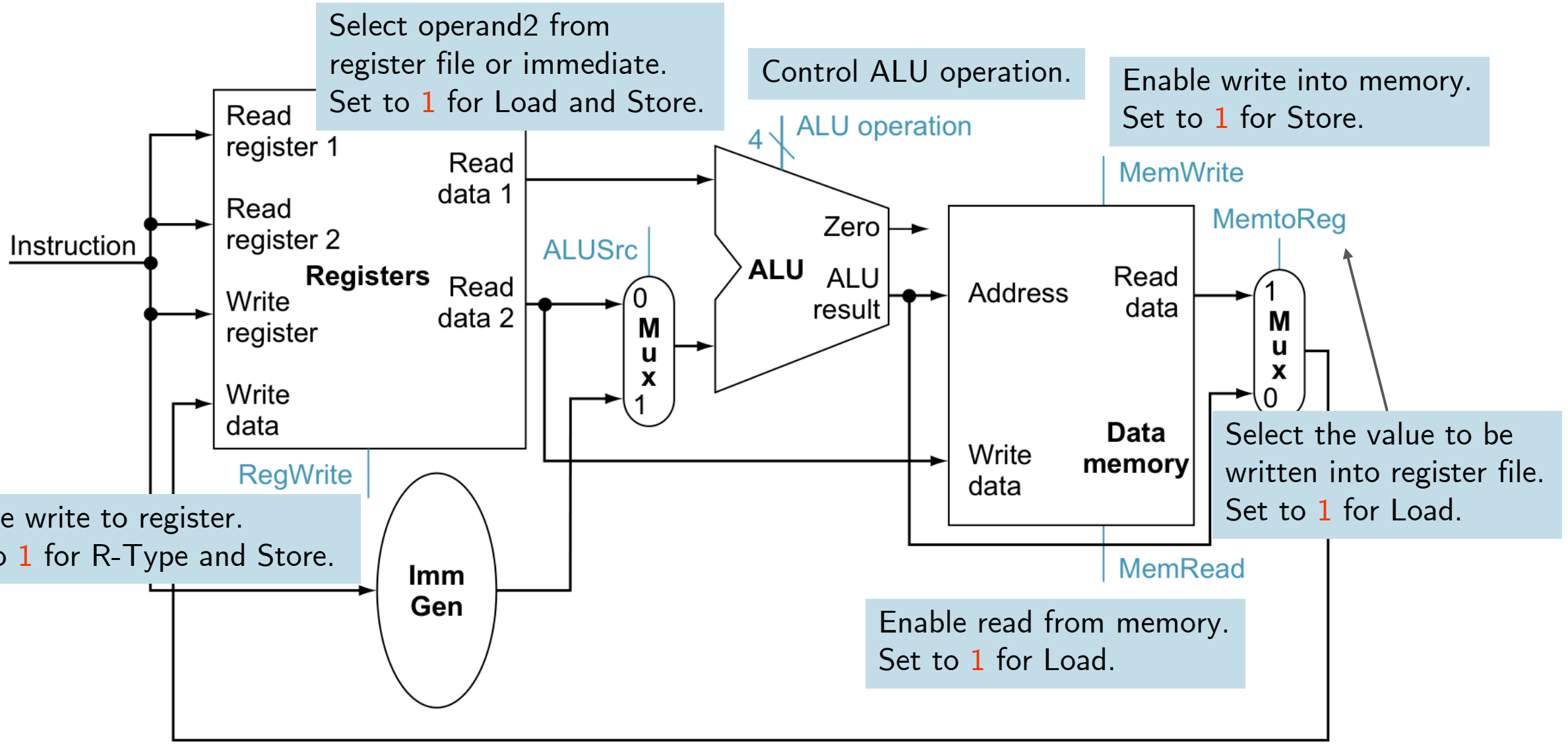
- Instructions share many components in their datapath.
  - They all use the ALU for some operation.
  - They all need PC and instruction memory.
  - They perform sign-extension on the immediate (except R-type instructions).
- We want to reuse components in their datapath.
  - Save hardware resources on area and power.
  - Increase hardware utilization.
- But we need to coordinate shared components.
  - E.g., 4-bit ALU op to dictate what operation the ALU to perform.
  - Essentially multiplexors to select value based on the control signal.



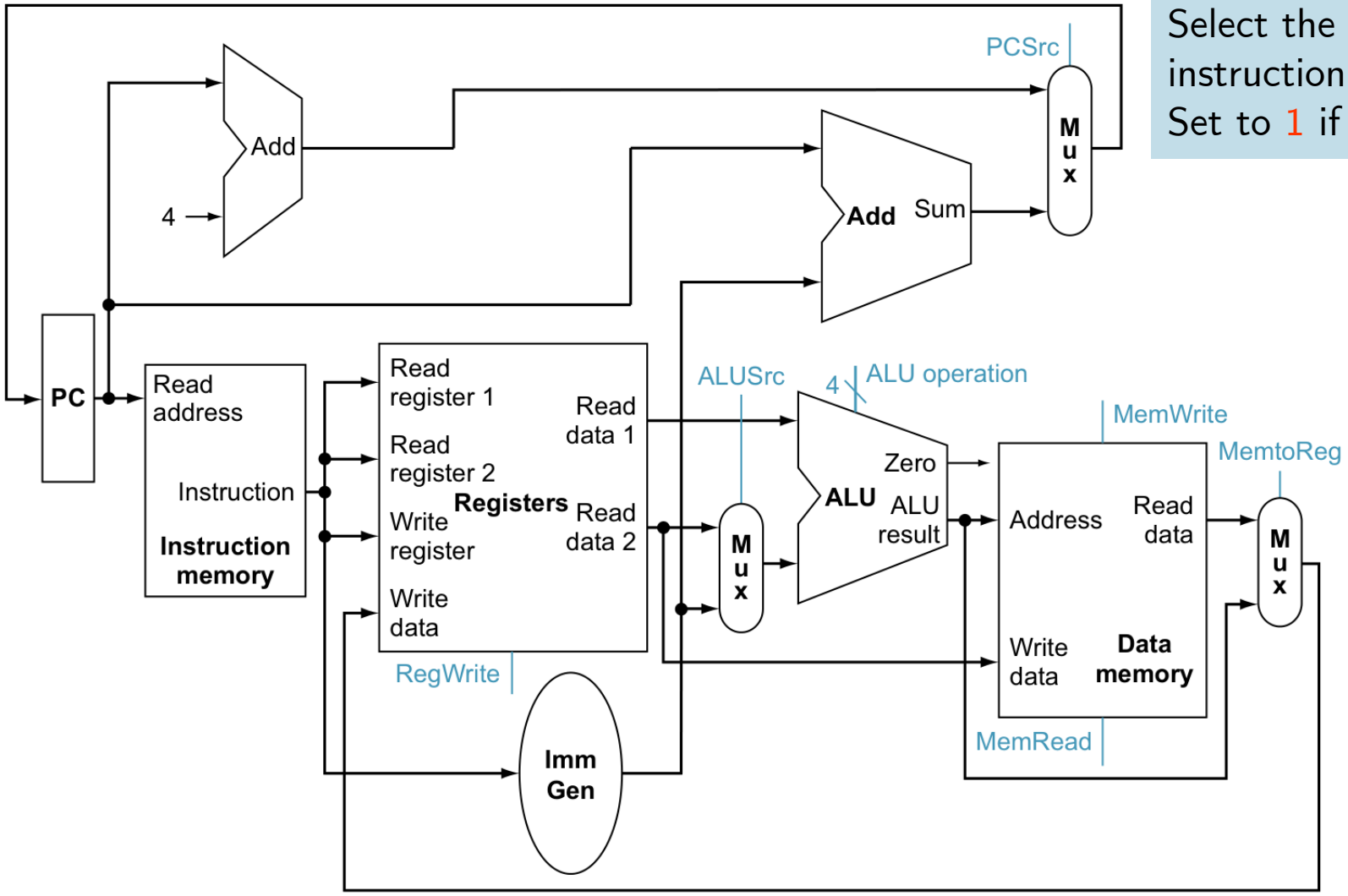
# Combine R-Type and Memory Instructions



# Combine R-Type and Memory Instructions



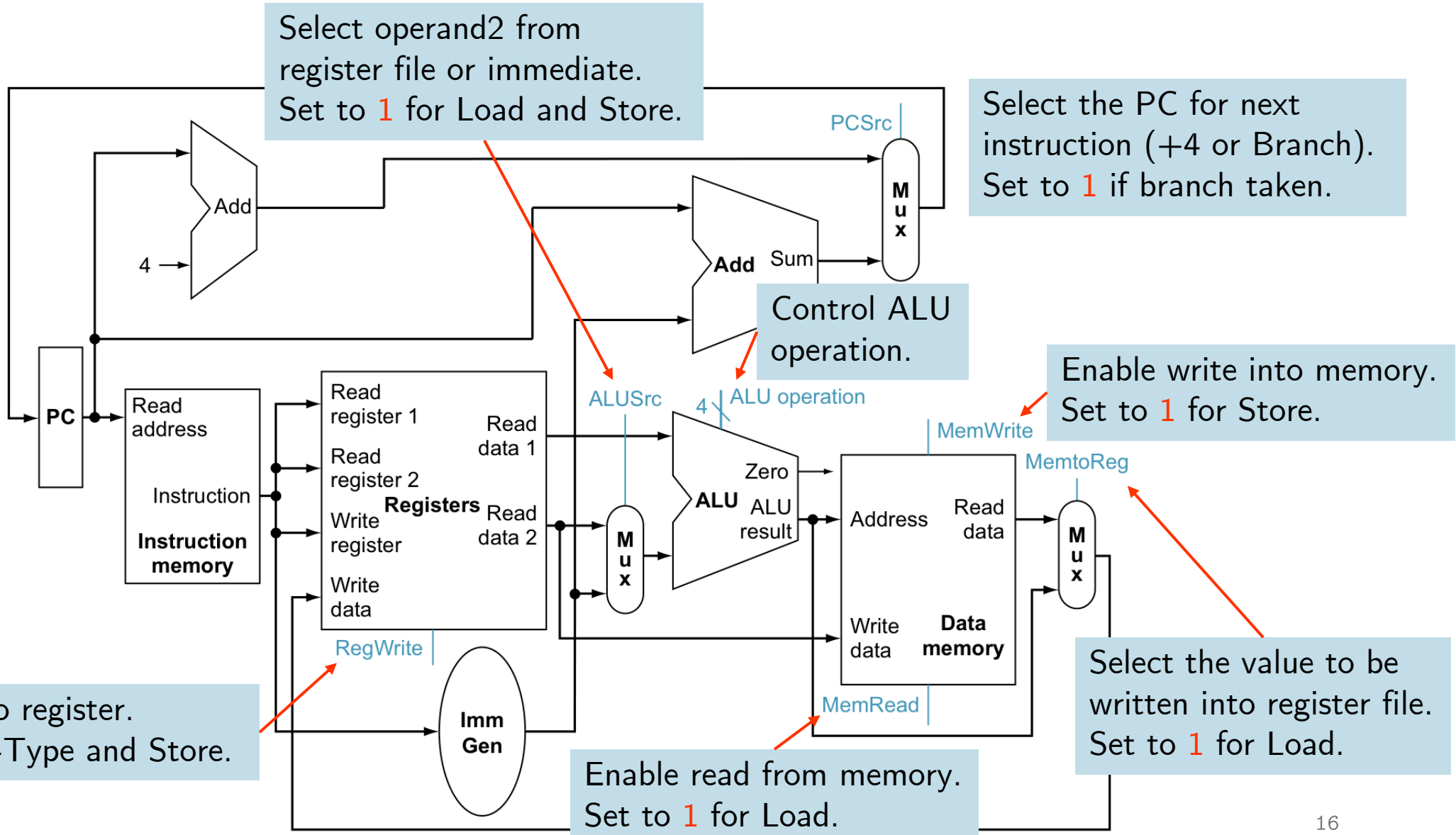
# Add Branch Datapath




Select the PC for next instruction (+4 or Branch). Set to 1 if branch taken.

# Control Unit

# How to Generate Control Signals?



# Detailed Meaning of Control Signals

- The main control unit need to set these based on instruction opcode/funct.

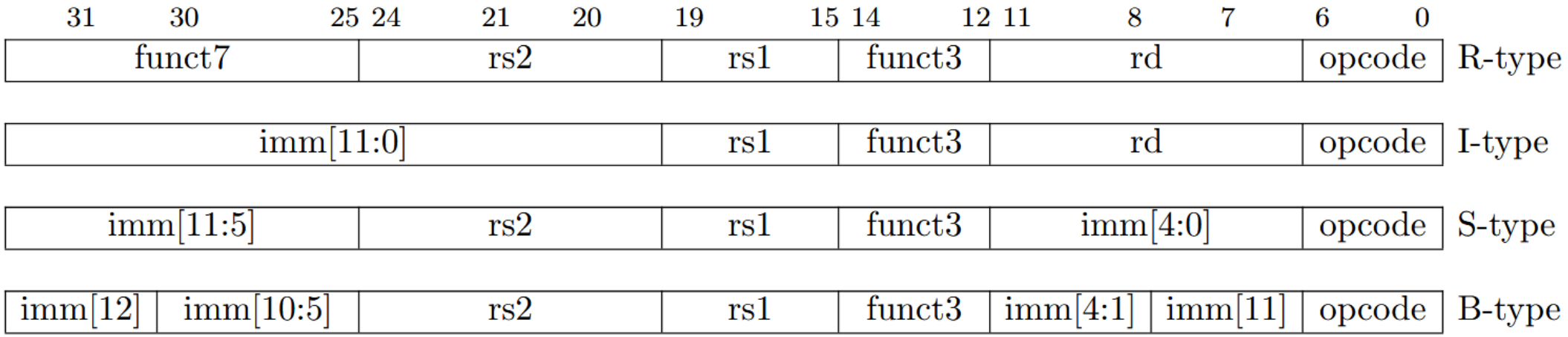
| Signal name | Effect when deasserted | Effect when asserted |
|---|---|---|
| RegWrite | None. | The register on the Write register input is written with the value on the Write data input. |
| ALUSrc | The second ALU operand comes from the second register file output (Read data 2). | The second ALU operand is the sign-extended, 12 bits of the instruction. |
| PCSrc | The PC is replaced by the output of the adder that computes the value of PC + 4. | The PC is replaced by the output of the adder that computes the branch target. |
| MemRead | None. | Data memory contents designated by the address input are put on the Read data output. |
| MemWrite | None. | Data memory contents designated by the address input are replaced by the value on the Write data input. |
| MemtoReg | The value fed to the register Write data input comes from the ALU. | The value fed to the register Write data input comes from the data memory. |

# Two-Level Control Logic for ALU

- Two-level control logic: main control unit → ALU control unit → ALU.
  - Main control unit generates 2-bit ALUOp based on instruction type.
  - ALU control unit generates 4-bit ALU control based on ALUOp and funct.
- Simplifies hardware complexity → less latency → higher frequency.
  - Main control unit only checks opcode field.
  - ALU control unit only checks funct7 and func3 for R-type.

| Instruction opcode | ALUOp | Operation | Funct7 field | Funct3 field | Desired ALU action | ALU control input |
|---|---|---|---|---|---|---|
| lw | 00 | load word | XXXXXXX | XXX | add | 0010 |
| sw | 00 | store word | XXXXXXX | XXX | add | 0010 |
| beq | 01 | branch if equal | XXXXXXX | XXX | subtract | 0110 |
| R-type | 10 | add | 0000000 | 000 | add | 0010 |
| R-type | 10 | sub | 0100000 | 000 | subtract | 0110 |
| R-type | 10 | and | 0000000 | 111 | AND | 0000 |
| R-type | 10 | or | 0000000 | 110 | OR | 0001 |

# Observation on Instruction Format

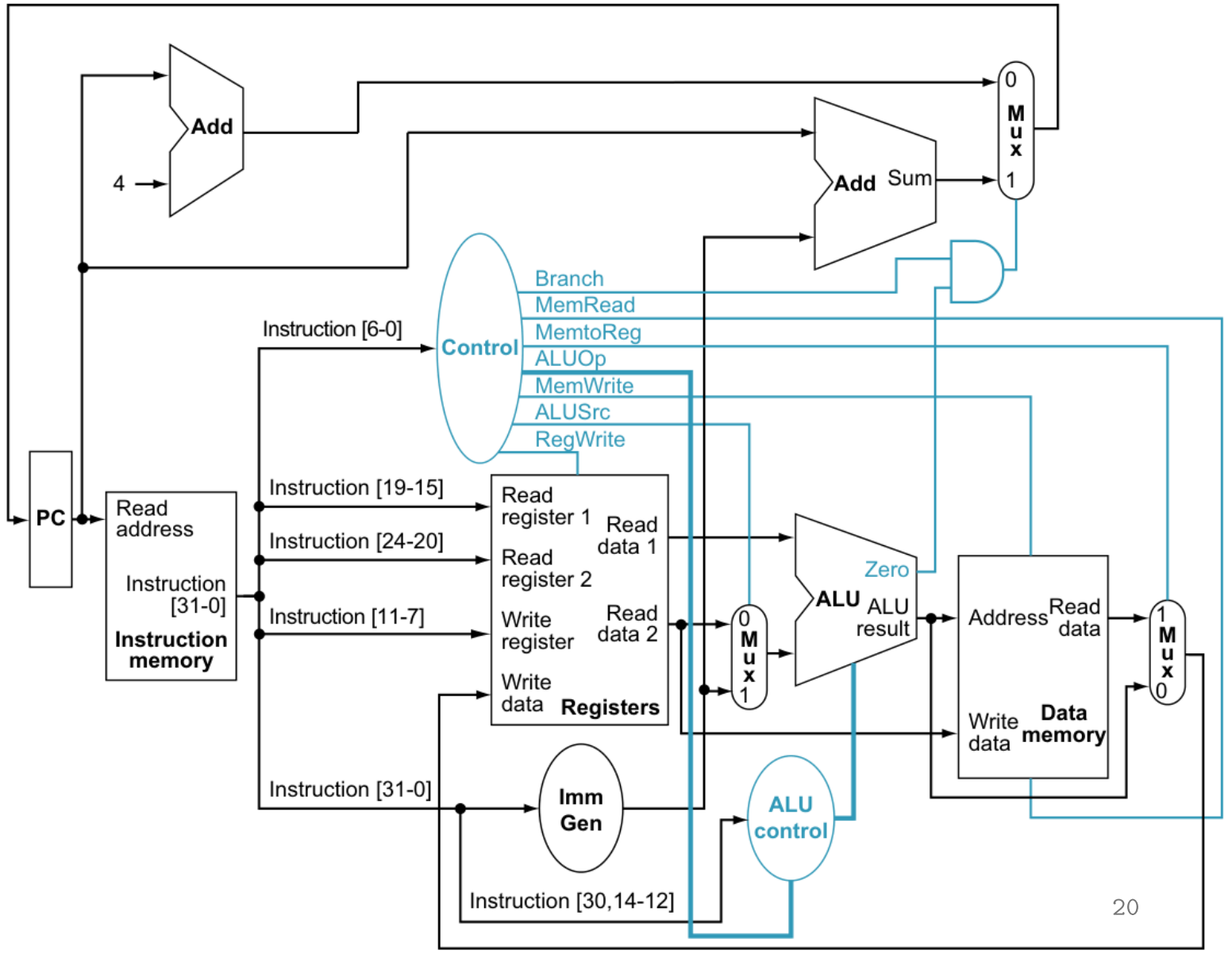



- The opcode field is always in bits 6:0. Depending on the opcode, the funct3 field (bits 14:12) and funct7 field (bits 31:25) serve as an extended opcode field.
- The first register operand is always in bit positions 19:15 (rs1) for R-type and branch instructions. This field also specifies the base register for load and store instructions.
- The second register operand is always in bit positions 24:20 (rs2) for R-type and branch instructions. This field also specifies the register operand that gets copied to memory for store.
- Another operand can also be a 12-bit offset for branch or load-store instructions.
- The destination register is always in bit positions 11:7 (rd) for R-type and load instructions.

# Simple Datapath with Control Unit

- Main control unit.
- ALU control.
- 2-bit ALUOp.
- 4-bit ALU control.
- Other control signals are 1-bit.

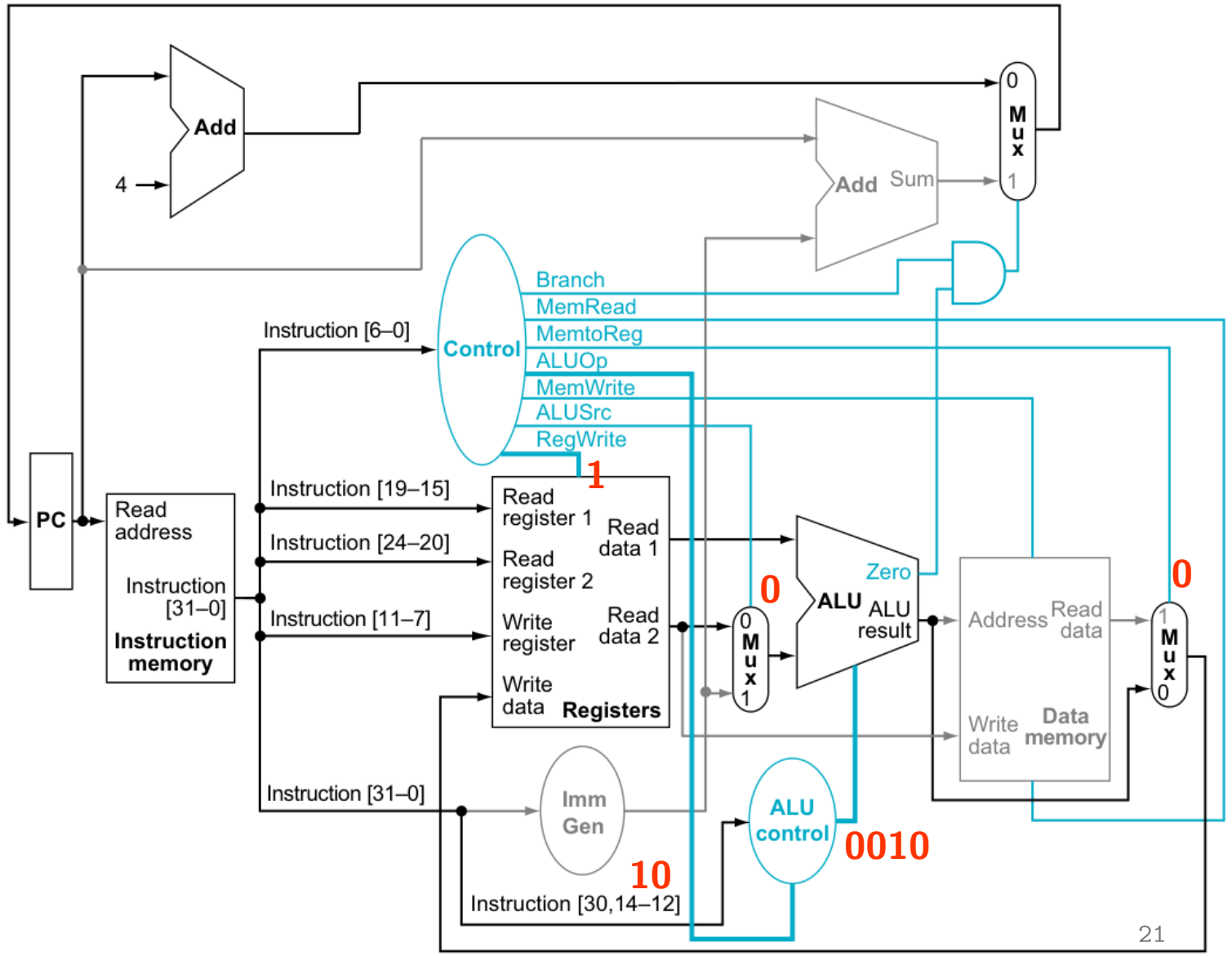
- Let's see some examples.



# Example: R-Type of add x1, x2, x3

- Black line – selected data.
- Gray line – unselected data.

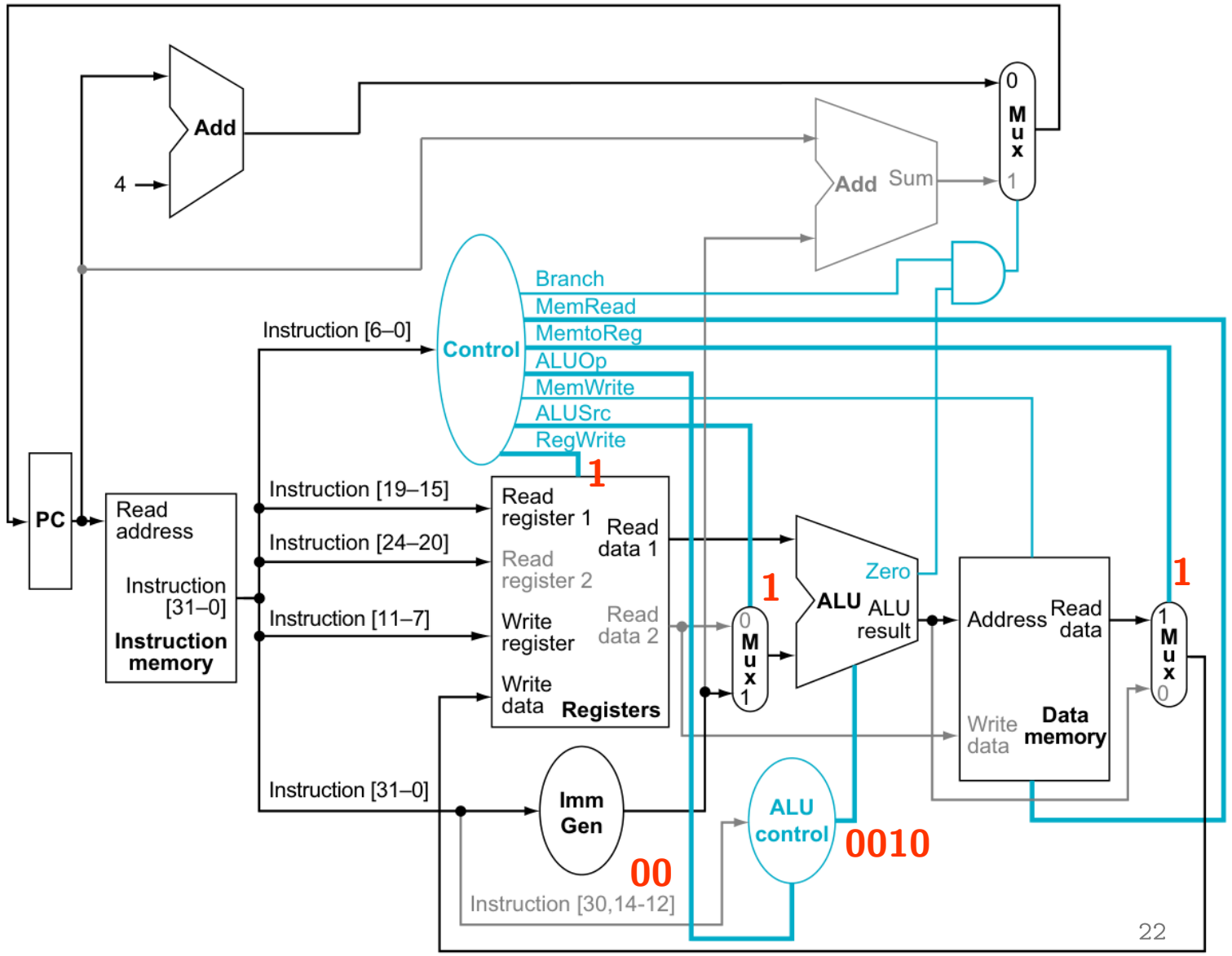
- ALUOp = 10 (R-type)
- ALUControl = 0010 (add)
- ALUSrc = 0
  - 2nd operand from rs2.
- MemtoReg = 0
  - Result from ALU
- RegWrite = 1
  - Write to rd.
- Others = 0
  - Not load/store/branch.



# Example: Load Instruction

- Black line – selected data.
- Gray line – unselected data.

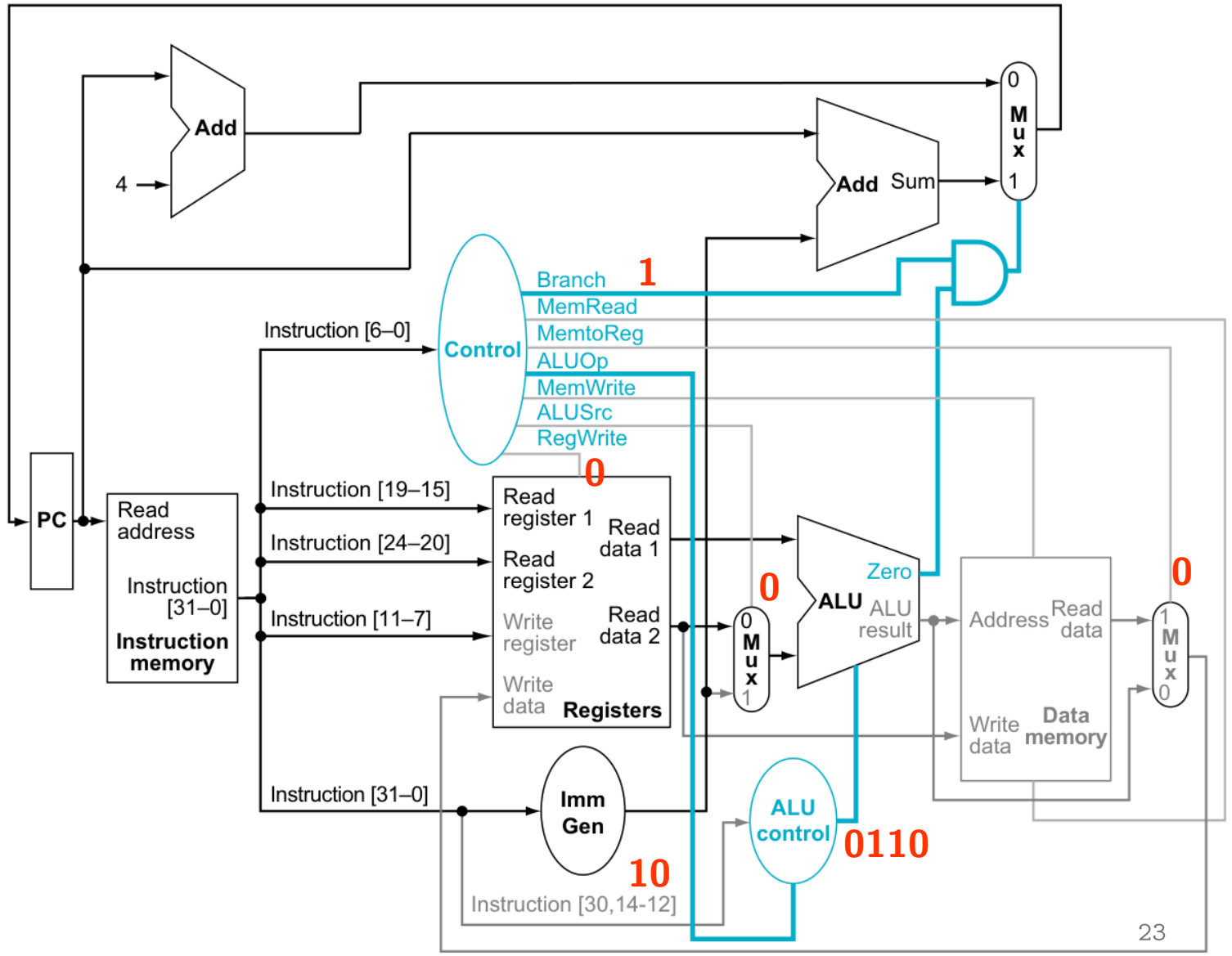
- ALUOp = 00 (lw/sw)
- ALUControl = 0010 (add)
- ALUSrc = 1
  - 2nd operand from imm.
- MemRead = 1
  - To read from memory
- MemtoReg = 1
  - Result from memory
- RegWrite = 1
  - Write to rd.



# Example: Branch Instruction

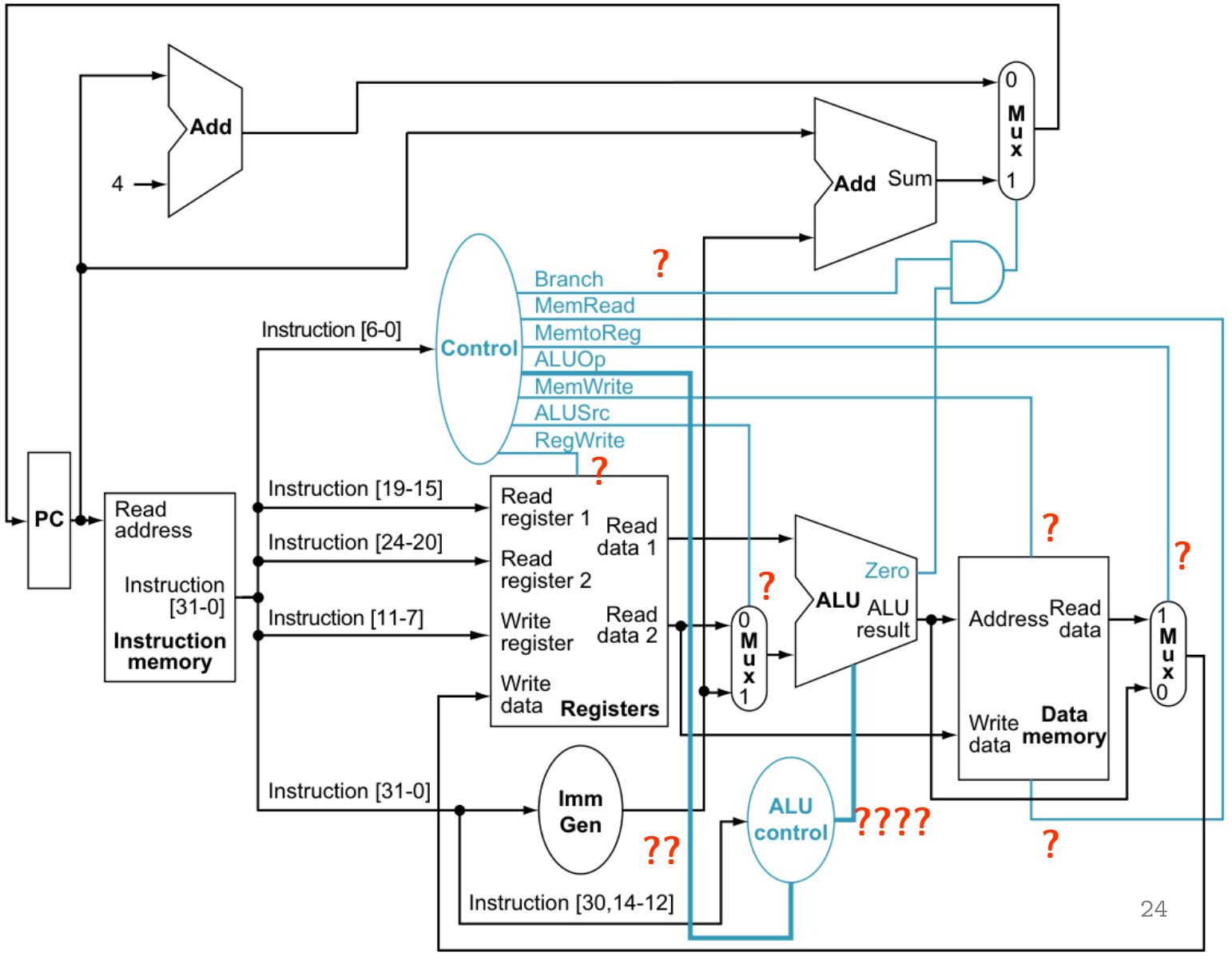
- Black line – selected data.
- Gray line – unselected data.

- ALUOp = 01 (beq)
- ALUControl = 0110 (sub)
- Branch = 1
  - And with zero.
  - Pick update PC.



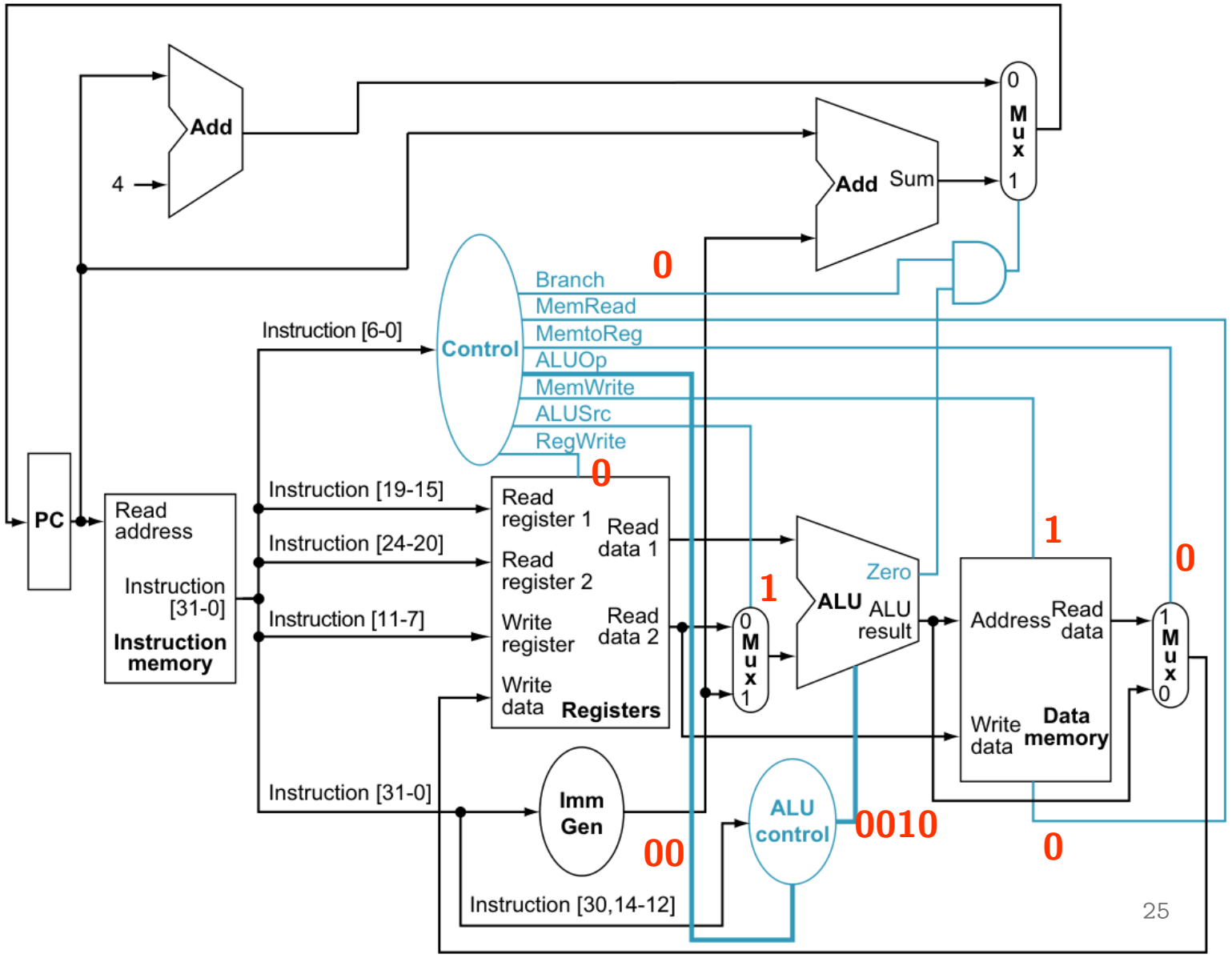
# Exercise: Show Control Signals of Store

- What sw does
  - Extend imm to offset.
  - Add offset with rs1.
  - Write rs2 to memory.
- What are the signals?



# Exercise: Show Control Signals of Store

- ALUOp = 00 (lw/sw)
- ALUControl = 0010 (add)
- ALUSrc = 1
  - 2nd operand from imm.
- MemWrite = 1
  - Write to memory



# Summarize Main Control Unit

- Generate control signals based on instruction opcode (bit 0-6).
- Final truth table of main control unit:

| Input or output | Signal name | R-format | lw | sw | beq |
|---|---|---|---|---|---|
| Inputs | I[6] | 0 | 0 | 0 | 1 |
| | I[5] | 1 | 0 | 1 | 1 |
| | I[4] | 1 | 0 | 0 | 0 |
| | I[3] | 0 | 0 | 0 | 0 |
| | I[2] | 0 | 0 | 0 | 0 |
| | I[1] | 1 | 1 | 1 | 1 |
| | I[0] | 1 | 1 | 1 | 1 |
| Outputs | ALUSrc | 0 | 1 | 1 | 0 |
| | MemtoReg | 0 | 1 | X | X |
| | RegWrite | 1 | 1 | 0 | 0 |
| | MemRead | 0 | 1 | 0 | 0 |
| | MemWrite | 0 | 0 | 1 | 0 |
| | Branch | 0 | 0 | 0 | 1 |
| | ALUOp1 | 1 | 0 | 0 | 0 |
| | ALUOp0 | 0 | 0 | 0 | 1 |