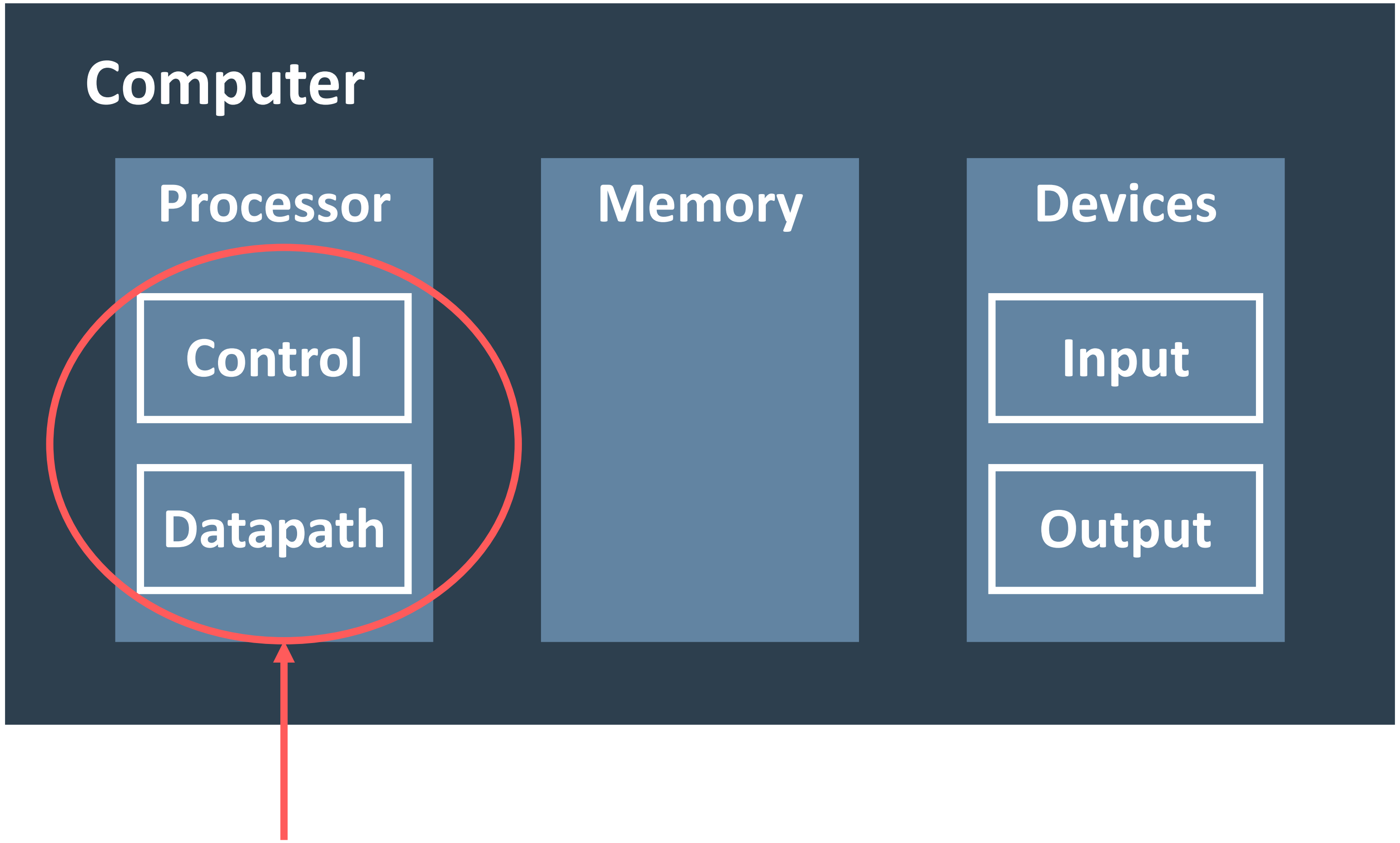
Computer Architecture

4. The Processor: Datapath and Control

Young Geun Kim
(younggeun_kim@korea.ac.kr)
Intelligent Computer Architecture & Systems Lab.

Computer Organization



Big Picture: Processor Implementation

- **Key ideas**
  - Concept of datapath and control
  - Where the instruction and data bits go

- **Approach**
  - Start with a simple implementation and iteratively improve it

- **We will examine three RISC-V implementations**
  - A simplified single-cycle version
  - An advanced multi-cycle version
  - A more realistic pipelined version

Subset of Instructions

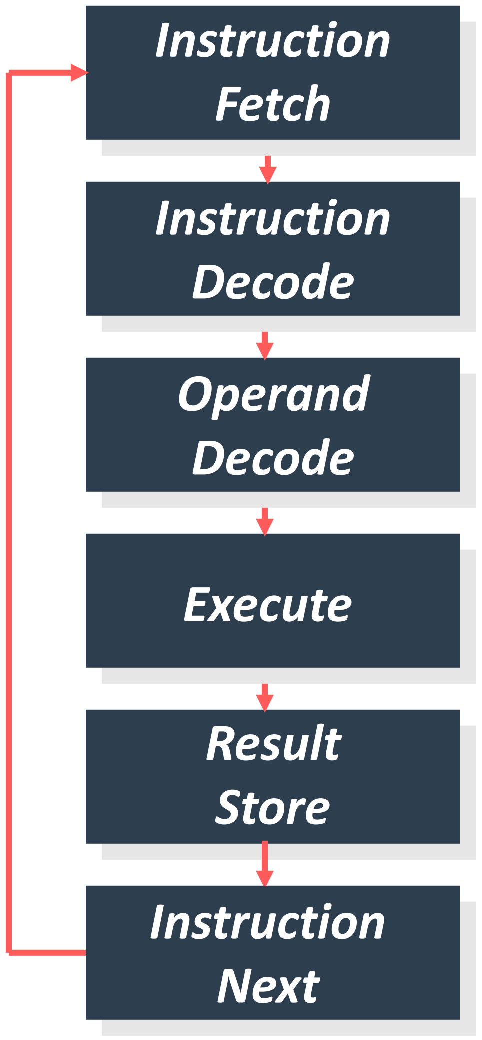
- **To simplify out study of processor design, we will focus on simple subset of RV32I, yet covers most aspects**
  - Data transfer: *lw*, *sw*
  - Arithmetic/logical: *add*, *sub*, *and*, *or*
  - Control transfer: *beq*

RISC-V Format Review

	31					0	
R-type	**funct7**	**rs2**	**rs1**	**funct3**	**rd**	**opcode**	**ALU**
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
I-type	**imm[11:0]**		**rs1**	**funct3**	**rd**	**opcode**	**Load** **ALU with imm.**
S-type	**imm[11:5]**	**rs2**	**rs1**	**funct3**	**imm[4:0]**	**opcode**	**Store**
SB-type	**imm[12,10:5]**	**rs2**	**rs1**	**funct3**	**imm[4:1,11]**	**opcode**	**Branch**
U-type	**imm[31:12]**				**rd**	**opcode**	**Upper imm.**
UJ-type	**imm[20,10:1,11,19:12]**				**rd**	**opcode**	**Jump**

Execution Cycle

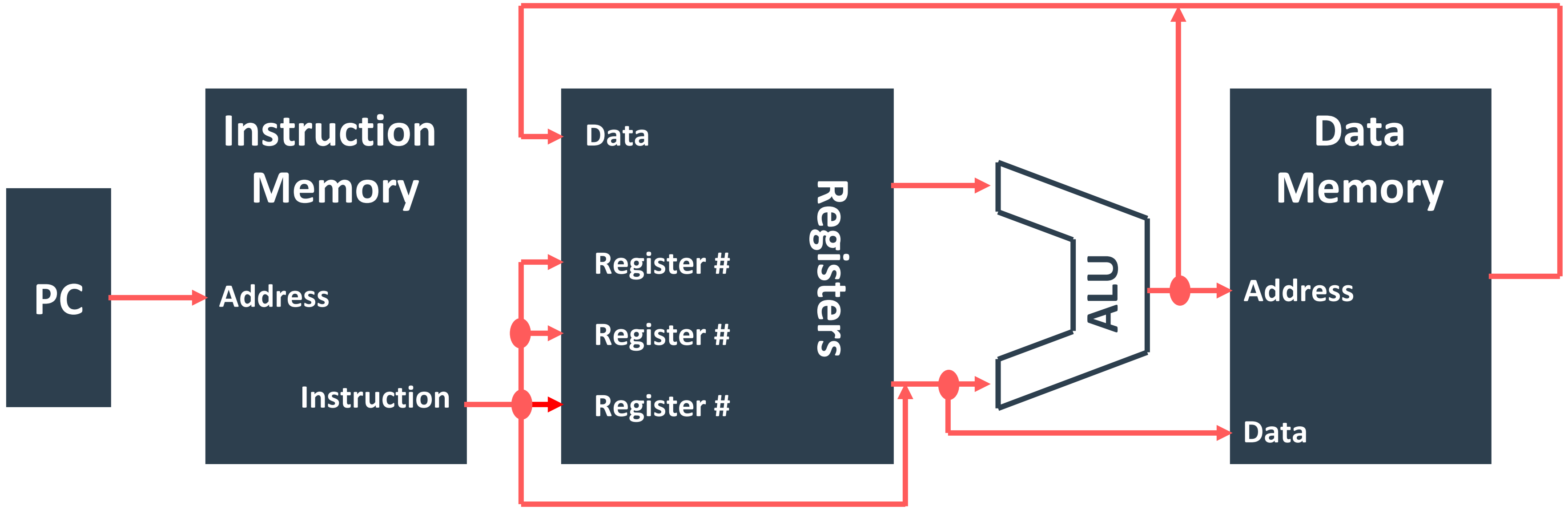
- The lifecycle of an instruction



- *Obtain instruction from instruction memory*
- *Determine required actions* by opcode
- *Locate and obtain operand data* source, imm ...
- *Compute result value or status*
- *Deposit results in storage for later use* mem or reg
- *Determine the following instruction* pc += 4 or branch inst.

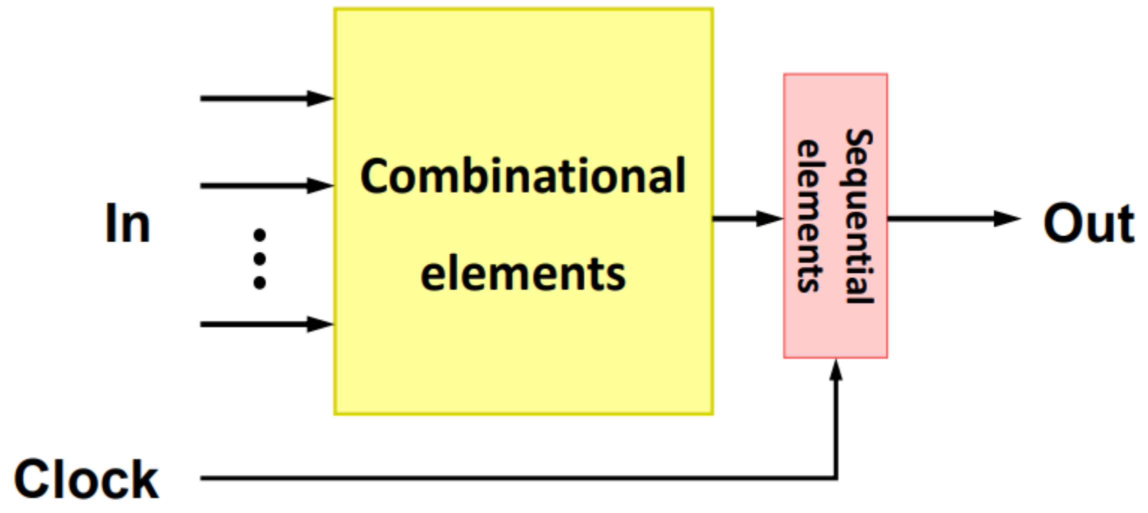
Implementation Overview

- Data flows through memory and functional units



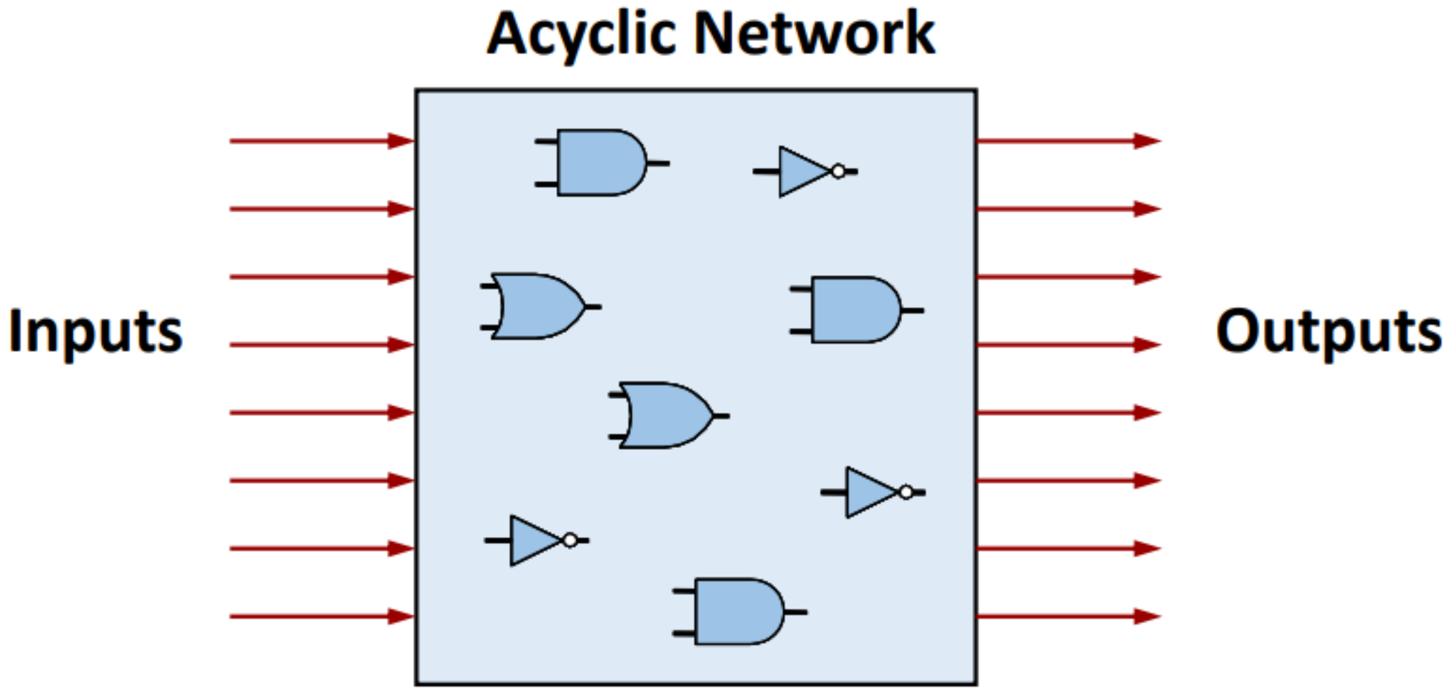
Digital Systems

- **Three components required to implement a digital system**
  - Combinational elements
    - Output is dependent only on current inputs
    - E.g., ALU
  - Sequential elements
    - Element contains state information (i.e., memory element)
    - E.g., registers
  - Clock signals regulate the updating of the memory elements



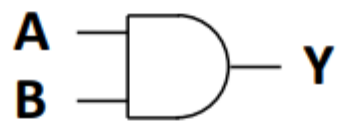
Combinational Elements

- **Acyclic network of logic gates**
  - Continuously responds to changes on primary inputs
  - Primary outputs become (after some delay) functions of primary inputs

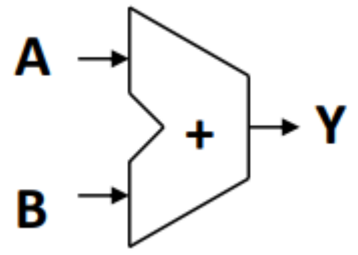


Combinational Elements: Examples

- **AND-gate**
  - Y = A & B

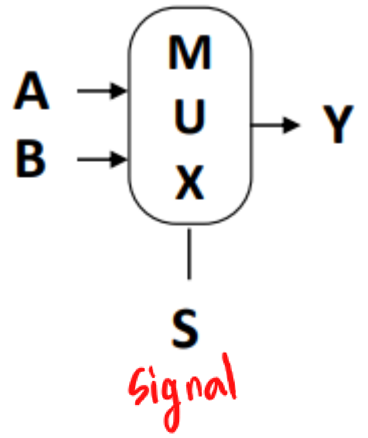


- **Adder**
  - Y = A + B

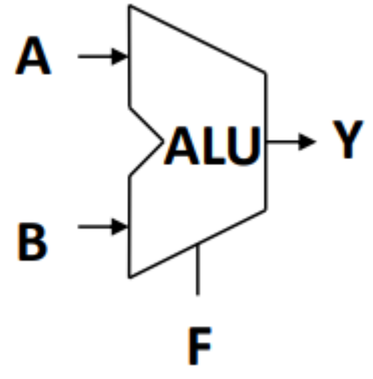


✓

- **Multiplexer** signal에 따라 결정
  - Y = S ? A : B



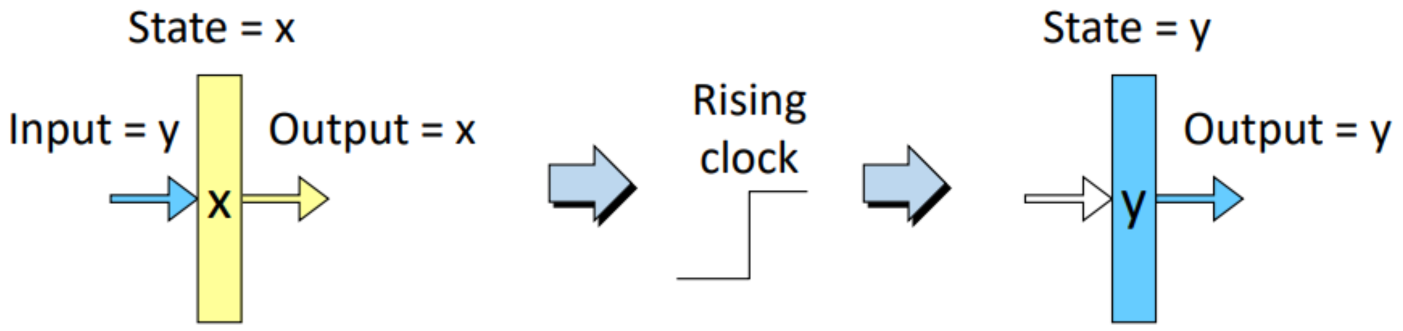
- **Arithmetic/Logic Unit**
  - Y = F (A, B)



Storage Elements: Register

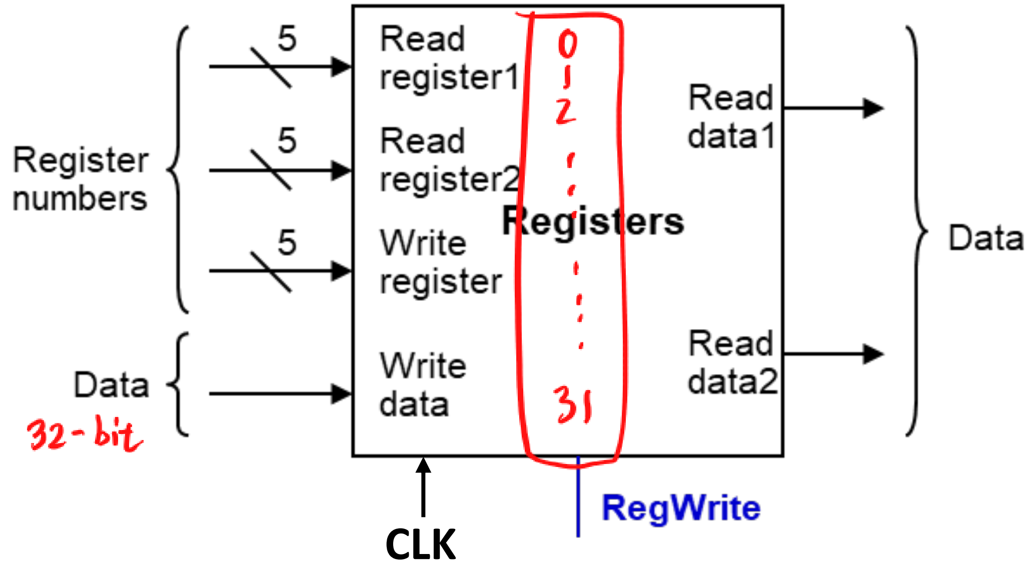
- **Register**
  - Based on the D Flip Flops
    - N-bit input and output
    - Write enable input
  - Write Enable:
    - 0: Data Out will not change
    - 1: Data Out will become Data in

  - Stored data changes only on rising (or falling) clock edge



Storage Elements: Register File

- **Register File consists of 32 registers**
  - Two 32-bit output busses:
    - Read data1 and Read data2
  - One 32-bit input bus: Write data
  - x0 hard-wired to value 0

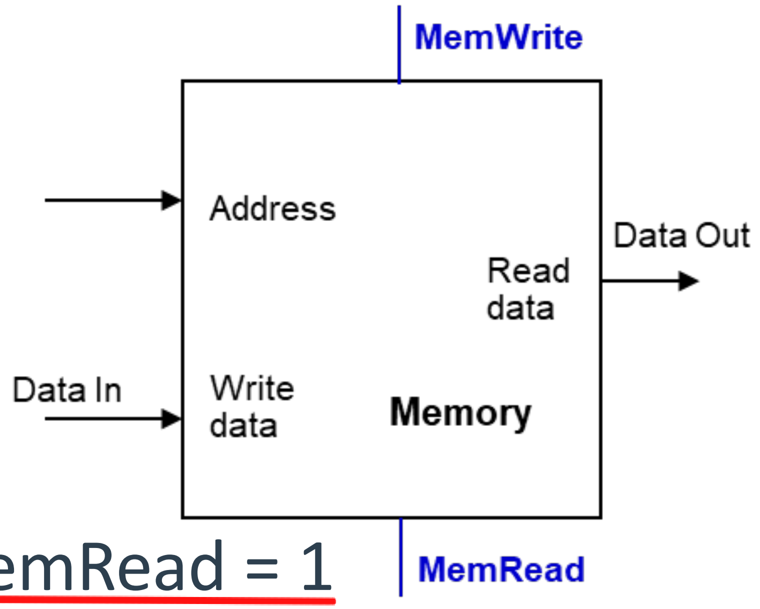


- **Register is selected by:**
  - Read register1 selects the register to put on Read data1
  - Read register2 selects the register to put on Read data2
  - Write register selects the register to be written with Write data when RegWrite = 1 / RegWrite = 0 ⇒ No writing

- **Clock input (CLK)**
  - The CLK input is a factor only for write operation

Storage Elements: Memory

- **Memory has two busses**
  - One output bus: Read data (Data Out)
  - One input bus: Write data (Data In)



- **Address**
  - Selects the word to put on Data Out when MemRead = 1
  - The word to be written via the Data In when MemWrite = 1

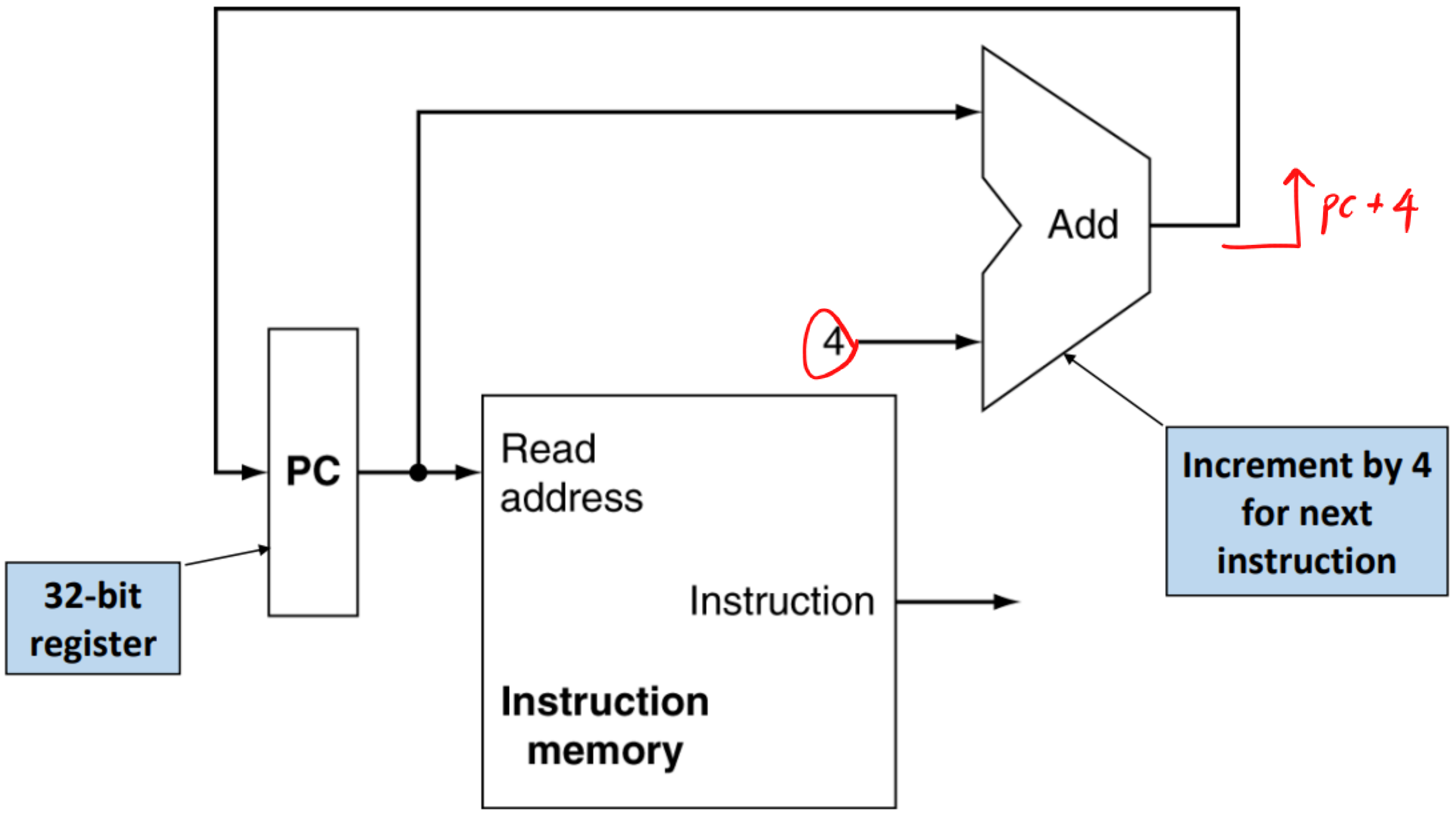
- **Clock input (CLK)**
  - The CLK input is a factor only for write operation
  - During read, behaves as combinational logic block
    - Valid address -> Data Out valid after "access time"

Instruction Fetch (IF) RTL

- **Common RTL operations**
  - Fetch instruction
    - Mem[PC]; // fetch instruction from instruction memory
  - Update program counter
    - PC <- PC + 4; // calculate next address

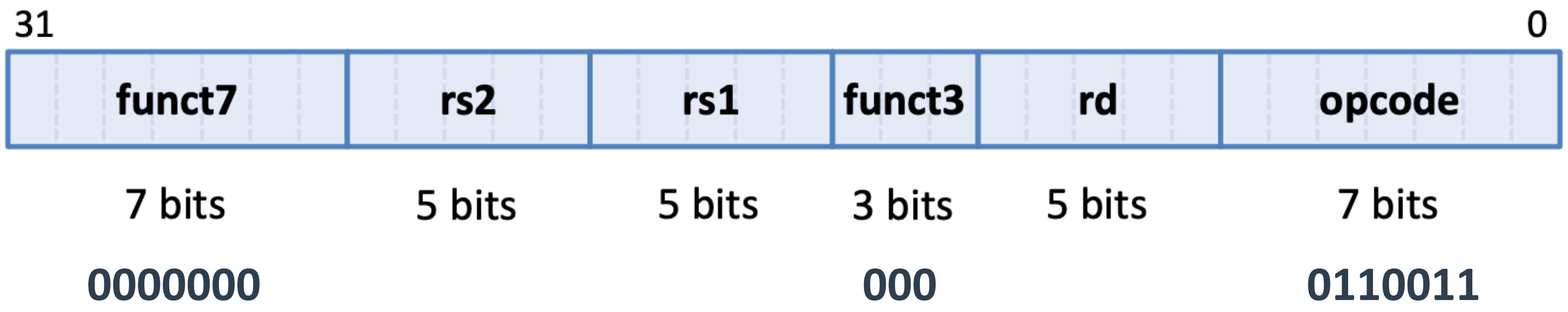
without branch or jump

Datapath: IF Unit



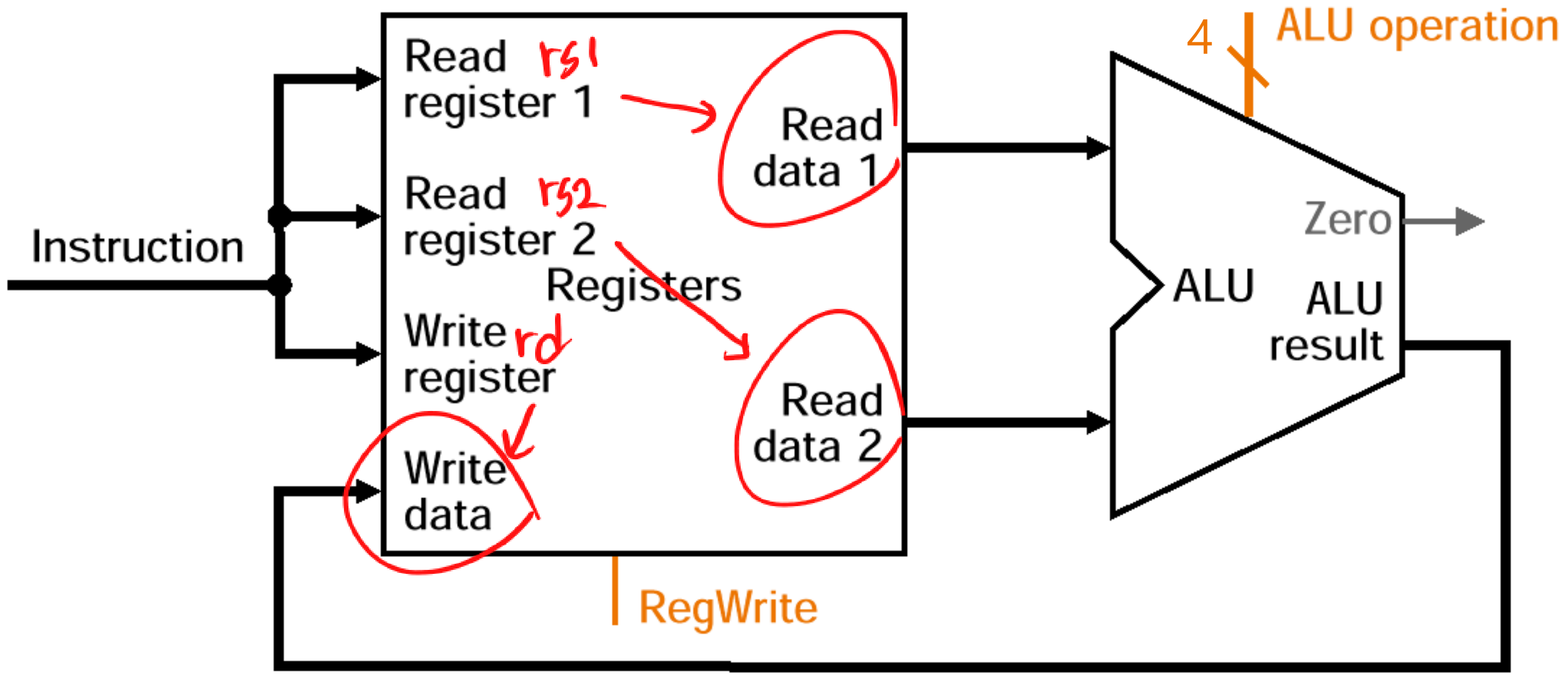
Add RTL

- **Add instruction**
  - add rd, rs1, rs2
    - Mem[PC]; // fetch instruction from instruction memory
    - R[rd] <- R[rs1] + R[rs2] // ADD instruction
    - PC <- PC + 4; // calculate next address



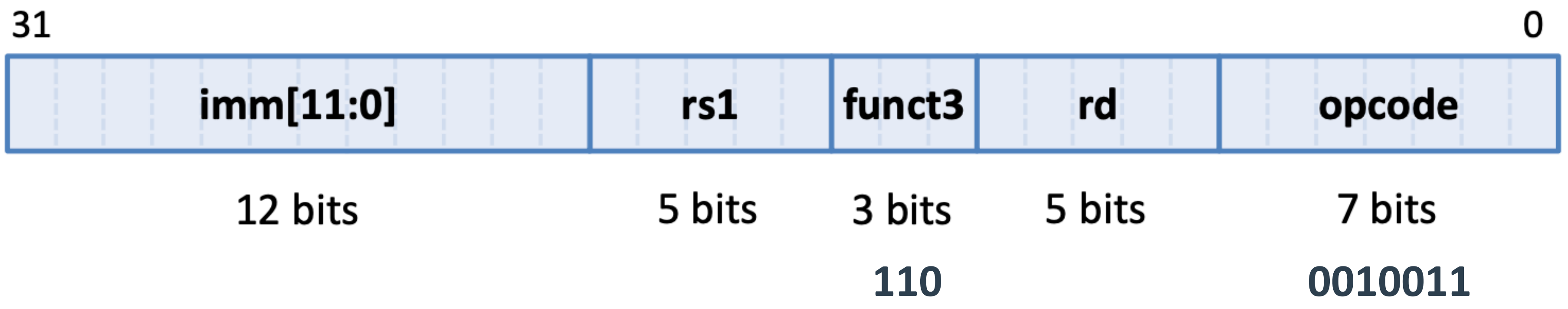
Datapath: Reg/Reg Operations R Type

- **R[rd] <- R[rs1] op R[rs2];**
  - ALU control and RegWrite based on decoded instruction
  - Read register1, read register2, and write register from rs1, rs2, rd fields



OR Immediate RTL

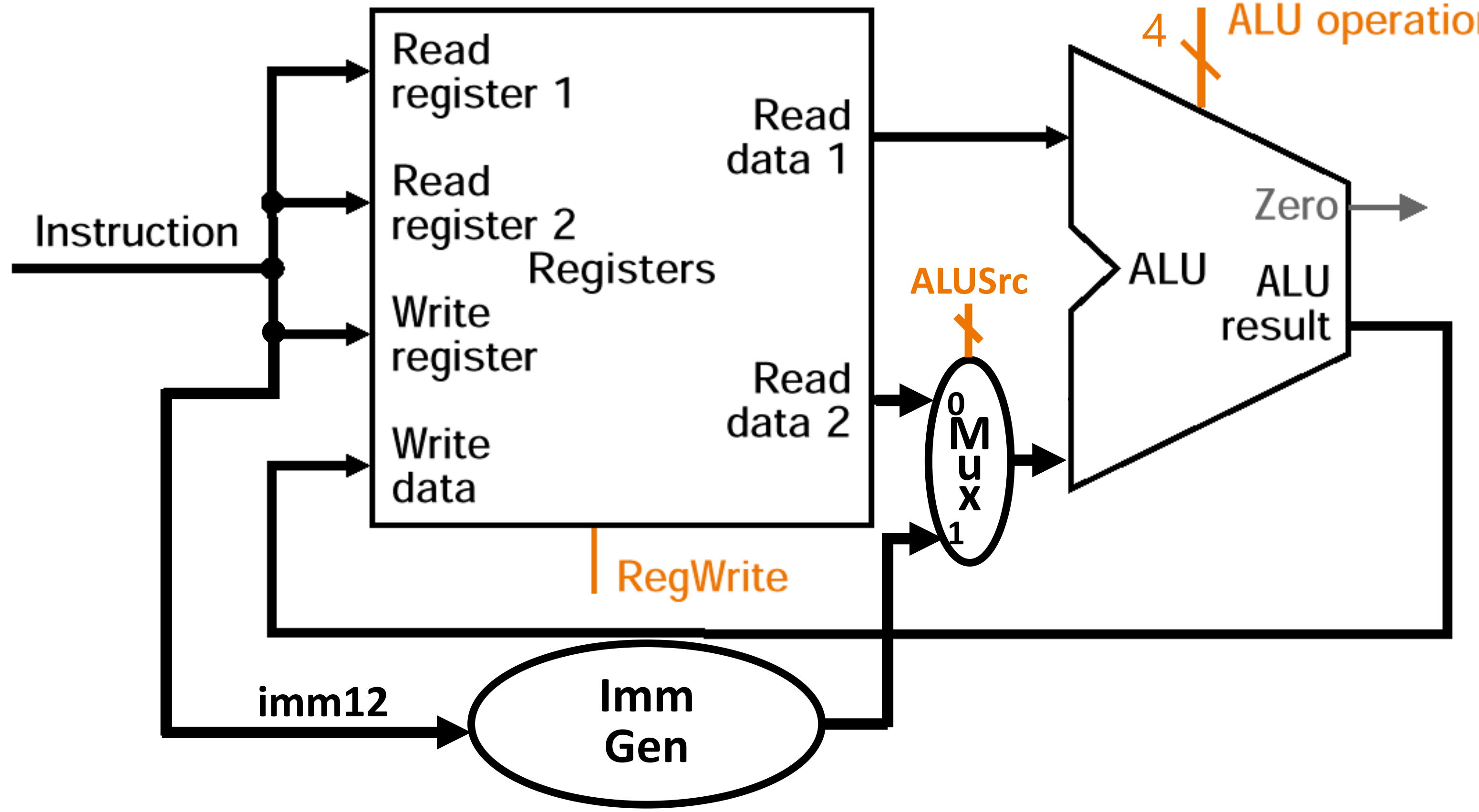
- **OR immediate instruction**
  - ori rd, rs1, imm12
    - Mem[PC]; // fetch instruction from instruction memory
    - R[rd] <- R[rs1] OR SignExt[imm12] // OR operation with Sign-Extension
      Imm Gen
    - PC <- PC + 4; // calculate next address



Datapath: Immediate Operations I Type

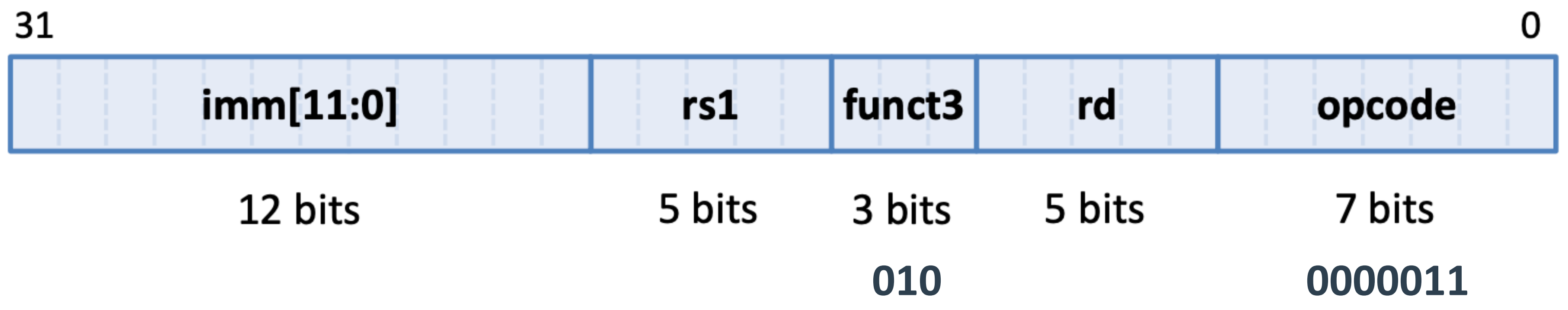
- 1 Mux and Immediate Generation Unit are added
  - Mux: selects data from register if R-format by ALUSrc

I type instruction 인 경우 ALUSrc =1 이어서 Read data 2 대신 imm을 Input으로 받음. ⇒ R type과 I type에 모두 사용하기 위한 장치



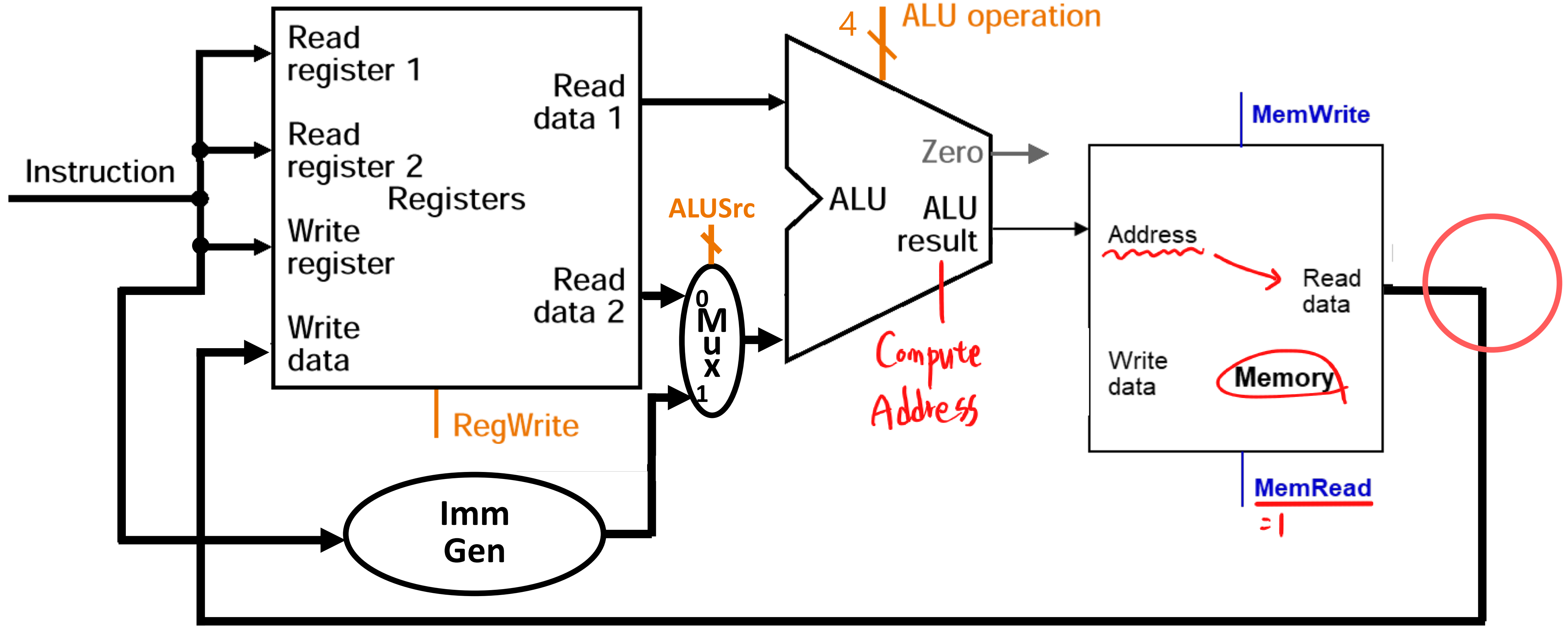
Load RTL

- **Load instruction**
  - lw rd, imm12(rs1)
    - Mem[PC]; // fetch instruction from instruction memory
    - Addr <- R[rs1] + SignExt(imm12); // Compute memory address ALU
    - R[rd] <- Mem[Addr]; // Load data into register
    - PC <- PC + 4; // calculate next address



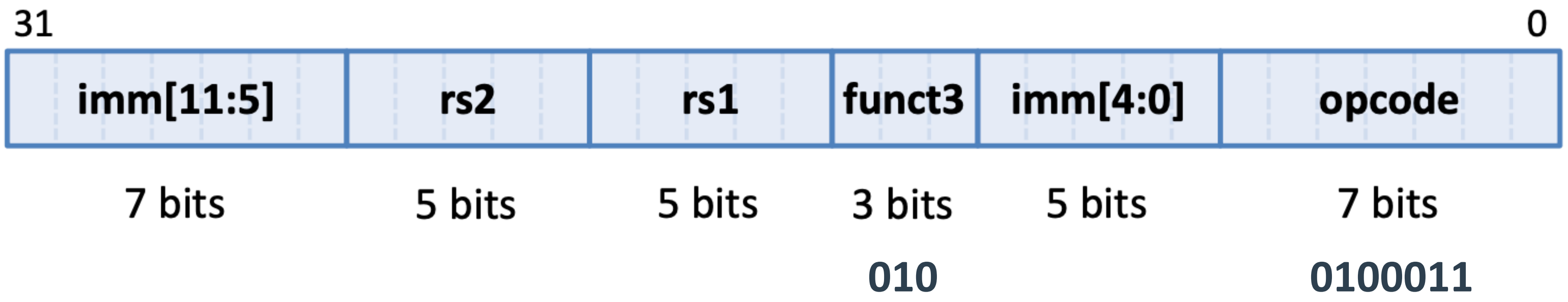
Datapath: Load L Type

- **Sign extension logic is added**
  - Offset can be either positive or negative



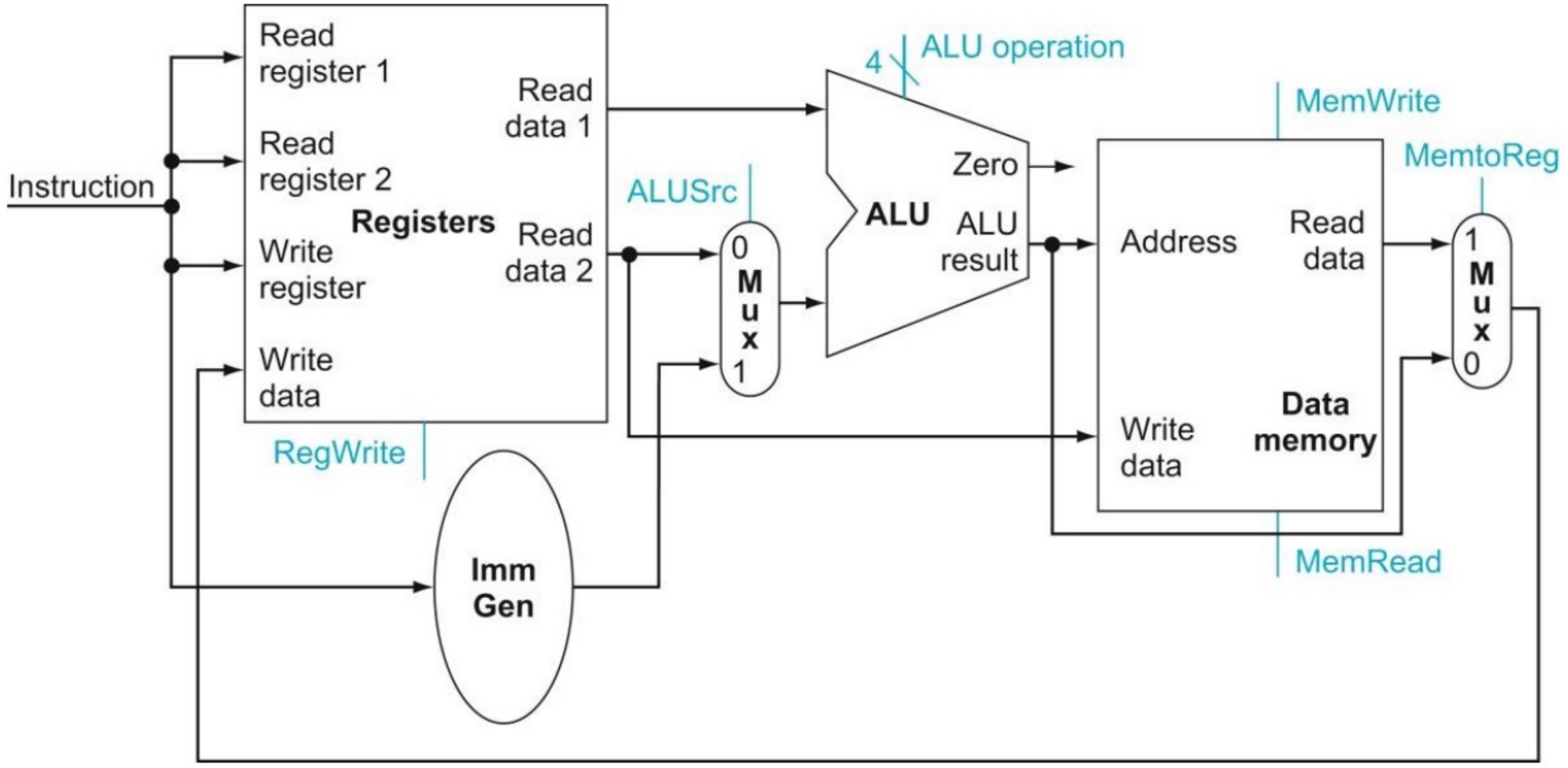
Store RTL

- **Store instruction**
  - sw rs2, imm12(rs1)
    - Mem[PC]; // fetch instruction from instruction memory
    - Addr <- R[rs1] + SignExt(imm12); // Compute memory address
    - Mem[Addr] <- R[rs2]; // Store data into memory
    - PC <- PC + 4; // calculate next address



Datapath: R-Type/Load/Store

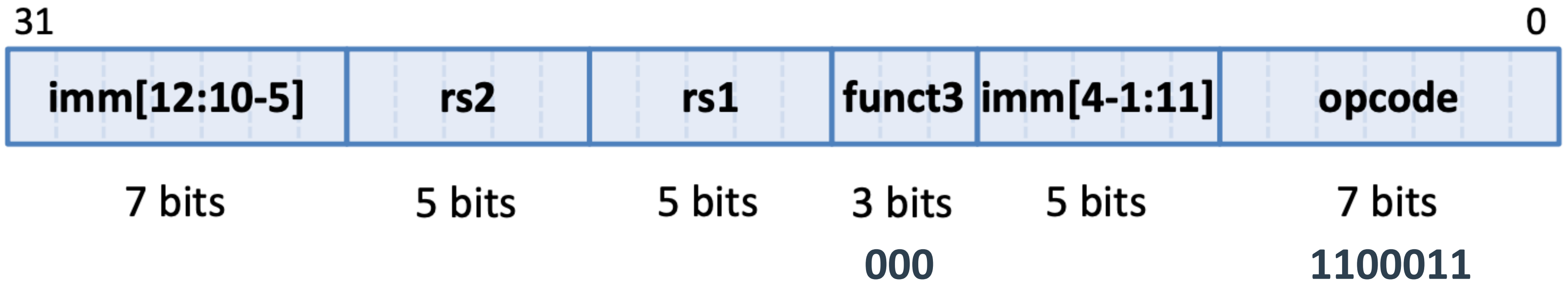
- A path from register to memory has been created
- 1 Mux is added to select data to be stored in registers



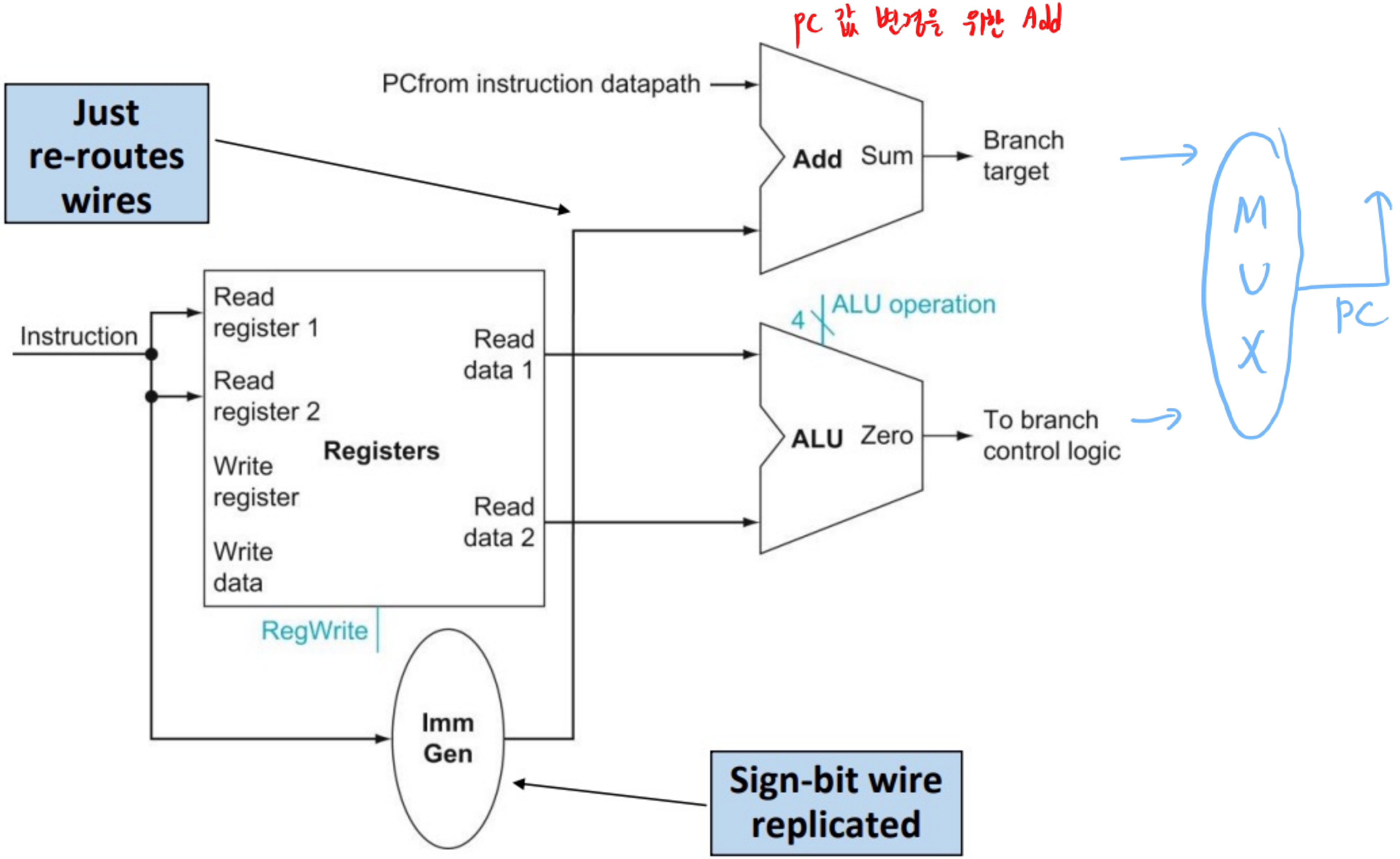
Branch RTL

- **Branch instruction**
  - beq rs1, rs2, imm12

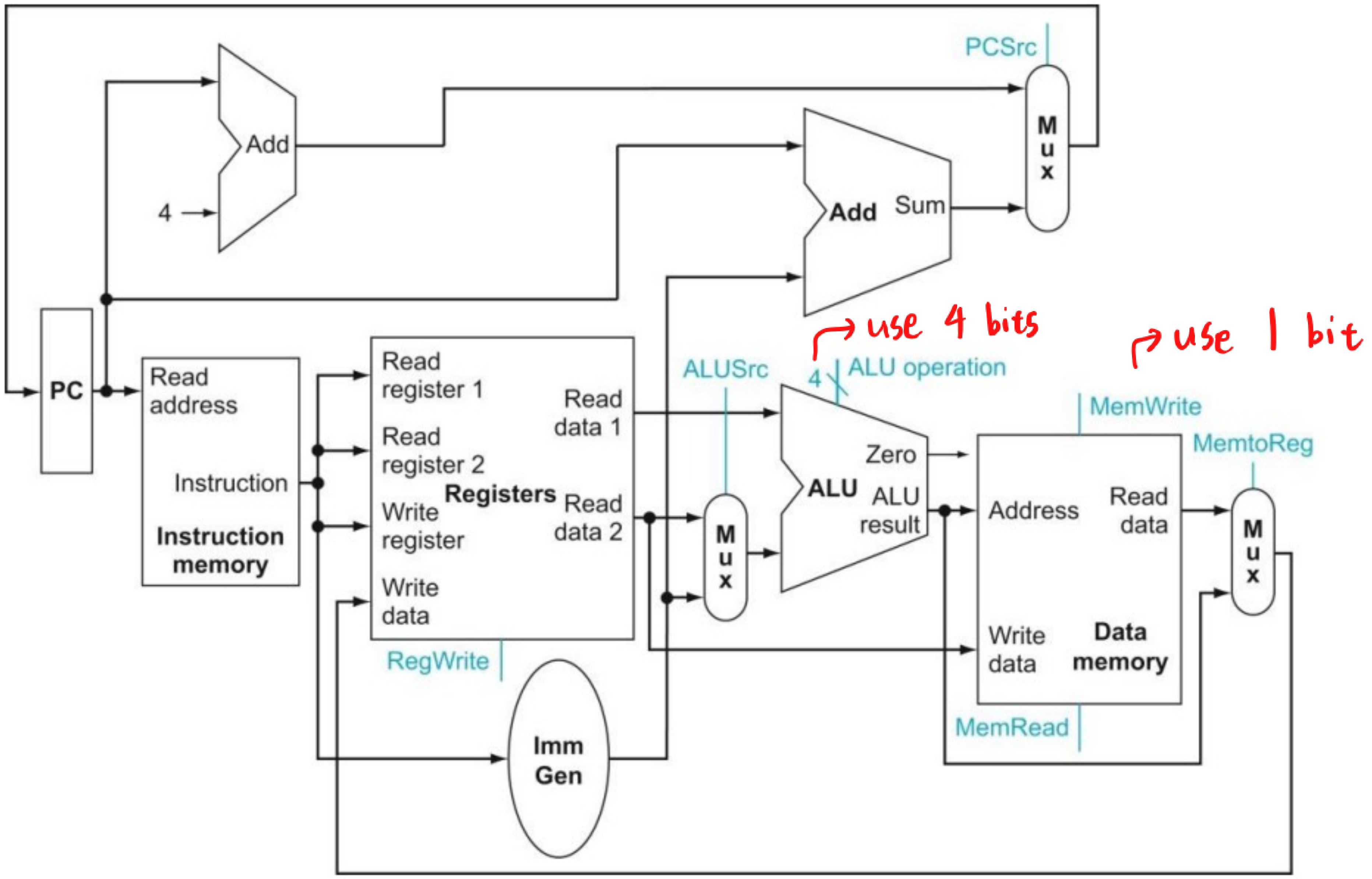
```
– Mem[PC];                        // fetch instruction from instruction memory
– Zero <- (R[rs1] – R[rs2]) + 1;    // Use ALU, subtract and check Zero output
– If (Zero == 1) then
    – PC <- PC + (SignExt(imm12) << 1);       // Branch if equal
– else
    – PC <- PC + 4;                            // Keep going otherwise
```



Datapath: Branch



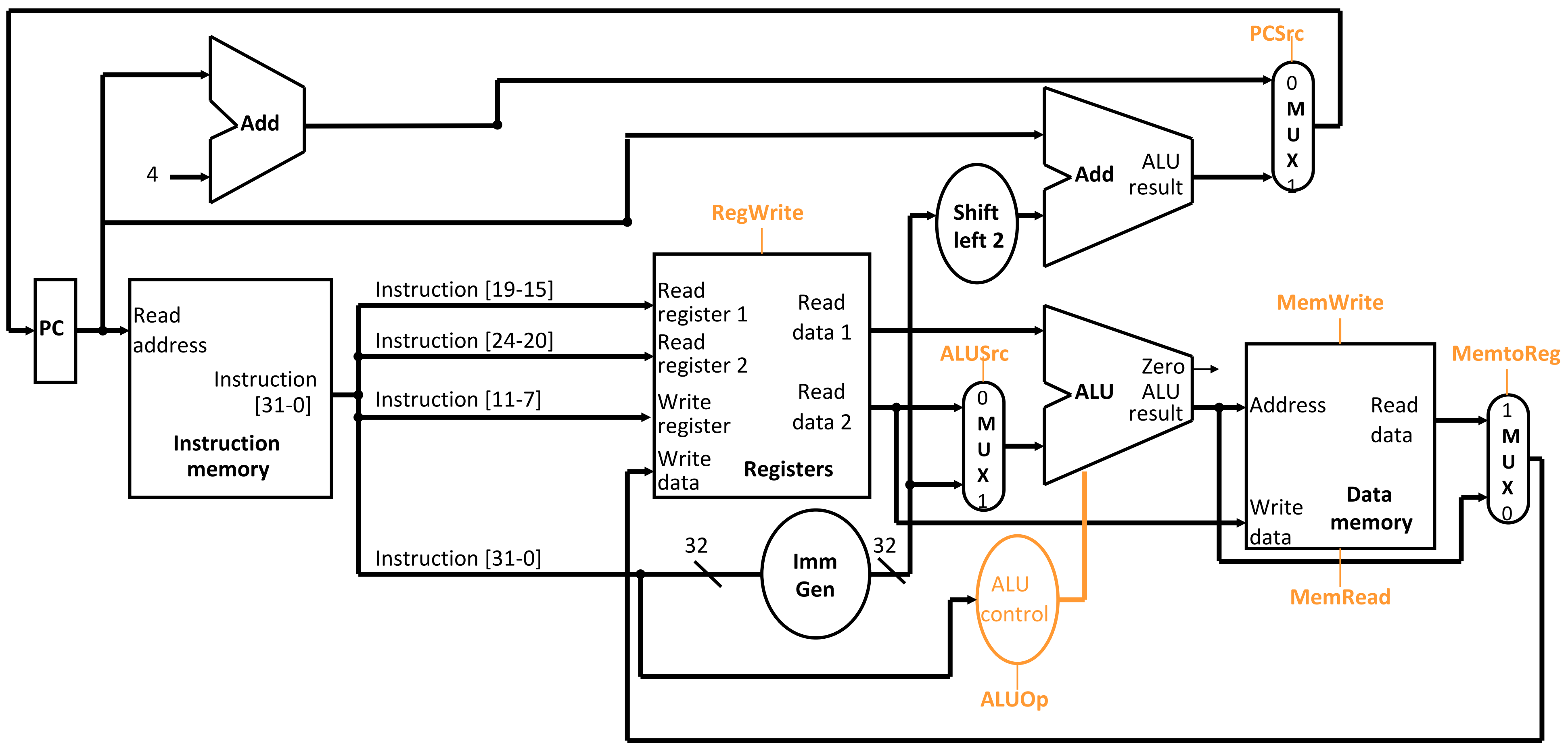
Putting It All Together



Adding Control

- **Design Steps**
  - Identify control points for pieces of the datapath
  - Categorize type of control signals
    - Flow of data through multiplexors
    - Writes of state information
  - Derive control signals for each instruction
  - Put it all together!

Single-Cycle Datapath



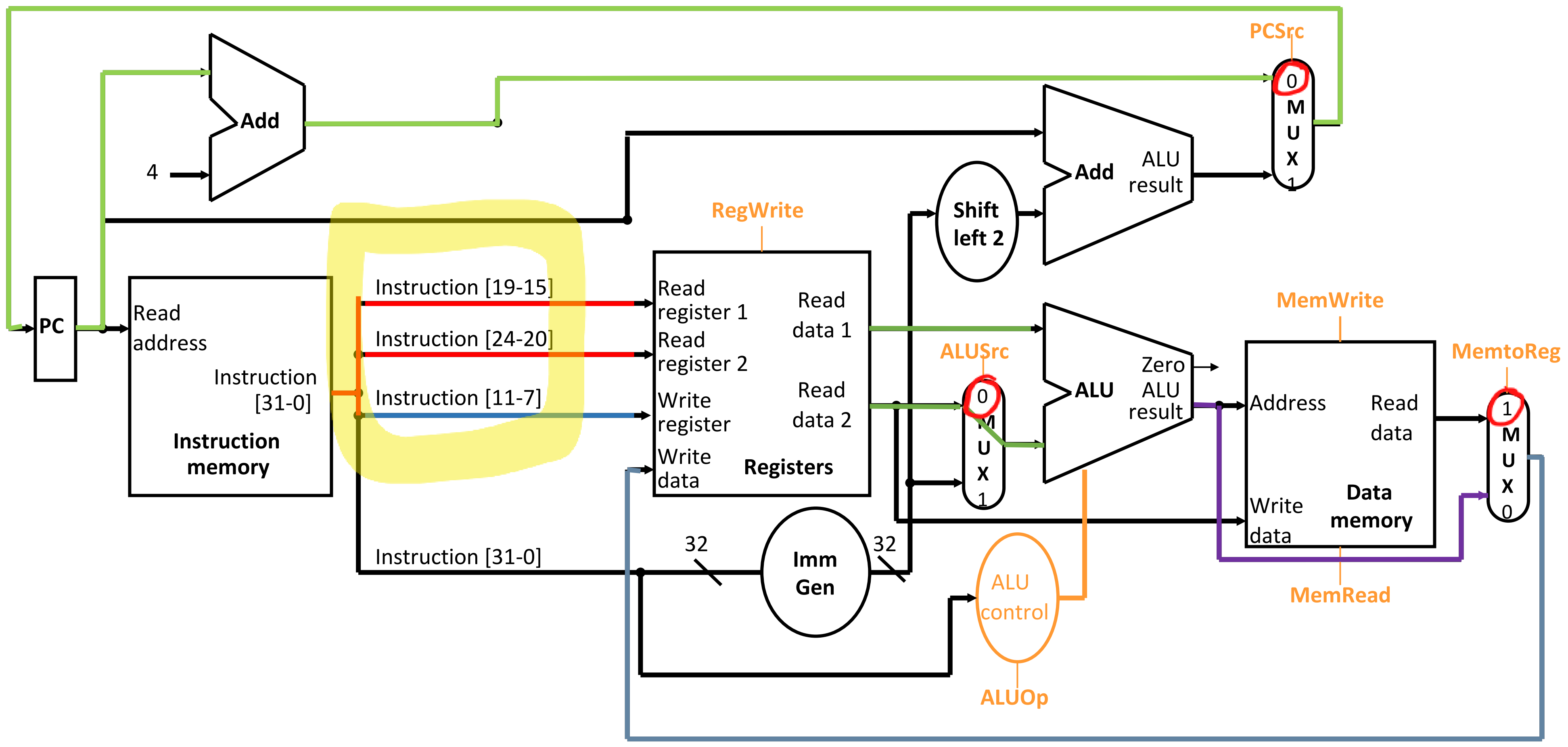
- **This datapath supports the following instructions**
  – add, sub, and, or, lw, sw, beq

Single-Cycle Control

Signal	Description
RegWrite	Specify if the destination register needs to be written
ALUSrc	Select whether source of ALU is register or immediate
ALUOp	Specify operation of ALU
MemWrite	Specify whether memory needs to be written
MemRead	Specify whether memory needs to read
MemtoReg	Select whether memory or ALU output is used for "Write data" of register
PCSrc	Select whether PC + 4 or branch target address is used for the next PC

R-Type Instruction Dataflow

- For add, sub, and, or instructions



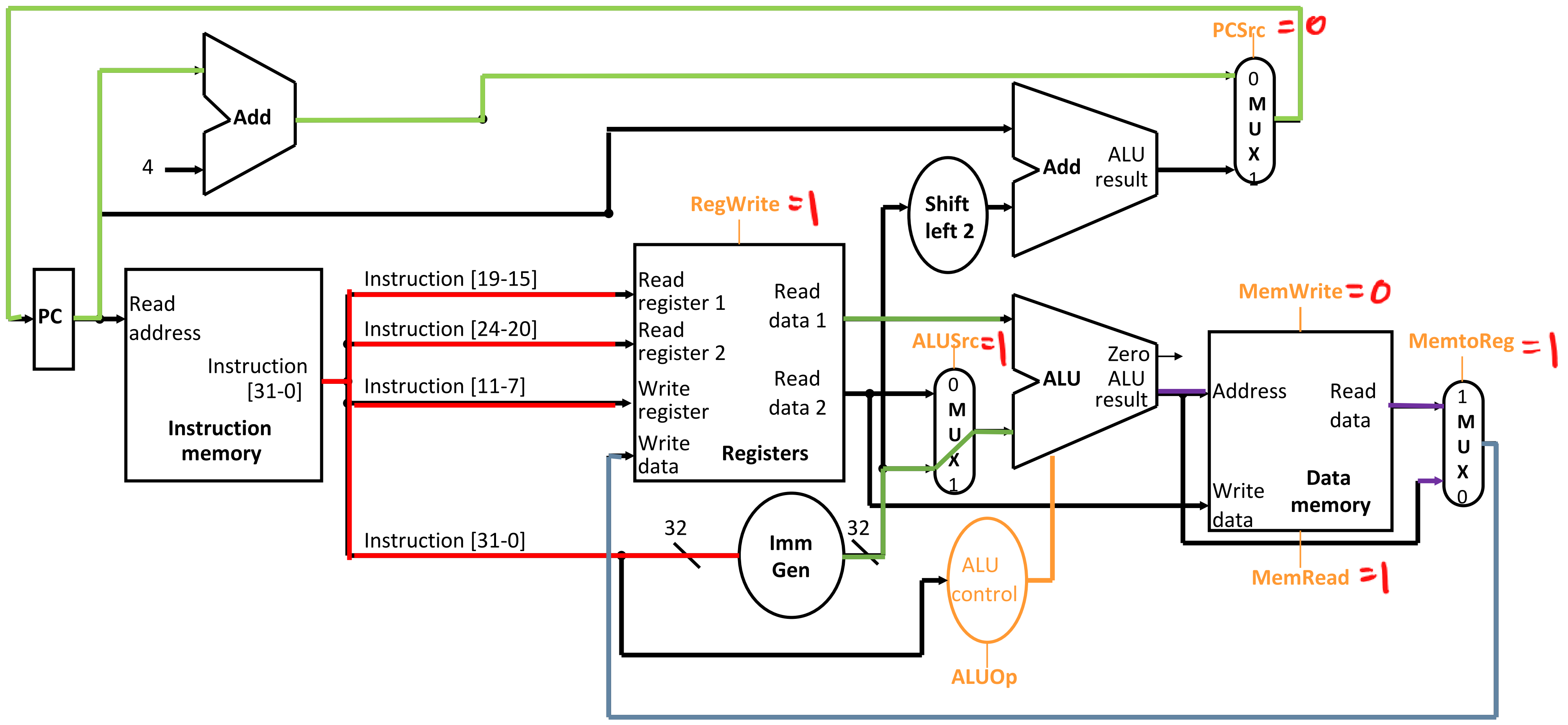
R-Type Instruction Control

Signal	Value	Description
RegWrite	1	To enable writing rd
ALUSrc	0	To select the Read Data 2 from register file
ALUOp	OP	To select an appropriate operation for ALU
MemWrite	0	To disable writing memory
MemRead	0	To disable reading memory
MemtoReg	0	To select ALU output for “Write Data” of register
PCSrc	0	To select PC + 4

외울 필요는 없음.

I-Type Load Instruction Dataflow

- For lw instruction



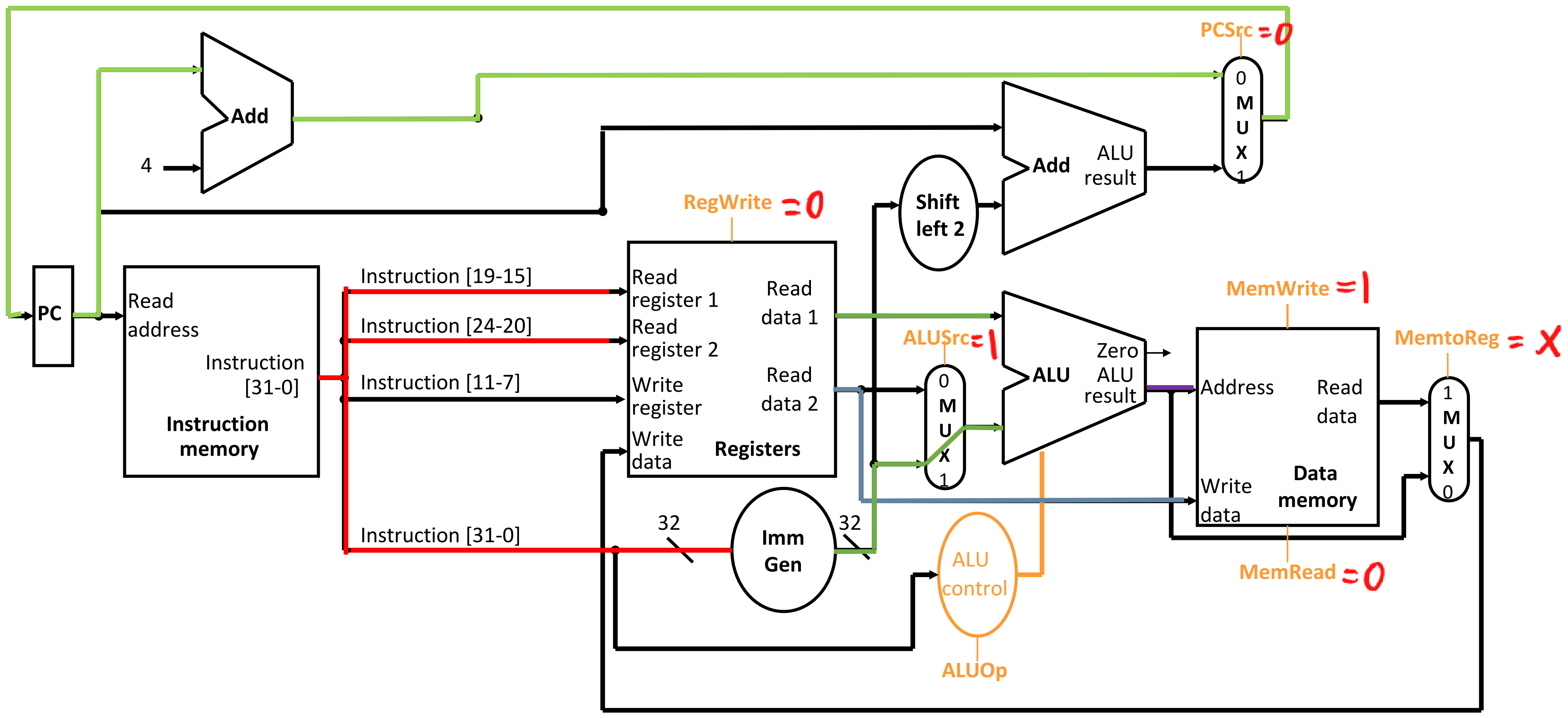
I-Type Load Instruction Control

Signal	Value	Description
RegWrite	1	To enable writing rd
ALUSrc	1	To select the Immediate Offset Value from instruction
ALUOp	OP	To add “Read Data 1” and Immediate Offset
MemWrite	0	To disable writing memory
MemRead	1	To enable reading memory
MemtoReg	1	To select memory output for “Write Data” of register
PCSrc	0	To select PC + 4

} Load

S-Type Store Instruction Dataflow

- For sw instruction

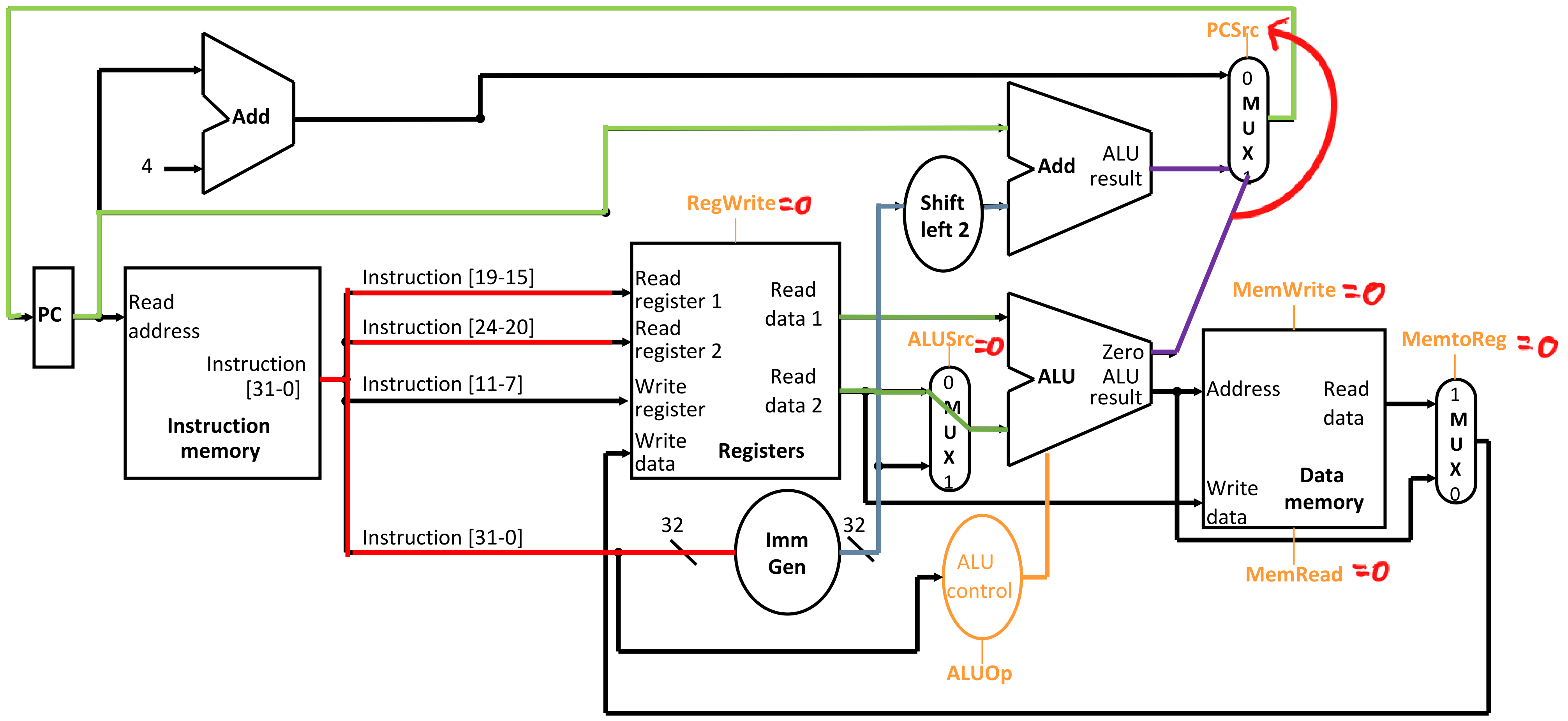


S-Type Store Instruction Control

Signal	Value	Description
RegWrite	0	To disable writing rd
ALUSrc	1	To select the Immediate Offset Value from instruction
ALUOp	OP	To add "Read Data 1" and Immediate Offset
MemWrite	1	To enable writing memory
MemRead	0	To disable reading memory
MemtoReg	X	Not used Don't care (0 or 1)
PCSrc	0	To select PC + 4

SB-Type Branch Instruction Dataflow

- **For beq instruction**



SB-Type Branch Instruction Control

Signal	Value	Description
RegWrite	0	To disable writing rd
ALUSrc	0	To select the Read Data 2 from register file
ALUOp	OP	To sub “Read Data 1” and “Read Data 2”
MemWrite	0	To disable writing memory
MemRead	0	To disable reading memory
MemtoReg	X	Not used
PCSrc	1	To select target address

ALU Control

- **ALU used for**
  - R-type: F depends on opcode
  - Load/Store: F = add
  - Branch: F = subtract

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract

ALU Control (Cont'd)

- **Assume 2-bit ALUOp derived from opcode**
  - Combinational logic derives ALU control

opcode	ALUOp	Operation	funct7	funct3	ALU function	ALU control
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch on equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
		sub	0100000	000	subtract	0110
		and	0000000	111	AND	0000
		or	0000000	110	OR	0001

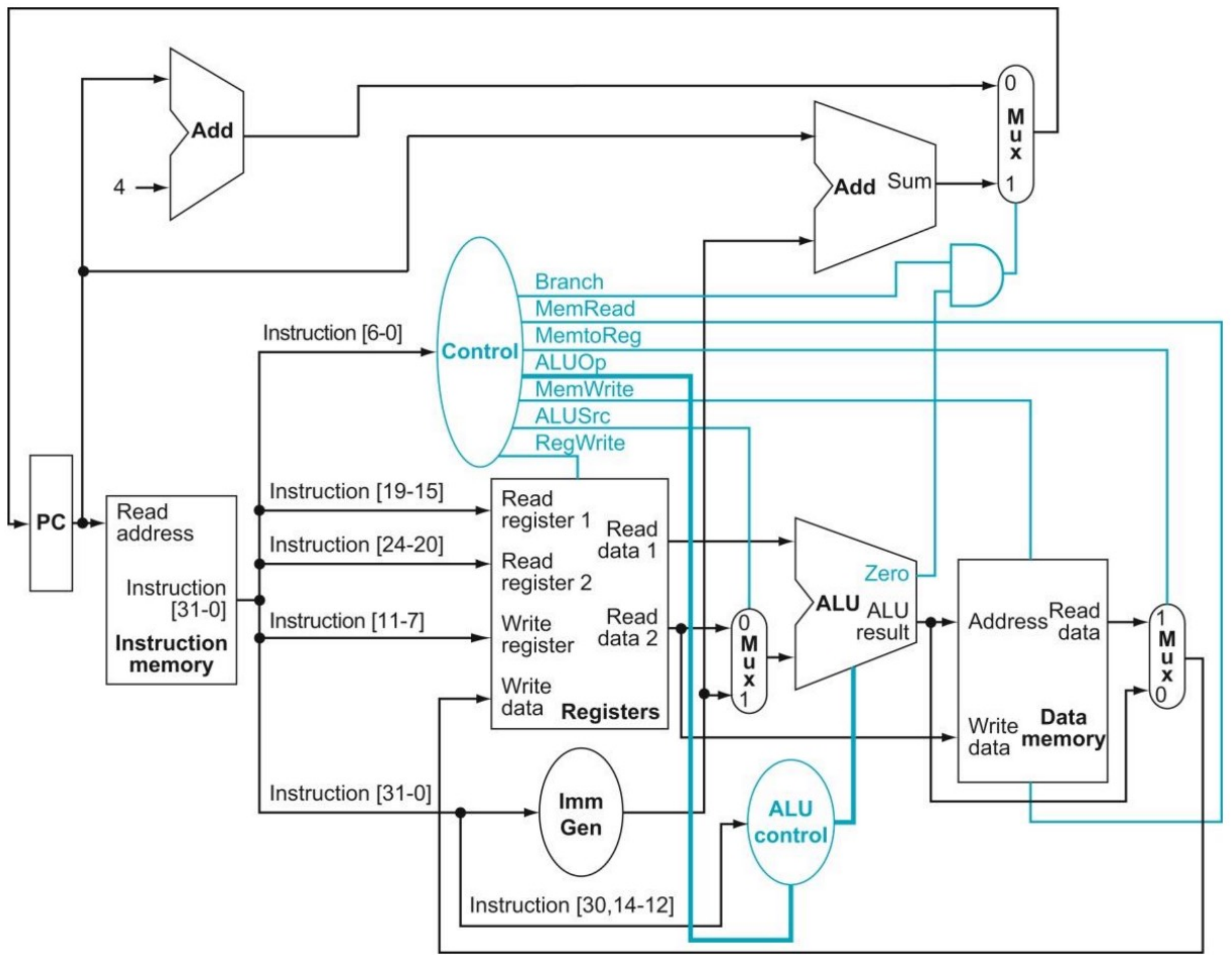
Main Control Unit

- Control signals derived from instruction

Name (Bit position)	Fields 31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

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
X	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

Put It All Together



Single Cycle Processor

- **Advantages**
  - Single cycle per instruction makes logic and clock simple

    모든 종류의 instruction이 같은 clock cycle 사용

- **Disadvantages**
  - Cycle time is determined by the worst-case path
    - Critical path: load instruction
      - Instruction memory -> register file –> ALU -> data memory -> register file
  - Not feasible to adapt to different instructions
    - Different instruction can have different length of time
  - Inefficient utilization of memory and functional units

- **We will improve performance with different approaches**

To Mitigate the Disadvantages

- **Multicycle Implementation**
  - Divide each instruction into a series of steps
  - Each step will take one clock cycle
  - Different instructions can have different CPI difference with single cycle.
- **Requires a few significant changes to organization**
  - Use registers to separate each stage
- **Example**
  - R-format instruction (4 cycles)
    - (1) Instruction fetch (2) Instruction decode/register fetch (3) ALU operation (4) Register write
  - Load instruction (5 cycles)
    - (1) Instruction fetch (2) Instruction decode/register fetch (3) Address computation (4) Memory read (5) Register write