Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2021

Sequential Processor

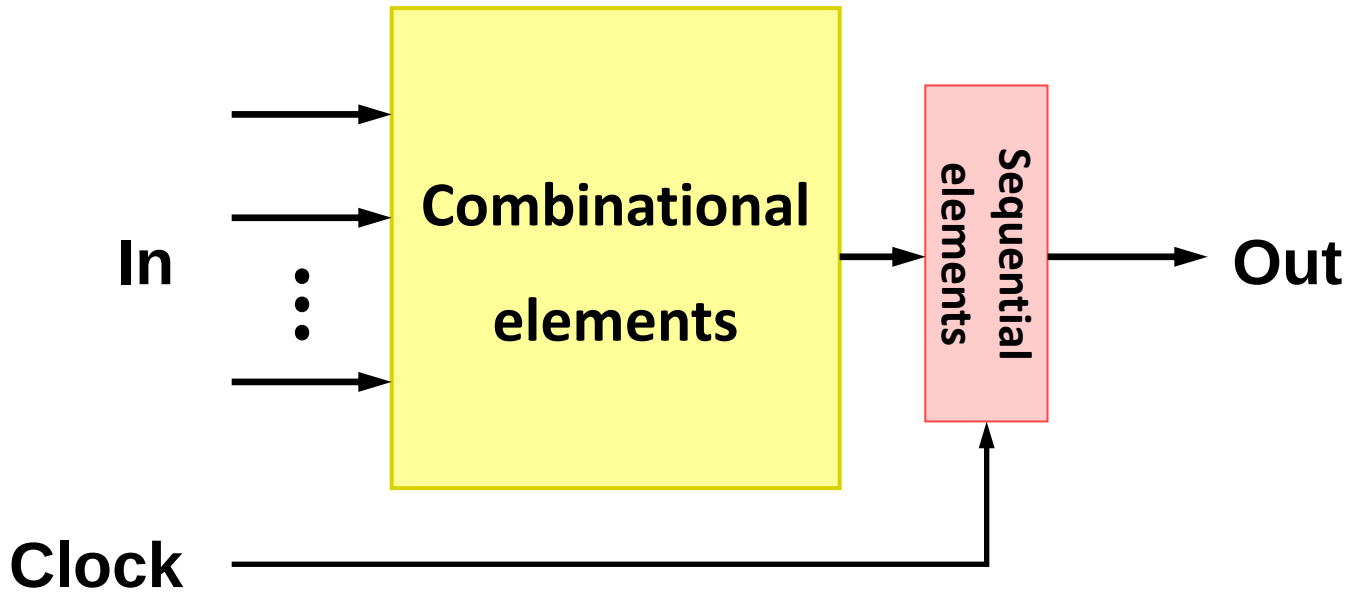
Chap. 4.1 – 4.4

Introduction

- CPU performance factors
  - Instruction count – Determined by ISA and compiler
  - CPI and cycle time – Determined by CPU hardware
- We will examine two RISC-V implementations
  - A simplified (sequential) version
  - A more realistic pipelined version
- Simple subset, shows most aspects (RV32I)
  - Memory reference: `lw`, `sw`
  - Arithmetic/logical: add, sub, and, or
  - Control transfer: `beq`

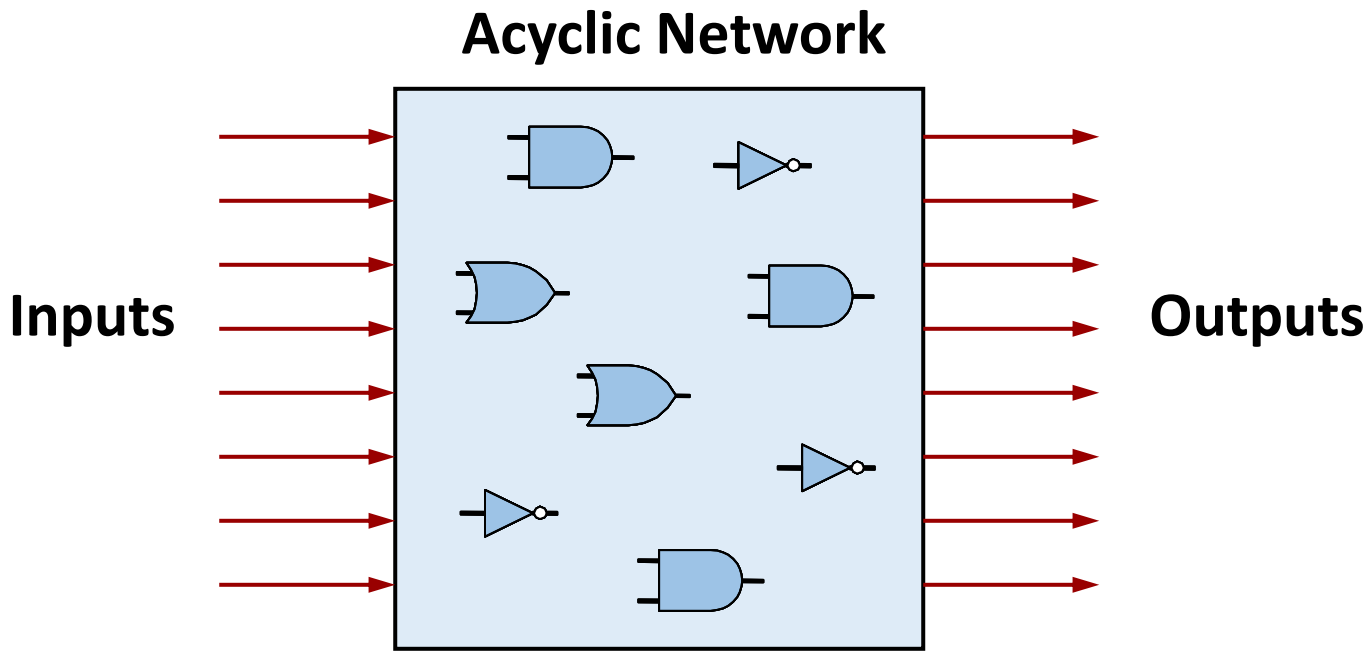
Digital Systems

- Three components required to implement a digital system
  - Combinational elements to compute Boolean functions
  - Sequential elements to store bits
  - Clock signals to regulate the updating of the memory elements



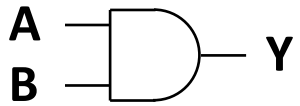
Combinational Circuits

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

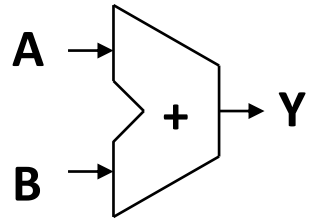


Combinational Elements: Examples

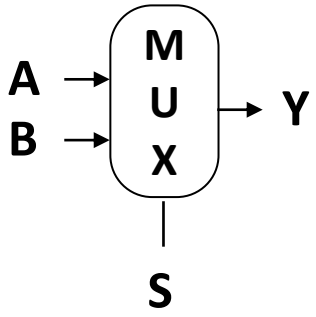
- AND-gate
  - `Y = A & B`



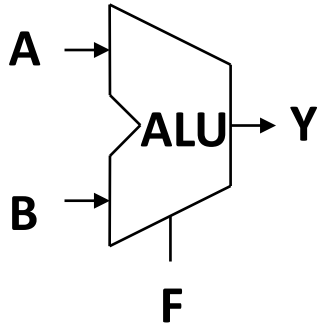
- Adder
  - `Y = A + B`



- Multiplexer
  - `Y = S? A : B`

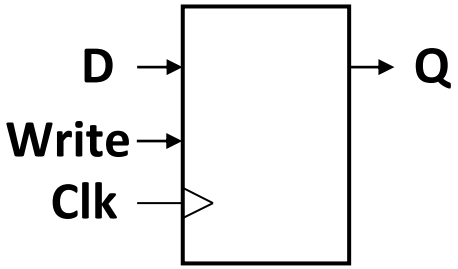


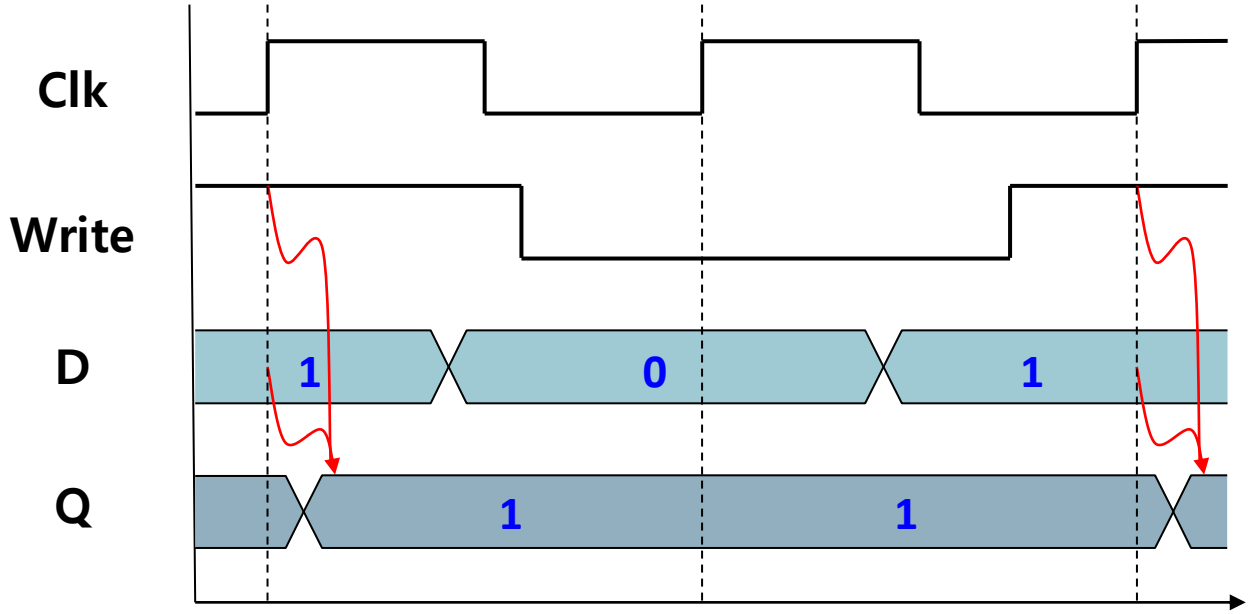
- Arithmetic/Logic Unit
  - `Y = F(A, B)`



Sequential Elements: Storing 1 bit

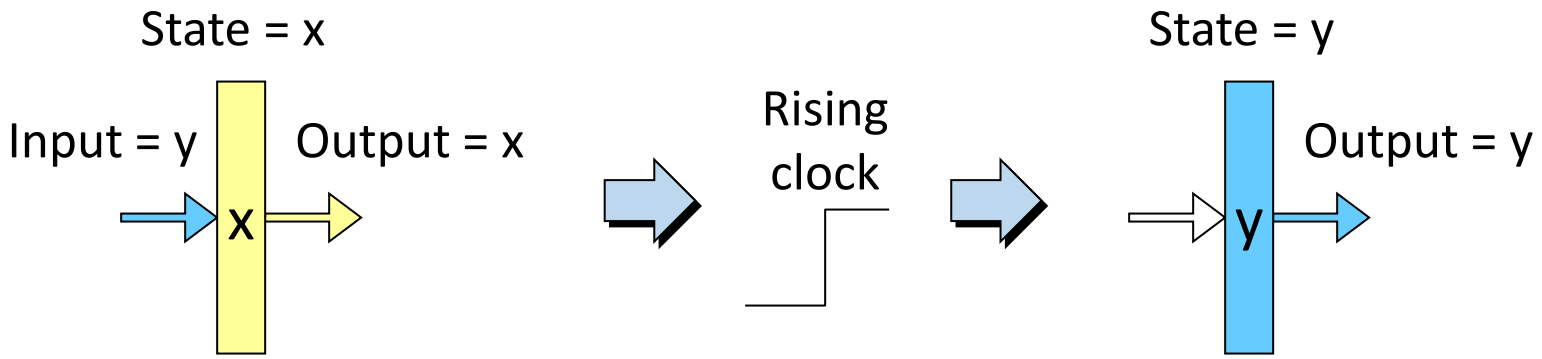
- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later





Register Operation

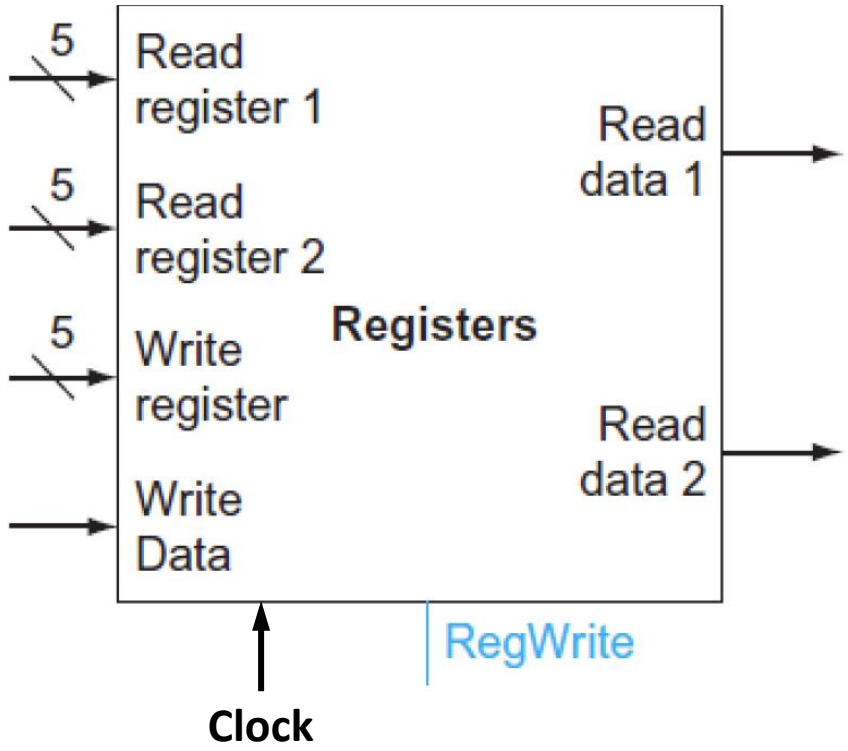
- Stores data bits
- For most of time acts as barrier between input and output
- As clock rises, loads input



Register File

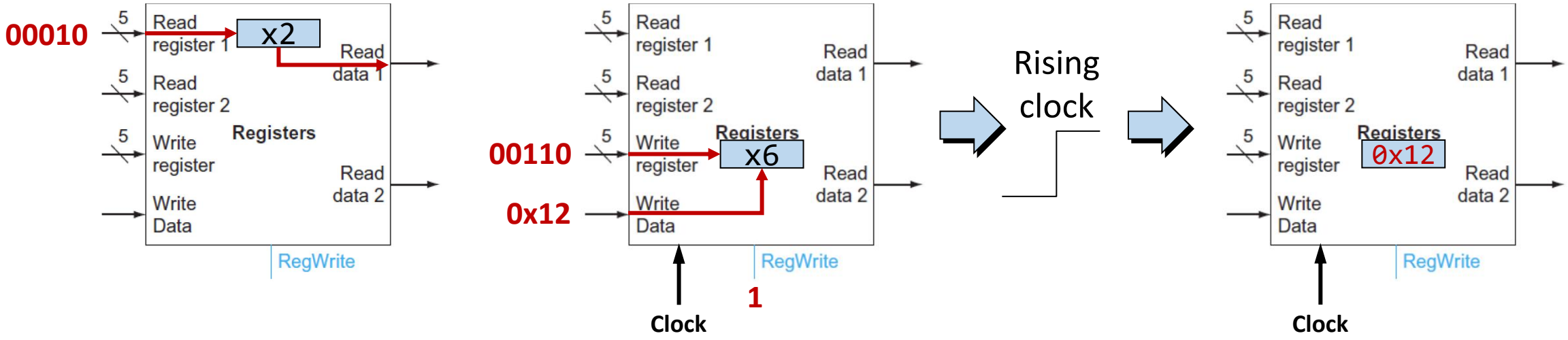
- A collection of registers
  - Holds values of "program registers"
  - x0 ~ x31 in RISC-V
  - Register identifier (5 bits) serves as address

- Multiple ports
  - Can read and/or write multiple words in one cycle
  - Each has separate address and data input/output
  - Data is written to the register only when RegWrite signal is enabled



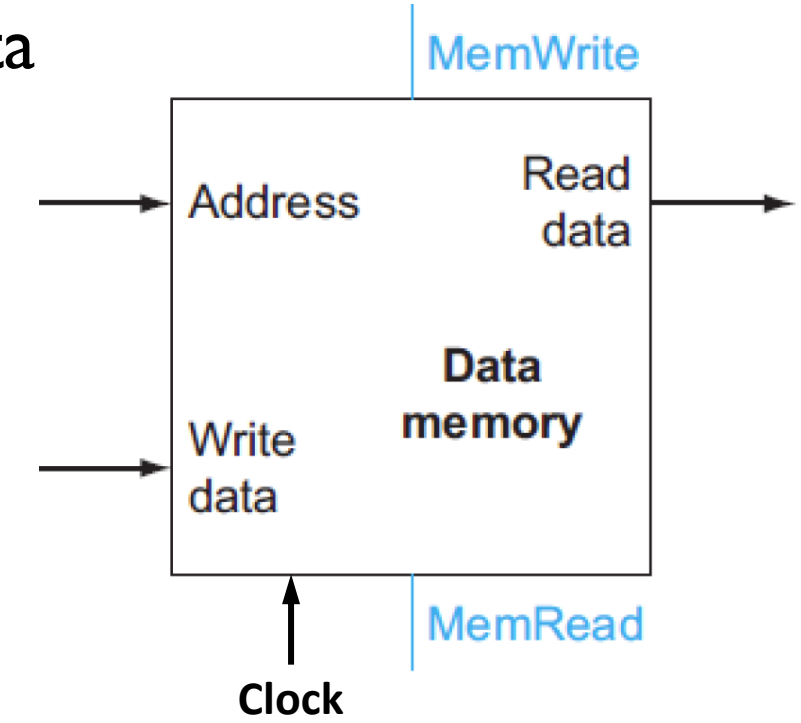
Register File Timing

- Reading
  - Like combinational logic
  - Output data generated based on input address (after some delay)
- Writing
  - Like register
  - Update only as clock rises



Data Memory

- Random access memory for storing program data
- Operations similar to registers
- Reading: like combinational logic
- Writing: update only as clock rises
- Read/Write controlled using MemWrite and MemRead signals



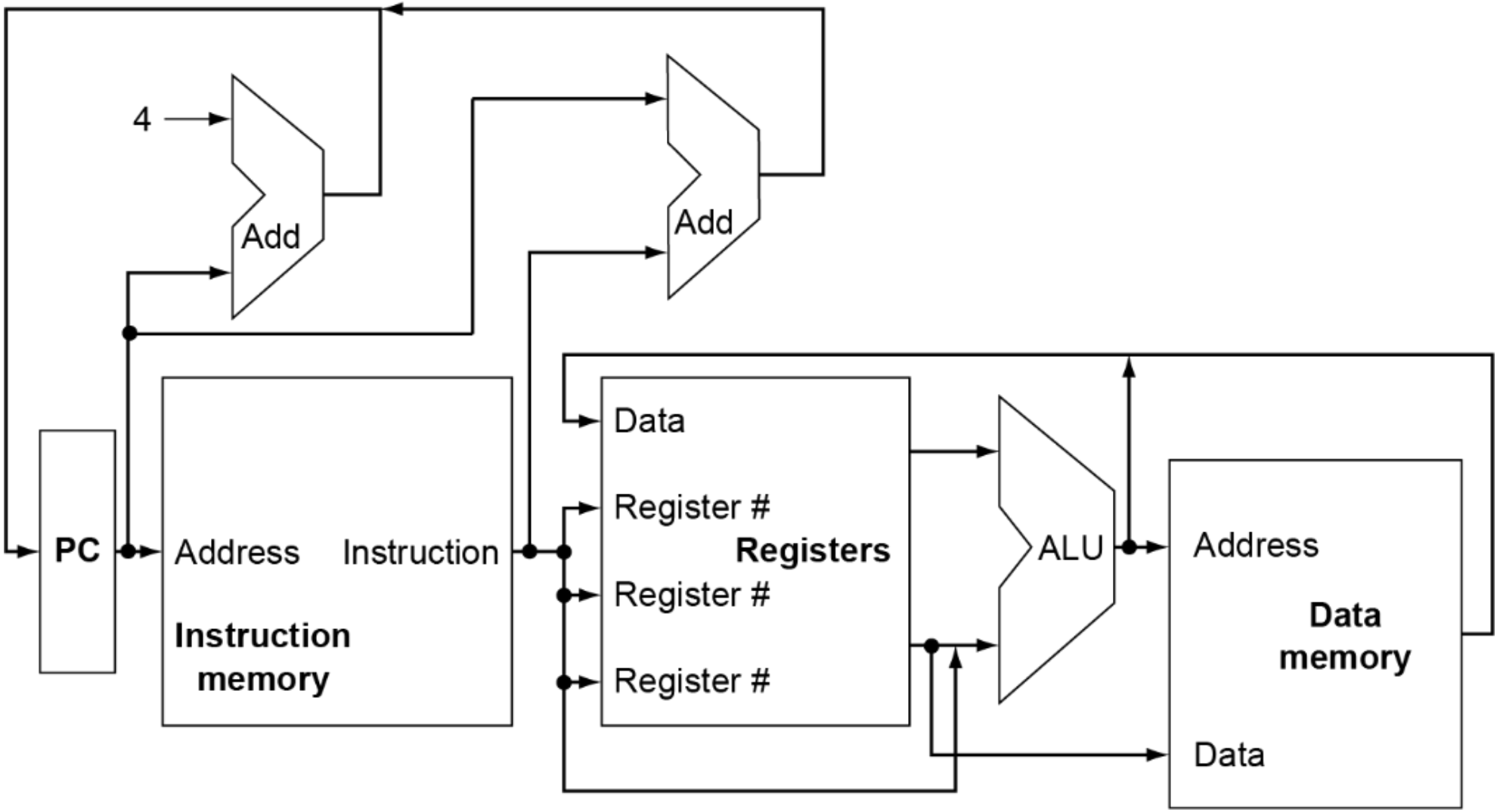
- Another read-only memory needed for instructions
- Dual-port memory: single memory for both instructions and data
  - One read port for instructions, another read or write port for data

Instruction Execution

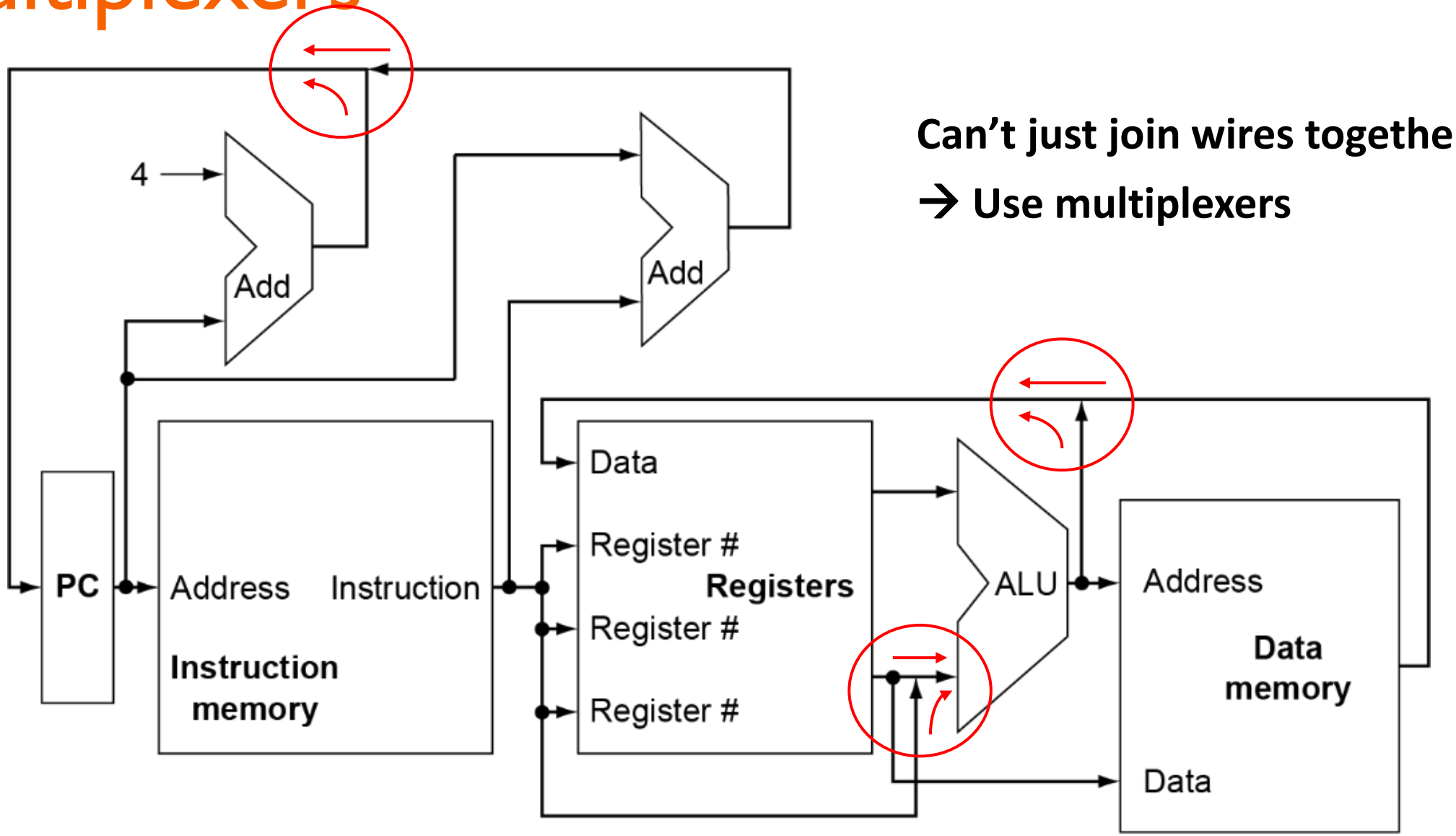
- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate: Arithmetic result
Memory address for load/store
Branch comparison
  - Access data memory for load/store
  - PC ← target address or PC + 4

CPU: Datapath + Control

Datapath


4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data

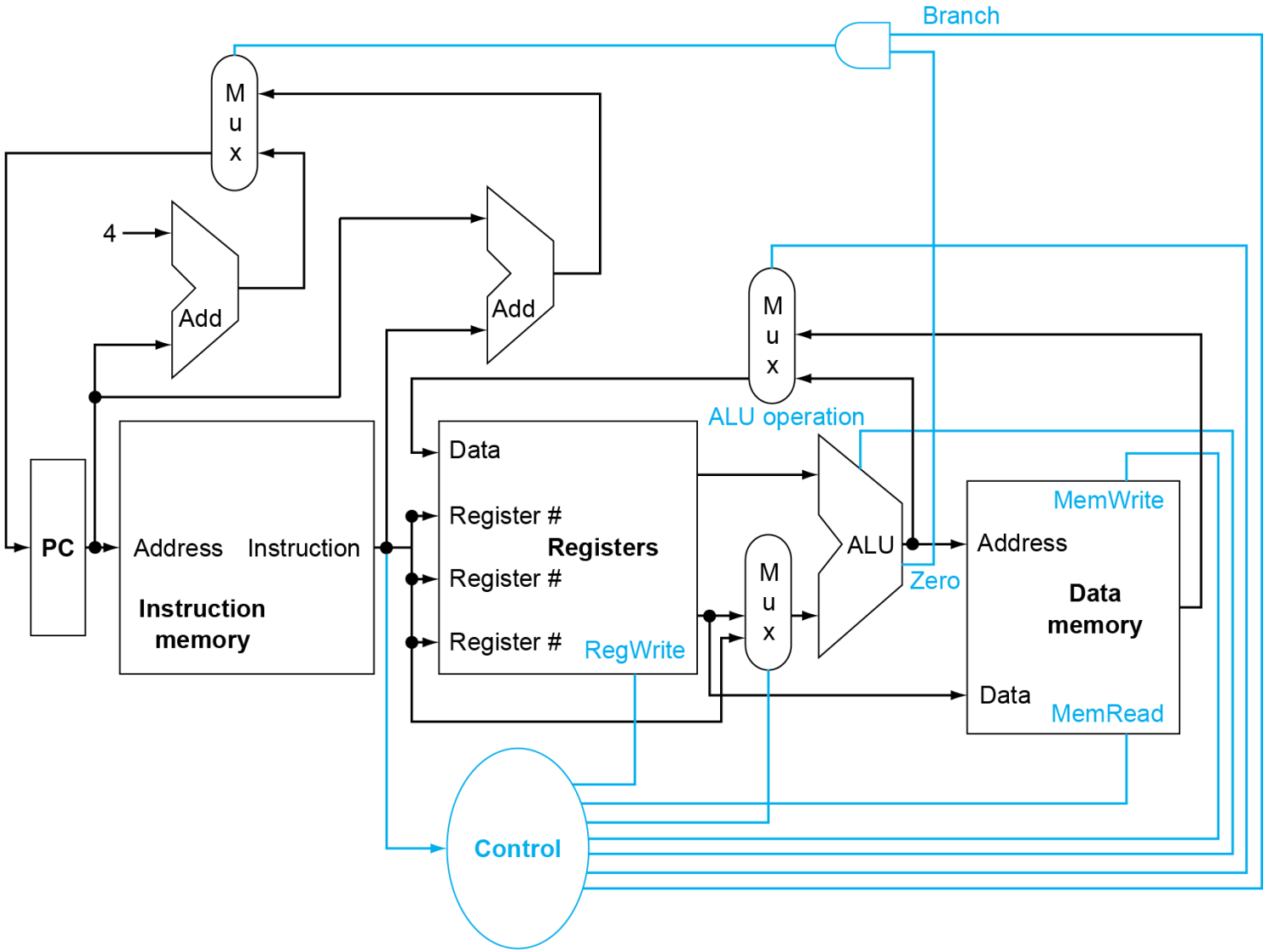
Multiplexers



Can't just join wires together

→ Use multiplexers

Control

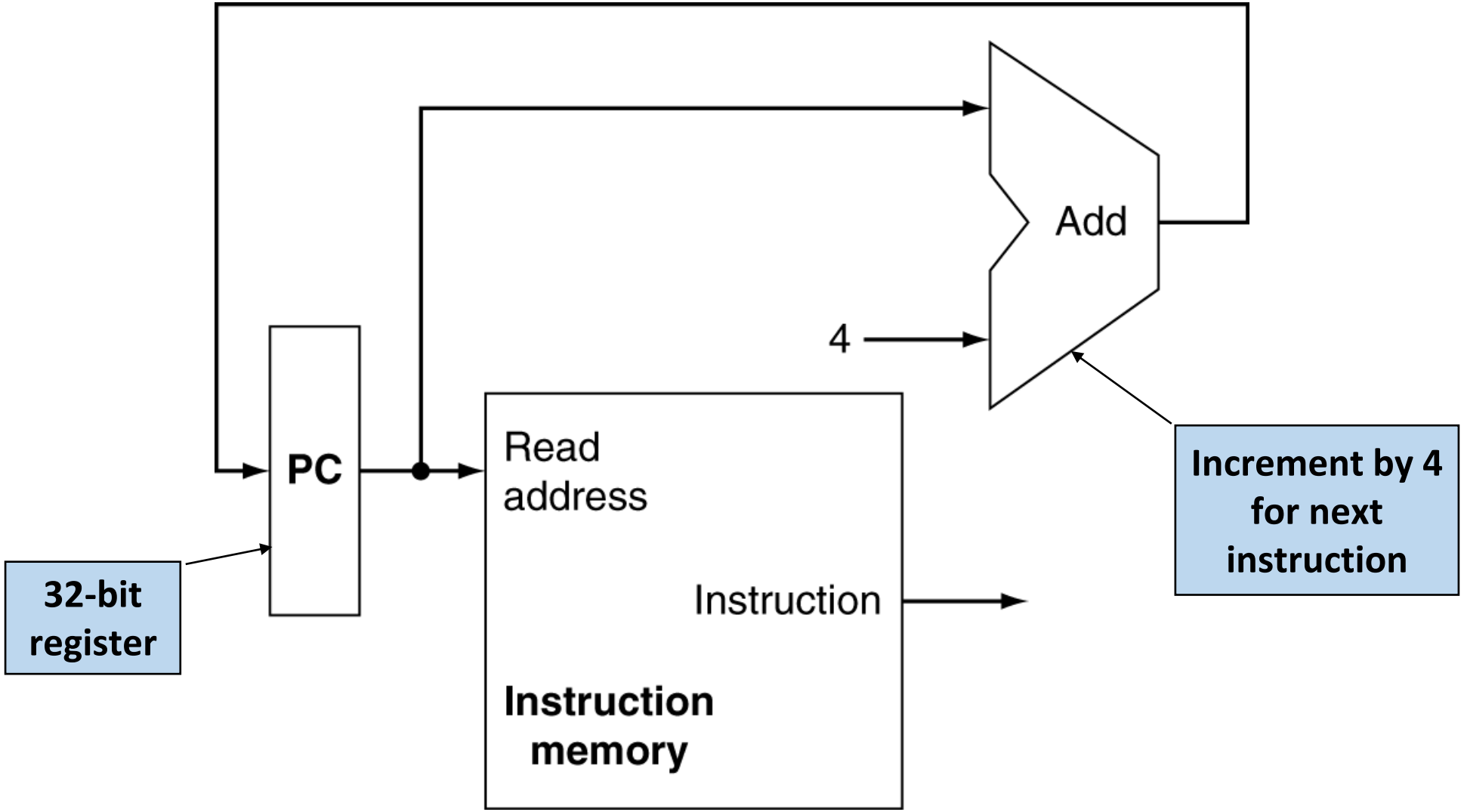

Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
  - Registers, ALUs, MUX's, Memories, …

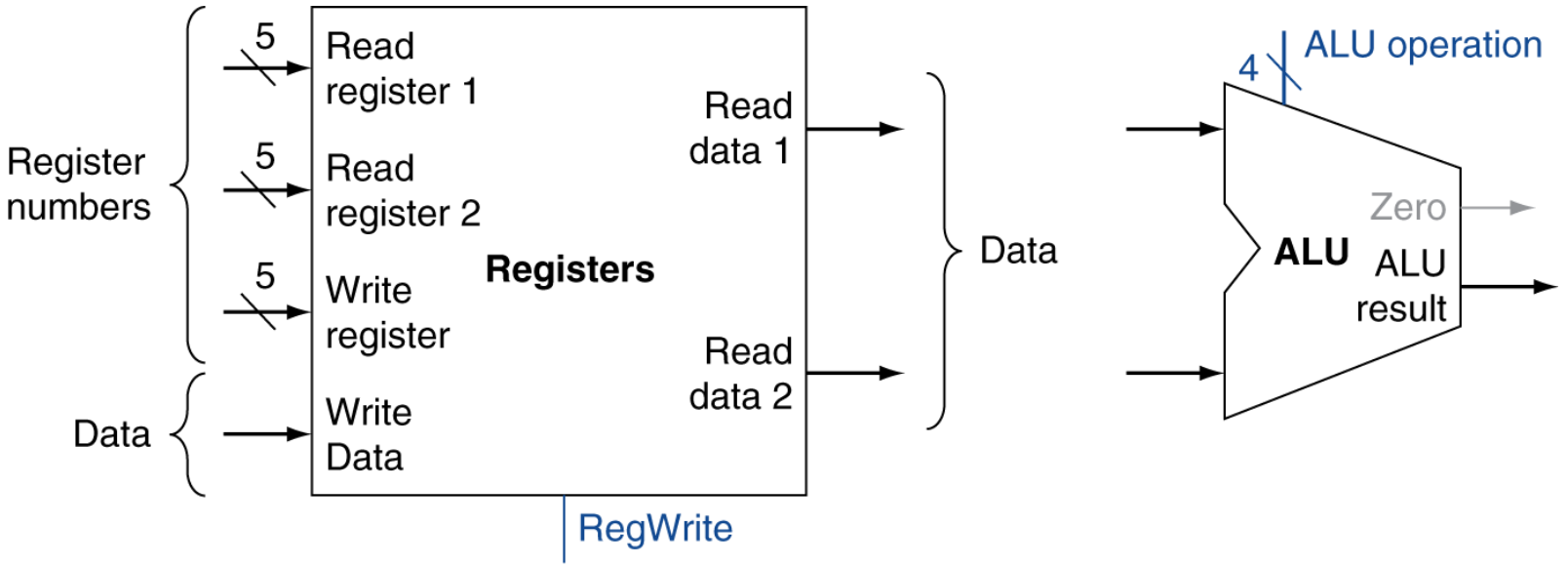
- We will build a RISC-V datapath incrementally
  - Refining the overview design

Instruction Fetch



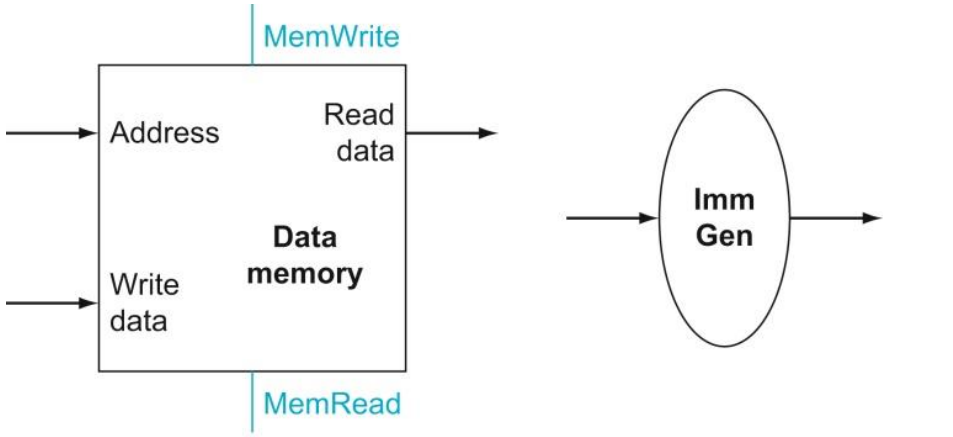
R-Format Instructions

- Read two register operands
- Perform arithmetic/logical operation
- Write register result



Load/Store Instructions

- Read register operand
- Calculate address using 12-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory



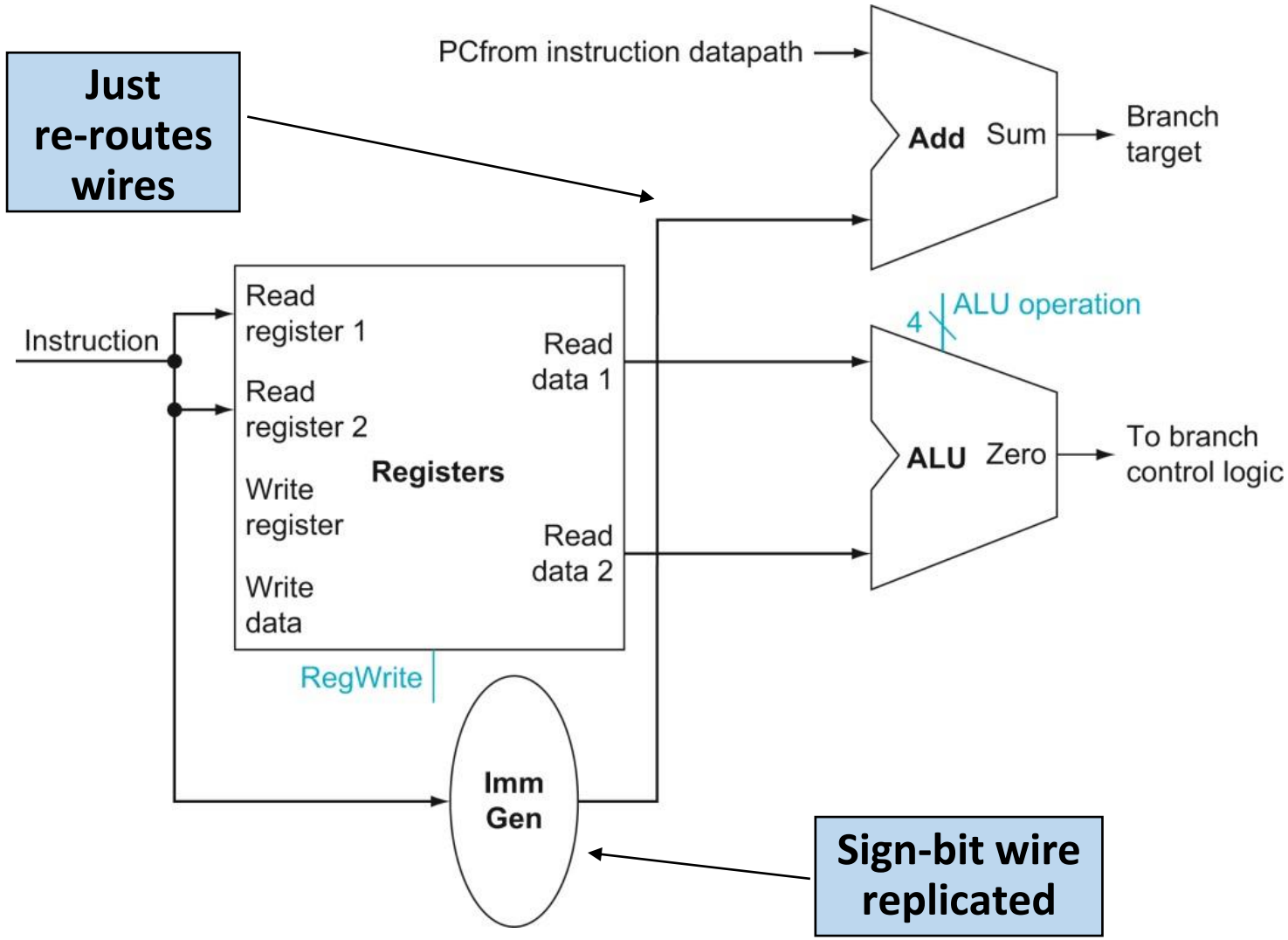
a. Data memory unit

b. Immediate generation unit

Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 1 place (halfword displacement)
  - Add to PC value

Branch Instructions (cont'd)

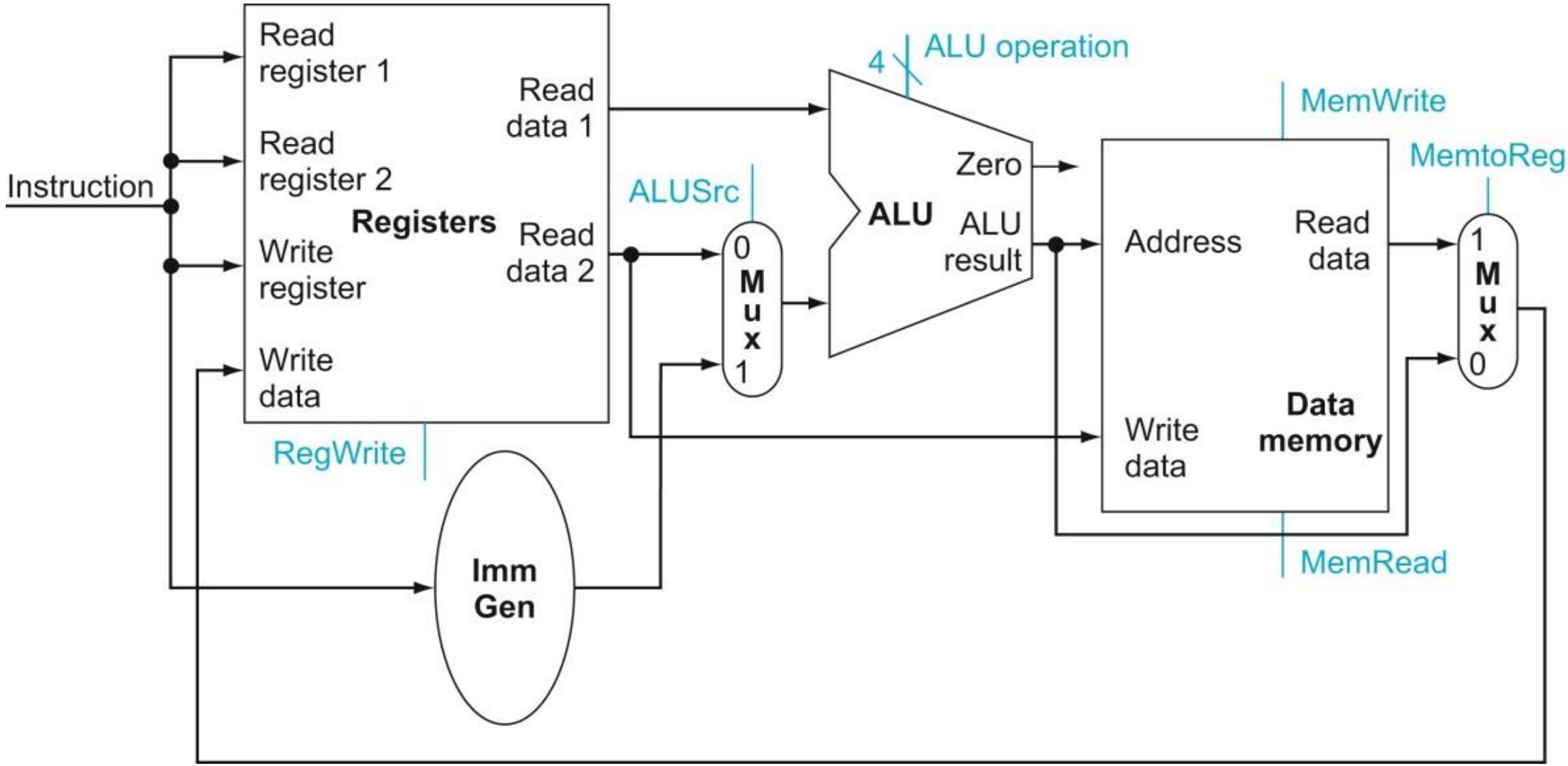


Composing the Elements

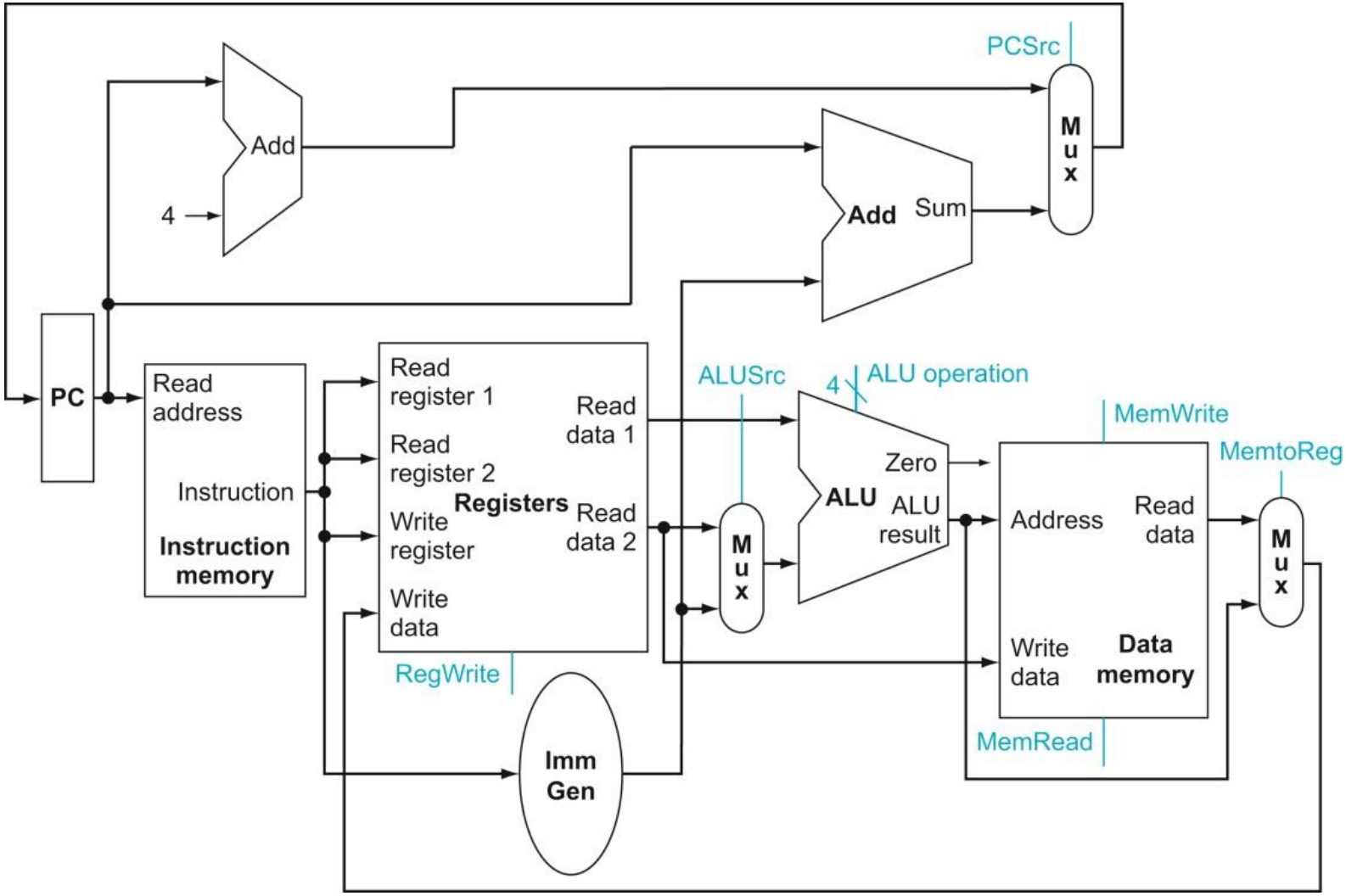
- First-cut datapath does an instruction in one clock cycle
  - Each datapath element can only do one function at a time
  - Hence, we need separate instruction and data memories

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

R-Type/Load/Store Datapath



Full Datapath



ALU Control

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

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

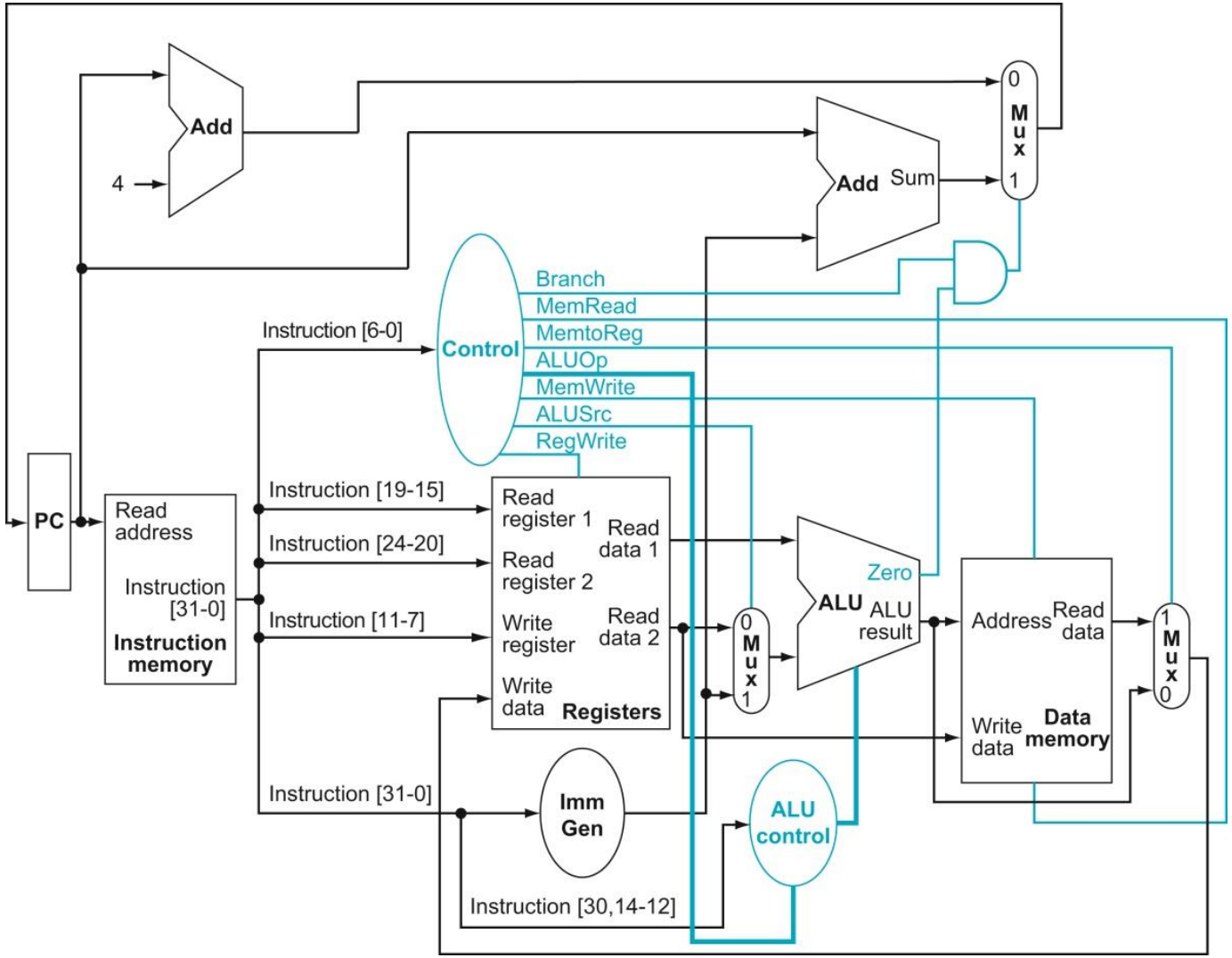
The Main Control Unit

- Control signals derived from instruction

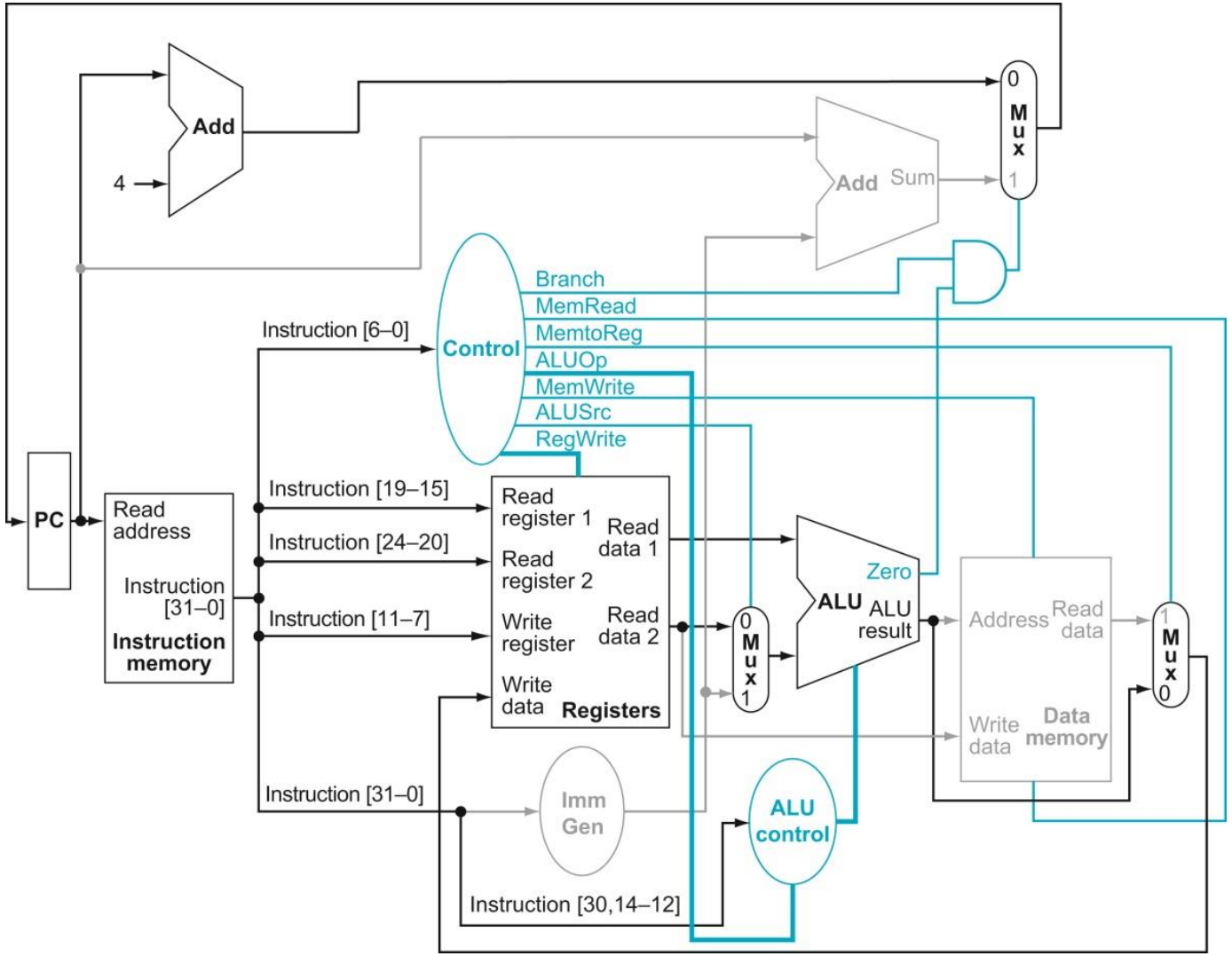
Name (Bit position)	Fields 31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

ALUOp		Funct7 field							Funct3 field			
ALUOp1	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

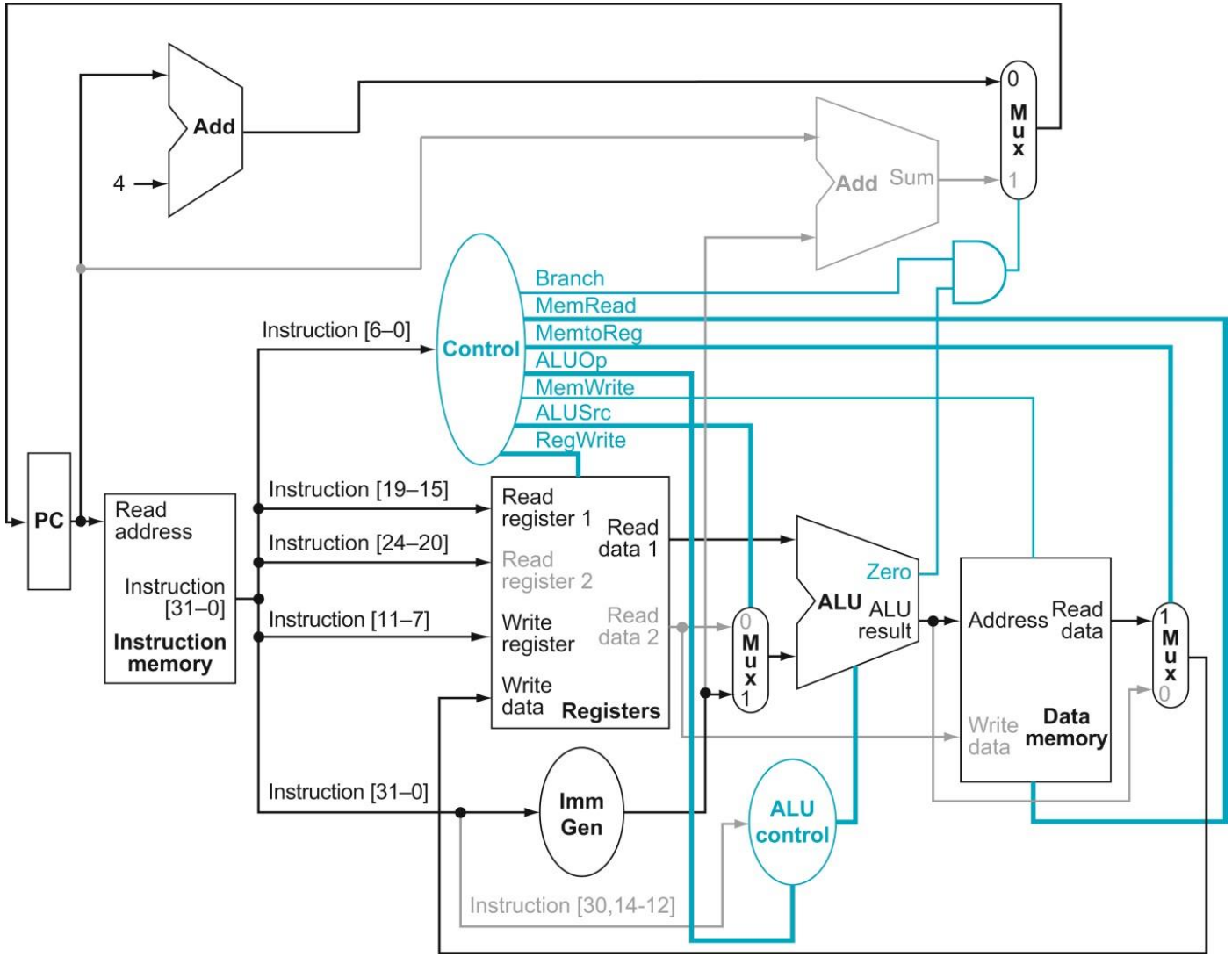
Datapath with Control



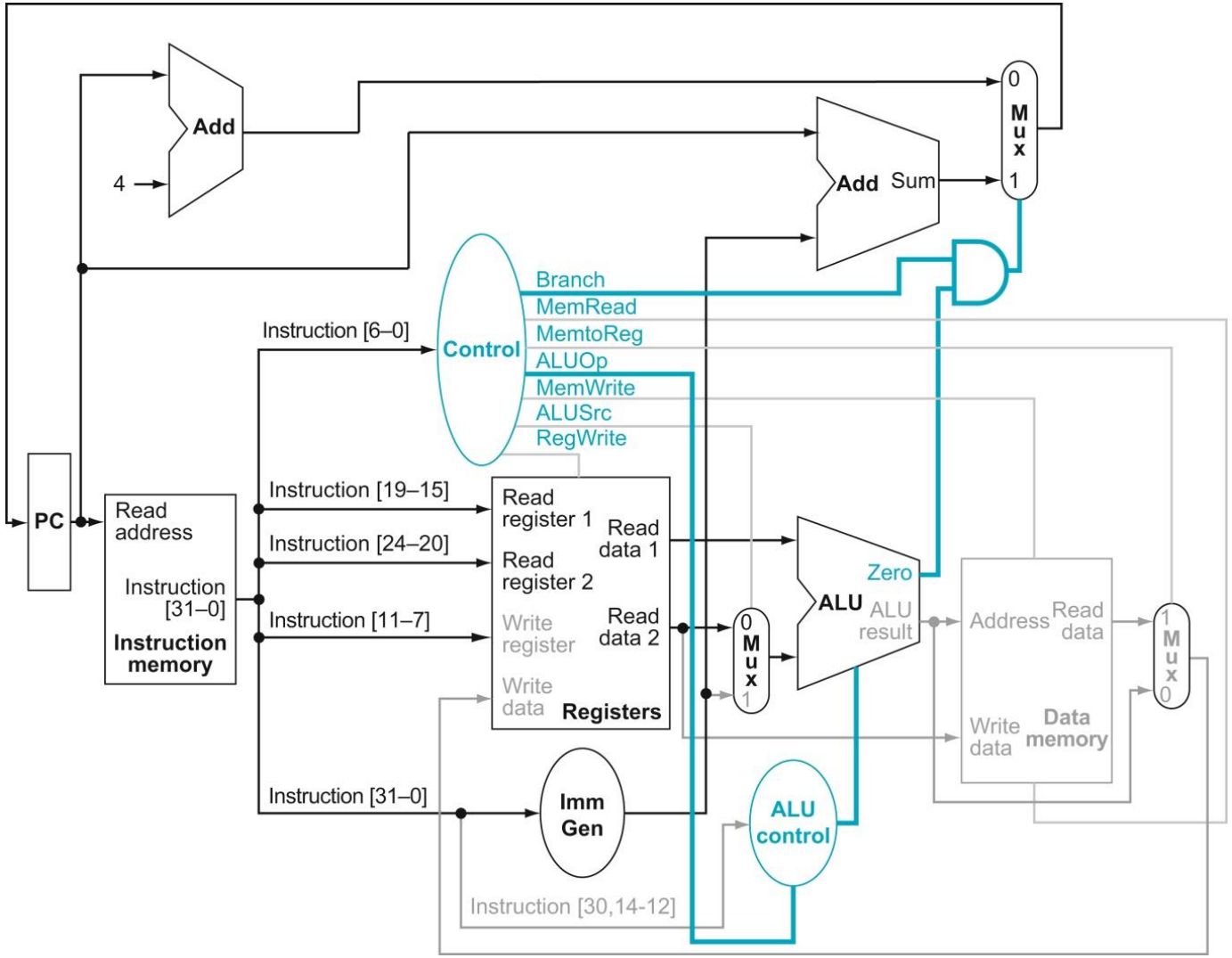
R-Type Instruction



Load Instruction



BEQ Instruction

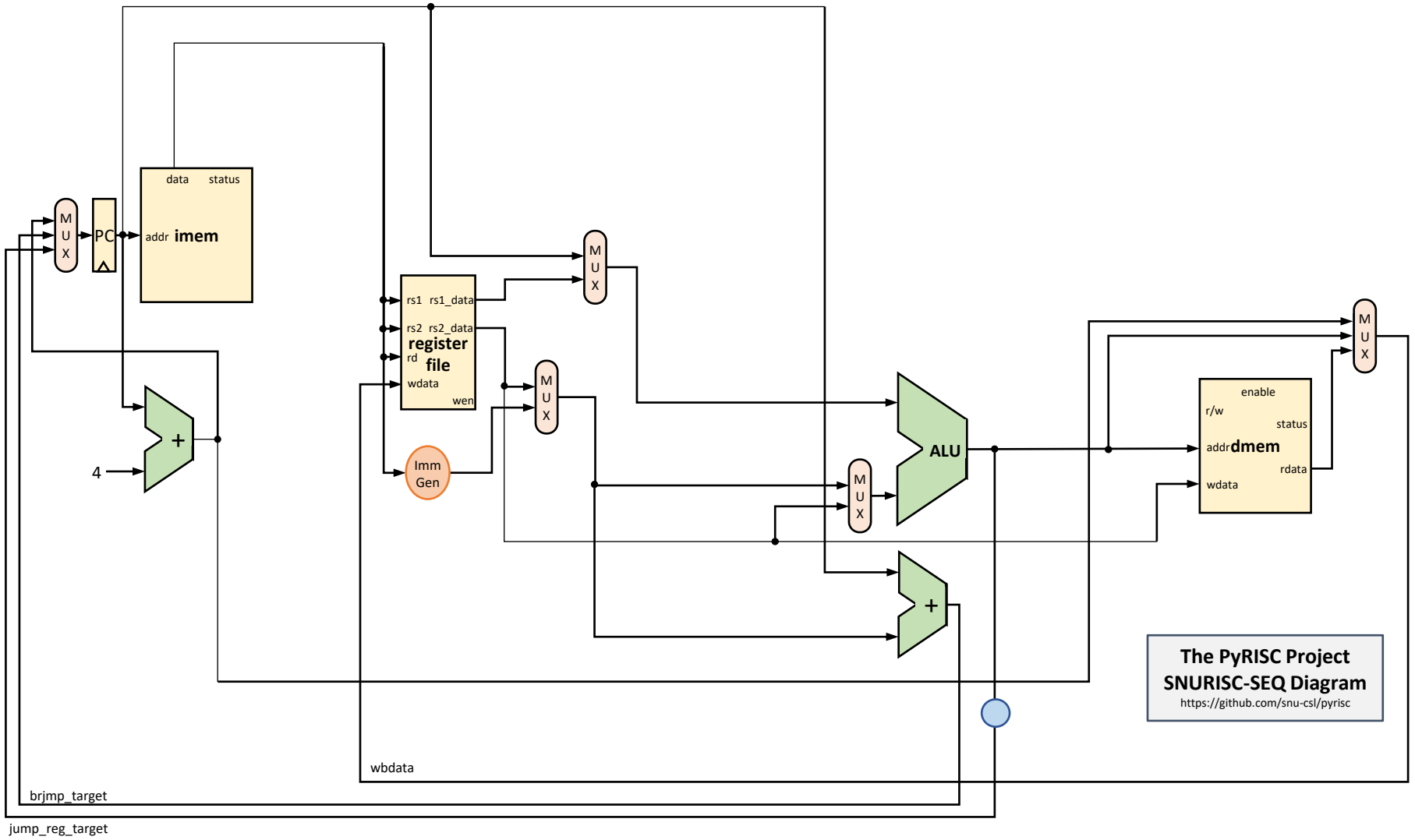


Performance Issues

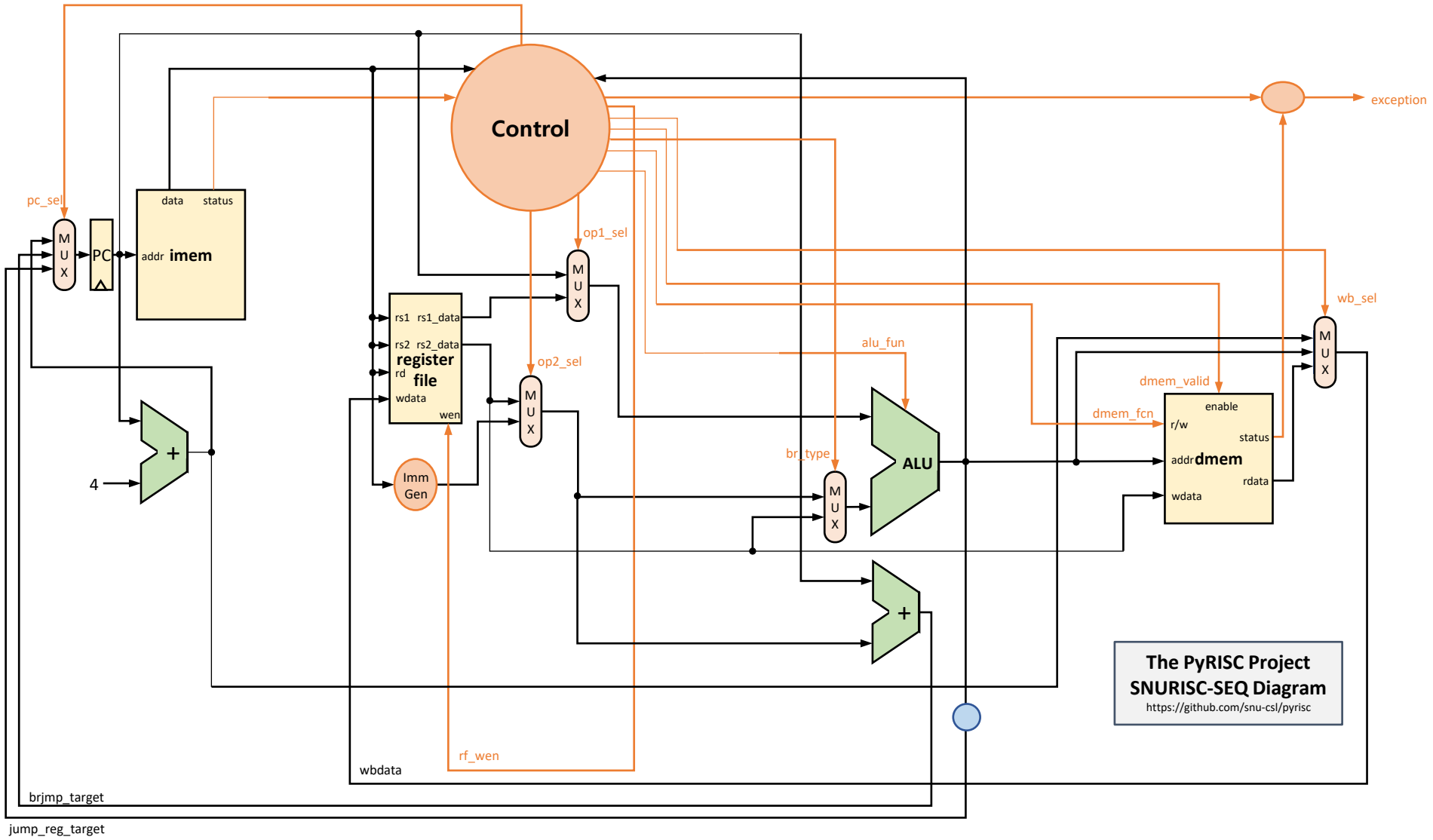
- Longest delay determines clock period
  - Critical path: load instruction
  - Instruction memory → register file → ALU → data memory → register file
- Not feasible to vary period for different instructions
- Violates design principle: making the common case fast
- We will improve performance by pipelining

SNURISC-SEQ

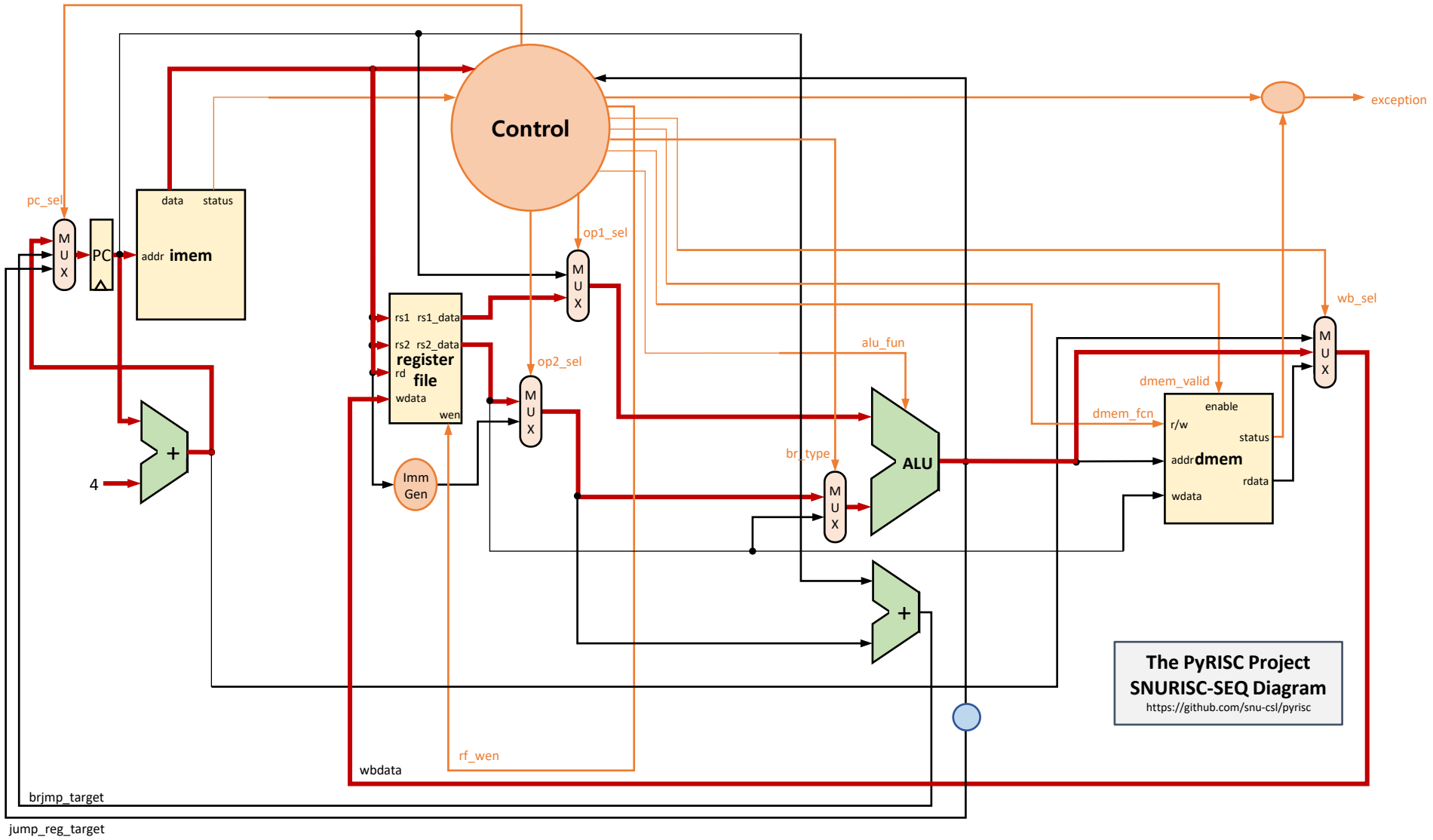
SNURISC-SEQ: Datapath



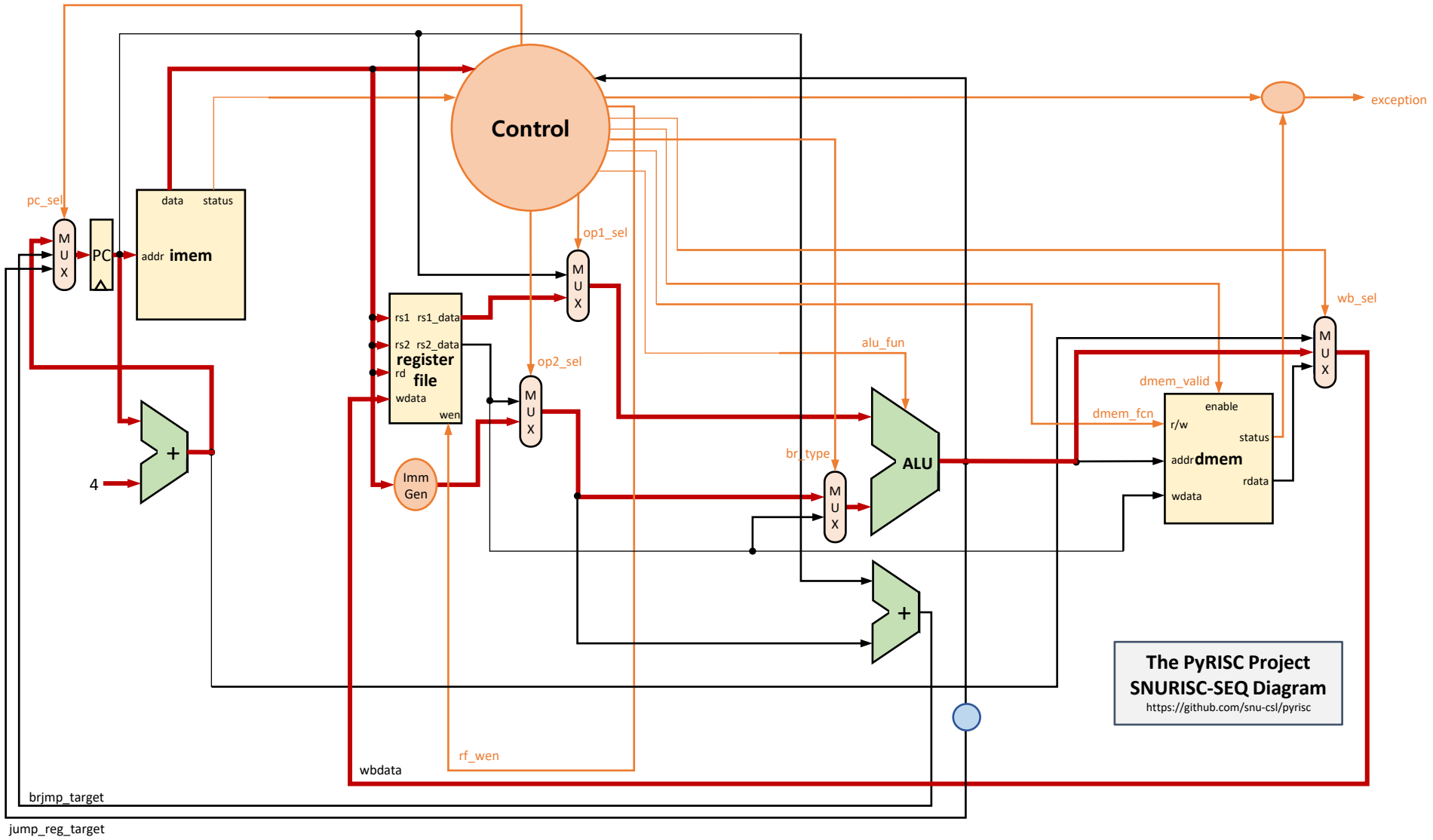
SNURISC-SEQ: Datapath + Control



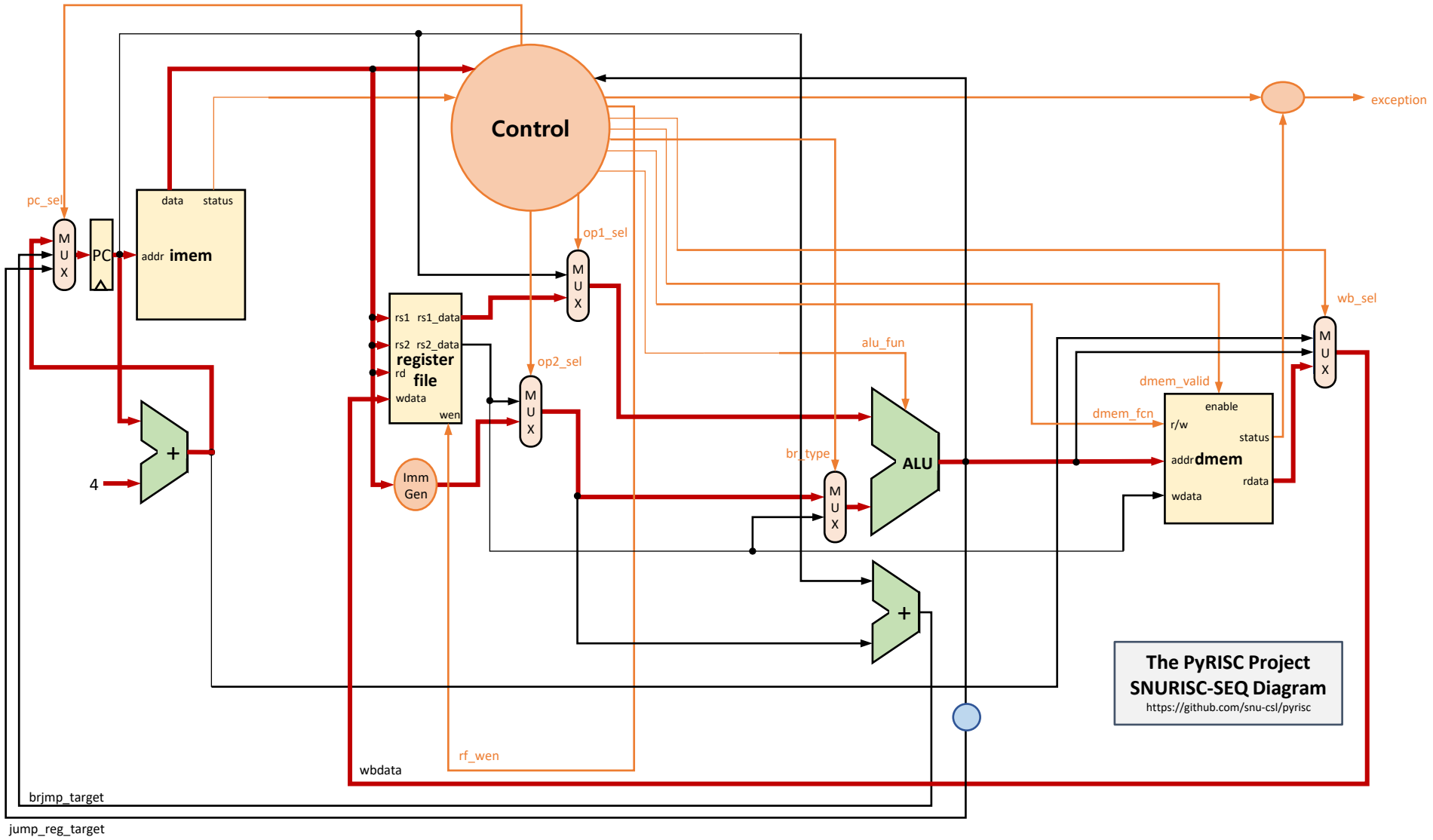
SNURISC-SEQ: add t0,t1,t2



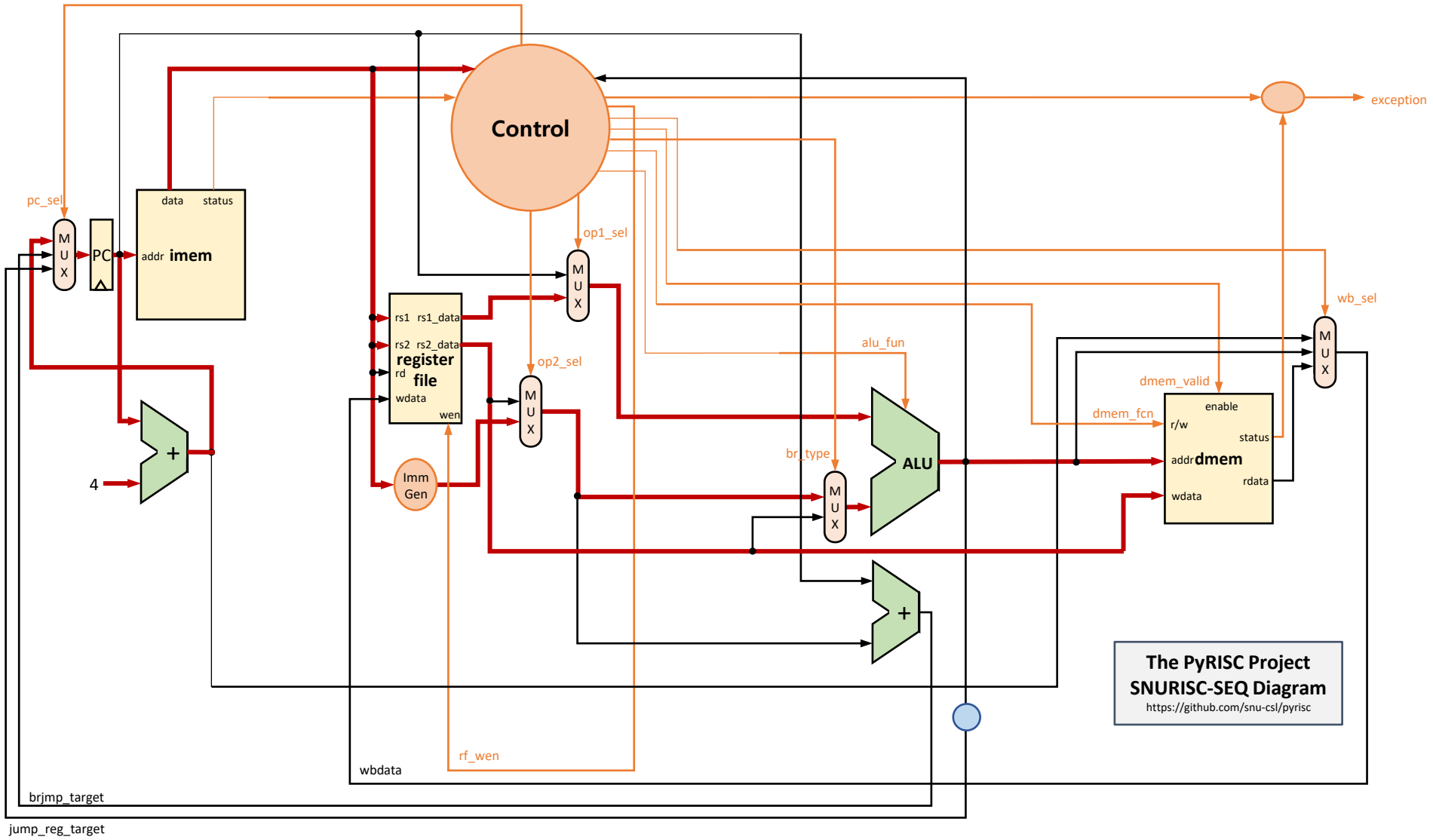
SNURISC-SEQ: addi t0,t1,1



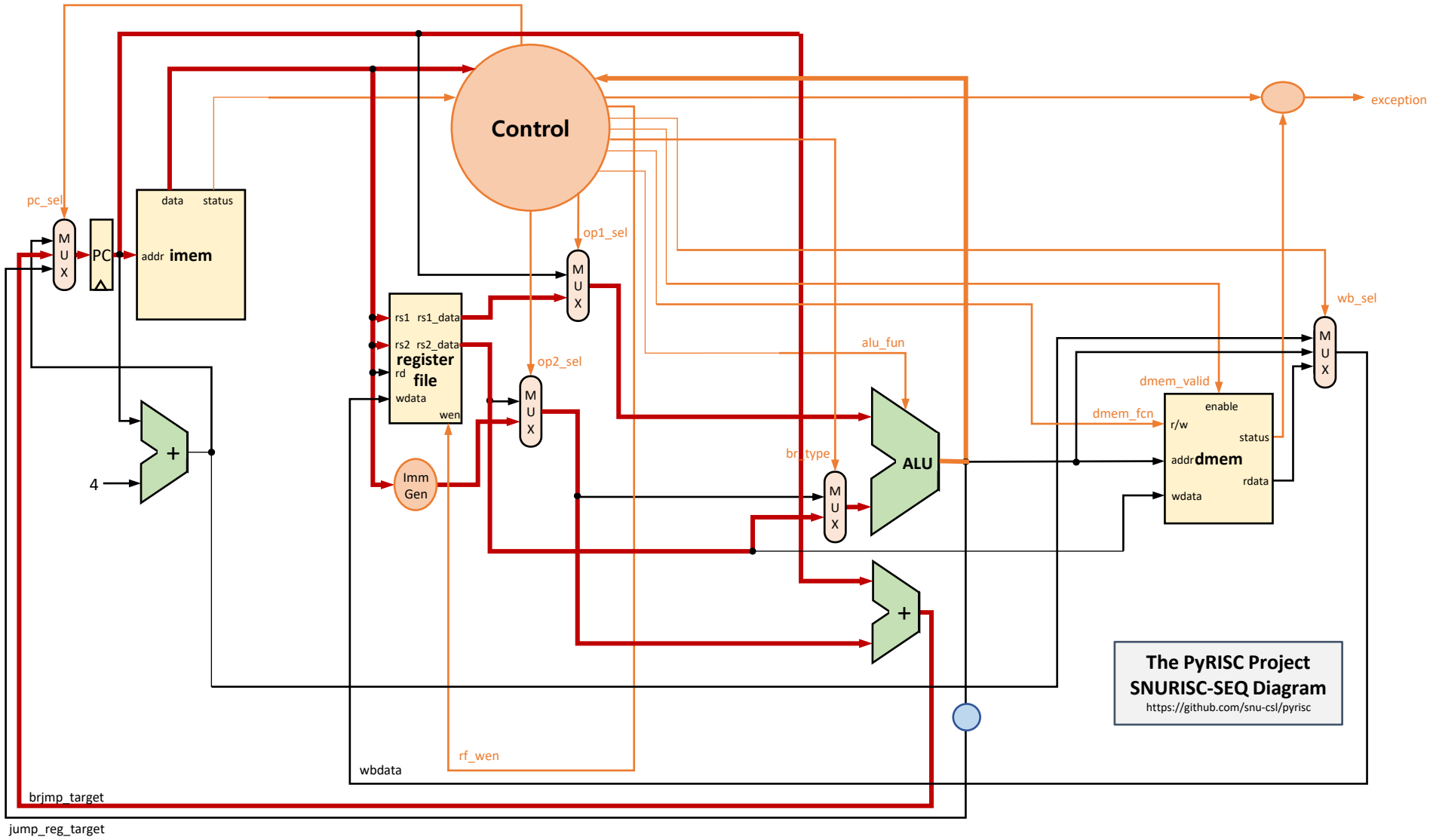
SNURISC-SEQ: lw t0,32(sp)



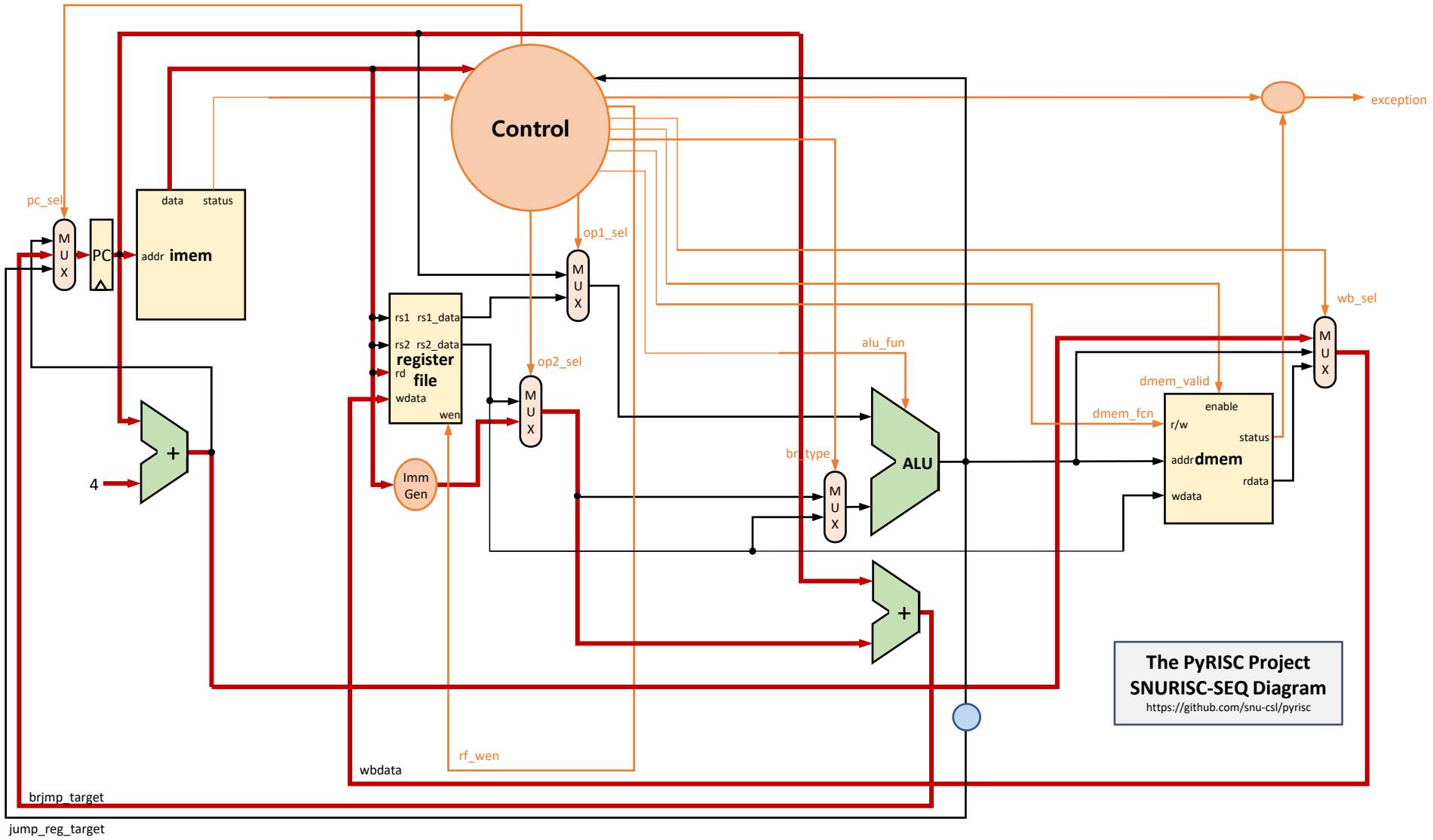
SNURISC-SEQ: sw t0,8(sp)



SNURISC-SEQ: `beq t0,t1,L1` (taken)



SNURISC-SEQ: `jal ra,func`



SNURISC-SEQ: jalr t0,0(ra)