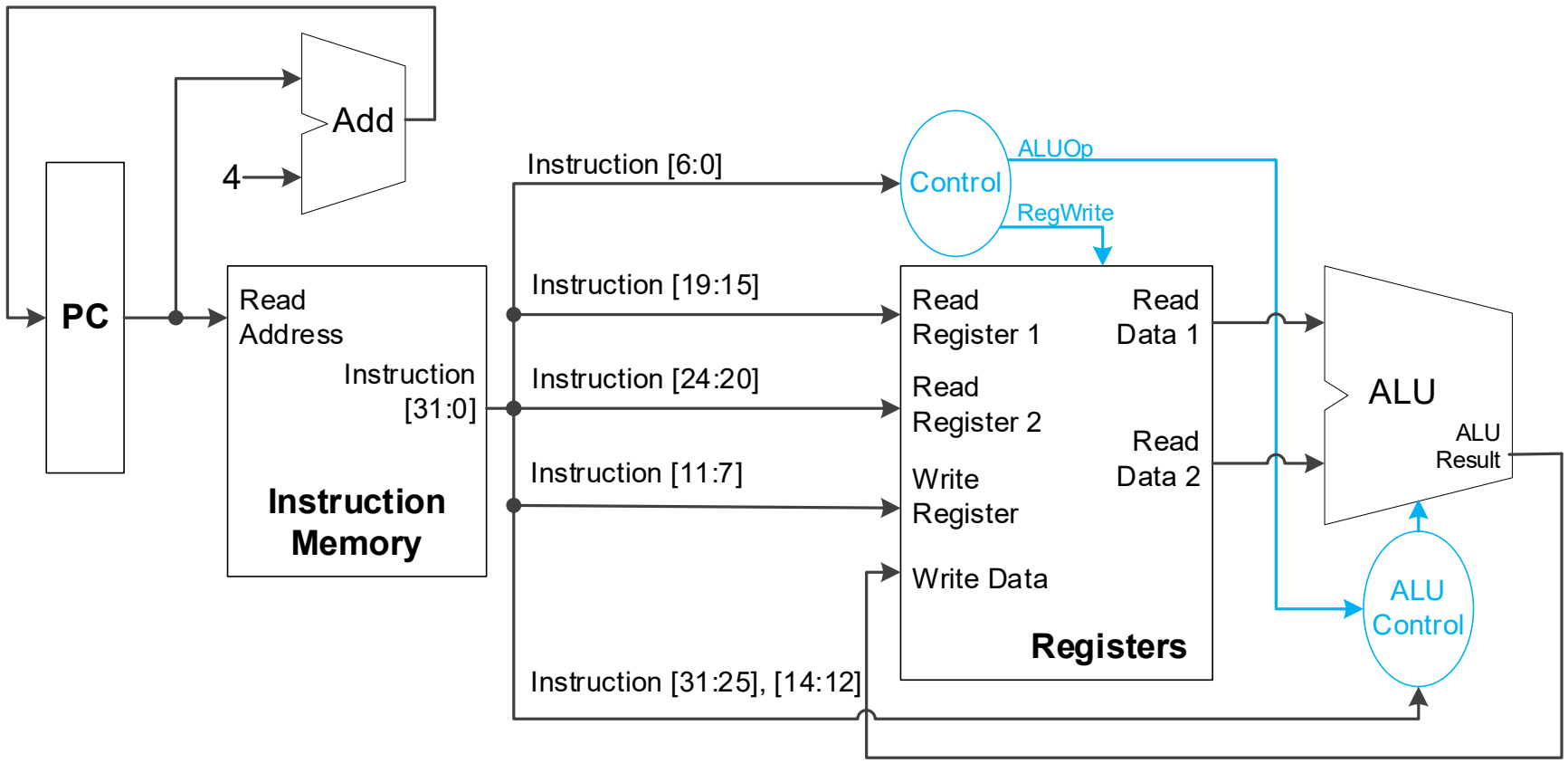
Introduction to Computer Architecture
Chapter 4 - 2

The Processor

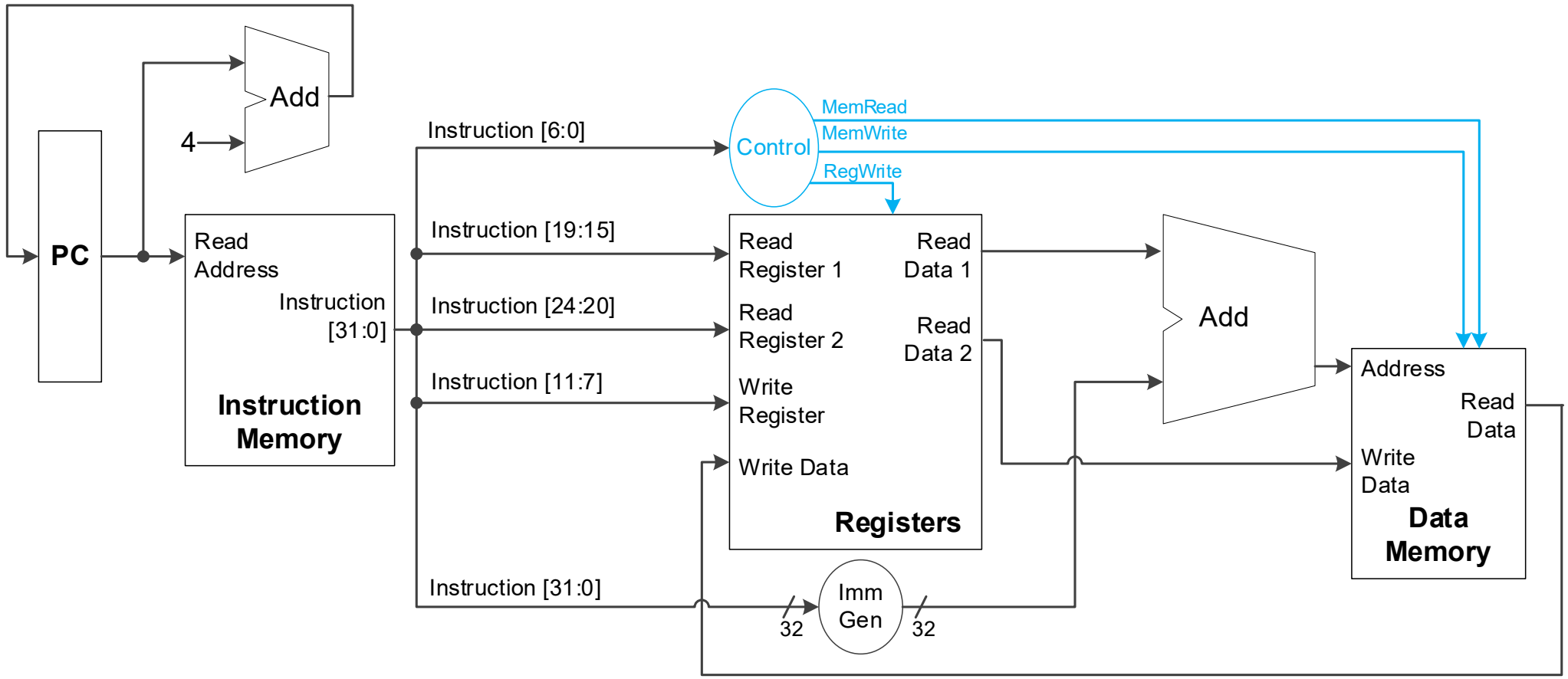
Hyungmin Cho

Department of Computer Science and Engineering
Sungkyunkwan University

CPU for R-type Instructions

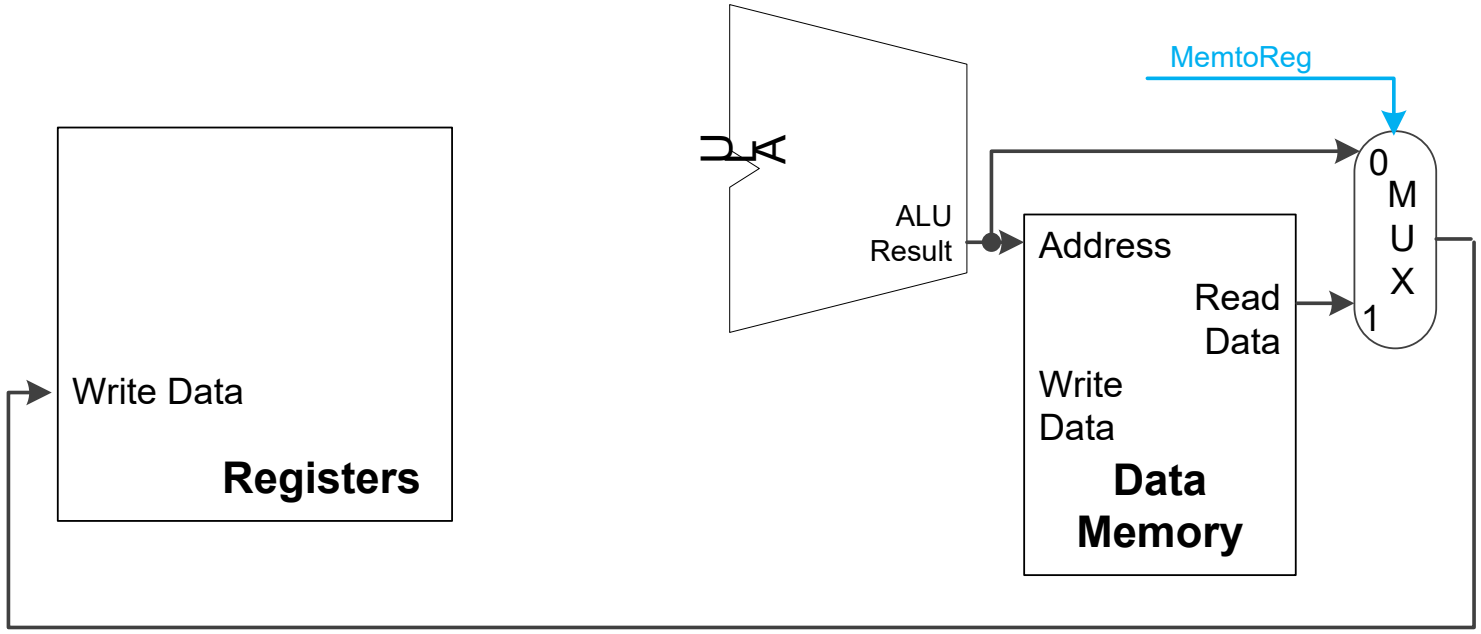


CPU for Load/Store



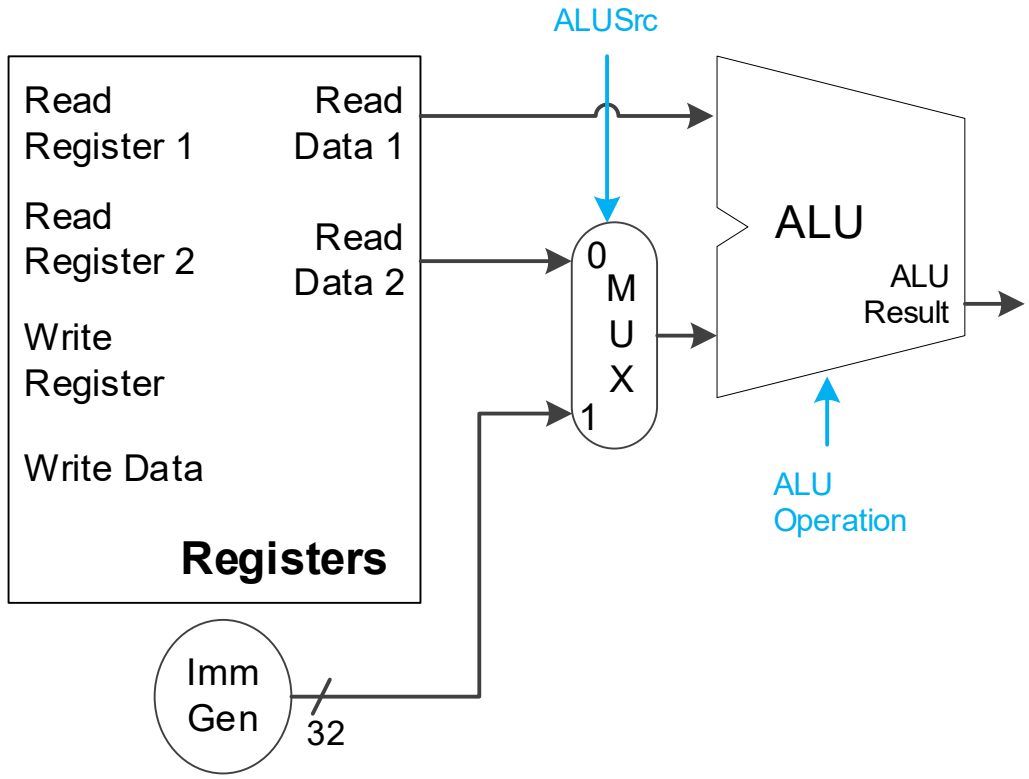
Combining R-type and Load/Store

- What to write in the destination register?
  - R-type: ALU result
  - Load: Read data from Data Memory



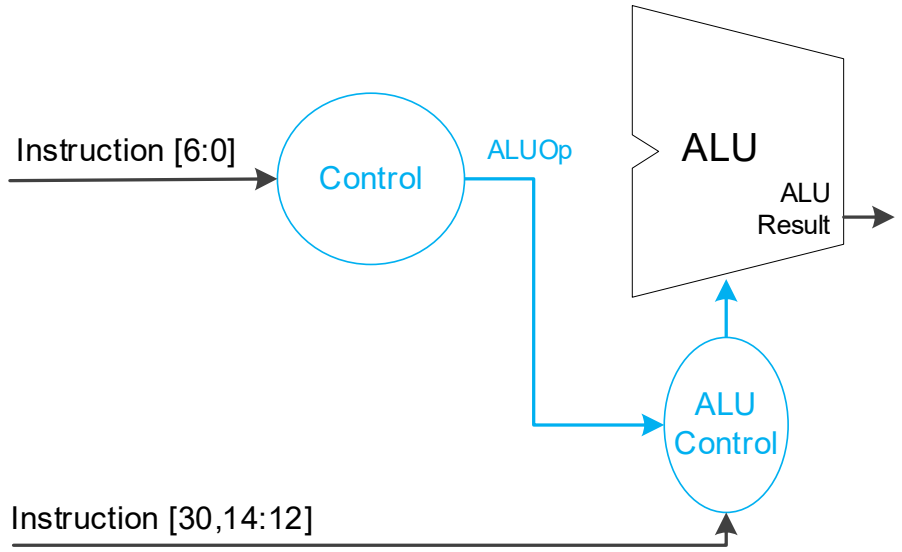
Combining R-type and Load/Store

- One ALU is used for both…
  - R-type Arithmetic/Logic operations
  - Load/Store address calculation

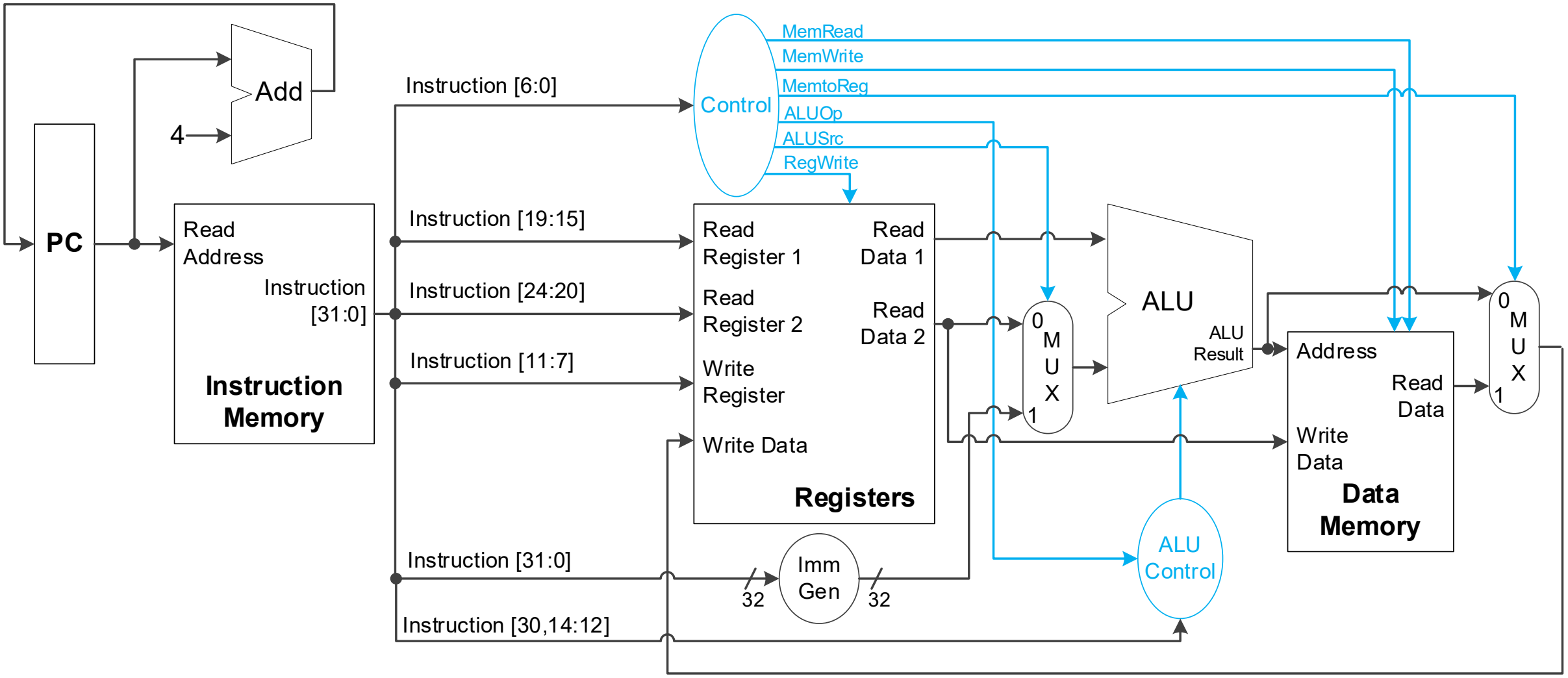


ALU Control Bits

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 if 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



Combining R-type and Load/Store

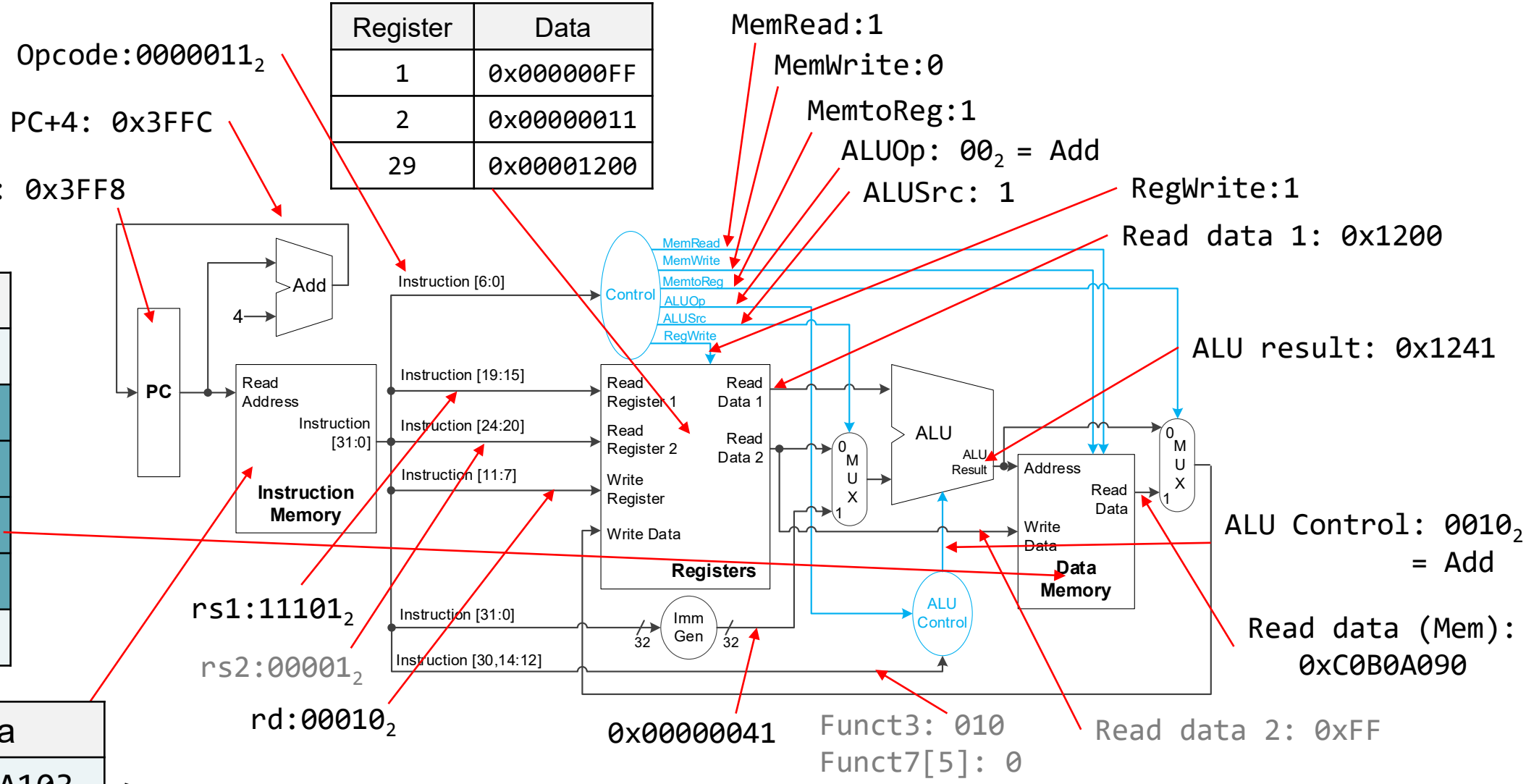


Load Execution Example

Register	Data
1	0x000000FF
2	0x00000011
29	0x00001200

Address	Data
0x1240	0x80
0x1241	0x90
0x1242	0xA0
0x1243	0xB0
0x1244	0xC0
0x1245	0xD0

Address	Data
0x3FF8	0x041EA103

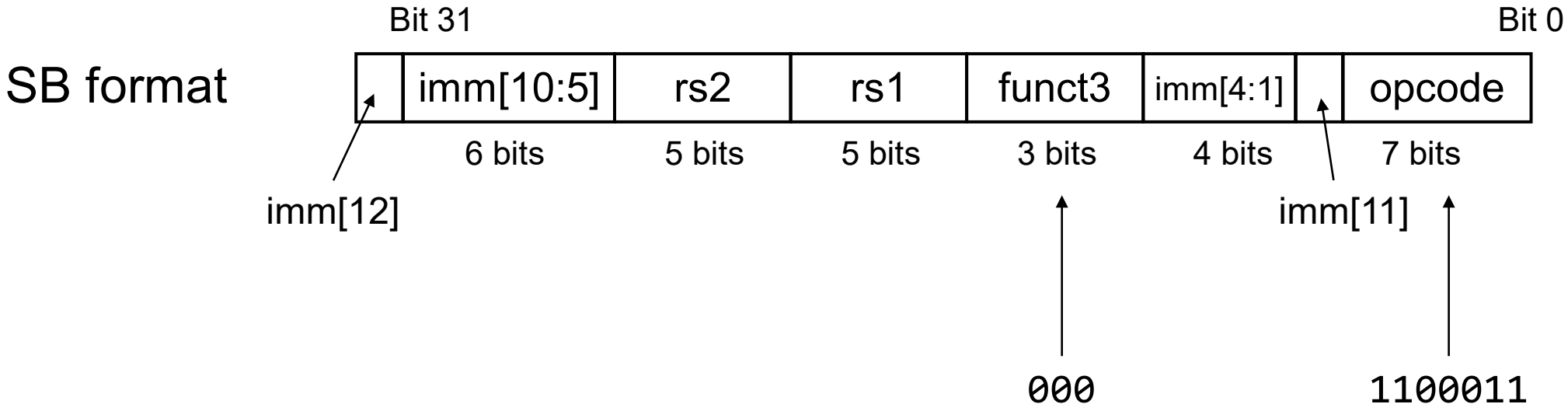


0000 0100 0001 1110 1010 0001 0000 0011 → lw x2, 65(x29)

000001000001 11101 010 00010 0000011 lw x2, 0x41(x29)

Imm[11:0] rs1=x29 funct3 rd opcode

Branch Instructions (BEQ)

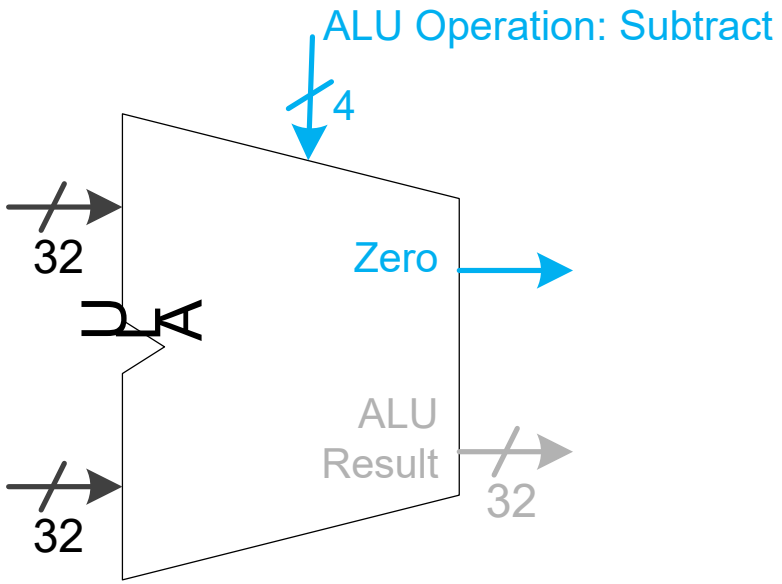


- If `*rs1 == *rs2`
  - New `PC` = `PC` + immediate

- Immediate is a 13-bit signed value, with `imm[0]` is assumed to be zero.
  - Need to sign-extend

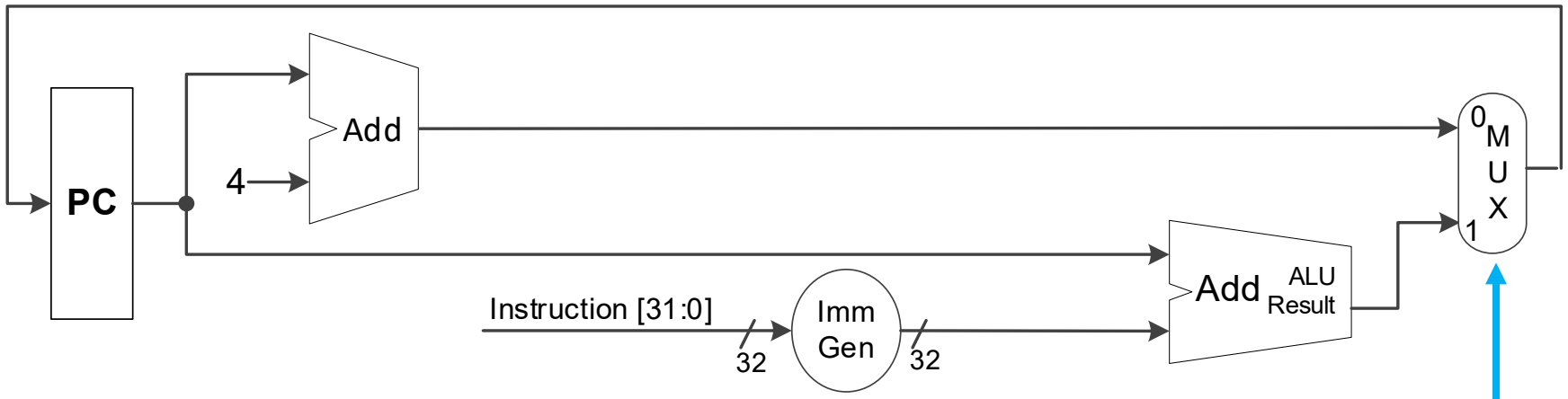
Branch Condition Test

- Subtract and check if the result is zero



- ALU has one extra output – result is equal to zero?

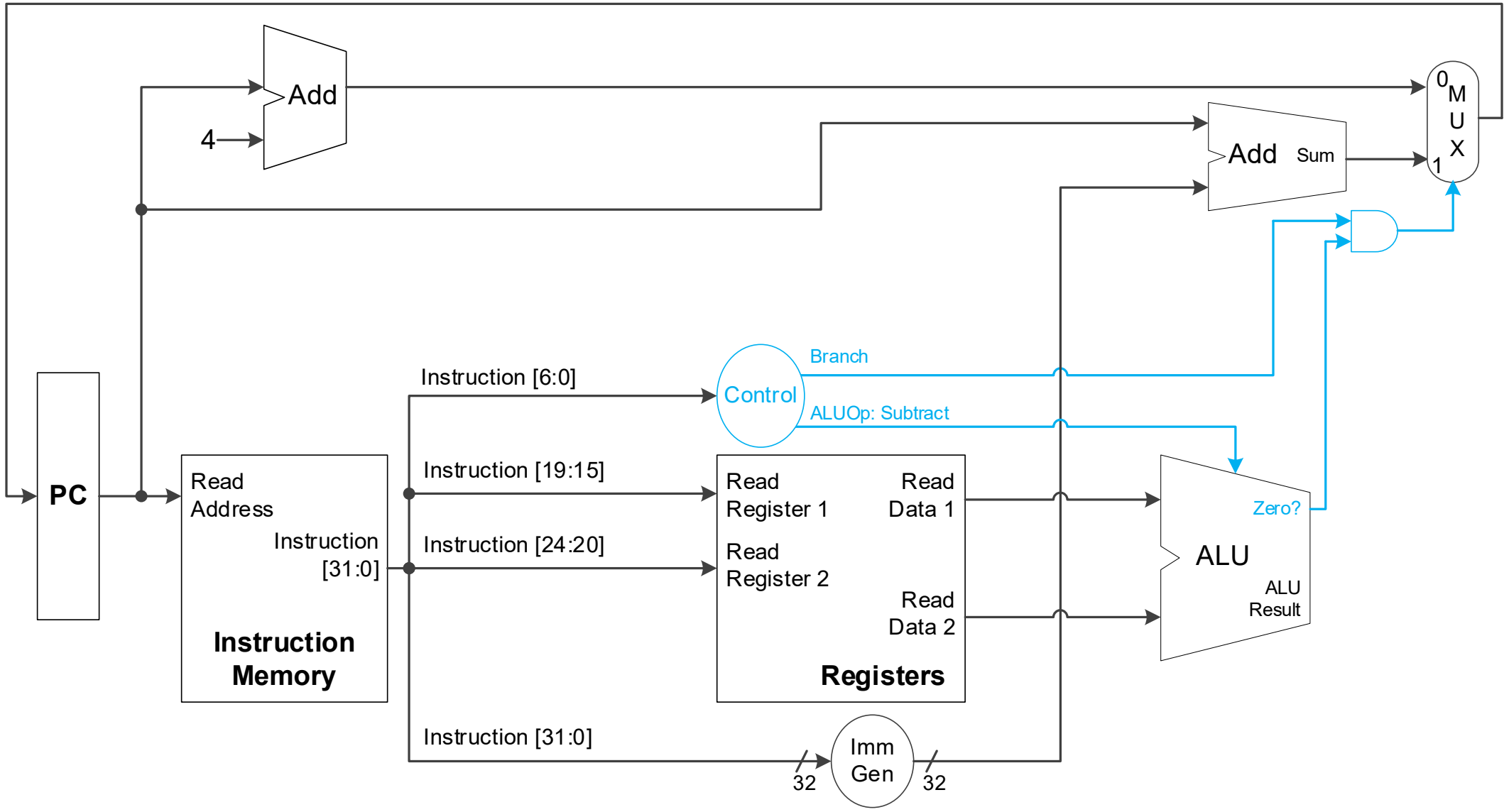
Branch Target (New PC) Calculation



1: zero is set, and the instruction is ‘beq’ branch

0: zero is not set or instruction is something else..

CPU for Branch Instruction



BEQ Execution Example

Register	Data
8	0x1
9	0x1

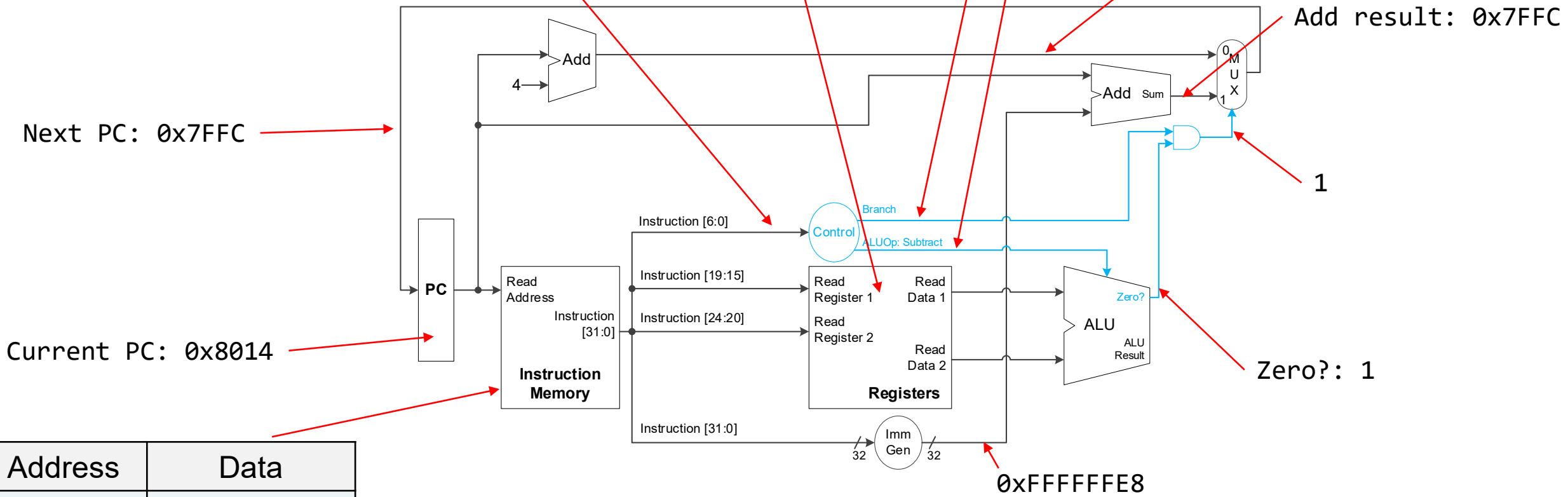
Branch:1

ALUOp:01_2 = subtract

Opcode:1100011_2

PC+4: 0x8018

Add result: 0x7FFC



Address	Data
0x8014	0xFE9404E3

1111 1110 1001 0100 0000 0100 1110 0011

1 111111 01001 01000 000 0100 1 1100011

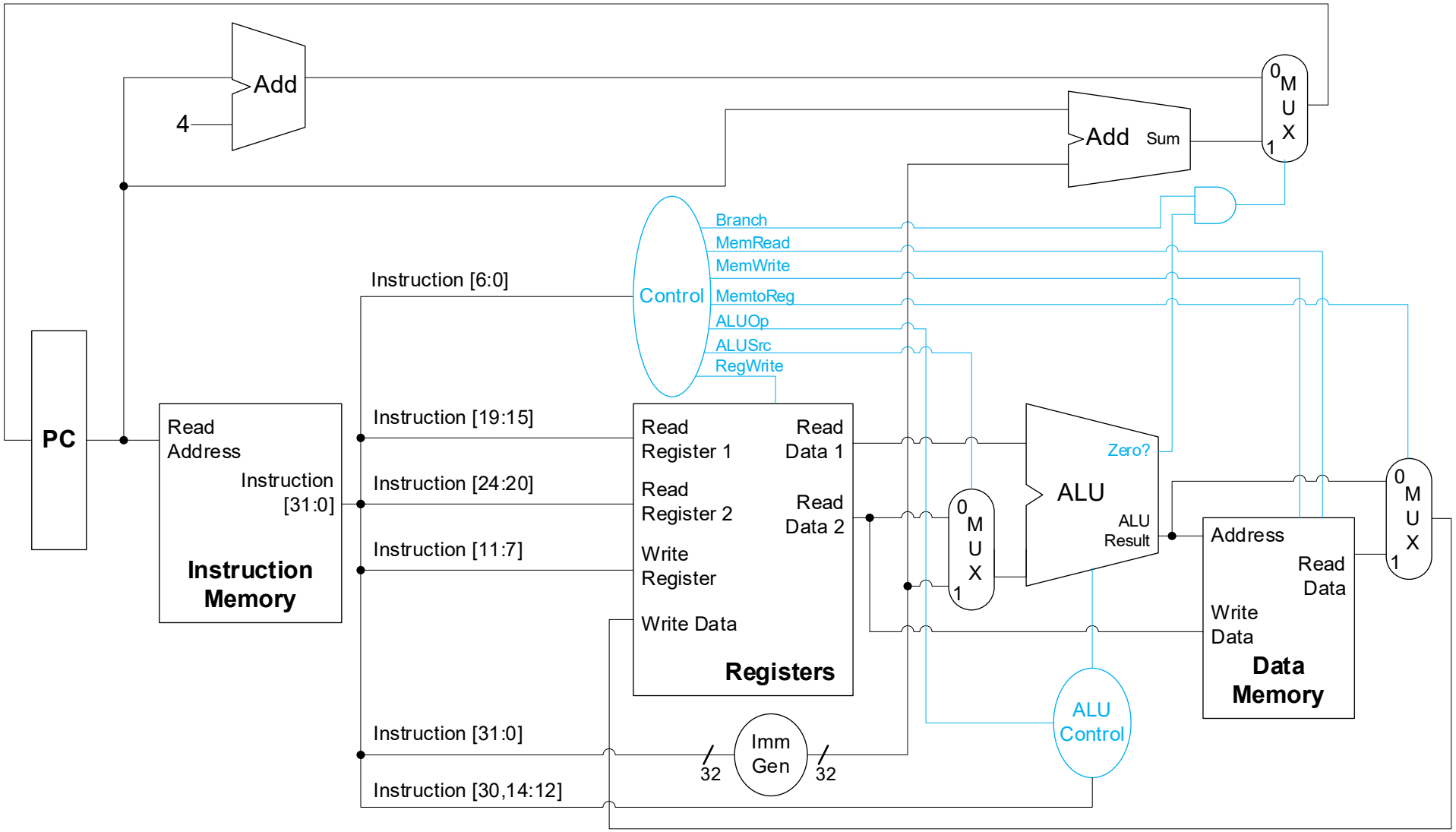
Imm[12] Imm[10:5] rs2=9 rs1=8 funct3 Imm[4:1] Imm[11] Opcode = beq

Imm[12:0] = $1111111101000_2 = -24_{10}$

→ beq $8, $9, *LABEL*

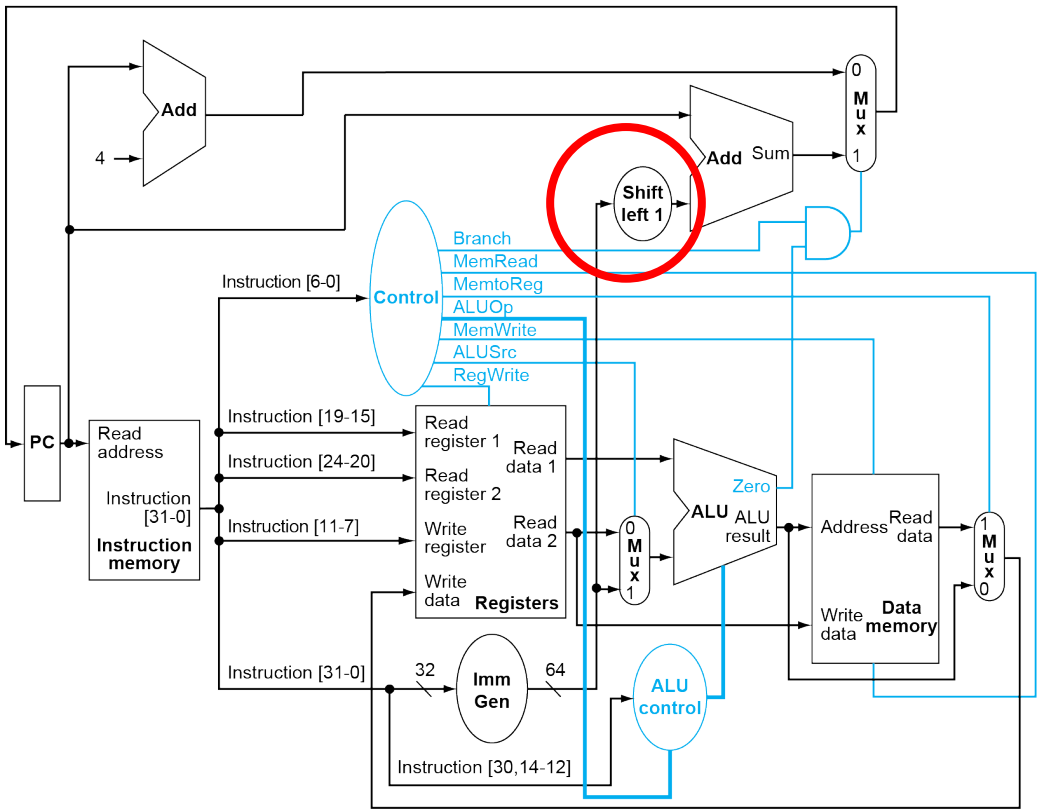
LABEL: points to 6 instrs before beq

CPU for R-type/Load/Store/Branch

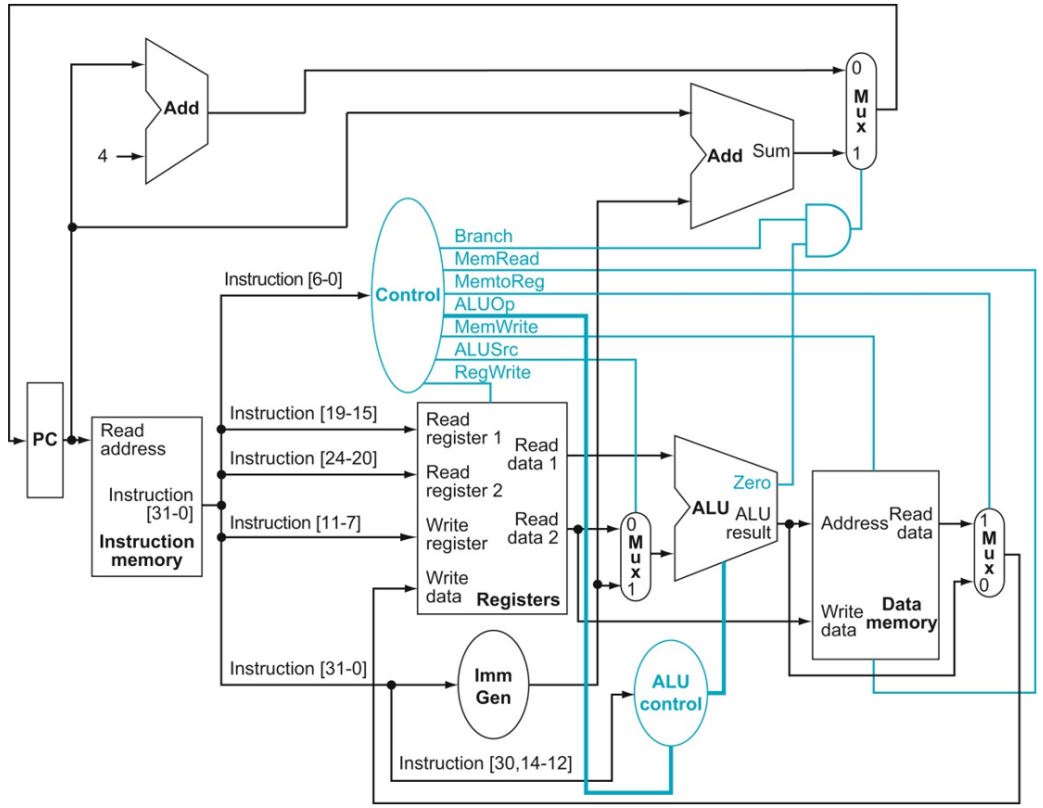


TextBook 1st & 2nd edition Difference

- The CPU datapath drawn in the 1st edition has an additional “shift left 1” after Imm Gen



1st edition



2nd edition

- There is no need for a shift logic after the immediate value is fully assembled and sign-extended to 32-bit data.

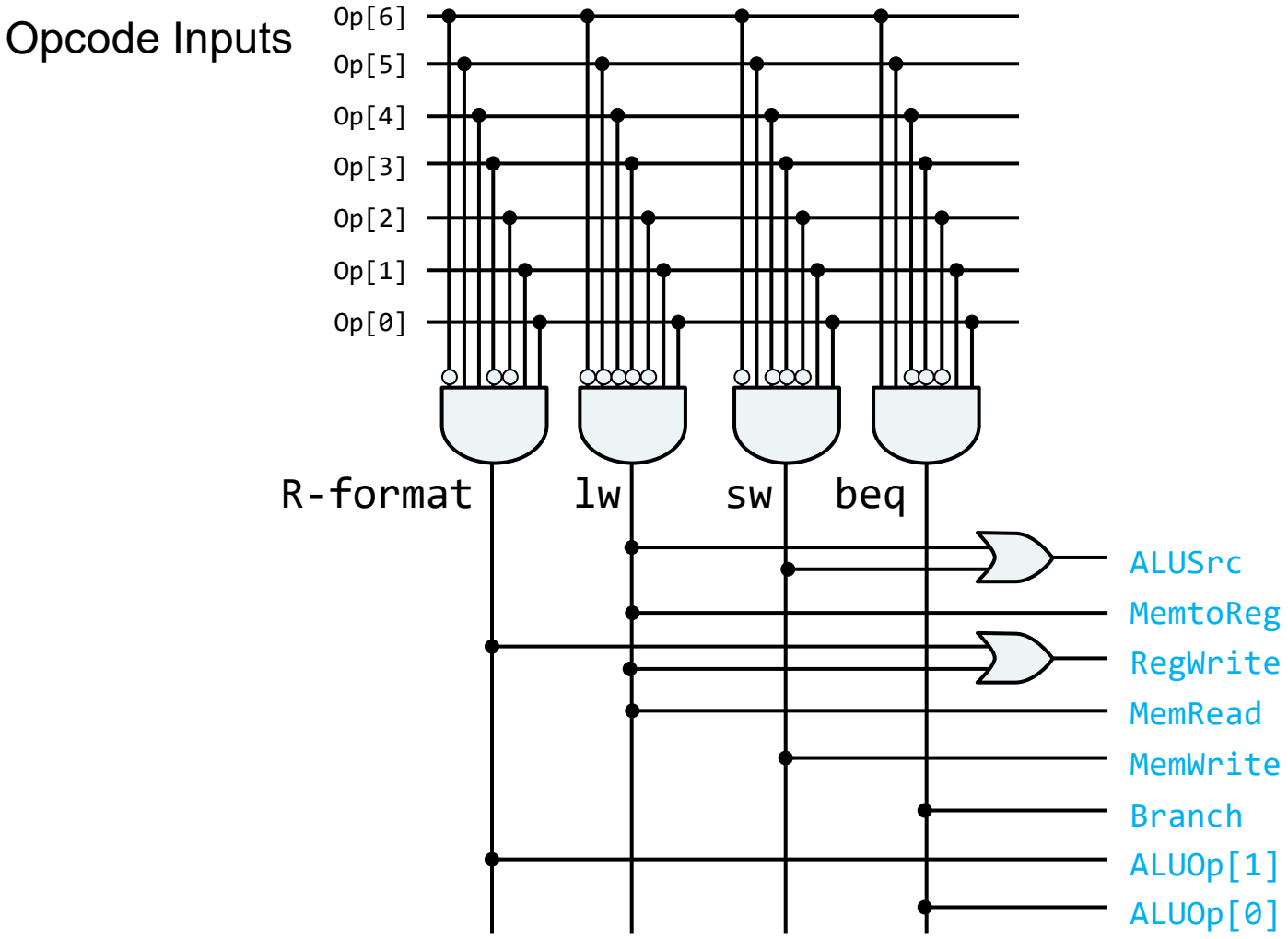
Control Signals

Instruction	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp[1]	ALUOp[0]
Arithmetic (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

- Opcode
  - Arithmetic: 0110011
  - lw: 0000011
  - sw: 0100011
  - beq: 1100011

- Need to build a combinational logic!

Implementation of Combinational Control Unit



- Synthesis tool automatically optimizes RTL code → Combinational logic

AUIPC Instruction

- AUIPC – Add Upper Immediate to PC

U format

LUI
AUIPC

imm[31:12] + 0 (lower 12 bits)

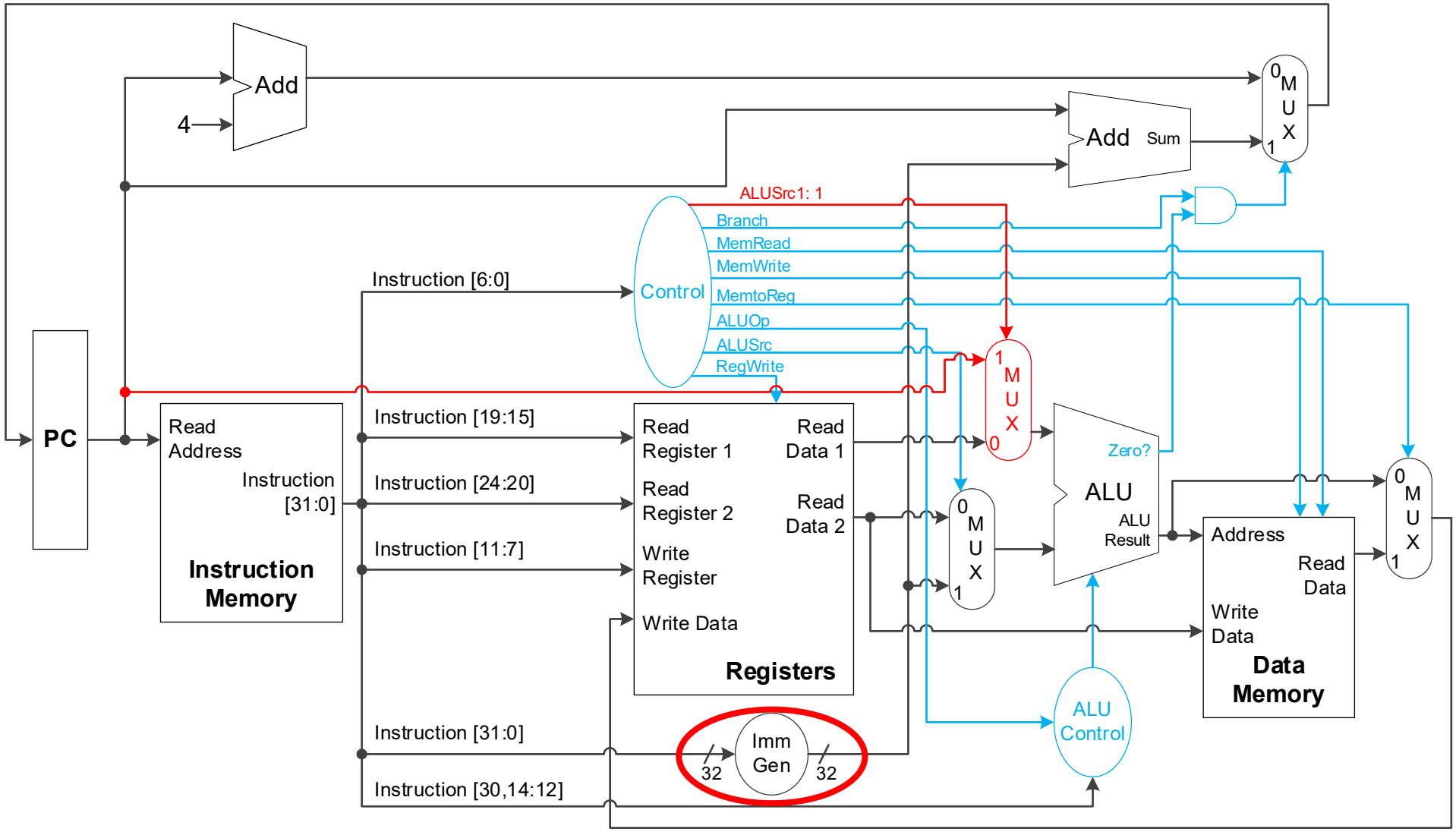
- Register *rd* = immediate + PC

Jump to Arbitrary Location

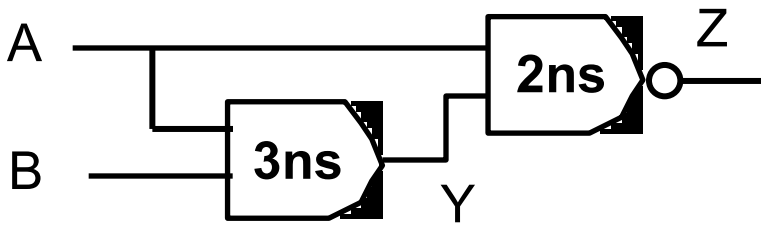
- BEQ relative branch offset range: 13-bit limitation (LSB is fixed to '0')
- JAL relative offset range: 21-bit limitation (LSB is fixed to '0')

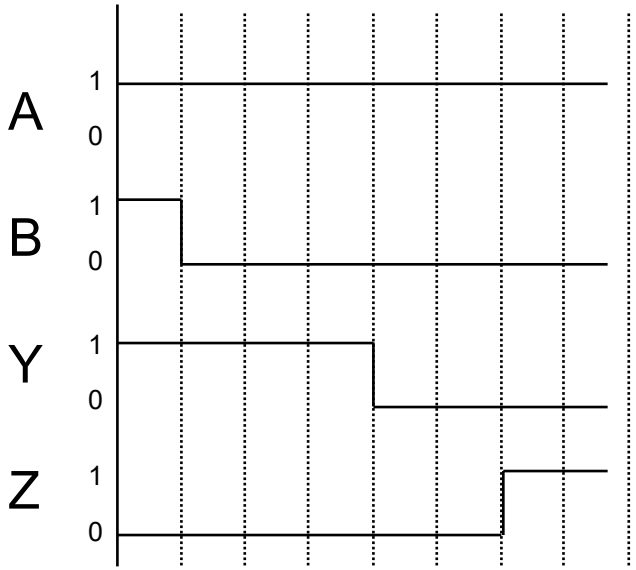
- (LUI, AUIPC) + JALR : Jump to arbitrary 32bit (relative, absolute) range
  - LUI absolute address jump: Jump to `0x12345678`
    - `lui x5, 0x12345`
    - `jalr x1, 0x678(x5)`
  - AUIPC relative address jump: Jump to `PC+0x12345678`
    - `auipc x5, 0x12345`
    - `jalr x1, 0x678(x5)`
  - AUIPC relative address jump: Jump to `PC-0x12345678` (or, `PC+0xEDCBA988`)
    - `auipc x5, 0xEDCBB`
    - `Jalr x1, -0x678(x5)`

Datapath for AUIPC



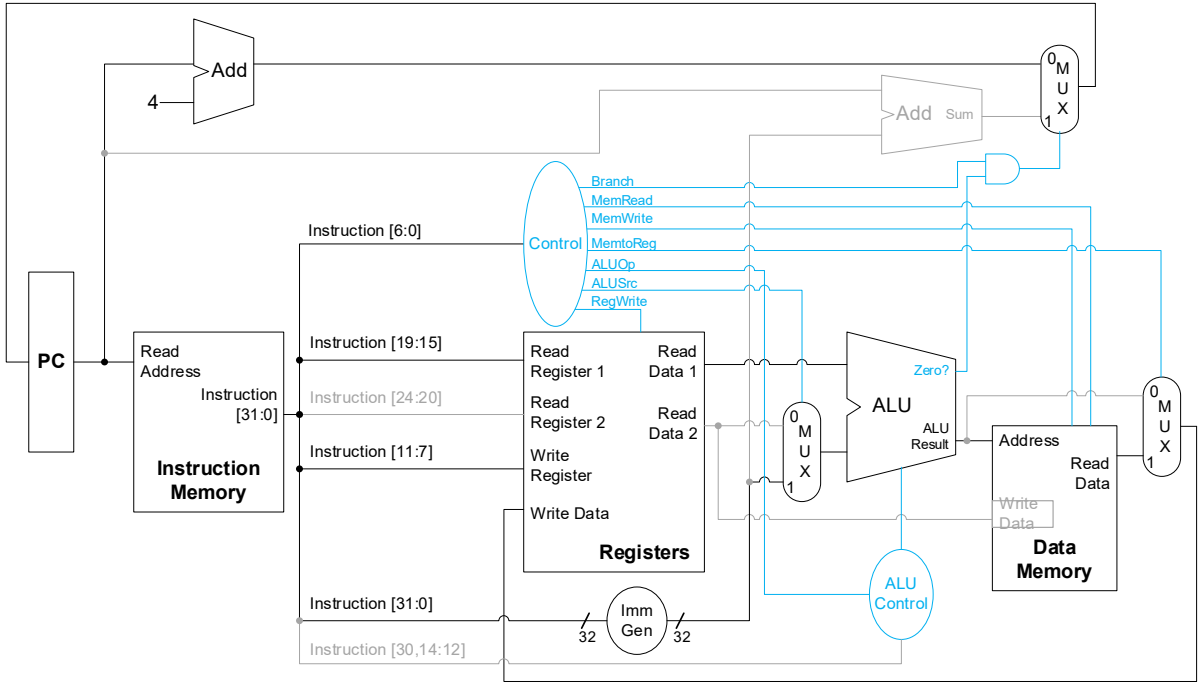
Combinational Logic Propagation Delay



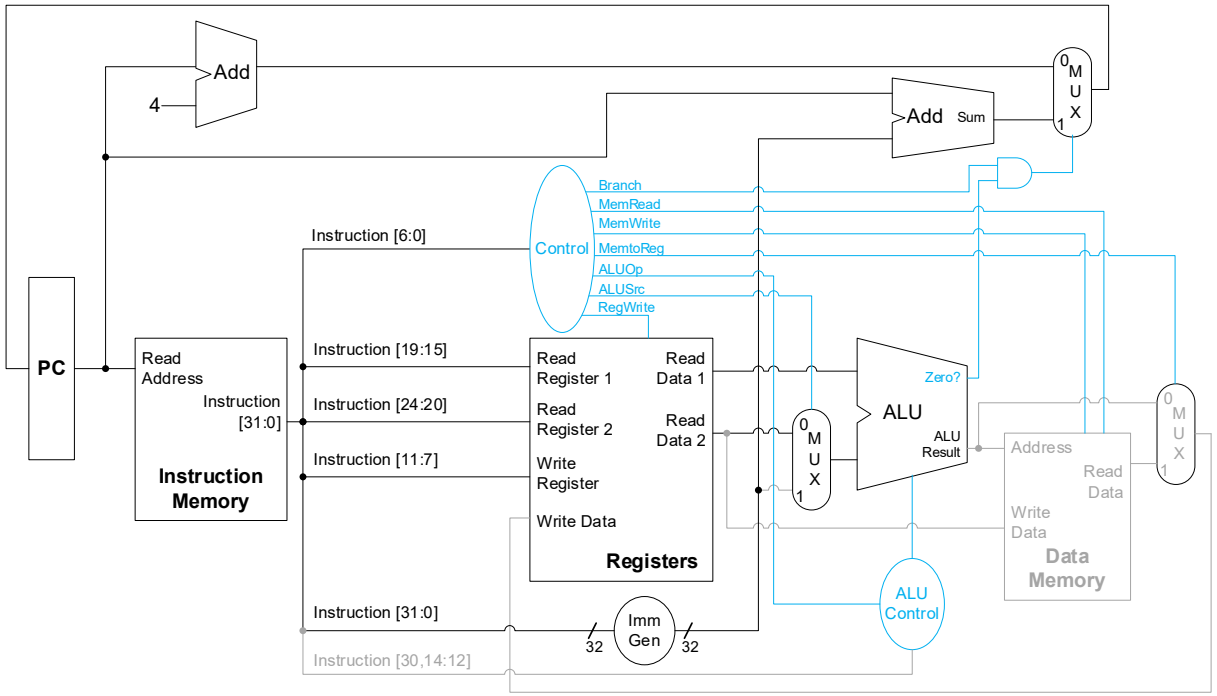


Propagation Delay

Single-Cycle for Every Instruction?



LW Instruction



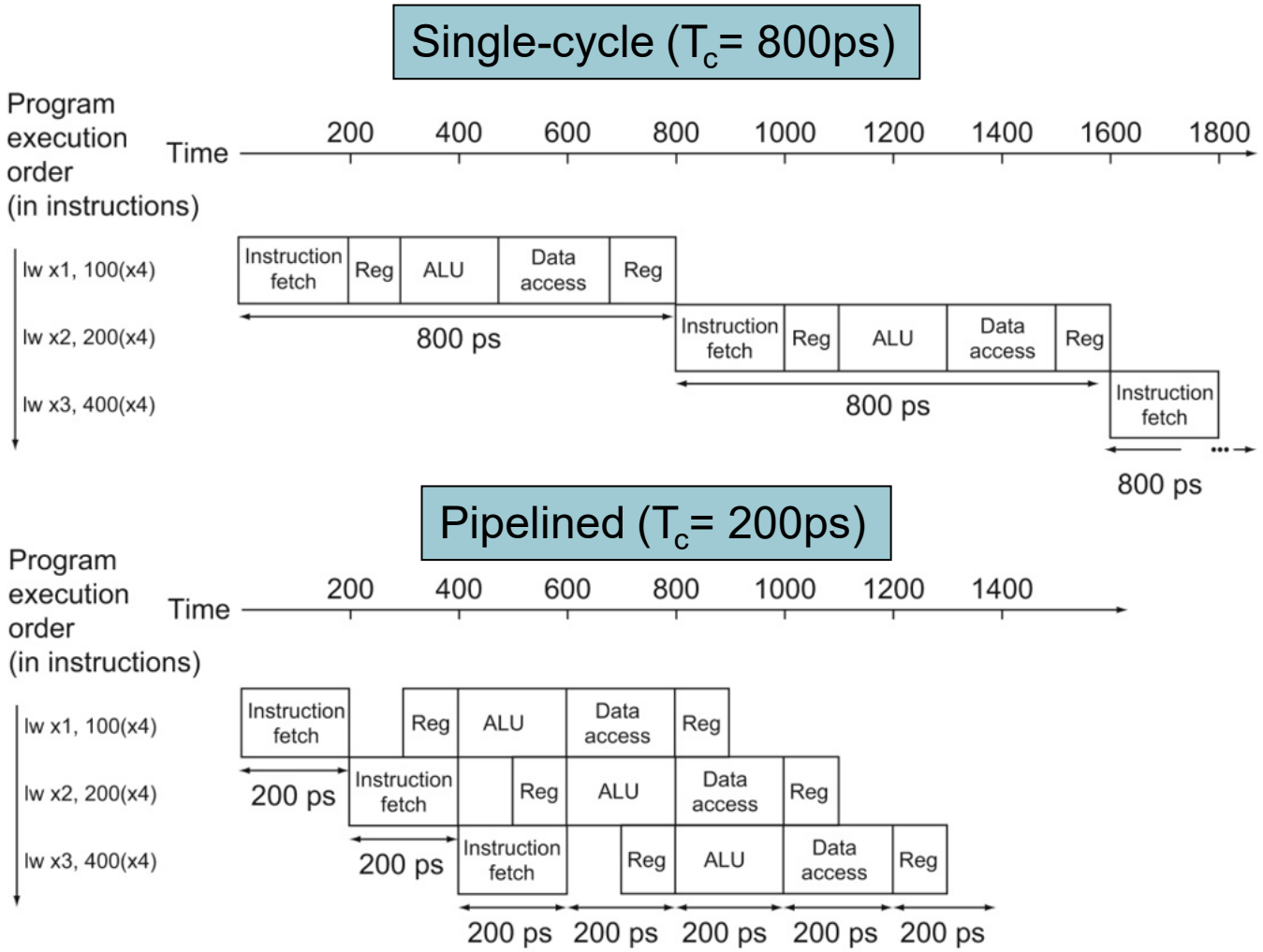
BEQ Instruction

Pipeline Performance

- Assume the time for each stages is
  - 100ps for register read or write
  - 200ps for other stages

Instruction type	Instruction fetch	Register read	ALU op	Memory access	Register write	Total time
`lw`	200ps	100 ps	200ps	200ps	100 ps	800ps
`sw`	200ps	100 ps	200ps	200ps		700ps
`R-format`	200ps	100 ps	200ps		100 ps	600ps
`beq`	200ps	100 ps	200ps			500ps

Pipeline Performance



Performance Issues

- Longest delay determines the clock period
  - **Critical path**: The datapath that takes the **longest** time
  - Critical path in the single-cycle processor: load instruction
    - Instruction memory → register file → ALU → data memory → register file

- Not feasible to vary the clock period for different instructions

- We will improve performance using pipelining