Computer Architecture

Chapter 4A.
The Processor- Unpipelined Datapath

Hyuk-Jun Lee, PhD

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea

Email: hyukjunl@sogang.ac.kr

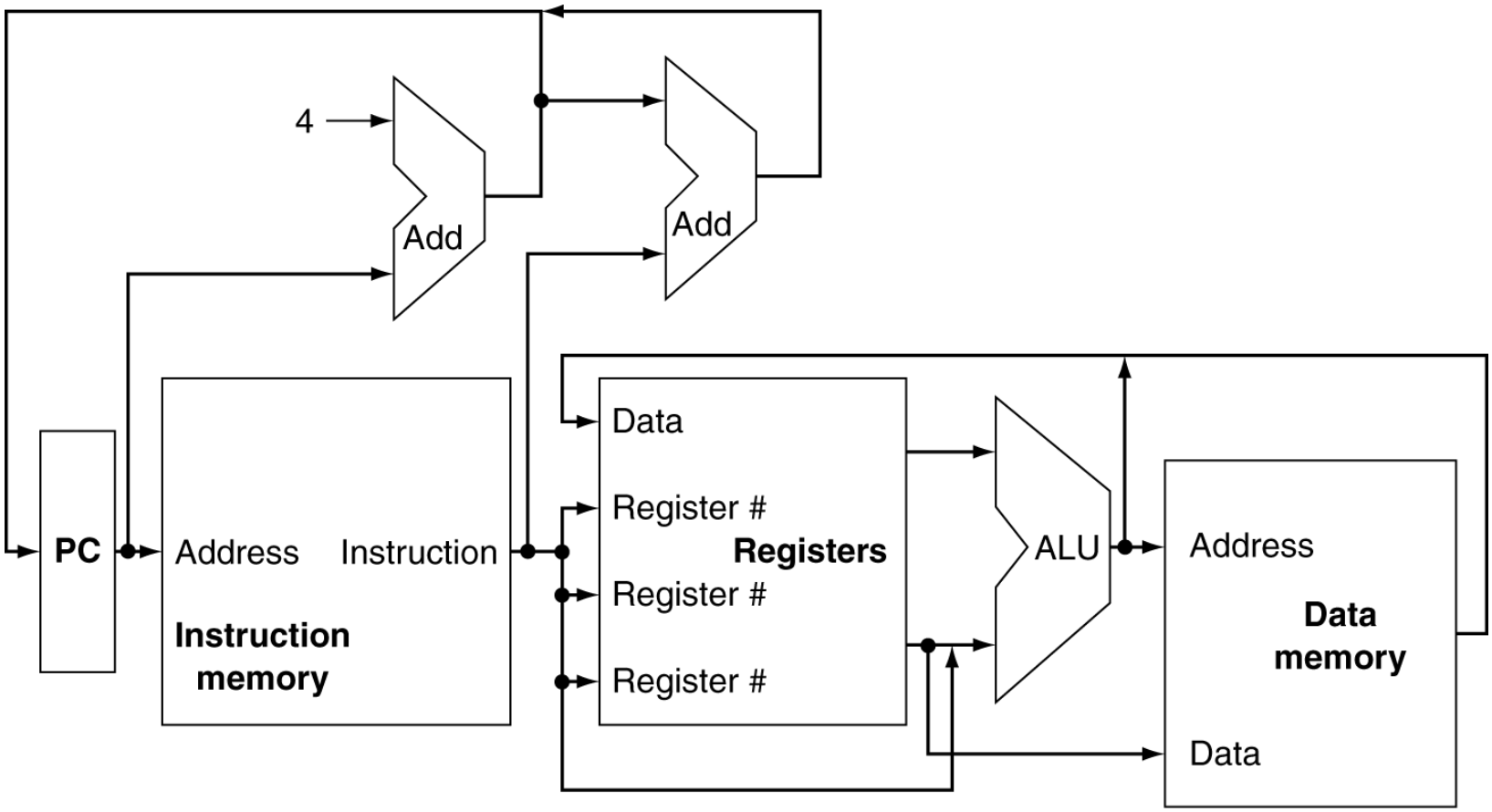
Introduction

- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware
- We will examine two MIPS implementations
  - A simplified version
  - A more realistic pipelined version
- Simple subset, shows most aspects
  - Memory reference: `lw`, `sw`
  - Arithmetic/logical: `add`, `sub`, `and`, `or`, `slt`
  - Control transfer: `beq`, `j`

Instruction Execution

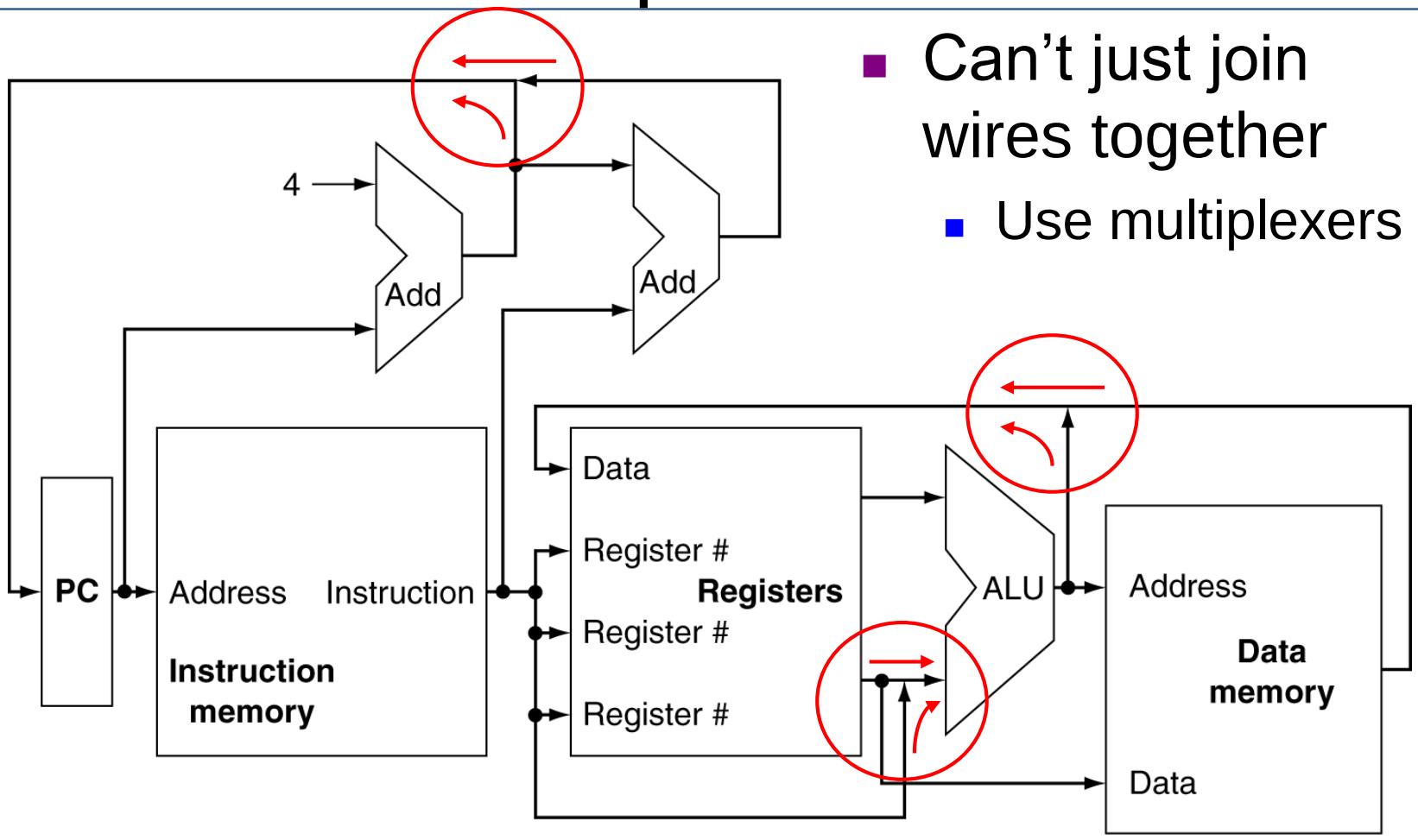
- PC → instruction memory, fetch instruction
- Register numbers → register file, read registers
- Depending on instruction class
  - Use ALU to calculate
    - Arithmetic result
    - Memory address for load/store
    - Branch target address
  - Access data memory for load/store
  - PC ← target address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

CPU Overview

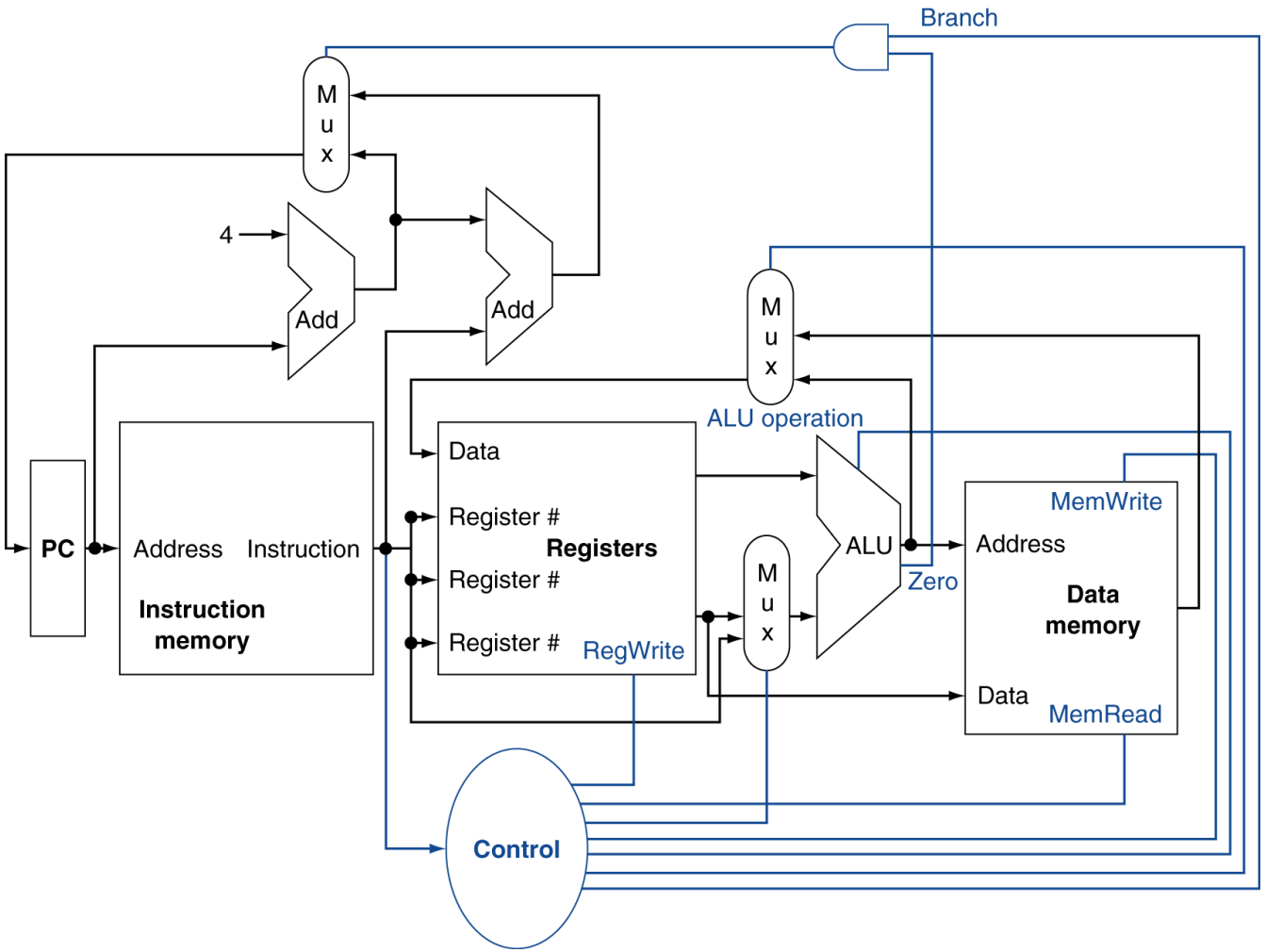


Multiplexers

- Can't just join wires together
  - Use multiplexers



Control

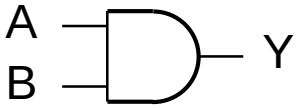

Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

Logic Design Basics

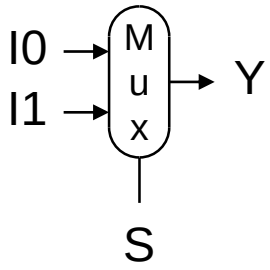
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information

Combinational Elements

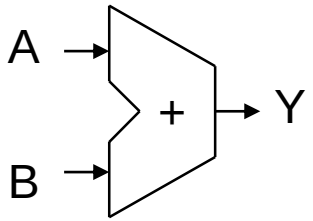
- AND-gate
  - Y = A & B



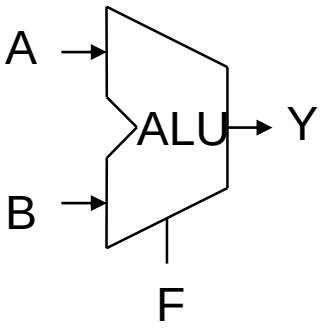
- Multiplexer
  - Y = S ? I1 : I0



- Adder
  - Y = A + B

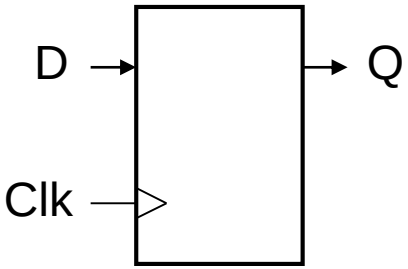


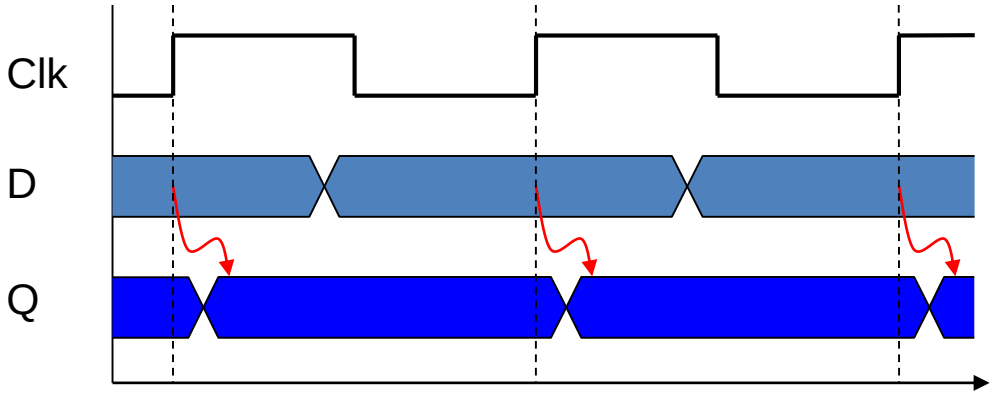
- Arithmetic/Logic Unit
  - Y = F(A, B)



Sequential Elements

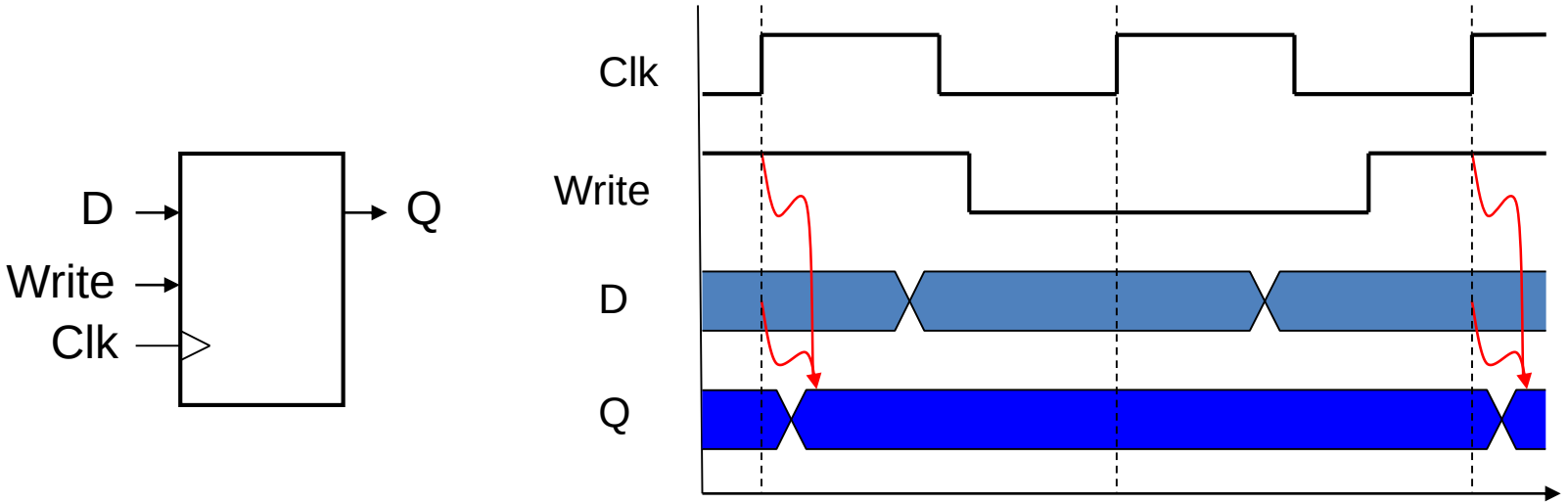
- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1





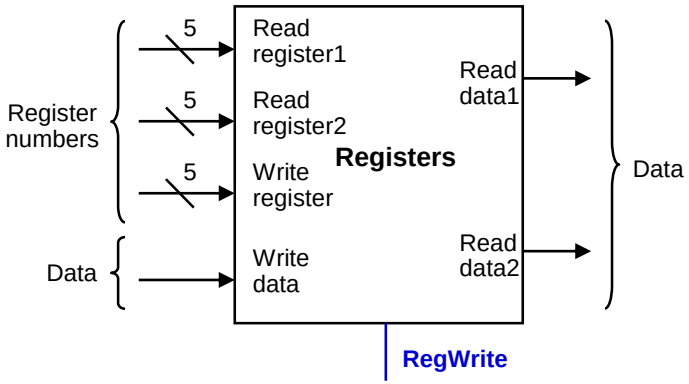
Sequential Elements

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



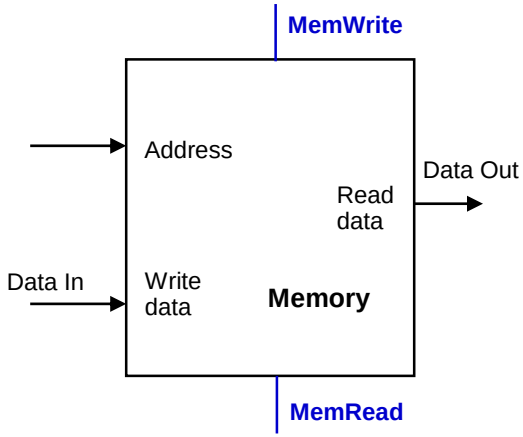
Storage element: register file

- Register File consists of 32 registers:
  - Two 32-bit output busses:
    - Read data1 and Read data2
  - One 32-bit input bus: Write data
  - Register 0 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 via Write data when RegWrite = 1
- Clock input (CLK)
  - The CLK input is a factor only for write operation (data changes only on falling clock edge)

Storage element: Memory

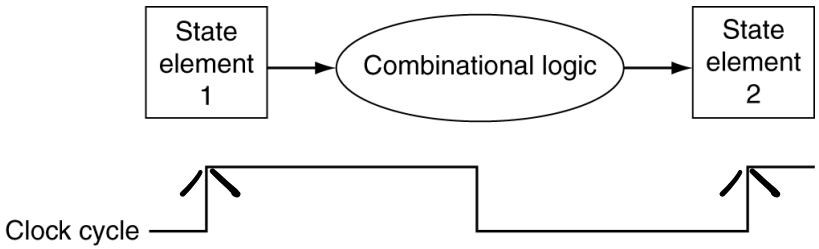


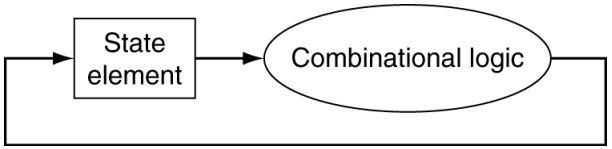
- Memory has two busses:
  - One output bus : Read data (Data Out)
  - One input bus : Write data (Data In)
- Address
  - Selects the word to put on Data Out when MemRead = 1
  - the word to be written via the Data In when MemWrite = 1
- Clock input (CLK)
  - The CLK input is a factor only for write operation
  - During read, behaves as combinational logic block
    - Valid address ➔ Data Out valid after “access time”
    - Minor simplification of reality

Clocking Methodology

- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period

다음 rising edge가 오기 전에 combinational logic의 계산이 완료되어야 함



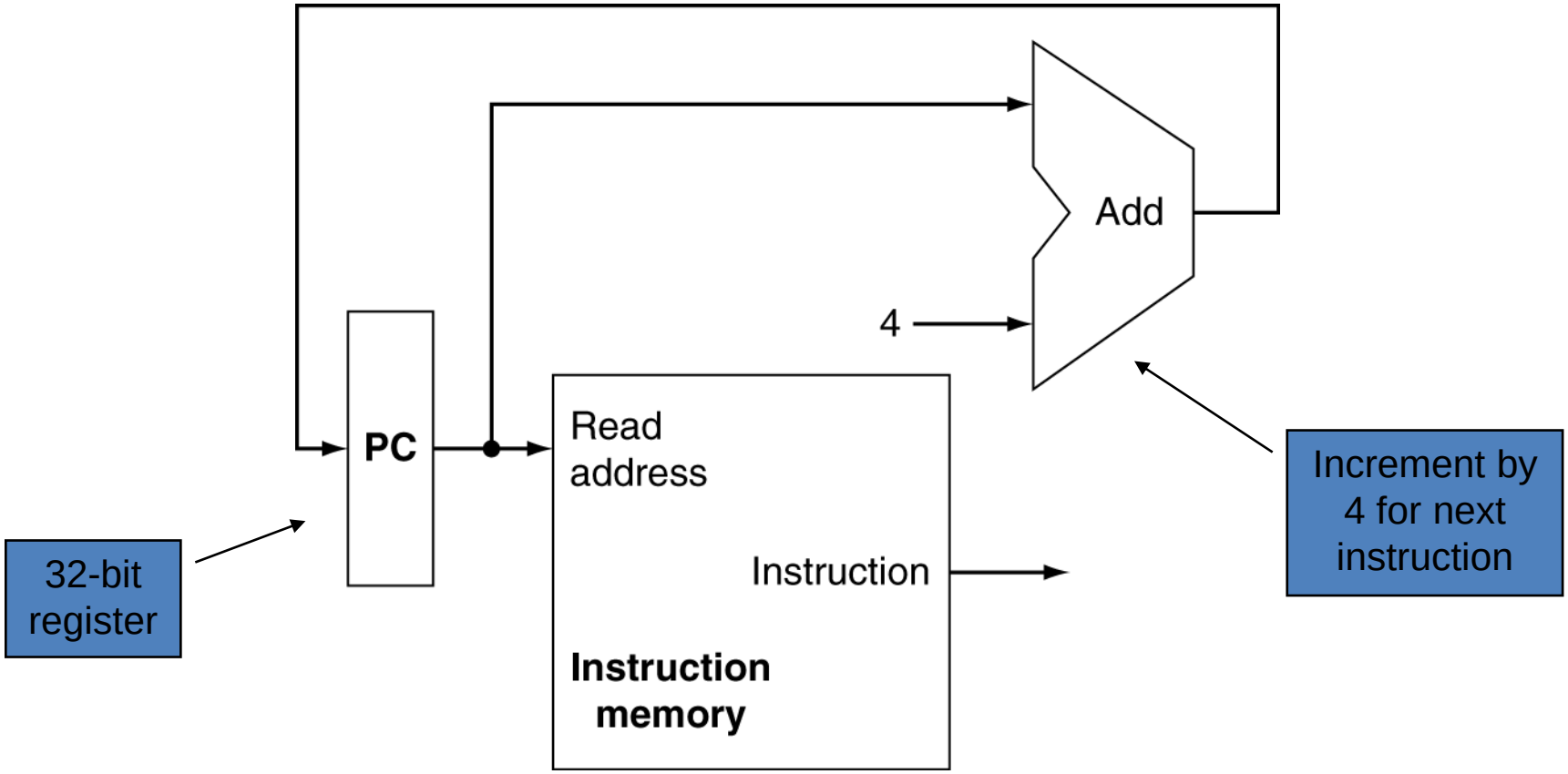


Building a Datapath

- Datapath
  - Elements that process data and addresses in the CPU
    - Registers, ALUs, mux's, memories, …
- We will build a MIPS datapath incrementally
  - Refining the overview design

pc + 4

Instruction Fetch



MIPS format review

- R-format
  - add rd, rs, rt
  - sub rd, rs, rt

- Machine-level configuration

Fields					
6bits	5bits	5bits	5bits	5bits	6bits
op	rs	rt	rd	shamt	funct
	1st source register	2nd source register	result register	shift amount	function code

MIPS format review (cont)

- I-format
  - lw rt, rs, imm
  - sw rt, rs, imm
  - beq rs, rt, imm
  - ori rt, rs, imm
- Reminders
  - Branch uses PC relative addressing: PC + 4 + (4 * imm)
- Machine-level configuration

Name	Fields					
Field size	6bits	5bits	5bits	5bits	5bits	6bits
I–format	op	rs	rt	address/immediate		

1st source register

2nd source register

immediate

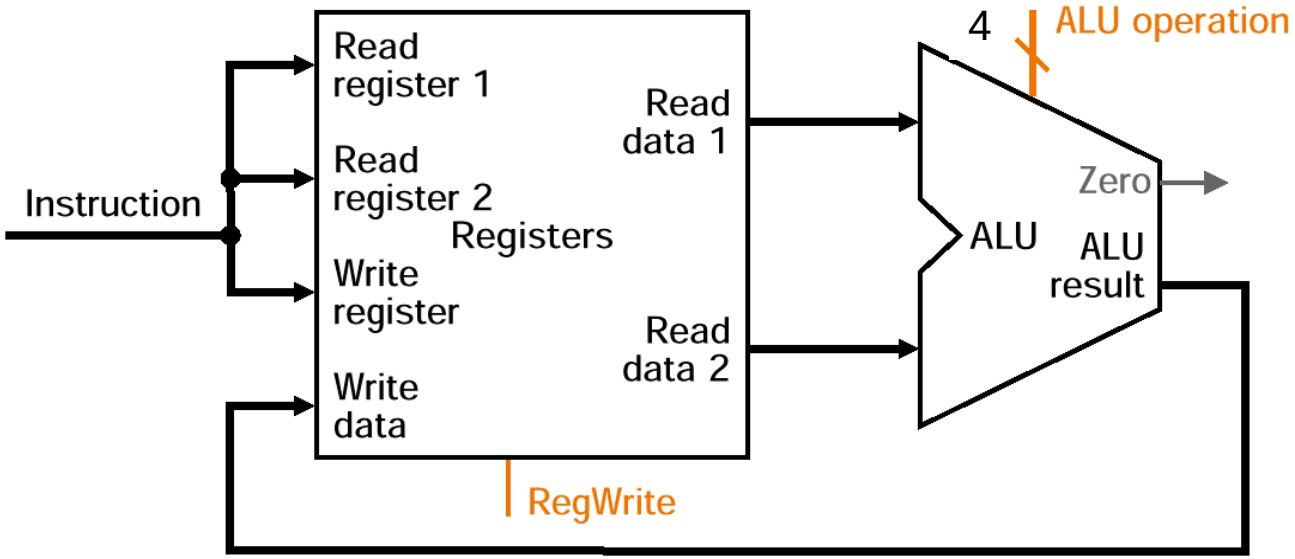
MIPS format review (cont)

- J-format
  - j target
- Reminders
  - Uses pseudodirect addressing (target * 4) to allow addressing 2^{28} bits directly
  - Uses top 4 bits from PC
- Machine-level configuration

Name	Fields					
Field size	6bits	5bits	5bits	5bits	5bits	6bits
J-format	op	target address				

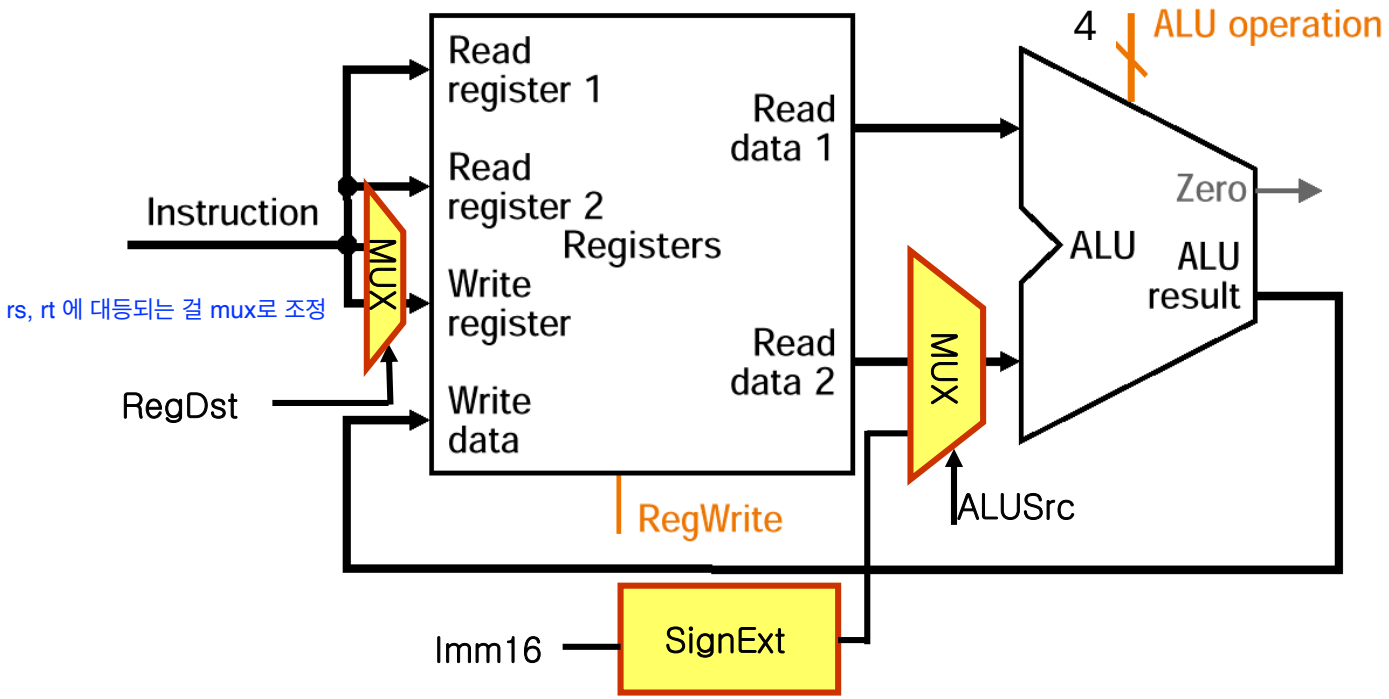
R-Format Instructions

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



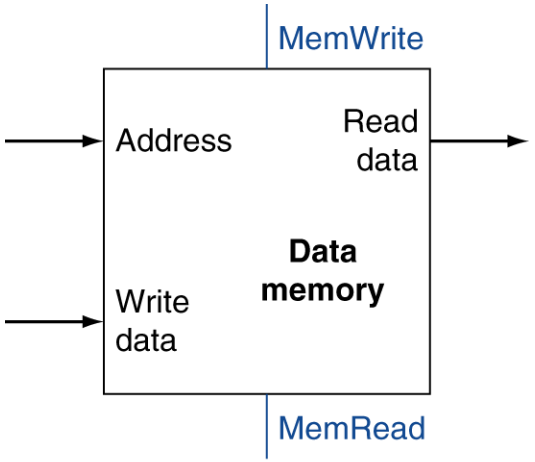
Immediate operations

- 2 Muxes and 1 SignExt are added
  - 1st mux: selects Rd if R-format by RegDst
  - 2nd mux: selects data from register if R-format by ALUsrc

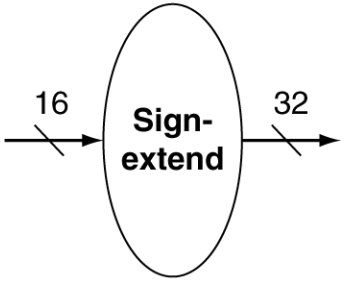


Load/Store Instructions

- Read register operands
- Calculate address using 16-bit offset
  - Use ALU, but sign-extend offset
- Load: Read memory and update register
- Store: Write register value to memory



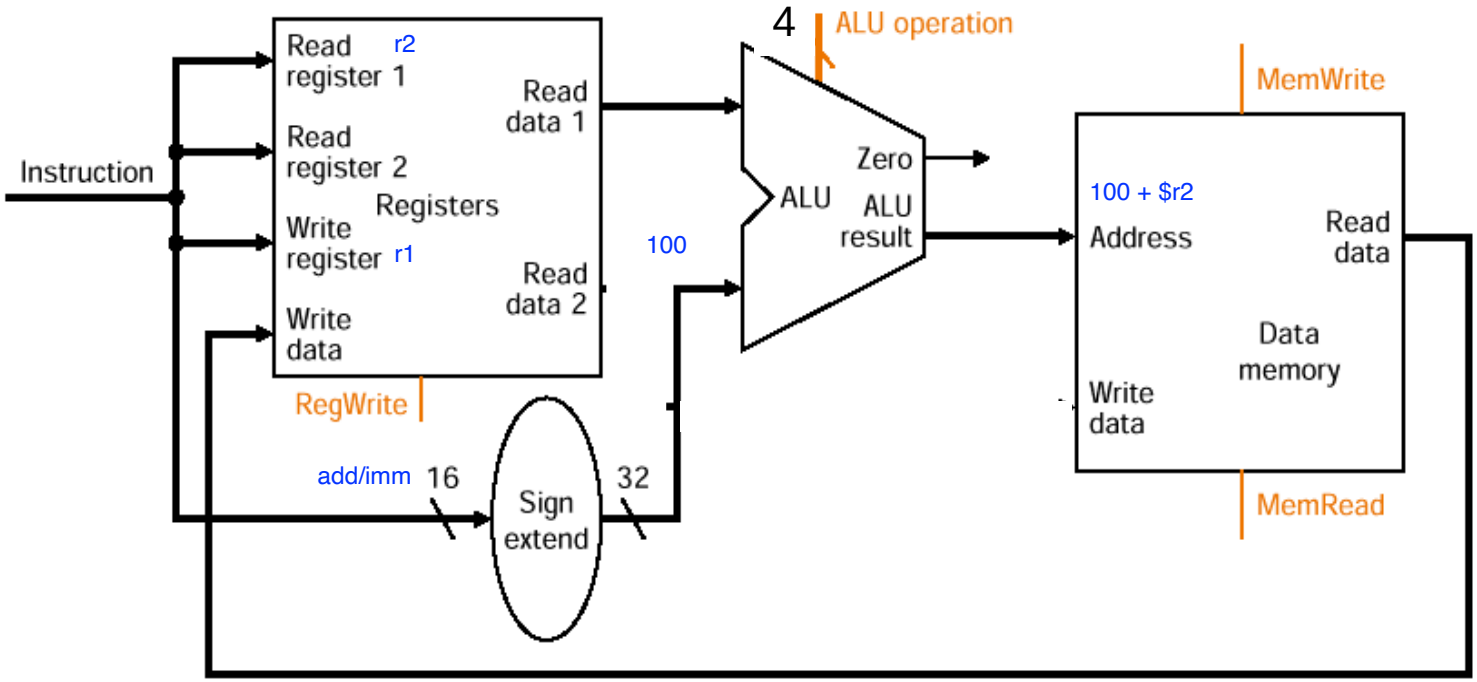
a. Data memory unit



b. Sign extension unit

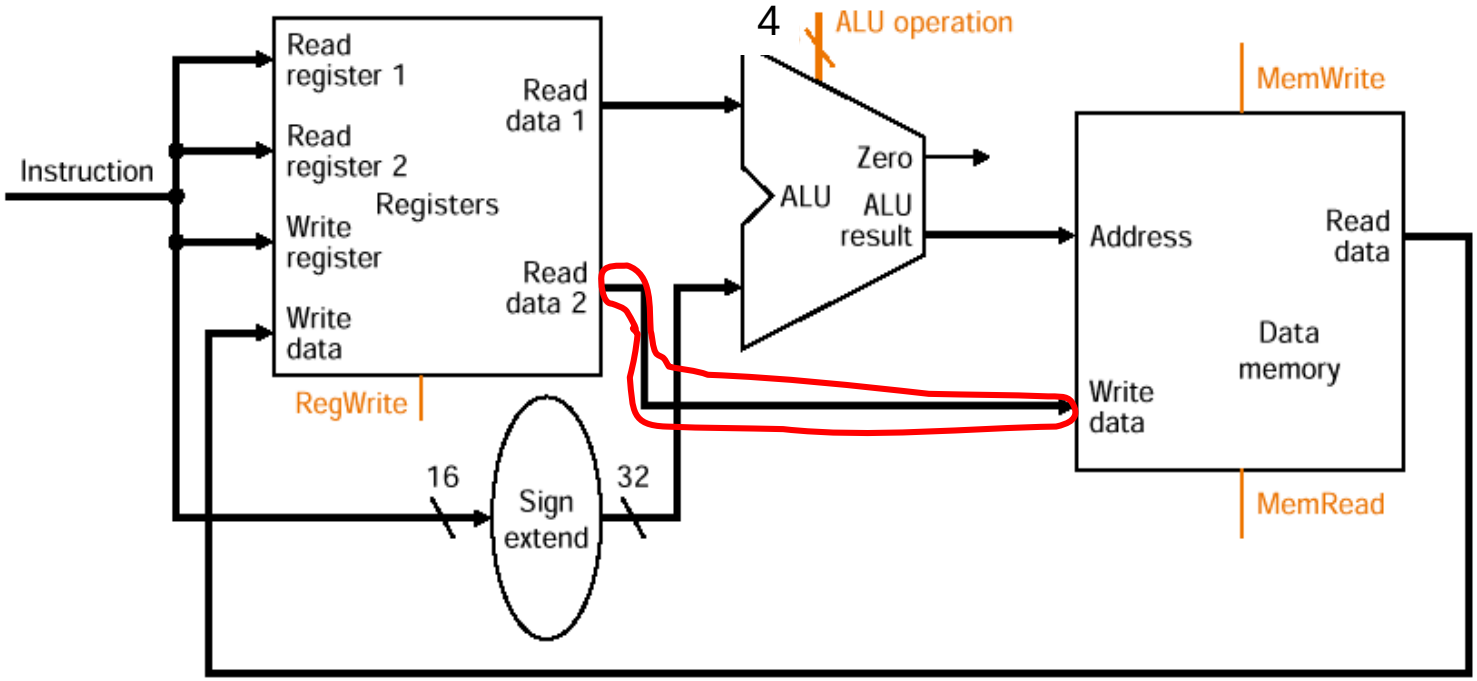
Load

- Sign extension logic is added
  - Offset can be either positive or negative
  - E.g. ld $r1, 100($r2), / ld $r1, -100($r2)



Store

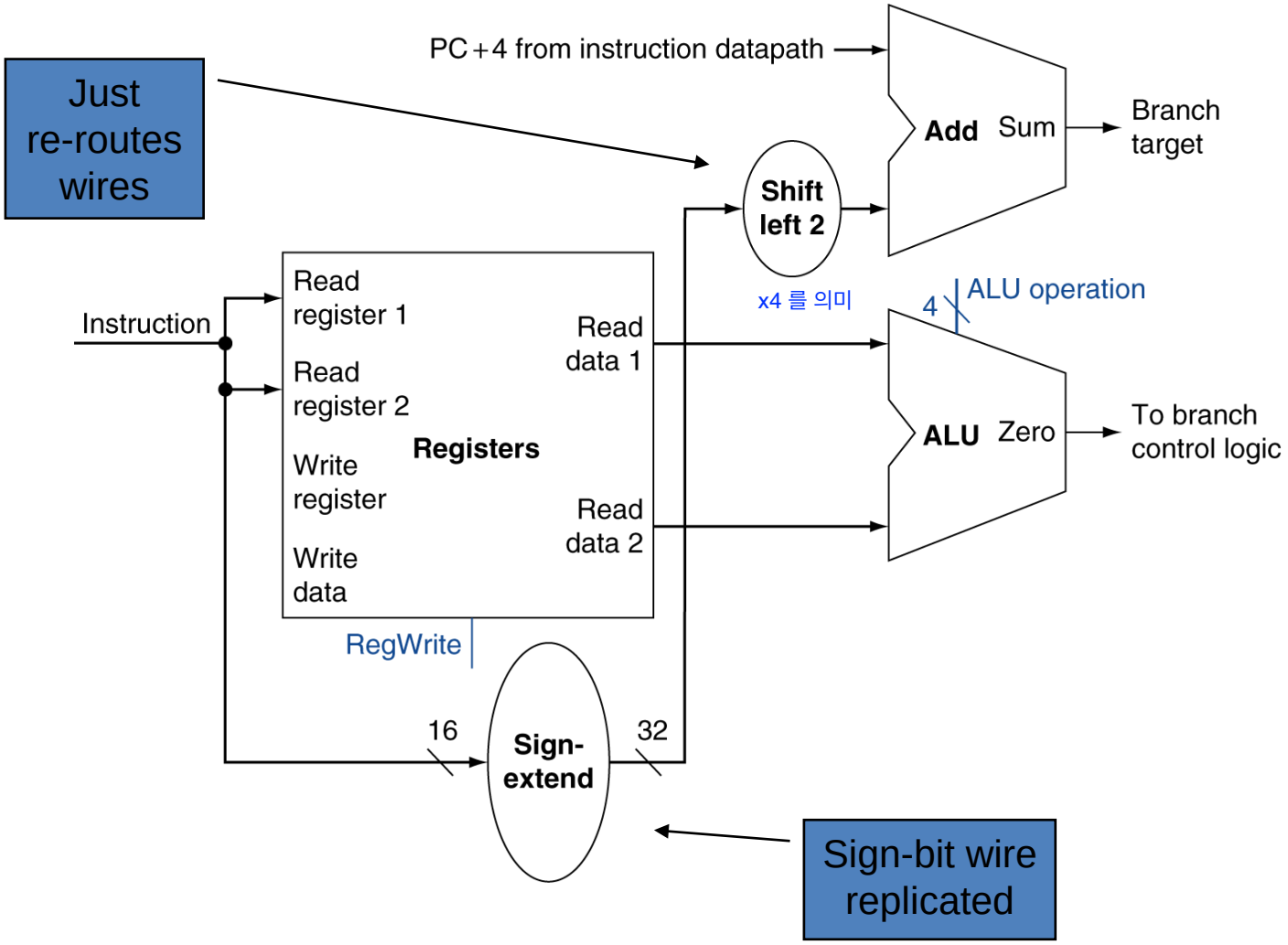
- A path from register to memory has been created



Branch Instructions

- Read register operands
- Compare operands
  - Use ALU, subtract and check Zero output
- Calculate target address
  - Sign-extend displacement
  - Shift left 2 places (word displacement)
  - Add to PC + 4
    - Already calculated by instruction fetch

Branch Instructions



The next address

- PC is byte-addressed into instruction memory
  - Sequential
    - PC[31:0] = PC[31:0] + 4
  - Branch operation
    - PC[31:0] = PC[31:0] + 4 + SignExt(imm) * 4
- Instruction addresses
  - PC is byte addressed, but instructions are 4 bytes long
  - Therefore 2 LSBs of the 32 bit PC are always 0
  - No reason to have hardware keep the 2 LSBs

so ignore bottom 2 bit
use word address calculation

➔ Simplify hardware by using 30 bit PC

- Sequential

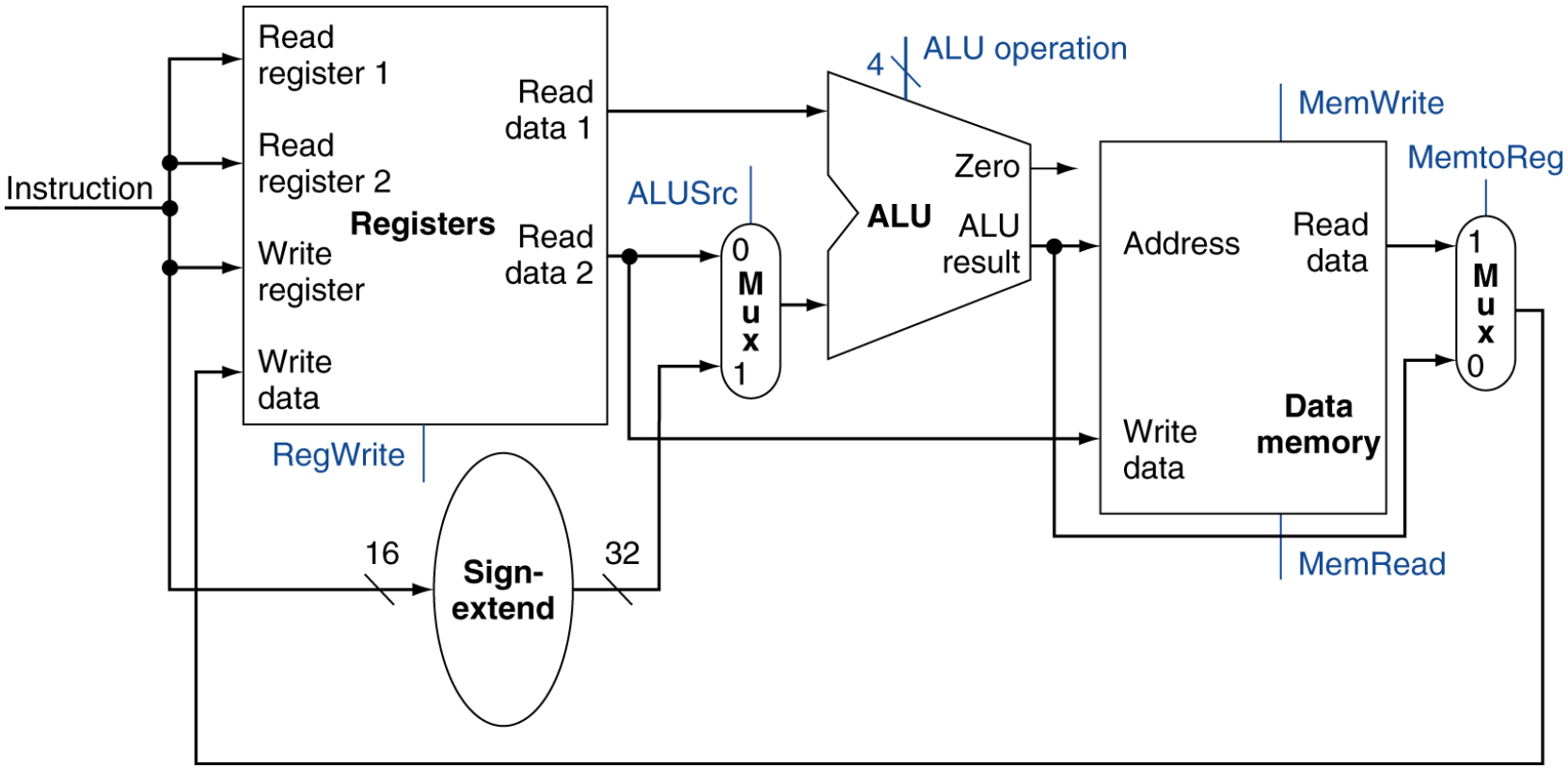
4 >> 2 (ignore bottom 2 bits)

  - PC[31:2] = PC[31:2] + 1
- Branch operation
  - PC[31:2] = PC[31:2] + 1 + SignExt(imm)

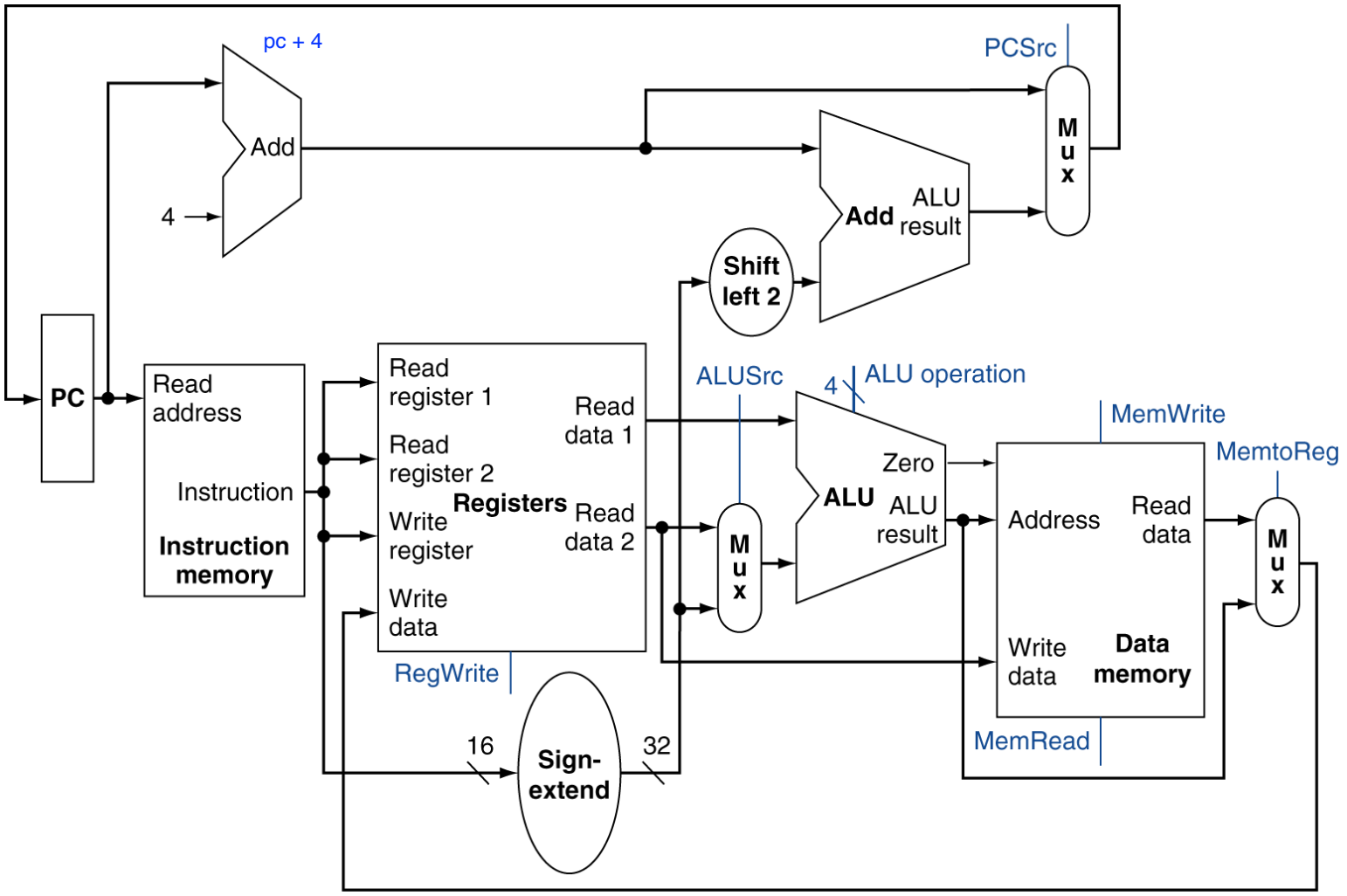
Composing the Elements

- First-cut data path 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 funct field

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set-on-less-than
1100	NOR

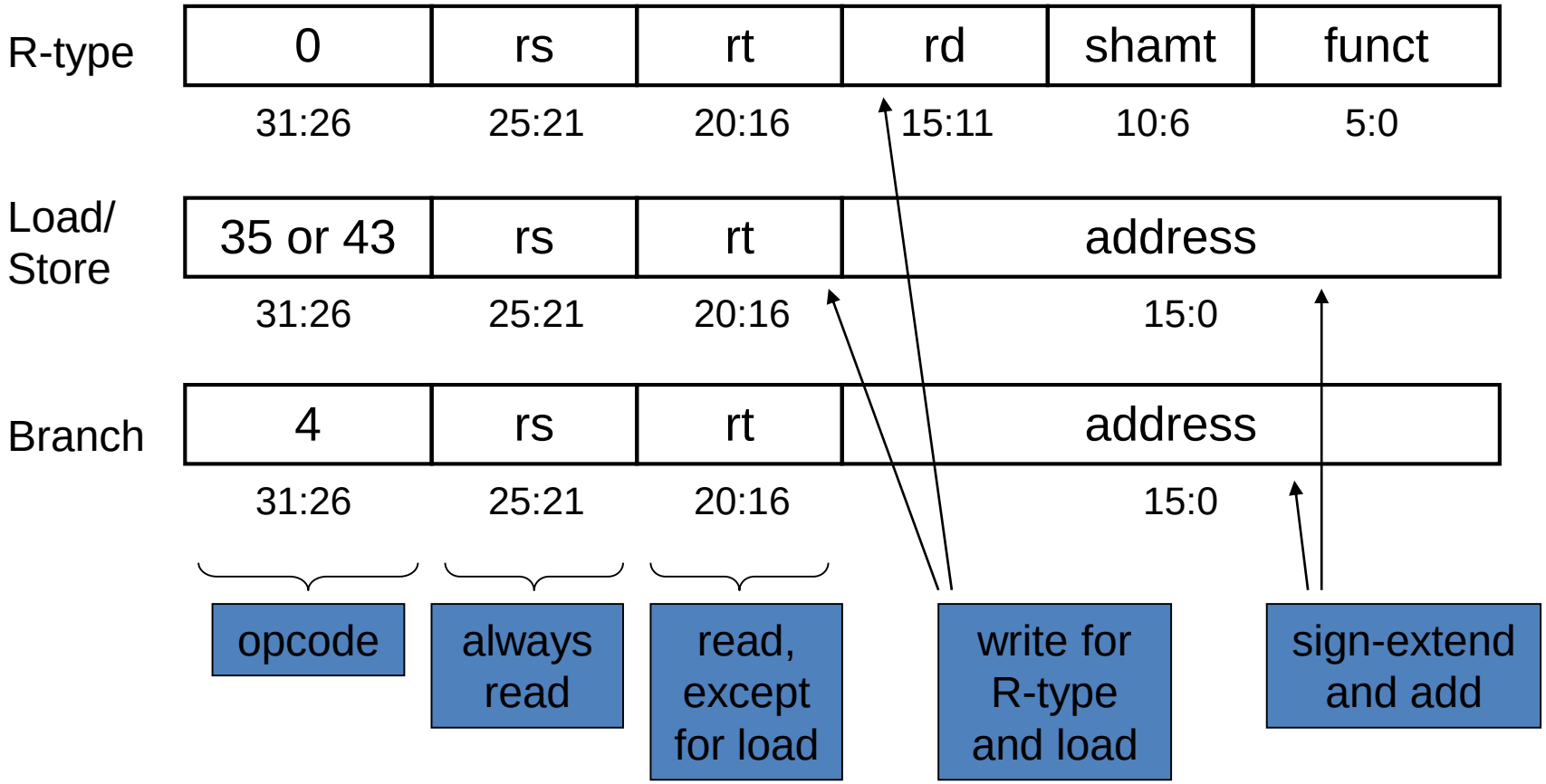
ALU Control

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

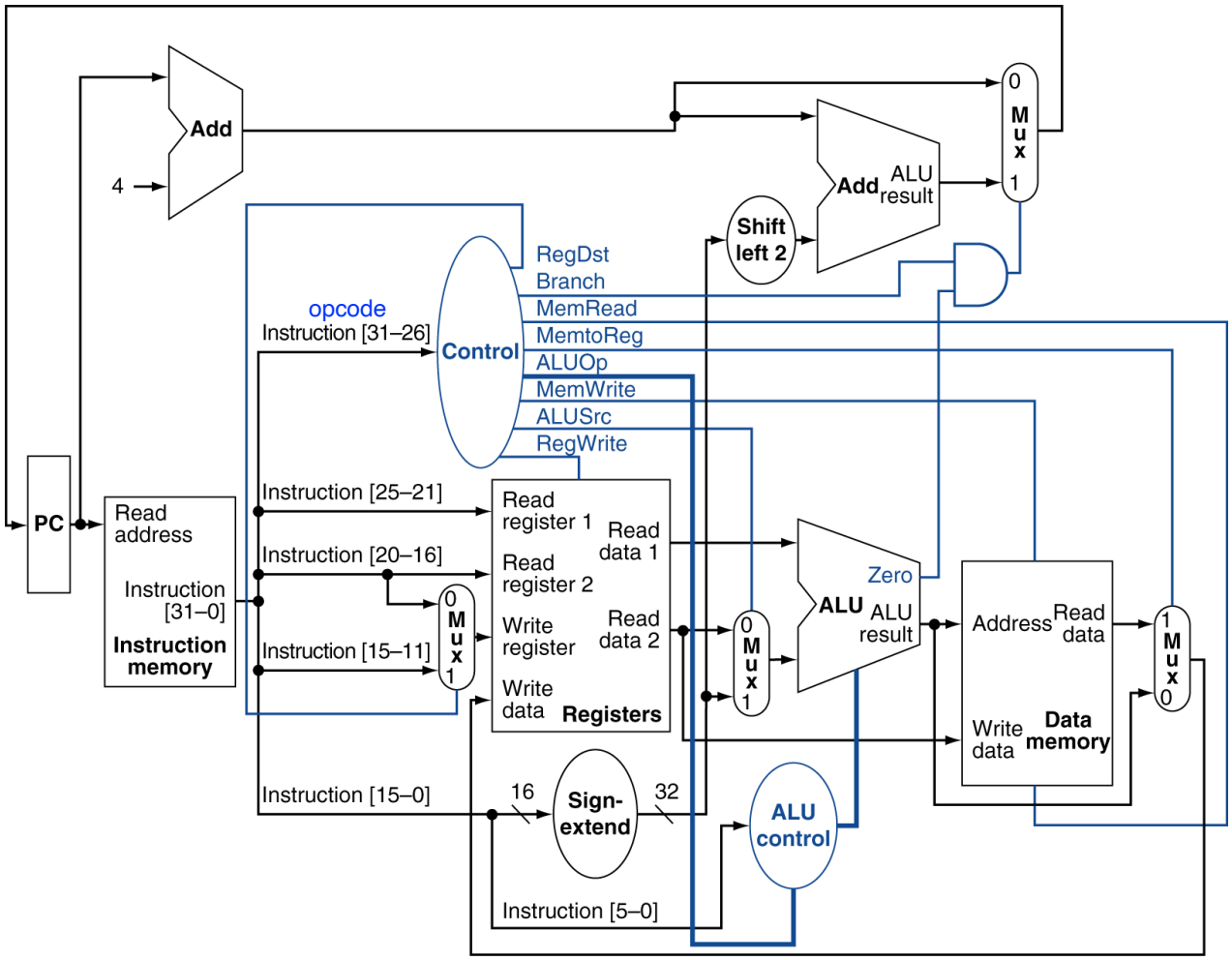
opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

The Main Control Unit

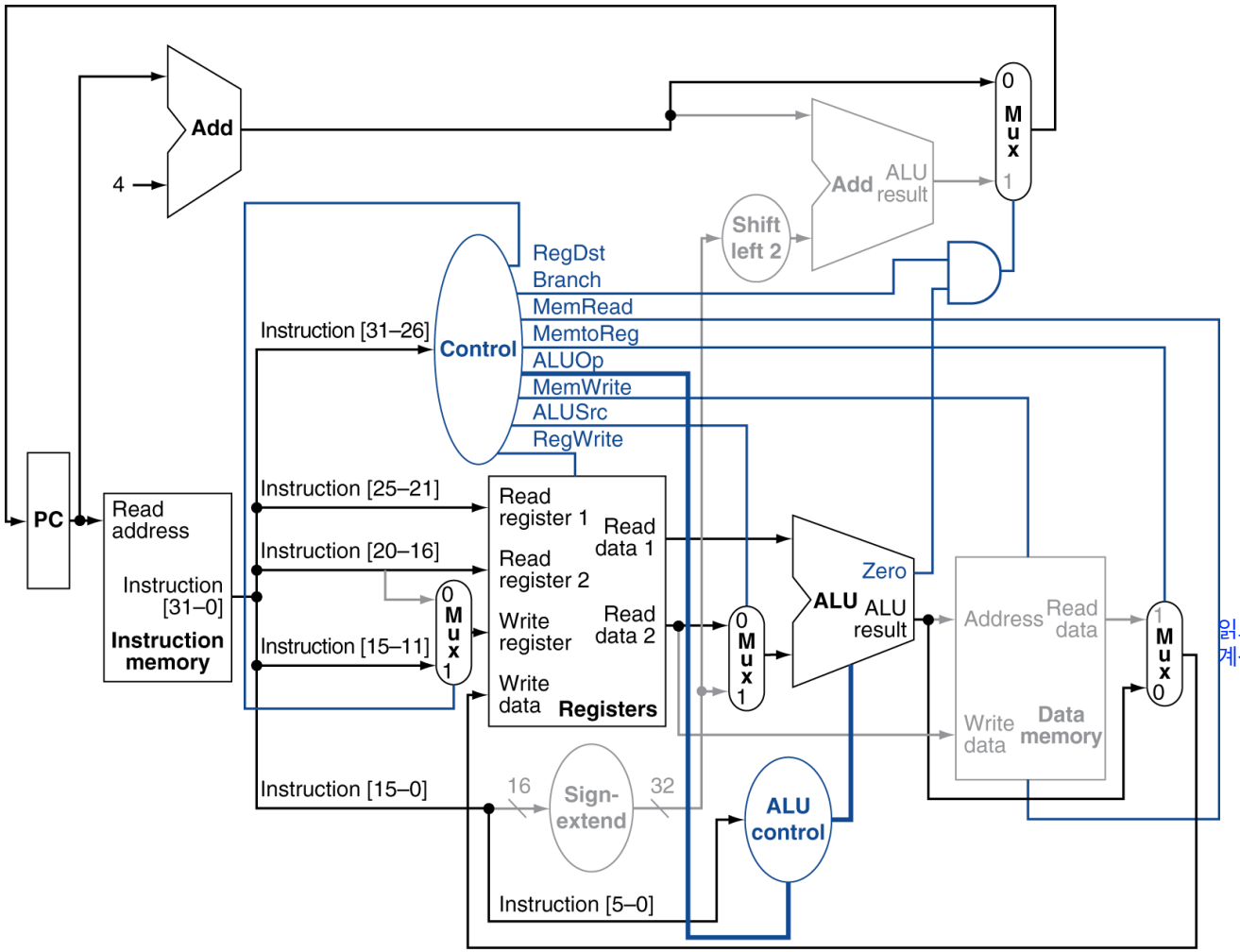
- Control signals derived from instruction



Datapath With Control



R-Type Instruction



읽고있는 데이터 아님
계산한 데이터 중 하나 선택

R-format instruction control

- Control signal summary

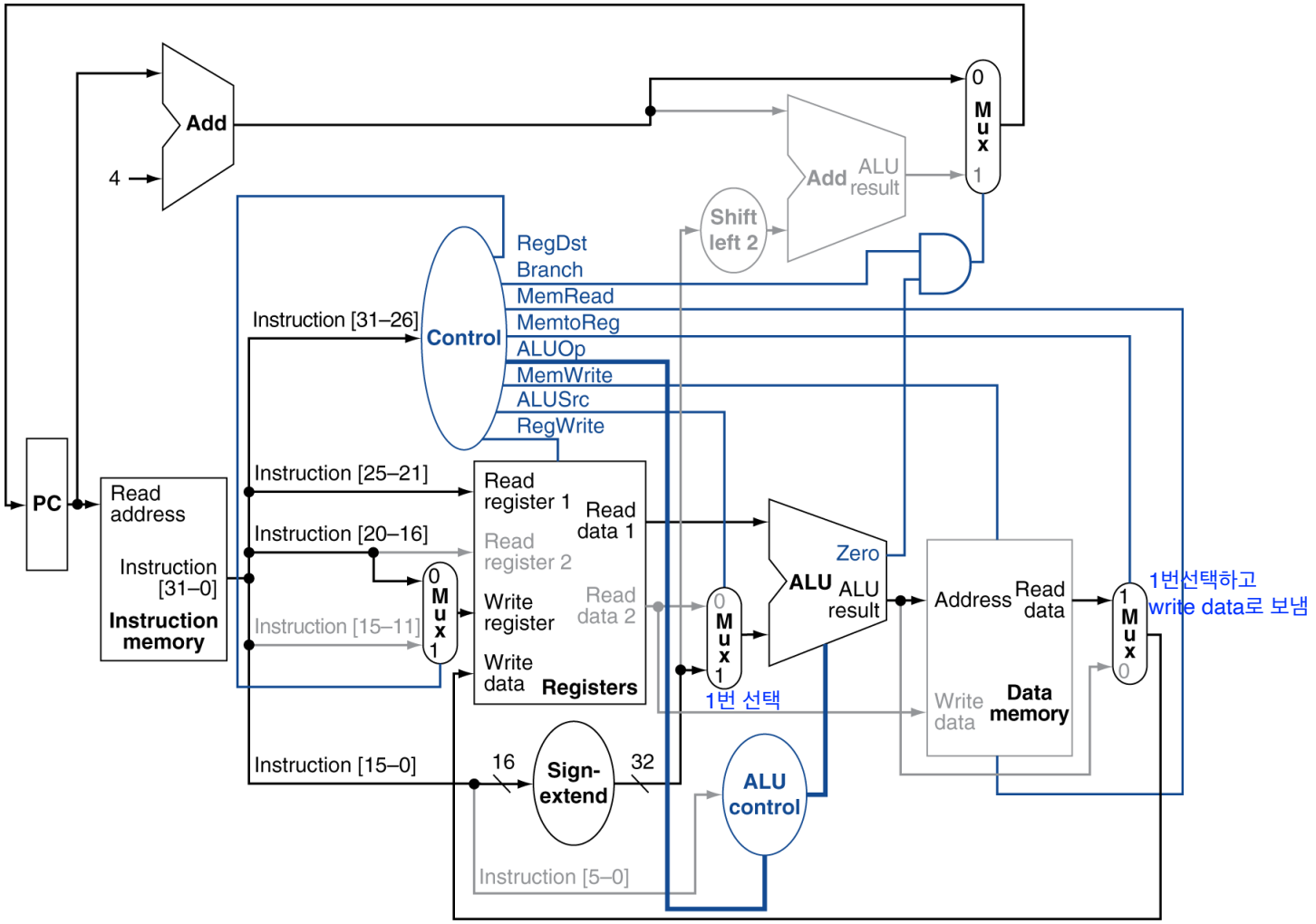
Signal	Value	Description
RegDst	1	to select Rd
RegWrite	1	to enable writing Rd
ALUSrc	0	to select Rt value from register file
ALUOp	OP	to select an appropriate operation for ALU
MemWrite	0	to disable writing memory
MemRead	0	to disable reading memory
MemtoReg	0	to select ALU output to register
PCSrc	0	to select next PC

R-format instruction control (cont)

- ALUOp summary

ALU control input	Function
000	and
001	or
010	add
110	subtract
111	set-on-less than

Load Instruction



I-format Load Instruction Control

- Control signal summary

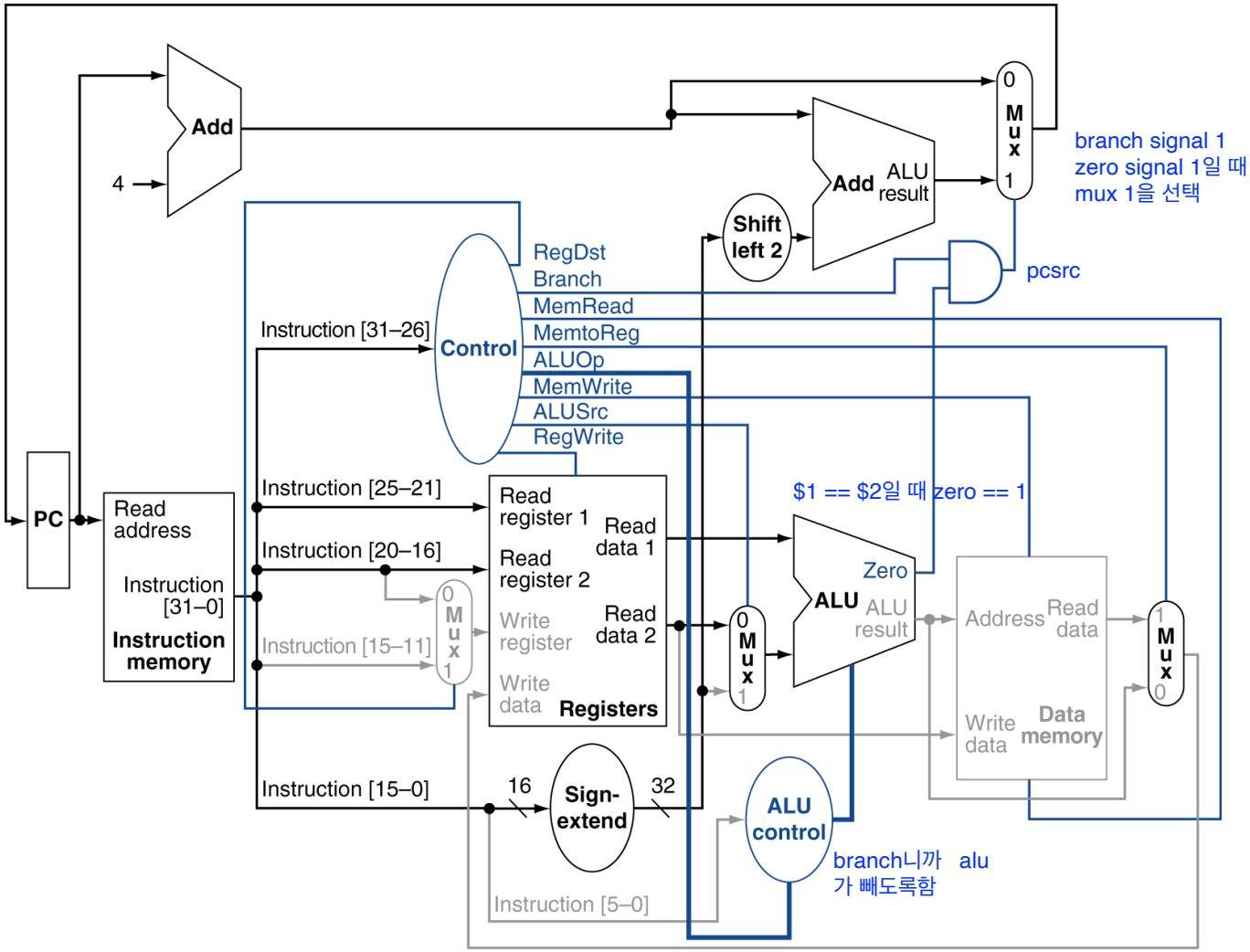
Signal	Value	Description
RegDst	0	to select Rt
RegWrite	1	to enable writing Rd
ALUSrc	1	to select immediate field value from instruction
ALUOp	OP	add
MemWrite	0	to disable writing memory
MemRead	1	to enable reading memory
MemtoReg	1	to select memory output to register
PCSrc	0	to select next PC

I-format Store Instruction Control

- Control signal summary

Signal	Value	Description
RegDst	X	Not used don't care
RegWrite	0	to disable writing a register
ALUSrc	1	to select immediate field value from instruction
ALUOp	OP	add
MemWrite	1	to enable writing memory
MemRead	0	to disable reading memory
MemtoReg	X	Not used
PCSrc	0	to select next PC

Branch-on-Equal Instruction

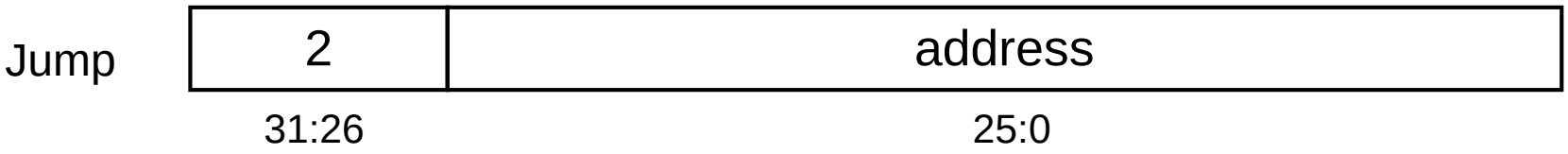


I-format Branch Instruction Control

- Control signal summary

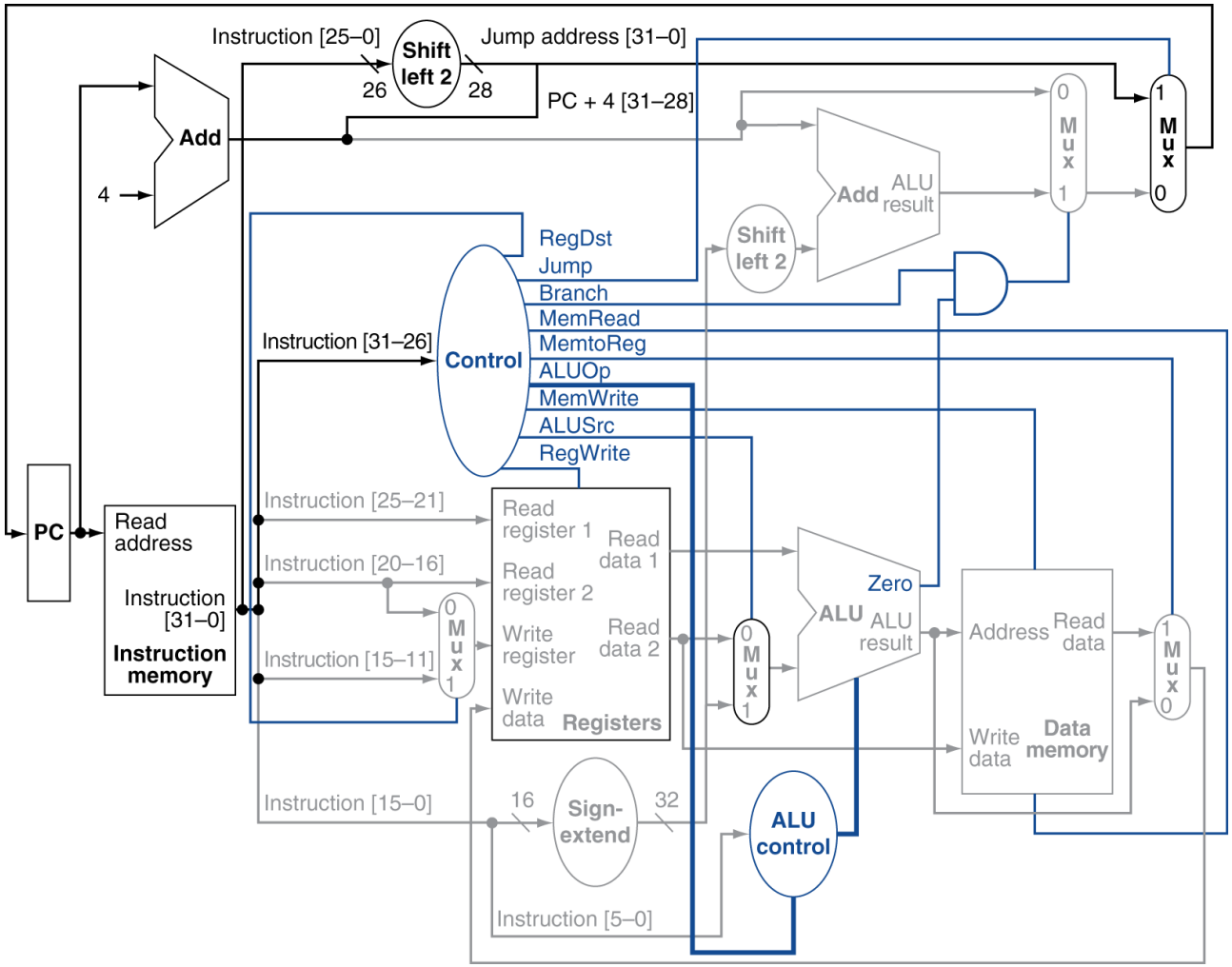
Signal	Value	Description
RegDst	X	Not used
RegWrite	0	to disable writing a register
ALUSrc	0	to select Rt value from register file
ALUOp	OP	sub
MemWrite	0	to disable writing memory
MemRead	0	to disable reading memory
MemtoReg	X	Not used
PCSrc	1	to select next PC

Implementing Jumps



- Jump uses word address
- Update PC with concatenation of
  - Top 4 bits of old PC
  - 26-bit jump address
- Need an extra control signal decoded from opcode

Datapath With Jumps Added



Unconditional Jump instruction control

- Control summary

Signal	Value	Description
RegDst	X	Not used
RegWrite	0	to disable writing a register
ALUSrc	X	Not used
ALUOp	X	Not used
MemWrite	0	to disable writing memory
MemRead	0	to disable reading memory
MemtoReg	X	Not used
PCSrc	X	Not used
Jump	1	to select next PC

Control Signals Summary

Signal	R-fmt	I-fmt(lw)	I-fmt(sw)	I-fmt (beq)	I-fmt(j)
RegDst	1	0	X	X	X
RegWrite	1	1	0	0	0
ALUSrc	0	1	1	1	X
ALUOp	OP	add	add	sub	X
MemWrite	0	0	1	0	0
MemRead	0	1	0	0	0
MemtoReg	0	1	X	X	X
PCSrc	0	0	0	Zero	X
ExtOp	X	1	1	X	X
Branch	0	0	0	1	0
Jump	0	0	0	0	1

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