ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

KIẾN TRÚC MÁY TÍNH

Computer Architecture

Course ID: IT3030, IT3034, IT3283

Version: CA.RISCV.2025

© Nguyễn Kim Khánh

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Hệ thống máy tính

Chương 3. Kiến trúc tập lệnh

Chương 4. Số học máy tính

Chương 5. Bộ xử lý

Chương 6. Bộ nhớ máy tính

Chương 7. Hệ thống vào-ra

Chương 8. Các kiến trúc song song

Chương 5
BỘ XỬ LÝ

Nội dung

5.1. Giới thiệu chung

5.2. Bộ xử lý đơn chu kỳ (Single-Cycle Processor)

5.3. Bộ xử lý đa chu kỳ (Multicycle Processor)

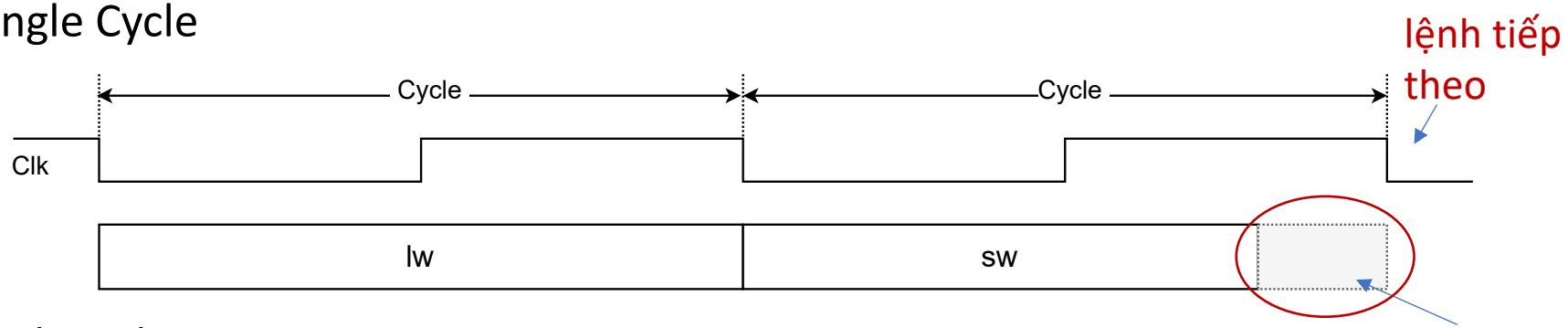
5.4. Bộ xử lý đường ống (Pipelined Processor)

5.1. Giới thiệu chung

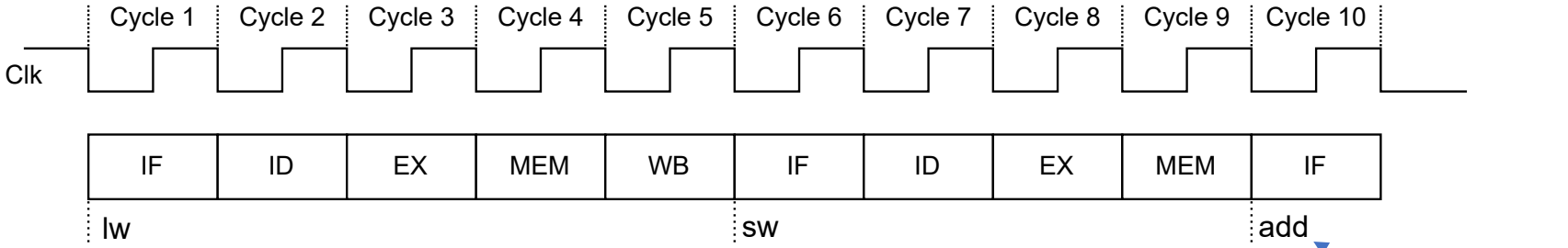
- Vi kiến trúc (microarchitecture): triển khai kiến trúc tập lệnh bằng phần cứng
- Cùng một kiến trúc tập lệnh có nhiều cách triển khai:
  - Đơn chu kỳ (Single-cycle): mỗi lệnh thực hiện trong một chu kỳ
  - Đa chu kỳ (Multicycle): mỗi lệnh được chia thành chuỗi các bước ngắn
  - Đường ống (Pipelined): mỗi lệnh được chia thành chuỗi các bước ngắn và nhiều lệnh cùng được thực hiện gối lên nhau

Minh hoạ các cách triển khai thiết kế bộ xử lý

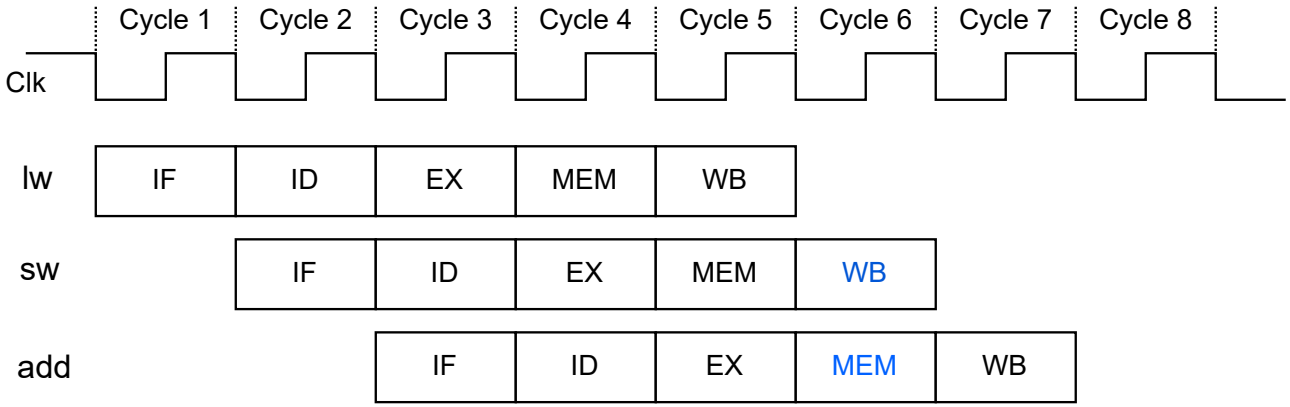
Single Cycle



Multicycle



Pipeline



Hiệu năng của bộ xử lý

- Hiệu năng phụ thuộc vào:
  - Số lệnh
    - Được xác định bởi kiến trúc tập lệnh
  - Số chu kỳ trên một lệnh (CPI) và Thời gian một chu kỳ
    - Được xác định bởi phần cứng của CPU

Thiết kế bộ xử lý RISC-V 32-bit đơn giản

- Triển khai với một số lệnh cơ bản của RISC-V:
  - Các lệnh số học/logic kiểu R:
    - `add, sub, and, or`
  - Các lệnh tham chiếu bộ nhớ:
    - `lw, sw`
  - Lệnh rẽ nhánh:
    - `beq`

Tổng quan quá trình thực hiện các lệnh

- Hai bước đầu tiên với mỗi lệnh:
  - Đưa địa chỉ từ bộ đếm chương trình PC đến bộ nhớ lệnh, tìm nạp lệnh từ bộ nhớ này
  - Sử dụng các số hiệu thanh ghi trong lệnh để chọn và đọc một hoặc hai thanh ghi:
    - Lệnh lw: đọc 1 thanh ghi (thanh ghi cơ sở)
    - Các lệnh khác: đọc 2 thanh ghi

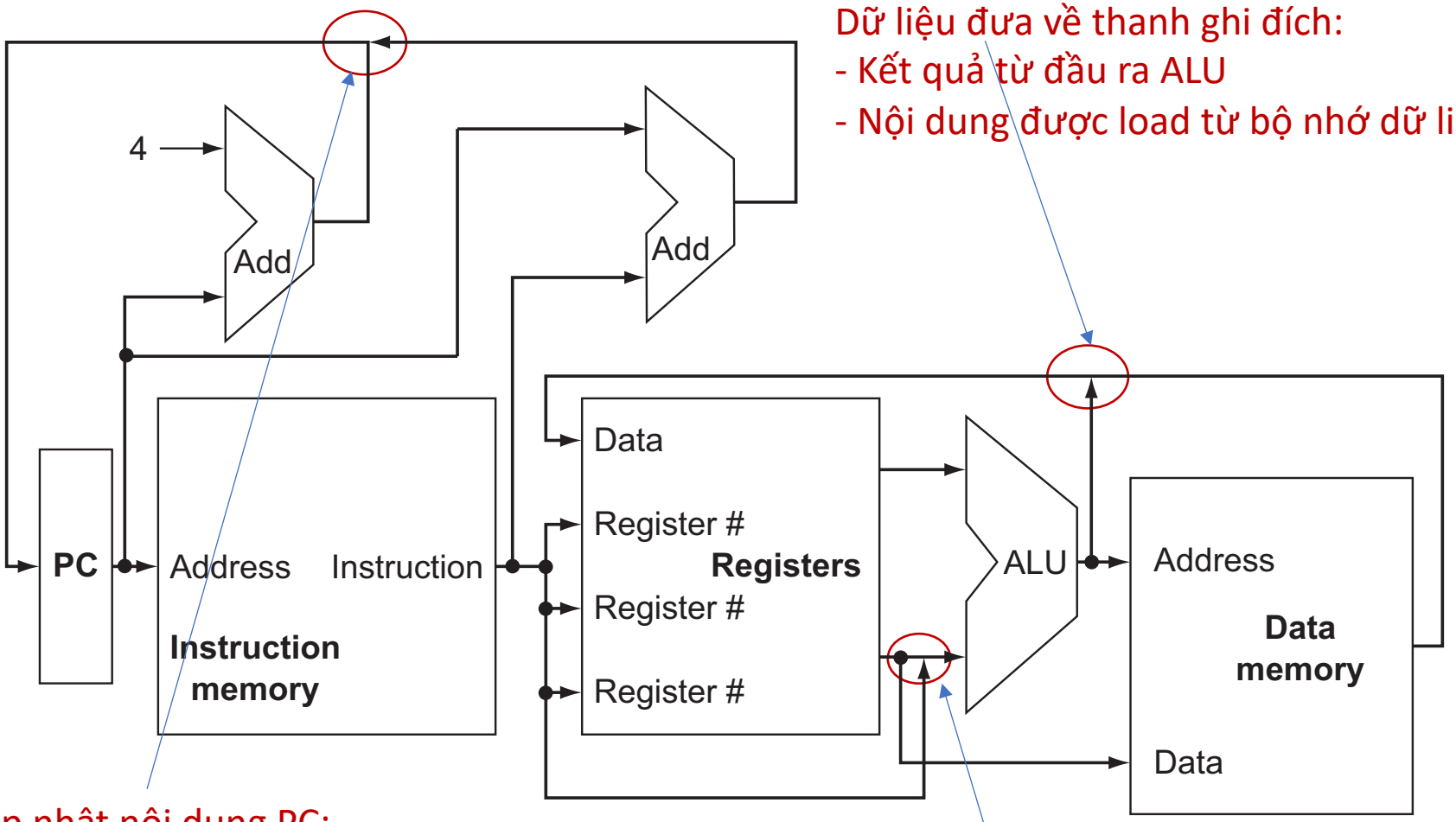
Tổng quan quá trình thực hiện các lệnh (tiếp)

- Các bước tiếp theo tùy thuộc vào loại lệnh:
  - Sử dụng ALU hoặc bộ cộng Add để:
    - Tính kết quả phép toán với các lệnh số học/logic
    - So sánh các toán hạng với các lệnh branch
    - Tính địa chỉ đích với các lệnh branch
    - Tính địa chỉ ngăn nhớ dữ liệu với lệnh load/store
  - Truy cập bộ nhớ dữ liệu với lệnh load/store
    - Lệnh lw: đọc dữ liệu từ bộ nhớ
    - Lệnh sw: ghi dữ liệu ra bộ nhớ
  - Ghi dữ liệu đến thanh ghi đích:
    - Các lệnh số học/logic: kết quả phép toán
    - Lệnh lw: dữ liệu được đọc từ bộ nhớ dữ liệu

Tổng quan quá trình thực hiện các lệnh (tiếp)

- Thay đổi nội dung bộ đếm chương trình PC:
  - Với các lệnh rẽ nhánh (branch), tùy thuộc vào kết quả so sánh:
    - Điều kiện thỏa mãn: PC ← địa chỉ đích BTA (địa chỉ của lệnh cần rẽ tới)
    - Điều kiện không thỏa mãn: PC ← PC + 4 (địa chỉ của lệnh kế tiếp)
  - Với các lệnh còn lại (tuần tự)
    - PC ← PC + 4 (địa chỉ của lệnh kế tiếp)

Sơ đồ khái quát của bộ xử lý RISC-V

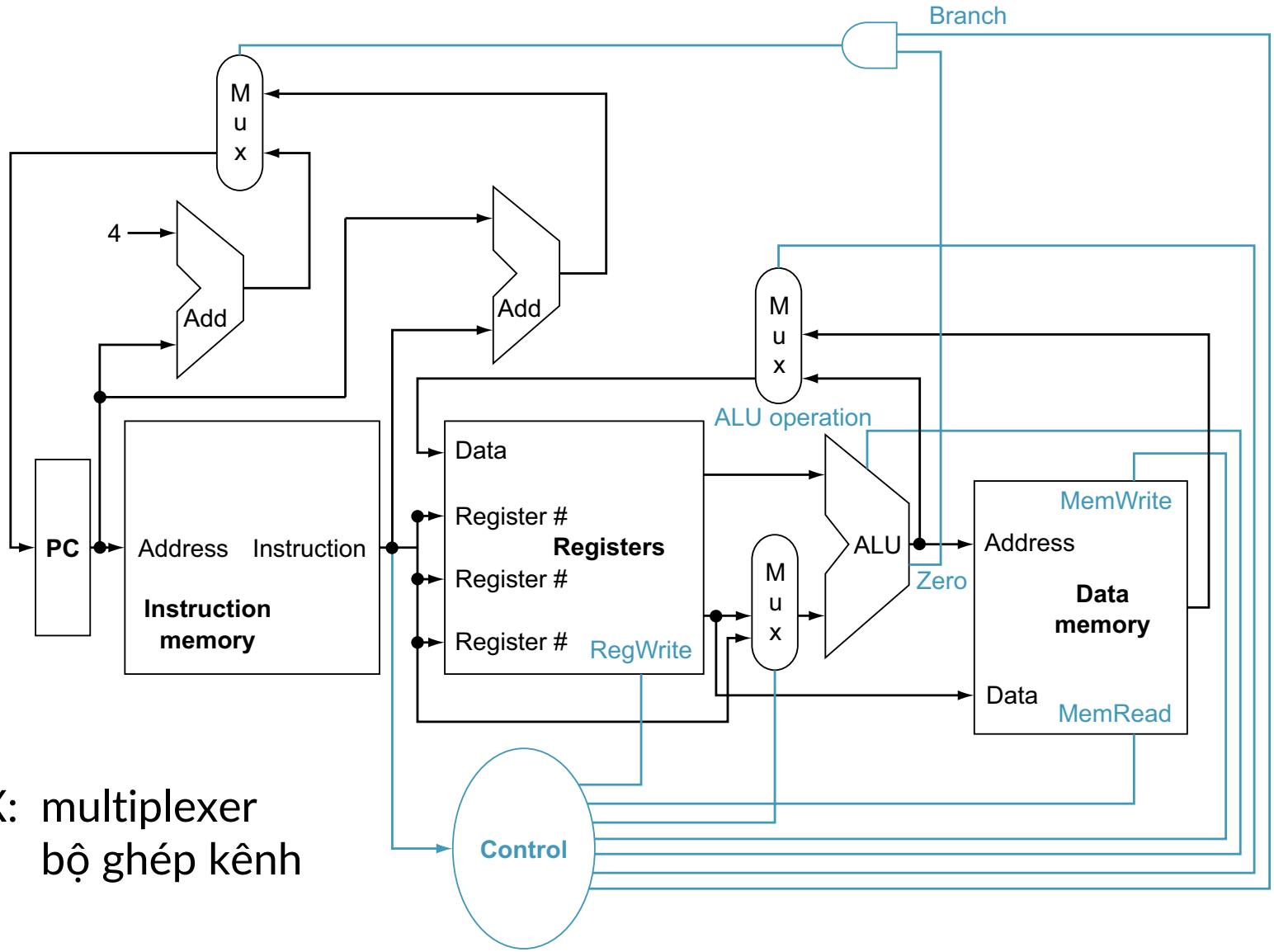


Dữ liệu đưa về thanh ghi đích:
- Kết quả từ đầu ra ALU
- Nội dung được load từ bộ nhớ dữ liệu

Cập nhật nội dung PC:
- PC=PC+4
- PC=địa chỉ đích (BTA)

Toán hạng thứ hai đưa đến ALU:
- Từ thanh ghi nguồn thứ hai (rs2)
- Hằng số imm trong lệnh

Bộ xử lý với các đường điều khiển chính



MUX: multiplexer
bộ ghép kênh

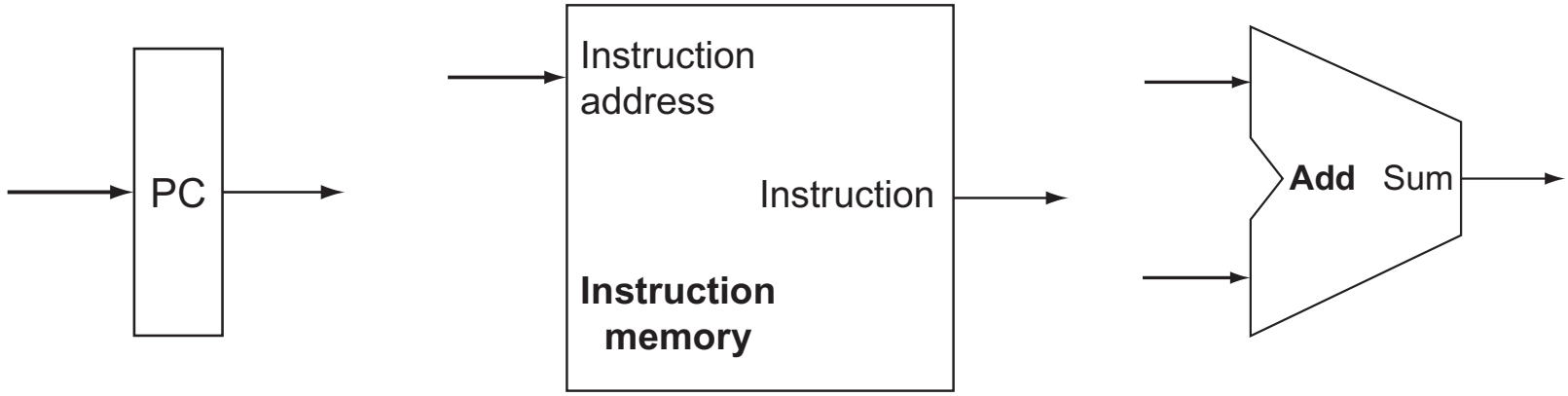
5.2. Bộ xử lý đơn chu kỳ

- Chia bộ xử lý thành hai phần:
  - Đường dẫn dữ liệu (Datapath)
  - Đơn vị điều khiển (Control Unit)

1. Thiết kế Datapath

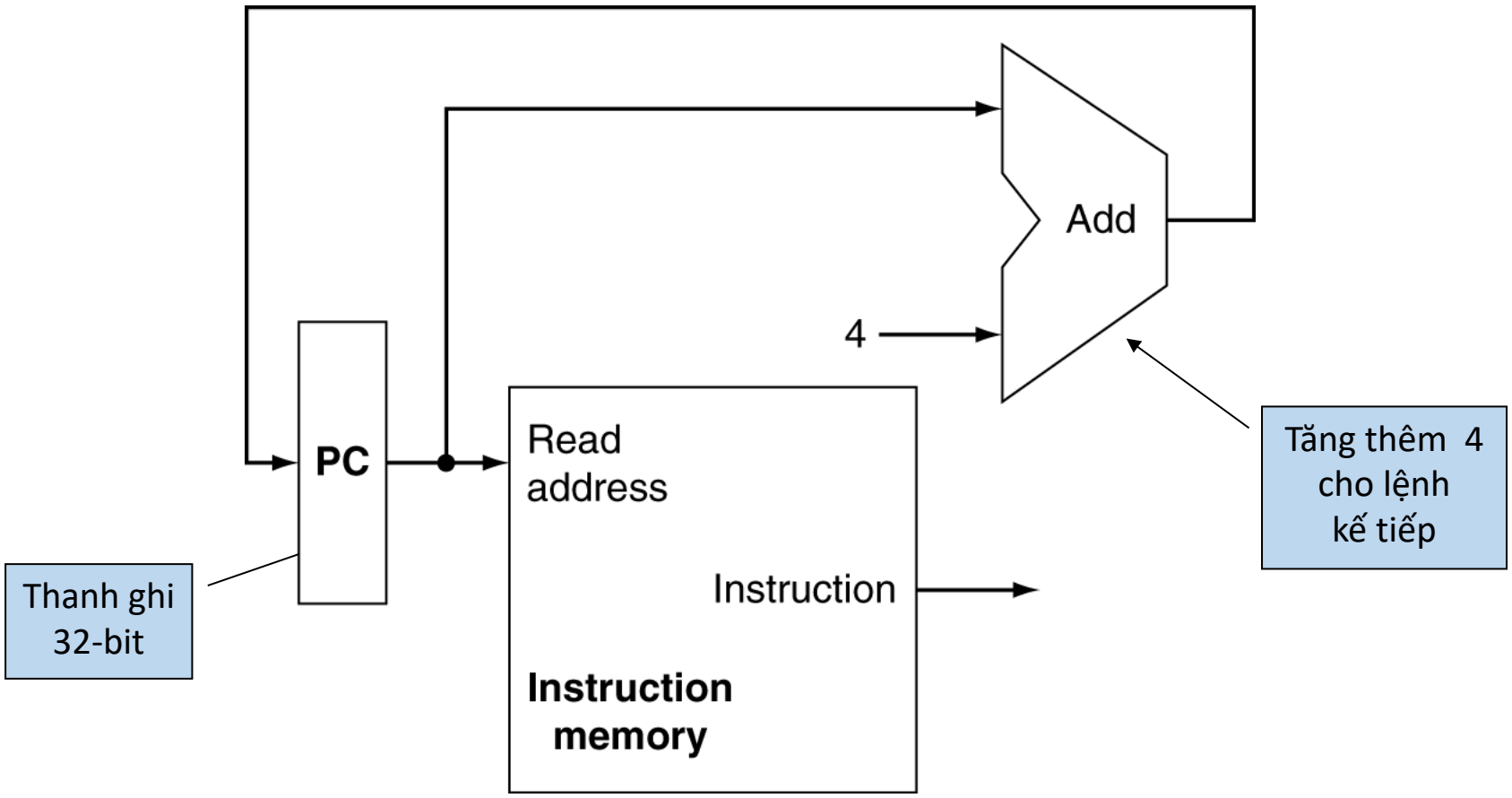
- Datapath: gồm các thành phần để xử lý dữ liệu và địa chỉ
  - Tập thanh ghi, ALUs, MUX's, bộ nhớ, …
- Sẽ xây dựng Datapath tăng dần

Các thành phần để thực hiện tìm nạp lệnh



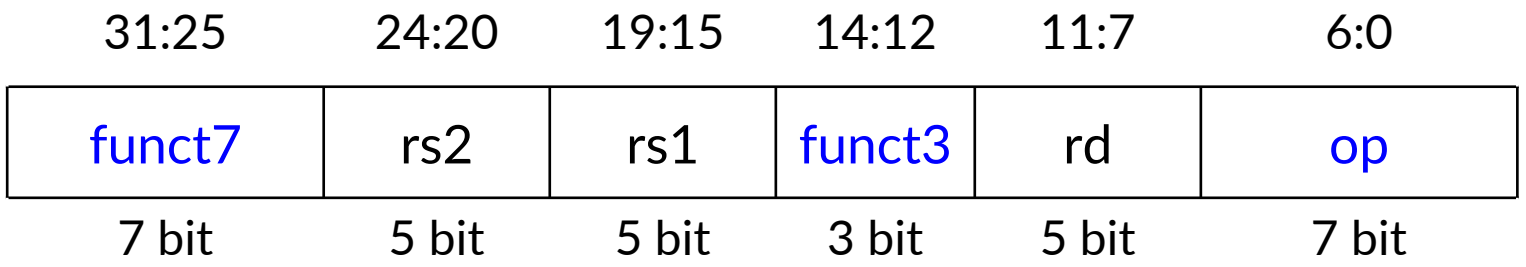
- Bộ đếm chương trình *PC*:
  - Thanh ghi 32-bit chứa địa chỉ của lệnh hiện tại
- Bộ nhớ lệnh (*Instruction memory*):
  - Chứa các lệnh của chương trình
  - Khi có địa chỉ lệnh từ PC đưa đến thì lệnh được đọc ra
- Bộ cộng (*Add*): được sử dụng tăng nội dung PC thêm 4 để trỏ tới lệnh kế tiếp

Thực hiện phần tìm nạp lệnh



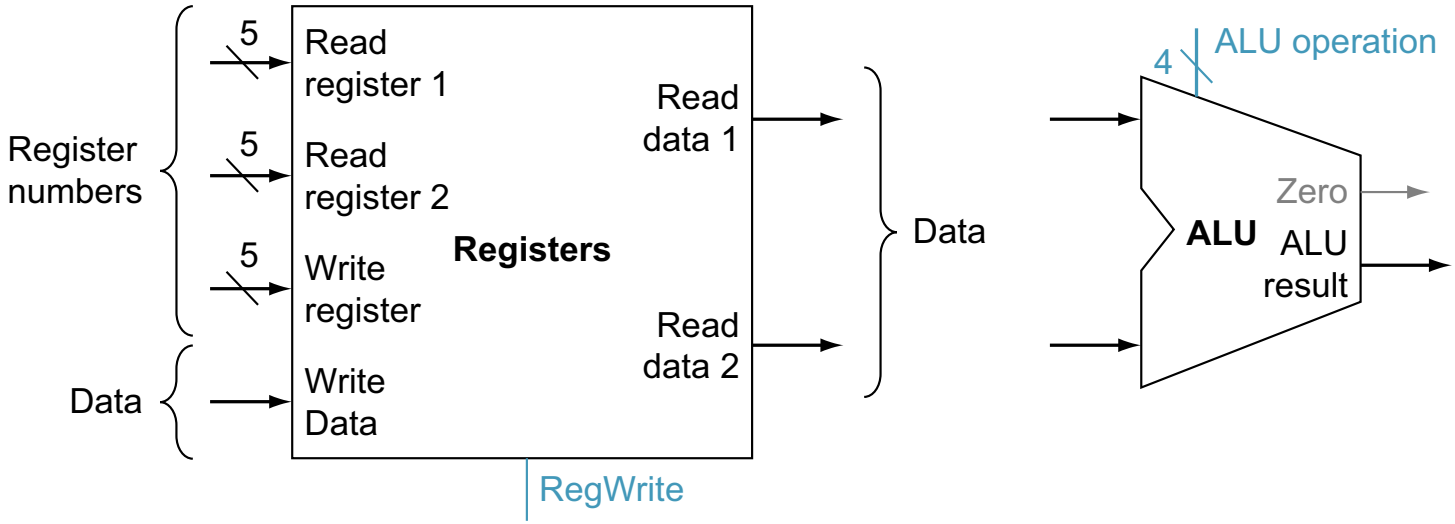
Thực hiện lệnh số học/logic kiểu R (Register-type)

```
add   rd, rs1, rs2   # rd = rs1 + rs2
and   rd, rs1, rs2   # rd = rs1 & rs2
```



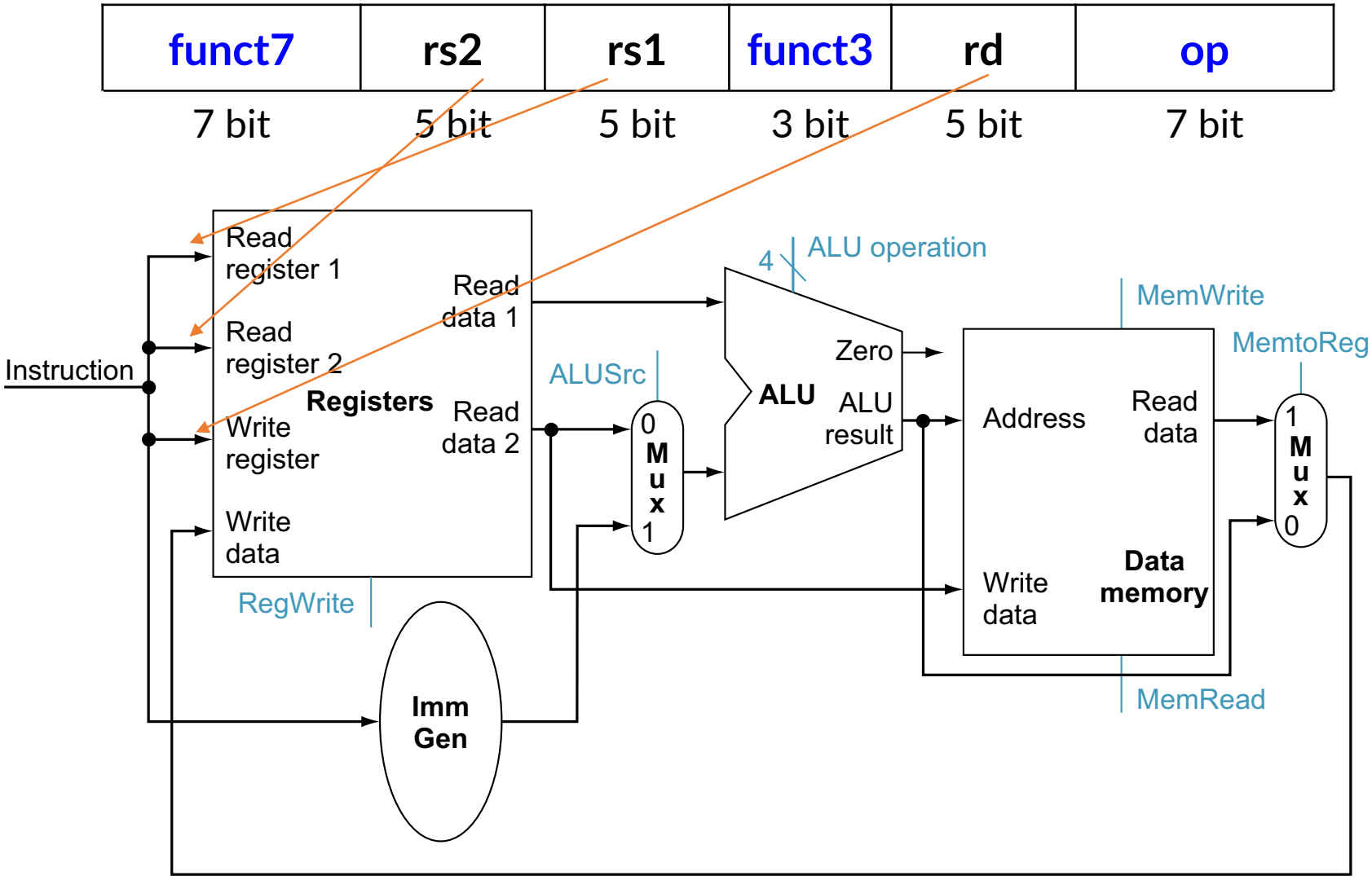
- rs1, rs2: các thanh ghi nguồn (source registers)
- rd: thanh ghi đích (destination register)
- op: mã thao tác (operation code hay opcode)
- funct7, funct3: các trường function, cùng với opcode chỉ ra cho CPU biết thao tác cần thực hiện

Các thành phần thực hiện lệnh SH/LG kiểu R



- Tập thanh ghi (*Registers*): có 32 thanh ghi 32-bit, mỗi thanh ghi được xác định bởi số hiệu 5-bit
  - Read register 1, Read register 2: các đầu vào để chọn các thanh ghi cần đọc
  - Write register: đầu vào để chọn thanh ghi cần ghi (thanh ghi đích)
  - Read data 1, Read data 2: hai đầu ra dữ liệu được đọc từ thanh ghi (32-bit)
  - Write Data: đầu vào dữ liệu ghi vào thanh ghi đích(32-bit)
  - RegWrite: tín hiệu điều khiển ghi dữ liệu vào thanh ghi
- *ALU* để thực hiện các phép toán số học/logic

Mô tả thực hiện lệnh SH/LG kiểu R

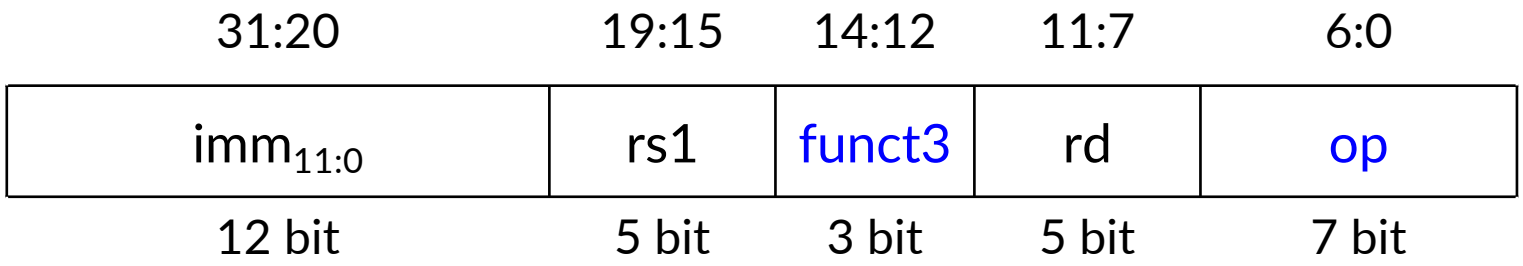


Mô tả thực hiện lệnh SH/LG kiểu R

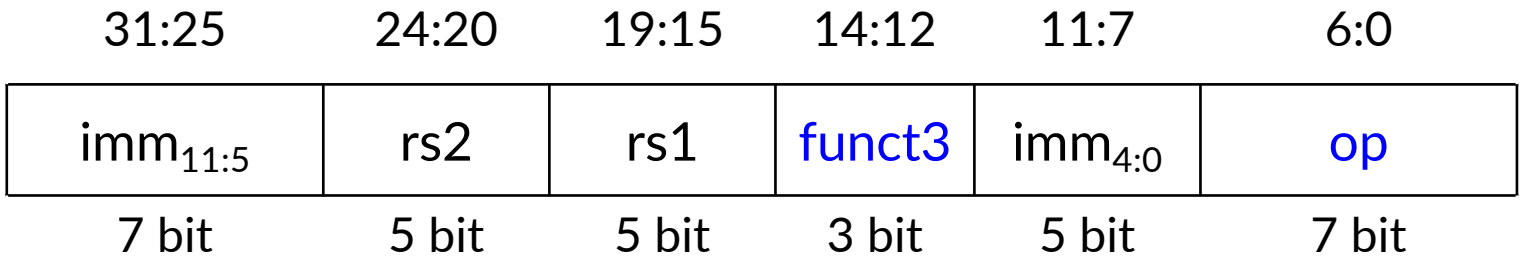
- Hai số hiệu thanh ghi rs1 và rs2 đưa đến hai đầu vào Read register 1, Read register 2 để chọn hai thanh ghi nguồn
- Số hiệu thanh ghi rd đưa đến đầu vào Write register để chọn thanh ghi đích
- Hai dữ liệu được đọc từ hai thanh ghi nguồn được đưa ra 2 đầu ra Read data 1 và Read data 2 , rồi đưa đến đầu vào của ALU
- ALU operation (4-bit): tín hiệu điều khiển chọn phép toán ở ALU
- ALU thực hiện phép toán tương ứng, kết quả phép toán ở đầu ra ALU result được đưa về đầu vào Write Data của tập thanh ghi để ghi vào thanh ghi đích
- RegWrite: tín hiệu điều khiển ghi dữ liệu vào thanh ghi đích

Thực hiện lệnh lw/sw

```
lw rd, imm(rs1) # rd = mem[rs1+SignExtImm]
```

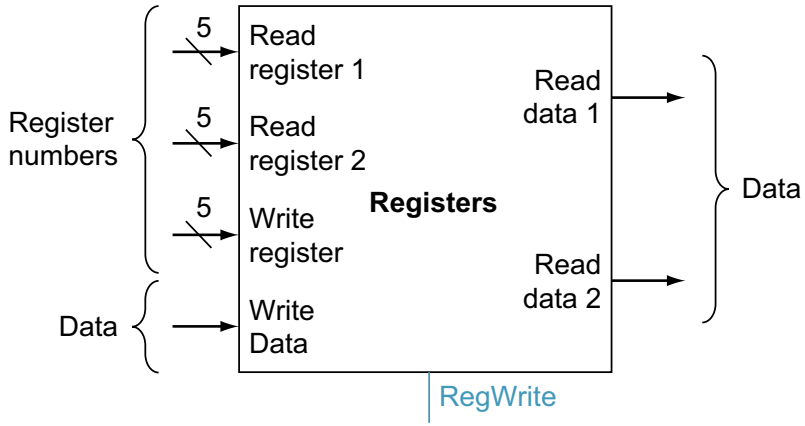



```
sw rs2, imm(rs1) # mem[rs1+SignExtImm]= rs2
```

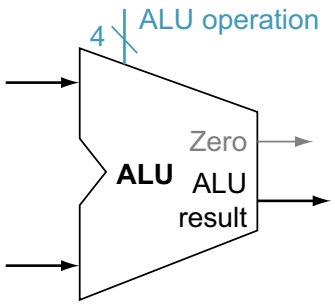


Các thành phần thực hiện các lệnh lw/sw

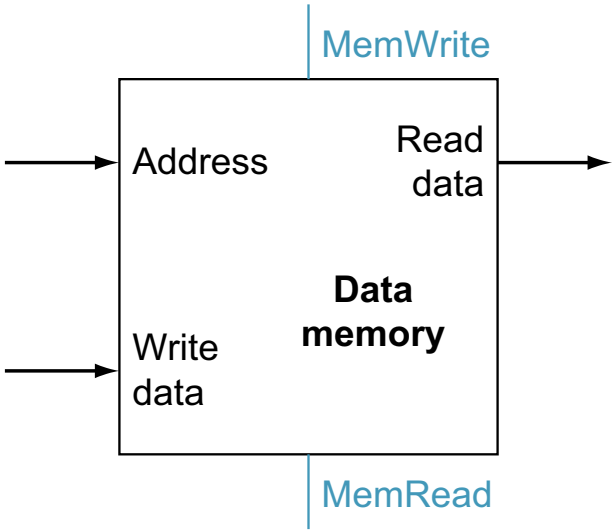
Tập thanh ghi



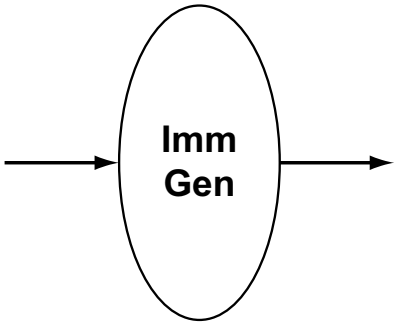
ALU



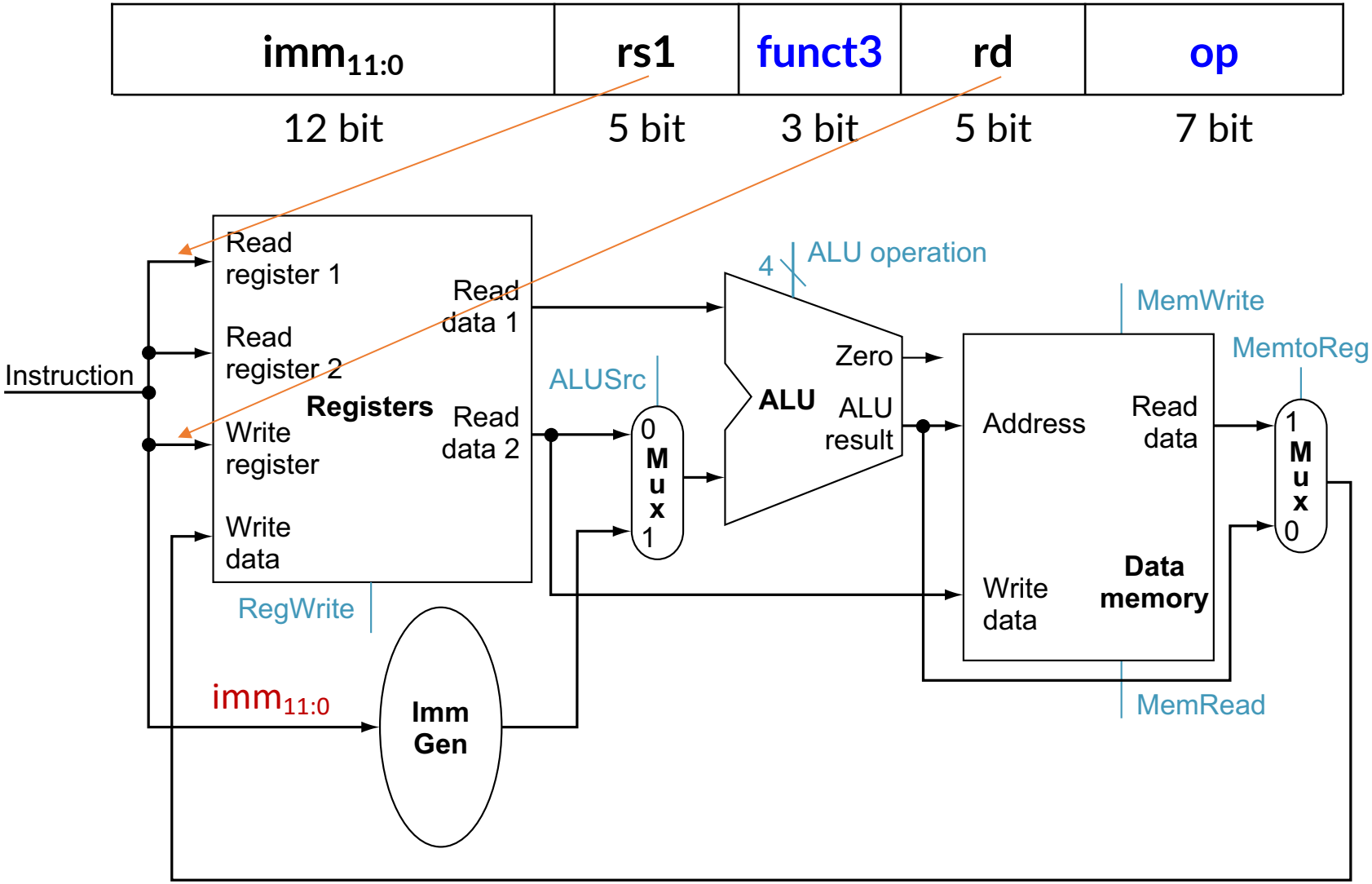
Bộ nhớ dữ liệu



Bộ mở rộng hằng số có dấu từ 12-bit → 32-bit



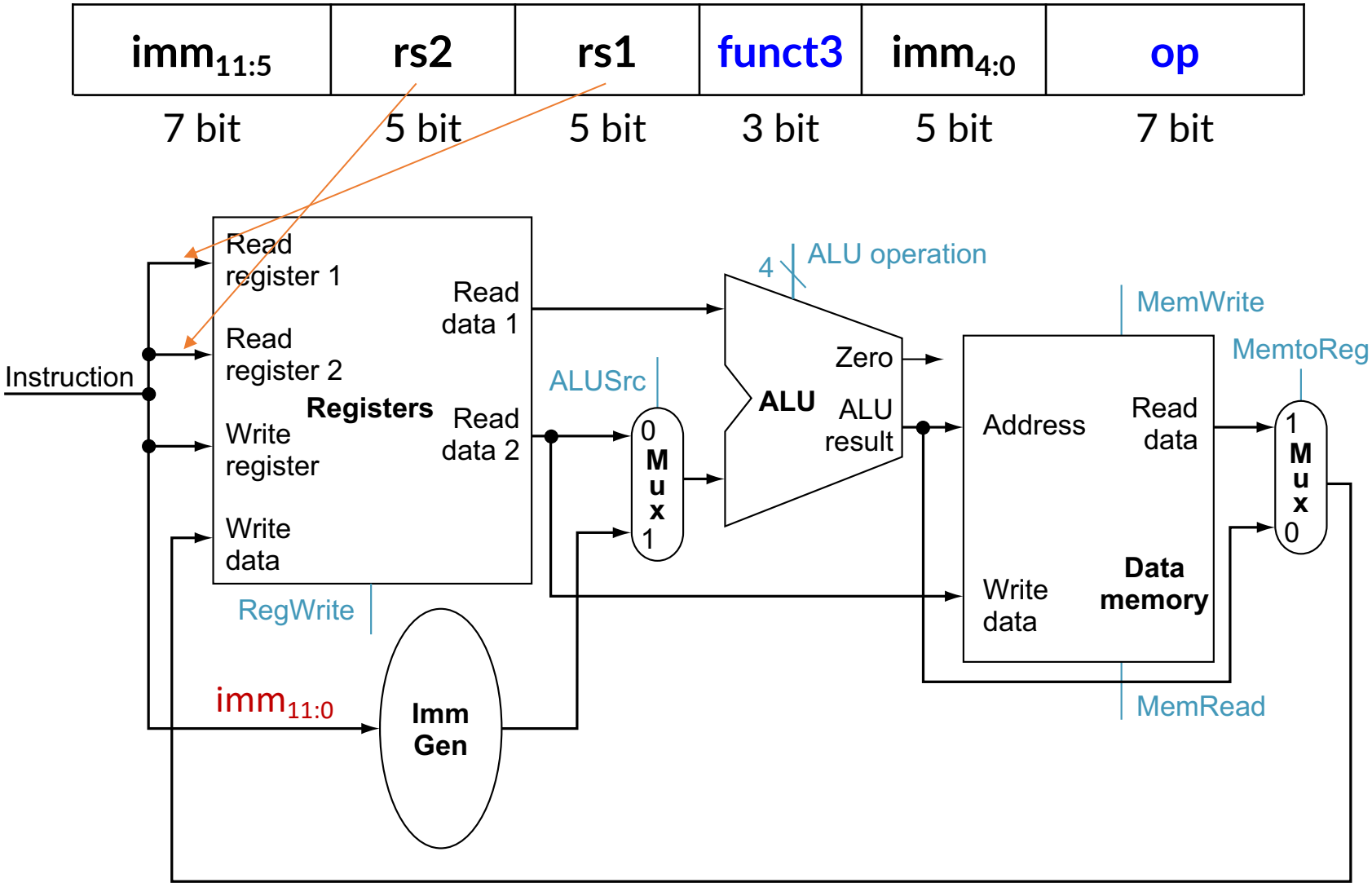
Mô tả thực hiện lệnh lw



Thực hiện lệnh lw

- Số hiệu thanh ghi rs1 đưa đến đầu vào Read register 1 để chọn thanh ghi cơ sở rs1, nội dung rs1 được đưa đến đầu ra Read Data 1, rồi chuyển đến ALU
- Hằng số imm 12-bit được đưa đến bộ Imm Gen để mở rộng thành 32-bit, rồi chuyển đến ALU
- ALU cộng hai giá trị trên, kết quả ở đầu ra ALU result chính là địa chỉ của dữ liệu cần đọc từ bộ nhớ dữ liệu; địa chỉ này được đưa đến đầu vào Address của bộ nhớ dữ liệu
- Số hiệu thanh ghi rd đưa đến đầu vào Write Register để chọn thanh ghi đích
- Dữ liệu 32-bit ở bộ nhớ dữ liệu, tại vị trí địa chỉ đã được tính, được đọc ra ở đầu ra Read data của bộ nhớ dữ liệu nhờ tín hiệu điều khiển MemRead, rồi đưa về đầu vào Write Data của tập thanh ghi để ghi vào thanh ghi đích rd

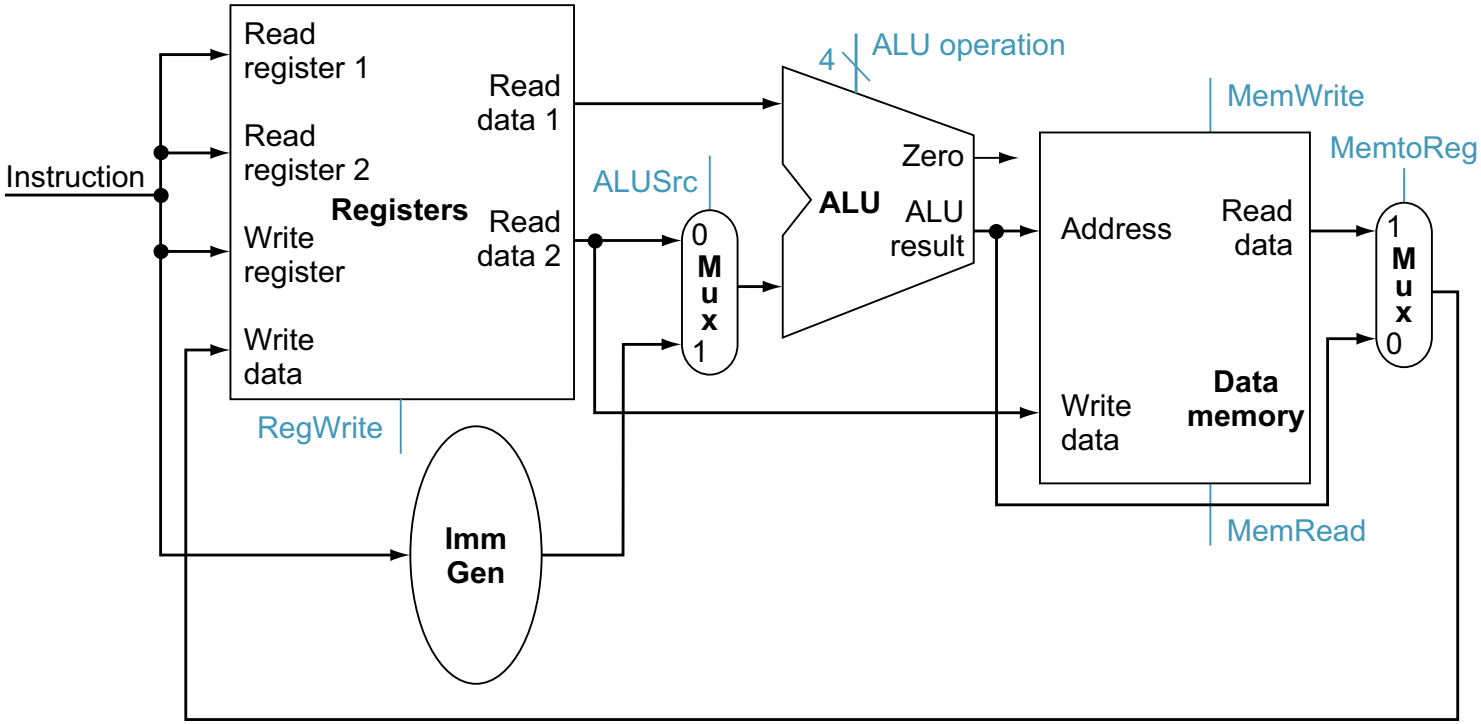
Mô tả thực hiện lệnh sw



Thực hiện lệnh sw

- Số hiệu thanh ghi rs1 đưa đến đầu vào Read register 1 để chọn thanh ghi cơ sở rs1, nội dung rs1 được đưa ra đầu ra Read Data 1, rồi chuyển đến ALU
- Hằng số imm 12-bit đưa đến bộ Imm Gen để mở rộng thành 32-bit, rồi chuyển đến ALU
- ALU cộng hai giá trị trên, kết quả ở đầu ra ALU result chính là địa chỉ của vị trí ở bộ nhớ dữ liệu; địa chỉ này được đưa đến đầu vào Address của bộ nhớ dữ liệu
- Số hiệu thanh ghi rs2 đưa đến đầu vào Read register 2 để chọn thanh ghi dữ liệu nguồn rs2, nội dung rs2 được đưa ra đầu ra Read data 2
- Dữ liệu 32-bit này sẽ được đưa đến đầu vào Write data của bộ nhớ dữ liệu và được ghi vào ngăn nhớ có địa chỉ đã được tính, nhờ tín hiệu điều khiển MemWrite

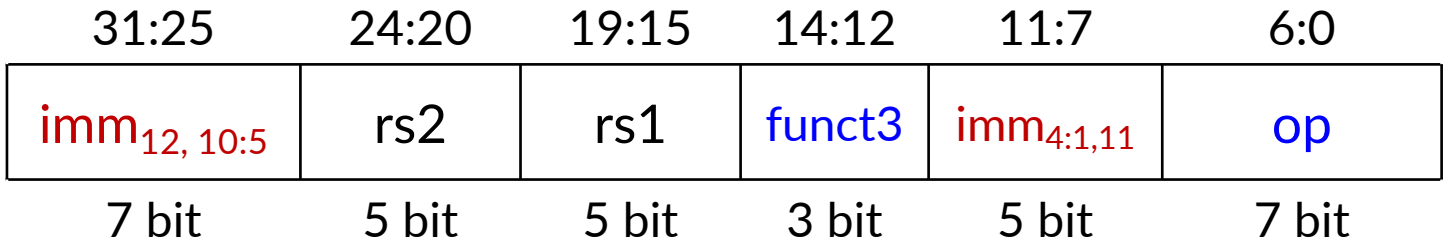
Datapath cho các lệnh SH/LG kiểu R/Load/Store



- ALUSrc: tín hiệu điều khiển chọn toán hạng đưa đến ALU:
  - Lệnh kiểu R: toán hạng từ thanh ghi nguồn thứ hai (ALUSrc = 0)
  - Lệnh lw/sw: Hằng số imm 12-bit được mở rộng thành 32-bit (tính địa chỉ) (ALUSrc=1)
- MemtoReg: tín hiệu điều khiển chọn dữ liệu đưa về thanh ghi đích:
  - Lệnh kiểu R: lấy kết quả từ ALU result (MemtoReg = 0)
  - Lệnh lw: dữ liệu đọc (Read data) từ bộ nhớ dữ liệu (MemtoReg = 1)

Thực hiện lệnh branch

```
beq   rs1, rs2, label
(bne, blt, bge, bltu, bgeu)
```

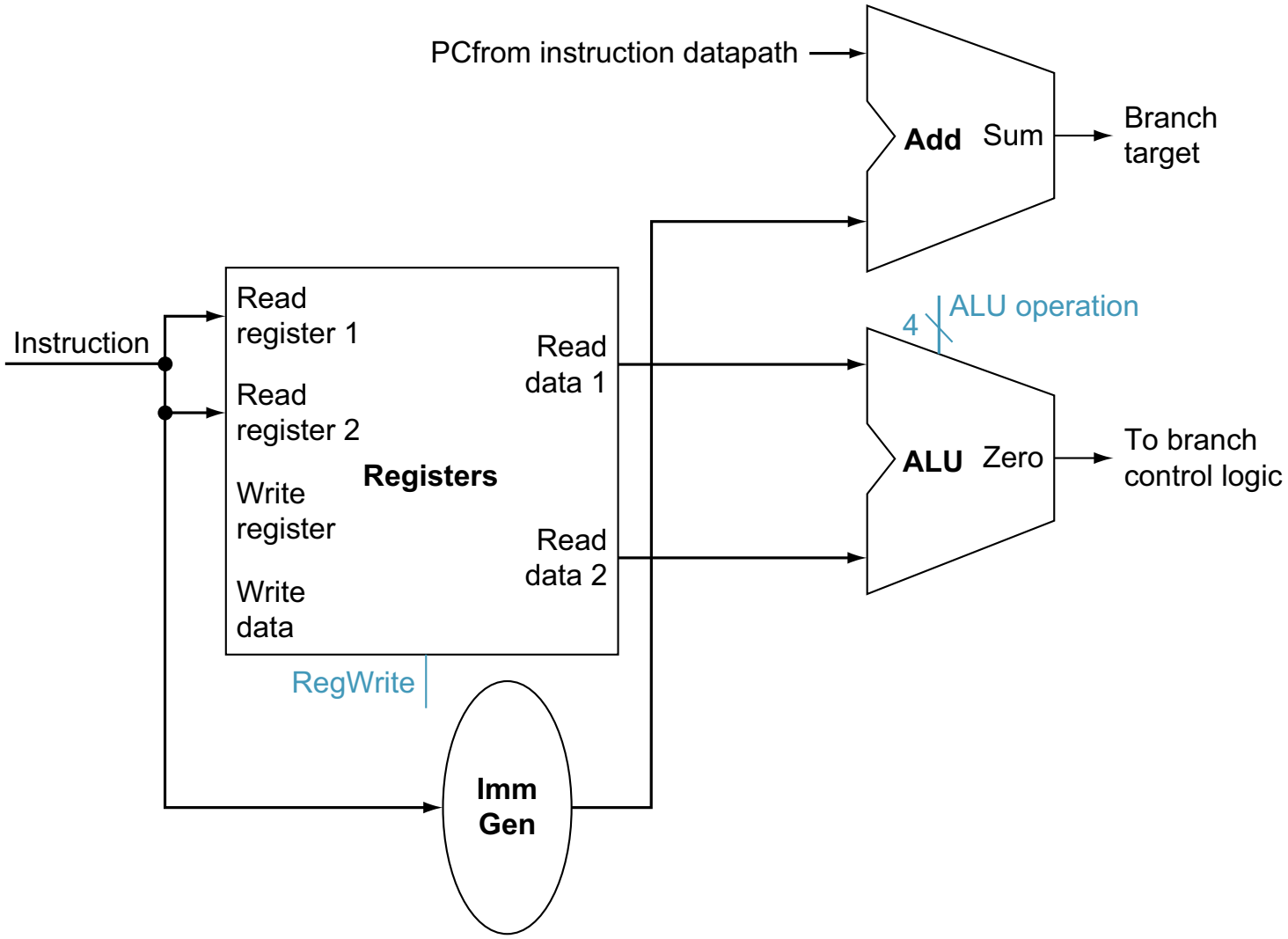


funct3

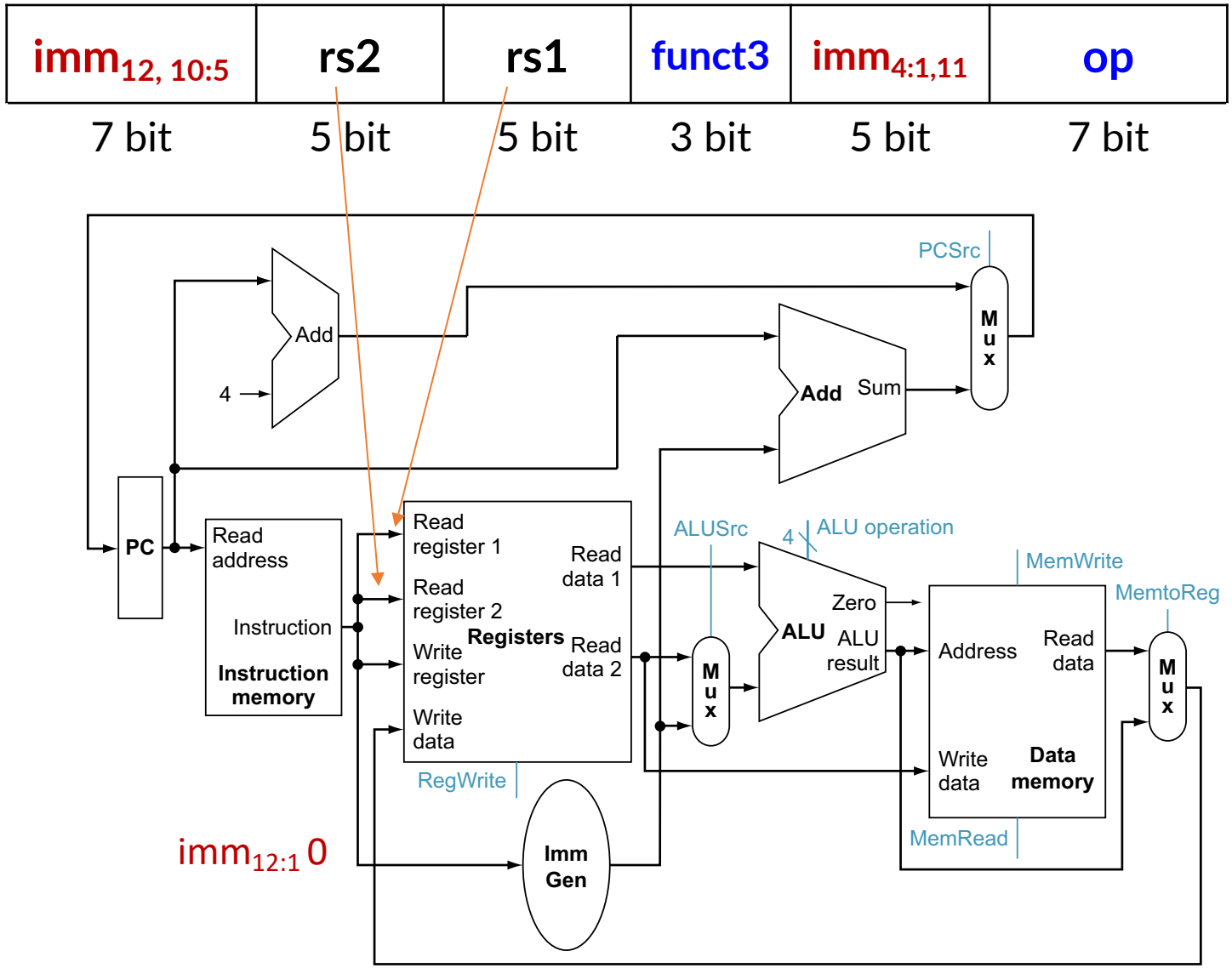
000	beq
001	bne
100	blt
101	bge
110	bltu
111	bgeu

- So sánh nội dung hai thanh ghi rs1 và rs2:
  - Nếu điều kiện đúng: rẽ nhánh đến nhãn đích
    - PC ← PC + SignExt({imm12:1,1'b0})
  - Nếu điều kiện sai: chuyển sang thực hiện lệnh kế tiếp
    - PC ← PC + 4

Các thành phần thực hiện lệnh Branch



Thực hiện lệnh beq



Thực hiện lệnh beq

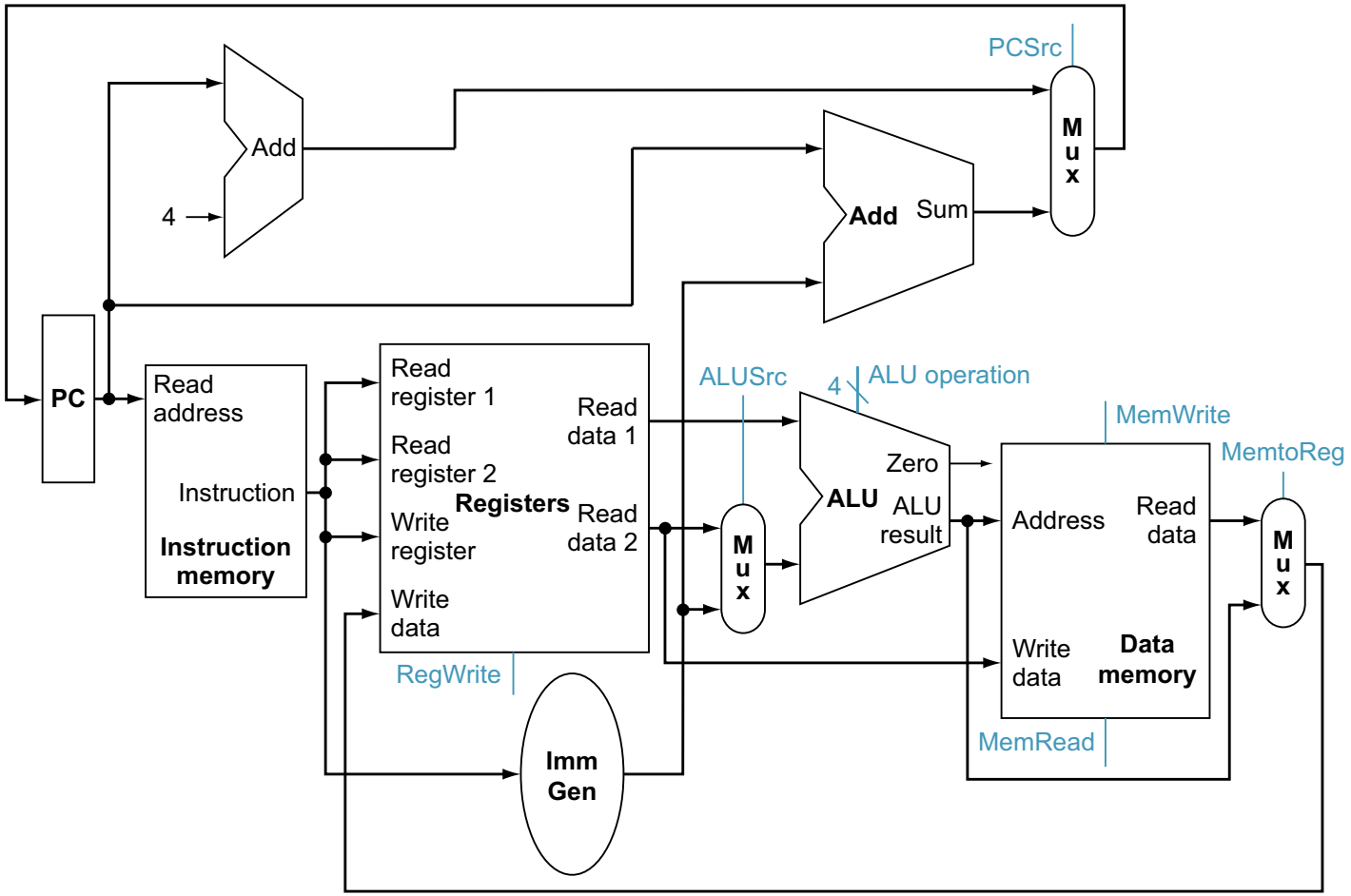
- Nhờ các số hiệu thanh ghi rs1 và rs2, hai toán hạng nguồn được đọc ra đưa đến ALU để so sánh
- ALU trừ hai toán hạng và thiết lập giá trị ở đầu ra "Zero"
  - Hiệu = 0 → đầu ra Zero = 1
  - Hiệu <> 0 → đầu ra Zero = 0
  - Đầu ra Zero này được đưa đến mạch logic điều khiển rẽ nhánh
- Bộ cộng Add tính địa chỉ đích rẽ nhánh
  - Hằng số imm 12-bit được mở rộng theo kiểu có dấu thành 32-bit, rồi dịch trái 1 bit
  - Cộng với PC

  → Địa chỉ đích = PC + (imm đã mở rộng 32-bit, dịch trái 1 bit)
- Điều kiện đúng: PC ← địa chỉ đích rẽ nhánh (rẽ nhánh xảy ra)
- Điều kiện sai: PC ← PC+4 (chuyển sang lệnh kế tiếp)

Hợp các thành phần cho các lệnh

- Datapath cho các lệnh thực hiện trong 1 chu kỳ
  - Mỗi phần tử của datapath chỉ có thể làm một chức năng trong mỗi chu kỳ
  - Do đó, cần tách rời bộ nhớ lệnh và bộ nhớ dữ liệu
- Sử dụng các bộ chọn kênh để chọn dữ liệu nguồn cho các lệnh khác nhau

Datapath đơn giản cho các lệnh R/lw/sw/branch



- PCSrc: tín hiệu điều khiển chọn giá trị cập nhật PC
  - Không rẽ nhánh: PC ← PC+4 (PCSrc = 0)
  - Rẽ nhánh: PC ← PC + (imm 12-bit được mở rộng thành 32-bit <<1) (PCSrc = 1)

2. Thiết kế Control Unit

- Đơn vị điều khiển có hai phần:
  - Bộ điều khiển ALU
  - Bộ điều khiển chính

Thiết kế bộ điều khiển ALU

- ALU được sử dụng để:
  - Load/Store: F = add (xác định địa chỉ bộ nhớ dữ liệu)
  - Branch: F = subtract (so sánh)
  - Các lệnh số học/logic : F phụ thuộc vào funct code

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

Tín hiệu điều khiển ALU

- Bộ điều khiển ALU sử dụng mạch logic tổ hợp:
  - Đầu vào: 2-bit ALUOp được tạo ra từ opcode của lệnh và các bit của funct7 và funct3
  - Đầu ra: các tín hiệu điều khiển ALU 4-bit (ALU control)

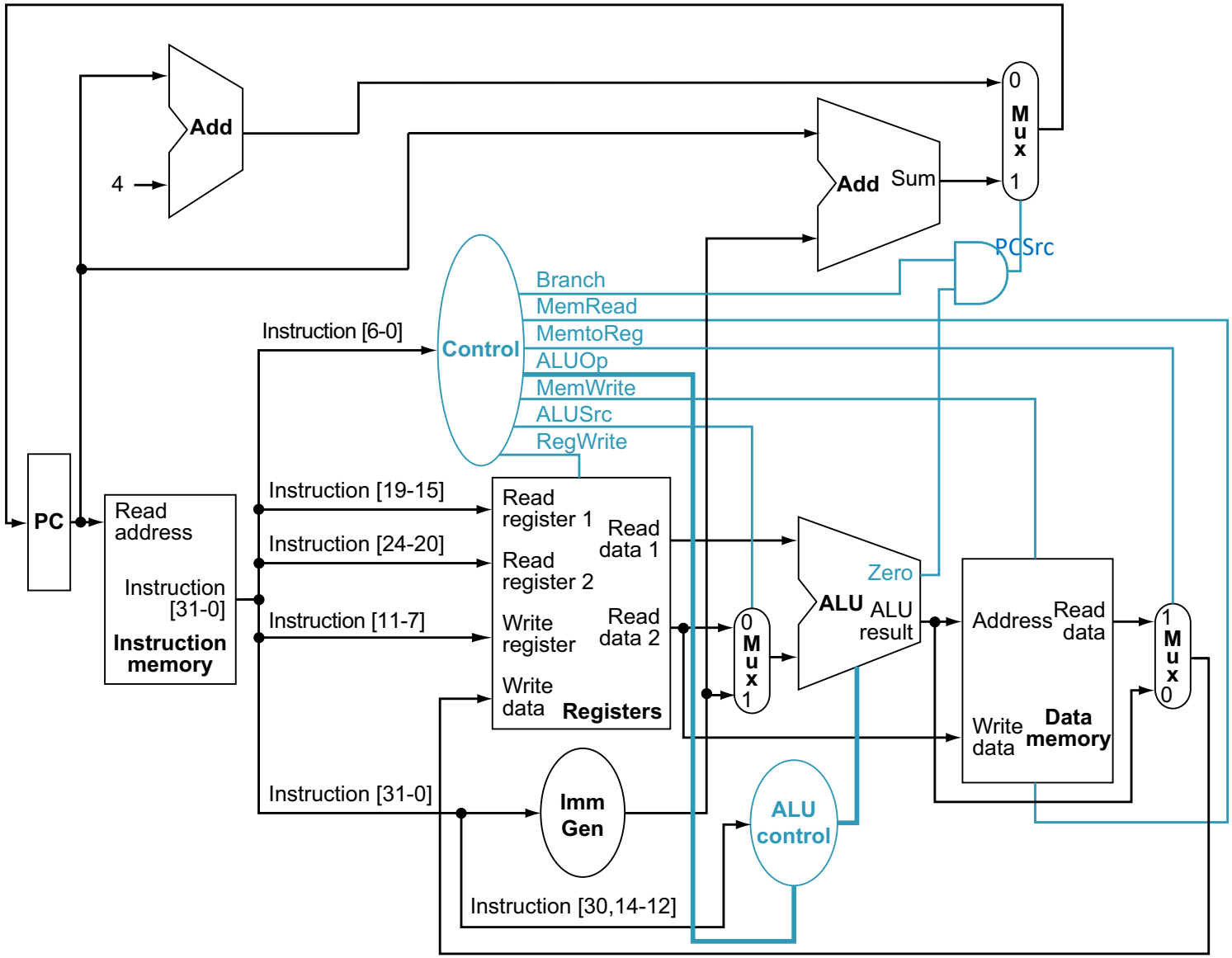
Opcode	ALUOp	Operation	funct7	funct3	ALU function	ALU control
lw	00	load word	XXXXXXX	XXX	add	0010
sw	00	store word	XXXXXXX	XXX	add	0010
beq	01	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	10	add	0000000	000	add	0010
		sub	0100000	000	subtract	0110
		and	0000000	111	AND	0000
		or	0000000	110	OR	0001

Thiết kế bộ điều khiển chính

- Các tín hiệu điều khiển được tạo ra từ lệnh

7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
funct7	rs2	rs1	funct3	rd	op	R-Type
$imm_{11:0}$		rs1	funct3	rd	op	I-Type
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	S-Type
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	B-Type
$imm_{31:12}$				rd	op	U-Type
$imm_{20,10:1,11,19:12}$				rd	op	J-Type
20 bit				5 bit	7 bit	

Datapath và Control Unit



Các tín hiệu điều khiển

Tên tín hiệu	Hiệu ứng khi tín hiệu = 0	Hiệu ứng khi tín hiệu = 1
Branch	Không có lệnh rẽ nhánh beq	Có lệnh rẽ nhánh beq (Branch =1) & (Zero=1): rẽ nhánh xảy ra (Branch =1) & (Zero=0): rẽ nhánh không xảy ra
RegWrite	Không	Ghi dữ liệu trên đầu vào Write Data ở tập thanh ghi đến thanh ghi đích
ALUSrc	Toán hạng thứ hai của ALU lấy từ thanh ghi nguồn thứ hai (Read data 2)	Toán hạng thứ hai của ALU là giá trị 12 bit của lệnh được mở rộng có dấu thành 32-bit
PCSrc	PC ← PC+4	PC ← PC + SignExt({imm12:1,1'b0})

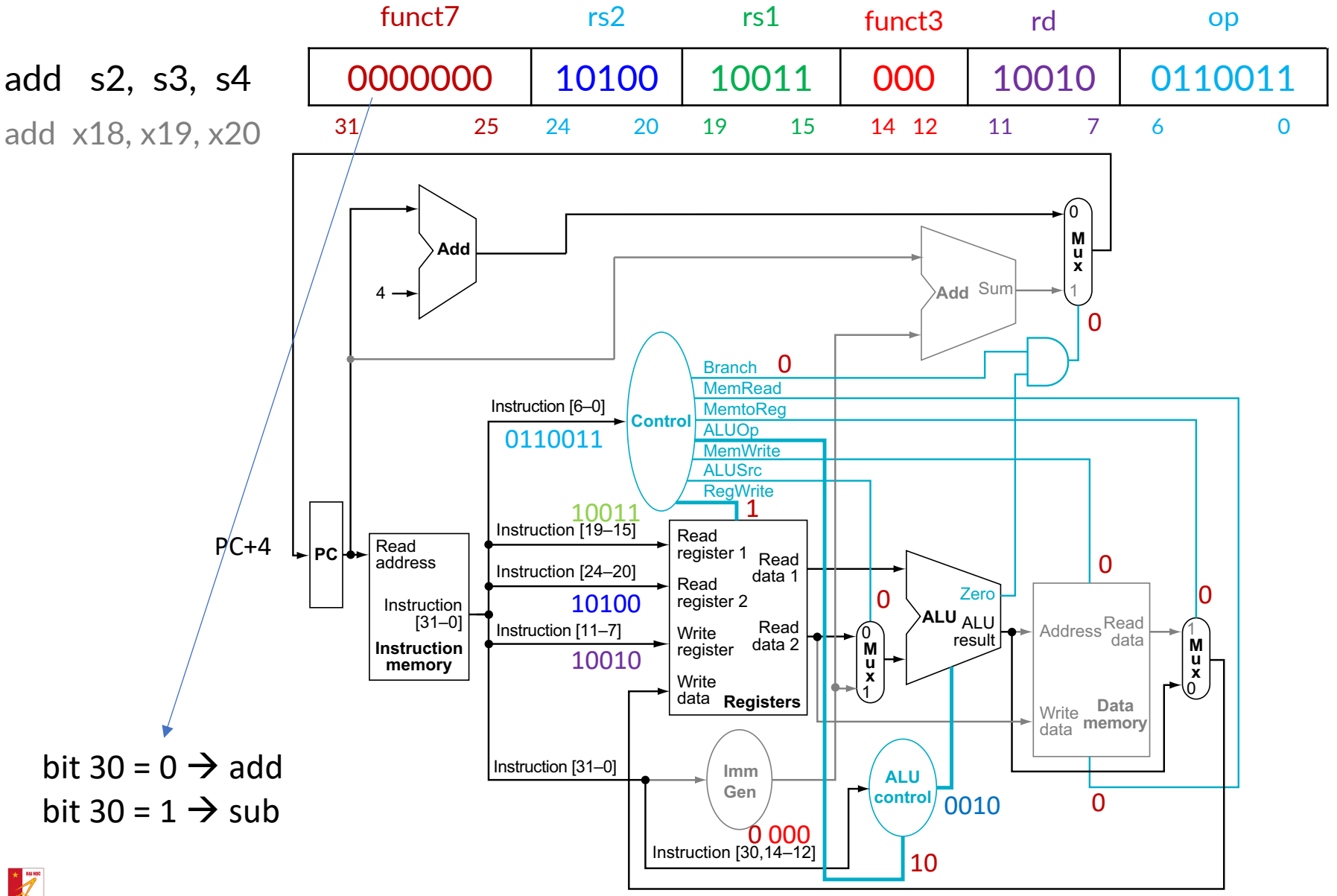
Các tín hiệu điều khiển (tiếp)

Tên tín hiệu	Hiệu ứng khi tín hiệu = 0	Hiệu ứng khi tín hiệu = 1
MemRead	Không	Nội dung ngăn nhớ dữ liệu, được xác định bởi địa chỉ do ALU tính, được đưa ra đầu ra *Read data* của bộ nhớ dữ liệu
MemWrite	Không	Dữ liệu trên đầu vào *Write Data* của bộ nhớ dữ liệu được ghi vào ngăn nhớ có địa chỉ do ALU tính
MemtoReg	Giá trị được đưa đến đầu vào *Write data* của tập thanh ghi là từ *ALU result*	Giá trị được đưa đến đầu vào *Write data* của tập thanh ghi là từ bộ nhớ dữ liệu

Ví dụ thực hiện lệnh kiểu R

add s2, s3, s4

add x18, x19, x20



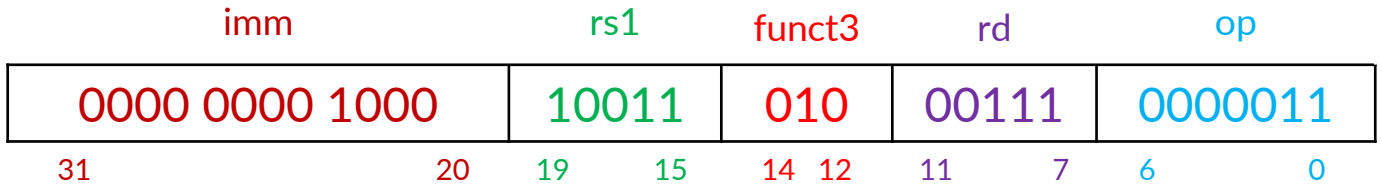
bit 30 = 0 → add

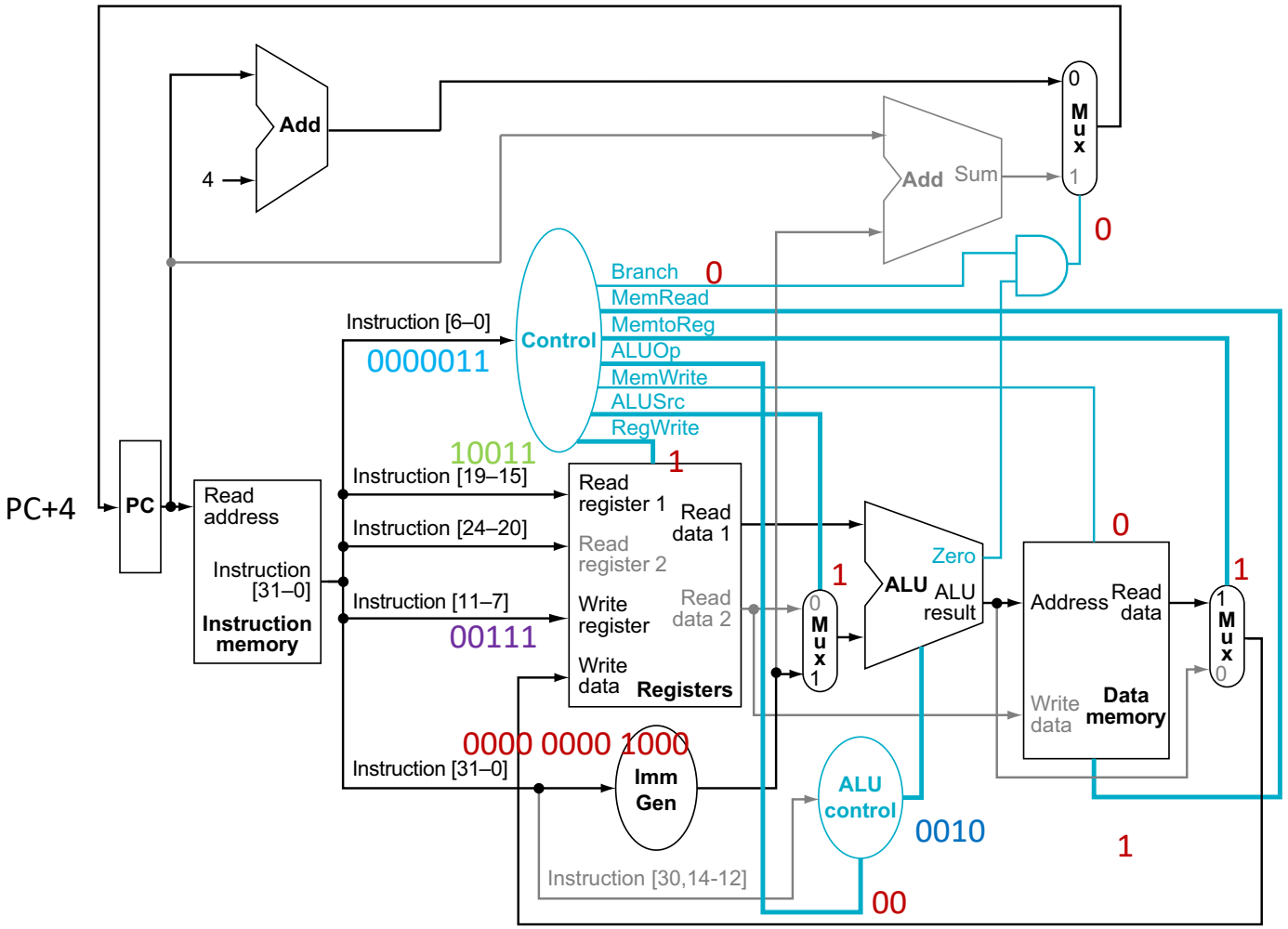
bit 30 = 1 → sub

Ví dụ thực hiện lệnh lw

lw t2, 8(s3)

lw x7, 8(x19)

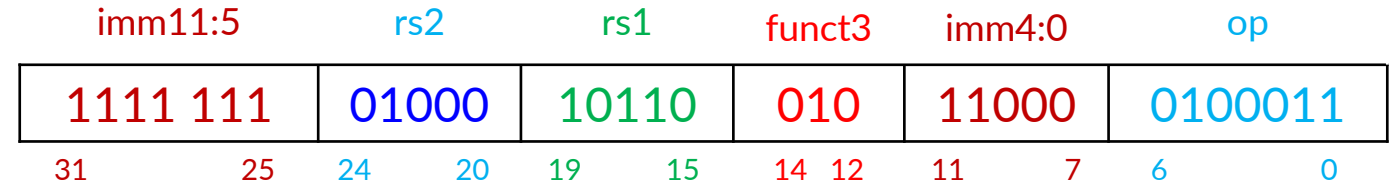


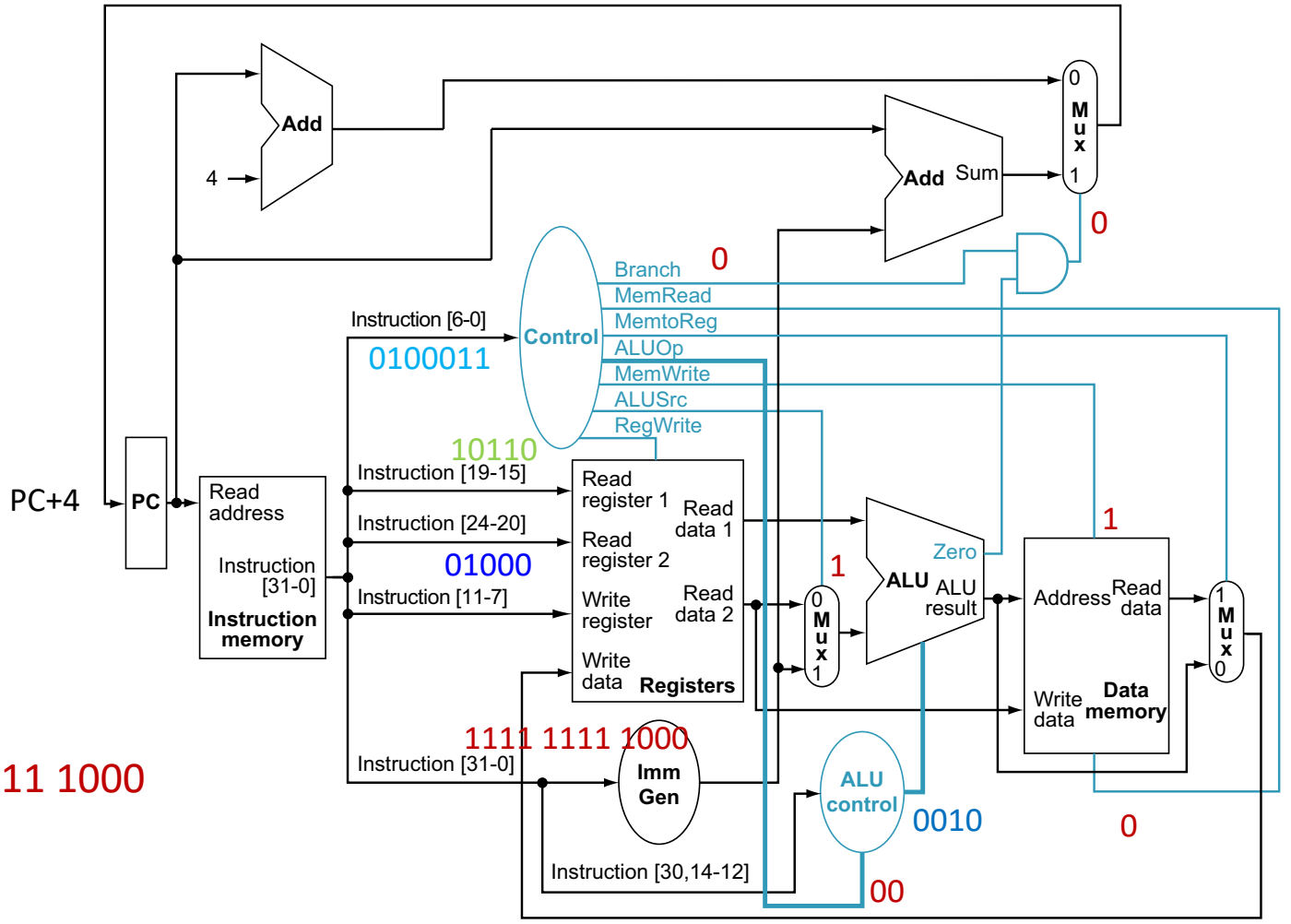


Ví dụ thực hiện lệnh sw

sw s0, -8(s6)

sw x8, -8(x22)





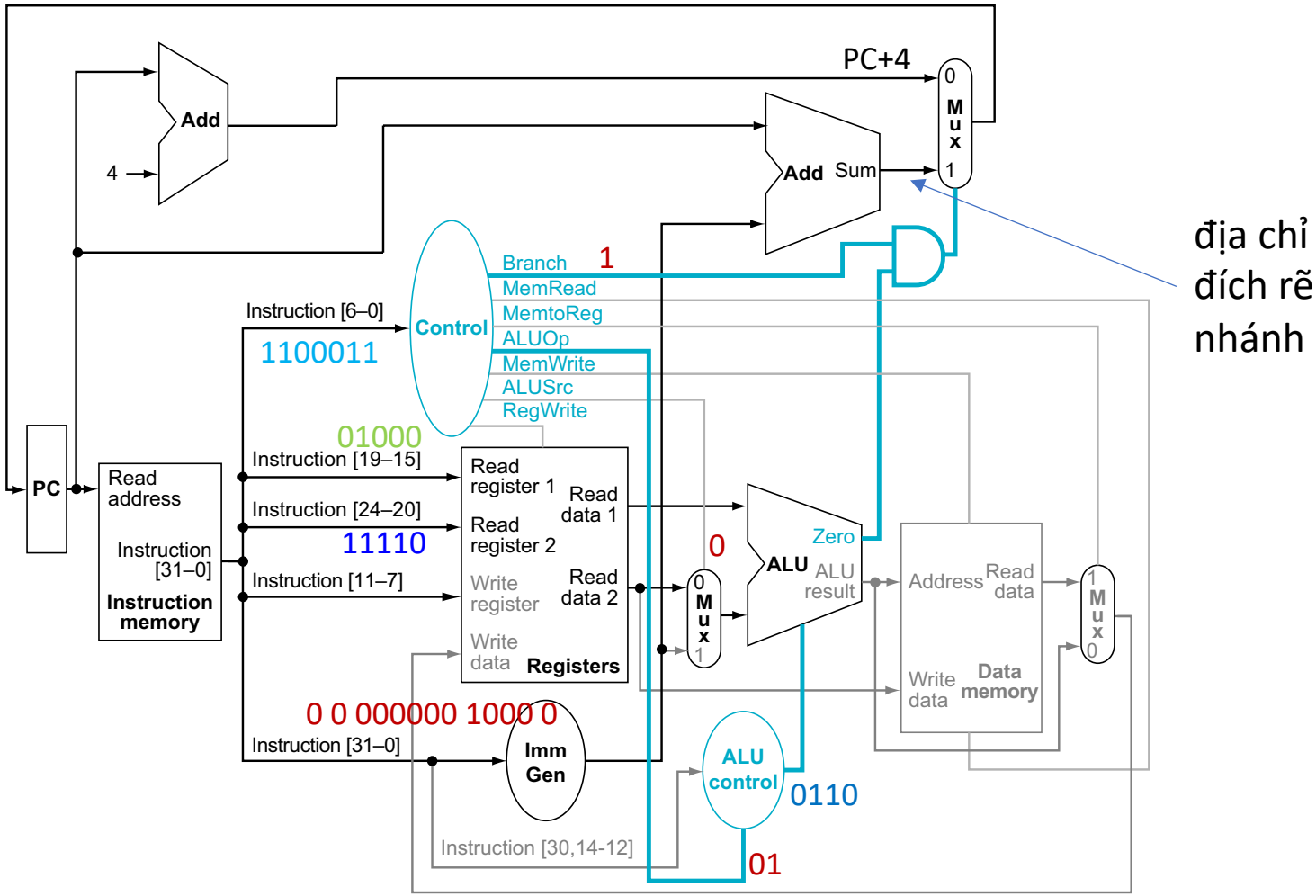
-8 = 1111 1111 1000

Ví dụ thực hiện lệnh kiểu B: beq

beq s0, t5, L1

beq x8, x30, 16

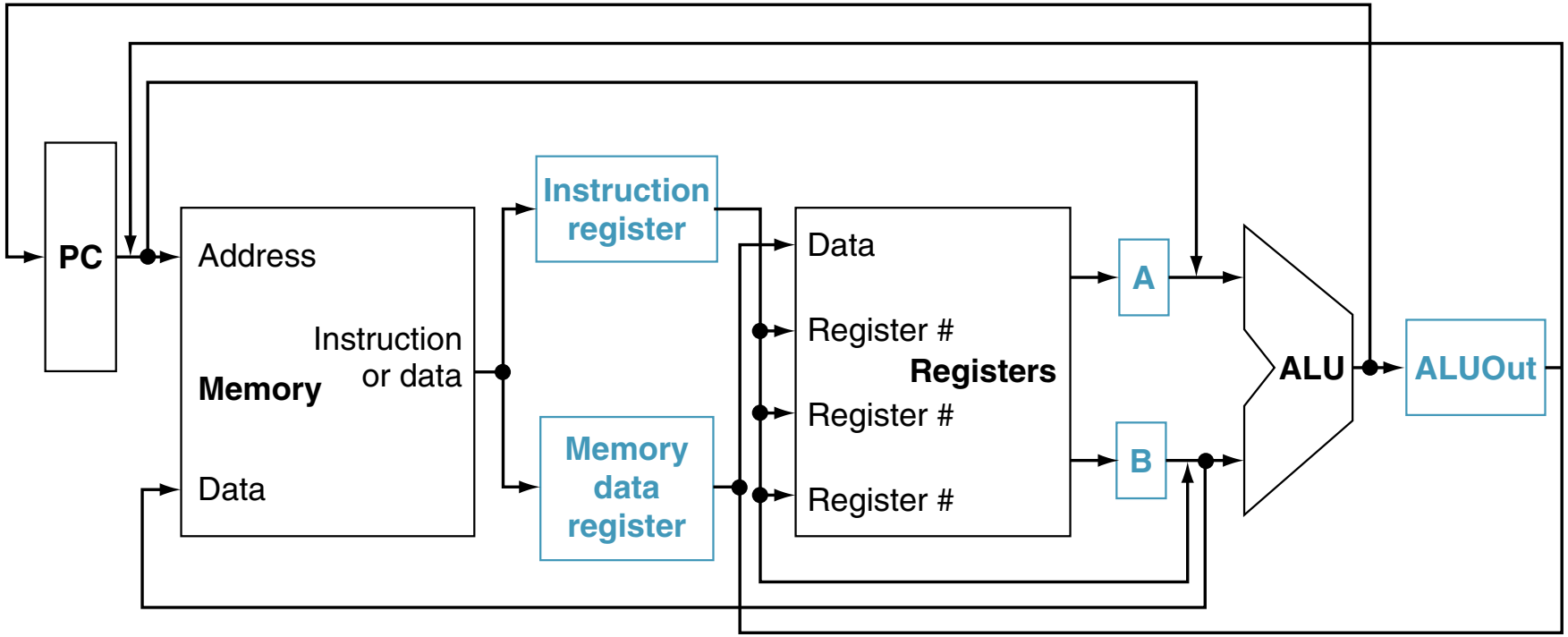
imm12, 10:5	rs2	rs1	funct3	imm4:1,11	op
0 000000	11110	01000	000	1000 0	1100011
31 25	24 20	19 15	14 12	11 7	6 0



5.3. Bộ xử lý đa chu kỳ

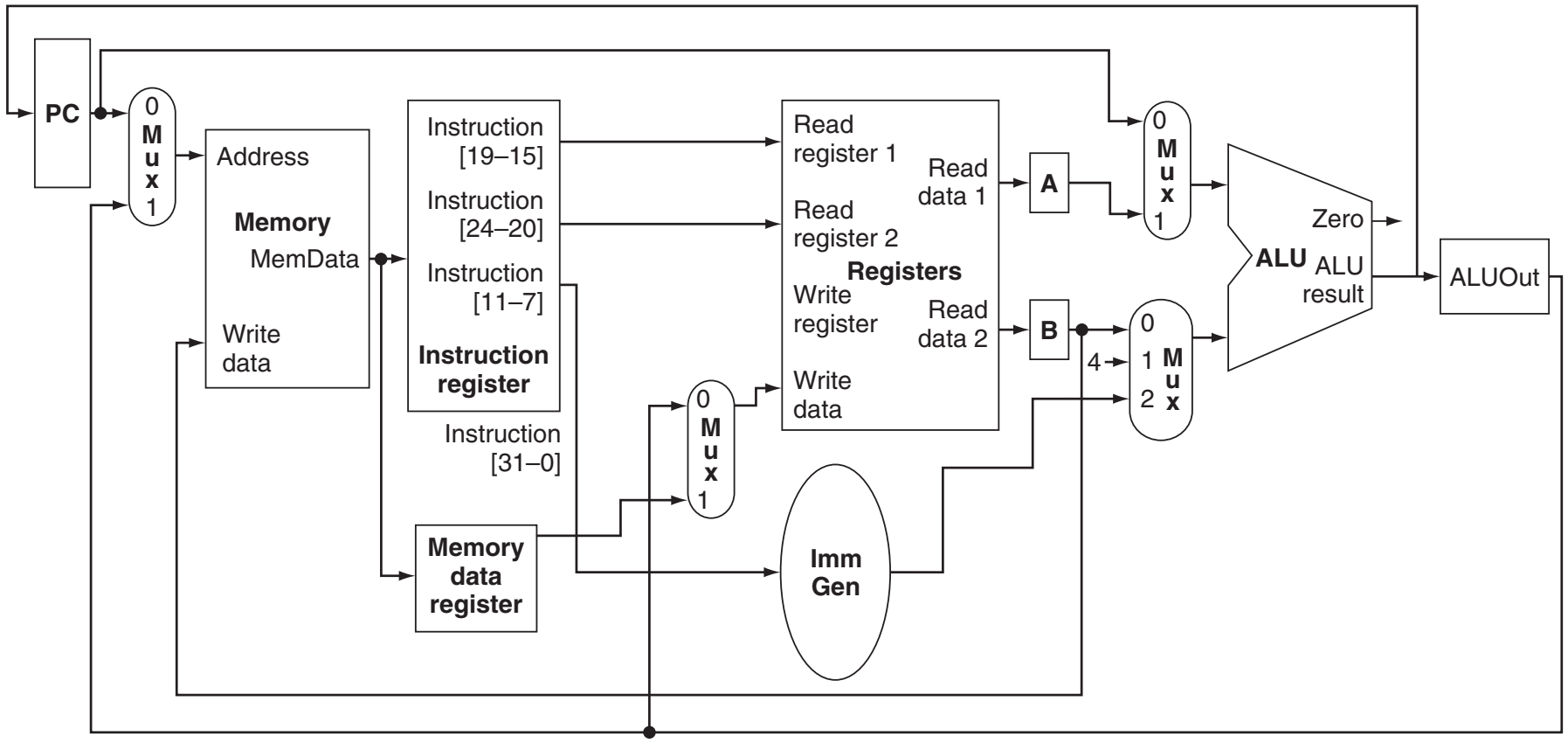
- Một bộ nhớ dùng chung cho cả lệnh và dữ liệu
- Chỉ có một bộ ALU
- Một hoặc nhiều thanh ghi được thêm sau mỗi đơn vị chức năng để giữ đầu ra của đơn vị đó cho đến khi giá trị đó được sử dụng ở chu kỳ tiếp theo

Đường dẫn dữ liệu đa chu kỳ

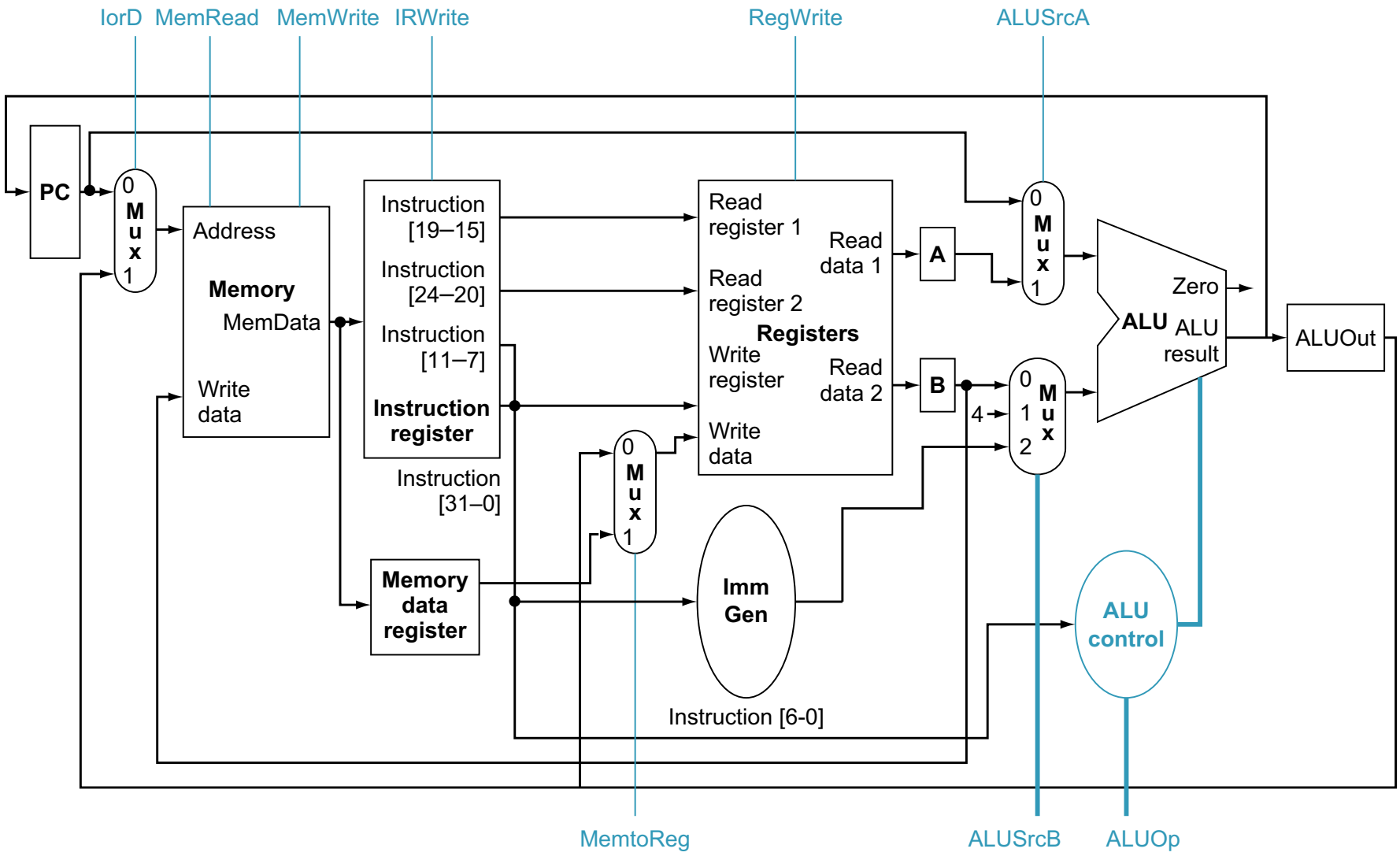


- Thanh ghi lệnh (IR) và Thanh ghi dữ liệu bộ nhớ (MDR): chứa lệnh và dữ liệu được đọc ra từ bộ nhớ
- Các thanh ghi A và B để chứa các giá trị toán hạng được đọc từ các thanh ghi
- Thanh ghi ALUOut chứa đầu ra của ALU

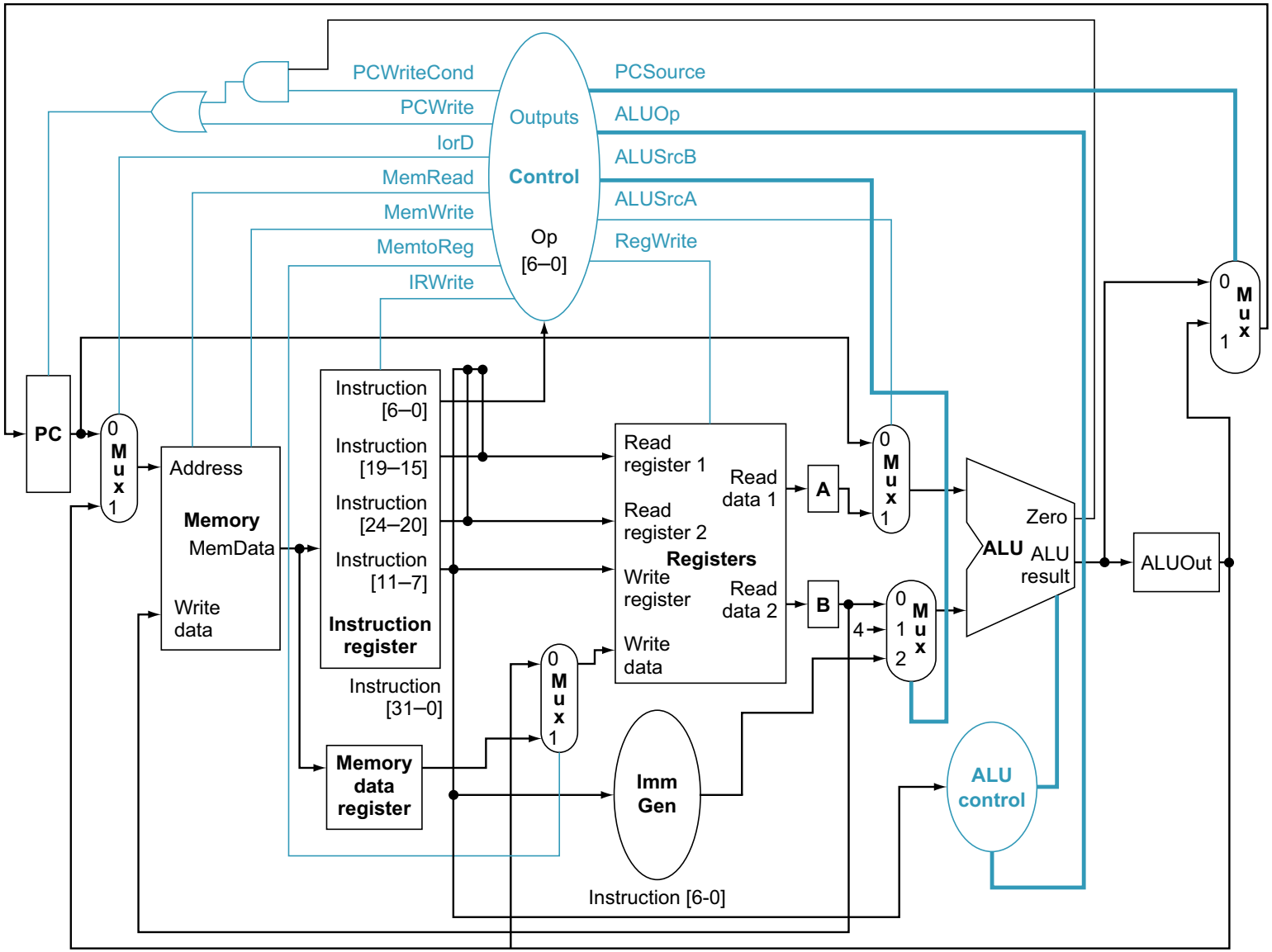
Đường dẫn dữ liệu đa chu kỳ cho các lệnh cơ bản



Đường dẫn dữ liệu đa chu kỳ với các đường điều khiển



Multicycle Datapath và Control Unit



Các tín hiệu điều khiển 1-bit

Tên tín hiệu	Hiệu ứng khi tín hiệu = 0	Hiệu ứng khi tín hiệu = 1
RegWrite	Không	Ghi dữ liệu trên đầu vào Write Data ở tập thanh ghi đến thanh ghi đích
ALUSrcA	Toán hạng thứ nhất của ALU là từ PC	Toán hạng thứ nhất của ALU là từ thanh ghi A
MemRead	Không	Nội dung ngăn nhớ dữ liệu, được xác định bởi địa chỉ do ALU tính, được đưa ra đầu ra *Memory data output* đưa đến *Memory data register (MDR)*
MemWrite	Không	Dữ liệu trên đầu vào *Write Data* của bộ nhớ dữ liệu được ghi vào ngăn nhớ có địa chỉ do ALU tính
MemtoReg	Giá trị được đưa đến đầu vào *Write data* của tập thanh ghi là từ *ALU Out*	Giá trị được đưa đến đầu vào *Write data* của tập thanh ghi là từ MDR
IorD	*PC* cung cấp địa chỉ đến bộ nhớ để đọc lệnh	*ALUOut* cung cấp địa chỉ đến bộ nhớ (với lệnh load/store

Các tín hiệu điều khiển 1-bit (tiếp)

Tên tín hiệu	Hiệu ứng khi tín hiệu = 0	Hiệu ứng khi tín hiệu = 1
IRWrite	Không	Đầu ra của bộ nhớ được ghi đến Thanh ghi lệnh IR
PCWrite	Không	PC được ghi; nguồn được điều khiển bởi PCSource
PCWriteCond	Không	PC được ghi nếu đầu ra *Zero output* = 1

Các tín hiệu điều khiển 2-bit

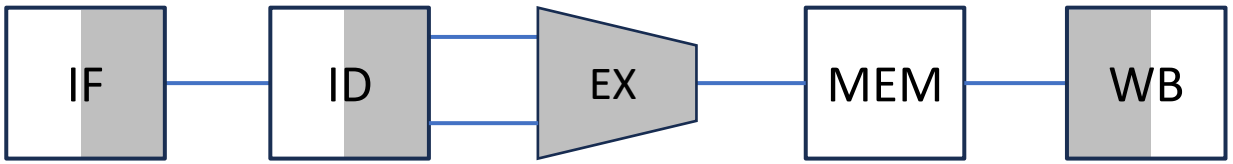
Tên tín hiệu	Giá trị	Hiệu ứng
ALUOp	00	ALU thực hiện phép cộng
	01	ALU thực hiện phép trừ
	10	Trường funct của lệnh xác định phép toán của ALU
ALUSrcB	00	Đầu vào thứ hai của ALU từ thanh ghi B
	01	Đầu vào thứ hai của ALU là hằng số 4
	10	Đầu vào thứ hai của ALU là hằng số imm đã được mở rộng
PCSource	00	Đầu ra của ALU (PC+4) đưa về PC
	01	Nội dung của ALUOut (địa chỉ đích rẽ nhánh) đưa về PC
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC + 4[31:28]) is sent to the PC for writing.

Ví dụ về hiệu năng

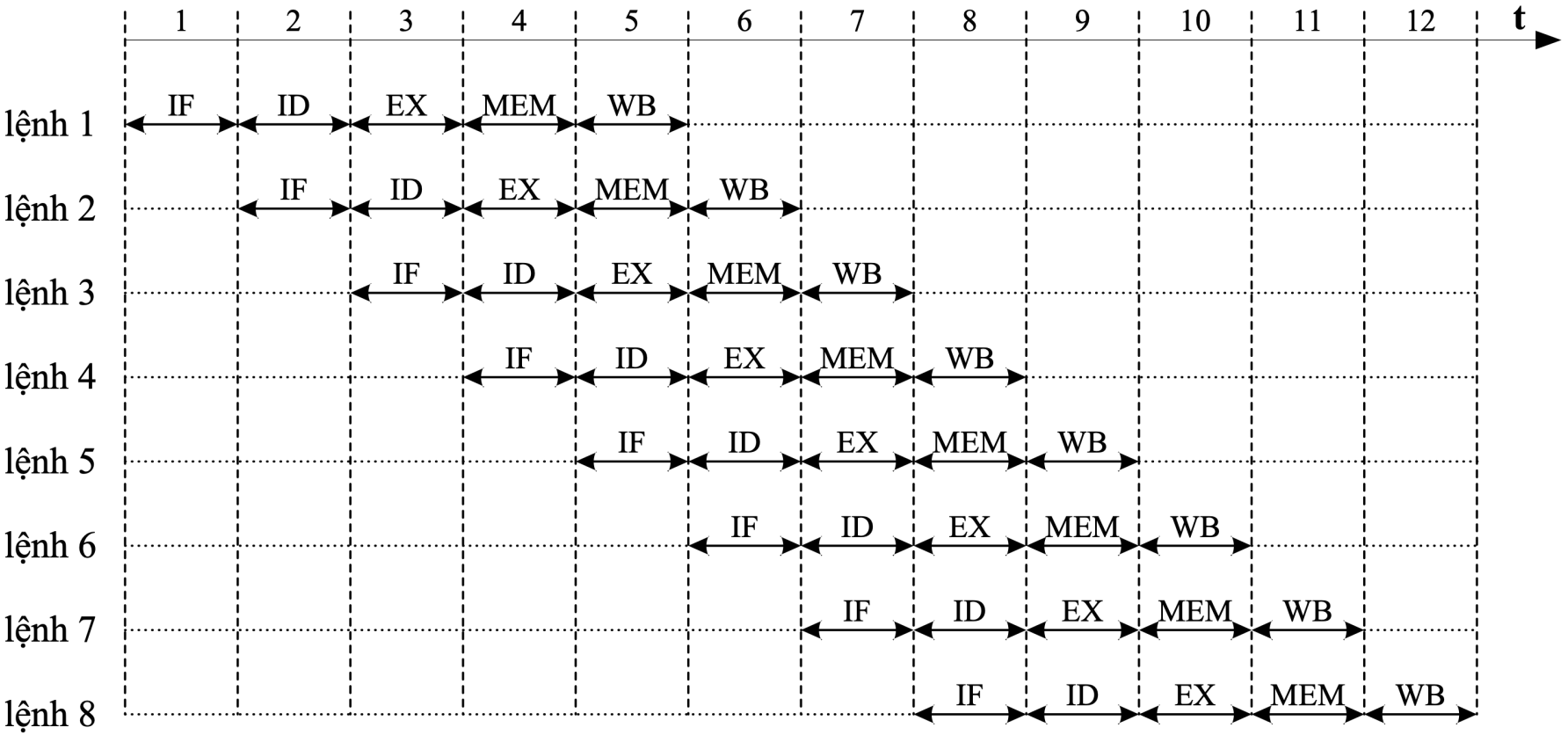
- Thống kê trong chương trình: 20% lệnh load, 8% lệnh store, 10% lệnh branch, 62% lệnh ALU
- CPI: các lệnh load: 5, các lệnh store: 4, các lệnh ALU: 4, các lệnh branch: 3
- CPI_{TB} = 0,2x5 + 0,08x4 + 0,1x3 + 0,62x4 = 4,10
- So với bộ xử lý đơn chu kỳ ?
  - Chu kỳ của bộ xử lý đơn chu kỳ = 5 chu kỳ của bộ xử lý đa chu kỳ (lệnh load) → CPI = 5

5.4. Bộ xử lý đường ống

- Kỹ thuật đường ống lệnh (Instruction Pipelining): Chia chu trình lệnh thành các công đoạn và cho phép thực hiện gối lên nhau (như dây chuyền lắp ráp)
- Bộ xử lý RISC-V có 5 công đoạn, mỗi công đoạn được mô tả liên quan với một thành phần chính của bộ xử lý:
  1. IF: Tìm nạp lệnh từ bộ nhớ (IM - Instruction Memory)
  2. ID: Giải mã lệnh và đọc thanh ghi (RF - Register File)
  3. EX: Thực thi thao tác hoặc tính toán địa chỉ (ALU)
  4. MEM: Truy nhập toán hạng bộ nhớ (DM – Data Memory)
  5. WB: Ghi kết quả trả về thanh ghi (RF - Register File)



Biểu đồ thời gian của đường ống lệnh



Thời gian thực hiện 1 công đoạn = T

Thời gian thực hiện tuần tự 8 lệnh: 8 x 5T = 40T

Thời gian thực hiện đường ống 8 lệnh: (1 x 5T) + [(8-1) x T] = 12T

Các trở ngại (Hazard) của đường ống lệnh

- Hazard: Tình huống ngăn cản bắt đầu của lệnh tiếp theo ở chu kỳ tiếp theo
  - Hazard cấu trúc: do tài nguyên được yêu cầu đang bận
  - Hazard dữ liệu: cần phải đợi để lệnh trước hoàn thành việc đọc/ghi dữ liệu
  - Hazard điều khiển: do rẽ nhánh gây ra

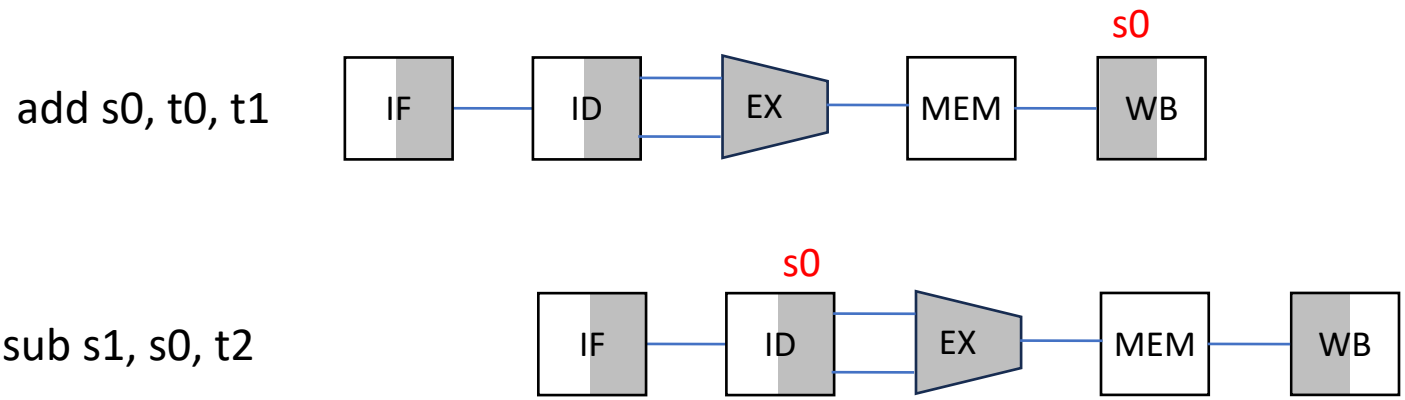
Hazard cấu trúc

- Xung đột khi sử dụng tài nguyên
- Giả sử trong đường ống của RISC-V với một bộ nhớ dùng chung
  - Lệnh Load/store yêu cầu truy cập dữ liệu
  - Tìm nạp lệnh cần trì hoãn cho chu kỳ đó
- Bởi vậy, datapath kiểu đường ống yêu cầu bộ nhớ lệnh và bộ nhớ dữ liệu tách rời (hoặc cache lệnh/cache dữ liệu tách rời)

Hazard dữ liệu

- Lệnh phụ thuộc vào việc hoàn thành truy cập dữ liệu của lệnh trước đó

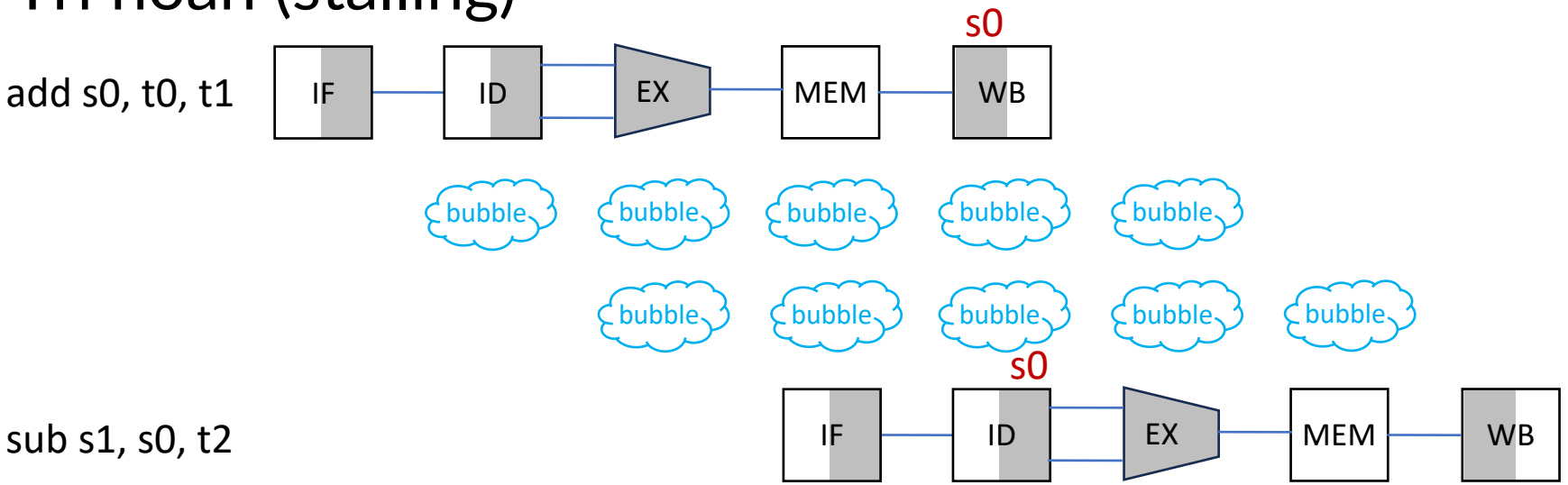
```
add  s0, t0, t1  # s0 = t0 + t1
sub  s1, s0, t2  # s1 = s0 - t2
```



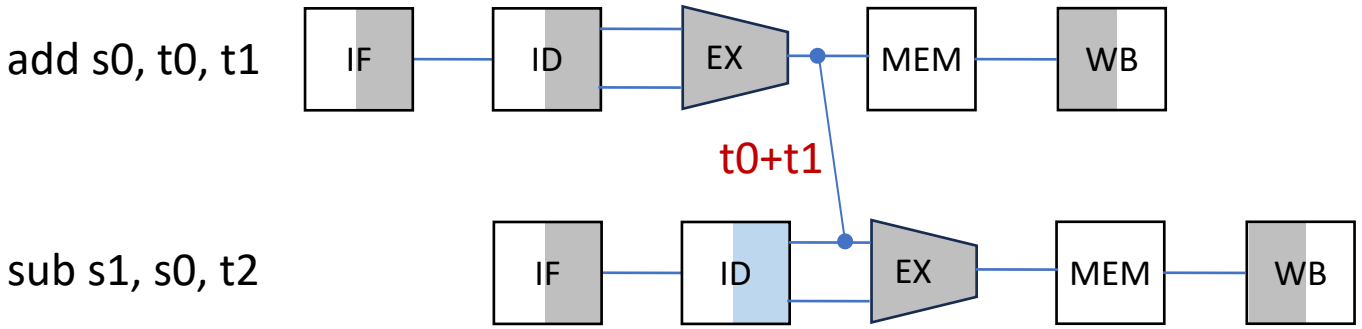
lệnh sub yêu cầu đọc giá trị ở s0 trước khi lệnh add ghi kết quả đến s0 !

Trì hoãn và Chuyển tiếp trước

- Trì hoãn (stalling)

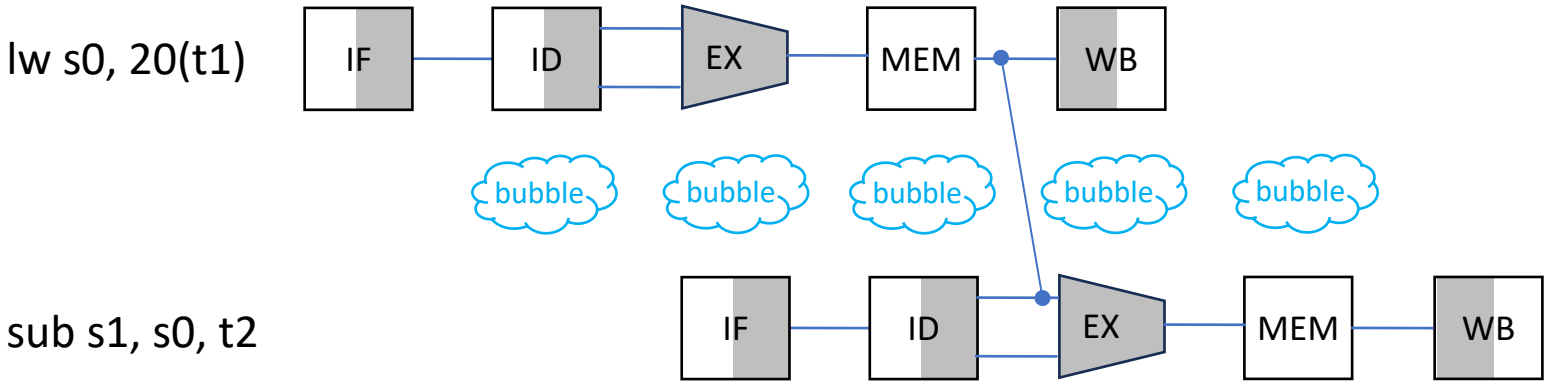


- Chuyển tiếp trước (forwarding)



Hazard dữ liệu với lệnh load

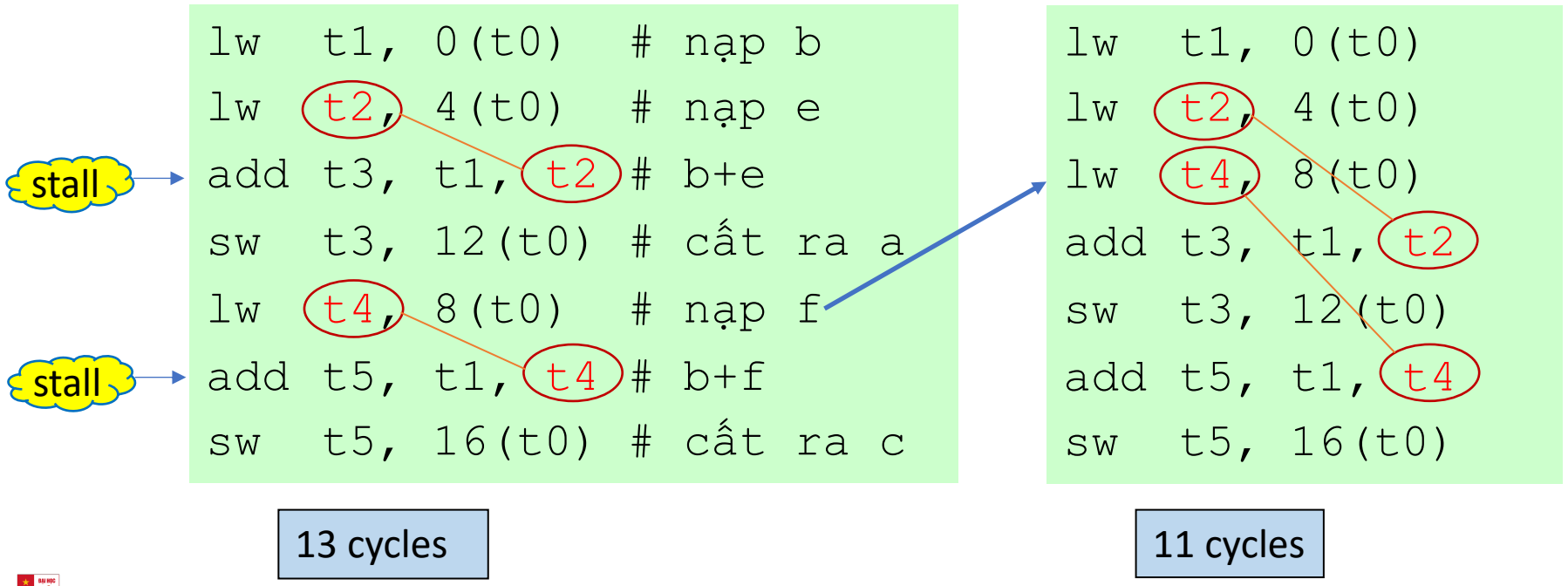
- Có sử dụng forwarding
- Cần chèn 1 bước trì hoãn



Lập lịch mã để tránh trì hoãn

- Thay đổi trình tự mã để tránh sử dụng kết quả load ở lệnh tiếp theo
- Mã C:

```
a = b + e;
c = b + f;
```

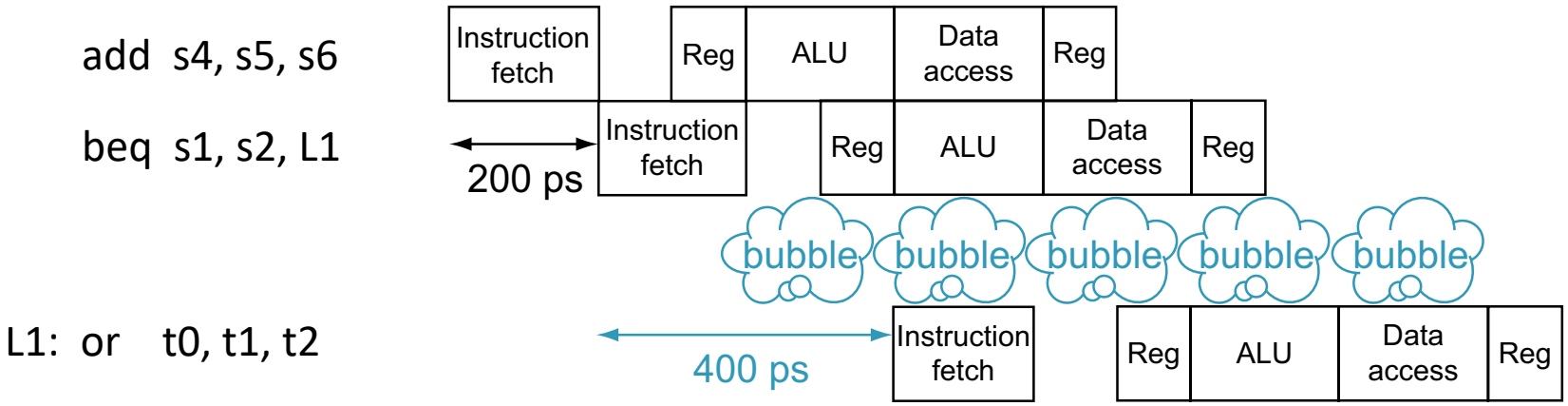


Hazard điều khiển

- Rẽ nhánh xác định luồng điều khiển
  - Tìm nạp lệnh tiếp theo phụ thuộc vào kết quả rẽ nhánh
  - Đường ống không thể luôn nhận đúng lệnh
    - Vẫn đang làm ở công đoạn giải mã lệnh (ID) của lệnh rẽ nhánh
- Trong đường ống của RISC-V
  - Cần so sánh thanh ghi và tính địa chỉ đích sớm trong đường ống
  - Thêm phần cứng để thực hiện việc đó trong công đoạn ID

Trì hoãn khi rẽ nhánh

- Đợi cho đến khi kết quả rẽ nhánh đã được xác định trước khi tìm nạp lệnh tiếp theo



Dự đoán rẽ nhánh

- Những đường ống dài hơn không thể sớm xác định dễ dàng kết quả rẽ nhánh
  - Cách trì hoãn không đáp ứng được
- Dự đoán kết quả rẽ nhánh
  - Chỉ trì hoãn khi dự đoán là sai
- Với RISC-V
  - Có thể dự đoán rẽ nhánh không xẩy ra
  - Tìm nạp lệnh ngay sau lệnh rẽ nhánh (không làm trễ)

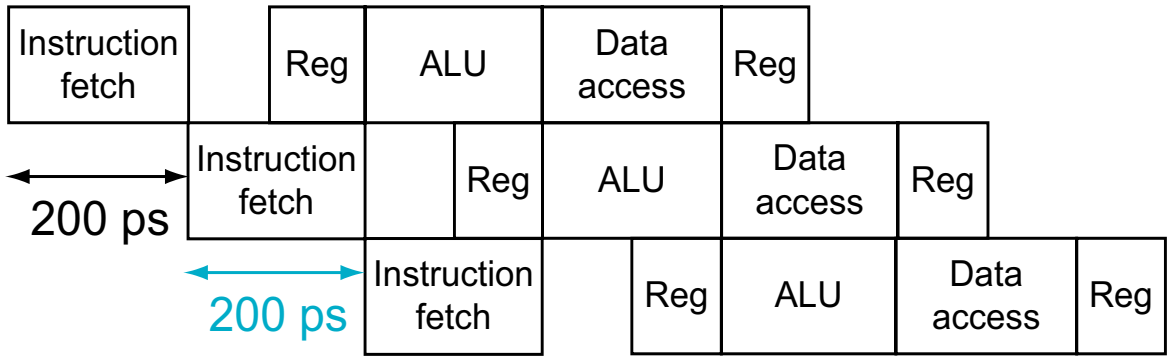
RISC-V với dự đoán rẽ nhánh không xẩy ra

Dự đoán đúng

add s4, s5, s6

beq s1, s2, L1

lw s3, 0(s0)

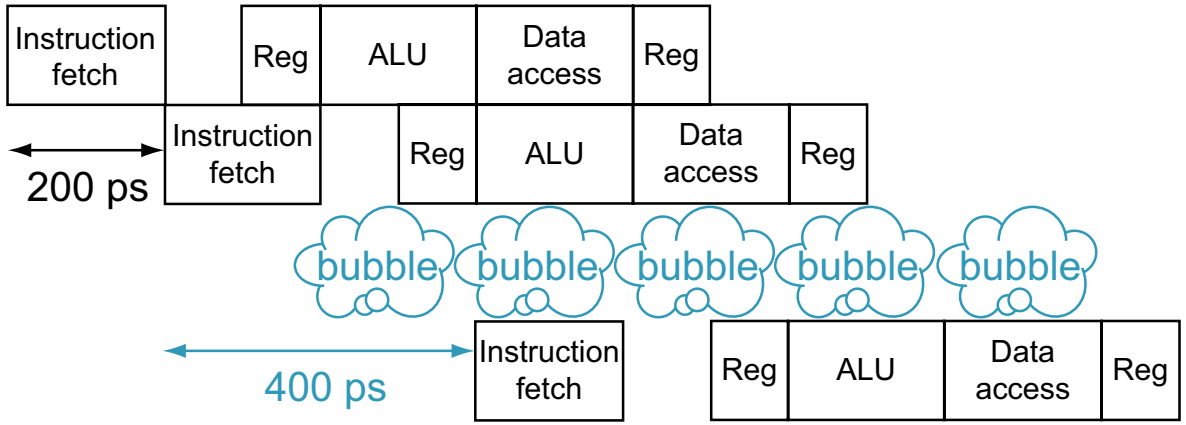


Dự đoán sai

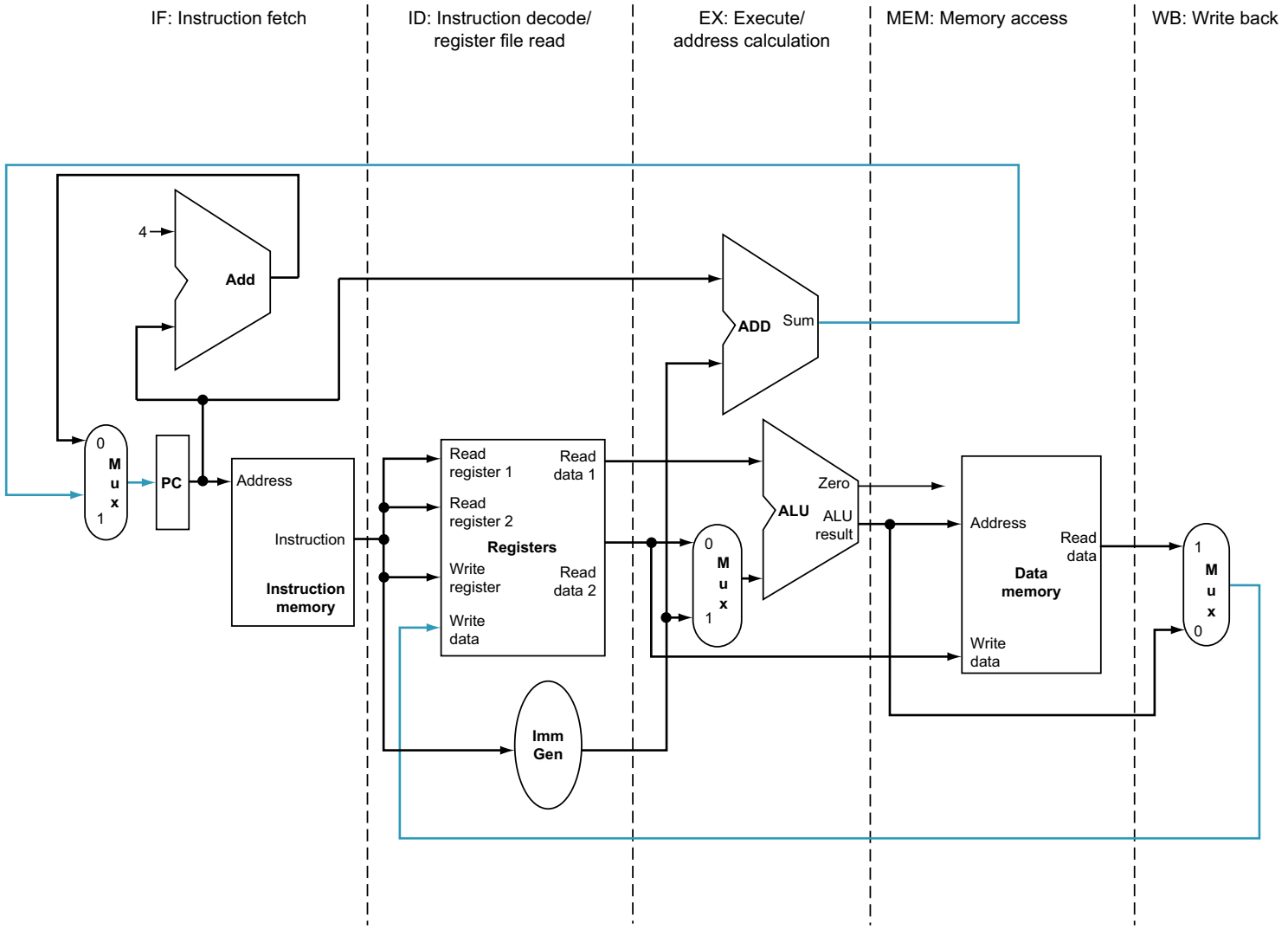
add s4, s5, s6

beq s1, s2, L1

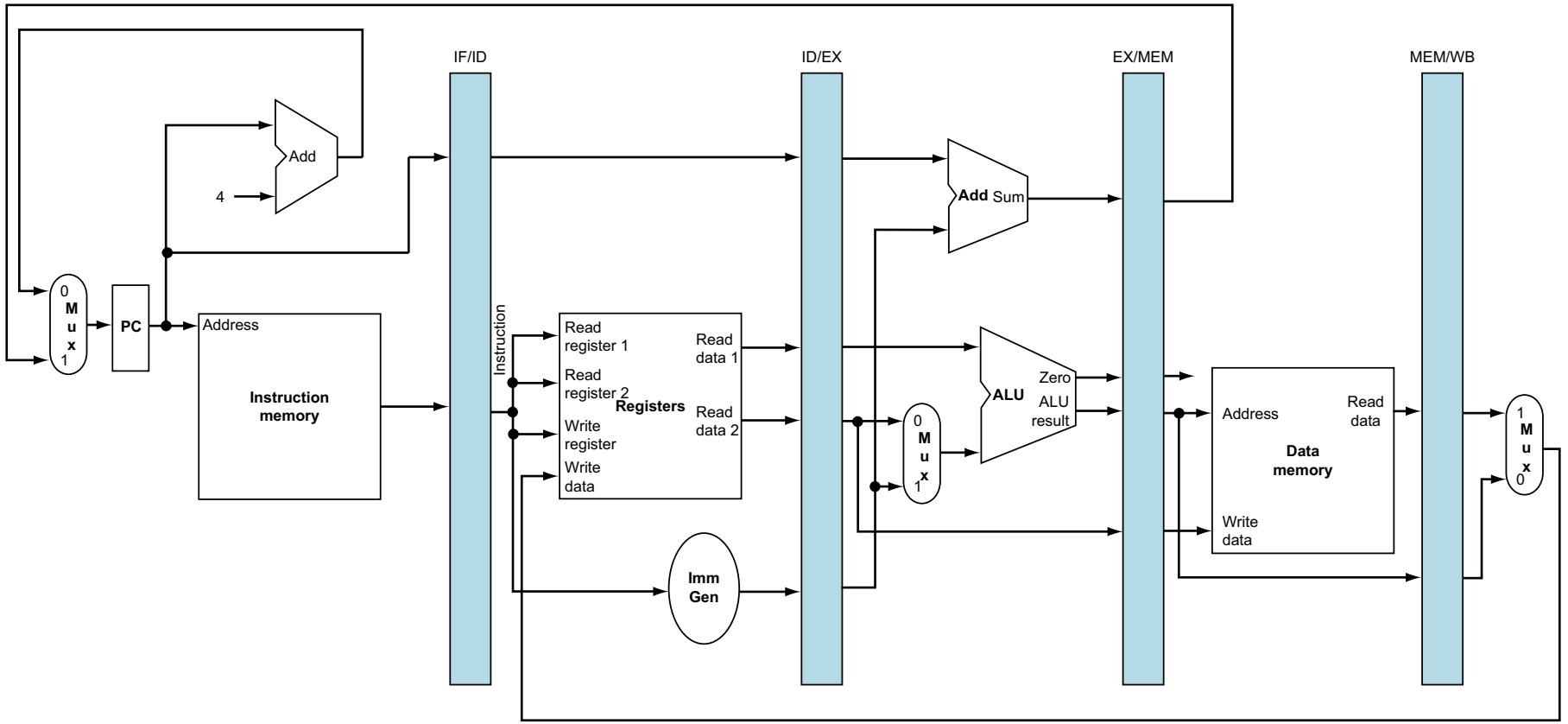
L1: or t0, t1, t2

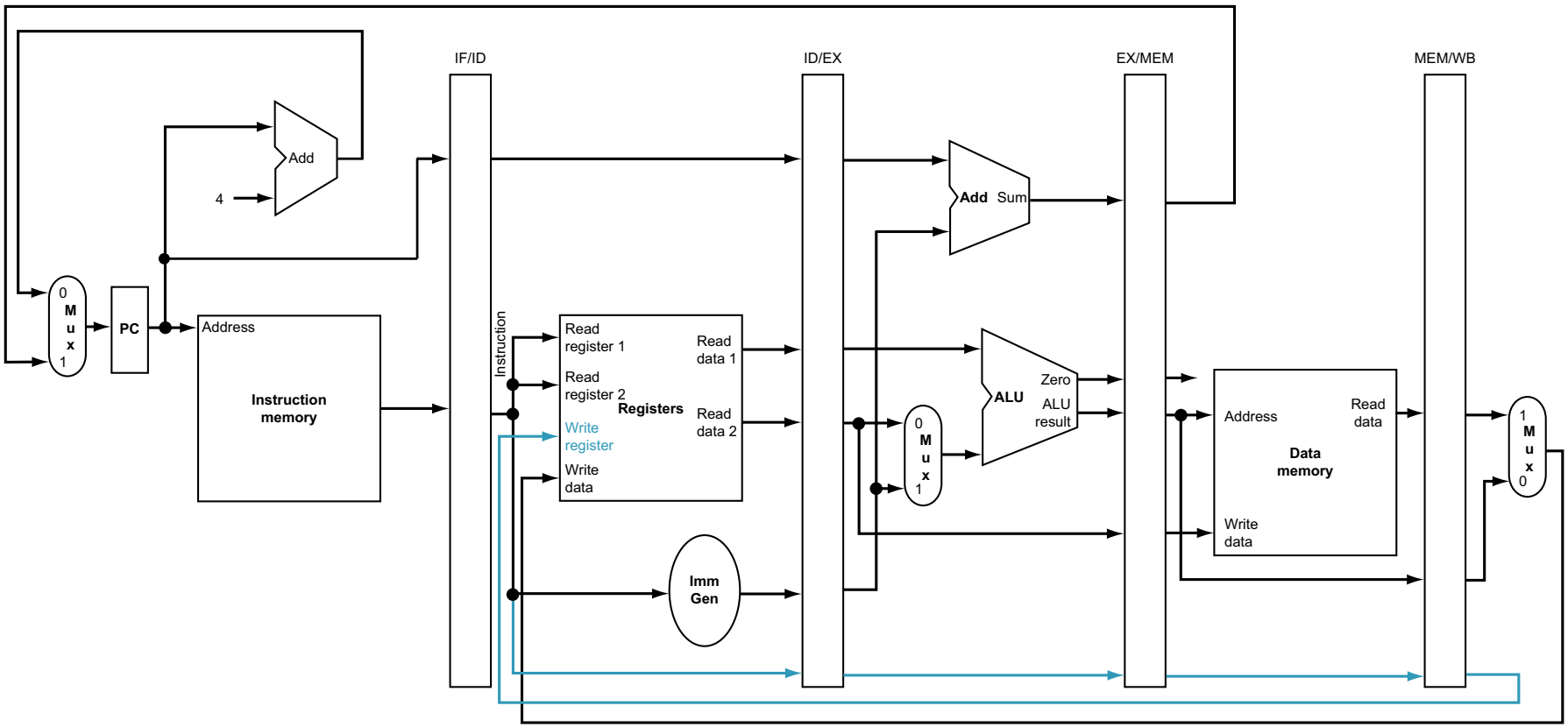


5 công đoạn của Pipelined Datapath



Pipelined Datapath



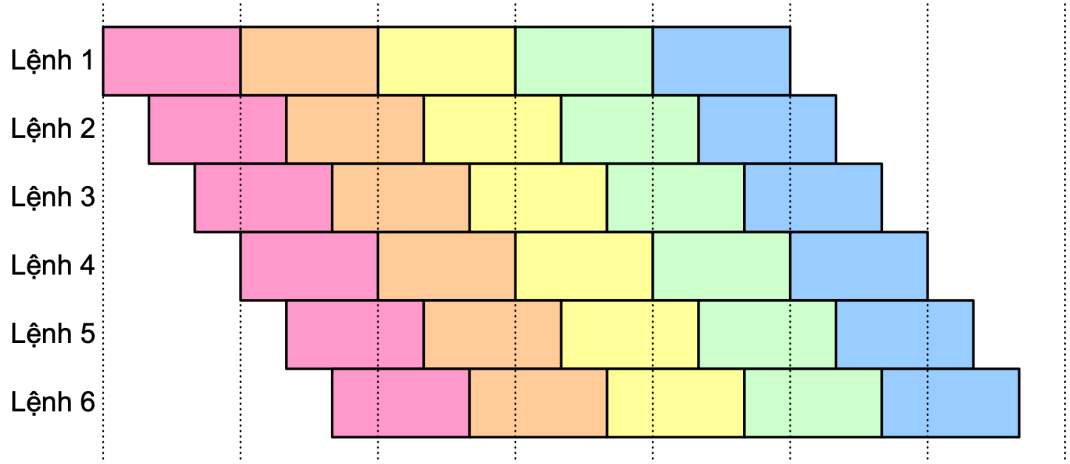

IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
0
M
u
x
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Imm Gen
Add Sum
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
1
M
u
x
0

Đặc điểm của đường ống

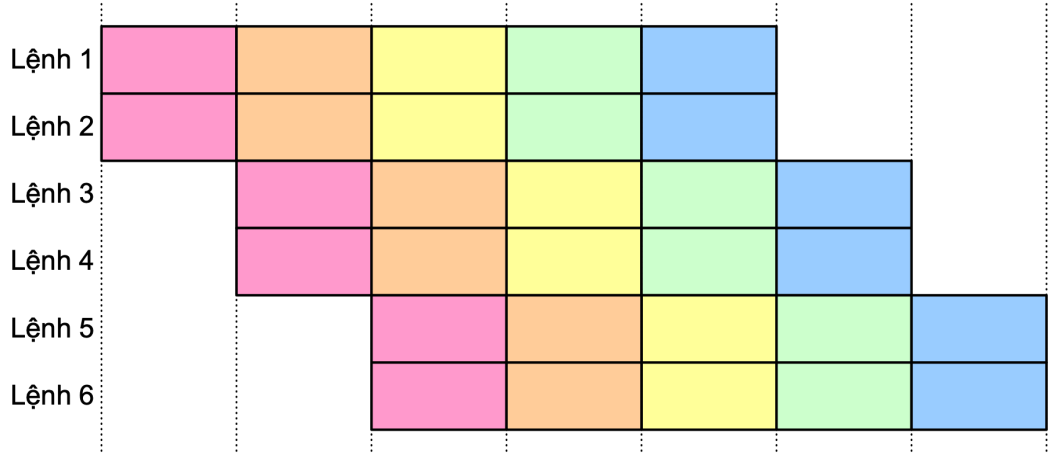
- Kỹ thuật đường ống cải thiện hiệu năng bằng cách tăng số lệnh thực hiện
  - Thực hiện nhiều lệnh đồng thời
  - Mỗi lệnh có cùng thời gian thực hiện
- Các dạng hazard:
  - Cấu trúc, dữ liệu, điều khiển
- Thiết kế tập lệnh ảnh hưởng đến độ phức tạp của việc thực hiện đường ống

Tăng cường khả năng song song mức lệnh

- Tăng số công đoạn của đường ống



- Siêu vô hướng (Superscalar)



Hết chương 5