



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

KIẾN TRÚC MÁY TÍNH

Computer Architecture

Course ID: IT3030

Nguyễn Kim Khánh

Nội dung học phần

Chương 1. Giới thiệu chung

Chương 2. Hệ thống máy tính

Chương 3. Số học và logic máy tính

Chương 4. Kiến trúc tập lệnh

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 của chương 5

- 5.1. Tổ chức của CPU
- 5.2. Thiết kế bộ xử lý theo kiến trúc MIPS
- 5.3. Kỹ thuật đường ống lệnh và song song mức lệnh



5.1. Tổ chức của CPU

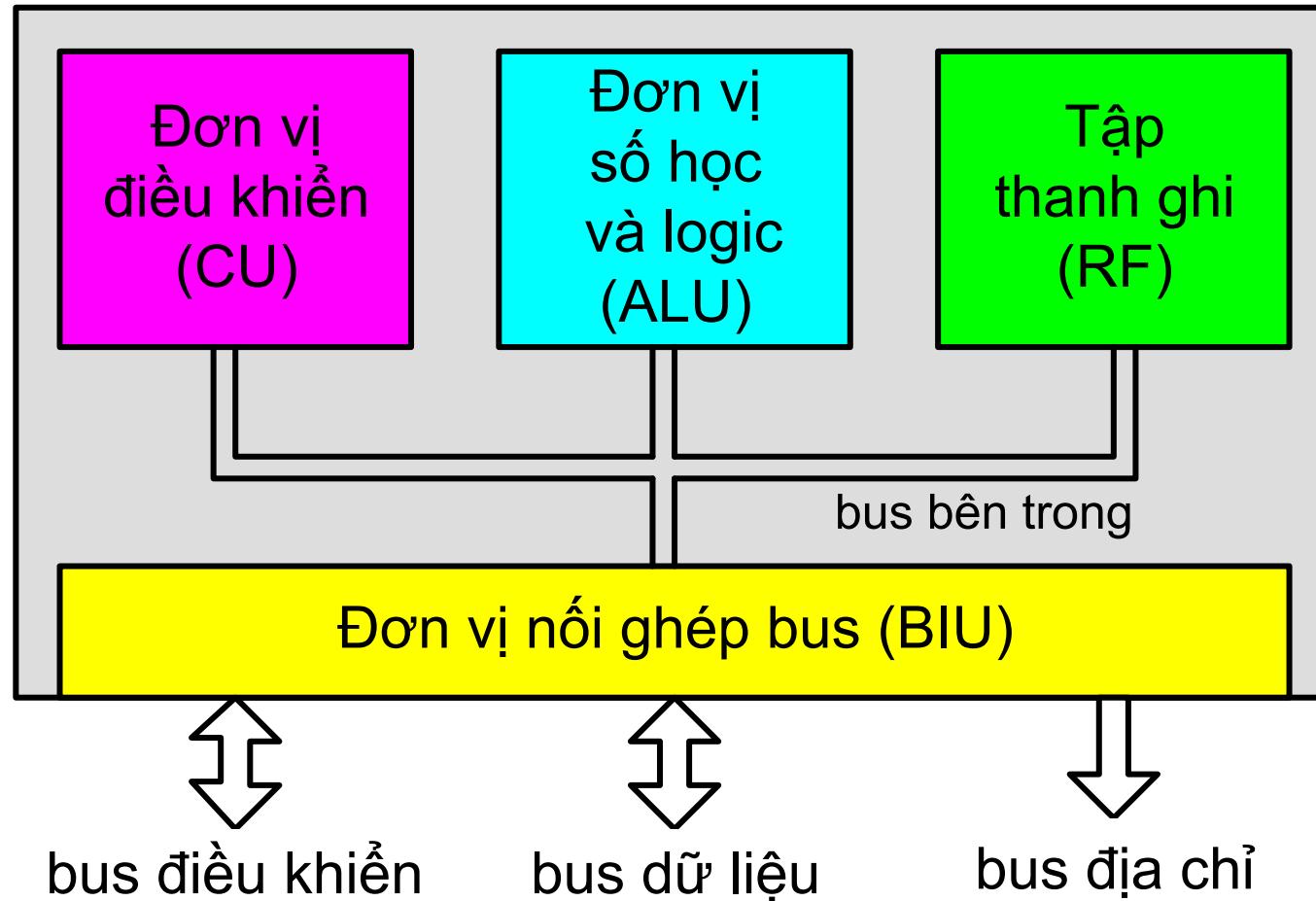
1. Cấu trúc cơ bản của CPU

▪ Nhiệm vụ của CPU:

- Nhận lệnh (Fetch Instruction): CPU đọc lệnh từ bộ nhớ
- Giải mã lệnh (Decode Instruction): xác định thao tác mà lệnh yêu cầu
- Nhận dữ liệu (Fetch Data): nhận dữ liệu từ bộ nhớ hoặc các cổng vào-ra
- Xử lý dữ liệu (Process Data): thực hiện phép toán số học hay phép toán logic với các dữ liệu
- Ghi dữ liệu (Write Data): ghi dữ liệu ra bộ nhớ hay cổng vào-ra



Sơ đồ cấu trúc cơ bản của CPU

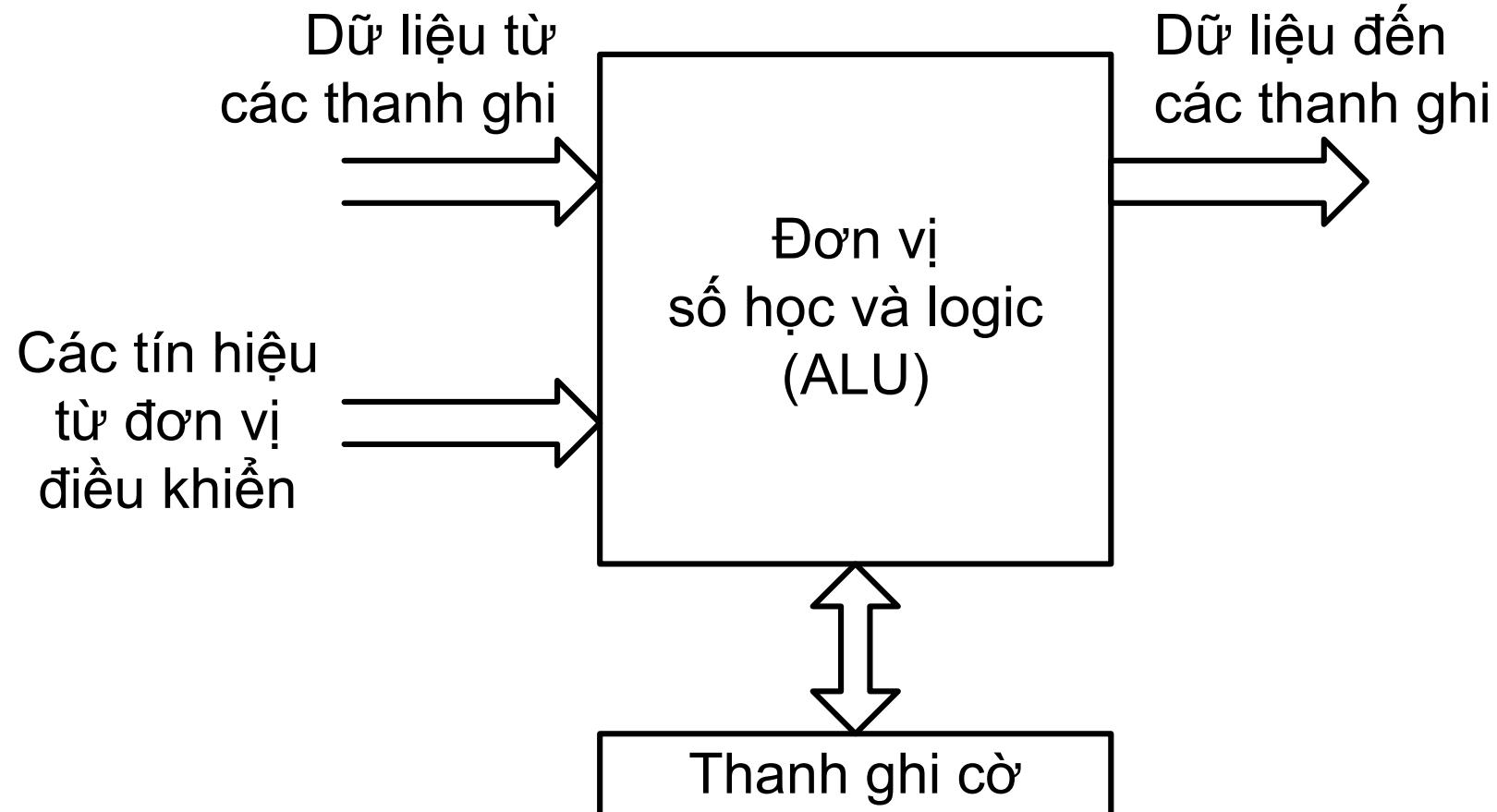


2. Đơn vị số học và logic

- **Chức năng:** Thực hiện các phép toán số học và phép toán logic:
 - Số học: cộng, trừ, nhân, chia, đảo dấu
 - Logic: AND, OR, XOR, NOT, phép dịch bit



Mô hình kết nối ALU



Thanh ghi cờ: hiển thị trạng thái của kết quả phép toán



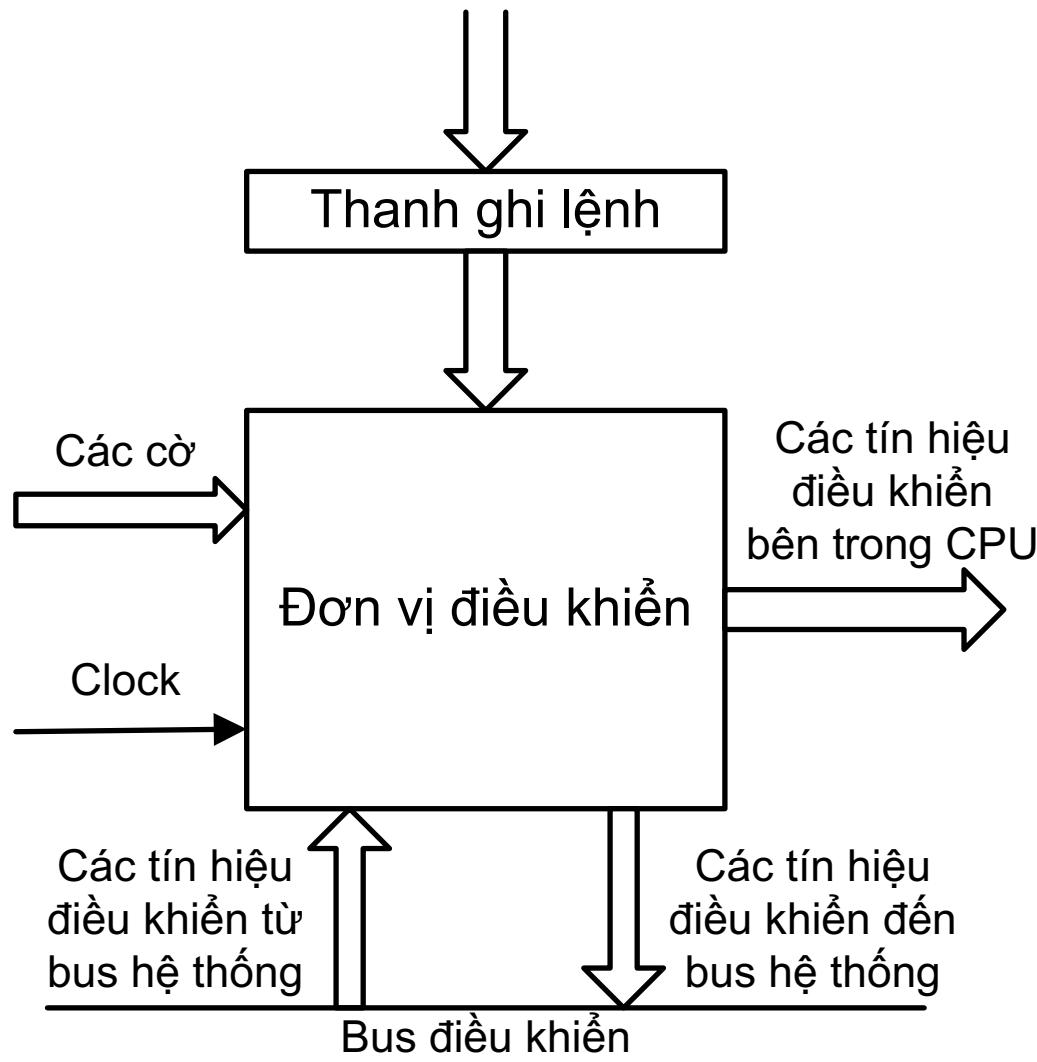
3. Đơn vị điều khiển

▪ Chức năng

- Điều khiển nhận lệnh từ bộ nhớ đưa vào CPU
- Tăng nội dung của PC để trỏ sang lệnh kế tiếp
- Giải mã lệnh đã được nhận để xác định thao tác mà lệnh yêu cầu
- Phát ra các tín hiệu điều khiển thực hiện lệnh
- Nhận các tín hiệu yêu cầu từ bus hệ thống và đáp ứng với các yêu cầu đó.



Mô hình kết nối đơn vị điều khiển



Các tín hiệu đưa đến đơn vị điều khiển

- Clock: tín hiệu nhịp từ mạch tạo dao động bên ngoài
- Lệnh máy từ thanh ghi lệnh đưa đến để giải mã
- Các cờ từ thanh ghi cờ cho biết trạng thái của CPU
- Các tín hiệu yêu cầu từ bus điều khiển



Các tín hiệu phát ra từ đơn vị điều khiển

- Các tín hiệu điều khiển bên trong CPU:
 - Điều khiển các thanh ghi
 - Điều khiển ALU
- Các tín hiệu điều khiển bên ngoài CPU:
 - Điều khiển bộ nhớ
 - Điều khiển các mô-đun vào-ra



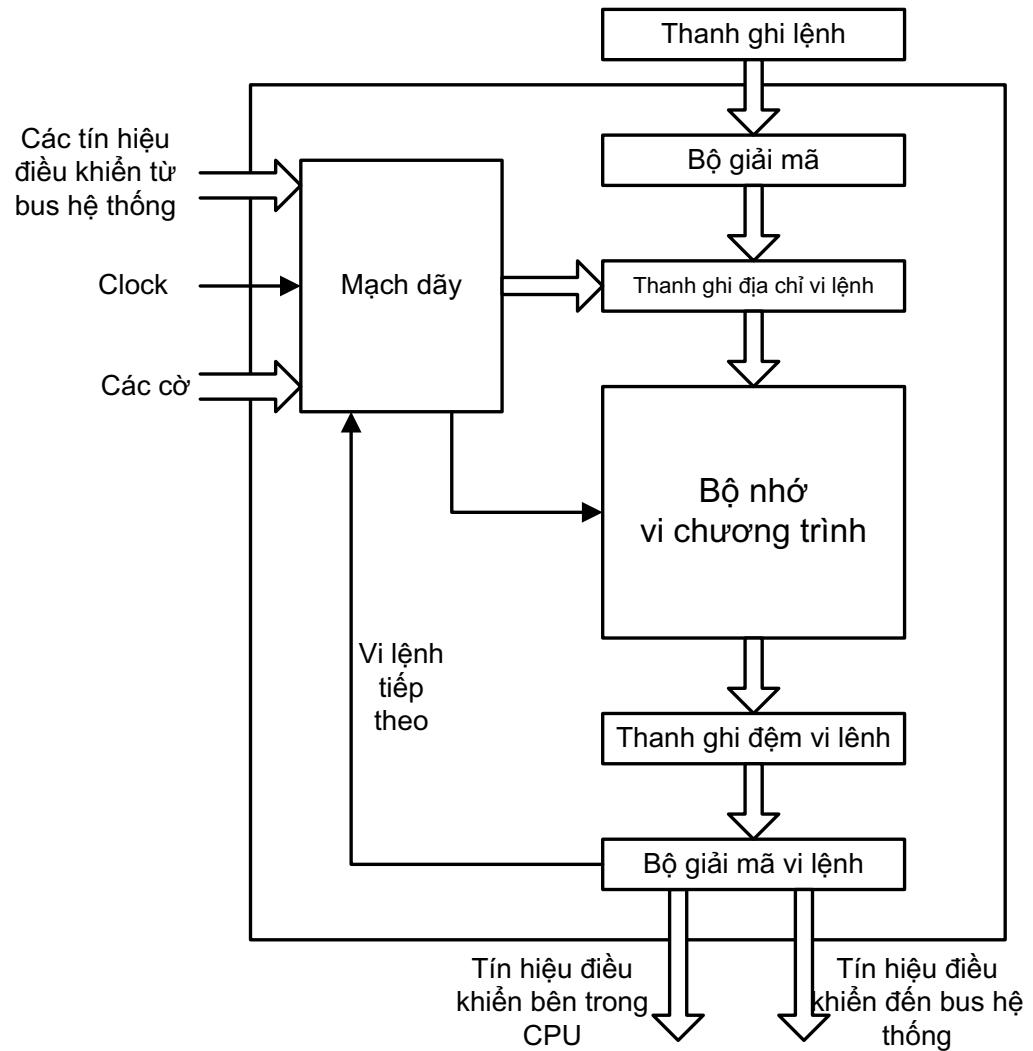
Các phương pháp thiết kế đơn vị điều khiển

- Đơn vị điều khiển vi chương trình (Microprogrammed Control Unit)
- Đơn vị điều khiển nối kết cứng (Hardwired Control Unit)



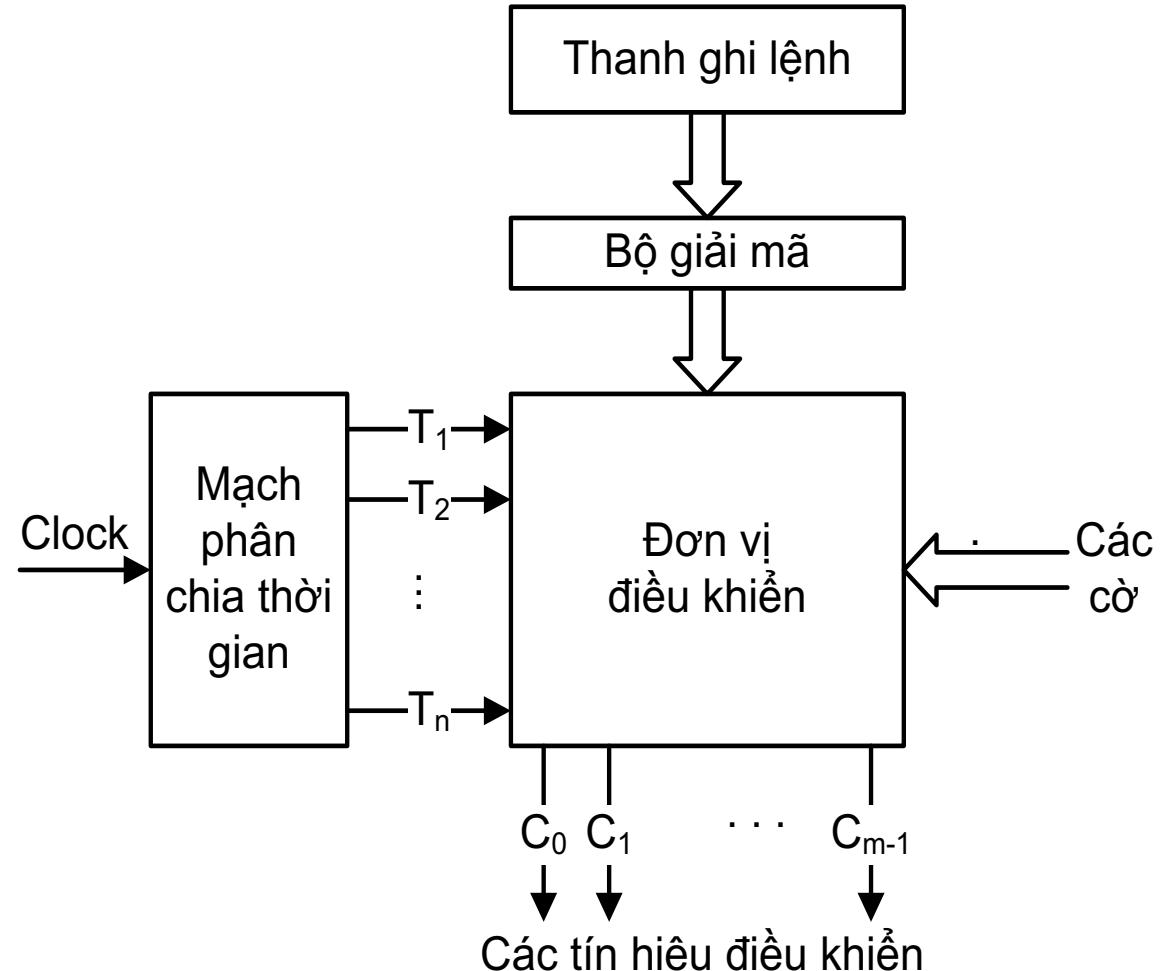
Đơn vị điều khiển vi chương trình

- Bộ nhớ vi chương trình (ROM) lưu trữ các vi chương trình (microprogram)
- Một vi chương trình bao gồm các vi lệnh (microinstruction)
- Mỗi vi lệnh mã hóa cho một vi thao tác (microoperation)
- Để hoàn thành một lệnh cần thực hiện một hoặc một vài vi chương trình
- Tốc độ chậm



Đơn vị điều khiển nối kết cứng

- Sử dụng mạch cứng để giải mã và tạo các tín hiệu điều khiển thực hiện lệnh
- Tốc độ nhanh
- Đơn vị điều khiển phức tạp



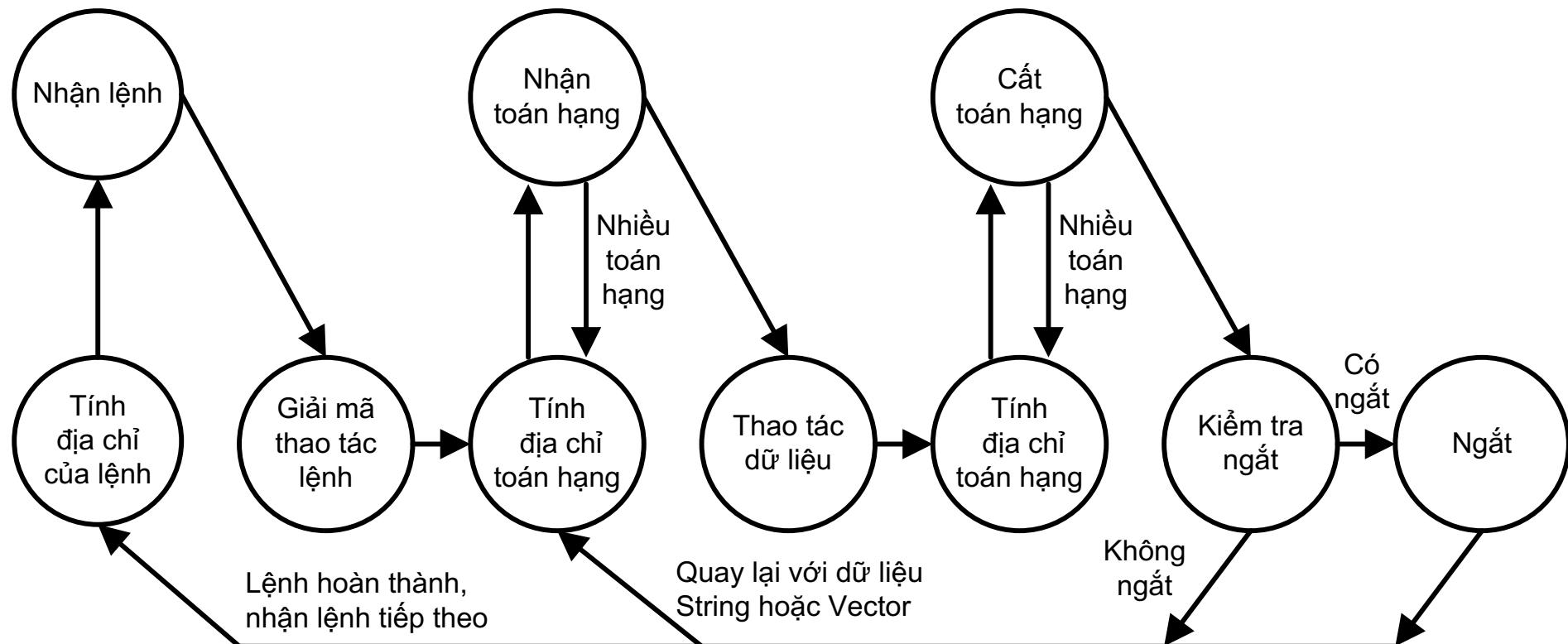
4. Hoạt động của chu trình lệnh

Chu trình lệnh

- Nhận lệnh
- Giải mã lệnh
- Nhận toán hạng
- Thực hiện lệnh
- Cất toán hạng
- Ngắt



Giản đồ trạng thái chu trình lệnh



Tính địa chỉ của lệnh

- PC chứa địa chỉ của lệnh sẽ được nhận vào
- Với MIPS:
 - Tuần tự: $PC = PC + 4$
 - Rẽ nhánh (đk đúng): $PC = PC + imm \times 4$
 - Chú ý: PC đã được tăng thêm 4 từ trước đó
 - Nhảy: $PC = PC_{31-28}$: (26bit địa chỉ) : 00

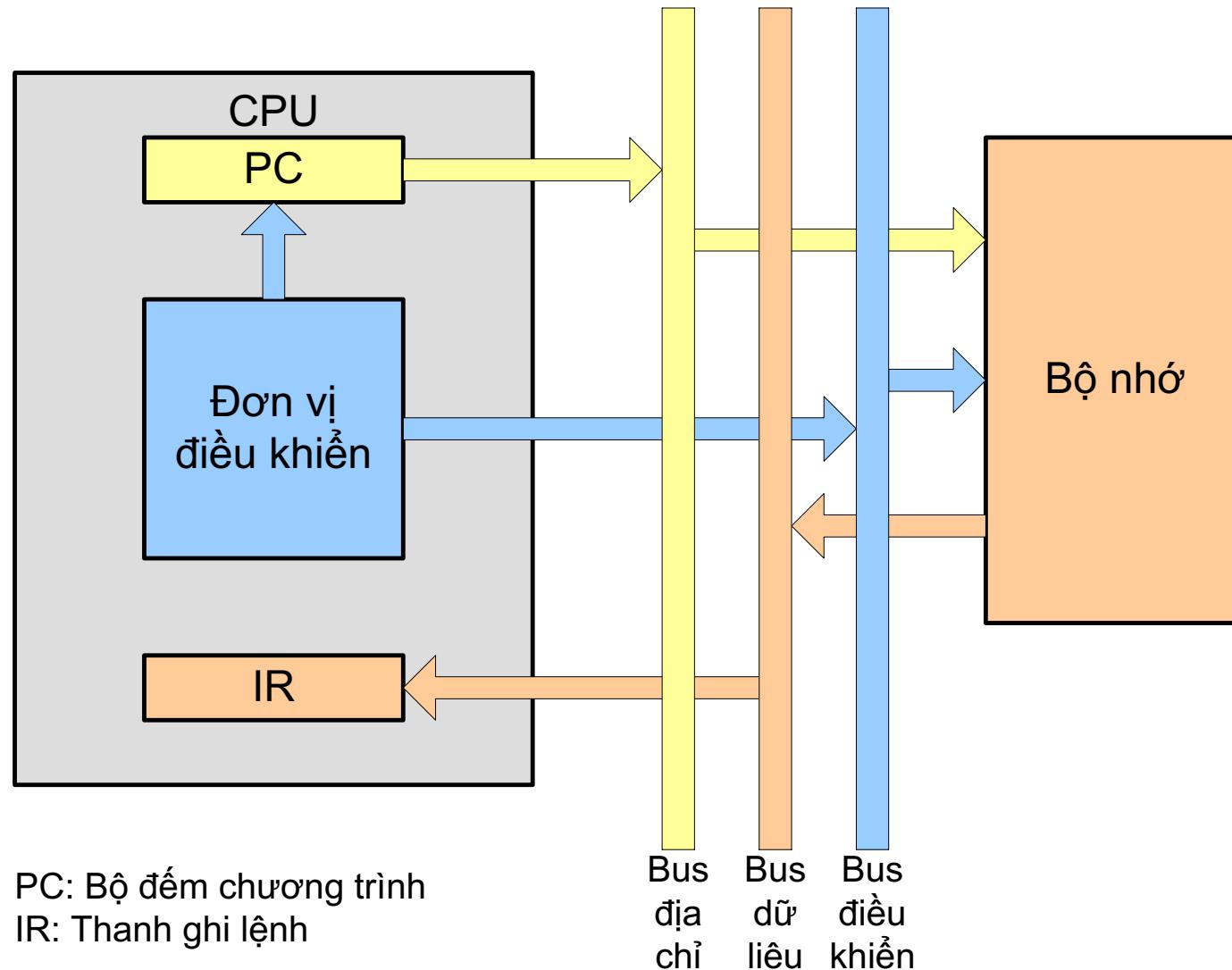


Nhận lệnh

- CPU đưa địa chỉ của lệnh cần nhận từ bộ đếm chương trình PC ra bus địa chỉ
- CPU phát tín hiệu điều khiển đọc bộ nhớ
- Lệnh từ bộ nhớ được đặt lên bus dữ liệu và được CPU copy vào thanh ghi lệnh IR
- CPU tăng nội dung PC để trở sang lệnh kế tiếp



Sơ đồ mô tả quá trình nhận lệnh



Giải mã lệnh

- Lệnh từ thanh ghi lệnh IR được đưa đến đơn vị điều khiển
- Đơn vị điều khiển tiến hành giải mã lệnh để xác định thao tác phải thực hiện
- Giải mã lệnh xảy ra bên trong CPU



Tính địa chỉ toán hạng

- Với MIPS: lệnh lw/ lh/ lb

`lw rt, imm(rs) #(rt) = mem[(rs) + imm]`

Địa chỉ toán hạng = (rs) + imm

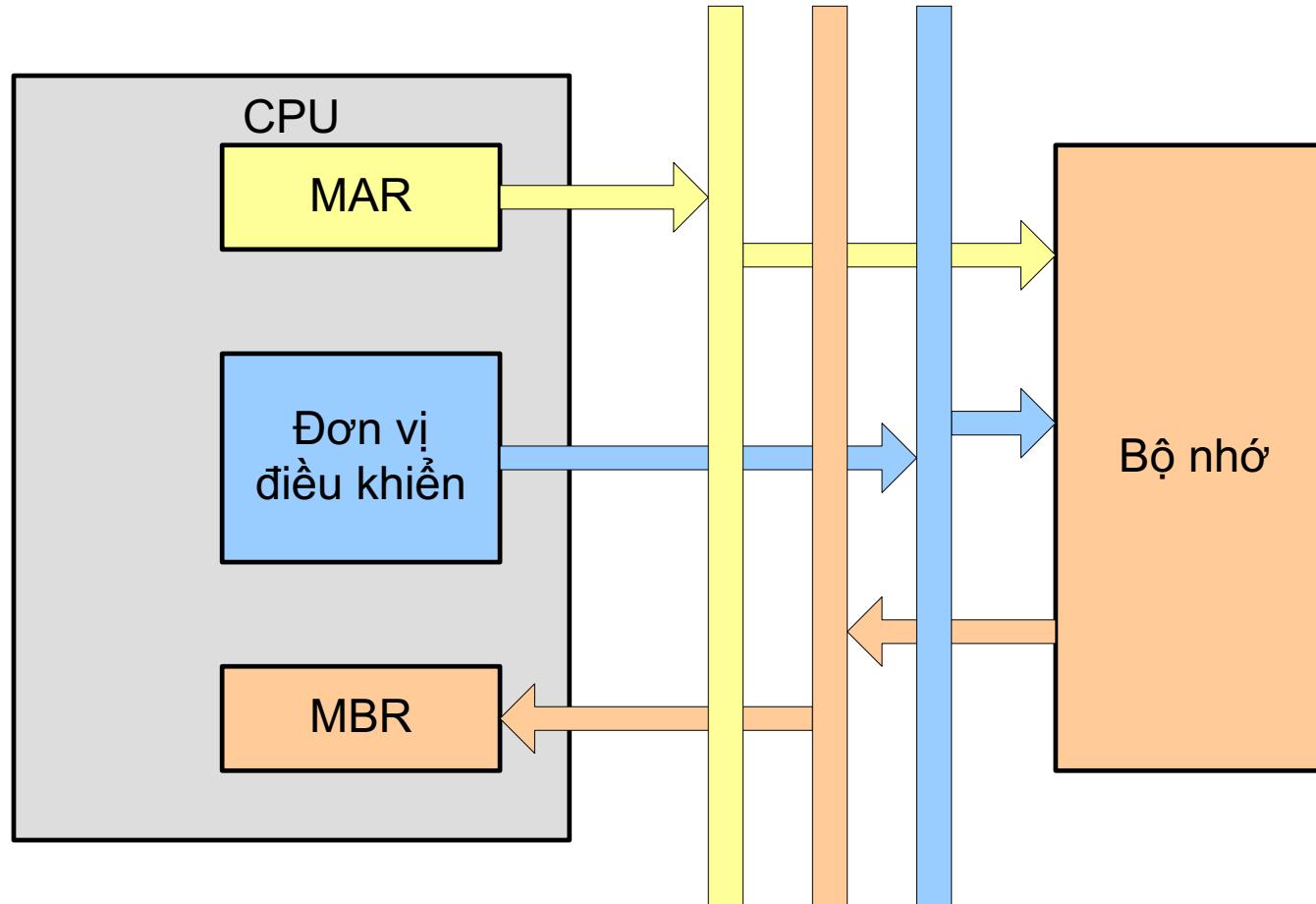


Nhận dữ liệu từ bộ nhớ

- CPU đưa địa chỉ của toán hạng ra bus địa chỉ
- CPU phát tín hiệu điều khiển đọc
- Toán hạng được đọc vào CPU
- Tương tự như nhận lệnh



Sơ đồ mô tả nhận dữ liệu từ bộ nhớ



MAR: Thanh ghi địa chỉ bộ nhớ
MBR: Thanh ghi đệm bộ nhớ

Bus
địa
chi
Bus
dữ
liệu
Bus
điều
khiển

Thực hiện lệnh

- Có nhiều dạng tuỳ thuộc vào lệnh
- Có thể là:
 - Đọc/Ghi bộ nhớ
 - Vào/Ra
 - Chuyển giữa các thanh ghi
 - Phép toán số học/logic
 - Chuyển điều khiển (rẽ nhánh)
 - ...



Tính địa chỉ toán hạng

- Với MIPS: lệnh sw/ sh/ sb

sw rt, imm(rs) # mem[(rs)+imm] = rt

Địa chỉ toán hạng = (rs) + imm

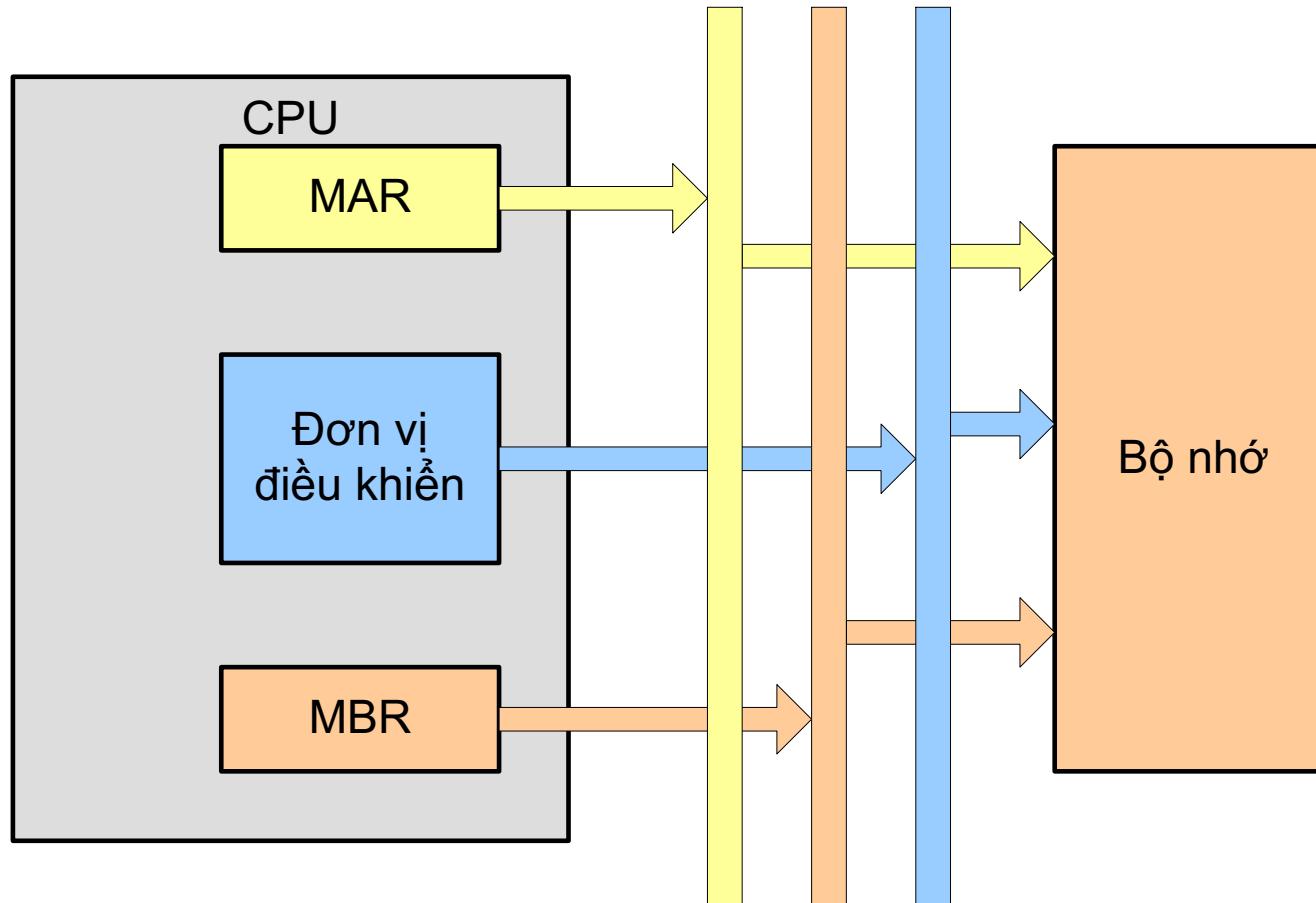


Ghi toán hạng

- CPU đưa địa chỉ ra bus địa chỉ
- CPU đưa dữ liệu cần ghi ra bus dữ liệu
- CPU phát tín hiệu điều khiển ghi
- Dữ liệu trên bus dữ liệu được copy đến vị trí xác định



Sơ đồ mô tả quá trình ghi toán hạng



MAR: Thanh ghi địa chỉ bộ nhớ
MBR: Thanh ghi đệm bộ nhớ

Bus
địa
chi
Bus
dữ
liệu
Bus
điều
khiển

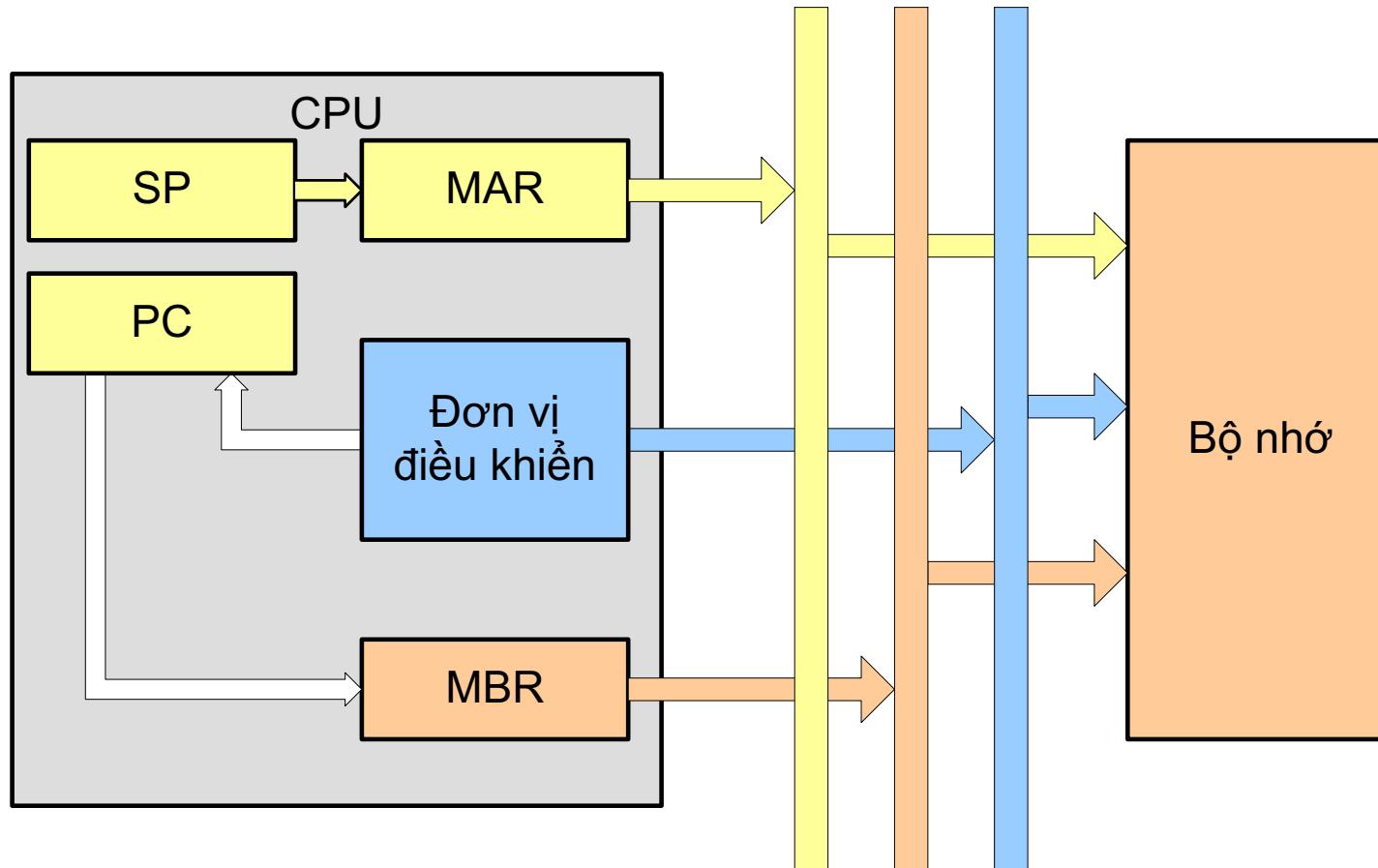


Ngắt

- Nội dung của bộ đếm chương trình PC (địa chỉ trả về sau khi ngắt) được đưa ra bus dữ liệu
- CPU đưa địa chỉ (thường được lấy từ con trỏ ngăn xếp SP) ra bus địa chỉ
- CPU phát tín hiệu điều khiển ghi bộ nhớ
- Địa chỉ trả về trên bus dữ liệu được ghi ra vị trí xác định (ở ngăn xếp)
- Địa chỉ lệnh đầu tiên của chương trình con điều khiển ngắt được nạp vào PC



Sơ đồ mô tả chu trình ngắn



MAR: Thanh ghi địa chỉ bộ nhớ

MBR: Thanh ghi đệm bộ nhớ

PC: Bộ đếm chương trình

SP: Con trỏ ngăn xếp

Bus
địa
chi

Bus
dữ
liệu

Bus
điều
khiển

5.2. Thiết kế bộ xử lý theo kiến trúc MIPS

- Chỉ thực hiện với một số lệnh cơ bản của MIPS, nhưng chỉ ra hầu hết các khía cạnh:
 - Các lệnh tham chiếu bộ nhớ: lw, sw
 - Các lệnh số học/logic: add, sub, and, or, slt
 - Các lệnh chuyển điều khiển: beq, j



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 và nhận 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
 - Các lệnh khác (không kể lệnh jump): 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 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

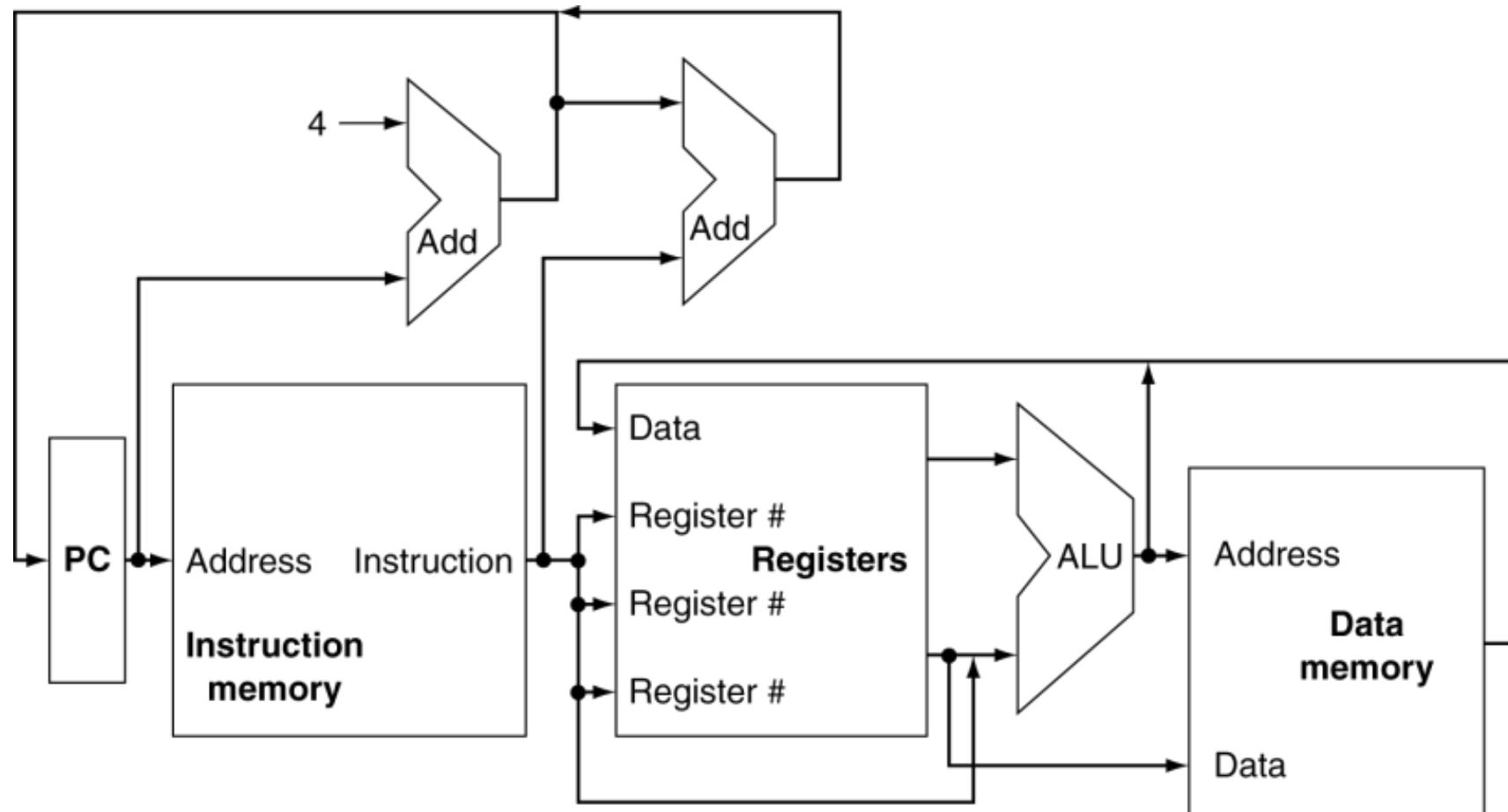


Tổng quan quá trình thực hiện các lệnh (tiếp)

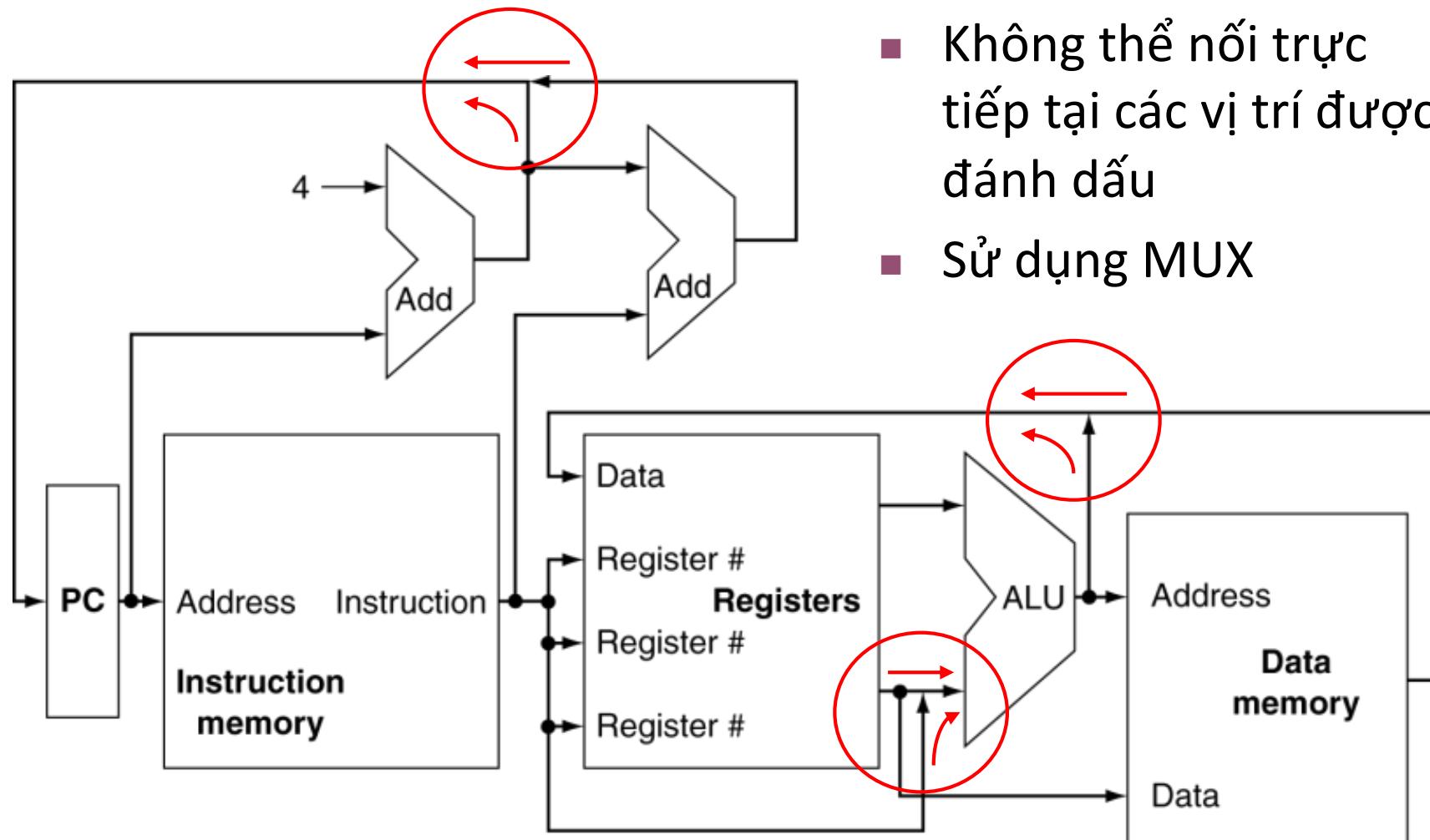
- Thay đổi nội dung bộ đếm chương trình PC:
 - Với các lệnh rẽ nhánh (branch), tùy thuộc vào kết quả so sánh:
 - Điều kiện thỏa mãn: $PC \leftarrow$ địa chỉ đích (địa chỉ của lệnh cần rẽ tới)
 - Điều kiện không thỏa mãn: $PC \leftarrow PC + 4$ (địa chỉ của lệnh kế tiếp)
 - Với các lệnh còn lại (không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



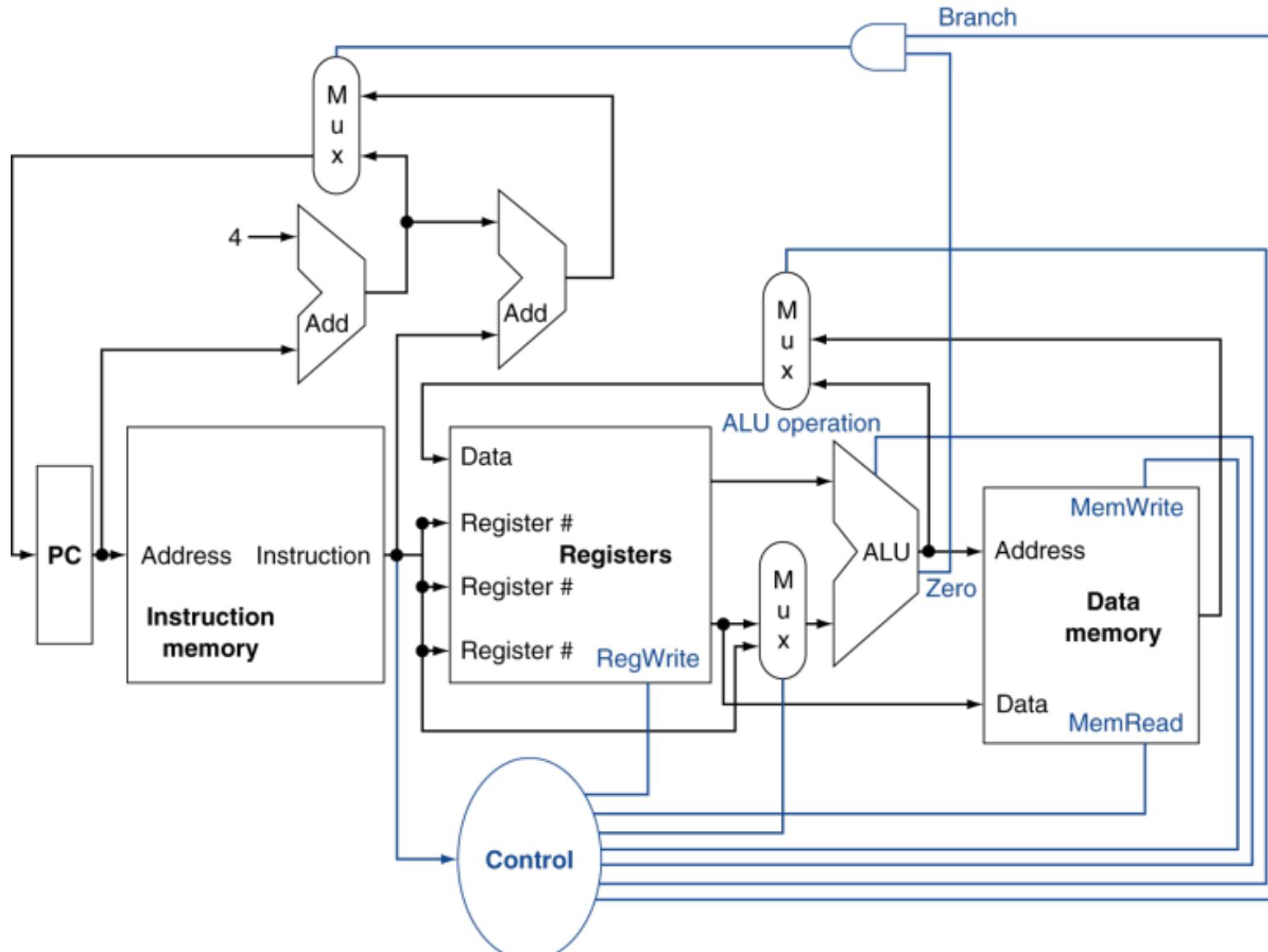
Sơ đồ khái quát của bộ xử lý MIPS



Sử dụng bộ chọn kênh (MUX)



Bộ xử lý với các đường điều khiển chính

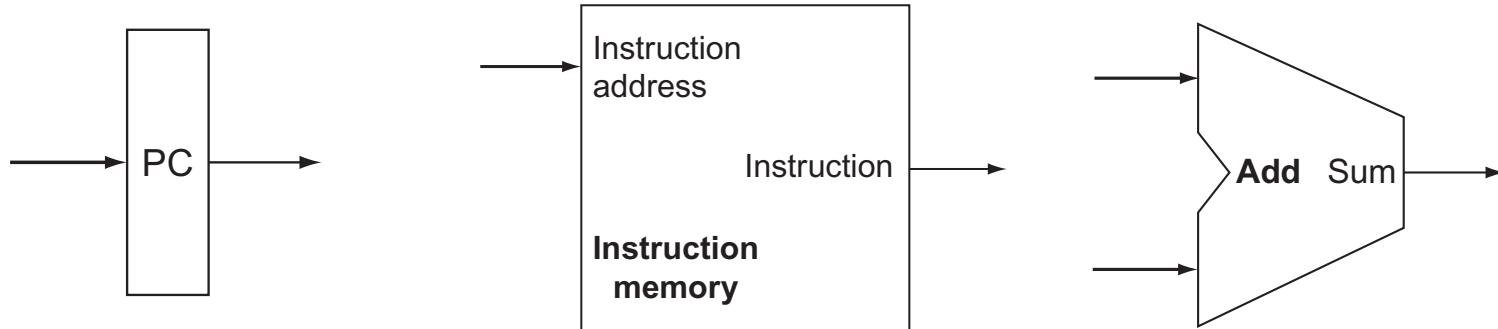


1. Thiết kế Datapath

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

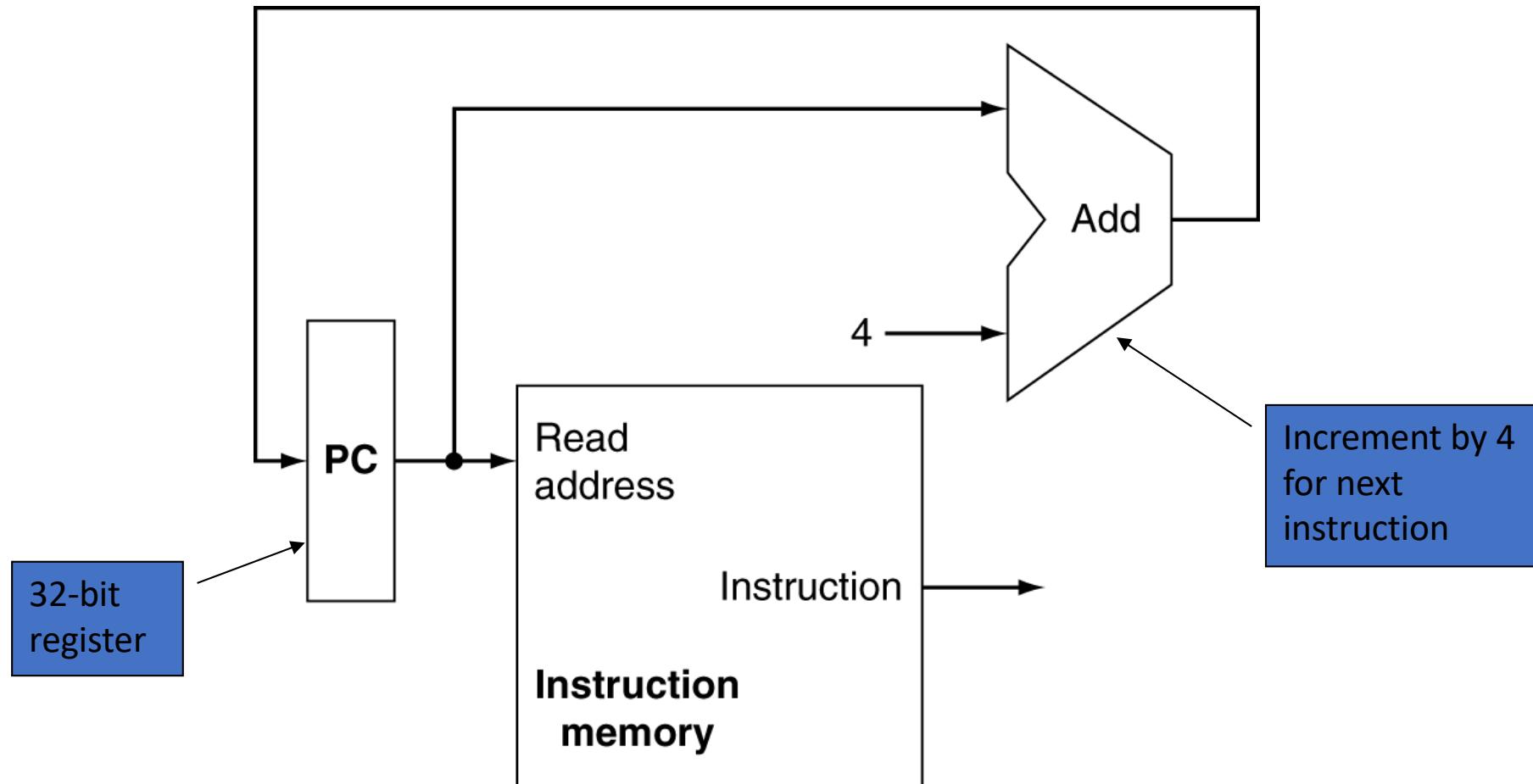


Các thành phần để thực hiện nhận lệnh



- Bộ đếm chương trình *PC*:
 - Thanh ghi 32-bit chứa địa chỉ của lệnh hiện tại
 - Địa chỉ khởi động = 0xBFC0 0000
- 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 trả tới lệnh kế tiếp

Thực hiện phần nhận lệnh



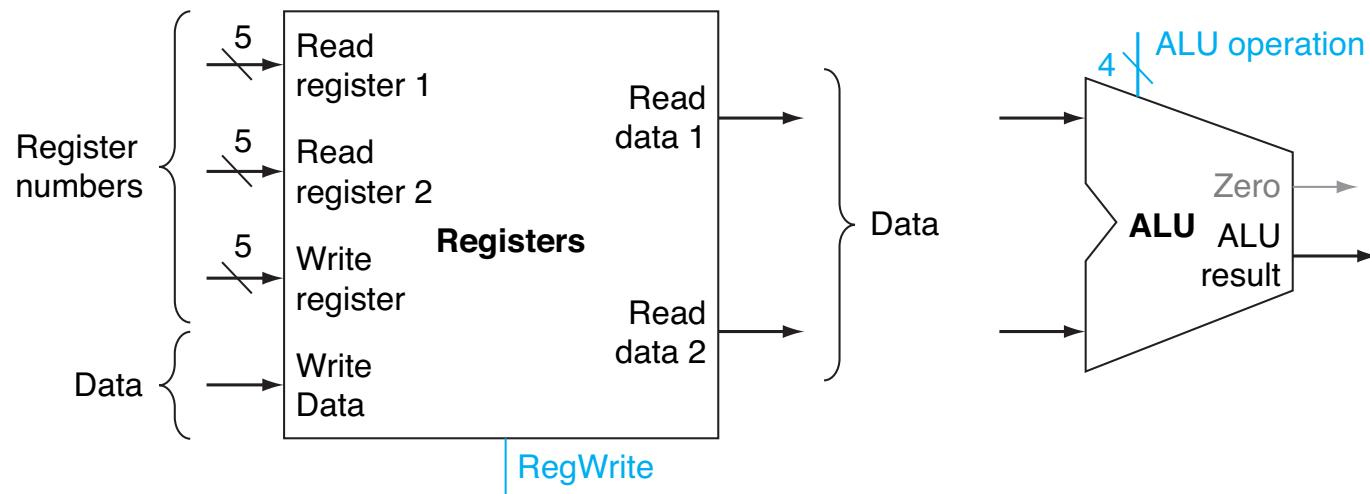
Thực hiện lệnh số học/logic kiểu R

op	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

- bits 31:26: mã thao tác (opcode)
 - 000000 với các lệnh kiểu R
- bits 25:21: số hiệu thanh ghi nguồn thứ nhất rs
- bits 20:16: số hiệu thanh ghi nguồn thứ hai rt
- bits 15:11: số hiệu thanh ghi đích rd
- bits 10:6: số bit được dịch với các lệnh dịch bit
 - 00000 với các lệnh khác
- bits 5:0: mã hàm để xác định phép toán (function code)



Các thành phần thực hiện lệnh kiểu R



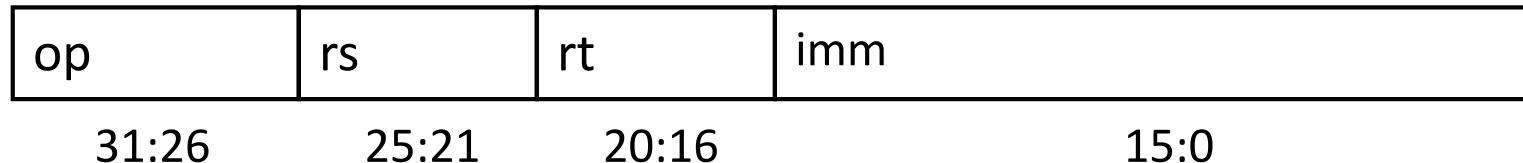
- Tập thanh ghi (*Registers*): có 32 thanh ghi 32-bit, mỗi thanh ghi được xác định bởi số hiệu 5-bit
 - *Read register 1*, *Read register 2*: các đầu vào để chọn các thanh ghi cần đọc
 - *Write register*: đầu vào để chọn thanh ghi cần ghi
 - *Read data 1*, *Read data 2*: hai đầu ra dữ liệu đọc từ thanh ghi (32-bit)
 - *Write Data*: đầu vào dữ liệu ghi vào thanh ghi (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 số học/logic kiểu R

- Hai số hiệu thanh ghi *rs* và *rt* 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 từ hai thanh ghi nguồn được đọc 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



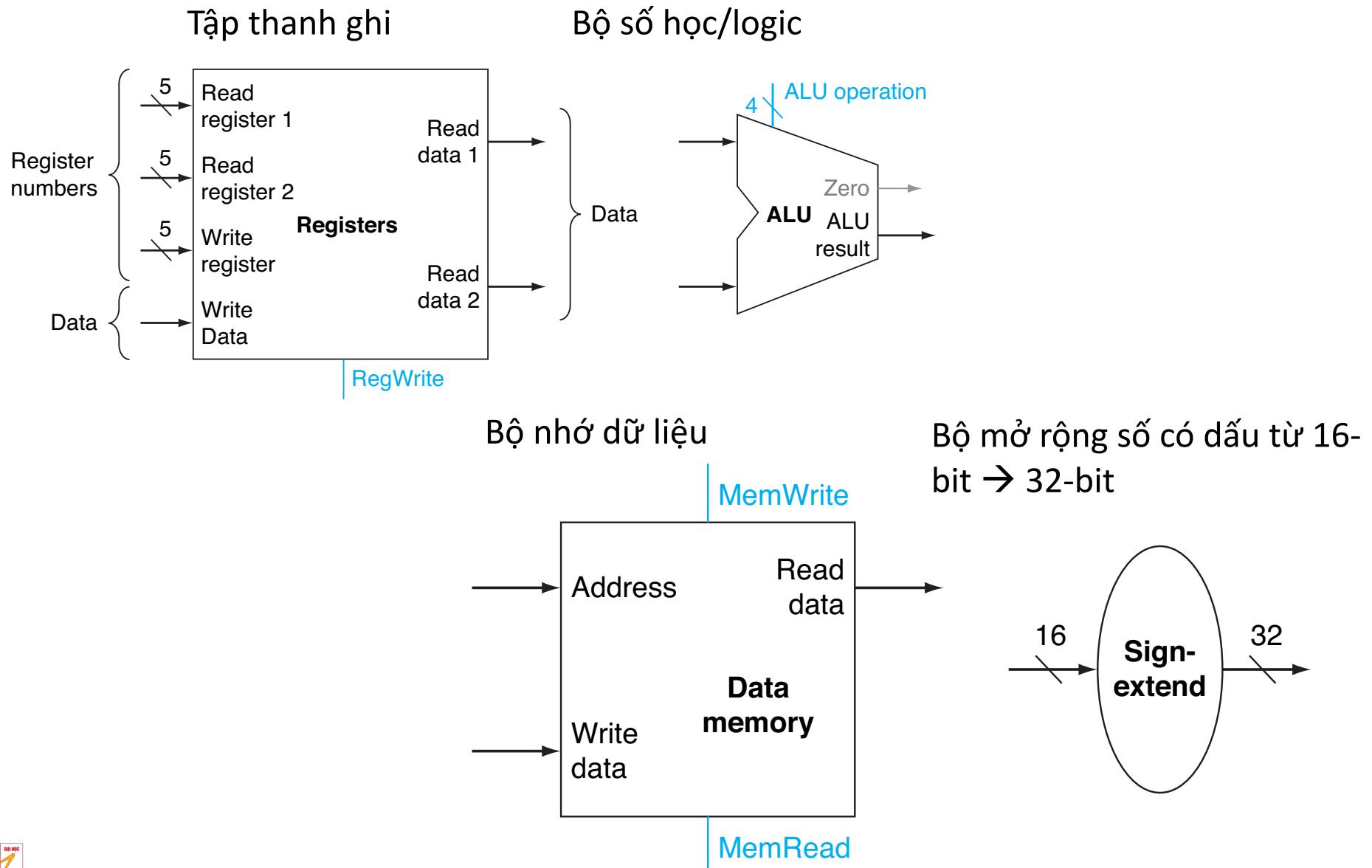
Thực hiện các lệnh lw/sw



- bits 31:26 là mã thao tác
 - 100011 (35) với lệnh lw
 - 101011 (43) với lệnh sw
- bits 25:21: số hiệu thanh ghi cơ sở rs
- bits 20:16: số hiệu thanh ghi rt
 - thanh ghi đích với lệnh lw
 - thanh ghi nguồn với lệnh sw
- bits 15:0: hằng số imm có giá trị trong dải $[-2^{15}, +2^{15} - 1]$
- Địa chỉ từ nhớ dữ liệu = nội dung thanh ghi rs + imm
- $lw \ rt, \ imm(rs) \ #(rt) = mem[(rs) + \text{SignExtImm}]$
- $sw \ rt, \ imm(rs) \ #mem[(rs) + \text{SignExtImm}] = (rt)$



Các thành phần thực hiện các lệnh lw/sw



Thực hiện lệnh lw

- Số hiệu thanh ghi **rs** đưa đến đầu vào **Read register 1** để chọn thanh ghi cơ sở **rs**, nội dung **rs** được đưa ra **Read Data 1**, rồi chuyển đến **ALU**
- Hằng số **imm** 16-bit đưa đến bộ **Sign-extend** để mở rộng thành 32-bit, rồi chuyển đến **ALU**
- **ALU** cộng hai giá trị trên đưa 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 **rt** 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 **rt**



Thực hiện lệnh sw

- Số hiệu thanh ghi **rs** đưa đến đầu vào **Read register 1** để chọn thanh ghi cơ sở **rs**, nội dung **rs** được đưa ra đầu ra **Read Data 1**, rồi chuyển đến **ALU**
- Hằng số imm 16-bit đưa đến bộ **Sign-extend** để mở rộng thành 32-bit, rồi chuyển đến **ALU**
- **ALU** cộng hai giá trị trên đưa 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 **rt** đưa đến đầu vào **Read register 2** để chọn thanh ghi dữ liệu nguồn **rt**, nội dung **rt** 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 vị trí nhớ có địa chỉ đã được tính, nhờ tín hiệu điều khiển **MemWrite**



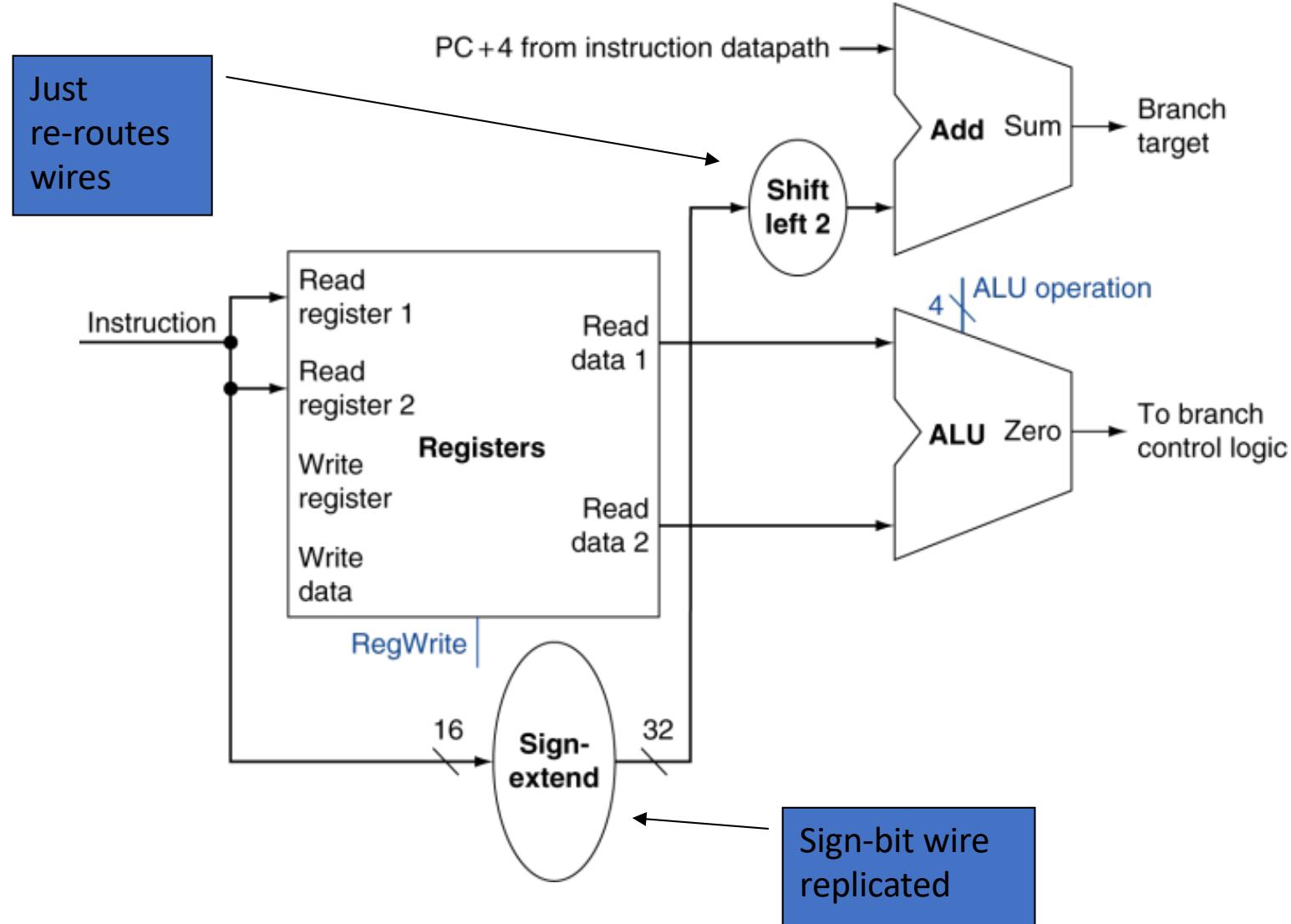
Thực hiện lệnh Branch (beq, bne)

4 or 5	rs	rt	imm
31:26	25:21	20:16	15:0

- bits 31:26 mã thao tác
 - 000100 (4) với lệnh beq
 - 000101 (5) với lệnh bne
- bits 25:21 số hiệu thanh ghi nguồn rs
- bits 20:16 số hiệu thanh ghi nguồn rt
- bits 15:0 hằng số imm có giá trị trong dải $[-2^{15}, +2^{15} - 1]$
- So sánh nội dung hai thanh ghi rs và rt:
 - Nếu điều kiện đúng: rẽ nhánh đến nhãn đích
 - $PC \leftarrow (PC + 4) + \text{hằng số} \times 4$
 - Nếu điều kiện sai: chuyển sang thực hiện lệnh kế tiếp
 - $PC \leftarrow PC + 4$



Các thành phần thực hiện lệnh Branch



Thực hiện lệnh Branch (beq, bne)

- Nhờ các số hiệu thanh ghi **rs** và **rt**, hai toán hạng nguồn được đọc ra đưa đến **ALU**
- **ALU** trừ hai toán hạng và thiết lập giá trị ở đầu ra “**Zero**”
 - Hiệu = 0 → đầu ra **Zero** = 1
 - Hiệu \neq 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 16-bit được mở rộng theo kiểu có dấu thành 32-bit, rồi dịch trái 2 bit
 - Cộng với PC (PC đã được tăng 4)
→ Địa chỉ đích = $(PC+4) + (\text{hằng số đã mở rộng 32-bit, dịch trái 2 bit})$
- Điều kiện đúng: $PC \leftarrow \text{địa chỉ đích rẽ nhánh}$ (rẽ nhánh xảy ra)
- Điều kiện sai: $PC \leftarrow PC+4$ (chuyển sang lệnh kế tiếp)

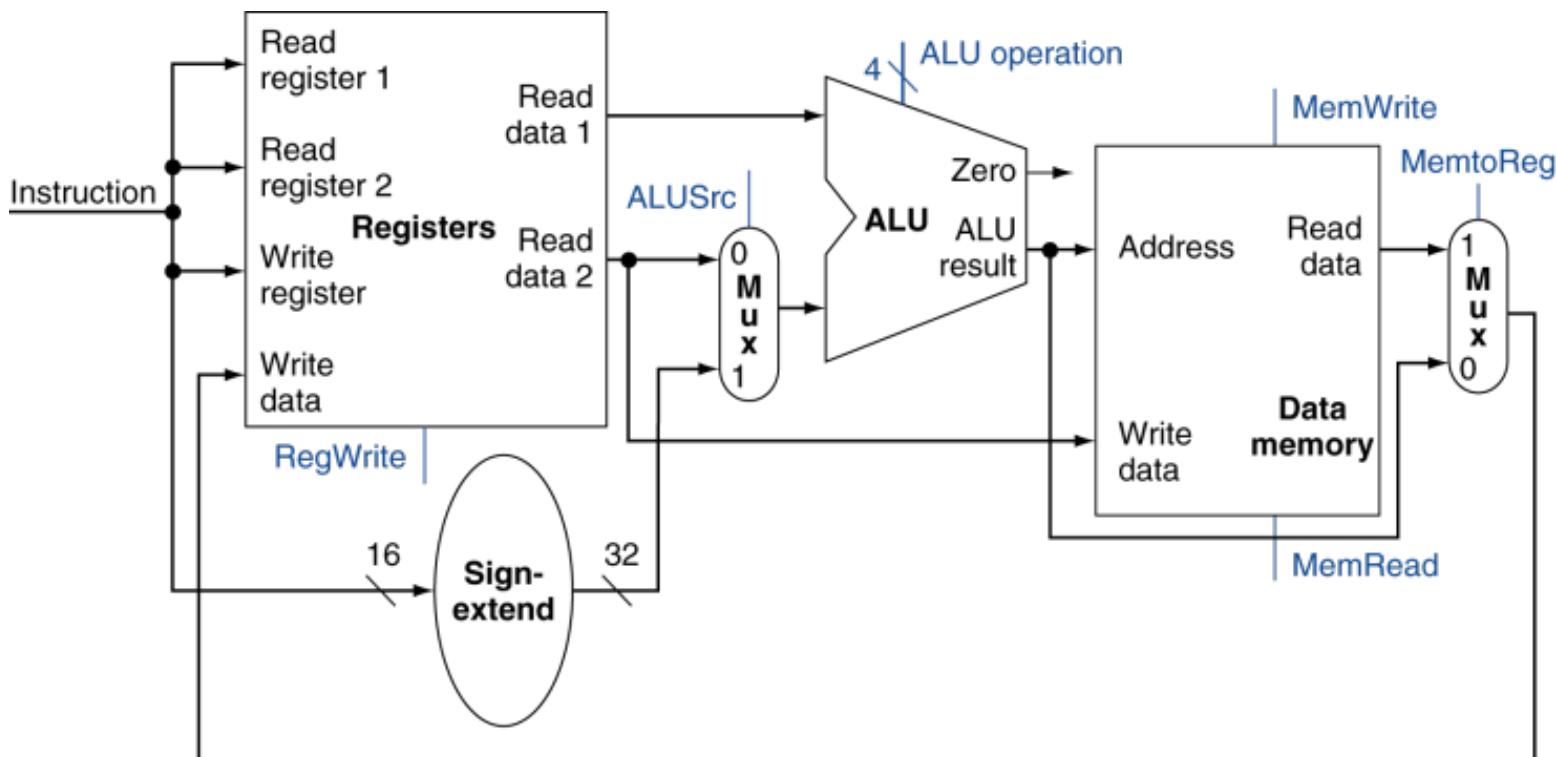


Hợp các thành phần cho các lệnh

- 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

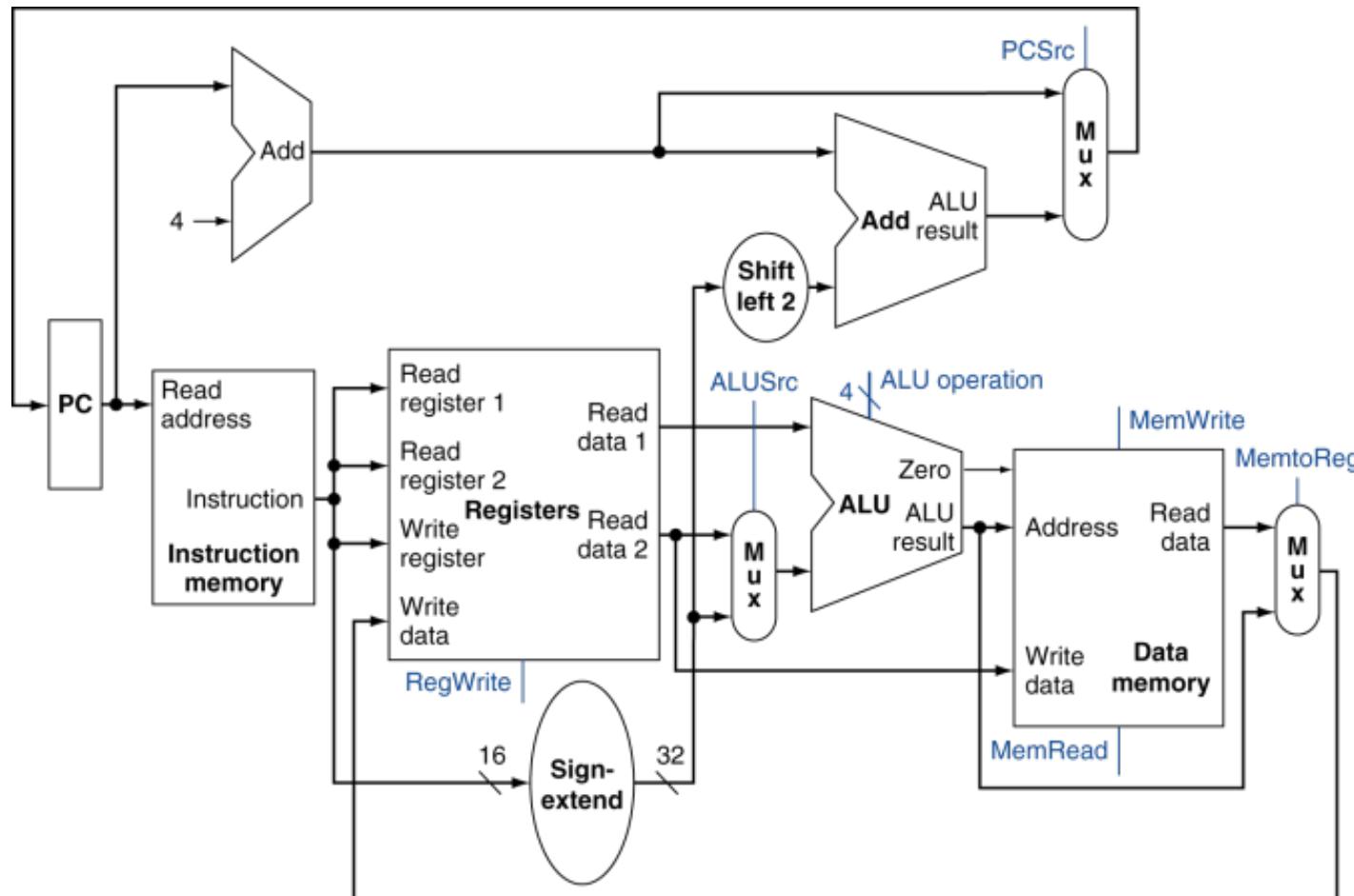


Datapath cho các lệnh R-Type/Load/Store



- **ALUSrc:** tín hiệu điều khiển chọn toán hạng đưa đến ALU:
 - Lệnh kiểu R: toán hạng từ thanh ghi nguồn thứ hai
 - Lệnh lw/sw: Hằng số imm 16-bit được mở rộng thành 32-bit (tính địa chỉ)
- **MemtoReg:** tín hiệu điều khiển chọn dữ liệu đưa về thanh ghi đích:
 - Lệnh kiểu R: lấy kết quả từ ALU result
 - Lệnh lw: dữ liệu đọc (Read data) từ bộ nhớ dữ liệu

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



- PCSrc: tín hiệu điều khiển chọn giá trị cập nhật PC
 - Không rẽ nhánh: $PC \leftarrow PC + 4$
 - Rẽ nhánh: $PC \leftarrow (PC + 4) + (\text{hằng số imm đã mở rộng thành 32-bit} \ll 2)$

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
0111	set-on-less-than
1100	NOR



Tín hiệu điều khiển ALU

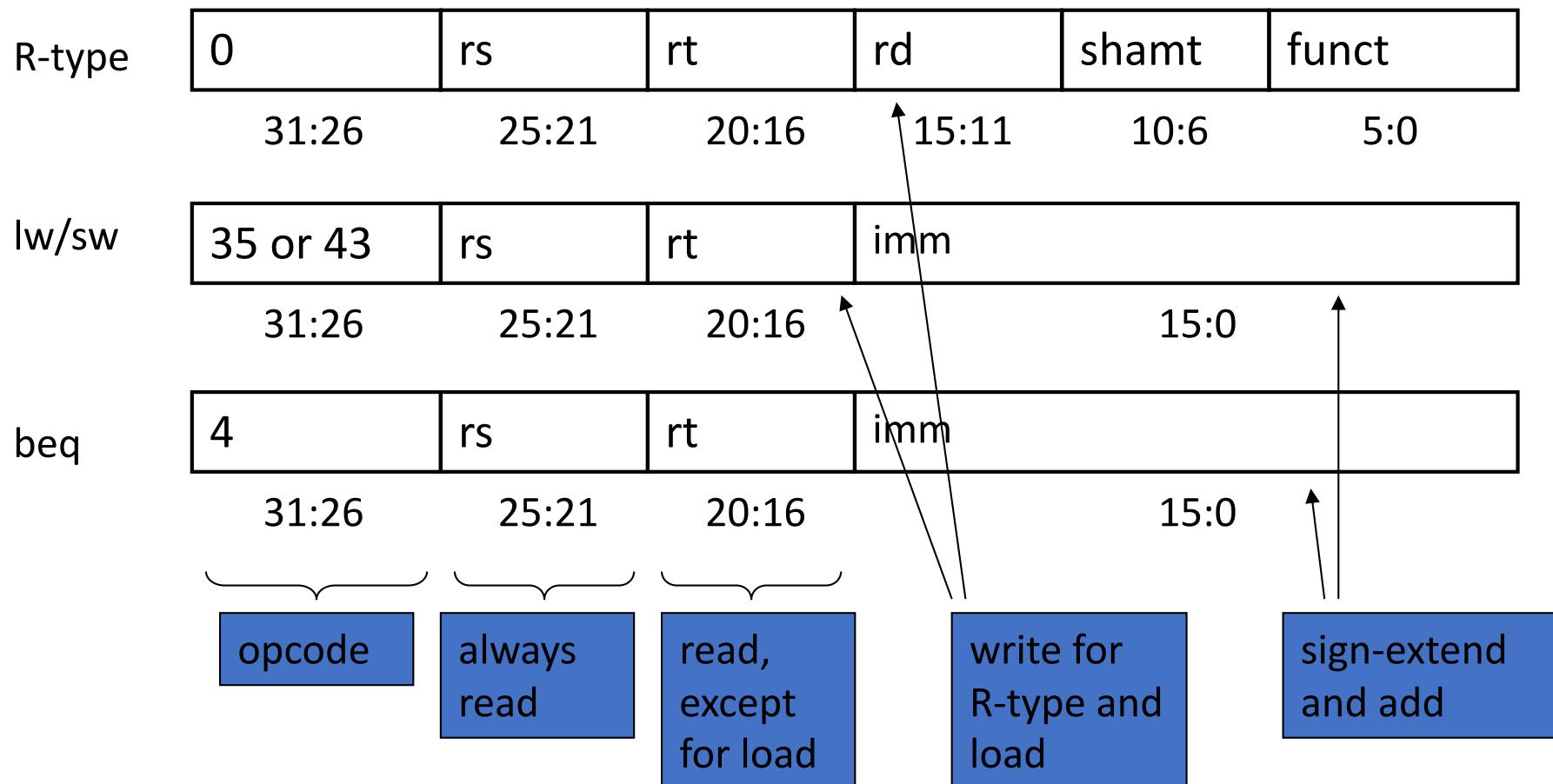
- Bộ điều khiển ALU sử dụng mạch logic tổ hợp:
 - Đầu vào: 2-bit ALUOp được tạo ra từ opcode của lệnh và 6-bit của function code
 - Đầu ra: các tín hiệu điều khiển ALU (ALU control) gồm 4 bit

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

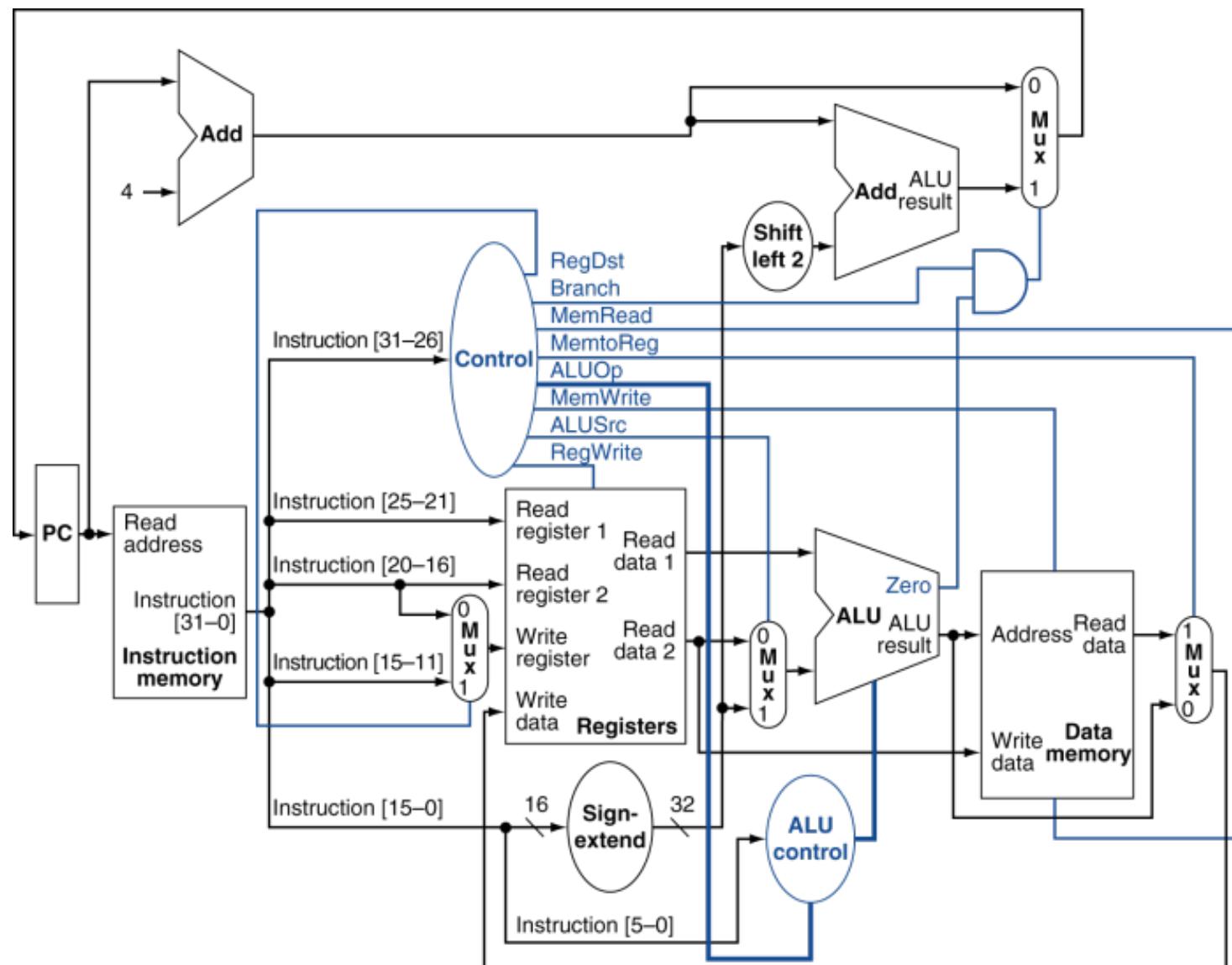


Thiết kế bộ điều khiển chính

- Các tín hiệu điều khiển được tạo ra từ lệnh



Datapath và Control Unit



Các tín hiệu điều khiển

Tên tín hiệu	Hiệu ứng khi tín hiệu = 0	Hiệu ứng khi tín hiệu = 1
RegDst	Số hiệu thanh ghi đích là các bit 20:16 (rt)	Số hiệu thanh ghi đích là các bit 15:11 (rd)
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 làm gì cả	Ghi dữ liệu trên đầu vào <i>Write Data</i> 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 16 bit thấp của lệnh (bits 15:0) được mở rộng có dấu thành 32-bit
PCSrc	$PC \leftarrow PC+4$	$PC \leftarrow$ địa chỉ đích

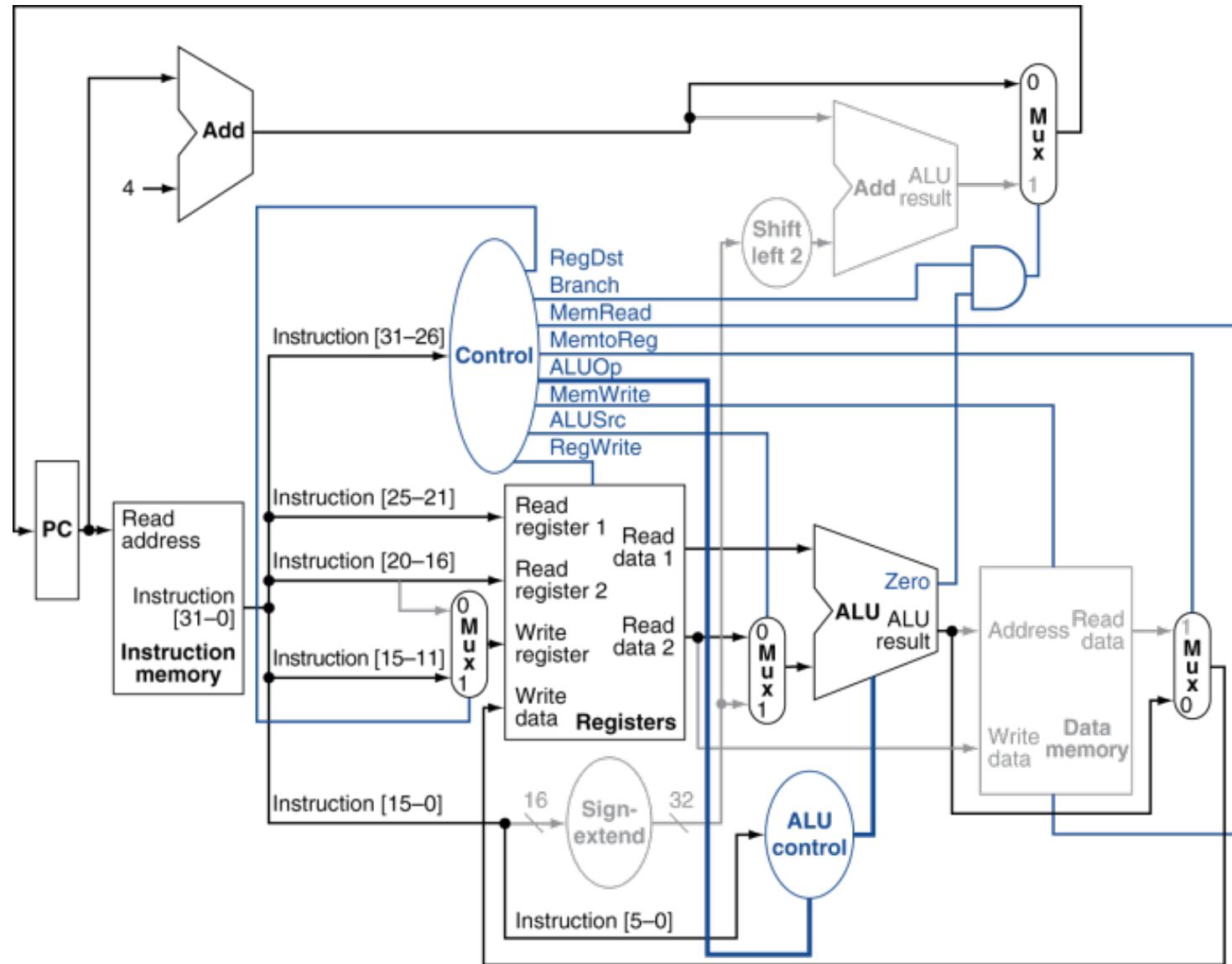


Các tín hiệu điều khiển (tiếp)

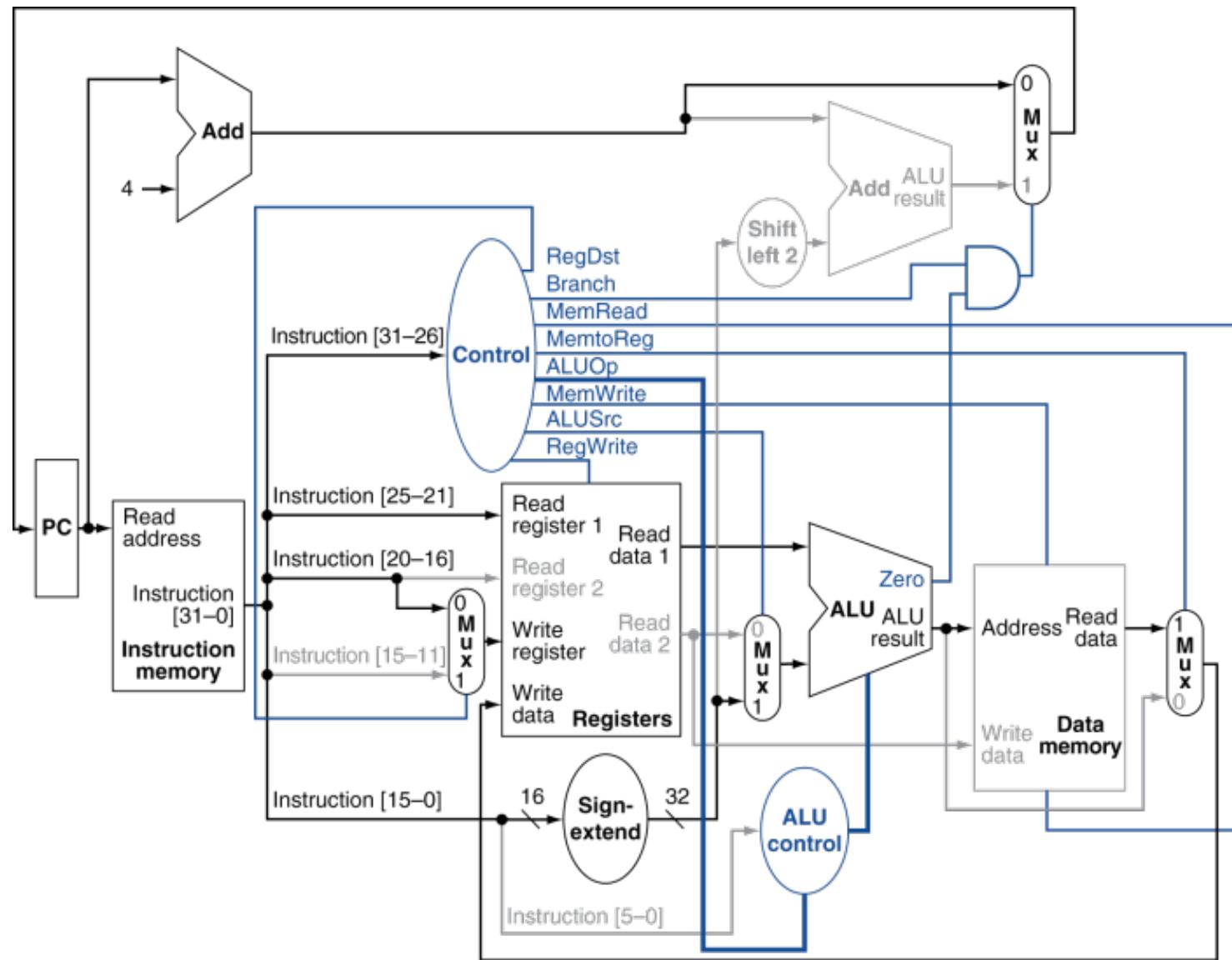
Tên tín hiệu	Hiệu ứng khi tín hiệu = 0	Hiệu ứng khi tín hiệu = 1
MemRead	Không làm gì cả	Nội dung ngăn nhớ dữ liệu, được xác định bởi địa chỉ do ALU tính, được đưa ra đầu ra <i>Read data</i> của bộ nhớ dữ liệu
MemWrite	Không làm gì cả	Dữ liệu trên đầu vào <i>Write Data</i> của bộ nhớ dữ liệu được ghi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



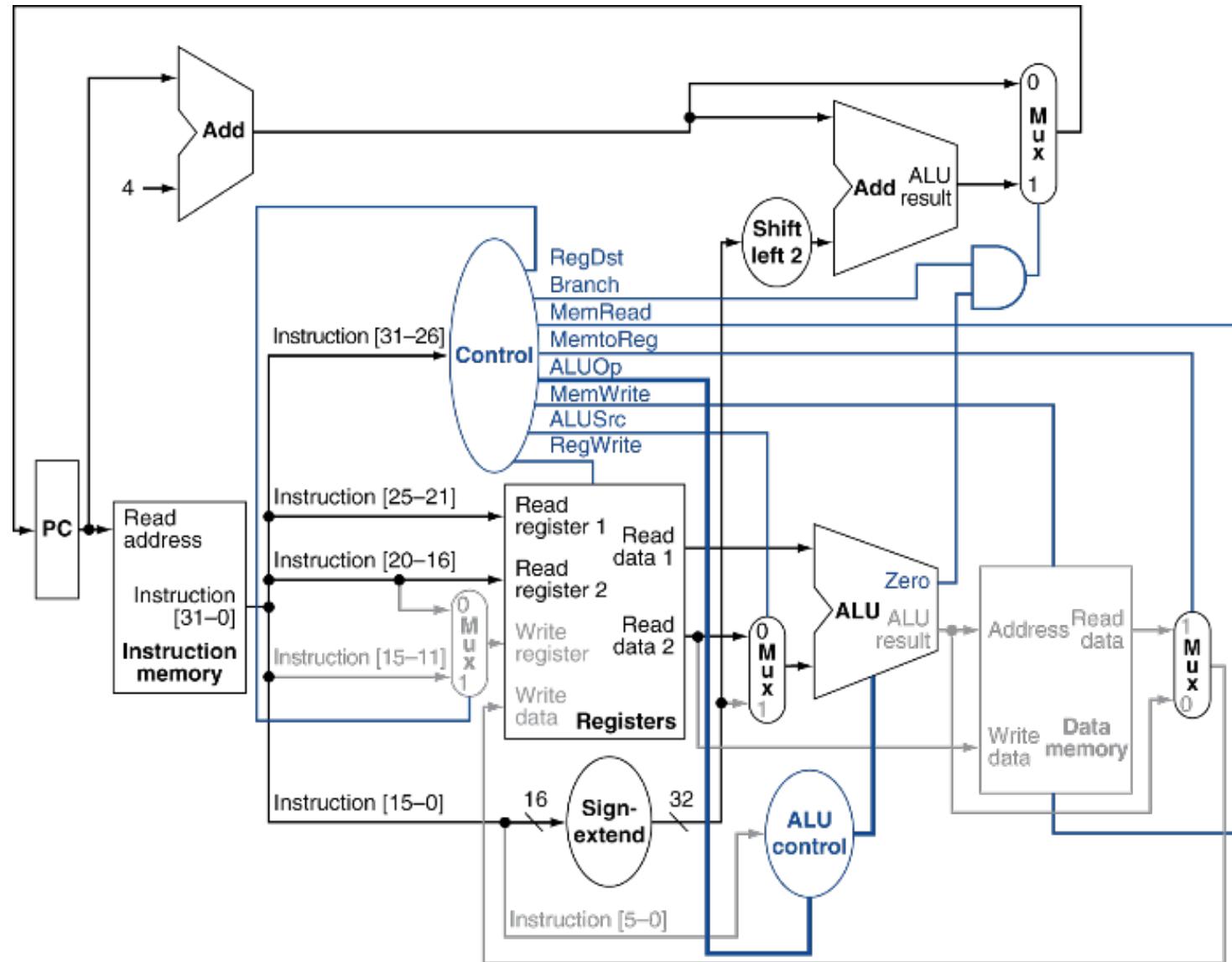
Thực hiện lệnh số học/logic kiểu R



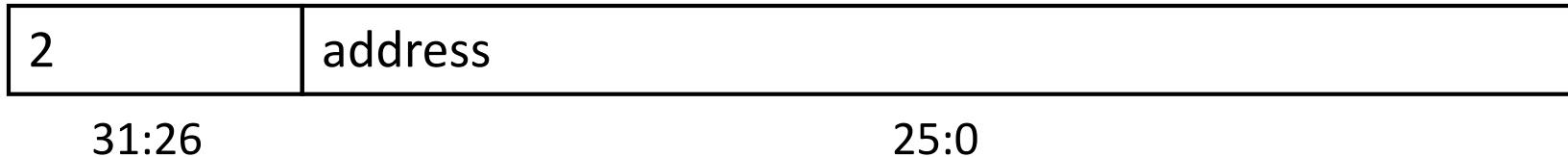
Thực hiện lệnh Load



Thực hiện lệnh beq



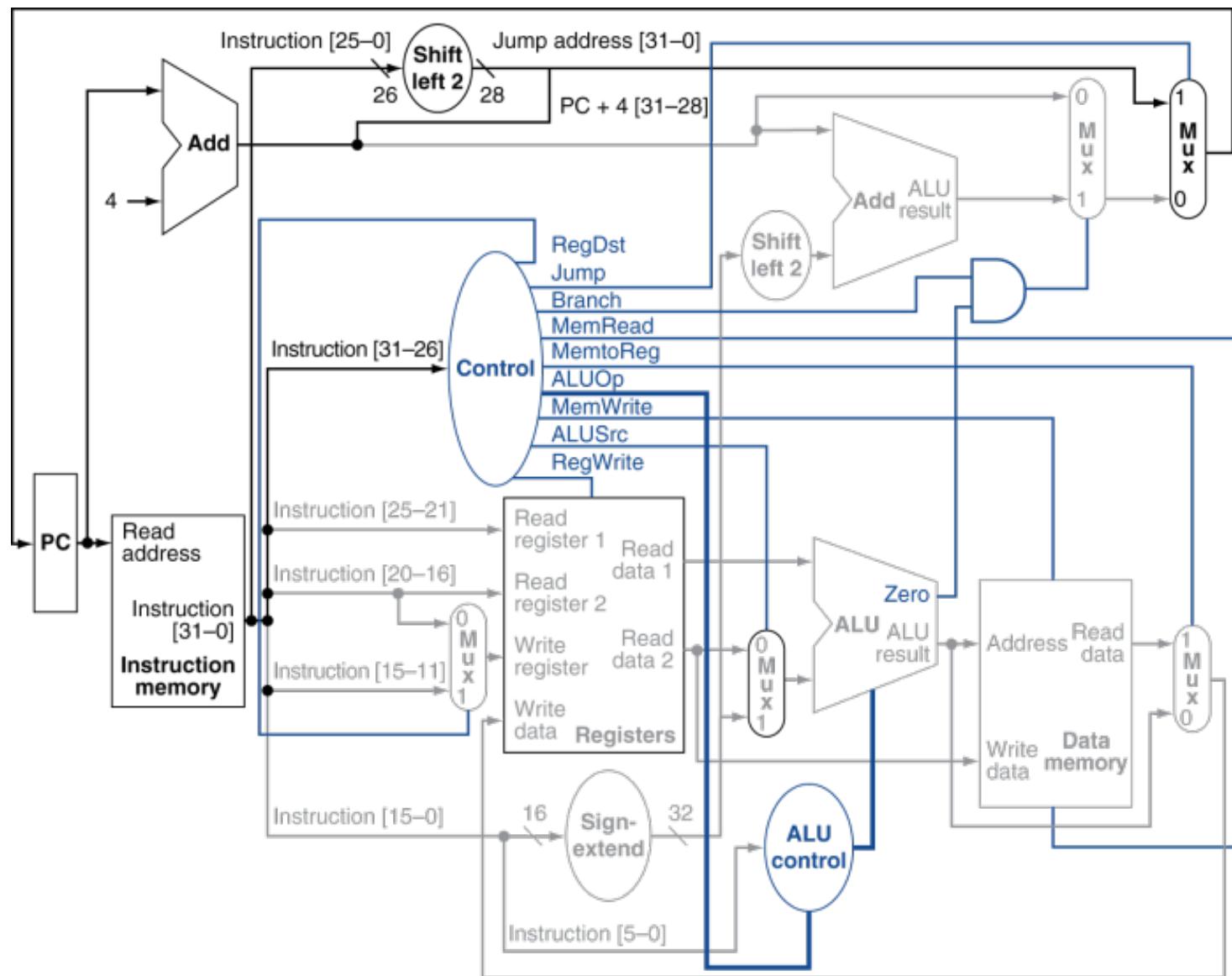
Thực hiện lệnh Jump



- Bits 31:26 là mã thao tác = 000010
- Bits 25:0: phần địa chỉ
- PC nhận giá trị sau:
 - Địa chỉ đích = $PC_{31\dots28} : (address \ll 2)$
 - 4 bit bên trái là của PC cũ
 - 26-bit của lệnh jump (bits 25:0)
 - 2 bit cuối là 00
 - Cần thêm tín hiệu điều khiển được giải mã từ opcode



Datapath thêm cho lệnh jump

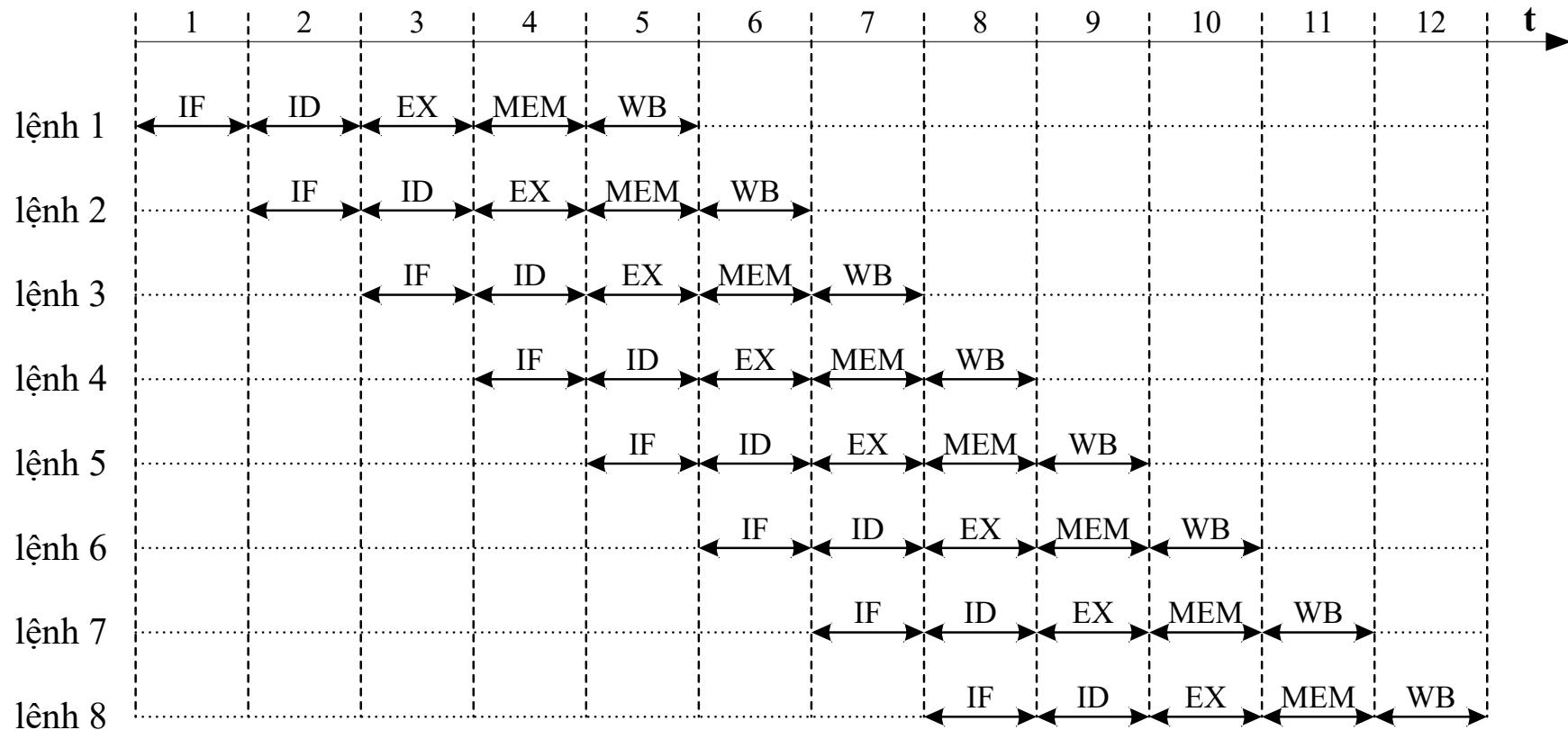


5.3. Kỹ thuật đường ống lệnh và song song mức lệnh

- 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 MIPS có 5 công đoạn:
 1. IF: Instruction fetch from memory – Nhận lệnh từ bộ nhớ
 2. ID: Instruction decode & register read – Giải mã lệnh và đọc thanh ghi
 3. EX: Execute operation or calculate address – Thực hiện thao tác hoặc tính toán địa chỉ
 4. MEM: Access memory operand – Truy nhập toán hạng bộ nhớ
 5. WB: Write result back to register – Ghi kết quả trả về thanh ghi



Biểu đồ thời gian của đường ống lệnh



Thời gian thực hiện 1 công đoạn = T

Thời gian thực hiện tuần tự 8 lệnh: $8 \times 5T = 40T$

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

Các mối 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
- Trong đường ống của MIPS với một bộ nhớ dùng chung
 - Lệnh Load/store yêu cầu truy cập dữ liệu
 - Nhận 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)

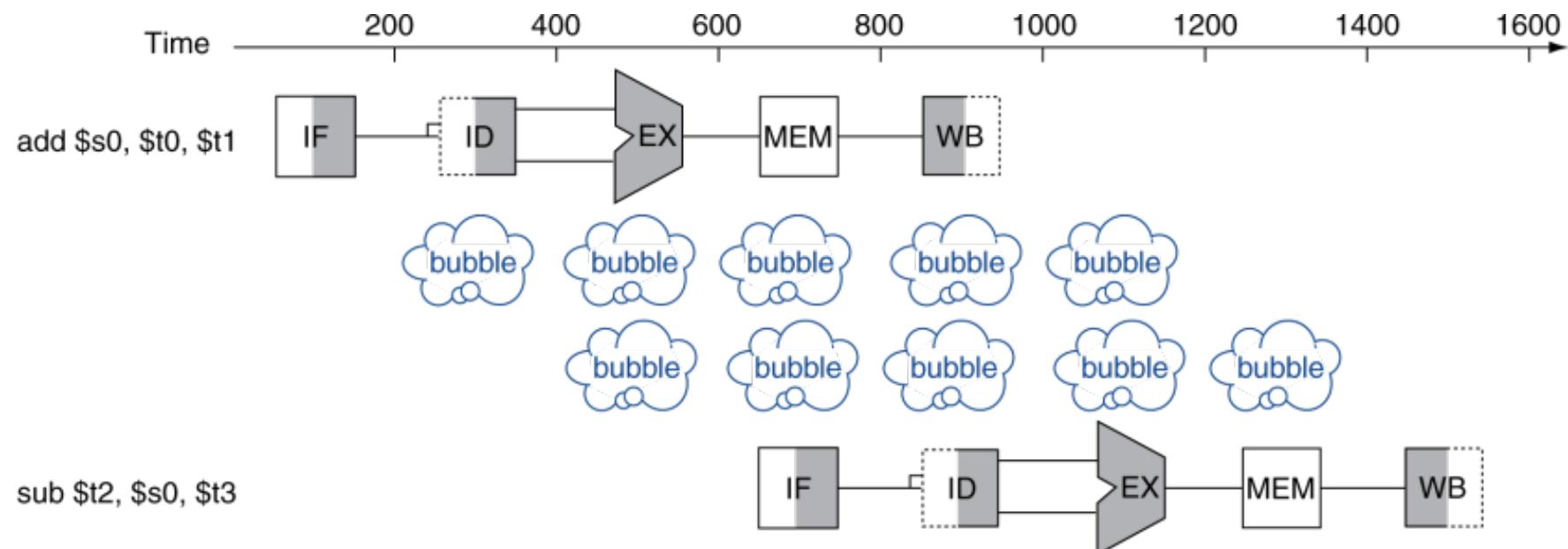


Hazard dữ liệu

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

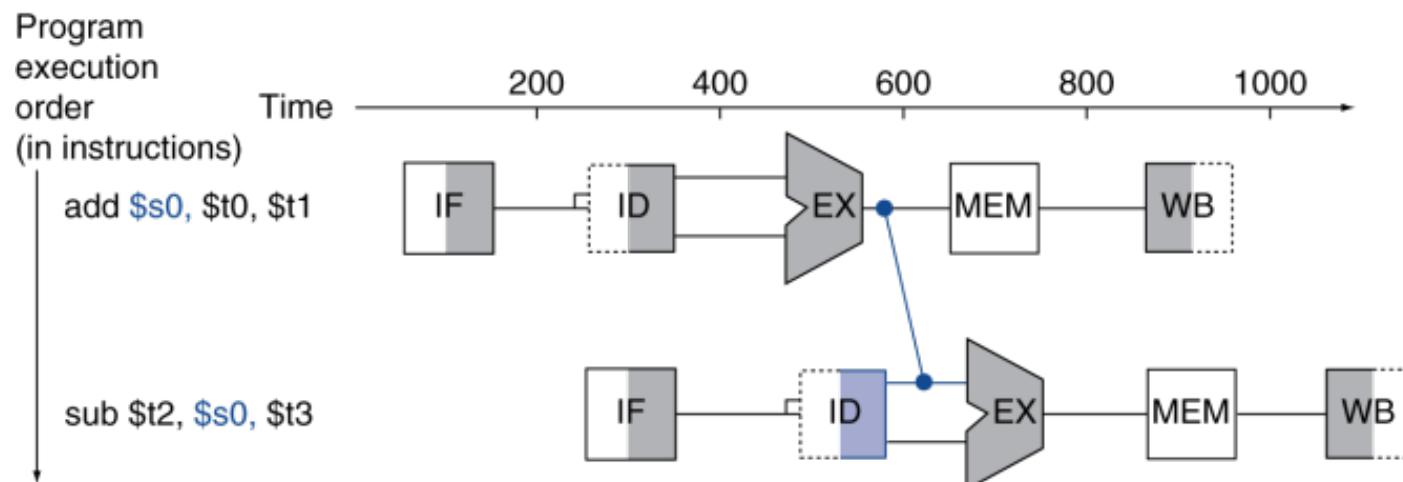
add \$s0, \$t0, \$t1

sub \$t2, \$s0, \$t3



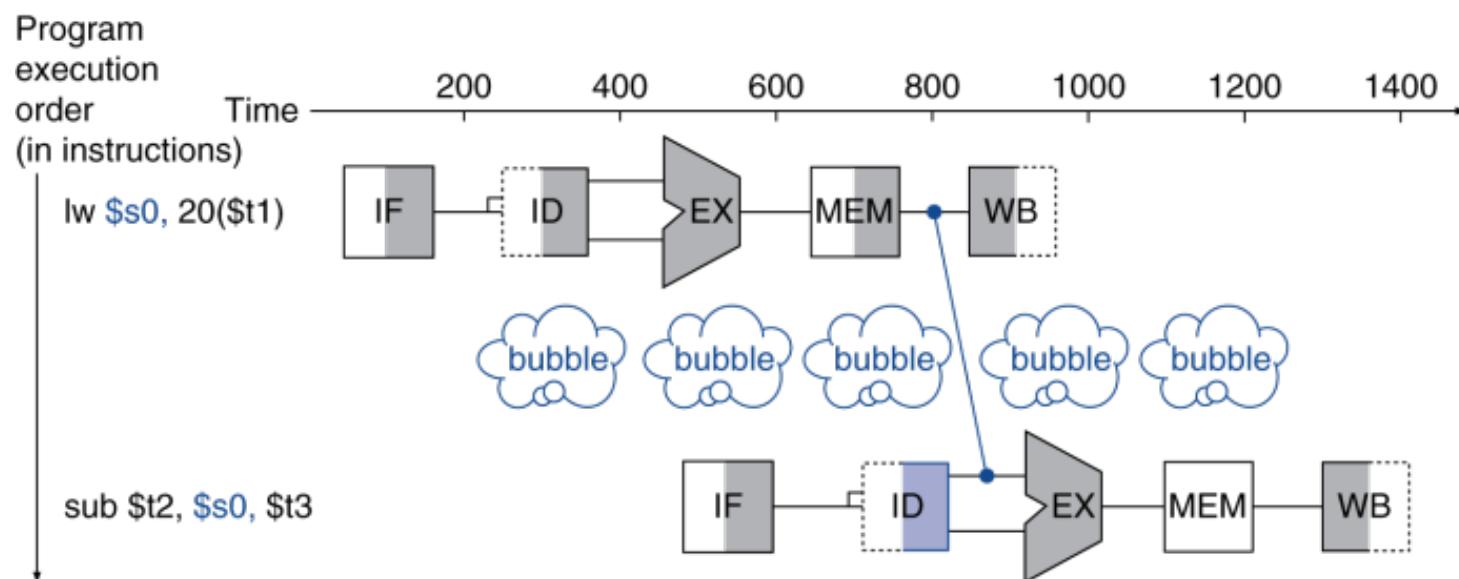
Forwarding (gửi vượt trước)

- Sử dụng kết quả ngay sau khi nó được tính
 - Không đợi đến khi kết quả được lưu đến thanh ghi
 - Yêu cầu có đường kết nối thêm trong datapath



Hazard dữ liệu với lệnh load

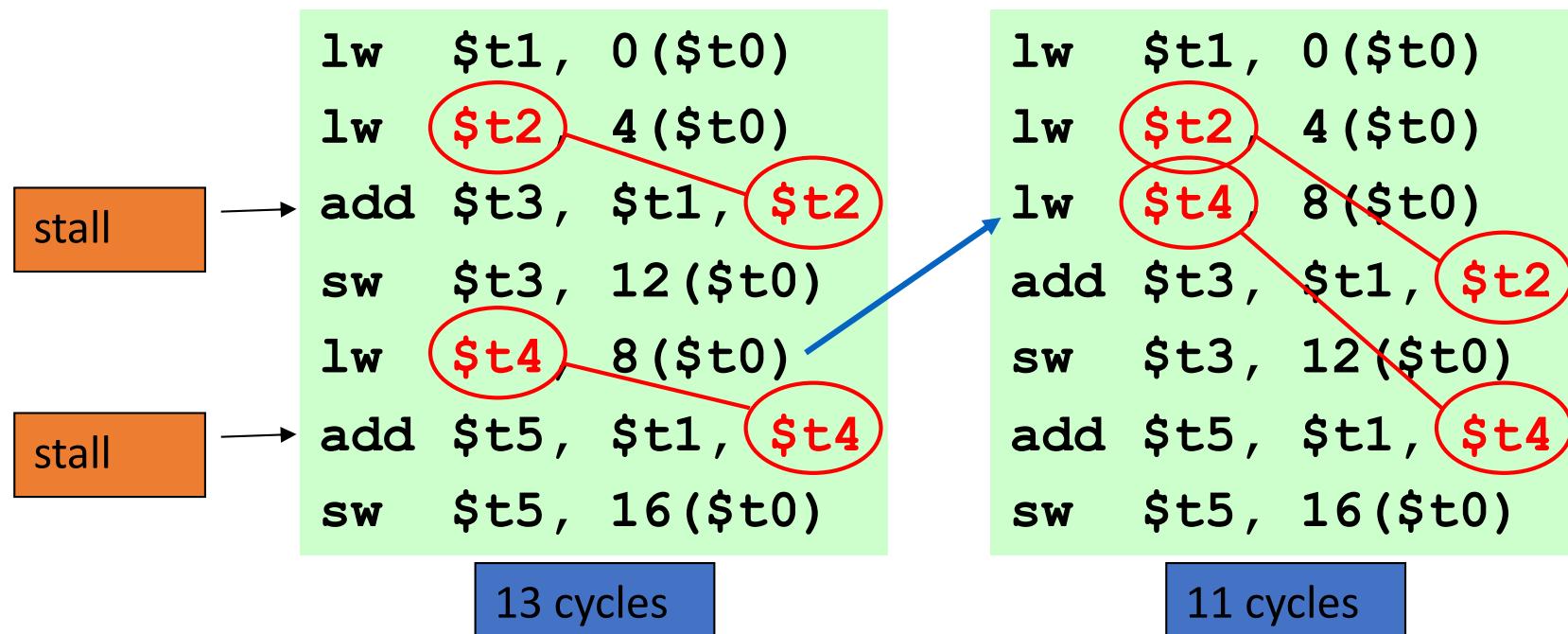
- Không phải luôn luôn có thể tránh trì hoãn bằng cách forwarding
 - Nếu giá trị chưa được tính khi cần thiết
 - Không thể chuyển ngược thời gian
 - Cần chèn bước trì hoãn (stall hay bubble)



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:

$$a = b + e; \quad c = b + f;$$



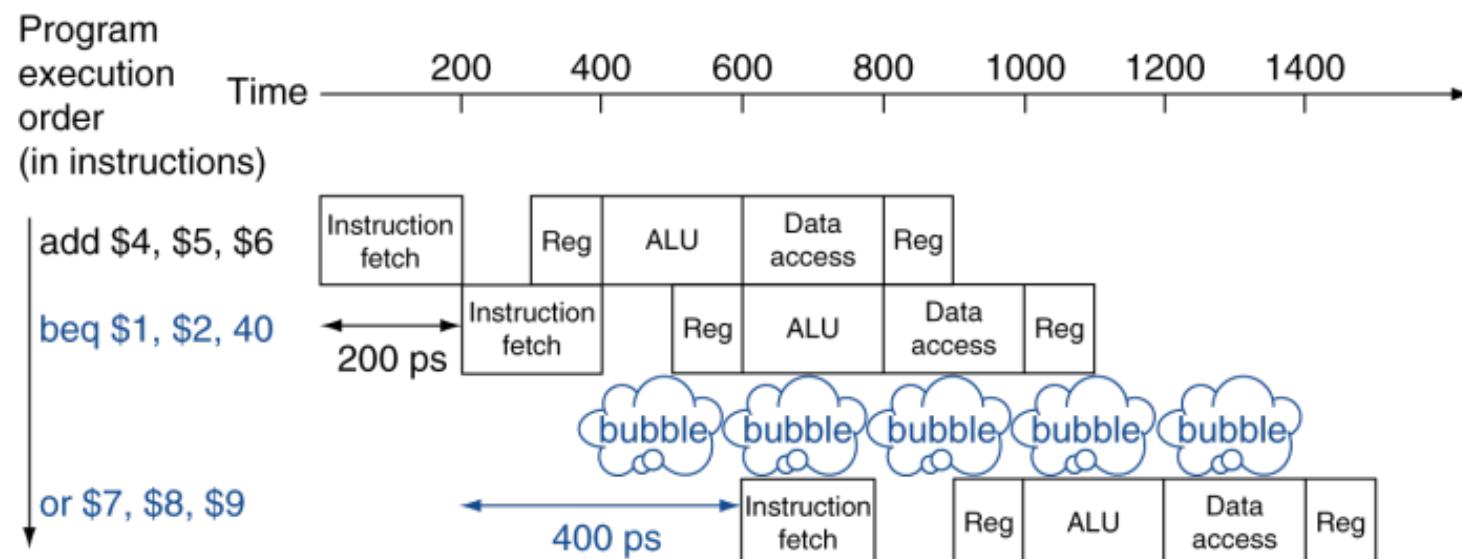
Hazard điều khiển

- Rẽ nhánh xác định luồng điều khiển
 - Nhận 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
- Với đường ống của MIPS
 - 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 nhận lệnh tiếp theo

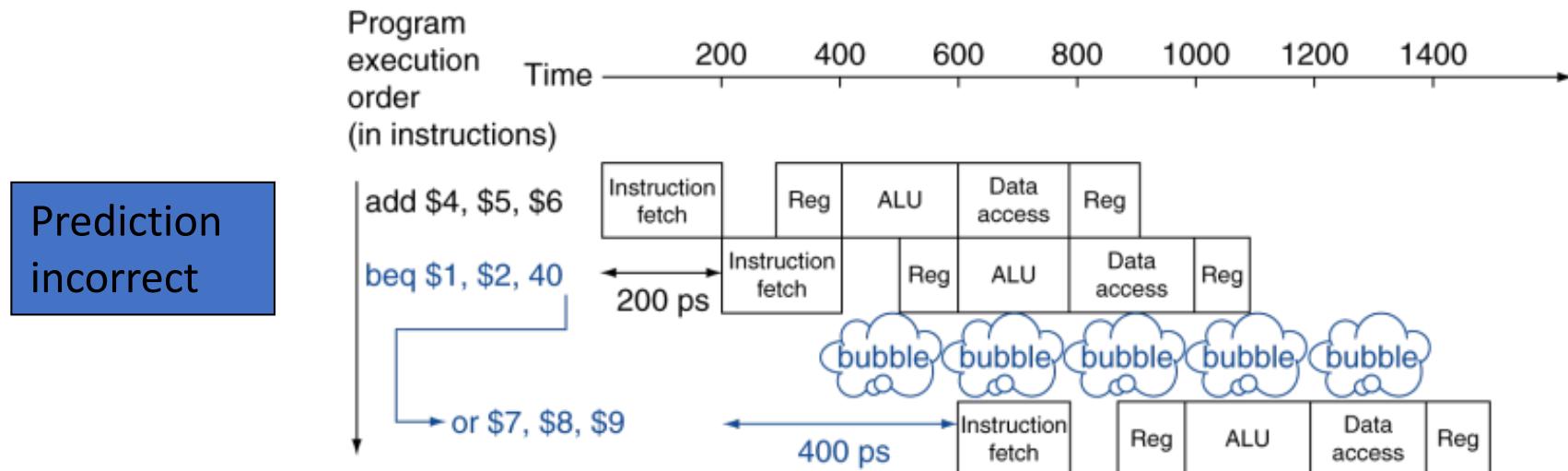
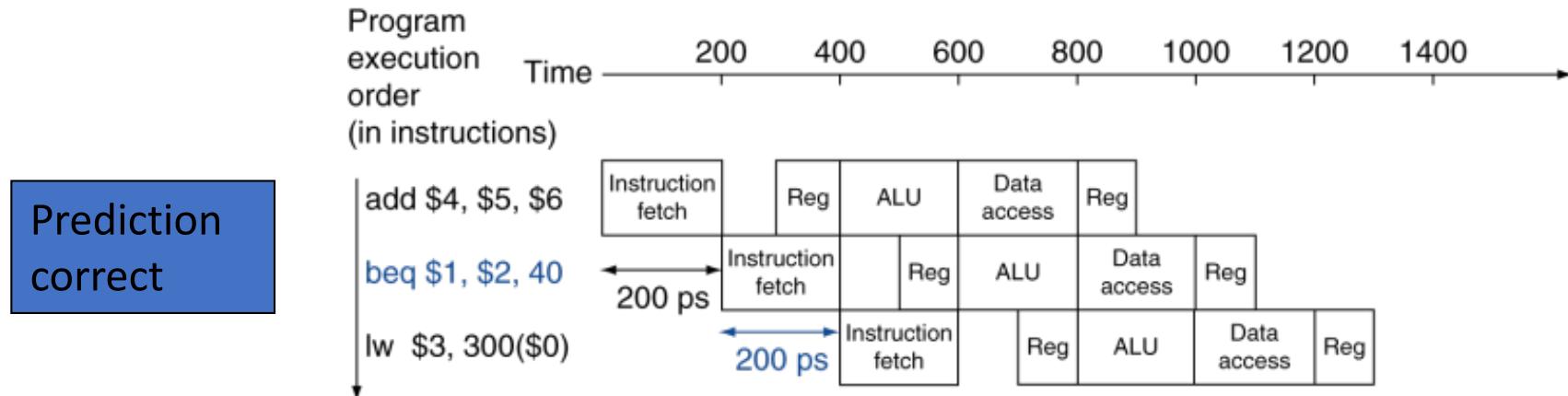


Dự đoán rẽ nhánh

- 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 MIPS
 - Có thể dự đoán rẽ nhánh không xảy ra
 - Nhận lệnh ngay sau lệnh rẽ nhánh (không làm trễ)



MIPS với dự đoán rẽ nhánh không xảy ra



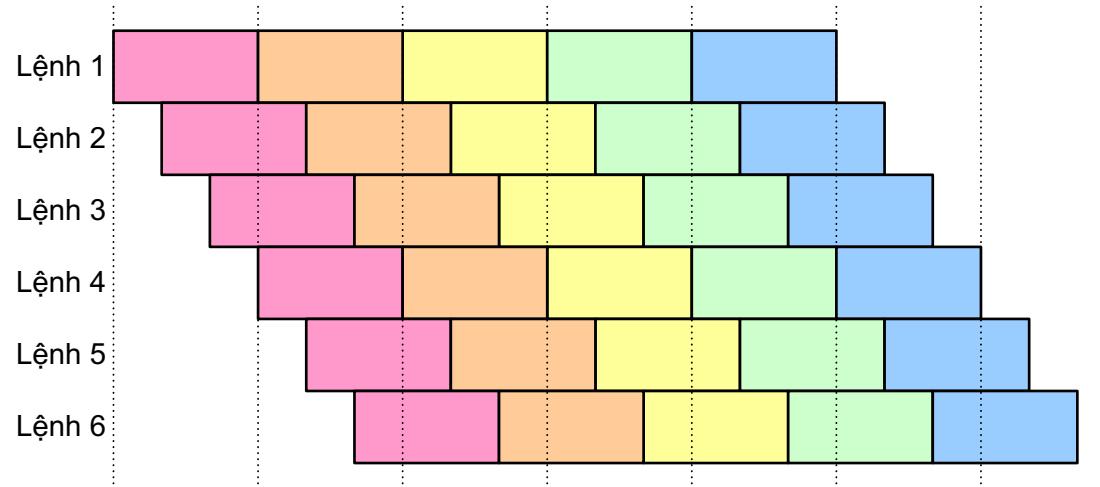
Đặc điểm của đường ống

- 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

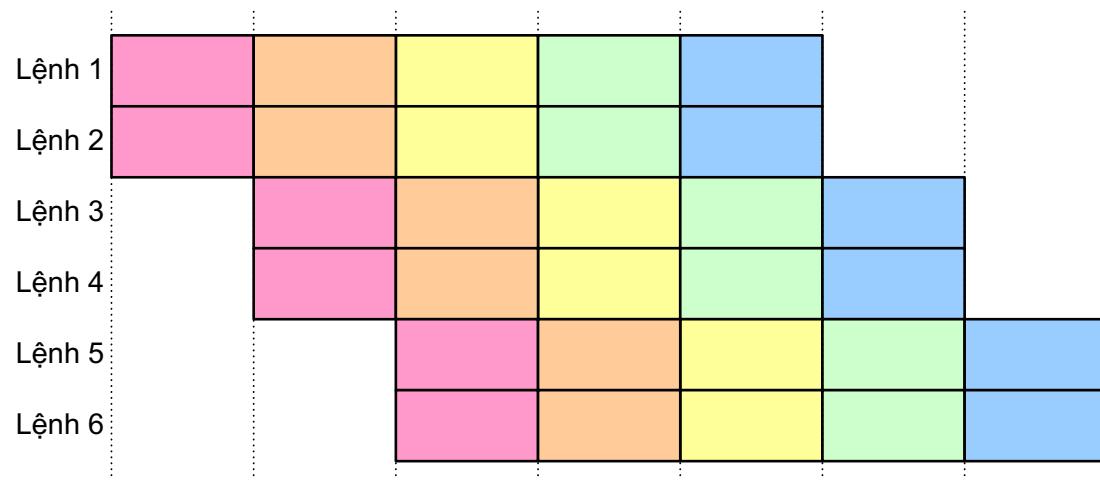


Tăng cường khả năng song song mức lệnh

■ Tăng số công đoạn của đường ống



■ Siêu vô hướng (Superscalar)



Hết chương 5

