



TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN – ĐHQG HCM
KHOA KỸ THUẬT MÁY TÍNH

TỔ CHỨC VÀ CẤU TRÚC MÁY TÍNH 2

CHƯƠNG 8 BỘ XỬ LÝ

PHAN ĐÌNH DUY

TP. Hồ Chí Minh, ngày 05 tháng 9 năm 2022



ÔN TẬP

- Tổng quan về kiến trúc tập lệnh - MIPS
- Các loại toán hạng trong lệnh MIPS
- Các định dạng lệnh trong MIPS
- Chuyển đổi giữa: ngôn ngữ cấp cao \Leftrightarrow AMS \Leftrightarrow mã máy
- Viết một chương trình ASM với lệnh MIPS
- Bài tập



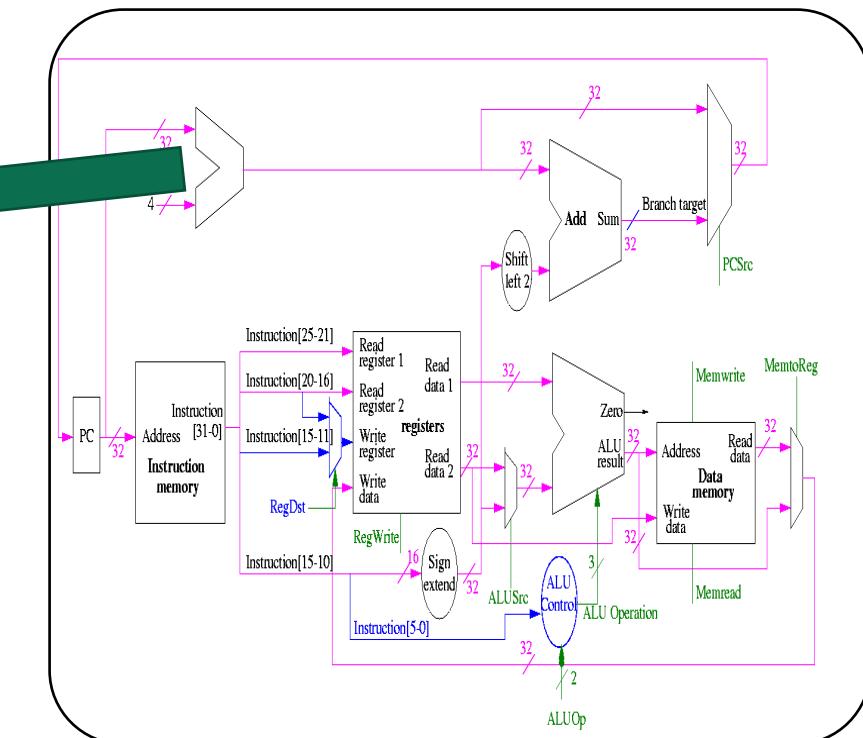
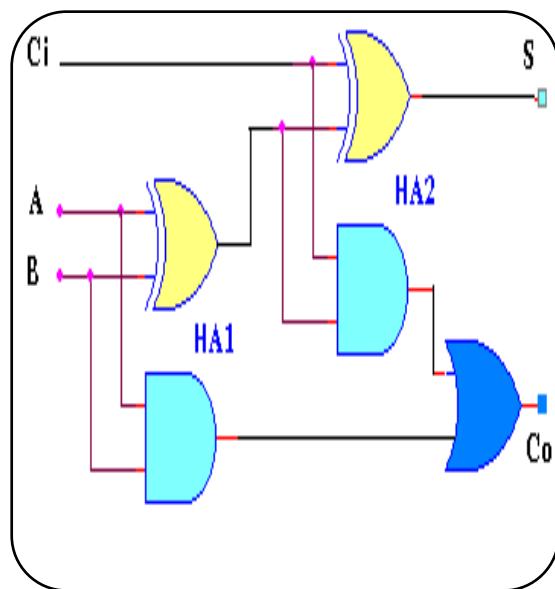
MỤC TIÊU CHƯƠNG

- Biết được các kiến trúc tổng quan bộ xử lý
- Hiểu được các thiết kế và hoạt động của datapath
- Hiểu được quá trình thực thi lệnh
- Bài tập



NỘI DUNG

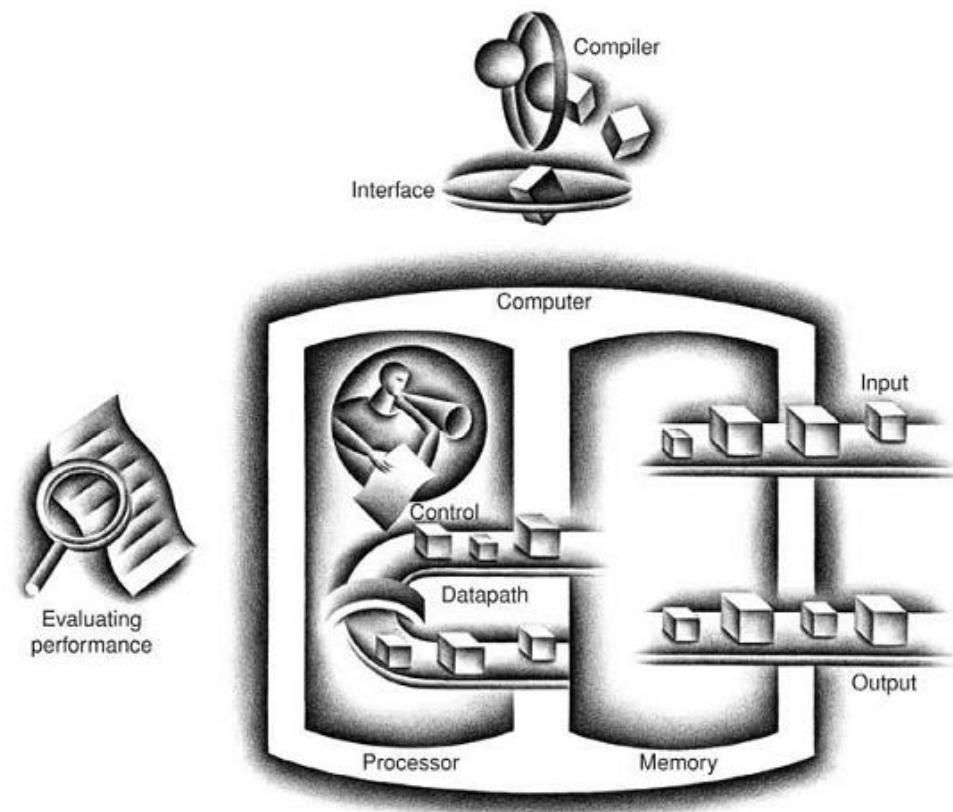
- Kiến trúc tổng quan bộ xử lý
- Datapath
- Thực thi lệnh
- Bài tập





NỘI DUNG

- **Kiến trúc tổng quan bộ xử lý**
- Datapath
- Thực thi lệnh
- Bài tập



Hình 1: Kiến trúc của 1 máy tính



Kiến trúc tổng quan bộ xử lý

- Kiến trúc máy tính bao gồm 3 thành phần chính:
 - Kiến trúc tập lệnh (ISA): Quy định máy tính có thể làm những việc gì?
 - Lệnh
 - **Vi kiến trúc (Tổ chức phần cứng máy tính): Quy định máy tính làm việc như thế nào?**
 - Hiện thực ISA
 - Hệ thống máy tính: Quy định các thành phần của máy tính phối hợp trong một hệ thống điện toán như thế nào?
 - Ảo hóa, Quản lý bộ nhớ, Xử lý đồ họa...



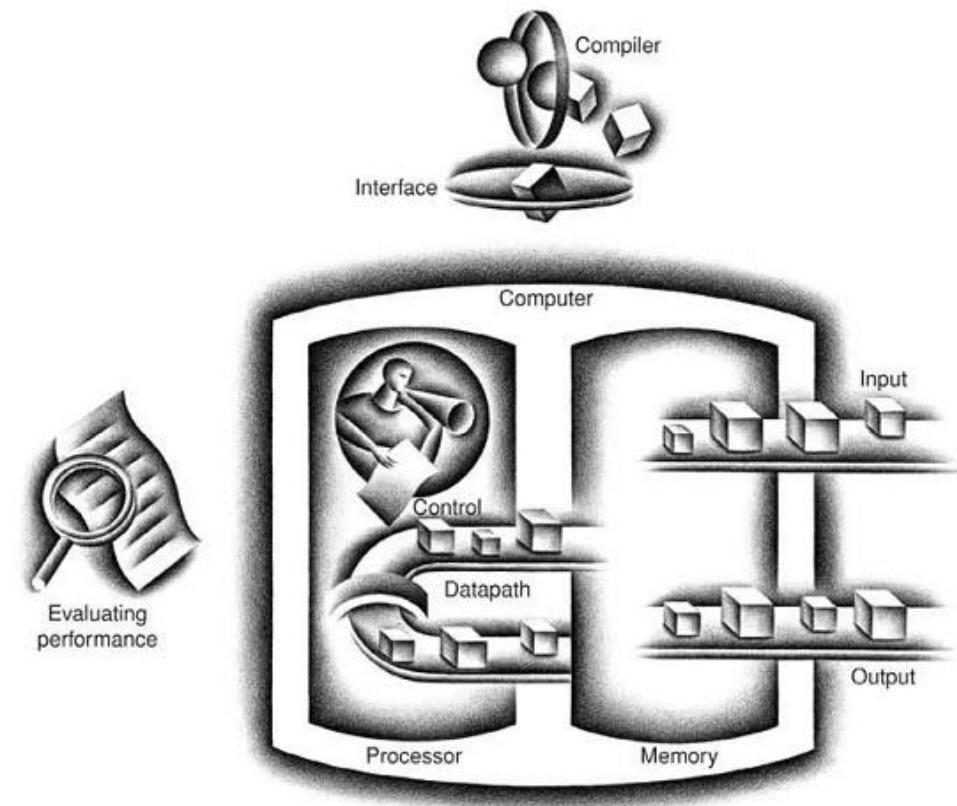
Kiến trúc tổng quan bộ xử lý (2)

- Về chức năng, vi kiến trúc là một tổ chức phần cứng dùng để hiện thực tập lệnh của một máy tính.
- Về cấu tạo, vi kiến trúc được chia thành 2 khối:
 - Khối đường dữ liệu (datapath): Thực thi lệnh
 - Lưu trữ: Bộ nhớ lệnh, Bộ nhớ dữ liệu, Tập thanh ghi, ...
 - Truyền/nhận: Các đường tín hiệu dữ liệu, địa chỉ, điều khiển
 - Xử lý: ALU, Bộ so sánh, Mux, Bộ mở rộng dấu, Bộ dịch, ...
 - Khối điều khiển (control unit): Điều khiển datapath hoạt động
 - Dựa trên opcode của lệnh và trạng thái của datapath



NỘI DUNG

- Kiến trúc tổng quan bộ xử lý
- **Datapath**
- Thực thi lệnh
- Bài tập



Hình 1: Kiến trúc của 1 máy tính



Datapath

- Datapath dùng để thực thi lệnh! Một lệnh thực thi như thế nào?
 - Chu kỳ thực thi lệnh!



- Chương này chỉ xem xét 8 lệnh trong 3 nhóm chính của tập lệnh MIPS:
 - Nhóm lệnh tham khảo bộ nhớ (lw và sw)
 - Nhóm lệnh liên quan đến logic và số học (add, sub, and, or và slt)
 - Nhóm lệnh rẽ nhánh (lệnh rẽ nhánh với điều kiện bằng beq)



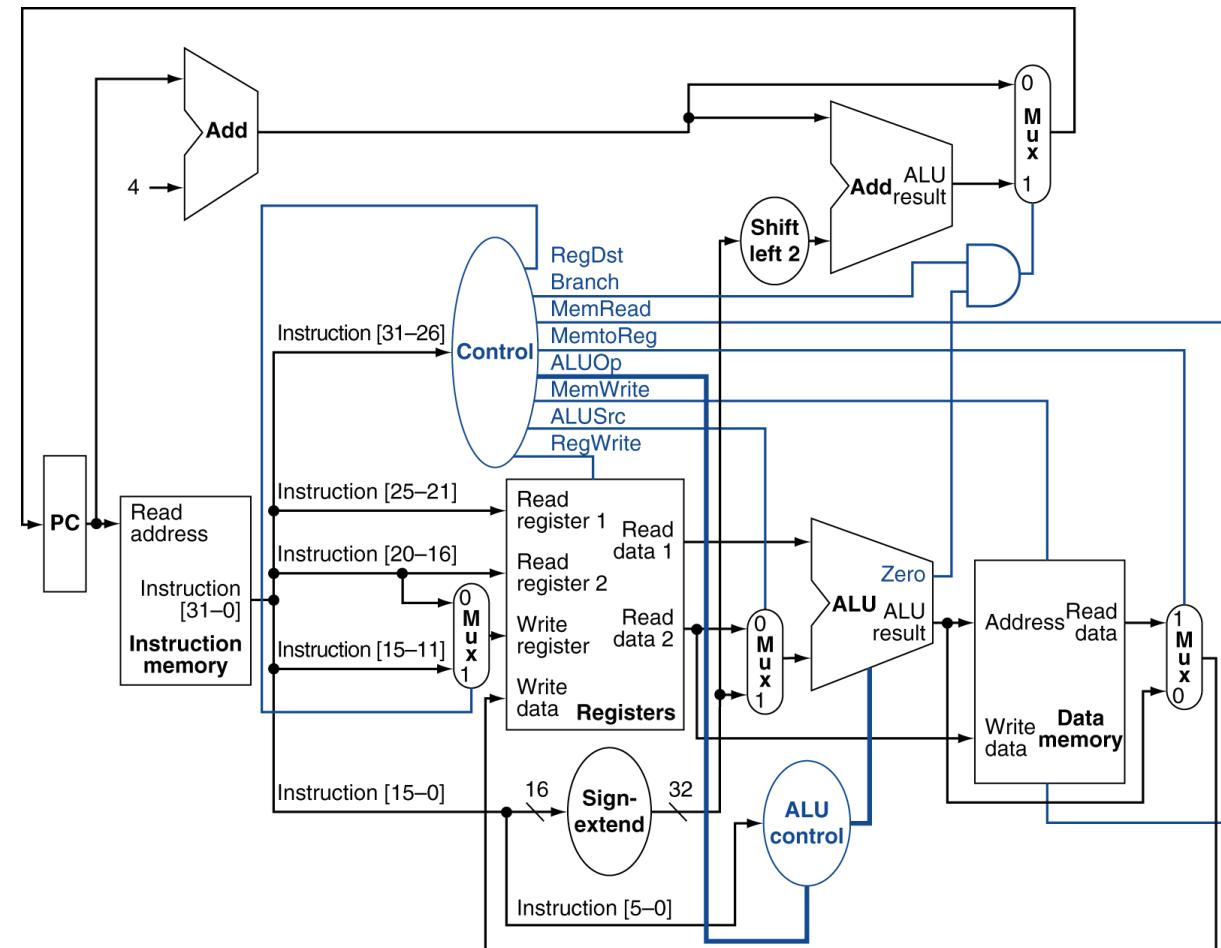
Quy trình thực thi lệnh của MIPS

	add \$3, \$1, \$2	lw \$3, 20(\$1)	beq \$1, \$2, label
Fetch (Nạp lệnh)	Đọc lệnh (địa chỉ của lệnh lưu trong thanh ghi PC)	Đọc lệnh (địa chỉ của lệnh lưu trong thanh ghi PC)	Đọc lệnh (địa chỉ của lệnh lưu trong thanh ghi PC)
Decode & Operand Fetch (Giải mã)	<ul style="list-style-type: none"> ○ Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i> ○ Đọc thanh ghi \$2, xem như toán hạng <i>opr2</i> 	<ul style="list-style-type: none"> ○ Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i> ○ Sử dụng 20 như toán hạng <i>opr2</i> 	<ul style="list-style-type: none"> ○ Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i> ○ Đọc thanh ghi \$2, xem như toán hạng <i>opr2</i>
ALU (Thực thi)	$Result = opr1 + opr2$	$MemAddr = opr1 + opr2$	$Taken = (opr1 == opr2) ?$ $Target = PC + Label^*$
Memory Access (Truy xuất bộ nhớ)		Sử dụng <i>MemAddr</i> để đọc dữ liệu từ bộ nhớ	
Result Write (Lưu kết quả)	<i>Result</i> được lưu trữ vào \$3	Dữ liệu của từ nhớ có địa chỉ <i>MemAddr</i> được được lưu trữ vào \$3	if (<i>Taken</i>) PC = <i>Target</i>



Datapath (2)

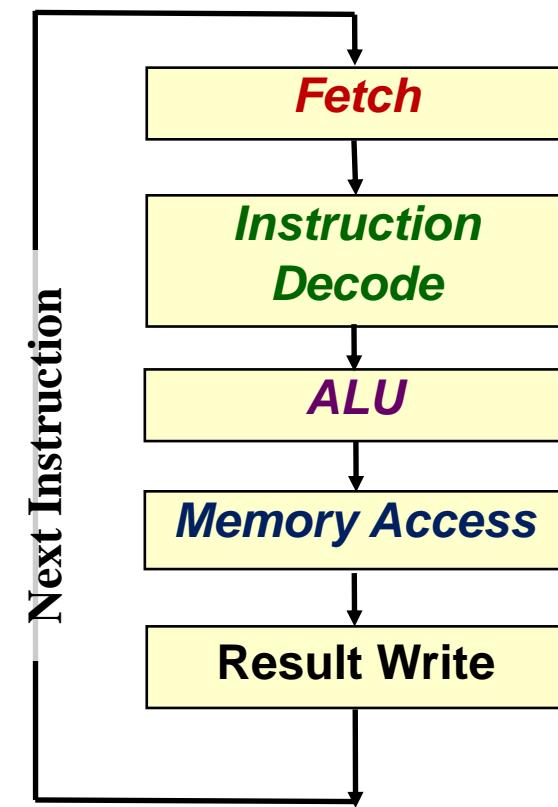
Hình ảnh
datapath của một
bộ xử lý với 8
lệnh MIPS: add,
sub, and, or, slt,
lw, sw và beq





Quy trình thực thi lệnh của MIPS

- **Instruction Fetch (Nạp lệnh)**
- Instruction Decode & Operand Fetch
(Giải mã và lấy các toán hạng cần thiết, Gọi tắt là “Instruction Decode”)
- ALU (Giai đoạn sử dụng ALU hay giai đoạn thực thi)
- Memory Access (Giai đoạn truy xuất vùng nhớ)
- Result Write (Giai đoạn ghi lại kết quả/lưu trữ)



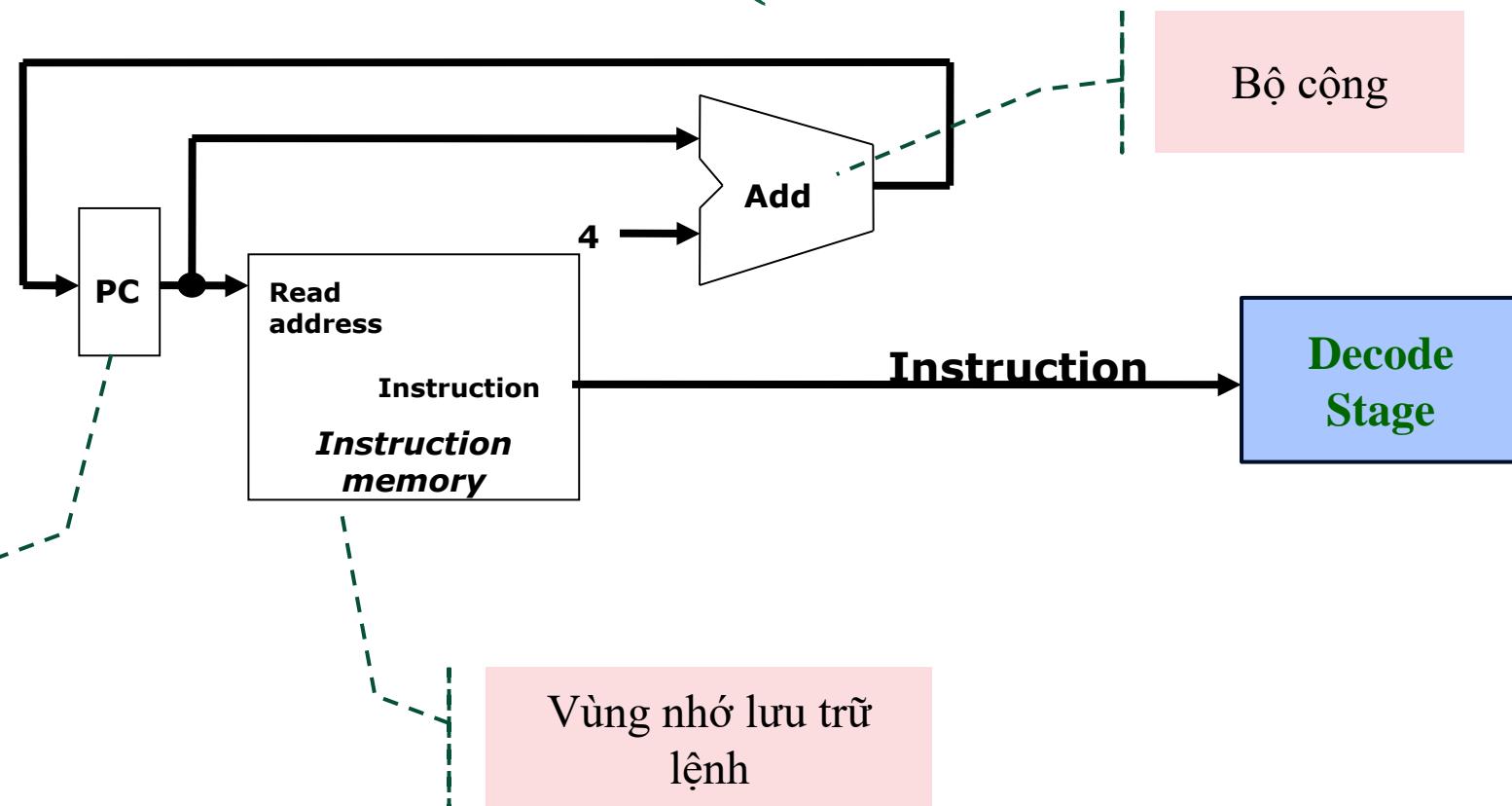


Giai đoạn tìm nạp lệnh (Instruction Fetch)

- Giai đoạn nạp lệnh:
 - Sử dụng thanh ghi Program Counter (PC) để tìm nạp lệnh từ bộ nhớ
 - Thanh ghi PC là một thanh ghi đặc biệt trong bộ vi xử lý
 - Tăng giá trị trong thanh ghi PC lên 4 đơn vị để lấy địa chỉ của lệnh tiếp theo
 - Chú ý, lệnh rẽ nhánh (branch) và lệnh nhảy (jump) là một trường hợp ngoại lệ
- Kết quả của giai đoạn này là đầu vào cho giai đoạn tiếp theo (Decode):
 - Kết quả của giai đoạn này là 32 bit mã máy của lệnh cần thực thi. Chuỗi 32 bits này sẽ sử dụng như đầu vào (input) cho giai đoạn tiếp theo là



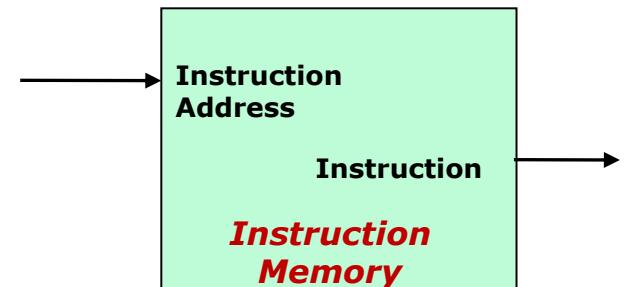
Giai đoạn tìm nạp lệnh (Instruction Fetch)





Khối Instruction Memory

- Vùng nhớ lưu trữ lệnh
- Đầu vào: là địa chỉ của lệnh
- Đầu ra: là nội dung lệnh tương ứng với địa chỉ được cung cấp



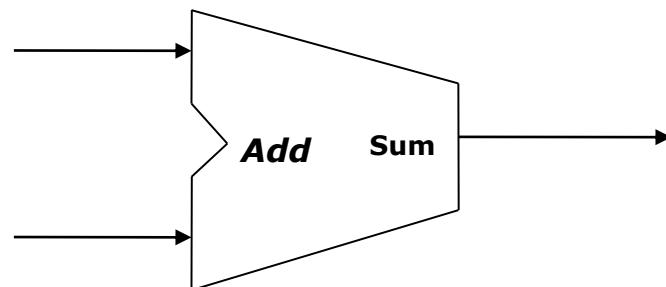
Cách sắp xếp của bộ nhớ giống như hình bên phải

Instruction Memory	
.....	
2048	add \$3, \$1, \$2
2052	sll \$4, \$3, 2
2056	andi \$1, \$4, 0xF
.....



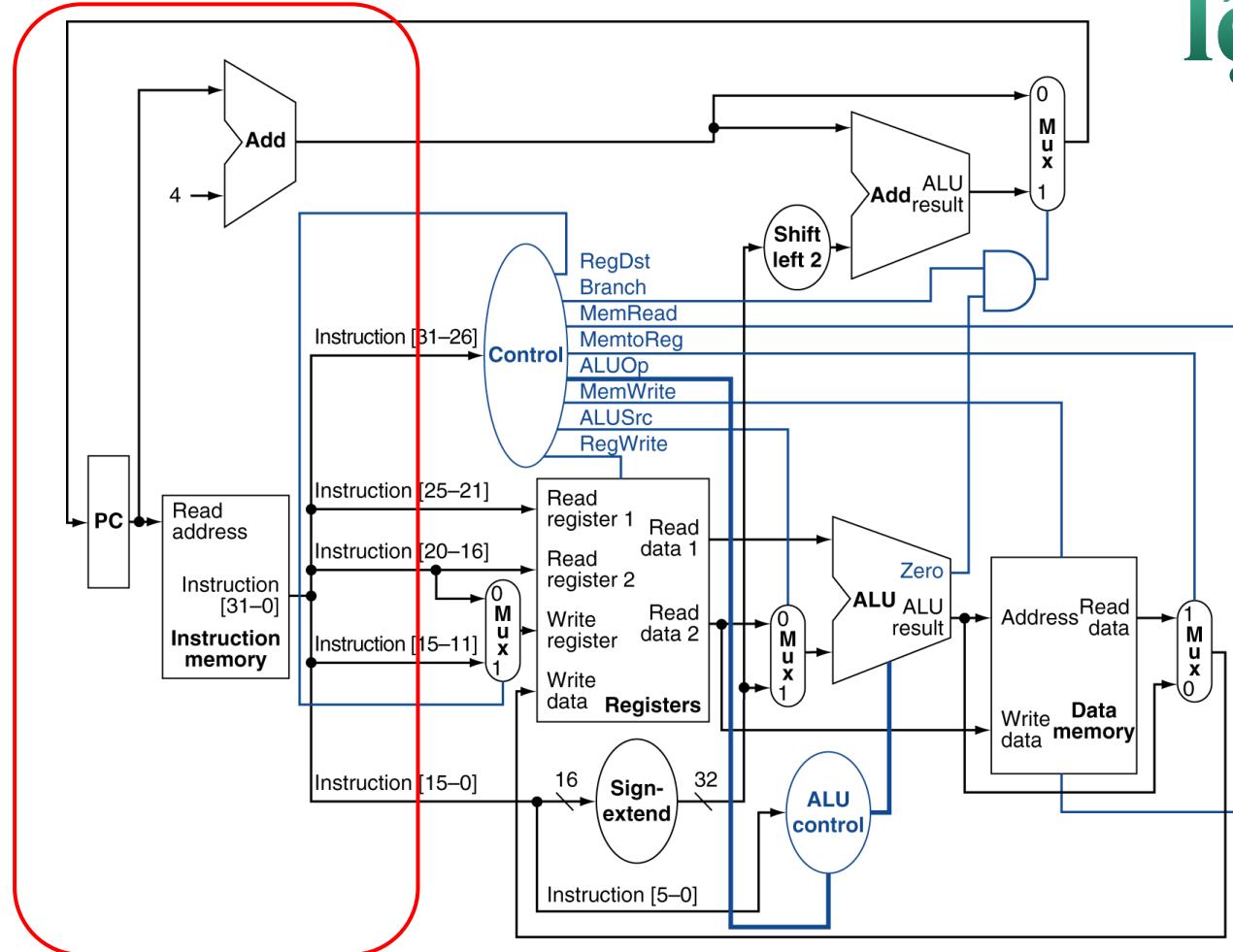
Bộ cộng

- Mạch logic kết hợp để cộng 2 số - bộ cộng
- Đầu vào:
 - Hai số 32-bit A, B
- Đầu ra:
 - $A + B$





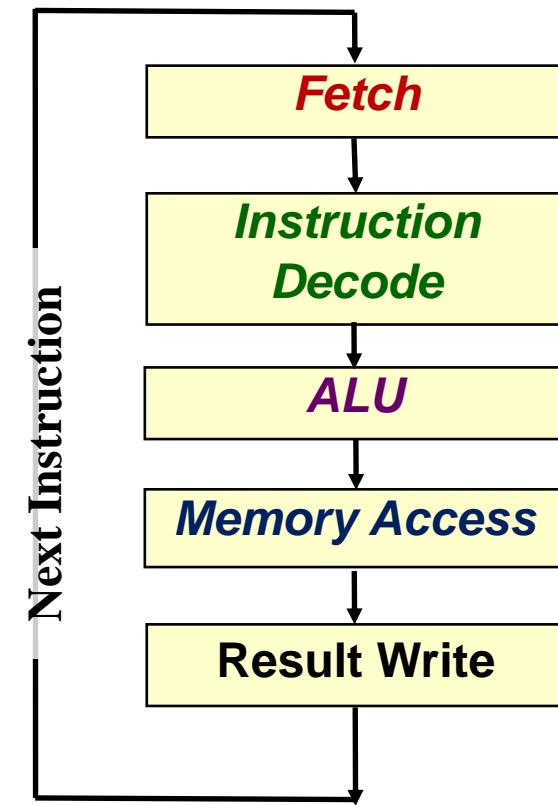
Datapath của một bộ xử lý với 8 lệnh MIPS





Quy trình thực thi lệnh của MIPS

- Instruction Fetch (Nạp lệnh)
- **Instruction Decode & Operand Fetch**
(Giải mã và lấy các toán hạng cần thiết, Gọi tắt là “Instruction Decode”)
- ALU (Giai đoạn sử dụng ALU hay giai đoạn thực thi)
- Memory Access (Giai đoạn truy xuất vùng nhớ)
- Result Write (Giai đoạn ghi lại kết quả/lưu trữ)



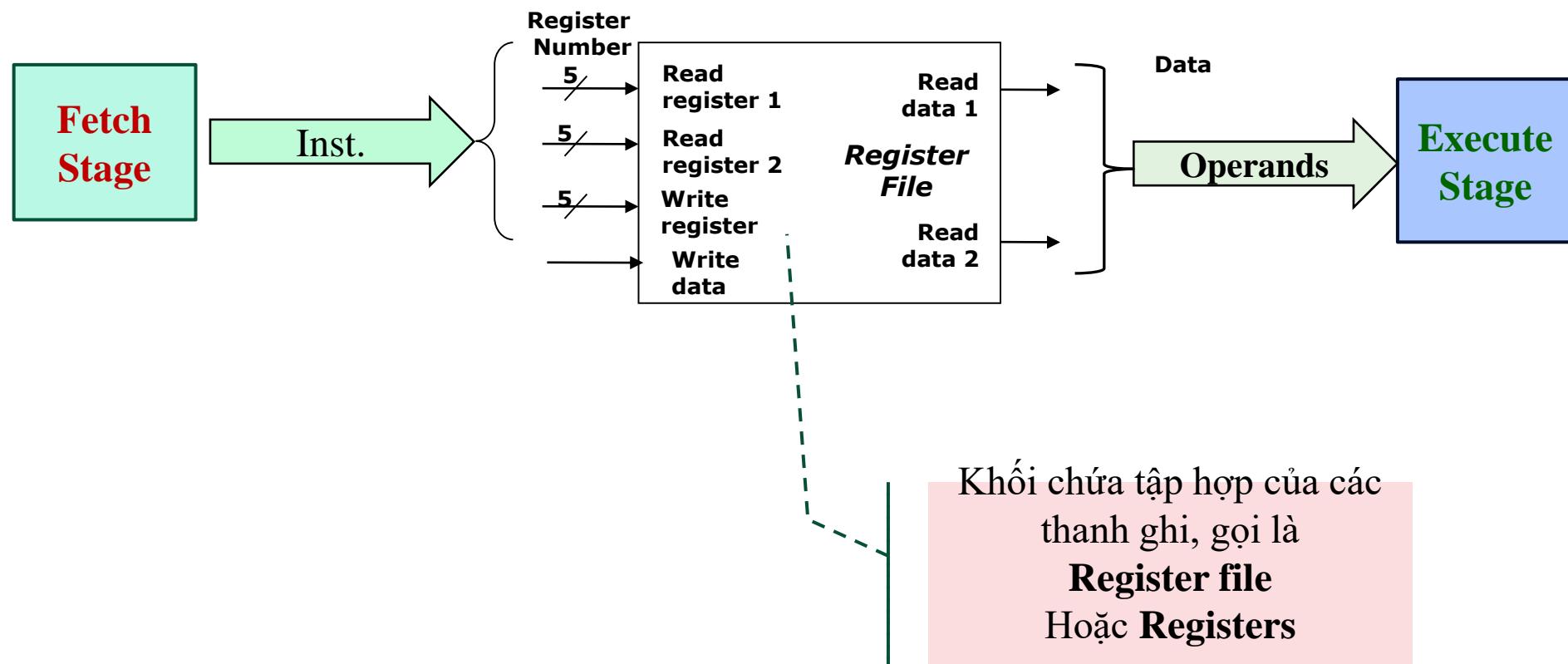


Giai đoạn giải mã (Decode)

- Giai đoạn decode:
 - Lấy nội dung dữ liệu trong các trường (field) của lệnh:
 - Đọc opcode để xác định kiểu lệnh và chiều dài của từng trường trong mã máy
 - Đọc dữ liệu từ các thanh ghi cần thiết
 - Có thể 2 (lệnh add), 1 (lệnh addi) hoặc 0 (lệnh j)
- Đầu vào từ giai đoạn trước (Fetch):
 - Lệnh cần được thực thi (Mã máy)
- Đầu ra cho giai đoạn tiếp theo (Execute):
 - Phép tính và các toán hạng cần thiết

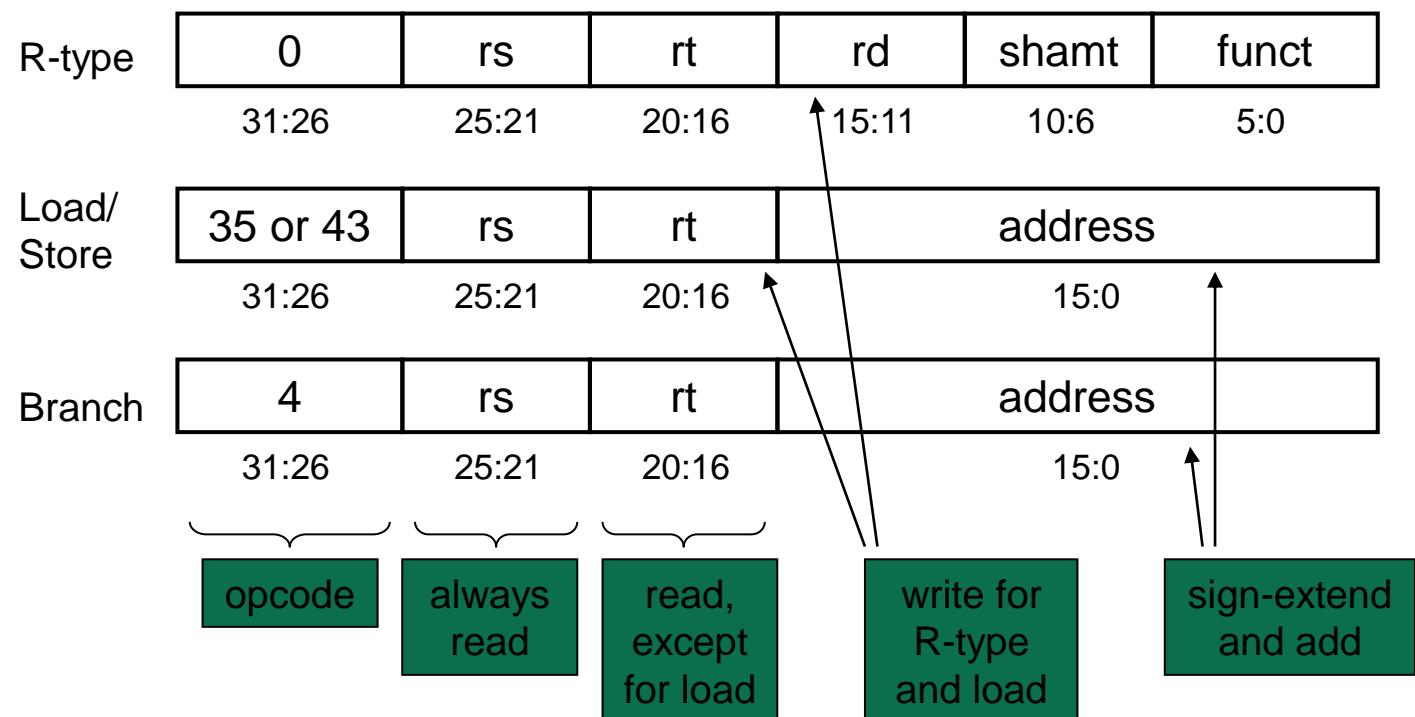


Giai đoạn giải mã (Decode)



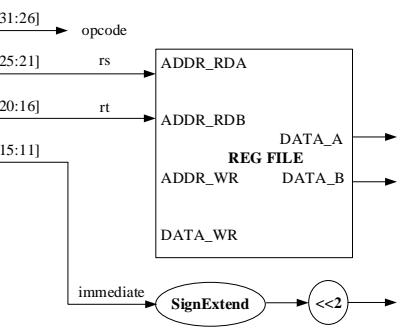
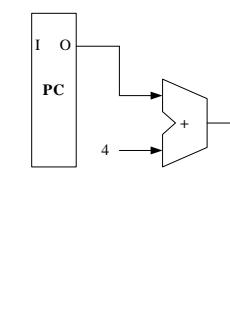
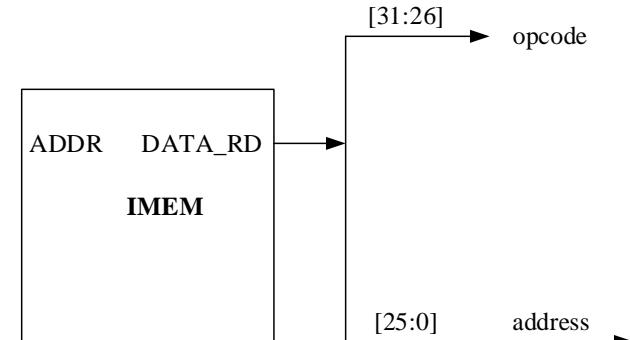
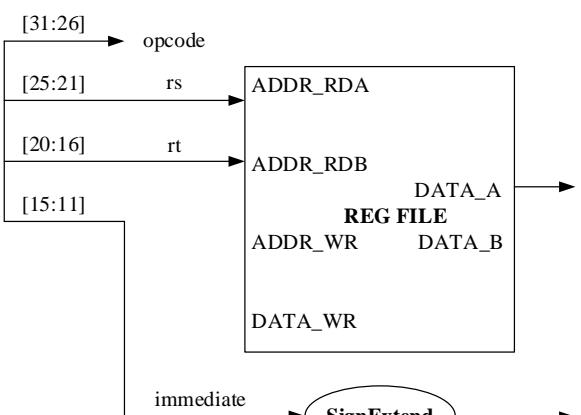
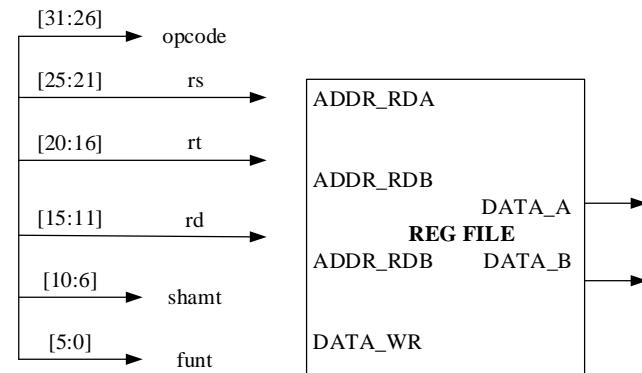
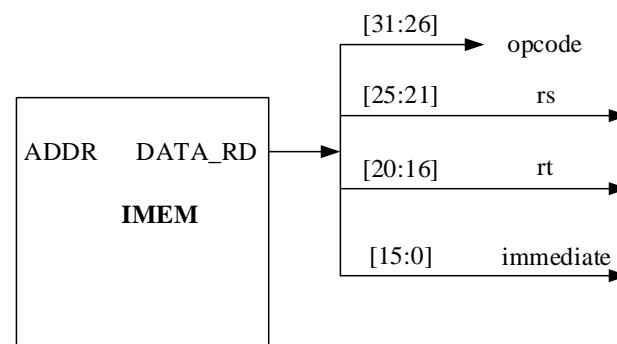
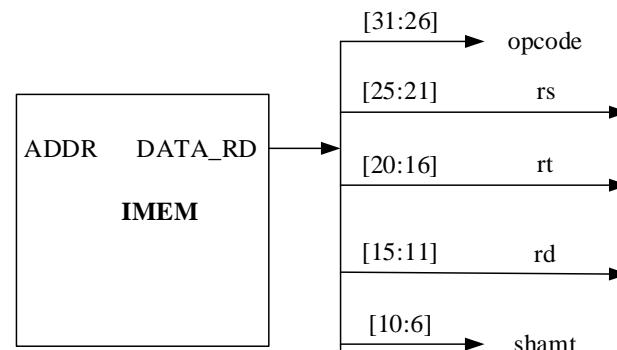


Giai đoạn giải mã (Decode)





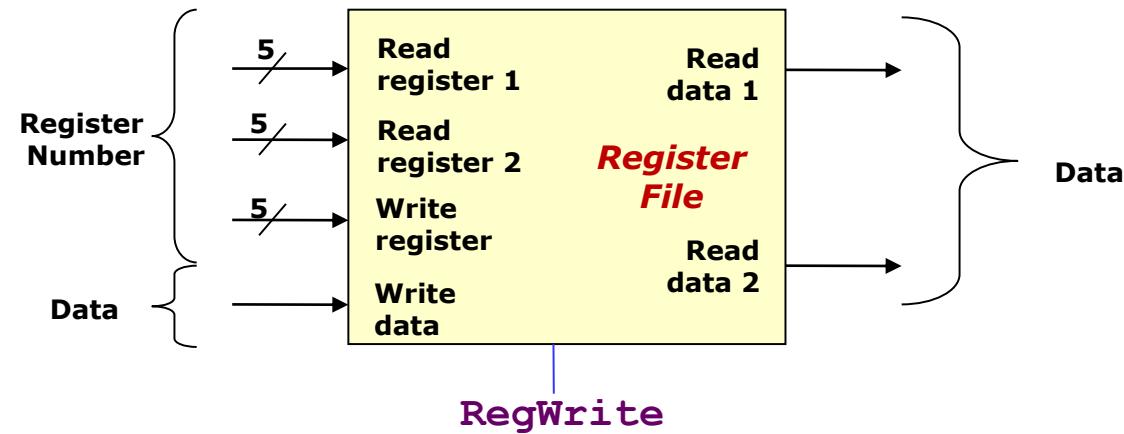
Giai đoạn giải mã (Decode)





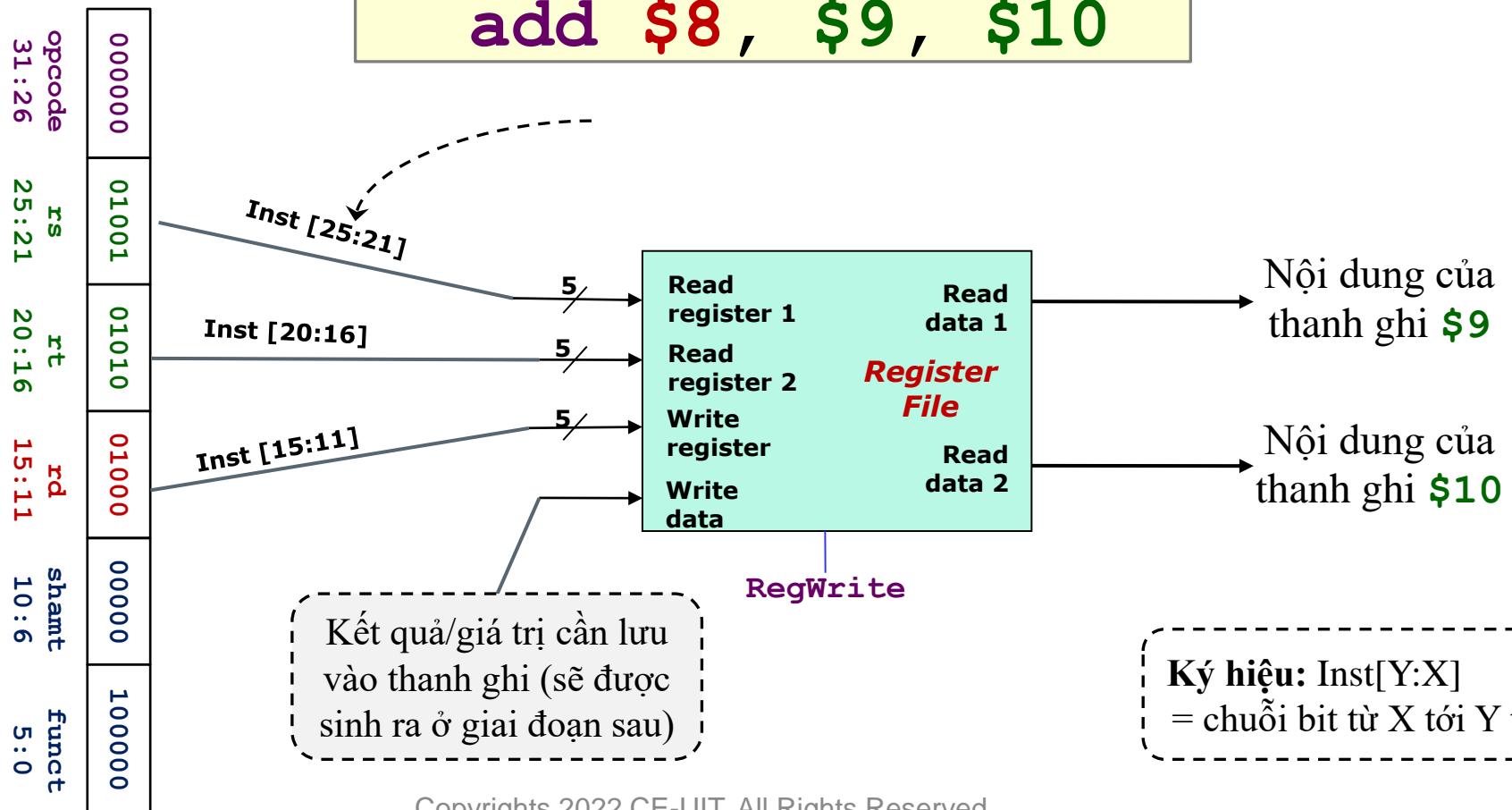
Khối Register File

- Một tập 32 thanh ghi:
 - Mỗi thanh ghi có chiều dài 32 bit và có thể được đọc hoặc ghi bằng cách chỉ ra chỉ số của thanh ghi
 - Với mỗi lệnh, cho phép đọc nhiều nhất từ 2 thanh ghi
 - Với mỗi lệnh, cho phép ghi vào nhiều nhất 1 thanh ghi
- RegWrite: là một tín hiệu điều khiển nhằm mục đích:
 - Cho phép ghi vào một thanh ghi hay không



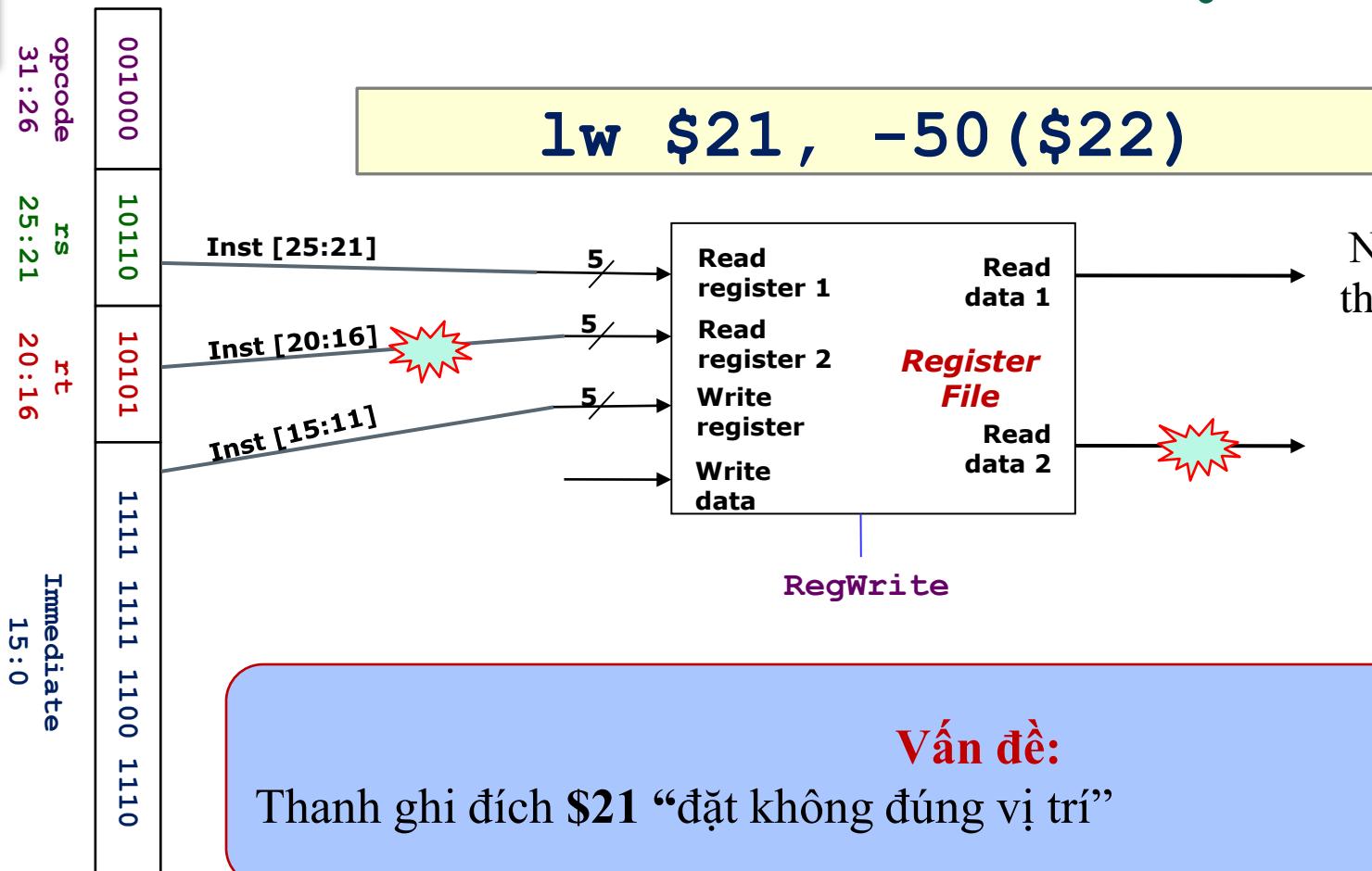


Giải mã: lệnh R-Type



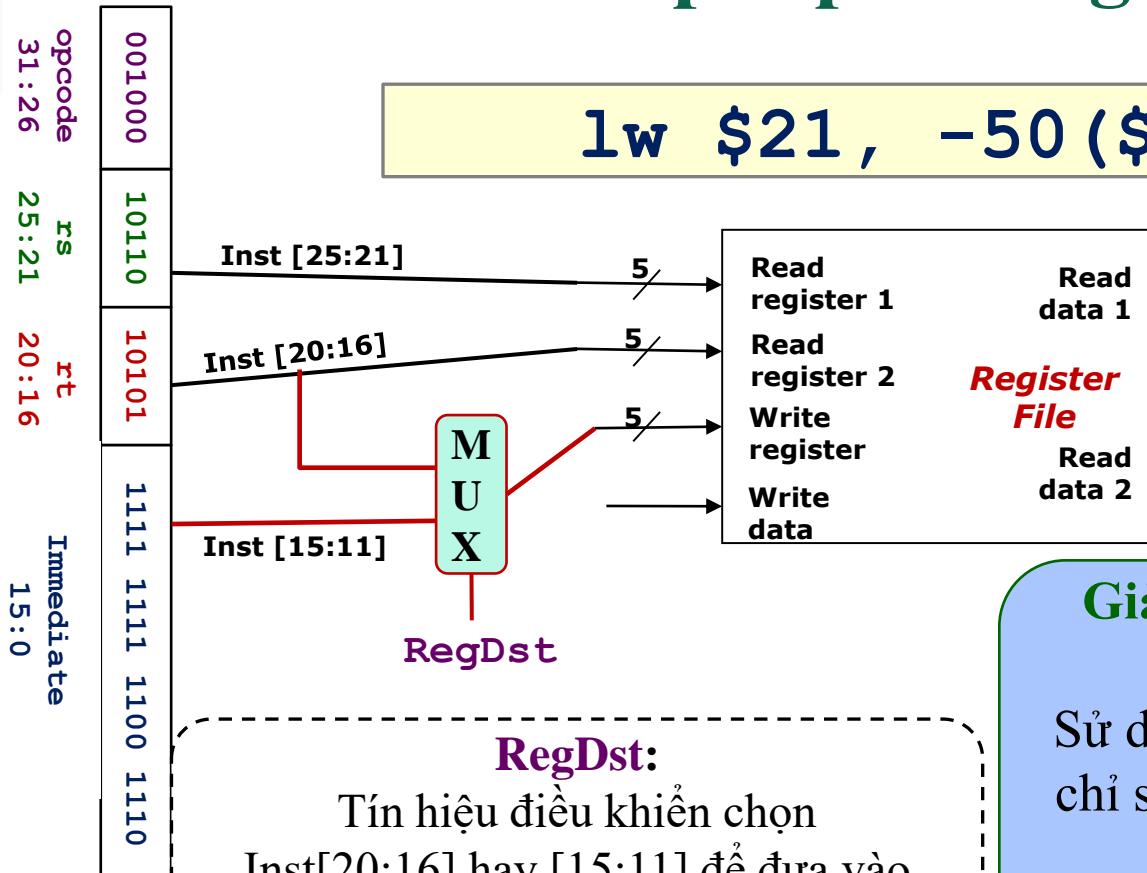


Giải mã: lệnh I-Type





Giải mã: Giải pháp cho ngõ “Write register”



RegDst:

Tín hiệu điều khiển chọn
Inst[20:16] hay [15:11] để đưa vào
ngõ write register

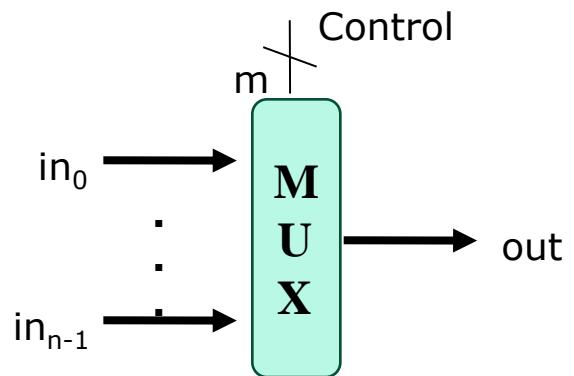
**Giải pháp (cho chỉ số thanh ghi sẽ
được ghi):**

Sử dụng một **multiplexer** để lựa chọn
chỉ số thanh ghi cho ngõ write register
chính xác dựa
trên từng loại lệnh



Khối Mux

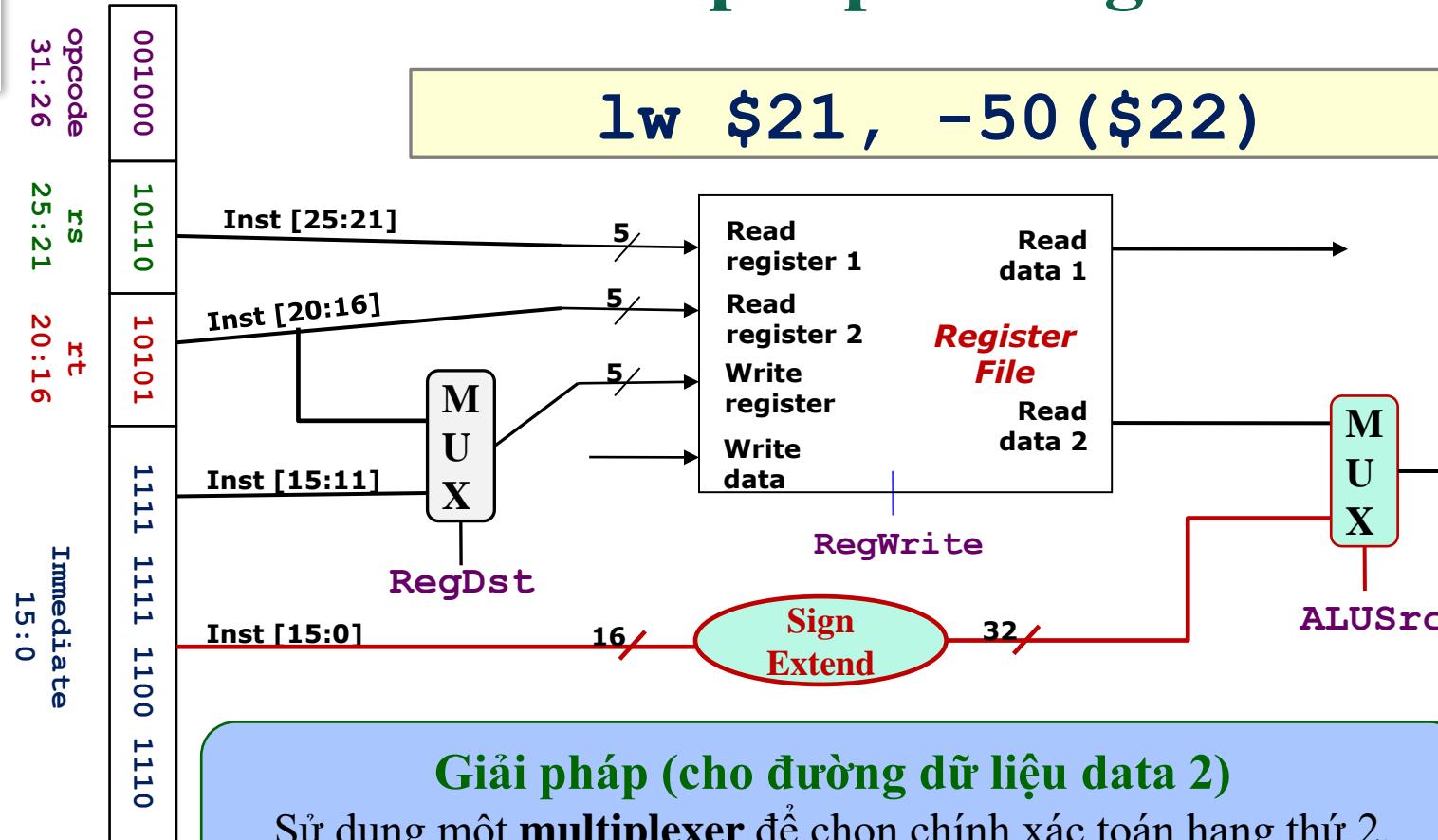
- Chức năng:
 - Chọn một input từ tập input đầu vào
- Inputs:
 - n đường vào có cùng chiều rộng
- Control:
 - Cần m bit trong đó $n = 2^m$
- Output:
 - Chọn đường input thứ i nếu giá trị tín hiệu điều khiển $control = i$



$Control=0 \rightarrow$ select in_0
 $Control=3 \rightarrow$ select in_3



Giải mã: Giải pháp cho ngo “Toán hạng 2”



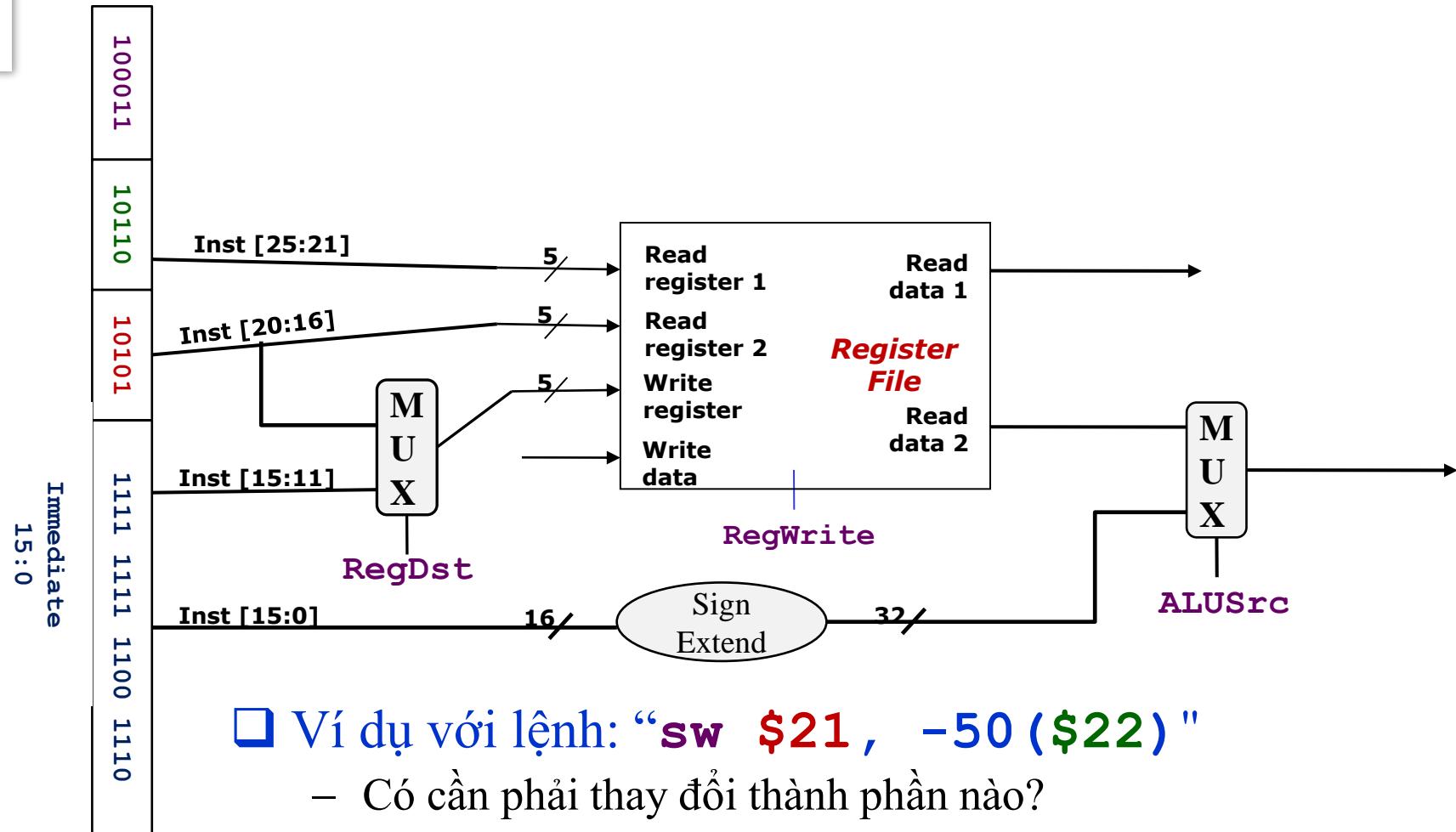
Giải pháp (cho đường dữ liệu data 2)

Sử dụng một multiplexer để chọn chính xác toán hạng thứ 2.
Sign extend: khôi mở rộng số tức thời 16 bit thành 32 bit

ALUSrc:
 Tín hiệu điều khiển để chọn “Read data 2” hay giá trị của Inst[15:0] (đã được mở rộng có dấu) cho toán hạng thứ hai



Giải mã: cho lệnh sw



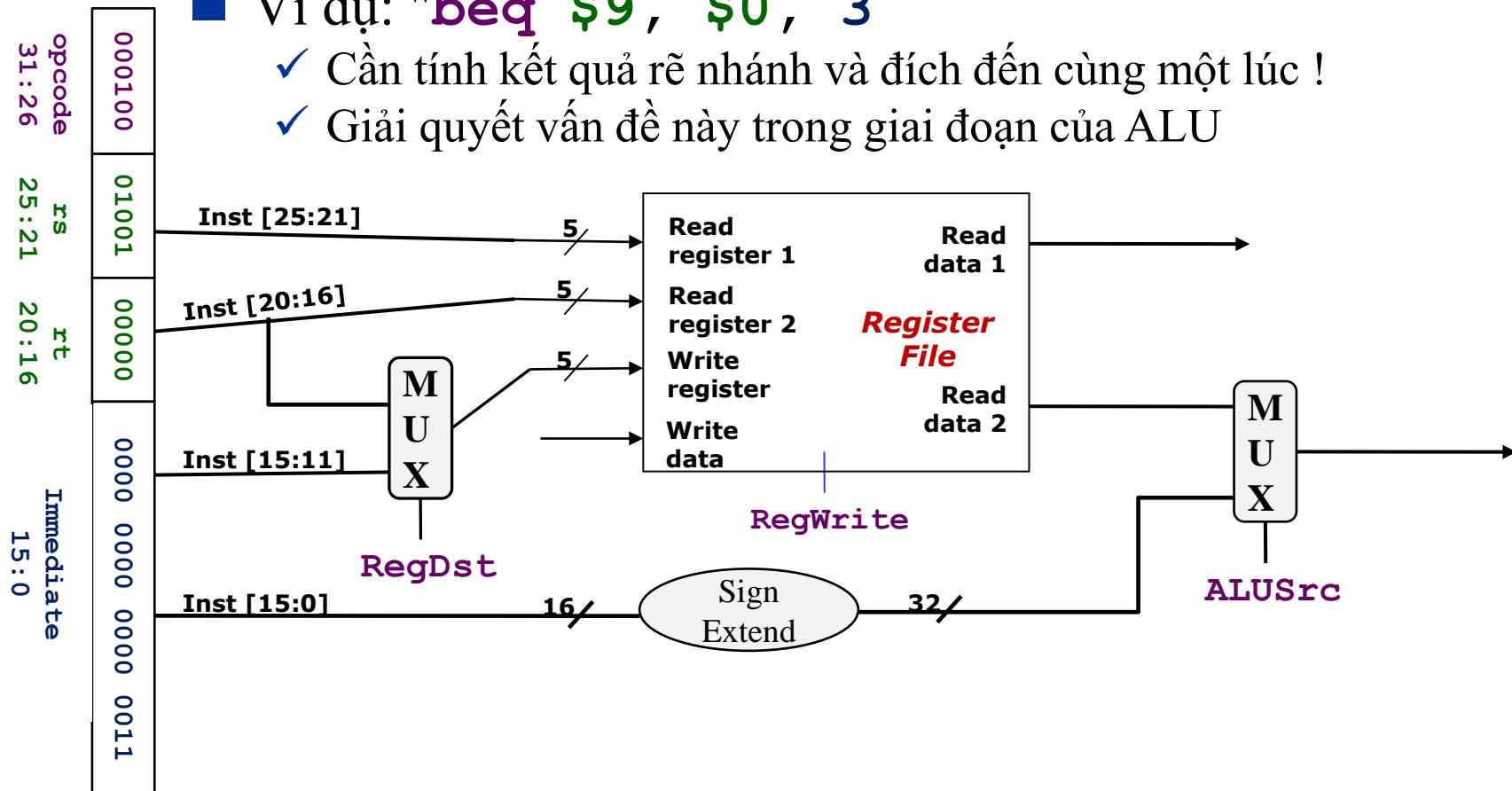
- ❑ Ví dụ với lệnh: “**sw \$21, -50(\$22)**”
- Có cần phải thay đổi thành phần nào?



Giải mã: cho lệnh beq

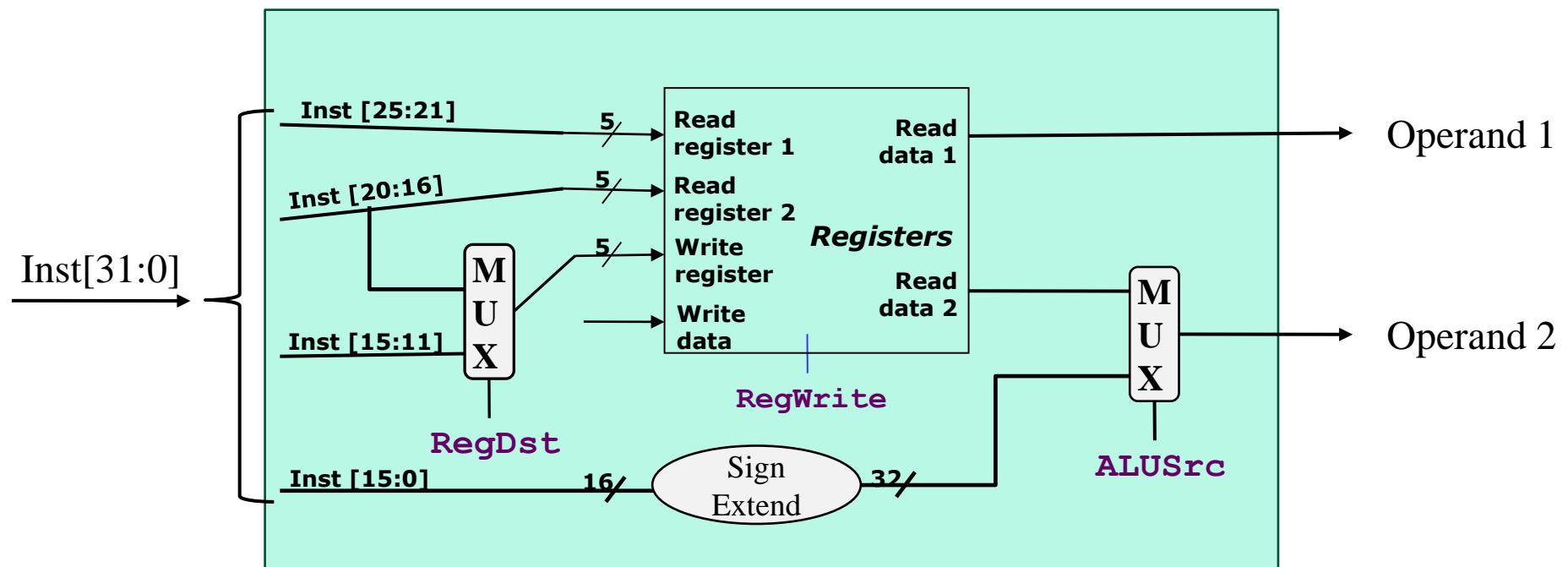
■ Ví dụ: "beq \$9, \$0, 3"

- ✓ Cần tính kết quả rẽ nhánh và đích đến cùng một lúc !
- ✓ Giải quyết vấn đề này trong giai đoạn của ALU



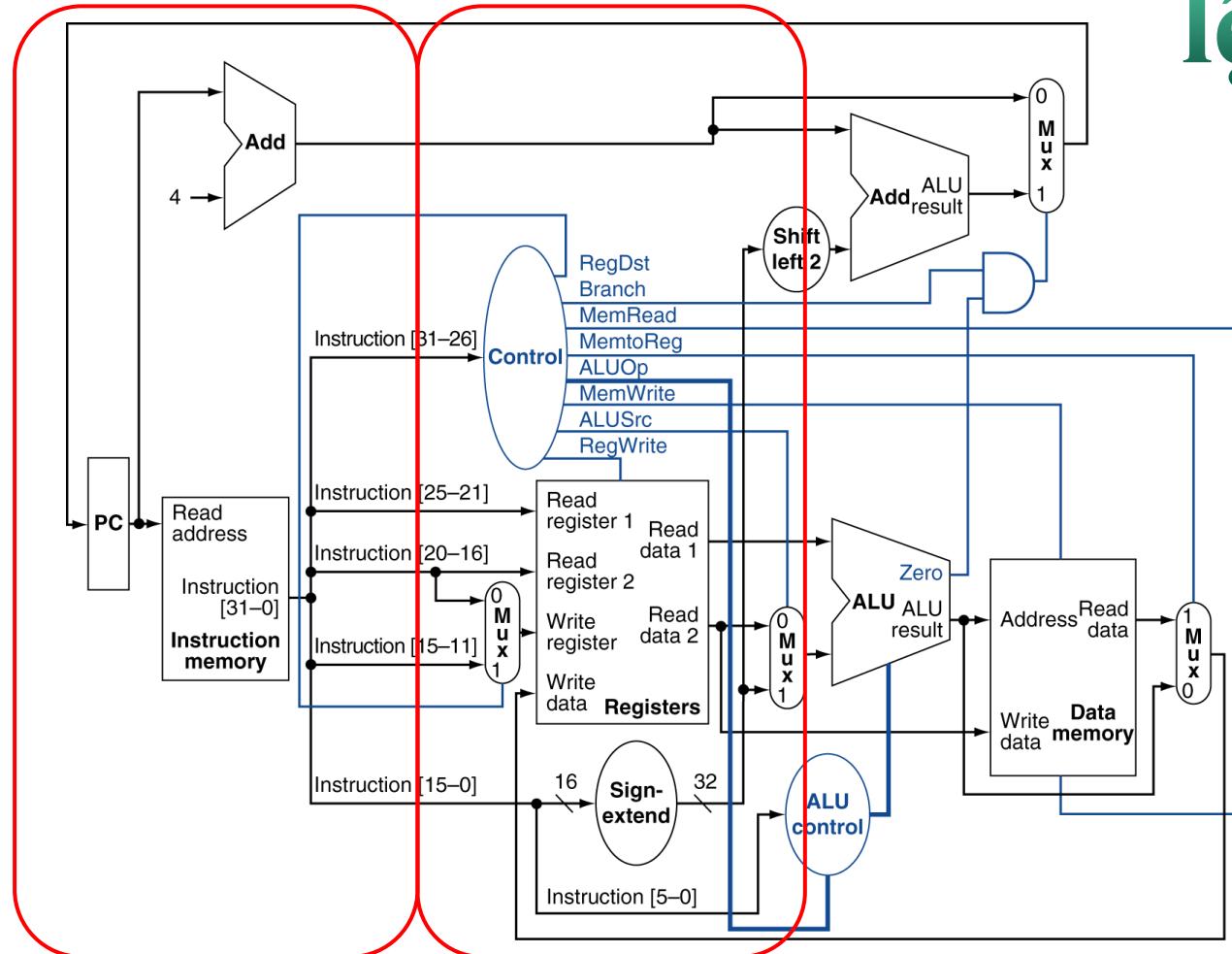


Giải mã: tổng hợp





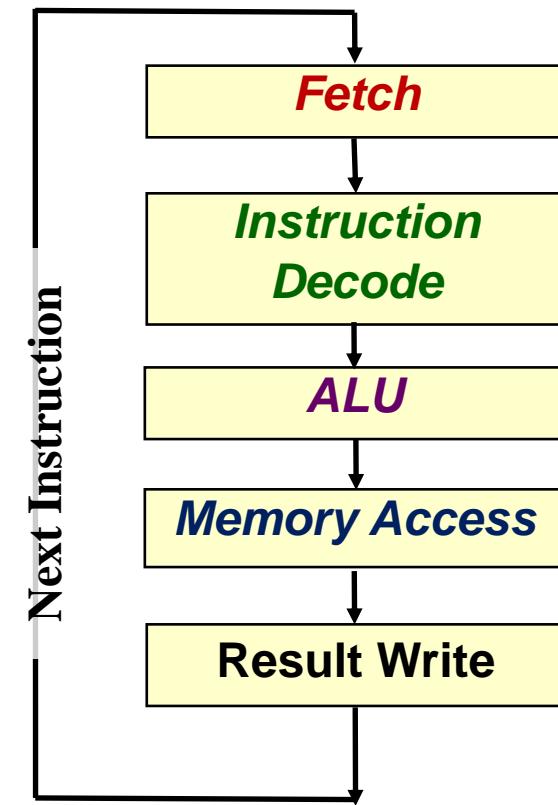
Datapath của một bộ xử lý với 8 lệnh MIPS





Quy trình thực thi lệnh của MIPS

- Instruction Fetch (Nạp lệnh)
- Instruction Decode & Operand Fetch
(Giải mã và lấy các toán hạng cần thiết, Gọi tắt là “Instruction Decode”)
- **ALU (Giai đoạn sử dụng ALU hay giai đoạn thực thi)**
- Memory Access (Giai đoạn truy xuất vùng nhớ)
- Result Write (Giai đoạn ghi lại kết quả/lưu trữ)



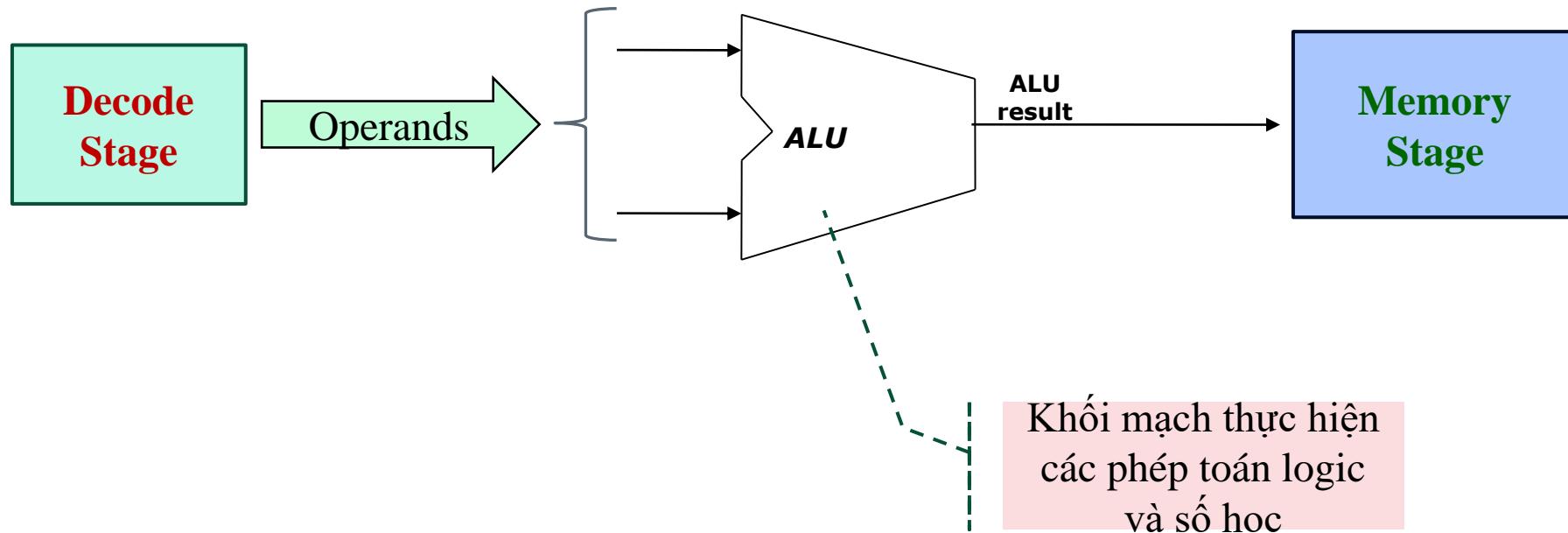


Công đoạn ALU (thực thi)

- Công đoạn ALU:
 - ALU = Arithmetic-Logic Unit
 - Công việc thật sự của hầu hết các lệnh được hiện chủ yếu trong giai đoạn này
 - Số học (Arithmetic) (ví dụ: add, sub), Logic (ví dụ: and, or): ALU tính ra kết quả cuối cùng
 - Lệnh làm việc với bộ nhớ (ví dụ: lw, sw): ALU dùng tính toán địa chỉ của bộ nhớ
 - Lệnh nhảy/nhánh (ví dụ: bne, beq): ALU thực hiện so sánh các giá trị trên thanh ghi và tính toán địa chỉ đích sẽ nhảy tới
- Đầu vào từ công đoạn trước (Decode):
 - Các thao tác (operation) và toán hạng (operand(s))
- Đầu ra cho công đoạn tiếp theo (Memory):
 - Tính toán kết quả
(Đối với lệnh lw và sw: Kết quả của công đoạn này sẽ là địa chỉ cung cấp cho memory để lấy dữ liệu)



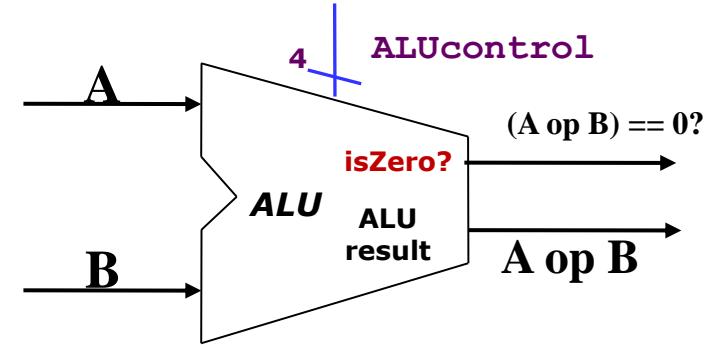
Công đoạn ALU (thực thi)





Khối ALU (Arithmetic Logical Unit)

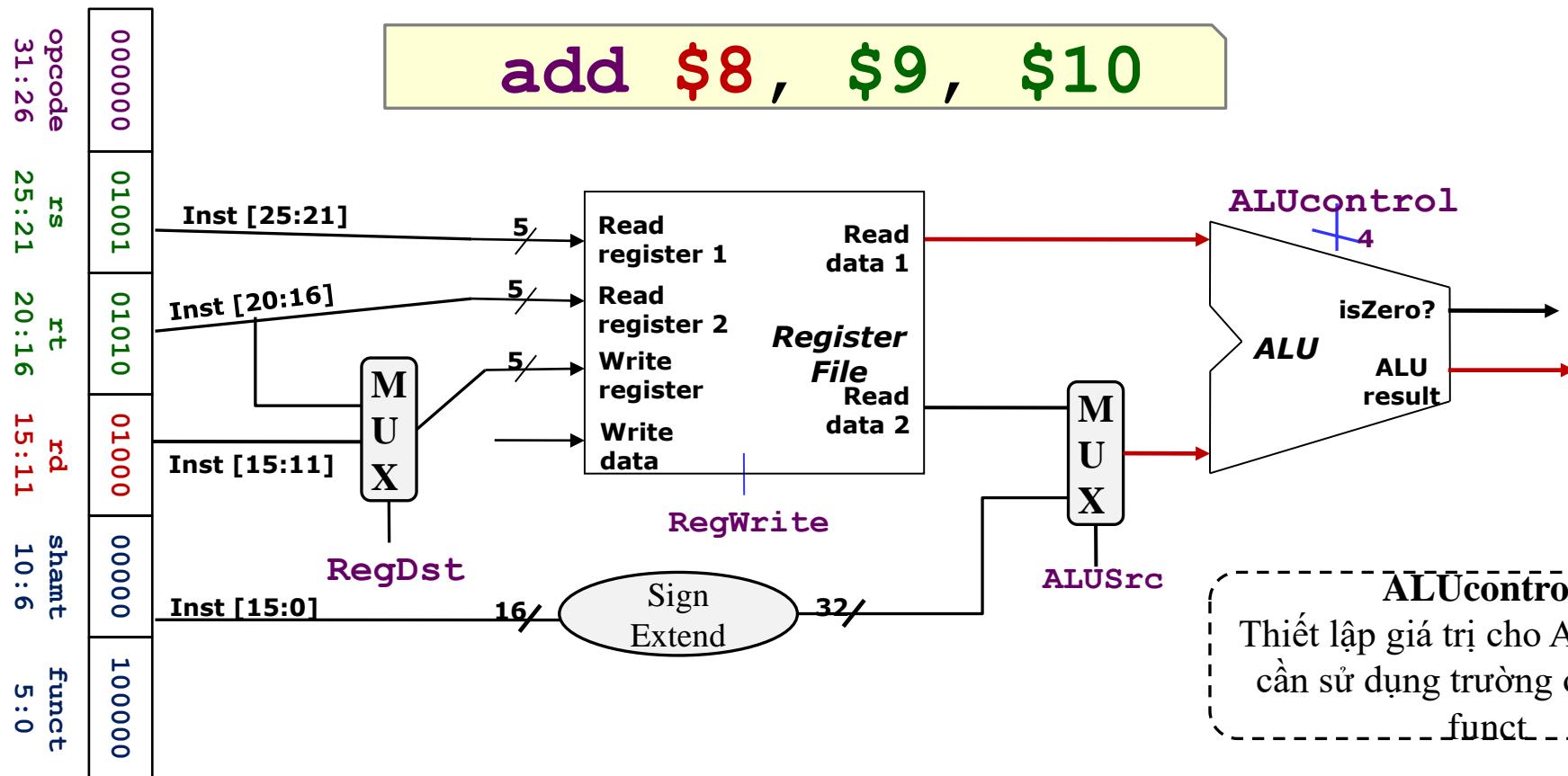
- ALU (Arithmetic-logical unit)
 - Khối dùng để thực hiện các phép tính logic và số học
- Inputs:
 - 2 số 32-bit
- Điều khiển khối ALU:
 - Do ALU có thể thực hiện nhiều chức năng -> dùng 4-bit để quyết định chức năng/phép toán cụ thể nào cho ALU
- Outputs:
 - Kết quả của phép toán số học hoặc logic
 - Một bit tín hiệu để chỉ ra rằng kết quả có bằng 0 hay không



ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR



Công đoạn ALU: các lệnh non-branch





Quy trình thực thi lệnh của MIPS

	add \$3, \$1, \$2	lw \$3, 20(\$1)	beq \$1, \$2, label
Fetch (Nạp lệnh)	Đọc lệnh (địa chỉ của lệnh lưu trong thanh ghi PC)	Đọc lệnh (địa chỉ của lệnh lưu trong thanh ghi PC)	Đọc lệnh (địa chỉ của lệnh lưu trong thanh ghi PC)
Decode & Operand Fetch (Giải mã)	<ul style="list-style-type: none"> ○ Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i> ○ Đọc thanh ghi \$2, xem như toán hạng <i>opr2</i> 	<ul style="list-style-type: none"> ○ Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i> ○ Sử dụng 20 như toán hạng <i>opr2</i> 	<ul style="list-style-type: none"> ○ Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i> ○ Đọc thanh ghi \$2, xem như toán hạng <i>opr2</i>
ALU (Thực thi)	$Result = opr1 + opr2$	$MemAddr = opr1 + opr2$	$Taken = (opr1 == opr2) ?$ $Target = PC + Label^*$
Memory Access (Truy xuất bộ nhớ)		Sử dụng <i>MemAddr</i> để đọc dữ liệu từ bộ nhớ	
Result Write (Lưu kết quả)	<i>Result</i> được lưu trữ vào \$3	Dữ liệu của từ nhớ có địa chỉ <i>MemAddr</i> được được lưu trữ vào \$3	if (<i>Taken</i>) PC = <i>Target</i>



Công đoạn ALU: Các lệnh Branch

- Lệnh rẽ nhánh thì khó hơn vì phải tính toán hai phép toán:
 - Ví dụ: "beq \$9, \$0, 3"
 - Kết quả rẽ nhánh:
 - Sử dụng ALU để so sánh thanh ghi
 - Tín hiệu 1-bit "isZero?" để kiểm tra tính chất bằng/không bằng
 - Địa chỉ đích của nhánh:
 - Sử dụng một bộ cộng để tính địa chỉ
 - Cần nội dung của thanh ghi PC (từ Fetch Stage)
 - Cần Offset (từ Decode Stage)

Branch On Equal

beq

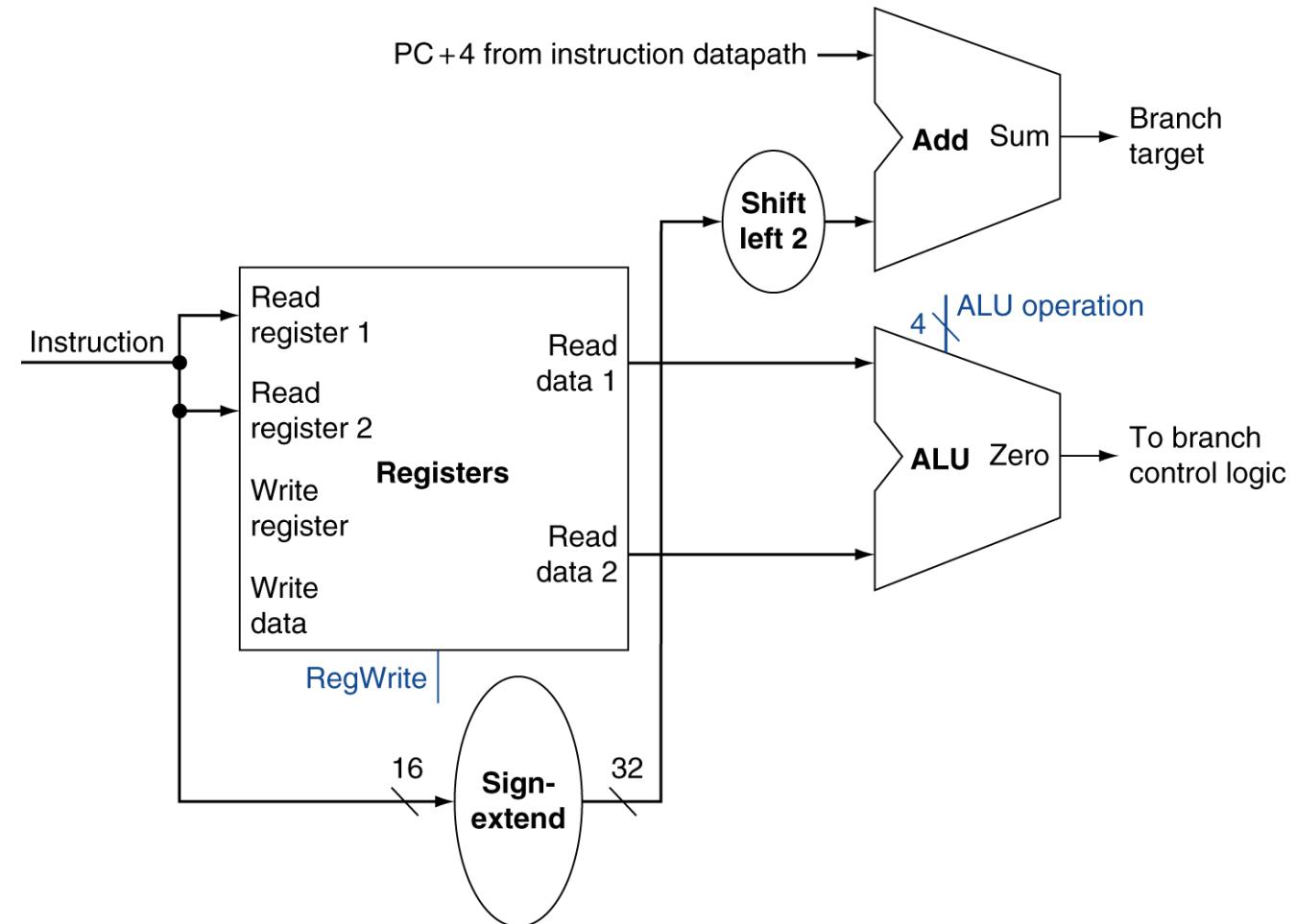
I if(R[rs]==R[rt])
 PC=PC+4+BranchAddr

(4) ${}^4\text{hex}$

39

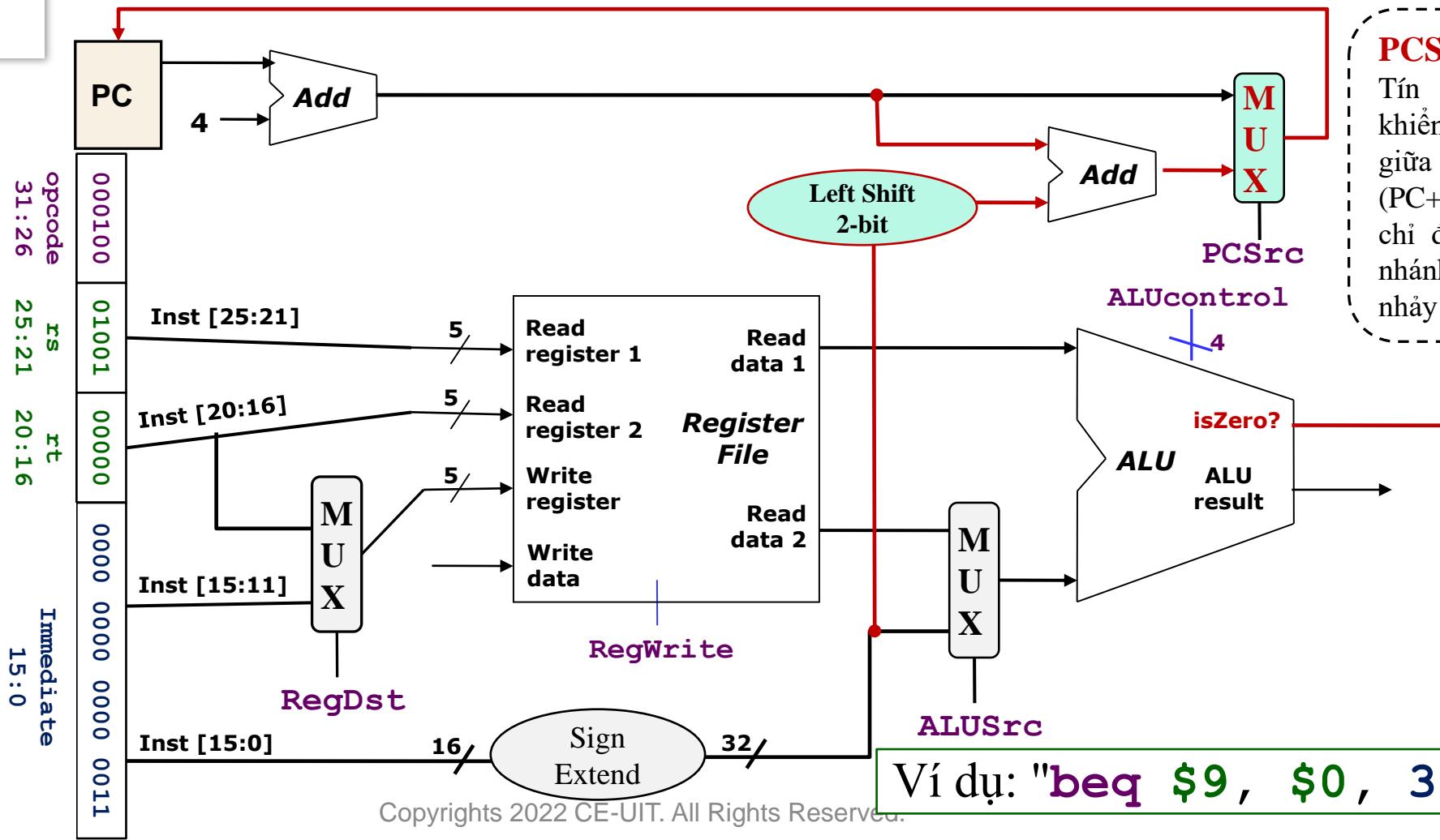


Công đoạn ALU: Các lệnh Branch



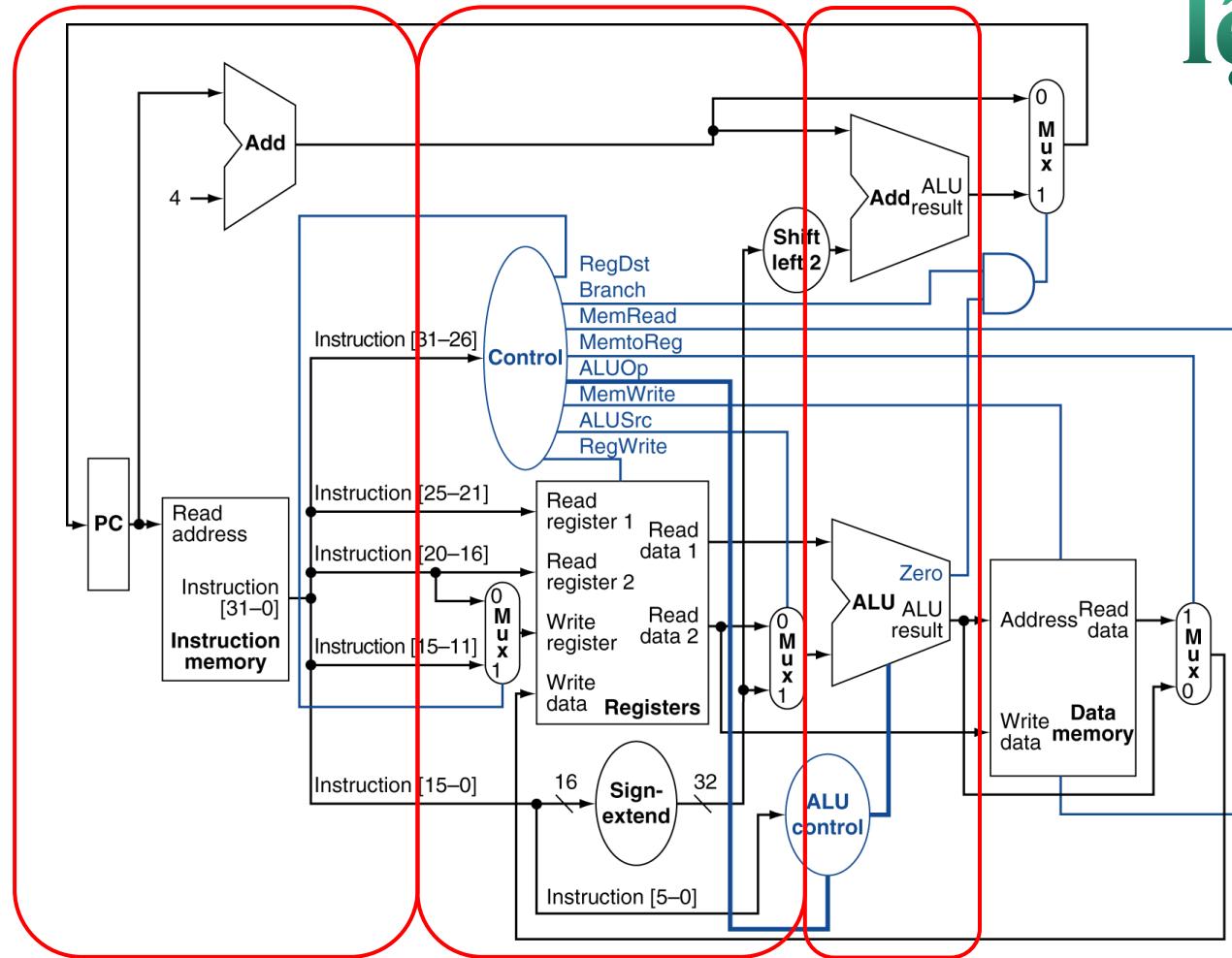


Datapath với công đoạn ALU hoàn chỉnh





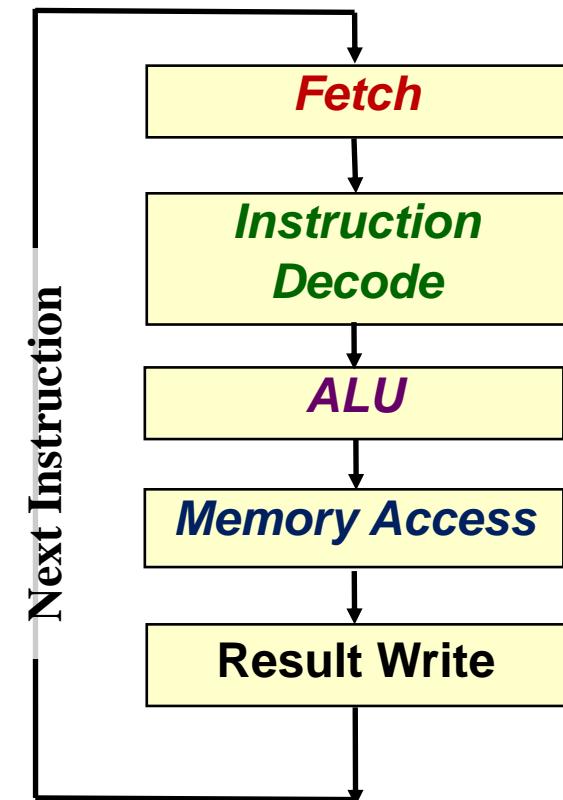
Datapath của một bộ xử lý với 8 lệnh MIPS





Quy trình thực thi lệnh của MIPS

- Instruction Fetch (Nạp lệnh)
- Instruction Decode & Operand Fetch
 - (Giải mã và lấy các toán hạng cần thiết, Gọi tắt là “Instruction Decode”)
- ALU (Giai đoạn sử dụng ALU hay giai đoạn thực thi)
- **Memory Access (Giai đoạn truy xuất vùng nhớ)**
- Result Write (Giai đoạn ghi lại kết quả/lưu trữ)



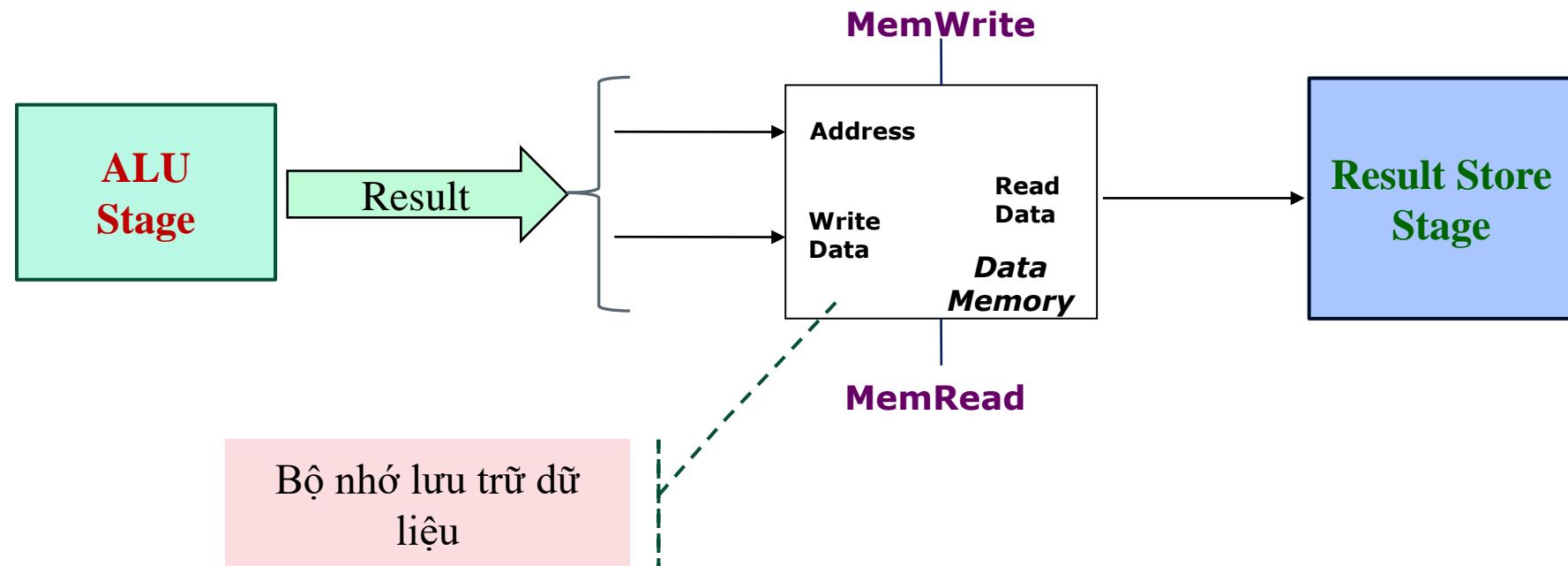


Giai đoạn truy xuất vùng nhớ (Memory stage)

- Giai đoạn truy xuất vùng nhớ:
 - Chỉ có lệnh Load và Store cần thực hiện các thao tác trong giai đoạn này:
 - Sử dụng địa chỉ vùng nhớ được tính toán ở giai đoạn ALU
 - Đọc dữ liệu ra hoặc ghi dữ liệu vào vùng nhớ dữ liệu
 - Tất cả các lệnh khác sẽ rảnh trong giai đoạn này
- Đầu vào từ giai đoạn trước (ALU):
 - Kết quả tính toán được dùng làm địa chỉ vùng nhớ (nếu có thể ứng dụng)
- Đầu ra cho giai đoạn tiếp theo (Result Write):
 - Dữ liệu được đọc từ vùng nhớ đối với lệnh Load
 - Kết quả được lưu trữ lại đối với lệnh Store



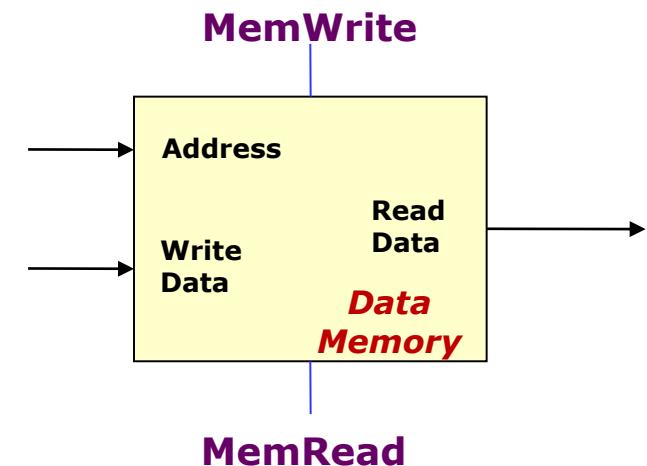
Giai đoạn truy xuất vùng nhớ (Memory stage)





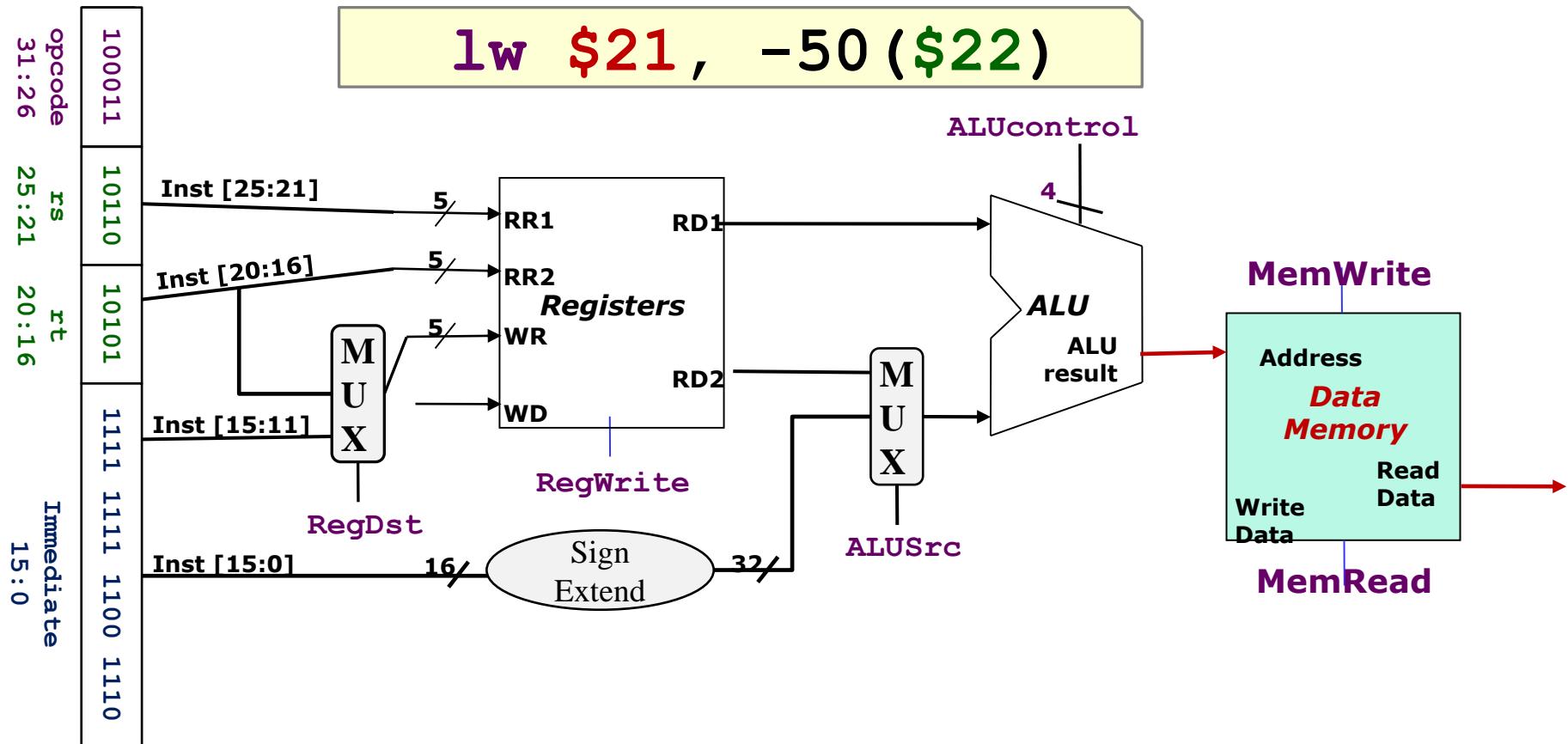
Khối Data Memory

- Vùng nhớ này lưu trữ dữ liệu cần thiết của chương trình
- Inputs:
 - Address: Địa chỉ vùng nhớ
 - Write Data: Dữ liệu sẽ được ghi vào vùng nhớ đối với lệnh Store
- Tín hiệu điều khiển:
 - Tín hiệu đọc (MemRead) và ghi (MemWrite); chỉ một tín hiệu được bật lên tại bất kỳ một thời điểm nào
- Output:
 - Dữ liệu được đọc từ vùng nhớ đối với lệnh Load



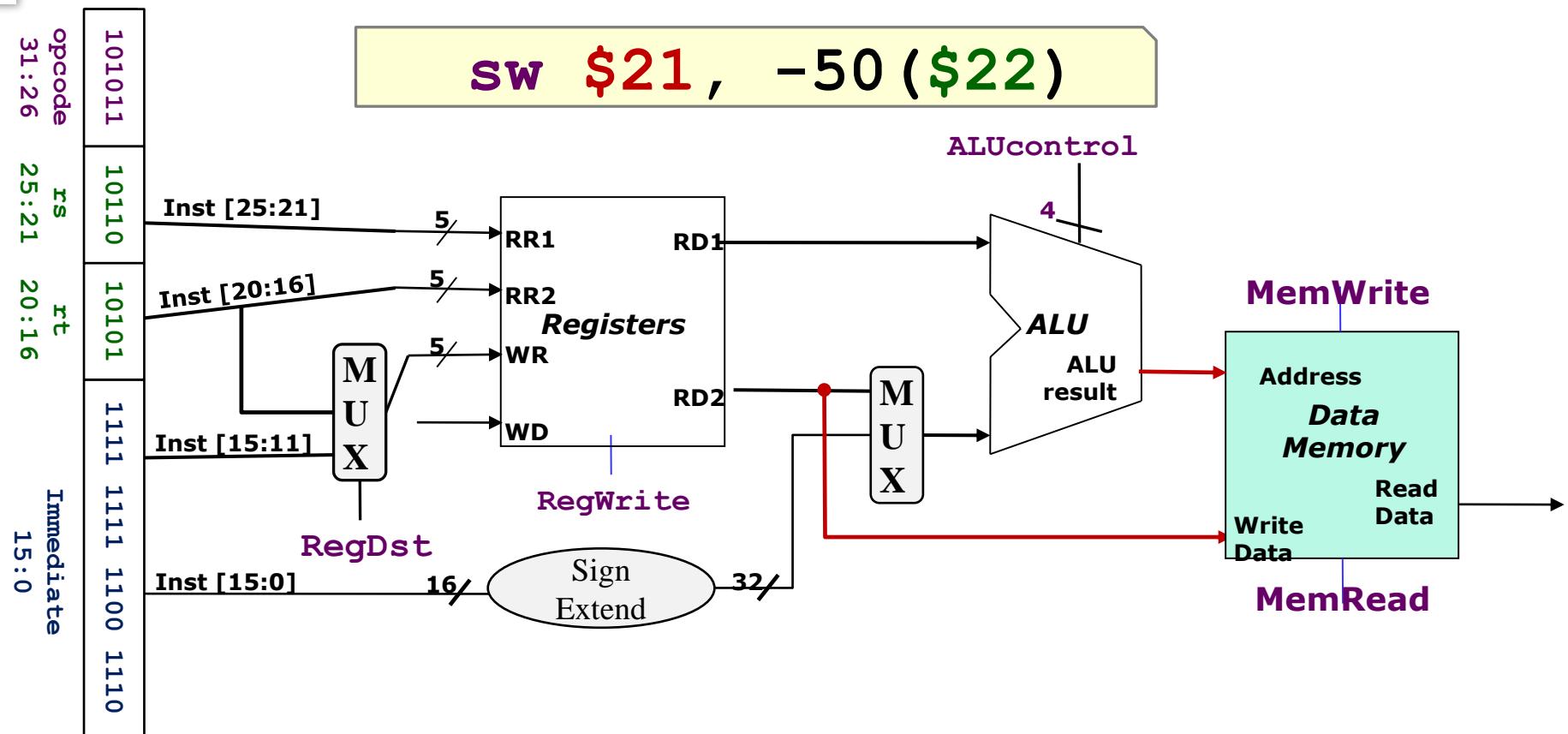


Giai đoạn Memory: lệnh Load



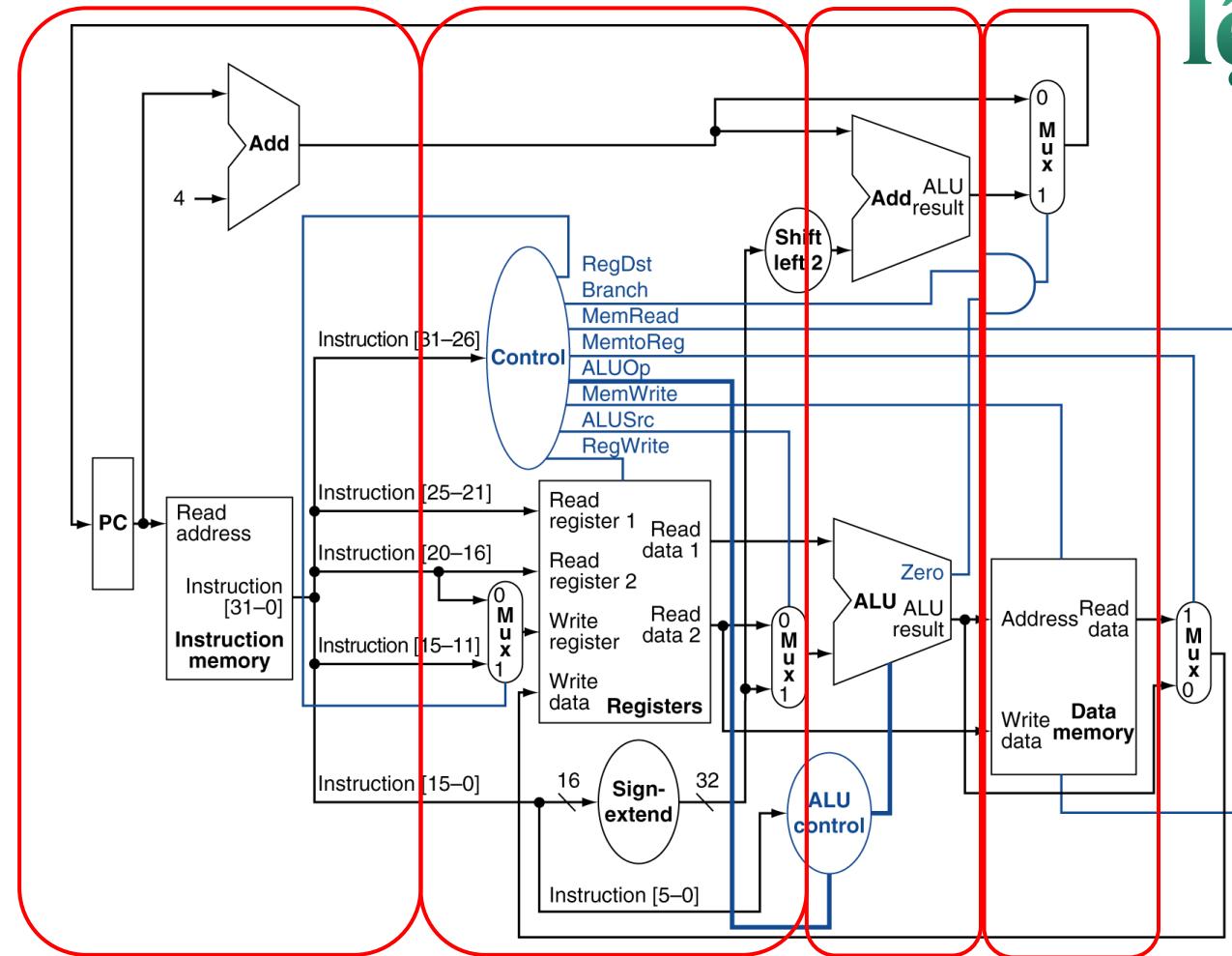


Giai đoạn Memory: lệnh Store





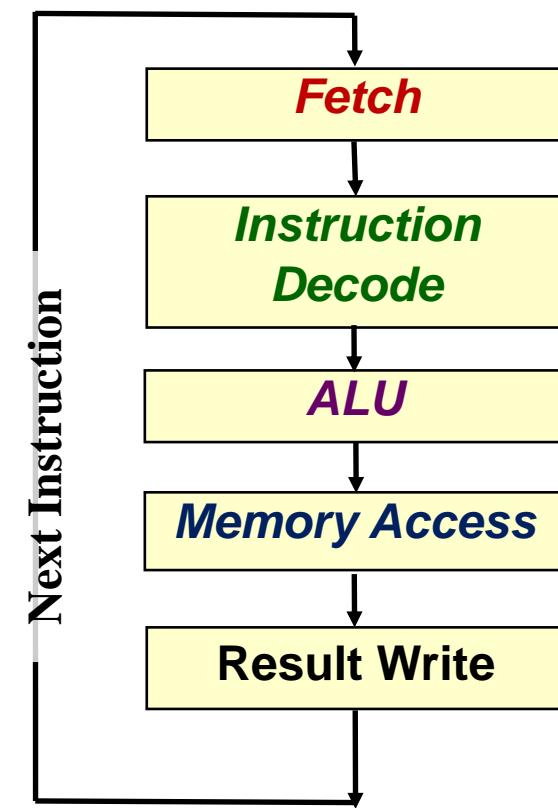
Datapath của một bộ xử lý với 8 lệnh MIPS





Quy trình thực thi lệnh của MIPS

- Instruction Fetch (Nạp lệnh)
- Instruction Decode & Operand Fetch
 - (Giải mã và lấy các toán hạng cần thiết, Gọi tắt là “Instruction Decode”)
- ALU (Giai đoạn sử dụng ALU hay giai đoạn thực thi)
- Memory Access (Giai đoạn truy xuất vùng nhớ)
- **Result Write (Giai đoạn ghi lại kết quả/lưu trữ)**





Giai đoạn truy xuất vùng nhớ (Memory stage)

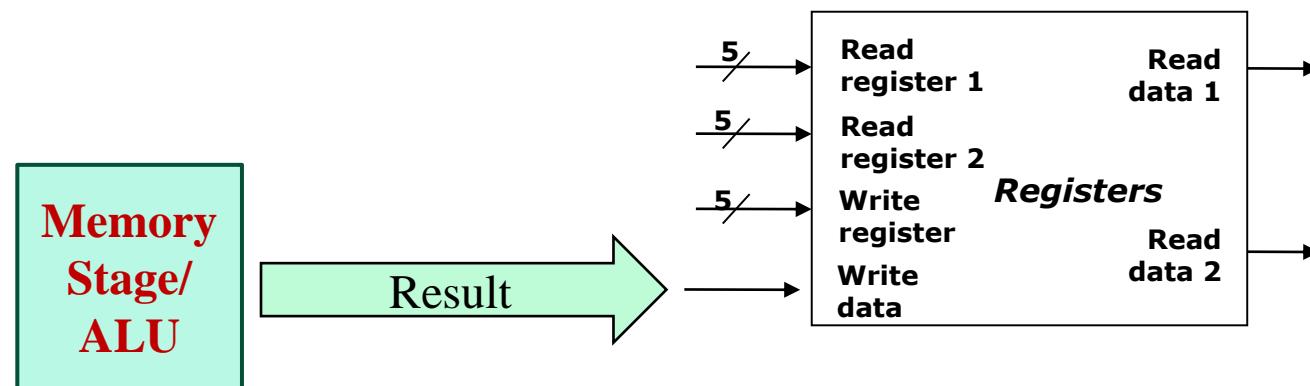
- Công đoạn Result Write:
 - Những lệnh ghi kết quả của các phép toán vào thanh ghi:
 - Ví dụ: số học, logic, shifts, load, set-less-than
 - Cần chỉ số thanh ghi đích và kết quả tính toán
 - Những lệnh không ghi kết quả như: store, branch, jump:
 - Không có ghi kết quả
 - ⇒ Những lệnh này sẽ rảnh trong giai đoạn này
- Đầu vào từ giai đoạn trước (Memory):
 - Kết quả tính toán: từ Memory hoặc từ ALU



Khối ALU (Arithmetic Logical Unit)

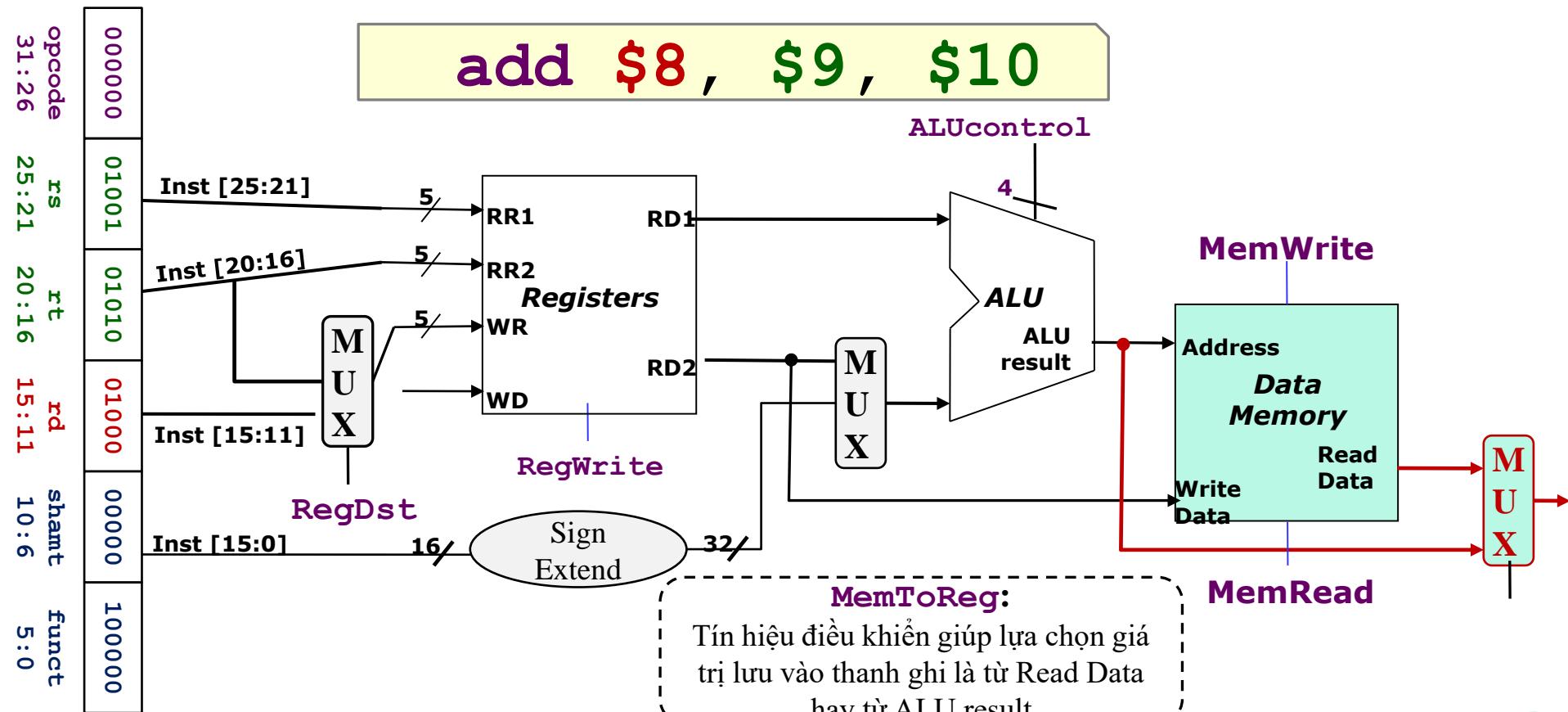
➤ Công đoạn Result Write:

- Đưa kết quả từ ALU hoặc Memory vào thanh ghi (ngõ Write data của khối Registers/Register file)
- Chỉ số của thanh ghi được ghi vào (ngõ vào Write Register) được sinh ra trong giai đoạn Decode Stage



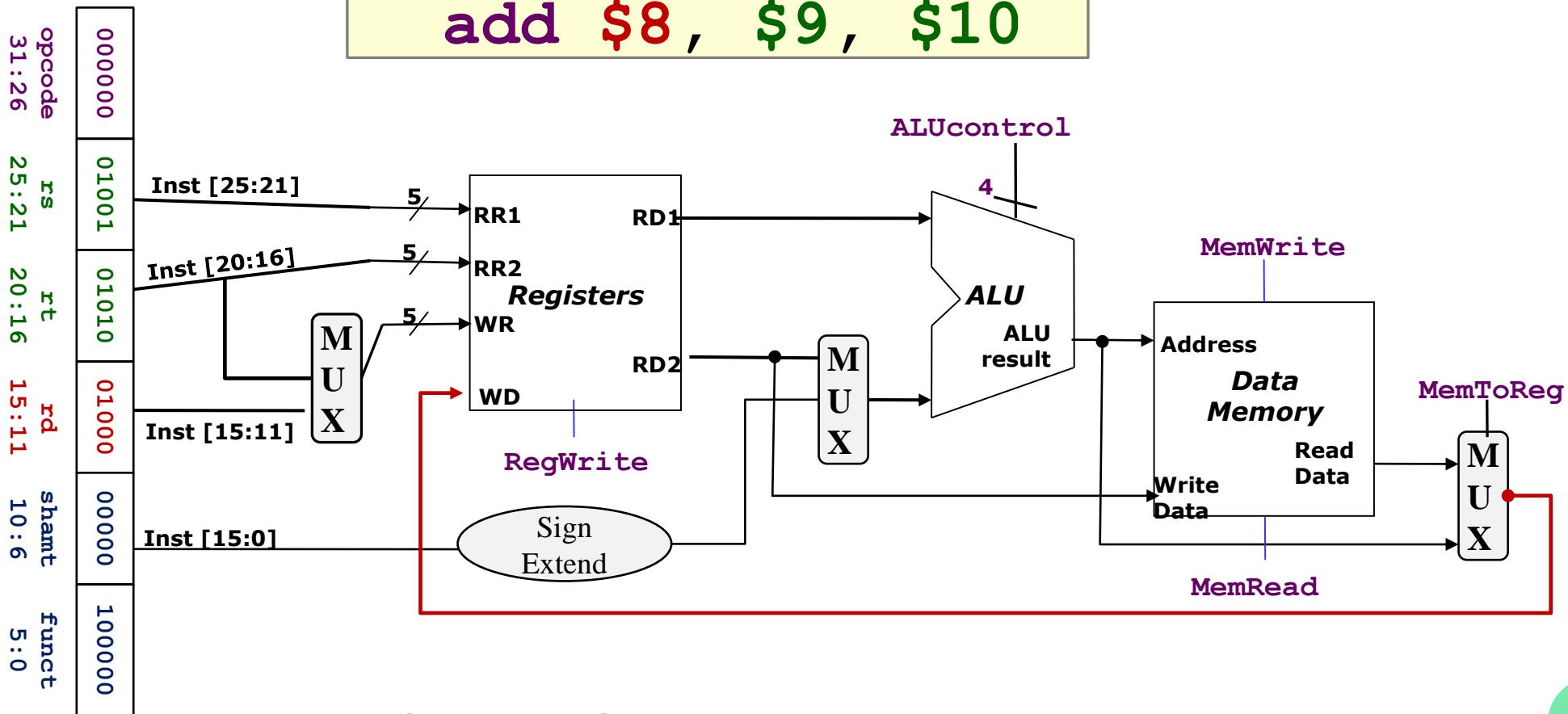


Giai đoạn lưu trữ kết quả (Result Write)



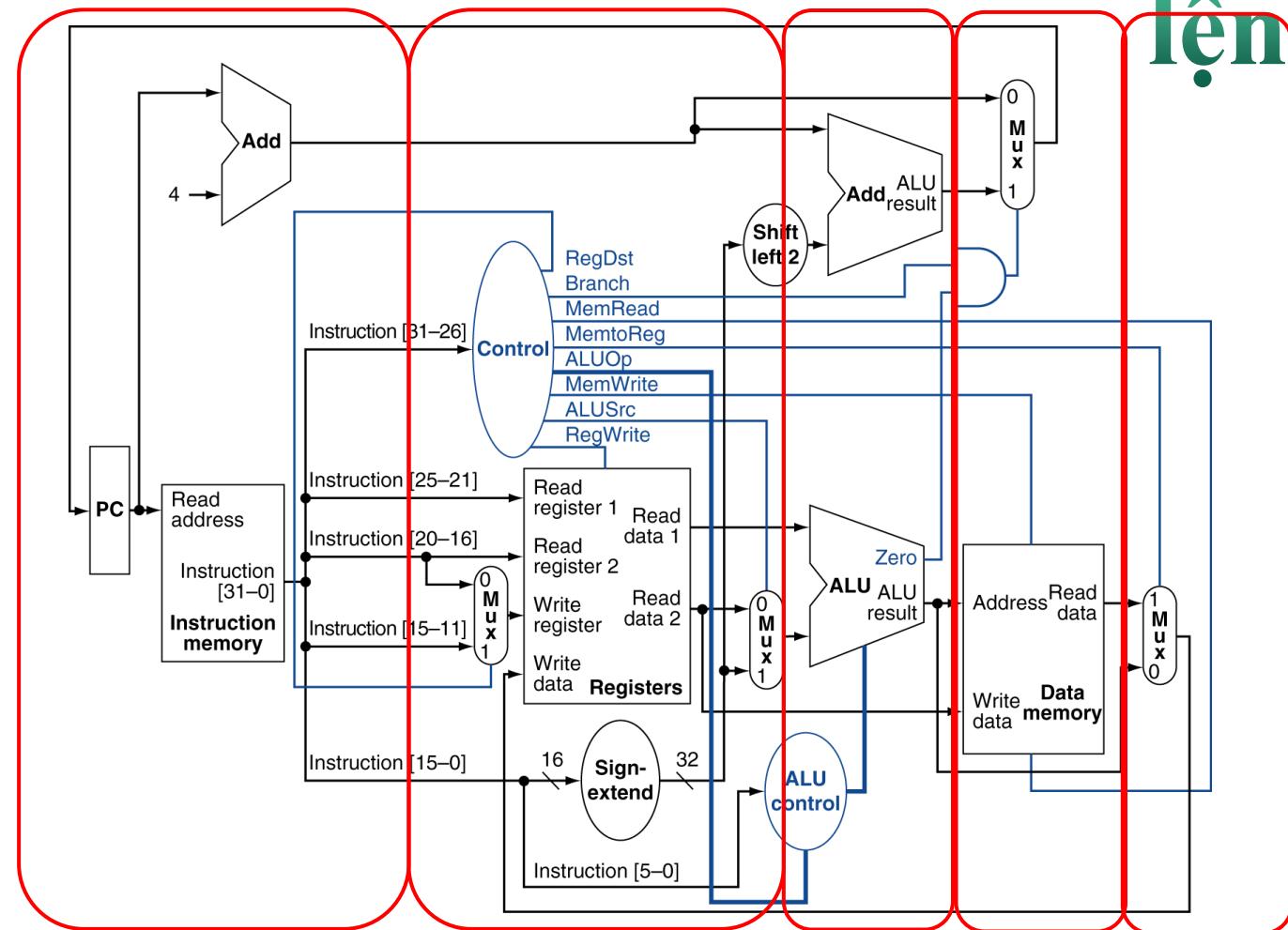


Giai đoạn lưu trữ kết quả (Result Write)



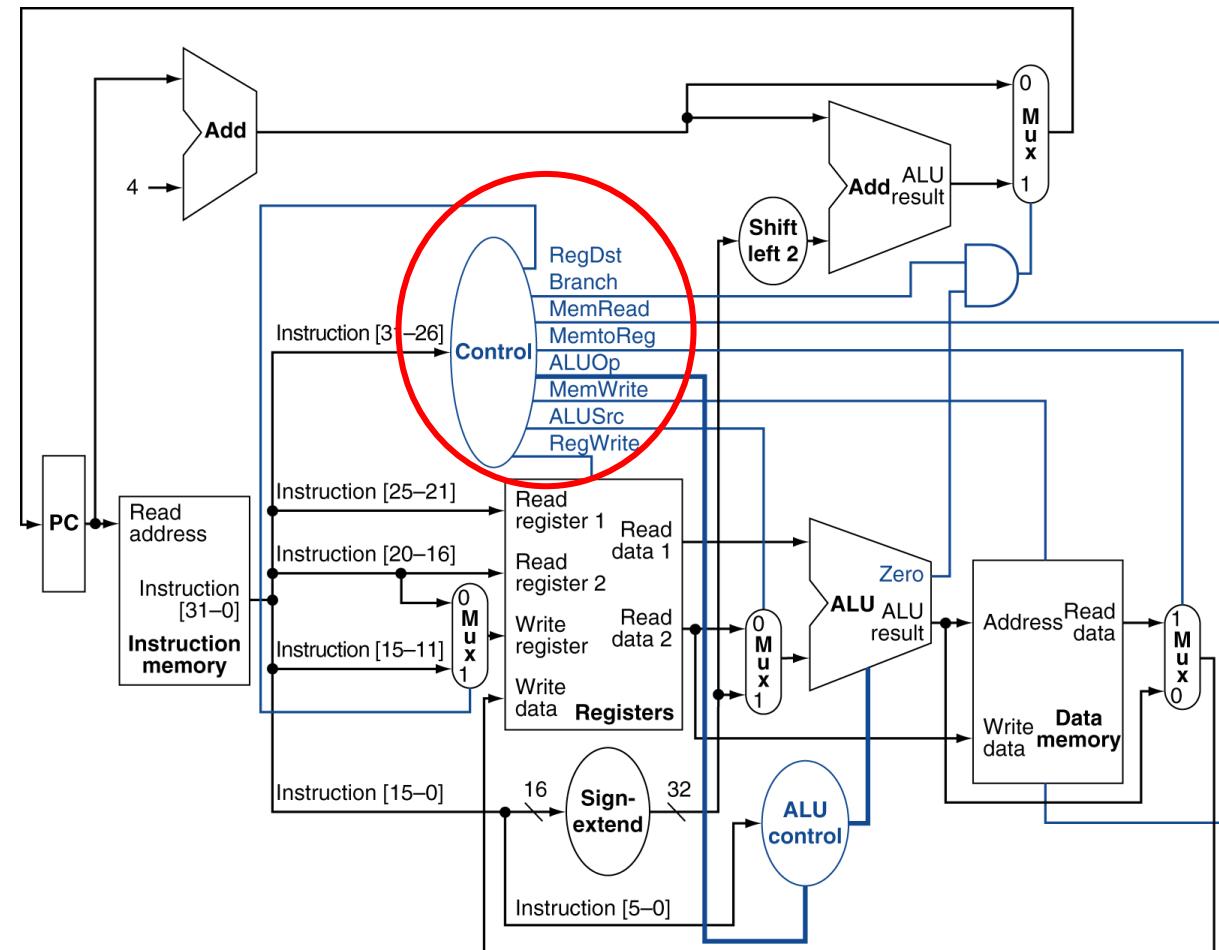


Datapath của một bộ xử lý với 8 lệnh MIPS





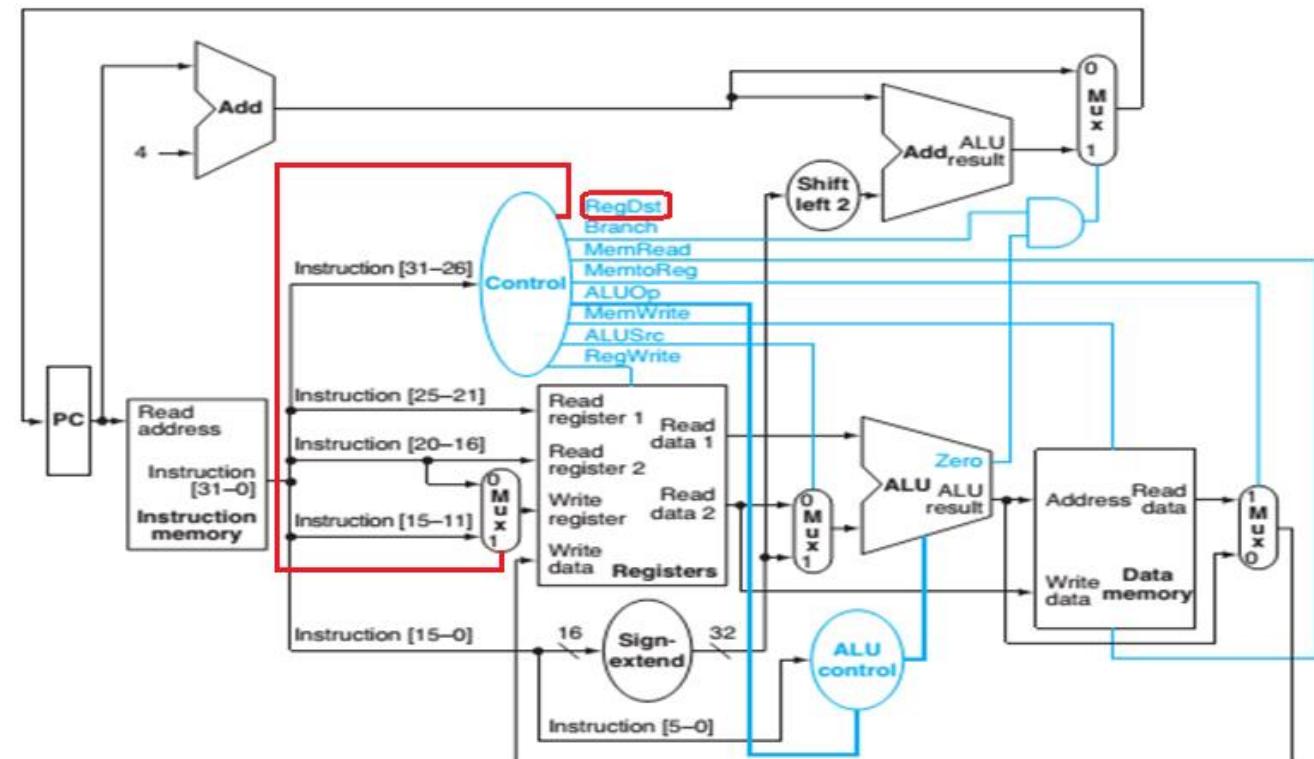
Datapath của một bộ xử lý với 8 lệnh MIPS





Hiện thực datapath - Control (1)

- **RegDst** dùng để chọn thanh ghi đích cho thao tác ghi:
 - RegDst = 0: Các bit từ 16:20 được chọn (rt - dành cho lệnh loadword)
 - RegDst = 1: Các bit từ 11:15 được chọn (rd - dành cho các lệnh còn lại như add, sub, and, or, slt)
 - Lệnh sw và beq không sử dụng giá trị này

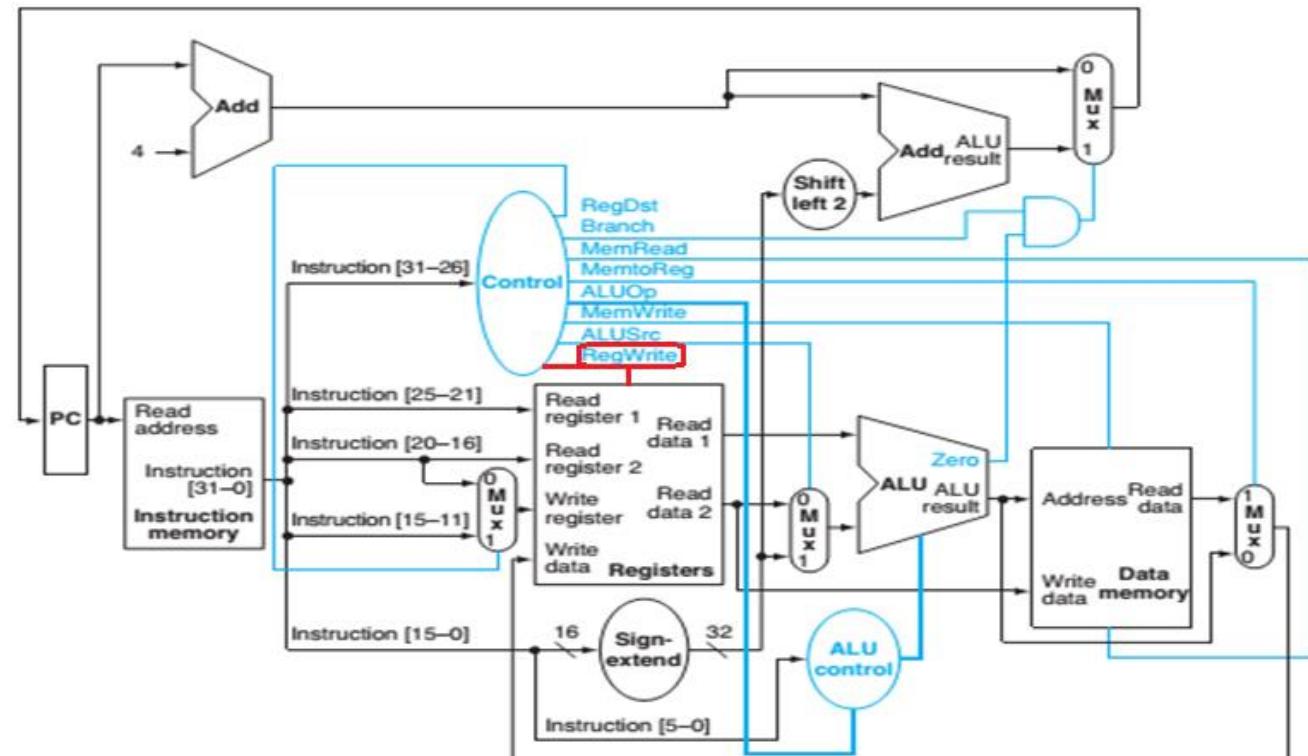


Tín hiệu RegDst trong datapath



Hiện thực datapath - Control (2)

- **RegWrite** dùng để cho phép thao tác ghi vào khôi thanh ghi:
 - RegWrite = 0: Khối thanh ghi chỉ có chức năng đọc (dành cho các lệnh sw và beq)
 - RegWrite = 1: Khối thanh ghi có thể thực hiện chức năng đọc và ghi. Thanh ghi được ghi là thanh ghi có chỉ số được đưa vào từ ngõ “Write register” và dữ liệu dùng ghi vào thanh ghi này được lấy từ ngõ “Write data” (dành cho các lệnh còn lại)

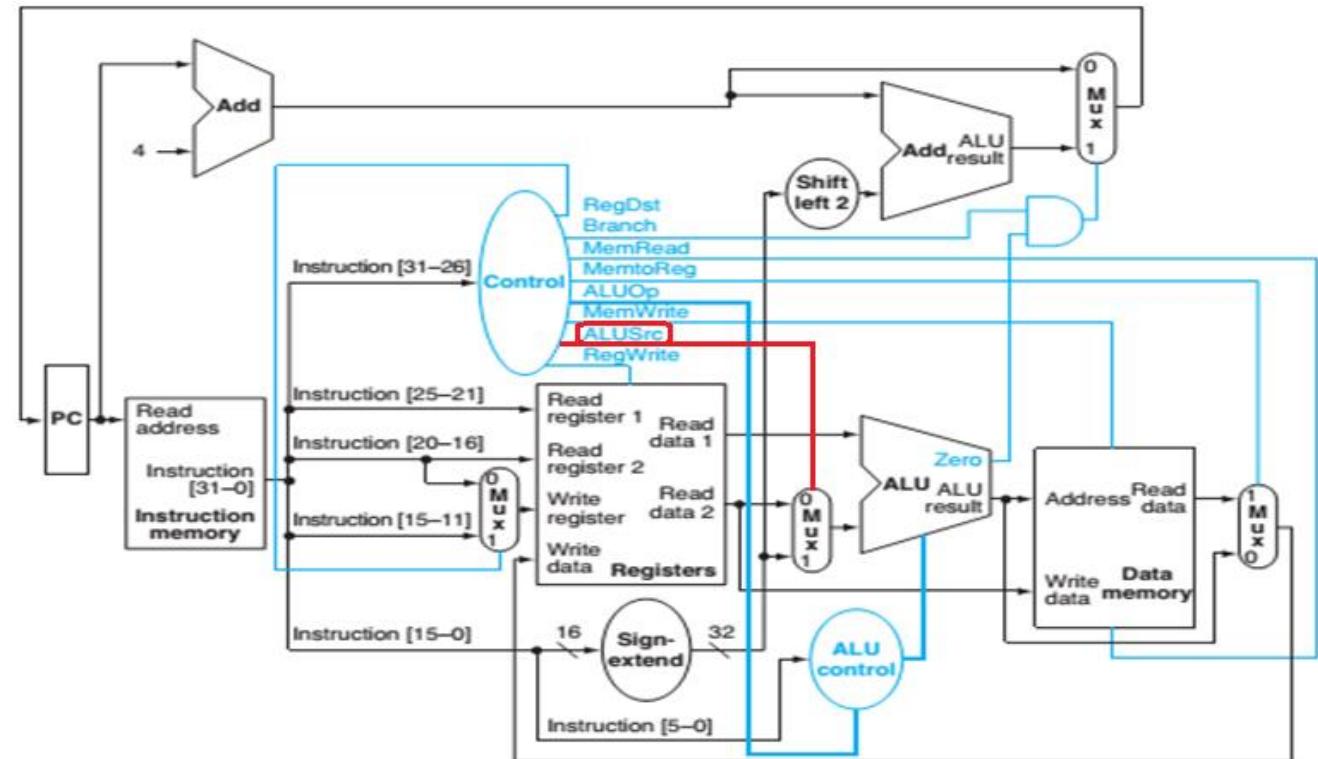


Tín hiệu RegWrite trong datapath



- **ALUSrc** dùng để chọn đầu vào thứ 2 cho khối ALU:
 - **ALUSrc = 0:** Đầu vào thứ 2 cho ALU đến từ “Read data 2” của khối “Registers (dành cho các lệnh add, sub, and, or, slt, beq) ”
 - **ALUSrc = 1:** Đầu vào thứ 2 cho ALU đến từ output của khối “Sign-extend” (dành cho các lệnh còn lại như lw và sw)

Hiện thực datapath - Control (3)

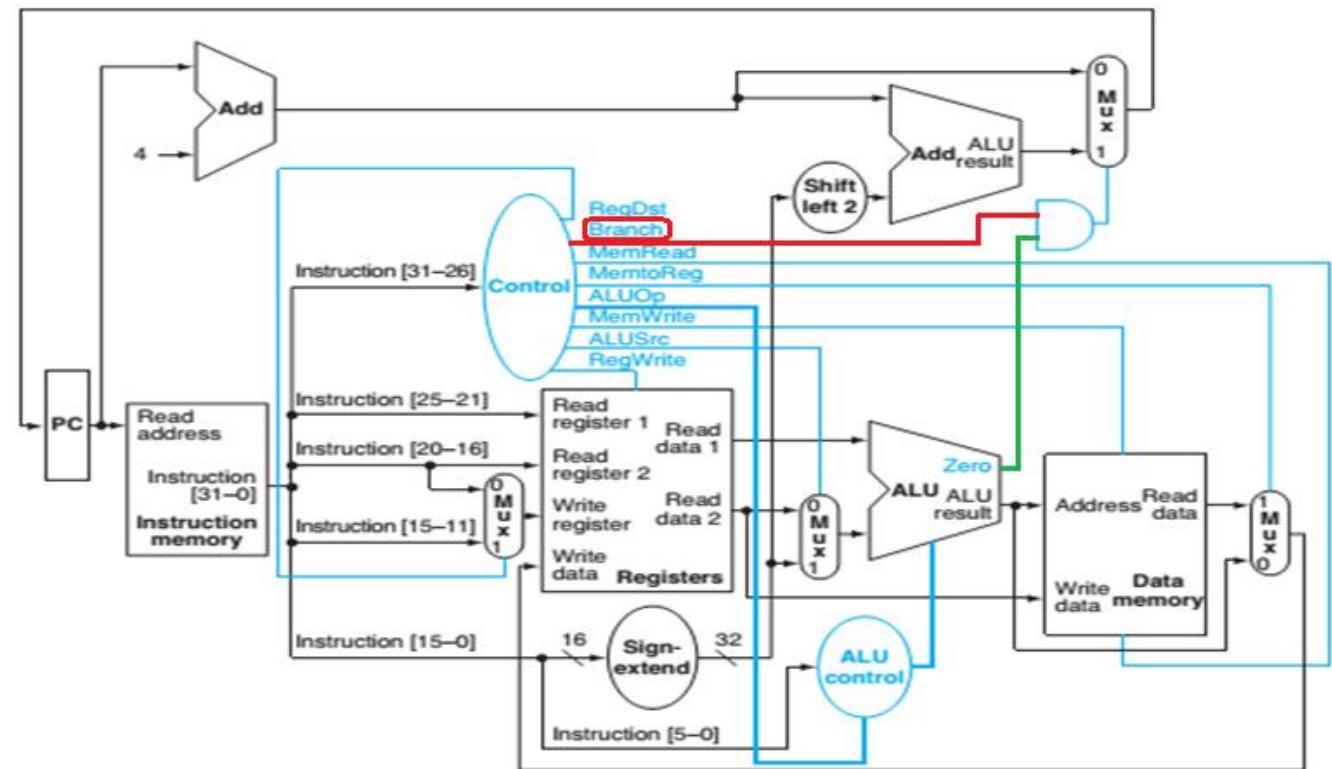


Tín hiệu ALUSrc trong datapath



Hiện thực datapath - Control (4)

- Branch dùng để kết hợp (AND) với giá trị Zero của ALU chọn giá trị sẽ được nạp vào thanh ghi PC:
 - Branch = 0: Giá trị của PC đến từ khối Add (4) $PC = PC + 4$ (dành cho các lệnh add, sub, and, or, slt, lw, sw)
 - Branch = 1: Tùy thuộc vào giá trị của Zero của khối ALU (dành cho lệnh beq).
 - Nếu Zero = 0: Giá trị của PC đến từ khối Add (4) $PC = PC + 4$
 - Nếu Zero = 1: Giá trị của PC đến từ khối Add (Shift left 2) $PC = PC + 4 + Label$

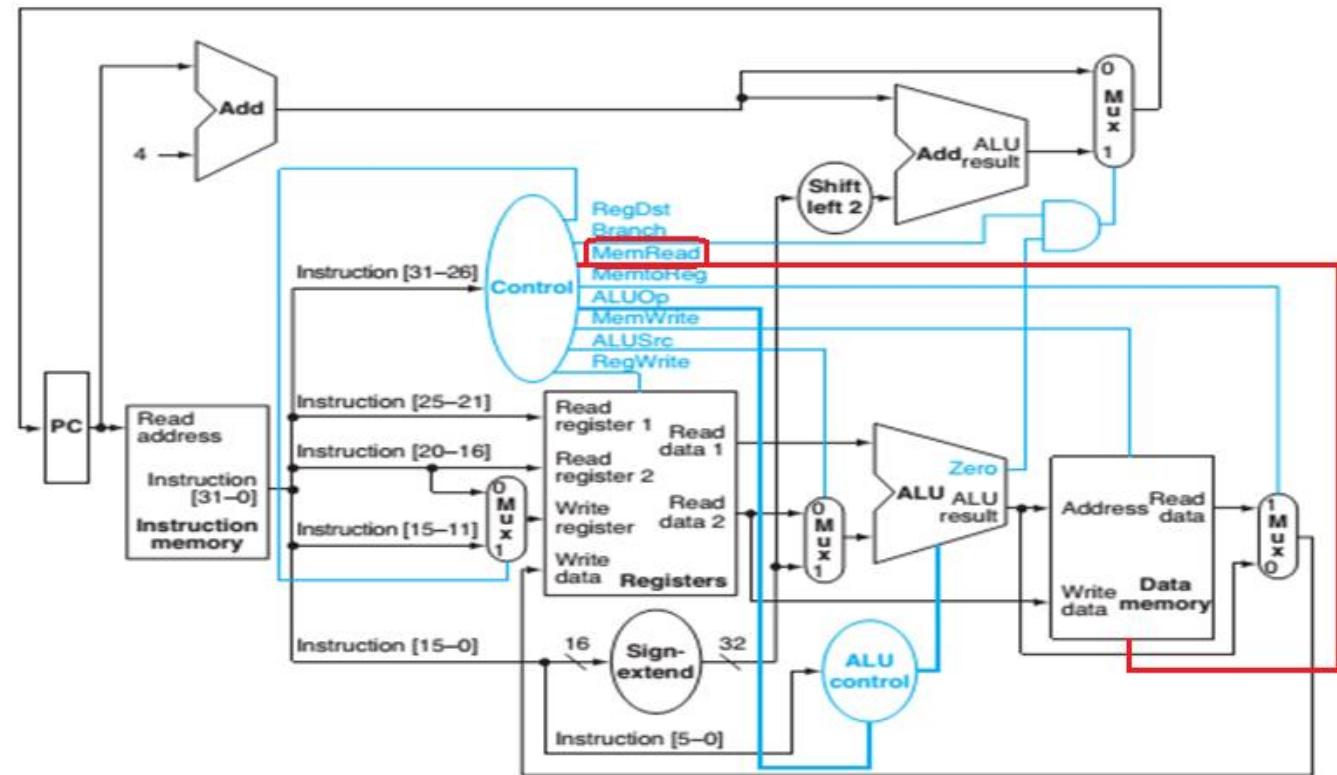


Tín hiệu Branch trong datapath



Hiện thực datapath - Control (5)

- **MemRead** dùng để cho phép thao tác đọc từ khói Data memory:
 - MemRead = 1: Khối “Data memory” thực hiện chức năng đọc dữ liệu. Địa chỉ dữ liệu cần đọc được đưa vào từ ngõ “Address” và nội dung đọc được xuất ra ngõ “Read data” (dành cho lệnh lw)
 - MemRead = 0: Khối “Data memory” không thực hiện chức năng đọc dữ liệu (dành cho các lệnh còn lại)

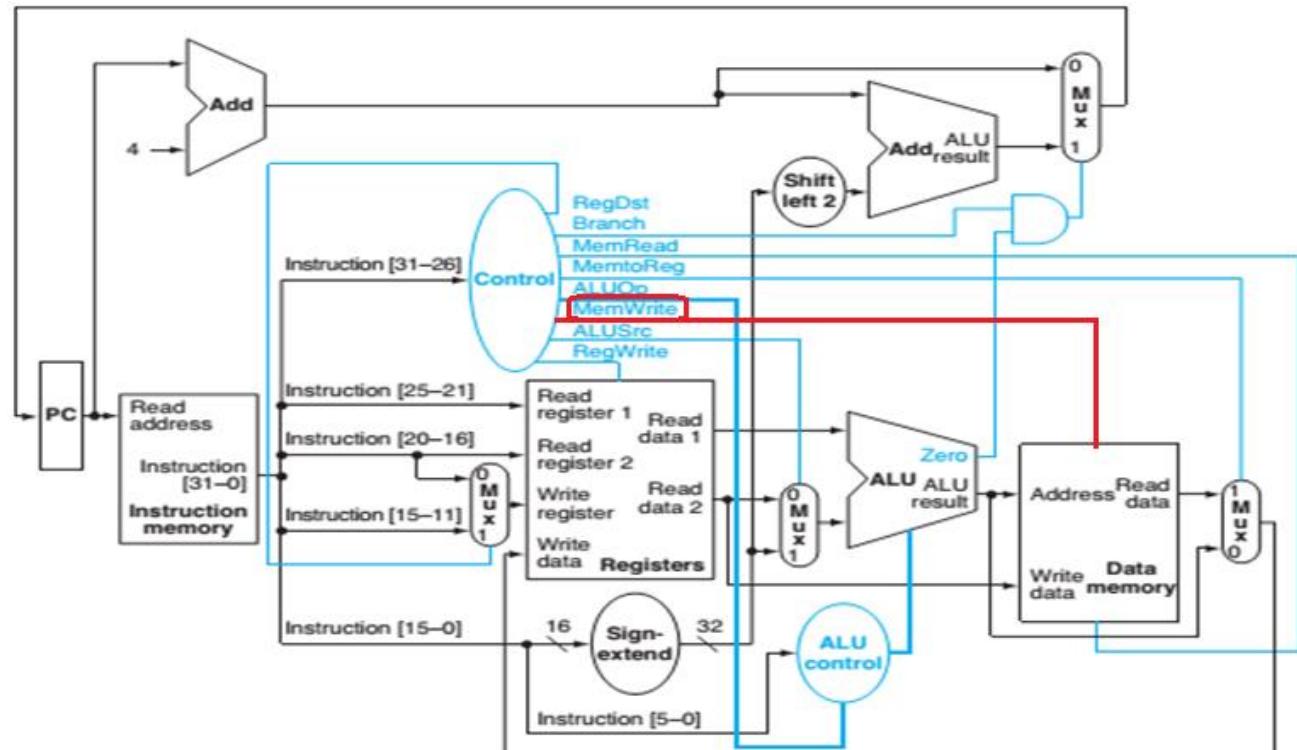


Tín hiệu MemRead trong datapath



Hiện thực datapath - Control (6)

- **MemWrite** dùng để cho phép thao tác ghi vào khối Data memory:
 - MemWrite = 1: Khối “Data memory” thực hiện chức năng ghi dữ liệu. Địa chỉ dữ liệu cần ghi được đưa vào từ ngõ “Address” và nội dung ghi vào lấy từ ngõ “Write data” (dành cho lệnh sw)
 - MemWrite = 0: Khối “Data memory” không thực hiện chức năng ghi dữ liệu (dành cho các lệnh còn lại)

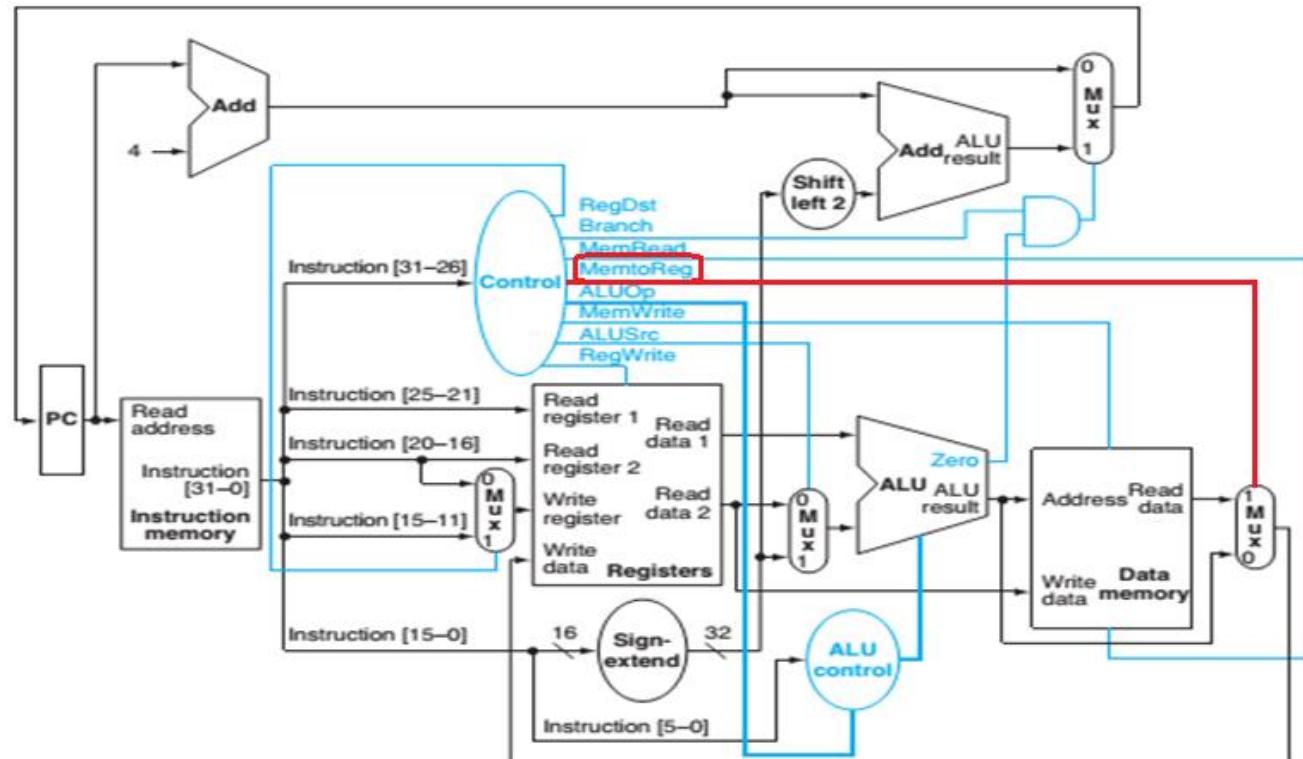


Tín hiệu MemWrite trong datapath



Hiện thực datapath - Control (7)

- **MemtoReg** dùng để chọn giá trị được đưa vào ngõ Write data của khối Registers:
 - MemtoReg = 0: Giá trị đưa vào ngõ “Write data” đến từ ALU (dành cho các lệnh add, sub, and, or, slt)
 - MemtoReg = 1: Giá trị đưa vào ngõ “Write data” đến từ khối “Data memory” (dành cho lệnh lw)
 - Lệnh sw và beq không sử dụng giá trị này



Tín hiệu MemtoReg trong datapath



Hiện thực datapath - Control (8)

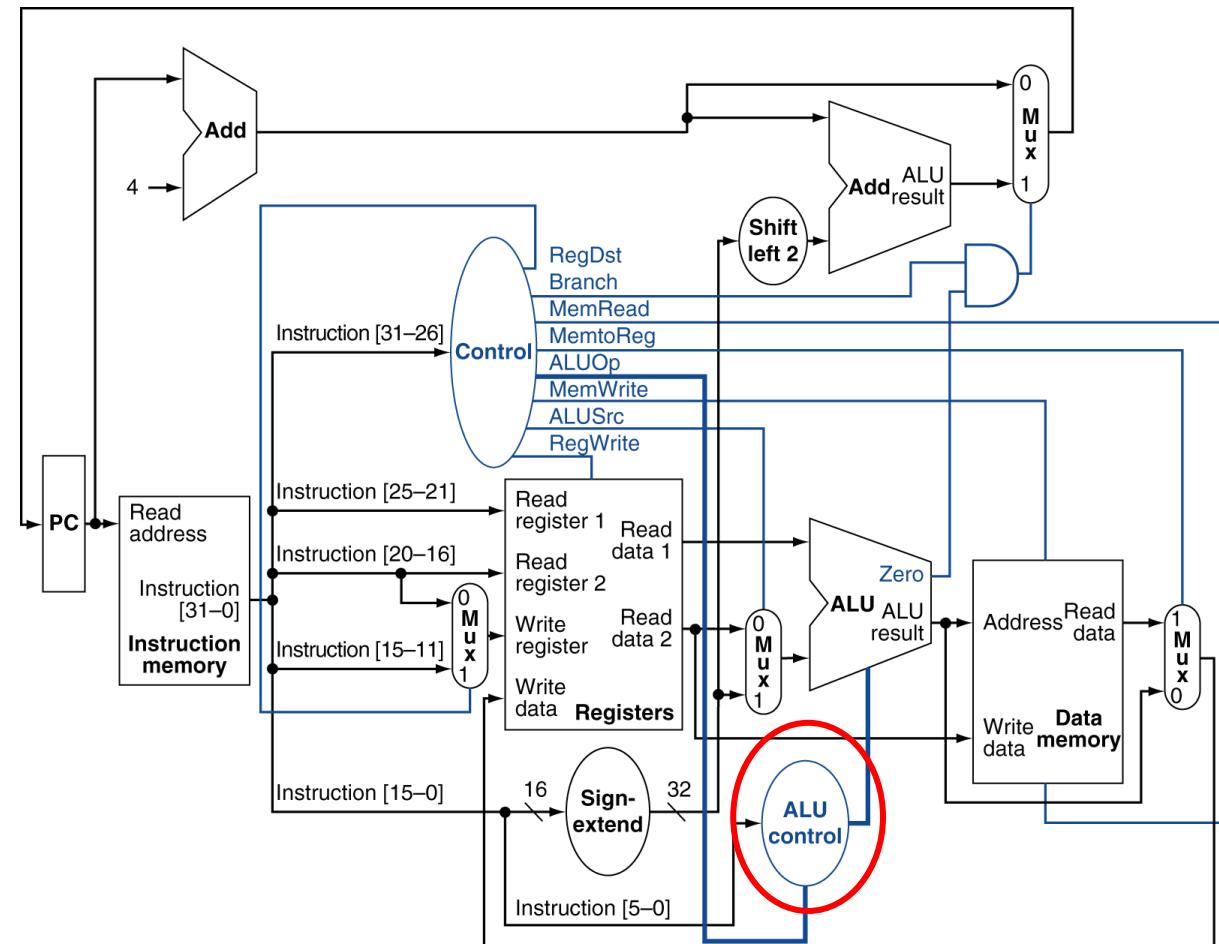
Giá trị các tín hiệu điều khiển tương ứng với mỗi lệnh như sau:

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Khối “Control” trong datapath nhận input là 6 bits từ trường “opcode” của mã máy, dựa vào đó các tín hiệu điều khiển được sinh ra tương ứng như bảng.



Datapath của một bộ xử lý với 8 lệnh MIPS





Hiện thực datapath - ALU Control(2)

- Bộ ALU của MIPS gồm 6 chức năng tính toán dựa trên 4 bits điều khiển đầu vào, tùy thuộc vào từng nhóm lệnh mà ALU sẽ thực hiện 1 trong 5 chức năng đầu (NOR sẽ được dùng cho các phần khác)
 - Với các lệnh load word và store word, ALU sử dụng chức năng ‘add’ để tính toán địa chỉ của bộ nhớ
 - Với các lệnh thuộc nhóm logic và số học, ALU thực hiện 1 trong 5 chức năng (and, or, subtract, add, và set on less than), tùy thuộc vào giá trị của trường funct (6 bits) trong mã máy lệnh.
 - Với lệnh nhảy nếu bằng, ALU thực hiện chức năng ‘subtract’ để xem điều kiện bằng có đúng không.

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



Hiện thực datapath - ALU Control (3)

- Như vậy, để sinh ra 4 bits điều khiển ALU, một trong số các cách hiện thực có thể là sử dụng thêm một khối điều khiển “ALU Control”
- “ALU Control” nhận input là 6 bits từ trường funct của mã máy, đồng thời dựa vào 2 bits “ALUOp” được sinh ra từ khối “Control” để sinh ra output là 4 bits điều khiển ALU, theo quy tắc như bảng sau:

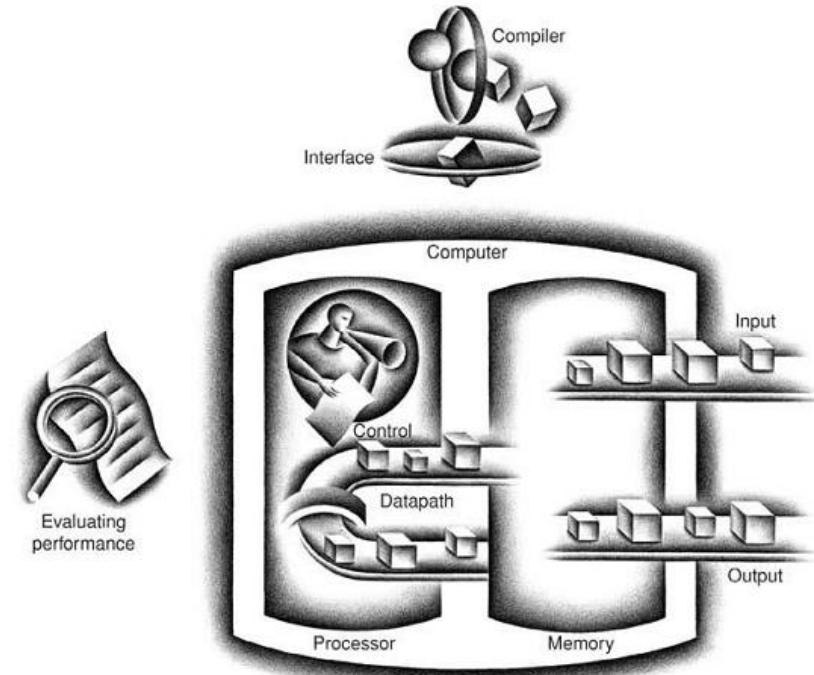
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

Một gợi ý để sinh ra 4 bits điều khiển ALU dựa vào trường “opcode” và trường “funct” của mã máy.



NỘI DUNG

- Kiến trúc tổng quan bộ xử lý
- Datapath
- **Thực thi lệnh**
- **Bài tập**

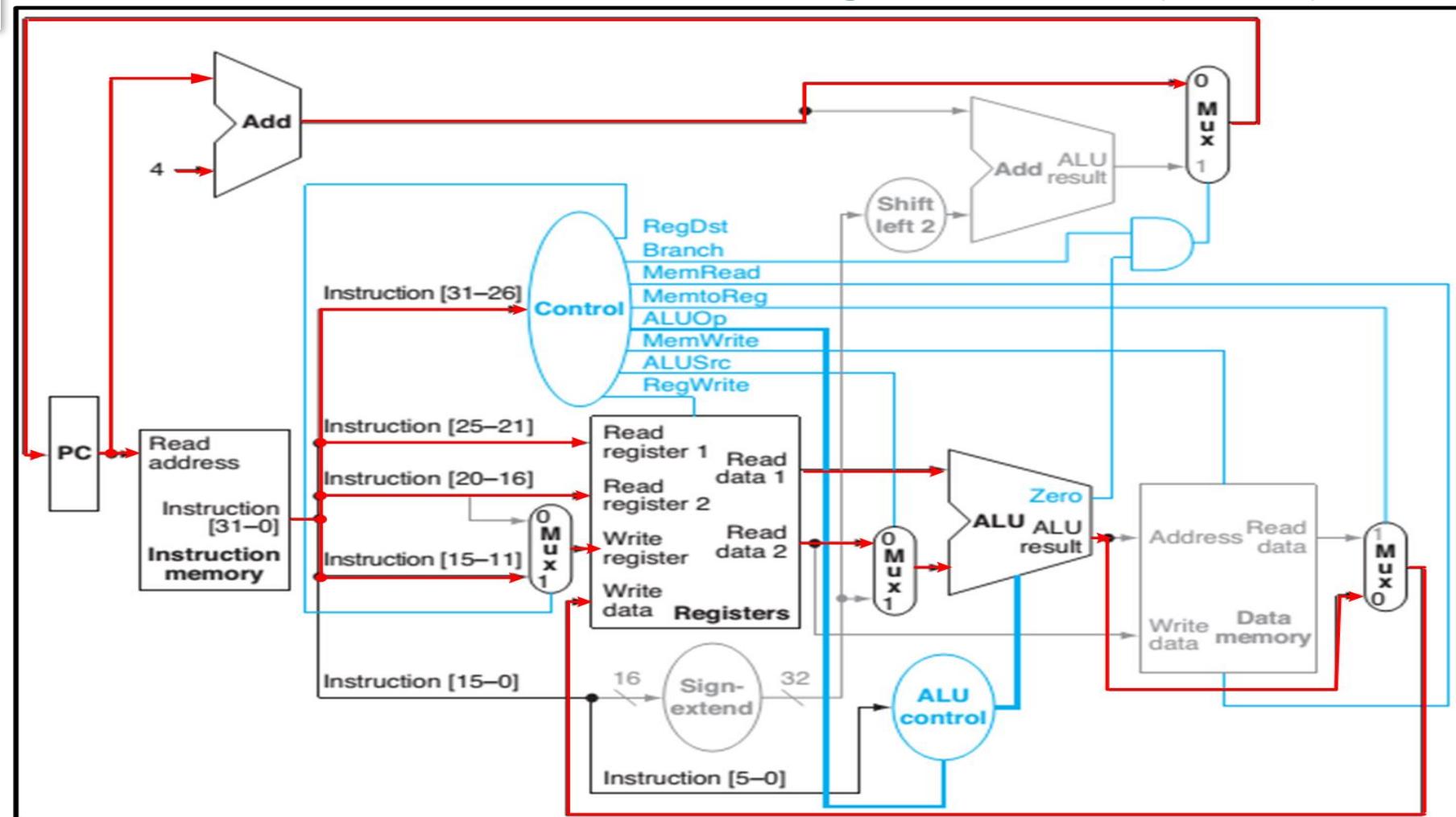


Hình 1: Kiến trúc của 1 máy tính



Các đường hoạt động khi lệnh thuộc nhóm logic và số học thực thi

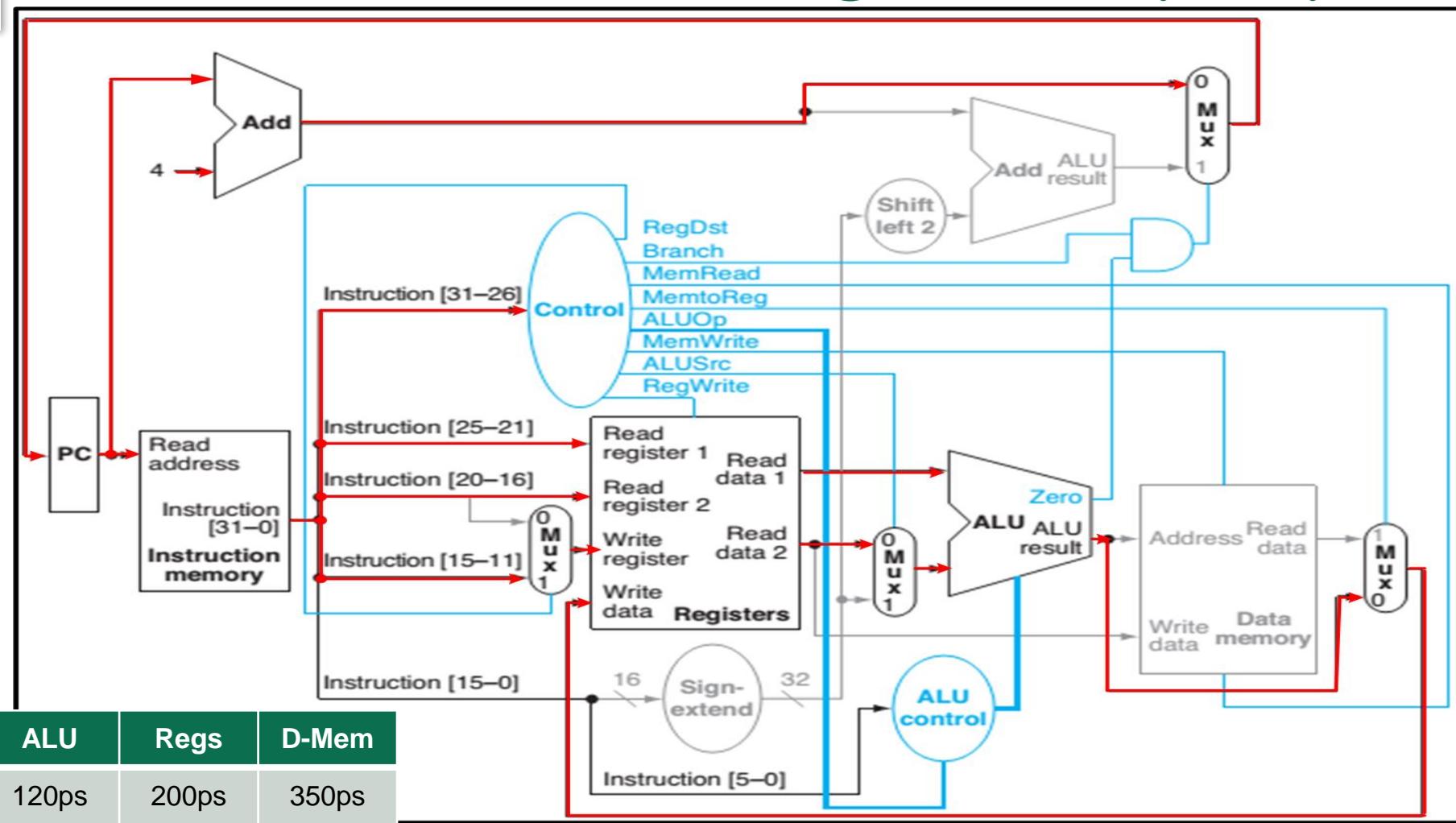
Trình bày
các khối
chức năng
được sử
dụng khi
thực thi
lệnh add?





Các đường hoạt động khi lệnh thuộc nhóm logic và số học thực thi

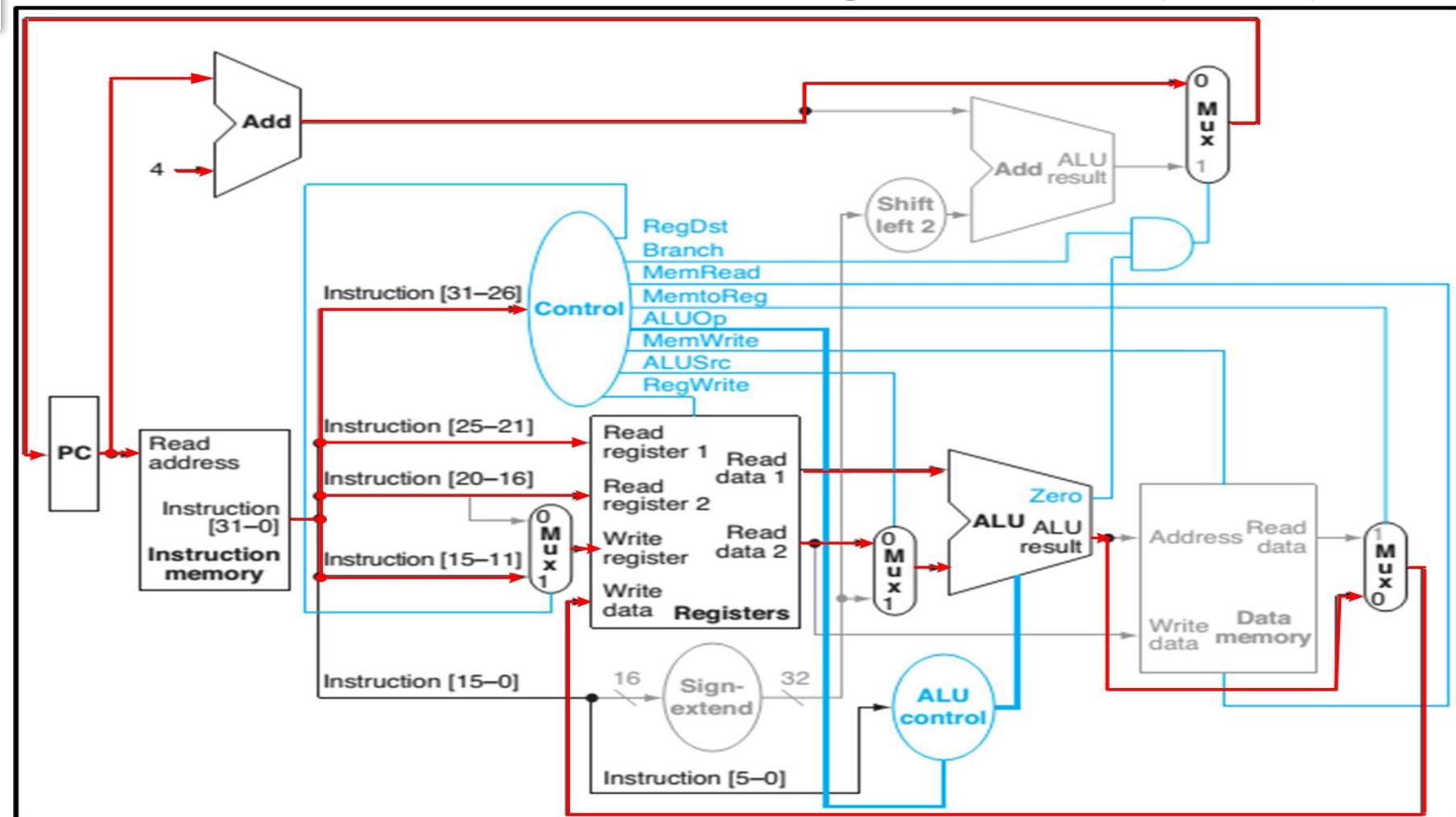
Tính thời gian cần thiết tối thiểu để thực thi lệnh add?





Các đường hoạt động khi lệnh thuộc nhóm logic và số học thực thi

Trình bày
các tín hiệu
điều khiển
khi thực thi
lệnh add?





Bài tập 1

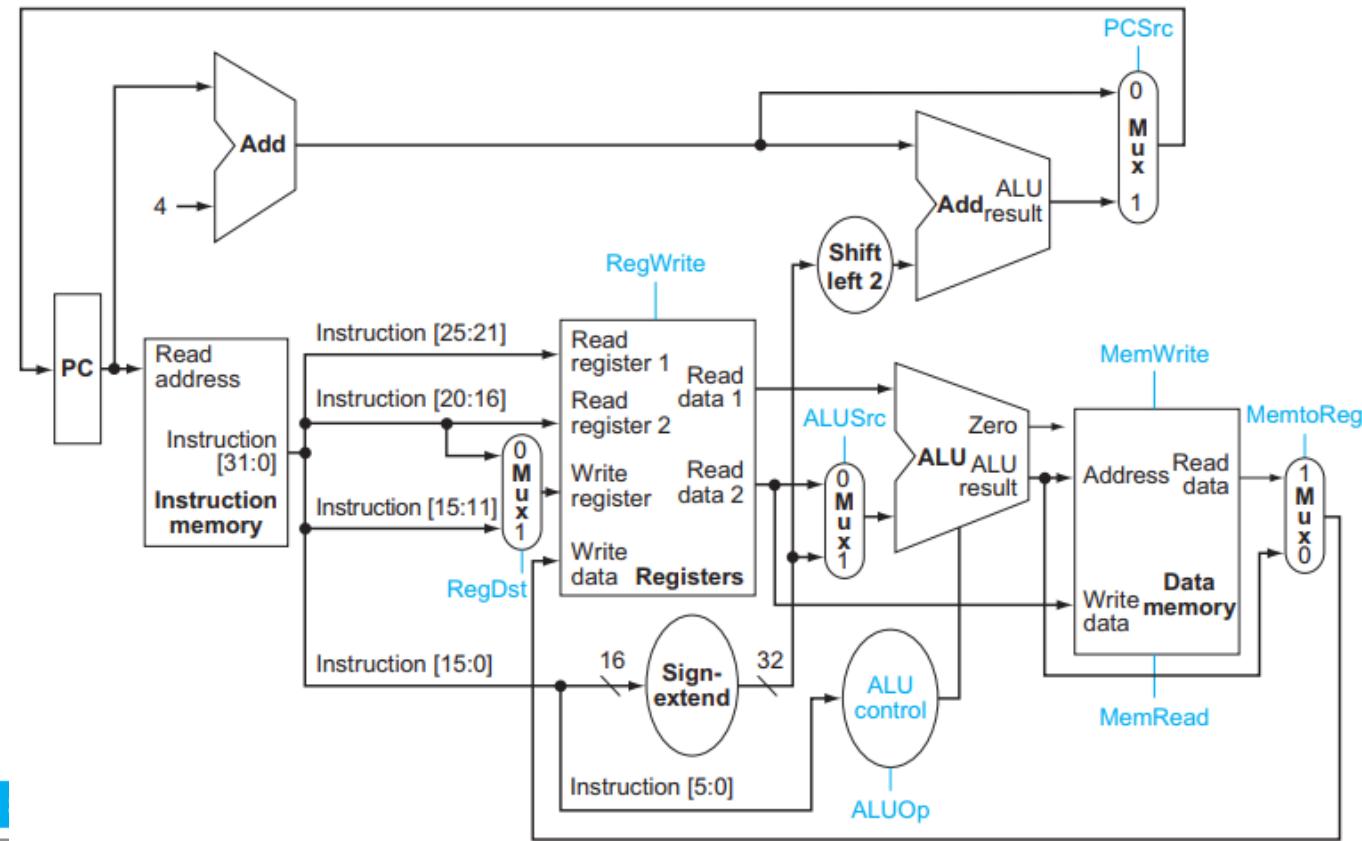
Mô tả đường đi
dữ liệu và tính
thời gian cần thiết
tối thiểu để thực
thi lệnh:

slt \$t0, \$s1, \$s2

s1 = 0x00002022

s2 = 0x00002023

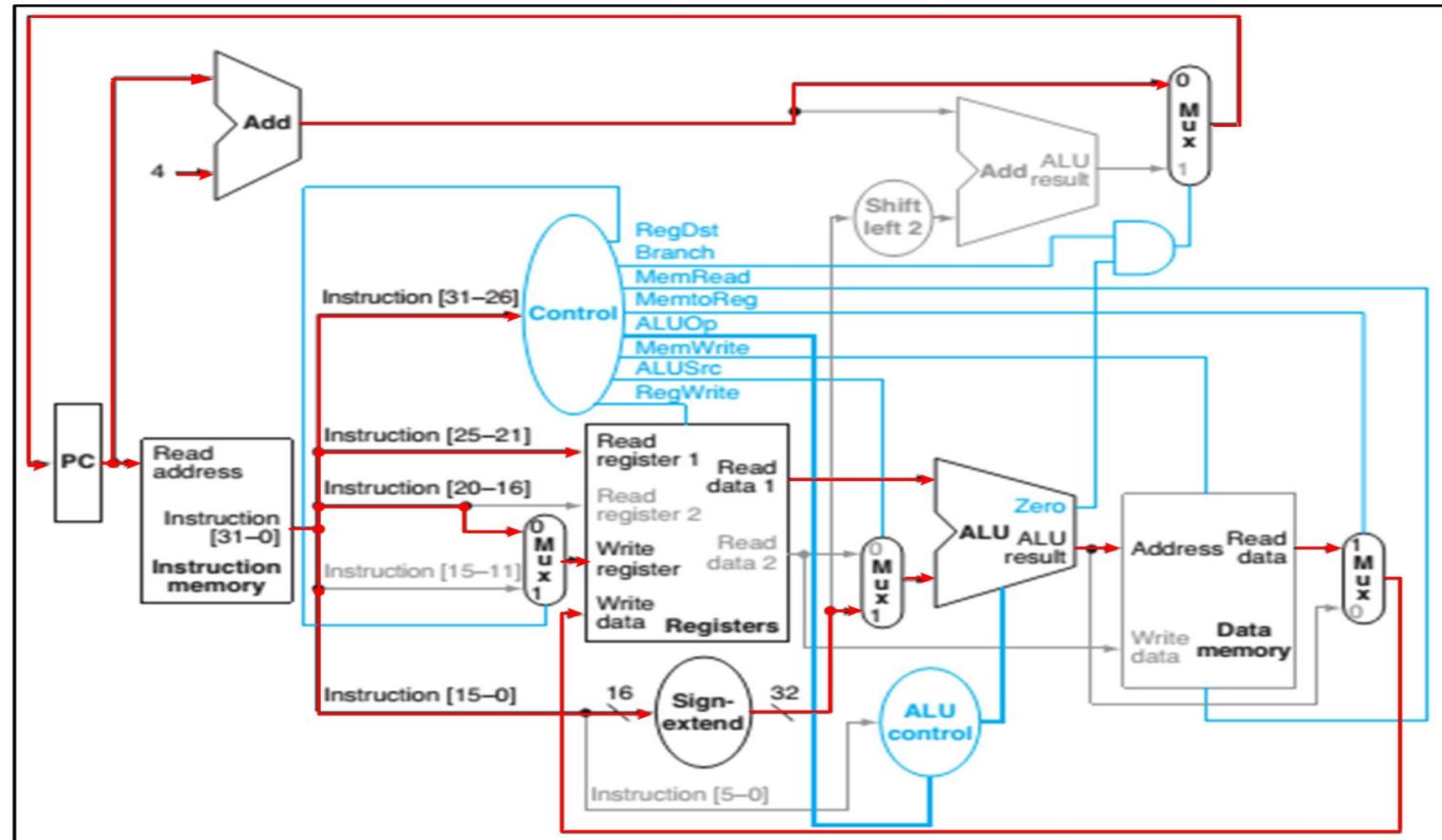
I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps





Trình bày
các khối
chức năng
được sử
dụng khi
thực thi
lệnh lw?

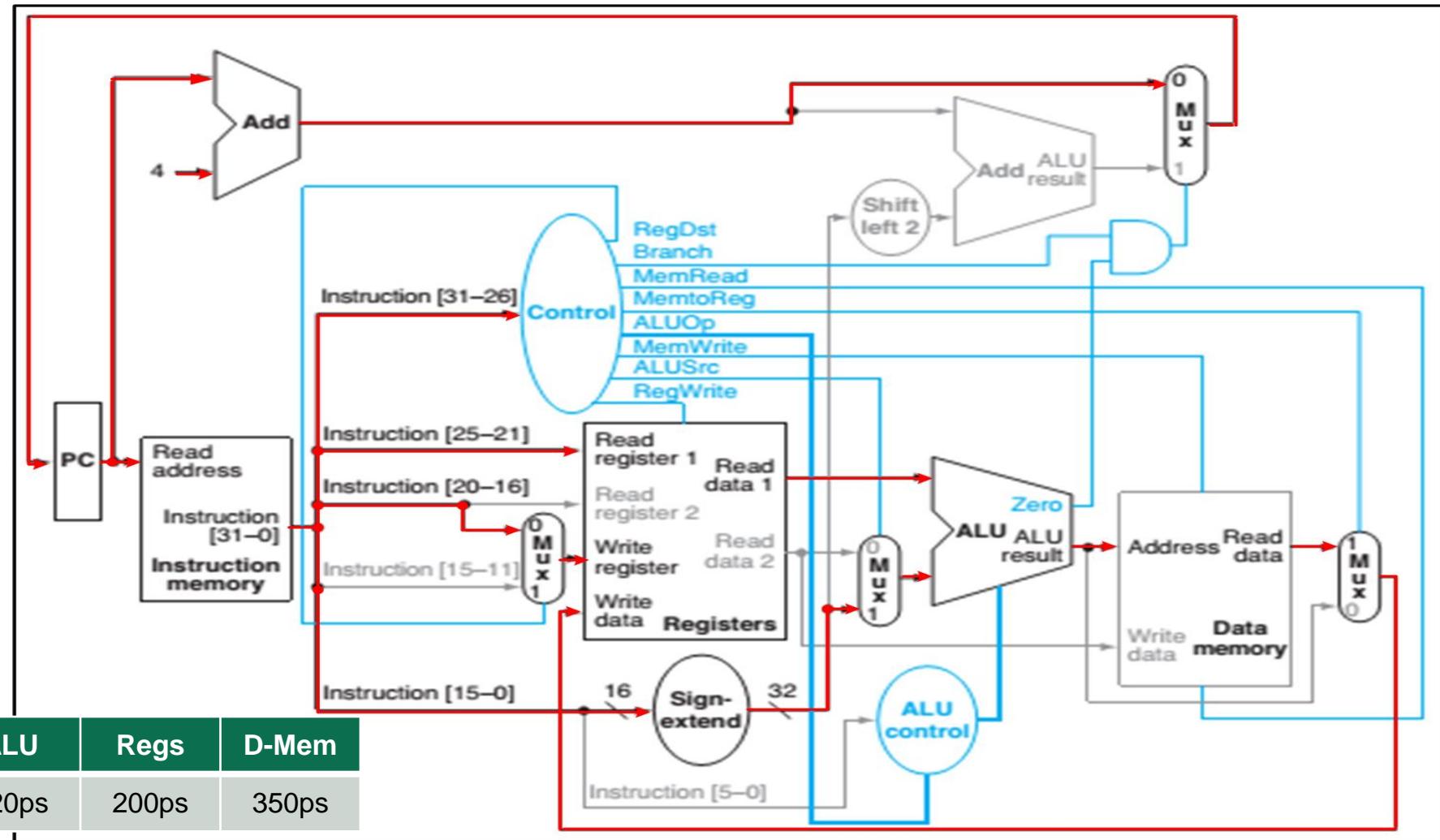
Các đường hoạt động khi lệnh lw thực thi





Tính thời gian cần thiết tối thiểu để thực thi lệnh lw?

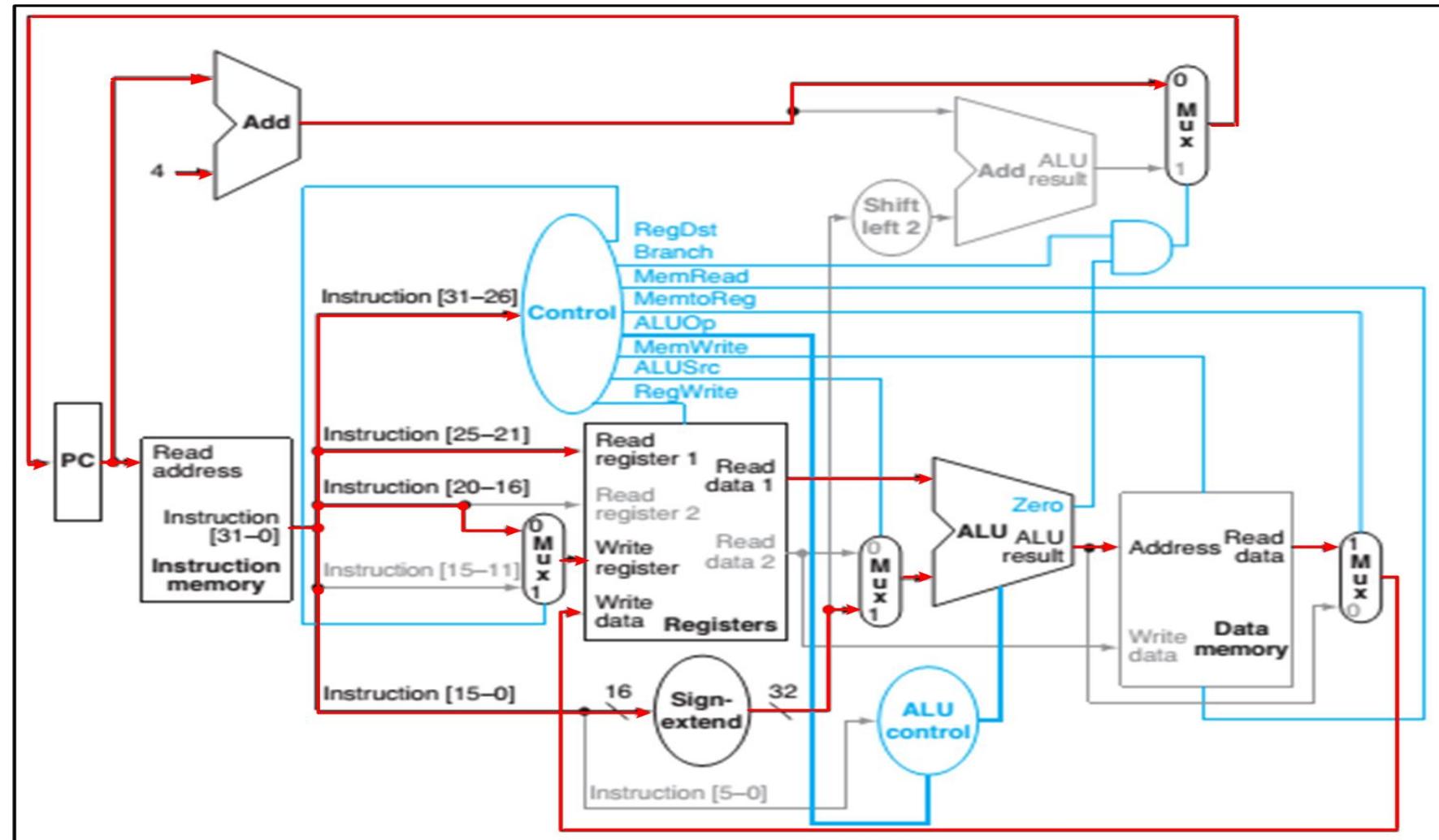
Các đường hoạt động khi lệnh lw thực thi





Trình bày
các tín hiệu
điều khiển
khi thực thi
lệnh lw?

Các đường hoạt động khi lệnh lw thực thi



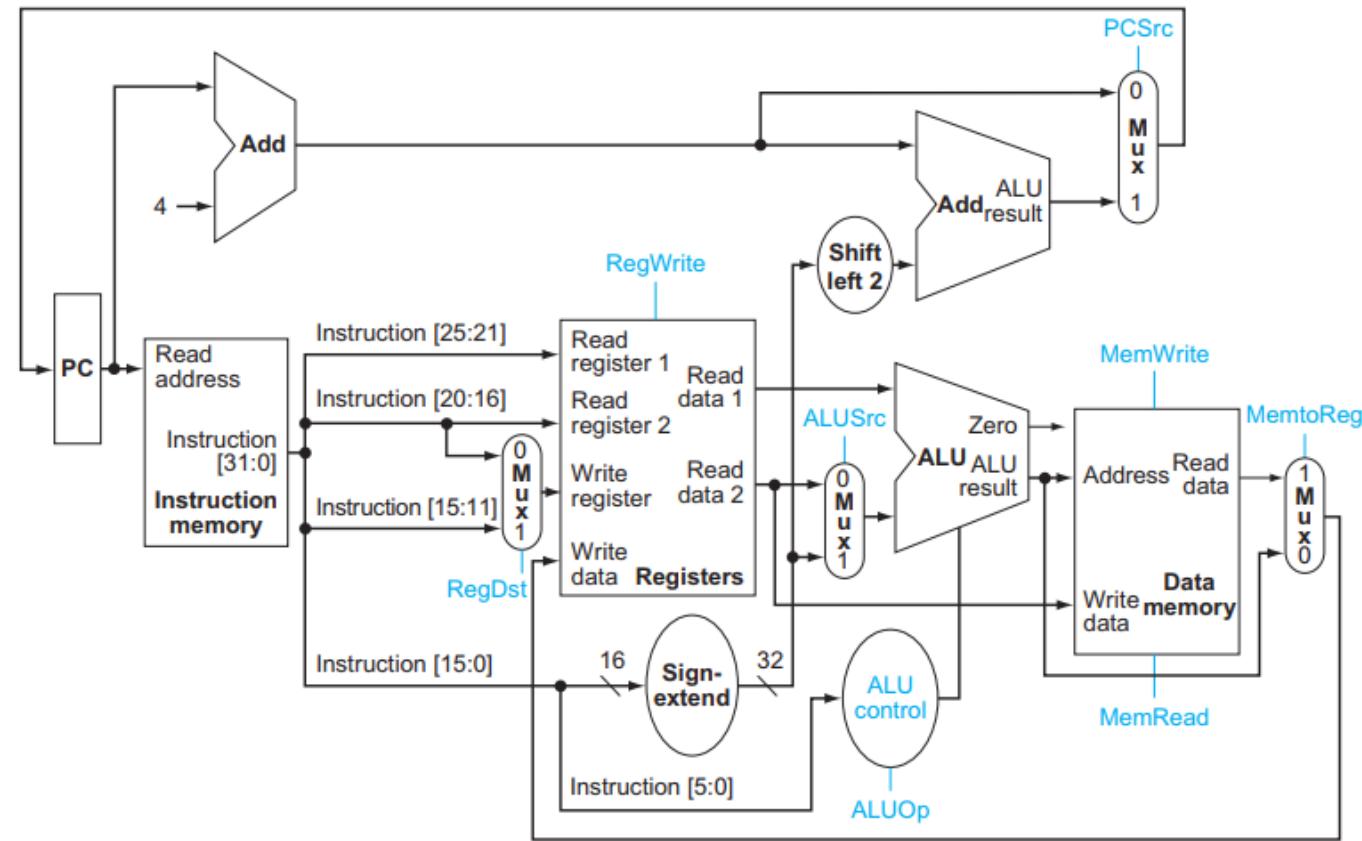


Bài tập 2

Mô tả đường đi dữ liệu và tính thời gian cần thiết tối thiểu để thực thi lệnh:

lw \$t0, 20(\$s2)

s2 = 0x00002023

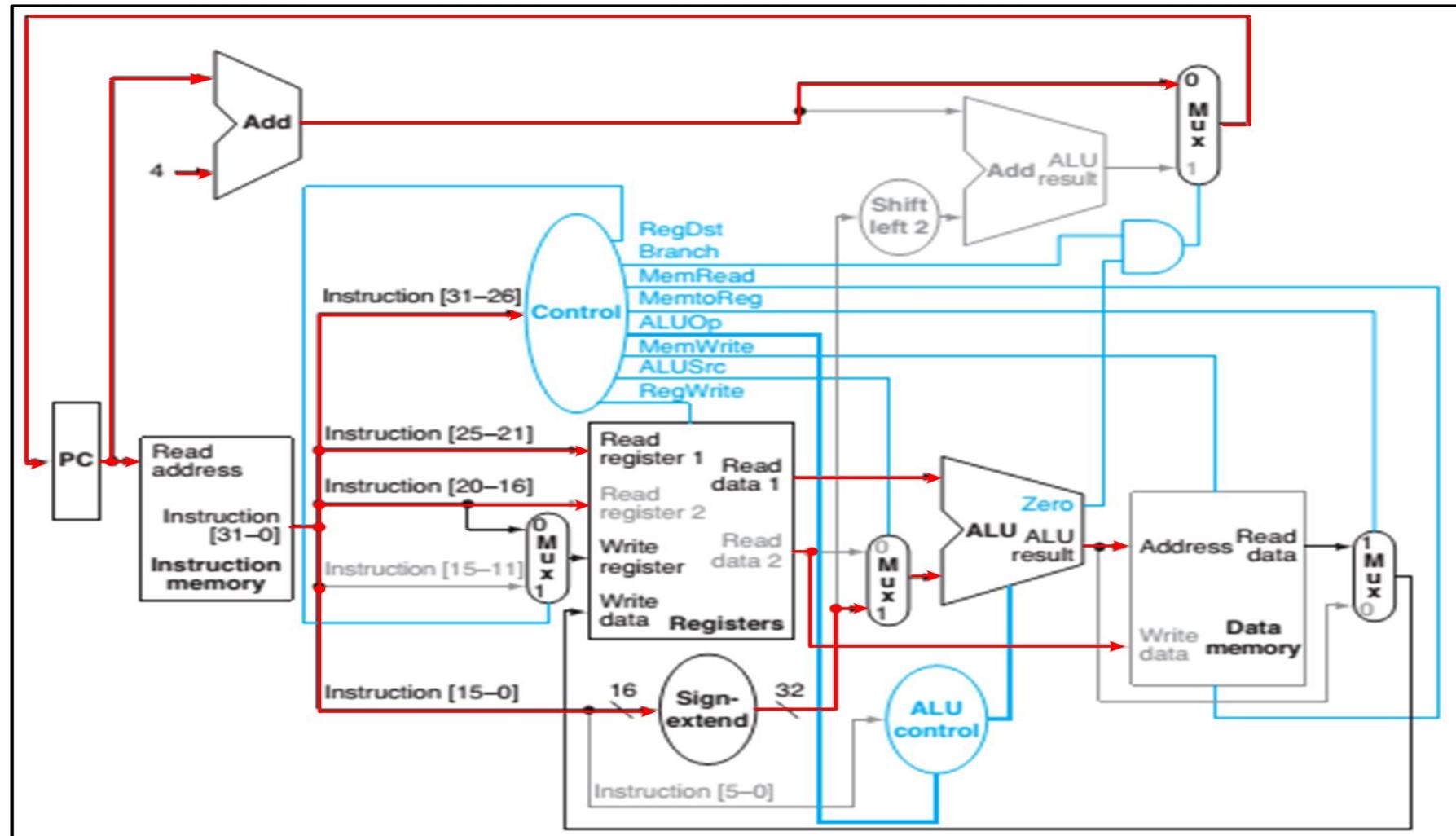


I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps



Trình bày
các khối
chức năng
được sử
dụng khi
thực thi
lệnh sw?

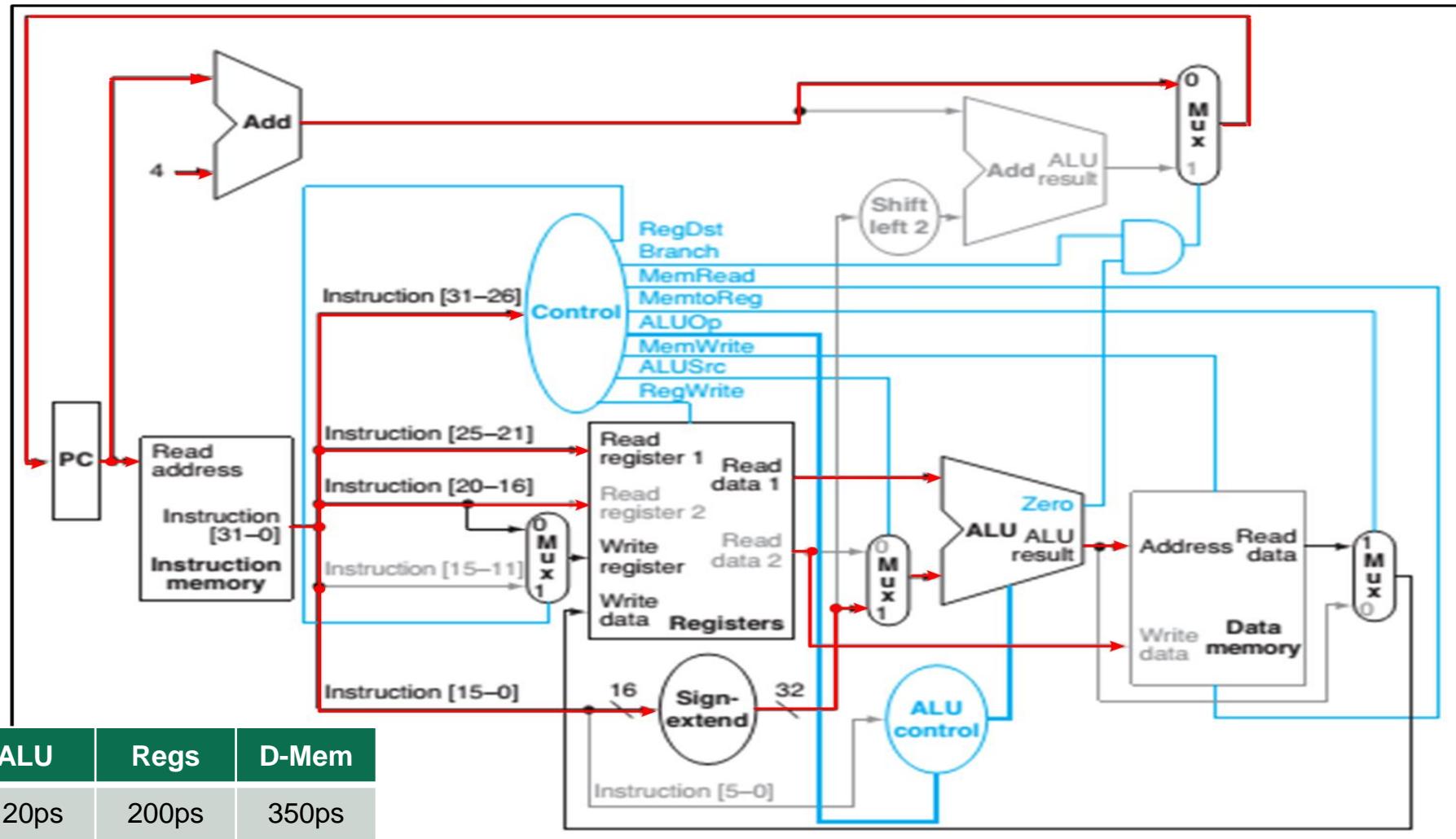
Các đường hoạt động khi lệnh sw thực thi





Các đường hoạt động khi lệnh sw thực thi

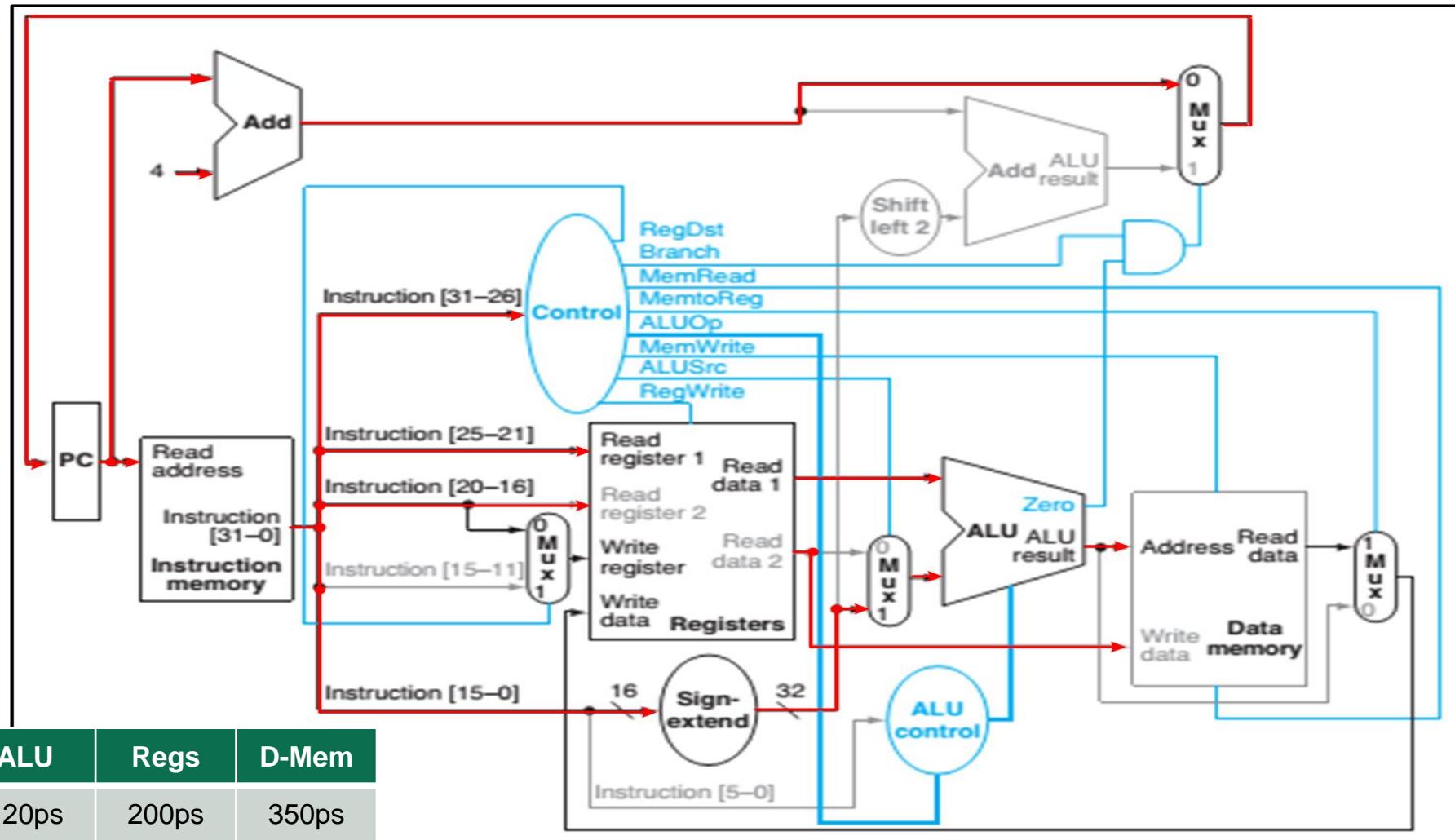
Tính thời gian cần thiết tối thiểu để thực thi lệnh sw?





Trình bày
các tín hiệu
điều khiển
khi thực thi
lệnh sw?

Các đường hoạt động khi lệnh sw thực thi





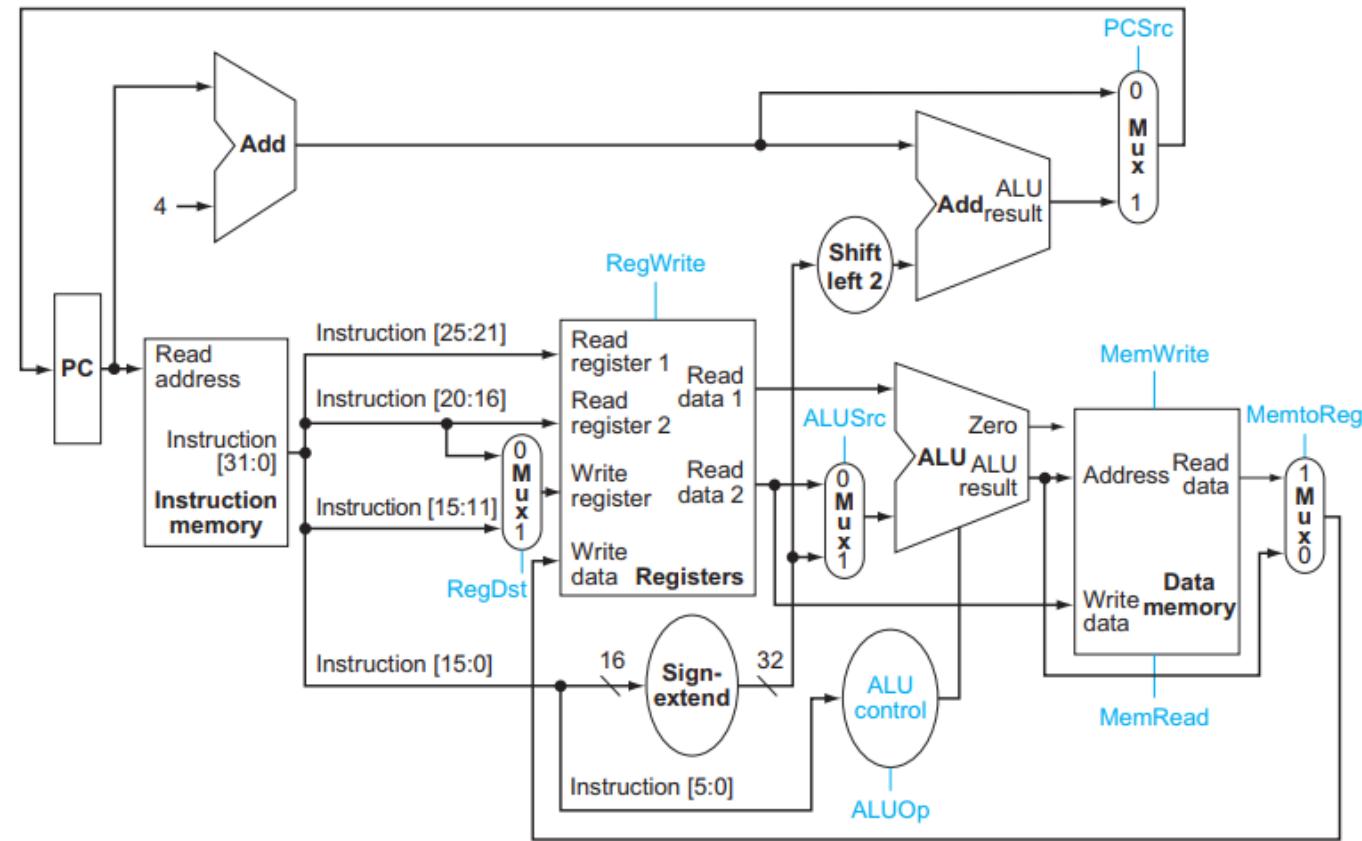
Bài tập 3

Mô tả đường đi dữ liệu và tính thời gian cần thiết tối thiểu để thực thi lệnh:

sw \$t0, 20(\$s2)

t0 = 0x00002022

s2 = 0x00002023

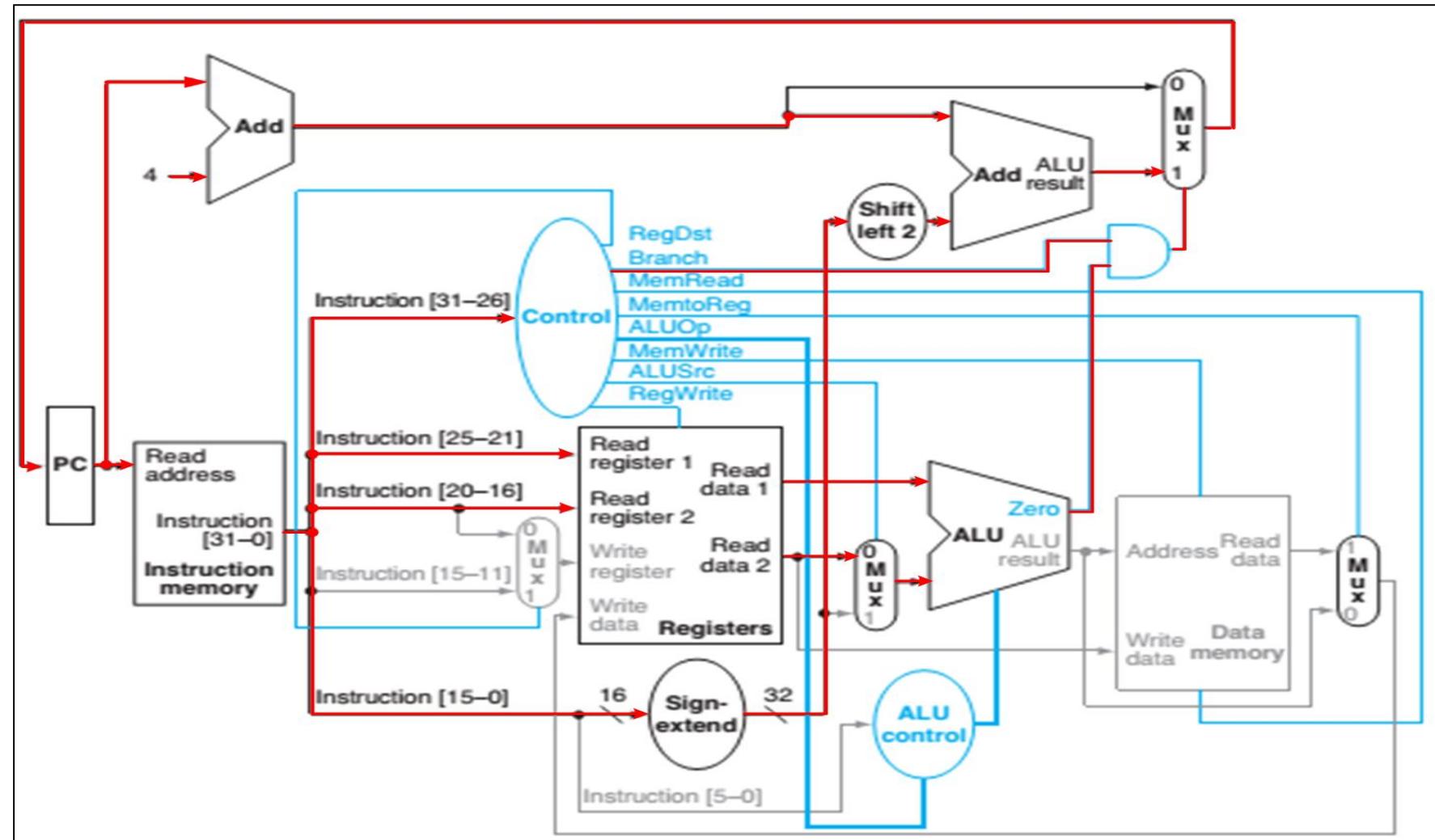


I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps



Trình bày
các khối
chức năng
được sử
dụng khi
thực thi
lệnh beq?

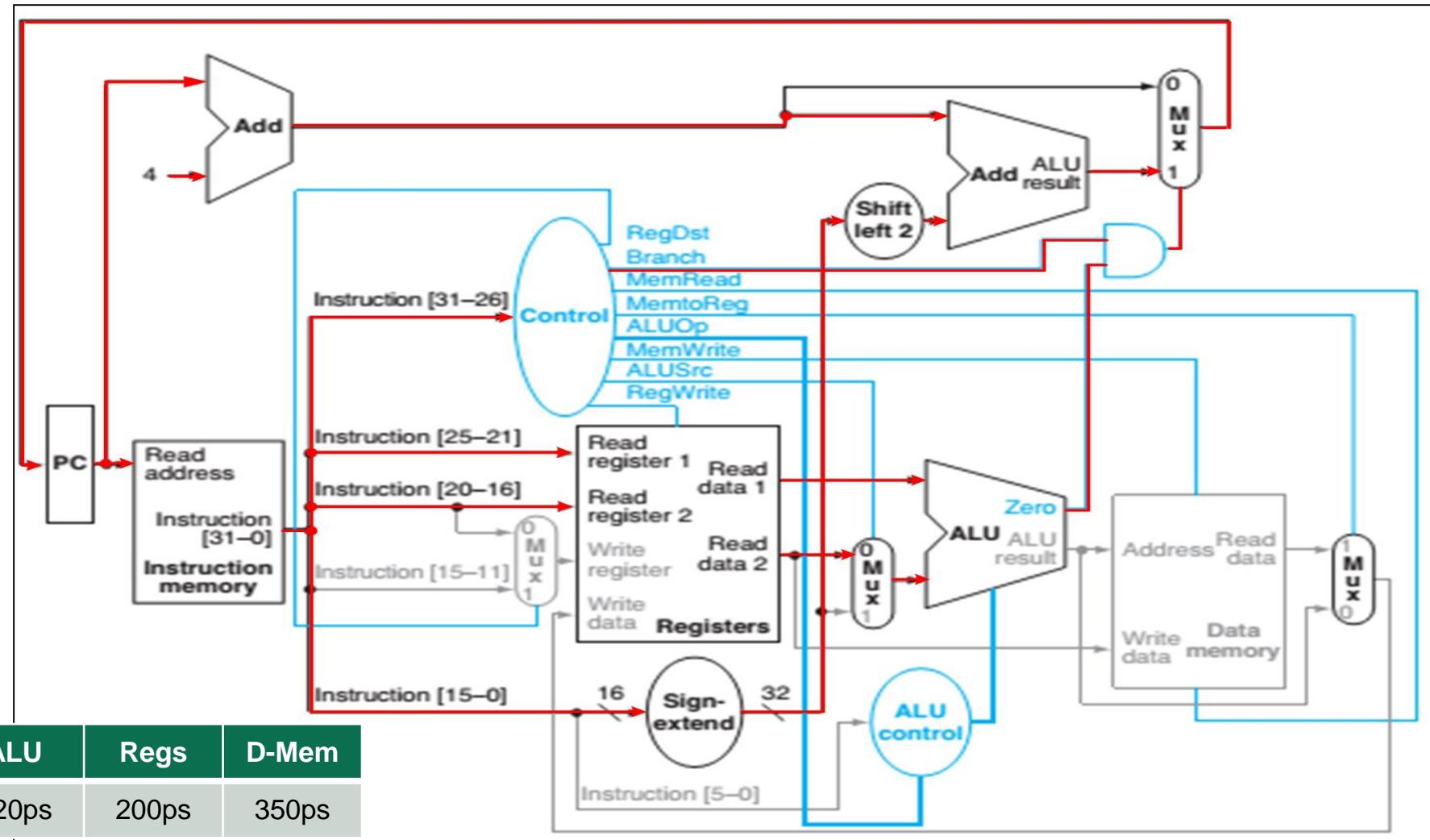
Các đường hoạt động khi lệnh beq thực thi





Tính thời gian cần thiết tối thiểu để thực thi lệnh beq?

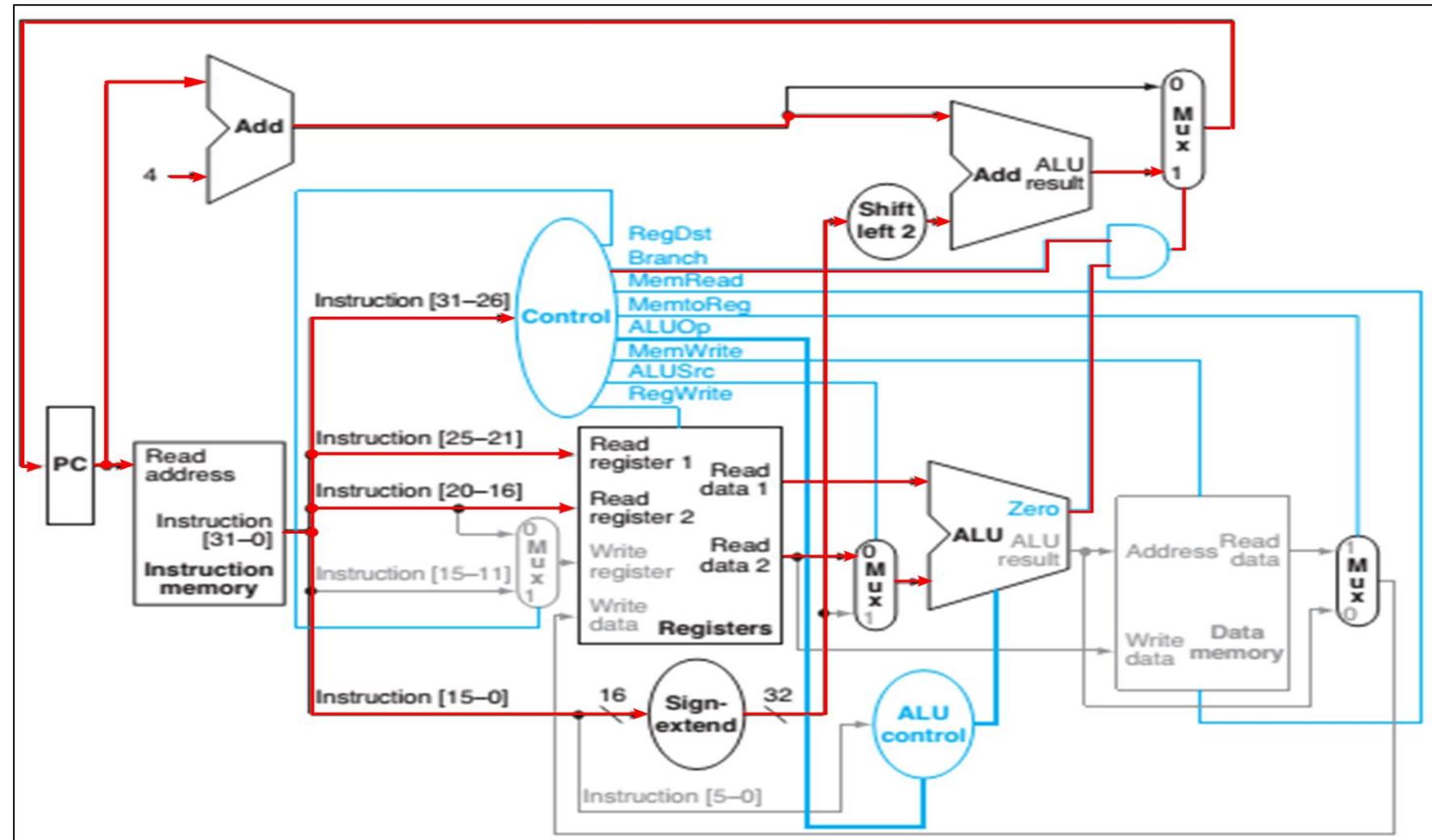
Các đường hoạt động khi lệnh beq thực thi





Trình bày
các tín hiệu
điều khiển
khi thực thi
lệnh beq?

Các đường hoạt động khi lệnh beq thực thi





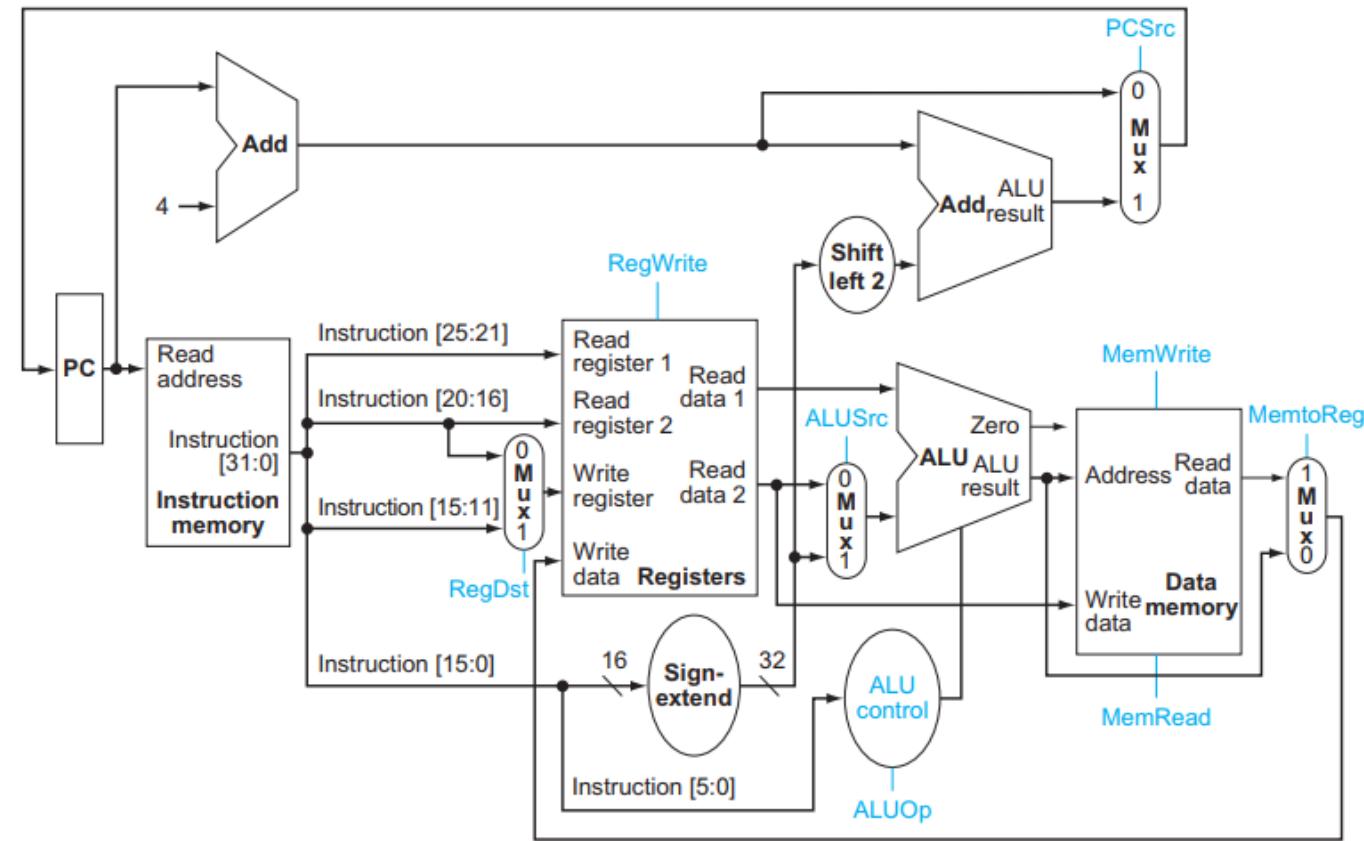
Bài tập 4

Mô tả đường đi
dữ liệu và tính
thời gian cần thiết
tối thiểu để thực
thi lệnh:

beq \$t0, \$t1, 3

t0 = 0x00002022

t1 = 0x00002022

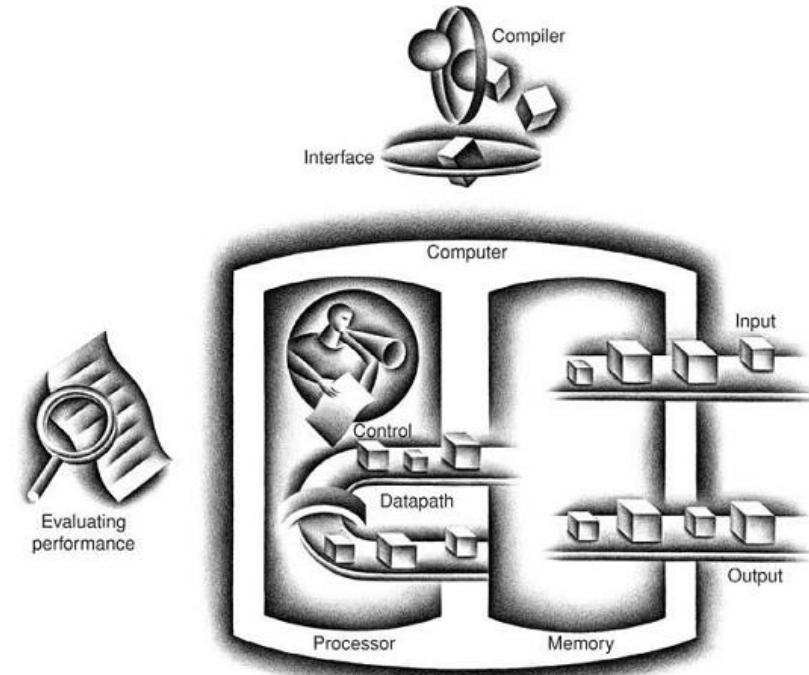


I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps



TỔNG KẾT

- **Kiến trúc tổng quan bộ xử lý**
- **Datapath**
- **Thực thi lệnh**
- **Bài tập**



Hình 1: Kiến trúc của 1 máy tính



BỘ XỬ LÝ

Thảo luận

