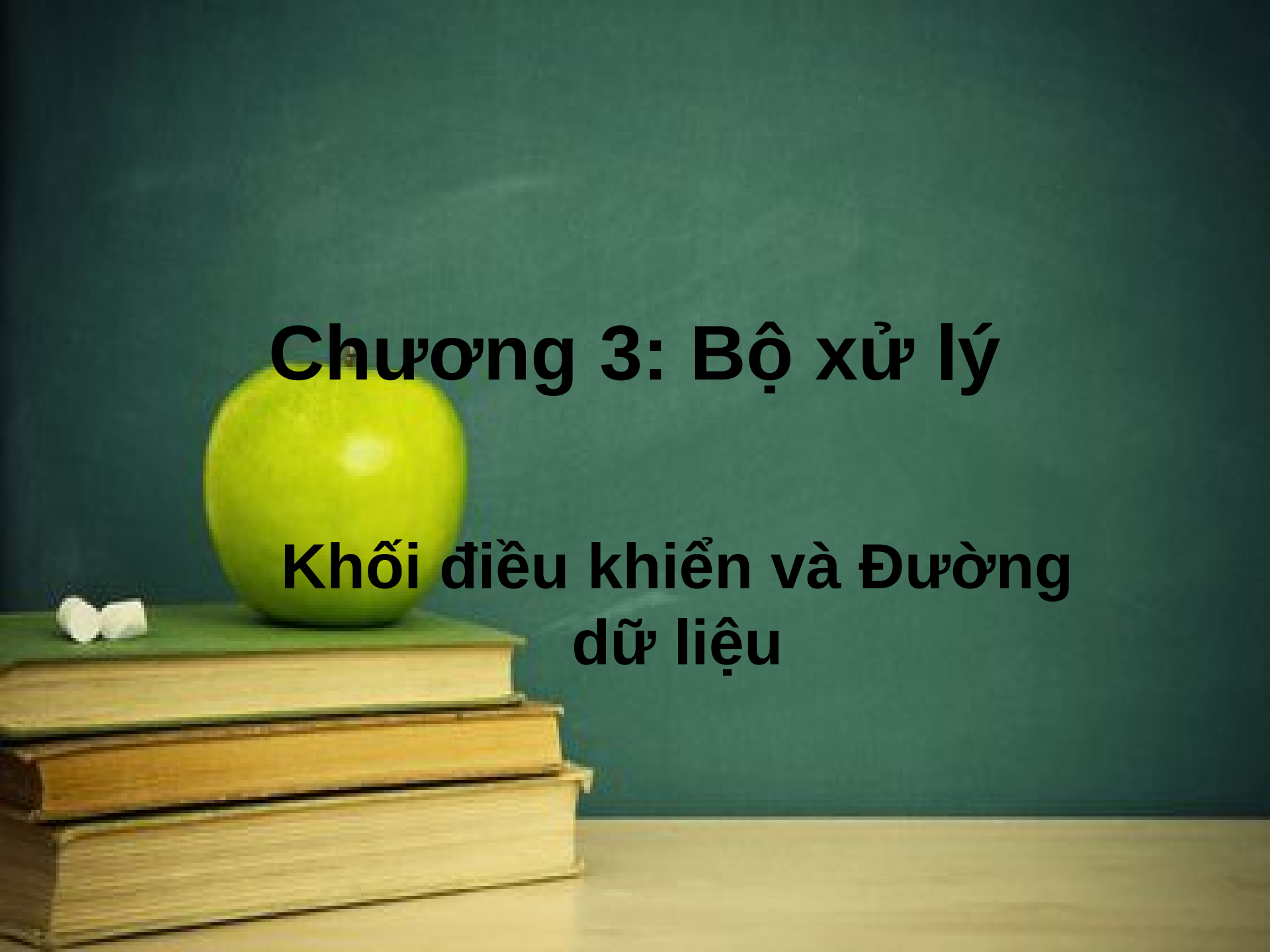


Chương 3: Bộ xử lý

Khối điều khiển và Đường dữ liệu

Nội dung

- **Thành phần cơ bản của bộ xử lý**
  - Lệnh truy cập
  - Các toán tử ALU
  - Toán tử bộ nhớ
- **Kết nối các thành phần**
  - Các tín hiệu điều khiển và bộ đồn kênh MUXes
  - Các chỉ thị giải mã lệnh

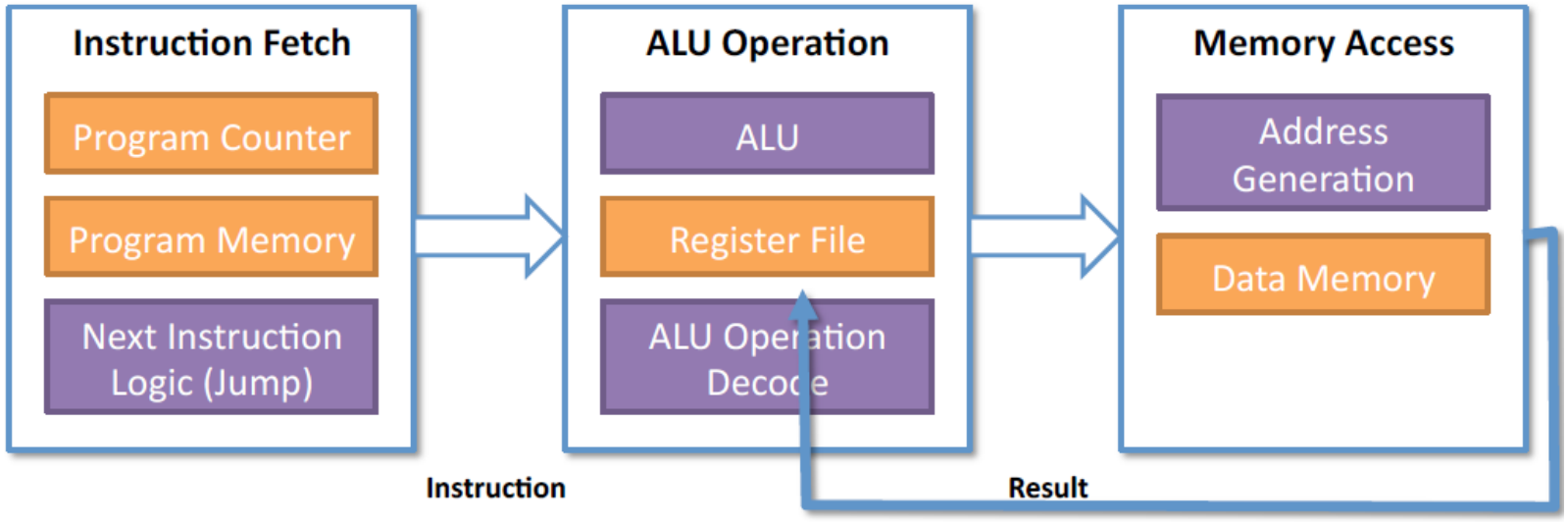
Material that is not in this lecture

- **Readings from the book**
  - – ALU Function field (fig.4.13 in 4.4)
  - – Some data path details
  - – The book has excellent descriptions of this topic.
- **Please read the book before watching this lecture.**
- **The reading assignment is on the**

Thực thi cơ bản của MIPS (from the book)

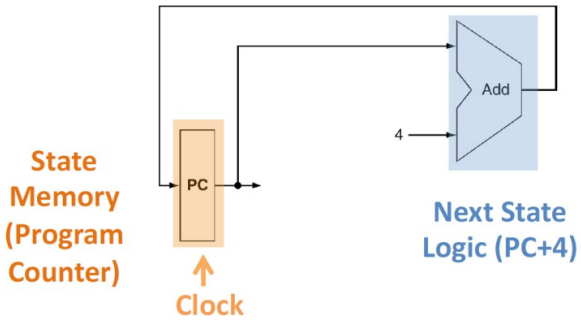
- Xem lại các tập lệnh cơ bản trong MIPS
  - Memory: lw, sw
  - Arithmetic: add,sub
  - Logic: and, or
  - Branch: beq
- Đọc thêm:
  - Multiply, divide
  - A bunch of logic operations
  - jump

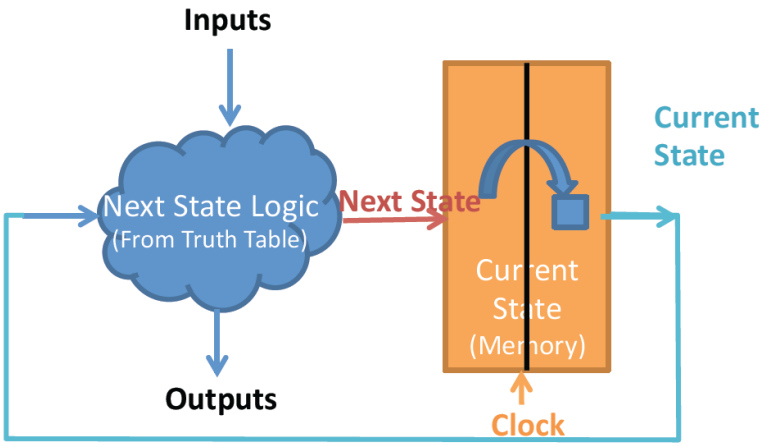
Các hoạt động chính của bộ xử lý?

- **Nạp lệnh**: tìm ra lệnh và tải lệnh
- **Tính toán trên ALU**: tìm ra toán tử và thực thi
- 

Thiết kế đơn xung nhịp

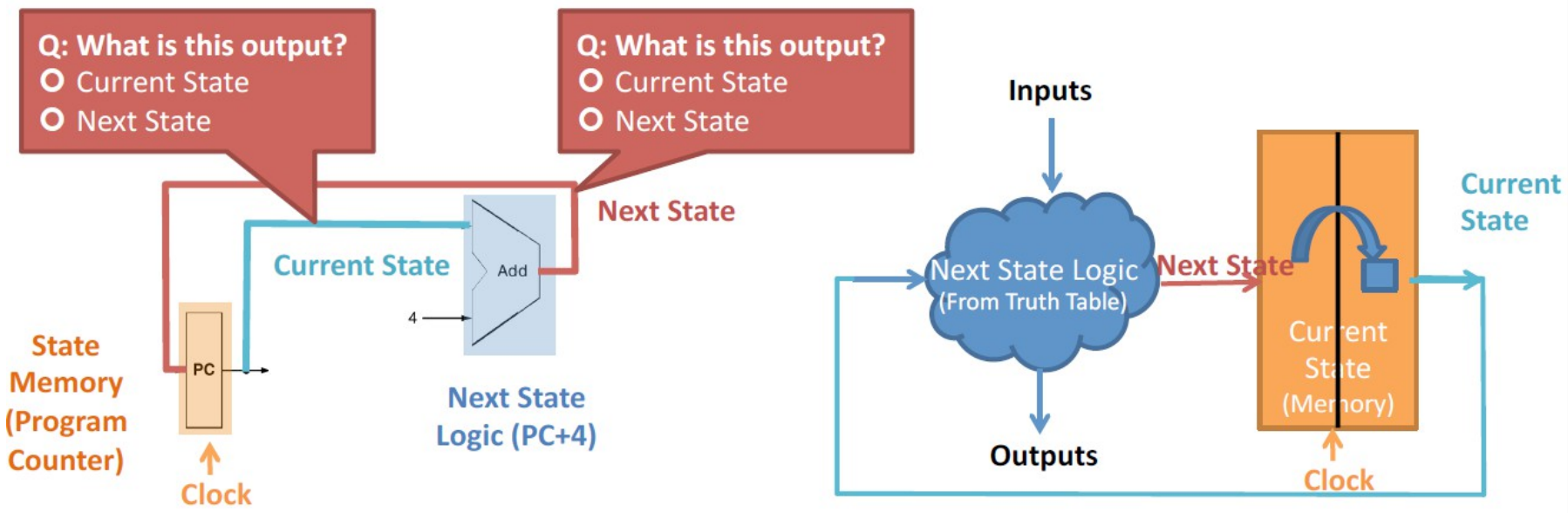
- Thiết kế đầu tiên sẽ xử lý một lệnh trong một chu kỳ đồng hồ. Chia lệnh thành các pha và thực hiện trong một chu kỳ đồng hồ.
- **Nhắc lại về thiết kế mức logic:**
  - **Tổ hợp các mức logic tạo ra trạng thái kế tiếp**
  - **Bộ nhớ** (các mạch chốt, RAM) lưu trữ trạng thái
  - **Bộ đồng hồ chuyển đổi trạng thái kế tiếp**
- **Quy trình nạp lệnh**
- **Trạng thái kế tiếp**: PC+4 (ngoại trừ các lệnh r
- **Trạng thái**: Program Counter (lệnh hiện tại)





Thiết kế đơn xung nhịp

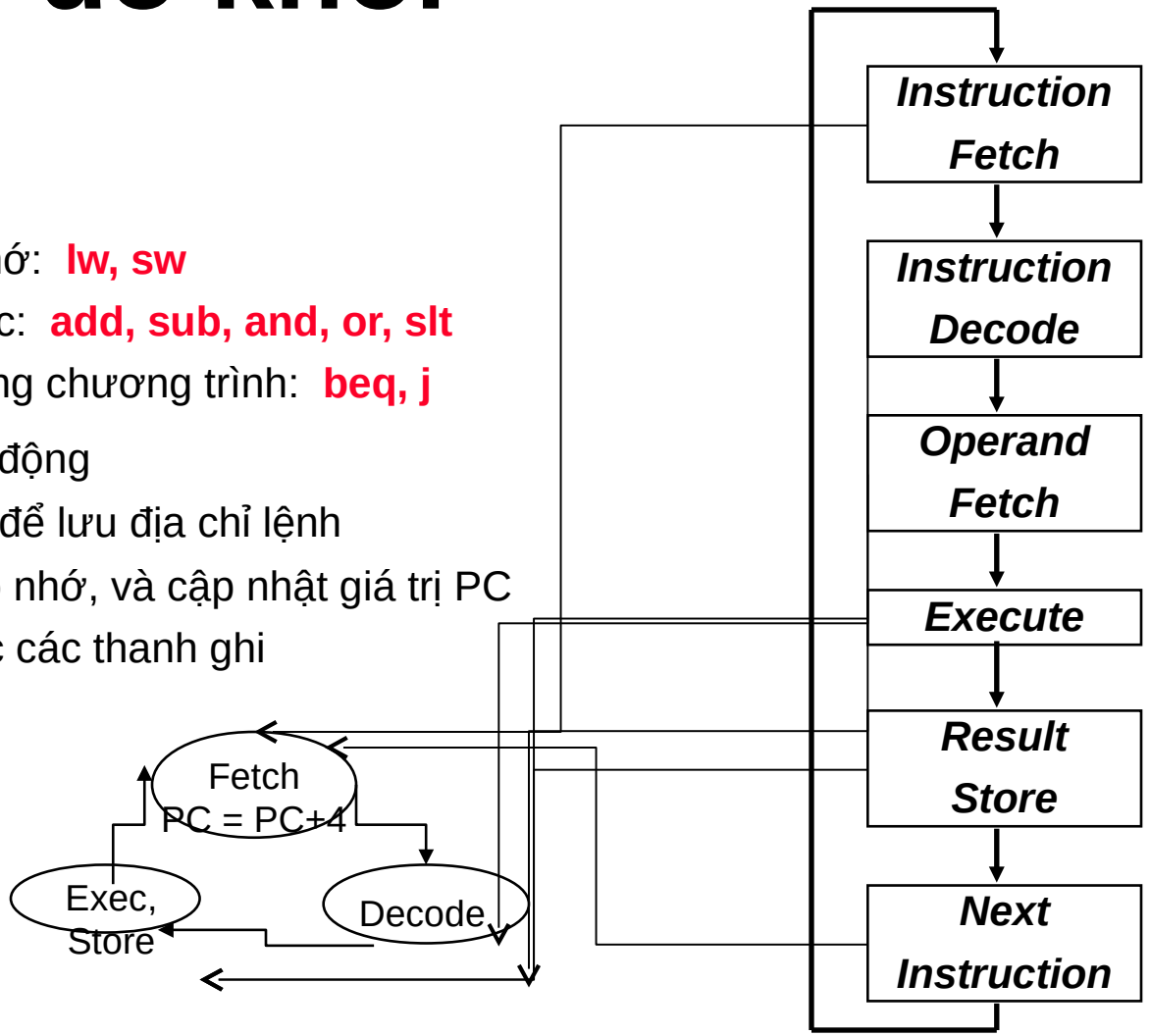
- Thiết kế xử lý một lệnh trong một chu kỳ đồng hồ
- **Các khối xử lý cơ bản:**
  - **Combinational logic tạo ra next state**



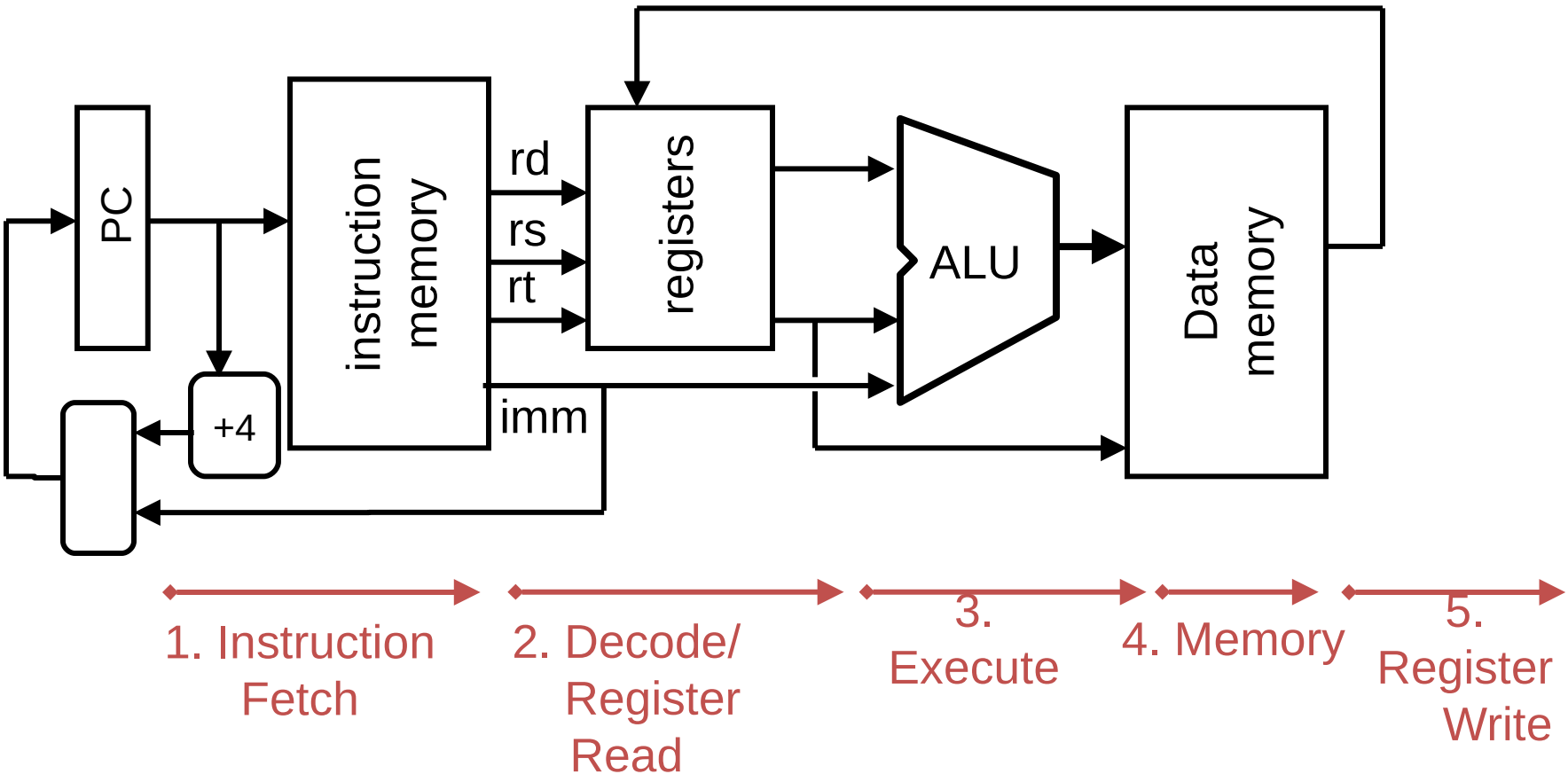
Sơ đồ khối

- Triển khai các lệnh
  - Lệnh truy cập bộ nhớ: **lw, sw**
  - Lệnh số học và logic: **add, sub, and, or, slt**
  - Lệnh điều khiển dòng chương trình: **beq, j**
- Triển khai các pha hoạt động
  - Dùng thanh ghi PC để lưu địa chỉ lệnh

    Đọc lệnh từ bộ nhớ, và cập nhật giá trị PC
  - Giải mã lệnh và đọc các thanh ghi
  - Thực hiện lệnh
  - Lưu kết quả

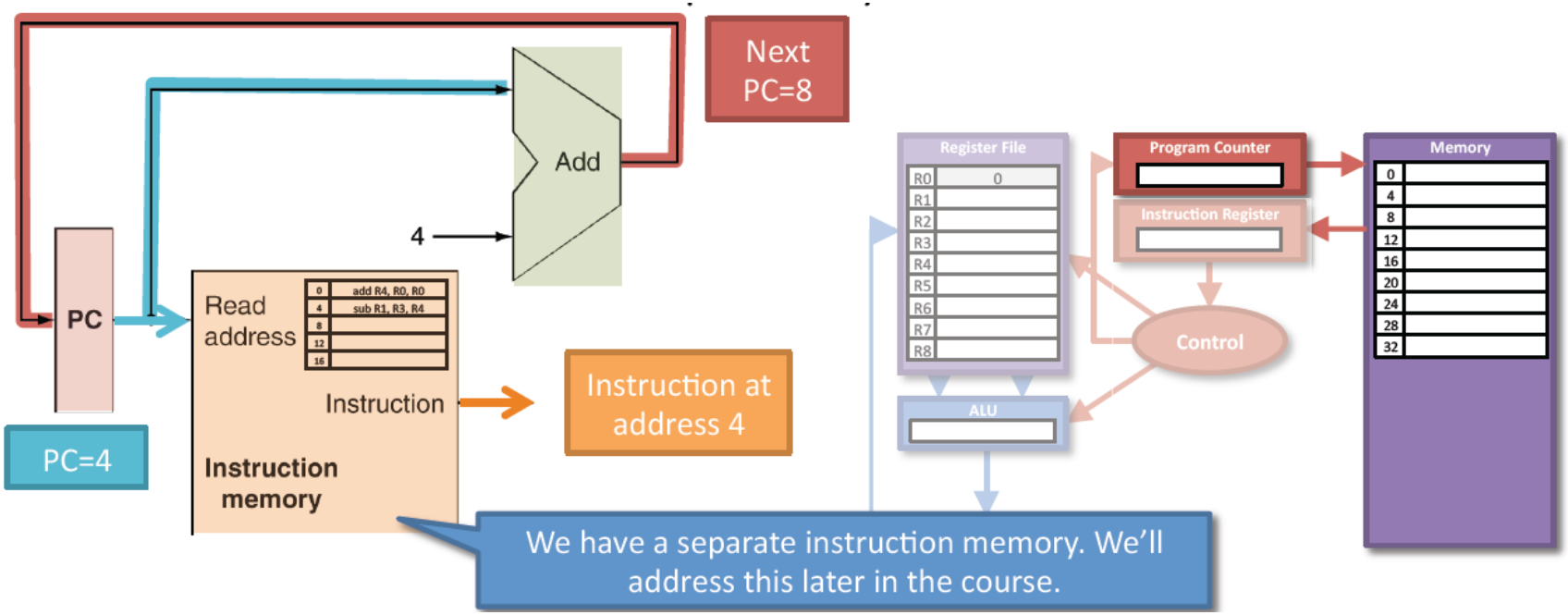


Stages of Execution on Datapath



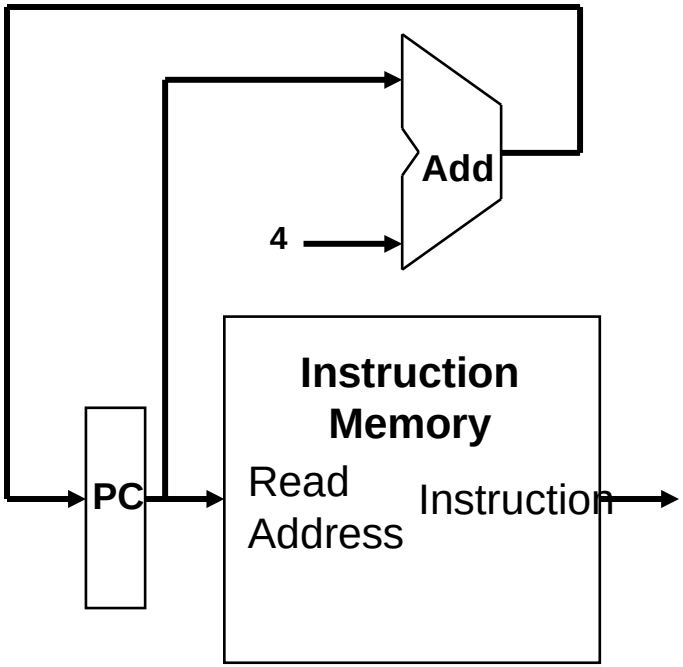
Nạp lệnh

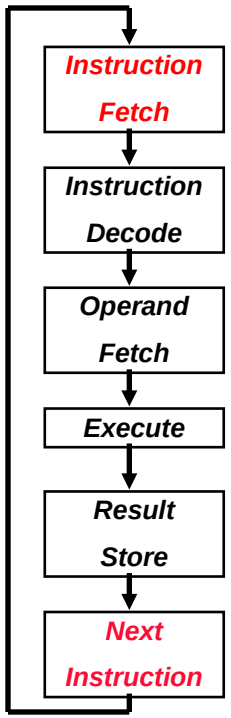
- Theo dõi địa chỉ lệnh hiện tại tên thanh PC.
  - **Tăng PC** lên 4 trong mỗi chu kỳ
  - **Tải lệnh tại địa chỉ được xác định bởi PC**



Nạp lệnh

- Đọc lệnh tại địa chỉ (lưu trong) PC từ bộ nhớ lệnh (*eng. Instruction Memory*)
- Cập nhật giá trị PC tới địa chỉ của lệnh kế tiếp

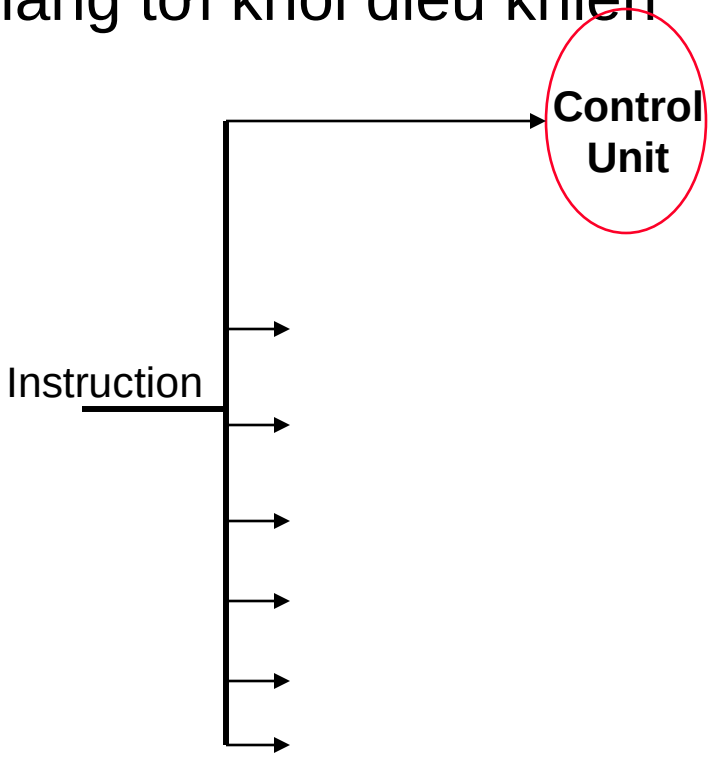


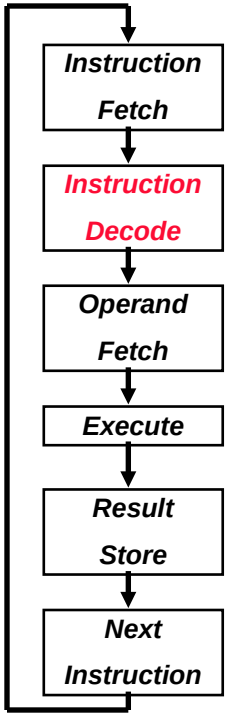


- PC được cập nhật ở mọi chu kỳ ➔ không cần tín hiệu điều khiển ghi PC.
- Đọc từ bộ nhớ lệnh được thực hiện bằng logic tổ hợp

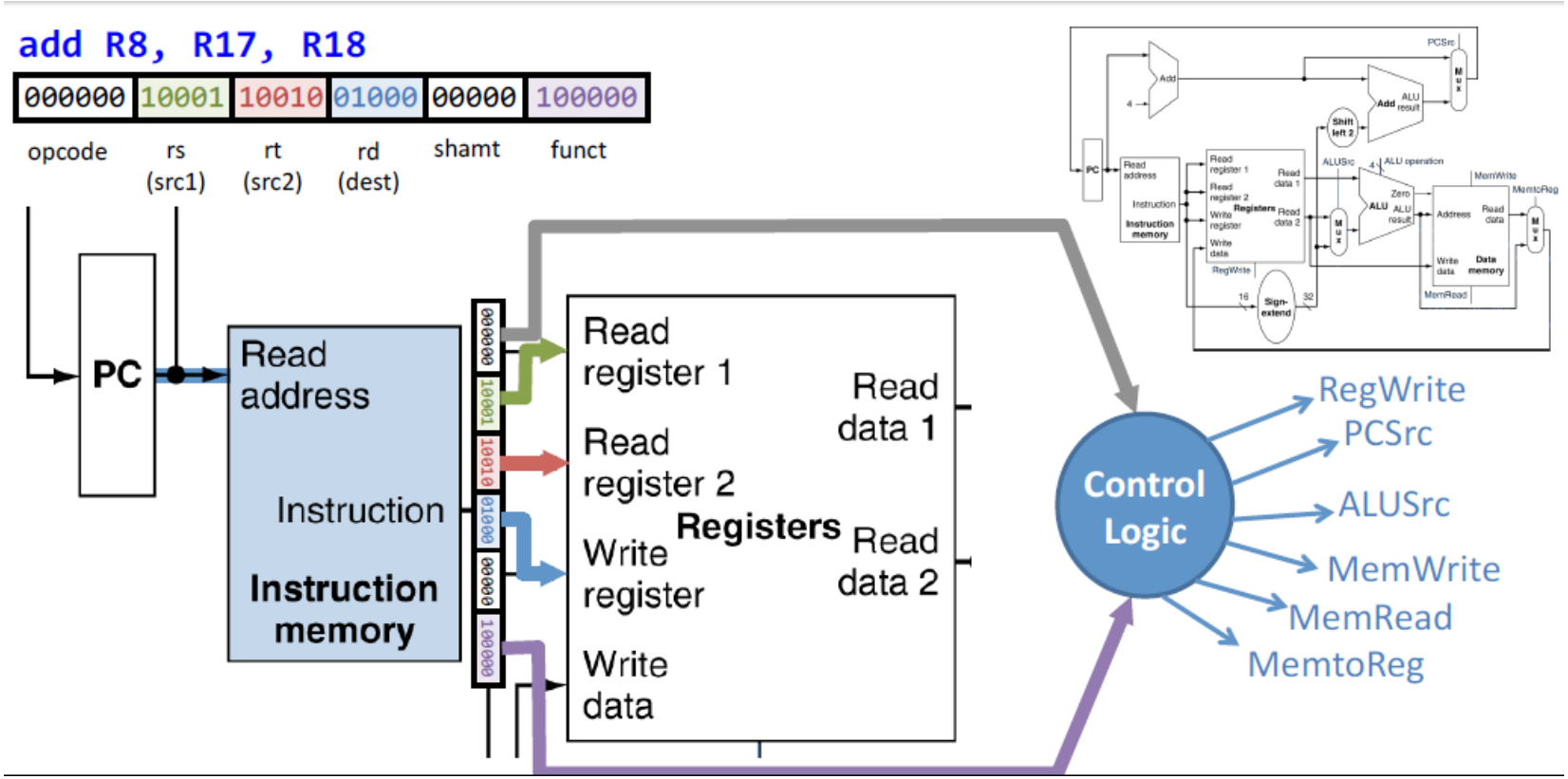
Giải mã lệnh

- Chuyển các bit thuộc trường mã lệnh và trường mã chức năng tới khối điều khiển

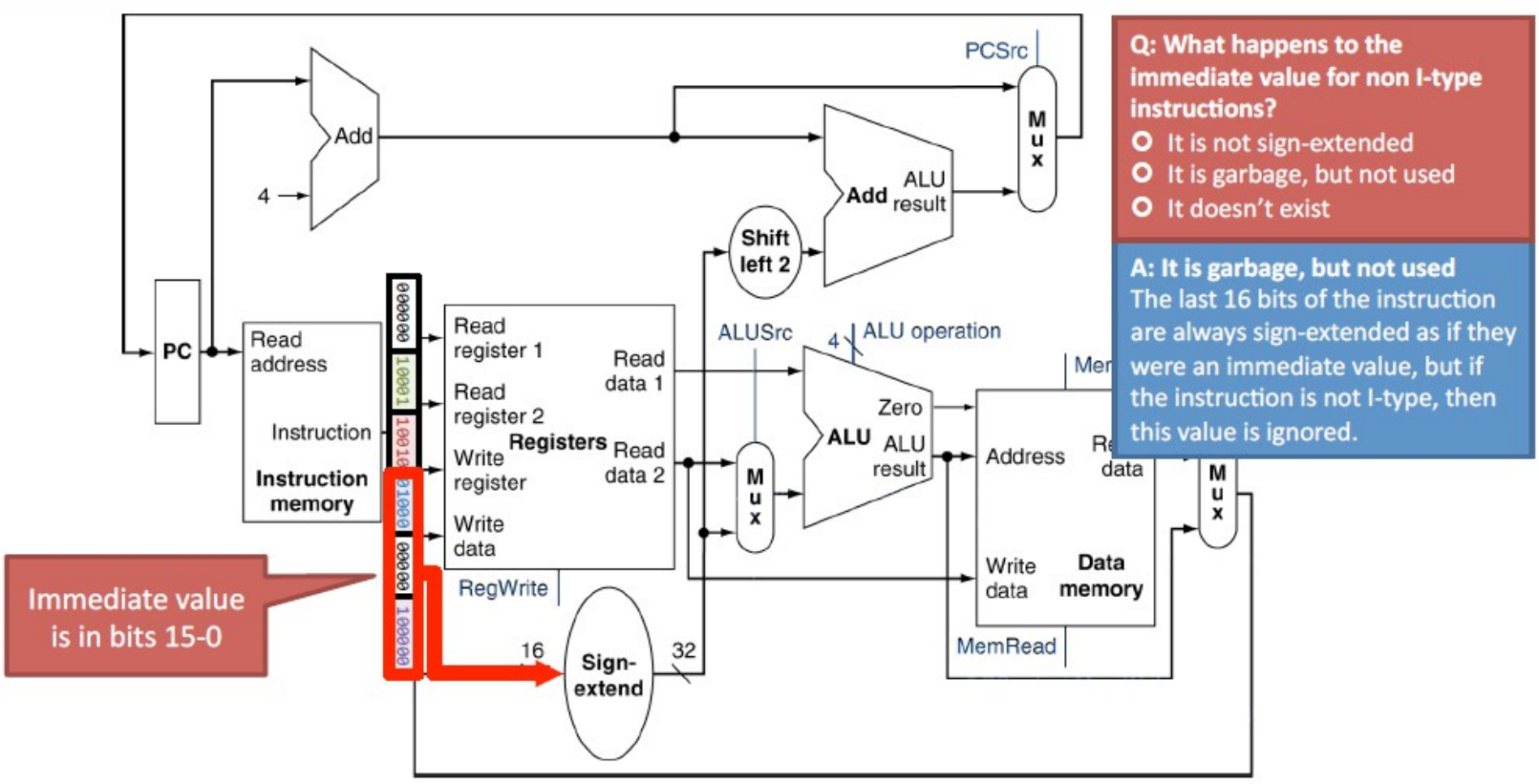




Giải mã lệnh (lệnh R)



Giải mã lệnh (Lệnh trực tiếp)



Q: What happens to the immediate value for non I-type instructions?

- It is not sign-extended
- It is garbage, but not used
- It doesn't exist

A: It is garbage, but not used

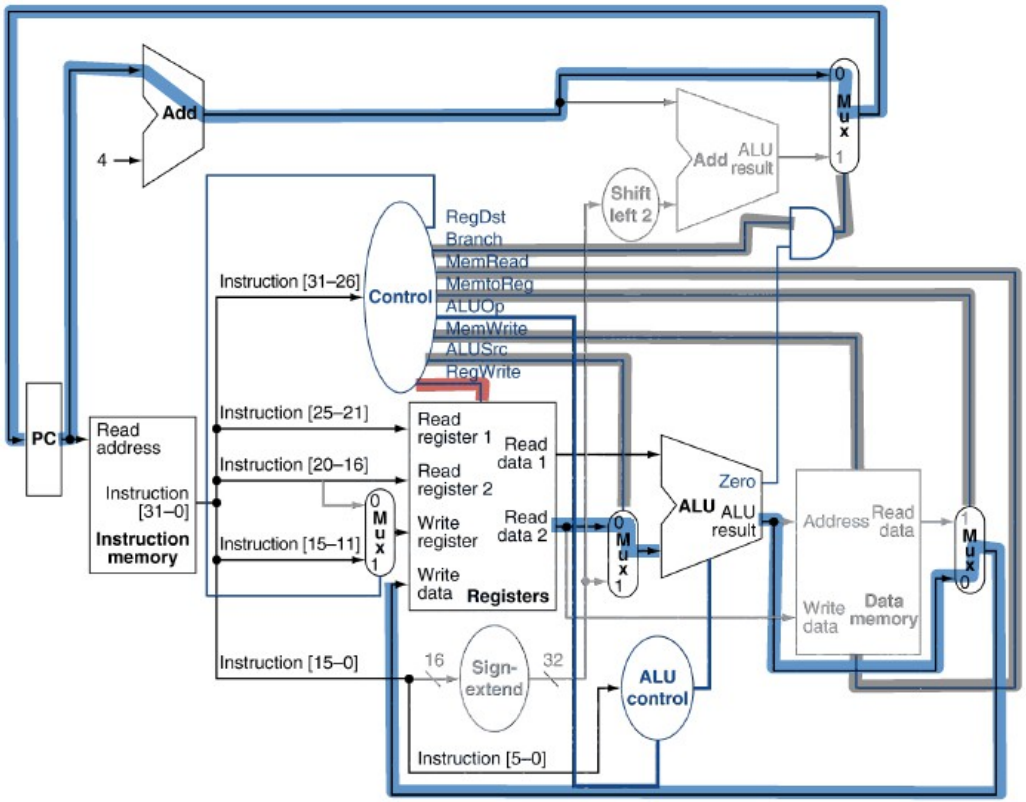
The last 16 bits of the instruction are always sign-extended as if they were an immediate value, but if the instruction is not I-type, then this value is ignored.

Giải mã lệnh (các tín hiệu điều khiển)

Signal	Meaning	When?
RegWrite	RF[Write register] ← write data (store write data in RF)	Any instruction that writes to the RF
ALUSrc	ALU uses sign-extended input (ALU operation with immediate)	Any I-format instruction
PCSrc	PC ← PC + 4 + Immediate (jump)	Branch taken
MemRead	Read from memory	loads
MemWrite	Write to memory	stores
MemtoReg	RF[Write register] ← memory (store memory data in RF)	loads
ALUop	ALU Operation	Any instruction that uses the ALU

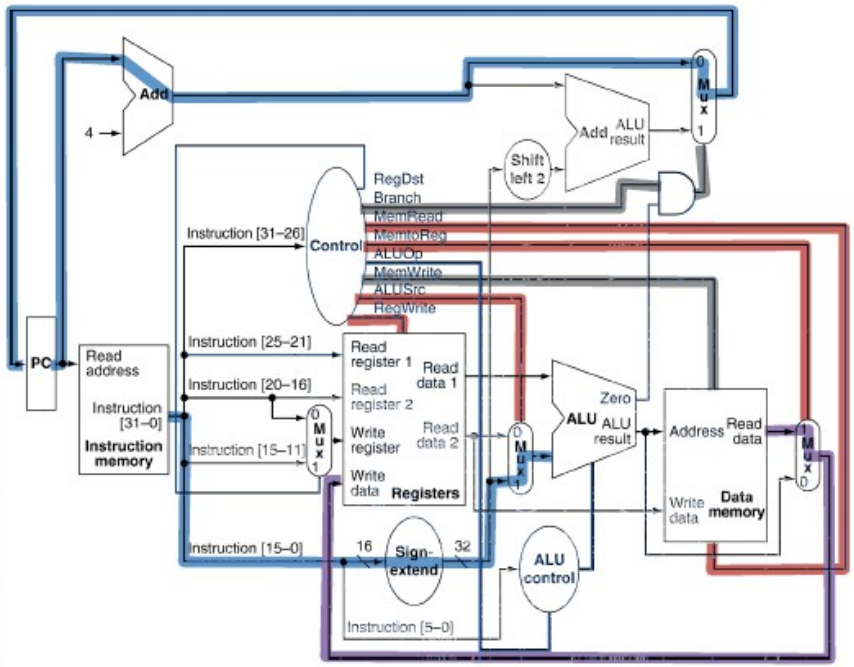
Giải mã lệnh (R-format)

Signal	Meaning	R-format
RegWrite	RF[Write register] ← write data (store write data in RF)	1 (writes to a register)
ALUSrc	ALU uses sign-extended input (ALU operation with immediate)	0 (does not use the immediate)
PCSrc	PC ← PC + 4 + Immediate (jump)	0 (does not jump)
MemRead	Read from memory	0 (does not read from memory)
MemWrite	Write to memory	0 (does not write to memory)
MemtoReg	RF[Write register] ← memory (store memory data in RF)	0 (does not store data from memory)
ALUop	ALU Operation	depends



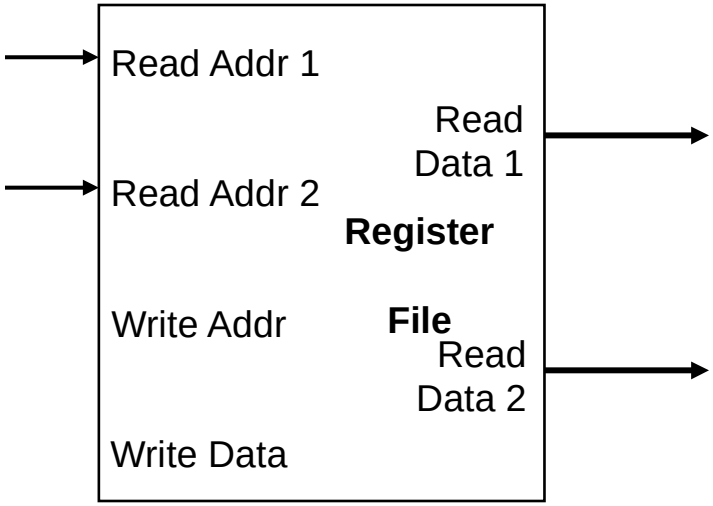
Giải mã lệnh (load)

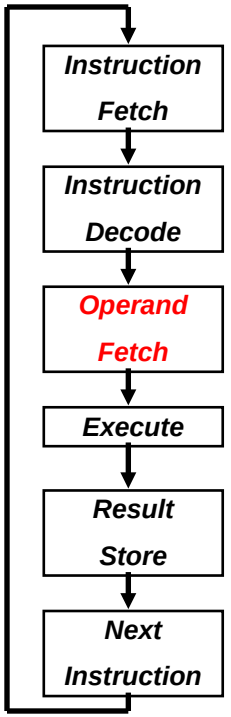
Signal	Meaning	R-format
RegWrite	RF[Write register] ← write data (store write data in RF)	1 (writes to a register)
ALUSrc	ALU uses sign-extended input (ALU operation with immediate)	1 (uses the immediate for address)
PCSrc	PC ← PC + 4 + Immediate (jump)	0 (does not jump)
MemRead	Read from memory	1 (reads from memory)
MemWrite	Write to memory	0 (does not write to memory)
MemtoReg	RF[Write register] ← memory (store memory data in RF)	1 (stores data from memory)
ALUop	ALU Operation	add (address calculation)



Nạp toán hạng

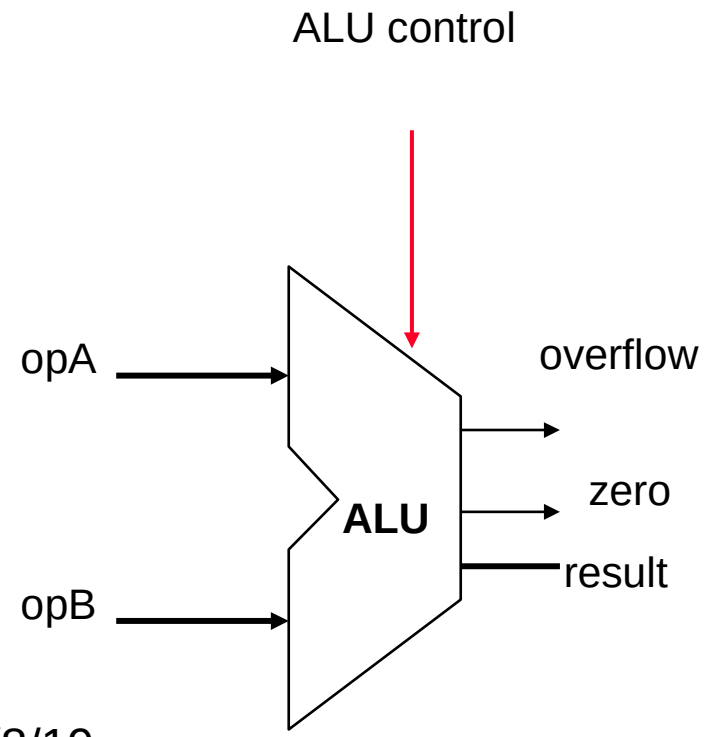
- Đọc 2 giá trị toán hạng nguồn từ tệp thanh ghi
  - Chỉ số các thanh ghi nằm trong lệnh
- Mở rộng dấu cho toán hạng trực tiếp I

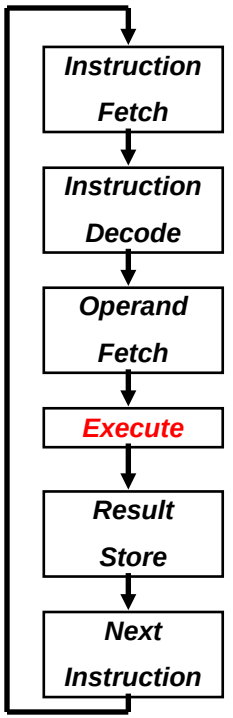




Thực hiện lệnh

- Thực hiện phép toán (mã hóa bởi op và funct) trên giá trị toán hạng opA và opB
  - Các phép toán của lệnh R và I
  - Phép toán tính địa chỉ trong lệnh lw, sw
  - Phép toán so sánh trong lệnh beq, bne

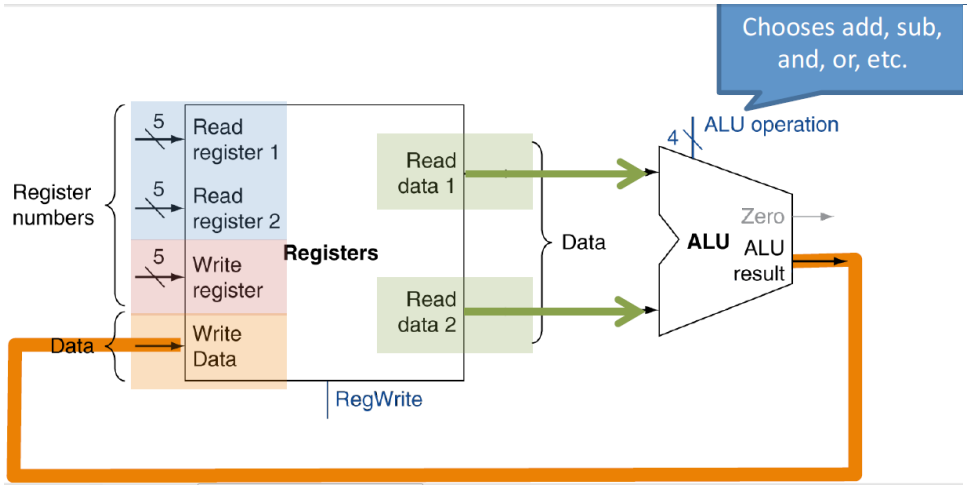


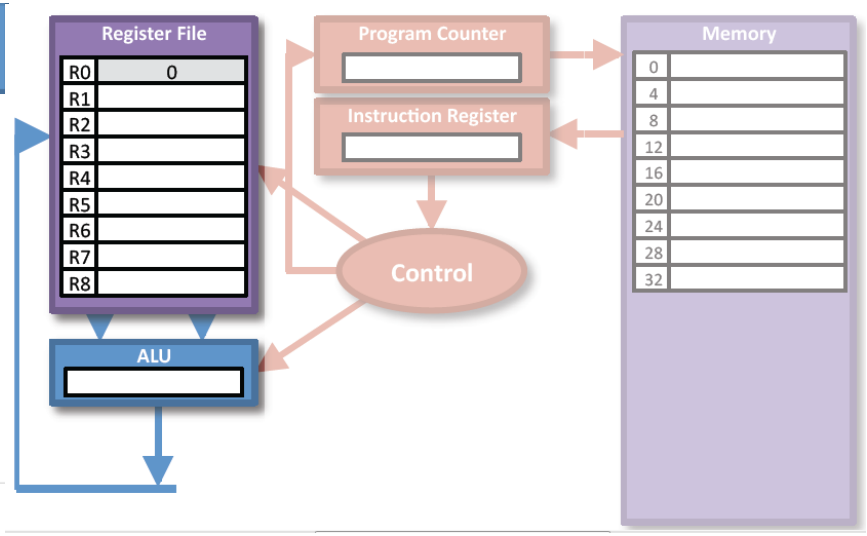


Tính toán trên ALU (các chỉ thị lệnh dạng R-type)

Các bước thực hiện?

- **Đọc dữ liệu từ tệp thanh ghi**(specify **rs** and **rt**)
- Thực thi tính toán **ALU**
- Ghi dữ liệu về tệp thanh ghi (specify **rd**)





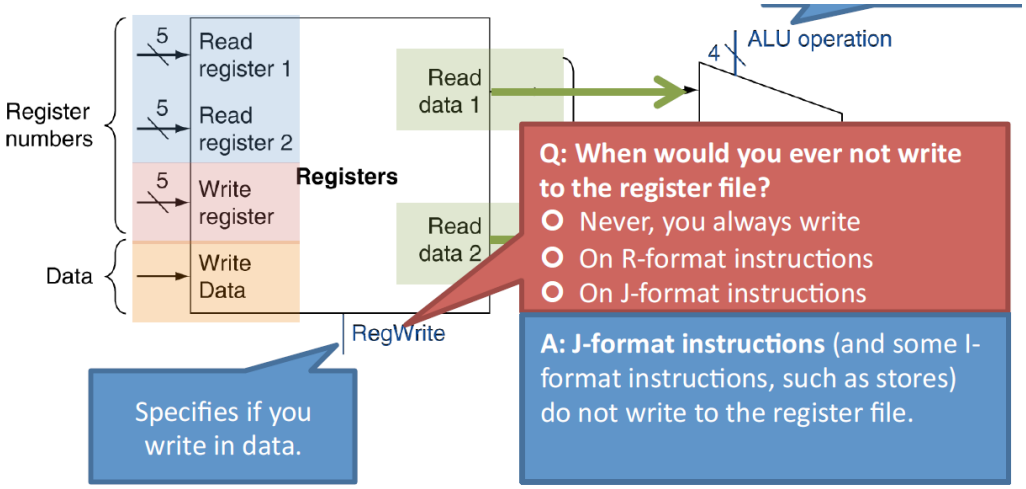
Tính toán trên ALU

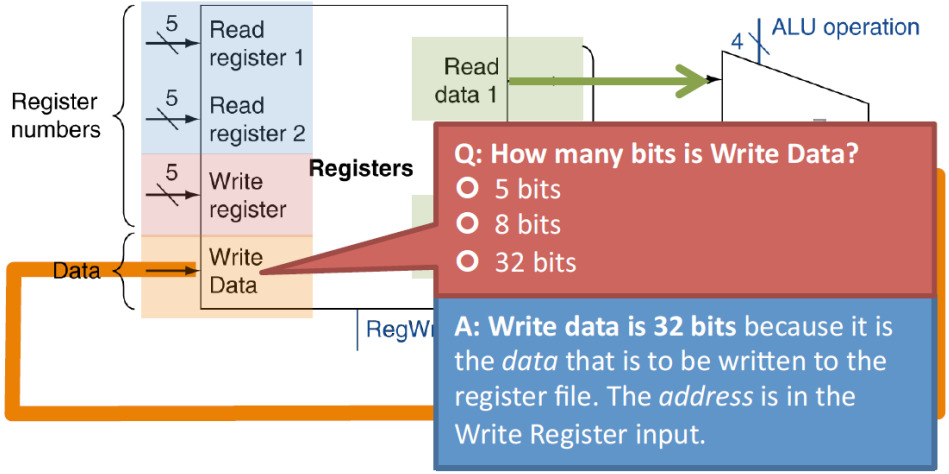
- Các bước thực hiện?

  – **Đọc dữ liệu từ tệp thanh ghi** (xác định rõ **rs** và **rt**)

  – Thực hiện tính toán **ALU**

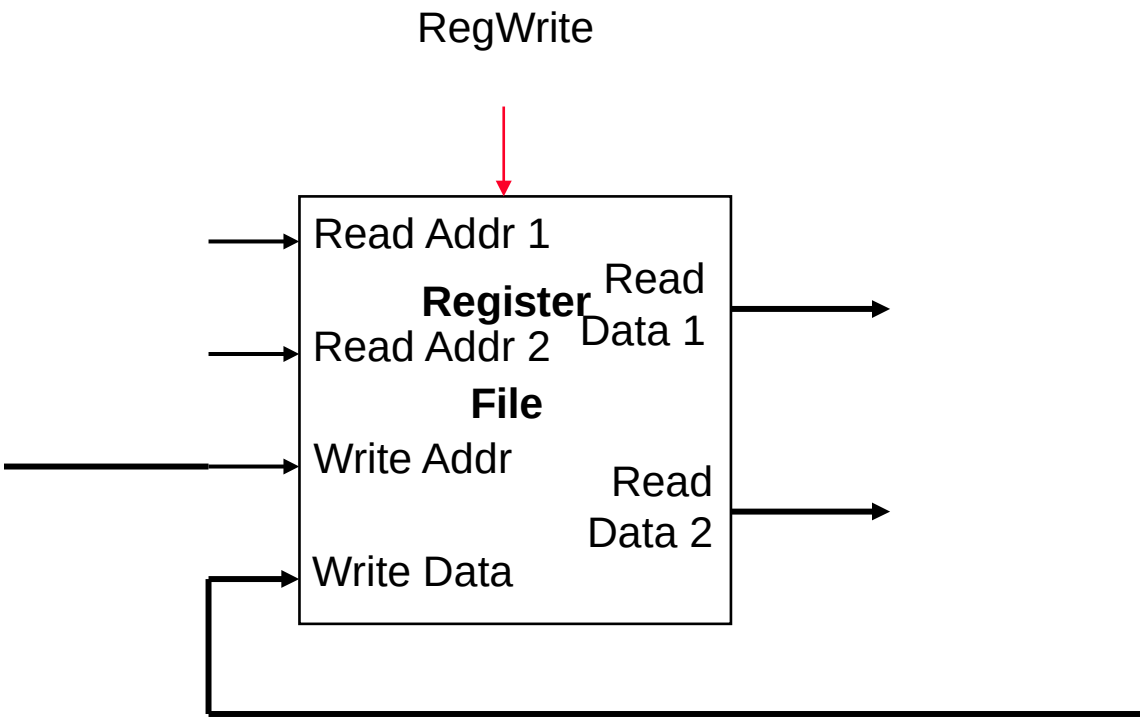
  – **Ghi dữ liệu trở lại tệp thanh ghi** (xác định rõ **rd**)

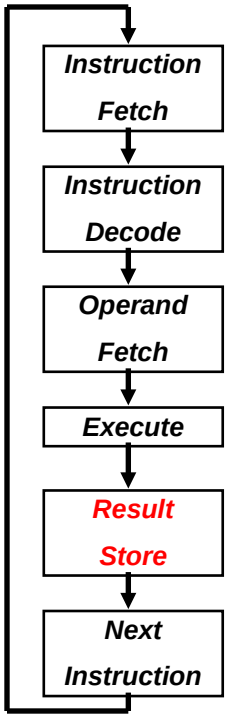




Ghi kết quả

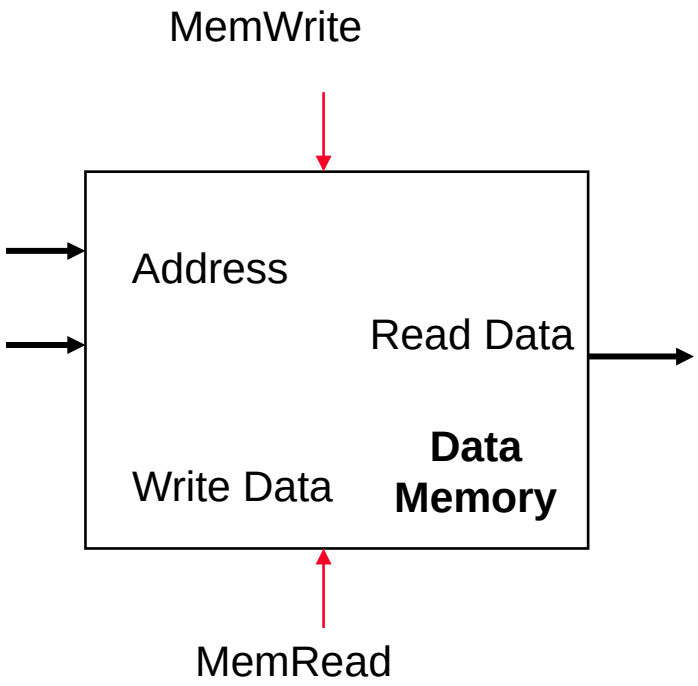
- Từ ALU với các lệnh tính toán
- Từ bộ nhớ với các lệnh truy cập bộ nhớ

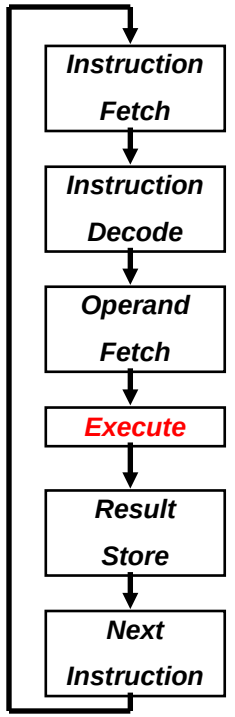




Truy cập bộ nhớ

- Địa chỉ theo byte
- Dữ liệu từ thanh ghi rt

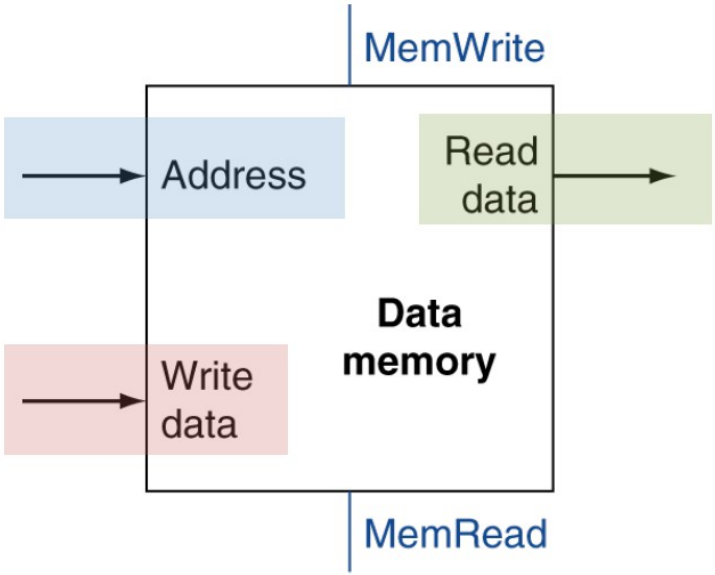




Truy cập bộ nhớ

Các bước thực hiện?

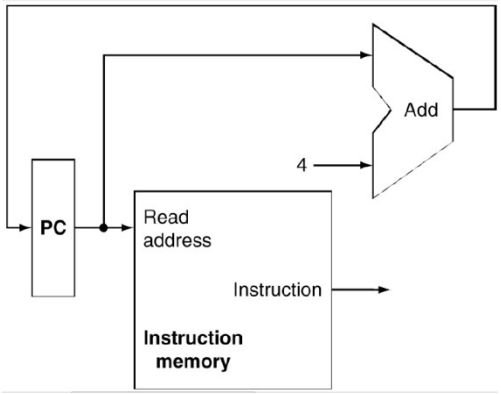
- **Tính toán địa chỉ**
- **Gửi địa chỉ đến bộ nhớ dữ liệu** (write: data)
- Đọc: nhận kết quả trả về và đưa vào tệp thanh ghi



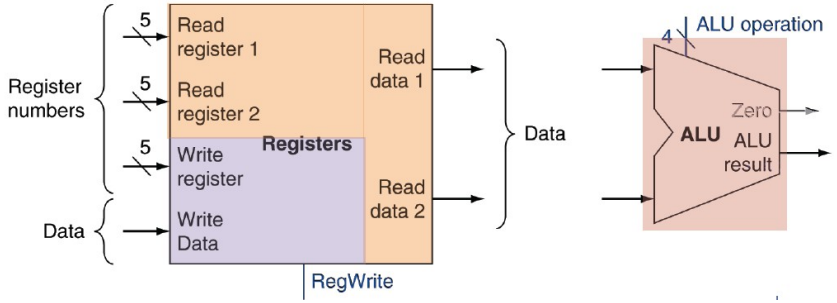
Q: Điều gì xảy ra nếu đọc ghi vào bộ nhớ cùng một thời điểm
1/ không thể xảy ra
2/ dữ liệu sẽ bị sai lệch hoặc không hợp lệ
3/ không có vấn để gì cả vì có đầu ra đọc dữ liệu và đầu vào ghi dữ liệu

A: 2
Hoạt động của RAM: kích hoạt một hàng đọc/ghi dữ liệu. Nếu thực hiện cả 2 cùng lúc sẽ làm các bit bị xáo trộn. Ngoài ra cần phải 2 địa chỉ cho quá trình ghi đọc

Kết nối các thành phần

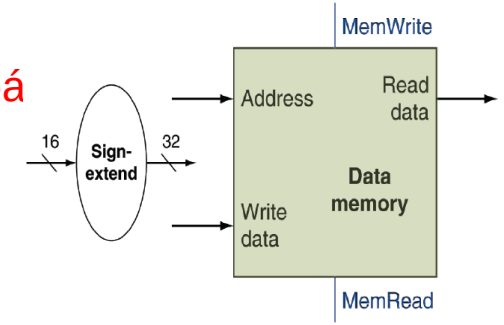


- Bộ tính toán **ALU**
  - tải lệnh
  - tính toán giá trị tiếp theo của PC
  - đọc từ tệp thanh ghi
  - **thực thi tính toán**
  - ghi lại tệp thanh ghi
  - **Truy nhập bộ nhớ (load/store)**
  - tải lệnh
  - tính toán giá trị tiếp theo củaPC
  - đọc từ tệp thanh ghi
  - **tính toán địa chỉ**
  - **Read/Write dữ liệu bộ**
  - Write dữ liệu vào tệp thanh ghi
  - Nạp lệnh **(branch)**
  - tải lệnh
  - tính toán giá trị tiếp theo của PC
  - đọc từ tệp thanh ghi

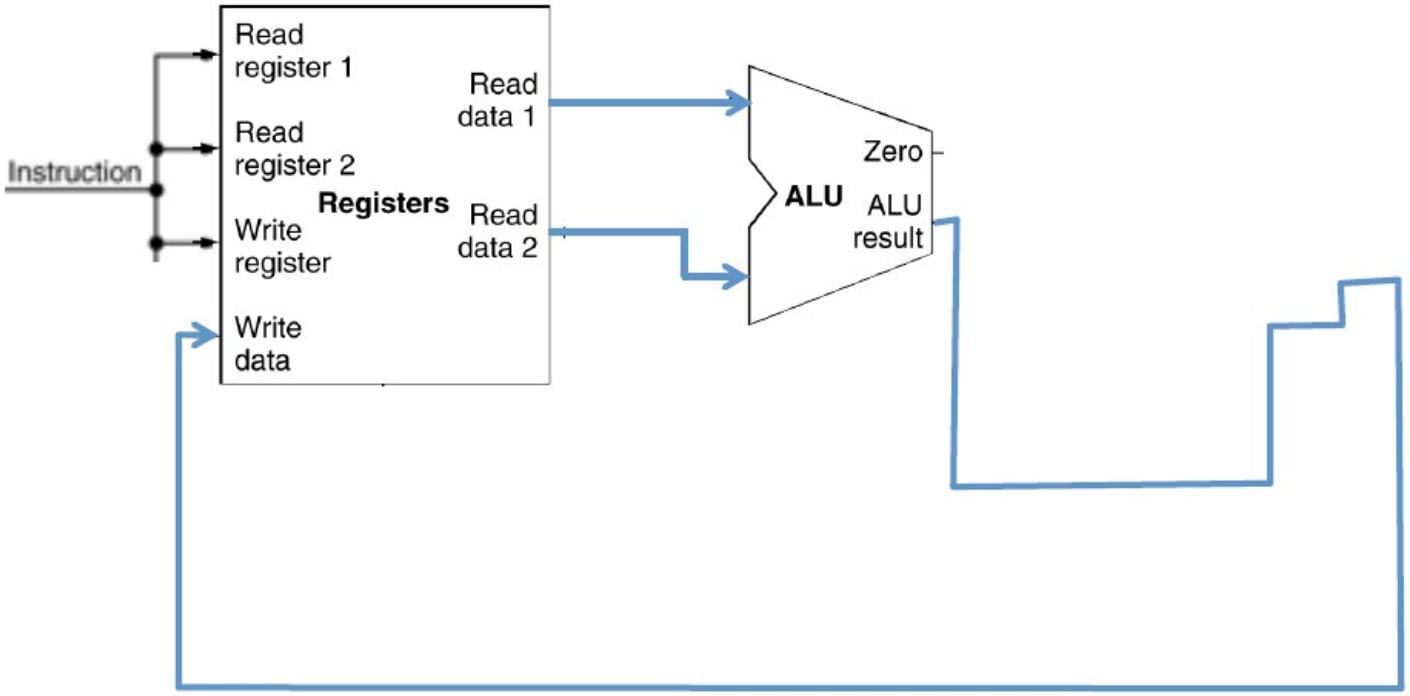


Tính toán địa chỉ rẽ nhánh: không sử dụng ALU cho phép toá so sánh và tính toán địa chỉ cùng một thời điểm.

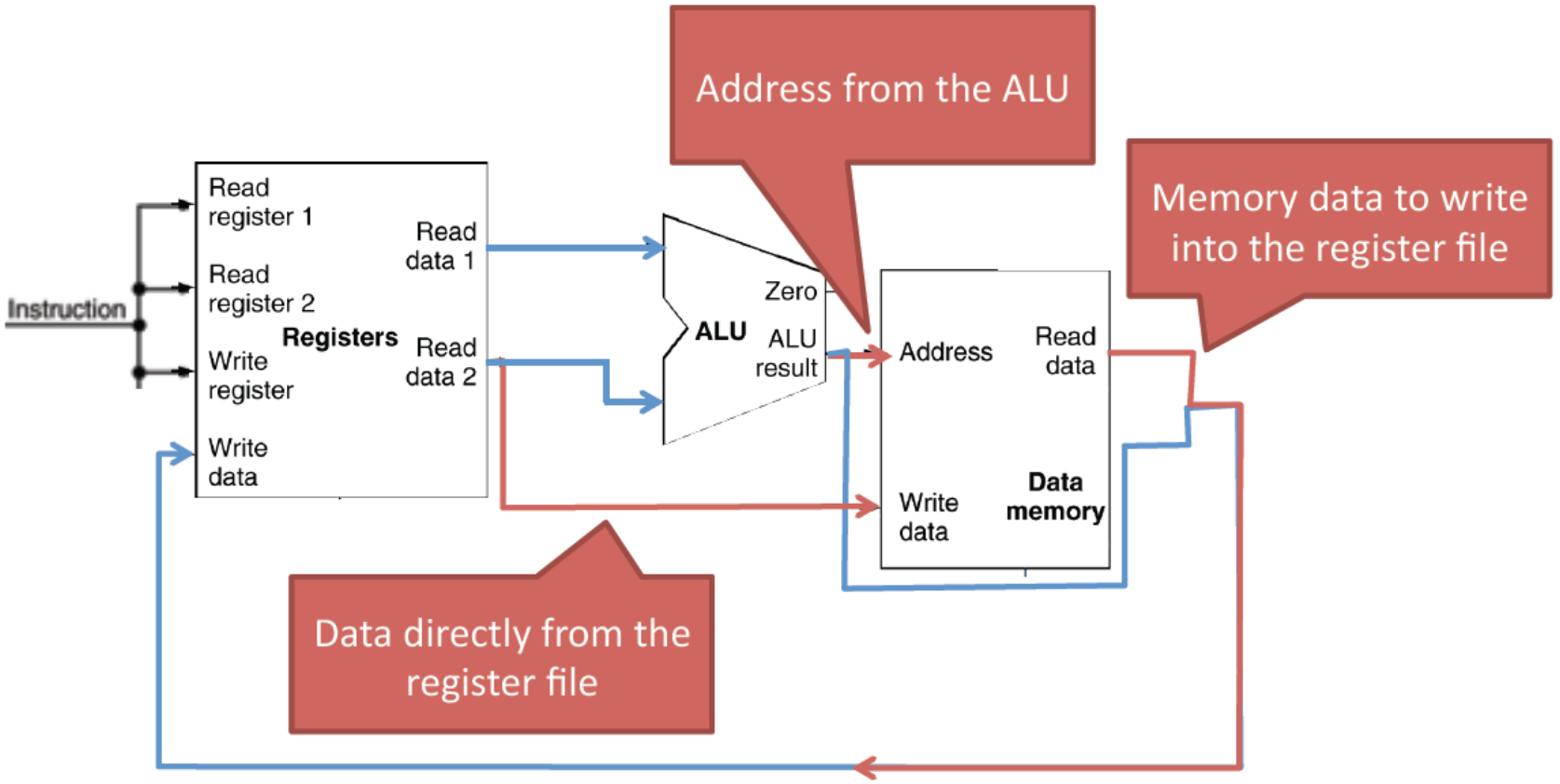
  - **thực thi các nhánh so sánh**
  - **cập nhật lại giá trị của PC**



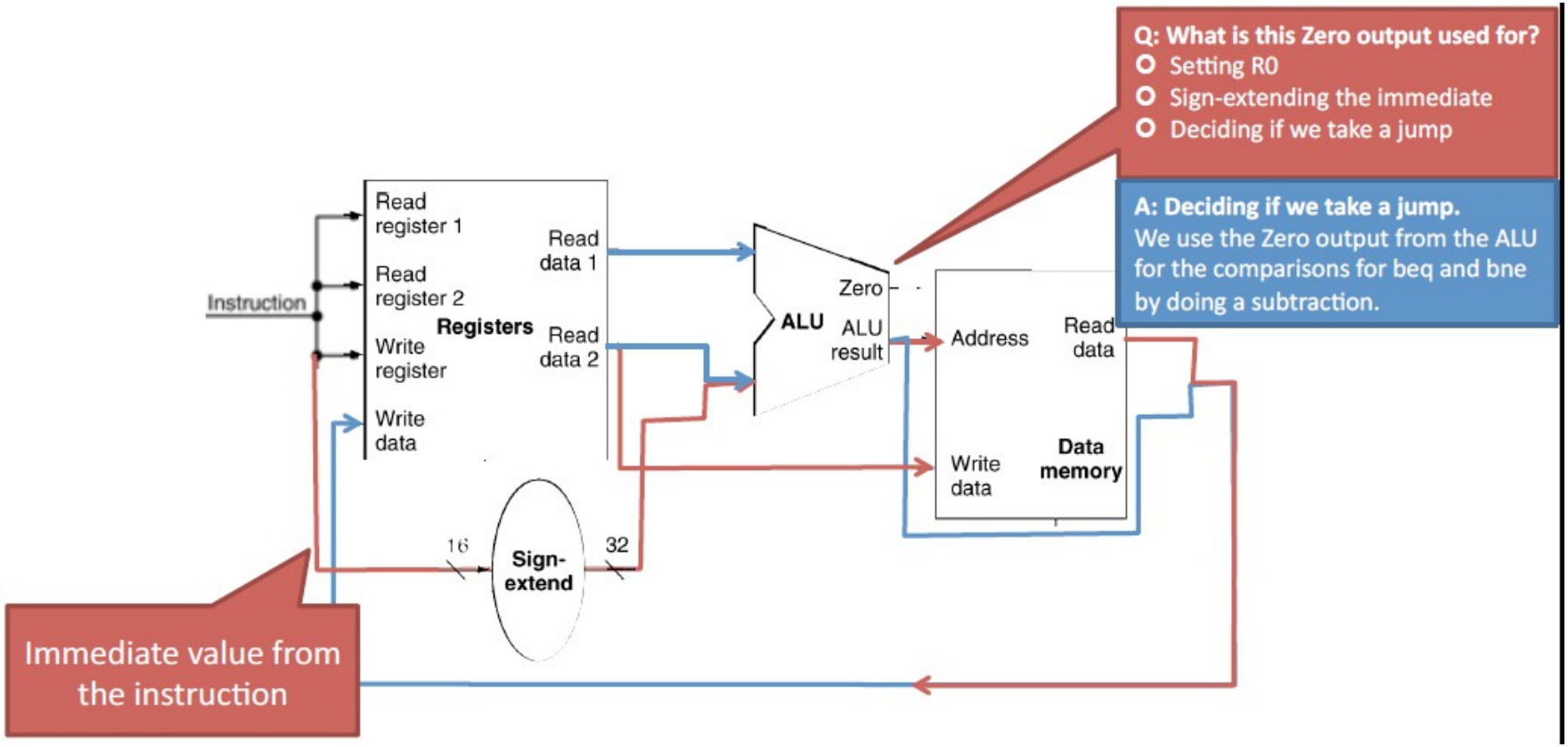
Kết nối RF và ALU



Kết nối ALU với bộ nhớ

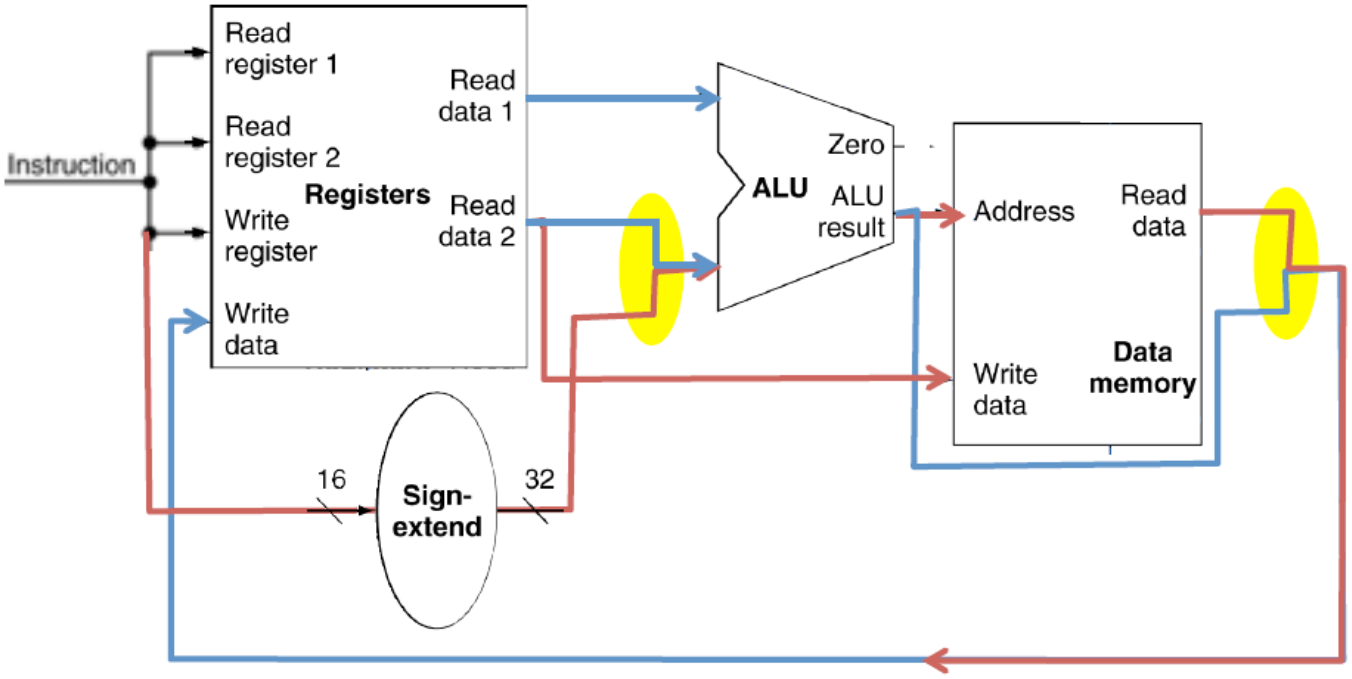


Kết nối giá trị tức thời để tính toán địa chỉ



Thêm vào bộ dồn kênh

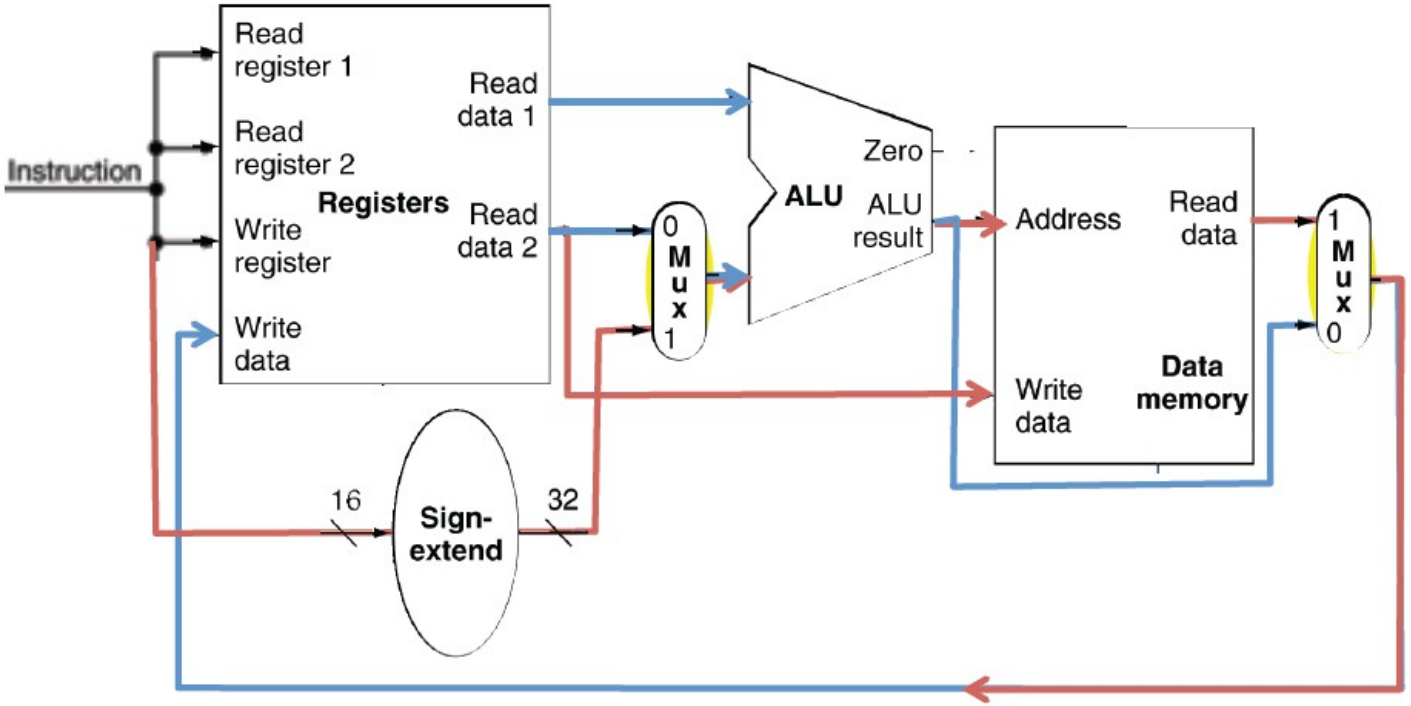
Lựa chọn tín hiệu bằng Bộ dồn kênh MUX



Định tuyến tính hiệu (control)

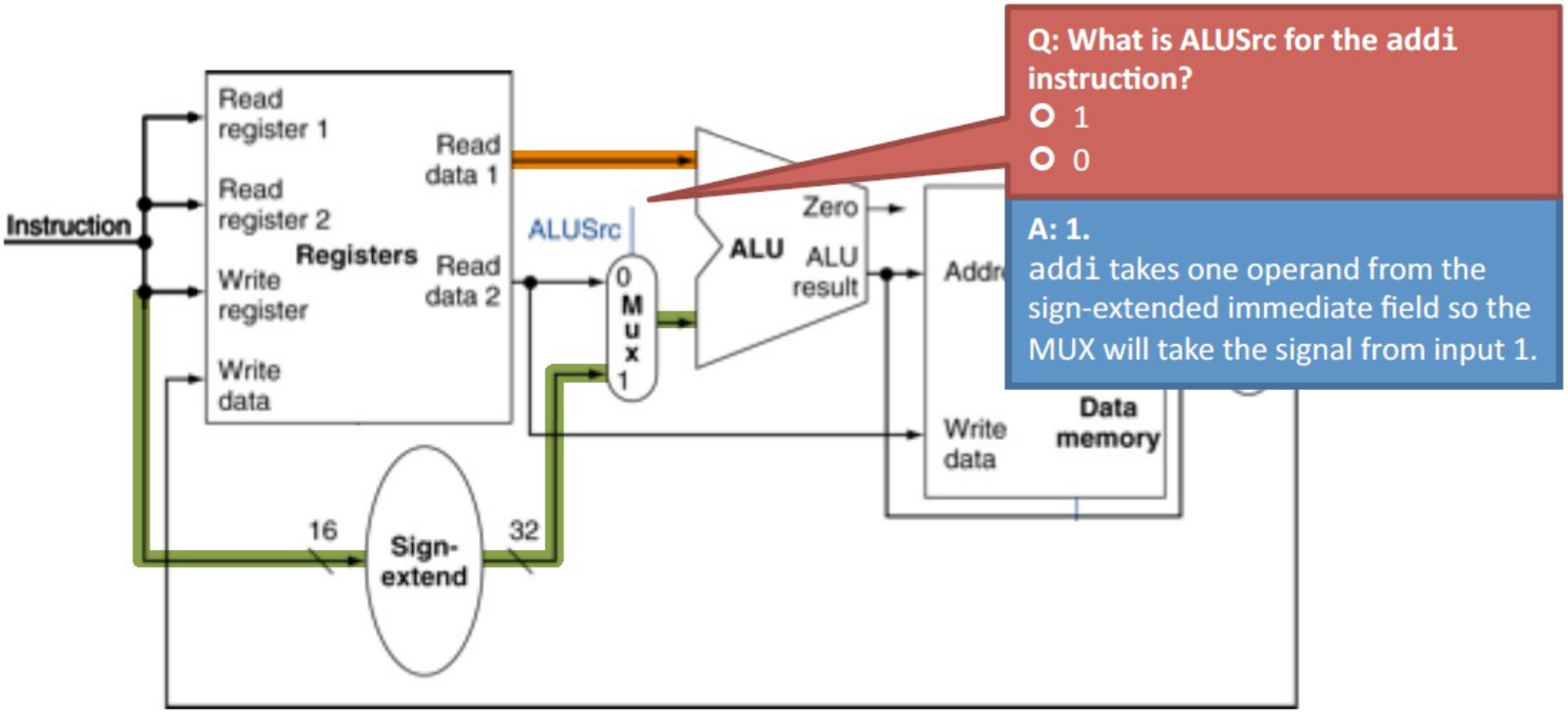
• MUX lựa chọn **ALU input** (tệp thanh ghi hoặc các giá trị hằng số có dấu tức thời)

• MUX lựa chọn tệp thanh ghi ghi dữ liệu (kết quả tính toán được tại ALU hoặc dữ liệu từ bộ nhớ)

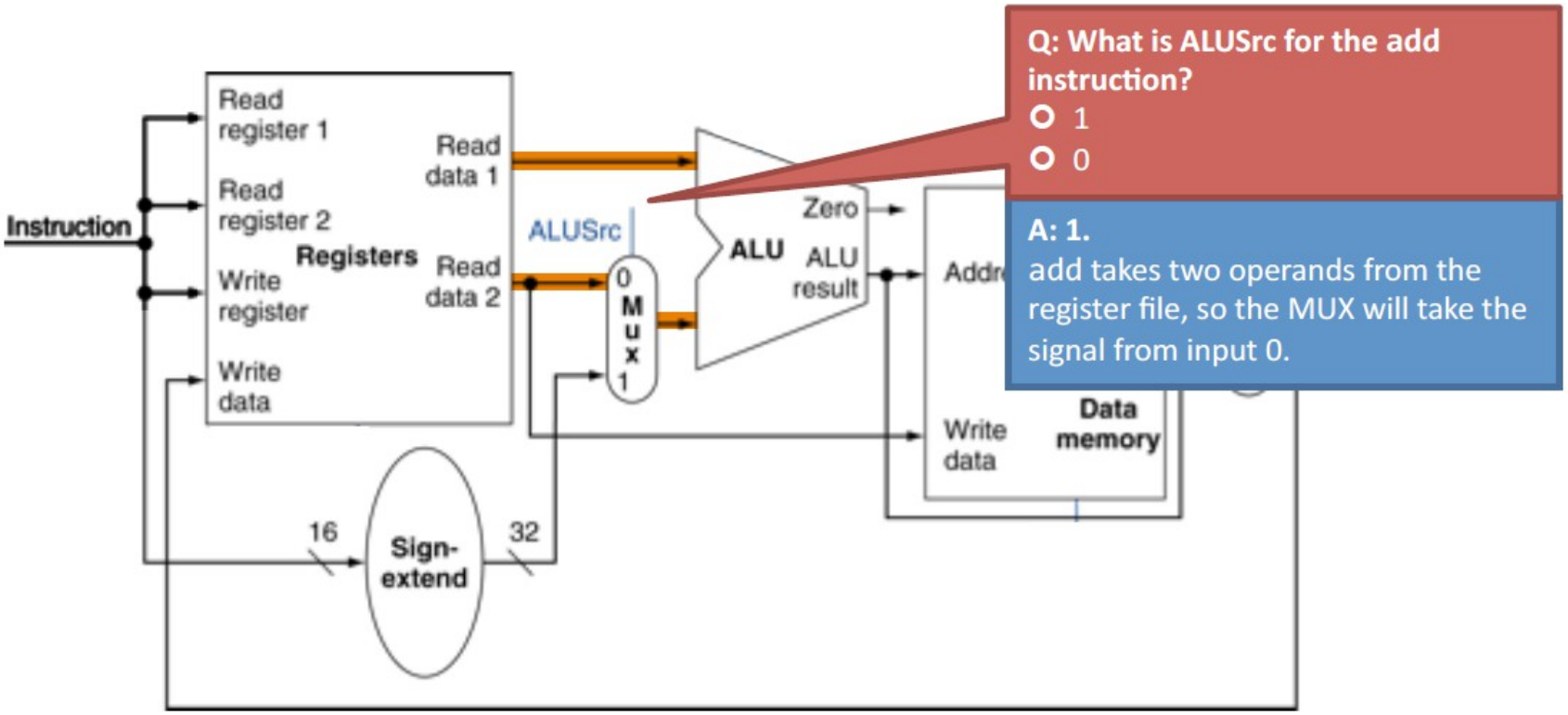


Các tín hiệu điều khiển xác định hoạt động: Lệnh addi

Định nghĩa về tín hiệu điều khiển: **ALUSrc** (ALU source) và **MemtoReg** (Memory to Register File)



Các tín hiệu điều khiển xác định hoạt động: lệnh add

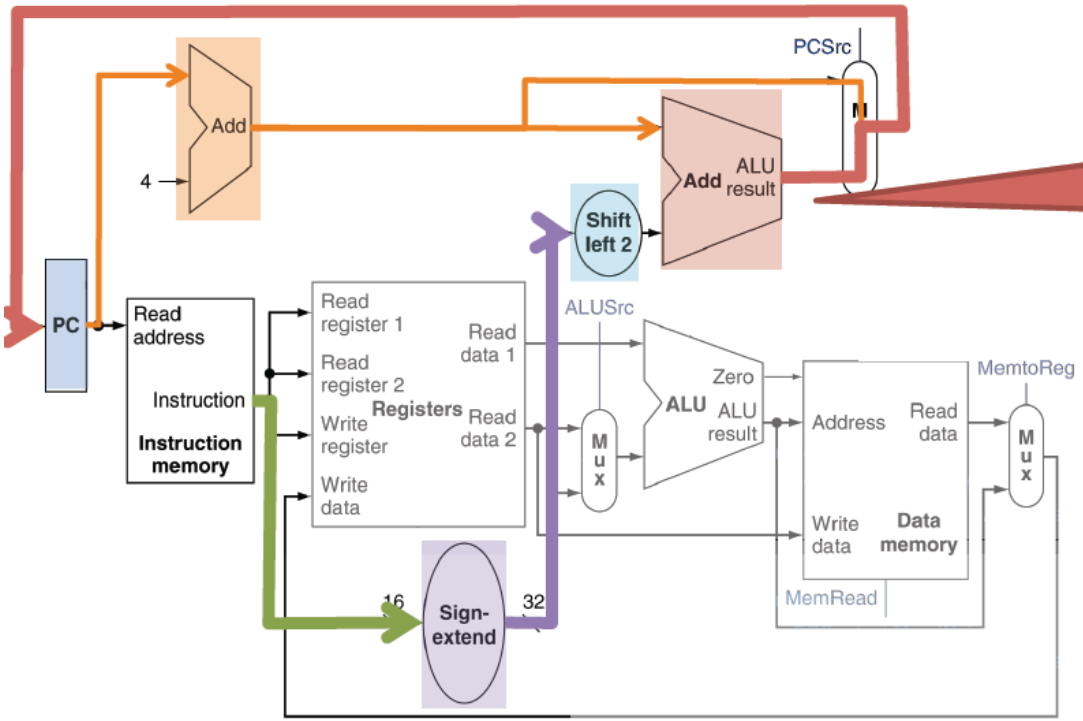


Xây dựng các khối rẽ nhánh có điều kiện

Các bước thực hiện?

– Mặc định: **PC = PC + 4**

– Trường hợp có điều kiện : **PC = PC + 4 + [Sign-extended immediate << 2]** *if* **branch**



Q: Why do we have a second adder for branches?

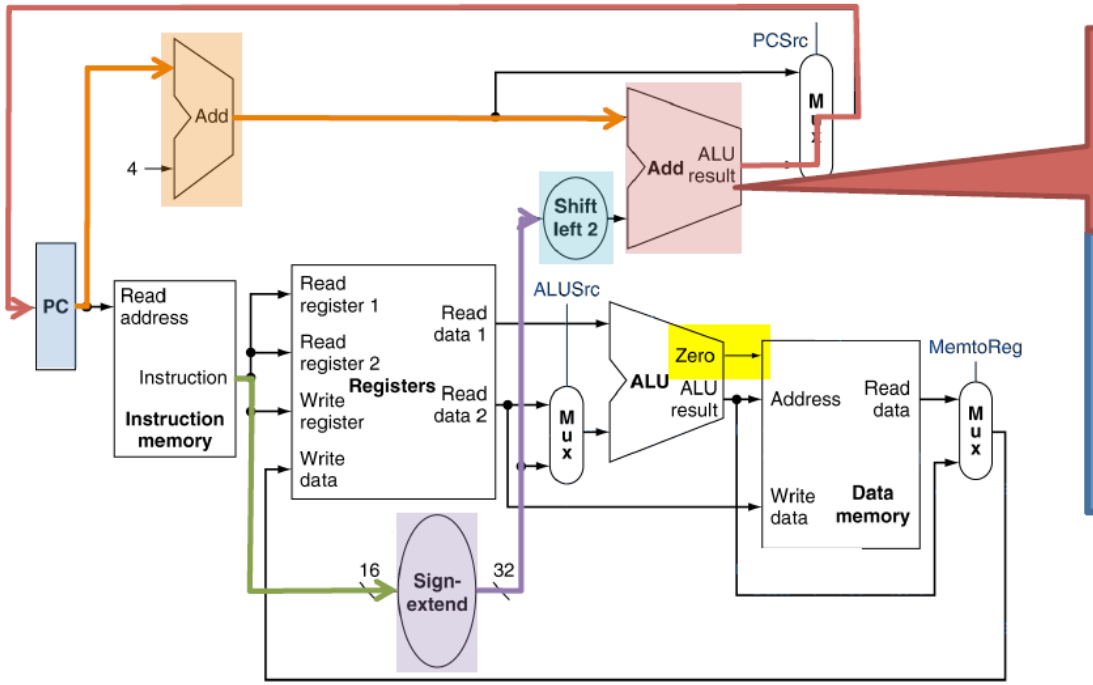
- We need fewer bits
- ALU adder is needed for testing equality
- Special adder needed for immediates

A: The ALU adder is needed for testing equality.

On a branch instruction we need to do a register subtraction to test equal/not equal, which is done on the ALU. We also need to calculate the branch address, so we need a second adder.

Tín hiệu PCSrc

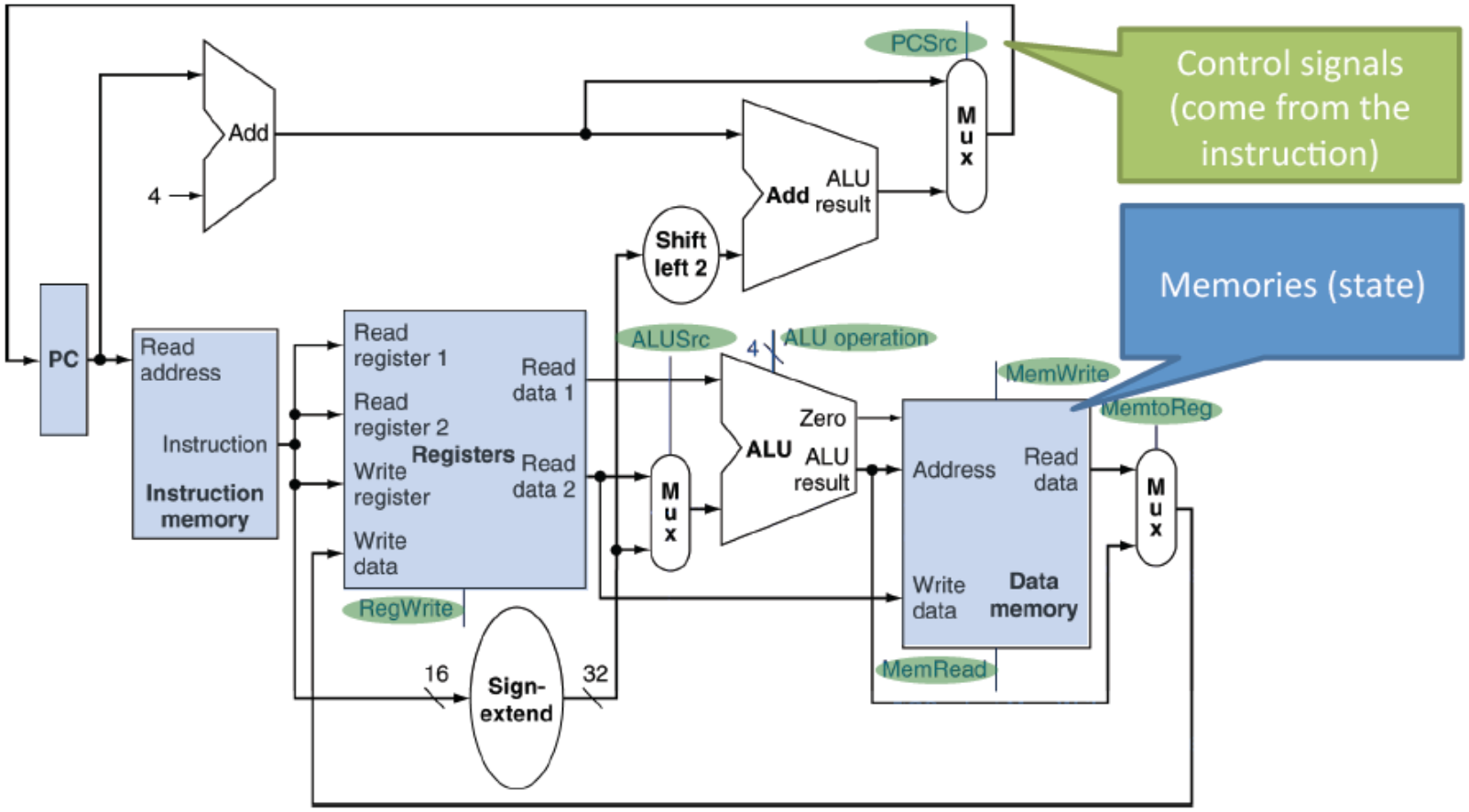
Các bước thực hiện?
– Mặc định: **PC = PC + 4**
– Trường hợp có điều kiện: **PC= PC+4+[Sign-extended immediate << 2]**
if **branch**



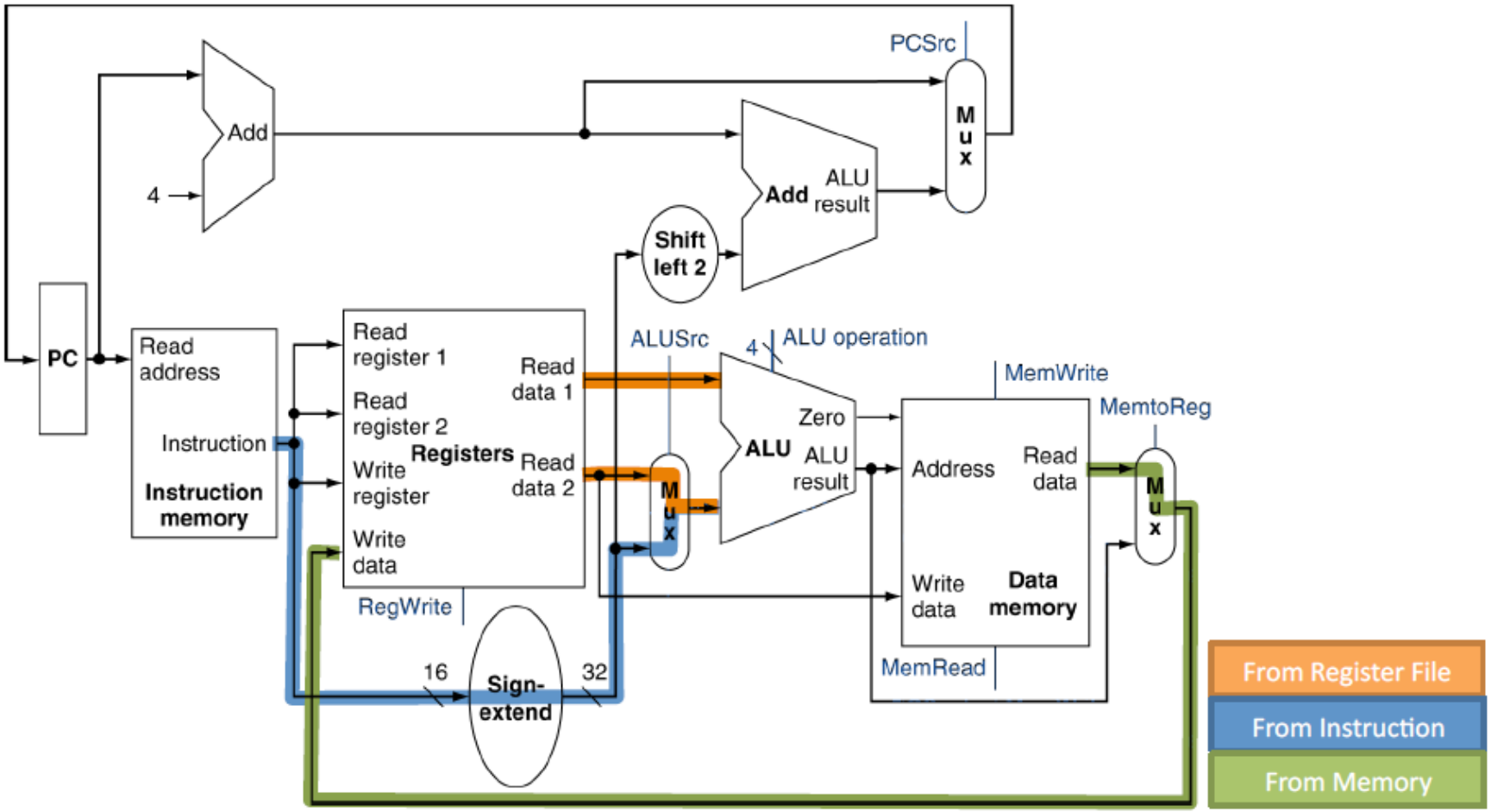
Q: When do we select the ALU result instead of PC+4?
- When we have a beq and ALU Zero is true
- Whenever we have a bne/beq instruction
- Whenever ALU Zero is true

A: When we have a branch AND ALU Zero is true.
If the instruction is a bne/beq AND the output of the ALU Zero is 1 (beq) or 0 (bne) then we branch. Otherwise we either don't have a branch (and so do PC+4) or we don't take the branch (and so do PC+4).

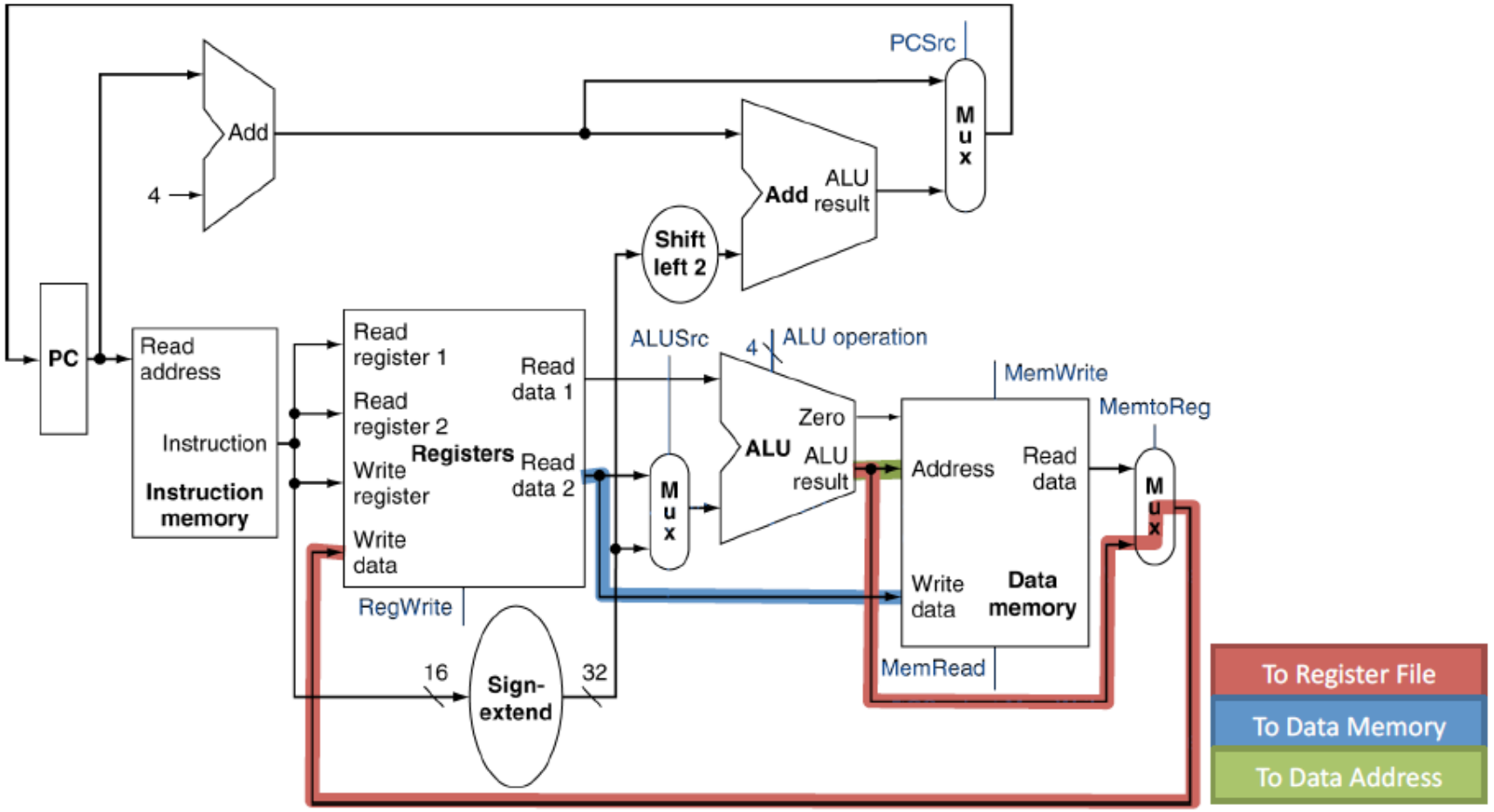
Đường tín hiệu đơn xung nhịp trong MIPS



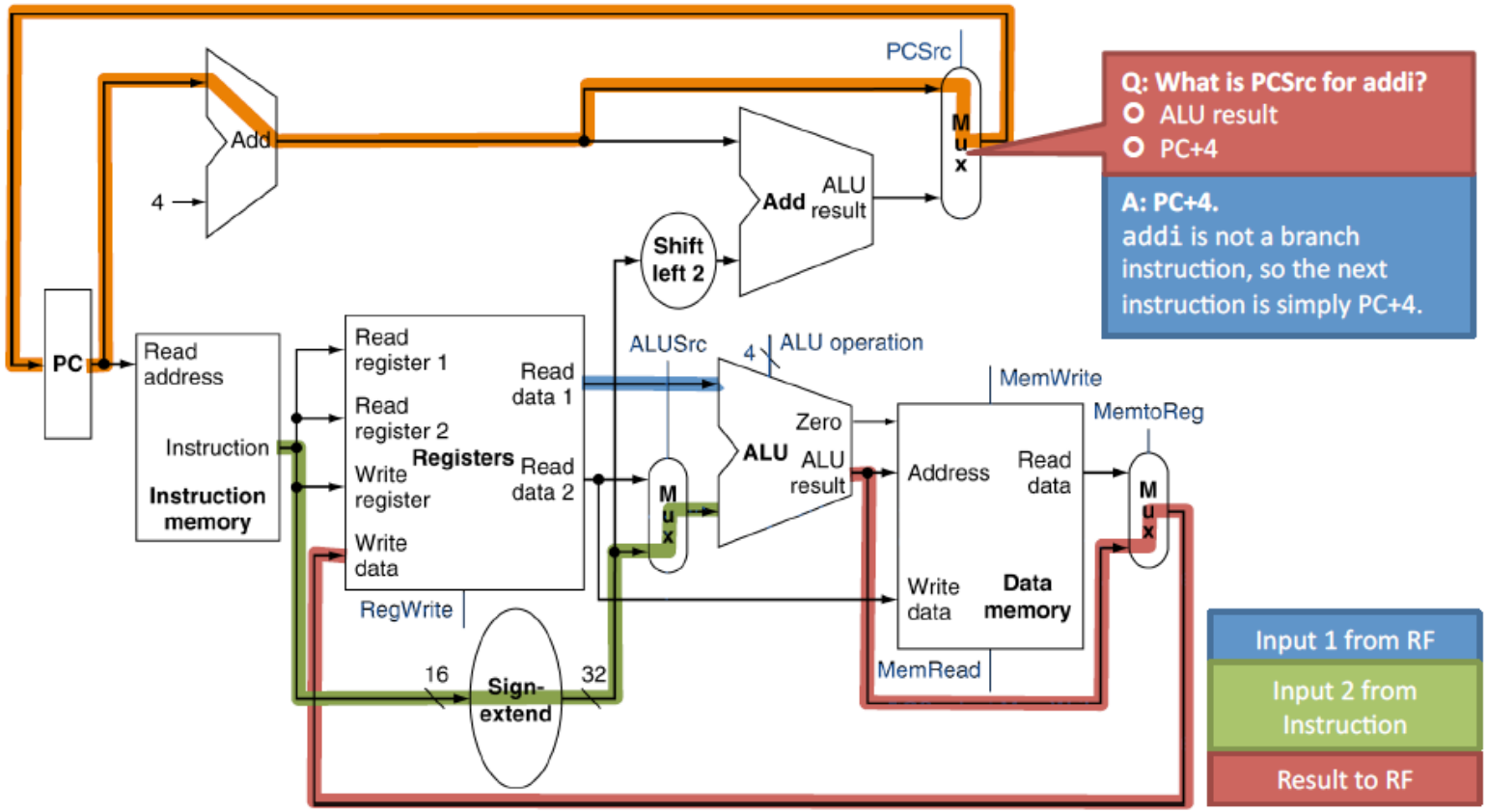
Các dữ liệu nguồn (dữ liệu lấy ra ở đâu?)



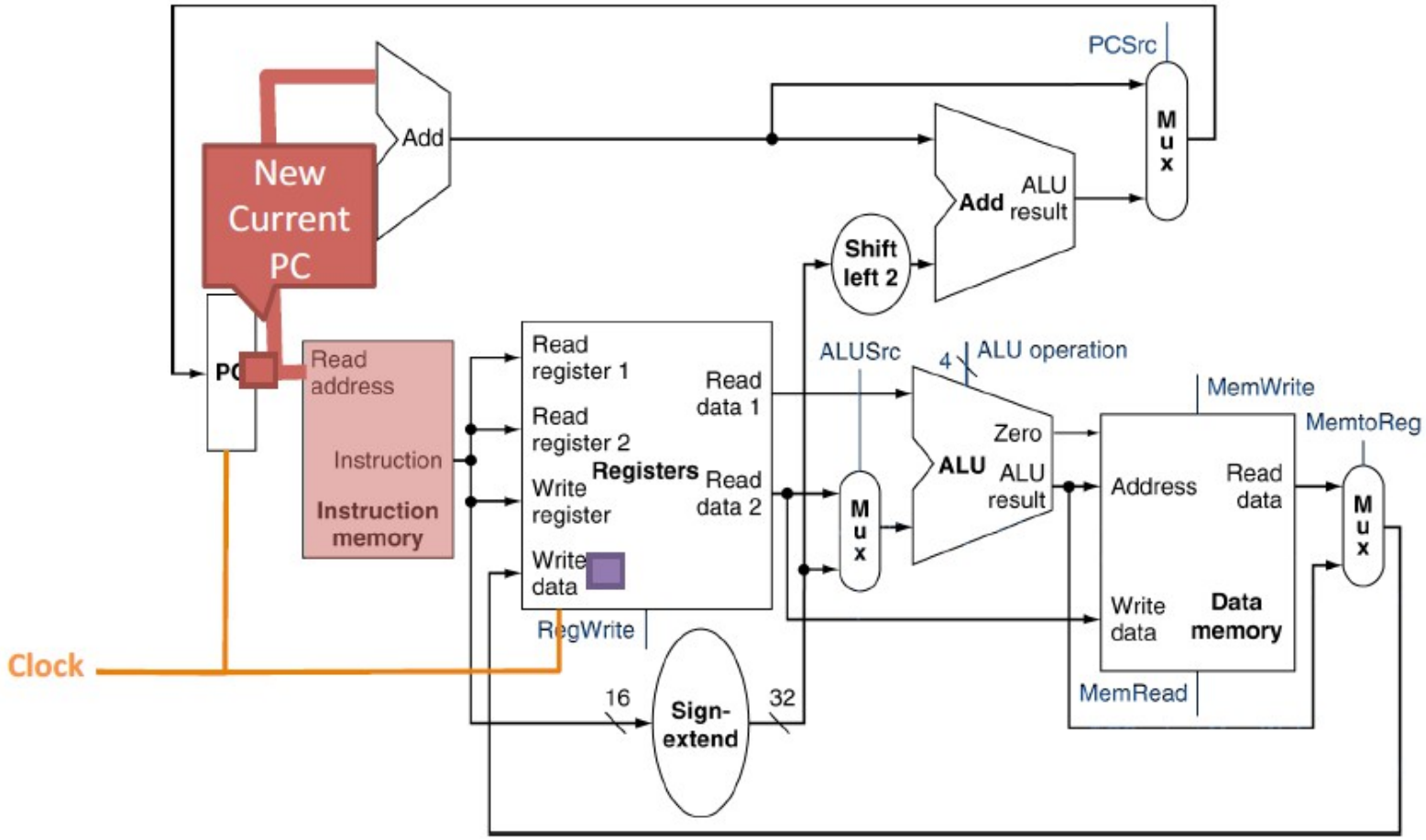
Các dữ liệu đích (dữ liệu đi đến đâu?)



Ví dụ: addi

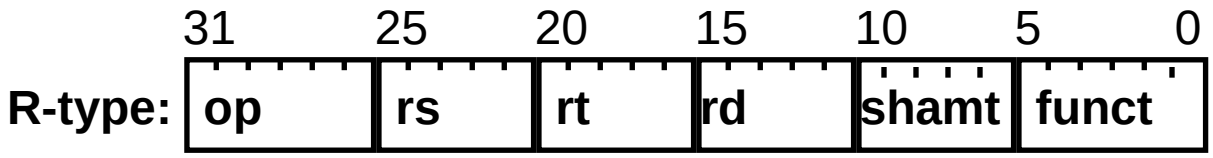

PCSrc
Add
4
Add
ALU result
Shift left 2
M u x
PC
Read address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
RegWrite
ALUSrc
4
ALU operation
ALU
Zero
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
Data memory
MemRead
16
Sign-extend
32
Q: What is PCSrc for addi?
ALU result
PC+4
A: PC+4.
addi is not a branch instruction, so the next instruction is simply PC+4.
Input 1 from RF
Input 2 from Instruction
Result to RF

Lặp lại quá trình…

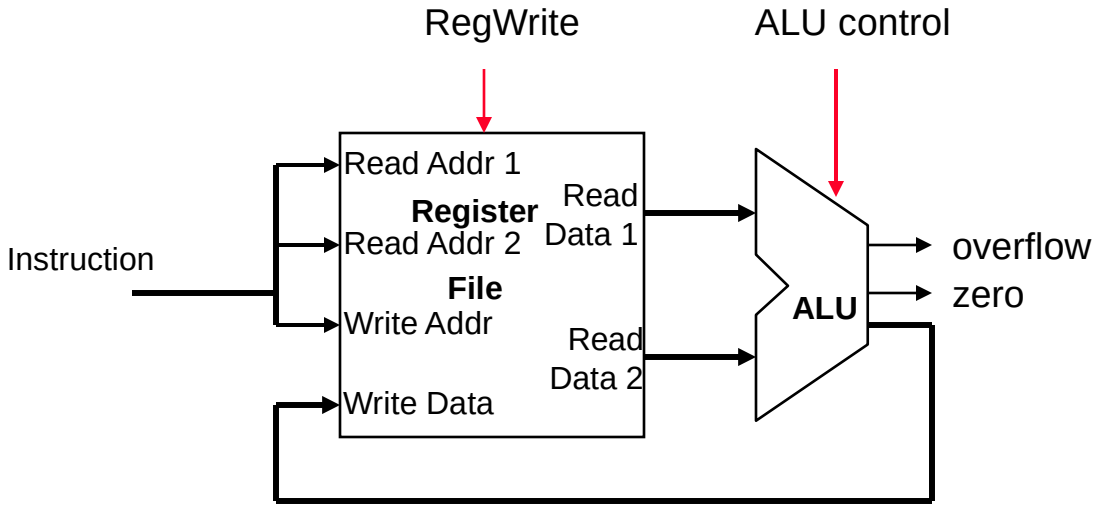


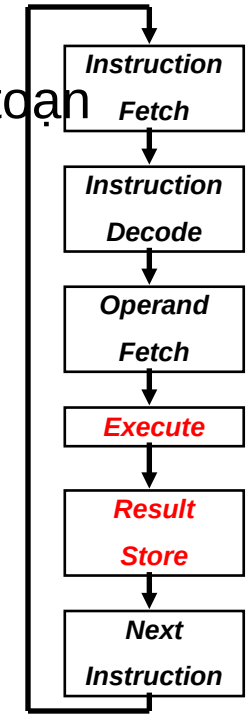
Thực hiện lệnh loại R và ghi kết quả

- Lệnh định dạng R (**`add, sub, slt, and, or`**)



- Thực hiện phép toán (mã hóa bởi op và funct) trên giá trị toạn hạng trong rs và rt
- Ghi kết quả vào tệp thanh ghi (tại vị trí rd)

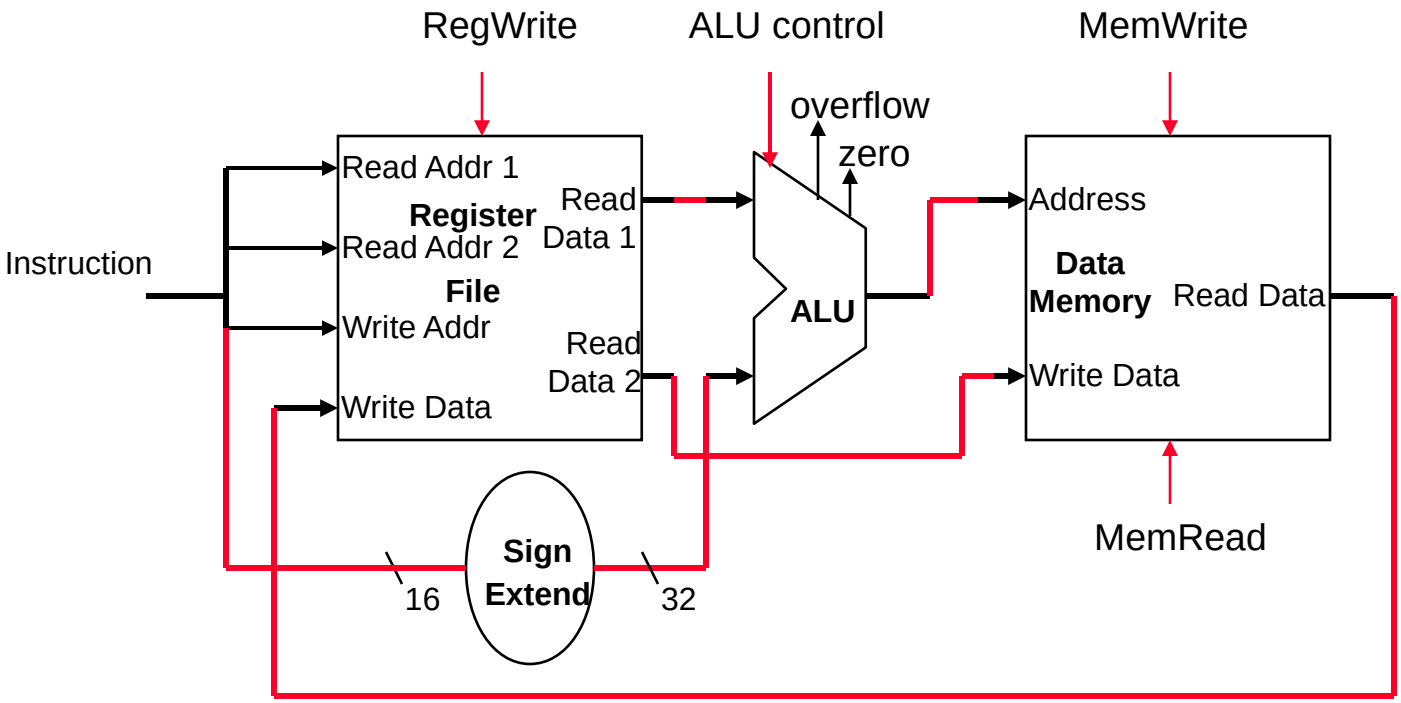


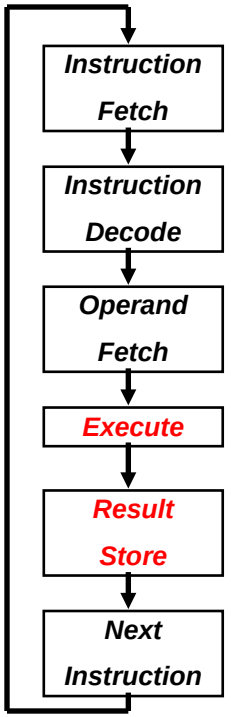


- Tệp thanh ghi không được ghi ở mọi chu kỳ ➔ cần tín hiệu điều khiển ghi riêng biệt.

Đọc ghi bộ nhớ

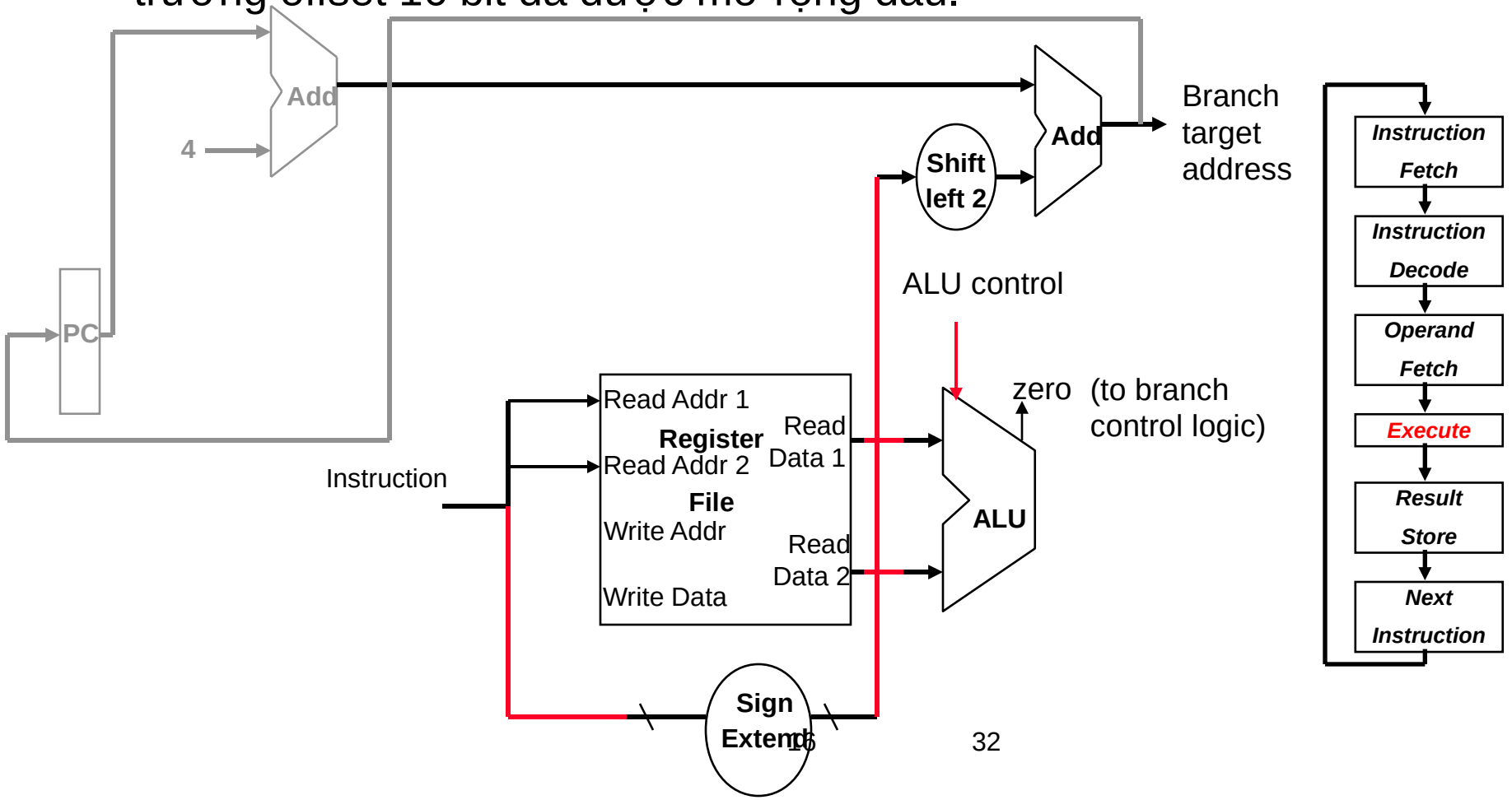
- Địa chỉ bộ nhớ tính ở bước EX: cộng thanh ghi cơ sở (đọc từ tệp thanh ghi khi giải mã lệnh) với giá trị offset
- ghi (sw) giá trị (được đọc từ tệp thanh ghi khi giải mã lệnh) vào bộ nhớ dữ liệu
- đọc (lw) giá trị từ bộ nhớ dữ liệu vào tệp thanh ghi





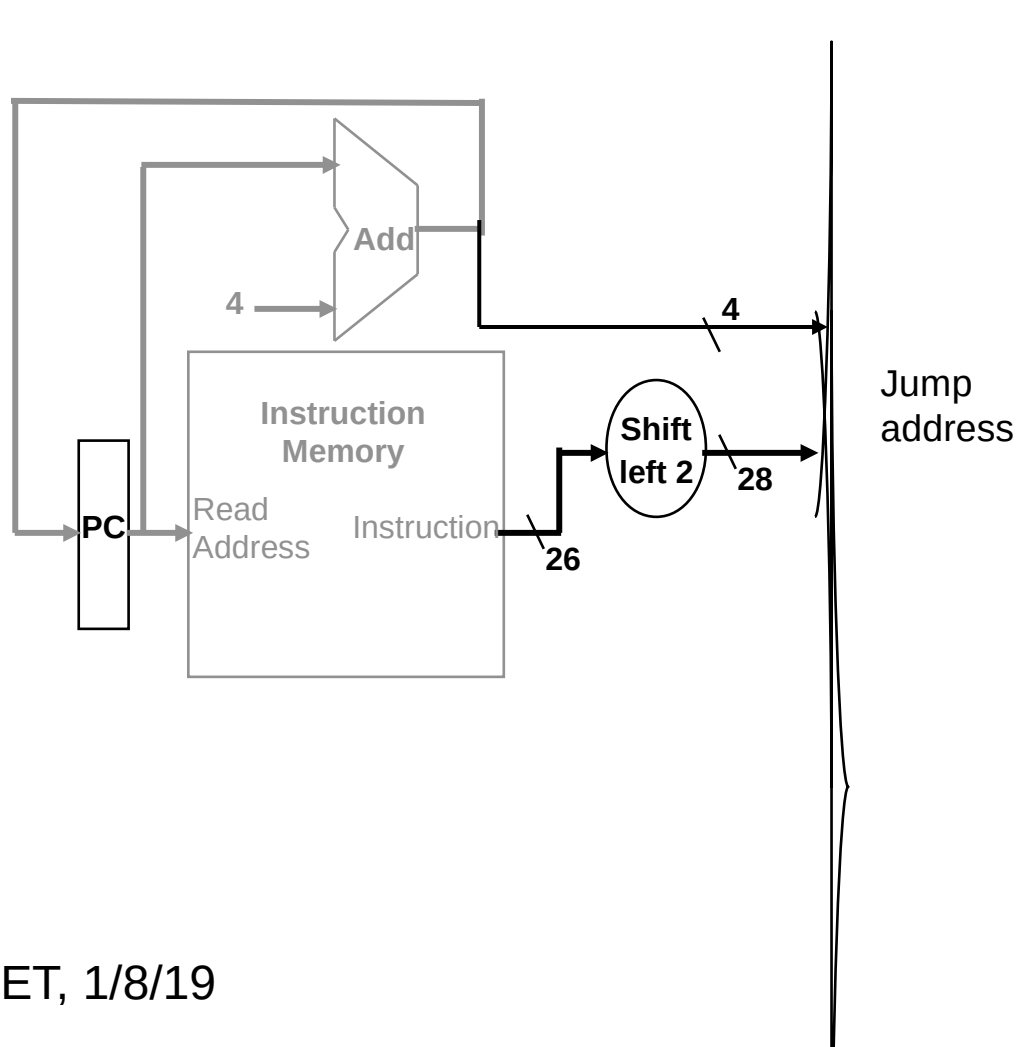
Lệnh rẽ nhánh có điều kiện

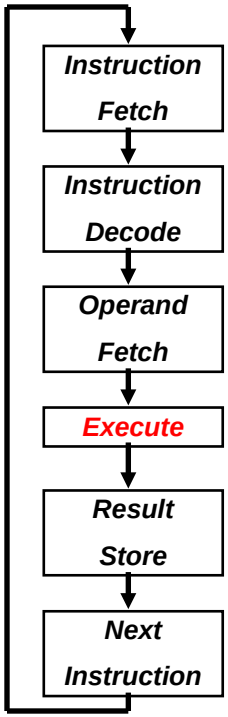
- So sánh toán hạng đọc từ tệp thanh ghi khi giải mã
- Tính địa chỉ đích bằng cách cộng giá trị PC (sau khi cập nhật) với trường offset 16 bit đã được mở rộng dấu.



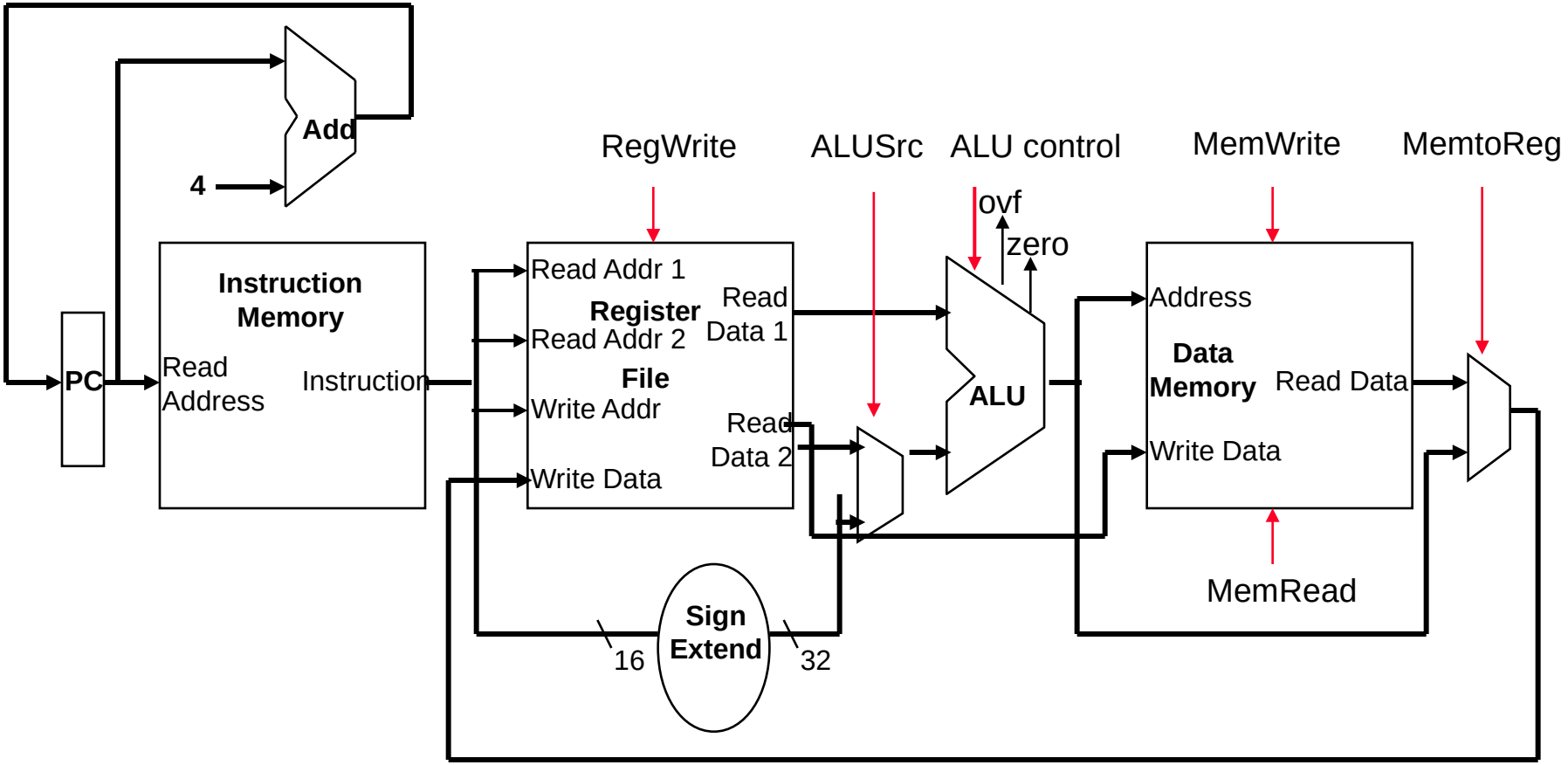
Lệnh nhảy không điều kiện

- Thay 28 bit thấp của PC bằng 26 bít thấp của lệnh được nạp và 2 bít 0

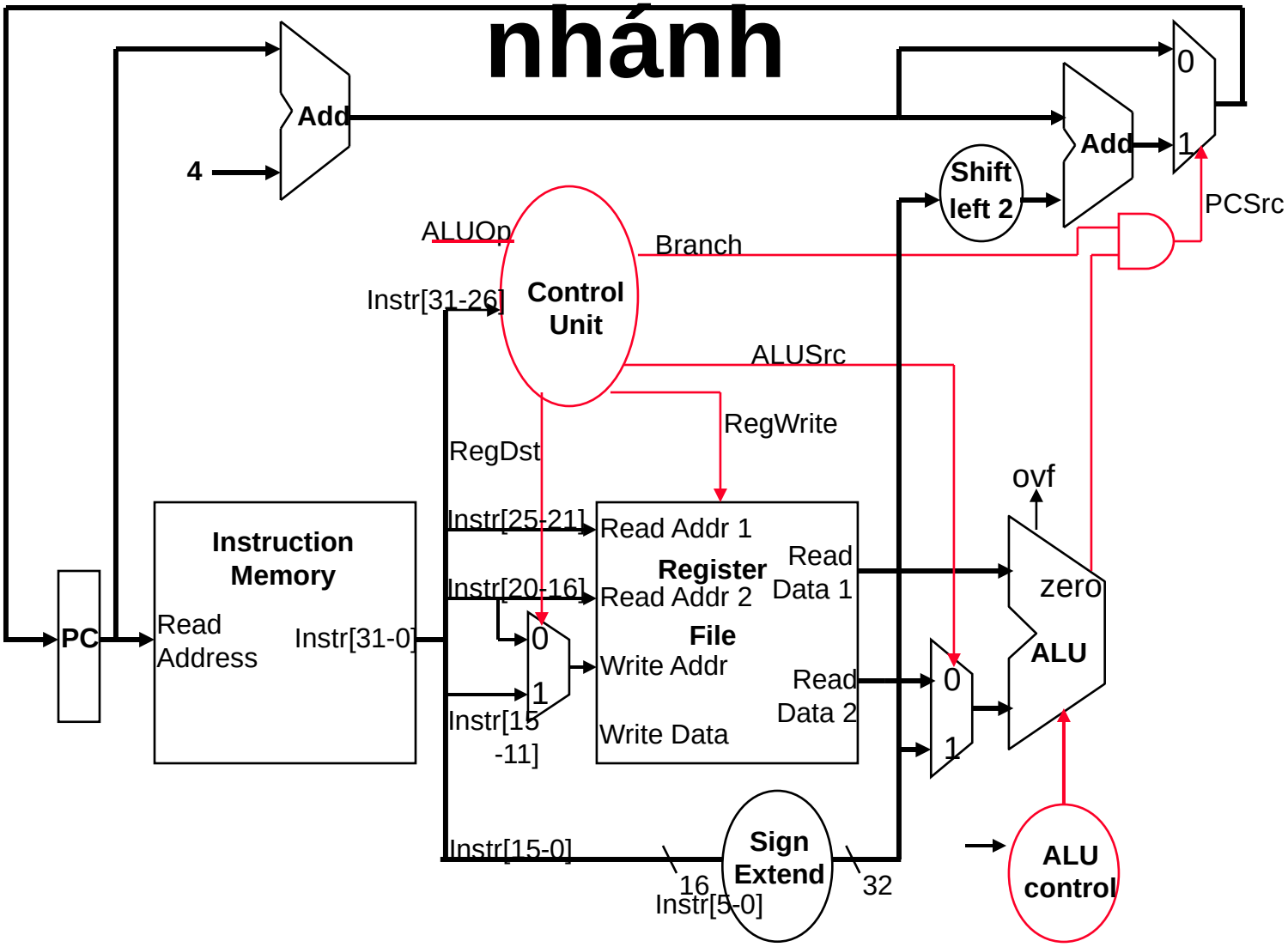




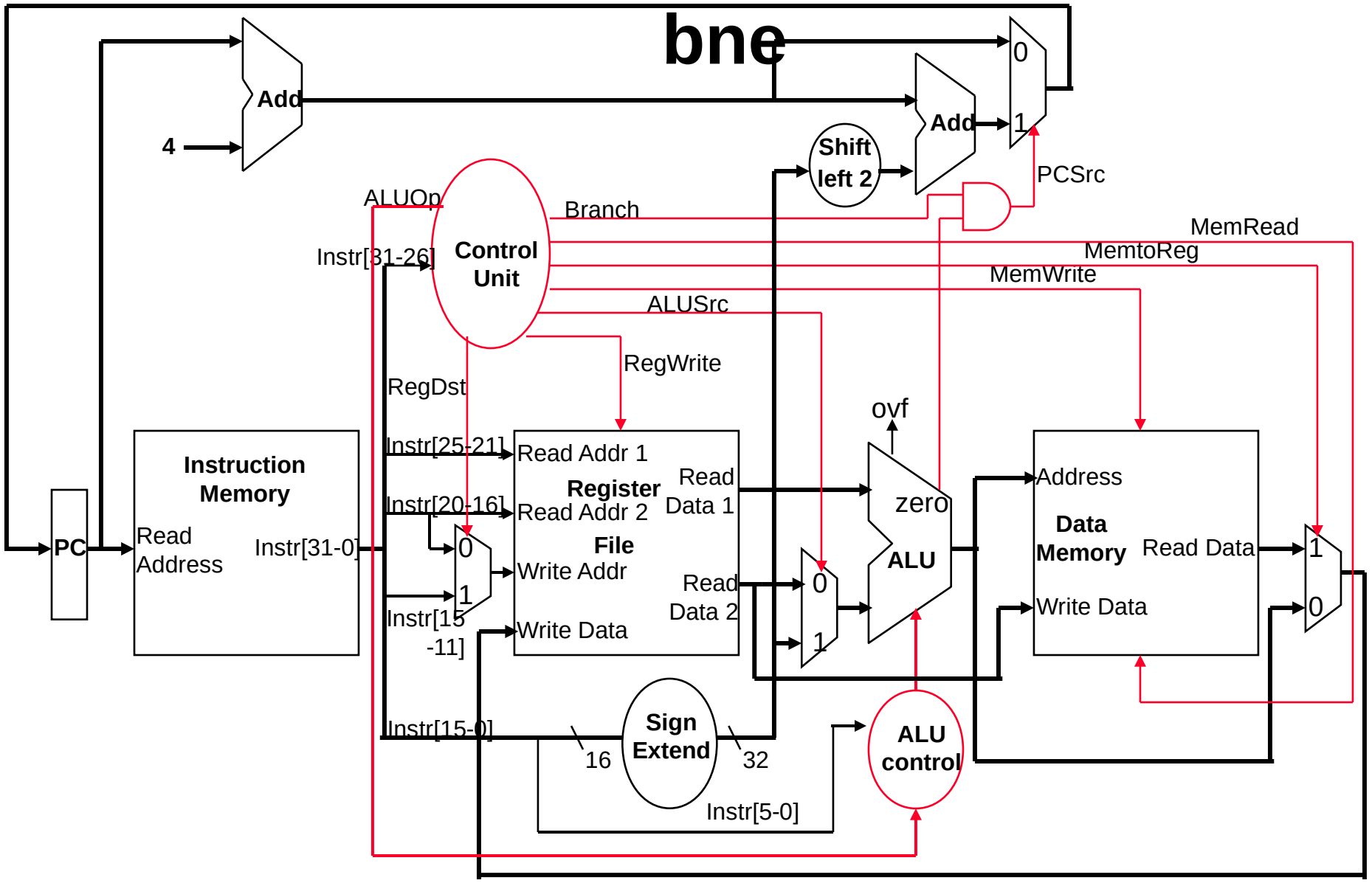
Đường dữ liệu: Lệnh R, I, lw,sw



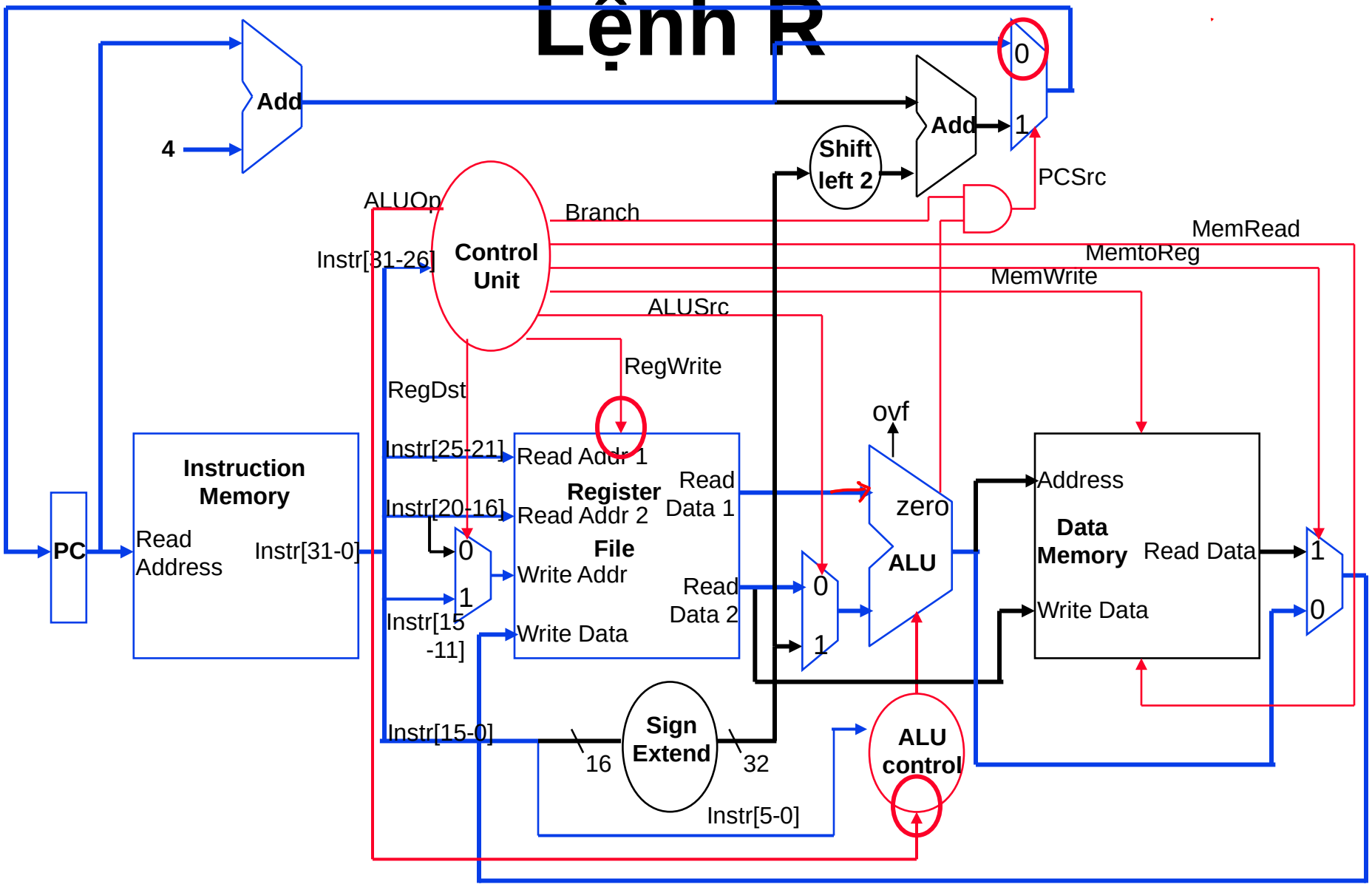
Đường dữ liệu: Lệnh rẽ nhánh



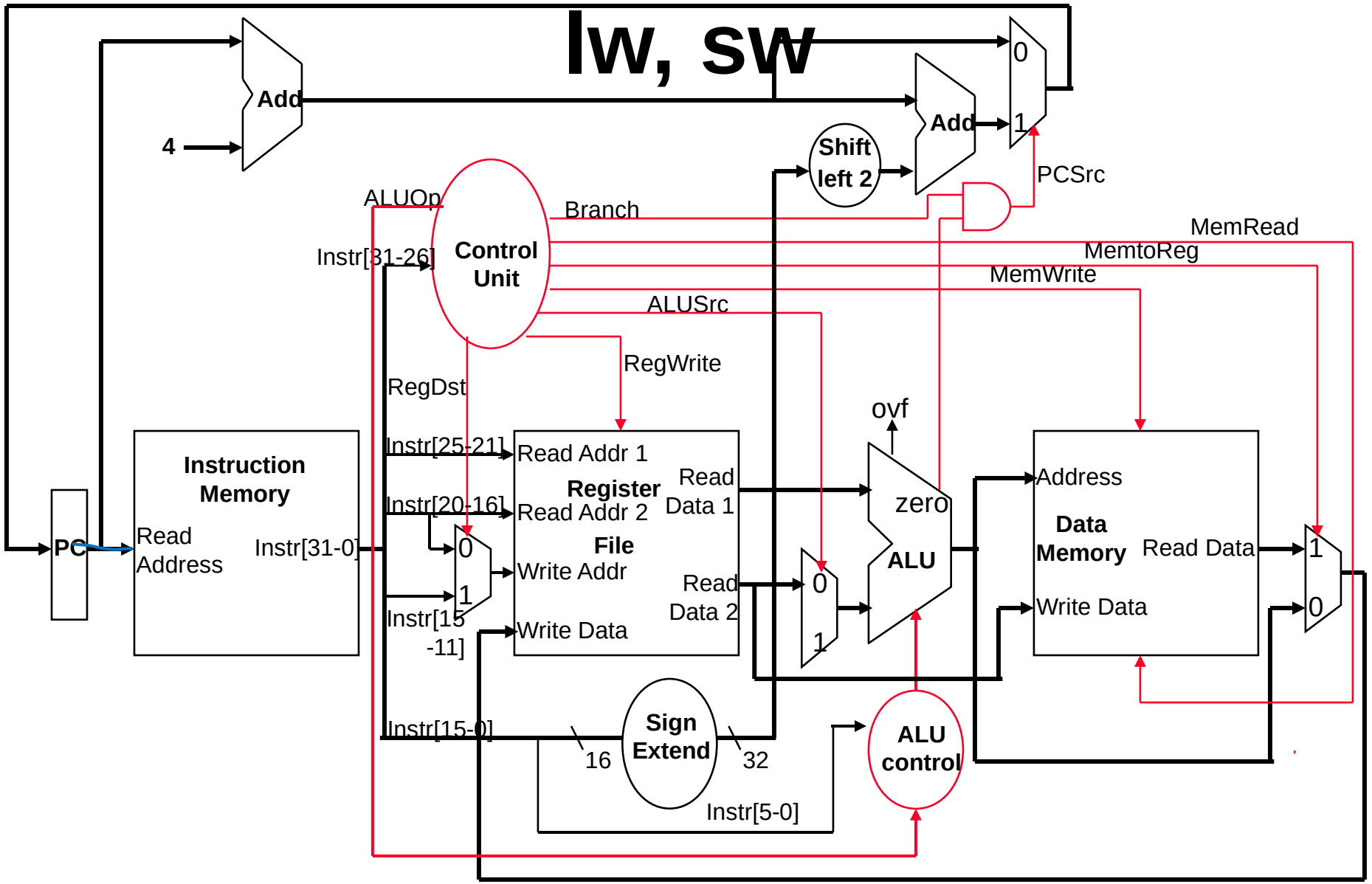
Đường dữ liệu: Lệnh R,I, lw, sw, beq, bne



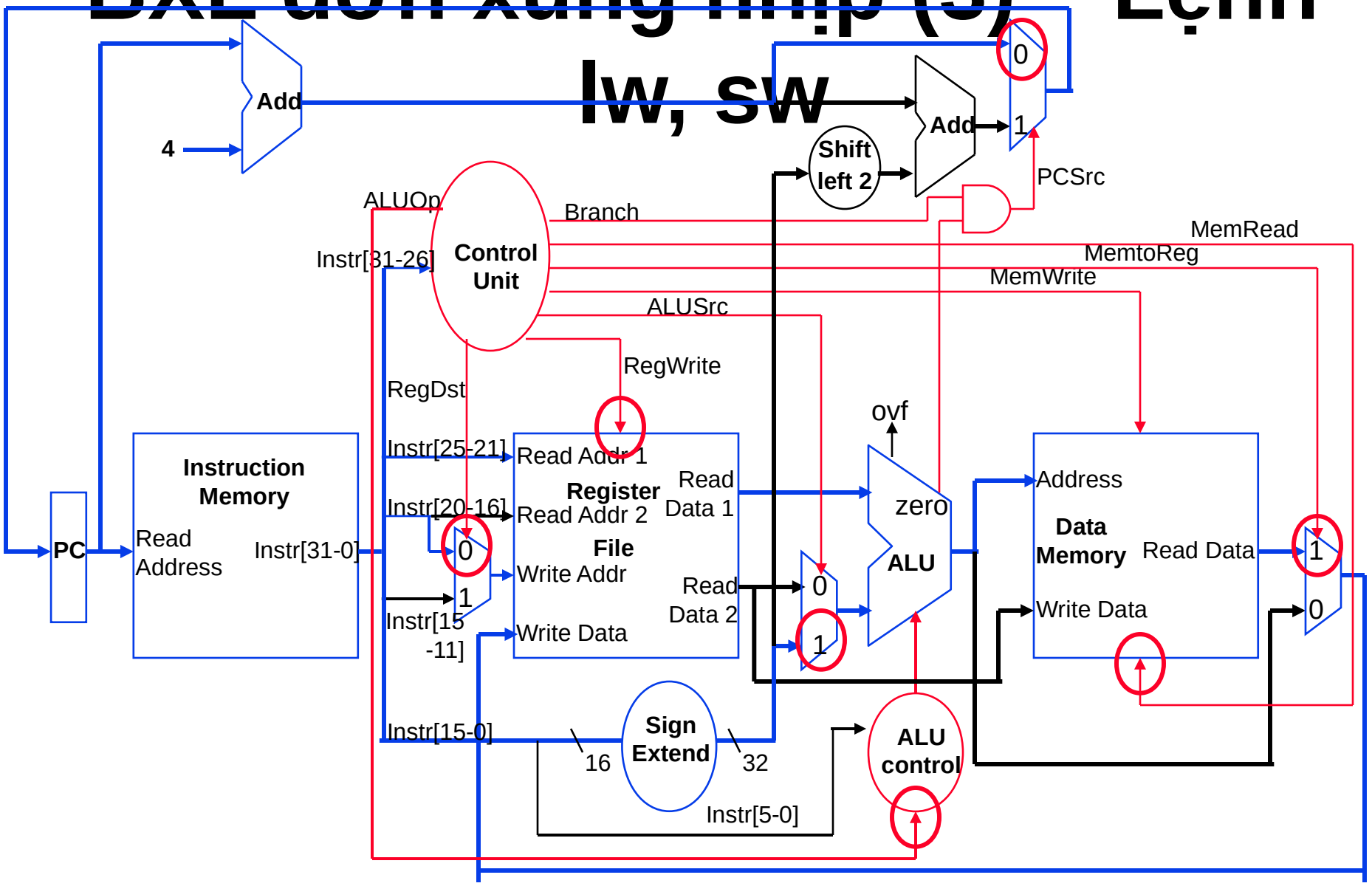
Bộ xử lý đơn xung nhịp (2) – Lệnh R



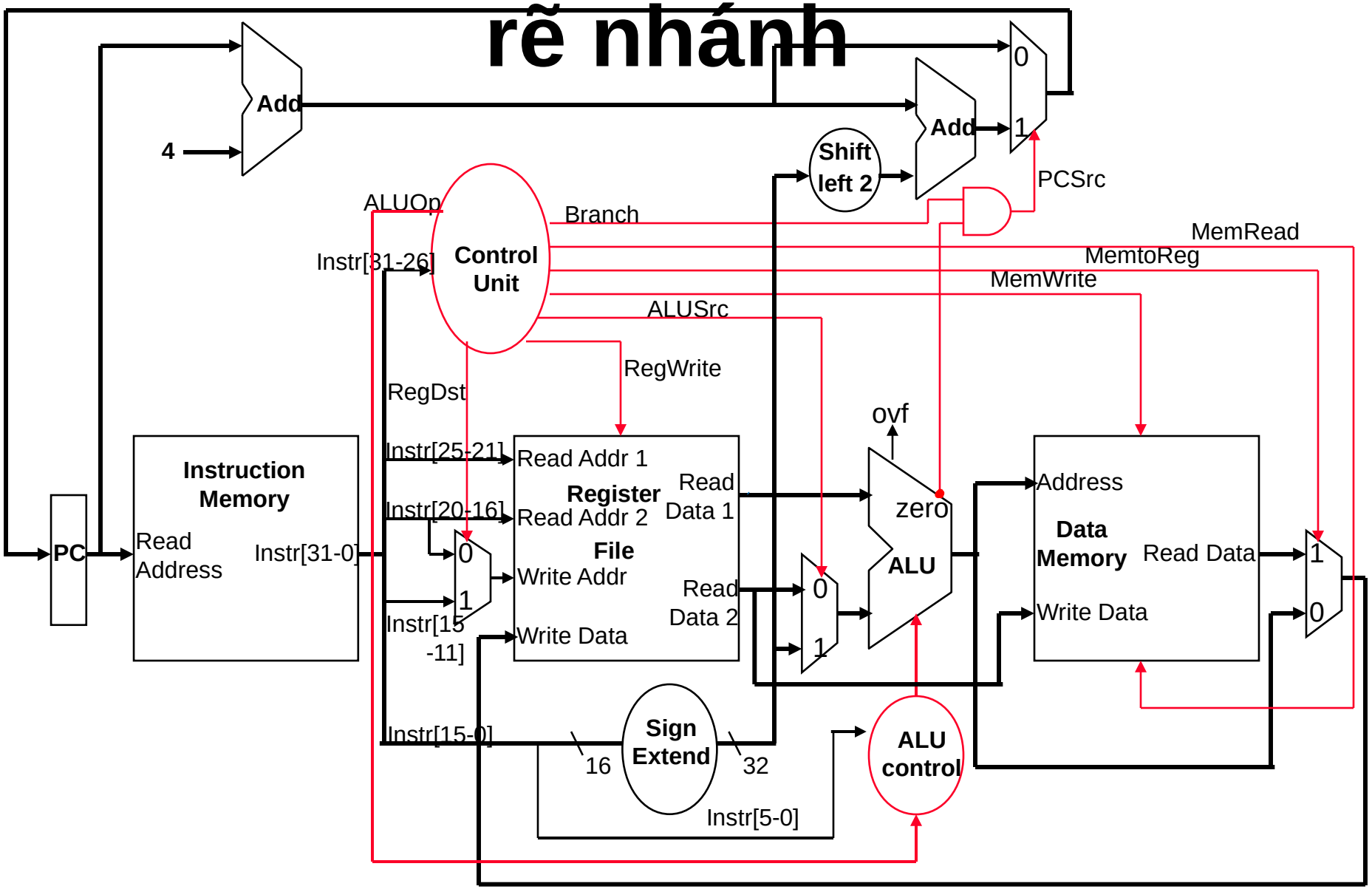
BXL đơn xung nhịp (3) – Lệnh lw, sw



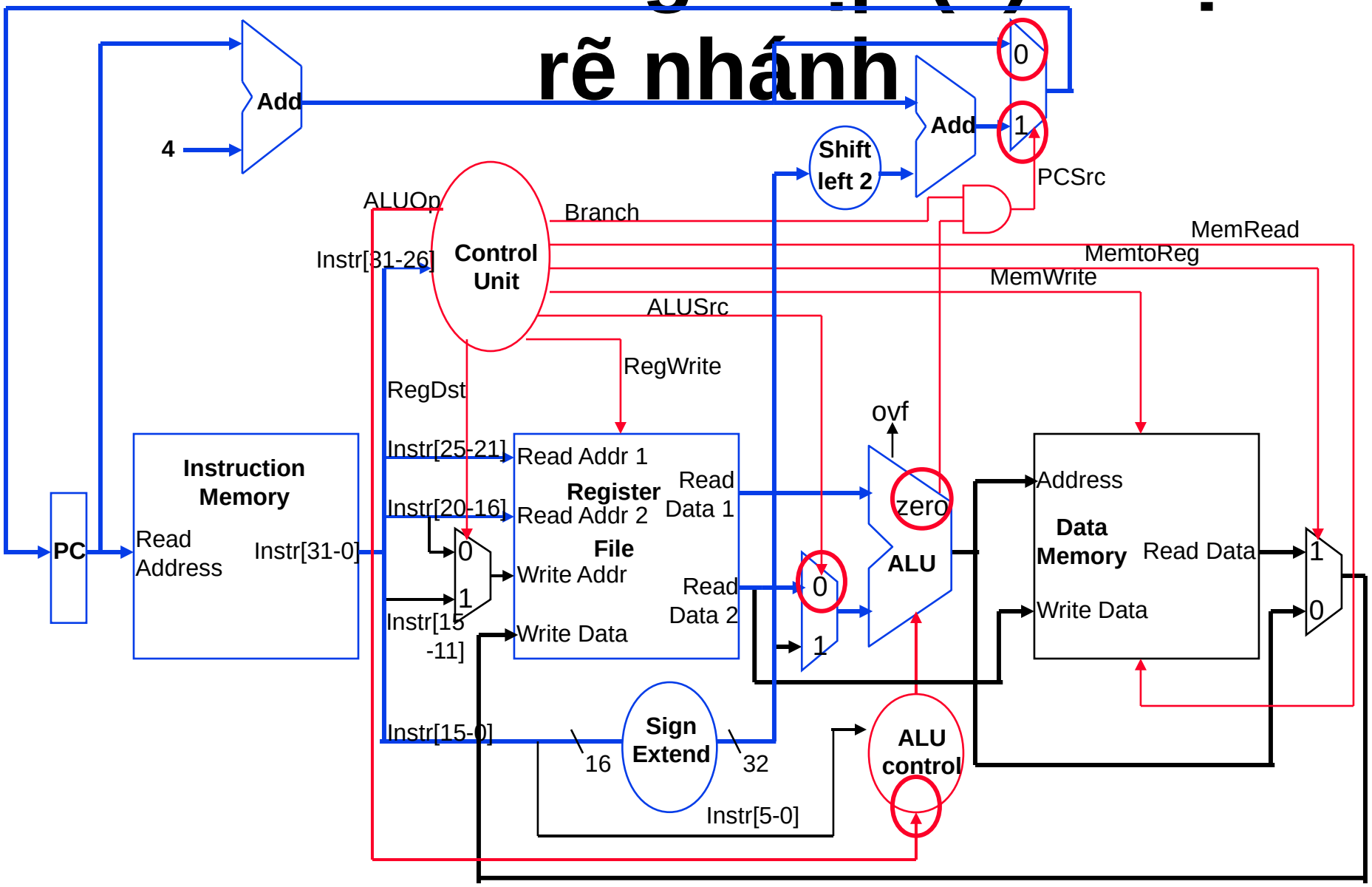
BXL đơn xung nhịp (3) – Lệnh lw, sw



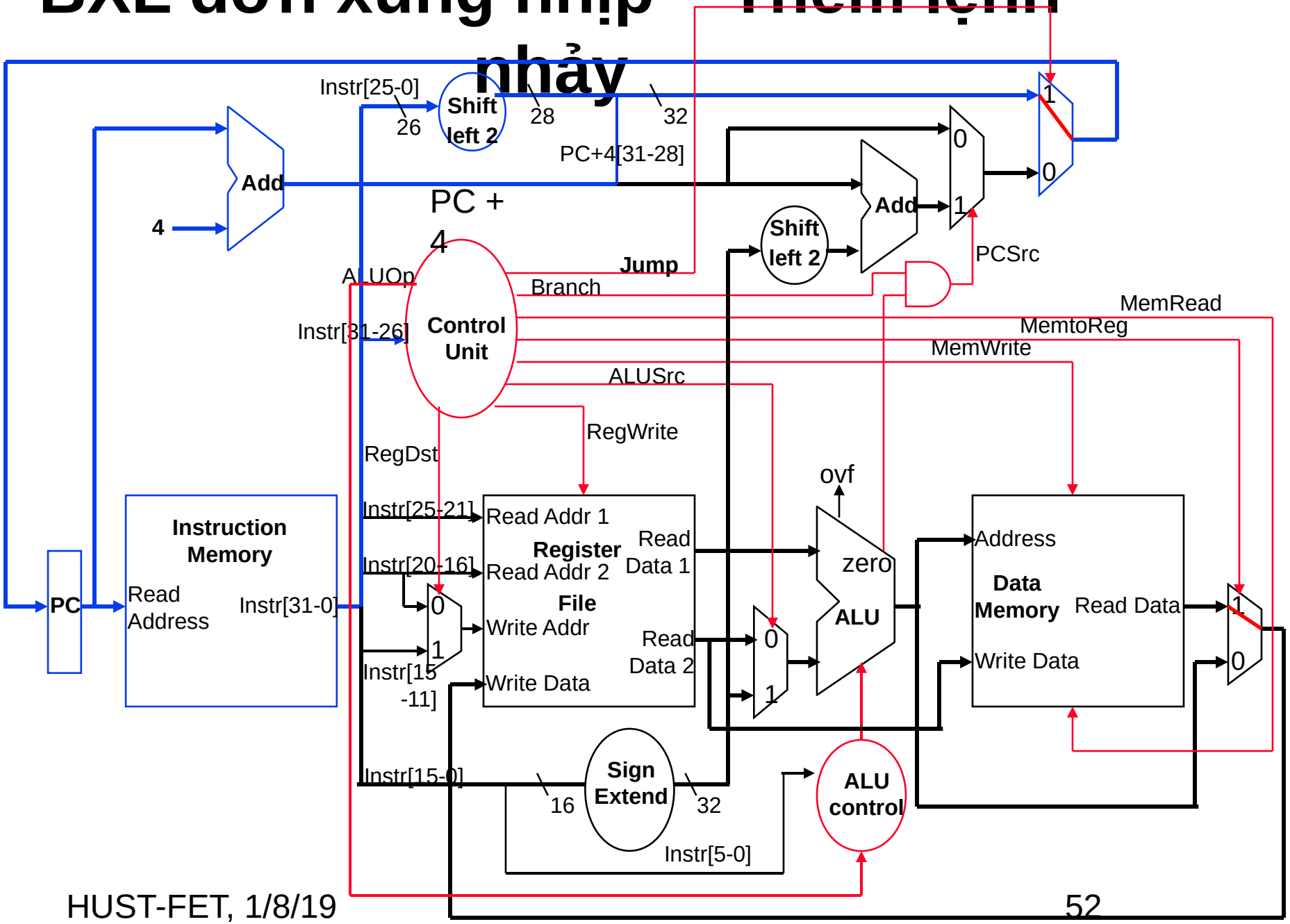
BXL đơn xung nhịp (4) – Lệnh rẽ nhánh



BXL đơn xung nhịp (4) – Lệnh rẽ nhánh

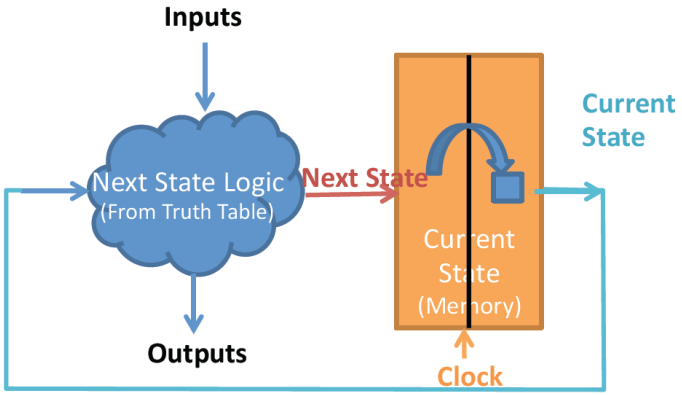


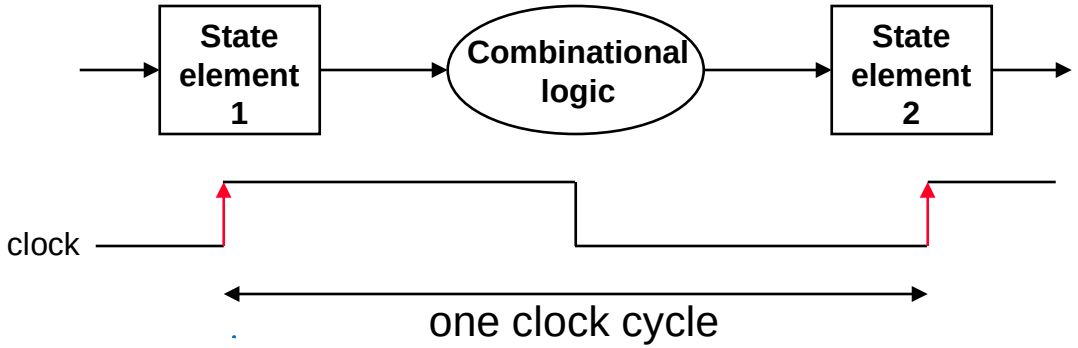
BXL đơn xung nhịp – Thêm lệnh nhảy



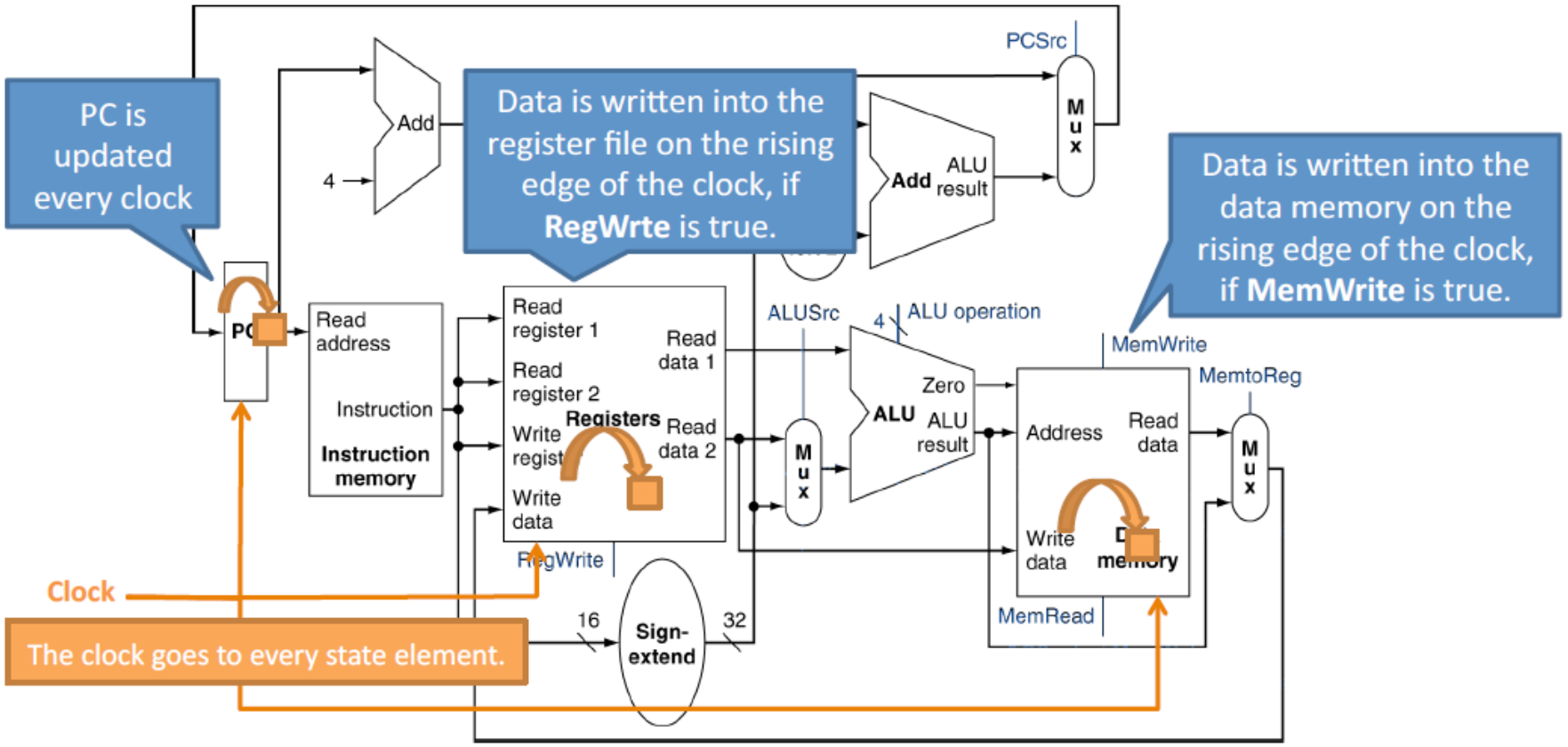
Thiết kế đồng bộ theo đồng hồ

- Mạch đồng bộ theo đồng hồ: 1 phần tử trạng thái là hợp lệ và ổn định được quy định bởi xung đồng hồ
  - Phần tử trạng thái - phần tử nhớ - VD. thanh ghi, bộ nhớ lệnh, bộ nhớ dữ liệu.
  - Kích hoạt theo sườn – các trạng thái thay đổi khi có sườn xung
- **Đọc nội dung của phần tử trạng thái ➔ tính giá trị bằng logic tổ hợp ➔ ghi kết quả vào phần tử trạng thái**
- Các phần tử trạng thái được ghi ở tất cả các chu kỳ đồng hồ.

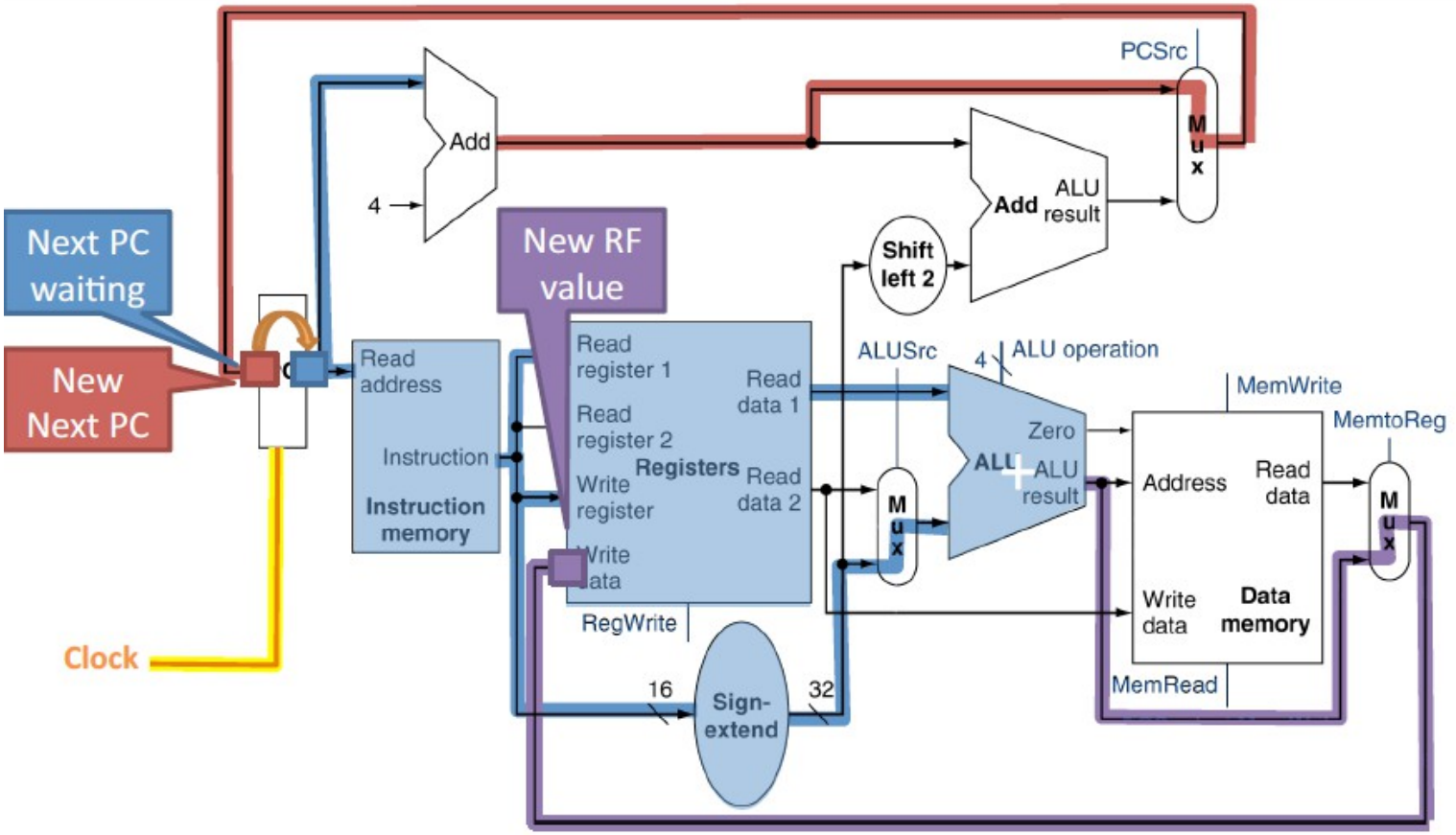




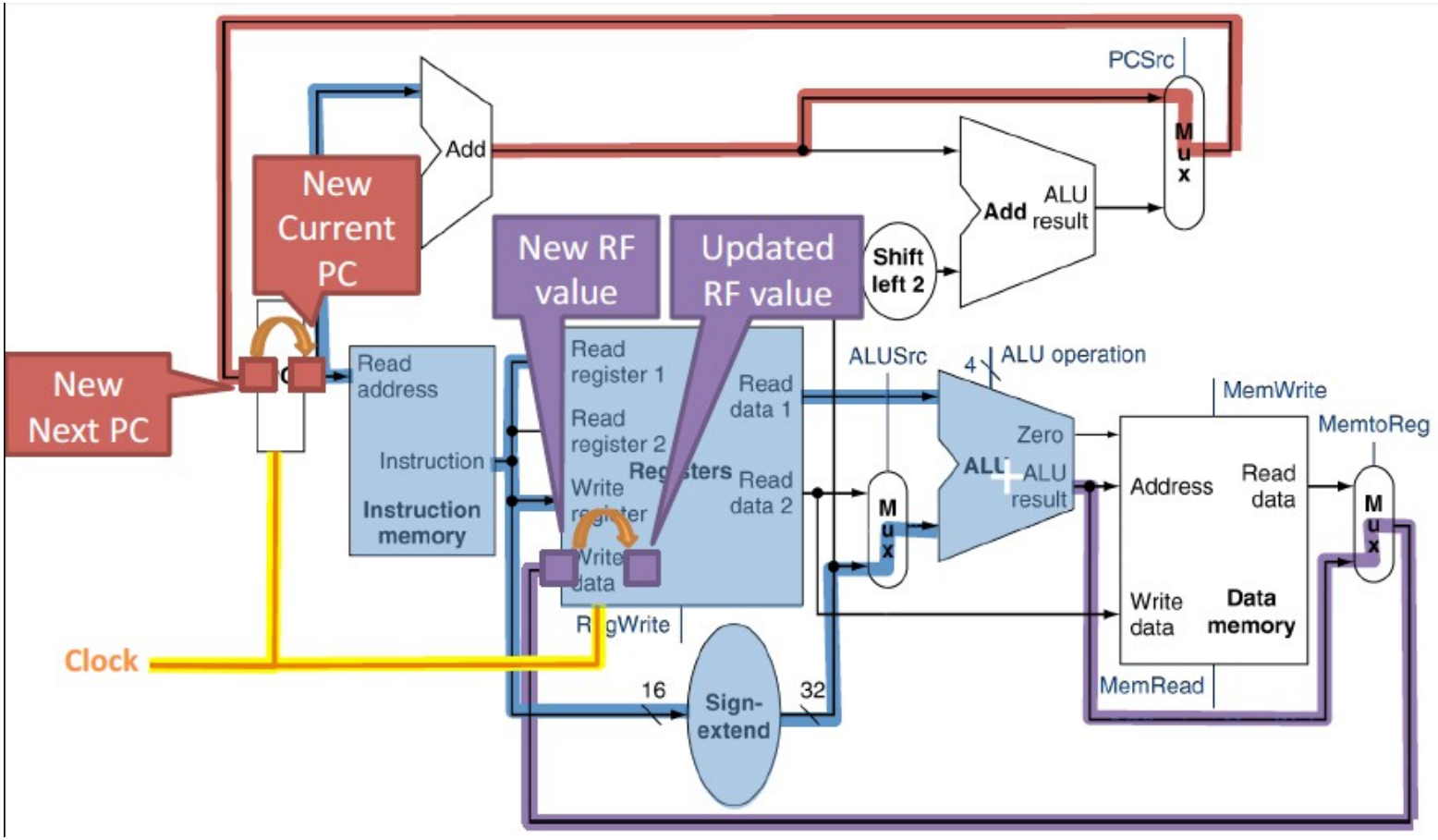
Đồng hồ ở đâu?


PC is updated every clock
Data is written into the register file on the rising edge of the clock, if RegWrte is true.
Data is written into the data memory on the rising edge of the clock, if MemWrite is true.
PCSrc
Add
4
Add
ALU result
M u x
PC
Read address
Instruction
Instruction memory
Read register 1
Read register 2
Write regist
Write data
Registers
Read data 1
Read data 2
RegWrite
ALUSrc
4
ALU operation
Zero
ALU
ALU result
MemWrite
MemtoReg
Address
Read data
Write data
D memory
MemRead
Clock
The clock goes to every state element.
16
Sign-extend
32

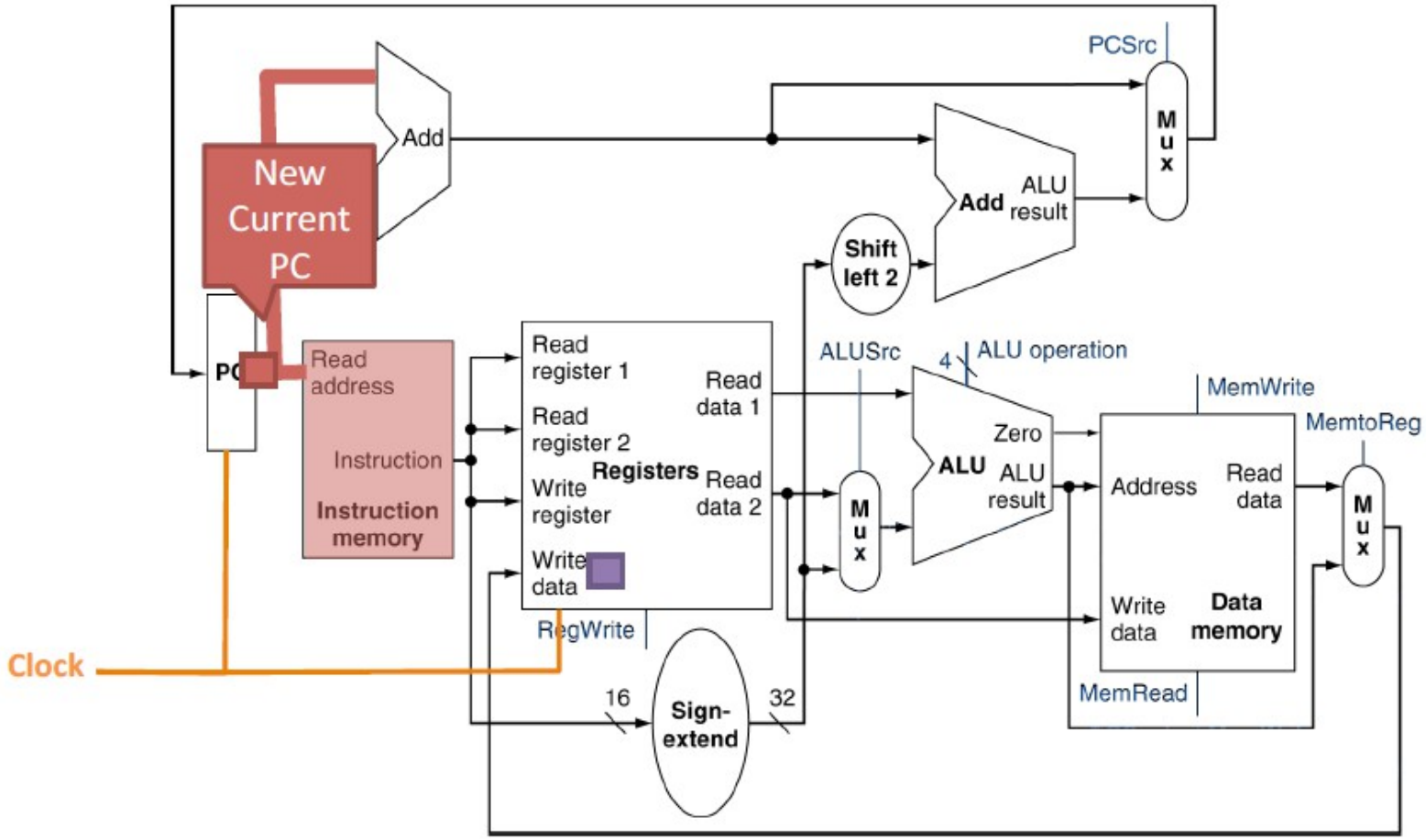
Ví dụ: Lệnh addi thực thi như thế nào?



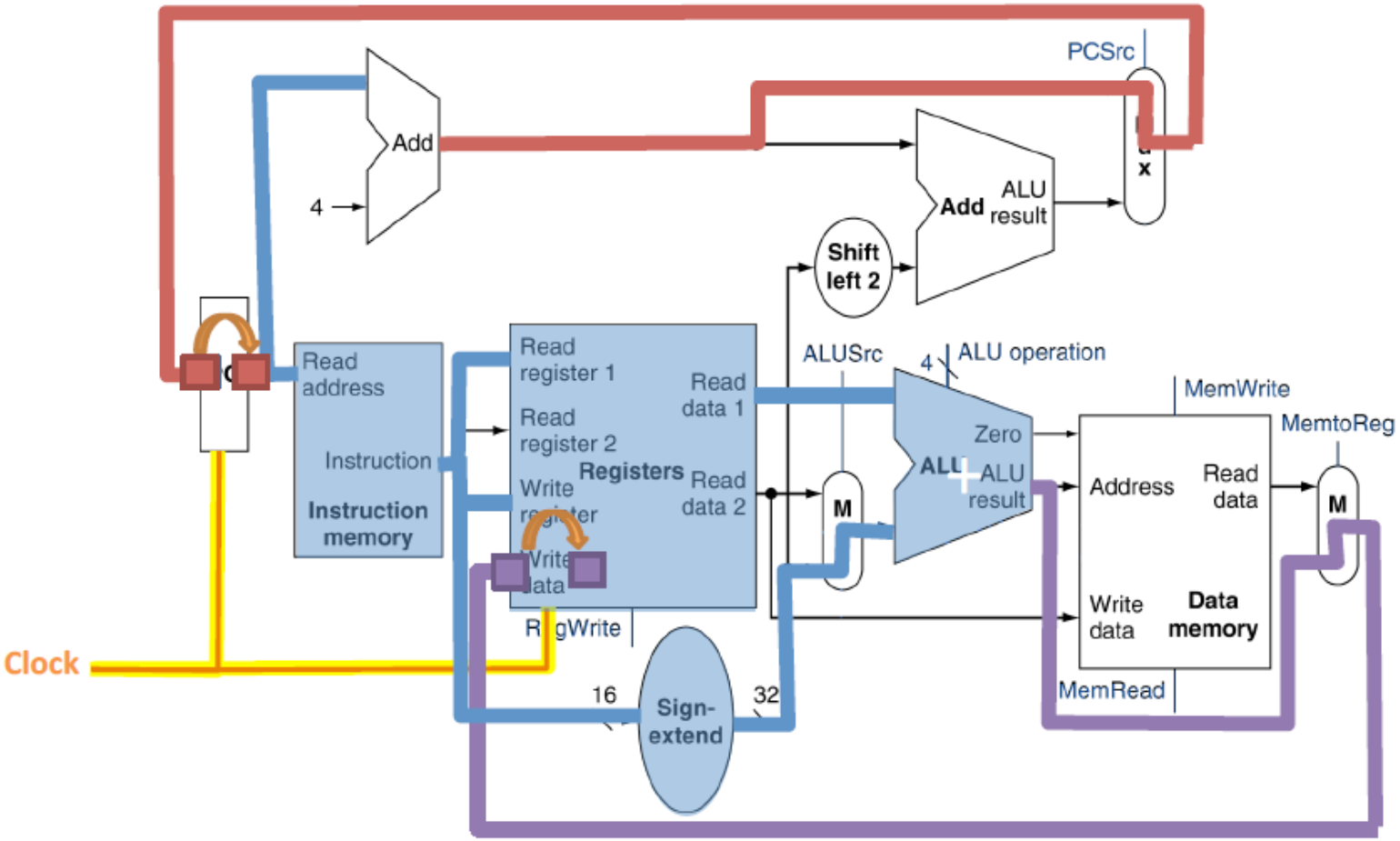
Khi có tín hiệu đồng hồ, biến mới được lưu trữ



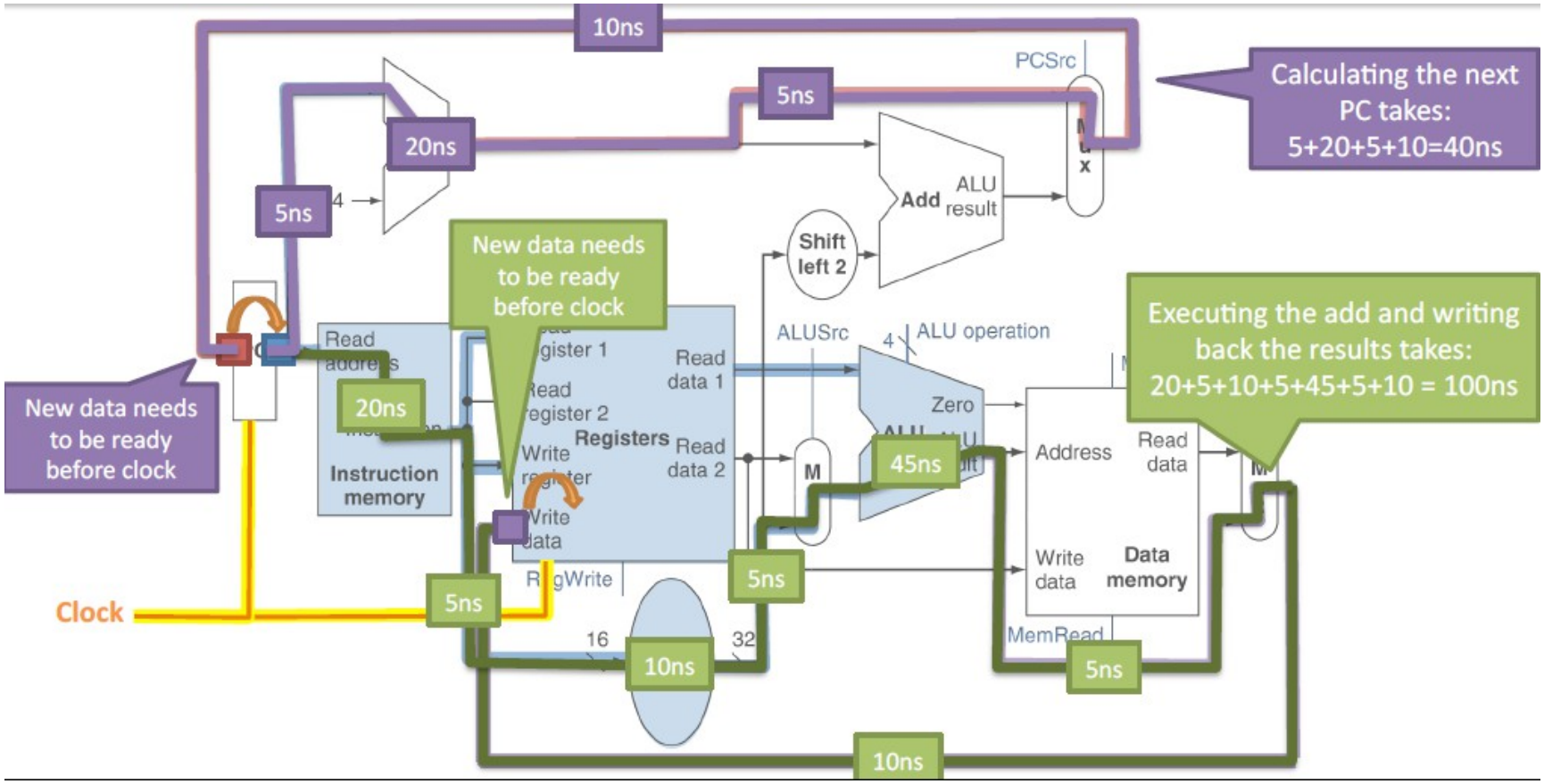
Lặp lại quá trình…



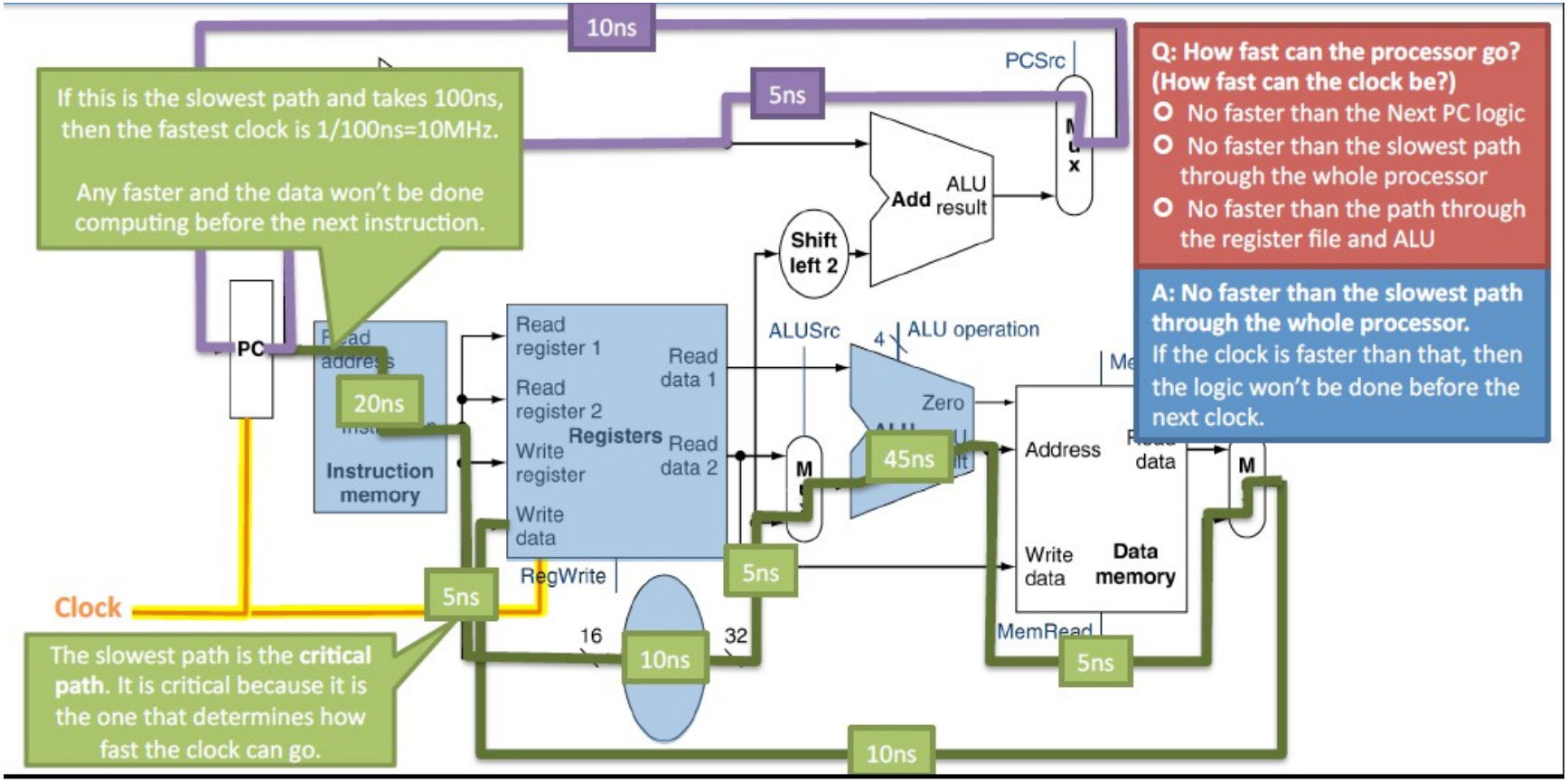
Ví dụ lệnh: addi



Các tuyến logic và các bộ trễ



Tốc độ xử lý xác định như thế nào?



Tính chu ky đồng hồ Tc – Đường dài nhất

- Tính chu kỳ đồng hồ trong trường hợp bỏ qua trễ ở bộ ghép, khối điều khiển, khối mở rộng dấu, khối đọc PC, khối dịch 2, dây dẫn, thời gian thiết lập và giữ. Cho biết độ trễ:
  - Truy cập bộ nhớ lệnh và bộ nhớ dữ liệu (2ns)
  - Khối số học logic và bộ cộng (2 ns)
  - Truy cập tệp thanh ghi (đọc hoặc ghi) (1 ns)

Instr.	I Mem	Reg Rd	ALU Op	D Mem	Reg Wr	Total
R-type	2	1	2		1	6ns
load	2	1	2	2	1	8ns
store	2	1	2	2		7ns
beq	2	1	2			5ns
jump	2					2ns

Hiệu năng thiết kế đơn xung nhịp

Độ trễ logic khi

Truy cập lệnh	2 ns
Đọc thanh ghi	1 ns
Hoạt động ALU	2 ns
Truy cập bộ nhớ DL	2 ns
Ghi thanh ghi	1 ns
Tổng	8 ns

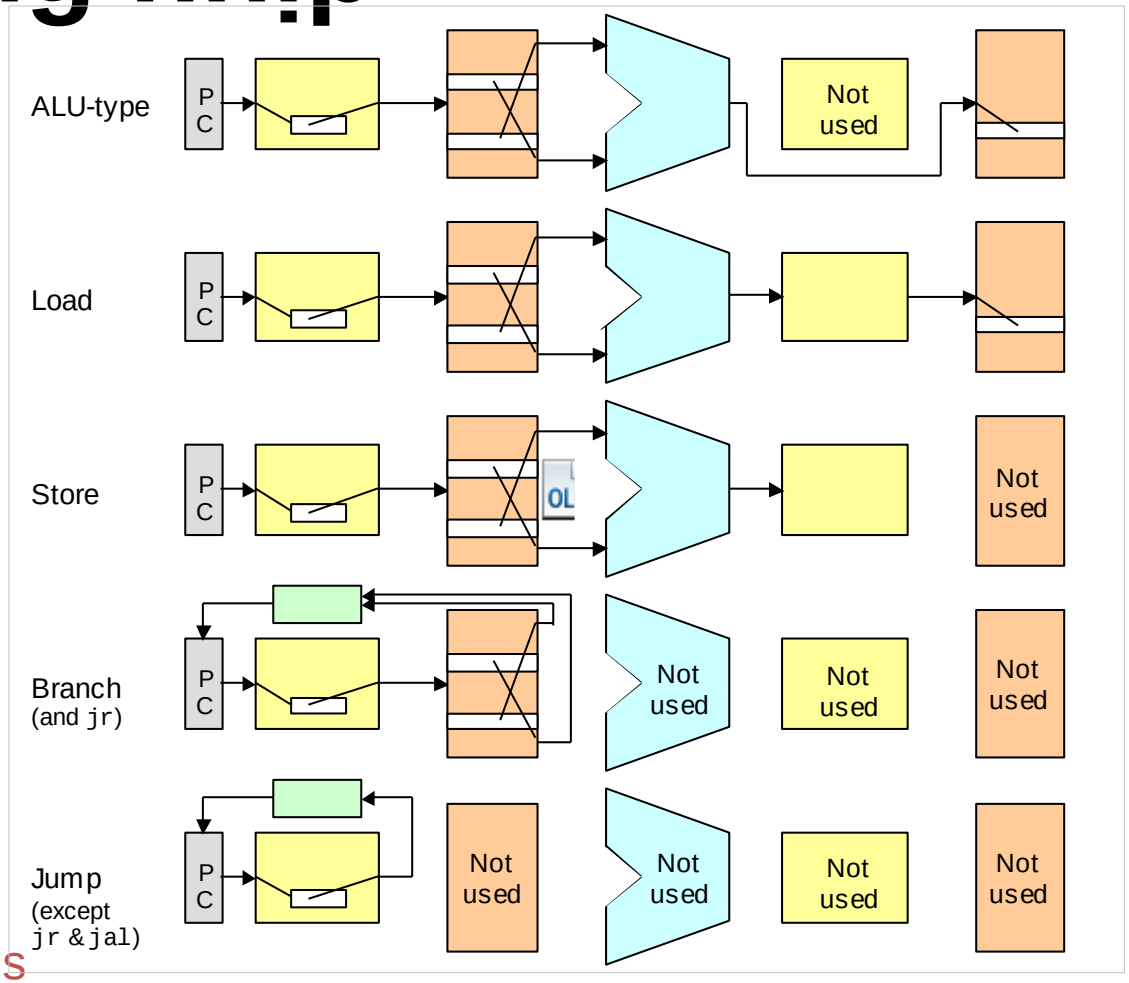
Tốc độ đồng hồ = 125 MHz

Các loại lệnh:

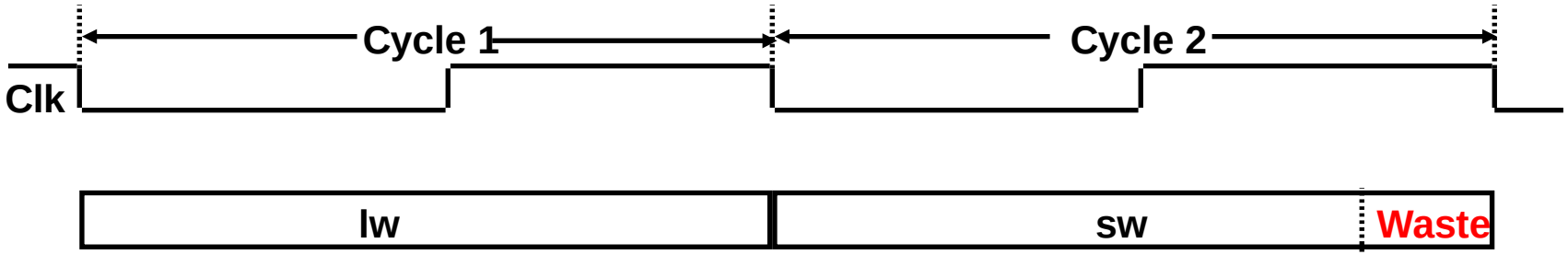
R-type	44%6 ns	
Load	24%	8 ns
Store	12%	7 ns
Branch	18%	5 ns
Jump	2%	4 ns

Thời gian trung bình 6.38 ns

CPI = 1.



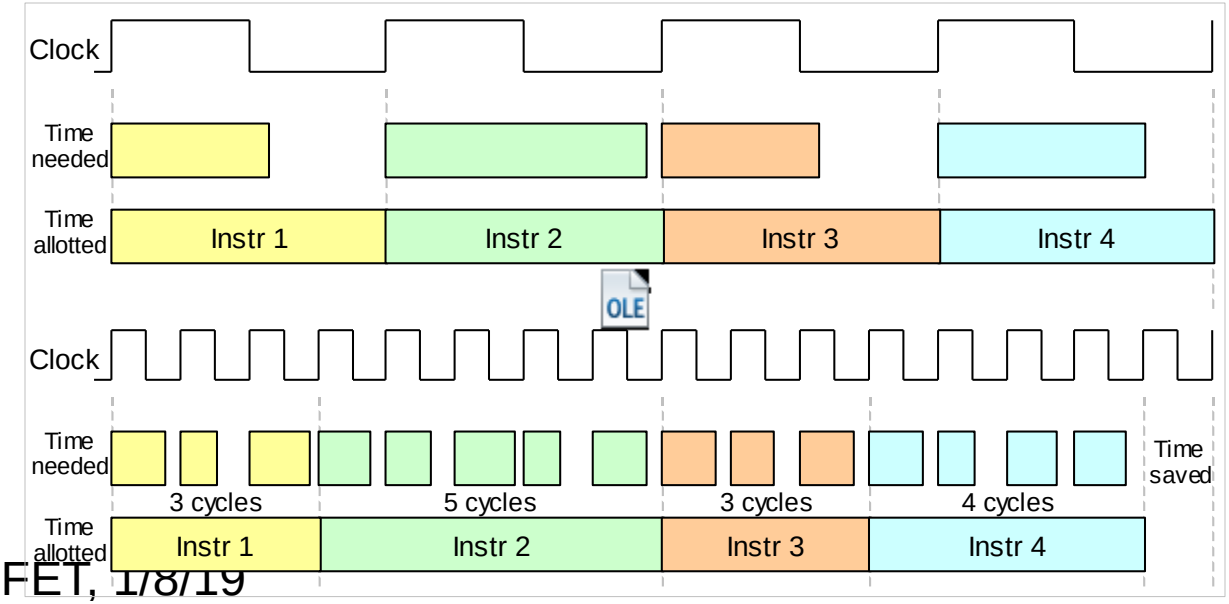
Thiết kế đơn xung nhịp – Ưu nhược điểm



- Sử dụng chu kỳ đồng hồ không hiệu quả – chu kỳ đồng hồ được đặt theo lệnh chậm nhất.
- Các lệnh phức tạp như lệnh nhân dấu phẩy động: Tốn diện tích thiết kế vì cần nhân đôi một số khối chức năng (VD. bộ cộng) vì chúng không thể được chia sẻ trong cùng 1 chu kỳ đồng hồ
- Đơn giản và dễ hiểu

Thiết kế đa xung nhịp

- Chia lệnh thành các pha thực hiện: IF, ID, EX, MEM, WB. Mỗi pha thực hiện trong 1 chu kỳ xung nhịp
- Thời gian thực hiện (= số pha) của mỗi lệnh được điều chỉnh tùy thuộc độ phức tạp của lệnh
- Các khối chức năng được chia sẻ giữa các pha khác nhau của lệnh do một khối chức năng cụ thể không cần trong toàn bộ các pha thực hiện của lệnh



Hiệu năng thiết kế đa xung nhịp

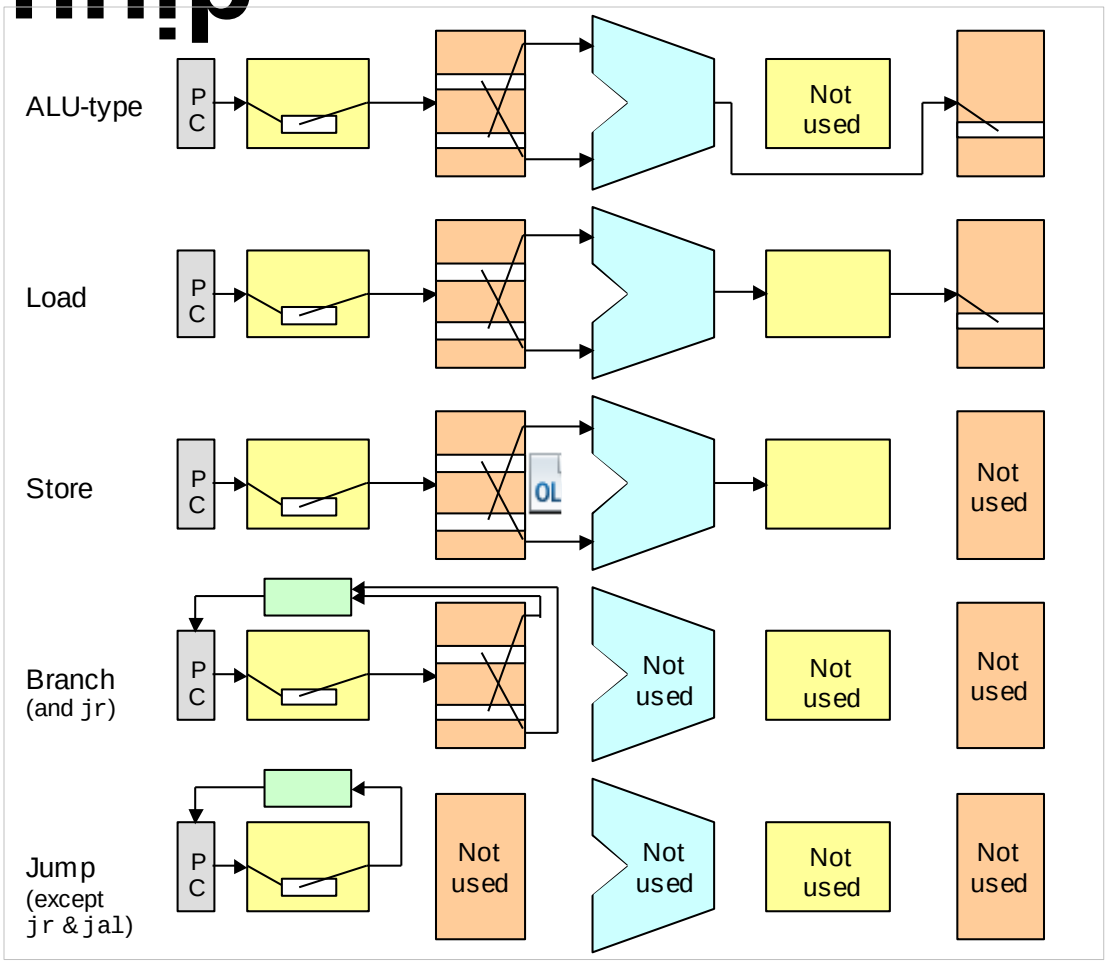
Các loại lệnh sử dụng số chu kỳ khác nhau

R-type	44%	4 cycles
Load	24%	5 cycles
Store	12%	4 cycles
Branch	18%	3 cycles
Jump	2%	2 cycles

Đóng góp vào số chu kỳ trung bình cần cho một lệnh:

R-type
Load
Store
Branch
Jump

CPI trung bình



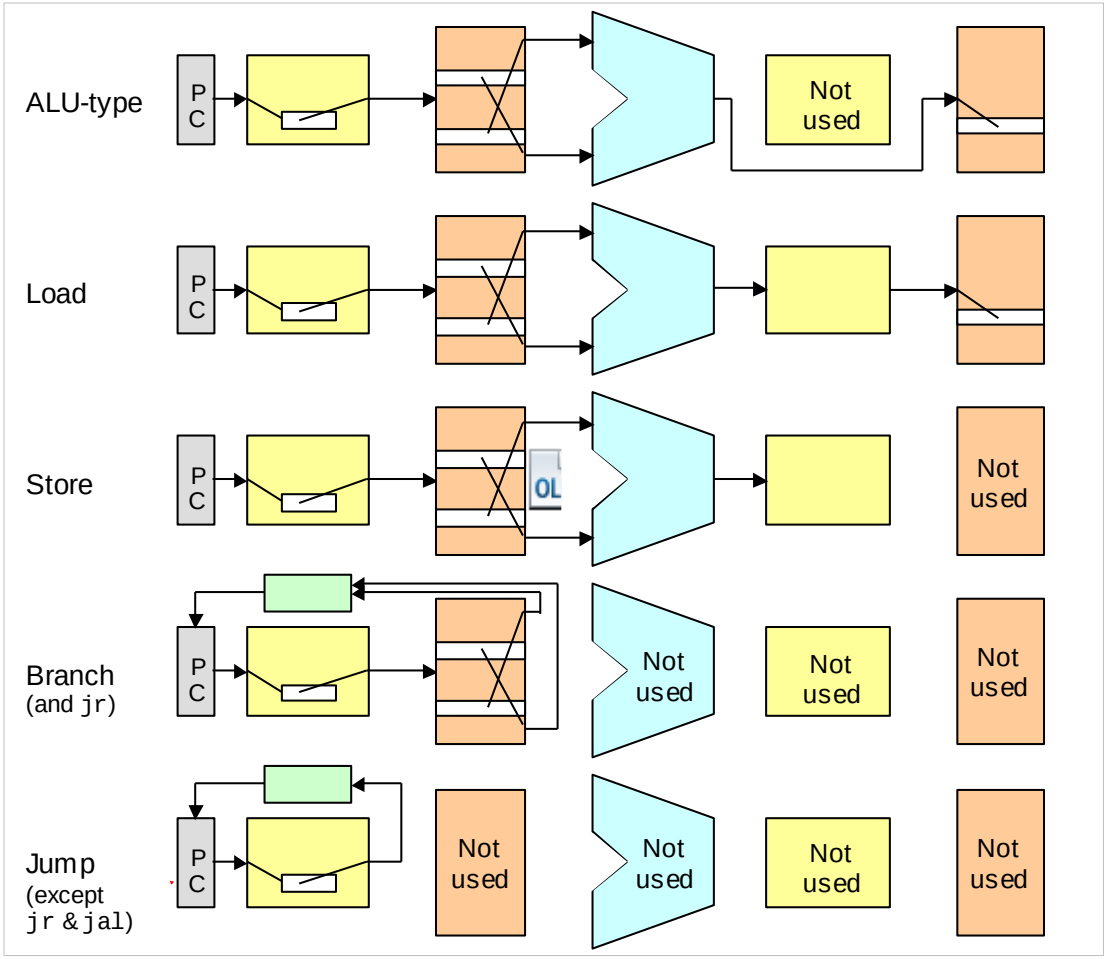
Hiệu năng thiết kế đa xung nhịp

Các loại lệnh sử dụng số chu kỳ khác nhau

R-type	44%	4 cycles
Load	24%	5 cycles
Store	12%	4 cycles
Branch	18%	3 cycles
Jump	2%	2 cycles

Tính số chu kỳ trung bình cần cho một lệnh:

R-type	0.44	4 =	1.76
Load	0.24	5 =	1.20
Store	0.12	4 =	0.48
Branch	0.18	3 =	0.54
Jump	0.02	2 =	0.04
CPI trung bình			4.02



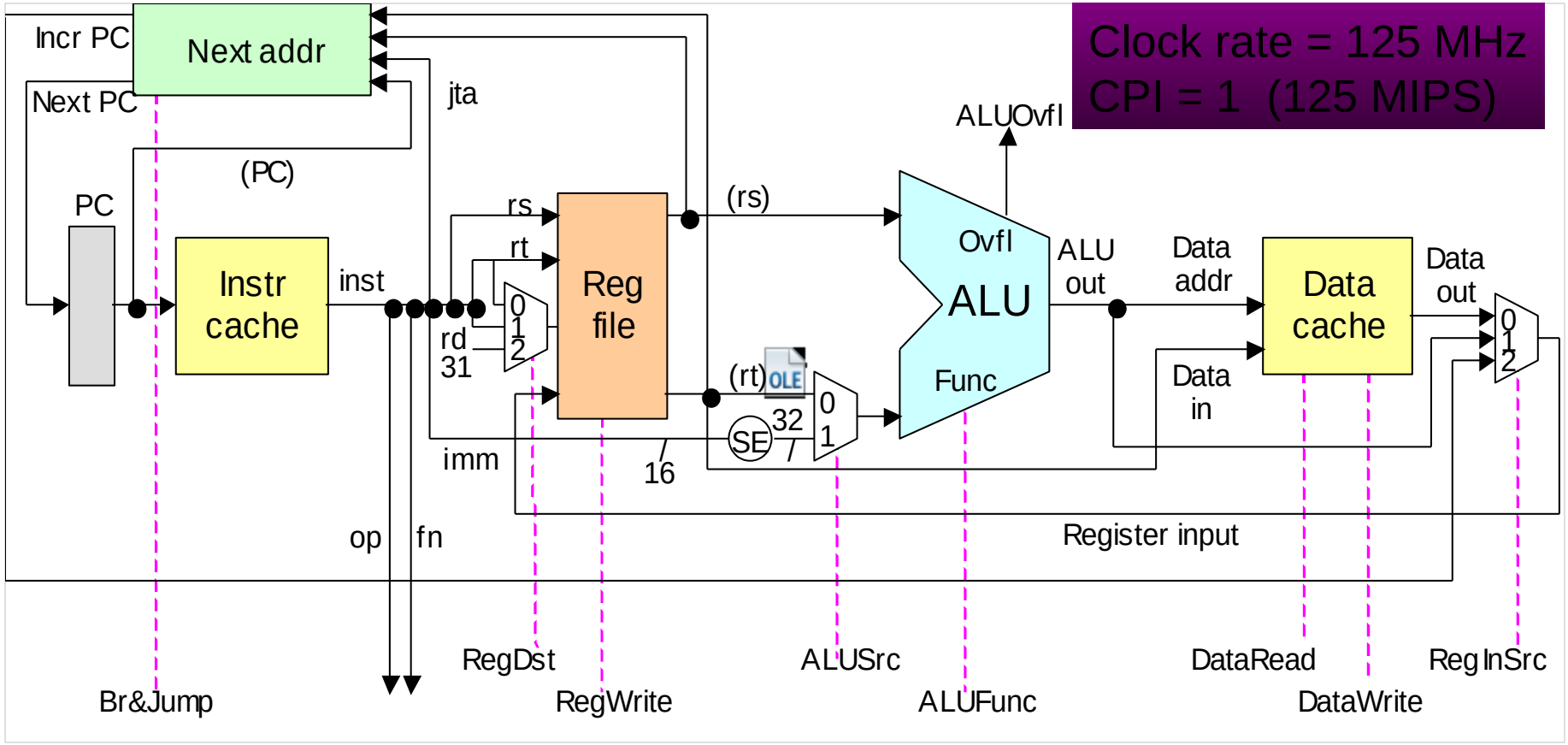
So sánh hiệu năng xử lý

Instr	IF Instruction Fetch	ID Decode & RF Read	EX Execute	MEM Access Memory	WB Write back to RF	Total time
lw	200ps	100ps	200ps	200ps	100ps	800ps
sw	200ps	100ps	200ps	200ps		700ps
R-format	200ps	100ps	200ps		100ps	600ps
beq	200ps	100ps	200ps			500ps

So sánh hiệu năng xử lý của thiết kế đa xung nhịp và thiết đơn xung nhịp, biết tần suất xuất hiện các lệnh như sau:

lw : 20%
sw: 20%
R - : 45 %
beq: 15%

Thiết kế đơn xung nhịp



Thiết kế đa xung nhịp

Clock rate = 500 MHz
CPI 4 (125 MIPS)