



COMPUTER ENGINEERING

# KIẾN TRÚC MÁY TÍNH



**UIT**  
TRƯỜNG ĐẠI HỌC  
CÔNG NGHỆ THÔNG TIN

## Tuần 10

# BỘ XỬ LÝ PROCESSOR



## Mục đích:

- ✓ Hiểu cơ chế thực thi lệnh và các quy ước về thiết kế logic
- ✓ Thiết kế Datapath với 8 lệnh cơ bản cho một bộ xử lý và cách hiện thực thiết kế này.

Slide tham khảo từ:

1. ***Computer Organization and Design: The Hardware/Software Interface***, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.
2. ***NUS***, Singapore



- 1. Giới thiệu**
2. Nhắc lại các quy ước thiết kế logic
3. Xây dựng đường dữ liệu (datapath) đơn giản
4. Hiện thực datapath đơn chu kỳ



# Giới thiệu

## ❖ Hiệu suất của một máy tính được xác định bởi ba yếu tố:

- Tổng số câu lệnh Được xác định bởi trình biên dịch và kiến trúc tập lệnh
- Chu kỳ xung clock
- Số chu kỳ xung clock trên một lệnh Được xác định bởi quá trình hiện thực bộ xử lý  
(Clock cycles per instruction – CPI)

## ❖ Mục đích chính của chương này:

- Giải thích quy tắc hoạt động và hướng dẫn xây dựng datapath cho một bộ xử lý chứa một số lệnh đơn giản (giống kiến trúc tập lệnh dạng MIPS), gồm hai ý chính:
  - Thiết kế datapath
  - Hiện thực datapath đã thiết kế

MIPS (bắt nguồn từ chữ viết tắt của ‘Microprocessor without Interlocked Pipeline Stages’) là một kiến trúc tập lệnh dạng RISC, được phát triển bởi MIPS Technologies (trước đây là MIPS Computer Systems, Inc.)



## 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 nhảy (Lệnh nhảy với điều kiện bằng **beq**)



## Tổng quan các lệnh cần xem xét:

### Nhóm lệnh tham khảo bộ nhớ:

Nạp lệnh ☐ Đọc một/hai thanh ghi ☐ Sử dụng ALU ☐ Truy xuất bộ nhớ để đọc/ghi dữ liệu

### Nhóm lệnh logic và số học:

Nạp lệnh ☐ Đọc một/hai thanh ghi ☐ Sử dụng ALU ☐ Ghi dữ liệu vào thanh ghi

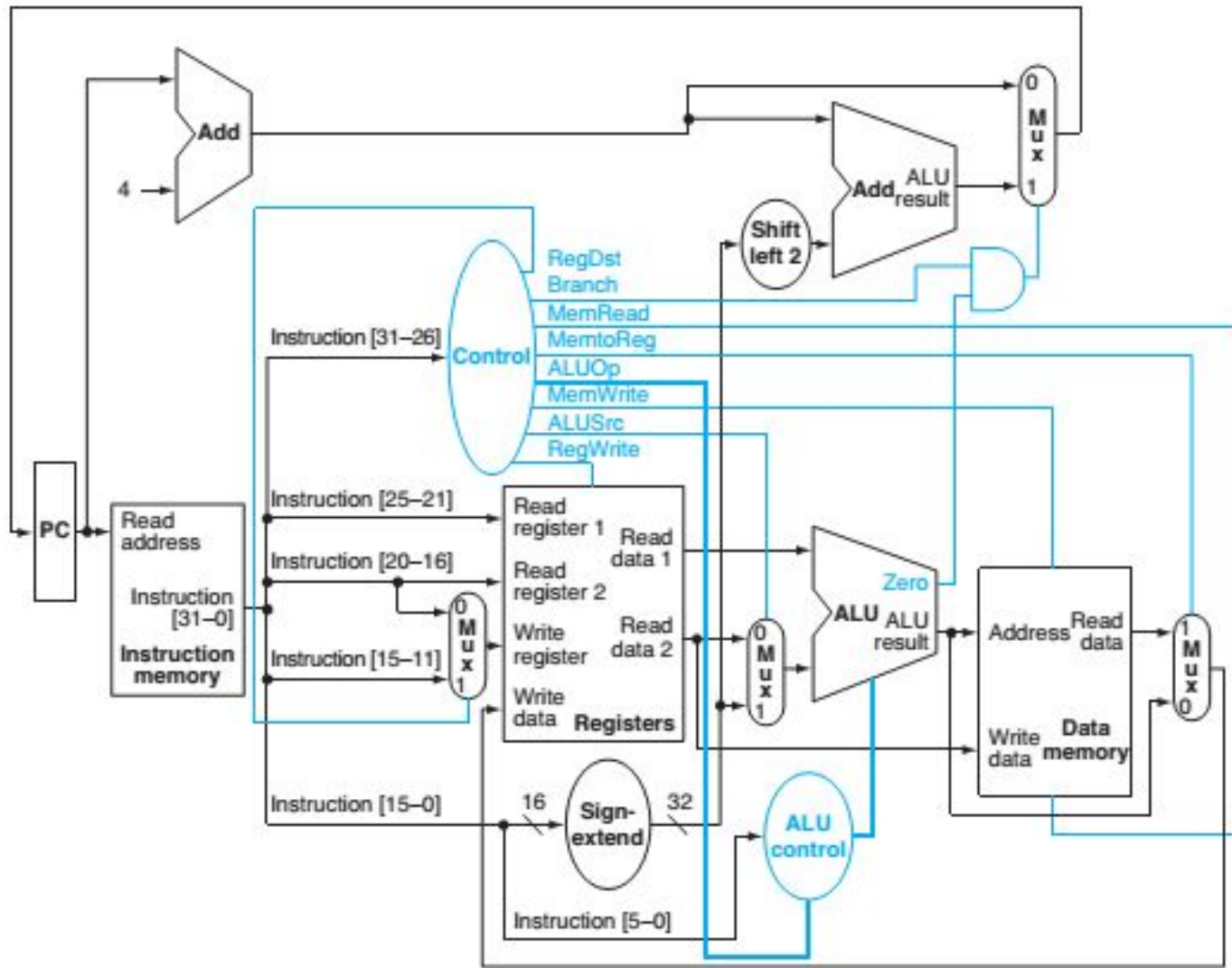
### Nhóm lệnh nhảy:

Nạp lệnh ☐ Đọc một/hai thanh ghi ☐ Sử dụng ALU ☐ Chuyển đến địa chỉ lệnh tiếp theo dựa trên kết quả so sánh



# Giới thiệu

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





1. Giới thiệu
- 2. Nhắc lại các quy ước thiết kế logic**
3. Xây dựng đường dữ liệu (datapath) đơn giản
4. Hiện thực datapath đơn chu kỳ





# Quy ước thiết kế



Phần này nhắc lại các khái niệm:

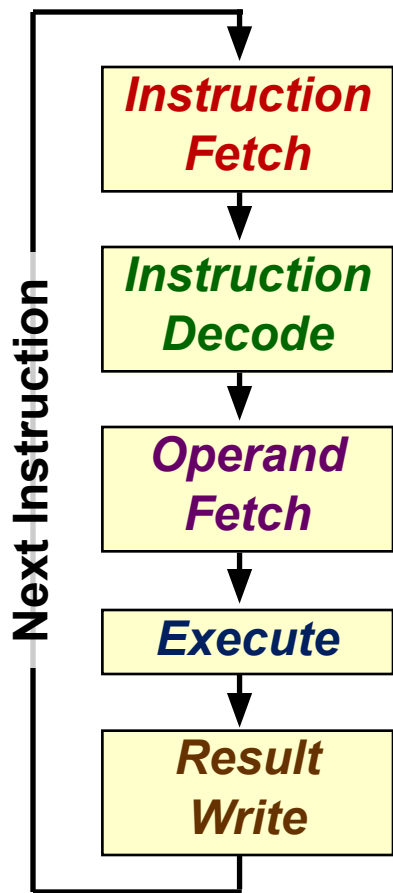
- ❖ **Mạch tổ hợp (Combinational): ALU**
- ❖ **Mạch tuần tự (Sequential): instruction/data memories và thanh ghi**
- ❖ **Tín hiệu điều khiển (Control signal)**
- ❖ **Tín hiệu dữ liệu (Data signal)**
  - **Asserted (assert):** Khi tín hiệu ở mức cao hoặc ‘true’
  - **Deasserted (deassert):** Khi tín hiệu ở mức thấp hoặc ‘false’
  - **Edge-triggered clocking (Rising/Falling)**



1. Giới thiệu
2. Nhắc lại các quy ước thiết kế logic
- 3. Xây dựng đường dữ liệu (datapath) đơn giản**
4. Hiện thực datapath đơn chu kỳ



# Quy trình thực thi lệnh



## ■ **Instruction Fetch (tìm nạp lệnh):**

- Nạp lệnh từ bộ nhớ (memory)
- Địa chỉ của lệnh lưu trong thanh ghi Program Counter (PC)

## ■ **Instruction Decode (giải mã lệnh):**

- Tìm ra lệnh thực hiện

## ■ **Operand Fetch (tìm nạp toán hạng):**

- Lấy các toán hạng cần thiết cho lệnh

## ■ **Execute (thực thi):**

- Thực hiện câu lệnh

## ■ **Result Write (lưu trữ):**

- Lưu trữ kết quả



# Quy trình thực thi lệnh

- Bảng sau mô tả ba giai đoạn thực thi lệnh trong ba nhóm lệnh cơ bản của MIPS (Giai đoạn *Fetch* and *Decode* không được hiển thị)

	<b>add \$3, \$1, \$2</b>	<b>lw \$3, 20( \$1 )</b>	<b>beq \$1, \$2, label</b>
Fetch & Decode	<i>standard</i>	<i>standard</i>	<i>standard</i>
Operand Fetch	<ul style="list-style-type: none"> <li>Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i></li> <li>Đọc thanh ghi \$2, xem như toán hạng <i>opr2</i></li> </ul>	<ul style="list-style-type: none"> <li>Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i></li> <li>Sử dụng <b>20</b> như toán hạng <i>opr2</i></li> </ul>	<ul style="list-style-type: none"> <li>Đọc thanh ghi \$1, xem như toán hạng <i>opr1</i></li> <li>Đọc thanh ghi \$2, xem như toán hạng <i>opr2</i></li> </ul>
Execute	$Result = opr1 + opr2$	<ul style="list-style-type: none"> <li><math>MemAddr = opr1 + opr2</math></li> <li>Sử dụng <i>MemAddr</i> để đọc dữ liệu từ bộ nhớ</li> </ul>	$Taken = (opr1 == opr2) ?$ $Target = PC + Label*$
Result Write	<i>Result</i> được lưu trữ vào <b>\$3</b>	Dữ liệu của từ nhớ có địa chỉ <i>MemAddr</i> được được lưu trữ vào <b>\$3</b>	if ( <i>Taken</i> ) $PC = Target$

- opr** = Operand
- MemAddr** = Memory Address
- \* = simplification, not exact



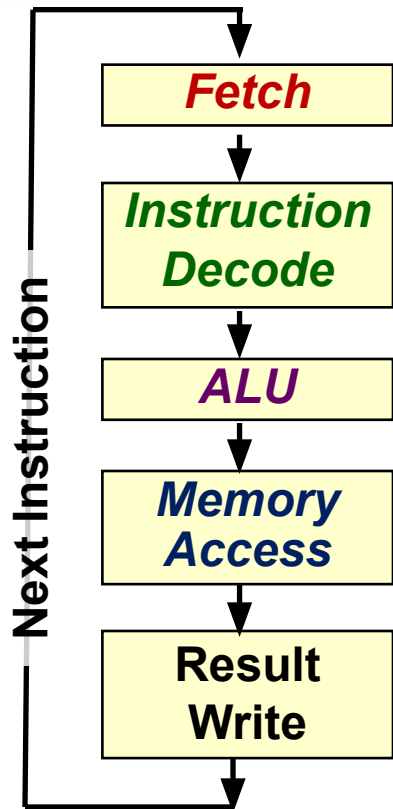
# Quy trình thực thi lệnh của MIPS (5 công đoạn)

- Thay đổi thiết kế các giai đoạn thực hiện lệnh:
- ✓ Gộp giai đoạn *Decode* và *Operand Fetch* – Giai đoạn *Decode* của MIPS khá đơn giản
- ✓ Tách giai đoạn *Execute* thành *ALU* (Calculation) và *Memory Access*

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



# Quy trình thực thi lệnh của MIPS (5 công đoạn)



■ **Instruction Fetch** (Nạp lệnh)

■ **Instruction Decode & Operand Fetch**

(Giải mã và lấy các toán hạng cần thiết, Gọi tắt là “Instruction Decode”)

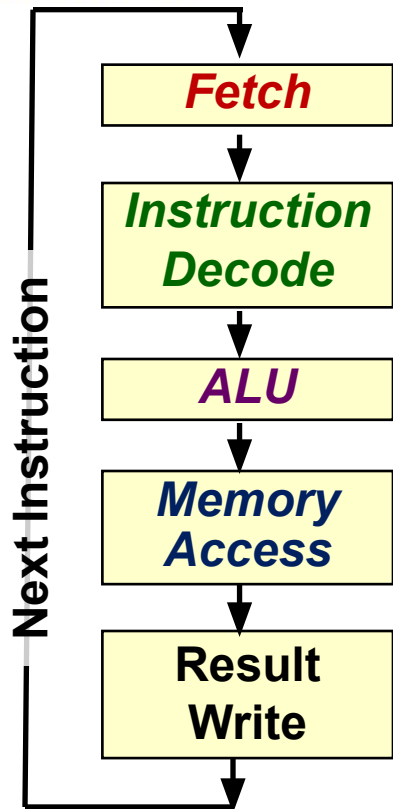
■ **ALU** (Giai đoạn sử dụng ALU hay giai đoạn thực thi)

■ **Memory Access** (Giai đoạn truy xuất vùng nhớ)

■ **Result Write** (Giai đoạn ghi lại kết quả/lưu trữ)



# Quy trình thực thi lệnh của MIPS (5 công đoạn)



- **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:

1. 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
2. **Tăng giá trị** trong thanh ghi PC lên 4 đơn vị để lấy địa chỉ của lệnh tiếp theo
  - Tại sao địa chỉ lệnh tiếp theo là  $PC + 4$ ?
  - **Chú ý**, lệnh rẽ nhánh (**branch**) và lệnh nhảy (**jump**) là một trường hợp ngoại lệ

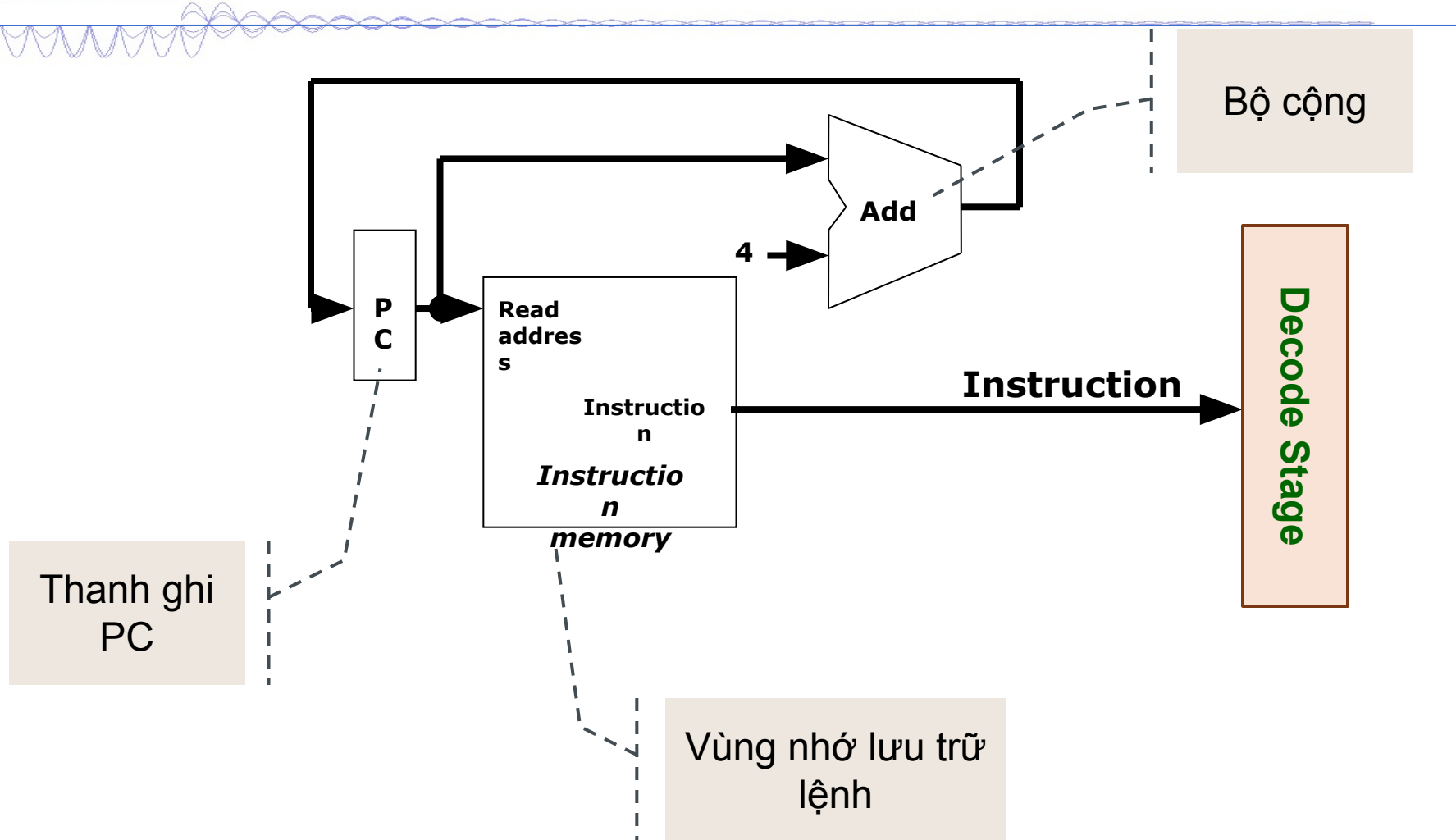
## ■ Kết quả của giai đoạn này là đầu vào cho giai đoạn tiếp theo (**Decode**):

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 Decode





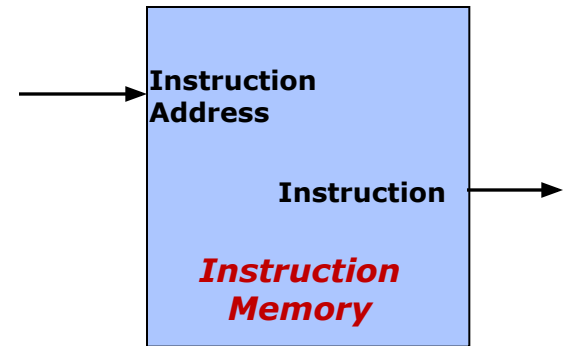
# Giai đoạn tìm nạp lệnh (Instruction Fetch)





# 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
.....	.....



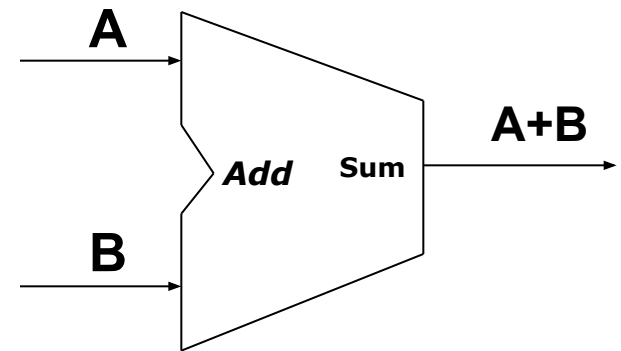
## ■ Mạch logic kết hợp để cộng 2 số - bộ cộng

### ■ Đầu vào:

□ Hai số 32-bit **A**, **B**

### ■ Đầu ra:

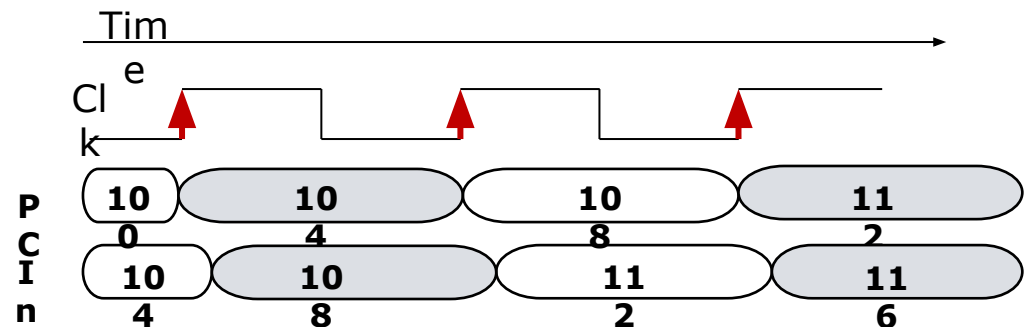
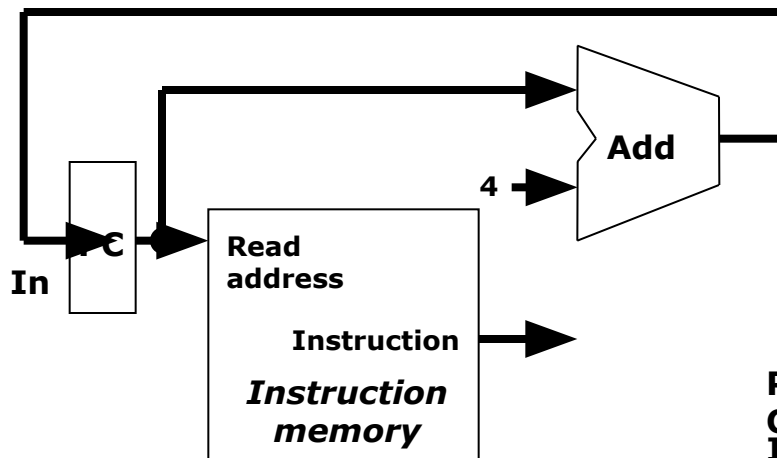
□ **A + B**





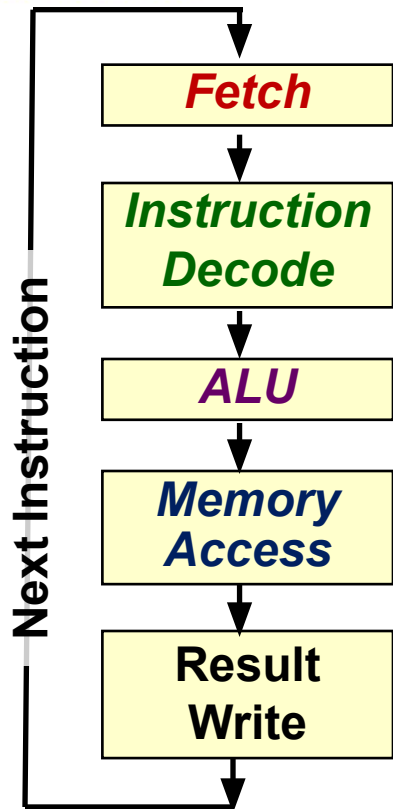
# Ý niệm về việc sử dụng xung clock

- Đường như thanh ghi PC được đọc và cập nhật cùng lúc:
  - PC hoạt động chính xác như thế nào?
- **Magic of clock:**
  - PC được đọc trong nửa clock đầu và cập nhật thành PC+4 trong lần **kích cạnh lên tiếp theo**





# Quy trình thực thi lệnh của MIPS (5 công đoạn)



- **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:

1. Đọc **opcode** để xác định kiểu lệnh và chiều dài của từng trường trong mã máy
2. 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**):

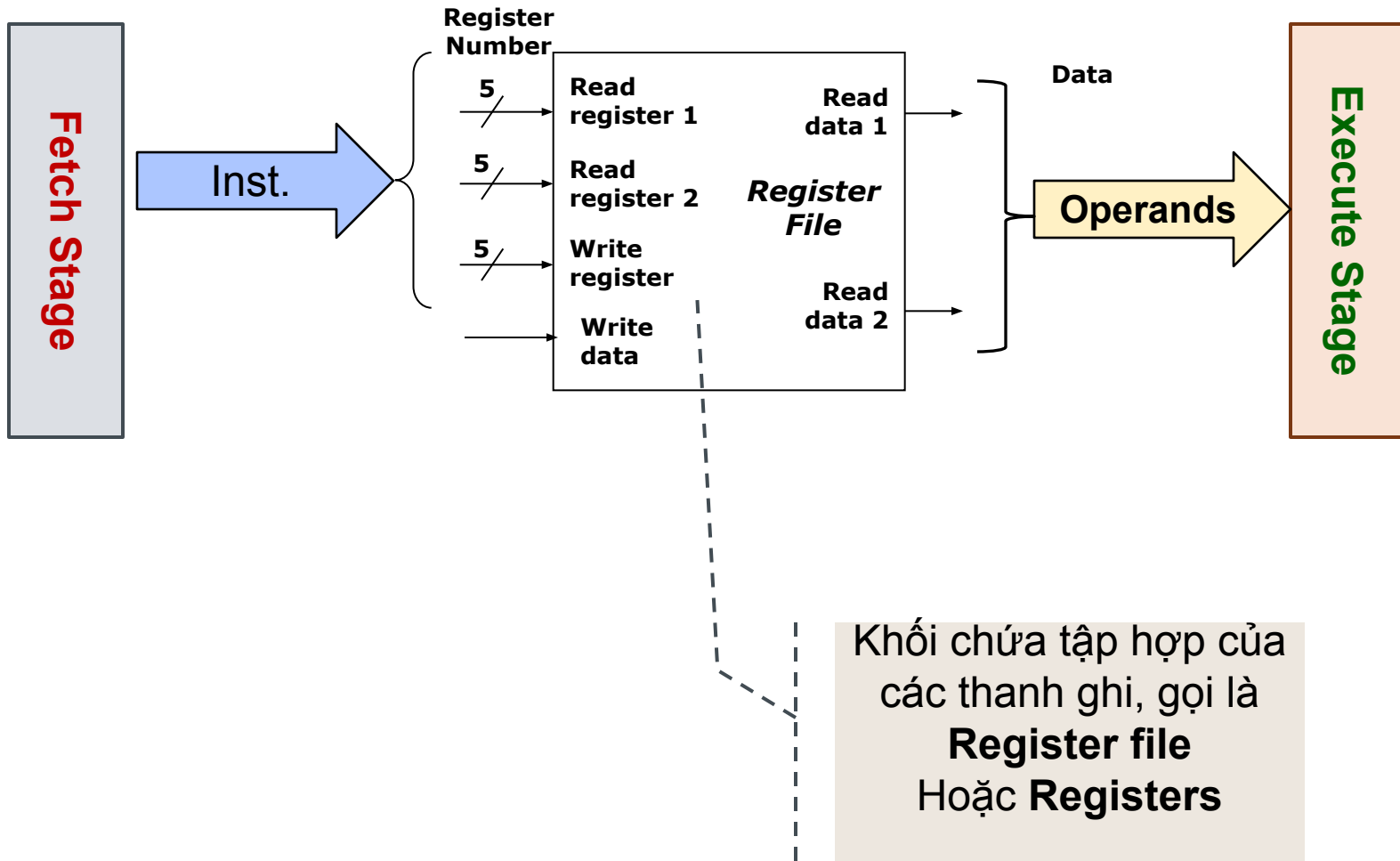
Lệnh cần được thực thi (Mã máy)

## ■ Đầu ra cho giai đoạn tiếp theo (**Execute**):

Phép tính và các toán hạng cần thiết



# Giai đoạn giải mã (Decode)





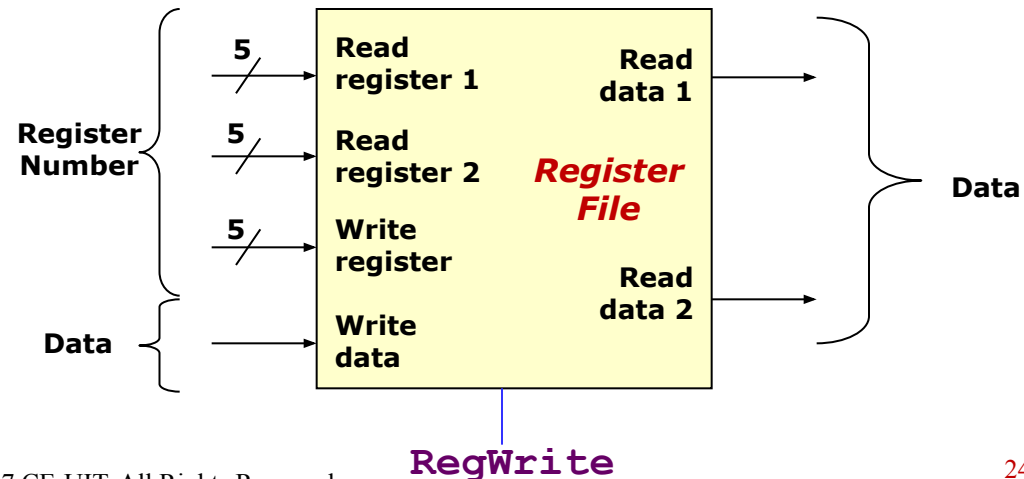
# Khôi *Register File*

## ■ 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
- 1(True) = Write, 0(False) = No Write

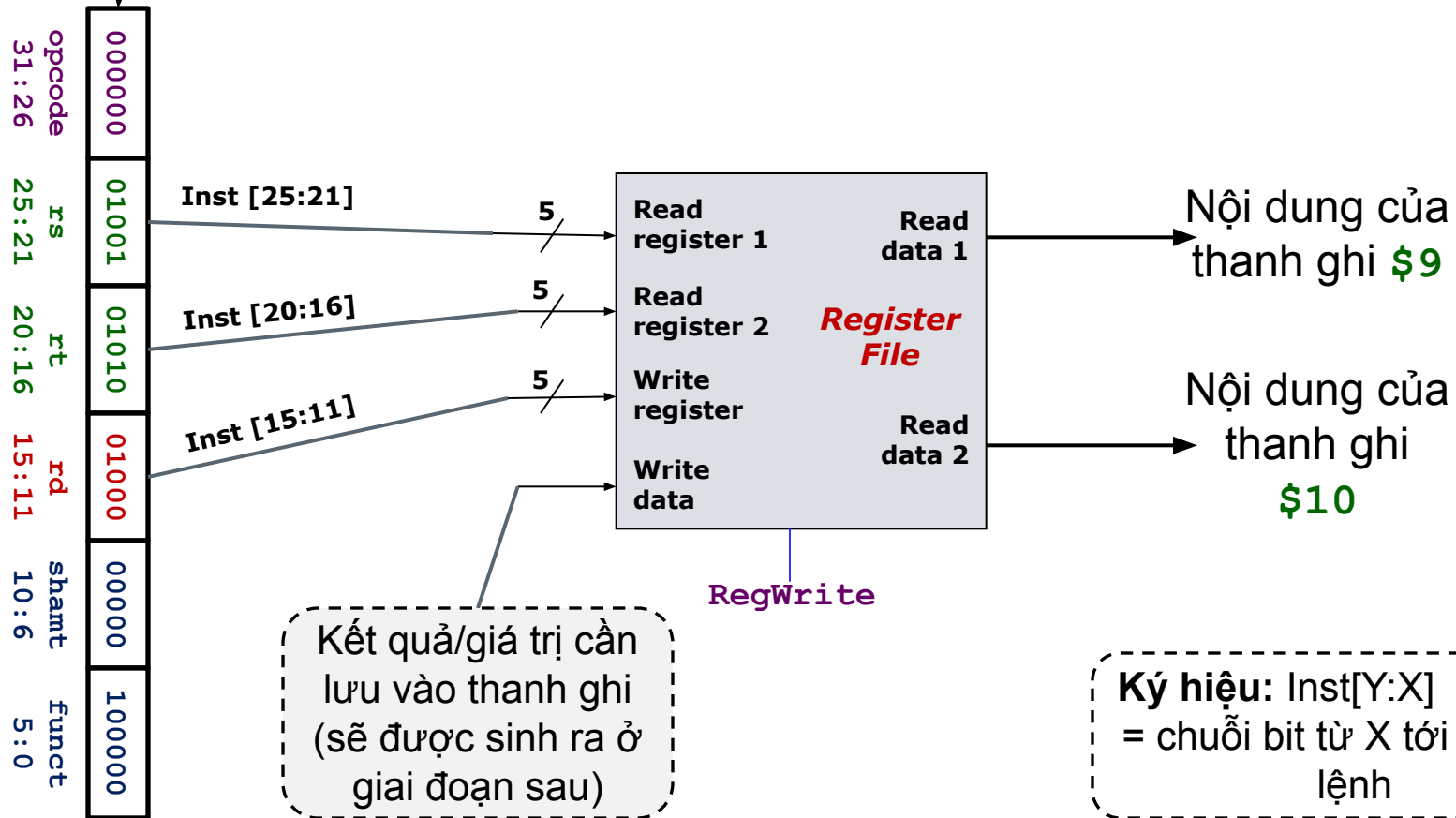






# Giải mã: lệnh R-Type

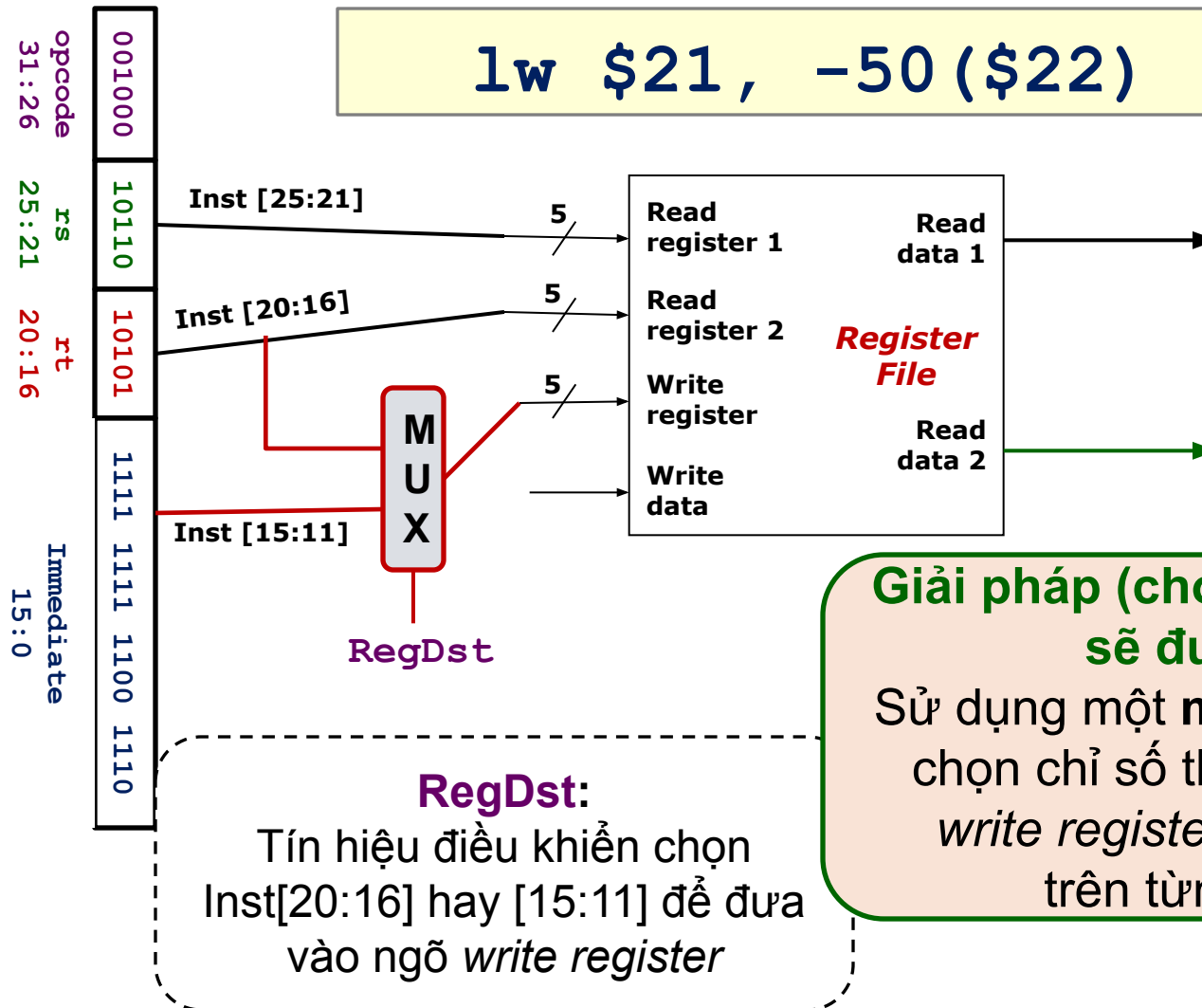
**add \$8, \$9, \$10**







# Giải mã: Giải pháp cho 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



# Multiplexer (MUX)

## ■ Chức năng:

- Chọn một input từ tập input đầu vào

## ■ Inputs:

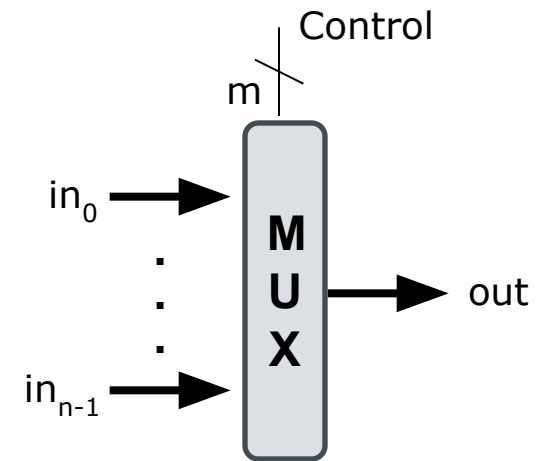
- $n$  đường vào có cùng chiều rộng

## ■ Control:

- Cần  $m$  bit trong đó  $n = 2^m$

## ■ Output:

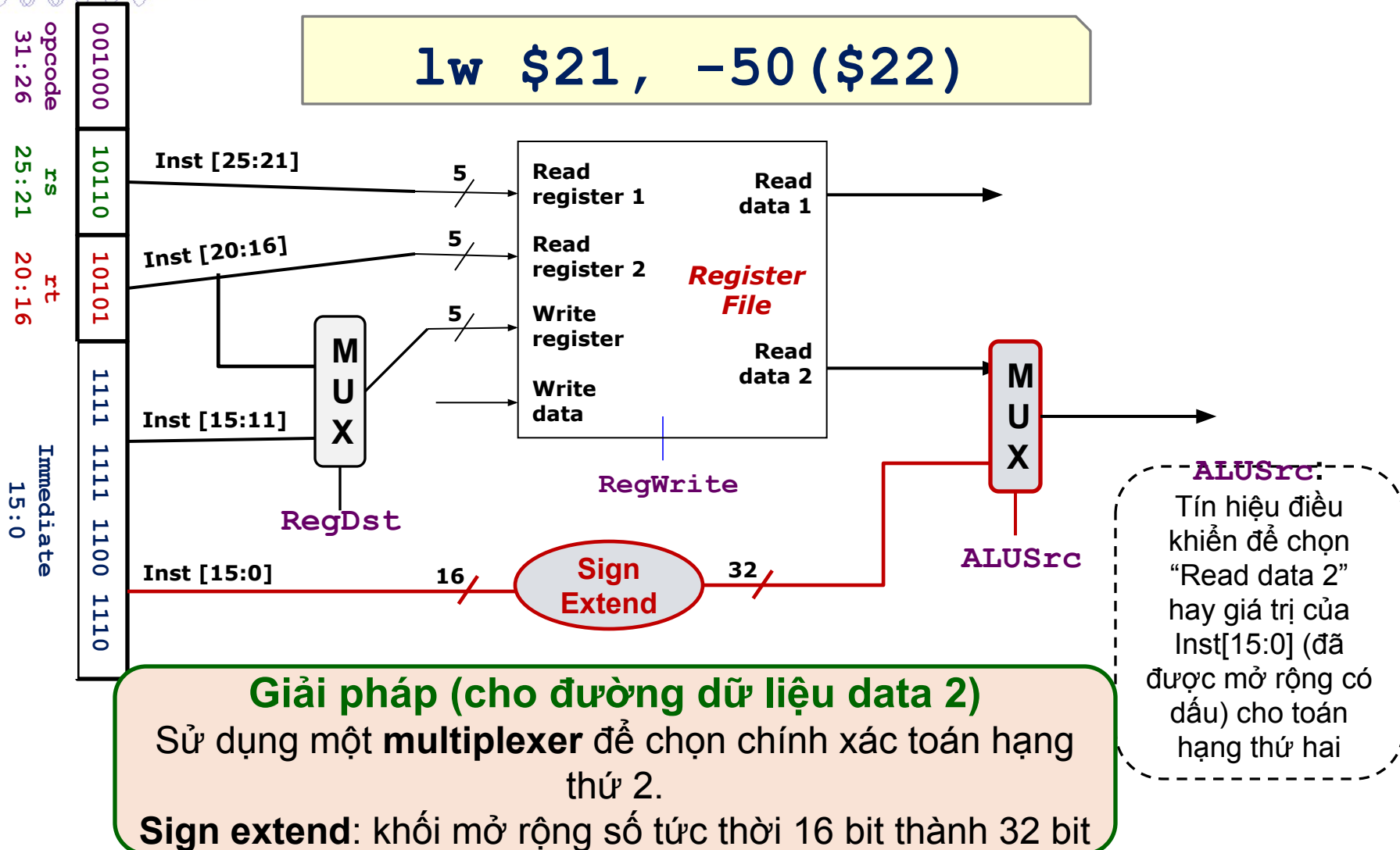
- Chọn đường input thứ  $i$  nếu giá trị tín hiệu điều khiển  $\text{control} = i$



Control=0 □ select  $\text{in}_0$   
Control=3 □ select  $\text{in}_3$

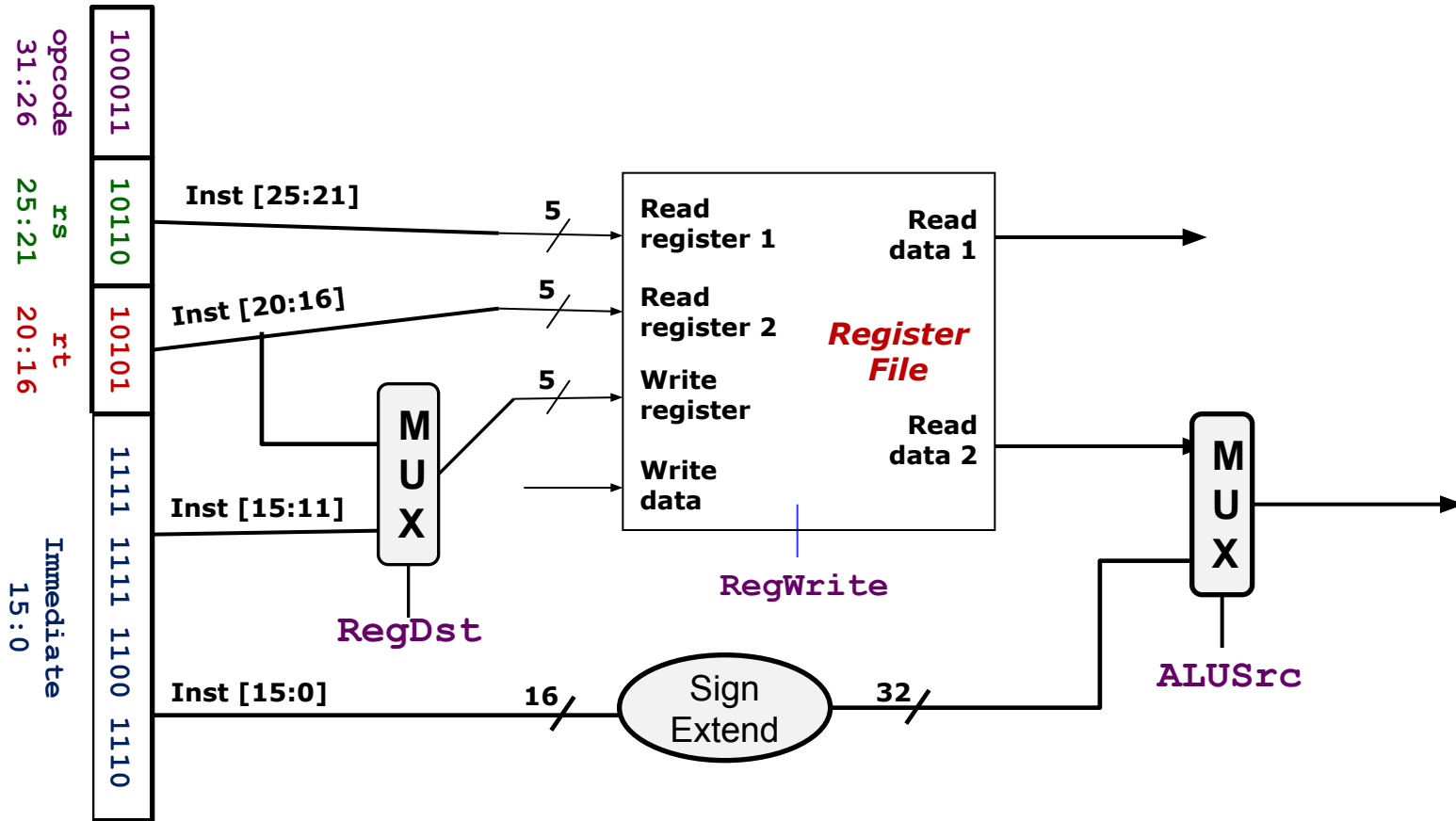


# Giải mã: giải pháp cho ngõ “Data 2”





# Giải mã: Lệnh Load Word



□ Ví dụ với lệnh: “**sw** **\$21**, **-50** (**\$22**)”

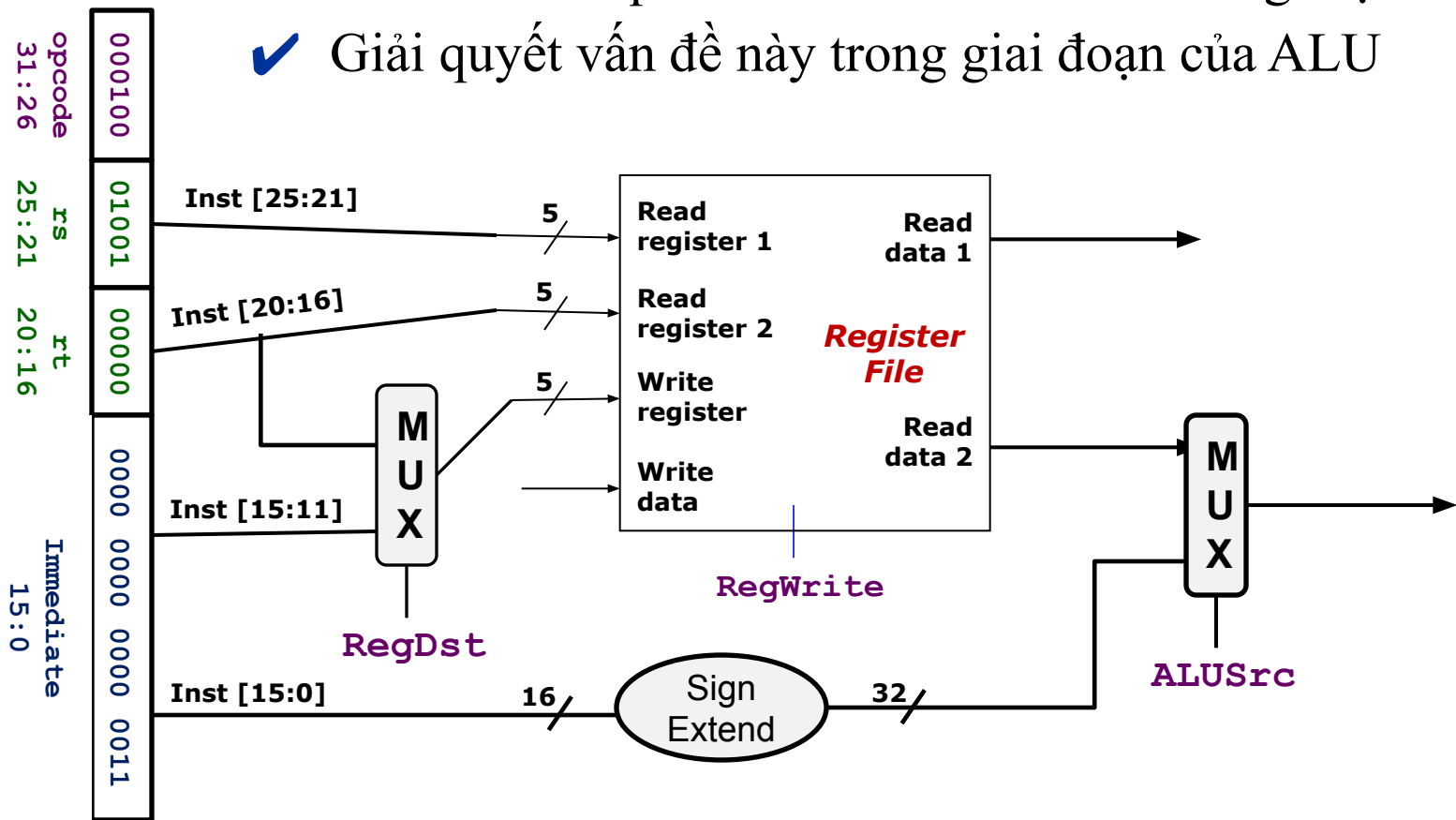
– Có cần phải thay đổi thành phần nào?



# Giải mã: Lệnh nhánh/nhảy

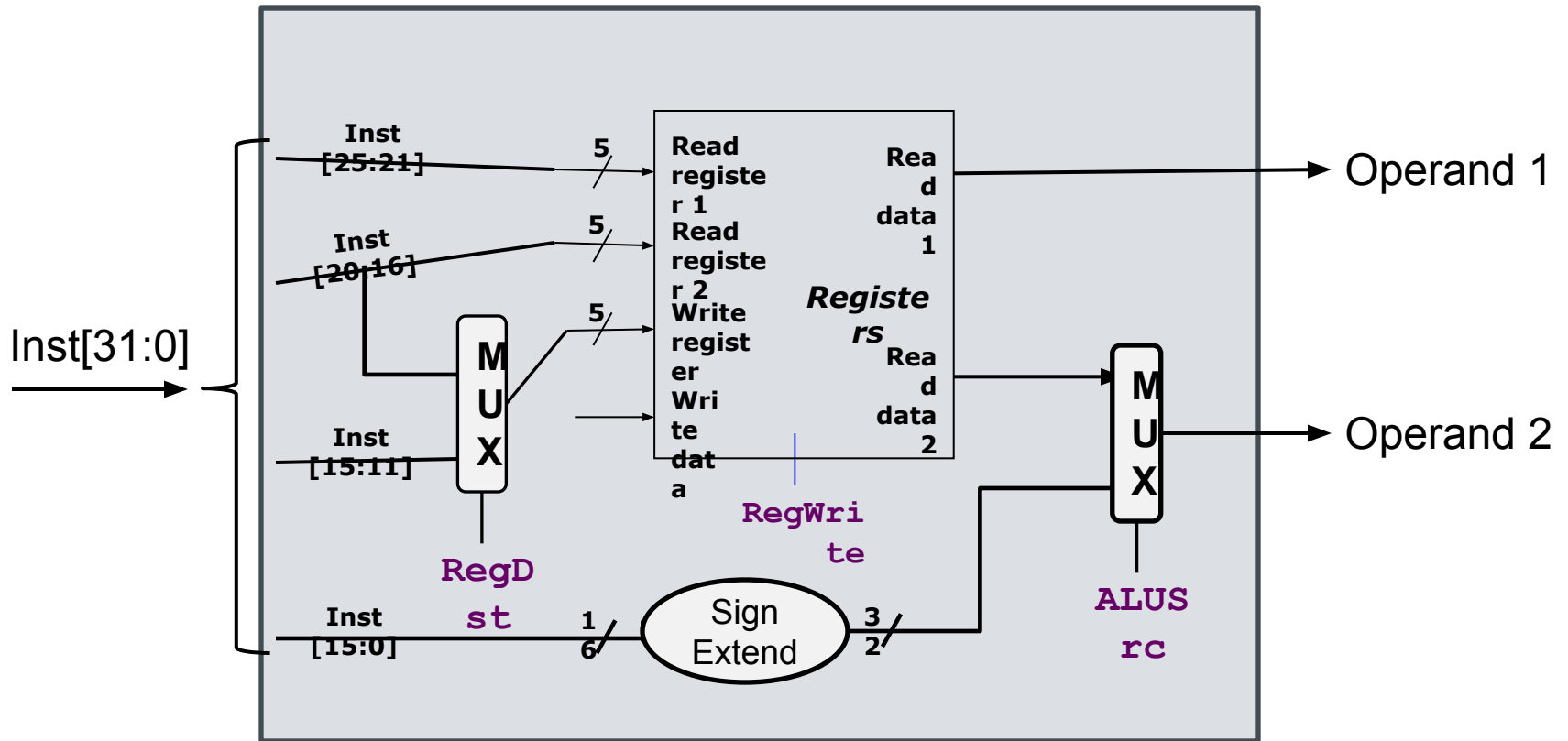
■ Ví dụ: "**beq** \$9, \$0, 3"

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





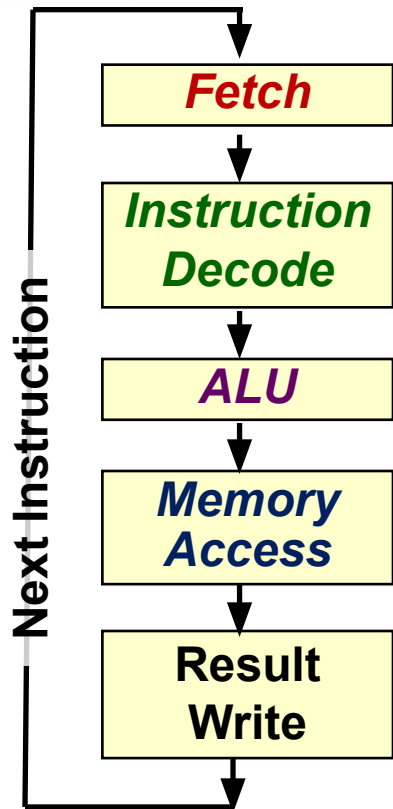
# Giải mã: tổng kết







# Quy trình thực thi lệnh của MIPS (5 công đoạn)



■ **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

## ■ 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**):

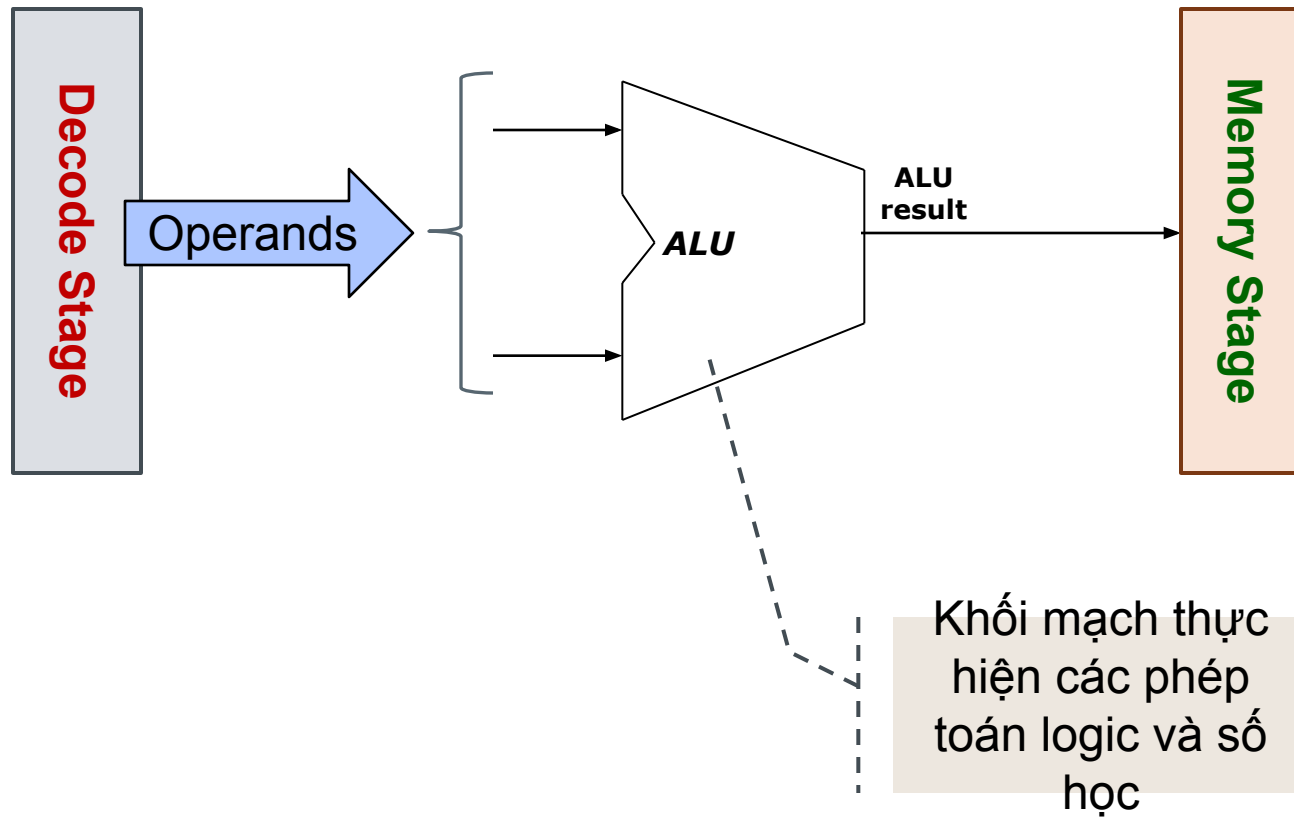
- ✓ Các thao tác (operation) và toán hạng (operand(s))

## ■ 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





# Khối *ALU* (*Arithmetic Logical Unit*)

## ■ **ALU (Arithmetic-logical unit)**

- ✓ Khối dùng để thực hiện các phép tính logic và số học

## ■ **Inputs:**

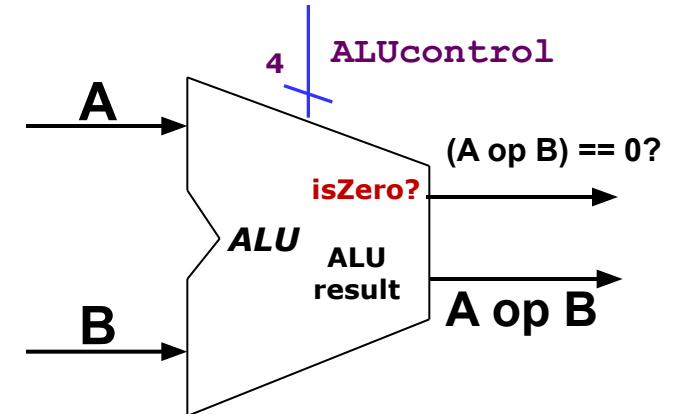
- ✓ 2 số 32-bit

## ■ **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



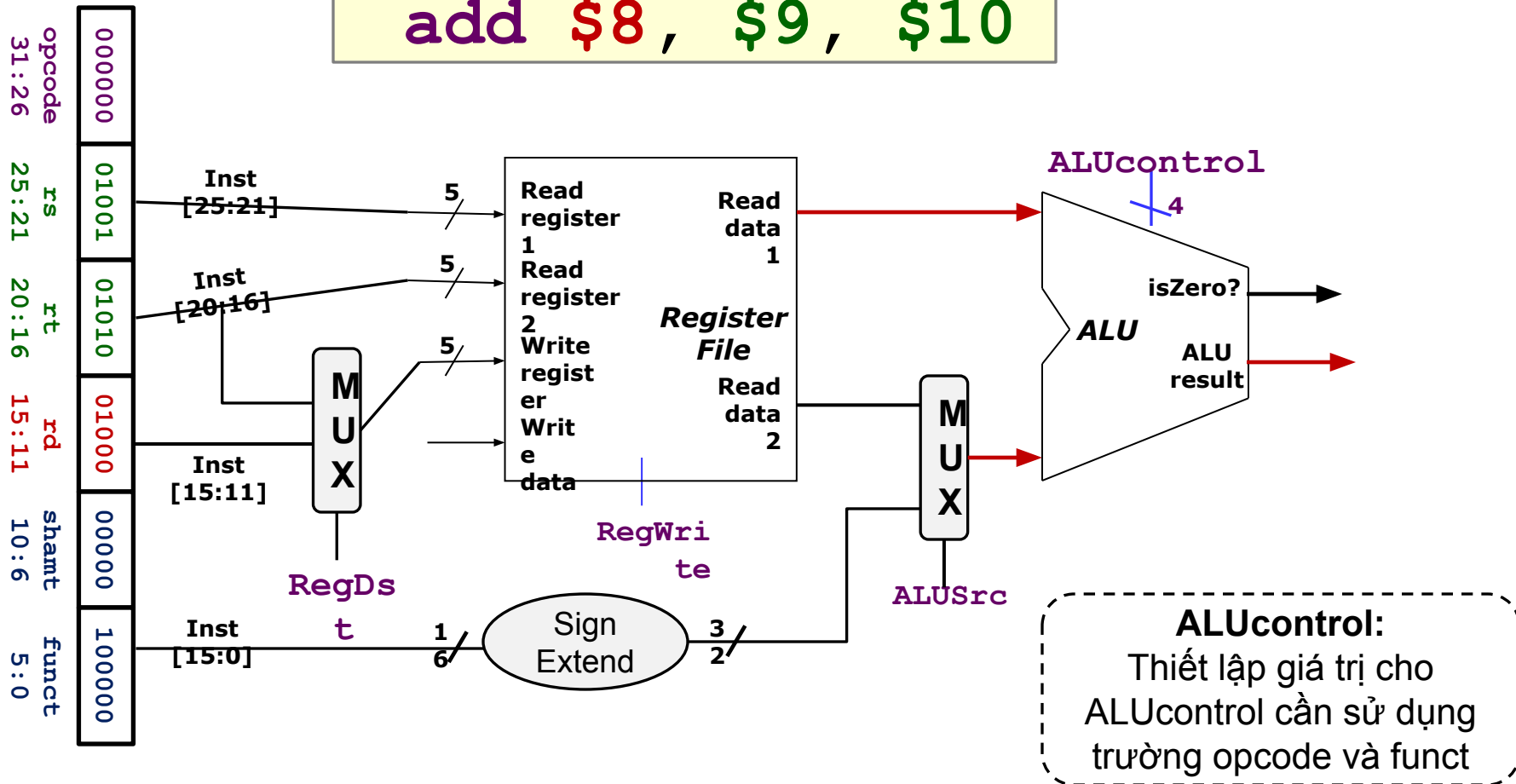
ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR



# Công đoạn ALU: các lệnh *non-branch*

- Các lệnh không nhánh/nhảy (non-branch) kết nối ALU như hình:

**add \$8, \$9, \$10**





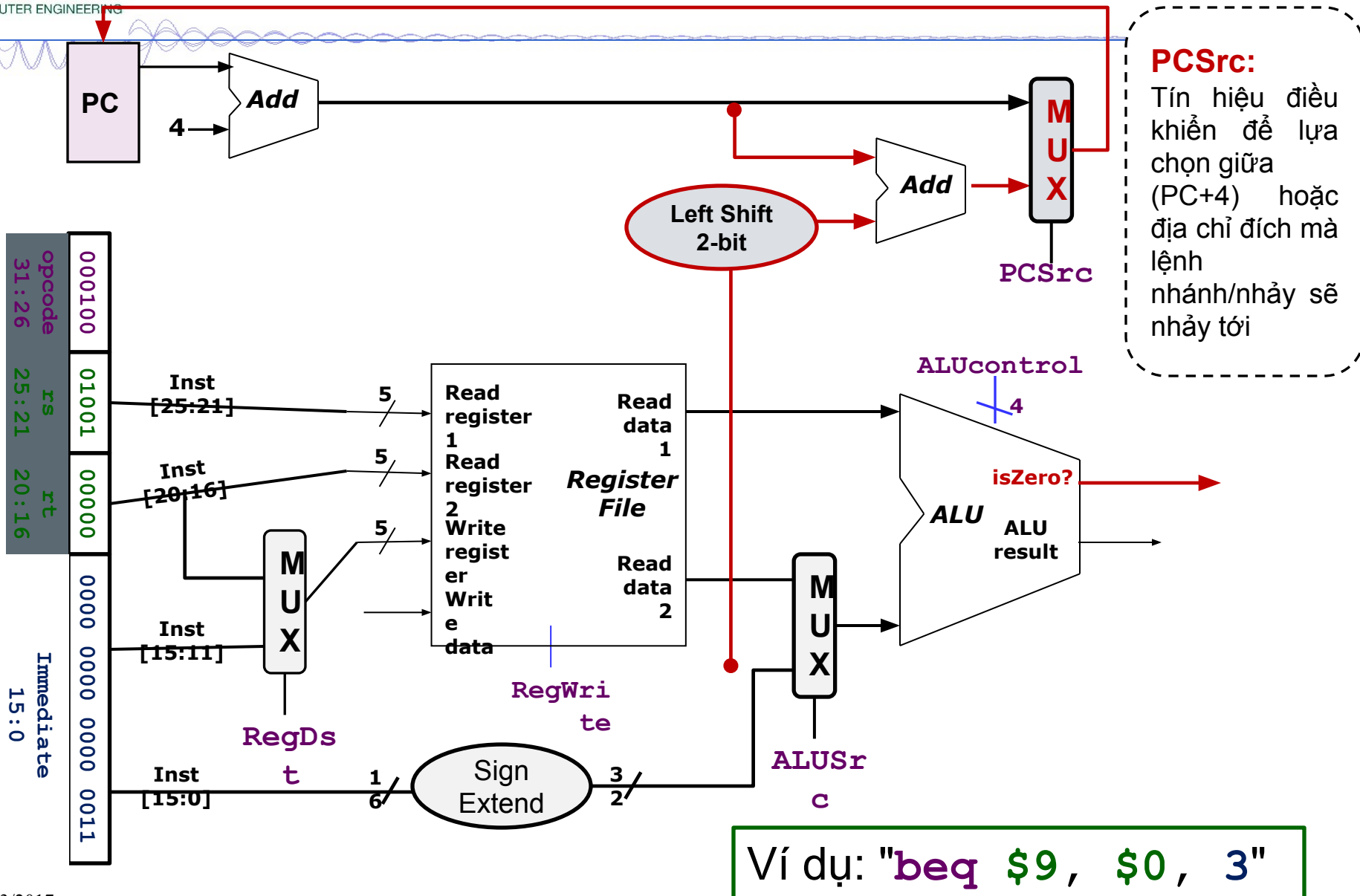
# Công đoạn ALU: Các lệnh *Branch*

- Lệnh rẽ nhánh thì khó hơn vì phải tính toán hai phép toán:
- Ví dụ: "**beq** \$9, \$0, 3"
  1. **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
  2. **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)



# Datapath với công đoạn ALU hoàn chỉnh

COMPUTER ENGINEERING

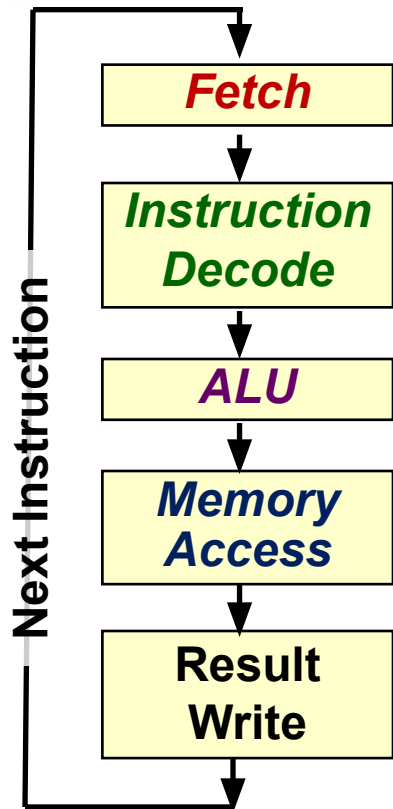


## PCSrc:

Tín hiệu điều khiển để lựa chọn giữa (PC+4) hoặc địa chỉ đích mà lệnh nhánh/nhảy sẽ nhảy tới



# Quy trình thực thi lệnh của MIPS (5 công đoạn)



■ **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):

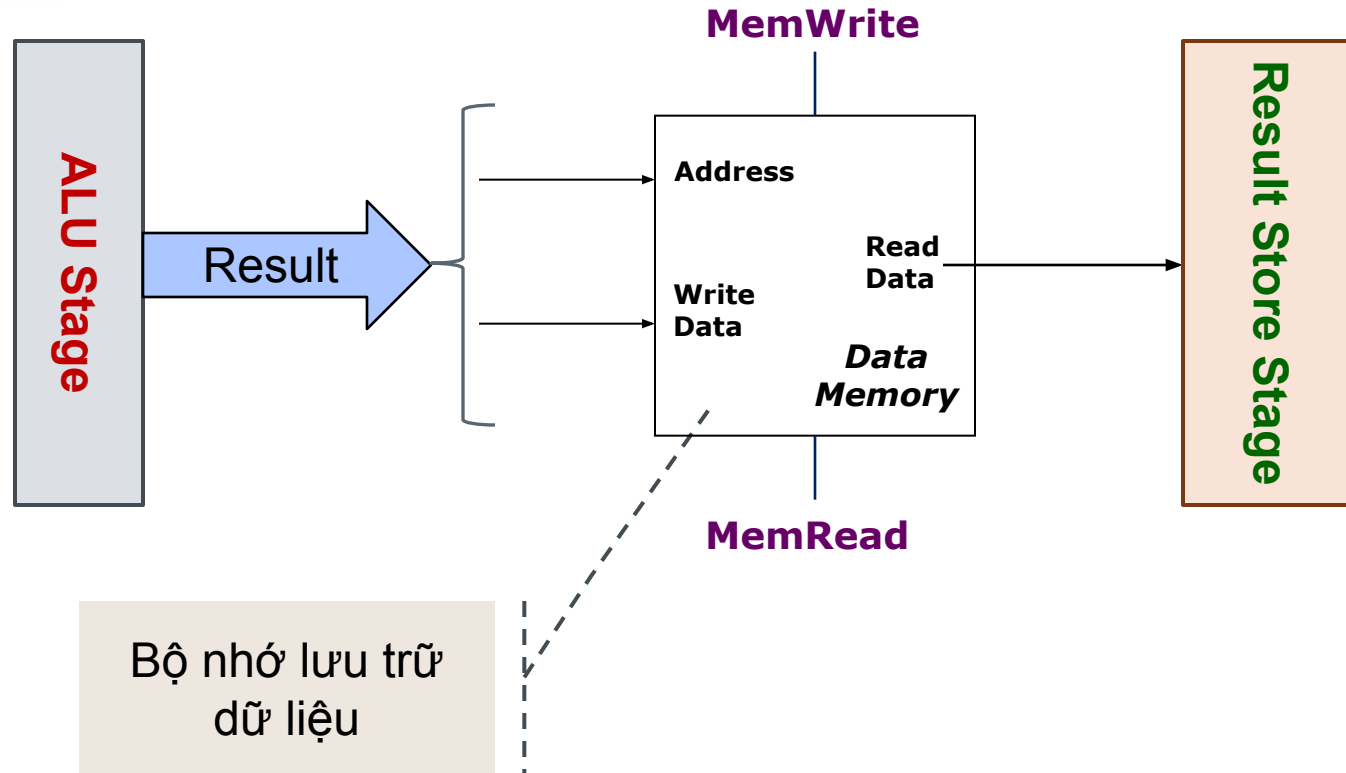
- ✓ Kết quả tính toán được dùng làm địa chỉ vùng nhớ (nếu có thể ứng dụng)

## ■ Đầu ra cho giai đoạn tiếp theo (**Result Write**):

- ✓ Kết quả được lưu trữ lại (nếu cần)



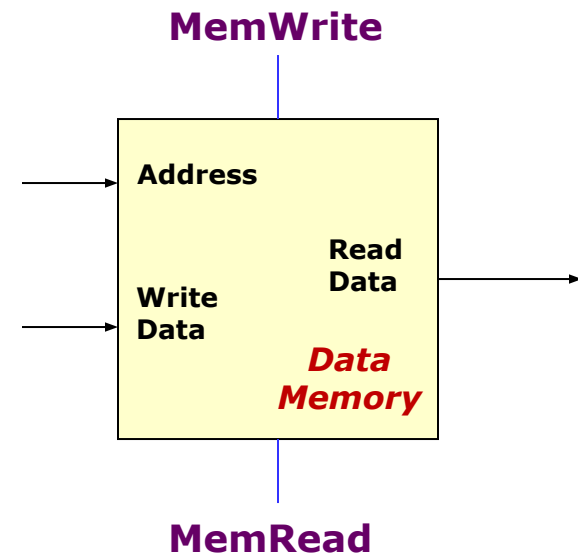
# Giai đoạn truy xuất vùng nhớ (Memory stage)





# 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 kì một thời điểm nào
- **Output:**
  - ✓ Dữ liệu được đọc từ vùng nhớ đối với lệnh Load

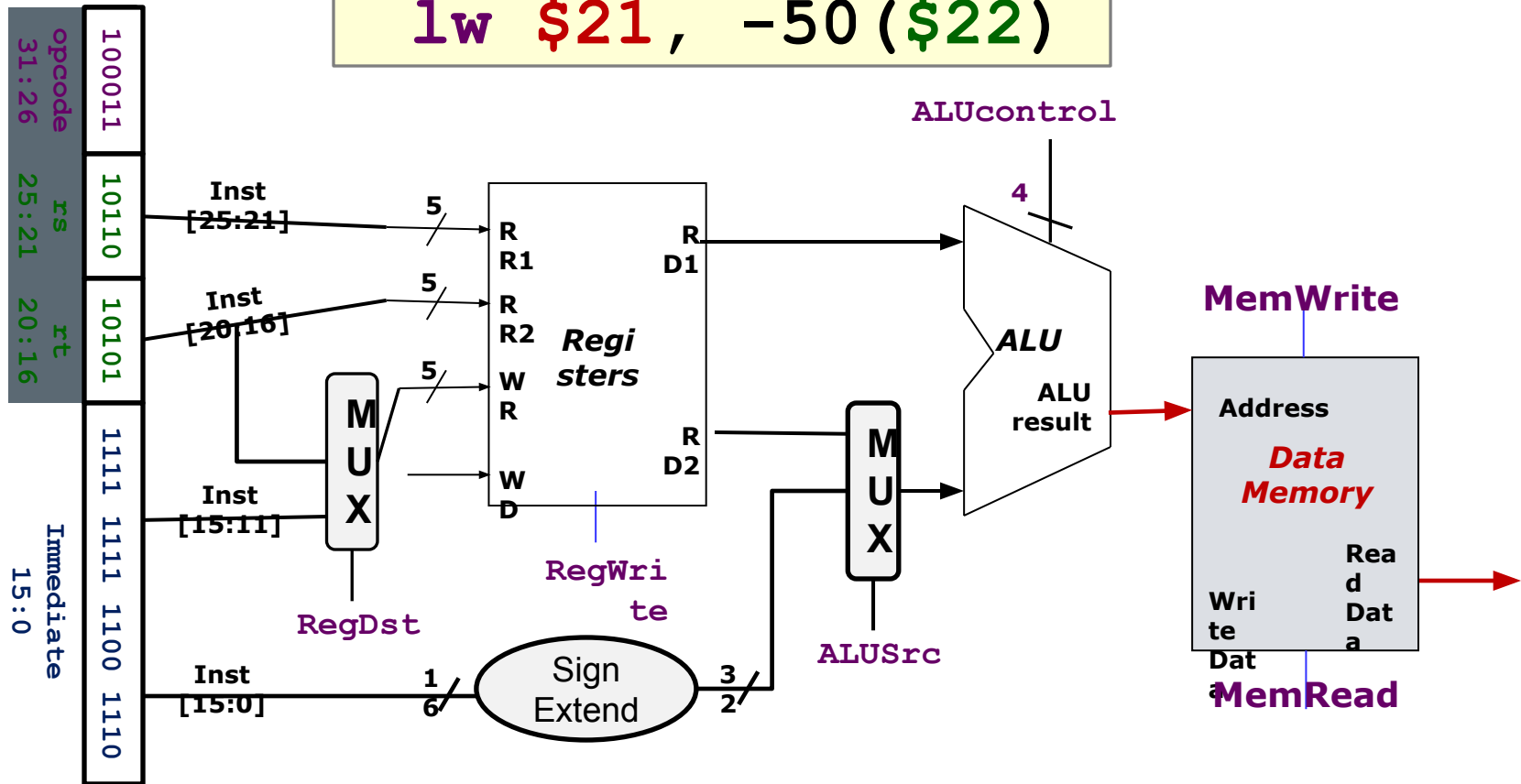




# Giai đoạn Memory: lệnh *Load*

- Chỉ những phần liên quan đến Decode & ALU Stage được trình bày

**lw \$21, -50(\$22)**

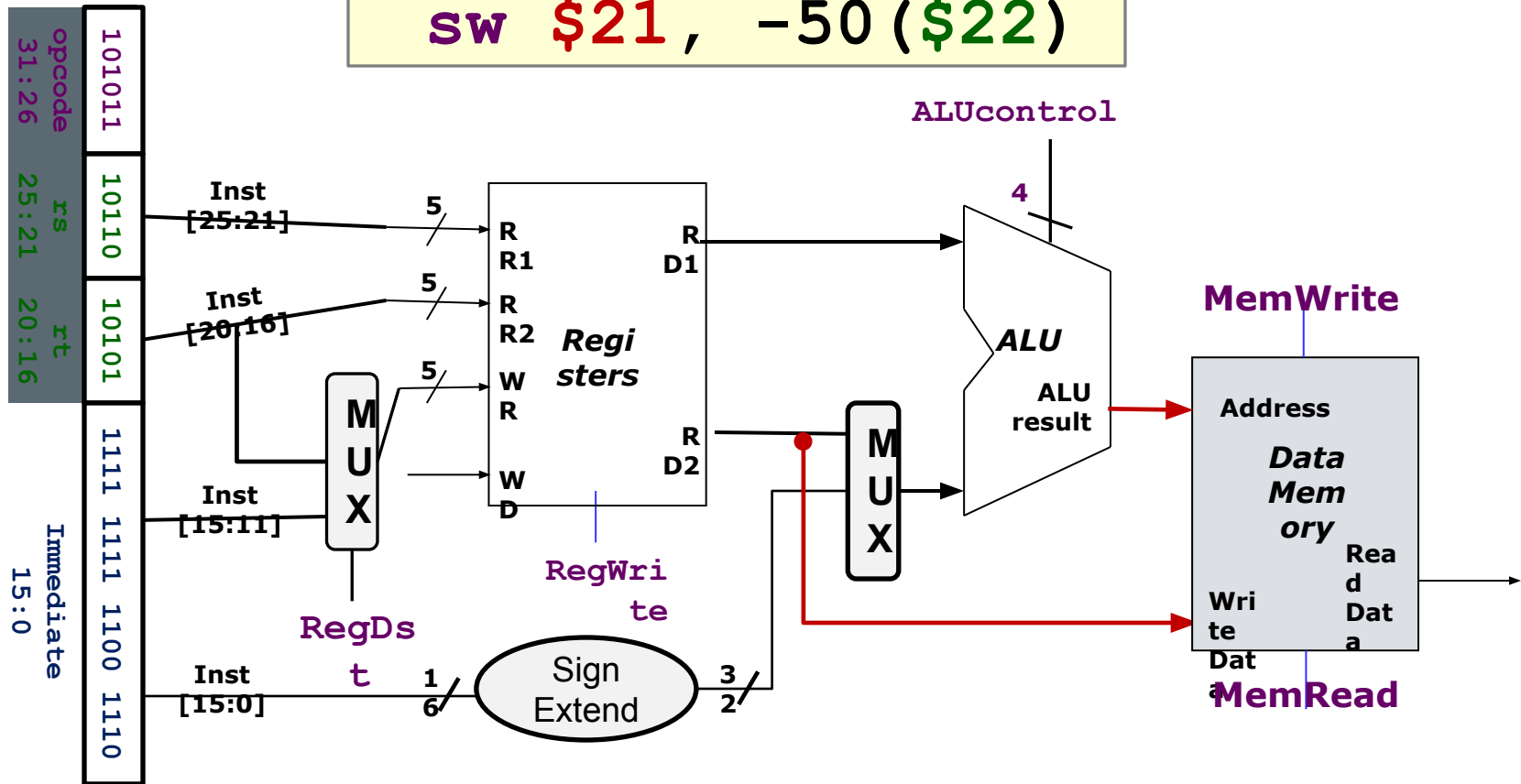




# Giai đoạn Memory: lệnh *Store*

- Cần *Read Data 2* (Decode) để đưa vào *Write Data*

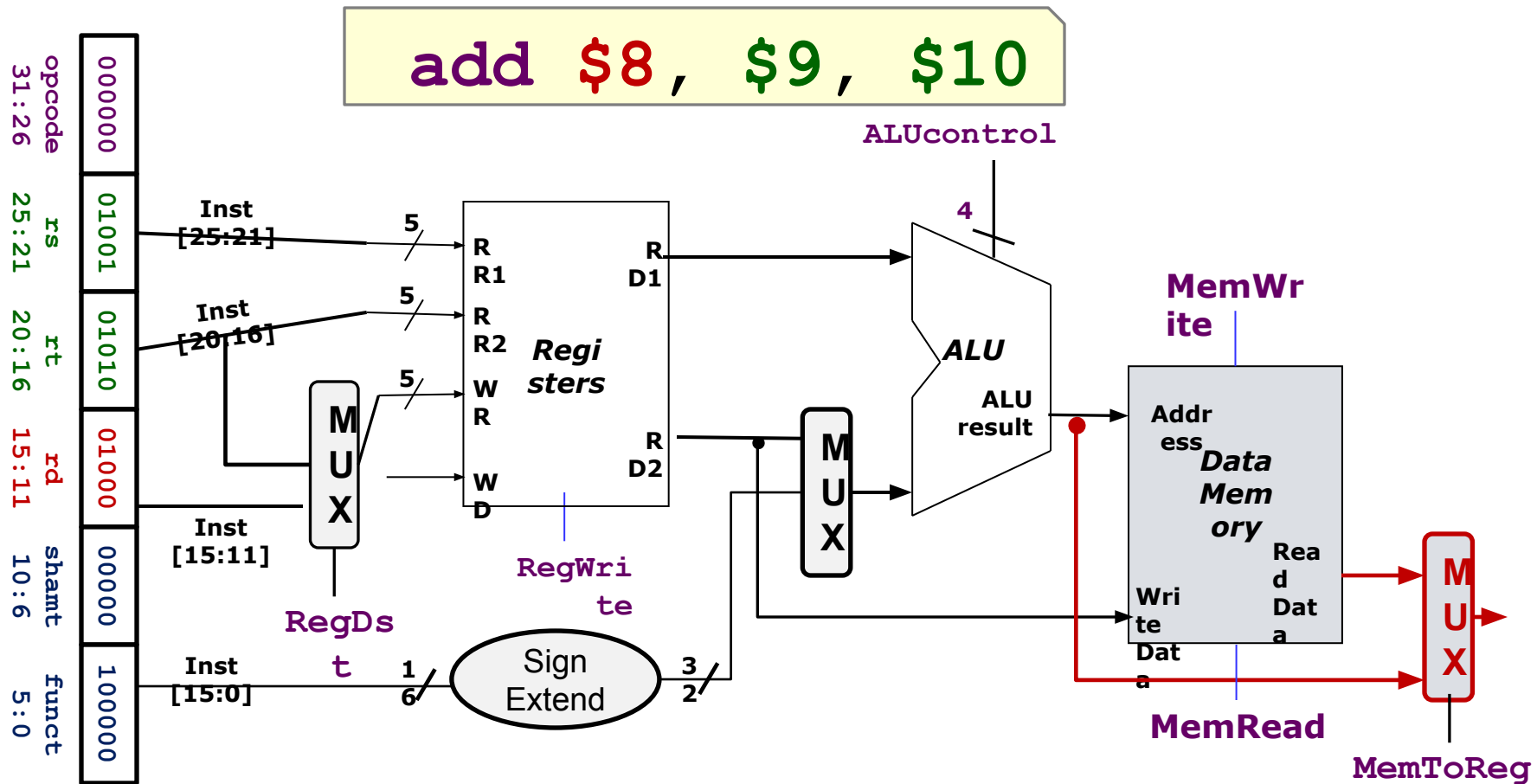
**sw \$21, -50(\$22)**





# Giai đoạn Memory: lệnh không truy xuất vùng nhớ

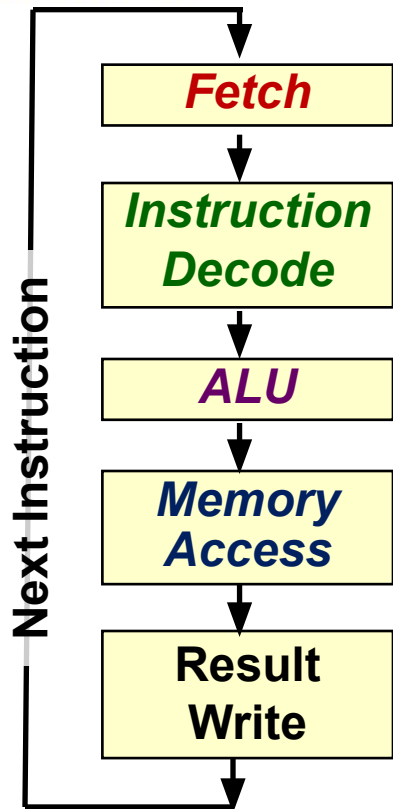
- Sử dụng thêm một multiplexer để lựa chọn kết quả lưu trữ vào thanh ghi



**MemToReg:**  
Tín hiệu điều khiển giúp lựa chọn giá trị lưu vào thanh ghi là từ Read Data hay từ ALU.result



# Quy trình thực thi lệnh của MIPS (5 công đoạn)



■ **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 lưu trữ kết quả (Result Write)

## ■ 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

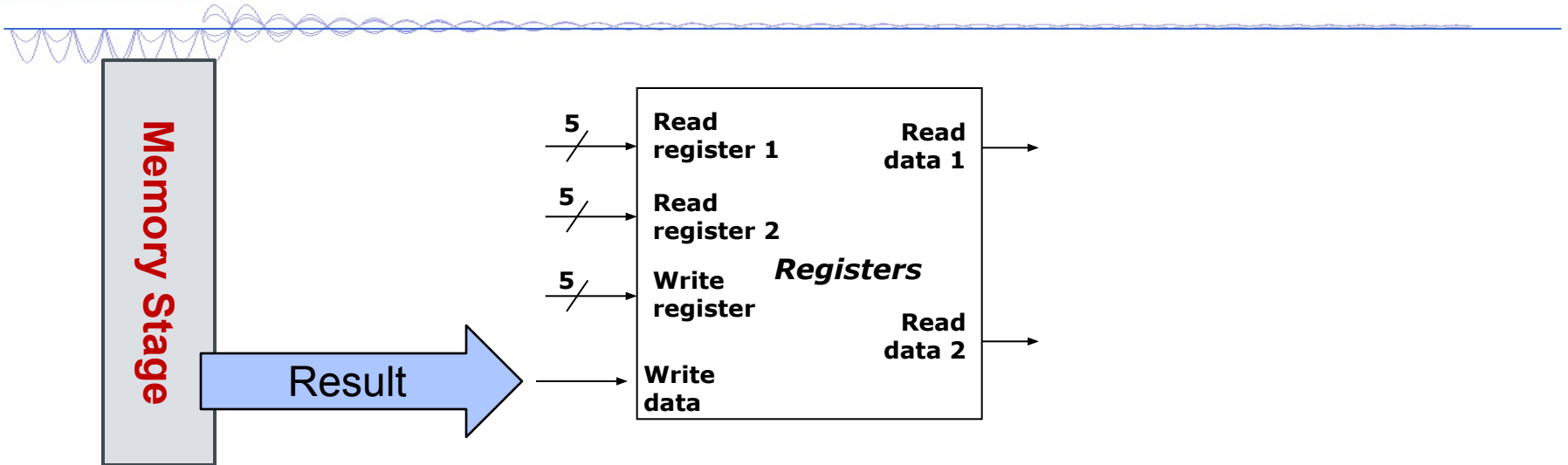
## ■ Đầu vào từ giai đoạn trước (**Memory**):

- ✓ Kết quả tính toán hoặc là từ Memory hoặc là từ ALU





# Giai đoạn lưu trữ kết quả (Result Write)

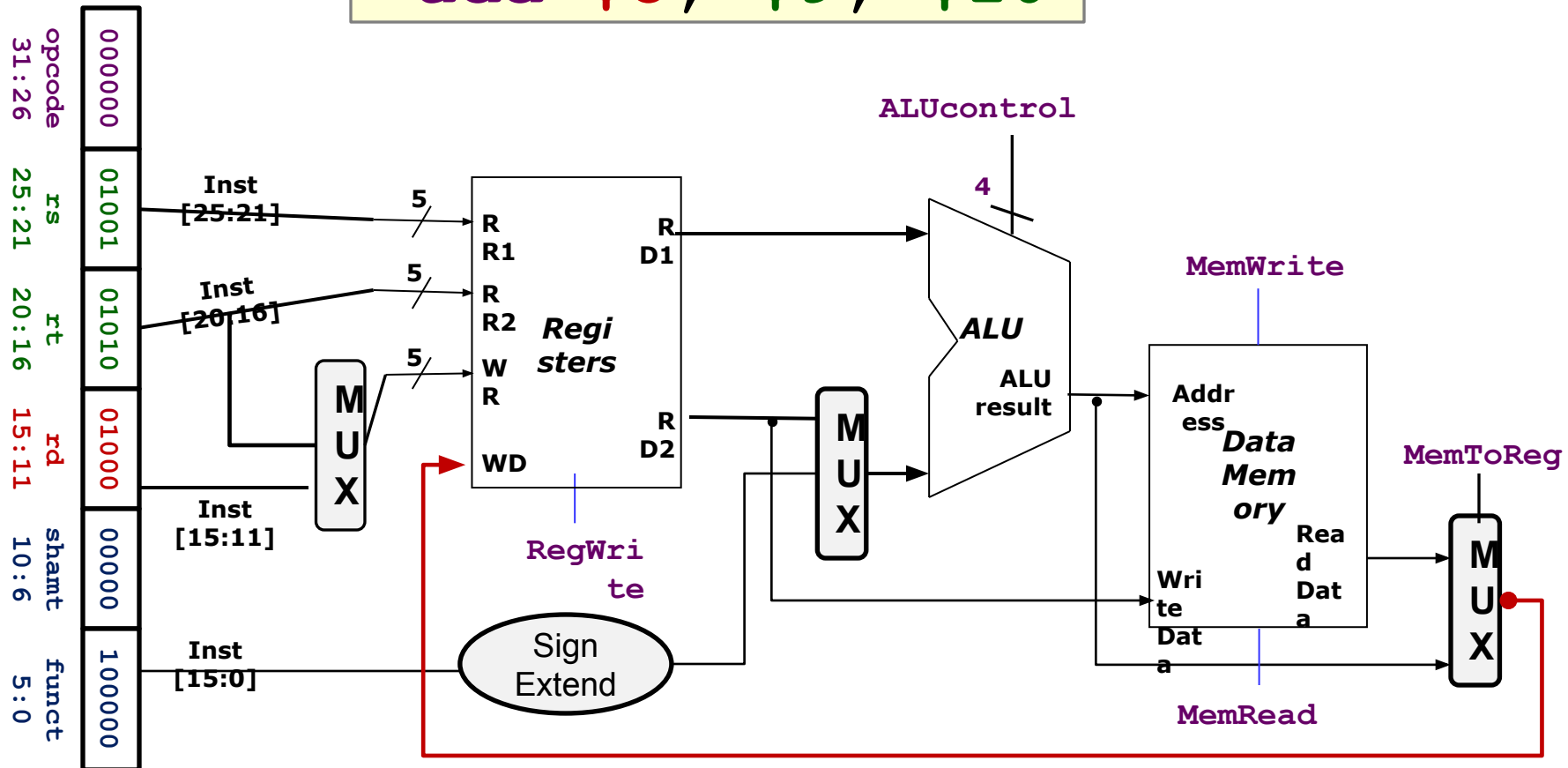


- Công đoạn Result Write không có thêm bất kỳ thành phần nào khác:
  - ✓ Chỉ đơn giản đưa kết quả vào thanh ghi (ngõ Write data của khối Registers/Register file)
  - ✓ Chỉ số của thanh ghi được ghi vào (ngõ vào *Write Register*) được sinh ra trong giai đoạn **Decode Stage**



# Giai đoạn lưu trữ kết quả (Result Write)

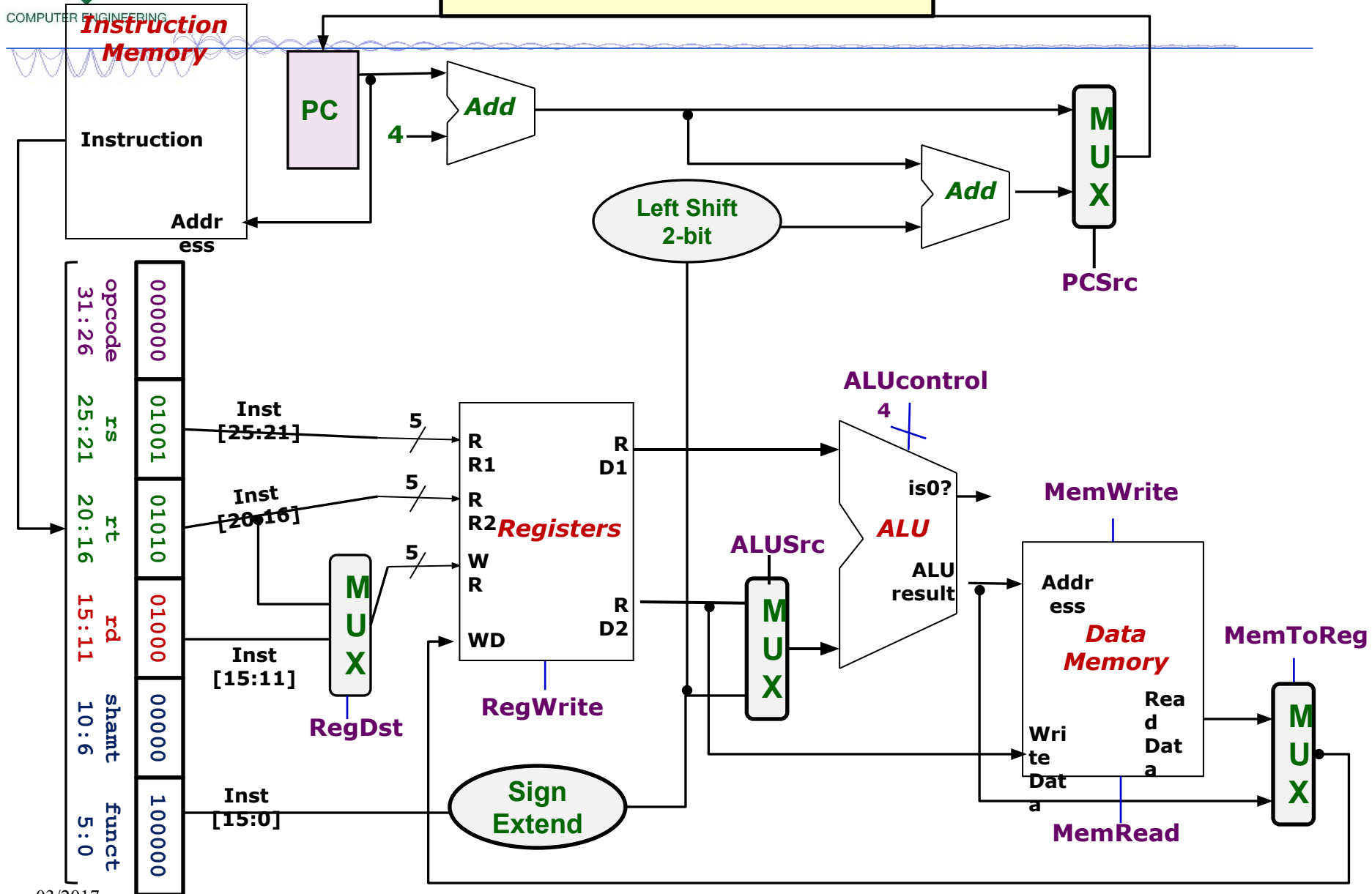
**add \$8, \$9, \$10**



(Chưa có khối Control để điều khiển các tín hiệu điều khiển)



COMPUTER ENGINEERING





## Tổng kết:

Phần này trình bày một cách thiết kế datapath đơn giản cho bộ xử lý 32 bits, với 8 lệnh cơ bản của MIPS:

- add, sub, and, or, slt
- lw, sw
- beq

Với khối chức năng cơ bản trong một bộ xử lý (tập thanh ghi, khối ALU, khối Control, thanh ghi PC, thanh ghi IR) và bộ nhớ chính, các khối này sẽ được kết nối với nhau để đảm bảo thực thi đúng 8 lệnh như trên.



- ◆ **Lý thuyết: Đọc sách tham khảo**
  - Mục: 4.1, 4.2, 4.3
  - Sách: *Computer Organization and Design: The Hardware/Software Interface*, Patterson, D. A., and J. L. Hennessy, Morgan Kaufman, Revised Fourth Edition, 2011.
  
- ◆ **Bài tập: file đính kèm**