Tuần 9-10

BỘ XỬ LÝ
PROCESSOR

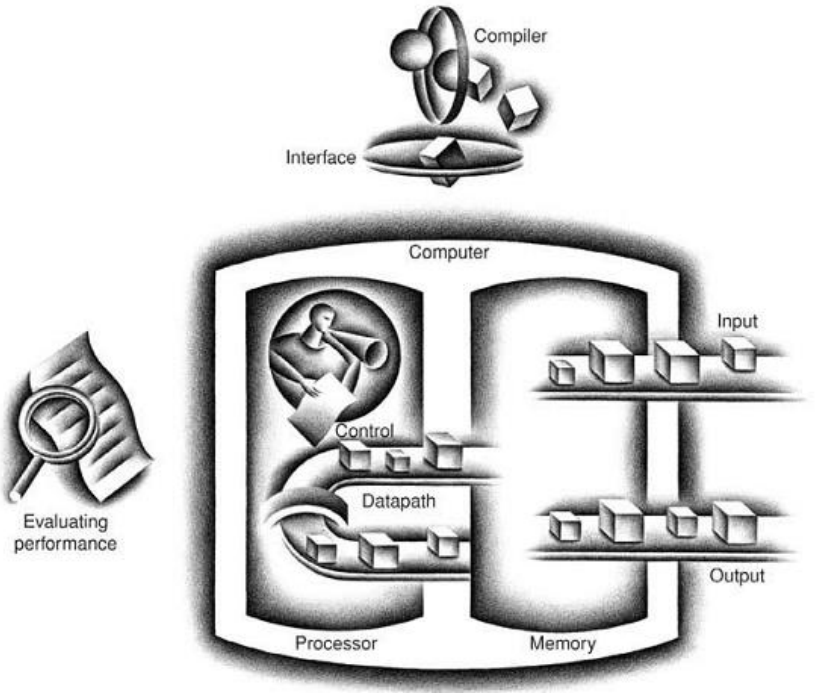
BỘ XỬ LÝ – Ôn tập kiến thức

1. **Tổng quan về Kiến trúc Máy tính**
2. **Hiệu suất máy tính**
3. **Kiến trúc bộ lệnh**
4. **Thực hiện các phép toán trên máy tính**

BỘ XỬ LÝ

Mục đích:

- ✓ Hiểu cơ chế thực thi lệnh
- ✓ Hướng dẫn thiết kế Datapath với 8 lệnh cơ bản cho một bộ xử lý và cách hiện thực thiết kế này.



Hình 1: Kiến trúc của 1 máy tính

BỘ XỬ LÝ

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**

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**

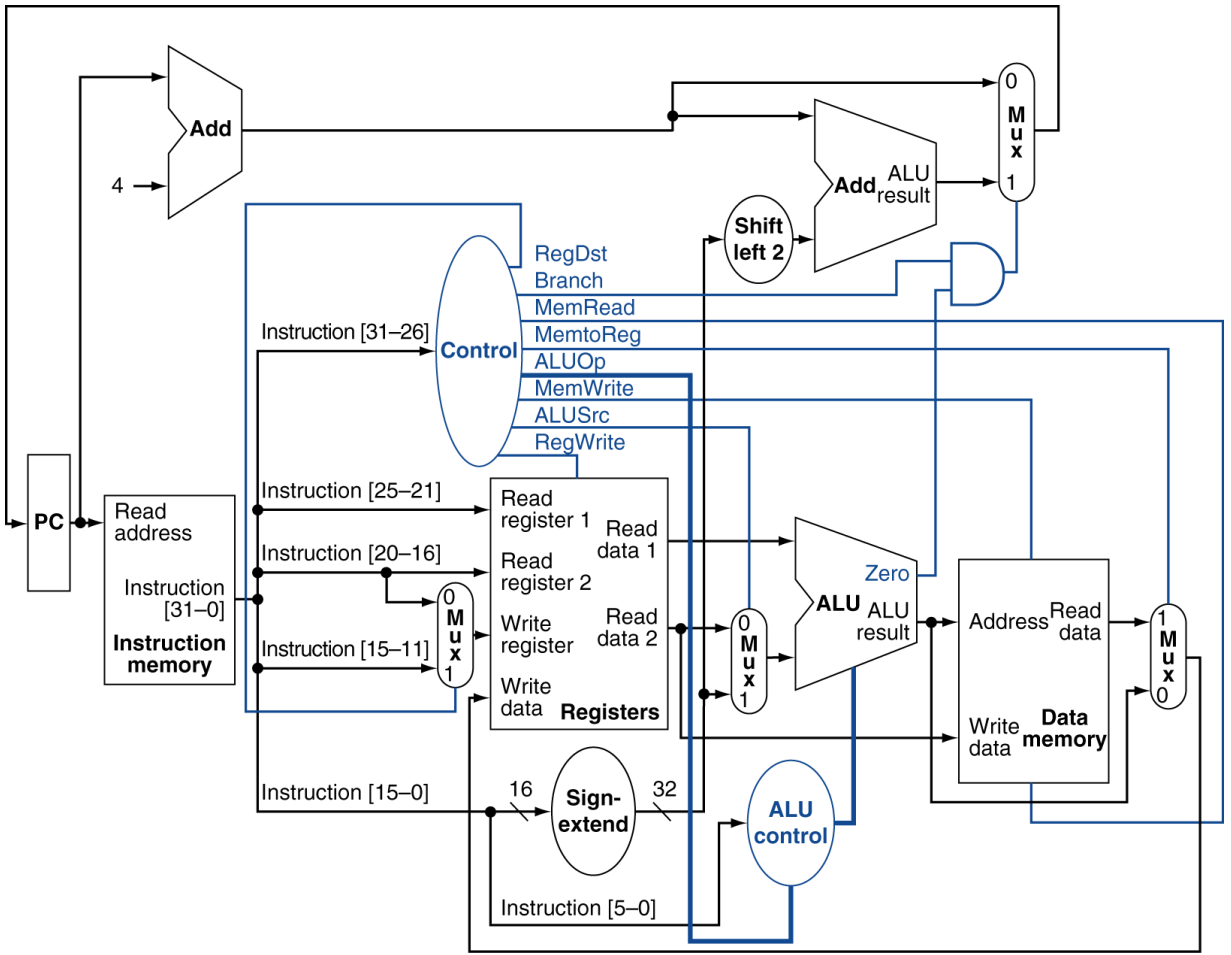
Giới thiệu

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**)

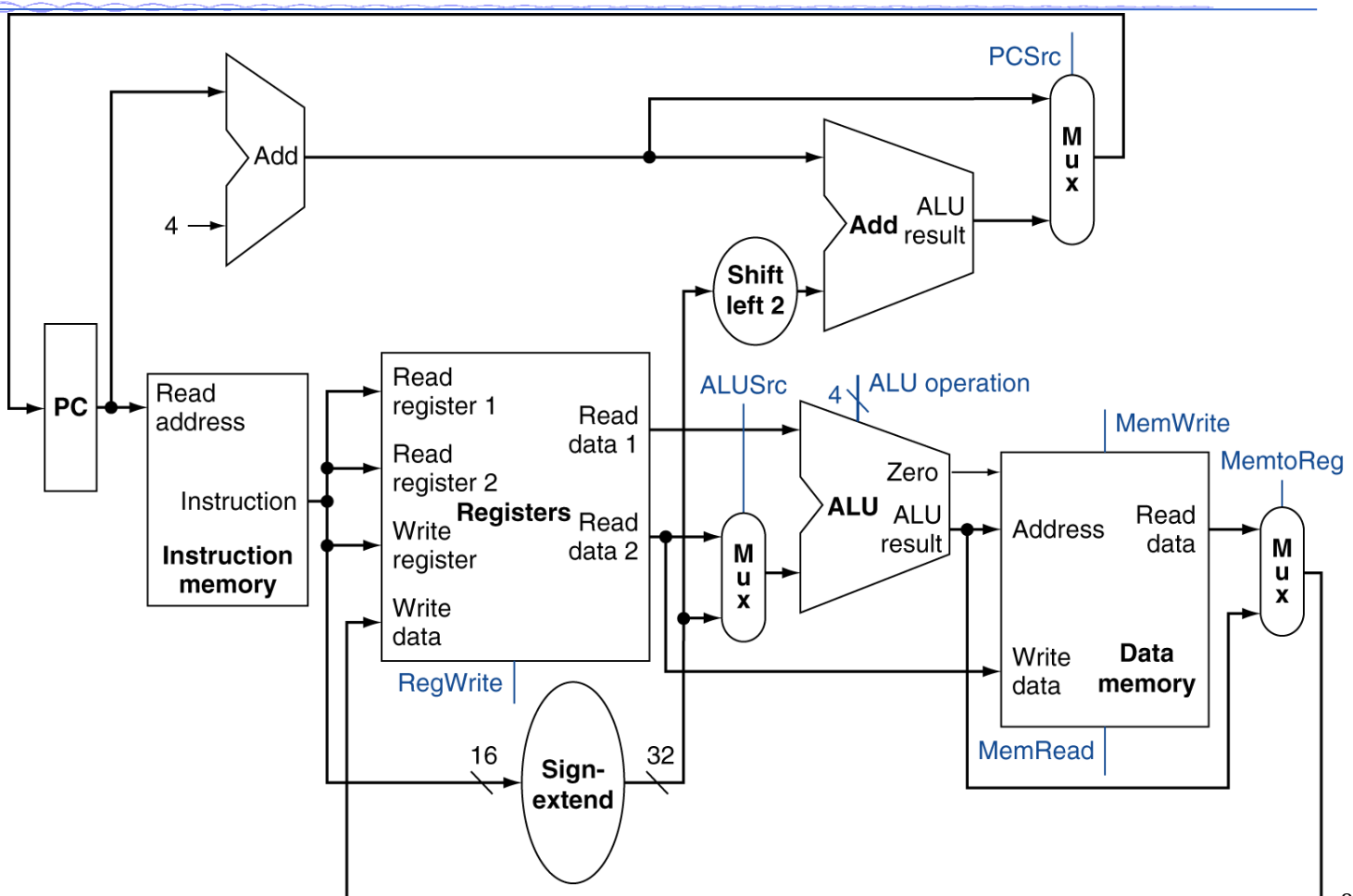
Giới thiệu

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



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/ADD/MUX**
- **Mạch tuần tự (Sequential): instruction/data memories và thanh ghi**
- **Tín hiệu điều khiển (Control signal)**
- **Tín hiệu dữ liệu (Data signal)**
- **Bus**

BỘ XỬ LÝ

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

Tổng quan các lệnh cần xem xét:

Nhóm lệnh tham khảo bộ nhớ (lw và sw): lw $s1, 20($s2)

Nạp lệnh → Đọc một thanh ghi → Sử dụng ALU → Truy xuất bộ nhớ để đọc/ghi dữ liệu → Ghi dữ liệu vào thanh ghi

Nhóm lệnh logic và số học (add, sub, AND, OR, và slt) : add $t0, $s1, $s2

Nạp lệnh → Đọc hai thanh ghi → Sử dụng ALU → Ghi dữ liệu vào thanh ghi

Nhóm lệnh rẽ nhánh (beq): beq $s1, $s2, Exit (PC = Exit: hoặc PC = PC + 4)

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

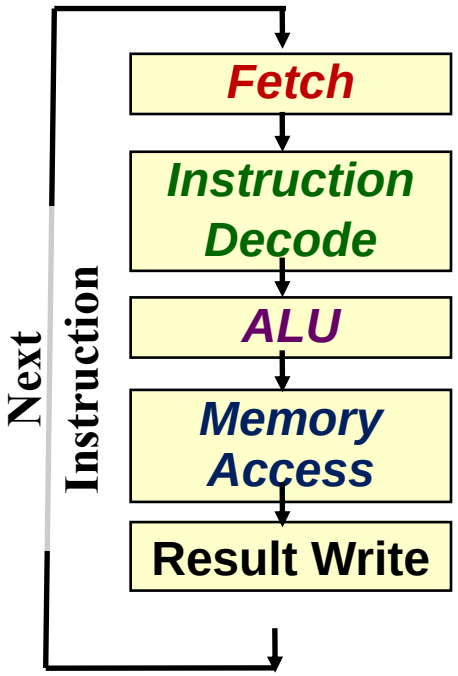
COMPUTER ENGINEERING

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

	add $3, $1, $2	**lw $3, 20($1)**	**beq $1, $2, label**
Fetch	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	○ Đọc thanh ghi $1, xem như toán hạng *opr1* ○ Đọc thanh ghi $2, xem như toán hạng *opr2*	○ Đọc thanh ghi $1, xem như toán hạng *opr1* ○ Sử dụng ***20*** như toán hạng *opr2*	○ Đọc thanh ghi $1, xem như toán hạng *opr1* ○ Đọc thanh ghi $2, xem như toán hạng *opr2*
ALU	*Result* = *opr1* + *opr2*	*MemAddr* = opr1 + opr2	*Taken* = (*opr1* == *opr2*)? *Target* = **PC** + Label*
Memory Access		Sử dụng *MemAddr* để đọc dữ liệu từ bộ nhớ	
Result Write	*Result* được lưu trữ vào **$3**	*Dữ liệu của từ nhớ có địa chỉ MemAddr được được lưu trữ vào* **$3**	if (*Taken*) **PC** = *Target*

COMPUTER ENGINEERING

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



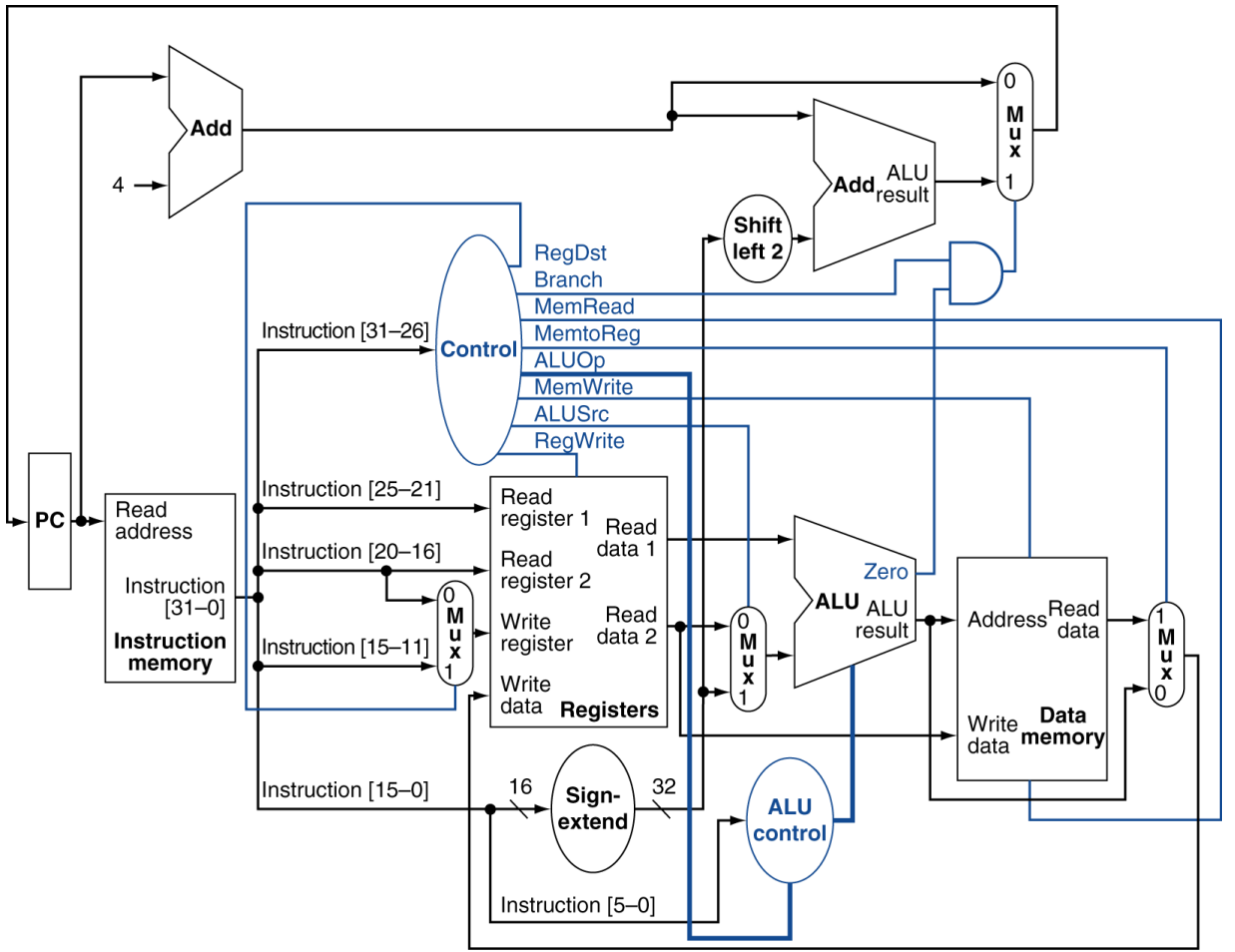
- **Instruction Fetch (Nạp lệnh)**
- **Instruction Decode & Operand Fetch**

(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)**

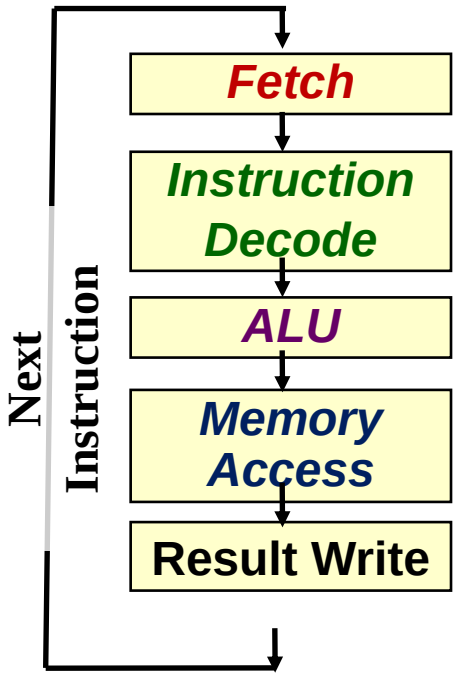
COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



COMPUTER ENGINEERING

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")

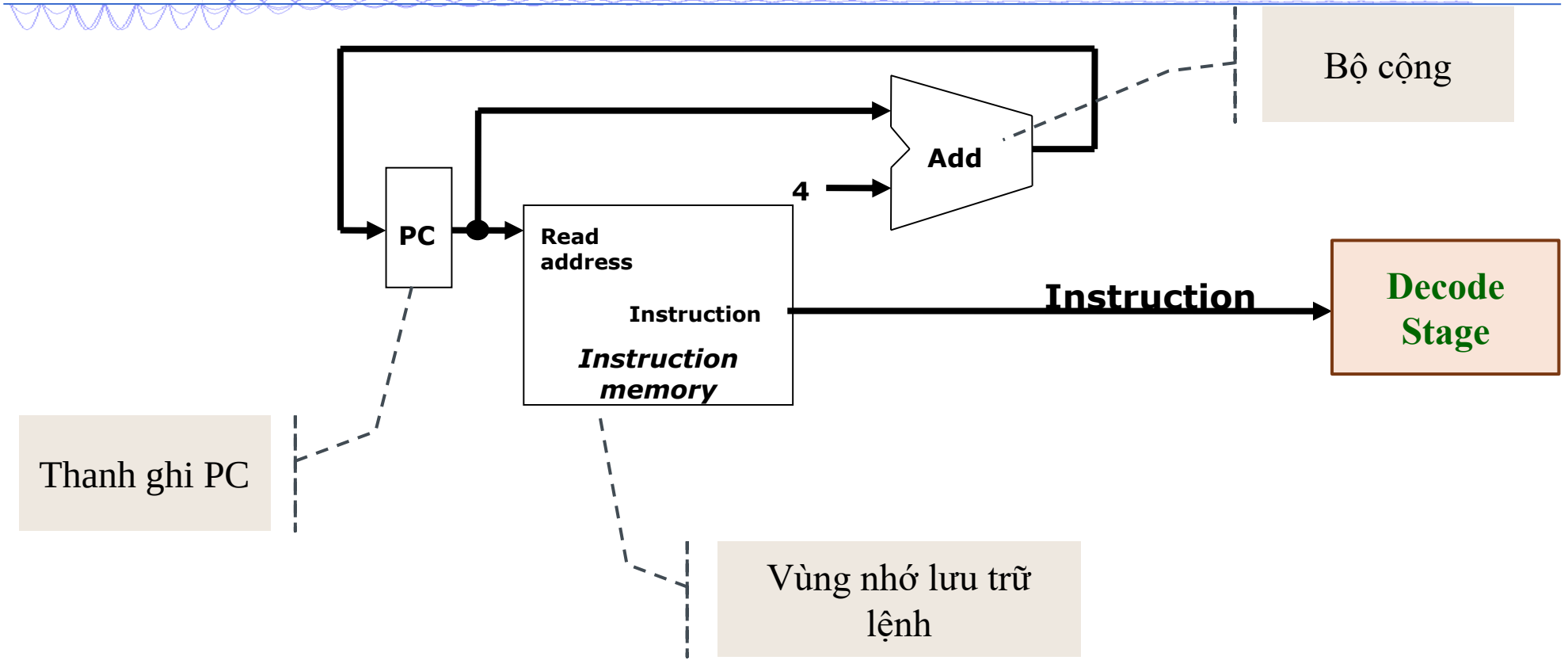
- **ALU (Giai đoạn sử dụng ALU hay giai đoạn thực thi)**
- **Memory Access (Giai đoạn truy xuất vùng nhớ)**
- **Result Write (Giai đoạn ghi lại kết quả/lưu trữ)**

Giai đoạn tìm nạp lệnh (Instruction Fetch)

- Giai đoạn nạp lệnh:
  1. Sử dụng thanh ghi **P**rogram **C**ounter (**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
     - 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**):

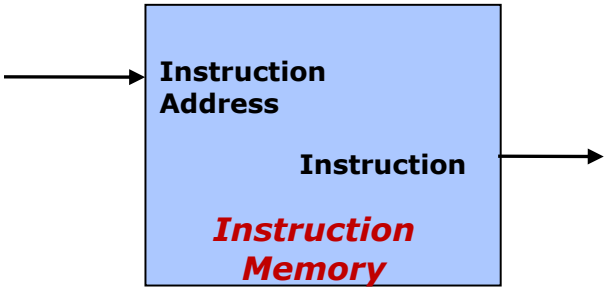
  Kết quả của giai đoạn này là 32 bit mã máy của lệnh cần thực thi. Chuỗi 32 bits này sẽ sử dụng như đầu vào (input) cho giai đoạn tiếp theo là

Giai đoạn tìm nạp lệnh (Instruction Fetch)

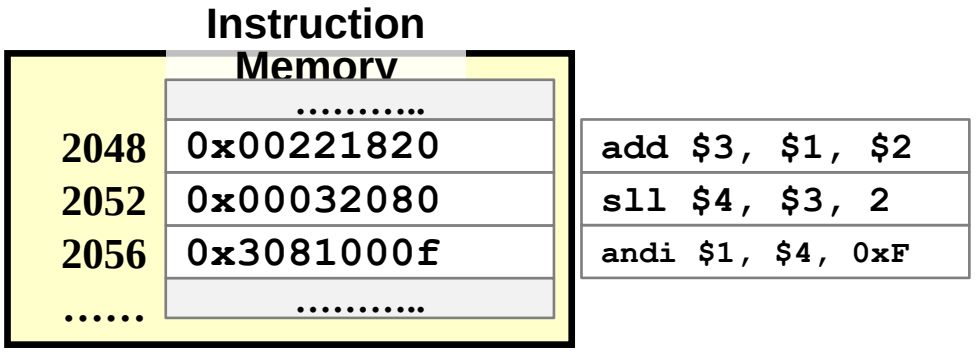


Khối *Instruction Memory*

- 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



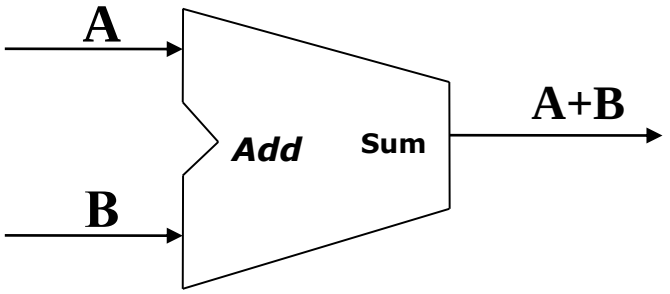
Cách sắp xếp của bộ nhớ giống như hình bên phải



Bộ cộng

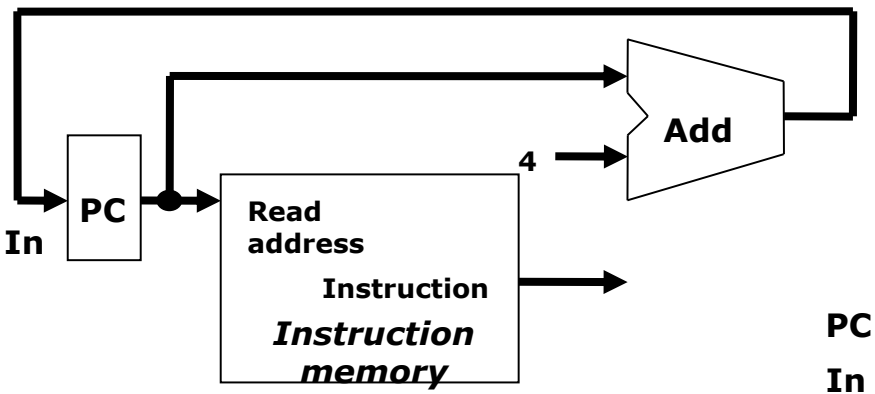
- Mạch logic kết hợp để cộng 2 số - bộ cộng

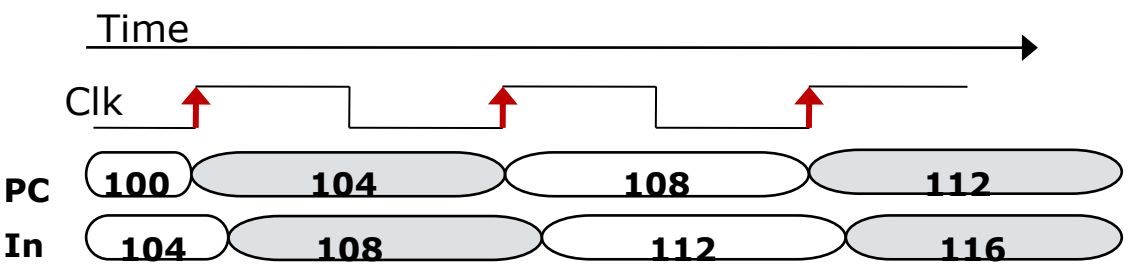
- **Đầu vào:**
  - Hai số 32-bit **A**, **B**
- **Đầu ra:**
  - **A + B**



Ý niệm về việc sử dụng xung clock

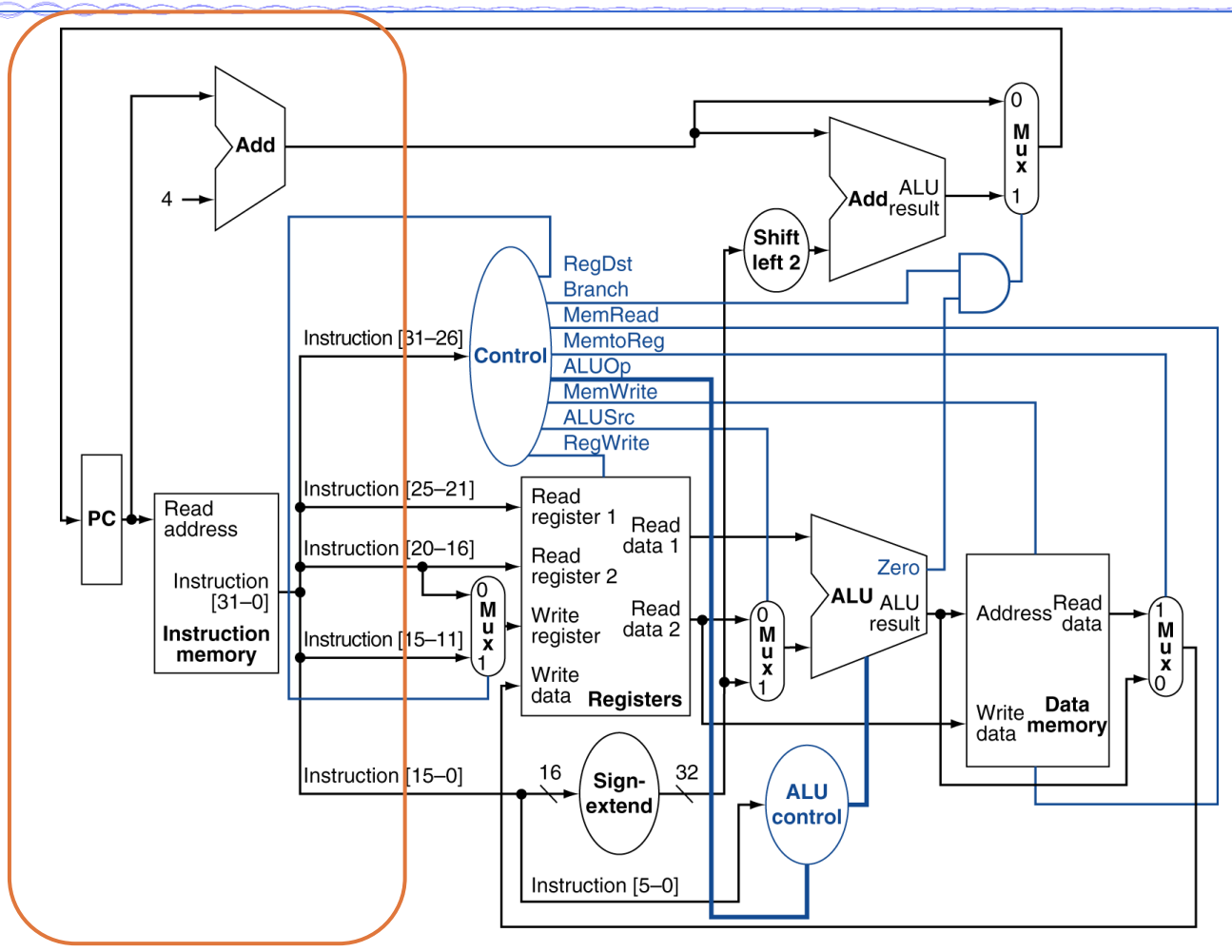
- Dường như thanh ghi PC được đọc và cập nhật cùng lúc:
  - PC hoạt động chính xác như thế nào?
- **Magic of clock**:
  - PC được đọc trong nửa clock đầu và cập nhật thành PC+4 trong lần **kích cạnh lên tiếp**



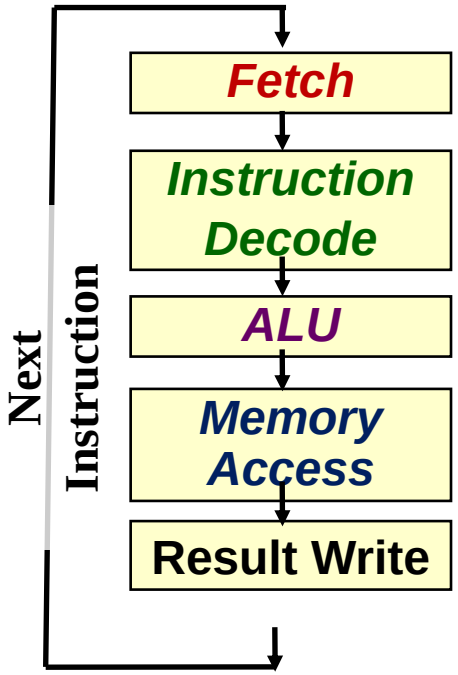


COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



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`**)

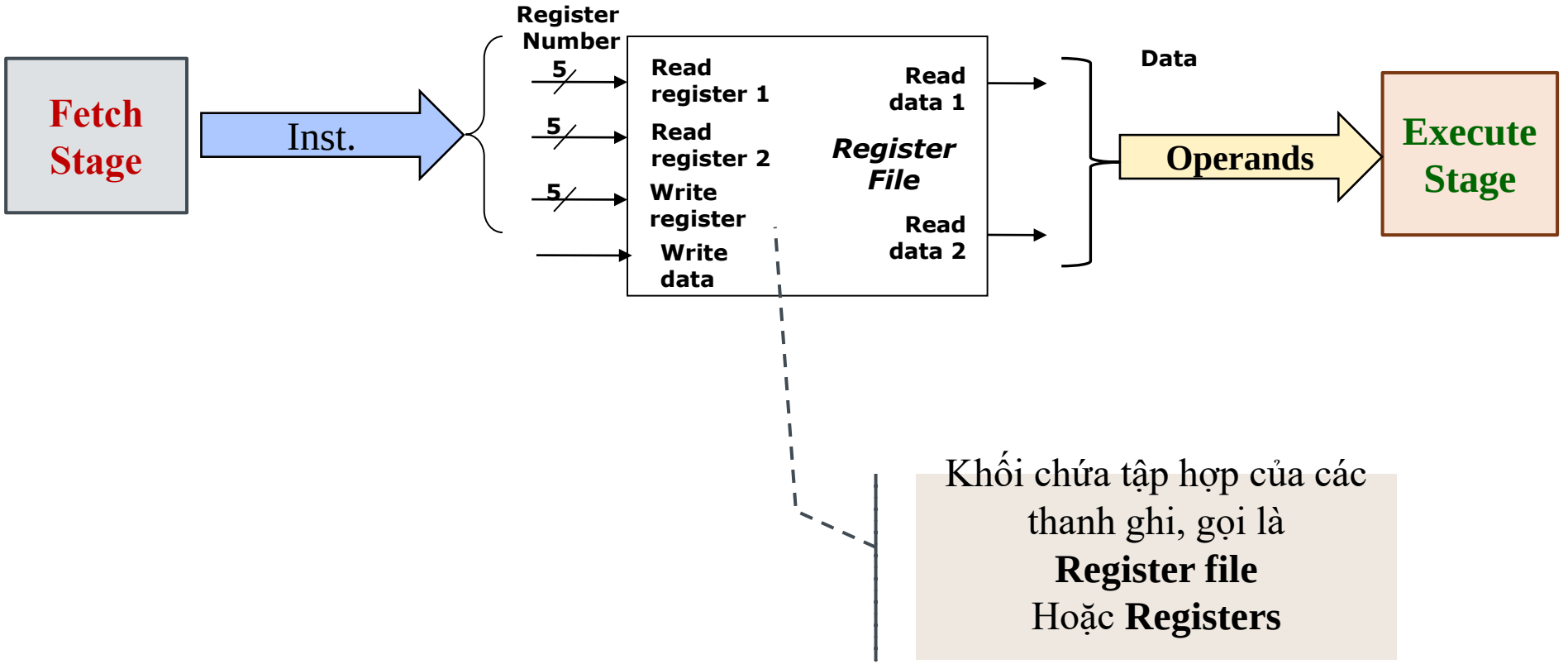
- Đầu vào từ giai đoạn trước (**Fetch**):

  Lệnh cần được thực thi (Mã máy)

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

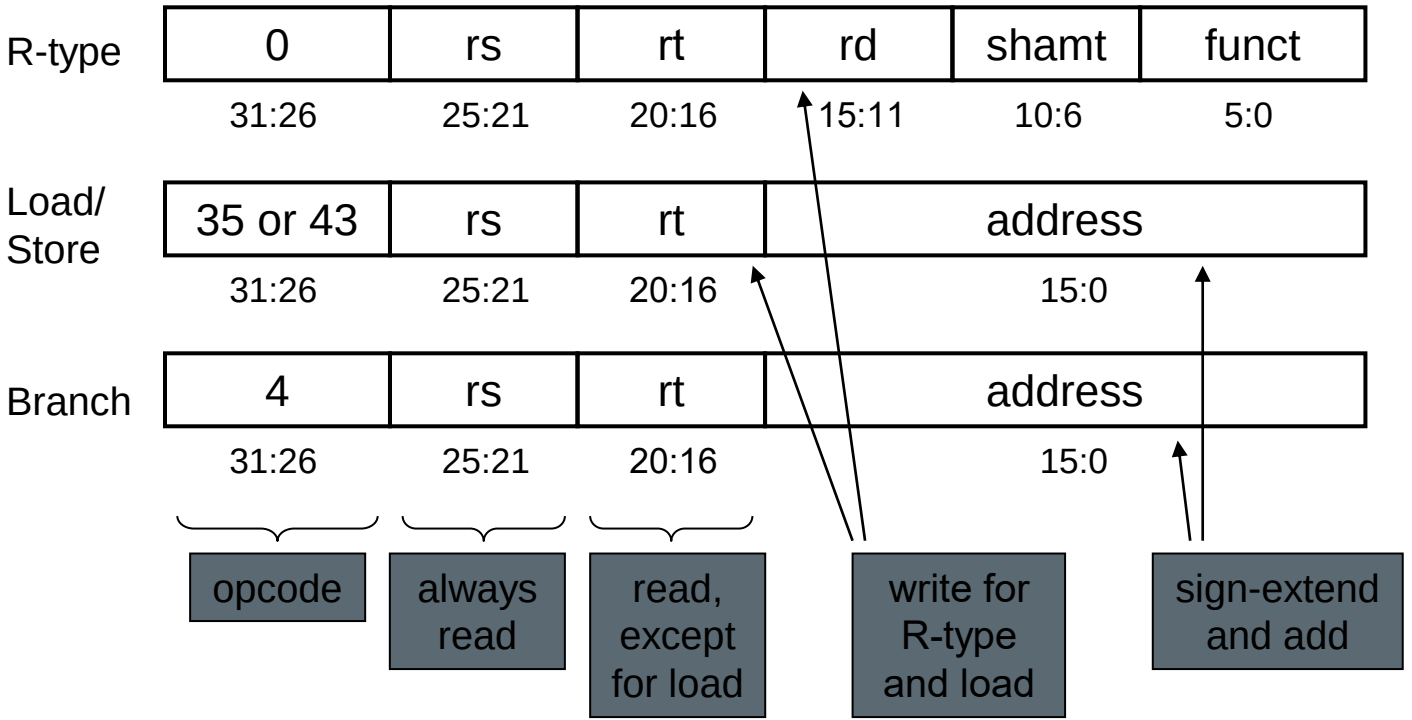
  Phép tính và các toán hạng cần thiết

Giai đoạn giải mã (Decode)



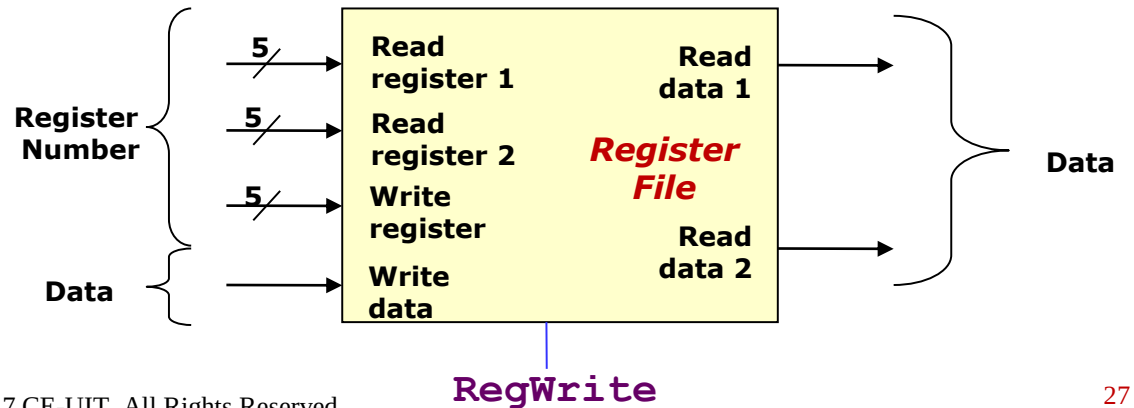
Khối chứa tập hợp của các thanh ghi, gọi là **Register file**
Hoặc **Registers**

Các định dạng lệnh

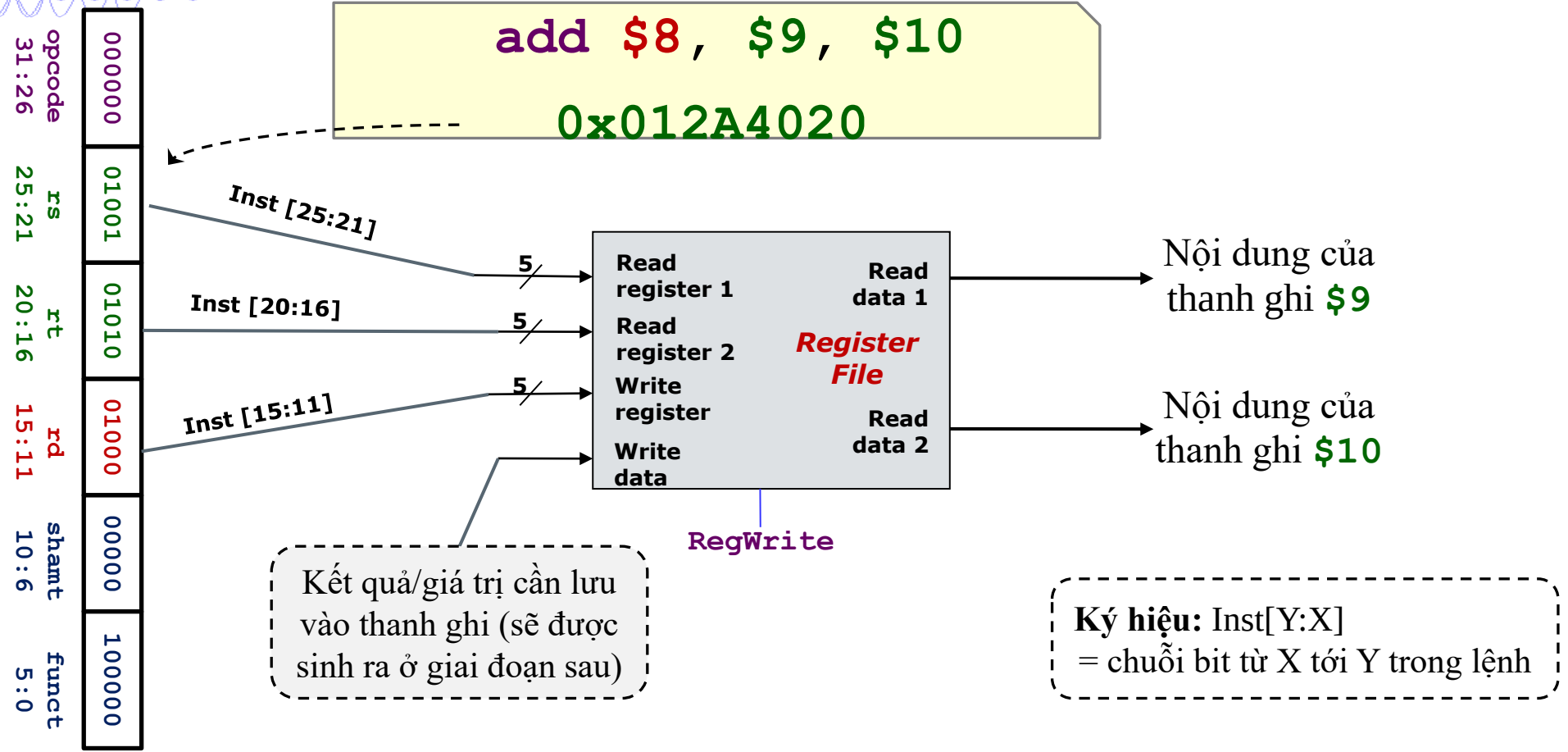


Khối *Register File*

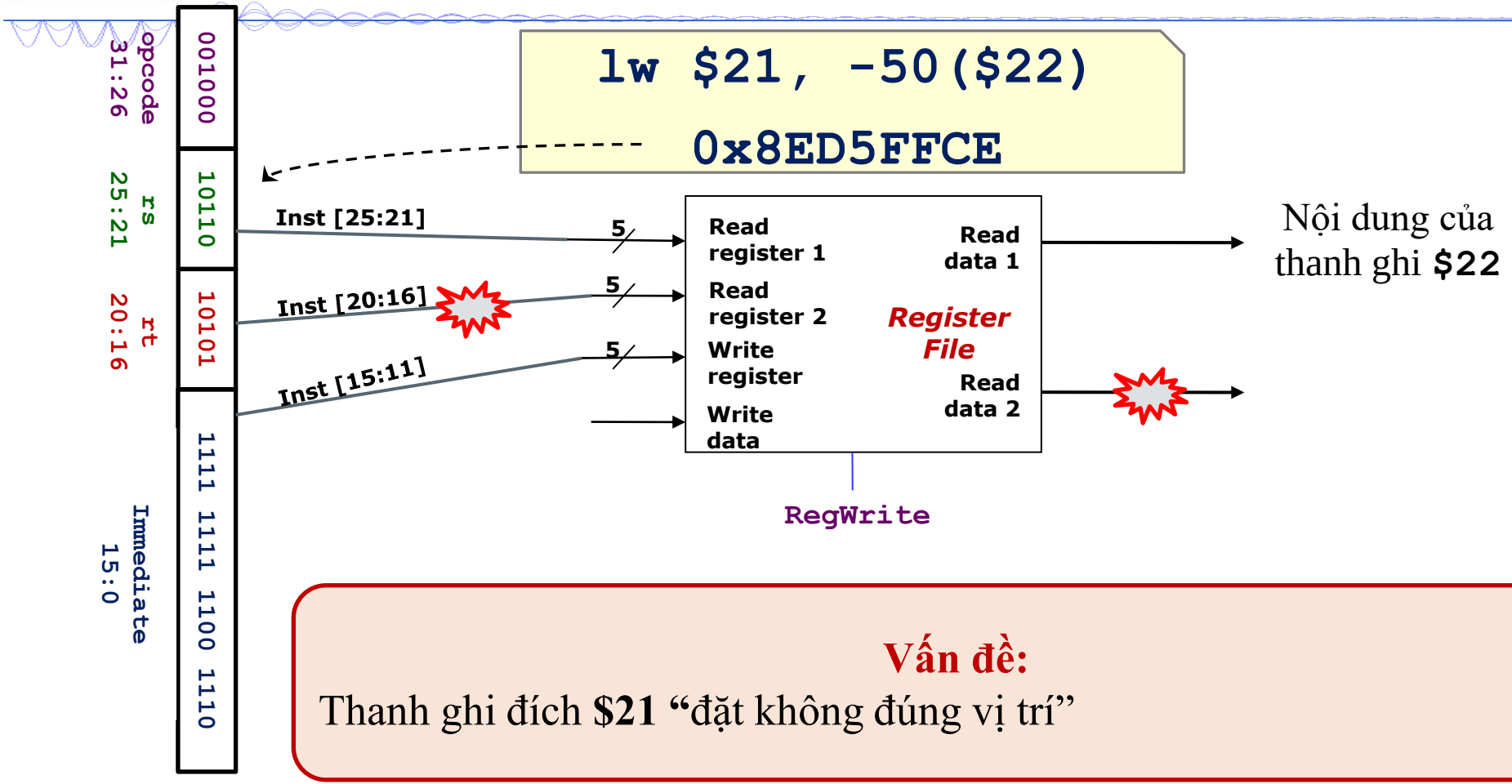
- Một tập 32 thanh ghi:
  - Mỗi thanh ghi có chiều dài 32 bit và có thể được đọc hoặc ghi bằng cách chỉ ra chỉ số của thanh ghi
  - Với mỗi lệnh, cho phép **đọc nhiều nhất từ 2 thanh ghi**
  - Với mỗi lệnh, cho phép **ghi vào nhiều nhất 1 thanh ghi**
- **`RegWrite`**: là một tín hiệu điều khiển nhằm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



Giải mã: lệnh **R-Type**



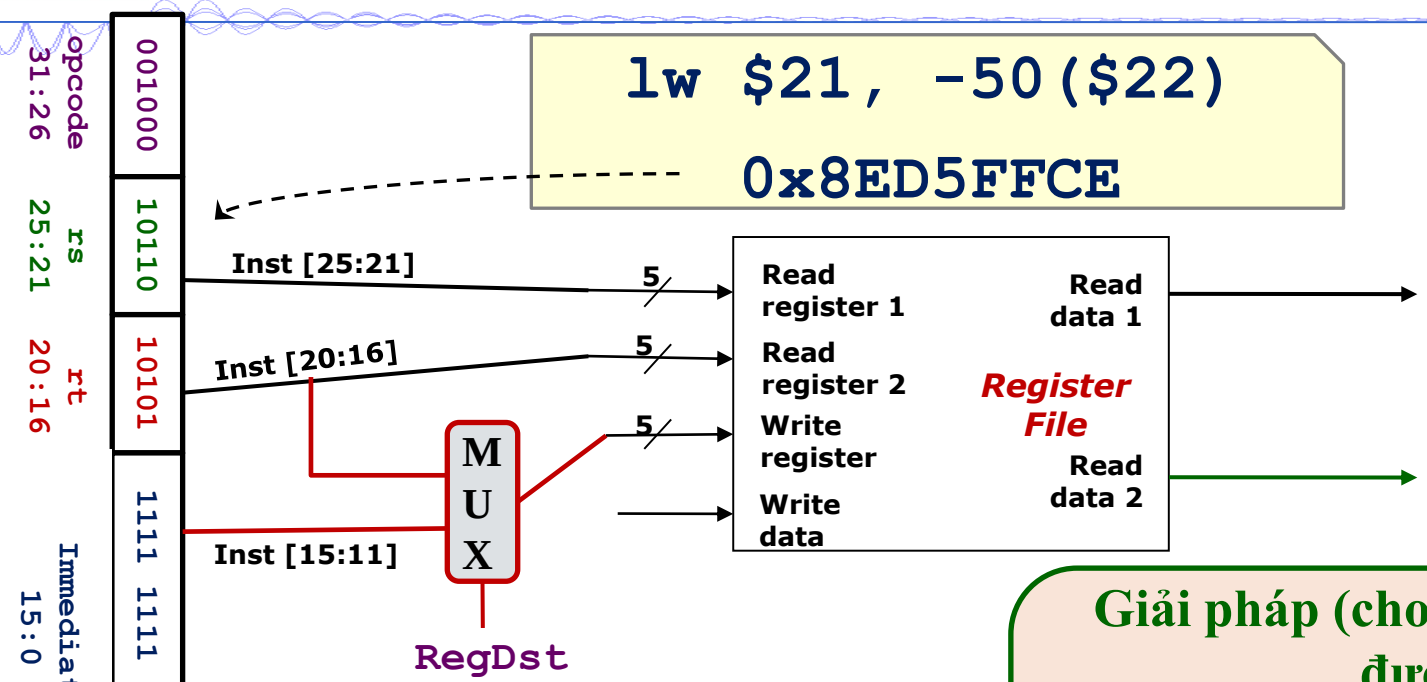
Giải mã: lệnh **I-Type**



Vấn đề:

Thanh ghi đích **$21** "đặt không đúng vị trí"

Giải mã: **Giải pháp cho ngõ "Write register"**

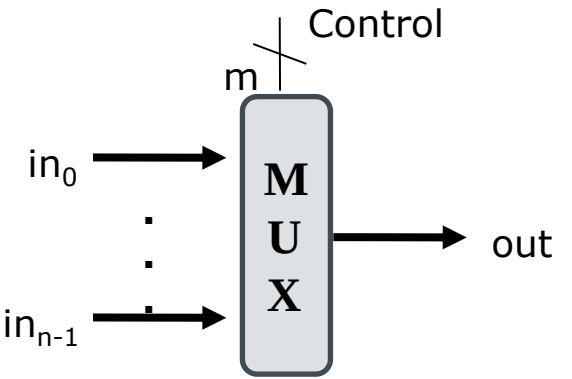


RegDst:
Tín hiệu điều khiển chọn Inst[20:16] hay [15:11] để đưa vào ngõ *write register*

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

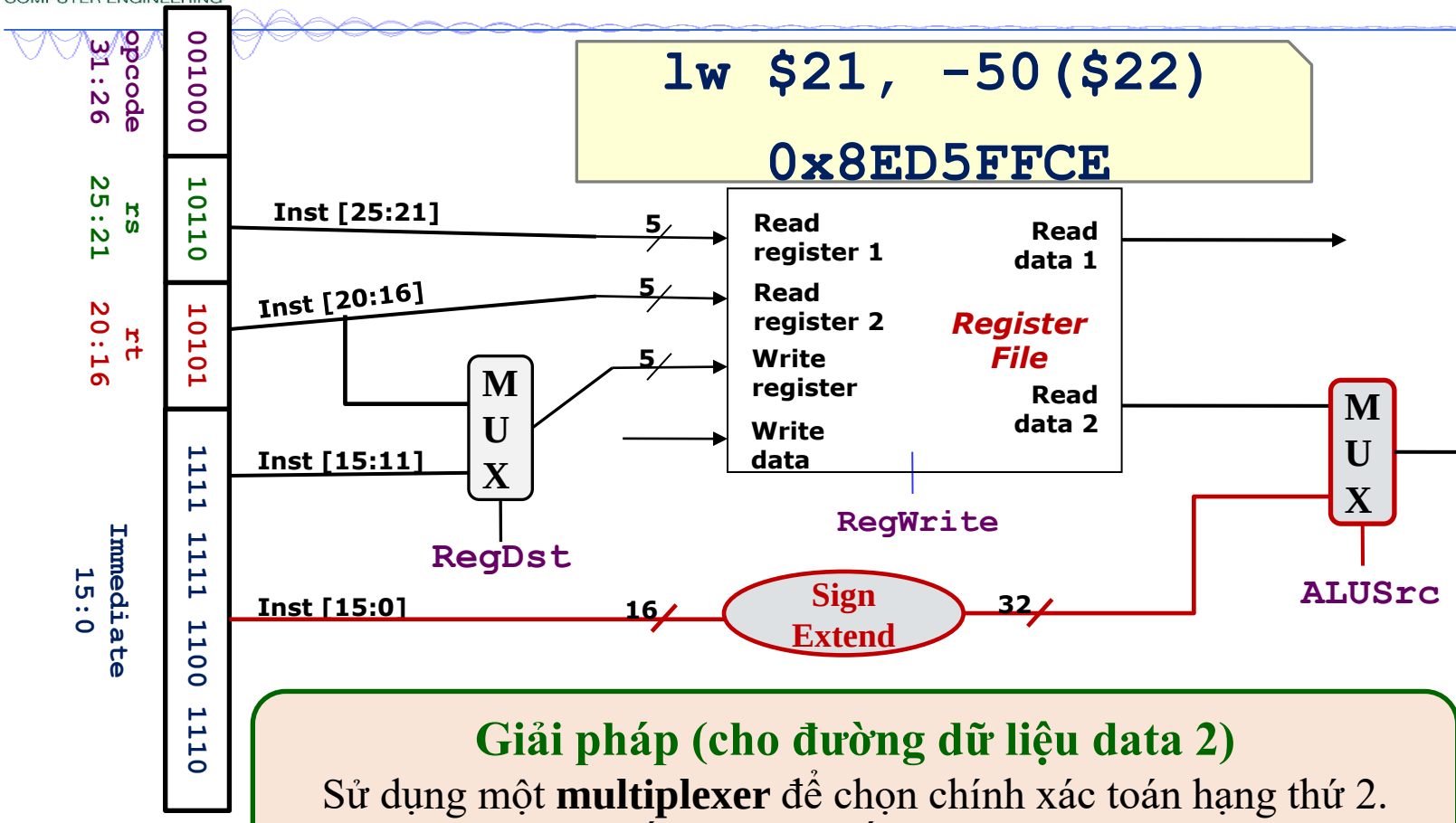
Multiplexer (MUX)

- **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 control = i



Control=0 → select in_0
Control=3 → select in_3

Giải mã: giải pháp cho ngõ "Data 2"



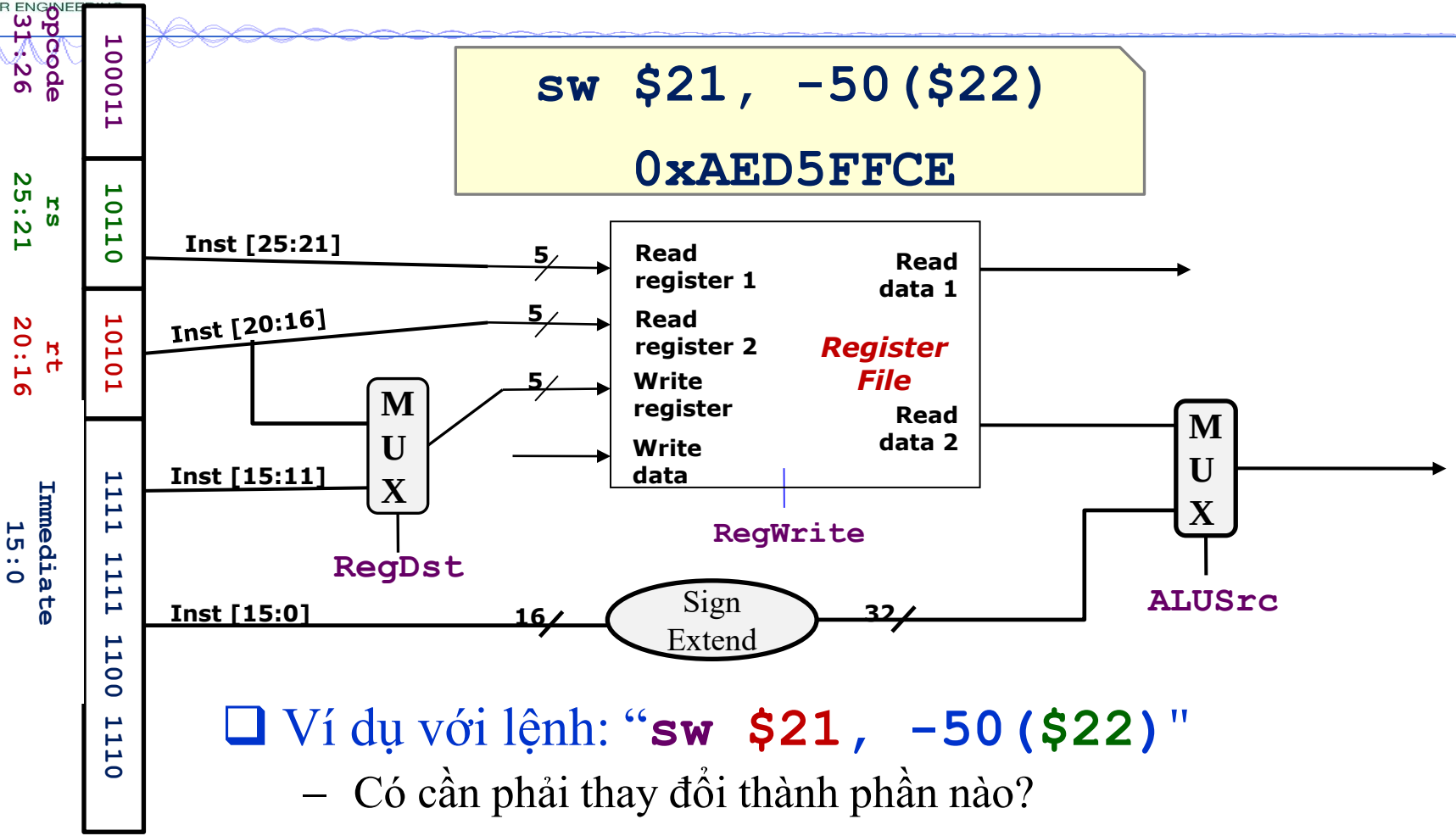
ALUSrc: Tín hiệu điều khiển để chọn "Read data 2" hay giá trị của Inst[15:0] (đã được mở rộng có dấu) cho toán hạng thứ hai

Giải pháp (cho đường dữ liệu data 2)

Sử dụng một **multiplexer** để chọn chính xác toán hạng thứ 2.

Sign extend: khối mở rộng số tức thời 16 bit thành 32 bit

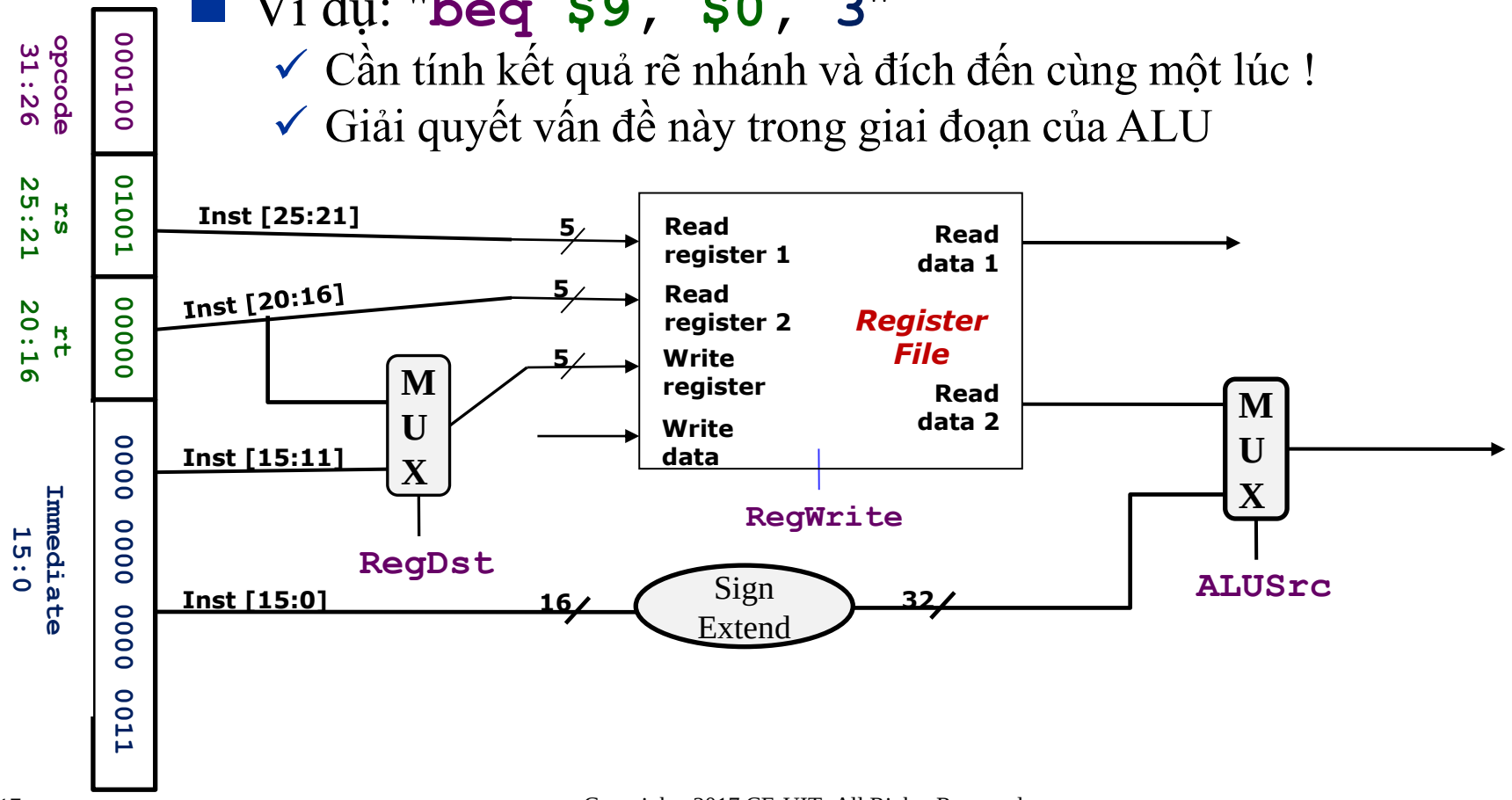
Giải mã: Lệnh Load Word



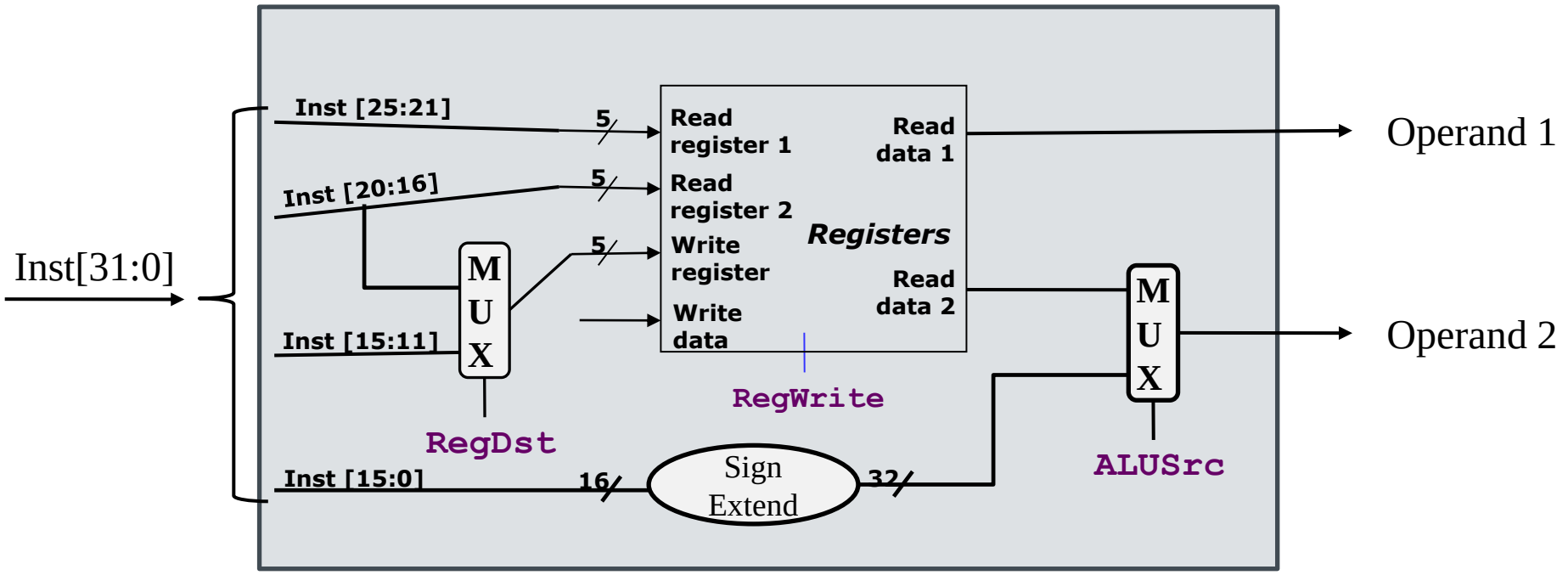
- Ví dụ với lệnh: "**sw $21, -50($22)**"
  - Có cần phải thay đổi thành phần nào?

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

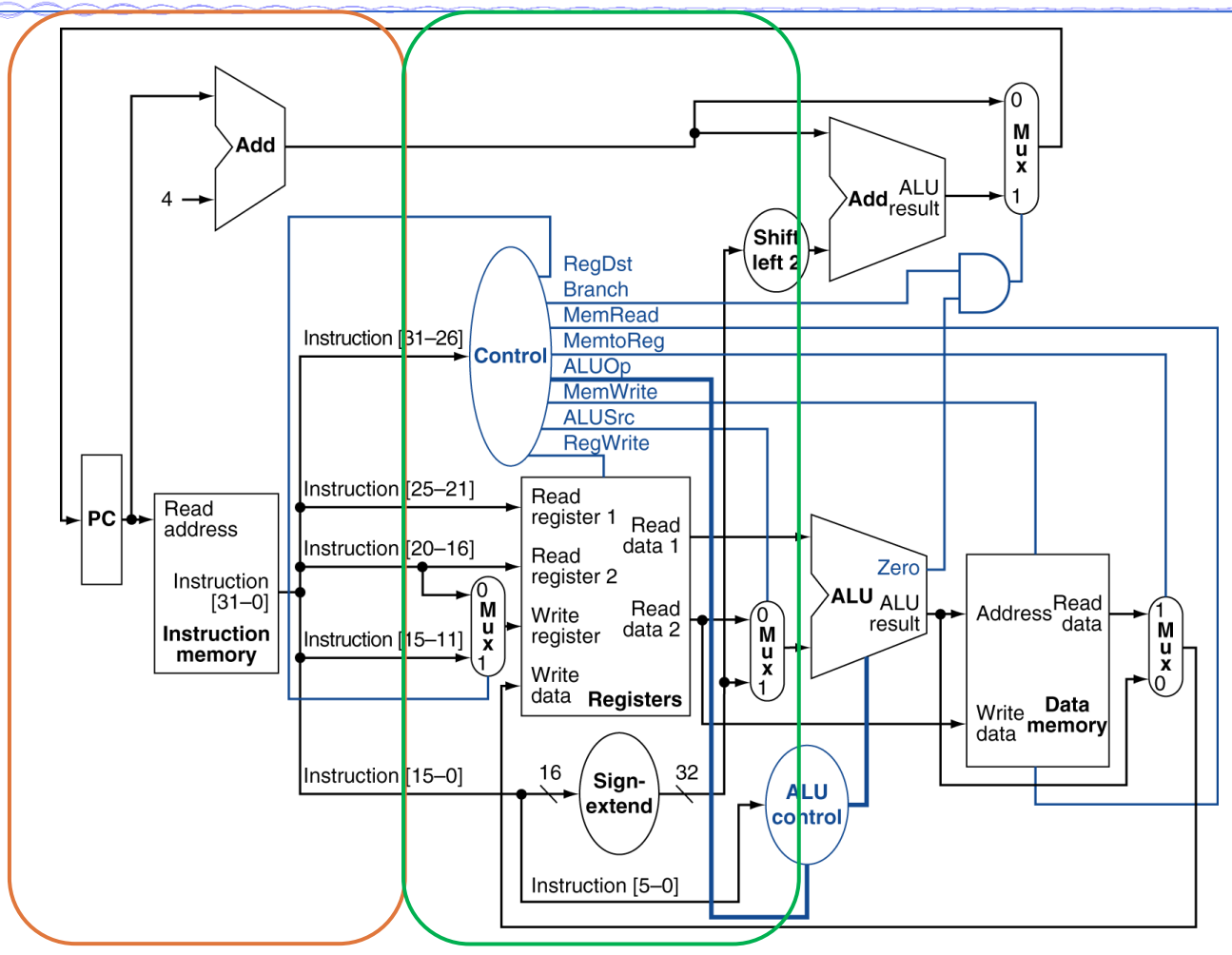


Giải mã: tổng kết



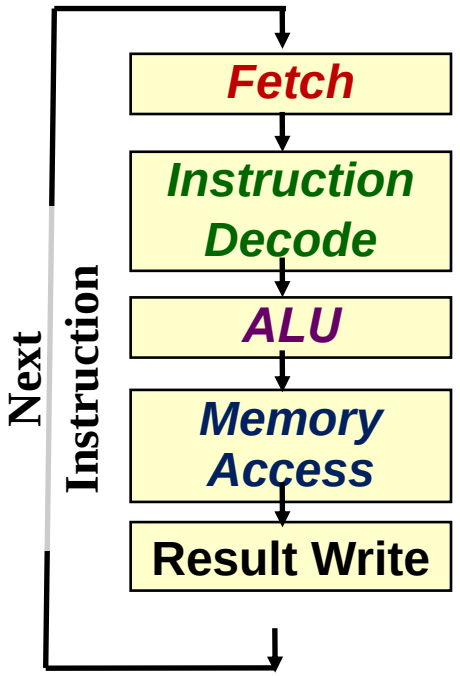
COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



COMPUTER ENGINEERING

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)

COMPUTER ENGINEERING

Công đoạn ALU

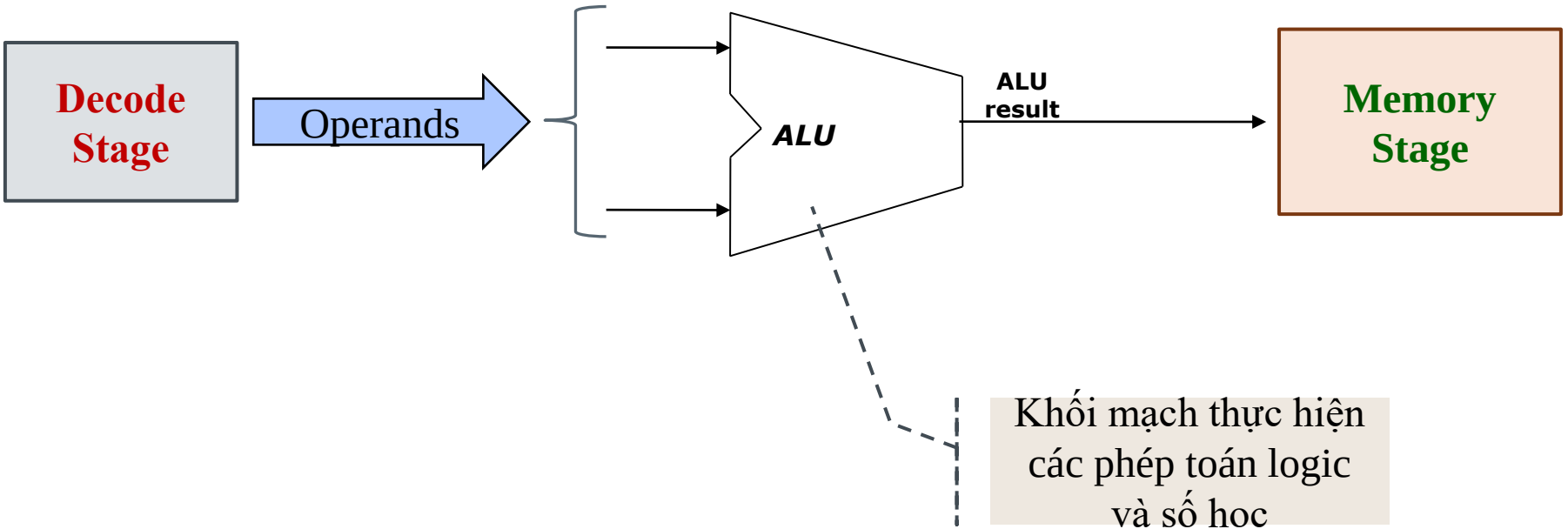
- Công đoạn ALU:
  - ✓ ALU = Arithmetic-Logic Unit
  - ✓ Công việc thật sự của hầu hết các lệnh được hiện chủ yếu trong giai đoạn này
    - Số học (Arithmetic) (ví dụ: **add**, **sub**), Logic (ví dụ: **and**, **or**): ALU tính ra kết quả cuối cùng
    - Lệnh làm việc với bộ nhớ (ví dụ: **lw**, **sw**): 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

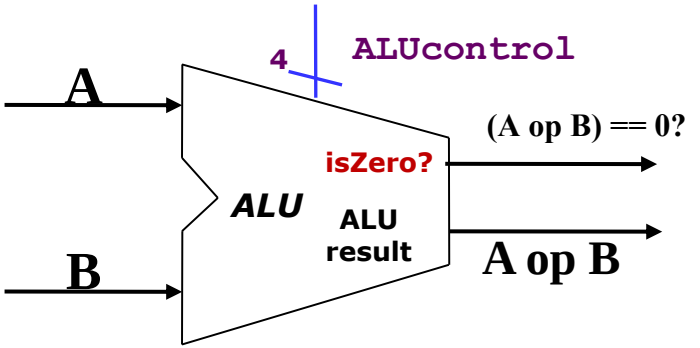
  (Đối với lệnh lw và sw: Kết quả của công đoạn này sẽ là địa chỉ cung cấp cho memory để lấy dữ liệu)

Công đoạn ALU



Khối *ALU (**Arithmetic Logical Unit)*

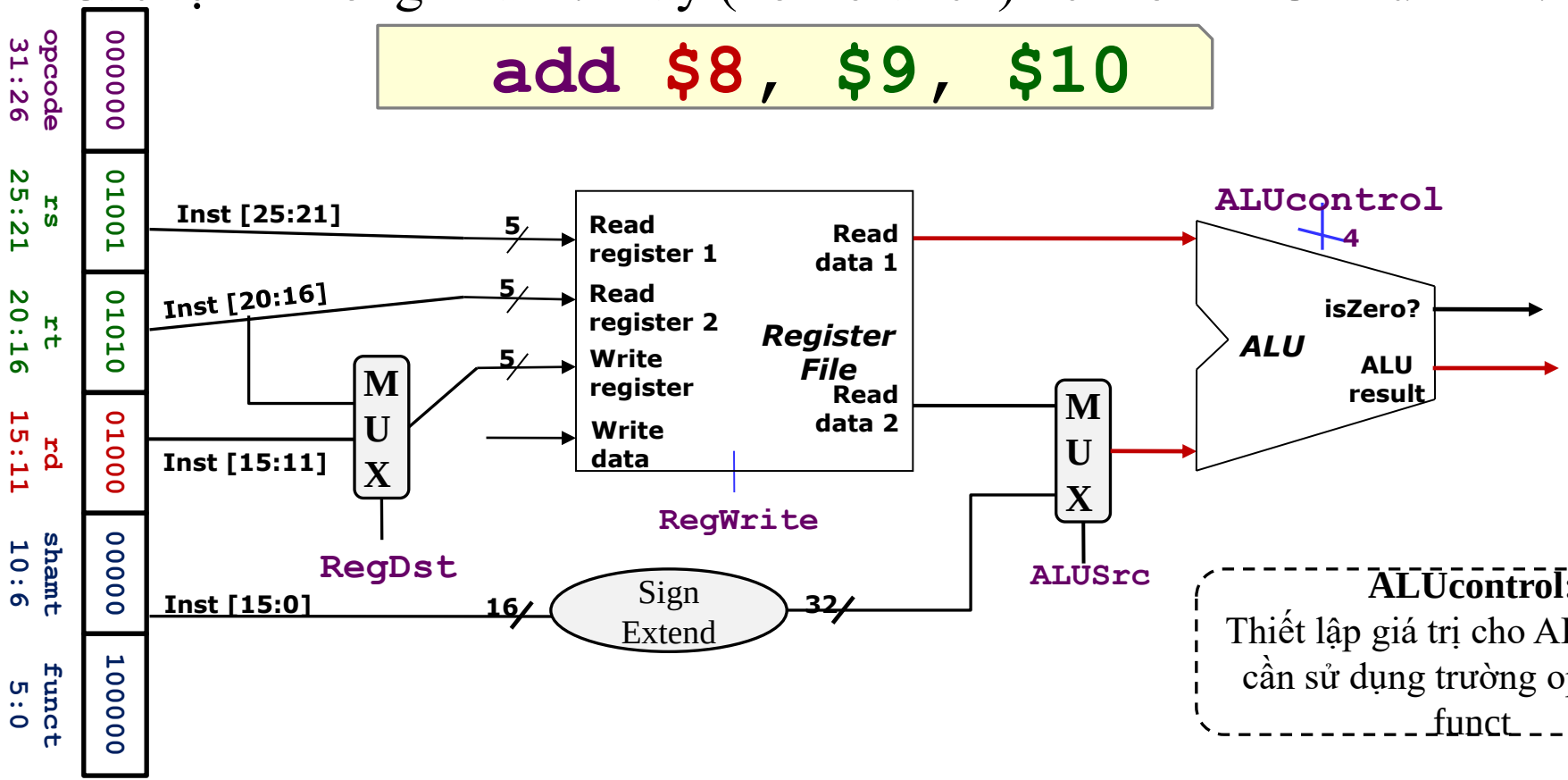
- **ALU (Arithmetic-logical unit)**
  - ✓ Khối dùng để thực hiện các phép tính logic và số học
- **Inputs:**
  - ✓ 2 số 32-bit
- **Điều khiển khối ALU:**
  - ✓ Do ALU có thể thực hiện nhiều chức năng → dùng 4-bit để quyết định chức năng/phép toán cụ thể nào cho ALU
- **Outputs:**
  - ✓ Kết quả của phép 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:



ALUcontrol:
Thiết lập giá trị cho ALUcontrol cần sử dụng trường opcode và funct

COMPUTER ENGINEERING

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

	add $3, $1, $2	**lw $3, 20($1)**	**beq $1, $2, label**
Fetch	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	○ Đọc thanh ghi $1, xem như toán hạng *opr1* ○ Đọc thanh ghi $2, xem như toán hạng *opr2*	○ Đọc thanh ghi $1, xem như toán hạng *opr1* ○ Sử dụng ***20*** như toán hạng *opr2*	○ Đọc thanh ghi $1, xem như toán hạng *opr1* ○ Đọc thanh ghi $2, xem như toán hạng *opr2*
ALU	*Result* = *opr1* + *opr2*	*MemAddr* = opr1 + opr2	*Taken* = (*opr1* == *opr2*)? *Target* = **PC** + Label*
Memory Access		Sử dụng *MemAddr* để đọc dữ liệu từ bộ nhớ	
Result Write	*Result* được lưu trữ vào **$3**	*Dữ liệu của từ nhớ có địa chỉ MemAddr được được lưu trữ vào* **$3**	if (*Taken*) **PC** = *Target*

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**"

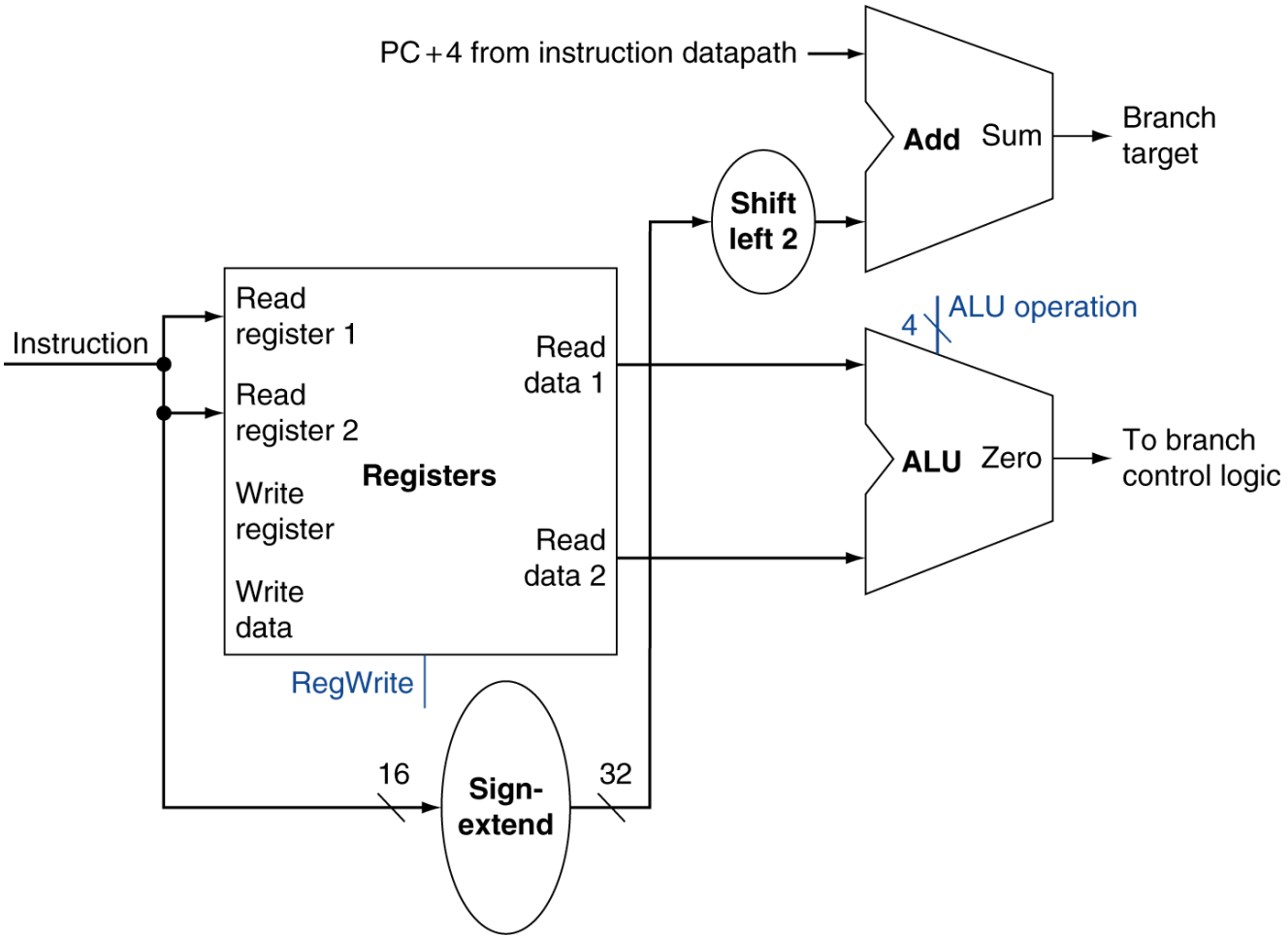
1. **Kết quả rẽ nhánh:**
   - ✓ Sử dụng ALU để so sánh thanh ghi
   - ✓ Tín hiệu 1-bit "**isZero?**“ để kiểm tra tính chất bằng/không bằng
2. **Địa chỉ đích của nhánh:**
   - ✓ Sử dụng một bộ cộng để tính địa chỉ
   - ✓ Cần nội dung của thanh ghi PC (từ Fetch Stage)
   - ✓ Cần Offset (từ Decode Stage)

BranchAddr = số lệnh từ lệnh beq đến nhãn * 4
=> Dùng lệnh sll 2

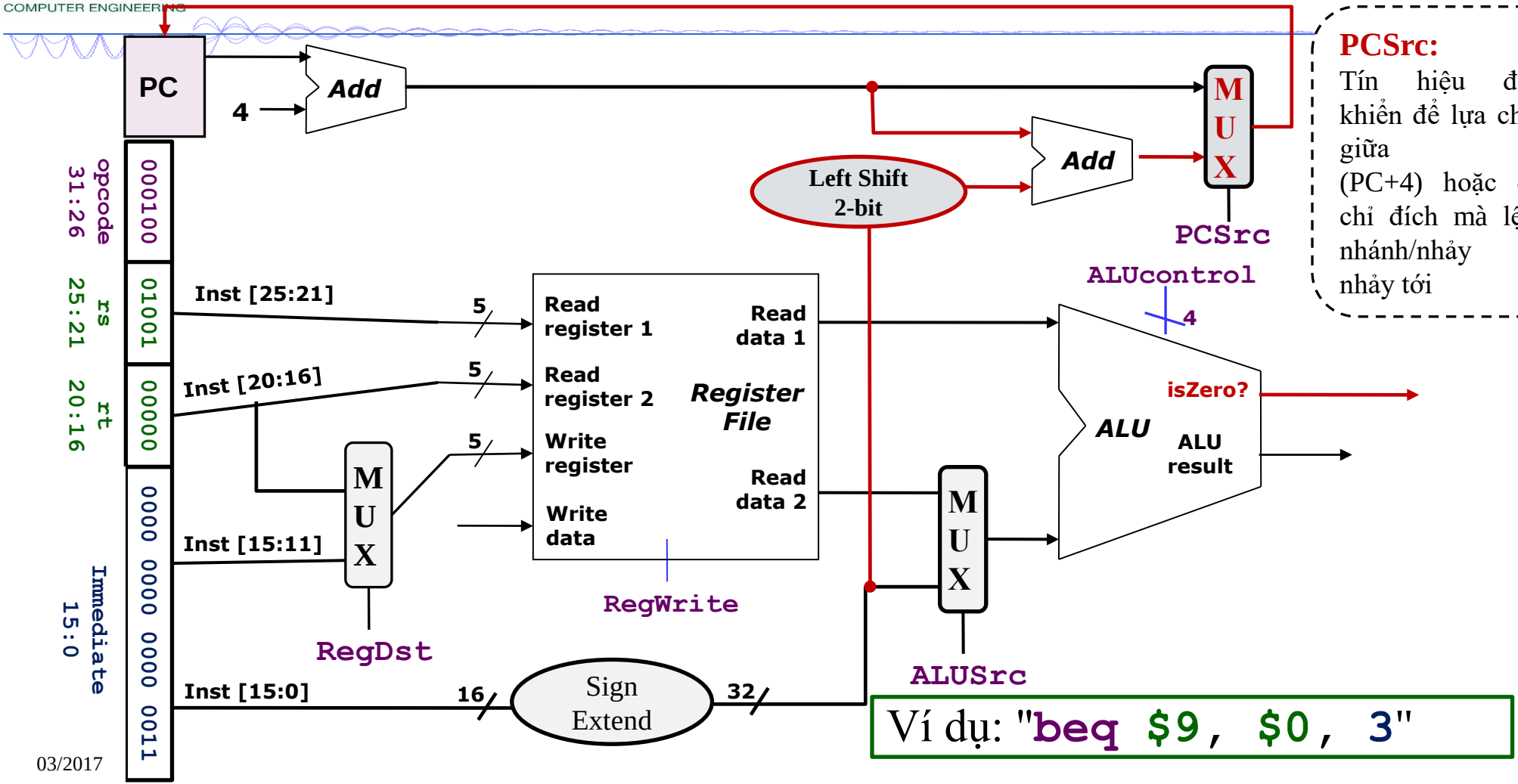
Branch On Equal	`beq`	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}

COMPUTER ENGINEERING

Công đoạn ALU: Các lệnh *Branch*



Datapath với công đoạn ALU hoàn chỉnh

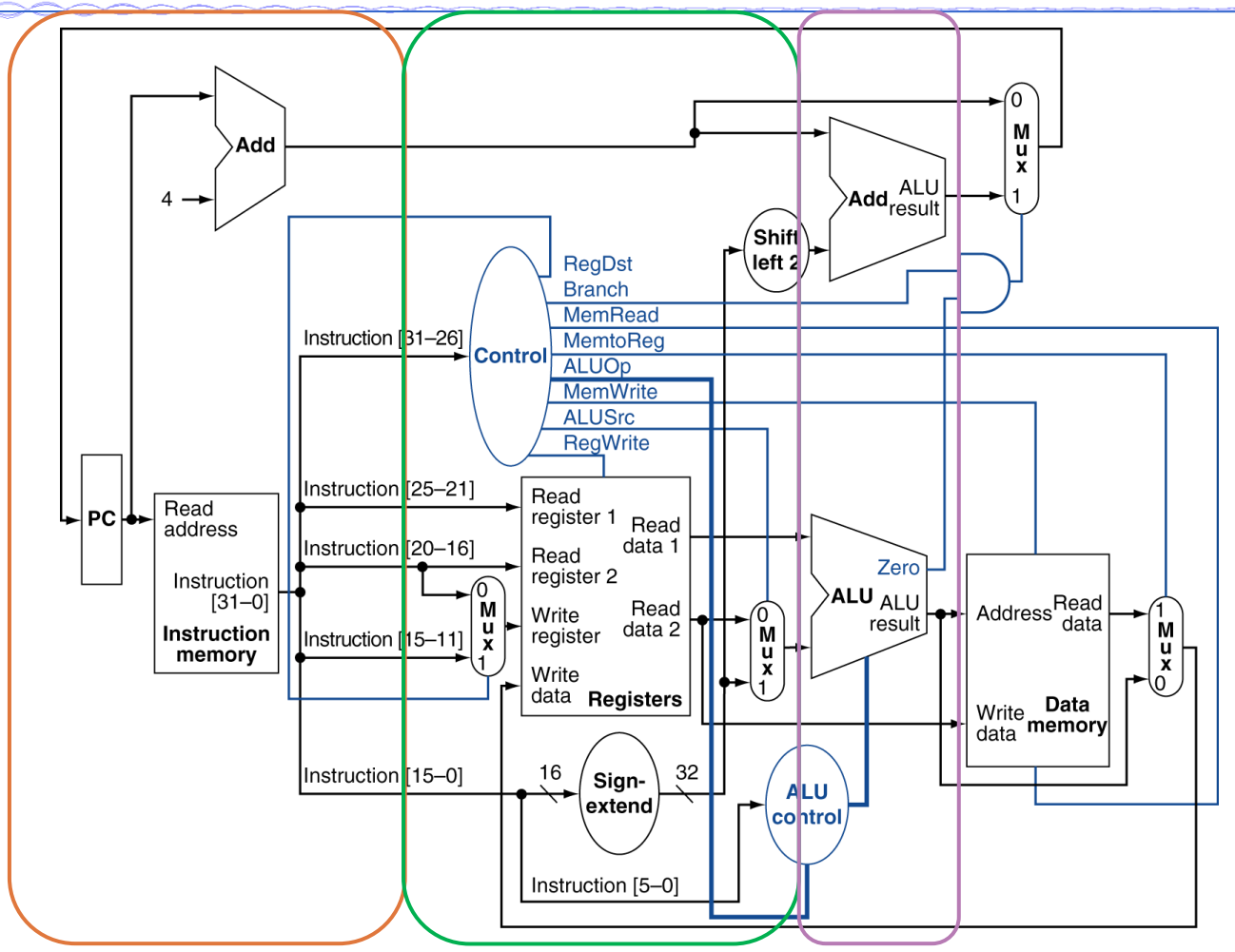


PCSrc:
Tín hiệu điều khiển để lựa chọn giữa (PC+4) hoặc địa chỉ đích mà lệnh nhánh/nhảy sẽ nhảy tới

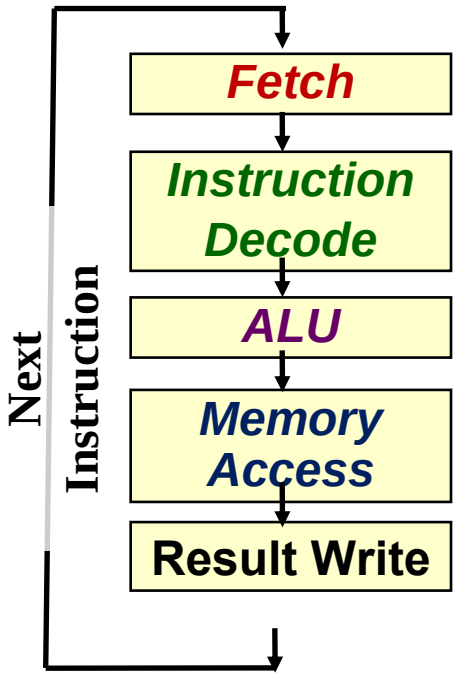
Ví dụ: "**beq $9, $0, 3**"

COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



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")

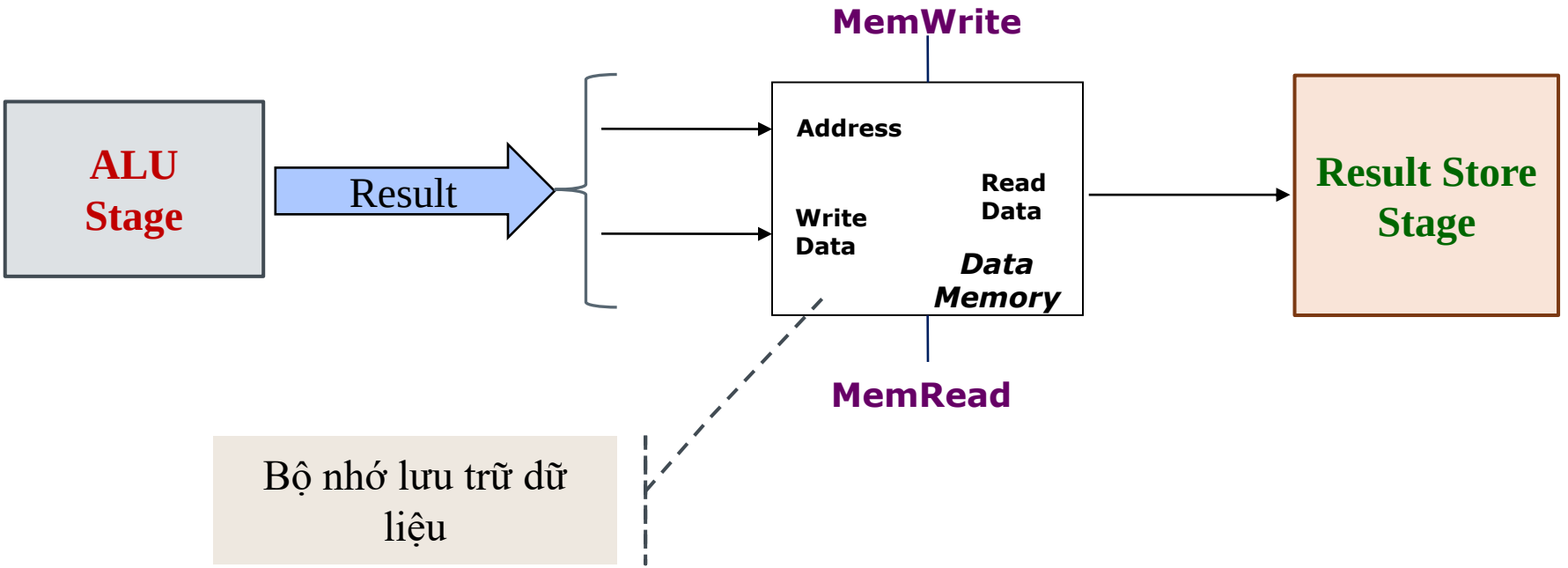
- **ALU** (Giai đoạn sử dụng ALU hay giai đoạn thực thi)
- **Memory Access** (Giai đoạn truy xuất vùng nhớ)
- **Result Write** (Giai đoạn ghi lại kết quả/lưu trữ)

COMPUTER ENGINEERING

Giai đoạn truy xuất vùng nhớ (Memory stage)

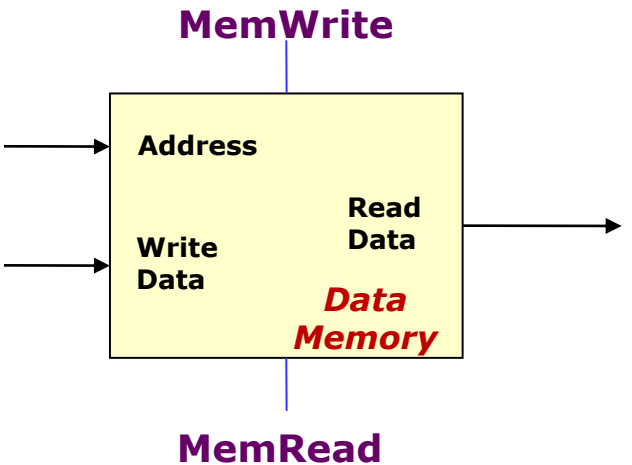
- Giai đoạn truy xuất vùng nhớ:
  - ✓ Chỉ có lệnh *Load* và *Store* cần thực hiện các thao tác trong giai đoạn này:
    - Sử dụng địa chỉ vùng nhớ được tính toán ở giai đoạn ALU
    - Đọc dữ liệu ra hoặc ghi dữ liệu vào vùng nhớ dữ liệu
  - ✓ Tất cả các lệnh khác sẽ rảnh trong giai đoạn này
- Đầu vào 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)



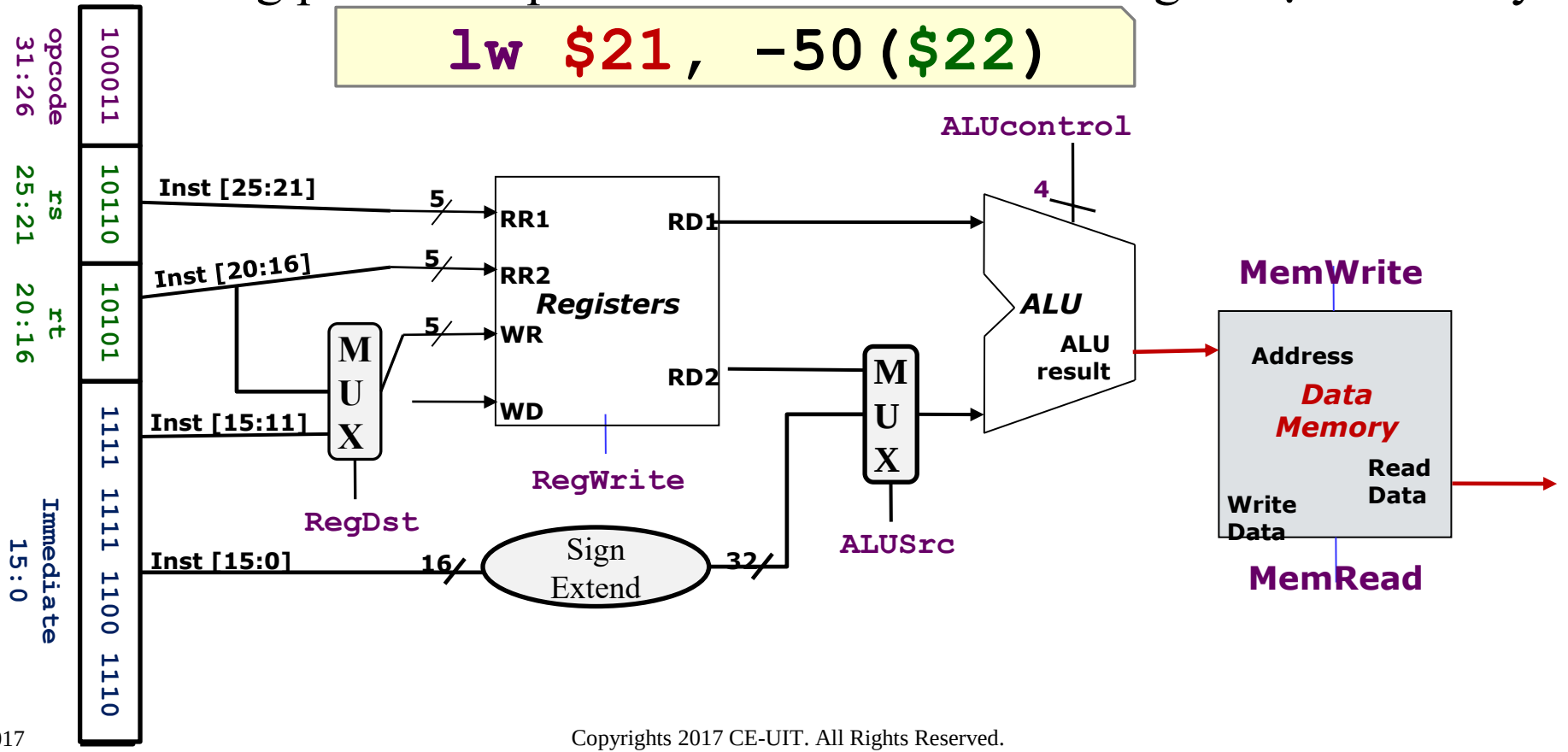
Khối *Data Memory*

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



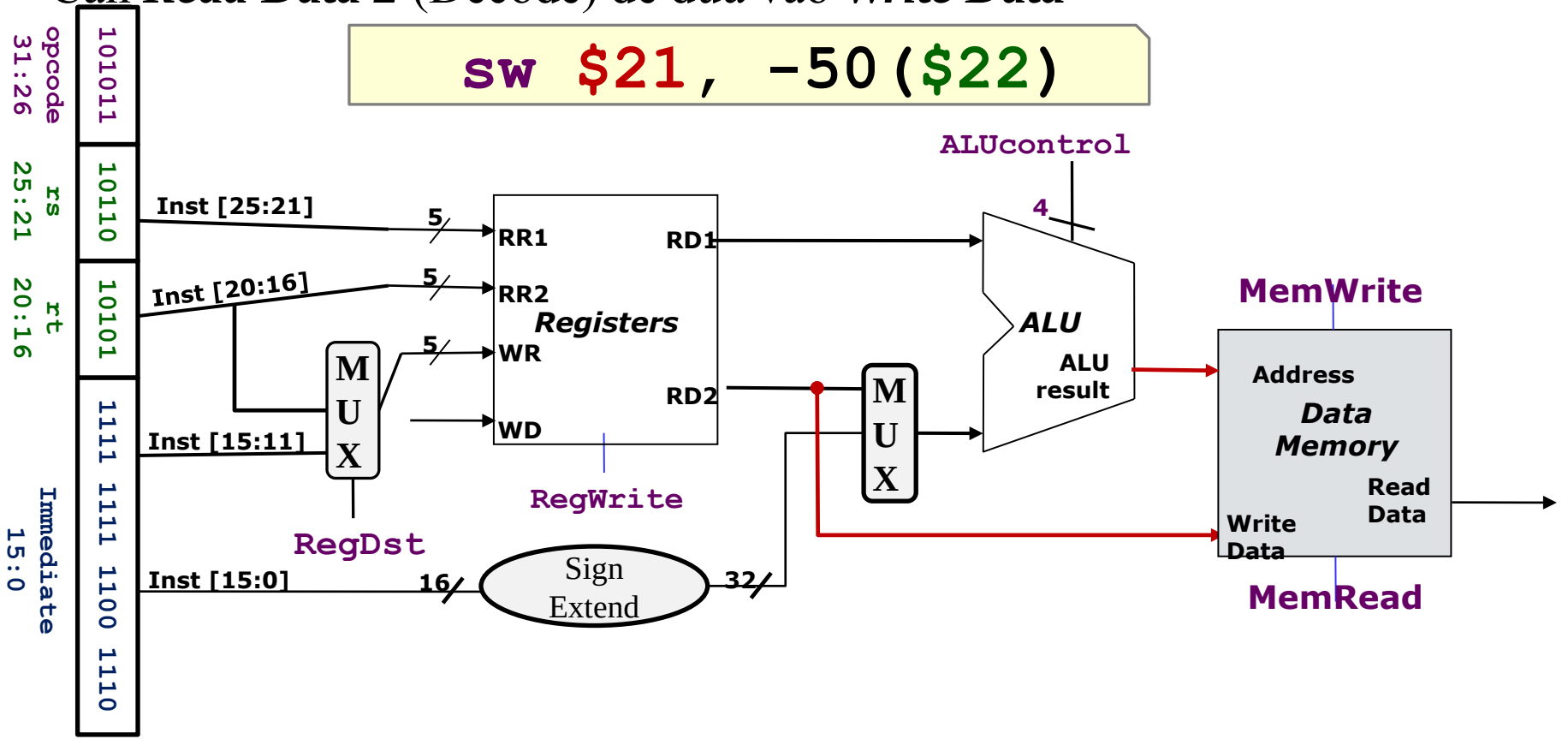
Giai đoạn Memory: lệnh *Load*

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



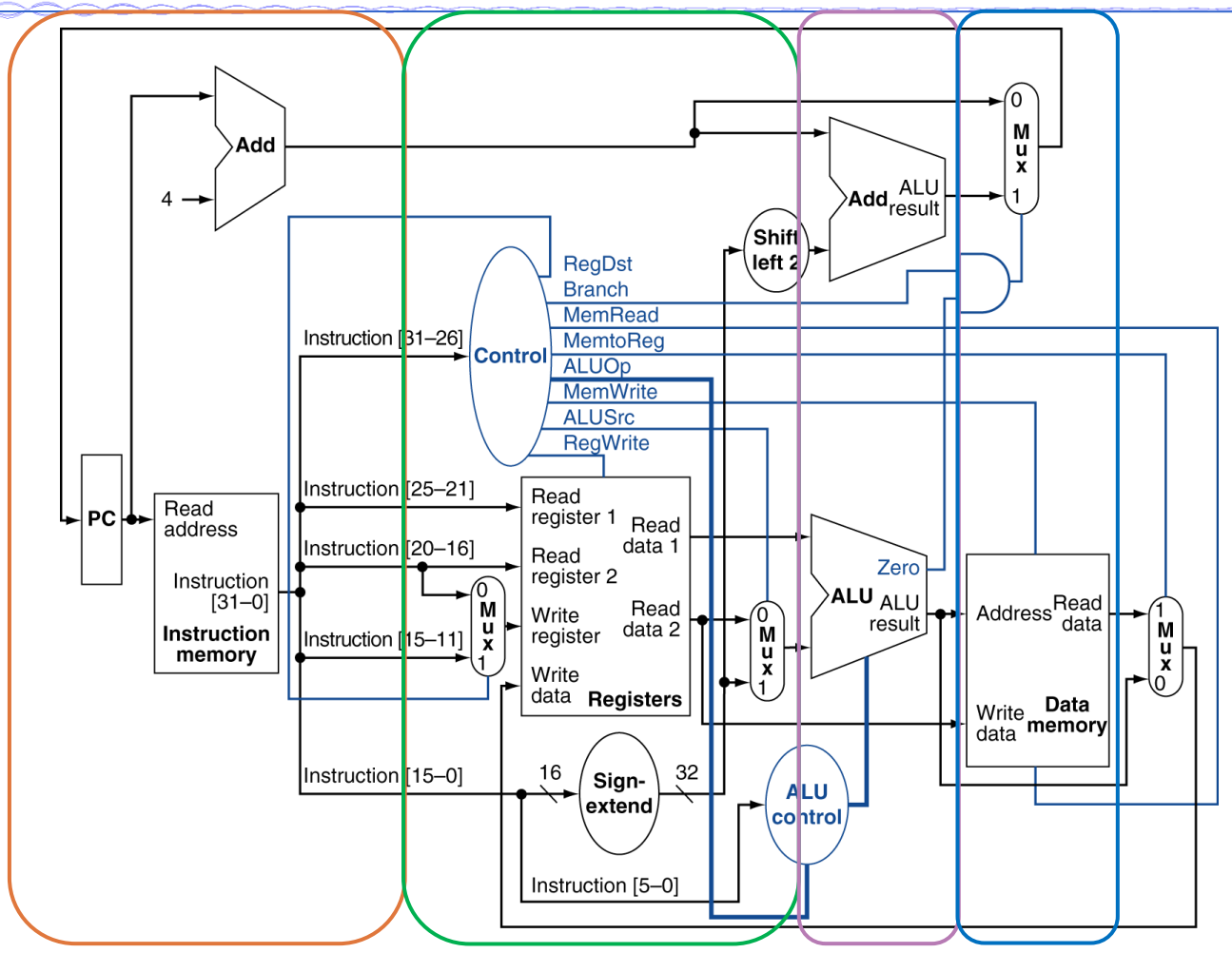
Giai đoạn Memory: lệnh *Store*

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

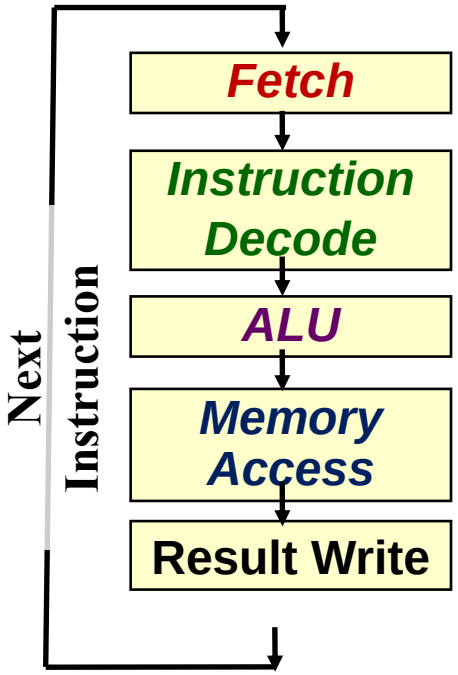


COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



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")

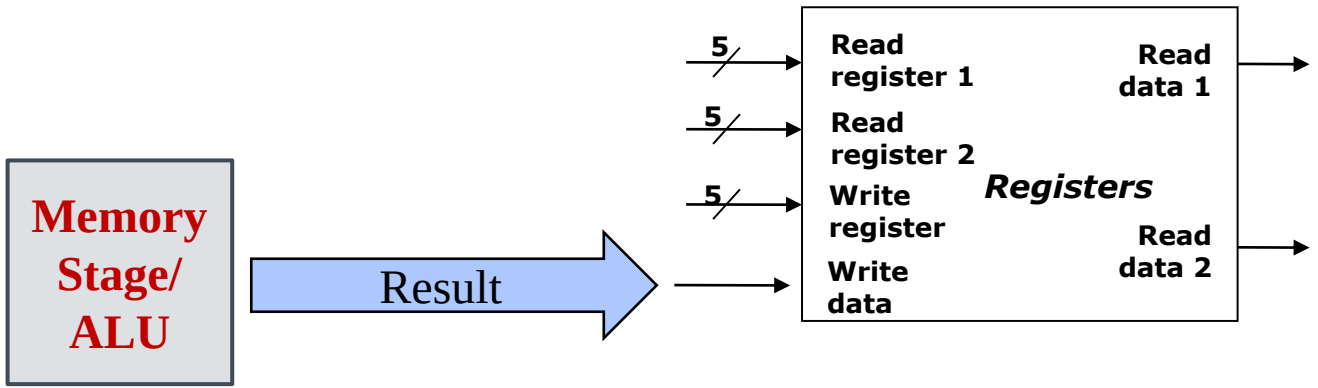
- **ALU** (Giai đoạn sử dụng ALU hay giai đoạn thực thi)
- **Memory Access** (Giai đoạn truy xuất vùng nhớ)
- **Result Write** (Giai đoạn ghi lại kết quả/lưu trữ)

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: từ Memory hoặc từ ALU

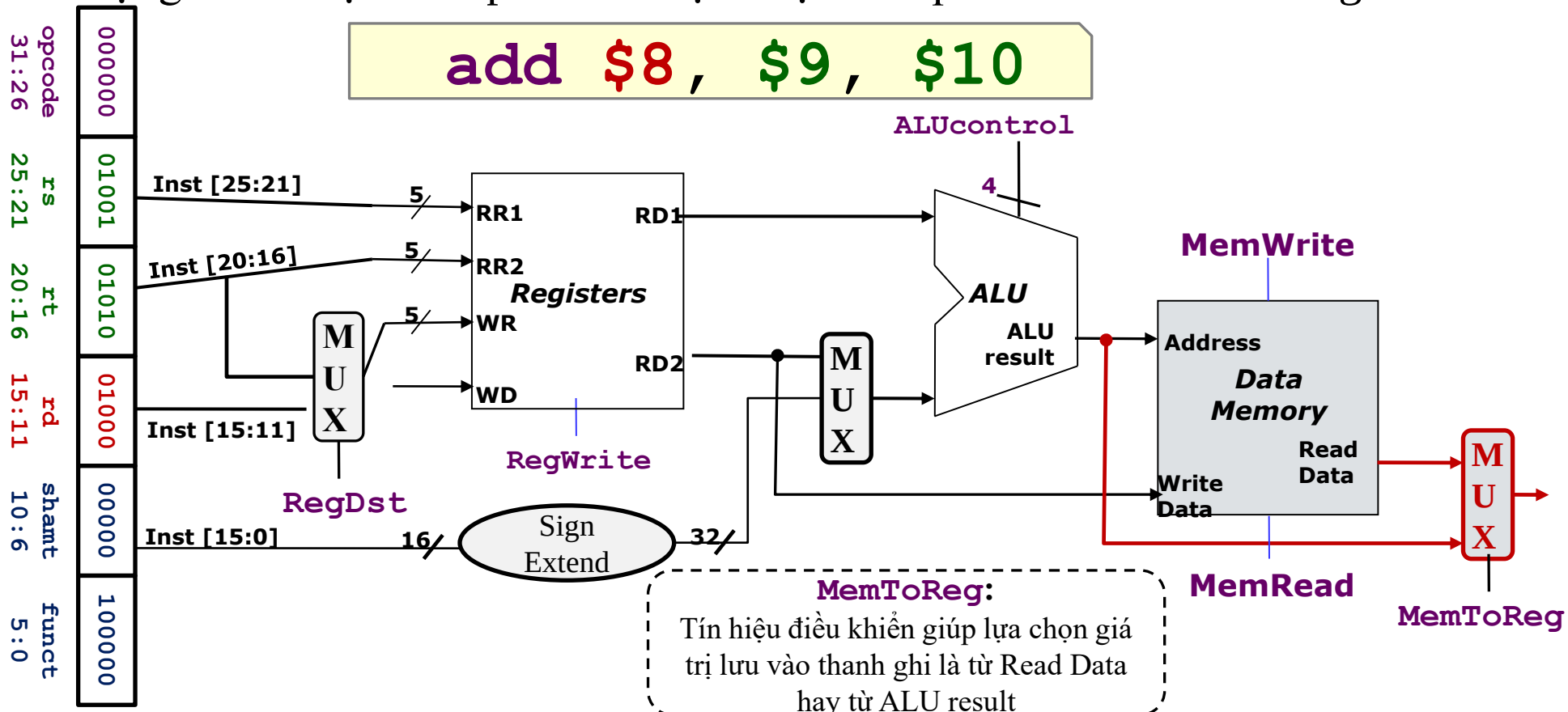
Giai đoạn lưu trữ kết quả (Result Write)



- 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**

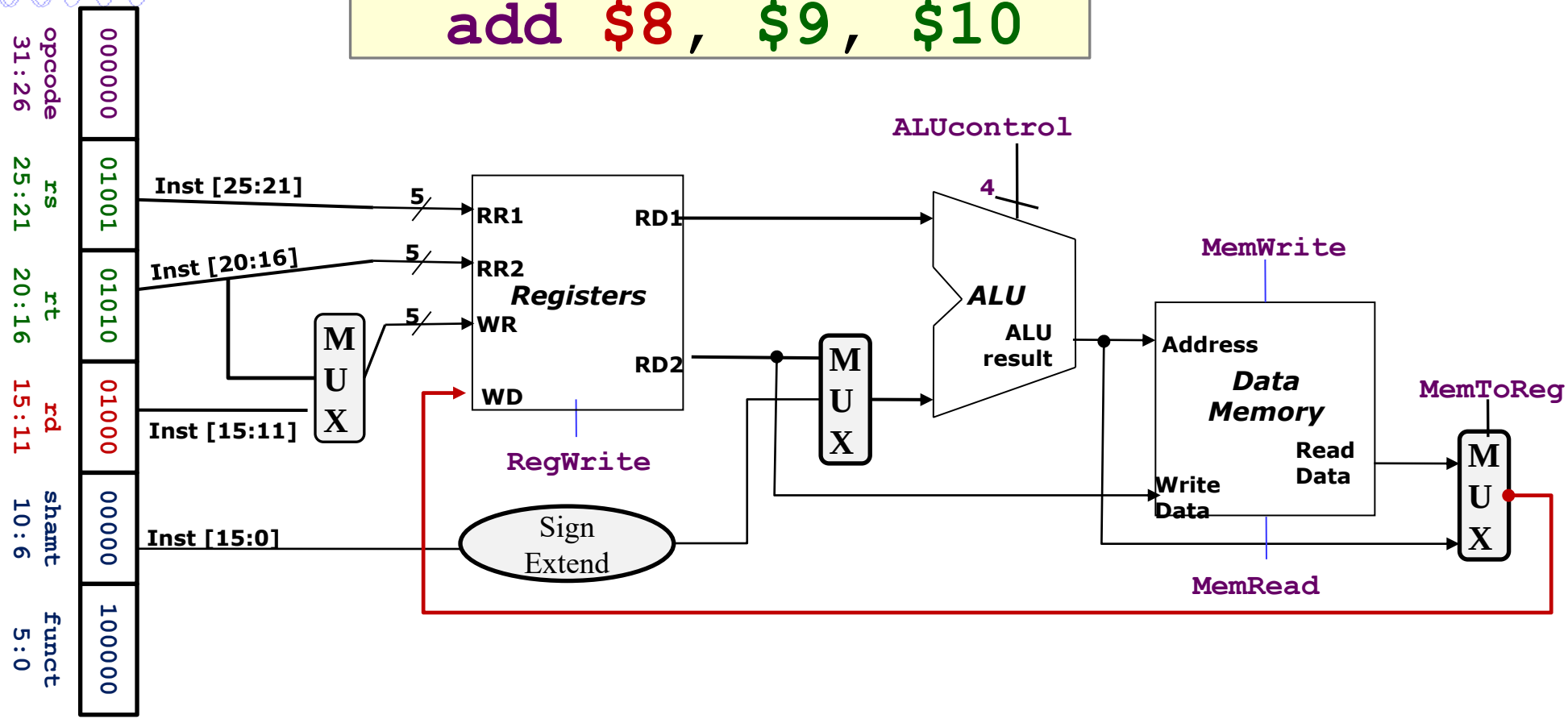
Giai đoạn lưu trữ kết quả (Result Write)

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



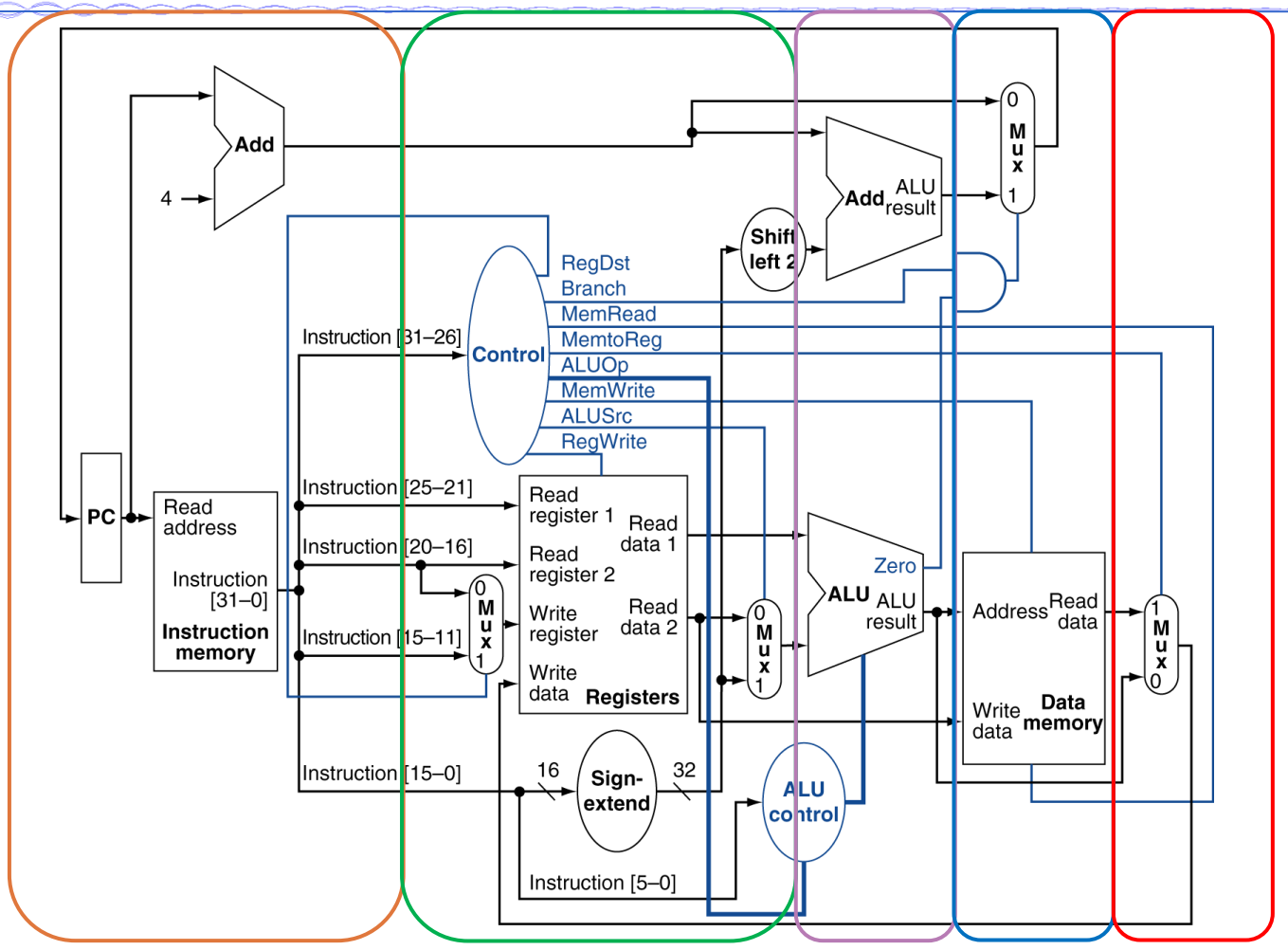
Giai đoạn lưu trữ kết quả (Result Write)

```
add $8, $9, $10
```



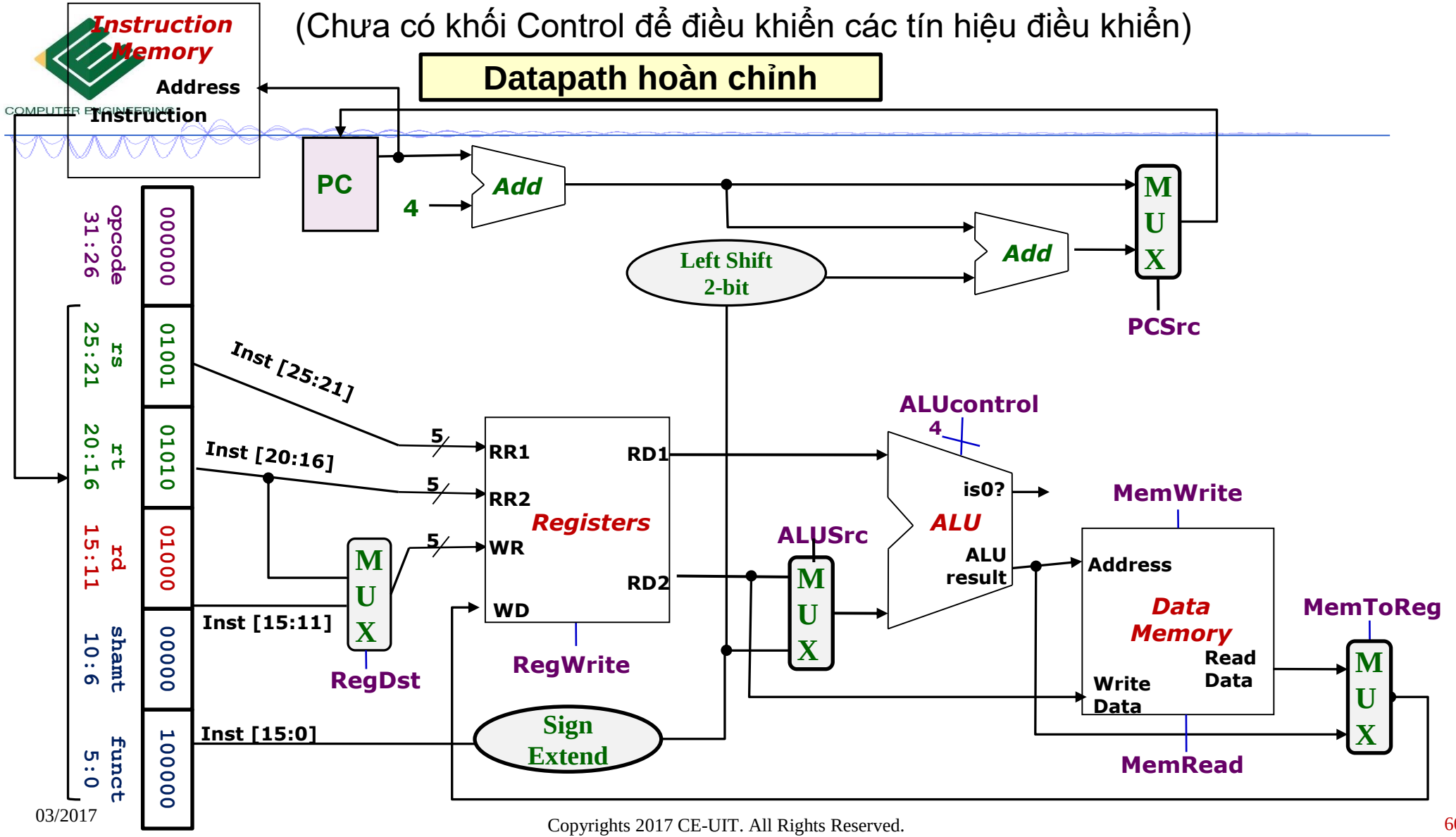
COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



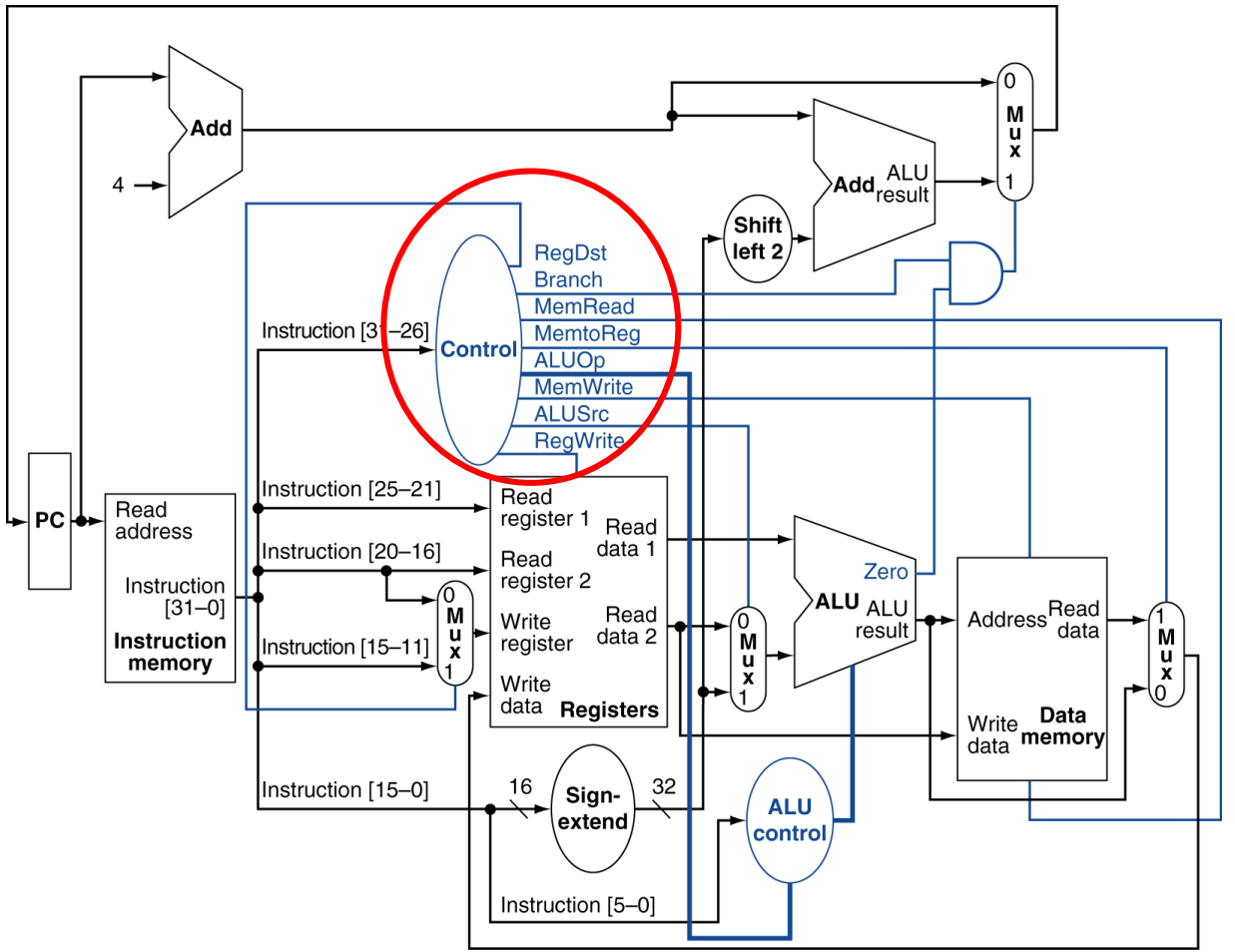
(Chưa có khối Control để điều khiển các tín hiệu điều khiển)

Datapath hoàn chỉnh



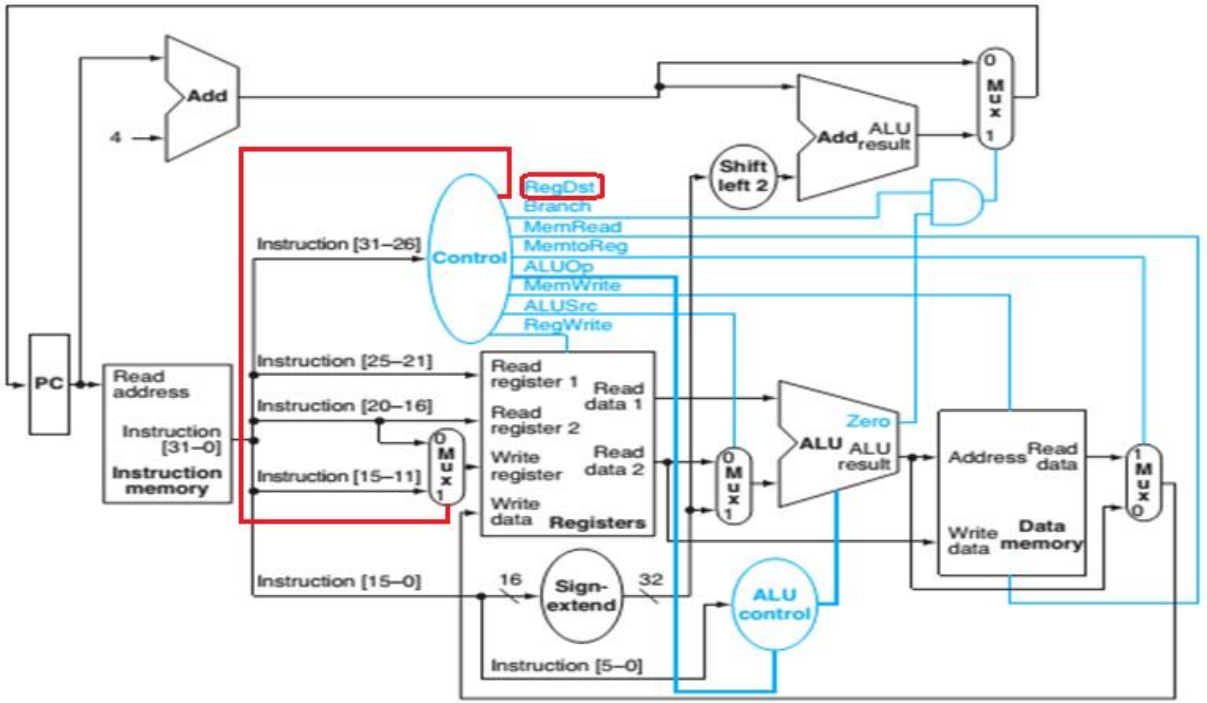
COMPUTER ENGINEERING

Datapath của một bộ xử lý với 8 lệnh MIPS



COMPUTER ENGINEERING

Hiện thực datapath - Control (2)

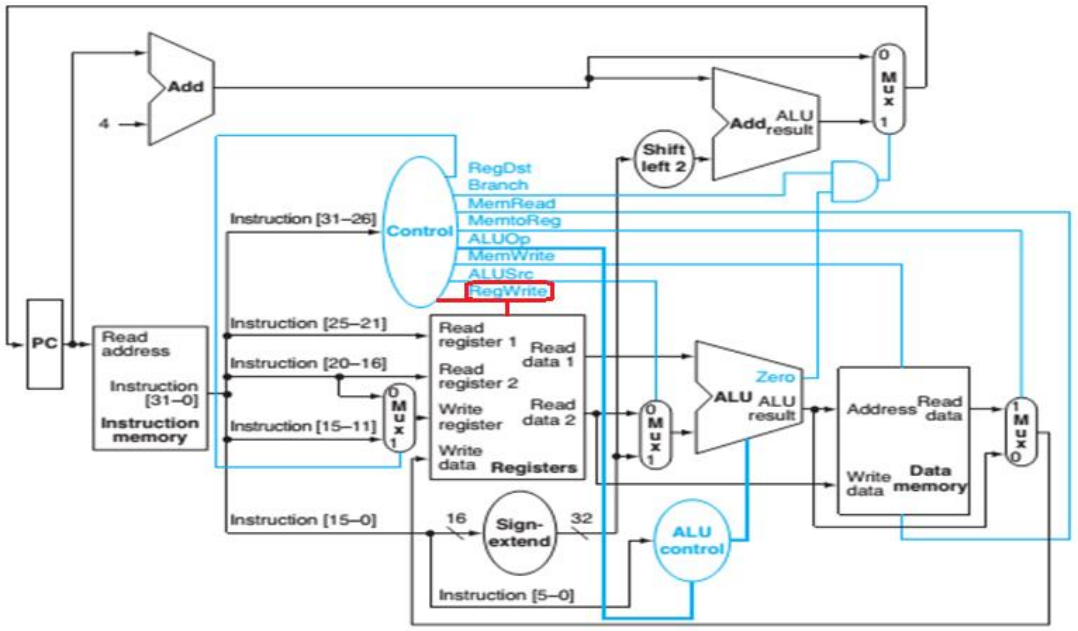


Tín hiệu RegDst trong datapath

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

Hiện thực datapath - Control (3)



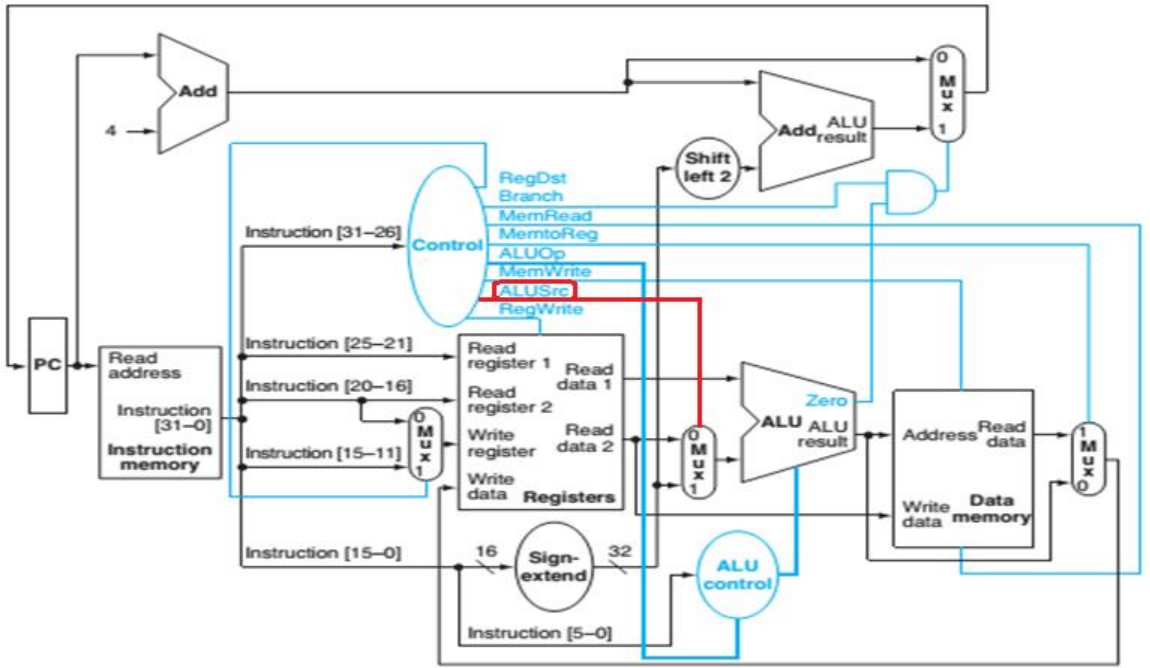
Tín hiệu RegWrite trong datapath

RegWrite dùng để cho phép thao tác ghi vào khối 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 chứ 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)

COMPUTER ENGINEERING

Hiện thực datapath - Control (4)

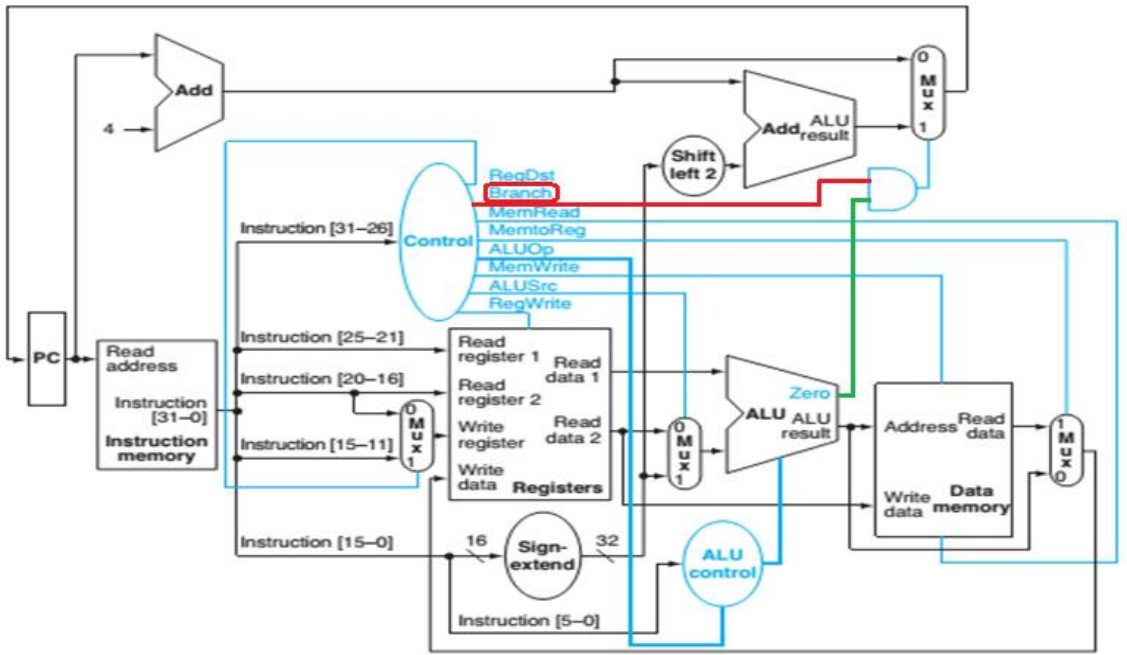


Tín hiệu ALUSrc trong datapath

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 (5)

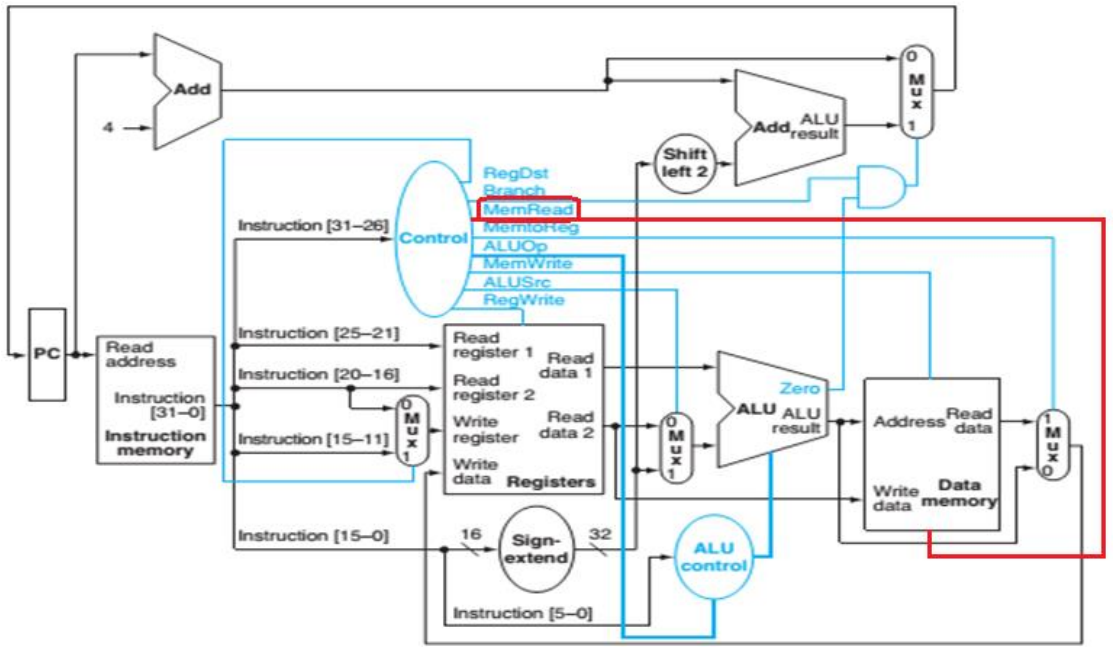


Tín hiệu Branch trong datapath

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 + Lable

Hiện thực datapath - Control (6)

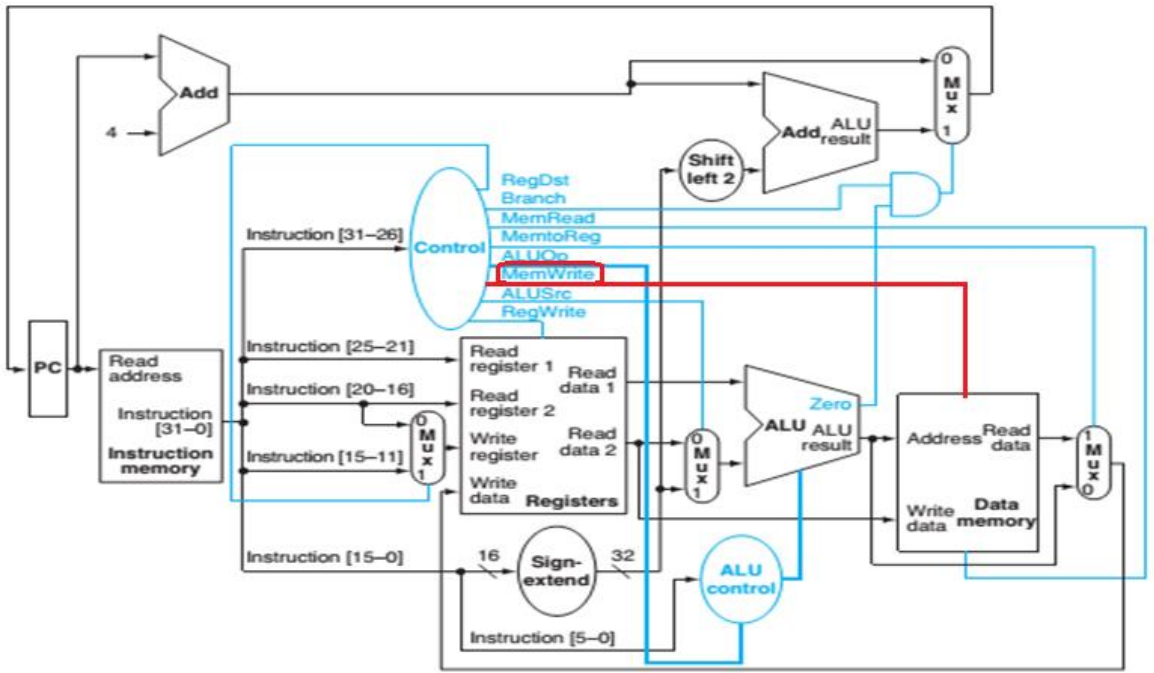


Tín hiệu MemRead trong datapath

MemRead dùng để cho phép thao tác đọc từ khối 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)

Hiện thực datapath - Control (7)



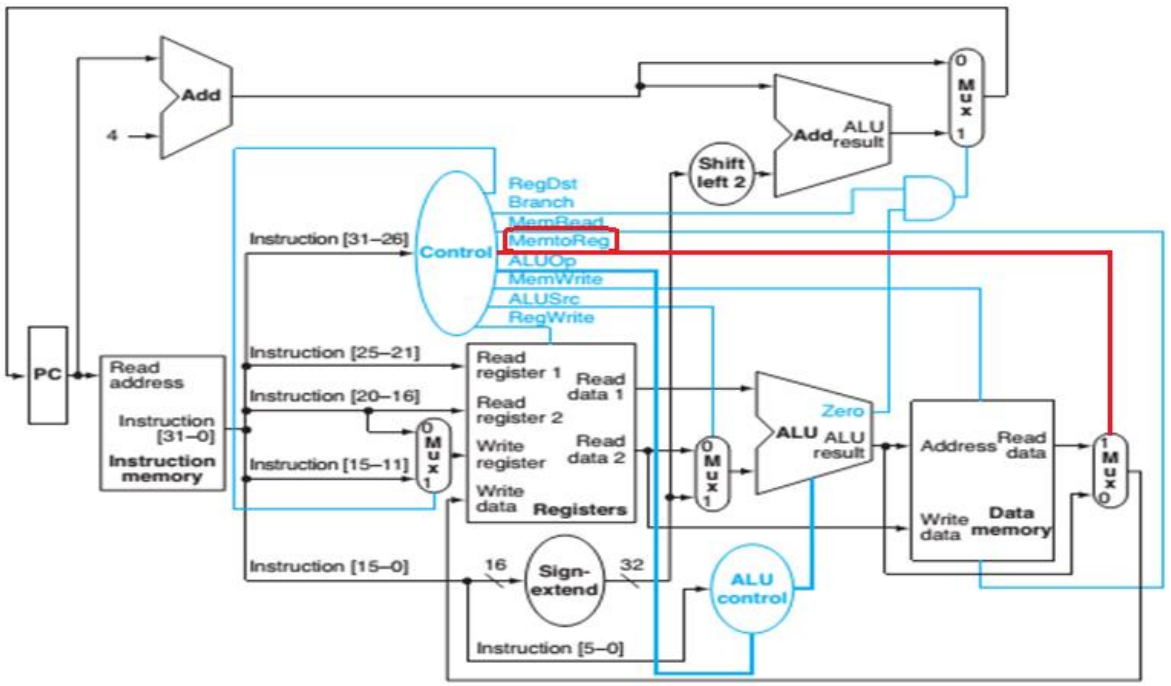
Tín hiệu MemWrite trong datapath

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)

COMPUTER ENGINEERING

Hiện thực datapath - Control (8)



Tín hiệu MemtoReg trong datapath

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

COMPUTER ENGINEERING

Hiện thực datapath - Control (9)

Tín hiệu điều khiển	Tác động khi ở mức thấp	Tác động khi ở mức cao
RegDst	Thanh ghi đích cho thao tác ghi sẽ từ thanh ghi *rt* (bits từ 20:16)	Thanh ghi đích cho thao tác ghi sẽ từ thanh ghi *rd* (bits từ 15:11)
RegWrite	Khối "Registers" chỉ thực hiện mỗi chức năng đọc thanh ghi	Ngoài chức năng đọc, khối "Register" sẽ thực hiện thêm chức năng 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"
ALUSrc	Input thứ hai cho ALU đến từ "Read data 2" của khối "Registers"	Input thứ hai cho ALU đến từ output của khối "Sign-extend"
Branch	Cho biết lệnh nạp vào không phải "beq". Thanh ghi PC nhận giá trị là PC + 4	Lệnh nạp vào là lệnh "beq", kết hợp với điều kiện bằng thông qua cổng AND nhằm xác định xem lệnh tiếp theo có nhảy đến địa chỉ mới hay không. Nếu điệu kiện bằng đúng, PC nhận giá trị mới từ kết quả của bộ cộng "Add"
MemRead	(Không)	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"
MemWrite	(Không)	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"
MemtoReg	Giá trị đưa vào ngõ "Write data" đến từ ALU	Giá trị đưa vào ngõ "Write data" đến từ khối "Data memory"

Tác động của các tín hiệu điều khiển

Hiện thực datapath - Control (10)

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.

Hiện thực datapath - Control (11)

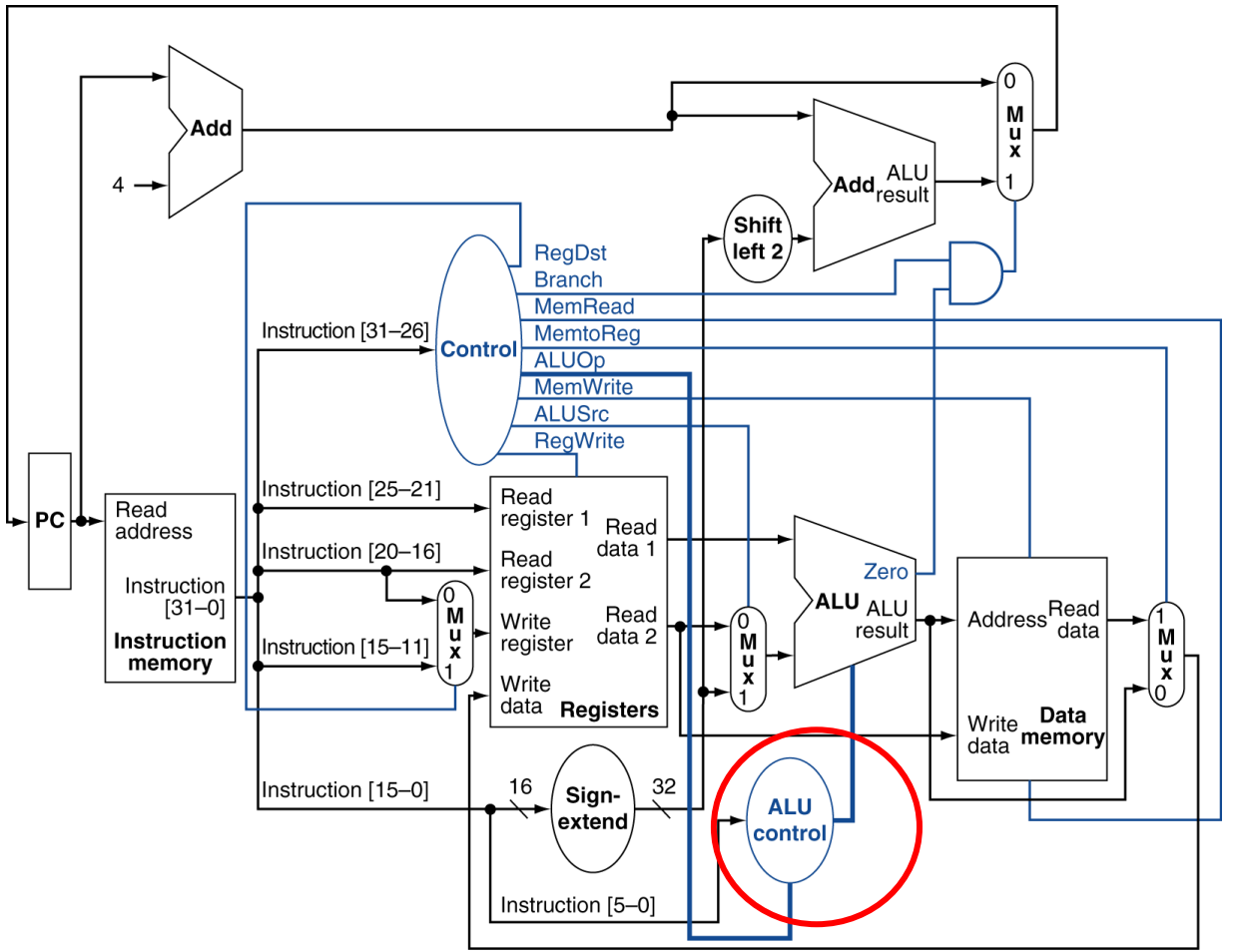
Bảng sự thật khối "Control":

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Bảng sự thật khối "Control"

COMPUTER ENGINEERING

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:

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

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 khiện bằng có đúng không.

COMPUTER ENGINEERING

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.

Hiện thực datapath - ALU Control(4)

Bảng sự thật: Từ quy tắc hoạt động, bảng sự thật gợi ý cho khối "ALU Control" như sau

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

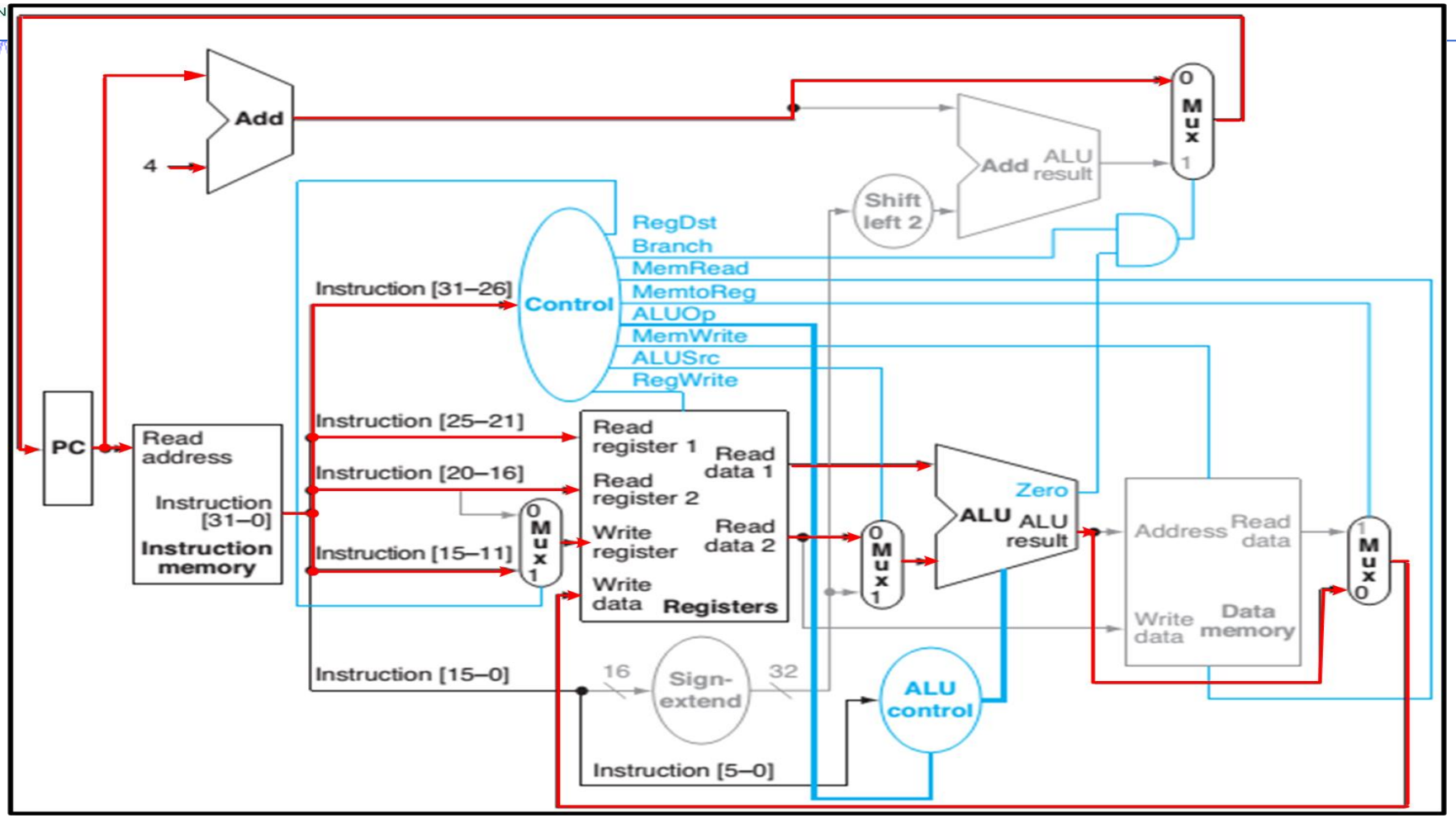
COMPUTER ENGINEERING

Hiện thực datapath

- **Hiện thực bộ xử lý đơn chu kỳ** (Single-cycle implementation hay single clock cycle implementation): là cách hiện thực sao cho bộ xử lý đáp ứng thực thi mỗi câu lệnh chỉ trong 1 chu kỳ xung clock ➔ đòi hỏi chu kỳ xung clock phải bằng thời gian của lệnh dài nhất.
- Cách hiện thực bộ xử lý như đã trình bày trên là cách hiện thực **đơn chu kỳ**:

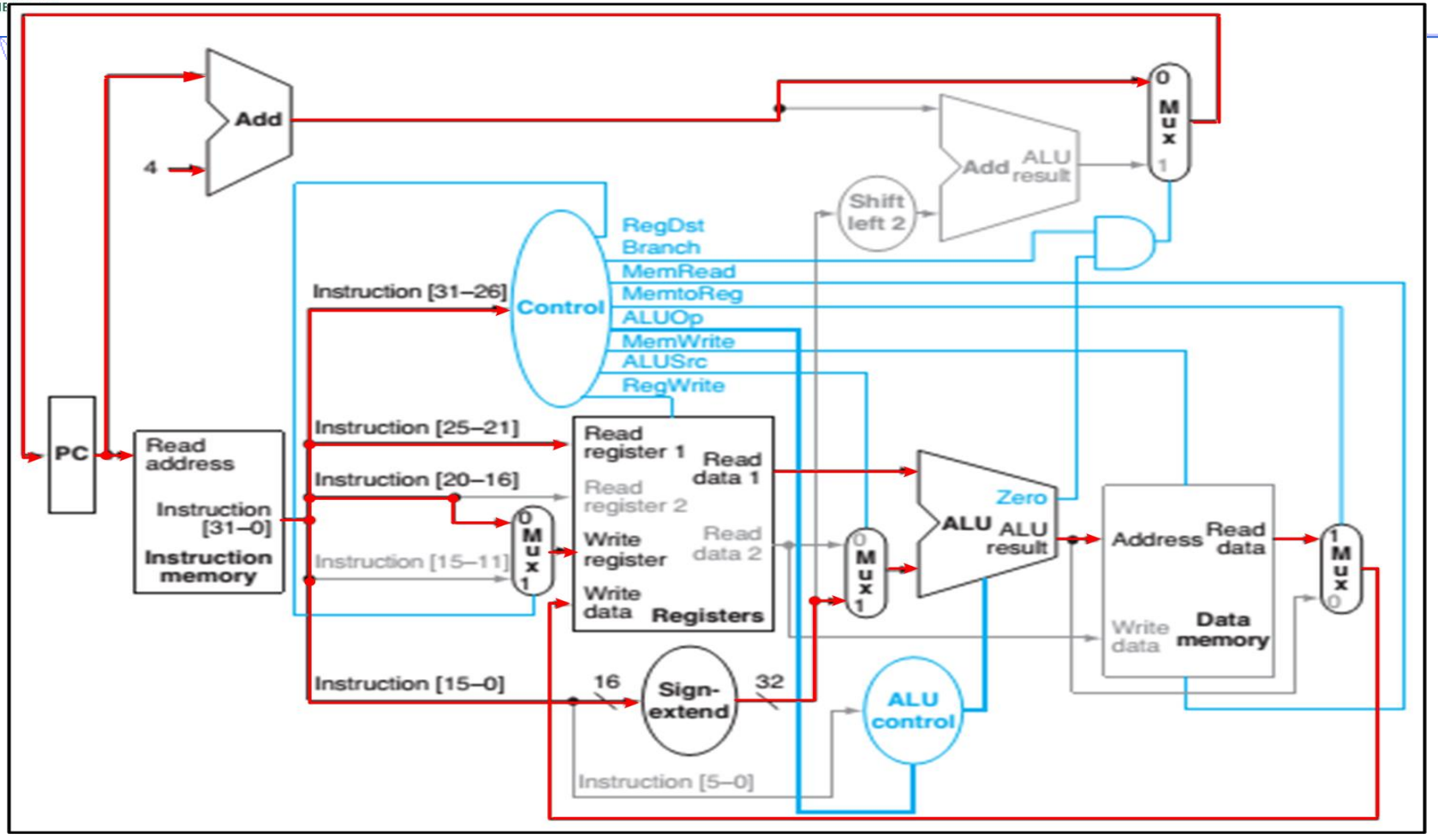
  Lệnh dài nhất là *lw,* gồm truy xuất vào "Instruction memory", "Registers", "ALU", "Data memory" và quay trở lại "Registers", trong khi các lệnh khác không đòi hỏi tất cả các công đoạn trên → chu kỳ xung clock thiết kế phải bằng thời gian thực thi lệnh *lw.*
- Mặc dù hiện thực bộ xử lý đơn chu kỳ có CPI = 1 nhưng hiệu suất rất kém, vì một chu kỳ xung clock quá dài, các lệnh ngắn đều phải thực thi cùng thời gian với lệnh dài nhất.

Xem lại Datapath với từng nhóm lệnh



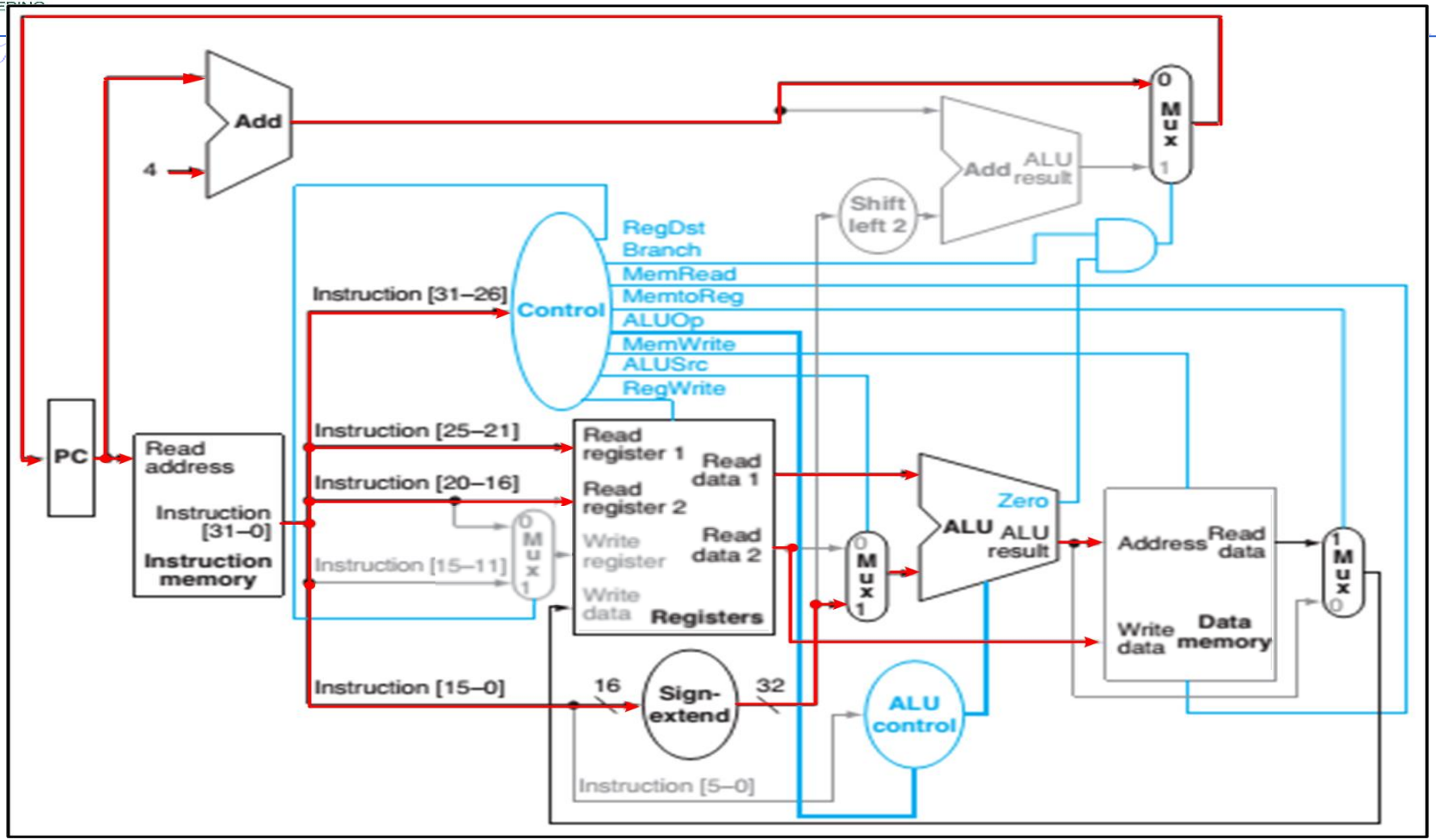
*Các đường đỏ là các đường hoạt động khi lệnh thuộc nhóm **logic và số học** thực thi*

Xem lại Datapath với từng nhóm lệnh



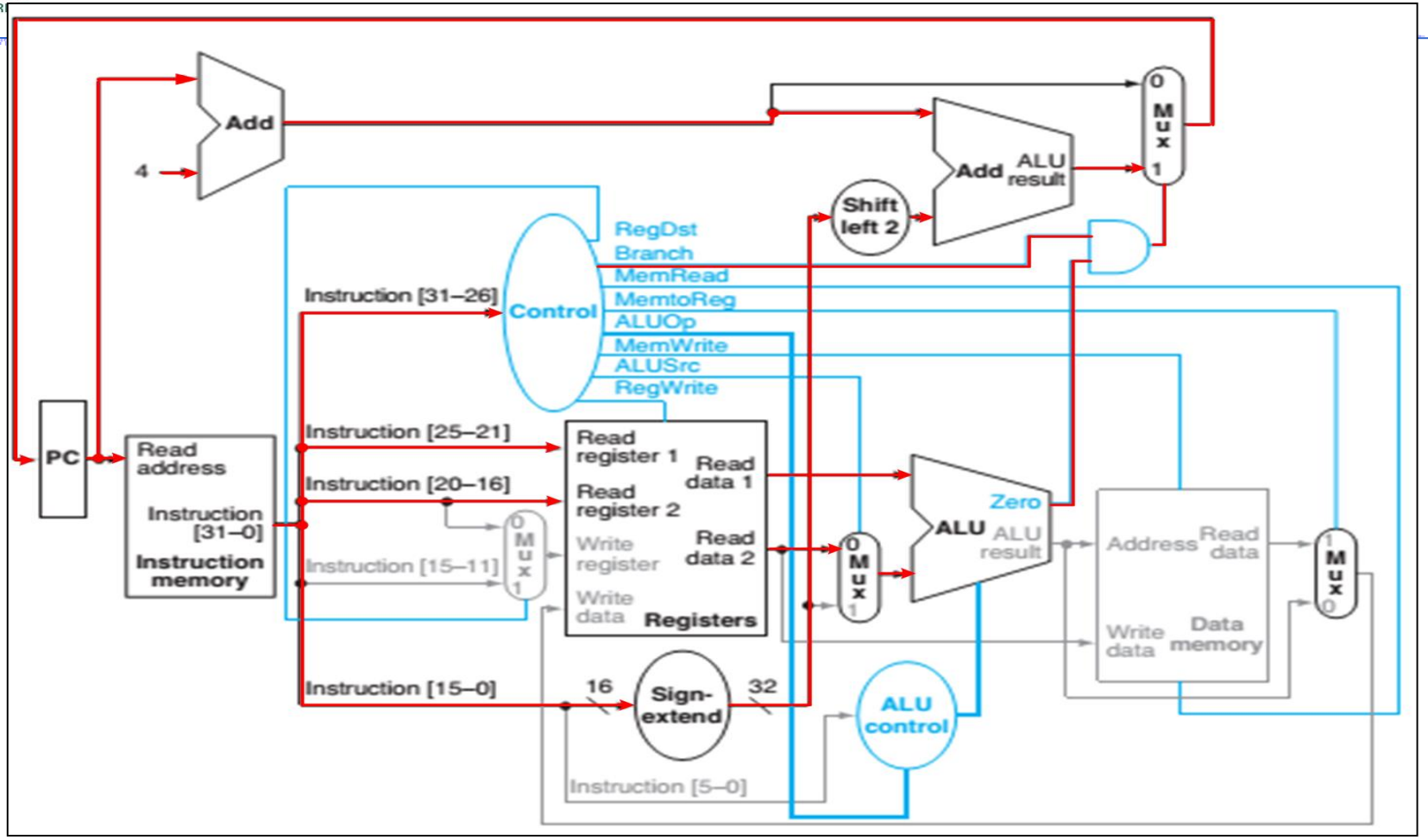
*Các đường đỏ là các đường hoạt động khi lệnh **lw** thực thi*

Xem lại Datapath với từng nhóm lệnh



*Các đường đậm nét (đỏ) là các đường hoạt động khi lệnh **sw** thực thi*

Xem lại Datapath với từng nhóm lệnh



Các đường đậm nét (đỏ) là các đường hoạt động khi lệnh ***beq*** *thực thi*

Tổng kết:

Phần này trình bày một cách thiết kế và hoàn chỉnh 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, khối Control và ALU control) 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