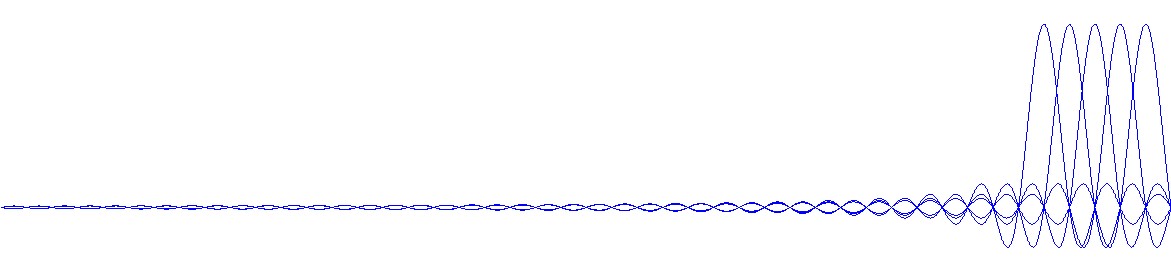
 **KIẾN TRÚC MÁY TÍNH** 

**BỘ XỬ LÝ**

**PROCESSOR**



BỘ XỬ LÝ

**Mục đích:**

üHiểu cơ chế thực thi lệnh và các quy ước về thiết kế logic

üThiết 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

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

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

|  |  |
| --- | --- |
| p Tổng số câu lệnh | Được xác định bởi trình biên dịch và kiến trúc tập lệnh |

* + Chu kỳ xung clock

Được xác định bởi quá

* + Số chu kỳ xung clock trên một lệnhtrình hiện thực bộ xử lý

(Clock cycles per instruction − CPI) v 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 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:**

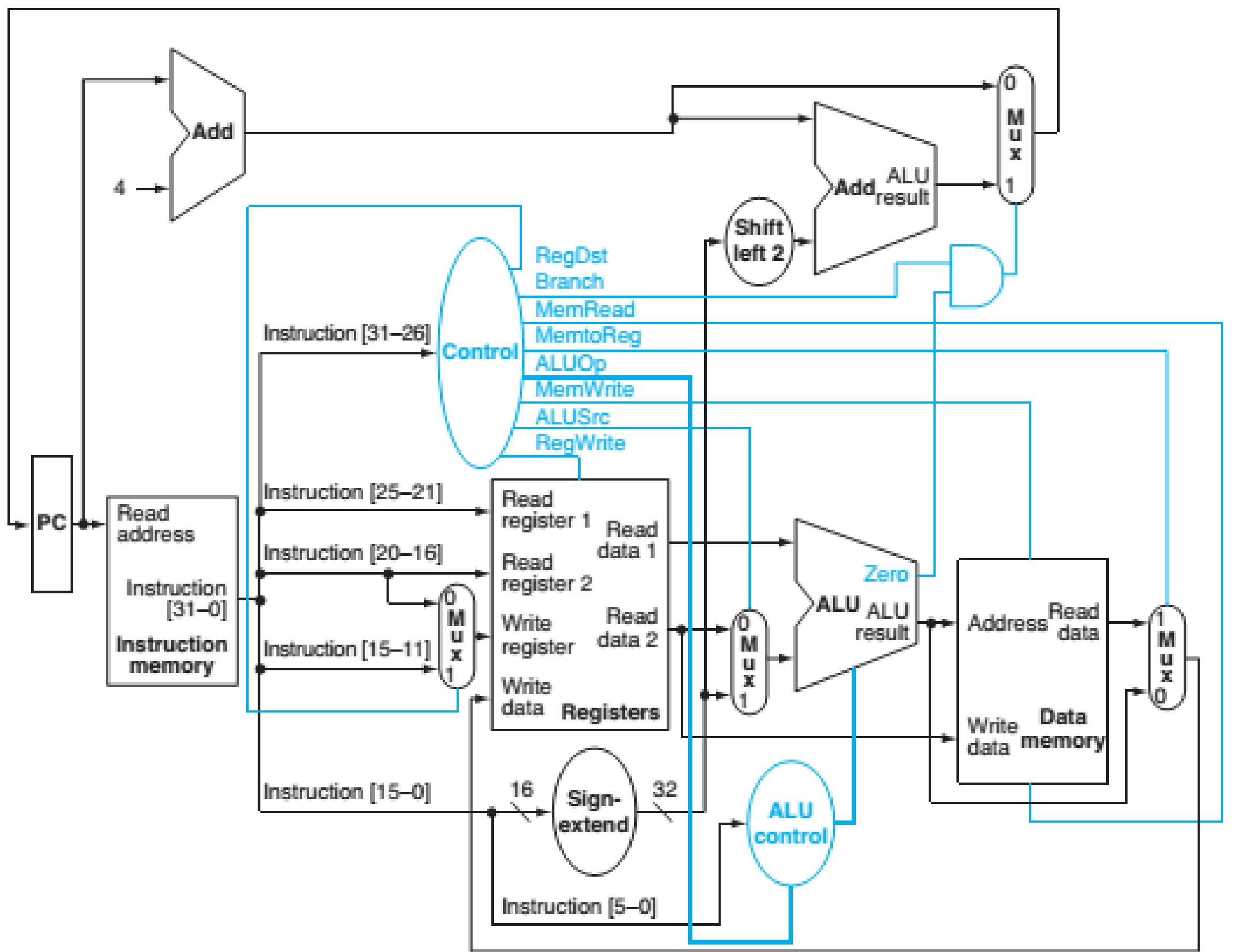
Nạp lệnh à Đọc một/hai thanh ghi à Sử dụng ALU à Truy xuất bộ nhớ để đọc/ghi dữ liệu

**Nhóm lệnh logic và số học:**

Nạp lệnh à Đọc một/hai thanh ghi à Sử dụng ALU à Ghi dữ liệu vào thanh ghi **Nhóm lệnh nhảy:**

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

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



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**

# Quy ước thiết kế

Phần này nhắc lại các khái niệm:

* **Mạch tổ hợp (Combinational): ALU** v **Mạch tuần tự (Sequential): instruction/data memories và thanh ghi** v **Tín hiệu điều khiển (Control signal)** v **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)**
* **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**

# Quy trình thực thi lệnh

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

***Instruction***

***Fetch***

***Instruction***

***Decode***

***Operand***

***Fetch***

***Execute***

***Result***

***Write***

**Next Instruction**

pNạp lệnh từ bộ nhớ (memory)

pĐịa chỉ của lệnh lưu trong thanh ghi **P**rogram

**C**ounter (PC) n **Instruction Decode (giải mã lệnh):** pTìm ra lệnh thực hiện

* **Operand Fetch (tìm nạp toán hạng):** pLấy các toán hạng cần thiết cho lệnh
* **Execute (thực thi):** pThực hiện câu lệnh
* **Result Write (lưu trữ):** pLưu 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)

|  |  |  |  |
| --- | --- | --- | --- |
|  | **add $3, $1, $2** | **lw $3, 20( $1 )** | **beq $1, $2, label** |
| **Fetch & Decode** | ***standard*** | ***standard*** | ***standard*** |
| **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* |
| **Execute** | *Result* = *opr1* + *opr2* | o *MemAddr* = *opr1 + opr2* o Sử dụng *MemAddr* để đọc dữ liệu từ bộ nhớ | *Taken* = (*opr1* == *opr2* )? *Target* = **PC** + Label\* |
| **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* |

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

## MIPS (5

n Thay đổi thiết kế các giai đoạn thực hiện lệnh:

ü Gộp giai đoạn *Decode* và *Operand Fetch* – Giai đoạn Decode của MIPS khá đơn giản ü Tách giai đoạn *Execute* thành ***ALU*** (Calculation) và ***Memory Access***

|  |  |  |  |
| --- | --- | --- | --- |
|  | **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 *opr*2 | * 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* |

## MIPS (5

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

***Fetch***

***Instruction***

***Decode***

***ALU***

***Memory***

***Access***

**Result**

**Write**

**Next Instruction**

**(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)**

## MIPS (5

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

***Fetch***

***Instruction***

***Decode***

***ALU***

***Memory***

***Access***

**Result**

**Write**

**Next Instruction**

**(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 (Instruction Fetch)

* Giai đoạn nạp lệnh:
  1. Sử dụng thanh ghi **P**rogram **C**ounter (**PC**) để tìm nạp lệnh từ **bộ nhớ**

nThanh ghi PC là một thanh ghi đặc biệt trong bộ vi xử lý

* 1. **Tăng giá trị** trong thanh ghi PC lên 4 đơn vị để lấy địa chỉ của lệnh tiếp theo nTại sao địa chỉ lệnh tiếp theo là PC + 4?

nChú 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)**

|  |  |  |
| --- | --- | --- |
| |  | | --- | | Thanh ghi PC | | **Add**  **PC**  **4**  **Read**  **address**  **Instruction**  ***Instruction***  ***memory***  Vùng  nh  ớ  l  ư  u  tr  ữ  l  ệ  nh  B  ộ  c  ộ  ng  **Decode Stage**  **Instruction** |

# Khối *Instruction Memory*

n Vùng nhớ lưu trữ lệnh n**Đầu vào:** là địa chỉ của lệnh n**Đầu ra:** là nội dung lệnh tương ứng

***Instruction***

***Memory***

**Instruction**

**Address**

**Instruction**

với địa chỉ được cung cấp

## **Instruction**

**Memory ………..** Cách sắp xếp của bộ nhớ giống như hình **2048 add $3, $1, $2**

**2052 sll $4, $3, 2**

bên phảiè **2056 andi $1, $4, 0xF …… ………..**

# Bộ cộng

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

n**Đầu vào:**

**Sum**

***Add***

**A**

**B**

**A+B**

pHai số 32-bit **A**, **B** n**Đầu ra:** p**A + B**

## Ý niệm về việc sử dụng xung clock

* Dường như thanh ghi PC được đọc và cập nhật cùng lúc: pPC hoạt động chính xác như thế nào?
* **Magic of clock**:

pPC được đọc trong nửa clock đầu và cập nhật thành PC+4 trong

lần **kích cạnh lên tiếp theo**

**Add**

**PC**

**4**

**Read**

**address**

**Instruction**

***Instruction***

***memory***

**In**

Clk

Time

**PC**

**100**

**104**

**108**

**112**

**In**

**104**

**108**

**112**

**116**

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

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

***Fetch***

***Instruction***

***Decode***

***ALU***

***Memory***

***Access***

**Result**

**Write**

**Next Instruction**

**(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

nCó thể 2 (lệnh **add**), 1 (lệnh **addi**) hoặc 0 (lệnh **j**) n Đầu vào từ giai đoạn trước (**Fetch**):

Lệnh cần được thưc thi (Mã máy)

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

Phép tính và các toán hạng cần thiết

# Giai đoạn giải mã (Decode)

**Fetch Stage**

Inst.

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Read**

**data 1**

**Read**

**data 2**

**Data**

**Register**

**Number**

***Register***

***File***

**5**

**5**

**5**

**Execute Stage**

**Operands**

Kh

ố

i

ch

ứ

a

t

ậ

p

h

ợ

p

c

ủ

a

các

thanh

ghi

,

g

ọ

i

là

**R**

**egister file**

Ho

ặ

c

**Registers**

**Write**

**data**

## **Khối *Register File***

n Một tập 32 thanh ghi:

pMỗi thanh ghi có chiều dài 32 bit và có thể được đọc hoặc

ghi bằng cách chỉ ra chỉ số của thanh ghi

p Với mỗi lệnh, cho phép **đọc nhiều nhất từ 2 thanh ghi** p Với mỗi lệnh, cho phép **ghi vào nhiều nhất 1 thanh ghi**

n**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

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

**Data**

**Data**

**Register**

**Number**

***Register***

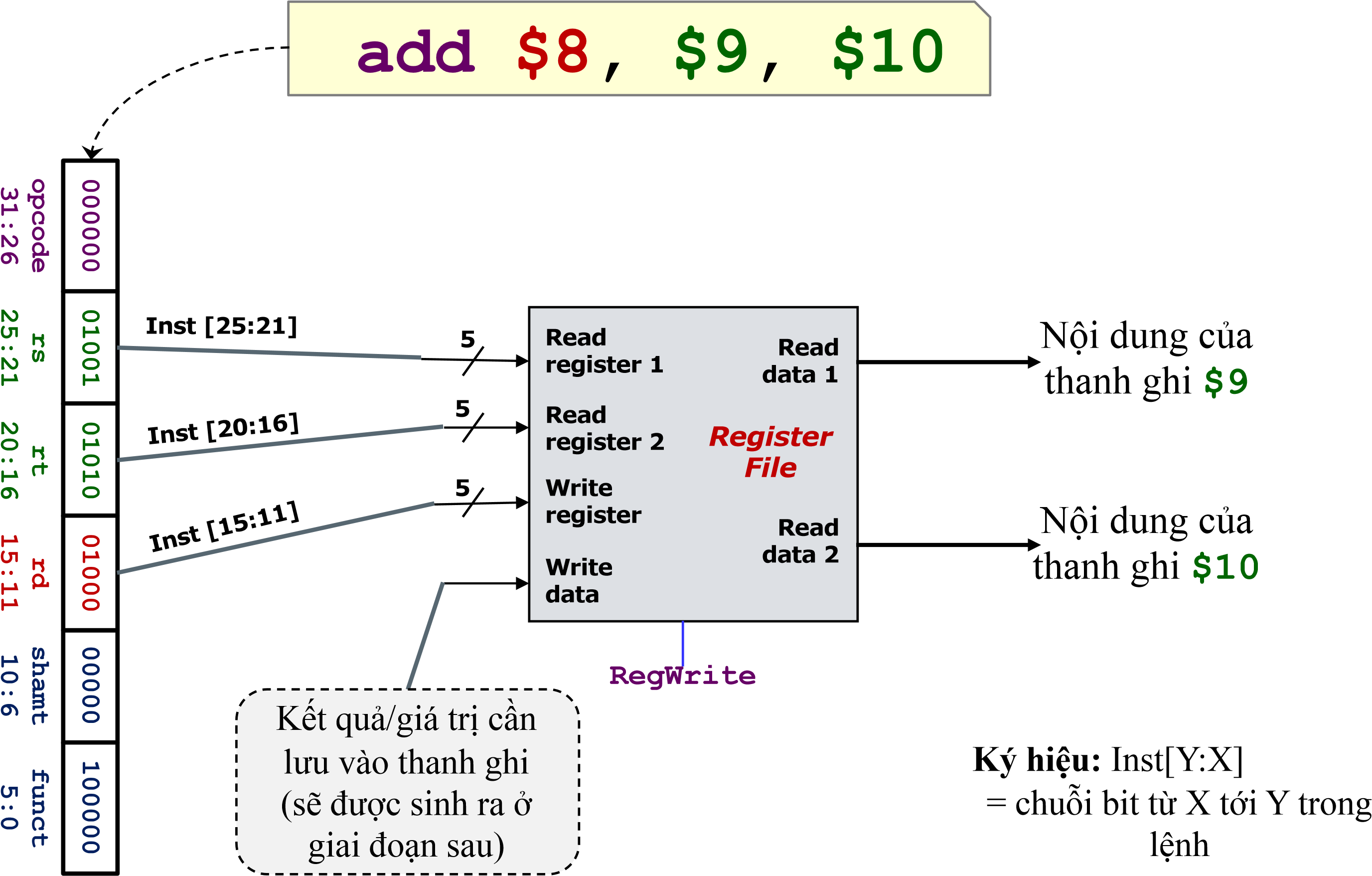
***File***

**5**

**5**

**5**

**RegWrite**

Giải mã: lệnh **R-Type**

### Giải mã: lệnh **I-Type**

**001000**

**10110**

**10101**

**1111 1111 1100 1110**

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

**RegWrite**

***Register***

***File***

**5**

**5**

**5**

N

ộ

i

dung

c

ủ

a

thanh

ghi

**$22**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Inst [25:21]**

**V**

**ấ**

**n**

**đề**

**:**

Thanh

ghi

đ

ích

**$21 “**

đặ

t

không

đ

úng

v

ị

trí

”

**lw**

**$21,**

**-**

**50($22)**

Giải mã: **Giải pháp cho ngõ “Write register”**

**001000**

**10110**

**10101**

**1111 1111 1100 1110**

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

***Register***

***File***

**5**

**5**

**5**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**RegDst**

**:**

Tín

hi

ệ

u

đ

i

ề

u

khi

ể

n

ch

ọ

n

Inst[20:16] hay [15:11]

để

đư

a

vào

ngõ

*write register*

**Gi**

**ả**

**i**

**pháp**

**(**

**cho**

**ch**

**ỉ**

**s**

**ố**

**thanh**

**ghi**

**s**

**ẽ**

**đượ**

**c**

**ghi**

**):**

S

ử

d

ụ

ng

m

ộ

t

**multiplexer**

để

l

ự

a

ch

ọ

n

ch

ỉ

s

ố

thanh

ghi

cho

ngõ

*write*

*register*

chính

xác

d

ự

a

trên

t

ừ

ng

lo

ạ

i

l

ệ

nh

**lw**

**$21,**

**-**

**50($22)**

**Multiplexer (MUX)**

* **Chức năng:**
  + Chọn một input từ tập input đầu vào
* **Inputs:** p ***n*** đường vào có cùng chiều rộng
* **Control:** p ***Cần m*** bit trong đó n = 2m n **Output:** 
  + Chọn đường input thứ i nếu giá trị tín hiệ

in

0

in

n

-

1

out

Control

m

**.**

**.**

**.**

**M**

**U**

**X**

Control=0 à select in0

Control=3 à select in3

u điều khiển control = i

# Giải mã: giải pháp cho ngõ “Data 2”

**001000**

**10110**

**10101**

**1111 1111 1100 1110**

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

**5**

**5**

**5**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**Inst [**

**15:0]**

**M**

**U**

**X**

**ALUSrc**

**RegWrite**

**Sign**

**Extend**

**16**

**32**

***Register***

***File***

**ALUSrc**

**:**

Tín

hi

ệ

u

đ

i

ề

u

khi

ể

n

để

ch

ọ

n

“Read data

2

” hay

giá

tr

ị

c

ủ

a

Inst[15:0] (

đ

ã

đượ

c

m

ở

r

ộ

ng

có

d

ấ

u

)

cho

toán

h

ạ

ng

th

ứ

hai

**Gi**

**ả**

**i**

**pháp**

**(**

**cho**

**đườ**

**ng**

**d**

**ữ**

**li**

**ệ**

**u**

**data 2)**

S

ử

d

ụ

ng

m

ộ

t

**multiplexer**

để

ch

ọ

n

chính

xác

toán

h

ạ

n

g

th

ứ

2.

**Sign extend**

:

kh

ố

i

m

ở

r

ộ

ng

s

ố

t

ứ

c

th

ờ

i

16

bit

thành

32

bit

**lw**

**$21,**

**-**

**50($22)**

## Giải mã: Lệnh Load Word

**100011**

**10110**

**10101**

**1111 1111 1100 1110**

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

**5**

**5**

**5**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**Inst [**

**15:0]**

**M**

**U**

**X**

**ALUSrc**

**RegWrite**

Sign

Extend

**16**

**32**

***Register***

***File***

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

## Giải mã: Lệnh nhánh/nhảy

n Ví dụ: "**beq $9, $0, 3**" ü Cần tính kết quả rẽ nhánh và đích đến cùng một lúc ! ü Giải quyết vấn đề này trong giai đoạn của ALU

**000100**

**01001**

**00000**

**0000 0000 0000 0011**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

**5**

**5**

**5**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**Inst [**

**15:0]**

**M**

**U**

**X**

**ALUSrc**

**RegWrite**

Sign

Extend

**16**

**32**

***Register***

***File***

## Giải mã: tổng kết

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Write**

**data**

**Read**

**data 1**

**Read**

**data 2**

***Registers***

**5**

**5**

**5**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**Inst [**

**15:0]**

**M**

**U**

**X**

**ALUSrc**

**RegWrite**

Sign

Extend

**16**

**32**

Operand 1

Operand 2

Inst[31:0]

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

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

***Fetch***

***Instruction***

***Decode***

***ALU***

***Memory***

***Access***

**Result**

**Write**

**Next Instruction**

**(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

n 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 n Đầu vào từ công đoạn trước (**Decode**):
* Các thao tác (operation) và toán hạng (operand(s)) n 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

**ALU**

**result**

***ALU***

**Decode Stage**

Operands

**Memory Stage**

Kh

ố

i

m

ạ

ch

th

ự

c

hi

ệ

n

các

phép

toán

logic

và

s

ố

h

ọ

c

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

* **ALU (Arithmetic-logical unit**) ü Khối dùng để thực hiện các phép tính logic và số học

**ALU**

**result**

***ALU***

**ALUcontrol**

**4**

**isZero**

**?**

**A**

**B**

**A op B**

**(**

**A op B) == 0?**

* **Inputs:** 
  + 2 số 32-bit n **Điều khiển khối ALU:**

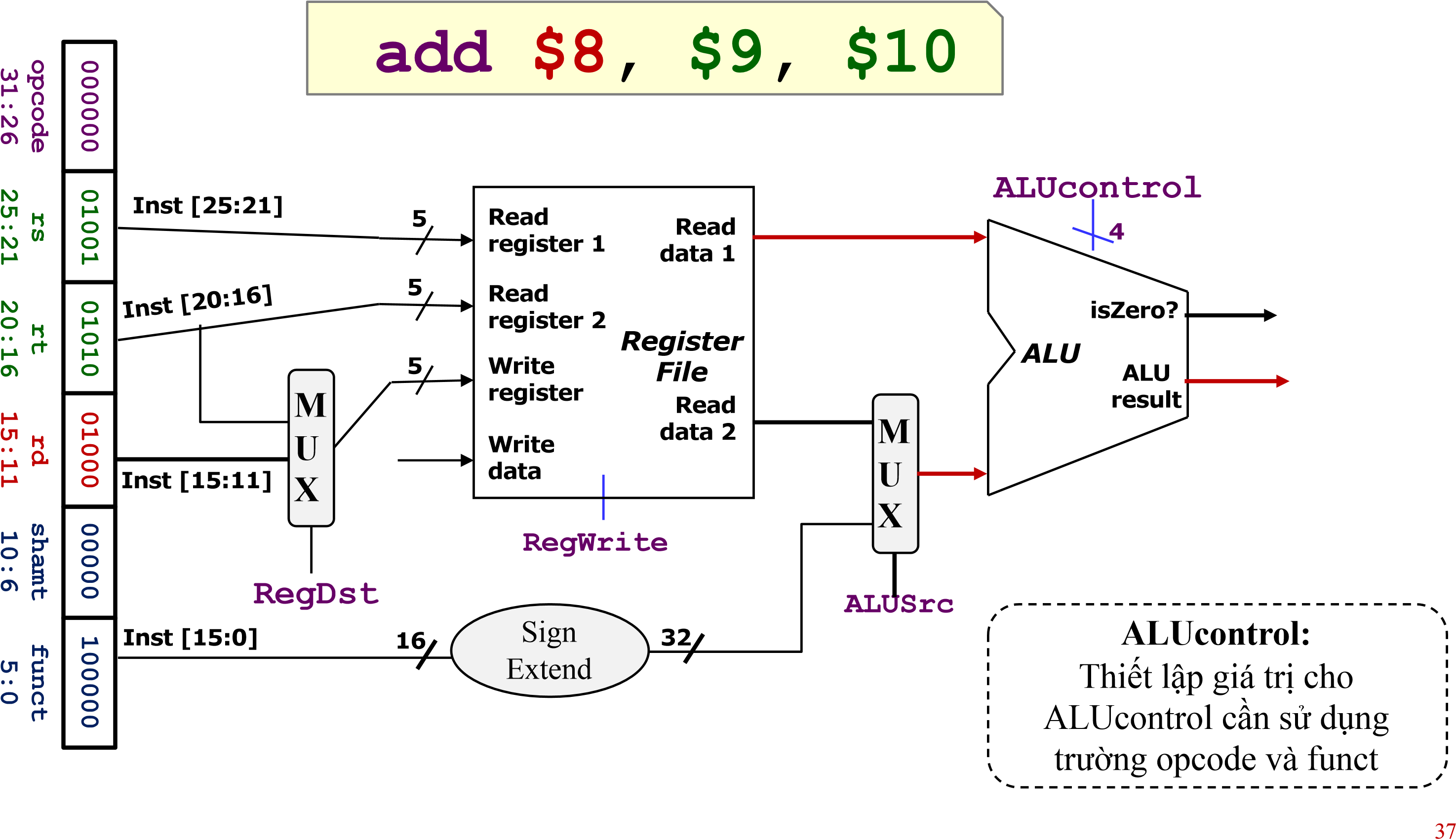
|  |  |
| --- | --- |
| **ALUcontrol** | **Function** |
| **0000** | **AND** |
| **0001** | **OR** |
| **0010** | **add** |
| **0110** | **subtract** |
| **0111** | **slt** |
| **1100** | **NOR** |

* + Do ALU có thể thực hiện nhiều chức năng à 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

**36**

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

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



### Công đoạn ALU: Các lệnh *Branch*

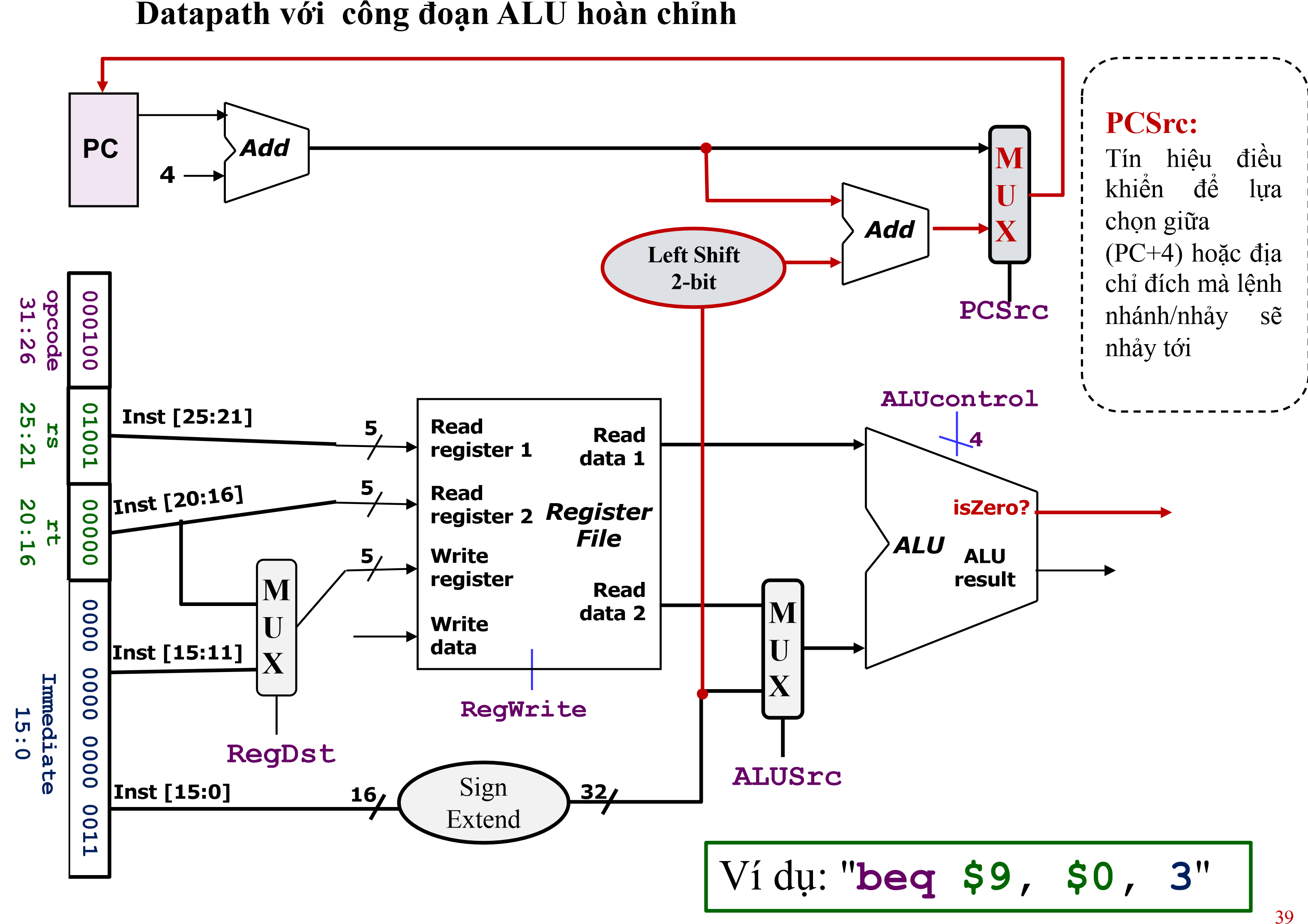
* Lệnh rẽ nhánh thì khó hơn vì phải tính toán hai phép toán:
* Ví dụ: "**beq $9, $0, 3**"
  1. **Kết quả rẽ nhánh:**

üSử dụng ALU để so sánh thanh ghi

üTín hiệu 1-bit "**isZero?**“ để kiểm tra tính chất bằng/không bằng

* 1. **Địa chỉ đích của nhánh:**

üSử dụng một bộ cộng để tính địa chỉ üCần nội dung của thanh ghi PC (từ Fetch Stage) üCần Offset (từ Decode Stage)



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

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

***Fetch***

***Instruction***

***Decode***

***ALU***

***Memory***

***Access***

**Result**

**Write**

**Next Instruction**

**(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:

üChỉ 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 üTất cả các lệnh khác sẽ rảnh trong giai đoạn này
* Đầu vào từ giai đoạn trước (**ALU**):

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

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

üKết quả được lưu trữ lại (nếu cần)

41

#### Giai đoạn truy xuất vùng nhớ (Memory stage)

**ALU Stage**

Result

**Result Store Stage**

B

ộ

nh

ớ

l

ư

u

tr

ữ

d

ữ

li

ệ

u

***Data***

***Memory***

**Address**

**Read**

**Data**

**Write**

**Data**

**MemRead**

**MemWrite**

## Khối ***Data Memory***

* Vùng nhớ này lưu trữ dữ liệu cần thiết của chương trình
* **Inputs: MemWrite** üAddress: Địa chỉ vùng nhớ üWrite Data: Dữ liệu sẽ được ghi vào vùng nhớ đối với lệnh Store

***Data***

***Memory***

**Address**

**Read**

**Data**

**Write**

**Data**

* **Tín hiệu điều khiển:**

üTín hiệu đọc (MemRead ) và ghi

(MemWrite); chỉ một tín hiệu được bật lên **MemRead**

tại bất kì một thời điểm nào

* **Output:**

üDữ liệu được đọc từ vùng nhớ đối với lệnh

Load

**43**

### Giai đoạn Memory: lệnh ***Load***

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

**000100**

**01001**

**00000**

**0000 0000 0000 0011**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**Inst [**

**15:0]**

**M**

**U**

**X**

**ALUSrc**

**RR1**

**RR2**

**WR**

**WD**

**RD1**

**RD2**

***Registers***

**5**

**5**

**5**

**RegWrite**

Sign

Extend

**16**

**32**

**ALU**

**result**

***ALU***

**ALUcontrol**

**4**

**100011**

**10110**

**10101**

**1111 1111 1100 1110**

**Address**

**Write**

**Data**

**MemRead**

**MemWrite**

***Data***

***Memory***

**Read**

**Data**

**lw**

**$21**

**,**

**-**

**50(**

**$22**

**)**

### Giai đoạn Memory: lệnh ***Store***

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

**000100**

**01001**

**00000**

**0000 0000 0000 0011**

**Immediate**

**15:0**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**RegDst**

**Inst [**

**15:0]**

**M**

**U**

**X**

**RR1**

**RR2**

**WR**

**WD**

**RD1**

**RD2**

***Registers***

**5**

**5**

**5**

**RegWrite**

Sign

Extend

**16**

**32**

**ALU**

**result**

***ALU***

**ALUcontrol**

**4**

**101011**

**10110**

**10101**

**1111 1111 1100 1110**

**Address**

**Write**

**Data**

**MemRead**

**MemWrite**

***Data***

***Memory***

**Read**

**Data**

**sw**

**$21**

**,**

**-**

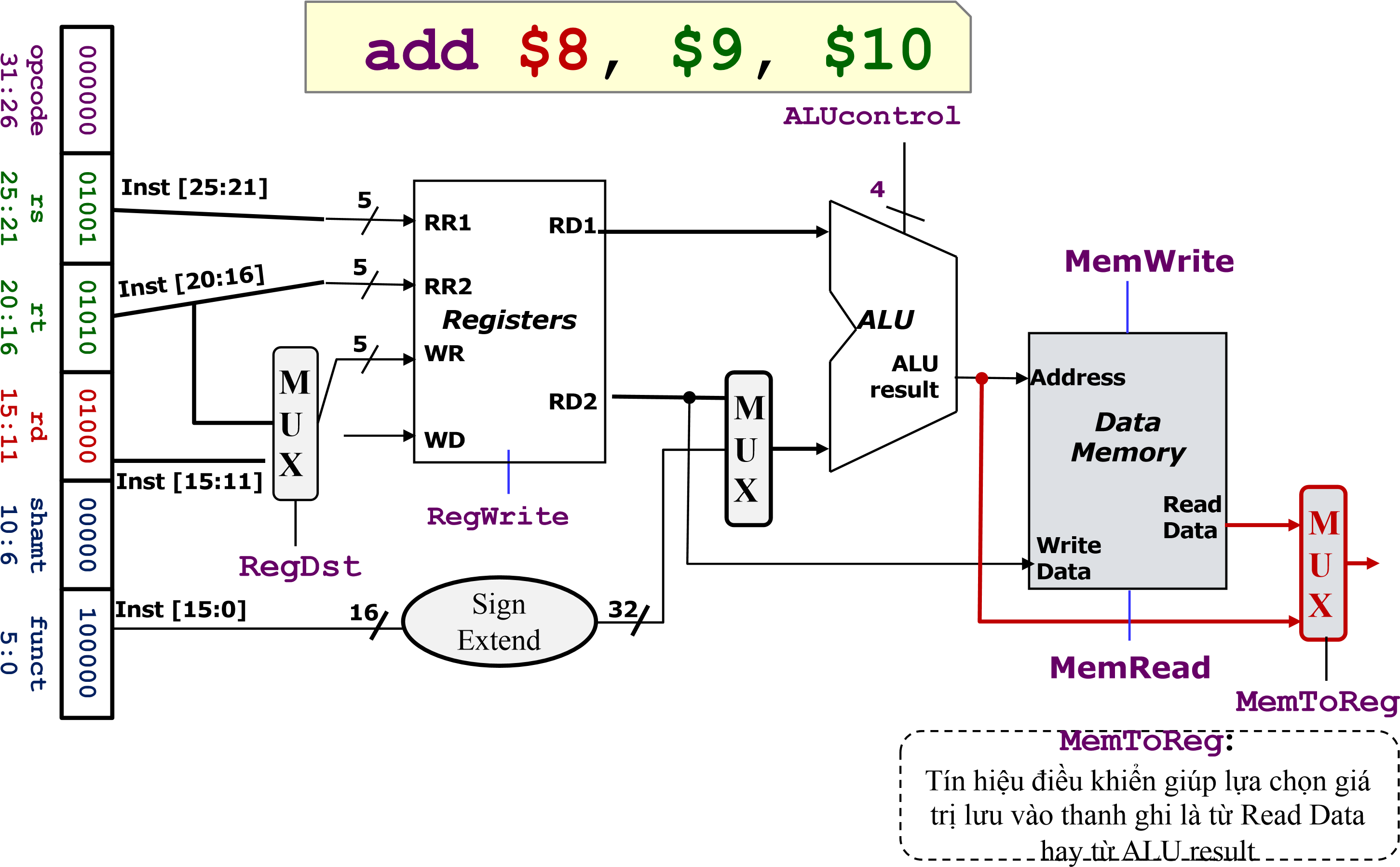
**50(**

**$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



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

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

***Fetch***

***Instruction***

***Decode***

***ALU***

***Memory***

***Access***

**Result**

**Write**

**Next Instruction**

**(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 Result Write:** üNhững 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 üNhững lệnh không ghi kết quả như: store, branch, jump:
* Không có ghi kết quả

èNhững lệnh này sẽ rảnh trong giai đoạn này n Đầu vào từ giai đoạn trước (**Memory**):

üKết quả tính toán hoặc là từ Memory hoặc là từ ALU

**Memory Stage**

Result

**Read**

**register 1**

**Read**

**register 2**

**Write**

**register**

**Read**

**data 1**

**Read**

**data 2**

***Registers***

**5**

**5**

**5**

**Write**

**data**

n 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**

#### **add $8, $9, $10**

**Inst [25:21]**

**Inst [15:11]**

**M**

**U**

**X**

**Inst [**

**15:0]**

**M**

**U**

**X**

**RR1**

**RR2**

**WR**

**WD**

**RD1**

**RD2**

***Registers***

**5**

**5**

**5**

**RegWrite**

Sign

Extend

**ALU**

**result**

***ALU***

**ALUcontrol**

**4**

***Data***

***Memory***

**Address**

**Read**

**Data**

**Write**

**Data**

**MemWrite**

**opcode**

**31:26**

**rs**

**25:21**

**rt**

**20:16**

**rd**

**15:11**

**shamt**

**10:6**

**funct**

**5:0**

**000000**

**01001**

**01010**

**01000**

**00000**

**100000**

**M**

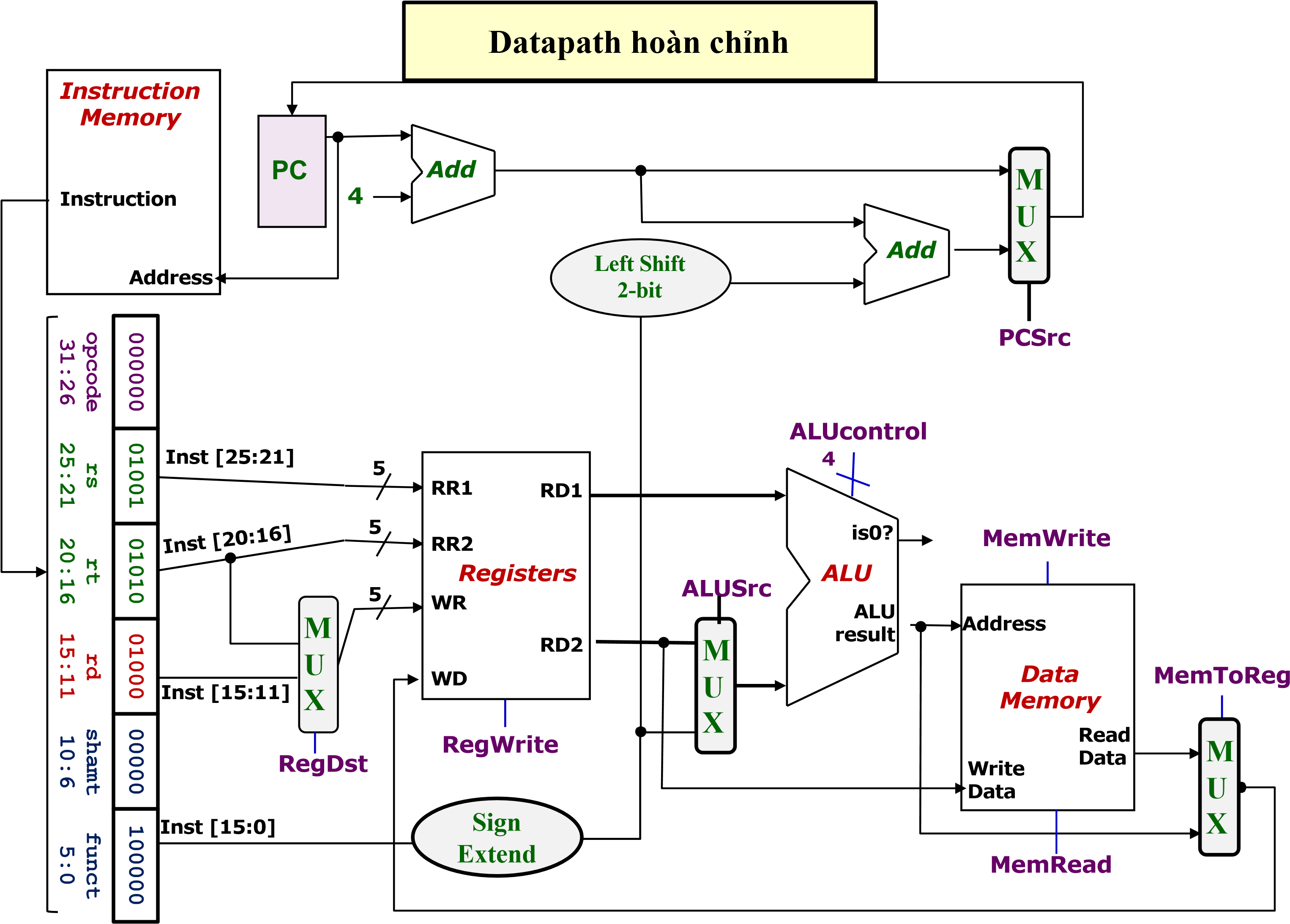
**U**

**X**

**MemToReg**

**MemRead**

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



Copyrights 2017 CE-UIT. All Rights Reserved. 51

BỘ XỬ LÝ

**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.

Copyrights 2017 CE-UIT. All Rights Reserved. 52

BỘ XỬ LÝ

1. **Giới thiệu**
2. **Nhắc lại các quy ước thiết kế logic**

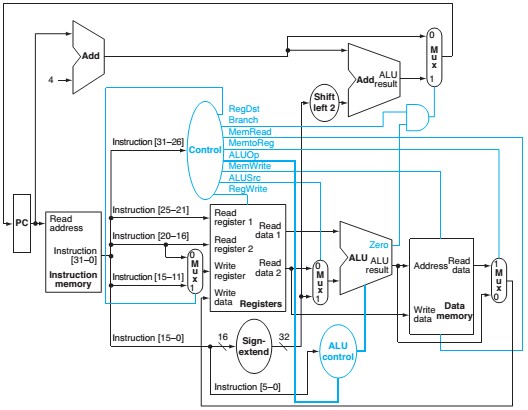
##### 3. Xây dựng đường dữ liệu (datapath) đơn giản

**4. Hiện thực datapath đơn chu kỳ**

53 Copyrights 2017 CE-UIT. All

Rights Reserved.

**1. Inputs của khối “Registers”, “Control” và “Sign-extend”**



Datapath với đầy đủ dữ54liệu input cho từng khối

**1. Inputs của khối “Registers”, “Control” và “Sign-extend”**

55

Datapath

v

ớ

i

đầ

y

đủ

d

ữ

li

ệ

u

input

cho

t

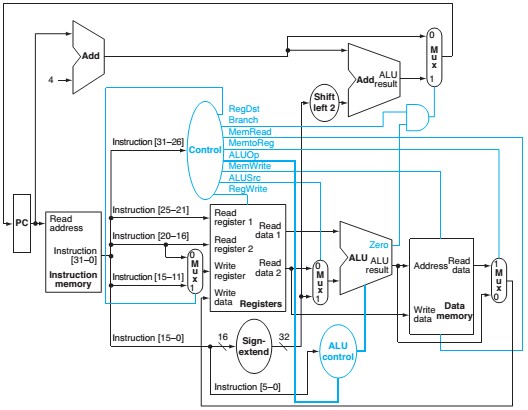
ừ

ng

kh

ố

i



???

?

MUX

có

thêm

0

và

1

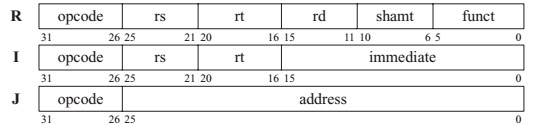
ở

các

ngõ

vào

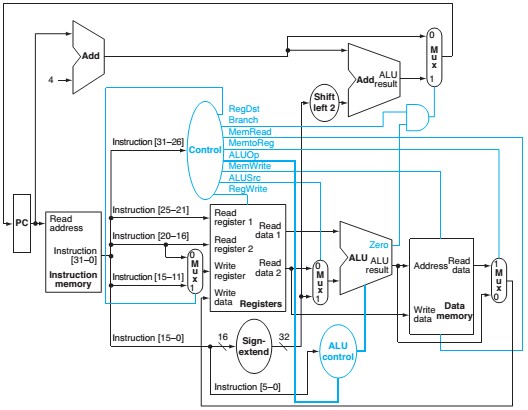
???



vTrường op (hay opcode) luôn chứa bits từ 31:26.

vHai thanh ghi dùng để đọc trong tất cả các lệnh luôn luôn là rs và rt, tại vị trí bits từ 25:21 và 20:26. vThanh ghi nền cho lệnh load và store luôn là *rs* và tại vị trí bits 25:21. v 16 bits offset cho *beq, lw* và *sw* thì luôn tại vị trí 15:0.

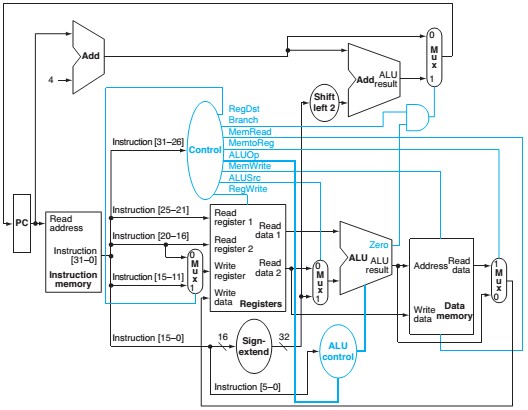
v Các thanh ghi đích dùng để ghi kết quả vào ở hai vị trí: Với *lw*, thanh ghi đích tại vị trí bits từ **20:16 (*rt*),** trong khi với nhóm lệnh logic và số học, thanh ghi đích ở vị trí **15:11 (*rd*).** Vì vậy, một multiplexer cần sử dụng ở đây để lựa chọn thanh ghi nào sẽ được ghi.



*Datapath với đầy đủ dữ liệu input cho từng khối*

57

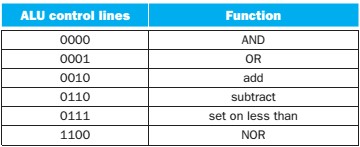
1. **Khối “ALU Control”**



???

?

Bộ ALU của MIPS gồm 6 chức năng tính toán dựa trên 4 bits điều khiển đầu vào:

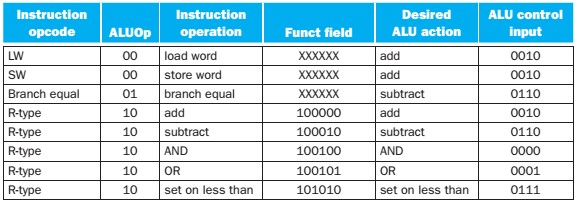


Tùy thuộc vào từng nhóm lệnh mà ALU sẽ thực hiện 1 trong 5 chức năng đầu (NOR sẽ được dùng cho các phần khác)

* + Với các lệnh **loadword** và **storeword**, ALU sử dụng chức năng **‘add’** để tính toán địa chỉ của bộ nhớ
  + Với các lệnh thuộc **nhómlogicvàsố 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 **nhảynếubằng**, ALU thực hiện chức năng **‘subtract’** để xem điều khiện bằng có đúng không.

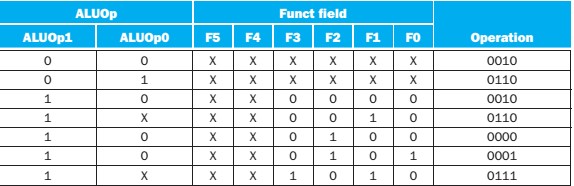
Như vậy, để sinh ra 4 bits điều khiển ALU, một trong số các cách hiện thực có thể là sử dụng thêm một khối điều khiển “ALU Control”

“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:

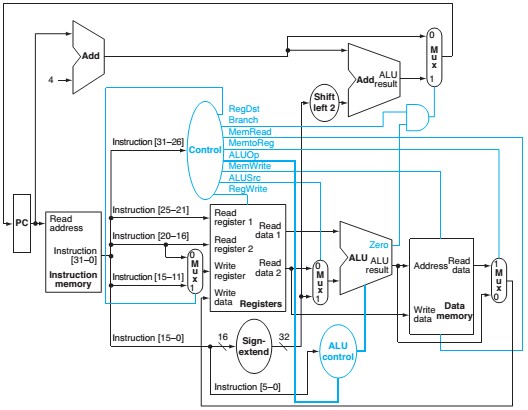


*Một gợi ý để sinh ra 4 bits điều khiển ALU dựa vào trường “opcode” và trường “funct” của mã máy.*

**Bảngsự thật: Từ quytắchoạt động, bảngsự thậtgợi ý chokhối“ALU Control” như sau**



1. **Khối điều khiển chính “Control”**

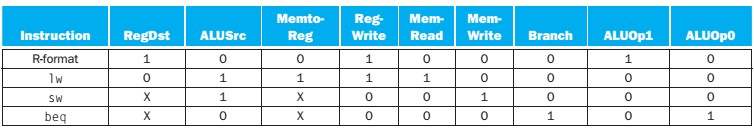


???

|  |  |  |
| --- | --- | --- |
| **Các 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 register” 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 register” 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*63 *hiệu điều khiển*

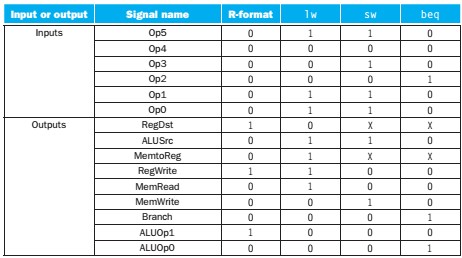
**Giá trị các tín hiệu điều khiển tương ứng với mỗi lệnh như sau:**



*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.*

Copyrights 2017 CE-UIT. All Rights Reserved.

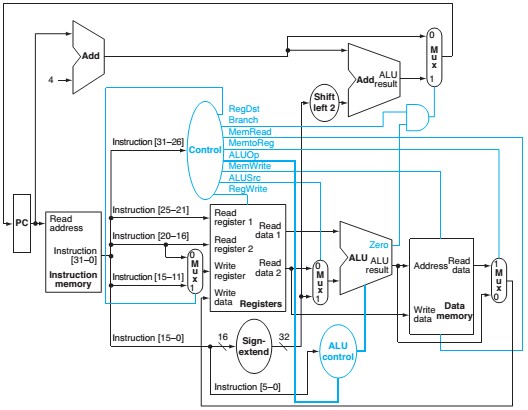
**Bảng sự thật khối “Control”:**



*Bảng sự thật khối “Control”*

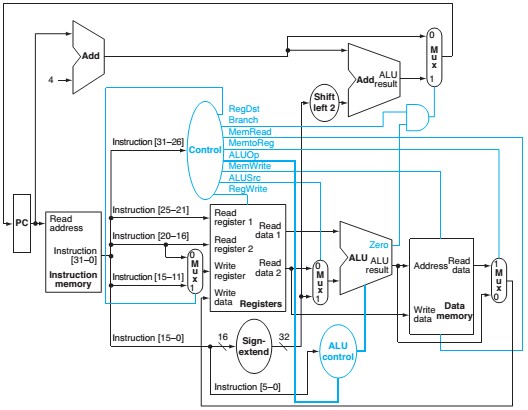
Copyrights 2017 CE-UIT. All Rights Reserved.

**3. Khối điều khiển chính “Control”**



???

**3. Khối điều khiển chính “Control”**



* **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 è đ i 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**:

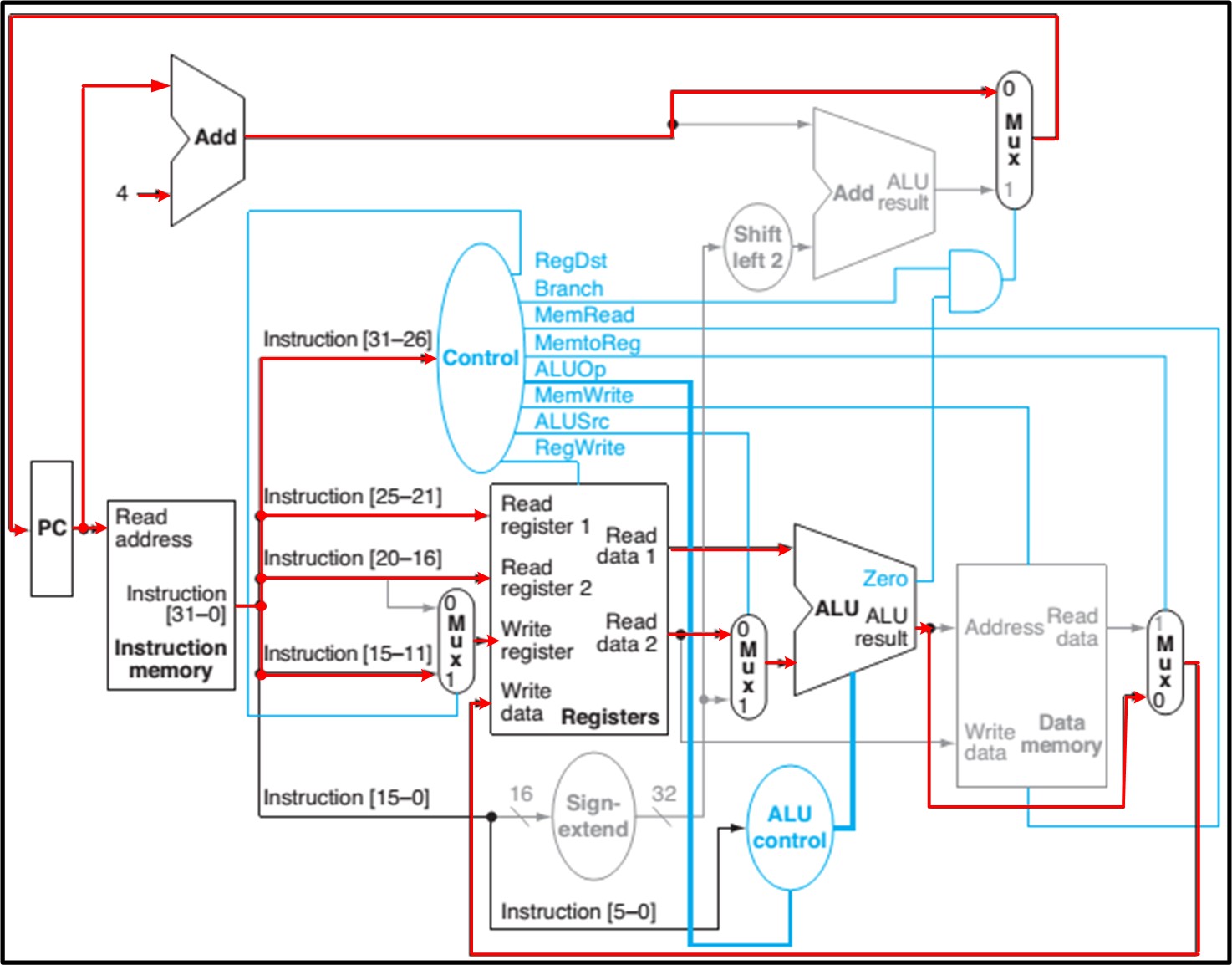
Lệnh dài nhất là *lw,* gồm truy xuất vào “Instruction memory”, “Registers”, “ALU”,

“Data memory” và quay trở lại “Registers”, trong khi các lệnh khác không đ i hỏi tất cả các công đoạn trên à chu kỳ xung clock thiết kế phải bằng thời gian thực thi lệnh *lw.*

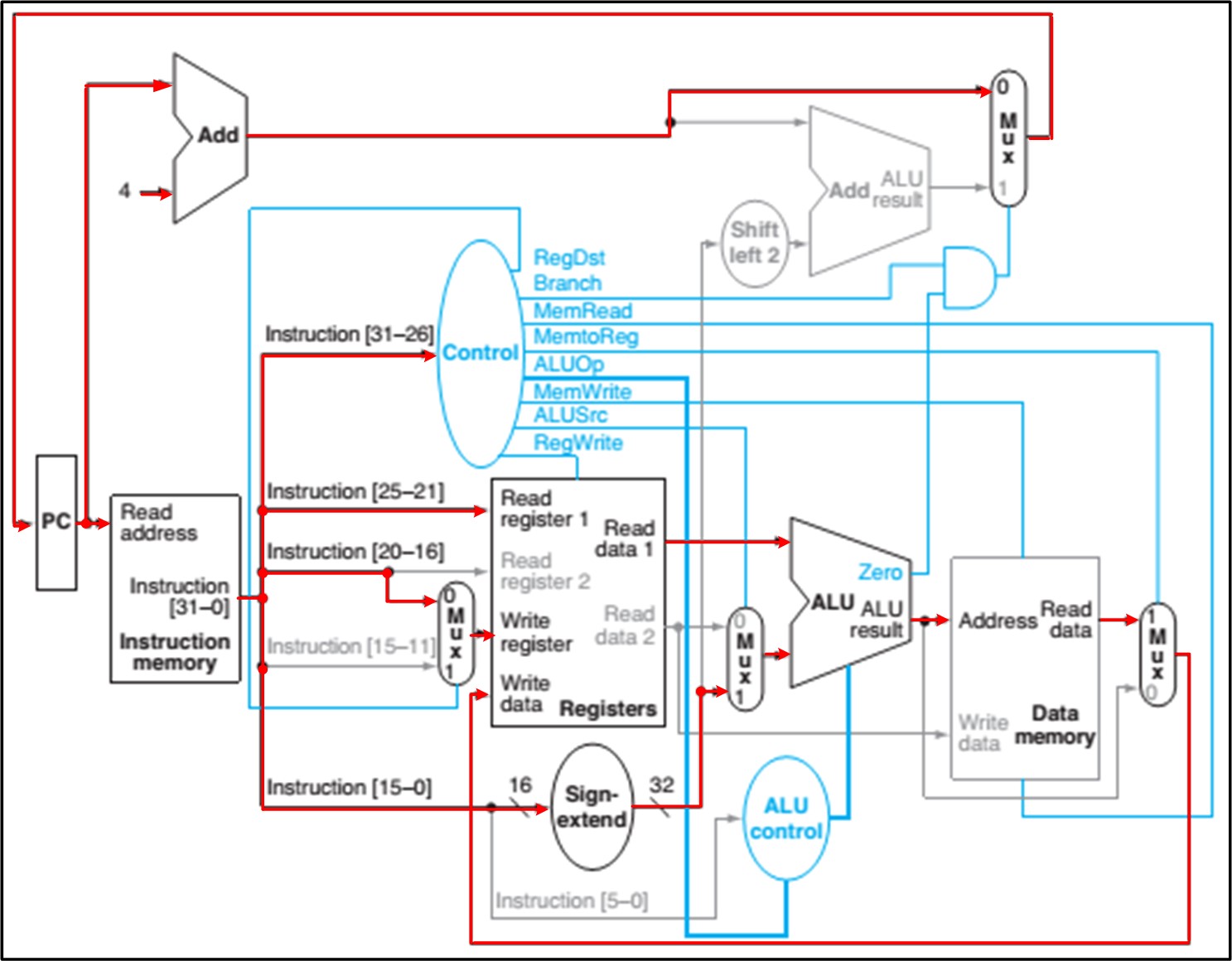
* Mặc dù hiện thực bộ xử lý đơn chu kỳ có CPI = 1 nhưng hiệu suất rất kém, vì một chu kỳ xung clock quá dài, các lệnh ngắn đều phải thực thi cùng thời gian với lệnh dài nhất.

Vì vậy, **Hiện thực đơn chu kỳ hiện tại không c n được sử dụng (hoặc chỉ có thể chấp nhận cho các tập lệnh nhỏ)**

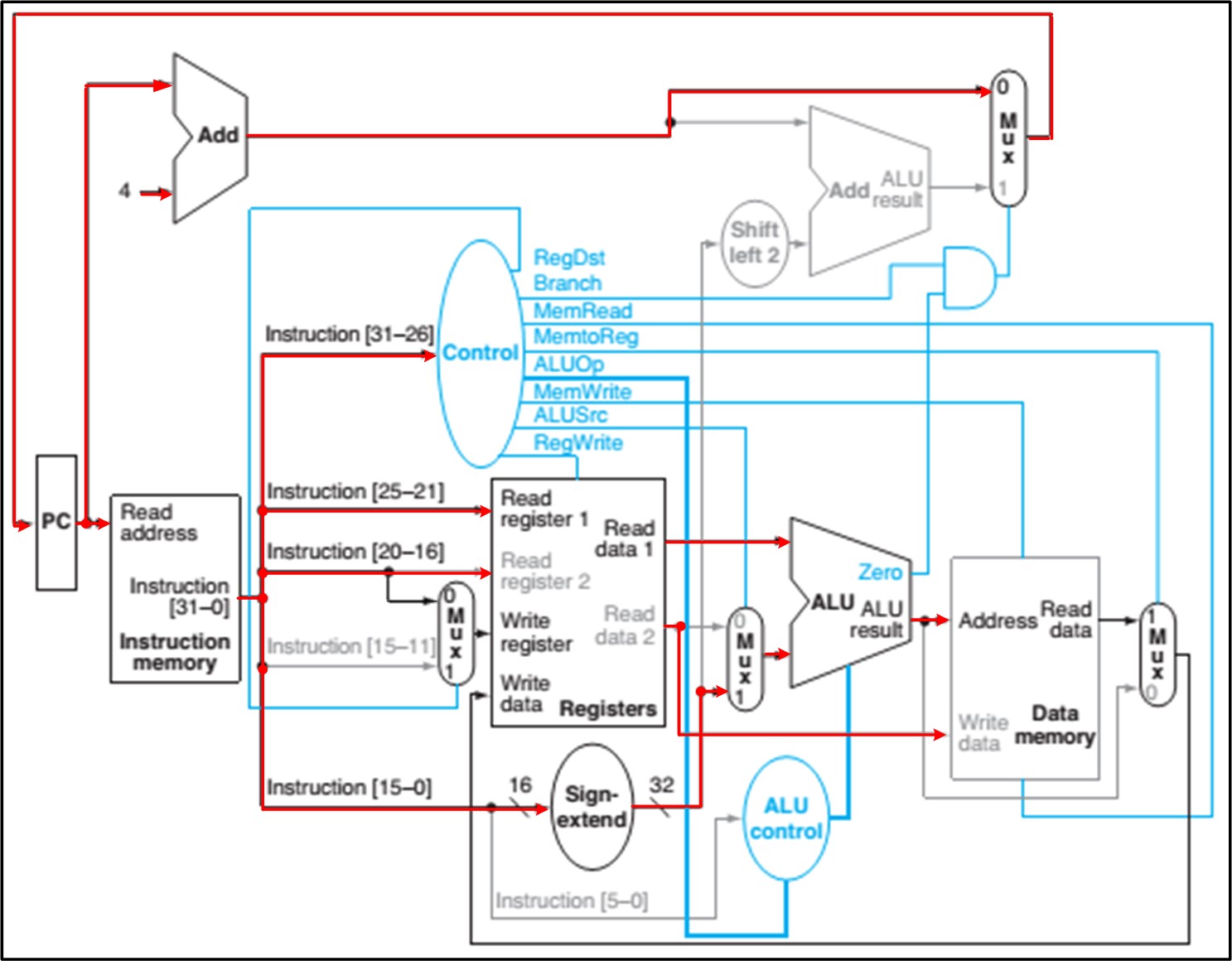
Copyrights 2017 CE-UIT. All Rights Reserved.



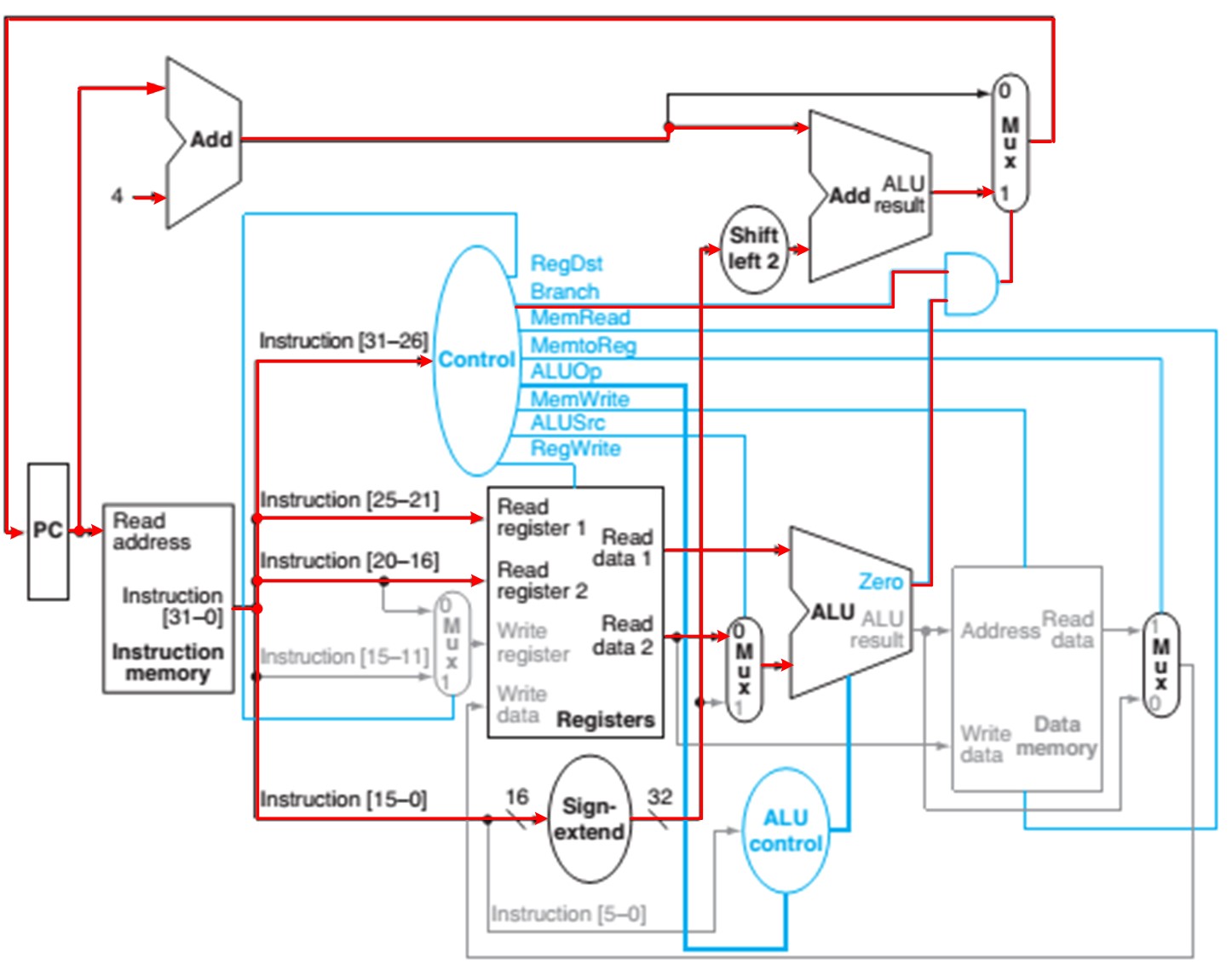
*Các đường đỏ là các đường hoạt động khi*69*lệnh thuộc* ***nhóm logic và số học*** *thực thi*



*Các đường đỏ là các đường ho*70 *ạt động khi lệnh* ***lw*** *thực thi*



*Các đường đậm nét (đỏ) là các đườ*71 *ng hoạt động khi lệnh* ***sw*** *thực thi*



*Các đường đậm nét (đỏ) là các đườ*72*ng hoạt động khi lệnh* ***beq*** *thực thi*

BỘ XỬ LÝ

**Tổng kết:**

Hoàn chỉnh datapath cho 8 lệnh cơ bản:

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

Hiểu cách hiện thực các khối chức năng cơ bản trong datapath

(khối Control, khối ALU Control)

Copyrights 2017 CE-UIT. All Rights Reserved. 73

BỘ XỬ LÝ

* **Lý thuyết: Đọc sách tham khảo**
  + Mục: 4.4
  + 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

Copyrights 2017 CE-UIT. All Rights Reserved. 74