Khoa
CÔNG NGHỆ THÔNG TIN
ĐH Khoa học Tự nhiên TPHCM

Bài 07: Thiết kế bộ xử lý

Phạm Tuấn Sơn

ptson@fit.hcmus.edu.vn

Quan điểm về cấu tạo CPU

- William Stallings
  - Registers
  - ALU
  - CU
  - Internal bus

  Mục tiêu: hiểu được cấu tạo và hoạt động của CPU
- Patterson & Hennessy
  - Datapath
  - Control

  Mục tiêu: thiết kế CPU

Các bước thiết kế một CPU

1. Phân tích kiến trúc bộ lệnh (ISA)
   ⇒ các yêu cầu về datapath
   - Trình bày từng lệnh dưới dạng register transfers language (RTL) để thấy rõ ý nghĩa các các lệnh
   - datapath phải có thành phần lưu trữ (bộ nhớ chính / cache) cho các thanh ghi trong kiến trúc bộ lệnh
   - datapath phải hỗ trợ thực thi tất cả các lệnh
2. Lựa chọn các khối mạch cần thiết để xây dựng datapath
   - Khối mạch tổ hợp
   - Khối mạch tuần tự
3. Lắp ráp các khối mạch đáp ứng yêu cầu bộ lệnh
4. Phân tích mỗi lệnh để xác định các tín hiệu điều khiển cần thiết
5. Thiết kế mạch cho các tín hiệu điều khiển

Vấn đề thiết kế datapath

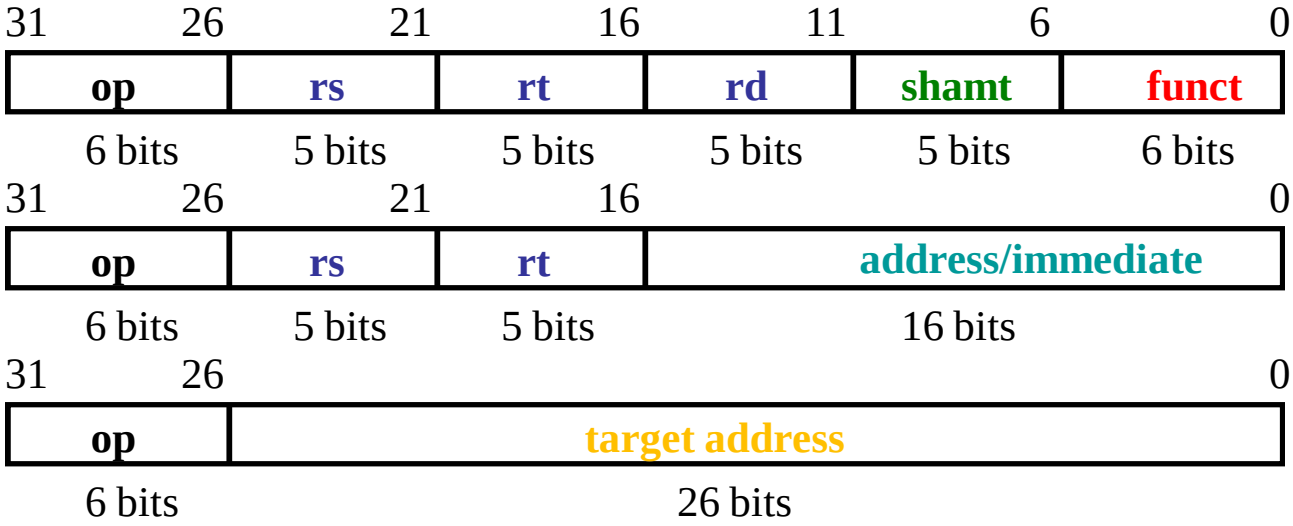
- Vấn đề: xây dựng một khối datapath phức tạp để xử lý một lệnh (nạp lệnh **à** thực thi lệnh **à** …) sẽ khó khăn và không hiệu quả
- Giải pháp: chia nhỏ quá trình xử lý một lệnh thành các công đoạn nhỏ (stages), xây dựng khối xử lý cho từng công đoạn rồi lắp ráp thành datapath
  - Các công đoạn nhỏ dễ thiết kế
  - Dễ thay đổi, tối ưu một công đoạn mà ít ảnh hưởng tới các công đoạn khác

Thiết kế bộ xử lý MIPS thu gọn

- Bộ xử lý MIPS thu gọn gồm 9 lệnh
  - `add  $1, $2, $3`
  - `sub  $1, $2, $3`
  - `and  $1, $2, $3`
  - `or  $1, $2, $3`
  - `lw  $1, 0($2)`
  - `sw  $1, 0($2)`
  - `beq  $1, $2, NHAN`
  - `slt  $1, $2, $3`
  - `j  NHAN`
- Tại sao là 9 lệnh này ?

Nhắc lại: Các cấu trúc lệnh của MIPS

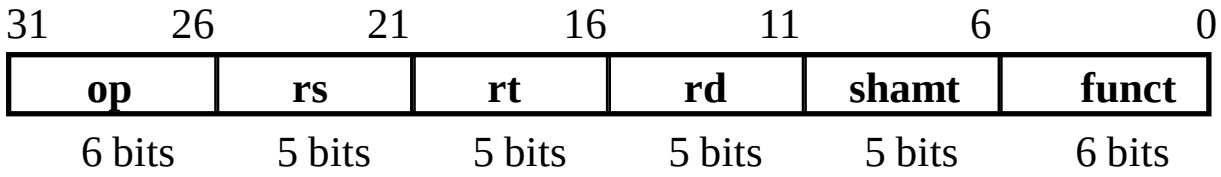
- Tất cả các lệnh MIPS đều dài 32 bit. Có 3 cấu trúc
  - R-type
  - I-type
  - J-type



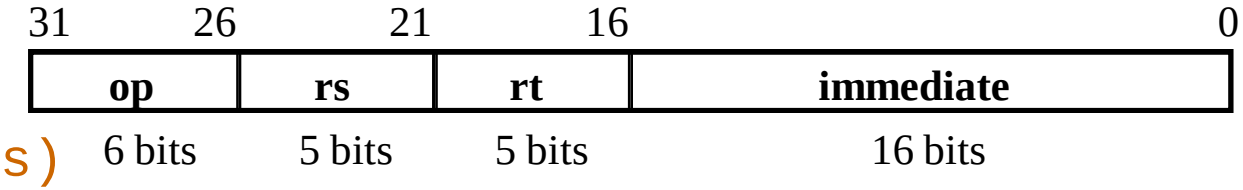
- Các trường
  - op ("opcode"): mã thao tác của lệnh, xác định lệnh làm gì
  - funct: kết hợp với op (nếu cần) để xác định lệnh làm gì
  - rs, rt, rd: địa chỉ các thanh ghi nguồn và đích
  - shamt: số bit dịch
  - address / immediate: địa chỉ hoặc hằng số tính toán
  - target address: địa chỉ cần nhảy tới

Cấu trúc các lệnh trong CPU MIPS thu gọn

- add, sub, and, or, slt
  - `add rd,rs,rt`
  - `sub rd,rs,rt`
  - `and rd,rs,rt`
  - `or  rd,rs,rt`
  - `slt rd,rs,rt`



- lw, sw, beq
  - `lw rt,imm16(rs)`
  - `sw rt,imm16(rs)`
  - `beq rs,rt,imm16`



- j
  - `j addr26`

Các công đoạn lệnh (1/3)

- Công đoạn 1: Nạp lệnh (Instruction Fetch)
  - Nạp lệnh 32 bit từ bộ nhớ tại địa chỉ trong thanh ghi PC vào thanh ghi lệnh. Công đoạn này như nhau cho tất cả các lệnh
  - Sau đó, tăng PC để chuẩn bị nạp lệnh kế tiếp sau khi xử lý xong lệnh này (PC = PC + 4)
- Công đoạn 2: Giải mã lệnh (Instruction Decode)
  - Phân tích các trường trong lệnh
    - Xác định opcode để biết loại lệnh và vị trí của các trường khác
    - Sau đó, đọc các thanh ghi nguồn để chuẩn bị thực hiện công đoạn tiếp theo
      Ví dụ
      - Lệnh add, đọc 2 thanh ghi nguồn
      - Lệnh lw, đọc 1 thanh ghi nguồn

Các công đoạn lệnh (2/3)

- Công đoạn 3: Tính toán (ALU – Arithmetic-Logic Unit)
  - Công việc chính của hầu hết các lệnh thực hiện tại công đoạn này: tính toán số học (+, -), luận lý (&, |), so sánh (`beq`, `slt`)
  - Lệnh `beq` tính vị trí cần nhảy tới
  - Còn lệnh `lw` và `sw` làm gì trong công đoạn này ?
    - `lw  $t0, 40($t1)`
    - Địa chỉ của vùng nhớ cần truy xuất = giá trị của `$t1` CỘNG 40
    - Do đó, thực hiện phép cộng trong công đoạn này

Các công đoạn lệnh (3/3)

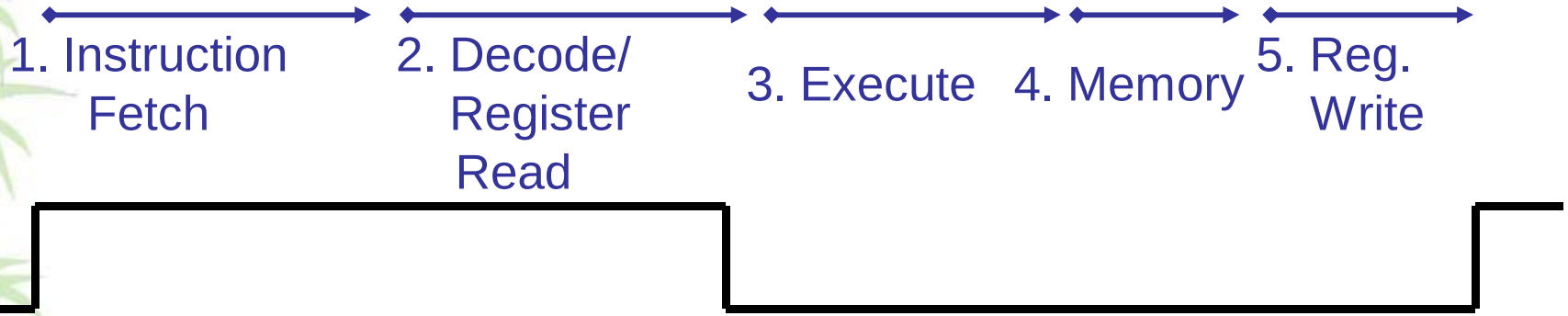
- Công đoạn 4: Truy xuất bộ nhớ (Memory Access)
  - Thực sự chỉ có lệnh `lw` và `sw` thực hiện công đoạn lệnh này
  - Do công việc truy xuất bộ nhớ mất thời gian tương đối nhiều nên cần một công đoạn riêng
- Công đoạn 5: Ghi kết quả vào thanh ghi (Register Write)
  - Hầu hết các lệnh đều ghi kết quả tính toán vào một thanh ghi như tính toán số học, luận lý, `lw`, `slt`
  - Còn các lệnh `sw`, lệnh nhảy ?
    - Không ghi kết quả gì vào thanh ghi
    - Do đó, các lệnh này không làm gì tại công đoạn lệnh này

Tại sao lại 5 công đoạn ?

- Chỉ có lệnh lw thực hiện cả 5 công đoạn. Vậy tại sao MIPS lại chia làm 5 công đoạn ?
  - Đó là sự tổ hợp đầy đủ cho tất cả các thao tác cần thiết của tất cả các lệnh
  - Thời gian thực hiện mỗi công đoạn không quá chênh lệch nhau
- Có thể có nhiều công đoạn lệnh hơn không ?
  - Có, các kiến trúc khác như x86

Kỹ thuật thiết kế CPU 1 chu kỳ

- Thiết kế CPU 1 chu kỳ: Tất cả các công đoạn của 1 lệnh được xử lý trong 1 chu kỳ đồng hồ
  - Chu kỳ đồng hồ phải đủ lâu để có thể hoàn thành xử lý mọi lệnh



Bước 1: Biểu diễn các lệnh dưới dạng RTL

Lệnh	Register Transfers	
Nạp	{op , rs , rt , rd , shamt , funct} ← MEM[PC]	
lệnh	{op , rs , rt , Imm16} ← MEM[PC]	
add	R[rd] ←R[rs] + R[rt];	PC ← PC + 4
sub	R[rd] ←R[rs] – R[rt];	PC ← PC + 4
and	R[rd] ←R[rs] & R[rt];	PC ← PC + 4
or	R[rd] ←R[rs] | R[rt];	PC ← PC + 4
lw	R[rt] ← MEM[R[rs] + sign_ext(imm16)];	PC ← PC + 4
sw	MEM[R[rs] + sign_ext(imm16)] ←R[rt];	PC ← PC + 4
beq	if (R[rs] == R[rt]) then PC ← PC + 4 + (sign_ext(imm16) << 2) else PC ← PC + 4	
slt	if (R[rs] < R[rt]) then R[rd] ← 1; PC ← PC + 4	
j	PC = {PC[31:28],Addr26 << 2}	

Bước 1: Các khối mạch cần thiết

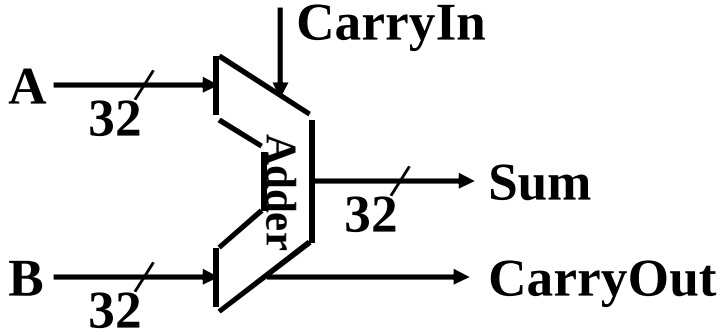
- Bộ nhớ (MEM)
  - Lệnh + Dữ liệu
- Thanh ghi (32 x 32)
  - Đọc RS
  - Đọc RT
  - Ghi RT / RD
- Thanh ghi PC (Program Counter)
- Sign Extender
- Đơn vị thực hiện các phép tính `add/sub/and/or` trên các thanh ghi hoặc hằng số
- Đơn vị thực hiện (PC + 4)
- So sánh thanh ghi ? (lệnh `beq,slt`)

Bước 2: Các thành phần của datapath

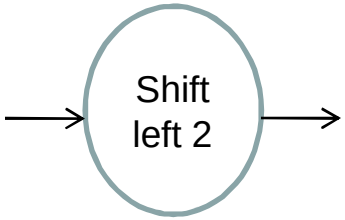
- Các khối mạch tổ hợp
- Các khối lưu trữ

Các khối mạch tổ hợp

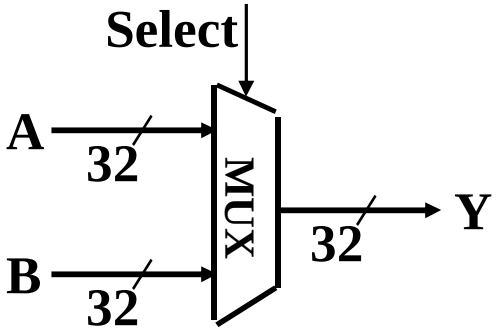
- Adder



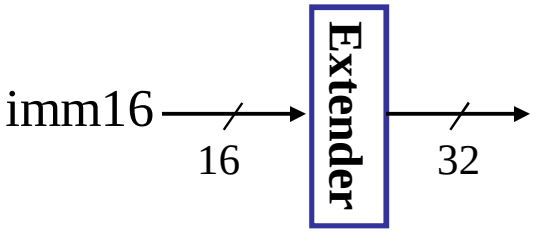
Shift left 2



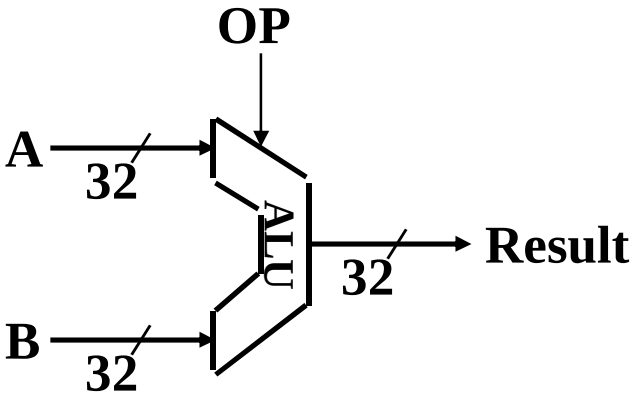
- MUX



Extender

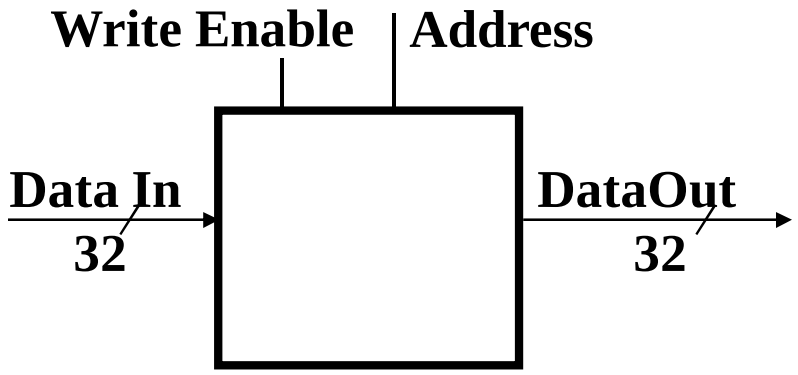


- ALU



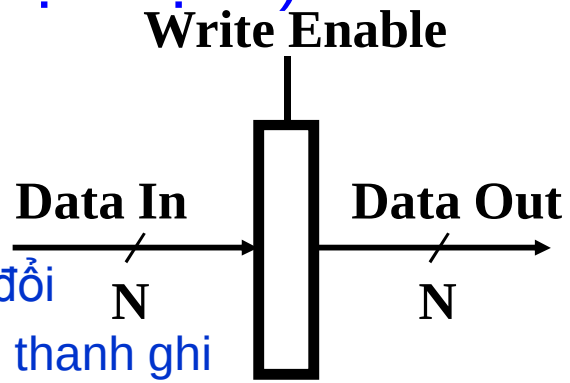
Khối lưu trữ: Bộ nhớ

- Bộ nhớ
  - Một đường dữ liệu vào
    - Data In
  - Một đường dữ liệu ra
    - Data Out
  - Đường địa chỉ (address) để xác định từ nhớ nào được truy xuất
  - Tính hiệu Write Enable = 1: xác định dữ liệu có được ghi vào bộ nhớ qua đường vào dữ liệu hay không

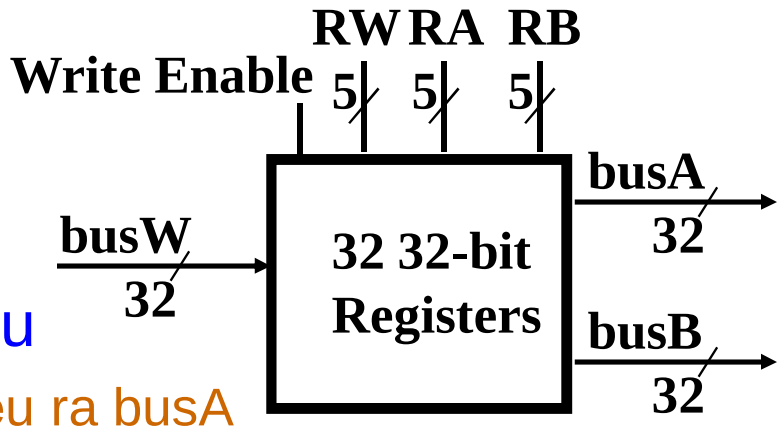


Khối lưu trữ: Thanh ghi

- Xây dựng dựa trên các mạch lật (như mạch lật D)
  - N bit đầu vào (Data In)
  - N bit đầu ra (Data Out)
  - Tín hiệu Write Enable
    - Giá trị 0: dữ liệu trong thanh ghi không thay đổi
    - Giá trị 1: cho phép ghi dữ liệu từ Data In vào thanh ghi



- Tập thanh ghi gồm 32 thanh ghi
  - 2 đường truyền dữ liệu ra 32 bit busA và busB
  - Một đường truyền dữ liệu vào busW
- Thanh ghi được lựa chọn như sau
  - RA lựa chọn thanh ghi để đưa dữ liệu ra busA
  - RB lựa chọn thanh ghi để đưa dữ liệu ra busB RB
  - RW lựa chọn thanh ghi để ghi dữ liệu từ busW vào khi Write Enable = 1

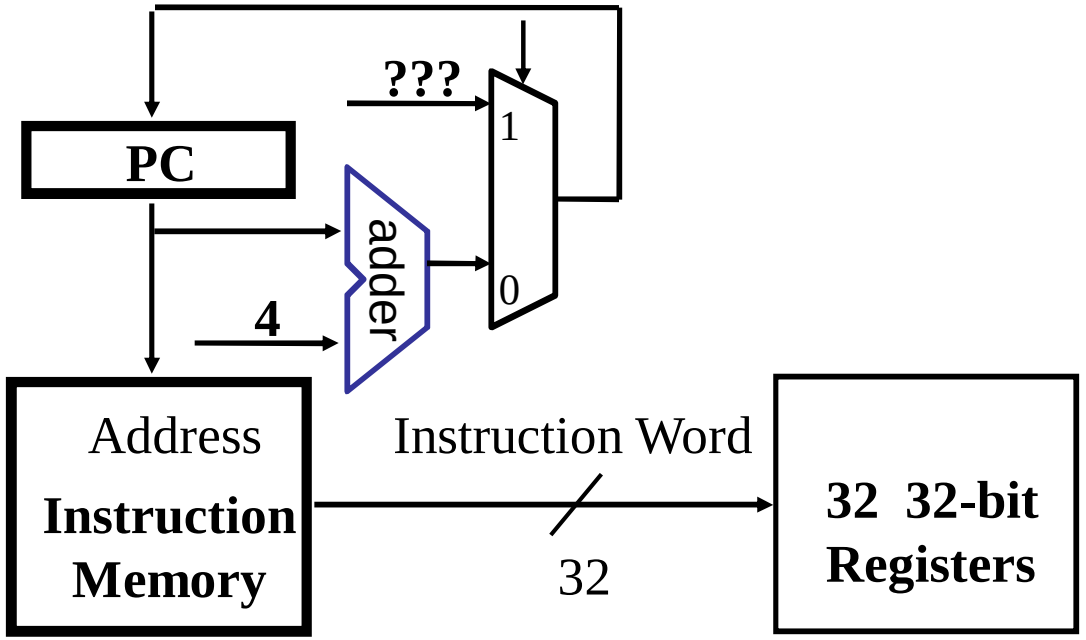


Bước 3 : Lắp ráp các khối mạch thành datapath

a. Công đoạn 1: Nạp lệnh
b. Công đoạn 2: Giải mã lệnh
c. Công đoạn 3: Thực thi lệnh
d. Công đoạn 4: Truy xuất bộ nhớ
e. Công đoạn 5: Ghi kết quả vào thanh ghi

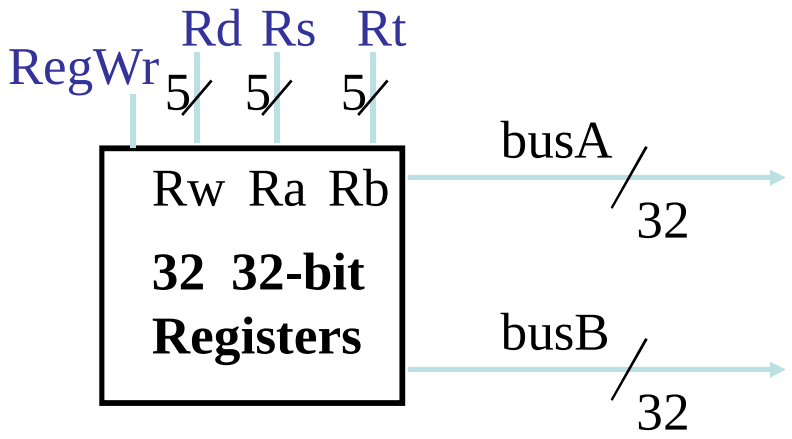
3a: Khối nạp lệnh

- Tất cả các lệnh đều thực hiện như nhau
  - Nạp lệnh
    - IR ← mem[PC]
  - Cập nhật thanh ghi PC
    - PC ← PC + 4
    - Lệnh nhảy: PC ← "???"



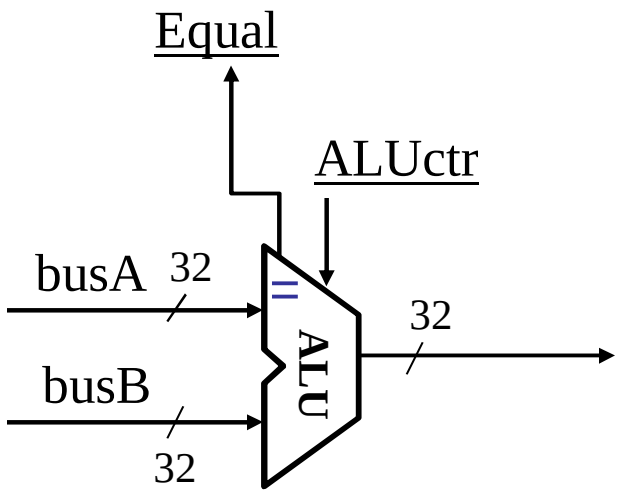
3b: Khối giải mã lệnh

- Tất cả các lệnh đều thực hiện như nhau
  - Giải mã lệnh sẽ xác định được các giá trị Ra, Rb, Rw (tương ứng với các trường Rs, Rt, Rd), và các tín hiệu điều khiển RegWr,…



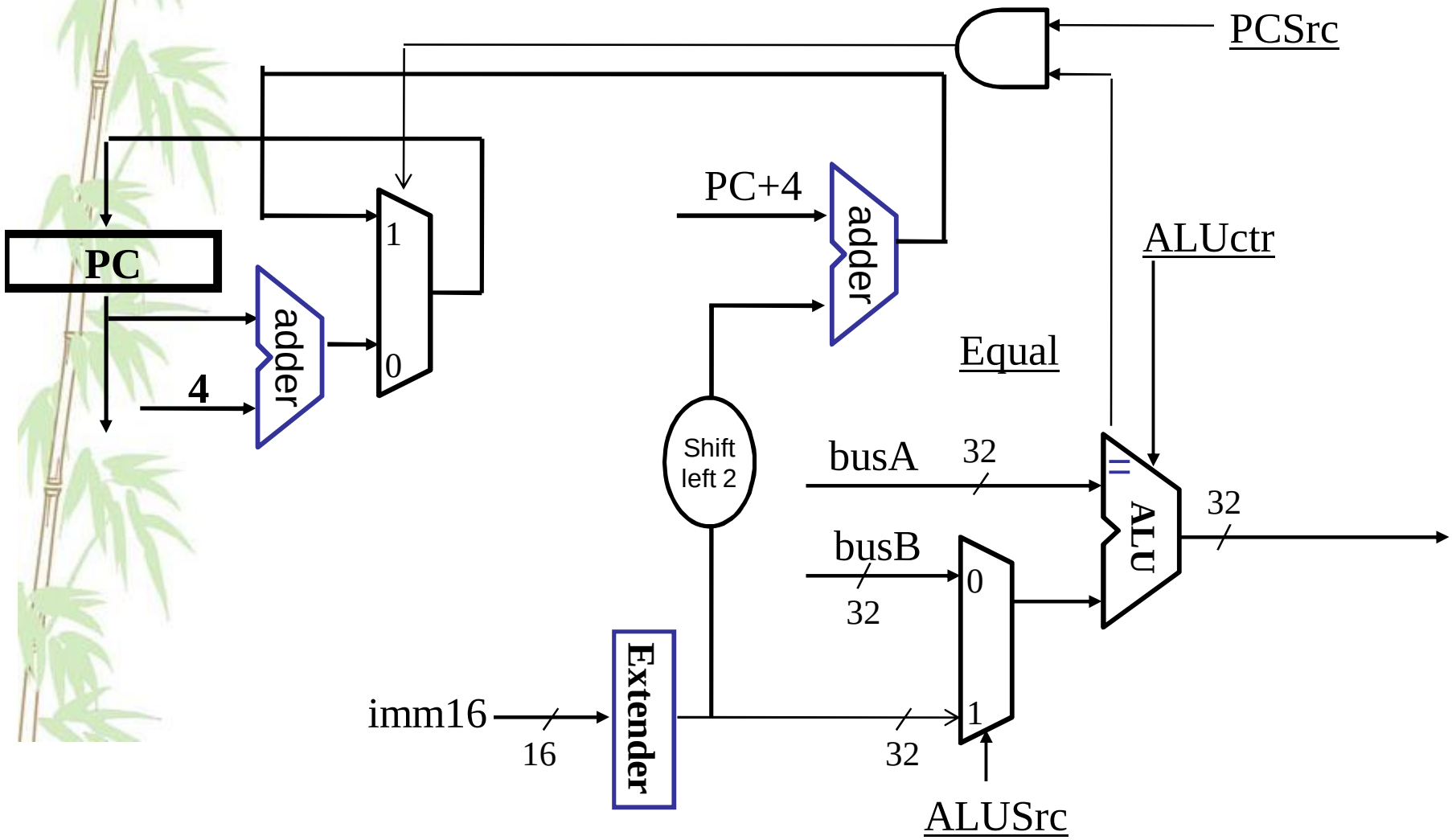
3c: Khối thực thi lệnh (1/2)

- Các lệnh R-Format
  `add,sub,and,or,beq,slt`
  - R[rs] op R[rt]



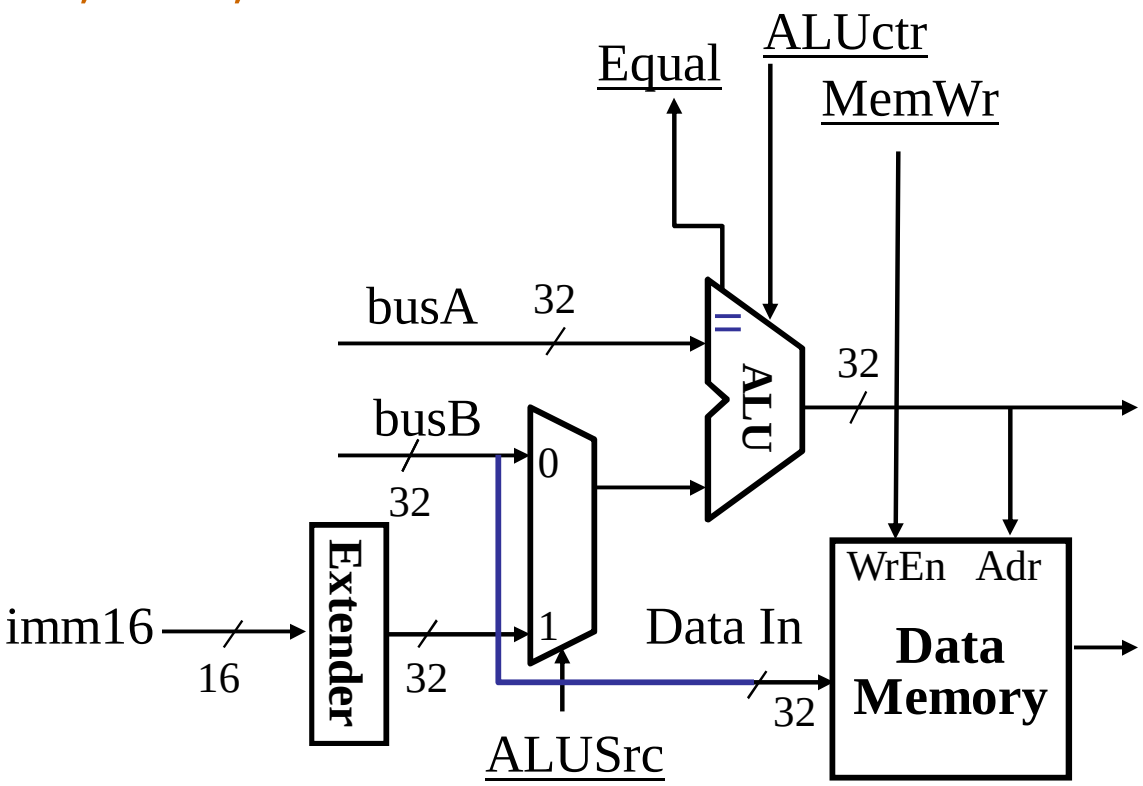
3c: Khối thực thi lệnh (2/2)

- Lệnh `lw, sw`
  - Mem[R[rs] + SignExt[imm16]]
- Lệnh `beq`
  - PC + 4 + (SignExt[imm16]] << 2)



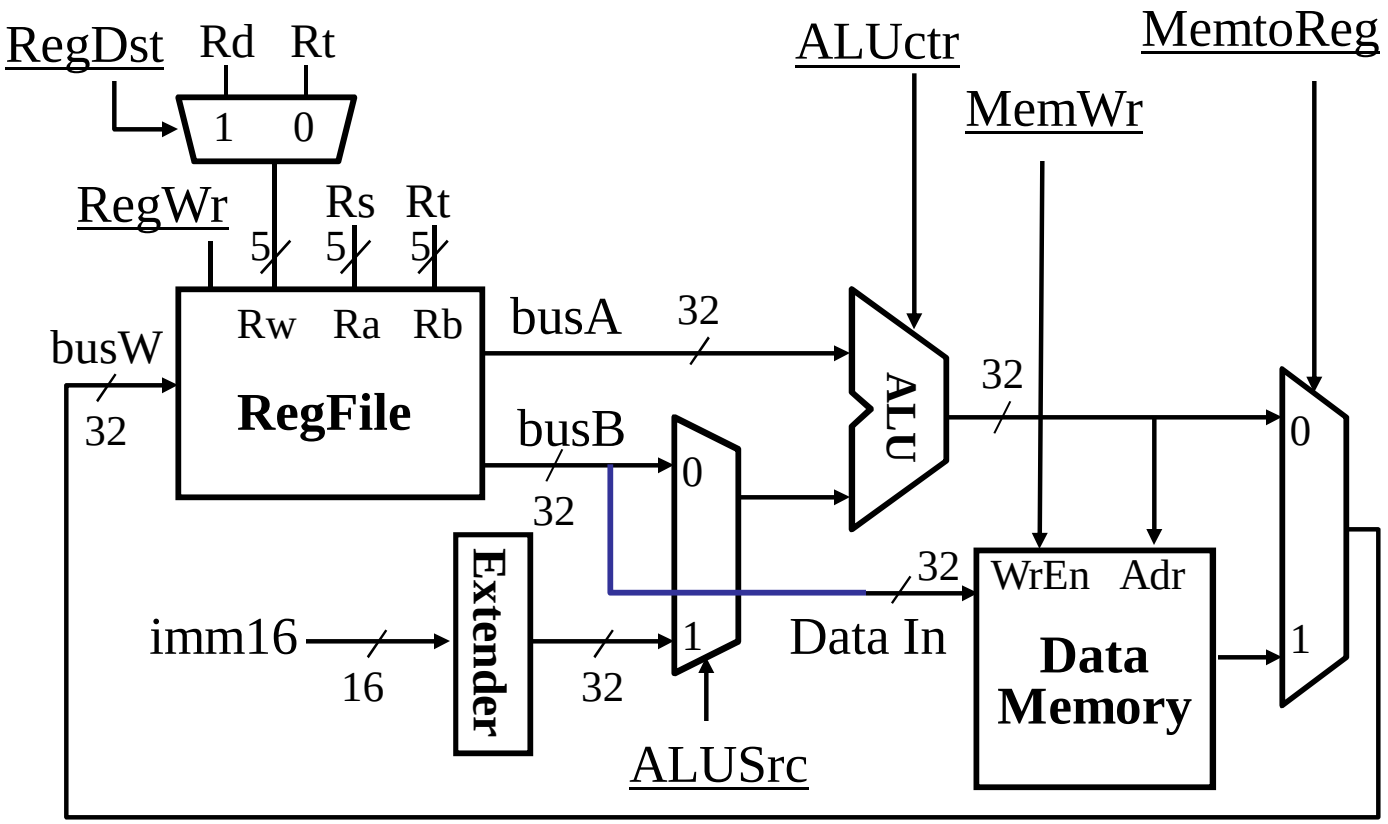
3d: Truy xuất bộ nhớ

- Lệnh `lw, sw`
  - Mem[R[rs] + SignExt[imm16]]
    Ví dụ: `lw rt,rs,imm16`
  - Mem[R[rs] + SignExt[imm16]] = R[rt]
    Ví dụ: `sw rt, rs, imm16`

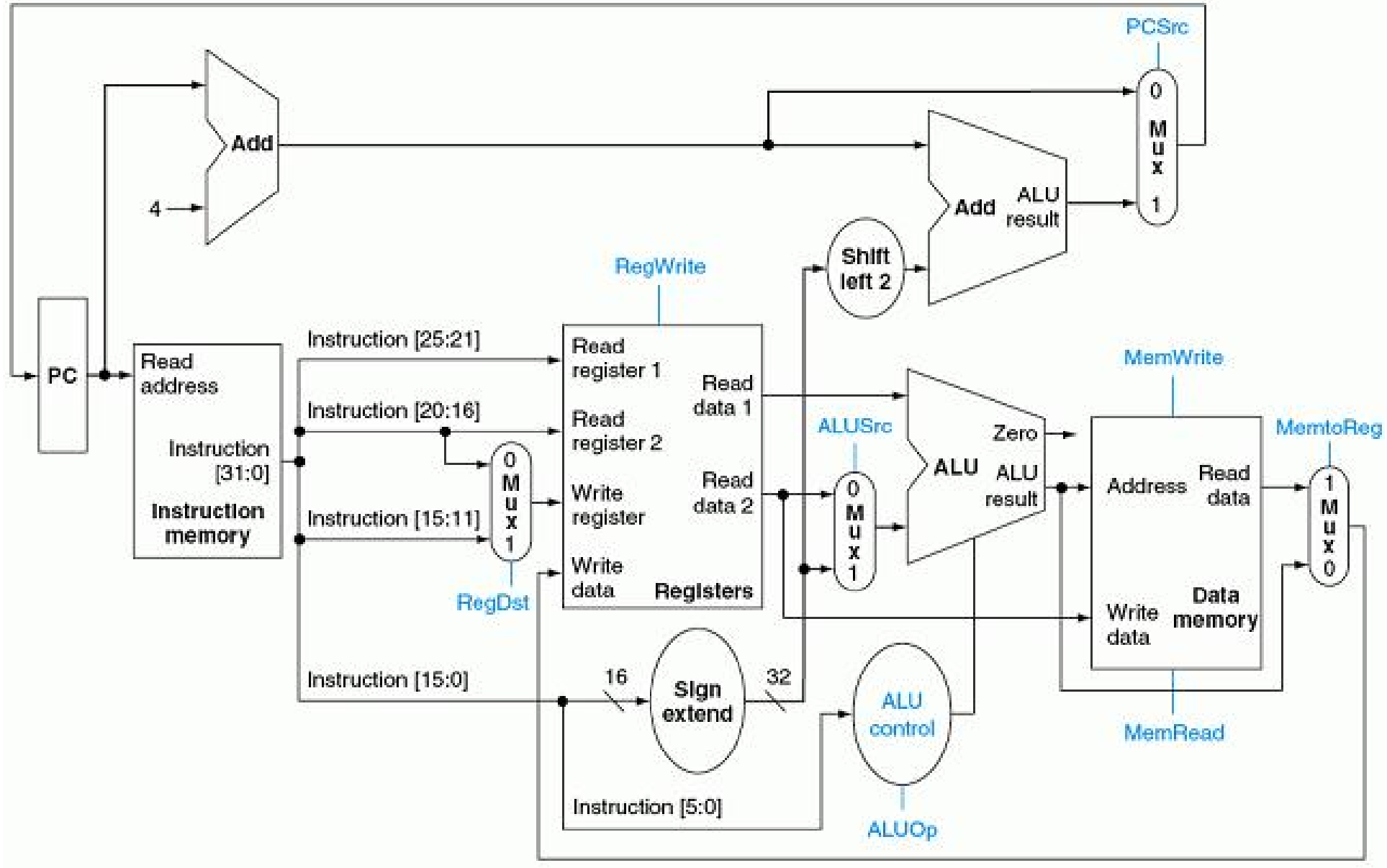


3f: Ghi kết quả vào thanh ghi

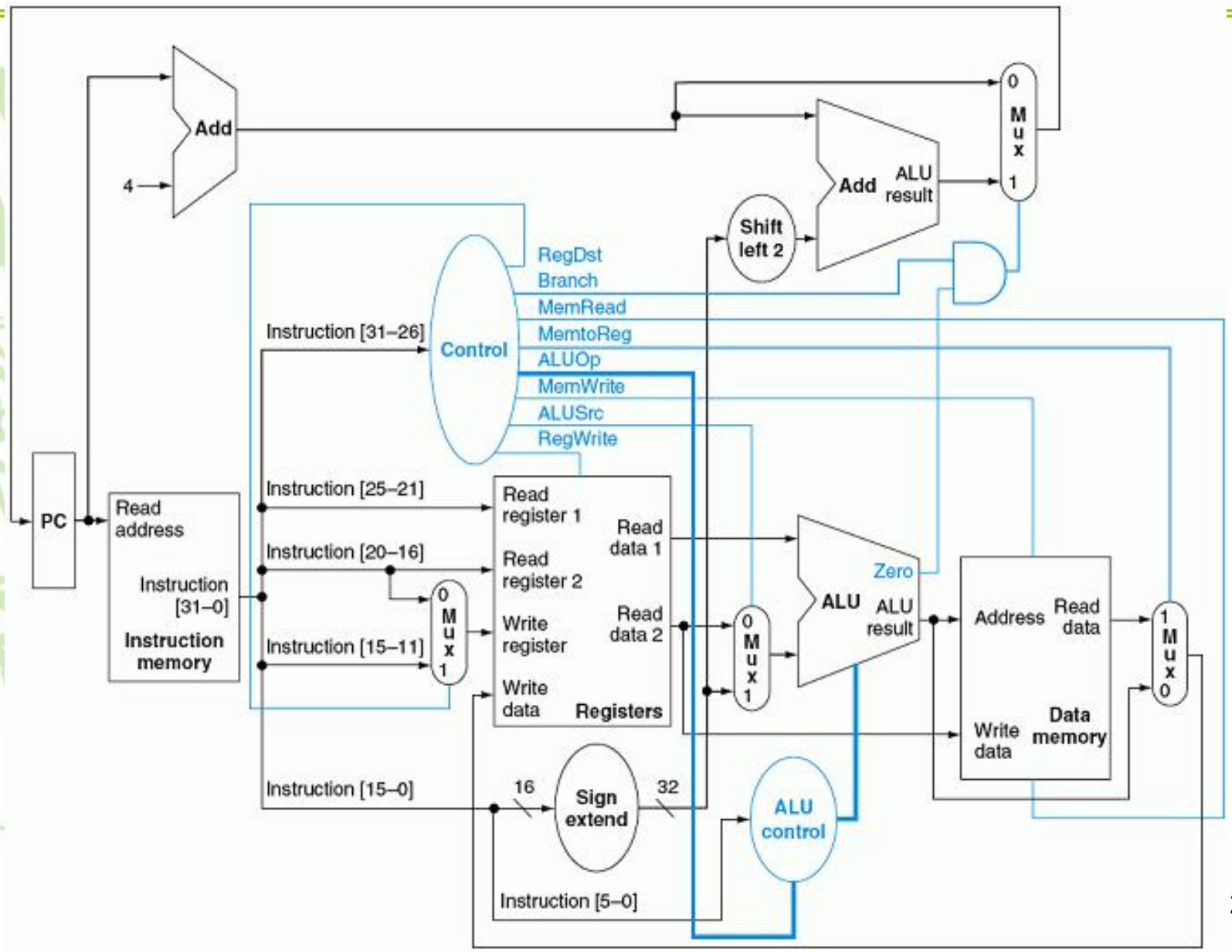
- Các lệnh `add,sub,add,or,lw,slt`
  - R[rd] = R[rs] op R[rt]
  - R[rt] = Mem[R[rs] + SignExt[imm16]]



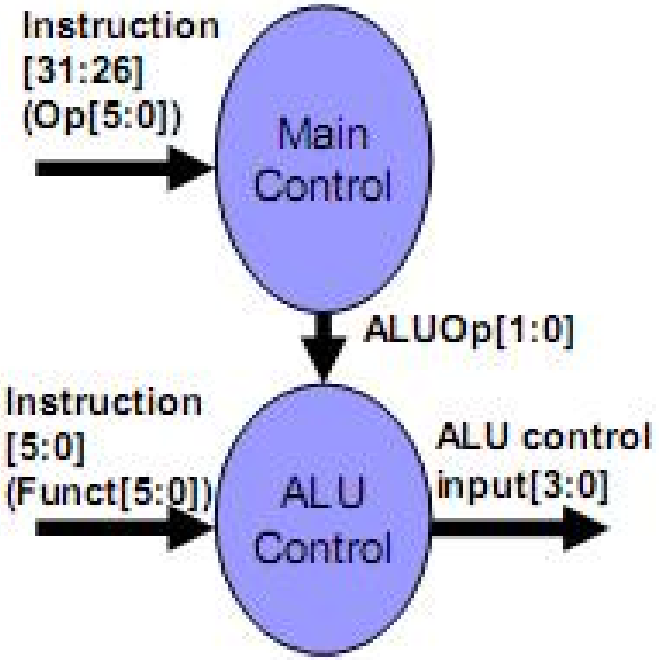
Sơ đồ datapath tổng quát



Bước 4+5: Thiết kế đơn vị điều khiển



Thiết kế đơn vị điều khiển chính (1/2)

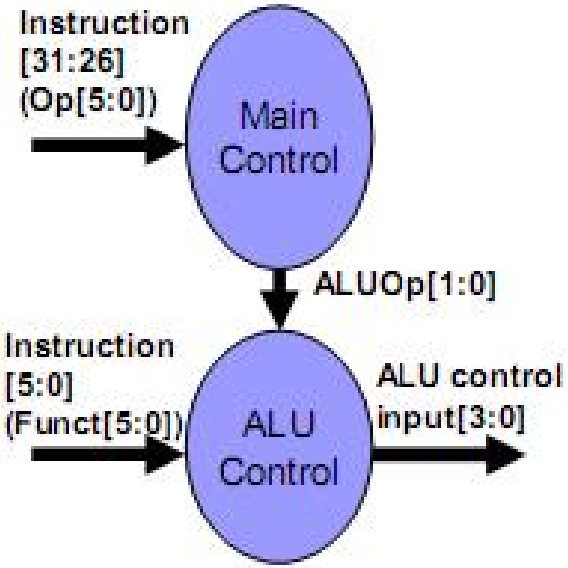


Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Thiết kế đơn vị điều khiển chính (2/2)

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

Thiết kế đơn vị điều khiển ALU (1/2)

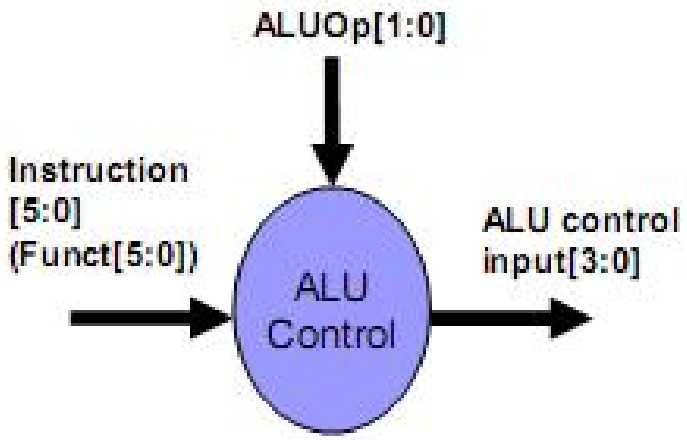


ALU control input	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

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 eqqual	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

Thiết kế đơn vị điều khiển ALU (2/2)

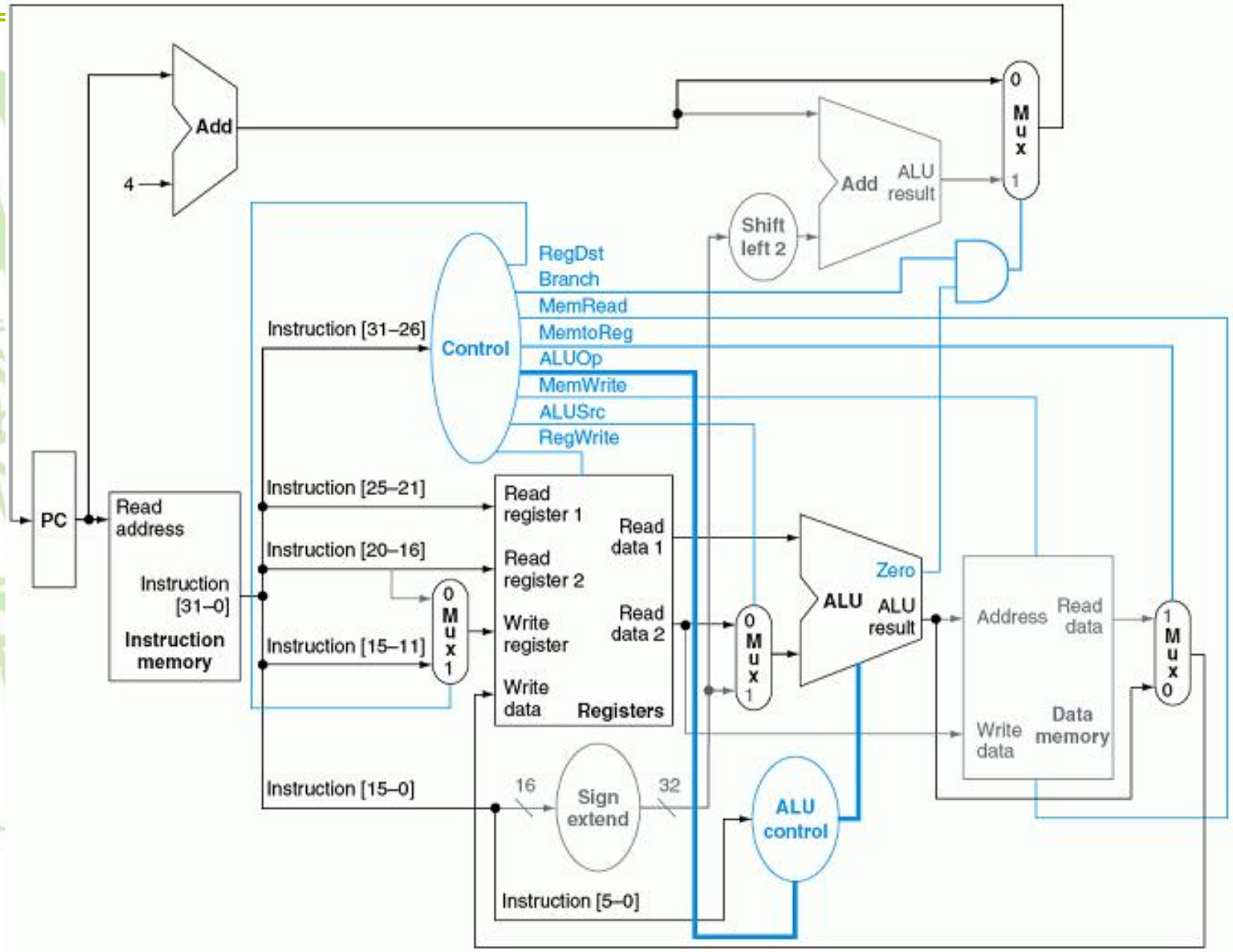
ALUOp		Funct field						ALU control input
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111



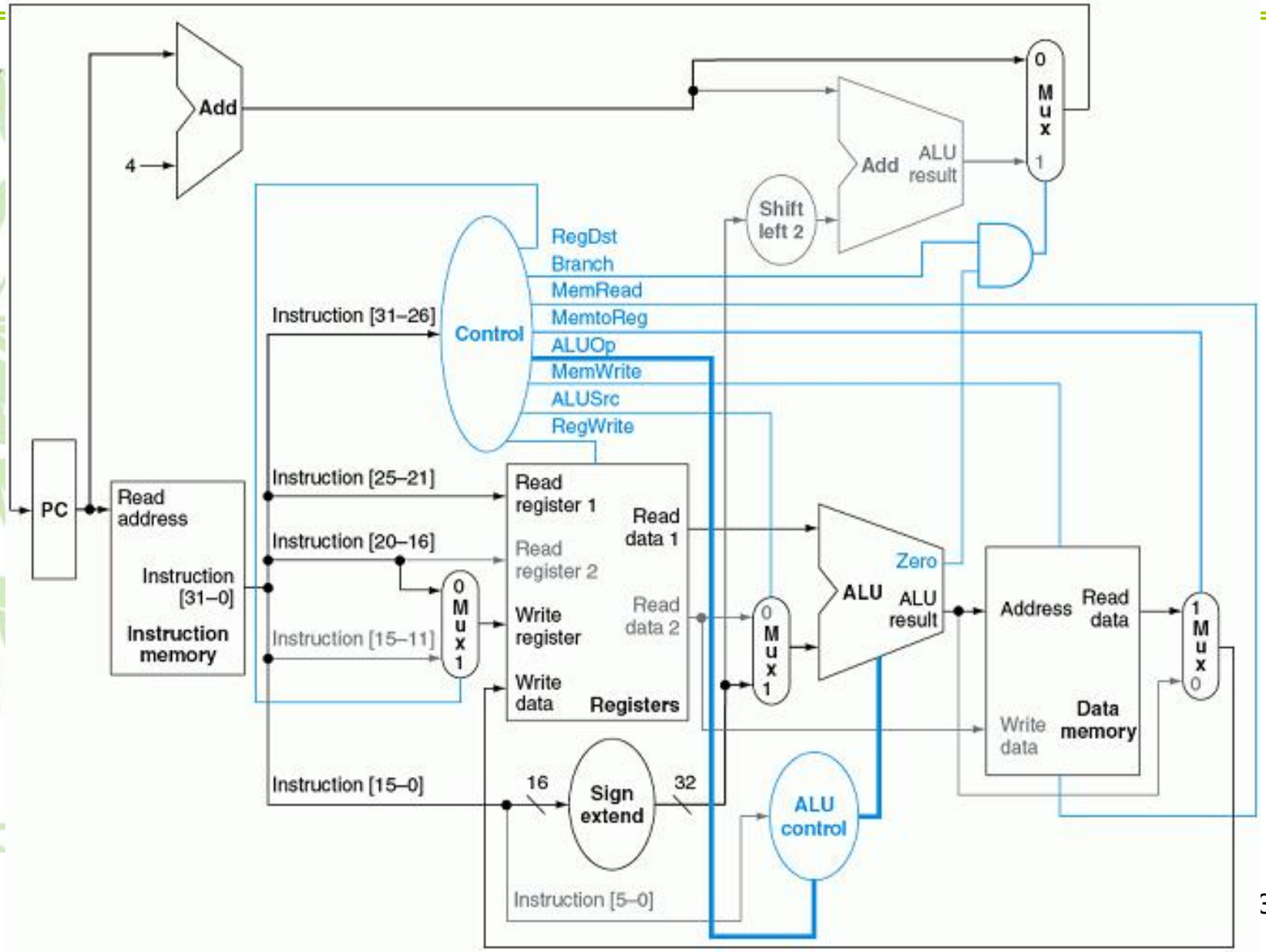
Ex.)

$$ALUOp0 + ALUOp1 \cdot \overline{F3} \cdot \overline{F2} \cdot F1 \cdot \overline{F0} + ALUOp1 \cdot F3 \cdot \overline{F2} \cdot F1 \cdot \overline{F0}$$

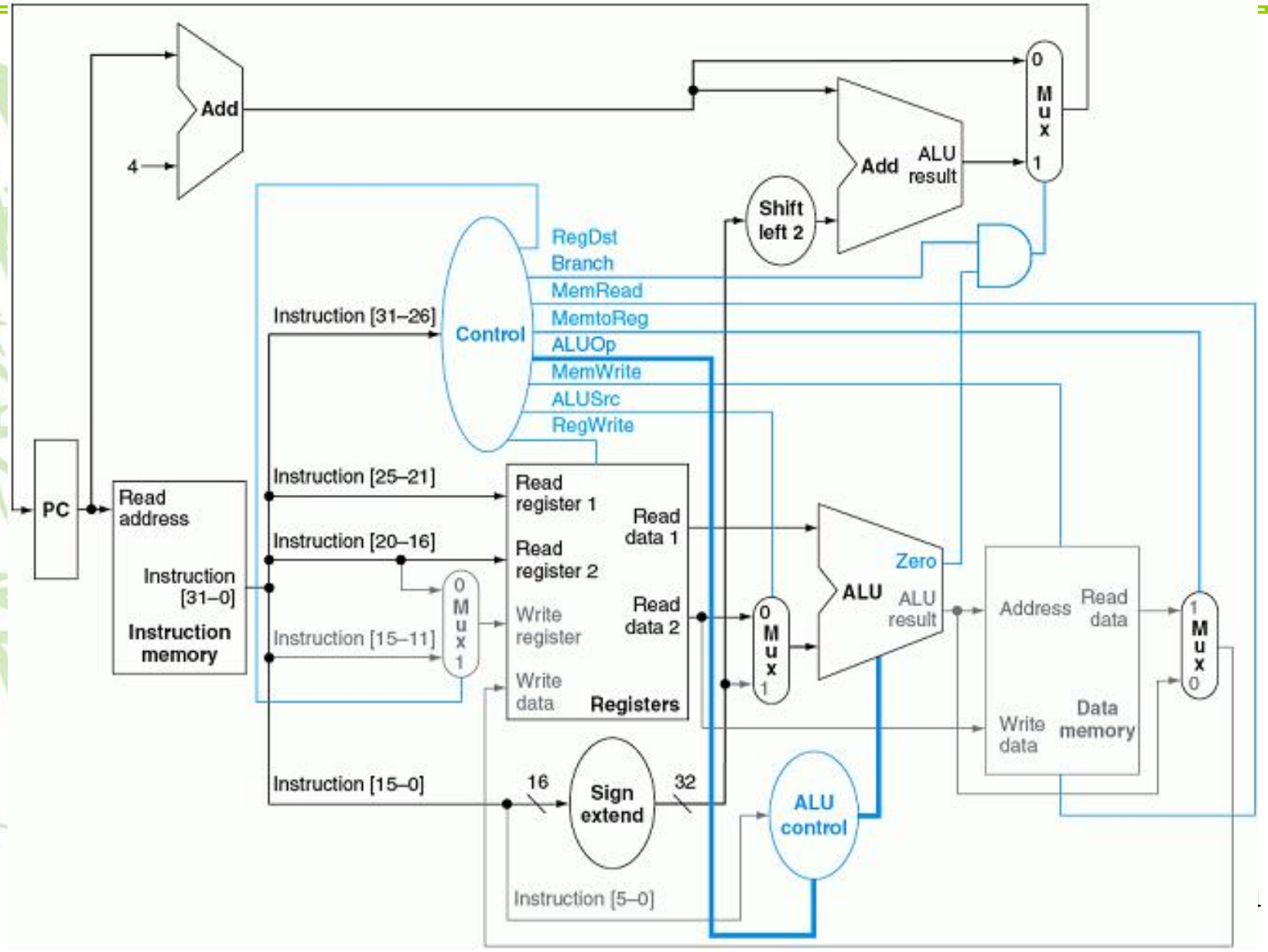
Sơ đồ xử lý lệnh R-Format



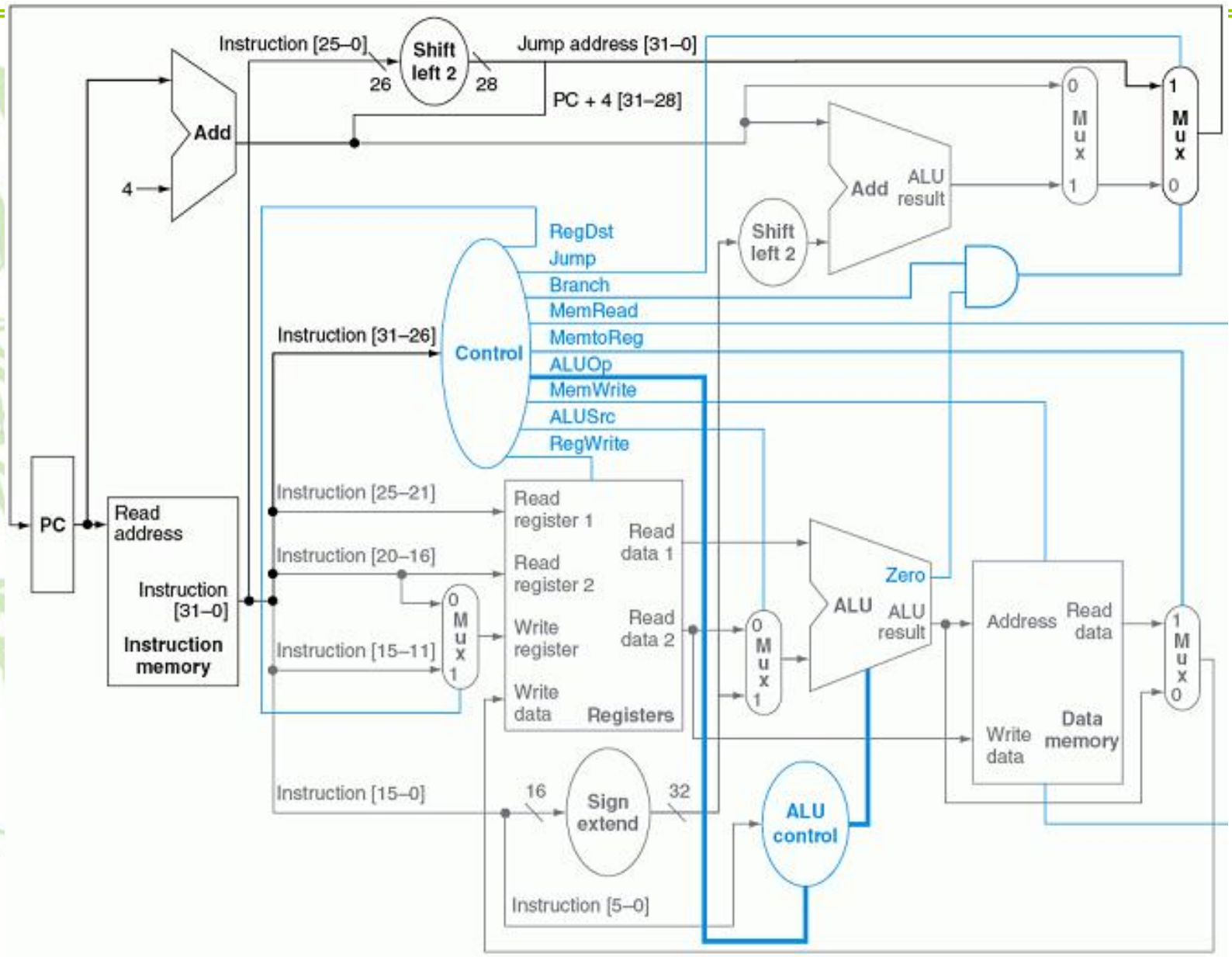
Sơ đồ xử lý lệnh lw



Sơ đồ xử lý lệnh beq



Bổ sung lệnh j



Hạn chế của kỹ thuật thiết kế CPU một chu kỳ

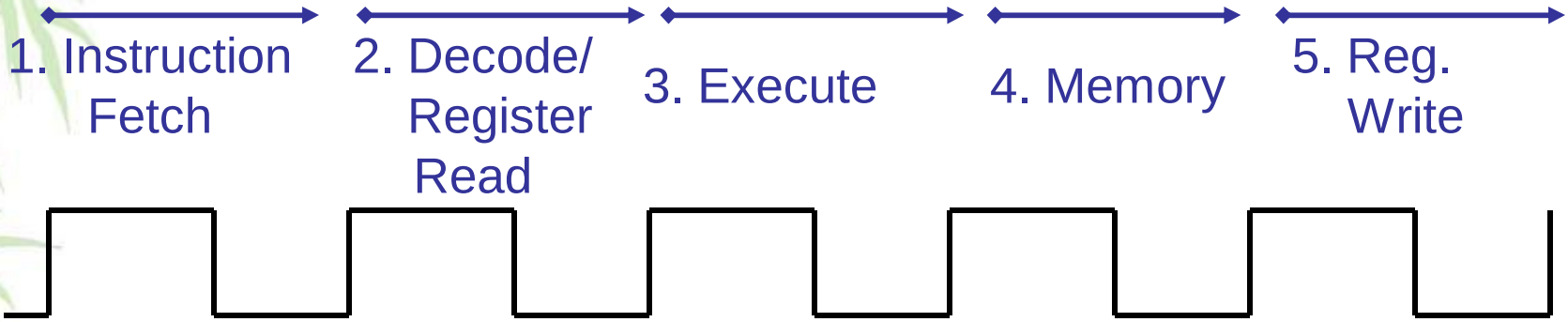
- Kỹ thuật thiết kế CPU 1 chu kỳ không còn được sử dụng vì không hiệu quả
  - Tất cả công đoạn của 1 lệnh phải xử lý trong một chu kỳ theo tín hiệu đồng bộ nên các thành phần mạch có khả năng dùng chung đều được tách riêng, làm cho sơ đồ mạch phức tạp hơn
    - Thành phần tính toán: ALU, Adder
    - Thành phần lưu trữ: Instruction memory, Data memory
  - Một chu kỳ đồng hồ phải đủ lâu để xử lý được lệnh phức tạp nhất. Trong MIPS, lệnh `lw` xử lý phức tạp nhất (5 công đoạn), trong khi tất cả các lệnh khác chỉ mất 3 (beq) hoặc 4 (R-Format, …) công đoạn

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (`sw`)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (`add, sub, and, or, slt`)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (`beq`)	200 ps	100 ps	200 ps			500 ps

  - Với chương trình có IC (instruction count) lệnh thì sẽ xử lý trong 800×IC (ps)

Kỹ thuật thiết kế CPU nhiều chu kỳ

- Thiết kế CPU nhiều chu kỳ: Mỗi công đoạn lệnh thực hiện trong 1 chu kỳ
  - Mỗi chu kỳ đồng hồ phải đủ lâu để thực hiện mọi công đoạn lệnh

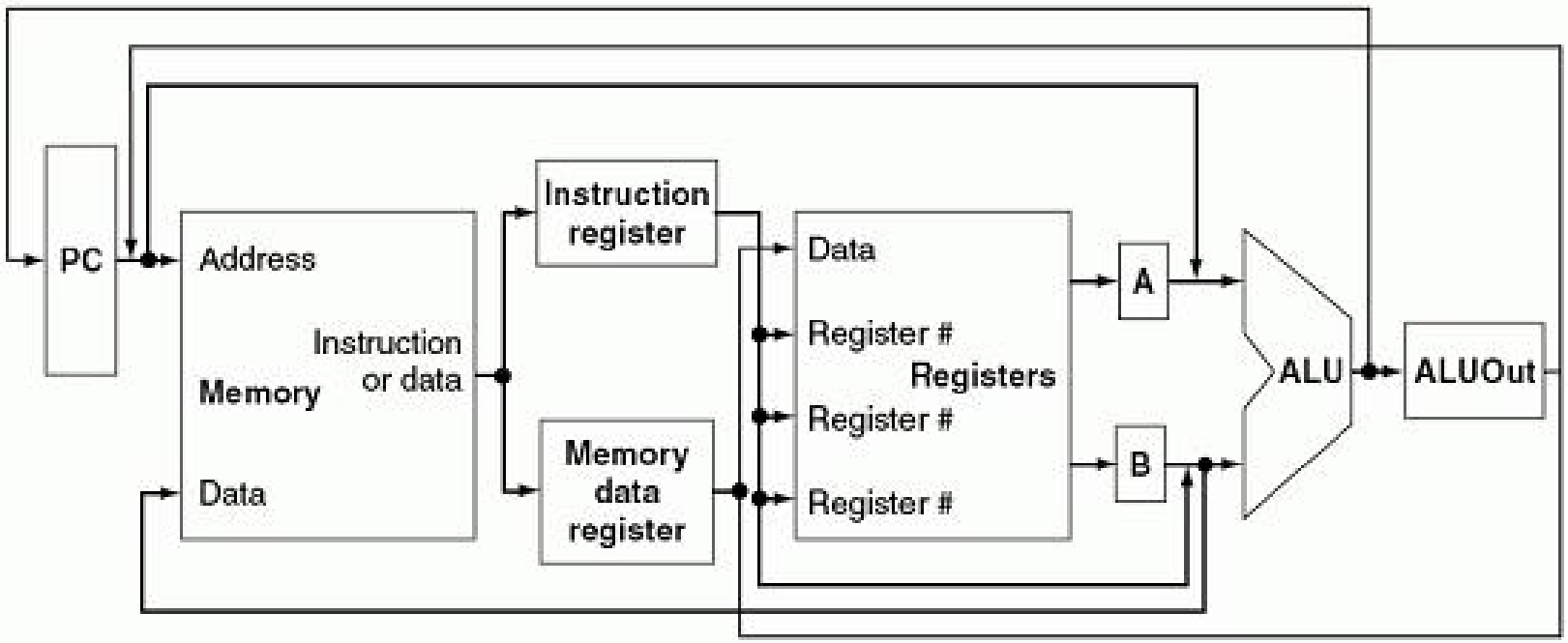


	Load	Store	ALU	Branch	Jump
Cycles	5	4	4	3	3
Frequency(%)	25	10	52	11	2

  - Với chương trình có IC (instruction count) lệnh thì sẽ xử lý trong
    (0.25×5 + 0.1×4 + 0.52×4 + 0.11×3 + 0.02×3) × 200 × IC
    = 824×IC (ps) !!!

Sơ đồ khối CPU nhiều chu kỳ

- Do mỗi công đoạn được thực thi trong một chu kỳ riêng, nên có thể ghép các thành phần mạch dùng chung (ALU + Adder, Imem + DMem) mà không xảy ra đụng độ
- Cần thêm các thanh ghi để lưu giữ kết quả trung gian của các công đoạn lệnh (A, B, ALUOut,…)



- Thiết kế chi tiết datapath và control của bộ xử lý theo kỹ thuật nhiều chu kỳ
  à môn KTMT nâng cao

Tham khảo

- Phần 5.5, P&H