COMPUTER ARCHITECTURE
CSE Fall 2017

Faculty of Computer Science and Engineering

Department of Computer Engineering

Vo Tan Phuong

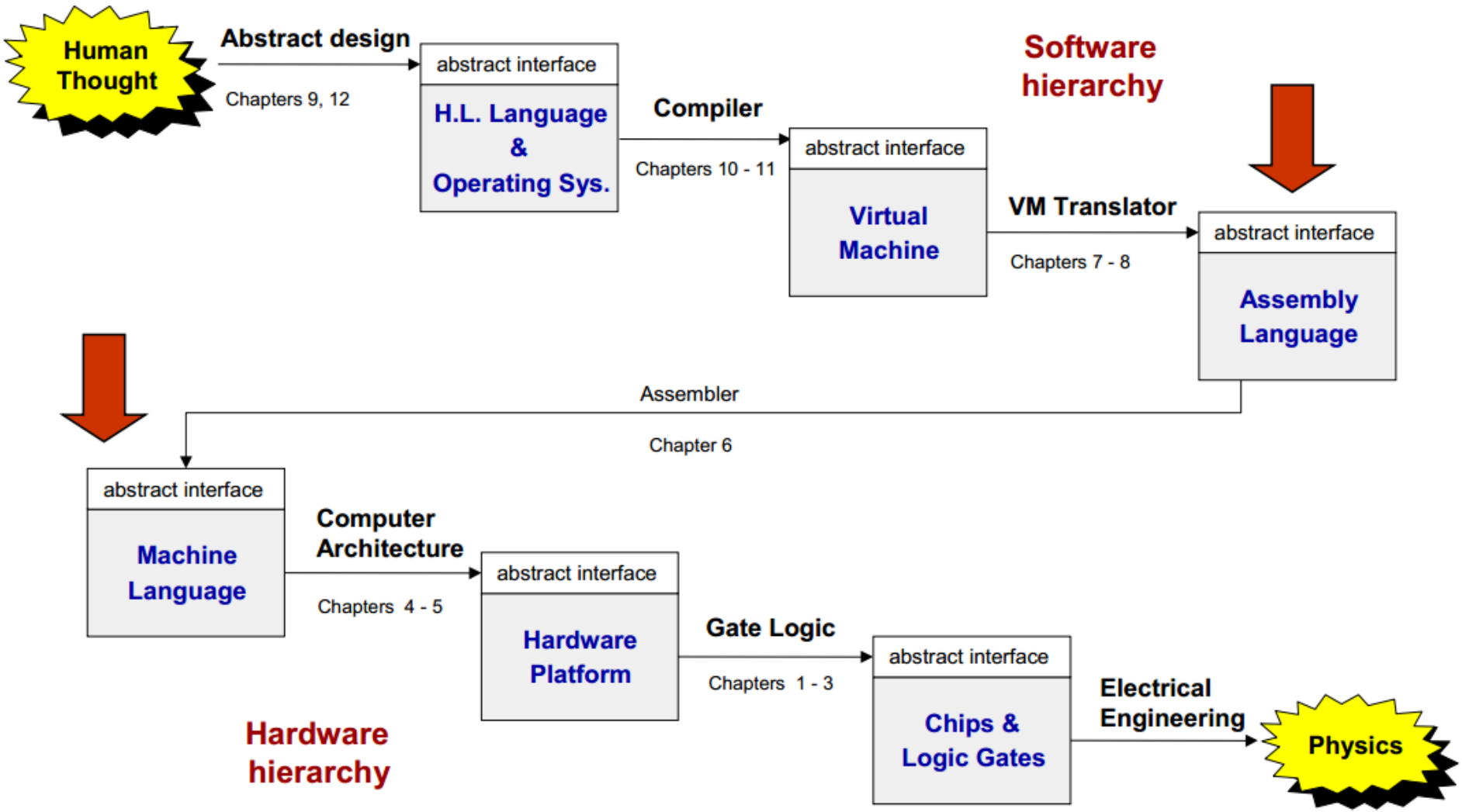
http://www.cse.hcmut.edu.vn/vtphuong

Chapter 4.1

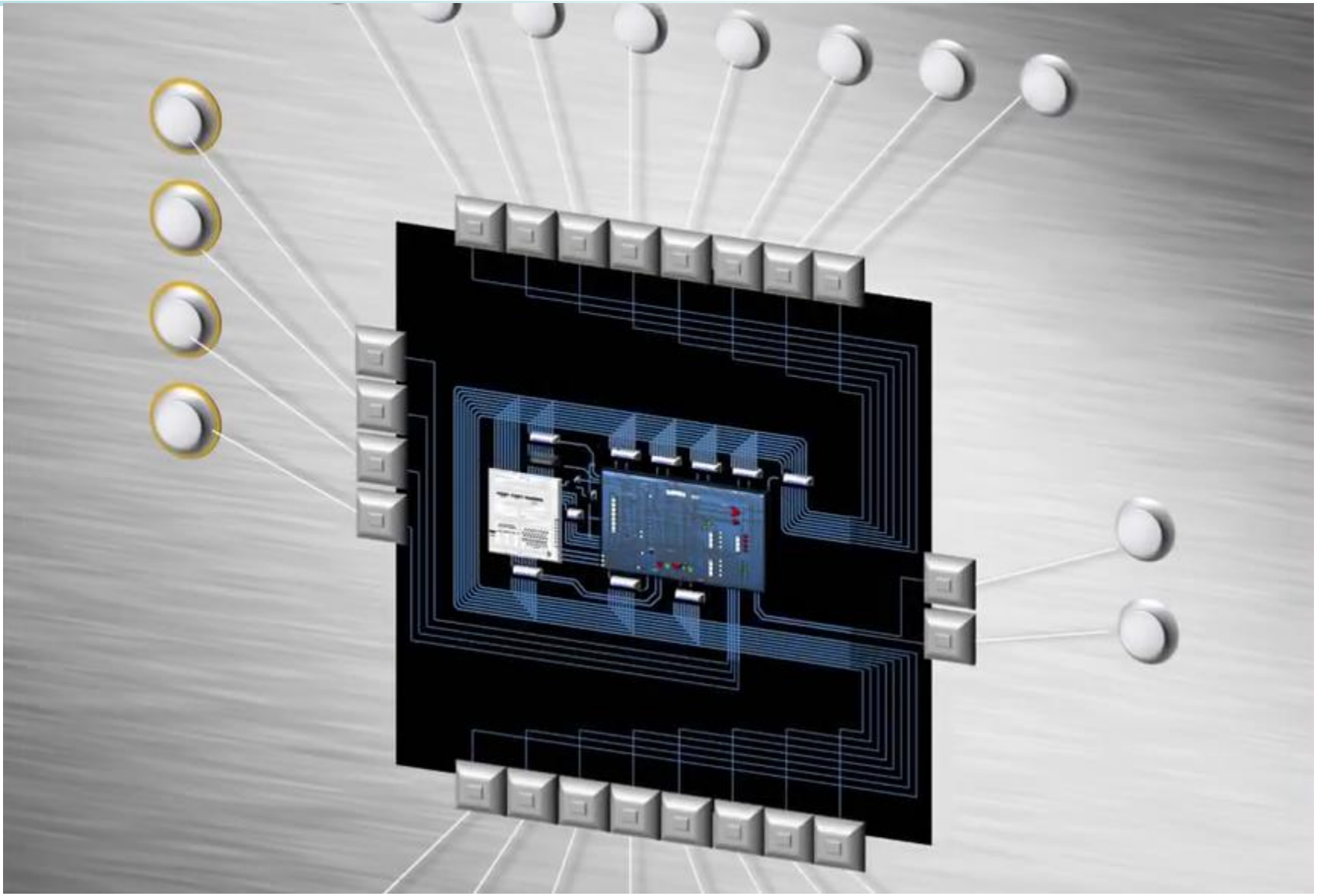
Thiết kế bộ xử lý đơn chu kỳ

(Single Cycle Processor)

Chúng ta đang ở đâu?



Bên trong bộ xử lý

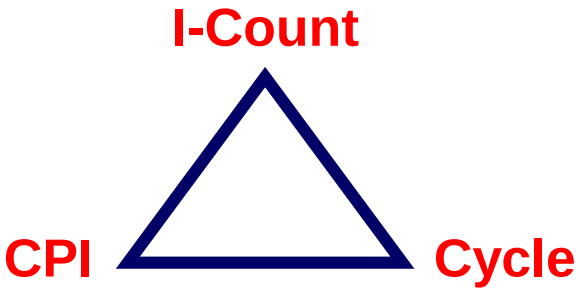


Nội dung

- Thiết kết bộ xử lý: Các bước thực hiện
- Các thành phần của Datapath và cấp xung nhịp
- Xây dựng Datapath đầy đủ
- Điều khiển quá trình thực thi của các lệnh
- Bộ điều khiển chính và bộ điều khiển ALU
- Hạn chế của thiết kế bộ xử lý đơn chu kỳ

Các yếu tố ảnh hưởng đến hiệu xuất

- Hiệu xuất được xác định bởi:
  - Số lệnh (Instruction count)
  - Số chu kỳ xung nhịp trung bình trên lệnh (CPI)
  - Thời gian của một chu kỳ xung nhịp
- Thiết kế bộ xử lý ảnh hưởng;
  - CPI
  - Thời gian của một chu kỳ



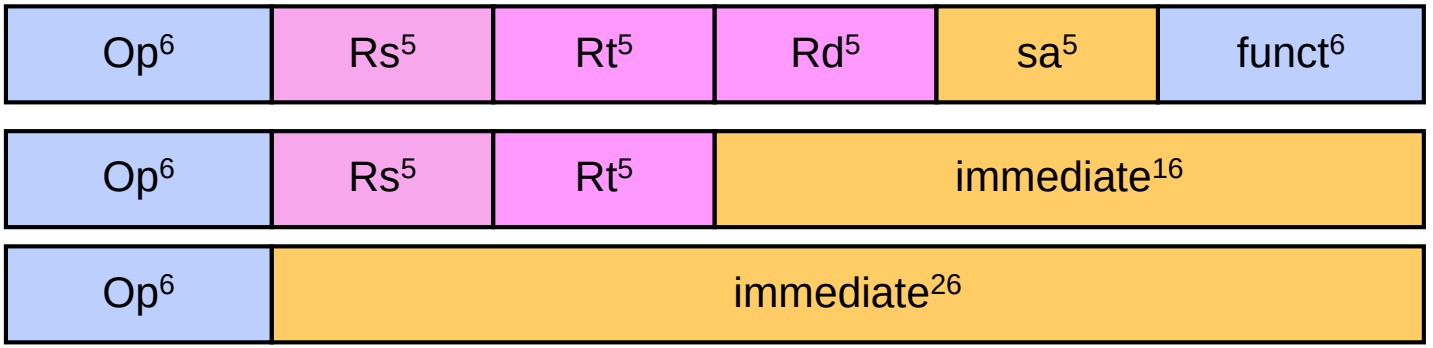
- Thiết kế bộ xử xý đơn chu kỳ:
  - Một lệnh thực hiện trong một chu kỳ (CPI =1)

Thiết kế một bộ xử lý: Các bước thực hiện

- Phân tích tập lệnh =>xác định các thành phần của datapath
- Thiết kế, lựa chọn các thành phần của datapath và phương pháp cấp xung nhịp
- Gắn các thành phần của datapath đáp ứng yêu cầu công việc của từng lệnh
  - Xác định các giá trị của các tín hiệu điều khiển cho việc điều khiển dòng lưu chuyển của dữ liệu
- Thiết kế và thêm vào bộ điều khiển

Các định dạng lệnh MIPS

- Tấc cả các lệnh: độ rộng 32-bit
- Ba loại: R-type, I-type, và J-type



- Op^6: 6-bit opcode of the instruction
- Rs^5, Rt^5, Rd^5: 5-bit source and destination register numbers
- sa^5: 5-bit shift amount used by shift instructions
- $funct^6$: 6-bit function field for R-type instructions
- $immediate^{16}$: 16-bit immediate value or address offset
- $immediate^{26}$: 26-bit target address of the jump instruction

Tập lệnh con MIPS

- Sử dụng tập lệnh con MIPS trong quá trình thiết kế
  - ALU instructions (R-type): **add, sub, and, or, xor, slt**
  - Immediate instructions (I-type): **addi, slti, andi, ori, xori**
  - Load and Store (I-type): **lw, sw**
  - Branch (I-type): **beq, bne**
  - Jump (J-type): **j**
- Tương đối đầy đủ để minh họa quá trình xây dựng

  Bộ xử lý = datapath + control
- Quá trình xây dựng bộ xử lý MIPS tương tự việc xây dựng các bộ xử ký khác

Chi tiết tập lệnh con

Instruction	Meaning	Format					
add rd, rs, rt	addition	$op^6 = 0$	rs^5	rt^5	rd^5	0	0x20
sub rd, rs, rt	subtraction	$op^6 = 0$	rs^5	rt^5	rd^5	0	0x22
and rd, rs, rt	bitwise and	$op^6 = 0$	rs^5	rt^5	rd^5	0	0x24
or rd, rs, rt	bitwise or	$op^6 = 0$	rs^5	rt^5	rd^5	0	0x25
xor rd, rs, rt	exclusive or	$op^6 = 0$	rs^5	rt^5	rd^5	0	0x26
slt rd, rs, rt	set on less than	$op^6 = 0$	rs^5	rt^5	rd^5	0	0x2a
addi rt, rs, im^{16}	add immediate	0x08	rs^5	rt^5	im^{16}		
slti rt, rs, im^{16}	slt immediate	0x0a	rs^5	rt^5	im^{16}		
andi rt, rs, im^{16}	and immediate	0x0c	rs^5	rt^5	im^{16}		
ori rt, rs, im^{16}	or immediate	0x0d	rs^5	rt^5	im^{16}		
xori rt, im^{16}	xor immediate	0x0e	rs^5	rt^5	im^{16}		
lw rt, im^{16}(rs)	load word	0x23	rs^5	rt^5	im^{16}		
sw rt, im^{16}(rs)	store word	0x2b	rs^5	rt^5	im^{16}		
beq rs, rt, im^{16}	branch if equal	0x04	rs^5	rt^5	im^{16}		
bne rs, rt, im^{16}	branch not equal	0x05	rs^5	rt^5	im^{16}		
j im^{26}	jump	0x02	im^{26}				

Register Transfer Level (RTL)

- RTL mô tả dòng dữ liệu giữa các thanh ghi
- RTL cho biết ý nghĩa (công việc chính) của lệnh
- Tấc cả các lệnh được nạp từ địa chỉ trong thanh ghi PC

Instruction	RTL Description	
ADD	Reg(Rd) ← Reg(Rs) + Reg(Rt);	PC ← PC + 4
SUB	Reg(Rd) ← Reg(Rs) – Reg(Rt);	PC ← PC + 4
ORI	Reg(Rt) ← Reg(Rs) | zero_ext(Im16);	PC ← PC + 4
LW	Reg(Rt) ← MEM[Reg(Rs) + sign_ext(Im16)];	PC ← PC + 4
SW	MEM[Reg(Rs) + sign_ext(Im16)] ← Reg(Rt);	PC ← PC + 4
BEQ	if (Reg(Rs) == Reg(Rt)) PC ← PC + 4 + 4 × sign_extend(Im16) else PC ← PC + 4	

Instructions are Executed in Steps

- **R-type**
  - Nạp lệnh: Instruction ← MEM[PC]
  - Nạp toán hạn: data1 ← Reg(Rs), data2 ← Reg(Rt)
  - Thực hiện phép toán: ALU_result ← func(data1, data2)
  - Ghi vào thanh ghi: Reg(Rd) ← ALU_result
  - Chuẩn bị lệnh kế: PC ← PC + 4
- **I-type**
  - Nạp lệnh : Instruction ← MEM[PC]
  - Nạp toán hạn : data1 ← Reg(Rs), data2 ← Extend(imm16)
  - Thực hiện phép toán : ALU_result ← op(data1, data2)
  - Ghi vào thanh ghi : Reg(Rt) ← ALU_result
  - Chuẩn bị lệnh kế : PC ← PC + 4
- **BEQ**
  - Nạp lệnh : Instruction ← MEM[PC]
  - Nạp toán hạn : data1 ← Reg(Rs), data2 ← Reg(Rt)
  - Kiểm tra bằng: zero ← subtract(data1, data2)
  - Rẽ nhánh: if (zero) PC ← PC + 4 + 4×sign_ext(imm16)
    else PC ← PC + 4

Instruction Execution – cont'd

- **LW**

Nạp lệnh :	Instruction ← MEM[PC]
Nạp thanh ghi nền:	base ← Reg(Rs)
Tính địa chỉ:	address ← base + sign_extend(imm16)
Đọc ô nhớ:	data ← MEM[address]
Ghi vào thanh ghi:	Reg(Rt) ← data
Chuẩn bị lệnh kế :	PC ← PC + 4

- **SW**

Nạp lệnh :	Instruction ← MEM[PC]
Nạp thanh ghi:	base ← Reg(Rs), data ← Reg(Rt)
Tính địa chỉ:	address ← base + sign_extend(imm16)
Ghi vào ô nhớ:	MEM[address] ← data
Chuẩn bị lệnh kế :	PC ← PC + 4

- **Jump**

Nạp lệnh :	Instruction ← MEM[PC]
Tính địa chỉ đích:	target ← PC[31:28] || Imm26 || '00'
Nhảy:	PC ← target

concatenation

Các thành phần yêu cầu từ tập lệnh

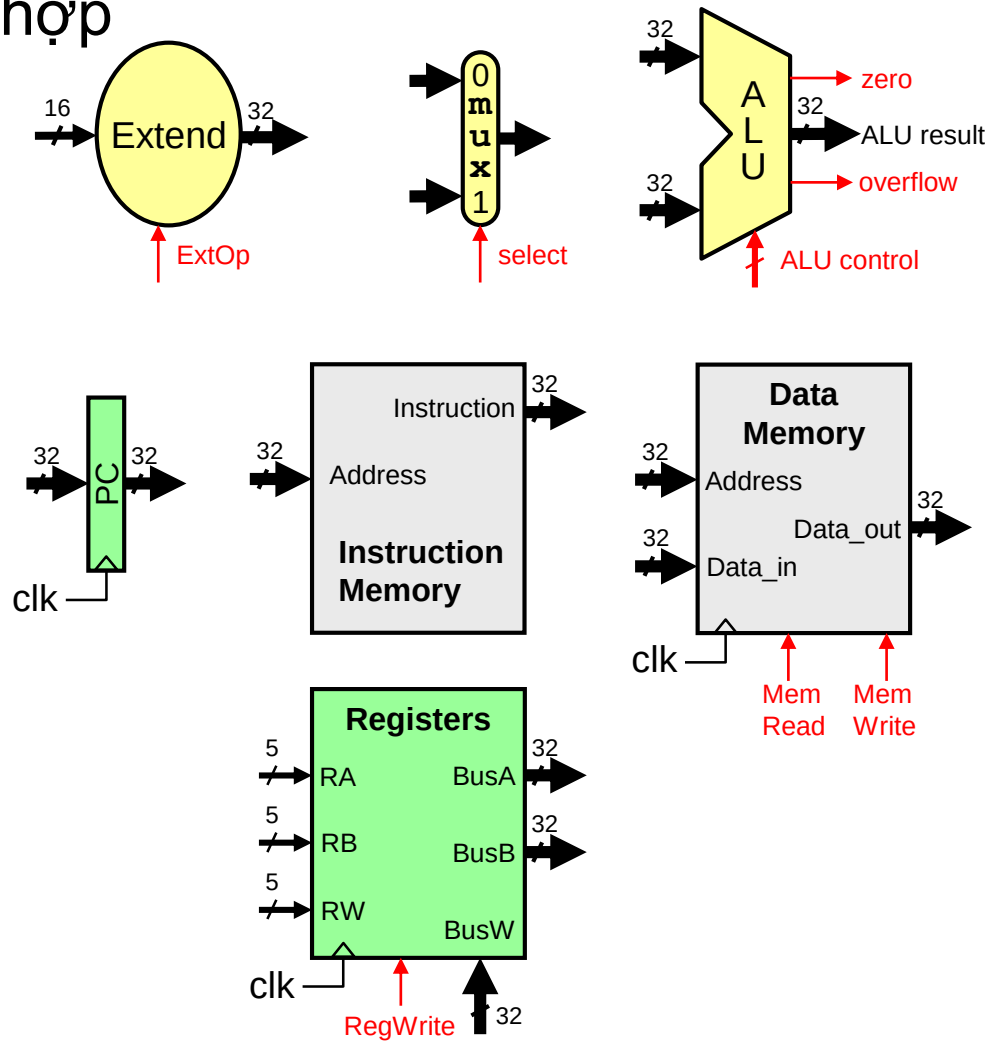
- Bộ nhớ
  - Bộ nhớ lệnh là nơi chứa lệnh
  - Bộ nhớ dữ liệu là nơi chứa dữ liệu
- Bộ thanh ghi
  - 31 × 32-bit thanh ghi đa dụng, R0 luôn bằng giá trị 0
  - Đọc thanh ghi nguồn Rs
  - Đọc thanh ghi nguồn Rt
  - Ghi vào thanh ghi đích Rt hoặc Rd
- Bộ đếm chương trình (thanh ghi PC) và Bộ cộng để tăng PC = PC + 4
- Bộ mở rộng dấu và 0 cho hằng số 16 bit
- Bộ tính toán số học luận lý ALU thực hiện tính toán

Tiếp theo . . .

- Thiết kết bộ xử lý: Các bước thực hiện
- Các thành phần của Datapath và cấp xung nhịp
- Xây dựng Datapath đầy đủ
- Điều khiển quá trình thực thi của các lệnh
- Bộ điều khiển chính và bộ điều khiển ALU
- Hạn chế của thiết kế bộ xử lý đơn chu kỳ

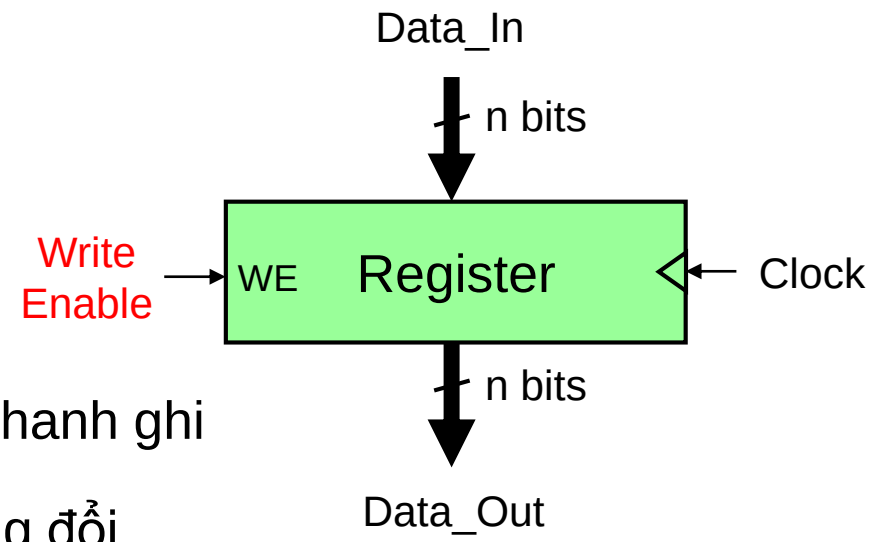
Các thành phần của Datapath

- Các phần tử mạch tổ hợp
  - ALU, Adder
  - Immediate extender
  - Multiplexers
- Các phần tử lưu trữ
  - Instruction memory
  - Data memory
  - PC register
  - Register file
- Xung nhịp
  - Đồng bộ quá trình ghi



Thanh ghi

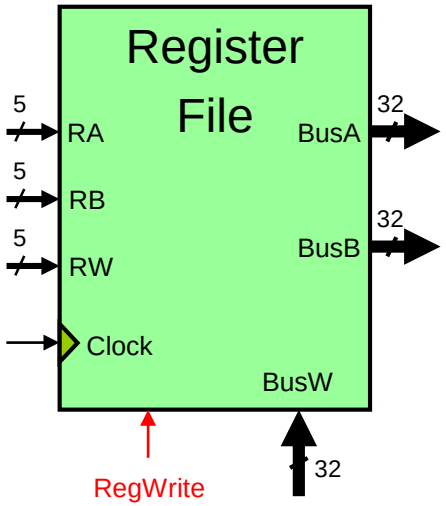
- Thanh ghi
  - Tương tự D Flip-Flop
- n-bit vào và ra
- Write Enable (WE):
  - Cho phép/ cấm ghi vào thanh ghi
  - Cấm (0): Data_Out không đổi
  - Cho phép (1): Data_Out = Data_In sau cạnh lên của xung nhịp
- Xung nhịp kích cạnh lên (0 -> 1)
  - Giá trị output được thay đổi tại cạnh lên của xung nhịp



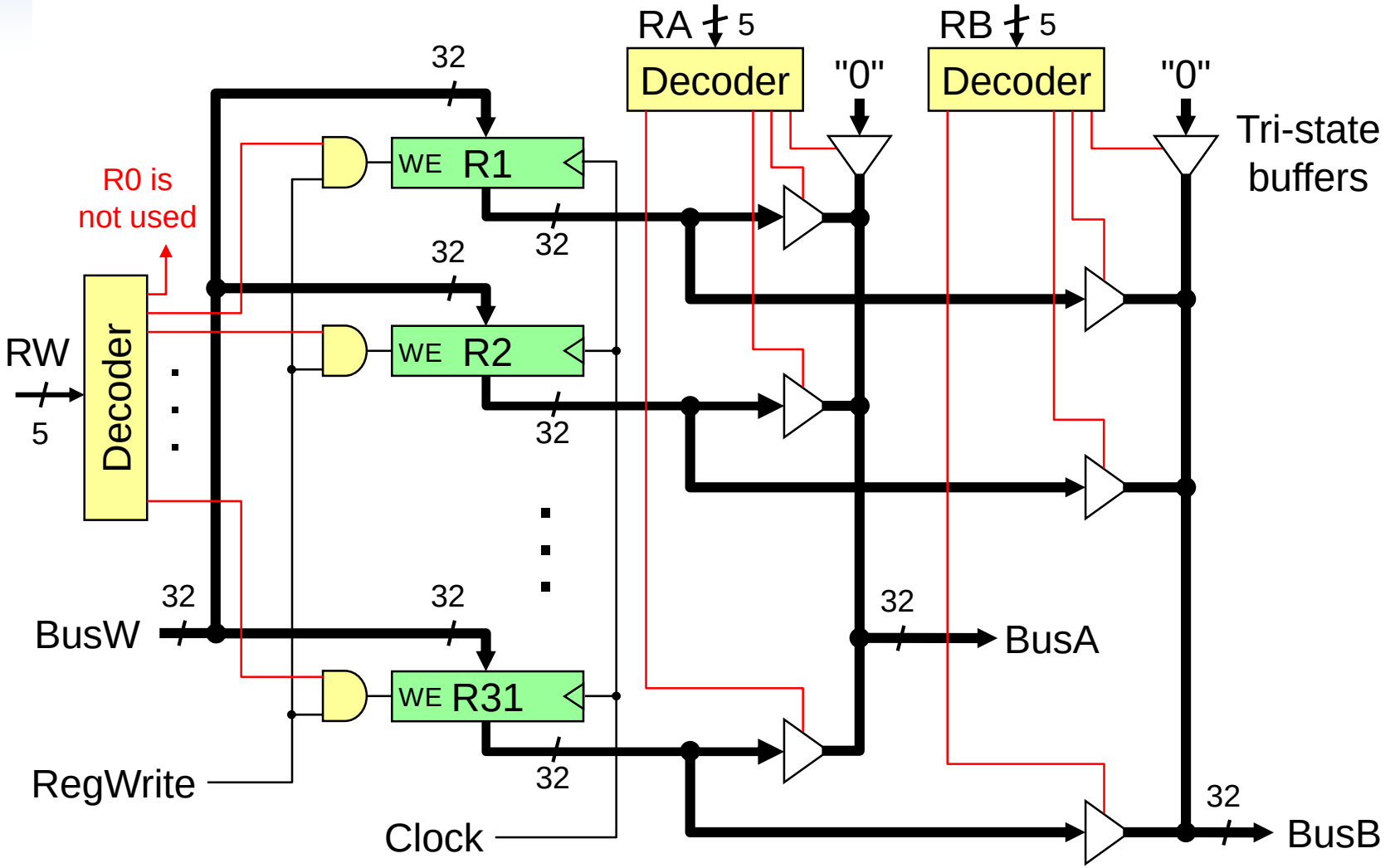
Bộ thanh ghi MIPS (Register File)

RW RA RB

- Bộ thanh ghi bao gồm 32 × 32-bit thanh ghi
  - BusA và BusB: 32-bit ngõ ra cho 2 toán hạn nguồn
  - BusW: 32-bit ngõ vào để ghi giá trị vào thanh ghi khi RegWrite = 1
- Lựa chọn thanh ghi:
  - RA lựa chọn thanh ghi đọc cho giá trị ở BusA
  - RB lựa chọn thanh ghi đọc cho giá trị ở BusB
  - RW lựa chọ thanh ghi được ghi vào
- Xung nhịp
  - Xung nhịp sử dụng khi GHI (cạnh lên)
  - Khi đọc, bộ thanh ghi như là một mạch tổ hợp
    - RA, RB hợp lệ => BusA, BusB là giá trị tương ứng sau thời gian truy xuất

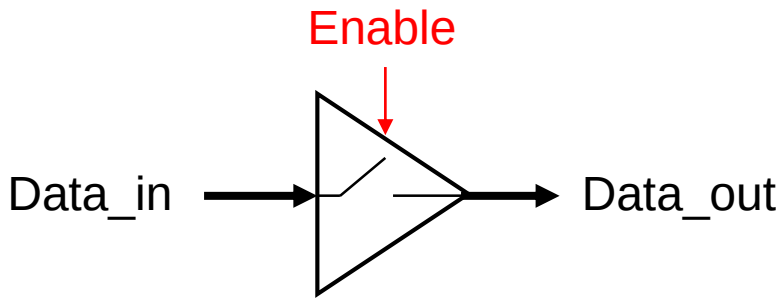


Chi tiết bộ thanh ghi



Bộ đệm 3 trạng thái

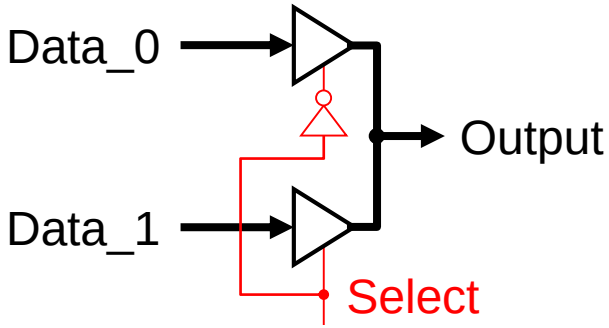
- Cho phép nhiều nguồn sử dụng chung một bus
- Hai ngõ vào:
  - Data_in
  - Enable (to enable output)



- Một ngõ ra: Data_out
  - If (Enable) Data_out = Data_in

    else Data_out = High Impedance state (output bị ngắt)

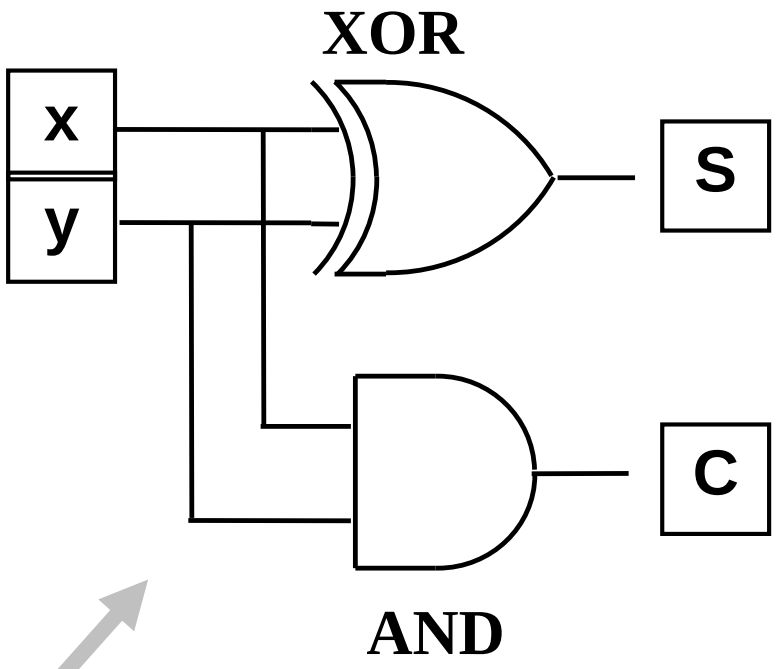
- Bộ đệm 3 trạng thái được sử dụng tạo thành bộ hợp kênh (multiplexor)



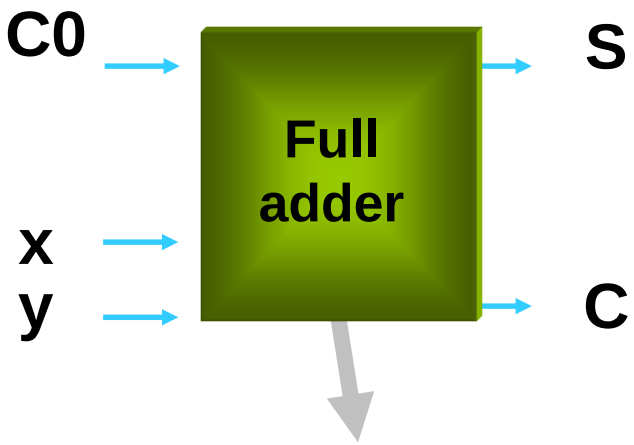
Mạch Half Adder

XOR AND

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Mạch Full Adder



S = x + y + C0

S = (x + y) + C0

Tính: S1 = x + y

Tính: S2 = S1 + C0

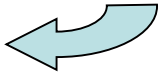
Half adder 1

Half adder 2

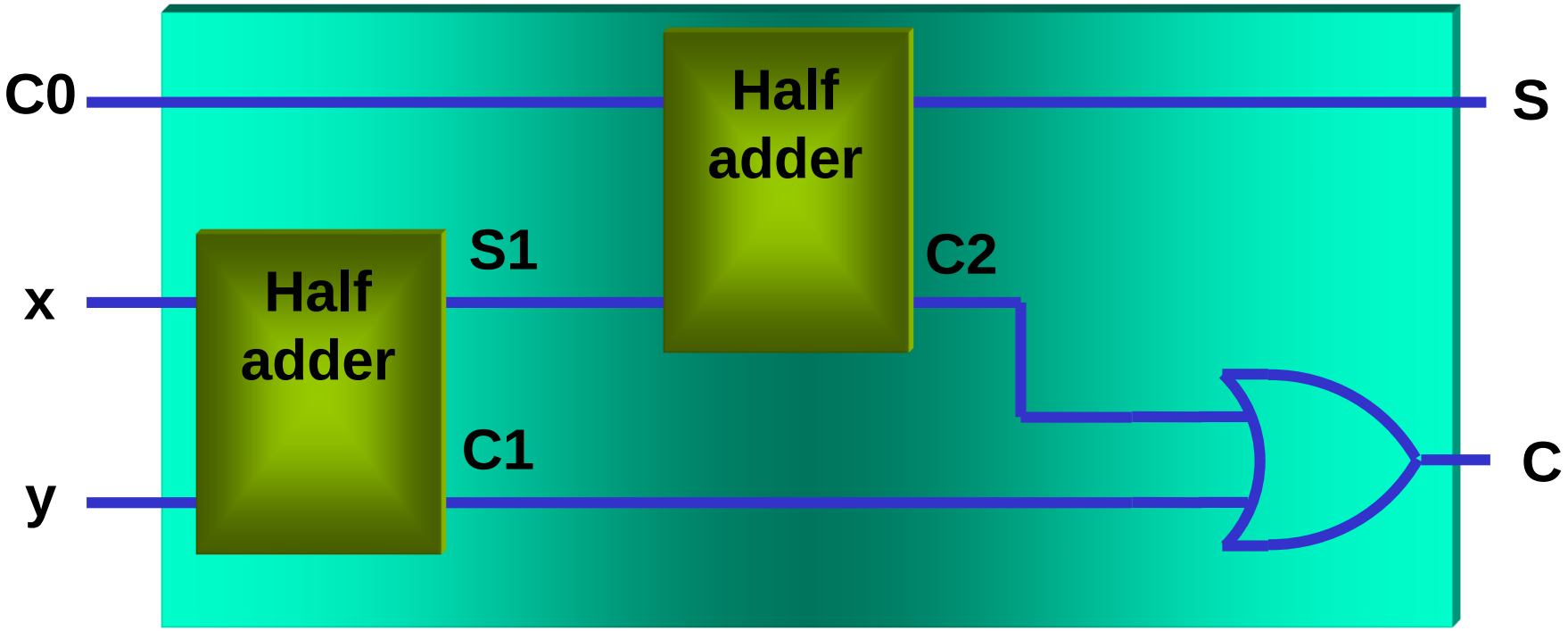
Full adder (2)

C_0	x	y	S	C	C_0	S_1	C_1	C_2	C
0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0	0	0
0	1	0	1	0	0	1	0	0	0
0	1	1	0	1	0	0	1	0	1
1	0	0	1	0	1	0	0	0	0
1	0	1	0	1	1	1	0	1	1
1	1	0	0	1	1	1	0	1	1
1	1	1	1	1	1	0	1	0	1

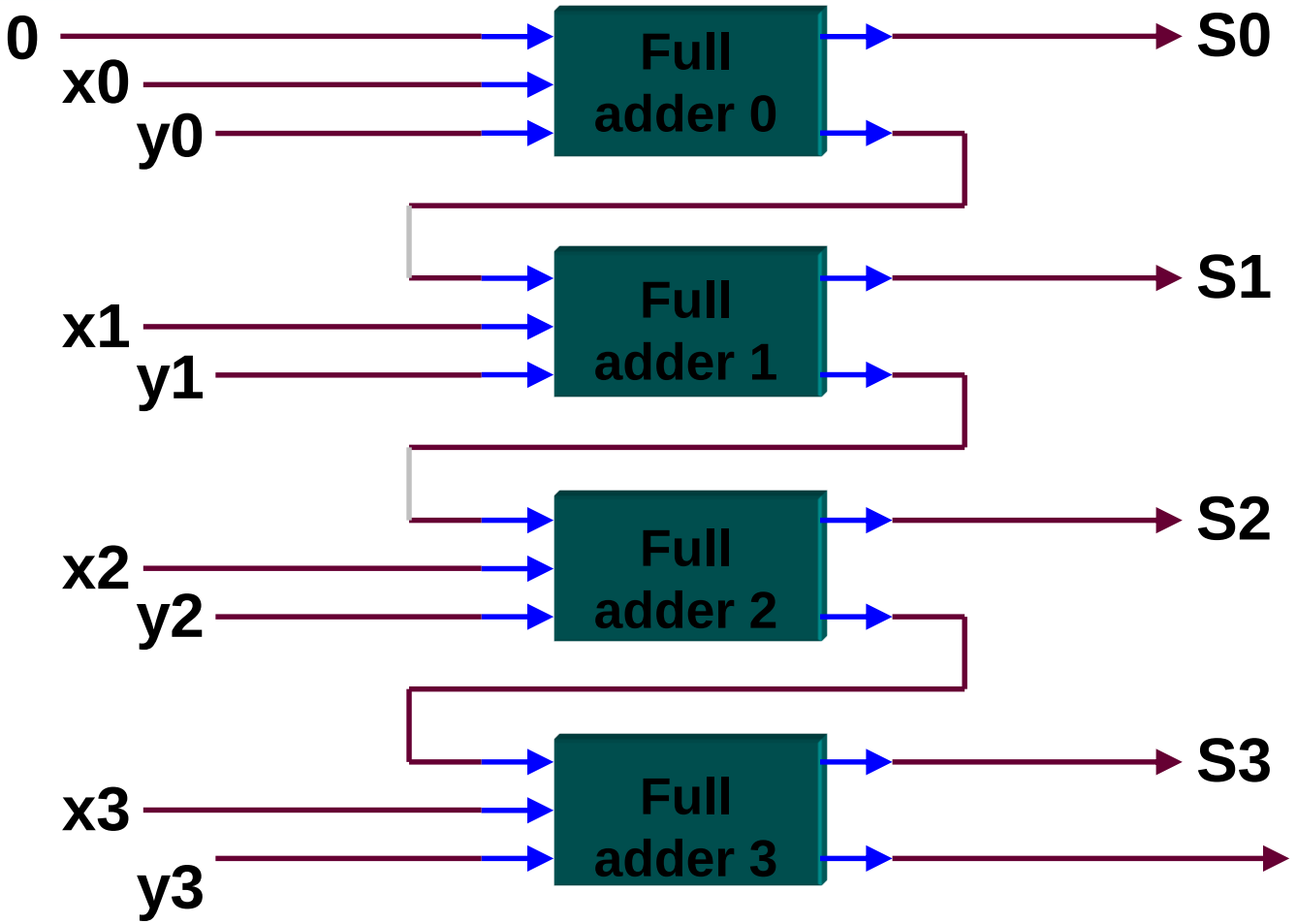
C = 1 when C1 = 1 or C2 = 1



Full adder (3)



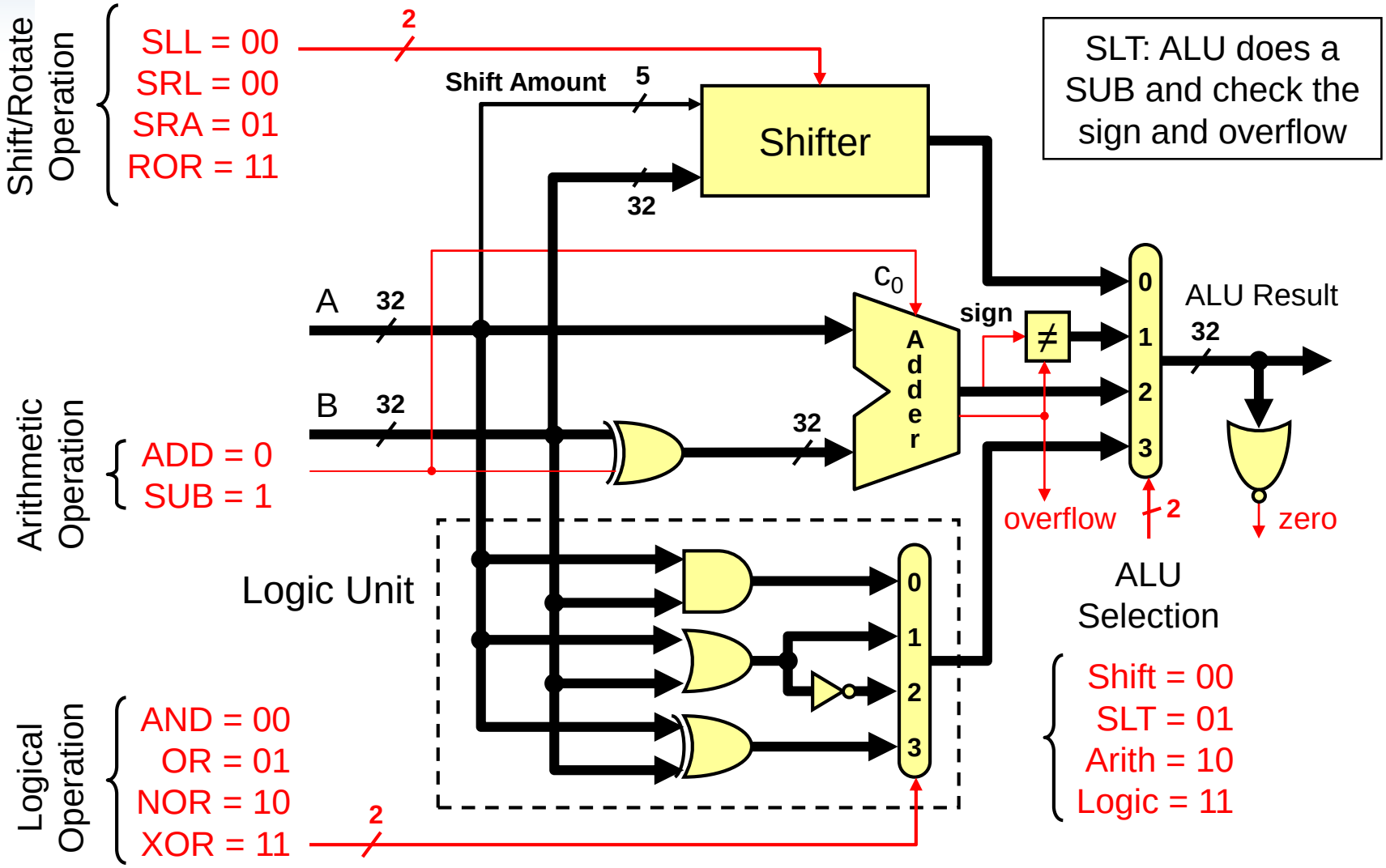
Cộng nhiều Bits



$$\begin{array}{r} x_3x_2x_1x_0 \\ + \quad y_3y_2y_1y_0 \\ \hline C\ S_3S_2S_1S_0 \end{array}$$

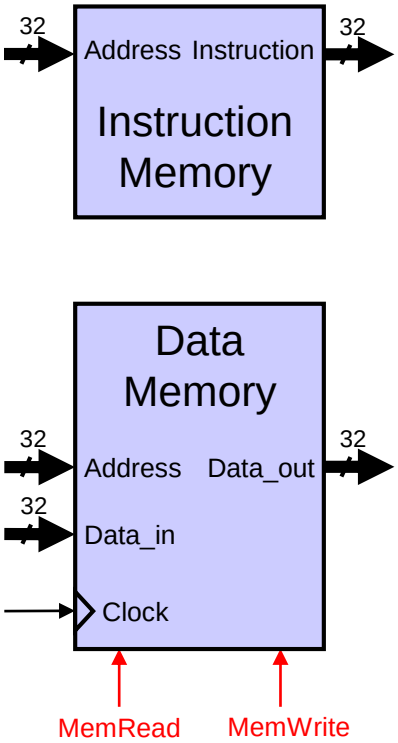
C

Xây dựng bộ ALU



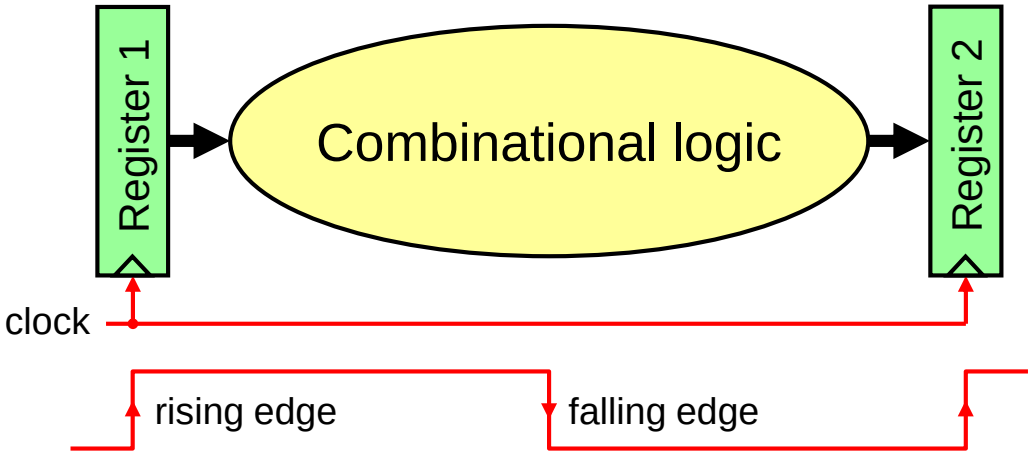
Bộ nhớ lệnh và bộ nhớ dữ liệu

- Bộ nhớ lệnh chỉ cần thao tác đọc
  - Datapath không thực hiện việc ghi lệnh
  - Chức năng giống mạch tổ hợp cho thao tác đọc
  - Address lựa chọn Instruction sau access time
- Bộ nhớ dữ liệu dùng cho lệnh load và store
  - MemRead: cho phép dữ liệu ra tại Data_out
    - Address lựa chọn ô nhớ để đưa ra Data_out
  - MemWrite: cho phép ghi dữ liệu từ Data_in
    - Address lựa chọn ô nhớ sẽ được ghi vào
    - Clock đồng bộ thao tác ghi
- Tách biệt bộ nhớ lệnh và bộ nhớ dữ liệu
  - Sau này đượ thay thế thành bộ nhớ đệm



Phương pháp cấp xung nhịp

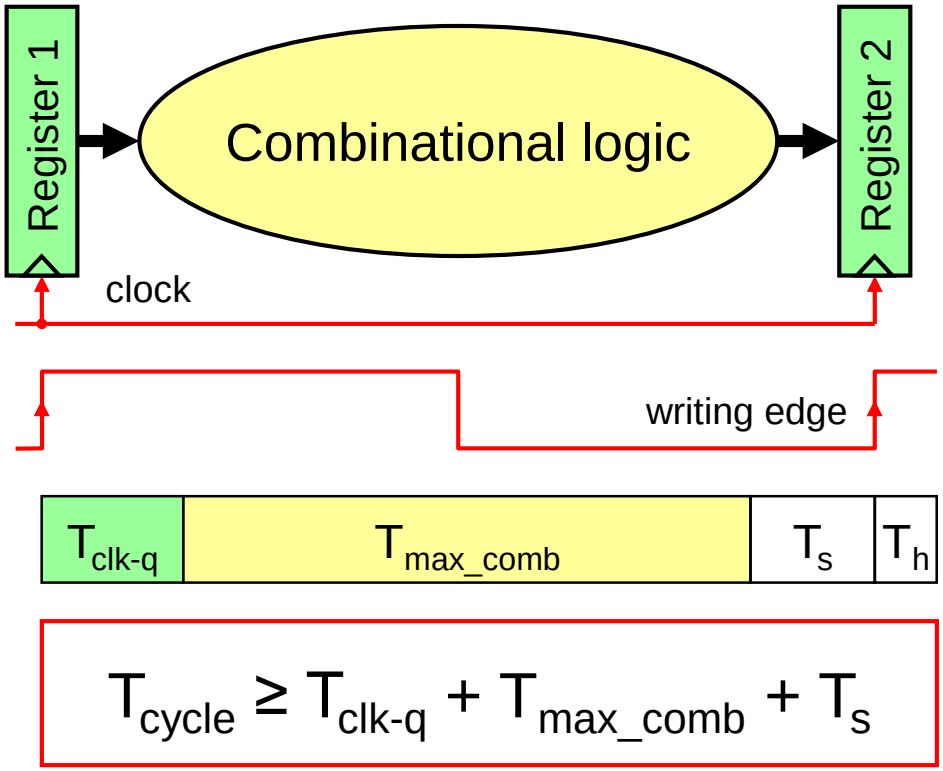
- Xung nhịp cần thiết cho các thành phần tuần tự (lưu trữ) cho biết khi nào cập nhật (ghi) vào
- Phương pháp cấp xung nhịp xác định khi nào dữ liệu có thể được ghi và được đọc



- Sử dụng cạnh lên
- Tấc cả trạng thái thay đổi tại thời điểm cạnh lên
- Dữ liệu phải hợp lệ và ổn định trước thời điểm cạnh lên
- Cho phép đọc và ghi được thực hiện trong cùng một chu kỳ xung nhịp

Xác định chu kỳ xung nhịp

- Với phương pháp kích cạnh lên, chu kỳ xung nhịp phải đủ dài để thỏa mãn thời gian cần thiết để dữ liệu hợp lệ và ổn định khi đi từ một thanh ghi qua mạch tổ hợp rồi đến thanh ghi khác



- T_{clk-q} : thời gian trễ từ cạnh lên đến khi dữ liệu mới hợp lệ trên đầu ra của thanh ghi
- T_{max_comb} : thời gian trễ dài nhất cho mạch tổ hợp
- T_s : (setup time) thời gian cần thiết để dữ liệu ổn định trước lúc cạnh lên
- T_h: (hold time) thời gian dữ liệu cần duy trùy sau khi có cạnh lên
- Hold time (T_h) không quan trọng vì $T_{clk-q} > T_h$

Clock Skew

- Clock skew sinh ra vì các tín hiệu xung nhịp sử dụng các đường đi khác nhau để đến các phần tử tuần tự
- Clock skew là sự sai biệt thời gian để hai phần tử tuần tự nhìn thấy cạnh xung nhịp
- Với clock skew, chu kỳ xung nhịp tăng lên

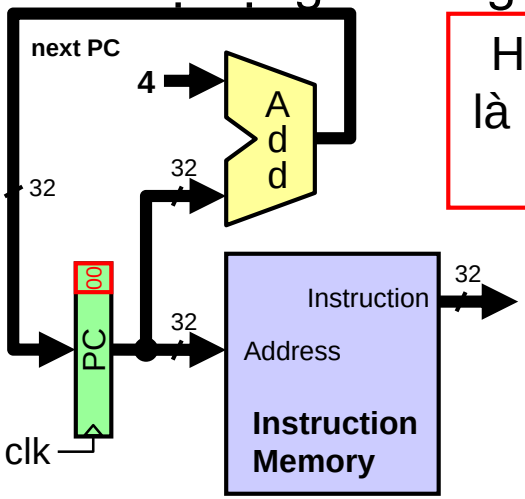
$$T_{cycle} \geq T_{clk\text{-}q} + T_{max\_combinational} + T_{setup} + T_{skew}$$

Tiếp theo . . .

- Thiết kết bộ xử lý: Các bước thực hiện
- Các thành phần của Datapath và cấp xung nhịp
- Xây dựng Datapath đầy đủ
- Điều khiển quá trình thực thi của các lệnh
- Bộ điều khiển chính và bộ điều khiển ALU
- Hạn chế của thiết kế bộ xử lý đơn chu kỳ

Đường dữ liệu (datapath) cho việc nạp lệnh

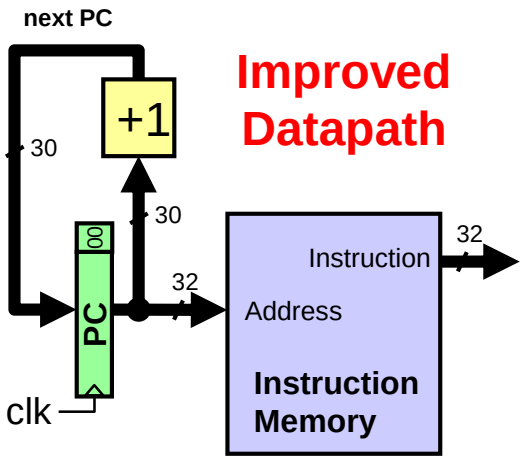
- Bắt đầu thực hiện xây dựng datapath từ các thành phần đã được định danh trong bước 1
- Cho việc nạp lệnh, chúng ta cần…
  - Thanh ghi bộ đếm chương trình (PC)
  - Bộ nhớ lệnh
  - Bộ cộng để tăng PC = PC + 4



Hai bit cuối của PC luôn là '00' vì đị chỉ lệnh là một số chia hết cho 4

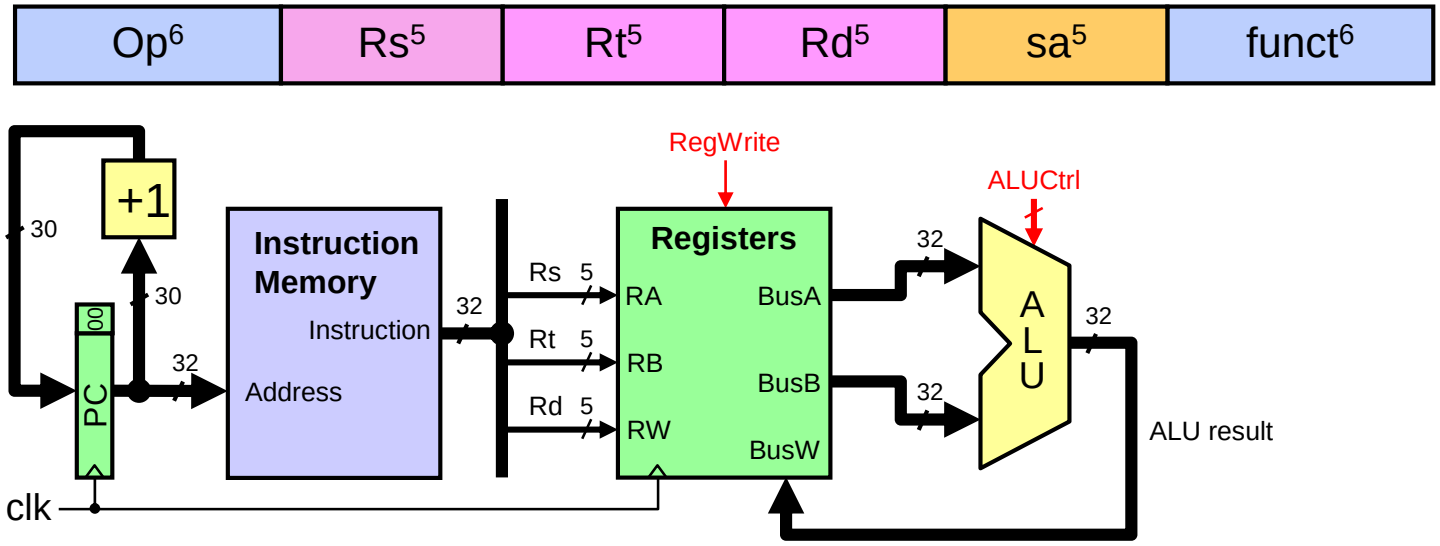
Chưa hỗ trợ lệnh nhảy và rẽ nhánh

Cải tiến: cộng 30 bit cao PC lên 1



Improved Datapath

Datapath cho lệnh R-type



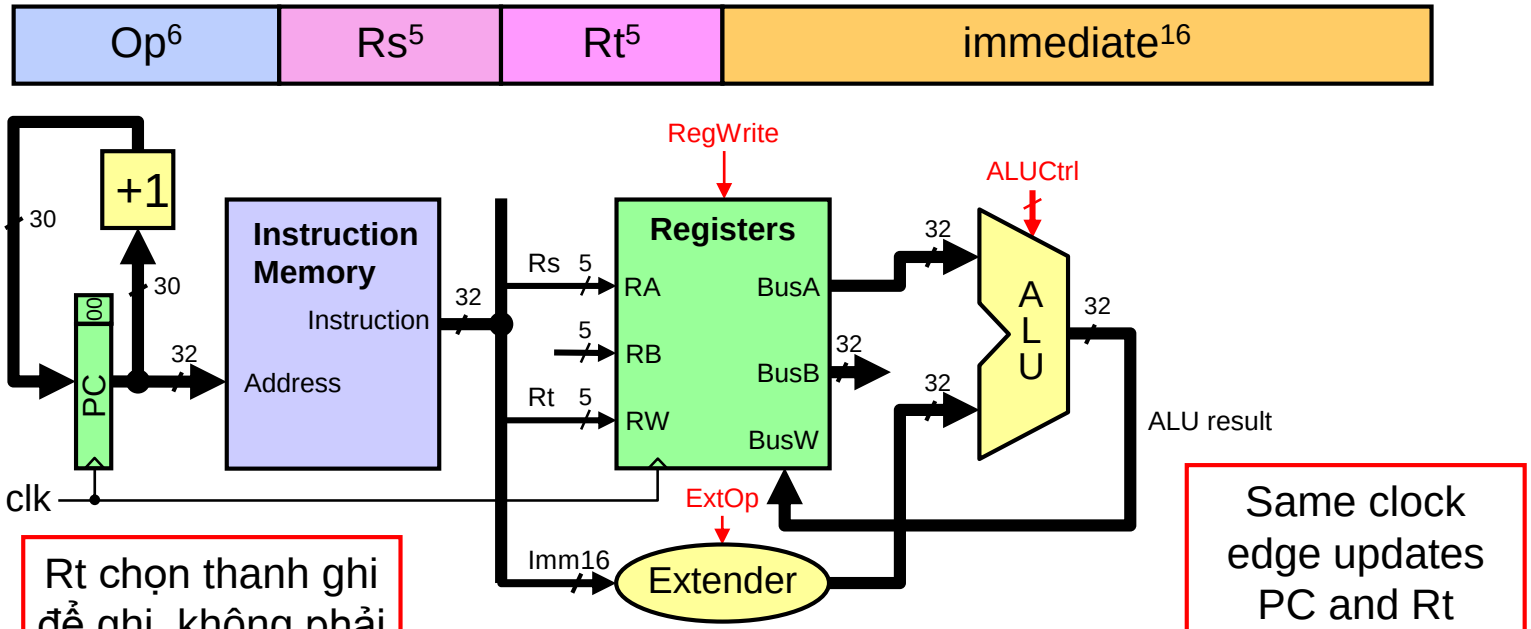
Rs và Rt chỉ ra hai thanh ghi đọc giá trị (BusA & BusB). Rd chọn thanh ghi để ghi

BusA & BusB là hai toán hạn của ALU. Kết quả từ ALU nối vào BusW để ghi

Clock cập nhật PC và thanh ghi Rd giống nhau

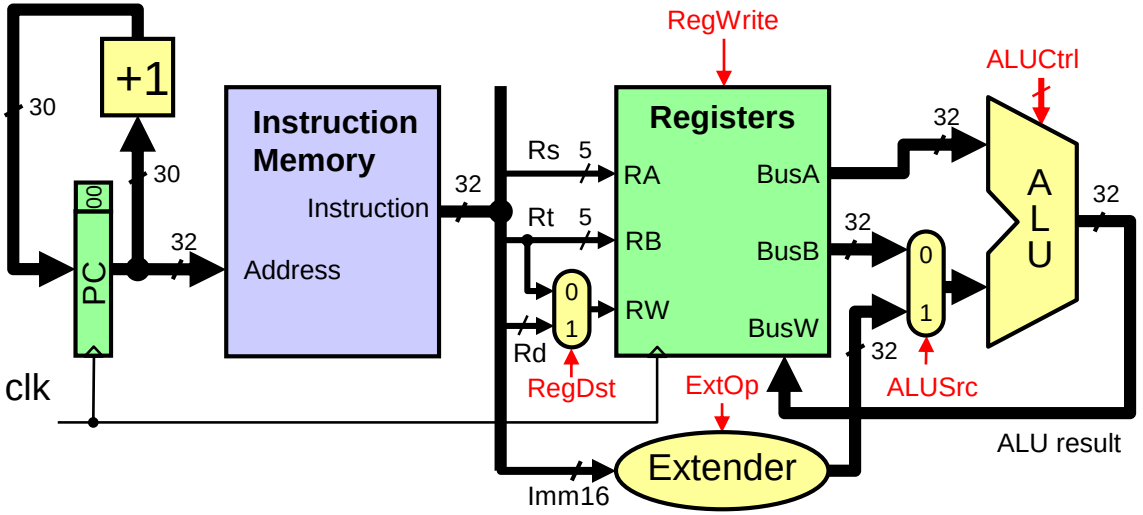
- Các tín hiệu điều khiển
  - ALUCtrl chọn lựa phép toán theo trường funct vì Op = 0 cho R-type
  - RegWrite cho phép ghi kết quả ALU vào thanh ghi Rd

Datapath cho lệnh I-type (ALU)



- Các tín hiệu điều khiển
  - ALUCtrl lựa chọn phép toán từ trường Op
  - RegWrite cho phép ghi kết quả ALU result vào thanh ghi Rt
  - ExtOp lựa chọn mở rộng dấu/không cho hằng số 16-bit

Kết hợp Datapath R-type & I-type

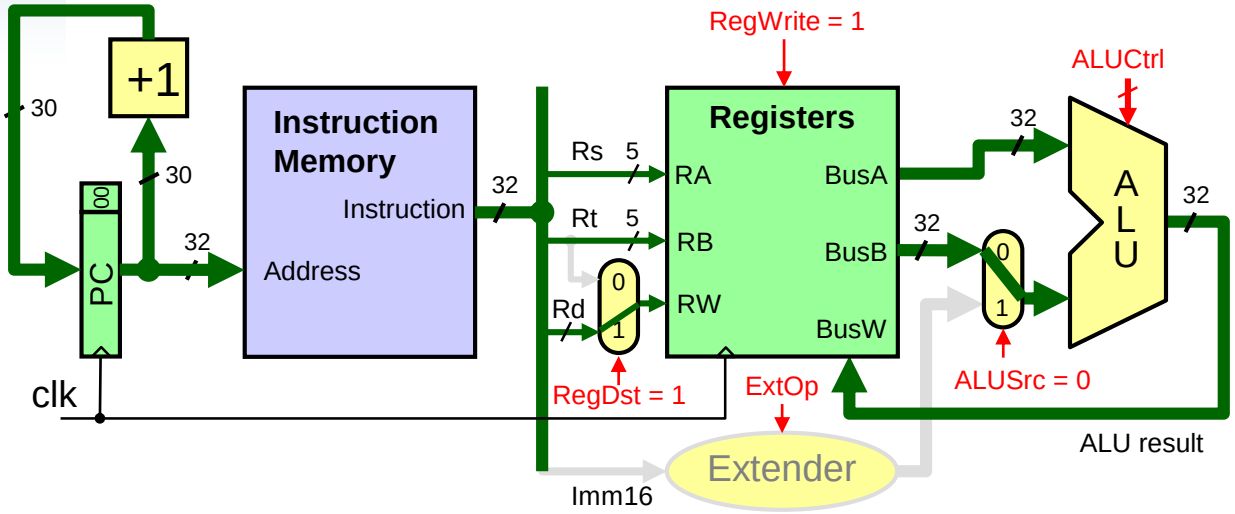


Thêm 1 mux lựa chọn RW từ Rt hoặc Rd

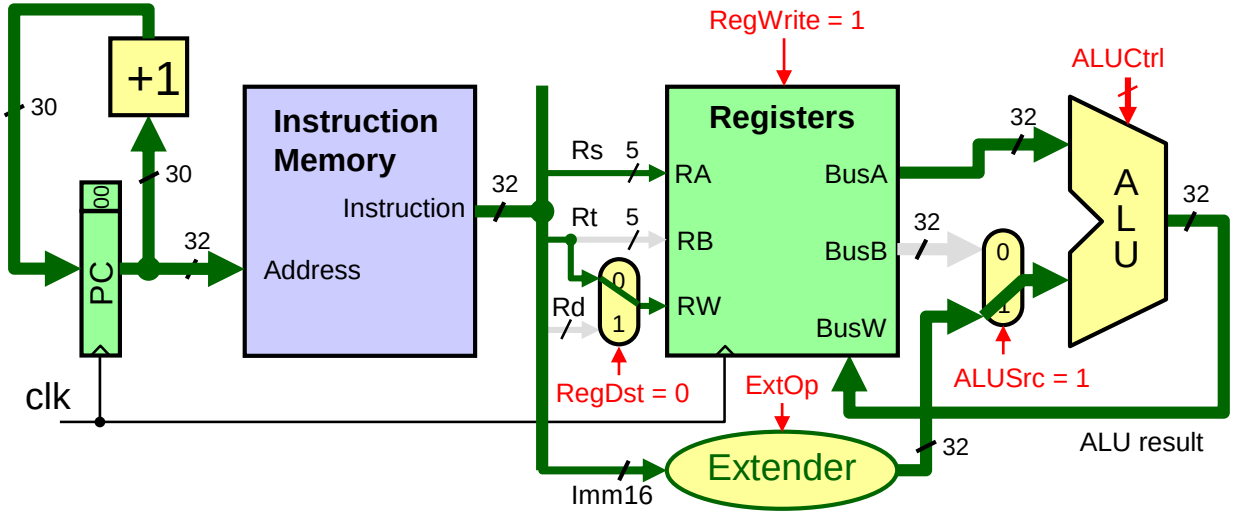
Thêm mux lựa chọn toán hạn thứ 2 của ALU từ BusB hoặc từ bộ mở rộng hằng số 16->32bit

- Các tín hiệu điều khiển
  - ALUCtrl lựa chọn phép toán ALU từ trường Op hoặc funct
  - RegWrite cho phép ghi kết quả ALU result vào thanh ghi
  - ExtOp chọn mở rộng dấu/không cho hằng số 16-bit
  - RegDst chọn lựa thanh ghi ghi từ Rt hoặc Rd
  - ALUSrc chọn toán hạn thứ 2 của ALU từ BusB hoặc Extender

Tổng hợp hoạt động các lệnh ALU



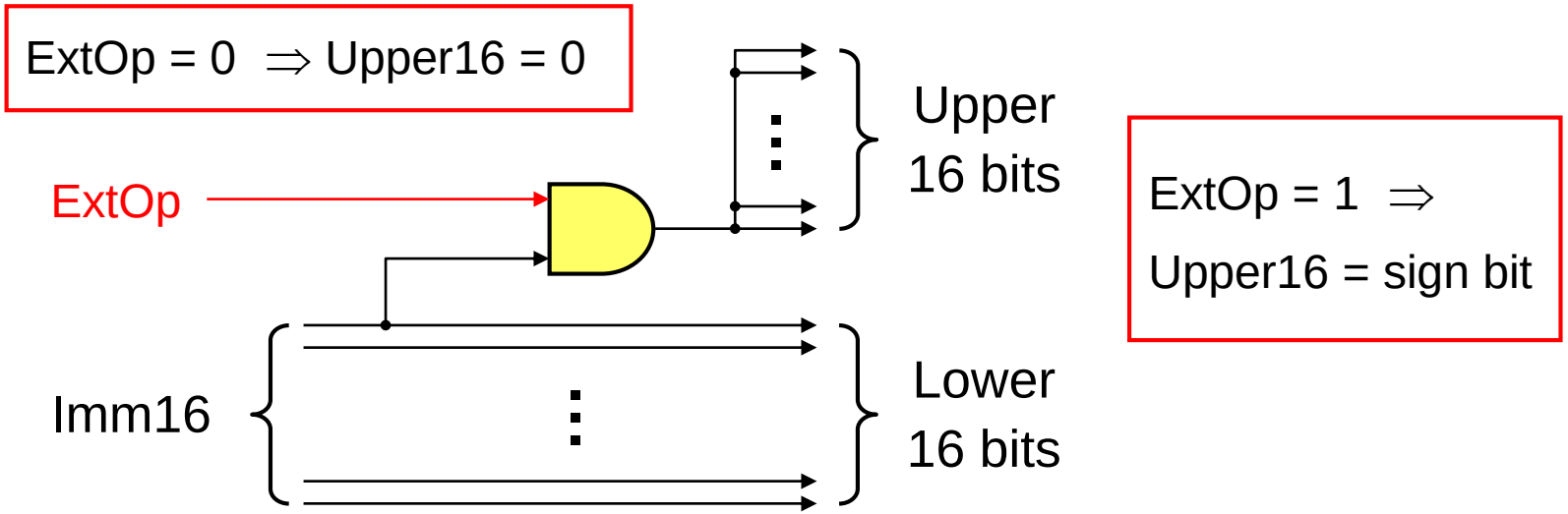
Cho lệnh ALU R-type: RegDst = '1' chọn RW = Rd, ALUSrc = '0' chọn toán hạn thứ 2 ALU từ BusB. Datapath mong muốn minh họa bằng màu **xanh lá**



Cho lệnh ALU I-type: RegDst = '0' chọn RW = Rt, ALUSrc = '1' chọn toán hạn thứ 2 ALU từ Extender. Datapath mong muốn minh họa bằng màu **xanh lá**

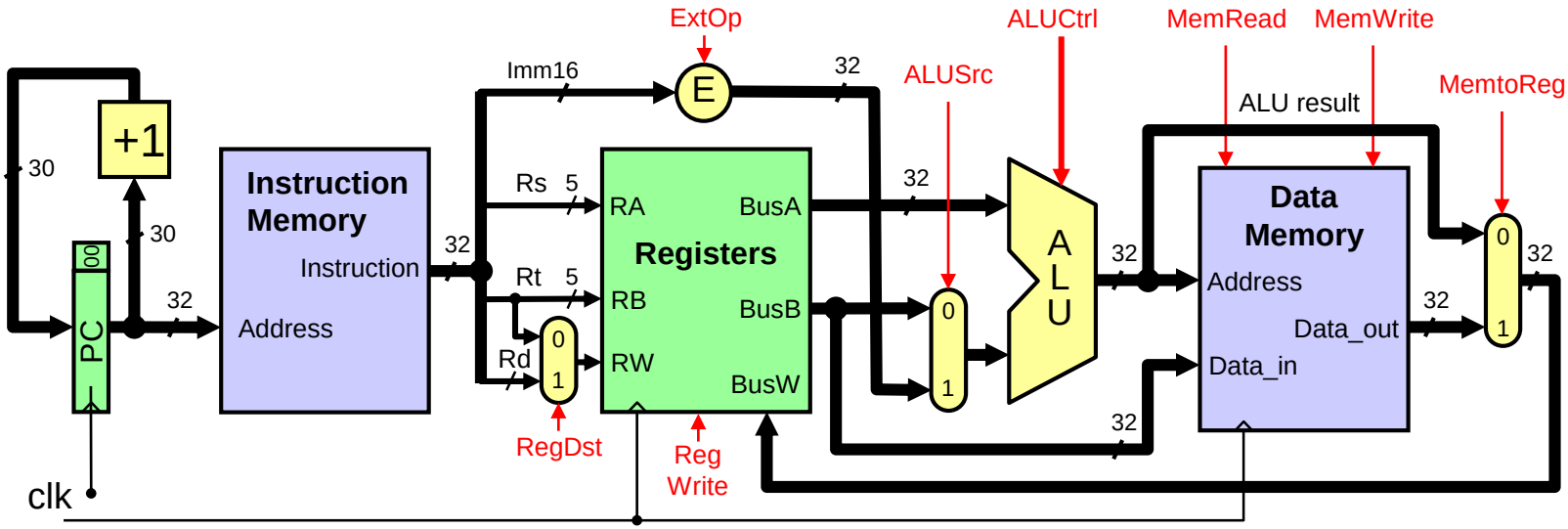
Chi tiết bộ mở rộng hằng số (Extender)

- Hỗ trợ hai kiểu mở rộng
  - Mở rộng không (Zero-extension) cho hằng số kiểu unsigned
  - Mở rộng dấu (Sign-extension) cho hằng số kiểu signed
- Tín hiệu điều khiển ExtOp chọn kiểu mở rộng
- Hiện thực Extender : sử dụng 1 cổng AND



Thêm bộ nhớ dữ liệu (Data Memory) vào Datapath

- Data memory được thêm vào cho lệnh load và store



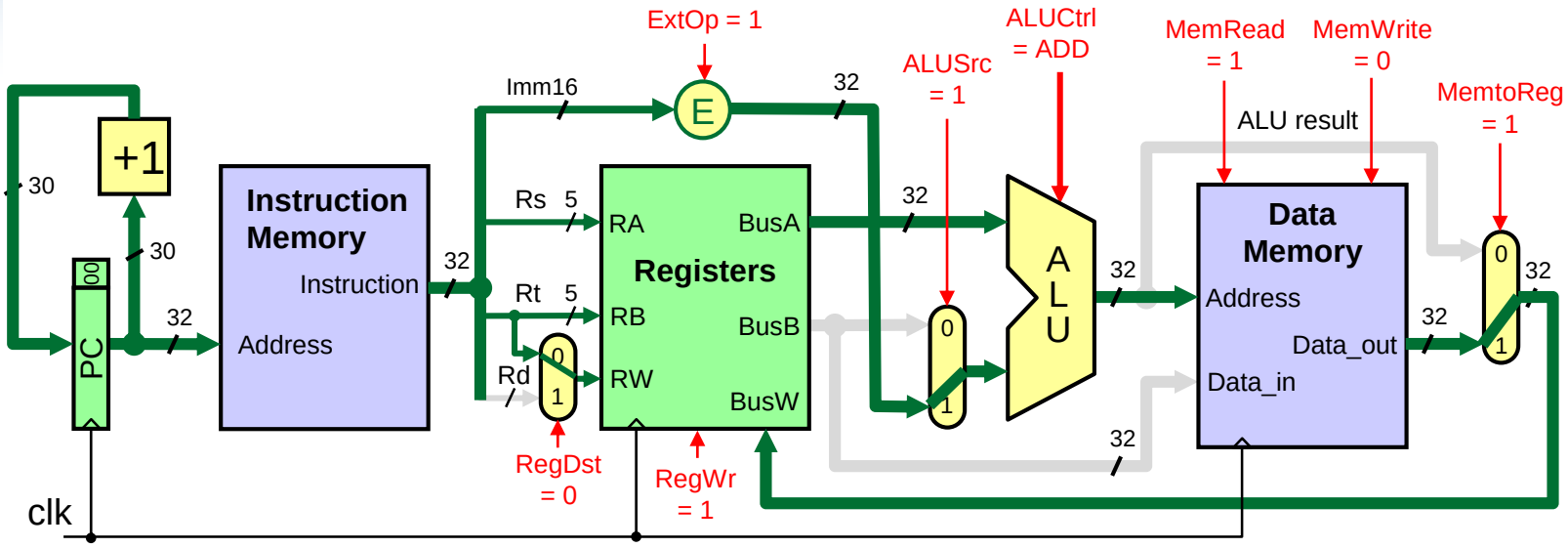
ALU tính địa chỉ của Data Memory

Thêm mux thứ 3 chọn dữ liệu BusW từ kết quả ALU hoặc từ Data_out của Data Memory

BusB is connected to Data_in of Data Memory for store instructions

- Các tín hiệu điều khiển mới
  - MemRead yêu cầu đọc (lw)
  - MemWrite yêu cầu ghi (sw)
  - MemtoReg chọn dữ liệu của BusW là kết quả từ ALU result hoặc từ Memory Data_out

Hoạt động của lệnh "Load"



RegDst = '0' chọn Rt là thanh ghi đích

RegWrite = '1' cho phép ghi giá trị vào thanh ghi

ExtOp = 1 thực hiện mở rộng dấu 16 -> 32 bit

ALUSrc = '1' lựa chọn toán hạn thứ 2 của ALU từ bộ Extender

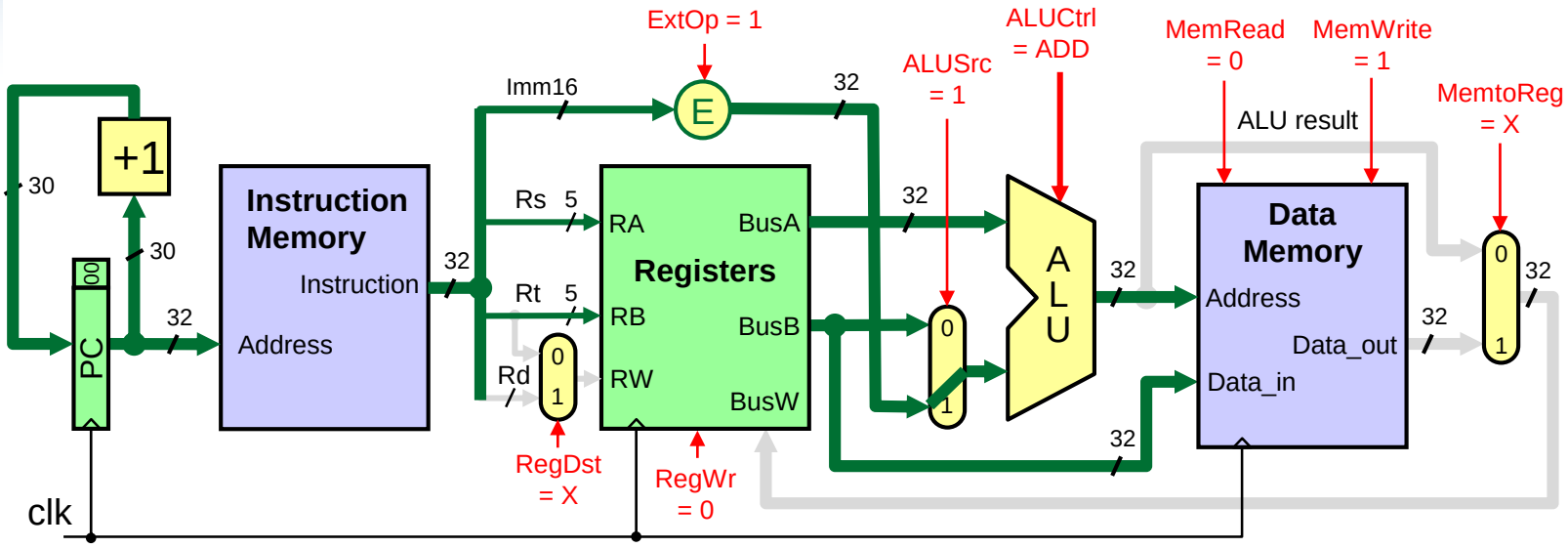
ALUCtrl = 'ADD' tính địa chỉ truy xuất bộ nhớ Reg(Rs) + sign-extend(Imm16)

MemRead = '1' yêu cầu đọc dữ liệu từ bộ nhớ

MemtoReg = '1' đưa dữ liệu từ bộ nhớ ra BusW để ghi vào thanh ghi đích

Clock cập nhật PC và Rt

Hoạt động của lệnh "Store"



RegDst = 'X' vì không thực hiện ghi

RegWrite = '0' không cho phép ghi vào thanh ghi

ExtOp = 1 thực hiện mở rộng dấu hằng số 16 -> 32 bit

ALUSrc = '1' chọn toán hạn thứ 2 của ALU từ bộ Extender

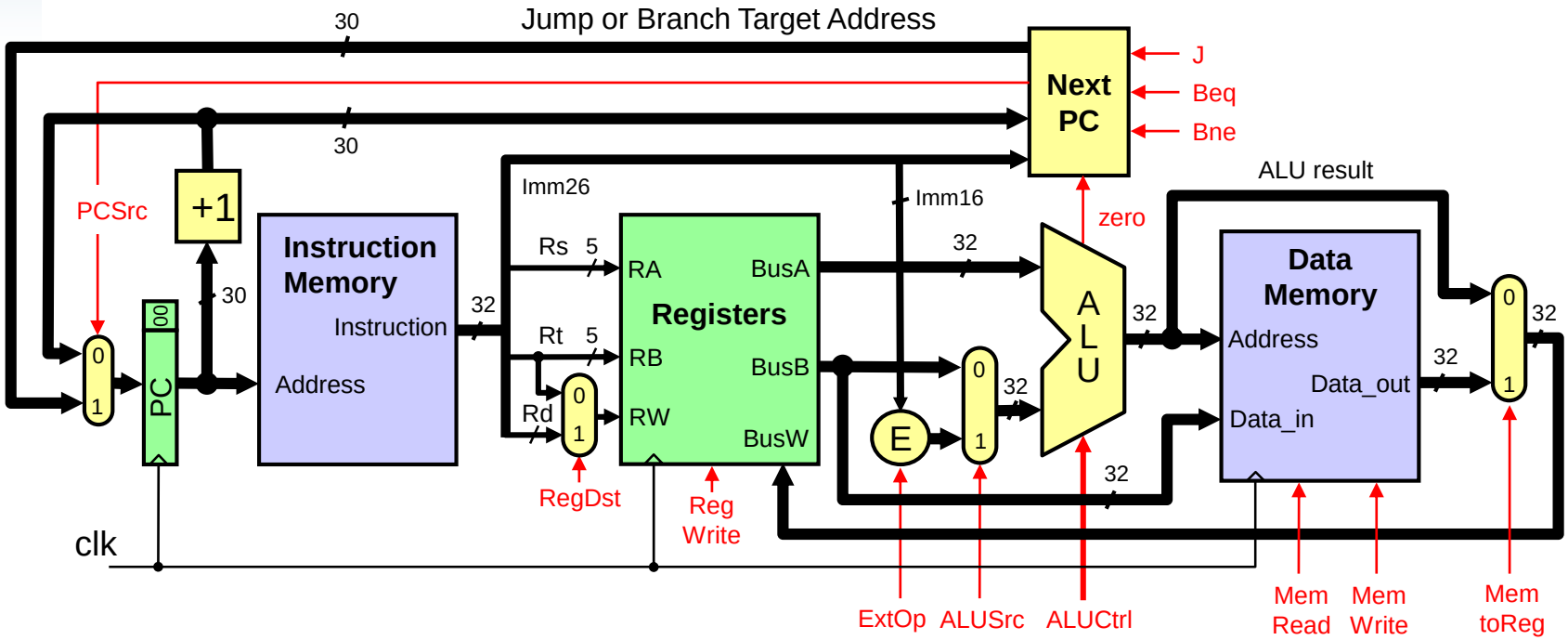
ALUCtrl = 'ADD' tính toán địa chỉ truy xuất bộ nhớ Reg(Rs) + sign-extend(Imm16)

MemWrite = '1' yêu cầu ghi dữ liệu vào bộ nhớ

MemtoReg = 'X' dữ liệu xuất hiện trên BusW không quan trọng

Clock cập nhật PC và giá trị vào bộ nhớ

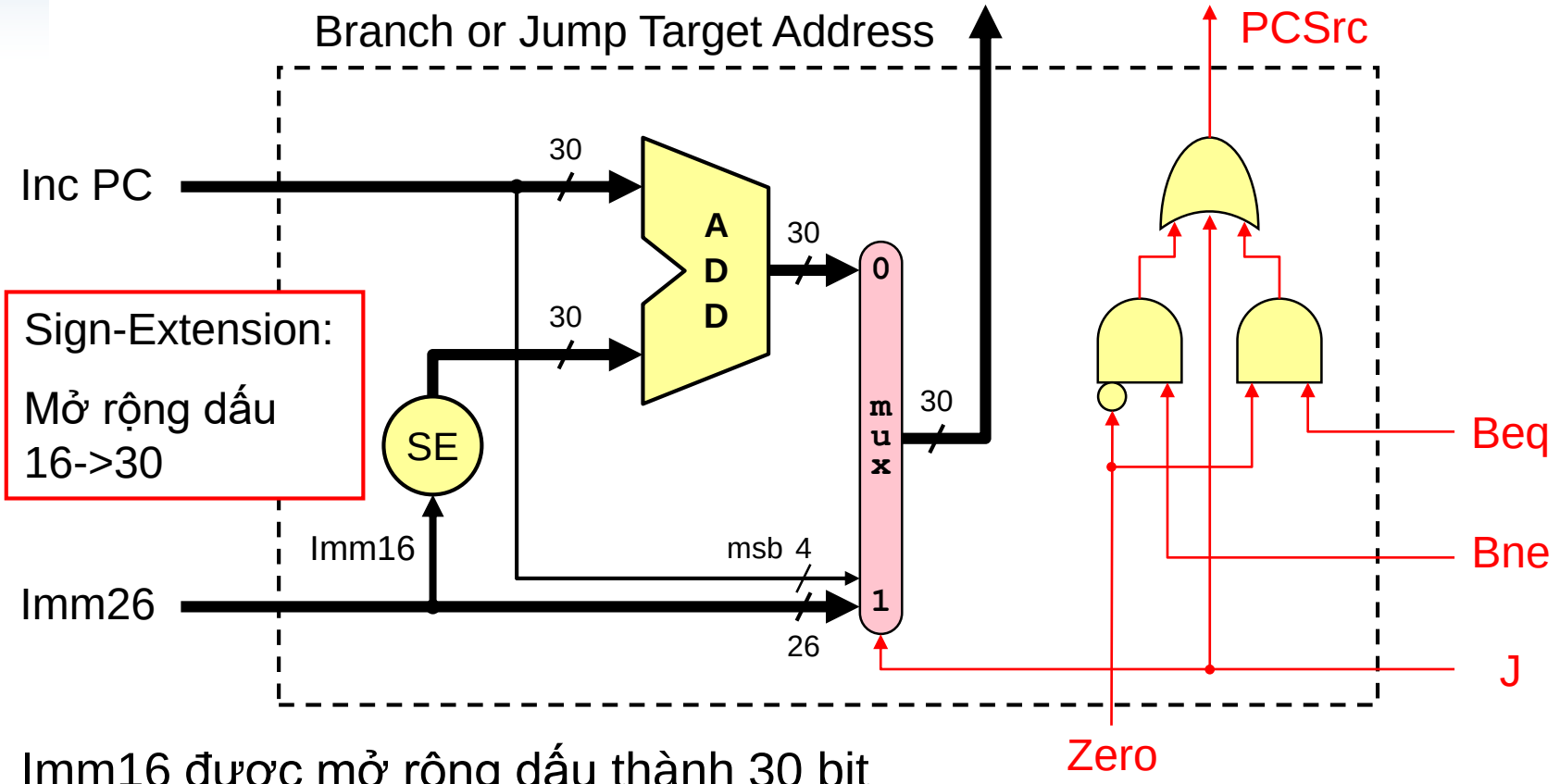
Thêm các thành phần hỗ trợ lệnh "Jump" và "Branch" vào Datapath



- Các tín hiệu điều khiển mới
  - J, Beq, Bne cho các lệnh nhảy và rẽ nhánh
  - Cờ Zero từ ALU, Zero = 1 khi ALU_result = 0
  - PCSrc = 1 cho lệnh nhảy & rẽ nhánh xảy ra

Next PC logic thực hiện tính địa chỉ nhảy đến cho lệnh nhảy và rẽ nhánh

Chi tiết bộ Next PC

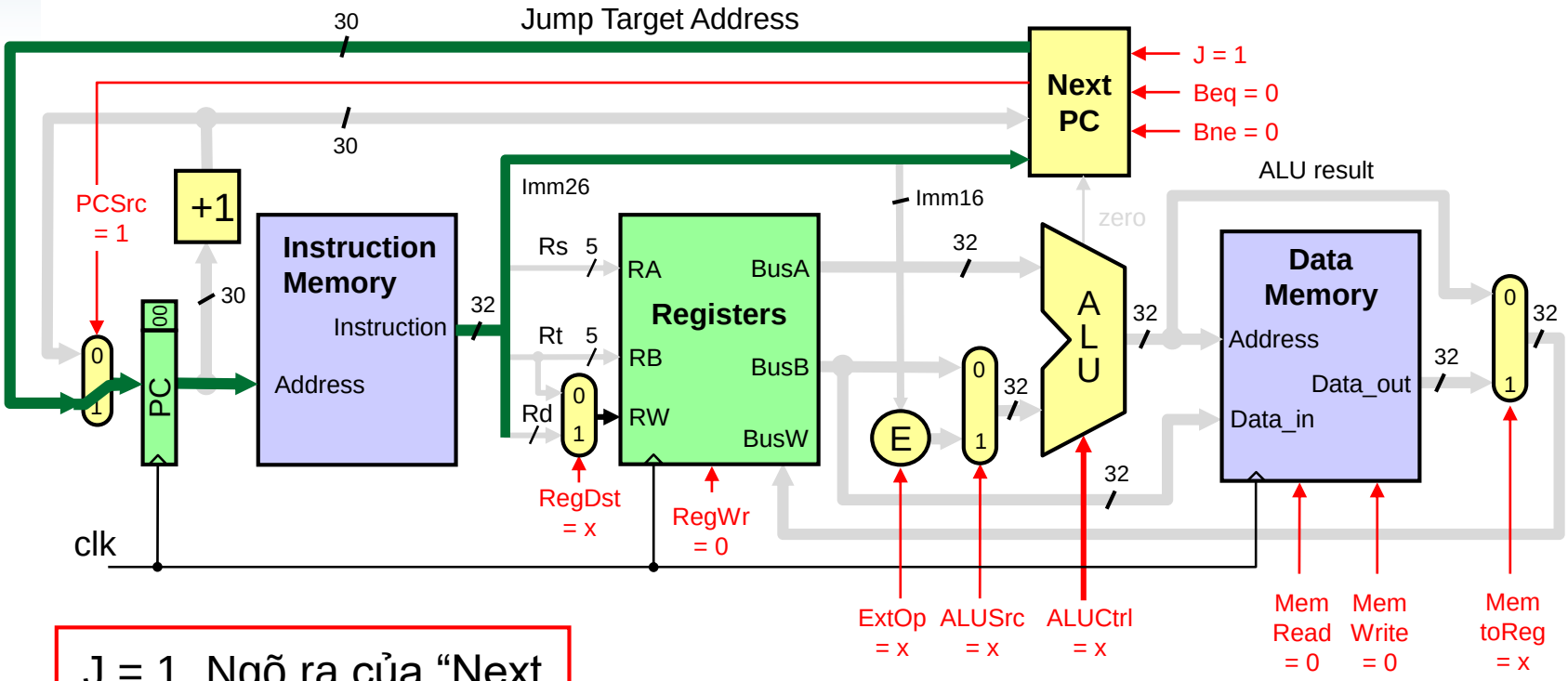


Imm16 được mở rộng dấu thành 30 bit

Địa chỉ đích của lệnh Jump: 4 bit cao của PC nối với Imm26

PCSrc = J + (Beq . Zero) + (Bne . $\overline{\text{Zero}}$)

Hoạt động của lệnh "Jump"



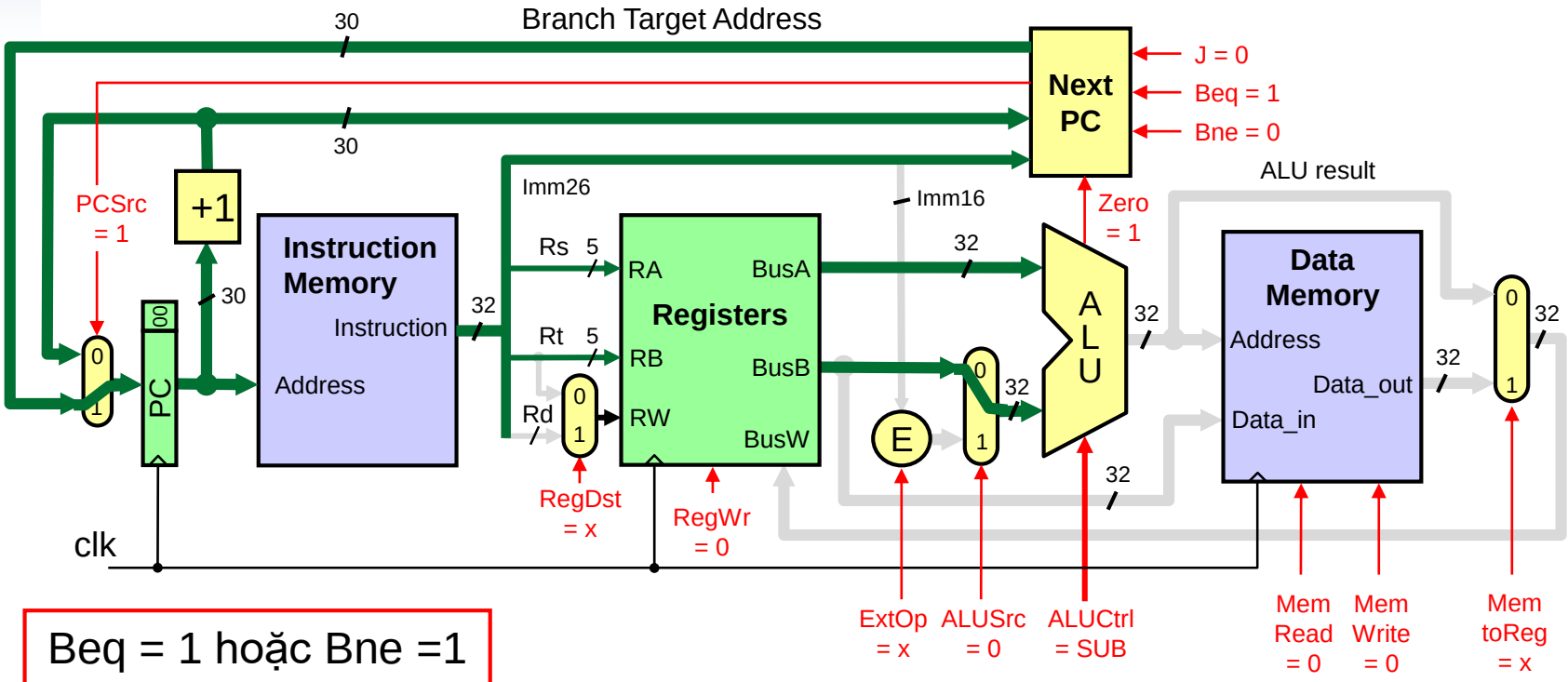
J = 1. Ngõ ra của "Next PC" là địa chỉ đích của lệnh Jump

MemRead, MemWrite, và RegWrite bằng 0

Không quan tâm: RegDst, ExtOp, ALUSrc, ALUCtrl, và MemtoReg

Clock chỉ cập nhật giá trị thanh ghi PC

Hoạt động của lệnh "rẽ nhánh"



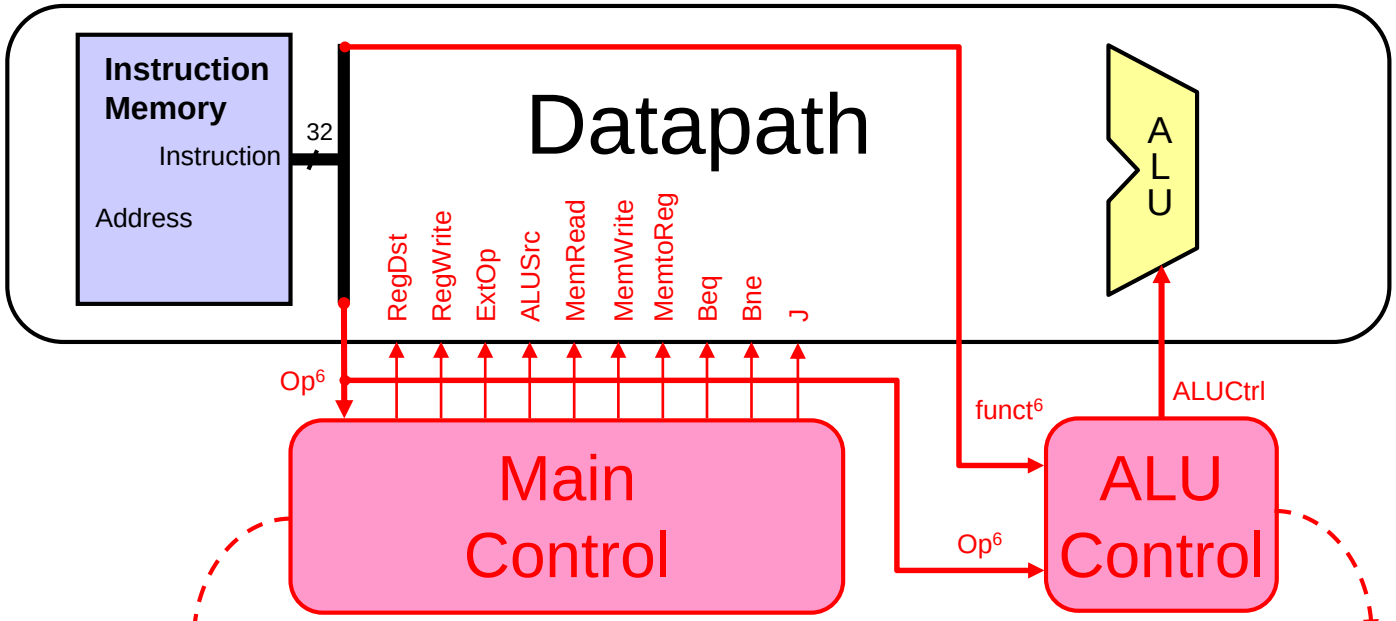
Beq = 1 hoặc Bne =1 tùy vào lệnh
ALUSrc = 0 lựa chọn giá trị trên BusB
ALUCtrl = SUB để tạo giá trị cờ Zero

Ngõ ra "Next PC" là địa chỉ đích khi rẽ nhánh PCSrc = 1 nếu rẽ nhánh xảy ra
RegWrite, MemRead và MemWrite bằng 0
Clock chỉ cập nhật giá trị thanh ghi PC

Tiếp theo . . .

- Thiết kết bộ xử lý: Các bước thực hiện
- Các thành phần của Datapath và cấp xung nhịp
- Xây dựng Datapath đầy đủ
- Điều khiển quá trình thực thi của các lệnh
- Bộ điều khiển chính và bộ điều khiển ALU
- Hạn chế của thiết kế bộ xử lý đơn chu kỳ

Bộ điều khiển chính (Main Control) và bộ điều khiển ALU (ALU Control)



Ngõ vào Main Control:

- 6-bit opcode từ 32 bit lệnh

Ngõ ra Main Control:

- 10 tín hiệu điều khiển cho Datapath

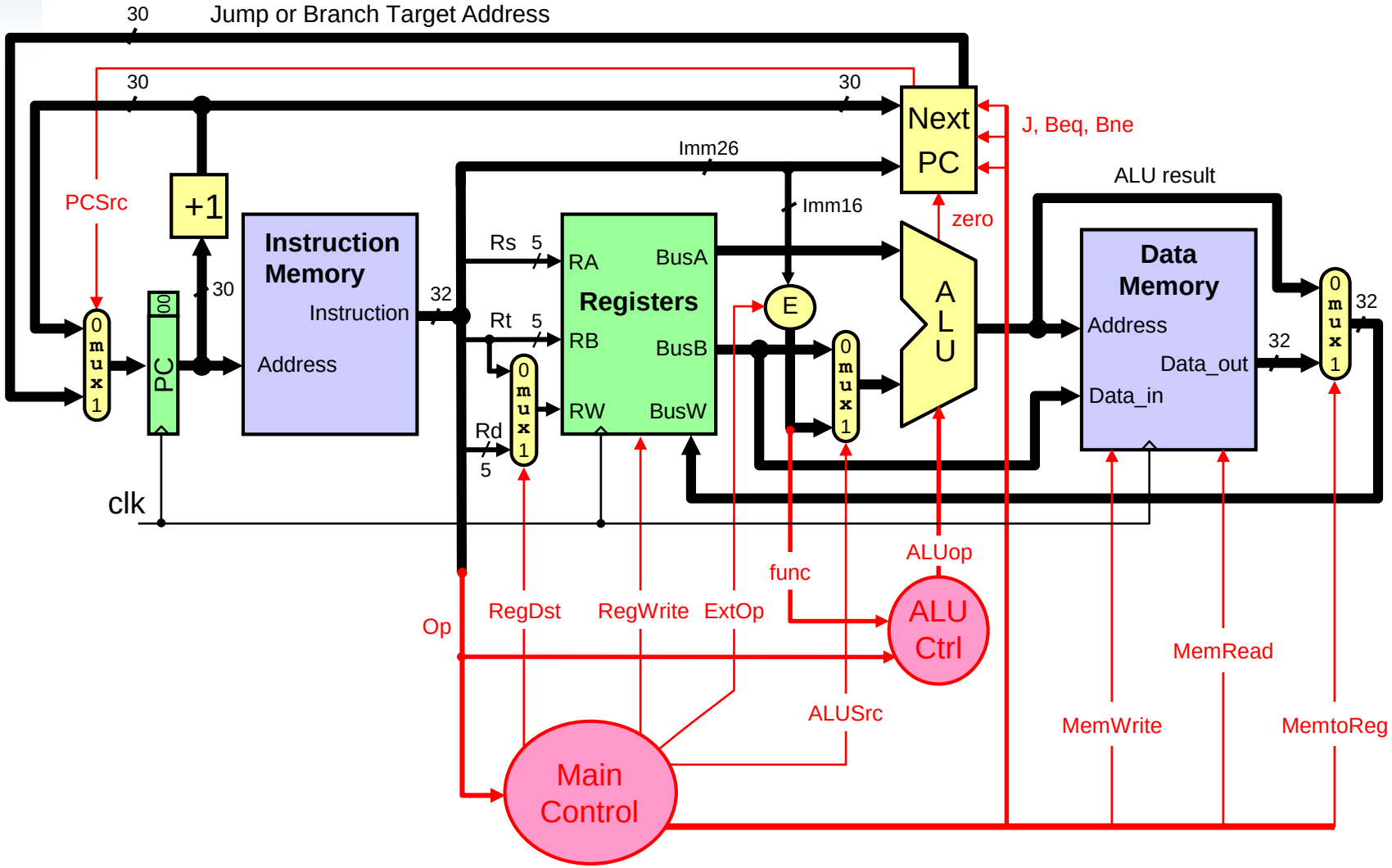
Ngõ vào ALU Control:

- 6-bit opcode từ 32 bit lệnh
- 6-bit function

Ngõ ra ALU Control:

- ALUCtrl cho bộ ALU

Bộ xử lý đơn chu kỳ Datapath + Control



Các tín hiệu ngõ ra từ Main Control

Tín hiệu	Trường hợp bằng '0'	Trường hợp bằng '1'
RegDst	Thanh ghi đích Rw = Rt	Thanh ghi đích Rw = Rd
RegWrite	Không cho phép ghi	Thanh ghi đích được ghi vào với giá trị trên BusW
ExtOp	16-bit hằng số được mở rộng không	16-bit hằng số được mở rộng dấu
ALUSrc	Toán hạn thứ 2 của ALU là giá trị từ thanh ghi Rt (BusB)	Toán hạn thứ 2 của ALU là giá trị bộ mở rộng 16->32 bit
MemRead	Không cho phép đọc từ bộ nhớ	Data memory được đọc Data_out ← Memory[address]
MemWrite	Không cho phép ghi vào bộ nhớ	Data memory được ghi Memory[address] ← Data_in
MemtoReg	BusW = ALU result	BusW = Data_out từ Data Memory
Beq, Bne	PC ← PC + 4	PC ← Địa chỉ đích của lệnh rẽ nhánh nếu xảy ra
J	PC ← PC + 4	PC ← Địa chỉ đích của lệnh nhảy

Bảng sự thật của Main Control

Op	Reg Dst	Reg Write	Ext Op	ALU Src	Beq	Bne	J	Mem Read	Mem Write	Mem toReg
R-type	1 = Rd	1	x	0=BusB	0	0	0	0	0	0
addi	0 = Rt	1	1=sign	1=Imm	0	0	0	0	0	0
slti	0 = Rt	1	1=sign	1=Imm	0	0	0	0	0	0
andi	0 = Rt	1	0=zero	1=Imm	0	0	0	0	0	0
ori	0 = Rt	1	0=zero	1=Imm	0	0	0	0	0	0
xori	0 = Rt	1	0=zero	1=Imm	0	0	0	0	0	0
lw	0 = Rt	1	1=sign	1=Imm	0	0	0	1	0	1
sw	x	0	1=sign	1=Imm	0	0	0	0	1	x
beq	x	0	x	0=BusB	1	0	0	0	0	x
bne	x	0	x	0=BusB	0	1	0	0	0	x
j	x	0	x	x	0	0	1	0	0	x

- X là giá trị don't care (có thể 0 hoặc 1), dùng để tối ưu

Phương trình luận lý của các tín hiệu ra

$$RegDst = R\text{-}type$$

$$RegWrite = \overline{(sw + beq + bne + j)}$$

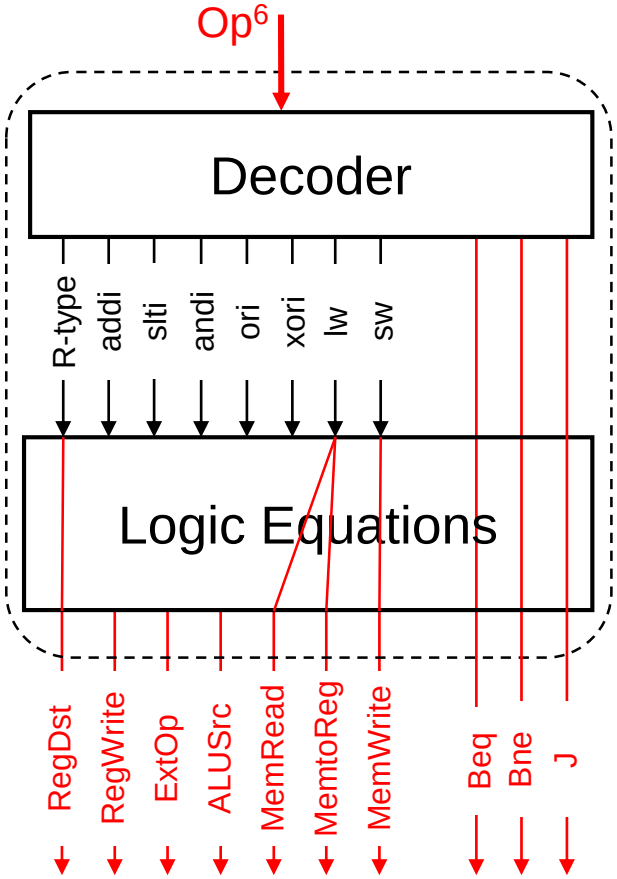
$$ExtOp = \overline{(andi + ori + xori)}$$

$$ALUSrc = \overline{(R\text{-}type + beq + bne)}$$

$$MemRead = lw$$

$$MemtoReg = lw$$

$$MemWrite = sw$$



Bảng sự thật của ALU Control

Input		Output	4-bit
Op^6	$funct^6$	ALUCtrl	Encoding
R-type	add	ADD	0000
R-type	sub	SUB	0010
R-type	and	AND	0100
R-type	or	OR	0101
R-type	xor	XOR	0110
R-type	slt	SLT	1010
addi	x	ADD	0000
slti	x	SLT	1010
andi	x	AND	0100
ori	x	OR	0101
xori	x	XOR	0110
lw	x	ADD	0000
sw	x	ADD	0000
beq	x	SUB	0010
bne	x	SUB	0010
j	x	x	x

Giá trị 4-bit ALUCtrl được mã hóa tùy theo hiện thực của bộ ALU

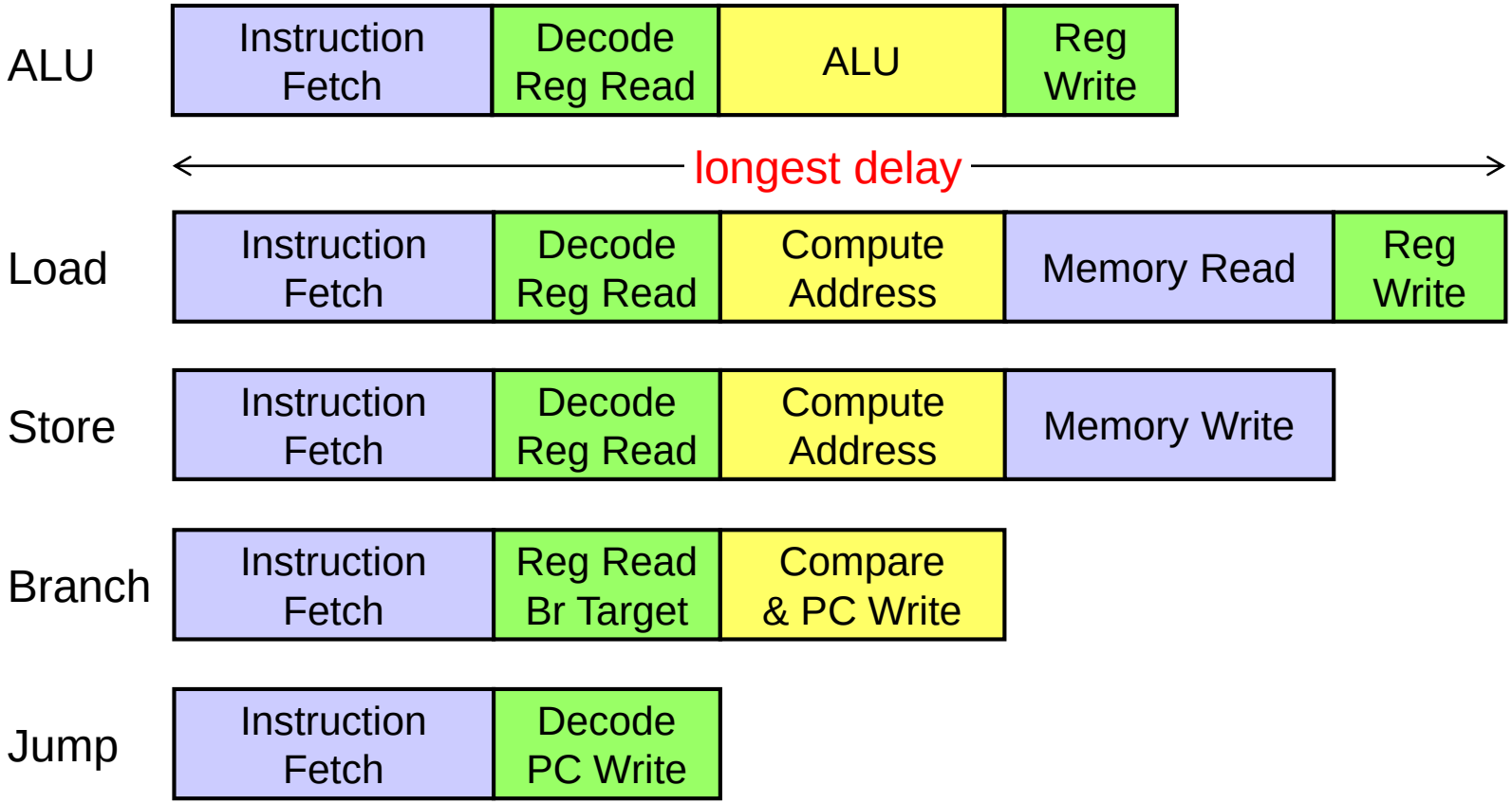
Giá trị cụ thể của ALU control có thể khác. Tùy vào hiện thực cụ thể của bộ ALU

Tiếp theo . . .

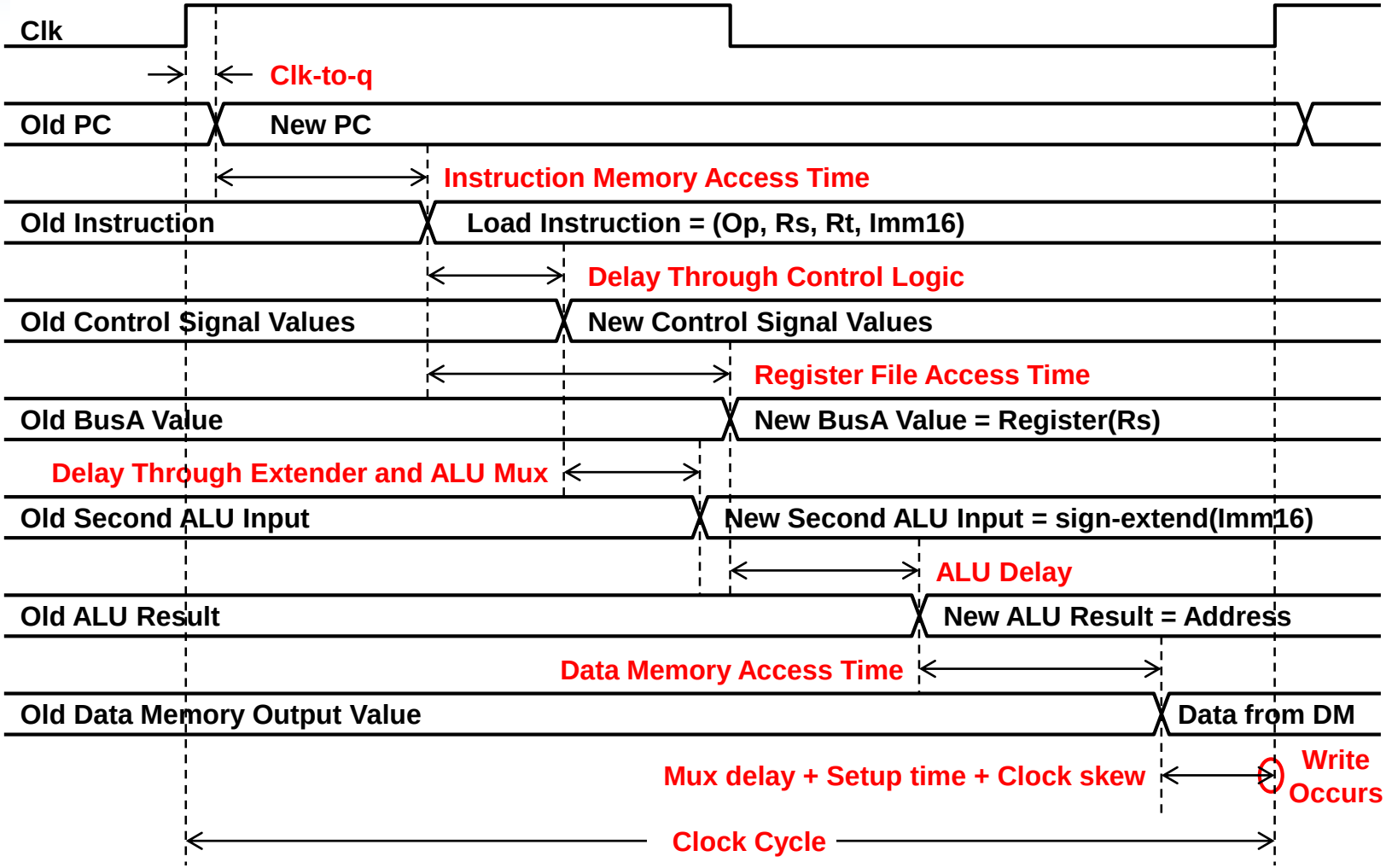
- Thiết kết bộ xử lý: Các bước thực hiện
- Các thành phần của Datapath và cấp xung nhịp
- Xây dựng Datapath đầy đủ
- Điều khiển quá trình thực thi của các lệnh
- Bộ điều khiển chính và bộ điều khiển ALU
- Hạn chế của thiết kế bộ xử lý đơn chu kỳ

Hạn chế của thiết kế bộ xử lý đơn chu kỳ

- Chu kỳ xung nhịp dài
  - Phải thõa mãn lệnh chậm nhất



Thời gian của lệnh Load



Thời gian của lệnh "Load" …

- Thời gian của chu kỳ xung nhịp: đủ dài cho lệnh chậm nhất

  Thời gian trễ PC cập nhật giá trị mới từ lúc cạnh lên của xung nhịp

  + Thời gian truy xuất mã máy từ bộ nhớ lệnh

  + Maximum của(

    Thời gian truy xuất thanh ghi Rs,

    Thời gian trễ của bộ Main Control + extender + ALU mux)

  + Thời gian ALU thực thi phép toán cộng

  + Thời gian truy xuất 4 ô nhớ (1 word) từ bộ nhớ dữ liệu

  + Thời gian trễ của bộ MemtoReg Mux

  + Thời gian để dữ liệu ổn định (setup) + Clock Skew

- Chu kỳ xung nhịp sẽ dài hơn mức cần thiết cho các lệnh khác
  - Do đó bộ xử lý theo thiết kế đơn chu kỳ không dùng trong thực tế

Cải tiến: Hiện thực đa chu kỳ

- Chi quá trình thực thi lệnh thành năm bước
  - Nạp lệnh
  - Giải mã lệnh, đọc thanh ghi, tính địa chỉ đích cho lệnh nhảy/rẽ nhánh
  - Thực thi phép toán , tính địa chỉ truy xuất bộ nhớ, rẽ nhánh xảy ra
  - Truy xuất bộ nhớ hoặc chuẩn bị ghi dữ liệu vào thanh ghi đích (lệnh ALU)
  - Chuẩn bị ghi dữ liệu vào thanh ghi đích (lệnh Load)
- Mỗi bước là một chu kỳ (thời gian một chu kỳ giảm)
  -

Instruction	# cycles	Instruction	# cycles
ALU & Store	4	Branch	3
Load	5	Jump	2

Ví dụ về hiệu suất

- Cho thời gian trễ của các thành phần của datapath:
  - Truy xuất bộ nhớ lệnh và dữ liệu: 200 ps
  - ALU và các bộ cộng khác: 180 ps
  - Đọc giá trị của thanh ghi: 150 ps
  - Thời gian ổn định trước khi ghi giá trị vào thanh ghi: 100ps
  - Bỏ qua thời gian trễ PC, mux, extender và dây nối
- Hiện thực nào nhanh hơn và bao nhiêu lần?
  - Hiện thực đơn chu kỳ
  - Hiện thực đa chu kỳ tối ưu cho từng lớp lệnh
- Giả sử chương trình bao gồm:
  - 40% ALU, 20% Loads, 10% stores, 20% branches, & 10% jumps

Solution

Instruction Class	Instruction Memory	Register Read	ALU Operation	Data Memory	Register Write	Total
ALU	200	150	180		100	630 ps
Load	200	150	180	200	100	830 ps
Store	200	150	180	200		730 ps
Branch	200	150	180			530 ps
Jump	200	150				350 ps

- Cho thiết kế đơn chu kỳ:
  - Chu kỳ xung nhịp = 830 ps thõa mãn lệnh dài nhất (lệnh load)
- Cho thiết kế đa chu kỳ:
  - Chu kỳ xung nhịp = max(200, 150, 180) = 200 ps (thõa mãn bước dài nhất)
  - CPI trung bình = 0.4×4 + 0.2×5 + 0.1×4+ 0.2×3 + 0.1×2 = 3.8
- Speedup = 830 ps / (3.8 × 200 ps) = 830 / 760 = 1.1

Tổng kết

- 5 bước trong quá trình thiết kế bộ xử lý = datapath+control
  - Phân tích tập lệnh => tìm ra các thành phần cần thiết cho datapath
  - Hiện thực các thành phần của datapath & thiết lập cách cấp xung nhịp
  - Kết nối các thành phần của datapath để thõa mãn tập lệnh
  - Phân tích quá trình kết nối ở bước 4 để xác định các tín hiệu điều khiển cần thiết
  - Hiện thực bộ điều khiển
- Bộ xử lý MIPS có thiết kế đơn giản
  - Các lệnh có cùng độ rộng 32 bit
  - Thanh ghi nguồn ở vị trí cố định
  - Hắng số imm16 ở vị trí cố định và cùng độ rộng 16 bit
  - Toán hạn của ALU là thanh ghi/hằng số
- Bộ xử lý đơn chu kỳ => CPI=1, nhưng chu kỳ sẽ dài