TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN ĐIỆN TỬ - VIỄN THÔNG

BÁO CÁO KIẾN TRÚC MÁY TÍNH

ĐỀ TÀI: THIẾT KẾ VÀ MÔ PHỎNG BỘ XỬ LÍ ĐƯỜNG ỐNG CHO RISC-V

Tên sinh viên: Nguyễn Văn Lưu

MSSV: 20192993

Mã lớp: 137336

Giảng viên hướng dẫn: TS. Tạ Thị Kim Huệ

Hà Nội – 3/2023

Mục lục

Lời nói đầu

Vận dụng kiến thức đã học ở môn Kiến trúc máy tính, em xin trình bày về thiết bộ xử lí đường ống dựa trên lí thuyết đã học được. Trong báo cáo là sự hiểu biết của em và tham khảo trong sách "***Computer Organization and Design RISC-V Edition***" , Phần lập trình và mô phỏng em có đọc hiểu code và tham khảo code rồi tự mô phỏng, do thời gian không cho phép vì lịch thi cuối kì nhiều môn nên bài báo cáo còn nhiều sai sót và chưa hoàn thiện nên em mong cô bỏ qua.

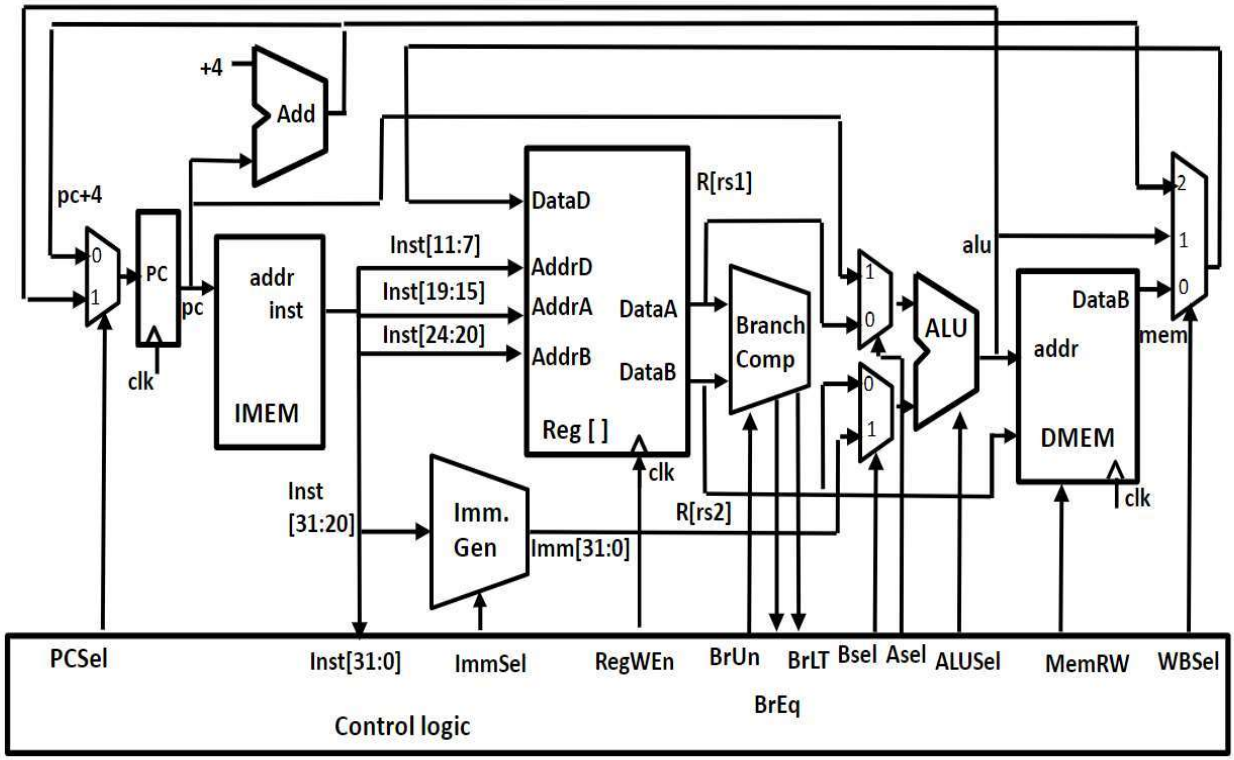
Em chân thành cảm ơn cô rất nhiều.

I, Tìm hiểu kiến trúc bộ xử lí RISC-V

1, Bộ xử lí đơn xung nhịp

Bộ xử lí đơn xung nhịp chỉ thực hiện một lệnh trong một chu kỳ, thời gian chu kỳ bằng thời gian thực hiện lệnh dài nhất

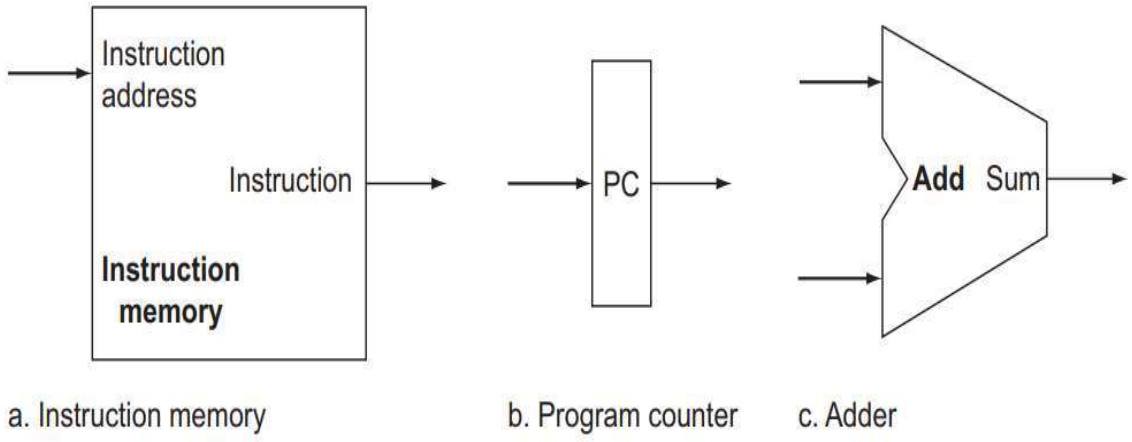
Quá trình xử lí một lệnh trải qua 5 giai đoạn: IF, ID, EX, MEM, WB, trong khi 1 khối chức năng đang làm việc thì các khối còn lại sẽ ở trạng thái không làm việc.



Hình : Đường dữ liệu tổng hợp của bộ xử lí đơn xung nhịp

Các khối cần có trong bộ xử lí đơn xung nhịp

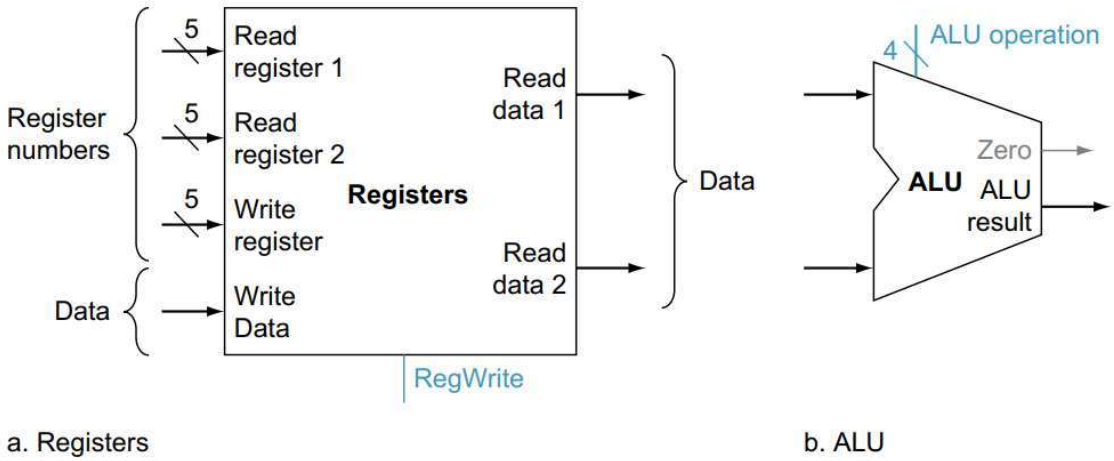
- **Khối bộ nhớ lệnh**



Hình khối bộ nhớ lệnh, PC, bộ ADD

Cần có hai phần tử trạng thái để lưu trữ và truy cập các lệnh, và một bộ cộng là cần thiết để tính toán địa chỉ lệnh tiếp theo. Các phần tử trạng thái là bộ nhớ lệnh và bộ đếm chương trình. Bộ nhớ lệnh chỉ cần cung cấp quyền truy cập đọc bởi vì datapath không ghi vào bộ nhớ này.

Khối Register file:



Hình tệp thanh ghi

Tệp thanh ghi chứa tất cả các thanh ghi và có hai cổng đọc và một cổng ghi.

Tệp luôn xuất dữ liệu của các thanh ghi tương ứng với các đầu vào thanh ghi

trên các đầu ra.

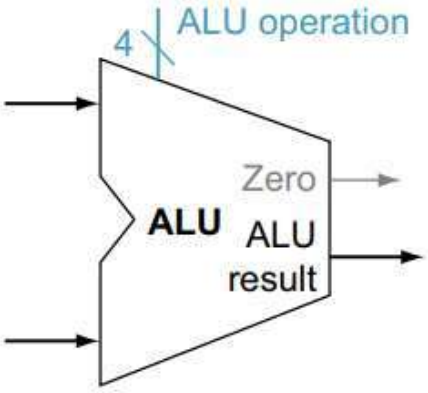
Tệp thanh ghi hoạt động phụ thuộc vào kích hoạt cạnh, do đó tất cả các đầu vào ghi (nghĩa là giá trị để được ghi, số thanh ghi và tín hiệu điều khiển ghi) phải hợp lệ ở cạnh đồng hồ. Muốn ghi đối với tệp đăng ký được kích hoạt cạnh lên 1, thiết kế để có thể đọc và ghi hợp pháp cùng một thanh ghi trong 1 chu kỳ đồng hồ: việc đọc sẽ nhận được giá trị được ghi trong chu kỳ đồng hồ trước đó, trong khi giá trị được ghi sẽ có sẵn cho một lần đọc trong một chu kỳ đồng hồ tiếp theo. Các đầu vào mang số thanh ghi đến tệp thanh ghi đều là 5 bit, trong khi các dòng mang giá trị dữ liệu 32 bit.

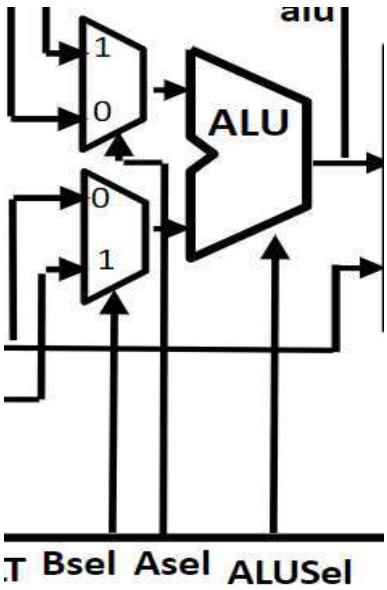
Khối ALU

Khối ALU dung để tính toán các phép toán logic và số học như AND, OR, …,

tính toán địa chỉ nhớ, tính toán địa chỉ lệnh cho PC.

Đầu vào khối ALU có thể là đến từ các thanh ghi, phần immediate bằng việc sử dụng các bộ MUX kết hợp với các tín hiệu điều khiển B-sel, A-sel để chọn toán hạng làm đầu vào cho khối ALU.

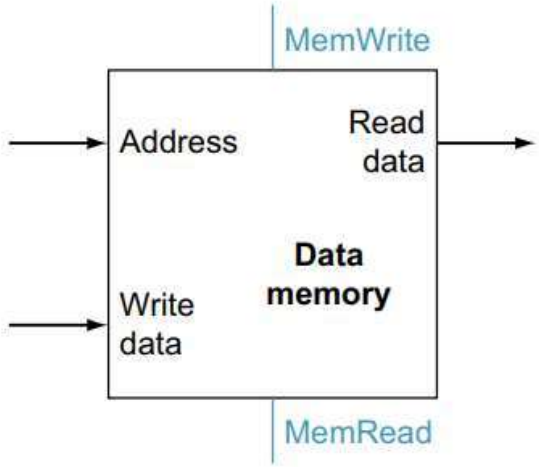




Hình khối ALU

Khối bộ nhớ dữ liệu

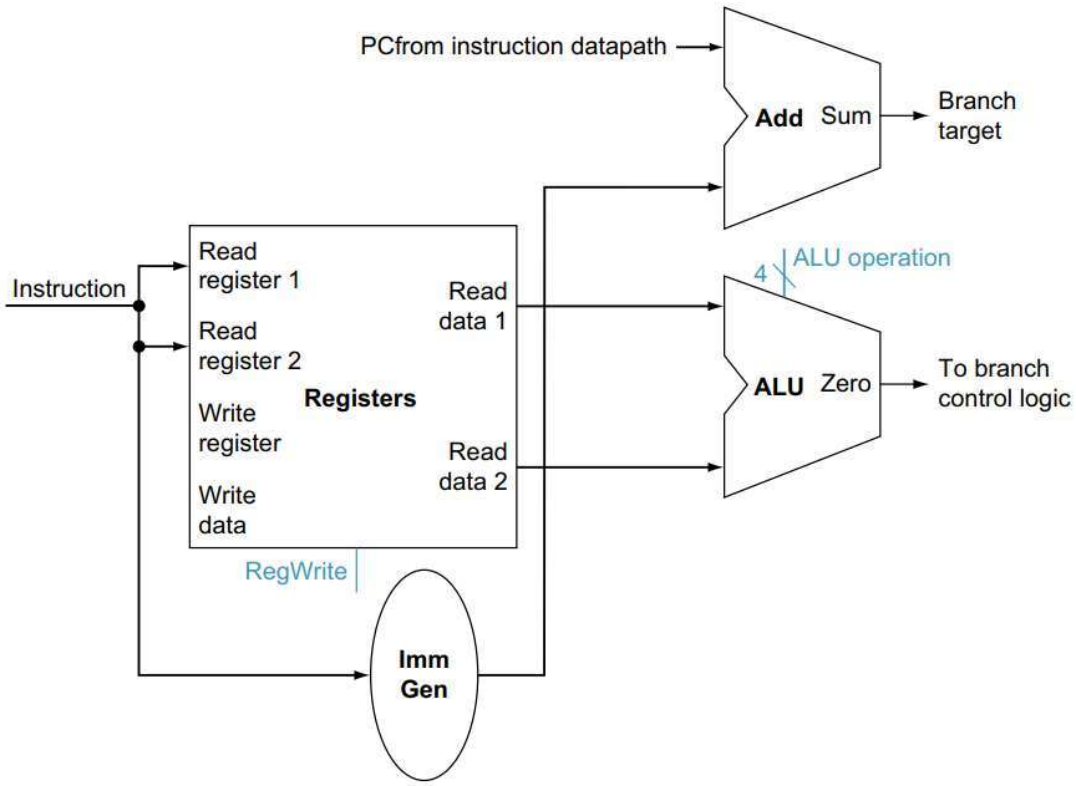
.Đơn vị bộ nhớ là một phần tử trạng thái có đầu vào cho địa chỉ và dữ liệu ghi và một đầu ra duy nhất cho kết quả đọc. Có các điều khiển đọc và ghi riêng biệt, mặc dù chỉ một trong số này có thể được xác nhận trên bất kỳ xung đồng hồ nhất định. Đơn vị bộ nhớ cần một tín hiệu đọc, vì không giống như tệp thanh ghi, việc đọc giá trị của một địa chỉ không hợp lệ có thể gây ra vấn đề, như chúng ta sẽ thấy trong.



Hình khối bộ nhớ dữ liệu

Khối Immediate Generator

Đơn vị phát điện tức thời (ImmGen) có lệnh 32 bit làm đầu vào chọn trường 12 bit để tải, lưu trữ và phân nhánh nếu bằng nhau, đó là dấu mở rộng thành kết quả 32 bit xuất hiện trên đầu ra .

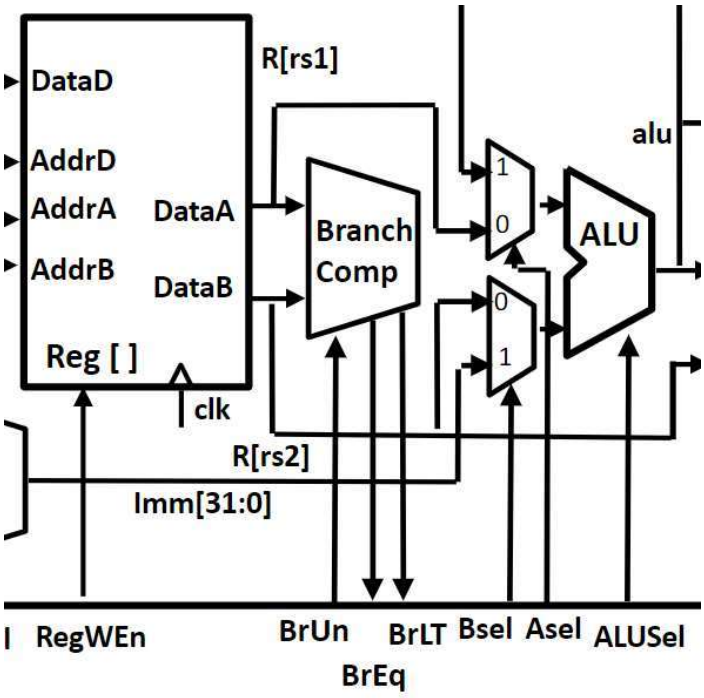


Hình khối Immediate Gen

Khối rẽ nhánh

Khối nhánh được thiết kế chịu trách nhiệm tính toán giá trị PC tiếp theo nếu lệnh nhánh được tìm nạp trong giai đoạn IF. Nếu lệnh JALR hoặc lệnh JAL hoặc BRANCH được thực hiện, đầu ra PcSel sẽ được đặt thành 1.

Đối với giá trị PC mới (đầu ra BrPc), nó sẽ được đặt thành PC+Imm nếu lệnh BRANCH/JAL được sử dụng hoặc AluResult nếu JALR được sử dụng.

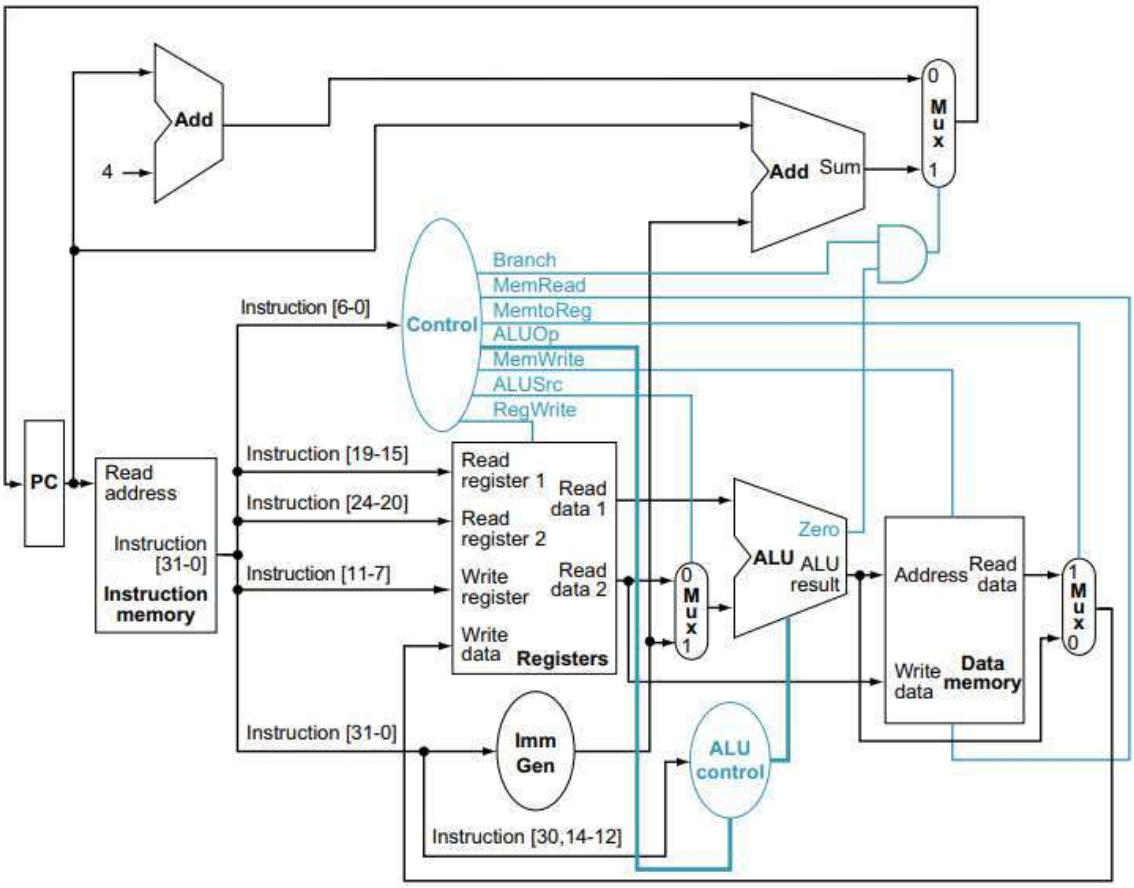


Hình khối kiểm tra rẽ nhánh

Khối tín hiệu điều khiển

Đầu vào của khối điều khiển là trường opcode 7 bit từ lệnh.

Đầu ra của khối điều khiển bao gồm hai tín hiệu 1 bit được sử dụng để điều khiển bộ ghép kênh (ALUSrc và MemtoReg), ba tín hiệu cho kiểm soát đọc và ghi trong tệp thanh ghi và bộ nhớ dữ liệu (RegWrite,MemRead và MemWrite), tín hiệu 1 bit được sử dụng để xác địnhcó thể phân nhánh (Branch) hay không và tín hiệu điều khiển 2 bit cho ALU (ALUOp). Một cổng AND được sử dụng để kết hợp điều khiển rẽ nhánh tín hiệu và đầu ra Zero từ ALU; đầu ra cổng AND kiểm soát việc lựa chọn PC tiếp theo chứ không phải là một đến trực tiếp từ các đơn vị điều khiển.



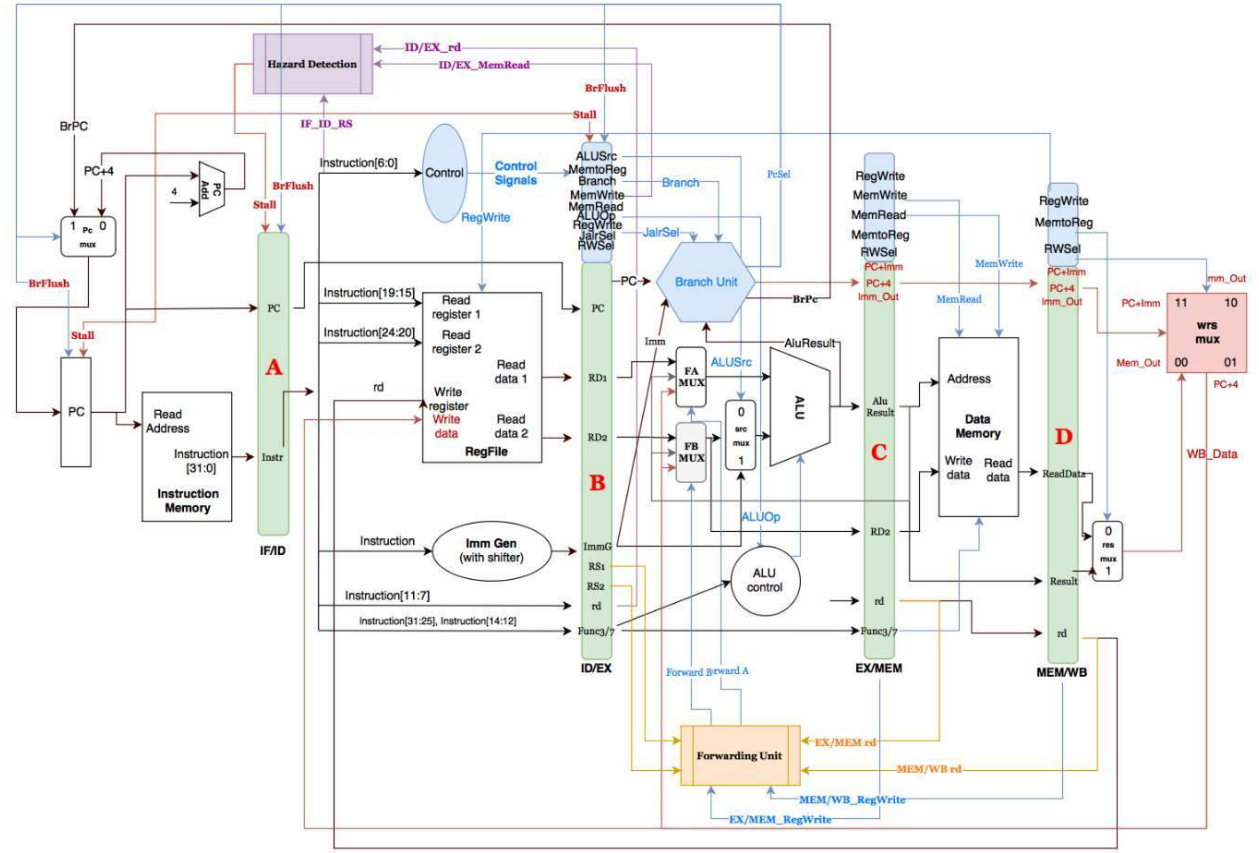
Hình khối tín hiệu điều khiển

Signal name	Effect when deasserted	Effect when asserted
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, 12 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

Bảng các tín hiệu điều khiển

2. Bộ xử lí đường ống

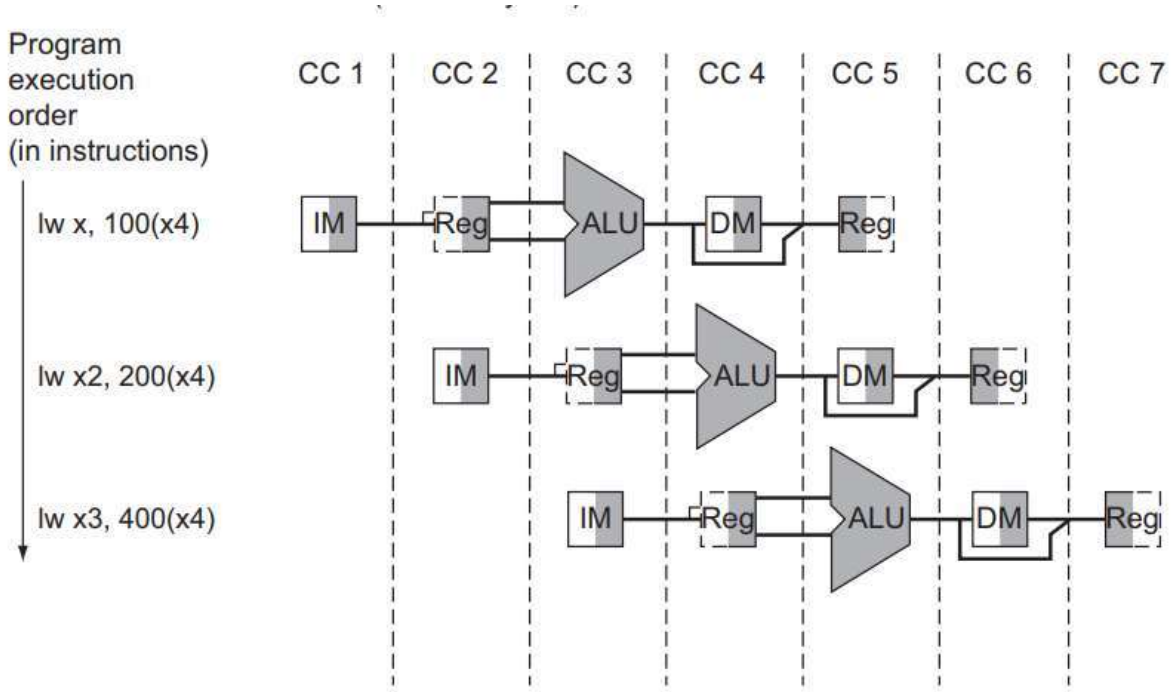
2.1, Các khối trong bộ xử lí đường ống.



Hình datapath của bộ xử lí đường ống

Khác với bộ xử lí đơn xung nhịp, bộ xử lí đường ống có thể xử lí nhiều lệnh trong một chu kì đồng hồ nhằm làm cho CPU bận nhất có thể.

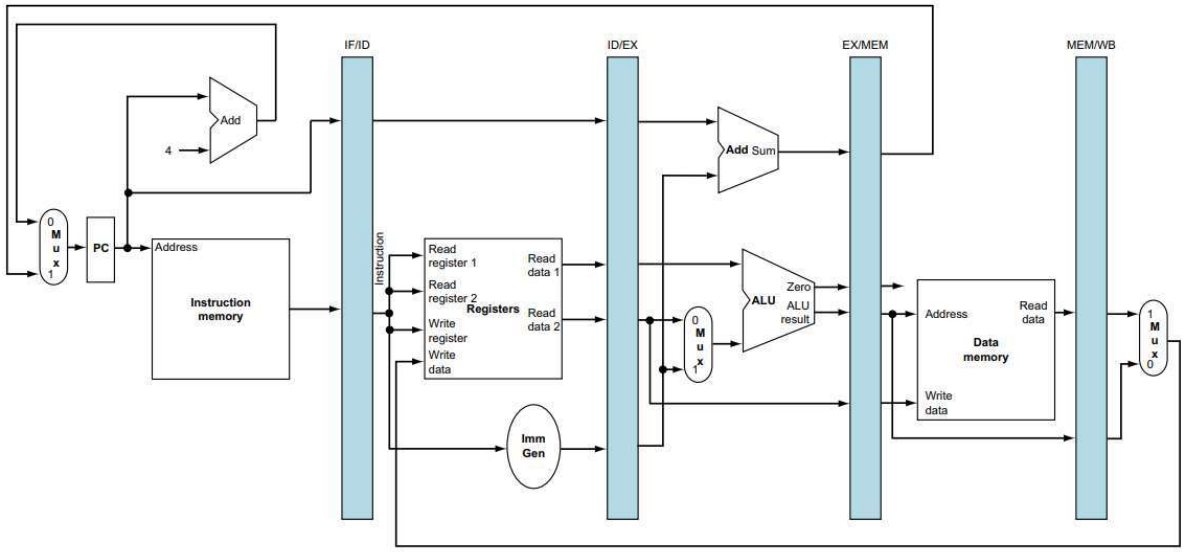
Thời gian chu kì đồng hồ được chọn bằng thời gian lớn nhất mà CPU mất để thực hiện ở từng giai đoạn: IF, ID, EX, MEM, WB.



Hình minh họa cách thức xử lí đường ống.

- Ngoài những khối có trong bộ xử lí đơn xung nhịp thì bộ xử lí đường ống cần có thêm các bổ sung về phần cứng để xử lí những vấn đề của bộ xử lí đường ống.
-

1,Thanh ghi đệm



Hình minh họa các thanh ghi đệm

Vì trong một chu kì lệnh CPU có thể thực thi nhiều hơn một lệnh nên giữa các khối chức năng cần có thêm thanh ghi để chứa dữ liệu tạm thời.

Thanh ghi A: Thanh ghi giữa 2 bước IF/ID

Thanh ghi B: Thanh ghi giữa 2 bước ID/EX

Thanh ghi C: Thanh ghi giữa 2 bước EX/MEM

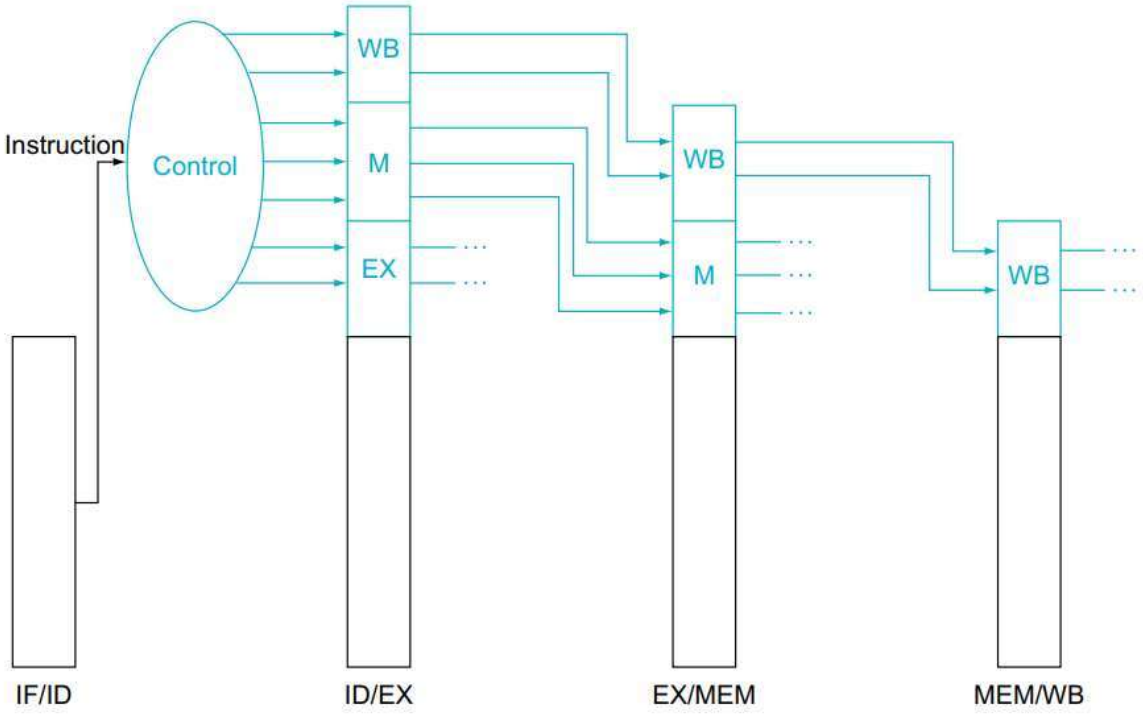
Thanh ghi D: Thanh ghi giữa 2 bước MEM/WB

Các thanh ghi đường ống, có màu, phân tách từng giai đoạn đường ống.Chúng được đặt tên bởi các giai đoạn mà chúng tách biệt.

ví dụ: cái đầu tiên được gọi là IF/ID vì nó phân tách lệnh tìm nạp và lệnh giai đoạn giải mã Các thanh ghi phải đủ rộng để lưu trữ tất cả dữ liệu tương ứng với các dòng đi qua .

Ví dụ: IF/ID thanh ghi phải rộng 96 bit, vì phải chứa cả lệnh 32 bit được lấy từ bộ nhớ và địa chỉ PC 64 bit tiếp theo, nhưng hiện tại ba thanh ghi đường ống khác chứa 256, 193 và 128 bit.

Tất cả bốn thanh ghi bộ đệm sẽ được cập nhật ở cạnh dương của tín hiệu đồng hồ. Ngoài ra, ngay từ đầu, đề cập đến cạnh dương của tín hiệu reset, các phần tử chính trong cả 4 thanh ghi sẽ được đặt thành 0 khi khởi tạo.



Hình tín hiệu điều khiển trong bộ xử lí đường ống.

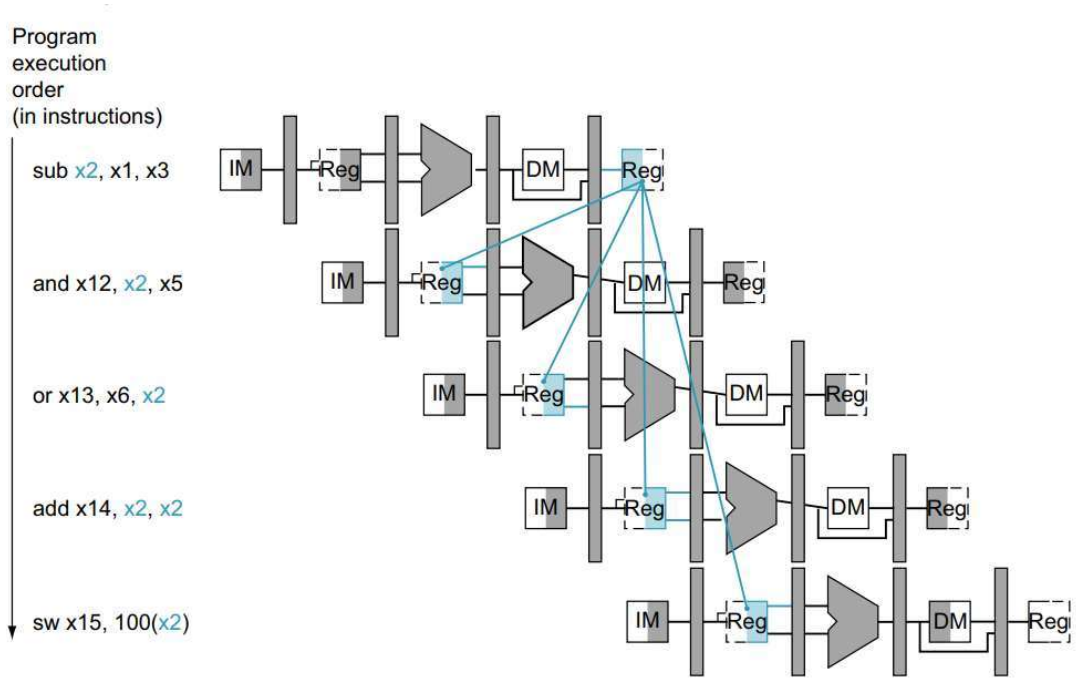
Như trường hợp bộ xử lí một chu kỳ, giả sử rằng PC là được ghi trên mỗi chu kỳ đồng hồ, do đó không có tín hiệu ghi riêng cho PC. Bằng cùng một đối số, không có tín hiệu ghi riêng cho các thanh ghi đường ống (IF/ ID, ID/EX, EX/MEM và MEM/WB), vì các thanh ghi đường ống cũng được ghi trong mỗi chu kỳ đồng hồ Đường dẫn dữ liệu này mượn điều khiển logic cho nguồn PC, thanh ghi đích và điều khiển ALU. Lưu ý rằng bây giờ chúng ta cần các trường chức năng của lệnh trong giai đoạn EX làm đầu vào cho điều khiển ALU, vì vậy các bit này cũng phải được đưa vào thanh ghi đường dẫn ID/EX.

=>Bộ xử lí đường ống cần có tín hiệu điều khiển đường ống .

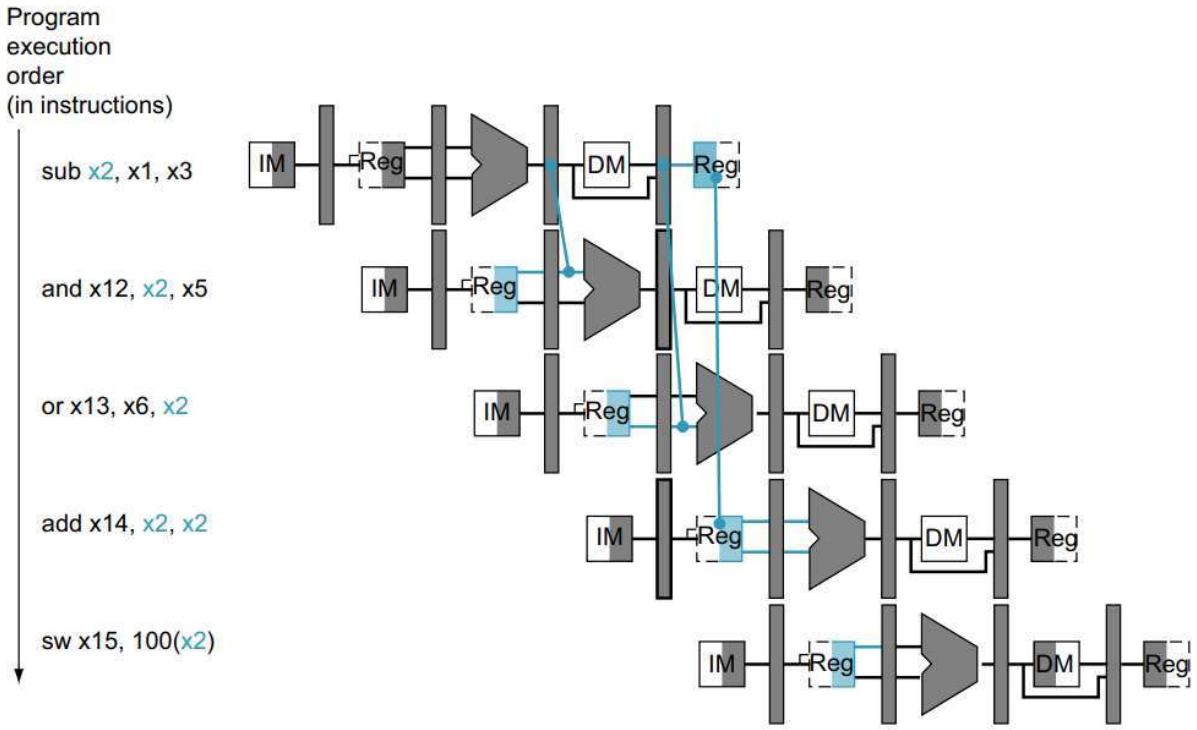
2, Xung đột dữ liệu trong bộ xử lí đường ống.

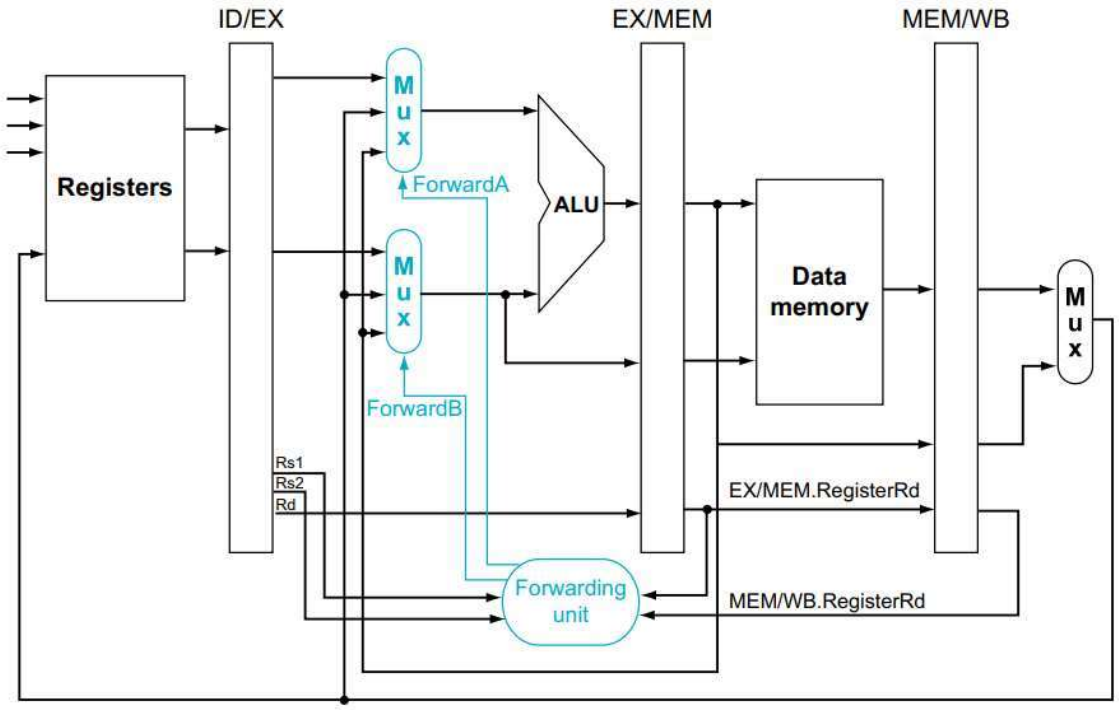
- Xung đột dữ liệu: Nếu một chương trình thực thi một lệnh có giá trị trả về cho thanh ghi đích, thì sau5 chu kì giá trị trong thanh ghi đó sẽ được cập nhật. Nếu thanh ghi đó được dùng làm thanh ghi đọc dữ liệu cho các lệnh

sau thì trong giai đoạn giải mã lệnh giá trị thanh ghi đó chưa được cập nhật. Gây nên xung dột về dữ liệu gây ra kết quả bị sai.

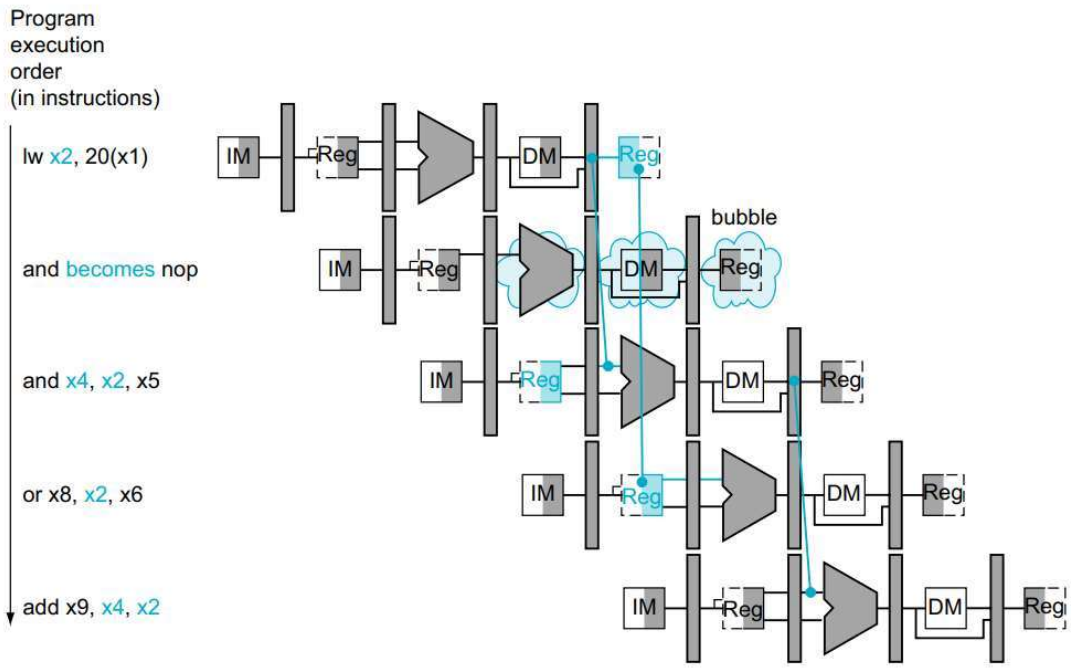


- **Cách khắc phục**: Forwarding- chuyển tiếp dữ liệu ngay sau khi tính toán xong ở khối ALU vào chu kì ALU của lệnh sau.
- Ưu điểm của forwarding là khắc phục được hiệu quả xung đột dữ liệu nhưng lại phức tạp về phần cứng gây ra tốn chi phí thi công.





Với lệnh load thì không thể chuyển tiếm giữa hai chu kì kề nhau được. nên giải pháp thứ 2 chèn them chu kì đợi (stalling).



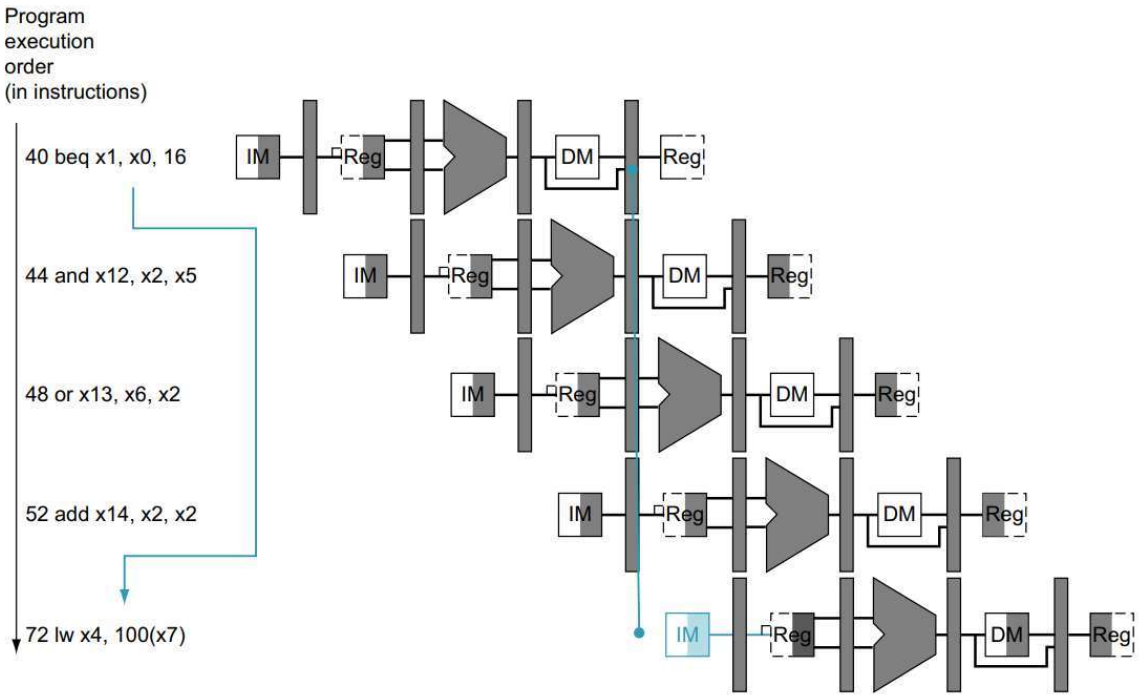
- Ưu điểm của stalling là dễ thực hiện nhưng lại làm giảm hiệu năng của CPU vì CPU không hoạt động ở các chu kì đợi.
- Giải pháp thứ 3 cho xung đột dữ liệu là sắp xếp lại các lệnh khi lập trình sao cho không bị xung đột thanh ghi .

Xung đột điều khiển (Khối phát hiện xung đột)

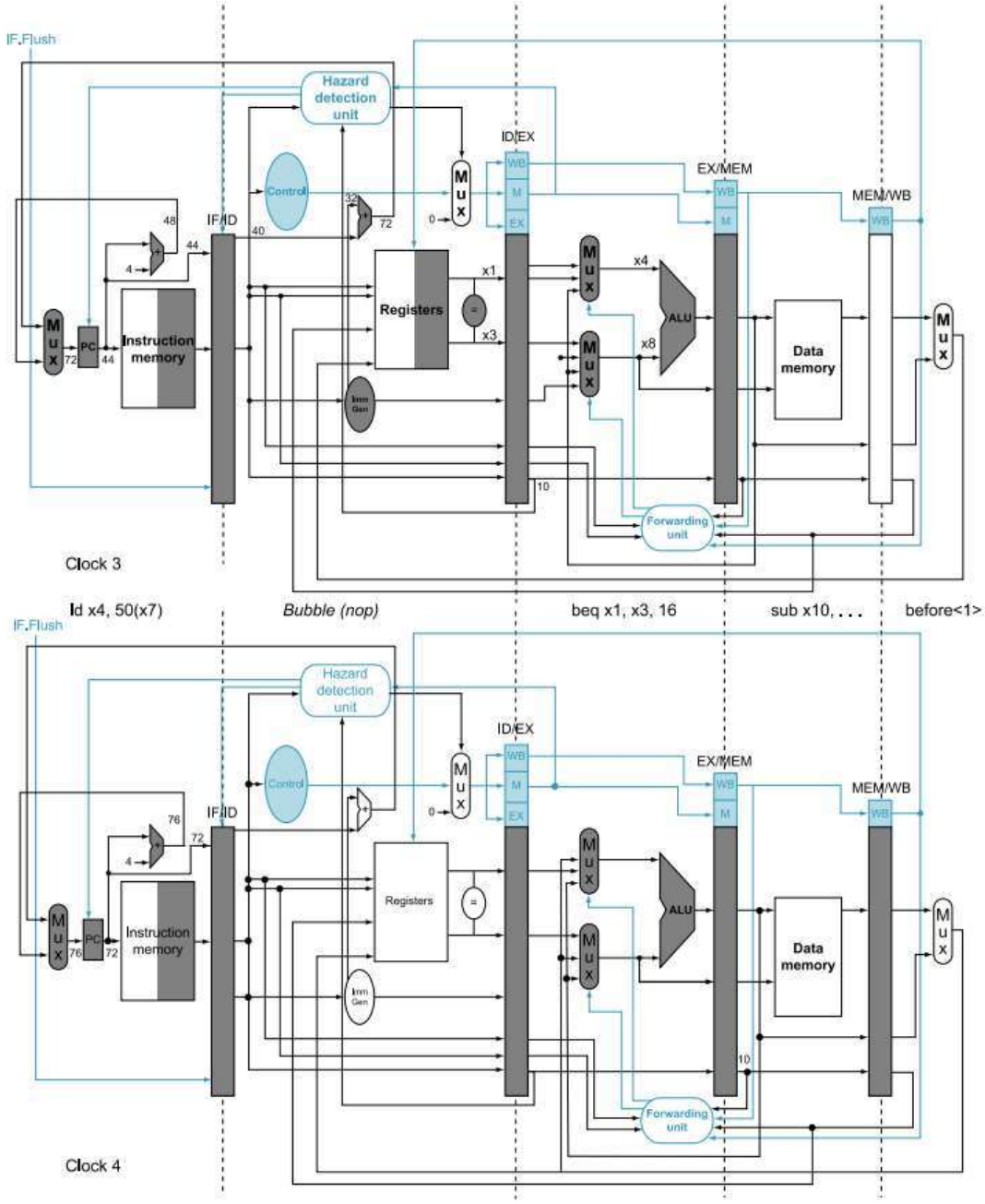
Khối phát hiện xung đột được thêm vào đường dẫn dữ liệu đường ống. Nếu phát hiện nguy cơ phụ thuộc dữ liệu, tín hiệu ngừng hoạt động sẽ được tạo và được gửi đến PC, thanh ghi A và thanh ghi B. PC kiểm tra tín hiệu stalling mỗi cạnh tích cực của đồng hồ. Bất cứ khi nào tín hiệu này có giá trị 1, đầu ra PC sẽ không bị thay đổi.

Xét lệnh rẽ nhánh trong RISC-V, nếu điều kiện rẽ nhánh đúng thì lệnh tiếp theo mà CPU sẽ thực hiện sẽ là PC=PC+ immediate(16), trong khi đó các chu kì tiếp theo đã chạy các lệnh khác, sau chu kì thứ 5 của lệnh rẽ nhánh thì giá trị PC mới được cập nhật, trong trường hợp có rẽ nhánh thì các chu kì tìm nạp lệnh của các lệnh tiếp theo sẽ bị loại bỏ.

```
36  sub  x10, x4, x8
40  beq  x1,  x3, 16  /
44  and  x12, x2, x5
48  or   x13, x2, x6
```



- Giải pháp: Dự đoán rẽ nhánh, dự đoán đơn giản nhất là giả sử ban đầu lệnh rẽ nhánh không xảy ra. Nếu lệnh rẽ nhánh không xảy ra thì giữ nhuyên các lệnh tiếp theo, còn không xóa các lệnh tiếp theo ra hỏi đường ống bằng cách chèn vào các lệnh NOP.



Hình xung đột điều khiển

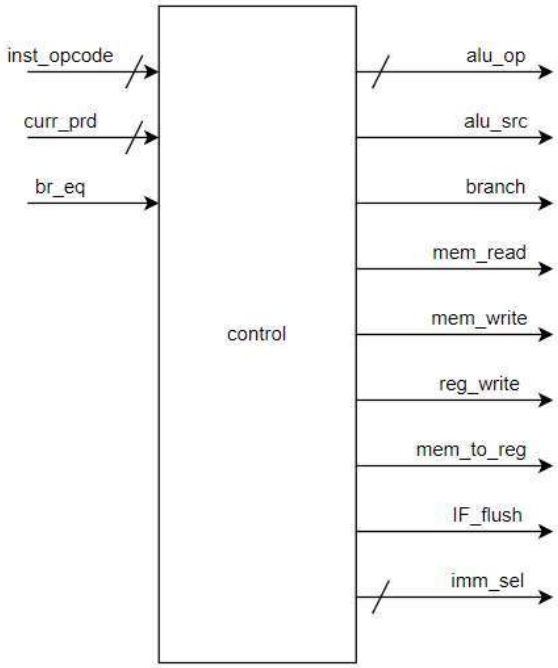
- Khối phát hiện xung được thêm vào đường dẫn dữ liệu đường ống. Nếu phát hiện nguy cơ phụ thuộc dữ liệu, tín hiệu ngừng hoạt động sẽ được tạo và được gửi đến PC, thanh ghi A và thanh ghi B. PC kiểm tra tín hiệu stalling mỗi cạnh tích cực của đồng hồ. Bất cứ khi nào tín hiệu này có giá trị 1, đầu ra PC sẽ không bị thay đổi.

II, Tiến hành thiết kế

Hệ thống có chức năng thực hiện các lệnh trong Risc-V, nhưng không phải tất cả các lệnh đều đi qua hết các khối. Hệ thống bao gồm:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Execute (EX) – ALU (Arithmetic-Logic Unit)
- Memory Access
- Write Back to Register (WB)
- Các thanh ghi tương ứng: IF/ID, ID/EX, EX/MEM, MEM/WB
- Hazard Detection Unit
- Forwarding Unit
- Control
- ALU Control

1, Khối Controller



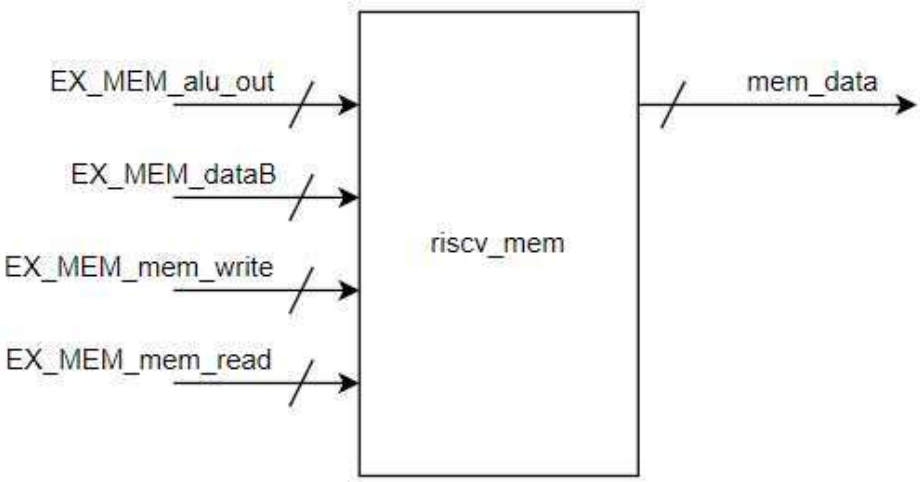
Name	Width	Input/Output
inst_opcode		Input
curr_prd		Input
br_eq		Input
alu_op		Output
alu_src		Output
branch		Output
mem_read		Output
mem_write		Output
reg_write		Output
mem_to_reg		Output
IF_flush		Output
imm_sel		Output

Bảng tên phần tử vào ra

Khai báo :

```
module Controller(

    //Input
    input logic [6:0] Opcode,

    //Outputs
    output logic ALUSrc,
    output logic MemtoReg,
    output logic RegWrite,
    output logic MemWrite,
    output logic [1:0] ALUOp,
    output logic Branch,
    output logic JalrSel,
    output logic [1:0] RWSel
);
```

2, Khối Memory access



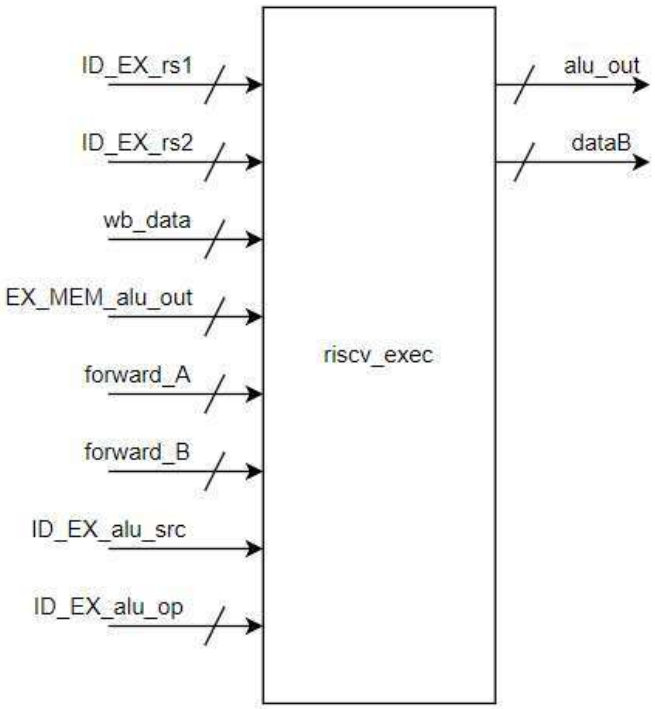
Name	Input/Output
EX_MEM_alu_out	Input
EX_MEM_dataB	Input
EX_MEM_mem_write	Input
EX_MEM_mem_read	Input
mem_data	Output

Bảng tên phần tử vào ra

Khai báo:

```
module datamemory#(
    parameter DM_ADDRESS = 9 ,
    parameter DATA_W = 32
    )(
    input logic clk,
        input logic MemRead ,
    input logic MemWrite ,
    input logic [ DM_ADDRESS -1:0] a ,
    input logic [ DATA_W -1:0] wd ,
    input logic [2:0] Funct3,
    output logic [ DATA_W -1:0] rd
    );
```

3, Khối EX.

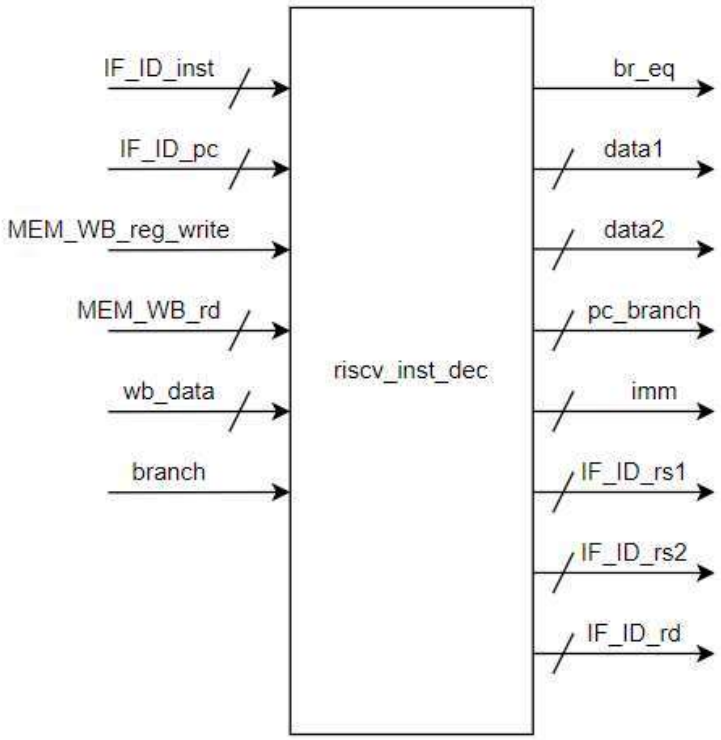


Name	Width	Input/Output
ID_EX_rs1	32	Input
ID_EX_rs2	32	Input
wb_data		Input
EX_MEM_alu_out		Input
forwardA		Input
forwardB		Input
ID_EX_alu_src		Input
ID_EX_alu_op		Input
alu_out		Output
dataB		Output

Bảng tên phần tử vào ra

```
module alu#(
        parameter DATA_WIDTH = 32,
        parameter OPCODE_LENGTH = 4
        )
        (
        input logic [DATA_WIDTH-1:0]    SrcA,
        input logic [DATA_WIDTH-1:0]    SrcB,

        input logic [OPCODE_LENGTH-1:0]    Operation,
        output logic[DATA_WIDTH-1:0] ALUResult
        );
```

4,Khối ID-Instruction decode.

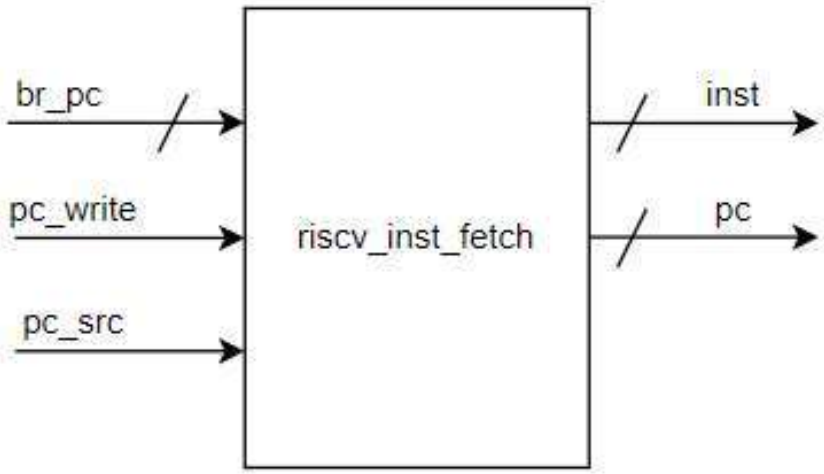


Name	Width	Input/ Output	Description
IF_ID_inst	32	Input	Instruction 32 bit từ khối Instruction Fetch
IF_ID_pc		Input	Giá trị pc từ thanh ghi IF/ID
MEM_WB_reg_write	1	Input	Tín hiệu Write Enable cho thanh ghi đích

MEM_WB_rd	1	Input	
wb_data		Input	
branch		Input	
br_eq		Output	
data1, data2		Output	
pc_branch		Output	
imm		Output	
IF_ID_rs1, IF_ID_rs2		Output	
IF_ID_rd		Output	

```
module instruction_decode (
    input                 clk              ,
    input                 reset_n          ,
    input        [31:0] IF_ID_pc           ,
    input        [31:0] IF_ID_inst         ,
    input        [ 4:0] IF_ID_rs1          ,
    input        [ 4:0] IF_ID_rs2          ,
    input                MEM_WB_reg_write  ,
    input        [ 4:0] MEM_WB_rd          ,
    input        [31:0] wb_data            ,
    input        [ 2:0] imm_sel            ,
    input                mem_to_reg        ,
    input                reg_write         ,
    input                mem_write         ,
    input                mem_read          ,
    input                alu_src           ,
    input        [ 1:0] alu_op             ,
    input                ctrl_sel          ,
    input        [ 1:0] forward_comp1      ,
    input        [ 1:0] forward_comp2      ,
    input        [31:0] alu_out            ,
    input        [31:0] mem_data           ,
    input        [31:0] EX_MEM_alu_out     ,
    input                EX_MEM_mem_to_reg ,
    output logic [31:0] pc_branch          ,
    output logic        br_eq              ,
    output logic        ID_EX_mem_to_reg   ,
    output logic        ID_EX_reg_write    ,
    output logic        ID_EX_mem_write    ,
    output logic        ID_EX_mem_read     ,
    output logic        ID_EX_alu_src      ,
    output logic [ 1:0] ID_EX_alu_op       ,
    output logic [31:0] ID_EX_data1        ,
    output logic [31:0] ID_EX_data2        ,
    output logic [ 4:0] ID_EX_rs1          ,
    output logic [ 4:0] ID_EX_rs2          ,
    output logic [ 4:0] ID_EX_rd           ,
    output logic [31:0] ID_EX_imm_gen      ,
```

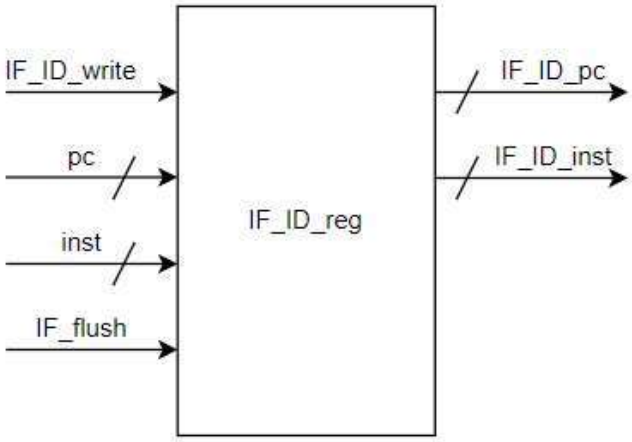
5,Khối IF-Instruction fetch.



Name	Width	Input/Output	Description
br_pc	32	Input	Kết quả của khối adder tính toán địa chỉ branch
pc_write	1	Input	Đầu ra khối Hazard detection unit, xác định giá trị PC thực hoặc tất cả bằng 0
pc_src	1	Input	Tín hiệu lựa chọn đầu vào PC
inst	32	Output	Instruction của lệnh cần thực hiện
pc	32	Output	Giá trị PC làm đầu vào cho khối IMEM

Bảng tên phần tử vào ra

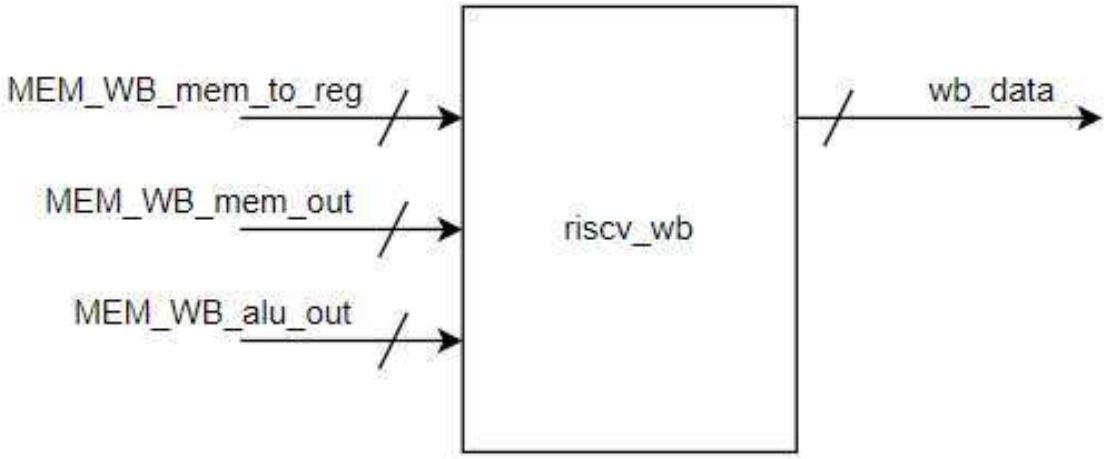
Thanh ghi IF/ID được thêm vào sau khối IF để lưu giá trị đầu ra khối IF phục vụ cho pipeline.



Name	Width	Input/Output	Description
IF_ID_write	1	Input	Tín hiệu điều khiển từ Hazard detection unit
pc	1	Input	Giá trị pc
inst	32	Input	Instruction của lệnh cần thực hiện, từ khối IF
IF_flush		Input	
IF_ID_pc	1	Output	Giá trị PC làm đầu vào cho khối IMEM

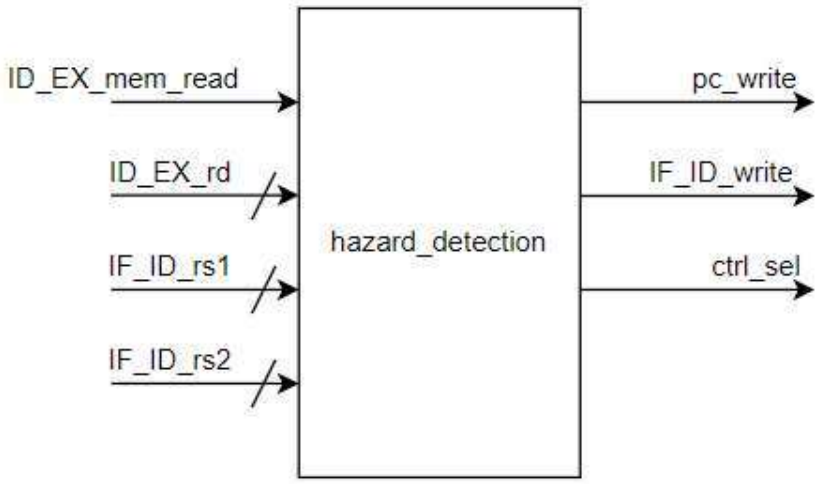
```
//----------------------------------------
module instruction_fetch (
    input                   clk         , //
    input                   reset_n     , //
    input          [31:0]   pc_branch   , //
    input                   pc_write    ,
    input                   pc_src      ,
    input                   IF_ID_write ,
    input                   IF_flush    ,
    output logic   [31:0]   IF_ID_pc    ,
    output logic   [31:0]   IF_ID_inst
);
```

6, Khối Write back




```
//----------------------------------------------------
module register_write (
    input                MEM_WB_mem_to_reg ,
    input         [31:0] MEM_WB_mem_data   ,
    input         [31:0] MEM_WB_alu_out    ,
    output logic  [31:0] wb_data
);

//
```

7, Khối phát hiện xung đột



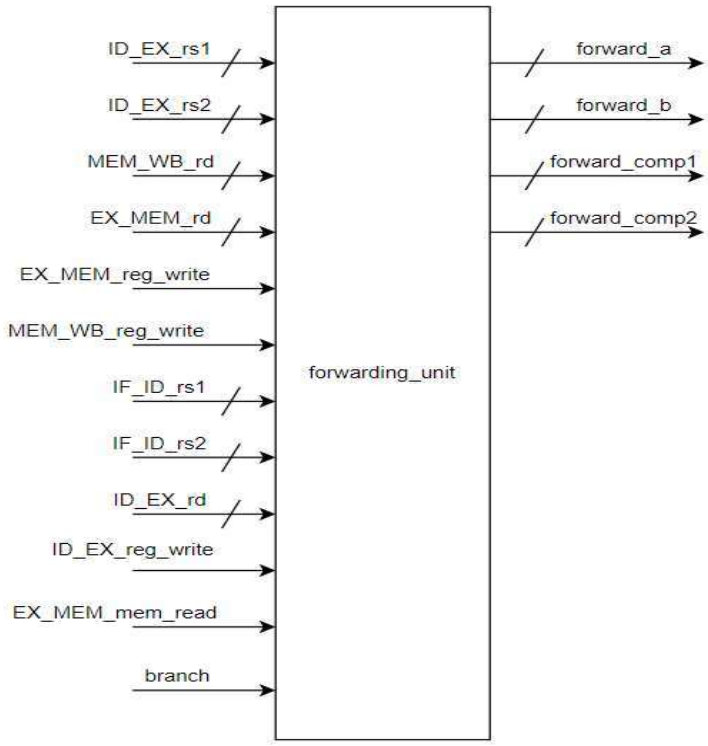
Name	Width	Input/Output	Description
ID_EX_mem_read		Input	
ID_EX_rd		Input	
IF_ID_rs1		Input	
IF_ID_rs2		Input	
pc_write		Output	
IF_ID_write		Output	
ctrl_sel		Output	

Bảng tên phần tử vào ra

```
module hazard_detection_unit (
    input                ID_EX_mem_read,
    input          [4:0] ID_EX_rd      ,
    input          [4:0] IF_ID_rs1     ,
    input          [4:0] IF_ID_rs2     ,
    output  logic        pc_write      ,
    output  logic        IF_ID_write   ,
    output  logic        ctrl_sel
);
```

8, Khối Forwarding (chuyển tiếp)

Khối forwarding giải quyết vấn đề xung dột dữ liệu, làm giảm chu kỳ stall của CPU làm tang hiệu năng CPU.

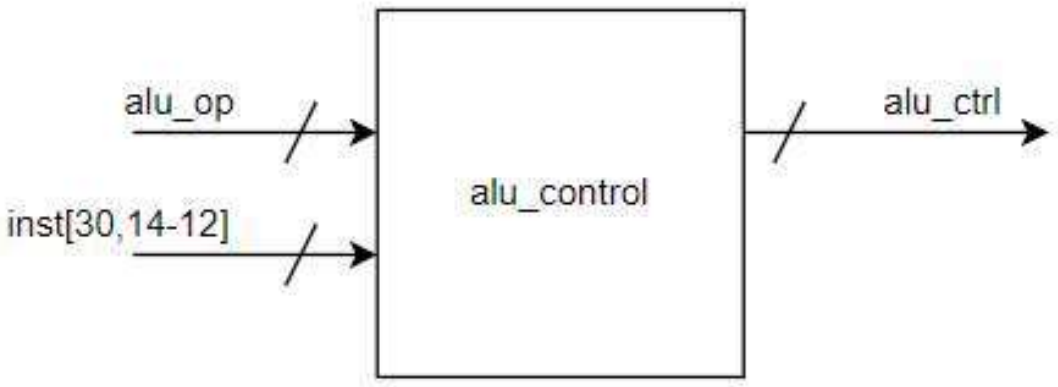


Name	Width	Input/Output	Description
ID_EX_rs1		Input	
ID_EX_rs2		Input	
MEM_WB_rd		Input	
EX_MEM_rd		Input	
EX_MEM_reg_write		Input	
MEM_WB_reg_write		Input	
IF_ID_rs1, IF_ID_rs2		Input	
ID_EX_rd		Input	
ID_EX_reg_write		Input	
EX_MEM_mem_read		Input	
branch		Input	
forward_a		Output	
forward_b		Output	
forward_comp1		Output	
forward_comp2		Output	

Bảng tên phần tử vào ra

```
module forwarding_unit (
    input          [4:0] ID_EX_rs1       ,
    input          [4:0] ID_EX_rs2       ,
    input          [4:0] ID_EX_rd        ,
    input          [4:0] IF_ID_rs1       ,
    input          [4:0] IF_ID_rs2       ,
    input          [4:0] MEM_WB_rd       ,
    input          [4:0] EX_MEM_rd       ,
    input                EX_MEM_reg_write,
    input                MEM_WB_reg_write,
    input                ID_EX_reg_write ,
    input                branch          ,
    output  logic  [1:0] forward_a       ,
    output  logic  [1:0] forward_b       ,
    output  logic  [1:0] forward_comp1   ,
    output  logic  [1:0] forward_comp2
);
```

9, Khối ALU control



Name	Width	Input/Output	Description
alu_op		Input	
inst[30, 14-12]		Input	
alu_ctrl		Output	

Bảng tên phần tử vào ra

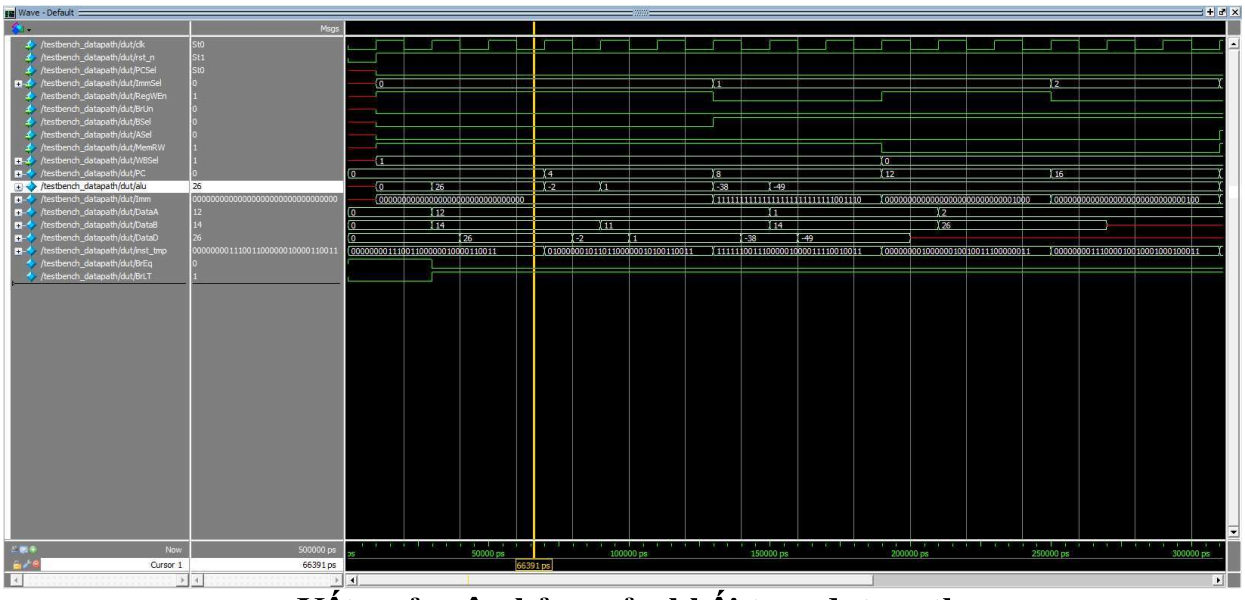
```
module ALUController(

    //Inputs
    input logic [1:0] ALUOp,
    input logic [6:0] Funct7,
    input logic [2:0] Funct3,

    //Output
    output logic [3:0] Operation //operation selection for ALU
);
```

III, Kết quả mô phỏng

Mô phỏng chương trình

- add x8, x12, x14 → 0000000 01110 01100 000 01000 0110011
- sub x10, x12, x11 → 0100000 01011 01100 000 01010 0110011
- addi x15, x1, -50 → 111111001110 00001 000 01111 0010011
- lw x14, 8(x2) → 000000001000 00010 010 01110 0000011
- sw x14, 8(x2) → 000000001110 00010 010 01000 0100011
- beq x19, x10, offset → 0 000000 01010 10011 000 1000 0 1100011

- Sinh các đầu vào bằng cách gán giá trị trong testbench



Kết quả mô phỏng của khối top_datapath

Nhận xét:

- Các tín hiệu PC, inst_tmp lên ngay cùng với sườn dương xung clk (trừ clk đầu tiên)
- DataA, DataB nhận dữ liệu ngay sau 1 chu kì clk
- ALU trả kết quả và đưa ra alu sau khi tính toán xong

- DataD nhận kết quả và ghi và thanh ghi sau nửa chu kì clk (ở sườn âm clk tiếp theo)