|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  Ảnh có chứa văn bản, ký hiệu, thực phẩm  Mô tả được tạo tự động  **BÁO CÁO BÀI TẬP LỚN**  **KIẾN TRÚC MÁY TÍNH**  Đề tài:  **Thiết kế bộ xử lý RISC-V Pipeline**  **bằng ngôn ngữ System Verilog**   |  |  | | --- | --- | | Họ và tên sinh viên | : Nguyễn Huy Nam | | Mã số sinh viên | : 20182695 | | Lớp | : Điện tử 07 – K63 | | Giảng viên hướng dẫn | : TS. Tạ Thị Kim Huệ |   ***Hà Nội – 01/2022*** |

LỜI NÓI ĐẦU

Kiến trúc RISC-V được phát triển tại Đại học California ở Berkeley bởi Yunsup Lee, Krste Asanović, David A. Patterson và Andrew Waterman. Nỗ lực này tiếp nối bốn dự án thiết kế kiến ​​trúc RISC lớn trước đây tại UC Berkeley, dẫn đến cái tên RISC-V, trong đó V đại diện cho số năm La Mã. RISC-V là một bộ kiến trúc bộ xử lý, là một mã nguồn mở cho một bộ xử lý tập lệnh rút gọn. Làm việc với Risc-V có thể biết được việc xử lý một tập lệnh của bộ xử lý diễn ra như thế nào với cấu trúc cụ thể. Và để có thể hiểu và nắm rõ hơn nữa thì em đã chọn đề tài “Thiết kế bộ xử lý RISC-V pipeline bằng ngôn ngữ System Verilog”. Đây là một đề tài hay để em có thể thực hành mô phỏng lại hệ thống RISC-V theo cơ chế pipeline thông qua ngôn ngữ System Verilog, từ đó có thể nắm vững được pipeline có ưu nhược điểm thế nào khi áp dụng cho RISC-V.

Qua những tiết học online do TS. Tạ Thị Kim Huệ giảng dạy cùng với sự nghiên cứu tìm hiểu thì em đã hoàn thành được bài tập lớn này, nhưng vẫn chưa đạt tới được mục tiêu cao nhất là pipeline cho RISC-V. Tuy nhiên trong quá trình làm bài tập lớn môn học em đã chạy được RISC-V pipeline cơ bản và nắm được kiến thức Pipeline trong RISC-V. Mong cô đưa ra những lời nhận xét cho em để em có thể rút kinh nghiệm và hoàn thành tốt hơn bài tập lớn môn học này.

MỤC LỤC

[LỜI NÓI ĐẦU i](#_Toc92223144)

[MỤC LỤC ii](#_Toc92223145)

[DANH MỤC HÌNH ẢNH v](#_Toc92223146)

[DANH MỤC BẢNG vii](#_Toc92223147)

[CHƯƠNG 1. GIỚI THIỆU RISC-V PIPELINE 1](#_Toc92223148)

[1.1 Bộ xử lý RISC-V 1](#_Toc92223149)

[1.1.1 Định nghĩa 1](#_Toc92223150)

[1.1.2 Cấu trúc RISC-V 1](#_Toc92223151)

[1.1.3 Tập lệnh RISC-V 2](#_Toc92223152)

[1.1.4 Địa chỉ RISC-V 2](#_Toc92223153)

[1.2 RISC-V Pipeline 3](#_Toc92223154)

[1.2.1 Khái niệm 3](#_Toc92223155)

[1.2.2 Pipeline Datapath 4](#_Toc92223156)

[1.2.3 Pipeline Control 8](#_Toc92223157)

[1.3 Hazard trong pipeline 10](#_Toc92223158)

[1.3.1 Structural Hazard 10](#_Toc92223159)

[1.3.2 Data Hazard 11](#_Toc92223160)

[1.3.3 Control Hazard 12](#_Toc92223161)

[CHƯƠNG 2. MÔ TẢ KIẾN TRÚC RISC-V PIPELINE 14](#_Toc92223162)

[2.1 Kiến trúc RISC-V pipeline 14](#_Toc92223163)

[2.1.1 Kiến trúc tổng quát 14](#_Toc92223164)

[2.1.2 Interface signals 14](#_Toc92223165)

[2.2 Khối Intruction Fetch (IF) 15](#_Toc92223166)

[2.2.1 Interface signals 15](#_Toc92223167)

[2.2.2 Chức năng 15](#_Toc92223168)

[2.2.3 Kiến trúc khối IF 16](#_Toc92223169)

[2.3 Khối Instruction Decode (ID) 17](#_Toc92223170)

[2.3.1 Interface signals 17](#_Toc92223171)

[2.3.2 Chức năng 18](#_Toc92223172)

[2.3.3 Kiến trúc khối ID 19](#_Toc92223173)

[2.4 Khối Excute (EX) 20](#_Toc92223174)

[2.4.1 Interface signals 21](#_Toc92223175)

[2.4.2 Chức năng 22](#_Toc92223176)

[2.4.3 Kiến trúc khối EX 22](#_Toc92223177)

[2.5 Khối Memory Access (MEM) 23](#_Toc92223178)

[2.5.1 Interface signals 23](#_Toc92223179)

[2.5.2 Chức năng 24](#_Toc92223180)

[2.5.3 Kiến trúc khối MEM 24](#_Toc92223181)

[2.6 Khối Register Write (WB) 25](#_Toc92223182)

[2.6.1 Interface signals 25](#_Toc92223183)

[2.6.2 Chắc năng 25](#_Toc92223184)

[2.6.3 Kiến trúc khối Register Write 26](#_Toc92223185)

[2.7 Khối Control Unit (control) 26](#_Toc92223186)

[2.7.1 Interface signals 27](#_Toc92223187)

[2.7.2 Chức năng 27](#_Toc92223188)

[2.8 Khối Hazard Detection Unit 27](#_Toc92223189)

[2.8.1 Interface signals 27](#_Toc92223190)

[2.8.2 Chức năng 28](#_Toc92223191)

[2.9 Khối Forwarding Unit 28](#_Toc92223192)

[2.9.1 Interface signals 29](#_Toc92223193)

[2.9.2 Chức năng 29](#_Toc92223194)

[2.10 Khối Top 31](#_Toc92223195)

[2.10.1 Interface signals 31](#_Toc92223196)

[2.10.2 Chức năng 32](#_Toc92223197)

[CHƯƠNG 3. MÔ PHỎNG VÀ KẾT QUẢ 33](#_Toc92223198)

[3.1 Mô phỏng 33](#_Toc92223199)

[3.1.1 Các phần mềm hỗ trợ 33](#_Toc92223200)

[3.1.2 Các trường hợp kiểm tra mô phỏng 34](#_Toc92223201)

[3.1.3 Đầu vào 35](#_Toc92223202)

[3.2 Kết quả và đánh giá 36](#_Toc92223203)

[3.2.1 Khối Instruction Fetch 36](#_Toc92223204)

[3.2.2 Khối Instruction Decode 36](#_Toc92223205)

[3.2.3 Khối Execute 37](#_Toc92223206)

[3.2.4 Khối Memory Access 37](#_Toc92223207)

[3.2.5 Khối Register Write 37](#_Toc92223208)

[3.2.6 Khối Control 38](#_Toc92223209)

[3.2.7 Khối Forwarding Unit 38](#_Toc92223210)

[3.2.8 Khối Hazard Detection Unit 38](#_Toc92223211)

[3.2.9 Khối Register 39](#_Toc92223212)

[3.2.10 Data Memory 40](#_Toc92223213)

[3.3 Đánh giá chung 40](#_Toc92223214)

[CHƯƠNG 4. KẾT LUẬN 41](#_Toc92223215)

[PHỤ LỤC 42](#_Toc92223216)

[TÀI LIỆU THAM KHẢO 43](#_Toc92223217)

DANH MỤC HÌNH ẢNH

[*Hình 1.1* *Single cycle versus pipelined* 3](#_Toc92223261)

[*Hình 1.2* *Single cycle versus pipelined* 4](#_Toc92223262)

[*Hình 1.3* *Single-cycle datapath* 5](#_Toc92223263)

[*Hình 1.4* *Pipeline Datapath* 6](#_Toc92223264)

[*Hình 1.5* *Giai đoạn IF của lệnh lw* 7](#_Toc92223265)

[*Hình 1.6* *Giai đoạn ID của lệnh lw* 7](#_Toc92223266)

[*Hình 1.7* *Giai đoạn EX của lệnh lw* 8](#_Toc92223267)

[*Hình 1.8* *Giai đoạn MEM của lw* 8](#_Toc92223268)

[*Hình 1.9* *Pipeline datapath với các đường tín hiệu điều khiển* 9](#_Toc92223269)

[*Hình 1.10* *ALU control bits* 10](#_Toc92223270)

[*Hình 1.11* *Đường điều khiển* 10](#_Toc92223271)

[*Hình 1.12* *Sự phụ thuộc dữ liệu giữa các câu lệnh trong pipeline* 11](#_Toc92223272)

[*Hình 1.13* *Stall và Forwarding trong pipeline* 12](#_Toc92223273)

[*Hình 1.14* *Control hazard trong pipeline* 13](#_Toc92223274)

[*Hình 2.1 Kiến trúc RISC-V pipeline* 14](#_Toc92223275)

[*Hình 2.2 Instruction Fetch block diagram* 15](#_Toc92223276)

[*Hình 2.3 Instruction fetch architecture* 16](#_Toc92223277)

[*Hình 2.4 Instruction decode block diagram* 17](#_Toc92223278)

[*Hình 2.5 Instruction Decode architecture* 19](#_Toc92223279)

[*Hình 2.6 Execute block diagram* 20](#_Toc92223280)

[*Hình 2.7 Execute architecture* 22](#_Toc92223281)

[Hình 2.8 Memory access block diagram 23](#_Toc92223282)

[*Hình 2.9 Memory access architecture* 24](#_Toc92223283)

[*Hình 2.10 Register write block diagram* 25](#_Toc92223284)

[*Hình 2.11 Register write architecture* 26](#_Toc92223285)

[*Hình 2.12 Control block diagram* 26](#_Toc92223286)

[*Hình 2.13 Hazard detection unit block diagram* 27](#_Toc92223287)

[*Hình 2.14 Forwarding unit block diagram 28*](#_Toc92223288)

[*Hình 2.15 RISC-V pipeline top block diagram 31*](#_Toc92223289)

[*Hình 3.1 Giao diện phần mềm Sublime Text 33*](#_Toc92223290)

[*Hình 3.2 Giao diện phần mềm mô phỏng Questa Sim 34*](#_Toc92223291)

[*Hình 3.3 Mô phỏng timing diagram khối Instruction Fetch 36*](#_Toc92223292)

[*Hình 3.4 Mô phỏng timing diagram khối Instruction Decode 36*](#_Toc92223293)

[*Hình 3.5 Mô phỏng timing diagram khối Execute 37*](#_Toc92223294)

[*Hình 3.6 Mô phỏng timing diagram khối Memory Access 37*](#_Toc92223295)

[*Hình 3.7 Mô phỏng timing diagram khối Register Write 37*](#_Toc92223296)

[*Hình 3.8 Mô phỏng timing diagram khối Control 38*](#_Toc92223297)

[*Hình 3.9 Mô phỏng timing diagram khối Forwarding Unit 38*](#_Toc92223298)

[*Hình 3.10 Mô phỏng timing diagram khối Hazard Detection Unit 38*](#_Toc92223299)

[*Hình 3.11 Mô phỏng 32 thanh ghi 39*](#_Toc92223300)

[*Hình 3.12 Mô phỏng Data Memory 40*](#_Toc92223301)

DANH MỤC BẢNG

[*Bảng 2.1 RISC-V pipeline port description 14*](#_Toc92223334)

[*Bảng 2.2 Instruction Fetch port description 15*](#_Toc92223335)

[*Bảng 2.3 Instruction decode port description 17*](#_Toc92223336)

[*Bảng 2.4 Execute port description 21*](#_Toc92223337)

[*Bảng 2.5 Memory access port description 23*](#_Toc92223338)

[*Bảng 2.6 Register write port description 25*](#_Toc92223339)

[*Bảng 2.7 Control port description 27*](#_Toc92223340)

[*Bảng 2.8 Forwarding Unit port description 29*](#_Toc92223341)

[*Bảng 2.9 Forwarding output ports 30*](#_Toc92223342)

[*Bảng 2.10 RISC pipeline top port description 31*](#_Toc92223343)

# GIỚI THIỆU RISC-V PIPELINE

Chương này đưa ra các khái niệm về RISC-V, cấu trúc, địa chỉ và tập lệnh trong RISC-V cùng với cái nhìn chung về RISC-V Pipeline.

## Bộ xử lý RISC-V

### Định nghĩa

**RISC** (*Reduced Instructions Set Computer*) là một phương pháp thiết kế các bộ vi xử lý theo hướng đơn giản hóa tập lệnh, trong đó thời gian thực thi tất cả các lệnh đều như nhau. Hiện nay các bộ vi xử lý RISC phổ biến là ARM, MIPS, DEC Alpha, SPARC, SuperH, PA-RISC, PIC, IBM và PowerPC.

RISC-V là một kiến trúc tập lệnh tiêu chuẩn mở (ISA) (*Instruction Set Architecture*) dựa trên nguyên tắc *Reduced Instruction Set Computer* (RISC) đã được thiết lập.

RISC–V ISA được cung cấp theo *Open Source Licenses* không yêu cầu phí sử dụng.

### Cấu trúc RISC-V

RISC-V cơ sở là một kiến ​​trúc bộ xử lý 32-bit với 31 thanh ghi có mục đích chung. Tất cả các câu lệnh đều dài 32 bit. ISA cơ sở hỗ trợ phép cộng và trừ số nguyên, nhưng không bao gồm phép nhân và phép chia số nguyên.

Các thanh ghi có chiều rộng 32 bit. Các thanh ghi mục đích chung **x1** đến **x31** có sẵn để sử dụng mà không có bất kỳ hạn chế hoặc các chức năng đặc biệt được phần cứng bộ xử lý chỉ định. Thanh ghi x0 được thiết kế cố định với giá trị trả về 0 khi nó được đọc và sẽ loại bỏ bất kỳ giá trị nào được ghi vào nó.

* **a0–a7** (x10-x17): tám thanh ghi đối số để truyền tham số và hai giá trị trả về (a0 -a1).
* **ra**: một thanh ghi địa chỉ để trả về địa chỉ ban đầu (x1).
* **s0-s1** (x8-x9) và **s2-s11** (x18-x27): các thanh ghi lưu trữ.

Thanh ghi **PC** (Program Counter) chứa bộ đếm chương trình 32-bit, giữ địa chỉ của lệnh hiện tại.

### Tập lệnh RISC-V

Tập lệnh RISC-V cơ bản chỉ bao gồm 47 lệnh. Chia làm 3 kiểu lệnh: Số học và logic (Arithmetic and Logical); Dịch chuyển dữ liệu (Data transfer and Shift); và Điều khiển rẽ nhánh (Branch).

Có 6 loại lệnh tương ứng:

* R-format: cho các phép toán số học với thanh ghi.
* I-format: cho các phép toán số học trực tiếp và tải.
* S-format: cho các lệnh lưu trữ.
* B-format: cho các lệnh rẽ nhánh.
* U-format: cho các lệnh trực tiếp với 20 bit cao.
* J-format: cho các lệnh nhảy.

Với lệnh loại **R,** 3 toán hạng là 3 thanh ghi; lệnh loại **I**, **S**, **B** là 2 thanh ghi và lệnh loại **U**, **J** là 1 thanh ghi. Các trường **funct7**, **funct3** và **imm** sẽ tương ứng với từng câu lệnh và loại lệnh khác nhau sẽ có giá trị khác nhau.

### Địa chỉ RISC-V

Có 5 loại địa chỉ:

* Immediate addressing: địa chỉ trực tiếp, trong đó toán hạng là một hằng số.
* Register addressing: địa chỉ thanh ghi, trong đó toán hạng là thanh ghi.
* Base or displacement addressing: địa chỉ cơ sở, trong đó toán hạng ở bộ nhớ có địa chỉ là tổng của một thanh ghi và một hằng số trong lệnh.
* PC-relative addressing: trong đó địa chỉ nhánh là tổng của PC và một hằng số trong lệnh.

Địa chỉ trong RISC-V được biểu diễn theo kiểu Little Endian, tức là địa chỉ được đánh số bit giảm dần từ trái sang phải. Byte bên phải nhất làm địa chỉ của từ. Đây là một trong những điểm khác biệt so với bộ xử lý MIPS. Địa chỉ của 2 ô nhớ liên tiếp cách nhau 1 từ (4 bytes)

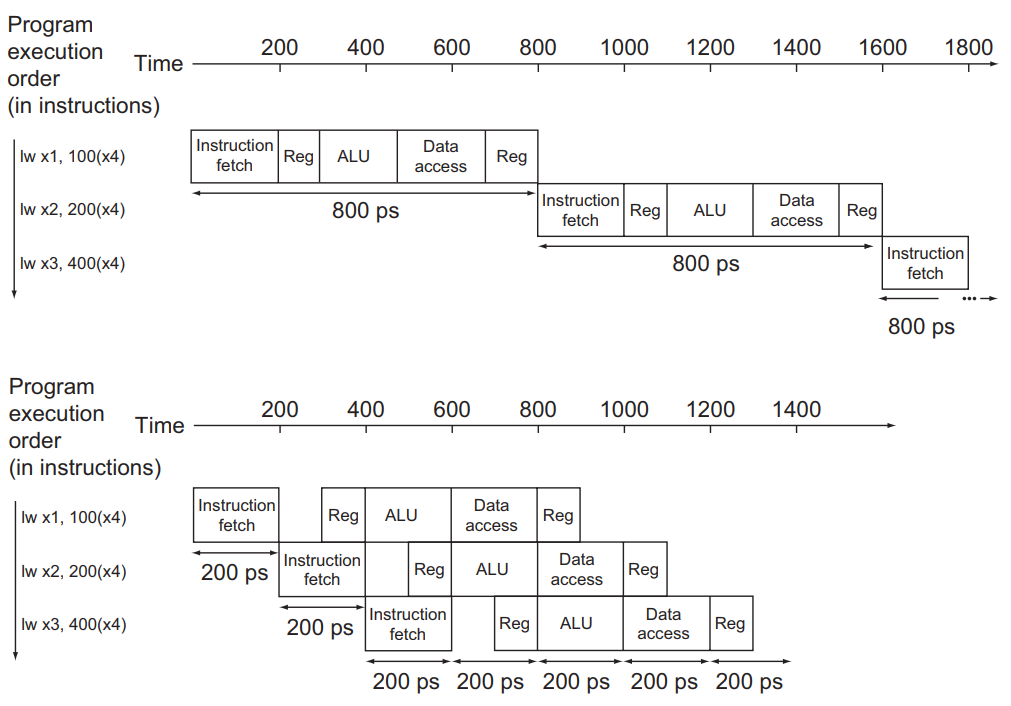
## RISC-V Pipeline

### Khái niệm

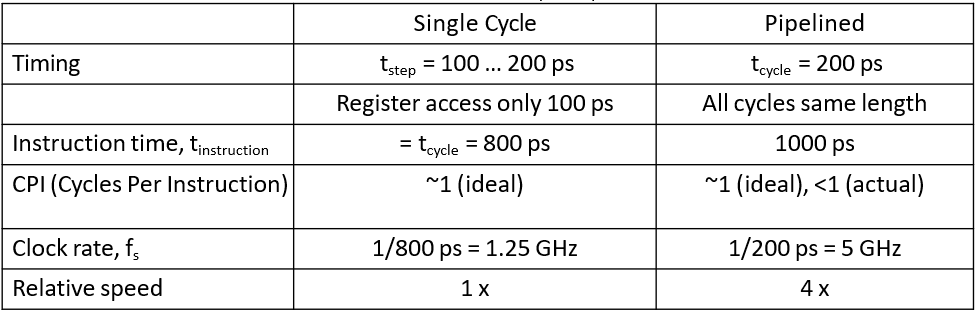
***Pipelining*** là một kĩ thuật triển khai trong đó nhiều lệnh được chồng lên nhau trong quá trình thực thi. Ngày nay, pipeline gần như phổ biến và có nhiều ứng dụng. Quá trình thực hiện một lệnh trong RISC – V cổ điển bao gồm 5 bước:

* Fetch the instruction from memory
* Read register and decode the instruction
* Excute the operation or calculate an address
* Access an operand in data memory (If necessary)
* Write result into a register (if necessary)

Hình sau mô tả so sánh giữa single – cycle (nonpipelined) và pipeline:



*Hình 1.1* *Single cycle versus pipelined*



*Hình 1.2* *Single cycle versus pipelined*

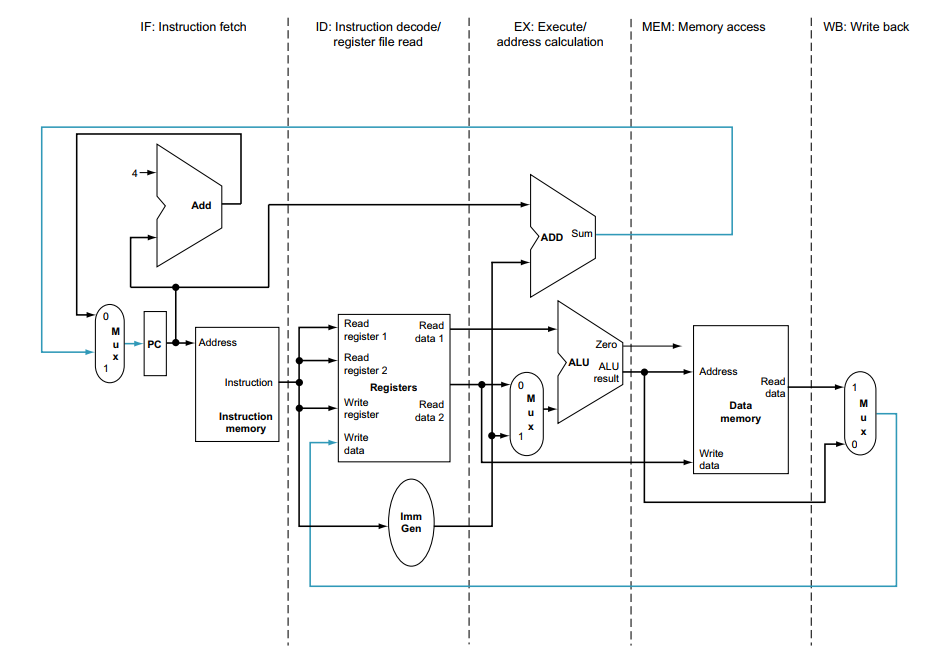
Dựa vào Hình 1.1, ta sử dụng chung một kiến trúc phần cứng, nhận thấy thời gian trung bình giữa các lệnh nhanh gấp 4 lần từ 800 ps xuống 200ps. Tuy nhiên, thời gian của giai đoạn pipeline cũng bị giới hạn bởi tài nguyên chậm nhất như hoạt động ALU hoặc truy cập vào bộ nhớ. Kết quả cụ thể được mô tả trên Hình 1.2.

Pipelining cải thiện hiệu suất bằng cách tăng thông lượng lệnh, ngược lại với việc giảm thời gian thực thi của một lệnh riêng lẻ, nhưng thông lượng lệnh là thước đo quan trọng vì các chương trình thực thực hiện hàng tỷ lệnh.

### Pipeline Datapath

Hình 1.3 dưới đây mô tả single – cycle datapath với pipeline được định nghĩa. Việc chia lệnh thành năm giai đoạn có nghĩa là một pipeline năm giai đoạn, do đó có nghĩa là tối đa năm lệnh sẽ được thực thi trong bất kỳ chu kỳ đồng hồ đơn nào (single clock cycle). Do đó, phải tách đường dữ liệu thành năm phần, với mỗi phần được đặt tên tương ứng với một giai đoạn thực thi lệnh:

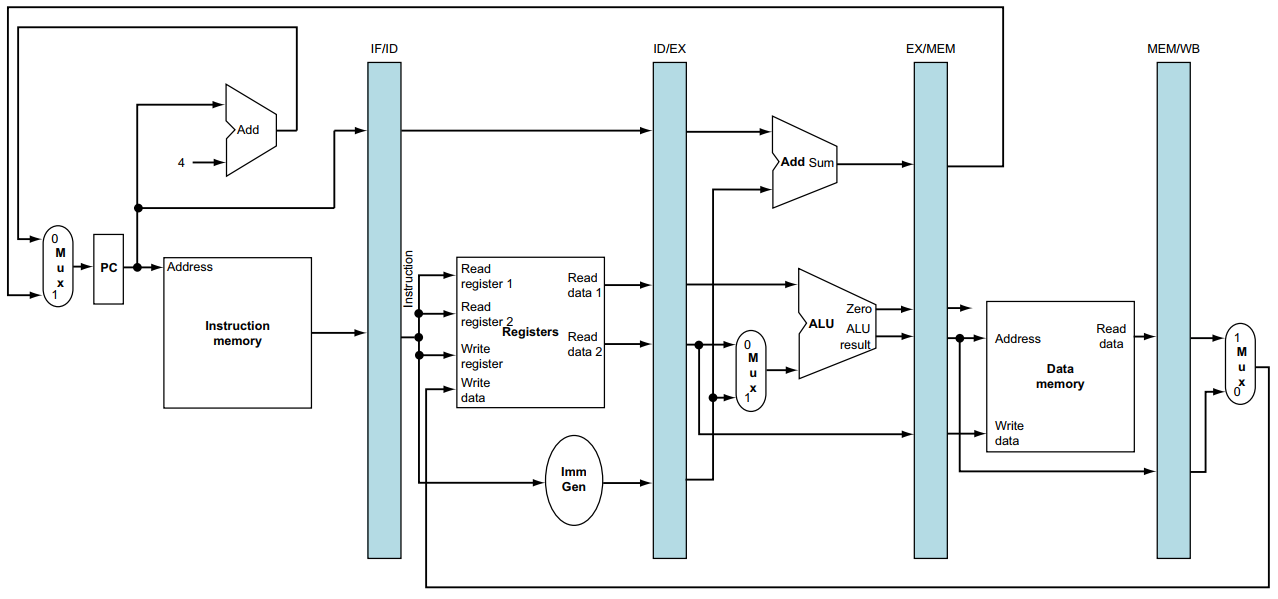
1. **IF**: *Instruction Fetch (nạp câu lệnh)*
2. **ID**: *Instruction Decode/register file read (giải mã lệnh)*
3. **EX**: *Excute/address calculation (thực thi/tính toán địa chỉ)*
4. **MEM**: *Memory access (truy cập bộ nhớ)*
5. **WB**: *Write back (ghi lại)*



*Hình 1.3* *Single-cycle datapath*

Tùy thuộc vào từng loại lệnh mà các giai đoạn có được thực thi hay không. Tại giai đoạn Write back, kết quả được lưu trữ lại thanh ghi ở phần giữa của datapath. Và việc lựa chọn giá trị tiếp theo của PC, chọn giữa giá trị PC được tăng lên 4 và địa chỉ rẽ nhánh ở giai đoạn MEM.

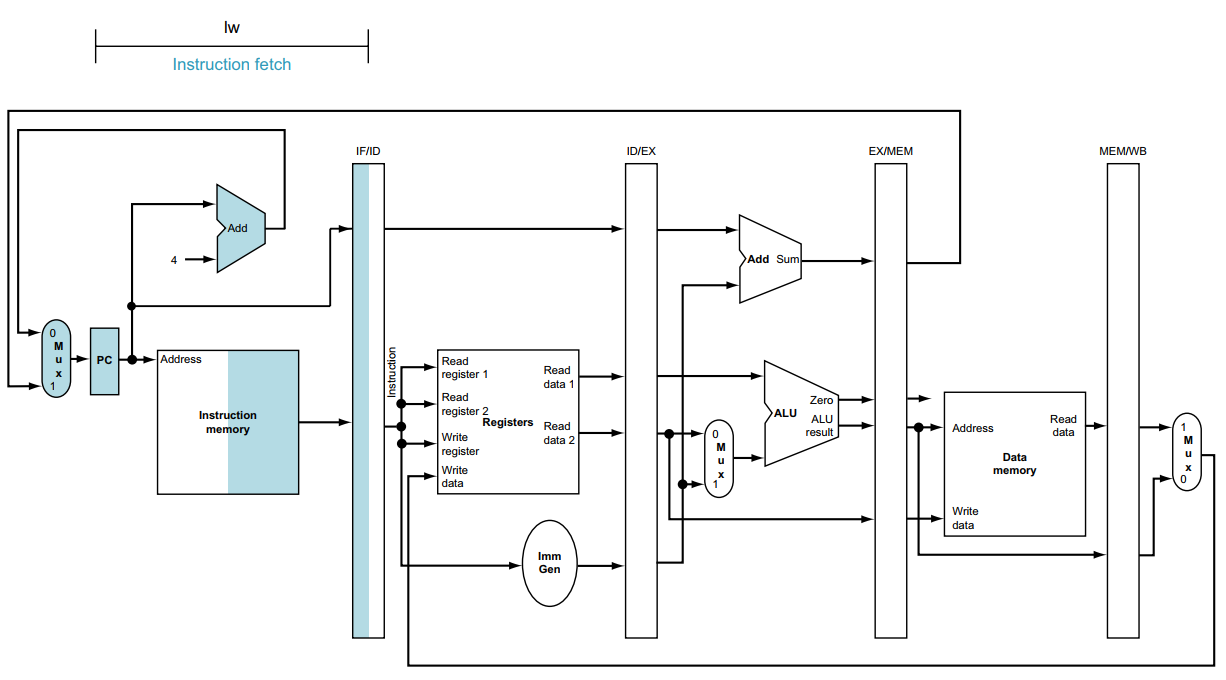
Hình 1.4 mô tả pipeline datapath với thanh ghi pipeline. Các thanh ghi pipeline được đặt tên theo hai giai đoạn bị ngăn cách bởi thanh ghi đó. Ví dụ, thanh ghi pipeline IF/ID ngăn cách hai giai đoạn IF và IF của datapath. Và không có thanh ghi tại giai đoạn cuối cùng WB. Các thanh ghi pipeline phải đủ lớn để lưu trữ các giá trị tương ứng đi qua chúng. Ví dụ, thanh ghi IF/ID phải có độ dài 96 bit vì chúng chứa 32 bit địa chỉ từ Instruction memory và 64 bit địa chỉ PC.



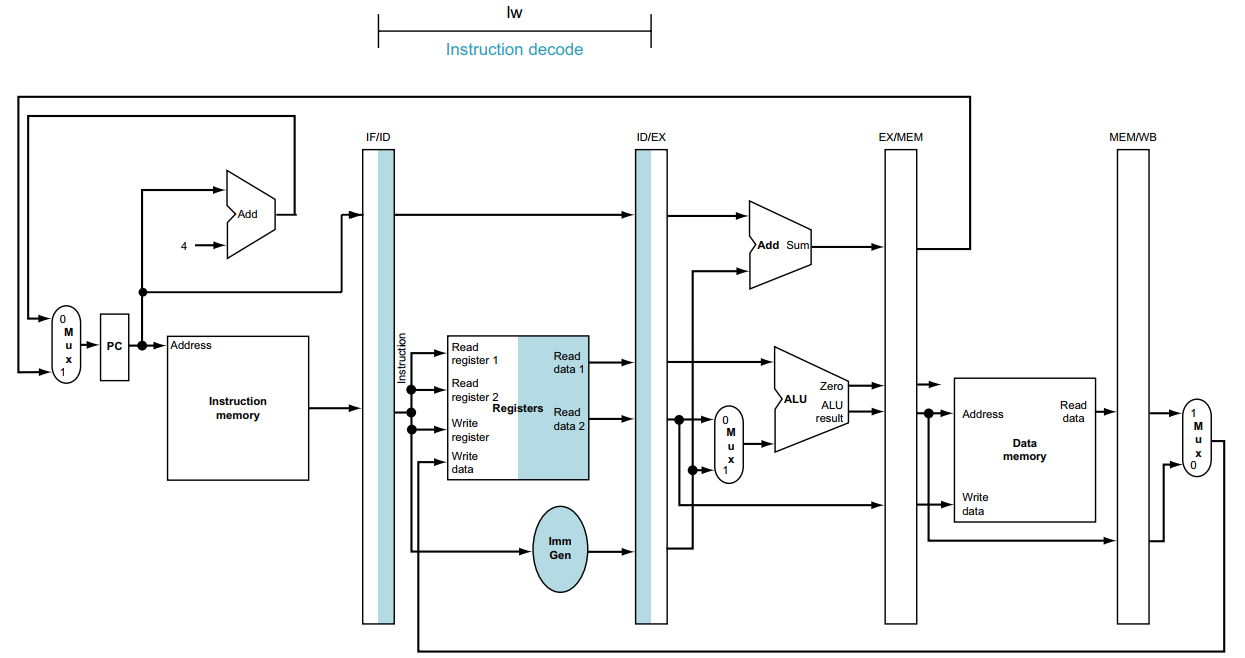
*Hình 1.4* *Pipeline Datapath*

Dưới đây là năm giai đoạn của lệnh *lw:*

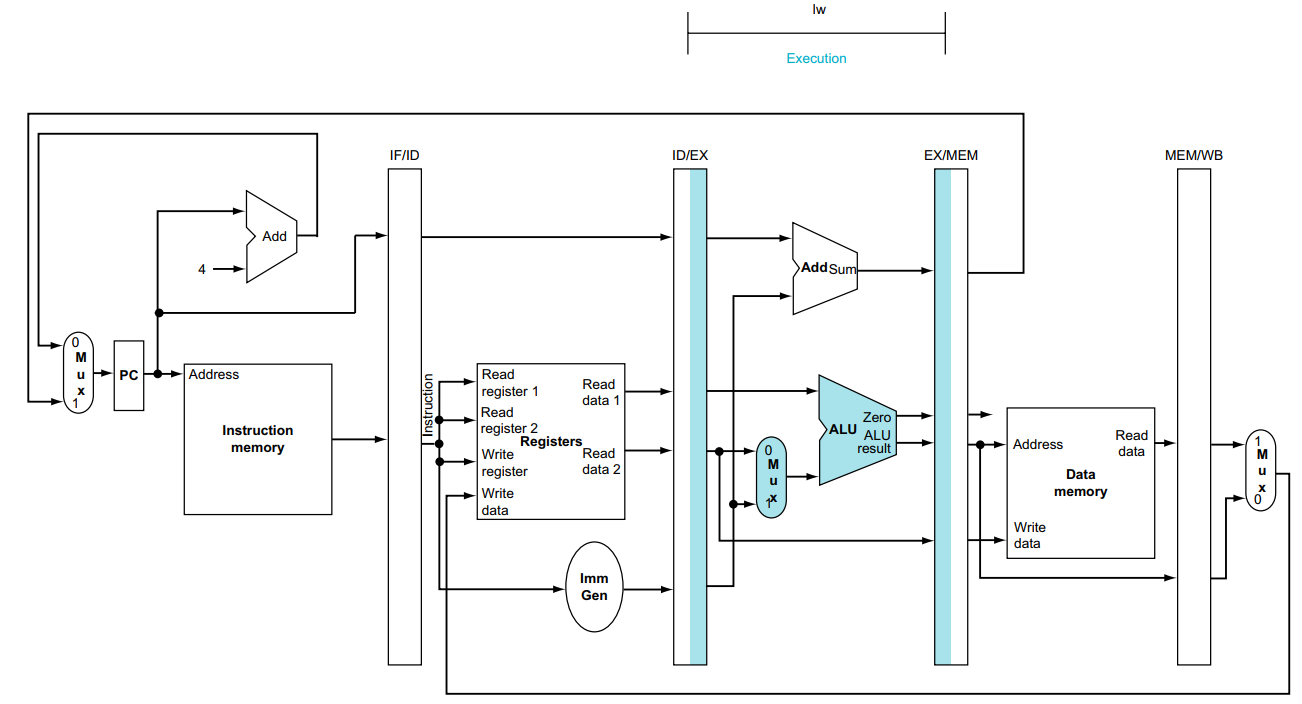
1. *Instruction Fetch (IF)*: Hình 1.5 cho thấy lệnh được đọc từ bộ nhớ bằng địa chỉ trong PC và sau đó được đặt trong thanh ghi pipeline IF/ID. Địa chỉ PC được tăng thêm 4 và sau đó được ghi lại vào PC để sẵn sàng cho chu kỳ xung nhịp tiếp theo. PC này cũng được lưu trong thanh ghi pipeline IF/ID trong trường hợp sau này cần thiết để hướng dẫn, chẳng hạn như beq. Máy tính không thể biết loại lệnh nào đang được tìm nạp, vì vậy nó phải chuẩn bị cho bất kỳ lệnh nào, chuyển thông tin có thể cần thiết xuống pipeline.
2. *Instruction Decode (ID) và register file read*: Hình 1.6 cho thấy phần lệnh của thanh ghi pipeline IF/ID cung cấp trường imediate được sign – extend đến 64 bit, và các register numbers để đọc hai thanh ghi. Tất cả ba giá trị được lưu trữ trong thanh ghi pipeline ID/EX, cùng với địa chỉ PC. Và chuyển mọi thứ có thể cần thiết bằng bất kỳ lệnh nào trong chu kỳ đồng hồ sau đó.
3. *Execute (EX) hoặc Address calculation*: Hình 1.7 cho thấy rằng lệnh *load* đọc nội dung của một thanh ghi và sign - extend imediate từ thanh ghi pipeline ID/EX và thêm chúng bằng cách sử dụng ALU. Tổng đó được đặt trong thanh ghi pipeline EX/MEM.
4. *Memory access (MEM)*: Hình 1.8 cho thấy lệnh *load* đọc nội dung bộ nhớ sử dụng địa chỉ từ thanh ghi pipeline EX/MEM và tải dữ liệu vào thanh ghi MEM/WB.
5. *Write back (WB)*: Đọc dữ liệu từ thanh ghi pipeline MEM/WB và ghi dữ liệu thanh ghi.



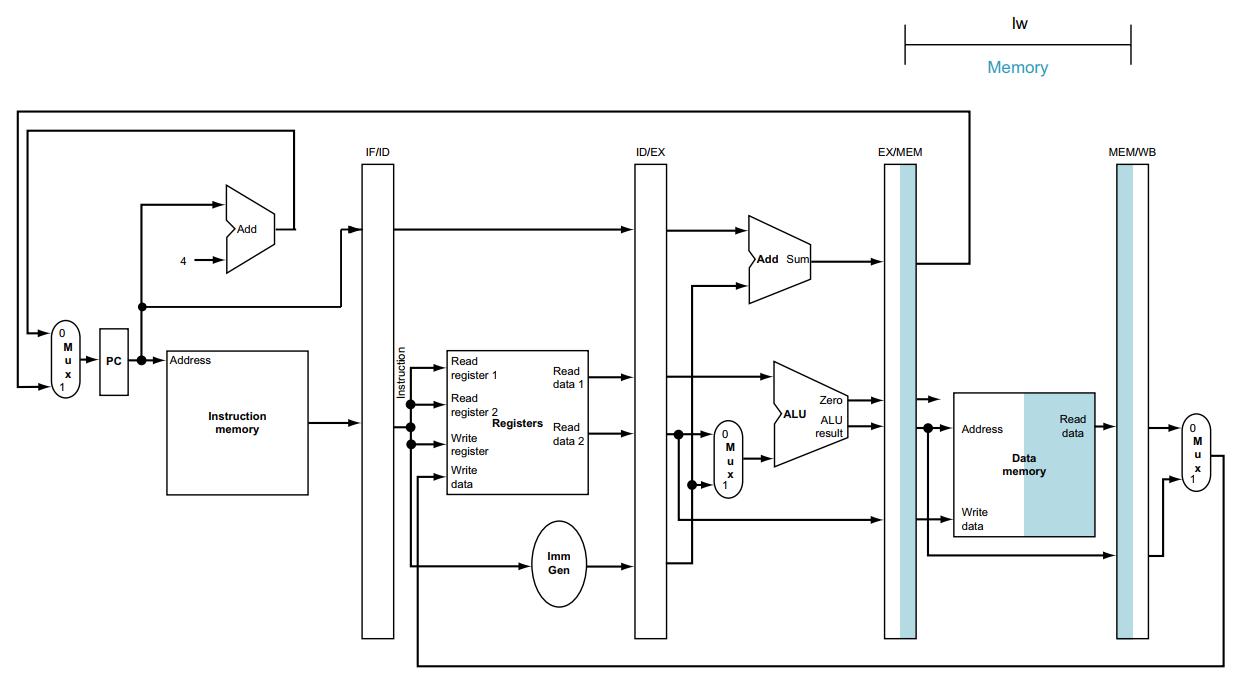
*Hình 1.5* *Giai đoạn IF của lệnh lw*



*Hình 1.6* *Giai đoạn ID của lệnh lw*



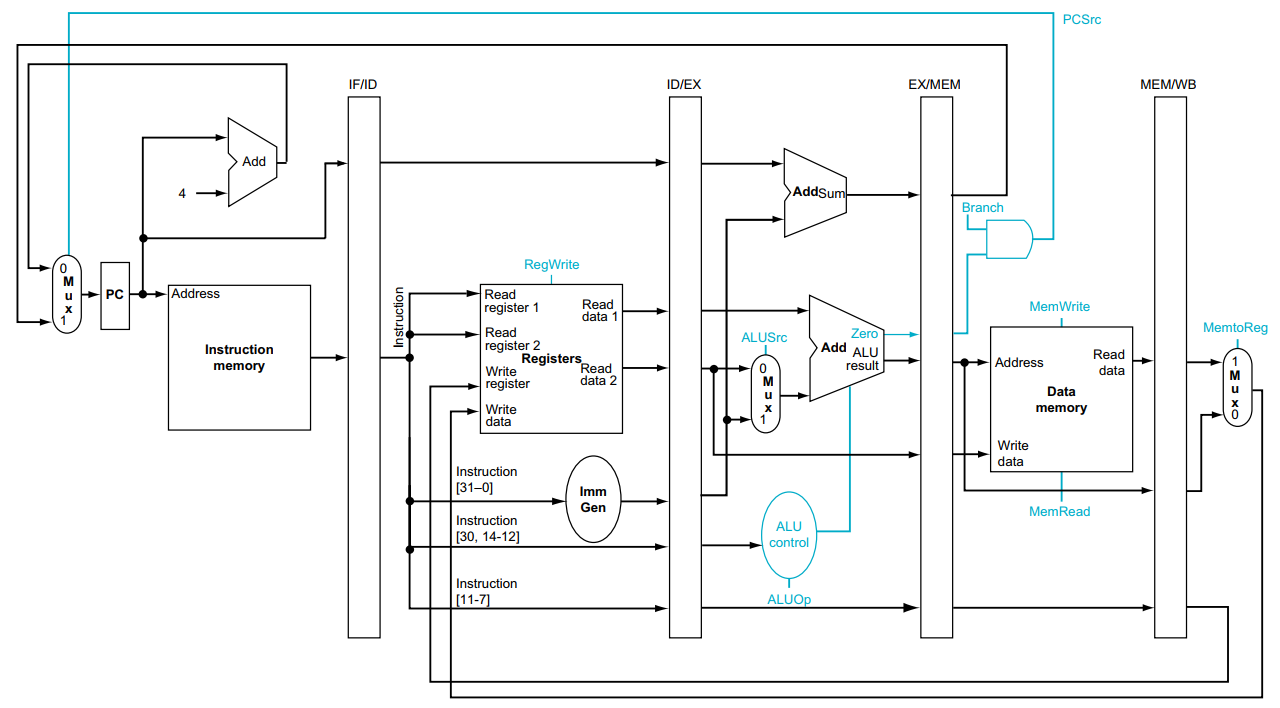
*Hình 1.7* *Giai đoạn EX của lệnh lw*



*Hình 1.8* *Giai đoạn MEM của lw*

### Pipeline Control

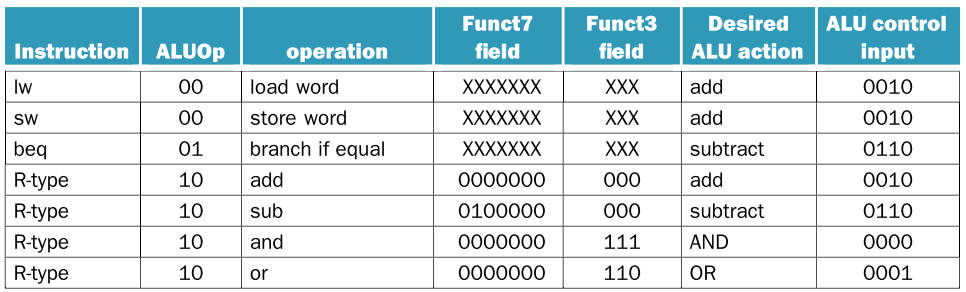
Như trường hợp thực hiện chu kỳ đơn, giả định rằng PC được ghi trên mỗi chu kỳ đồng hồ, do đó không có tín hiệu ghi riêng cho PC. Theo cùng một đối số, không có tín hiệu ghi riêng biệt cho các thanh ghi pipeline (IF/ID, ID/EX, EX/MEM và MEM/WB) vì các thanh ghi pipeline cũng được ghi trong mỗi chu kỳ đồng hồ.



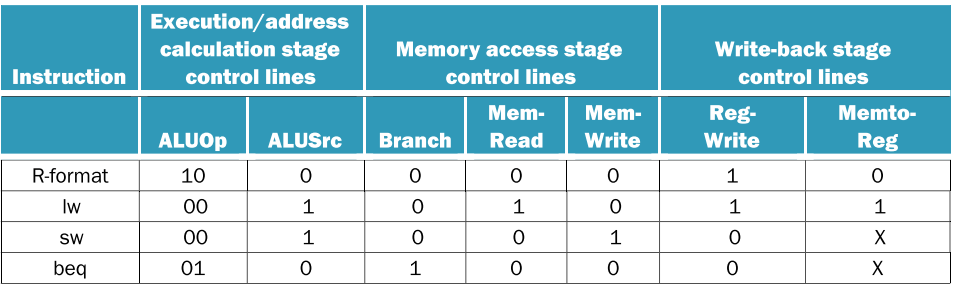
*Hình 1.9* *Pipeline datapath với các đường tín hiệu điều khiển*

Để chỉ định điều khiển cho đường ống, chúng ta chỉ cần đặt các giá trị điều khiển trong mỗi giai đoạn pipeline. Bởi vì mỗi đường điều khiển được liên kết với một thành phần chỉ hoạt động trong một giai đoạn pipeline duy nhất, chúng ta có thể chia các đường điều khiển thành năm nhóm theo giai đoạn đường ống:

1. *Instruction fetch:* Các tín hiệu điều khiển để đọc bộ nhớ lệnh và ghi PC luôn được xác nhận, vì vậy không có gì đặc biệt để điều khiển trong giai đoạn đường ống này.
2. *Instruction decode/register file read:* Hai thanh ghi nguồn luôn ở cùng một vị trí trong các định dạng lệnh RISC-V, vì vậy không có gì đặc biệt để kiểm soát trong giai đoạn đường ống này.
3. *Execute/address calculation:* Các tín hiệu được đặt là ALUOp và ALUSrc. Các tín hiệu chọn hoạt động ALU và Read data 2 hoặc sign – extended imediate làm đầu vào cho ALU.
4. *Memory access:* Các dòng điều khiển được thiết lập trong giai đoạn này là Branch, MemRead và MemWrite. Các lệnh rẽ nhánh nếu bằng nhau, tải và lưu trữ sẽ đặt các tín hiệu này tương ứng.
5. *Write back*: Hai dòng điều khiển là MemtoReg, quyết định giữa việc gửi kết quả ALU hoặc giá trị bộ nhớ vào tệp thanh ghi và RegWrite, ghi giá trị đã chọn.



*Hình 1.10* *ALU control bits*



*Hình 1.11* *Đường điều khiển*

## Hazard trong pipeline

Một vấn đề xảy ra với pipeline là hiện tượng hazard. Hazard là một tình huống ngăn cản việc bắt đầu lệnh tiếp theo trong chu kì tiếp theo. Có ba loại hazard:

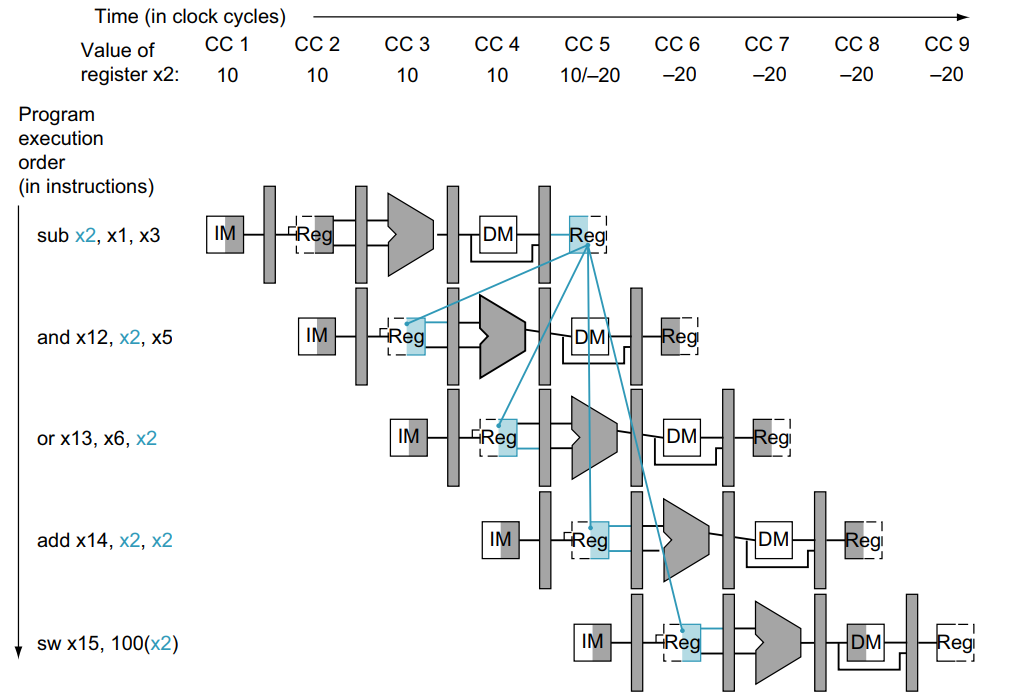
* ***Structural hazard:*** Tài nguyên yêu cầu đang bận (ví dụ: cần trong nhiều giai đoạn)
* ***Data hazard:*** Sự phụ thuộc dữ liệu giữa các câu lệnh, cần đợi lệnh trước đó hoàn thành việc đọc ghi dữ liệu của nó
* ***Control hazard:*** Luồng thực hiện phụ thuộc vào lệnh trước đó

### Structural Hazard

***Structural Hazard*** là hai hay nhiều lệnh trong pipeline cùng truy cập, yêu cầu một phần tài nguyên vật lý. Khi một lệnh được lập kế hoạch không thể thực thi trong chu kỳ đồng hồ thích hợp vì phần cứng không hỗ trợ sự kết hợp của các lệnh được thiết lập để thực thi. Structural Hazard là xung đột trong việc sử dụng tài nguyên. Trong pipeline RISC-V sử dụng một bộ nhớ duy nhất, lệnh load/store yêu cầu truy cập dữ liệu, và nếu không có phân chia bộ nhớ, việc tìm nạp lệnh sẽ phải dừng lại trong chu kì đó, các hoạt động khác trong pipeline phải dừng lại chờ đợi. Do đó, pipelined datapath yêu cầu bộ nhớ lệnh/dữ liệu riêng biệt.

### Data Hazard

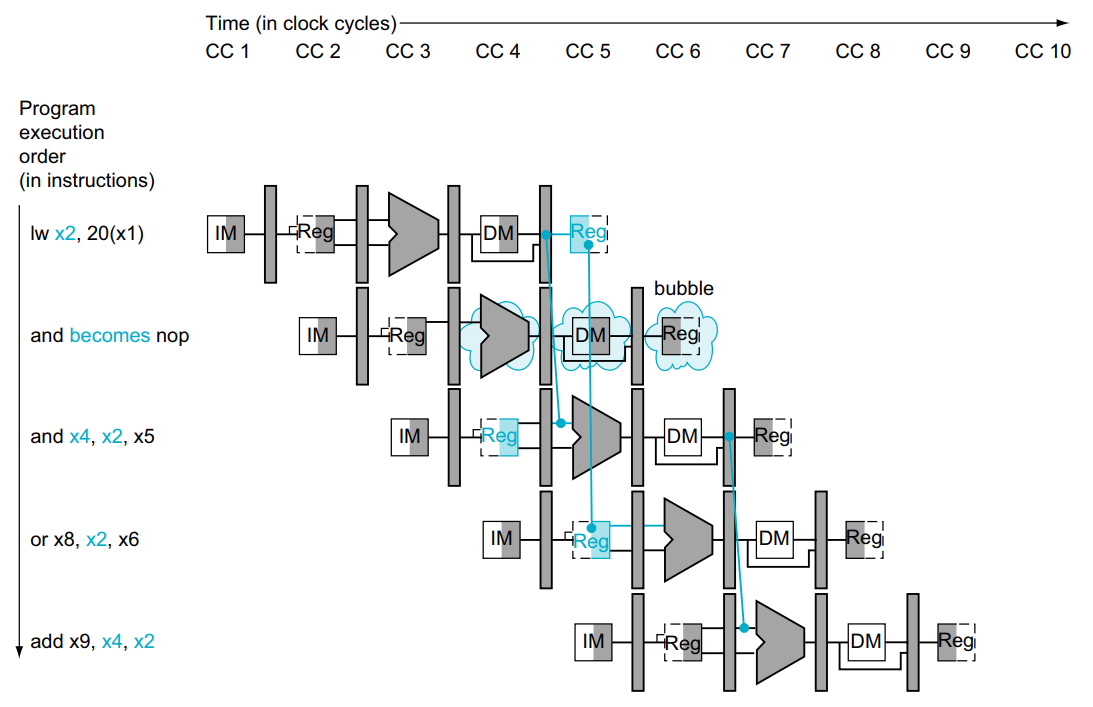
*Data Hazard* thể hiện sự phụ thuộc dữ liệu giữa các câu lệnh trong pipeline, câu lệnh trước cần hoàn thành việc đọc ghi dữ liệu của nó. Hình 1.12 dưới đây thể hiện sử phụ thuộc giữa các câu lệnh trong pipeline.



*Hình 1.12* *Sự phụ thuộc dữ liệu giữa các câu lệnh trong pipeline*

Lệnh đầu tiên ghi dữ liệu vào x2, các lệnh sau nó đều đọc dữ liệu từ x2 để thực hiện lệnh. Tuy nhiên, việc ghi dữ liệu vào x2 được thực hiện ở chu kì 5 (CC5), các chu kì trước đó hoàn toàn chưa có giá trị của x2 sau khi được tính toàn và lưu vào trong x2.

Hướng giải quyết cho vấn đề data hazard là stall và forwarding. Stall là chuyển lệnh thành “nops”, các giai đoạn pipeline không thực thi gì và các giai đoạn tiếp theo thực thi bình thường. Forwarding dựa vào việc khi thực hiện lệnh, giá trị của kết quả có từ các giai đoạn trước của pipeline trước khi được ghi vào memory hoặc register. Nếu lệnh trong giai đoạn ID bị stall, thì lệnh trong giai đoạn IF cũng phải bị stall; nếu không, sẽ mất instruction đã được nạp. Việc ngăn cản hai lệnh này thực hiện tiến trình được thực hiện đơn giản bằng cách ngăn thanh ghi PC và thanh ghi pipeline IF/ID thay đổi. Miễn là các thanh ghi này được giữ nguyên, lệnh trong giai đoạn IF sẽ tiếp tục được đọc bằng cách sử dụng cùng một PC và các thanh ghi trong giai đoạn ID sẽ tiếp tục được đọc bằng cách sử dụng các trường lệnh tương tự trong thanh ghi pipeline IF/ID. Nửa sau của pipeline bắt đầu với giai đoạn EX phải đang làm gì đó; những gì nó đang làm là thực hiện các lệnh không có tác dụng thực thi: nops. Cụ thể phương pháp stall và forwarding được mô tả trên Hình 1.13 dưới đây.

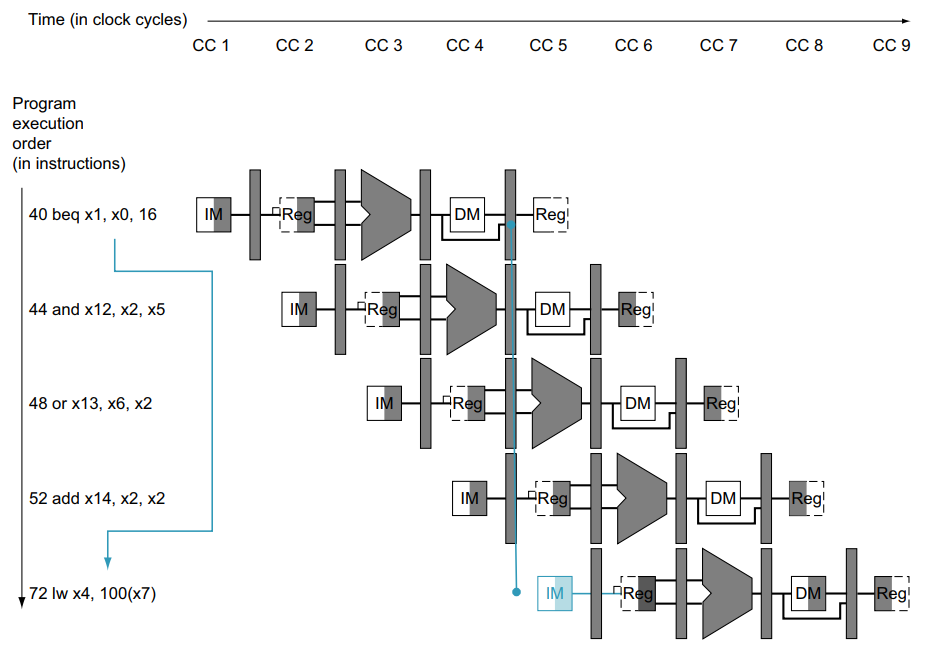


*Hình 1.13* *Stall và Forwarding trong pipeline*

Một bubble được chèn vào bắt đầu từ chu kỳ đồng hồ 4, bằng cách thay đổi lệnh *and* thành *nop*. Lưu ý rằng lệnh và thực sự được tìm nạp và giải mã trong chu kỳ đồng hồ 2 và 3, nhưng giai đoạn EX của nó bị trì hoãn cho đến chu kỳ đồng hồ 5 (so với vị trí không được chặn trong chu kỳ đồng hồ 4). Tương tự như vậy, lệnh hoặc được tìm nạp trong chu kỳ đồng hồ 3, nhưng giai đoạn ID của nó bị trì hoãn cho đến chu kỳ đồng hồ 5 (so với vị trí của chu kỳ đồng hồ 4 không được chặn). Sau khi chèn bubble, tất cả các phần phụ thuộc sẽ tiếp tục theo thời gian và không có thêm hazard nào xảy ra.

### Control Hazard

Control hazard do lệnh rẽ nhánh gây ra, luồng thực hiện phụ thuộc vào lệnh trước đó. Nếu lệnh rẽ nhánh không được thực hiện, các instruction fetch sau nó được thực hiện chính xác. Nếu lệnh rẽ nhánh được thực hiện, phải loại bỏ các hướng dẫn không chính xác ra khỏi pipeline bằng cách chuyển chúng về NOPs. Hình 1.14 dưới đây mô tả vấn đề control hazard.



*Hình 1.14* *Control hazard trong pipeline*

Một cách để cải thiện hiệu suất của nhánh có điều kiện là giảm chu kì clk phải đánh đổi khi lệnh rẽ nhánh được thực hiện. Giả sử PC tiếp theo cho một nhánh được chọn trong giai đoạn MEM, nhưng nếu chúng ta di chuyển việc thực thi nhánh có điều kiện sớm hơn trong pipeline, thì cần phải xóa ít lệnh hơn do vậy số chu kì clk phải đánh đổi khi lệnh rẽ nhánh được thực hiện. Chuyển việc quyết định có rẽ nhánh lên giai đoạn sớm hơn (thay vi trong giai đoạn MEM) đòi hỏi hai hành động xảy ra trước đó: tính toán địa chỉ mục tiêu nhánh và đánh giá quyết định nhánh. Việc tính toán địa chỉ nhánh sớm hơn khá dễ dàng triển khai do đã có giá trị PC và trường ngay lập tức trong thanh ghi pipeline IF/ID, vì vậy chúng ta chỉ cần di chuyển bộ cộng nhánh từ giai đoạn EX sang giai đoạn ID; tất nhiên, tính toán địa chỉ cho các mục tiêu nhánh sẽ được thực hiện cho tất cả các lệnh, nhưng chỉ được sử dụng khi cần thiết.

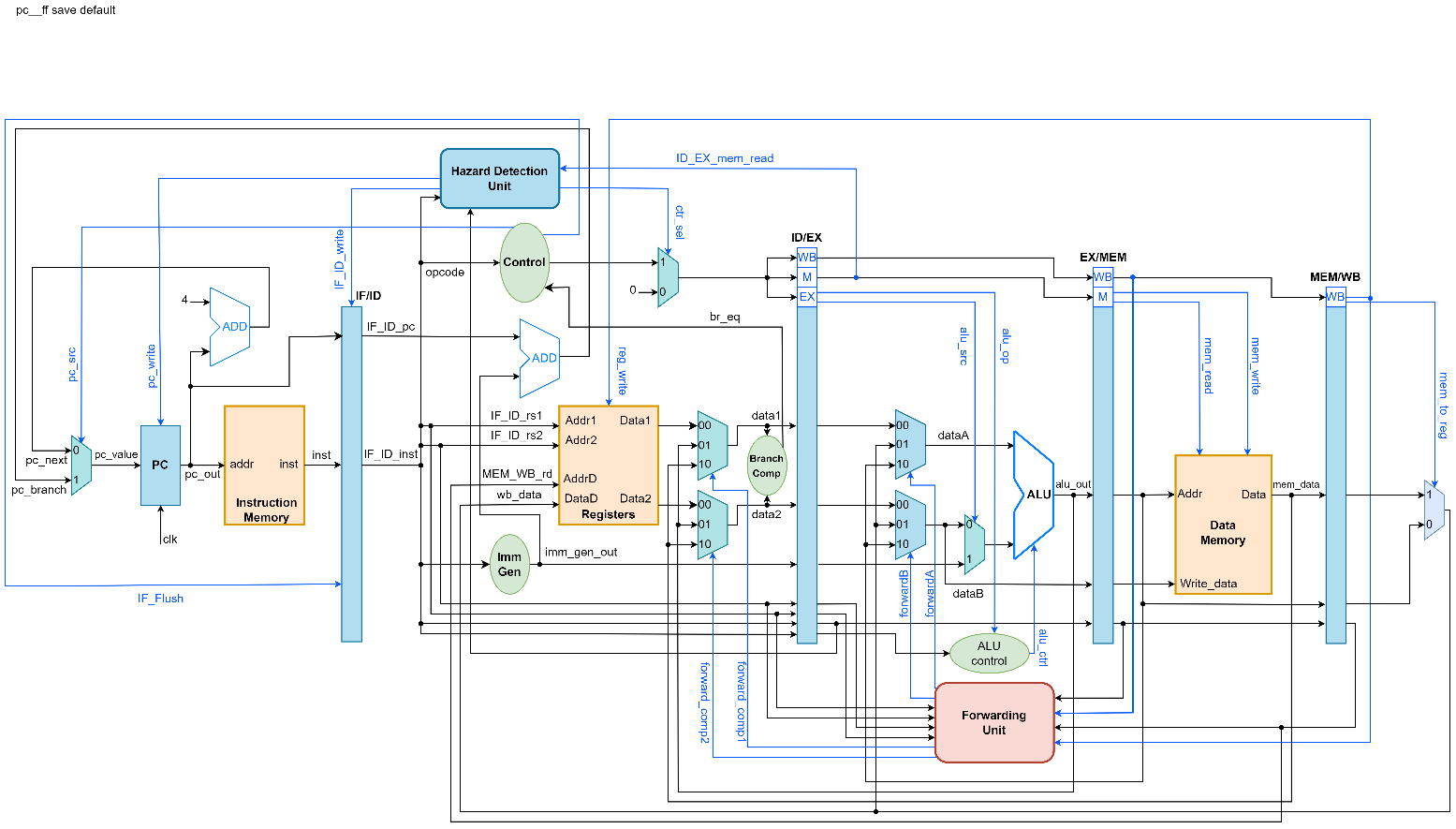
Phần khó hơn là quyết định nhánh nào được chọn. Đối với nhánh nếu bằng nhau, chúng ta sẽ so sánh hai lần đọc thanh ghi trong giai đoạn ID để xem chúng có bằng nhau hay không. Sự bằng nhau có thể được kiểm tra bằng cách XOR các vị trí bit riêng lẻ của hai thanh ghi và OR kết quả XOR (một đầu ra bằng không của cổng OR có nghĩa là hai thanh ghi bằng nhau), và có thể sử dụng "dự đoán nhánh" để đoán nhánh nào sẽ đi sớm hơn trong pipeline và chỉ flush pipeline nếu dự đoán nhánh không chính xác.

# MÔ TẢ KIẾN TRÚC RISC-V PIPELINE

Chương này sẽ mô tả kiến trúc tổng quát của RISC-V Pipeline và chi tiết cấu trúc của từng phần có trong kiến trúc RISC-V Pipeline

## Kiến trúc RISC-V pipeline

### Kiến trúc tổng quát



*Hình 2.1 Kiến trúc RISC-V pipeline*

Hình 2.1 mô tả kiến trúc hoàn chỉnh cho cpu **riscv32i** hỗ trợ tập lệnh đơn giản gồm đầy đủ 4 loại lệnh R-type, I-type, S-type, B-type,được áp dụng kĩ thuật pipeline và xử lí các loại hazard (data, mem, control) bằng sự kết hợp của **Hazard detection unit** và **Forwarding unit**.

### Interface signals

Bảng 2.1 RISC-V pipeline port description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |

CPU sẽ nhận xung **clk** và tín hiệu **reset** để hoạt động, chương trình cần CPU thực hiện sẽ được nạp sẵn vào bộ nhớ **Instruction memory**.

## Khối Intruction Fetch (IF)

Diagram

Description automatically generated

*Hình 2.2 Instruction Fetch block diagram*

### Interface signals

Bảng 2.2 Instruction Fetch port description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **pc\_branch** | 32 | Input | Giá trị PC cần nhảy đến khi có lệnh rẽ nhánh |
| **pc\_write** | 1 | Input | Cho phép thay đổi đầu ra PC |
| **pc\_src** | 1 | Input | Cho phép chọn PC+4 hoặc PC\_branch |
| **IF\_flush** | 1 | Input | Cho phép xóa thanh ghi IF\_ID |
| **IF\_ID\_write** | 1 | Input | Cho phép thanh ghi IF\_ID ghi dữ liệu mới |
| **IF\_ID\_pc** | 32 | Output | Đầu ra thanh ghi cho biết giá trị PC |
| **IF\_ID\_inst** | 32 | Output | Đầu ra thanh ghi cho biết mã lệnh tương ứng với PC |

### Chức năng

Khối Instruction Fetch tìm nạp lệnh từ bộ *instruction memory*. Mã lệnh được đọc từ bộ nhớ bằng địa chỉ PC và được đặt trong thanh ghi pipeline IF/ID. Địa chỉ PC được tăng thêm 4 và sau đó được đưa vào bộ mux để chọn giữa PC+4 và PC cần nhảy đến. Tiếp theo PC được ghi lại để sẵn sàng cho chu kỳ xung nhịp tiếp theo.

Giá trị PC được lưu trong thanh ghi IF/ID trong trường hợp cần cho lệnh rẽ nhánh vì vậy nó phải chuẩn bị cho bất kỳ lệnh nào, chuyển thông tin có thể cần thiết xuống pipeline.

### Kiến trúc khối IF

Diagram

Description automatically generated

*Hình 2.3 Instruction fetch architecture*

Hình 2.3 mô tả các phần tử có trong kiến trúc của khối IF gồm:

* **Mux2to1** nhận tín hiệu **pc\_src** từ khối **control** và quyết định xem giá trị **pc** tiếp theo đưa vào thanh ghi lưu **PC**
* Thanh ghi **PC** nhận tín hiệu **pc\_write** từ khối **hazard detection unit** để nhằm mục đích stall cho CPU, khi **pc\_write = 1** thì **pc** tiếp theo từ **mux2to1** sẽ được đưa vào thanh ghi **PC** nếu **pc\_write = 0** thì thanh ghi **PC** sẽ giữ nguyên giá trị khí đó giá trị **pc** sẽ được giữ nguyên (CPU được stall)
* Bộ cộng nhằm mục đích cộng thêm 4 vào giá trị **pc** để chương trình nạp lệnh tiếp theo có địa chỉ **pc + 4**
* **Instruction memory** là bộ nhớ lưu chương trình gồm các lệnh assembly cần **CPU** thực hiện. **Instruction memory** nhận đầu vào là địa chỉ **pc** của lệnh và đưa ra lệnh (**inst**) tại địa chỉ **pc** tương ứng cần nạp vào **CPU** để thực hiện
* **IF/ID** là thanh ghi pipeline lưu kết quả **pc** và **inst** tương ứng đã được nạp trong giai đoạn nạp lệnh IF (instruction fetch). Tín hiệu điều khiển **IF\_flush** nhận từ khối **control** để xoá flush thanh ghi khi có **control hazard** (khi lệnh được nạp ngay sau lệnh **beq** sai). Tín hiệu điều khiển **IF\_ID\_write** nhằm mục đích stall pipeline nhằm xử lí hazard, khi **IF\_ID\_write = 1** thì thanh ghi **IF/ID** sẽ nạp kết quả **pc** và **inst** của lệnh tiếp theo như bình thường, nếu **IF\_ID\_write = 0** thì thanh ghi sẽ giữ nguyên giá trị của lệnh đã được nạp trước đó (stall pipeline)

## Khối Instruction Decode (ID)

Table

Description automatically generated with medium confidence

*Hình 2.4 Instruction decode block diagram*

### Interface signals

Bảng 2.3 Instruction Decode port description

|  |  |  |  |
| --- | --- | --- | --- |
| **Signal name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **IF\_ID\_pc** | 32 | Input | Giá trị PC lưu trong thanh ghi IF\_ID |
| **IF\_ID\_inst** | 32 | Input | Mã lệnh lưu trong thanh ghi IF\_ID |
| **MEM\_WB\_reg\_write** | 1 | Input | Tín hiệu cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_rd** | 5 | Input | Địa chỉ thanh ghi đích lưu trong thanh ghi MEM/WB |
| **wb\_data** | 32 | Input | Dữ liệu được write back |
| **imm\_sel** | 3 | Input | Chọn kiểu cho Immidiate Generate |
| **mem\_to\_reg** | 1 | Input | Tín hiệu cho phép write back |
| **reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi |
| **mem\_write** | 1 | Input | Cho phép memory ghi dữ liệu |
| **mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu |
| **alu\_src** | 1 | Input | Chọn chế độ địa chỉ trực tiếp |
| **alu\_op** | 2 | Input | Opcode chọn phép toán |
| **ctrl\_sel** | 1 | Input | Chọn đầu ra control |
| **forward\_comp1** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs1 |
| **forward\_comp2** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs2 |
| **alu\_out** | 32 | Input | Kết quả alu được forward về |
| **mem\_data** | 32 | Input | Dữ liệu memory được forward về |
| **EX\_MEM\_alu\_out** | 32 | Input | Kết quả của alu trong thanh ghi EX/MEM được forward về |
| **EX\_MEM\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi EX/MEM |
| **pc\_branch** | 32 | Output | Giá trị pc cần nhảy đến |
| **br\_eq** | 1 | Output | Kết quả so sánh data1 và data2 |
| **ID\_EX\_mem\_to\_reg** | 1 | Output | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi ID/EX |
| **ID\_EX\_reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_write** | 1 | Output | Cho phép memory ghi dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_read** | 1 | Output | Cho phép memory đọc dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_src** | 1 | Output | Chọn chế độ địa chỉ trực tiếp lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_op** | 2 | Output | Opcode chọn phép toán lưu trong thanh ghi ID/EX |
| **ID\_EX\_data1** | 32 | Output | Dữ liệu từ thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_data2** | 32 | Output | Dữ liệu từ thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs1** | 5 | Output | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs2** | 5 | Output | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Output | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **ID\_EX\_imm\_gen** | 32 | Output | Kết quả immidiate lưu trong thanh ghi ID/EX |
| **ID\_EX\_inst\_func** | 4 | Output | Function chọn phép toán lưu trong thanh ghi ID/EX |

### Chức năng

Khối Instruction Decode lấy đầu vào là instruction đã được nạp vào thanh ghi IF/ID và thực hiện:

* Đọc dữ liệu từ tệp thanh ghi dựa vào địa chỉ rs1, rs2, rd lấy từ instruction
* Gen ra 32bit immediate từ trường imm trong instruction
* Gửi trường opcode của instruction sang khối control để đưa ra các tín hiệu điều khiển đến các khối khác
* So sánh giá trị data của của 2 thanh ghi rs1, rs2 trong lệnh branch và gửi kết quả so sanh sang khối control. Nhận tín hiệu forward từ khối forwarding unit để quyết định xem sẽ so sánh giá trị thanh ghi hay giá trị được forward về trong trường hợp có hazard.

### Kiến trúc khối ID

Diagram, schematic

Description automatically generated

*Hình 2.5 Instruction Decode architecture*

Hình 2.5 mô tả kiến trúc của khối **Instruction decode** gồm:

* Tệp thanh ghi nhận các giá trị địa chỉ thanh ghi **rs1**, **rs2** từ **inst** để đưa ra giá trị **data1, data2** tương ứng cho 2 thanh ghi. Nhận địa chỉ **rd**, **wb\_data** từ khối **Register write** (thực hiện giai đoạn write back) và tín hiệu **reg\_write** từ **control** để ghi kết quả tính toán vào thanh ghi **rd**
* Bộ cộng nhằm tính toán địa chỉ **pc** cho lệnh brach
* Imm Gen nhằm signed extend cho giá trị **immediate**
* 2 bộ **mux4to1** nhận tín hiệu điều khiển từ khối **Forwarding unit** nhằm forward các kết quả tính toán của giai đoạn EX và MEM về bộ so sánh (branch compare) nhằm khắc phục data hazard khi thực hiện lệnh branch.
* Branch comp là bộ so sánh so sánh giá trị của 2 thanh ghi **rs1**, **rs2** từ lệnh branch và đưa ra kết quả đến khối **control** để xác định xem chương trình sẽ nhảy hay sẽ thực hiện lệnh tiếp theo có địa chỉ **pc + 4**
* Thanh ghi **ID/EX** sẽ nhận kết quả từ khối **Instruction Decode** đồng thời nhận các tín hiệu điều khiển được đưa ra từ khối **control** để gửi đến các khối **Execute, Memory access** và **Register write** để thực hiện tính toán cho các giai đoạn pipeline sau của lệnh

## Khối Excute (EX)

A picture containing text

Description automatically generated

*Hình 2.6 Execute block diagram*

### Interface signals

Bảng 2.4 Execute port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **ID\_EX\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi ID/EX |
| **ID\_EX\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_write** | 1 | Input | Cho phép memory ghi dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_src** | 1 | Input | Chọn chế độ địa chỉ trực tiếp lưu trong thanh ghi ID/EX |
| **ID\_EX\_alu\_op** | 2 | Input | Opcode chọn phép toán lưu trong thanh ghi ID/EX |
| **ID\_EX\_data1** | 32 | Input | Dữ liệu từ thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_data2** | 32 | Input | Dữ liệu từ thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_imm\_gen** | 32 | Input | Kết quả immidiate lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Input | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **ID\_EX\_inst\_func** | 4 | Input | Function chọn phép toán lưu trong thanh ghi ID/EX |
| **forward\_a** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs1 |
| **forward\_b** | 2 | Input | Forwarding khi có hazard tại thanh ghi rs2 |
| **wb\_data** | 32 | Input | Dữ liệu được write back |
| **EX\_MEM\_alu\_out** | 32 | Output | Kết quả alu được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_to\_reg** | 1 | Output | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi EX/MEM |
| **EX\_MEM\_reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_write** | 1 | Output | Cho phép memory ghi dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_read** | 1 | Output | Cho phép memory đọc dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_dataB** | 32 | Output | Dữ liệu thanh ghi 2 được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_rd** | 5 | Output | Địa chỉ thanh ghi đích được lưu trong thanh ghi EX/MEM |
| **alu\_out** | 32 | Output | Kết quả alu |

### Chức năng

Khối Execute sẽ nhận các tín hiệu điều khiển cho giai đoạn EX đã được lưu trong thanh ghi ID/EX và thực hiện tính toán dữ liệu hoặc địa chỉ tương ứng với yêu cầu của các instruction bằng khối ALU và ALU control.

### Kiến trúc khối EX

Diagram, schematic

Description automatically generated

*Hình 2.7 Execute architecture*

Hình 2.7 mô tả kiến trúc của khối **Execute** gồm:

* 2 mộ **mux4to1** nhận tín hiệu từ khối **Forwarding unit** nhằm forward đầu vào cho khối **ALU** trong trường hợp hazard
* **Mux2to1** nhận tín hiệu **alu\_src** từ khối **control** để chọn giữa dữ liệu thanh ghi **rs2** hoặc giá trị đầu ra của khối **Imm Gen** đã được lưu trong thanh ghi **ID/EX**
* **ALU control** nhận tín hiệu **alu\_op** từ khối **control** và đưa ra tín hiệu **alu­\_ctrl** đến khối **ALU** để điều khiển hoạt động của **ALU** theo bảng sự thật
* **ALU** sẽ dựa vào tín hiệu **alu­\_ctrl** nhận được từ khối **ALU control** để thực hiện các phép toán tương ứng (add, sub, AND, OR, XOR,…)
* **ID/EX** là thanh ghi pipeline nhận các kết quả tính toán từ **ALU, dataB** và các giá trị của tín hiệu điều khiển để phục vụ cho các giai đoạn pipeline sau của lệnh. Bên cạnh đó còn nhận giá trị địa chỉ thanh ghi **rd** phục vụ cho việc nhận diện các trường hợp có hazard

## Khối Memory Access (MEM)

Text

Description automatically generated

Hình . Memory access block diagram

### Interface signals

Bảng 2.5 Memory Access port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **EX\_MEM\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi EX/MEM |
| **EX\_MEM\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_mem\_write** | 1 | Input | Cho phép memory ghi dữ liệu lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_alu\_out** | 32 | Input | Kết quả alu được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_dataB** | 32 | Input | Dữ liệu thanh ghi 2 được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_rd** | 5 | Input | Địa chỉ thanh ghi đích được lưu trong thanh ghi EX/MEM |
| **MEM\_WB\_reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_mem\_to\_reg** | 1 | Output | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi MEM/WB |
| **MEM\_WB\_mem\_data** | 32 | Output | Dữ liệu từ memory được lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_alu\_out** | 32 | Output | Kết quả alu được lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_rd** | 5 | Output | Địa chỉ thanh ghi đích được lưu trong thanh ghi MEM/WB |
| **mem\_data** | 32 | Output | Dữ liệu từ memory |

### Chức năng

Khối Memory access nhận các tín hiệu điều khiển đọc ghi cho giai đoạn MEM từ thanh ghi EX/MEM để đọc hoặc ghi dữ liệu từ Data memory.

### Kiến trúc khối MEM

Diagram

Description automatically generated

*Hình 2.9 Memory access architecture*

Hình 2.9 mô tả kiến trúc khối **Memory access**, gồm một bộ nhớ **Data Memory** nhận tín hiệu điều khiển **mem\_read**, **mem\_write** điều khiển việc đọc ghi memory. **Data Memory** nhận địa chỉ đã được tính toán từ **ALU** trong giai đoạn pipeline trước đó được lưu vào thanh ghi **EX/MEM** và đưa ra data tại địa chỉ đó nếu có tín hiệu đọc, hoặc sẽ ghi data được chuyển từ **rs2** (**DataB**) sang thanh ghi **EX/MEM** khi có tín hiệu ghi. Thanh ghi **MEM/WB** lưu lại dữ liệu được đọc ra từ memory và kết quả tính toán từ **ALU** được lấy ra ở thanh ghi **EX/MEM** để làm dữ liệu thực hiện giai đoạn pipeline tiếp theo là **write back**.

## Khối Register Write (WB)

Graphical user interface, text, application, chat or text message

Description automatically generated

*Hình 2.10 Register write block diagram*

### Interface signals

Bảng 2.6 Register Write port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **MEM\_WB\_mem\_to\_reg** | 1 | Input | Tín hiệu điều khiển chọn dữ liệu từ memory về thanh ghi của thanh ghi MEM/WB |
| **MEM\_WB\_mem\_data** | 32 | Input | Dữ liệu từ memory được lưu trong thanh ghi MEM/WB |
| **MEM\_WB\_alu\_out** | 32 | Input | Kết quả alu được lưu trong thanh ghi MEM/WB |
| **wb\_data** | 32 | Output | Dữ liệu write back |

### Chắc năng

Khối Register Write thực hiện việc quyết định xem ghi kết quả vào thanh ghi hay không dựa vào tín hiệu **reg\_write** nhận từ MEM/WB và quyết định xem ghi dữ liệu từ Data memory về thanh ghi hay giá trị được tính toán từ ALU chuyển sang.

### Kiến trúc khối Register Write

Chart

Description automatically generated with low confidence

*Hình 2.11 Register write architecture*

Hình 2.11 mô mô tả kiến trúc khối **Register Write** thực hiện giai đoạn pipeline write back. Bộ **mux2to1** nhận tín hiệu điều khiển **mem\_to\_reg** để thực hiện việc quyết định giữa đưa giá trị dữ liệu từ **Data memory** về thanh ghi hay kết quả tính toán của **ALU**.

## Khối Control Unit (control)

Diagram

Description automatically generated

*Hình 2.12 Control block diagram*

### Interface signals

Bảng 2.7 Control port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **IF\_ID\_inst** | 32 | Input | Mã lệnh được lưu trong thanh ghi IF/ID |
| **br\_eq** | 1 | Input | Kết quả so sánh data1 và data2 |
| **alu\_op** | 2 | Output | Opcode chọn phép toán |
| **alu\_src** | 1 | Output | Chọn chế độ địa chỉ trực tiếp |
| **branch** | 1 | Output | Cho phép nhảy đến địa chỉ offset |
| **pc\_src** | 1 | Output | Cho phép chọn PC+4 hoặc PC\_branch |
| **mem\_read** | 1 | Output | Cho phép memory đọc dữ liệu |
| **mem\_write** | 1 | Output | Cho phép memory ghi dữ liệu |
| **reg\_write** | 1 | Output | Cho phép ghi dữ liệu vào thanh ghi |
| **mem\_to\_reg** | 1 | Output | Tín hiệu cho phép write back |
| **IF\_flush** | 1 | Output | Cho phép xóa thanh ghi IF\_ID |
| **imm\_sel** | 3 | Output | Chọn kiểu cho Immidiate Generate |

### Chức năng

Khối **control unit** nhận 32 bit instruction làm đầu vào và dựa vào phần opcode của instruction rồi đưa ra các tín hiệu điều khiển tương ứng theo bảng sự thật Bảng 2.9

## Khối Hazard Detection Unit

Graphical user interface, text, application

Description automatically generated

*Hình 2.13 Hazard detection unit block diagram*

### Interface signals

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **ID\_EX\_mem\_read** | 1 | Input | Cho phép memory đọc dữ liệu lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Input | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **IF\_ID\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **IF\_ID\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **pc\_write** | 1 | Output | Cho phép thay đổi đầu ra PC |
| **IF\_ID\_write** | 1 | Output | Cho phép thanh ghi IF\_ID ghi dữ liệu mới |
| **ctrl\_sel** | 1 | Output | Chọn đầu ra control |

### Chức năng

Khối **Hazard detection unit** có nhiệm vụ xác định có hazard mà không thể sử dụng kĩ thuật forwarding để xử lí mà cần stall pipeline 1 chu kì clk. Khối **Hazard detection unit** xác định trường hợp cần stall như sau:

|  |
| --- |
| if (ID\_EX\_mem\_read) and (ID\_EX\_rd == IF\_ID\_rs1 or ID\_EX\_rd == IF\_ID\_rs2) then stall the pipline. |

## Khối Forwarding Unit

Diagram, table

Description automatically generated

*Hình 2.14 Forwarding unit block diagram*

### Interface signals

Bảng 2.8 Forwarding Unit port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **ID\_EX\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi ID/EX |
| **ID\_EX\_rd** | 5 | Input | Địa chỉ của thanh ghi đích lưu trong thanh ghi ID/EX |
| **IF\_ID\_rs1** | 5 | Input | Địa chỉ của thanh ghi 1 lưu trong thanh ghi IF/ID |
| **IF\_ID\_rs2** | 5 | Input | Địa chỉ của thanh ghi 2 lưu trong thanh ghi IF/ID |
| **MEM\_WB\_rd** | 5 | Input | Địa chỉ thanh ghi đích được lưu trong thanh ghi MEM/WB |
| **EX\_MEM\_rd** | 5 | Input | Địa chỉ thanh ghi đích được lưu trong thanh ghi EX/MEM |
| **EX\_MEM\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi EX/MEM |
| **MEM\_WB\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi MEM/WB |
| **ID\_EX\_reg\_write** | 1 | Input | Cho phép ghi dữ liệu vào thanh ghi lưu trong thanh ghi ID/EX |
| **branch** | 1 | Input | Cho phép nhảy đến địa chỉ offset |
| **forward\_a** | 2 | Output | Forwarding cho ALU khi có hazard tại thanh ghi rs1 EX |
| **forward\_b** | 2 | Output | Forwarding cho ALU khi có hazard tại thanh ghi rs2 EX |
| **forward\_comp1** | 2 | Output | Forwarding cho bộ branch compare khi có hazard tại thanh ghi rs1 ID |
| **forward\_comp2** | 2 | Output | Forwarding cho bộ branch compare khi có hazard tại thanh ghi rs2 ID |

### Chức năng

Khối **Forwarding unit** sẽ nhận các giá trị địa chỉ **rs1**, **rs2**, **rd** và các tín hiệu điều khiển từ các thanh ghi pipeline nhằm xác định các trường hợp các lệnh liên tiếp nhau có xảy ra hazard (data hazard, mem hazard, control hazard) để đưa ra các tín hiệu điều khiển cho các bộ **mux** nhằm forward các dữ liệu bị hazard tương ứng để có được kết quả tính toán chính xác nhất mà không cần phải stall pipeline quá nhiều chu kì clk.

Khối **Forwarding unit** sẽ xác định các trường hợp cần forward như sau:

|  |
| --- |
| if (MEM\_WB\_reg\_write  and (MEM\_WB\_reg\_write ≠ 0)  and not(EX\_MEM\_reg\_write and (EX\_MEM\_rd ≠ 0)  and (EX\_MEM\_rd = ID\_EX\_rs1))  and (MEM\_WB\_reg\_write = ID\_EX\_rs1)) forward\_a = 01  if (MEM\_WB\_reg\_write  and (MEM\_WB\_reg\_write ≠ 0)  and not(EX\_MEM\_reg\_write and (EX\_MEM\_rd ≠ 0)  and (EX\_MEM\_rd = ID\_EX\_rs2))  and (MEM\_WB\_reg\_write = ID\_EX\_rs2)) forward\_b = 01  if (branch && (ID\_EX\_rd != 0) && ID\_EX\_reg\_write && (IF\_ID\_rs1 == ID\_EX\_rd))  forward\_comp1 = 01;  else if (branch && (EX\_MEM\_rd != 0) && ~(ID\_EX\_reg\_write && (IF\_ID\_rs1 == ID\_EX\_rd)) && EX\_MEM\_reg\_write && (EX\_MEM\_rd == IF\_ID\_rs1))  forward\_comp1 = 10;  else forward\_comp1 = 00;  if (branch && (ID\_EX\_rd != 0) && ID\_EX\_reg\_write && (IF\_ID\_rs2 == ID\_EX\_rd))  forward\_comp2 = 01;  else if (branch && (EX\_MEM\_rd != 0) && ~(ID\_EX\_reg\_write && (IF\_ID\_rs2 == ID\_EX\_rd)) && EX\_MEM\_reg\_write && (EX\_MEM\_rd == IF\_ID\_rs2))  forward\_comp2 = 10;  else forward\_comp2 = 00; |

Bảng 2.9 Forwarding output ports

|  |  |  |
| --- | --- | --- |
| **Mux control** | **Source** | **Explaination** |
| **forward\_a = 00** | **ID/EX** | Đường input thứ nhất của **ALU** đến từ tệp thanh ghi. |
| **forward\_a = 10** | **EX/MEM** | Đường input thứ nhất của **ALU** được foward từ kết quả tính toán của **ALU** cho lệnh ngay trước đó. |
| **forward\_a = 01** | **MEM/WB** | Đường input thứ nhất của **ALU** được forward từ kết quả đọc data từ **memory** hoặc kết quả tính toán của **ALU** cho lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_b = 00** | **ID/EX** | Đường input thứ hai của **ALU** đến từ tệp thanh ghi. |
| **forward\_b = 10** | **EX/MEM** | Đường input thứ hai của **ALU** được foward từ kết quả tính toán của **ALU** cho lệnh ngay trước đó. |
| **forward\_b = 01** | **MEM/WB** | Đường input thứ hai của **ALU** được forward từ kết quả đọc data từ memory hoặc kết quả tính toán của **ALU** cho lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_comp1 = 00** | **Register** | Đường input thứ nhất của **branch compare** đến từ tệp thanh ghi. |
| **forward\_comp1 = 10** | **mem** or **EX/MEM** | Đường input thứ nhất của **branch compare** đến từ kết quả đọc data từ **memory** hoặc kết quả **ALU** của lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_comp1 = 01** | **ALU** | Đường input thứ nhất của **branch compare** đến từ kết quả tính toán **ALU** của lệnh trước đó. |
| **forward\_comp2 = 00** | **Register** | Đường input thứ hai của **branch compare** đến từ tệp thanh ghi. |
| **forward\_comp2 = 10** | **mem** or **EX/MEM** | Đường input thứ hai của **branch compare** đến từ kết quả đọc data từ **memory** hoặc kết quả **ALU** của lệnh trước đó cách lệnh đang thực hiện 1 lệnh. |
| **forward\_comp2 = 01** | **ALU** | Đường input thứ hai của **branch compare** đến từ kết quả tính toán **ALU** của lệnh trước đó. |

## Khối Top

Diagram, schematic

Description automatically generated

Hình . RISC-V pipeline top block diagram

Hình 2.15 mô tả toàn bộ các module con có trong khối top và các interface tương ứng được kết nối với nhau.

### Interface signals

Bảng 2.10 RISC pipeline top port description

| **Signal name** | **Width** | **Input/Output** | **Description** |
| --- | --- | --- | --- |
| **clk** | 5 | Input | Tín hiệu xung đồng hồ |
| **reset\_n** | 5 | Input | Tín hiệu reset không đồng bộ, tích cực mức thấp |

### Chức năng

Khối top là khối kết nối tất cả các module con của CPU, nhận xung **clk** và tín hiệu **reset** để hoạt động, chương trình cần CPU thực hiện sẽ được nạp sẵn vào bộ nhớ **Instruction memory**.

# MÔ PHỎNG VÀ KẾT QUẢ

Chương này trình bày các kết quả mô phỏng cho từng khối và toàn bộ thiết kế được triển khai bằng ngôn ngữ System Verilog trên phần mềm Questa Sim*.*

## Mô phỏng

### Các phần mềm hỗ trợ

#### Sublime Text

Sublime Text được biết đến là trình soạn thảo mã nguồn đa nền tảng độc quyền với giao diện của ngôn ngữ lập trình Python. Nó có nhiều ưu điểm như có khả năng tùy biến, tìm kiếm mạnh mẽ, tự động hoàn thành, đánh dấu cú pháp, nhanh và ổn định.

A screenshot of a computer

Description automatically generated with medium confidence

Hình . Giao diện phần mềm Sublime Text

#### Questa Sim

Questa Sim là một trình mô phỏng hỗ trợ biên dịch các thiết kế bằng Verilog HDL, VHDL và SystemC. Trình tự mô phỏng gồm 5 bước:

* Thu thập các file đầu vào (Verilog, VHDL, SysemC) và ánh xạ các thư viện.
* Phân tích kiểm tra code và biên dịch
* Tối ưu và tạo cơ sở dữ liệu đã biên dịch
* Mô phỏng
* Debug

Graphical user interface, application

Description automatically generated

Hình . Giao diện phần mềm mô phỏng Questa Sim

### Các trường hợp kiểm tra mô phỏng

* Kiểm tra các trường hợp instruction không có hazard.
* Kiểm tra các trường hợp instruction có data hazard.
* Kiểm tra các trường hợp instruction có mem hazard.
* Kiểm tra các trường hợp instruction có control hazard.
* Kiểm tra trường hợp reset bất thường khi mạch đang hoạt động.
* Kiểm tra quá trình reset, đảm bảo các đầu ra được thiết lập về đúng giá trị.

### Đầu vào

* Nạp sẵn các mã lệnh trong instruction memory:

|  |  |  |  |
| --- | --- | --- | --- |
| **Localparam** | **Address** | **Assembly Code** | **Instruction** |
| NONE | 32'h0000 | Nothing | Nothing |
| INST1 | 32'h0004 | add x10, x5, x1 | 0000000\_00001\_01010\_000\_01010\_0110011 |
| INST2 | 32'h0008 | sub x9, x4, x3 | 0100000\_00011\_00100\_000\_01001\_0110011 |
| INST3 | 32'h000C | addi x15, x1, -50 | 111111001110\_00001\_000\_01111\_0010011 |
| INST4 | 32'h0010 | lw x20, 10(x2) | 000000001010\_00010\_010\_10100\_0000011 |
| INST5 | 32'h0014 | sw x14, 8(x2) | 0000000\_01110\_00010\_010\_01000\_0100011 |
| INST6 | 32'h0018 | sub x12,x20,x8 | 0100000\_01000\_10100\_000\_01010\_0110011 |
| INST7 | 32'h001C | addi x8, x12, 10 | 000000001010\_01100\_000\_01000\_0010011 |
| INST8 | 32'h0020 | beq x6, x6, offset(8) | 0\_000000\_00110\_00110\_000\_0100\_0\_1100011 |
| INST9 | 32'h0024 | sw x24, 60(x5) | 0000001\_11001\_00111\_010\_11100\_0100011 |
| INST10 | 32'h0028 | sub x22, x30, x8 | 0100000\_01000\_11110\_000\_10110\_011011 |
| INST11 | 32'h002C | and x26, x24, x22 | 0000000\_10110\_11000\_111\_11010\_011011 |
| INST12 | 32'h0030 | addi x21, x21, 1 | 000000000001\_10101\_000\_10101\_0010011 |
| INST13 | 32'h0034 | lw x1, 120(x10) | 000001011000\_01010\_010\_00001\_0000011 |
| INST14 | 32'h0038 | add x1, x1, x21 | 0000000\_10101\_00001\_000\_00001\_0110011 |
| INST15 | 32'h003C | sw x1, 120(x10) | 0000011\_00001\_01010\_010\_11000\_0100011 |
| INST16 | 32'h0040 | add x15, x25, x14 | 0000000\_01110\_11001\_000\_01111\_0110011 |

* Các lệnh trên được tạo ra để kiểm tra các trường hợp có thể xảy ra khi thực hiện chương trình bằng kiến trúc vi xử lí RISC-V sử dụng kĩ thuật pipeline. Các trường hợp kiểm tra đã bao gồm data hazard, mem hazard và cuối cùng là control hazard.

## Kết quả và đánh giá

### Khối Instruction Fetch

Graphical user interface

Description automatically generated

*Hình 3.3 Mô phỏng timing diagram khối Instruction Fetch*

Kết quả đầu ra hoạt động đúng logic khi thực hiện các lệnh được nạp sẵn vào **Instruction Memory**.

### Khối Instruction Decode

A screenshot of a computer

Description automatically generated with medium confidence

*Hình 3.4 Mô phỏng timing diagram khối Instruction Decode*

Kết quả đầu ra của khối **Instruction Decode** hoạt động đúng với logic của thiết kế khi thực hiện các lệnh được nạp sẵn vào **Instruction Memory**.

### Khối Execute

A picture containing timeline

Description automatically generated

*Hình 3.5 Mô phỏng timing diagram khối Execute*

Kết quả đầu ra của khối **Execute** hoạt động đúng với logic của thiết kế khi thực hiện các lệnh được nạp sẵn.

### Khối Memory Access

A screenshot of a computer

Description automatically generated with medium confidence

*Hình 3.6 Mô phỏng timing diagram khối Memory Access*

Kết quả đầu ra của khối **Memory Access** hoạt động đúng với logic của thiết kế khi thực hiện các lệnh được nạp sẵn.

### Khối Register Write



*Hình 3.7 Mô phỏng timing diagram khối Register Write*

Kết quả đầu ra của khối **Register Write** hoạt động đúng với logic của thiết kế.

### Khối Control

Graphical user interface

Description automatically generated

*Hình 3.8 Mô phỏng timing diagram khối Control*

Các tín hiệu điều khiển **control** gửi đến **datapath** đúng với logic thiết kế khi thực hiện các lệnh được nạp sẵn trong **Instruction Memory**.

### Khối Forwarding Unit

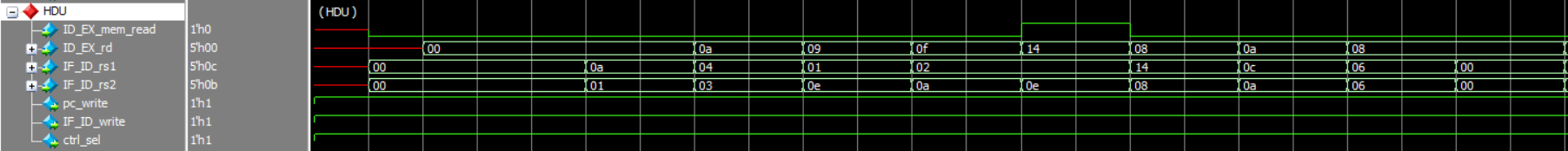
Diagram

Description automatically generated with medium confidence

*Hình 3.9 Mô phỏng timing diagram khối Forwarding Unit*

Tín hiệu đầu ra của module khối **forwarding unit** theo các kích thích đầu vào tương ứng, từ hình 3.9 cho thấy khối **forwarding unit** hoạt động đúng logic đã định nghĩa.

### Khối Hazard Detection Unit



*Hình 3.10 Mô phỏng timing diagram khối Hazard Detection Unit*

Hình 3.10 mô tả tín hiệu đầu ra của module khối **hazard detection unit** theo các kích thích đầu vào tương ứng, từ wave form cho thấy khối này hoạt động đúng logic đã định nghĩa.

### Khối Register

A picture containing window

Description automatically generated

Hình . Mô phỏng 32 thanh ghi

Dựa vào hình trên có thể thấy giá trị của tệp thanh ghi đã được thay đổi sau khi thực hiện chương trình. Tín hiệu kể từ khi lệnh được nạp vào CPU (tương ứng với giá trị **pc** thay đổi) thì sau 5 chu kì tiếp theo giá trị thanh ghi được thay đổi đúng như kết quả tính toán từ trước cho các lệnh nạp sẵn theo kịch bản cho trước.

Khi có tín hiệu **reset** bất thường, CPU được reset về trạng thái ban đầu, giá trị thanh ghi được khởi tạo tương ứng với địa chỉ và **pc** được reset về 0, chương trình trong **Instruction Memory** được chạy lại từ đầu và vẫn đũng so với kết quả đã tính toán trước được nạp sẵn.

### Data Memory

A computer screen capture

Description automatically generated with low confidence

Hình . Mô phỏng Data Memory

Hình 3.12 mô tả sự thay đổi của dữ liệu bên trong **Data Memory**, dữ liệu thay đổi đúng như tính toán của các câu lệnh được nạp sẵn trong **Instruction Memory**.

## Đánh giá chung

Kết quả thu được đúng với yêu cầu xảy ra hazard, đã xử lý stalling và forwarding. Tuy nhiên, số lượng lệnh nạp vào Instructioin Memory vẫn chưa lớn và quá trình chạy vẫn chưa đạt được mức độ cao nhất của pipeline cho RISC-V.

# KẾT LUẬN

Qua tìm hiểu cùng với sự hướng dẫn của TS. Tạ Thị Kim Huệ, em đã hoàn thành báo cáo của bài tập lớn. Trong quá trình làm báo cáo em đã hiểu và nắm rõ hơn cơ chế làm việc xử lý mã lệnh của RISC-V, cũng như cơ chế RISC-V pipeline, những ưu và nhược điểm của cơ chế pipeline và những phương án khắc phục. Em cũng đã thực hiện được một chương trình sử dụng ngôn ngữ System Verilog được mô phỏng trên phần mềm Questa Sim, và cho ra kết quả hoạt động đúng với yêu cầu đầu vào đầu ra. Tuy vẫn chưa phải là RISC-V pipeline nhưng qua quá trình làm bài tập lớn thì em cũng đã hiểu rõ hơn cách hoạt động của RISC-V, có kiến thức về RISC-V pipeline và có thể hoàn thành được RISC-V pipeline trong thời gian tiếp theo.

Với lượng kiến thức có hạn và thời gian chưa đủ nên em vẫn chưa đạt được mục tiêu cao nhất của bài tập lớn đề ra, nhưng em hy vọng những gì em tìm hiểu được là kiến thức hữu ích cho em, cũng như cho công việc sau này. Bài báo cáo của em không thể tránh khỏi những sai sót không mong muốn, mong đưa ra những nhận xét để em hoàn thiện tốt hơn và rút ra kinh nghiệm cho những bài tập lớn tiếp theo.

Em xin cảm ơn sự giảng dạy và hướng dẫn tận tình của cô!

PHỤ LỤC

Link source code in Github: <https://github.com/huynam1802/riscv_pipeline>

TÀI LIỆU THAM KHẢO

1. <https://en.wikipedia.org/wiki/RISC-V> Ngày truy cập cuối cùng: 31/12/2021
2. Slide: RISC – V 2021. TS. Tạ Thị Kim Huệ
3. Computer Organization and Design Risc – V The Hardware Software Interface by David A. Patterson and John L. Hennessy
4. <https://github.com/ultraembedded/riscv>