|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **TRƯỜNG ĐIỆN - ĐIỆN TỬ**  logo_128  BÁO CÁO BÀI TẬP LỚN  **KIẾN TRÚC MÁY TÍNH**  **Đề tài:**  **THIẾT KẾ BỘ XỬ LÝ RISCV ĐƯỜNG ỐNG BẰNG VERILOG VÀ MÔ PHỎNG CHẠY CHƯƠNG TRÌNH SẮP XẾP NỔI BỌT**   |  |  |  | | --- | --- | --- | | Họ tên sinh viên | MSSV | Lớp – Khóa | | Nguyễn Việt Thi | 20182798 | Điện tử 10-K63 | | Vũ Tiến Thịnh | 20182809 | Điện tử 11-K63 | | Đỗ Thành Đạt | 20182411 | Điện tử 11-K63 |   Giảng viên hướng dẫn: PGS TS. NGUYỄN ĐỨC MINH  Hà Nội, 8-2022 |

**LỜI NÓI ĐẦU**

Hiện nay, công nghệ phát triển ngày càng nhanh, máy tính cũng yêu cầu phải làm được nhiều việc hơn với tốc độ xử lý nhanh và hiệu suất cao hơn. Hàng loạt các cấu trúc x86, x64 theo đà phát triển của công nghệ đang dần trở nên lỗi thời và kém hiệu quả khi sử dụng. Để theo kịp xu thế đó thì bộ xử lý RISC-V đã được ra đời nhằm đáp ứng yêu cầu ngày càng cao của con người. Kiến trúc này tuy ra đời chưa lâu nhưng nó đã giúp đáp ứng được những yêu cầu cao trong quá trình xử lý của máy tính , đồng thời khắc phục những hạn chế, yếu điểm của các kiến trúc cũ.

Chương trình học môn Kiến trúc máy tính (ET4041) RISC-V Processor Design là một phần quan trọng trong RISC-V nhằm giúp chúng em hiểu rõ về quá trình hoạt động khi thực hiện các lệnh. Trong bài báo cáo này, chúng em có thực hiện thiết kế bộ xử lý RISC-V đường ống bằng ngôn ngữ VERILOG và mô phỏng chạy chương trình sắp xếp nổi bọt trên phần mềm Questasim.

Nhóm chúng em xin chân thành cảm ơn PSG.TS Nguyễn Đức Minh đã tận tâm giảng dạy và cung cấp kiến thức trong quá trình học tập môn Kiến trúc máy tính. Qua đó giúp chúng em có thêm kiến thức để thực hiện đề tài một cách tốt nhất.

**MỤC LỤC**

[DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT i](#_Toc112186626)

[DANH MỤC HÌNH VẼ ii](#_Toc112186627)

[DANH MỤC BẢNG BIỂU iv](#_Toc112186628)

[CHƯƠNG 1. GIỚI THIỆU CHUNG 1](#_Toc112186629)

[1.1 Giới thiệu chung 1](#_Toc112186630)

[1.2 Pipeline 1](#_Toc112186631)

[1.3 Hazard trong pipeline 3](#_Toc112186632)

[CHƯƠNG 2. THIẾT KẾ RISC-V 5-STAGE 32BIT PIPELINE 4](#_Toc112186633)

[2.1 Tổng quan hệ thống 4](#_Toc112186634)

[2.1.1 Mô tả hệ thống 4](#_Toc112186635)

[2.1.2 Sơ đồ khối 5](#_Toc112186636)

[2.2 Mô tả chi tiết 3](#_Toc112186637)

[2.2.1 Instruction memory 3](#_Toc112186638)

[2.2.2 Data Memory 4](#_Toc112186639)

[2.2.3 Imm\_gen 5](#_Toc112186640)

[2.2.4 PC 6](#_Toc112186641)

[2.2.5 Branch Comparator 7](#_Toc112186642)

[2.2.6 Intruction decode 8](#_Toc112186643)

[2.2.7 Register 9](#_Toc112186644)

[2.2.8 Control 11](#_Toc112186645)

[2.2.9 Hazard Detection 16](#_Toc112186646)

[2.2.10 Forwarding 17](#_Toc112186647)

[2.2.11 Control Mux 21](#_Toc112186648)

[2.2.12 ALU control 23](#_Toc112186649)

[2.2.13 Adder 24](#_Toc112186650)

[2.2.14 IF/ID 25](#_Toc112186651)

[2.2.15 EX\_MEM 26](#_Toc112186652)

[2.2.16 MEM\_WB 28](#_Toc112186653)

[2.2.17 ID/EX 29](#_Toc112186654)

[2.2.18 MUX21 32](#_Toc112186655)

[2.2.19 MUX31 33](#_Toc112186656)

[CHƯƠNG 3. KIỂM THỬ (VERIFICATION) 34](#_Toc112186657)

[3.1 Test case 34](#_Toc112186658)

[3.2 Tạo đầu vào vào kiểm tra các test case 34](#_Toc112186659)

[3.2.1 Quá trình reset 34](#_Toc112186660)

[3.2.2 Các trường hợp instruction không có hazard 35](#_Toc112186661)

[3.2.3 Trường hợp instruction có structural hazard 36](#_Toc112186662)

[3.2.4 Trường hợp instruction có data hazard 38](#_Toc112186663)

[3.2.5 Trường hợp instruction có control hazard 41](#_Toc112186664)

[3.2.6 Kiểm tra lệnh JAL 43](#_Toc112186665)

[CHƯƠNG 4. MÔ PHỎNG CHẠY THUẬT TOÁN SẮP XẾP NỔI BỌT 44](#_Toc112186666)

[4.1 Triển khai thuật toán nổi bọt bằng assembly và chuyển sang ngôn ngữ máy 44](#_Toc112186667)

[4.2 Nạp mã lệnh vào hệ thống và mô phỏng 47](#_Toc112186668)

[CHƯƠNG 5. KẾT LUẬN 50](#_Toc112186669)

[TÀI LIỆU THAM KHẢO 51](#_Toc112186670)

# DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

|  |  |
| --- | --- |
| ISA | Instruction Set Architecture |
| RISC | Reduced Instruction Set Computer |
| ALU | Arithmetic logic unit |
|  |  |

# DANH MỤC HÌNH VẼ

[Hình 1.1 Single cycle versus pipelined (1) 2](#_Toc112186671)

[Hình 1.2 Single cycle versus pipelined (2) 2](#_Toc112186672)

[Hình 2.1. Sơ đồ khối tổng quát của hệ thống RISC-V 5-stage pipeline 32 bit 2](#_Toc112186673)

[Hình 2.2. Sơ đồ khối của khối instruction memory 3](#_Toc112186674)

[Hình 2.3. Sơ đồ khối của khối data memory 4](#_Toc112186675)

[Hình 2.4. Cấu trúc của instruction trong RISC-V 5](#_Toc112186676)

[Hình 2.5. Ví dụ về tạo giá trị immediate của lệnh R 5](#_Toc112186677)

[Hình 2.6. Sơ đồ khối của khối imm\_gen 5](#_Toc112186678)

[Hình 2.7. Sơ đồ khối của khối PC 6](#_Toc112186679)

[Hình 2.8. Sơ đồ khối của khối branch comparator 7](#_Toc112186680)

[Hình 2.9. Sơ đồ khối của khối Intruction decode 9](#_Toc112186681)

[Hình 2.10. Sơ đồ khối của khối register 10](#_Toc112186682)

[Hình 2.11. Sơ đồ khối của khối control 11](#_Toc112186683)

[Hình 2.12. Sơ đồ khối của khối hazard dectection. 16](#_Toc112186684)

[Hình 2.13. Sơ đồ khối của khối forwarding 17](#_Toc112186685)

[Hình 2.14. Mô phỏng quá trình EX haxzard 18](#_Toc112186686)

[Hình 2.15. Mô phỏng quá trình MEM hazard 18](#_Toc112186687)

[Hình 2.16 Quá trình forward đối với trường hợp hazard cho lệnh loại B 19](#_Toc112186688)

[Hình 2.17. Quá trình forward đối với trường hợp hazard cho lệnh loại B 19](#_Toc112186689)

[Hình 2.18. Sơ đồ khối của khối control mux 21](#_Toc112186690)

[Hình 2.19. Sơ đồ khối của khối ALU 24](#_Toc112186691)

[Hình 2.20. Sơ đồ khối của adder 24](#_Toc112186692)

[Hình 2.21. Sơ đồ khối của khối IF/ID 26](#_Toc112186693)

[Hình 2.22. Sơ đồ khối của khối EX/MEM 27](#_Toc112186694)

[Hình 2.23. Sơ đồ khối của khối MEM/WB 28](#_Toc112186695)

[Hình 2.24. Sơ đồ khối của khối ID/EX 30](#_Toc112186696)

[Hình 2.25. Sơ đồ khối của mux21 32](#_Toc112186697)

[Hình 2.26. Sơ đồ khối của khối mux31 33](#_Toc112186698)

[Hình 3.1. Kết quả mô phỏng reset của hệ thống 34](#_Toc112186699)

[Hình 3.2. Giá trị của của các thanh ghi sau quá trình mô phỏng 35](#_Toc112186700)

[Hình 3.3. Giá trị của các ô nhớ trong data memory sau quá trình mô phỏng 36](#_Toc112186701)

[Hình 3.4. Wave của quá trình mô phỏng 36](#_Toc112186702)

[Hình 3.5. Mô tả cùng ghi vào memory 37](#_Toc112186703)

[Hình 3.6. Kết quả của các thanh ghi ở 2 thời điểm 37](#_Toc112186704)

[Hình 3.7. Mô tả cùng đọc ghi vào register 38](#_Toc112186705)

[Hình 3.8. Kết quả của thanh ghi và data memory ở 2 thời điểm 38](#_Toc112186706)

[Hình 3.9. Mô phỏng quá trình EX haxzard 39](#_Toc112186707)

[Hình 3.10. Kết quả của các thanh ghi và data memory ở 3 thời điểm 39](#_Toc112186708)

[Hình 3.11. Mô phỏng quá trình MEM hazard 40](#_Toc112186709)

[Hình 3.12. Phát hiện có hazard 40](#_Toc112186710)

[Hình 3.13. Kết quả của MEM hazard 40](#_Toc112186711)

[Hình 3.14 Quá trình forward đối với trường hợp hazard cho lệnh loại B 41](#_Toc112186712)

[Hình 3.15. Kết quả sau quá trình mô phỏng 41](#_Toc112186713)

[Hình 3.16. Quá trình forward đối với trường hợp hazard cho lệnh loại B 42](#_Toc112186714)

[Hình 3.17. Kết quả của quá trình mô phỏng 42](#_Toc112186715)

[Hình 3.18. Mô tả chèn 2 lệnh NOP trên questasim 43](#_Toc112186716)

[Hình 3.19. Kết quả sau mô phỏng 43](#_Toc112186717)

[Hình 4.1. Khởi tạo instruction memory 47](#_Toc112186718)

[Hình 4.2. Dữ liệu trong data memory ở thời điểm thực hiện xong các lệnh load dữ liệu khởi tạo 48](#_Toc112186719)

[Hình 4.3. Dữ liệu trong data memory ở thời điểm thực hiện xong sắp xếp 48](#_Toc112186720)

# DANH MỤC BẢNG BIỂU

[Bảng 2.1. Tính năng hỗ trợ của hệ thông 4](#_Toc112186721)

[Bảng 2.2. Tín hiệu vào ra của hệ thống 5](#_Toc112186722)

[Bảng 2.3. Chân vào ra của instruction memory 3](#_Toc112186723)

[Bảng 2.4. Chân vào ra của khối data memory 4](#_Toc112186724)

[Bảng 2.5. Chân vào ra của immediate generation 6](#_Toc112186725)

[Bảng 2.6. Định giá các giá trị cảu immediate select 6](#_Toc112186726)

[Bảng 2.7. Chân vào ra của khối PC 7](#_Toc112186727)

[Bảng 2.8. Chân vào ra của khối branch comparator 8](#_Toc112186728)

[Bảng 2.9. Chân vào ra của khối Intruction decode 9](#_Toc112186729)

[Bảng 2.10. Chân vào ra của khối register 10](#_Toc112186730)

[Bảng 2.11. Chân vào ra của khối control 11](#_Toc112186731)

[Bảng 2.12. Các tín hiệu điều khiển ứng với các lệnh 13](#_Toc112186732)

[Bảng 2.13. Chân vào ra của khối Hazard detection 16](#_Toc112186733)

[Bảng 2.14. Mô tả chọn đường dữ liệu forward 19](#_Toc112186734)

[Bảng 2.15. Chân vào ra của khối forwarding 20](#_Toc112186735)

[Bảng 2.16. Chân vào ra của khối control mux 22](#_Toc112186736)

[Bảng 2.17. Giá trị ứng với từng chức năng của ALU 23](#_Toc112186737)

[Bảng 2.18. Chân tín hiệu vào ra của khối ALU 24](#_Toc112186738)

[Bảng 2.19. Chân vào ra của khối adder 25](#_Toc112186739)

[Bảng 2.20. Chân vào ra của khối IF/ID 26](#_Toc112186740)

[Bảng 2.21. Chân vào ra của khối EX/MEM 27](#_Toc112186741)

[Bảng 2.22. Chân vào ra của khối MEM/WB 29](#_Toc112186742)

[Bảng 2.23. Chân vào ra của khối ID/EX 30](#_Toc112186743)

[Bảng 2.24. Chân vào ra của khối mux21 32](#_Toc112186744)

[Bảng 2.25. Chân vào ra của khối mux31 33](#_Toc112186745)

# GIỚI THIỆU CHUNG

## Giới thiệu chung

RISC-V là một kiến trúc tập lệnh tiêu chuẩn mở (ISA) (*Instruction Set Architecture*) phần cứng mã nguồn mở dựa trên kiến trúc tập lệnh máy tính với tập lệnh đơn giản hóa *Reduced Instruction Set Computer* (RISC).Không giống như hầu hết các ISA khác, thiết kế RISC – V ISA được cung cấp theo *Open Source Licenses* không yêu cầu phí sử dụng.

Các tính năng đáng chú ý của RISC – V bao gồm *load-store architecture*, các bit mẫu để đơn giản hóa ghép kênh trong CPU, dấu phẩy IEEE 754. Thiết kế trung lập về mặt kiến trúc và đặt *most-significant* bits tại một vị trí cố định để tăng tốc độ *sign extension.* Kiến trúc tập lệnh RISC – V cho nhiều mục đích sử dụng, nó có thể thay đổi kích thước bit mã hóa và có thể mở rộng để luôn có thể thêm nhiều bit mã hóa hơn. Nó hỗ trợ các thanh ghi có chiều dài 32, 64, 128 bits và nhiều tập hợp con. Với mỗi thanh ghi có kích thước khác nhau thì có cách đánh địa chỉ khác nhau khi sử dụng. Đặc biệt kiến trúc tập lệnh phù hợp cho tất cả hệ thống máy tính, hệ thống nhúng và siêu máy tính.

## Pipeline

Pipelining là kỹ thuật mà nhiều lệnh được thực hiện theo dạng nạp chồng (overlap). Kỹ thuật này được sử dụng phổ biến trong các kiến trúc CPU hiện nay. Quá trình thực hiện một lệnh trong RISC – V cổ điển bao gồm 5 bước:

1. Nạp lệnh từ bộ nhớ
2. Giải mã lệnh và đọc các thanh ghi cần
3. Thực thi các phép tính hoặc tính toán địa chỉ
4. Truy xuất các toán hạng trong bộ nhớ
5. Ghi kết quả cuối vào thanh ghi

Dưới đây là Hình 1.1 mô tả so sánh giữa single – cycle (nonpipelined) và pipeline.

Table

Description automatically generated with medium confidence

Hình 1.1 Single cycle versus pipelined (1)

Table

Description automatically generated

Hình 1.2 Single cycle versus pipelined (2)

Trong trường hợp lý tưởng : khi mà các công đoạn pipeline hoàn toàn bằng nhau thì thời gian giữa hai lệnh liên tiếp được thực thi trong pipeline bằng:

Trong thực tế : Các công đoạn thực tế không bằng nhau, việc áp dụng pipeline phải chọn công đoạn dài nhất để làm một chu kỳ pipeline. Vì vậy trong ví dụ trên thời gian liên tiếp giữa hai lệnh pipeline là 200ps Dựa vào Hình 1.1, ta sử dụng chung một kiến trúc phần cứng, thời gian khi thực hiện các lệnh không sử dụng pipeline là 3 x 800 = 2400 ps, nhưng khi sử dụng pipeline là 3 x 200 = 600 ps. Như vậy khi áp dụng pipeline tăng tốc gấp 4 lần so với không sử dụng pipeline.

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.

Kỹ thuật pipeline không giúp giảm thời gian thực thi của từng lệnh riêng lẻ mà giúp giảm tổng thời gian thực thi của từng đoạn lệnh/ chương trình chứa nhiều lệnh (từ đó giúp thời gian trung bình của mỗi lệnh giảm). Việc giúp giảm thời gian thực thi cho nhiều lệnh vô cùng quan trọng, vì chương trình chạy trong thực tế thông thường lên đến hàng tỉ lệnh.

## 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.

Việc loại bỏ hazard sẽ được trình bày chi tiết ở CHƯƠNG 2.

# THIẾT KẾ RISC-V 5-STAGE 32BIT PIPELINE

Chương này nêu tổng quan về các khối có trong RISC-V 5-stages pipeline 32-bit và ý nghĩa của chúng, cũng như sơ đồ kết nối giữa các khối để tạo ra một thiết kế hoàn chỉnh.

## Tổng quan hệ thống

### Mô tả hệ thống

Bảng 2.1 mô tả các chức năng được hỗ trợ của hệ thống RISC-V. Thiết kế có tham khảo ở [1].

Bảng 2.1. Tính năng hỗ trợ của hệ thông

|  |  |
| --- | --- |
| Tính năng | RISC-V CPU |
| ISA | RISC-V |
| Pipelining | 5 stages |
| Data forwarding | Có |
| Instruction | 32 bit |
| Hazard detection | Data hazard (MEM-Hazard, EX-Hazard), Structural harzard, Control hazard |

Các lệnh được hỗ trợ:

* Lệnh loại R: add sub xor or and sll srl sra slt sltu
* Lệnh loại I: addi xori ori andi slli srli srai slti sltiu lw
* Lệnh loại S: sw
* Lệnh Loại B: beq bne blt bge bltu bgeu
* Lệnh loại J: jal (jump and link)
* Lệnh U: lui (Load Upper Imm)

### Sơ đồ khối

Hình 2.1 mô tả sơ đồ khối của hệ thống bao gồm datapath và control logic trong RISC-V 32 pipeline để thực hiện các chức năng đã nêu rõ ở mục 2.1.1.

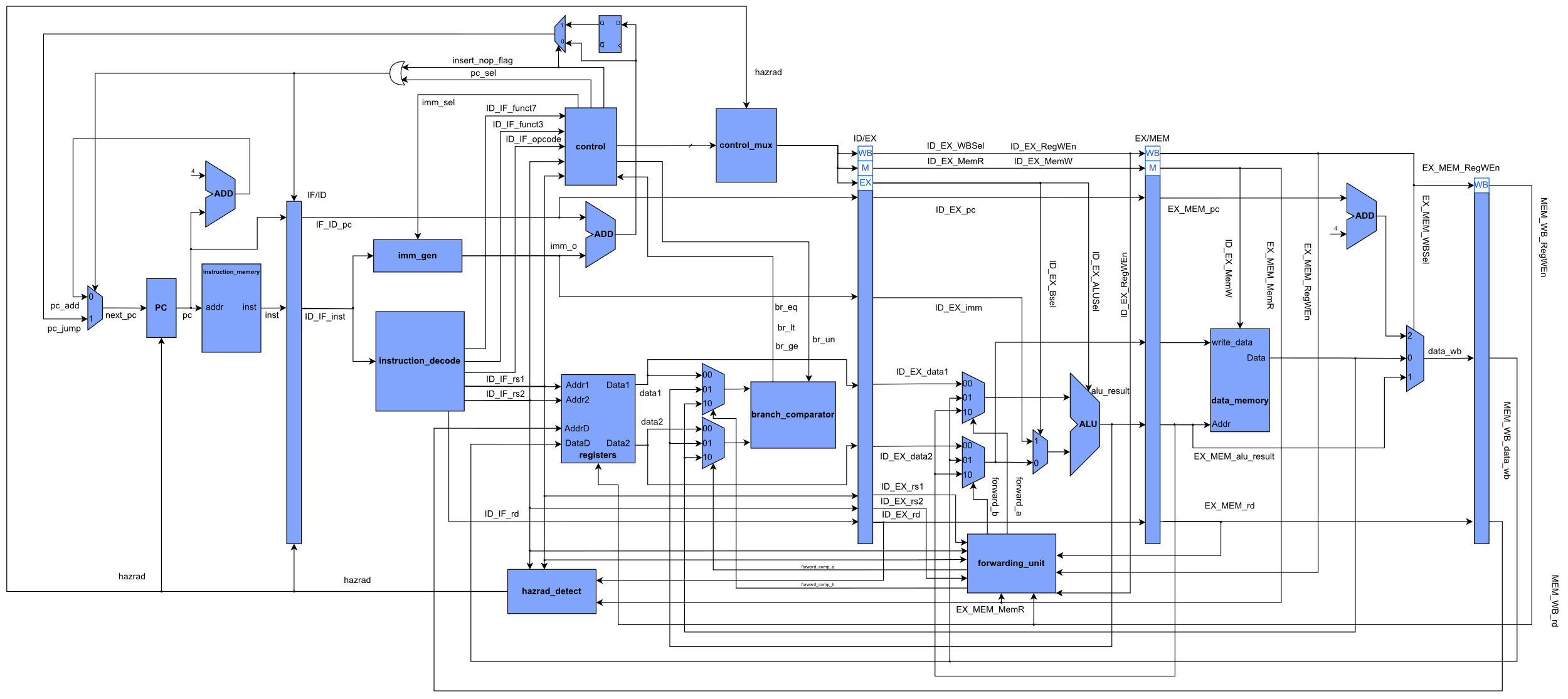
Bảng 2.2 liệt kê các tín hiệu vào ra của hệ thống RISC-V pipeline 32bit.

Bảng 2.2. Tín hiệu vào ra của hệ thống

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

Hệ thống đã triển khai được lệnh loại B với 1 chu kỳ stall (1 lệnh NOP) và lệnh JAL với 2 chu kỳ stall.

Khối top là khối kết nối tất cả các module con của CPU, nhận xung clk và tín hiệu reset (Bảng 2.2) để hoạt động, chương trình cần CPU thực hiện sẽ được nạp sẵn vào bộ nhớ Instruction memory.



Hình 2.1. Sơ đồ khối tổng quát của hệ thống RISC-V 5-stage pipeline 32 bit

## Mô tả chi tiết

### Instruction memory

Instruction memory có nhiệm vụ lưu trữ các lệnh được nạp vào hệ thống. Từ đó, nó sẽ trả về lệnh có địa chỉ tương ứng (PC). Hình 2.2 mô tả sơ đồ tổng quát của instruction memory.

Diagram

Description automatically generated

Hình 2.2. Sơ đồ khối của khối instruction memory

Bảng 2.3 chỉ rõ chức năng, độ rộng của các tín hiệu có trong instruction memory.

Để tránh trường hợp structural harzard, khối instruction memory sẽ được tách riêng biệt với khối data memory.

Bảng 2.3. Chân vào ra của instruction memory

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **pc\_i** | 32 | Input | Tín hiệu PC đầu vào |
| **inst\_o** | 32 | Output | Lệnh trả ra với PC tương ứng |

Ở đây, để nạp lệnh vào hệ thống, nhóm đang sử dụng lệnh readmemh trong khối initial.

### Data Memory

Khối data memory (Hình 2.3) thực hiện chức năng lưu trữ dữ liệu (RAM). Các tín hiệu cho phép đọc ghi (mem\_read, mem\_write) sẽ điều khiển dữ liệu vào ra.

Tương tự khối instruction memory, để tránh trường hợp structural harzard, khối data memory sẽ tách riêng biệt.

Diagram

Description automatically generated

Hình 2.3. Sơ đồ khối của khối data memory

Bảng 2.4 mô tả chức năng của các tín hiệu vào ra của khối.

Bảng 2.4. Chân vào ra của khối data memory

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **op\_addr** | 5 | Input | Địa chỉ đọc dữ liệu từ bên ngoài |
| **addr\_i** | 32 | Input | Địa chỉ đọc ghi |
| **data\_i** | 32 | Input | Dữ liệu ghi vào RAM |
| **mem\_write\_i** | 1 | Input | Tín hiệu cho phép ghi |
| **mem\_read\_i** | 1 | Input | Tín hiệu cho phép đọc |
| **data\_o** | 32 | Output | Dữ liệu đọc từ RAM |

### Imm\_gen

Khối Immediate generation có chức năng nhận diện các loại lệnh động thời sắp xếp kết hợp các bit loại immediate của các loại lệnh đó theo cấu trúc như Hình 2.4, đồng thời mở rộng bit dấu để tạo ra đầu ra 32 bit.

Hình 2.5 là một ví dụ về tạo giá trị immediate của lệnh loại R.

Table

Description automatically generated

Hình 2.4. Cấu trúc của instruction trong RISC-V

Diagram

Description automatically generated

Hình 2.5. Ví dụ về tạo giá trị immediate của lệnh R

Hình 2.6 mô tả sơ đồ tổng quát của khối immediate generation

Diagram

Description automatically generated with medium confidence

Hình 2.6. Sơ đồ khối của khối imm\_gen

Bảng 2.5 mô tả chức năng, độ rộng các tín hiệu vào ra của khối immediate generation.

Bảng 2.5. Chân vào ra của immediate generation

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **inst\_i** | 32 | Input | Lệnh được đưa vào |
| **imm\_sel\_i** | 3 | Input | Tín hiệu xác định loại lệnh |
| **imm\_o** | 32 | Output | Giá trị immediate đã được tách và mở rộng dấu |

Các giá trị đầu vào imm\_sel được định nghia như Bảng 2.6.

Bảng 2.6. Định giá các giá trị cảu immediate select

|  |  |
| --- | --- |
| Tên | Giá trị |
| ImmSelI | 3'b000 |
| ImmSelS | 3'b001 |
| ImmSelB | 3'b010 |
| ImmSelJ | 3'b011 |
| ImmSelU | 3'b100 |
| ImmSelR | 3'b111 |

### PC

Khối PC đưa ra giá trị PC phù hợp ở từng thời điểm để đưa vào khối instruction memory (Hình 2.7).

A picture containing text, clock, screenshot

Description automatically generated

Hình 2.7. Sơ đồ khối của khối PC

Ở các trường hợp xuất hiện hazard, giá trị PC sẽ được giữ nguyên như ở chu kỳ trước. Giá trị PC đưa vào là giá trị PC + 4 hoặc giá trị nhảy PC + imm.

Bảng 2.7 chỉ rõ chức năng, độ rộng của các tín hiệu có trong khối PC.

Bảng 2.7. Chân vào ra của khối PC

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **hazardpc\_i** | 1 | Input | Tín hiệu chỉ ra có hazard |
| **pc\_i** | 32 | Output | Giá trị PC tiếp theo |
| **pc\_out** | 32 | Output | Giá trị PC đưa vào khối instruction memory |

### Branch Comparator

Khối Branch comparator nhận 2 dữ liệu đầu vào DataA và DataB, đồng thời so sánh 2 giá trị đó. Ở đây chúng ta có thể chọn so sánh có dấu hay không có dấu bằng tín hiệu BrUn\_i (Hình 2.8).

Graphical user interface, application, PowerPoint

Description automatically generated

Hình 2.8. Sơ đồ khối của khối branch comparator

Bảng 2.8 chỉ rõ chức năng, độ rộng của các tín hiệu có trong khối branch comparator. Dưới đây là mức hành vi của khối:

|  |
| --- |
| assign BrEq\_o = (DataA == DataB) ? 1'b1 : 1'b0;  always @(\*) begin  if (BrUn\_i) begin  BrLt\_o = (DataA < DataB) ? 1'b1 : 1'b0;  end  else begin  BrLt\_o = ($signed(DataA) < $signed(DataB)) ? 1'b1 : 1'b0;  end  end  always @(\*) begin  if (BrUn\_i) begin  BrGe\_o = (DataA >= DataB) ? 1'b1 : 1'b0;  end  else begin  BrGe\_o = ($signed(DataA) >= $signed(DataB)) ? 1'b1 : 1'b0;  end  end |

Bảng 2.8. Chân vào ra của khối branch comparator

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **BrUn\_i** | 1 | Input | Tín hiệu chọn so sánh có dấu hay không có dấu.   * 1: có dấu * 0: không dấu |
| **DataA** | 32 | Input | Dữ liệu đầu vào A |
| **DataB** | 32 | Input | Dữ liệu đầu vào B |
| **BrEq\_o** | 1 | Output | Đầu ra biểu thị 2 đầu vào bằng nhau |
| **BrLt\_o** | 1 | Output | Đầu ra biểu thị A < B |
| **BrGe\_o** | 1 | Output | Đầu ra biểu thị A >= B |

### Intruction decode

Intruction decode (Hình 2.9) có chức năng giải mã lệnh đưa vào. Từ đó, tách ra các trường tương ứng với các loại lệnh. Các giá trị đầu ra (opcode, rd, rs1, rs2, function3, function7) sẽ được đưa vào khối control để tính toán các tín hiệu điều khiển hệ thống và làm giá trị tính toán cho các stage tiếp theo.

Diagram

Description automatically generated

Hình 2.9. Sơ đồ khối của khối Intruction decode

Các giá trị đầu ra ở các loại lệnh sẽ khác nhau, như Hình 2.4.

Bảng 2.9. Chân vào ra của khối Intruction decode

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **inst\_i** | 32 | Input | Mã lệnh đưa vào để giải mã |
| **rd\_o** | 5 | output | Giá trị rd |
| **rs1\_out** | 5 | output | Giá trị rs1 |
| **rs2\_out** | 5 | output | Giá trị rs2 |
| **funct3\_o** | 3 | output | Giá trị function 3 |
| **funct7\_o** | 7 | output | Giá trị function 7 |
| **opcode\_o** | 7 | output | Giá trị opcode |

Tín hiệu vào ra của khối được mô tả trong Bảng 2.9.

### Register

Khối register là nơi lưu trữ các giá trị của các thanh ghi. Đối với kiến trúc RISC-V, chúng ta sẽ triển khai 32 thanh ghi () có độ rộng là 32 bít với thanh ghi sẽ mặc định là giá trị 0 và không bao giờ thay đổi.

Để tránh structural hazard, trong 1 chu kỳ, ở sườn dương của clk sẽ là quá trình đọc giá trị từ thanh ghi có địa (), còn ở sườn âm sẽ là quá trình ghi vào địa chỉ .

Diagram

Description automatically generated

Hình 2.10. Sơ đồ khối của khối register

Hình 2.10 mô tả sơ đồ tổng quát của khối register.

Bảng 2.10. Chân vào ra của khối register

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **RSaddr\_i** | 5 | Input | Địa chỉ đọc thanh ghi rs1 |
| **RTaddr\_i** | 5 | Input | Địa chỉ đọc thanh ghi rs2 |
| **RDaddr\_i** | 5 | Input | Địa chỉ của thanh ghi rd |
| **RDdata\_i** | 32 | Input | Giá trị ghi vào thanh ghi có địa chỉ rd |
| **RegWrite\_i** | 1 | Input | Tín hiệu cho phép ghi vào register |
| **RSdata\_o** | 32 | Output | Giá trị của thanh ghi có địa chỉ rs1 |
| **RTdata\_o** | 32 | Output | Giá trị của thanh ghi có địa chỉ rs2 |

Tín hiệu vào ra của khối được mô tả trong Bảng 2.10.

### Control

Khối control (Hình 2.11) có chức năng đưa ra các tín hiệu điều khiển ứng với các lệnh. Các tín hiệu đầu ra được mô tả như bảng chân lý (Bảng 2.12).

Diagram

Description automatically generated

Hình 2.11. Sơ đồ khối của khối control

Bảng 2.11. Chân vào ra của khối control

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **BrEq\_i** | 1 | Input | Đầu vào biểu thị 2 giá trị bằng nhau |
| **BrLT\_i** | 1 | Input | Đầu vào biểu thị A<B |
| **BrGe\_i** | 1 | Input | Đầu vào biểu thị A >= B |
| **opcode\_i** | 7 | Input | Giá trị opcode |
| **funct7\_i** | 7 | Input | Giá trị function 7 |
| **funct3\_i** | 3 | Input | Giá trị function 3 |
| **ImmSel\_o** | 3 | Output | Tín hiệu xác định loại lệnh |
| **PCSel\_o** | 1 | Output | Giá trị chọn PC.   * 1: PC = PC + JUMP * 0 : PC = PC + 4 |
| **BrUn\_o** | 1 | Output | So sánh số có dấu:   * 1: có dấu * 0: không dấu |
| **ASel\_o** | 1 | Output | A select |
| **BSel\_o** | 1 | Output | A select |
| **MemR\_o** | 1 | Output | Tín hiệu cho phép đọc vào data memory |
| **MemW\_o** | 1 | Output | Tín hiệu cho phép ghi vào data memory |
| **RegWEn\_o** | 1 | Output | Tín hiệu cho phép ghi vào register |
| **WBSel\_o** | 2 | Output | Tín hiệu chọn đường Write back |
| **ALUSel\_o** | 4 | Output | Tín hiệu chọn chức năng cho khối ALU |
| **insert\_nop\_flag\_o** | 1 | Output | Tín hiệu cho biết cần chèn lệnh NOP, dùng cho lệnh JAL |

Bảng 2.11 mô tả chức năng của chân vào ra của khối.

Bảng 2.12. Các tín hiệu điều khiển ứng với các lệnh

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PCSel\_o | ImmSel\_o | BrUn\_o | ASel\_o | BSel\_o | ALUSel\_o | MemR\_o | MemW\_o | RegWen\_o | WBSel\_o |
| add | 0 | ImmSelR | x | x | 0 | ALUadd | 0 | 0 | 1 | 01 |
| sub | 0 | ImmSelR | x | x | 0 | ALUsub | 0 | 0 | 1 | 01 |
| sll | 0 | ImmSelR | x | x | 0 | ALUsll | 0 | 0 | 1 | 01 |
| slt | 0 | ImmSelR | x | x | 0 | ALUslt | 0 | 0 | 1 | 01 |
| sltu | 0 | ImmSelR | x | x | 0 | ALUsltu | 0 | 0 | 1 | 01 |
| xor | 0 | ImmSelR | x | x | 0 | ALUxor | 0 | 0 | 1 | 01 |
| srl | 0 | ImmSelR | x | x | 0 | ALUsrl | 0 | 0 | 1 | 01 |
| sra | 0 | ImmSelR | x | x | 0 | ALUsra | 0 | 0 | 1 | 01 |
| or | 0 | ImmSelR | x | x | 0 | ALUor | 0 | 0 | 1 | 01 |
| and | 0 | ImmSelR | x | x | 0 | ALUand | 0 | 0 | 1 | 01 |
| addi | 0 | ImmSelI | x | x | 1 | ALUadd | 0 | 0 | 1 | 01 |
| xori | 0 | ImmSelI | x | x | 1 | ALUxor | 0 | 0 | 1 | 01 |
| ori | 0 | ImmSelI | x | x | 1 | ALUor | 0 | 0 | 1 | 01 |
| andi | 0 | ImmSelI | x | x | 1 | ALUand | 0 | 0 | 1 | 01 |
| slli | 0 | ImmSelI | x | x | 1 | ALUsll | 0 | 0 | 1 | 01 |
| lw | 0 | ImmSelI | x | x | 1 | ALUadd | 1 | 0 | 1 | 00 |
| S | 0 | ImmSelS | x | x | 1 | ALUadd | 0 | 1 | 0 | xx |
| beq | (BrEq\_i) ? 1 : 0 | ImmSelB | x | x | 1 | ALUadd | 0 | 0 | 0 | xx |
| bne | (BrLT\_i) ? 0 : 1 | ImmSelB | x | x | 1 | ALUadd | 0 | 0 | 0 | xx |
| blt | (BrLT\_i) ? 1 : 0 | ImmSelB | 0 | x | 1 | ALUadd | 0 | 0 | 0 | xx |
| bge | (BrGe\_i) ? 1 : 0 | ImmSelB | 0 | x | 1 | ALUadd | 0 | 0 | 0 | xx |
| bltu | (BrLT\_i) ? 1 : 0 | ImmSelB | 1 | x | 1 | ALUadd | 0 | 0 | 0 | xx |
| bgeu | (BrGe\_i) ? 1 : 0 | ImmSelB | 1 | x | 1 | ALUadd | 0 | 0 | 0 | xx |
| jal | 0 | ImmSelJ | x | x | 1 | ALUadd | 0 | 0 | 1 | 10 |
| lui | 0 | ImmSelU | 1 | x | 1 | ALUadd | 0 | 0 | 1 | 01 |
| default | 0 | 1111 | 0 | x | 1 | ALUnop | 0 | 0 | 0 | 01 |

### Hazard Detection

Khối Hazard detection (Hình 2.12) 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)and (ID\_EX\_rd !=0) then stall the pipline. |

A picture containing diagram

Description automatically generated

Hình 2.12. Sơ đồ khối của khối hazard dectection.

Tín hiệu hazard sẽ được gửi đi các khối để điều khiển stall 1 chu kỳ.

Bảng 2.13 mô tả chức năng của tín hiệu vào ra của khối

Bảng 2.13. Chân vào ra của khối Hazard detection

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **IF\_IDrs1\_i** | 5 | Input | Giá trị rs1 được lưu ở thanh ghi IF/ID |
| **IF\_IDrs2** | 5 | Input | Giá trị rs2 được lưu ở thanh ghi IF/ID |
| **ID\_EXrd\_i** | 5 | Input | Giá trị rsd được lưu ở thanh ghi ID/EX |
| **ID\_EX\_MemRead\_i** | 1 | Input | Giá trị cho phép đọc dữ liệu từ data memory |
| **hazard\_o** | 1 | Output | Tín hiệu cho biết phát hiện hazard |

### Forwarding

Khối Forwarding 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.

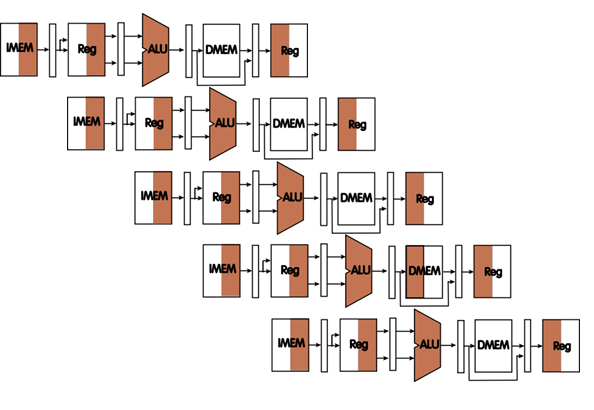
A picture containing diagram

Description automatically generated

Hình 2.13. Sơ đồ khối của khối forwarding

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

|  |
| --- |
| if(EX\_MEM\_RegWrite\_i && (EX\_MEM\_RD\_i != 5'b00000) && (EX\_MEM\_RD\_i == ID\_EX\_RS\_i)) Forward = 2'b10; //EX hazard  else if(MEM\_WB\_RegWrite\_i && (MEM\_WB\_RD\_i != 5'b00000) && MEM\_WB\_RD\_i == ID\_EX\_RS\_i) Forward = 2'b01; //MEM hazard  else Forward = 2'b00; |





**sw x12, 0(x5)**

**addi x12, x0, 50**

Hình 2.14. Mô phỏng quá trình EX haxzard

Diagram

Description automatically generated



**addi x14, x13, 5**

**stall**

**lw x13, 0(x5)**



Hình 2.15. Mô phỏng quá trình MEM hazard

Quá trình xử lý hazard dùng forward như Hình 2.14 Hình 2.15.

Vì thiết kế đề xuất cải tiến tiết kiệm 1 chu kỳ stall cho lệnh loại B, nên bộ so sánh sẽ được đặt ở ở ID/IF nên cần 1 forward data về sau bộ so sánh. Xác định như sau:

|  |
| --- |
| if ((ID\_EX\_RD\_i != 0) && ID\_EX\_RegWrite\_i && (IF\_ID\_RS\_i == ID\_EX\_RD\_i))  forward\_comp1 = 2'b01;  else if (EX\_MEM\_MemR\_i && (EX\_MEM\_RD\_i != 0) && EX\_MEM\_RegWrite\_i && (EX\_MEM\_RD\_i == IF\_ID\_RS\_i)) forward\_comp1\_o = 2'b10;  else forward\_comp = 2'b00; |

Diagram

Description automatically generated

**beq x3, x2, J**



**addi x3, x0, 54**

Hình 2.16 Quá trình forward đối với trường hợp hazard cho lệnh loại B

Diagram

Description automatically generated



**beq x3, x2, J**

**stall**

**lw x3, 0(x0)**



Hình 2.17. Quá trình forward đối với trường hợp hazard cho lệnh loại B

Quá trình xử lý hazard dùng forward như Hình 2.16 Hình 2.17.

Bảng 2.14 mô tả các đường dữ liệu forward ứng với các giá trị select tương ứng.

Bảng 2.14. Mô tả chọn đường dữ liệu forward

|  |  |  |
| --- | --- | --- |
| **Mux control** | Nguồn | Mô tả |
| **forward = 00** | ID/EX | Đường input thứ nhất của ALU đến từ tệp thanh ghi. |
| **forward = 10** | EX/MEM | Đường input thứ nhất của ALU được foward từ kết quả tính toán của ALU cho lệnh ngay trước đó. |
| **forward = 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\_comp = 00** | Register | Đường input thứ hai của branch compare đến từ tệp thanh ghi. |
| **forward\_comp = 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\_comp = 01** | ALU | Đường input thứ hai của branch compare đến từ kết quả tính toán ALU của lệnh trước đó |

Bảng 2.15. Chân vào ra của khối forwarding

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **EX\_MEM\_RegWrite\_i** | 1 | Input | Giá trị cho phép ghi vào register lưu ở EX/MEM |
| **ID\_EX\_RegWrite\_i** | 1 | Input | Giá trị cho phép ghi vào register lưu ở ID/EX |
| **MEM\_WB\_RegWrite\_i** | 1 | Input | Giá trị cho phép ghi vào register lưu ở MEM/WB |
| **EX\_MEM\_MemR\_i** | 1 | Input | Giá trị cho phép đọc data memory lưu ở EX/MEM |
| **EX\_MEM\_RD\_i** | 5 | Input | Giá trị rd lưu ở EX/MEM |
| **IF\_ID\_RS\_i** | 5 | Input | Giá trị rs1 lưu ở IF/ID |
| **IF\_ID\_RT\_i** | 5 | Input | Giá trị rs2 lưu ở IF/ID |
| **ID\_EX\_RS\_i** | 5 | Input | Giá trị rs1 lưu ở ID/EX |
| **ID\_EX\_RT\_i** | 5 | Input | Giá trị rs2 lưu ở ID/EX |
| **ID\_EX\_RD\_i** | 5 | Input | Giá trị rd lưu ở ID/EX |
| **MEM\_WB\_RD\_i** | 5 | Input | Giá trị rd lưu ở MEM/WB |
| **ForwardA\_o** | 2 | Output | Giá trị chọn đường forward của 1 |
| **ForwardB\_o** | 2 | Output | Giá trị chọn đường forward của 2 |
| **forward\_comp1\_o** | 2 | Output | Giá trị chọn đường forward của khối comp 1 |
| **forward\_comp2\_o** | 2 | Output | Giá trị chọn đường forward của khối comp 2 |

### Control Mux

Khi phát hiện có hazard, khối control mux (Hình 2.18) sẽ xóa hết các tín hiệu các tín hiệu điều khiển về 0 và đẩy sang thanh ghi ID/EX.

Diagram

Description automatically generated

Hình 2.18. Sơ đồ khối của khối control mux

Bảng 2.16 mô tả chức năng của từng chân vào ra của khối.

Bảng 2.16. Chân vào ra của khối control mux

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **Hazard\_i** | 1 | Input | Tín hiệu chỉ ra có hazard |
| **RegDst\_i** | 5 | Input | Giá trị rd ở thanh ghi IF/ID |
| **ASel\_i** | 1 | Input | A select |
| **BSel\_i** | 1 | Input | B select |
| **MemR\_i** | 1 | Input | Tín hiệu cho phép đọc từ memory |
| **MemW\_i** | 1 | Input | Tín hiệu cho phép ghi từ memory |
| **RegWEn\_i** | 1 | Input | Tín hiệu cho phép ghi vào register |
| **WBSel\_i** | 2 | Input | Tín hiệu chọn đường Write Back |
| **ALUSel\_i** | 4 | Input | Lựa chọn toán tử |
| **RegDst\_o** | 5 | Output | Giá trị rd đẩy sang thanh ghi ID/EX |
| **ASel\_o** | 1 | Output | A select đẩy sang thanh ghi ID/EX |
| **BSel\_o** | 1 | Output | B select đẩy sang thanh ghi ID/EX |
| **MemR\_o** | 1 | Output | Tín hiệu cho phép đọc từ memory sang thanh ghi ID/EX |
| **MemW\_o** | 1 | Output | Tín hiệu cho phép ghi từ memory sang thanh ghi ID/EX |
| **RegWEn\_o** | 1 | Output | Tín hiệu cho phép ghi vào register đẩy sang thanh ghi ID/EX |
| **WBSel\_o** | 2 | Output | Tín hiệu chọn đường Write Back đẩy sang thanh ghi ID/EX |
| **ALUSel\_o** | 4 | Output | Lựa chọn toán tử đẩy sang thanh ghi ID/EX |

### ALU control

Khối ALU (Arithmetic logic unit) thực hiện các chức năng tính toán số học bao gồm (add, sub, sll, slt, sltu, xor, srl, sra, or, and).

Tín hiệu ALU\_Sel sẽ chọn chức năng thực hiện cho ALU, các chức năng được định nghĩa với giá trị như Bảng 2.17.

Bảng 2.17. Giá trị ứng với từng chức năng của ALU

|  |  |
| --- | --- |
| ALUadd | 4'b0000 |
| ALUsub | 4'b0001 |
| ALUsll | 4'b0010 |
| ALUslt | 4'b0011 |
| ALUsltu | 4'b0100 |
| ALUxor | 4'b0101 |
| ALUsrl | 4'b0110 |
| ALUsra | 4'b0111 |
| ALUor | 4'b1000 |
| ALUand | 4'b1001 |
| ALUnop | 4'b1111 |

Graphical user interface, application, Word

Description automatically generated

Hình 2.19. Sơ đồ khối của khối ALU

Bảng 2.18 mô tả các tín hiệu vào ra của khối.

Bảng 2.18. Chân tín hiệu vào ra của khối ALU

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **src1** | 32 | Input | Dữ liệu nguồn 1 |
| **src2** | 32 | Input | Dữ liệu nguồn 2 |
| **ALU\_Sel** | 4 | Itput | Lựa chọn chức năng |
| **alu\_result** | 32 | Output | Kết quả đầu ra |
| **zero\_flag** | 1 | Output | Cờ zero |

### Adder

Khối adder thực hiện chức năng của bộ full adder 32 bit.

|  |
| --- |
| assign result\_o = data1\_i + data2\_i; |

A picture containing application

Description automatically generated

Hình 2.20. Sơ đồ khối của adder

Bảng 2.19. Chân vào ra của khối adder

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **data1\_i** | 32 | Input | Số hạng thứ 1 |
| **data2\_i** | 32 | Input | Số hạng thứ 2 |
| **result\_o** | 32 | Output | Kết quả sau phép cộng |

Bảng 2.19, Hình 2.20 mô tả tín hiệu vào ra và sơ đồ khối của khối adder

### IF/ID

Khối IF/ID là 1 thanh ghi phục vụ cho pipeline (Hình 2.21).

Khi xuất hiện có hazard giá trị PC và mã lệnh sẽ được giữ nguyên và khi có tín hiệu flush mã lệnh sẽ gán bằng 0 (lệnh NOP). Khối sẽ thực hiện như sau:

|  |
| --- |
| if(hazard\_i) begin  pc\_o <= pc\_o;  inst\_o <= inst\_o;  end else  if(flush\_i) begin  pc\_o <= pc\_i;  inst\_o <= 32'b0;  end else begin  pc\_o <= pc\_i;  inst\_o <= inst\_i;  end |

Diagram

Description automatically generated

Hình 2.21. Sơ đồ khối của khối IF/ID

Bảng 2.20 mô tả các tín hiệu của khối IF/ID.

Bảng 2.20. Chân vào ra của khối IF/ID

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **pc\_i** | 32 | Input | Giá trị PC ở tầng IF |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **hazard\_i** | 1 | Input | Tín hiệu chỉ ra có hazard |
| **flush\_i** | 1 | Input | Tín hiệu thông báo chèn lệnh NOP |
| **inst\_i** | 32 | Input | Mã lệnh lưu từ tầng IF |
| **pc\_o** | 32 | Output | Giá trị PC đẩy sang tầng ID |
| **inst\_o** | 32 | Output | Mã lệnh đẩy sang tầng ID |

### EX\_MEM

Khối EX/MEM (Hình 2.22) là thanh ghi trung gian giữa 2 tầng EX và MEM.

Diagram

Description automatically generated

Hình 2.22. Sơ đồ khối của khối EX/MEM

Bảng 2.21 mô tả các tín hiệu của khối EX/MEM.

Bảng 2.21. Chân vào ra của khối EX/MEM

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **pc\_i** | 32 | Input | Giá trị PC ở tầng EX |
| **ALU\_i** | 32 | Input | Giá trị được tính toán ở ALU |
| **forward\_b\_data\_i** | 32 | Input | Giá trị sau bộ mux A ở tầng EX |
| **RegDst\_i** | 5 | Input | Giá trị rd ở EX |
| **MemR\_i** | 1 | Input | Tín hiệu cho phép đọc từ memory từ tầng EX |
| **MemW\_i** | 1 | Input | Tín hiệu cho phép ghi từ memory từ tầng EX |
| **RegWEn\_i** | 1 | Input | Tín hiệu cho phép ghi vào register từ tầng EX |
| **WBSel\_i** | 2 | Input | Tín hiệu chọn đường Write Back từ tầng EX |
| **pc\_o** | 32 | Output | Giá trị PC đẩy sang tầng MEM |
| **ALU\_o** | 32 | Output | Giá trị kết quả ALU đẩy sang tầng MEM |
| **forward\_b\_data\_o** | 32 | Output | Giá trị sau bộ mux A đẩy sang tầng MEM |
| **RegDst\_o** | 5 | Output | Giá trị rd đẩy sang tầng MEM |
| **MemR\_o** | 1 | Output | Tín hiệu cho phép đọc từ memory đẩy sang tầng MEM |
| **MemW\_o** | 1 | Output | Tín hiệu cho phép ghi từ memory sang tầng MEM |
| **RegWEn\_o** | 1 | Output | Tín hiệu cho phép ghi vào register đẩy sang tầng MEM |
| **WBSel\_o** | 2 | Output | Tín hiệu chọn đường Write Back đẩy sang tầng MEM |

### MEM\_WB

Khối MEM/WB (Hình 2.23) là thanh ghi lưu lại các giá trị dùng cho việc write back.

Diagram

Description automatically generated

Hình 2.23. Sơ đồ khối của khối MEM/WB

Bảng 2.22 mô tả chức năng của các tín hiệu có trong khối.

Bảng 2.22. Chân vào ra của khối MEM/WB

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **RegWen\_i** | 1 | Input | Tín hiệu cho phép ghi vào register từ tầng MEM |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực mức thấp |
| **RegDst\_i** | 5 | Input | Giá trị rd ở tầng MEM |
| **data\_wb\_i** | 32 | Input | Giá trị write back xử lý ở tầng MEM |
| **RegDst\_o** | 5 | output | Giá trị rd đẩy sang tầng WB |
| **RegWen\_o** | 1 | output | Tín hiệu cho phép ghi vào register đẩy sang tầng WB |
| **data\_wb\_o** | 32 | output | Giá trị write back |

### ID/EX

Khối EX/MEM (Hình 2.24) là thanh ghi trung gian giữa 2 tầng ID và EX phục vụ cho pipeline.

Chart

Description automatically generated

Hình 2.24. Sơ đồ khối của khối ID/EX

Bảng 2.23 mô tả chức năng của các tín hiệu có trong khối.

Bảng 2.23. Chân vào ra của khối ID/EX

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **clk** | 1 | Input | Tín hiệu xung đồng hồ |
| **rst\_n** | 1 | Input | Tín hiệu reset tích cực ở mức thấp |
| **pc\_i** | 32 | Input | Giá trị PC ở tầng ID |
| **imm\_i** | 32 | Input | Giá trị immediate đã mở rộng dấu |
| **RegDst\_i** | 5 | Input | Giá trị rd ở tầng ID |
| **RegS1\_i** | 5 | Input | Giá trị rs1 |
| **RegS2\_i** | 5 | Input | Giá trị rs2 |
| **data1\_i** | 32 | Input | Giá trị đầu vào 1 ở tầng ID |
| **data2\_i** | 32 | Input | Giá trị đầu vào 2 ở tầng ID |
| **ASel\_i** | 1 | Input | A select |
| **BSel\_i** | 1 | Input | B select |
| **MemR\_i** | 1 | Input | Tín hiệu cho phép đọc từ memory từ tầng ID |
| **MemW\_i** | 1 | Input | Tín hiệu cho phép ghi từ memory từ tầng ID |
| **RegWEn\_i** | 1 | Input | Tín hiệu cho phép ghi vào register từ tầng ID |
| **WBSel\_i** | 2 | Input | Tín hiệu chọn đường write back từ tầng ID |
| **ALUSel\_i** | 4 | Input | Tín hiệu lựa chọn chức năng của ALU |
| **pc\_o** | 32 | Output | Giá trị PC đẩy sang tầng EX |
| **imm\_o** | 32 | Output | Giá trị immediate đã mở rộng dấu đẩy sang tầng EX |
| **RegDst\_o** | 5 | Output | Giá trị rd đẩy sang tầng EX |
| **RegS1\_o** | 5 | Output | Giá trị rs1 sang tầng EX |
| **RegS2\_o** | 5 | Output | Giá trị rs2 sang tầng EX |
| **data1\_o** | 32 | Output | Giá trị đầu vào 1 sang tầng EX |
| **data2\_o** | 32 | Output | Giá trị đầu vào 1 sang tầng EX |
| **ASel\_o** | 1 | Output | A select đẩy sang tầng EX |
| **BSel\_o** | 1 | Output | B select đẩy sang tầng EX |
| **MemR\_o** | 1 | Output | Tín hiệu cho phép đọc từ memory sang tầng EX |
| **MemW\_o** | 1 | Output | Tín hiệu cho phép ghi từ memory sang tầng EX |
| **RegWEn\_o** | 1 | Output | Tín hiệu cho phép ghi vào register đẩy sang tầng EX |
| **WBSel\_o** | 2 | Output | Tín hiệu chọn đường Write Back đẩy sang tầng EX |
| **ALUSel\_o** | 4 | Output | Tín hiệu lựa chọn chức năng của ALU đẩy sang tầng EX |

### MUX21

Bộ Mux này (Hình 2.25) đơn giản là gán dữ liệu ra theo một trong số các tín hiệu vào dựa vào tín hiệu select\_i. Trong Verilog ta dùng toán tử ? : để rẽ nhánh các điều kiện của sel và xác định đầu ra, code của bộ Mux ở dưới hình Mux tương ứng.

Diagram

Description automatically generated

Hình 2.25. Sơ đồ khối của mux21

Mô tả như sau:

|  |
| --- |
| assign data\_o = (select\_i) ? data2\_i : data1\_i; |

Bảng 2.24. Chân vào ra của khối mux21

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **select\_i** | 1 | Input | Tín hiệu lựa chọn |
| **data1\_i** | 32 | Input | Giá trị đầu vào 1 |
| **data2\_i** | 32 | Input | Giá trị đầu vào 2 |
| **data\_o** | 32 | Output | Giá trị đầu ra |

### MUX31

Bộ Mux này tương tự bộ mux21 chỉ khác là có 3 đầu vào.

Diagram

Description automatically generated

Hình 2.26. Sơ đồ khối của khối mux31

Khối thực hiện như sau:

|  |
| --- |
| always @(\*) begin  case(select\_i)  2'b00: data\_o = data1\_i;  2'b01: data\_o = data2\_i;  2'b10: data\_o = data3\_i;  default : data\_o = data1\_i;  endcase  end |

Bảng 2.25. Chân vào ra của khối mux31

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Width** | **Input/Output** | **Description** |
| **select\_i** | 1 | Input | Tín hiệu lựa chọn |
| **data1\_i** | 32 | Input | Giá trị đầu vào 1 |
| **data2\_i** | 32 | Input | Giá trị đầu vào 2 |
| **data3\_i** | 32 | Input | Giá trị đầu vào 3 |
| **data\_o** | 32 | Output | Giá trị đầu ra |

# KIỂM THỬ (VERIFICATION)

Chương này trình bày các kết quả mô phỏng cho từng khối và toàn bộ thiết kế được triển khai bằng ngôn ngữ SystemVerilog trên phần mềm Questasim.

## Test case

Các kịch bản kiểm tra:

* Kiểm tra quá trình reset, đảm bảo các đầu ra được thiết lập về đúng giá trị
* 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 structural hazard
* Kiểm tra các trường hợp instruction có control hazard
* Kiểm tra lệnh JAL

## Tạo đầu vào vào kiểm tra các test case

Sử dụng mã nguồn chuyển đổi từ assembly sang mã máy dạng kiến trúc RISC-V [2]. Từ đó, nạp mã máy vào instruction memory để hệ thống vận hành.

### Quá trình reset

Hình 3.1 cho thấy quá trình reset đảm bảo các đầu ra thiết lập về đúng giá trị

A picture containing timeline

Description automatically generated

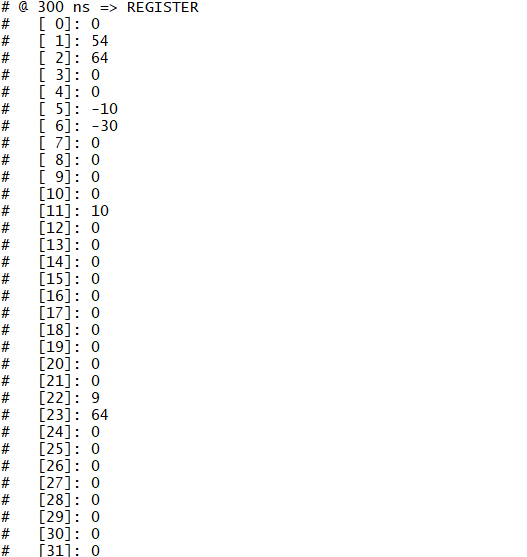
Hình 3.1. Kết quả mô phỏng reset của hệ thống

### Các trường hợp instruction không có hazard

Thực hiện nạp code sau vào hệ thống:

|  |  |
| --- | --- |
| addi x1, x0, 54  addi x2, x0, 64  addi x5, x0, -10  addi x6, x0, -30  sub x11, x2, x1  sw x2, 0(x7)  blt x5, x6, J  addi x22, x0, 1  addi x23, x0, 1  J: addi x22, x0, 9  lw x23, 0(x7) | 0x03600093  0x04000113  0xFF600293  0xFE200313  0x401105B3  0x0023A023  0x0062C663  0x00100B13  0x00100B93  0x00900B13  0x0003AB83 |

Kết quả sau mô phỏng:



Hình 3.2. Giá trị của của các thanh ghi sau quá trình mô phỏng

Table

Description automatically generated

Hình 3.3. Giá trị của các ô nhớ trong data memory sau quá trình mô phỏng

A picture containing graphical user interface

Description automatically generated

Hình 3.4. Wave của quá trình mô phỏng

Kết luận: Từ kết quả ở Hình 3.2 Hình 3.3 Hình 3.4, cho thấy hệ thống hoạt động đúng với chức năng.

### Trường hợp instruction có structural hazard

Đồng thời ghi và đọc từ memory tách thành 2 khối memory

Đồng thời đọc ghi từ register đọc ở sườn dương, ghi ở sườn âm

|  |  |
| --- | --- |
| addi x1, x0, 54  addi x2, x0, 64  sw x2, 0(x3)  add x4, x1, x2  lw x5, 0(x3)  slt x6, x0, x3  sw x4, 0(x3)  add x5, x1, x2 | 03600093  04000113  0021A023  00208233  0001A283  00302333  0041A023  002082B3 |

Diagram

Description automatically generated

**slt x6, x0, x3**

**sw x4, 0(x3)**

**addi x5, x1, x2**

**lw x5, 0(x3)**

**add x4, x1, x2**

Hình 3.5. Mô tả cùng ghi vào memory

A picture containing table

Description automatically generatedTable

Description automatically generated

Hình 3.6. Kết quả của các thanh ghi ở 2 thời điểm

Diagram

Description automatically generated

**slt x6, x0, x3**

**sw x4, 0(x3)**

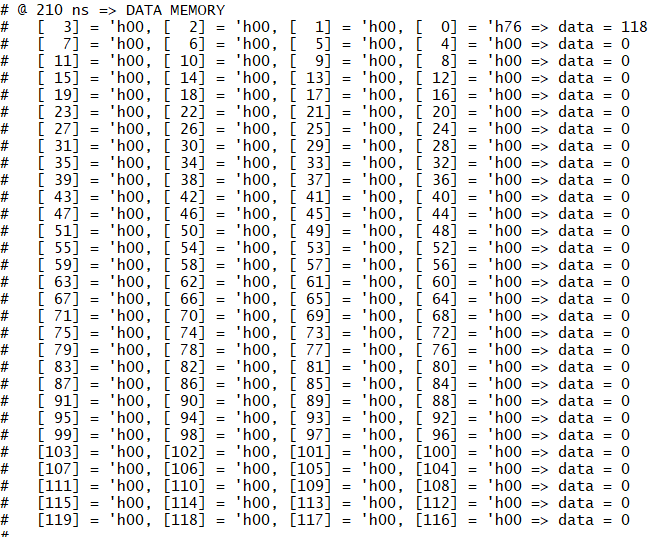
**addi x5, x1, x2**

**lw x5, 0(x3)**

**add x4, x1, x2**

Hình 3.7. Mô tả cùng đọc ghi vào register

Table

Description automatically generated

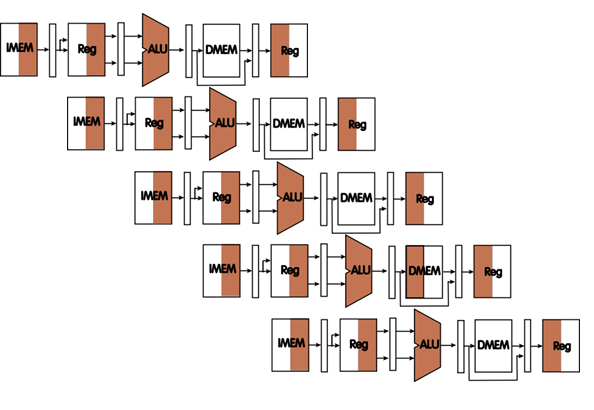
Hình 3.8. Kết quả của thanh ghi và data memory ở 2 thời điểm

Kết luận: Hình 3.6, Hình 3.8 cho thấy, hệ thống hoạt động đúng.

### Trường hợp instruction có data hazard

EX Haxzard:

|  |  |
| --- | --- |
| addi x5, x0, 4  addi x12, x0, 50  sw x12, 0(x5)  addi x12, x0, -30  sw x12, 4(x5) | 00400293  03200613  00C2A023  FE200613  00C2A223 |





**sw x12, 0(x5)**

**addi x12, x0, 50**

Hình 3.9. Mô phỏng quá trình EX haxzard

A picture containing table

Description automatically generatedA picture containing table

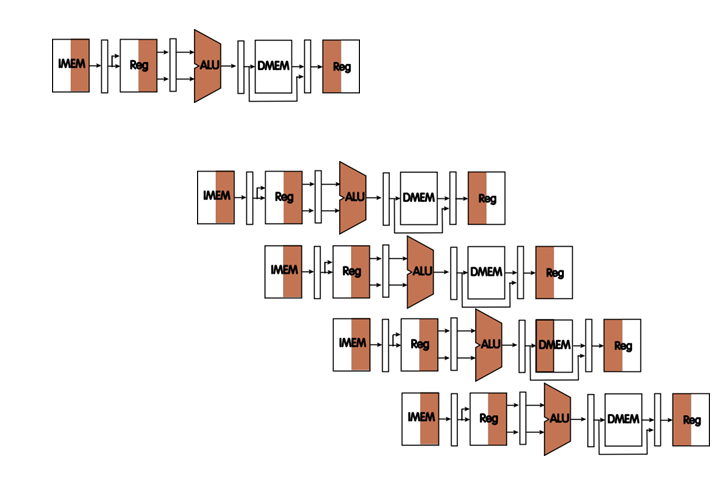
Description automatically generated Table

Description automatically generated

Hình 3.10. Kết quả của các thanh ghi và data memory ở 3 thời điểm

MEM Hazard:

|  |  |
| --- | --- |
| addi x5, x0, 4  addi x12, x0, 50  sw x12, 0(x5)  lw x13, 0(x5)  addi x14, x13, 5 | 00400293  03200613  00C2A023  0002A683  00560713 |





**addi x14, x13, 5**

**stall**

**lw x13, 0(x5)**



Hình 3.11. Mô phỏng quá trình MEM hazard

A screenshot of a video game

Description automatically generated

Hình 3.12. Phát hiện có hazard

A picture containing table

Description automatically generated

Hình 3.13. Kết quả của MEM hazard

Kết luận: Hình 3.12, Hình 3.13 cho thấy, hệ thống hoạt động đúng khi xuất hiện data hazard.

### Trường hợp instruction có control hazard

Trước lệnh loại B là lệnh số học:

|  |  |
| --- | --- |
| addi x2, x0, 54  addi x3, x0, 54  beq x3, x2, J  addi x4, x0, 3  J: addi x4, x0, 4 | 03600113  03600193  00218463  00300213  00400213 |

Diagram

Description automatically generated

**beq x3, x2, J**



**addi x3, x0, 54**

Hình 3.14 Quá trình forward đối với trường hợp hazard cho lệnh loại B

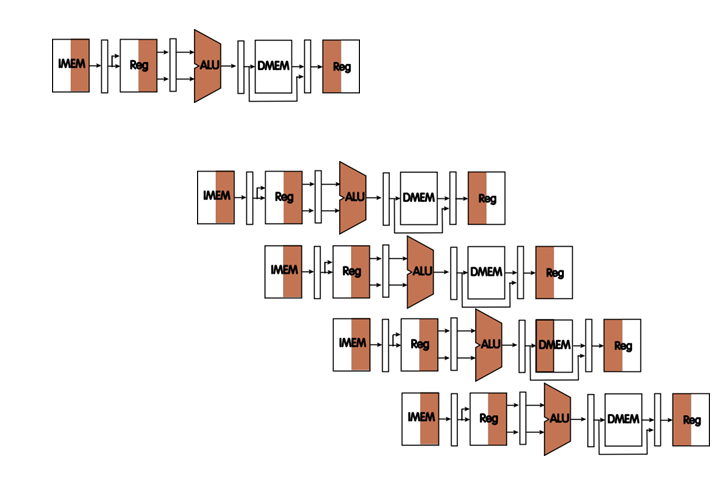
A picture containing shape

Description automatically generated

Hình 3.15. Kết quả sau quá trình mô phỏng

Trước lệnh loại B là lệnh load:

|  |  |
| --- | --- |
| addi x2, x0, 54  sw x2, 0(x0)  lw x3, 0(x0)  beq x3, x2, J  addi x4, x0, 3  J: addi x4, x0, 4 | 03600113  00202023  00002183  00218463  00300213  00400213 |





**beq x3, x2, J**

**stall**

**lw x3, 0(x0)**



Hình 3.16. Quá trình forward đối với trường hợp hazard cho lệnh loại B

A picture containing table

Description automatically generated

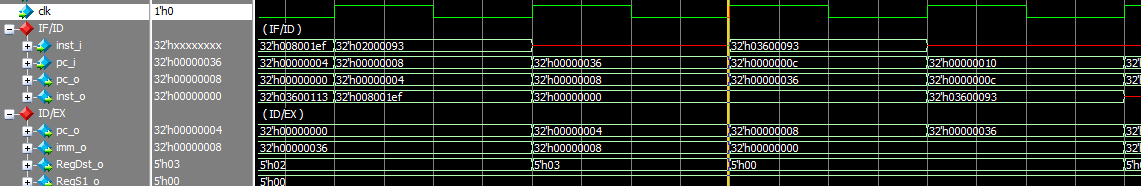
Hình 3.17. Kết quả của quá trình mô phỏng

Kết luận: Hình 3.15, Hình 3.17 cho thấy hệ thống hoạt động đúng chức năng

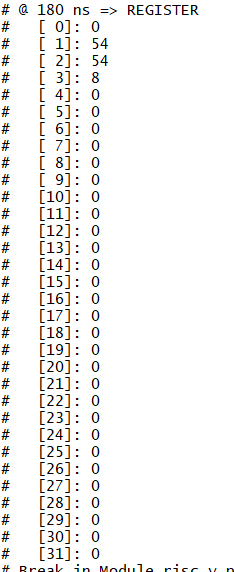
### Kiểm tra lệnh JAL

Đối với lệnh JAL, hệ thống sẽ chèn 2 lệnh NOP giữa 2 lệnh hazard (Hình 3.18).

|  |  |
| --- | --- |
| addi x2, x0, 54  jal x3, A  addi x1, x0, 32  A:addi x1,x0, 54 | 03600113  008001EF  02000093  03600093 |



Hình 3.18. Mô tả chèn 2 lệnh NOP trên questasim



Hình 3.19. Kết quả sau mô phỏng

Kết luận: Hình 3.19 cho thấy hệ thống hoạt động đúng với lệnh JAL.

# MÔ PHỎNG CHẠY THUẬT TOÁN SẮP XẾP NỔI BỌT

Chương này trình bày, mô phỏng thuật toán sắp xếp nổi bọt chạy trên hệ thống đã thiết kế ở CHƯƠNG 2.

## Triển khai thuật toán nổi bọt bằng assembly và chuyển sang ngôn ngữ máy

Thuật toán sắp xếp nổi bọt viết bằng assembly:

|  |
| --- |
| # bubble sort  addi x5, x0, 4 #x3 base address  addi x12, x0, 50  sw x12, 0(x5)  addi x12, x0, -30  sw x12, 4(x5)  addi x12, x0, 70  sw x12, 8(x5)  addi x12, x0, 1  sw x12, 12(x5)  addi x12, x0, 3  sw x12, 16(x5)  addi x12, x0, 56  sw x12, 20(x5)  addi x12, x0, 12  sw x12, 24(x5)  addi x12, x0, -85  sw x12, 28(x5)  addi x12, x0, -1  sw x12, 32(x5)  addi x12, x0, 17  sw x12, 36(x5)  addi x12, x0, 18  sw x12, 40(x5)  addi x12, x0, -47  sw x12, 44(x5)  addi x12, x0, 78  sw x12, 48(x5)  addi x1, x0, 0  addi x2, x0, 11  outer\_loop\_start:  add x10, x0, x5  lw x7, 0(x10)  sub x11, x2, x1 # Inner loop count: outer loop count - sorted elements  inner\_loop\_start:  lw x4, 4(x10)  blt x7,x4, noswap  sw x4, 0(x10)  sw x7, 4(x10)  noswap:  slt x6, x7, x4  beq x6, x0, skip  add x7, x4, x0  skip:  addi x10, x10, 4 #Increment the address to next element  addi x11, x11, -1  bne x11, x0, inner\_loop\_start  inner\_loop\_end:  addi x1, x1, 1  blt x1, x2, outer\_loop\_start  outer\_loop\_end: |

Dùng [2] để chuyển mã nguồn trên sang mã máy:

|  |
| --- |
| 0x00400293  0x03200613  0x00C2A023  0xFE200613  0x00C2A223  0x04600613  0x00C2A423  0x00100613  0x00C2A623  0x00300613  0x00C2A823  0x03800613  0x00C2AA23  0x00C00613  0x00C2AC23  0xFAB00613  0x00C2AE23  0xFFF00613  0x02C2A023  0x01100613  0x02C2A223  0x01200613  0x02C2A423  0xFD100613  0x02C2A623  0x04E00613  0x02C2A823  0x00000093  0x00B00113  0x00500533  0x00052383  0x401105B3  0x00452203  0x0043C663  0x00452023  0x00752223  0x0043A333  0x00030463  0x000203B3  0x00450513  0xFFF58593  0xFC059EE3  0x00108093  0xFC20C4E3 |

## Nạp mã lệnh vào hệ thống và mô phỏng

Ở thời điểm bắt đầu, tiến hành nạp mã lệnh vào instrution memory (Hình 4.1).

Table

Description automatically generated

Hình 4.1. Khởi tạo instruction memory

Tiến hành đọc giá trị trong data memory khi có giá trị thay đổi.

Table

Description automatically generated

Hình 4.2. Dữ liệu trong data memory ở thời điểm thực hiện xong các lệnh load dữ liệu khởi tạo

Table

Description automatically generated

Hình 4.3. Dữ liệu trong data memory ở thời điểm thực hiện xong sắp xếp

Hình 4.2, Hình 4.3 mô tả dữ liệu trong data memory lúc chưa sắp xếp và sắp xếp đã xong.

Tiến hành xem log ở [3], nhận thấy đã thực hiện đúng thuật toán sắp xếp nổi bọt.

# KẾT LUẬN

Báo cáo này đã trình bày và triển khai kiến trúc của một RISCV32I 5-stage processor áp dụng kĩ thuật pipeline nhằm tăng tốc độ xử lí, bên cạnh đó, thiết kế đã xữ lí được toàn bộ các hazard có thể xảy ra trong quá trình xử lí thực hiện lệnh của CPU bao gồm data hazard, mem hazard và control hazard. Thiết kế được tiến hành triển khai bằng ngôn ngữ mô tả phần cứng SystemVerilog và mô phỏng kiểm thử trên phần mềm Questasim. Thực hiện chạy thuật toán sắp xếp nổi bọt, cho ra kết quả hoạt động đúng với yêu cầu đặt ra. Kiến trúc có thể thực hiện 5 kiểu lệnh assembly trong tập lệnh của RISCV gồm: R-type, I-type, S-type, B-type, U-type tuy nhiên chưa thể thực hiện được tất cả các lệnh có trong tập lệnh của RISCV (xấp xỉ 40 lệnh). Trong tương lai nhóm sẽ tiến hành hoàn thiện và triển khai kiến trúc của một RISCV32I 5-stage processor hoàn chỉnh với đầy đủ các chức năng, thực hiện được đầy đủ các lệnh trong kiến trúc tập lệnh của RISCV một cách tối ưu nhất.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | P. David A. and H. John L., Computer Organization and Design RISC-V Edition: The Hardware Software Interface, Morgan Kaufmann Publishers Inc., 2017. |
| [2] | A. Kritagya, "Assembly To Machine Code Translation," github, 2019. [Online]. Available: https://github.com/Kritagya-Agarwal/Assembly-To-Machine-Code-RISC-V. |
| [3] | nguyenviethi, "RISC-V processor - 32 bit, 5-stage pipeline," github, 8 2022. [Online]. Available: https://github.com/nguyenvietthi/KTMT\_20212/tree/main/risc\_v\_pipeline. |