TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO MÔN HỌC
KIẾN TRÚC MÁY TÍNH

Thiết kế mô phỏng bộ xử lý RISC V – 32I bằng ngôn ngữ lập trình phần cứng Verilog

Hoàng Khánh Linh 20240359E
linh.hk240359E@sis.hust.edu.vn

Chuyên ngành Điện tử Viễn thông

Giảng viên hướng dẫn: Tạ Thị Kim Huệ

Chữ ký của GVHD

Bộ môn: Kiến trúc máy tính
Viện: Điện tử Viễn thông

HÀ NỘI, 4/2025

LỜI MỞ ĐẦU

Trong thời đại công nghệ số phát triển, cùng với sự bùng nổ của các ứng dụng nhúng, trí tuệ nhân tạo, IoT, điện tử tiêu dùng… việc lựa chọn một kiến trúc vi xử lý vi điều khiển mang tính linh hoạt và hiệu suất cao là lựa chọn tối ưu cho các nhà sản xuất. RISC-V một kiến trúc vi xử lý mở, đang thu hút sự quan tâm và phát triển mạnh mẽ trong ngành công nghệ thông tin. Với khả năng linh hoạt, hiệu suất cao và chi phí thấp, RISC-V đang trở thành một lựa chọn hấp dẫn. RISC-V đang trở thành một lựa chọn phổ biến trong vi xử lý nhúng do khả năng linh hoạt và hiệu suất của nó. Với khả năng tùy chỉnh, RISC-V cho phép các nhà phát triển tạo ra các hệ thống nhúng tối ưu cho các ứng dụng cụ thể như IoT (Internet of Things), ô tô tự động, thiết bị y tế và nhiều ứng dụng khác. Trí tuệ nhân tạo (AI) đang trở thành một lĩnh vực quan trọng và RISC-V có vai trò quan trọng trong việc phát triển các hệ thống với linh hoạt và hiệu suất của RISC-V cho phép nó được sử dụng trong việc xử lý dữ liệu lớn, học máy và các ứng dụng AI khác. RISC-V cung cấp một nền tảng tốt để phát triển các mô hình AI phức tạp và tối ưu hoá hiệu suất tính toán. RISC-V cũng có ứng dụng trong lĩnh vực điện tử tiêu dùng như điện thoại di động, máy tính bảng, thiết bị gia dụng thông minh và nhiều thiết bị khác. Với hiệu suất cao và tiêu thụ năng lượng thấp, RISC-V giúp tối ưu hoá hoạt động và tăng cường trải nghiệm người dùng trong các thiết bị này. Với tính ứng dụng cao của RISC-V như vậy là lý do em lựa chọn đề tài "Thiết kế mô phỏng bộ xử lý RISC-V 32I bằng ngôn ngữ lập trình phần cứng Verilog" để nghiên cứu.

Bài báo cáo này sẽ trình bày chi tiết quá trình nghiên cứu, thiết kế, mô phỏng của bộ vi điều khiển RISC-V 32bit đã thiết kế:

Bài báo cáo gồm các phần sau:

- Chương 1: TỔNG QUAN VỀ ĐỀ TÀI
- Chương 2: CƠ SỞ LÝ THUYẾT
- Chương 3: THIẾT KẾ VÀ MÔ PHỎNG

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1.1. Giới thiệu sơ lược về đề tài

Đề tài "Thiết kế mô phỏng bộ xử lý RISC-V 32I bằng ngôn ngữ lập trình phần cứng Verilog" nhằm mục tiêu nghiên cứu về kiến trúc của vi điều khiển RISC-V, và sử dụng ngôn ngữ lập trình phần cứng để thiết kế bộ nhớ, bộ xử lý trung tâm (CPU), bộ điều khiển và các thành phần khác trong vi điều khiển RISC-V.

Theo yêu cầu môn học, đề tài này sẽ tập trung vào việc nghiên cứu kiến trúc của vi điều khiển RISC-V và thiết kế lõi xử lý theo kỹ thuật ống dẫn (Pipeline).

Lõi xử lý RISC-V theo kỹ thuật Pipeline sẽ thực thi các lệnh theo kiểu chồng lên nhau (overlap), và đồng thời điều khiển hướng đi dữ liệu một cách chính xác và kịp thời nhằm nâng cao hiệu suất của vi xử lý. Ngôn ngữ verilog được sử dụng để mô tả chi tiết các thành phần cấu tạo nên vi xử lý RISC-V 32 bit, và đồng thời cung cấp một phương tiện để tổng hợp mạch thành sản phẩm cuối cùng có thể được sản xuất.

Với ngôn ngữ verilog, chúng ta có thể tạo ra các mô hình mô phỏng để kiểm tra tính chính xác và chức năng của vi xử lý trước khi tiến hành tổng hợp và sản xuất mạch thực tế. Điều này không chỉ giúp tiết kiệm thời gian và chi phí trong quá trình phát triển mà còn giúp phát hiện và sửa chữa các lỗi tiềm ẩn trước khi mạch được sản xuất ra thị trường.

Đề tài này sẽ khám phá kiến trúc của một vi điều khiển RISC-V, cách thức thiết kế, mô phỏng vả tổng hợp một lõi vi xử lý sử dụng verilog, đặt nền móng cho việc phát triển các bộ vi xử lý, vi điều khiển RISC-V tiên tiến trong tương lai.

1.2. Mục tiêu nghiên cứu

Mục tiêu của việc chọn đề tài này là hiểu rõ kiến trúc của một bộ vi xử lý RISC-V, kỹ thuật Pipeline, đồng thời phát triển kỹ năng thiết kế hệ thống số, hiểu rõ về nguyên tắc hoạt động và cách sử dụng của FPGA để kết nối cũng như mô phỏng hệ thống. Nghiên cứu này hướng dẫn triển khai lõi vi xử

lý RISC-V 32 bit, kỹ thuật Pipeline 5 tầng:

- Tìm nạp lệnh
- Giải mã lệnh và đọc các thanh ghi
- Thực thi câu lệnh
- Truy xuất bộ nhớ
- Ghi lại kết quả

1.3. Phạm vi nghiên cứu

- Nghiên cứu lý thuyết:
  + Nghiên cứu kiến trúc vi xử lý RISC-V.
  + Nghiên cứu kỹ thuật Pipeline và xứ lý các xung đột trong kỹ thuật Pipeline.
- Thiết kế và lập trình:
  + Thiết kế lõi vi xử lý RISC-V 32bit với ngôn ngữ mô tả phần cứng VERILOG và phần mềm mô phỏng Vivado.
- Nghiên cứu phương pháp thiết kế một lõi vi xử lý, mô phỏng, debug và kiểm tra hiệu suất của chương trình.
- Thử nghiệm và tối ưu:
  + Thực hiệu thử nghiệm kiểm tra hoạt động của lõi xử lý RISC-V.
  + Tối ưu code để hoạt động chính xác nhất.

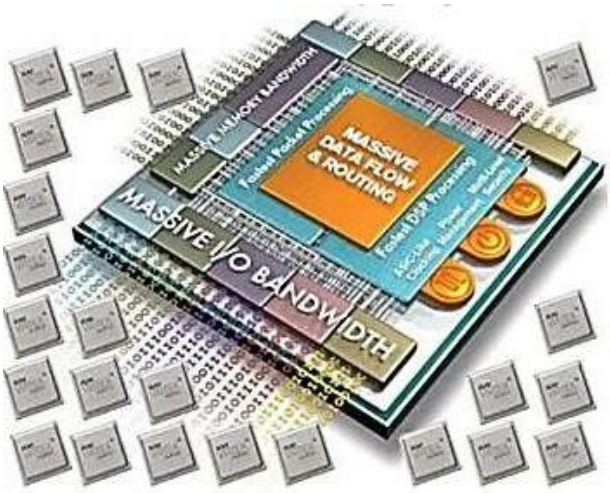
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Giới thiệu về FPGA, ngôn ngữ lập trình và phần mềm

2.1.1. Giới thiệu về FPGA

FPGA là công nghệ vi mạch tích hợp khả trình (PLD- Programmable Logic Device) mới nhất và tiên tiến nhất hiện nay. Theo định nghĩa của Xlinx thì FPGA là một vi mạch bán dẫn cũng giống như các vi mạch được cấu thành từ ma trận của các phần tử logic có khả năng tái cấu hình CLB (Configurable Logic Blocks) và có thể được nối với nhau thông qua các đường kết nối cấu hình được (Programmable interconnects). FPGA có thể tái cấu hình sau khi đã được sản xuất.

Thuật ngữ Field-Programmable chỉ quá trình tái cấu trúc IC có thể được thực hiện bởi người dùng cuối trong điều kiện thông thường, hay nói cách khác người kỹ sư thiết kế có thể dễ dàng hiện thực hoá vi mạch chức năng của mình sử dụng FPGA mà không phụ thuộc vào một quy trình công nghệ phức tạp nào trong nhà máy bán dẫn. Bên cạnh mức độ tích hợp rất lớn và đa dạng, các tài nguyên tái cấu trúc, khả năng lập trình tại chỗ làm FPGA trở thành công nghệ được lựa chọn nghiên cứu, phát triển cà ứng dụng ngày càng rộng rãi. Một trong những lý do chính là sự thay đổi, xuất hiện rất nhanh của các nền tảng ứng dụng và công nghệ mới mà vòng đời quá dài và chi phí thiết kế quá lớn của các chip ASIC chuyên dụng trở nên không còn phù hợp.

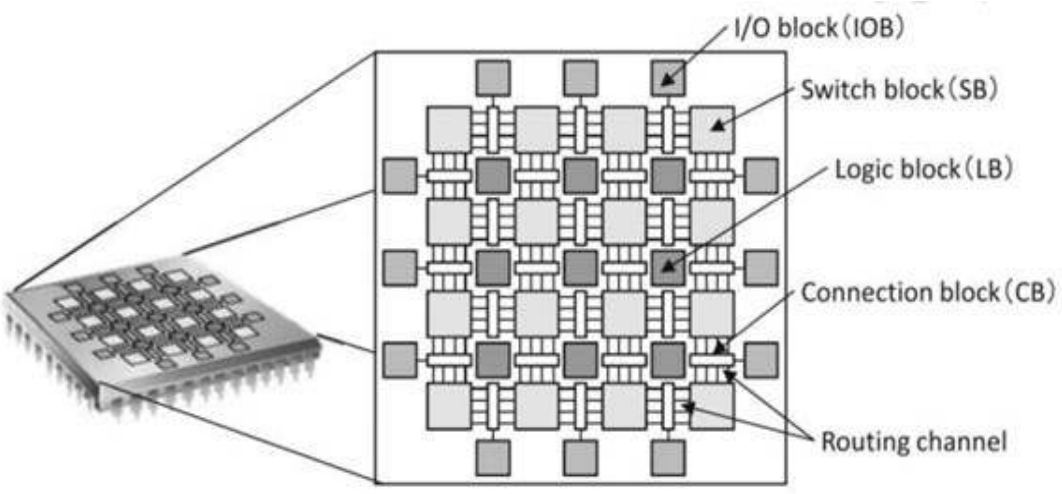


Hình 2. 1. Hình ảnh minh hoạ về chip FPGA

Hiện nay công nghệ FPGA đang được phát triển bởi nhiều công ty bán dẫn khác nhau nhưng nổi bật chỉ có XlinX và Altera[2], một công ty con của Intel là các nhà cung cấp chính.

a) Kiến trúc FPGA

Về mặt tổng quan FPGA truyền thống là mảng của các phần tử logic khả trình (gate-array) được bố trí dưới dạng ma trận và được liên kết với nhau thông qua hệ thống các đường kết nối khả trình. Thành phần cơ bản là các Logic Block (LB) có khả năng cấu hình để thực thi các chức năng logic khác nhau. Thành phần cơ bản thứ hai là các hệ thống kết nối khả trình được cấu tạo từ các kênh kết nối (interconnects) và các ma trận chuyển mạch cấu hình được. Thành phần cơ bản thứ ba là các cổng vào ra có khả năng kết nối tuỳ ý với các thành phần logic với các ngoại vi của bên ngoài. FPGA hiện đại sẽ phức tạp hơn so với mô tả trên và chứa nhiều hơn các tài nguyên có chức năng như bộ nhớ, các khối tính toán DSP, khối điều chỉnh xung nhịp clock (PLL) cho tới các thành phần phức tạp như một bộ vi xử lý và các lõi tăng tốc xử lý AI.



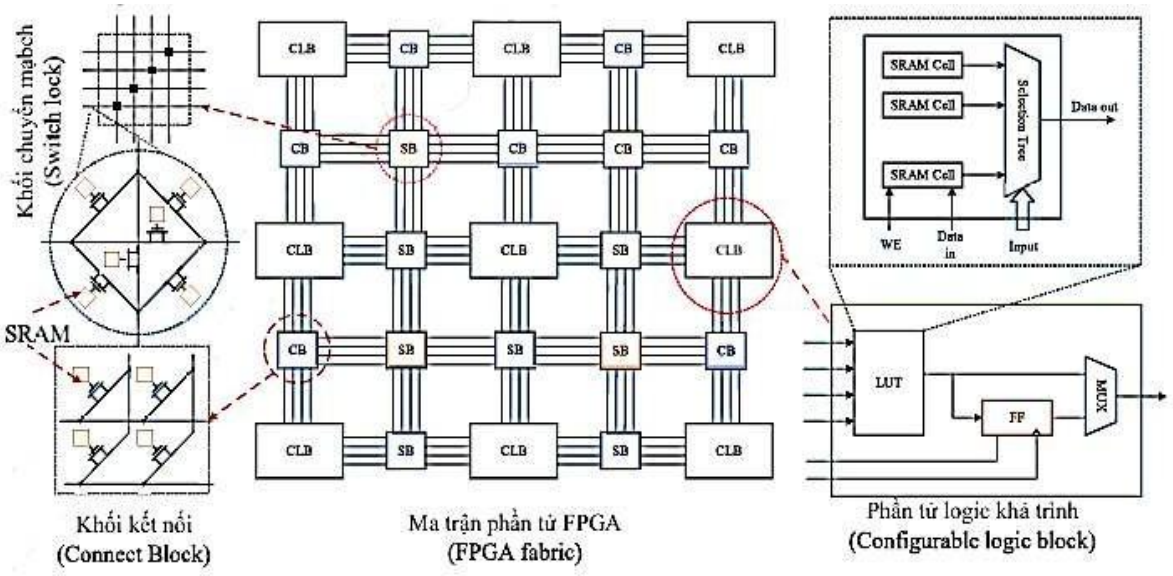
Hình 2. 2. Kiến trúc tổng quan của FPGA

FPGA sử dụng khối logic đa năng-LUT (Look Up Table) để xây dựng một nền tảng cấu hình động nhằm thực thi các chức năng mạch khác nhau (AND, OR, NOT,…) trên cùng một nền tảng mạch điện. Chức năng logic sẽ được quyết định bởi các ô nhớ đầu vào lựa chọn của khối LUT, từ đó mà

ta sẽ lập trình tái cấu trúc được khối LUT để thực hiện chức năng logic của mạch thông qua ô nhớ đầu vào.

b) Công nghệ tái cấu trúc FPGA

Cấu hình của FPGA được hiểu một cách đơn giản nhất là các thông tin quy định về chức năng logic của các khối LB và quy định các khối này được kết nối thế nào với nhau cũng như với các cổng vào ra.



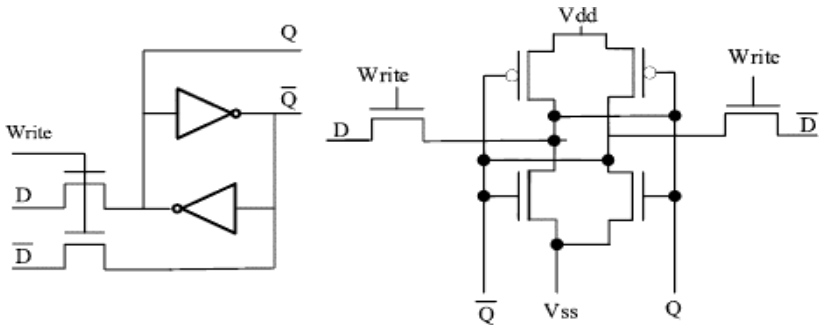
Hình 2.3. Tái cấu hình FPGA

Như hình 2.3, hai đường tín hiệu có thể được nối/ngắt kết nối với nhau bằng một khoá bán dẫn, thường là cổng chuyển tiếp (transmission gate), khoá này điều khiển bởi một bit thông tin được lưu trữ tương tự trong một ô nhớ cấu hình.

Công nghệ được sử dụng cho các ô nhớ cấu hình được hiểu là công nghệ nền tạo nên tính tái cấu trúc FPGA. Trong lịch sử phát triển có khá nhiều công nghệ tái cấu trúc cho FPGA được sử dụng, tuy nhiên có ba công nghệ điển hình là cầu chì ngược, EEPROM, và SRAM trong đó SRAM là công nghệ phổ biến được sử dụng cho hầu hết các FPGA ở thời điểm hiện nay.

Công nghệ SRAM (Static Random Access Memory) là công nghệ được sử dụng trong FPGA hiện đại. SRAM sử dụng thuần tuý các CMOS transistor. Ô nhớ SRAM được cấu tạo từ hai mạch đảo được kết nối với nhau dùng một vòng phản hồi dương đảm bảo hai đầu ra Q và $\overline{Q}$ luôn có mức logic

ngược nhau, ở trạng thái này mạch đạt trạng thái bền vững và lý tưởng, không có dòng điện chạy trong mạch. Ngoài ra có thể các transistor dùng cho việc ghi thông tin vào ô nhớ bằng cách nối với tín hiệu *Data* và *Data*. Mạch đọc không cần thiết trong trường hợp này mà các giá trị *Q* và *Q* có thể được đưa trực tiếp vào điều khiển việc cấu hình kết nối hay logic. Như vậy ô nhớ SRAM trong FPGA được thiết kế tối thiểu với 6 transistor như hình 2.4, SRAM có tốc độ chuyển mạch rất nhanh và tiêu tốn ít năng lượng, dễ dàng tích hợp vào các mạch CMOS vì chỉ cấu tạo từ các transistor. Khả năng đọc/ghi của SRAM cũng gần như vô hạn (10^{15} lần). Tuy nhiên kích thước của SRAM khá lớn nếu so sánh với các công nghệ bộ nhớ khác như Flash, antifuse hay DRAM.



Hình 2.4. Sơ đồ logic của SRAM

Và bảng dưới đây so sánh một số ưu nhược điểm của công nghệ tái cấu hình dùng trong các mạch logic khả trình.

Tính năng	**Flash**	**Antifuse**	**SRAM**
Lưu trữ tĩnh	Có	Có	Không
Tái cấu hình nhiều lần	Có	Không	Có
Kích thước	Vừa (1 transistor)	Nhỏ (tiếp điểm)	Lớn (6 transistor)
Công nghệ	Flash	CMOS+ antifuse	CMOS
Lập trình nóng	Có thể	Không	Có thể
Điện trở tương đương	500-1000	20-100	500-1000
Điện dung ký dinh(fF)	1-2	<1	1-2

Tính tin cậy cấu hình	100	<90	100
Số lần lập trình lại	10^5	1	10^{15}

Bảng 2. 1. Bảng so sánh các tính năng của các công nghệ tái cấu hình

2.1.2. Ngôn ngữ lập trình mô tả phần cứng verilog

a) Verilog là gì?

Verilog HDL là một chuẩn "ngôn ngữ mô tả phần cứng" của IEEE, là một ngôn ngữ dạng text thuần túy được sử dụng để mô tả các mạch Số (thậm chí các mạch Số này có thể được hiện thực thành phần cứng).

Verilog HDL được sử dụng để mô hình hóa phần cứng cho cả mục đích synthesis (tổng hợp) và simulation (mô phỏng).

b) Lịch sử Verilog HDL

Ban đầu Verilog HDL (sau này gọi là Verilog cho tiện) được phát minh bởi Getway Design Automation và được giới thiệu lần đầu tiên vào năm 1984. Năm 1989 được mua lại bởi Cadence, sau đó Verilog được phát hành miễn phí. Tổ chức Open Verilog International (OVI) đã được thành lập nhằm kiểm soát các đặc tả ngôn ngữ. Năm 1995, OVI Verilog đã được thông qua bởi IEEE như là một chuẩn mới của hiệp hội. Năm 2001, IEEE đã giới thiệu phiên bản mới của Verilog và hiện tại đang là phiên bản được sử dụng rộng rãi nhất - Verilog HDL 2001. Năm 2005, SystemVerilog được giới thiệu như là một sự mở rộng của Verilog với các kỹ thuật kiểm tra hướng đối tượng. Năm 2009, IEEE kết hợp chuẩn Verilog với SystemVerilog thành một chuẩn duy nhất: IEEE 1800-2009. Và hiện tại, chuẩn mới nhất là: 1800-2012 - IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language.

c) Các thuật ngữ quan trọng trong verilog

Xuyên suốt chuỗi bài này và các bài viết khác liên quan đến Verilog, bạn có thể bắt gặp nhiều thuật ngữ mới. Có 4 thuật ngữ quan trong sau đây:

- HDL: Là viết tắt của Hardware Description Language (ngôn ngữ mô tả phần cứng) dựa trên ngôn ngữ lập trình mà được sử dụng cho việc mô hình phần cứng. Khi bạn sử dụng HDL, có hai cách để bạn mô hình hoá mạch số của bạn, đó là Behavior Modeling (mô hình hành vi) và Structural Modeling (mô hình cấu trúc).

- RTL: Là viết tắt của Register Transfer Level, là một thuật ngữ thông dụng dùng để mô tả phong cách Behavior Modeling mà định nghĩa mối quan hệ giữa các input và output theo các thao tác data-flow (dòng dữ liệu) bên trong mô hình phần cứng của bạn. Cấu trúc RTL là có thể tổng hợp được. Synthesis (tổng hợp) đề cập đến cả việc chuyển đổi và việc tối ưu HDL code thành mạch số theo một công nghệ cụ thể. Một số ví dụ điển hình là Lookup Table và Flip-Flop của FPGA. RTL Synthesis đơn giản chỉ đề cập đến việc chuyển đổi mô hình RTL.

- Behavior Modeling: Là một thành phần được mô tả bởi đáp ứng Input/output của nó.

- Structural Modeling: Là một thành phần được mô tả bởi các kết nối mức thấp giữa các thành phần con của mạch.

2.2. Giới thiệu về RISC-V

2.2.1. Tổng quan về RISC

RISC (Reduced Instructions Set Computer - Máy tính với tập lệnh đơn giản hóa) 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 lênh đều như nhau. Hiện nay các bộ vi xử lý RISC phổ biến là ARM, SuperH, MIPS, SPARC, DEC Alpha, PA-RISC, PIC, và PowerPC của IBM.

CISC (Complex Instructions Set Computer – Máy tính với tập lệnh phức tạp) là những tập lệnh phức tạp trong CPU tạo lên những máy tính với tập lệnh phức tạp. Trong đó mỗi lệnh có thể thực hiện một số hoạt động cấp thấp chẳng hạn phép toán số học và một bộ nhớ để lưu trữ.

Ta có thể định nghĩa mạch xử lý RISC bởi các tính chất sau:

- Có một số ít lệnh (thông thường dưới 100 lệnh).
- Có một số ít các kiểu định vị (thông thường hai kiểu: định vị tức thì và định vị gián tiếp thông qua một thanh ghi).
- Có một số ít dạng lệnh (một hoặc hai).
- Các lệnh đều có cùng chiều dài.
- Chỉ có các lệnh ghi hoặc đọc ô nhớ mới thâm nhập vào bộ nhớ.
- Dùng bộ tạo tín hiệu điều khiển bằng mạch điện để tránh chu kỳ giải

mã các vi lệnh làm cho thời gian thực hiện lệnh kéo dài.

- Bộ xử lý RISC có nhiều thanh ghi để giảm bớt việc thâm nhập vào bộ nhớ trong.
- Ngoài ra các bộ xử lý RISC đầu tiên thực hiện tất cả các lệnh trong một chu kỳ máy.

Bộ xử lý RISC có các ưu điểm sau: Diện tích của bộ xử lý dùng cho bộ điều khiển giảm từ 60% (cho các bộ xử lý CISC) xuống còn 10% (cho các bộ xử lý RISC). Như vậy có thể tích hợp thêm vào bên trong bộ xử lý các thanh ghi, các cổng vào ra và bộ nhớ cache. Tốc độ tính toán cao nhờ vào việc giải mã lệnh đơn giản, nhờ có nhiều thanh ghi (ít thâm nhập bộ nhớ), và nhờ thực hiện kỹ thuật ống dẫn liên tục và có hiệu quả (các lệnh đều có thời gian thực hiện giống nhau và có cùng dạng). Thời gian cần thiết để thiết kế bộ điều khiển là ít. Điều này góp phần làm giảm chi phí thiết kế. Bộ điều khiển trở nên đơn giản và gọn làm cho ít rủi ro mắc phải sai sót mà ta gặp thường trong bộ điều khiển.

Nhược điểm của RISC: Các chương trình dài ra so với chương trình viết cho bộ xử lý CISC. Điều này do các nguyên nhân sau: Cấm thâm nhập bộ nhớ đối với tất cả các lệnh ngoại trừ các lệnh đọc và ghi vào bộ nhớ. Do đó ta buộc phải dùng nhiều lệnh để làm một công việc nhất định. Cần thiết phải tính các địa chỉ hiệu dụng vì không có nhiều cách định vị. Tập lệnh có ít lệnh nên các lệnh không có sẵn phải được thay thế bằng một chuỗi lệnh của bộ xử lý RISC. Các chương trình dịch gặp nhiều khó khăn vì có ít lệnh làm cho có ít lựa chọn để diễn dịch các cấu trúc của chương trình gốc. Sự cứng nhắc của kỹ thuật đường ống cũng gây khó khăn. Có ít lệnh trợ giúp cho ngôn ngữ cấp cao.

2.2.2. Tổng quan về RISC-V

RISC-V là một kiến trúc tập lệnh tiêu chuẩn mở (ISA) dựa trên các nguyên tắc máy tính tập lệnh giảm (RISC) đã được thiết lập. Không giống như hầu hết các thiết bị ISA khác, RISC-V được cung cấp giấy phép mã nguồn mở miễn phí bản quyền. Nhiều công ty đang cung cấp hoặc đã công

bố phần cứng RISC-V, các hệ điều hành mã nguồn mở có hộ trợ RISC-V có sẵn và tập lệnh được hỗ trợ trong một số chuỗi công cụ phần mềm phổ biến.

Hình 2.5. Mã nguồn mở RISC-V

RISC-V là một kiến trúc RISC, là một kiến trúc lưu trữ-tải (load-store architecture). Câu lệnh dấu phẩy động (floating-point instruction) sử dụng là dấu phẩy động IEEE 754. Các tính năng nổi bật của RISC-V ISA bao gồm: vị trí trường bit lệnh được chọn để đơn giản hoá việc sử dụng bộ ghép kệnh trong CPU, một thiết kế trung lập về mặt kiến trúc và một vị trí cố định cho các bit dấu của các giá trị tức thời (immediate) để tăng tốc mở rộng ký hiệu (sign extension).

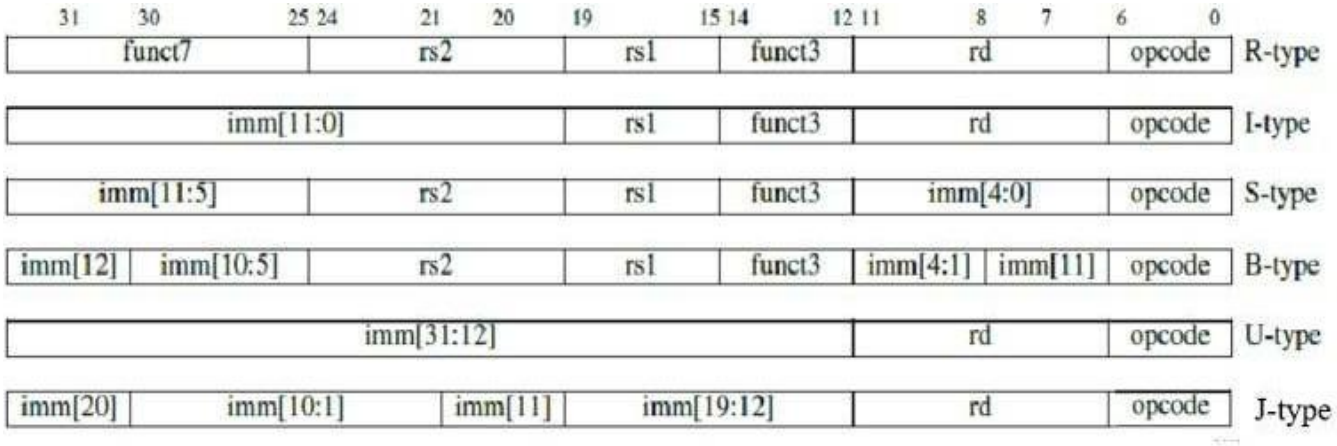
Tập lệnh được thiết kế cho nhiều mục đích sử dụng; tập lệnh cơ sở có độ dài cố định là các lệnh được căn chỉnh tự nhiên 32bit và ISA hỗ trợ các phần mở rộng độ dài thay đổi trong đó mỗi lệnh có thể là độ dài bất kỳ 16bit nào. Tiện ích mở rộng hỗ trợ các hệ thống nhúng nhỏ, máy tính cá nhân, siêu máy tính có bộ xử lý vector và máy tính song song quy mô lớn (WSCs).

2.2.2.1. Tập lệnh RV32I cho vi xử lý RISC-V

Trong đồ án này, em sẽ thiết kế một CPU đơn giản cho RISC-V, thục hiện một số hướng dẫn trong tập lệnh RV32I. Trong tập lệnh RV32I gồm có

6 định dạng lệnh cơ bản:

+ R-format: sử dụng cho các phép tính logic, số học trên thanh ghi.
+ I-Format: sử dụng cho thao tác giữa thanh ghi và một hằng số lưu sẵn trong lệnh (immediate) và dùng tải dữ liệu vào thanh ghi.
+ S-Format: dùng để lưu trữ dữ liệu vào bộ nhớ.
+ B-Format: dùng để thực hiện các câu lệnh rẽ nhánh.
+ U-Format: dùng cho các câu lệnh tức thì 20bit có trọng số cao.
+ J-Format: dùng để thực hiện các câu lệnh nhảy vô điều kiện.



Hình 2.6. Các định dạng lệnh trong RISC V 32I

Trong đó:

- opcode: Mã lệnh, chỉ hoạt động thực thi của câu lệnh.
- rd (Destination Register): Thanh ghi đích, được sử dụng để lưu trữ kết quả của phép tính.
- rs1(Source Register #1): Thanh ghi toán hạng nguồn đầu tiên.
- rs2(Source Register #2): Thanh ghi toán hạng nguồn thứ hai.
- funct3+funct7: kết hợp với opcode để mô tả cách thức hoạt động câu lệnh như thế nào.
- imm (Immediate): giá trị tức thời trong câu lệnh.

a) Tập lệnh R-Format

funct 7	**rs2**	**rs1**	**funct3**	**rd**	**opcode**	
0000000	rs2	rs1	000	rd	0110011	**Add**

0100000	rs2	rs1	000	rd	0110011	**Sub**
0000000	rs2	rs1	001	rd	0110011	**Sll**
0000000	rs2	rs1	010	rd	0110011	**Slt**
0000000	rs2	rs1	011	rd	0110011	**Sltu**
0000000	rs2	rs1	100	rd	0110011	**Xor**
0000000	rs2	rs1	101	rd	0110011	**Srl**
0100000	rs2	rs1	101	rd	0110011	**Sra**
0000000	rs2	rs1	110	rd	0110011	**Or**
0000000	rs2	rs1	111	rd	0110011	**And**

Bảng 2. 2. Bảng tập lệnh của R-Format

Trong định dạng lệnh R-Format thì opcode luôn là 0110011, và để mô tả chính xác hành động thực thi câu lệnh của R-Format thì cần thêm 2 trường bổ sung cho opcode là funct3 và funct7.

Ví dụ cách hoạt động của các câu lệnh:

1	Add a0, a1, a2	a0 = a1+a2
2	Sub a0, a1, a2	a0 = a1-a2
3	Sll a0, a1, a2	a0 = a1 << a2 (dịch trái logic, điền thêm 0 vào bên phải)

4	Slt a0, a1, a2	a1 < a2 (có dấu): True thì a0=1, False thì a0=0
5	Sltu a0, a1, a2	a1 < a2 (không dấu): True thì a0=1, False thì a0=0
6	Xor a0, a1, a2	a0 = a1 xor a2
7	Srl a0, a1, a2	a0 = a1 >> a2 (dịch phải logic, điền thêm 0 vào bên trái)
8	Sra a0, a1, a2	a0 = a1>>a2(dịch phải số học, điền thêm bit dấu của a1 vào bên trái)
9	Or a0, a1, a2	a0 = a1 or a2
10	And a0, a1, a2	a0 = a1 and a2

Bảng 2. 3. Bảng ví dụ thực thi các câu lệnh R-Format

b) Tập lệnh I-Format

imm[11:0]	**rs1**	**funct3**	**rd**	**opcode**	
imm[11:0]	rs1	000	rd	0010011	**Addi**
imm[11:0]	rs1	010	rd	0010011	**Slti**
imm[11:0]	rs1	011	rd	0010011	**Sltiu**
imm[11:0]	rs1	100	rd	0010011	**Xori**
imm[11:0]	rs1	110	rd	0010011	**Ori**
imm[11:0]	rs1	111	rd	0010011	**Andi**
0000000 shamt (5bit)	rs1	001	rd	0010011	**Slli**
0000000 shamt (5bit)	rs1	101	rd	0010011	**Srli**
0100000 shamt (5bit)	rs1	101	rd	0010011	**Srai**

Bảng 2. 4. Bảng tập lệnh của I-Format

Trong định dạng lệnh I-Format thì opcode luôn là 0010011, và để mô tả chính xác hành động thực thi câu lệnh của R-Format thì cần thêm 1 trường bổ sung cho opcode là funct3. Với các câu lệnh dịch bit thì cần xác định giá trị shamt (5bit) sử dụng 5 bit thấp của giá trị immediate để tính số lượng bit cần dịch chuyển (chỉ có thể dịch chuyển theo vị trí từ 0-31).

Ví dụ cách hoạt động của các câu lệnh:

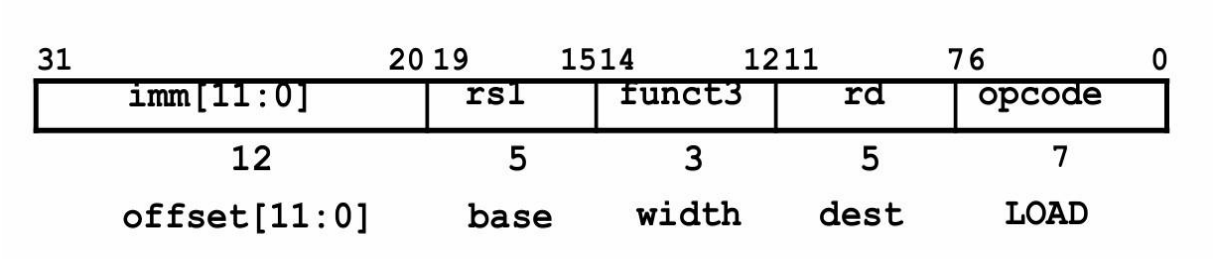
1	Addi a0, a1, 0x05	a0 = a1+0x05
2	Slli a0, a1, 0x05	a0 = a1 << 0x05 (dịch trái logic, điền thêm 0 vào bên phải)
3	Slti a0, a1, 0x05	a1 <<0x05 (có dấu): True thì a0=1, False thì a0=0
4	Sltiu a0, a1, 0x05	a1 << 0x05 (không dấu): True thì a0=1, False thì a0=0
5	Xori a0, a1, 0x05	a0 = a1 xor 0x05
6	Srli a0, a1, 0x05	a0 = a1 >> 0x05 (dịch phải logic, điền thêm 0 vào
		bên trái)

7	Srai a0, a1, 0x05	a0 = a1>>0x05 (dịch phải số học, điền thêm bit dấu của a1 vào bên trái)
8	Ori a0, a1, 0x05	a0 = a1 or 0x05
9	Andi a0, a1, 0x05	a0 = a1 and 0x05

Bảng 2. 5. Bảng ví dụ thực thi các câu lệnh I-Format

❖ Lệnh Load

Câu lệnh Load cũng thuộc định dạng lệnh I-Format. Trong đó 12bit cao nhất imm[11:0] sẽ được định nghĩa là địa chỉ offset[11:0] cùng với đó kết hợp với địa chỉ cơ sở (trong thanh ghi rs1) sẽ tạo thành địa chỉ vật lý của ô nhớ trong bộ nhớ để lấy dữ liệu và sẽ lưu dữ liệu đó vào thanh ghi đích rd. Và trường funct3 sẽ giải mã ra kích thước và có dấu hay không dấu của dữ liệu được tải trong bộ nhớ.



Hình 2.7. Định dạng của câu lệnh Load

offset[11:0]	base	funct3	rd	opcode	
offset[11:0]	base	000	rd	0000011	**Lb**
offset[11:0]	base	001	rd	0000011	**Lh**
offset[11:0]	base	010	rd	0000011	**Lw**
offset[11:0]	base	011	rd	0000011	**Lbu**
offset[11:0]	base	101	rd	0000011	**Lhu**

Bảng 2. 6. Bảng tập lệnh trong lệnh Load

Ví dụ cách hoạt động của các câu lệnh:

1	Lb x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lấy ra **1byte có dấu** lưu vào thanh ghi x14
2	Lh x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lấy ra **2byte có dấu** lưu vào thanh ghi x14

3	Lw x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lấy ra **4byte có dấu** lưu vào thanh ghi x14
4	Lbu x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lấy ra **1byte không dấu** lưu vào thanh ghi x14
5	Lhu x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lấy ra **2byte không dấu** lưu vào thanh ghi x14

Bảng 2. 7. Bảng ví dụ thực thi các lệnh Load

+ Lbu (Load unsigned byte) là tải 1byte không dấu. Do thanh ghi trong RISC-V là các thanh ghi 32bit nên khi lưu vào thanh ghi thì dữ liệu lấy từ bộ nhớ cần được mở rộng (extends) làm đầy thanh ghi 32bit, và ở câu lệnh này thì 24bit trọng số cao là bit 0 và 8bit trọng số thấp là dữ liệu lấy từ bộ nhớ (zero- extends).

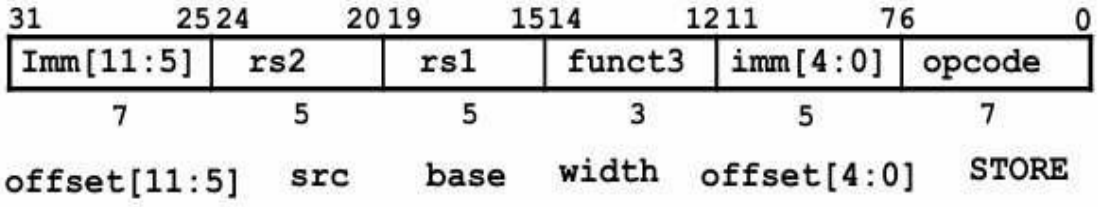
+ Lb (Load byte) là tải 1byte có dấu, tương tự lệnh Lbu thì 24bit trọng số cao là bit dấu của dữ liệu lấy từ bộ nhớ và 8bit trọng số thấp là dữ liệu lấy từ bộ nhớ (sign-extends).

+ Lhu (Load unsigned halfwword) là tải ½ từ (2byte) không dấu, 16bit trọng số cao là bit 0 và 16bit trọng số thấp là dữ liệu lấy từ bộ nhớ (zero-extends).

+ Lh (Load halfword) là tải ½ từ (2byte) có dấu, 16bit trọng số cao là bit dấu của dữ liệu lấy từ bộ nhớ và 16bit trọng số thấp là dữ liệu lấy từ bộ nhớ (sign- extends).

+ Lw (Load word) là tải 1 từ (4byte), trong tập lệnh RV32I không có câu lệnh Lwu bởi vì ta không cần mở rộng khi copy 32bit dữ liệu từ bộ nhớ vào thanh ghi 32bit.

c) Tập lệnh S-Format



Hình 2.8. Định dạng của câu lệnh S-Format

Các câu lệnh thuộc định dạng lệnh S-Format sẽ lưu trữ dữ liệu tù thanh ghi nguồn vào trong bộ nhớ. Do đó lệnh lưu trữ cần phải đọc 2 thanh ghi, thanh ghi rst1 sẽ là địa chỉ cơ sở của bộ nhớ, thanh ghi rs2 là thanh ghi nguồn, dữ liệu trong rs2 sẽ được lưu trữ vào trong bộ nhớ. Điểm lưu ý ở trong S-Format đó là lệnh lưu trữ không viết giá trị vào tập thanh ghi nên không cần có rd trong câu lệnh. Khi thiết kế RISC-V thì 5bit thấp của giá trị tức thời (immediate) sẽ được chuyển đến trường rd, giữ nguyên vị trí trường rs1/rs2.

offset[11:5]	**src**	**base**	**funct3**	**offset[4:0]**	**opcode**	
offset[11:5]	src	base	000	offset[4:0]	0100011	**Sb**
offset[11:5]	src	base	001	offset[4:0]	0100011	**Sh**
offset[11:5]	src	base	010	offset[4:0]	0100011	**Sw**

Bảng 2. 8. Bảng câu lệnh trong định dạng lệnh S-Format Ví dụ về cách hoạt động của các câu lệnh:

1	Sb x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lưu **1byte** từ thanh ghi nguồn x14 vào bộ nhớ theo địa chỉ tính được
2	Sh x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lưu **2byte** từ thanh ghi nguồn x14 vào bộ nhớ theo địa chỉ tính được
3	Sw x14, 8(x2)	Giá trị trong thanh ghi x2(base) cộng thêm 8(offset) tạo ra địa chỉ vật lý, từ đó lưu **4byte** từ thanh ghi nguồn x14 vào bộ nhớ theo địa chỉ tính được

d) Tập lệnh B-Format

Các tập lệnh rẽ nhánh (nhảy có điều kiện) cần đọc dữ liệu từ 2 thanh ghi nhưng không lưu trữ vào thanh ghi nào cả. Khi thực hiện lệnh rẽ nhánh,

giá trị của thanh ghi PC sẽ được cộng thêm giá trị immediate (có giá trị từ -4096 đến 2094). B-Format gần giống với S-Format, với hai thanh ghi nguồn rs1/rs2 và 12bit giá trị tức thời (imm[12:1]), 12bit của giá trị tức thời được mã hoá thành địa chỉ offset bằng cách mở rộng giữ nguyên dấu 13bit (bit imm[0] luôn bằng 0 nên không cần lưu trữ bit này).

imm[12] | [10:5]	**rs2**	**rs1**	**funct3**	**imm[4:1] | [11]**	**opcode**	
imm[12] | [10:5]	rs2	rs1	000	imm[4:1] | [11]	1100011	**Beq**
imm[12] | [10:5]	rs2	rs1	001	imm[4:1] | [11]	1100011	**Bne**
imm[12] | [10:5]	rs2	rs1	100	imm[4:1] | [11]	1100011	**Blt**
imm[12] | [10:5]	rs2	rs1	101	imm[4:1] | [11]	1100011	**Bge**
imm[12] | [10:5]	rs2	rs1	110	imm[4:1] | [11]	1100011	**Bltu**
imm[12] | [10:5]	rs2	rs1	111	imm[4:1] | [11]	1100011	**Bgeu**

Bảng 2. 9. Bảng các lệnh của B-Format

Ví dụ về cách hoạt động của các câu lệnh:

1	Beq x1, x2, Lable	Nếu x1 = x2 thì rẽ nhánh đến nhãn Lable
2	Bne x1, x2, Lable	Nếu x1 ≠ x2 thì rẽ nhánh đến nhãn Lable
3	Bblt x1, x2, Lable	Nếu x1 < x2 thì rẽ nhánh đến nhãn Lable
4	Bge x1, x2, Lable	Nếu x1 ≥ x2 thì rẽ nhánh đến nhãn Lable
5	Bltu x1, x2, Lable	Nếu x1 < x2 thì rẽ nhánh đến nhãn Lable (không dấu)
6	Bgeu x1, x2, Lable	Nếu x1 ≥ x2 thì rẽ nhánh đến nhãn Lable (không dấu)

Bảng 2. 10. Bảng ví dụ về thực thi các câu lệnh

e) Tập lệnh U-Format

U-Format dùng cho câu lệnh giá trị tức thời bit trọng số cao, tập lệnh U- Format có 20bit giá trị tức thời (imm[31:12]) là 20bit có trọng số cao của câu lệnh 32bit, có 1 thanh ghi đích rd. Tập lệnh U-Format thường sử dụng 2 câu lệnh:

+ Lui (Load upper immediate): ghi 20bit cao của thanh ghi đích là giá trị tức thời và xoá 12bit thấp, cùng với câu lệnh Addi sẽ đặt 12bit thấp, có thể tạo bất kỳ gị trị 32bit trong thanh ghi khi sử dụng 2 câu lệnh này(Lui/Addi).
+ Auipc (Adds upper immediate value to PC): Thêm giá trị tức thời có trọng số cao vào thanh ghi PC và đặt kết quả vào thanh ghi đích. Câu lệnh này được sử dụng để đánh địa chỉ tương đối PC.

imm[31:12]	rd	opcode	
imm[31:12]	rd	0110111	**Lui**
imm[31:12]	rd	0010111	**Auipc**

Bảng 2. 11. Bảng các câu lệnh của U-Format

f) Tập lệnh J-Format

imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	
offset[20:1]				rd	1101111	**Jal**
imm[11:0]	**rs1**		**funct3**	**rd**	**opcode**	
offset[11:0]	base		000	dest	1100111	**Jalr**

Bảng 2. 12. Bảng các câu lệnh của J-Format

Các câu lệnh nhảy vô điều kiện được tập hợp trong J-Format. Và có 2 câu lệnh được sử dụng để nhảy là lệnh Jal và Jalr.

+ Lệnh Jal sẽ lưu giá trị PC+4 vào thanh ghi đích rd (địa chỉ trả về), người biên dịch lệnh nhảy sẽ đặt thanh ghi nguồn rd =x0 để loại bỏ địa chỉ trả về; đồng thời đặt giá trị PC= PC+ offset (lệnh nhảy tới PC tương đối).

+ Lệnh Jalr sẽ lưu giá trị PC+4 vào thanh ghi đích rd (địa chỉ trả về), người biên dịch lệnh nhảy sẽ đặt thanh ghi nguồn rd =x0 để loại bỏ địa chỉ trả về; đồng thời đặt giá trị PC= rs1+immediate.

2.2.2.2. Các thành phần trong RISC-V

a) Bộ xử lý

Bộ xử lý CPU- Processor) là một phần quan trọng của máy tính và các thiết bị điện tử khác. Nó thực hiện các phép tính và điều khiển các hoạt động của hệ thống.

Dưới đây là một số điểm tổng quan về bộ xử lý:

1. Chức năng chính: Bộ xử lý thực hiện các phép tính logic và toán học cơ bản, điều khiển và thực hiện các lệnh từ các chương trình và ứng dụng.

2. Cấu trúc: Bộ xử lý thường bao gồm một hoặc nhiều nhân xử lý (cores), bộ nhớ đệm (cache), bộ điều khiển (control unit), và bộ nhớ chính (main memory). Mỗi nhân xử lý thực hiện các tác vụ tính toán riêng biệt đồng thời.

3. Tốc độ xử lý: Tốc độ xử lý của một bộ xử lý được đo bằng Hz (Hertz), tương ứng với số lần mà nó có thể thực hiện một phép tính trong mỗi giây. Các bộ xử lý hiện đại thường có tốc độ đo bằng GHz (Gigahertz).

4. Kiến trúc: Có nhiều loại kiến trúc bộ xử lý, bao gồm kiến trúc RISC (Reduced Instruction Set Computing) và kiến trúc CISC (Complex Instruction Set Computing). Kiến trúc này đề cập đến cách mà các lệnh được thực hiện và xử lý bên trong bộ xử lý.

5. Hiệu suất: Hiệu suất của một bộ xử lý phụ thuộc vào nhiều yếu tố như tốc độ xử lý, số lượng nhân xử lý, kích thước bộ nhớ đệm và kiến trúc.

6. Tiến triển công nghệ: Công nghệ bộ xử lý liên tục phát triển, với các phiên bản mới có hiệu suất cao hơn, tiêu thụ điện năng thấp hơn và tính linh hoạt cao hơn.

7. Ứng dụng: Bộ xử lý được sử dụng trong mọi thiết bị điện tử từ máy tính cá nhân, máy tính xách tay, máy chủ, điện thoại thông minh cho đến thiết bị nhúng trong các sản phẩm điện tử tiêu dùng.

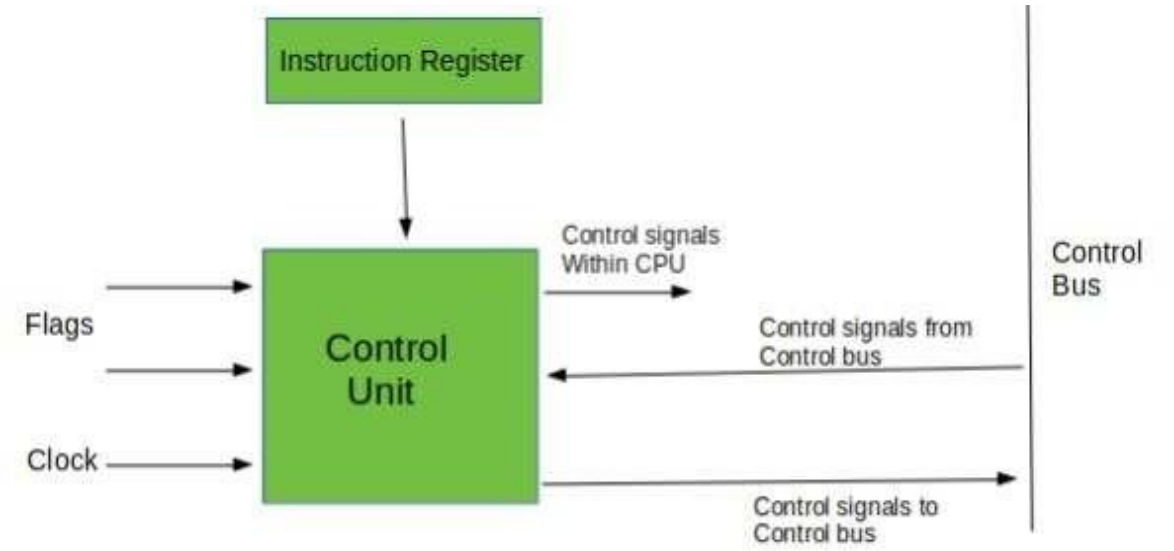
Tóm lại, bộ xử lý là trái tim của mọi hệ thống máy tính và đóng vai trò quan trọng trong việc thực hiện các tác vụ tính toán và điều khiển.

b) Bộ điều khiển

Bộ điều khiển là một phần của bộ xử lý trung tâm (CPU) của máy tính, chỉ đạo hoạt động của bộ xử lý. Đơn vị điều khiển có trách nhiệm cho bộ nhớ, đơn vị số học/logic và các thiết bị đầu vào và đầu ra của máy tính biết cách phản hồi các hướng dẫn đã được gửi đến bộ xử lý.

Nó tìm nạp các lệnh bên trong của các chương trình từ bộ nhớ chính đến thanh ghi lệnh của bộ xử lý và dựa trên nội dung thanh ghi này, bộ điều khiển tạo ra tín hiệu điều khiển giám sát việc thực hiện các lệnh này. Một đơn vị điều khiển hoạt động bằng cách nhận thông tin đầu vào mà nó chuyển đổi thành tín hiệu điều khiển, sau đó được gửi đến bộ xử lý trung tâm. Bộ xử lý của máy tính sau đó cho phần cứng kèm theo biết những hoạt động cần thực hiện. Các chức năng mà bộ điều khiển thực hiện phụ thuộc vào loại CPU vì kiến trúc của CPU thay đổi từ nhà sản xuất này sang nhà sản xuất

khác.



Hình 2.9. Sơ đồ khối của bộ điều khiển

Chức năng của bộ điều khiển:

+ Nó điều phối chuỗi các chuyển động dữ liệu vào, ra và giữa nhiều đơn vị con của bộ xử lý.
  + Nó giải thích các tập lệnh.
  + Nó kiểm soát luồng dữ liệu bên trong bộ xử lý.
+ Nó nhận được các lệnh hoặc lệnh bên ngoài mà nó chuyển đổi thành chuỗi tín hiệu điều khiển.
+ Nó kiểm soát nhiều đơn vị thực thi (tức là ALU, bộ đệm dữ liệu và thanh ghi) chứa trong CPU.
+ Nó cũng xử lý nhiều tác vụ, chẳng hạn như tìm nạp, giải mã, xử lý thực thi và lưu trữ kết quả.

c) Datapath

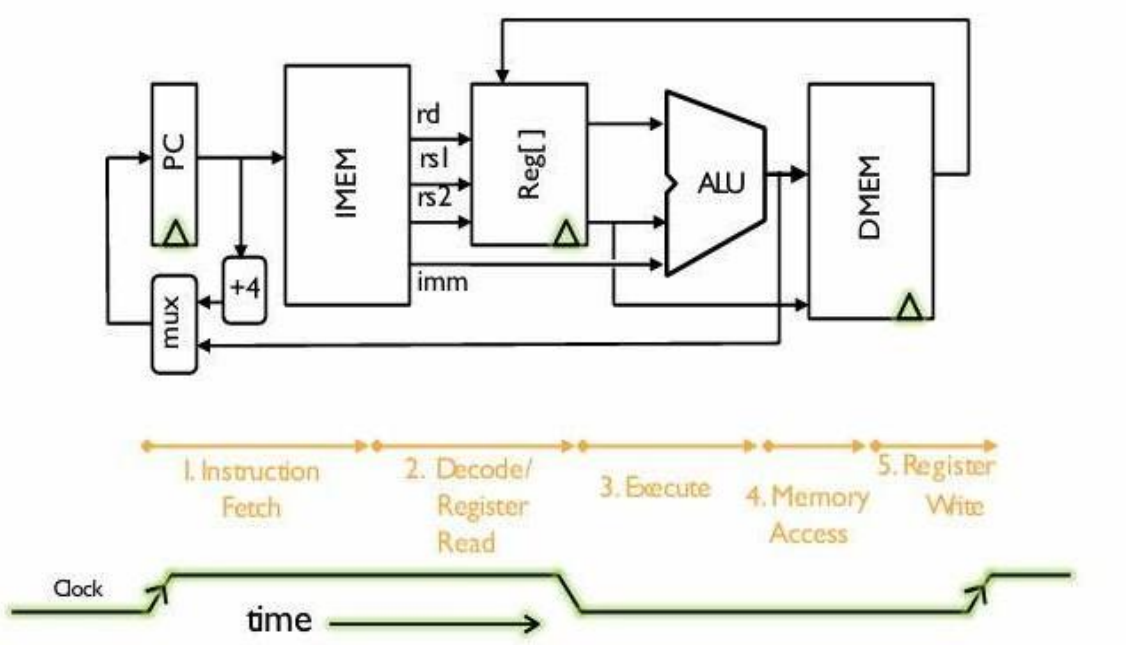
Datapath- Đường dẫn dữ liệu là một phần của bộ vi xử lý (Processor) chứa phần cứng cần thiết để thực hiện các hoạt động theo yêu cầu của bộ xử lý. Datapath của RISC-V được thiết kế để thực hiện tất cả các lệnh RISC-V trong một chu kỳ duy nhất. Tuy nhiên, không phải tất cả các thành phần phần cứng đều được sử dụng bởi mọi lệnh. Một số lệnh chỉ kích hoạt các thành phần cụ thể. Khi thực thi một lệnh chúng ta có thể chia quá trình đó thành

nhiều giai đoạn, sau đó kết nối các giai đoạn để tạo toàn bộ đường dẫn dữ liệu (Datapath). Với các làm này, việc thiết kế sẽ dễ dàng hơn, có thể dễ dàng tối ưu hoá (thay đổi) một giai đoạn nà không cần đụng chạm vào các giai đoạn khác (đây được gọi là tính modul).

Đường dẫn dữ liệu trong RISC-V được chia làm 5 giai đoạn:

- IF (Instruction Fetch): Lấy lệnh từ bộ nhớ.
- ID (Instruction Decode): Giải mã lệnh và xác định thao tác cần thực hiện.
- EX (Execute): Thực hiện thao tác (ví dụ: tính toán ALU).
- MEM (Memory): Truy cập bộ nhớ (ví dụ: lệnh load/store).
- WB (Write Back): Ghi kết quả trở lại một thanh ghi.

Điểm lưu ý ở đây là không phải tất cả các lệnh đều hoạt động trong tất cả các giai đoạn (trừ lệnh load).



Hình 2.10. Các giai đoạn cơ bản trong thực thi lệnh

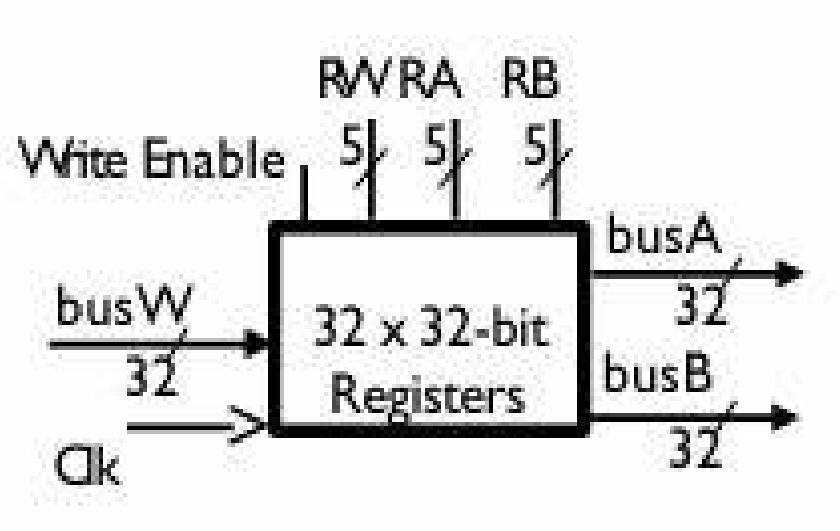
d) Tệp thanh ghi 32bit

Tên thanh ghi	**Tên tượng trưng**	**Miêu tả**

x0	zero	Luôn bằng không
x1	ra	Địa chỉ trả về
x2	sp	Con trỏ ngăn xếp
x3	gp	Con trỏ toàn cục
x4	tp	Con trỏ luồng
x5	t0	Địa chỉ trả về tạm thời / thay thế
x6–7	t1–2	Tạm thời
x8	s0 / fp	Lưu vào thanh ghi / con trỏ khung
x9	s1	Lưu vào thanh ghi
x10–11	a0–1	Đối số hàm / giá trị trả về
x12–17	a2–7	Đối số hàm
x18–27	s2–11	Lưu vào thanh ghi
x28–31	t3–6	Tạm thời

Bảng 2.13. Tập thanh ghi 32bit

Trong RISC-V có tập thanh ghi (regfile, RF) bao gồm 32 thanh ghi, mỗi thanh ghi có 32bit để lưu trữ. Một tập thanh ghi có 2 đường bus dữ liệu ra 32bit: bus A và bus B, một đường bus dữ liệu vào 32bit: bus W.



Hình 2.12. Cấu tạo của tệp thanh ghi 32bit

Thanh ghi được chọn lựa bởi các tín hiệu RA, RB, RW, trong đó:

- RA chọn thanh ghi sẽ tải dữ liệu ra bus A.
- RB chọn thanh ghi sẽ tải dữ liệu ra bus B.
- RW chọn thanh ghi sẽ lưu trữ dữ liệu từ bus W khi giá trị Write Enable là 1.

Xung đồng hồ (clk) là tín hiệu vào, là yếu tố chỉ trong quá trình ghi hoạt động mới dùng tới; còn trong quá trình đọc, tệp thanh ghi hoạt động như một khối logic tổ hợp.

e) Bộ đếm chương trình PC

Bộ đếm chương trình, còn được gọi là con trỏ lệnh hoặc đơn giản là PC, là một thành phần cơ bản của bộ xử lý trung tâm (CPU) của máy tính. Nó là một thanh ghi đặc biệt theo dõi địa chỉ bộ nhớ của lệnh tiếp theo sẽ được thực thi trong một chương trình.

Bộ đếm chương trình rất quan trọng vì nó cho phép bộ xử lý trung tâm (CPU) lấy các lệnh từ bộ nhớ một cách tuần tự. Bằng cách theo dõi địa chỉ của lệnh hiện tại, bộ đếm chương trình đảm bảo rằng CPU biết lệnh nào cần tìm nạp tiếp theo.

Bộ đếm là một trường hợp đặc biệt của bộ cộng tích lũy, giá trị được tăng lên mỗi khi một lệnh được tìm nạp. Nếu ta cho đầu vào của bộ cộng A luôn nhận

giá trị bằng 1 thì sau mỗi xung nhịp giá trị trong thanh ghi tăng thêm 1. Trong trường hợp đếm ngược thì cho giá trị của A bằng - 1. Vì vậy nó luôn trỏ đến địa chỉ của lệnh tiếp theo trong bộ nhớ. Sau khi tìm nạp, bộ đếm chương trình được cập nhật đến địa chỉ của lệnh tiếp theo, cho phép bộ xử lý trung tâm (CPU) tiếp tục thực thi chương trình. Giá trị đếm là giá trị lưu trong thanh ghi còn xung đếm chính là xung nhịp hệ thống.

Khi bộ đếm chương trình được sửa đổi, bộ xử lý trung tâm (CPU) sẽ tìm nạp lệnh từ địa chỉ mới được chỉ định bởi bộ đếm chương trình đã sửa đổi. Điều này cho phép thực thi không tuần tự và cho phép các tính năng như vòng lặp, điều kiện và cuộc gọi hàm bằng ngôn ngữ lập trình.

Bộ đếm chương trình khác với địa chỉ bộ nhớ. Bộ đếm chương trình trỏ đến lệnh tiếp theo sẽ được thực thi, trong khi địa chỉ bộ nhớ đề cập đến một vị trí cụ thể trong bộ nhớ nơi dữ liệu hoặc lệnh được lưu trữ.

Nếu bộ đếm chương trình trỏ đến một địa chỉ không hợp lệ, nó có thể dẫn đến sự cố chương trình hoặc lỗi. Bộ xử lý trung tâm (CPU) có thể cố gắng tìm nạp lệnh từ vị trí bộ nhớ không hợp lệ, dẫn đến hành vi không xác định hoặc ngoại lệ.

f) Khối ALU

Khối ALU, hay Đơn vị Logic Số học (Arithmetic Logic Unit), là một thành phần chính của các đơn vị xử lý trung tâm (CPU) của một hệ thống máy tính. ALU thực hiện tất cả các quá trình liên quan đến phép tính số học và logic mà cần phải được thực hiện trên những lời hướng dẫn. ALU là một khối xây dựng cơ bản của nhiều loại mạch điện toán, bao gồm CPU, FPU (Floating Point Unit - Đơn vị xử lý số thực) và GPU (Graphics Processing Unit - Đơn vị xử lý đồ họa). Một CPU, FPU hoặc GPU có thể chứa nhiều ALU.

Khối ALU (Arithmetic Logic Unit) thực hiện nhiều hoạt động khác nhau bao gồm:

- Phép tính số học: ALU thực hiện các phép tính số học như cộng, trừ, nhân và chia.

- Phép toán logic: ALU thực hiện các phép toán logic như AND, OR, XOR và dịch bit.
- So sánh: ALU có khả năng thực hiện các phép so sánh giữa hai giá trị.

ALU nhận dữ liệu, lựa chọn phép toán, thực hiện phép tính và đưa ra kết quả. Các đầu vào của ALU là dữ liệu được vận hành trên, được gọi là toán hạng và mã cho biết thao tác được thực hiện; và đầu ra của ALU là kết quả của hoạt động được thực hiện. Trong nhiều thiết kế, ALU cũng có đầu vào hoặc đầu ra trạng thái, hoặc cả hai, truyền đạt thông tin về hoạt động trước đó hoặc hoạt động hiện tại, tương ứng, giữa ALU và các thanh ghi trạng thái bên ngoài.

g) Bộ dồn kênh

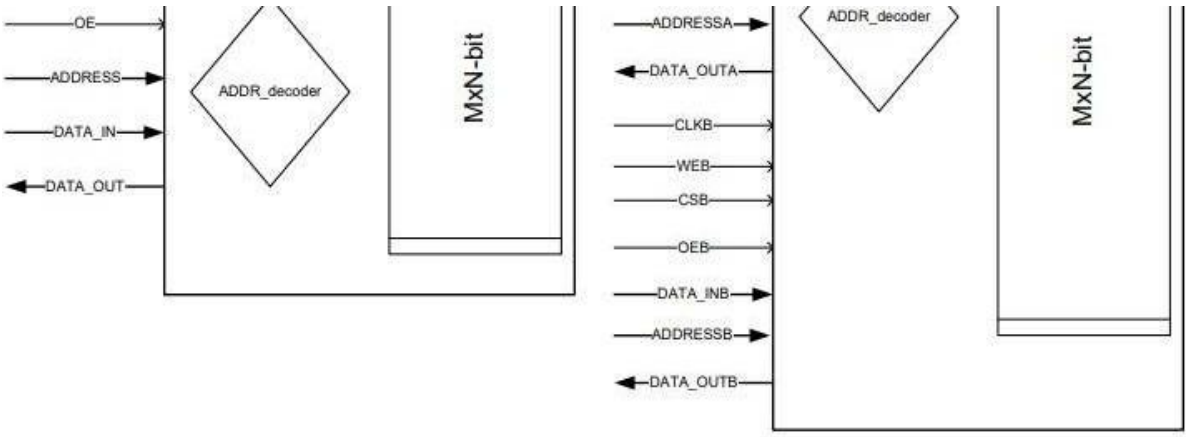
Bộ hợp kênh còn gọi là bộ dồn kênh (hay bộ ghép kênh), nó cũng được gọi là bộ chọn dữ liệu (Data Selector). Chức năng logic cơ bản của bộ hợp kênh là dưới sự điều khiển của tín hiệu chọn (n đầu vào điều khiển) thực hiện chọn ra kênh nào đó (trong số 2n kênh đầu vào) để nối thông tín hiệu đầu vào được chọn đến đầu ra.

Để người dùng không bị nhầm lẫn trong việc xác định địa chỉ kênh, các nhà sản xuất vi mạch đã dùng các chỉ số kênh 0, 1, 2, ... trùng với giá trị thập phân của tổ hợp nhị phân tương ứng của các đầu vào điều khiển.

h) Bộ nhớ

Bộ nhớ dùng trong RISC-V là nơi lưu trữ lệnh và dữ liệu trong một không gian bộ nhớ 32bit có đại chỉ là 1byte. Ở trong kiến trúc vi xử lý RISC-V sẽ sử dụng bộ nhớ chuyên biệt cho lưu trữ lệnh (IMEM) và lưu trữ dữ liệu (DMEM). Những câu lệnh sẽ được đọc (tìm nạp) từ bộ nhớ lệnh (ở đây là IMEM chỉ đọc), còn với câu lệnh tải hoặc lưu trữ sẽ truy cập bộ nhớ dữ liệu (DMEM). Trong kiến trúc RISC-V của bài đồ án này, chúng em dùng bộ nhớ RAM để thiết kế.

RAM (Random Access Memory) là một phần tử rất hay được sử dụng trong thiết kế các hệ thống số. RAM có thể phân loại theo số lượng cổng và cách thức làm việc đồng bộ hay không đồng bộ của các thao tác đọc và ghi.



Single-port RAM Dual-port RAM

- Single port RAM là RAM chỉ có một kênh đọc và ghi, một đường vào địa chỉ, các động tác đọc ghi trên kênh này chỉ có thể thực hiện lần lượt.

- Dual-port RAM là RAM có hai kênh đọc ghi riêng biệt tương ứng hai kênh địa chỉ, các kênh đọc ghi này có thể dùng chung xung nhịp đồng bộ cũng có thể không dùng chung. Đối với Dual-port RAM có thể đọc và ghi đồng thời trên hai kênh.

- Asynchronous RAM- RAM không đồng bộ là RAM thực hiện thao tác đọc hoặc ghi không đồng bộ, thời gian kể từ khi có các tín hiệu yêu cầu đọc ghi cho tới khi thao tác thực hiện xong thuần túy là trễ tổ hợp.

Kết hợp cả hai đặc điểm trên có thể tạo thành nhiều kiểu RAM khác nhau, ví dụ single-port RAM synchronous READ synchronous WRITE nghĩa là RAM một cổng đọc ghi đồng bộ, hay Dual-port RAM synchronous WRITE asynchronous READ là RAM hai cổng ghi đồng bộ đọc không đồng bộ… Khối RAM được cấu thành từ hai bộ phận là khối giải mã địa chỉ và dãy các thanh ghi, khối giải mã địa chỉ sẽ đọc địa chỉ và quyết định sẽ truy cập tới vị trí thanh ghi nào để thực hiện thao tác đọc hoặc ghi. Kích thước của khối RAM thường được ký hiệu là Mx N-bit trong đó M là số lượng thanh ghi, N là số bit trên 1 thanh ghi.

2.3. Kỹ thuật Pipeline và xử lý các xung đột trong Pipeline

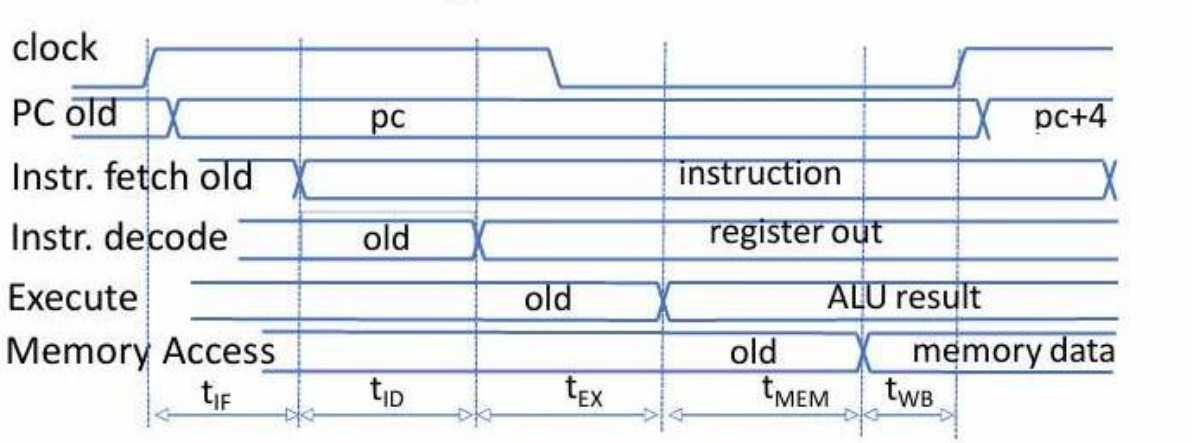
a) Tổng quan về kỹ thuật Pipeline

Kỹ thuật Pipeline- kỹ thuật ống dẫn là một kỹ thuật mà trong đó các lệnh được thực thi theo kiểu chồng lấn lên nhau (overlap). Đây là kỹ thuật

làm cho các giai đoạn khác nhau của nhiều lệnh thực thi cùng một lúc. Tức là bắt đầu thực hiện một lệnh khác trong quá trình thực hiện một lệnh, điều này làm cho rút ngắn thời gian thực thi của bộ xử lý.

Trong kiến trúc RISC-V, để thực thi một lệnh cần trải qua 5 giai đoạn: giai đoạn nạp lệnh (ÌF), giai đoạn giải mã lệnh (ID), giai đoạn thực thi (EX), giai đoạn truy cập bộ nhớ (MEM/MA) và giai đoạn viết lại vào thanh ghi(WB). Khi sử dụng kỹ thuật ống dẫn, các giai đoạn này sẽ được thực hiện song song. Một lệnh mới sẽ được lấy trong giai đoạn IF cùng với việc giải mã lệnh của giai đoạn trước đó. Ngay sau đó, lệnh tiếp theo sẽ được giải mã cùng lúc với việc thi hành lệnh trước đó và tiếp tục cho đến khi tất cả các giai đoạn được hoàn thành.

Với bộ xử lý RISC-V, giả sử thời gian thực thi lệnh của các giai đoạn như sau:

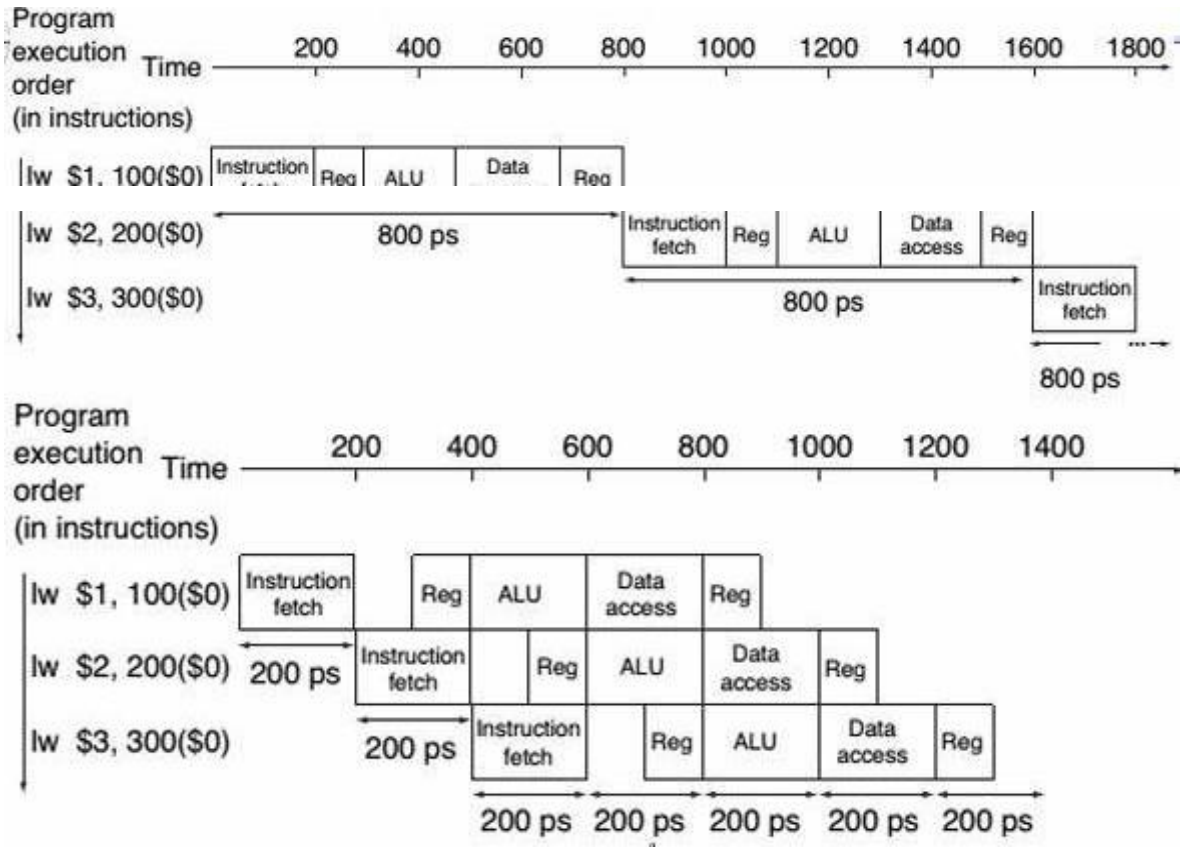


IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	**800 ps**

Hình 2.13. Thời gian thực thi các lệnh của các giai đoạn

Instr	IF = 200ps	ID = 100ps	ALU = 200ps	MEM=200ps	WB = 100ps	Total
add	X	X	X		X	600ps
beq	X	X	X			500ps
jal	X	X	X		X	600ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

Từ bảng trên ta tính được tần số xung clock lớn nhất dùng cho bộ vi xử lý ở chế độ chu kỳ đơn: f_{max} = 1/800ps=1.25Ghz, việc thực thi hoàn tất một câu lệnh mất 800ps.



Thời gian thực thi 3 lệnh Lw theo kiểu chu kỳ đơn mất 3x800=2400ps, hưng ở chế độ Pipeline thì thời gian thực thi hết 3x200=600ps.

Trong thực tế, Pipeline sẽ tăng tốc so với không Pipeline với số lần nhỏ hơn số tầng của Pipeline.

Một số lưu ý với Pipeline tăng tốc so với không Pipeline là kỹ thuật Pipeline không giúp giảm thời gian thực thi của từng lệnh riêng lẻ mà giúp giảm tổng thời gian thực thi của 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 các chương trình chạy trong thực tế thông thường lên đến hàng tỉ lệnh, nên việc sử dụng kỹ thuật Pipeline là giải pháp hợp lý để tăng tốc.

b) Các xung đột trong Pipeline (Pipeline Hazards)

Xung đột là trạng thái mà lệnh tiếp theo không thể thực thi trong chu kỳ Pipeline ngay sau đó (hoặc thực thi nhưng sẽ cho kết quả sai), thường do một trong ba nguyên nhân sau:

- Xung đột cấu trúc (Structural Hazard): là khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ Pipeline của nó do phần cứng không thể hỗ trợ. Nói cách khác, xung đột cấu trúc xảy ra khi có hai lệnh cùng truy xuất vào một tài nguyên phần cứng nào đó cùng một lúc.
- Xung đột dữ liệu (Data Hazard): là khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do dữ liệu mà lệnh này cần vẫn chưa sẵn sàng.
- Xung đột điều khiển (Control Hazard): là khi một lệnh dự kiến không thể thực thi trong đúng chu kỳ pipeline của nó do lệnh nạp vào không phải là lệnh được cần. Xung đột này xảy ra trong trường hợp luồng thực thi chứa các lệnh nhảy.

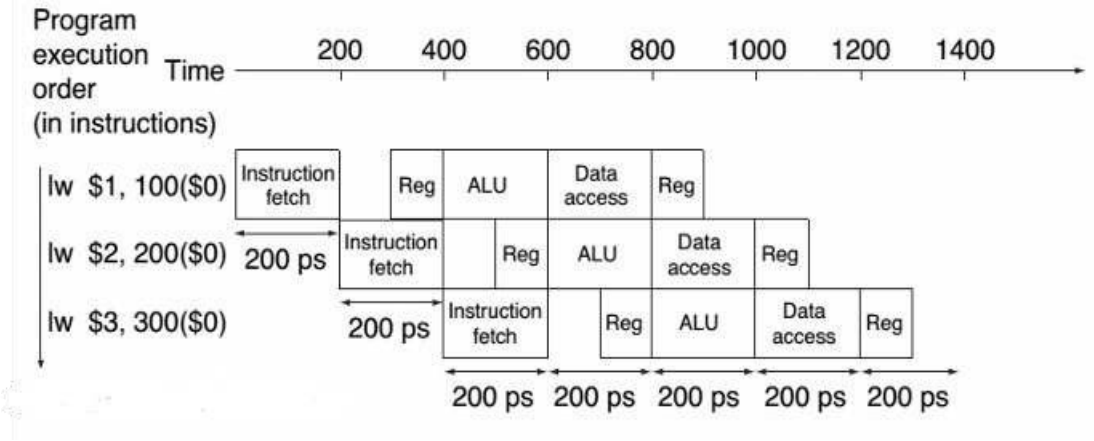
❖ Xung đột cấu trúc

Trong kiến trúc RISC-V, có xung đột cấu trúc tệp thanh ghi và truy cập bộ nhớ.

- Xung đột cấu trúc tệp thanh ghi: Với mỗi câu lệnh có thể đọc từ

hai toán hạng trong giai đoạn giải mã và có thể viết một giá trị ở giai đoạn viết lại vào thanh ghi. Nếu sử dụng viết và đọc toán hạng chung cổng thì sẽ xảy ra xung đột cấu trúc. Để tránh xung đột cấu trúc, chúng ta sẽ thiết kế tệp thanh ghi có các cổng riêng biệt; với hai cổng dùng để đọc và một cổng dùng để viết. Như vậy ba lần truy cập vào tệp thanh ghi có thể xảy ra đồng thời trên mỗi chu kỳ.

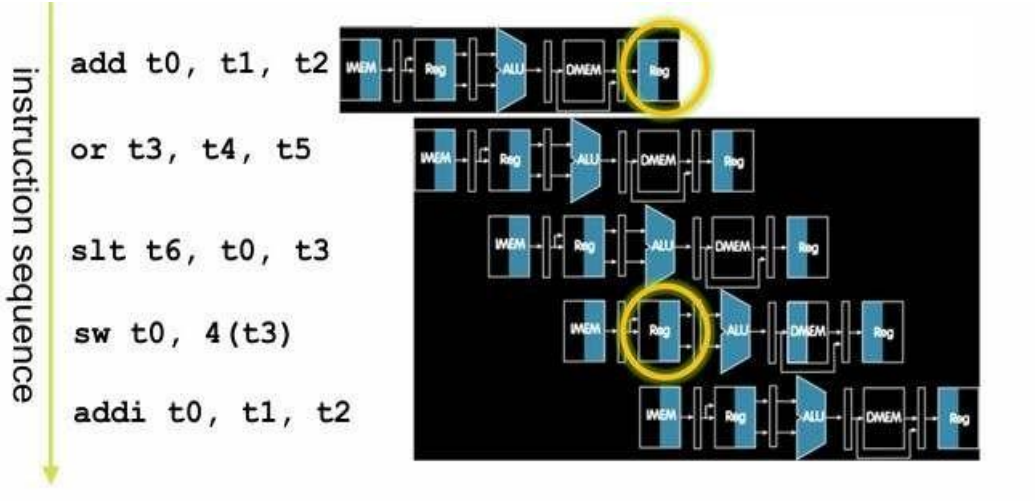
- Xung đột cấu trúc truy cập bộ nhớ: Giả sử rằng chúng ta có một bộ nhớ đơn duy nhất thay vì hai bộ nhớ lệnh và dữ liệu rời rạc nhau. Nếu pipeline trong ví dụ ở hình dưới có thêm lệnh thứ tư thì trong chu kỳ pipeline từ 600 tới 800 khi lệnh thứ nhất thực hiện truy xuất bộ nhớ lấy dữ liệu thì lệnh thứ tư sẽ thực hiện truy xuất bộ nhớ lấy lệnh. Do không có bộ nhớ lệnh và dữ liệu riêng lẻ, trong trường hợp này sẽ có xung đột cấu trúc xảy ra.



Tóm lại, xung đột cấu trúc xảy ra khi sử dụng chung tài nguyên, trong kỹ thuật Pipeline của RISC-V với một bộ nhớ duy nhất, việc tải/lưu trữ đều yêu cầu truy cập dữ liệu. Nếu không có bộ nhớ riêng biệt, quá trình nạp lệnh sẽ phải ở chế độ đợi ở chu kỳ đó. Tất cả các hoạt động khác trong ống dẫn sẽ phải đợi. Từ đó datapath Pipeline yêu cầu cần có bộ nhớ lệnh/ bộ nhớ dữ liệu chuyên biệt. RISC ISA (bao gồm RISC-V) đượ thiết kế để tránh các xung đột về cấu trúc nên chúng ta không cần để ý đến xung đột này.

❖ Xung đột dữ liệu

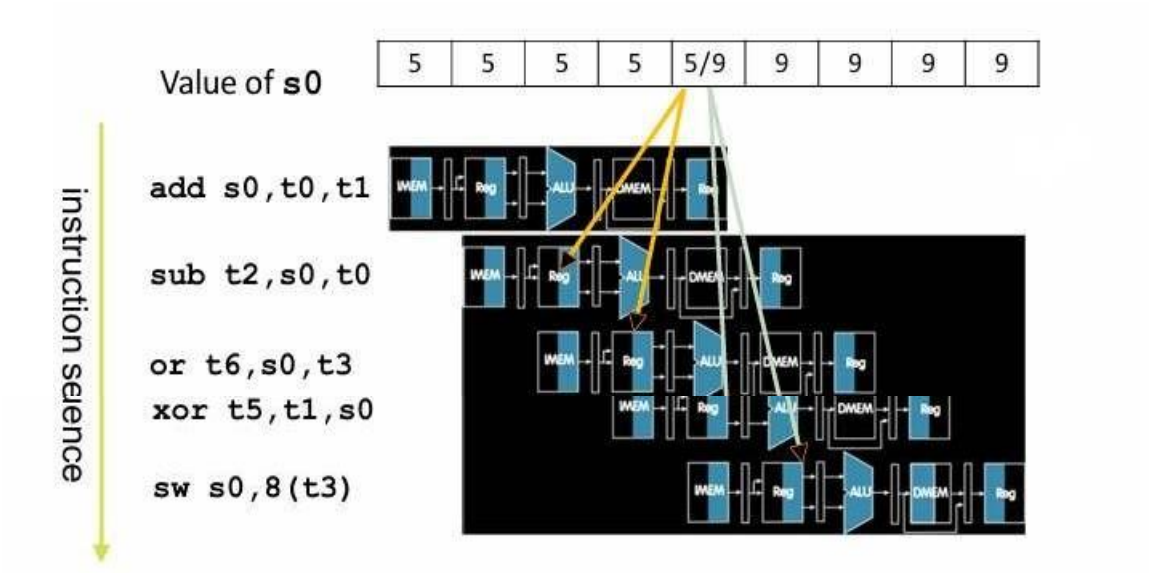
- Xung đột dữ liệu truy cập thanh ghi:



Hình 2.14. Ví dụ về xung đột dữ liệu truy cập tệp thanh ghi

Với ví dụ 4 câu lệnh được thực thi Pipeline trong ví dụ trên, câu lệnh sw sẽ lưu giá trị dữ liệu của thanh ghi t0 là giá trị mới hay giá trị cũ. Có thể không phải lúc nào cũng có thể viết sau đó đọc giá trị tệp thanh ghi trong cùng một chu kỳ, đặc biệt là trong các thiết kế tần số cao, nên để tránh xung đột này ta sẽ để tệp thanh ghi hoạt động viết dữ liệu ở chu kỳ sườn âm xung clock.

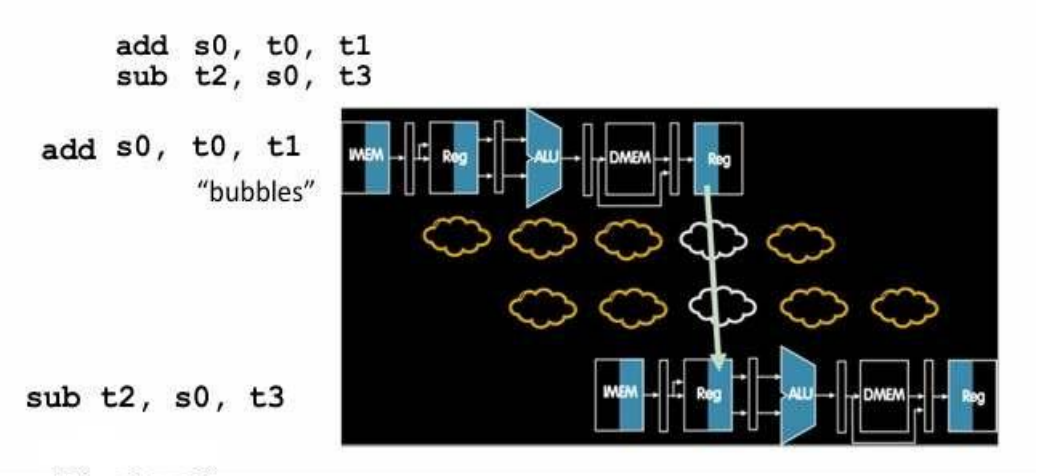
- Xung đột dữ liệu kết quả của ALU:



Trong công đoạn ID, lệnh sub sẽ cần đọc giá trị của thanh ghi s0, trong khi đó giá trị mới của thanh ghi s0 phải tới công đoạn WB của lệnh add mới sẵn

sàng. Vì vậy, nếu thực hiện pipeline thông thường, trường hợp này sẽ xảy ra xung đột dữ liệu. Tương tự với lệnh or cũng giống với lệnh sub. Để tránh xung đột này chúng ta có hai giải pháp:

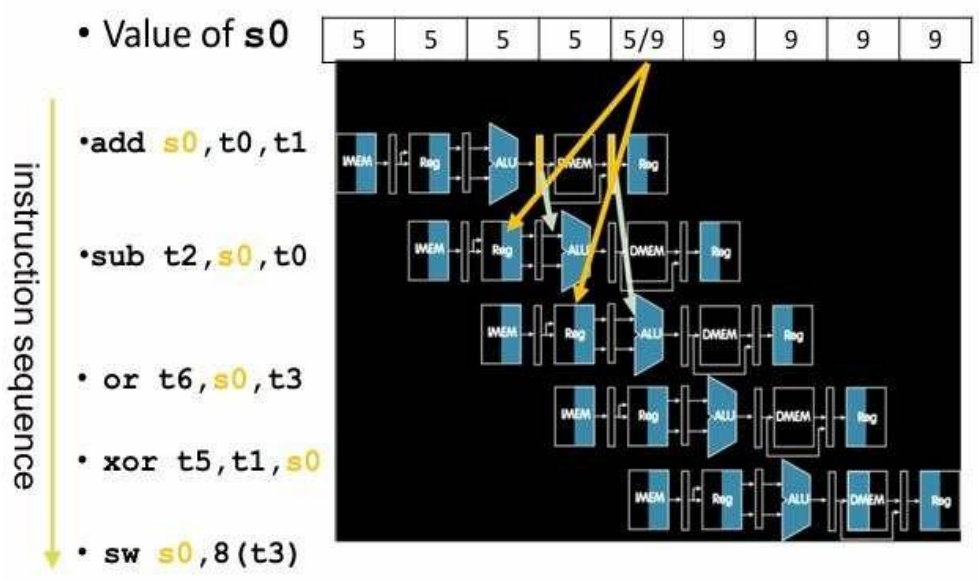
+ Giải pháp thứ nhất: Thêm chế độ chờ, chờ thêm hai chu kỳ xung clock thì lệnh sub mới được thực thi công đoạn tìm nạp lệnh IF. Nhưng giải pháp này sẽ làm giảm hiệu suất của Pipeline



Hình 2.15. Giải pháp thêm chế độ chờ

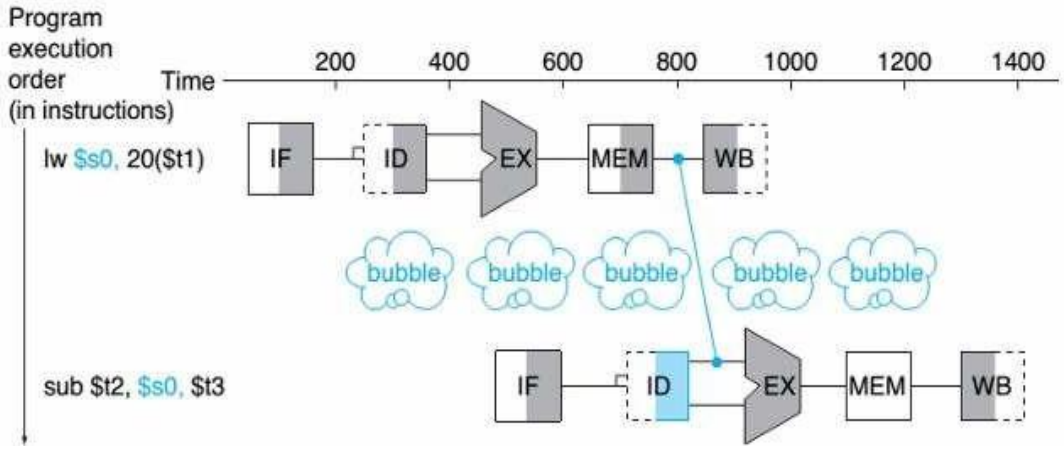
+ Giải pháp thứ hai: kỹ thuật nhìn trước (Forwarding/Bypassing), Thay vì chờ một số chu kỳ đến khi dữ liệu cần sẵn sàng, một kỹ thuật có thể được áp dụng để rút ngắn số chu kỳ rỗi, gọi là kỹ thuật nhìn trước. Như trong ví dụ trước, thay vì chờ sau hai chu kỳ rỗi mới nạp lệnh sub vào, ngay khi ALU hoàn thành tính toán tổng cho lệnh add thì tổng này cũng được cung cấp ngay cho công đoạn EX của lệnh sub (thông qua một bộ đệm dữ liệu gắn thêm bên trong- hay là ống dẫn) để ALU tính toán kết quả cho sub nhanh. Và tương tự lệnh or, thì kết quả của ALU sẽ được lưu trong ống dẫn công đoạn MEM sẽ cung cấp ngay

ho công đoạn EX của lệnh or. Kỹ thuật nhìn trước trong Pipeline sẽ lấy dữ liệu trong ống dẫn thay vì lấy ở tệp thanh ghi.



Hình 2.16. Giải pháp kỹ thuật nhìn trước

Lưu ý, với lệnh lw và các lệnh có chức năng tương tự, thông thường kết quả cuối của nó không phải khi hoàn tất công đoạn EX mà là khi hoàn tất công đoạn MEM. Với lệnh lw, dữ liệu mong muốn sẽ chỉ sẵn sàng sau 4 chu kỳ pipeline (tức sau khi công đoạn MEM hoàn tất). Vì vậy, giả sử dữ liệu đầu ra của công đoạn MEM của lệnh lw được truyền tới đầu vào công đoạn EX của lệnh sub theo sau, thì lệnh sub vẫn phải chờ sau một chu kỳ rỗi mới được nạp vào.



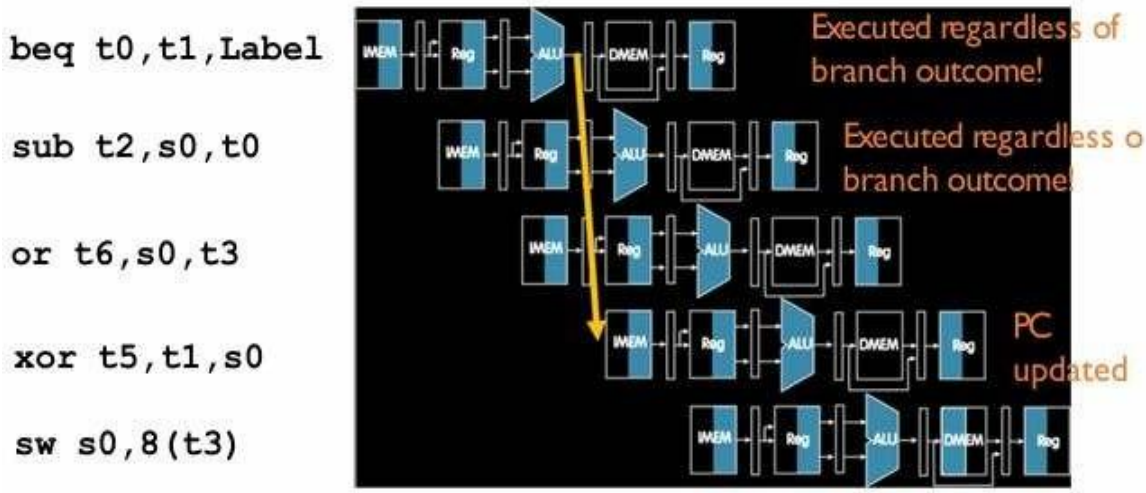
Hình 2.17. Kỹ thuật nhìn trước với lệnh lw

Kỹ thuật forwarding có thể hỗ trợ giải quyết xung đột dữ liệu hiệu quả, tuy nhiên nó không thể ngăn chặn tất cả các trường hợp chu kỳ rỗi.

Tóm lại, với kỹ thuật forwarding có: ALU-ALU forwarding hay EX- EX forwarding và MEM-ALU forwarding hay MEM-EX forwarding.

❖ Xung đột điều khiển

Một số lệnh nhảy có điều kiện và không điều kiện trong RISC-V (branches, jumps) tạo ra xung đột điều kiển này.



Hình 2.18. Ví dụ về xung đột điều khiển

Nếu áp dụng pipeline thông thường, tại chu kỳ thứ ba của pipeline, khi beq đang thực thi công đoạn ID thì lệnh sub sẽ được nạp vào. Nhưng nếu điều kiện bằng của lệnh beq xảy ra thì lệnh thực hiệp tiếp sau đó không phải là sub mà là lệnh được gán nhãn ‘label’, lúc này xảy ra xung đột điều khiển.

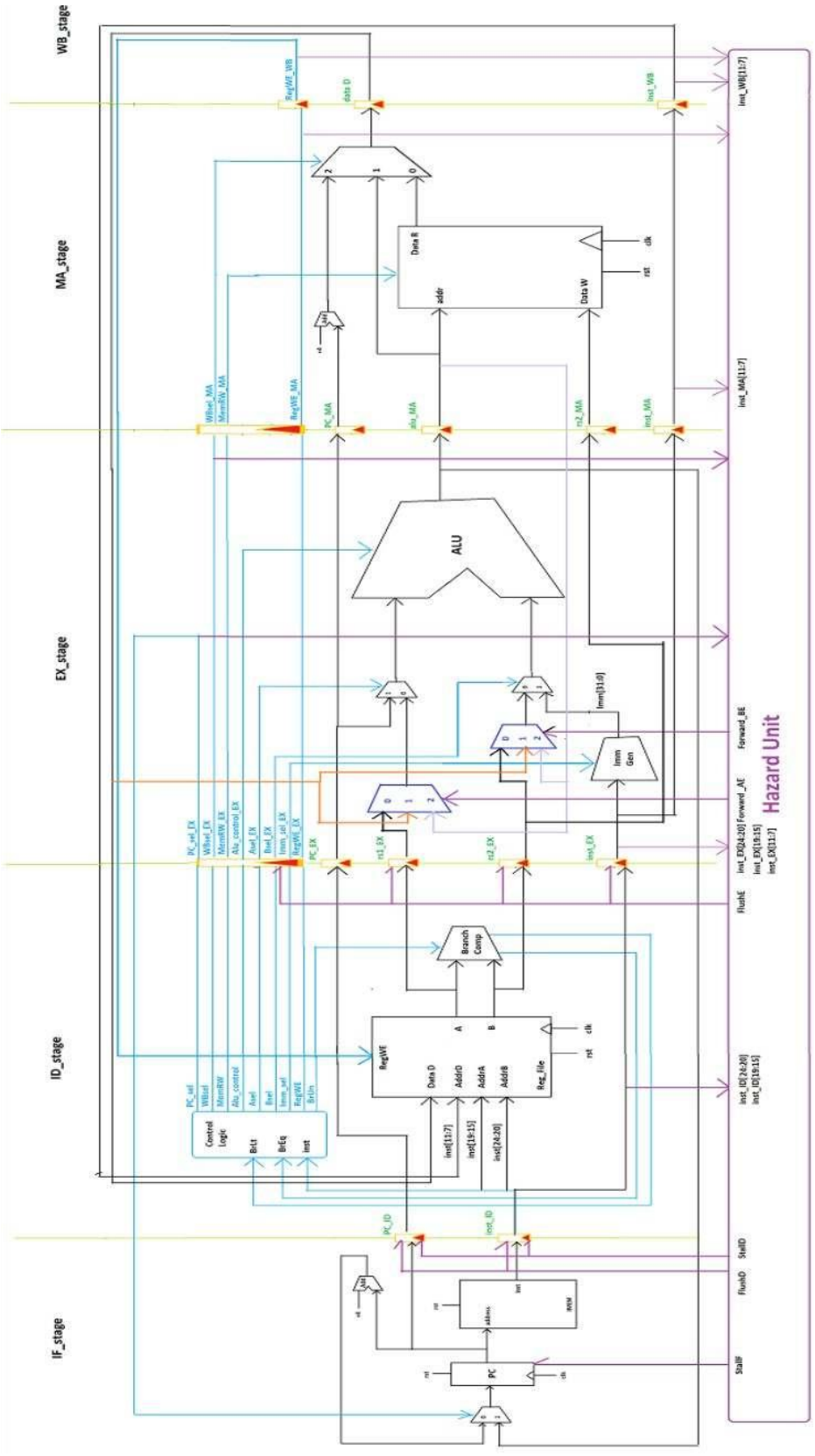
Các giải pháp giải quyết xung đột điều khiển cho lệnh nhảy có điều kiện, ví dụ với beq:

- Cơ bản nhất là chờ cho tới khi điều kiện bằng được tính toán thì lệnh tiếp theo mới được nạp vào. Luôn lãng phí một chu kỳ xung clock để chờ điều kiện bằng xảy ra.

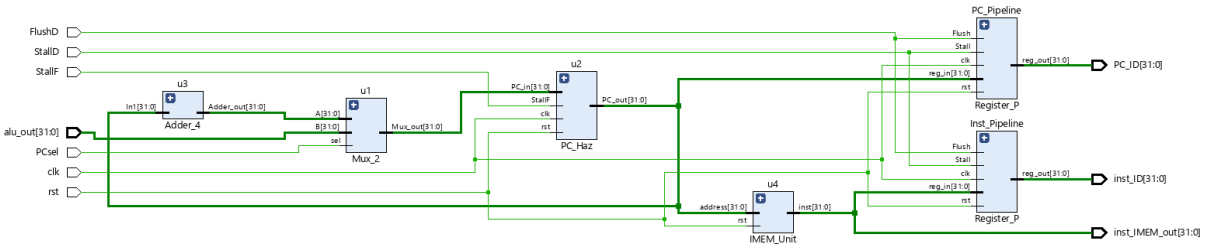
- Cải tiến hơn, có thể dùng phương pháp dự đoán. Có hai cách dự đoán: dự đoán điều kiện bằng sẽ xảy ra (tức nhánh nhảy tới sẽ được lấy); hoặc dự đoán điều kiện bằng sẽ không xảy ra (tức nhánh nhảy tới sẽ không được lấy). Nếu dự đoán đúng, chương trình sẽ không lãng phí chu kỳ xung clock nào; còn nếu dự đoán sai, lệnh đúng sẽ được nạp lại và lãng phí một chu kỳ xung clock.

CHƯƠNG 3: THIẾT KẾ, MÔ PHỎNG

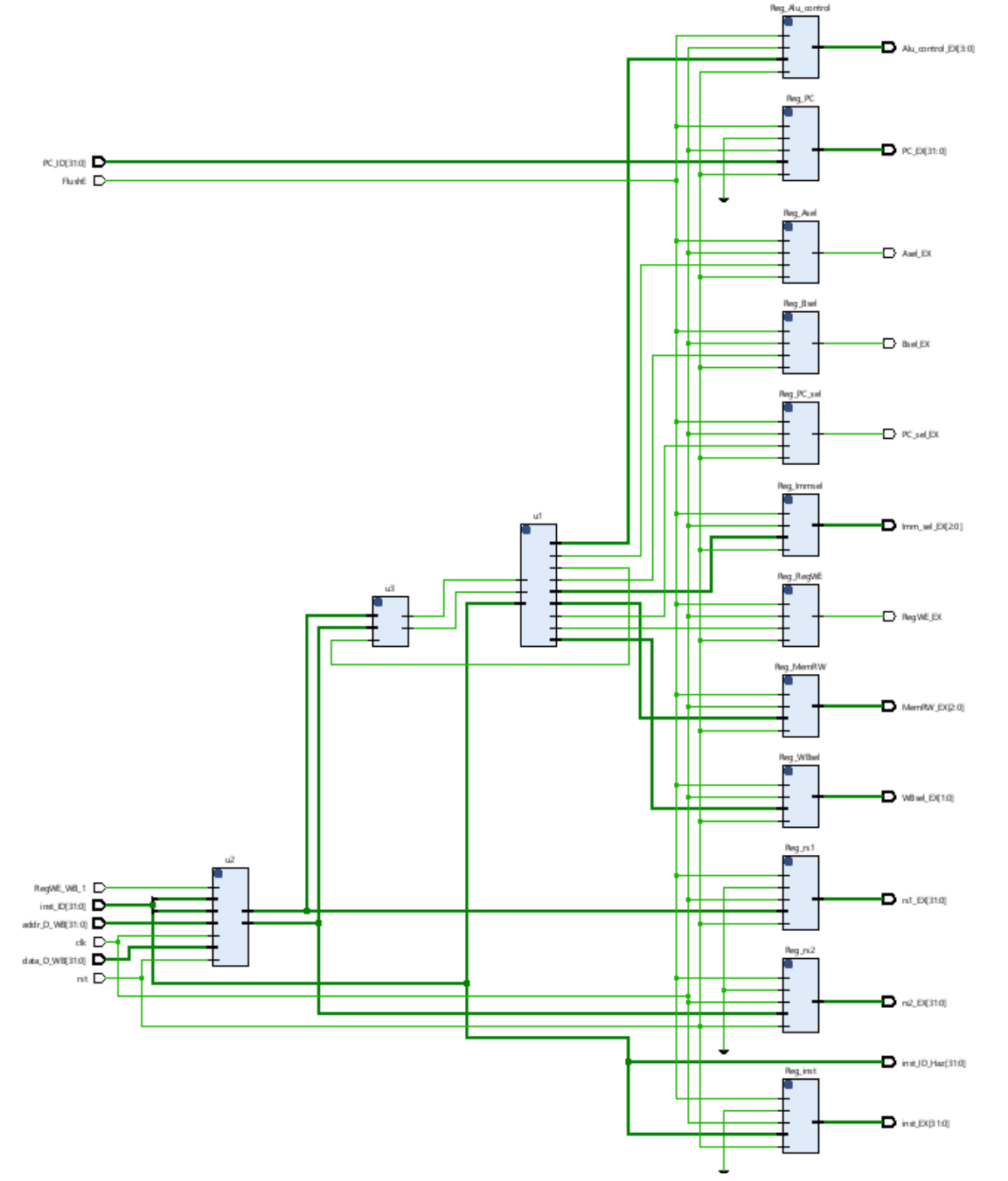
3.1. Thiết kế kiến trúc RISC-V



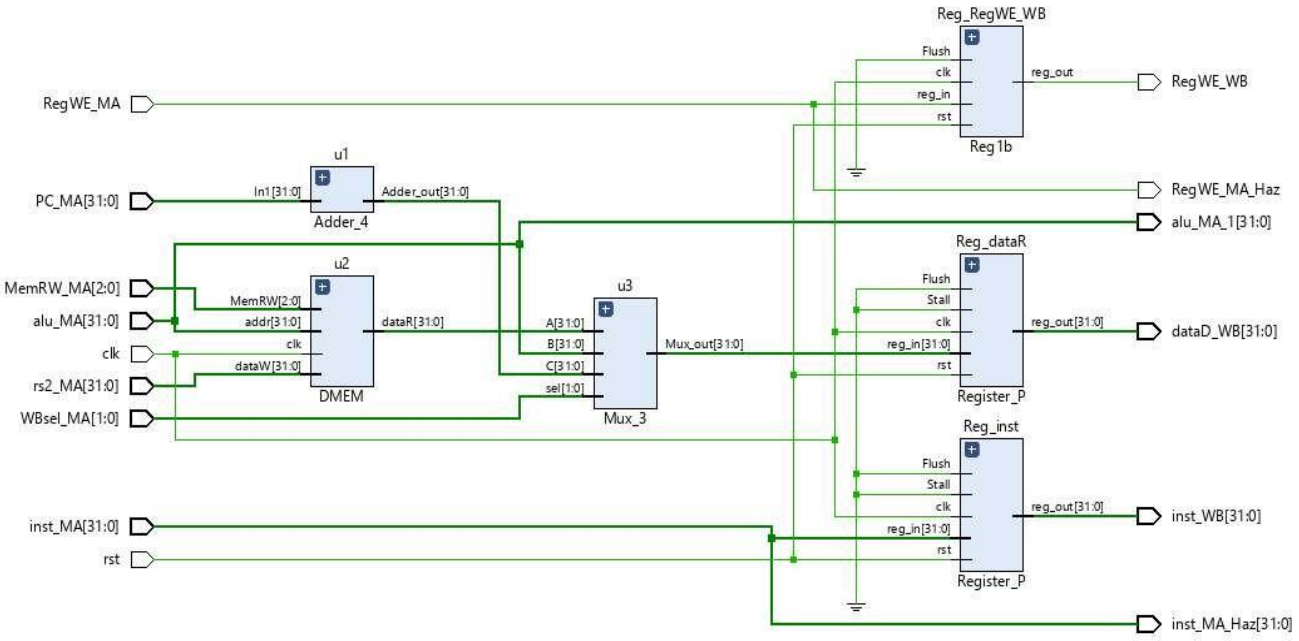
Hình 3. 1. Sơ đồ khối của kiến trúc RISC-V



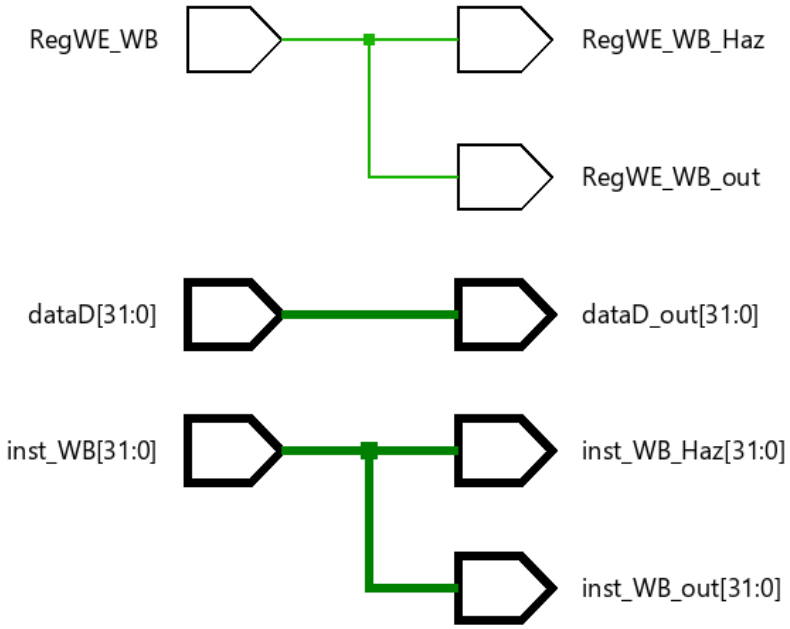
Hình 3. 2. Sơ đồ khối giai đoạn IF

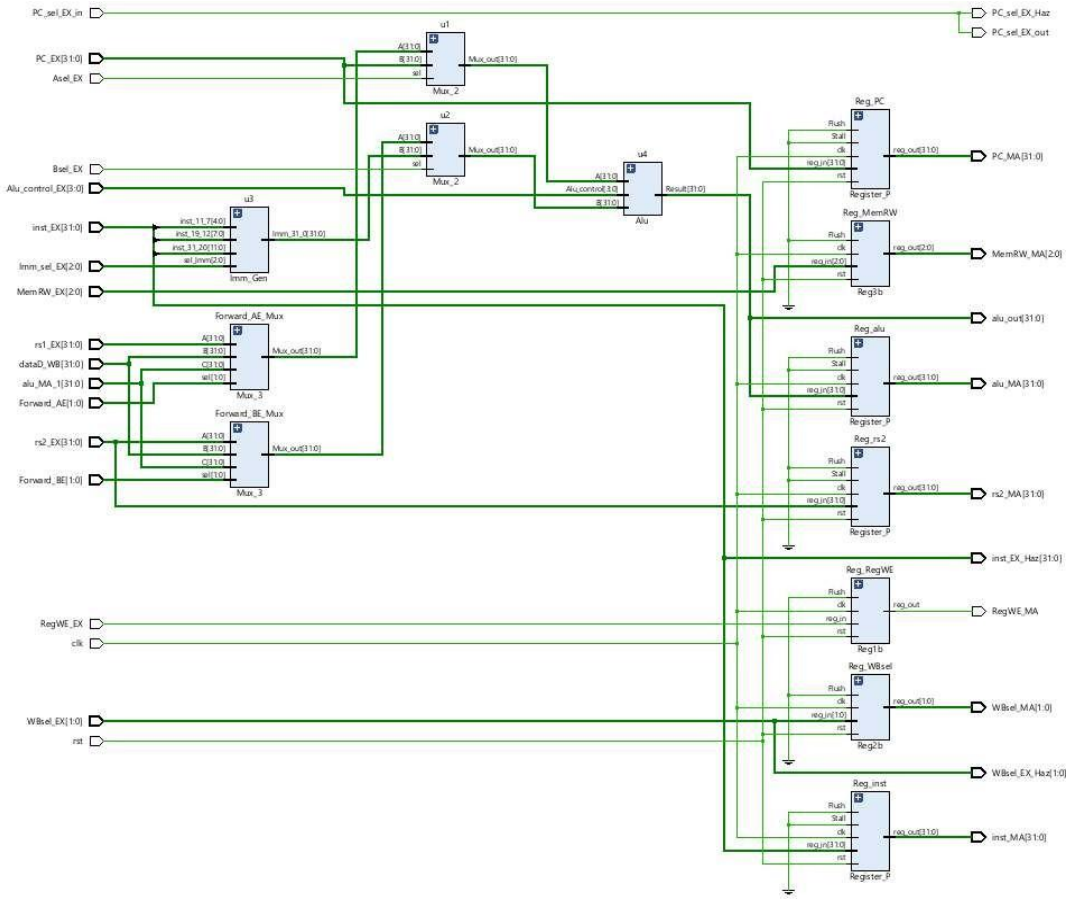


Hình 3. 3. Sơ đồ khối giai đoạn ID

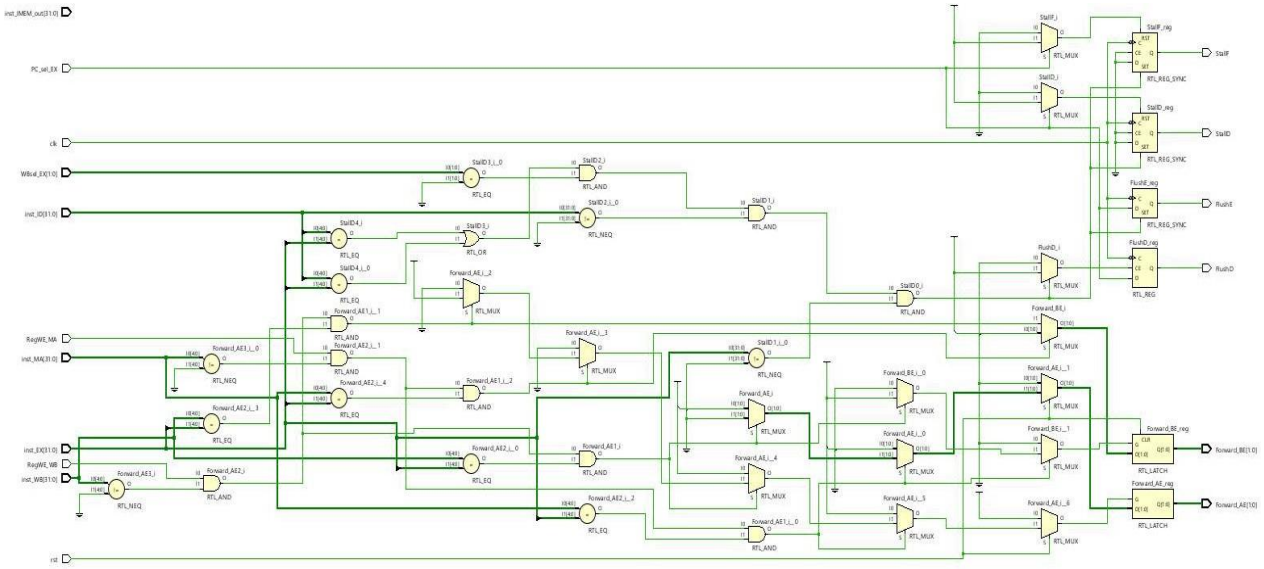


Hình 3. 4. Sơ đồ khối giai đoạn EX





Hình 3. 5. Đường dẫn tín hiệu giai đoạn WB



Hình 3.6. Sơ đồ khối khối Hazard

3.2. Mô phỏng

Giả sử ta nạp vào DMEM giá trị như sau:

Address	Hex	Bin
0	02	0000 0010
1	00	0000 0000
2	00	0000 0000
3	00	0000 0000
4	01	0000 0001
5	00	0000 0000
6	00	0000 0000
7	00	0000 0000
8	03	0000 0011
9	00	0000 0000
10	00	0000 0000
11	00	0000 0000
12	04	0000 0100
13	00	0000 0000
14	00	0000 0000
15	00	0000 0000
16	05	0000 0101
17	00	0000 0000
18	00	0000 0000

19	00	0000 0000
20	06	0000 0110
21	00	0000 0000
22	00	0000 0000
23	00	0000 0000
24	07	0000 0111
25	00	0000 0000
26	00	0000 0000
27	00	0000 0000
28	08	0000 1000
29	00	0000 0000
30	00	0000 0000
31	00	0000 0000
32	09	0000 1001
33	00	0000 0000
34	00	0000 0000
35	00	0000 0000
36	0A	0000 1010
37	00	0000 0000

Bảng 3. 1. Bảng giá trị nạp vào DMEM để mô phỏng Nạp vào

IMEM các giá trị sau:

Address	Hex	Bin	Function
0-3	00 80 23 03	000000001000_00000_010_00110_0000011	w x6, 8(x0) x6 = 03h
4-7	00 70 22 03	000000000111_00000_010_00100_0000011	w x4, 7(x0) X4= 300h
8-11	01 80 24 83	000000011000_00000_010_01001_0000011	v x9, 24(x0) X9 = 07h
12-15	00 A4 A4 03	000000001010_01001_010_01000_0000011	Lw x8, 10(x9) X8 = 6000000h
16-19	00 C3 20 83	000000001100_00110_010_00001_0000011	w x1, 12(x6) X1 = 500h
20-23	01 40 22 83	000000010100_00000_010_00101_0000011	v x5, 20(x0) X5 = 06h
24-27	00 D3 23 83	000000001110_00110_010_00111_0000011	v x7, 13(x6) X7 = 05h
28-31	01 42 A1 03	000000010100_00001_010_00010_0000011	Lw x2, 20(x5) X2 = 80000h
32-35	00 51 72 33	0000000_00101_00010_111_00100_0110011	l x4, x2, x5 X4 =0h
36-39	00 61 64 33	0000000_00110_00010_110_01000_	Or x8, x2, x6

		0110011	X8 = 80003h
40-43	00 22 04 B3	0000000_00010_00100_000_01001_0110011	Add x9, x4, x2 X9 = 80000h
44-47	00 73 20 B3	0000000_00111_00110_010_00001_0110011	Slt x1, x6, x7 X1 = ffffffffh
48-51	00 24 44 63	0_000000_00010_01000_100_0100_0_1100011	x8, x2, Lab1 Không rẽ
52-55	00 41 88 63	0_000000_00100_00011_000_1000_0_1100011	, x4, Lab2 Rẽ
56-59	40 A2 80 B3	0100000_01010_00101_000_00001_0110011	)1: sub x1, x5, x10(None)
60-63	00 44 61 33	0000000_00100_01000_110_00010_0110011	x2, x8, x4 (None)
64-67	00 62 C3 B3	0000000_00110_00101_100_00111_0110011	x7, x5, x6 (None)
68-71	00 03 A4 23	0000000_00000_00111_010_01000_0100011	Lab2: Sw x0, 8(x7) DMem(16:13)=0h
72-75	01 01 83 93	000000010000_00011_000_00111_0010011	di x7, x3, 16 X7 = 10h
76-79	00 83 A7 23	0000000_01000_00111_010_01110_0100011	Sw x8, 14(x7) DMem(33:30)=80003h

Bảng 3. 2. Bảng giá trị nạp vào IMEM để mô phỏng

KẾT LUẬN

Sau khi tìm hiểu và nghiên cứu kiến trúc vi xử lý RISC-V, chúng em có rút ra một số kết luận sau:

1, Ưu điểm

+ Mã nguồn mở: RISC-V giúp giảm chi phí phát triển và mở rộng khả năng tùy chỉnh. Vì là mã nguồn mở, nó không yêu cầu các khoản phí cấp phép và cho phép cộng đồng tham gia phát triển và cải tiến.

+ Hiệu suất cao và tiêu thụ năng lượng thấp: RISC-V được thiết kế để tối ưu hóa hiệu suất và tiết kiệm năng lượng.

2, Nhược điểm

+ Hạn chế về hệ sinh thái: So với các kiến trúc vi xử lý phổ biến khác như ARM hay x86, hệ sinh thái RISC-V vẫn còn hạn chế. Các công cụ phát triển, thư viện và tài liệu không phong phú bằng những kiến trúc đã được phát triển từ lâu.

+ Hiệu năng không đạt tới mức tối ưu: Mặc dù RISC-V có nhiều ưu điểm, hiệu năng của nó vẫn chưa đạt tới mức tối ưu so với một số kiến trúc khác. Điều này có thể ảnh hưởng đến hiệu suất của hệ thống sử dụng RISC-V.

+ Thiếu chuẩn hóa trong việc xử lý hậu quả (hazard): RISC-V chưa có chuẩn hóa rõ ràng về cách xử lý hậu quả (hazard) trong quá trình thực thi lệnh. Điều này có thể gây ra các vấn đề liên quan đến đồng bộ hóa và hiệu suất.

3, Ứng dụng thực tế

+ Điện tử tiêu dùng: RISC-V có ứng dụng trong lĩnh vực điện tử tiêu dùng như điện thoại di động, máy tính bảng, thiết bị gia dụng thông minh và nhiều thiết bị khác. Với hiệu suất cao và tiêu thụ năng lượng thấp, RISC-V giúp tối ưu hoá hoạt động và tăng cường trải nghiệm người dùng trong các thiết bị này.

+ Internet vạn vật (IoT): RISC-V được sử dụng trong các hệ thống IoT như cảm biến, thiết bị theo dõi, và các ứng dụng liên quan đến thu thập và xử lý dữ liệu từ môi trường xung quanh.

+ Công nghiệp ô tô: RISC-V có tiềm năng trong việc phát triển các hệ thống điều khiển và xử lý dữ liệu trong ô tô thông minh và tự động.

+ Công nghiệp ô tô: RISC-V có tiềm năng trong việc phát triển các hệ thống điều khiển và xử lý dữ liệu trong ô tô thông minh và tự động.

+ Công nghiệp ô tô: RISC-V có tiềm năng trong việc phát triển các hệ thống điều khiển và xử lý dữ liệu trong ô tô thông minh và tự động.

4, Phương hướng phát triển

+ Thiết kế thêm bộ nhớ cache tích hợp vào vi xử lý RISC-V nhằm tăng tốc truy cập bộ nhớ cho lệnh và dữ liệu, tăng hiệu suất cho bộ xử lý.

+ Thiết kế thêm bus truyền dữ liệu, các giao thức giao tiếp với cách cổng vào ra để hoàn chỉnh bộ vi xử lý RISC-V 32bit.

+ Tối ưu hóa xử lý song song: Sử dụng các phương pháp tối ưu để tăng mức độ xử lý song song bên trong CPU như superscalar.

TÀI LIỆU THAM KHẢO

[1]. Computer Organization and Design, RISC-V Edition, Patterson & Hennessy, © 2018, MK.

[2]. Great ideas in Computer Architecture (CS 61C) lecture slides, Garcia and Nikolíc, © 2020, UC Berkeley.

[3]. 简记一次计算机系统课程实验-RISCV 流水线处理器设计与实现. Web:简记一次计算机系统课程实验-RISCV 流水线处理器设计与实现 risc-v 处理 器实验设计-CSDN 博客

[4]. Xilinx, *"Programming and Debugging",* năm 2019.

[5]. Xilinx, "*Vivado Design Suite User Guide: Synthesis*", năm 2021.

[6]. Designing a CPU in VERILOG Series Quick Links, Web: Designing a CPU in