|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  Detail - Tổ chức hỗ trợ  **BÀI TẬP LỚN MÔN HỌC**  **KIẾN TRÚC MÁY TÍNH**  **ĐỀ TÀI:**  **Mô phỏng đường dữ liệu bộ xử lý RISCV cho các chương trình viết bằng ngôn ngữ C**   |  |  |  |  | | --- | --- | --- | --- | | **GVHD:** | | Tạ Thị Kim Huệ | | | Họ tên: | Lưu Văn Công | | | |  | | MSSV | 20182390 | | |  |   Hà Nội, 4-1-2021 |

**LỜI NÓI ĐẦU**

Hiện nay, với sự phát triển mạnh mẽ của khoa học và kỹ thuật nói chung và ngành khoa học máy tính nói riêng thì việc ứng dụng máy tính vào đời sống giúp tính toán, giải quyết những bài toán phức tạp và ngày càng trở nên phổ biến.

Việc tìm hiểu về kiến trúc máy tính giúp chúng ta có kiến thức cơ bản để hiểu về nguyên lý và cấu tạo của máy tính, tạo tiền đề để phát triển các loại máy tính ưu việt hơn trong tương lai. Kiến trúc RISC-V được sử dụng trong các hệ thống máy tính đồ họa silicon, các hệ thống nhúng, các thiết bị điện tử.... Chính vì vậy, em thực hiện đề tài **“Mô phỏng đường dữ liệu bộ xử lý RISC-V cho các chương trình viết bằng ngôn ngữ C”** để hiểu rõ hơn kiến trúc RISC-V thực hiện.

Do kiến thức còn hạn hẹp hẹp cộng thêm vấn đề thời gian cho nên đề tài vẫn còn nhiều sai sót, hạn chế. Nội dung đề tài được thực hiện nhằm tìm hiểu các kiến thức, lý thuyết cơ bản vế bộ xử lý RISC V và mô phỏng chương trình C đơn giản. Em rất mong nhận được sự đóng góp của cô Tạ Thị Kim Huệ và các bạn để đề tài hoàn thiện hơn.

**MỤC LỤC**

[DANH MỤC HÌNH VẼ 1](#_Toc76075199)

[CHƯƠNG 1. Cơ sở lý thuyết 4](#_Toc76075200)

[1.1 Giới thiệu RISC-V: 4](#_Toc76075201)

[1.2 Các bước thực hiện: 5](#_Toc76075202)

[1.3 Các giai đoạn của datapath: 6](#_Toc76075203)

[1.3.1 Xây dựng đường dữ liệu (Building a Datapath) 7](#_Toc76075204)

[1.3.2 R-Format Instructions 9](#_Toc76075205)

[1.3.3 Load/Store Instructions 13](#_Toc76075206)

[1.3.4 Branch Instructions 14](#_Toc76075207)

[1.3.5 Kết hợp các thành phần của đường dẫn dữ liệu Datapath. 15](#_Toc76075208)

[1.3.6 Thực hiện chương trình đơn giản hoạt động của đường dữ liệu trong bộ xử lý Risc-v. 17](#_Toc76075209)

[1.3.7 RV32 Datapath 23](#_Toc76075210)

[1.4 Datapath cho các format lệnh: 26](#_Toc76075211)

[1.4.1 Lệnh loại R 26](#_Toc76075212)

[1.4.2 Lệnh loại I 28](#_Toc76075213)

[1.4.3 Lệnh loại S 31](#_Toc76075214)

[1.4.4 Lệnh loại B 32](#_Toc76075215)

[1.4.5 Lệnh loại J 33](#_Toc76075216)

[1.4.6 Lệnh loại U 34](#_Toc76075217)

[CHƯƠNG 2. Thực hành 37](#_Toc76075218)

[2.1 Giới thiệu phần mềm ripes: 37](#_Toc76075219)

[2.2 Mô phỏng: 40](#_Toc76075220)

# DANH MỤC HÌNH VẼ

[Hình 1. 1: Mô tả từ ngôn ngữ bậc cao đến phần cứng 5](#_Toc76075221)

[Hình 1. 2: Các giai đoạn đường dữ liệu 6](#_Toc76075222)

[Hình 1. 3: CPU Components 7](#_Toc76075223)

[Hình 1. 4: Khối Instruction Fetch 8](#_Toc76075224)

[Hình 1. 5: Khối Instruction Memory 8](#_Toc76075225)

[Hình 1. 6: Program counter 9](#_Toc76075226)

[Hình 1. 7: Adder 9](#_Toc76075227)

[Hình 1. 8: Khối Register File 10](#_Toc76075228)

[Hình 1. 9: Register File Read 11](#_Toc76075229)

[Hình 1. 10: Register File Write 11](#_Toc76075230)

[Hình 1. 11: Mô tả tệp thanh ghi 12](#_Toc76075231)

[Hình 1. 12: The Arithmetic Logic Unit 12](#_Toc76075232)

[Hình 1. 13: Data memory unit 13](#_Toc76075233)

[Hình 1. 14: Immediate generation unit 14](#_Toc76075234)

[Hình 1. 15: Sơ đồ đường dữ liệu lệnh branch 15](#_Toc76075235)

[Hình 1. 16: Sơ đồ đường dữ liệu cho lệnh loại R, lệnh ld, sd 16](#_Toc76075236)

[Hình 1. 17: Sơ đồ datapath cho lệnh loại R, lệnh rẽ nhánh có điều kiện, ld,sd 16](#_Toc76075237)

[Hình 1. 18: Đường dữ liệu thể hiện các đoạn bit mã lệnh được gửi từ bộ nhớ lệnh đến tập thanh ghi. 19](#_Toc76075238)

[Hình 1. 19: Toàn bộ đường dữ liệu với khối điều khiển Control Unit 20](#_Toc76075239)

[Hình 1. 20: RISC-V instruction encoding 22](#_Toc76075240)

[Hình 1. 21: Immediate generation unit 23](#_Toc76075241)

[Hình 1. 22: Branch Comparator 23](#_Toc76075242)

[Hình 1. 23: Complete RV32 Datapath 24](#_Toc76075243)

[Hình 1. 24: Cấu trúc lệnh loại R 26](#_Toc76075244)

[Hình 1. 25: Datapath lệnh add 27](#_Toc76075245)

[Hình 1. 26: Datapath lệnh addi 28](#_Toc76075246)

[Hình 1. 27: Datapath lệnh LW 30](#_Toc76075247)

[Hình 1. 28: Datapath lệnh SW 31](#_Toc76075248)

[Hình 1. 29: Datapath lệnh BEQ 33](#_Toc76075249)

[Hình 1. 30: Datapath lệnh jal 34](#_Toc76075250)

[Hình 1. 31: Datapath lệnh Lui 35](#_Toc76075251)

[Hình 1. 32: Datapath lệnh AUIPC 36](#_Toc76075252)

[Hình 2. 1: The Editor Tab 37](#_Toc76075253)

[Hình 2. 2: The Processor Tab 38](#_Toc76075254)

[Hình 2. 3: Datapath 39](#_Toc76075255)

[Hình 2. 4: Select Processor 39](#_Toc76075256)

[Hình 2. 5: Code C 40](#_Toc76075257)

[Hình 2. 6: Label Assembly Code 41](#_Toc76075258)

[Hình 2. 7: Label \_Start in Assembly Code 42](#_Toc76075259)

[Hình 2. 8: Label <main> Assembly Code 43](#_Toc76075260)

[Hình 2. 9: Datapath lệnh add 44](#_Toc76075261)

[Hình 2. 10: Thống kê chương trình 44](#_Toc76075262)

# Cơ sở lý thuyết

## Giới thiệu RISC-V:

RISC-V là một ISA mở, miễn phí cho phép một kỷ nguyên đổi mới bộ xử lý mới thông qua cộng tác tiêu chuẩn mở. Ra đời trong lĩnh vực học thuật và nghiên cứu, RISC-V ISA mang đến một cấp độ mới về tự do phần mềm và phần cứng có thể mở rộng trên kiến trúc, mở đường cho thiết kế và đổi mới máy tính trong 50 năm tới.

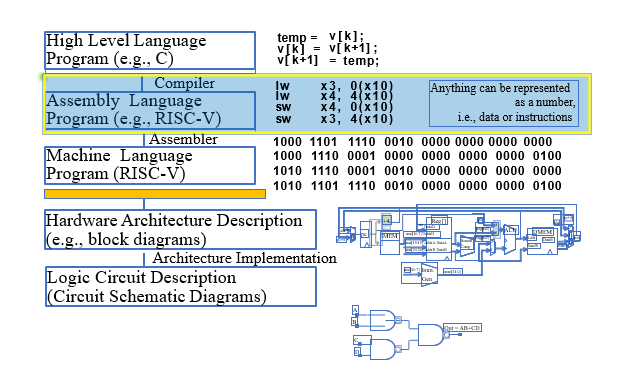
Kiến trúc RISC-V, được công bố công khai vào năm 2014, được phát triển tại Đại học California ở Berkeley bởi Yunsup Lee, Krste Asanović, David A. Patterson và Andrew Waterman. Nỗ lực này tiếp nối bốn dự án thiết kế kiến ​​trúc RISC lớn trước đây tại UC Berkeley, dẫn đến cái tên RISC-V, trong đó *V* đại diện cho số năm La Mã.

Dự án RISC-V bắt đầu như một bản sạch với các mục tiêu chính sau:

* Thiết kế kiến ​​trúc tập lệnh RISC (ISA) phù hợp để sử dụng trong nhiều ứng dụng, trải rộng từ các thiết bị nhúng công suất vi mô đến các bộ đa xử lý máy chủ đám mây hiệu suất cao.
* Cung cấp ISA mà bất kỳ ai cũng có thể sử dụng miễn phí cho bất kỳ ứng dụng nào. Điều này trái ngược với ISA của hầu hết tất cả các bộ vi xử lý có sẵn trên thị trường khác, vốn là tài sản trí tuệ được bảo vệ cẩn thận của công ty thiết kế chúng.
* Kết hợp các bài học kinh nghiệm từ nhiều thập kỷ trước về thiết kế bộ xử lý, tránh những sai lầm và các tính năng không tối ưu mà các kiến ​​trúc khác phải giữ lại trong các thế hệ mới hơn để duy trì khả năng tương thích với các thế hệ trước, đôi khi là cổ xưa.
* Cung cấp ISA cơ sở nhỏ nhưng đầy đủ phù hợp để sử dụng trong các thiết bị nhúng. ISA cơ sở là tập hợp các khả năng tối thiểu mà bất kỳ bộ xử lý RISC-V nào phải thực hiện. RISC-V cơ sở là một kiến ​​trúc bộ xử lý 32-bit với 31 thanh ghi có mục đích chung. Tất cả các hướng dẫn đều dài 32 bit. ISA cơ sở hỗ trợ phép cộng và trừ số nguyên, nhưng không bao gồm phép nhân và phép chia số nguyên. Điều này là để tránh buộc các triển khai bộ xử lý tối thiểu phải bao gồm phần cứng nhân và chia khá đắt tiền cho các ứng dụng không yêu cầu các hoạt động đó.
* Cung cấp các phần mở rộng ISA tùy chọn để hỗ trợ toán học dấu phẩy động, các phép toán trong bộ nhớ nguyên tử cũng như phép nhân và chia.
* Cung cấp các phần mở rộng ISA bổ sung để hỗ trợ các chế độ thực thi đặc quyền, tương tự như các kiến ​​trúc đặc quyền x86, x64 và ARM.
* Hỗ trợ tập lệnh nén, triển khai các phiên bản 16-bit của nhiều lệnh 32-bit. Trong các bộ xử lý triển khai phần mở rộng này, các lệnh 16 bit có thể được xen kẽ tự do với các lệnh 32 bit.
* Cung cấp các phần mở rộng ISA tùy chọn để hỗ trợ kích thước từ bộ xử lý 64 bit và thậm chí 128 bit bằng cách sử dụng bộ nhớ ảo được phân trang trên bộ xử lý đơn và đa lõi cũng như trong các cấu hình đa xử lý.

Bộ xử lý RISC-V hiện có sẵn với giá cả cạnh tranh và, với sự tinh tế của thiết kế ISA và những ưu điểm của tính chất miễn phí sử dụng, chúng ta có thể mong đợi việc sử dụng bộ xử lý RISC-V sẽ phát triển nhanh chóng trong những năm tới. Có sẵn các bản phân phối RISC-V Linux, bao gồm tất cả các công cụ phát triển phần mềm cần thiết để xây dựng và chạy các ứng dụng trên máy tính và thiết bị thông minh dựa trên RISC-V.

## Các bước thực hiện.



Hình 1. 1: Mô tả từ ngôn ngữ bậc cao đến phần cứng

Mô tả 1 chương trình bậc cao chạy trên phần cứng:

Từ hình ảnh ta có thể thấy được từ ngôn ngữ bậc cao chúng ta phải chuyển nó về ngôn ngữ assembly sau đó chuyển các câu lệnh đó thành mã máy cụ thể theo cấu trúc tập lệnh RISC-V từ đó phần cứng thực hiện các phép toán để đưa ra kết quả.

## Các giai đoạn của datapath:

Diagram

Description automatically generated

Hình 1. 2: Các giai đoạn đường dữ liệu

Quá trình thực thi lệnh trong 1chu kỳ của bộ xử lý RISC-V được chia thành 5 giai đoạn:

Giai đoạn IF (Instruction Fetch Stage): Giai đoạn nạp lệnh.

Giai đoạn ID (Instruction Decode Stage): Giai đoạn giải mã lệnh.

Giai đoạn EX (Execute Stage): Giai đoạn thực thi lệnh.

Giai đoạn MEM (Memory Access Stage): Giai đoạn truy cập bộ nhớ.

Giai đoạn WR (Register Write): Ghi dữ liệu vào tệp thanh ghi.

### Xây dựng đường dữ liệu (Building a Datapath)

Diagram, schematic

Description automatically generated

Hình 1. 3: CPU Components

#### Combinational Circuits

* AND -gate

Y = A&B A picture containing diagram

Description automatically generated

* Adder

Y = A + B Diagram

Description automatically generated

* Multiplexer

Y = S ? I1 : I0 A picture containing text, clock

Description automatically generated

* Arithmetic/Logic Unit

Y = F(A,B) Diagram, schematic

Description automatically generated

#### Nạp lệnh (Instruction Fetch)

Khối Instruction Fetch là phần đầu của một Datapath, dung để lưu trữ các lệnh của chương trình dưới dạng mã máy (dạng nhị phân). Mỗi lệnh khi dịch ra mã máy tương ứng với 32 bits, cần 32 bits để lưu trữ. Đầu vào khối Instruction Memory là địa chỉ của lệnh cần lấy ra, đầu ra là mã máy của lệnh tương ứng.

Diagram, schematic

Description automatically generated

Hình 1. 4: Khối Instruction Fetch

Khối bộ nhớ lệnh Instruction Memory chỉ có một cổng đọc , khối này nhận một đầu vào là địa chỉ lệnh 32 bit và đầu ra là 32 bit mã máy của lệnh.

Diagram

Description automatically generated

Hình 1. 5: Khối Instruction Memory

Đầu vào của bộ nhớ lệnh chính là đầu ra của bộ đếm chương trình Program Counter. Bộ đếm chương trình là một thanh ghi 32 bit, hoạt động đồng bộ theo sườn lên của xung Clock, thực hiện đếm địa chỉ trong quá trình nạp lệnh. Đầu ra khối này trỏ tới lệnh hiện tại, đầu vào là địa chỉ của lệnh tiếp theo cần thực thi. Khi có tín hiệu thực hiện lệnh mới, PC sẽ được cộng thêm 4.

Chart, box and whisker chart

Description automatically generated

Hình 1. 6: Program Counter

Bộ cộng gồm 2 đầu vào, một là địa chỉ lệnh ở thời điểm hiện tại (dầu ra của khối PC), hai là hằng số có giá trị bằng 4. Đầu ra là kết quả 2 giá trị đầu vào của bộ cộng lúc này là địa chỉ của lệnh được thực hiện kế tiếp. Bộ đệm thực chất là một khối ALU chỉ thực hiện phép cộng.

Diagram

Description automatically generated

Hình 1. 7: Adder

### R-Format Instructions

Quá trình thực hiện 1 lệnh loại R thông thường cần đọc dữ liệu của 2 toán hạng từ thanh ghi. Thực hiện phép toán với 2 giá trị vừa đọc rồi ghi kết quả vào thanh ghi đích tring khối Register File.

#### Khối Register File

Tệp thanh ghi (Register File) là phần trung tâm trong toàn bộ đường dữ liệu. Nó là một tập hợp các thanh ghi trong đó bất kỳ thanh ghi nào cũng có thể được đọc hoặc ghi. Số thanh ghi được chỉ định. Đối với bộ xử lý RISC-V, ta chỉ định tập thanh ghi gồm 32 thanh ghi, mỗi thanh ghi có chiều dài 32 bit dữ liệu. Ví dụ với lệnh loại R gồm tất cả 3 toán hạng. Mỗi lệnh cần đọc hai từ dữ liệu từ tệp thanh ghi và ghi một từ dữ liệu vào tệp thanh ghi. Đối với mỗi từ dữ liệu được đọc từ các thanh ghi, cần một đầu vào cho tệp thanh ghi chỉ định số thanh ghi sẽ được đọc và một đầu ra từ tệp thanh ghi mang giá trị dữ liệu đã được đọc trước đó. Để ghi một từ dữ liệu cần hai đầu vào: một để chỉ định số thanh ghi sẽ được ghi và một để cung cấp dữ liệu được ghi vào thanh ghi. Tệp thanh ghi luôn xuất ra nội dung của bất kỳ số thanh ghi nào có trên các đầu vào của thanh ghi đọc. Tuy nhiên, quá trình ghi được điều khiển bởi tín hiệu điều khiển ghi, tín hiệu này phải được xác nhận để hành động ghi xảy ra ở cạnh của xung đồng hồ (Hay nói cách khác quá trình đọc luôn luôn diễn ra còn quá trình ghi chỉ xảy ra khi tín hiệu cho phép ghi được bật ở cạnh lên của xung đồng hồ). Tóm lại tập thanh ghi của bộ xử lý RISC-V cần 4 dầu vào (3 đầu vào chỉ định số thanh ghi 5 bit (Vì 32 = 25) và một đầu vào mang dữ liệu 32 bit), 2 đầu ra mang dữ liệu 32 bit.

Diagram, schematic

Description automatically generated

Hình 1. 8: Khối Register File

Tệp thanh ghi được chia thành 2 phần ứng với 2 chức năng đọc và ghi.

Tệp thanh ghi đọc (Register File Read)

Diagram, schematic

Description automatically generated

Hình 1. 9: Register File Read

* Input:

Gồm 2 tín hiệu Read register number 1 và Read register number 2 đều có chiều dài 5 bit, có tác dụng lựa chọn thanh ghi cần đọc.

* Output:

Gồm 2 tín hiệu Read data 1và Read data 2 (32 bit) mang dữ liệu đọc

Tệp thanh ghi ghi (Register File Write)

Diagram

Description automatically generated

Hình 1. 10: Register File Write

* Input:

Write: 1 bit. (cho phép đọc dũ liệu).

Register number: 5 bit. (lựa chọn thanh ghi)

Write data: 32 bit. (chứa dữ liệu ghi vào tệp thanh ghi)

* Ở đây 32 thanh ghi được quy định như sau:

Table

Description automatically generated

Hình 1. 11: Mô tả tệp thanh ghi

#### The arithmetic logic unit (ALU)

Diagram, schematic

Description automatically generated

Hình 1. 12: The Arithmetic Logic Unit

Khối ALU được minh họa như trên có 2 đầu vào là các toán hạng A và B 32 bit, một đầu ra Result 32 bit, 1 tín hiệu điều khiển OpSelect lựa chọn phép toán cần thực. Tín hiệu OpSelect (hay ALUControl) 4 bit được quy định.

### Load/Store Instructions

Thông thường một lệnh load hoặc store có thể trải qua các công đoạn:

* Lấy dữ liệu được đọc ra từ tệp thanh ghi.
* Tính toán giá trị địa chỉ cần để lưu trữ giá trị đang có ( Sử dụng khối ALU, sign-extend offset).
* Đối với lệnh load: Đọc dữ liệu từ bộ nhớ và ghi giá trị đó vào địa chỉ thanh ghi mong muốn.
* Đối với lệnh store: Đọc dữ liệu từ thanh ghi hiện tại, ghi giá trị đó vào địa chỉ bộ nhớ mong muốn.

#### Khối Data Memory

Khối Data Memory được tác động chủ yếu bởi các lệnh thao tác với bộ nhớ (load, store).

Diagram

Description automatically generated

Hình 1. 13: Data memory unit

Khối Data Memory là một phần tử trạng thái có đầu vào là địa chỉ (Address) và dữ liệu ghi (Write data), một đầu ra duy nhất mang dữ liệu được đọc ra khỏi bộ nhớ (Read data). Các tín hiệu MemRead và MemWrite điều khiển việc đọc và ghi của bộ nhớ một cách riêng biệt. Khác với tệp thanh ghi, khối Data memory cần 1 tín hiệu cho phép đọc vì nếu đọc giá trị của một địa chỉ không hợp lệ có thể gây ra nhiều tác hại.

#### Khối Immediate generation

Khối immediate generation (ImmGen) có đầu vào là mã lệnh 32 bit trong đó 12 bit được sử dụng cho các lệnh load, store, và điều kiện rẽ nhánh để mở rộng kích thước dữ liệu bằng cách tái tạo các bit có dấu bậc cao ở dữ liệu gốc (mã lệnh) thành các bit trong mã immediate 32 bit (đầu ra).

Diagram, venn diagram

Description automatically generated

Hình 1. 14: Immediate generation unit

### Branch Instructions

Lệnh điều khiển rẽ nhánh có 3 toán hạng, hai thanh ghi chứa dữ liệu được so sánh với nhau, và 12 bit offset xác định giá trị địa chỉ của bộ đếm PC nhảy tới (PC + immediate) sau khi thực hiện xong phép so sánh nếu điều kiện thỏa mãn. Nếu điều kiện không thỏa mãn, bộ đếm PC + 4 để thực hiện lệnh kế tiếp.

Diagram

Description automatically generated

Hình 1. 15: Sơ đồ đường dữ liệu lệnh branch

### Kết hợp các thành phần của đường dẫn dữ liệu Datapath.

Một lát cắt của đường dẫn dữ liệu thực hiện một lệnh trong một chu kỳ đồng hồ.

* Mỗi phần tử của đường dữ liệu chỉ có thể thực hiện một chức năng tại một thời điểm.
* Do đó, cần phân tách các lệnh và bộ nhớ dữ liệu riêng biệt

Sử dụng bộ ghép kênh trong đó các nguồn dữ liệu thay thế được sử dụng cho các hướng dẫn khác nhau.

Vì đường dẫn dữ liệu của các lệnh tính toán loại R và các lệnh thao tác với bộ nhớ (load/store) có những điểm chung, ta có kết hợp sau.

Diagram, schematic

Description automatically generated

Hình 1. 16: Sơ đồ đường dữ liệu cho lệnh loại R, lệnh ld, sd

Kết hợp cả quá trình cung cấp địa chỉ lệnh và nạp lệnh, ta có sơ đồ đường dữ liệu khá hoàn chỉnh như sau.

Diagram

Description automatically generated

Hình 1. 17: Sơ đồ datapath cho lệnh loại R, lệnh rẽ nhánh có điều kiện, ld,sd

### Thực hiện chương trình đơn giản hoạt động của đường dữ liệu trong bộ xử lý Risc-v.

#### Khối ALU Control.

Khối ALU Control có chức năng quyết định khối ALU thực hiện hoạt động gì dựa trên các trường funct7, funct3 và trường điều khiển 2 bit ALUOp.

ALUOp = 00 với các lệnh ld, sd.

ALUOp = 01 với lệnh beq.

ALUOp = 10 với các lệnh R-type.

Đầu ra của khối ALU Control là 1 tín hiệu 4 bit lựa chọn hoạt động của khối ALU.

Hoạt động của các lệnh trong bộ xử lý Risc-v chủ yếu dựa trên các phép đơn giản AND, OR, add, subtract. Ta có phân bố đầu ra của khối ALU Control như sau.

Table

Description automatically generated

Ta có bảng:

Table

Description automatically generated

Bảng truth table cho khối ALU Control 4 bit.

A picture containing table

Description automatically generated

#### Khối Control Unit.

Bảng cấu trúc mã lệnh của 4 loại lệnh như dưới đây.

Table

Description automatically generated

Trường opcode là các bit 6:0.

Thanh ghi nguồn rs1 là các bit ở vị trí 19:15 đối với lệnh loại R.

Thanh ghi nguồn rs2 là các bit ở vị trí 24:20 đối với lệnh loại R.

Mã lệnh cóa thể xuất hiện 12 bit offset.

Thanh ghi đích rd ở vị trí 11:7 đối với lệnh loại R.

Diagram

Description automatically generated

Hình 1. 18: Đường dữ liệu thể hiện các đoạn bit mã lệnh được gửi từ bộ nhớ lệnh đến tập thanh ghi.

Khối Control Unit

Khối điều khiển Control Unit tính toán các tín hiệu điều khiển dựa trên trường opcode[6:0] tương ứng các tín hiệu điều khiển Branch, MemRead, MemtoReg, ALOUp, MemWrite, ALUSrc, RegWrite.

Tín hiệu ALOUP 2 bit điều khiển khối ALU Control. Các tín hiệu còn lại đều có giá trị 1 bit.

|  |  |  |
| --- | --- | --- |
| Tín hiệu | Tắt (bit 0) | Bật (bit 1) |
| RegWrite | None | Dữ liệu được phép ghi ở đầu vào Writedata của tệp thanh ghi |
| ALUSrc | Toán hạng thứ 2 của khối ALU là đầu ra của tệp thanh ghi (Read data 2) | Toán hạng thứ 2 của khối ALU là dấu mở rộng, 12 bit của mã lệnh |
| PCSrc | Giá trị của bộ đếm PC được cập nhật bởi đầu ra kết của của bộ cộng PC + 4 | Giá trị của bộ đếm PC được cập nhật bởi đầu ra kết của của bộ cộng tính giá trị địa chỉ PC đạt tới khi điều kiện rẽ nhánh thỏa mãn. |
| MemRead | None | Tín hiệu cho phép đầu ra của bộ nhớ được đọc dữ liệu |
| MemWrite | None | Tín hiệu cho phép đầu vào của bộ nhớ được ghi dữ liệu |
| RegWrite | Giá trị đầu ra của ALU được ghi trở lại vào tệp thanh ghi | Giá trị đầu ra của Bộ nhớ được ghi trở lại vào tệp thanh ghi |

Diagram

Description automatically generated

Hình 1. 19: Toàn bộ đường dữ liệu với khối điều khiển Control Unit

**Lưu ý: Hình 1.19 và 1.18 đường datapath truyền dữ liệu kiểu doubleword. Đối với kiểu dữ liệu là word đường đi tương tự, khác là thay vì truyền 64 bit, ta truyền dữ liệu 32 bit.**

Ta có bảng kết quả tín hiệu điều khiển ứng vơi trường opcode 1 số lệnh thường gặp.

Table

Description automatically generated

The RISC-V encoding of the opcodes for the RISC-V machine language.

Table

Description automatically generated

Hình 1. 20: RISC-V instruction encoding

Trường opcode, trường Funct6/7 và Funct3 với giá trị cụ thể quy đinh các loại lênh khác nhau và từng lệnh một.

### RV32 Datapath

#### Immediate generation unit

Diagram

Description automatically generated

12 bit của mã lệnh (inst[30:20]) sao chép đến 12 bit thấp của mã immediate (imm[31:12])

Lấp đầy 20 bit còn lại (imm[31:12]) của mã immediate bằng giá trị của bit thứ 32 (inst[31]) của mã lệnh.

Diagram

Description automatically generated

Hình 1. 21: Immediate generation unit

#### Branch Comparator

Diagram, schematic

Description automatically generated

Hình 1. 22: Branch Comparator

* BrEq = 1, if A=B.
* BrLt = 1, if A<B.
* BrUn = 1, nếu kết quả âm, = 0 nếu dương.

BGE branch: A>=B, if Icon

Description automatically generated with low confidence

Icon

Description automatically generated

#### Complete RV32 Datapath

Diagram, schematic

Description automatically generated

Hình 1. 23: Complete RV32 Datapath

Quá trình thực thi lệnh trong 1chu kỳ của bộ xử lý RISC-V được chia thành 5 giai đoạn được mô tả lại như sau.

1, Giai đoạn IF (Instruction Fetch Stage): Giai đoạn nạp lệnh. Quá trình thực hiện các lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình (PC) cung cấp địa chỉ lệnh được thực thi. Đầu vào khối PC là một bộ Mux2-1 thực hiện lựa chọn hoạt động của bộ đếm chương trình trong chu kỳ lệnh kế tiếp với tín hiệu lựa chọn PCSel. Khi PCSel được bật, tín hiệu vào khối PC là giá trị địa chỉ được tính toán trả về từ khối ALU sau một quâ trình thực hiện lệnh. Nếu PCSel = 0, đầu vào PC là địa chỉ lệnh kế tiếp (PC + 4) (kết quả của bộ cộng Adder). Khối bộ nhớ lệnh IMEM nhận đầu vào là địa chỉ lệnh 32 bit từ PC và đầu ra là 32 bit mã máy của lệnh.

2, Giai đoạn ID (Instruction Decode Stage): Giai đoạn giải mã lệnh. Khối Register File nhận mã lệnh Inst[31:0] từ bộ nhớ lệnh IMEM lần lượt qua các tín hiệu AddrA, AddrB, AddrD đều có độ dài 5 bit dung lựa chọn thanh ghi cần đọc và ghi.

* AddrA thường là tín hiệu của thanh ghi nguồn 1 nhận mã lệnh Inst[19:15].
* AddrB là tín hiệu của thanh ghi nguồn 2 nhận mã lệnh Inst[24:20].
* AddrD là tín hiệu của thanh ghi đích nhận mã lệnh Inst[11:7].

12 bit đầu ra còn lại của bộ nhớ lệnh Inst[31:20] được truyền qua khối Immediate generation. Immediate generation unit thực hiện biến đổi đầu vào thanh đầu ra imm[31:0] được sử dụng như là một hằng số trong các phép toán addi, địa chỉ offset của các lệnh load, store, branch, … ở các giai đoạn đường dữ liệu kế tiếp. Khối Immediate generation được điều khiển bởi tín hiệu ImmSel, khối hoạt động khi ImmSel được bật.

Đầu vào còn lại của Register File là tín hiệu 32 bit DataD chứa dữ liệu được ghi vào tệp thanh ghi. Quá trình ghi chỉ được thực hiện khi tín hiệu RegWEn = 1. Đầu ra của tệp thanh ghi gồm 2 tín hiệu DataA, DataB chứa dữ liệu 32 bit được đọc.

3, Giai đoạn EX (Execute Stage): Giai đoạn thực thi lệnh. Hai tín hiệu chứa dữ liệu vừa đọc từ tệp thanh ghi được truyền như sau.

Trường hợp 1, qua khối Branch Comp, 2 tín hiệu đầu vào được tiến hành qua các lệnh điều kiện. Các lệnh thực thị phụ thuộc vào 3 tín hiệu điều khiển BrUn, BrEq, BrLT như đã trình bày ở trên.

Trường hợp 2 qua 2 bộ Mux2\_1.

* Bộ Mux2\_1A với tín hiệu điều khiển Asel lựa chọ giữa DataA và giá trị địa chỉ của PC hiện tại. Đầu ra chính là đầu vào hay toán hạng thứ nhất của khối ALU.
* Bộ Mux2\_1B với tín hiệu điều khiển Bsel lựa chọ giữa DataB và giá trị đầu ra Imm[31:0] của khối Immediate generation. Đầu ra là đầu vào hay toán hạng thứ hai của khối ALU.

Khối ALU thực hiện các phép toán với 2 toán hạng đầu vào. Tín hiệu điều khiển ALUSel điều khiển hoạt động của khối ALU. Đầu ra của khối ALU có đường đi theo 3 hướng.

* Hướng thứ nhất tín hiệu đầu ra của khối ALU được đọc vào bộ nhớ DMEM.
* Hướng thứ 2, tín hiệu đầu ra của khối ALU là đầu vào của bộ Mux3\_1 để gửi dữ liệu quay về ghi vào tệp thanh ghi.
* Hướng thứ 3, tín hiệu đầu ra của khối ALU là đầu vào của bộ Mux2\_1 quyết đinh giá trj địa chỉ lệnh trong chu kỳ kế tiếp của khối PC.

4, Giai đoạn MEM (Memory Access Stage): Giai đoạn truy cập bộ nhớ. Khối DMEM gốm 2 tín hiệu đầu vào là địa chỉ được tính toán từ khối ALU và dữ liệu đầu ra DataB của tệp thanh ghi. Đầu ra DataB mang dữ liệu được đọc ra từ bộ nhớ, Quá trình đọc và ghi được quyết đinh bởi tín hiệu MemRW.

5, Giai đoạn WR (Register Write), bộ Mux3\_1 tiến hành nhận 3 tín hiệu đầu vào. Đầu ra phụ thuộc vào tín hiệu điều khiển WBSel được đưa quay trở lại tệp thanh ghi để ghi dữ liệu qua tín hiệu DataD. Ba đầu vào gồm dữ liệu được đọc ra từ bộ nhớ, kết quả đầu ra của khối ALU, kết quả đầu ra của bộ cộng Adder tính giá trị địa chỉ lệnh kê tiếp của bô nhớ lệnh PC.

## Datapath cho các format lệnh:

### Lệnh loại R

Cấu trúc các lệnh loại R:

Ảnh có chứa bàn

Mô tả được tạo tự động

Hình 1. 24: Cấu trúc lệnh loại R

Triển khai lệnh (ví dụ cho lệnh add):

add rd, rs1, rs2

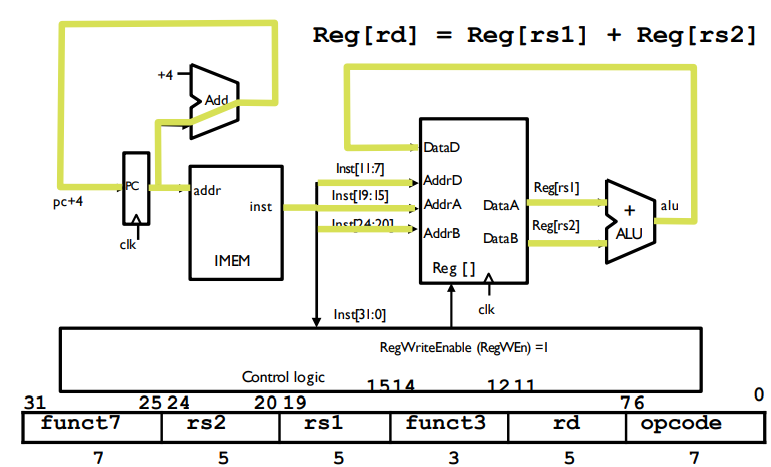
Ảnh có chứa bàn

Mô tả được tạo tự động

Lệnh add thực hiện 2 bước:

* Reg[rd] = Reg[rs1] + Reg[rs2]
* PC = PC + 4

Datapath cho lệnh add:



Hình 1. 25: Datapath lệnh add

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo.
* Sau khi lệnh được tìm nạp, 2 toán hạng thanh ghi rs1 và rs2, thanh ghi đích rd được giải mã bởi các trường của lệnh đó với địa chỉ lần lượt: Addr A, Addr B, Addr D ứng với các thanh ghi data A, data B, data D trong tệp thanh ghi.
* Sau khi 2 toán hạng thanh ghi được tìm nạp sẽ đi vào khối ALU để thực hiện phép cộng logic số học.
* Kết quả đầu ra của ALU là giá trị thanh ghi rd sẽ được ghi vào thanh ghi Data D ứng với địa chỉ Addr D. Tín hiệu điều khiển RegWrite =1 cho phép thực hiện việc ghi dữ liệu này.
* Sau khi thực thi xong PC sẽ cập nhật lên 4 để nhảy đến câu lệnh tiếp theo.

### Lệnh loại I

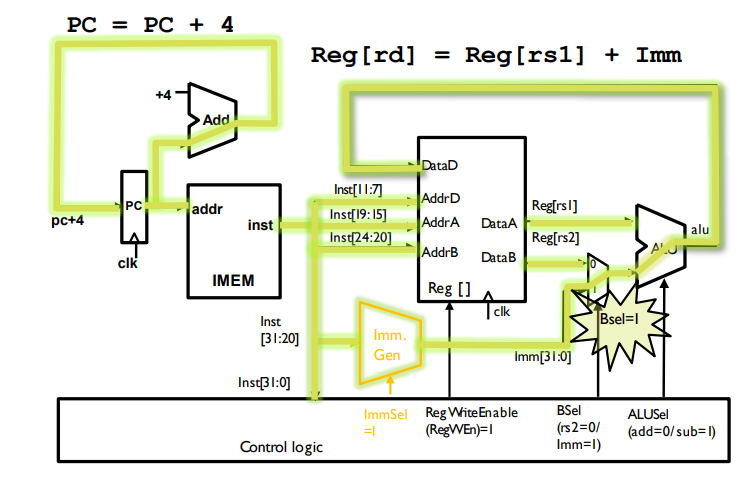
* Ví dụ: Lệnh tính toán số học/logic với giá trị trực tiếp:

addi x15, x1, -50

Ảnh có chứa bàn

Mô tả được tạo tự động

*Datapath cho lệnh addi:*



Hình 1. 26: Datapath lệnh addi

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo.
* Sau khi lệnh được tìm nạp, 2 thanh ghi x1 thanh ghi đích x15 được giải mã bởi các trường của lệnh đó với địa chỉ lần lượt: Addr A, Addr D ứng với các thanh ghi data A, data D trong tệp thanh ghi.
* Gía trị imm -50 đưa vào khối Imm Gen (ImmSel =1)
* ALU thực hiện phép cộng với đầu vào là giá trị thanh ghi Data A (x1) và giá trị imm -50 trong khối Imm Gen.
* Các tín hiệu điều khiển: Bsel=1( rs2=0/Imm=1) vì đây là cộng giá trị trực tiếp, giá trị này nằm trong khối Imm Gen, rs2 trống.

ALUSel=1 (cộng với giá trị âm)

* Kết quả đầu ra ghi vào thanh ghi data D. Tín hiệu điều khiển RegWrite =1 cho phép thực hiện việc ghi dữ liệu này.
* Sau khi thực thi xong PC sẽ cập nhật lên 4 để nhảy đến câu lệnh tiếp theo.
* Ví dụ lệnh dịch chuyển dữ liệu:

lw x14, 8(x2)

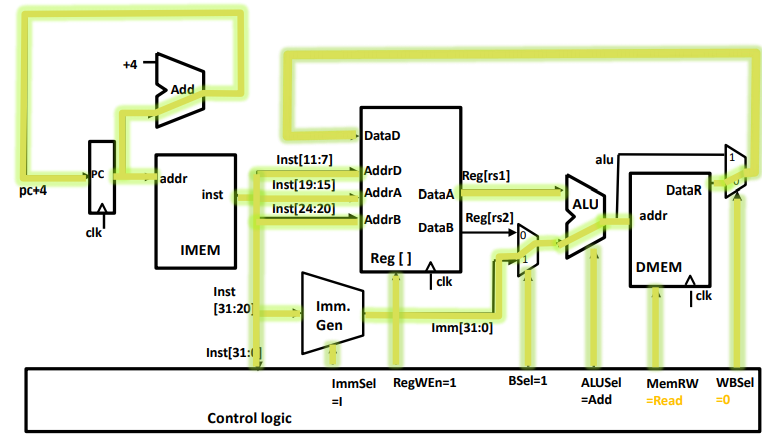
Ảnh có chứa bàn

Mô tả được tạo tự động

Nạp từ nhớ có địa chỉ (x2+8) vào thanh ghi x14:

* Opcode = 0000011 xác định thao tác load từ nhớ
* Rd (01110) xác định thanh ghi đích x14
* Rs1 = 00010 xác định thanh ghi cơ sở x2
* Imm=+8 là độ dời (offset) để truy cập các phần tử mảng có địa chỉ x2+8

*Datapath cho lệnh lw:*



Hình 1. 27: Datapath lệnh LW

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo.
* Sau khi lệnh được tìm nạp, thanh ghi cơ sở x2 và thanh ghi đích x14 được giải mã bởi các trường của lệnh đó với địa chỉ lần lượt: Addr A, Addr D ứng với các thanh ghi data A, data D trong tệp thanh ghi.
* Gía trị imm +8 đưa vào khối Imm Gen (ImmSel =1)
* ALU thực hiện tính địa chỉ của Data Memmory với đầu vào là giá trị thanh ghi cơ sở Data A (x2) và giá trị imm+8 trong khối Imm Gen.
* Các tín hiệu điều khiển: Bsel=1( rs2=0/Imm=1).

ALUSel= Add thực hiện cộng địa chỉ x2+8

* Kết quả đầu ra ALU đi vào Dmem có địa chỉ là addr ứng với thanh ghi DataR, MemRw=Read cho phép đọc dữ liệu từ bộ nhớ lưu vào thanh ghi DataR
* Gía trị thanh ghi DataR được ghi vào thanh ghi data D. Tín hiệu điều khiển RegWrite =1 cho phép thực hiện việc ghi dữ liệu này.
* Sau khi thực thi xong PC sẽ cập nhật lên 4 để nhảy đến câu lệnh tiếp theo.

### Lệnh loại S

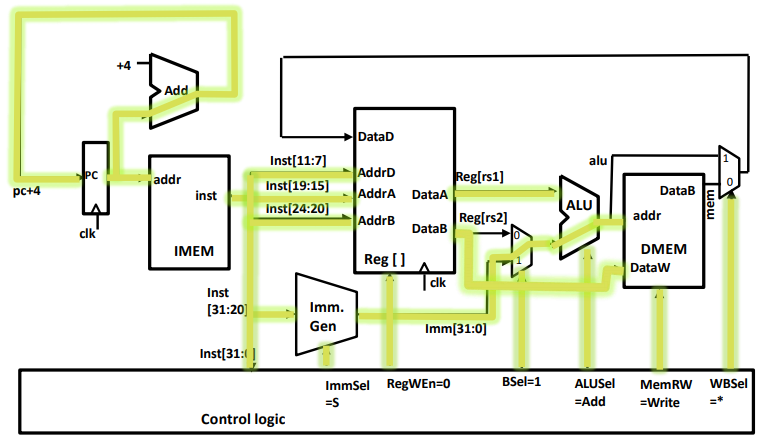
Ví dụ:

sw x14, 8(x2)

Ảnh có chứa bàn

Mô tả được tạo tự động

*Datapath cho lệnh sw:*



Hình 1. 28: Datapath lệnh SW

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo.
* Sau khi lệnh được tìm nạp, thanh ghi cơ sở x2 và thanh ghi toán hạng x14 được giải mã bởi các trường của lệnh đó với địa chỉ lần lượt là Addr A, addr B ứng với thanh ghi data A, data B trong tệp thanh ghi.
* Gía trị imm +8 đưa vào khối Imm Gen (ImmSel =S)
* ALU thực hiện tính địa chỉ của Data Memmory với đầu vào là giá trị thanh ghi cơ sở Data A (x2) và giá trị imm+8 trong khối Imm Gen.
* Các tín hiệu điều khiển: Bsel=1( rs2=0/Imm=1).

ALUSel= Add thực hiện cộng địa chỉ x2+8

* Kết quả đầu ra ALU đi vào Dmem có địa chỉ là addr ứng với thanh ghi DataR, MemRw=Write cho phép ghi dữ liệu từ thanh ghi x14 (Data B) vào bộ nhớ.
* Sau khi thực thi xong PC sẽ cập nhật lên 4 để nhảy đến câu lệnh tiếp theo.

### Lệnh loại B

Có 6 lệnh rẽ nhánh: beq, bne, blt, bge, bltu, bgeu

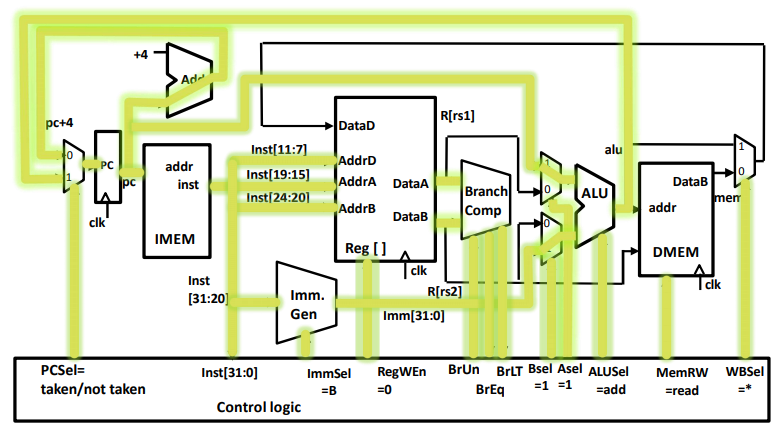
Ảnh có chứa văn bản, bàn

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

*Datapath cho lệnh beq:*

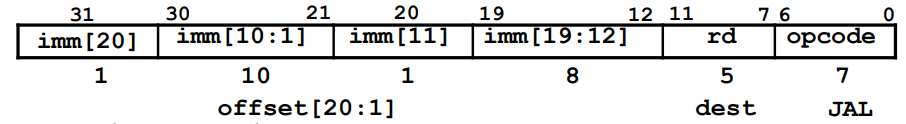


Hình 1. 29: Datapath lệnh BEQ

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo nếu không rẽ nhánh, và sẽ cộng thêm phần bù imm của nhãn so với lệnh hiện tại nếu có rẽ nhánh.
* Sau khi lệnh được tìm nạp, 2 thanh ghi toán hạng rs1, rs2 được giải mã bởi các trường của lệnh đó với địa chỉ lần lượt là Addr A, addr B ứng với thanh ghi data A, data B trong tệp thanh ghi.
* Gía trị imm đưa vào khối Imm Gen (ImmSel =B)
* Bộ so sánh Branch Comp thực hiện so sánh giá trị lưu trữ trong 2 thanh ghi A, B (MemRW=read cho phép đọc dữ liệu từ bộ nhớ phục vụ cho việc so sánh)
* ALU thực hiện phép cộng địa chỉ với đầu vào là PC hiện và giá trị immtrong khối Imm Gen xác định địa chỉ lệnh tiếp theo quyết định lệnh có rẽ nhánh hay không

### Lệnh loại J

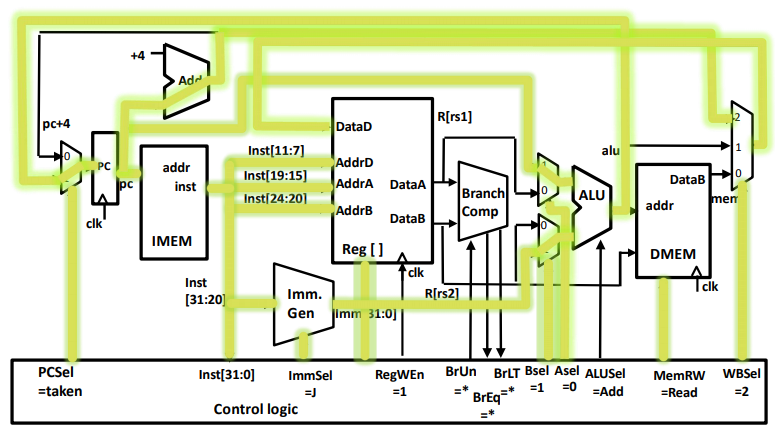
Ví dụ: jal



Lệnh jal thực hiện gồm 2 bước:

* Lưu PC+4 vào thanh ghi rd (địa chỉ trả về)
* Đặt PC = PC + offset (PC-relative jump)

*Datapath lệnh jal:*



Hình 1. 30: Datapath lệnh jal

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, lưu địa chỉ trả về là PC+4 và sẽ cộng thêm phần bù imm của nhãn so với lệnh hiện tại để nhảy đến nhãn. ( RegWEn=1 cho phép ghi PC+4 vào thanh ghi rd)
* Sau khi lệnh được tìm nạp, giá trị imm đưa vào khối Imm Gen (ImmSel =J)
* ALU thực hiện phép cộng địa chỉ với đầu vào là PC hiện và giá trị imm trong khối Imm Gen xác định địa chỉ lệnh nhãn cần nhảy đến. (Bsel=1 đầu vào là imm, ALUSel=add thực hiện cộng địa chỉ)

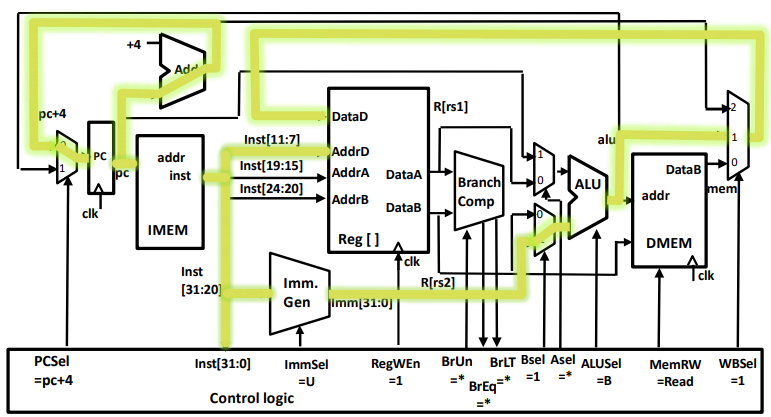
### Lệnh loại U

Ví dụ: lui, auipc

Ảnh có chứa bàn

Mô tả được tạo tự động

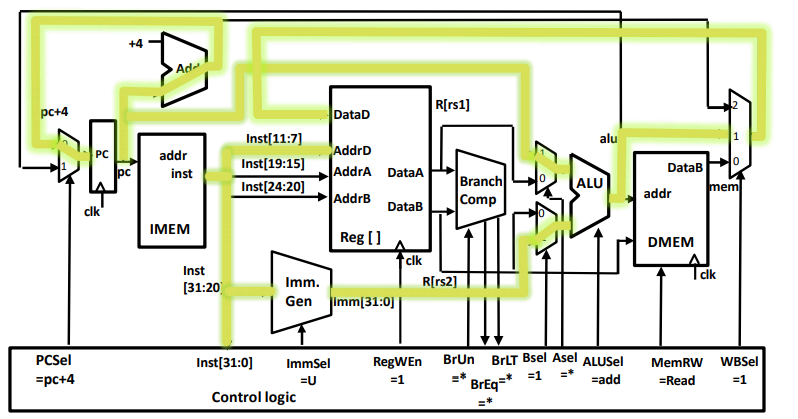
*Datapath cho lệnh lui:*



Hình 1. 31: Datapath lệnh Lui

* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo.
* Sau khi lệnh được tìm nạp, thanh ghi đích rd được giải mã bởi các trường của lệnh đó với địa chỉ là Addr D ứng với thanh ghi data D trong tệp thanh ghi.
* giá trị imm đưa vào khối Imm Gen (ImmSel =U)
* Khối ALU để thực hiện phép cộng logic số học.
* Kết quả đầu ra của ALU là giá trị thanh ghi rd sẽ được ghi vào thanh ghi Data D ứng với địa chỉ Addr D. Tín hiệu điều khiển RegWrite =1 cho phép thực hiện việc ghi dữ liệu này.
* Sau khi thực thi xong PC sẽ cập nhật lên 4 để nhảy đến câu lệnh tiếp theo.

*Datapath cho lệnh auipc:*



Hình 1. 32: Datapath lệnh AUIPC

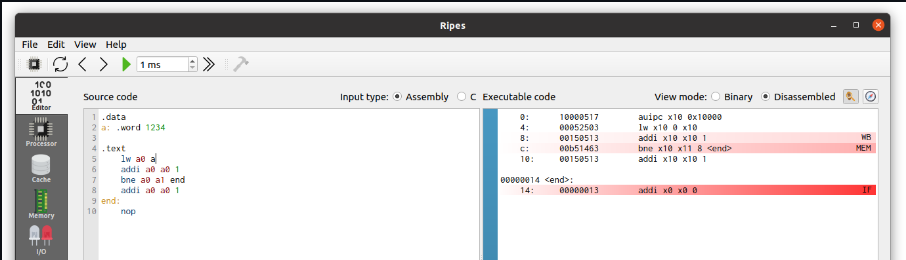
* Lệnh bắt đầu bằng cách sử dụng bộ đếm chương trình PC để cung cấp địa chỉ lệnh cho bộ nhớ lệnh, PC sẽ tăng lên 4 để xác định địa chỉ lệnh tiếp theo.
* Sau khi lệnh được tìm nạp, thanh ghi đích rd được giải mã bởi các trường của lệnh đó với địa chỉ là Addr D ứng với thanh ghi data D trong tệp thanh ghi.
* Giá trị imm đưa vào khối Imm Gen (ImmSel =U)
* Khối ALU để thực hiện phép cộng logic số học với đầu vào là PC và imm
* Kết quả đầu ra của ALU là giá trị thanh ghi rd sẽ được ghi vào thanh ghi Data D trong bộ nhớ ứng với địa chỉ Addr D. Tín hiệu điều khiển RegWEn =1 cho phép thực hiện việc ghi dữ liệu này.
* Sau khi thực thi xong PC sẽ cập nhật lên 4 để nhảy đến câu lệnh tiếp theo.

# Thực hành

## Giới thiệu phần mềm ripes:

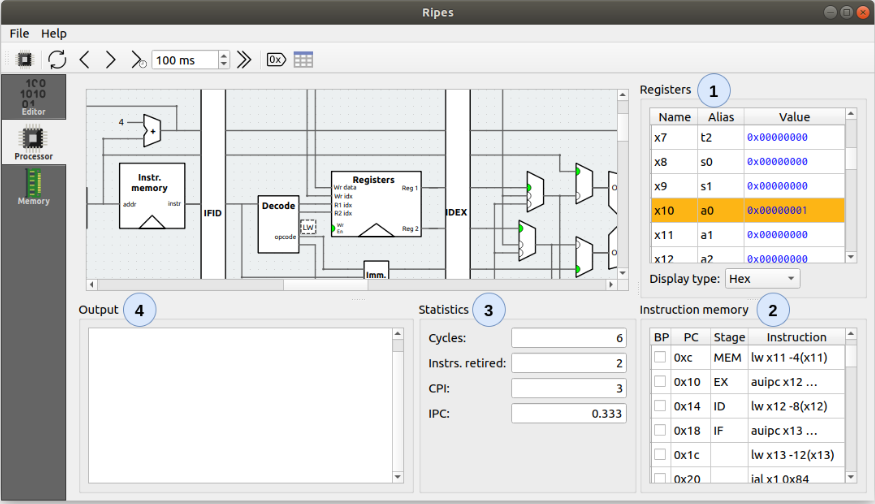
Ripes là một trình mô phỏng đồ họa bộ xử lý và môi trường soạn thảo mã hợp ngữ tương ứng với kiến trúc tập lệnh RISC-V, phù hợp với lập trình hợp ngữ và thực thi các biến trong vi kiến trúc.

Phần mềm bao gồm các chức năng chính liên quan đến datapath như:



Hình 2. : The Editor Tab

* **The editor tab**: ở đây chia ra 2 phần ở phía bên trái, nơi có thể viết các chương trình hợp ngữ, C. Sau đó có thể biên dịch và thực thi các chương trình đó. Phía bên phải nơi các câu lệnh assembly của cấu trúc lệnh RISC-V được hiển thị.



Hình 2. : The Processor Tab

**The processor tab**: Tab Processor hiển thị sơ đồ các đường dữ liệu của bộ xử lí được chọn, cũng như các tín hiệu liên quan khi chạy. Ngoài đường dữ liệu, Tab Processor còn có các tab nhỏ hơn được liệt kê sau đây:

***1- Registers***: Tập các thanh ghi của bộ xử lí. Các giá trị của thanh ghi có thể được chỉnh sửa bằng cách click chuột vào phần Value và thay đổi. Thay đổi giá trị thanh ghi sẽ ngay lập tức thể hiện trên tệp các thanh ghi. Giá trị vừa được thay đổi sẽ được tô đậm màu vàng.

***2- Intruction memory*** : Hiển thị các lệnh của chương trình hiện tại đang được thực thi trong bộ xử lí:

o BP: Breakpoints, click để đánh dấu. Các breakpoint được đánh dấu trong Editor sẽ được thể hiện ở của sổ này

o PC: Địa chỉ của lệnh hiện tại

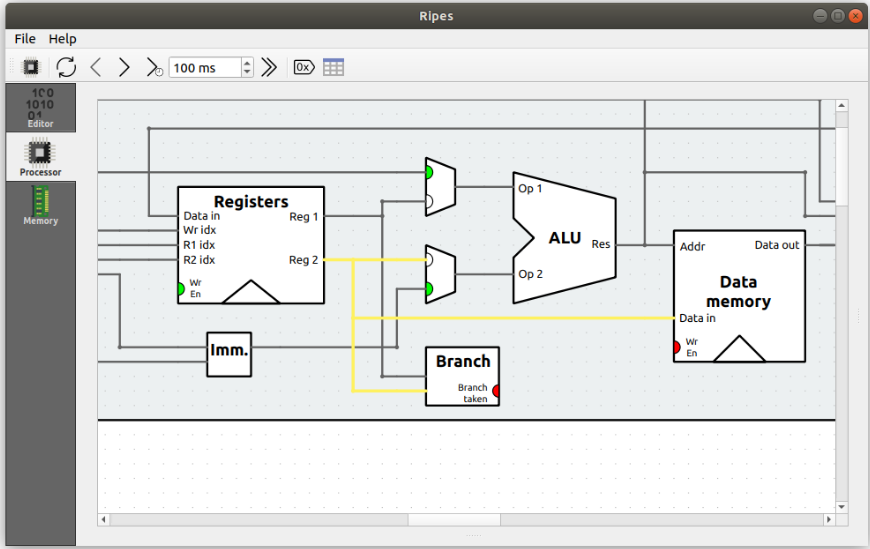
o Stage: Trạng thái thực hiện của lệnh

o Instruction: Lệnh dưới dạng hợp ngữ

***3- Statistics***: Thống kê số chu kì thực hiện, số lệnh đã thực hiện, số chu kì trên 1 lệnh.

***4- Output***: : Output của các system call ghi giá trị ra ngoại vi (ecall)

**The processor view:** Các mô hình bộ xử lý trong Ripes mô phỏng các trạng thái của datapath thông qua hình ảnh trực quan.



Hình 2. 3: Datapath

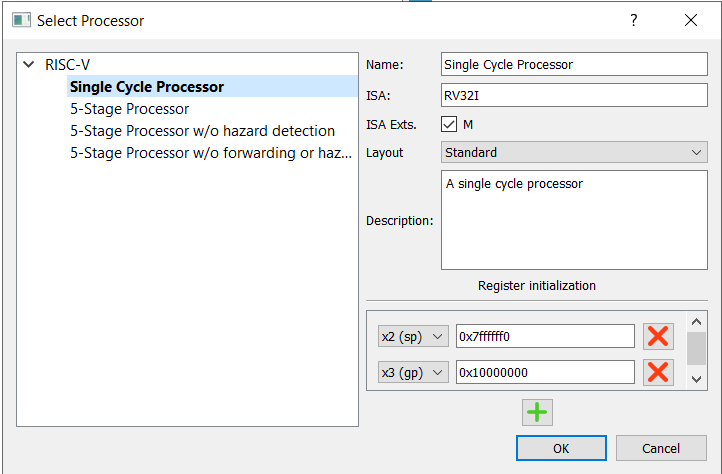
**Selecting Processor Model**: Do hỗ trợ mô phỏng nhiều mô hình bộ xử lí, Ripes cung cấp khả năng thực hiện mô phỏng sự khác nhau giữa các kiến trúc khi thực hiện 1 chương trình. Danh sách các mô hình được hỗ trợ trong phiên bản 2.0 (phía dưới) giúp người dùng thấy được sự khác nhau, mức độ phức tạp khi chuyển từ bộ xử lí đơn xung nhịp sang bộ xử lí đường ống. Ripes hỗ trợ các mô hình bộ xử lí dưới đây:

* *RISC-V Single Cycle Processor*
* *RISC-V 5-Stage Processor w/o Forwarding or Hazard Detection*
* *RISC-V 5-Stage Processor w/o Hazard Detection*
* *RISC-V 5-Stage Processor*

Ở mỗi mô hình, Ripes cung cấp nhiều layout khác nhau. Mặc định, 2 layout dưới đây được hỗ trợ

* **Standard**: Mô hình đơn giản, khối điều khiển và các tín hiệu điều khiển bị ẩn
* **Extended**: Mô hình mở rộng, khối điều khiển, các tín hiệu điều khiển cùng với độ rộng bus được thể hiện

Mở hộp thoại Select Processor để lực chọn và cấu hình cho bộ xử lí theo yêu cầu



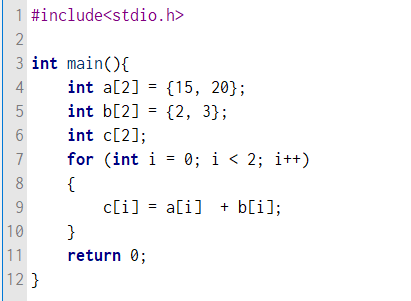
Hình 2. 4: Select Processor

Ở bên trái, các mô hình bộ xử lí được liệt kê. Để cấu hình cho bộ xử lí, layout sẽ được chọn ở thẻ Layout. Chú ý, việc chọn layout sẽ không ảnh hưởng đến hoạt động của bộ xử lí, chỉ hiển thị cụ thể hơn các thành phần tín hiệu điều khiển. Người dùng có thể khởi tạo giá trị các thanh ghi ở đây. Giá trị khởi tạo sẽ được dán mỗi khi bộ xử lí được Reset

## Mô phỏng:

Trong phần này em sẽ trình bầy các thực hiện 1 chương trình C mô phỏng datapath của bộ xử lý đơn xung nhịp.

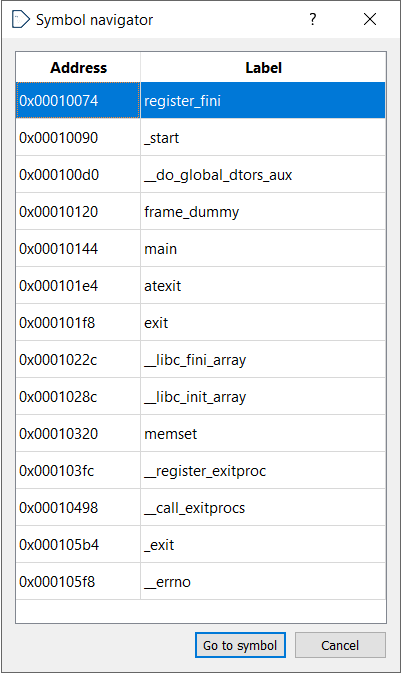
*Chương trình C được thực hiện:*



Hình 2. 5: Code C

Ở đây em lựa chọn một trương trình C đơn giản là cộng hai mảng với nhau, cụ thể là 2 mảng mỗi mảng 2 phần tử và kết quả được lưu lại ở một mảng khác.

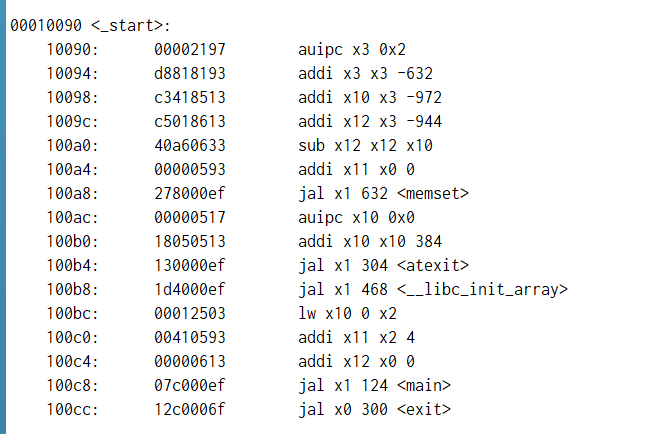
Sau khi viết được code và kiểm tra không có lỗi bước tiếp theo chạy chương trình C đó chuyển qua assembly code.



Hình 2. 6: Label Assembly Code

Sau khi biên dịch được chúng ta sẽ thấy được một chương trình assembly code có cấu trúc như hình trên. Chúng ta sẽ tập trung vào các Label \_start, main, exit.

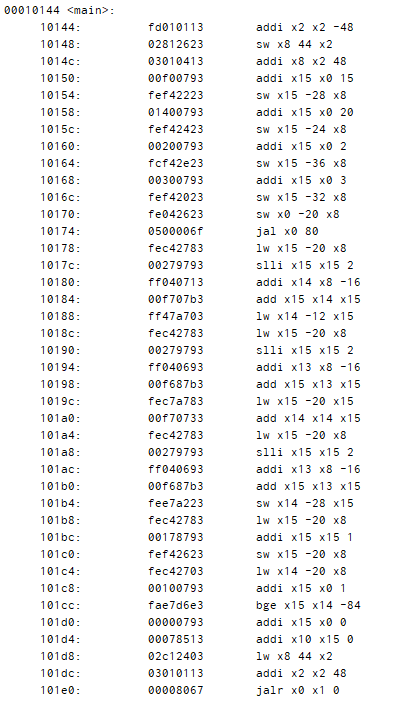
Khi bắt đầu chạy thì lệnh sẽ được thực hiện đầu tiên ở label \_start:



Hình 2. 7: Label \_Start in Assembly Code

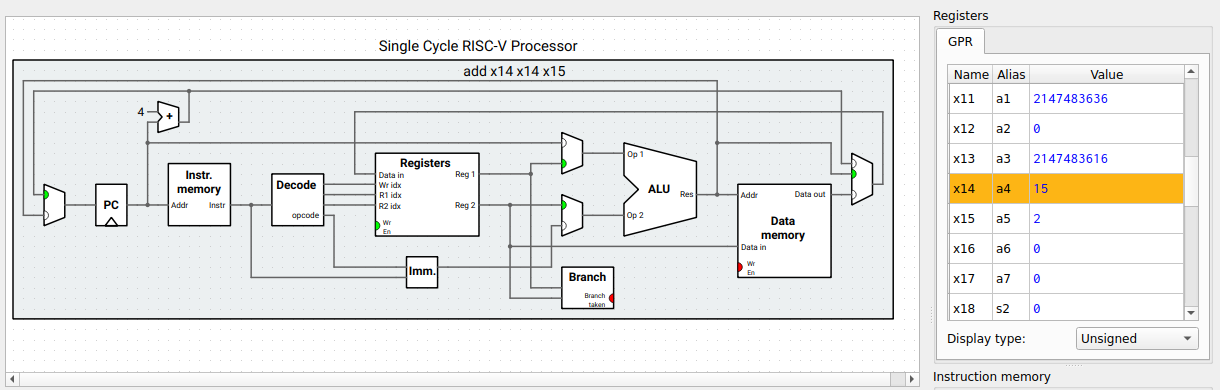
Ở phần này thì chương trình sẽ tạo ra các lệnh khởi động của bộ xử lý sau đó các lệnh sẽ được thực hiện ở hàm main.

Sau khi các giá trị khởi tạo được thực hiện xong lúc này chương trình sẽ nhảy đến hàm main thông qua câu lệnh jal x1 124 <main>:



Hình 2. 8: Label <main> Assembly Code

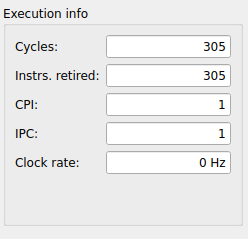
ở trong hàm main công việc chính là sẽ thực hiện cộng 2 mảng lại với nhau thông qua các dữ liệu khởi tạo trước đó.



Hình 2. 9: Datapath lệnh add

Hình trên mô phỏng việc cộng 2 phần tử đầu tiên của hai mảng với nhau. Hiện tại giá trị đầu tiên của mảng a đang lưu thanh ghi x14 với giá trị là 15, giá trị đầu tiên của mảng b đang được lưu ở thanh ghi x15 với giá trị là 2. Tiếp theo chương trình sẽ thực hiện **add x14 x14 x15**  để cộng 2 phần tử của mảng kết quả lưu lại ở thanh ghi **x14.**

Sau khi chương trình thực hiện hoành thành:



Hình 2. 10: Thống kê chương trình

Như thống kê của Ripes thì chương trình C như trên đã thực hiện 305 cycles, 305 instruction.

Tài liệu kham khảo

[1] John L. Hennessy, D. A. (n.d.). *Computer Organization And Design.* 1993.

[2] Petersen, M. B. (2021, 6 27). *Building and Executing C programs with Ripes*. Retrieved from https://github.com/mortbopet/Ripes/wiki/Building-and-Executing-C-programs-with-Ripes

[3] Petersen, M. B. (2021, 6 27). *Ripes Introduction*. Retrieved from https://github.com/mortbopet/Ripes/wiki/Ripes-Introduction