

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



**BÁO CÁO**  
**KIẾN TRÚC MÁY TÍNH**

**Đề tài:**

**Mô phỏng đường dữ liệu bộ xử lý RISC-V cho chương trình C bằng Ripes**

Giảng viên hướng dẫn: TS. Tạ Thị Kim Huệ

Sinh viên thực hiện:

MSSV

Trần Thế Rinh

20172783

Hà Nội, 12/2021

## LỜI MỞ ĐẦU

Máy tính ngày càng trở thành một công cụ không thể thiếu cũng như không thể thay thế được trong đời sống thường nhật. Ứng dụng công nghệ thông tin trong sinh hoạt hằng ngày, trong sản xuất ra của cải vật chất cũng như trong công việc điều hành, quản lý ngày càng phổ biến. Có thể nói, mọi người, không phân biệt giới tính hay tuổi tác đều tìm được ở công cụ này một niềm hứng khởi, say mê, kể cả trong giải quyết công việc cũng như học hỏi, nghiên cứu sáng tạo hay giải trí. Cấu tạo của máy tính ngày càng hiện đại, tinh vi và phức tạp, bao gồm nhiều thành phần chức năng, đòi hỏi sự liên kết, hợp tác của nhiều ngành khoa học, công nghệ mũi nhọn tạo nên. Kiến trúc máy tính (Computer Architecture) là ngành khoa học nghiên cứu hoạt động, tổ chức máy tính từ các thành phần chức năng cơ bản- cấu trúc và tổ chức phần cứng, tập lệnh - mà qua đó, các lập trình viên có thể nhận thấy, sử dụng, khai thác và sáng tạo để đáp ứng tốt hơn, đầy đủ hơn những yêu cầu của người dùng.

Trong môn học này, em không chỉ có thêm nhiều kiến thức mới về hệ vi xử lý, cấu trúc cũng như hoạt động của một máy tính cơ bản, mà còn được trực tiếp thiết kế và mô phỏng lại một cấu trúc đơn giản của vi xử lý theo mô hình RISC-V pipeline. Em xin chân thành cảm ơn cô, TS. Tạ Thị Kim Huệ đã tận tình hướng dẫn em trong quá trình tìm hiểu và thực hiện đề tài.

## Mục Lục

Chương I. Tổng quan về quá trình biên dịch.....	5
1.1 Tổng quan về kiến trúc máy tính.....	5
1.2 Quá trình biên dịch một chương trình.....	5
Chương II. Giới thiệu về kiến trúc RISC V và các thành phần cơ bản.....	11
2.1 Định nghĩa.....	11
2.2 Tổng quan về các giai đoạn của datapath.....	11
Chương III. Tổng quan về phần mềm ripes.....	19
3.1 Giới thiệu.....	19
3.2 Thanh công cụ.....	19
3.3 Các tab chính.....	20
3.3.1 Tab editor.....	20
3.3.2 Tab processor.....	21
3.3.3 Tab memory.....	22
3.3.4 Tab cache.....	23
3.3.5 Tab I/O .....	24
Chương IV. Mô phỏng và thực thi chương trình C trên ripes.....	26
4.1 Toolchain.....	26
4.2 Toolchain registration.....	27
4.3 Compiling and executing a C program.....	28
Chương V. Kết luận.....	31
Tài liệu tham khảo.....	32

## Danh mục hình vẽ

Hình 1. Sơ đồ phân cấp dịch cho ngôn ngữ C.....	6
Hình 2. Đoạn mã trước(bên trái) và sau (bên phải) tiền xử lý.....	7
Hình 3. Ví dụ về Compiler.....	8
Hình 4. Các giai đoạn cơ bản của thực thi chỉ lệnh.....	12
Hình 5. Khối instruction memory.....	14
Hình 6. Bộ cộng.....	14
Hình 7. Khối Register file.....	16
Hình 8. Khối Memory.....	18
Hình 9. Thanh công cụ Ripes.....	19
Hình 10. Tab Editor.....	20
Hình 11. Tab Processer.....	21
Hình 12. Tab memory.....	22
Hình 13. Tab Cache.....	23
Hình 14. Tab I/O.....	24
Hình 15. Toolchain Registration.....	27
Hình 16. Một chương trình C.....	28
Hình 17. Thực thi chương trình C.....	28
Hình 18. Đường dữ liệu RISC-V.....	29
Hình 19. Tab Cache.....	30
Hình 20. Tab Memory.....	30

# Chương I: Tổng quan về quá trình biên dịch

## 1.1 Tổng quan về kiến trúc máy tính

Trong kỹ thuật máy tính, kiến trúc máy tính là thiết kế và cấu trúc hoạt động căn bản của 1 hệ thống máy tính. Ngoài ra, nó cũng có thể được định nghĩa như là việc lựa chọn và kết nối các thành phần phần cứng để tạo thành các máy tính đáp ứng được các mục đích về tính năng, hiệu suất và giá cả.

Kiến trúc máy tính bao gồm ít nhất 3 khái niệm cơ bản cần nắm rõ:

- Kiến trúc tập lệnh: là hình ảnh trừu tượng của một hệ thống tính toán được nhìn từ góc độ của một lập trình viên sử dụng ngôn ngữ máy (hay hợp ngữ), bao gồm tập lệnh, cách đánh địa chỉ bộ nhớ (memory address modes), các thanh ghi, và các định dạng địa chỉ và dữ liệu.
- Tổ chức máy tính: là một mô tả bậc thấp, cụ thể hơn về hệ thống. Mô tả này nói về các bộ phận cấu thành của hệ thống được kết nối với nhau như thế nào và chúng hoạt động tương hỗ như thế nào để thực hiện kiến trúc tập lệnh.
- Thiết kế hệ thống: bao gồm tất cả các thành phần phần cứng khác bên trong một hệ thống tính toán như các đường kết nối hệ thống như bus (máy tính) và switch, các bộ điều khiển bộ nhớ (memory controller) và các cây phả hệ bộ nhớ, các cơ chế CPU off-load như Direct memory access (truy nhập bộ nhớ trực tiếp), các vấn đề như đa xử lý (multi-processing). Thông thường, trong các ứng dụng giúp con người giao tiếp với máy thường được viết rất phức tạp với hàng nghìn, thậm chí hàng triệu dòng mã với các ngôn ngữ bậc cao với các thư viện phức tạp. Ngoài ra, máy tính chỉ có thể hiểu được và thực hiện những lệnh của các ngôn ngữ bậc thấp. Do đó, cần phải có phần mềm biên dịch, giúp diễn giải các câu lệnh bậc cao ra các câu lệnh bậc thấp để máy tính có thể hiểu được.

## 1.2 Quá trình biên dịch một chương trình

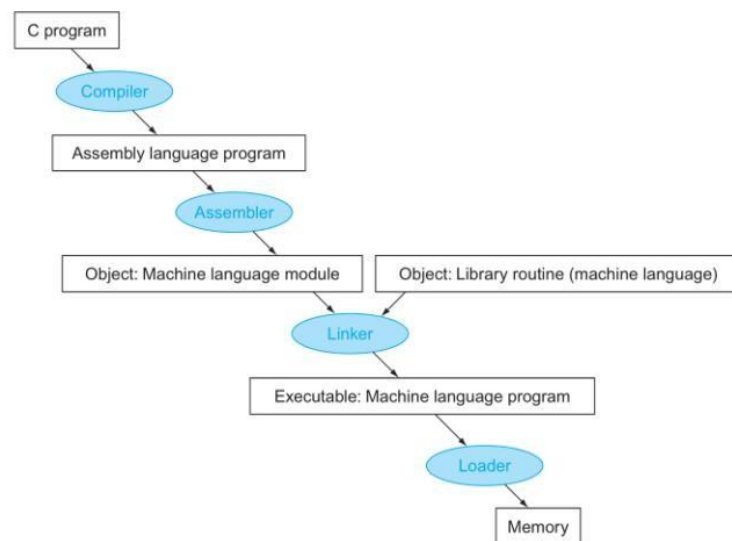
### 1.2.1 Định nghĩa

## KIẾN TRÚC MÁY TÍNH

Quy trình dịch là quá trình chuyển đổi từ ngôn ngữ bậc cao (NNBC) (C/C++, Pascal, Java, C#...) sang ngôn ngữ đích (ngôn ngữ máy) để máy tính có thể hiểu và thực thi. Ngôn ngữ lập trình C là một ngôn ngữ dạng biên dịch. Chương trình được viết bằng C muốn chạy được trên máy tính phải trải qua một quá trình biên dịch để chuyển đổi từ dạng mã nguồn sang chương trình dạng mã thực thi.

Quá trình được chia ra làm 4 giai đoạn chính:

- Giai đoạn tiền xử lý (Pre-processor)
- Giai đoạn dịch NNBC sang Assembly (Compiler)
- Giai đoạn dịch assembly sang ngôn ngữ máy (Assembler)
- Giai đoạn liên kết (Linker)



Hình 1. Hệ thống phân cấp dịch cho ngôn ngữ C

### 1.2.2 Hoạt động

#### ❖ Giai đoạn tiền xử lý – Preprocessor

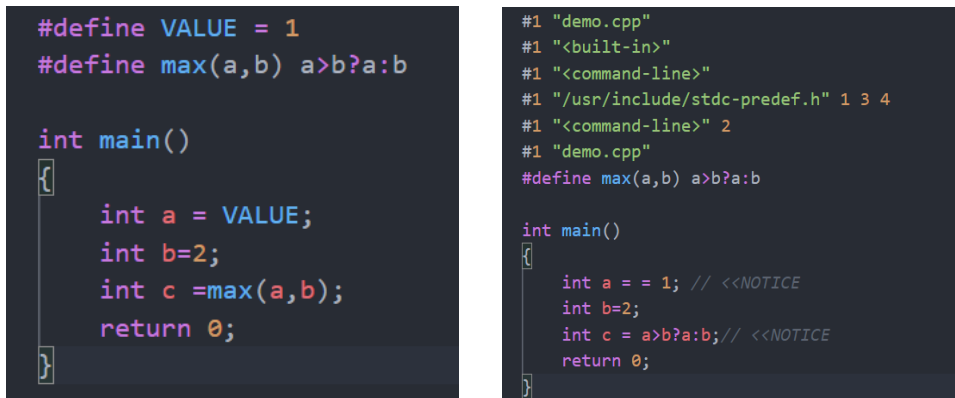
Bộ tiền xử lý chịu trách nhiệm về những việc sau:

- Lấy mã nguồn.
- Xóa tất cả các bình luận, bình luận chương trình.

- Các lệnh tiền xử lý (bắt đầu bằng #) cũng được xử lý.

Chúng ta có thể bắt lỗi thông qua việc sử dụng đúng cách ở giai đoạn này Các lệnh #if và #error. Bằng cách sử dụng tùy chọn -E của trình biên dịch, như hình dưới đây. Tiếp theo, chúng ta có thể dừng quá trình biên dịch trong giai đoạn tiền xử lý (nếu có) Lỗi ở giai đoạn này.

Ví dụ: lệnh #include cho phép thêm mã chương trình của tệp tiêu đề nhập mã nguồn cần dịch. Hằng số được xác định bằng #define sẽ thay thế bằng một giá trị cụ thể tại mỗi vị trí sử dụng trong chương trình.



```
#define VALUE = 1
#define max(a,b) a>b?a:b

int main()
{
    int a = VALUE;
    int b=2;
    int c =max(a,b);
    return 0;
}
```

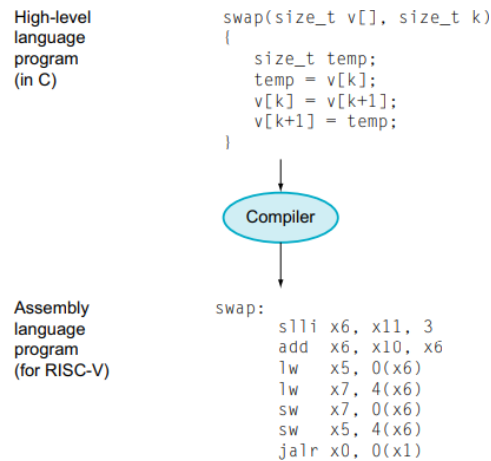
```
#1 "demo.cpp"
#1 "<built-in>"
#1 "<command-line>"
#1 "/usr/include/stdc-predef.h" 1 3 4
#1 "<command-line>" 2
#1 "demo.cpp"
#define max(a,b) a>b?a:b

int main()
{
    int a = 1; // <<NOTICE
    int b=2;
    int c = a>b?a:b; // <<NOTICE
    return 0;
}
```

Hình 2. Đoạn mã trước(bên trái) và sau (bên phải) tiền xử lý

### ❖ Công đoạn dịch ngôn ngữ bậc cao sang Assembly (Compiler)

Bước biên dịch được thực hiện trên mỗi đầu ra của bộ tiền xử lý. biên tập Phân tích cú pháp mã nguồn C ++ thuần túy (hiện không có bất kỳ chỉ thị tiền xử lý nào Logic) và chuyển nó thành mã hợp ngữ, một ngôn ngữ ký hiệu có thể chuyển giao Chuyển nó thành định dạng nhị phân mà máy tính có thể hiểu được. Ở giai đoạn này, trình biên dịch sẽ bắt lỗi kiểu dữ liệu, phân tích cú pháp (cú pháp) và có thể Tự động tối ưu mã nguồn để chương trình chạy hiệu quả hơn. Nếu có lỗi, trình biên dịch sẽ thông báo cho chúng tôi và chúng tôi phải chỉnh sửa mã để xử lý.



Hình 3. Ví dụ về Compiler

Quá trình này sẽ biên dịch các tệp tin .c/.cpp sang tệp tin “object.s”

### ❖ Công đoạn dịch Assembler

Trên hệ thống UNIX, tệp mã đối tượng có hậu tố “.o”(.OBJ trên Windows) Tệp mã đối tượng chứa mã đã biên dịch (nhị phân) .Các ký hiệu (ký hiệu được hiểu đơn giản ở đây là hàm và biến) được định nghĩa trong nhập tệp nguồn. Các ký hiệu trong tệp mã đối tượng được tham chiếu bằng tên. Tệp đối tượng của hệ thống UNIX thường chứa sáu phần riêng biệt:

- Tiêu đề tệp đích mô tả kích thước và vị trí của phần các tệp mục tiêu khác.
- Đoạn văn bản chứa mã ngôn ngữ máy.
- Phân đoạn dữ liệu tĩnh chứa vòng đời chương trình.
- Hướng dẫn xác định thông tin định vị và từ dữ liệu phụ thuộc vào địa chỉ tuyệt đối khi chương trình được tải vào trí nhớ.
- Bảng ký hiệu chứa các thẻ chưa xác định còn lại, chẳng hạn như tài liệu tham khảo bên ngoài.



- Thông tin gỡ lỗi chứa mô tả ngắn gọn về cách tiến hành mô-đun được biên dịch để trình gỡ lỗi có thể so sánh các hướng dẫn máy với C tập tin nguồn và làm cho cấu trúc dữ liệu có thể đọc được.

### ❖ Giai đoạn Linker

Những gì chúng tôi đã thể hiện cho đến nay cho thấy rằng chỉ có một sự thay đổi đối với một dòng yêu cầu toàn bộ quá trình biên dịch và lắp ráp chương trình. Việc dịch lại toàn bộ sẽ lãng phí tài nguyên máy tính. đặc biệt là nó đặc biệt lãng phí cho các chương trình thư viện tiêu chuẩn, bởi vì các lập trình viên sẽ biên dịch và lắp ráp hầu như không bao giờ được xử lý theo định nghĩa biến đổi. Một cách khác là biên dịch và lắp ráp từng chương trình một cách độc lập thiết lập để việc thay đổi một dòng chỉ yêu cầu biên dịch và lắp ráp chương trình ở đó. Phương pháp thay thế này yêu cầu một chương trình hệ thống mới được gọi là một trình soạn thảo liên kết hay trình liên kết (Linker), chương trình này các chương trình ngôn ngữ máy được tập hợp độc lập và kết hợp chúng với cùng với nhau. Lý do khiến trình liên kết hữu ích là việc vá mã nhanh hơn nhiều so với biên dịch dịch và lắp ráp lại.

Nó liên kết tất cả các tệp mã đối tượng bằng cách thay thế các tham chiếu ký hiệu với địa chỉ chính xác. Mỗi ký hiệu này có thể được định nghĩa trong các tệp hoặc thư viện mã đối tượng khác. Nếu họ được chỉ định có nghĩa là trong các thư viện khác với thư viện tiêu chuẩn, bạn cần sử dụng giới thiệu về trình liên kết của họ (điều này được thực hiện thông qua xây dựng cấu hình).

Trình liên kết tạo một tệp thực thi có thể chạy trên máy tính. Thông thường, tệp này có cùng định dạng với tệp đích, ngoại trừ nó không chứa các tham chiếu chưa được giải quyết. Có thể có các tệp được liên kết một phần của liên kết, chẳng hạn như quy trình thư viện, vẫn chưa được giải quyết phân tích cú pháp và do đó dẫn đến tệp đích. Ở giai đoạn này, các lỗi phổ biến nhất là thiếu định nghĩa hoặc định nghĩa trùng lặp.

- Thiếu định nghĩa điều này có nghĩa là các định nghĩa không tồn tại (các class, các hàm không được implement) hoặc object code files hoặc thư viện nơi chúng cư trú không được cung cấp cho trình linker biết.

- Trùng lặp cùng một symbol được định nghĩa trong hai hoặc nhiều object code files hoặc thư viện khác nhau.

- Chương trình chính không có hàm main ().

### ❖ Load:

Loader là một chương trình hệ thống, đặt một chương trình đối tượng (object program) vào bộ nhớ chính để nó sẵn sàng thực thi.

Ở giai đoạn này, tệp thực thi (executable file) đã nằm trên đĩa (disk), hệ điều hành sẽ đọc tệp đó vào bộ nhớ và khởi động tệp. Trình tải (loader) thực hiện theo các bước sau trong hệ thống UNIX:

- Đọc tiêu đề tệp thực thi để xác định kích thước của phân đoạn văn bản và dữ liệu.
- Tạo không gian địa chỉ đủ lớn cho văn bản và dữ liệu.
- Sao chép các hướng dẫn và dữ liệu từ tệp thực thi vào bộ nhớ.
- Sao chép các tham số (nếu có) của chương trình chính vào ngăn xếp.
- Khởi tạo các thanh ghi bộ xử lý và đặt con trỏ ngăn xếp đến vị trí trống đầu tiên.
- Các nhánh tới một quy trình khởi động sao chép các tham số vào các thanh ghi đối số và gọi quy trình chính của chương trình. Khi quy trình chính trở lại, quy trình khởi động sẽ kết thúc chương trình bằng một lệnh gọi hệ thống thoát.

## Chương II. Datapath trong bộ xử lý RISC V

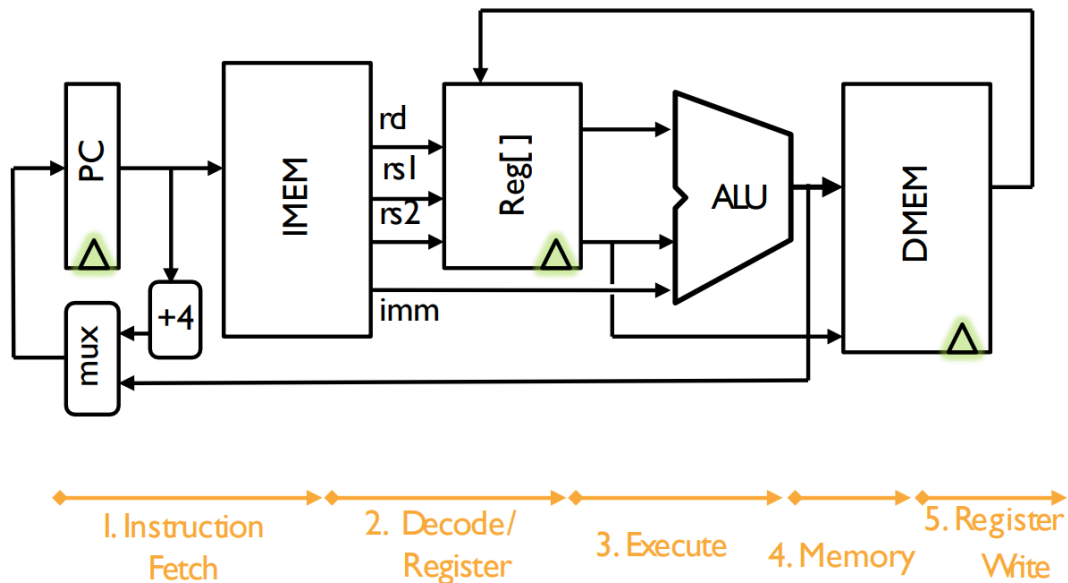
### 2.1 Định nghĩa

Datapath là một đơn vị được sử dụng để thực thi hoặc giữ dữ liệu trong một bộ xử lý. Trong triển khai RISC V, datapath bao gồm tập lệnh, data memories, register file, ALU và adder.

### 2.2 Tổng quan về các giai đoạn của datapath

- Vấn đề: một khối "nguyên khối" duy nhất "thực thi một hướng dẫn" (thực hiện tất cả các hoạt động cần thiết bắt đầu với việc tìm nạp hướng dẫn) sẽ quá công kênh và không hiệu quả
- Giải pháp: chia nhỏ quy trình "thực hiện một lệnh" thành các giai đoạn và sau đó kết nối các giai đoạn để tạo ra toàn bộ đường dẫn dữ liệu.
  - Các giai đoạn nhỏ dễ thiết kế hơn.
  - Dễ dàng tối ưu hóa (thay đổi) một giai đoạn mà không cần chạm và những người khác (mô-đun).
- 5 giai đoạn trong datapath:
  - Giai đoạn 1: Instruction Fetch (IF)
  - Giai đoạn 2: Instruction Decode (ID)

- Giai đoạn 3: Execute(EX) – ALU
- Giai đoạn 4: Memory Access(MEM)
- Giai đoạn 5: Write Back to Register (WB)



Hình 4. Các giai đoạn cơ bản của thực thi chỉ lệnh

1. Tìm nạp lệnh(IF): Cho thấy lệnh đang được đọc từ bộ nhớ bằng địa chỉ trong PC và sau đó được đặt trong thanh ghi đường ống IF / ID. Địa chỉ PC được tăng thêm 4 và sau đó được ghi lại vào PC để sẵn sàng cho chu kỳ xung nhịp tiếp theo. PC này cũng được lưu trong thanh ghi đường dẫn IF / ID trong trường hợp sau này cần thiết cho một lệnh, chẳng hạn như beq. Máy tính không thể biết loại lệnh nào đang được tìm nạp, vì vậy nó phải chuẩn bị cho bất kỳ lệnh nào, chuyển thông tin có thể cần thiết xuống đường dẫn.

2. Giải mã lệnh và đọc tệp thanh ghi (ID: Hiển thị phân lệnh của thanh ghi đường ống IF / ID cung cấp trường ngay lập tức, được mở rộng dấu hiệu đến 64 bit, và số thanh ghi để đọc hai thanh ghi. Tất cả ba giá trị được lưu trữ trong thanh ghi đường ống ID / EX, cùng với địa chỉ PC. Chúng tôi lại chuyển mọi thứ có thể cần thiết bởi bất kỳ lệnh nào trong chu kỳ xung nhịp sau đó.

3. Thực thi hoặc tính toán địa chỉ(EX – ALU: Lệnh tải đọc nội dung của một thanh ghi và dấu mở rộng ngay lập tức từ thanh ghi đường ống ID / EX và thêm chúng bằng cách sử dụng ALU. Tổng đó được đặt trong thanh ghi đường ống EX / MEM.

4. Truy cập bộ nhớ (MEM): Lệnh tải đọc bộ nhớ dữ liệu bằng cách sử dụng địa chỉ từ thanh ghi đường ống EX / MEM và tải dữ liệu vào thanh ghi đường ống MEM / WB.

5. Ghi lại (WB): Đọc dữ liệu từ thanh ghi đường ống MEM / WB và ghi nó vào tệp thanh ghi ở giữa hình.

a, Giai đoạn tìm nạp lệnh(IF)

+ Giai đoạn nạp lệnh:

- Sử dụng thanh ghi Program Counter (PC) để tìm nạp lệnh từ bộ nhớ: thanh ghi PC là một thanh ghi đặc biệt trong bộ vi xử lý.
- Tăng giá trị trong thanh ghi PC lên 4 đơn vị để lấy địa chỉ của lệnh tiếp theo

+ Kết quả của giai đoạn này là đầu vào cho giai đoạn tiếp theo:

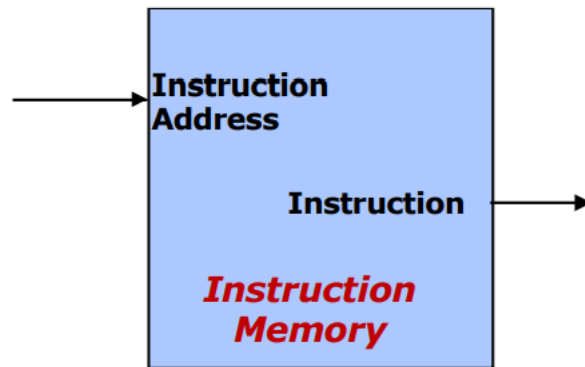
Kết quả của giai đoạn này là 32 bit mã máy của lệnh cần thực thi. Chuỗi 32 bit này sẽ sử dụng như đầu vào input cho giai đoạn tiếp theo là decode.

- Khối Instruction Memory

Vùng nhớ lưu trữ lệnh

Đầu vào: là địa chỉ của lệnh

Đầu ra: là nội dung lệnh tương ứng với địa chỉ được cung cấp

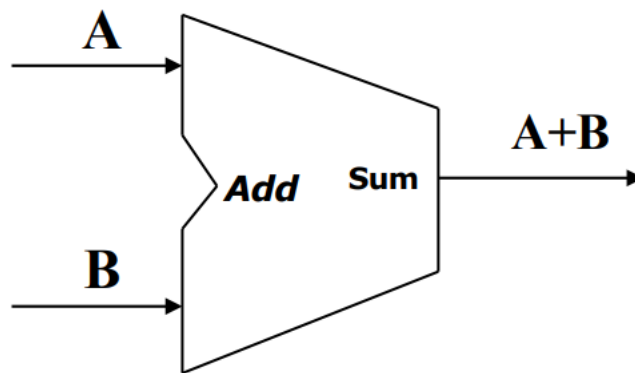


Hình 5: Khối instruction memory

- Bộ cộng

Mạch logic kết hợp để cộng 2 số - bộ cộng

Đầu vào: hai số 32 bit A,B



Đầu ra:  $A + B$

Hình 6: Bộ cộng

b, Giai đoạn giải mã (Decode)

- Giai đoạn decode:

Lấy nội dung dữ liệu trong các trường của lệnh:

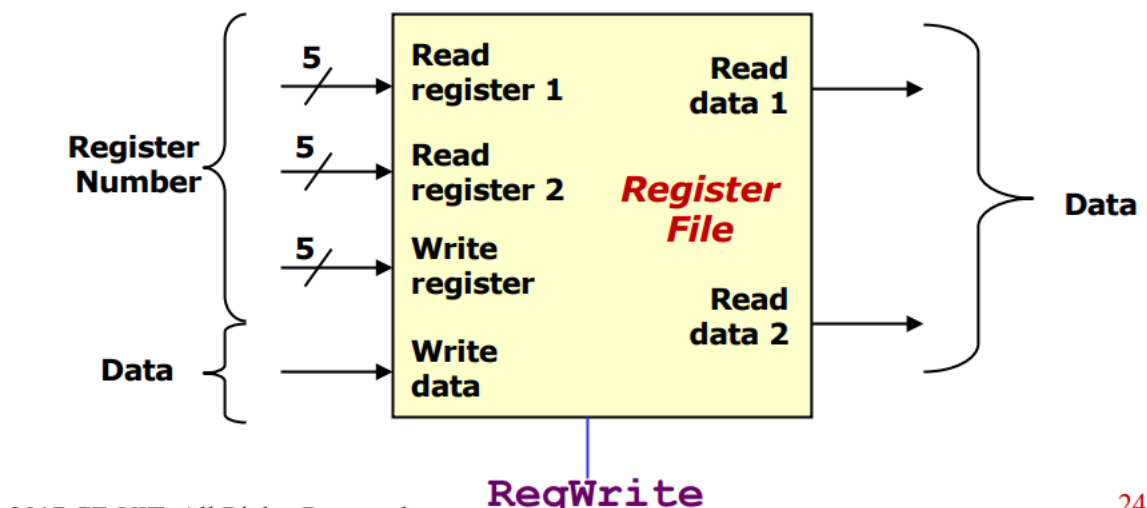
- + Đọc opcode để xác định kiểu lệnh và chiều dài của từng trường trong mã máy.
- + Đọc dữ liệu từ các thanh ghi cần thiết
  - Đầu vào từ giai đoạn trước (Fetch): Lệnh cần được thực thi (mã máy)
  - Đầu ra cho giai đoạn tiếp theo (Execute): Phép tính và các toán hạng cần thiết.
- + Khởi Register File

Mỗi tập 32 thanh ghi:

- Mỗi thanh ghi có chiều dài 32 bit và có thể được đọc hoặc ghi bằng cách chỉ ra chỉ số của thanh ghi.
- Với mỗi lệnh, cho phép đọc nhiều nhất từ 2 thanh ghi.
- Với mỗi lệnh, cho phép ghi vào nhiều nhất 1 thanh ghi.

RegWrite: là một tín hiệu điều khiển nhằm mục đích:

- Cho phép ghi vào một thanh ghi hay không
- 1(True) = Write, 0 (False)= No Write



Hình 7. Khối Register file

## + Khối Multiplex(Mux)

- Chức năng: chọn một input từ tập input đầu vào
- Input: n đường vào có cùng chiều rộng
- Control: cần m bit trong đó  $n = 2^m$
- Output: Chọn đường input thứ I nếu giá trị tín hiệu điều khiển  $control = i$

## c, Giai đoạn Execute – ALU

- ALU: Arithmetic-Logic Unit
- Công việc thật sự của hầu hết các lệnh được hiện chủ yếu trong giai đoạn này:
  - o Số học (Arithmetic) (ví dụ: add, sub), Logic (ví dụ: and, or): ALU tính ra kết quả cuối cùng
  - o Lệnh làm việc với bộ nhớ (ví dụ: lw, sw): ALU dùng tính toán địa chỉ của bộ nhớ.
  - o Lệnh nhảy/nhánh (ví dụ: bne, beq): ALU thực hiện so sánh các giá trị trên thanh ghi và tính toán địa chỉ đích sẽ nhảy tới
- Đầu vào từ giai đoạn trước (Decode): Các thao tác (operation) và toán hạng (operands).



## KIẾN TRÚC MÁY TÍNH

- Đầu ra cho giai đoạn tiếp theo (Memory): Tính toán kết quả

(Đối với lệnh lw và sw : Kết quả của giai đoạn này sẽ là địa chỉ cung cấp cho memory để lấy dữ liệu.

### + Khối ALU

- Chức năng: Khối dùng để thực hiện các phép tính logic và số học
- Input: hai số 32 bit
- Điều khiển khối ALU: Do ALU có thể thực hiện nhiều chức năng => dung 4 bit để quyết định chức năng/ phép toán cụ thể nào cho ALU
- Outputs: Kết quả của phép toán số học hoặc logic, mỗi bit tín hiệu để chỉ ra rằng kết quả có bằng 0 hay không.

### d, Giai đoạn truy xuất vùng nhớ (Memory)

+ Giai đoạn truy xuất vùng nhớ: Chỉ có lệnh Load và Store cần thực hiện các thao tác trong giai đoạn này:

- Sử dụng địa chỉ vùng nhớ được tính toán ở giai đoạn ALU
- Đọc dữ liệu ra hoặc ghi dữ liệu vào vùng nhớ dữ liệu

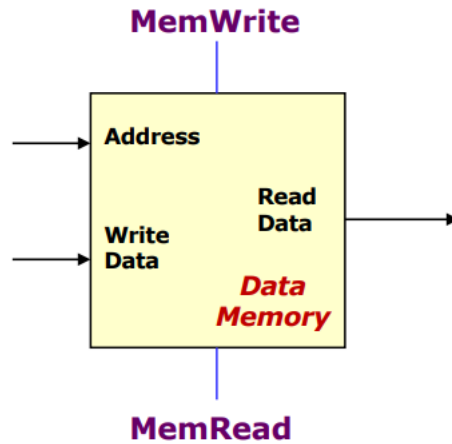
Tất cả các lệnh khác sẽ rảnh trong giai đoạn này

+ Đầu vào từ giai đoạn trước (ALU): Kết quả tính toán được dùng làm địa chỉ vùng nhớ (nếu có thể ứng dụng).

+ Đầu ra cho giai đoạn tiếp theo (Result Write): Kết quả được lưu trữ lại (nếu cần).

### + Khối Data Memory

- Vùng nhớ này lưu trữ dữ liệu cần thiết của chương trình.
- Input:
  - Address: Địa chỉ vùng nhớ
  - Write Data: Dữ liệu sẽ được ghi vào vùng nhớ đối với lệnh store
- Tín hiệu điều khiển: tín hiệu đọc (MemRead) và ghi (MemWrite) , chỉ có tín hiệu được bật lên tại bất kỳ một thời điểm nào.
- Output: Dữ liệu được đọc từ vùng nhớ đối với lệnh Load



Hình 8: Khối Memory

e, Giai đoạn ghi lại, lưu trữ kết quả (Write back to Register)

+ Những lệnh ghi kết quả của phép toán vào thanh ghi :

- Ví dụ: số học, logic, shift, load, set-less-than
- Cần chỉ số thanh ghi đích và kết quả tính toán

+ Những lệnh không ghi kết quả như: store, branch, jump : không ghi kết quả và những lệnh này sẽ rảnh trong giai đoạn này.

+ Đầu vào từ giai đoạn trước (Memory): Kết quả tính toán hoặc là từ memory hoặc là từ ALU

+ Giai đoạn Write back to register không có thêm bất kì thành phần nào khác: chỉ đơn giản đưa kết quả vào thanh ghi ( ngõ Write data của khối Register/Register file), chỉ số của thanh ghi được ghi vào (ngõ vào write register) được sinh ra trong giai đoạn ID.

## CHƯƠNG III: Tổng quan về phần mềm RIPES

### 3.1 Giới thiệu

Ripes là trình mô phỏng bộ xử lý đồ họa và trình soạn thảo mã lắp ráp được xây dựng cho kiến trúc tập lệnh RISC-V, thích hợp để dạy cách mã cấp độ lắp ráp được thực thi trên các vi kiến trúc khác nhau.

### 3.2 Thanh công cụ



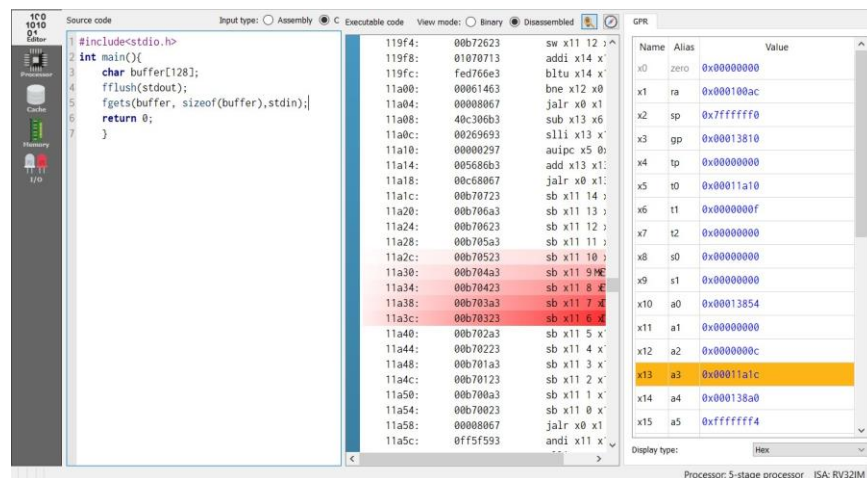
Hình 9. Thanh công cụ RIPES

- Select Processor: Mở hộp thoại chọn bộ xử lý
- Reset: Đặt lại bộ xử lý, đặt bộ đếm chương trình thành điểm vào của chương trình hiện tại và đặt lại bộ nhớ giả lập.
- Reverse: Hoàn tác một chu kỳ đồng hồ.
- Clock: Đồng hồ tắt cả các phần tử bộ nhớ trong mạch và cập nhật trạng thái của mạch. 15
- Auto - clock: Đồng hồ mạch với tần số nhất định được chỉ định bởi khoảng thời gian tự động đồng hồ. Đồng hồ tự động sẽ dừng sau khi chạm vào điểm ngắt.
- Run: Thực thi trình mô phỏng mà không cần thực hiện cập nhật GUI để nhanh nhất có thể. Mọi ecalls chức năng in sẽ vẫn được in ra bảng điều khiển đầu ra. Việc chạy sẽ dừng lại sau khi chạm điểm ngắt hoặc một lần thoát ecalls đã được thực hiện.
- Show stage table: Hiển thị biểu đồ cho biết hướng dẫn nào nằm trong (các) giai đoạn đường ống cho mỗi chu kỳ. Các giai đoạn tạm dừng được biểu thị bằng giá trị '-'. Lưu ý: Thông tin giai đoạn không được ghi lại trong khi thực thi bộ xử lý thông qua tùy chọn Chạy.
- Chọn View->Show processor signal values để hiển thị tất cả các giá trị

cổng đầu ra của bộ xử lý.

## 3.3 Các Tab chính

### 3.3.1 Tab Editor



Hình 10: Tab Editor

Tab trình chỉnh sửa hiển thị hai đoạn mã. Ở phía bên trái, có thể viết một chương trình hợp ngữ được viết bằng các tập lệnh RISC-V RV32 (I/ M/ C). Bất cứ khi nào thực hiện bất kỳ chỉnh sửa nào trong chương trình hợp ngữ này- và không tìm thấy lỗi cú pháp- mã hợp ngữ sẽ tự động được lắp ráp và chèn vào trình mô phỏng. Nếu một trình biên dịch C đã được đăng ký, thì kiểu đầu vào có thể được đặt thành C. Sau đó, có thể viết, biên dịch và thực thi các chương trình ngôn ngữ C trong Ripes.

Tiếp theo, ở phía bên phải, một chế độ xem mã thứ hai được hiển thị. Đây là chế độ xem không tương tác của chương trình hiện tại ở trạng thái đã lắp ráp của nó, được ký hiệu là trình xem chương trình. Chúng tôi có thể xem chương trình đã lắp ráp dưới dạng hướng dẫn RISC-V đã được tháo rời hoặc dưới dạng mã nhị phân phân thô. Có thể nhấp vào thanh bên màu xanh lam của chế độ xem bên phải để đặt điểm ngắt tại địa chỉ mong muốn.

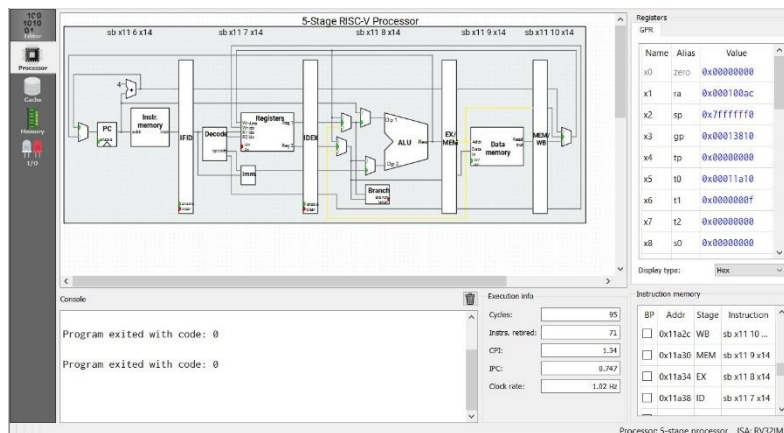
Ripes được đóng gói với nhiều ví dụ khác nhau về các chương trình hợp ngữ RISC-V, có thể được tìm thấy trong thanh menu File-> Load Examples.

## 3.3.2 Tab Processor

Tab trình chỉnh sửa hiển thị hai đoạn mã. Ở phía bên trái, có thể viết một chương trình hợp ngữ được viết bằng các tập lệnh RISC-V RV32 (I/ M/ C). Bất cứ khi nào thực hiện bất kỳ chỉnh sửa nào trong chương trình hợp ngữ này- và không tìm thấy lỗi cú pháp- mã hợp ngữ sẽ tự động được lắp ráp và chèn vào trình mô phỏng. Nếu một trình biên dịch C đã được đăng ký, thì kiểu đầu vào có thể được đặt thành C. Sau đó, có thể viết, biên dịch và thực thi các chương trình ngôn ngữ C trong Ripes.

Tiếp theo, ở phía bên phải, một chế độ xem mã thứ hai được hiển thị. Đây là chế độ xem không tương tác của chương trình hiện tại ở trạng thái đã lắp ráp của nó, được ký hiệu là trình xem chương trình. Chúng tôi có thể xem chương trình đã lắp ráp dưới dạng hướng dẫn RISC-V đã được tháo rời hoặc dưới dạng mã nhị phân thô. Có thể nhấp vào thanh bên màu xanh lam của chế độ xem bên phải để đặt điểm ngắt tại địa chỉ mong muốn.

Ripes được đóng gói với nhiều ví dụ khác nhau về các chương trình hợp ngữ RISC-V, có thể được tìm thấy trong thanh menu File-> Load Examples.



Hình 11: Tab Processor

Tab Processor là nơi Ripes hiển thị chế độ xem của bộ xử lý hiện được chọn, cũng như bất kỳ thông tin bổ sung nào liên quan đến việc thực thi. Ngoài chế độ xem bộ xử lý, tab bộ xử lý chứa các chế độ xem sau:

1. Registers: Một danh sách của tất cả các thanh ghi của bộ xử lý. Giá trị

thanh ghi có thể được chỉnh sửa thông qua việc nhấp vào giá trị của thanh ghi đã cho. Việc chỉnh sửa giá trị thanh ghi ngay lập tức được phản ánh trong mạch bộ xử lý. Sổ đăng ký được sửa đổi gần đây nhất được đánh dấu bằng nền màu vàng.

2. Instruction Memory: Chế độ xem chương trình hiện tại được tải trong trình mô phỏng.

- ✓ BP Các điểm ngắt, nhấp để chuyển đổi. Bất kỳ điểm ngắt nào được đặt trong tab trình chỉnh sửa sẽ được phản ánh ở đây.
- ✓ Addr: Địa chỉ của lệnh đã cho
- ✓ Stage: Liệt kê (các) giai đoạn hiện đang thực hiện lệnh đã cho
- ✓ Instruction: Hướng dẫn tháo rời

3. Execution info: Các số liệu thống kê khác nhau dựa trên số lượng chu kỳ và số lượng lệnh ngừng hoạt động hiện tại.

4. Solution: Bất kỳ đầu ra nào thông qua chức năng ecall in sẽ được hiển thị ở đây.

### 3.3.3 Tab Memory

Address	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x00000098	0x00000000	0x00	0x00	0x00	0x00
0x00000094	0x00000000	0x00	0x00	0x00	0x00
0x00000090	0x00000000	0x00	0x00	0x00	0x00
0x0000008c	0x00000003	0x03	0x00	0x00	0x00
0x00000088	0x00000000	0x00	0x00	0x00	0x00
0x00000084	0x00013864	0x64	0x38	0x01	0x00
0x00000080	0x00000000	0x00	0x00	0x00	0x00
0x0000007c	0x0070003	0x03	0x00	0x07	0x00
0x00000078	0x00000000	0x00	0x00	0x00	0x00
0x00000074	0x00013854	0x54	0x38	0x01	0x00
0x00000070	0x00000000	0x00	0x00	0x00	0x00
0x0000006c	0x00060003	0x03	0x00	0x06	0x00
0x00000068	0x00000000	0x00	0x00	0x00	0x00
0x00000064	0x00013840	0x40	0x38	0x01	0x00
0x00000060	0x00000000	0x00	0x00	0x00	0x00
0x0000005c	0x00050003	0x03	0x00	0x05	0x00

Name	Size	Range
.symtab	2496	0x00000000 - 0x0000009c0
.text	10700	0x00010074 - 0x00012a40
.eh_frame	4	0x00013000 - 0x00013004
.init_array	8	0x00013004 - 0x0001300c
.fini_array	4	0x0001300c - 0x00013010
.data	2096	0x00013010 - 0x00013840
.sdata	20	0x00013840 - 0x00013854
.sbss	0	0x00013854 - 0x00013854
.bss	0	0x00013864 - 0x00013864
LED Matrix 0	3500	0xf0000000 - 0xf0000dac
D-Pad 0	16	0xf0000dac - 0xf0000dbc
Switches 0	4	0xf0000dbc - 0xf0000dc0

Hình 12. Tab memory

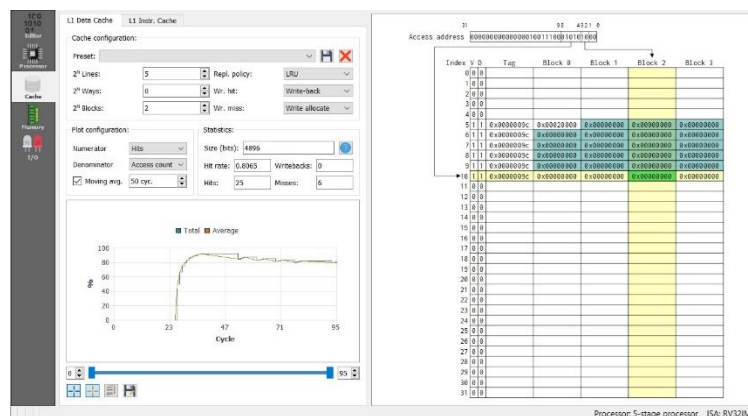
Tab bộ nhớ cung cấp một cái nhìn vào toàn bộ không gian địa chỉ có thể xác định của bộ xử lý. Điều hướng bộ nhớ có thể được thực hiện như sau

- Cuộn chế độ xem bộ nhớ
- Tới đăng ký sẽ cuộn chế độ xem bộ nhớ đến giá trị hiện có trong thanh ghi đã chọn

- Đi tới phần sẽ cuộn chế độ xem bộ nhớ đến địa chỉ của giá trị phần đã cho trong bộ nhớ (tức là. textphân đoạn bộ nhớ lệnh, dataphân đoạn dữ liệu tĩnh, v.v.). Hơn nữa, một địa chỉ tùy chỉnh có thể được chỉ định thông qua tùy chọn "Địa chỉ".

Bất cứ khi nào bộ xử lý được đặt lại, tất cả bộ nhớ được ghi trong quá trình thực thi chương trình sẽ được đặt lại về trạng thái ban đầu.

## 3.3.4 Tab Cache



Hình 13: Tab Cache

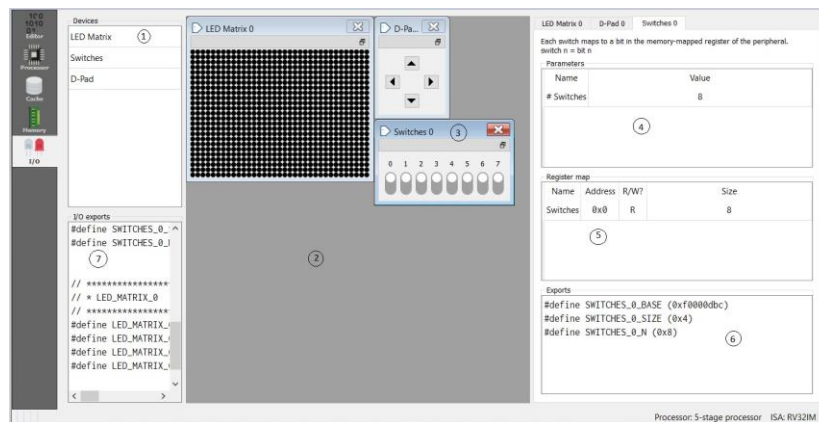
Kể từ phiên bản 2.1.0, Ripes bao gồm mô phỏng bộ nhớ cache. Trình mô phỏng bộ nhớ cache mô phỏng bộ nhớ đệm L1D (dữ liệu) và L1I (lệnh), trong đó có thể định cấu hình bố trí và hành vi của từng loại bộ nhớ cache. Do đó, chúng ta có thể phân tích hiệu suất bộ nhớ cache của các chương trình để xem các thiết kế bộ nhớ cache khác nhau tương tác như thế nào với các mẫu truy cập bộ nhớ mà chương trình của chúng ta thể hiện. Một số lưu ý chung về mô phỏng bộ nhớ cache trong Ripes:

- Bạn nên sử dụng bộ xử lý chu kỳ đơn để mô phỏng bộ nhớ cache, với điều kiện: o Nếu mô phỏng với các mô hình bộ xử lý pipelined, có thể xảy ra trường

hợp chúng ta đang dừng một giai đoạn hiện đang đọc từ bộ nhớ. Nếu một giai đoạn bị dừng, mỗi chu kỳ bị dừng sẽ được tính là một lần truy cập bộ nhớ bổ sung. Đây là hoạt động dành riêng cho việc triển khai và như vậy có thể giống hoặc không giống với các hệ thống máy tính khác

- o Mô hình bộ vi xử lý chu kỳ đơn có tốc độ thực thi nhanh hơn đáng kể so với các mô hình bộ vi xử lý đường ống.
- Các kiểu bộ xử lý không truy cập trình mô phỏng bộ nhớ cache khi truy cập bộ nhớ. Thay vào đó, bộ mô phỏng bộ nhớ đệm kết nối vào một mô hình bộ xử lý 13 và phân tích các truy cập bộ nhớ trong mỗi chu kỳ đồng hồ. Sau đó, các truy cập bộ nhớ này được sử dụng như một dấu vết để thực hiện mô phỏng bộ nhớ cache của chúng tôi. Ý nghĩa của việc này là:
  - Ripes không mô phỏng độ trễ truy cập bộ nhớ cache. Do đó, Ripes không cung cấp bất kỳ ước tính nào về thời gian thực thi CPU thực tế, mà chỉ có thể cung cấp thông tin chi tiết về các yếu tố như tỷ lệ bỏ lỡ, truy cập và ghi lại.
  - Các dòng bộ nhớ cache bản (khi bộ nhớ cache được định cấu hình ở chế độ ghi ngược) sẽ vẫn hiển thị trong chế độ xem bộ nhớ. Nói cách khác, các từ luôn được ghi qua bộ nhớ chính, ngay cả khi bộ đệm được cấu hình ở chế độ ghi ngược 1.

### 3.3.5 Tab I/O



Hình 14. Tab I/O

Kể từ phiên bản 2.2, Ripes bao gồm các thiết bị I/O được ánh xạ bộ nhớ khác nhau. Thông qua những điều này, có thể nhanh chóng nhận ra một hệ thống nhúng nhỏ. Trang sau cung cấp tổng quan về việc sử dụng các thiết bị cũng như cách tạo thiết bị của riêng bạn.

Điều hướng đến tab I / O, bộ thiết bị khả dụng có sẵn tại (1). Bấm đúp vào



bất kỳ cái nào trong số này sẽ tạo ra thiết bị. Khi một thiết bị được tạo, nó sẽ tự động được chỉ định một vị trí trong bản đồ bộ nhớ của hệ thống và từ đó có thể đọc hoặc ghi từ bộ xử lý.

Tất cả các thiết bị được hiển thị trong khu vực (2). Mọi thiết bị có thể được bật ra khỏi khu vực này nếu bạn nhấp vào nút (3). Điều này cho phép điều hướng đến các 14 tab khác của chương trình, nếu cần thiết để xem một thiết bị và tức là chương trình đang thực thi, cùng một lúc.

Ở phía bên phải, tổng quan về các chi tiết bên trong của mỗi thiết bị được hiển thị (4). Một số thiết bị có thể có các thông số cấu hình, chẳng hạn như số lượng công tắc trong thiết bị chuyển mạch hoặc kích thước của ma trận LED. (5) hiển thị bản đồ đăng ký của thiết bị. Ở đây, mỗi mục nhập liệt kê địa chỉ của thanh ghi (liên quan đến độ lệch cơ sở của vùng nhớ), cho dù thanh ghi chỉ được đọc hay ghi, cũng như độ rộng bit của thanh ghi. (6) hiển thị các ký hiệu được xuất bởi thiết bị đã chọn. Tối thiểu, điều này chứa địa chỉ cơ sở và kích thước (tính bằng byte) của thiết bị, trong bản đồ bộ nhớ.

Tập hợp tất cả các định nghĩa được xuất bởi các thiết bị được khởi tạo hiện tại được hiển thị trong (7). Các định nghĩa này có sẵn để tham khảo trong các chương trình hợp ngữ hoặc ngôn ngữ C. Để sử dụng, truy cập các thiết bị:

- Trong Assembly: các định nghĩa có thể được sử dụng ở bất kỳ đâu nơi cung cấp giá trị tức thời.
- Trong C: `#include "ripes_system.h"` trong chương trình, và tham chiếu các tên giống như bất kỳ tên nào khác `#define`

## CHƯƠNG IV: Mô phỏng và thực thi chương trình C trên RIPES

Bất kỳ mô hình bộ xử lý nào trong Ripes không phụ thuộc vào mã do người dùng lập lịch đều có thể thực thi bất kỳ tệp thực thi nào được biên dịch cho tập lệnh mà mô hình bộ xử lý triển khai.

Các phần sau. c sẽ trình bày về cách một chương trình có thể được biên dịch và sau đó được thực thi trong trình mô phỏng.

Ví dụ như chương trình## (được biên dịch trước), hãy tham khảo ranpi.c. Lưu ý rằng một phiên bản biên dịch trước của chương trình này được đóng gói cùng với Ripes là một trong những ví dụ (trong lựa chọn Ripes File-> Load Example. -> RanPI).

Đáng chú ý, chương trình này chứa các phép toán dấu phẩy động trong khi bộ xử lý mô phỏng không chứa đơn vị dấu chấm động. Do đó (được suy ra bởi thiết lập-march = rv32im, như được thấy bên dưới), trình biên dịch sẽ bao gồm các thói quen soft- float thực hiện các phép toán dấu phẩy động thông qua số học số nguyên.

### 4.1 Toolchain

Ban đầu, hãy lưu ý rằng chuỗi công cụ RISC-V có sẵn thông qua các kho lưu trữ gói tiêu chuẩn, chẳng hạn như gcc-7-riscv64-linux-gnu sẽ không thể tạo tệp có thể thực thi với phiên bản Ripes hiện tại, ngay cả khi công-march = rv32im tắc được sử dụng.

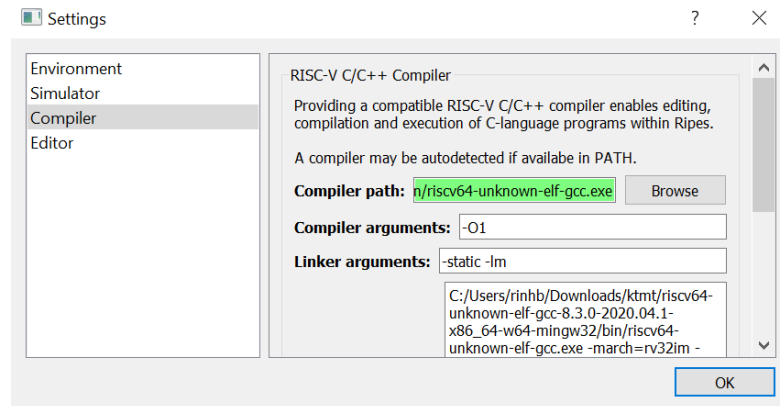
Ripes yêu cầu chuỗi công cụ RISC-V bằng kim loại trần, cụ thể là bộ tuple riscv64- unknown- elf-. Điều này có thể được tìm nạp từ:

- Tải xuống chuỗi công cụ dựng sẵn cho nền tảng của bạn từ sifive/freedom- tools. Phiên bản này được cho là hoạt động tốt với Ripes.
- Hoặc, nếu bạn muốn xây dựng chuỗi công cụ của riêng mình, hãy làm theo hướng dẫn <https://github.com/riscv/riscv-gnu-toolchain> để xây dựng chuỗi công cụ newlib.

Lưu ý: chuỗi công cụ phải được xây dựng-- enable-multilib để cho phép nhắm mục tiêu các hệ thống RISC-V 32-bit.

### 4.2 Toolchain Registration

Để mô phỏng một chương trình bằng ngôn ngữ không phải là Assembly, hãy Edit → Settings → Compiler. Sau đó, người ta có thể đưa ra một khu vực mà trình biên dịch tương thích có thể thực thi, ví dụ “riscv64-hidden-elf-gcc”. Ripes sẽ tự động cố gắng tìm kiếm tệp thực thi trong current PATH. Tuy nhiên, người ta có thể duyệt và điều hướng đến tệp thực thi trình biên dịch, Nếu không tìm thấy trình biên dịch tự động. Tuy nhiên, trường đường dẫn trình biên dịch sẽ chuyển sang màu xanh lục, nếu trình biên dịch hợp lệ đã được chỉ định.



Hình 15: Toolchain Registration

### 4.3 Compiling and Executing a C program

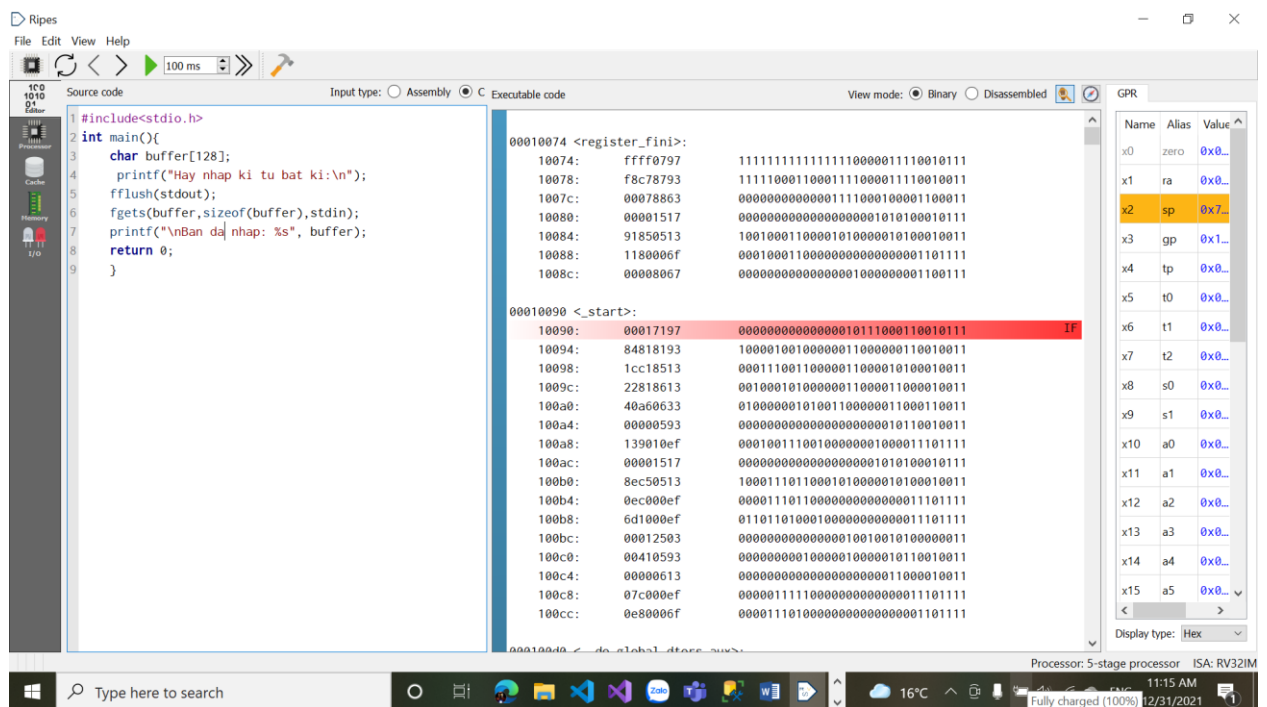
Điều hướng đến Tab Editor thảo và chọn Input Type C. (Lưu ý: sẽ xảy ra lỗi nếu chưa thiết lập trình biên dịch C hợp lệ)

## KIẾN TRÚC MÁY TÍNH

```
1 #include<stdio.h>
2 int main(){
3     char buffer[128];
4     printf("Hay nhap ki tu bat ki:\n");
5     fflush(stdout);
6     fgets(buffer,sizeof(buffer),stdin);
7     printf("\nBan da nhap: %s", buffer);
8     return 0;
9 }
```

Hình 16. Một chương trình C

Nếu không tìm thấy lỗi cú pháp nào và chương trình đã được tạo thành công, tệp thực thi được tạo sẽ tự động được tải vào trình mô phỏng và hiển thị ở bên phải trong màn hình trình chỉnh sửa. Ấn Compile C program:

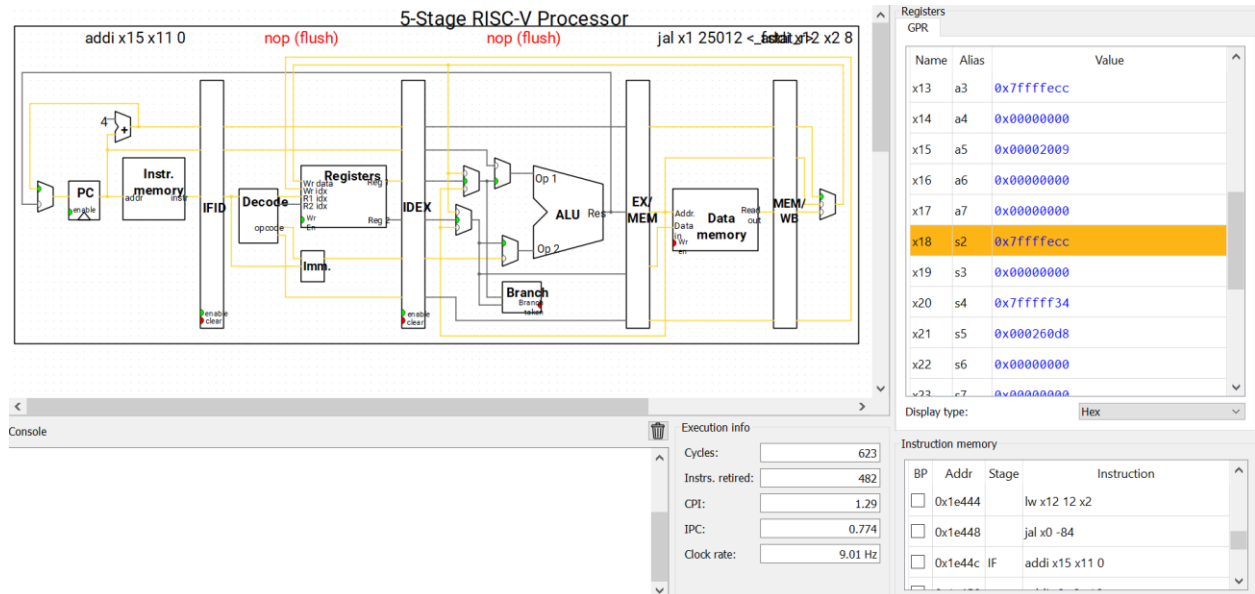


Hình 17. Thực thi chương trình C

## KIẾN TRÚC MÁY TÍNH

### + Tab Processor:

Ở phần này, chúng ta có thể quan sát được đường dữ liệu của từng câu lệnh, sự thay đổi trên các thanh ghi, các giai đoạn của từng lệnh, số lượng chu kỳ và số lượng lệnh đã hoạt động.

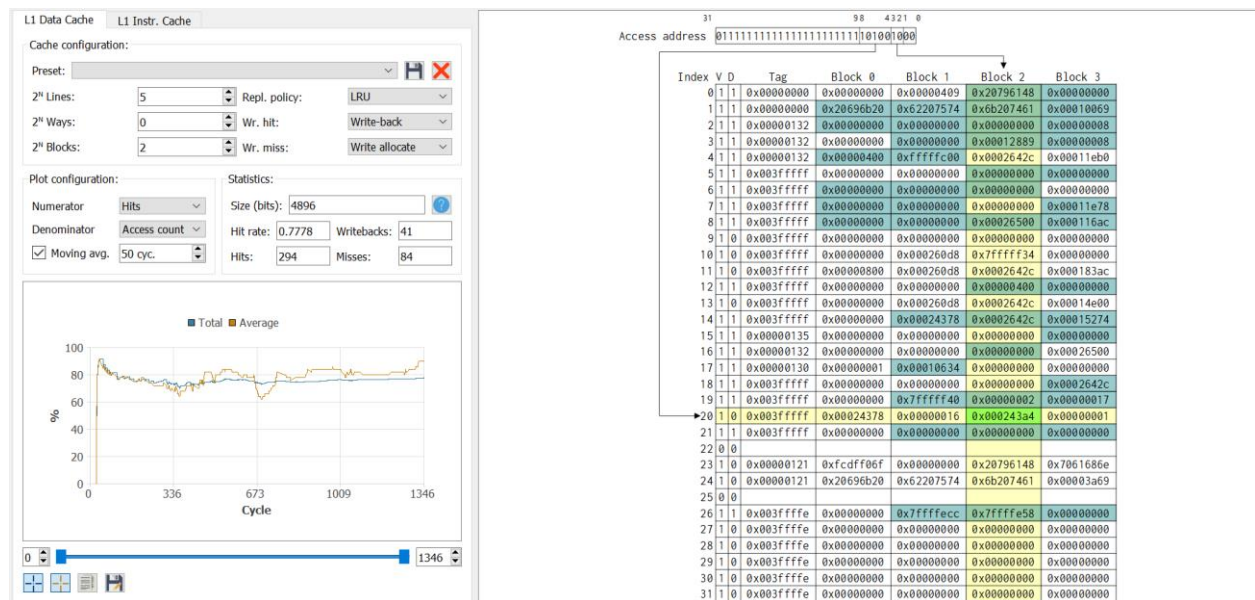


Hình 18. Đường dữ liệu RISC-V

### + Tab Cache

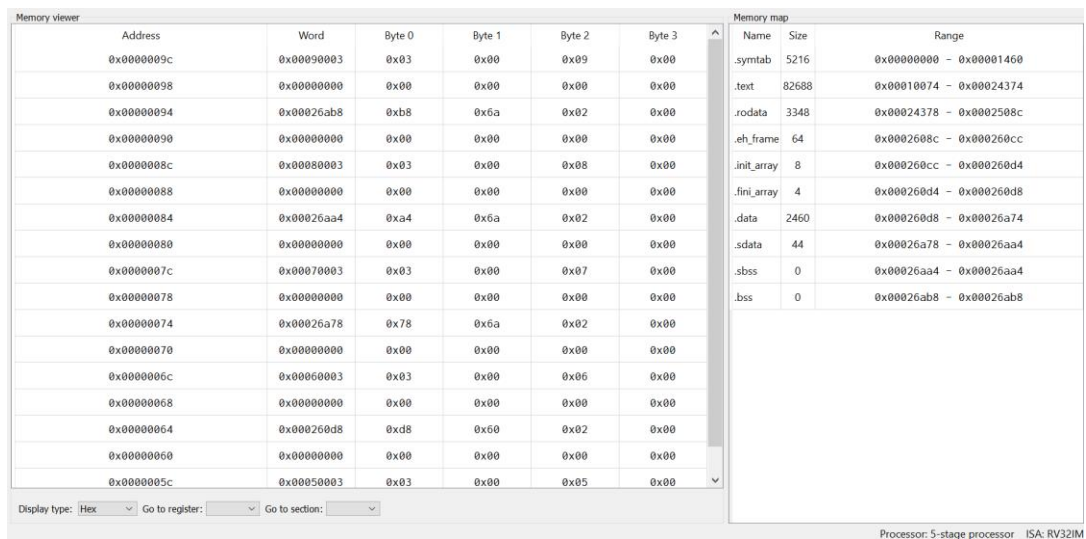
Nó sẽ phân tích hiệu suất bộ nhớ cache của các chương trình:

# KIẾN TRÚC MÁY TÍNH



### Hình 19. Tab Cache

+ Tab I/O



Hình 20. Tab Memory

## CHƯƠNG V. Kết Luận

Qua bài tập tập, sau một quá trình tìm hiểu và được sự hướng dẫn của cô Tạ Kim Huệ, em đã hoàn thành bài tập lớn với đề tài về mô phỏng đường dữ liệu bộ xử lý RISC V cho chương trình C bằng RICES. Qua đề tài em đã hiểu hơn về bộ xử lý RISC V cũng như datapath, bộ cache của nó. Sau quá trình thực hiện đề tài em đã học hỏi thêm rất nhiều kiến thức của phần cứng cấu trúc một bộ xử lý, cách lập trình và mô phỏng trên một ngôn ngữ mới. Kết thúc đề tài, em đã thu được một số kết quả nhất định. Trong thời gian thực hiện chúng em đã thu được những kết quả hỗ trợ cho các đề tài nghiên cứu tiếp theo.

Một lần nữa, nhờ có sự hướng dẫn và giảng dạy tận tình của cô, TS. Tạ Thị Kim Huệ trong các buổi học lý thuyết trên lớp, em đã có thể triển khai và hoàn thành được bài tập. Trong quá trình thực hiện bài tập lớn, em còn nhiều sai sót, mong cô đóng góp và giúp em sửa đổi để hoàn thiện hơn.

Em xin chân thành cảm ơn.

## Tài Liệu Tham Khảo

[1]. David A. Patterson & John L. Hennessy. Computer Organization and Design: The Hardware/ Software Interface, Sixth Edition.

[2]. [Building and Executing C programs with Ripes · mortbopet/Ripes Wiki · GitHub](#)

[3]. [Ripes Introduction · mortbopet/Ripes Wiki · GitHub](#)

Link source code: [RinhBKTrueMission/KTMT Project \(github.com\)](#)