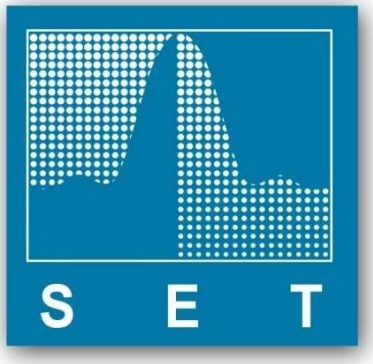
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

VIỆN ĐIỆN TỬ - VIỄN THÔNG

***



Bài Tập Lớn Môn:

KIẾN TRÚC MÁYTÍNH

Tên Đề Tài:

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

Giáo Viên Hướng Dẫn: TS. Tạ Thị Kim Huệ

Họ tên sinh viên:	Bùi Trung Kiên
Mã số SV:	20172638
Lớp:	ĐTVT.11-K62
Email:	kien.bt172638@sis.hust.edu.vn

Hà Nội – 1/2022

LỜI MỞ ĐẦU

Trải qua một kỳ học, chúng em đã được TS. Tạ Thị Kim Huệ giảng dạy bộ môn Kiến trúc máy tính. Được hiểu thêm về cấu trúc của bộ xử lý RISC-V, hay biết thêm hiệu năng của cache, … Chúng em cảm ơn cô rất nhiều đã giúp đỡ cho chúng em thời gian qua.

Trong các đề tài bảo vệ môn học, chúng em chọn đề tài "Mô phỏng đường dữ liệu bộ xử lý RISC-V cho chương trình C bằng Ripes". Để tìm hiểu sâu sắc hơn quá trình biên dịch một chương trình C và nắm rõ cấu trúc xử lý RISC-V.

Cùng với việc thực hiện đề tài này, em xin gửi lời cảm ơn sâu sắc đến giảng viên hướng dẫn TS. Tạ Thị Kim Huệ đã nhiệt tình chỉ dẫn các bước, hướng nghiên cứu, thực hiện cũng như yêu cầu cần có của đề tài trong suốt thời gian thực hiện đề tài. Trong quá trình thực hiện, dù đã rất cố gắng nhưng không tránh khỏi những sai sót và hạn chế nhất định. Vì vậy chúng em rất mong nhận được sự góp ý, bổ sung thêm của cô để đề tài được hoàn thiện hơn

Mục Lục

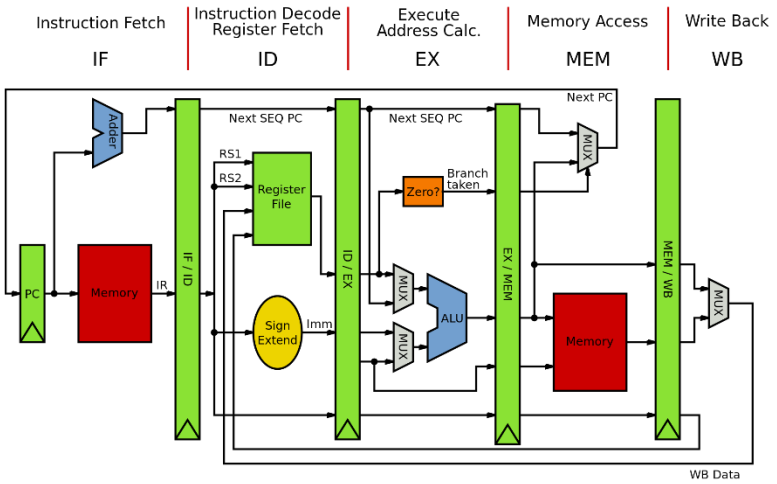
Danh Mục Hình Vẽ

Chương I. 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.



Hình 1: Một thiết kế đường ống của kiến trúc MIPS

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

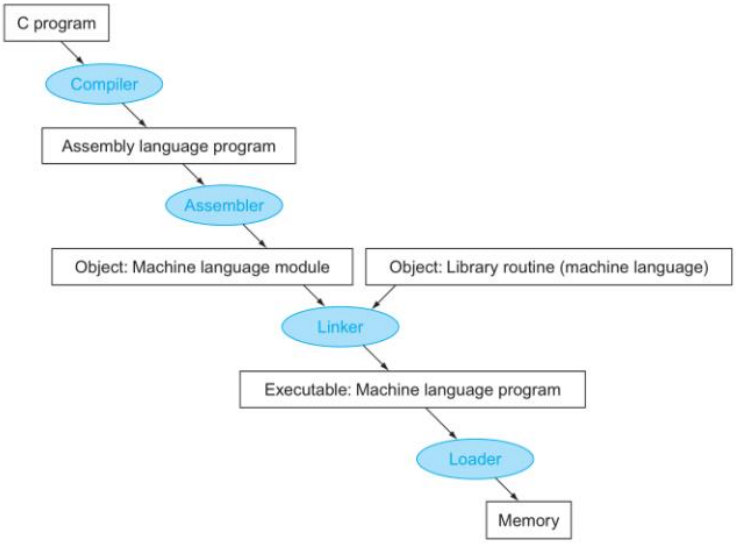
- Kiến trúc tập lệnh: là hình ảnh trừu tượng của một hệ thống tính toán được nhìn từ góc độ của một lập trình viên sử dụng ngôn ngữ máy (hay hợp ngữ), bao gồm tập lệnh, cách đánh địa chỉ bộ nhớ (memory address modes), các thanh ghi, và các định dạng địa chỉ và dữ liệu.
- Tổ chức máy tính: là một mô tả 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 thể 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

Máy tính chỉ có thể hiểu các tín hiệu điện có dạng 0 và 1, hay còn gọi là mã nhị phân. Kết hợp các mã nhị phân thành các dãy nhị phân có độ dài nhất định sẽ tạo nên mã máy. Để máy tính có thể hiểu được các câu lệnh bậc cao, ta cần chuyển đổi nó sang dạng mã máy. Để làm được điều đó máy tính phải trải qua 5 bước chính theo tuần tự: Preprocessing (Tiền xử lý), Complation (Biên dịch), Assembling (Thông dịch), Linker (Liên kết), Load (Tải).



Hình 2: Hệ thống phân cấp dịch cho ngôn ngữ C

❖ Pre-processing:

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;
}
```

Hình 3: Đoạn mã trước tiền xử lý

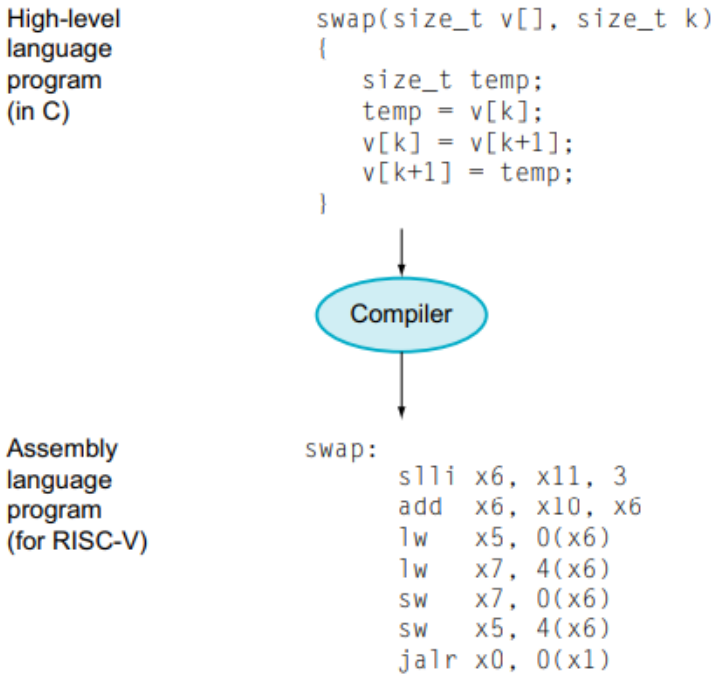
```
# 1 "hello.cpp"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "hello.cpp"

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

Hình 4: Đoạn mã sau tiền xử lý

❖ 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.

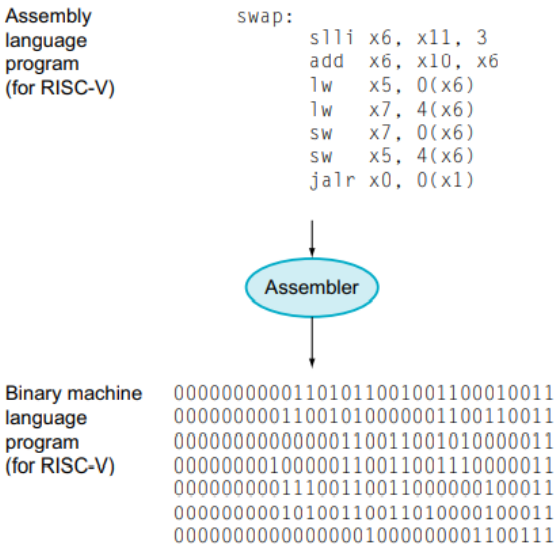


Hình 5: Ví dụ về Compiler

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

❖ Assembler:

Xử lý những lỗi này và biên dịch lại Trong bước này, trình hợp dịch sẽ chuyển đổi mã lắp ráp trong tệp hợp ngữ thành mã máy (mã mà CPU có thể hiểu được) trong tệp mã đối tượng. Xin lưu ý rằng trình lắp ráp phụ thuộc vào kiến trúc CPU (x86, PowerPC, ARM, v.v.), vì vậy các kiến trúc CPU khác nhau sẽ có các trình lắp ráp khác nhau.



Hình 6: Ví dụ về 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.

❖ 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 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 đíchỞ 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.

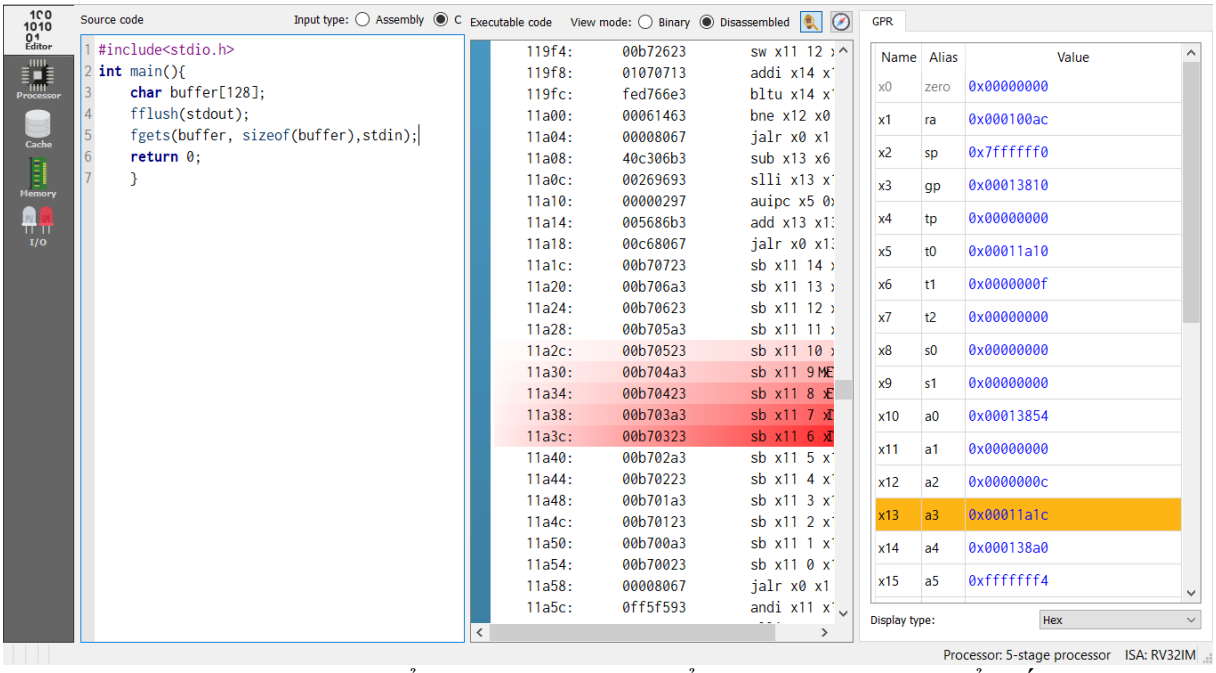
Chương II. Tổng Quan Về Phần Mềm RIPES

2.1. Giới Thiệu Khái Quát

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.

2.2. Các Tab Chính

2.2.1. Tab Editor

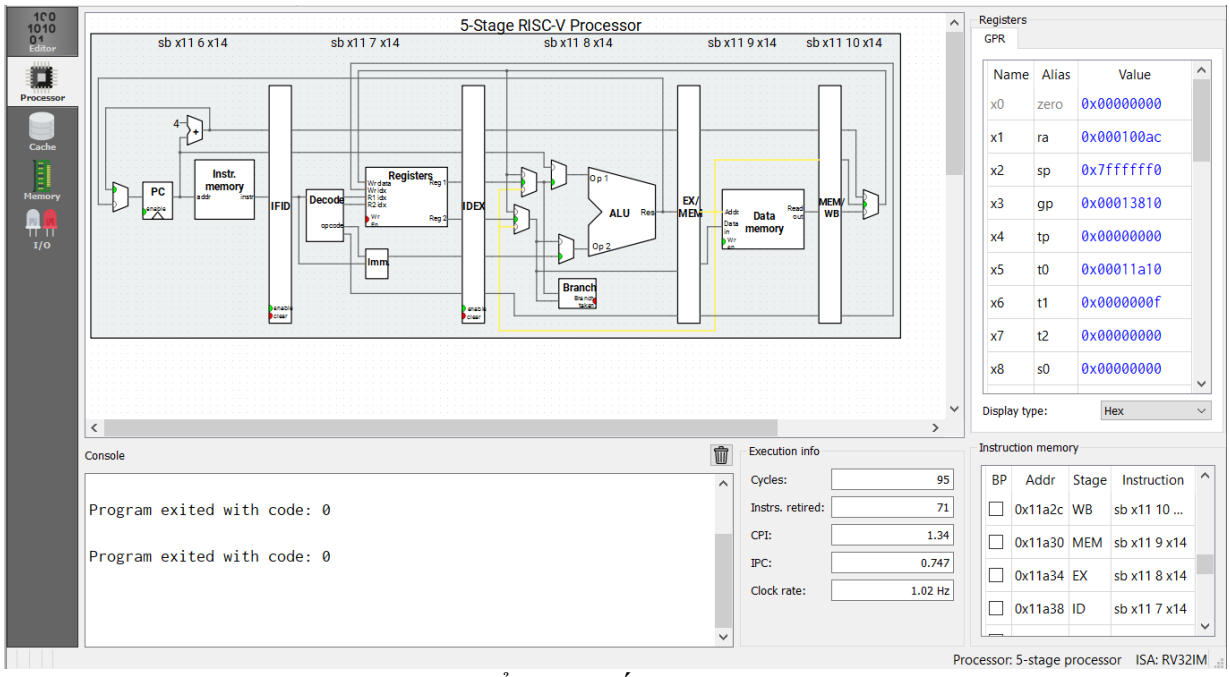


Tab trình chỉnh sửa hiển thị hai đoạn mã. Ở phía bên trái, có thể viết một chương trình hợp ngữ được viết bằng các tập lệnh RISC-V RV32 (I/ M/ C). Bất cứ khi nào thực hiện bất kỳ chỉnh sửa nào trong chương trình hợp ngữ này-và không tìm thấy lỗi cú pháp-mã hợp ngữ sẽ tự động được lắp ráp và chèn vào 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 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.

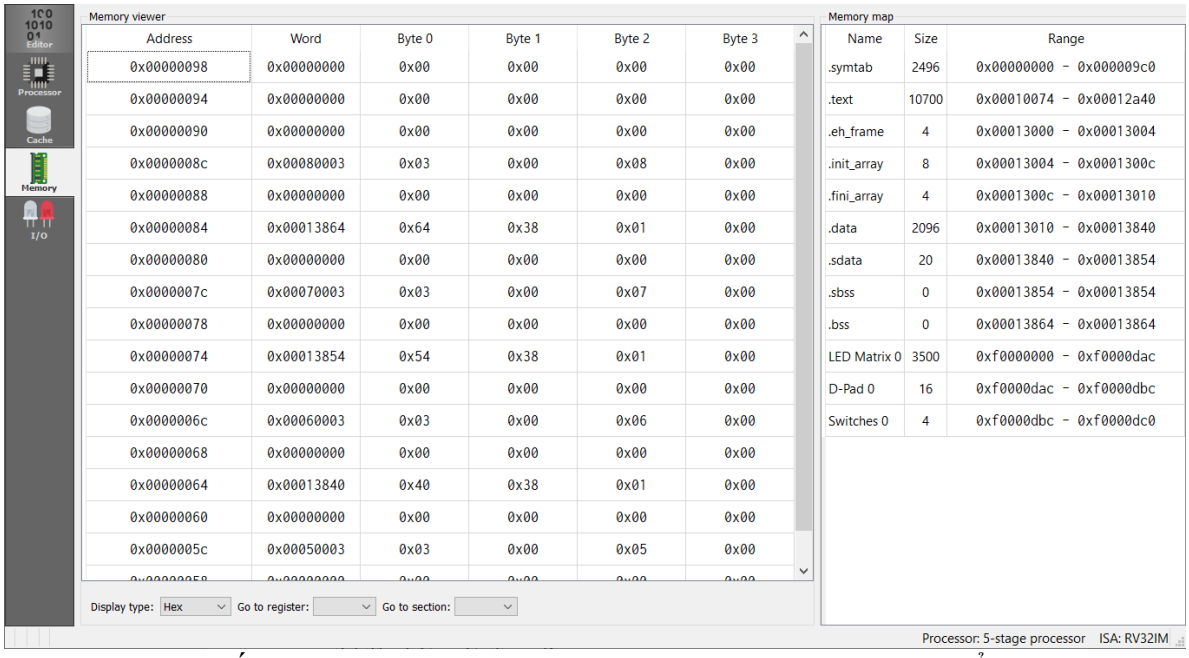
2.2.2. Tab Processer



Tab Processer 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.

2.2.3. Tab Memory

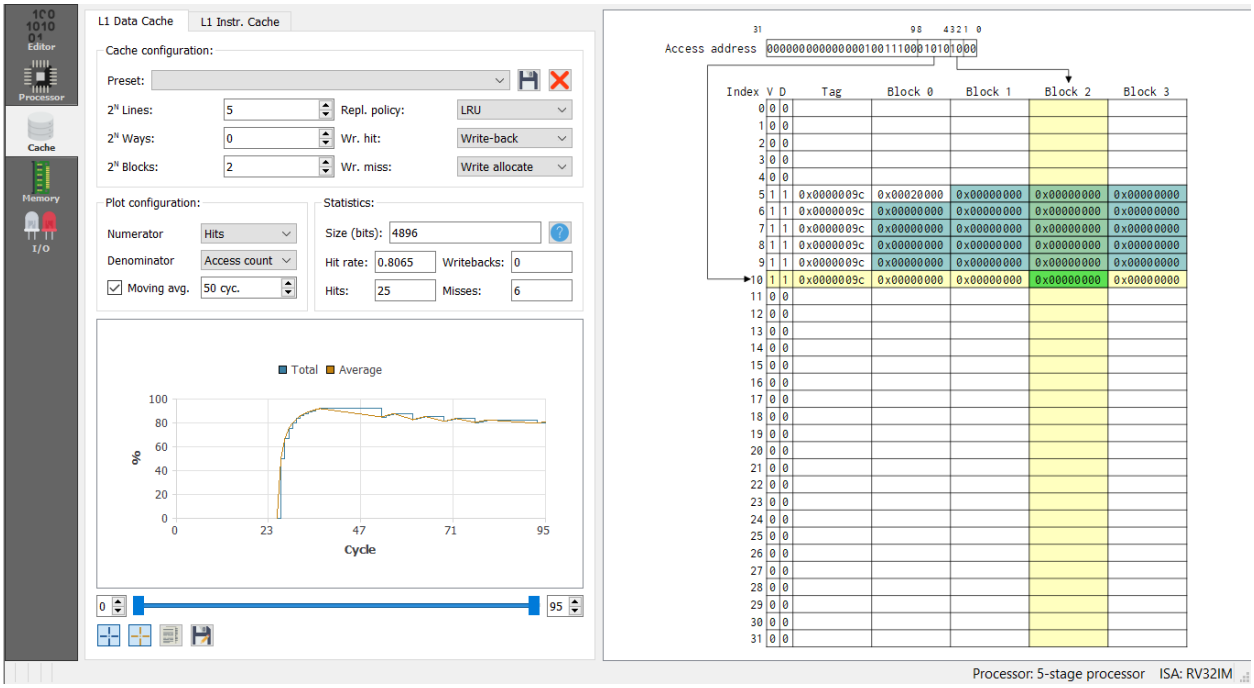


Tab bộ nhớ cung cấp một cái nhìn vào toàn bộ không gian địa chỉ có thể xác định của bộ xử lý. Điều hướng bộ nhớ có thể được thực hiện như sau

- 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.

2.2.4. Tab Cache

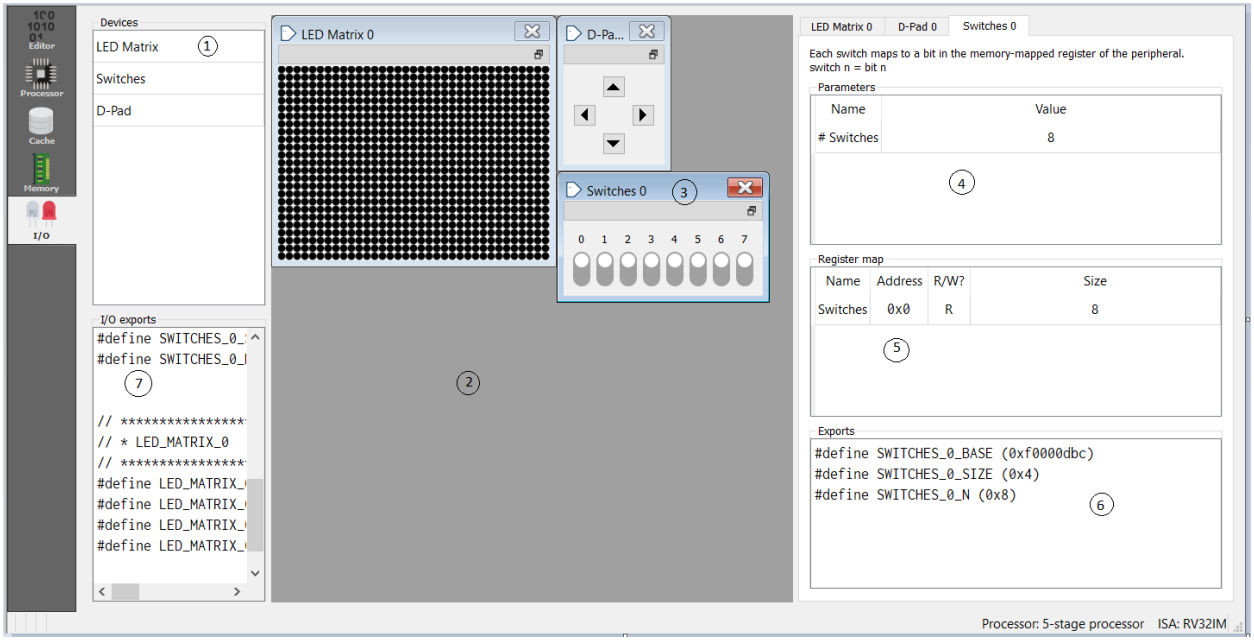


Kể từ phiên bản 2.1.0, Ripes bao gồm mô phỏng bộ nhớ cache. Trình mô phỏng bộ nhớ cache mô phỏng bộ nhớ đệm L1D (dữ liệu) và L1I (lệnh), trong đó có thể định cấu hình bố trí và hành vi của từng loại bộ nhớ cache. Do đó, chúng ta có thể phân tích hiệu suất bộ nhớ cache của các chương trình để xem các 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:
    - 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
    - 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.

o Các dòng bộ nhớ cache bẩn (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.

2.2.5. 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

2.3. Thanh Công Cụ Trong 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 tất 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 ecallchức 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 ecallđã 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.

Chương III. 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.

3.1. 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 = rv32imtắc 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:

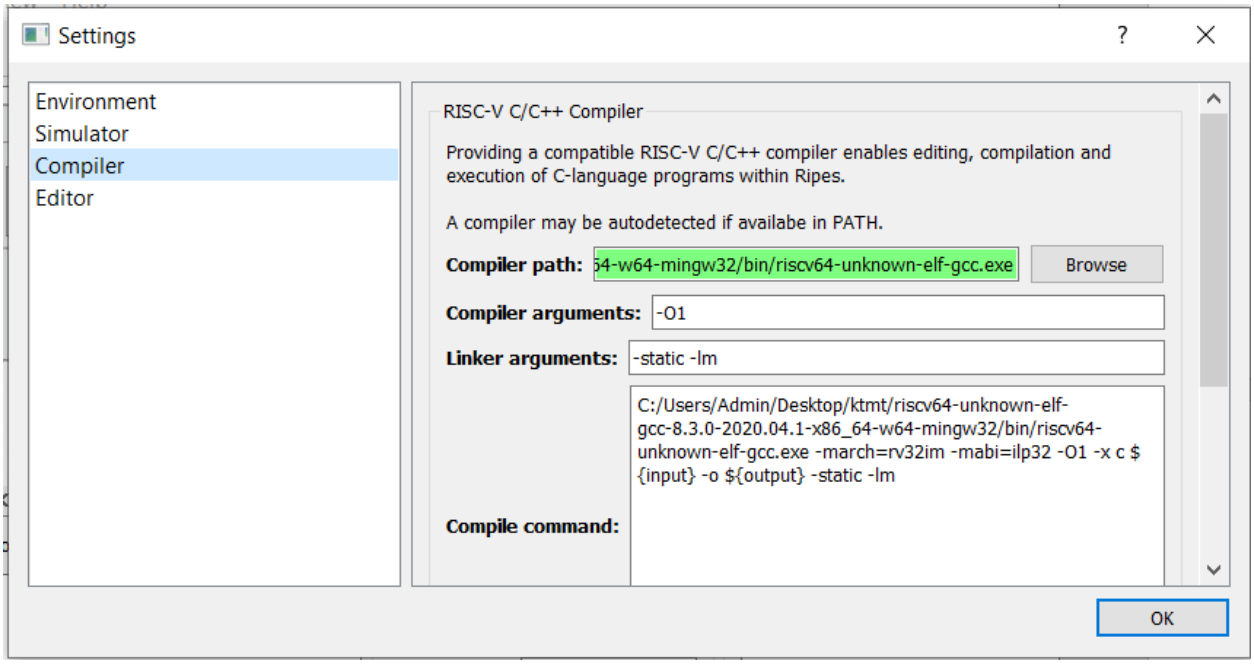
- Tải xuống chuỗi công cụ dựng sẵn cho nền tảng của bạn từ sifive/ freedom-tools. Phiên bản này được cho là hoạt động tốt với Ripes.
- Hoặc, nếu bạn muốn xây dựng chuỗi công cụ của riêng mình, hãy làm theo hướng dẫn https//github.com/riscv/riscv-gnu-toolchain để xây dựng chuỗi công cụ newlib.

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

3.2. 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

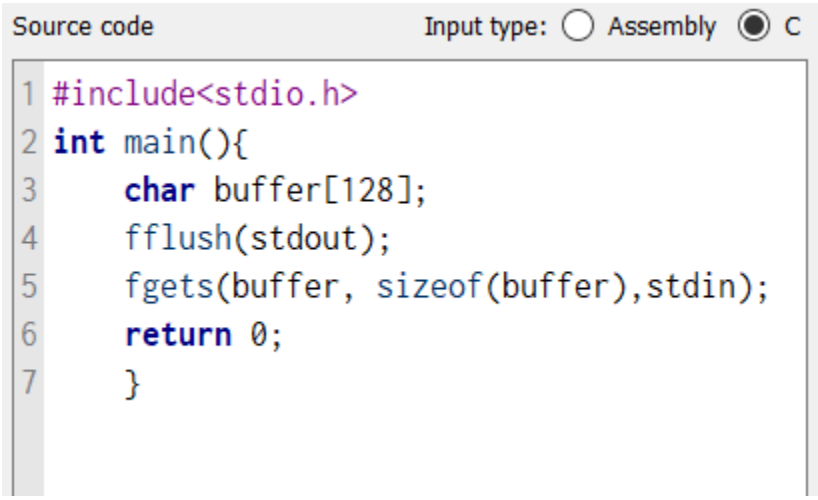
đã được chỉ định.



Hình 7: Toolchain Registration

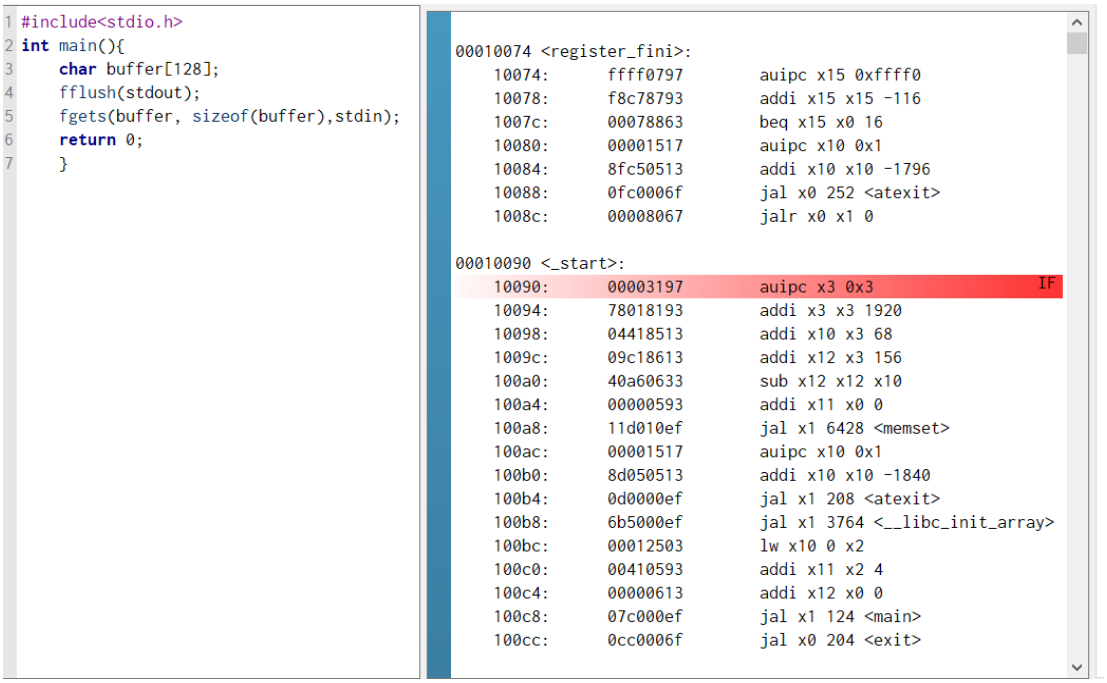
3.3. Compiling and Executing a C program

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)



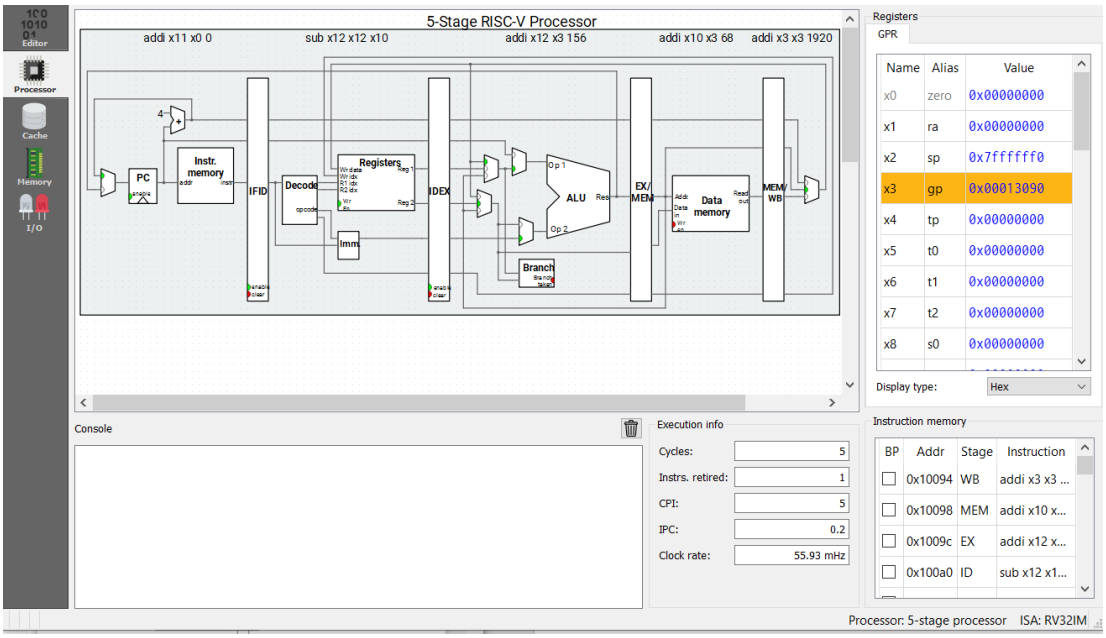
Hình 8: Một chương trình C

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:



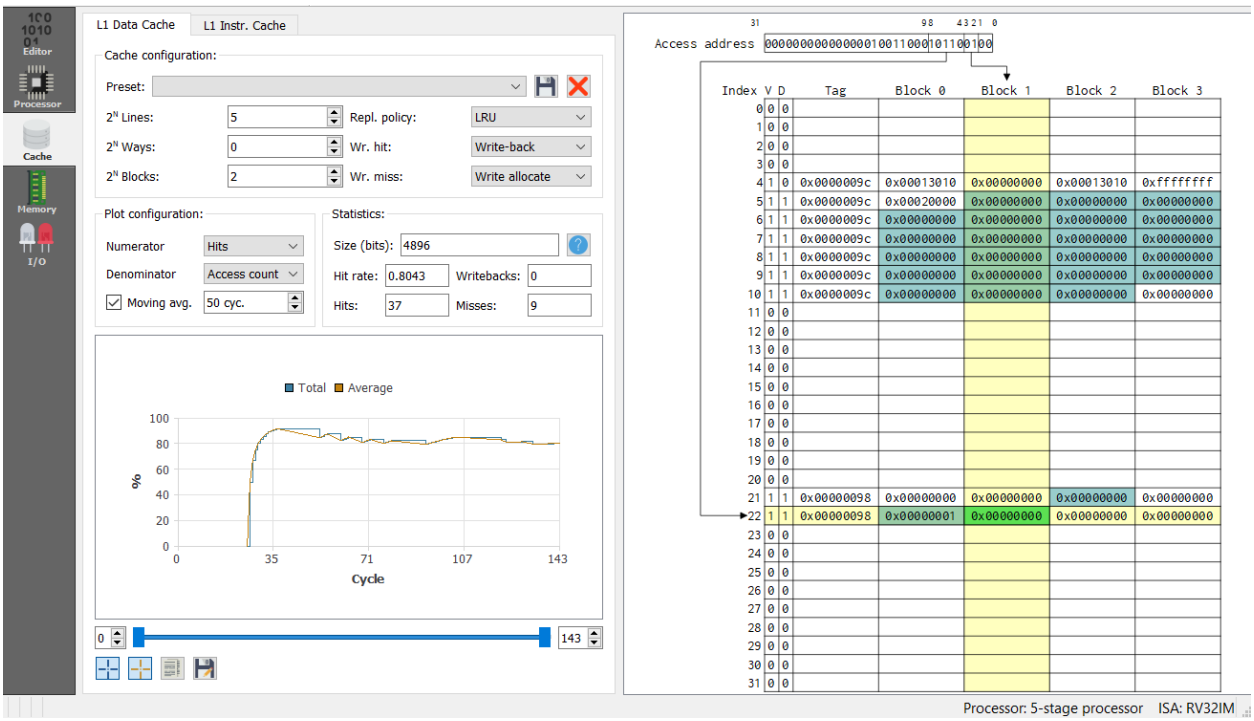
Hình 9: Mô phỏng,thực thi chương trình C

Chuyển sang 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.

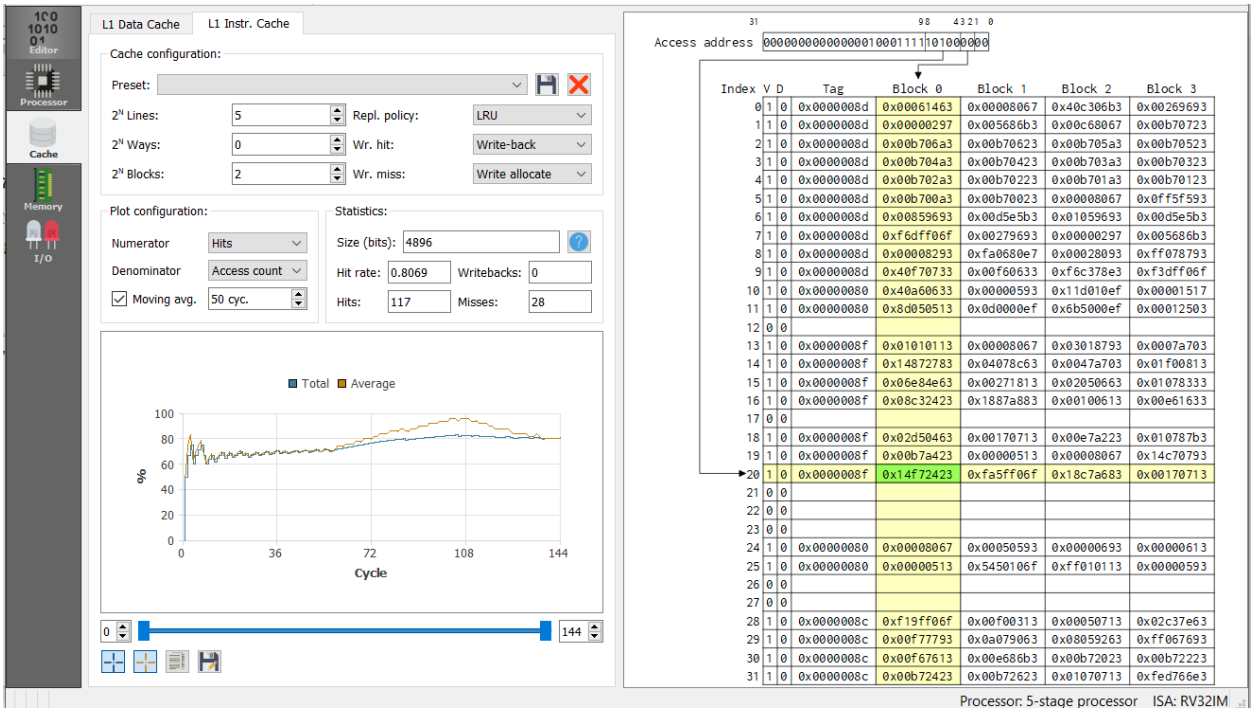


Hình 10: Mô phỏng đường dữ liệu RISC-V

Chuyển Tab Cache, Phần này sẽ phân tích hiệu suất bộ nhớ cache của các chương trình:

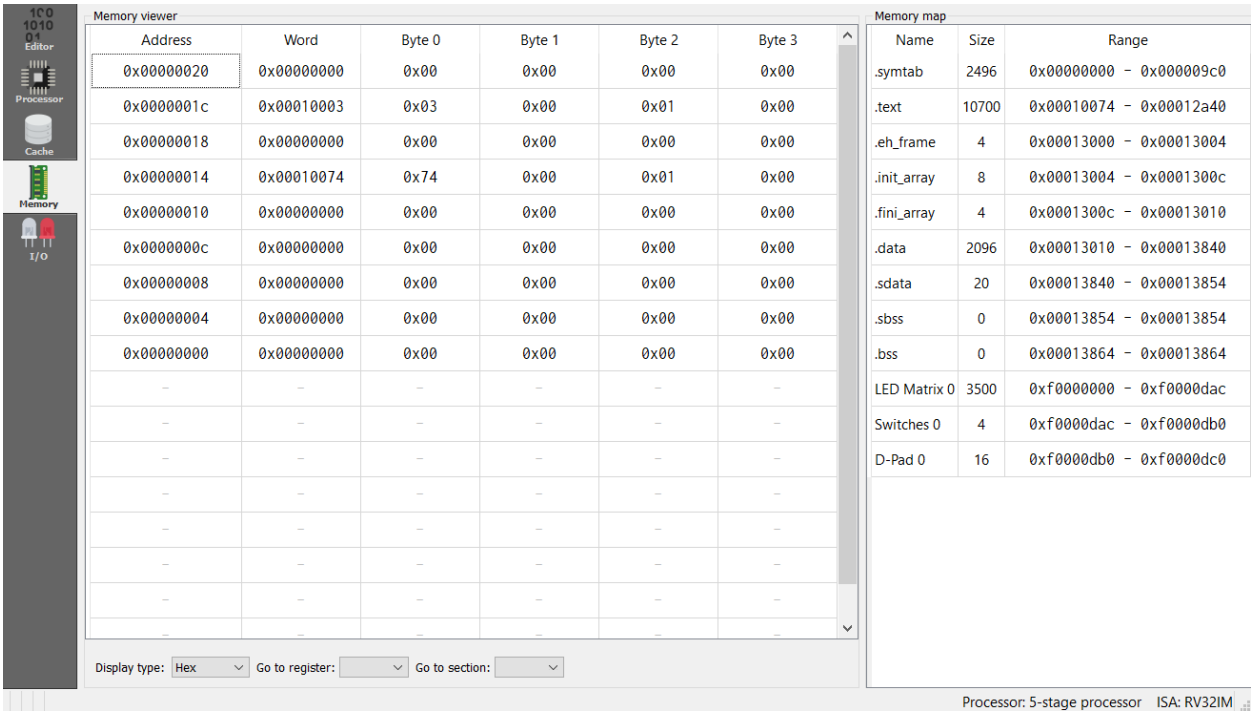


Hình 11: Mô phỏng bộ dữ liệu đệm



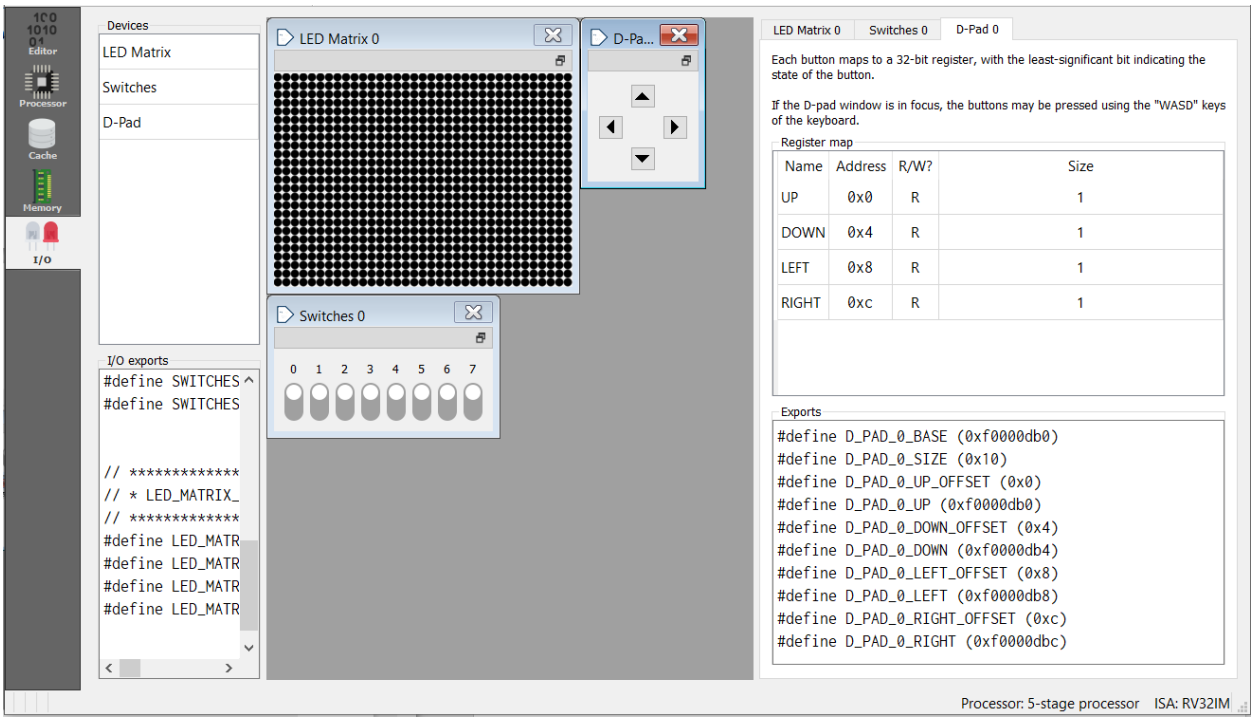
Hình 12: Mô phỏng bộ dữ liệu lệnh

Chuyển Tab Memory:



Hình 13: Tab Memory

Chuyển Tab I/O:



Hình 14: Tab I/O

Chương IV. Kết Luận

Sau một thời gian tìm hiểu và được sự hướng dẫn tận tình của cô Tạ Thị Kim Huệ, chúng em đã hoàn thành bài tập lớn với đề tài "Mô phỏng đường dữ liệ bộ xử lý RISC-V cho chương trình C bằng Ripes". Với đề tài này chúng em đã hiểu hơn về bộ xử lý RISC-V, đường dữ liệu và bộ bộ cache. Trong thời gian thực hiện chúng em đã thu được những kết quả hỗ trợ cho các đề tài nghiên cứu tiếp theo.

Tuy nhiên với thời gian cho phép cũng như kiến thức còn hạn chế sản phẩm vẫn còn những thiếu sót và chưa đạt được kết quả như mong muốn. Một lần nữa chúng em xin chân thành cảm ơn cô Tạ Thị Kim Huệ đã nhiệt tình hướng dẫn, truyền đạt kiến thức trong suốt quá trình học tập và thực hiện đề tài này.

Tài Liệu Tham Khảo

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

[2]. https://github.com/mortbopet/Ripes/wiki/Building-and-Executing-C-programs-with-Ripes.

[3]. https://github.com/mortbopet/Ripes/wiki/Ripes-Introduction.

Link source code: https://github.com/Huster2017/Project_ktmt_hust