

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

**TÀI LIỆU HƯỚNG DẪN THÍ NGHIỆM**  
**KIẾN TRÚC MÁY TÍNH**

**Họ và tên:** .....  
**MSSV:** .....  
**Lớp - khóa:** ĐTVT ... – K...

Hà Nội, 2021

# MỤC LỤC

<b>GIỚI THIỆU .....</b>	<b>4</b>
<b>THÔNG TIN CHUNG VỀ CÁC BÀI THÍ NGHIỆM .....</b>	<b>5</b>
<b>BÀI 1. THỰC HÀNH CƠ BẢN VỀ HỢP NGỮ VÀ KIẾN TRÚC TẬP LỆNH RISC V .....</b>	<b>6</b>
<b>MỤC TIÊU.....</b>	<b>6</b>
<b>CƠ SỞ LÝ THUYẾT CHUNG VÀ CÂU HỎI KIỂM TRA .....</b>	<b>6</b>
<b>Installation .....</b>	<b>6</b>
<b>Getting Started .....</b>	<b>7</b>
<b>UI and launch parameters (Giao diện và các tham số thực thi) .....</b>	<b>7</b>
View memory.....	9
Accessing registers.....	10
<b>LED Matrix.....</b>	<b>11</b>
Ecall 0x100: thiết lập cho từng Pixel .....	11
Ecall 0x101: thiết lập cho toàn bộ điểm Pixels .....	11
Using the LED matrix view .....	12
<b>Toy Robot LED Matrix.....</b>	<b>12</b>
Ecall 0x110: Set LED row .....	13
Using the LED matrix view .....	13
<b>Seven Segment Board.....</b>	<b>14</b>
Ecall 0x120: Set seven segment display .....	14
Ecall 0x121: Set LEDs.....	15
Ecall 0x122: Read push buttons.....	15
Using the board view .....	16
Terminal: Ecall 0x130 and 0x131 .....	16
Example assignment .....	17
<b>BÀI 2 : HƯỚNG DẪN CÀI ĐẶT RIPE MÔ PHỎNG ĐƯỜNG DỮ LIỆU BỘ XỬ LÝ RISC V .....</b>	<b>18</b>
<b>MỤC TIÊU .....</b>	<b>18</b>
<b>CƠ SỞ LÝ THUYẾT VÀ CÂU HỎI KIỂM TRA .....</b>	<b>18</b>
<b>The Editor Tab .....</b>	<b>21</b>
<b>The Processor Tab.....</b>	<b>23</b>
<b>Tổng quan về bộ xử lí.....</b>	<b>24</b>

<b>Điều khiển trình mô phỏng.....</b>	<b>26</b>
<b>Chọn bộ xử lí .....</b>	<b>27</b>
<b>Tab Memory .....</b>	<b>29</b>
<b>Tab cache .....</b>	<b>30</b>
<b>MÔ PHỎNG BỘ NHỚ CACHE .....</b>	<b>30</b>
<b>Bộ nhớ cache.....</b>	<b>32</b>
Cấu hình cho cache .....	32
The Cache View.....	32
Cache Access Statistics & Plotting .....	35
<b>Example.....</b>	<b>36</b>
Chương trình ví dụ:.....	36
Simulating Different Cache Configurations .....	37

# GIỚI THIỆU

Tài liệu hướng dẫn thí nghiệm này trình bày chi tiết các khái niệm cơ bản, trao đổi, thực hành, bài tập, câu hỏi ôn tập và các bước tiến hành thí nghiệm thông qua việc mô hình hóa bộ xử lý RISC V trên phần mềm mô phỏng mã nguồn mở VENUS và RIPES, từ đó hình thành tư duy thiết kế cấu trúc bộ xử lý và thực hành với ngôn ngữ lập trình phần cứng để có thể thiết lập một bộ xử lý RISC V đơn giản trên FPGA.

Các bài thí nghiệm cung cấp cho sinh viên kiến thức và kỹ năng vững chắc về tư duy thiết kế bộ xử lý RISC V theo kiến trúc máy tính có tập lệnh giản lược. Nắm được các phân lớp trừu tượng được phân chia trong kiến trúc máy tính, từ đó hiểu được quá trình biên dịch và thực hiện lệnh điều khiển trong máy tính như thế nào, quá trình hình thành mã máy để trở thành tín hiệu điều khiển trong các khối logic hình thành Datapath trong máy tính. Hiểu được tư duy thiết kế và cách hoạt động của bộ xử lý RISC V đơn xung nhịp, đa xung nhịp và xử lý đường ống. Các tín hiệu điều khiển hoạt động như thế nào trong bộ xử lý ở mức vi kiến trúc. ....

Sinh viên cần chấp hành nghiêm túc các quy định về an toàn điện, nội quy phòng thí nghiệm và hướng dẫn của cán bộ phụ trách trong suốt quá trình làm thí nghiệm tại phòng Lab. Sinh viên được yêu cầu xem lại và nắm vững phần lý thuyết cơ bản và hoàn thành các câu hỏi kiểm tra trước khi thực hiện thí nghiệm; thực hiện đầy đủ và tuân thủ các bước tiến hành thí nghiệm theo hướng dẫn, ghi lại đầy đủ các kết quả thực nghiệm; trả lời đầy đủ các câu hỏi ôn tập sau khi làm thực nghiệm.

## THÔNG TIN CHUNG VỀ CÁC BÀI THÍ NGHIỆM

- **Tên học phần:** Kiến trúc máy tính
- **Mã học phần:** ET4040
- **Cấu trúc học phần:** 2(2-0-1-4)
- **Khối lượng thí nghiệm:** 1 tín chỉ (thời lượng 15 tiết/học kỳ)
- **Số bài thí nghiệm:** 3 bài

TT	Nội dung	Chuẩn đầu ra HP	Bài đánh giá	Thời lượng	Địa điểm
Bài 1	<b>Khởi tạo môi trường thực hành hợp ngữ RISC V:</b> <ul style="list-style-type: none"><li>- Luyện tập các tập lệnh hợp ngữ RISC V</li><li>- Cài đặt một số chương trình đơn giản, hiểu quá trình biên dịch lệnh máy</li><li>- Sử dụng bộ mô phỏng Venus (JavaScript in Browser)</li></ul>	M1, M2, M5, M6	A3	5 tiết	
Bài 2	<b>Hướng dẫn cài đặt RIPES mô phỏng cấu trúc đường dữ liệu RISC V</b> <ul style="list-style-type: none"><li>-</li></ul>	M1, M2, M5, M6	A3	5 tiết	
Bài 3	<b>Thiết kế bộ xử lý RISC V đơn giản bằng ngôn ngữ lập trình phần cứng và thực hiện trên FPGA</b> <ul style="list-style-type: none"><li>-</li></ul>	M1, M2, M5, M6	A3	5 tiết	

### Chú ý:

- Nội dung từng bài thí nghiệm phải đóng góp vào chuẩn đầu ra tương ứng của cả học phần
- Chuẩn đầu ra, bài đánh giá được định nghĩa và mô tả chi tiết trong Đề cương học phần đã xây dựng (như ở PHỤ LỤC I)

# BÀI 1. THỰC HÀNH CƠ BẢN VỀ HỢP NGỮ VÀ KIẾN TRÚC TẬP LỆNH RISC V

## MỤC TIÊU

Bộ mô phỏng RISC V Venus được tích hợp trong VS code

Đây là một bộ mô phỏng thông dụng Venus RISC V simulator. Cung cấp môi trường thực hành độc lập và không cần các công cụ hỗ trợ khác. Có thể chạy mã hợp ngữ RISC V với khả năng debugging bởi VS code.

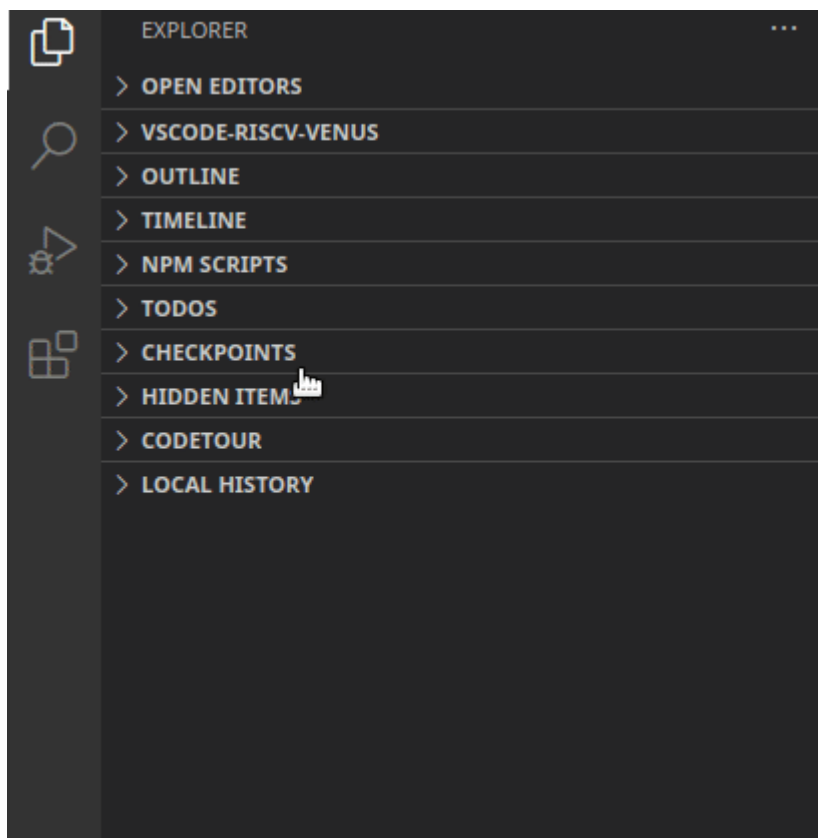
## CƠ SỞ LÝ THUYẾT CHUNG VÀ CÂU HỎI KIỂM TRA

### INSTALLATION

Cài đặt Visual Studio Code

<https://code.visualstudio.com/download>

Mở chức năng extension: marketplace tìm kiếm từ khóa "riscv":

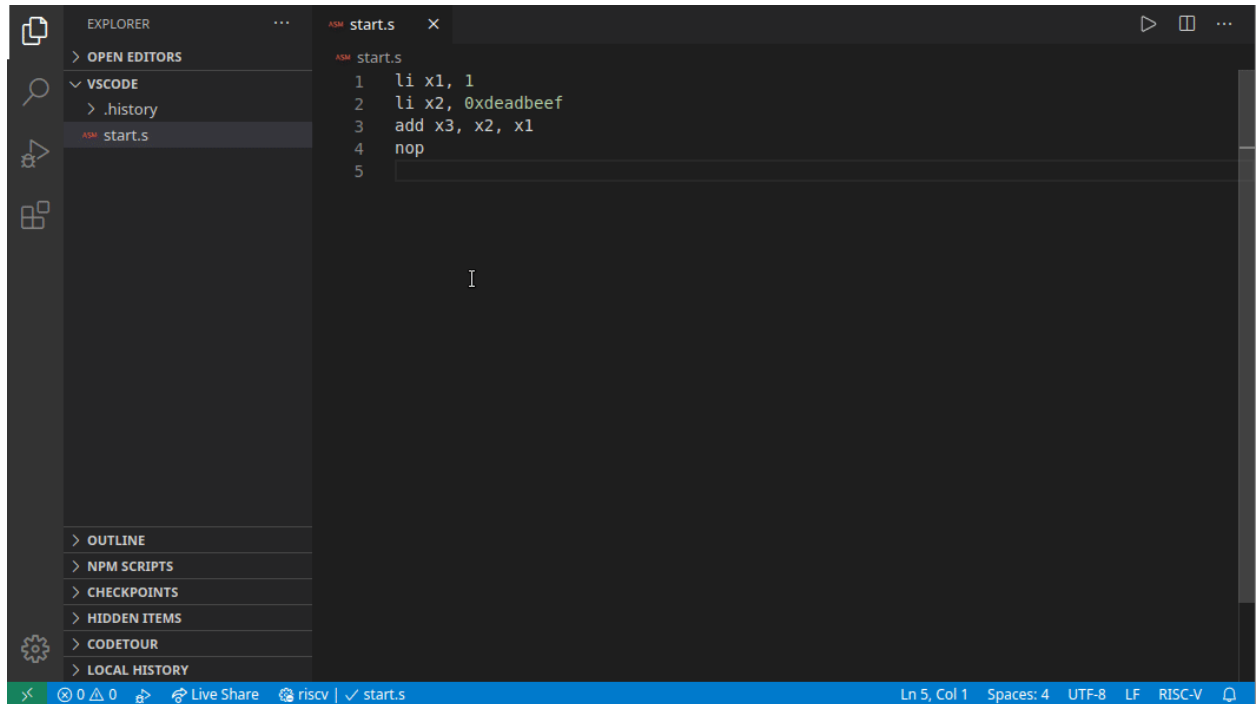


Hoặc mở bằng phím tắt qua VS Code Quick Open (CTRL+P):  
ext install hm.riscv-venus

## GETTING STARTED

Bạn có thể debug bất kỳ file hợp ngữ nào bạn đang chỉnh sửa. Nên sử dụng [RISC-V Support](#) extension tô sáng cú pháp (syntax highlighting).

Bắt đầu chạy Debugging với chức năng "Run and Debug" để debug file hiện tại :

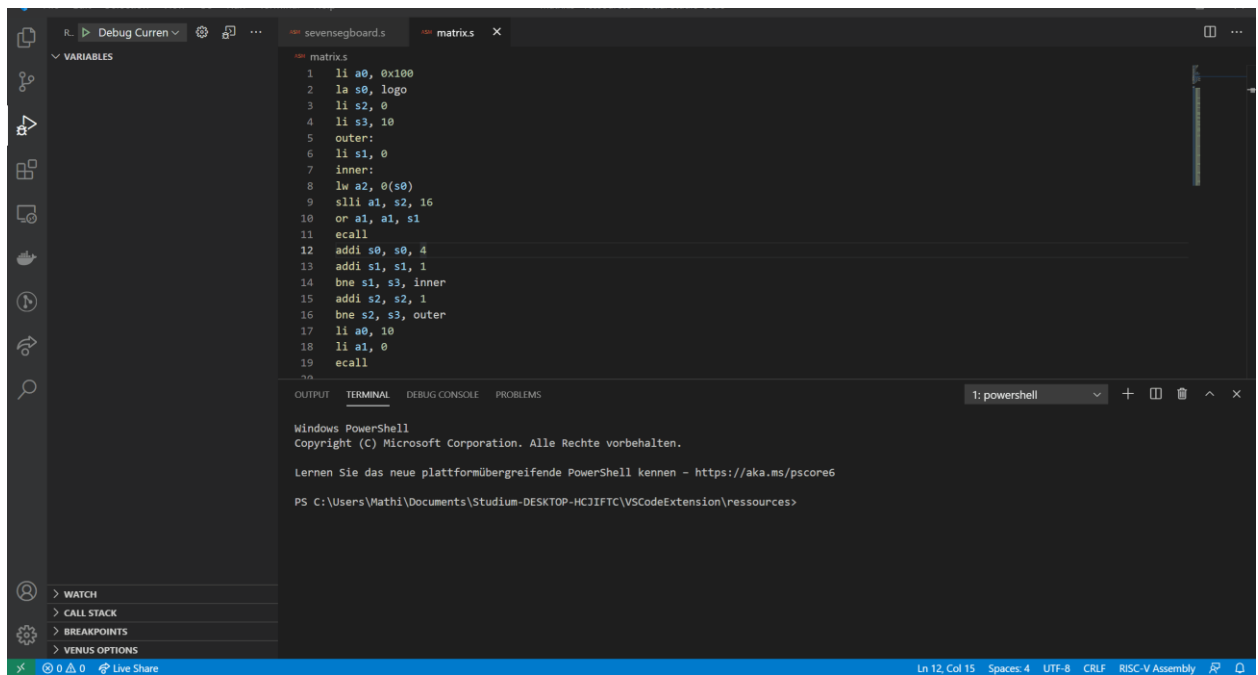


Bạn có thể chạy từng đoạn code và từng giả lệnh toán tử một cách tự động, các lệnh thực (actual instructions) sẽ được hiển thị trong cửa sổ assembly view (được mở tự động nếu nó được thiết lập trong cài đặt mở rộng). Quá trình debug hỗ trợ cả việc tạo các điểm debug thông qua thiết lập breakpoints.

Các lệnh gọi biến môi trường cơ bản cho Venus ([Venus environmental calls](#)) cũng được hỗ trợ.

## UI AND LAUNCH PARAMETERS (GIAO DIỆN VÀ CÁC THAM SỐ THỰC THI)

Từ biểu tượng Debugger, bạn có thể mở đến “Venus options”. Từ đây có thể cài đặt, xem các tài liệu hướng dẫn. Đây là lệnh để có thể thực hiện những thao tác này: CTRL + P và gõ "Venus: ..."

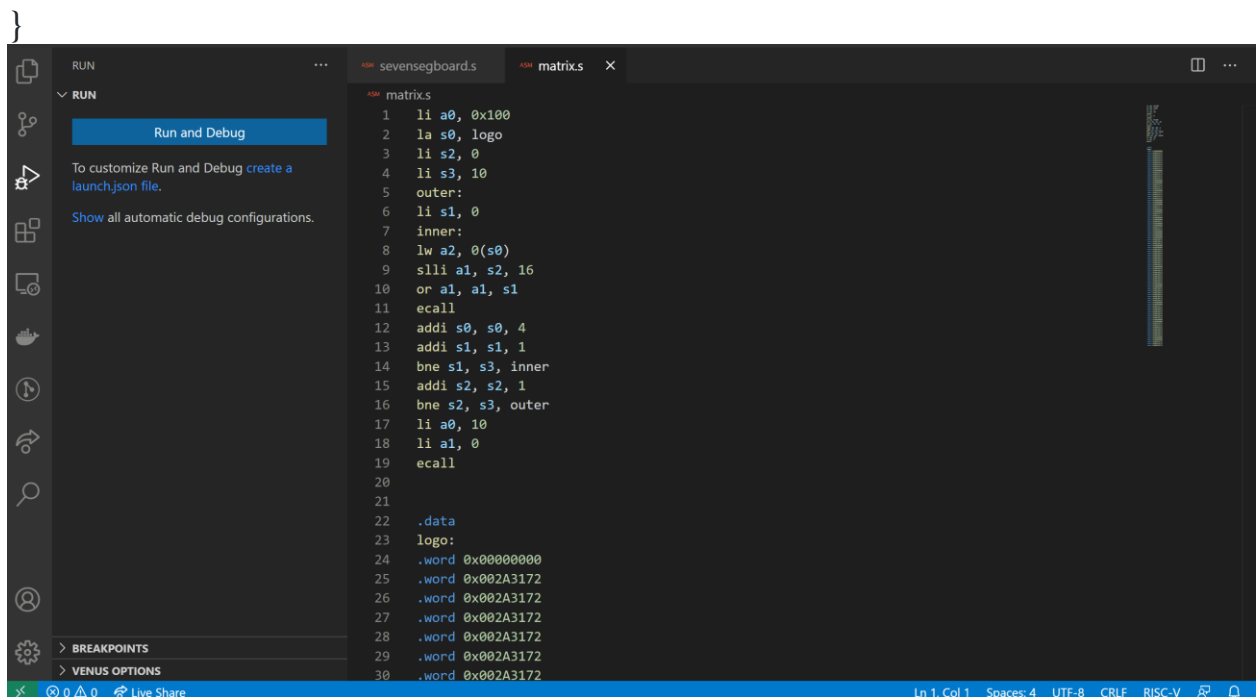


Đây là Support for the VSCode-Inherent [launch.json](#). Để tìm kiếm có thể gỡ file cấu hình này "Venus launch.json".

Một file launch.json đầy đủ sẽ có cấu trúc như thế này

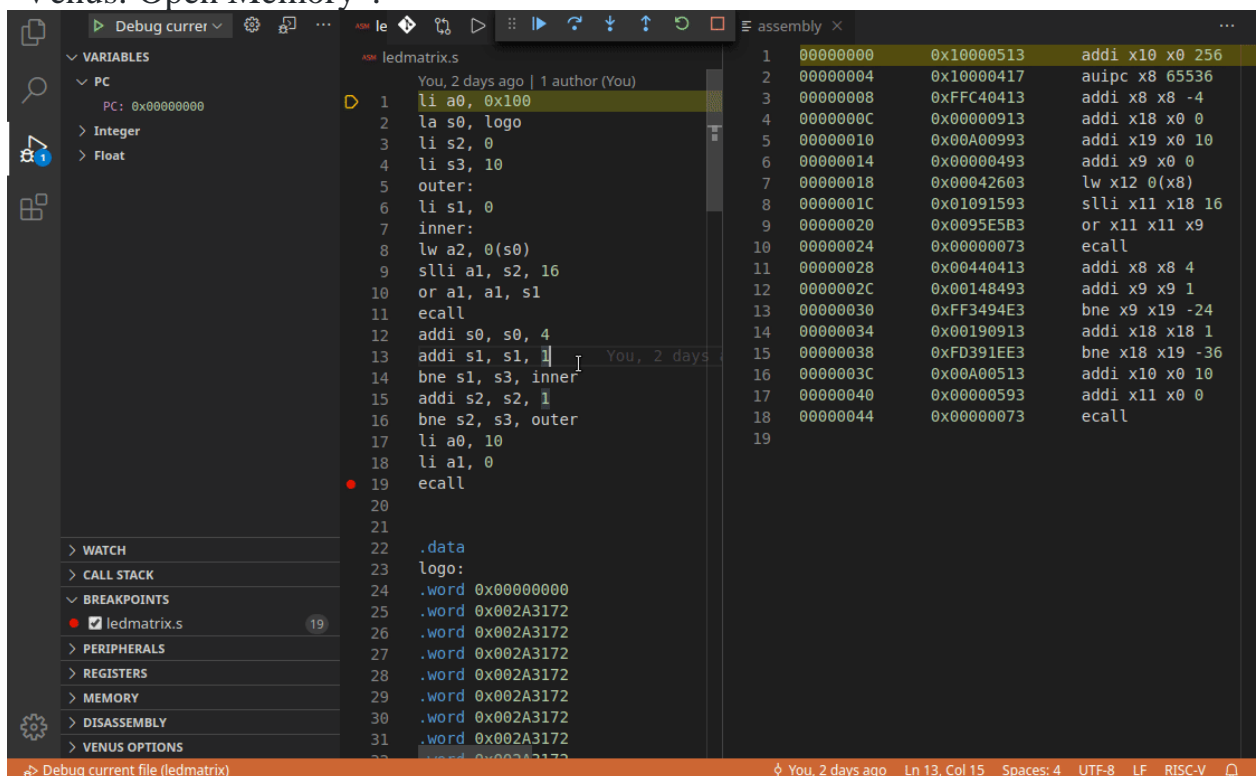
```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "venus",
      "request": "launch",
      "name": "Launch current file with all options",
      "program": "${file}",
      "stopOnEntry": true,
      "stopAtBreakpoints": true,
      "openViews": [
        "Robot",
        "LED Matrix",
        "Seven Segment Board"
      ],
      "ledMatrixSize": {
        "x": 10,
        "y": 10
      }
    }
  ]
}
```





## View memory

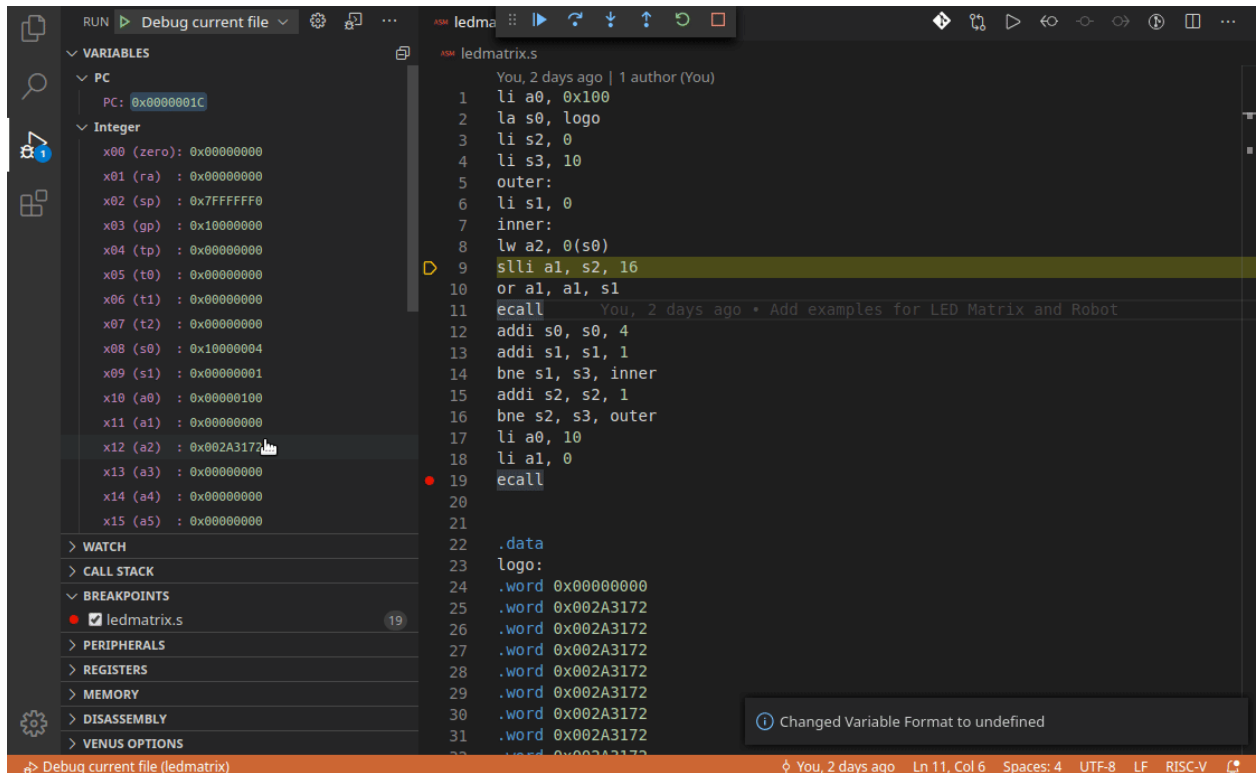
Bạn có thể xem nội dung trong bộ nhớ bằng cách mở với thao tác CTRL+P and "Venus: Open Memory".



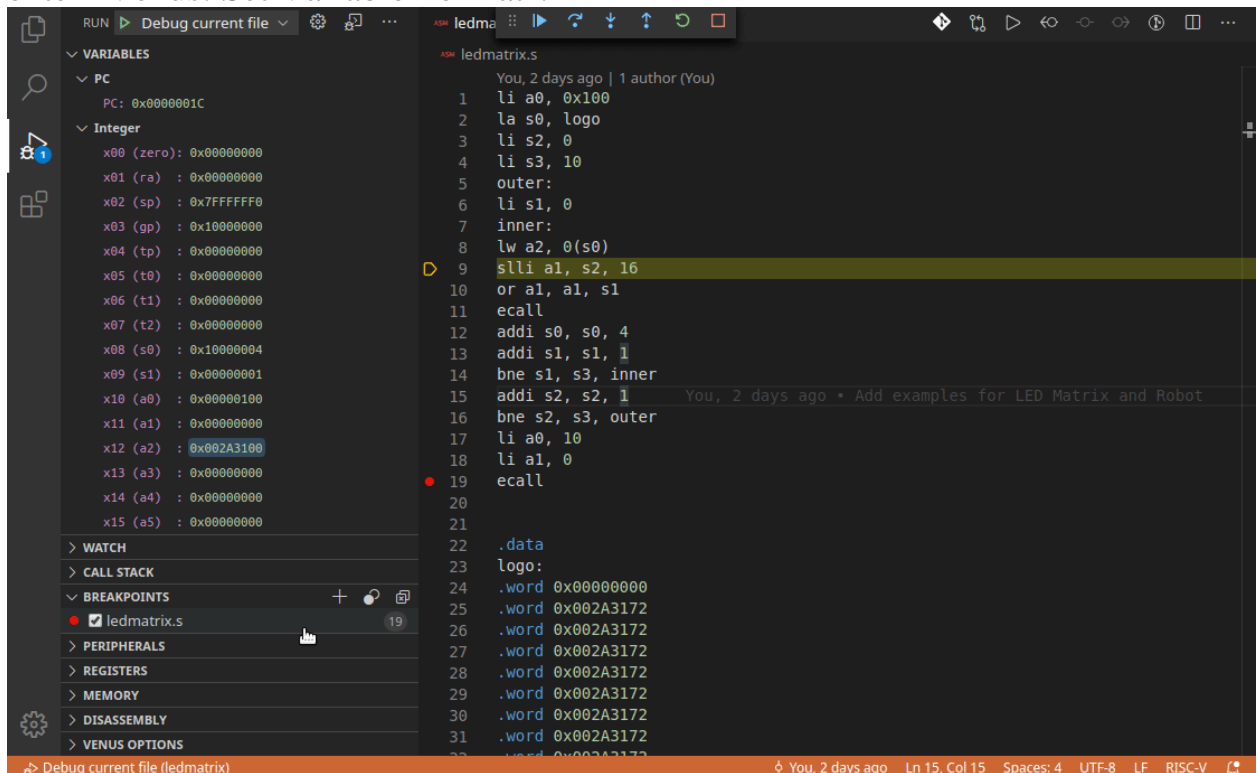
Có thể lựa chọn các đoạn code để xem và cuộn xem xuyên suốt các từ nhớ (có thể thực hiện bằng phiên bản Venus online)

## Accessing registers

Bạn có thể thay đổi giá trị các thanh ghi bằng cách click vào thanh nhập giá trị và cập nhật giá trị mới rồi ấn Enter

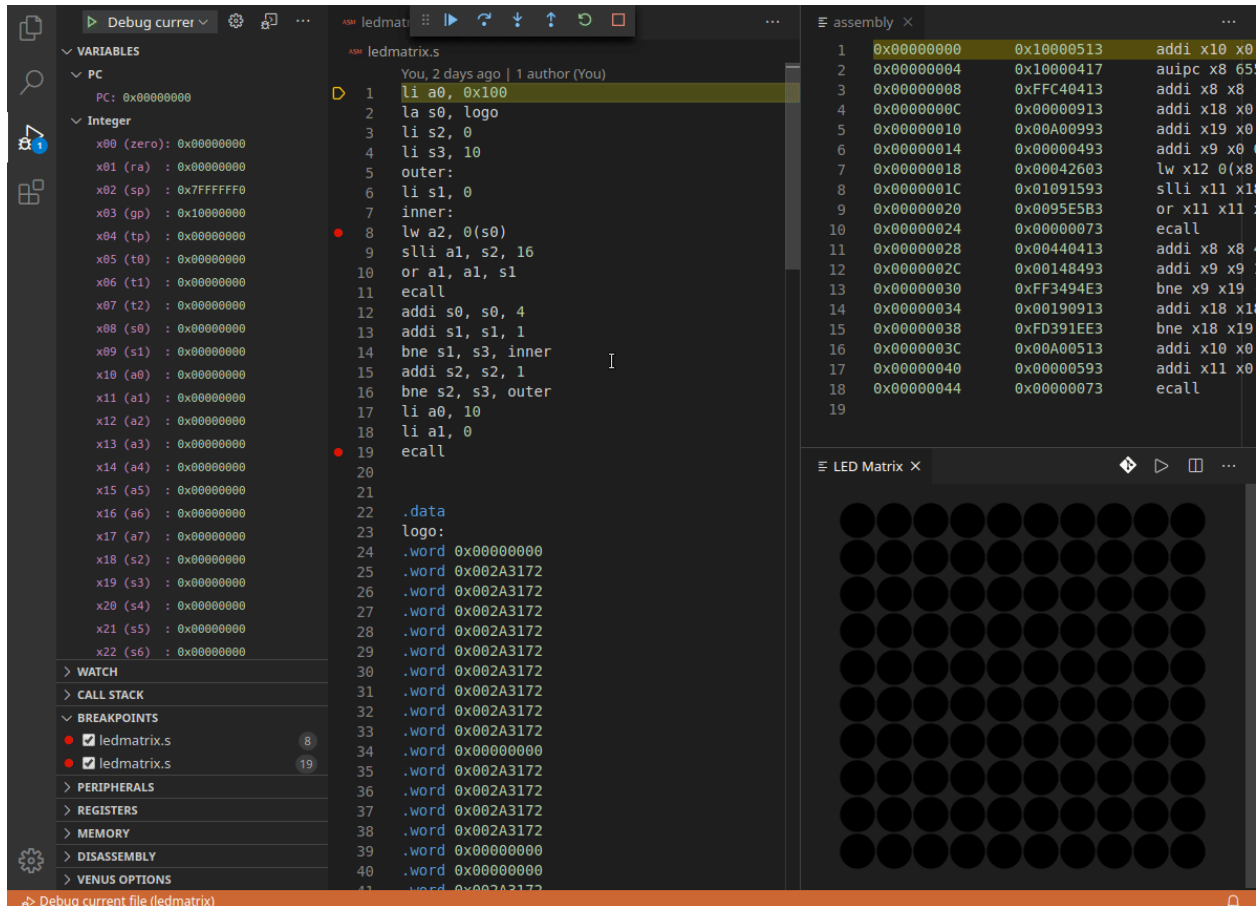


Để thay đổi định dạng thanh ghi, click on "Venus Options" or press CTRL+P and enter "Venus: Set Variable Format".



# LED MATRIX

The LED matrix (ma trận LED) đang đề mặc định kích thước ma trận LED RGB là 10x10 RGB LED, có thể thay đổi kích thước bằng cách thay đổi tham số trong file launch.json. Mỗi đèn LED được cài đặt độc lập bằng hàm gọi môi trường environment call 0x100.



## Ecall 0x100: thiết lập cho từng Pixel

Các thiết lập về vị trí pixel theo tọa độ (x,y) và màu RGB color (r, g, b) :

- Thanh ghi a1 chứa giá trị x trong các bit 31-16 và giá trị y trong các bit 15-0`
- Thanh ghi a2 chứa giá trị r trong các bit 23-16, g trong các bit 15-8 và b trong các bit 7-0

Ví dụ thiết lập pixel màu đỏ tại vị trí (2,4):

```
li a0, 0x100
li a1, 0x00020004
li a2, 0x00FF0000
ecall
```

## Ecall 0x101: thiết lập cho toàn bộ điểm Pixels

Đây là thiết lập cho toàn bộ điểm ảnh pixels theo màu RGB color (r, g, b) :

- Thanh ghi a1 chứa giá trị r trong các bit 23-16, giá trị g trong các bit 15-8 và giá trị b trong các bit 7-0

Ví dụ cho việc thiết lập toàn bộ điểm ảnh màu đỏ như sau:

```
li a0, 0x101
li a1, 0x00FF0000
ecall
```

## Using the LED matrix view

Bạn có thể bằng xem (view) bằng thao tác CTRL+P và sau đó select/enter "Venus: Open LED Matrix".

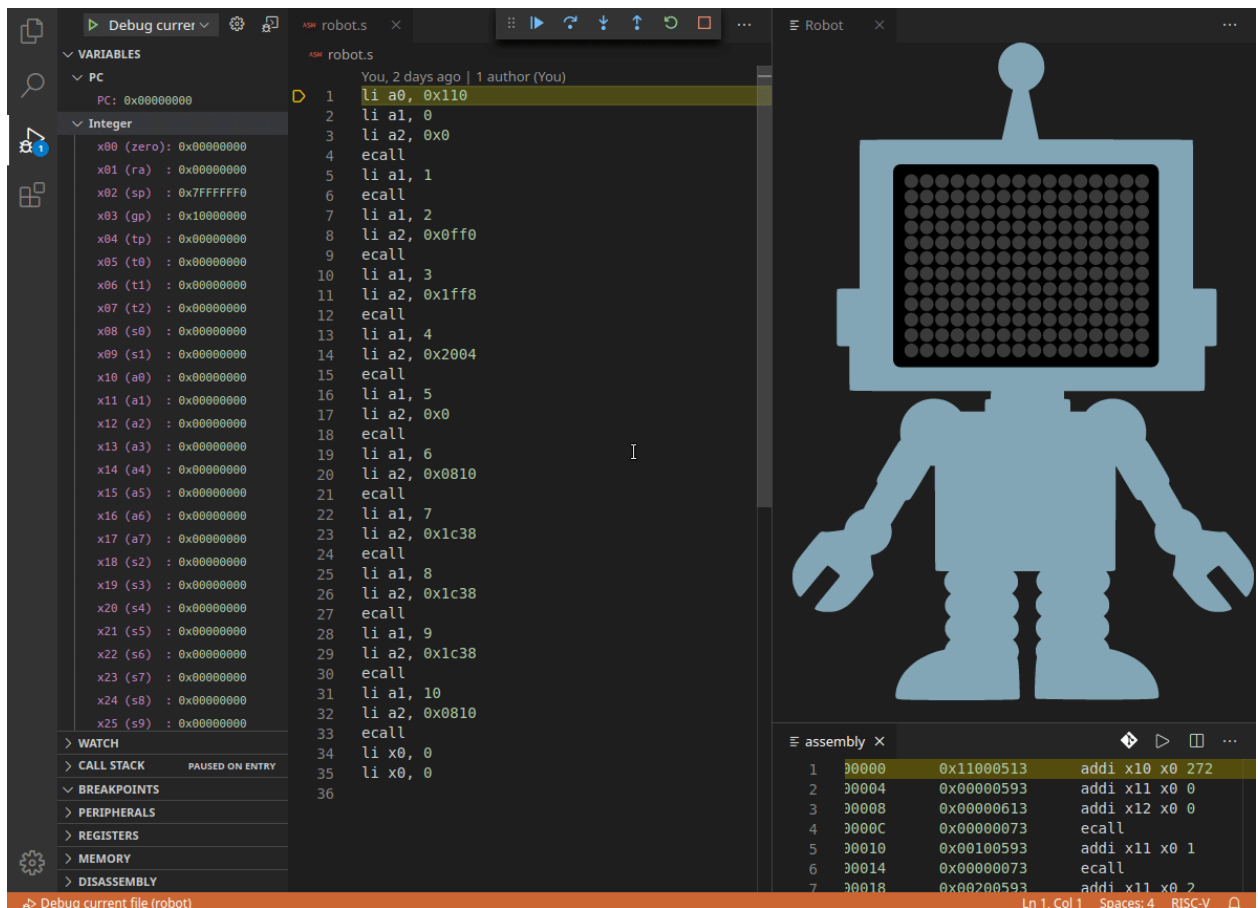
Có thể mở tự động bằng việc thêm vào file `.vscode/launch.json` trong project:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "venus",
      "request": "launch",
      "name": "Debug current file",
      "program": "${file}",
      "stopOnEntry": true,
      "openViews": [ "LED Matrix" ]
    }
  ]
}
```

Có thể tìm ví dụ để vẽ logo RISC V ở đây [here](#).

## TOY ROBOT LED MATRIX

Ví dụ về toy robot là một biến thể của ma trận LED, nhưng với một hình thức khác và cách thức thiết lập ma trận LED theo một cách khác: Ma trận LED tất cả đều là màu xanh và thiết lập bởi một mặt nạ bit đơn lẻ trên một dòng. Có 12 dòng và mỗi dòng có 16 đèn LEDs.



## Ecall 0x110: Set LED row

Cập nhật trên từng dòng đơn lẻ

- Thanh ghi a1 thiết lập dòng, khi đó 0 sẽ là dòng ở cuối cùng
- Thanh ghi a2 chứa giá trị để thiết lập trạng thái LEDs on/off (1/0) trong các bit 15-0 trong khi đó bit thứ 15 là đèn LED trái nhất

## Using the LED matrix view

Bạn có thể mở bảng View bằng cách ấn tổ hợp CTRL+P sau đó select/enter "Venus: Open LED Matrix".

Hoặc có thể tự động đưa vào file `.vscode/launch.json` trong project của bạn:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "venus",
      "request": "launch",
      "name": "Debug current file",
      "program": "${file}",
      "stopOnEntry": true,
      "openViews": [ "Robot" ]
    }
  ]
}
```

```

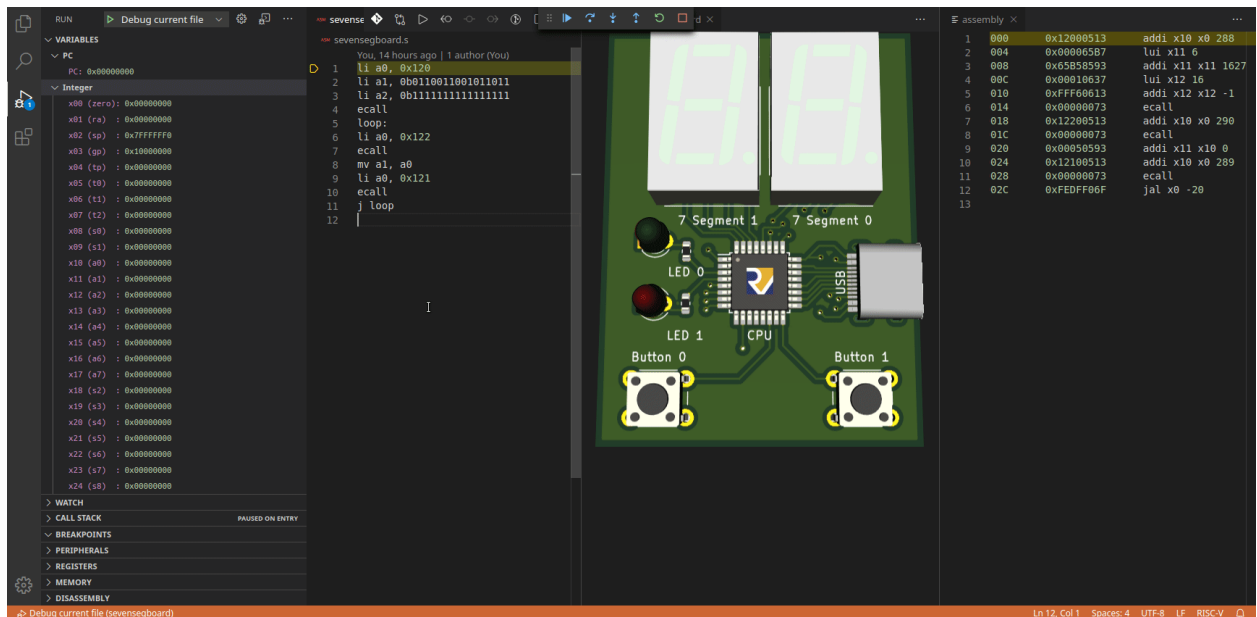
    }
  ]
}

```

Xem ví dụ về vẽ RISC-V logo ở đây [here](#).

## SEVEN SEGMENT BOARD

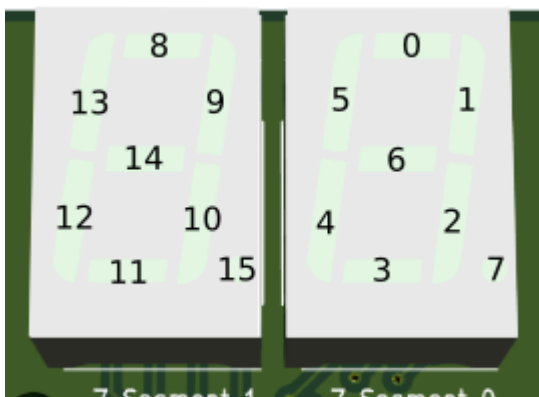
Màn hình LED 7 đoạn là một ví dụ phức tạp hơn. Có 2 nút bấm như là thiết bị đầu vào và 2 đèn LED 7 đoạn hiển thị như là đầu ra.



### Ecall 0x120: Set seven segment display

Đây là hàm gọi môi trường ecall cập nhật, các đoạn hiển thị có được điều khiển một cách độc lập. Hàm gọi môi trường kỳ vọng các giá trị và các mặt nạ hợp lệ cập nhật trong 2 thanh ghi.

Các đoạn hiển thị được ánh xạ theo vị trí bit trong hai thanh ghi (thanh ghi giá trị và thanh ghi mặt nạ) được thiết lập như sau:



Thanh ghi a1 chứa các giá trị và thanh ghi a2 chứa mặt nạ cập nhật giá trị mới. Xem đoạn code sau đây hiển thị giá trị 42 trong LED 7 đoạn:

```
li a0, 0x120
li a1, 0b0110011001011011
li a2, 0b1111111111111111
ecall
```

Cập nhật 2 đoạn hiển thị sau đây:

```
li a0, 0x120
li a1, 0b00000000000000100
li a2, 0b00000000000010100
ecall
```

Màn hình thanh ghi 7 đoạn sẽ hiển thị như thế nào?

### **Ecall 0x121: Set LEDs**

Hàm gọi môi trường thiết lập LEDs được định nghĩa trong 2 bit thấp nhất của thanh ghi a1. Bit 0 thiết lập LED 0 màu xanh và bit 1 thiết lập LED 1 màu đỏ.

Các LEDs ở trạng thái on tương ứng với bit 1 và off tương ứng với bit 0. Các đèn LED có thể chuyển trạng thái đến khi gọi lại ecall. Đoạn code sau đây tạo ra sự nhấp nháy của đèn LEDs như sau:

```
loop:
li a0, 0x121
li a1, 0b01
ecall
li a0, 0x121
li a1, 0b10
ecall
j loop
```

### **Ecall 0x122: Read push buttons**

Lệnh để đọc các nút ấn xem đã được nhấn hay chưa, khi các nút ấn được nhấn, chúng sẽ được sáng lên rõ ràng khi đó lệnh gọi môi trường sẽ gọi bộ đếm cho việc nhấn này. Hàm ecall kiểm tra 2 nút đã được nhấn hay chưa bằng 2 bit thấp nhất của a0 và làm mới lại bộ đếm.

Đoạn mã sau đây đọc để biết các nút đã được nhấn hay chưa và đặt đèn LED cho phù hợp.

```
loop:
li a0, 0x122
ecall
mv a1, a0
li a0, 0x121
ecall
j loop
```

## Sử dụng the board view

Bạn có thể sử dụng bảng view bằng việc ấn tổ hợp CTRL+P sau đó select/enter "Venus: Open Seven Segment Board UI". Có thể thay thế tổ hợp phím trên bằng cách tự động thêm vào bản ghi `.vscode/launch.json` trong project của bạn:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "venus",
      "request": "launch",
      "name": "Debug current file",
      "program": "${file}",
      "stopOnEntry": true,
      "openViews": [ "Seven Segment Board" ]
    }
  ]
}
```

Có thể xem ví dụ ở đây [here](#).

## Terminal: Ecall 0x130 and 0x131

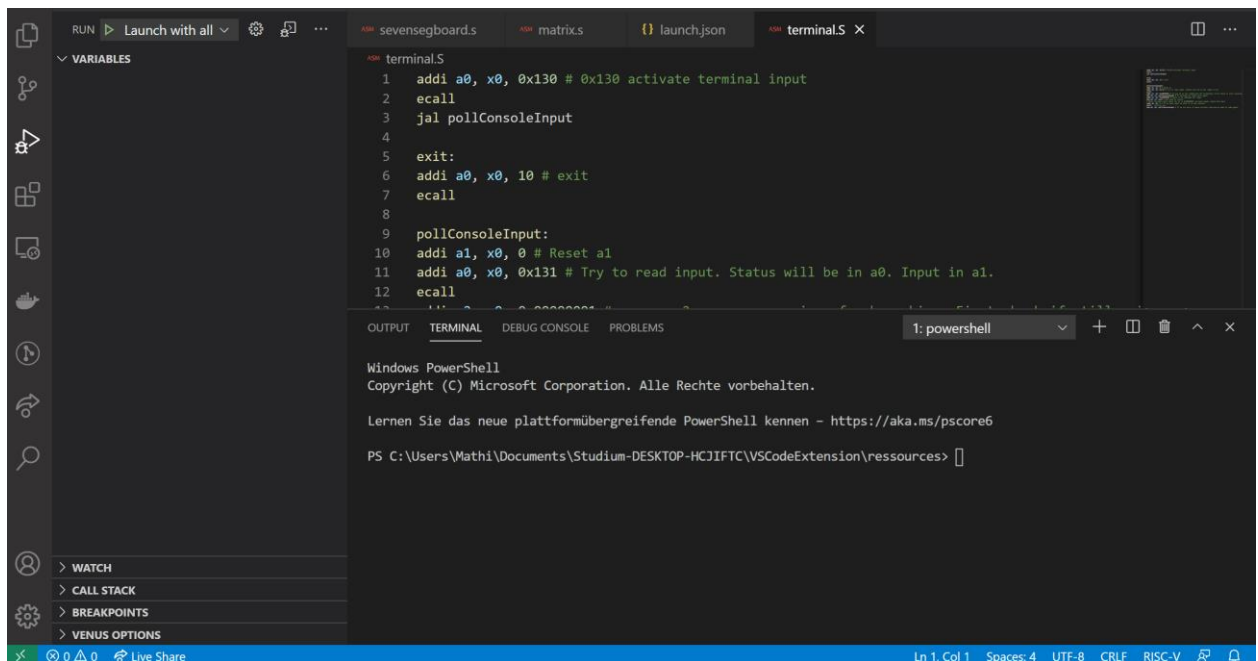
Thanh Terminal hỗ trợ theo chuẩn của Venus Ecalls để in số nguyên và chuỗi cũng như đầu ra lỗi. Ngoài ra, Đầu vào cũng được hỗ trợ (hiện chỉ thông qua polling cho đến khi hàm Interrupts được hỗ trợ).

Để kích hoạt Input, Ecall 0x130 phải được gọi. Sau đó, bạn có thể gọi hàm ecall 0x131 để đọc đầu vào. Sau khi gửi thông tin đầu vào của bạn cùng nhập vào trong bảng điều khiển, bên trong đầu vào được lưu vào bộ đệm và có thể được đọc từng cái một bằng cách gọi 0x131. Có 3 trạng thái khi gọi 0x131 (đọc Đầu vào):

- a0 == 0x00000001: Still waiting for input.
- a0 == 0x00000000: All Input has been read. Buffer is empty.
- a0 == 0x00000002: Input has been detected and one character has been read.  
a1 == input in UTF-16 code.

Có thể xem ví dụ trong file terminal.s





## Example assignment

### Bài tập ví dụ

Thực hiện một bộ đếm với LED bảy phân đoạn. Đếm tăng khi nhấn nút 1 và đếm ngược khi nhấn nút 0. Hiển thị giá trị bộ đếm hiện tại trong LED bảy phân đoạn. Đèn LED màu đỏ nhấp nháy khi bộ đếm bị tràn.

# BÀI 2 : HƯỚNG DẪN CÀI ĐẶT RIPE MÔ PHỎNG ĐƯỜNG DỮ LIỆU BỘ XỬ LÝ RISC V

## MỤC TIÊU

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.

## CƠ SỞ LÝ THUYẾT VÀ CÂU HỎI KIỂM TRA

### I. HƯỚNG DẪN CÀI ĐẶT

B1: Truy cập liên kết <https://github.com/mortbopet/Ripes/releases>. Di chuyển xuống mục VSRTL như hình dưới và tải file AppImage



B2: Thay đổi quyền truy cập của file vừa tải:

Có 2 cách:

- Cách 1: Sử dụng Terminal

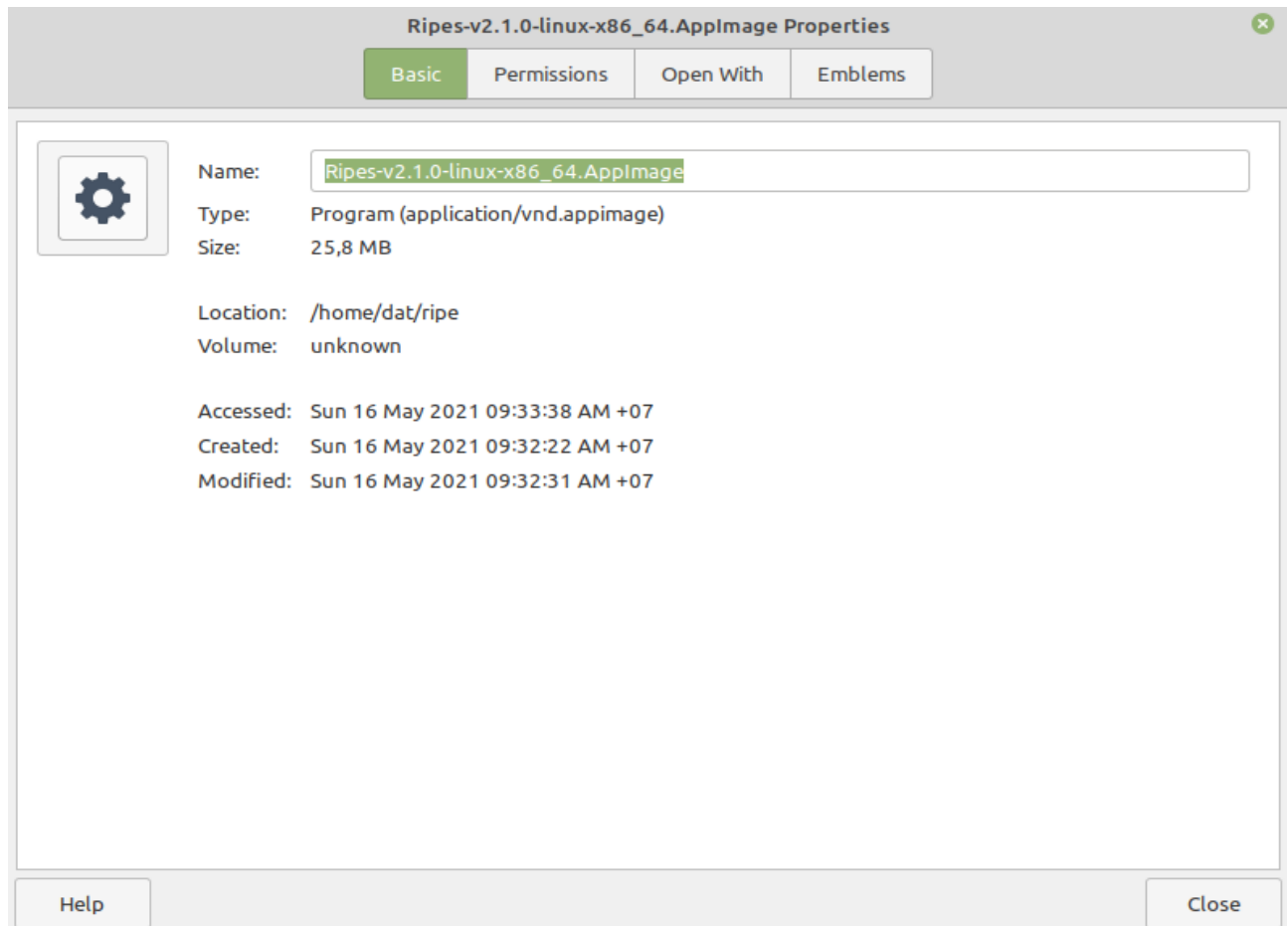
Gõ lệnh cd <Đường dẫn đến thư mục lưu file AppImage vừa tải>

Ví dụ: Trong trường hợp này file AppImage lưu trong thư mục RIPE với đường dẫn là: ~/RIPE

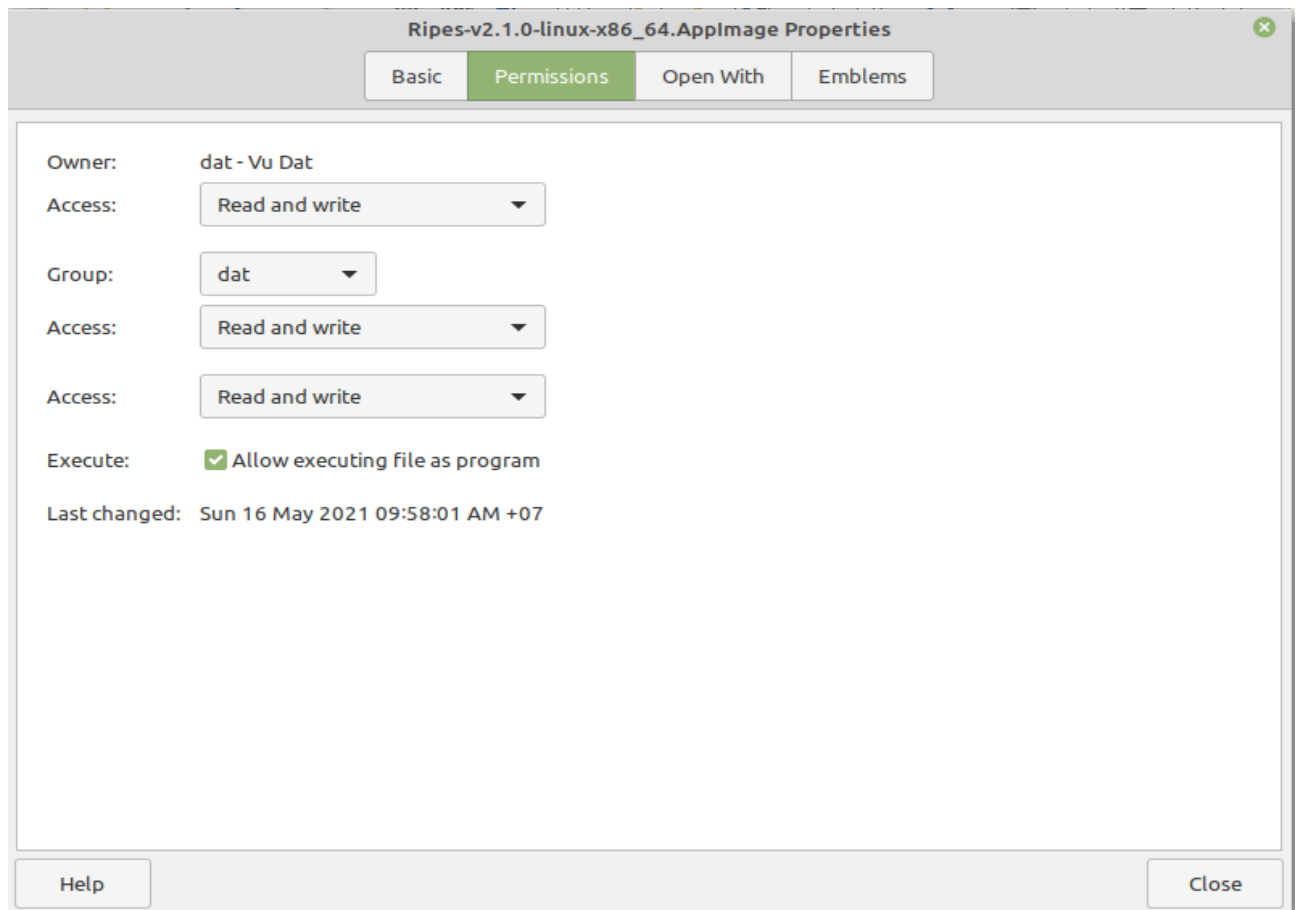
```
dat@DAT:~$ cd ~/RIPE
dat@DAT:~/RIPE$
```

Sau khi vào được thư mục chứa file, gõ lệnh `chmod 777 + <Tên file AppImage>` để thay đổi quyền truy cập file (Lệnh trên cấp phát quyền truy cập và thực thi cho tất cả User)

- Cách 2: Truy cập thư mục chứa file AppImage bằng cách click chuột đến thư mục đó (giống Windows)  
Click chuột phải vào file AppImage, sau đó chọn Properties  
Cửa sổ Properties hiện ra như hình dưới



Chọn Tab Permissions và chọn các Properties như hình dưới

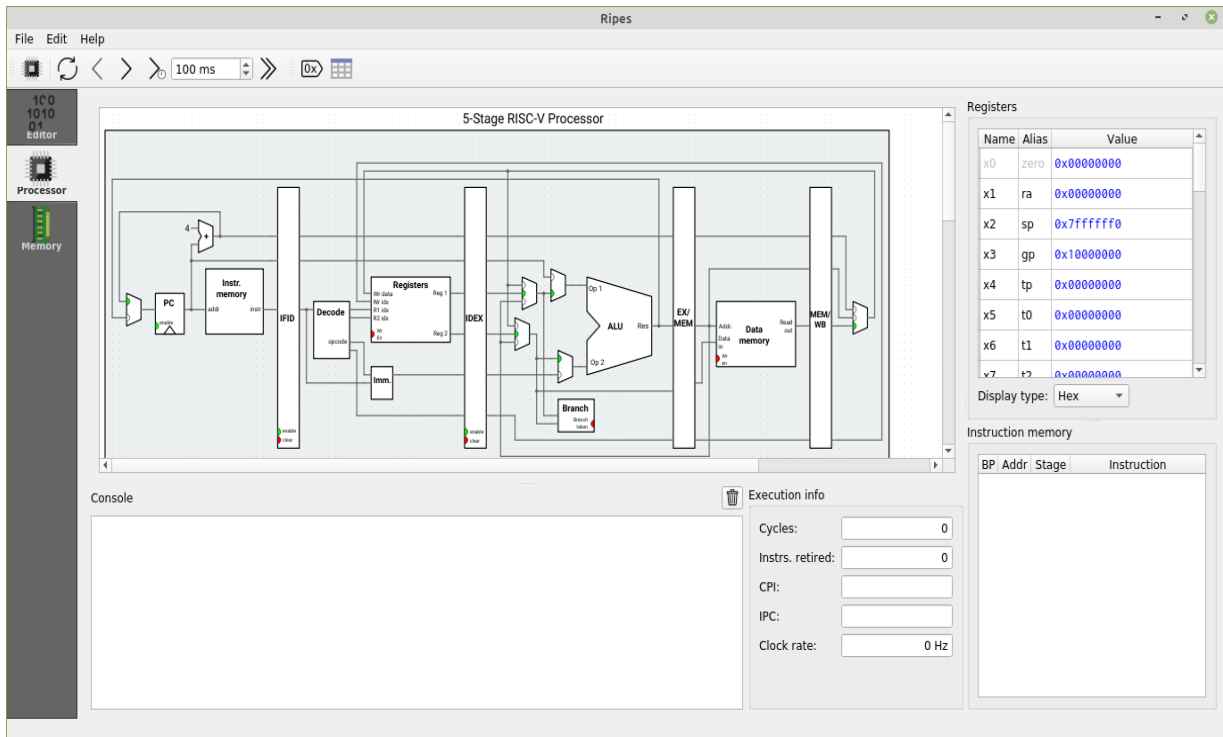


Sau khi chọn xong thì đóng cửa sổ Properties.

### B3: Chạy RIZES

Trên Terminal gõ tên file AppImage và nhấn Enter (Lưu ý: Terminal phải đang ở thư mục lưu file AppImage)

Hoặc trên giao diện GUI, double click vào file AppImage. Giao diện của RIZES hiện lên như hình vẽ.

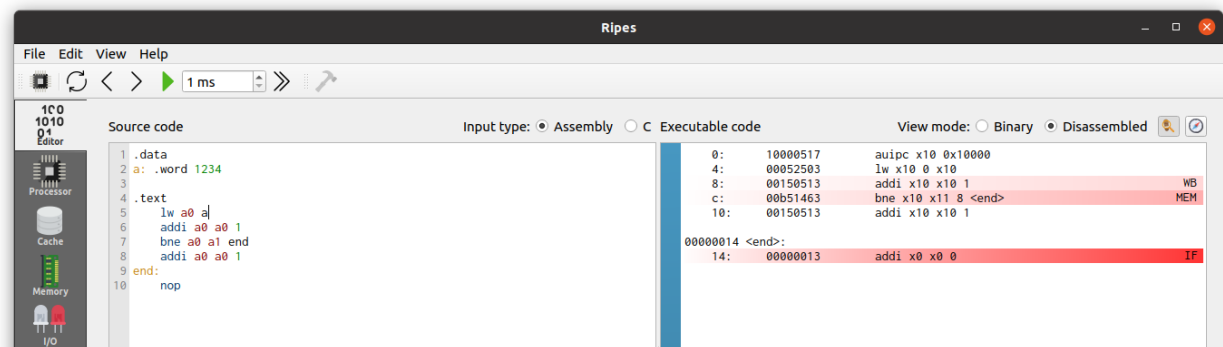


Cài đặt RIPES trên WINDOWS

<https://github.com/mortbopet/Ripes/releases>

Các tính năng của Ripes được mô tả dưới đây

## THE EDITOR TAB

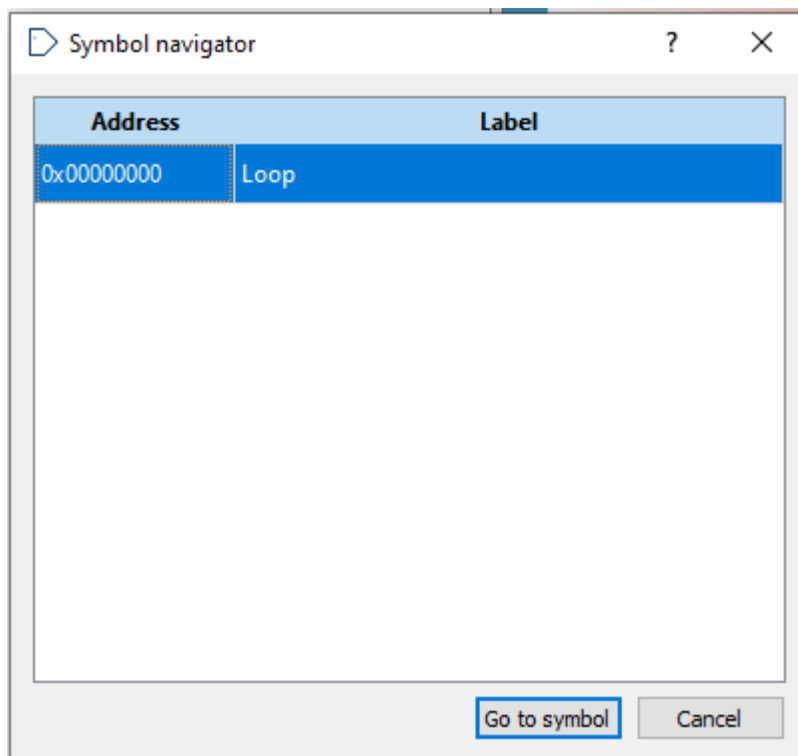


Tính năng ở bảng soạn thảo (Editor tab) hiển thị 2 đoạn code. Đoạn ở phía bên tay trái là vị trí có thể viết chương trình bằng hợp ngữ tương ứng với tập lệnh RISC-V RV32(I/M). Bất kỳ chỉnh sửa nào cũng được thực thi trên chương trình hợp ngữ này và không có báo lỗi cú pháp – mã hợp ngữ sẽ tự động được thực hiện vào chèn vào trong bộ mô phỏng. Nếu có trình biên dịch C được cài đặt, có thể đưa vào file định dạng C và có thể viết chương trình trực tiếp bằng ngôn ngữ với Ripes [this wiki page](#)

Kế bên phải trình soạn thảo (Editor Tab) là cửa sổ Program viewer cho phép người dùng nhìn được mã hợp ngữ của các lệnh đã soạn. Click vào thanh màu xanh dương bên trái cửa sổ Program Viewer để đặt các breakpoint (chương trình sẽ break việc thực hiện ở vị trí đó để chờ người dùng bấm chạy từng lệnh). Nhấn vào biểu tượng la bàn ở góc phải trên của sổ Program viewer để hiển thị tất cả các nhãn sử dụng trong chương trình, vị trí của chúng trong bộ nhớ. Như hình dưới



Sau khi nhấn cửa sổ Symbol Navigator hiện ra như hình dưới, các nhãn cùng địa chỉ được liệt kê:

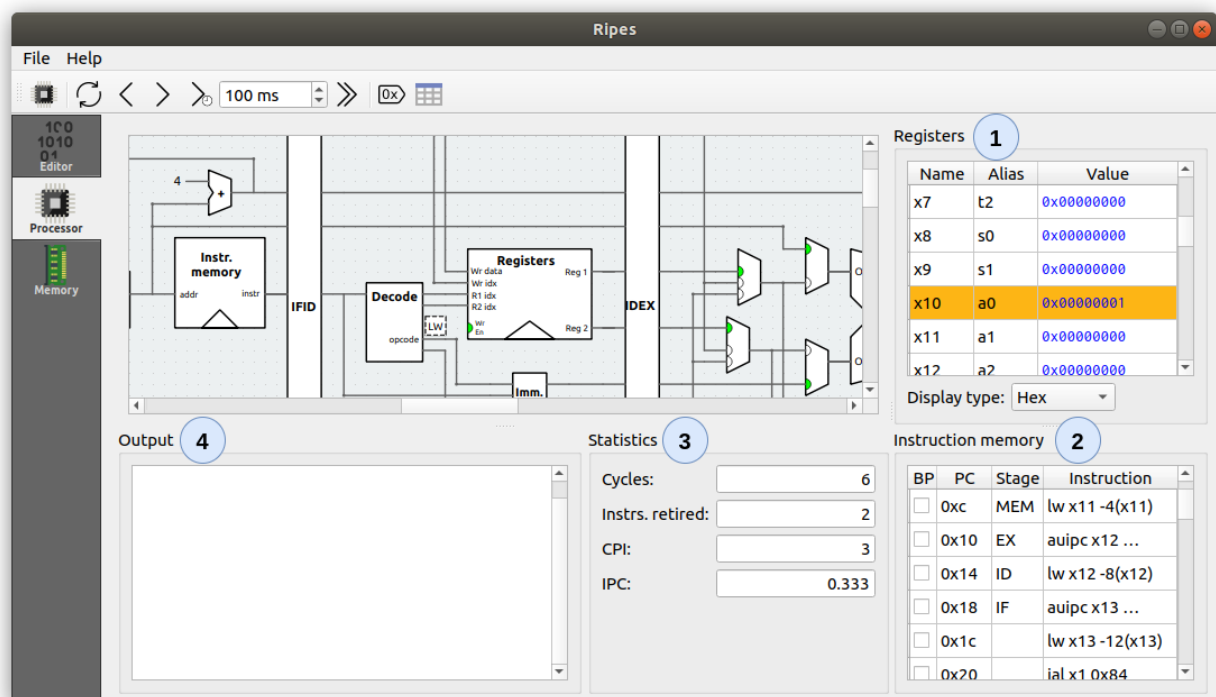


Ripes có sẵn rất nhiều ví dụ viết bằng assembly, để xem các ví dụ đó **Chọn File -> Load Example.**

Dưới đây là 1 ví dụ. Chương trình tải 1 giá trị từ bộ nhớ vào thanh ghi, sau đó tăng giá trị đó lên 1

```
.data
w: .word 0x1234
.text
lw a0 w
addi a0 a0 1
```

## THE PROCESSOR TAB



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

- 1: **Registers**: Tập các thanh ghi của bộ xử lý. 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: **Instruction memory**: Hiển thị các lệnh của chương trình hiện tại đang được thực thi trong bộ xử lý
  - BP**: Breakpoints, click để đánh dấu. Các breakpoint được đánh dấu trong Editor sẽ được thể hiện ở cửa sổ này

- **PC:** Địa chỉ của lệnh hiện tại
- **Stage:** Trạng thái thực hiện của lệnh
- **Instruction:** Lệnh dưới dạng hợp ngữ
- 3: **Statistics:** Thống kê số chu kỳ thực hiện, số lệnh đã thực hiện, số chu kỳ trên 1 lệnh.
- 4: **Output:** Output của các system call ghi giá trị ra ngoài vi (ecall)

## TỔNG QUAN VỀ BỘ XỬ LÝ

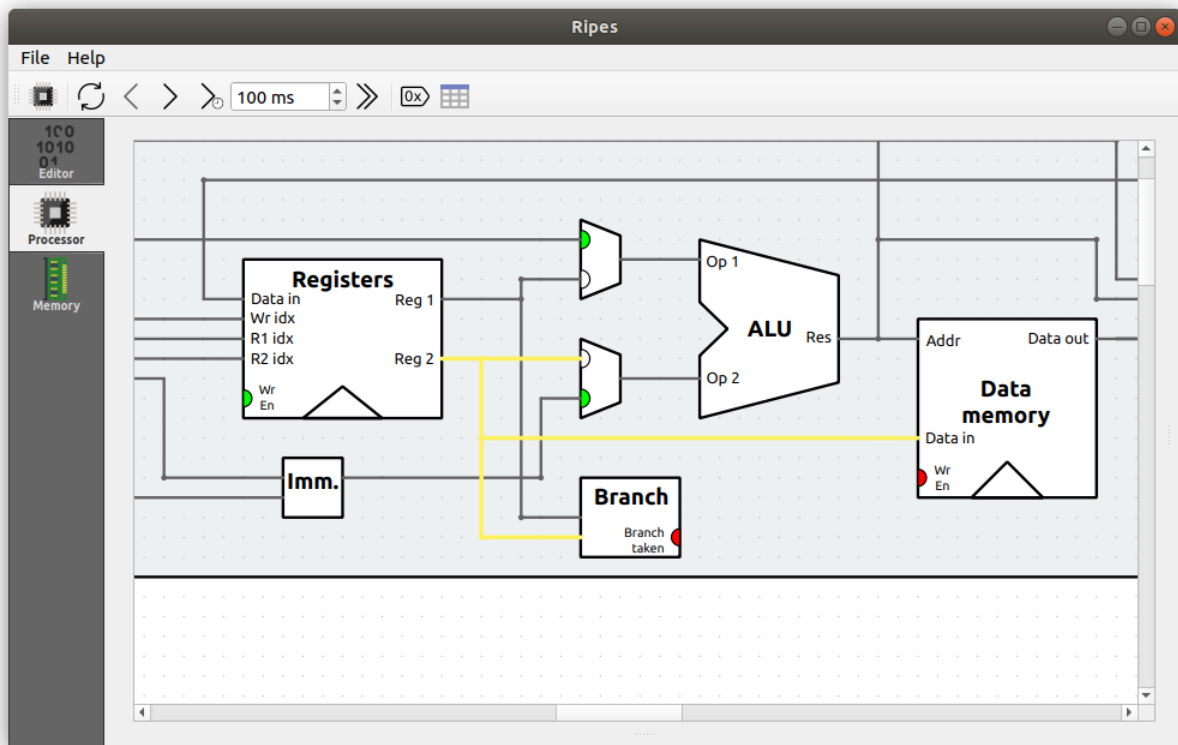
Các mô hình bộ xử lý trong Ripes thể hiện các đường dữ liệu từ các thành phần cơ bản như mux, bộ nhớ, thanh ghi

- Multiplexers thể hiện đầu vào nào được chọn bằng việc highlight màu xanh cổng vào.
- Rất nhiều thành phần cần có tín hiệu để điều khiển hoạt động như clock đối với các thanh ghi, tín hiệu điều khiển lệnh nhảy,...
- Giá trị xuất ra ở các port được thể hiện trên các dây:
  - *Boolean (1-bit signals):* Nếu tín hiệu ở mức cao (1) thì dây màu xanh, mức thấp (0) thì dây màu xám
  - *Với các dây khác, nếu tín hiệu(giá trị) trên dây thay đổi thì dây sẽ nháy màu xanh*

Để zoom to hơn vào các thành phần của bộ xử lý nhấn **Ctrl + Lăn chuột (Cmd + Lăn chuột** trên MAC OS)

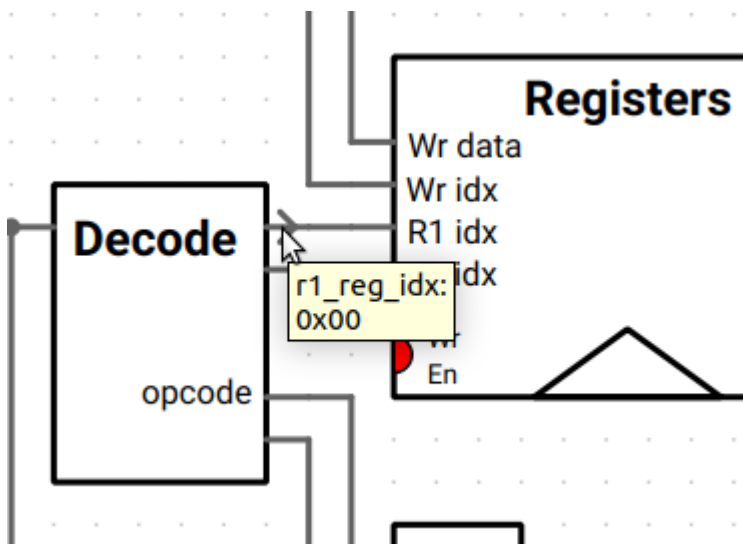
Click vào 1 dây, dây đó sẽ được highlight màu vàng, điều này sẽ giúp ích trong trường hợp mô hình phức tạp, bằng việc highlight sẽ giúp người dùng biết được dây đó được nối đến các vị trí nào



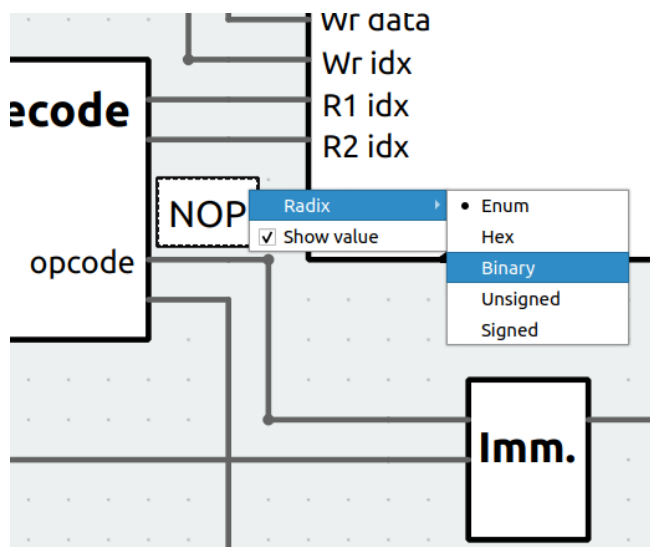


Ngoài việc mô phỏng, Ripes có thể hiển thị giá trị trên bất cứ dây nào, bất cứ vị trí nào trong quá trình hoạt động

1. Trỏ vào bất cứ vị trí nào trên dây bất kì, giá trị trên dây sẽ được hiển thị cùng tên cổng dây đó nối đến



2. Click chuột phải vào dây cần biết giá trị tín hiệu, chọn Show value để xem giá trị tín hiệu trên đó. Để show tất cả giá trị trên các dây **Chọn View -> Show processor signal value**. Sau khi hiển thị giá trị, click chuột phải vào dây đó 1 lần nữa để chọn cơ số, có thể chọn giữa nhị phân, hexa, thập phân có dấu, không dấu như hình dưới



## ĐIỀU KHIỂN TRÌNH MÔ PHỎNG

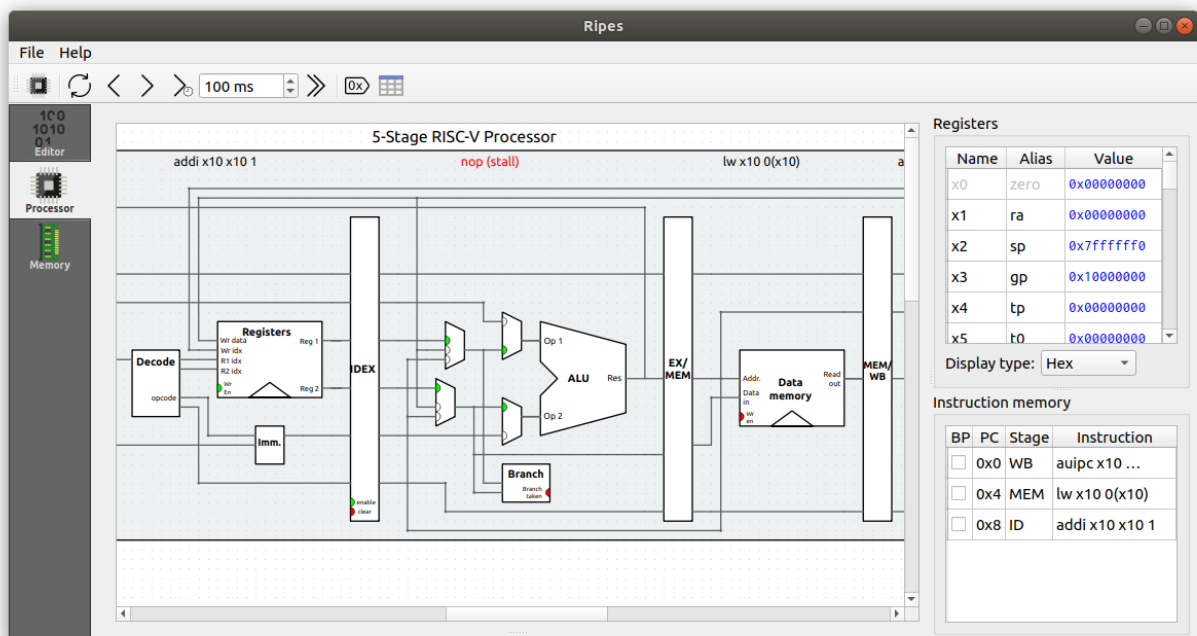
Thanh công cụ trong Ripes chứa tất cả các tùy chọn để điều khiển trình mô phỏng.

Select Processor	Reset	Reverse	Clock	Auto-clock	Run	Show stage table

- **Select Processor:** Mở hộp thoại Processor Selection, chọn bộ xử lý ở đây (sẽ minh họa ở dưới)
- **Reset:** Reset bộ xử lý, đưa thanh ghi PC về đầu chương trình hiện tại và xóa bộ nhớ, thanh ghi hiện tại
- **Reverse:** Quay lại lệnh trước đó
- **Clock:** Thời gian thực hiện trên 1 lệnh
- **Auto-clock:** Tự động chạy với 1 clock được đặt sẵn. Nếu gặp breakpoint chương trình sẽ dừng chế độ tự chạy
- **Run:** Chạy chương trình (chế độ này chạy rất nhanh, sẽ không nhìn thấy cá thay đổi qua từng lệnh, chỉ thấy kết quả). Chế độ sẽ dừng nếu gặp breakpoint.
- **Show stage table:** Hiển thị các lệnh đang trong các tầng pipeline trên 1 chu kỳ. Chú ý nếu ta chạy chế độ Run thì cá stage sẽ không được ghi lại
- Select View->Hiển thị giá trị ở các port của bộ xử lý

Stage table									
	0	1	2	3	4	5	6	7	8
auipc x10 0x65536	IF	ID	EX	MEM	WB				
lw x10 0(x10)		IF	ID	EX	MEM	WB			
addi x10 x10 1			IF	ID	-	EX	MEM	WB	

Khi đang thực thi, ta có thể nhận thấy chương trình được tải sớm hơn, ở chu kỳ thứ 4, việc tải lệnh vào sẽ phụ thuộc vào việc thực thi của lệnh thứ 2 và lệnh thứ 3. Kết quả là ở pha ID (Giải mã lệnh) sẽ bị trễ đi 1 chu kỳ để chờ thực hiện. Chờ trong pipeline (do xảy ra xung đột) và flush (do luồng điều khiển) sẽ dẫn tới các tầng pipeline cần thêm lệnh NOP được highlight màu đỏ



## CHỌN BỘ XỬ LÝ

Do hỗ trợ mô phỏng nhiều mô hình bộ xử lý, 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 lý đơn xung nhịp sang bộ xử lý đường ống. Ripes hỗ trợ các mô hình bộ xử lý dưới đây:

- Bộ xử lý Riscv đơn xung nhịp
- Bộ xử lý Riscv 5 tầng pipeline (không phát hiện xung đột, không forwarding)

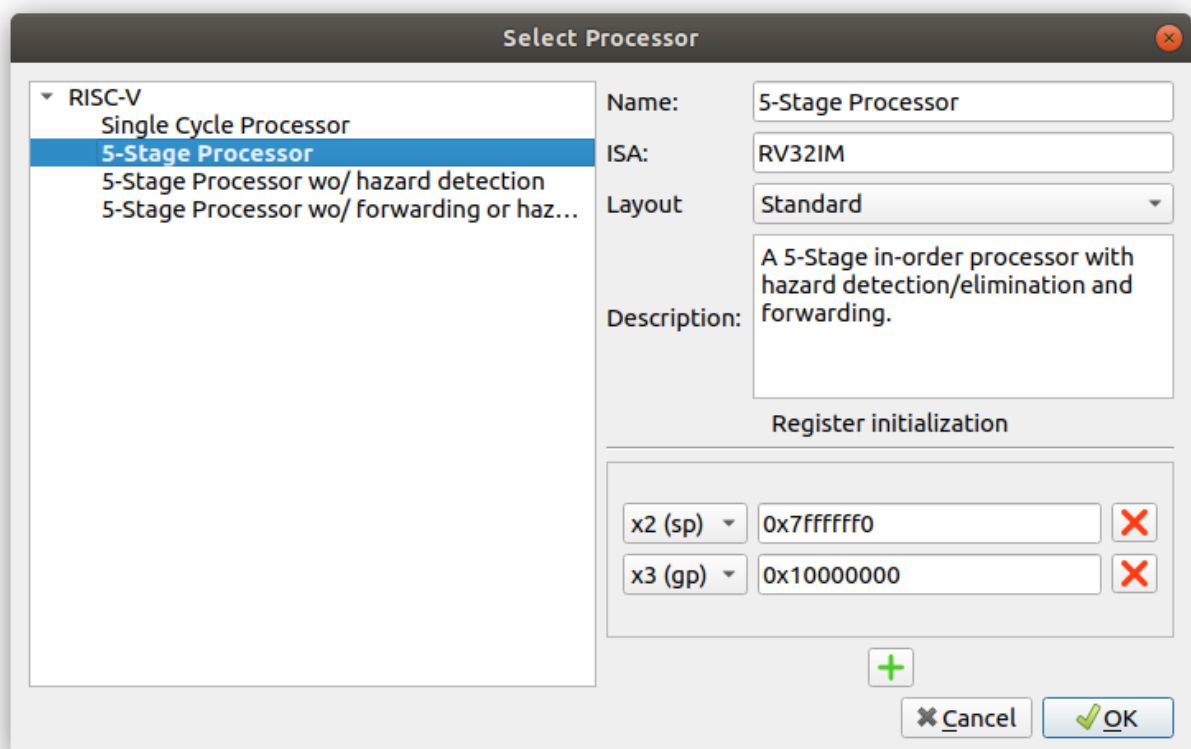
- Bộ xử lí Riscv 5 tầng pipeline (không phát hiện xung đột)
- Bộ xử lí Riscv 5 tầng pipeline

Furthermore, each processor provides multiple layouts of the processor. By default, the following two layouts are provided:

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

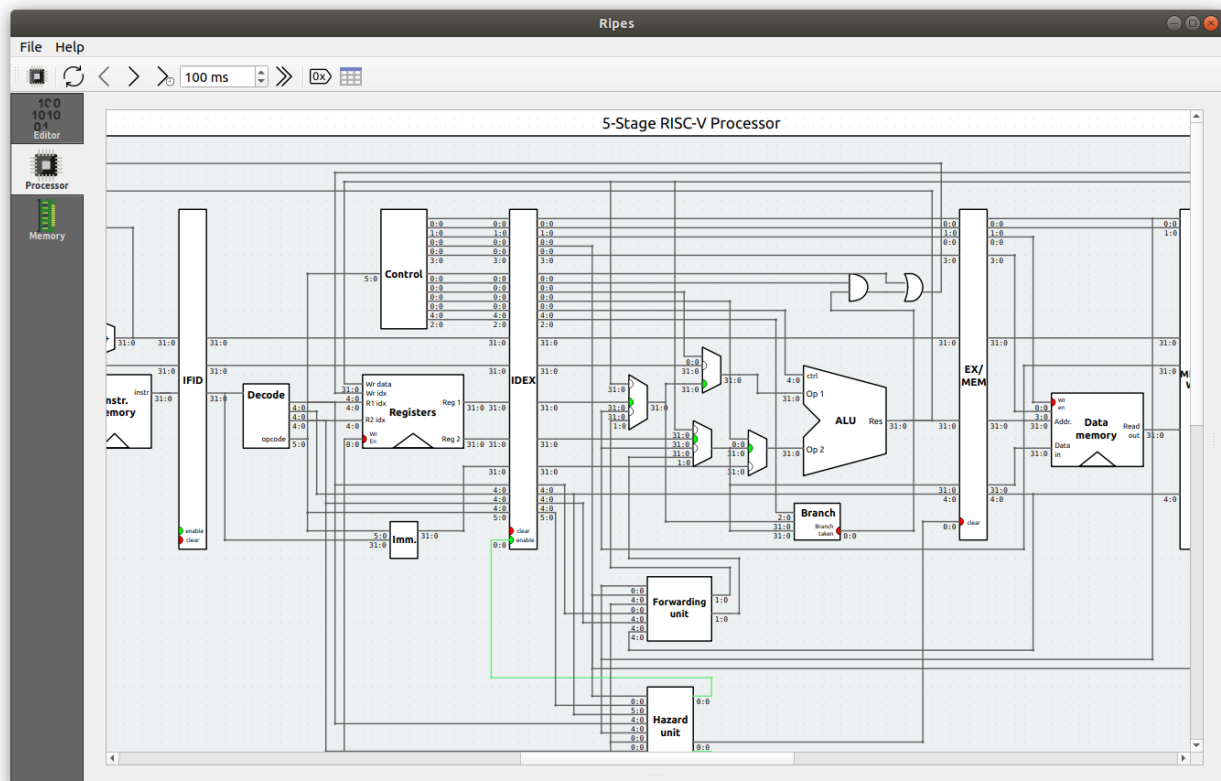
- **Standard:** Mô hình đơn giản, khối điều khiển và các tín hiệu điều khiển bị ẩn
- **Extended:** Mô hình mở rộng, khối điều khiển, các tín hiệu điều khiển cùng với độ rộng bus được thể hiện

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



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

Một ví dụ về lựa chọn bộ xử lí, ảnh dưới đây thể hiện layout mở rộng của bộ xử lí Riscv 5 tầng pipeline



## TAB MEMORY

Address	Word	Byte 0	Byte 1	Byte 2	Byte 3
0x00000024	X	X	X	X	X
0x00000020	X	X	X	X	X
0x0000001c	X	X	X	X	X
0x00000018	0x00000000	0x00	0x00	0x00	0x00
0x00000014	0x00000013	0x13	0x00	0x00	0x00
0x00000010	0x00150513	0x13	0x05	0x15	0x00
0x0000000c	0x00b51463	0x63	0x14	0xb5	0x00
0x00000008	0x00150513	0x13	0x05	0x15	0x00
0x00000004	0x00052503	0x03	0x25	0x05	0x00
0x00000000	0x10000517	0x17	0x05	0x00	0x10
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-
-	-	-	-	-	-

Name	Size	Range
.text	24	0x00000000 - 0x00000018
.data	4	0x10000000 - 0x10000004
.bss	0	0x11000000 - 0x11000000
LED Matrix 0	32	0xf0000000 - 0xf0000020
Switches 0	4	0xf0000020 - 0xf0000024

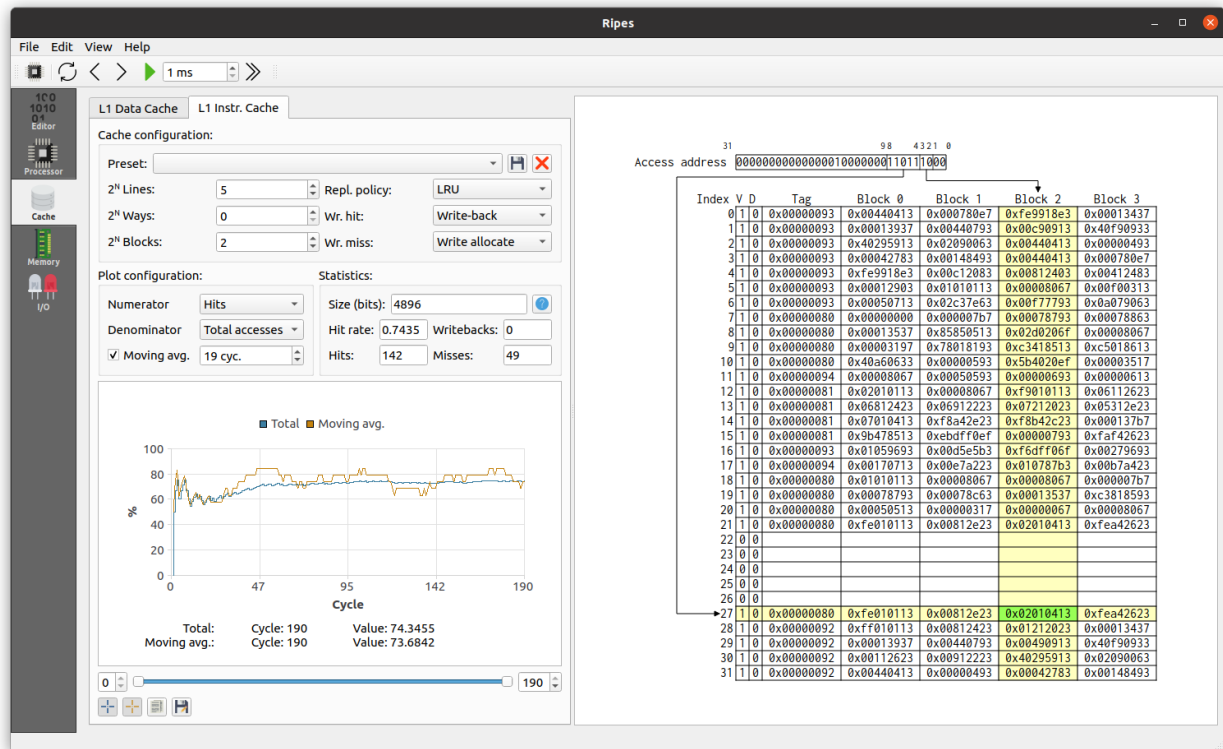
Tab memory thể hiện bộ nhớ (địa chỉ và giá trị) của bộ xử lý. Định vị ô nhớ được thực hiện như sau:

- **Lăn chuột**
- **Go to register** chuyển đến ô nhớ mà địa chỉ của nó lưu trong 1 thanh ghi cụ thể

- **Go to section** chuyển đến ô nhớ của 1 section cụ thể (ví dụ: vùng nhớ lệnh, vùng nhớ tĩnh, vùng nhớ động,...). Thêm vào đó ta có thể lựa chọn hệ cơ số để hiển thị giá trị cũng như địa chỉ ô nhớ

Khi bộ xử lý được reset, tất cả ô nhớ được ghi bởi chương trình sẽ được khôi phục lại giá trị ban đầu lúc chương trình chưa chạy

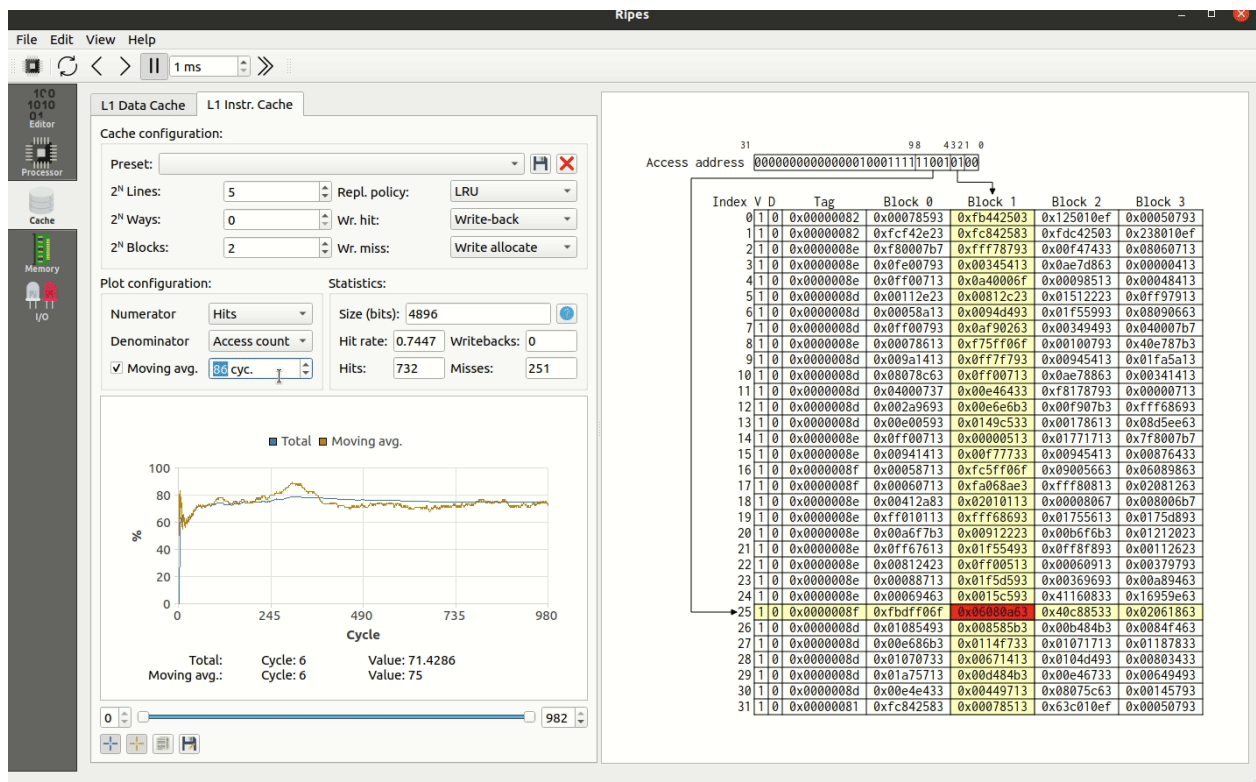
## TAB CACHE



## Ripes cung cấp khả năng mô phỏng bộ nhớ cache

## MÔ PHỎNG BỘ NHỚ CACHE

Từ phiên bản 2.1.0, Ripes thêm tính năng mô phỏng bộ nhớ cache. Trình mô phỏng cache mô phỏng các cache dữ liệu cấp 1, cache lệnh cấp 1, ở tab cấu hình layout ta có thể cấu hình thêm cho cache. Với công cụ này, người dùng có thể phân tích hiệu năng của chương trình khi sử dụng bộ nhớ cache và phân tích sự khác biệt giữa cách phối hợp hoạt động của bộ nhớ cache với bộ nhớ chính.



Trước khi bắt đầu, ta có một số lưu ý trong việc sử dụng cache:



- Khuyến khích sử dụng bộ xử lý đơn xung nhịp để mô phỏng cache
  - Nếu mô phỏng cache với bộ xử lý đường ống thì ta phải chowfddocj từ bộ nhớ chính. Nếu 1 tầng phải chờ, mỗi chu kỳ chờ sẽ được tính là tăng dung số lượt truy cập bộ nhớ. Điều đó dẫn đến kết quả thực thi có thể giống hoặc khác giữa các hệ thống khác nhau. Nên để tránh điều đó xảy ra, ta sử dụng mô hình bộ xử lý đơn xung nhịp.
  - Bộ xử lý đơn xung nhịp có tốc độ thực thi nhanh hơn đáng kể so với bộ xử lý đường ống.
- Bộ xử lý sẽ không truy cập vào cache khi đang truy cập bộ nhớ chính. Thay vào đó, trình mô phỏng cache được móc nối với mô hình bộ xử lý và phân tích việc truy cập bộ nhớ trong mỗi chu kỳ xung nhịp. Sau đó những lần truy cập bộ nhớ đó được sử dụng để tính toán hiệu năng trong trình mô phỏng cache. Điều đó có nghĩa là
  - Ripes không mô phỏng trễ truy nhập cache, cũng không cung cấp bất cứ công cụ nào mô phỏng thời gian chạy thực của CPU. Nó chỉ tính toán các yếu tố như: tỉ lệ hit, miss, writeback
  - Dirty cache lines (Khi cache được cấu hình mở chế độ writeback) vẫn có thể nhìn thấy trong bộ nhớ. Nói cách khác, các dữ liệu luôn luôn được ghi vào bộ nhớ chính cho dù cache có đang được cấu hình ở chế độ writeback



## BỘ NHỚ CACHE

### Cấu hình cho cache

Cache configuration:

Preset:	32-entry 4-word fully associative			
2 <sup>N</sup> Lines:	<input type="text" value="0"/>	Repl. policy:	LRU	
2 <sup>N</sup> Ways:	<input type="text" value="5"/>	Wr. hit:	Write-back	
2 <sup>N</sup> Blocks:	<input type="text" value="2"/>	Wr. miss:	Write allocate	

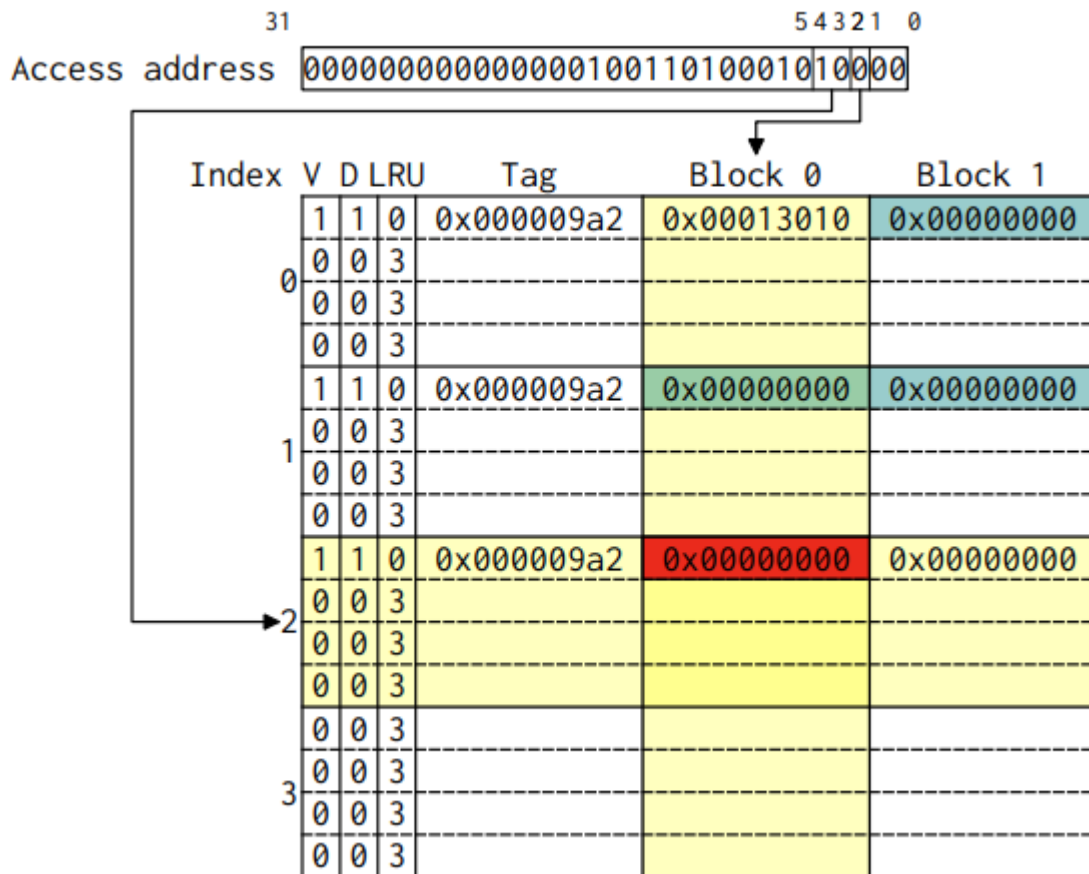
Có thể cấu hình cache với các tùy chọn sau đây:

- **Ways:** Đặc tả dưới dạng lũy thừa của 2 (vd: 2 ways thì giá trị cần set là 1, trong khi cache ánh xạ trực tiếp (1 way) thì set bằng 0 )
- **Lines:** Số hàng cache. Số hàng cache được định nghĩa là max của chỉ số dùng để đánh chỉ số cho hàng cache. Đặc tả với lũy thừa của 2
- **Blocks:** Số khối trong mỗi hàng cache. Số khối thể hiện max của chỉ số để lựa chọn khối trong 1 hàng. Đặc tả dưới dạng lũy thừa của 2
- **Wr. hit/Wr. miss:** Quy tắc ghi vào cache. Truy cập [this Wikipedia article](#) để biết thêm chi tiết
- **Repl. policy:** Quy tắc thay thế cache. Truy cập [this Wikipedia article](#) để biết thêm chi tiết
- Thêm vào đó, có rất nhiều thiết lập sẵn có thể tái sử dụng, người dùng cũng có thể tạo ra các thiết lập riêng của mình để phục vụ việc tái sử dụng

### The Cache View

Dựa trên cấu hình hiện tại của cache, Ripes cung cấp trình mô phỏng trực quan trạng thái của cache





Cache được vẽ dưới dạng bảng, trong đó các hàng được định nghĩa là:

- **Cache lines** được phân cách bởi đường nét liền được đánh số bên trái
- **Cache ways** chứa trong cache line. Các way trong cùng 1 line được phân tách bởi đường nét đứt

Thông thường, một set-associative cache được vẽ thành các bảng riêng biệt cho từng way. Điều này được thể hiện trong Ripes như trình bày dưới đây

Các cột trong cache được kí hiệu:

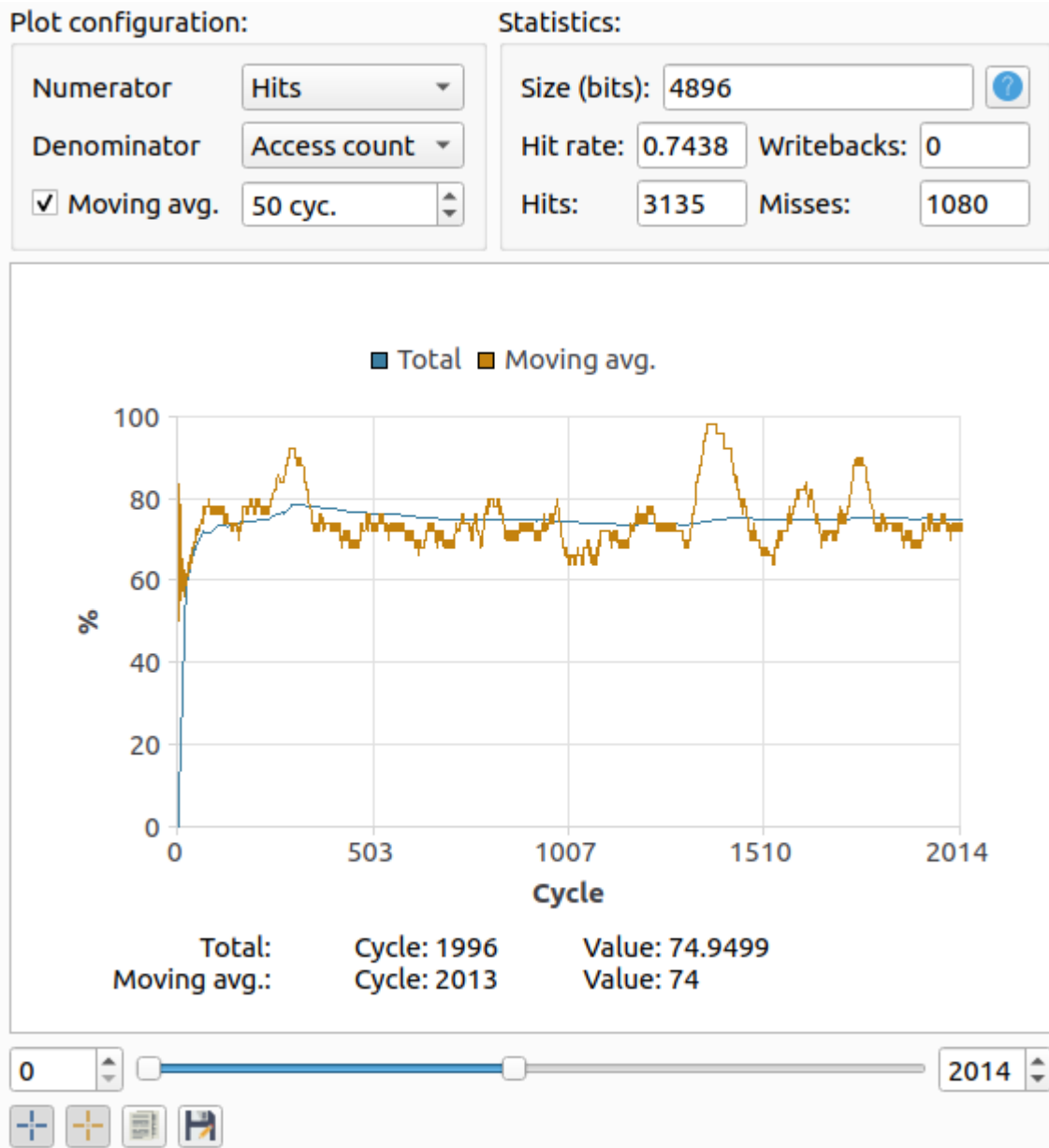
- **V**: Valid bit. Xác định way đó có chứa dữ liệu hay không
- **D**: Dirty bit. Xác định way đó có chứa dirty data hay không (the cache way was written to, in write-back mode).
- **LRU**: Hiển thị khi ways > 0 và Repl. policy = LRU. Giá trị vừa được truy nhập sẽ có LRU = 0, và giá trị sắp được lấy ra sẽ có giá trị LRU =  $2^{(ways)} - 1$ .
- **Tag**: Nhận diện tại cửa của way trong cache.
- **Block #**: Dữ liệu trong cache.

Ta có thể tương tác với cache view như sau:

- Di chuột đến 1 block sẽ hiển thị giá trị vật lý của dữ liệu trong cache
- Nhấn chuột vào block sẽ chuyển chế độ theo dõi bộ nhớ thành chế độ theo dõi địa chỉ vật lý tương ứng của giá trị trong cache.
- Chế độ theo dõi cache có thể được zoom bằng cách bấm ctrl+lăn chuột (cmd+scroll trong OSX).

Khi cache được đánh địa chỉ, hàng *line* và cột *block* tương ứng sẽ được in đậm màu vàng. Giao giữa hàng và cột này tương ứng với tất cả các ô có thể chứa giá trị được lưu trữ trong bộ nhớ cache. Do đó đối với bộ đệm được ánh xạ trực tiếp, chỉ có 1 ô nằm trong giao điểm trong khi đối với bộ đệm N-way, N ô sẽ được in đậm. Trong hình ảnh bộ đệm kết hợp 4 chiều trên, chúng ta thấy rằng 4 ô được in đậm Ô được in đậm là màu xanh cho biết a cache hit, trong khi màu đỏ cho biết là cache miss. Ô được in đậm màu xanh nước biển cho biết giá trị là dirty (với chính sách write-hit "write-back" – ghi lại).

## Cache Access Statistics & Plotting



Để cung cấp thông tin chi tiết về mô phỏng bộ đệm, các thông số bộ đệm khác nhau có thể được vẽ theo thời gian thực. Với mỗi chu kỳ, thông tin truy nhập cache tiếp theo sẽ được ghi lại:

- **Ghi:** # số lần cache được truy nhập thông qua ghi
- **Reads:** # số lần cache được truy nhập thông qua đọc
- **Hits:** # số lần truy nhập cache (đọc hoặc ghi) là hit
- **Misses:** # số lần truy nhập cache (đọc hoặc ghi) là miss
- **Writebacks:** # số lần a cache được ghi ngược lại vào bộ nhớ
- **Access count:** # số lần cache được đọc- ghi hoặc hit- miss một cách hiệu quả

Từ đây, ta có thể vẽ tỷ lệ bất kỳ giữa các trọng số này, bằng cách chọn tử số và mẫu số. Ví dụ, để theo dõi tỷ lệ cache hit, chọn số lần hit và số lần truy nhập. Quá trình

vẽ biểu đồ trung bình của các biến được chọn cũng có thể được bật. Điều này rất hữu ích khi ta xác định điểm nào trong chương trình mà tỉ lệ hit thay đổi đáng kể. Để xem bảng phân tích của kích thước cache (bit) theo lý thuyết, ta bấm nút. Ở phần dưới cùng của cửa chế độ xem, ta có thể thực hiện các hành động sau:

- : Copy tất cả thông tin truy nhập bộ đệm trong tất cả chu kỳ lên clipboard
- : Lưu biểu đồ vừa vẽ vào file
- : Bật các điểm đánh dấu cho đồ thị động trung bình và đồ thị tổng.

## EXAMPLE

Ví dụ sau minh họa cách các cấu hình bộ đệm khác nhau có thể có tác động lên tỷ lệ hit của cache. Điều này được thực thi qua việc thực thi một chương trình dùng các cấu hình bộ đệm khác nhau

### Chương trình ví dụ:

Chương trình ví dụ sau đây sẽ cho chúng ta nhìn 1 cách thấu đáo một số quy luật truy nhập bộ nhớ, tại đó việc điều chỉnh quy luật này sẽ ảnh hưởng đến hiệu năng của bộ đệm cache. Chương trình minh họa phía dưới được viết bằng ngôn ngữ C và được biên dịch thành ngôn ngữ assembly cho RISC-V dùng trình biên dịch Explorer (Tra khảo wiki về các phương pháp chuyển đổi assembly cho RISC-V tạo bởi explorer sang ngôn ngữ assembly tương thích với trình của assembler của ripes

```
unsigned stride = 4;
unsigned accessesPerTurn = 128;
unsigned turns = 2;
unsigned* baseAddress = (unsigned*)0x1000;
```

```
void cacheLoop() {
    for(unsigned i = 0; i < turns; i++) {
        volatile unsigned* address = baseAddress;
        for(unsigned j = 0; j < accessesPerTurn; j++) {
            *address;
            address += stride;
        }
    }
}
```

Chương trình trên sẽ được chuyển thành ngôn ngữ assembly như sau:

```
.data
stride:      .word  512 # in words
accessesPerTurn: .word  2
```

```

turns:      .word 128
baseAddress: .word 4096

```

```

.text
cacheLoop():
    lw    a6, turns
    lw    a7, baseAddress
    lw    a2, stride
    lw    a0, accessesPerTurn
    mv    a3, zero
    slli  a4, a2, 2
    j     .LBB0_3
.LBB0_2:
    addi  a3, a3, 1
    beq   a3, a6, .LBB0_5
.LBB0_3:
    add   a5, zero, a7
    add   a2, zero, a0
    beqz  a0, .LBB0_2
.LBB0_4:
    lw    a1, 0(a5)
    addi  a2, a2, -1
    add   a5, a5, a4
    bnez  a2, .LBB0_4
    j     .LBB0_2
.LBB0_5:
    # exit

```

## Simulating Different Cache Configurations

Ban đầu, chúng ta sẽ phải xử lý các biến sau trong mã nguồn:

```

.data
stride:      .word 512 # ở dạng từ
accessesPerTurn: .word 2
turns:      .word 128
baseAddress: .word 4096

```

Đi tới Memory tab và với cache dữ liệu, hãy chọn bộ đệm đã được ánh xạ 32-entry 4 – từ sẵn. Tiếp theo, nhấn nút Run để chạy chương trình. Ta thấy rằng cache có thể đạt đến tỷ lệ hit là 0.01154. Trong chương trình ví dụ, ta thấy rằng chúng ta đã chỉ định 1 sải là 512 từ. Điều này dẫn đến cách truy nhập sau: Truy nhập 2 bị trí bộ nhớ khác nhau:

- 1: 4096 = 0b00010000 00000000
- 2: 4608 = 0b00010010 00000000
- 3: 4096 = 0b00010000 00000000

4: 4608 = 0b00010010 00000000

...

Đối với cấu hình bộ đệm được chọn, chúng ta thấy rằng mục dòng của bộ đệm là bitmask sau:

0b00000001 11110000

Áp dụng bitmask cho quy luật truy nhập trên, chúng ta thấy rằng tất cả các địa chỉ truy nhập đều có mặt nạ thành 0x0 và do đó sẽ được đánh địa chỉ tới dòng bộ đệm tương tự. Nói cách khác chúng ta không có sự lựa chọn cho việc đánh địa chỉ bộ đệm đối với quy tắc truy nhập được cho.

Trong trường hợp này, bộ đệm liên kết tập hợp có thể phù hợp hơn bộ đệm được ánh xạ trực tiếp. Chọn bộ đệm 32-entry, 4-từ 2-way được cài đặt sẵn thành bộ đệm liên kết tập hợp. Lưu ý rằng cách thiết kế bộ đệm này cho ta số lượng block được lưu trữ trong bộ đệm tương đương với cách thiết kế bộ đệm ánh xạ trực tiếp ở phần trước. Sau đó chạy lại chương trình. Trong trường hợp này, chúng ta thấy rằng cache đạt được tỷ lệ hit lên tới 0.9885. Chúng ta không còn gặp xung đột liên kết nữa, bởi vì tại mỗi truy nhập được đặt bằng cách khác nhau trong khi đang ánh xạ tới địa chỉ của cùng một bộ đệm.

Bài tập lớn 1: Sử dụng toolchain để biên dịch hệ điều hành Linux cho RISC-V QEMU Emulator theo tài liệu tham khảo:

<https://riscv.org/wp-content/uploads/2015/02/riscv-software-stack-tutorial-hpca2015.pdf>

và

<https://risc-v-getting-started-guide.readthedocs.io/en/latest/linux-introduction.html>

Bài tập lớn 2: Sử dụng toolchain để biên dịch hệ điều hành Zephyr cho RISC-V QEMU Emulator theo tài liệu tham khảo

<https://risc-v-getting-started-guide.readthedocs.io/en/latest/zephyr-introduction.html>

Bài tập lớn 3: Đọc và trình bày về bộ nhớ theo tài liệu tham khảo:

What Every Programmer Should Know About Memory

và mô phỏng hiệu năng của cache dựa trên tài liệu tham khảo

<https://github.com/mortbopet/Ripes/wiki/Cache-Simulation>

viết chương trình tính ma trận, bộ lọc trung vị cho ảnh và khảo sát cấu trúc chương trình tối hiệu năng cache.

Bài tập lớn 4: Mô phỏng đường dữ liệu bộ xử lý RISC-V cho các chương trình viết bằng ngôn ngữ C

Tài liệu tham khảo:

<https://github.com/mortbopet/Ripes/wiki/Building-and-Executing-C-programs-with-Ripes>

<https://github.com/mortbopet/Ripes/wiki/Ripes-Introduction>

Bài tập lớn 5: Thiết kế bộ xử lý RISC-V pipeline bằng ngôn ngữ Verilog

Tham khảo tài liệu: <https://github.com/ultraembedded/riscv>

Bài tập lớn 6: Thiết kế bộ điều khiển cache cho bộ xử lý RISC-V pipeline bằng ngôn ngữ Verilog

Bài tập lớn 7: Đọc và trình bày về kiến trúc bộ xử lý đồ họa GPU. Minh họa việc lập trình bộ xử lý đồ họa GPU để triển khai bộ lọc trung vị cho ảnh và chương trình tính ma trận. Khảo sát hiệu năng với kích thước bộ lọc, kích thước ảnh và kích thước ma trận khác nhau.

Tài liệu tham khảo: Phục lục B, sách “Computer Organization and Design RISC-V Edition The Hardware Software Interface” by David A. Patterson, John L. Hennessy