

Table of Contents

Table of Contents	1
RISCV Heritage	2
Valid Instructions	2
Team Members	2
Technology Stack	2
File Structure	2
Features	3
General Instructions To Run Locally	3
Functionalities Our Project Offers	3

RISCV Heritage

≡ About	🖥️ A web simulator which converts the Assembly code written in RISCV syntax to Machine code.
≡ Link	https://github.com/subhamX/riscv

The following website is a course project under the guidance of [Dr T.V Kalyan](#). It is a web simulator which converts the **Assembly code** written in **RISCV syntax** to **Machine code** and provides a user-friendly environment for its execution. The simulator implements pipelining, branch prediction and data forwarding, which can be enabled or disabled by the user.

Valid Instructions

R format:

add, and, or, sll, slt, sra, srl, sub, xor, mul, div, rem

I format:

addi, andi, ori, lb, ld, lh, lw, jalr

S format:

sb, sw, sd, sh

SB format:

beq, bne, bge, blt

U format:

auipc, lui

UJ format:

jal

Assembler Directives:

.text, .data, .byte, .half, .word, .dword, .ascii

Team Members

[Bharat Ladrecha](#) 2018CSB1080

[Subham Sahu](#) 2018EEB1183

Technology Stack

1. Typescript
2. HTML/CSS
3. NodeJS (For Development Only)

File Structure

```
src
├── README.md
├── tsconfig.json
├── assets
├── index.html
├── css
├── style.css
└──
```

```
└─js
  └─main.ts
    └─encode
      └─execute
```

encode contains files required to convert **Assembly Code** written in **RISC V** syntax into **Machine Code**

execute contains files required to execute the generated **Machine Code**

main.ts is the entrypoint of the application

Features

1. On pressing **Ctrl+S** on **Editor Pane** current file will be downloaded.
2. On pressing **Ctrl+S** on **Simulator Pane** Output file will be downloaded if it's assembled.
3. **Editor Pane** supports **Ctrl+/** as comment shortcut
4. On pressing **Ctrl+D** on **Simulator Pane** stats file will be downloaded provided that execution is complete.

General Instructions To Run Locally

1. If you do not have a copy of **RISCV Heritage** clone the repo and checkout **GUI** branch. If you do have the project then go to **step 2**

```
git clone URL
```

```
cd riscv
```

1. Install all dependencies

```
npm install
```

1. Now to run the development server run the following command in terminal

```
npm run start:dev
```

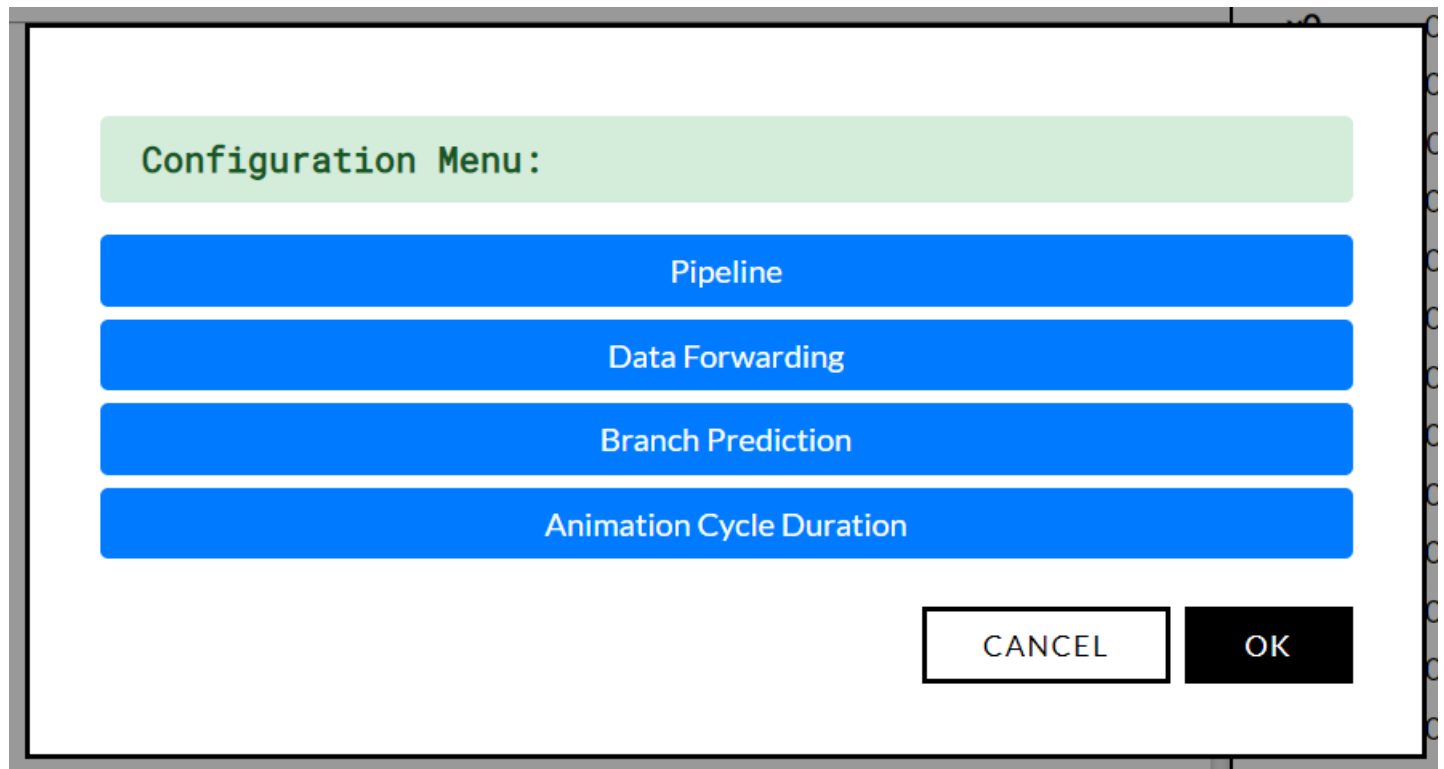
1. Now access the **RISCV Heritage** by visiting <http://localhost:1234/>
2. To build the project run the following command in terminal

```
npm run build
```

Functionalities Our Project Offers

1. **BreakPoint:-** Users can use a breakpoint for debugging. Users can select as many breakpoints as s(he) wants. During the execution of instructions, the program will stop after the execution of selected instructions in the order of program flow.
2. **Run:-** To run the complete program in a single go, users can use it. For pipelined version the *animation cycle duration* can be tuned by the user from the settings menu.
3. **Step:-** Run the program instruction wise(one instruction at a time).
4. **Stop:-** Stop the execution process using the stop button.
5. **Reset:-** Reset the current execution process. It will also reset the memory and registers state.
6. **Dump:-** Users can copy the machine code to clipboard using it.
7. **Cancel:-** Destroys the current execution and moves one step back.
8. **Settings:-** Execution configuration can be changed by clicking this button before assembling the code. If

assembled then it shows the configuration in READ ONLY mode.



- We provide our user with some options to enable/disable
 1. Pipelined execution
 2. Data Forwarding
 3. Branch prediction
 4. Animation Cycle Duration, to increase or decrease program execution speed for pipelined execution
 5. Each pipelined instruction execution passes through five stages, Fetch, Decode, ALU, Memory, and Write Back. All these stages are colour-coded. So when an instruction has completely executed in any stage, the instruction gets colour-coded with a colour corresponding to the pipeline stage.

PC	Machine Code	Original Code
0x30	0X00008067	jalr x0 x1 0
0x34	0XFF410113	addi x2 x2 -12
0x38	0X00812223	sw x8 4(x2)
0x3c	0X00112023	sw x1 0(x2)
0x40	0XFF40413	addi x8 x8 -1
0x44	0XFD1FF0EF	jal x1 fibo
0x48	0X00A00333	add x6 x0 x10
0x4c	0X00612423	sw x6 8(x2)
0x50	0X00412403	lw x8 4(x2)

Name	Value
CLOCK	19
RA	0x00000006
RB	0x00000010
inA	0x7FFFFFFE4
inB	0x00000000
RZ	0x7FFFFFFE4
RM	0x00000010
MDR	0x00000000
RY	0x00000000

Registers	Memory
x0	0x00000000
x1	0x00000010
x2	0x7FFFFFFE4
x3	0x00000001
x4	0x00000000
x5	0x00000001
x6	0x00000000
x7	0x00000000
x8	0x00000006
x9	0x00000000
x10	0x00000000
x11	0x00000000
x12	0x00000000
x13	0x00000000
x14	0x00000000
x15	0x00000000
x16	0x00000000

Display Settings: ☒ Hex ☐ Decimal ☐ ASCII

1. We have added a separate pane for pipelined execution through which the user can see the current status of pipelined ISA.

The screenshot shows the RISC-V Heritage Simulator interface. At the top, there are three tabs: Editor, Simulator (selected), and About. Below the tabs, there are several buttons: Run, Step, Reset, Dump, Stop, Cancel, and Settings. The main display area is divided into three sections: PC, Machine Code, and Original Code. The PC section shows the current instruction address (0x0). The Machine Code section shows the instruction (0X00600413). The Original Code section shows the instruction (addi x8 x0 6). The right side of the interface shows the Register File and Memory. The Register File shows the current values of the registers (x0 to x10). The Memory section shows the current state of memory (0x00000000 to 0x00000000).

1. To make the app more informative, whenever there is a stall, data forwarding, pipeline flushing or branch prediction etc. Our application shows a toast to notify such event.

The screenshot shows the RISC-V Heritage Simulator interface with a toast notification. The toast message is "0x44: Branch Prediction -> True". The main display area shows the pipeline state. The PC section shows the current instruction address (0x44). The Machine Code section shows the instruction (0XFD1FF0EF). The Original Code section shows the instruction (jal x1 fibo). The right side of the interface shows the Register File and Memory. The Register File shows the current values of the registers (x0 to x16). The Memory section shows the current state of memory (0x00000000 to 0x00000000). The Display Settings section shows the selected display format (Hex).

1. Memory Segment displays only those values into which something is explicitly written during program execution or in the data segment and all other values which are not shown are 0x00
2. Although code segment data is shown in memory it cannot be retrieved by the user, by accessing that location. This is to ensure that there is no structural hazard.
3. On any error, the program will alert the user and will not proceed further for execution
4. Any instruction which uses Double like sd, ld are invalid since it is 32-bit system
5. We have assumed that hazards are those dependencies which causes our pipeline to stall. So, any data dependency which are successfully handled by data forwarding won't be considered as a data hazard.

Latest Deployed version: [RISC-V Heritage](#)

