Indian Institute of Technology, Ropar

CS204 Course Project - Phase II

Team Members:

1. Aditya Agarwal - 2019CSB1064

2. Aneeket Mangal - 2019CSB1071

3. Fadia Het Rakeshkumar - 2019CSB1084

4. Shikhar Soni - 2019CSB1119

5. Tanmay Aeron - 2019CSB1124

Project Description

The aim of this project is to simulate the machine level execution of RISC V 32-bit instructions using a high level language.

The Project also aims to give updates to the user regarding each step of the execution of the program. It also returns the final status of the memory and registers as output for the user to analyse the working of their programs thoroughly.

The Project currently allows the user to use 29 different instructions and can be extended to allow the use of any number of instructions by editing the .csv files as long as the instructions are supported by 32-bit RISC V ISA.

The program supports the use of 'knobs' which will enable the user to select whether or not they want pipelining, data forwarding, and other details to be printed at the end of running the program.

The program executes each instruction using five stages as described in the RISC V architecture.

There are 5 separate knobs, all with a different purpose.

1. **Knob1:** Used to enable/disable pipelining during runtime.

2. **Knob2:** Used to enable/disable data forwarding

3. **Knob3:** Used to enable/disable printing values in the register file at the end of each cycle.

4. **Knob4:** Used to enable/disable printing information in the pipeline registers at the end of each cycle, along with the cycle number.

5. **Knob5:** This is like enabling knob4 for a specific instruction. We will be able to see pipeline register for that particular instruction only.

Technologies employed

Back-end - Python3

1. **pandas** for reading **.csv** files.

2. **os** for getting and adding path to certain file locations.

3. **collections.defaultdict** to make a hash map for memory.

4. **sys** for reading and editing files with ease.

5. **json** for crunching data

Front-end - Python3

1. **PyQt5** for the Graphic User Interface.

2. **qdarkstyle** for dark theme in the GUI.

How to run the code:

Instructions to run GUI version:

python main.py -g

After running this command in the terminal, the GUI will pop up.

You can then enable each knob by ticking the checkbox for each. For the fifth knob you can add the instruction number by adding the instruction in the box.

Information about the Fetch, Decode, etc. can be seen in the info tab on the right side when you click on the respective buttons in the DataPath tab

---

Instructions to run Non GUI version:

add '-k1' to enable knob1

add '-k2' to enable knob2

add '-k3' to enable knob3

add '-k4' to enable knob4

add '-k5 instruction number' to enable for that particular instruction

If you want to run a particular file, add that file in test folder and add '-f filename', otherwise main.mc is run

For example, if you want to enable knobs knob1, knob2 and knob3 and run fib.mc write

python main.py -k1 -k2 -k3 -f fib.mc

Other features:

Arguments: [-h] [-g] [-f F] [-k1] [-k2] [-k3] [-k4] [-k5 INS]

Description:

1. [-h, -help] show help message and exit

2. [-g, -gui] enable GUI

3. [-f F, -filename F] specify file which is to be run

4. [-k1, -knob1] enable Pipelining

5. [-k2, -knob2] enable Data Forwarding

6. [-k3, -knob3] show value in registerFile at end of each cycle

7. [-k4, -knob4] show value in Pipeline Registers at end of each cycle

8. [-k5 INS, -knob5 INS] show value in Pipeline Registers at end of each cycle for particular instruction
   All the arguments are optional.
   If no arguments are given, for example, python main.py then the main.mc file is run

Implementation Details

We have implemented the project using the following method:

First it takes the input from a **.mc** file that contains the input in the required format, the content of this file can be modified using the GUI editor or by changing the main.mc file.
Then, it stores the **.data** part into the data memory and the **.text** part in the text part of the memory. Once that is completed, we run the program using the following method:

1. Instruction Fetch: PC is incremented by 4, and the instruction is loaded from the memory and stored in the Instruction Register.

2. Instruction Decode: We are identifying the instruction using the .csv file and **pandas** library and returning all the required fields. The registers RA and RB are also set during this stage.

3. Execute: The ALU executes the instruction by computing the desired output using values stored in registers in RA, RB, imm, etc. The required type of ALU instruction is determined by another .csv file. We are updating the output in the RZ register.

4. Memory Access: Memory is read and written in this stage. This stage is used only for load and store instructions. For the remaining instructions, this stage is redundant. The value of the RY register is also consequently updated.

5. Register Write-back: Here we are storing the value of the temporary register RY in the destination register, resulting in the register files being updated. The register update is extremely fast compared to memory read and update.

We have implemented data forwarding by passing the required values from the source buffer to the destination buffer without the need of stalling in most cases. This helps to reduce the number of stalls required. The types of data forwarding are:

1. Execute to Execute

2. Memory to Execute

3. Execute to Execute

The program detects hazards present which can cause an incorrect value to be register due to one instruction's stage being executed before another. This is fixed by either transferring data values directly with the help of data forwarding, or with the help of stalling which delays the next instruction so that the previous one is updated in time.

While data forwarding is enabled, there is no use of stalling except for one particular case where we use it along with Memory-to-Execute data forwarding.

Input/Output format details

Input Format:

0x0 0x00500513
0x4 0x008000EF
0x8 0x0440006F
$
0x10000000 0x64

The input is given through GUI if the GUI is run else it is taken from the input file.

The section given before the '$' sign is the text segment of the code and all the lines after it signify the data segment of the code.

Output is generated in the 'generated' folder. It contains the following things

1. 'registers.txt' contains the register values

2. 'memory.txt' contains the data part of the memory

3. 'stats.txt' contains the statistical details of the program

4. 'outputLog.txt' contains the buffer data of each cycle

In *stats.txt* we have printed the following:

1. Total number of cycles

2. Total number of instructions executed

3. CPI

4. Data Transfer instructions executed

5. ALU instructions executed

6. Control instructions executed

7. Total Stall Count

8. Number of Data Hazards

9. Total Control Hazards

10. Total Branch misprediction

11. Number of stalls due to data hazards

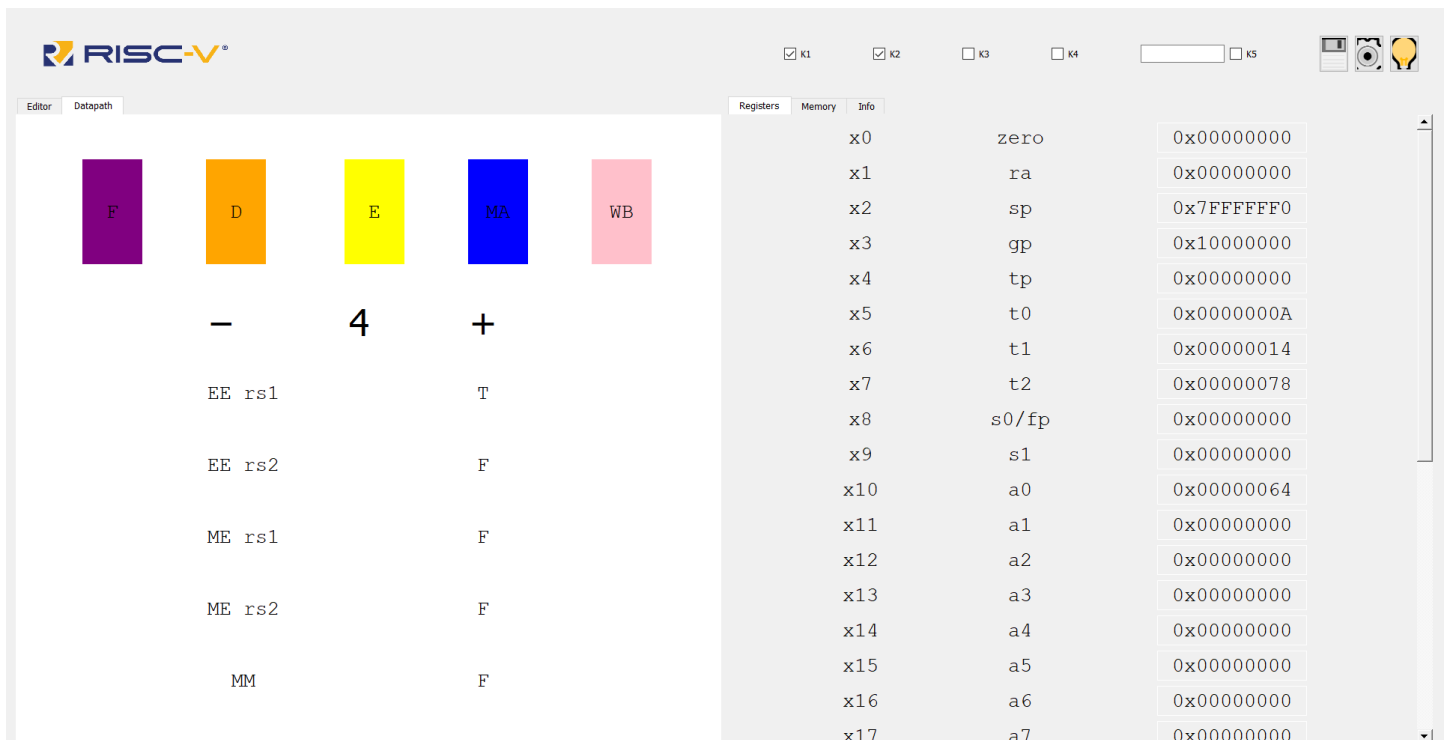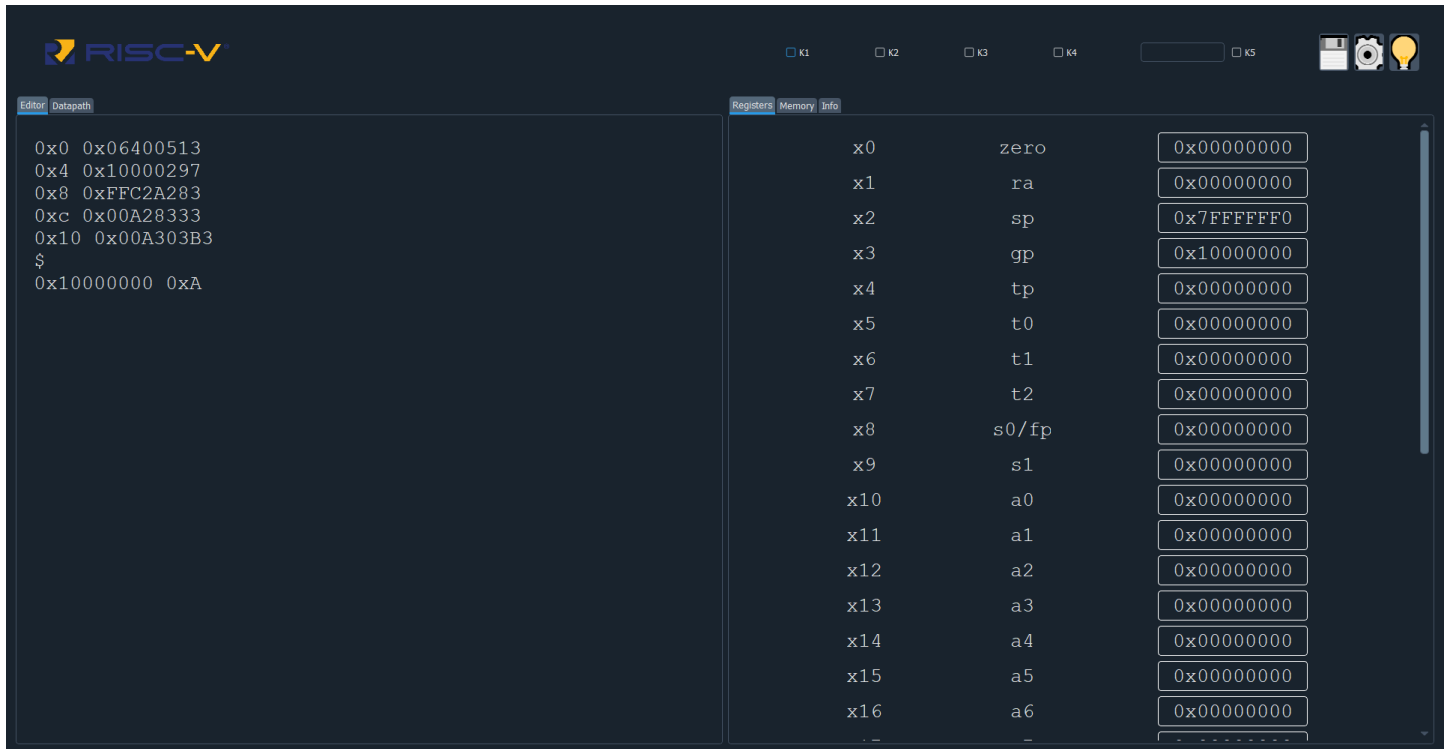12. Number of stalls due to control hazards

---

An example output for *stats.txt* is :

Total number of Cycles in the program :14
Total number of Instructions executed :4
CPI is :3.5
Data Transfer Instructions Executed :0
ALU instructions are :4
Contol instructions :0
Total Stall Count are :6
Number of Data Hazards are :3
Total Control Hazards are :0
Total Branch mispredictions are :0
Number of stalls due to data hazards are :6
Number of stalls due to control hazards are :0

The instruction fed was:
0x0 0x00A50513
0x4 0x00A50513
0x8 0x00A50513
0xC 0x00A50513

Graphical User Interface

Editor (Datapath)

```
0x0  0x06400513
0x4  0x10000297
0x8  0xFFC2A283
0xc  0x00A28333
0x10 0x00A303B3
$
0x10000000 0xA
```

K1 K2 K3 K4 K5

Editor | Datapath

Registers | Memory | Info

Reg	Name	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7FFFFFF0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x00000000
x6	t1	0x00000000
x7	t2	0x00000000
x8	s0/fp	0x00000000
x9	s1	0x00000000
x10	a0	0x00000000
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000

Datapath

K1 K2 K3 K4 K5

Editor | Datapath

F   D   E   MA   WB

−   4   +

EE rs1   T
EE rs2   F
ME rs1   F
ME rs2   F
MM   F

Registers | Memory | Info

Reg	Name	Value
x0	zero	0x00000000
x1	ra	0x00000000
x2	sp	0x7FFFFFF0
x3	gp	0x10000000
x4	tp	0x00000000
x5	t0	0x0000000A
x6	t1	0x00000014
x7	t2	0x00000078
x8	s0/fp	0x00000000
x9	s1	0x00000000
x10	a0	0x00000064
x11	a1	0x00000000
x12	a2	0x00000000
x13	a3	0x00000000
x14	a4	0x00000000
x15	a5	0x00000000
x16	a6	0x00000000
x17	a7	0x00000000

In the memory we can see the address and the value at each address. For example if the address is 0x1000000C then the address of each of the right box are 0x1000000C , 0x1000000D, 0x1000000E and 0x1000000F Here we can see the jump to register functionality, where we have stored a value in the register 0x10000500, and are jumping to it and seeing the value.

For example to jump on the location 0x10000500 write 10000500 in the Jump to box which is in the bottom right in the memory tab.

We have made the GUI in both light and dark theme. The theme can be changed by clicking on the "yellow bulb" which can be seen in the top right corner.

We have shown memory, registers and info on the right section of the GUI while on the left part there is Code Editor. In the Code Editor you can edit your code or copy paste your code and also save it. The info contains the details of the instruction being executed in each stage during a cycle. First of all we will paste the code in the Editor on the left side and then save and compile the code by clicking the respective 'save' button which is the first button on the top right and 'compile' button. The code will get updated in **main.mc** file and then the code will be executed and a ✓ will appear. The memory and registers will be updated simultaneously.

- We can jump to the memory using the 'Jump to' box on the bottom right corner in the memory section.

- All the registers can be seen in the register file by scrolling in the register file section.

- Executing **main.py** always generates a GUI window where you can code in the Editor.

- We can also execute non GUI code by writing→python main.py

- The DataPath explains how the data is flowing through each stage.

- Here the 5 stages can be seen in the DataPath tab.

- The number below it is the cycle number for which the DataPath is shown.

- You can see the DataPath in the respective cycle by going to that cycle.

- Click on + to go to the next cycle and on - to go to the previous cycle.