

README FILE (Phase 3)

Group-6

Group members:

1. Abhishek Choudhary (2019CSB1061)
2. Pradeep Kumar (2019CSB1107)
3. Nitish Goyal (2019CSB1103)
4. Himanshu Yadav (2019CSB1263)
5. Deepan Maitra (2019CSB1044)

Git-Hub link to the project: <https://github.com/Silent-faith/RISC-V-Simulator>

Welcome to the phase 3 of the RISC-V simulator. In this phase, we have tried to incorporate Instruction Cache and Data Cache. We have built on our initial Phase 2 code.

You can either use the Non-GUI version of the code (`non_gui_phase3.py`) or use the GUI version (`phase3gui.py`)

INPUT TO CODE: A file with the machine code (here it is `code.mc`). This file has the machine code divided into two segments: data segment and text segment.

0x0	0x10000197	Text segment (instructions)
0x4	0x0001A183	
0x8	0x10000217	
0xc	0xFFC22203	
0x10000008	0x17020010	Data segment
0x10000004	0x83A10100	
0x10000000	0x97010010	

OUTPUT OF CODE:

In console: Displays the **5 step execution** of each instruction, by displaying the **fields** (*opcode, rs, rd, func3, func7, imm*) of each instruction when applicable. Also, displays the **register values** and **memory state** before and after execution. The **Instruction and Data Cache** are shown. Also displays **number of hits and misses** of the cache.

In GUI simulator window: The register states and the Instruction and Data Cache contents can also be checked using the display GUI window. (To directly go to the process to use this simulator, jump to **HOW TO USE** part of this document)

After execution, all the data memory is written in the `memory.mc` file before the program terminates. (additional files `console.txt`, `datacache.txt`, `instructioncache.txt` and `register.mc` are used to facilitate the GUI operation)

INSTRUCTIONS SUPPORTED BY OUR CODE:

- R format - add, and, or, sll, slt, sra, srl, sub, xor, mul, div, rem
- I format - addi, andi, ori, lb, lh, lw, jalr
- S format - sb, sw, sh
- SB format - beq, bne, bge, blt
- U format - auipc, lui
- UJ format - jal
- This code ***DOES NOT*** support any pseudo instruction.

HOW TO RUN NON-GUI VERSION?

Step 1) Run `non_gui_phase3.py` on console.

Step 2) The user is asked for **Size, Block Size** (in bytes) and **Set Associativity** of both **Data and Instruction Caches**.

Step 2) The user is given a choice to determine the method of the execution of machine code:

- 1 for non pipelined execution
- 2 for pipelined execution without data forwarding
- 3 for pipelined execution with data forwarding

Step 3) After user enters 1, 2 or 3, the execution begins and output is shown on console.

HOW TO USE THE SIMULATOR?

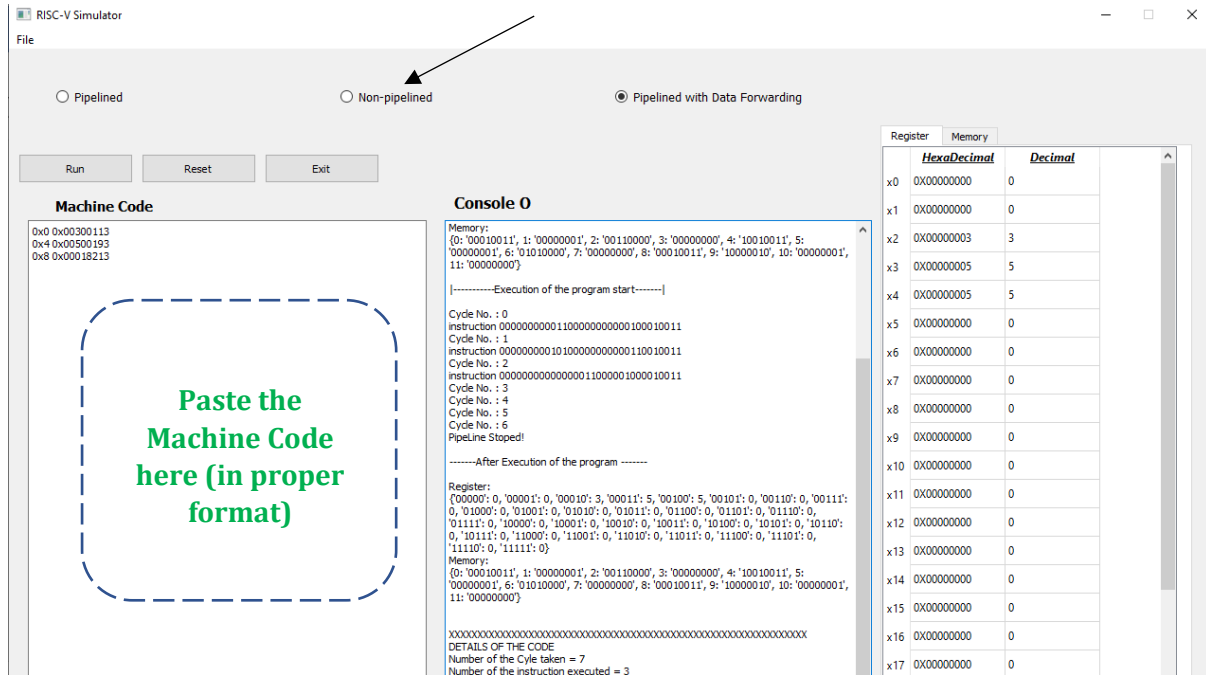
The Simulator has an in-built GUI feature to make the code more user-friendly. (Please install `bitstring` module and `PyQt5` module in Python before running this code)
The following steps will show how to use the simulator.

Step 1) Run `phase3gui.py` on the console. A GUI window will pop-up automatically.

Step 2) Paste the machine code from `code.mc` into the 'Machine Code' text field.

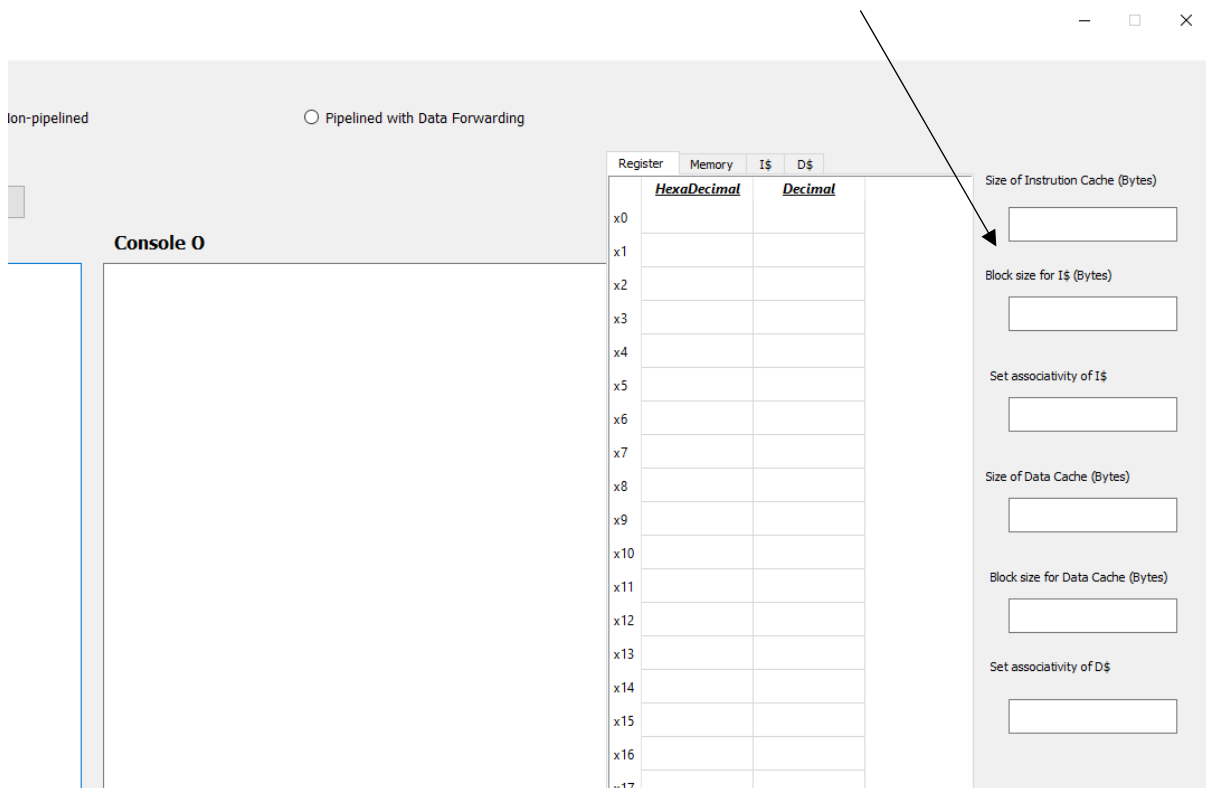
Step 3) Select appropriate checkbox for **Pipelined, Non-pipelined, Pipelined with Data Forwarding**.

Click on these Checkboxes to select pipelined/non-pipelined



Step 4) Put the **Size of Cache, Block Size (in bytes)** and **Set Associativity** of both **Data and Instruction Caches** in the text boxes.

Put all the sizes and values here



Step 5) **Double click** on RUN button to start execution.

Step 6) The Console text field will show the console output. The tables of **Register** and **Memory** will also show required values. The tables of **Instruction and Data caches (I\$ and D\$)** shows the current stages of Instruction and Data Cache.

Please Note: The first column of cache is the tag array (the block addresses). The next columns are the block values byte wise. Each ROW represents a BLOCK.

I\$ and D\$ shown here

The screenshot displays a simulation window with several components:

- Execution Mode:** Radio buttons for "Non-pipelined" and "Pipelined with Data Forwarding".
- Exit Button:** A button labeled "Exit".
- Console O:** A text area showing the state of the system before and after execution. It includes register values (e.g., 0x0 0x00300113, 0x4 0x00500193, 0x8 0x00018213), memory contents, and cache states.
- Register Table:** A table with 18 rows and 4 columns: Register, Memory, I\$, and D\$. The first column (Register) contains values 0, 4, 8, 12, etc. The second column (Memory) contains values 00010011, 10010011, etc. The third column (I\$) contains values 00000001, 00000001, etc. The fourth column (D\$) contains values 00000001, 10000010, etc.
- Cache Configuration:** On the right side, there are input fields for "Size of Instruction Cache (Bytes)" (128), "Block size for I\$ (Bytes)" (4), "Set associativity of I\$" (2), "Size of Data Cache (Bytes)" (64), "Block size for Data Cache (Bytes)" (4), and "Set associativity of D\$" (2).

A green arrow points from the text "I\$ and D\$ shown here" to the I\$ and D\$ columns of the Register table.