README FILE (Phase 2)

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 2 of the RISC-V simulator. In this phase, we have tried to incorporate pipelined execution (with or without data forwarding). We have built on our initial Phase 1 code.

You can either use the Non-GUI version of the code (`phase2_final_non_gui.py`) or use the GUI version (`phase2gui.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     |
0x4    0x0001A183     |    Text segment (instructions)
0x8    0x10000217     |
0xc    0xFFC22203     |
0x10000008  0x17020010    |
0x10000004  0x83A10100    |    Data segment
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. In order to check successful completion of the code, the user has to check necessary memory and register states at the end of execution.

In GUI simulator window: The register states 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` 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 `phase2_final_non_gui.py` on console.

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.

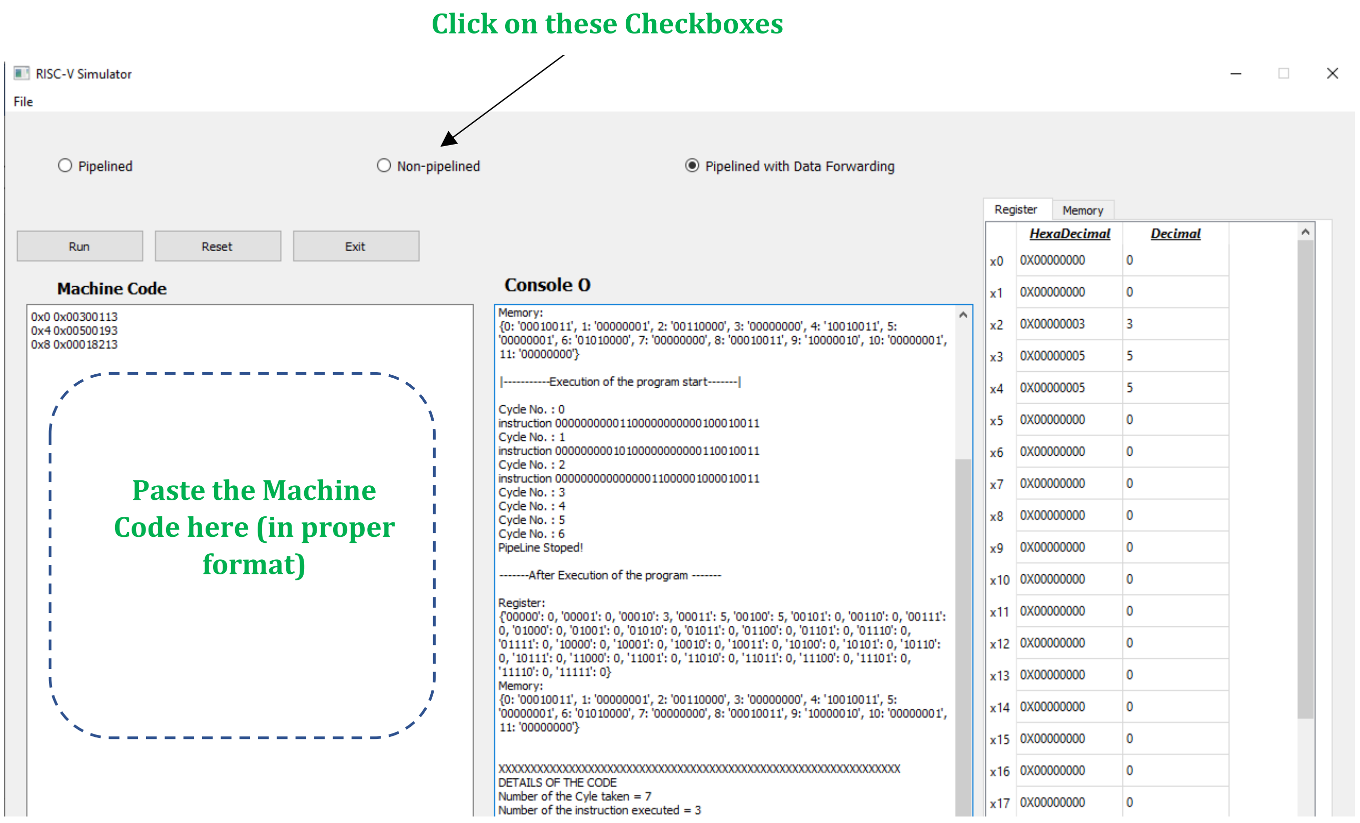
HOW TO USE THE SIMULATOR?

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 `phase2gui.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
RISC-V Simulator
File
Pipelined
Non-pipelined
Pipelined with Data Forwarding
Run
Reset
Exit
Machine Code
Paste the Machine Code here (in proper format)
Console 0
Register
Memory

Pipelined [✓ Pipelined with Data Forwarding] Non-pipelined

Machine Code

```
0x0 0x00300113
0x4 0x00500193
0x8 0x00018213
```

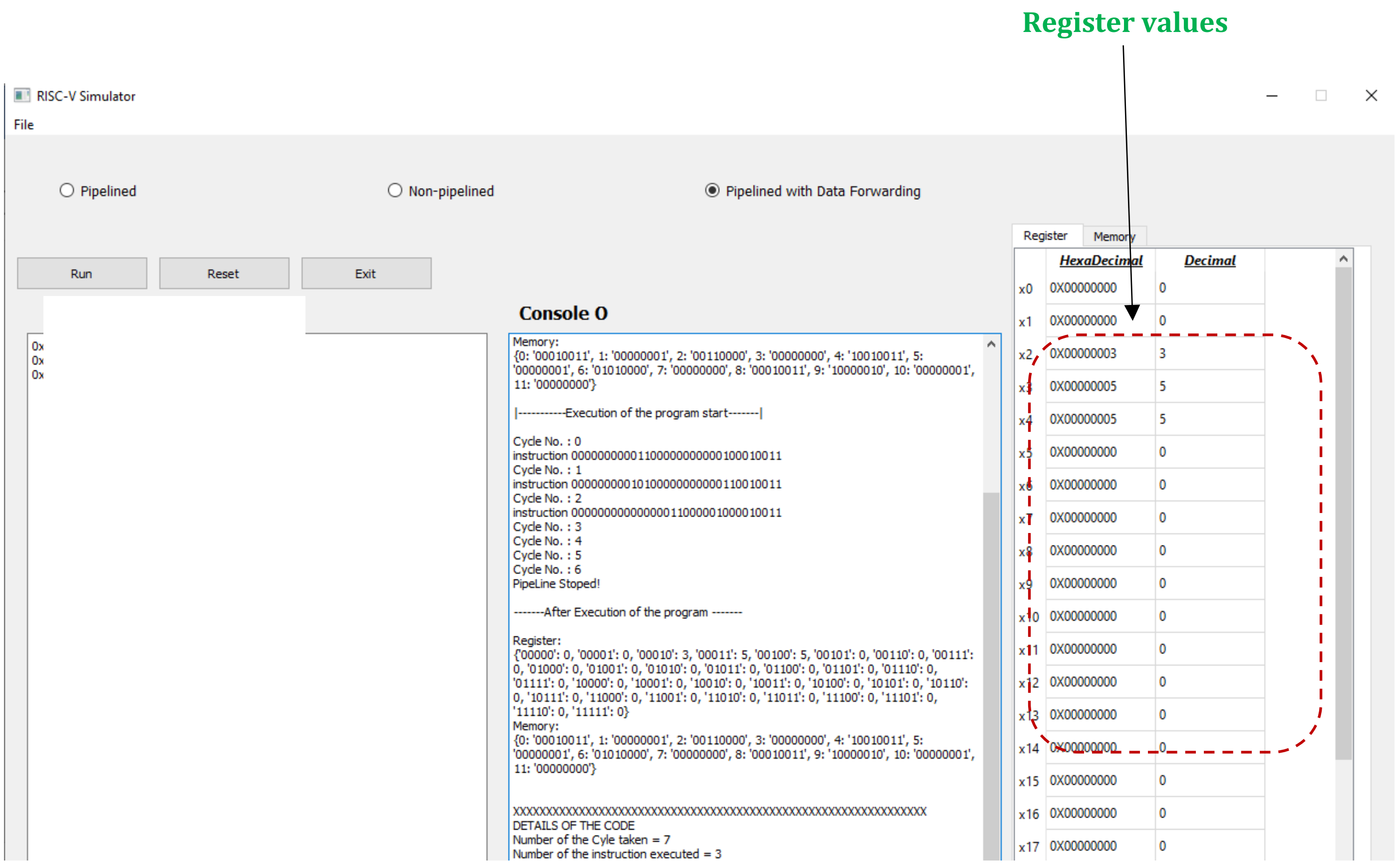
Console 0

```
Memory:
{0: '00010011', 1: '00000001', 2: '00110000', 3: '00000000', 4: '10010011', 5:
'00000001', 6: '01010000', 7: '00000000', 8: '00010011', 9: '10000010', 10: '00000001',
11: '00000000'}

|-----------Execution of the program start-------|

Cycle No. : 0
instruction 00000000001100000000000100010011
Cycle No. : 1
instruction 00000000010100000000000110010011
Cycle No. : 2
instruction 00000000000000011000001000010011
Cycle No. : 3
Cycle No. : 4
Cycle No. : 5
Cycle No. : 6
PipeLine Stoped!

-------After Execution of the program -------

Register:
{'00000': 0, '00001': 0, '00010': 3, '00011': 5, '00100': 5, '00101': 0, '00110': 0, '00111':
0, '01000': 0, '01001': 0, '01010': 0, '01011': 0, '01100': 0, '01101': 0, '01110': 0,
'01111': 0, '10000': 0, '10001': 0, '10010': 0, '10011': 0, '10100': 0, '10101': 0, '10110':
0, '10111': 0, '11000': 0, '11001': 0, '11010': 0, '11011': 0, '11100': 0, '11101': 0,
'11110': 0, '11111': 0}
Memory:
{0: '00010011', 1: '00000001', 2: '00110000', 3: '00000000', 4: '10010011', 5:
'00000001', 6: '01010000', 7: '00000000', 8: '00010011', 9: '10000010', 10: '00000001',
11: '00000000'}


XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
DETAILS OF THE CODE
Number of the Cyle taken = 7
Number of the instruction executed = 3
```

	HexaDecimal	*Decimal*
x0	0X00000000	0
x1	0X00000000	0
x2	0X00000003	3
x3	0X00000005	5
x4	0X00000005	5
x5	0X00000000	0
x6	0X00000000	0
x7	0X00000000	0
x8	0X00000000	0
x9	0X00000000	0
x10	0X00000000	0
x11	0X00000000	0
x12	0X00000000	0
x13	0X00000000	0
x14	0X00000000	0
x15	0X00000000	0
x16	0X00000000	0
x17	0X00000000	0

Step 4) **Double click** on RUN button to start execution.

Step 5) The Console text field will show the console output. The tables of Register and Memory will also show required values.



Pipelined [✓ Pipelined with Data Forwarding] Non-pipelined

```
0x
0x
0x
```

Console 0

```
Memory:
{0: '00010011', 1: '00000001', 2: '00110000', 3: '00000000', 4: '10010011', 5:
'00000001', 6: '01010000', 7: '00000000', 8: '00010011', 9: '10000010', 10: '00000001',
11: '00000000'}

|-----------Execution of the program start-------|

Cycle No. : 0
instruction 00000000001100000000000100010011
Cycle No. : 1
instruction 00000000010100000000000110010011
Cycle No. : 2
instruction 00000000000000011000001000010011
Cycle No. : 3
Cycle No. : 4
Cycle No. : 5
Cycle No. : 6
PipeLine Stoped!

-------After Execution of the program -------

Register:
{'00000': 0, '00001': 0, '00010': 3, '00011': 5, '00100': 5, '00101': 0, '00110': 0, '00111':
0, '01000': 0, '01001': 0, '01010': 0, '01011': 0, '01100': 0, '01101': 0, '01110': 0,
'01111': 0, '10000': 0, '10001': 0, '10010': 0, '10011': 0, '10100': 0, '10101': 0, '10110':
0, '10111': 0, '11000': 0, '11001': 0, '11010': 0, '11011': 0, '11100': 0, '11101': 0,
'11110': 0, '11111': 0}
Memory:
{0: '00010011', 1: '00000001', 2: '00110000', 3: '00000000', 4: '10010011', 5:
'00000001', 6: '01010000', 7: '00000000', 8: '00010011', 9: '10000010', 10: '00000001',
11: '00000000'}


XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
DETAILS OF THE CODE
Number of the Cyle taken = 7
Number of the instruction executed = 3
```

	HexaDecimal	*Decimal*
x0	0X00000000	0
x1	0X00000000	0
x2	0X00000003	3
x3	0X00000005	5
x4	0X00000005	5
x5	0X00000000	0
x6	0X00000000	0
x7	0X00000000	0
x8	0X00000000	0
x9	0X00000000	0
x10	0X00000000	0
x11	0X00000000	0
x12	0X00000000	0
x13	0X00000000	0
x14	0X00000000	0
x15	0X00000000	0
x16	0X00000000	0
x17	0X00000000	0