ECE GY 6913 Computing Systems Architecture

RISCV Simulator Project – Phase 2
Fall 2025

Goal of the Project

- Have a working model of a 32 bit RISC-V interpreter.
- Accurately maintain the values in all registers and memory locations.
- Project is to be done either in Python or C++. The boiler-plate code will been provided on brightspace.
- Project is to be done individually.

Grading Scheme and support

- Weightage of the RISCV project will be 10% of your total grade.
- Each phase (Single stage and Five stage) will be out of 50 marks
- Test cases – total 10 – 3 of which will be released a week before submission of each phase.
- Project report – contains questions and marking scheme which will be posted on Brightspace
- Mondays 9:30 AM – 11:00 AM will be project office hours starting 6th Oct 2025
- Phase 1 submission is on 7th Nov 2025
- Phase 2 submission is on 12th Dec 2025

Input

imem.txt

contains binary input of all the instructions

dmem.txt

contains binary input of all the memory locations

Output

Performance Metrics

Register States after each cycle

DMem at the end of program execution

State Results

What needs to be done in IF

Read 4 lines of the IMEM file.

What needs to be done in ID

Convert the 32 bits into an instruction. (Big endian)

```
00000000

01010010

00000001

10110011
```

Instruction: `00000000010100100000000110110011`

Decoded instruction: `add x3, x4, x5`

What needs to be done in EX

Instruction: `add x3, x4, x5`

`lw x10, 20(x12)`

Be sure to perform the right execution at this stage.

R and I type instructions will perform the normal executions.

Load and Store instructions will perform calculations of offset.

What needs to be done in MEM

Make sure to access memory in this stage.

LOAD and STORE instructions

What needs to be done in WB

Update the values of registers in this stage.

Eg. loading a value into a register, arithmetic result to be written into register

Instructions to be implemented

Please refer Brightspace Project document which contains all instructions that need to be implemented.

Hazards – EX to 1st

add x5, x2, x3
add x6, x5, x4

1	2	3	4	5	6
IF	ID	EX	ME	WB	
	IF	ID	EX	ME	WB

Hazards – EX to 2nd

```
add x5, x2, x3
sub x7, x8, x9
add x6, x5, x4
```

1	2	3	4	5	6	7
IF	ID	EX	ME	$\boxed{\text{WB}}$		
	IF	ID	EX	ME	WB	
		IF	ID	$\boxed{\text{EX}}$	ME	WB

MEM to 2nd

```
lw x5, 20(x10)
sub x7, x8, x9
add x6, x5, x4
```

1	2	3	4	5	6	7
IF	ID	EX	ME	WB		
	IF	ID	EX	ME	WB	
		IF	ID	EX	ME	WB

MEM to 1st

lw x5, 20(x10)
add x6, x5, x4

1	2	3	4	5	6	
IF	ID	EX	ME	WB		
	IF	ID	EX	ME	WB	

1	2	3	4	5	6	7
IF	ID	EX	ME	WB		
	IF	ID	NOP	EX	ME	WB

Cannot forward since they are both in the same cycle.

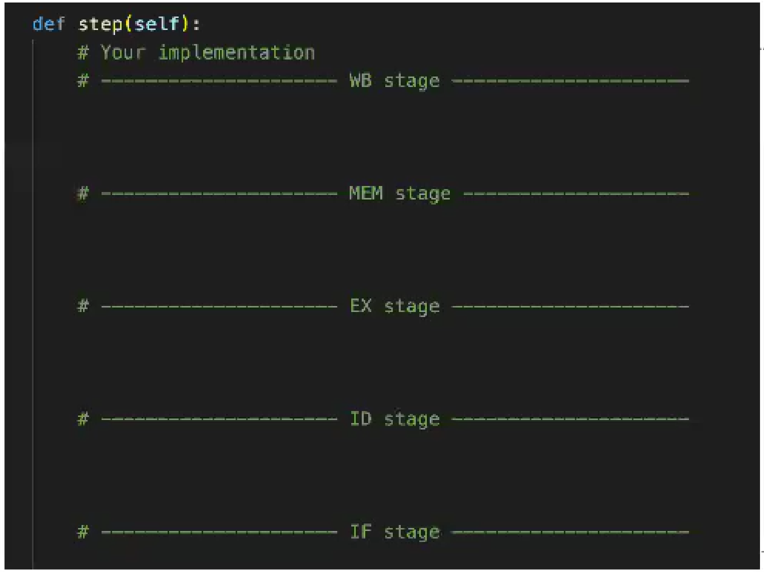
Use NOP

NOP halts one stage of the pipeline

Test Case 1

	0	1	2	3	4	5	6		
Lw	IF	ID	EX	ME	WB				
Lw		IF	ID	EX	ME	WB			
Add			IF	ID	----	EX	MEM	WB	
Sub				IF	----	ID	EX	MEM	WB
SW					----	IF	ID	EX	MEM

Why are the stages in reverse order in code?



```
def step(self):
    # Your implementation
    # --------------------- WB stage ---------------------


    # --------------------- MEM stage --------------------


    # --------------------- EX stage ---------------------


    # --------------------- ID stage ---------------------


    # --------------------- IF stage ---------------------
```

If implementing a NOP say in the EX stage, every lower state (before the EX stage) will be in NOP till the upper state NOP is resolved.

Essentially the lower states are affected by the upper states.

Common Question – Why extra cycle?

- There is always an extra cycle at the end after If.nop is set to true
- Refer the code to see how self.halted is set to true in the code
- Checkout where our print statements for each cycle are.
- Simple example:

```
ADD x2, x3, x1
HALT
```

Number of instructions = 2, cycles = 3

```
State after executing cycle: 0
IF.PC: 4
IF.nop: False
------------------------------------------------
------------------------
State after executing cycle: 1
IF.PC: 8
IF.nop: True
------------------------------------------------
------------------------
State after executing cycle: 2
IF.PC: 8
IF.nop: True
```