Computer Architecture 2025 Fall Lab 1

1. Problem Description

In this homework, we are going to implement a simple RISC-V CPU with Verilog.

- The CPU is **single-cycle**: all operations are assumed to be completed equally in one cycle.
- 32 registers, 1KB instruction memory.
- Input: 32-bit binary codes.
- Scope: Only arithmetic operations (no load/store).
- **Evaluation: dump register values after each cycle.**

1.1 Data Path

- Figure 1 describes the CPU data path.
- **Provided by TAs: Registers, PC, Instruction Memory.**
- **You may implement** the modules listed in Section 1.6, and integrate them in `CPU.v,` it will be easier if you follow Figure 1

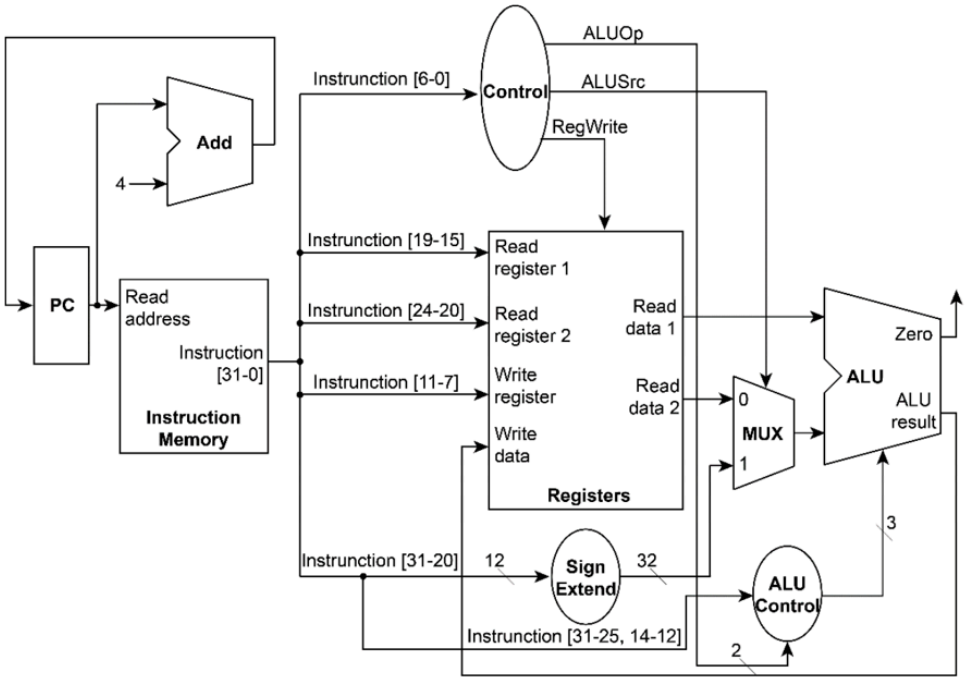

Figure 1 Data path of the CPU in this homework

1.2 Supported Instructions

- `and rd, rs1, rs2` (bitwise AND)

- `xor rd, rs1, rs2` (bitwise XOR)
- `sll rd, rs1, rs2` (logical shift left, use rs2[4:0])
- `add rd, rs1, rs2` (addition)
- `sub rd, rs1, rs2` (subtraction)
- `mul rd, rs1, rs2` (multiplication)
- `addi rd, rs1, imm` (addition with sign-extended imm)
- `srai rd, rs1, imm` (arithmetic right shift, imm[4:0])
- **EOF** (all-0 instruction, tb terminates simulation immediately)

1.3 Instruction Format

32-bit RISC-V standard.

31:25	24:20	19:15	14:12	11:7	6:0	
funct7	rs2	rs1	funct3	rd	opcode	function
0000000	rs2	rs1	111	rd	0110011	and
0000000	rs2	rs1	100	rd	0110011	xor
0000000	rs2	rs1	001	rd	0110011	sll
0000000	rs2	rs1	000	rd	0110011	add
0100000	rs2	rs1	000	rd	0110011	sub
0000001	rs2	rs1	000	rd	0110011	mul
imm[11:0]		rs1	000	rd	0010011	addi
0100000	imm[4:0]	rs1	101	rd	0010011	srai
0000000	00000	00000	000	00000	0000000	EOF

1.4 Input / Output Format

- Provided: `testbench.v`, `instruction.txt`.
- Compile all together. Example:

  ```
  iverilog -g2012 -o CPU.out ./tb/*.v ./supplied/*.v ./src/*.v
  (iverilog -g2012 -o cpu.out <tb::path>/*.v <supplied::path>/*.v <source::path>/*.v)
  ```

- `instruction.txt`: one 32-bit ASCII binary string per line.
  - Should be placed at the same directory as CPU.out
  - Underscores `_` and `// comments` could be ignored.
- `output.txt`: dump of registers after each cycle (generated from tb automatically).

1.5 Required Modules (You may integrate it)

- **Control**: decode opcode → control signals.
- **ALU Control**: decode funct7/funct3 → ALU operation.
- **Sign Extend**: 12-bit → 32-bit signed.
- **ALU**: perform arithmetic/logic operations.
- **CPU**: integrate all modules.
- Provided (not modifiable): PC, Instruction Memory, Registers, testbench.

1.6 Synthesizability and Area

- All RTL code must be synthesizable (no unintended latches).
- The TA will check your design using **yosys** (`synth.log`). You do not need to modify any yosys scripts. The results can be found in Section 10.6 of `./log/synth.log`, including the estimated area of your module.
- run parse.py to get area and latch results, paste the results, and fill the number in the report

```
=== Module Areas ===
ALU: 4284.462
ALU_Control: 31.920
Control: 10.374
PC: 170.240
Registers: 9447.256
====================
Total area: 13944.252

=== Latch Statistics (10.6 PROC_DLATCH) ===
No latch inferred count : 3
Latch inferred count    : 1
```

1.7 Testcases

- TAs will test your modules with **public** and **hidden** testcases.
- You may implement a Python simulator for debugging. The TA will use their own simulators to generate test cases from C programs and compile them into the corresponding assembly code.
- The TA will compare the output of your testbench (TB) with the expected output. They must match **exactly (100%)** in order to receive credit, so do not modify the provided testbench if you want to simulate TA's checking process.

1.8 Reminder

Lab 2 and 3 will be strongly related to this Lab. Please make sure you can fully understand how to write this homework; otherwise you may encounter difficulties in your future labs. Also, because this homework is rather simple, it is recommended for you to get familiar to waveform visualization tool (e.g. gtkwave), which is much more useful than "display method". You may not have enough time to learn this tool at the labs, or hard to learn it from scratch because of the difficulty of the labs.

2.1 Report

- **Explain modules** in plain language (not Verilog).
- Example (acceptable): describe PC behavior with reset/clk/start.

> PC module reads clock signals, reset bit, start bit, and next cycle PC as input, and outputs the PC of current cycle. This module changes its internal register "pc_o" at positive edge of clock signal. When reset signal is set, PC is reset to 0. And PC will only be updated by next PC when start bit is on.

- Example (unacceptable): just pasting Verilog code. (0 point)

> The inputs of PC are clk_i, rst_i, pc_i, and ouput pc_o.
> It works as follows:
>
> ```
> always@(posedge clk_i or negedge rst_i) begin
>     if(~rst_i) begin
>         pc_o <= 32'b0;
>     end
>     else begin
>             pc_o <= pc_i;
>     end
> end
> ```

- Report also includes:
  - Synthesizability / latch check screenshot.
  - Development environment (OS, compiler, IDE).

3.1 Development Environment

Requirement

- **Docker** is required for TA's evaluation process.
- Iverilog & Yosys can be installed without docker, you can refer the installation tutorial from dockerfile or tutorial we provided.

Directory Structure (after unzip)

```
Lab1/
├── Lab1_spec.pdf
├── Lab1_slide.pdf
├── Lab1/
│   ├── Makefile
│   ├── code/
│   │   ├── src/          # <Implement your CPU here>
│   │   ├── supplied/     # <supplied codes, don't modify them>
│   │   └── tb/           # <you may add debug print here>
```

```
│   ├── docker-compose.yml
│   ├── dockerfile
│   ├── judge.yaml
│   ├── parse.py            # calculate area from ./log/synth.log
│   ├── testcases/          # <sample testcases>
│   └── log/                # <logs after execution>
```

💡 **You should modify files in** `code/src/`

Do not change anything in `code/supplied/` `code/tb/`

3.2 Run with Docker

After implementation, execute:

```
sudo make run
```

- Results (logs, text, waveforms) will be generated under `log/`.

3.3 Run without Docker (alternative)

If you don't have Docker, run on **Ubuntu 22.04** with **iverilog 11.0-1.1 & Yosys & nangate45**

```
# 模擬
cp testcases/instruction_n.txt instruction.txt   # tb 會去讀 instruction.txt
iverilog -g2012 -o CPU.out ./tb/*.v ./supplied/*.v ./src/*.v
(iverilog -g2012 -o cpu.out <tb::path>/*.v <supplied::path>/*.v <source::path>/*.v)

diff -u output.txt testcases/output_n.txt       # 會去比較 tb 產生的 output.txt

# 合成
yosys -p "
 read_verilog and2.sv;
 hierarchy -top and2;    #要改成 Top module 名字: CPU
 proc; opt; fsm; opt; memory; opt;
 techmap; opt;
 #make sure nangate libraray path is correct
   dfflibmap -liberty ~/nangate45/NanGate45/lib/NangateOpenCellLibrary_typical.lib
   abc       -liberty ~/nangate45/NanGate45/lib/NangateOpenCellLibrary_typical.lib
   stat      -liberty ~/nangate45/NanGate45/lib/NangateOpenCellLibrary_typical.lib
 write_verilog -noattr mapped.v
"
```

```
# area results
python parse.py
```

⚠️ **Note:** You will get **0 points** if your code cannot run correctly on the official environmen

---

4.1 Submission Rules

- Directory structure after unzip:

```
studentID_lab1/    ( use lowercase: b, d... )
  ├── src/   (all Verilog codes you wrote)
  ├── studentID_lab1_report.pdf
```

- Do **not** include: Instruction_Memory.v, PC.v, Registers.v, testbench.v, instruction.txt, output.txt.
- File name: `studentID_lab1.zip`.
- Submit via NTU COOL.
- Deadline: 2025/11/04 (Tue.) 14:20

5.1 Evaluation Criteria

Submitting on time and checking the content of the submission are your responsibility. There will be no exceptions except under special circumstances.

- **Programming (80%)**
  - Public testcases: 40% (5 × 8%)
  - Hidden testcases: 40% (5 × 8%)
  - Not synthesizable: −10%
  - Compilation error (naming, or minor errors in 3 lines): −5%
- **Report (20%)**
  - Latch & Area results (10%)
  - Description (10%)
- **Other penalties**:
  - Plagiarism: get 0 in this homework, and TA will inform teacher
  - Submitting unnecessary files (tb or supplied): -5%
  - Late submission: -10% per day, at most a week
  - Major mistakes causing compilation error → programming part 0. (Judeged by TA)
  - Wrong directory format: −5%.

Contact

Questions: email **eclab.ca.ta@gmail.com**
