Computer Architecture Fall 2025

Programming Assignment 2

- **Deadline**: 23:59, 2025-12-08 (no late submissions allowed) **(extended by 1 week)**
- **Submission**: via NTU COOL
- **Contact**: If you have any questions, please email the TAs with the subject "[CA2025 PA2]".
  - Circle: r14943176@ntu.edu.tw
  - Alex: r14943141@ntu.edu.tw

General Description

In this assignment, you will implement a **RISC-V CPU** using **Chisel**, a hardware construction language (HCL) based on **Scala** (ref. https://www.chisel-lang.org/docs/category/getting-started).

The set of **instructions** you must support, along with their corresponding **machine code formats**, is listed at the end of this document in **List of Instructions to Support**.

You may choose to design your CPU as **either a single-cycle** or a **five-stage pipelined** processor. Please note that your **final grade will be partially evaluated based on the cycle count needed for executing each program**. The detailed grading criteria can be found in the **Grading Policy** section below.

You will receive a project folder named **`CA2025_PA2.tgz`**, which contains the following file structure:

```
CA2025_PA2/
│
├── project/
│   └── build.properties
├── src/
│   ├── main/scala/cpu
│   │   ├── Core.scala
│   │   └── (other design files)
│   └── test/
│       ├── scala/cpu
│       │   └── CoreTest.scala
│       └── pattern/
│           ├── p1.hex
│           └── (other test patterns)
└── build.sbt
```

Some pre-defined modules are provided for your convenience.

Note: Do not modify the following modules: `DataMem` , `InstMem` , `CoreTest` and the I/O ports of `Core`

Other than these modules, you are free to:

- Modify existing modules
- Add new modules
- Change or extend I/O interfaces as needed

Environmental Setup

1. Prerequisites

Install **JDK 17** and **sbt** (Scala Build Tool)

You can verify your installations using the commands:

```
java -version
sbt --version
```

2. Compilation and Testing

Inside the project directory, run the following commands. You can replace the values and file paths as needed:

```
PIPELINE=0 INST_FILE=pattern/p1.hex GOLDEN_FILE=pattern/p1_golden.hex sbt test
```

3. Waveforms

Generated `.vcd` files are in `test_run_dir/`.

```
gtkwave test_run_dir/<test_name>/<module_name>.vcd
```

Grading Policy

There are **5 public patterns** and **5 hidden patterns** in total, each designed to test different **hazard scenarios** that may occur in your CPU implementation.

Your grade will be based on implementation correctness and performance, evaluated in terms of the used cycle count.

You must pass all test patterns to receive any score. After that, your final score will depend on whether your CPU meets the baseline performance or achieves the bonus performance standard.

In the following table, *sc* stands for single-cycle; *pipe* stands for pipeline.

	hazard pattern	baseline	points	bonus	points
p1	no hazard	sc: 45 + 4 nop	6	pipe: 50	4
p2	data (ALU)	sc: 20 + 4 nop, pipe: 50	6	pipe: 25	4
p3	data (load-use)	sc: 31 + 4 nop, pipe: 50	6	pipe: 41	4
p4	control (branch, jump)	sc: 31 + 4 nop, pipe: 58	6	pipe: 48	4

	hazard pattern	baseline	points	bonus	points
p5	general	sc: 55 + 4 nop, pipe: 108	6	pipe: 72	4

Submission Guidelines

Please follow the file structure below when submitting your homework.

```
<studentID>_PA2/
│
├── Core.scala
└── (other design files)
```

- Only submit the files under main/scala/cpu.
- **Do not modify** any parts marked with "DO NOT MODIFY"**, or you will receive a penalty.**
- The top-level folder must be named with your student ID (first letter **capitalized**).
- Compress the entire folder into a **`.tgz`** file and submit it via NTU Cool.

List of Instructions to Support

	Instr.	Type	Reg. Transfer	Description
arith.	`add`	R	`x[rd] = x[rs1] + x[rs2]`	Add
	`sub`	R	`x[rd] = x[rs1] - x[rs2]`	Sub
	`and`	R	`x[rd] = x[rs1] & x[rs2]`	And
	`or`	R	`x[rd] = x[rs1] | x[rs2]`	Or
	`xor`	R	`x[rd] = x[rs1] ^ x[rs2]`	Xor
	`slt`	R	`x[rd] = (x[rs1] < x[rs2]) ? 1 : 0`	Set Less Than (Signed)
	`sll`	R	`x[rd] = x[rs1] << (x[rs2][4:0])`	Shift Left Logical
	`srl`	R	`x[rd] = x[rs1] >> (x[rs2][4:0])`	Shift Right Logical
	`sra`	R	`x[rd] = x[rs1] >> (x[rs2][4:0])`	Shift Right Arithmetic
immed.	`addi`	I	`x[rd] = x[rs1] + imm`	Add Immediate
	`slti`	I	`x[rd] = (x[rs1] < imm) ? 1 : 0`	Set Less Than Immediate (Signed)
	`slli`	I	`x[rd] = x[rs1] << shamt`	Shift Left Logical Immediate
	`srli`	I	`x[rd] = x[rs1] >> shamt`	Shift Right Logical Immediate
	`srai`	I	`x[rd] = x[rs1] >> shamt`	Shift Right Arithmetic Immediate
load	`lw`	I	`x[rd] = MEM[x[rs1] + imm]`	Load Word
store	`sw`	S	`MEM[x[rs1] + imm] = x[rs2]`	Store Word
branch	`beq`	B	`if (x[rs1] == x[rs2]) PC += imm`	Branch Equal
	`bne`	B	`if (x[rs1] != x[rs2]) PC += imm`	Branch Not Equal
	`blt`	B	`if (x[rs1] < x[rs2]) PC += imm`	Branch Less Than (Signed)
	`bge`	B	`if (x[rs1] >= x[rs2]) PC += imm`	Branch Greater/Equal (Signed)
jump	`jal`	J	`x[rd] = PC + 4; PC += imm`	Jump and Link
	`jalr`	I	`x[rd] = PC + 4; PC = (x[rs1] + imm) & ~1`	Jump and Link Register

	Instr.	Type	Reg. Transfer	Description
U-type	`lui`	U	`x[rd] = imm << 12`	Load Upper Immediate
	`auipc`	U	`x[rd] = PC + (imm << 12)`	Add Upper Immediate to PC

- References:
  - RISC-V Instruction Set Specifications
  - RISC-V Card