Department of Computer Science
National Tsing Hua University
EECS403000 Computer Architecture
Spring, 2023, Homework 5
Due date: 5/28/2023 23:59

In this assignment, we will use a RISC-V pipeline simulator (Ripes) to observe the working of the pipeline. Please set up the Ripes before doing this assignment. After finishing your homework, please submit your assembly code and report by following the requested format on eLearn.

● Environment Setup

We will use a RISC-V pipeline simulator (Ripes) to observe the working of the pipeline. Before starting, let's install the tool and test it. There are different versions of the setup guide (in the `setup` folder) for different operating systems (Linux, macOS, Windows 10/11).

● Assignment

All the work, including source code and report, should be done by yourself. It is not allowed to use any tool or compiler to generate your source code. And plagiarism is strictly forbidden.

1. Assembly Coding (60%)

Please refer to `cmul.S` to (1) write your own assembly code (each testcase 25%) and (2) draw a flowchart of your code with the evidence of correct results in your report (10%). The code function assigned to you is based on the last digit of your student ID. Please implement your code base on the template in the individual folder. There are also reference links or reference CPP files in each folder.

- Task A: `MJ_template.S`: the last digits of `0, 1`
- Task B: `PSSS_template.S`: the last digits of `2, 3`
- Task C: `CNPT_template.S`: the last digits of `4 ,5, 6`
- Task D: `AFROG1_template.S`: the last digits of `7, 8, 9`

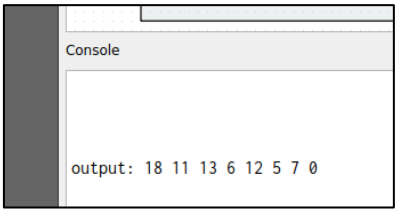
Please run your program with Ripes. Try to familiarize yourself with the Ripes simulator before doing this part: `https://github.com/mortbopet/Ripes/blob/master/docs/introduction.md`
Please make sure your code can pass the two testcases we provide and **print out the correct results** (listed in the comments of each template). For example, you must complete Task B and print out

```
output: 12 9 8 5 7 4 3 0
```

for the testcase 1, and

```
output: 5 2 3 0
```

for the testcase 2 if your student ID ends with '2' or '3'.
In your report, you must show the evidence of correct results with the screenshots of simulation results, and provide the flowchart of your implementation with precise explanation. For example, here is a sample simulation result on the Ripes' GUI:



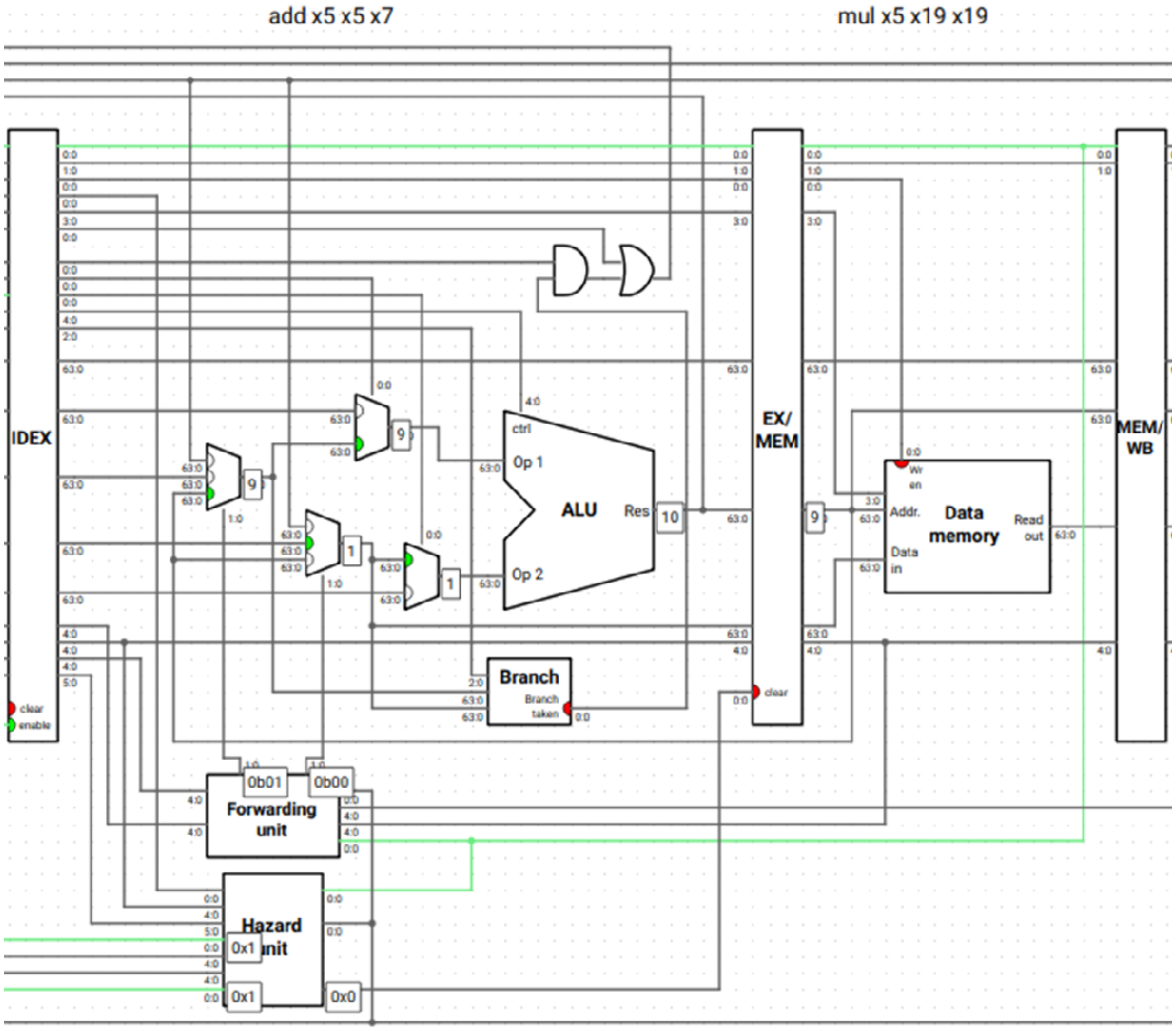
2. Hazards in Your Code (40%)

Please find in your code the following dependency types and control hazard:
Type (1): R-types RAW (read after write) at the following 1st instruction.
Type (2): R-types RAW at the following 2nd instruction.
Type (3): Load RAW at the following 1st instruction.
Type (4): Load RAW at the following 2nd instruction.
Type (5): Branch instruction (control hazard).
Run your code cycle by cycle with the 5-stage pipeline processor in Ripes. If these dependencies happen in your code, record the corresponding pipeline stages when the dependency happens. Observe and explain

how Ripes handles it by using the screenshot.
Note: There can be many cases of the same dependency type in your code. Only one example of each type should be noted. If there is no such type of dependency (which is rare), write a simple code example instead. Remember that you will not get the credit if you fail to locate a specific dependency type in your code, even if you write a simple code example instead.
For example, we report a Type (1) dependency (i.e., R-types RAW at the following 1st instruction) as follows:

```
30      mul t0, s3, s3
31      add t0, t0, t2
```

Show the code segment with line numbers in the report. The register `t0` is used to store the result of multiplying `s3` and `s3` in the first instruction. However, `t0` is also used in the next instruction `add` as rs1.



Assume that in our example, `s3 = 3` and `t2 = 0`. In Ripes, the processor detects dependency on MEM and EX stages for Type (1). Then, the Forwarding unit sets the control signal (`0b01`) of MUX before rs1 to select the `t0` value forwarded from the MEM stage to the EX stage.

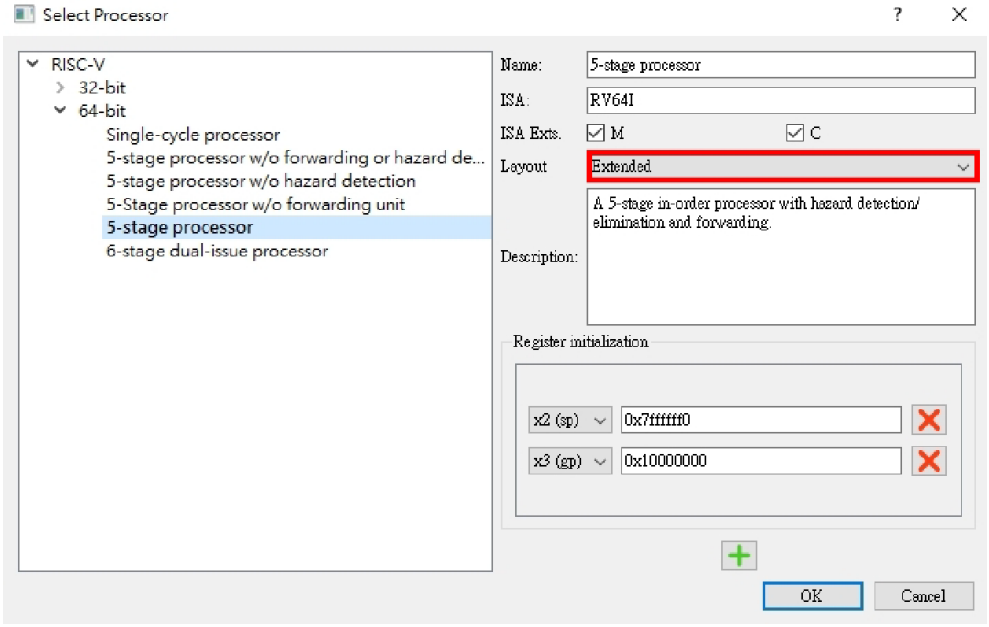
Note:

1. You must use **RV64i base integer instructions only** to compose the code (refer to `RV64i_Base_Integer_Instructions.pdf`).
2. For the EX hazard, the MUX control of ForwardA is (`0b01`) in Ripes, which is different from the definition in lecture notes and textbook (ForwardA=10); for the MEM hazard, the MUX control of ForwardA is (`0b10`) is in Ripes, which is different from the definition in lecture notes and textbook (ForwardA=01). **Please follow Ripes' definitions in this homework.** As a result, the `rs1` of ALU will be the result of the preceding instruction. You should take a screenshot of your observation and explain the details.
3. The Ripes pipeline simulator detects control hazards at the EX stage. By contrast, the control hazard

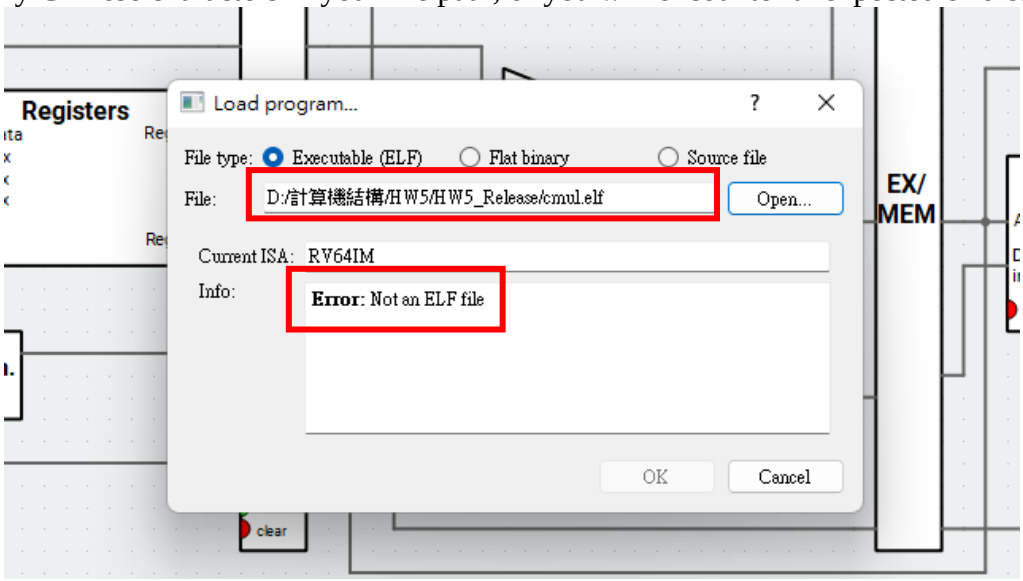
detector locates at the ID stage in the lecture notes and textbook for an improved pipelined implementation. **Please follow Ripes' definitions in this homework.**

Hint:

1. You can change the processor's layout into the extended mode to see more details of the components of the processor.



2. Do not use any Chinese characters in your file path, or you will encounter unexpected errors.



● Submission Rules

Please submit your report in PDF format and your assembly code.
The **PDF report** should be named as `HW5_{your student ID}.pdf`
(e.g., `HW5_123456789.pdf`).
The **source code** should be named as

- `HW5_MJ_{your student ID}.S`
- `HW5_PSSS_{your student ID}.S`
- `HW5_CNPT_{your student ID}.S`
- `HW5_AFROG1_{your student ID}.S`

(e.g., `HW5_MJ_123456789.S`).
Please upload each file separately. DO NOT submit any zipped file.