Milestone 3 — Computer Architecture

Design of Pipelined RISC-V Processors

Hai Cao

rev 2.0.0

Contents

Abstract

This document provides an overview of the tasks, expectations, and requirements for the final milestone in the Computer Architecture course. It details the specific components and specifications students must follow to successfully design at least 2 models of pipelined processors using ISA RV32I. For any errors found or suggestions for enhancement, contact the TA via email at cxhai.sdh221@hcmut.edu.vn using the subject line "**[CA203 FEEDBACK]**".

1 Objectives

- Review understanding of pipeline techniques
- Design several models of pipelined processors for comparison
- Explore the Branch Prediction technique

2 Overview

In this significant milestone, students are assigned the responsibility of designing multiple models of the RV32I processor employing pipelined techniques, as outlined in the lectures. The primary objective of this milestone is to compare at least two techniques to comprehend the functionality of pipelining a processor and address its limitations to attain enhanced performance. Given that the base processor has been successfully implemented in Milestone 2, students are permitted to reuse its components, thereby enabling a feasible timeframe of four

weeks for implementing these strategies. However, it is noteworthy that individuals seeking to enhance the frequency and capacity of their processor must employ BRAMs. While not mandatory, their utilization is highly recommended and may result in additional credit.

Furthermore, students who demonstrate a thorough understanding of the concept of branch prediction are eligible to incorporate branch predictors into their processors, which will be awarded bonus credits. It is important to note that all the prerequisites for communication between your custom processor (soft-core) and external peripherals remain unchanged.

For undergraduate students, an additional penalty week is allocated to ensure the fulfillment of the milestone requirements.

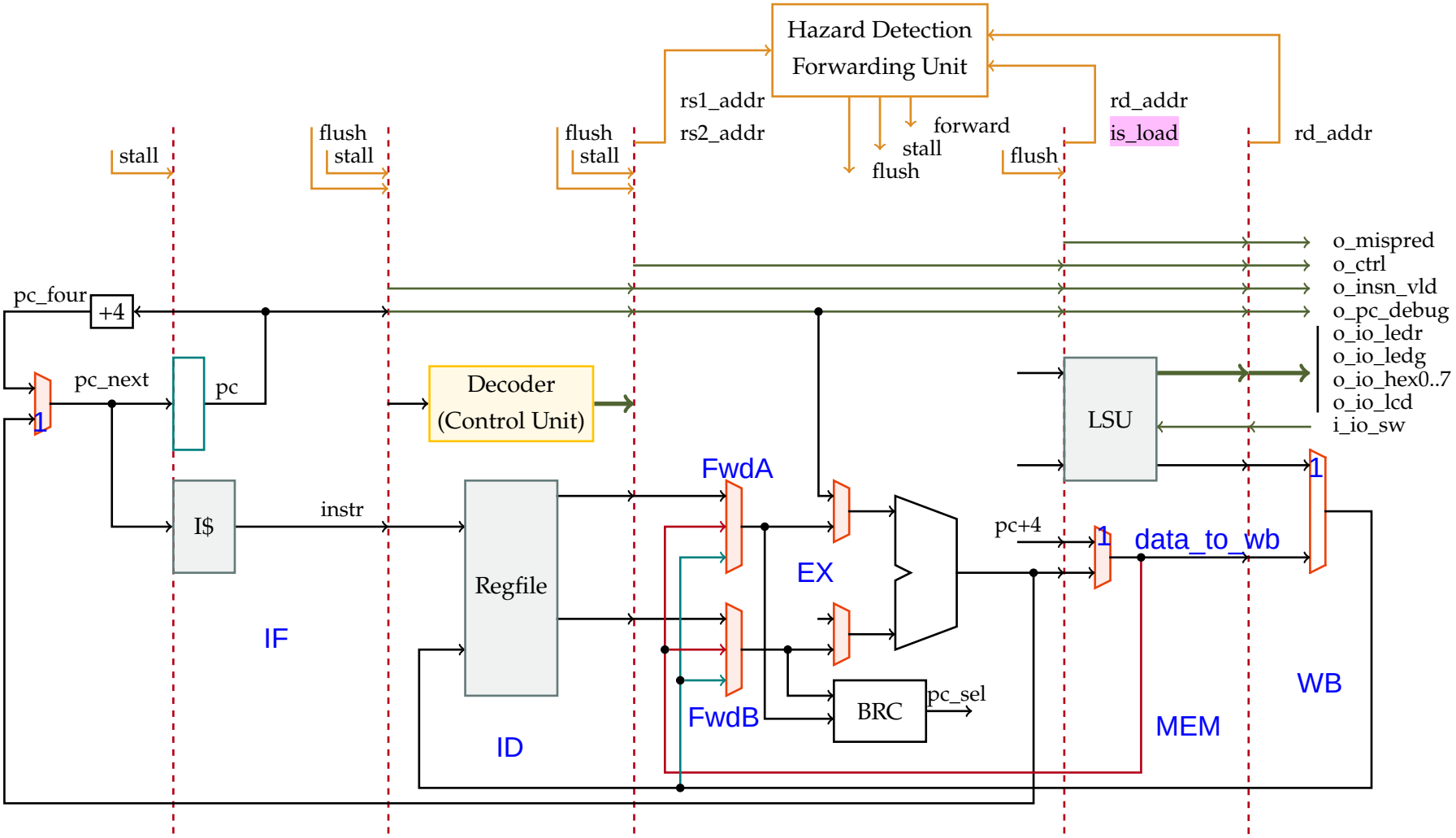

Figure 1: Pipelined Processor with Hazard Detection and Forwarding Unit

2.1 Processor Specification

- **Top-level module:** `pipelined.sv`
- **I/O ports:**

Signal name	Width	Direction	Description
`i_clk`	1	input	Global clock, active on the rising edge.
`i_reset`	1	input	Global low active reset.
`o_pc_debug`	32	output	Debug program counter.
`o_insn_vld`	1	output	Instruction valid.
`o_ctrl`	1	output	Control transfer instructions (branch or jump).
`o_mispred`	1	output	Mispredict.
`o_io_ledr`	32	output	Output for driving red LEDs.
`o_io_ledg`	32	output	Output for driving green LEDs.
`o_io_hex0..7`	7	output	Output for driving 7-segment LED displays.
`o_io_lcd`	32	output	Output for driving the LCD register.
`i_io_sw`	32	input	Input for switches.

2.2 General Guidelines

2.2.1 Directory structure

The project should maintain a well-organized directory hierarchy for efficient management and submission:

```
milestone2
|-- 00_src          # Verilog source files
|-- 01_bench        # Testbench files
|-- 02_test         # Memory files
|-- 10_sim          # Simulation files
|-- 20_syn          # Synthesis files
|   `-- quartus
|       |-- run     # Makefile for synthesis
|       `-- src     # Source files specific to synthesis
`-- 99_doc          # Documentation files
```

2.2.2 Functional verification

To ensure that the implemented design meets the expected functional behavior, leveraging a systematic testing environment, students are provided with a verification environment for straightforward ISA tests. Students' designs act as DUT (Device Under Test) in the environment,

with provided modules for driving stimuli and collecting results. Students are expected to organize their source code correctly, navigate to the simulation directory, and execute the provided scripted makefile for automated simulation. If they already follow the previous structure, copying and pasting the content of `00_src` will suffice.

```
pl-test
|-- 00_src          # Verilog source files
|-- 01_bench        # Testbench files
|   |-- driver.sv
|   |-- scoreboard.sv
|   |-- tbench.sv
|   `-- tlib.svh
|-- 02_test         # Testing files
|   `-- isa.mem     # Hex file
|-- 10_sim          # Verilator
|   |-- flist
|   `-- Makefile
`-- 11_xm           # Xcelium
    |-- flist
    `-- Makefile
```

When the verification with ISA tests completes, students must copy that directory `pl_test` and paste into their designated directories for submission.

3 Week 1: Divide and Conquer

In the first week, students should focus on pipelining the processor and design new memory models.

3.1 Pipelining

Referring to Figure 1, students are tasked with segmenting (**divide**) their single-cycle processor into distinct stages and employing (**conquer**) flip-flops to control the data flow of incoming instructions through the pipeline. Initially, there may be several challenges to overcome. However, it is essential to consider the following key points:

1. Enable and Reset signals of each stage are critical for a proper pipelined processor. Any issues at first arise from their improper control.

2. `insn_vld` is now playing an important role, since flushing an instruction invalidates it, preventing it from being counted as an instruction in the code flow during the final stage (WB). This ensures accurate computation.

3. `o_ctrl` signal is added to record control transfer instructions. If a branch or jump instruction is in the WB stage, this signal is asserted with a value of 1.

4. `o_mispred` signal is added to record mispredictions, when the Hazard Detection asserts a flush due to a control transfer instruction being mispredicted. This signal is propagated to the WB stage as well.

5. Placing a single instruction in the pipeline to verify its correct flow through each stage. Subsequently, introduce more instructions **without any hazards** and compare the number of active `o_insn_vld` signals (high) with the expected number of instructions.

3.2 Memory Modeling and Memory Mapping

This milestone necessitates the utilization of **synchronous read and synchronous write** memory models to achieve optimal timing and frequency. Consequently, students are prohibited from reusing memory models in Milestone 2 and are required to design new ones. The requirement to load the same data file into both data memory and instruction memory remains valid.

Memory mapping is now anticipated to allocate 64KiB in the Memory/RAM region for future benchmarking purposes. Individuals fail to utilize memory bits or BRAM and, consequently, resort to logic elements for generating memory, they should consider implementing the memory mapping of Milestone 2 if they still intend to utilize FPGAs and execute applications.

Base address	Top address	Mapping
0x1001_1000	0xFFFF_FFFF	(Reserved)
0x1001_0000	0x1001_0FFF	Switches *(required)*
0x1000_5000	0x1000_FFFF	(Reserved)
0x1000_4000	0x1000_4FFF	LCD Control Registers
0x1000_3000	0x1000_3FFF	Seven-segment LEDs 7-4
0x1000_2000	0x1000_2FFF	Seven-segment LEDs 3-0
0x1000_1000	0x1000_1FFF	Green LEDs *(required)*
0x1000_0000	0x1000_0FFF	Red LEDs *(required)*
0x0001_0000	0x0FFF_FFFF	(Reserved)
0x0000_0000	0x0000_FFFF	Memory (64 KiB) *(required)*

Table 1: Pipelined Processor Memory Mapping

3.3 BRAM Modeling

If students are interested in utilizing BRAMs, kindly refer to the following link for comprehensive information on their functionality and proper implementation: Cyclone II Memory Blocks.

4 Week 2-3: Five Models and Beyonds

Within the next two weeks, students are required to select two or more models introduced in this section for implementation. The successful "divide and conquer" approach without hazard instructions in Week 1 serves as the foundation for any subsequent modifications based on the models below.

The baseline submission should design **Non-forwarding** and **Forwarding**, ensuring the inclusion of at least two models with their respective parameters for comparison. Milestone 3 is designed to be Incremental, which the next model builds upon the previous models. In other words, "designing and verifying the functionality of the previous model before modifying it into the next one." Students who select more than two models are permitted to exclude the initial two models: **Non-forwarding** and/or **Forwarding**.

Remember:

1. The `o_ctrl` signal should be asserted a control transfer instruction is present.
2. The `o_mispred` signal should be asserted when a flush happens to clear a wrong control transfer instruction (branch or jump).

4.1 Non-forwarding

Building upon the design established in Week 1, you are now required to incorporate Hazard Detection functionality to identify potential hazards between instructions. After doing so,

you must control two crucial signals, `stall` and `flush`, to direct the execution of instructions accordingly.

4.2 Forwarding

With the assistance of a forwarding network and the dedicated Forwarding Unit, the transformation of Non-forwarding into Forwarding can be accomplished with relative ease. However, it is imperative to exercise caution when handling load instructions, as they possess the potential to engender deadlocks.

4.3 Forwarding with Always Taken

By default, the previous design is "always not-taken". To implement a forwarding model where the previous instruction is "always taken", a Branch Target Buffer (BTB) is required. This buffer records the predicted program counter (PC) for each branch or jump instruction. Consequently, the Fetch stage must be modified to retrieve the predicted PC from the BTB if the current PC is stored in the BTB.

4.4 Two-Bit Dynamic Branch Prediction

The preceding models are limited to static prediction due to the absence of a predictor. This model necessitates a straightforward predictor employing the two-bit prediction scheme. The one-bit scheme is inadvisable due to its susceptibility to corner cases, and the distinction between designing two-bit and one-bit schemes is minimal.

4.5 G-share Branch Prediction

Those who decide to implement this model know what they are doing. Good luck!

5 Week 4: Implementation, Presentation, and Submission

5.1 Implementation

In this final week, students are expected to complete their processor designs. The primary focus of this week is to conduct synthesis on Quartus and implement their designs on DE-2 board. Students are welcome to utilize the labs located on Campuses 1 and 2 for hardware implementation. Please approach the lab supervisors for access. Any excuses for not having access to DE-2 will not be considered as a valid reason for failing to meet the milestone.

Following a successful synthesis attempt, kindly record the following data:

Model	**Non-forwarding**	**Forwarding**	**Two-bit**
Frequency F_{max} (MHz)			
Total logic elements (%)			
Total registers (%)			
Total memory bits (%)			

Table 2: Pipelined Processors Synthesis Comparison

Additionally, you have to record the IPC and Mispredict Rate of the ISA test and at least one program you write. Record the result and make a table and a chart for illustration.

Note: you may observe the `tbench.sv` to write your calculation for IPC and Mispredict Rate.

Test		**Non-forwarding**	**Forwarding**	**Two-bit**
ISA test	IPC			
	Mispred Rate (%)			
My Test 1	IPC			
	Mispred Rate (%)			
My Test 2	IPC			
	Mispred Rate (%)			

Table 3: Pipelined Processors Performance Comparison

5.2 Presentation

During the weekend of the fourth week, typically on Saturday or Sunday, students are obligated to be present **in person** to submit their milestone. Should you be unable to present, an email to the Teaching Assistant (TA) should be sent by Friday. However, if you do not receive a confirmation email, your absence will still be considered.

5.3 Submission

Your submission project is the directory `pl-test` that you have executed ISA tests. You must verify that your source code files are in `00_src` and simulation can be executed successfully in either `10_sim` or `11_xm`.

You should number your models as `pl-test-model-1`, `pl_test-model-2`, ...

Designated submission directories are located in `~/submission`. For instance, if your username is `ca101`, your submission folder would be `~/submission/ca101`. The next step is to copy your `pl-test` and place it in `~/submission/ca101` under the same name. You must submit your code and report prior to the Presentation day.

Note: Your code will be scrutinized and run again by TA.

6 Week 5: Penalty

Groups who are unable to complete the assessment within the four-week timeframe may present in the following Saturday only. However, your score will be reduced by one point.

7 ISA Test – Functional Verification

The following are key aspects to understand before proceeding with the simulation:

1. The test environment monitors `o_pc_debug` and `o_io_ledr` signals to determine test outcomes, indicating either a "**pass**" or "**error**" condition.
2. Due to the flexible memory mapping requirements, your design must support 32-bit load/store operations to the LEDR register, mapped to address `0x1000_0000`.

7.1 Environment Setup

Prior to commencing the test, students must copy the singlecycle test from the common directory to their home directory and navigate to it.

```
cd ~
cp -rf ~/common/pl-test .
cd sc-test
```

7.2 Project Directory Hierarchy

The project structure adheres to a hierarchical organization to facilitate efficient simulation and verification. Directories are structured as follows:

```
pl-test
|-- 00_src          # Verilog source files
|-- 01_bench        # Testbench files
|   |-- driver.sv
|   |-- scoreboard.sv
|   |-- tbench.sv
|   `-- tlib.svh
|-- 02_test         # Testing files
|   `-- isa.mem     # Hex file
|-- 10_sim          # Verilator
|   |-- flist
|   `-- Makefile
`-- 11_xm           # Xcelium
    |-- flist
    `-- Makefile
```

`00_src` Place all SystemVerilog source files here.

`01_bench` This directory provides insight into the testbench setup. Study its contents to understand the simulation environment.

`02_test` Contains the file `isa.mem`, a hexadecimal file representing the instruction set test.

`10_sim` This directory includes scripts for simulation using Verilator.

`11_xm` This directory contains scripts for simulation using Cadence Xcelium.

7.3 Memory Configuration

To ensure compatibility with the testbench, make the following modifications to your memory models:

- Since the testbench requires 64 KiB, your design must use an address width of at least 16 bits to supporting a memory size of 64 KiB as in Table 3.
- The memory must be preloaded with the contents of `02_test/isa.mem` to provide test instructions.

7.4 File Setup for Verilator

To run simulations using Verilator, change into `10_sim` directory and edit `flist` file to include all relevant design files. For example, if your top-level module is named `pipelined.sv`, include the entry:

```
./../00_src/pipelined.sv
```

7.5 File Setup for Xcelium

To execute simulations using Cadence Xcelium, change into `11_xm` directory and edit `flist` file to include all relevant design files. For example, if your top-level module is named `pipelined.sv`, include the entry:

```
./../00_src/pipelined.sv
```

7.6 Simulation

With all setup completed, you must first access computing resource and run Makefile script.

1. Run the command below to access a computing node. Without "`--x11`", you cannot use GUI.

   ```
   srun --x11 --pty bash
   ```

2. If you use Verilator, navigate to directory `10_sim` and run "`make`". To observe waveforms, run "`make wave`" to open Surfer or GTKWave.

3. In case you use Xcelium, navigate to directory `11_xm` and run "`make`". To observe waveforms, run "`make gui`" to open SimVision.

7.7 Expected Result

A correctly functioning design should produce the expected output as below:

```
PIPELINE - ISA tests

add......PASS
addi.....PASS
sub......PASS
and......PASS
andi.....PASS
or.......PASS
ori......PASS
xor......PASS
xori.....PASS
slt......PASS
slti.....PASS
sltu.....PASS
sltiu....PASS
sll......PASS
slli.....PASS
srl......PASS
srli.....PASS
sra......PASS
srai.....PASS
lw.......PASS
lh.......PASS
lhu......PASS
lb.......PASS
lbu......PASS
sw.......PASS
sh.......PASS
sb.......PASS
auipc....PASS
lui......PASS
beq......PASS
bne......PASS
blt......PASS
bltu.....PASS
```

```
bge......PASS
bgeu.....PASS
jal......PASS
jalr.....PASS
malgn....ERROR
iosw.....PASS

Result

IPC = 0.xx
Mispred Rate = x.xx

END of ISA tests
```

As handling of misaligned memory addresses is not mandatory, and such scenarios may result in an error status.

7.8 Troubleshooting Common Issues

While the test is termed an "ISA Test," it is designed to validate functional correctness by integrating multiple instructions in each stage. This approach ensures robust verification rather than isolated instruction testing.

Hint: Ensure that the "Elite Four" of instructions — `addi`, `beq`, `jal`, and `lui` — is correctly implemented, as errors in these instructions can cascade and cause other tests to fail.

8 Benchmarking

Content to be added in future versions.

9 Modified Processor

You may implement a modified processor design to incorporate additional features or optimizations. However, it is essential to first demonstrate a complete understanding of the standard processor design. All modifications should be clearly documented and justified.

10 I/O System Conventions

For consistent operation and testing, adhere to the following conventions for setting up with DE2 boards and interacting with the I/O system.

Note: Only when connected to DE2 board, those peripherals data will be truncated accordingly.

- **LEDs** Use the output ports `o_io_ledr` and `o_io_ledg` to control the red and green LEDs, respectively. These can be used for status indicators or debugging.

  `o_io_ledr`

Bits	**Usage**
`31 - 17`	(Reserved)
`16 - 0`	17-bit data connected to the array of 17 red LEDs in order.

  `o_io_ledg`

Bits	**Usage**
`31 - 8`	(Reserved)
`7 - 0`	8-bit data connected to the array of 8 green LEDs in order.

- **Seven-Segment** Utilize `o_io_hex0..7` to display numerical values or messages. Each port has 7 bits in total, so four of them can represent a 32-bit data as shown below. To control each seven-segment display, stores a byte at the corresponding address, such as `SB` at `0x1000_2000` will change the value of HEX2, while `SH` at the same location will affect both HEX2 and HEX3. For the case of misaligned addresses, your assumption is critical.

Address	**0x1000_2000**
Bits	**Usage**
`31`	(Reserved)
`30 - 24`	7-bit data to HEX3.
`23`	(Reserved)
`22 - 16`	7-bit data to HEX2.
`15`	(Reserved)
`14 - 8`	7-bit data to HEX1.
`7`	(Reserved)
`6 - 0`	7-bit data to HEX0.
Address	**0x1000_3000**
Bits	**Usage**
`31`	(Reserved)
`30 - 24`	7-bit data to HEX7.
`23`	(Reserved)
`22 - 16`	7-bit data to HEX6.
`15`	(Reserved)
`14 - 8`	7-bit data to HEX5.
`7`	(Reserved)
`6 - 0`	7-bit data to HEX4.

- **LCD Display** Manage more complex visual output through `o_io_lcd`. To drive LCD properly, visit this link to investigate the specification of LCD HD44780.

Bits	Usage
`31`	ON
`30 - 11`	(Reserved)
`10`	EN
`9`	RS
`8`	R/W
`7 - 0`	Data.

- **Switches** Use `i_io_sw` to receive input from external switches, which can be used for user interaction or control signals.

Bits	Usage
`31 - 18`	(Reserved)
`17`	Reset.
`16 - 0`	17-bit data from SW16 to SW0 respectively.

11 Applications

Develop an application that utilizes the designed processor, demonstrating its capabilities and practical use. The complexity and innovation of the application will impact your grading. Simple applications might include basic input/output handling, while more advanced applications could involve complex calculations or data processing.

Below are some example programs with its expected score:

- Design a stopwatch using seven-segment LEDs as the display.
- Convert a hexadecimal number to a decimal number and display on seven-segment LEDs.
- Convert a hexadecimal number to its decimal and binary forms and display on LCD.
- Input 3 2-D coordinates of A, B, and C. Determine which point, A or B, is closer to C using LCD as the display.

12 Rubric

Your project will be evaluated based on the following criteria:

1. **Baseline Submission – 7 pts:** If your designs successfully pass the ISA test, you will earn 7 points. Please ensure that your source code is submitted to the server for verification and transparency and your reports on LMS. No in-person presentation is required. If you choose not to present your work, please indicate your consent in the Group Sign-Up sheet.
2. **BRAM Utilization – 1 pts:** Individuals who employ BRAM to enhance the frequency of their processors will receive a one-point bonus.
3. **Branch Prediction – 2 pts:** Those that integrate branch predictors introduced in the lecture, such as the two-bit scheme and G-share, will earn additional points depending on the complexity of the design.
4. **Advanced or Alternative Design – 2 pts:** For students who incorporate substantial modifications or enhancements to the baseline processor design, up to two additional points will be awarded. The assessment will be based on the innovation, complexity, and functionality of these improvements.
5. **Bonus: Benchmarking – 1 pts:** Students who benchmark their processors with provided benchmark tests in this course will receive an additional credit.

12.1 Report

A comprehensive project report must be submitted, detailing the design process, challenges faced, and solutions implemented. Refer to the report guidelines on Google Drive. The report should be clear and concise, with sections for introduction, methodology, results, and conclusions. Visual aids such as diagrams and charts are encouraged to illustrate key points.