Milestone 2 — Computer Architecture

Design of a Single Cycle RISC-V Processor

Hai Cao

rev 2.0.0

Contents

Abstract

This document provides an overview of the tasks, expectations, and requirements for the second milestone in the Computer Architecture course. It details the specific components and specifications students must follow to successfully design a single-cycle RV32I processor. 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 SystemVerilog
- Review understanding of RV32I instructions
- Design a single-cycle RV32I processor

2 Overview

In this milestone, students are tasked with designing a single-cycle RV32I processor, as discussed in lectures. The processor is implemented with the ISA RV32I, excluding FENCE instructions. To enable communication between your custom processor (soft-core) and external peripherals, some modifications to the standard processor design are necessary. The standard peripherals it must support are LCD, LEDs, and SWITCHes; seven-segment LEDs are optional. In order to accommodate the lecture, for memory, the processor utilizes two identical logic element memory models. One is read-only for the instruction memory, while the other is read-write for the data memory.

The design must adhere to the specifications outlined below and will be tested against both student-created testbenches and a comprehensive testbench provided by the TA. Adherence to the suggested specifications, while not mandatory, is strongly recommended.

This milestone provides the timeline for students to follow, so that they can complete their processor within four weeks. For undergraduate students, there is an additional penalty week to fulfill the milestone requirements.

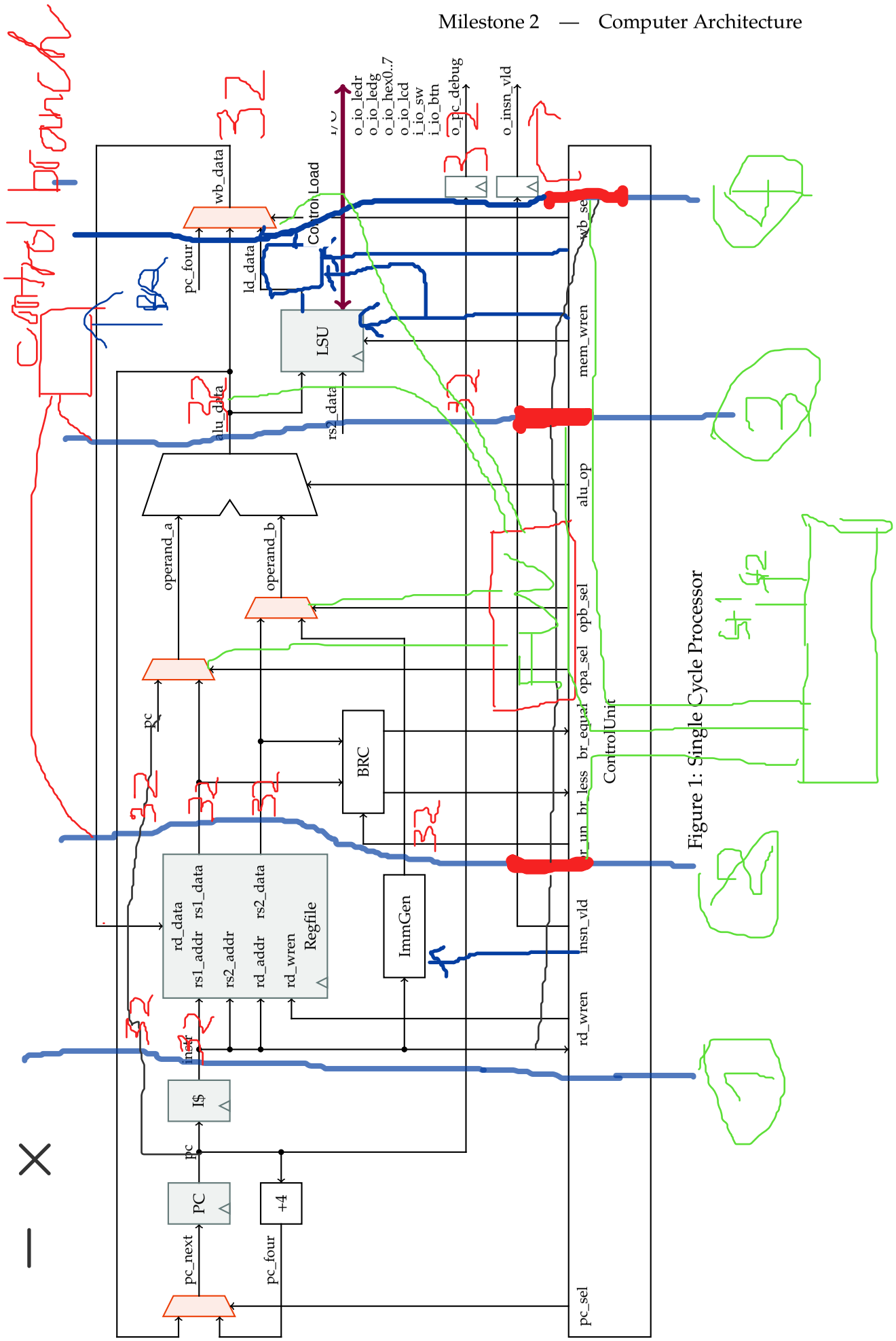

Figure 1: Single Cycle Processor

2.1 Processor Specification

- **Top-level module:** `singlecycle.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_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 Design 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         # Testing files
|   |-- asm         # Assembly test code
|   `-- dump        # Binary/hex dump files
|-- 10_sim          # Simulation files
|-- 20_syn          # Synthesis files
|   `-- quartus
|       |-- run     # Makefile for synthesis
|       `-- src     # Source files specific to synthesis
`-- 99_doc          # Documentation files
```

This project directory is supposed to be copied into the submission directory on the server.

3 Week 1: ALU and BRC

In the first week, students are expected to complete the ALU (Arithmetic and Logic Unit) and BRC (Branch Comparison) modules. Students should utilize randomized stimuli and place some assertions on the outputs to test these modules.

3.1 Arithmetic Logic Unit (ALU)

3.1.1 Requirement

The ALU must be capable of executing a variety of arithmetic and logical operations as defined by the RV32I instruction set.

`alu_op`	**Description (R-type)**	**Description (I-type)**
ADD	$rd \leftarrow rs1 + rs2$	$rd \leftarrow rs1 + imm$
SUB	$rd \leftarrow rs1 - rs2$	n/a
SLT	$rd \leftarrow (rs1 < rs2)? \, 1 : 0$	$rd \leftarrow (rs1 < imm)? \, 1 : 0$
SLTU	$rd \leftarrow (rs1 < rs2)? \, 1 : 0$	$rd \leftarrow (rs1 < imm)? \, 1 : 0$
XOR	$rd \leftarrow rs1 \oplus rs2$	$rd \leftarrow rs1 \oplus imm$
OR	$rd \leftarrow rs1 \vee rs2$	$rd \leftarrow rs1 \vee imm$
AND	$rd \leftarrow rs1 \wedge rs2$	$rd \leftarrow rs1 \wedge imm$
SLL	$rd \leftarrow rs1 \ll rs2[4:0]$	$rd \leftarrow rs1 \ll imm[4:0]$
SRL	$rd \leftarrow rs1 \gg rs2[4:0]$	$rd \leftarrow rs1 \gg imm[4:0]$
SRA	$rd \leftarrow rs1 \ggg rs2[4:0]$	$rd \leftarrow rs1 \ggg imm[4:0]$

Note: Refrain from employing built-in SystemVerilog operators for subtraction ($-$), comparison ($<$, $>$), shifting ($\ll$, $\gg$, and $\ggg$), multiplication ($*$), division ($/$), modulo (%), and other unsynthesizable operators in designs, except for verification purposes.

3.1.2 Suggested specification

- **Module name:** `alu.sv`
- **I/O ports:**

Signal name	**Width**	**Direction**	**Description**
`i_op_a`	32	input	First operand for ALU operations.
`i_op_b`	32	input	Second operand for ALU operations.
`i_alu_op`	4	input	The operation to be performed.
`o_alu_data`	32	output	Result of the ALU operation.

3.2 Branch Comparison Unit (BRC)

3.2.1 Requirement

This unit is responsible for comparing two registers' values to determine the outcome of branch instructions. It should be capable of handling both signed and unsigned comparisons, and thus needs an additional signal to determine the output.

Note: Refrain from employing built-in SystemVerilog operators for subtraction ($-$), comparison ($<$, $>$), shifting ($\ll$, $\gg$, and $\ggg$), multiplication ($*$), division ($/$), modulo (%), and other unsynthesizable operators in designs, except for verification purposes.

3.2.2 Suggested specification

- **Module name:** `brc.sv`
- **I/O ports:**

Signal name	Width	Direction	Description
`i_rs1_data`	32	input	Data from the first register.
`i_rs2_data`	32	input	Data from the second register.
`i_br_un`	1	input	Comparison mode (1 if signed, 0 if unsigned).
`o_br_less`	1	output	Output is 1 if $rs1 < rs2$.
`o_br_equal`	1	output	Output is 1 if $rs1 = rs2$.

4 Week 2: Regfile, Memory, and LSU

In the second week, students must design a register file and a memory model using logic element in order to satisfy the requirement of asynchronous read. Also, for memory, students have to learn to use the keyword `$readmemh` to load a file including data/instruction in to memory array(s) of memory models. Another important part is to understand how different peripherals and memory are allocated into address space, and hence the necessity of Load-Store Unit.

Note: These memory models will **NOT** be reused in Milestone 3.

4.1 Regfile and Memory

4.1.1 Requirement

Implementing a register file with 32 registers, each 32-bit wide. The register file must have two read ports and one write port, with register 0 always reading as zero.

In the manner, students must design two identical memory models using **logic elements**. For the purpose of illustrating how singlecycle processor works, these two models are asynchronous read, but synchronous write.

4.1.2 Suggested specification

- **Module name:** `regfile.sv`
- **I/O ports:**

Signal name	Width	Direction	Description
`i_clk`	1	input	Global clock.
`i_reset`	1	input	Global active reset.
`i_rs1_addr`	5	input	Address of the first source register.
`i_rs2_addr`	5	input	Address of the second source register.
`o_rs1_data`	32	output	Data from the first source register.
`o_rs2_data`	32	output	Data from the second source register.
`i_rd_addr`	5	input	Address of the destination register.
`i_rd_data`	32	input	Data to write to the destination register.
`i_rd_wren`	1	input	Write enable for the destination register.

- **Module name:** `memory.sv`
- **I/O ports:**

Signal name	Width	Direction	Description
`i_clk`	1	input	Global clock.
`i_reset`	1	input	Global active reset.
`i_addr`	?	input	Address for both read and write operations.
`i_wdata`	32	input	Write data.
`i_bmask`	4	input	Byte mask, 1 for enable, otherwise 0.
`i_wren`	1	input	Write enable, 1 if writing, 0 if reading.
`o_rdata`	32	output	Read data.

4.2 I/O System and Memory

4.2.1 Memory Mapping

In real-world applications, a processor interfaces with peripheral devices to either transmit data or receive data through the implementation of an Input/Output (I/O) System. Common peripheral devices include LEDs, LCDs, and switches, among others. These peripherals essentially function as a form of "memory" or "registers". For instance, when a 32-bit register is linked to a set of 32 LEDs, depositing data into that register results in manipulating the state of the LED array.

Memory mapping is a strategic method used to organize the layout of memory with different memory regions serving specific functions. Figure 2 illustrates the memory map and the register boundary addresses of some peripherals of STM32F030.

Bus	Boundary address	Size	Peripheral
-	0x4800 1800 - 0x5FFF FFFF	~384 MB	Reserved
AHB2	0x4800 1400 - 0x4800 17FF	1 KB	GPIOF
	0x4800 1000 - 0x4800 13FF	1 KB	Reserved
	0x4800 0C00 - 0x4800 0FFF	1 KB	GPIOD
	0x4800 0800 - 0x4800 0BFF	1 KB	GPIOC
	0x4800 0400 - 0x4800 07FF	1 KB	GPIOB
	0x4800 0000 - 0x4800 03FF	1 KB	GPIOA
-	0x4002 4400 - 0x47FF FFFF	~128 MB	Reserved
AHB1	0x4002 3400 - 0x4002 43FF	4 KB	Reserved
	0x4002 3000 - 0x4002 33FF	1 KB	CRC
	0x4002 2400 - 0x4002 2FFF	3 KB	Reserved
	0x4002 2000 - 0x4002 23FF	1 KB	FLASH Interface
	0x4002 1400 - 0x4002 1FFF	3 KB	Reserved
	0x4002 1000 - 0x4002 13FF	1 KB	RCC
	0x4002 0400 - 0x4002 0FFF	3 KB	Reserved
	0x4002 0000 - 0x4002 03FF	1 KB	DMA

Figure 2: STM32F030 peripheral register boundary addresses

In this course, the traditional Data Memory is replaced by (and put into) Load-Store Unit (LSU), which in turn handles data operations and I/O related tasks. The basic implementation of LSU is presented in Figure 3.

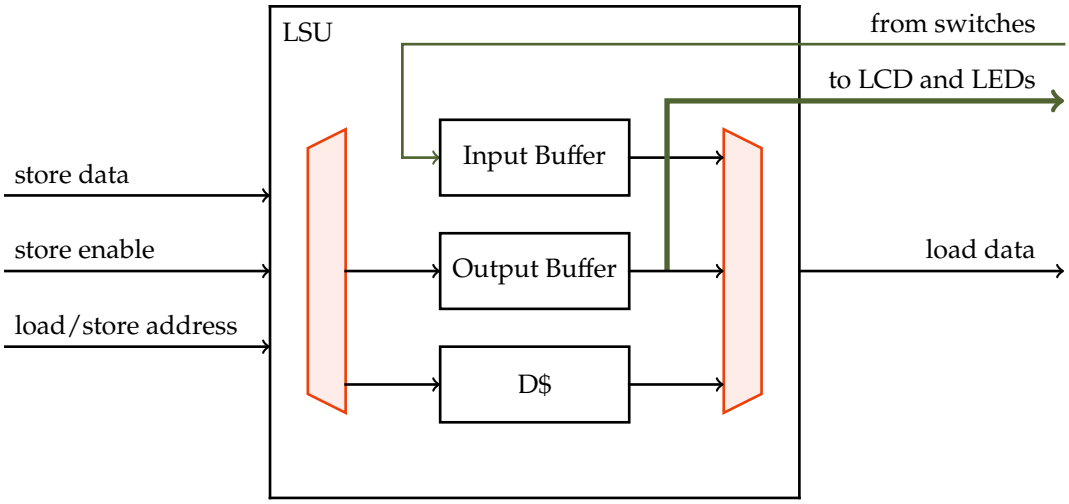

Figure 3: Load Store Unit

4.2.2 Requirement

Implement a Load-Store Unit (LSU) to manage memory-mapped in Table 1.

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)*
0x0000_0800	0x0FFF_FFFF	(Reserved)
0x0000_0000	0x0000_07FF	Memory (2KiB) *(required)*

Table 1: LSU memory mapping

Students are allowed to use reserved spaces for their own modifications.

4.2.3 Suggested specification

- **Module name:** `lsu.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 active reset.
i_lsu_addr	32	input	Address for data read/write.
i_st_data	32	input	Data to be stored.
i_lsu_wren	1	input	Write enable signal (1 if writing).
o_ld_data	32	output	Data read from memory.
o_io_ledr	32	output	Output for red LEDs.
o_io_ledg	32	output	Output for green LEDs.
o_io_hex0..7	7	output	Output for 7-segment displays.
o_io_lcd	32	output	Output for the LCD register.
i_io_sw	32	input	Input for switches.

4.2.4 Special considerations

1. Instruction memory and Data memory are identical: 2KiB in size and initialized from a file named `mem.dump` file located in `02_test/dump`.
2. Memory write operations require a clock edge, whereas read operations do not.

3. Memory models **have to** implement load (LB, LH, LBU, LHU) and store (SB, SH) instructions.

4. Misaligned addresses must be handled as below:

   (a) Addresses for accessing to a word must be end with `0x0, 0x4, 0x8, 0xC` (`0b00`), otherwise the last two bits will be truncated. Addresses of `0x2100` and `0x2101` both access word `0x2103, 0x2102, 0x2101, 0x2100`. If you can handle misaligned addresses, the latter would access `0x2104, 0x2103, 0x2102, 0x2101`.

   (b) Addresses for accessing to a half-word must be end with `0x0, 0x2, 0x4, 0x6, 0x8, 0xA, 0xC, 0xE` (`0b0`), otherwise the last bit will be truncated. Addresses of `0x2100` and `0x2101` both access word `0x2101, 0x2100`. If you can handle misaligned addresses, the latter would access `0x2102, 0x2101`.

   (c) Addresses for accessing to a byte should not create misaligned addresses.

5 Week 3: Control Logic and Testing

In this week, students should have all modules working properly except Control Unit and Immediate Generator. For a team of two, one could investigate the block diagram in Figure 1 to have a sufficient number of datapath scenarios for testing, and the other tackles those two designs. Then, integrate these components with the memory modules to complete their processor design.

Note: The signal `o_insn_vld` has to be implemented, for the grand test will use it as a metric to decide if your design is running properly or not. As the name suggests, if the instruction is valid, it is set to 1.

From now on, they can test each instruction, and then do some combinations of 5 to 10 different instructions. Together with Venus, students can write more complex assembly code and verify their design more intensively.

This week is crucial for two heavyweight requirements:

1. Students are required to run TA's grand test in order to proceed to the next stage. Details will be provided later.

2. A demo application written in RISC-V assembly is needed for the presentation should be coded in this week. Please read Section 11 for more details.

6 Week 4: Implementation and Presentation

In this final week, students are expected to complete their processor design. 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.

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.

Another consideration is submitting your project on the server. Your project directory **has to** follow the directory hierarchy, named `milestone2` and placed into your directory in the folder `submission`. Your code will be scrutinized and run again by TA.

7 Week 5: Penalty

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

8 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.

9 Verification

A comprehensive testbench is crucial for verifying the functionality of your processor. Your testbench should cover a wide range of scenarios and corner cases to ensure thorough testing. The TA will provide a final, comprehensive testbench for further validation one week before the presentation. You are expected to create your testbenches and verify your design before this point.

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:

1 pt Design a stopwatch using seven-segment LEDs as the display.

1 pt Convert a hexadecimal number to a decimal number and display on seven-segment LEDs.

1.5 pts Convert a hexadecimal number to its decimal and binary forms and display on LCD.

2 pts 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 design successfully passes all provided test cases, 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 Presentation sheet.

2. **Demonstration – 2 pts:** Students who choose to present their project will, additionally, receive up to 2 points. A successful demonstration entails synthesizing and implementing your RISC-V processor on the DE2 board, along with running a pre-prepared application on it. The complexity and innovation shown in your demonstration will directly influence your score.

3. **Technical Enquiries – 2 pts:** Each group member will be asked a question related to the design and implementation of their project. Responses will be evaluated based on depth of understanding and clarity.

4. **Advanced or Alternative Design – 1 pts:** For students who incorporate substantial modifications or enhancements to the baseline processor design, up to 1 additional point will be awarded. The assessment will be based on the innovation, complexity, and functionality of these improvements. It's recommended to implement alternative or advanced features early, as this effort will benefit you in Milestone 3.

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.