Milestone 2 — Design of a Single-Cycle Processor

Lecturer: PhD. Linh Tran
TA: Hai Cao
Department of Electronics
HCMC University of Technology, VNU-HCM

Abstract

*This document presents the second milestone in course EE4423. In case you meet an error or have any improvement in this document, please email the TA: cxhai.sdh221@hcmut.edu.vn with the subject “**[COMPARCH203: FEEDBACK]**”*

1 Objectives

- Review understanding of SystemVerilog
- Review understanding of RV32I instructions
- Design a single-cycle RV32I processor

2 ALU and Branch Comparison

2.1 Preliminary

The ALU in CPUs performs some sort of operation, depending on the ISA. Branch Comparison, on the other hand, only compares two data. In this course, students will use an RV32I ISA. However, first, there are some concepts to be reviewed.

Questions

You should complete those exercises by hand. Binary Representation

- Convert this binary number to its hexadecimal representation:
  `16'b0010001110110011`
- Convert this hexadecimal number to its binary representation:
  `16'hEECA`
- Two's complement hexadecimal number of this decimal number:
  `16'd3043`
- Two's complement hexadecimal number of this NEGATIVE decimal number:
  `-16'd2022`

Sign Extension

- Extend to a 16-bit two's complement hexadecimal number: `8'h15`
- Extend to a 16-bit two's complement hexadecimal number: `8'hE9`

Addition and Subtraction

- Determine: `16'h5A78 + 16'h11FE`
- Determine: `16'hEFB7 + 16'h6AA9`
- Determine: `16'h7713 - 16'h3BC1`
- Explain the concept of overflow and underflow.

Logic Operation

- Determine: `16'h5A78 and 16'h11FE`
- Determine: `16'hEFB7 or 16'h6AA9`
- Determine: `16'h7713 xor 16'h3BC1`

Shift Operation

- Determine: `16'h5A78 << 5`
- Determine: `16'hEFB7 >> 5`
- Determine: `16'hF713 >>> 5`

Comparison

- Given two 16-bit numbers, what operators could be used to know they're identical?
- Given two 16-bit numbers, how to determine which one is less than the other?

2.2 Requirements

2.2.1 ALU

Design an ALU to perform operations in an RV32I processor. The table below shows its operations an RV32I ALU needs to be implemented.

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

1. Do NOT use `-, >, <` (SystemVerilog operations for subtraction and comparison)

2. The module name is `alu`

3. Inputs:

`operand_a` the first operand — $rs1$.

`operand_b` the second operand — $rs2$.

`alu_op` an operation that the ALU has to perform.

4. Outputs

`alu_data` the result of the operation.

2.2.2 Branch Comparison

Design a Branch Comparison, which gathers two register values and compares.

1. Do NOT use `-, >, <` (SystemVerilog operations for subtraction and comparison)

2. The module name is `brcomp`

3. Inputs:

`rs1_data` the first operand — $rs1 \to A$.

`rs2_data` the first operand — $rs2 \rightarrow B$.

`br_unsigned` 1 if the two operands are unsigned.

4. Outputs

`br_less` 1 if $A < B$.

`br_equal` 1 if $A = B$.

3 Regfile and Load-Store Unit

3.1 Regfile

In this lab, memory is understood as a storage to store instructions and store/load data. As "packed array" gives designers a vector/scalar of bits, they could use packed arrays to model a memory, yet "unpacked array" will also help designers in the simulation phase. Unlike memory usually with one read port and one write port, the register file, or RegFile, in RISC-V has a fixed number of registers, 32-bit data, and two read ports and one write port. Another difference is that it has one constant register with the value of 0.

Questions

- What are the differences between a packed array and an unpacked array?
- How many address bits are required for a 8KB memory?
- What do those keywords mean?
  `$writememh, $writememb, $readmemh, $readmemb.`

3.1.1 Requirements

Design a register file according to the RISC-V specification: 32 32-bit registers, the register 0 has the value of 0. Notice: the clock has to be named `clk_i`, and the low active reset has to be named `rst_ni`. This convention has already been mentioned in Milestone 1.

1. The module name is `regfile.sv`.
2. Inputs:

`clk_i` positive clock.

`rst_ni` low negative reset.

`rs1_addr` the address for $rs1$.

`rs2_addr` the address for $rs2$.

`rd_addr` the address for rd.

`rd_data` the write data for rd.

`rd_wren` 1 if write to rd.

3. Outputs

`rs1_data` the write data for $rs1$.

`rs2_data` the write data for $rs2$.

4. The Register File writes data into a file called `regfile.data`.

5. In the memory and register file, the write data are available to read in the next cycle, but the read data dont need the clock.

3.2 I/O System and Memory Mapping

In reality, a processor communicates with peripherals in order to export data or read data. This could be completed by designing an I/O System. Some standard peripherals are LEDs, LCD, or switches, etc. Such peripherals, in fact, are seen as "memory." For example, when a 32-bit register is assigned to 32 LEDs, writing data to that register means driving the state of the LED array.

Memory mapping is a technique to lay out the structure of memory. Different regions in memory may serve different purposes. The figure below is the memory map of MSP430.

00FFFFh 00FF80h	Interrupt Vector
0243FFh 004004	Code Memory
0043FFh 002400h	RAM
0023FFh 00IC00h	USB Ram
0091FFh 001800H	Information Memory
0017FFh 00I000h	Boot Ladder Memory
000FFFh 0h	Peripherals

Figure 1: Memory-mapping of MSP430

3.3 Requirements

Design a Load-Store Unit with a memory map and diagram in Figure 2.

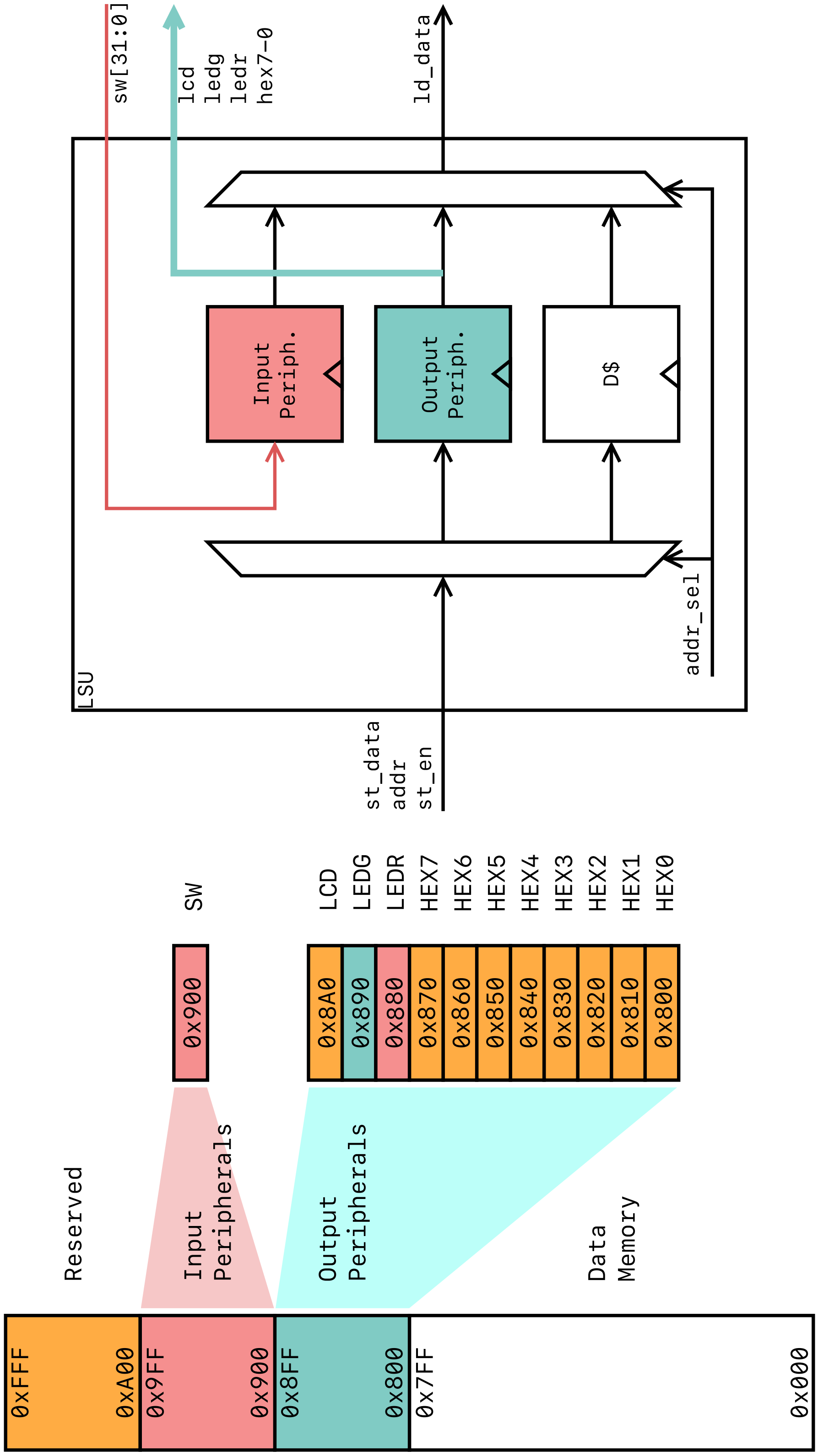

Figure 2: Memory-mapping and LSU diagram

Notice: the clock has to be named `clk_i`, and the low active reset has to be named `rst_ni`. This convention has already been mentioned in Milestone 1.
You have to design two memory models for Intruction (8KB) and Data (2KB).

1. The module name is `lsu`.
2. Inputs:

   `clk_i` positive clock.

   `rst_ni` low negative reset.

   `addr` the address for both read and write.

   `st_data` the store data.

   `st_en` 1 if write, 0 if read.

   `io_sw` 32-bit from switches.

3. Outputs:

   `ld_data` the load data.

   `io_lcd` 32-bit data to drive LCD.

   `io_ledg` 32-bit data to drive green LEDs.

   `io_ledr` 32-bit data to drive red LEDs.

   `io_hex0..7` 8 32-bit data to drive 7-segment LEDs.

4. There are some instructions that do NOT write or read 32-bit but less: LB, LH, LBU, LHU, SB, SH. You may add signals to achieve desired operations. You must show your work in the report.

4 Single-Cycle Processor

To complete the processor, you must design Control Unit, Immediate Generator, and then integrate the memory modules into your project. The standard processor in this course is described in Figure 3.

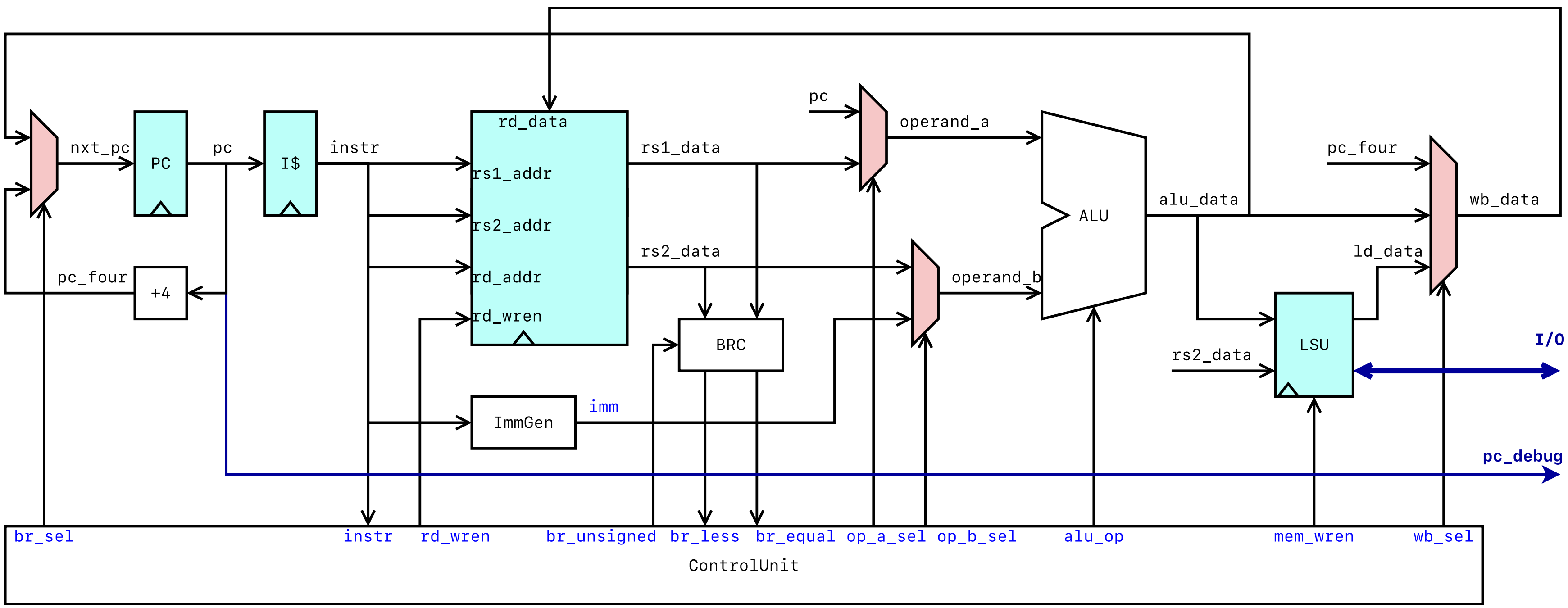

Figure 3: Standard Single-Cycle Processor

4.1 Control Unit and Immediate Generator

Control Unit requires and produces the most number of signals in the processor. Hence, you have two approaches:

- Using `case` and/or `if`.
- Using `instr` as the index to generate the data from ROM table.

1. The module name is `ctrl_unit`.
2. Inputs:

   `instr` the 32-bit instruction.

   `br_less` data from Branch Comparison, 1 if $A < B$.

   `br_equal` data from Branch Comparison, 1 if $A = B$.

3. Outputs

   `br_sel` select PC source: 0 if $PC + 4$, 1 if computed in ALU.

   `br_unsigned` 1 if the two operands are unsigned.

   `rd_wren` 1 if the instruction writes data into Regfile.

   `mem_wren` 1 if the instruction writes data into LSU.

   `op_a_sel` select operand A source: 0 if $rs1$, 1 if PC.

   `op_b_sel` select operand B source: 0 if $rs2$, 1 if imm.

   `wb_sel` select data to write into Reggile: 0 if `alu_data`, 1 if `ld_data`, and 2 or 3 if `pc_four`.

4. You may add signals for LB, LH, LBU, LHU, SB, SH.

Immediate Generator, however, is left for you to design.

4.2 Modified Processor

Besides the standard processor, a modified processor is presented in Figure 4. You may design yours using this model, but you still need to understand the standard one thoroughly first.

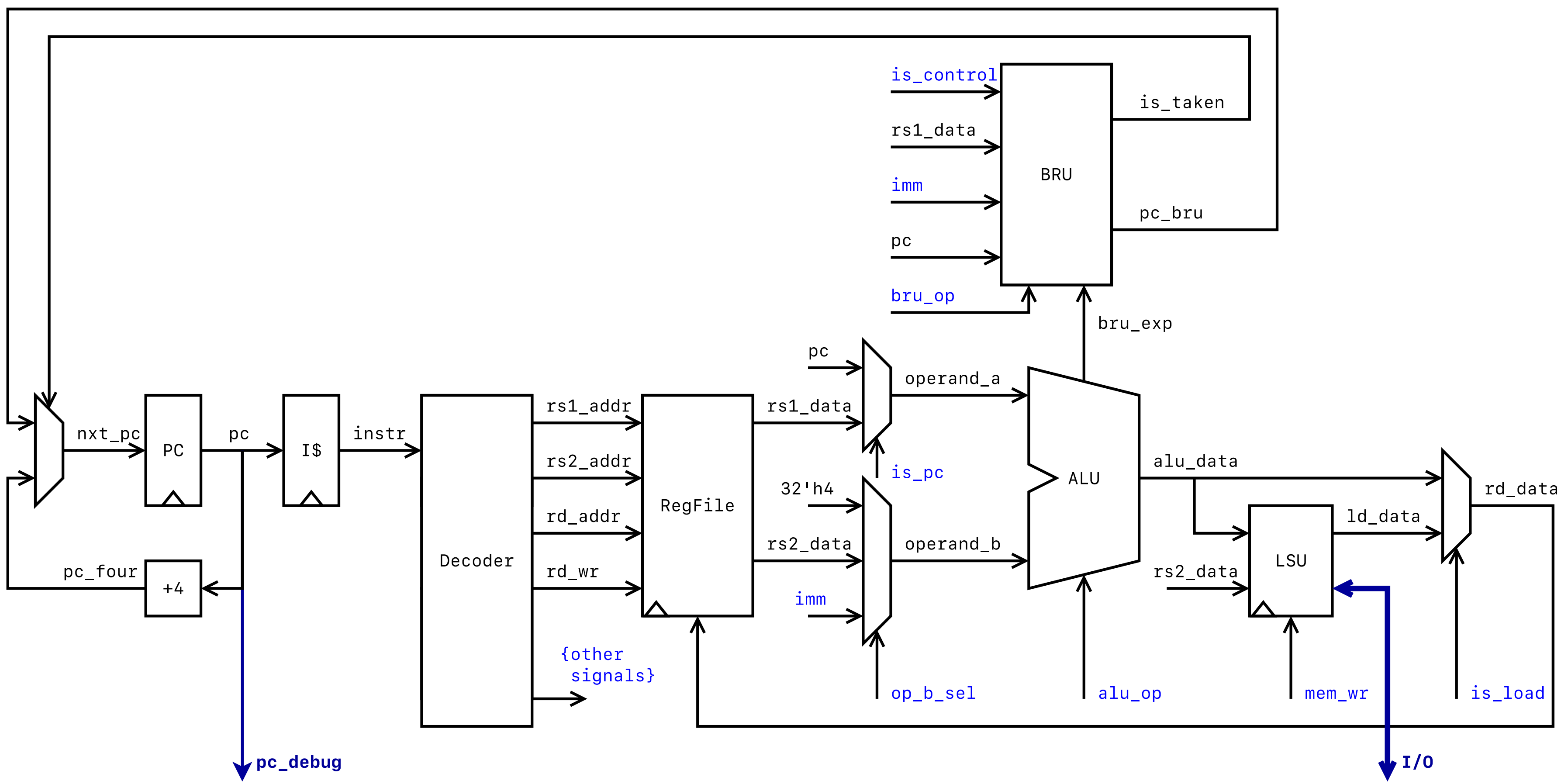

Figure 4: Modified Single-Cycle Processor

4.3 Requirements

No matter what diagram you use to design your processor, your final design must satisfy the naming below to ensure the testbench running properly.

1. The module name is `singlecycle`.
2. Inputs:

   `clk_i` positive clock.

   `rst_ni` low negative reset.

   `io_sw_i` 32-bit from switches.

3. Outputs

   `pc_debug_o` PC.

   `io_lcd_o` 32-bit data to drive LCD.

   `io_ledg_o` 32-bit data to drive green LEDs.

   `io_ledr_o` 32-bit data to drive red LEDs.

   `io_hex0..7_o` 8 32-bit data to drive 7-segment LEDs.

4.4 Testbench

First, you must manage your project with these directories:

`mem` contains binary text files, which you use for your memory.

`quartus` contains Quartus related-files.

`src` contains all your source code, so make sure all your code files are put into it.

`tb` contains test files, such as `makefile`.

Second, you may use `lab0` directory as a template to formulate **your own testbench**. Your testbench should include a sufficient number of test cases, which you will present later.

4.5 Conventions

When you write assembly, you will have to convert it into binary code, and put it into your memory model using `$readmemh`, please visit this website to convert your assembly. Below are the conventions for you to drive output peripherals and read switches:

SWITCH

	3										2										1										
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0															17-bit data from SW 16 to 0																

**SW17 is for RESET

LEDR

	3										2										1										
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0															17-bit data to red LEDs																

LEDG

	3										2										1										
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0																								8-bit data to green LEDs							

7 SEGMENT LEDs - HEX

	3										2										1										
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
0																									7-bit data to 7-seg LED						

LCD - HD44780

	3										2										1										
1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	3	2	1	0
ON	0																				EN	RS	R/W	Data							

Figure 5: Conventions

To drive LCD properly, you may visit this link to read the specification of LCD HD44780.

5 Application

You may come with your own idea of an application that utilizes your first processor. The points you get depends on the complexity of your program.

Below are some example programs you could use:

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

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

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

2pts 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.

6 Evaluation

1. Baseline Functionality — 3 points
2. Verification Quality — 3 points
3. Application Demonstration — 2 points
4. Alternative Design — 2 points
5. Code Quality — 1 point
6. Report Quality — 1 point

- **Baseline Functionality** Your design satisfies RV32I ISA and can implement on FPGA.
- **Verification Quality** Your testbench should cover a sufficient number of cases and your verification strategy is reasonable.
- **Application Demonstration** Refer to section 5
- **Alternative Design** Any additional designs or modifications that you add to your processor to improve its performance are evaluated. You may discuss with TA for this matter.
- **Code Quality** Your coding style must be consistent and clean.
- **Report Quality** If you don't submit the report, you have the penalty of 1 point. However, if the report doesn't convey enough content, you have no point for this criteria.

6.1 Report

Your report should be format as the following guideline:

1. Content:
   (a) Student name, student ID, and group ID
   (b) Title
   (c) Introduction
   (d) Design Strategy
   (e) Verification Strategy

(f) Alternative Design
(g) Evaluation
(h) Conclusion

2. Font: Times New Roman or Garamond 12-point
3. Line Spacing: Double-space or 1.5
4. Paper size: A4
5. Margins: one-inch margins all around