ECE 443/543
Final Project

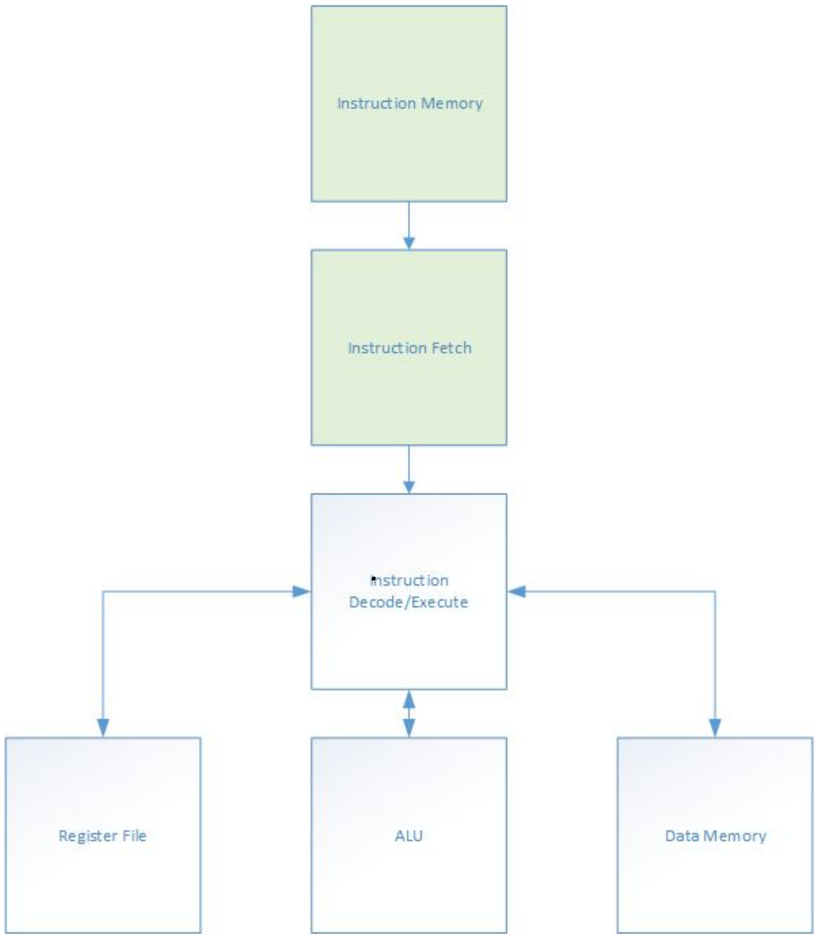
General Guideline

1. **Due on December 2, 11:59 PM- Presentation**
2. **For ECE 543 students, this is an individual project. For ECE 443 students, this is a two people project. For ECE 443 students, only one of your team members need to submit your work to Canvas.**
3. **Late submission is not acceptable.**
4. **You need to submit your VHDL Workspaces and electronic version of your report via Canvas (see the Section of Deliverable for details). You have to zip them into one file and clearly name and organize them.**
5. **For your report, you need to have one cover page to identify name, ECE 443 or ECE 533 student, and your team member (if applicable).**
6. **You need to put the comments to your code**
7. **You should schedule time with the instructor to present your project as a team**

ALDEC will be used, not Quartus.

Objective

For the final project, you will be implementing a simple processor. The processor will support 7 different instructions, two instruction formats, support 256 half words of RAM (256 addresses by 16 bits) and have 8 registers. The following figure shows the high level overview of this processor.



High level overview of the processor

For this project, you will be implementing a processor. The basic design is a single cycle processor which implements the Instruction Decode/Execute, ALU, Register File and Data Memory sections. Each instruction is 16 bits wide. There are two formats, R-Type (register) and I-Type (immediate). Their formats are shown below. R-Type instructions perform an operation using the registers specified in **a** and **b** portions. The result is stored in the register specified by **c** portion. I-Type instructions specify an immediate value. In this project, the ldi instruction is the load immediate instruction. It loads the immediate value into the register specified by d portion. The sh and lh instructions store and load halfwords into the memory address specified by the immediate value.

R-Type

unused	**opcode**	**c**	**a**	**b**
1 bit	3 bits	4 bits	4 bits	4 bits

I-Type

unused	**opcode**	**d**	**value**
1 bit	3 bits	4 bits	8 bits

The opcodes for the different instructions are specified below.

S0	S1	S2	Instruction Format	Operation	Mnemonic
0	0	0	R	Signed Addition	add
0	0	1	R	Signed Multiplication	mult
0	1	0	R	Passthrough A	pa
0	1	1	R	Passthrough B	pb
1	0	0	R	Signed Subtraction	sub
1	0	1	I	Load Immediate	ldi
1	1	0	I	Store halfword	sh
1	1	1	I	Load halfword	lh

Requirement: Both the register file and the RAM/memory can be implemented behaviorally using arrays. The instruction decode/execute is implemented using structural model.

Your processor should execute the following program.

```
ldi $r0, 10
ldi $r1, 5
ldi $r2, 0
ldi $r3, 0
ldi $r4, 0
ldi $r5, 0
ldi $r6, 0
ldi $r7, 0
add $r2, $r0, $r1
mult $r3, $r0, $r1
sub $r4, $r0, $r1
sh $r3, 0x0B
sh $r4, 0x0A
lh $r6, 0x0A
lh $r7, 0x0B
```

The following is the program compiled into hexadecimal.

```
500A
5105
5200
5300
```

```
5400
5500
5600
5700
0201
1301
4401
630B
640A
760A
770B
```

Here is the basic requirement for ALU. **This ALU must be implemented structurally using VHDL, except for** multiplier**.** The ALU must take two 16-bit input arguments (A and B) and produce a 16-bit output argument (R). A, B and R are **signed** numbers. The instruction will be selected using three signal lines (S2, S1 and S0). There will also be a 3 bit status output (status). The status output is described in Table 2. Table 1 list the name of the I/O lines, their size and direction.

Table 1 – I/O Lines

Name	**Direction**	**Size**
A	I	16
B	I	16
R	O	16
S2	I	1
S1	I	1
S0	I	1
status	O	3

Table 2 – status output description

Bit Position	**Meaning when high**
2	overflow condition
1	result is zero
0	result is negative

The instruction set is shown in Table 3. All undefined operations should result in **R** being zero.

Table 3 - Opcodes

S2	S1	S0	Operation
0	0	0	Signed Addition
0	0	1	Signed Multiplication
0	1	0	Passthrough A
0	1	1	Passthrough B
1	0	0	Signed Subtraction
1	0	1	Undefined
1	1	X	Undefined

You must show that all of the parts of the ALU (adder, subtractor, multiplier and multiplexer) and the ALU itself work by showing the functional simulation diagrams. You should come up and show enough test cases to prove that it is working. For example, if you were testing the adder you could do 2+2, 2+(-2) and (-2)+(-2).

Hint:

The multiplexer should be an 8-input multiplexer. This selects the correct output for the instruction and there are 5 possible outputs.

This ALU works by executing all instructions at the same time. This means that no matter what the op-code is, it adds and multiplies the numbers. Take for instance this example where we have two 4-bit inputs and a 4-bit output.

```
R1[0 .. 3] = add(A, B);
R2[0 .. 3] = mult(A, B);
```

R1 is the result of the addition of **A** and **B**. **R2** is the result of the multiplication of **A** and **B**. If we want the output to be **R**, we can use a multiplexer to select the correct output. If the op-code is determined by **S** where if **S** = 0, the output is **R1** and if **S** = 1, the output is **R2**.

```
Multiplexer(R1(0), R2(0), S, R(0))
Multiplexer(R1(1), R2(1), S, R(1))
Multiplexer(R1(2), R2(2), S, R(2))
Multiplexer(R1(3), R2(3), S, R(3))
```

The multiplier should multiply two 16-bit numbers. That means that the result could require 32 bits. In order to take care of this problem, the status register should indicate an overflow condition if the result is greater than 16 bits.

In this project, the propagation delay for logic gates in ALU needs to be taken into consideration. Please assign 10 ns for each logic operation, AND, OR, NOT,…etc, as the propagation delay so that you can estimate how long it will take for the arithmetic operations. However, the multiplexer will be considered to work ideally without having propagation delay.

Grading/Deliverables

Screenshots must be readable!

- Implement the ALU (15 pts). Generate a testbench and show that it works for addition, subtraction, multiplication, passthrough A and passthrough B by including screenshots. In the report, you also need to explain how you design your ALU, such as, circuit, etc.
- Implement the 8 x 16 bit register file (15 pts). Generate a testbench and show that you can store and retrieve 16 bit values. Generate a testbench and show that it works using a readable screenshot. In the report, you also need to explain how you design your register file.
- Implement the instruction decoder (15 pts). Generate a testbench which shows it operating and include a readable screenshot. In the report, you also need to explain how you design your instruction decoder, such as circuit, etc.
- Implement the 256 address x 16 bit RAM (20 pts). Generate a testbench and show that you can store and retrieve values from it. Include a screenshot. In the report, you also need to explain how you design your RAM.
- Show the processor works (20 pts). Generate a testbench to run the above program. At the end of the program, what are the values in $r6 and $r7? Write these values in your lab report.
- Implement the instruction memory and instruction fetch cycle (15 pts). Generate a testbench and show that it works by using a screenshot.
- Answer following questions for ALU using 16 bit adder, 16 bit multiplier, 16 bit multiplexer and gates
    - How long does it take to add 1000 to 2000? You should show the timing diagram and explain it.
    - How long does it take to multiply 128 * 128? Show the timing diagram* and explain it.
    - Using the results from 1 and 2, how many additions and multiplications per second can your ALU perform?
    - Describe some ways that you could optimize the design to make it faster.

The deliverables for this project are the report and the code. Your code should be put into a zip file and sent to the instructor through email (can share via Google Drive, Dropbox, etc..) and via Canvas. The report should be submitted on the Canvas portal for the course.

- Report Sections:
    - Cover Page (University, Department, Class, Semester, Instructor, Project #, Your name, Signed Honor Pledge)
    - Table of contents
    - **Introduction** (up to 2 paragraphs describing the project, purpose, and expectations.
    - **Background** (Assume the reader has no/limited knowledge on the subject matter and explain the project in much higher detail than in the Introduction).
    - **Preliminary** (describe the workspaces that was done to prepare for the project, i.e. VHDL analysis, etc.).
    - **Project** (The core of the report, describe your contributions, include flow-charts, pseudo code for workspaces, all that was done in order to get results).
    - **Results** (Record observed results from executing the processor program.)
    - **Conclusion** (Describe the meaning of the observed results, if purpose and expectations were met, draw conclusions, and what you would do differently if you had to do it all over again.)
    - **Appendices** (If needed, include source code and any additional reference materials).