ELEC 2602. A Processor

20 Semester 1 Project
Due: Week 13, your final lab session

Completion of this project is likely to require working from home as well as during lab sessions. Most of the design work can/should be completed in simulation. The tutors can provide guidance and advice, but they will not tell you how to implement your project. You must demonstrate your final project to your tutors during your final lab session (or before).

Problem
Your task is to implement a processor that can execute the following instructions:

Operation	Function performed	Description
load Rx, *D*	Rx ← *D*	Load constant value D into Rx
mov Rx, Ry	Rx ← [Ry]	Move contents of Ry into Rx
add Rx, Ry	Rx ← [Rx] + [Ry]	Add contents of Rx and Ry and put result into Rx
xor Rx, Ry	Rx ← [Rx] xor [Ry]	Bitwise XOR contents of Rx and Ry and put result into Rx

The load instruction allows an n-bit constant to be loaded into a register (it is recommended to start with n=3. Once you add memory, extend this to n=8 or n=16)
The mov instruction allows data to be copied from one register to another.
The add and xor instructions perform addition and bitwise XOR functions on the two operands and places the result back into the first operand.
The ldpc and branch instructions alter the flow of a program. The ldpc instruction stores the current address of the program counter (which points to a position in the memory). This can be used to store an address. As an example, you may wish to branch to a procedure, run the procedure, and then branch back to the previous address in program memory before the branch.

The number of bits used to encode your instructions is up to you. It is recommended that you use at least 3 or 4 instruction bits so that you can add additional instructions later. The number of registers in the processor is up to you.

Do not leave the project until the last minute!

Instructions/Advice (before any coding):

1. Listen to the lectures that describe a basic processor!

2. Make a block diagram of the datapath of your processor, showing all of the registers, arithmetic operations and multiplexers needed to implement your design.

3. Write a table showing all the control signals required in your datapath.

4. Complete a state diagram for a FSM to control your datapath (a similar design is provided in lectures). Note that some instructions may take more than one clock cycle to complete.

5. Complete a state transition table to generate the next-state as a function of the input signals for your FSM. (note that the next is a function of the current state and the codes representing the load/mov/add/xor instruction). Also show the outputs for each state in this table.

6. Draw a complete hierarchical circuit diagram of your processor (datapath + state machine). A separate figure should be drawn for each distinct entity/architecture pair that you intend to use in your Verilog design and all pinouts and their connections should be shown. Your top-level pin assignments should include *clock, reset*, and a n-bit data input.

7. **Divide out work**. Given the available time, you are highly encouraged to divide the work into FSM and Datapath sections. You should then put these separate sections together. Demonstrators will encourage you to do this.

Instructions (Coding):

1. Implement each individual block in Verilog. To help demonstrators and collaborate within your group, please ensure your FSM and Datapath are separate modules. Each should consist of several test modules.

2. Make a simple design first – follow the mark scheme!

3. Once complete, enhance your processor to use memory to run programs. You will create an internal synchronous RAM block for the processor. The RAM will need input ports for *clock*, write enable, 16-bit data input, write address, and read address. You can hard-code your program in this RAM to begin. It will need an output port for reading a 16-bit data output.

   The size of the internal memory is up to you. Remember, there are speed tradeoffs involved when implementing a larger memory. Also, your address width is dependent on the size of your memory.

4. Finally, modify the processor to include one extra register as the *program counter*, which will keep track of your instruction address in memory.

5. Load (or hardcode) a program into your processor by changing the RAM Verilog so that the memory is initialized to values corresponding to your program code.

6. Ideally, your final design should use Use Key(0) as a clock signal and create a circuit to use the HEX(0) to display the contents of the register chosen by the switches (if Switches = 0, Hex(0) displays the hex value for Reg(0), if Switches = 1, Hex(0) displays the hex value for Reg(1)

Marks

Marks will be awarded based on demonstrations to the tutors in the laboratory sessions, a micro report and your relative contribution to peers.

Marking Scheme

1. 10 Marks (FSM in simulation): Be able to demonstrate a FSM that performs state transitions for appropriate processor instructions. You do not need to create any output signals. (Note that this should be less difficult than the first part of lab 6). You must draw your FSM and it must match the drawing.

2. 5 Marks (FSM outputs in simulation): Add output logic to your FSM above to create relevant register enable signals and bus control signals. You must be able to explain to a demonstrator what these signals are meant to do (they do not necessarily have to be correct, but at least along the correct lines). (Again, this should be not much more difficult than lab 6). You must draw your FSM and it must match the drawing.

3. 5 Marks (Datapath in simulation): You can demonstrate that using control signals, you can load data from an external input data to a register, using the shared bus. The control signals should be provided by a testbench (so you do not need a FSM for this stage). You only need a bus, an input signal and one register. You must draw your datapath and it must match the drawing.

4. 5 Marks (Datapath in simulation): You can demonstrate that you can perform load, move and add instructions. Once again, the control signals should be provided by a testbench. Here you only need a bus, an input signal and two registers.

5. 5 marks (Integration in simulation): Show your FSM and Datapath can work together to execute a simple program that is specified in your testbench

6. 5 Marks (Four operations in simulation): Show basic operations work

7. 5 Marks (Processor consists of at least 8 16-bit registers)

8. 10 marks (Can access instruction memory and execute a basic program with program counter)

9. 10 marks (Works on FPGA)

10. 15 marks: (Micro Report): To allow you to focus on implementation, the report is simplified to a Hierarchical Datapath Diagram with brief description and FSM State diagram with brief description. You will use this to describe how your circuit works to your lab demonstrator in Week 13. It should be 1-2 pages max. **It is assessed during your lab session.**

(Max total marks: 75 - Distinction)

11. In order to receive further marks, you must suggest improvements to a processor and post them on Ed. You should discuss why you think they are valuable. After discussion, the proposed improvements will be put in one of the four categories based on difficulty/importance (category 1 – least difficult, category 4, most difficult):
    a. The first person to propose, justify and implement a unique item in Category 2, 3, and 4 will get 2, 2 and 1 bonus marks boost for coming up with an idea.

Category 1	Category 2	Category 3	Category 4	Extra marks
1	0	0	0	3
3	1	0	0	5
5	2	1	0	10
5	3	2	0	15
5	3	2	1	20