Designing a simple RISC Processor

INTRODUCTION

1. LEARNING OBJECTIVE

- Practice using Verilog HDL to design digital logic circuits.
- Practice skills in analysing and designing digital circuits at different levels: function-level, module-level, etc. and enhance creative & critical thinking skills.
- Foster the spirit of self-learning and researching through attaining related knowledge in Digital System Design and related fields during the course of the project.

2. GENERAL REQUIREMENTS

- Work in groups (maximum 5 members each).
- Project report: Using LaTeX is mandatory. Submit a hardcopy of your report during presentation day. The content of the report should include:
  - **Part 1**: Introduction. Include an introduction to your work, tools and equipment used, as well as the features of the final product.
  - **Part 2**: Theoretical overview. Include a brief theoretical description of the project.
  - **Part 3**: Design. Include block diagrams, functional blocks and a description of each block. Flow charts detailing the workflows of the system may also be included.
  - **Part 4**: Implementation. Include the process through which the design is realized (through diagrams, circuits, design models, etc.).
  - **Part 5**: Testing and Evaluation. Include simulation results as well as the intended test cases and the rationale behind them. Evaluate results on resource, time and power usage.
  - **Part 6**: Conclusions. Include a self-assessment of the work completed in the project and a discussion on possible improvements. Afterward, difficulties encountered during the project may be stated and a table detailing the work assigned and done by each member is required.
- Students may suggest changes within the project's specific requirements to better fit with their group's design or to improve on usability, etc. Discuss with your instructor on such changes.

3. GRADING CRITERIA

End product-wise:

- Basic requirements of the assignment should be met.
- Ease of usage and UX can be considered.
- Additional functionalities as put forward by the group are also taken into consideration.

Technical Criteria:

- The design is well thought-out and reasonably modularised.
- Well-written source code (in terms of hardware conventions and coding style, etc.)

4. REFERENCES

- Lecture slides
- Lab exercises and their guides.
- The Internet

TOPICS

Topic list

1. Designing a simple RISC Processor

1.1. Overview

RISC (Reduced Instructions Set Computer) is a computer architecture designed to keep the each individual instruction to the computer simple. In this assignment, we will design a simple RISC processor with 3-bit opcodes and 5 bits of operand width. This means that there will be 8 types of instructions and 32 separate addresses.
The processor will work on the basis of a clock and reset signal. Its operation will stop upon reaching the HALT instruction.

1.2. Project Requirements

Apply your what you've learned so far to design a simple RISC processor with tools from Cadence.

Your system needs to have the following functional blocks:

- **Program Counter**: register to store the program address
- **Address Mux**: allow for the selection of the program address or instruction address
- **Memory**: load and store data for the program
- **Instruction Register**: process instruction data
- **Accumulator Register**: process data from the ALU
- **ALU**: process data from Memory, Accumulator and the opcode of the Instruction,

Each functional block should have their own Verilog testbench.

You system must also be able to do the following:

- Load instructions from Memory
- Decode said instructions
- Retrieve operand data from Memory if needed
- Execute the instructions and perform arithmetic and logical operations as needed
- Store the result back to Memory or Accumulator
- Repeat the process until the system halts

1.3. Detailed information on the Functional blocks

Program Counter

- The Program Counter (PC) keeps track of the current instruction of the program and also the current state of the program.
- The PC must activate at the rising edge of clk.
- An active-HIGH $reset$ that returns the PC to 0 must be added.
- The bitwidth of PC here should be 5.
- An arbitrary number can be loaded into the PC if requested. Otherwise, the PC increments as usual.

Address MUX

- The Address MUX block multiplexes the instruction address during the instruction loading phase and the operand address during the instruction execution phase.
- The default bitwidth of the MUX here should be 5, though it should be parameterised for reusability.

ALU

- The ALU carries out calculations. The operation performed is dependent on the opcode.
- The ALU can perform 8 operations on 8-bit operands (inA và inB). The output will be an 8-bit result and 1 is_zero bit
- is_zero is asynchronous and indicates whether input inA is 0 or not.
- The 3-bit opcodes of the processor are give in the table below:

Opcode	Code	Operation	Output
HLT	000	Halt the program	inA
SKZ	001	First check if the output of the ALU is 0. If TRUE then the next instruction is skipped, otherwise the programs continue as usual	inA
ADD	010	Add the value stored in the Accumulator with the value from the memory address given in the instruction and then store the output back to the accumulator	$inA + inB$
AND	011	Perform a logical AND between the value stored in the Accumulator with the value from the memory address given in the instruction and then store the output back to the accumulator	inA & inB
XOR	100	Perform a logical XOR between the value stored in the Accumulator with the value from the memory address given in the instruction and then store the output back to the accumulator	$inA \oplus inB$
LDA	101	Load the value at the address given in the instruction to the Accumulator	inB
STO	110	Store the value currently in the Accumulator at the address given in the instruction	inA
JMP	111	Unconditional jump. The program jumps to the target address given in the instruction and continues with the next instruction as usual	inA

Controller

- The Controller handles the control signals of the processor, including loading and executing instructions.
- The Controller must work at the positive edge of clk.
- The rst signal is synchronous and active-HIGH.
- The 3-bit opcode input corresponds with the ALU.
- The Controller can have any of the below 7 outputs:

Output	**Function**
sel	select
rd	memory read
ld_ir	load instruction register
halt	halt
inc_pc	increment program counter
ld_ac	load accumulator
ld_pc	load program counter
wr	memory write
data_e	data enable

- The Controller has 8 working states in 8 consecutive *clk* cycles in the below order: $INST\_ADDR, INST\_FETCH, INST\_LOAD, IDLE, OP\_ADDR, OP\_FETCH$, $ALU\_OP, STORE$. The reset state is $INST\_ADDR$.
- The Output of the Controller is dependent on the current state and opcode like so:

Outputs	**Phase**								**Notes**
	INST_ ADDR	INST_ FETCH	INST_ LOAD	IDLE	OP_ ADDR	OP_ FETCH	ALU_ OP	STORE	
sel	1	1	1	1	0	0	0	0	ALU OP = 1 if opcode is ADD, AND, XOR or LDA
rd	0	1	1	1	0	ALUOP	ALUOP	ALUOP	
ld_ir	0	0	1	1	0	0	0	0	
halt	0	0	0	0	HALT	0	0	0	
inc_pc	0	0	0	0	1	0	SKZ & & zero	0	
ld_ac	0	0	0	0	0	0	0	ALUOP	
ld_pc	0	0	0	0	0	0	JMP	JMP	
wr	0	0	0	0	0	0	0	STO	
data_e	0	0	0	0	0	0	STO	STO	

Register

- The input should be 8 bits wide.
- The *rst* signal is synchronous and active-HIGH.
- The Register must activate at the positive edge of the *clk* signal.
- When there is a *load* signal, the input value is outputted on the other side, otherwise the output value is unchanged.

Memory

- The Memory store both the instructions và data.
- The Memory must be designed using *Single bidirectional data ports* such that reading and writing at the same time is not possible.
- The memory address is 5 bits wide and each address stores 8 bits of data.
- The should be a 1-bit signal that enables reading and writing on the memory.
- The Memory must work at the positive edge of *clk*.

1.4. Suggested extra work

- Apply knowledge from the Computer Architecture course to improve the processor and handle Hazards
- Add more functionalities to the ALU (fix/floating point arithmetic, multiplier, divider, etc.)
- Others. Student can come up with their own extra goals.