COE 405
Design & Modeling of Digital Systems

Course Project – Term 162
Design and Modeling of a RISC Processor
Due on: Thursday May 25, 2017

In this project, you will design, model in Verilog and synthesize a 16-bit MIPS-like processor. The details about the processor are given below.

Instruction Set Architecture

In this project, you will design a simple 16-bit MIPS-like processor with seven 16-bit general-purpose registers: R1 through R7. R0 is hardwired to zero and cannot be written. There is also one special-purpose 12-bit register, which is the program counter (PC). All instructions are 16 bits and there are three instruction formats: R-type, I-type, and J-type as shown below:

R-type format

4-bit opcode (Op), 3-bit register numbers (Rs, Rt, and Rd), and 3-bit function field (funct)

Op^4	Rs^3	Rt^3	Rd^3	$funct^3$

I-type format

4-bit opcode (Op), 3-bit register numbers (Rs and Rt), and 6-bit immediate constant

Op^4	Rs^3	Rt^3	$Immediate^6$

J-type format

4-bit opcode (Op) and 12-bit immediate constant

Op^4	$Immediate^{12}$

For R-type instructions, Rs and Rt specify the two source register numbers, and Rd specifies the destination register number. The function field can specify at most eight functions for a given opcode. Opcodes 0 and 1 are reserved for R-type instructions.

For I-type instructions, Rs specifies a source register number, and Rt can be a second source or a destination register number. The immediate constant is only 6 bits because of the fixed-size nature of the instruction. The 6-bit immediate constant is assumed to be sign-extended for all instructions.

For J-type, a 12-bit immediate constant is used for J (jump), JAL (jump-and-link), and LUI (load upper immediate) instructions.

Instruction Encoding

Sixteen R-type instructions, eleven I-type instructions, and three J-type instructions are defined. These instructions, their meaning, and their encoding are shown below:

Instr.	Meaning	Encoding				
ADD	Reg(Rd) = Reg(Rs) + Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 000
SUB	Reg(Rd) = Reg(Rs) – Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 001
AND	Reg(Rd) = Reg(Rs) & Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 010
OR	Reg(Rd) = Reg(Rs) | Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 011
NOR	Reg(Rd) = ~(Reg(Rs) | Reg(Rt))	Op = 0000	Rs	Rt	Rd	f = 100
XOR	Reg(Rd) = Reg(Rs) ^ Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
SLT	Reg(Rd) = Reg(Rs) signed< Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 110
SLTU	Reg(Rd) = Reg(Rs) unsigned< Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 111
SLL	Reg(Rd) = Reg(Rs) << Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 000
SRL	Reg(Rd) = Reg(Rs) zero>> Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 001
SRA	Reg(Rd) = Reg(Rs) sign>> Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 010
ROL	Reg(Rd) = Reg(Rs) rotate<< Reg(Rt)	Op = 0001	Rs	Rt	000	f = 011
ROR	Reg(Rd) = Reg(Rs) rotate>> Reg(Rt)	Op = 0001	Rs	Rt	000	f = 100
LW	Reg(Rt) = Mem(Reg(Rs))	Op = 0001	Rs	Rt	Rd	f = 101
SW	Mem(Reg(Rs)) = Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 110
JR	PC = lower 12 bits of Reg(Rs)	Op = 0001	Rs	000	000	f = 111
ADDI	Reg(Rt) = Reg(Rs) + ext(im^6)	Op = 0010	Rs	Rt	Immediate^6	
ANDI	Reg(Rt) = Reg(Rs) & im_6	Op = 0011	Rs	Rt	Immediate^6	
ORI	Reg(Rt) = Reg(Rs) | im_6	Op = 0100	Rs	Rt	Immediate^6	
SLTI	Reg(Rd) = Reg(Rs) signed< ext(im^6)	Op = 0101	Rs	Rt	Immediate^6	
SLTIU	Reg(Rd) = Reg(Rs) unsigned< im^6	Op = 0110	Rs	Rt	Immediate^6	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 0111	Rs	Rt	Immediate^6	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 1000	Rs	Rt	Immediate^6	
BLTZ	Branch if (Reg(Rs) < 0)	Op = 1001	Rs	Rt	Immediate^6	
BLEZ	Branch if (Reg(Rs) ≤ 0)	Op = 1010	Rs	Rt	Immediate^6	
BGTZ	Branch if (Reg(Rs) > 0)	Op = 1011	Rs	Rt	Immediate^6	
BGEZ	Branch if (Reg(Rs) ≥ 0)	Op = 1100	Rs	Rt	Immediate^6	
J	PC = Immediate^{12}	Op = 1101	Immediate^{12}			
JAL	R7 = PC + 1, PC = Immediate^{12}	Op = 1110	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1111	Immediate^{12}			

There are three shift and two rotate instructions. For shift and rotate instructions, the least significant 4 bits of register Rt are used as the shift/rotate amount. The Load Upper Immediate (LUI) is of the J-type to have a 12-bit immediate constant loaded into the upper 12 bits of register R1. The LUI can be combined with ORI (or ADDI) to load any 16-bit constant into a register. Although the instruction set is reduced, it is still rich enough to write useful programs. We can have procedure calls and returns using the JAL and JR instructions.

Memory

Your processor will have separate instruction and data memories with $2^{12} = 4096$ words each (this is the maximum that can be supported under the current version of Logisim). Each word is 16 bits or 2 bytes. Memory is *word addressable*. Only words (not bytes) can be read and

written to memory, and each address is a word address. This will simplify the processor implementation. The PC contains a word address (not a byte address). Therefore, it is sufficient to increment the PC by 1 (rather than 2) to point to the next instruction in memory. Also, the Load and Store instructions can only load and store words. There is no instruction to load or store a byte in memory.

Addressing Modes

For branch instructions (BEQ, BNE, BLTZ, BLEZ, BGTZ and BGEZ), PC-relative addressing mode is used: PC = PC + sign-extend(immediate^{6}). For jump instructions (J and JAL), direct addressing is used: PC = Immediate^{12}. For LW and SW instructions, base-addressing mode is used. The base address in register Rs contains the memory address.

Program Execution

The program will be loaded and will start at address 0 in the instruction memory. The data segment will be loaded and will start also at address 0 in the data memory. You may also have a stack segment if you want to support procedures. The stack segment can occupy the upper part of the data memory and can grow backwards towards lower addresses. The stack segment can be implemented completely in software.

To terminate the execution of a program, the last instruction in the program can jump or branch to itself indefinitely.

Application Program

You need to develop an application program that randomly generates two randomly generated 2-git decimal numbers and randomly generates a required operation to be performed by the user which is either addition or subtraction and displays these numbers on the LCD screen. For example, the program will display 15 + 05 = ?. Then the program will wait for the user to enter the required input. The program will display the number entered by the user on the LCD screen followed by either Correct or Incorrect and then displays the correct result. You need to interface your CPU with a keypad to enter the required result.

Project Report

The report document must contain sections highlighting the following:

1 – Design and Implementation

- Specify clearly the design giving detailed description of the datapath, its components, control, and the implementation details (highlighting the design choices you made and why, and any notable features that your processor might have.) Document clearly design alternatives explored and why a given design is selected.
- Provide drawings of the component circuits and the overall datapath.
- Provide a complete description of the control logic and the control signals. Provide a table giving the control signal values for each instruction.
- Use a hierarchical Verilog modeling style when modeling your processor. Your CPU should be composed of a control unit and datapath. The datapath should be composed of ALU, Register file, NextPC Block, Instruction Memory, Data Memory and other necessary components.
- Provide list of sources for any parts of your design that are not entirely yours (if any).
- Carry out the design and implementation with the following aspects in mind:
  - Consider alternative design solutions and justify your design selection
  - Correctness of the individual components

- Correctness of the overall design when wiring the components together
- Completeness: all instructions were implemented properly.

2 – Simulation and Testing

- Carry out the simulation of the processor developed using Modelsim or Isim.
- Test each of the components individually and demonstrate its correct operation including the ALU and register file.
- Describe the test programs that you used to test your design with enough comments describing the program, its inputs, and its expected output. List all the instructions that were tested and work correctly. List all the instructions that do not run properly.
- Test the correct functionality of your CPU by implementing the **selection sort procedure** along with **Max procedure**. Use this procedure to sort an array of 8 words of your choice.
- Also provide snapshots of the Simulator window with your test program loaded and showing the simulation output results.
- Synthesize the processor on FPGA and demonstrate its correct functionality by correct implementation of the calculator application.

3 – Teamwork

- This project is a team work project with **two** students per team. Make sure to write the names of all the group members on the project report title page.
- Each group should assign a group leader that leads the conduction of the project, divide the project tasks among the team members.
- Project tasks should be divided among the group members so that each group member contributes equally in the project and everyone is involved in all the following activities:
  - Design and Implementation
  - Simulation and Testing
  - Synthesis and FPGA implementation
  - Design and results reporting
- Come up with a project plan detailing the tasks to be performed in the project, their planned start and finish dates and the team member primarily responsible for performing the task. Submit the project plan by **Thursday April 20**.
- Clearly show in the report the work done by each group member, and how the work deviated from the proposed plan.
- Each group member will be evaluated based on his contribution in the project. Thus, it is expected that each group member could have a different mark in the project.
- Students who **help** other team members should mention that to earn credit for that.

Submission Guidelines

All submissions will be done through WebCT.

Attach one zip file containing all Verilog files used in your design, a video demo of your project, as well as the report document.

Submit also a hard copy of the report during the class lecture.

Grading policy

Grading Criteria	Mark
Project Plan	5
Demonstration of correct functionality of components and whole processor design by simulation	60
Demonstration of correct functionality of Required Application on FPGA	20
Project Documentation and Report Organization	15
Total	100

The project will be evaluated based on the final report, project demonstration and oral project evaluation with all team members. Note that all team members will be responsible and will be evaluated based on the final outcome of the project.

Deadlines

Task	Deadlines
Project Plan	Sunday, April 16
Progress Report Submission Demonstrating Correct Functionality of CPU Design	Sunday, May 14
Final Report Submission Demonstrating Correct Functionality if Application Program Development	Thursday, May 25
Total	100