CO224 – Computer Architecture

Labs + Project

Goal

The goal of this is to walk you through the design and implementation of a 16-bit pipelined RISC micro-processor that follows *Computer Organization & Design*. The overall project is broken into seven milestones (you will have to do only 4, the rest are for those who need challenging stuff). The strategy of the milestones are to minimize the complexity of the overall CPU design by successively forming more complex components from earlier, smaller and simpler components. Using this strategy, you will reach the final pipelined CPU in an orderly and easily achievable fashion. This strategy also introduces a structured design approach in defining components and interfaces and an orderly integration of simpler components into more complex components. In general, the milestones follow the bottom-up design approach followed in the textbook, starting with primitive components and then "gluing" them together into a more complex set of subsystems and systems.

Complete RISC Microprocessor Specification

The complete instruction set architecture (ISA) for the RISC microprocessor is given in the table below.

Instruction	Pseudo description	Op_code 4 bits	Rs 4 bits	Rt 4 bits	Rd 4 bits
ADD R_d, R_s, R_t	$R_d := R_s + R_t$	2	0–15	0–15	0–15
SUB R_d, R_s, R_t	$R_d := R_s - R_t$	6	0–15	0–15	0–15
AND R_d, R_s, R_t	$R_d := R_s$ "•" R_t	0	0–15	0–15	0–15
OR R_d, R_s, R_t	$R_d := R_s$ "+" R_t	1	0–15	0–15	0–15
SLT R_d, R_s, R_t	if ($R_s < R_t$) $R_d := 1$ else $R_d := 0$	7	0–15	0–15	0–15
LW R_d, off(R_s)	$R_d := M[off + R_s]$	8	0–15	0–15	offset
SW R_d, off(R_s)	$M[off + R_s] := R_r$	A	0–15	0–15	offset
BNE R_s, R_t<offset>	if (R_s != R_t) pc := pc + off +4	E	0–15	0–15	offset
JMP <addr_off>	pc := pc:addr:0	F	12 bit offset		

The instruction set has been defined with sufficient number and types of instructions to allow you to write small, effective simulation and test programs. As shown in the table, all instructions are 16 bits in width. Offsets for loads and stores are 4-bit, 2s-complement numbers, and the absolute jump address is 12 bits. The jump address is computed as:

15:13	12 1	0
PC 15:13	addr_offset	0

The opcodes have been defined such that the lower three bits can be fed directly in the ALU as control lines. The ALU control lines and functions are shown below.

ALU Control Lines	Function
0 0 0	And
0 0 1	Or
0 1 0	Add
0 1 1	Subtract (beq)
1 1 1	Set-on-less-than

Hazard Avoidance

You should use the following two hazard avoidance techniques to simplify your design. These techniques should be guaranteed by the compiler or adhered to if hand assembling.

Avoidance Technique	Description
Load-Word Hazard Prevention	Guarantees that the next instruction will not use, as a source, the register being written by an immediately preceding load instruction
Conditional Branch Hazard	Guarantees that the three instructions following a conditional branch should always be executed

Milestone Descriptions

The seven milestones are listed below. Only the first 4 are compulsory and will be marked.

Laboratory	Name	Description
1	Verilog Components	Implement familiar components in Verilog to be used throughout the semester
2	Instruction Interpreter	Implement and simulate a behavioral instruction interpreter for the given ISA
3	Register File/ALU	Implement and simulate behavioral and structural models of a register file and ALU
4	Multicycle CPU	Integrate the Register File/ALU with a structural model of a data path and a behavioral model of a multicycle controller to form a complete multicycle CPU
5	Pipelined CPU	Implement a pipelined version of the multicycle CPU
6	Data Hazard and Forwarding Unit	Implement and integrate a data hazard detection and forwarding unit into the pipelined CPU
7	Cache Module	Implement a cache for the CPU

Milestone 1

The objective of this milestone is to provide you with experience in specifying behavioral and structural Verilog descriptions of familiar components.

Prelab

Problem 1

Develop a behavioral Verilog model for a 4-to-1 word multiplexer (MUX). Your model should work with arbitrary length words; i.e., you should not place hard constraints on the lengths of inputs and outputs. Develop a test bench for your MUX that demonstrates each function.

Problem 2

Develop a behavioral Verilog model for a 1-bit, 2-to-4 demultiplexer. Your device should include an ENABLE signal as well as normal inputs and outputs. Develop a test bench for your Verilog demultiplexer model that demonstrates basic functionality. Simulate your design to demonstrate correctness.

Lab 1 Exercise

Problem 3

Develop a behavioral Verilog model for a 4-bit shift register. Your shift register should implement functions for LOAD, HOLD, RIGHT SHIFT and LEFT SHIFT. In addition to regular inputs, your shift register should provide a SHIFT LEFT INPUT and a SHIFT RIGHT INPUT that input the value shifted into the right-most and left-most bits, respectively. Your register should also include an ENABLE input and a CLOCK input. Design your register to be a positive level-triggered device. Develop a test bench for your Verilog shift register that demonstrates each function.

Problem 4

Use your 4-bit shift register from Problem 3 to implement a structural Verilog model for an 8-bit shift register. This device should perform the same functions as the 4-bit shift register, but over 8 bits. Develop a test bench for your shift register that demonstrates each function.

Project

Problem 5

Modify your design from Problem 3 to implement a rising edge triggered device. If you designed your system carefully, this should involve changing exactly one line of Verilog. Modify your test bench from Problem 3 to demonstrate the distinction between the two shift registers.

Problem 6

Use your shift register model from Problem 5 and your MUX model from Problem 1 to develop a 4-bit Grey-code counter. Your counter should implement LOAD, HOLD, COUNT UP and COUNT DOWN functions. Your counter should also include an ENABLE input and a CLOCK input. Design your counter to be a rising edge-triggered device.