EC413 -- Computer Organization
Multicycle CPU Project -- Spring 2016

1. Introduction

This project requires you to

- Design and test a CPU that meets the set of requirements detailed below.
- Document the design, debug, and test of your work.

An important aspect of this project is that it is qualitatively more complex than previous labs. For example, you will be constructing multiple components that must work together. Some things to consider:

- ***Careful initial design will save huge amounts of time later.***
- Be sure to allocate enough time especially to assembling, debugging, and testing the system. In typical industrial projects, one sixth of the time is spent on design and construction, one third on initial debug, and one half on test.
- Because such an important part of engineering is to get systems working in a cost-effective manner, a substantial part of the grade is on the final demo.
- Note that it is possible to complete the project successfully supporting different sets of instructions (see end for reduced credit and extra credit options). If you are severely time constrained, a reasonable strategy might be to create a simple version first and add features as you have time.

2. Groups

You are strongly encouraged to work in groups (of size no more than 2). Let us know if you need help finding a partner. If you want to work by yourself, please talk with me first. You can let us know your group at the Milestone 1 demo.

3. Deliverables and Grading

Deliverables are as follows; details are given in later sections. Please note that the expectation is that you will have a completed and working system.

Design Review (required!) (20 points)	- Milestone 1, due 3/25
Controller (demo and documents)	- Milestone 2, due 4/1
Datapath (demo and documents) (25 points)	- Milestone 3, due 4/8
Completed System (demo and documents) (35 points)	- due 4/22
Overall Quality (15 points)	
Extra/Reduced Features (see below) (-10 to +15 points)	
Early finish bonus (5 points)	- due 4/15 or earlier

Note: There will be one overall grade for the project given upon completion. Failure to meet any milestone will be penalized for that section -- 10% for < 2 days and more thereafter.

Milestone 1: Design Review and Report (and group formation).
Review Dates: Week of 3/21 - 3/25 -- sign up for a time, instructions to follow.
Documents:

- Very brief description of your design
- Classification of instructions by type - which instructions will be sharing substantial logic
- Specifications of the state machines for each instruction type
- Specification of overall state machine - state transition diagram with next state logic shown, no control yet
- Schematic for the datapath - not all details need to be completed
- Comments on design choices

Note on design review: There is no working code to be demonstrated here, but ***you are required to meet with one of the TAs to go over your design on or before Friday 3/25.***

Milestone 2: Working controller with instruction fetch and decode logic.
Implement the 10 state Patterson & Hennessy Datapath Controller from your notes (see controller lecture notes) using Behavioral Verilog. Since the control lines have not yet been finalized this should be a simple task. The fetch logic should use the PC to index instruction memory, fetch the instruction into the IR, increment the PC, and use the opcode in IR to direct the execution of the state machine. Each instruction should execute correctly and in sequence with other instructions. Demo Date: Friday, 4/1.

Documents:

- Verilog code for the controller
- Test bench
- Test waveforms
- Description – enough to tie together the other documents with the idea that together your documents should make a convincing case that your controller will work with your datapath.

Milestone 3: Working datapath.
All instructions should be working when driven by a testbench. Instructions do not need (yet) to be working in conjunction with your Milestone 2 controller.
Demo Date: Friday, 4/8.
Documents:

- Verilog code for the Datapath
- Test bench
- Test waveforms
- Description – enough to tie together the other documents with the idea that together your documents should make a convincing case that your datapath will work with your controller.

Milestone 4: Working system.
Demo Date: Friday, 4/22.
Documents:

- Final Verilog code for the entire system
- Final test bench
- Final test waveforms
- Final versions of the documents handed in for the previous milestones
- Write-up the results from your working system. This may include changes made to your original design.

3. Design Requirements

The design is for a 32-bit CPU partitioned between a *multicycle datapath* and a *controller*.
Motivation: This CPU is similar to the multicycle CPU that we discussed in class. The advantage of this CPU as a project is that it requires both a significant controller and datapath which must interact correctly.

Overall

- The instruction and data sizes are 32 bits.
- The standard of correctness is that the final system should be able to execute machine language programs.
- The design should have a global reset to initialize the circuit to a known state.
- When initialized, your design should start executing a program starting at address 0 of instruction memory.

- You have substantial flexibility in specifying control.
- Instructions will necessarily take multiple cycles to execute.
- All parts can be implemented any way you like, including behavioral Verilog.
- The design is not pipelined (unless you discuss with the instructor for extra credit).

Memory
A 16-bit address line/32-bit data line pseudo-SRAM module will be provided. The interface will be identical to the register file (combinational read, latched/enabled write).

CPU
The features listed below are required. Other hardware (e.g. additional adders) should be created as necessary. Please note that the additional internal registers specified below must be used. This is to make the project more realistic and avoid having it devolve into a single cycle CPU.

- A register file with (32) 32-bit general purpose registers.
- The following additional internal CPU registers: 32-bit Program Counter (PC), 32-bit Instruction Register (IR), 32-bit Memory Data Register, ALUOut (for the output of the ALU), A and B (two outputs from the register file).
- An ALU.
- A controller.

Operation

- Assume that at start-up the registers hold random values; you can initialize them with LI and LUI instructions. LI is a real instruction in this CPU (not a pseudoinstruction).
- To make things easier, the instructions can be sequenced as 32-bit words. That is, the first instruction is at address 0 in instruction memory, the second at address 1, etc. Because of this, there is no need to left-shift the sign-extended immediate in the branch instructions.
- The size of the PC is an obvious mismatch with the size of the instruction memory provided (16 bit memory, 32 bit PC). This is because we chose a small memory to ensure faster synthesis. In this and all other analogous cases, ignore the unneeded high-order bits.
- The internal registers must be used, that is:
    - All memory reads and writes must go through the MDR Register (32 bits).
    - All register operands must go through A&B Register (32 bits).
    - All ALU outputs must go through ALUOut Register (32 bits).

Instruction Format
Instructions are 32-bit. There are three formats: *R-Type*, *I-Type,* and *J-Type*.

The instruction fields are laid out as follows:

Type	**31**		**format (bits)**		**0**
R	opcode(6)	r1(5)	r2(5)	r3(5)	
I	opcode(6)	r1(5)	r2(5)	imm(16)	
J	opcode(6)	imm(26)			

- Opcodes – For the project, the <u>most significant</u> two bits of the opcode specify the type of operation as follows:
    - *00* Jump
    - *01* Arithmetic/Logical R-type

    - *10* Branch (I-type)
    - *11* Arithmetic/Logical I-type
- The least significant 4 bits of the opcode are fed into the ALU selection.
- The opcode pattern of all zeroes is reserved for NOOP.

Some sample instructions:
LI of the immediate value FFFE into register 0 bits [15:0]:
32’b11100100000000001111111111111110 = 32’hE400FFFE
LUI of the immediate value FFFF into register 0 bits [31:16]:
32’b11101000000000001111111111111111 = 32’hE800FFFF
ADD register 0 and immediate -1 and put result into register 31:
32’b11001011111000001111111111111111 = 32’hCBE0FFFF
Branch forward in memory by 16 32-bit words if register 31 is equal to zero:
32’b10000011111000000000000000010000 = 32’h83E00010

Complete List of Instructions

Op3	**Op2**	**Op1**	**Op0**	**Op1**	**Op0**	**Output**	**Function Name**
0	0	0	0	0	0		NOOP
0	1	0	0	0	0	R1 = R2	MOV
0	1	0	0	0	1	R1 = ~R2	NOT
0	1	0	0	1	0	R1 = R2 + R3	ADD
0	1	0	0	1	1	R1 = R2 - R3	SUB
0	1	0	1	0	0	R1 = R2 | R3	OR
0	1	0	1	0	1	R1 = R2 & R3	AND
0	1	0	1	1	0	R1 = R2 ^ R3	XOR
0	1	0	1	1	1	R1 = 1 if R2 < R3, else 0	SLT
0	0	0	0	0	1	PC 🡐 PC[31:26] || Limm	J

1	0	0	0	0	1	IF (R1 != R2)
						THEN PC 🡐 PC + SE(Imm)
						ELSE PC 🡐 PC + 1
							BNE

1	1	0	0	1	0	R1 = R2 + SE(Imm)	ADDI
1	1	0	0	1	1	R1 = R2 – SE(Imm)	SUBI
1	1	0	1	0	0	R1 = R2 | ZE(Imm)	ORI
1	1	0	1	0	1	R1 = R2 & ZE(Imm)	ANDI
1	1	0	1	1	0	R1 = R2 ^ ZE(Imm)	XORI
1	1	0	1	1	1	R1 = 1 if R2 < SE(Imm), else 0	SLTI
1	1	1	0	0	1	R1[15:0] 🡐 ZE(Imm)	LI
1	1	1	0	1	1	R1 🡐 M[ZE(Imm)]	LWI
1	1	1	1	0	0	M[ZE(Imm)] 🡐 R1	SWI

Notes: Unless otherwise noted, all instructions advance the PC by 1. SE: Sign Extended. ZE: Zero Extended.

Reduced Credit (minus 10 points)
If you are likely to be time constrained you may do a project without BNQ and J.

Extra Credit (up to 15 points)

Extra credit is possible for additional instructions or complexity. BNQ and J must be completed before extra credit will be granted.

Possible extra instructions: BLT, BLE, LW, SW, JAL. Example of complexity: exception handling (see instructor first!).