**Spring 2011**

**ELEC 5200/6200**

**CPU Design Project**

**Assigned January 28, 2011**

A RISC CPU is to be designed in the VHDL/Verilog HDL modeling language, verified via Mentor Graphics "ModelSim" simulator and implemented on the DE2 FPGA board from Altera using Quartus II software. The project consists of five parts as defined below along with their due dates. It is advised that you read problem definitions of all five parts before actually starting with part-1, i.e., Instruction Set Architecture (ISA). Please submit only the **List Format** (do not submit wave format of the results) of the simulations in part 3 and part 4. Always **annotate** your simulation results. Maintain a single 1” folder for submitting the project parts. When submitting a later part, all the previous parts need to be in the folder.

**CPU Design Project – Part 1 – ISA, Report Due Feburary 11, 2011**

An instruction set architecture (ISA) for a new microprocessor (μP) is to be designed. The μP will be designed and modeled in VHDL/Verilog in later parts. Your ISA is to be designed using RISC design principles, with primary design goals being low cost and a minimal number of clock cycles per instruction. Following are the requirements for your ISA.

1. The ISA may contain no more than 16 unique instructions. However, you may have multiple formats for a given type of instruction, if necessary.
2. Of the 16 instructions, at least one instruction should make your processor **HALT.**
3. Since it’s a small processor, you need not use the pointers like stack pointer, global pointer, etc.
4. The ISA is to support 16-bit data words only. (No byte operands.)

 a. All operands are to be 16-bit signed integers (2’s complement).

 b. Each instruction must be encoded using one 16-bit word.

1. The ISA is to support linear addressing of 1K, 16-bit words memory. The memory is to be word-addressable only - **not byte-addressable**.
2. The ISA should contain appropriate numbers and types of user-programmable registers to support it.
3. The ISA must “support” the following C Programming Language constructs:
* Assignment operator: variable = expression;
	+ Expressions must support only two arithmetic operators:
	+ add (+) and subtract (-). Multiply(\*) and divide(/) **NOT** necessary to implement.
	+ Expressions must support Logical operators: And and Or.
	+ Data are limited to:
		- 16-bit two’s-complement integers (Example: int a;)
		- One-dimensional integer arrays (Example: int a[10];)
* Control flow structures: “if-else” structures, “while” loops, “for” loops
	+ These should support the six standard relational operators:
	+ ==, !=, >, <=, <, >=
* Functions (call and return), with parameters able to be passed by value or by reference.

Provide the following information about your ISA:

1. List and describe the user-programmable registers.
2. List and describe the different instruction formats used.
3. For each instruction in your instruction set, list the following:
* Assembly language for each form of the instruction - mnemonic and operands
* Machine language for each form of the instruction: instruction code format, op-code, and operand encoding
* Justification for including each form of the instruction in your ISA

4) For each C construct listed in item 6 above, provide an example showing how the construct would be “compiled”, i.e. implemented with your instruction set, by writing an example of the C construct and the corresponding assembly language implementation.

**CPU Design Project – Part 2 – Datapath, Report Due March 4, 2011**

In this part, you have to design the datapath of a CPU that will realize the instruction set architecture (ISA) designed in the previous part (including any “adjustments” made to the ISA). Include the following in your submission.

1. A block diagram (register level) of the datapath, with all components and control signals clearly labeled.
2. A description of the function of each component in the datapath.
3. For each instruction of your ISA, list the register transfers, or sequence of register transfers, required to fetch and execute the instruction. Register names should correspond to components in your datapath diagram.
4. A discussion of the tradeoffs and other design decisions made in developing your datapath. This should include:
* Cost vs. speed tradeoffs that you considered.
* Why you chose a single-cycle or multi-cycle design.
* Decisions related to “shared” and/or “dedicated” components.
* Selection of edge-triggered vs. latching registers.
* Other decisions that were considered.

**CPU Design Project- Part 3-Datapath Verification, Report Due March 11, 2011**

Develop and verify a VHDL/Verilog model of the datapath of your CPU, as described in the block diagram and register transfers defined in Part 2. The CPU must be capable of working with at least a single memory outside the CPU (you might need another instance of the memory if your datapath has a separate data and instruction memory); the memory for Logic Simulation in ModelSim will be added in Part 4, which will be the RAM block in the Altera Megafunctions Library. The datapath must have at least three 16-bit external “ports” to connect the CPU to the memory: a Read data bus, A Write data bus (16-bit) and an address bus (10-bit). The datapath must also have the various control and status signals as external “ports”. Apart from those the datapath also needs to have the input “inr” used as a multiplexor to decide the register number and the output “outvalue” which is used to display the contents of that register in the final stage on the FPGA board. These will be connected to a Control Unit in the next part.

Notes:

1. This is to be a register-transfer-level (RTL) design (not gate level).
2. Refer the LeonardoSpectrum guide for [Altera HDL synthesis manual](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Spr09/PROJECT/VHDLSynthesisGuide.pdf) to write the VHDL/verilog code according to the synthesis guidelines so that in the final stage, your design is synthesized correctly by the FPGA.
3. The top-level design should contain **only component instantiations**, matching your block diagram (changes may be made to the diagram as necessary).
4. Design and **test** VHDL/verilog models of **each unique component** used in your datapath.
5. Create a **table / state diagram** listing all control signals and the values of each control signal required for the different clock cycles like fetch, decode, execute, etc.
6. You will have to **submit the control signal table, the VHDL/verilog code and Simulation List of all the components in the datapath as well as the VHDL/verilog code the complete top level Datapath.**

**Major Datapath Components Likely to be needed:**

1. **ALU**: The ALU must provide all arithmetic and logic functions required to support your instruction set. It should not provide unnecessary functions.
2. **Register file**: Design as a multi-port “memory array”. DO NOT instantiate individual registers. You might want to refer the [register file](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Fall10/PROJECT/regfile.vhd) code provided on the course website.
3. **Sign/zero extension logic**, as appropriate, for ALU inputs.
4. **Program counter (PC).**
5. **Instruction register** (**IR**) (if required).
6. Assorted **multiplexers** for data paths and register address inputs.

**As you may recall, in Homework 3,** we have already ALU. PC is nothing but register with a ‘write enable’.

**Thoroughly simulate each new component** individually, before inserting it into the datapath. **Annotate and submit each simulation**.

**You need not submit the simulation results for the ALU since you have already simulated it in Homework 3.**

**CPU Design Project – Part 4 – Control Unit, Report Due April 1, 2011**

1. Design and test a VHDL/verilog “behavioral” model of the **control unit** to realize the behavior described in your Control Signal Table from the previous part of the project. Submit the VHDL/verilog code and simulation results of the Control Unit.
2. Simulate the datapath component, verifying all required register transfers, by applying control signals through the control unit created above **and show in the simulation where you verified each required register transfer for the CPU.** (If some register transfers are common to multiple instructions, you do not need to show them separately for every instruction – but it might be a good idea to do so anyway.)

For this verification you will have to create a component which will include both datapath and the control unit connected to each other. This component will be very similar to the top level CPU component you are going to create in step (6) except the memory. Choose a test program. Force the inputs of control unit(which are otherwise fetched from the program memory)as per the program so as to mimic the normal operation and test all the stages of operation.

1. Hand compile the chosen test program into binary code and then use this code and modify the given RAM\_init.mif file according to your program code. You need to submit your assembly language code and binary code of the test program.
2. Create a 16 bit memory module from Altera’s Megafunction Library as explained in ‘[Run time content editable memory tutorial’](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Spr10/PROJECT/Run_time_content_editable_memory_tutorial.pdf) file (posted on course website). As explained on page 5 of the manual, you will need the supplied [RAM\_init.mif](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Spr08/PROJECT/RAM_init.mif) file. A .vhd file will be created in your working directory. (If .mif file creates problems in your design, save the .mif file as .hex file using Quartus II)
3. Include the memory.vhd file, created in Note(3) above, in your datapath.
4. Create a **CPU component** by instantiating and connecting your control unit and datapath components. CPU I/O ports should be limited to a clock, reset, inr as the input ports and outvalue as the output port.
5. For the **final simulation of the test program**, to minimize the size of the listing, display only **one line per clock transition** (i.e., trigger only on clock signal transitions). Show a sufficient set of control signals to demonstrate correct operation of each instruction (control unit state, address bus, data bus, ALU output, register file outputs, register file input, memory control signals, etc.) On the simulation listing, annotate by writing the corresponding assembly language instruction next to each execute cycle and highlighting the “significant” result register or bus value.

**CPU Design Project – Part 5 – Hardware Implementation and a Working Processor Demo, Report Due April 18, 2011**

1. Follow the [Altera Quartus II and DE2 Manual](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Fall10/HW/HW3/Altera%20Quartus%20II%20and%20DE2%20manual.pdf) (posted on course website) for designing and implementing your circuit on the FPGA.
2. Reset can be connected to any of the 4 Keys on DE2 Board. These Keys are always at logic ‘1’. And pressing them will change the logic to ‘0’. So make the changes in your design as needed.
3. Clock can be connected to any of the two free-running clock frequencies available, 27MHz and 50MHZ. To connect to any of these clock inputs, the pin numbers are mentioned in the [Pin Assignment MSExcel](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Spr08/PROJECT/DE2_pin_assignments.csv) Sheet. You can also debug your design by connecting the clock to any of the manual keys on the DE2 board instead of using free-running clock.
4. The “inr” input that selects the register number can be connected to any 4 switches on the board. And the “outvalue” that displays the contents of the register selected, can be connected to the LEDs or LCD on the board. For using 7 segment displays on the board you will require a [HEX to 7 segment](http://www.eng.auburn.edu/~vagrawal/COURSE/E6200_Fall10/PROJECT/hexto7seg.vhd) conversion code provided on the course website.
5. Run the program given to you and verify the results with your simulation in part 4.
6. You will have to show the implemented design on your DE2 Board. You will be conduction a demo as follows:

(a) Briefly describe what is implemented, what program you will run and what result is expected.

(b) Run the program pointing to the functions of the buttons you press. Let the viewer examine the result.

(c) Offer to make a change to some parameter to a viewer selected value and rerun the demo.

(d) Total duration of demo: FIVE MINUTES.

1. Part 5 report must be a one-page reply to three questions:

(a) What did you learn from this project?

(b) What would you do differently next time?

(c) What is your advice to someone who is going to work on a similar project?