**Spring 2013**

**ELEC 5200/6200 CPU Design Project**

**Assigned February 11, 2013**

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 6 parts as defined below along with their due dates. It is **highly advised** that you read problem definitions of all six 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, part 4, and part 5. Always **annotate** your simulation results. Maintain a single folder for submitting the project parts. When submitting a later part, all the previous parts need to be in the folder.

To complete the project, you will need to use the equipment provided in the lab in Broun 320. If you do not already have access to the lab, kindly contact Joe Haggerty, haggejk@auburn.edu.

Contacts:

Vishwani Agrawal, Professor, vagrawal@eng.auburn.edu, Broun 423, 334-844-1853

Nikhil Garrepalli, GTA, nzg0010@tigermail.auburn.edu, Broun 359, 334-844-1865

**CPU Design Project – Part 1 – ISA, Report Due February 22, 2013**

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 7 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 1, 2013**

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. For single-cycle and pipeline datapath, create a **truth table** listing all control signals and the values of each control signal required for executing each instruction in your ISA. For multicycle datapath, create a **state diagram**, and also a **state table** listing all control signals and the values of each control signal corresponding to the different states (like fetch, decode, ALU operation, etc) in your diagram. (Note that in state like “ALU operation”, the control signals may vary depending on different opcodes.)
5. 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, multi-cycle design or pipeline datapath.
* Decisions related to “shared” and/or “dedicated” components.
* Selection of edge-triggered vs. latching registers.
* Other decisions that were considered.

**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).** PC is nothing but a register.
5. **Instruction register** (**IR**) (if required).
6. Assorted **multiplexers** for data paths and register address inputs.

**CPU Design Project- Part 3-Datapath components and control unit implementation, Report Due March 22, 2013**

Develop and verify the VHDL/Verilog model for each unique component within your datapath. Also, design and test a VHDL/Verilog “behavioral” model of the control unit to realize the behavior described in your control signal table in part 2. Thoroughly simulate the control unit and each component individually. **Guaranteeing the correct functioning of each component is very important** since the control unit and all components will be connected in the next part to form a top-level component. Any failing component will induce the malfunctioning of the top-level module.

Note that the register file needs **an additional 4-bit input named “inr”** used as a multiplexor to decide which register to select. It also needs **an additional 16-bit output named** “outvalue” to display the value of the selected register. These two ports will be part of the primary inputs and primary outputs of the final CPU. These two ports are added to help us correctly implement our CPU all the way through this project. In the simulation phase, they are used to help us decide whether we get the correct results saved in the correct register. In the experiment phase, these two ports are used to display the content of target register on the FPGA board to demonstrate the correct execution of your program. **In real CPU, we don’t have these two ports.** They are only used to help us implement this project.

For single-cycle and pipeline datapath, the lecture slides show separate instruction memory and data memory. If your datapath uses a separate data and instruction memory, you can actually write your own instruction memory in Modelsim, and leave the data memory to be generated from Alera’s Megafunction library in part 5. In this way, it will be easier for you to simulate a small program in part 4. Of course, you can also leave both memories to be generated in part 5. In this way you will have to manually supply each instruction to the datapath in part 4.

If your design is multi-cycle datapath, you can leave your memory to be generated in part 5.

Notes:

1. 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.
2. For every sequential component within your datapath, you will need an input “reset” signal.
3. The VHDL/Verilog model is to be a register-transfer-level (RTL) design (not gate level).
4. Design and **test** VHDL/Verilog models of **each unique component**.
5. Submit all VHDL/Verilog code. Submit all simulation results and clearly annotate each simulation results.

 **CPU Design Project- Part 4-Datapath Verification, Report Due March 29, 2013**

Combine all the components created in part 3 together. Create a top-level component which will include both datapath and the control unit connected to each other (also include the instruction memory in your datapath if you have created it in part 3). This top-level component will be very similar to the final CPU you are going to create in part 5 except the memory.

The top-level component must be capable of working with at least a single memory outside the CPU. So it must have at least three 16-bit external “ports” to connect the component to the memory: a Read data bus (16-bit), A Write data bus (16-bit) and an address bus (10-bit). For these who have separate data and instruction memory in the datapath, if you have created an instruction memory in part 3, you will just need one more data memory. Otherwise, you will need two memories (both instruction and data memory) for Logic Simulation in ModelSim. The memory you need will be added in Part 5, which will be the RAM block in the Altera Megafunctions Library.

Verify this top-level component as described in the block diagram and register transfers in Part 2. Write a testbench to verify the correct execution of each instruction in your ISA (except the instruction which need memory access, like “lw” and “sw”). S**how 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.) After that, write a little program (containing all kinds of instructions except “lw” and “sw”), hand compile the chosen test program into binary code and then use this code to verify if your datapath give you the expected results.

Think about it, can you verify execution of “lw” and “sw” instruction in this part? If yes, how?

Notes:

1. In the simulation, if you have included the instruction memory in your datapath, you can write instructions into the instruction memory and do simulation. **Otherwise**, you need to supply instruction(s) to the datapath manually in the testbench.
2. You will have to submit the VHDL/verilog code of the top level component.
3. The top-level design should contain only component instantiations, matching your block diagram (changes may be made to the diagram as necessary).
4. You need to submit simulation results showing correct execution of each instruction and also the little program. Clearly label the important information on the simulation results.
5. You need to submit both the assembly code and the binary code of the program you used.

**CPU Design Project – Part 5 – Final CPU, Report Due April 12, 2013**

Now, write a test program that contains all kinds of instructions in your ISA. Manually compile the test program into binary code. You need to submit your assembly language code and binary code of the test program.

How to create memory from Altera’s Megafunction Library? 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 “.mif file” is a memory initialization file. **What’s written in the “.mif” file will be written into corresponding address of the generated memory**. **So you can put any 16-bit word in any cell of the memory by modifying the RAM\_init.mif file.** For example, you can modify the given RAM\_init.mif file according to the binary code of your test program. You can also put some data in some cells of the memory. At last, a .vhd/.v 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).

**For students who use separate instruction and data memories:** If you have written the instruction memory in modelsim, you just need to generate a data memory in this part; otherwise, you need to generate two memories, one for storing the test program, the other for storing data.

Include the memory.vhd/memory.v file created above in your datapath. Create the final **CPU component** by instantiating and connecting the memory to the top-level component created in part 4. CPU I/O ports should be limited to a clock, reset, and inr as the input ports and outvalue as the output port. Submit the code of the memory and the final CPU.

Now, verify the correct execution of “lw” and “sw” instruction, submit and annotate the simulation results. Also write a testbench to simulate the test program. For the **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 6 – Hardware Implementation and a Working Processor Demo, Report Due April 19, 2013**

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 test program and verify the results with your simulation in part 5.
6. You will have to show the implemented design on your DE2 Board. You will be conducting 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?