**ECE 2504: Introduction to Computer Engineering (Fall 2017)**
**Design Project 3: Implementing the Simple Computer on the DE0 Nano Board**

**Read the entire specification before you begin working on this project!**

## Honor Code Requirements

Each student must complete this project and the associated report individually. Do not discuss any aspect of your solution or approach with anyone except for your instructor or a CEL GTA. Consider all information that you derive from your design process to be proprietary. Among other things, this includes the manner in which you implement your operations, and the number of gates that you use. Copying or using any other person's design is a violation of the Virginia Tech Honor Code, and will be prosecuted as such. You may discuss general features of Quartus and the DE0 Nano board. Direct all other questions to your GTA or to your instructor.

## Objectives

- Design, simulate, and implement new instructions for the Simple Computer from a specification.
- Write a project report describing the design process and its results.

## Preparation

You must have access to a computer that can run Quartus. You must have a DE0 Nano board.

Read this project specification in its entirety. Consult the appropriate sections of Chapter 3, Chapter 6 and Chapter 8 of the textbook. You may also consult the ECE2504 Lab Manual, the DE0 Nano board user's manual, which is on the DVD included with your board, particularly Chapter 3 and 6, and the Quartus instructions from previous assignments.

## Project Description

The Simple Computer from Chapter 8 of the textbook is a single-cycle, load-store central processing unit (CPU). The single-cycle Simple Computer illustrates many of the major principles and design constraints involved in implementing a CPU. For this project, you will trace the execution of a small program on the Simple Computer in a Verilog simulation. You will then modify the design of the Simple Computer to include several new instructions and demonstrate the correct operation of those instructions in simulation and on the DE0 Nano board. Table 1 shows the new instructions that you must implement in the Simple Computer, along with one optional instruction for extra credit. Table 1 shows similar information for the new instructions as Table 8-8 in the text does for the original instructions, where zf stands for zero-filled.

**Table 1: New Simple Computer instructions to be implemented**

| Instruction | Mnemonic | Format | Description |
|---|---|---|---|
| Add and Increment | ADDINC | RD, RA, RB | R[DR] ← R[SA] + R[SB] + 1 |
| Subtract Immediate | SUBI | RD, RA, OP | R[DR] ← R[SA] – zf OP |
| Negate Immediate | NEGI | RD, OP | R[DR] ← – zf OP |
| NAND | NAND | RD, RA, RB | $\mathrm{R[DR]} \leftarrow \overline{\mathrm{R[SA]} \wedge \mathrm{R[SB]}}$ |
| **Optional:** Jump and link | JAL | RA, RB | PC ← R[SA], R[SB] ← PC + 1 |

The files for the project are in the Quartus archived project posted with this description. The project includes a working version of the Simple Computer with a small program. The system takes as input the four DIP switches on the DE0 Nano board (SW[3:0]) and the two pushbuttons (KEY[1:0]). KEY0 is the reset signal for the project, while KEY1 advances the

Simple Computer by one clock cycle. For this project, reset is synchronous with the clock, which means that to reset the Simple Computer, you must press KEY0 and hold it down while pressing and releasing KEY1.

The DIP switches control a mux in the top-level project module; the mux is used to select which value is displayed on the LEDs. The Verilog model provided for the module already has an 8 bit wide, 16-to-1 mux instantiated in it with the DIP switches connected to the select lines. The mux provides the outputs shown in Table 2. SW[3:1] select which register, and SW[0] selects between the most significant byte and least significant byte of the register.

**Table 2: DIP switch select lines and value displayed on LEDs**

| **SW[3:1]** | **Value displayed on LEDs**<br>**SW[0] selects between MSB (1) and LSB (0)** |
|---|---|
| 000 | R0 |
| 001 | R1 |
| 010 | R2 |
| 011 | R3 |
| 100 | R4 |
| 101 | R5 |
| 110 | Program Counter (PC) |
| 111 | Instruction Register (IR) |

As part of your design process, you will have to assign specific 7-bit opcode values for each new instruction. Do not arbitrarily assign a specific opcode to a particular instruction: You must choose the opcodes based upon the control signals available in the datapath. Your choice of opcode will affect the design of the instruction decoding logic in the Simple Computer. <u>Your project report must include a table showing the opcode values that you chose for each instruction</u>.

**Requirements and Constraints**

1. You are permitted to modify these files: `cpu.v` (the processor CPU), `pc_controller.v` (the Program Counter controller), `function_unit.v` (the processor's Function Unit), `instr_decoder.v` (the processor's Instruction Decoder), `instruction.txt` (a representation of the processor's Instruction Memory), and `data.txt` (a representation of the processor's Data Memory). You must not modify any other file or schematic in the project. Any additional modules that you might need to create to implement your design should be included in the `cpu.v` file.
2. You are not permitted to modify the port declarations of any of the above-mentioned modules.
3. The original instruction set must still be functional. You may not modify or replace any of the instructions or opcodes in the original instruction set. Your new instructions and opcodes must be in addition to the original instruction set.
4. You must implement your design using structural and dataflow Verilog constructs (gate primitives, assign statements with operators). You are not permitted to create any schematics or to use behavioral Verilog constructs (e.g., case statements, for loops, if-else statements).
5. Each of your instructions must take only one clock cycle to complete.
6. Each of your instructions must use one of the three instruction formats for the original instructions.
7. Your validation program must use the last four digits of your student ID number as the input data to the program as described in the procedure section and validation sheet.

**Procedure**

The Quartus archive project file posted for this project includes the Verilog files necessary to build the system. Do not modify any file that you are not allowed to change. In particular, if you modify the top-level Verilog file or change the pin assignments, there is a chance you could damage your DE0 Nano board.

The Quartus project provided with this project implements the Simple Computer described in Chapter 8 of the textbook. The implementation allows you to step through the execution of the program (contained in `instruction.txt`) one instruction at a time by pressing and releasing KEY1. KEY0 is a synchronous reset, as described previously. The DIP switches will allow you to view the program counter (PC), the Instruction Register (IR), and the first six registers (R0-R5) in the register file.

The procedure for this project has three major steps:

**Step 1**. To help you understand the operation of the Simple Computer, your first step is to simulate the Simple Computer as it executes a small program. (Note: You should use the Simple Computer as provided, without modification, for this step.) The small program loads the value $2504_{16}$ from the data memory into register R3, goes through a for loop, then performs arithmetic on 2504 and stores the various results in several registers. Table 3 shows the hexadecimal values of the first six instructions in memory. You should determine the mnemonic and the operands for each instruction in the first fifteen locations in the provided `instruction.txt` memory (addresses 0 through E). Complete Table 3 and include it in your report. Address 0 has been done for you as an example.

**Table 3: Table to be used for disassembling the first fifteen instructions in memory**

| Instruction Memory Address | Machine code Value | Instruction (values in decimal) |
|---|---|---|
| 0 | 1400 | XOR R0, R0, R0 |
| 1 | 2100 | |
| 2 | 8443 | |
| 3 | 0480 | |
| 4 | 00E0 | |
| 5 | 0290 | |
| 6 | 0ADA | |
| ... to address E | | |

As with the previous project, for the simulation of the whole project, you must create input waveforms for the clock, the pushbuttons, and the switches. The pushbuttons should change on the negative edge of clock and should hold their value for several clock cycles after any change to reflect the operation of the actual hardware. Include a waveform of the simulation in the report. Your waveform should include at least the following signals: clock, PC, instruction register, instruction decoder output, registers 0-5, and the register file input data bus. To select these signals in the simulation waveform, in QSim, when using the Node Finder to list the available pins, change the filter from “Pins:all” to “Design entry (all names)”. This will allow you to select internal nodes of the design rather than just the external connections (the pins). Given the size of this design, there will be a large number of nodes listed. You can use “...” button next to the “List button” to limit the nodes listed to a particular module of the design hierarchy. For example, Figure 1 shows the module hierarchy being limited to showing only nodes in the cpu module, while Figure 2 shows the node finder list after the cpu module has been selected from the module hierarchy.

Your report should compare the results of the simulation to what you expected given the instructions in Table 3.

Once you are satisfied that the simulation executes the program as you would expect it to, you should compile the design and download it to the DE0 Nano board. Please refer to the previous lab assignment and sections 6.8 and 6.9 of the DE0 Nano user's manual for instructions on downloading a design to the board.

To test the CPU on the board, you should begin by resetting the CPU by pressing and holding the KEY0 pushbutton, pressing and releasing the KEY1 pushbutton, and then releasing KEY0. You can now operate the CPU by using the DIP switches to control what is displayed on the LEDs and using the KEY1 pushbutton to advance the CPU by one clock cycle. Step through the program one instruction at a time and verify that the hardware behaves the same as the simulations by using the DIP switches and LEDs to examine the values in the various registers.

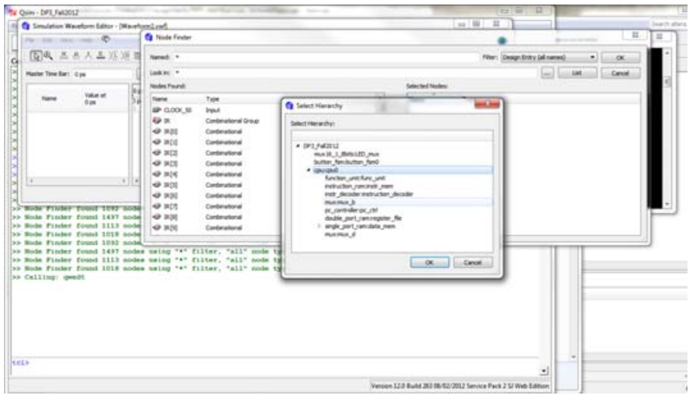
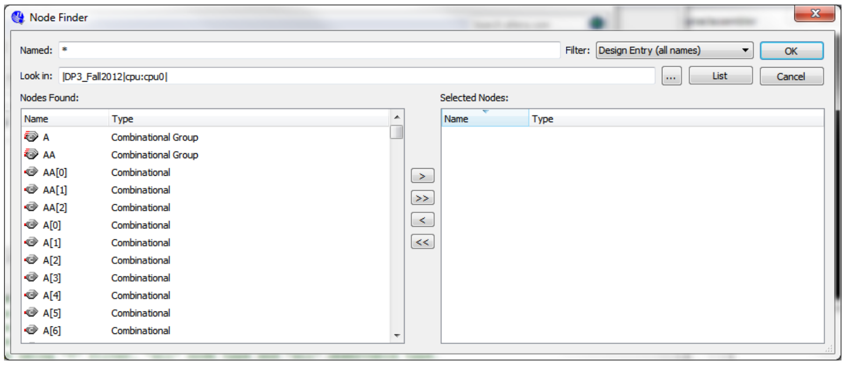


**Figure 1. Selecting part of the hierarchy to limit the list of signals in the node finder**



Node Finder

Named: *

Filter: Design Entry (all names)

Look in: |DP3_Fall2012|cpu:cpu0|

Nodes Found:

| Name | Type |
|---|---|
| A | Combinational Group |
| AA | Combinational Group |
| AA[0] | Combinational |
| AA[1] | Combinational |
| AA[2] | Combinational |
| A[0] | Combinational |
| A[1] | Combinational |
| A[2] | Combinational |
| A[3] | Combinational |
| A[4] | Combinational |
| A[5] | Combinational |
| A[6] | Combinational |

Selected Nodes:

| Name | Type |
|---|---|

**Figure 2. The node finder after limiting the hierarchy to the cpu module.**

**Step 2**. Once you have traced the execution of instructions on the original Simple Computer and understand how the instructions are decoded to control the datapath, modify the Simple Computer to include the new instructions shown in Table 1. It may be possible to implement some of the required instructions by modifying only the instruction decoding logic in the CPU datapath. The optional instruction might require additional changes to the CPU datapath as well as the instruction decoder. See requirement 1 above for which files you are permitted to modify.

For each of the instructions, you will have to choose an opcode to represent it. You must not use one of the opcodes that are in the original instruction set. A careful choice of the opcodes will reduce the changes that you must make to the design. Your report should include a table of the opcodes for each instruction that you implement. Your report should also include a brief explanation of the changes made to the design to implement your instructions, along with any problems that you faced that you were unable to solve.

For each of the instructions that you implement, you should include annotated simulation results in your report showing the correct behavior of the instruction. Your report should also include a table (similar to Table 4) that contains the information necessary to understand the simulation results for each instruction. This table is similar to Table 8-11 in the text and should explain the operation of each of your instructions. You are permitted to include additional information in the table if that is necessary to understand your implementation, particularly if you implement the optional instruction.

**Table 4: Values from the simulation results for each new instruction implemented.**

| Assembly instruction | Opcode (hex) | Other fields of instruction (e.g., DR) | FS | MB | MD | RW | MW | PL | JB | BC |
|---|---|---|---|---|---|---|---|---|---|---|
| ADDINC | | | | | | | | | | |
| SUBI | | | | | | | | | | |
| NEGI | | | | | | | | | | |
| NAND | | | | | | | | | | |
| JAL | | | | | | | | | | |

**Step 3**. After your model simulates satisfactorily, you must compile it and then program the DE0 Nano board with it. You should test your instructions on the board in the same way that you tested the original CPU in step 1.

Once you are confident that your design is working on the DE0 Nano board, you should change the `instruction.txt` file so that it contains the program shown in Table 5, which will be used to validate your project in the CEL. If you implemented the optional JAL instruction, it should be included in address 8 as shown in Table 5; if you did not implement the optional instruction, replace the JAL instruction with "MOVA R0, R0" from address 9. The `data.txt` file should be modified so that the last four digits of your student ID are stored as BCD in address 0 (where "2504" is in the provided file). The program in Table 5, if properly implemented, will then perform operations on your student ID number using the new instructions and store the results in several registers.

Note: As in previous projects (e.g. `rom.txt` in DP2), both `instruction.txt` and `data.txt` are formatted in hex, i.e. 2504 in the provided file indicates the bits 0010010100000100. For example, to store the four BCD codes 0001 0010 0011 0100, "1234" should be entered into `data.txt` in the correct location. To store the machine instruction 0001010010010010, "1492" should be entered into `instruction.txt` in the correct location.

**Table 5: Program to be used for validation (all values are in decimal).**

| Instruction Memory Address | Instruction |
|---|---|
| 0 | XOR R0, R0, R0 |
| 1 | LD R1, R0 |
| 2 | NOT R4,R1 |
| 3 | SUBI R2, R1, 5 |
| 4 | NAND R3, R1, R2 |
| 5 | ADDINC R4, R4, R2 |
| 6 | SUB R5, R1, R4 |
| 7 | NEGI R5, 3 |
| 8 | JAL R2, R0 |
| 9 | MOVA R0, R0 |

## Submission Requirements

### *Validation*

This project does not require CEL validation. Instead, you will provide the source files that will allow the GTA to compile and test your design on the DE0 Nano Board. Create an archive of your work by choosing ***Project > Archive Project*** after you complete the implementation. When you create the archive, it should appear in the same folder that was created when you opened the original archive. Upload the archive (*.qar) to Canvas. *Make certain that you upload the completed archive that you created, and not the one that was provided to you*.

### *Report*

Prepare and submit a written lab report that presents a detailed discussion of the project. It should include the design approach you followed, the final design you implement, the design decisions that you made and the alternatives you considered, your simulation results, your observations, and your conclusions. The validation sheet should be included as the last two pages of your report. You must complete Page 1 of the validation sheet. Subdivide your report into logical sections and label them as appropriate. Be sure to include all the required elements. Refer to the Canvas Assignment rubric to see how this project will be graded.

## Submission Requirements:

Submit the following files on Canvas. **Do not put them in a zip file!**

1. Your Archived Project **DP3_Fall2017_*YourPID*.QAR**
2. Your project report as a pdf or doc. Remember the ***<u>last two pages</u>*** should be the validation sheet. Remember to complete page 1 of the validation sheet!

## Grading

The design project will be graded on a 100 point basis, as shown in the Assignment Rubric.