**ECE 2504: Introduction to Computer Engineering (Spring 2015)**

**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 chips 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 13.1. You must have a DE0 Nano board.

Read this project specification in its entirety. Consult the appropriate sections of Chapter 4, Chapter 7 and Chapter 9 of the textbook. You may also consult the DE0 Nano board user’s manual, which is on the DVD included with your board, particularly Chapter 3 and 6, and the Altera instructions from previous assignments.

**Project Description**

The Simple Computer from Chapter 9 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 four 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 9-8 in the text does for the original instructions where zf stands for zero-filled. Finally, bit-reversal of 10100000 is 00000101.

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Mnemonic** | **Format** | **Description** |
| Addition and Increment | ADDINC | RD, RA, RB | R[DR] ← R[SA] + R[SB] + 1 |
| And Immediate | ANDI | RD, RA, OP | R[DR] ←R[SA] & zf OP |
| Subtract Immediate | SUBI | RD, RA, OP | R[DR] ← R[SA] – zf OP |
| Reverse bits | REV | RD, RA | R[DR] ← Bit-Reversal(R[SA]) |
| **Optional:**  Jump and link | JAL | RA, RB | PC ← R[SA], R[SB] ← PC+1 |

Table 1: New Simple Computer instructions to be implemented

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.

|  |  |
| --- | --- |
| **SW[3:1]** | **Value displayed on LEDs**  **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) |

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

As part of your design process, you will have to assign specific 7-bit opcode values for each new instruction. Do not simply 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. Your project report must include a table showing the opcode values that you chose for each instruction.

**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 module.
3. 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).
4. Each of your instructions must take only one clock cycle to complete.
5. Each of your instructions must use one of the three instruction formats for the original instructions.
6. 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 files except for cpu.v, instruction.txt and data.txt. In particular, if you change the top level Verilog file or modify 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 9 of the textbook. The implementation allows you to step through the execution of the program on 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. The small program loads the value 250416 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.

|  |  |  |
| --- | --- | --- |
| **Instruction Memory Address** | **Machine code Value** | **Instruction (values in decimal)** |
| 0 | 1400 | XOR R0, R0, R0 |
| 1 | 20C0 | Load, R3, R0, R0 |
| 2 | 8444 | ADI R1, R0, R4 |
| 3 | 0480 | ADD R2, R0, R0 |
| 4 | 0290 | INC R2, R2, R0 |
| 5 | 0C48 | DEC R1, R1, R0 |
| 6 | C00A | BRZ R0 R1, R2 |
| 7 | C1C5 | BRZ IR, R0 R5 |
| 8 | 0158 | MOVA R5, R3, R0 |
| 9 | 0BAA | SUB PC, R5, R2 |
| 10 | 0368 | INC, R5, R5, R0 |
| 11 | 0C48 | DEC R1, R1, R0 |
| 12 | 1A41 | SHR R1, R0, R1 |
| 13 | C20A | BRN R0, R1, R2 |
| 14 | E000 | JUMP R0, R0, R0 |

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

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 ModelSim, 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.

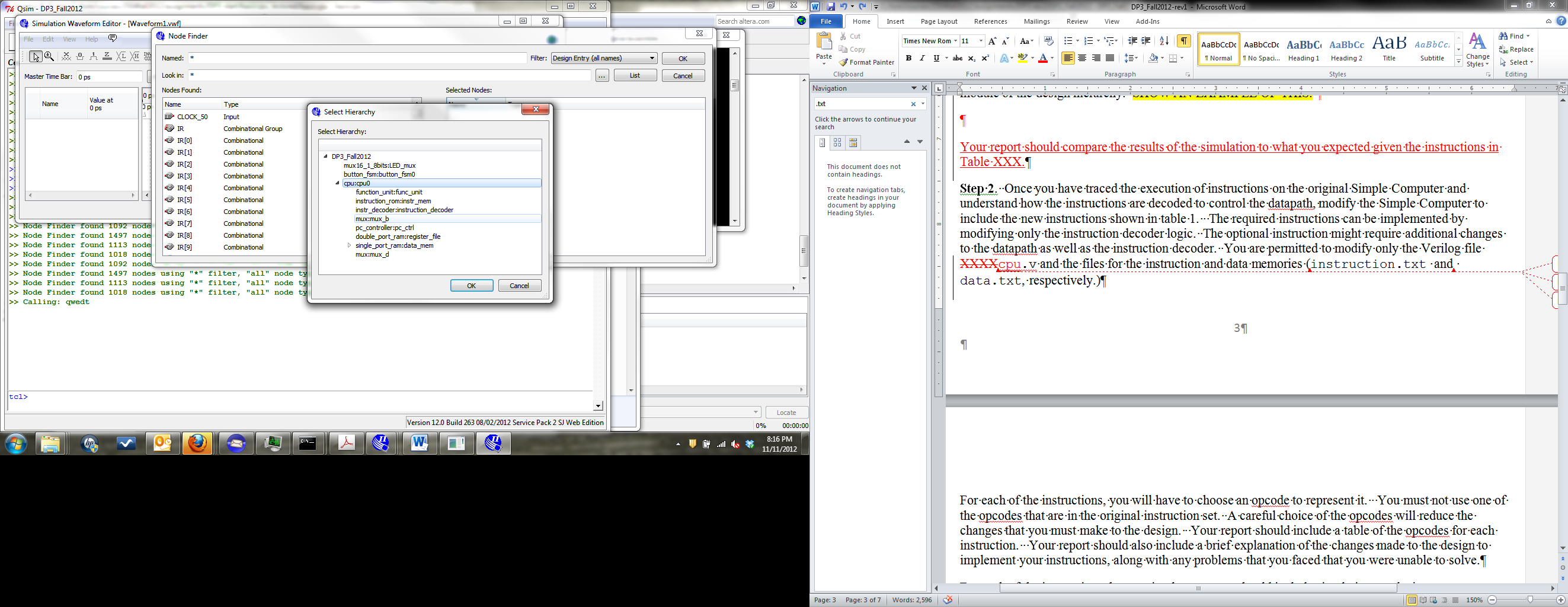


Figure 1. Selecting part of the hierarchy to limit the list of signals in the node finder

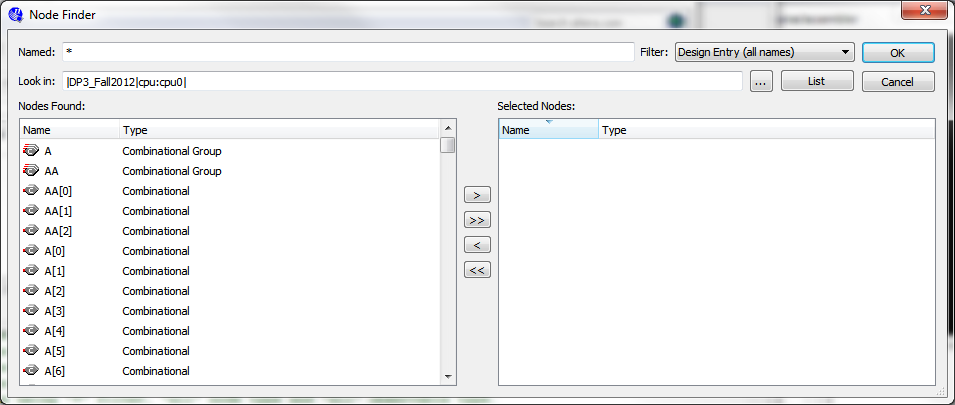


Figure 2. The node finder after limiting the hierarchy to the cpu module.

**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. The required instructions can be implemented 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. You are permitted to modify only the Verilog file cpu.v and the files for the instruction and data memories (instruction.txt and data.txt, respectively.)

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 9-11 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.

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

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

**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 (all values are in decimal, but you must use hexadecimal in the instruction.txt file), which will be used to validate your project in the CEL. If you implemented the optional JAL instruction, it should be included in address 7 as shown in Table 5; if you did not implement the optional instruction, replace the JAL instruction with “MOVA R0, R0” in address 8. 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.

|  |  |
| --- | --- |
| **Instruction Memory Address** | **Instruction** |
| 0 | XOR R0, R0, R0 |
| 1 | LD R1, R0 |
| 2 | INC R2,R1 |
| 3 | REV R3, R2 |
| 4 | ADDINC R3, R1, R2 |
| 5 | ANDI R4, R1, 5 |
| 6 | SUBI R5, R1, 7 |
| 7 | JAL R2, R0 |
| 8 | MOVA R0, R0 |

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

After you have built your circuit and are satisfied with its operation for these operations, take your computer and DE0 Nano board to the CEL and have the GTA validate the circuit by completing the included validation form. The waiting lines in the CEL can be very long as the due date approaches, so validate as early as you can. You should have your Quartus project open on your computer before validation for the GTA so that he can examine your design if necessary.

**Project Report**

After you have validated your logic circuit, 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. Subdivide your report into logical sections and label them as appropriate.

**Your lab report should be submitted on Scholar. Your validation sheet should be submitted as a hard copy.**

Refer to Section 6 of the course Lab Manual for details on preparing your report. Prepare your report on your word processor. Do not include hand-written items. Proofread your report to ensure that it is free of spelling and grammar errors.

Use the cover sheet included with this specification as the first page of your report. Do not use any other cover page. Please submit your report with completed cover sheet on Scholar**. You must also submit your instruction.txt, data.txt, and cpu.v files that you used for validation on Scholar.**

**Grading**

The design project will be graded on a 100 point basis, as shown on the cover sheet.

**ECE 2504, Spring 2015**

**Design Project 3: Implementing the Simple Computer on the DE0 Nano board**

**Validation Sheet**

**All sections of this Validation Sheet must be completed in INK. Failure to use ink will result in a validation grade of 0.**

**The Validation Sheet for Project 3 is two pages long. Make sure that you take both pages of this Validation Sheet and your student ID card to the CEL when you go to have your design validated.** Before you begin validation, you should have your Quartus project open on your laptop with your DE0 Nano board connected, and the programmer window open and ready to program the board. Your rom.txt file should be open in the Quartus window for the GTA to see.

**No GTA or student should discuss any aspect of a student’s design with another student. Among other things, this includes the manner in which a student implements specific operations.**

**It is the student’s responsibility to make sure that this validation sheet is completed correctly. If there are any questions about the validation, the student should check with the instructor.**

Student Name (Printed in ink):

GTA Validation Instructions

Do not validate this lab if the student has not printed their name in ink above and on the next page. Before continuing, verify that the name printed in both places is the name on the student’s ID card. For the DIP switch settings, SW[3:1] values from 000 to 101 select registers R0-R5, with SW[0] selecting between the most significant byte (SW[0]=1) and the least significant byte (SW[0]=0) being displayed on the LEDs. The PC register is displayed for SW[3:1] = 110.

1. Program the FPGA on the DE0 Nano board using the Start button on the programmer window.
2. When the programming has successfully completed, reset the design by pressing and holding KEY0, and while keeping KEY0 pressed, pressing and releasing the KEY1 pushbutton.
3. Set the DIP switches to “1100” to show the program counter (PC). Press and release KEY1 two times. The LEDs should read 02 (hex).
4. Set the DIP switches SW[3:1] to “001”, and then use SW[0] to record the 16-bit value for R1 in Table 1 (next page) as four digit hex.
5. Compare the four digits from step 4 to the last four digits of the student’s ID number on their ID card. **If the four digits do not match the last four digits of the student’s ID number on their ID card, STOP THE VALIDATION. DO NOT CONTINUE.**
6. Press and release KEY1. Set the SW[3:1] for R2, and record the 16-bit value for R2 as four digit hex in Table 1.
7. Press and release KEY1. Set the SW[3:1] for R3, and record the 16-bit value for R3 as four digit hex in Table 1.
8. Press and release KEY1. Set the SW[3:1] for R3, and record the 16-bit value for R3 as four digit hex in Table 1.
9. Press and release KEY1. Set the SW[3:1] for R4, and record the 16-bit value for R4 as four digit hex in Table 1.
10. Press and release KEY1. Set the SW[3:1] for R5, and record the 16-bit value for R5 as four digit hex in Table 1.
11. *If the student has implemented the optional instruction:* Press and release KEY1. Set the SW[3:1] for R0 and then the PC, and record the 16-bit values for R0 and the PC register as four digit hex in Table 1.

Student Name (Printed in ink):

Table 1: Checking the operation of the CPU.

All values of the registers should be recorded as four digit hexadecimal numbers, two digits for the most significant byte of the register and two digits for the least significant byte.

DIP switch settings SW[3:1]

* Values from 000 to 101 select Registers R0 – R5.
* The PC register is displayed for SW[3:1] = 110.
* SW[0] selects between the most significant byte (SW[0]=1) and the least significant byte (SW[0]=0) being displayed on the LEDs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Step number** | **Register** | **SW[3:1] setting** | **SW[0]=1** | **SW[0]=0** |
| 4 | R1 | 001 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 6 | R2 | 010 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 7 | R3 | 011 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 8 | R3 | 011 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 9 | R4 | 100 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 10 | R5 | 101 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 11 (optional) | R0 | 000 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |
| 11 (optional) | PC | 110 | \_\_\_ \_\_\_ | \_\_\_ \_\_\_ |

Comments (only required if something is unusual or wrong):

GTA Printed Name and Signature:

Date and Time of Validation: