Design Project 3: Implementing (and Modifying) the Simple Computer on the DE0 Nano Board

---

Jamahl Savage

ECE 2504: Introduction to Computer Engineering

November 18, 2018

Purpose and Project Description

The purpose for this design project was to design, simulate, and implement new instructions for an operational simple computer. The simple computer utilized for this design project is a single-cycle, load-store central processing unit (CPU). The new instructions for the simple computer must be implemented only using structural and dataflow Verilog constructs. With the implementation of the new instructions, it is especially required that the original instructions must continue to be operational as well. The new instructions must be assigned specific 7-bit opcode values, which should also not be arbitrarily assigned opcodes. It is also important to note that only certain files related to the simple computer may be modified. Additionally, each new instruction implemented in the simple computer must also only take one clock cycle to complete. The DE0 Nano board's DIP switches, KEY1, and KEY0 will be used to display certain values on the DE0 Nano's LEDs.

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 4: The DIP switch select lines and the value to be displayed on LEDs

Technical Approach and Implementation

Before implementing the new instructions for the simple computer, I first traced the execution of the default instructions already in the simple computer. To help with this process I deciphered the instructions from the default instruction words given in hex.

Instruction Memory Address	Machine code Value	Instruction (values in decimal)
0	1400 -0001010000000000	XOR R0, R0, R0
1	20C0 -0010000011000000	LD R3, R0
2	8444 -1000010001000100	ADI R1, R0, 4
3	0480 -0000010010000000	ADD R2, R0, R0
4	0290 -0000001010010000	INC R2, R2
5	0C48 -0000110001001000	DEC R1, R1
6	C00A -1100000000001010	BRZ R1, 2
7	C1C5-1100000111000101	BRZ R0, -3
8	0158 -0000000101011000	MOVA R5, R3
9	0BAA -0000101110101010	SUB R6, R5, R2
A	0368 -0000001101101000	INC R5, R5
B	0C48 -0000110001001000	DEC R1, R1
C	1A41 -0001101001000001	SHR R1, R1
D	C20A -1100001000001010	BRN R1, 2
E	E000 -1110000000000000	JMP R0

Table 1: The table used for disassembling the first fifteen instructions in memory

As can be seen in **Table 1**, all the instructions represented specific operations and the registers that would be used to perform those operations. Doing this will help me notice what the operations are and the type of effect they would have on the various registers during a simulation. Next, I created a simulation of the simple computer with its default instructions.

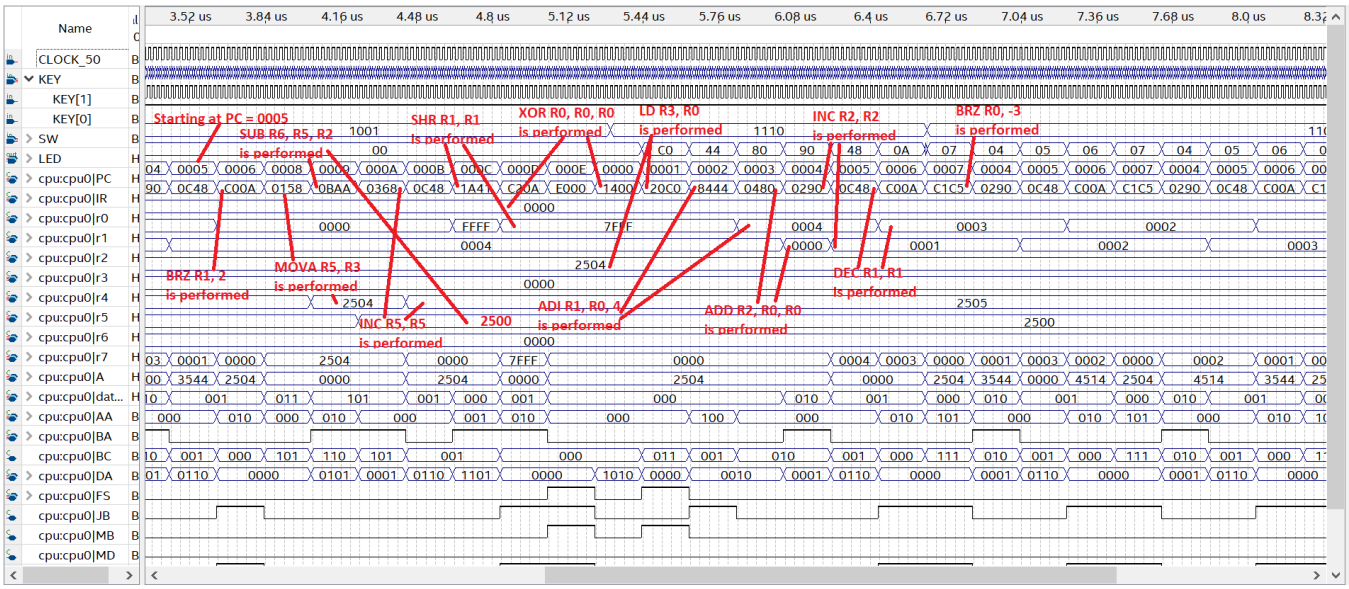

Figure 1: The Simple Computer's simulation with the default instructions

As can be seen in **Figure 1**, the various operations affect each of their respective registers, and in some cases the program counter (PC), depending on the clock cycle, KEY1, and KEY0. The DIP switch values just allow me to be able to see a specific value if I were to program the simple computer onto the DE0 Nano board. As I assumed, when certain operations used the same register that was already used for a previous operation, the new value from the previous operation is stored in the register and used instead of the original value. The BRN, BRZ, and JMP operations manipulated the PC and made it repeat some operations because of its new value.

After analyzing the results of the simulation and the function unit that produces the results, I began to design the specific 7-bit opcodes for the required operations called ADDINC (Add and Increment), ORI (OR Immediate), SUBI (Subtract Immediate), REV (Bit Reverse), and CSR (Circular Shift Right). The first thing I did was figure out which operations I could reuse for any of these new instructions. After reading through the function unit I discovered that I could use the already implemented ADD, OR, and SUB operations to achieve the desired results with ADDINC, ORI, and SUBI respectively. I took advantage of the how the function unit selects the values for operands in the arithmetic section. The operand B determines its value by checking bit 2 and 1 of the Function Select (FS), which is determined based on last four bits of the opcode. In order for operand B to be the buffer of B, FS[2:1] must be 0 and 1. To determine whether the value from the arithmetic unit or logic unit should be displayed, the FS[3] must be 0. And finally to choose between the arithmetic unit's and the shift unit's output, the AND of FS[3] and FS[2] must be 0. Based on this analysis, the opcode so far is 000001. To determine the LSB for the opcode, a picture like **Figure 3** should be consulted.

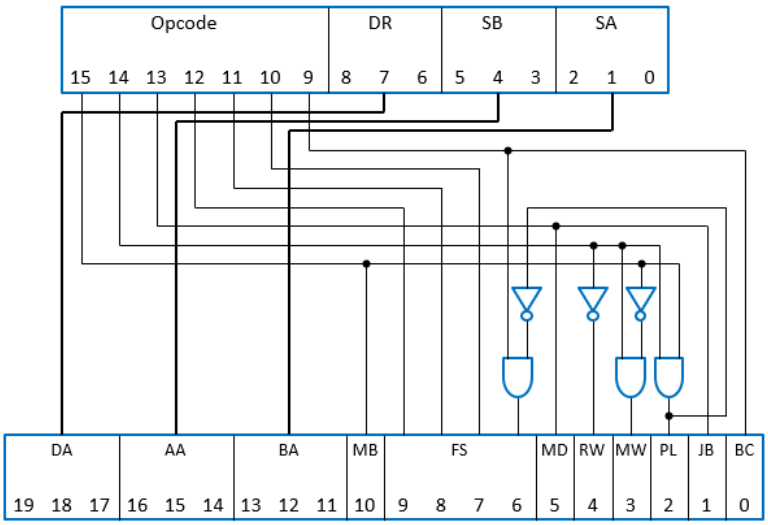

Figure 3: The connections used to derive each component of the instruction word and processor control word

Based on the connections shown in **Figure 3** and the value of PL having to be 0, the complete opcode for ADDINC is 0000011. Following a similar process, I derived the opcode of SUBI to be 1000101, for the operation still utilizes the subtraction function but instead uses a constant to subtract from operand A. The opcode for ORI was also derived because it would still be utilizing the already implemented OR function but instead would be performing an OR between operand A and a constant; therefore, the appropriate opcode was determined to be 1001001. Because I had all the opcodes grouped with their respective units, I had to take a different approach with the REV and CSR operations. I made REV's opcode be 0000111 and CSR's opcode be 0001111 because I believe it was appropriate to make these operations belong to the shift unit and additionally, these opcodes were unused by the simple computer. However, because of this decision I made, I had to alter the simple computer to have a condition for when the opcodes were either 0000111 (REV) or 0001111 (CSR). **Figure 4** demonstrates the changes I had to make to the function unit to implement REV and CSR the way I designed it.

```
// The Shifter uses FS[1:0] to perform (B, sr B, sl B).
assign shift_out = (FS[3:0] == 4'b1111/*The CSR Operation*/) ? { B[0], B[15:1] } :
                   (FS[3:0] == 4'b0111/*The REV Operation*/) ? { A[0], A[1], A[2], A[3], A[4], A[5],
                                                                A[6], A[7], A[8], A[9], A[10], A[11],
                                                                A[12], A[13], A[14], A[15] } :
                   (FS[1:0] == 2'b00) ? {1'b0, B} :
                    (FS[1:0] == 2'b01) ? {1'b0, 1'b0, B[15:1]} :
                    (FS[1:0] == 2'b10) ? {1'b0, B[14:0], 1'b0} : 17'bxxxxxxxxxxxxxxxxx;

// The mux that chooses between the ALU and shifter has a select equal to (FS[3] & FS[2]).

wire[3:0] FSDecide;

assign FSDecide = (FS[3:0] == 4'b0111) ? { ~FS[3], FS[2], FS[1], FS[0] } : FS[3:0];

assign MF = FSDecide[3] & FS[2]; //Used to be assign MF = FS[3] & FS[2];

// The mux on the Function Unit output uses MF to perform (ALU, Shifter).
assign temp_F = (MF == 1'b0) ? temp_G : shift_out;
```

Figure 4: The changes implemented in the Function Unit for the REV and CSR operations

As can be seen in **Figure 4**, I have made it to where REV and CSR will be treated as part of the shift unit if they are selected by using conditional Verilog statements. With these last operations implemented, I have now implemented all the new instructions required by the design project. The complete list for the new instructions respective opcodes and other values can be found in **Table 2** of **Appendix C**. With this part of the design project completed, I proceeded to verify my implementation through simulations.

Results

After creating and implementing the new instructions, I conducted a simulation on the simple computer. The results of the simulation can be seen in the annotated **Figure 2**, which has fixed switch settings and a clock set to 50 MHz for a total of 10 microseconds.

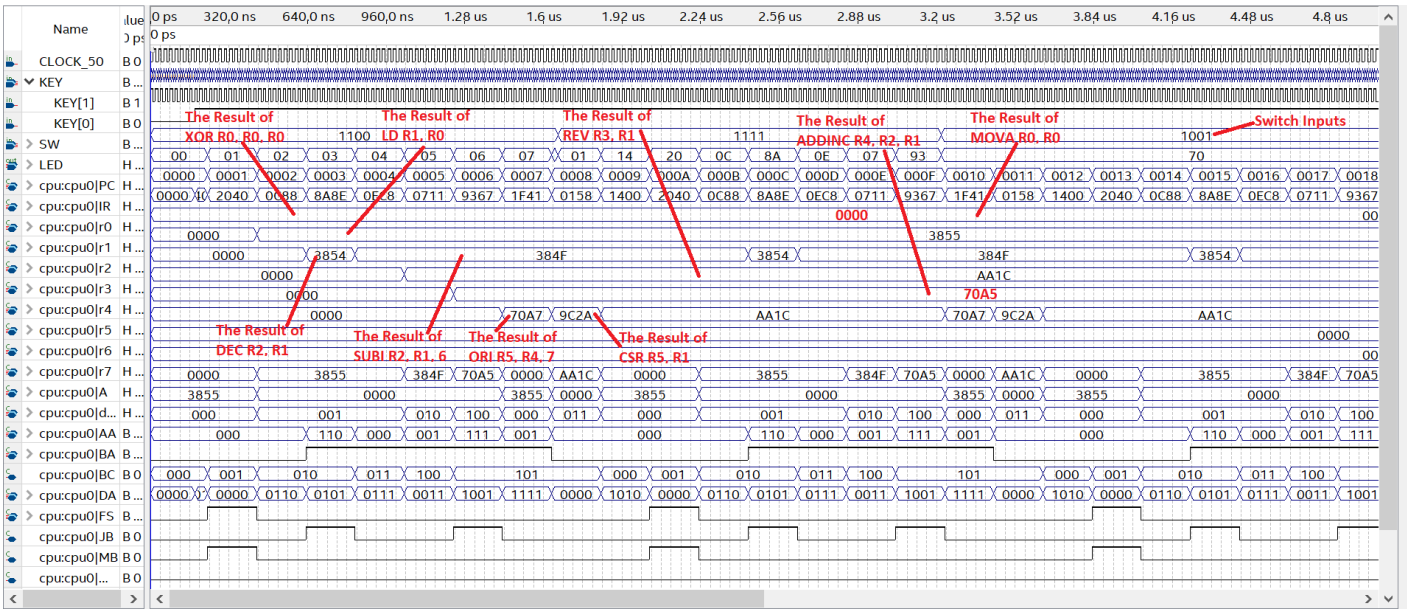

Figure 2: The Simple Computer's simulation with the new instructions

The full details regarding the new instructions' values for the datapath multiplexer selects, register file write, data memory write, etc. can be found in **Appendix C**. The full list of the instruction words and the operations used on the registers can be found in **Appendix D**. When programming the DE0 Nano board, the results did not fail to meet my expectations because of its success with the aforementioned simulation. The LEDs correctly turned on and off for the appropriate switch input and provided the desired output based on the simulation results.

Conclusion

There were multiple ways this design project could have been approached, fortunately I chose a design that required me to make minor changes and add only two new operations. I also believe my design approach to be one of the most efficient options due to the simplicity of the its changes to the simple computer. This design project has provided me the opportunity to practice translating instruction words into processor control words and vice versa. It has also given me the chance to study the inside of a CPU and observe how certain components allow a CPU to perform the decisions it does. By being required to only to use structural and dataflow Verilog, I was also able to analyze how the wires, inputs, and outputs interacted with each other on a much simpler level. After conducting a simulation on the simple computer with the new

instructions implemented and the programmed DE0 Nano board displaying the desired results, I believe it is possible to say that I have successfully implemented the new instructions into the simple computer. Therefore, I believe I have met the design project’s specifications.

Appendix A: Simulating the Initial Simple Computer

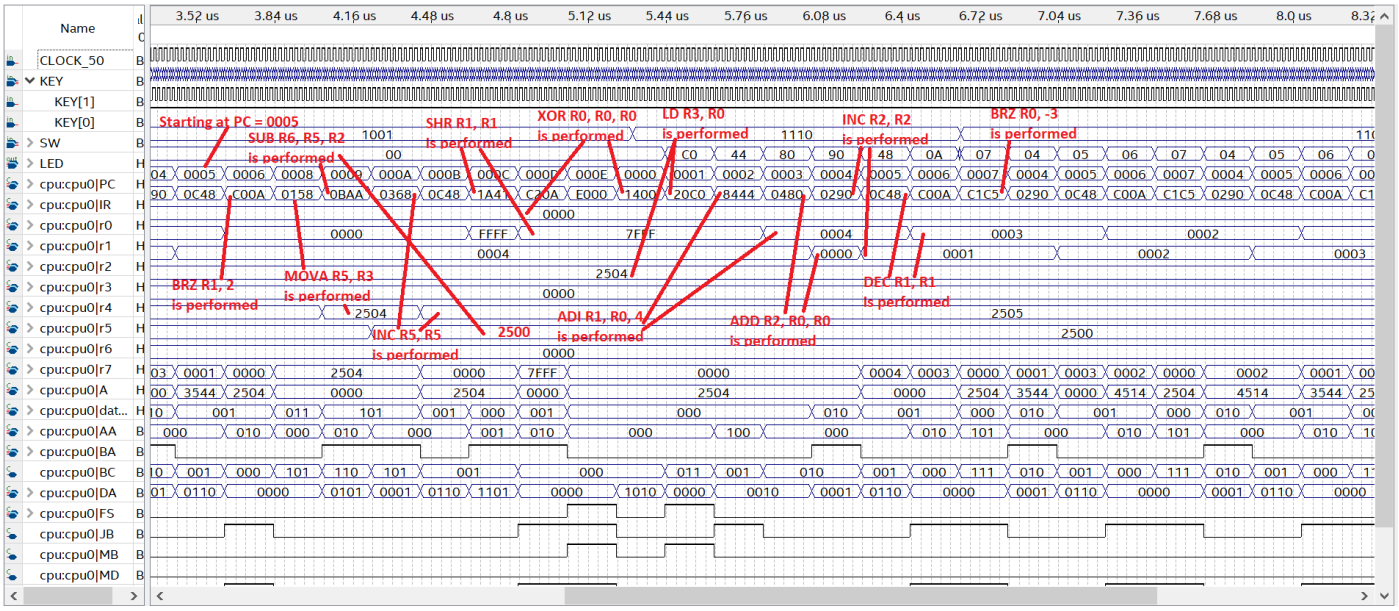

Figure 1: The Simple Computer's simulation with the default instructions

Appendix B: The Simple Computer's Default Table

Instruction Memory Address	Machine code Value	Instruction (values in decimal)
0	1400 -0001010000000000	XOR R0, R0, R0
1	20C0 -0010000011000000	LD R3, R0
2	8444 -1000010001000100	ADI R1, R0, 4
3	0480 -0000010010000000	ADD R2, R0, R0
4	0290 -0000001010010000	INC R2, R2
5	0C48 -0000110001001000	DEC R1, R1
6	C00A -1100000000001010	BRZ R1, 2
7	C1C5-1100000111000101	BRZ R0, -3
8	0158 -0000000101011000	MOVA R5, R3
9	0BAA -0000101110101010	SUB R6, R5, R2
A	0368 -0000001101101000	INC R5, R5
B	0C48 -0000110001001000	DEC R1, R1
C	1A41 -0001101001000001	SHR R1, R1
D	C20A -1100001000001010	BRN R1, 2
E	E000 -1110000000000000	JMP R0

Table 1: The table used for disassembling the first fifteen instructions in memory

Appendix C: Values for the Implemented Instructions

Assembly Instruction	Opcode (hex)	Other fields of instruction (e.g., DR)	FS	MB	MD	RW	MW	PL	JB	BC
ADDINC	03	DR, SA, SB	0011	0	0	1	0	0	0	1
ORI	49	DR, SA, OP	1001	1	0	1	0	0	0	1
SUBI	45	DR, SA, OP	0101	1	0	1	0	0	0	1
REV	07	DR, SA	0111	0	0	1	0	0	0	1
CSR	0F	DR, SB	1111	0	0	1	0	0	0	1

Table 2: Values used for each new instruction implemented

Appendix D: The Simple Computer's New Instruction Table

Instruction Memory Address	Machine code Value	Instruction (values in decimal)
0	1400 - 0001010000000000	XOR R0, R0, R0
1	2040 - 0010000001000000	LD R1, R0
2	0C88 - 0000110010001000	DEC R2, R1
3	8A8E - 1000101010001110	SUBI R2, R1, 6
4	0EC8 - 0000111011001000	REV R3, R1
5	0711 - 0000011100010001	ADDINC R4, R2, R1
6	9367 - 1001001101100111	ORI R5, R4, 7
7	1F41 - 0001111101000001	CSR R5, R1
8	0158 - 0000000000000000	MOVA R0, R0

Table 3: The Newly Implemented Operations Used in the Simple Computer with its Complete Instructions.

Appendix E: Simulation for the Simple Computer with the New Instructions

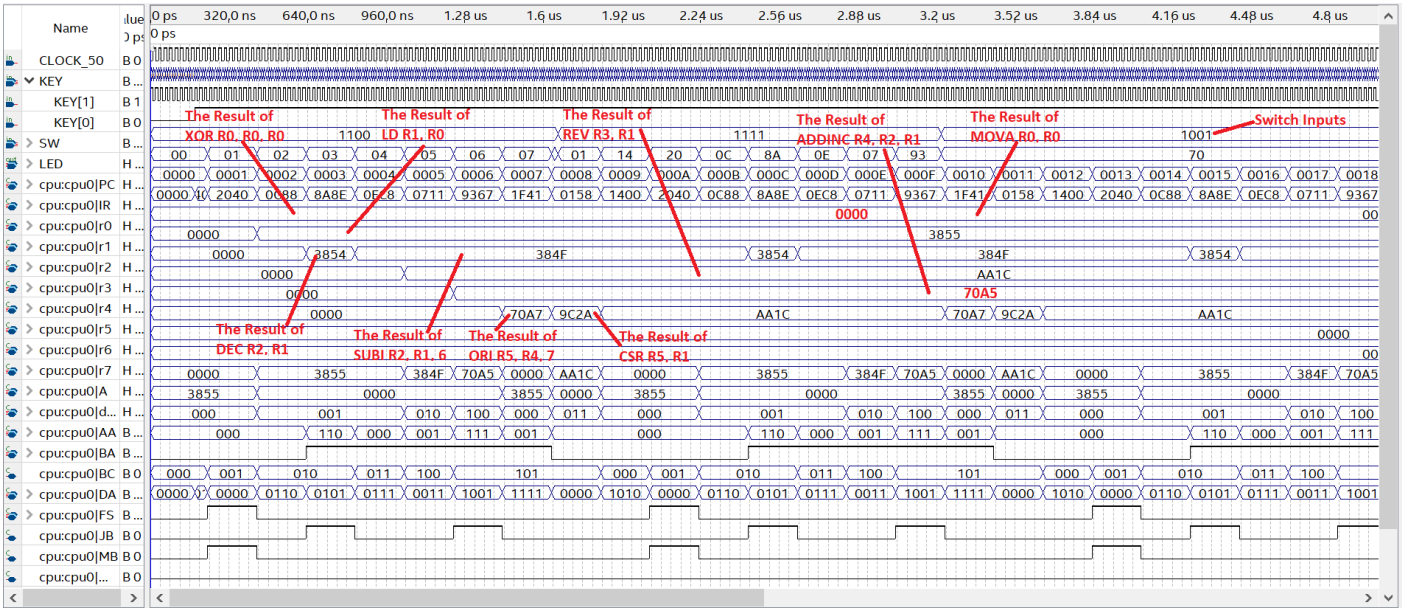

Figure 2: The Simple Computer's simulation with the new instructions

Appendix F: The Possible Switch Values for Displaying Values on the DE0 Nano

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: The DIP switch select lines and the value to be displayed on LEDs

Appendix G: The Relationship Between the Instruction Word and the Processor Control Word

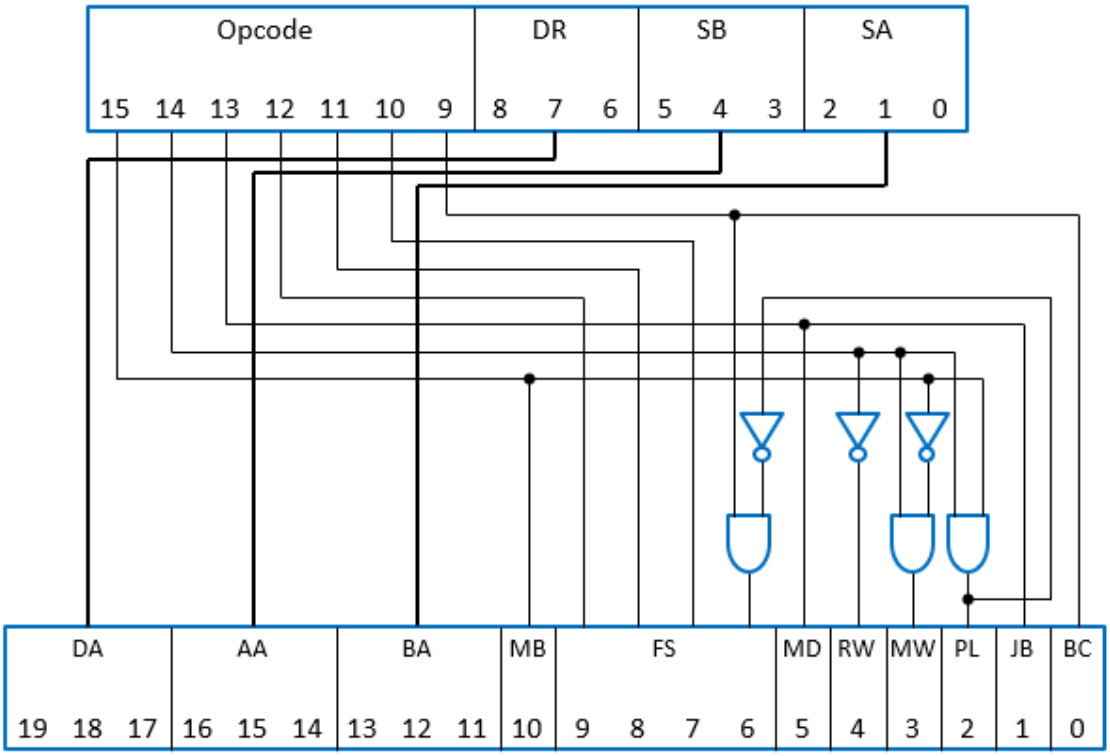

Figure 3: The connections used to derive each component of the instruction word and processor control word

Appendix H: The Implementation of the REV and CSR Operations

```
// The Shifter uses FS[1:0] to perform (B, sr B, sl B).
assign shift_out = (FS[3:0] == 4'b1111/*The CSR Operation*/) ? { B[0], B[15:1] } :
                   (FS[3:0] == 4'b0111/*The REV Operation*/) ? { A[0], A[1], A[2], A[3], A[4], A[5],
                                                                A[6], A[7], A[8], A[9], A[10], A[11],
                                                                  A[12], A[13], A[14], A[15] } :
                   (FS[1:0] == 2'b00) ? {1'b0, B} :
                    (FS[1:0] == 2'b01) ? {1'b0, 1'b0, B[15:1]} :
                    (FS[1:0] == 2'b10) ? {1'b0, B[14:0], 1'b0} : 17'bxxxxxxxxxxxxxxxxx;

// The mux that chooses between the ALU and shifter has a select equal to (FS[3] & FS[2]).

wire[3:0] FSDecide;

assign FSDecide = (FS[3:0] == 4'b0111) ? { ~FS[3], FS[2], FS[1], FS[0] } : FS[3:0];

assign MF = FSDecide[3] & FS[2]; //Used to be assign MF = FS[3] & FS[2];

// The mux on the Function Unit output uses MF to perform (ALU, Shifter).
assign temp_F = (MF == 1'b0) ? temp_G : shift_out;
```

Figure 4: The changes implemented in the Function Unit for the REV and CSR operations

Design Project 3: Implementing the Simple Computer on the DE0 Nano Board

Validation Sheet – Page 1 (Instructions)

The validation sheet is provided as an example of how the GTA will test your implementation.

You do not have to go the CEL to have your implementation validated in person.

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.

Student Name: Jamahl Savage

Last four digits of your student ID: 3855

Optional instruction implemented: Yes **No**

Validation Instructions

Table 1 shows the correspondence between the DIP switch settings on the DE0-Nano board and the register value that the LEDs display. SW[3:1] choose a register. SW[0] chooses between the upper and lower byte.

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 1: DIP switch select lines and value displayed on LEDs

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 0x02.
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. If the four digits do not match the last four digits of the student’s ID number, STOP THE VALIDATION. DO NOT CONTINUE.
6. Press and release KEY1 (PC = 0x03). 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 (PC = 0x04). Set the SW[3:1] for R2, and record the 16-bit value for R2 as four digit hex in Table 1.
8. Press and release KEY1 (PC = 0x05). 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 (PC = 0x06). 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. 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.
12. 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.

Design Project 3: Implementing the Simple Computer on the DE0 Nano Board

Validation Sheet – Page 2 (Results)

Record values in Table 2 as four-digit hexadecimal numbers: two digits for the most significant byte of the register and two digits for the least significant byte.

Step number	PC Value	Register	SW[3:1] setting	SW[0]=1	SW[0]=0
4	0x02	R1	001	___ ___	___ ___
6	0x03	R2	010	___ ___	___ ___
7	0x04	R2	010	___ ___	___ ___
8	0x05	R3	011	___ ___	___ ___
9	0x06	R4	100	___ ___	___ ___
10	0x07	R5	101	___ ___	___ ___
11	0x08	R5	101	___ ___	___ ___
12 (optional)	0x09	R0	000	___ ___	___ ___
12 (optional)	0x09	PC	110	___ ___	___ ___

Table 2: Checking the operation of the CPU.

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