ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21335

LECTURER : Prof. Dr. Deniz Turgay Altılar

GROUP MEMBERS:

150230734 : TAŞKIN ÖKMEN

SPRING 2024

Contents

1 INTRODUCTION

In this first project of BLG222E Computer Organization we have used Verilog Hardware Description language to implement a ,part of CPU, the Arithmetic Logic Unit. The design has files of multiple registers and can perform basic logic and arithmetic operations. By combining all the essential parts of ALU we can simulate a part of basic computer.

2 MATERIALS AND METHODS

2.1 TASK DISTRIBUTION

The whole project including the implementation of the system modules (Register, IR, RF, ARF, ALU, ALUSystem) in verilog, debugging and the report is done by TAŞKIN ÖKMEN 150230734. Therefore, TAŞKIN ÖKMEN is the group representative.

2.2 PARTS OF THE PROJECT

2.3 Part-1

The first part consist an 16-bit register. It can also perform 8 operations by selection of 3-bit control signals (FunSel) and an enable input (E). The selection of Funsel performs increment, decrement, loading an input, clearing the bits and more operations. Register itself is consist of 16 flip-flops synchronized by the Clock signal. Register module is also used for the Register File and Address Register File parts.

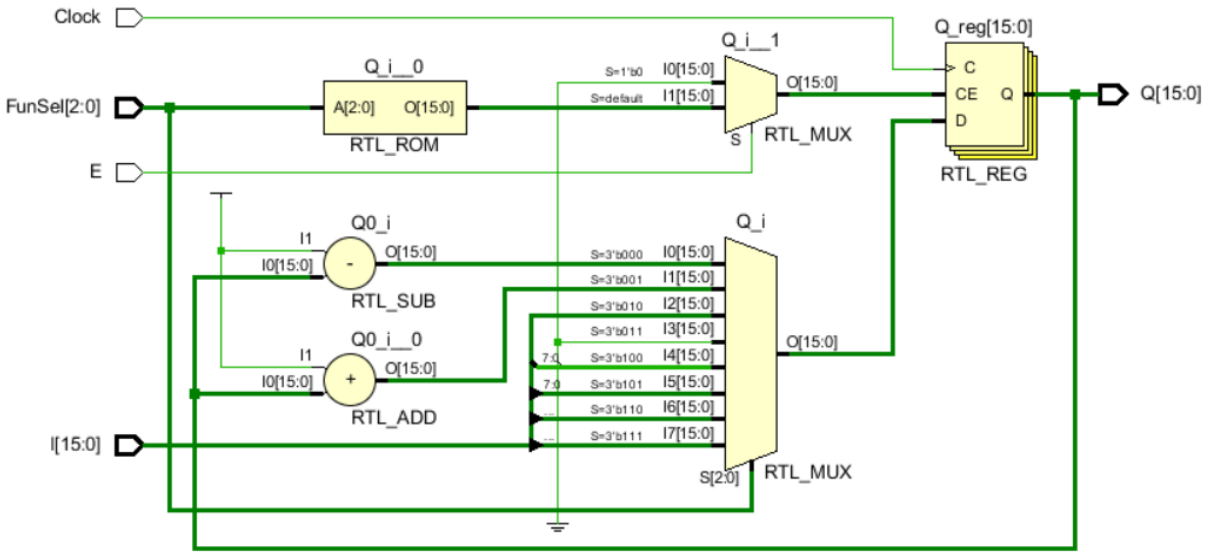

Figure 1: 16-bit Register schematic

2.4 Part-2

2.4.1 Part-2a

The second part consist an 16-bit Instruction Register. It has 4 functionalities by selection of L'H (LOW/HIGH) and E (Enable), we can load the 8-bit input to low or high bits of instruction register. Instruction registers holds the current instruction to be executed in the fetch cycle of CPU. Register itself is consist of 16 flip-flops synchronized by the Clock signal. Instruction Register module is also used for the Address Register File part.

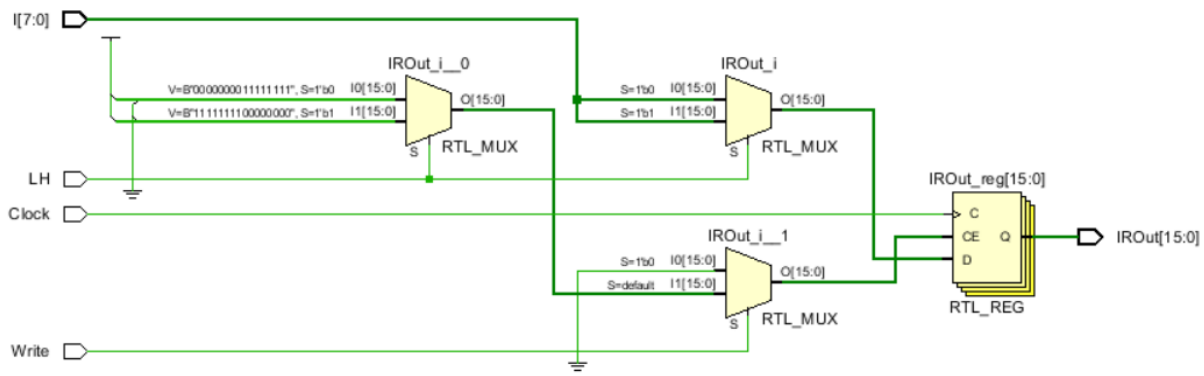

Figure 2: 16-bit Instruction Register schematic

FunSel	R_x^+ (Next State)
000	R_x-1 (Decrement)
001	R_x +1 (Increment)
010	I (Load)
011	0 (Clear)
100	R_x (15-8) ← Clear, R_x (7-0) ← I (7-0) (Write Low)
101	R_x (7-0) ← I (7-0) (Only Write Low)
110	R_x (15-8) ← I (7-0) (Only Write High)
111	R_x (15-8) ← Sign Extend (I (7)) R_x (7-0) ← I (7-0) (Write Low)

Figure 3: Control inputs of the registers by Funsel

2.4.2 Part-2b

The 2b part of the second part consist of an Register File that has 16-bit general purpose registers(R1, R2, R3, R4) and four 16-bit scratch registers(S1, S2, S3, S4). It has multiple functionalities. The 3-bit OutASel and OutBSel are used to reflect the selected register to output lines OutA and OutB. The 3-bit FunSel signal is applied to the enabled registers. The RegSel and ScrSel are used to select the enabled registers normal registers and scratch registers, respectively. Negation of RegSel and ScrSel signals determines the enabled registers each bit of the negated signal controls the enable signal of R1, R2, R3 and R4 (from LSB to MSB). The registers itself is consist of 16 flip-flops synchronized by the Clock signal. Register File module is also used for the Arithmetic Logic Unit System part.

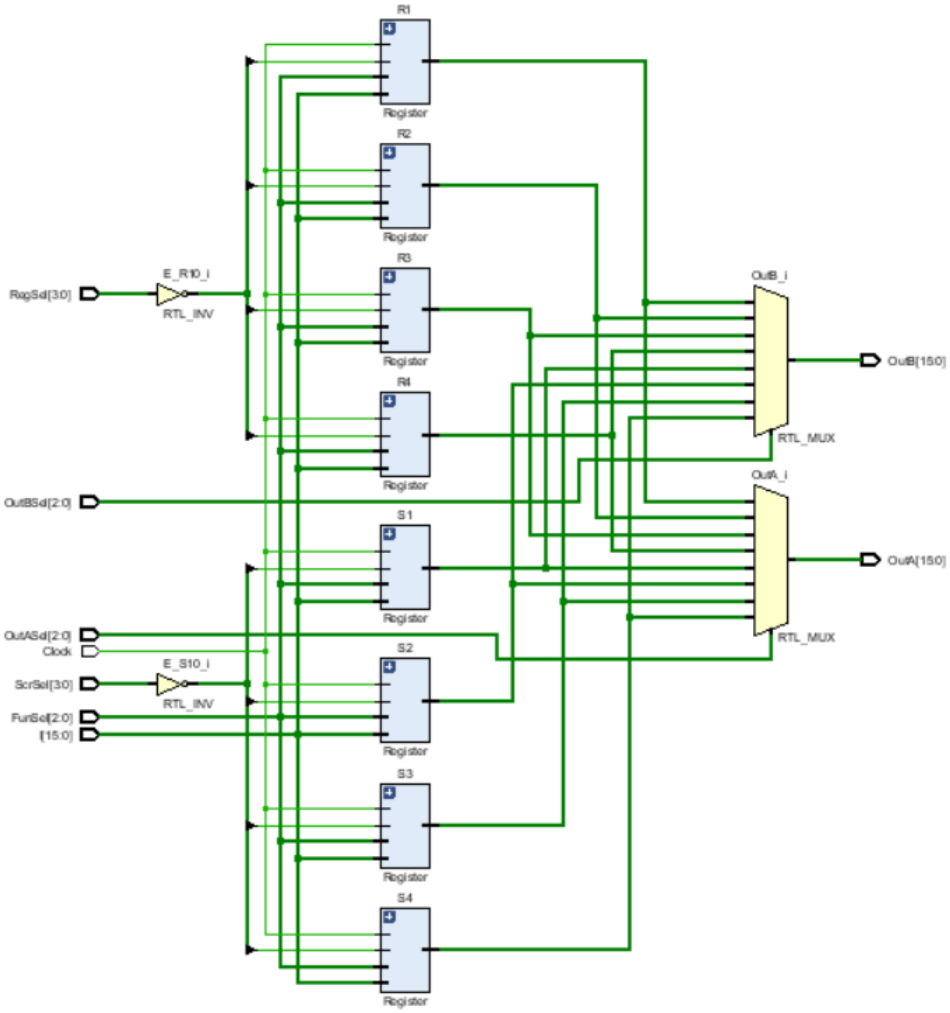

Figure 4: Register File module schematic

2.4.3 Part-2c

The 2c part of the second part is an address register file (ARF) that consists of three 16-bit address registers program counter (PC), address register (AR), and stack pointer (SP). The enabled registers are modified by the selection of Funsel. Negation of RegSel signal determines the enabled registers each bit of the negated signal controls the enable signal of PC, AR, and SP (from LSB to MSB). The 2-bit OutCSel and OutDSel are used to reflect the selected register to output lines OutC and OutC. Program counter holds the next instruction to be executed it is incremented in the fetch phase after address in PC copied into AR. After that, instruction at address register is copied into instruction register. The registers itself is consist of 16 flip-flops synchronized by the Clock signal. Address Register File module is also used for the Arithmetic Logic Unit System part.

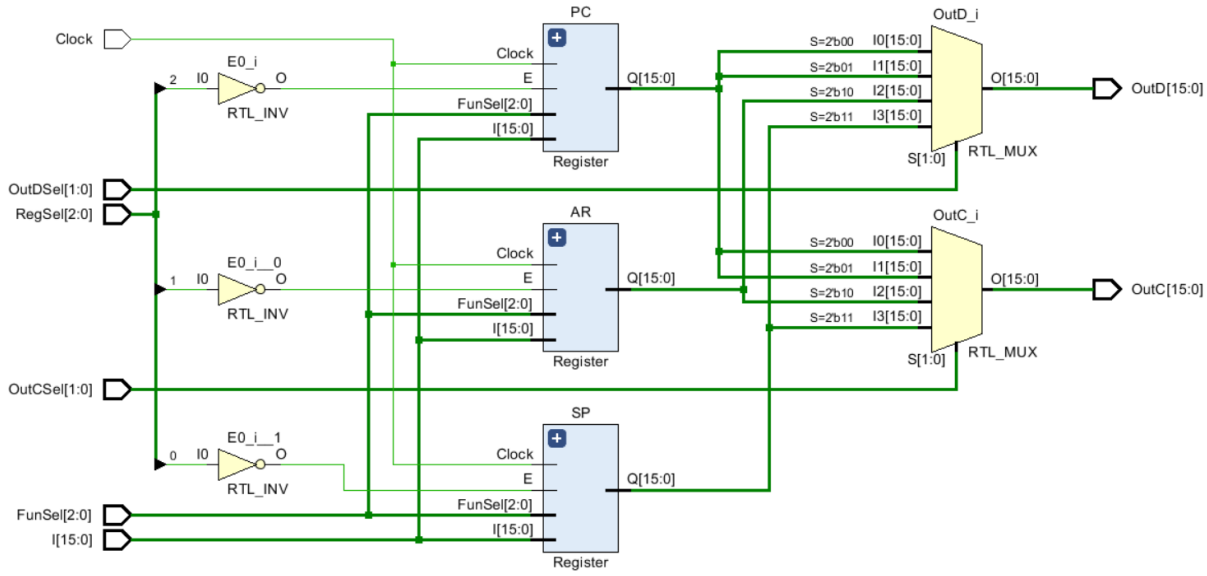

Figure 5: Address Register File module schematic

2.5 Part-3

The third part consist of building the Arithmetic Logic Unit. The arithmetic logic unit performs various logic, arithmetic (addition, subtraction) and shifting operations. The arithmetic operations is performed on 2's complement logic. For example, to perform subtraction second operand is negated and incremented. Similarly, the 5-bit Funsel signal selects the operations to be performed on A and B operands. If MSB of Funsel is 0 it is an 8-bit operation other bits of ALUOut are setted to zero. When MSB of Funsel is 1 it is an 16-bit operation is performed ordinarily. The arithmetic logic unit has four flags (FlagsOut) ZCNO (Zero, Carry, Negative, Overflow) altered by selected functions when WF (Write Flag) is enabled. Z (zero) bit is set if ALUOut is zero (e.g. when NOT B is zero). C (carry) bit is set if ALUOut sets the carry. N (negative) bit is set if the ALU operation generates a negative result. O (overflow) bit is set if an overflow occurs. These flags helps us to understand and validate the state of the current operation. The flags itself is consist of registers synchronized by the Clock signal.

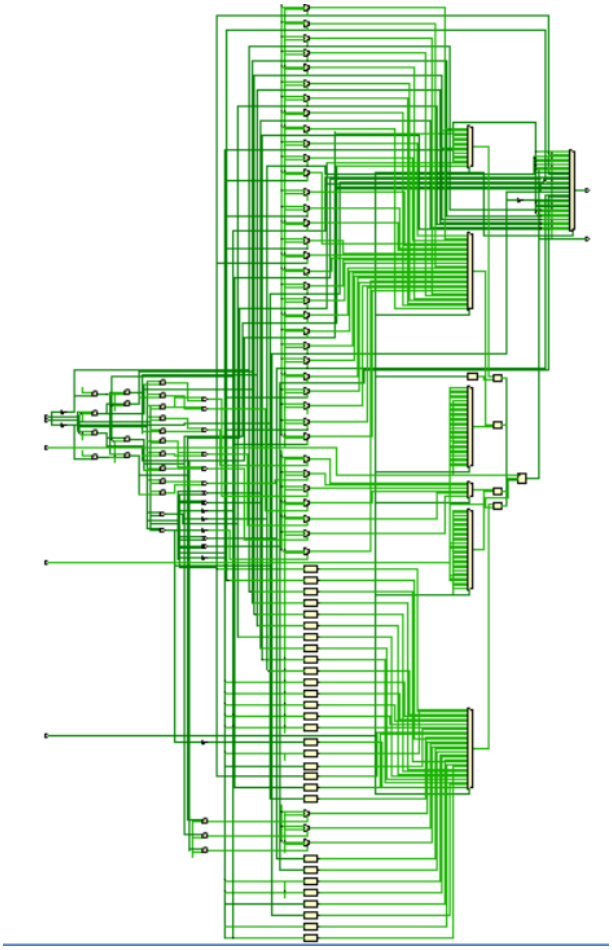

Figure 6: ALU module schematic

FunSel	ALUOut	Z	C	N	O
00000	A (8-bit)	+	-	+	-
00001	B (8-bit)	+	-	+	-
00010	NOT A (8-bit)	+	-	+	-
00011	NOT B (8-bit)	+	-	+	-
00100	A + B (8-bit)	+	+	+	+
00101	A + B + Carry (8-bit)	+	+	+	+
00110	A – B (8-bit)	+	+	+	+
00111	A AND B (8-bit)	+	-	+	-
01000	A OR B (8-bit)	+	-	+	-
01001	A XOR B (8-bit)	+	-	+	-
01010	A NAND B (8-bit)	+	-	+	-
01011	LSL A (8-bit)	+	+	+	-
01100	LSR A (8-bit)	+	+	+	-
01101	ASR A (8-bit)	+	+	-	-
01110	CSL A (8-bit)	+	+	+	-
01111	CSR A (8-bit)	+	+	+	-

FunSel	ALUOut	Z	C	N	O
10000	A (16-bit)	+	-	+	-
10001	B (16-bit)	+	-	+	-
10010	NOT A (16-bit)	+	-	+	-
10011	NOT B (16-bit)	+	-	+	-
10100	A + B (16-bit)	+	+	+	+
10101	A + B + Carry (16-bit)	+	+	+	+
10110	A – B (16-bit)	+	+	+	+
10111	A AND B (16-bit)	+	-	+	-
11000	A OR B (16-bit)	+	-	+	-
11001	A XOR B (16-bit)	+	-	+	-
11010	A NAND B (16-bit)	+	-	+	-
11011	LSL A (16-bit)	+	+	+	-
11100	LSR A (16-bit)	+	+	+	-
11101	ASR A (16-bit)	+	+	-	-
11110	CSL A (16-bit)	+	+	+	-
11111	CSR A (16-bit)	+	+	+	-

Figure 7: Control inputs of the ALU by Funsel

2.6 Part-4

The last part consist of combining all the previous modules to form the whole organization Arithmetic Logic Unit System that utilizes the same single clock. It has an external memory connected with other parts of the arithmetic logic unit. Moreover, OutD output of Address Register File can be used to write or read the data from memory specified by the address register (AR). We have connected all the necessary parts together. It has multiple multiplexers to drive the selected input to other parts of the system. MUX A is 2:4 its output is connected to Register File from part 2b. MUX B is 2:4 its output is connected to Address Register File (ARF) from part 2c. MUX C is 1:2 its output is connected to memory. The third input of MuxA and MuxB are extended with zeros to satisfy the 16-bit bus (instruction registers low or high 8-bits are loaded). Whole system has an instruction register to store current instruction to be executed, a memory system, an arithmetic logic unit that uses register file's outputs as operands and address register file to store program counter, address register and stack pointer.

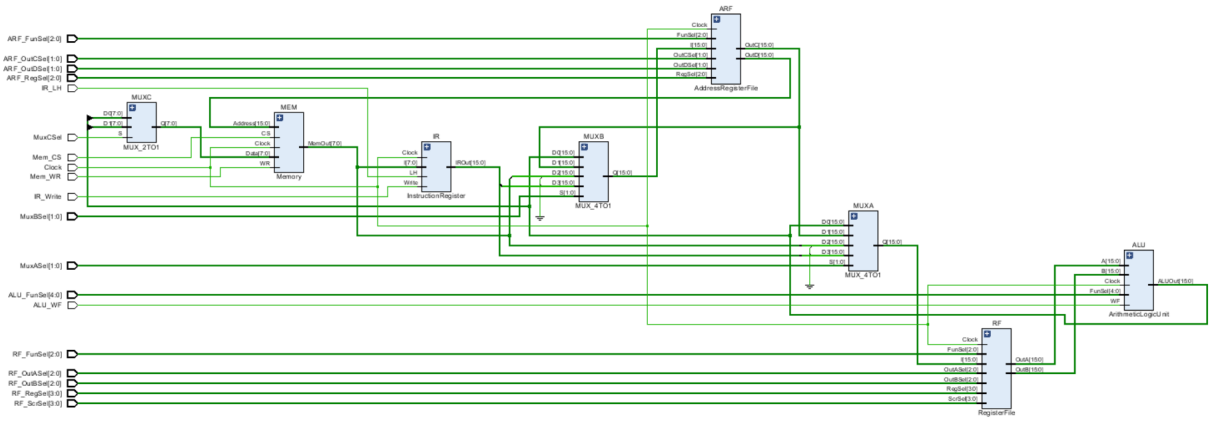

Figure 8: ALU System module schematic

3 RESULTS OF THE SIMULATIONS

We have tested our system to see if it performs the operations properly. The simulation files have helped us to debug and take actions for the deficient parts of our system. The following figures are the results of the given simulation files in Vivado respectively.

3.1 Part-1

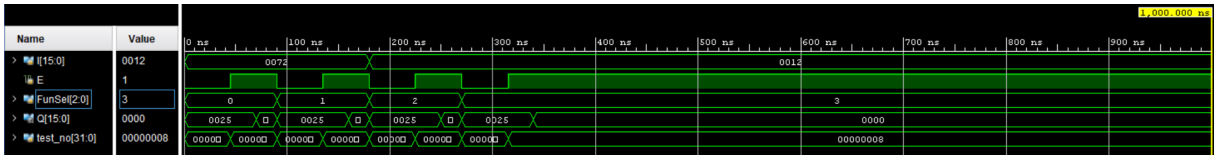

Figure 9: Simulation of part 1

3.2 Part-2a

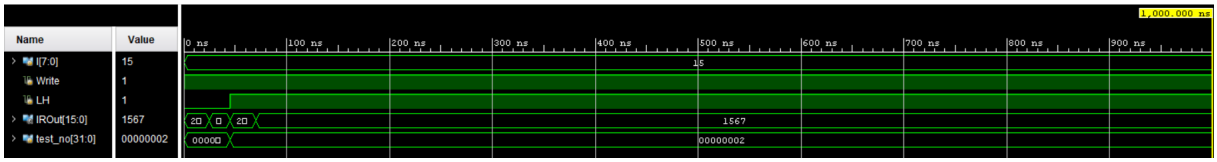

Figure 10: Simulation of part 2a

3.3 Part-2b

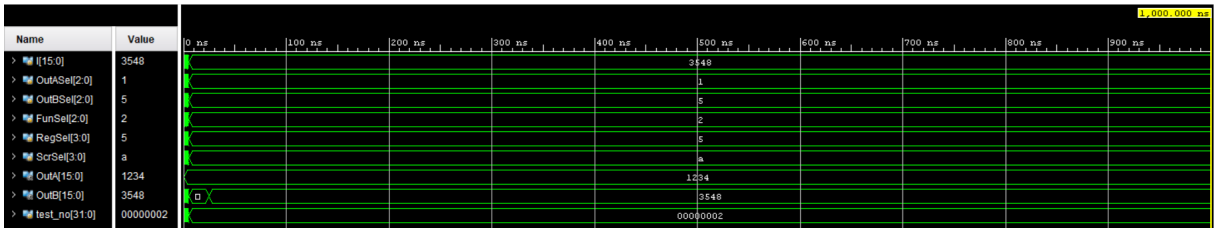

Figure 11: Simulation of part 2b

3.4 Part-2c

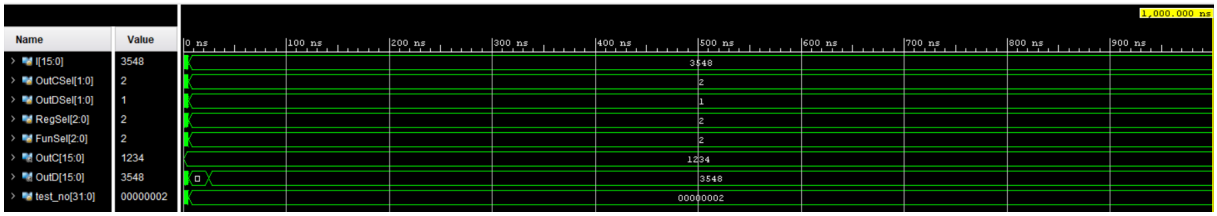

Figure 12: Simulation of part 2c

3.5 Part-3

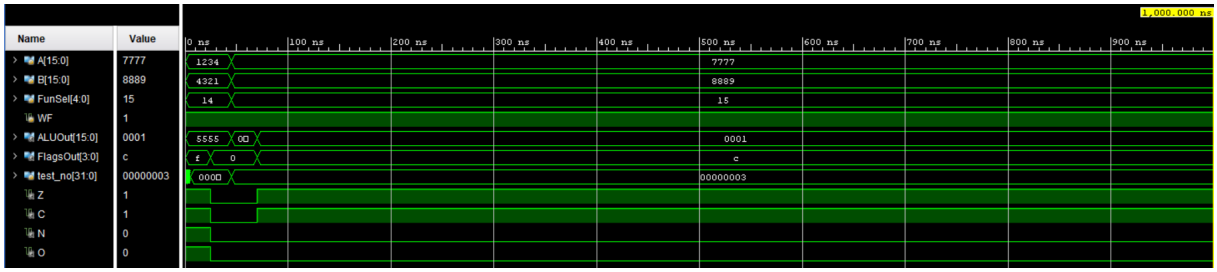

Figure 13: Simulation of part 3

3.6 Part-4

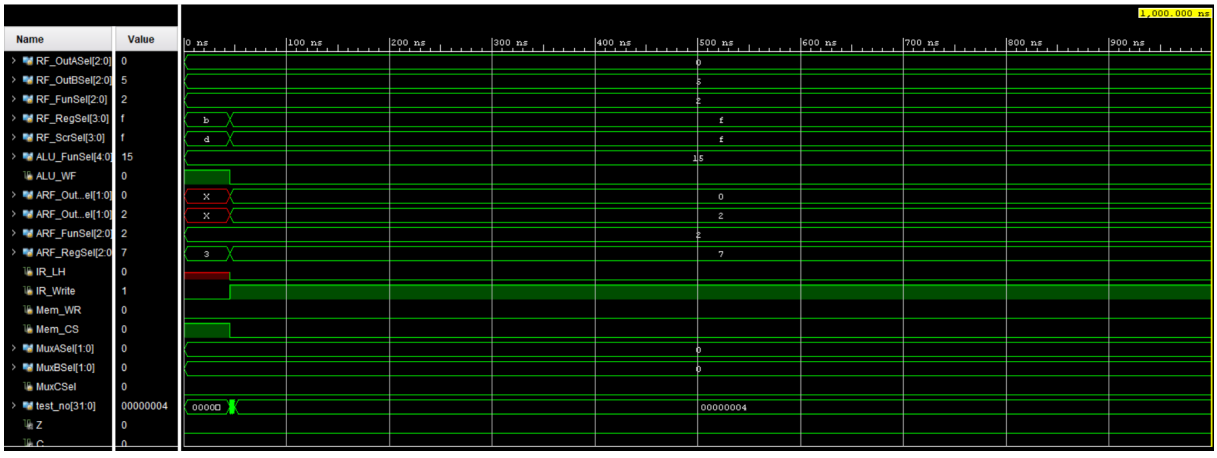

Figure 14: Simulation of part 4

3.7 CUSTOM SIMULATIONS

We also added custom tests to simulation files in order to analyze and validify our implementation of system modules.

3.7.1 Part-1

The custom tests of Register module's inputs and outputs are given in the table below. Register retains its value when the enable signal is at low state. Afterwards, when register is enabled it performs the function selected by Funsel. Tests were conducted subsequently one after another. Output of register set to 0x0000 first then corresponding function operations were held. Each function selected by Funsel are applied to operate at one clock cycle. First input is loaded to Q, incremented by one, decremented by one and write the LSB 8 bits of input to output Q. At last, output Q is cleared to 0x0000.

Enable (E)	Input (I)	Funsel	Q
0	0xFFFF	010 - load	0x0000
1	0x0005	010 - load	0x0005
1	0xXXXX	001 - increment	0x0006
1	0xXXXX	000 - decrement	0x0005
1	0xFFAA	101 - write low	0x00AA
1	0xXXXX	011 - clear	0x0000

Figure 15: Custom test results of Register module

We have tested 6 cases. And the results were consistent with our expected values for output Q.

3.7.2 Part-3

The custom tests of ArithmeticLogicUnit module's inputs and outputs are given in the table below. ALU performs the function selected by 5-bit Funsel signal. If MSB of Funsel is 0 it is an 8-bit operation other bits of ALUOut are set to zero. When MSB of Funsel is 1 it is an 16-bit operation is performed ordinarily. The arithmetic logic unit has four flags (FlagsOut) ZCNO (Zero, Carry, Negative, Overflow) altered by selected functions when WF (Write Flag) is enabled. Tests were conducted subsequently one after another. We can observe that 8-bit operations result in zeros in MSB 8 bits of ALUOut. Results alter only the LSB 8 bits of ALUOut. First flags were set to 0x0000, then corresponding function operations were held. Each function selected by Funsel are applied to operate at one clock cycle.

Write Flag (WF)	Input A	Input B	Operation	ALUOut	Z/C/N/O
0	0xFFF8	0xFFFF	A (8-bit)	0x00F8	0/0/0/0
0	0xFFF8	0xFFFF	B (8-bit)	0x00FF	0/0/0/0
0	0xAAAA	0x5555	NOT A (8-bit)	0x0055	0/0/0/0
0	0xAAAA	0x5555	NOT B (8-bit)	0x00AA	0/0/0/0
1	0x000F	0x000F	A + B (8-bit)	0x001E	0/0/0/0
1	0x0005	0x0005	A + B (8-bit)	0x000A	0/0/0/0
1	0x00A3	0x00B7	A + B + Carry (8-bit)	0x005A	0/0/0/0
1	0xFFF8	0xFFFF	A - B (8-bit)	0x00F9	0/1/1/0
1	0xAAAA	0x5555	A OR B (8-bit)	0x00FF	0/0/1/0
1	0xAAAA	0x5555	A AND B (8-bit)	0x0000	1/0/0/0
1	0xFFFF	0x0000	A AND B (8-bit)	0x0000	1/0/0/0
1	0x5555	0x0000	LSL A (8-bit)	0x00AA	0/0/1/0
1	0xAAAA	0x0000	CSR A (8-bit)	0x0055	0/1/0/0

Figure 16: Custom test results of ALU module (8-bit)

We can observe FlagsOut retain its values when write flag is in its low state. Flag registers change based on the operation type accordingly. For example, the operation A - B (8-bit) for the operands A and B results in 0x00F9 = -7 (in decimal). Result is negative for 8-bits and generates a carry bit 1. Therefore, we can see the result of flag registers being 0 for Z (zero), 1 for C (Carry), 1 for N (Negative) and 0 for O (Overflow).

Similarly, 16-bit operations were held. We have tested 13 cases. And the results were consistent with our expected values for ALUOut.

Write Flag (WF)	Input A	Input B	Operation	ALUOut	Z/C/N/O
0	0xFFF8	0xFFFF	A (16-bit)	0xFFF8	0/0/0/0
0	0xFFF8	0xFFFF	B (16-bit)	0xFFFF	0/0/0/0
1	0xAAAA	0x5555	NOT A (16-bit)	0x5555	0/0/0/0
1	0xAAAA	0x5555	NOT B (16-bit)	0xAAAA	0/0/0/0
1	0x000F	0x000F	A + B (16-bit)	0x001E	0/0/0/0
1	0x0005	0x0005	A + B (16-bit)	0x000A	0/0/0/0
1	0x00A3	0x00B7	A + B + Carry (16-bit)	0x015A	0/0/0/0
1	0xFFF8	0xFFFF	A - B (16-bit)	0xFFF9	0/1/1/0
1	0xAAAA	0x5555	A OR B (16-bit)	0xFFFF	0/0/1/0
1	0xAAAA	0x5555	A AND B (16-bit)	0x0000	1/0/0/0
1	0xFFFF	0x0000	A AND B (16-bit)	0x0000	1/0/0/0
1	0x5555	0x0000	LSL A (16-bit)	0xAAAA	0/0/1/0
1	0xAAAA	0x0000	CSR A (16-bit)	0x5555	0/1/0/0

Figure 17: Custom test results of ALU module (16-bit)

4 DISCUSSION

The project overall required dedication to implement the all the parts. It required the understanding of correlation and relevance between successive parts. First we have implemented all the modules with the help of project file. After that, we have troubleshoot the arising errors with the help of simulation files. Overall, we can confidently say we have an basic understanding of the arithmetic logic unit system of basic computer.

5 CONCLUSION

The most difficult part of the project was debugging the system parts. The simulations files provided an insight, however it was exhaustive and repetitive to observe and interpret the input and output signals at Xilinx Vivado's simulation. It was difficult to grasp and understand the whole system at the tests. Moreover, we had times when all the previous modules were passing the simulation tests including address register file while the arithmetic logic unit system was not. After long observations, we have found that the error was at the enable inputs of the address register file's registers PC, AR and SP. The enable sequence of Regsel by accident is reversed. The MSB of negated Regsel signal were driving the SP it should be PC instead. In breif, we can say, we have learned the general system of arithmetic logic unit.

REFERENCES

John Doe. Computer organization. An example journal, 22(4):10–16, February 2020.