ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21334, 21336

LECTURER : Mustafa Ersel Kamaşak, Gökhan İnce

GROUP MEMBERS:

150210076 : Mehmetcan KUL (group representative)

150220053 : Mustafa Emin ŞİMŞEK

SPRING 2024

Contents

1 INTRODUCTION

In this project, we first learned Verilog and wrote our first register module. Then we created the register file, address register and instruction register modules using this module. We wrote our ALU unit. We set up our multiplexers and finished our project. We understood the basic computer concept.

2 MATERIALS AND METHODS

2.1 Task Distribution

The register in part 1, the address register module(AR), the Multiplexer designs and new tests are the sole responsibility of Mehmetcan. The Register File(RF) and the Instruction Register(IR) modules designs are the sole responsibility of Mustafa Emin. The AlU, AlU system design and Report of the project the jointly responsibility of Mehmetcan and Mustafa Emin.

2.2 Designing a 16-bit register

We built a 16 bit register controlled by 3 bit control signals (FunSel) and an enable input (E). Also, the register is clock driven. We chose an output of type reg because we want to store the state value. Inputs and outputs can be seen in figure 1. Register also has 8 different operations. The operations can be seen in figure 2 and figure 3.

```
input [2:0] FunSel,
input [15:0] I,
input Clock,
input E,
output reg [15:0] Q
```

Figure 1: Register inputs and output

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

Figure 2: Register Function Table [1]

```
3'b000: Q <= Q - 1;
3'b001: Q <= Q + 1;
3'b010: Q <= I;
3'b011: Q <= 16'd0;
3'b100: Q[15:0] <= {{8{1'b0}},I[7:0]};
3'b101: Q[7:0] <= I[7:0];
3'b110: Q[15:8] <= I[15:8];
3'b111: Q[15:0] <= {{8{I[7]}},I[7:0]};
```

Figure 3: Register Function's Code

2.3 Part 2

2.3.1 Designing Instruction Register (IR)

We built a 16 bit register but input of this register is only 8 bit. Therefore we use L'H signal to load either the lower (bits 7-0) or higher (bits 15-8) half. We also have write input to control writing.Inputs and outputs can bee seen figure 4. The use of the L'H and write inputs can be seen in figure 5.

```
input [7:0] I,
input Write,
input LH,
input Clock,
output reg [15:0] IROut);
```

Figure 4: IR's inputs and output

L'H	Write	IR+
ϕ	0	IR (retain value)
0	1	IR (7-0) ← I (Load LSB)
1	1	IR (15-8) ← I (Load MSB)

Figure 5: L'H Signal Table [2]

2.3.2 Designing Register File (RF)

We built a register file with 4 general purpose register and 4 scratch register. We use the first register module for general purpose registers and scratch registers. We have the OutASel and OutBSel inputs to select which register is to be connected to the outputs. OutASel and OutBSel controls can be seen figure 6. We also have FunSel, RegSel and ScrSel inputs. We use the 3-bit FunSel to select which function to use. FunSel control can be seen figure 7. Also We use RegSel and ScrSel to select the registers to which the function is to be applied. RegSel and ScrSel controls can be seen figure 8 and 9. All inputs and outputs can be seen figure 10. We connect our registers to wires after that we connect the wires to outputs. This can bee seen figure 11, 12 and 13.

OutASel	OutA
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

OutBSel	OutB
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

Figure 6: OutASel and OutBSel controls [3]

E	FunSel	Q+
0	ϕ	Q (Retain value)
1	000	Q-1 (Decrement)
1	001	Q+1 (Increment)
1	010	I (Load)
1	011	0 (Clear)
1	100	Q (15-8) ← Clear, Q (7-0) ← I (7-0) (Write Low)
1	101	Q (7-0) ← I (7-0) (Only Write Low)
1	110	Q (15-8) ← I (7-0) (Only Write High)
1	111	Q (15-8) ← Sign Extend (I (7)) Q (7-0) ← I (7-0) (Write Low)

Figure 7: FunSel Control [1]

Table 3: RegSel Control Input

RegSel	Enable General Purpose Registers	RegSel	Enable General Purpose Registers
0000	All general purpose registers are enabled. (Function selected by FunSel will be applied to R1, R2, R3 and R4.)	1000	R2, R3, and R4 are enabled. (Function selected by FunSel will be applied to R2, R3, and R4.)
0001	R1, R2 and R3 are enabled. (Function selected by FunSel will be applied to R1, R2, and R3.)	1001	R2 and R3 are enabled. (Function selected by FunSel will be applied to R2 and R3.)
0010	R1, R2, and R4 are enabled. (Function selected by FunSel will be applied to R1, R2, and R4.)	1010	R2 and R4 are enabled. (Function selected by FunSel will be applied to R2 and R4.)
0011	R1 and R2 are enabled. (Function selected by FunSel will be applied to R1 and R2.)	1011	Only R2 is enabled. (Function selected by FunSel will be applied to R2.)
0100	R1, R3, and R4 are enabled. (Function selected by FunSel will be applied to R1, R3, and R4.)	1100	R3 and R4 are enabled. (Function selected by FunSel will be applied to R3 and R4.)
0101	R1 and R3 are enabled. (Function selected by FunSel will be applied to R1 and R3.)	1101	Only R3 is enabled. (Function selected by FunSel will be applied to R3.)
0110	R1 and R4 are enabled. (Function selected by FunSel will be applied to R1 and R4.)	1110	Only R4 is enabled. (Function selected by FunSel will be applied to R4.)
0111	Only R1 is enabled. (Function selected by FunSel will be applied to R1.)	1111	NO general purpose register is enabled. (All registers retain their values.)

Figure 8: RegSel controls [4]

ScrSel	Enable General Purpose Registers	ScrSel	Enable General Purpose Registers
0000	All general purpose registers are enabled. (Function selected by FunSel will be applied to S1, S2, S3, and S4.)	1000	S2, S3, and S4 are enabled. (Function selected by FunSel will be applied to S2, S3, and S4.)
0001	S1, S2, and S3 are enabled. (Function selected by FunSel will be applied to S1, S2, and S3.)	1001	S2 and S3 are enabled. (Function selected by FunSel will be applied to S2 and S3.)
0010	S1, S2, and S4 are enabled. (Function selected by FunSel will be applied to S1, S2, and S4.)	1010	S2 and S4 are enabled. (Function selected by FunSel will be applied to S2 and S4.)
0011	S1 and S2 are enabled. (Function selected by FunSel will be applied to S1 and S2.)	1011	Only S2 is enabled. (Function selected by FunSel will be applied to S2.)
0100	S1, S3, and S4 are enabled. (Function selected by FunSel will be applied to S1, S3, and S4.)	1100	S3 and S4 are enabled. (Function selected by FunSel will be applied to S3 and S4.)
0101	S1 and S3 are enabled. (Function selected by FunSel will be applied to S1 and S3.)	1101	Only S3 is enabled. (Function selected by FunSel will be applied to S3.)
0110	S1 and S4 are enabled. (Function selected by FunSel will be applied to S1 and S4.)	1110	Only S4 is enabled. (Function selected by FunSel will be applied to S4.)
0111	Only S1 is enabled. (Function selected by FunSel will be applied to S1.)	1111	NO general purpose register is enabled. (All registers retain their values.)

Figure 9: ScrSel controls [4]

```
input [15:0] I,
input [2:0] OutASel,
input [2:0] OutBSel,
input [2:0] FunSel,
input [3:0] RegSel,
input [3:0] ScrSel,
input Clock,
output reg [15:0] OutA,
output reg [15:0] OutB);
```

Figure 10: Register File's inputs and outputs

```
wire [15:0] Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8;
Register R1(.I(I), .E(!RegSel[3]), .FunSel(FunSel), .Clock(Clock), .Q(Q1));
Register R2(.I(I), .E(!RegSel[2]), .FunSel(FunSel), .Clock(Clock), .Q(Q2));
Register R3(.I(I), .E(!RegSel[1]), .FunSel(FunSel), .Clock(Clock), .Q(Q3));
Register R4(.I(I), .E(!RegSel[0]), .FunSel(FunSel), .Clock(Clock), .Q(Q4));

Register S1(.I(I), .E(!ScrSel[3]), .FunSel(FunSel), .Clock(Clock), .Q(Q5));
Register S2(.I(I), .E(!ScrSel[2]), .FunSel(FunSel), .Clock(Clock), .Q(Q6));
Register S3(.I(I), .E(!ScrSel[1]), .FunSel(FunSel), .Clock(Clock), .Q(Q7));
Register S4(.I(I), .E(!ScrSel[0]), .FunSel(FunSel), .Clock(Clock), .Q(Q8));
```

Figure 11: Register File's registers

```
3'b000: OutA = Q1;
3'b001: OutA = Q2;
3'b010: OutA = Q3;
3'b011: OutA = Q4;
3'b100: OutA = Q5;
3'b101: OutA = Q6;
3'b110: OutA = Q7;
3'b111: OutA = Q8;
```

Figure 12: Register File A output

```
3'b000: OutB = Q1;
3'b001: OutB = Q2;
3'b010: OutB = Q3;
3'b011: OutB = Q4;
3'b100: OutB = Q5;
3'b101: OutB = Q6;
3'b110: OutB = Q7;
3'b111: OutB = Q8;
```

Figure 13: Register File B output

2.3.3 Designing Address Register File (ARF)

We built a address register file with three 16-bit inputs registers. WE use the firs register module for this three registers:program counter (PC), address register(AR), and stack pointer (SP). We have FunSel and RegSel inputs. These inputs work like the Register's File inputs. We also have OutCSel and OutDSel control inputs. We use these inputs to choose which register will be connected to outputs. This can bee seen figure 14. All inputs and outputs can be seen figure 15. We use wires to connect our register to outputs. This can be seen figure 16 and 17.

OutCSel	OutC
00	PC
01	PC
10	AR
11	SP

OutDSel	OutD
00	PC
01	PC
10	AR
11	SP

Figure 14: OutCSel and OutDSel controls [5]

```
input [15:0]I,
input [1:0] OutCSel,
input [1:0] OutDSel,
input [2:0] FunSel,
input [2:0] RegSel,
input Clock,
output reg [15:0] OutC,
output reg [15:0] OutD);
```

Figure 15: Address Register File's inputs and outputs

```
wire [15:0] q1,q2,q3;
  Register PC(.I(I), .E(!RegSel[2]), .FunSel(FunSel), .Clock(Clock), .Q(q1));
  Register AR(.I(I), .E(!RegSel[1]), .FunSel(FunSel), .Clock(Clock), .Q(q2));
  Register SP(.I(I), .E(!RegSel[0]), .FunSel(FunSel), .Clock(Clock), .Q(q3));
```

Figure 16: Address Register File's registers

```
2'b00: OutC <= q1;
2'b01: OutC <= q1;
2'b10: OutC <= q2;
2'b11: OutC <= q3;
```

Figure 17: Address Register File's C output

```
2'b00: OutD <= q1;
2'b01: OutD <= q1;
2'b10: OutD <= q2;
2'b11: OutD <= q3;
```

Figure 18: Address Register File's D output

2.4 Designing an Arithmetic Logic Unit (ALU)

We built an arithmetic logic unit which has two 16-bit inputs, a 16-bit output, and a 4-bit output for zero, negative, carry, and overflow flags. We also have WF and FunSel inputs to control writing and select functions. Inputs and outputs can be seen in figure 19. We have 32 different functions(16 for 16-bits and 16 for 8-bits). ALUOut functions can be seen figure 20 and 21. FlagOut functions can be seen figure 22 and 23.

```
input [15:0] A,
input [15:0] B,
input [4:0] FunSel,
input WF,
input Clock,
output reg [15:0] ALUOut,
output reg [3:0] FlagsOut);

reg carry;
```

Figure 19: Arithmetic Logic Unit's inputs and outputs

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 20: Arithmetic Logic Unit's functions table [6]

```
5'b00000: ALUOut[7:0] = A[7:0];
5'b00001: ALUOut[7:0] = B[7:0];
5'b00010: ALUOut[7:0] = ~A[7:0];
5'b00011: ALUOut[7:0] = ~B[7:0];
5'b00100: {carry,ALUOut[7:0]} = A[7:0]+B[7:0];
5'b00101: {carry,ALUOut[7:0]} = A[7:0]+B[7:0]+FlagsOut[2];
5'b00110: {carry,ALUOut[7:0]} = A[7:0]-B[7:0];
5'b00111: ALUOut[7:0] = A[7:0]&B[7:0];
5'b01000: ALUOut[7:0] = A[7:0]|B[7:0];
5'b01001: ALUOut[7:0] = A[7:0]^B[7:0];
5'b01010: ALUOut[7:0] = ~(A[7:0]&B[7:0]);
5'b01011: {carry,ALUOut[7:0]} = {A[7:0],1'b0};
5'b01100: {ALUOut[7:0],carry} = {1'b0,A[7:0]};
5'b01101: {ALUOut[7:0],carry} = {A[7],A[7:0]};
5'b01110: {carry,ALUOut[7:0]} = {A[7:0],FlagsOut[2]};
5'b01111: {ALUOut[7:0],carry} = {FlagsOut[2],A[7:0]};
```

Figure 21: Arithmetic Logic Unit's ALUOut functions

```
5'b10000: ALUOut[15:0] = A[15:0];
5'b10001: ALUOut[15:0] = B[15:0];
5'b10010: ALUOut[15:0] = ~A[15:0];
5'b10011: ALUOut[15:0] = ~B[15:0];
5'b10100:{carry,ALUOut} = A+B;
5'b10101:{carry,ALUOut[15:0]} = A[15:0]+B[15:0]+FlagsOut[2];
5'b10110:{carry,ALUOut[15:0]} = A-B;
5'b10111:ALUOut[15:0] = A[15:0]&B[15:0];
5'b11000:ALUOut[15:0] = A[15:0]|B[15:0];
5'b11001:ALUOut[15:0] = A[15:0]^B[15:0];
5'b11010:ALUOut[15:0] = ~(A[15:0]&B[15:0]);
5'b11011:{carry,ALUOut[15:0]} = {A[15:0],1'b0};
5'b11100:{ALUOut[15:0],carry} = {1'b0,A[15:0]};
5'b11101:{ALUOut[15:0],carry} = {A[15],A[15:0]};
5'b11110:{carry,ALUOut[15:0]} = {A[15:0],FlagsOut[2]};
5'b11111:{ALUOut[15:0],carry} = {FlagsOut[2],A[15:0]};
```

Figure 22: Arithmetic Logic Unit's ALUOut functions

```
5'b00000: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b00001: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b00010: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b00011: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b00100: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],(carry ^ (A[7] ^ B[7] ^ ALUOut[7]))};
5'b00101: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],(carry  ^ (A[7] ^ B[7] ^ ALUOut[7]))};
5'b00110: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],(carry  ^ (A[7] ^ B[7] ^ ALUOut[7]))};
5'b00111: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b01000: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b01001: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b01010: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[7],FlagsOut[0]};
5'b01011: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],FlagsOut[0]};
5'b01100: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],FlagsOut[0]};
5'b01101: FlagsOut = {ALUOut==0?1:0,carry,FlagsOut[1],FlagsOut[0]};
5'b01110: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],FlagsOut[0]};
5'b01111: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[7],FlagsOut[0]};
```

Figure 23: Arithmetic Logic Unit's FlagOut functions

```
5'b10000: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b10001: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b10010: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b10011: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b10100: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],(carry  ^ (A[15] ^ B[15] ^ ALUOut[15]))};
5'b10101: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],(carry  ^ (A[15] ^ B[15] ^ ALUOut[15]))};
5'b10110: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],(carry  ^ (A[15] ^ B[15] ^ ALUOut[15]))};
5'b10111: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b11000: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b11001: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b11010: FlagsOut = {ALUOut==0?1:0,FlagsOut[2],ALUOut[15],FlagsOut[0]};
5'b11011: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],FlagsOut[0]};
5'b11100: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],FlagsOut[0]};
5'b11101: FlagsOut = {ALUOut==0?1:0,carry,FlagsOut[1],FlagsOut[0]};
5'b11110: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],FlagsOut[0]};
5'b11111: FlagsOut = {ALUOut==0?1:0,carry,ALUOut[15],FlagsOut[0]};
```

Figure 24: Arithmetic Logic Unit's FlagOut functions

2.5 Part-4

We implement our modules with MUXA, MUXB and MUXC. Firstly, we defined all our inputs and outputs. This can be seen figure 24. We have MuxASel, MuxBSel and MuxCSel inputs to control multiplexers. Multiplexers control table can can be seen figure 25. We also defined the wire and reg type variable to connect our modules. This can be seen figure 27. After that we integrated our modules into the circuit. This can be seen figure 28. Finally, we adjusted the output values of the multiplexers. This can be seen figure 29.

```
input [2:0] RF_OutASel,//
input [2:0] RF_OutBSel, //
input [2:0] RF_FunSel,     //
input [3:0] RF_RegSel,//
input [3:0] RF_ScrSel,  //
input [4:0] ALU_FunSel,//
input ALU_WF, //
input [1:0] ARF_OutCSel,//
input [1:0] ARF_OutDSel,//
input [2:0] ARF_FunSel,//
input [2:0] ARF_RegSel,//
input IR_LH,//
input IR_Write,//
input Mem_WR,//
input Mem_CS,//
input [1:0] MuxASel,//
input [1:0] MuxBSel,//
input MuxCSel,//
input Clock);
```

Figure 25: All inputs and outputs

MuxASel	MuxAOut
00	ALUOut
01	ARF OutC
10	Memory Output
11	IR (7:0)

MuxBSel	MuxBOut
00	ALUOut
01	ARF OutC
10	Memory Output
11	IR (7:0)

MuxCSel	MuxCOut
0	ALUOut(7-0)
1	ALUOut(15-8)

Figure 26: Multiplexers control table[7]

```
wire [15:0] OutC,OutA,OutB,ALUOut,Address,IROut;
wire [7:0] MemOut;
wire [3:0] Flags;
reg [15:0] MuxBOut,MuxAOut;
reg [7:0] MuxCOut;
```

Figure 27: Wire and Reg Variables

```
Memory MEM(.Address(Address), .Data(MuxCOut), .WR(Mem_WR), .CS(Mem_CS), .Clock(Clock), .MemOut(MemOut));

AddressRegisterFile ARF(.I(MuxBOut), .OutCSel(ARF_OutCSel), .OutDSel(ARF_OutDSel),
                  .FunSel(ARF_FunSel), .RegSel(ARF_RegSel), .Clock(Clock),
                  .OutC(OutC), .OutD(Address));
RegisterFile RF(.I(MuxAOut), .OutASel(RF_OutASel), .OutBSel(RF_OutBSel),
                                    .FunSel(RF_FunSel), .RegSel(RF_RegSel), .ScrSel(RF_ScrSel),
                                    .Clock(Clock), .OutA(OutA), .OutB(OutB));

InstructionRegister IR(.I(MemOut), .Write(IR_Write), .LH(IR_LH),
                     .Clock(Clock), .IROut(IROut));

 ArithmeticLogicUnit ALU( .A(OutA), .B(OutB), .FunSel(ALU_FunSel), .WF(ALU_WF),
                                            .Clock(Clock), .ALUOut(ALUOut), .FlagsOut(Flags));
```

Figure 28: Modules integration

```
case(MuxASel)
    2'b00: MuxAOut = ALUOut;
    2'b01: MuxAOut = OutC;
    2'b10: MuxAOut = MemOut;
    2'b11: MuxAOut = {{8{IROut[7]}},IROut[7:0]};
endcase
case(MuxBSel)
    2'b00: MuxBOut = ALUOut;
    2'b01: MuxBOut = OutC;
    2'b10: MuxBOut = MemOut;
    2'b11: MuxBOut = {{8{IROut[7]}},IROut[7:0]};
endcase
case(MuxCSel)
    1'b0: MuxCOut = ALUOut[7:0];
    1'b1: MuxCOut = ALUOut[15:8];
endcase
```

Figure 29: Multiplexers functions

3 RESULTS

For part 1, Our registers schematic can be seen figure 30. Terminal screenshots and waves can be seen figre 31 and 32.

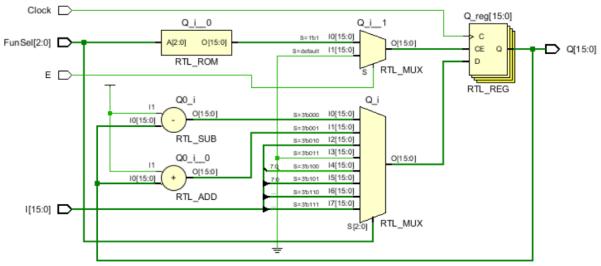

Figure 30: Register Schematic

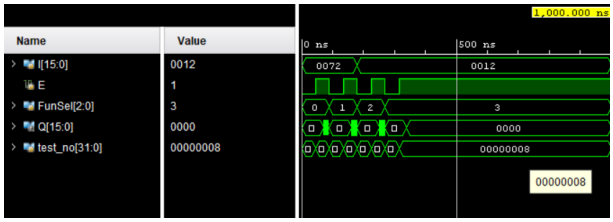

Figure 31: Register Waves

```
Register Simulation Started
[PASS] Test No: 1, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 2, Component: Q, Actual Value: 0x0024, Expected Value: 0x0024
[PASS] Test No: 3, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 4, Component: Q, Actual Value: 0x0026, Expected Value: 0x0026
[PASS] Test No: 5, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 6, Component: Q, Actual Value: 0x0012, Expected Value: 0x0012
[PASS] Test No: 7, Component: Q, Actual Value: 0x0025, Expected Value: 0x0025
[PASS] Test No: 8, Component: Q, Actual Value: 0x0000, Expected Value: 0x0000
Register Simulation Finished
0 Test Failed
8 Test Passed
-------------------------------------
```

Figure 32: Terminal Screenshots

For part 2a, Our IR schematic can be seen figure 33. Terminal screenshots and waves can be seen figure 34 and 35.

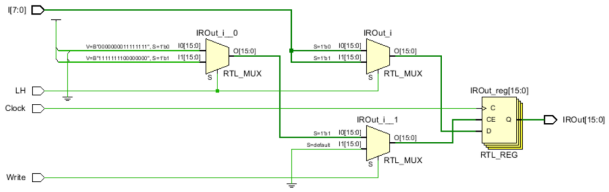

Figure 33: IR Schematic

Figure 34: IR Waves

```
-------------------------------------
InstructionRegister Simulation Started
[PASS] Test No: 1, Component: IROut, Actual Value: 0x2315, Expected Value: 0x2315
[PASS] Test No: 2, Component: IROut, Actual Value: 0x1567, Expected Value: 0x1567
InstructionRegister Simulation Finished
0 Test Failed
2 Test Passed
-------------------------------------
```

Figure 35: Terminal Screenshots

For part 2b, Our RF schematic can be seen figure 36. Terminal screenshots and waves can be seen figure 37 and 38.

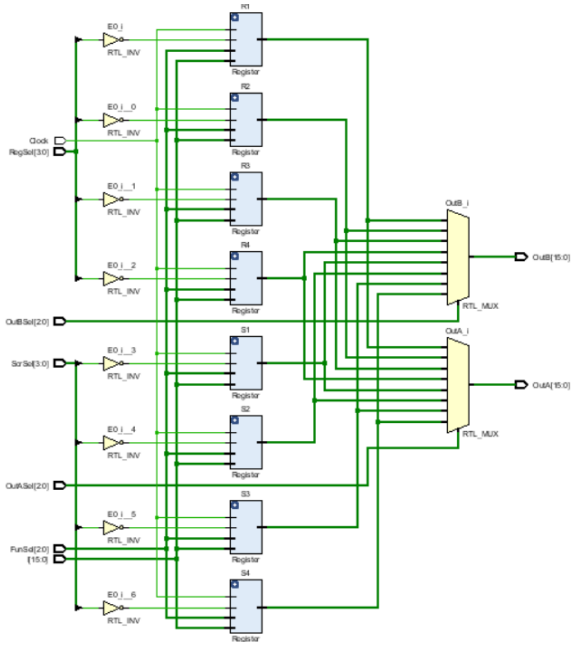

Figure 36: RF Schematic

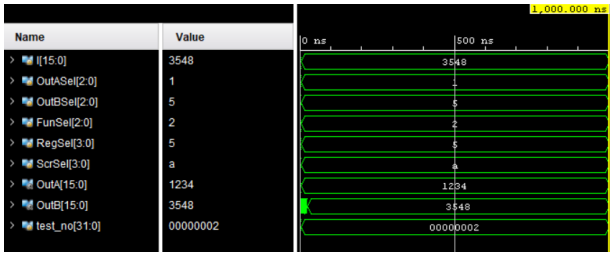

Figure 37: RF Waves

```
RegisterFile Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 1, Component: OutB, Actual Value: 0x5678, Expected Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected Value: 0x3548
RegisterFile Simulation Finished
0 Test Failed
4 Test Passed
-------------------------------------
```

Figure 38: Terminal Screenshots

For part 2c, Our ARF schematic can be seen figure 39. Terminal screenshots and waves can be seen figure 40 and 41.

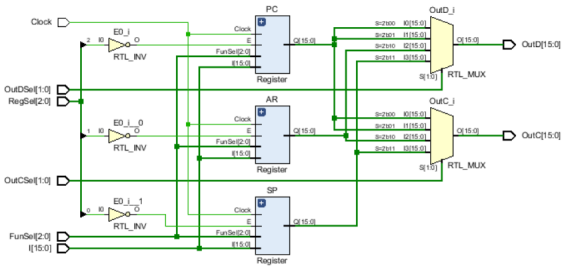

Figure 39: ARF Schematic

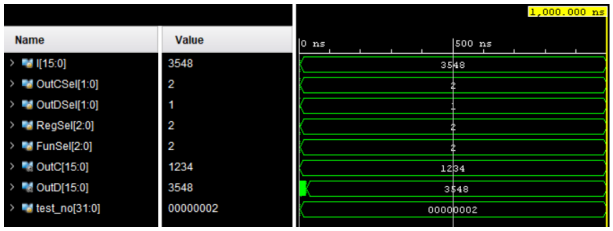

Figure 40: ARF Waves

```
AddressRegisterFile Simulation Started
[PASS] Test No: 1, Component: OutC, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 1, Component: OutD, Actual Value: 0x5678, Expected Value: 0x5678
[PASS] Test No: 2, Component: OutA, Actual Value: 0x1234, Expected Value: 0x1234
[PASS] Test No: 2, Component: OutB, Actual Value: 0x3548, Expected Value: 0x3548
AddressRegisterFile Simulation Finished
0 Test Failed
4 Test Passed
-------------------------------------
```

Figure 41: Terminal Screenshots

For part 3, Our ALU schematic can be seen figure 42. Terminal screenshots and waves can be seen figure 43 and 44.

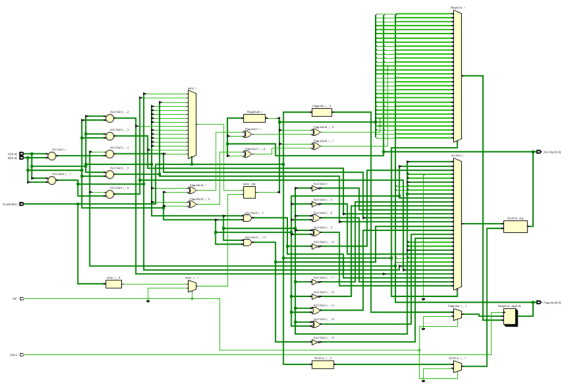

Figure 42: ALU Schematic

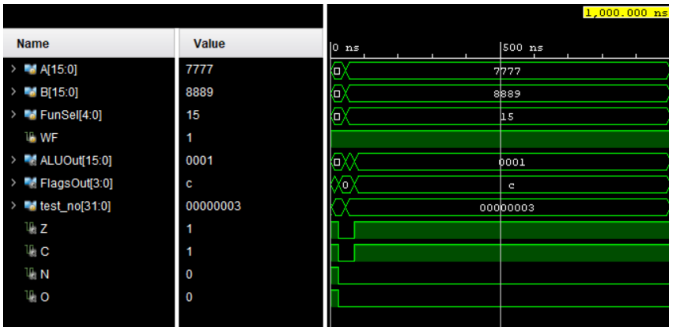

Figure 43: ALU Waves

```
ArithmeticLogicUnit Simulation Started
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 1, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 1, Component: O, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0x5555, Expected Value: 0x5555
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 3, Component: ALUOut, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: Z, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: C, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 3, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 3, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
ArithmeticLogicUnit Simulation Finished
0 Test Failed
15 Test Passed
--------------------------------------
```

Figure 44: Terminal Screenshots

For part 4, Our full schematic can be seen figure 42. Terminal screenshots and waves can be seen figure 43 and 44.

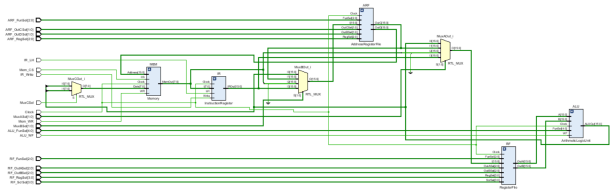

Figure 45: Project Schematic

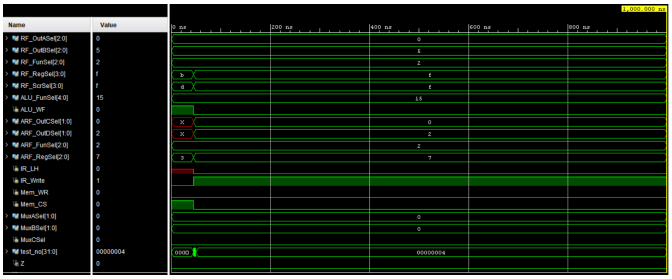

Figure 46: Project Waves

```
ArithmeticLogicUnitSystem Simulation Started
[PASS] Test No: 1, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 1, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 1, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: N, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 1, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[PASS] Test No: 1, Component: R2, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 1, Component: S3, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: OutA, Actual Value: 0x7777, Expected Value: 0x7777
[PASS] Test No: 2, Component: OutB, Actual Value: 0x8887, Expected Value: 0x8887
[PASS] Test No: 2, Component: ALUOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: Z, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: C, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: N, Actual Value: 0x0001, Expected Value: 0x0001
[PASS] Test No: 2, Component: O, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 2, Component: MuxAOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: MuxBOut, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: MuxCOut, Actual Value: 0x00fe, Expected Value: 0x00fe
[PASS] Test No: 2, Component: R2, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: S3, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 2, Component: PC, Actual Value: 0xfffe, Expected Value: 0xfffe
[PASS] Test No: 3, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 3, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 3, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 3, Component: IROut, Actual Value: 0x0000, Expected Value: 0x0000
[PASS] Test No: 4, Component: OutC, Actual Value: 0x1254, Expected Value: 0x1254
[PASS] Test No: 4, Component: Address, Actual Value: 0x0023, Expected Value: 0x0023
[PASS] Test No: 4, Component: Memout, Actual Value: 0x0015, Expected Value: 0x0015
[PASS] Test No: 4, Component: IROut, Actual Value: 0x0015, Expected Value: 0x0015
ArithmeticLogicUnitSystem Simulation Finished
0 Test Failed
33 Test Passed
--------------------------------------
```

Figure 47: Terminal Screenshots

4 DISCUSSION

The system is operated through control inputs, such as FunSel and RegSel, which are incorporated in the design of all components. Furthermore, the modules operate as a part of a whole system, with the output of one module serving as the input to another. It is worth noting that the ALU system is a sequential circuit that operates based on a single clock. It is important to mention that all tests have been successfully passed. At the conclusion of the project, we have successfully developed a system that is capable of performing fundamental arithmetic calculations based on selected control inputs. Additionally, the system has the capability to store and update data in registers.

5 CONCLUSION

In conclusion, we have gained a thorough understanding of designing a basic computer using Verilog HDL. Our proficiency in designing and simulating modules, as well as reusing them in other files, has been greatly enhanced. We also developed our teamwork and problem solving skills. At the beginning, we faced some challenges due to our limited experience with Verilog and needed extra resources to finish our project. In the end, we were able to complete the project successfully. However, we found that the provided test file was not enough, which led us to create new test cases.

REFERENCES

[1] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 1, February 2024.

[2] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 1, February 2024.

[3] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 2, February 2024.

[4] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 3, February 2024.

[5] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 4, February 2024.

[6] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 5, February 2024.

[7] Istanbul Technical University Department of Computer Engineering. Blg 222e project 1.pdf. page 7, February 2024.