ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION PROJECT REPORT

PROJECT NO : 1

GROUP NO : G71

GROUP MEMBERS:

150220901 : MOHAMAD CHAHADEH

150190068 : ÖZGÜR SEFEROĞLU

150200913 : FITNETE GUNI

SPRING 2023

Contents

0 Introduction

0.1 Project Parts

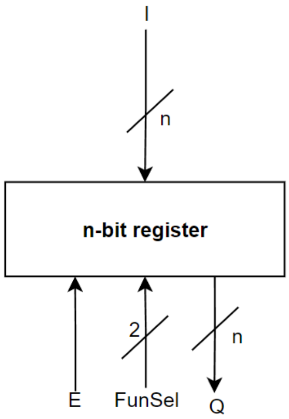
- Part 1: n-bit register
- Part 2: Register Files
  - Part 2a: 16-bit IR Register
  - Part 2b: Register File (RF)
  - Part 2c: Address Register File (ARF)
- Part 3: 8-bit ALU
- Part 4: Whole System Integration

0.2 Task Distribution

1. ÖZGÜR SEFEROĞLU: Part 3, Part 4
2. MOHAMAD CHAHADEH: Part 1, Part 2a, Part 2b, Part 4
3. FITNETE GUNI: Part 2c

1 Part 1

Implementing an n-bit Register, controlled using a 2-bit FunSel signal and an Enable signal. Figure 1 Shows the diagram and characteristic equation of Part 1.



E	FunSel	Q^+
0	ϕ	Q (retain value)
1	00	0 (Clear)
1	01	I (Load)
1	10	Q-1 (Decrement)
1	11	Q+1 (Increment)

Figure 1: Diagram and characteristic equation of Part 1

1.1 Implementation

the module implemented takes one parameter n, which is the number of bits of the register, a clock signal, two control inputs being FunSel and Enable, and one output of n-bits. the module is implemented by use the always block which executes on every positive edge of the clock, an if statement to check the enable signal, and a case block for the various inputs of FunSel. the following is the code for implementing the module.

```
module part1 #(parameter n = 4)
(input clk,
 input [1:0] FunSel,
 input [n-1:0] data_in,
 input enable,
 output reg [n-1:0] data_out);

    wire [n-1:0] zero = 0;
    always @(posedge clk)
    begin
        if (enable == 0)
            data_out <= data_out;
        else
            case (FunSel)
```

```
            2'b00: data_out <= zero;
            2'b01: data_out <= data_in;
            2'b10: data_out <= data_out - 1;
            2'b11: data_out <= data_out + 1;
        endcase
    end
endmodule
```

1.2 Simulation

Figure 2 Shows a simulation of the module implemented earlier of n = 4.

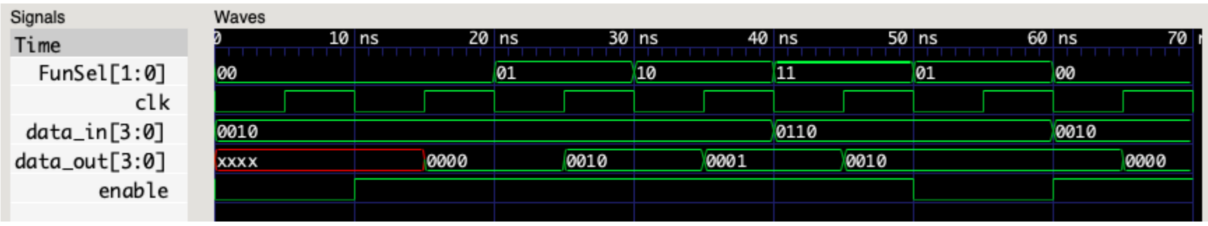
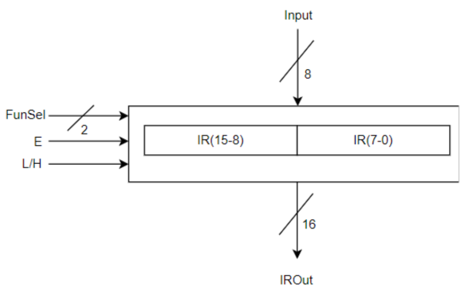


Figure 2: Simulation of Part 1, n-bit Register.

2 Part 2

2.1 Part 2a

Designing a 16-bit IR Register whose Diagram and Characteristic table is shown in Figure 3.



L'/H	E	FunSel	IR'
ϕ	0	ϕϕ	IR(Retain Value)
ϕ	1	00	0 (Clear)
0	1	01	IR(7-0) <- I (Load LSB)
1	1	01	IR(15-8) <- I (Load MSB)
ϕ	1	10	IR-1 (Decrement)
ϕ	1	11	IR+1(Increment)

Figure 3: Diagram and Characteristics of Part 2a

2.1.1 Implementation

To implement the IR register in Verilog, we need 1 data input and 3 control signals, it can be implemented as follows:

```
module part2a_IRreg(input clk, input[7:0] I, input [1:0] FunSel, input LH,
    input enable, output reg [15:0] data_out);

    always @(posedge clk)
    begin
        if (enable == 0)
            data_out = data_out;
        else
            if(FunSel == 2'b00) data_out = 16'b0000000000000000;
            else if(FunSel == 2'b01) begin
                if(LH == 0) data_out[7:0] = I;
                else if(LH == 1) data_out[15:8] = I;
            end
            else if(FunSel == 2'b10) data_out = data_out - 1;
            else if(FunSel == 2'b11) data_out = data_out + 1;
    end

endmodule
```

2.1.2 Simulation

Figure 4 Shows the simulation for the IR register implemented in this part.

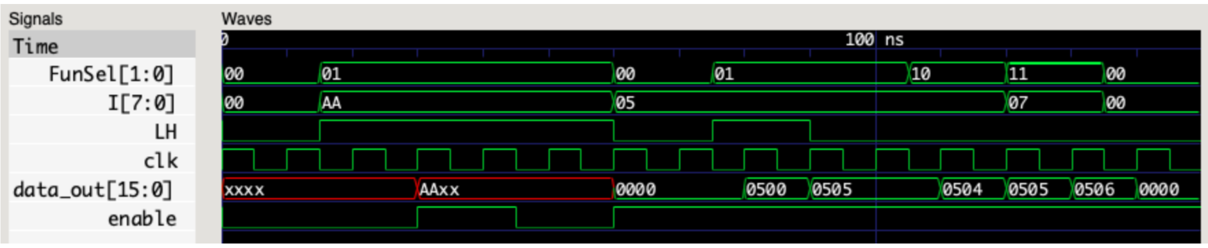
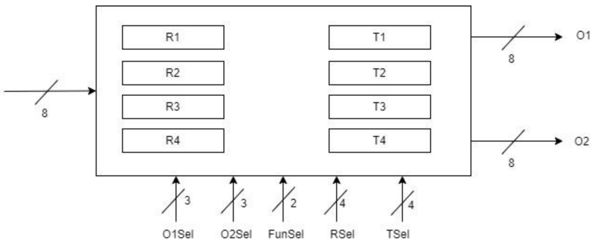


Figure 4: Simulation of part2a

2.2 Part 2b

Implementing a Register file with 4 General Registers and 4 Temporary Register with the following diagram and characteristics:



FunSel	R_x^+(Next State)
00	0 (Clear)
01	I (Load)
10	R_x-1 (Decrement)

Figure 5: Diagram of Part 2b

2.2.1 Implementation

The Register File implemented Takes in 6 inputs:

- A single 8-bit data input.
- O1Sel and O2Sel Output Control signals.
- FunSel Function control signal.
- RSel Register activation signal.
- TSel Temporary Register activation signal.

And it is implemented as follows:

```
module part2b ( input clk, input [7:0] I, input [2:0] O1Sel, input [2:0]
    O2Sel, input [1:0] FunSel, input [3:0] RSel, input [3:0] TSel, output reg
    [7:0] O1, O2 );

    reg [7:0] R1;
    reg [7:0] R2;
    reg [7:0] R3;
    reg [7:0] R4;

    reg [7:0] T1;
    reg [7:0] T2;
    reg [7:0] T3;
    reg [7:0] T4;

    always @ (posedge clk) begin

        if(FunSel == 2'b00) begin
            if(RSel[0] == 1)
                R4 = 8'b00000000;
```

```
    if(RSel[1] == 1)
        R3 = 8'b00000000;
    if(RSel[2] == 1)
        R2 = 8'b00000000;
    if(RSel[3] == 1)
        R1 = 8'b00000000;
    if(TSel[0] == 1)
        T4 = 8'b00000000;
    if(TSel[1] == 1)
        T3 = 8'b00000000;
    if(TSel[2] == 1)
        T2 = 8'b00000000;
    if(TSel[3] == 1)
        T1 = 8'b00000000;
end
else if(FunSel == 2'b01) begin
    if(RSel[0]) R4 = I;
    if(RSel[1]) R3 = I;
    if(RSel[2]) R2 = I;
    if(RSel[3]) R1 = I;
    if(TSel[0]) T4 = I;
    if(TSel[1]) T3 = I;
    if(TSel[2]) T2 = I;
    if(TSel[3]) T1 = I;
end
else if(FunSel == 2'b10) begin
    if(RSel[0]) R4 = R4 - 1;
    if(RSel[1]) R3 = R3 - 1;
    if(RSel[2]) R2 = R2 - 1;
    if(RSel[3]) R1 = R1 - 1;
    if(TSel[0]) T4 = T4 - 1;
    if(TSel[1]) T3 = T3 - 1;
    if(TSel[2]) T2 = T2 - 1;
    if(TSel[3]) T1 = T1 - 1;
end
else if(FunSel == 2'b11) begin
    if(RSel[0]) R4 = R4 + 1;
    if(RSel[1]) R3 = R3 + 1;
    if(RSel[2]) R2 = R2 + 1;
    if(RSel[3]) R1 = R1 + 1;
```

```
            if(TSel[0]) T4 = T4 + 1;
            if(TSel[1]) T3 = T3 + 1;
            if(TSel[2]) T2 = T2 + 1;
            if(TSel[3]) T1 = T1 + 1;
        end

        if( O1Sel == 3'b000) O1 = T1;
        else if( O1Sel == 3'b001) O1 = T2;
        else if( O1Sel == 3'b010) O1 = T3;
        else if( O1Sel == 3'b011) O1 = T4;
        else if( O1Sel == 3'b100) O1 = R1;
        else if( O1Sel == 3'b101) O1 = R2;
        else if( O1Sel == 3'b110) O1 = R3;
        else if( O1Sel == 3'b111) O1 = R4;


        if( O2Sel == 3'b000) O2 = T1;
        else if( O2Sel == 3'b001) O2 = T2;
        else if( O2Sel == 3'b010) O2 = T3;
        else if( O2Sel == 3'b011) O2 = T4;
        else if( O2Sel == 3'b100) O2 = R1;
        else if( O2Sel == 3'b101) O2 = R2;
        else if( O2Sel == 3'b110) O2 = R3;
        else if( O2Sel == 3'b111) O2 = R4;

    end
endmodule
```

2.2.2 Simulation

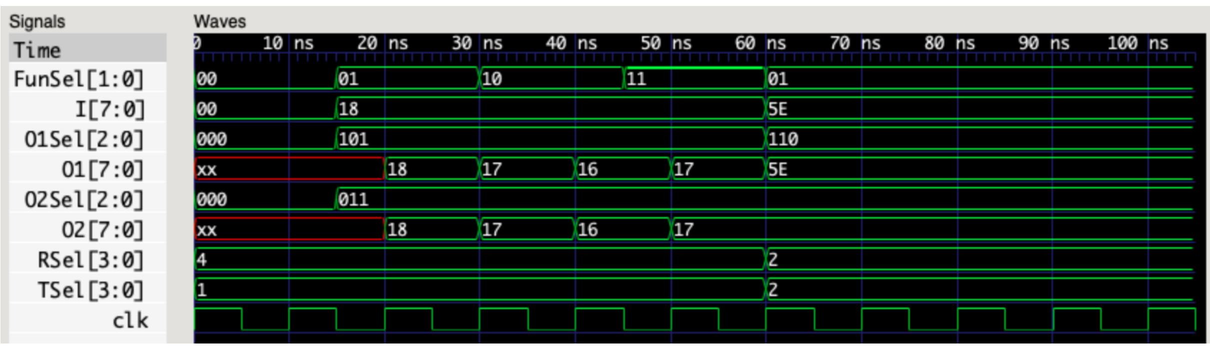


Figure 6: Simulation of Part2b, Register File

2.3 Part 2c

an Address Register File which consists of four 8-bit address registers as shown in Figure 7.

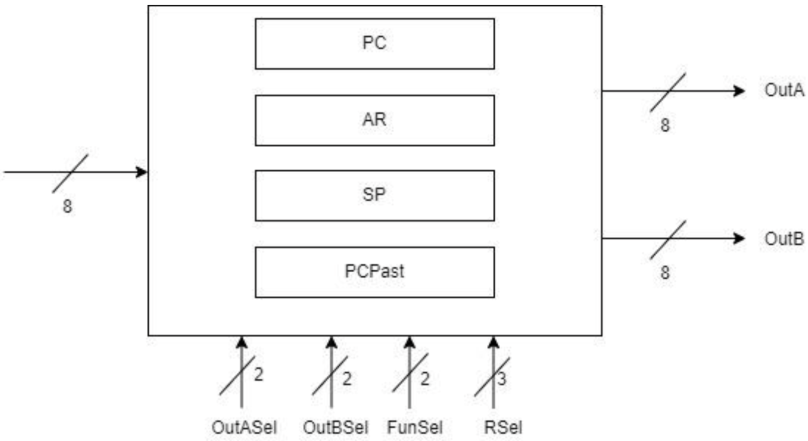


Figure 7: Diagram of Part 2c, Address Register File

This Register will use the same Characteristic function as the previous part.

2.3.1 Implementation

The Address Register File implemented Takes in 6 inputs:

- A single 8-bit data input.
- OutASel and OutBSel Output Control signals.
- FunSel Function control signal.
- RSel Register activation signal.

it is implemented in verilog as such:

```
module part2c_ARF( input clk, input [7:0] I, input [1:0] OutASel, input [1:0]
    OutBSel, input [1:0] FunSel, input [3:0] RSel, output reg [7:0] OutA,
    output reg [7:0] OutB);

reg [7:0] PC;
reg [7:0] AR;
reg [7:0] SP;
reg [7:0] PCPast;
```

```
always @ (posedge clk) begin

    if(FunSel == 2'b00)
    begin
        if(RSel[3] == 1)
            PC = 8'b00000000;
        if(RSel[2] == 1)
            AR = 8'b00000000;
        if(RSel[1] == 1)
            SP = 8'b00000000;
        if(RSel[0] == 1)
            PCPast = 8'b00000000;
    end
    else if(FunSel == 2'b01)
    begin
        if(RSel[3] == 1)
            PC = I;
        if(RSel[2] == 1)
            AR = I;
        if(RSel[1] == 1)
            SP = I;
        if(RSel[0] == 1)
            PCPast = I;
    end
    else if(FunSel == 2'b10)
    begin
        if(RSel[3] == 1)
            PC = PC + 1;
        if(RSel[2] == 1)
            AR = AR + 1;
        if(RSel[1] == 1)
            SP = SP + 1;
        if(RSel[0] == 1)
            PCPast = PCPast + 1;
    end
    else if(FunSel == 2'b11)
    begin
        if(RSel[3] == 1)
            PC = PC - 1;
        if(RSel[2] == 1)
```

```
            AR = AR - 1;
        if(RSel[1] == 1)
            SP = SP - 1;
        if(RSel[0] == 1)
            PCPast = PCPast - 1;
    end



    if(OutASel == 2'b00) OutA = AR;
    else if(OutASel == 2'b01) OutA = SP;
    else if(OutASel == 2'b10) OutA = PCPast;
    else if(OutASel == 2'b11) OutA = PC;

    if(OutBSel == 2'b00) OutB = AR;
    else if(OutBSel == 2'b01) OutB = SP;
    else if(OutBSel == 2'b10) OutB = PCPast;
    else if(OutBSel == 2'b11) OutB = PC;

end

endmodule
```

2.3.2 Simulation

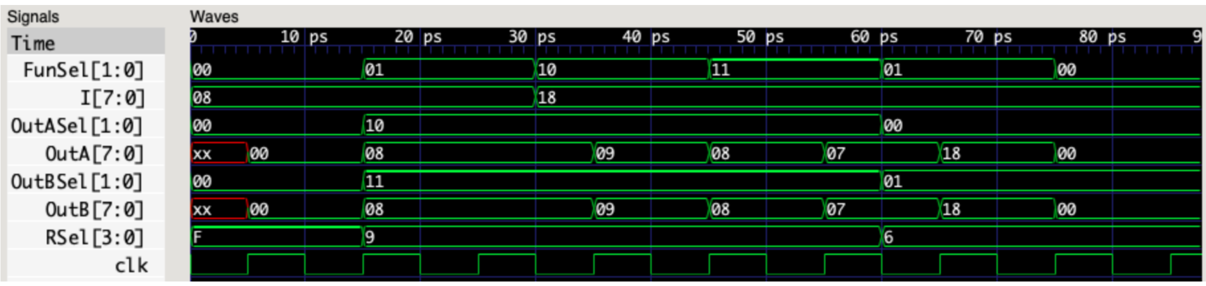
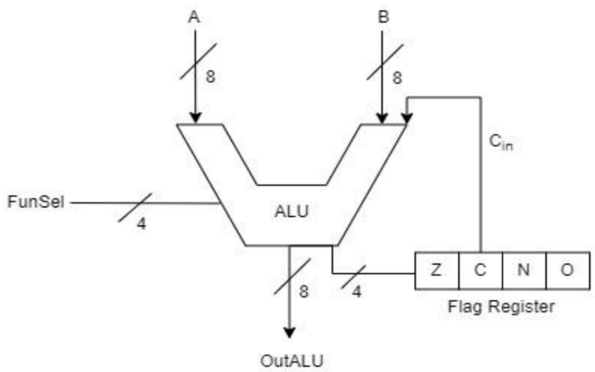


Figure 8: Simulation of Part 2c, address Register File

3 Part 3

Implementing an 8-bit ALU Module that does the following:



FunSel	OutALU	Z	C	N	O
0000	A	+	-	+	-
0001	B	+	-	+	-
0010	NOT A	+	-	+	-
0011	NOT B	+	-	+	-
0100	A + B	+	+	+	+
0101	A - B	+	+	+	+
0110	Compare A, B	+	+	+	+
0111	A AND B	+	-	+	-
1000	A OR B	+	-	+	-
1001	A NAND B	+	-	+	-
1010	A XOR B	+	-	+	-
1011	LSL A	+	+	+	-
1100	LSR A	+	+	+	-
1101	ASL A	+	-	+	+
1110	ASR A	+	-	-	-
1111	CSR A	+	+	+	-

Figure 9: Diagram of ALU module and its characteristics

3.1 Implementation

```
module part3_ALU (input clk, input [7:0] A, input [7:0] B, input [3:0] FunSel,
    output reg [7:0] OutALU, output reg [3:0] Flags);

    reg [7:0] B_neg;
    reg cout;

    always @(posedge clk) begin
        B_neg = (~B) + 8'b00000001; // 2's complement of B
        cout = Flags[2];

        if(FunSel == 4'b0000)
            OutALU = A;
        else if(FunSel == 4'b0001)
            OutALU = B;
        else if(FunSel == 4'b0010)
            OutALU = ~A;
        else if(FunSel == 4'b0011)
```

```
    OutALU = ~B;
else if(FunSel == 4'b0100) begin // A+B
    {cout, OutALU} = {1'b0, A} + {1'b0, B};
    if(cout == 1) Flags[0] = 1;
    else Flags[0] = 0;
end
else if(FunSel == 4'b0101)begin
    {cout, OutALU} = {1'b0, A} + {1'b0, B_neg};
    if(cout !== OutALU[7]) Flags[0] = 1; //Overflow
    else Flags[0] = 0;
end
else if(FunSel == 4'b0110)
    begin
        if(A > B) OutALU = A;
        else OutALU = 0;
    end
else if(FunSel == 4'b0111)
    OutALU = A & B;
else if(FunSel == 4'b1000)
    OutALU = A | B;
else if(FunSel == 4'b1001)
    OutALU = ~(A & B);
else if(FunSel == 4'b1010)
    OutALU = (~A & B) | (A & ~B);
else if(FunSel == 4'b1011) begin // LSL
    cout = A[7];
    OutALU = A << 1;
end
else if (FunSel == 4'b1100) begin //LSR
    cout = A[0];
    OutALU = A >> 1;
end
else if (FunSel == 4'b1101) //ASL
    OutALU = A << 1;
else if (FunSel == 4'b1110)
    OutALU = {A[7], A[7:1]};
else if (FunSel == 4'b1111) begin //CSR
    cout = A[0];
    OutALU = {Flags[2], A[7:1]};
end
```

```
        // Set flags

        if (OutALU == 8'b00000000) Flags[3] = 1; // Z Flag
        else Flags[3] = 0;

        Flags[2] = cout; // C Flag

        if (OutALU[7] == 1) Flags[1] = 1; // N Flag
        else Flags[1] = 0;

    end
endmodule
```

3.2 Simulation

Figure 10 Shows the simulation of the ALU module implemented in part 3.

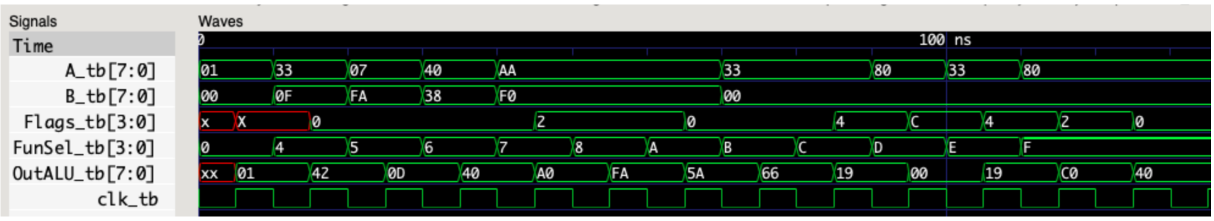


Figure 10: Simulation of Part 3, ALU Module.

4 Part 4

Combining all the module to implement a complete ALU System. Figure 11 Shows the diagram of the system implement in this part.

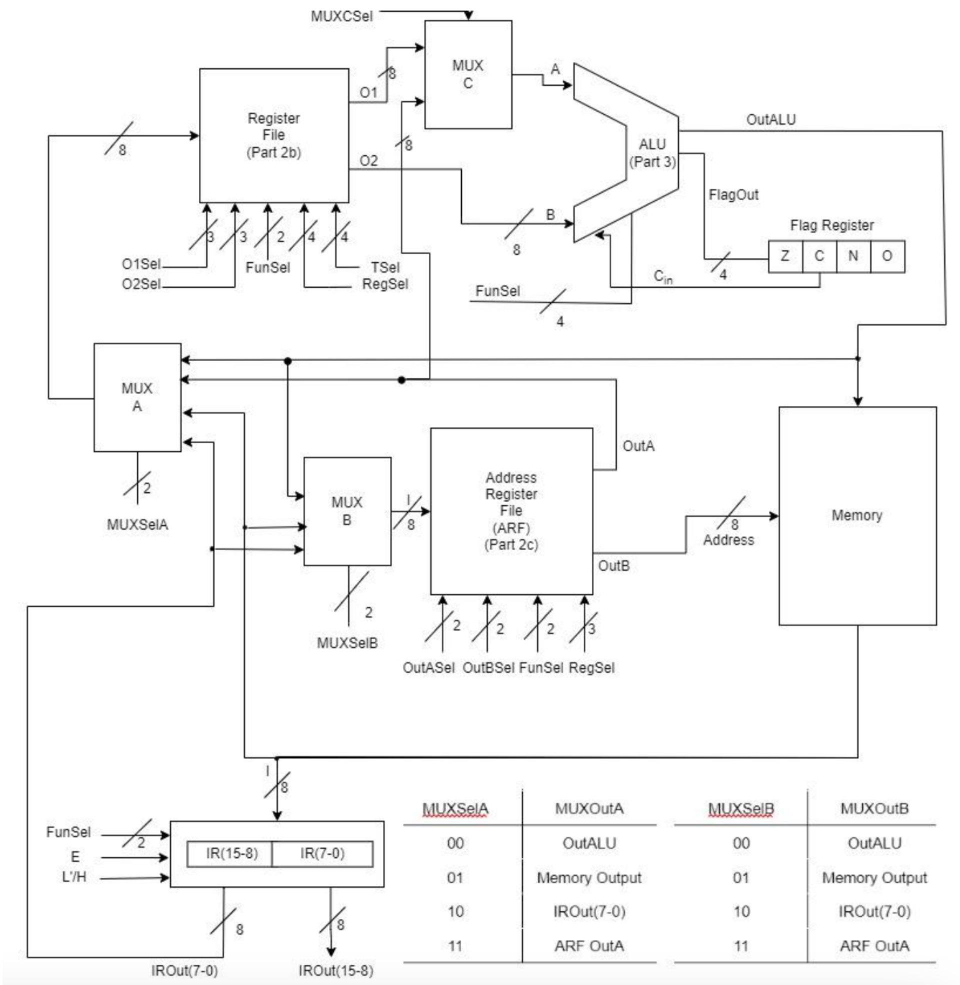


Figure 11: Diagram of the ALU System

4.1 Implementation

We can implement the system by combining all the module implement by implementing all the previous module in the module as shown:

```
module ALUSystem( input[2:0] RF_O1Sel, input[2:0] RF_O2Sel, input[1:0]
    RF_FunSel, input[3:0] RF_RSel, input[3:0] RF_TSel, input[3:0] ALU_FunSel,
    input[1:0] ARF_OutASel, input[1:0] ARF_OutBSel, input[1:0] ARF_FunSel,
    input[3:0] ARF_RSel, input IR_LH, input IR_Enable, input[1:0] IR_FunSel,
    input Mem_WR, input Mem_CS, input[1:0] MuxASel, input[1:0] MuxBSel, input
    MuxCSel, input Clock );
```

```
    wire [7:0] MemOut;
    wire [7:0] RF_O1,RF_O2;
    wire [7:0] MuxAOut, MuxBOut, MuxCOut;
    wire [3:0] ALU_FlagOut;
    wire [7:0] ALU_Out;
    wire [7:0] ARF_OutA, ARF_OutB;
    wire [15:0] IR_Out;

    part2c_ARF ARF(Clock, MuxBOut, ARF_OutASel, ARF_OutBSel, ARF_FunSel,
       ARF_RSel, ARF_OutA, ARF_OutB);

    mux_4to1 MuxA(Clock, MuxASel, ALU_Out, MemOut, IR_Out[7:0], ARF_OutA,
       MuxAOut);

    mux_4to1 MuxB(Clock, MuxBSel, ALU_Out, MemOut, IR_Out[7:0], ARF_OutA,
       MuxBOut);

    part2b_RF RF(Clock, MuxAOut, RF_O1Sel, RF_O2Sel, RF_FunSel, RF_RSel,
       RF_TSel, RF_O1, RF_O2);

    mux_2to1 MuxC(Clock, MuxCSel, RF_O1, ARF_OutA, MuxCOut);

    part3_ALU ALU(Clock, MuxCOut, RF_O2, ALU_FunSel, ALU_Out, ALU_FlagOut);

    Memory Mem(Clock, ARF_OutB, ALU_Out, Mem_WR, Mem_CS, MemOut);

    part2a_IRreg IR(Clock, MemOut, IR_FunSel, IR_LH, IR_Enable, IR_Out);

endmodule
```

4.2 Simulation

By using the TestBench supplied with the Project, we get the following results shown in Figure 12.

```
Input Values:
Operation: x
Register File: O1Sel: x, O2Sel: x, FunSel: x, RSel:  x, TSel:  x
ALU FunSel:  x
Addres Register File: OutASel: x, OutBSel: x, FunSel: x, Regsel:  x
Instruction Register: LH: x, Enable: x, FunSel: x
Memory: WR: x, CS: x
MuxASel: x, MuxBSel: x, MuxCSel: x

Output Values:
Register File: AOut:   x, BOut:   x
ALUOut:   x, ALUOutFlag:  X, ALUOutFlags: Z:0, C:x, N:0, O:x,
Address Register File: AOut:   x, BOut (Address):   x
Memory Out:   x
Instruction Register: IROut:     x
MuxAOut:   x, MuxBOut:   x, MuxCOut:   x

--------------------------------------------

Input Values:
Operation: 1
Register File: O1Sel: 4, O2Sel: 5, FunSel: 1, RSel: 12, TSel:  0
ALU FunSel:  3
Addres Register File: OutASel: 3, OutBSel: 3, FunSel: 3, Regsel: 15
Instruction Register: LH: 0, Enable: 0, FunSel: 0
Memory: WR: 0, CS: 1
MuxASel: 3, MuxBSel: 1, MuxCSel: 1

Output Values:
Register File: AOut:   x, BOut:   x
ALUOut:   x, ALUOutFlag:  X, ALUOutFlags: Z:0, C:x, N:0, O:x,
Address Register File: AOut:   x, BOut (Address):   x
Memory Out:   z
Instruction Register: IROut:     x
MuxAOut:   x, MuxBOut:   z, MuxCOut:   x

--------------------------------------------

Input Values:
Operation: 1
Register File: O1Sel: 4, O2Sel: 6, FunSel: 1, RSel: 10, TSel:  0
ALU FunSel: 12
Addres Register File: OutASel: 3, OutBSel: 3, FunSel: 3, Regsel: 15
Instruction Register: LH: 0, Enable: 0, FunSel: 0
Memory: WR: 0, CS: 1
MuxASel: 3, MuxBSel: 3, MuxCSel: 1

Output Values:
Register File: AOut:   x, BOut:   x
ALUOut:   X, ALUOutFlag:  X, ALUOutFlags: Z:0, C:x, N:0, O:x,
Address Register File: AOut:   x, BOut (Address):   x
Memory Out:   z
Instruction Register: IROut:     x
MuxAOut:   x, MuxBOut:   x, MuxCOut:   x

--------------------------------------------
```

Figure 12: Simulation of Part 4