

Arithmetic Logic Unit

Date: 29/11/2025

***Processor
Architecture
(EECG241-17)***

Our Team

Code	Section	Name	ID
91240733	4	مريم احمد محمد علي رضا	1
91240202	1	باسل شريف حميد مصطفى	2

Instructor:
Eng. Ahmed Atef

Codes are Uploaded in our [Github Repository](#)

Table of contents:

1- Introduction

2- ALU Architecture overview

1. Arithmetic Unit
2. Logic Unit
3. Shift Unit

3- Verilog Code

1. Arithmetic Unit
2. Logic Unit
3. Shift Unit
4. ALU Unit
5. ALU_TB Unit

4- Testbench

5- Do File

1. Introduction

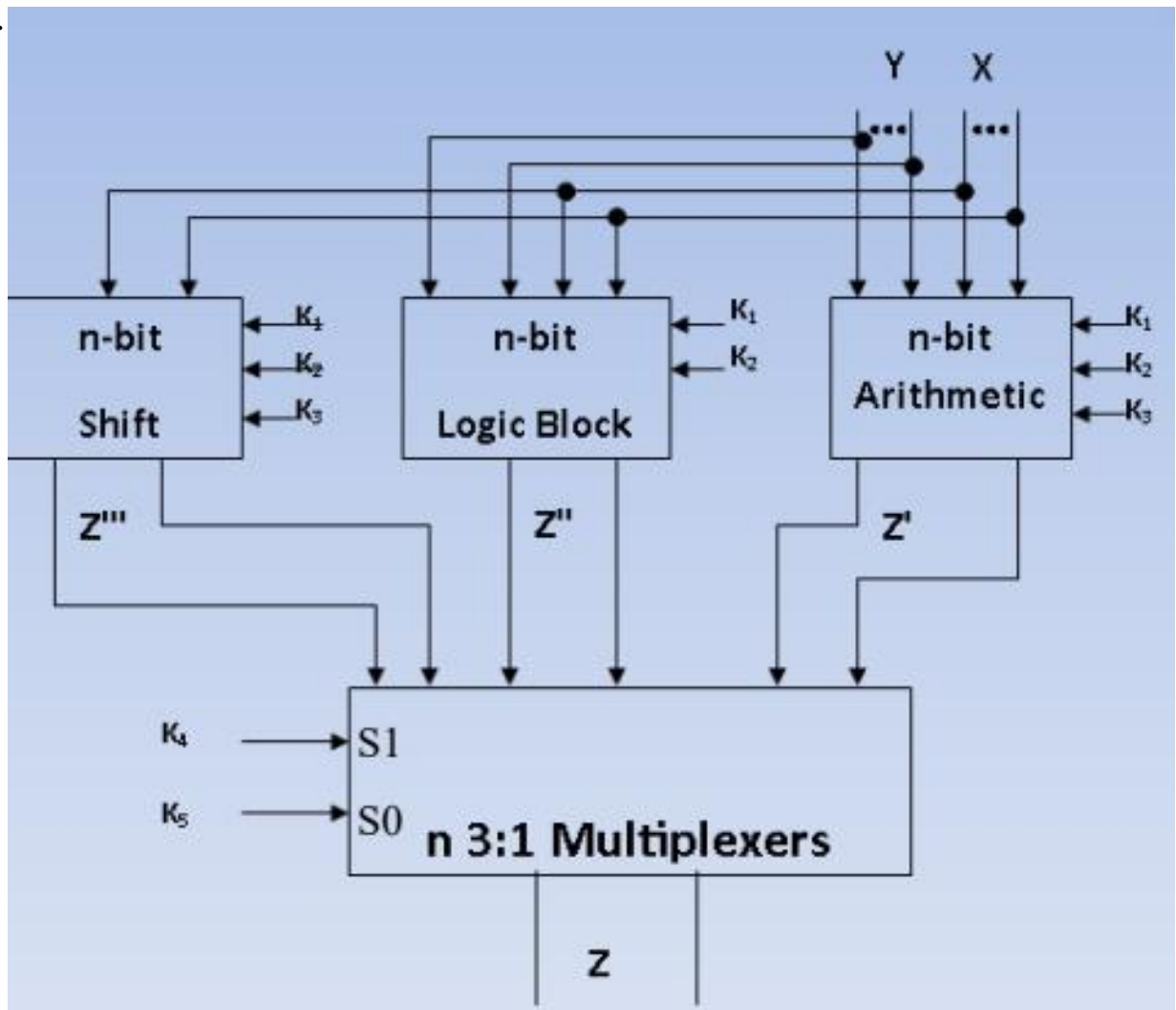
In this project, we designed a **16-bit combinational ALU** that implements the required **20 functions** across three major blocks:

- **Arithmetic Unit**
- **Logic Unit**
- **Shift/Rotate Unit**

Our Project ALU Codes are uploaded to the [Github Repository](#)

The main objective was to implement all operations and generate the 6 status flags (C, Z, N, V, P, A) as required in the assignment brief (page 1–3 of the PDF)

The main Design of the ALU was from Dr. Ibrahim Qamar Slides in Course Processor Architecture.



2. ALU Architecture Overview

The ALU uses the upper bits of F[4:3] to select a block:

00 \rightarrow Arithmetic

01 \rightarrow Logic

10 \rightarrow Shift/Rotate

2.1 Arithmetic Unit

Implements: INC, DEC, ADD, ADD+CIN, SUB, SUB-CIN.







Generates flags: C, Z, N, V, P, Af.

		Mems															
<div><div></div><div></div><div></div><div></div><div></div><div></div></div>	/ALU_0b/A	0011010100100100	0011010100100100	1101011000001001	0111101100001101	1000010001100101	1110001100000001	1111000101101110	0101011111101101	1110100111111001	1000010011000101	1111...					
	/ALU_0b/B	01011111010000001	0101111101000001	0101011001100011	1001100110001101	0101001000010010	1100110100001101	1100110100111101	1111011110001100	0010010011000110	1101001010101010	0111...					
	/ALU_0b/F	00000	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001					
	/ALU_0b/Cin	1															
	/ALU_0b/Out	00000000000000000	0000000000000000	1101011000001010	0000000000000000	1000010001100100	1011000000001110	1011111010101000	0110000001100001	1100010100110010	1000000010000000	1111...					
	/ALU_0b/Status	000000	000000	001000	000000	001000		101010		100000	001000	001010					

2.2 Logic Unit

Implements: AND, OR, XOR, NOT.

Generates Z, N, P.

		Msgs															
	/ALU_tb/A	001101010100100100	1000010011000101	1111011111100101	1101011000010010	0110100111110010	0111101011101000	0100100101011100	0101100000101101	010001001100011	0010001010000000	01...					
	/ALU_tb/B	0101111010000001	00000	011001001110111	1101101110001111	1001011011001110	0100111011000101	0010100010111101	0010011001100101	100001110001010	0010000100100000	11...					
	/ALU_tb/F	00000	01000	01001	01010	01011	01100	01101	01110	01111	10000	10...					
	/ALU_tb/Cin	1															
	/ALU_tb/Out	00000000000000000	1000000001000000	1111011111110111	0000110110011101	1001011000001101	0000000000000000					0100010100000000	00...				
	/ALU_tb/Status	000000										000000					

2.3 Shift Unit

Implements: SHL, SHR, SAL, SAR, ROL, ROR, RCL, RCR.

Carry = shifted/rotated-out bit.

		Msgs	
	/ALU_0b/A	001101010100100100	0100001010000000 0100010101010101 0011111010101110 0011100000001101 1101110101101011 0100101000000010 1110100100011101 0100100100100011 0000101011010101 10...
	/ALU_0b/B	010111110100000001	0010000100100000 1100110010011101 1011100000010011 1101011001010011 0010101011010101 0011111010101110 0111001011001111 0110010100010101 0100110000111100 01...
	/ALU_0b/F	00000	010000 10001 10010 10011 10100 10101 10110 10110 10100 11000
	/ALU_0b/Cn	1	11...
	/ALU_0b/Out	0000000000000000	0100010100000000 0010001011010101 0111110100010100 0001110000000110 1011101011010111 0010010100000001 1101001000111010 1010010010010001 0000000000000000
	/ALU_0b/Status	0000000	000000 100000 101000 000010 101010 000000 101010 000000

3. Verilog Code

3.1 Arithmetic Unit

```

tmodule Arithmetic
#(parameter Width=16)
(
    input    [Width-1:0] A,B,
    input    Cin,
    input    [2:0] F,
    output reg [Width-1:0] Out,
    output reg Cout,
    output reg [5:0] Status,
    output reg C,Z,N,V,P,Af
);

reg [Width:0] temp;
reg overflow;

always @(*)
begin

    case(F)
    3'b001:                                //INC
    begin
        temp=A+1;
        Out=temp[Width-1:0];
        Cout=temp[Width];
        if(A[Width-1]^Out[Width-1])
            overflow=1'b1;
        else
            overflow=1'b0;
    end
    //////////////////////////////////////
    3'b011:                                //DEC
    begin
        temp=A-1;
        Out=temp[Width-1:0];
        Cout=temp[Width];
        if(A[Width-1]^Out[Width-1])
            overflow=1'b1;
        else
            overflow=1'b0;
    end
    //////////////////////////////////////
    3'b100:                                //ADD
    begin
        temp=A+B;
        Out=temp[Width-1:0];
        Cout=temp[Width];
        if( (A[Width-1] & B[Width-1] & ~Out[Width-1]) | ( ~A[Width-1] & ~B[Width-1] & Out[Width-1]) )
            overflow=1'b1;
        else
            overflow=1'b0;
    end
    //////////////////////////////////////
    3'b101:                                //ADD_CARRY
    begin

```

ALU

```
        temp=A+B+Cin;
        Out=temp[Width-1:0];
        Cout=temp[Width];
        if( (A[Width-1] & B[Width-1]& ~Out[Width-1]) | ( ~A[Width-1] & ~B[Width-1]& Out[Width-1]) )
            overflow=1'b1;
        else
            overflow=1'b0;
    end
////////////////////////////////////
3'b110:                                     //SUB
    begin
        temp=A-B;
        Out=temp[Width-1:0];
        Cout=temp[Width];
        if( (A[Width-1] & ~B[Width-1]& ~Out[Width-1]) | ( ~A[Width-1] & B[Width-1]& Out[Width-1]) )
            overflow=1'b1;
        else
            overflow=1'b0;
    end
////////////////////////////////////
3'b111:                                     //SUB_BORROW
    begin
        temp=A-B-Cin;
        Out=temp[Width-1:0];
        Cout=temp[Width];
        if( (A[Width-1] & ~B[Width-1]& ~Out[Width-1]) | ( ~A[Width-1] & B[Width-1]& Out[Width-1]) )
            overflow=1'b1;
        else
            overflow=1'b0;
    end
////////////////////////////////////
    default:
    begin
        Out=0;
        Cout=0;
        overflow=0;
    end
////////////////////////////////////
    endcase
////////////////////////////////////

C = (F[2] | F[0]) ? Cout           : 1'b0;           // Carry Flag
Z = (F[2] | F[0]) ? (Out=={Width{1'b0}})      : 1'b0;           // Zero Flag
N = (F[2] | F[0]) ? Out[Width-1]              : 1'b0;           // Negative Flag
V = (F[2] | F[0]) ? overflow                  : 1'b0;           // Overflow Flag
P = (F[2] | F[0]) ? ~^Out                     : 1'b0;           // Parity Flag
Af = (F[2] | F[0]) ? ((A[3:0]+B[3:0]+Cin)>4'hF) : 1'b0;           // Auxiliary Flag

// The if condition here is just to make the Flags = 0 when the two not used OP Codes in the Project Details

    Status={C,Z,N,V,P,Af}; // Concatenate

end
endmodule
```


3.2 Logic Unit

```

module Logic
#(parameter Width = 16)
(
    input    [Width-1:0] A,B,
    input    [2:0] F,
    output reg Z,N,P,
    output reg [Width-1:0] Out
);

always@(*) begin
    case(F)
        3'b000:                                //AND
        begin
            Out = A & B;
            Z = ~|Out;                          //Zero Flag
            N = Out[Width-1];                    //Negative Flag
            P = ~^Out;                          //Parity Flag
        end
        //////////////////////////////////////////////////
        3'b001:                                //OR
        begin
            Out = A | B;
            Z = ~|Out;                          //Zero Flag
            N = Out[Width-1];                    //Negative Flag
            P = ~^Out;                          //Parity Flag
        end
        //////////////////////////////////////////////////
        3'b010:                                //XOR
        begin
            Out = A ^ B;
            Z = ~|Out;                          //Zero Flag
            N = Out[Width-1];                    //Negative Flag
            P = ~^Out;                          //Parity Flag
        end
        //////////////////////////////////////////////////
        3'b011:                                //NOT
        begin
            Out = ~A;
            Z = ~|Out;                          //Zero Flag
            N = Out[Width-1];                    //Negative Flag
            P = ~^Out;                          //Parity Flag
        end
        //////////////////////////////////////////////////
        default:
        begin
            Out = 1'b0;
            Z = 1'b0;                          //Zero Flag
            N = 1'b0;                          //Negative Flag
            P = 1'b0;                          //Parity Flag
        end
    endcase
end
endmodule

```

3.3 Shift Unit

```

module Shift
#(parameter Width=16)
(
    input    [Width-1:0] A,
    input    [2:0] F,
    output reg C,Z,N,P,
    output reg [Width-1:0] Out
);

reg Cshift;

always @(*)
begin
    case (F)
        3'b000:                                //SHL
            {C,Out} = {A,1'b0};
        ///////////////////////////////////////////////////
        3'b001:                                //SHR
            {Out,C} = {1'b0,A};
        ///////////////////////////////////////////////////
        3'b010:                                //SAL
            {C,Out} = {A,1'b0};
        ///////////////////////////////////////////////////
        3'b011:                                //SAR
            {Out,C} = {A[Width-1],A};
        ///////////////////////////////////////////////////
        3'b100:                                //ROL
            {C,Out} = {A,A[Width-1]};
        ///////////////////////////////////////////////////
        3'b101:                                //ROR
            {Out,C} = {A[0],A};
        ///////////////////////////////////////////////////
        3'b110:                                //RCL
            begin
                Cshift = C;
                {C,Out} = {A,Cshift};
            end
        ///////////////////////////////////////////////////
        3'b111:                                //RCR
            begin
                Cshift = C;
                {Out,C} = {Cshift,A};
            end
        ///////////////////////////////////////////////////
        default:
            begin
                Out = {Width{1'b0}};
                C = 1'b0;
                Z = 1'b0;
                N = 1'b0;
                P = 1'b0;
            end
        ///////////////////////////////////////////////////
    endcase

    Z = ~|Out;           //Zero Flag
    N = Out[Width-1];    //Negative Flag
    P = ~^Out;           //Parity Flag
end

endmodule

```

3.4 ALU Unit

```

module ALU
#(parameter Width=16)
(
    input    [Width-1:0] A,B,
    input    Cin,
    input    [4:0] F,          // F[4:3] -> block select, lower bits -> sub-op
    output reg [Width-1:0] Out,
    output reg [5:0] Status
);

// Registers for the Flags Used
reg    C,Z,N,V,P,Af,Cout;

// Arithmetic Wires
wire [Width-1:0] arith_out;
wire    arith_C,arith_Z,arith_N,arith_V,arith_P,arith_Af;

// Logic Wires
wire [Width-1:0] logic_out;
wire    logic_Z,logic_N,logic_P;

// Shift Wires
wire [Width-1:0] shift_out;
wire    shift_C,shift_Z,shift_N,shift_P;

// ===== Instantiations =====

// Arithmetic block (F[2:0] from ALU F)
Arithmetic#(.Width(Width)) U_ARITH (
    .A(A),
    .B(B),
    .Cin(Cin),
    .F(F[2:0]),
    .Out(arith_out),
    .Cout(),          // not used
    .Status(),        // not used
    .C(arith_C),
    .Z(arith_Z),
    .N(arith_N),
    .V(arith_V),
    .P(arith_P),
    .Af(arith_Af)
);

// Logic block (F[1:0] from ALU F)
Logic#(.Width(Width)) U_LOGIC (
    .A(A),
    .B(B),
    .F(F[2:0]),
    .Z(logic_Z),
    .N(logic_N),
    .P(logic_P),
    .Out(logic_out)
);

// Shift block (F[2:0] from ALU F)
Shift#(.Width(Width)) U_SHIFT (
    .A(A),
    .F(F[2:0]),
    .Out(shift_out),

```

ALU

```
.C(shift_C),
.Z(shift_Z),
.N(shift_N),
.P(shift_P)
);

// ===== ALU Mux Logic =====

always@(*) begin
    case (F[4:3])
        2'b00:
            begin
                // Arithmetic operations
                Out = arith_out;
                C = arith_C;
                Z = arith_Z;
                N = arith_N;
                V = arith_V;
                P = arith_P;
                Af = arith_Af;
            end
        //////////////////////////////////////
        2'b01:
            begin
                // Logic operations
                Out = logic_out;
                C = 1'b0;
                Z = logic_Z;
                N = logic_N;
                V = 1'b0;
                P = logic_P;
                Af = 1'b0;
            end
        //////////////////////////////////////
        2'b10:
            begin
                // Shift operations
                Out = shift_out;
                C = shift_C;
                Z = shift_Z;
                N = shift_N;
                V = 1'b0;
                P = shift_P;
                Af = 1'b0;
            end
        //////////////////////////////////////
        default:
            begin
                // Unused opcodes -----> everything zero
                Out = {Width{1'b0}};
                C = 1'b0;
                Z = 1'b0;
                N = 1'b0;
                V = 1'b0;
                P = 1'b0;
                Af = 1'b0;
            end
        //////////////////////////////////////
    endcase

    Cout = C;
    Status = {C,Z,N,V,P,Af};
end

endmodule
```

3.5 ALU_TB Unit

```
`timescale 1ns/1ps
```

```
module ALU_tb
#(parameter Width = 16);
  reg [Width-1:0] A,B;
  reg [4:0] F;
  reg Cin;
  wire[Width-1:0] Out;
  wire[5:0] Status;
```

```
// Instantiation of ALU Block
ALU#(Width) U_ALU (
  .A(A),
  .B(B),
  .Cin(Cin),
  .F(F),
  .Out(Out),
  .Status(Status)
);
```

```
initial
begin
  $display(" time | A | B | Cin | F | Out | Status");
  $display("-----");
  F = 5'b00000;
  repeat(100)
  begin
    A = $random;
    B = $random;
    Cin = 1; // $random;
    #1
    $display("%5t | %b %b %b %b | %b %b",
      $time, A, B, Cin, F, Out, Status);
    F = F + 1;
    #10;
  end
end
```

```
end
endmodule
```

4. Testbench

# time	A		B		Cin	F	Out		Status
#	-----	-----	-----	-----	-----	-----	-----	-----	-----
# 10000	0011010100100100	13604	0101111010000001	24193	1	00000	0000000000000000	0	000000
# 20000	1101011000001001	54793	0101011001100011	22115	1	00001	1101011000001010	54794	001000
# 30000	0111101100001101	31501	1001100110001101	39309	1	00010	0000000000000000	0	000000
# 40000	1000010001100101	33893	0101001000010010	21010	1	00011	1000010001100100	33892	001000
# 50000	1110001100000001	58113	1100110100001101	52493	1	00100	1011000000001110	45070	101010
# 60000	1111000101110110	61814	1100110100111101	52541	1	00101	1011111010110100	48820	101010
# 70000	0101011111101101	22509	1111011110001100	63372	1	00110	0110000001100001	24673	100000
# 80000	1110100111111001	59897	0010010011000110	9414	1	00111	1100010100110010	50482	001000
# 90000	1000010011000101	33989	1101001010101010	53930	1	01000	1000000010000000	32896	001010
# 100000	1111011111100101	63461	0111001001110111	29303	1	01001	1111011111110111	63479	001010
# 110000	1101011000010010	54802	1101101110001111	56207	1	01010	0000110110011101	3485	000010
# 120000	0110100111110010	27122	1001011011001110	38606	1	01011	1001011000001101	38413	001000
# 130000	0111101011101000	31464	0100111011000101	20165	1	01100	0000000000000000	0	000000
# 140000	0100100101011100	18780	0010100010111101	10429	1	01101	0000000000000000	0	000000
# 150000	0101100000101101	22573	0010011001100101	9829	1	01110	0000000000000000	0	000000
# 160000	0110001001100011	25187	1000011100001010	34570	1	01111	0000000000000000	0	000000
# 170000	0010001010000000	8832	0010000100100000	8480	1	10000	0100010100000000	17664	000000
# 180000	0100010110101010	17834	1100110010011101	52381	1	10001	0010001011010101	8917	000000
# 190000	0011111010010110	16022	1011100000010011	47123	1	10010	0111110100101100	32044	000000
# 200000	0011100000001101	14349	1101011001010011	54867	1	10011	0001110000000110	7174	100000
# 210000	1101110101101011	56683	0010101011010101	10965	1	10100	1011101011010111	47831	101000
# 220000	0100101000000010	18946	0011111010101110	16046	1	10101	0010010100000001	9473	000010
# 230000	1110100100011101	59677	0111001011001111	29391	1	10110	1101001000111010	53818	101010
# 240000	0100100100100011	18723	0110010100001010	25866	1	10111	1010010010010001	42129	101010
# 250000	0000101011001010	2762	0100110000111100	19516	1	11000	0000000000000000	0	000000
# 260000	1011110111110010	48626	0110000110001010	24970	1	11001	0000000000000000	0	000000
# 270000	1011001101000001	45889	0011010011011000	13528	1	11010	0000000000000000	0	000000
# 280000	1111001101111000	62328	0001001010001001	4745	1	11011	0000000000000000	0	000000
# 290000	0000110111101011	3563	0110010110110110	26038	1	11100	0000000000000000	0	000000
# 300000	1111100111000110	63942	0001001110101110	5038	1	11101	0000000000000000	0	000000
# 310000	0000001010111100	700	1101110100101010	56618	1	11110	0000000000000000	0	000000
# 320000	1001101000001011	39435	1011111001110001	48753	1	11111	0000000000000000	0	000000
# 330000	0100000110000101	16773	0101010101001111	21839	1	00000	0000000000000000	0	000000
# 340000	0110000000111011	24635	0011001100111010	13114	1	00001	0110000000111100	24636	000010
# 350000	0011001001111110	12926	0100101100010101	19221	1	00010	0000000000000000	0	000000

ALU

Transcript

Loading work.Shift

#	time	A		B		Cin	F		Out		Status
#											
#	10000	0011010100100100	13604	0101111010000001	24193	1	00000		0000000000000000	0	000000
#	20000	1101011000001001	54793	0101011001100011	22115	1	00001		1101011000001010	54794	001000
#	30000	0111101100001101	31501	1001100110001101	39309	1	00010		0000000000000000	0	000000
#	40000	1000010001100101	33893	0101001000010010	21010	1	00011		1000010001100100	33892	001000
#	50000	1110001100000001	58113	1100110100001101	52493	1	00100		1011000000001110	45070	101010
#	60000	1111000101110110	61814	11001101000111101	52541	1	00101		1011111010110100	48820	101010
#	70000	0101011111101101	22509	1111011110001100	63372	1	00110		0110000001100001	24673	100000
#	80000	1110100111111001	59897	0010010011000110	9414	1	00111		1100010100110010	50482	001000
#	90000	1000010011000101	33989	1101001010101010	53930	1	01000		1000000010000000	32896	001010
#	100000	1111011111100101	63461	0111001001110111	29303	1	01001		1111011111101111	63479	001010
#	110000	1101011000010010	54802	1101101110001111	56207	1	01010		0000110110011101	3485	000010
#	120000	0110100111110010	27122	1001011011001110	38606	1	01011		1001011000001101	38413	001000
#	130000	0111101011101000	31464	0100111011000101	20165	1	01100		0000000000000000	0	000000
#	140000	0100100101011100	18780	0010100010111101	10429	1	01101		0000000000000000	0	000000
#	150000	0101100000101101	22573	0010011001100101	9829	1	01110		0000000000000000	0	000000
#	160000	0110001001100011	25187	1000011100001010	34570	1	01111		0000000000000000	0	000000
#	170000	0010001010000000	8832	0010000100100000	8480	1	10000		0100010100000000	17664	000000
#	180000	0100010110101010	17834	1100110010011101	52381	1	10001		0010001011010101	8917	000000
#	190000	0011111010010110	16022	1011100000010011	47123	1	10010		0111110100101100	32044	000000
#	200000	0011100000001101	14349	1101011001010011	54867	1	10011		0001110000000110	7174	100000
#	210000	1101110101101011	56683	0010101011010101	10965	1	10100		1011101011010111	47831	101000
#	220000	0100101000000010	18946	0011111010101110	16046	1	10101		0010010100000001	9473	000010
#	230000	1110100100011101	59677	0111001011001111	29391	1	10110		1101001000111010	53818	101010
#	240000	0100100100100011	18723	0110010100001010	25866	1	10111		1010010010010001	42129	101010
#	250000	0000101011001010	2762	0100110000111100	19516	1	11000		0000000000000000	0	000000
#	260000	1011110111110010	48626	0110000110001010	24970	1	11001		0000000000000000	0	000000
#	270000	1011001101000001	45889	0011010011011000	13528	1	11010		0000000000000000	0	000000
#	280000	1111001101111000	62328	0001001010001001	4745	1	11011		0000000000000000	0	000000
#	290000	0000110111101011	3563	0110010110110110	26038	1	11100		0000000000000000	0	000000
#	300000	1111100111000110	63942	0001001110101110	5038	1	11101		0000000000000000	0	000000
#	310000	0000001010111100	700	1101110100101010	56618	1	11110		0000000000000000	0	000000
#	320000	1001101000001011	39435	1011111001110001	48753	1	11111		0000000000000000	0	000000
#	330000	0100000110000101	16773	0101010101001111	21839	1	00000		0000000000000000	0	000000
#	340000	0110000000111011	24635	0011001100111010	13114	1	00001		0110000000111100	24636	000010
#	350000	0011001001111110	12926	0100101100010101	19221	1	00010		0000000000000000	0	000000

5. Do file

```
vlib work  
vlog ALU.v ALU_TB.v  
vsim -voptargs=+acc work.ALU_tb  
add wave -r /*  
run -all
```

To run the simulation the Transcript we type :

Do run.txt