שפת תכנון חומרה
ורילוג - Verilog

Verilog – Dataflow and Behavioral description Practice & ALU

Dr. Avihai Aharon

Practice - Dataflow description

Dataflow description Questions:

1. Dataflow description of 2-to-1 line multiplexer
2. Dataflow description of 2-to-4 line decoder with enable input (E)

Dataflow description of 2-to-1 line multiplexer

```
module mux2x1_df (A,B,select,OUT);
    input A,B,select;
    output OUT;
    assign OUT = select ? A:B;
endmodule
```

Condition ? true-expression: false-expression;

Dataflow description of 2-to-1 line multiplexer

```
module mux2x1_df (A,B,select,OUT);
    input A,B,select;
    output OUT;
    assign OUT = (A & select) | (B & ~select);
endmodule
```

Dataflow description of 2-to-4 line decoder with enable input (E)

```
module decoder_df (A,B,E,D);
    input A,B,E;
    output [3:0] D;

        assign D[3] = (A & B & E);
        assign D[2] = (A & ~B & E);
        assign D[1] = (~A & B & E);
        assign D[0] = (~A & ~B & E);

endmodule
```

Or

```
assign D[3] =(A & B & E),
       D[2] =(A & ~B & E),
       D[1] =(~A & B & E),
       D[0] =(~A & ~B & E);
```

Dataflow description of 2-to-4 line decoder with enable input (E)

```
module decoder_df (A,B,E,D);
    input A,B,E;
    output [3:0] D;

        assign D[3] =~(~A | ~B | ~E);
        assign D[2] =~(~A | B | ~E);
        assign D[1] =~( A | ~B | ~E);
        assign D[0] =~( A | B | ~E);

endmodule
```

Or

```
assign D[3] =~(~A | ~B | ~E),
       D[2] =~(~A | B | ~E),
       D[1] =~( A | ~B | ~E),
       D[0] =~( A | B | ~E);
```

Practice - Behavioral description

Behavioral description Questions:

1. Behavioral description of 2-to-1 line multiplexer
2. Behavioral description of 4-to-1 line multiplexer

Behavioral description of 2-to-1 line multiplexer

```
module mux2x1_bh (A,B,select,OUT);
    input A,B,select;
    output OUT;
    reg OUT;
    always @(select or A or B)
        if (select==1) OUT=A;
        else OUT=B;
endmodule
```

The if statement can be written without the equality symbol:
If (select) OUT=A;

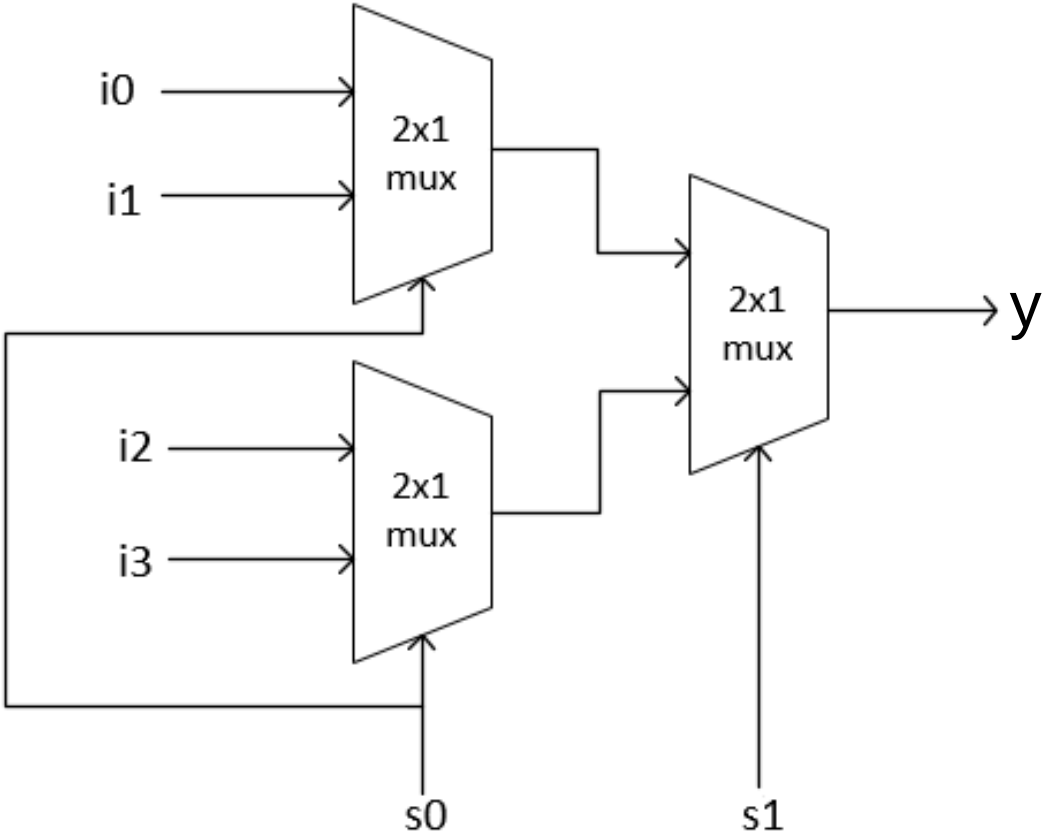
Behavioral description of 4-to-1 line multiplexer

```
module mux4x1_bh (i0,i1,i2,i3,select,y);
    input i0,i1,i2,i3;
    input [1:0]select;
    output y;
    reg y;
    always @( i0 or i1 or i2 or i3 or select)
        case (select)
            2’b00: y=i0;
            2’b01: y=i1;
            2’b10: y=i2;
            2’b11: y=i3;
        endcase
endmodule
```

Practice - Summary

Summary Questions:

1. Instantiation using 2-to-1 multiplexer module
2. Dataflow description
3. behavioral description



Practice - Instantiation

```
module mux4x1_INST (i0,i1,i2,i3,select,y);
    input i0,i1,i2,i3;
    input [1:0]select;
    output y;
    wire S0,S1;

    mux2x1_bh M0 (i1,i0,select[0],S0);
    mux2x1_bh M1 (i3,i2,select[0],S1);
    mux2x1_bh M2 (S1,S0,select[1],y);

endmodule
```

```
module mux2x1_bh (A,B,select,OUT);
    input A,B,select;
    output OUT;
    reg OUT;
    always @(select or A or B)
        if (select==1) OUT=A;
        else OUT=B;
endmodule
```

Practice - dataflow description

```
module mux4x1_DF (i0,i1,i2,i3,select,y);
    input i0,i1,i2,i3;
    input [1:0]select;
    output y;

    assign y = select[1] ? select[0] ? i3:i2 : select[0] ? i1:i0;

endmodule
```

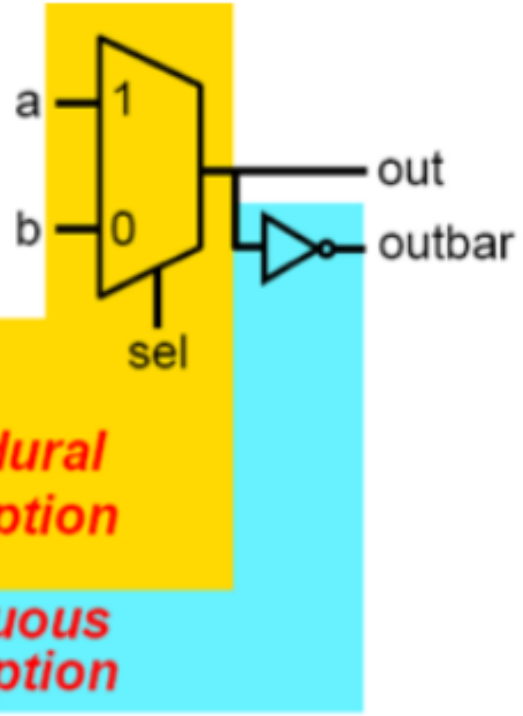
Practice - Behavioral description

```
module mux4x1_BH (i0,i1,i2,i3,select,y);
    input i0,i1,i2,i3;
    input [1:0]select;
    output y;
    reg y;
    always @( i0 or i1 or i2 or i3 or select)
      if (select[1])
        begin
          if (select[0])
              y = i3;
          else
              y = i2;
        end
      else
        begin
          if (select[0])
              y = i1;
          else
              y = i0;
        end
    endmodule
```

Mixing procedural & continuous assignments

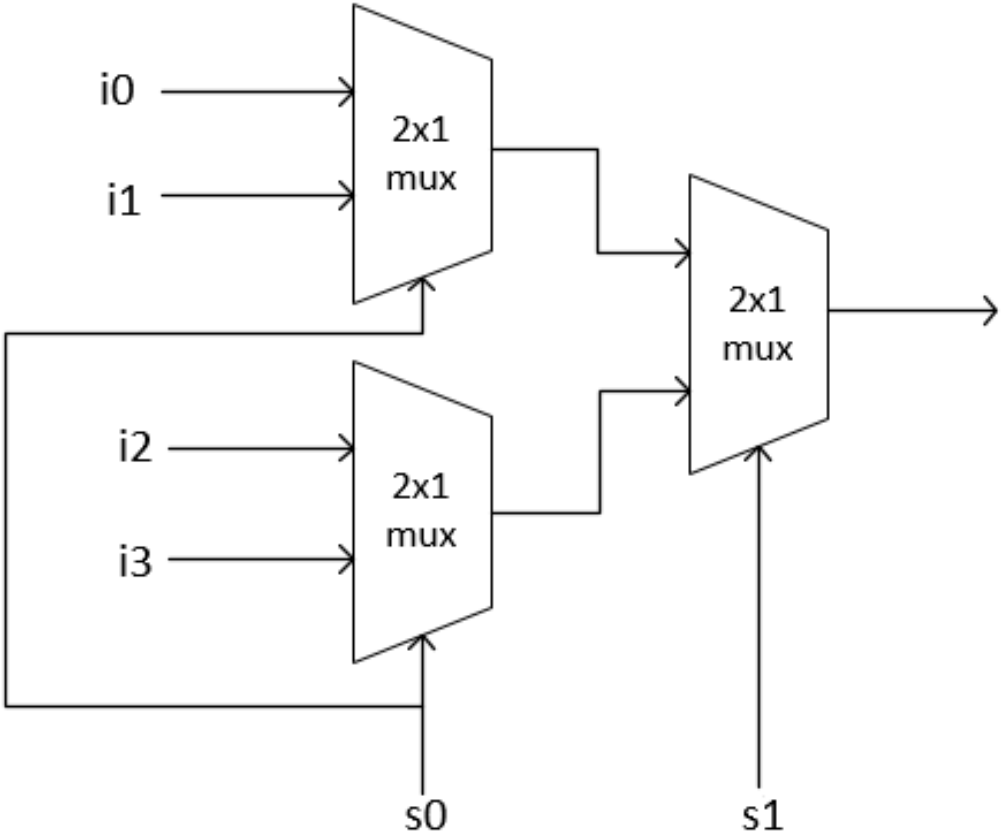
- Procedural and continuous assignments can co-exist within a module
- In procedural assignments, the value of variables declared as **reg** are changed only once when the procedural block is invoked by changes in the sensitivity list
- In continuous assignments, the right-hand expression is constantly evaluated and the left-side net is updated all the time

```
module mux_2_to_1(a, b, out,
                  outbar, sel);
  input a, b, sel;
  output out, outbar;
  reg out;

  always @ (a or b or sel)
  begin
    if (sel) out = a;
    else out = b;
  end

  assign outbar = ~out;

endmodule
```



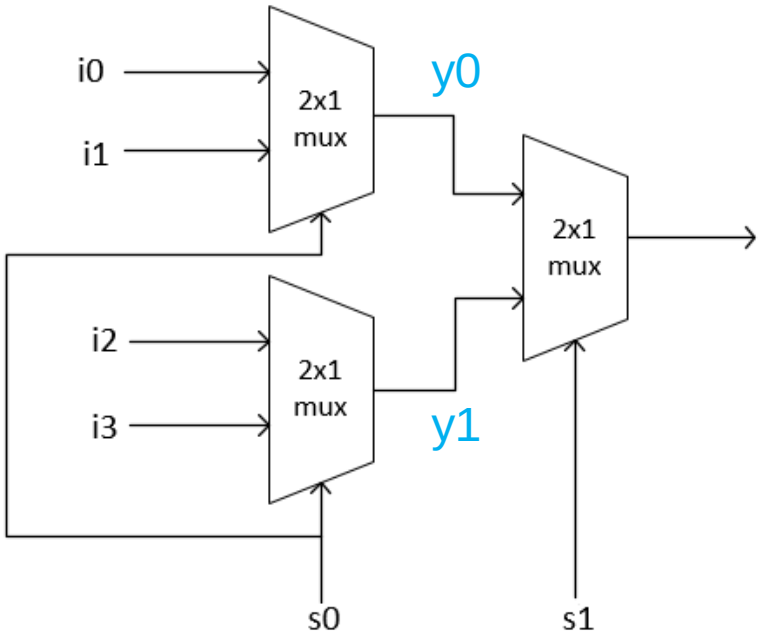
Practice – Mixing Assignments

1. Use mixing procedural & continuous assignments in order to describe that scheme in verilog:



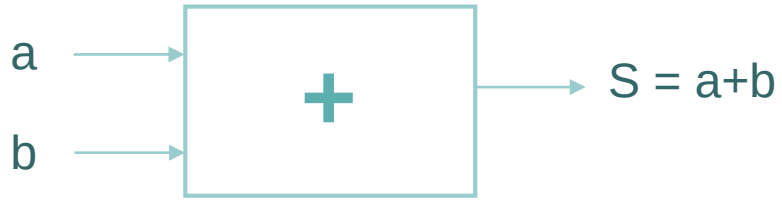
Practice - Mixing Assignments

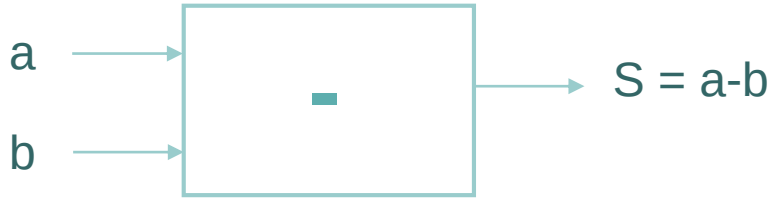
```
module mux4x1_MIX (i0,i1,i2,i3,select,y);
    input i0,i1,i2,i3;
    input [1:0]select;
    output y;
    reg y1,y0;
    always @( i2 or i3 or select )
      if (select[0])
            y1 = i3;
          else
            y1 = i2;
    always @( i0 or i1 or select )
      if (select[0])
            y0 = i1;
         else
            y0 = i0;
       end
    assign y = select[1] ? y1 : y0;

    endmodule
```

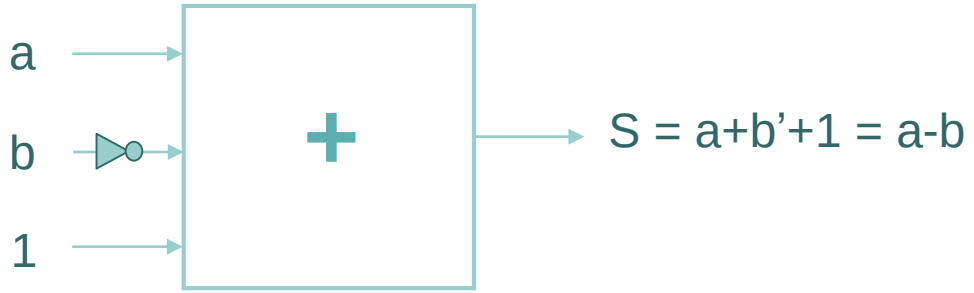


Adder, Subtractor and ALU

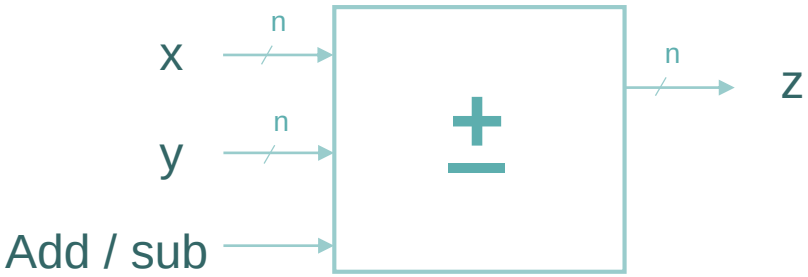
Adder & Subtractor

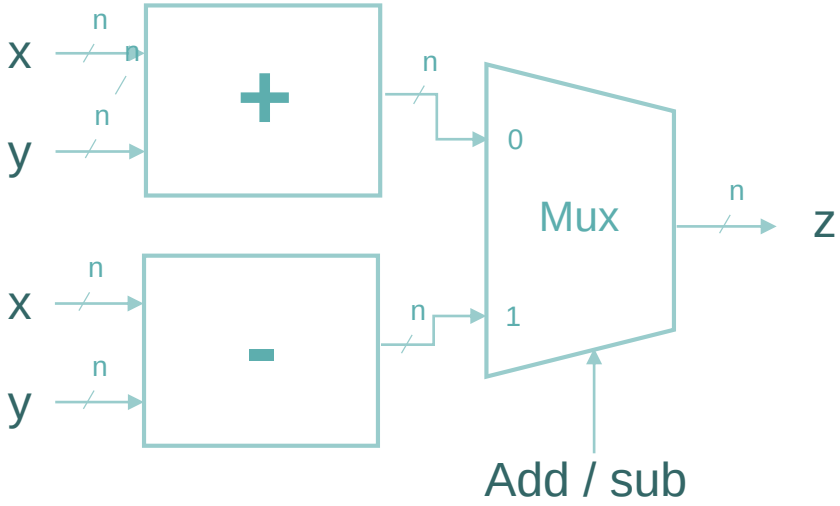


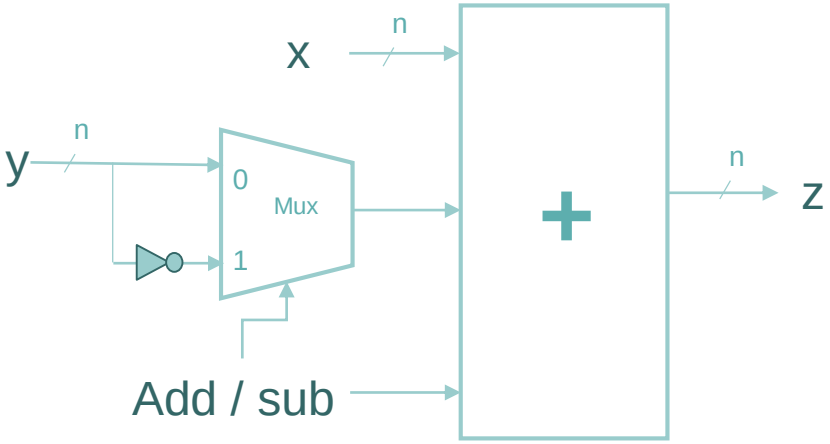




Adder & Subtractor







ALU – Arithmetic Logic unit

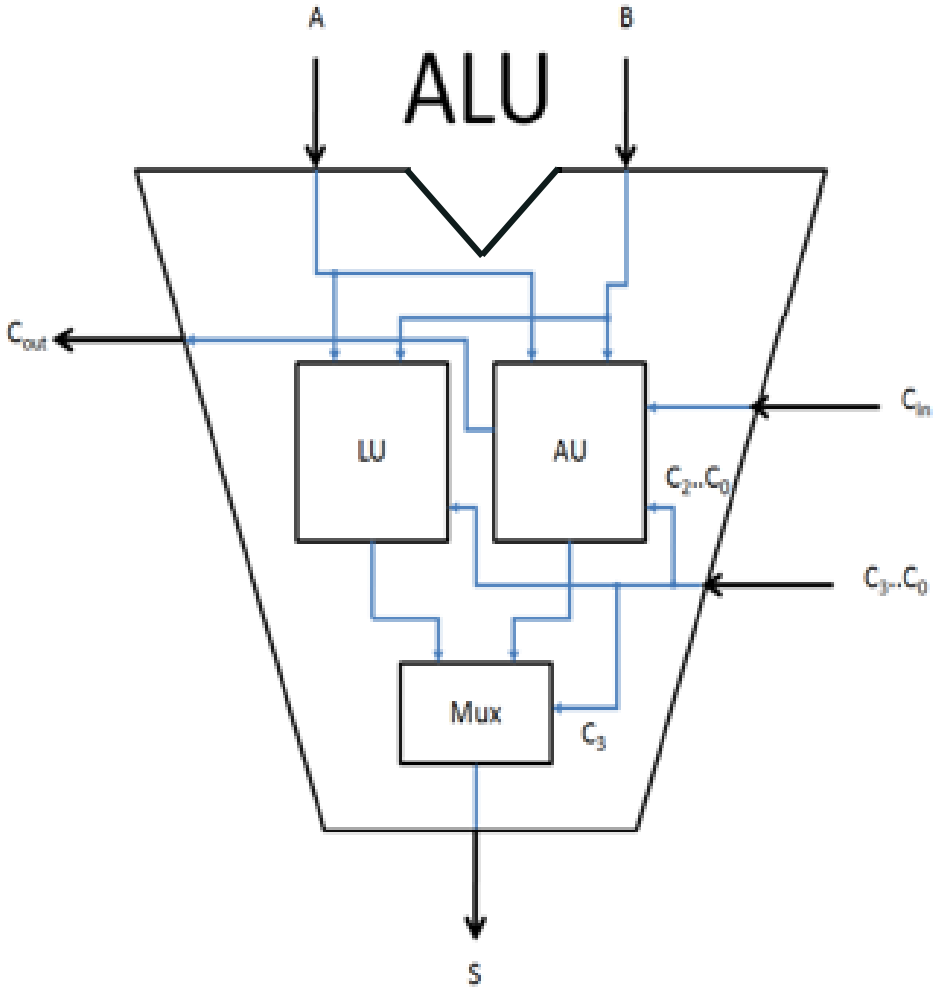


Fig. 2 .- Internal Architecture of the ALU.

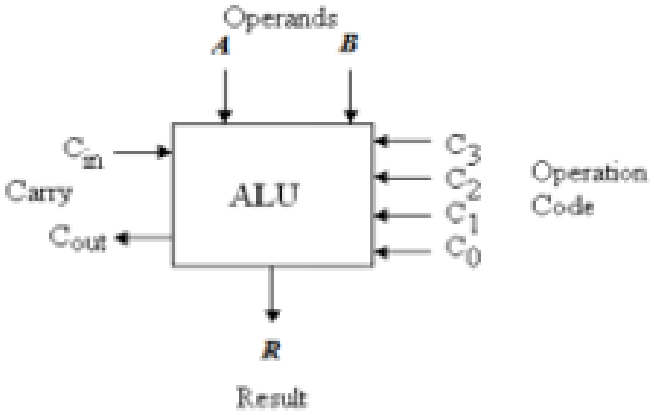
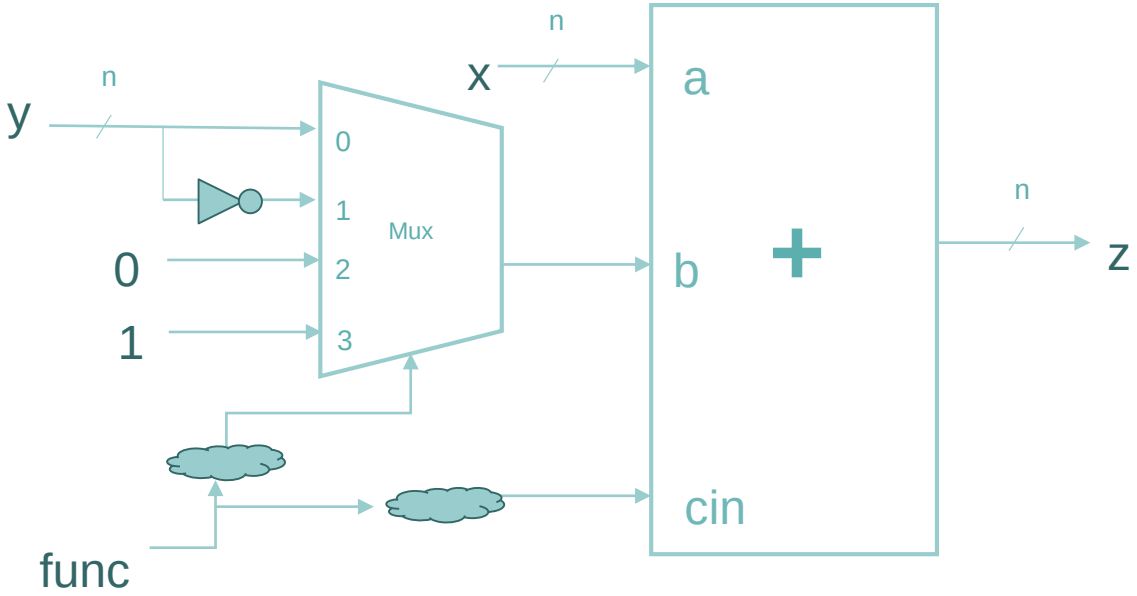


Fig. 1 .- ALU Block Diagram

AU – Arithmetic unit



func	operation	B	Cin	Z
0	x+y	y	0	x+y+0 = x+y
1	x-y	y'	1	x+y'+1 = x-y
2	x+1	0	1	x+0+1 = x+1
3	x-1	1…1	0	x+1…1+0 = x+(-1)
4	x+y+1	y	1	x+y+1
5	x+y'	y'	0	x+y'+0 = x+y'
6	x	0	0	x+0+0 = x