© 2020 Institute of Computer Science, University of the Philippines Los Banos

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

CMSC 132, First Semester AY 2020-2021

Computer Architecture

Joseph Anthony C. Hermocilla

BY NC SA

Welcome!

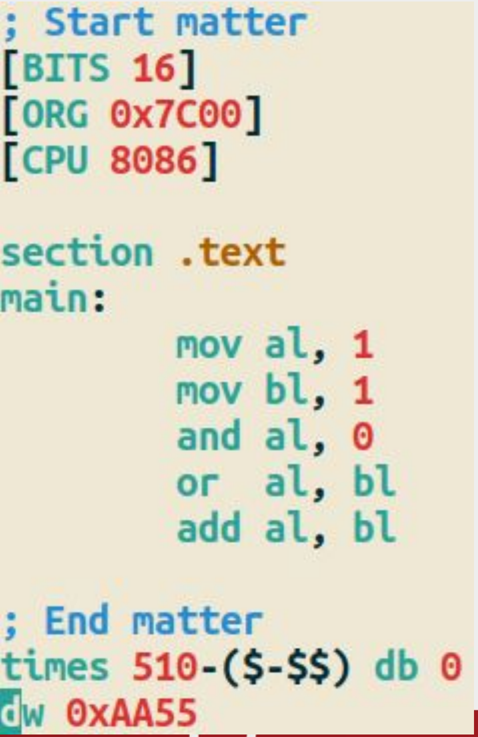
Combinational Elements and Sequential/State Elements

Resources

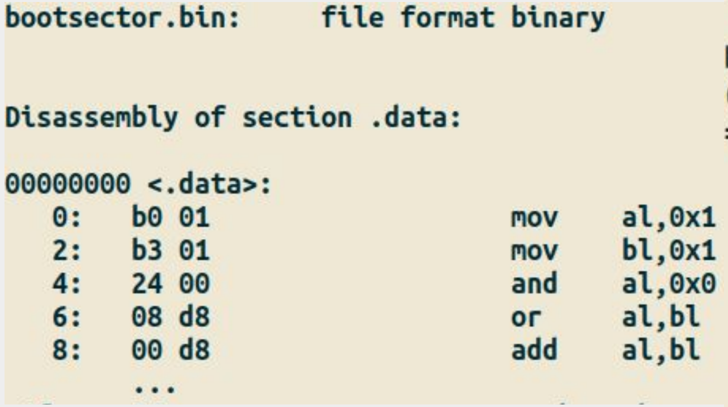
- Source code: https://bit.ly/2CJ7qVT
- http://esd.cs.ucr.edu/labs/tutorial/
- David A. Patterson and John L. Hennessy. 2017. Computer Organization and Design, ARM Edition: The Hardware/Software Interface (ARM ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

University of the Philippines
LOS BAÑOS

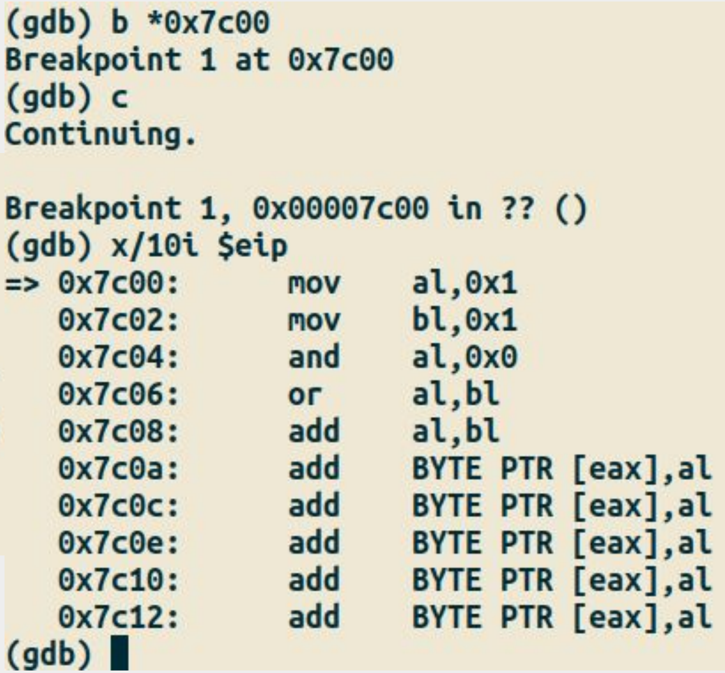
A disassembled x86(16-bit) boot sector code



```
; Start matter
[BITS 16]
[ORG 0x7C00]
[CPU 8086]

section .text
main:
        mov al, 1
        mov bl, 1
        and al, 0
        or  al, bl
        add al, bl

; End matter
times 510-($-$$) db 0
dw 0xAA55
```



```
bootsector.bin:     file format binary


Disassembly of section .data:

00000000 <.data>:
   0:   b0 01                   mov    al,0x1
   2:   b3 01                   mov    bl,0x1
   4:   24 00                   and    al,0x0
   6:   08 d8                   or     al,bl
   8:   00 d8                   add    al,bl
        ...
```



```
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.

Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/10i $eip
=> 0x7c00:      mov    al,0x1
   0x7c02:      mov    bl,0x1
   0x7c04:      and    al,0x0
   0x7c06:      or     al,bl
   0x7c08:      add    al,bl
   0x7c0a:      add    BYTE PTR [eax],al
   0x7c0c:      add    BYTE PTR [eax],al
   0x7c0e:      add    BYTE PTR [eax],al
   0x7c10:      add    BYTE PTR [eax],al
   0x7c12:      add    BYTE PTR [eax],al
(gdb) 
```

How do we implement a CPU that will execute the code?

Definition of Terms

- Datapath - The component of the processor that performs arithmetic operations
  - “brawn”
- Control - The component of the processor that commands the datapath, memory, and I/O devices according to the instructions of the program
  - “brain”

University of the Philippines
LOS BAÑOS

Logic elements in the datapath

- Combinational elements - Logic elements in the datapath that operate on data values
  - Outputs depend only on the current inputs
  - Given the same input, a combinational element always produces the same output.
  - Examples: ALU

Logic elements in the datapath (cont.)

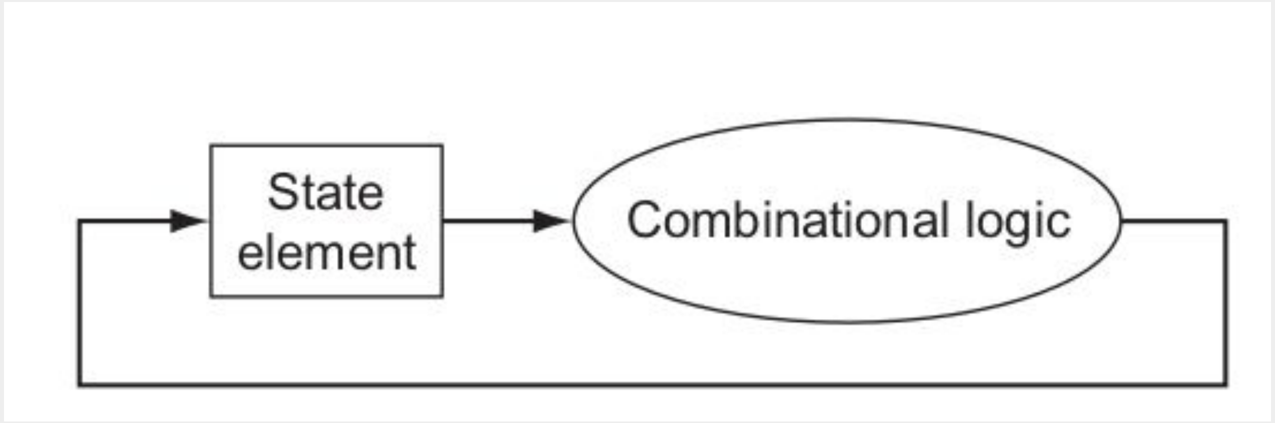
- State elements - contains *state*, has some *internal storage*
  - If we pull the power plug on the computer, we could restart it accurately by loading the state elements with values they contained before we pulled the plug
  - If we saved and restored the state elements it would be as if the computer had never lost power
  - Examples: instruction and data memories, registers
  - Has at least two inputs and one output: (data, clock) -> data, ex: D-type flip-flop
  - Clock determines when the state element should be written
  - Can be read at any time

Clocking Methodology

- Defines when signals can be read and when they can be written
- Makes hardware predictable
- Problems occur if the signal is written at the same time that it is read,
  - the value of the read could correspond to the old value, the newly written value, or a mix!
- Edge-triggered clocking - that any values stored in a sequential logic element are updated only on a clock edge, which is a quick transition from low to high or high to low

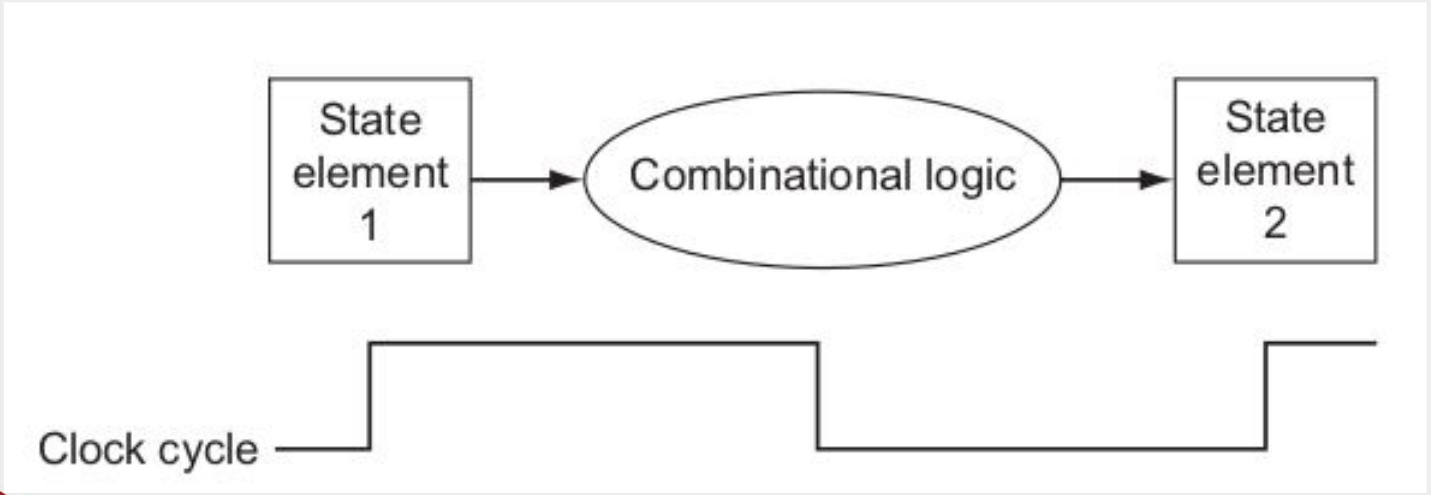
Example: Edge-triggered

- An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could lead to indeterminate data values

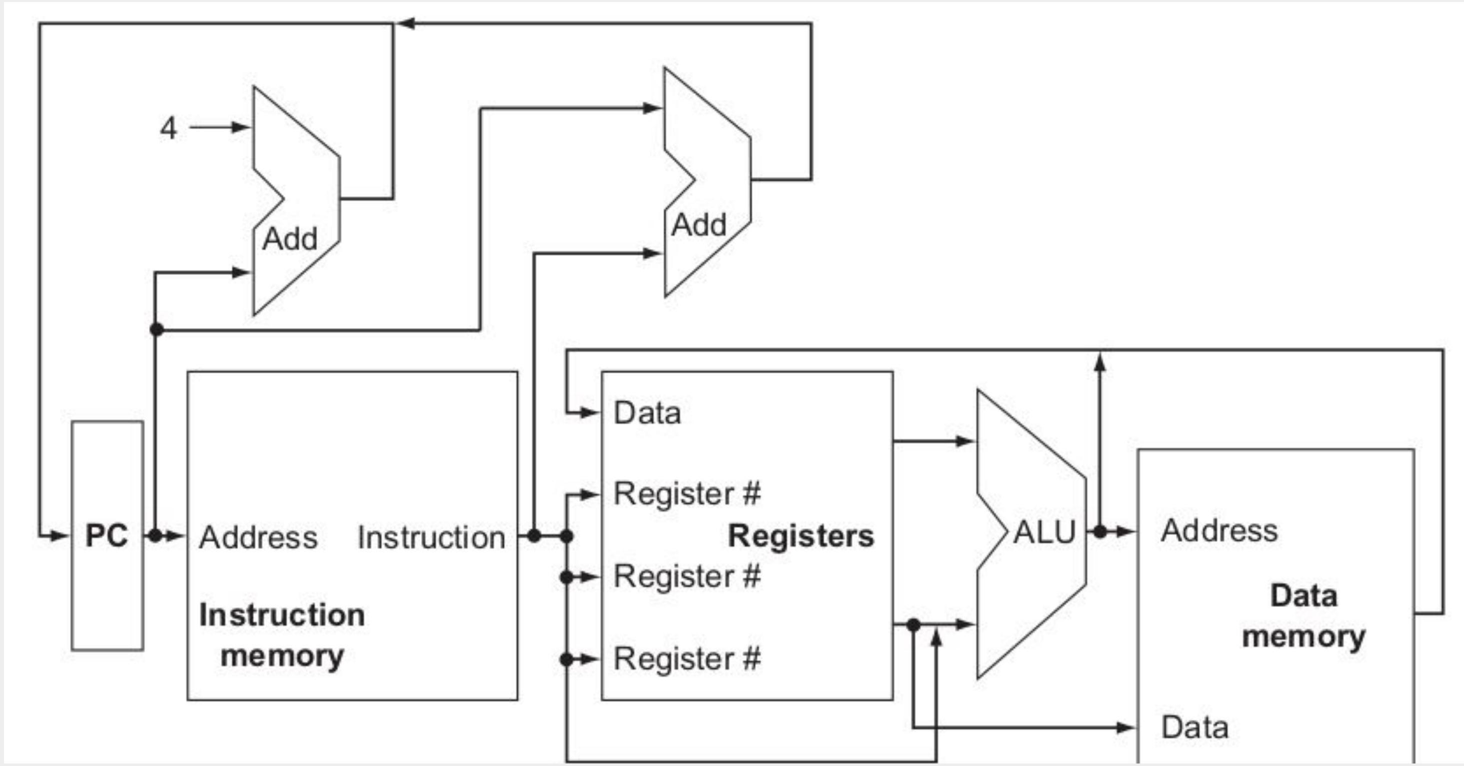


Example: Positive edge-triggered

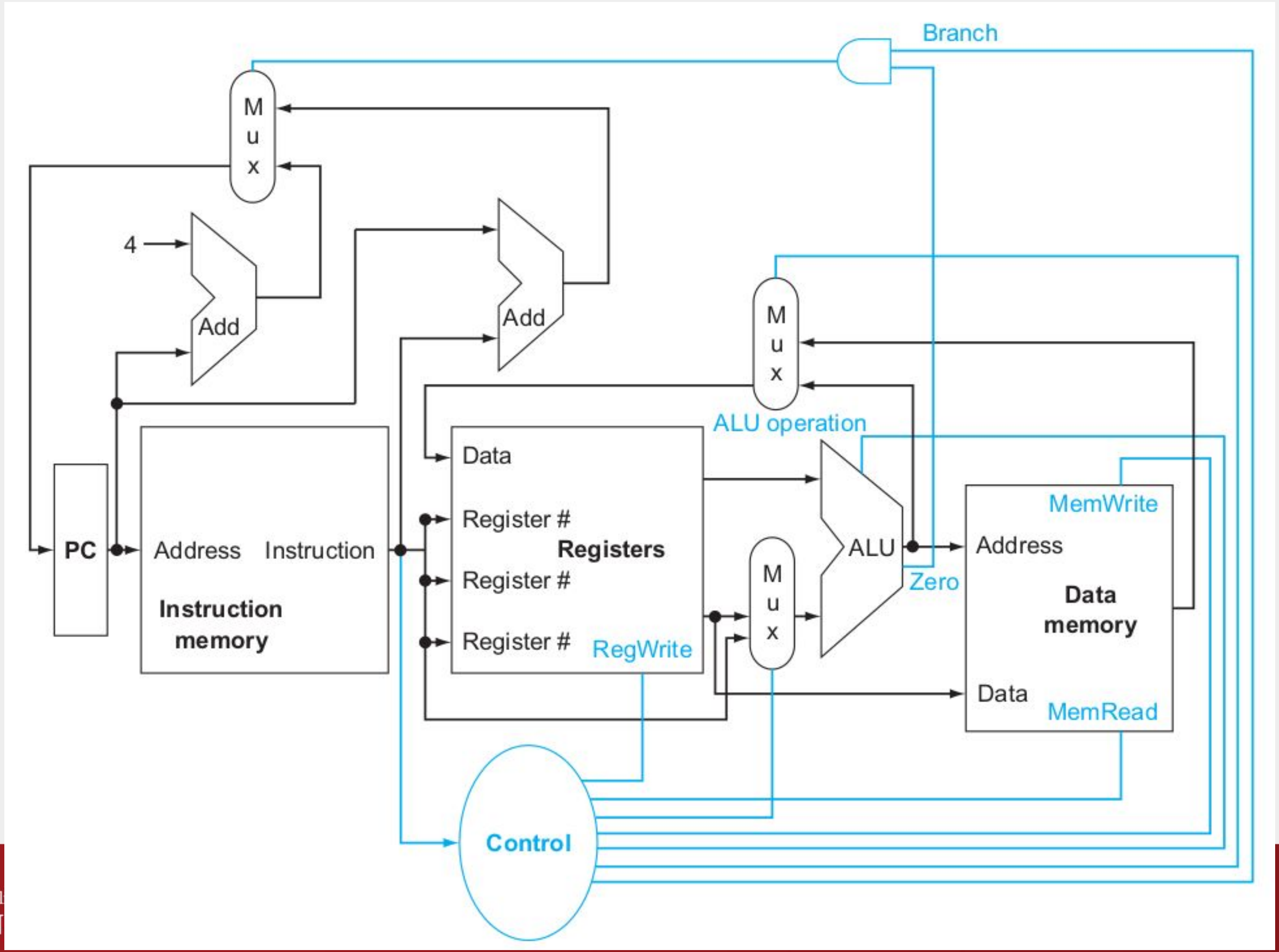
- Clock cycle length - time to propagate the signal from state element 1, through the combinational element, and state element 2



Single-Cycle Datapath Functional Components



Single-Cycle Datapath Functional Components with Control

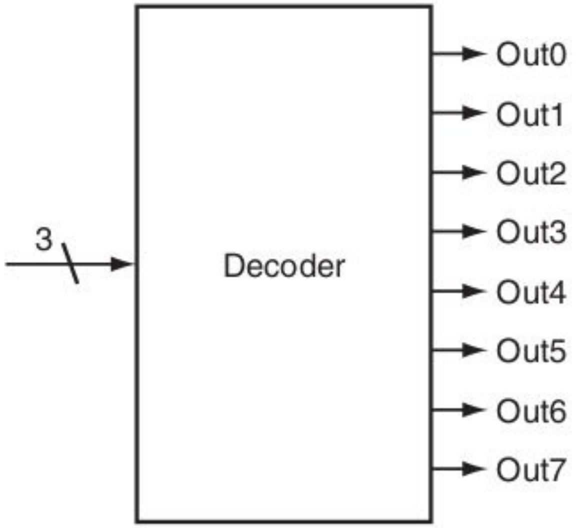


University of the Phil
LOS BAÑ

Combinational Building Blocks

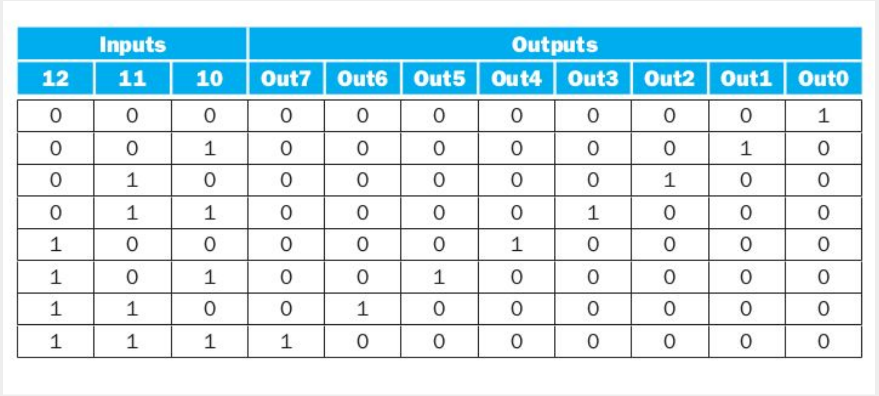
Decoders

A 3-bit decoder

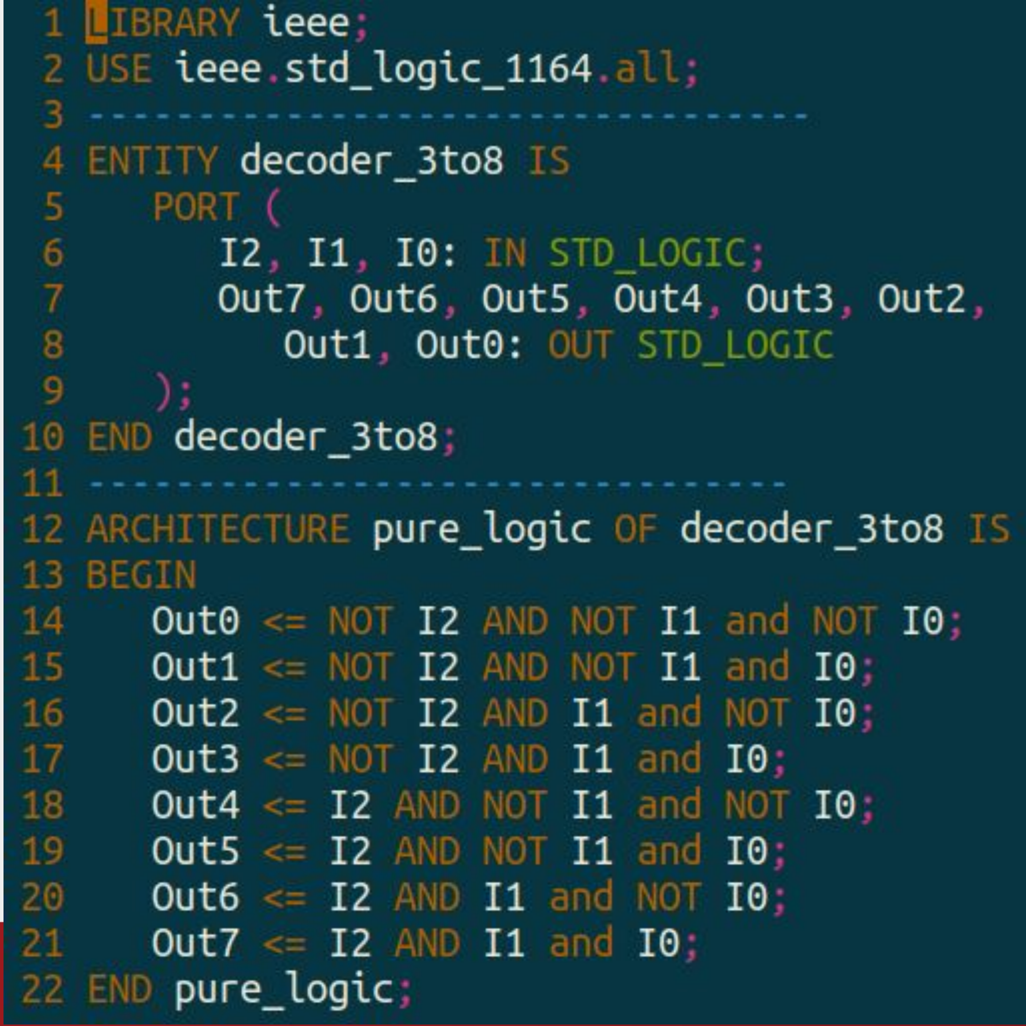


Inputs			Outputs							
I2	I1	I0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

VHDL Code



Inputs			Outputs							
I2	I1	I0	Out7	Out6	Out5	Out4	Out3	Out2	Out1	Out0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0



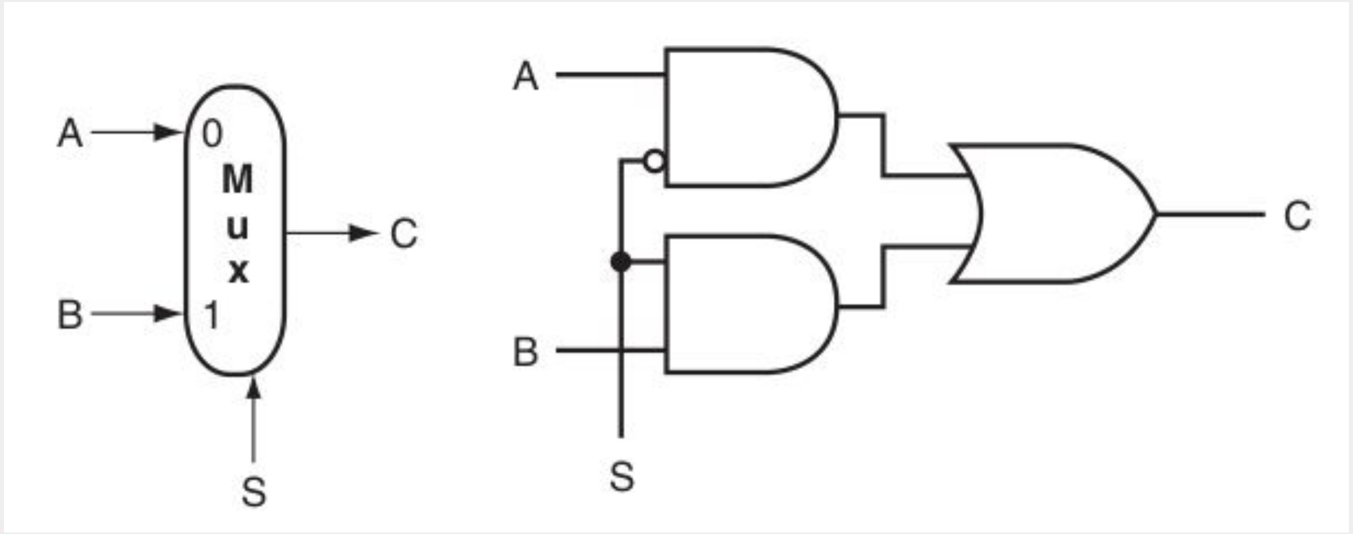
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------
ENTITY decoder_3to8 IS
   PORT (
      I2, I1, I0: IN STD_LOGIC;
      Out7, Out6, Out5, Out4, Out3, Out2,
         Out1, Out0: OUT STD_LOGIC
   );
END decoder_3to8;
---------------------------------------
ARCHITECTURE pure_logic OF decoder_3to8 IS
BEGIN
   Out0 <= NOT I2 AND NOT I1 and NOT I0;
   Out1 <= NOT I2 AND NOT I1 and I0;
   Out2 <= NOT I2 AND I1 and NOT I0;
   Out3 <= NOT I2 AND I1 and I0;
   Out4 <= I2 AND NOT I1 and NOT I0;
   Out5 <= I2 AND NOT I1 and I0;
   Out6 <= I2 AND I1 and NOT I0;
   Out7 <= I2 AND I1 and I0;
END pure_logic;
```

Testing

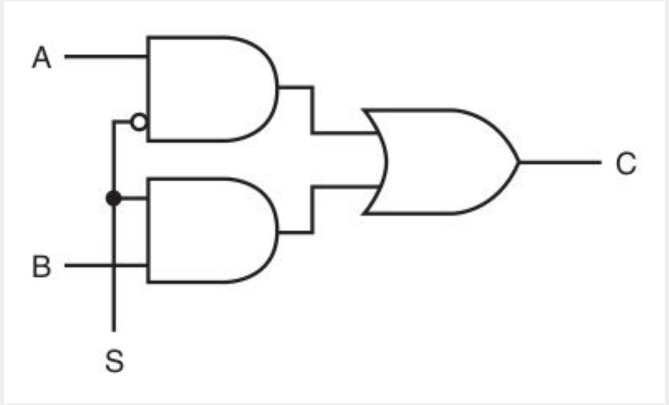
```
$ ghdl -a decoder_3to8.vhdl
$ ghdl -a decoder_3to8_tb.vhdl
$ ghdl -e decoder_3to8_tb
$ ghdl -r decoder_3to8_tb --vcd=decoder_tb.vcd
--stop-time=500ns
$ gtkwave decoder_tb.vcd
```

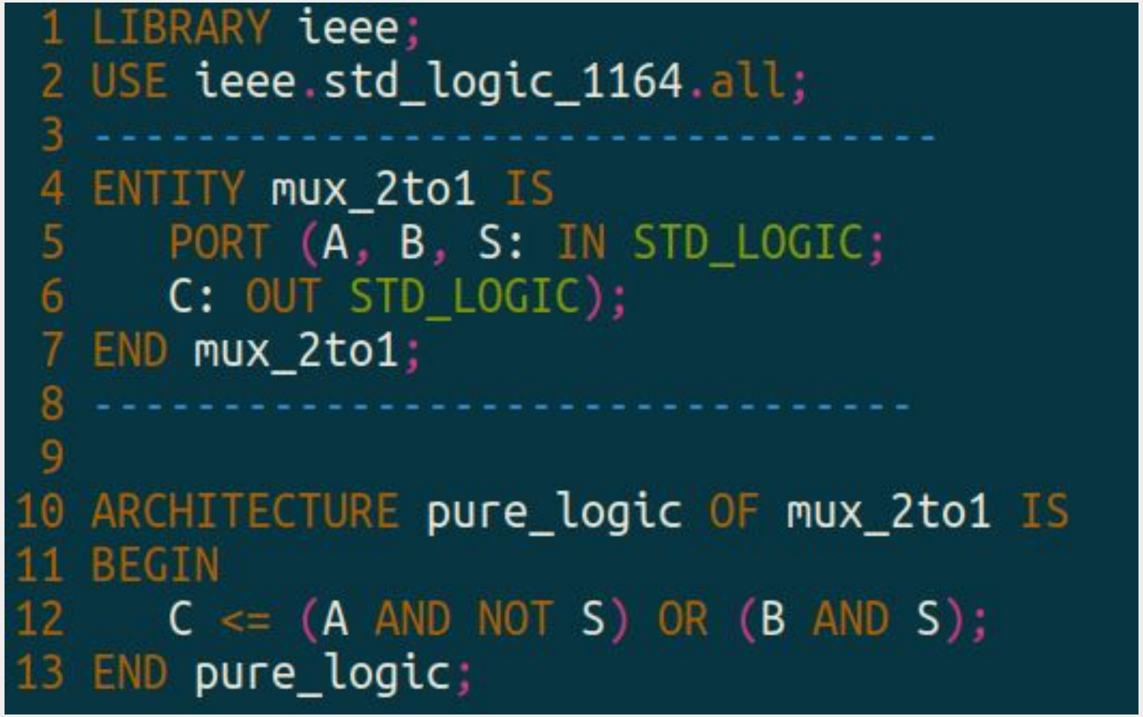
Multiplexers/Selectors (1 bit, 2:1 mux)

$$C = (\bar{A} \cdot S) + (\bar{B} \cdot S)$$



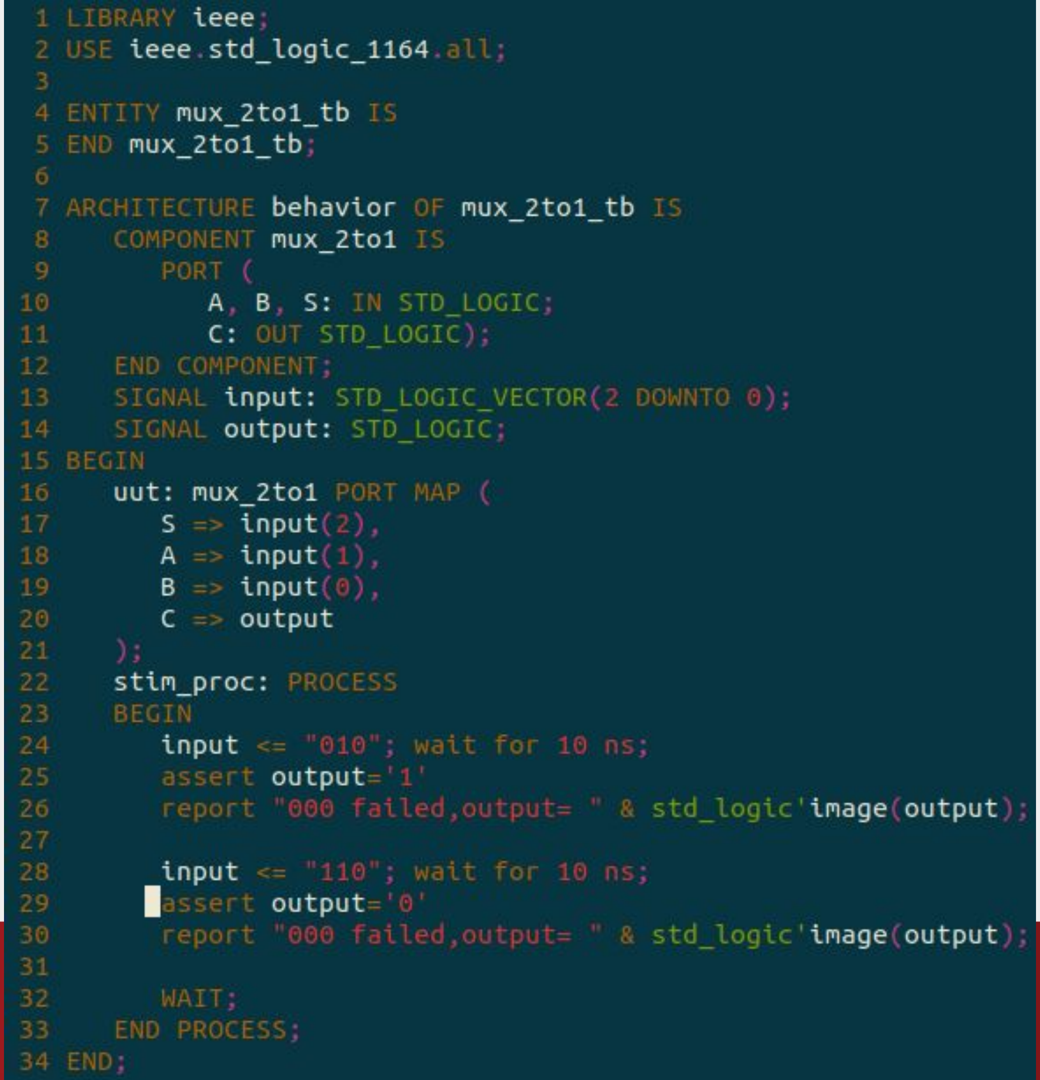
VHDL Code





```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY mux_2to1 IS
    PORT (A, B, S: IN STD_LOGIC;
    C: OUT STD_LOGIC);
END mux_2to1;
---------------------------------

ARCHITECTURE pure_logic OF mux_2to1 IS
BEGIN
    C <= (A AND NOT S) OR (B AND S);
END pure_logic;
```

Testbench
Signal-Port Mapping



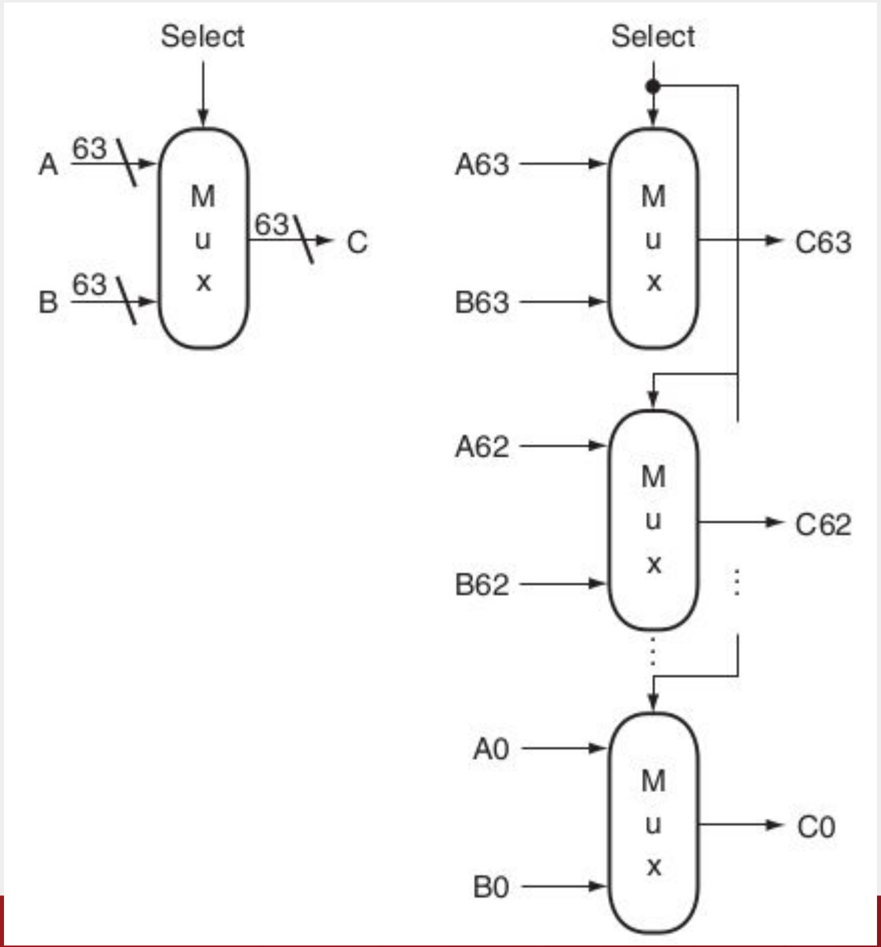
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux_2to1_tb IS
END mux_2to1_tb;

ARCHITECTURE behavior OF mux_2to1_tb IS
   COMPONENT mux_2to1 IS
      PORT (
         A, B, S: IN STD_LOGIC;
         C: OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL input: STD_LOGIC_VECTOR(2 DOWNTO 0);
   SIGNAL output: STD_LOGIC;
BEGIN
   uut: mux_2to1 PORT MAP (
      S => input(2),
      A => input(1),
      B => input(0),
      C => output
   );
   stim_proc: PROCESS
   BEGIN
      input <= "010"; wait for 10 ns;
      assert output='1'
      report "000 failed,output= " & std_logic'image(output);

      input <= "110"; wait for 10 ns;
      assert output='0'
      report "000 failed,output= " & std_logic'image(output);

      WAIT;
   END PROCESS;
END;
```

University of the Philippines
LOS BAÑOS

Testing

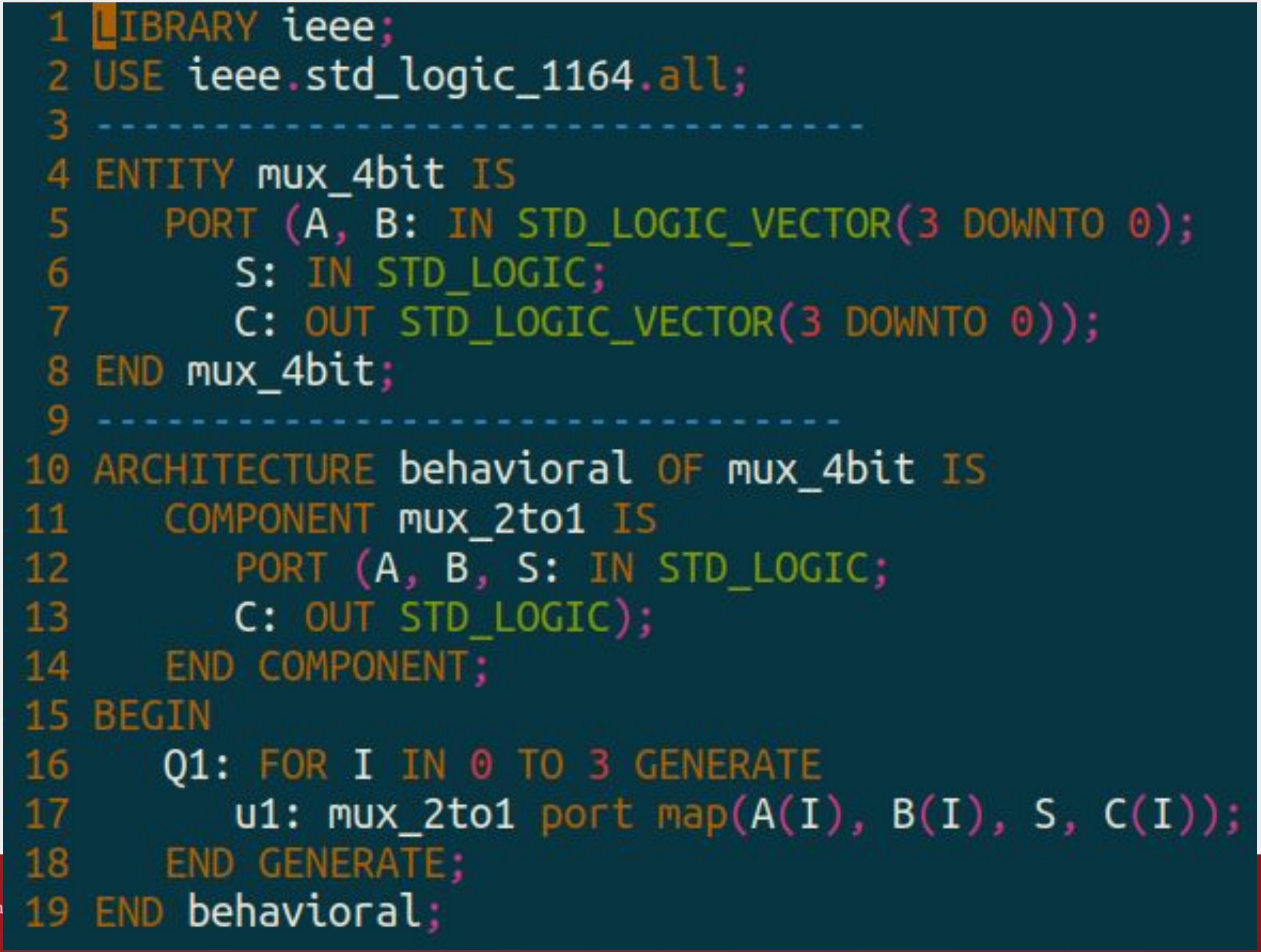
```
$ ghdl -a mux_2to1.vhdl
$ ghdl -a mux_2to1_tb.vhdl
$ ghdl -e mux_2to1_tb
$ ghdl -r mux_2to1_tb --vcd=mux_tb.vcd --stop-time=500ns
$ gtkwave mux_tb.vcd
```

An array of 1-bit multiplexers to select from two 64-bit inputs



University of the Philippines
LOS BAÑOS

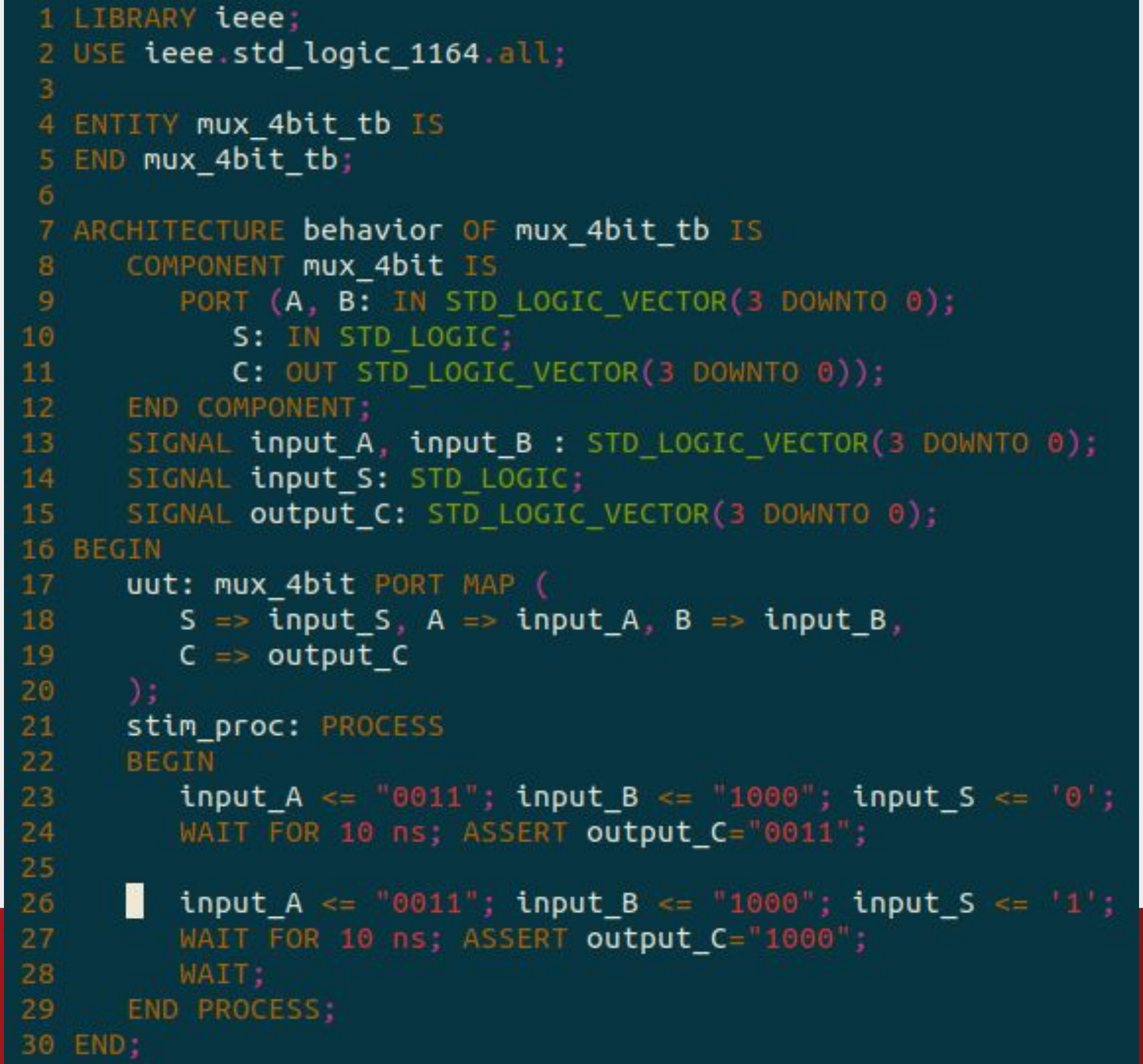
An array of 1-bit multiplexers to select from two 4-bit inputs in VHDL



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
ENTITY mux_4bit IS
    PORT (A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        S: IN STD_LOGIC;
        C: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END mux_4bit;
---------------------------------------
ARCHITECTURE behavioral OF mux_4bit IS
    COMPONENT mux_2to1 IS
        PORT (A, B, S: IN STD_LOGIC;
        C: OUT STD_LOGIC);
    END COMPONENT;
BEGIN
    Q1: FOR I IN 0 TO 3 GENERATE
        u1: mux_2to1 port map(A(I), B(I), S, C(I));
    END GENERATE;
END behavioral;
```

University of the Philippin
LOS BAÑO

Testbench

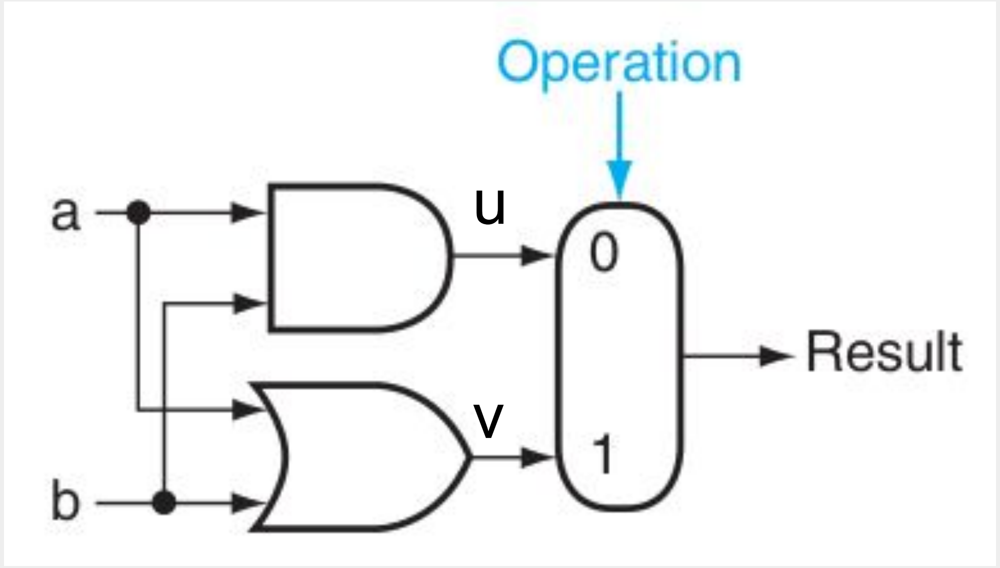


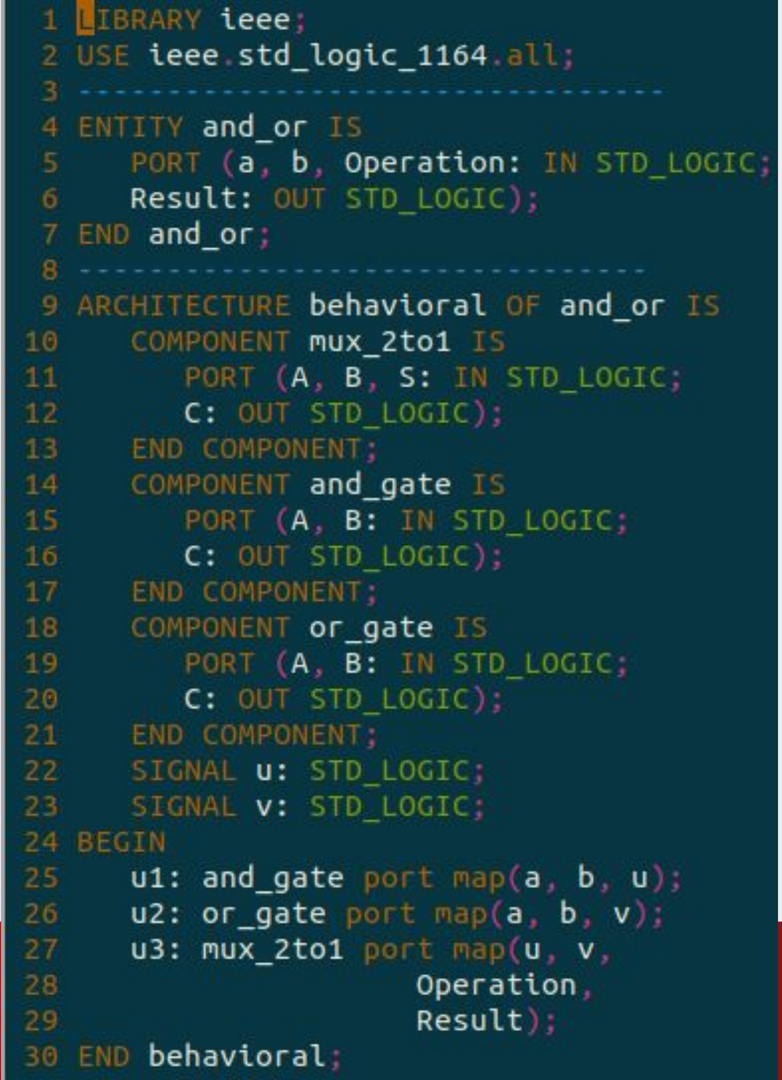
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux_4bit_tb IS
END mux_4bit_tb;

ARCHITECTURE behavior OF mux_4bit_tb IS
   COMPONENT mux_4bit IS
      PORT (A, B: IN STD_LOGIC_VECTOR(3 DOWNTO 0);
         S: IN STD_LOGIC;
         C: OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
   END COMPONENT;
   SIGNAL input_A, input_B : STD_LOGIC_VECTOR(3 DOWNTO 0);
   SIGNAL input_S: STD_LOGIC;
   SIGNAL output_C: STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
   uut: mux_4bit PORT MAP (
      S => input_S, A => input_A, B => input_B,
      C => output_C
   );
   stim_proc: PROCESS
   BEGIN
      input_A <= "0011"; input_B <= "1000"; input_S <= '0';
      WAIT FOR 10 ns; ASSERT output_C="0011";

      input_A <= "0011"; input_B <= "1000"; input_S <= '1';
      WAIT FOR 10 ns; ASSERT output_C="1000";
      WAIT;
   END PROCESS;
END;
```

University of the Philippines
LOS BAÑOS

Constructing a 1-bit ALU that performs AND, OR, ADD

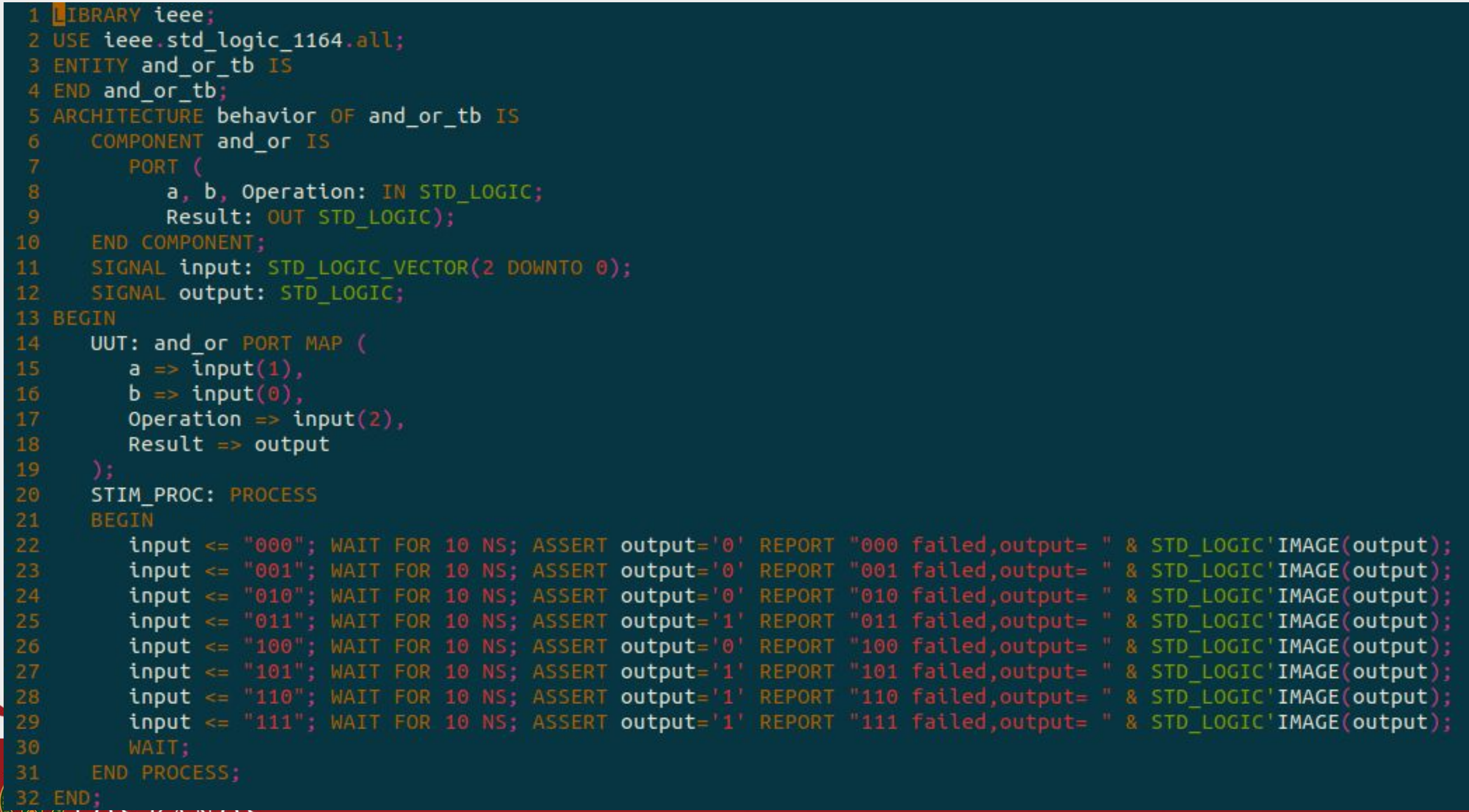
1-bit logical unit for AND and OR





```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY and_or IS
   PORT (a, b, Operation: IN STD_LOGIC;
   Result: OUT STD_LOGIC);
END and_or;
---------------------------------
ARCHITECTURE behavioral OF and_or IS
   COMPONENT mux_2to1 IS
      PORT (A, B, S: IN STD_LOGIC;
      C: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT and_gate IS
      PORT (A, B: IN STD_LOGIC;
      C: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT or_gate IS
      PORT (A, B: IN STD_LOGIC;
      C: OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL u: STD_LOGIC;
   SIGNAL v: STD_LOGIC;
BEGIN
   u1: and_gate port map(a, b, u);
   u2: or_gate port map(a, b, v);
   u3: mux_2to1 port map(u, v,
                         Operation,
                         Result);
END behavioral;
```

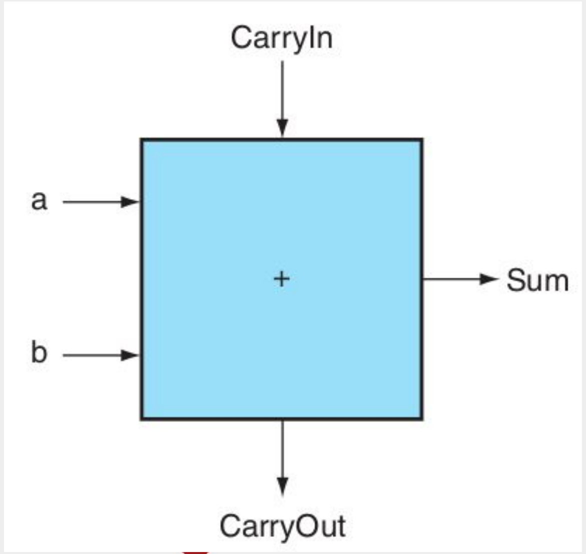
Testbench

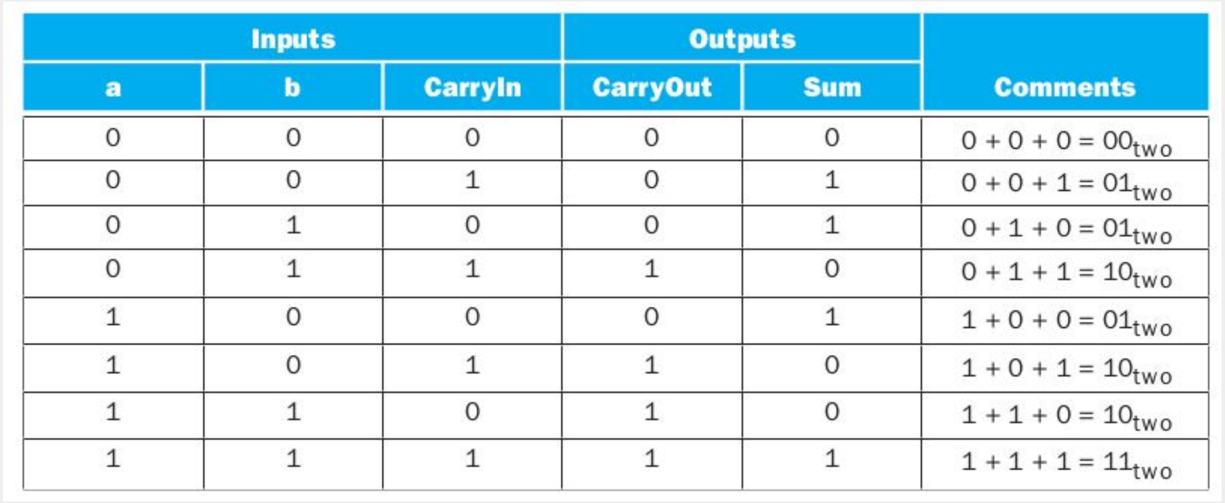


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY and_or_tb IS
END and_or_tb;
ARCHITECTURE behavior OF and_or_tb IS
    COMPONENT and_or IS
        PORT (
            a, b, Operation: IN STD_LOGIC;
            Result: OUT STD_LOGIC);
    END COMPONENT;
    SIGNAL input: STD_LOGIC_VECTOR(2 DOWNTO 0);
    SIGNAL output: STD_LOGIC;
BEGIN
    UUT: and_or PORT MAP (
        a => input(1),
        b => input(0),
        Operation => input(2),
        Result => output
    );
    STIM_PROC: PROCESS
    BEGIN
        input <= "000"; WAIT FOR 10 NS; ASSERT output='0' REPORT "000 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "001"; WAIT FOR 10 NS; ASSERT output='0' REPORT "001 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "010"; WAIT FOR 10 NS; ASSERT output='0' REPORT "010 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "011"; WAIT FOR 10 NS; ASSERT output='1' REPORT "011 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "100"; WAIT FOR 10 NS; ASSERT output='0' REPORT "100 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "101"; WAIT FOR 10 NS; ASSERT output='1' REPORT "101 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "110"; WAIT FOR 10 NS; ASSERT output='1' REPORT "110 failed,output= " & STD_LOGIC'IMAGE(output);
        input <= "111"; WAIT FOR 10 NS; ASSERT output='1' REPORT "111 failed,output= " & STD_LOGIC'IMAGE(output);
        WAIT;
    END PROCESS;
END;
```

LOS BANOS

1-bit adder: truth table



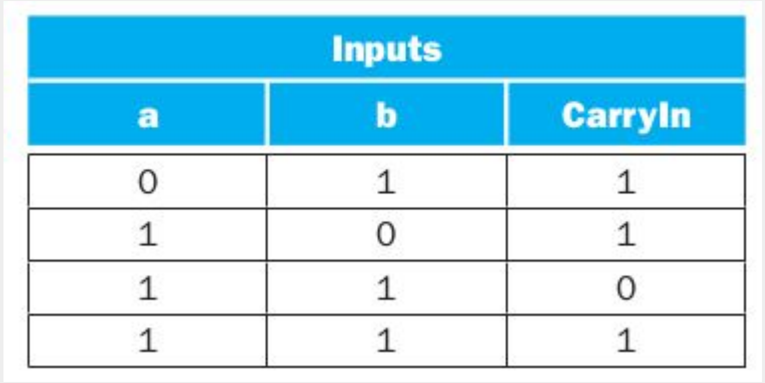


Inputs			Outputs		Comments
a	b	CarryIn	CarryOut	Sum	
0	0	0	0	0	$0 + 0 + 0 = 00_{two}$
0	0	1	0	1	$0 + 0 + 1 = 01_{two}$
0	1	0	0	1	$0 + 1 + 0 = 01_{two}$
0	1	1	1	0	$0 + 1 + 1 = 10_{two}$
1	0	0	0	1	$1 + 0 + 0 = 01_{two}$
1	0	1	1	0	$1 + 0 + 1 = 10_{two}$
1	1	0	1	0	$1 + 1 + 0 = 10_{two}$
1	1	1	1	1	$1 + 1 + 1 = 11_{two}$

1-bit adder: CarryOut (cont..)

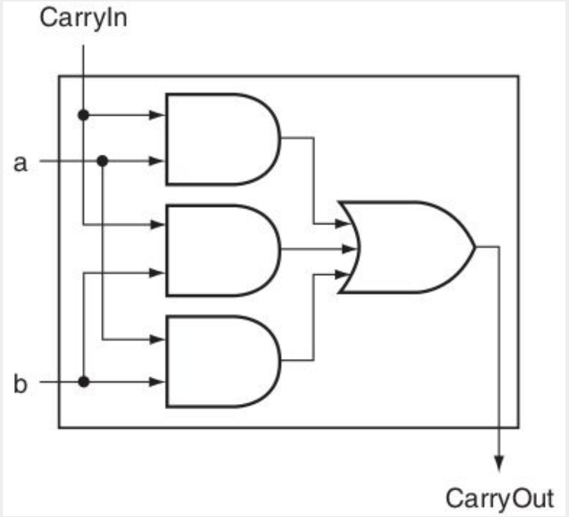
Link: YT video to derive the equations(CMSC 130 topic)

$$\text{CarryOut} = (b \cdot \text{CarryIn}) + (a \cdot \text{CarryIn}) + (a \cdot b)$$



Inputs		
a	**b**	**CarryIn**
0	1	1
1	0	1
1	1	0
1	1	1

Inputs when CarryOut is 1



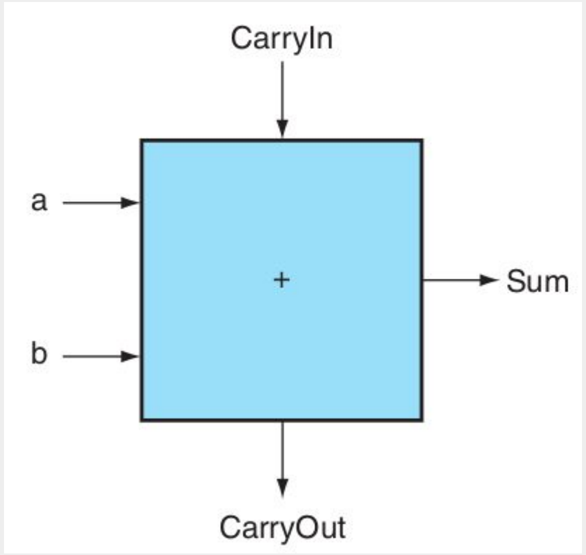
1-bit adder: Sum (cont..)

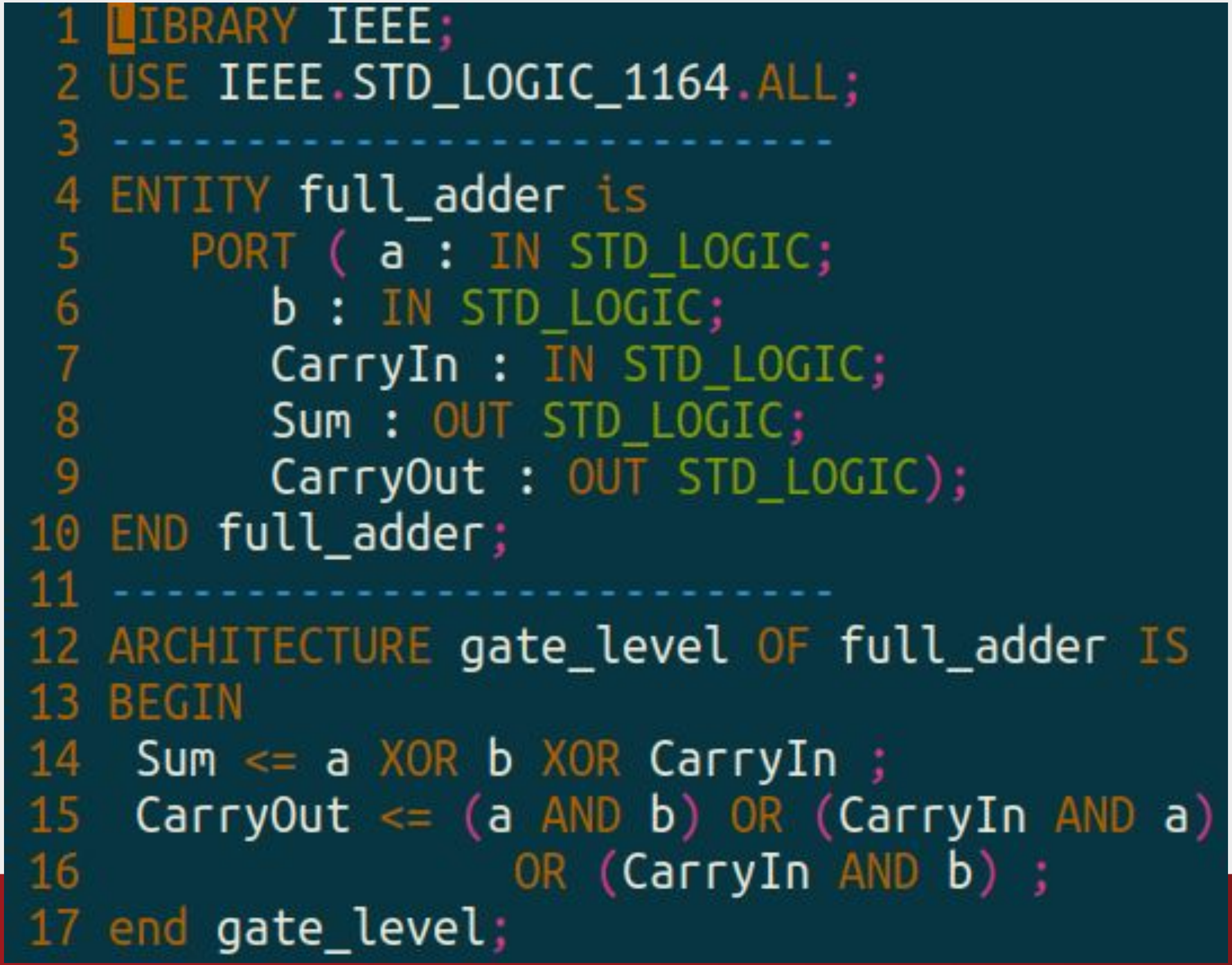
$$\text{Sum} = (a \cdot \overline{b} \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\overline{a} \cdot \overline{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$

```
Sum = a xor b xor CarryIn
```

University of the Philippines
LOS BAÑOS

Final Adder

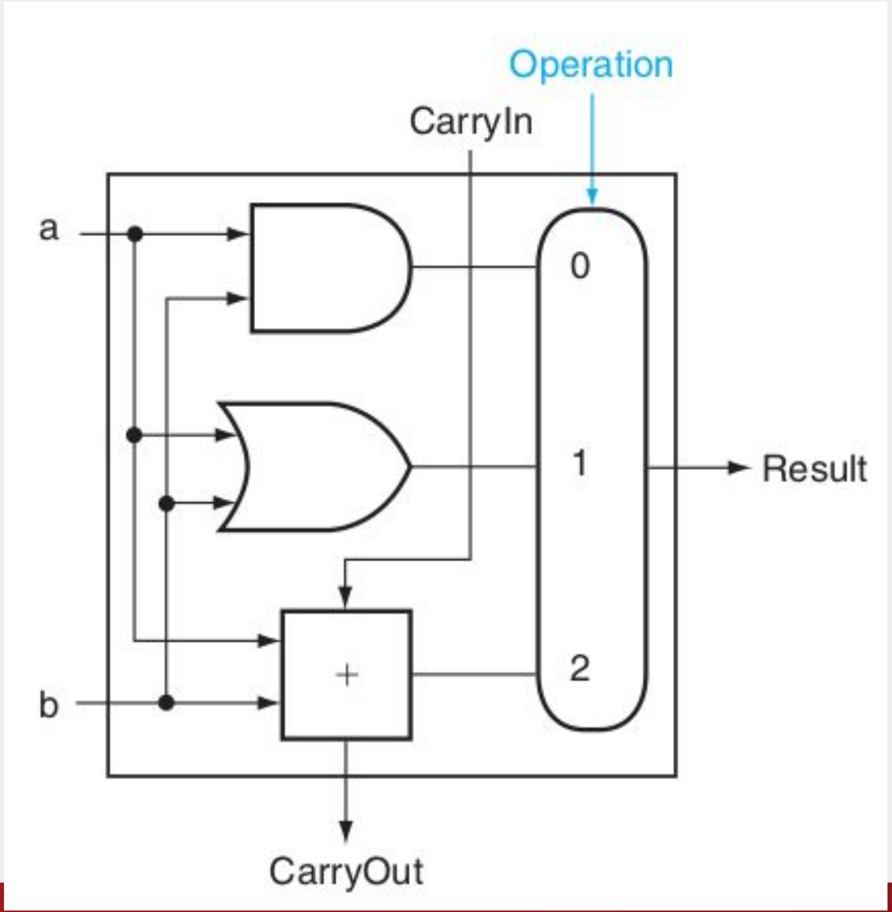




```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
---------------------------------
ENTITY full_adder is
   PORT ( a : IN STD_LOGIC;
         b : IN STD_LOGIC;
         CarryIn : IN STD_LOGIC;
         Sum : OUT STD_LOGIC;
         CarryOut : OUT STD_LOGIC);
END full_adder;
---------------------------------
ARCHITECTURE gate_level OF full_adder IS
BEGIN
 Sum <= a XOR b XOR CarryIn ;
 CarryOut <= (a AND b) OR (CarryIn AND a)
                  OR (CarryIn AND b) ;
end gate_level;
```

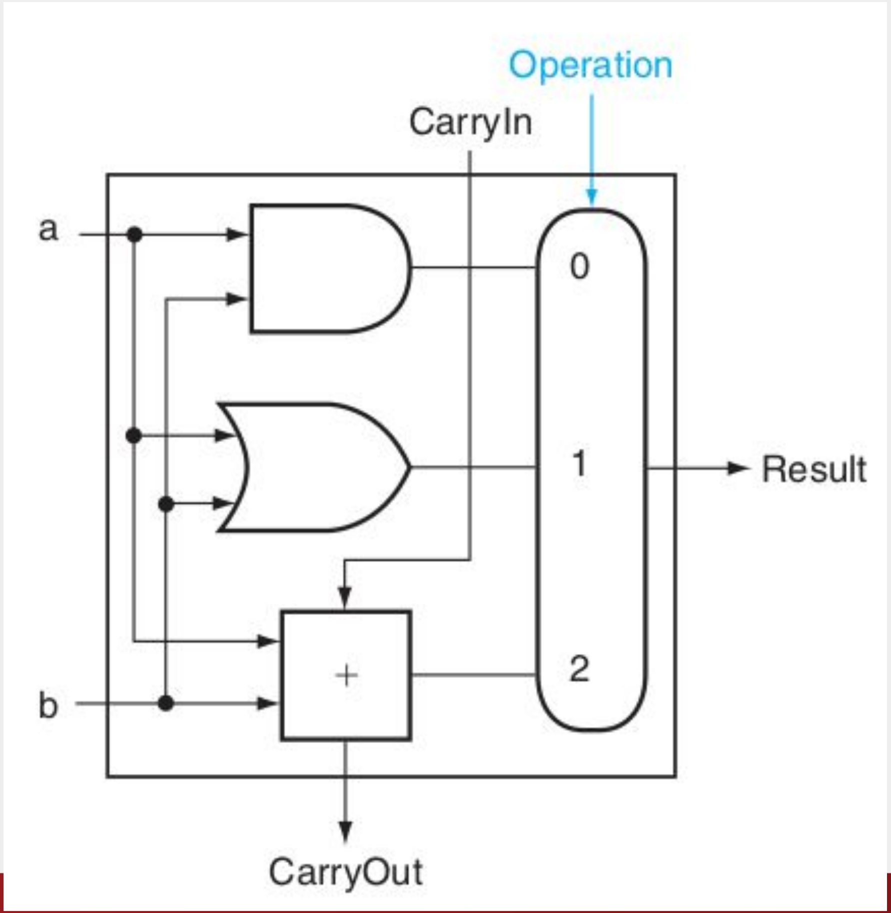
University of the Philippines
LOS BAÑOS

Final design for 1-bit ALU

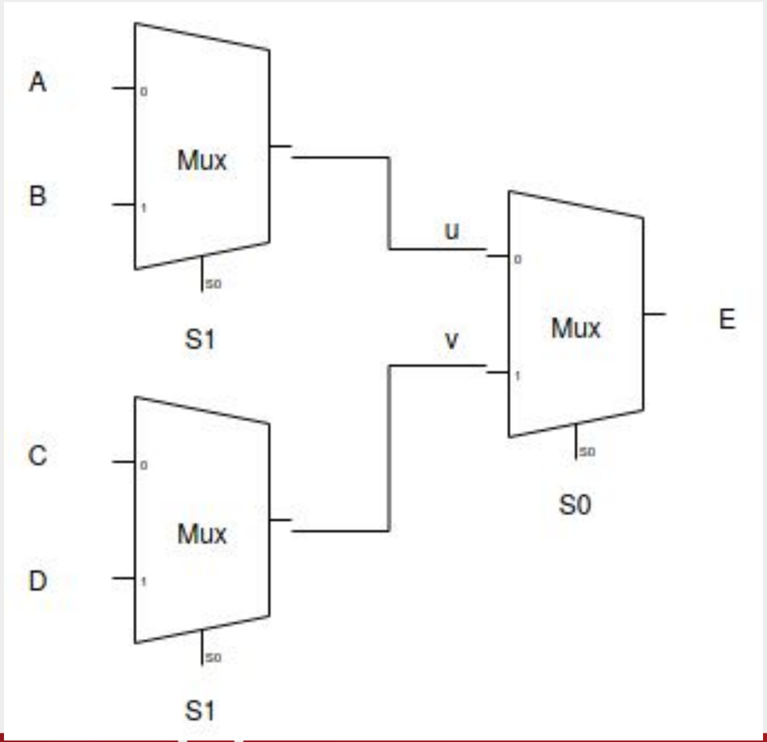


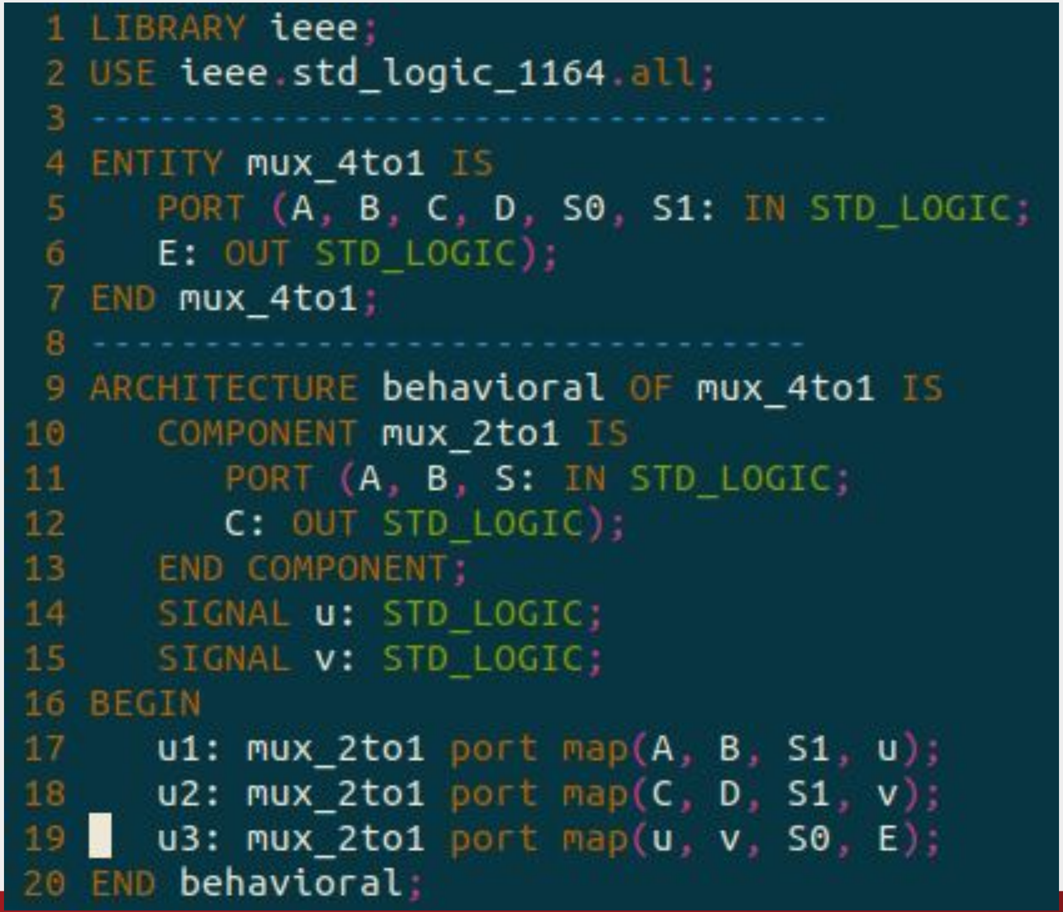
Final design for 1-bit ALU

- Three supported operations, 2:1 mux is not enough!
  - 00 - AND
  - 01 - OR
  - 10 - ADD
- 4:1 mux is needed

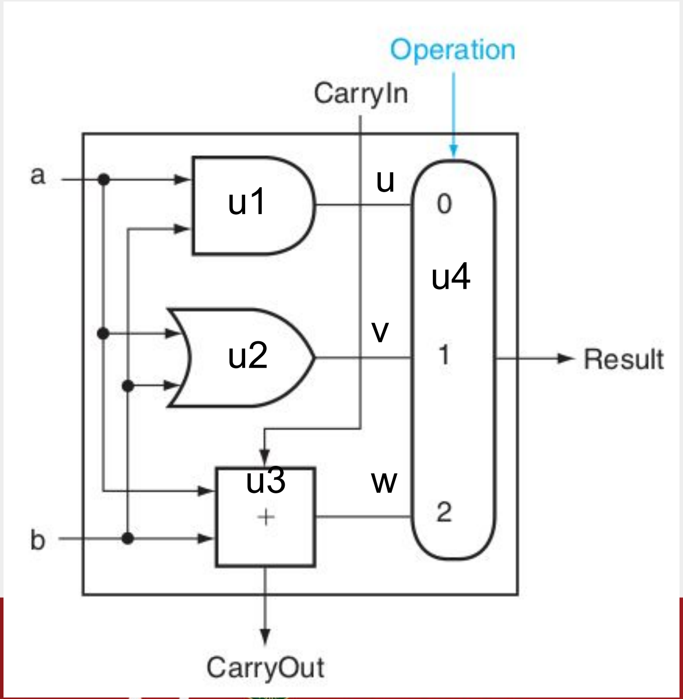


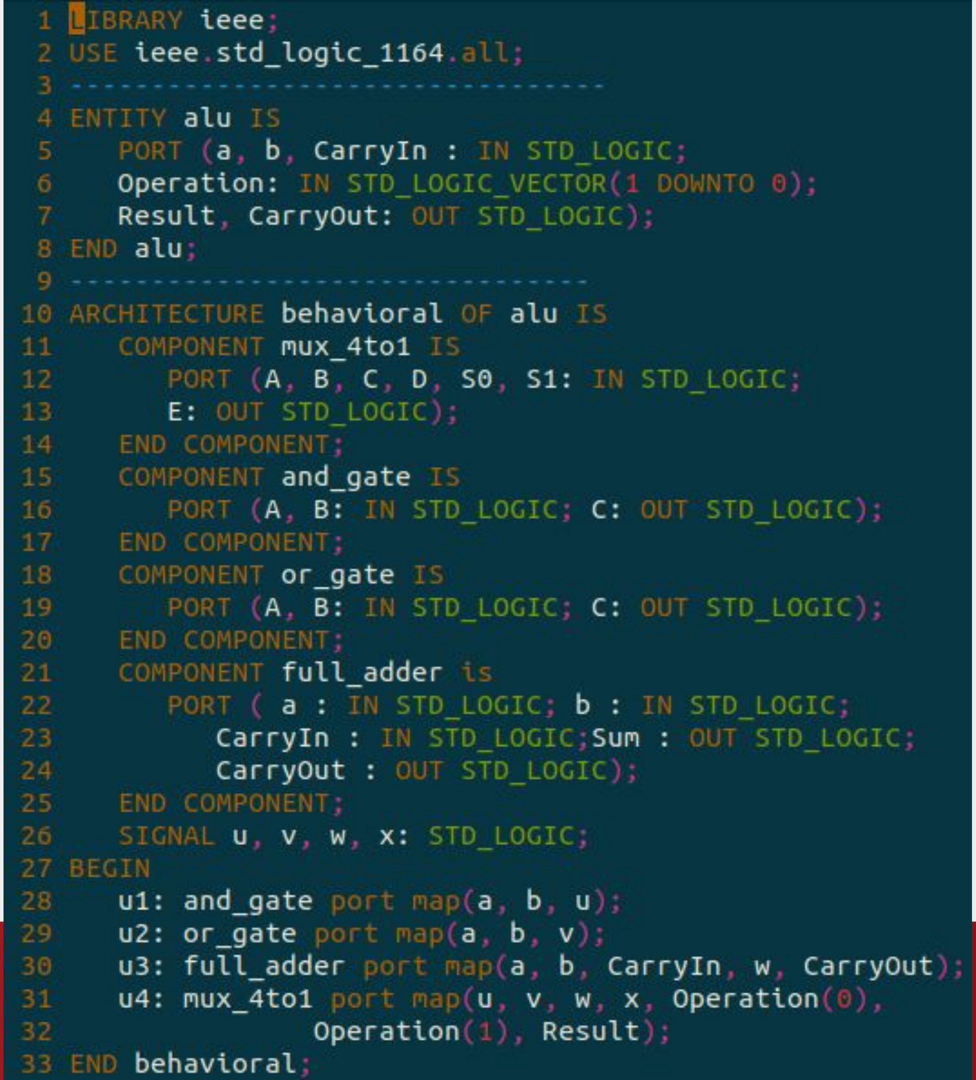
4:1 mux




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
ENTITY mux_4to1 IS
    PORT (A, B, C, D, S0, S1: IN STD_LOGIC;
    E: OUT STD_LOGIC);
END mux_4to1;
---------------------------------------
ARCHITECTURE behavioral OF mux_4to1 IS
    COMPONENT mux_2to1 IS
        PORT (A, B, S: IN STD_LOGIC;
        C: OUT STD_LOGIC);
    END COMPONENT;
    SIGNAL u: STD_LOGIC;
    SIGNAL v: STD_LOGIC;
BEGIN
    u1: mux_2to1 port map(A, B, S1, u);
    u2: mux_2to1 port map(C, D, S1, v);
    u3: mux_2to1 port map(u, v, S0, E);
END behavioral;
```

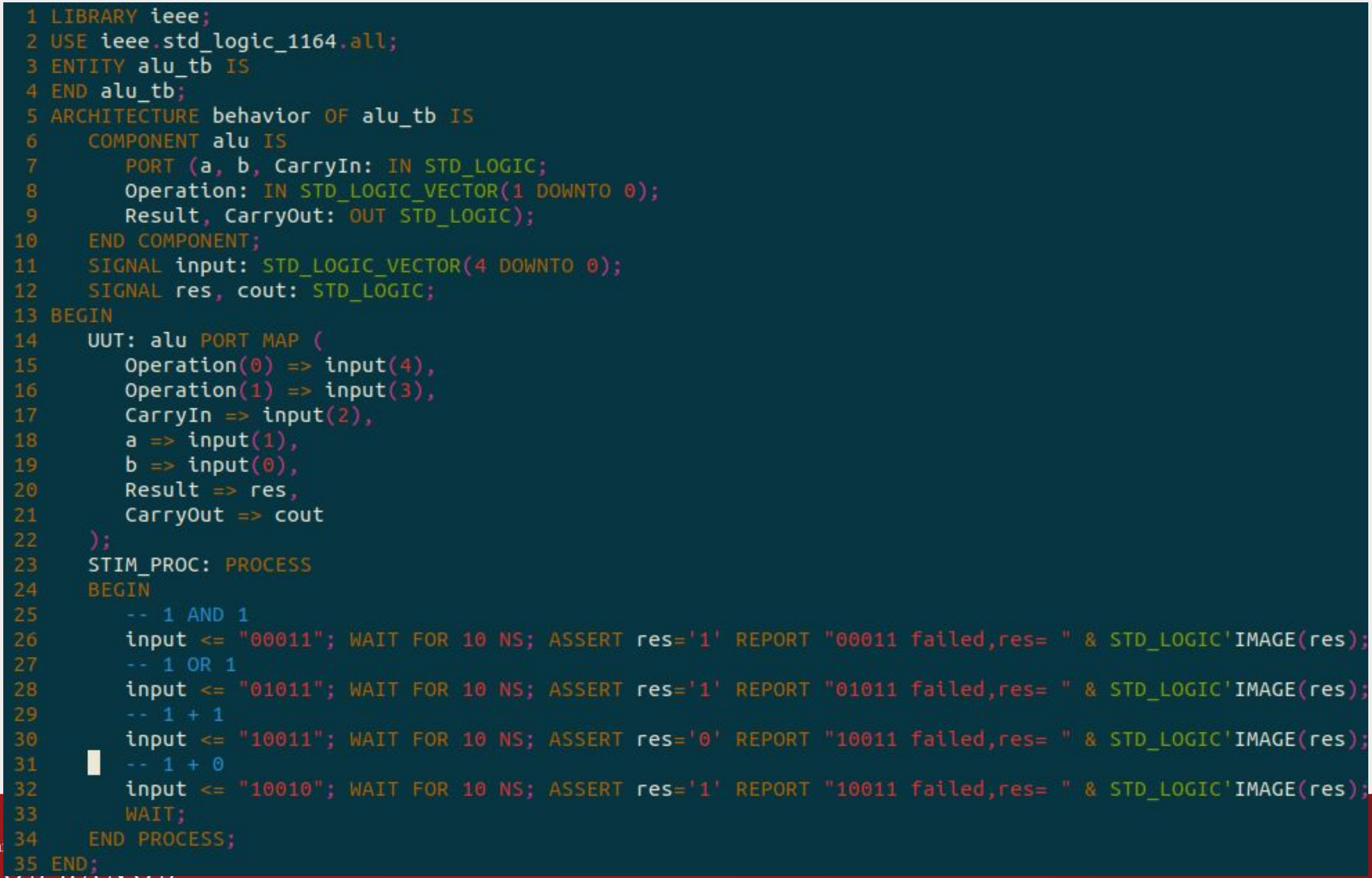
Final design and implementation of a 1-bit ALU





```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------
ENTITY alu IS
   PORT (a, b, CarryIn : IN STD_LOGIC;
   Operation: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
   Result, CarryOut: OUT STD_LOGIC);
END alu;
--------------------------------
ARCHITECTURE behavioral OF alu IS
   COMPONENT mux_4to1 IS
      PORT (A, B, C, D, S0, S1: IN STD_LOGIC;
      E: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT and_gate IS
      PORT (A, B: IN STD_LOGIC; C: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT or_gate IS
      PORT (A, B: IN STD_LOGIC; C: OUT STD_LOGIC);
   END COMPONENT;
   COMPONENT full_adder is
      PORT ( a : IN STD_LOGIC; b : IN STD_LOGIC;
         CarryIn : IN STD_LOGIC;Sum : OUT STD_LOGIC;
         CarryOut : OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL u, v, w, x: STD_LOGIC;
BEGIN
   u1: and_gate port map(a, b, u);
   u2: or_gate port map(a, b, v);
   u3: full_adder port map(a, b, CarryIn, w, CarryOut);
   u4: mux_4to1 port map(u, v, w, x, Operation(0),
               Operation(1), Result);
END behavioral;
```

Testbench



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY alu_tb IS
END alu_tb;
ARCHITECTURE behavior OF alu_tb IS
   COMPONENT alu IS
      PORT (a, b, CarryIn: IN STD_LOGIC;
      Operation: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
      Result, CarryOut: OUT STD_LOGIC);
   END COMPONENT;
   SIGNAL input: STD_LOGIC_VECTOR(4 DOWNTO 0);
   SIGNAL res, cout: STD_LOGIC;
BEGIN
   UUT: alu PORT MAP (
      Operation(0) => input(4),
      Operation(1) => input(3),
      CarryIn => input(2),
      a => input(1),
      b => input(0),
      Result => res,
      CarryOut => cout
   );
   STIM_PROC: PROCESS
   BEGIN
      -- 1 AND 1
      input <= "00011"; WAIT FOR 10 NS; ASSERT res='1' REPORT "00011 failed,res= " & STD_LOGIC'IMAGE(res);
      -- 1 OR 1
      input <= "01011"; WAIT FOR 10 NS; ASSERT res='1' REPORT "01011 failed,res= " & STD_LOGIC'IMAGE(res);
      -- 1 + 1
      input <= "10011"; WAIT FOR 10 NS; ASSERT res='0' REPORT "10011 failed,res= " & STD_LOGIC'IMAGE(res);
      -- 1 + 0
      input <= "10010"; WAIT FOR 10 NS; ASSERT res='1' REPORT "10011 failed,res= " & STD_LOGIC'IMAGE(res);
      WAIT;
   END PROCESS;
END;
```