***Central Processing Unit (CPU)***

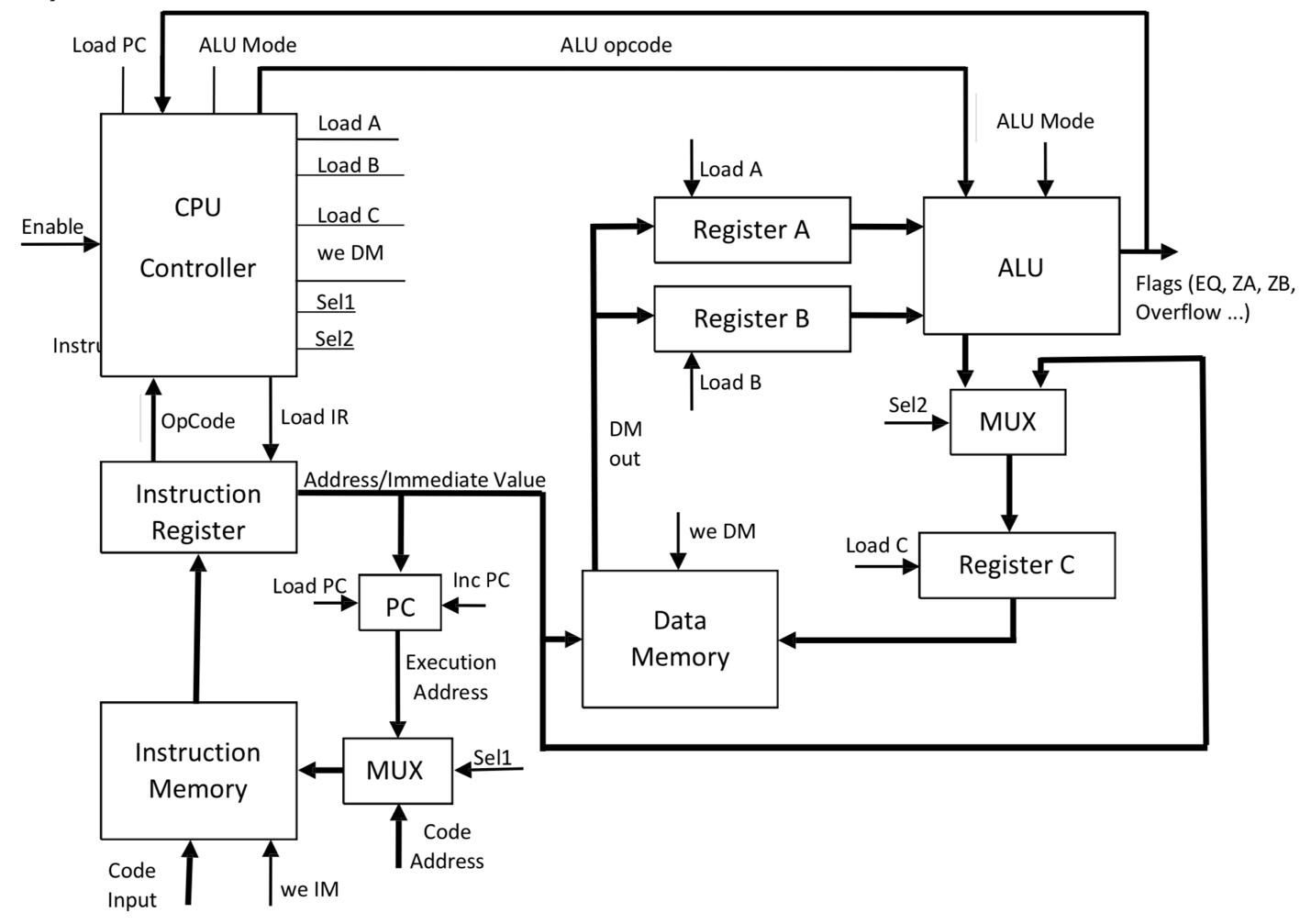
Central processing unit (CPU), principal part of any digital [computer](https://www.britannica.com/technology/computer) system, generally composed of the main [memory](https://www.britannica.com/technology/computer-memory), control unit, and arithmetic-logic unit. It [constitutes](https://www.merriam-webster.com/dictionary/constitutes) the physical heart of the entire computer system; to it is linked various [peripheral](https://www.britannica.com/technology/input-output-device) equipment, including [input/output devices](https://www.britannica.com/technology/input-output-device) and [auxiliary](https://www.merriam-webster.com/dictionary/auxiliary) storage units. In modern computers, the CPU is contained on an [integrated circuit](https://www.britannica.com/technology/integrated-circuit) [chip](https://www.britannica.com/technology/computer-chip) called a [microprocessor](https://www.britannica.com/technology/microprocessor).

The [control unit](https://www.britannica.com/technology/control-unit) of the central processing unit regulates and [integrates](https://www.merriam-webster.com/dictionary/integrates) the operations of the computer. It selects and retrieves instructions from the main memory in proper sequence and interprets them so as to activate the other functional elements of the system at the appropriate moment to perform their respective operations. All input data are transferred via the main memory to the [arithmetic-logic unit](https://www.britannica.com/technology/arithmetic-logic-unit) for processing, which involves the four basic [arithmetic](https://www.britannica.com/science/arithmetic) functions (i.e., addition, subtraction, multiplication, and division) and certain [logic](https://www.britannica.com/topic/logic) operations such as the comparing of data and the selection of the desired problem-solving procedure or a viable [alternative](https://www.merriam-webster.com/dictionary/alternative) based on predetermined decision [criteria](https://www.merriam-webster.com/dictionary/criteria).

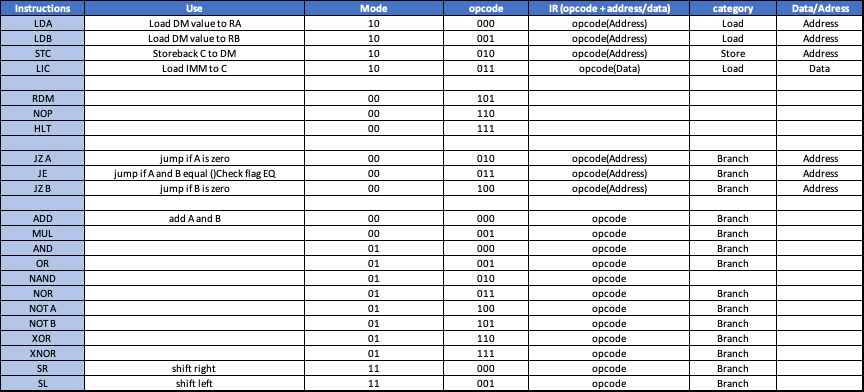
In the past, computer processors used numbers to identify the processor and help identify faster processors. For example, the Intel [80486 (486)](https://www.computerhope.com/jargon/num/80486.htm) processor is faster than the [80386 (386)](https://www.computerhope.com/jargon/num/80386.htm) processor. After the introduction of the Intel Pentium processor (which would technically be the 80586), all computer processors started using names like Athlon, Duron, Pentium, and Celeron.

Today, in addition to the different names of computer processors, there are different architectures ([32-bit](https://www.computerhope.com/jargon/num/32bit.htm) and [64-bit](https://www.computerhope.com/jargon/num/64bit.htm)), speeds, and capabilities. Below is a list of the more common types of CPUs for home or business computers.

***Block Diagram:***

******

***Opcodes and Instructions:***



***Source Code of 16 bit ALU:***

--Implementation of a 16 BIT ALU

-- We have Arithmetic Unit, Logical Unit, and Shift operation to be performed by ALU.

-- To select the required operation we have modes for each unit and opcode for every operation to be performed.

-- We have several operations which are port mapped to the ALU and using a multiplexer we select the required output by the ALU

---- and gate-------

library ieee;

use ieee.std\_logic\_1164.all;

entity and2 is

port ( In1, In2 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end and2;

architecture behavior of and2 is

begin

Out1 <= In1 and In2;

end behavior ;

--------- or gate ------

library ieee;

use ieee.std\_logic\_1164.all;

entity or2 is

port ( In1, In2 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end or2;

architecture behavior of or2 is

begin

Out1 <= In1 or In2;

end behavior;

---------nand gate--------

library ieee;

use ieee.std\_logic\_1164.all;

entity nand2 is

port (In1, In2 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end nand2;

architecture behavior of nand2 is

begin

Out1 <= In1 nand In2;

end behavior;

----nor gate-----

library ieee;

use ieee.std\_logic\_1164.all;

entity nor2 is

port (In1, In2 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end nor2;

architecture behavior of nor2 is

begin

Out1 <= In1 nor In2;

end behavior;

-----------------notgate-----

library ieee;

use ieee.std\_logic\_1164.all;

entity not2 is

port (In1 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end not2;

architecture behavior of not2 is

begin

Out1 <= not In1;

end behavior;

---------------xor-----------

library ieee;

use ieee.std\_logic\_1164.all;

entity xor2 is

port (In1, In2 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end xor2;

architecture behavior of xor2 is

begin

Out1 <= In1 xor In2;

end behavior;

-----------xnor------------

library ieee;

use ieee.std\_logic\_1164.all;

entity xnor2 is

port (In1, In2 : in STD\_LOGIC\_VECTOR (15 downto 0);

Out1 : out STD\_LOGIC\_VECTOR (15 downto 0));

end xnor2;

architecture behavior of xnor2 is

begin

Out1 <= In1 xnor In2;

end behavior;

-----------multiplier-----------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_std.all;

entity mul is

port ( In1,In2 : in std\_logic\_vector(15 downto 0);

Out1 : out std\_logic\_vector(31 downto 0));

end entity mul;

architecture behavior of mul is

signal in1\_unsig , in2\_unsig : unsigned (15 downto 0);

signal out\_unsig : unsigned (31 downto 0);

begin

in1\_unsig <= unsigned(In1);

in2\_unsig <= unsigned (In2);

out\_unsig <= in1\_unsig \* in2\_unsig;

Out1 <= std\_logic\_vector (out\_unsig);

end architecture behavior;

---------------adder--------------

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

use ieee.std\_logic\_arith.all;

--Entity Declaration

entity FullAdder16Bit is

port( A,B: IN std\_logic\_vector(15 DOWNTO 0); -- Two 16 Bit inputs

Cin: IN std\_logic;

Sum: OUT std\_logic\_vector(15 DOWNTO 0); -- 16 Bit Output

Cout: OUT std\_logic);

end FullAdder16Bit;

-- Behavioral design of 16 bit FA

architecture behavior of FullAdder16Bit is

begin --architecture

process (A,B,Cin)

variable carry: std\_logic;

begin

carry := Cin;

G1: for i in 0 to 15 loop

carry := (A(i) and B(i)) or (carry and (a(i) or b(i)));

end loop G1;

Sum <= A + B + Cin;

Cout <= carry;

end process;

end behavior;

------------------ALU-----------------

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ALU is

port (A : in std\_logic\_vector (15 downto 0);

B : in std\_logic\_vector (15 downto 0);

opcode1: in std\_logic\_vector (2 downto 0);

Mode : in std\_logic\_vector (1 downto 0);

output : out std\_logic\_vector(31 downto 0);

Overflow, EQ, GT, ZA, ZB : out std\_logic);

end ALU;

architecture ALU of ALU is

----------------------------AND----------------------------------

component and2

port( in1 : in std\_logic\_vector (15 downto 0);

in2 : in std\_logic\_vector (15 downto 0);

out1 : out std\_logic\_vector (15 downto 0));

end component;

--------------------------OR-------------------------------------

component or2

port( in1 : in std\_logic\_vector (15 downto 0);

in2 : in std\_logic\_vector (15 downto 0);

out1: out std\_logic\_vector (15 downto 0));

end component;

--------------------------NAND------------------------------------

component nand2

port( in1 : in std\_logic\_vector (15 downto 0);

in2 : in std\_logic\_vector (15 downto 0);

out1: out std\_logic\_vector (15 downto 0));

end component;

---------------------------NOR------------------------------------

component nor2

port( in1 : in std\_logic\_vector (15 downto 0);

in2 : in std\_logic\_vector (15 downto 0);

out1: out std\_logic\_vector (15 downto 0));

end component;

----------------------------NOT-----------------------------------

component not2

port( in1 : in std\_logic\_vector (15 downto 0);

out1: out std\_logic\_vector (15 downto 0));

end component;

---------------------------XOR------------------------------------

component xor2

port( in1 : in std\_logic\_vector (15 downto 0);

in2 : in std\_logic\_vector (15 downto 0);

out1: out std\_logic\_vector (15 downto 0));

end component;

--------------------------XNOR------------------------------------

component xnor2

port( in1 : in std\_logic\_vector (15 downto 0);

in2 : in std\_logic\_vector (15 downto 0);

out1: out std\_logic\_vector (15 downto 0));

end component;

--------------------Multiplier-----------------------------------

component mul

port ( In1,In2 : in std\_logic\_vector(15 downto 0);

Out1 : out std\_logic\_vector(31 downto 0));

end component;

--------------------Adder-----------------------------------------

component FullAdder16Bit

port( A,B: IN std\_logic\_vector(15 DOWNTO 0);

Cin: IN std\_logic;

Sum: OUT std\_logic\_vector(15 DOWNTO 0);

Cout: OUT std\_logic);

end component;

-------------------------BS---------------------------------------

component barrelshifter

port(A : IN std\_logic\_vector (15 downto 0);

B : IN std\_logic\_vector (15 downto 0);

Direction : IN std\_logic\_vector (2 downto 0);

Shiftout : OUT std\_logic\_vector(15 downto 0));

end component;

-------------------PORT MAPPING-----------------------------------

signal alu\_and, alu\_or, alu\_nand, alu\_nor, alu\_notA, alu\_notB, alu\_xor , alu\_xnor , alu\_sum, alu\_shift: std\_logic\_vector (15 downto 0);

signal alu\_cin : std\_logic;

signal alu\_mul : std\_logic\_vector (31 downto 0);

begin

logicand : and2 port map (A , B, alu\_and);

logicor : or2 port map (A , B, alu\_or);

logicnand : nand2 port map (A , B, alu\_nand);

logicnor : nor2 port map (A , B, alu\_nor);

logicnotA : not2 port map (A , alu\_notA);

logicnotB : not2 port map (B , alu\_notB);

logicxor : xor2 port map (A ,B, alu\_xor);

logicxnor : xnor2 port map (A ,B, alu\_xnor);

logicmul : mul port map (A ,B, alu\_mul);

Arithadd : FullAdder16Bit port map (A , B, alu\_cin, alu\_sum , overflow);

Shift : barrelShifter port map (A, B, opcode1,alu\_shift );

process(a, b)

begin

if (a = b) then

EQ <= '1';

else

EQ <= '0';

end if;

if ( a > b) then

GT <= '1';

else

GT <= '0';

end if;

if (a = "0000000000000000") then

ZA <= '1';

else

ZA <='0';

end if;

if (b = "0000000000000000") then

ZB <= '1';

else

ZB <= '0';

end if;

end process;

process (opcode1, a, b, alu\_and, alu\_or, alu\_nand, alu\_nor, alu\_notA, alu\_notB, alu\_xor, alu\_xnor, alu\_mul, alu\_sum, alu\_cin,alu\_shift)

begin

if (Mode = "01") then

if (opcode1 = "000") then

output <= "0000000000000000" & alu\_and;

elsif (opcode1 = "001") then

output <= "0000000000000000" & alu\_or;

elsif (opcode1 = "010") then

output <= "0000000000000000" & alu\_nand;

elsif (opcode1 = "011") then

output <= "0000000000000000" & alu\_nor;

elsif (opcode1 = "100") then

output <= "0000000000000000" & alu\_notA;

elsif (opcode1 = "101") then

output <= "0000000000000000" & alu\_notB;

elsif (opcode1 = "110") then

output <= "0000000000000000" & alu\_xor;

elsif (opcode1 = "111") then

output <= "0000000000000000" & alu\_xnor;

end if;

elsif (Mode = "00") then

if (opcode1 = "000") then

output <= "0000000000000000" & alu\_sum;

elsif (opcode1 = "001") then

output <= alu\_mul;

end if;

elsif (Mode = "11") then

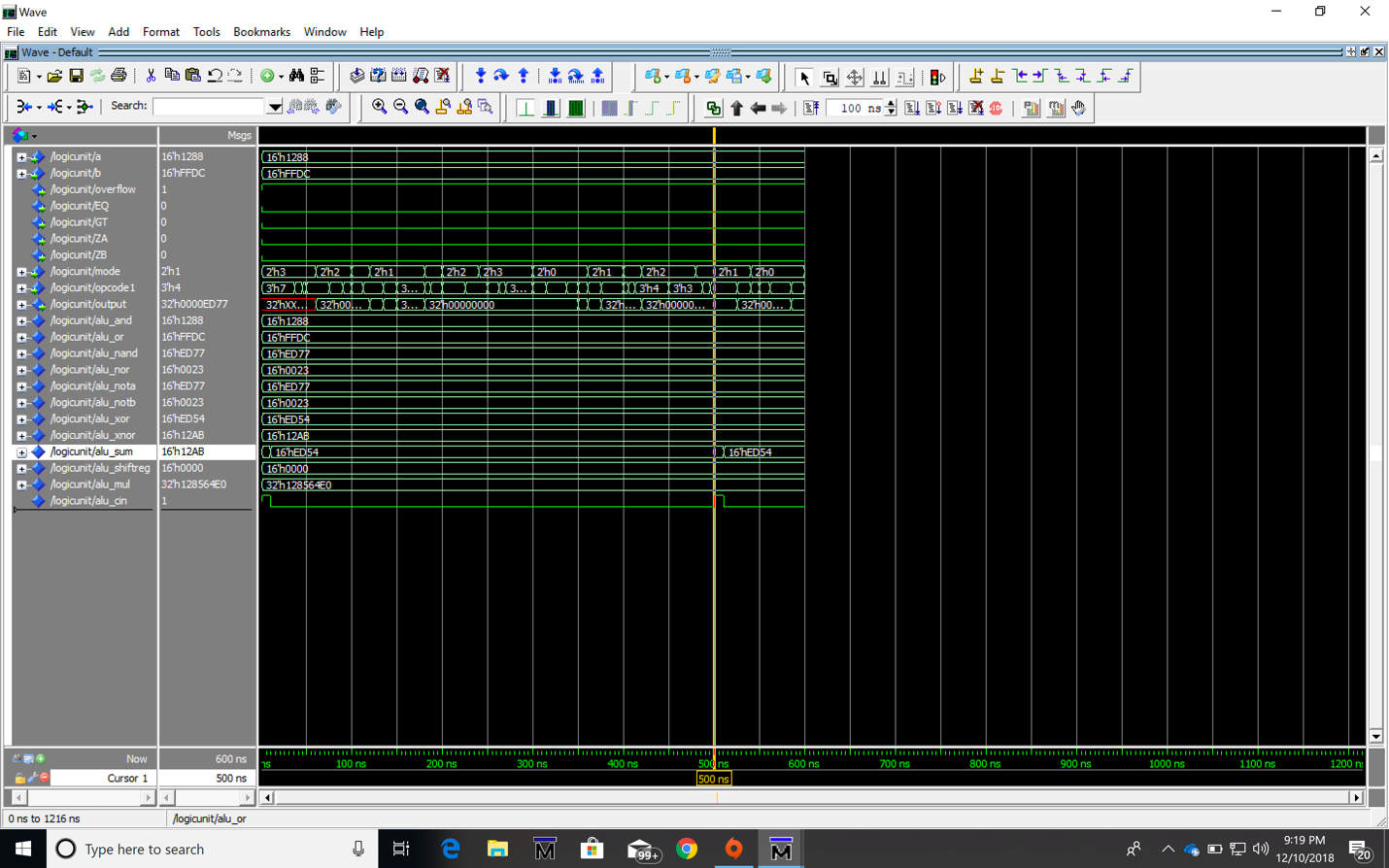
output <= "0000000000000000" & alu\_shift;

end if;

end process;

end ALU;

***Waveform of 16 bit ALU source code:***



***Testbench for 16 bit ALU:***

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_ARITH.ALL;

USE ieee.std\_logic\_UNSIGNED.ALL;

USE ieee.numeric\_std.ALL;

entity ALU\_tb is

end ALU\_tb;

architecture behavior of ALU\_tb is

component ALU

port (A : in std\_logic\_vector (15 downto 0);

B : in std\_logic\_vector (15 downto 0);

opcode1: in std\_logic\_vector (2 downto 0);

Mode : in std\_logic\_vector (1 downto 0);

output : out std\_logic\_vector(31 downto 0);

Overflow, EQ, GT, ZA, ZB : out std\_logic);

end component;

signal A\_tb, B\_tb : std\_logic\_vector (15 downto 0);

signal mode\_tb : std\_logic\_vector (1 downto 0);

signal opcode\_tb : std\_logic\_vector (2 downto 0);

signal Overflow, EQ, GT, ZA, ZB : std\_logic;

signal output : std\_logic\_vector (31 downto 0);

begin

DUT : ALU port map ( A => A\_tb, B => B\_tb , mode => mode\_tb , opcode1 => opcode\_tb , output => output, Overflow =>Overflow , EQ =>EQ , GT => GT, ZA =>ZA , ZB => ZB );

process

begin

A\_tb <= "0000000000000000";

B\_tb <= "0101010101010101";

mode\_tb <= "00";

opcode\_tb <= "001";

wait for 10 ns;

A\_tb <= "0000000000111111";

B\_tb <= "0000000000000000";

mode\_tb <= "01";

opcode\_tb <= "000";

wait for 10 ns;

A\_tb <= "0000000000111111";

B\_tb <= "0000000000111111";

mode\_tb <= "01";

opcode\_tb <= "111";

wait for 10 ns;

A\_tb <= "1111111111111111";

B\_tb <= "1111111111111111";

mode\_tb <= "01";

opcode\_tb <= "101";

wait for 10 ns;

A\_tb <= "1010100000111111";

B\_tb <= "0111110000100010";

mode\_tb <= "01";

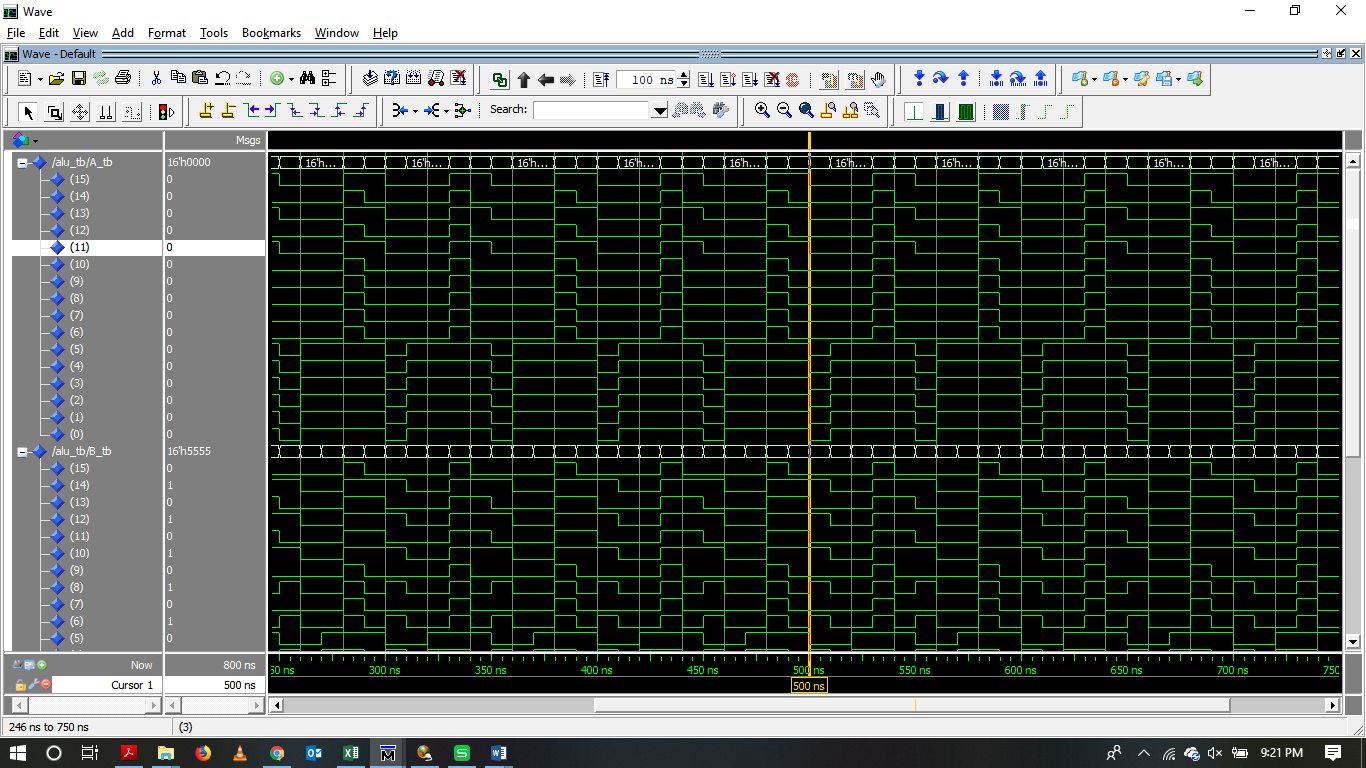
opcode\_tb <= "011";

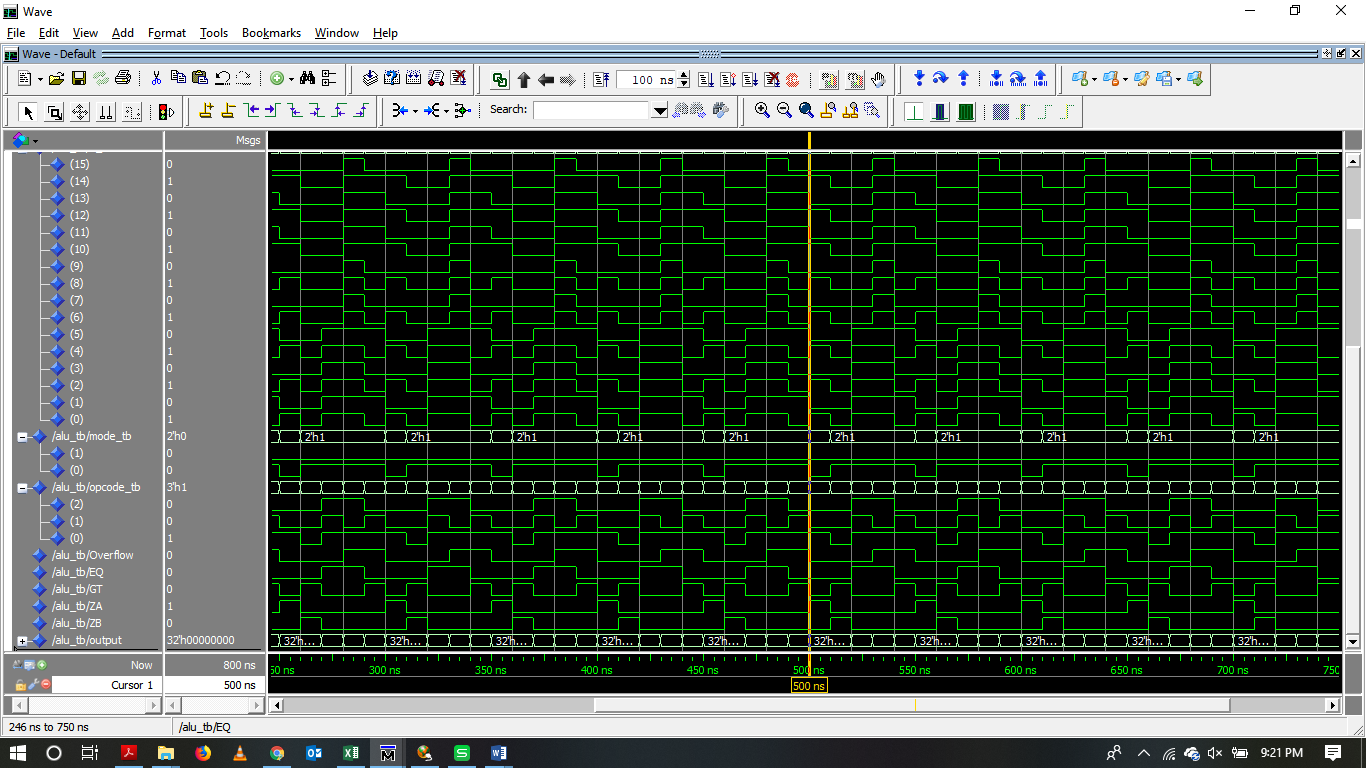
wait for 10 ns;

end process;

end behavior;

***Waveform of Testbench for 16 bit ALU:***





***Source code for Generic Memory:***

--We have a Generic Memory that takes data in, address port, clock and write enable. All read and write operations are clocked.

-- We read when enable is ‘0’ and write when enable is ‘1’.

-- Output displays data at specific address when read is activated, else the output has all zeros.’

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.Numeric\_Std.all;

entity memory is

generic (N : integer := 16;

M : integer := 3);

port ( clock : in std\_logic;

we : in std\_logic;

address : in std\_logic\_vector (M-1 downto 0);

datain : in std\_logic\_vector (N-1 downto 0);

dataout : out std\_logic\_vector (N-1 downto 0));

end entity memory;

architecture mem of memory is

type ram\_type is array (0 to (2\*\*address'length)-1) of std\_logic\_vector (N-1 downto 0);

signal ram : ram\_type;

--signal read\_address : std\_logic\_vector(address'range);

begin

process(clock) is

begin

if (clock'event and clock='1') then

if we = '1' then

ram(to\_integer(unsigned(address))) <= datain;

end if;

end if;

end process;

dataout <= ram(to\_integer(unsigned(address)));

end architecture mem;

***Testbench and Waveform for Generic Memory:***

library IEEE;

use ieee.std\_logic\_1164.all;

USE IEEE.STD\_LOGIC\_ARITH.ALL;

USE IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity memory\_tb is

end memory\_tb;

architecture mem of memory\_tb is

component memory

generic (N : integer := 16;

M : integer := 3);

port ( clock : in std\_logic;

we : in std\_logic;

address : in std\_logic\_vector (M-1 downto 0);

datain : in std\_logic\_vector (N-1 downto 0);

dataout : out std\_logic\_vector (N-1 downto 0));

end component;

signal data\_in : std\_logic\_vector (15 downto 0):= (others => '0');

signal data\_out: std\_logic\_vector (15 downto 0);

signal address: std\_logic\_vector (2 downto 0) := (others => '0');

signal enable: std\_logic := '0';

signal clock: std\_logic:='0';

begin

dut : memory port map (clock, enable, address, data\_in, data\_out);

clock <= not clock after 5 ns;

enable <= '1' after 5 ns, not enable after 10 ns;

address <= address + 1 after 20 ns;

data\_in <= data\_in + 1 after 20 ns;

Simulation\_process: process

begin

enable <= '0';

address <= "000";

data\_in <= x"00FF";

wait for 20 ns;

for i in 0 to 5 loop

address <= address + "001";

wait for 100 ns;

end loop;

address <= "000";

enable <= '1';

-- start writing to memory

wait for 100 ns;

for i in 0 to 5 loop

address <= address + "001";

data\_in <= data\_in-x"0001";

wait for 100 ns;

end loop;

enable <= '0';

address <= "000";

-- start reading data from memory

for i in 0 to 5 loop

address<= address + "001";

wait for 100 ns;

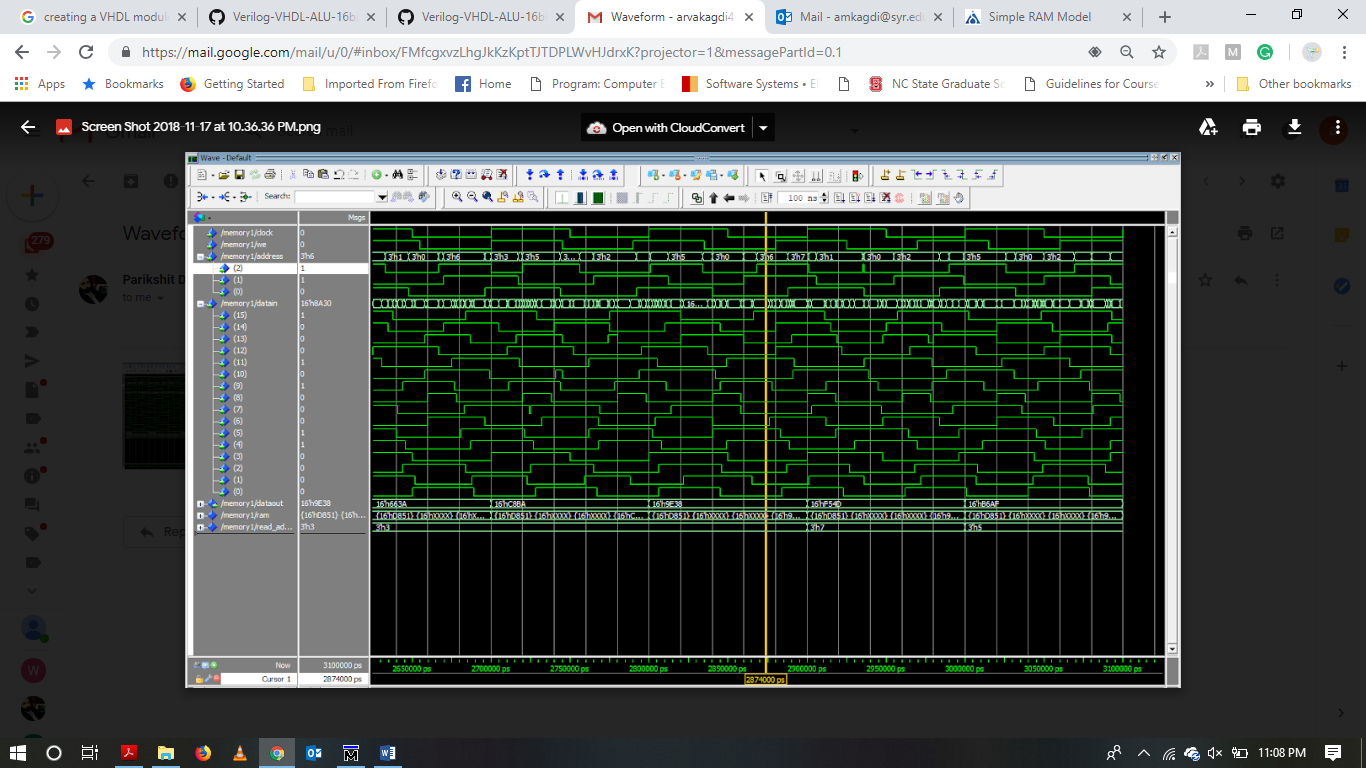
end loop;

wait;

end process;

end mem;

***Waveform of testbench:***



***Source code of Mux:***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Mux IS

Port ( ALUout, IRout: IN std\_logic;

Sel2: IN std\_logic; -- select line for the multiplexer.

MuxOut: OUT std\_logic); -- output the selected input (A for Sel=0)

end Mux;

architecture behavior of Mux is

begin

process(ALUout, IRout ,Sel2)

begin

if (Sel2 = '0') then MuxOut<= ALUout ;

elsif (Sel2 = '1') then MuxOut<= IRout ;

end if;

end process;

end behavior;

***Source code of Program counter:***

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.STD\_LOGIC\_UNSIGNED.ALL;

entity PC is

PORT (clk, en,loadPC,IncPC, reset: IN std\_logic;

IRout: IN std\_logic\_vector(15 downto 0);--INPUT

PCout: OUT std\_logic\_vector(15 downto 0)); --output

end PC;

architecture behaviour of PC is

signal pcReg: std\_logic\_vector(15 downto 0);

begin

process(clk)

begin

if clk'event and clk='1' then

if reset='1' then

pcReg <= x"0000";

elsif loadPC='1' then

pcReg <= IRout;

elsif IncPC ='1' then

pcReg <= pcReg+ x"0001";

end if;

end if;

end process;

PCout <= pcReg when en='1' else "ZZZZZZZZZZZZZZZZ";

end behaviour;

***Source code of Instruction Register:***

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

entity IR is

port(clk,ldir,enableAB, enableDB,reset: IN std\_logic;

abus: OUT std\_logic\_vector(15 downto 0);

dbus: INOUT std\_logic\_vector(15 downto 0);

load,store,add,halt,jump:OUT std\_logic;

Cjump,Iload,Istore,Dload,Dadd:OUT std\_logic);

end IR;

architecture behavior of IR is

signal irReg:std\_logic\_vector(15 downto 0);

begin

process(clk)

begin

if clk'event and clk='0' then

if reset='1' then irReg <= x"0000";

elsif ldir= '1' then irReg <= dbus;

end if;

end if;

end process;

abus <= "0000" & irReg(11 downto 0) when enableAB= '1'

else "ZZZZZZZZZZZZZZZZ";

dbus <= "0000" & irReg(11 downto 0) when enableDB= '1'

else "ZZZZZZZZZZZZZZZZ";

load <= '1' when irReg(15 downto 12) = x"0"

else '0';

store <= '1' when irReg(15 downto 12) = x"1"

else '0';

add <= '1' when irReg(15 downto 12) = x"2"

else '0';

halt <= '1' when irReg= x"3" & x"001"

else '0';

jump <= '1' when irReg(15 downto 12) = x"4"

else '0';

Cjump <= '1' when irReg(15 downto 12) = x"5"

else '0';

Iload <= '1' when irReg(15 downto 12) = x"6"

else '0';

Istore <= '1' when irReg(15 downto 12) = x"7"

else '0';

Dload <= '1' when irReg(15 downto 12) = x"8"

else '0';

Dadd <= '1' when irReg(15 downto 12) = x"9"

else '0';

end behavior;

***Source code of Instruction Memory:***

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_arith.all;

entity instructionmemory is

port(we\_IM, reset, valu: IN std\_logic;

abus: IN std\_logic\_vector(15 downto 0);--input pins

dbus: INOUT std\_logic\_vector(15 downto 0));--output pins

end instructionmemory;

architecture behavioral of instructionmemory is

type ramtype is array(0 to 63) of std\_logic\_vector(15 downto 0);

signal memory:ramtype;

begin

process(reset,we\_IM)

begin

if reset='1' then

memory(0)<= x"000A";

memory(1)<= x"3000";

memory(2)<= x"200B";

memory(3)<= x"100C";

memory(4)<= x"3001";

memory(10)<= x"0010";

memory(11)<= x"0011";

for i in 12 to 63 Loop

memory(i) <=x"0000";

END Loop;

elsif we\_IM'event and we\_IM='0' then

memory(conv\_integer(unsigned(abus)))<=dbus;

end If;

end process;

dbus <= memory(conv\_integer(unsigned(abus))) when reset = '0' and valu='1' and we\_IM='1' else "ZZZZZZZZZZZZZZZZ";

END behavioral;

***Source code of Controller:***

library ieee;

use ieee.std\_logic\_1164.all;

entity controller is

port ( EQ,ZA, ZB, Overflow, en, clb : IN std\_logic;

opcode1: in std\_logic\_vector (2 downto 0);

ALU\_opcode:out std\_logic\_vector (2 downto 0);

Mode : in std\_logic\_vector (1 downto 0);

loadIR, weDM , loadPC, incPC , sel1, sel2 : out std\_logic;

ALUmode : out std\_logic\_vector(1 downto 0);

loadA, loadB, loadC : inout std\_logic);

end controller;

architecture controller of controller is

signal presentstate, nextstate : std\_logic\_vector (1 downto 0);

begin

process (en)

begin

if (clb = '0') then

presentstate <= "00";

else

presentstate <= nextstate;

end if;

end process;

process (ZA, ZB, Overflow, en, clb,opcode1, Mode)

begin

if(presentstate = "00") then

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '0';

sel1 <= '0';

sel2 <= '0';

incPC <= '0';

end if;

if (presentstate = "01") then

loadIR <= '1';

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '0';

ALU\_opcode <= opcode1;

ALUmode <= Mode;

sel1 <= '0';

sel2 <= '0';

incPC <= '0';

end if;

if (presentstate = "10") then

loadIR <= '0';

if (Mode = "01") then

case opcode1 is

when "000" => ALU\_opcode <= opcode1; ----AND

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "001" => ALU\_opcode <= opcode1; ----OR

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "010" => ALU\_opcode <= opcode1; ----NAND

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "011" => ALU\_opcode <= opcode1; ----NOR

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "100" => ALU\_opcode <= opcode1; ----NOTA

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "101" => ALU\_opcode <= opcode1; ----NOTB

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "110" => ALU\_opcode <= opcode1; ----XOR

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "111" => ALU\_opcode <= opcode1; ----XNOR

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when others => ALU\_opcode <= opcode1; ----HALT

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

end case;

elsif (Mode = "00") then

case opcode1 is

when "000" => ALU\_opcode <= opcode1; ----ADD

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "001" => ALU\_opcode <= opcode1; ----MUL

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "010" => ALU\_opcode <= opcode1; ----JZA

ALUmode <= Mode;

if (ZA = '1') then

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '1';

incPC <= '0';

else

loadA <= '0';

loadB <= '0';

loadC <= '1';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

end if;

when "011" => ALU\_opcode <= opcode1; ----JE

ALUmode <= Mode;

if (EQ = '1') then

loadA <= '0';

loadB <= '0';

loadC <= '1';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '1';

incPC <= '0';

else

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '1';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '1';

incPC <= '1';

end if;

when "100" => ALU\_opcode <= opcode1; ----JZB

ALUmode <= Mode;

if (ZA = '1') then

loadIR <= '1';

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '0';

else

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

end if;

when "101" => ALU\_opcode <= opcode1; ----RDM

ALUmode <= Mode;

loadIR <= '1';

loadA <= '0';

loadB <= '0';

loadC <= '1';

weDM <= '0';

loadPC <= '1';

sel1 <= '1';

sel2 <= '0';

incPC <= '1';

when "110" => ALU\_opcode <= opcode1; ----NOP

ALUmode <= Mode;

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "111" => ALU\_opcode <= opcode1; ----HALT

ALUmode <= Mode;

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when others => ALU\_opcode <= opcode1;

ALUmode <= Mode;

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

end case;

elsif (Mode = "10") then

case opcode1 is

when "000" => ALU\_opcode <= opcode1; ----LDA

ALUmode <= Mode;

loadIR <= '1';

loadA <= '1';

loadB <= '0';

loadC <= '1';

weDM <= '0';

loadPC <= '1';

sel1 <= '1';

sel2 <= '0';

incPC <= '1';

when "001" => ALU\_opcode <= opcode1; ----LDB

ALUmode <= Mode;

loadIR <= '1';

loadA <= '0';

loadB <= '1';

loadC <= '1';

weDM <= '0';

loadPC <= '1';

sel1 <= '1';

sel2 <= '0';

incPC <= '1';

when "010" => ALU\_opcode <= opcode1; ----STC

ALUmode <= Mode;

loadIR <= '1';

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '1';

loadPC <= '1';

sel1 <= '0';

incPC <= '1';

when "011" => ALU\_opcode <= opcode1; ----LIC

ALUmode <= Mode;

loadIR <= '1';

loadA <= '0';

loadB <= '0';

loadC <= '1';

loadPC <= '0';

sel1 <= '0';

sel2 <= '1';

incPC <= '1';

when others => ALU\_opcode <= opcode1; ----HALT

ALUmode <= Mode;

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

end case;

elsif (Mode = "11") then

case opcode1 is

when "000" => ALU\_opcode <= opcode1; ----Shift right

ALUmode <= Mode;

loadA <= '1';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when "001" => ALU\_opcode <= opcode1; ----Shift left

ALUmode <= Mode;

loadA <= '0';

loadB <= '0';

loadC <= '0';

weDM <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

when others => ALU\_opcode <= opcode1; ----HALT

ALUmode <= Mode;

loadIR <= '0';

loadA <= '0';

loadB <= '0';

loadC <= '0';

loadPC <= '1';

sel1 <= '0';

sel2 <= '0';

incPC <= '1';

end case;

end if;

end if;

end process;

end controller;

***Source code of Top Level:***

library ieee ;

use ieee.std\_logic\_1164.all;

entity CPU is

port (  A : in std\_logic\_vector (15 downto 0);

        B : in std\_logic\_vector (15 downto 0);

        ALU\_out : out std\_logic\_vector(31 downto 0);

loadIR : out std\_logic;

Overflow, EQ, GT, ZA, ZB : in std\_logic;

enable : in std\_logic;

opcode2 : in std\_logic\_vector (2 downto 0);

Mode1   : in std\_logic\_vector (1 downto 0));

end CPU;

architecture CPU of CPU is

----------------ALU-------------------------

component ALU

port (A : in std\_logic\_vector (15 downto 0);

      B : in std\_logic\_vector (15 downto 0);

      opcode1: in std\_logic\_vector (2 downto 0);

      Mode   : in std\_logic\_vector (1 downto 0);

      output : out std\_logic\_vector(31 downto 0);

      Overflow, EQ, GT, ZA, ZB : out std\_logic);

end component;

--------------MEMORY---------------------------

component memory

port ( clock   : in  std\_logic;

         we      : in  std\_logic;

         address : in  std\_logic\_vector (2 downto 0);

         datain  : in  std\_logic\_vector (15 downto 0);

         dataout : out std\_logic\_vector (15 downto 0));

end component;

----------------MUX----------------------------------

component Mux

Port (  ALUout :  IN std\_logic\_vector(31 downto 0);

IRout: IN std\_logic\_vector(15 downto 0);

Sel2: IN std\_logic; -- select line for the multiplexer.

MuxOut: OUT std\_logic\_vector (15 downto 0)); -- output the selected input (A for Sel=0)

end component;

-----------------PROGRAM COUNTER-----------------------

component PC

PORT (clk, en,loadPC,IncPC, reset: IN std\_logic;

      IRout: IN std\_logic\_vector(15 downto 0);--INPUT

      PCout: OUT std\_logic\_vector(15 downto 0)); --output

end component;

-------------------IR-----------------------------------

component IR

port(clk,ldir,enableAB, enableDB,reset: IN std\_logic;

     abus: OUT std\_logic\_vector(15 downto 0);

     dbus: INOUT std\_logic\_vector(15 downto 0);

     load,store,add,halt,jump:OUT std\_logic;

     Cjump,Iload,Istore,Dload,Dadd:OUT std\_logic);

end component;

--------------------IM-----------------------------------

component instructionmemory

port(we\_IM, reset, valu: IN std\_logic;

     abus: IN std\_logic\_vector(15 downto 0);--input pins

     dbus: INOUT std\_logic\_vector(15 downto 0));--output pins

end component;

---------------------CONTROLLER---------------------------

component controller is

port ( EQ,ZA, GT, ZB, Overflow, en, clb: IN std\_logic;

       opcode1: in std\_logic\_vector (2 downto 0);

       ALU\_opcode:out std\_logic\_vector (2 downto 0);

       Mode   : in std\_logic\_vector (1 downto 0);

       loadIR, weDM , loadPC, incPC , sel1, sel2 : out std\_logic;

       ALUmode : out std\_logic\_vector(1 downto 0);

       loadA, loadB, loadC : inout std\_logic);

end component;

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

signal datain, dataout, Muxout, IRout, PCout, abus: std\_logic\_vector (15 downto 0);

signal address, Opcode1: std\_logic\_vector (2 downto 0);

signal Mode : std\_logic\_vector (1 downto 0);

signal alu\_cin ,we\_IM,valu, DMwe, clk, sel2, en ,Overflow1, EQ1, GT1, ZA1, ZB1 ,loadA, loadB, loadC, clb, sel1, ldIR, loadPC , IncPC, enableAB, enableDB, reset, load, store, add, halt, jump, Cjump,Iload,Istore,Dload,Dadd: std\_logic;

signal ALUout : std\_logic\_vector (31 downto 0);

begin

ALU1 : ALU port map (  A => A,  B => B, opcode1=> opcode1, Mode => Mode, output => ALUout, Overflow => Overflow1, EQ => EQ1, GT => GT1, ZA => ZA1, ZB =>ZB1 );

Memory1 : memory port map (clock => clk , we => DMwe, address => address, datain => datain, dataout => dataout);

MUX1 : MUX port map (ALUout => ALUout, IRout => IRout, sel2 =>sel2, Muxout => Muxout);

PC1  : PC port map (clk => clk, en => en, loadPC => loadPC , IncPC => IncPC, reset => reset, IRout => IRout, PCout => PCout  );

IR1 : IR port map (clk => clk, ldIR => ldIR, enableAB => enableAB, enableDB => enableDB, reset => reset, abus => abus, dbus => IRout, load => load, store => store, add => add, halt => halt, jump => jump, Cjump => Cjump, Iload => Iload, Istore => Istore, Dload => Dload, Dadd => Dadd);

IMem : instructionmemory port map (we\_IM, reset, valu, abus, IRout);

Con : controller port map ( EQ, ZA, GT, ZB, Overflow, enable , clb, opcode2, opcode1, Mode1, loadIR, DMwe, loadPC, IncPC, sel1, sel2, Mode, loadA, loadB, loadC);

end CPU;

***Source code of Test bench:***

LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

USE ieee.std\_logic\_ARITH.ALL;

USE ieee.std\_logic\_UNSIGNED.ALL;

USE ieee.numeric\_std.ALL;

entity CPU\_tb is

end CPU\_tb;

architecture behavior of CPU\_tb is

component CPU

 port(  A : in std\_logic\_vector (15 downto 0);

        B : in std\_logic\_vector (15 downto 0);

        ALU\_out : out std\_logic\_vector(31 downto 0);

Overflow, EQ, GT, ZA, ZB : in std\_logic;

enable : in std\_logic;

loadIR : out std\_logic;

opcode2 : in std\_logic\_vector (2 downto 0);

Mode1   : in std\_logic\_vector (1 downto 0));

end component;

signal  A\_tb : std\_logic\_vector (15 downto 0);

signal  B\_tb : std\_logic\_vector (15 downto 0);

signal  ALU\_out\_tb : std\_logic\_vector(31 downto 0);

signal Overflow\_tb, EQ\_tb, GT\_tb, ZA\_tb, ZB\_tb :  std\_logic;

signal enable\_tb, loadIR\_tb :  std\_logic;

signal opcode2\_tb :  std\_logic\_vector (2 downto 0);

signal Mode1\_tb   :  std\_logic\_vector (1 downto 0);

begin

DUT : CPU port map (A\_tb, B\_tb, ALU\_out\_tb, Overflow\_tb, EQ\_tb, GT\_tb, ZA\_tb, ZB\_tb, enable\_tb, loadIR\_tb, opcode2\_tb, Mode1\_tb);

process

begin

loadIR\_tb <= '1';

Mode1\_tb <= "10";

        opcode2\_tb <= "000";

A\_tb <= "0111011111000110";

wait for 10 ns;

enable\_tb <='1';

loadIR\_tb <= '1';

Mode1\_tb <= "10";

        opcode2\_tb <= "001";

B\_tb <= "0101000001010101";

wait for 10 ns;

enable\_tb <= '1';

loadIR\_tb <= '1';

A\_tb <= "0000000000000000";

        B\_tb <= "0000000000000000";

        Mode1\_tb <= "01";

        opcode2\_tb <= "100";

wait for 10 ns;

Mode1\_tb <= "01";

        opcode2\_tb <= "101";

B\_tb <= "0000000000000000";

wait for 10 ns;

Mode1\_tb <= "00";

        opcode2\_tb <= "011";

wait for 20 ns;

enable\_tb <= '1';

Mode1\_tb <= "00";

        opcode2\_tb <= "010";

wait for 20 ns;

enable\_tb <= '1';

  A\_tb <= "0000011111000110";

        B\_tb <= "0101010101010101";

        Mode1\_tb <= "00";

        opcode2\_tb <= "001";

loadIR\_tb <= '1';

        wait for 10 ns;

enable\_tb <='0';

wait for 20 ns;

enable\_tb <= '1';

loadIR\_tb <= '0';

wait for 10 ns;

enable\_tb <= '1';

loadIR\_tb <= '1';

A\_tb <= "1111111111000110";

        B\_tb <= "0000000000000000";

        Mode1\_tb <= "00";

        opcode2\_tb <= "000";

end process;

end behavior;

***Assembly code of Test bench:***

1. LDA
2. LDB
3. NOTA
4. NOTB
5. JE
6. JZA
7. MUL
8. ADD

***Remarks***

* This is the final project of Digital Machine Design which implements CPU design.
* In this implementation of 16-bit CPU we utilize various modules such as IR, ALU, PC, Memory, Registers, and Controller.
* In this architecture the computed results are obtained by applying various opcodes and modes.