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

**Definition - What does *Central Processing Unit (CPU)* mean?**

The central processing unit (CPU) is the unit which performs most of the processing inside a computer. To control instructions and data flow to and from other parts of the computer, the CPU relies heavily on a chipset, which is a group of microchips located on the motherboard.

The CPU has two components:

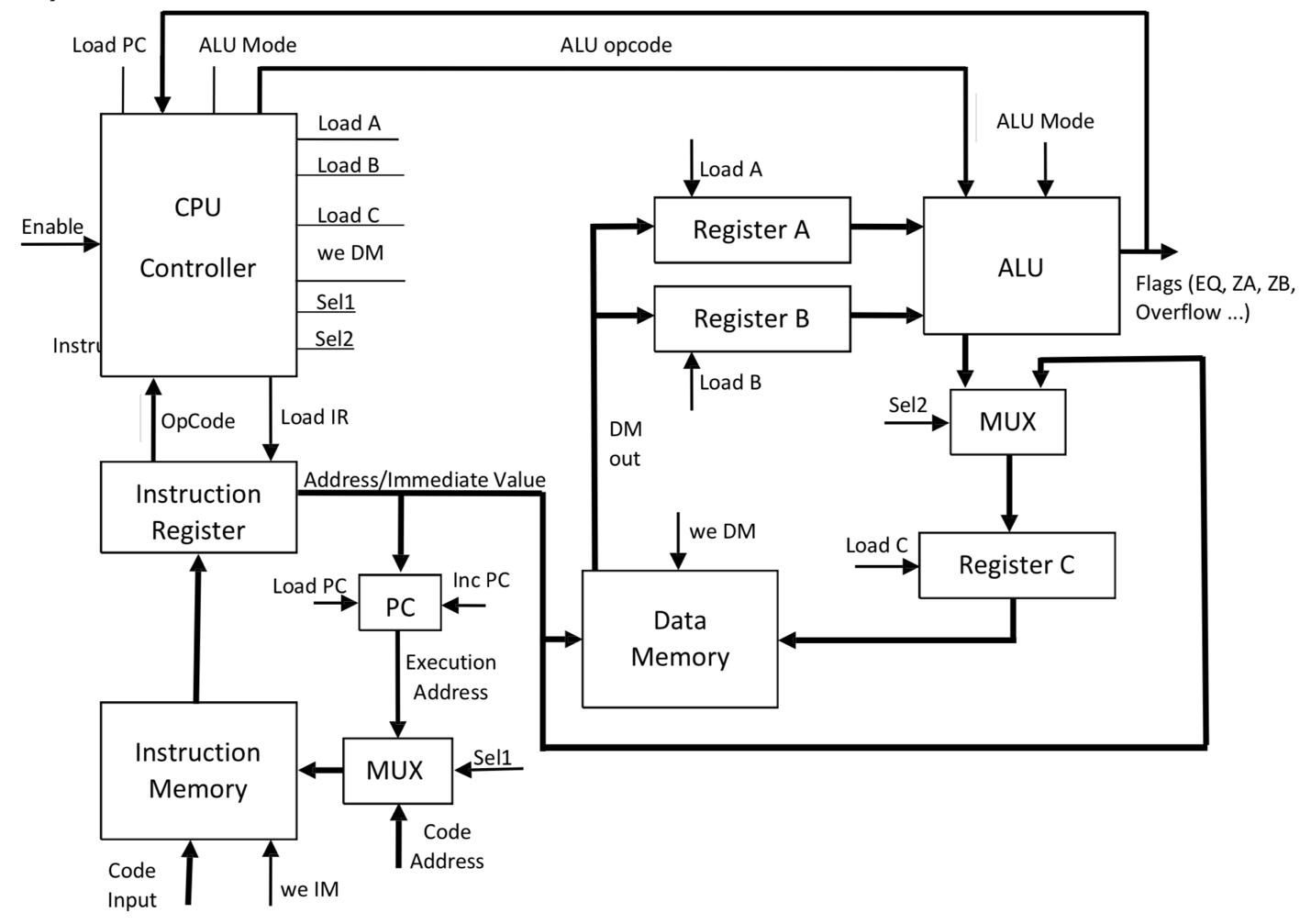
* Control Unit: extracts instructions from memory and decodes and executes them
* To function properly, the CPU relies on the system clock, memory, secondary storage, and data and address buses.
* This term is also known as a central processor, microprocessor or chip.

The CPU is the heart and brain of a computer. It receives data input, executes instructions, and processes information. It communicates with input/output (I/O) devices, which send and receive data to and from the CPU. Additionally, the CPU has an internal bus for communication with the internal cache memory, called the backside bus. The main bus for data transfer to and from the CPU, memory, chipset, and AGP socket is called the front-side bus.

The CPU contains internal memory units, which are called registers. These registers contain data, instructions, counters and addresses used in the ALU's information processing.

Some computers utilize two or more processors. These consist of separate physical CPUs located side by side on the same board or on separate boards. Each CPU has an independent interface, separate cache, and individual paths to the system front-side bus. Multiple processors are ideal for intensive parallel tasks requiring multitasking. Multicore CPUs are also common, in which a single chip contains multiple CPUs.

***Block Diagram:***

******

***Instructions and Opcodes:***

******

***Components of CPU (Source-Code, Testbench and waveforms)***

*Source code of ALU:*

--*In this design of a 16-bit ALU, we implement 3 modes of operation which execute independently with the combination of modes.*

*-- This design implements 2 arithmetic functions, 8 logic functions and two shift operations.*

*-- The various status bits are set as per the input/output conditions.*

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

---\*\* AND GATE\*\*---

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

library ieee;

use ieee.std\_logic\_1164.all;

entity and16 is

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end and16;

architecture and16 of and16 is

begin

out1 <= in1 and in2;

end and16;

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

---\*\* OR \*\*---

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

library ieee;

use ieee.std\_logic\_1164.all;

entity or16 is

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end or16;

architecture orgate of or16 is

begin

out1 <= in1 or in2;

end orgate;

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

--\*\* Nand \*\*--

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

library ieee;

use ieee.std\_logic\_1164.all;

entity nand2 is

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end nand2;

architecture behavior of nand2 is

begin

out1 <= in1 nand in2;

end behavior;

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

--\*\* NOT \*\*--

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

library ieee;

use ieee.std\_logic\_1164.all;

entity nor2 is

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end nor2;

architecture behavior of nor2 is

begin

out1 <= in1 nor in2;

end behavior;

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

--\*\* NOR \*\*---

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

library ieee;

use ieee.std\_logic\_1164.all;

entity not1 is

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end not1;

architecture behavior of not1 is

begin

out1 <= not in1;

end behavior;

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

--\*\* XOR \*\*---

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

library ieee;

use ieee.std\_logic\_1164.all;

entity xor2 is

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 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

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 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

generic (n : integer := 16);

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (2\*n-1 downto 0));

end entity mul;

architecture behavior of mul is

signal in1\_uns, in2\_uns : unsigned (n-1 downto 0);

signal out\_uns : unsigned (2\*n-1 downto 0);

begin

in1\_uns <= unsigned(in1);

in2\_uns <= unsigned(in2);

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

out1 <= std\_logic\_vector(out\_uns);

end behavior;

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

--\*\* Barrel Shifter \*\*--

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

-- Source code of Barrel Shifter

-- Barrel shifter is a circuit which shifts the input right or left based on the inputs received as a and b. Where d determines the direction of shifting, when d is 0 the output is left shifted.

-- When d is equal to 1, the output is shifted right and the amount of shift is determined by the input combination of b

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.numeric\_bit;

use ieee.std\_logic\_arith;

entity barrelshifter is

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

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

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

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

end barrelshifter;

architecture behavior of barrelshifter is

begin

process (in1,in2, direction)

begin

if Direction = "001" then

case in2 is

when "0000000000000000" => Shiftout <= in1;

when "0000000000000001" => Shiftout <= in1(14 downto 0) & in1(15);

when "0000000000000010" => Shiftout <= (in1(13 downto 0) & in1(15) & in1(14));

when "0000000000000011" => Shiftout <= (in1(12 downto 0) & in1(15 downto 13));

when "0000000000000100" => Shiftout <= (in1(11 downto 0) & in1(15 downto 12));

when "0000000000000101" => Shiftout <= (in1(10 downto 0) & in1(15 downto 11));

when "0000000000000110" => Shiftout <= (in1(9 downto 0) & in1(15 downto 10));

when "0000000000000111" => Shiftout <= (in1(8 downto 0) & in1(15 downto 9));

when "0000000000001000" => Shiftout <= (in1(7 downto 0) & in1(15 downto 8));

when "0000000000001001" => Shiftout <= (in1(6 downto 0) & in1(15 downto 7));

when "0000000000001010" => Shiftout <= (in1(5 downto 0) & in1(15 downto 6));

when "0000000000001011" => Shiftout <= (in1(4 downto 0) & in1(15 downto 5));

when "0000000000001100" => Shiftout <= (in1(3 downto 0) & in1(15 downto 4));

when "0000000000001101" => Shiftout <= (in1(2 downto 0) & in1(15 downto 3));

when "0000000000001110" => Shiftout <= (in1(1 downto 0) & in1(15 downto 2));

when "0000000000001111" => Shiftout <= (in1(0) & in1(15 downto 1));

when others => Shiftout <= "0000000000000000";

end case;

else

case in2 is

when "0000000000000000" => Shiftout <= in1;

when "0000000000000001" => Shiftout <= (in1(0) & in1(15 downto 1));

when "0000000000000010" => Shiftout <= (in1(1) & in1(0) & in1(15 downto 2));

when "0000000000000011" => Shiftout <= (in1(2 downto 0) & in1(15 downto 3));

when "0000000000000100" => Shiftout <= (in1(3 downto 0) & in1(15 downto 4));

when "0000000000000101" => Shiftout <= (in1(4 downto 0) & in1(15 downto 5));

when "0000000000000110" => Shiftout <= (in1(5 downto 0) & in1(15 downto 6));

when "0000000000000111" => Shiftout <= (in1(6 downto 0) & in1(15 downto 7));

when "0000000000001000" => Shiftout <= (in1(7 downto 0) & in1(15 downto 8));

when "0000000000001001" => Shiftout <= (in1(8 downto 0) & in1(15 downto 9));

when "0000000000001010" => Shiftout <= (in1(9 downto 0) & in1(15 downto 10));

when "0000000000001011" => Shiftout <= (in1(10 downto 0) & in1(15 downto 11));

when "0000000000001100" => Shiftout <= (in1(11 downto 0) & in1(15 downto 12));

when "0000000000001101" => Shiftout <= (in1(12 downto 0) & in1(15 downto 13));

when "0000000000001110" => Shiftout <= (in1(13 downto 0) & in1(15 downto 14));

when "0000000000001111" => Shiftout <= (in1(14 downto 0) & in1(15));

when others => Shiftout <= "0000000000000000";

end case;

end if;

end process;

end architecture;

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

--\*\* Full 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;

G2: for i in 0 to 15 loop

sum(i) <= A(i) xor B(i) xor cin;

end loop G2;

--Sum <= A xor B xor Cin;

Cout <= carry;

end process;

end behavior;

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

--\*\* Logic unit \*\*--

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

library ieee;

use ieee.std\_logic\_1164.all;

entity logicunit is

generic (n : integer := 16);

port (a : in std\_logic\_vector (n-1 downto 0);

b : in std\_logic\_vector (n-1 downto 0);

overflow, EQ, GT, ZA, ZB : out std\_logic;

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

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

output : out std\_logic\_vector(2\*n-1 downto 0));

end logicunit;

architecture logicunit of logicunit is

component and16

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component or16

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component nand2

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component nor2

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component xor2

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component xnor2

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component not1

port( in1 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (n-1 downto 0));

end component;

component mul

port( in1 : in std\_logic\_vector (n-1 downto 0);

in2 : in std\_logic\_vector (n-1 downto 0);

out1 : out std\_logic\_vector (2\*n-1 downto 0));

end component;

component fulladder16bit

port( a,b: in std\_logic\_vector(n-1 downto 0); -- two 16 bit inputs

cin: in std\_logic;

sum: out std\_logic\_vector(n-1 downto 0); -- 16 bit output

cout: out std\_logic);

end component;

component barrelshifter -- Entity declaration of barrelshifter

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

in2 : 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;

signal alu\_and, alu\_or, alu\_nand, alu\_nor, alu\_nota, alu\_notb, alu\_xor, alu\_xnor, alu\_sum, alu\_shiftreg : std\_logic\_vector (n-1 downto 0);

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

signal alu\_cin : std\_logic;

begin

p1: and16 port map(a, b, alu\_and);

p2: or16 port map(a , b, alu\_or);

p3: nand2 port map(a, b, alu\_nand);

p4: nor2 port map(a , b, alu\_nor);

p5: not1 port map(a, alu\_nota);

p6: not1 port map(b, alu\_notb);

p7: xor2 port map(a, b, alu\_xor);

p8: xnor2 port map(a , b, alu\_xnor);

pau1: mul port map(a, b, alu\_mul);

paufa: fulladder16bit port map(a, b, alu\_cin, alu\_sum, overflow);

pshift : barrelshifter port map(a, b, opcode1, alu\_shiftreg);

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, mode, alu\_and, alu\_or, alu\_nand, alu\_nor, alu\_nota, alu\_notb, alu\_xor, alu\_xnor, alu\_sum, alu\_mul, alu\_cin, alu\_shiftreg)

begin

if (mode = "01") then

case opcode1 is

when "000" => output <= ("0000000000000000" & alu\_and);

when "001" => output <= "0000000000000000" & alu\_or;

when "010" => output <= "0000000000000000" & alu\_nand;

when "011" => output <= "0000000000000000" & alu\_nor;

when "100" => output <= "0000000000000000" & alu\_nota;

when "110" => output <= "0000000000000000" & alu\_xor;

when "111" => output <= "0000000000000000" & alu\_xnor;

when others => output <= "0000000000000000" & "0000000000000000";

end case;

elsif (mode = "00") then

case opcode1 is

when "000" => output <= "0000000000000000" & alu\_sum;

when "001" => output <= alu\_mul;

when others => output <= "0000000000000000" & "0000000000000000";

end case;

elsif (mode = "11") then

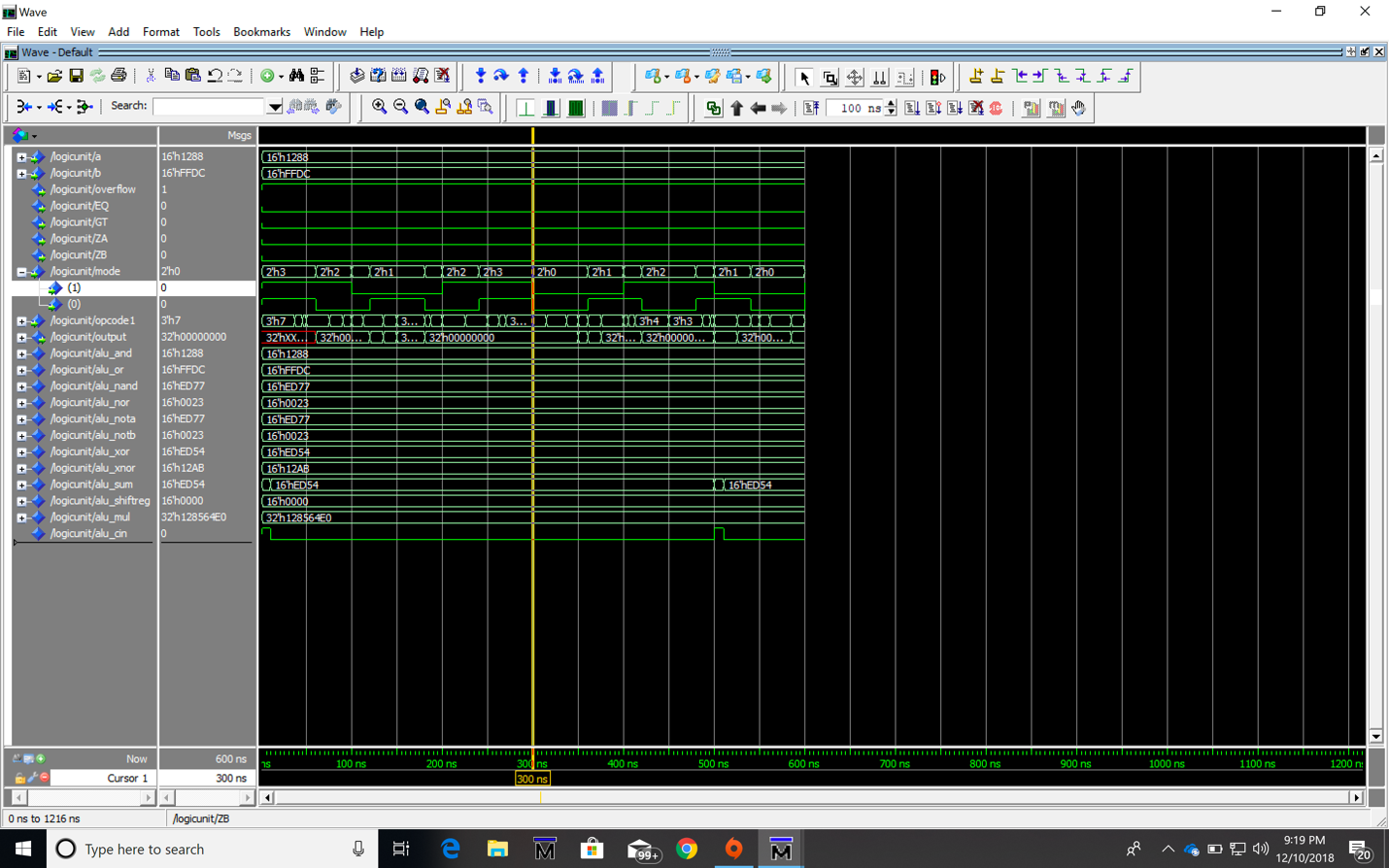
output <= "0000000000000000" & alu\_shiftreg;

end if;

end process;

end logicunit;

*Waveform for source code for ALU:*



*Testbench code for 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 logicunit

generic (n : integer := 16);

port (a : in std\_logic\_vector (n-1 downto 0);

b : in std\_logic\_vector (n-1 downto 0);

overflow, EQ, GT, ZA, ZB : out std\_logic;

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

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

output : out std\_logic\_vector(2\*n-1 downto 0));

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);

begin

DUT : logicunit port map ( a => a\_tb, b => b\_tb , mode => mode\_tb , opcode1 => opcode\_tb);

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";

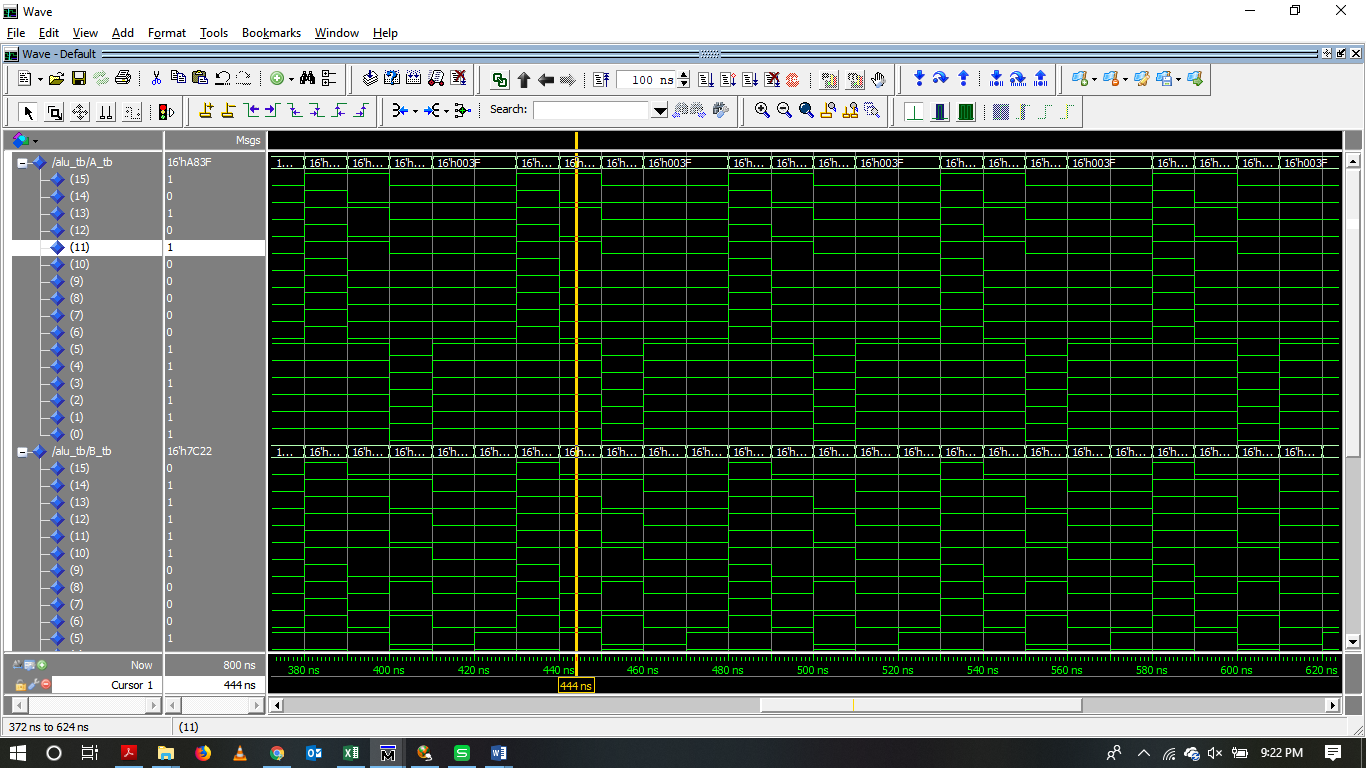
opcode\_tb <= "011";

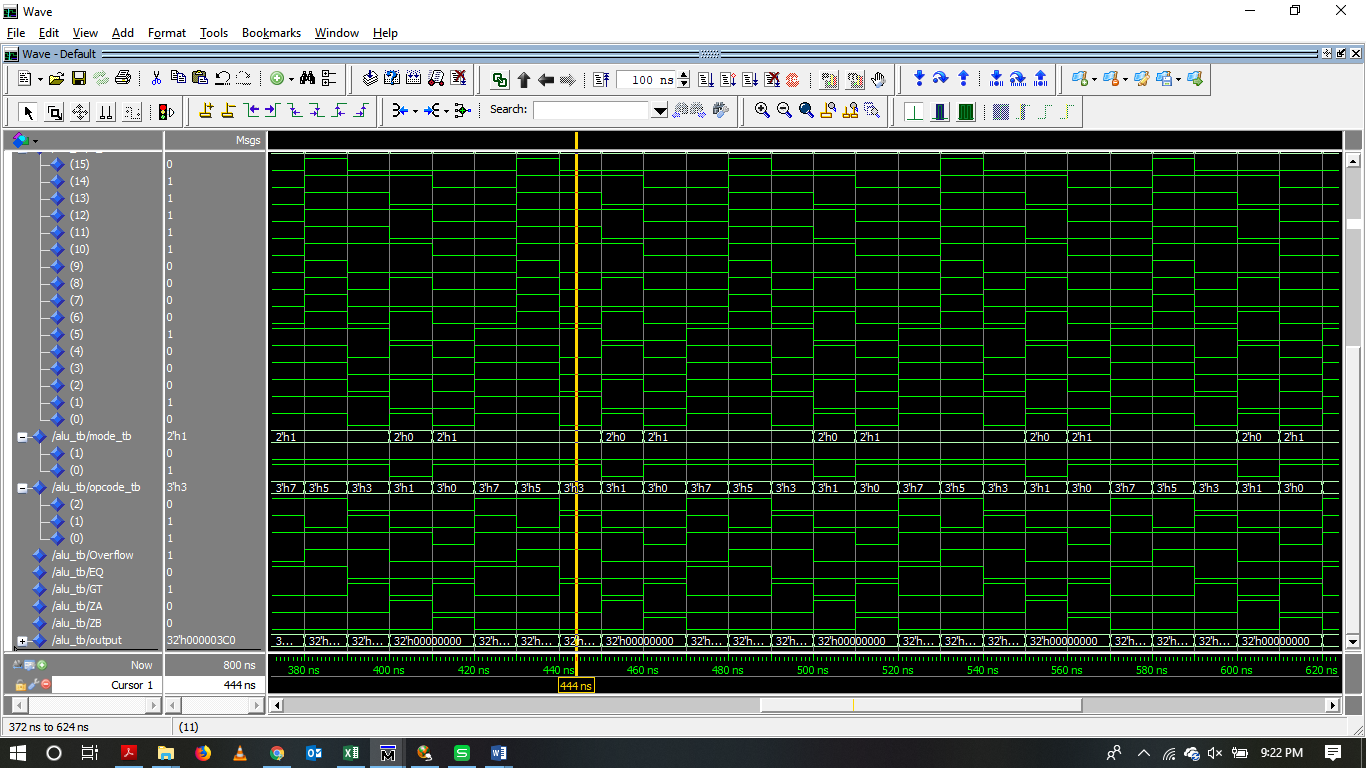
wait for 10 ns;

end process;

end behavior;

*Waveform of Testbench code for ALU:*





*Source code for the generic Memory:*

*--This design implements a generic memory which has a clock, data in and the address port.*

*-- We have used generic parameters to the inputs which can be read/written with various combinations of enable signal.*

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 code of the 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;

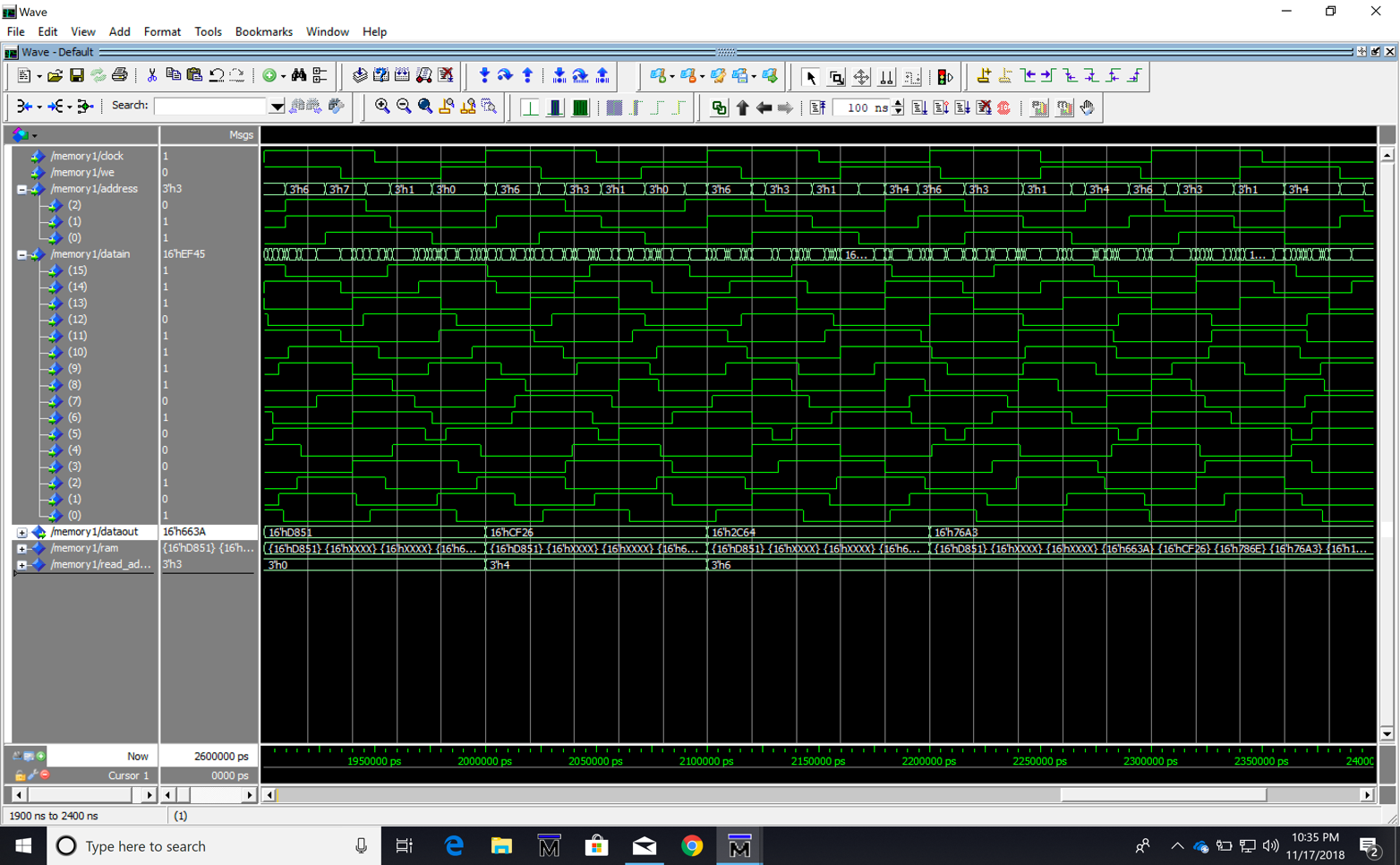
end loop;

wait;

end process;

end mem;

*Waveform of Testbench Design for Generic Memory:*



***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);

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 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 CPU Testbench:***

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 <= "10";

        opcode2\_tb <= "010";

wait for 10 ns;

Mode1\_tb <= "00";

        opcode2\_tb <= "011";

B\_tb <= "0000000011111111";

wait for 10 ns;

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

Mode1\_tb <= "01";

        opcode2\_tb <= "011";

wait for 20 ns;

enable\_tb <= '1';

Mode1\_tb <= "11";

        opcode2\_tb <= "000";

wait for 20 ns;

enable\_tb <= '1';

  A\_tb <= "0000011111000110";

        B\_tb <= "0101010101010101";

        Mode1\_tb <= "01";

        opcode2\_tb <= "110";

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 <= "11";

        opcode2\_tb <= "001";

end process;

end behavior;

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

1. LDA
2. LDB
3. STC
4. JE
5. NOR
6. SR
7. NOR
8. SL

***Remarks***

* *In this final project a 16-bit CPU was designed in VHDL using ModelSIm simulator.*
* *This project utilized the ALU architecture implemented in the previous project.*
* *In this project we also used the generic memory designed in earlier project.*
* *In this implementation of CPU various opcodes are utilized to get desired results of operations.*