Zagazig University

Hardware Design Project

No	Name	Academic Number	Email
1	Matilda Ashraf Malak Mikhael	20812021201241	matkda.ashrafmalak@gmail.com
2	Mariam Osman Hussein Attia	20812021201200	mariamosman281@gmail.com
3	Mariam Farouk Slama Adly	20812021201289	mariamfareed804@gmail.com
4	Norhan Mohamed Elsayed	20812021200520	norhan2312003@gmail.com
5	Hager Mohamed Ali Abdullah	20812021200197	hm7125065@gmail.com

Under Supervision

Dr.Howida Abd AlLatif

Eng.Alaa

Eng.Ferial

Computer and Systems Engineering

Table of Contents

Introduction

The Assignment Aim

- **The primary goal of this project is to design, implement, and evaluate a Single Cycle MIPS Processor using VHDL. This project aims to provide hands-on experience in processor architecture and digital system design using VHDL.**

- **This objective involves a detailed study of the MIPS instruction set architecture (ISA). The ISA specifies the data types, registers, instruction formats, addressing modes, and native operations like arithmetic, bit manipulation, and memory access. A thorough understanding of these elements is essential for designing the processor.**

- **This objective focuses on creating the data path and control units of the processor. The data path includes components such as the ALU (Arithmetic Logic Unit), registers, and buses, while the control unit manages the flow of data within the CPU. The design should be efficient and optimized for the MIPS instruction set.**

- **This step involves converting the processor design into VHDL code. VHDL is a hardware description language used for electronic design automation to describe digital and mixed-signal systems. The implementation should be modular, with distinct VHDL files for different processor components. This modular approach will simplify understanding, testing, and debugging the code.**

A single-cycle MIPS processor

Components of our project

✓ pc.vhd

The program counter for pointing to the next instruction.

✓ instruction_memory.vhd

The block of memory that reads the instructions from a file and saves it into a 128 byte block of memory.

✓ control.vhd

Sets all the flags coming out of the controller appropriately given the 6-bit opcode

✓ registers.vhd

The block of 32 32-bit registers.

✓ sign_extend.vhd

Turns the 16-bit immediate to a 32-bit one by appending zeros.

✓ shifter.vhd

For all your bit shifting needs.

✓ alu_control.vhd

Given the 6-bit opcode and the 6-bit function, this chooses the operation the ALU should perform.

✓ alu.vhd

The ALU that performs either addition, subtraction, and-ing, or-ing, or set-on-less-than operations given the output of the ALU control.

✓ mux.vhd

Simple multiplexer implementation. Can only choose from 2 different inputs for now since 2 are all that's required.

✓ adder.vhd

The only reason why I made this is because the picture had an adder component in it. If the picture had a dragon on it, I would print out an ascii dragon to the wave view.

✓ memory. Vhd

Sample text

✓ main.vhd

The main script that is run. This is what should be selected as the design unit when simulating.

The Problem Solution

Inputs and Outputs

pc

Inputs

- CK
- address_to_load (31:0)

Outputs

- current_address (31:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity pc is
    port(
        ck: in std_logic;
        address_to_load: in std_logic_vector(31 downto 0);
        current_address: out std_logic_vector(31 downto 0)
    );
end pc;

architecture beh of pc is

    signal address: std_logic_vector(31 downto 0):= "00000000000000000000000000000000";

    begin

    process(ck)
        begin
        current_address <= address;
        if ck='0' and ck'event then
            address <= address_to_load;
        end if;
    end process;

end beh;
```

instruction_memory

Inputs

- read_address (31:0)

Outputs

- instruction (31:0)
- last_inst_address (31:0)

```
entity instruction_memory is
    port (
        read_address: in STD_LOGIC_VECTOR (31 downto 0);
        instruction, last_instr_address: out STD_LOGIC_VECTOR (31 downto 0)
    );
end instruction_memory;
architecture behavioral of instruction_memory is

    type mem_array is array(0 to 31) of STD_LOGIC_VECTOR (31 downto 0);
    signal data_mem: mem_array := (
        "00000000000000000000000000000000", -- initialize data memory
        "00000000000000000000000000000000", -- mem 1
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000", -- mem 10
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000", -- mem 20
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000", -- mem 30
        "00000000000000000000000000000000"
    );
    begin
    process
        file file_pointer : text;
        variable line_content : string(1 to 32);
        variable line_num : line;
        variable i: integer := 0;
        variable j : integer := 0;
        variable char : character:='0';
        begin
        file_open(file_pointer, "instructions.txt", READ_MODE);
        while not endfile(file_pointer) loop
            readline(file_pointer,line_num); -- Read a line from the file
            READ(line_num,line_content); -- Turn the string into a line (looks wierd right? Than
            for j in 1 to 32 loop
                char := line_content(j);
                if(char = '0') then
                    data_mem(i)(32-j) <= '0';
                else
                    data_mem(i)(32-j) <= '1';
                end if;
            end loop;
            i := i + 1;
        end loop;
        if i > 0 then
            last_instr_address <= std_logic_vector(to_unsigned((i-1)*4, last_instr_address'lengt
        else
            last_instr_address <= "00000000000000000000000000000000";
        end if;

        file_close(file_pointer);
        wait;
    end process;
    instruction <= data_mem(to_integer(unsigned(read_address(31 downto 2))));

end behavioral;
```

control

Inputs

- opcode

Outputs

- alu_op(1:0)
- alu_src
- branch
- jump
- mem_read
- mem_to_reg
- reg_dest
- reg_write

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity control is
    port (
        opcode: in std_logic_vector(5 downto 0);
        reg_dest,jump, branch, mem_read, mem_to_reg, mem_write, alu_src, reg_write: out std_logic;
        alu_op: out std_logic_vector(1 downto 0)
    );
end control;

architecture beh of control is
    begin

    -- The consequences of vhdl syntax
    --                  R-types                 addi                beq                 bne                 jump
    reg_dest <=     '1' when opcode="000000"  else '0' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '0' when
    jump <=         '0' when opcode="000000"  else '0' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '1' when opcode="000010"  else '0' when
    branch <=       '0' when opcode="000000"  else '0' when opcode="001000"  else '1' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '0' when
    mem_read <=     '0' when opcode="000000"  else '0' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '1' when
    mem_to_reg <=   '0' when opcode="000000"  else '0' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '1' when
    mem_write <=    '0' when opcode="000000"  else '0' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '0' when
    alu_src <=      '0' when opcode="000000"  else '1' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '1' when
    reg_write <=    '1' when opcode="000000"  else '1' when opcode="001000"  else '0' when opcode="000100"  else '0' when opcode="000101"  else '0' when opcode="000010"  else '1' when
    alu_op <=       "10" when opcode="000000" else "00" when opcode="001000" else "01" when opcode="000100" else "11" when opcode="000101" else "00" when opcode="000010" else "00" when

end beh;
```

register

Inputs

- ck
- reg_write (4:0)
- read_reg_1 (31:0)
- read_reg_2 (31:0)
- write_data (4:0)

Outputs

- read_data_1 (31:0)
- read_data_2 (31:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity registers is
    port (
        ck: in std_logic;
        reg_write: in std_logic;
        read_reg_1, read_reg_2, write_reg: in std_logic_vector(4 downto 0);
        write_data: in std_logic_vector(31 downto 0);
        read_data_1, read_data_2: out std_logic_vector(31 downto 0)
    );
end registers;

architecture beh of registers is

    -- 128 byte register memory (32 rows * 4 bytes/row)
    type mem_array is array(0 to 31) of STD_LOGIC_VECTOR (31 downto 0);
    signal reg_mem: mem_array := (
        "00000000000000000000000000000000", -- $zero
        "00000000000000000000000000000000", -- mem 1
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000", -- test add
        "00000000000000000000000000000000", -- test add
        "00000000000000000000000000000000", -- mem 10
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000", -- mem 20
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000",
        "00000000000000000000000000000000", -- mem 30
        "00000000000000000000000000000000"
    );

    begin

    read_data_1 <= reg_mem(to_integer(unsigned(read_reg_1)));
    read_data_2 <= reg_mem(to_integer(unsigned(read_reg_2)));

    process(ck)
        begin
        if ck='0' and ck'event and reg_write='1' then
            -- write to reg. mem. when the reg_write flag is set and on a fallin
            reg_mem(to_integer(unsigned(write_reg))) <= write_data;
        end if;
    end process;

end beh;
```

sign_extend

Inputs

- x

(15:0)

Outputs

- y

(31:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity sign_extend is
    port (
        x: in std_logic_vector(15 downto 0);
        y: out std_logic_vector(31 downto 0)
    );
end sign_extend;

architecture beh of sign_extend is
    begin
    y <= std_logic_vector(resize(signed(x), y'length));
end beh;
```

shifter

Inputs

- x

(n1-1:0)

Outputs

- y

(n2-1:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity shifter is
    generic (n1: natural:= 32; n2: natural:= 32; k: natural:= 2);
    port (
        x: in std_logic_vector(n1-1 downto 0);
        y: out std_logic_vector(n2-1 downto 0)
    );
end entity;

architecture beh of shifter is
    signal temp: std_logic_vector(n2-1 downto 0);

    begin
    temp <= std_logic_vector(resize(unsigned(x), n2)); -- This is required if you
    y <= std_logic_vector(shift_left(signed(temp), k));
end beh;
```

alu_control

Inputs

- alu_op (1:0)
- funct (5:0)

Outputs

- alu_control_fuct (3:0)

```
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity alu_control is
    port (
        funct: in std_logic_vector(5 downto 0);
        alu_op: in std_logic_vector(1 downto 0);
        alu_control_fuct: out std_logic_vector(3 downto 0)
    );
end alu_control;

architecture beh of alu_control is
    signal and_op: std_logic_vector(3 downto 0):= "0000";
    signal or_op: std_logic_vector(3 downto 0):= "0001";
    signal add: std_logic_vector(3 downto 0):= "0010";
    signal subtract_not_equal: std_logic_vector(3 downto 0):= "0011";
    signal subtract: std_logic_vector(3 downto 0):= "0110";
    signal set_on_less_than: std_logic_vector(3 downto 0):= "0111";

    begin

    alu_control_fuct <= add when(alu_op="00" or (alu_op="10" and funct="100000")) else
                        subtract when(alu_op="01" or (alu_op="10" and funct="100010")) else
                        subtract_not_equal when(alu_op="11") else
                        and_op when(alu_op="10" and funct="100100") else
                        or_op when(alu_op="10" and funct="100101") else
                        set_on_less_than when(alu_op="10" and funct="101010") else
                        "0000";


end beh;
```

alu

Inputs

- in_1(31:0)
- in_2(31:0)
- alu_control_fuct (3:0)

Outputs

- zero
- alu_results (31:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity alu is
    port (
        in_1, in_2: std_logic_vector(31 downto 0);
        alu_control_fuct: in std_logic_vector(3 downto 0);
        zero: out std_logic;
        alu_result: out std_logic_vector(31 downto 0)
    );
end alu;

architecture beh of alu is
    signal and_op: std_logic_vector(3 downto 0):= "0000";
    signal or_op: std_logic_vector(3 downto 0):= "0001";
    signal add: std_logic_vector(3 downto 0):= "0010";
    signal subtract_not_equal: std_logic_vector(3 downto 0):= "0011";
    signal subtract: std_logic_vector(3 downto 0):= "0110";
    signal set_on_less_than: std_logic_vector(3 downto 0):= "0111";

    begin

    alu_result <=   in_1 + in_2 when(alu_control_fuct=add) else
                    in_1 - in_2 when(alu_control_fuct=subtract or alu_control_fuct=subtract_not_equal) else
                    in_1 and in_2 when(alu_control_fuct=and_op) else
                    in_1 or in_2 when(alu_control_fuct=or_op) else
                    "00000000000000000000000000000001" when(alu_control_fuct=set_on_less_than and in_1 < in_2) else
                    "00000000000000000000000000000000" when(alu_control_fuct=set_on_less_than);

    zero <= '1' when(in_1/=in_2 and alu_control_fuct=subtract_not_equal) else
            '0' when(in_1=in_2 and alu_control_fuct=subtract_not_equal) else
            '1' when(in_1=in_2) else
            '0';

end beh;
```

adder

Inputs

- x (31:0)
- y (31:0)

Outputs

- z (31:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adder is
    port (
        x,y: in std_logic_vector(31 downto 0);
        z: out std_logic_vector(31 downto 0)
    );
end entity;

architecture beh of adder is
    begin
    z <= x+y;
end beh;
```

mux

Inputs

- x (n-1:0)
- y (n-1:0)
- s

Outputs

- z (n-1:0)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity mux is
    generic (n: natural:= 1);
    port (
        x,y: in std_logic_vector(n-1 downto 0);
        s: in std_logic;
        z: out std_logic_vector(n-1 downto 0)
    );
end mux;

architecture beh of mux is
    begin
    z <= x when (s='0') else y;
end beh;
```

memory

Inputs

- address (31:0)
- write_data (31:0)
- MemRead
- MemWrite
- ck

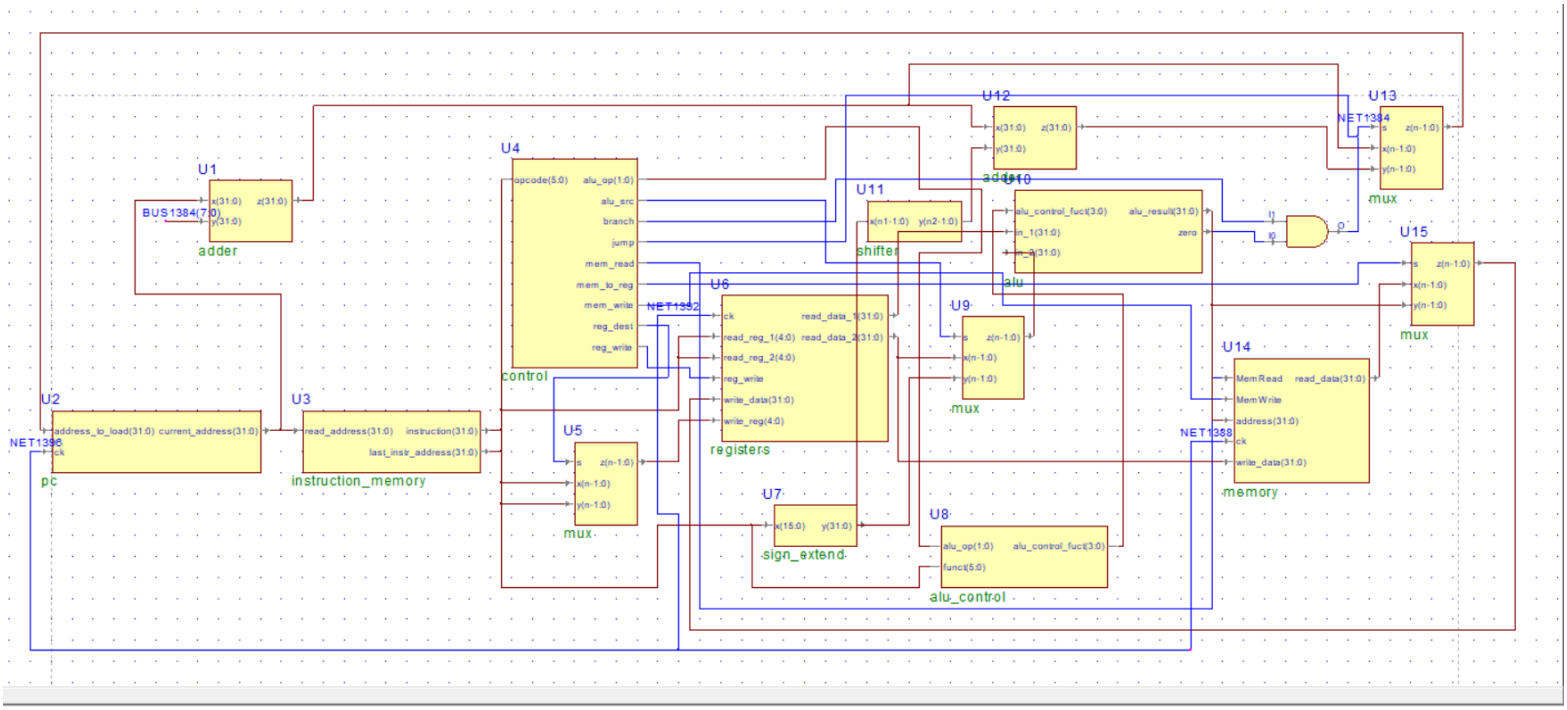
Outputs

- read_data (31:0)

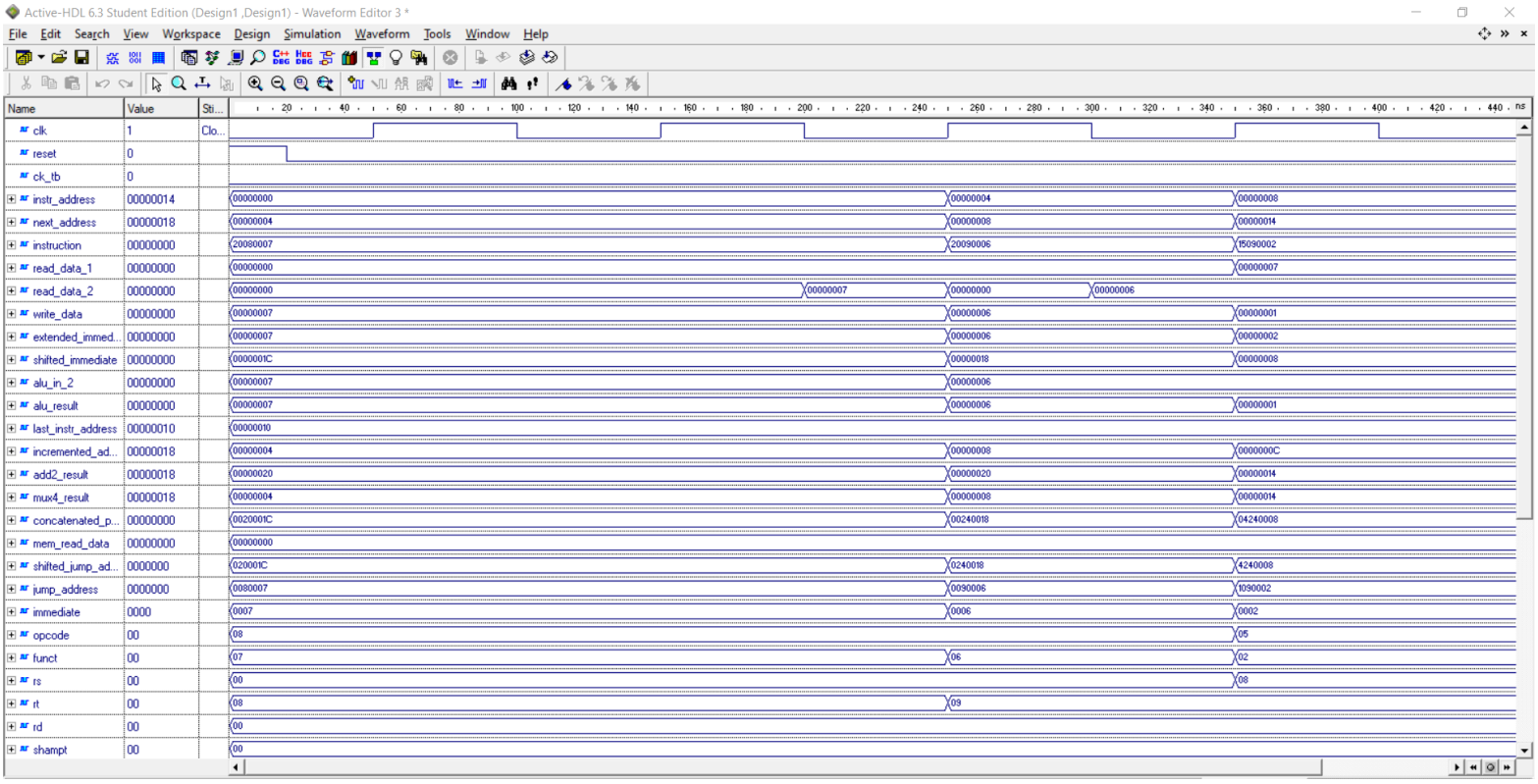
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity memory is
    port (
        address, write_data: in STD_LOGIC_VECTOR (31 downto 0);
        MemWrite, MemRead,ck: in STD_LOGIC;
        read_data: out STD_LOGIC_VECTOR (31 downto 0)
    );
end memory;


architecture behavioral of memory is

type mem_array is array(0 to 31) of STD_LOGIC_VECTOR (31 downto 0);

signal data_mem: mem_array := (
    X"00000000", -- initialize data memory
    X"00000000", -- mem 1
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000", -- mem 10
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000", -- mem 20
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000",
    X"00000000", -- mem 30
    X"00000000");

begin

read_data <= data_mem(conv_integer(address(6 downto 2))) when MemRead = '1' else X"00000000";

mem_process: process(address, write_data,ck)
begin
    if ck = '0' and ck'event then
        if (MemWrite = '1') then
            data_mem(conv_integer(address(6 downto 2))) <= write_data;
        end if;
    end if;
end process mem_process;

end behavioral;
```

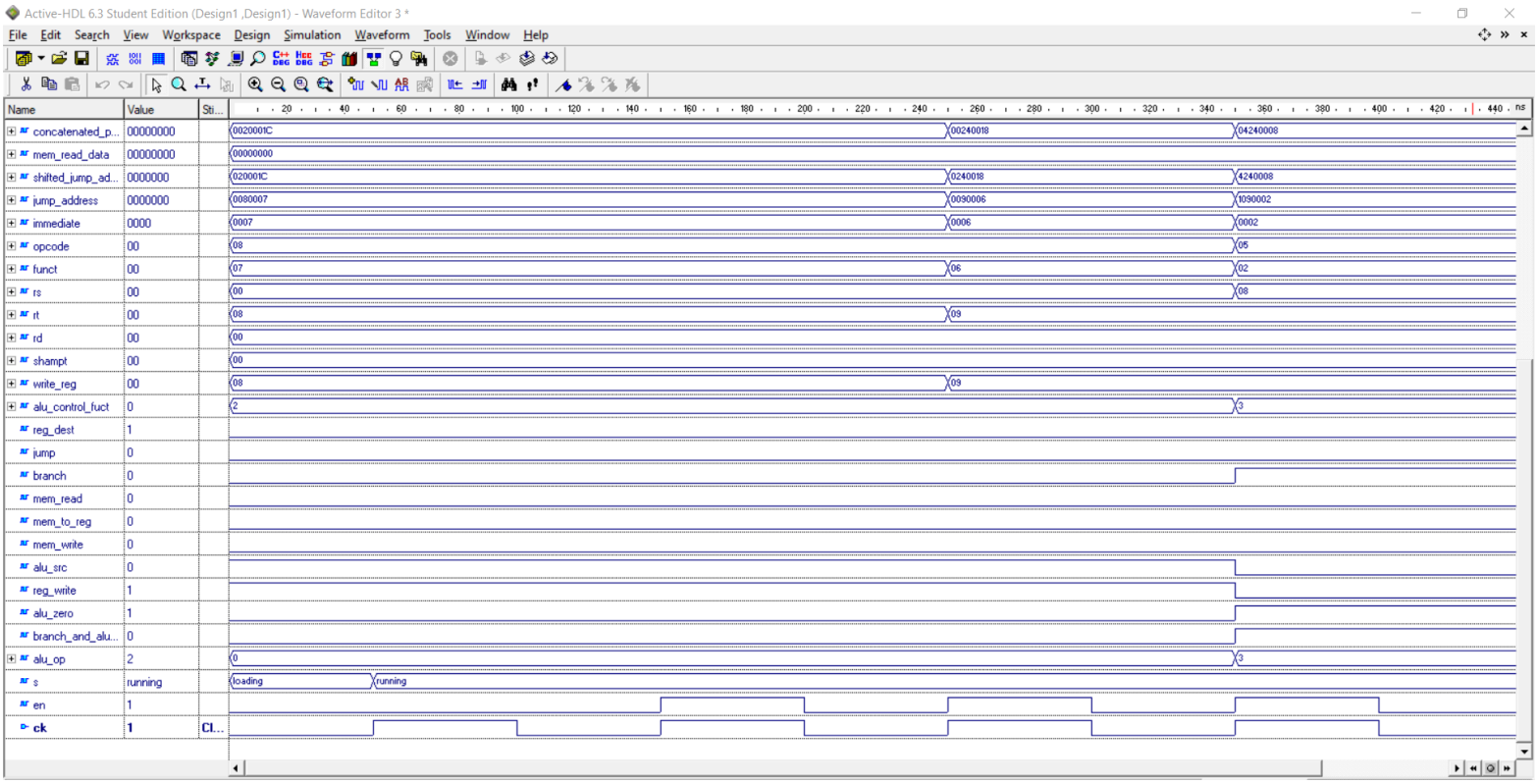
Implementation

Give the Schematic diagram using Active HDL



Simulate result





References

1	https://www.youtube.com/watch?v=kTIiAORIQvU
2	https://www.youtube.com/watch?v=o38kMnM7hGg
3	https://www.youtube.com/watch?v=J-e2sFPRI7c
4	https://www.youtube.com/watch?v=3tK_qdOb434&list=PLW7Cvy3HywwwiTivCN8jm2yDn9N5Eyxza&index=31
5	https://www.youtube.com/watch?v=e-n8wja4nJA&list=PLW7Cvy3HywwwiTivCN8jm2yDn9N5Eyxza&index=32
6	https://www.youtube.com/watch?v=LrdEmHoE5U8&list=PLW7Cvy3HywwwiTivCN8jm2yDn9N5Eyxza&index=35
7	https://www.youtube.com/watch?v=FfWMtUybkqI&list=PLW7Cvy3HywwwiTivCN8jm2yDn9N5Eyxza&index=37

No	Name	Task
1	Matilda Ashraf Malak	ALU , alu control
2	Mariam Osman Hussein	Registers , Sign Extend
3	Mariam Farouk Slama	PC , instruction memory
4	Norhan Mohamed Elsayed	shifter , memory
5	Hager Mohamed Ali	control , adder , mux

we worked together on main


Thank
You