Lab 8 CPU Report

Ismail Akram
12/20/2021
CSc34200 & 34300
Prof Gertner

Contents

Objective

The objective of this final lab is to create a simplified CPU that runs the program which computes the sum of five integers using MIPS instructions I have designed in previous labs: LW, ADD, SW.

We will be using the following components: Data Memory, Instruction Memory, 3-ported Register File, IR-Instruction Register, PC- Program Counter, 3 ADD/SUB units, Signed and Zero extension from 16 bits to 32 bits, 2:1 MUX.

Screenshots

Components

CPU, 2:1 MUX, Sign Ext

This is the main vital code that connects all the subsequent components together. Includes multiplexer and sign ext and zero ext. It is the heart of this final lab.

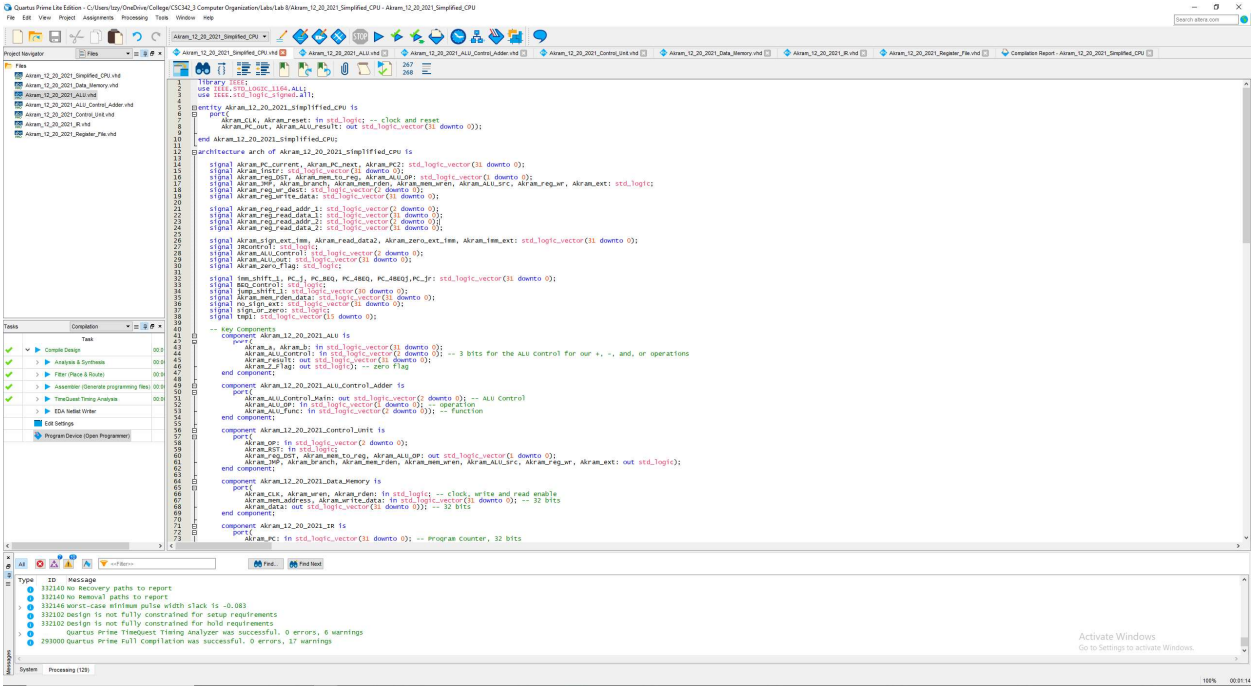


Fig 1: Akram_12_20_2021_Simplified_CPU.vhdl (Code)

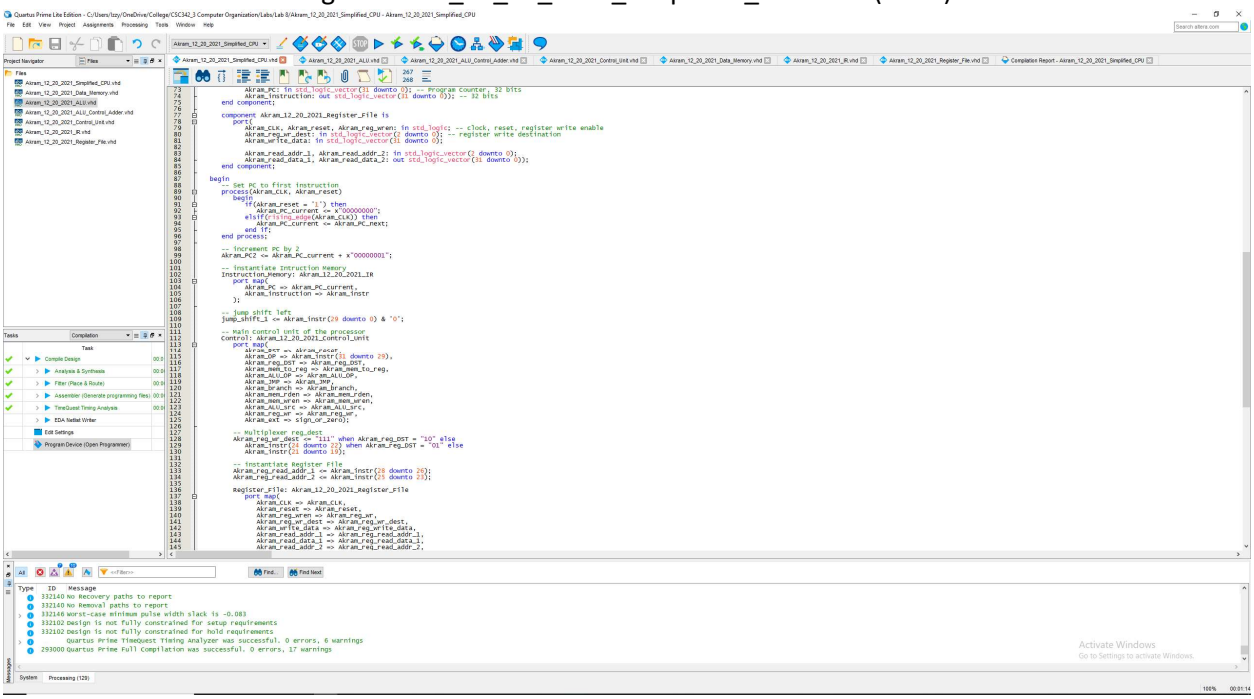


Fig 1a: Akram_12_20_2021_Simplified_CPU.vhdl (code) cont.

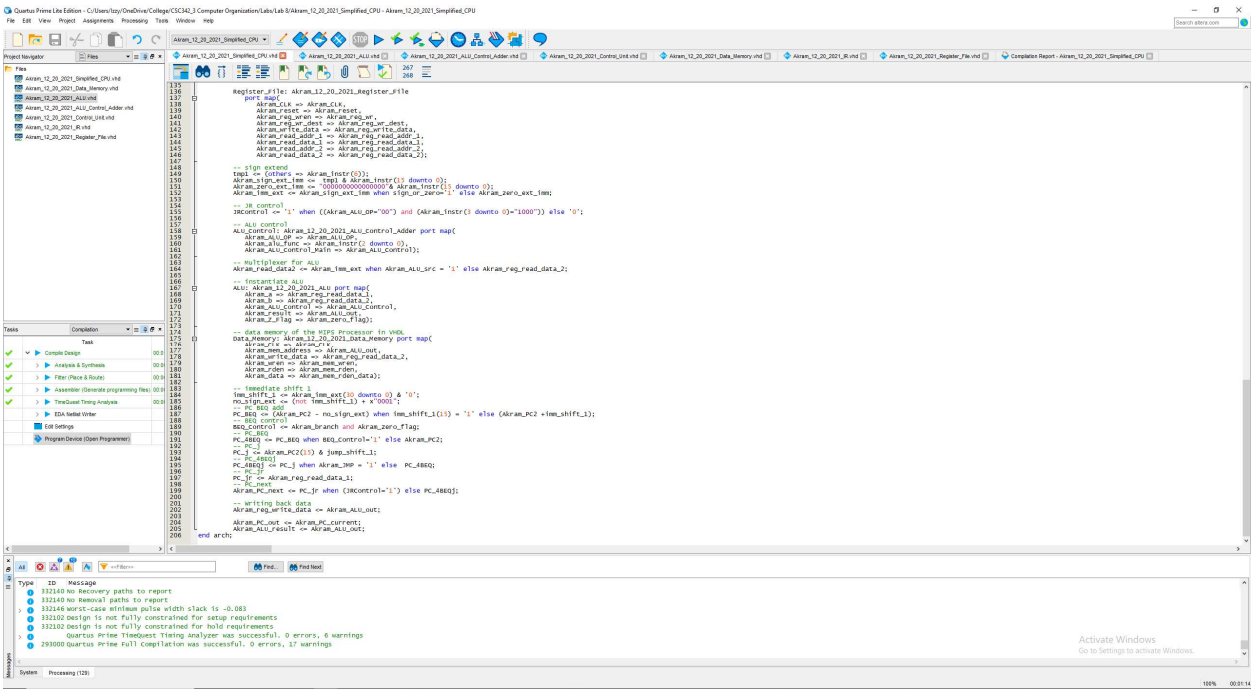


Fig 1b: Akram_12_20_2021_Simplified_CPU.vhdl (Code) cont.

ALU

Arithmetic Logic Unit, responsible for arithmetic operations; self-explanatory. This is the vital main code in our previous lab (7).

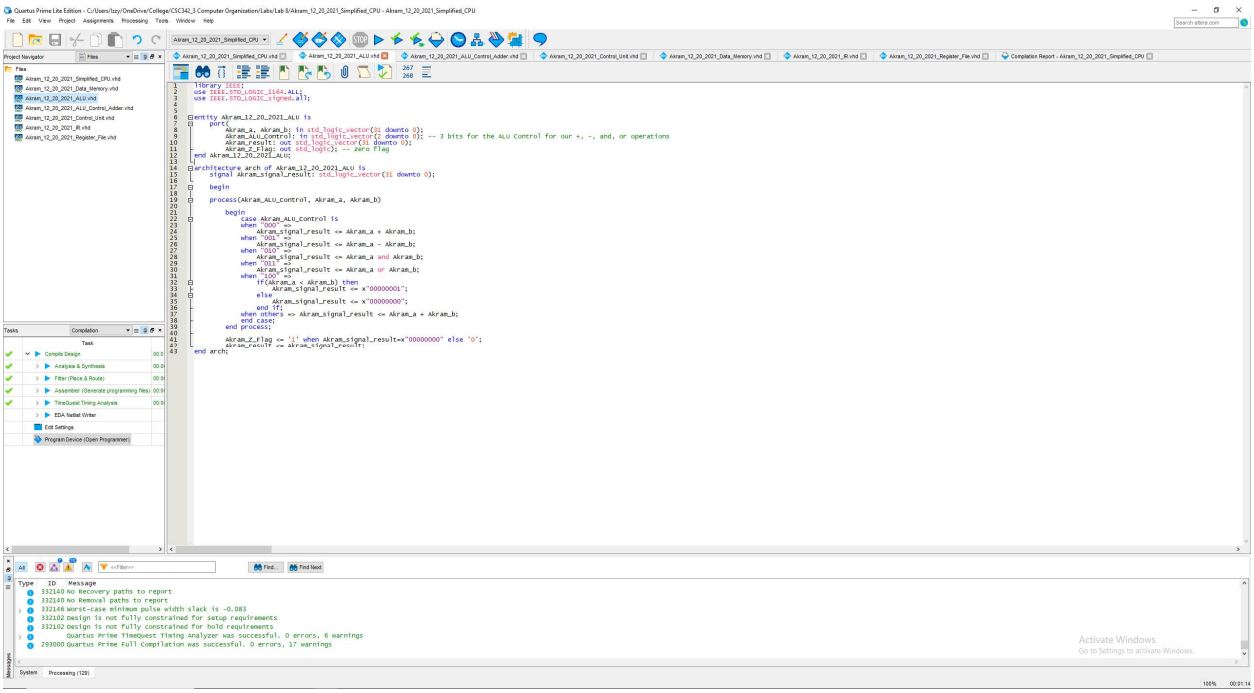


Fig 2: Akram_12_20_2021_ALU.vhdl (Code)

ALU Control Adder

For ALU operation and function. We use this as a component in the CPU.

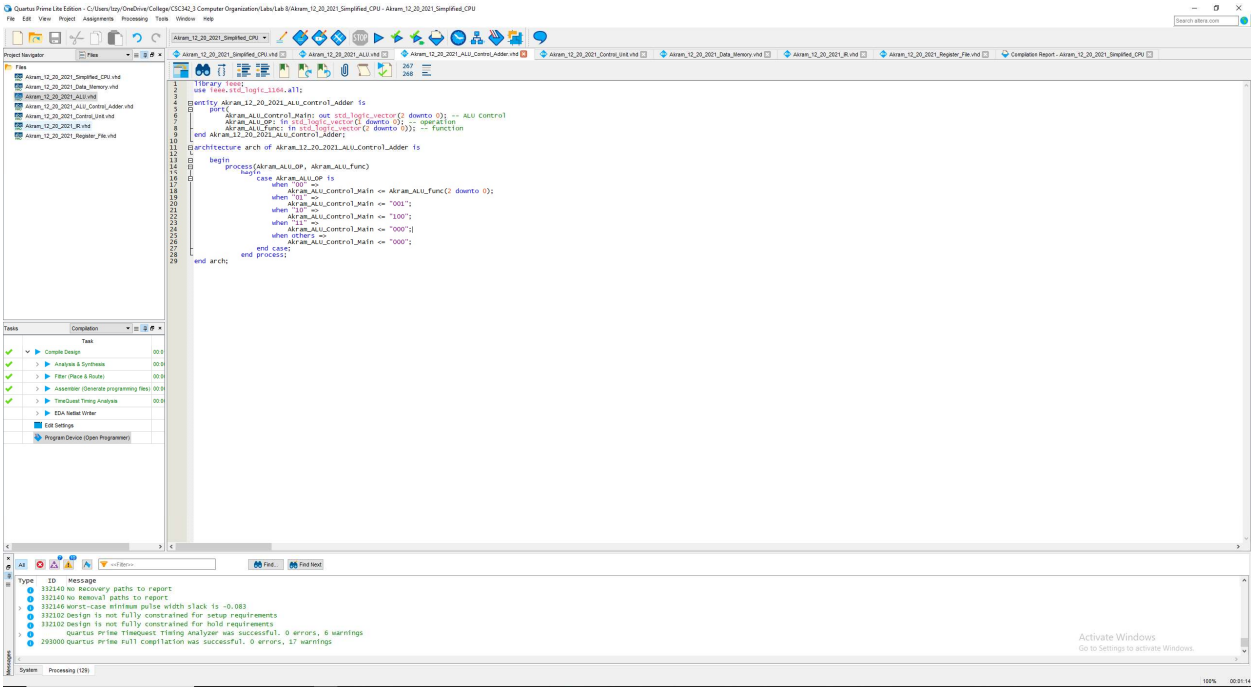


Fig 3: Akram_12_20_2021_ALU_Control_Adder.vhdl (Code)

Control Unit

Generates all control signals: Akram_JMP, Akram_branch, Akram_mem_rden, Akram_mem_wren, Akram_ALU_src, Akram_reg_wr, Akram_ext. It takes the OP (OP code) and the func values from the IR and assigns them appropriate values to each of the control signals.

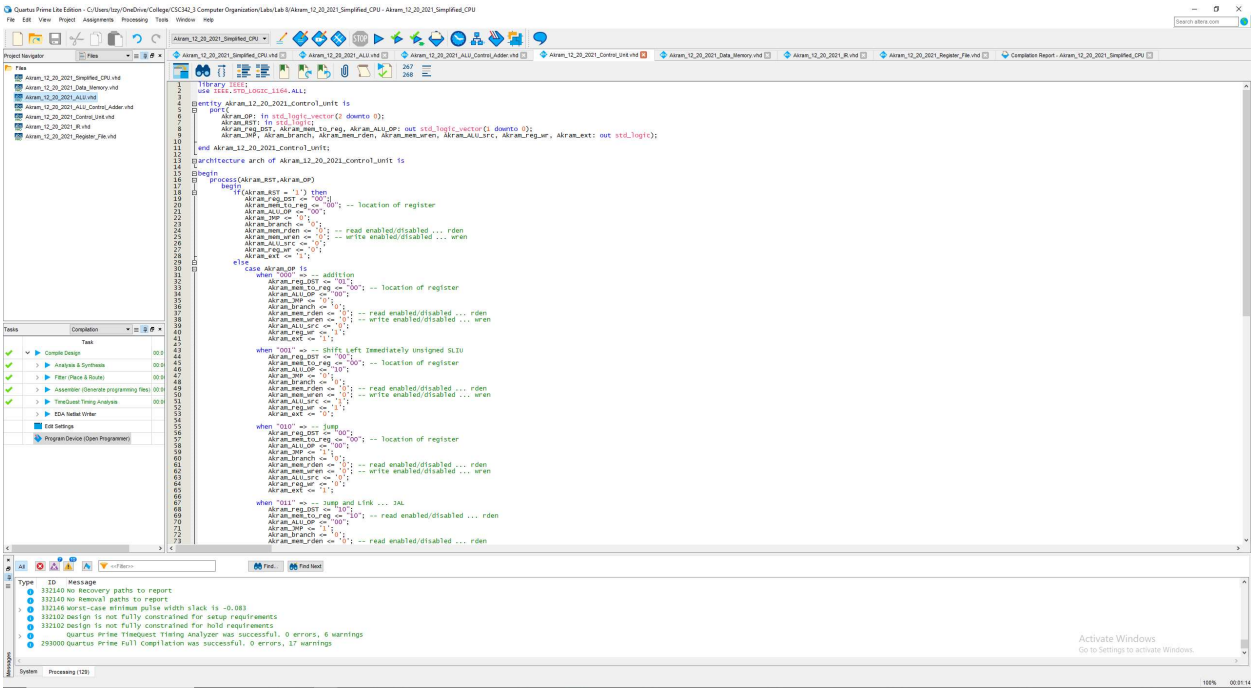


Fig 4: Akram_12_20_2021_Control_Unit.vhdl (Code)

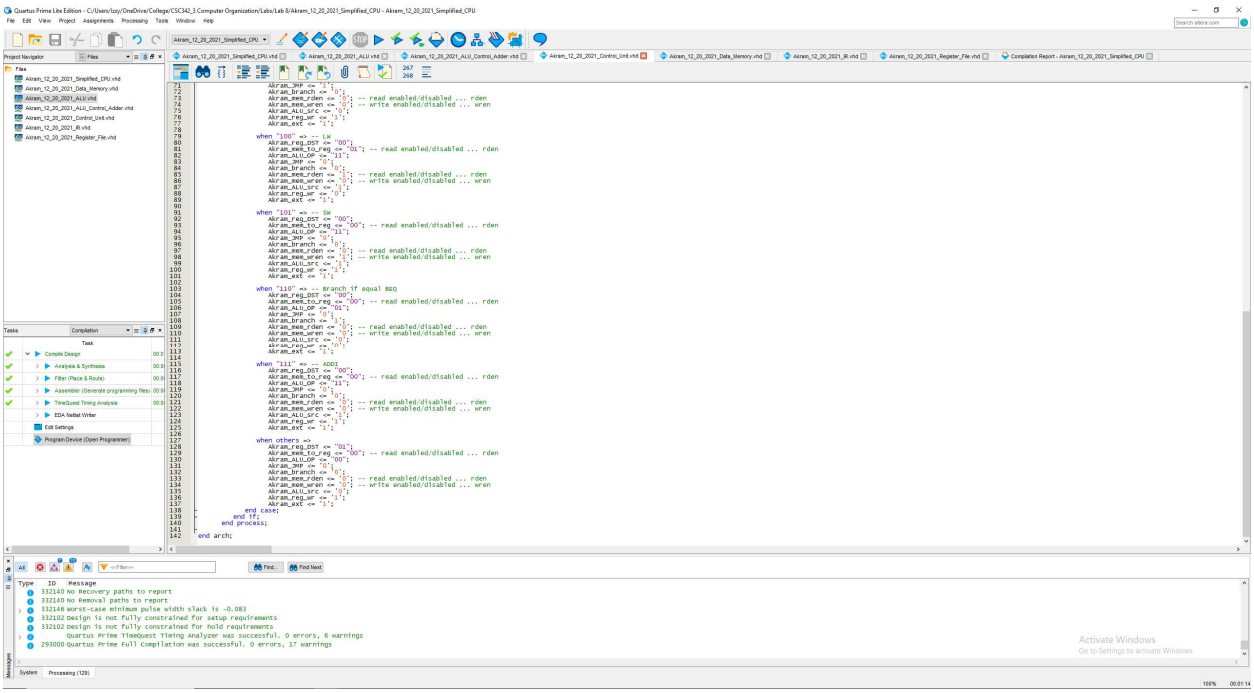
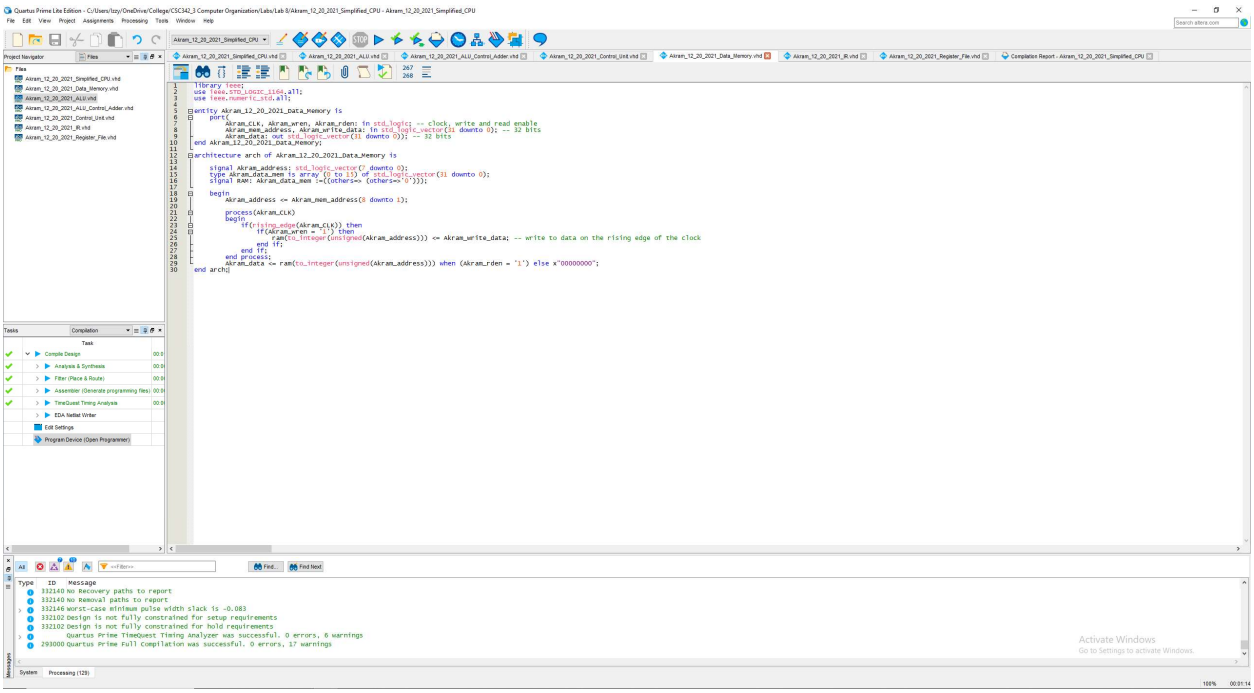


Fig 4a: Akram_12_20_2021_Control_Unit.vhdl (Code) cont.

Data Memory

Reads and writes data in memory. The memory size of the component is 64 bit bytes and each word size is 32 bits. 5 inputs and 1 output. Output is 32-bit memory value that is read.

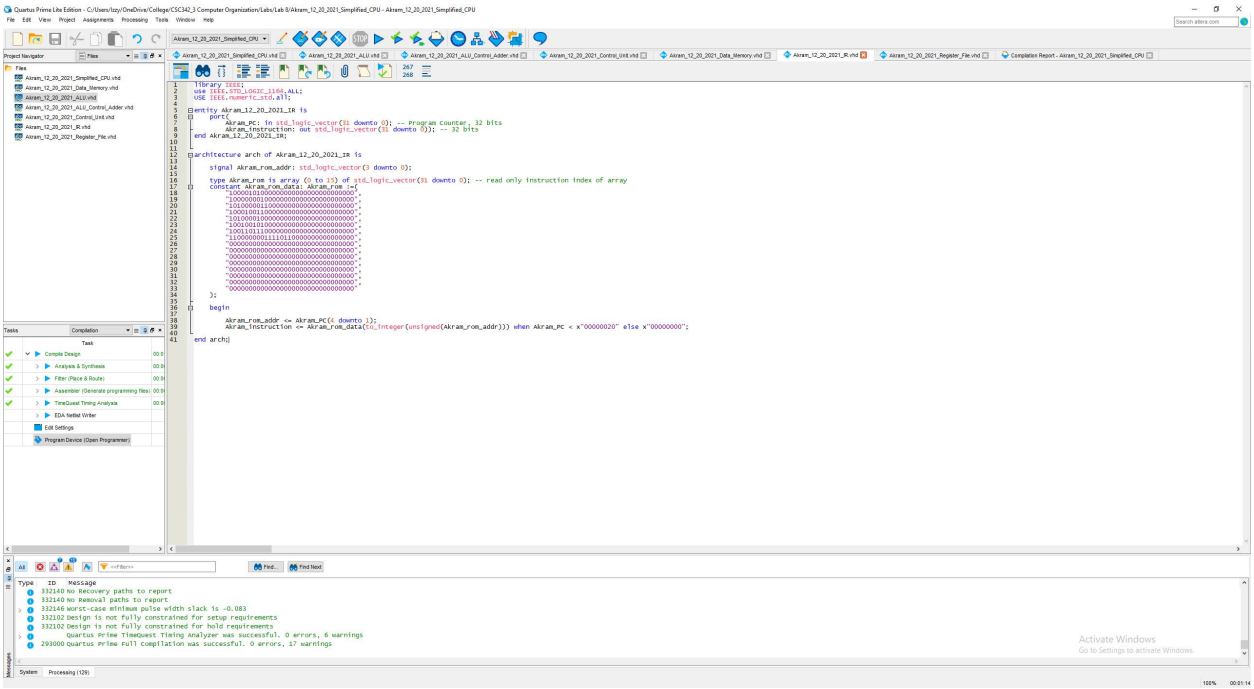


```
library ieee;
use ieee.STD_LOGIC_1164.all;
use ieee.numeric_std.all;

entity Akram_12_20_2021_Data_Memory is
    port(
        Akram_CLK, Akram_wren, Akram_rden: in std_logic; -- clock, write and read enable
        Akram_mem_address, Akram_write_data: in std_logic_vector(31 downto 0); -- 32 bits
        Akram_data: out std_logic_vector(31 downto 0)); -- 32 bits
end Akram_12_20_2021_Data_Memory;

architecture arch of Akram_12_20_2021_Data_Memory is

    signal Akram_address: std_logic_vector(7 downto 0);
    type Akram_data_mem is array (0 to 13) of std_logic_vector(31 downto 0);
    signal RAM: Akram_data_mem :=(others=> (others=>'0'));

    begin
        Akram_address <= Akram_mem_address(8 downto 1);

        process(Akram_CLK)
        begin
            if(rising_edge(Akram_CLK)) then
                if(Akram_wren = '1') then
                    ram(to_integer(unsigned(Akram_address))) <= Akram_write_data; -- write to data on the rising edge of the clock
                end if;
            end if;
        end process;
        Akram_data <= ram(to_integer(unsigned(Akram_address))) when (Akram_rden = '1') else x"00000000";
end arch;
```

Fig 5: Akram_12_20_2021_Data_Memory.vhdl (Code)

Instruction Register

A register that stores the instruction that is currently executed. Being a register means that data gets read and stored only during the RISING EDGE of the CLK (clock). It gets its current instruction from the IR or Instruction Memory, for out simplified CPU, we are only concerned with storage.



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE IEEE.numeric_std.all;

entity Akram_12_20_2021_IR is
    port(
        Akram_PC: in std_logic_vector(31 downto 0); -- Program Counter, 32 bits
        Akram_Instruction: out std_logic_vector(31 downto 0)); -- 32 bits
end Akram_12_20_2021_IR;

architecture arch of Akram_12_20_2021_IR is

    signal Akram_rom_addr: std_logic_vector(3 downto 0);

    type Akram_rom is array (0 to 15) of std_logic_vector(31 downto 0); -- read only instruction index of array
    constant Akram_rom_data: Akram_rom :=(
            "10000001000000000000000000000000",
            "10100000110000000000000000000000",
            "10001001100000000000000000000000",
            "10100001000000000000000000000000",
            "10010010100000000000000000000000",
            "10011011100000000000000000000000",
            "11000000111101100000000000000000",
            "00000000000000000000000000000000",
            "00000000000000000000000000000000",
            "00000000000000000000000000000000",
            "00000000000000000000000000000000",
            "00000000000000000000000000000000",
            "00000000000000000000000000000000",
            "00000000000000000000000000000000"
        );
    begin

        Akram_rom_addr <= Akram_PC(4 downto 1);
        Akram_Instruction <= Akram_rom_data(to_integer(unsigned(Akram_rom_addr))) when Akram_PC < x"00000020" else x"00000000";

end arch;
```

Fig 6: Akram_12_20_2021_IR.vdhl (Code)

Register File

32 32bit Register File. It has 3-port RAM to read and write data to registers. This was our previous lab (6).

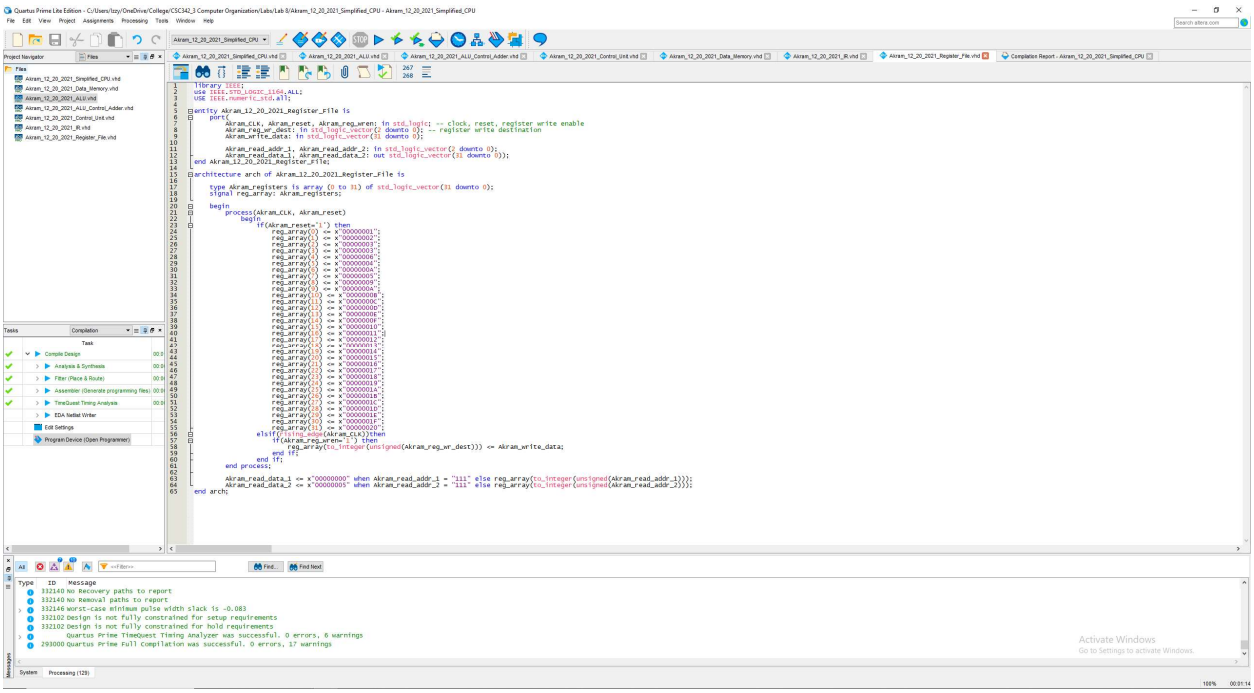


Fig 7: Akram_12_20_2021_Register_File.vhdl (Code)

Simulations

Ultimate test of design: inputting 5 integers and adding them. Note that throughout these screenshots, we see the gradual addition taking place of 1+2+3+4+5 = 15

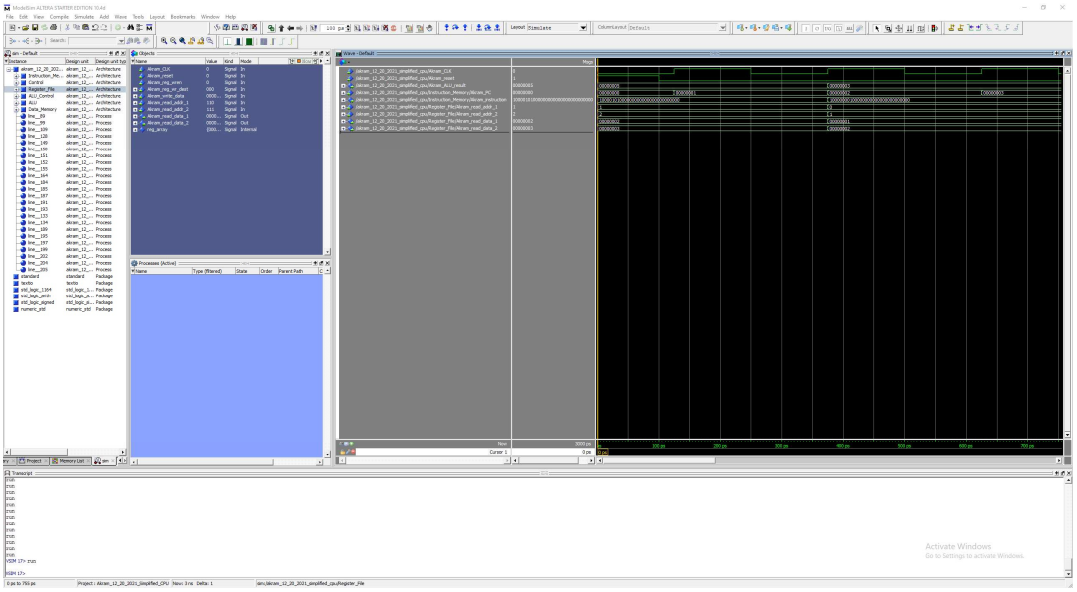


Fig 8: ModelSim of Akram_12_20_2021_Simplified_CPU

Akram_CLK: Falling 250 ps

Akram_reset: 0

@ 125 ps: Akram_PC = 1, and we have Akram_read_data_1 = 2 and Akram_read_data_2 = 3

@ 375 ps: Akram_ALU_result = 3, Akram_PC = 2

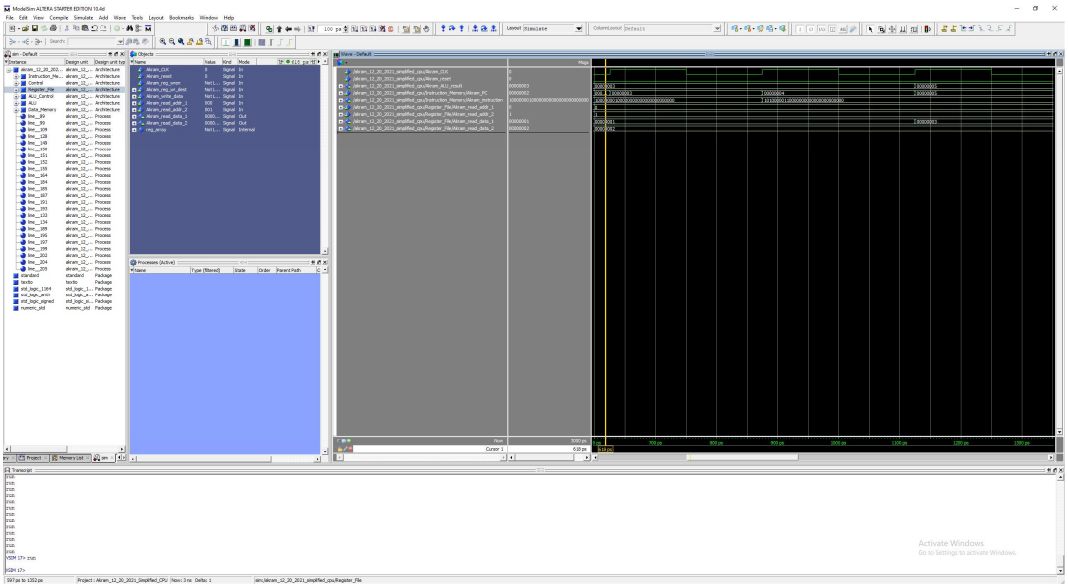


Fig 8a: ModelSim of Akram_12_20_2021_Simplified_CPU

Akram_reset: 1

@ 700 ps: Akram_PC = 3, and we have Akram_read_data_1 = 1 and Akram_read_data_2 = 2

@ 1125 ps: Akram_ALU_result = 5, Akram_PC = 5

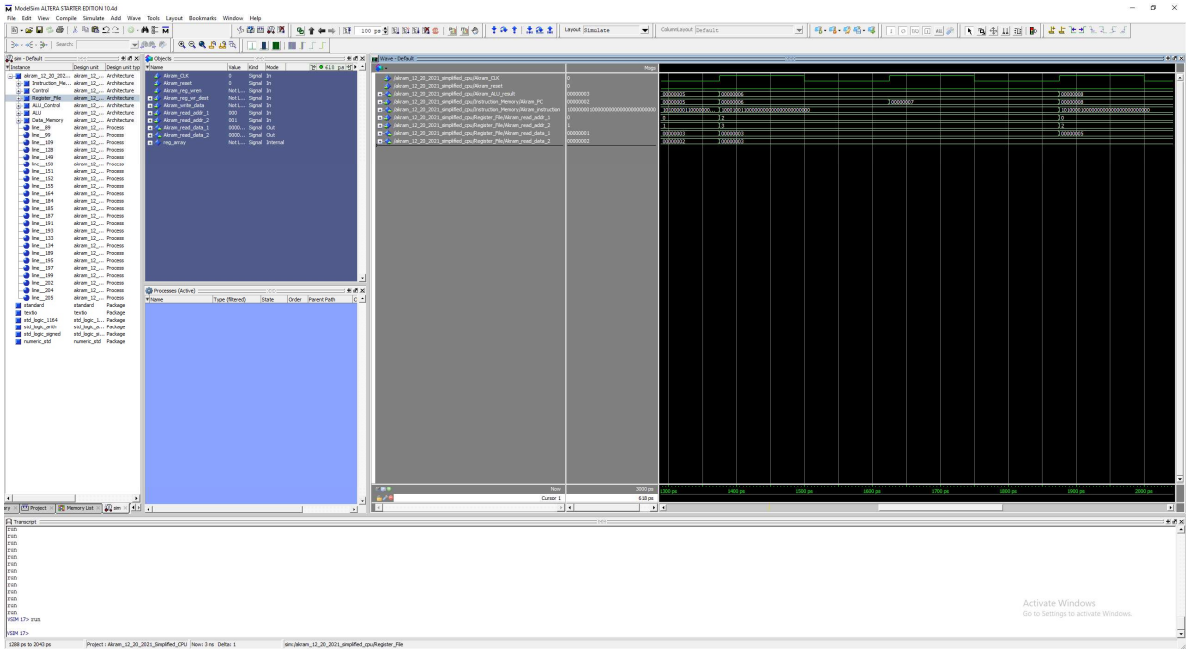


Fig 8b: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 1300 ps: Akram_PC = 5, and we have Akram_read_data_1 = 3 and Akram_read_data_2 = 2

@ 1375 ps: Akram_ALU_result = 6, Akram_PC = 6, and Akram_read_data_1 = 3 and Akram_read_data_2 = 3

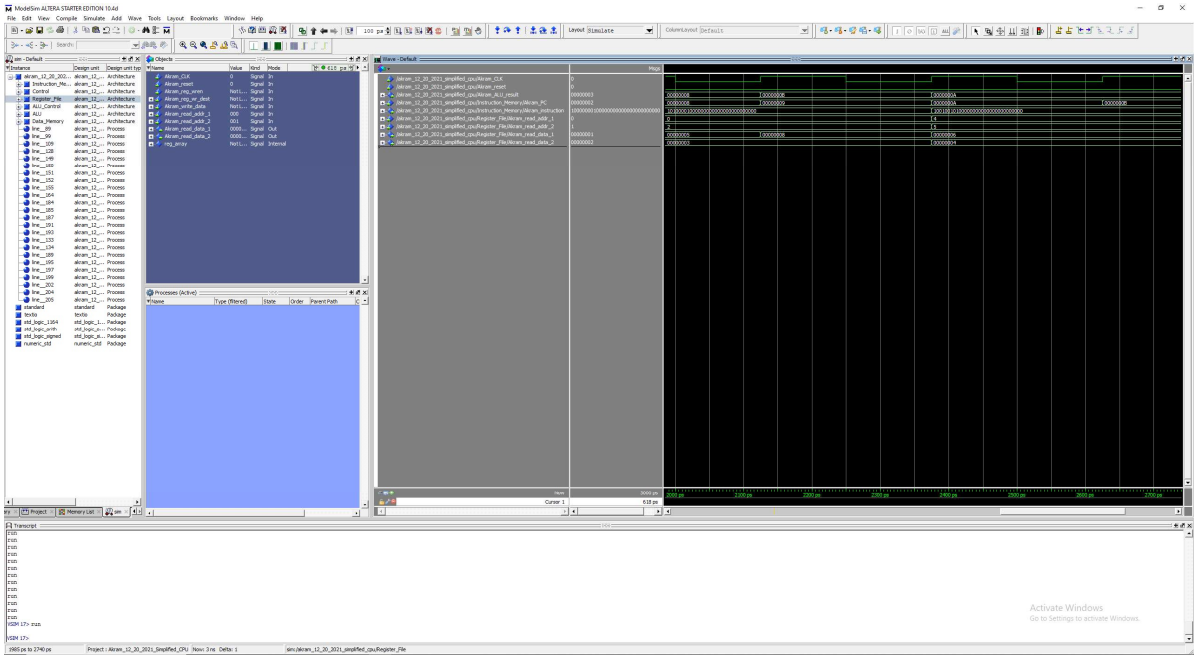


Fig 8c: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 2000 ps: Akram_PC = 8, and we have Akram_read_data_1 = 5 and Akram_read_data_2 = 3

@ 2375 ps: Akram_ALU_result = 10, Akram_PC = 10, and Akram_read_data_1 = 6 and Akram_read_data_2 = 4

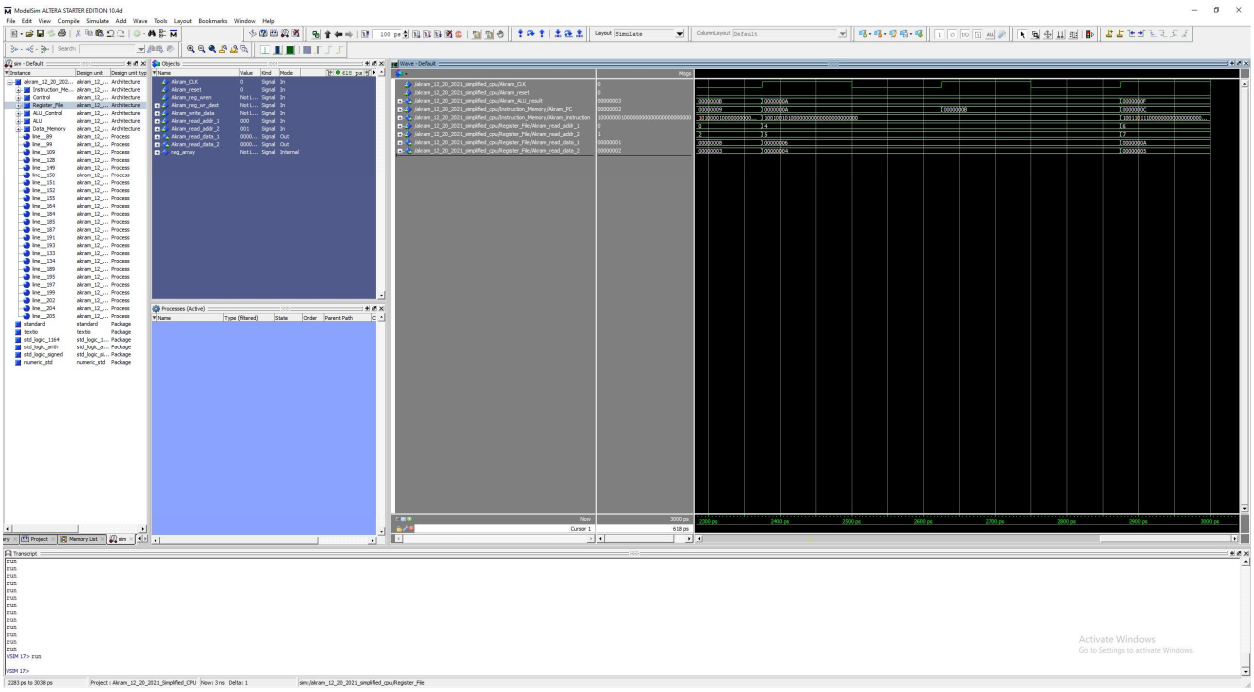


Fig 8d: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 2875 ps: Akram_PC = 12, and we have Akram_read_data_1 = 10 and Akram_read_data_2 = 5

@ 2875+ ps: Akram_ALU_result = 15, Akram_PC = 12, and Akram_read_data_1 = 10 and Akram_read_data_2 = 5

Proof that PC values are not forced (note that we still get the same final sum of 15 in the end):

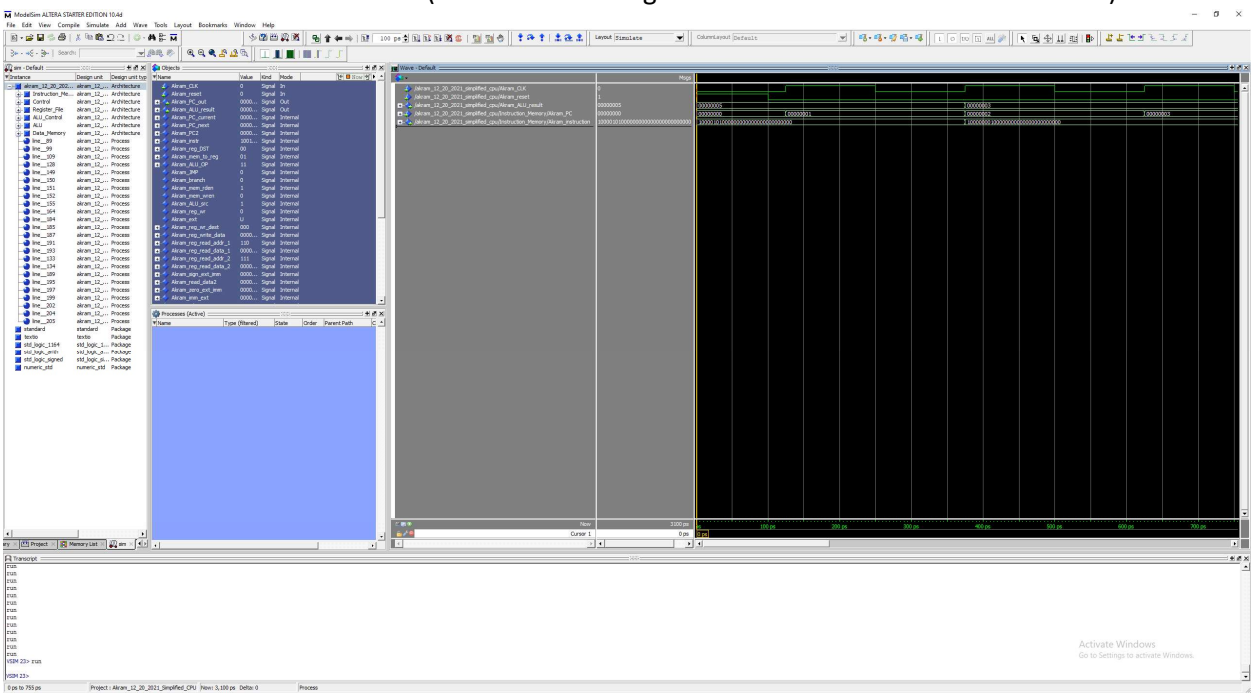


Fig 9: (Same as Fig 8)

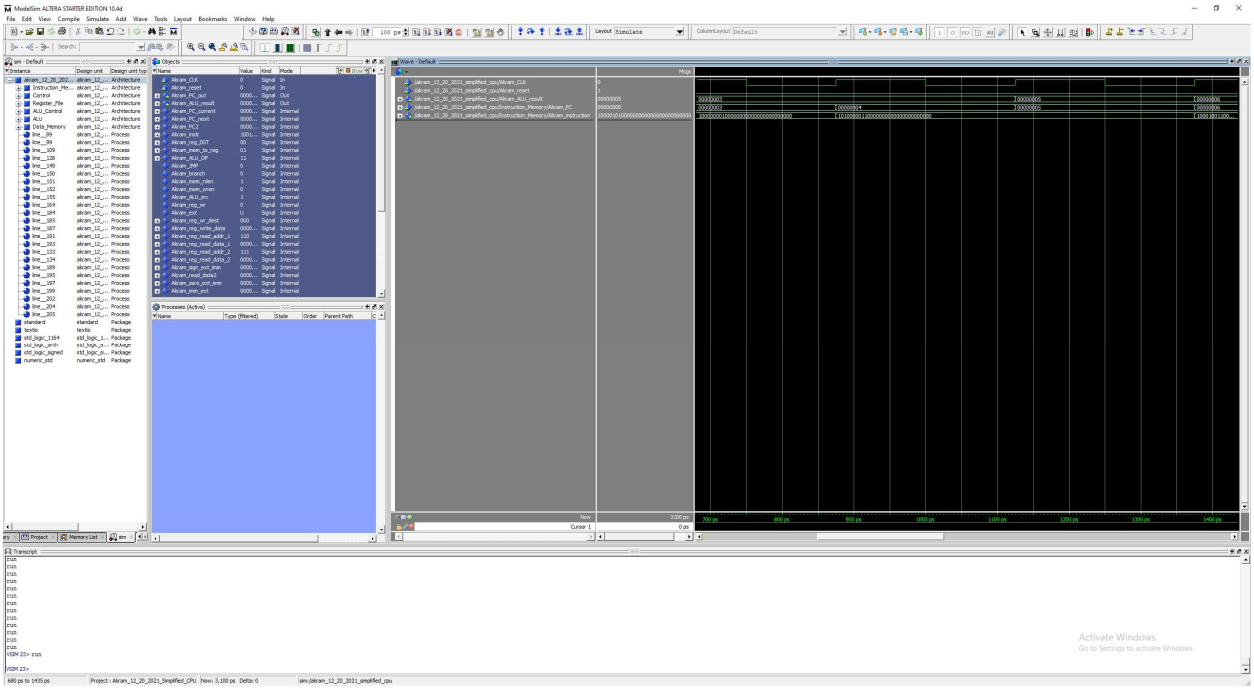


Fig 9a: (Same as Fig 8a)

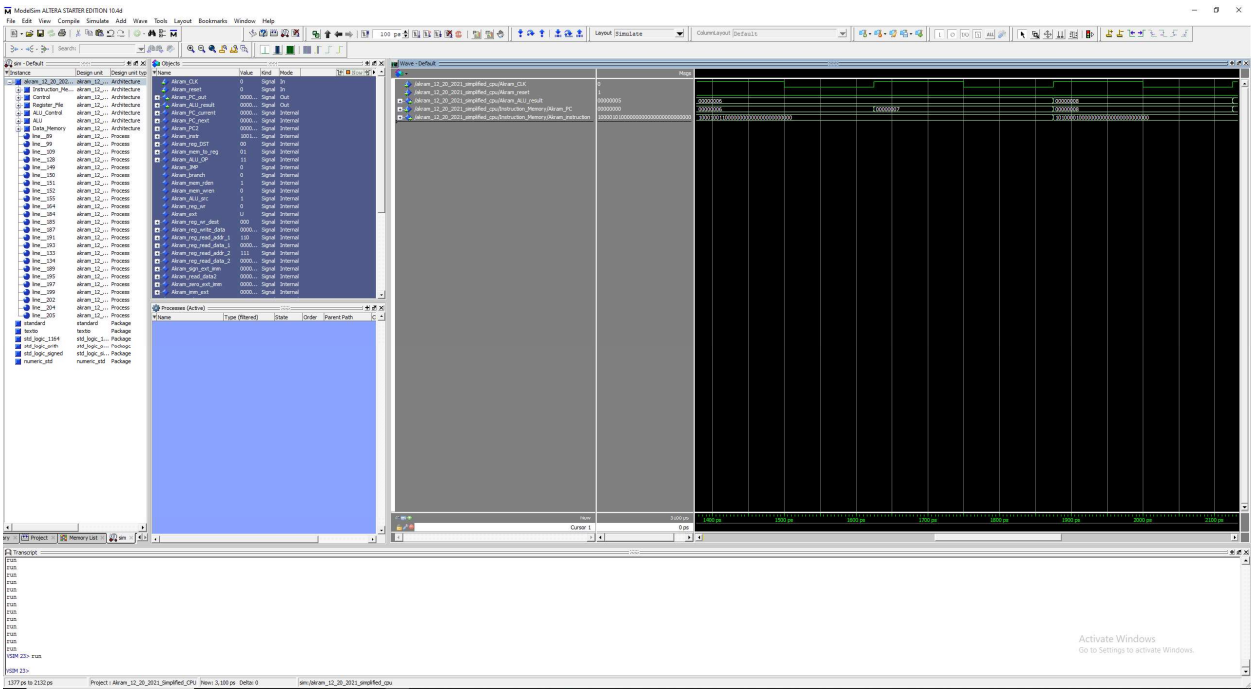


Fig 9b: (Same as Fig 8b)

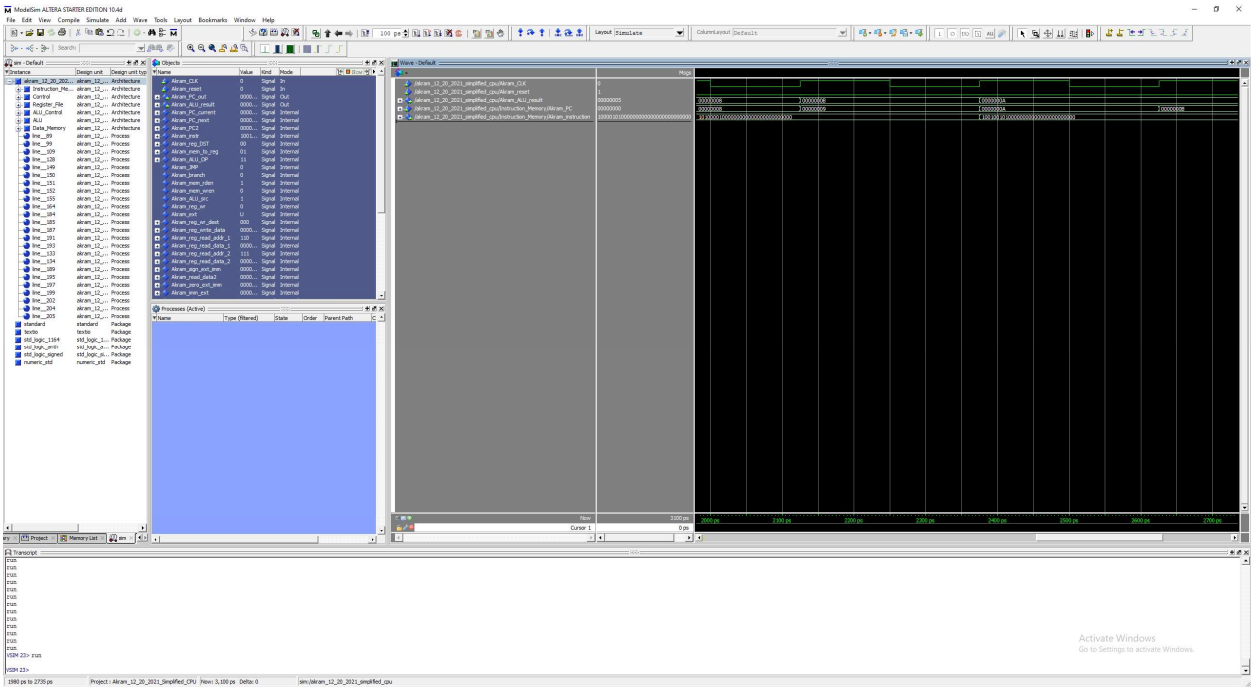


Fig 9c: (Same as Fig 8)c

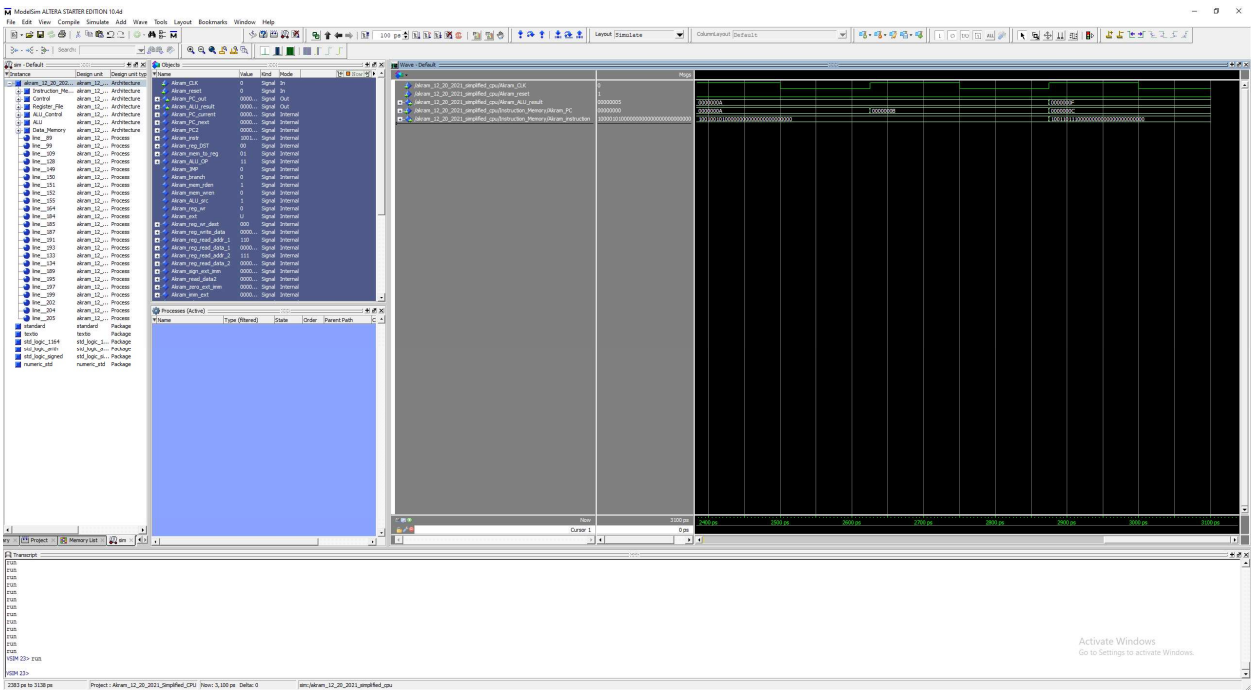


Fig 9d: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 2875 ps: Akram_PC = 12, and we have Akram_read_data_1 = 10 and Akram_read_data_2 = 5

@ 2875+ ps: Akram_ALU_result = 15, Akram_PC = 12, and Akram_read_data_1 = 10 and Akram_read_data_2 = 5

MIPS

Finally, we are testing for correctness in MIPS using MARS.

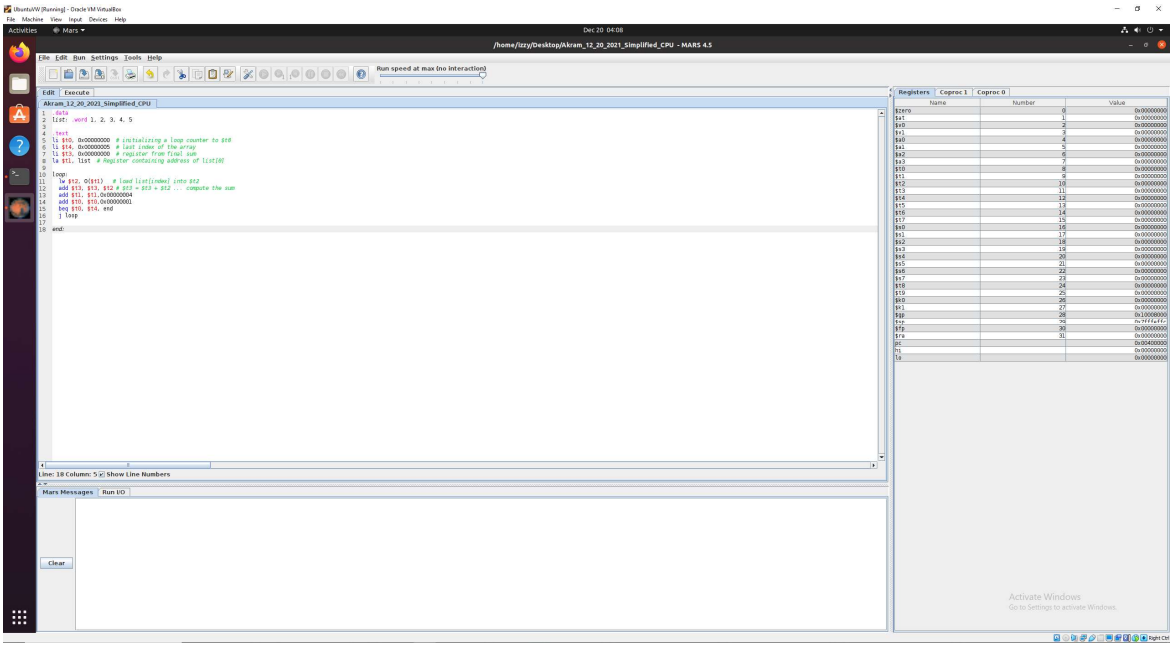


Fig 10: Akram_12_20_2021_Simplified_CPU.asm (Code)

We are simply loading up this code and assembling it.

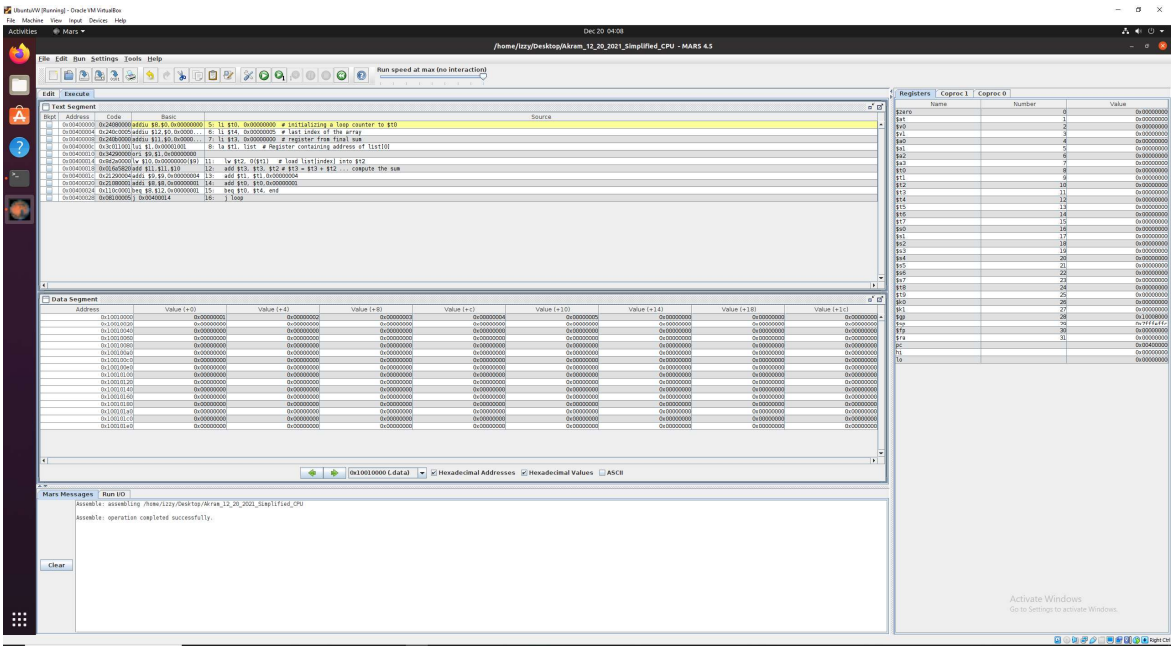


Fig 10a: Akram_12_20_2021_Simplified_CPU.asm (Code)

We are running a loop and adding 1+2+3+4+5 to store in register $t3

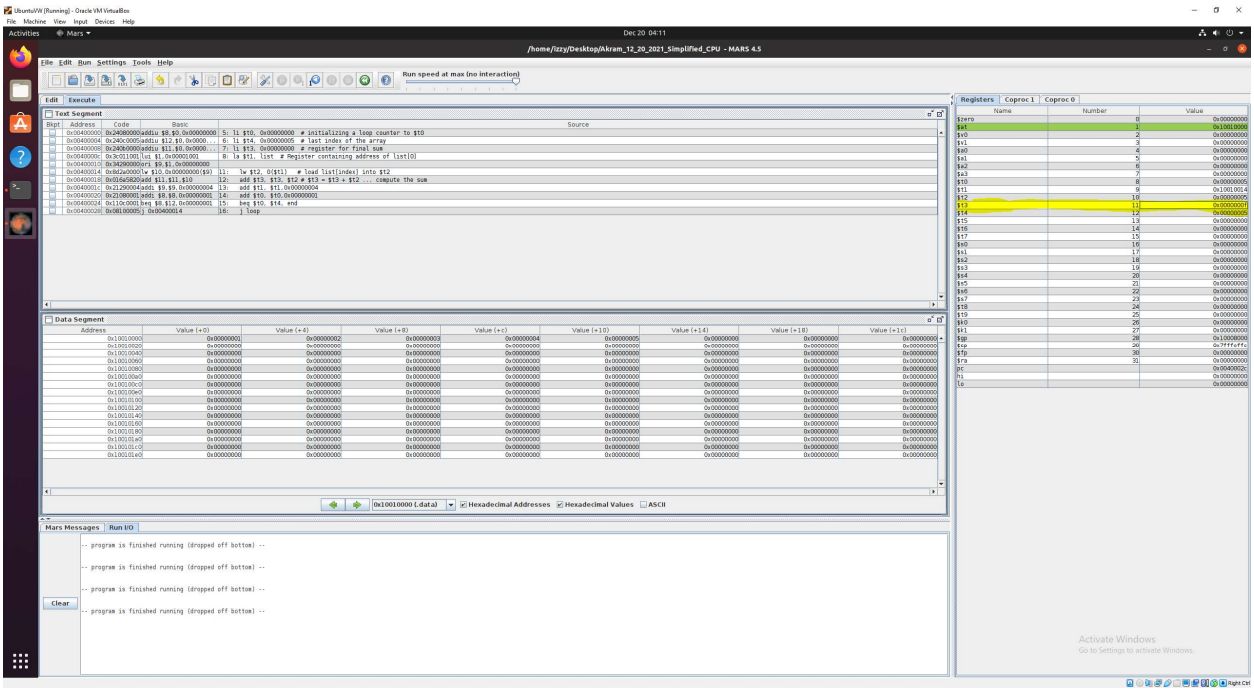


Fig 10c: Akram_12_20_2021_Simplified_CPU.asm (Code)

And finally, once the loop exits, we see that register $t3 stores our final sum; F = 15 (highlighted in yellow.

Conclusion

Our program output the correct sum of 15 by adding 1+2+3+4+5. Our simplified CPU passes the ultimate test. I learned a lot this semester from each lab and as we build each component together to form this final project. I learned how to form logical operators in VHDL and test our design using MIPS on MARS (via VM in Linux). We input data in the Data Memory which was built using LPM RAM from a previous lab. We load the address of the first instruction to the PC-Program Counter register (seen in waveforms). And we executed the code by fetching the first IR-Instruction Register and continued that step-by-step. And finally, we demonstrated correctness of our program via MIP program on MARS simulator.

Thank you.