

Lab 8 CPU Report

Ismail Akram

12/20/2021

CSc34200 & 34300

Prof Gertner

Contents

Objective.....	3
Screenshots.....	4
Components.....	4
CPU, 2:1 MUX, Sign Ext	4
ALU.....	6
ALU Control Adder	7
Control Unit.....	8
Data Memory	9
Instruction Register.....	10
Register File.....	11
Simulations.....	12
MIPS	18
Conclusion.....	20

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.vhd (Code)

Fig 1a: Akram 12 20 2021 Simplified CPU.vhd (code) cont.

Fig 1b: Akram_12_20_2021_Simplified_CPU.vhd (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.vhd (Code)

ALU Control Adder

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

The screenshot shows the Quartus Prime Lite Edition interface with the project 'Akram_12_20_2021_ALU_Control_Adder'. The main window displays the VHDL code for the ALU control adder. The code defines a process based on the ALU operation and control inputs to generate the ALU control output. The project navigator on the left lists various files including 'Alu_Control.vhd', 'Alu_Op.vhd', and 'Alu_Control_Adder.vhd'. The bottom status bar indicates the compilation was successful with 0 errors and 17 warnings.

```

library IEEE;
use IEEE.STD.TEXT.all;
entity Akram_12_20_2021_ALU_Control_Adder is
    port(
        Alu_Control : out STD_LOGIC_VECTOR(3 downto 0); -- ALU Control
        Alu_Op : in STD_LOGIC_VECTOR(1 downto 0); -- operation
        Alu_Control_In : in STD_LOGIC_VECTOR(3 downto 0)); -- function
end Akram_12_20_2021_ALU_Control_Adder;

architecture arch of Akram_12_20_2021_ALU_Control_Adder is
begin
    process(Alu_Op, Alu_Control_In)
    begin
        case Alu_Op is
            when "00" => Alu_Control<= Alu_Control_In;
            when "01" => Alu_Control<= Akram_ALU_Func(0, Alu_Control_In);
            when "10" => Alu_Control<= "000";
            when "11" => Alu_Control<= "100";
            when others => Alu_Control<= "000";
        end case;
    end process;
end arch;

```

Tasks	Completion
Compile Design	00:01
Analyze & Synthesis	00:01
Filter (Pack & Route)	00:01
Assembler (Generate programming file)	00:01
CDA Netlist Verify	00:01
Fit Settings	
Program Device (Open Programmer)	

Messages

- Type ID Message 332140 No Recovery paths to report
- Type ID Message 332140 No paths to report
- Type ID Message 332146 worst-case minimum pulse width slack is -0.083
- Type ID Message 332102 Design is not fully constrained. For setup requirements
- Type ID Message 332102 Design is not fully constrained. For hold requirements
- Type ID Message 332102 Design is not fully constrained. For quartus prime Timewalk Analyzer was successful, 0 errors, 6 warnings
- Type ID Message 2900000 quartus prime full compilation was successful, 0 errors, 17 warnings

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.

```

library IEEE;
use IEEE.STD.TEXT.all;
use IEEE.STD.TEXT.all;
entity Akram_12_20_2021_Control_Unit is
    port(Akram_OP : in std_logic_vector(3 downto 0);
        Akram_ALU_src : out std_logic_vector(3 downto 0);
        Akram_Reg_Wr : out std_logic;
        Akram_Mem_Rden : out std_logic;
        Akram_Mem_Wren : out std_logic;
        Akram_Ex : out std_logic);
end entity;
architecture arch of Akram_12_20_2021_Control_Unit is
begin
    process(Akram_ALU_src)
    begin
        if(Akram_ALU_src = "1") then
            Akram_Reg_Wr <= '0'; -- location of register
            Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
            Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
            Akram_ALU_src <= "0";
            Akram_Ex <= '1';
        else
            case Akram_OP is
                when "000" => -- addition
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when "001" => -- subtraction
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "1";
                    Akram_Ex <= '0';
                when "010" => -- multiply
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when "011" => -- divide
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "1";
                    Akram_Ex <= '0';
                when "100" => -- Jump and Link .... JAL
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '1';
                when "101" => -- Jump .... J
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '1';
                when "110" => -- SW
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '1'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when "111" => -- ADDC
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when others =>
                    Akram_Reg_Wr <= '1'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '1';
            end if;
        end process;
    end architecture;

```

Type ID Message
332140 No recovery paths to report
332140 No removal paths to report
> 332140 worst-case minimum pulse width slack is -0.003
332140 design is not fully constrained for setup requirements
> 332140 design is not fully constrained for hold requirements
> 293000 Quartus Prime Full Compilation was successful. 0 errors, 6 warnings

Fig 4: Akram_12_20_2021_Control_Unit.vhdl (Code)

```

library IEEE;
use IEEE.STD.TEXT.all;
use IEEE.STD.TEXT.all;
entity Akram_12_20_2021_Control_Unit is
    port(Akram_OP : in std_logic_vector(3 downto 0);
        Akram_ALU_src : out std_logic_vector(3 downto 0);
        Akram_Reg_Wr : out std_logic;
        Akram_Mem_Rden : out std_logic;
        Akram_Mem_Wren : out std_logic;
        Akram_Ex : out std_logic);
end entity;
architecture arch of Akram_12_20_2021_Control_Unit is
begin
    process(Akram_ALU_src)
    begin
        if(Akram_ALU_src = "1") then
            Akram_Reg_Wr <= '0'; -- location of register
            Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
            Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
            Akram_ALU_src <= "0";
            Akram_Ex <= '1';
        else
            case Akram_OP is
                when "000" => -- addition
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when "001" => -- subtraction
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "1";
                    Akram_Ex <= '0';
                when "010" => -- multiply
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when "011" => -- divide
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "1";
                    Akram_Ex <= '0';
                when "100" => -- Jump and Link .... JAL
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '1';
                when "101" => -- Jump .... J
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '1';
                when "110" => -- SW
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '1'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when "111" => -- ADDC
                    Akram_Reg_Wr <= '0'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '0';
                when others =>
                    Akram_Reg_Wr <= '1'; -- location of register
                    Akram_Mem_Rden <= '0'; -- read enabled/disabled ... rden
                    Akram_Mem_Wren <= '0'; -- write enabled/disabled ... wren
                    Akram_ALU_src <= "0";
                    Akram_Ex <= '1';
            end if;
        end process;
    end architecture;

```

Type ID Message
332140 No recovery paths to report
332140 No removal paths to report
> 332140 worst-case minimum pulse width slack is -0.003
332140 design is not fully constrained for setup requirements
> 332140 design is not fully constrained for hold requirements
> 293000 Quartus Prime Full Compilation was successful. 0 errors, 6 warnings

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.

A screenshot of the Quartus Prime Lite Edition software interface. The top menu bar includes File, Edit, View, Project, Assignments, Processing, Tools, Window, and Help. The toolbar contains icons for New, Open, Save, Close, Undo, Redo, Find, Copy, Paste, Cut, Select All, Find Next, Find Previous, and Delete. The Project Navigator on the left shows files like 'Akram_12_20_2021_Simplified_CPU.vhd' and 'Akram_12_20_2021_Data_Memory.vhd'. The main window displays VHDL code for 'Akram_12_20_2021_Data_Memory.vhd'. The code defines a memory component with a 32-bit width and a process for reading and writing data based on address and control signals. The bottom left shows the 'Tasks' panel with various synthesis and analysis options. The bottom right shows the 'Messages' panel with a list of warnings and errors from the compilation process.

Fig 5: Akram_12_20_2021_Data_Memory.vhd (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 our simplified CPU, we are only concerned with storage.

The screenshot shows the Quartus Prime Lite Edition interface. The top menu bar includes File, Edit, View, Project, Assemblies, Processing, Tools, Window, Help, and a search bar. The toolbar contains icons for New, Open, Save, Import, Export, Run, Stop, and other project management functions.

The main window displays a VHDL code editor with the file `Akran_12_20_2021_Simplified_CPU.vhd`. The code defines a memory map from address 0x00000000 to 0x00000020, with specific memory regions for ROM, RAM, and ROM. It also includes a process for reading data from memory based on the program counter (Akran_Pc).

The left sidebar shows the project tree with files like `Akran_12_20_2021_Simplified_CPU.vhd`, `Akran_12_20_2021_Data_Memory.vhd`, `Akran_12_20_2021_Registers.vhd`, and `Akran_12_20_2021_R.vhd`.

The bottom status bar shows the system and processing progress, and a message from Microsoft about activating Windows.

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).

The screenshot shows the Quartus Prime software interface with the following details:

- Project Navigator:** Shows files like `Airan_12_20_2021_Simplified_CPU.vhd`, `Airan_12_20_2021_ALU.vhd`, `Airan_12_20_2021_Control_Registers.vhd`, `Airan_12_20_2021_Data_Memory.vhd`, `Airan_12_20_2021_Register_File.vhd`, and `Completion Report - Airan_12_20_2021_Simplified_CPU`.
- File Explorer:** Shows files such as `Airan_12_20_2021_Simplified_CPU.vhd`, `Airan_12_20_2021_ALU.vhd`, `Airan_12_20_2021_Control_Registers.vhd`, `Airan_12_20_2021_Data_Memory.vhd`, `Airan_12_20_2021_Register_File.vhd`, and `Completion Report - Airan_12_20_2021_Simplified_CPU`.
- Code Editor:** Displays the VHDL code for the `Architecture` of the `Airan` component. The code includes declarations for `std_logic`, `std_logic_vector`, and `std.natural`. It defines a `Register_File` type and its architecture. The architecture uses a `for loop` to iterate through registers from 0 to 31, setting their initial values to 0. It also includes logic for reading and writing data to specific registers based on address and control signals.
- Tasks:** A list of completed tasks including:
 - Compile Design
 - Analysis & Synthesis
 - Filter (Place & Route)
 - Assembler (Generate program files)
 - TimeQuest Timing Analysis
 - EDA Netlist Writer
 - Edit Settings
- Terminal:** Shows the output of the build process, indicating success with messages like "Quartus Prime Full Compilation was successful. 0 errors, 0 warnings".
- Status Bar:** Shows "Activate Windows Go to Settings to activate Windows".
- Bottom Navigation:** Includes tabs for System, Processing (12), and a status bar showing 100% completion at 00:31.

Fig 7: Akram_12_20_2021_Register_File.vhd (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 $1+2+3+4+5 = 15$

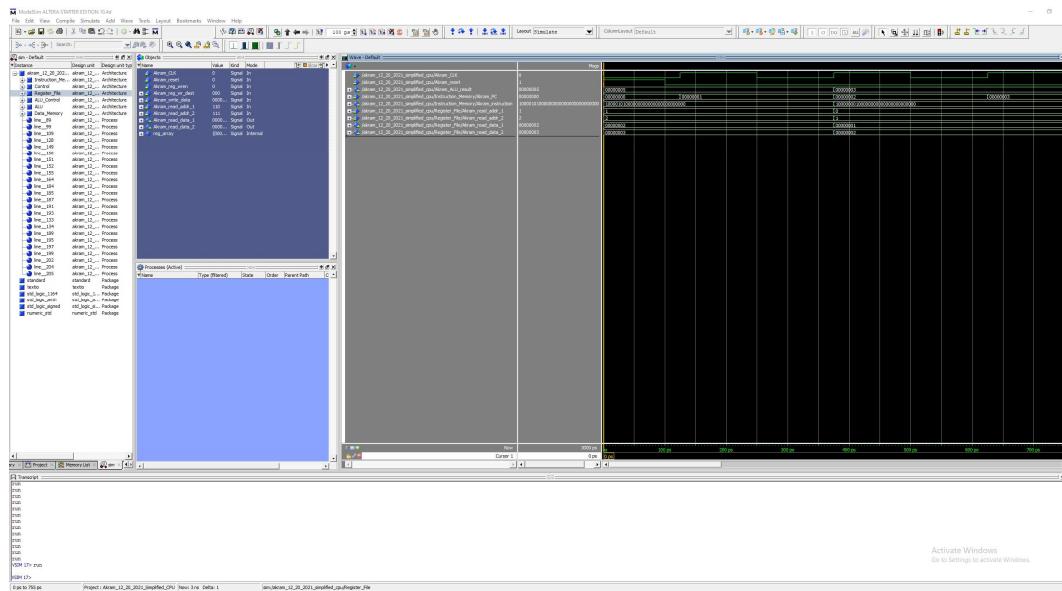


Fig 8: ModelSim of Akram_12_20_2021_Simplified_CPU

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

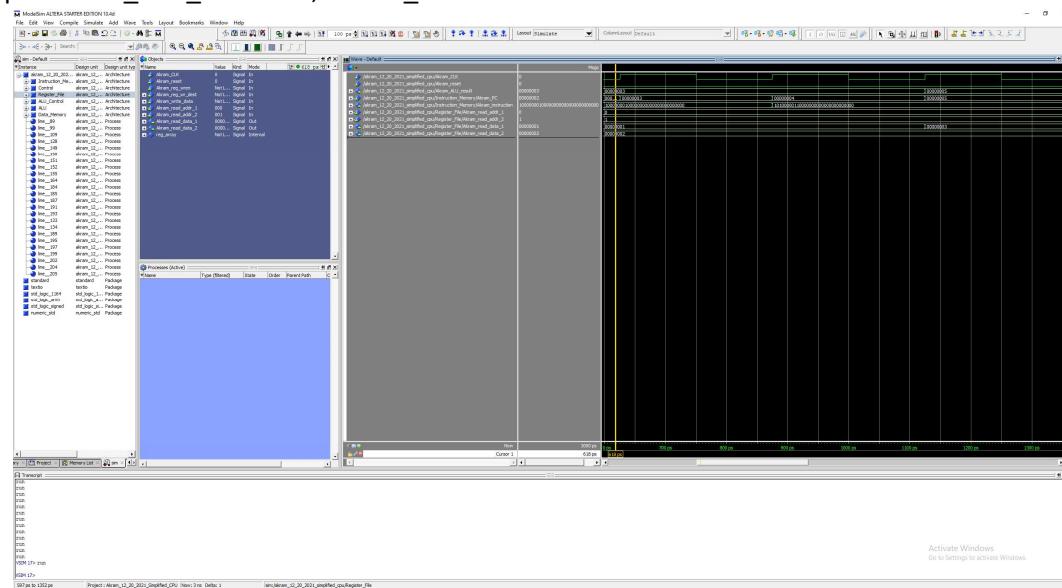


Fig 8a: ModelSim of Akram_12_20_2021_Simplified_CPU

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

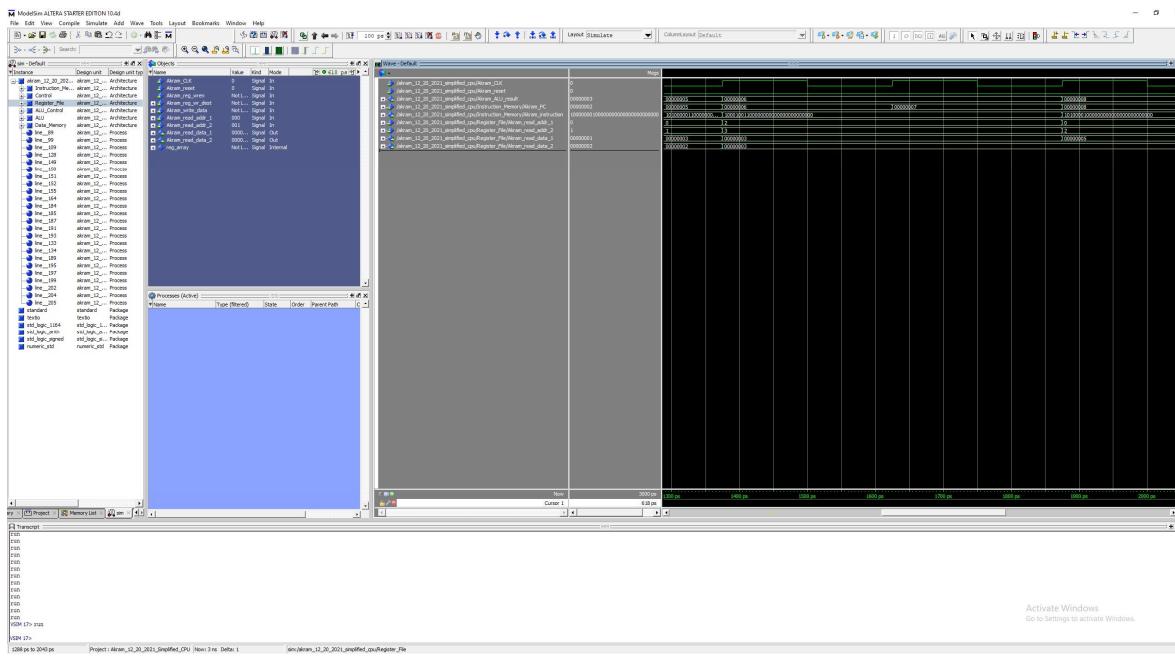


Fig 8b: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 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

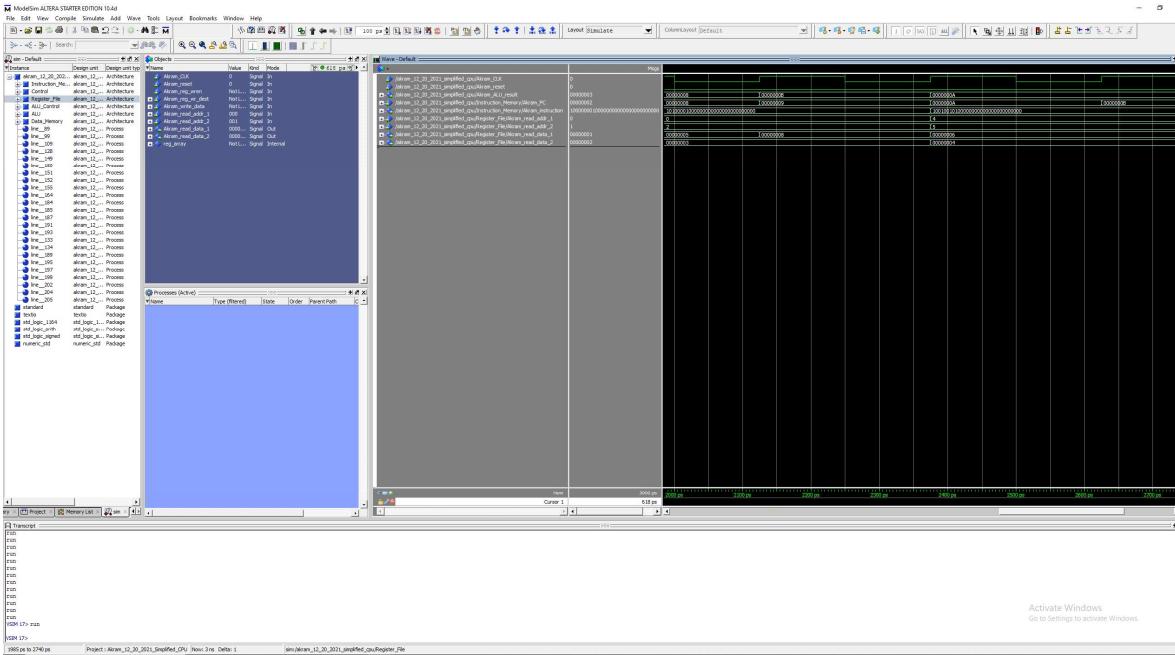


Fig 8c: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 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

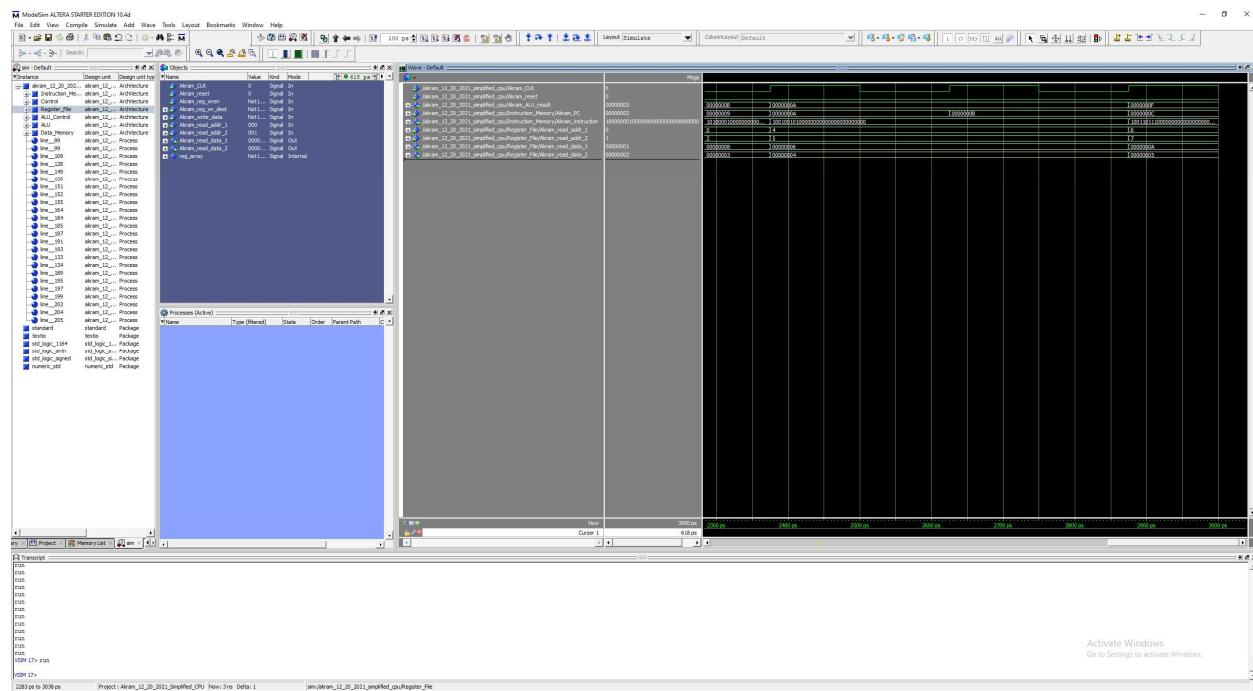


Fig 8d: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 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):

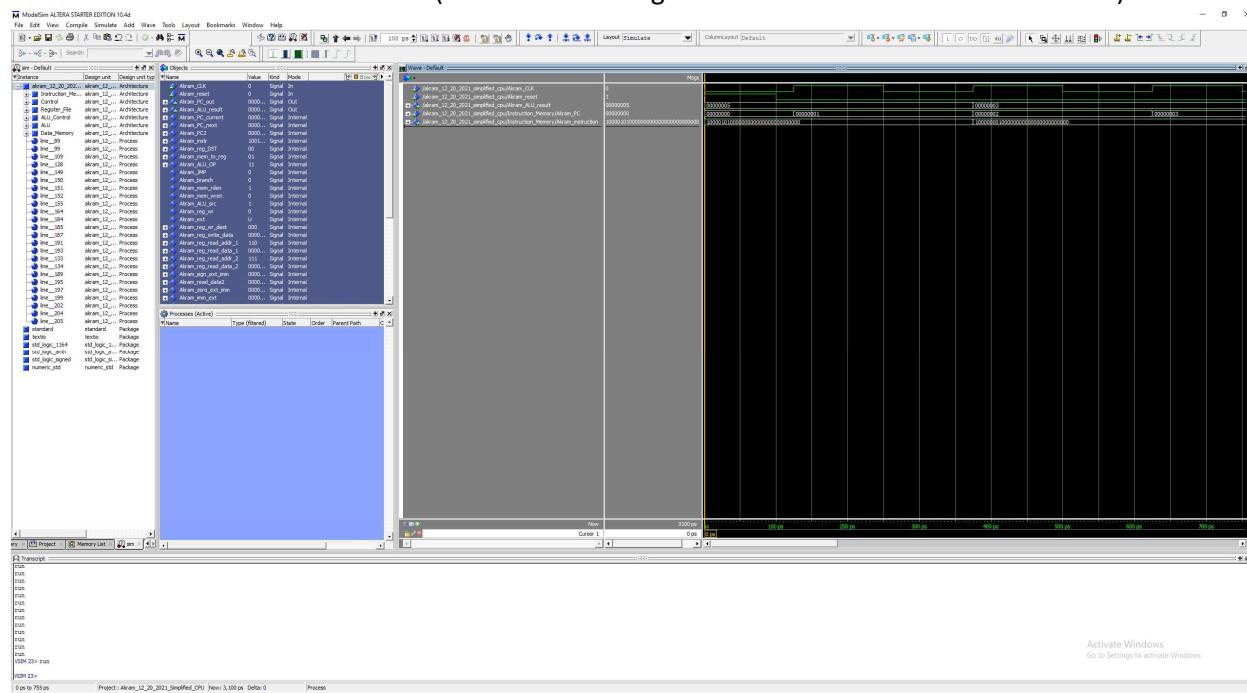


Fig 9: (Same as Fig 8)

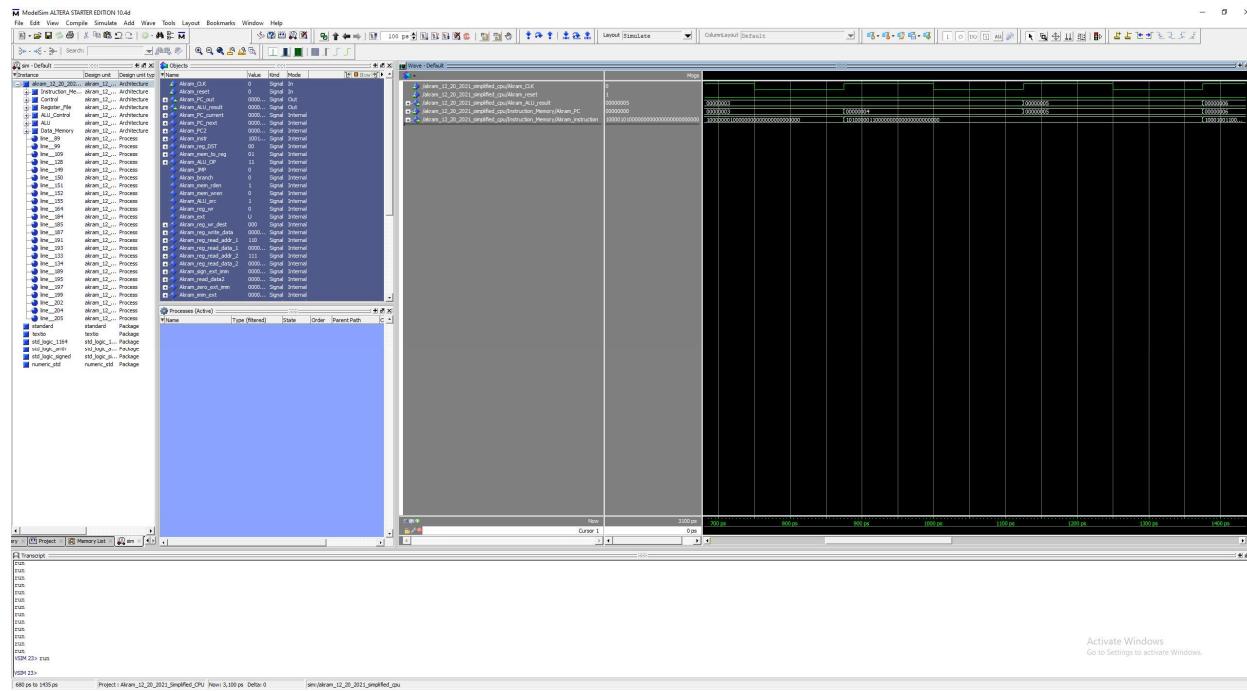


Fig 9a: (Same as Fig 8a)

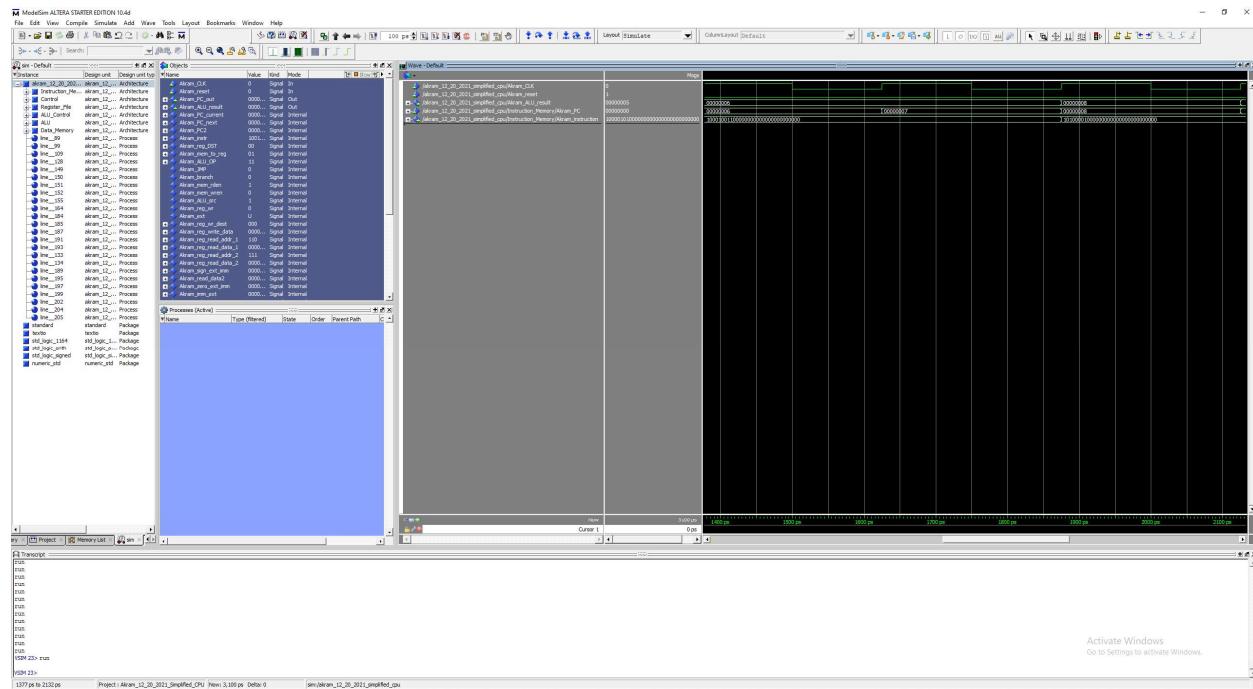


Fig 9b: (Same as Fig 8b)

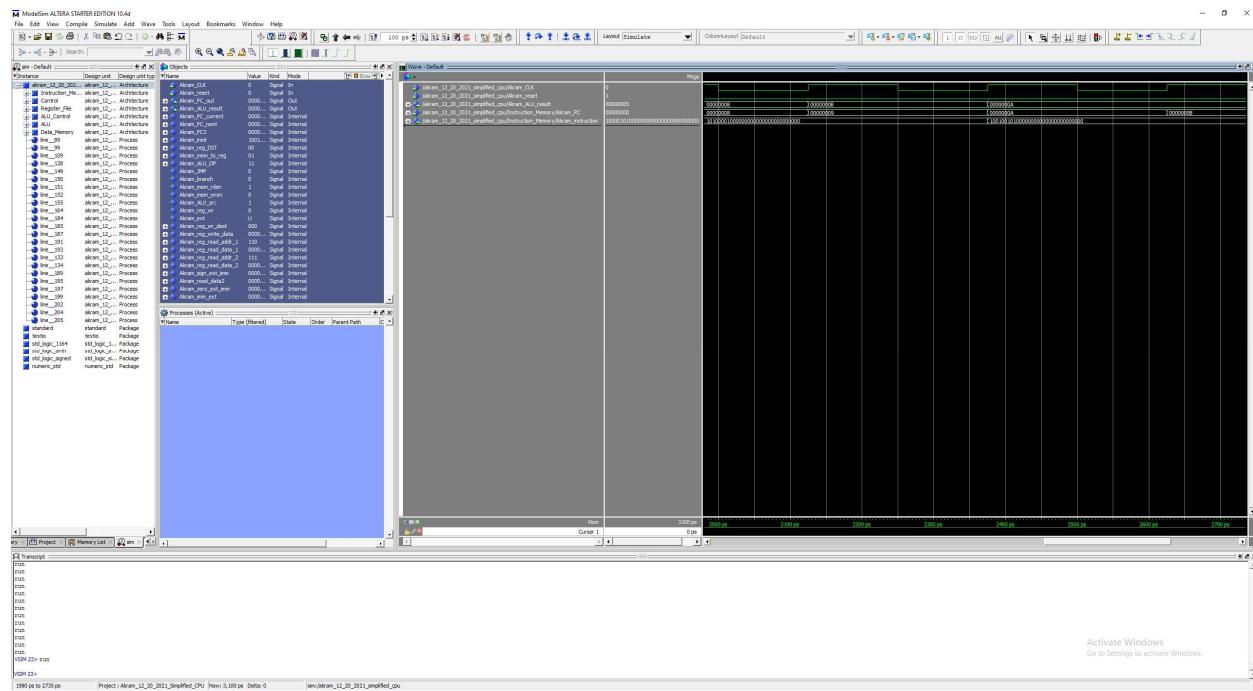


Fig 9c: (Same as Fig 8c)

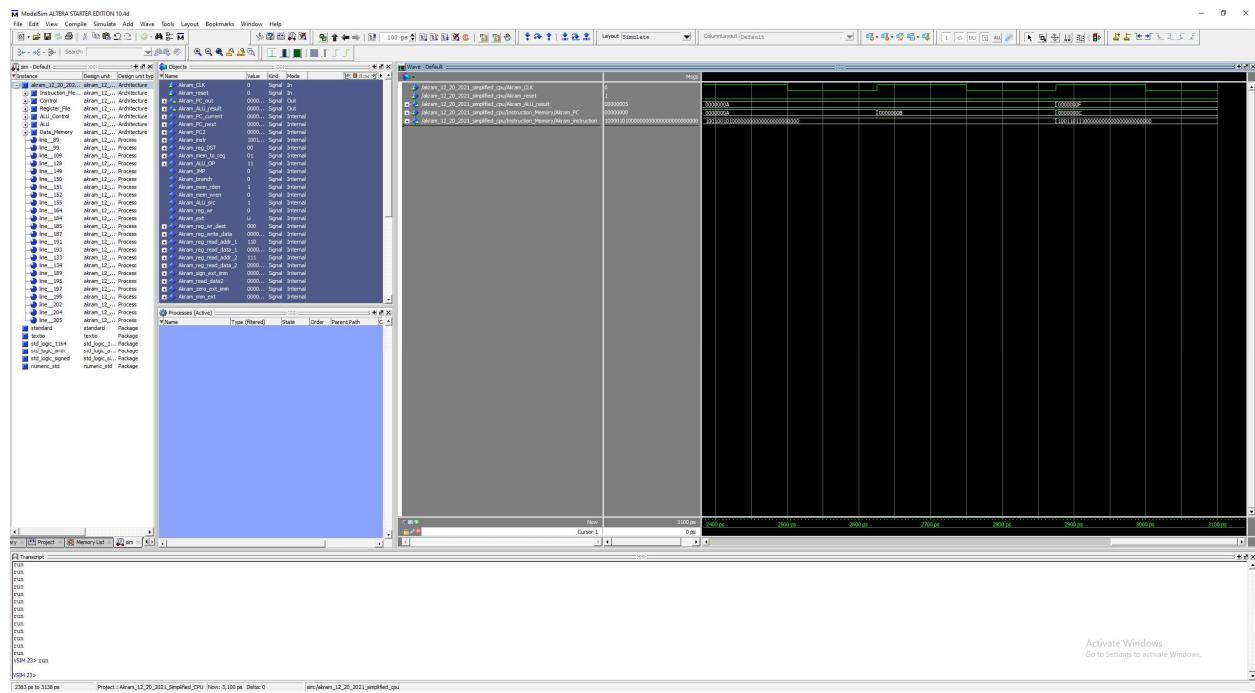


Fig 9d: ModelSim of Akram_12_20_2021_Simplified_CPU

@ 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.

The screenshot shows the MARS 4.5 assembly editor interface. The top menu bar includes File, Edit, Run, Settings, Tools, Help, and Activities. The title bar displays "Ubuntu/Ubuntu - Oracle VM VirtualBox" and the file path "/home/itzzy/Desktop/Akram_12_20_2021_Simplified_CPU - MARS 4.5". The status bar at the bottom left shows "Line: 18 Column: 5" and "Show Line Numbers". The status bar at the bottom right says "Activate Windows Go to Settings to activate Windows".

The main window contains the assembly code:

```
File Machine View Input Device Help
Activities Mars
Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Akram_12_20_2021_Simplified_CPU
A
2 ldi r0, #1, 2, 3, 4, 5
3 test
4 li $t0, 0x00000000
5 li $t1, 0x00000000
6 li $t4, 0x00000000
7 la $t1, list
8 la $t2, list + 4
9 add $t3, $t1, $t2
10 lop:
11 add $t3, $t3, $t4
12 add $t5, $t3, $t4
13 add $t6, $t5, $t4
14 add $t7, $t6, $t4
15 add $t8, $t7, $t4
16 add $t9, $t8, $t4
17 lop
18 end
```

The Registers window on the right lists the following register values:

Registers	Coprocs 1	Coprocs 0
\$t0	0	0x00000000
\$t1	1	0x00000001
\$t2	2	0x00000002
\$t3	3	0x00000003
\$t4	4	0x00000004
\$t5	5	0x00000005
\$t6	6	0x00000006
\$t7	7	0x00000007
\$t8	8	0x00000008
\$t9	9	0x00000009
\$t10	10	0x0000000A
\$t11	11	0x0000000B
\$t12	12	0x0000000C
\$t13	13	0x0000000D
\$t14	14	0x0000000E
\$t15	15	0x0000000F
\$t16	16	0x00000010
\$t17	17	0x00000011
\$t18	18	0x00000012
\$t19	19	0x00000013
\$t20	20	0x00000014
\$t21	21	0x00000015
\$t22	22	0x00000016
\$t23	23	0x00000017
\$t24	24	0x00000018
\$t25	25	0x00000019
\$t26	26	0x0000001A
\$t27	27	0x0000001B
\$t28	28	0x0000001C
\$t29	29	0x0000001D
\$t30	30	0x0000001E
\$t31		0x00400000
\$t32		0x00000000
\$t33		0x00000000

The bottom left corner has a "Clear" button.

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.