Computer Architecture Report

Done By Omar Elmahy
ID: 210237

Module Leader: Prof. Mervat Mohamed
Teacher Assistant: Eng. Zahraa Ismail

Table of Contents

Table of Figures

Introduction

Elma7eCPU represents a personal endeavor in CPU design, conceived with the aim of exploring the intricacies of Computer Architecture within the context of electrical engineering. This project serves as a comprehensive exploration of the fundamental principles underlying the construction and operation of a central processing unit (CPU).

At its core, this project seeks to provide a holistic understanding of CPU architecture, offering students an immersive experience in the conceptualization, design, and implementation of a fully functional CPU. By embarking on this journey, students are equipped with valuable insights into the inner workings of computing systems, preparing them for future challenges in the field of digital design and engineering.

The primary objective of the elma7eCPU project is to facilitate hands-on learning in CPU design through the utilization of VHDL (VHSIC Hardware Description Language). Through the creation of essential CPU components such as the Arithmetic Logic Unit (ALU), instruction memory, program counter, and register file. Students are tasked with translating theoretical concepts into tangible hardware descriptions.

Moreover, elma7eCPU serves as a practical platform for students to apply theoretical knowledge acquired in lectures on computer architecture. By engaging in the design and simulation of a CPU, students deepen their understanding of critical concepts including instruction fetching, decoding, execution, and memory management.

Throughout the development process, students encounter challenges inherent in translating abstract architectural principles into functional hardware descriptions. This journey fosters the development of essential skills such as critical thinking, problem-solving, and attention to detail, all of which are essential in the field of digital system design.

Ultimately, the elma7eCPU project transcends its role as a mere academic exercise; it represents a journey of exploration and discovery. It empowers students to unravel the mysteries of CPU architecture, laying a solid foundation for their future endeavors in the dynamic realm of digital design and computer engineering.

Theoretical Background

The elma7eCPU project is grounded in the theoretical underpinnings of computer architecture, a field that explores the design principles and organization of digital computing systems. At its core, computer architecture encompasses the structure and behavior of various components within a computer system, with a particular focus on the central processing unit (CPU).

1. CPU Architecture:

The CPU serves as the brain of a computer, responsible for executing instructions and performing computations. It comprises several key components, including the Arithmetic Logic Unit (ALU), control unit, registers, and various interconnection pathways.

2. VHDL (VHSIC Hardware Description Language):

VHDL is a hardware description language used for modeling and simulating digital systems. It provides a means to describe the behavior and structure of digital circuits, making it an ideal tool for CPU design and implementation.

3. ALU Operations:

The ALU is a critical component of the CPU responsible for performing arithmetic and logic operations on data. Common operations include addition, subtraction, logical AND, and incrementing.

4. Register File:

Registers are small, high-speed memory units within the CPU used for storing temporary data and operands during instruction execution. The register file is a collection of registers accessible to the CPU for data storage and manipulation.

5. Program Counter (PC):

The program counter is a special register that keeps track of the memory address of the next instruction to be fetched and executed by the CPU. It is incremented after each instruction execution to point to the next sequential instruction.

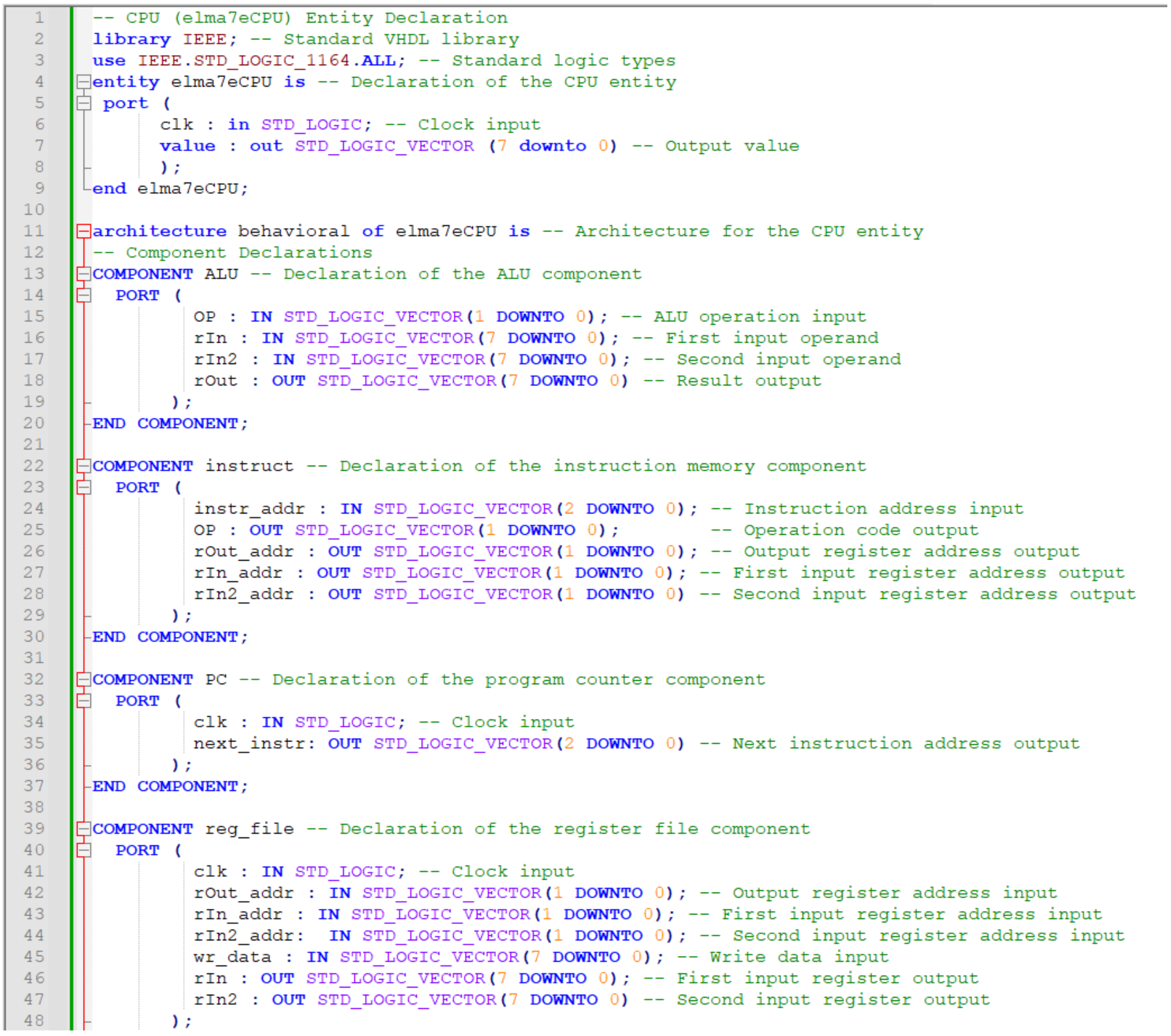
6. Instruction Memory:

Instruction memory stores the program instructions to be executed by the CPU. It provides the CPU with the necessary instructions based on the current value of the program counter.

Tests and evaluations

The code that was tested in this project is split into 6 different code segments:

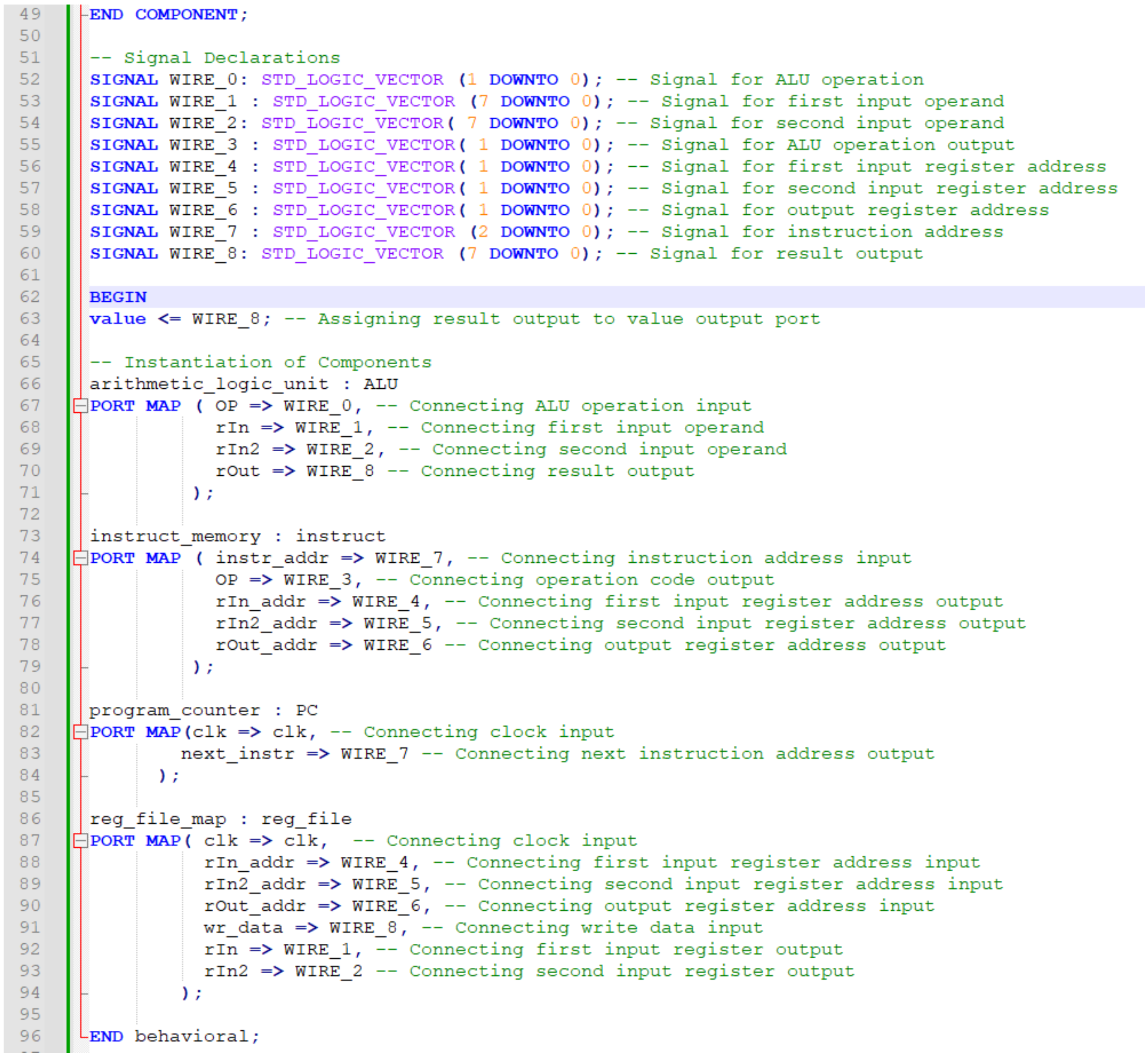
1. CPU:



```vhdl
-- CPU (elma7eCPU) Entity Declaration
library IEEE; -- Standard VHDL library
use IEEE.STD_LOGIC_1164.ALL; -- Standard logic types
entity elma7eCPU is -- Declaration of the CPU entity
 port (
      clk : in STD_LOGIC; -- Clock input
      value : out STD_LOGIC_VECTOR (7 downto 0) -- Output value
      );
end elma7eCPU;

architecture behavioral of elma7eCPU is -- Architecture for the CPU entity
-- Component Declarations
COMPONENT ALU -- Declaration of the ALU component
  PORT (
         OP : IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- ALU operation input
         rIn : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- First input operand
         rIn2 : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Second input operand
         rOut : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) -- Result output
       );
END COMPONENT;

COMPONENT instruct -- Declaration of the instruction memory component
  PORT (
         instr_addr : IN STD_LOGIC_VECTOR(2 DOWNTO 0); -- Instruction address input
         OP : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);          -- Operation code output
         rOut_addr : OUT STD_LOGIC_VECTOR(1 DOWNTO 0); -- Output register address output
         rIn_addr : OUT STD_LOGIC_VECTOR(1 DOWNTO 0); -- First input register address output
         rIn2_addr : OUT STD_LOGIC_VECTOR(1 DOWNTO 0) -- Second input register address output
       );
END COMPONENT;

COMPONENT PC -- Declaration of the program counter component
  PORT (
         clk : IN STD_LOGIC; -- Clock input
         next_instr: OUT STD_LOGIC_VECTOR(2 DOWNTO 0) -- Next instruction address output
       );
END COMPONENT;

COMPONENT reg_file -- Declaration of the register file component
  PORT (
         clk : IN STD_LOGIC; -- Clock input
         rOut_addr : IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- Output register address input
         rIn_addr : IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- First input register address input
         rIn2_addr:  IN STD_LOGIC_VECTOR(1 DOWNTO 0); -- Second input register address input
         wr_data : IN STD_LOGIC_VECTOR(7 DOWNTO 0); -- Write data input
         rIn : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- First input register output
         rIn2 : OUT STD_LOGIC_VECTOR(7 DOWNTO 0) -- Second input register output
       );
```

Figure 1

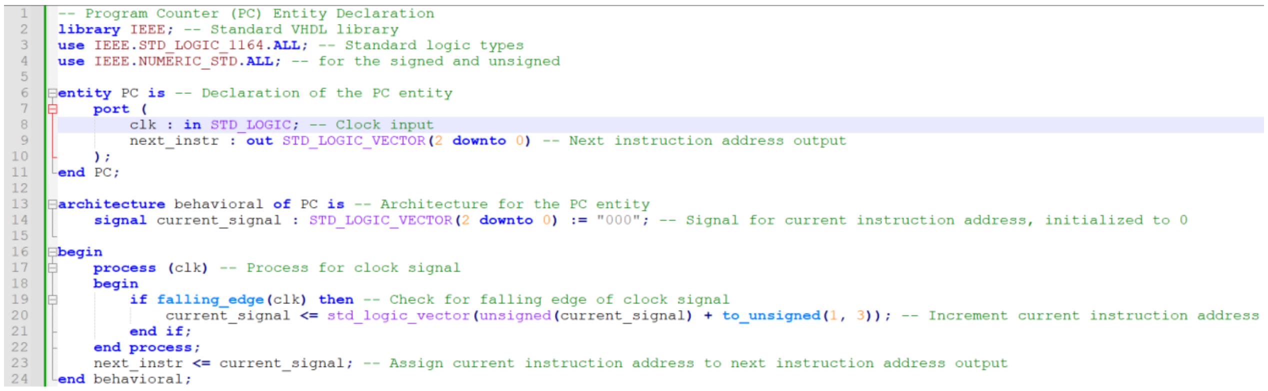
The rest of the CPU CODE:



```vhdl
END COMPONENT;

-- Signal Declarations
SIGNAL WIRE_0: STD_LOGIC_VECTOR (1 DOWNTO 0); -- Signal for ALU operation
SIGNAL WIRE_1 : STD_LOGIC_VECTOR (7 DOWNTO 0); -- Signal for first input operand
SIGNAL WIRE_2: STD_LOGIC_VECTOR( 7 DOWNTO 0); -- Signal for second input operand
SIGNAL WIRE_3 : STD_LOGIC_VECTOR( 1 DOWNTO 0); -- Signal for ALU operation output
SIGNAL WIRE_4 : STD_LOGIC_VECTOR( 1 DOWNTO 0); -- Signal for first input register address
SIGNAL WIRE_5 : STD_LOGIC_VECTOR( 1 DOWNTO 0); -- Signal for second input register address
SIGNAL WIRE_6 : STD_LOGIC_VECTOR( 1 DOWNTO 0); -- Signal for output register address
SIGNAL WIRE_7 : STD_LOGIC_VECTOR (2 DOWNTO 0); -- Signal for instruction address
SIGNAL WIRE_8: STD_LOGIC_VECTOR (7 DOWNTO 0); -- Signal for result output

BEGIN
value <= WIRE_8; -- Assigning result output to value output port

-- Instantiation of Components
arithmetic_logic_unit : ALU
PORT MAP ( OP => WIRE_0, -- Connecting ALU operation input
           rIn => WIRE_1, -- Connecting first input operand
           rIn2 => WIRE_2, -- Connecting second input operand
           rOut => WIRE_8 -- Connecting result output
         );

instruct_memory : instruct
PORT MAP ( instr_addr => WIRE_7, -- Connecting instruction address input
           OP => WIRE_3, -- Connecting operation code output
           rIn_addr => WIRE_4, -- Connecting first input register address output
           rIn2_addr => WIRE_5, -- Connecting second input register address output
           rOut_addr => WIRE_6 -- Connecting output register address output
         );

program_counter : PC
PORT MAP(clk => clk, -- Connecting clock input
        next_instr => WIRE_7 -- Connecting next instruction address output
      );

reg_file_map : reg_file
PORT MAP( clk => clk,  -- Connecting clock input
          rIn_addr => WIRE_4, -- Connecting first input register address input
          rIn2_addr => WIRE_5, -- Connecting second input register address input
          rOut_addr => WIRE_6, -- Connecting output register address input
          wr_data => WIRE_8, -- Connecting write data input
          rIn => WIRE_1, -- Connecting first input register output
          rIn2 => WIRE_2 -- Connecting second input register output
        );

END behavioral;
```

Figure 2

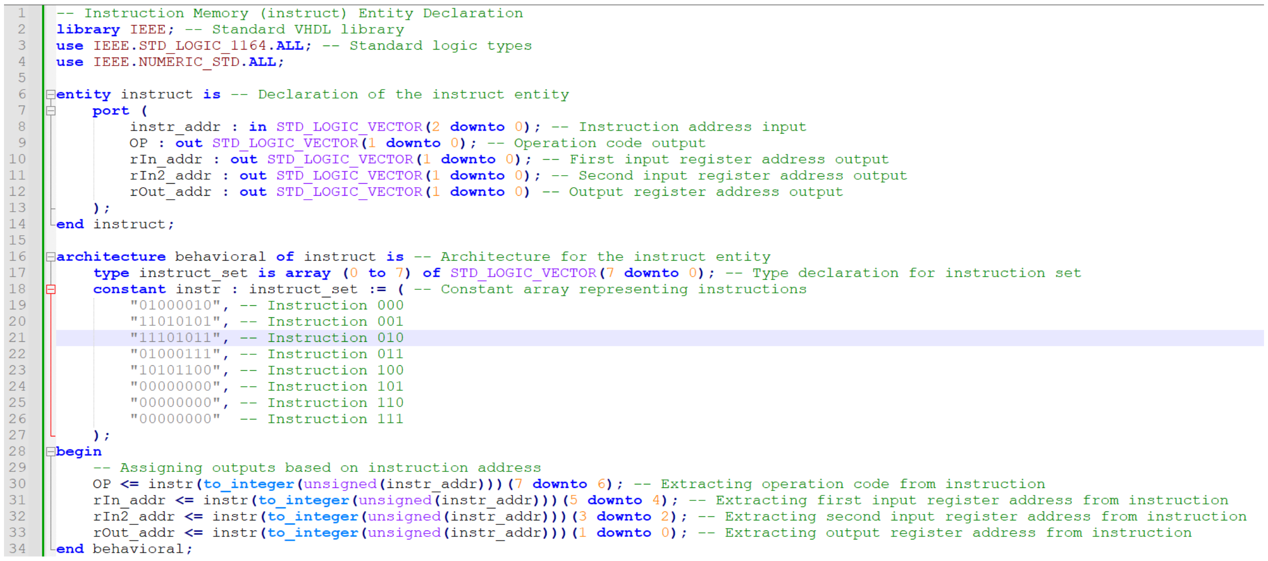
Program Counter (PC):



```vhdl
-- Program Counter (PC) Entity Declaration
library IEEE; -- Standard VHDL library
use IEEE.STD_LOGIC_1164.ALL; -- Standard logic types
use IEEE.NUMERIC_STD.ALL; -- for the signed and unsigned

entity PC is -- Declaration of the PC entity
    port (
        clk : in STD_LOGIC; -- Clock input
        next_instr : out STD_LOGIC_VECTOR(2 downto 0) -- Next instruction address output
    );
end PC;

architecture behavioral of PC is -- Architecture for the PC entity
    signal current_signal : STD_LOGIC_VECTOR(2 downto 0) := "000"; -- Signal for current instruction address, initialized to 0

begin
    process (clk) -- Process for clock signal
    begin
        if falling_edge(clk) then -- Check for falling edge of clock signal
            current_signal <= std_logic_vector(unsigned(current_signal) + to_unsigned(1, 3)); -- Increment current instruction address
        end if;
    end process;
    next_instr <= current_signal; -- Assign current instruction address to next instruction address output
end behavioral;
```

Figure 3

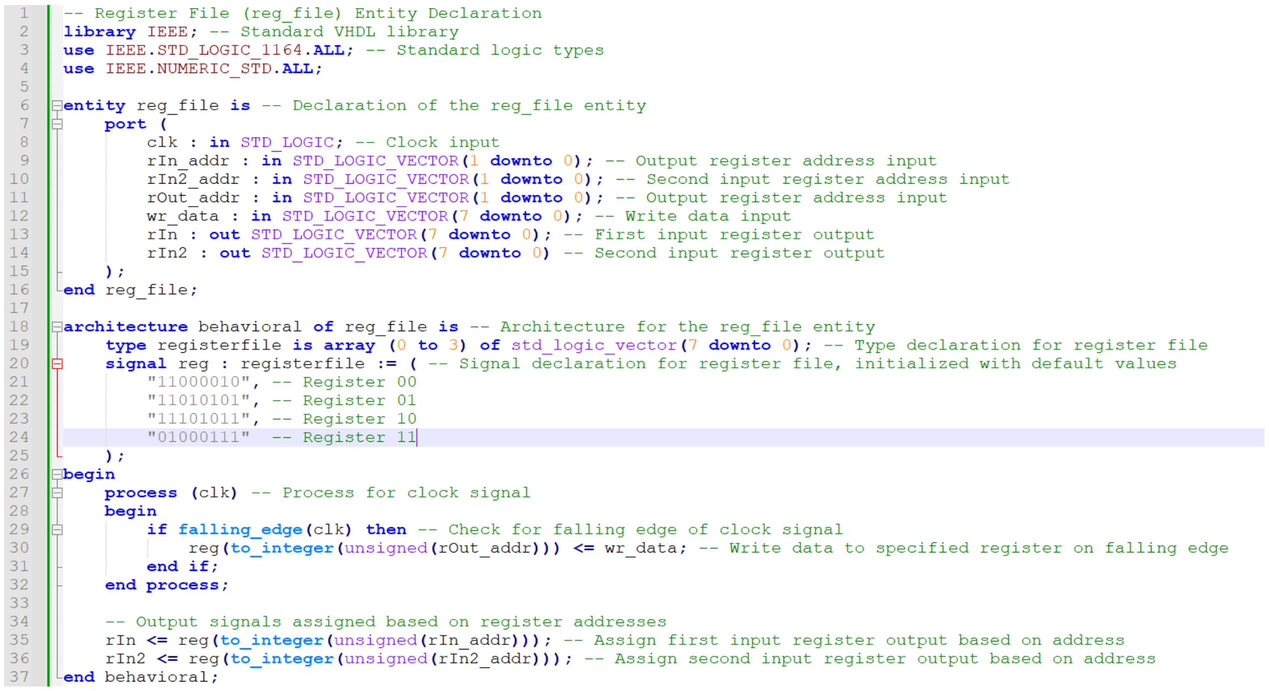
Memory Instruction:



```vhdl
-- Instruction Memory (instruct) Entity Declaration
library IEEE; -- Standard VHDL library
use IEEE.STD_LOGIC_1164.ALL; -- Standard logic types
use IEEE.NUMERIC_STD.ALL;

entity instruct is -- Declaration of the instruct entity
    port (
        instr_addr : in STD_LOGIC_VECTOR(2 downto 0); -- Instruction address input
        OP : out STD_LOGIC_VECTOR(1 downto 0); -- Operation code output
        rIn_addr : out STD_LOGIC_VECTOR(1 downto 0); -- First input register address output
        rIn2_addr : out STD_LOGIC_VECTOR(1 downto 0); -- Second input register address output
        rOut_addr : out STD_LOGIC_VECTOR(1 downto 0) -- Output register address output
    );
end instruct;

architecture behavioral of instruct is -- Architecture for the instruct entity
    type instruct_set is array (0 to 7) of STD_LOGIC_VECTOR(7 downto 0); -- Type declaration for instruction set
    constant instr : instruct_set := ( -- Constant array representing instructions
        "01000010", -- Instruction 000
        "11010101", -- Instruction 001
        "11101011", -- Instruction 010
        "01000111", -- Instruction 011
        "10101100", -- Instruction 100
        "00000000", -- Instruction 101
        "00000000", -- Instruction 110
        "00000000"  -- Instruction 111
    );
begin
    -- Assigning outputs based on instruction address
    OP <= instr(to_integer(unsigned(instr_addr)))(7 downto 6); -- Extracting operation code from instruction
    rIn_addr <= instr(to_integer(unsigned(instr_addr)))(5 downto 4); -- Extracting first input register address from instruction
    rIn2_addr <= instr(to_integer(unsigned(instr_addr)))(3 downto 2); -- Extracting second input register address from instruction
    rOut_addr <= instr(to_integer(unsigned(instr_addr)))(1 downto 0); -- Extracting output register address from instruction
end behavioral;
```

Figure 4

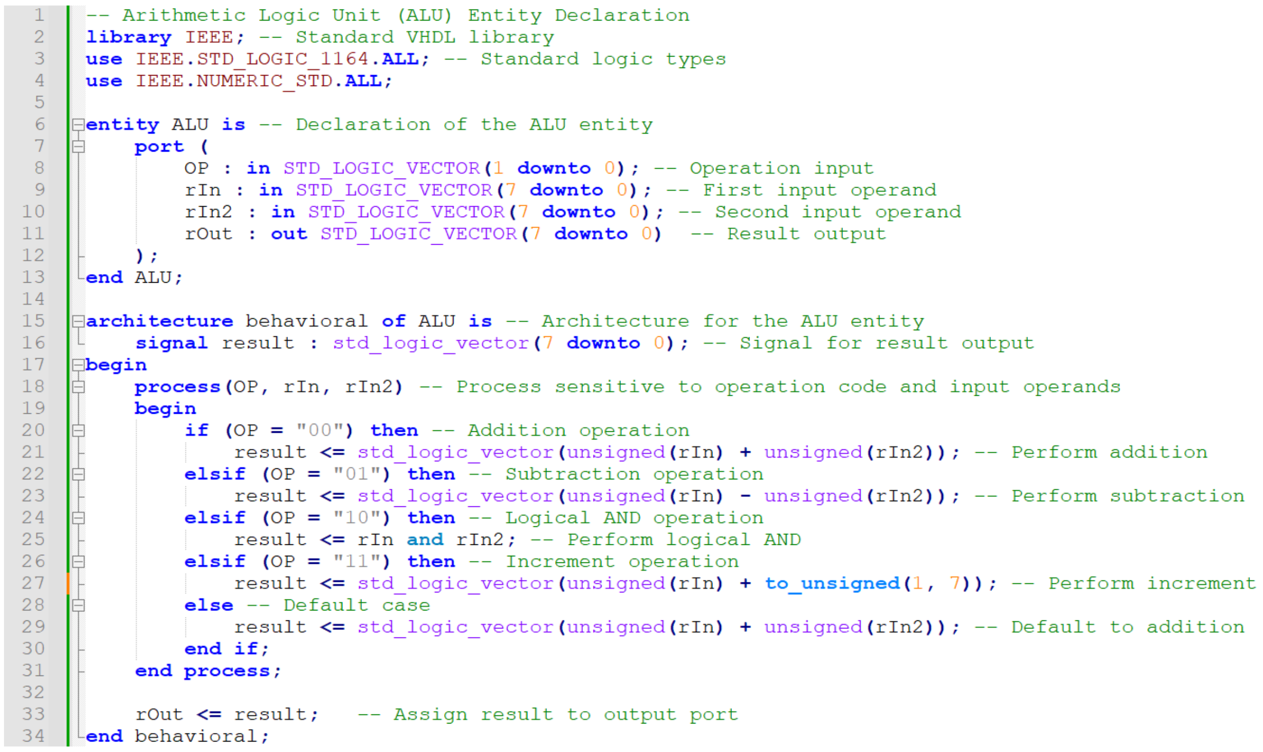
Register File:



```vhdl
-- Register File (reg_file) Entity Declaration
library IEEE; -- Standard VHDL library
use IEEE.STD_LOGIC_1164.ALL; -- Standard logic types
use IEEE.NUMERIC_STD.ALL;

entity reg_file is -- Declaration of the reg_file entity
    port (
        clk : in STD_LOGIC; -- Clock input
        rIn_addr : in STD_LOGIC_VECTOR(1 downto 0); -- Output register address input
        rIn2_addr : in STD_LOGIC_VECTOR(1 downto 0); -- Second input register address input
        rOut_addr : in STD_LOGIC_VECTOR(1 downto 0); -- Output register address input
        wr_data : in STD_LOGIC_VECTOR(7 downto 0); -- Write data input
        rIn : out STD_LOGIC_VECTOR(7 downto 0); -- First input register output
        rIn2 : out STD_LOGIC_VECTOR(7 downto 0) -- Second input register output
    );
end reg_file;

architecture behavioral of reg_file is -- Architecture for the reg_file entity
    type registerfile is array (0 to 3) of std_logic_vector(7 downto 0); -- Type declaration for register file
    signal reg : registerfile := ( -- Signal declaration for register file, initialized with default values
        "11000010", -- Register 00
        "11010101", -- Register 01
        "11101011", -- Register 10
        "01000111"  -- Register 11
    );
begin
    process (clk) -- Process for clock signal
    begin
        if falling_edge(clk) then -- Check for falling edge of clock signal
            reg(to_integer(unsigned(rOut_addr))) <= wr_data; -- Write data to specified register on falling edge
        end if;
    end process;

    -- Output signals assigned based on register addresses
    rIn <= reg(to_integer(unsigned(rIn_addr))); -- Assign first input register output based on address
    rIn2 <= reg(to_integer(unsigned(rIn2_addr))); -- Assign second input register output based on address
end behavioral;
```

Figure 5

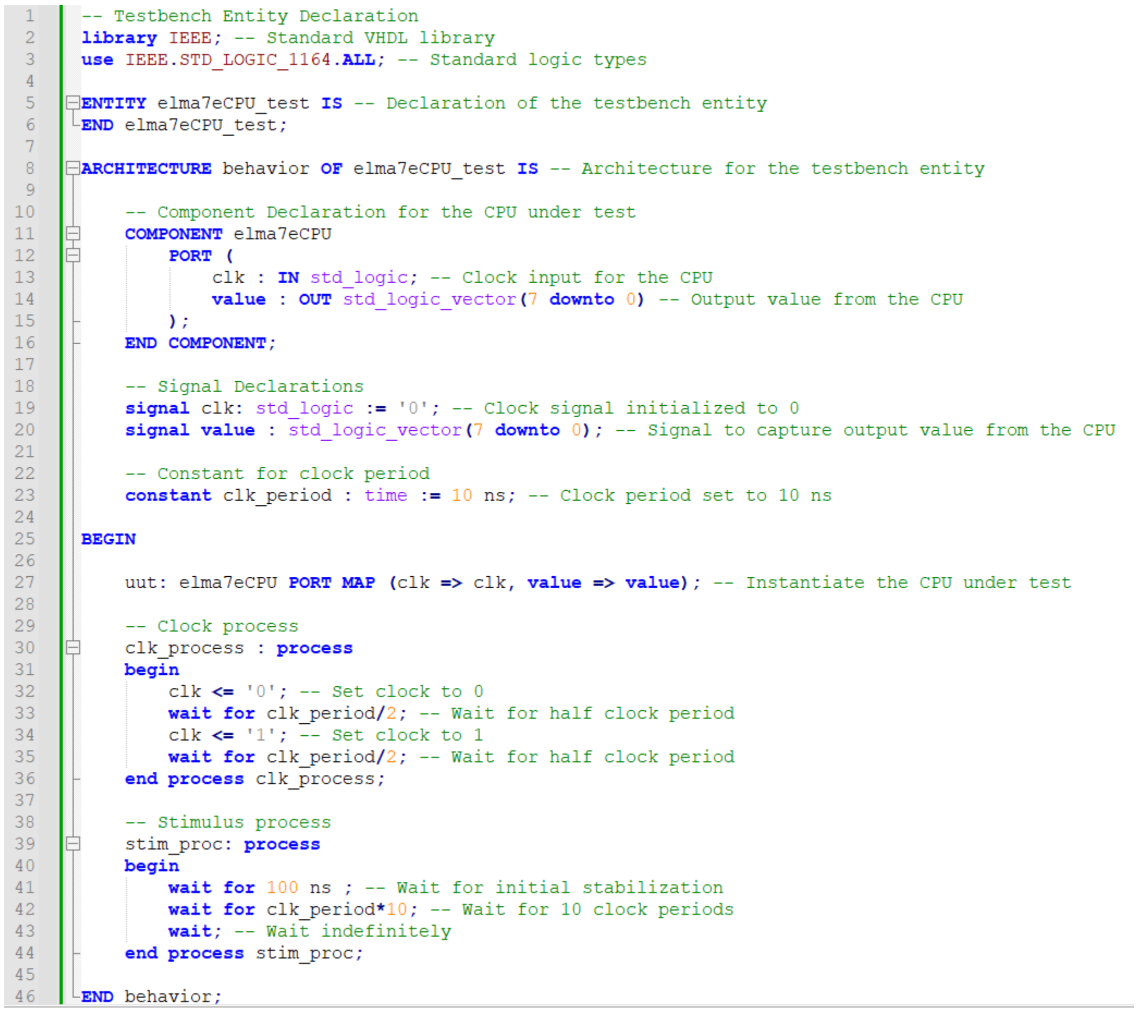
ALU:



```vhdl
-- Arithmetic Logic Unit (ALU) Entity Declaration
library IEEE; -- Standard VHDL library
use IEEE.STD_LOGIC_1164.ALL; -- Standard logic types
use IEEE.NUMERIC_STD.ALL;

entity ALU is -- Declaration of the ALU entity
    port (
        OP : in STD_LOGIC_VECTOR(1 downto 0); -- Operation input
        rIn : in STD_LOGIC_VECTOR(7 downto 0); -- First input operand
        rIn2 : in STD_LOGIC_VECTOR(7 downto 0); -- Second input operand
        rOut : out STD_LOGIC_VECTOR(7 downto 0)  -- Result output
    );
end ALU;

architecture behavioral of ALU is -- Architecture for the ALU entity
    signal result : std_logic_vector(7 downto 0); -- Signal for result output
begin
    process(OP, rIn, rIn2) -- Process sensitive to operation code and input operands
    begin
        if (OP = "00") then -- Addition operation
            result <= std_logic_vector(unsigned(rIn) + unsigned(rIn2)); -- Perform addition
        elsif (OP = "01") then -- Subtraction operation
            result <= std_logic_vector(unsigned(rIn) - unsigned(rIn2)); -- Perform subtraction
        elsif (OP = "10") then -- Logical AND operation
            result <= rIn and rIn2; -- Perform logical AND
        elsif (OP = "11") then -- Increment operation
            result <= std_logic_vector(unsigned(rIn) + to_unsigned(1, 7)); -- Perform increment
        else -- Default case
            result <= std_logic_vector(unsigned(rIn) + unsigned(rIn2)); -- Default to addition
        end if;
    end process;

    rOut <= result;   -- Assign result to output port
end behavioral;
```

Figure 6

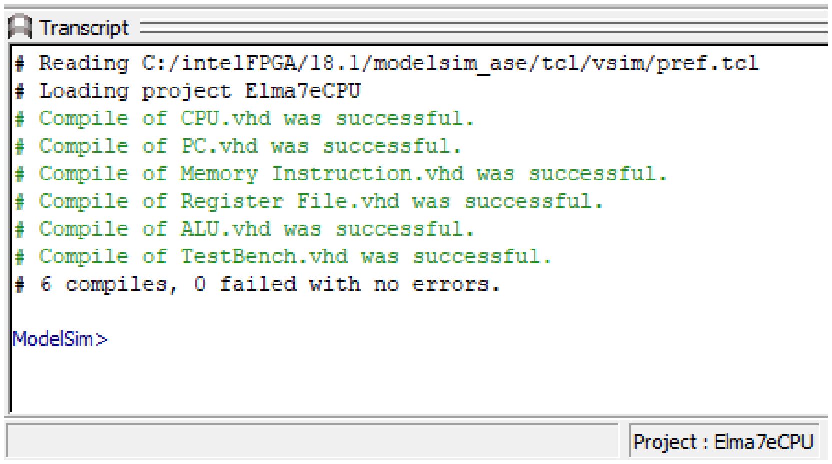
TestBench:



```vhdl
-- Testbench Entity Declaration
library IEEE; -- Standard VHDL library
use IEEE.STD_LOGIC_1164.ALL; -- Standard logic types

ENTITY elma7eCPU_test IS -- Declaration of the testbench entity
END elma7eCPU_test;

ARCHITECTURE behavior OF elma7eCPU_test IS -- Architecture for the testbench entity

    -- Component Declaration for the CPU under test
    COMPONENT elma7eCPU
        PORT (
            clk : IN std_logic; -- Clock input for the CPU
            value : OUT std_logic_vector(7 downto 0) -- Output value from the CPU
        );
    END COMPONENT;

    -- Signal Declarations
    signal clk: std_logic := '0'; -- Clock signal initialized to 0
    signal value : std_logic_vector(7 downto 0); -- Signal to capture output value from the CPU

    -- Constant for clock period
    constant clk_period : time := 10 ns; -- Clock period set to 10 ns

BEGIN

    uut: elma7eCPU PORT MAP (clk => clk, value => value); -- Instantiate the CPU under test

    -- Clock process
    clk_process : process
    begin
        clk <= '0'; -- Set clock to 0
        wait for clk_period/2; -- Wait for half clock period
        clk <= '1'; -- Set clock to 1
        wait for clk_period/2; -- Wait for half clock period
    end process clk_process;

    -- Stimulus process
    stim_proc: process
    begin
        wait for 100 ns ; -- Wait for initial stabilization
        wait for clk_period*10; -- Wait for 10 clock periods
        wait; -- Wait indefinitely
    end process stim_proc;

END behavior;
```

Figure 7

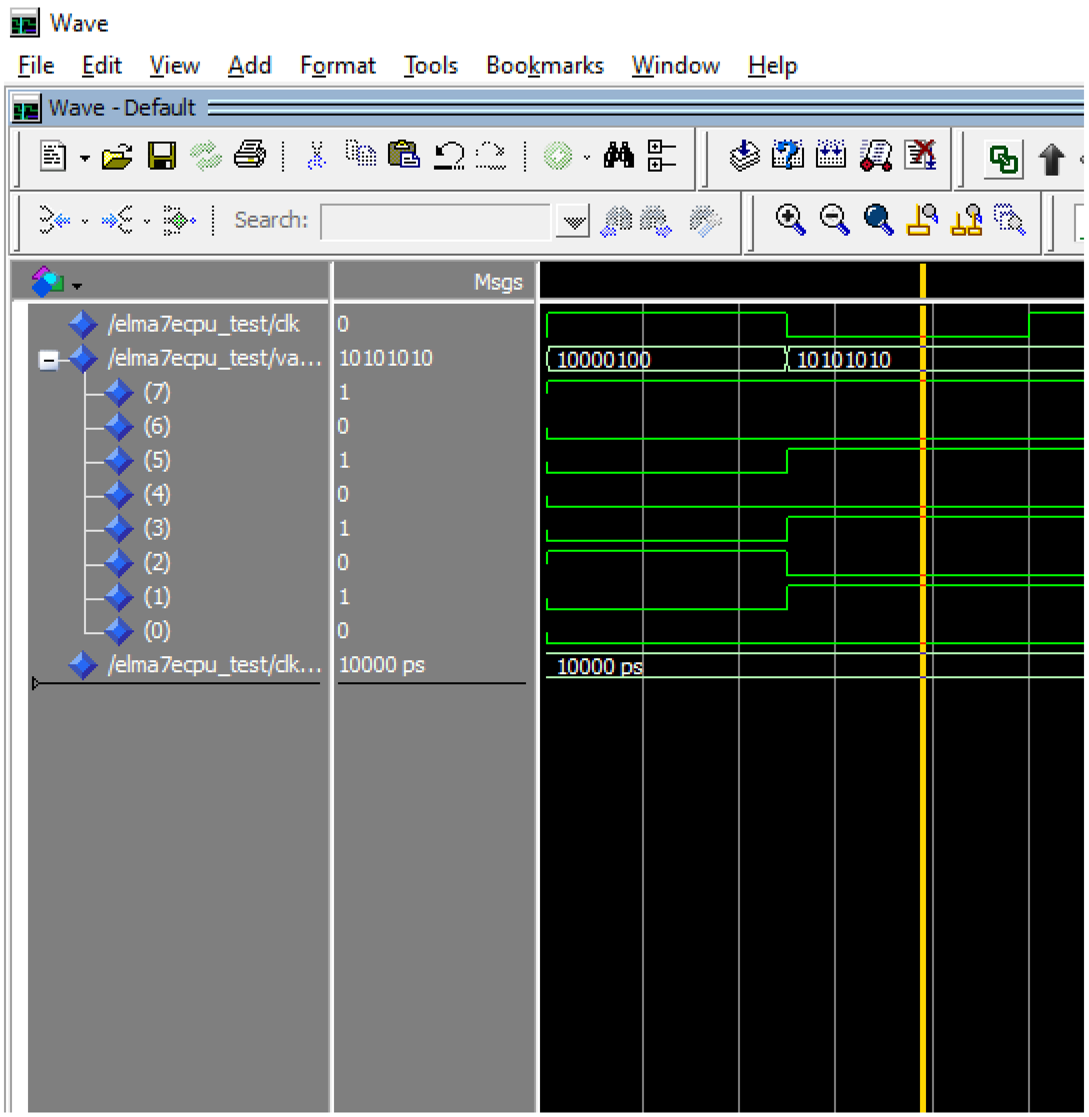
Successful compiling of the code:



```
# Reading C:/intelFPGA/18.1/modelsim_ase/tcl/vsim/pref.tcl
# Loading project Elma7eCPU
# Compile of CPU.vhd was successful.
# Compile of PC.vhd was successful.
# Compile of Memory Instruction.vhd was successful.
# Compile of Register File.vhd was successful.
# Compile of ALU.vhd was successful.
# Compile of TestBench.vhd was successful.
# 6 compiles, 0 failed with no errors.

ModelSim>
```

Figure 8

Wave output:



	Msgs
/elma7ecpu_test/clk	0
/elma7ecpu_test/va...	10101010
(7)	1
(6)	0
(5)	1
(4)	0
(3)	1
(2)	0
(1)	1
(0)	0
/elma7ecpu_test/clk...	10000 ps

```
10000100   10101010
10000 ps
```

Figure 9

The waveform output provides a visual representation of the simulation results, showing the behavior of signals over time

Circuit Diagram:

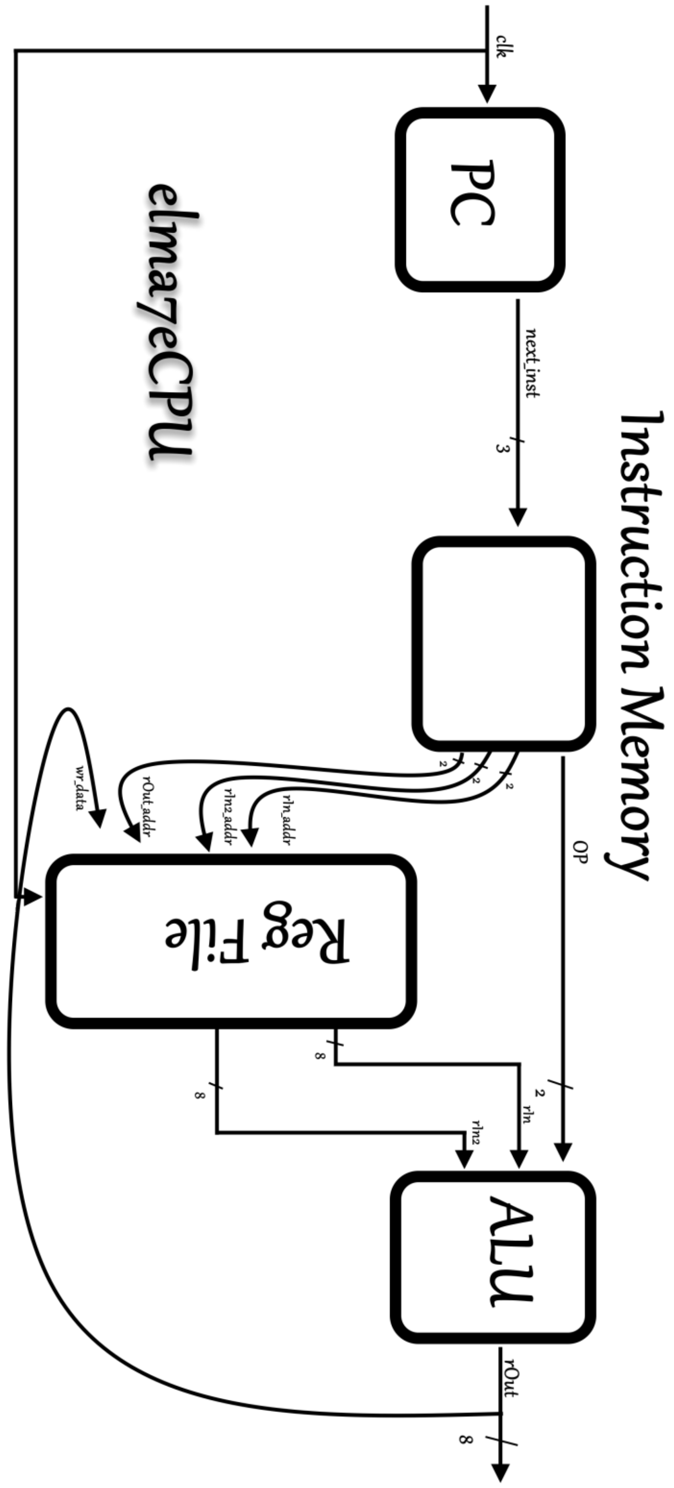


Figure 10: Diagram

Findings

After simulating the testbench and analyzing the waveform output, several key findings emerged regarding the behavior of the elma7eCPU design.

Firstly, the clock signal exhibited the expected behavior, transitioning between logic low (0) and logic high (1) states at regular intervals determined by the specified clock period. This rhythmic pulsing of the clock signal regulates the timing of the entire system, orchestrating the execution of instructions and the flow of data within the CPU.

Next, the output value from the CPU, represented by the signal "value," demonstrated varying patterns over time. These patterns corresponded to the computations and operations performed by the CPU in response to the clock signal and input stimuli. By observing the changes in the output value signal, it was possible to infer the execution of instructions and the manipulation of data within the CPU.

Furthermore, the behavior of internal signals within the CPU, such as those representing the operation code, register addresses, and data operands, revealed intricate interactions between different components of the CPU architecture. These signals facilitated the flow of control and data within the CPU, enabling the execution of instructions and the processing of data according to the specified logic and algorithms.

Additionally, the waveform output provided insights into the timing relationships between different signals within the CPU. By analyzing the relative timing of signals such as the clock, input stimuli, and internal CPU signals, it was possible to assess the synchronization and coordination of operations within the CPU, ensuring proper functioning and reliable execution of instructions.

Overall, the simulation of the testbench and analysis of the waveform output yielded valuable insights into the operation and behavior of the elma7eCPU design. These findings contribute to a deeper understanding of CPU architecture and digital system design, paving the way for further refinement and optimization of the CPU design in future iterations.

Conclusion and Recommendation

In this project, the simulation of the elma7eCPU design provided valuable insights into the intricacies of CPU architecture and digital system design. By analyzing the waveform output, we gained a comprehensive understanding of the behavior and operation of the CPU under various conditions. The rhythmic pulsing of the clock signal-regulated the timing of the entire system, orchestrating the execution of instructions and the flow of data within the CPU. Furthermore, the behavior of internal signals within the CPU revealed intricate interactions between different components, facilitating the execution of instructions and the processing of data according to the specified logic and algorithms.

The findings from the simulation underscore the importance of meticulous design and thorough testing in the development of CPU architectures. By leveraging tools such as VHDL and simulation environments, engineers can iteratively refine and optimize CPU designs, ensuring robust performance and reliability in real-world applications.

Recommendations for Improvement:

1. Optimize Clock Frequency: Evaluate the possibility of increasing the clock frequency to enhance the CPU's processing speed, while ensuring that all components can operate reliably at the higher frequency.
2. Implement Pipelining: Introduce pipelining techniques to improve instruction throughput and maximize CPU utilization. Pipelining can help reduce the overall latency of instruction execution and increase the efficiency of the CPU.
3. Enhance Instruction Set: Expand the instruction set to include a broader range of operations and functionalities, catering to diverse computing requirements and enhancing the versatility of the CPU.
4. Implement Caching Mechanisms: Integrate caching mechanisms such as instruction and data caches to reduce memory access latency and improve overall system performance. Caches can help mitigate the impact of memory latency on CPU execution time.
5. Optimize ALU Operations: Fine-tune the Arithmetic Logic Unit (ALU) operations to minimize resource utilization and improve computational efficiency. By optimizing ALU operations, the CPU can perform arithmetic and logic operations more quickly and effectively.

By implementing these recommendations, the elma7eCPU design can be further refined and optimized to meet the demands of modern computing environments. These enhancements will contribute to the development of a high-performance CPU architecture capable of powering a wide range of applications and computing tasks.