REPORT

Design and Implementation of a Simple Processor

Ver 2.0

6/1/2022

	Full name	Function		Date
Written by	Vũ Minh Tuyến Nguyễn Văn Thành			6/1/2022
Verified by	Nguyễn Khiêm Hùng			
Approved by	Nguyễn Khiêm Hùng			

Abstract (from 5 to 10 lines)
Recommended RTL level design, VHDL modeling, ModelSIM simulation and FPGA implementation of a simple 16-bit microprocessor. Processor application to build an application SoC system in the controller.

Keywords
FSM, FSMD, ModelSim, 16 bit microprocessor, VHDL, RTL

Work context

1. Learn the basics of VHDL, ModelSim
2. Analyze problem requirements (Learn 16-bit microprocessor structure)
3. Start writing learned 16-bit microprocessor component files
4. Combine component files into complete cpu file
5. Write and run testbench and check the output
6. Finalize the report based on output

Document History

Version	Time	Revised by	Description
V1.0	15/11/2021	Nguyễn Kiêm Hùng	Original Version
V2.0	10/1/2022	Vũ Minh Tuyến Nguyễn Văn Thành	2.0 Version

Table of Contents

1. Introduction

A General-Purpose Processor[1], also known as a microprocessor or Central Processing Unit (CPU), is a programmable digital system that solves complex tasks. computation in many applications. The same processor can solve computational problems in a wide variety of applications such as embedded systems in communications, industry, automobiles, etc. Choosing to use a general-purpose processor for partial implementation The functionality of the system can help the designer achieve a number of benefits. First, the cost of a single processor unit can be very low, often just under a few dollars. One reason for this low cost is that processor manufacturers can allocate NRE (Non-recurring engineering) costs for processor research and development across a large number of processors sold – often in the number of millions or billions of units.

2. Requirements

Table 1: Instruction Structure

TT	**Assembly Instruction**	**First Byte**		**Second Byte**		**Operation**
		Opcode	Operand1	Operand2		
1	MOV Rn, direct	0000	Rn	direct		Rn = M(direct)
2	MOV direct, Rn	0001	Rn	direct		M(direct) = Rn
3	MOV Rn, @Rm	0010	Rn	Rm		M(Rm) = Rn
4	MOV Rn, #immed	0011	Rn	immediate		Rn = immediate
5	ADD Rn, Rm	0100	Rn	Rm		Rn = Rn + Rm
6	SUB Rn, Rm	0101	Rn	Rm		Rn = Rn - Rm
7	JZ Rn, Addr	0110	Rn	Addr		PC = Addr only if Rn = 0
8	OR Rn, Rm	0111	Rn	Rm		Rn = Rn OR Rm

[1] "General Purpose Processor - an overview | ScienceDirect Topics." https://www.sciencedirect.com/topics/computer-science/general-purpose-processor. Ngày truy cập 10 thg 1. 2022.

9	AND Rn, Rm	1000	Rn	Rm		Rn = Rn AND Rm
10	JMP Addr	1010	Rn	Adrr		PC = Addr

The microprocessor's structure includes function blocks as shown in Figure 2, where:

The Control Unit has:

- **PC register (Program Counter)**: 16-bit, used to hold the address of the next instruction that the microprocessor will execute
- **IR register (Instruction Register)**: 16-bit, used to store instructions that the processor will execute
- **Controller**: controls the process of reading instructions from memory into the IR register, then decodes the instruction and generates datapath control signals to execute the instruction loaded in the IR.

Data processing unit [2](Datapath):

- **RF Register File (Register File)**: 16×16 bits, used to store data during computation of ALU
- **ALU (arithmetic and logic unit)**: supports operations on 16-bit data

Memory: 64K×16 bit, used to store programs and data for the microprocessor

[2] "Data processing unit - Wikipedia." https://en.wikipedia.org/wiki/Data_processing_unit. Ngày truy cập 10 thg 1. 2022.

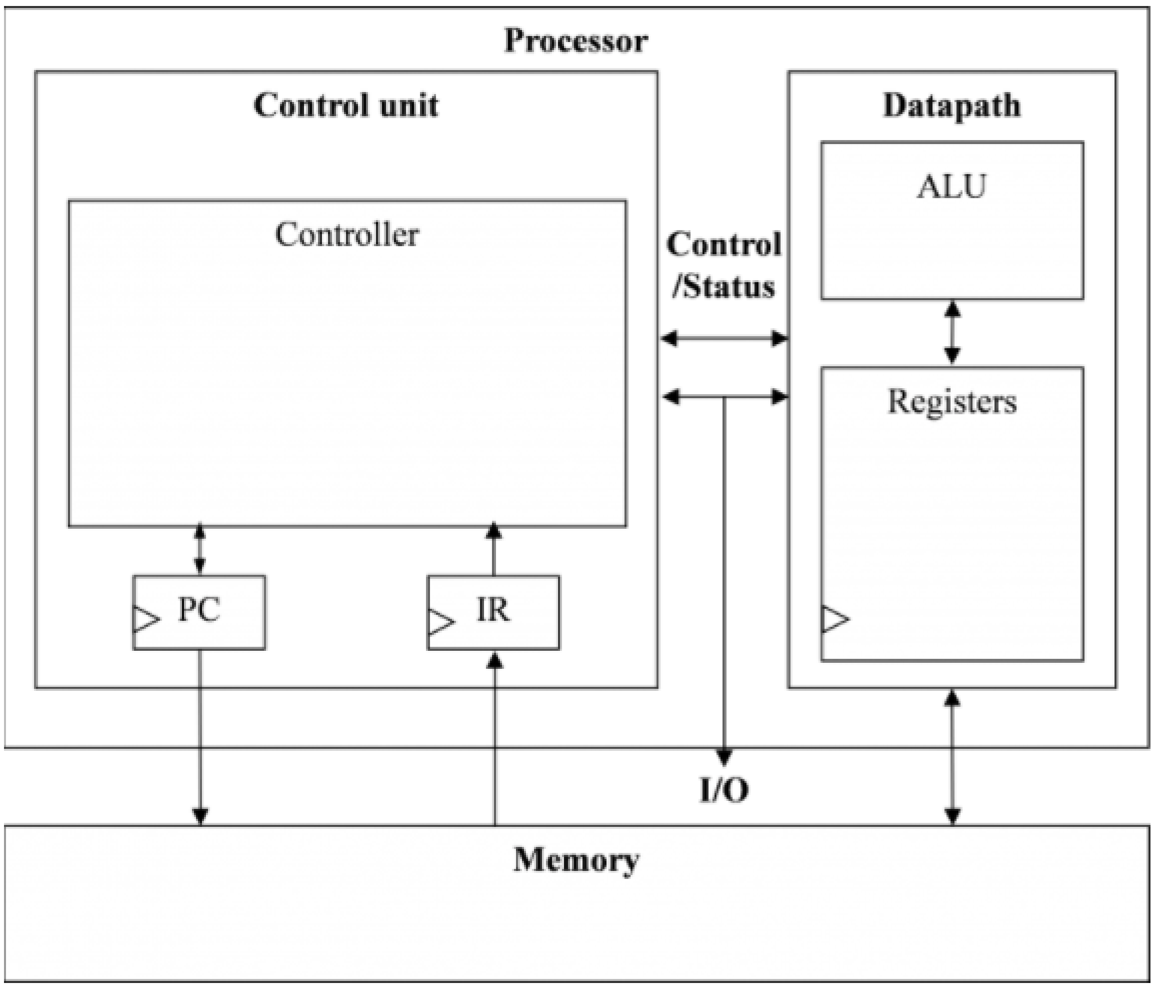


Figure 2: Structure of the basic function blocks of a microprocessor.

3. Architecture Design

We apply the integrated circuit design step-by-step process to come up with the RTL[3] level architecture of the microprocessor. Since the microprocessor does not implement any particular algorithm, but only executes a given set of instructions, we will begin the design process by directly introducing the complex state machine model. FSMD (FSM with Datapath) integration for the processor. From the FSMD model we construct the data processing unit structure – the datapath then construct the FSM state machine by removing the computations performed by the Datapath from the FSMD. The FSM part is the model that describes the operation of the controller block (Controller) in Figure 2. After having the FSM state machine model and Datapath

[3] "RTL - Wikipedia." https://en.wikipedia.org/wiki/RTL. Ngày truy cập 10 thg 1. 2022.

architecture, we can proceed to describe them using VHDL so that we can implement them. perform the simulation and verify the functional operation of the microprocessor on the computer.FSMD

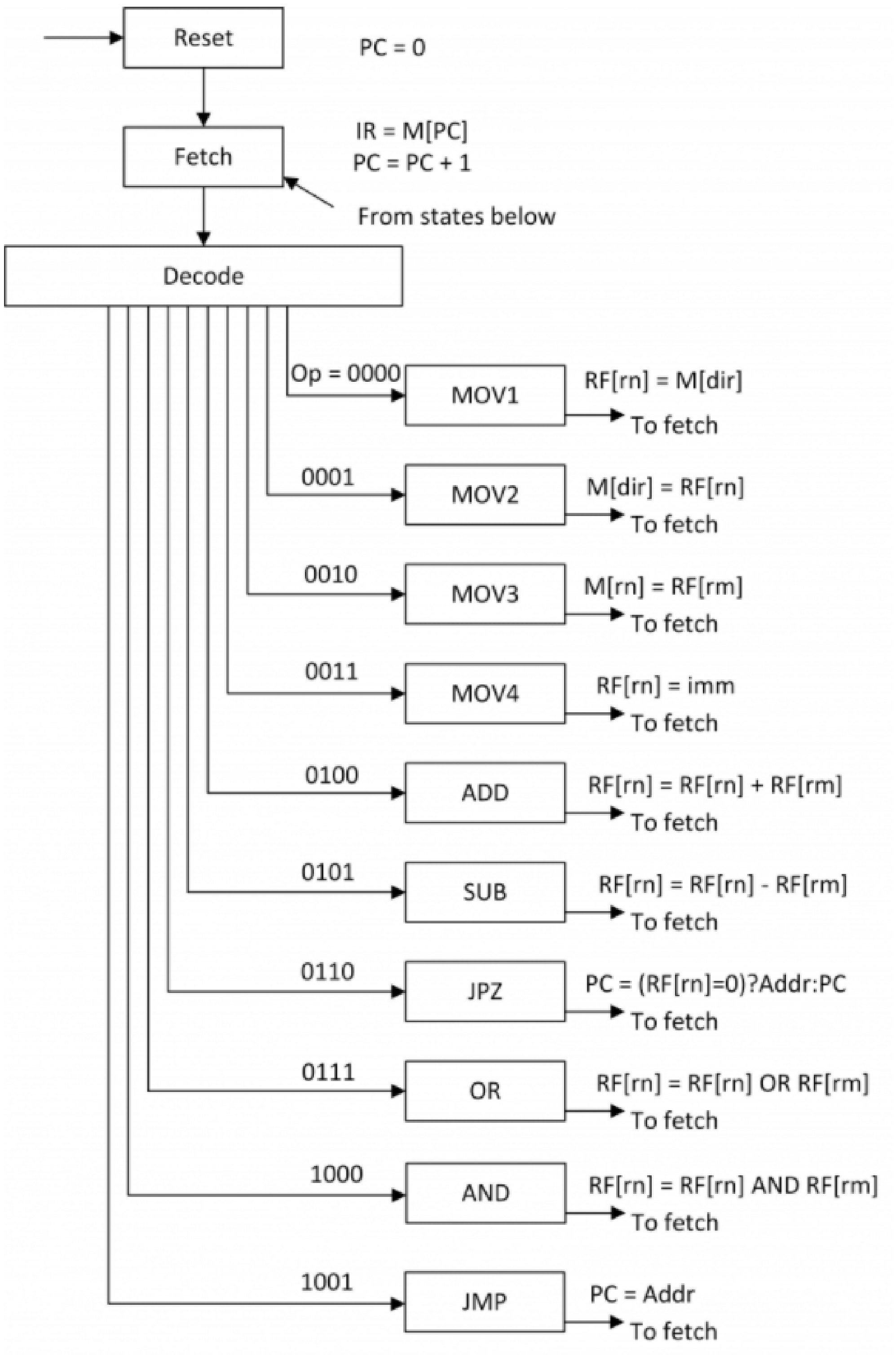


Figure 3: FSMD state machine.

The operation of the microprocessor is modeled by a complex state machine as shown in Figure 2. It is called a complex state machine because it includes more than just the control part (control corresponds to the state machine). FSM) but also the hardware that performs the calculations and data processing (Datapath).

3.1. Datapath architecture

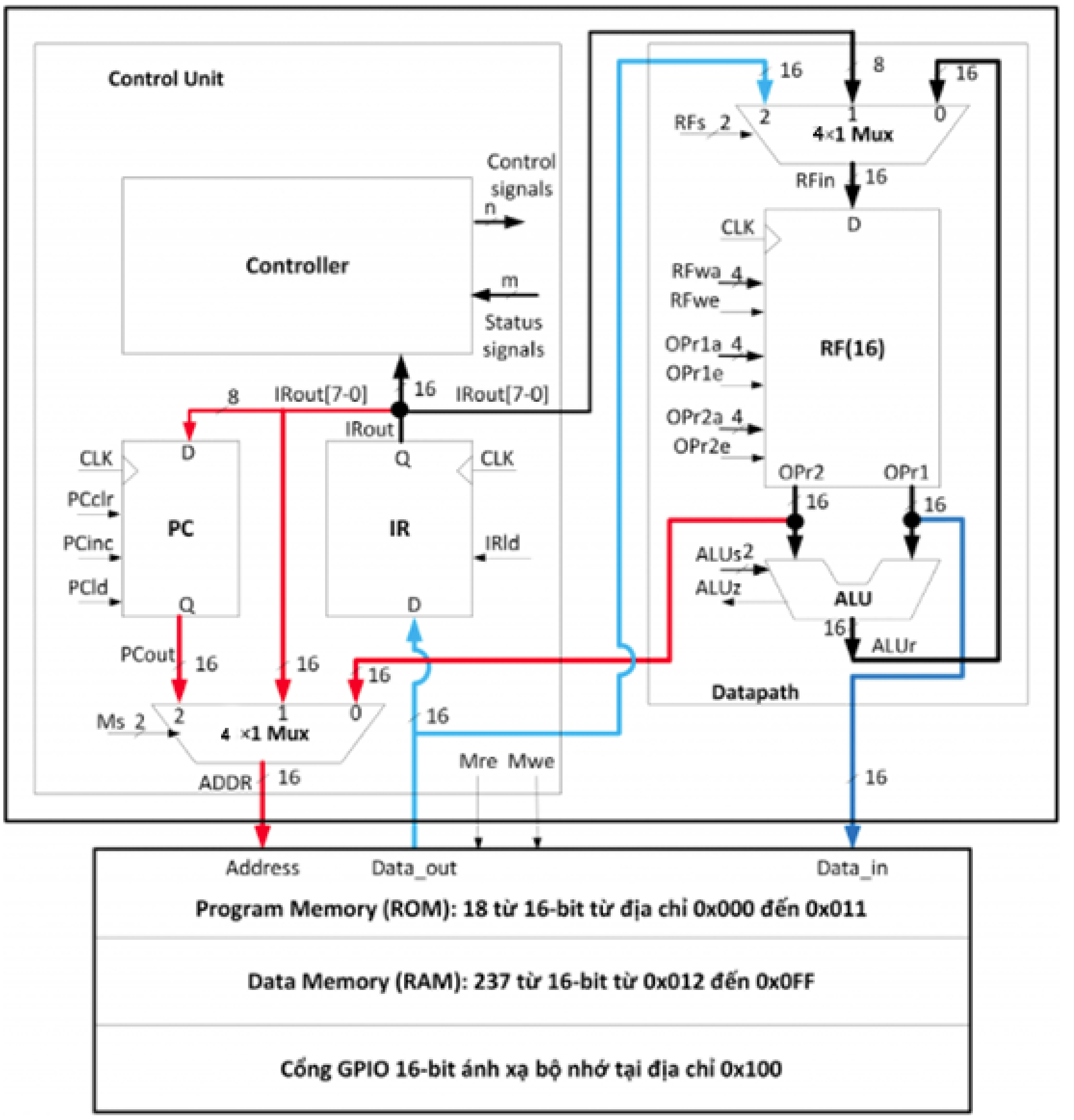


Figure 4: Datapath structure.

In a microprocessor, the Datapath data processing unit performs all calculations required by the microprocessor. Datapaths are usually composed of circuits to perform arithmetic operations (addition, subtraction, multiplication, comparison ...) and logical operations (and, or, ..), multiplexers and demultiplexers to control oriented data flow, memory elements (usually registers) to store data and intermediate results during microprocessor operations.

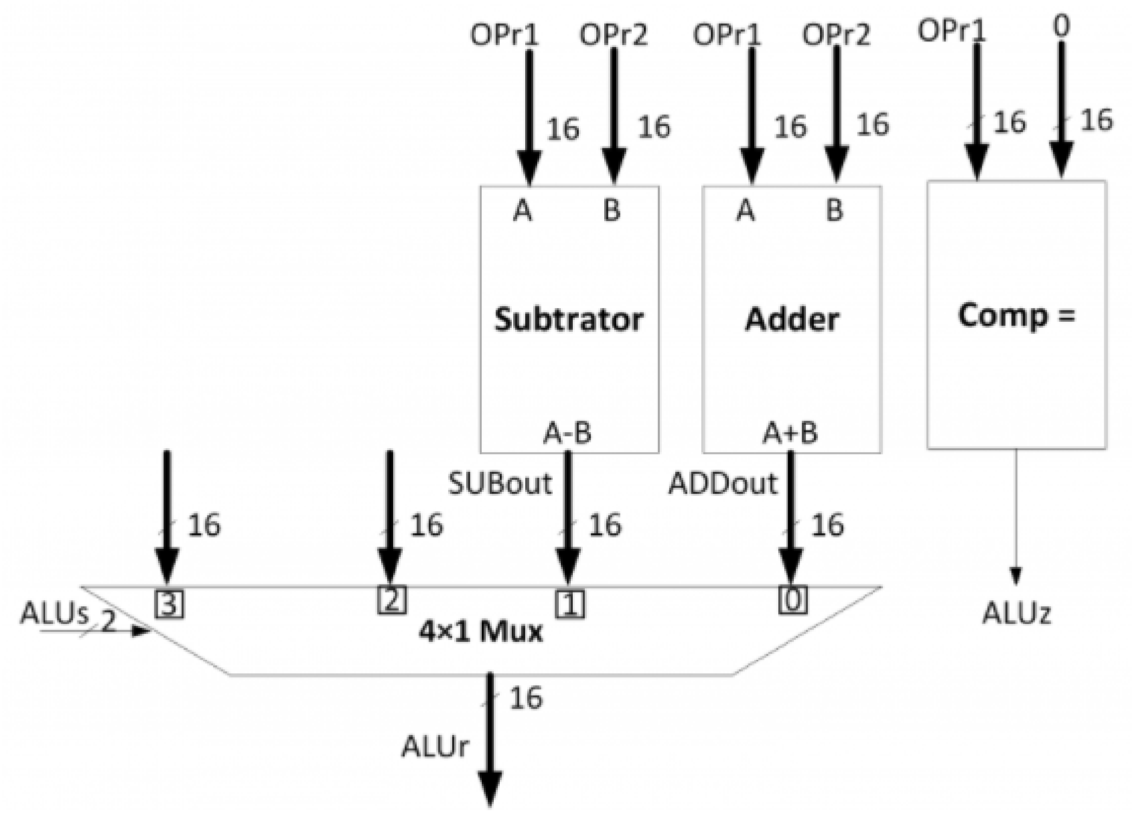


Figure 5: Calculations supported by the ALU.

ALUs	ALUr
00	OPr1 + OPr2
01	OPr1 – Opr2
10	OPr1 OR OPr2
11	OPr1 AND OPr2

3.2. Controller

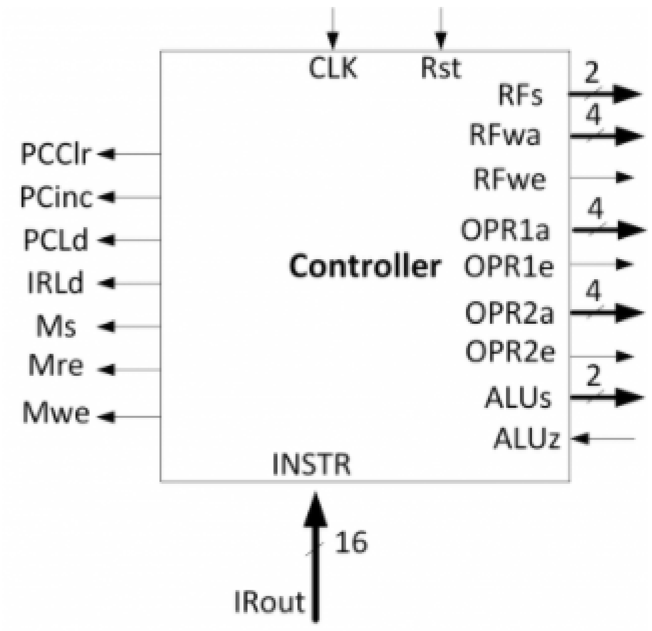
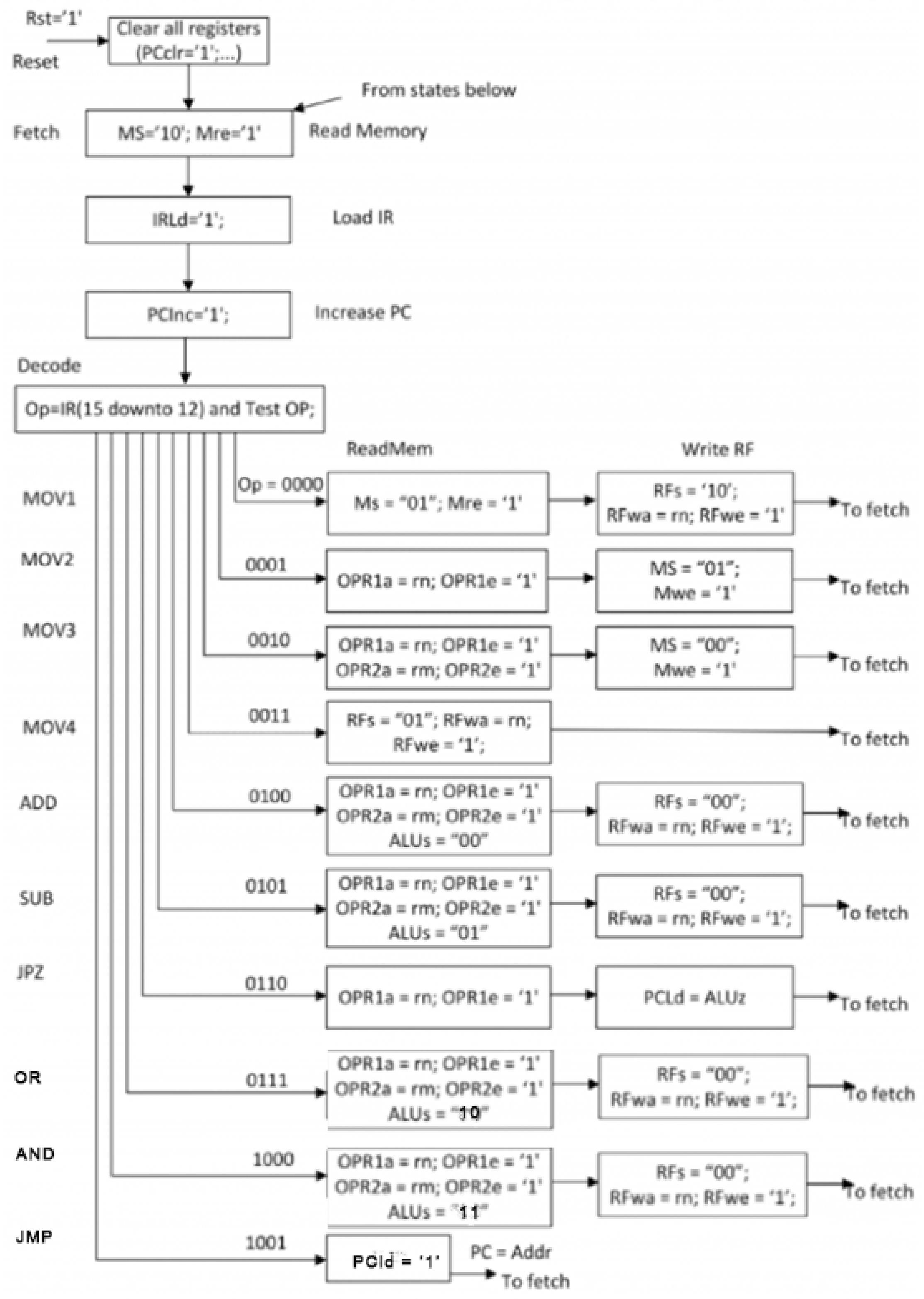


Figure 6: Controller I/O pairing interface.



Rst='1'
Reset
Clear all registers (PCclr='1';...)
From states below
Fetch
MS='10'; Mre='1'
Read Memory
IRLd='1';
Load IR
PCInc='1';
Increase PC
Decode
Op=IR(15 downto 12) and Test OP;
ReadMem
Write RF
MOV1
Op = 0000
Ms = "01"; Mre = '1'
RFs = '10'; RFwa = rn; RFwe = '1'
To fetch
MOV2
0001
OPR1a = rn; OPR1e = '1'
MS = "01"; Mwe = '1'
To fetch
MOV3
0010
OPR1a = rn; OPR1e = '1' OPR2a = rm; OPR2e = '1'
MS = "00"; Mwe = '1'
To fetch
MOV4
0011
RFs = "01"; RFwa = rn; RFwe = '1';
To fetch
ADD
0100
OPR1a = rn; OPR1e = '1' OPR2a = rm; OPR2e = '1' ALUs = "00"
RFs = "00"; RFwa = rn; RFwe = '1';
To fetch
SUB
0101
OPR1a = rn; OPR1e = '1' OPR2a = rm; OPR2e = '1' ALUs = "01"
RFs = "00"; RFwa = rn; RFwe = '1';
To fetch
JPZ
0110
OPR1a = rn; OPR1e = '1'
PCLd = ALUz
To fetch
OR
0111
OPR1a = rn; OPR1e = '1' OPR2a = rm; OPR2e = '1' ALUs = "10"
RFs = "00"; RFwa = rn; RFwe = '1';
To fetch
AND
1000
OPR1a = rn; OPR1e = '1' OPR2a = rm; OPR2e = '1' ALUs = "11"
RFs = "00"; RFwa = rn; RFwe = '1';
To fetch
JMP
1001
PCld = '1'
PC = Addr
To fetch

Figure 7: FSM state machine model of the controller.

The overall architecture of the processor

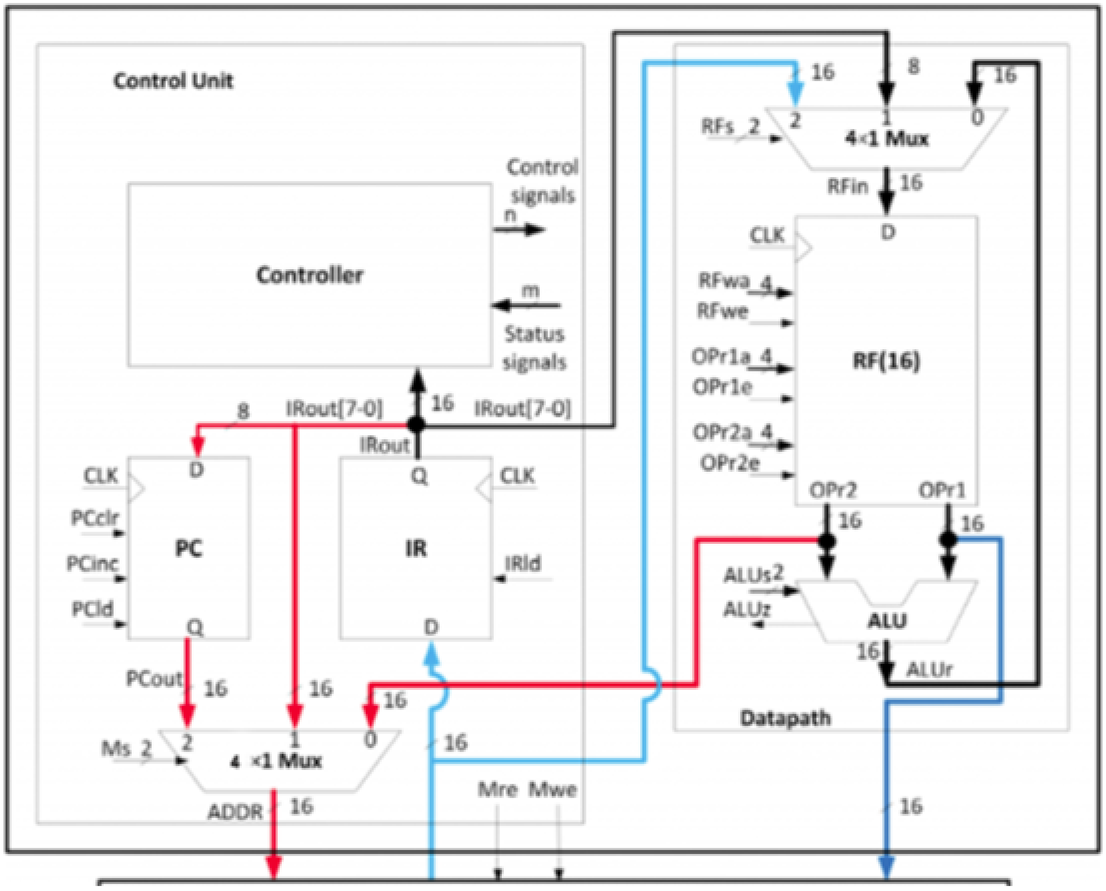


Figure 8: Complete RTL level structure of the microprocessor

4. Modeling

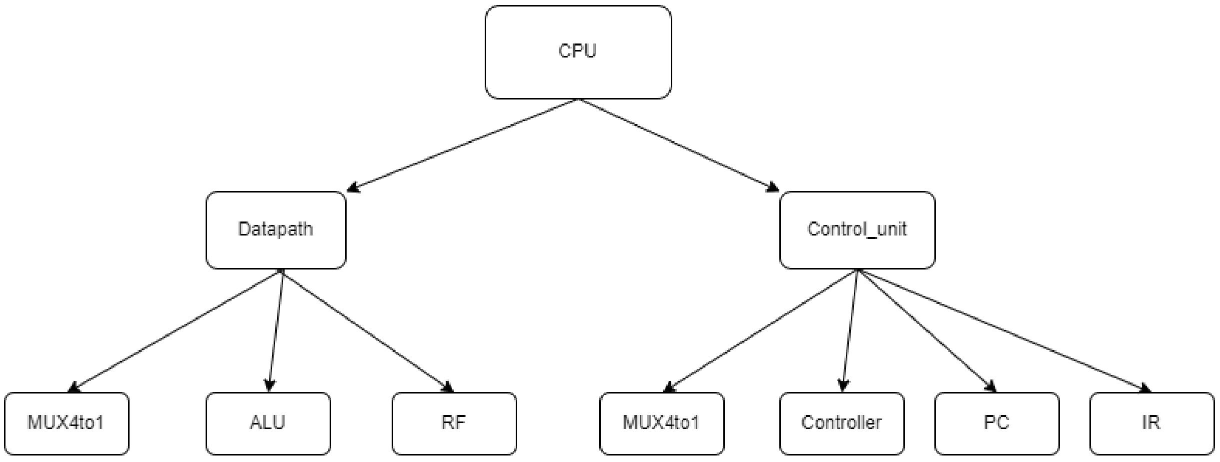


Figure 9. Organization of the VHDL files.

5. Simulation and Synthesis

5.1. Simulation results

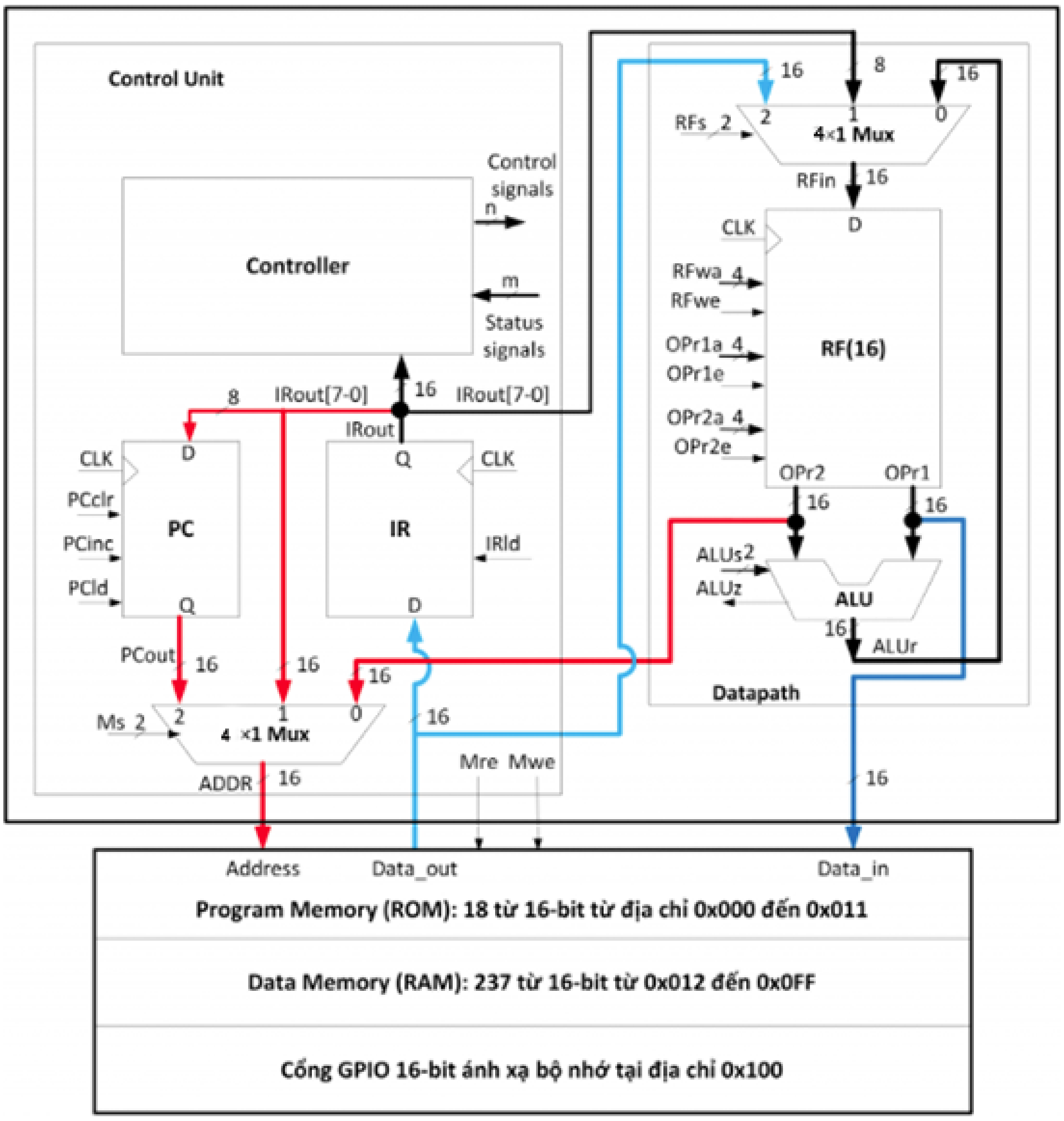
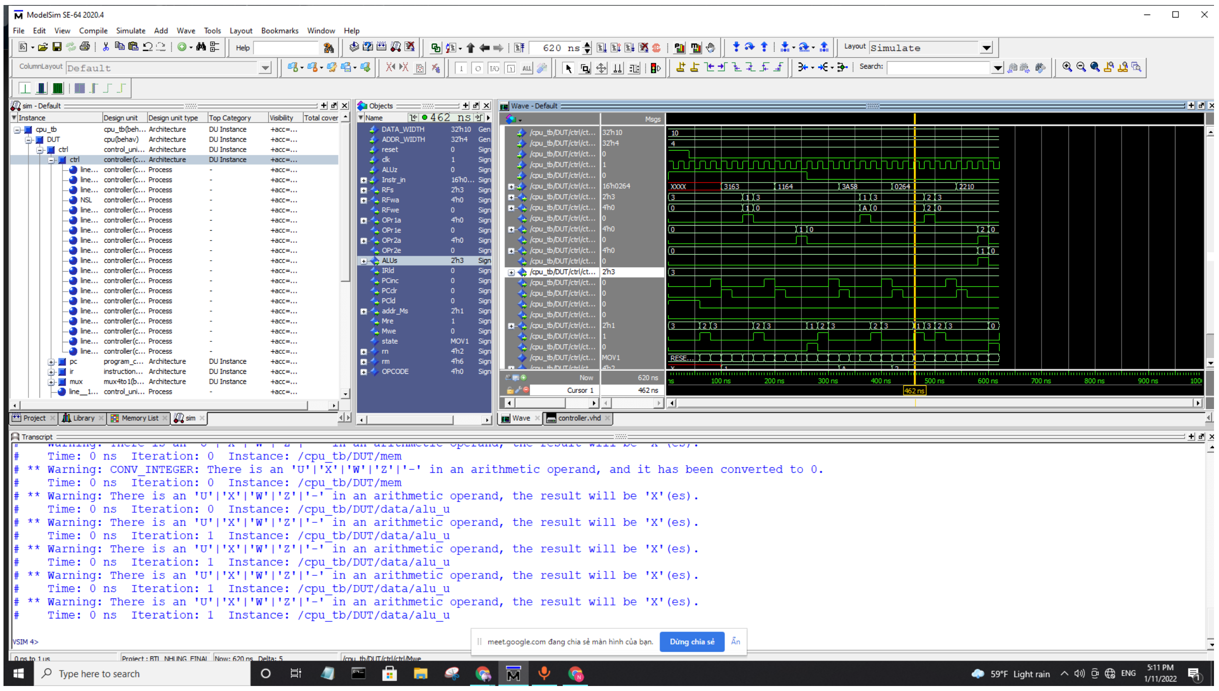


Figure 10: Testbench model to verify the functionality of the processor.



```
#    Time: 0 ns  Iteration: 0  Instance: /cpu_tb/DUT/mem
# ** Warning: CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it has been converted to 0.
#    Time: 0 ns  Iteration: 0  Instance: /cpu_tb/DUT/mem
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ns  Iteration: 0  Instance: /cpu_tb/DUT/data/alu_u
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ns  Iteration: 1  Instance: /cpu_tb/DUT/data/alu_u
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ns  Iteration: 1  Instance: /cpu_tb/DUT/data/alu_u
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ns  Iteration: 1  Instance: /cpu_tb/DUT/data/alu_u
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ns  Iteration: 1  Instance: /cpu_tb/DUT/data/alu_u

VSIM 4>
```

Figure 11: Simulation results

Appendix A: VHDL Code

1. **ALU.vhd**

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ALU IS
        GENERIC (DATA_WIDTH : INTEGER := 16);
        PORT (
                OPr1 : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
                OPr2 : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
                ALUs : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                ALUr  :  OUT  STD_LOGIC_VECTOR(DATA_WIDTH  -  1  DOWNTO
0);
                ALUz : OUT STD_LOGIC;
                ALUeq : OUT STD_LOGIC;
                ALUgt : OUT STD_LOGIC
        );
END
ALU;
ARCHITECTURE ALU OF ALU IS
        SIGNAL result : STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
BEGIN
        PROCESS (ALUs, Opr1, Opr2)
        BEGIN
```

```
                CASE(ALUs) IS
                        WHEN "00" => -- Add
                        result <= OPr1 + OPr2;
                        WHEN "01" => -- Sub
                        result <= OPr1 - OPr2;
                        WHEN "10" =>
                        result <= OPr1 OR OPr2;
                        WHEN "11" =>
                        result <= OPr1 AND OPr2;

                        WHEN OTHERS => -- Preset
                        result <= (OTHERS => '1');
                END CASE;
        END PROCESS;
        ALUr <= result;
        ALUz <= '1' WHEN OPr1 = x"0000" ELSE '0';
        ALUeq <= '1' WHEN OPr1 = OPr2 ELSE '0';
        ALugt <= '1' WHEN OPr1 > OPr2 ELSE '0';
END
ALU;
```

2. **control_unit.vhd**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY control_unit IS
```

```
        GENERIC (
                ADDR_WIDTH : INTEGER := 4;
                DATA_WIDTH : INTEGER := 16);
        PORT (
                reset : IN STD_LOGIC;
                clk : IN STD_LOGIC;
                ALUz : IN STD_LOGIC;
                ALUeq : IN STD_LOGIC;
                ALUgt : IN STD_LOGIC;

                addr_in : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
                ir_data_in : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO

                RFs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                RFwa : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO
0);
                RFwe : OUT STD_LOGIC;
                OPr1a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO
0);
                OPr1e : OUT STD_LOGIC;
                OPr2a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO
0);
                OPr2e : OUT STD_LOGIC;
                ALUs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);

                ADDR : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO
0);
                Mre : OUT STD_LOGIC;
                Mwe : OUT STD_LOGIC;
        imm : OUT STD_LOGIC_VECTOR(8 - 1 DOWNTO 0);
        op : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)


    );
END control_unit;

ARCHITECTURE behav OF control_unit IS

    COMPONENT controller
        GENERIC (
            DATA_WIDTH : INTEGER := 16;
            ADDR_WIDTH : INTEGER := 4
        );

        PORT (
            reset : IN STD_LOGIC;
            clk : IN STD_LOGIC;

            ALUz, ALUeq, ALUgt : IN STD_LOGIC;

            Instr_in : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0);
            RFs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
            RFwa : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1
DOWNTO 0);
            RFwe : OUT STD_LOGIC;
            OPr1a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1
DOWNTO 0);
```

```
                OPr1e : OUT STD_LOGIC;
                OPr2a   :   OUT   STD_LOGIC_VECTOR(ADDR_WIDTH   -   1
DOWNTO 0);
                OPr2e : OUT STD_LOGIC;
                ALUs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                IRld : OUT STD_LOGIC;
                PCinc : OUT STD_LOGIC;
                PCclr : OUT STD_LOGIC;
                PCld : OUT STD_LOGIC;
                addr_Ms : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                Mre : OUT STD_LOGIC;
                Mwe : OUT STD_LOGIC
        );
    END COMPONENT;

    COMPONENT instruction_register
        PORT (
                clk : IN STD_LOGIC;
                IRld : IN STD_LOGIC;
                IRin : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
                IRout : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT program_counter
        PORT (

                clk : IN STD_LOGIC;
                PCclr : IN STD_LOGIC;
                PCinc : IN STD_LOGIC;
                PCld : IN STD_LOGIC;
                PCd_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
                PCd_out : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
            );
        END COMPONENT;
        COMPONENT mux4to1
            GENERIC (DATA_WIDTH : INTEGER := 16);

            PORT (
                data_in_mux0, data_in_mux1, data_in_mux2, data_in_mux3 : IN
STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO 0);
                SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
                Z : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1
DOWNTO 0)
            );
        END COMPONENT;

        --signal  Instr_in : std_logic_vector(15 downto 0);
        SIGNAL IRout : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL IRout_7_to_0 : STD_LOGIC_VECTOR(7 DOWNTO 0);
        SIGNAL IRout_mux : STD_LOGIC_VECTOR(15 DOWNTO 0);
        -- tin hieu cho pc va ir
        SIGNAL PCout : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL IRld : STD_LOGIC;
        SIGNAL PCinc : STD_LOGIC;
        SIGNAL PCclr : STD_LOGIC;
        SIGNAL PCld : STD_LOGIC;
        SIGNAL addr_Ms : STD_LOGIC_VECTOR(1 DOWNTO 0);

        SIGNAL data_in_mux3_map : STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0) := x"0000";

BEGIN
        ctrl : controller
        PORT MAP(
                reset, clk, ALUz, ALUeq, ALUgt, IRout, RFs,
                RFwa, RFwe, OPr1a, OPr1e, OPr2a, OPr2e, ALUs, IRld,
                PCinc, PCclr, PCld, addr_Ms, Mre, Mwe);

        pc : program_counter
        PORT MAP(clk, PCclr, PCinc, PCld, IRout_7_to_0, PCout);

        ir : instruction_register
        PORT MAP(clk, IRld, ir_data_in, IRout);

        mux : mux4to1
        PORT MAP(addr_in, IRout_mux, PCout, data_in_mux3_map, addr_Ms,
ADDR);

        IRout_7_to_0 <= IRout(7 DOWNTO 0);
        IRout_mux <= x"00" & IRout_7_to_0;
        imm <= IRout_7_to_0;
        op <= IRout(15 DOWNTO 12);
END behav;
```

3. **controller.vhd**

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY controller IS
	GENERIC (
		DATA_WIDTH : INTEGER := 16;
		ADDR_WIDTH : INTEGER := 4
	);

	PORT (
		reset : IN STD_LOGIC;
		clk : IN STD_LOGIC;
		ALUz, ALUeq, ALUgt : IN STD_LOGIC;
		Instr_in : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
		RFs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
		RFwa : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO
0);
		RFwe : OUT STD_LOGIC;
		OPr1a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO
0);
		OPr1e : OUT STD_LOGIC;
		OPr2a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1 DOWNTO
0);
		OPr2e : OUT STD_LOGIC;
		ALUs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
```

```
        IRld : OUT STD_LOGIC;
        PCinc : OUT STD_LOGIC;
        PCclr : OUT STD_LOGIC;
        PCld : OUT STD_LOGIC;
        addr_Ms : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
        Mre : OUT STD_LOGIC;
        Mwe : OUT STD_LOGIC

    );
END controller;

ARCHITECTURE controller OF controller IS
    TYPE state_type IS (RESET_S, FETCH, Load_IR, Increase_PC, DECODE,
MOV1, MOV1a, MOV2, MOV2a,
        MOV3, MOV3a, MOV4, MOV4a, ADD, ADDa, SUB, SUBa,
        OR_S, OR_Sa, AND_S, AND_Sa, JPZ, JPZa, JMP, JMPa, JMPb,
NOPE);
    SIGNAL state : state_type;
    SIGNAL rn, rm, OPCODE : STD_LOGIC_VECTOR(3 DOWNTO 0);
BEGIN
    rn <= Instr_in(11 DOWNTO 8);
    rm <= Instr_in(7 DOWNTO 4);
    OPCODE <= INSTR_in(15 DOWNTO 12);

    NSL : PROCESS (clk, reset, OPCODE) --chuyen trang thai
    BEGIN
        IF reset = '1' THEN
            state <= RESET_S;
```

```
ELSIF clk'event AND clk = '1' THEN
CASE state IS
        WHEN RESET_S =>
                state <= FETCH;
        WHEN FETCH =>
                state <= Load_IR;
        WHEN Load_IR =>
                state <= Increase_PC;
        WHEN Increase_PC =>
                state <= DECODE;
        WHEN DECODE =>
                CASE OPCODE IS
                        WHEN "0000" =>
                                state <= MOV1;
                        WHEN "0001" =>
                                state <= MOV2;
                        WHEN "0010" =>
                                state <= MOV3;
                        WHEN "0011" =>
                                state <= MOV4;
                        WHEN "0100" =>
                                state <= ADD;
                        WHEN "0101" =>
                                state <= SUB;
                        WHEN "0110" =>
                                state <= JPZ;
                        WHEN "0111" =>
                                state <= OR_S;
                WHEN "1000" =>
                        state <= AND_S;
                WHEN "1001" => state <= JMP;
                        --when "1001"=>
                        --    state <= JMP;
                WHEN OTHERS => state <= NOPE;
        END CASE;
WHEN MOV1 =>


        state <= MOV1a;
WHEN MOV1a =>


        state <= FETCH;
WHEN MOV2 =>


        state <= MOV2a;
WHEN MOV2a =>


        state <= FETCH;
WHEN MOV3 =>
        state <= MOV3a;
WHEN MOV3a =>
        state <= FETCH;
WHEN MOV4 =>


        state <= FETCH;
WHEN ADD =>
```

```
        state <= ADDa;
WHEN ADDa =>


        state <= FETCH;
WHEN SUB =>


        state <= SUBa;
WHEN SUBa =>
        state <= FETCH;


WHEN JPZ =>
        state <= JPZa;
WHEN JPZa =>
        state <= FETCH;
WHEN OR_S =>


        state <= OR_Sa;


WHEN OR_Sa =>
        state <= FETCH;


WHEN AND_S =>


        state <= AND_Sa;
WHEN AND_Sa =>
        state <= FETCH;


WHEN OTHERS => State <= FETCH;
```

```
                    END CASE;
            END IF;
END PROCESS;
-- lenh cho PC
PCClr <= '1' WHEN (State = RESET_S) ELSE '0';
PCinc <= '1' WHEN (State = Increase_PC) ELSE '0';
PCLd <= ALUz WHEN (State = JPZa) ELSE '0';
-- lenh cho IR
IRld <= '1' WHEN (state = Load_IR) ELSE '0';
-- Address slect
WITH State SELECT
addr_Ms <= "10" WHEN Fetch,
"01" WHEN MOV1 | MOV2a,
"00" WHEN MOV3a,
"11" WHEN OTHERS;
WITH State SELECT
Mre <= '1' WHEN Fetch | MOV1,
'0' WHEN OTHERS;
WITH State SELECT
Mwe <= '1' WHEN MOV2a | MOV3a,
'0' WHEN OTHERS;
-- Write RF
WITH State SELECT
RFs <= "10" WHEN MOV1a,
"01" WHEN MOv4,
"00" WHEN ADDa | SUBa | OR_S,
"11" WHEN OTHERS;
WITH State SELECT
```

```
        RFwe <= '1' WHEN MOV1a | MOv4 | ADDa | SUBa | OR_Sa | AND_Sa,
        '0' WHEN OTHERS;
        WITH State SELECT
        RFwa <= rn WHEN MOV1a | MOv4 | ADDa | SUBa | OR_Sa | AND_Sa,
        "0000" WHEN OTHERS;
        WITH State SELECT
        OPr1e <= '1' WHEN MOV2 | MOV3 | ADD | SUB | JPZ | OR_S | AND_S,
        '0' WHEN OTHERS;
        WITH State SELECT
        OPr1a <= rn WHEN MOV2 | MOV3 | ADD | SUB | JPZ | OR_S | AND_S,
        "0000" WHEN OTHERS;
        WITH State SELECT
        OPr2e <= '1' WHEN MOV3 | ADD | SUB | OR_S | AND_S,
        '0'WHEN OTHERS;
        WITH State SELECT
        OPr2a <= rm WHEN MOV3 | ADD | SUB | OR_S | AND_S,
        "0000" WHEN OTHERS;
        WITH State SELECT
        ALUs <= "00" WHEN ADD | ADDa,
        "01" WHEN SUB | SUBa,
        "10" WHEN OR_S,
        "11" WHEN OTHERS;
END controller;
```

4. **cpu.vhd**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY cpu IS
        GENERIC (
                DATA_WIDTH : INTEGER := 16;
                ADDR_WIDTH : INTEGER := 4
        );
        PORT (
                clk : IN STD_LOGIC;
                reset : IN STD_LOGIC;

                address_cpu : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0);

                Mre_cpu : OUT STD_LOGIC;
                Mwe_cpu : OUT STD_LOGIC;

                data_out : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0);
                data_in : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0)
        );

END cpu;

ARCHITECTURE behav OF cpu IS
        COMPONENT control_unit
                GENERIC (ADDR_WIDTH : INTEGER := 4);
        PORT (
                reset : IN STD_LOGIC;
                clk : IN STD_LOGIC;
                ALUz, ALUeq, ALUgt : IN STD_LOGIC;
                addr_in : IN STD_LOGIC_VECTOR(16 - 1 DOWNTO 0);
                ir_data_in : IN STD_LOGIC_VECTOR(16 - 1 DOWNTO
0);

                RFs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                RFwa : OUT STD_LOGIC_VECTOR(ADDR_WIDTH - 1
DOWNTO 0);
                RFwe : OUT STD_LOGIC;
                OPr1a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH -
1 DOWNTO 0);
                OPr1e : OUT STD_LOGIC;
                OPr2a : OUT STD_LOGIC_VECTOR(ADDR_WIDTH -
1 DOWNTO 0);
                OPr2e : OUT STD_LOGIC;
                ALUs : OUT STD_LOGIC_VECTOR(1 DOWNTO 0);

                ADDR : OUT STD_LOGIC_VECTOR(DATA_WIDTH -
1 DOWNTO 0);


                Mre_cu : OUT STD_LOGIC;
                Mwe_cu : OUT STD_LOGIC;

                imm : OUT STD_LOGIC_VECTOR(8 - 1 DOWNTO 0);
                op : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
```

```
	END COMPONENT;

	COMPONENT datapath
		GENERIC (
			DATA_WIDTH : INTEGER := 16;
			ADDR_WIDTH : INTEGER := 4
		);
		PORT (
			rst : IN STD_LOGIC;
			clk : IN STD_LOGIC;
			imm : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
			data_in_mux2                    :                   IN
STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
			RFs : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			ALUs : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			ALUz : OUT STD_LOGIC;
			ALUeq : OUT STD_LOGIC;
			ALUgt : OUT STD_LOGIC;

			RFwa : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
			RFwe : IN STD_LOGIC;
			OPr1a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
			OPr1e : IN STD_LOGIC;
			OPr2a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
			OPr2e : IN STD_LOGIC;
			add_out : OUT STD_LOGIC_VECTOR(DATA_WIDTH -
1 DOWNTO 0);
```

```
                    data_out : OUT STD_LOGIC_VECTOR(DATA_WIDTH -
1 DOWNTO 0)
                );
        END COMPONENT;

        COMPONENT dpmem
                GENERIC (
                        DATA_WIDTH : INTEGER := 16;
                        ADDR_WIDTH : INTEGER := 16
                );

                PORT (
                        Clk : IN STD_LOGIC;
                        Reset : IN STD_LOGIC;
                        addr  :  IN  STD_LOGIC_VECTOR(ADDR_WIDTH  -  1
DOWNTO 0);
                        Wen : IN STD_LOGIC;
                        Datain  :  IN  STD_LOGIC_VECTOR(DATA_WIDTH  -  1
DOWNTO 0) := (OTHERS => '0');
                        Ren : IN STD_LOGIC;
                        Dataout  : OUT STD_LOGIC_VECTOR(DATA_WIDTH -
1 DOWNTO 0)
                );
        END COMPONENT;

        SIGNAL address : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL imm : STD_LOGIC_VECTOR(7 DOWNTO 0);
        SIGNAL OPr2 : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL Mre : STD_LOGIC;
```

```
	SIGNAL Mwe : STD_LOGIC;
	SIGNAL data_out_mem : STD_LOGIC_VECTOR(15 DOWNTO 0);
	SIGNAL data_in_mem : STD_LOGIC_VECTOR(15 DOWNTO 0);

	SIGNAL RFs : STD_LOGIC_VECTOR(1 DOWNTO 0);
	SIGNAL RFwa : STD_LOGIC_VECTOR(4 - 1 DOWNTO 0);
	SIGNAL RFwe : STD_LOGIC;

	SIGNAL OPr1a : STD_LOGIC_VECTOR(4 - 1 DOWNTO 0);
	SIGNAL OPr1e : STD_LOGIC;
	SIGNAL OPr2a : STD_LOGIC_VECTOR(4 - 1 DOWNTO 0);
	SIGNAL OPr2e : STD_LOGIC;
	SIGNAL ALUs : STD_LOGIC_VECTOR(1 DOWNTO 0);
	SIGNAL ALUz : STD_LOGIC;
	SIGNAL ALUeq : STD_LOGIC;
	SIGNAL ALUgt : STD_LOGIC;

	SIGNAL op : STD_LOGIC_VECTOR(3 DOWNTO 0);

BEGIN
	ctrl : ENTITY work.control_unit
		PORT MAP(
			reset, clk, ALUz, ALUeq, ALUgt, OPr2, data_out_mem,
RFs, RFwa,
			RFwe, OPr1a, OPr1e, Opr2a, OPr2e, ALUs, address, Mre,
Mwe, imm, op);

	data : datapath
```

```
        PORT MAP(
                reset, clk, imm, data_out_mem, RFs, ALUs, ALUz, ALUeq,
ALUgt, RFwa, RFwe,
                OPr1a, OPr1e, Opr2a, OPr2e, OPr2, data_in_mem);


        mem : dpmem
        PORT MAP(clk, reset, address, Mwe, data_in_mem, Mre,
data_out_mem);
        address_cpu <= address;
        Mre_cpu <= Mre;
        Mwe_cpu <= Mwe;


        data_out <= data_out_mem;
        data_in <= data_in_mem;
END behav;
```

5. **cpu_tb.vhd**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY cpu_tb IS
END cpu_tb;

ARCHITECTURE behav OF cpu_tb IS
        COMPONENT cpu
                PORT (
                        clk : IN STD_LOGIC;
                        reset : IN STD_LOGIC;
                        address_cpu : OUT STD_LOGIC_VECTOR(15
DOWNTO 0);

                        Mre_cpu : OUT STD_LOGIC;
                        Mwe_cpu : OUT STD_LOGIC;

                        data_out_mem_t : OUT STD_LOGIC_VECTOR(15
DOWNTO 0);
                        data_in_mem_t : OUT STD_LOGIC_VECTOR(15
DOWNTO 0)
                );
        END COMPONENT;

        CONSTANT clk_period : TIME := 15 ns;

        SIGNAL clk : STD_LOGIC := '1';
        SIGNAL reset : STD_LOGIC := '1';
        SIGNAL address_cpu : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL Mre_cpu : STD_LOGIC;
        SIGNAL Mwe_cpu : STD_LOGIC;
        SIGNAL data_out_mem_t : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL data_in_mem_t : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN

        clk <= NOT clk AFTER clk_period / 2;

        DUT : ENTITY work.cpu
                PORT MAP(clk, reset, address_cpu, Mre_cpu, Mwe_cpu,
data_out_mem_t, data_in_mem_t);
```

```
        PROCESS
        BEGIN
            reset <= '1';
            WAIT FOR clk_period * 2;

            reset <= '0';
            WAIT FOR clk_period * 110;
        END PROCESS;

    END behav;
```

6. **datapath.vhd**

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.std_logic_arith.ALL;

ENTITY datapath IS
    GENERIC (
        DATA_WIDTH : INTEGER := 16;
        ADDR_WIDTH : INTEGER := 4
    );
    PORT (
        rst : IN STD_LOGIC;
        clk : IN STD_LOGIC;
        imm : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
        data_in_mux2   :   IN   STD_LOGIC_VECTOR(DATA_WIDTH   -   1
DOWNTO 0);
```

```
		RFs : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
		ALUs : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
		ALUz : OUT STD_LOGIC;
		ALUeq : OUT STD_LOGIC;
		ALUgt : OUT STD_LOGIC;

		RFwa : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
		RFwe : IN STD_LOGIC;
		OPr1a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
		OPr1e : IN STD_LOGIC;
		OPr2a : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
		OPr2e : IN STD_LOGIC;

		add_out : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO
0);
		data_out : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO
0)
	);
END datapath;

ARCHITECTURE struct OF datapath IS
	COMPONENT alu
		PORT (
			OPr1 : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0);
			OPr2 : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0);
			ALUs : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			ALUz : OUT STD_LOGIC;
```

```
            ALUeq : OUT STD_LOGIC;
            ALUgt : OUT STD_LOGIC;
            ALUr : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1
DOWNTO 0)
        );
    END COMPONENT;

    COMPONENT mux4to1
        GENERIC (DATA_WIDTH : INTEGER := 16);
        PORT (
            data_in_mux0, data_in_mux1, data_in_mux2, data_in_mux3 : IN
STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO 0);
            SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            Z : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1
DOWNTO 0)
        );
    END COMPONENT;
    COMPONENT register_file
        GENERIC (
            DATA_WIDTH : INTEGER := 16;
            ADDR_WIDTH : INTEGER := 4
        );
        PORT (
            reset : IN STD_LOGIC;
            clk : IN STD_LOGIC;
            RFin : IN STD_LOGIC_VECTOR (DATA_WIDTH - 1
DOWNTO 0);
            RFwa : IN STD_LOGIC_VECTOR (ADDR_WIDTH - 1
DOWNTO 0);
```

```
                        RFwe : IN STD_LOGIC;
                        OPr1a   :   IN   STD_LOGIC_VECTOR   (ADDR_WIDTH   -   1
DOWNTO 0);
                        OPr1e : IN STD_LOGIC;
                        OPr2a   :   IN   STD_LOGIC_VECTOR   (ADDR_WIDTH   -   1
DOWNTO 0);
                        OPr2e : IN STD_LOGIC;
                        OPr1   :   OUT   STD_LOGIC_VECTOR   (DATA_WIDTH   -   1
DOWNTO 0);
                        OPr2   :   OUT   STD_LOGIC_VECTOR   (DATA_WIDTH   -   1
DOWNTO 0));
        END COMPONENT;
        SIGNAL ALUr : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL RFin : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL   data_in_mux3   :   STD_LOGIC_VECTOR(15   DOWNTO   0)   :=
x"0000";

        SIGNAL o1 : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL o2 : STD_LOGIC_VECTOR(15 DOWNTO 0);
        SIGNAL data_in_mux1 : STD_LOGIC_VECTOR(15 DOWNTO 0);

BEGIN
        data_in_mux1 <= x"00" & imm;

        mux : mux4to1
        PORT MAP(ALUr, data_in_mux1, data_in_mux2, data_in_mux3, RFs, RFin);

        rf_u : register_file
        PORT MAP(rst, clk, RFin, RFwa, RFwe, OPr1a, OPr1e, OPr2a, OPr2e, o1, o2);
        alu_u : ALU
        PORT MAP(o1, o2, ALUs, ALUz, ALUeq, ALUgt, ALUr);
        add_out <= o2;
        data_out <= o1;

END struct;
```

7. **dpmem.vhd**

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

-------------------------------------------------------------------------------
-- Synchronous Dual Port Memory
-------------------------------------------------------------------------------
entity dpmem is
  generic (
    DATA_WIDTH       :    integer   := 16;     -- Word Width
    ADDR_WIDTH       :    integer   := 16      -- Address width
    );

  port (
    -- Writing
    Clk            : in  std_logic;          -- clock
        Reset           : in  std_logic; -- Reset input
    addr            : in  std_logic_vector(ADDR_WIDTH -1 downto 0);   --  Address
        -- Writing Port
        Wen             : in  std_logic;         -- Write Enable
     Datain           : in  std_logic_vector(DATA_WIDTH -1 downto 0) := (others => '0');
-- Input Data
    -- Reading Port

    Ren             : in  std_logic;         -- Read Enable
    Dataout          : out std_logic_vector(DATA_WIDTH -1 downto 0)   -- Output data

    );
end dpmem;

architecture dpmem_arch of dpmem is

  type DATA_ARRAY is array (integer range <>) of std_logic_vector(DATA_WIDTH
-1 downto 0); -- Memory Type
   signal    M        :       DATA_ARRAY(0 to (2**ADDR_WIDTH) -1) := (others =>
(others => '0'));  -- Memory model
-- you can add more code for your application by increase the PM_Size
  constant PM_Size : Integer := 4; -- Size of program memory :(range 255 downto 0 )
    --type P_MEM is array (0 to PM_Size-1) of std_logic_vector(DATA_WIDTH -1
downto 0); -- Program Memory
  constant PM : DATA_ARRAY(0 to (2**PM_Size) - 1) := (
-- Machine code for your application is initialized here
    X"3A58",   -- mov4 RF(10) = 58(hex)
    X"8A10",   -- and RF(10) = RF(10) and RF(1)
    X"3163",   -- mov4 RF(1) = 63(hex)
    X"4A10",    -- add RF(10) = RF(10) + RF(1)
    X"5A10",   -- sub RF(10) = RF(10) - RF(1)
    X"7210",    --or RF(2) = RF(2) or RF(1)
   X"2210",   -- mov3 M((rm==1)) = (rn==2)   ghi vào memory can 1 dia chi => ghi
gia tri rf2 vao  M cua rf1
   X"0264",   -- mov1 rf(2) = M (64(hex));   thanh ghi rf2 duoc gan bang gia tri M cua
tai dia chi 64
  X"1164",  -- mov2 M(64(hex)) = RF(1);    ghi gia tri RF1 vao M 64
  X"6302",  --jz RF(3) addr = 2 quay lai lenh 3163
  X"9903",  --jmp rn =9  addr = 3
  others => x"0000");

 -- signal checking proper address
 signal en : std_logic:='0';
begin  -- dpmem_arch

         -- checking address process
 check_addr : process(addr)
 begin
            if(conv_integer(addr) >= (2**PM_Size) and conv_integer(addr) <=
(2**ADDR_WIDTH) -1) then
    en <= '1';
  else
    en <= '0';
  end if;
 end process;
 --  Read/Write process

 RW_Proc : process (clk, Reset)
 begin
  if Reset = '1' then
```

```
          Dataout <= (others => '0');
          M(0 to (2**PM_Size)-1) <= PM; -- initialize program memory
      elsif (clk'event and clk = '1') then   -- rising clock edge
        if Wen = '1' and en = '1' then
                              M(conv_integer(addr))        <= Datain; -- ensure that data cant
overwrite on program
        else
                             if Ren = '1' then
                                     Dataout <= M(conv_integer(addr));
                             else
                                  Dataout <= (others => 'Z');
                           end if;
                  end if;
      end if;
  end process  RW_Proc;


end dpmem_arch;
```

8. **instrction_register.vhd**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;


ENTITY instruction_register IS
        GENERIC (DATA_WIDTH : INTEGER := 16);
        PORT (
                clk : IN STD_LOGIC;
                IRld : IN STD_LOGIC;
                IRin : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
```

```
		IRout : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0)
	);
END instruction_register;

ARCHITECTURE behav OF instruction_register IS
	SIGNAL data : STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
BEGIN
	PROCESS (clk)
	BEGIN
		IF (clk = '1' AND clk'event) THEN
			IF (IRld = '1') THEN
				data <= IRin;
			END IF;
		END IF;
	END PROCESS;
	IRout <= data;
END behav;
```

9. **MUX4to1.vhd**

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE IEEE.std_logic_unsigned.ALL;

ENTITY mux4to1 IS
	GENERIC (DATA_WIDTH : INTEGER := 16);
	PORT (
		data_in_mux0, data_in_mux1, data_in_mux2, data_in_mux3 : IN
STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO 0);
		SEL : IN STD_LOGIC_VECTOR (1 DOWNTO 0);
		Z : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO 0)
	);
END mux4to1;

ARCHITECTURE bev OF mux4to1 IS
BEGIN
	WITH sel SELECT
		z <= data_in_mux0 WHEN "00",
		data_in_mux1 WHEN "01",
		data_in_mux2 WHEN "10",
		data_in_mux3 WHEN OTHERS;
END bev;
```

10. **program_counter.vhd**

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY program_counter IS
	PORT (
		clk : IN STD_LOGIC;
		PCclr : IN STD_LOGIC;
		PCinc : IN STD_LOGIC;
		PCld : IN STD_LOGIC;

		PCd_in : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
```

```
                PCd_out : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
        );
END program_counter;

ARCHITECTURE behav OF program_counter IS
        SIGNAL data : STD_LOGIC_VECTOR(15 DOWNTO 0);
BEGIN
        PROCESS (clk, PCclr)
        BEGIN
                IF (PCclr = '1') THEN
                        data <= X"0000";
                ELSIF (clk = '1'AND clk'event) THEN
                        IF (PCld = '1') THEN
                                data <= X"00" & PCd_in;
                        END IF;
                        IF (PCinc = '1') THEN
                                data <= data + "1";
                        END IF;
                END IF;
        END PROCESS;
        PCd_out <= data;

END behav;
```

11. register_file.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
ENTITY register_file IS
        GENERIC (
                DATA_WIDTH : INTEGER := 16;
                ADDR_WIDTH : INTEGER := 4
        );
        PORT (
                reset : IN STD_LOGIC;
                clk : IN STD_LOGIC;
                RFin : IN STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO 0);
                RFwa : IN STD_LOGIC_VECTOR (ADDR_WIDTH - 1 DOWNTO 0);
                RFwe : IN STD_LOGIC;
                OPr1a : IN STD_LOGIC_VECTOR (ADDR_WIDTH - 1 DOWNTO 0);
                OPr1e : IN STD_LOGIC;
                OPr2a : IN STD_LOGIC_VECTOR (ADDR_WIDTH - 1 DOWNTO 0);
                OPr2e : IN STD_LOGIC;
                OPr1 : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO
0);
                OPr2 : OUT STD_LOGIC_VECTOR (DATA_WIDTH - 1 DOWNTO 0)
        );
END register_file;
ARCHITECTURE register_file OF register_file IS
        TYPE DATA_ARRAY IS ARRAY (INTEGER RANGE <>) OF
STD_LOGIC_VECTOR(16 - 1 DOWNTO 0);
        SIGNAL RF : DATA_ARRAY(0 TO 15) := (OTHERS => (OTHERS => '0'));
BEGIN
        PROCESS (clk, reset)
        BEGIN
                IF reset = '1' THEN
```

```
                    --
                    OPr1 <= (OTHERS => '0');
                    OPr2 <= (OTHERS => '0');
                    RF <= (OTHERS => (OTHERS => '0'));
                    --
                    ELSIF clk'event AND clk = '1' THEN
                    IF RFwe = '1' THEN
                        RF(conv_integer(RFwa)) <= RFin;
                    END IF;
                    --
                    IF OPr1e = '1' THEN
                        OPr1 <= RF(conv_integer(OPr1a));
                    END IF;
                    --
                    IF OPr2e = '1' THEN
                        OPr2 <= RF(conv_integer(OPr2a));
                    END IF;
                END IF;
            END PROCESS;
END register_file;
```