LAB # 10:
Follow the procedure to reproduce the Control unit and Data Memory of MIPS 32-bit microprocessor using VHDL and implementation on FPGA

Objective

- To design the Control and Data memory module of a single-cycle MIPS 32-bit processor

Pre-Lab

In this lab you will be designing the control unit and data memory of MIPS-32 processor. The control module and its control signals are shown in light blue color.

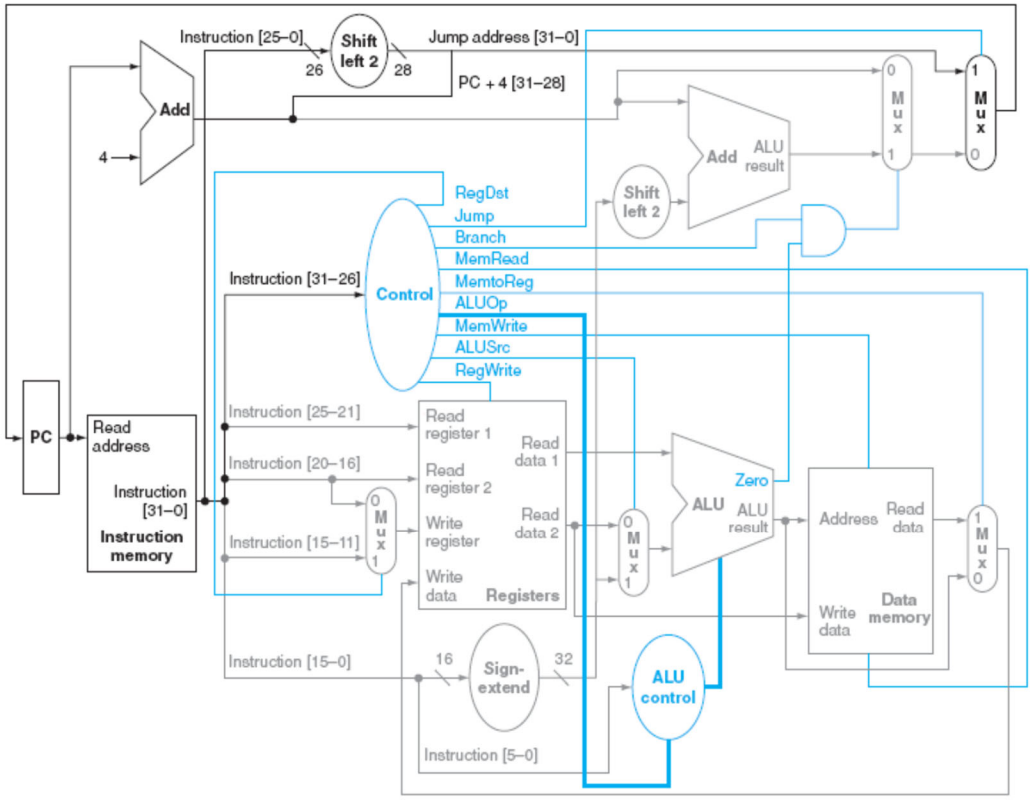


Figure 10-1 Single cycle MIPS datapath

The control module (control.vhd)

The control module simply generates the control signals based on the instruction op code.

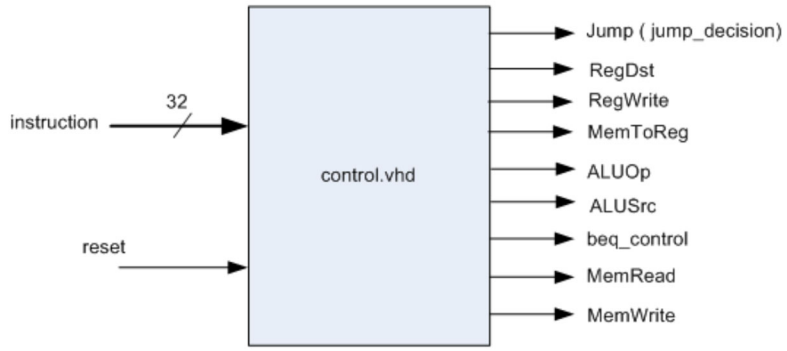


Figure 10.2 The control module

The implementation of the control module basically follows the table given in the textbook. It is relatively simple to implement this module.

Pre-Lab Task

Task 1: Control Module Functionality

Below gives a partially implemented example:

```
process (instruction, reset)
    variable rformat, lw, sw, beq, jmp, addi : std_logic;   -- 1 if the instruction is identified
    variable opcode : std_logic_vector(5 downto 0);
begin
if reset = '1' then
    RegDst <= '0';
    ALUSrc <= '0';
    MemToReg <= '0';
    RegWrite <= '0';
    MemRead <= '0';
    MemWrite <= '0';
    ALUOp <= "00";
    Jump <= '0';
    beq_control <= '0';
else
    opcode(5 downto 0) := instruction(31 downto 26);

-- identify each instruction type from instruction
if opcode = "000000" then
    rformat := '1';
else
    rformat := '0';
end if;
```

```
if opcode = "100011" then
    lw := '1';
else
    lw := '0';
end if;

--- The instructions you should implement are
--- lw, rformat, sw, beq, jmp, and addiu
--- this is a partial MIPS instruction set, but it is sufficient to program many applications
-- implement all of the control output signals here.
-- see the table in Figure 4.22, the textbook page 327 and implement all of the signals. an example
is
-- shown below
RegDst <= rformat;
ALUSrc <= (lw or sw or addi);
MemToReg <= lw;
…
…
beq_control <= beq;
end if;
end process;
```

Figure 10.3 shows the logic in the control unit as one large truth table that combines all the outputs and that uses the opcode bits as inputs. It completely specifies the control function, and we can implement it directly in gates in an automated fashion.

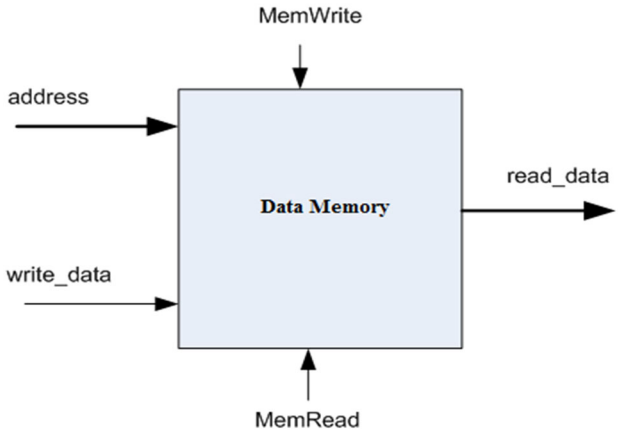
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

Figure 10.3 Control function for the simplified function

For pre-lab task 1, implement the table given in figure 10.3. An easier approach will be to implement this table like we implemented truth table in lab 2.

Task 2: Data Memory module

Data memory module implementation is nearly identical to the instruction memory. Please refer to your previous program. The block diagram is shown in Figure 10.4 and implemented using an array of 32bits



10.4 Data Memory Module

Lab Tasks

Testing will be done by connecting all of the modules you have constructed. Your understanding on the data path is important for debugging

Lab Task 1: Make a port map and test the Control and Memory Unit

Make module to module connections

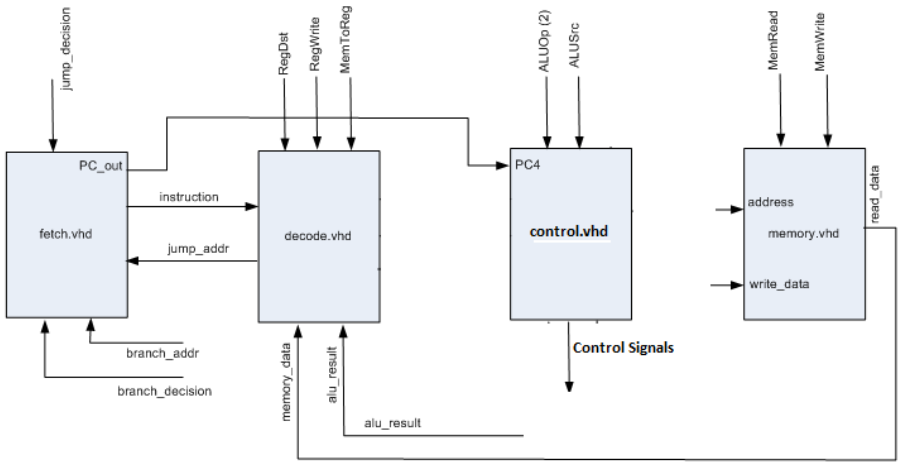


Figure 10.5 Control and Memory Unit

Create a top module file by port mapping all the modules.

Lab Task 2: Implement the Data Memory module

```
entity memory is
generic (module_delay: time := 10 ns);
Port (
    address : in STD_LOGIC_VECTOR (31 downto 0);
    write_data : in STD_LOGIC_VECTOR (31 downto 0);
    MemWrite, MemRead : in std_logic;
    read_data : out STD_LOGIC_VECTOR (31 downto 0));
end memory;

architecture Behavioral of memory is
    type mem_array is array(0 to 7) of std_logic_vector(31 downto 0);

begin
ReadWrite1: process (address, write_data)
        variable data_mem : mem_array := (
        X"00000000", --initialize the data memory to constants to test
        X"11111111",
        X"22222222",
        X"33333333",
        X"44444444",
        X"55555555",
        X"66666666",
        X"77777777");
        variable addr: integer;
        variable mem_content : std_logic_vector(31 downto 0);
        begin

            addr := conv_integer(address(2 downto 0)); --since there are only 8 words
            mem_content := write_data;
            if MemWrite = '1' then
                data_mem(addr) := mem_content;
            elsif MemRead = '1' then
                mem_content := data_mem(addr);
            read_data <= mem_content after module_delay;
            end if;
        end process ReadWrite1;
end Behavioral;
```

Lab Task 3: Demo

Show the control module outputs on FPGA LEDs according to the following table

Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Rubric for Lab Assessment

The student performance for the assigned task during the lab session was:			
Excellent	The student completed assigned tasks without any help from the instructor and showed the results appropriately.	4	
Good	The student completed assigned tasks with minimal help from the instructor and showed the results appropriately.	3	
Average	The student could not complete all assigned tasks and showed partial results.	2	
Worst	The student did not complete assigned tasks.	1	

Instructor Signature: _____ Date: _____