LAB MANUAL

Subject: DIGITAL DESIGN AND COMPUTER ORGANIZATION

Lesson No/ Session No	Topics	No. of hours
1.	Implement Half and Full Adder in VHDL	2 Hrs
2.	Implement 4:1 MUX in VHDL Implement 1:4DEMUX in VHDL	2 Hrs
3.	Design a JK flip-flop in VHDL Design a D flip-flop in VHDL	2 Hrs
4.	Design a SR flip-flop in VHDL Design a T flip-flop in VHDL	2 Hrs
5.	Design and develop the VHDL code for mod 8 up counter	2 Hs
6.	Assembling and Disassembling of a computer	2 Hrs
7.	Demonstrate Datapath and instruction execution stages using MarieSim Simulator	2 Hrs
8.	Implementation of ARM Programs using ARMSim	2Hrs
9.	Implementation of ARM Programs using ARMSim	2 Hrs
10.	Design an 8-bit ALU to perform various functions using Logisim simulator.	2Hrs
11.	Design a memory system using Logisim simulator.	2Hrs
12.	Demonstration of pipelining using CPU OS Simulator	2 Hrs
13.	Revision	2Hrs
14.	Test	2 Hrs

1. **Implement Half and Full Adder in VHDL**

Half Adder

```
library ieee;
use ieee.std_logic_1164.all;
entity ha is
        port(a,b:in bit;
        sum,carry:out bit);
end ha;
architecture desc of ha is
begin
        sum<= a xor b;
        carry <= a and b;
end desc;
```

Full Adder

```
library ieee;
use ieee.std_logic_1164.all;
entity fa is
        port(a,b,c:in bit;
        sum,carry:out bit);
end fa;
architecture desc of fa is
begin
        sum <= a xor b xor c;
        carry<= ( a and b) or ( a and c) or (b and c);
end desc;
```

2. **Implement 4:1 MUX in VHDL**
   Implement 1:4DEMUX in VHDL

4 to 1 MUX

```
library ieee;
use ieee.std_logic_1164.all;
entity mux_4to1 is
        port( A,B,C,D : in bit;
        S1,S0: in bit;
        Y: out bit);
end mux_4to1;
architecture bhv of mux_4to1 is
begin
        process (A,B,C,D,S0,S1) is
        begin
                if (S1 ='0' and S0 = '0') then
                        Y<= A;
                elsif (S1 ='0' and S0 = '1') then
                        Y<= B;
                elsif (S1 ='1' and S0 = '0') then
                        Y<= C;
                 else
                        Y<= D;
                end if;
        end process;
end bhv;
```

1 to 4 DEMUX

```
library ieee;
use ieee.std_logic_1164.all;
entity demux_1to4 is
	port(F : in bit;
	S0,S1: in bit;
A,B,C,D: out bit);
end demux_1to4;
architecture bhv of demux_1to4 is
begin
	process (F,S0,S1) is
	begin
		if (S1 ='0' and S0 = '0') then
			A <= F;
		elsif (S1 ='0' and S0 = '1') then
			B <= F;
		elsif (S1 ='1' and S1 = '0') then
			C <= F;
		else
			D <= F;
		end if;
	end process;
end bhv;
```

3. **Design a JK flip-flop in VHDL**

   Design a D flip-flop in VHDL

JK flip-flop

```
library ieee;
use ieee. std_logic_1164.all;
entity JK_FF is
        port( J,K,CLOCK: in std_logic;
                Q, Qb: out std_logic);
end JK_FF;
architecture behavioral of JK_FF is
begin
        process(CLOCK)
        variable TMP: std_logic;
        begin
                if(rising_edge(CLOCK)) then
                        if(J='0' and K='0')then
                                TMP:=TMP;
                        elsif(J='1' and K='1')then
                                TMP:= not TMP;
                        elsif(J='0' and K='1')then
                                TMP:='0';
                        else
                                TMP:='1';
                        end if;
                end if;
                Q <= TMP;
                Qb <= not TMP;
        end PROCESS;
end behavioral;
```

D flip-flop

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

entity D_FF is
        PORT( D,CLOCK: in std_logic;
                Q,Qb: out std_logic);
end D_FF;

architecture behavioral of D_FF is
begin
        process(CLOCK)
        variable TMP: std_logic;
        begin
                if(rising_edge(CLOCK)) then
                 TMP :=D;
                end if;
                Q <= TMP;
                Qb<=not TMP;
        end process;
end behavioral;
```

4. **Design a SR flip-flop in VHDL**
   Design a T flip-flop in VHDL

SR flip-flop

```
library ieee;
use ieee. std_logic_1164.all;
entity SR_FF is
        port( S,R, CLOCK: in std_logic;
            Q, Qb: out std_logic);
end SR_FF;
architecture behavioral of SR_FF is
begin
        process(CLOCK)
        variable TMP: std_logic:='0';
        begin
                if(rising_edge(CLOCK)) then
                        if(S='0' and R='0')then
                                TMP:=TMP;
                        elsif(S='1' and R='1')then
                                TMP:='X';
                        elsif(S='0' and R='1')then
                                TMP:='0';
                        else
                                TMP:='1';
                        end if;
                end if;
                Q <= TMP;
                Qb <= not TMP;
        end PROCESS;
end behavioral;
```

T flip-flop

```
library ieee;
use ieee. std_logic_1164.all;
entity T_FF is
        port( T,CLOCK: in std_logic;
        Q, Qb: out std_logic);
end T_FF;
architecture behavioral of T_FF is
begin
        process(CLOCK)
        variable TMP: std_logic:='0';
        begin
                if(rising_edge(CLOCK)) then
                        if(T='0')then
                                TMP:=TMP;
                        else
                                TMP:= not TMP;
                        end if;
                end if;
                Q <= TMP;
                Qb <= not TMP;
        end PROCESS;
end behavioral;
```

5. **Design and develop the VHDL code for mod 8 up counter**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity MOD8 is
    Port ( CLK,RST : in  STD_LOGIC;
             COUNT : inout  STD_LOGIC_VECTOR (3 downto 0));
end MOD8;

architecture Behavioral of MOD8 is

begin
       process (CLK,RST)
       begin

              if (RST = '1')then
                     COUNT <= "0000";
              elsif(COUNT = "0111" and rising_edge(CLK)) then
                     COUNT <= "0000";
              elsif(rising_edge(CLK))then
                     COUNT <= COUNT + 1;
              end if;

       end process;
end Behavioral;
```

6. Design and develop the VHDL code for Encoder and Decoder.

4:2 Encoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity encoder1 is
 port(
 a : in STD_LOGIC_VECTOR(3 downto 0);
 b : out STD_LOGIC_VECTOR(1 downto 0)
 );
end encoder1;

architecture bhv of encoder1 is
begin

        process(a)
        begin
                if (a="0001") then
                        b <= "00";
                elsif (a="0010") then
                        b <= "01";
                elsif (a="0100") then
                        b <= "10";
                elsif (a="1000") then
                        b <= "11";
                else
                        b <= "ZZ";
                end if;
        end process;

end bhv;
```

2:4 Decoder

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity decoder1 is
        port(
                a : in STD_LOGIC_VECTOR(1 downto 0);
                b : out STD_LOGIC_VECTOR(3 downto 0));
end decoder1;

architecture bhv of decoder1 is
begin

        process(a)
        begin
                if (a="00") then
                        b <= "0001";
                elsif (a="01") then
                        b <= "0010";
                elsif (a="10") then
                        b <= "0100";
                else
                        b <= "1000";
                end if;
        end process;

end bhv;
```

7. **Demonstrate Datapath and instruction execution stages using MarieSim Simulator**

1. Simple Program

```
      ORG 100

      LOAD x
      ADD Y
      Store Z
      HALT

      X, Dec 35
      Y, Dec -23
      Z, Hex 0
```

2. Program to get two values from user
/ add them together, and print result
/ read first value into ACC
/ then copy into variable X

```
      input
      store X
      input
      store Y

      add X / ACC = ACC + X
      output / print result
      halt / end program

      / local variables, initialized to 0 (using base 10)
      X, dec 0
      Y, dec 0
```

Program 3

/ program to get x,y from user and print largest of two numbers

```
      org 100
      load X
      subt Y / acc = X - Y
      skipcond 800 / if positive then X bigger, skip
further tests
      jump Test
      load X
      output
      halt

Test, load Y
      output
      halt
```

```
X, dec 20
y, dec 10
```

Example 4: simple loop
A program to get the user to enter an integer less than 10 (repeats until a valid response is given).

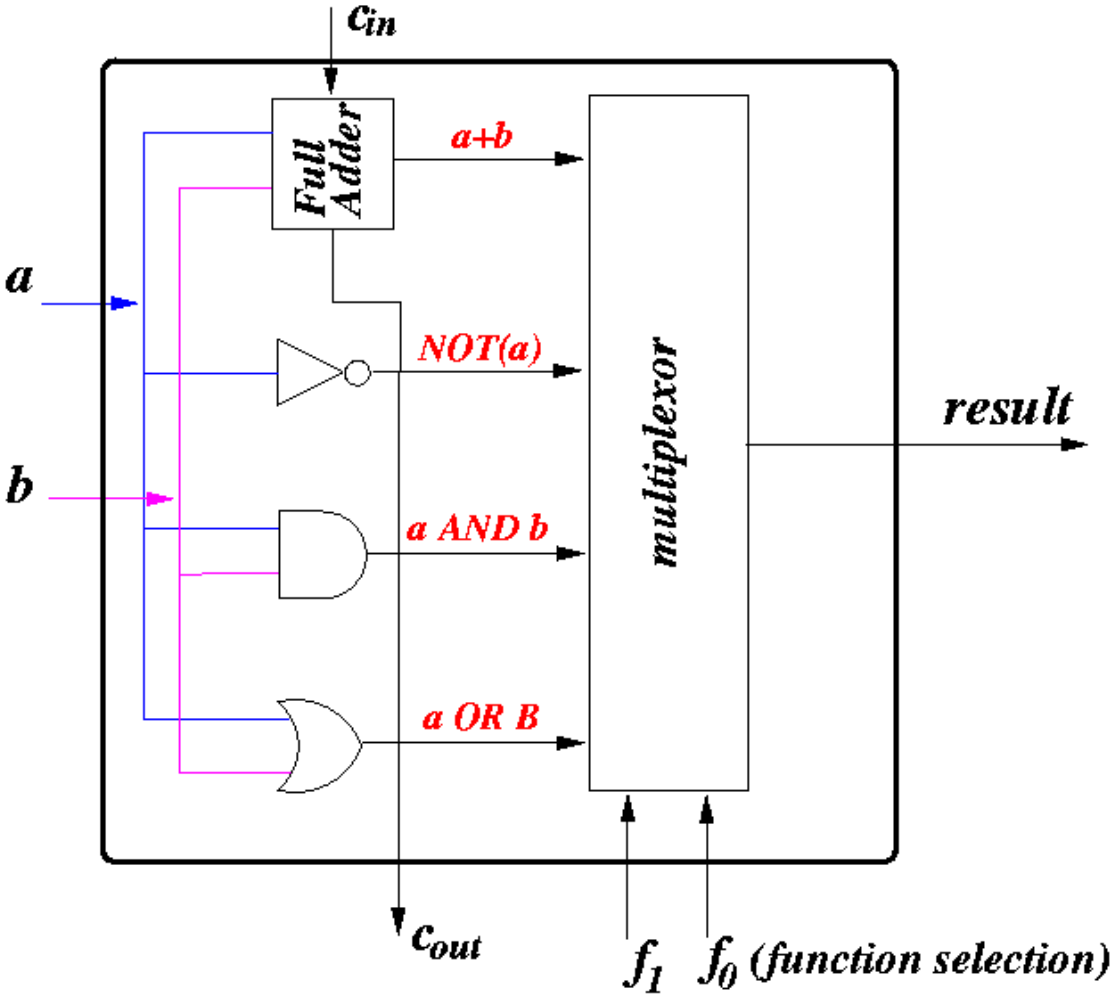
```
      / program to get int < 10 from user
      / keep reading til valid value obtained

      org 100
NewVal, input / get value for x
      store X
      subt TooBig / compute x - toobig
      skipcond 000 / if negative then x ok (less than
toobig)
      jump NewVal / otherwise go back and try again

Show, load X / got valid value, print x, then done
      output
      halt

X, dec 0 / default value for x
TooBig, dec 10 / X must be less than this
```

8. Design a 1-bit ALU to perform various functions using Logisim simulator.



Function selection	Operation
00	Addition
01	Not
10	And
11	Or

9. Demonstration of pipelining using CPU OS Simulator

Exercise 1 – Difference between the sequential and the pipelined execution of CPU instructions

Enter the following source code, compile it and load in simulator's memory:

```
program Ex1
    for n = 1 to 20

        p = p + 1
    next

end
```

Open the CPU pipeline window by clicking on the **SHOW PIPELINE…** button in the CPU simulator's window. You should now see the **Instruction Pipeline** window. This window simulates the behaviour of a CPU pipeline. Here we can observe the different stages of the pipeline as program instructions are processed. This pipeline has five stages. The stages are colour-coded as shown in the key for the "Pipeline Stages".

List the names of the stages here:

The instructions that are being pipelined are listed on the left side (in white text boxes). The newest instruction in the pipeline is at the bottom and the oldest at the top. You'll see this when you run the instructions. The horizontal yellowish boxes display the stages of an instruction as it progresses through the pipeline. At the bottom left corner pipeline statistics are displayed as the instructions are executed.

Check the box titled **Stay on top** and make sure **No instruction pipeline** check box is selected. In the CPU simulator window bring the speed slider down to around a reading of 30. Run the program and observe the pipeline. Wait for the program to complete. Now make a note of the following values

CPI (Clocks Per Instruction)	
SF (Speed-up Factor)	

Next, uncheck the **No instruction pipeline** checkbox, reset and run the above program again and wait for it to complete.

Note down your observation on how the pipeline visually behaved differently

Now once again make a note of the following values

CPI (Clocks Per Instruction)	
SF (Speed-up Factor)	

Briefly explain why you think there is a difference in the two sets of values:

10. Demonstration of Cache Organization using CPU OS Simulator

Exercise 1 – Investigating Directly Mapped cache organization

Create a new program, call it **CacheTest1** and enter the following code:

```
MOV #0, R01
STB R01, @R01
CMP #63, R01
JEQ 31
ADD #1, R01
JMP 6HLT
```

The above code writes numbers 0 to 63 in memory locations 0 to 63. Run it and observe the contents of the data in memory. To see the memory click onthe **SHOW PROGRAM DATA MEMORY…** button.

Click on the **SHOW CACHE…** button to display the data cache window. Makesure the **Stay on top** check box is checked. Now, flush the cache by clicking on the **FLUSH** button and configure the cache with the following settings:

Block Size = 4
Cache Type = Direct Mapped
Cache Size = 16

Now insert the following code <u>below</u> the instruction **JMP 6** in the above code:

```
LDB 0, R00
LDB 1, R00
LDB 2, R00
LDB 3, R00
```

To execute the above LDB instructions individually, double-click on each of the above LDB instructions. Write down what you observe in the table below:

Addr	Data	Hits	Block

Also make a note of the following data displayed in the **Cache Stats** frame:

Hits		**%Hits**	
Misses		**%Misses**	

Note: %Hits = 100 - %Misses

Insert the following LDB instruction after the last LDB instruction above and

execute it by double-clicking on it:

LDB 4, R00

Write down the additional contents (i.e. in addition to the above data) of the cache below:

Addr	**Data**	**Block**

Briefly explain your observations below:
