Lab 09 – Worksheet

Name: Breeha Qasim and Ali Ahsan	ID:	Section:

Task 1.

Code: Design module & testbench

Provide appropriately commented code for designed module, the code should contain meaningful variable naming.

**Add snippet or Copy paste the code written in the Editor window. Make sure the irrelevant area of the snip is cropped.*

```
module Program_Counter(
   input clk,
   input reset,
   input [63:0] PC_In,
   output reg [63:0] PC_Out
   );
   initial
   begin
   PC_Out=0;
   end

   always @(posedge clk)
   begin
   //Initialize PC_Out to 0 if reset signal is high, else reflect the value of PC_In to PC_Out
   if (reset)
      PC_Out=0;
   else
      PC_Out=PC_In;
   end
endmodule
```


Q. Why are we initializing Program Counter with '0'?

PC is initialized with 0 because it needs a starting address from where it will start incrementing by four bytes to fetch for next instruction.

Task 2.

Code: Design module & testbench

Provide appropriately commented code for designed module, the, code should contain meaningful variable naming.

**Add snippet or Copy paste the code written in the Editor window. Make sure the irrelevant area of the snip is cropped.*

```
module Adder(
    input [63:0] a,
    input [63:0] b,
    output [63:0] out
    );
    //adds the two input in ADDER
    assign out=a+b;
endmodule
```


Comments

Task 3

Code: Design module & testbench

Provide appropriately commented code for designed module & its testbecnch, code should contain meaningful variable naming.

**Add snippet or Copy paste the code written in the Editor window. Make sure the irrelevant area of the snip is cropped.*

Design Module

```
module Instruction_Fetch(
    input clk,
    input reset,
    output [31:0] Instruction
    );
    wire [63:0] PC_In;
    wire [63:0] PC_Out;

    Program_Counter pC(clk,reset,PC_In,PC_Out);
    Adder add(PC_oUT, 4, PC_In);
    Instruction_Memory im(PC_Out,Instruction);
endmodule
```

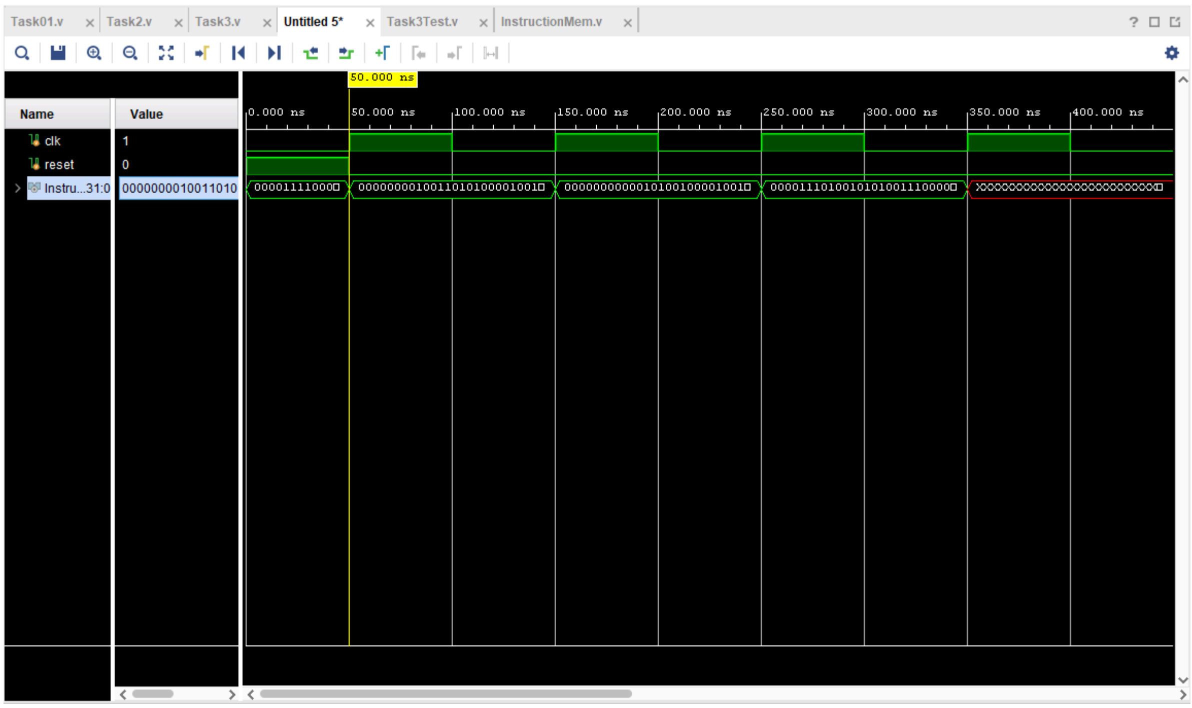
TestBench

```
module InstructionFetch_TestBench(

    );
    reg clk;
    reg reset;
    wire [31:0] Instruction;

    Instruction_Fetch IF(clk, reset,Instruction);

    initial
    begin
    clk = 0;
    reset = 1;

    #50
    reset = 0;

    #50
    reset = 0;

    #50
    reset = 0;
```


```
    end
    always #50 clk = ~clk;
endmodule
```

Results (Waveforms)

**Add snip of relevant signals' waveforms. Make sure the irrelevant area of the snip is cropped.*



```
Name          Value
clk           1
reset         0
Instru...31:0 0000000010011010

000011110000□ | 00000000100110101000010010□ | 00000000000101001000010010□ | 00001110100101010011100000□ | XXXXXXXXXXXXXXXXXXXXXXXXX□
```


Comments

**Observation/Comments on the obtained results/working of code.*

In the Instruction Fetch module, two inputs are introduced: clock and reset. Starting with a PC_In value of 0, the address contained in PC_In increases by one with each positive clock edge. This increment happens through the adder before the value is transferred to PC_Out, which in turn is fed back as the next PC_In for the Program Counter. The output from this module is a 32-bit instruction that arises from the PC_Out address, which is then forwarded to the Instruction Memory module. The final outcome of the simulation remains unspecified due to the PC address surpassing the byte capacity of the Instruction Memory.

Assessment Rubrics

Marks Distribution:

For description of different levels of the mapped rubrics, please refer the provided Lab Evaluation Assessment Rubrics.

Task No.	**LR2** **Code**	**LR 5** **Results**	**AR 7** **Report Submission**
Task 1 **Program Counter**	/20	-	/20
Task 2 **Adder**	/15	-	
Task 3 **Instruction Fetch**	/25	/20	
Total Points	/100 Points		
CLO Mapped	CLO 1		

Comments
