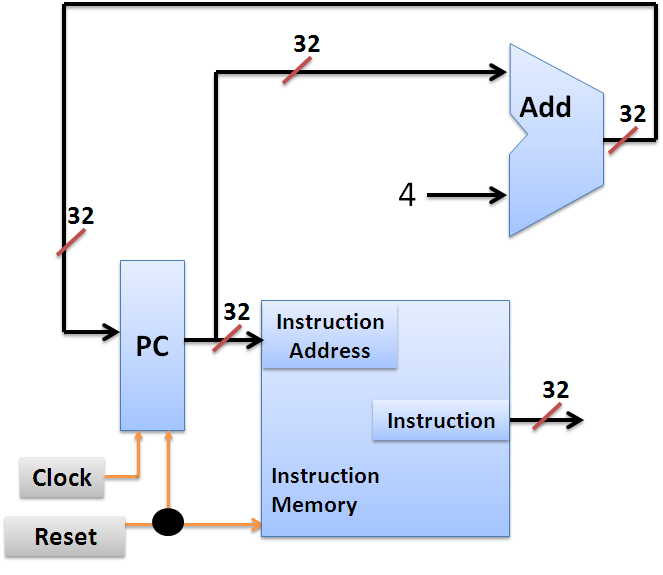
**Name-Yashi Malik**

**ID-2021A3PS3056G**

**Lab 1: Design of Instruction Fetch and ALU**

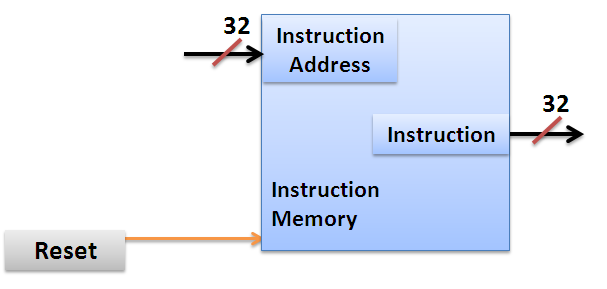
**Exercise 1.1: Implement Instruction Fetch Unit in Verilog.**

This exercise aims to implement the Behavioral model for the instruction fetch unit. Instruction fetch is the first stage of any processor. The instruction fetch unit for the MIPS processor consists of three main units: (1) A 32-bit program counter (PC) register also called the Instruction register which holds the address of the instruction that is to be fetched. (2) A byte addressable **BigEndian** Instruction Memory which accepts a 32-bit address and gives as output a 32-bit instruction code. (3) An adder to increment the contents of the PC to point to the next instruction. The instruction fetch unit contains two inputs a clock and a reset. When reset becomes zero PC should be initialized to 0 and the Instruction memory should be initialized with specific values. When the reset is not zero the instruction fetch unit should output a 32-bit instruction code corresponding to the address in the PC at the positive edge of the clock. The PC should be incremented to point to the next instruction after each clock cycle. The figure below shows the block-level diagram of the instruction fetch unit.



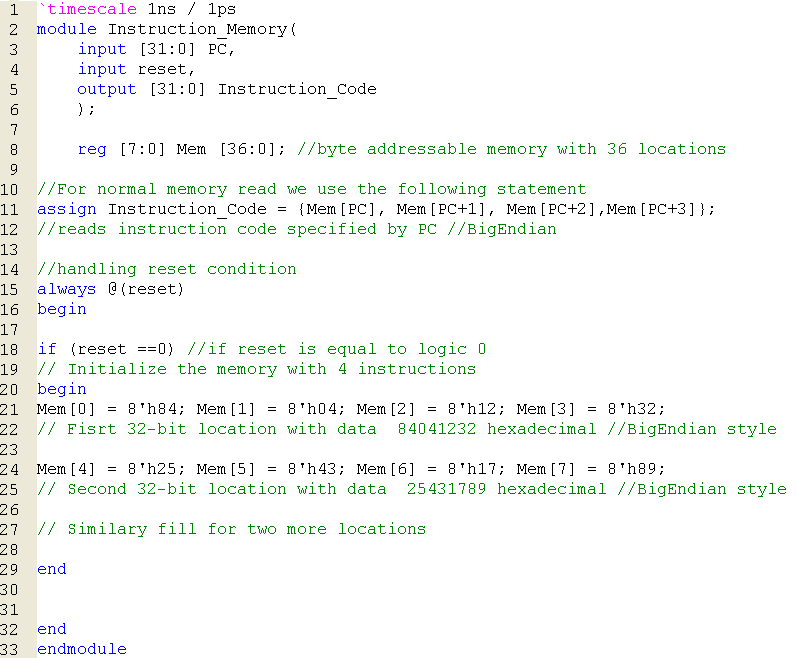
The instruction fetch is implemented in stages. The first stage is to implement the Instruction memory.

**Exercise 1.1.1: Implement Instruction Memory in Verilog.**



The instruction memory has two inputs a 32-bit Input coming from PC and a 1-bit reset. It has one 32-bit output indicating the output instruction code. According to the specifications when reset is logic 0 the Instruction memory should be initialized with specific data. This initialization is necessary to write the instruction codes into the memory. When reset is logic 1 the instruction memory should output the 32-bit instruction code corresponding to the 32-bit input address. The partial code for the Instruction memory is shown below. Please read the comments for a better understanding of the design.

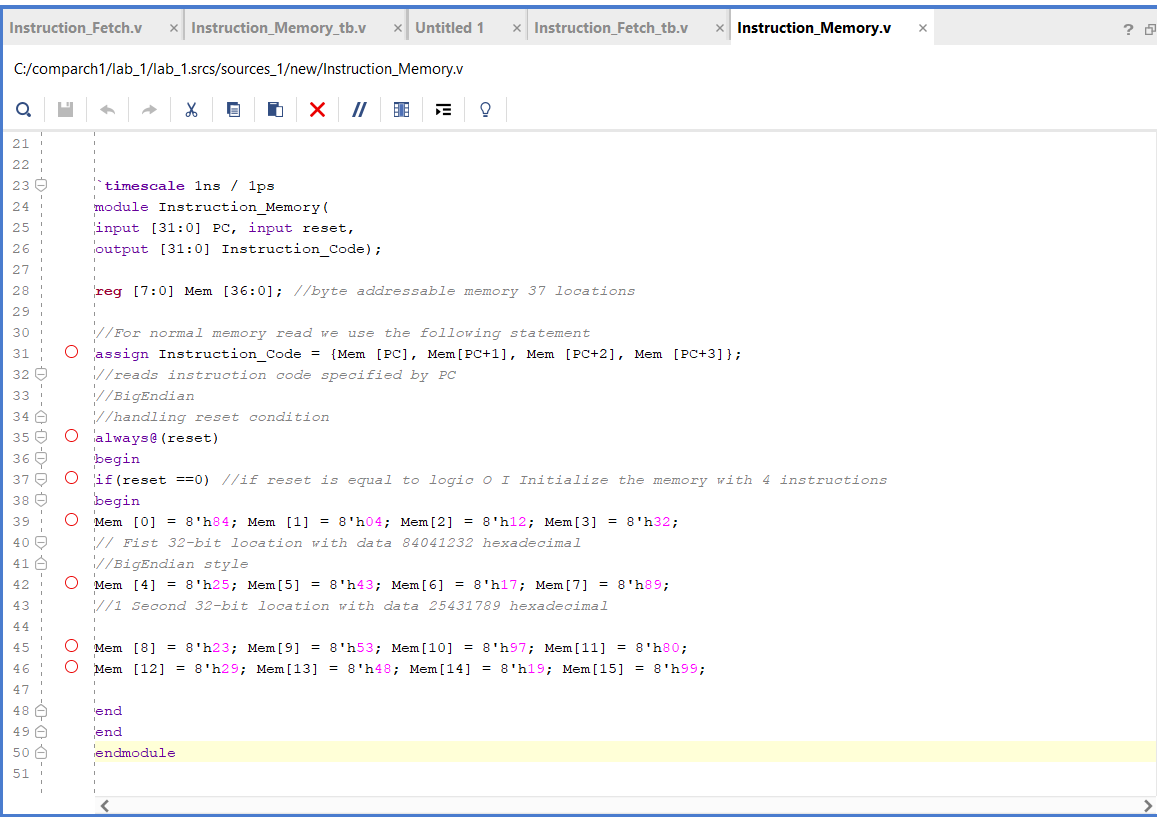
**Partial code: Instruction\_Memory.v**



37

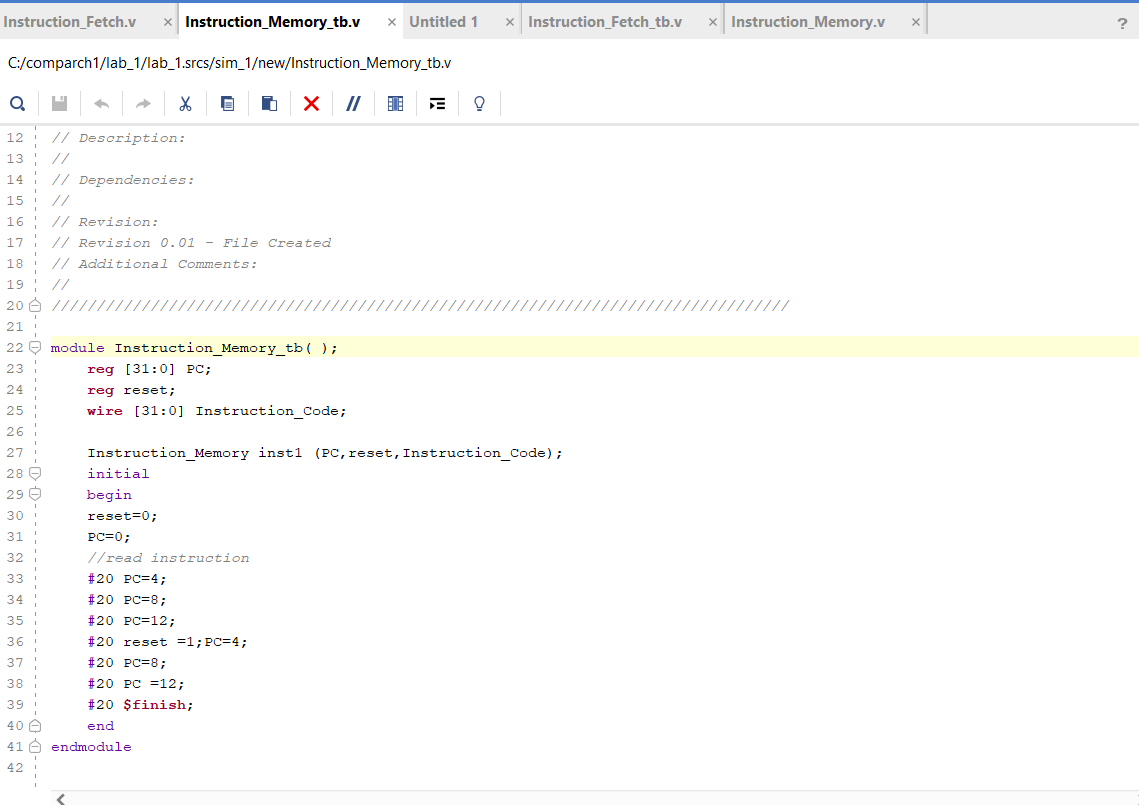
**reg [7:0] Mem [36:0]; defines byte addressable memory with 37 locations.**

**Copy the image of the completed Instruction memory module.?**

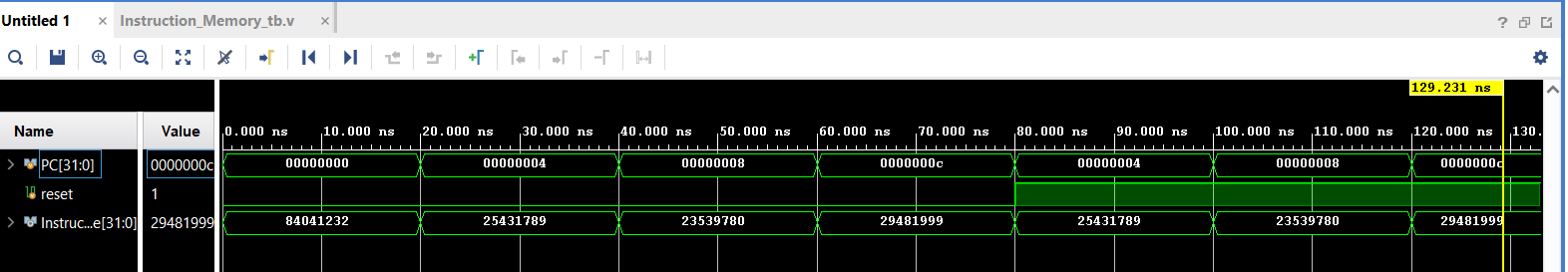
Answer: 

**Exercise 1.1.2 Write the TestBench to test the functionality of the Instruction Memory Module. (As part of your testbench enable reset initially and then give different values of PC)**

**Copy the image of the Testbench code?**

Answer: ****

**Copy the image of the waveform window that is generated for your Testbench? (Change display radix to Hexadecimal)**

Answer: 

**What changes will you make to line 11 of the Instruction\_Memory module if the memory is of Little-Endian type?**

Answer: Instruction\_Code = {Mem[PC+3],Mem[PC+2],Mem[PC+1],Mem[PC]};

**What changes will you make to line 21 of the Instruction\_Memory module if the memory is of Little-Endian type?**

Answer: Mem[0] = 8'h32; Mem[1] = 8'h12; Mem[2] = 8'h04; Mem[3] = 8'h84

**The data read out from Instruction memory is 32-bits. Then what is the reason for defining Mem as [7:0] Mem [No. of locations] instead of [31:0] Mem [No. of locations]**

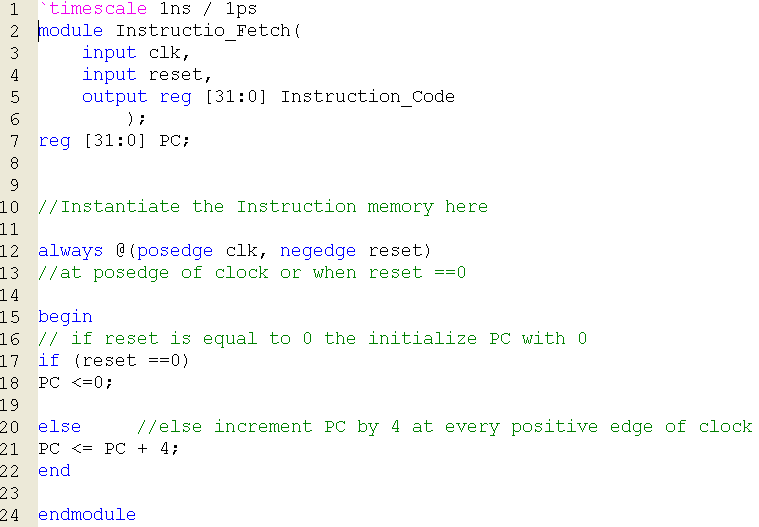
Answer:It is a byte addressable memory. We need 4 bytes for 32 bits.

**Find out and list other ways of initializing the memory.**

Answer: Initialization can be done from a designated disc file. For simulations, Verilog's readmemb() [for binary files] and readmemh() [for hexadecimal files] functions can be used.   
Memory initialization can also be performed through parameterized initialization, wherein parameters are employed to establish initial values for memory or various components within a Verilog module. This approach improves the module's reusability and configurability. By setting a default value for memory initialization, the parameter facilitates the initialization of the entire memory array within an always block when a reset is activated.

**Exercise 1.1.3 Implement and test (using test bench) the Instruction fetch unit by instantiating the Instruction memory block. (As part of Instruction fetch test bench enable reset initially and then generate continuous clock).**

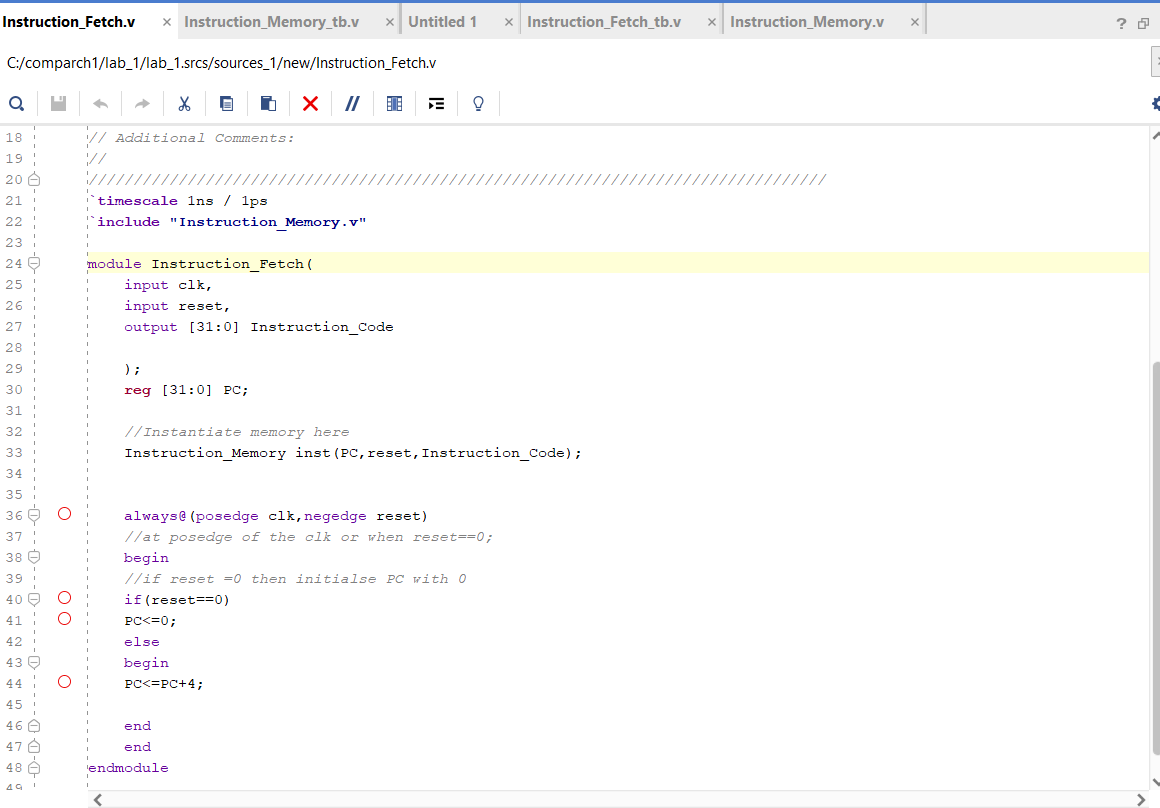
The instruction fetch unit has clock and reset pins as inputs and Instruction code as output. Internally it has a PC register which holds the address of current instruction. It also has an adder to compute PC + 4. The partial code for the instruction fetch unit (without instantiation of instruction memory) is shown below.



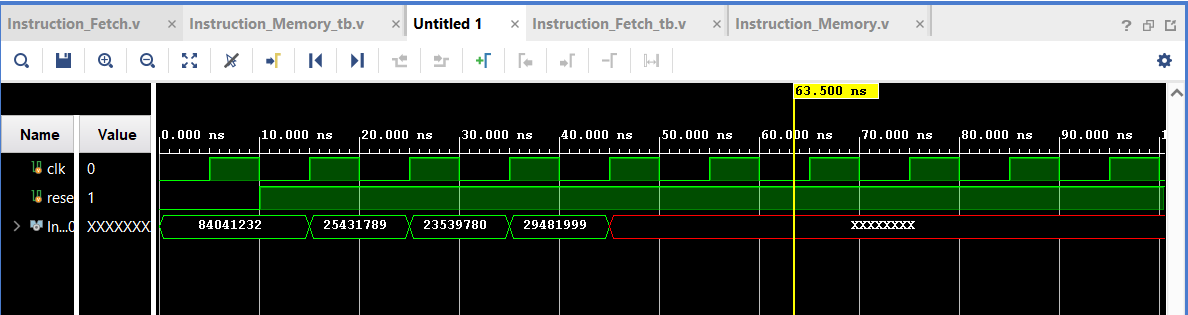
**There is an error in the code above. What is the error and what should be done to solve this error?**

Answer: Output instruction code cannot be declared as reg type in this module as it was a wire in another module which is instantiated in this module.

**Copy the image of the completed Instruction fetch module?**

Answer: 

**Copy the image of the waveform window that is generated for your Testbench? (Change display radix to Hexadecimal).**

Answer: 

**Exercise 1.2 Design of simple ALU.**

ALU (Arithmetic and Logical unit) is an important component of any processor. ALU performs different arithmetic and logical operations depending on the control lines. In this section, you will be implementing a simple ALU for a given set of specifications.

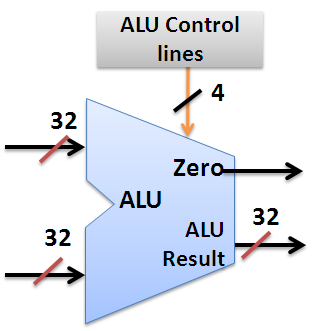
**ALU Specifications:**

This ALU has two 32-bit operands and 4 control lines as inputs. It has two outputs a 32-bit ALU result and a Zero indicator which becomes logic 1 if and only if the 32-bit ALU result is Zero. The ALU has to perform different functions according to the value of 4 control lines. The block diagram of ALU along with the mapping of control lines to functions performed is shown below.

|  |  |
| --- | --- |
| **ALU Control lines** | **Function** |
| 0000 | Bitwise-AND |
| 0001 | Bitwise-OR |
| 0010 | Add (A+B) |
| 0100 | Subtract (A-B) |
| 1000 | Set on less than |

\*Set on less than (ALU Result = 1 if A<B

else ALU Result =0)

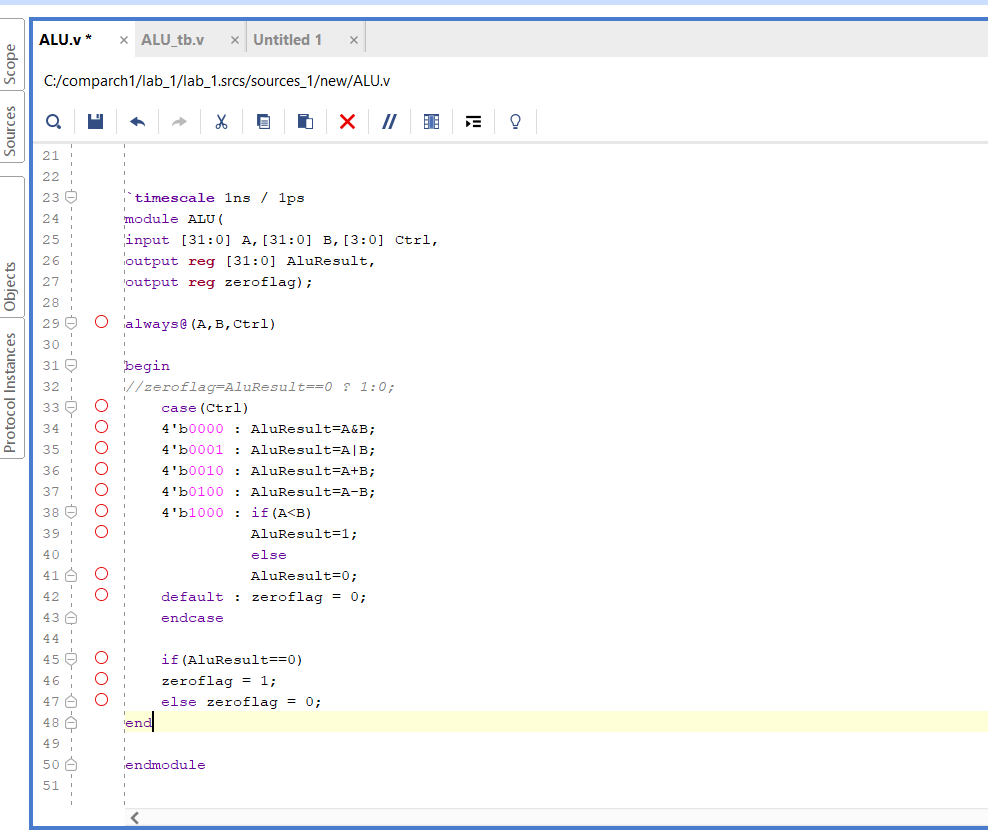


**B**

**A**

**Implement the ALU for the above specifications in Verilog using behavioral modeling (Hint: Use Case statement).**

**Paste the image of your Verilog code.**

Answer:  ****

**Write the test bench to test the ALU. Your test bench should have 7 different test patterns as mentioned below. (Assume test pattern changes after every 20 time units)**

Test case 1: A = 23, B = 42, ALUContol = 4`b0000.

Test case 2: A = 23, B = 42, ALUContol = 4`b0001.

Test case 3: A = 23, B = 42, ALUContol = 4`b0010.

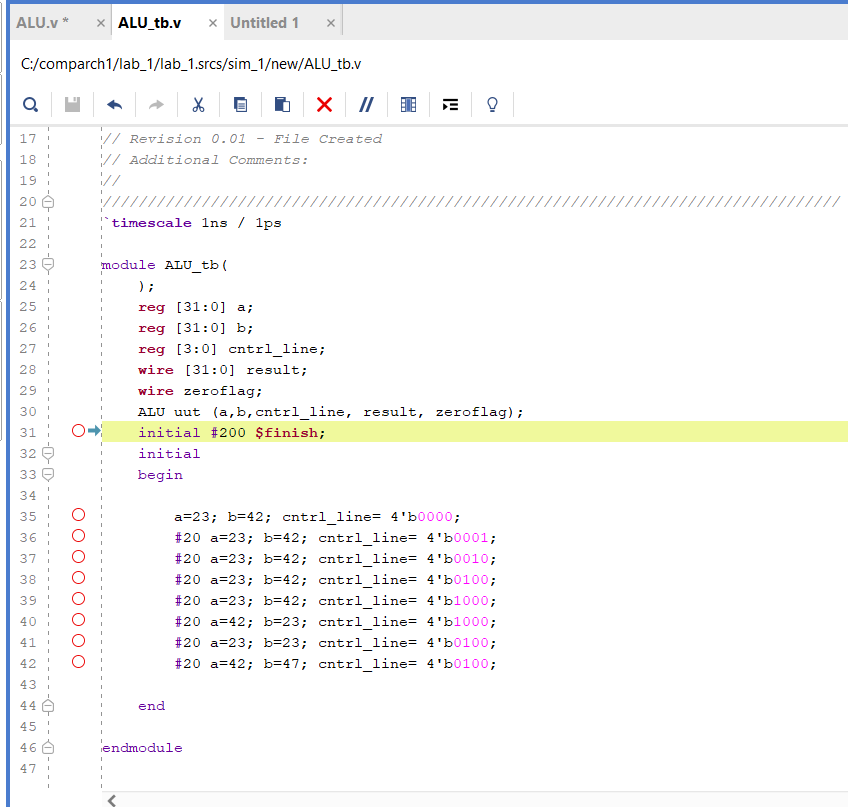
Test case 4: A = 23, B = 42, ALUContol = 4`b0100.

Test case 5: A = 23, B = 42, ALUContol = 4`b1000.

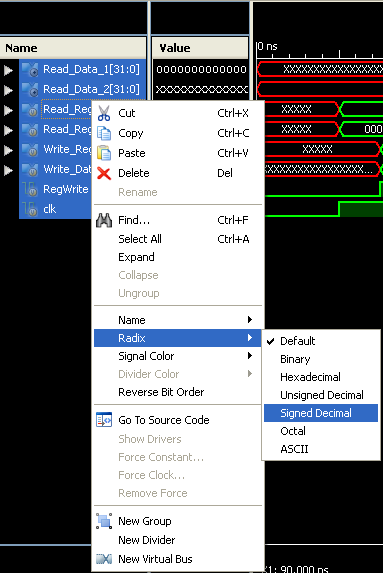
Test case 6: A = 42, B = 23, ALUContol = 4`b1000.

Test case 7: A = 42, B = 23, ALUContol = 4`b0100.

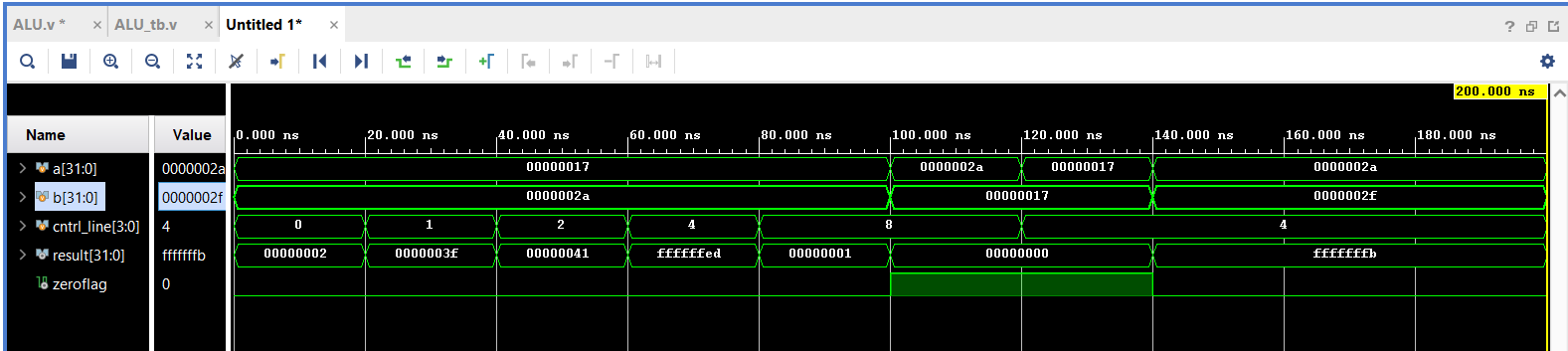
**Paste the image of your test bench with the above test cases.**

Answer:  ****

For easy viewing of the testbench waveforms where signals are multi-bit, you can change the view in the simulation window to decimal/hexadecimal mode. Right-click on all the input output signals select Radix and then click on signed decimal/hexadecimal.



**Copy the image of the waveform window that is generated for your Testbench? (Change display radix to Hexadecimal).**

Answer: 

**How many different functions can be implemented by ALU with 4-control lines?**

Answer: 16

**Exercise 1.3 Synthesis of Instruction Fetch Unit and ALU**

**Do you have a synthesisable design? If not write a synthesisable Verilog description of your design.**

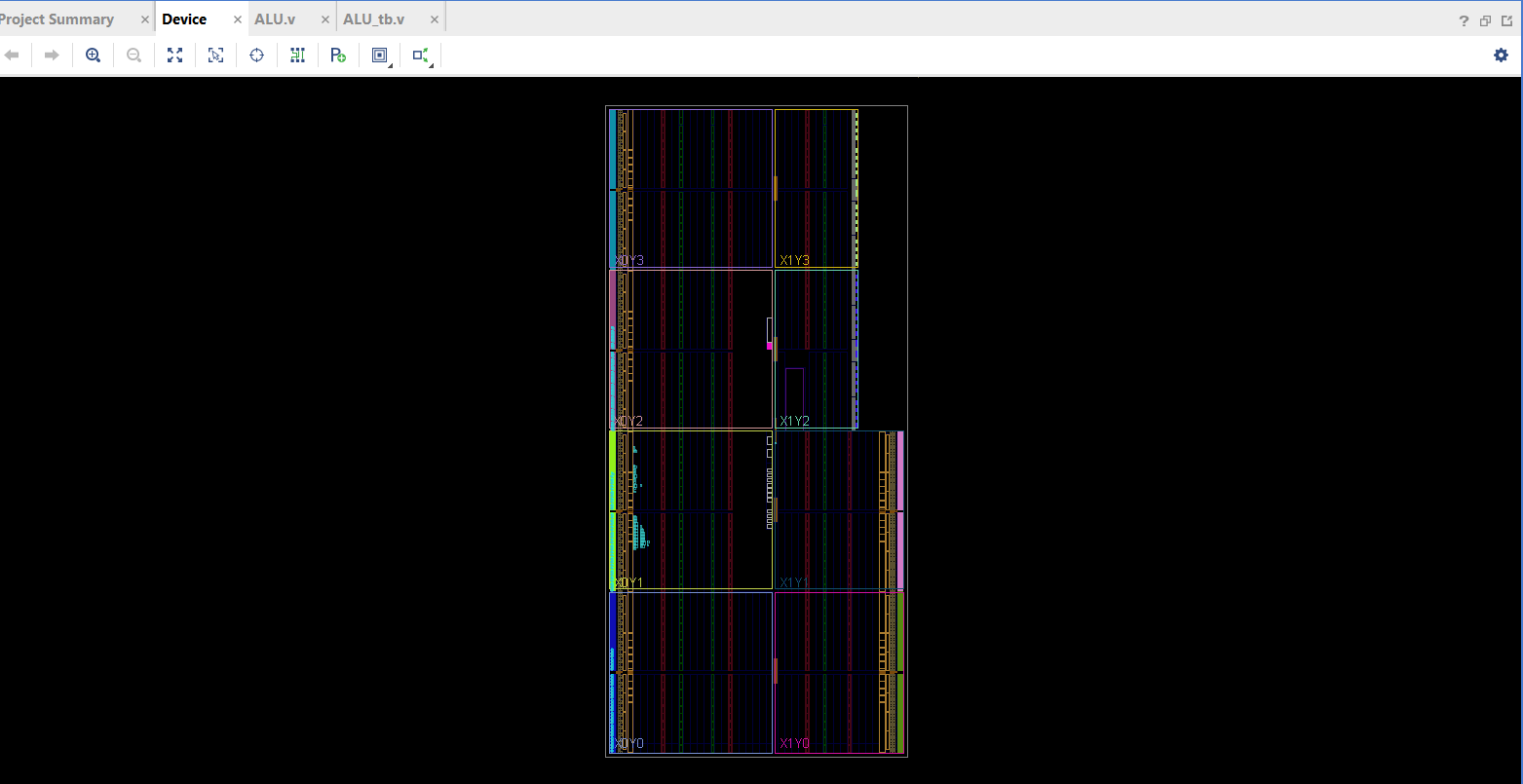
Use the Xilinx Vivado, synthesise, and obtain the RTL. Verify the RTL with your architecture.

Note down your observation below:

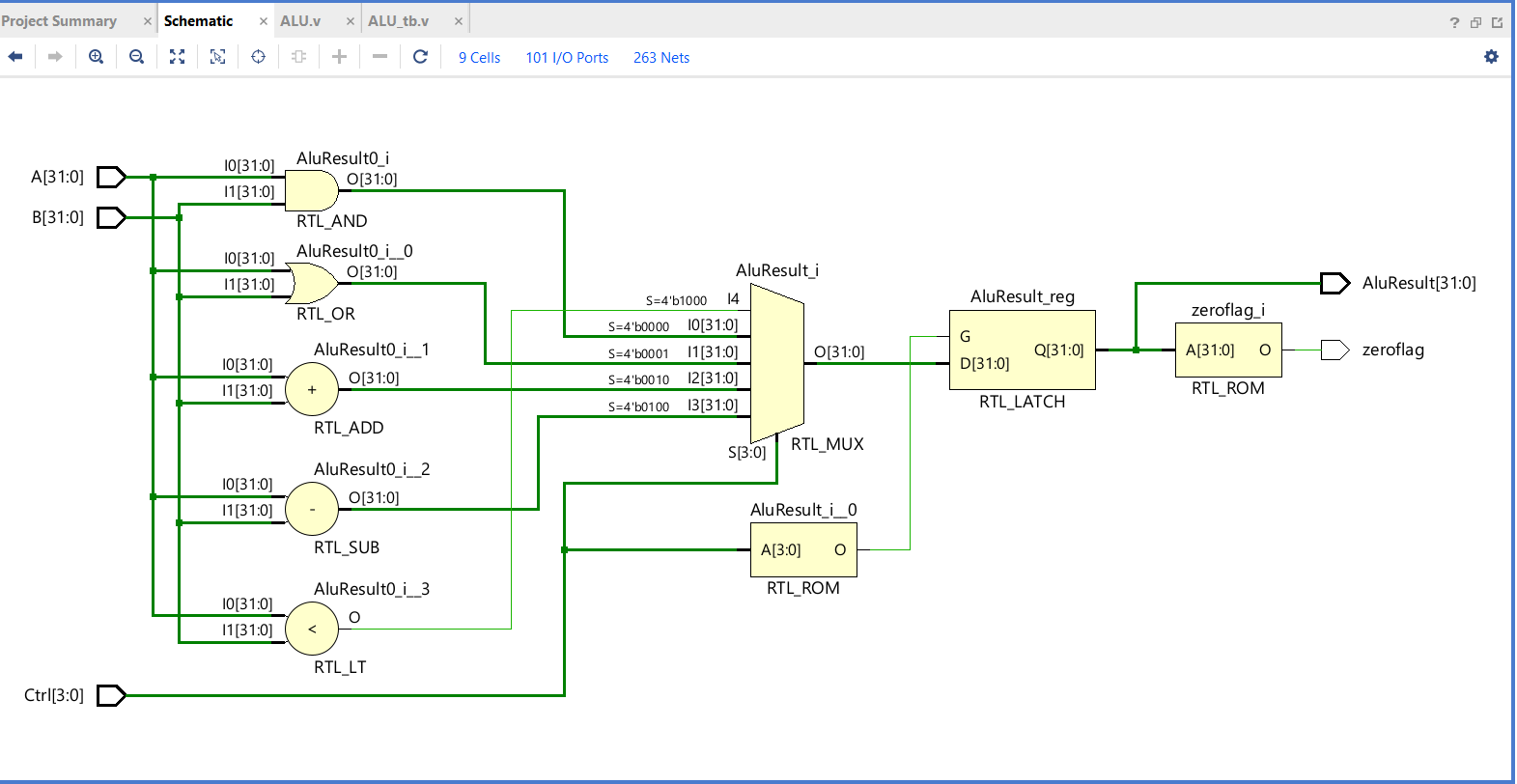
Yes, we have a synthesizable design because we have mostly used if….else statements that can be synthesized using a MUX.

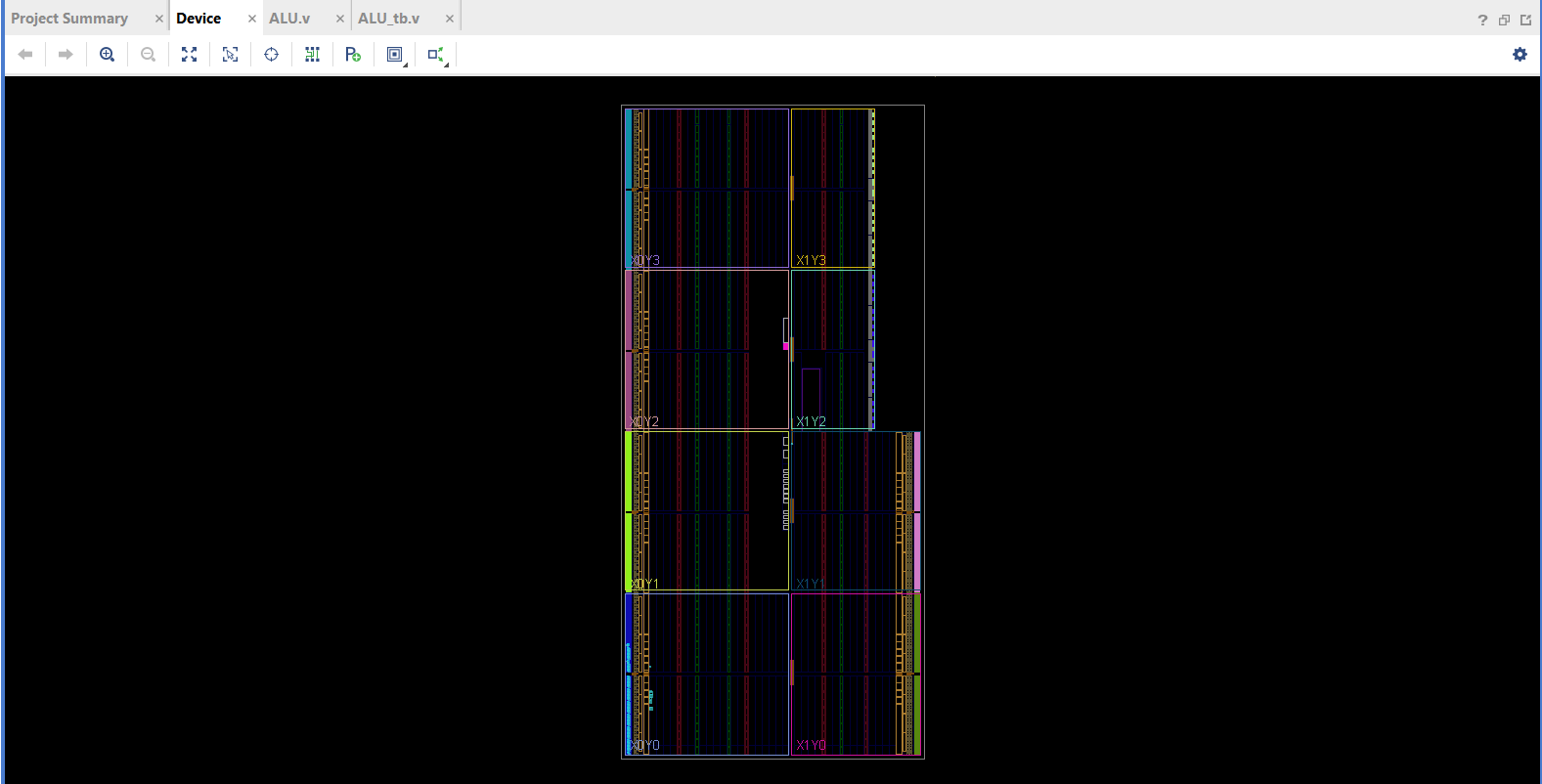
This was verified by the successful synthesis and RTL implementation of Instruction fetch unit and the ALU.

Synthesis of ALU

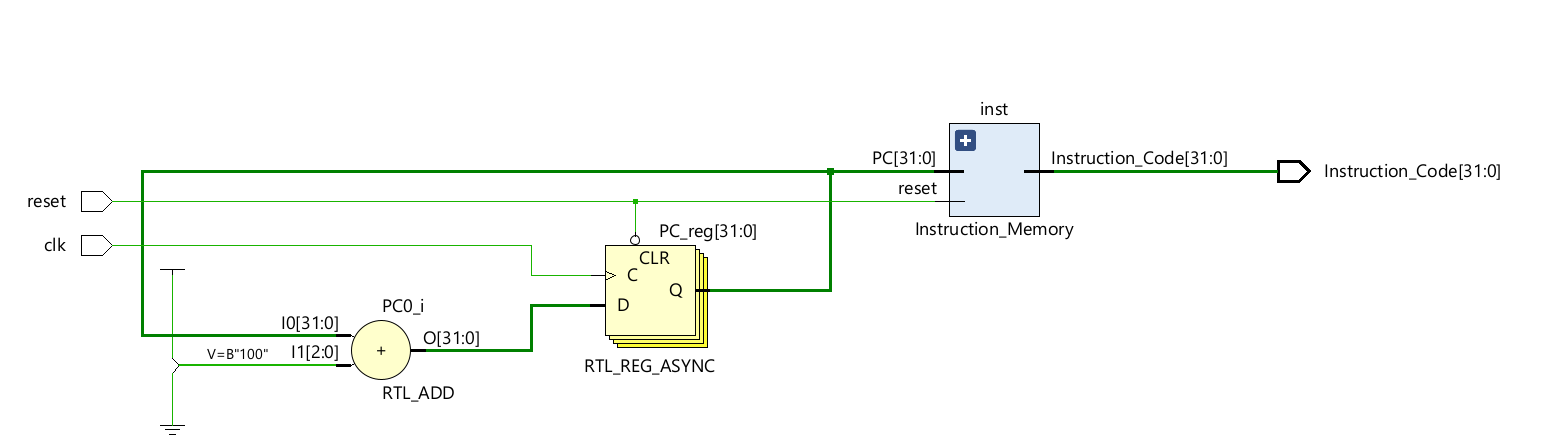


RTL implementation of ALU



Synthesis of Instruction fetch unit

RTL implementation of instruction fetch unit



The above designs have all the components that we intended to execute with the Verilog code, thus the Instruction fetch unit and ALU are synthesisable