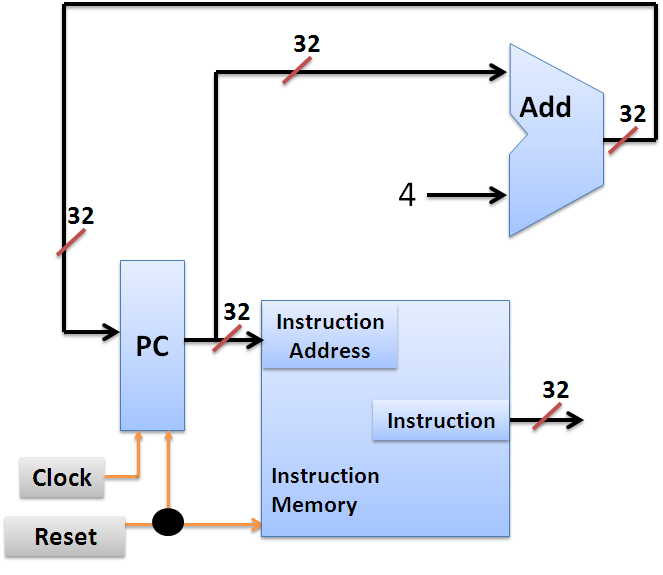
**Experiment No 5: Implementation of Instruction fetch Unit**

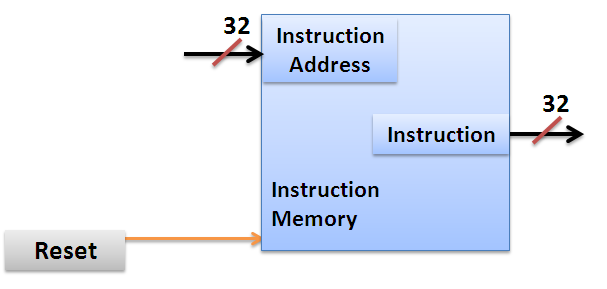
|  |  |  |
| --- | --- | --- |
| **Sl No** | **Name** | **ID No** |
| **1** | **VISHWAS VASUKI GAUTAM** | **2019A3PS0443H** |

The aim of this experiment is to implement the Behavioral model for instruction fetch unit. Instruction fetch is the first stage of any processor. The instruction fetch unit for RISC V processor consists of three main units: (1) A 32-bit program counter (PC) register also called Instruction register which holds the address of instruction that is to be fetched. (2) A byte addressable **LittleEndian** 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 PC to point to 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 PC at positive edge of clock. The PC should be incremented to point to next instruction after each clock cycle. The figure below shows the block level diagram of the instruction fetch unit.



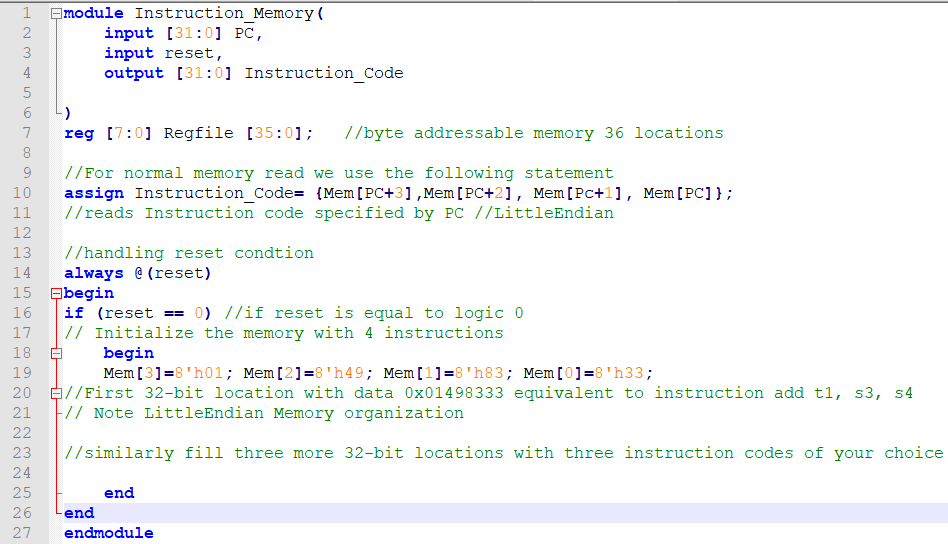
The instruction fetch is implemented in stages. The first stage is to implement the Instruction memory.

**Exercise 5.1: 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 necessary to write the instruction codes in to the memory. When reset is logic 1 the instruction code 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 better understanding of the design.

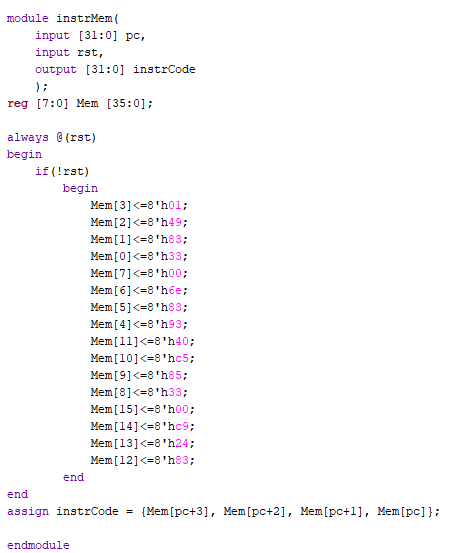
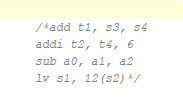
**Partial code: Instruction\_Memory.v**



Mem

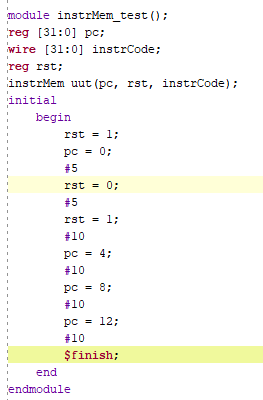
**reg [7:0] Mem [35:0];defines byte addressable memory with 36 locations.**

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

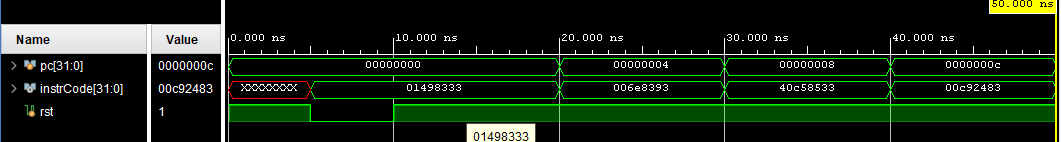
Answer: **Instructions:**

**Exercise 5.2 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)**

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

Answer: 

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

Answer: 

1. **What changes will you make to the line 10 of the Instruction\_Memory module if the memory is of BigEndian type?**

Answer: **Just reverse the order, and put {mem[pc], mem[pc+1], mem[pc+2] mem[pc+3]} as the instruction code.**

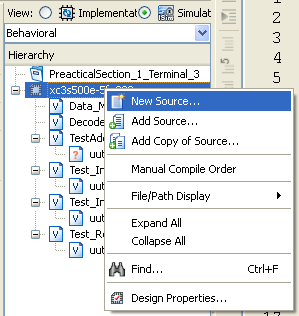
1. **What changes will you make to the line 19 of the Instruction\_Memory module if the memory is of BigEndian type?**

Answer: Similar to the previosu question, reverse the order and let mem[0] take the msb and mem[3] take the lsb.

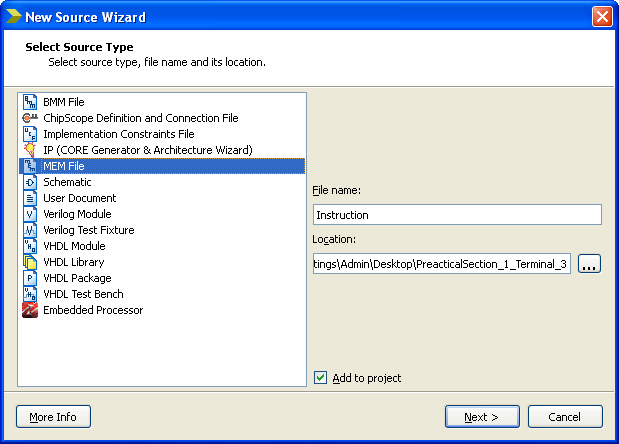
1. **The data read out from Instruction memory is 32-bits. Then what is the reason for defining Mem as [7:0] Mem [No. of locations] instead of [31:0] Mem [No. of locations]**

Answer: **The processor we are implementing is byte addressable**.

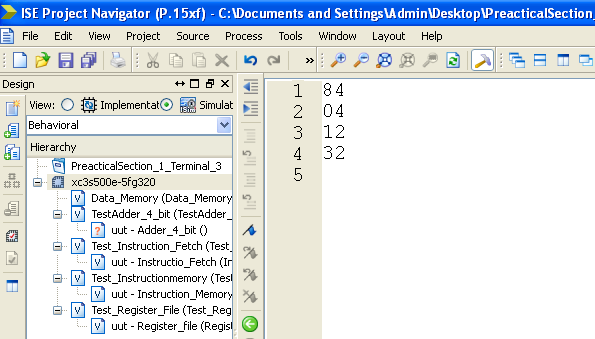
**Large memories Initialization are done using memory files. An example of memory initialization using memory file is shown below. Right click on the Project Select New source.**



Select MEM file give the name to memory. Clock Next and the finish.



Give the hexadecimal values in the text window that appears. Filling memory for 4 byte (one word) location is shown below. Save the file.



After the memory is filled with required number of locations add directive to load this memory.

**$readmemh("Instruction.mem",Mem);**

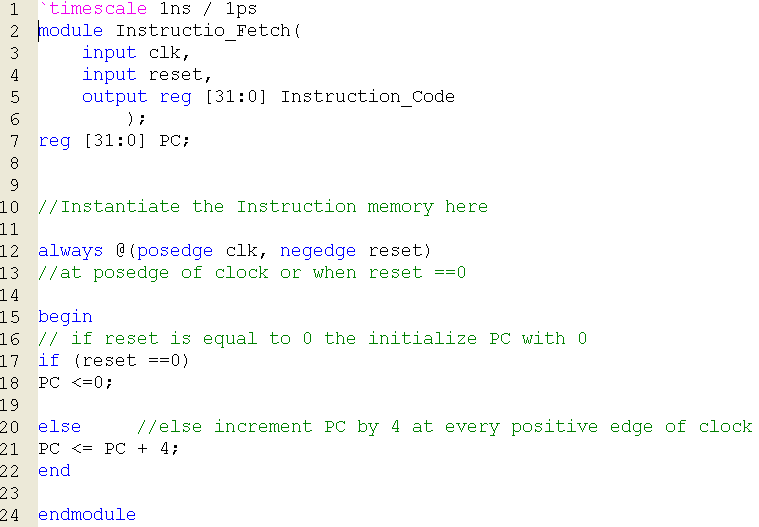
This statement can be used as substitute for the lines 19-24 of the Instruction\_Memory Module figure. This statement reads each line in the memory file (Instruction.mem) as hexadecimal value and stores them in **Mem.**

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

Answer: You can use $readmemb if binary is needed to be read. In addition, you can also use .txt files instead of .mem files.

**Exercise 5.3 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).**

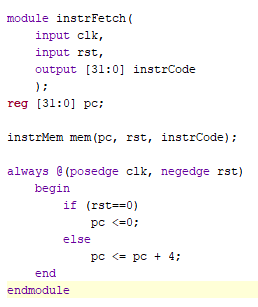
The instruction fetch unit has clock and reset pins as inputs and Instruction code as output. Internally it has a PC register which holds the address of current instruction. It also has an adder to compute PC + 4. The partial code for instruction fetch unit (without instantiation of instruction memory)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



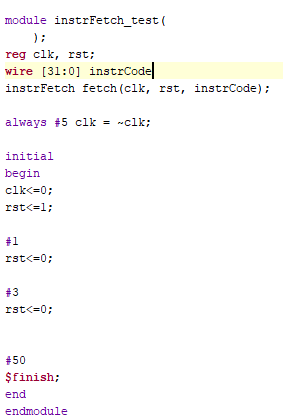
1. **There is an error in the code above. What is the error and what should be done to solve this error.**

Answer: Instruction code is a register in the above figure, it’s supposed to be a wire. In addition the current program counter doesn’t take branch instr into account as programCounter only increments by 4 each time.

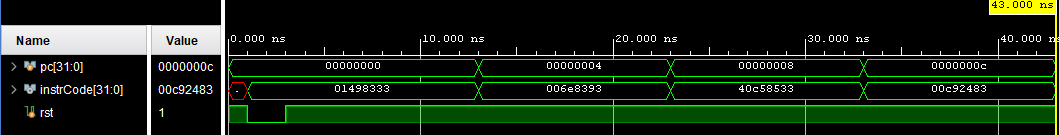
1. **Copy the image of completed Instruction fetch module?**

Answer: ****

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

Answer: 

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

Answer: 

1. **List the concepts you learnt from this experiment (Conclusions/Observations)**

Answer: In this lab we learnt to implement instruction fetch and instruction memory modules of single cycle processor. Further we learnt how to store memory in different forms.