THIS DOCUMENT IS CREATED TO UNDERSTAND THE CODE

- **How to Load Instructions and Data into the processor ?**
  Instructions memories and Data memories has to be copied into the code only in **binary** form even before to start the simulation.
- **How to Encode the Instructions into Binary form ?**
  Please read the Supported Instructions given in report two to understand how to define instructions.
- **What makes this Implementation Simple ?**
  At each posedge clock, the always block will execute all the 5 stages and those 5 stages are linked through pipeline registers such that practically they work like a Multi Cycle Processor.
- **How does this processor's stall its cycle when there is a Jump ?**
  When we identify the Branch or Jump at EX or ID stages respectively it creates a signal and that signal is propagated making the others freeze to work. These signals are StallID, StallEX, StallMEM, StallWB. That is the reason why we check for the stalling of previous stage at the beginning of every stage.
- **What is the output of this Processor ?**
  A Extra Register called Result is made as output of this entire processor.
- **Brief Explanation of CODE !**
  - All the values are initialized in INITIAL block.
  - At the beginning of always block all status signals are made zero, the reason for this is that Suppose for example there is a JUMP and because of it PCSrc with be high, so we get the new address in the next cycle and it should continue with PC + 1 and to do that PCSrc must be set again to low. This is job is done here, but fortunately because of Procedural Assignments all the assignments made in this clk cycle will take place effect in next clk cycle. Now something is bothering in your mind like if every time we making all these signals zero, how will they change ?. For this we exploited an important property of ALWAYS block that all the instructions execute in parallel, so when ever we need to update these signals in that clk cycle there will be two assigments one making zero and the other changing it in the middle of the code since most recent updates are stored, it will save the new value.
  - Now the PC Loads the Instruction from the memory, stores it in the Pipeline register and PC itself is copied into Pipieline register to calculate the new address.

- From here in every stage we check for the stall signal is active or not and start doing it's job. Now in this ID stage if checks for Data Forwarding and assigns corresponding signals.
- Execution stage is simple to understand from the code, based on the signals from the ID stage this continues to work.
- Memory and Write back stages aren't too complex to understand from the code itself.