**CPU Design Project Report Part-6**

By Varun Akula and Jayanth Vuddagiri

The first important take away for us from the project is about the flow of the CPU design process. The division of the process into different sections (the six steps/parts specified by Dr. Agarwal) aids a lot in the design process and helps to keep the designer organized. Hence it is very important to follow them perfectly in the order specified and be thorough with each step before moving on to the next step. The design process also gave us a chance to explore the different terms in which the design can be thought of, choose the best one that suits us and get accustomed to it. Being able to think clearly about the different concepts involved in a CPU design like the instruction format, pipeline timing diagrams is crucial to the way in which a design is implemented and eases the process of design. One important advantage that comes out of a good way of visualization is that it makes it easy to identify most of the hazards and possible ways in which the design would not work or needs to be improved. A good knowledge of the compiler quickens the process of debugging. Consistent naming also helps in keeping track of the code. Proper coding habits must be followed and any inconsistent code must be changed completely and made consistent with the rest of the code instead of opting quick ways to fix the problem without addressing inconsistency, poor readability, redundancy and the overall heuristic change in the design.

While attempting a similar problem in the future we would try to explore alternative designs and implementations of these designs like trying different instruction sets, examine how different instruction sets will affect the time the compiler takes to implement a given C code in the given assembly language instructions of the ISA and also examine different ISAs which might minimize the length of assembly code required to implement a given C code. We also would like to try out different ISAs in order to check which ones would make it easy to compile a given C code. We would try designing the data path in order to facilitate effective synthesis based on timing or available hardware limitations and to provide better timing. We would also like to implement advanced architectures like super pipelining, try including flags and interrupts and also designs with a different register configuration like having more access to the program counter etc.

We would advise others working on a similar project to start with an effective design and debugging procedure more importantly a strategy to deal with the complexity of the design and overwhelming amount of things to keep track of. What helped us was a good naming strategy and a good graphical representation that helped in keeping track of the outputs of pipeline registers after each clock cycle. Naming signals has been very crucial for us. One useful signal format is: InputBlockName \_OutputBlockName\_SignalNameAsPerItsUse, especially when dealing with large number of components. They could be split and combined wherever necessary in order to maintain the naming consistency. It might help to take note about timing of memory components and the register file especially if it is a pipeline since it decides what instruction is read or written in what cycle and whether they should be given a clock, or a rising edge, a falling edge or just left as asynchronous non sequential circuits. We would like to suggest that using both falling and rising clock edges at a few instances can solve some problems. We would also like to remind about the reset signals to a register and the decision about what gets a preference over the others especially while designing the halt instruction data path. Finally, one should always keep in mind the guidelines for effective synthesis while writing the VHDL code.