Milestone 1 Memo:

We have started the top-down design process, and our vision for the processor design will:

- Implement a load-store design
- Make use of 16-bit words
- Use 16 general purpose registers
- Support 4 different instruction formats
- Support 16 instructions
- Support I/O using the display
- Support interrupts using the display
- Execute instructions using pipelining

As of Milestone 1, we've set the "done" criteria for our processor but have not begin the implementation.

Milestone 2 Memo:

We began with the RTL of our 16 instructions from Milestone 1, then we decided to sketch a diagram for the entire processor, assuming a multicycle datapath. From this diagram, we took each component and documented their purpose, input, output, control, and tests. This was an effective approach since seeing how everything would piece together was a more insightful way to look at the individual components.

Milestone 3 Memo:

We've specified every component needed to operate the RTL's, and have implemented the missing components in Xilinx, along with unit tests for these components. Our next step is to follow the integration test plan, and begin constructing the full datapath.

Milestone 4 Memo:

Began integration testing with components from previous milestone. Build Program Counter (PC) register, Main memory, Memory Data Register (MDR), and Instruction Register (IR). Each new component build has been rigorously unit tested.

Began using version control for file sharing of data path files.

Created control unit in Verilog which properly sets control signals for each cycle of each instruction. Standardizes each instruction to 5 cycles (Might change this).

Milestone 5:

We discussed how to handle interrupts and came up with several valid methods. Each valid method was given careful consideration with the Professors input. Once we had finished evaluating each method, we began implementing what we had thought to be the best and most suitable method of handling interrupts for our project.

We made the Interrupt Register, updated the control unit, added additional logistics coming into control for interrupt handling, and began writing the Interrupt Handler code.

Integration testing for all components apart from the components related to Interrupts were completed and approved by each team member.

Milestone 6:

We finalized and implemented control into our data path and rigorously tested each of our instructions. Some instructions were slightly modified to fix any latency issues. The Assembler was also modified to meet the demands and changes of each instruction.

Once the instructions were tested and confirmed to be working, we implemented the entire Euclid's Algorithm code base and tested it. The code was modified due to unforeseen errors and issues. The algorithm was then used to test values 1, 3, 23, and 120.

With Euclid's working, we began implementing all of the Interrupt components and tested interrupt specific instructions. The instructions and Control were modified to fix any unexpected bugs.

With everything working, we began to gather or final data collections.