# Project Plan

Team Jordan^2
Jordan Faith
Lucas Gregory
Benjamin Matase
Jordan Voves

**Overview:**

The purpose of this project is to create a behavioral verilog module for a MIPS pipelined CPU. In phase one, we created a single cycle MIPS processor, and now we must consider how to correctly implement a pipelined version given what we know about hazard. In this project, each instruction will be broken down into five stages: Fetch, Decode, Execute, Memory, and Writeback. We must implement this iverilog processor so that it can support a C program with a simple printf, a fibonacci program, and a program that we create (with 25 or more instructions).

**Goals:**

- Use an iterative approach to produce a pipelined processor. By creating the pipelined CPU to only have the functionality for a small set of instructions, it becomes easier to tackle and focus. Every iteration will build on the previous.
- Create a very modular architecture with good design principles in mind. By making modules generic and keeping hard coding out of the code, we will be able to re-use modules in other places.
- Unify the coding style so that working on another developer's code is easy and quick.
- Splitting the work evenly across members to share the workload as well as ensure everyone gets an opportunity to learn how the CPU works and get better at writing and debugging Verilog.
- Keeping organized by using a Kanban methodology to track the tasks needed to be done. By having a board and the status of each task, we will be able to coordinate our efforts to be as efficient as possible.

**Roles:**

We decided to not have a product manager, but instead each member to be in charge of keeping himself in line and keeping others in line. With the Kanban board, we will be able to track our own and other members progress.

**Organization:**

We are planning on splitting our progress into three phases: First, implementing the hello world. Second, implementing fibonacci. Third, implementing our own program.

Plan for each:

- Hello World
    - This program requires the functionality of the following MIPS instructions
        - Addiu
        - Jal
        - Jr
        - Li
        - Lui
        - Lw
        - Move (pseudo-add)
        - Ori
        - Sw
    - Our plan is to go through the pipeline diagram and map out each of these instructions. Once we have done that, we will know which modules, control bits, and auxiliary functionality will be required in order to get hello world working.
    - Once we know the functionality, we will delegate each task across the four members of our team.
    - Each module will be written and tested against in a controlled environment. Once it is confirmed that a module (take ALU for example) can work locally, we will begin to piece them together.
    - After bringing individually tested modules together, we will test the entire pipeline with our own code before trying it on hello world.
- Fibonacci
    - This program requires the following MIPS instructions:
        - addiu, addu, andi, b, bltz
        - bne, bnez, break, div, jal (break only used for div/0)
        - jr, li, lui, lw, mfhi
        - sra, subu, sw
    - The strategy for Fibo will be very similar, modeling each instruction's path and understanding the required functionality. Since we will already have implemented many of the modules, some of the overhead might come from modifying / improving existing modules.
    - The delegation, creation, testing, and consolidation of modules will be carried out in the same way as specified in hello world.
- Our own program
    - We will come up with a program that challenges us and adds at least three new instructions to our pipeline.
    - Some ideas
        - A factorial counter
        - Fibo with multiplication

**Schedule:**

| Milestones | Description | Milestone Criteria | Planned Date |
|---|---|---|---|
| M0 | Start Project | Complete Contract and work plan | 9-26-17 |
| | Define all goals and acceptance criteria for final project | Determine workflow | |
| M1 | Start integration of existing systems | Choose one version of existing code, or multiple versions, to refine and build on for the rest of the project. | 9-28-17 |
| | Setup initial systems for pipeline register memory and begin basic hazard unit coding | Finish full integration and begin work on inter-system registers | |
| M2 | Complete registers and continue work on the hazard unit | Ensure that all systems have the correct memory access for functional registers in a zero-hazard environment | 10-5-17 |
| | | | |
| M3 | Finish hazard unit | Make sure the system is able to fully detect and reroute hazards | 10-10-17 |
| | Finished is not defined as tested and debugged, but simply built and ready to be assessed. | All architecture must be in place to start wiring the system together. | |
| M4 | Wire together are points and start the debugging process | Ensure that everything is properly connected and begin debugging | 10-12-17 |
| M5 | Ensure all debugging and programs run without error | Make sure the 3 programs we need run correctly | 10-17-17 |
| M6 | Present the project | | 10-19-17 |

**Policy explanations:**

- Everyone must be a hard worker.
  - Everyone must ensure that their work is done on time and in full.
  - Work should not be handed in partially complete or sloppily

- Everyone participates in the module productions, they each contribute meaningfully to the final result.
    - An even amount of work should be done by each member. Tasks on the Kanban board should be evenly distributed between members.

- Everyone should fully test modules they create.
    - Ensure your modules are as bug free as is possible. It is understood that full testing is not available until integration is achieved, but basic debugging should be done before submission.

**Version History Management:**
We will use Gitlab to manage all version history. We are utilizing the built in trackers on git to ensure that we are maintaining a steady pace in development.
Finally, each individual will be working in their own private branch, merging into a developer branch only after reviews and check ins have been completed. The branches will be organized through feature, bug, and refactoring. This will allow us to keep good records of our progress and ensure we have a solidly managed version history.