Project Report

# Problem Statement

The assigned task was to design, implement, and test a Mips Single Cycle Processor capable of executing the given instructions. I wish to be frank and honest; I did not succeed. My given program is incapable of executing certain, key commands which, of course, result in the entire system not functioning. This is truly disheartening, but the work that remains is high quality, and I imagine that I am not far off from a full implementation, but I can delay no longer. The specific instructions that have not been implemented are the syscall functions, and the Jump and Link. These are likely not difficult instructions to implement, as they only require access “dot notation” access to the relevant registers to perform the operation, in the case of Jump and Link, and a few simple funct field parsing’s for the syscall, but, as I said, I had difficulty in other areas which prevented this implementation. To the best of my knowledge, the program as written is capable of cycling though arithmetic type, branching, jumping instructions, (jump and jr), all I type and R- type instructions requested, including load word, store word, all shifts, and all logic comparisons. To the best of my knowledge, this program, if it was given a test bench that fully prints output, would be capable to executing most, (but sadly not all), mips programs using only these functions. It is my hope that in this report I can provide greater understanding an analysis of my design, note the flaws and successes, and perhaps learn for the future.

# Approach in Design and Short Comings

I feel that it would be beneficial to first begin with the process by which I came to my current design, and then begin a more detailed discussion of the individual modules and how they came to be, including what they lack. I find, in review, that much of my time loss and sticking points were a general unfamiliarity with the mips architecture design it self. I consulted the mips cycle diagram, (and even received a slightly modified one from my professor:

A screenshot of a computer screen

Description automatically generated

*Notice the addition of hardware, and the labeling of the Multiplexers, this is vital for future reference when discussing the purpose of these modules. Also note the floating AND gate, which served as both my greatest enemy and momentary savior.*

I an effort to guide the design and limitations of each module, and the hardware between them, I attempted to maximize the amount of logic that exists in the modules themselves, and minimize the amount of logic conducted by the testbench. This decision, which I understand now was artificially imposed, was a grave mistake. You see, as shown by the floating AND gate, and the “four” input to the PC Adder module, this diagram is woefully incomplete. So much so, that my professor provided additions. As such, I spend too much time trying to work within impossible limitations, and ended up with a lot of work that simply needed to be scrapped, and an overall design that is likely bloated and overly complex. It didn’t help matters that I consistently failed to realize the parallel nature of this system. I would attempt to conduct the operations needed to perform, for example, the “link” in Jump and Link, which required access to the PCUnit, incrementing it by an amount, and then adding an immediate. This process should have been done in stages, by multiple modules working together. I however, wasted a full day trying to get it to be done within only the ALU. Ultimately, these operations should be done in the testbench, and would have likely taken at a minimum 3 cycles to complete.

It was these decisions that ultimately lead to the failure of the project.

On a more optimistic note, however, the revisions and additions to the existing modules as a result of feed back from previous projects, (the absence of professor assistance and guidance not withstanding), are a highlight. The notations of program 4 provided insight into some mistakes regarding my ALU and ALU control modules, notably that I had signals that had too few bits, and thus could not distinguish between many different instructions. This forced many instructions to share ALU operations, which, as I discovered, was a significant mistake. While this did take some time to rectify, it was, and after looking about online, I am confident that I have a robust and strong ALU design, and that it performs its duties excellently. While such praise for one’s work is certainly a bit gregarious given both my track record of shoddy work in the class, and the ultimate failure of the final project, I feel it is justified to take some pride in my work, even if it does not meet expectations. It is mine after all, and I did build it.

A screenshot of a computer

Description automatically generated

*In this screen shot of notes from my professor, he provides the solution to my syscall, Jump and link, and PC incrementation issues. If I had spent my time making use of the test bench, and its access to data from all modules, I likely would have had a solid platform for debugging. Alas, never to be.*

# Solution and Design Discussion

Here we begin the real meat and potatoes, a detailed analysis and discussion of my own work. Harsh if fair grading notwithstanding, one is always one’s own worst critic. We will begin with modules that are new to this program, and have not appeared in previous programs, and later discuss how the older modules improved and changed over the course of the project. Note, some modules are so small and specific that the introduction preceding them in their module files is more than sufficient. In addition, I would encourage that the individual module files be looked at, as the project5 file, which contains all the modules for ease of testing, often lacks notes and preamble, as revisions were frequent. If possible, I would strongly prefer that the individual module files be looked to when assessing the quality of comments and notes.

A screenshot of a computer screen

Description automatically generated

While an empty, no text compile does nothing to prove the operations conducted by the program, it should at least be noted the program complies and runs.

A screenshot of a computer

Description automatically generated

Let’s take the single most important module as our case study, The ALU, or in my case ALU2.

The naming convention is odd, and inacrrate, but I decided to keep the name in as a personal joke, more information can be found in the ALU1 file. This module serves as the ALU, performing operations on registers and immediates to pass to data memory and the rest of the cycle. Initially, as of project 5, this was a tiny module, with only 6 operations. After increasing the size of the ALuControl signal, allowing for more instructions to be added, every single instruction that is performed by the ALU has been added, for a total of 25. This was necessary, as the wide variety of instructions required by the code file necessitated all of them. Unfortunately, in my desire to place the j type instructions in the module, rather than have them exist in the testbench, I wasted too much time, and failed to build out the syscall instructions. You can still see a gap where the syscall logic should go. Despite this obvious failing, the increased size of the control signals a complete remapping of the Control to ALUControl to ALU module path, which, while a lot of work to get it all right, was well worth it, and by far the most robust portion of my code.

The module has functionality HI and LO registers to deal with division and multiplication, as well as signage to distinguish signed and unsigned operations. It also has a wide range of operations contained within a case statement, all labeled. It includes a zero control singal, to push to the branching logic, needed to properly branch on those instructions, and, of course, it is clocked to ensure stability and timing issues are kept to a minimum. One area in particular I wish to call attention to, the case statement where the instruction operations are actually performed. This code shows an understanding and some degree of skill in Verilog syntax, and towards the end, after learning some new code to write more efficiently, even evidence of improvement. While it is ultimately unfinished and is marked as the only standalone module to be so, it is an example of my best work.

A screenshot of a social media post

Description automatically generated