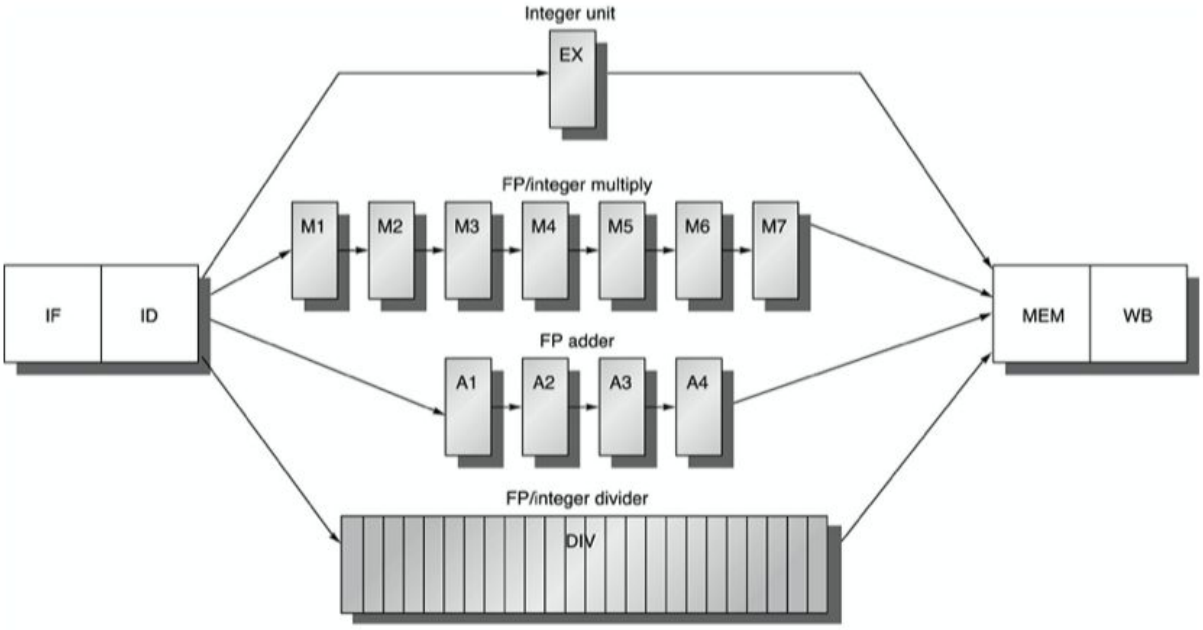
Bennett James
ECE563-001

5 Stage Pipelined Floating Point and Integer Data Path

To handle floating point commands I took my design that I used to successfully implement a 5 stage pipeline for integer operations and added the ability to handle multiple execution units as well as execution unit delays. Several of the new opcodes that needed to be included were ADDS, SUBS, MULTS, DIVS, LWS, and SWS. The overall structure of how the new execution units would be handled can be seen below.



The main changes I had to make to my design was adding the ability to accept and accomodate for the four different execution units as well as the potential to have multiple instances of the same units. To determine how to and when to insert instructions into their correct execution unit by parsing through opcode to determine destination. When to release instruction from execution units to the MEM stage based upon when busy times expired. Last new feature added was the ability to handle stalls caused by the EXE stage and adding WAW hazard detection and mitigation. To mitigate WAW hazards I included a function that would check through all of the execution units and determine if the destination was the same for a unit and the instruction in the ID stage. If this was the case, it would stall passing the current

instruction to the EXE stage until the instruction's busy time in the execution unit was one less than the busy time for the ID stage instruction.

Outside of the several things I had to add, there were several things I had to modify from the integer pipeline which include my way of implementing control stalls and data stalls (RAW). For control stalls my integer pipeline did not attribute for the variable exec units and assumed zero as this was the timing in the integer pipeline. The number of stalls I made was static and therefore since the branch opcodes all go through the integer pipeline and in most cases the delay for this was one cycle, I would only stall 2 cycles when I needed 3 to accomodate for the execution unit stall. For data stall, I had a similar issue as I would look at opcodes of the MEM and WB stage neglecting the EXE stage as that was the stage I was immediately adding stuff to and had already been passed to the MEM stage from the previous cycle. For those of the issues I adapted a flag variable over a set counter which would check if a stall is needed after every cycle. For data stalls specifically RAW this included adding logic to look through execution to determine if delays were necessary.

For my performance, I was able to achieve the correct clock cycles, stalls, and instructions executed for each test case which in turn gave me the correct CPI. All of my registers were correct except for on test case 1 and 2 the output of the hex representation of a DIV operation was slightly off but the decimal value was correct. With this one issue not affecting the decimal value of my number I believe I was able to achieve 100% successful implementation of the FP pipeline. I did not make any modifications to the makefile, all changes were done in the .cc or .h files.