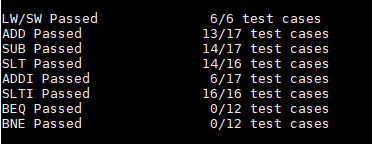
CoE 113 ME3

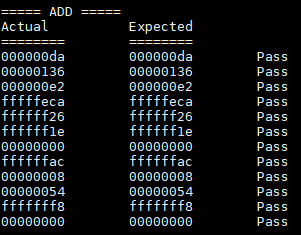
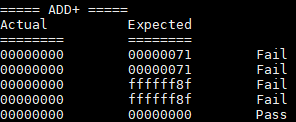
**1 Instruction set test results**

The results of the test are shown below.



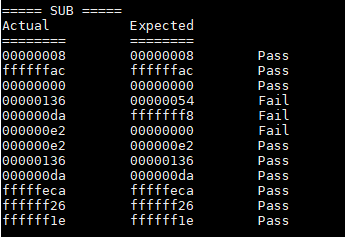
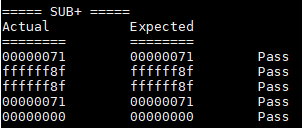
**2 Erroneous intructions**

For the ADD instruction (ADD+ in this case just means additional test cases that were added later), the first 12 cases had only non-zero inputs, while the five additional cases had at least one zero input. For the five additional cases, all the outputs were zero, which means that whenever at least one input is zero, ADD immediately outputs a zero.

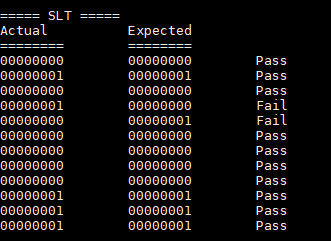
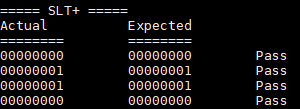
 

For the SUB instruction, it gave a wrong output for the test cases where both inputs were negative. For example, the first failed case was -113 –(-197) = 84 or 0x54. However, the output was 0x136 or 310, which is the result of 113 + 197. The same thing happened with the next two cases.

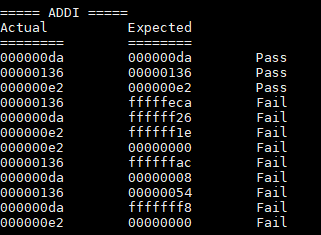
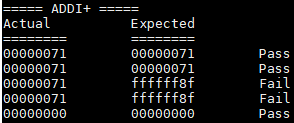
This error may be caused by the instruction being implemented to check if both inputs are negative. If they are, the first input is turned positive, then subtracted with the 2nd input. So instead of the instruction doing , it does . For the other test cases, this did not happen.

For the SLT instruction, errors happened with the test cases where both inputs were negative and not equal. The errors were similar to the errors in SUB. When both inputs are negative, they are first turned positive before being compared, which results in the output being the opposite of what is expected. Note that when the inputs are both negative and equal, the output is still correct.

For ADDI, the errors happened when one of the inputs were negative. For the first three cases, both inputs are positive, so the outputs are as expected. For the next 9 cases, however, if an input is negative, it is turned positive first before being added. So, if both inputs are negative, both of them are turned positive before being added. The same thing happened in the five added cases under ADDI+. For the two failed cases, one of the inputs were negative, which resulted in it being turned positive first before being added.

For BEQ and BNE, the errors happened on all test cases. In the test assembly code, a value was first stored into data memory, then BEQ/BNE was called, then another value was stored on the same data memory address as the first, which would overwrite the first data stored if the branch does not succeed. This means that for BEQ, if the inputs are equal, the 2nd SW call should be skipped. Similarly, for BNE, if the inputs aren’t equal, the 2nd SW call is skipped.

For the first BEQ test case, the inputs aren’t equal, so the 2nd SW call should be executed, which stores 0x69 in memory. For the third test case, the inputs are equal, so the 2nd SW call should be skipped, which means that 0xffffff8f isn’t overwritten in memory. Following this, it is clear that the opposite of what is expected is happening for all test cases, which means that the implementations of BEQ and BNE were swapped.

