**Lab 0: MIPS and Broken SPIM**

*Shane Cincotta*

***Abstract:***

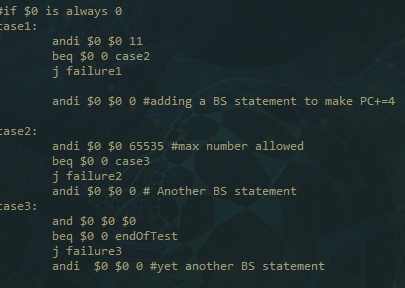
The goal of this lab is to develop test cases for various MIPS instructions which may or may not contain bugs. The overall goal of this lab is to build a foundation of building diagnostic test cases, as well as identifying bugs using these test cases.

Throughout this lab, I tested 10 different MIPS instructions and was able to find 3 bugs in 3 different instructions. The ten instructions I tested were as follows: addu, addiu, and, or, andi, slt, slti, beq, bne and j. The 3 buggy instructions I was able to find were beq, bne and slti.

***Detailed Strategy:***

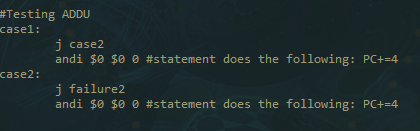
To begin testing the 10 instructions, I first had to verify that a few basic instructions worked. Once these instructions were verified as working, I could then begin building test cases with these instructions to test the remaining instructions. The order and methodology of testing the instructions were as follows:

1. **Testing $0 always contains 0d0:**



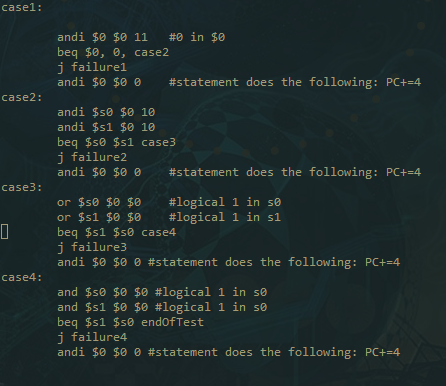
* To test if *$0* always contains 0d0, I used 3 test cases.
  + The first test case uses *andi* to put 0d11 into *$0*.
    - I then compare *$0* to 0d0 using BEQ
  + For my second test case, I use *andi* to put 0d65535 into *$0* and compare *$0* to 0d0 0 using BEQ
  + For my final test case I use *and* to put *$0* into *$0*.
    - I then used BEQ to compare *$0* to 0d0
  + All of my test cases passed

1. Instruction: **Jump**

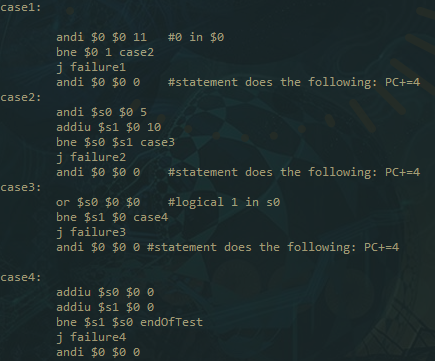


* To test the jump instruction, I used 2 test cases.
  + The first test case jumps to case 2, and case 2 jumps to failure 2.
  + Both cases passed my tests

1. Instruction **BEQ**:

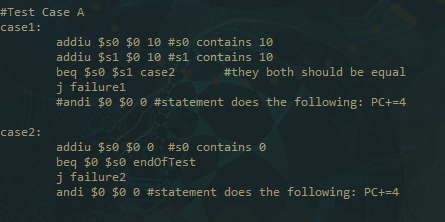


* To test BEQ, I used *and, andi* and *or* statements.
  + After testing I realized that my jump statement in BEQ was jumping to the wrong instruction. Through further debugging using the debugger, I realized that the address BEQ was jumping to was 4 short of the correct address.
    - To solve this, I added the statement *andi $0 $0 0*, this will increase the PC by 4 without altering the program (Since *$0* is always 0). After implementing this fix, my BEQ statement was fixed.

1. Instruction **BNE**:

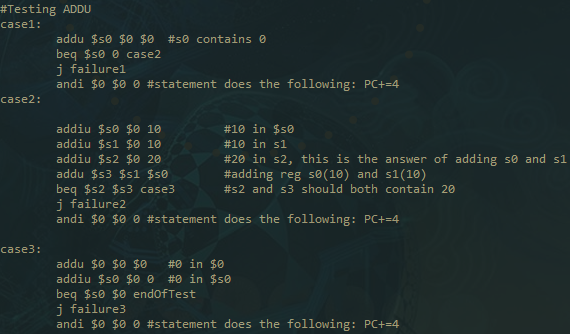
* To test BNE, I followed the same methodology as BEQ. I again found the same error (jump case was 4 short).
  + To fix this, I again used the statement *andi $0 $0 0*.

1. Instruction **ADDIU**:



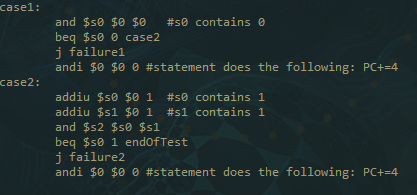
* To test ADDIU, I used 2 different cases
  + The First case puts 0d10 into register *$s0* and 0d10 into register $s1. I then compared *$s0* & *$s1*, the results should be equal if *ADDIU* works correctly
  + The second case puts 0d0 into register *$s0*, I then compare this to register *$0* (which should always contain 0d0).
    - Both test cases passed.

1. Instruction **ADDU**:



* To test *ADDU*, I used 3 test cases.
  + The first test case puts the register *$0* into register *$s0*. I then compare *$s0* to register *$0* and compared with BEQ (they should be the same)
  + The second test case put 0d10 into register *$s0*, 0d10 into register *$s1*, 0d20 into register *$s2.*
    - I then use *addu* to put registers *$s1&s0* into register *$s3.* Then I compared *$s2&s3* (they should be equal) using BEQ.
  + For my final test case, I used *addu* to put *$0* into register *$0* and *addiu* to put 0d0 into register *$s0* and compared *$s0* to register *$0* and compared with BEQ (they should be equal).
  + All my test cases passed

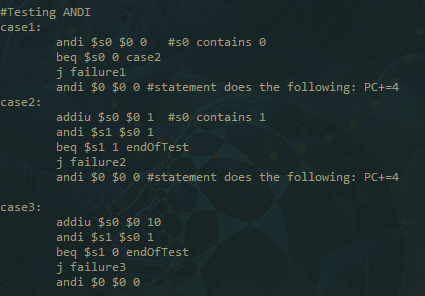
1. Instruction **AND**:



1. Instruction **OR**:

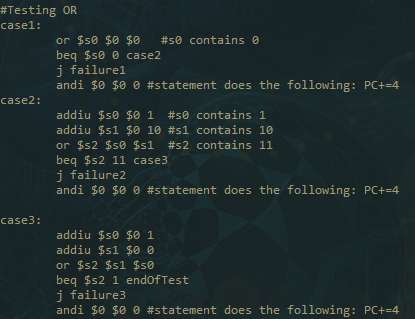
* To test the *and* instruction, I used 2 test cases.
  + The first test case put *$0* into register *$s0*
    - I then check to verify that *$s0* contains 0d0
  + The second test cases put 0d1 into register *$s0* and 0d1 into register *$s1*
    - I then used the *and* statement to and the results of *$s0* and *$s1* into *$s2*, the result should be 0d1 into *$s2*
  + Both of my cases passed the test

1. Instruction **ANDI**:



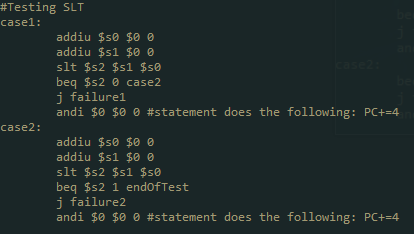
* To test *andi* I used 3 test cases.
  + The first test case places 0d0 into register *$s0*.
    - I then use BEQ to compare *$s0* to *$0*
  + The second test case places 0d1 into *$s0*. I then put 0d1 and *$s0* into *$s1* using *andi*.
    - I compared *$s1* to 0d0 using BEQ (they should be equal).
  + For the final test case, I place 0d10 into register *$s0* and use *andi* to and the results of *$s0* and 0d1.
    - I compare *$s1* to 0d0 using BEQ (they should be equal).
  + All my test cases passed the test

1. Instruction **OR:**

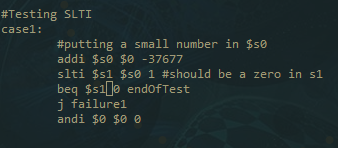
****

* To test *or*, I used 3 test cases.
  + In the first test case, I *or* *$0, $0* and place the result into *$s0*.
    - I then compare to 0d0 using BEQ (they should be equal).
  + In the second test case, I put 0d1 into *$s0* using *addiu*, 0d10 into *$s1* again using *addiu*.
    - I then used *or* on registers *$s0&s1*, I put the result into *$s2* and compared *$s2* to 0d11 using BEQ.
  + For the third test case, I used *addiu* to put 0d1 into *$s0* and again used *addiu* to put 0d0 into *$s1*.
    - I then used *or* on registers *$s1&s0* and placed the result into *$s2*. I compared the result to 0d01 using BEQ.
  + All my test cases passed

1. Instruction **SLT**:



* To test *SLT*, I used 2 test cases.
  + For the first test case, I used *addiu* to put 0d0 into registers *$s0&s1*
    - I then used *slt* to compare the registers and store the result in *$s2*
      * I then compared $s2 to 0 using BEQ(should be equal).
  + For the second test case, I followed the same procedure except I compared *$s2* to 0d1.
  + All my test cases passed.

1. Instruction **SLTI**:

* To test *SLTI*, I only needed one test case
  + For my test case, I used *addi* to place 0d-37677 into register *$s0*.
    - I then used *SLTI* to compare *$s0* to 0d1, I placed the result into *$s1* and compared it to 0d0 using BEQ.
  + My test case failed. So, I used the debugger tool to step through my program. I found that *SLTI* was not extending the sign correctly. This was the third and final bug that I found.

***Results:***

As a result of my testing, I was able to find 3 bugs in the 10 total instructions.

* Bug 1
  + BEQ not branching to the correct address
* Bug 2
  + BNE not branching to correct address
* Bug3
  + SLTI not sign extending properly

***Conclusion:***

To recapitulate, I began this lab by testing a few basic instructions worked. This included that *$0* always contains 0d0. Then I tested that the *J* instruction worked. Once I had these 2 basic tests completed, I began systematically testing 10 instructions. I was able to identify 3 bugs in 3 instructions. The 3 bugged instructions were, *SLTI, BNE* and *BEQ.* Within *BNE* and *BEQ*, the branch was not jumping to the correct address. I fixed this in both instructions by adding a statement that had no effect. In *SLTI*, the bug was that the sign was not being extended properly. I tested this by using *SLTI* on a very small negative number and using the debugger to view the values.