Testing LC3 Assembly

Testing

- Loads of input.
- Could be anything.
  - A value in a register
  - A value at a location
  - A memory address containing a value
  - A string
  - An array
  - And even user input
- Output can be any of the above as well.
- How do we test these easily.

Lc3 test

- A program that takes in xml files and your code and automatically runs the test-suite found in the xml file against your code.
- Automatically handles randomizing memory and setting parameters for you and checks the all outputs given. If it doesn't match then the program will alert you.
- The xml file specifies test cases and in each test case you can tell how input is given, some other various parameters of the test and what is the expected output.
- Usage: lc3test testfile.xml asmfile.asm

Example problem specification

- Write a program that adds two things.
- The input will be located at symbols U and V
- You are to store your answer (U + V) at symbol ANSWER

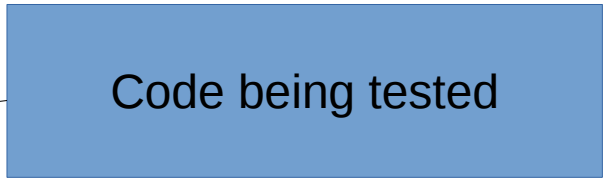
Example program under test

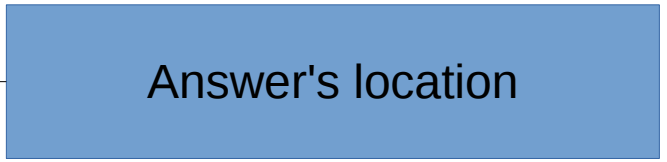
```
.orig x3000
        LD R0, U
        LD R1, V

        ADD R2, R0, R1
        ST R2, ANSWER
        HALT

        ; PARAMS
        U .fill 2000
        V .fill 8

        ; ANSWER
        ANSWER .blkw 1
.end
```





Example test case xml

```
<?xml version="1.0"?>
<test-suite>
    <test-case>
        <name>2 + 3 = 5</name>
        <has-max-executions>1</has-max-executions>
        <max-executions>1000000</max-executions>
        <randomize>1</randomize>
        <input>
            <test-value><address>U</address><value>2</value></test-value>
            <test-value><address>V</address><value>3</value></test-value>
        </input>
        <output>
            <test-value><address>ANSWER</address><value>5</value></test-value>
        </output>
    </test-case>
</test-suite>
```

Name the test case

Randomize and give
Limit to number of instructions executed
Program could run infinitely without it
if it had an infinite loop

Handle parameters

Expected output in memory address

Specifying Input

- Values
- Pointers (Addresses)
- Register
- PC
- Arrays
- Strings
- Standard Input

Values

- <test-value><address>ADDR</address><value>VALUE</value></test-value>

- Test runner will substitute the value at memory address specified by ADDR with VALUE.
- ADDR can be any address or symbol or an expression.
- VALUE can be any expression.

Pointers

- <test-pointer><address>ADDR</address><value>VALUE</value></test-pointer>

- Test runner will treat the value in the memory address specified by ADDR as a pointer (memory address) and replace the contents of address pointed to by the pointer with VALUE.
  - Essentially MEM[MEM[ADDR]] = Value
  - Similar to LDI instruction
- ADDR can be any address or symbol or an expression.
- VALUE can be any expression.

Register

- <test-register><register>R0-R7</register><value>VALUE</value></test-register>

- Test runner will set the register given to the specified value.
- Register must be any register name R0 through R7
- VALUE can be any expression.

PC

- <test-pc><value>VALUE</value></test-pc>

- Test runner will set the PC to the specified value, the default is x3000
- VALUE can be any expression.

Array

- <test-array><address>ADDR</address><value>COMMA_SEPARATED_EXPRESSIONS</value></test-array>

- Test runner will treat the contents of ADDR as a an address and store the values computed by COMMA_SEPARATED_EXPRESSIONS at MEM[ADDR] sequentially
  - That is MEM[MEM[ADDR]] = ARRAY[0]
  - That is MEM[MEM[ADDR] + 1] = ARRAY[1]
  - And so on...
- ADDR can be any address or symbol or an expression.
- COMMA_SEPARATED_EXPRESSIONS is a list of comma separated expressions.

Strings

- <test-string><address>ADDR</address><value>HELLO_WORLD</value></test-string>

- Test runner will treat the contents of ADDR as a an address and store the string given (including the nul character at the end) at MEM[ADDR] sequentially.
  - That is MEM[MEM[ADDR]] = H
  - That is MEM[MEM[ADDR] + 1] = E
  - …
  - MEM[MEM[ADDR] + 11] = 0
- ADDR can be any address or symbol or an expression.
- HELLO_WORLD is a string without the quotes.

Stdin

- <test-stdin><value>STRING</value></test-stdin>

- Test runner will use the string given as input (without the nul terminator)
- ADDR can be any address or symbol or an expression.
- HELLO_WORLD is a string without the quotes.

Specifying Output

- All the same types except stdin is now stdout
- Xml Tags use the same syntax.
- The only difference is you specify a condition parameter to the <test-type> element (The default just checks if they are equal).
  - Ex. <test-value condition="!=">...</test-value>

Conditions

- They are optional. The default being check if the values are the same.
- Three types integral, string, and array

Integral	String	Array
1. equals, ==, = 2. notEquals, != 3. less, < 4. greater, > 5. lessOrEquals, <= 6. greaterOrEquals, >=	1. equals, ==, = 2. notEquals, != 3. equalsIgnoreCase 4. notEqualsIgnoreCase 5. contains, c 6. notContains, !c 7. containsIgnoreCase 8. notContainsIgnoreCase	1. equals 2. notEquals

Notes

- For the array comparisons the length of the array tested from the code will be equal to the length of the array given in the test
  - that is, if the actual array the code generated is [2, 3, 4, 6, 3, 4, 2, 3] and the expected array given in the test is [2, 3, 4] then it will say the test passed.
  - If you really need to check if two arrays are identical then you will need to also output the size of it. Or alternatively write a value at the past the end of the array and see if it was changed.
- Note for the test-string postcondition. A nul terminator must be written at the end of the string so that the test runner knows where the string ends.
  - If this doesn't happen the test-runner will terminate the string as soon as a value outside the range (0, 255] is found.