Testing is done through xml files the hierarchical structure of these xml files are as follows

test-suite
- test-case
  - name
  - *true-traps*
  - *has-max-executions*
  - *max-executions*
  - *randomize*
  - *interrupt-enabled*
  - input
    - test-value
      - address
      - value
    - test-pointer
      - address
      - value
    - test-register
      - register
      - value
    - test-pc
      - value
    - test-array
      - address
      - value
    - test-string
      - address
      - value
    - test-stdin
      - value
  - output
    - test-value *condition=”condition_integral” points=”integer”*
      - address
      - value
    - test-pointer *condition=”condition_integral” points=”integer”*
      - address
      - value
    - test-register *condition=”condition_integral” points=”integer”*
      - register
      - value
    - test-pc *condition=”condition_integral” points=”integer”*
      - value
    - test-array *condition=”condition_array” points=”integer”*
      - address
      - value
    - test-string *condition=”condition_string” points=”integer”*
      - address
      - value
    - test-stdout *condition=”condition_string” points=”integer”*
      - value

test-suite is the root of the xml file and can have one or more test cases associated with it.

A test case has a name, some parameters associated with it, zero or more preconditions (input) and one or more postconditions (output).

A precondition/postcondition can be one of the following 7 types

- value – Which is just a value at a memory address (you specify the memory address and what value to give it).
- pointer – Unlike value it will treat what's at the memory address you specify as a memory address and store to that location, that is, MEM[MEM[address]] will equal value when the test is run.
- pc – Just stores the value given into the pc when the test is ran.
- register – Given a register R0-R7 will store the value given into the register when the test is run.
- string - Starting at memory address MEM[MEM[address]] it will write the string given (including the \0).
- array – Starting at memory address MEM[MEM[address]] it will write the array given.
- stdin / stdout – Will use the string given as input.

For postconditions as optional attributes you can also specify how to compare the expected and actual results. In addition to this you can also specify how many points to give the postcondition in the case that this file is an autograder.

For checking conditions you may use one of these values (case doesn't matter)

condition_integral

1. equals, ==, =
2. notEquals, !=
3. less, <
4. greater, >
5. lessOrEquals, <=
6. greaterOrEquals, >=

condition_string

1. equals, ==, =
2. notEquals, !=
3. equalsIgnoreCase
4. notEqualsIgnoreCase
5. contains, c
6. notContains, !c
7. containsIgnoreCase
8. notContainsIgnoreCase

condition_array

1. equals
2. notEquals

For the test-string precondition the string will be terminated with a nul terminator

Note 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 students array is [2, 3, 4, 6, 3, 4, 2, 3] and the array given in the test is [2, 3, 4] and the condition is equals then it will say the test passed. So if you really need to check if two arrays are identical make the student output the size of it and then check the size and the values in the array.

Note for the test-string postcondition. The student needs to output a nul terminator at the end of the string so that the system knows when the string ends. In the case the student doesn't do this it will terminate the string as soon as a value outside the range (0, 255] is found.