MIPSUnit: A Unit Testing Framework for MIPS Assembly

Grand Valley State University
Zachary Kurmas & Jack Rosenhauer II

MIPSUnit Overview

- Quickly and easily verify correctness of MIPS assembly code
  - Fast grading!
    - Grading is faster, but not automatic
    - Instructor must still comment on style
- Makes it easy for students to verify the correctness of their own assembly code
  - Find and fix mistakes instead of "writing off" points
- "Testing across the curriculum"
  - Students find testing difficult
  - There is no substitute for lots of practice
  - The more courses that require testing, the more proficient students become

MUnit

MUnit is JUnit (Java) based

- Familiar syntax for students
- Lists each failure
- Tests can't affect each other
  - Each test runs in a new "sandbox"
- Less expressive than MSpec

MSpec

MSpec is RSpec (Ruby) based

- Much more expressive than MUnit
  - Easy to write exhaustive tests
- Generates assembly file containing tests
  - Students need not learn framework if instructor writes all tests
- Students must learn a little Ruby to write their own tests
- Only reports first failure
- Tests can affect each other
  - No way to re-set initial memory states between tests

Contact Information

Zachary Kurmas
kurmasz@gvsu.edu

Jack Rosenhauer II
jackrosenhauer@gmail.com

http://www.cis.gvsu.edu/~kurmasz/Software/

MUnit Usage

Always randomize registers that we check (verifies that students explicitly set $v0)

Places 2 in $a0, 3 in $a1, then runs the code starting at the label "dateFashion"

Uses JUnit's built-in Assert methods

Creates a label in the .data section

Verifies that the assembly code only modified the 8 words returned by getWords

```
1  public class DateFashionTest {
2
3    public void before() {
4      randomizeRegister(v0);
5    }
6
7    @Test
8    public void you_unstylish_date_unstylish() {
9      run("dateFashion", 2, 3);
10       Assert.assertEquals(0, get(v0));
11   }
12
13   @Test
14   public void you_ok_date_ok() {
15     run("dateFashion", 3, 3);
16     Assert.assertEquals(1, get(v0));
17   }
18 }
19
20 public class ReverseTest {
21
22   @Test
23   public void reversePartialList() {
24     //Reverse the first n elements of an array
25     Label array1 = wordData(1, 2, 3, 4, 5, 6, 7, 8);
26     run("reverse", array1, 6);
27
28     int[] expected = {6, 5, 4, 3, 2, 1, 7};
29     int[] observed = getWords(array1, 0, 7);
30     Assert.assertArrayEquals(expected,
           observed);
31     Assert.assertTrue(noOtherMemoryModifica
           tions());
32   }
33 }
```

MUnit Command Line

Compile tests, then run

```
$ javac -cp munit.jar InRangeTest.java
$ java -jar munit.jar in_range_error.asm InRangeTest.class
```

```
Failure: verify_true_at_min(InRangeTest): expected:<1> but was :<0>
Tests run: 5, Failures: 1
```

Uses familiar JUnit output format

MSpec Usage

Call refers to the label on describe block

Always initialize $v0 to 44 (verifies that students explicitly set $v0)

Easy to give tests descriptive names

Concisely give each "$s" register a value to verify that the code maintains preserved registers

Calculate solution instead of hard-coding expected answers

Use Ruby blocks to easily create a separate test for every possible input (this is difficult to do concisely in Java)

Verify each "$s" register's original value was restored after code finishes

```
1  describe :dateFashion do
2    before do
3      set :v0 => 44
4    end
5    it "returns 0 if you unstylish and date is ok" do
6      call 2, 7
7      verify :v0 => 0
8    end
9
10   it "returns 2 if you and your date are stylish" do
11     call 8,8
12     verify :v0 => 2
13   end
14 end
15
16 describe :reverse do
17   data :array1 => [:word + (1..8)].to_a
18   it "reverses the array" do
19     call :array1, 6
20     expected = [:word, 6, 5, 4, 3, 2, 1, 7]
21     verify_memory(:array1, expected)
22   end
23 end
24
25 describe :dateFashion do
26   sregs = {}
27   (0..7).each { |i| sregs["s#{i}".to_sym] = 430 + i}
28   before do
29     set :v0 => 44
30     set sregs
31   end
32
33   def self.date_fasion(you, date)
34     return 0 if (you <= 2 || date <= 2)
35     return 2 if (you >= 8 || date >= 8)
36     1
37   end
38
39   (1..10).each.do |you|
40     (1..10).date do |date|
41       exp = date_fasion(you, date)
42       if "returns #{exp} given #{you}, #{date}" do
43         call you, date
44         verify :v0 => expected
45         verify sregs
46       end
47     end
48   end
49 end
```