Testing Assembly with Ruby

by Ethan Waldo

What the hell?!?

- Writing assembly is hard
- Compilers may optimize some things nicely, but they can't do better than a smart human
- The more layers you create between you and the machine, the more the computer spends time doing unnecessary work
- Embedding asm into C/C++ works but is unnecessary overhead if we're doing just pure asm
- Because I can

Problem I was trying to solve

- I don't have experience with higher levels of statistics and math
- Couldn't find the algorithms I needed online
- Wanted to determine exact probability of a win of one Texas Hold-em hand of poker to another (http://en.wikipedia.org/wiki/Poker_probability_(Texas_hold_'em))
  - ((52*51)/2)*((50*49)/2)*((48*47*46*45*44)/5) = 66,753,144,060,000 (66 billion) for 2 players
  - 2.117 x 10^{28} (21 octillion) possible combinations for 9 players
- Wanted to compute as fast as possible
  - Use registers only
  - No L1, L2, or L3 cache except for asm instructions
  - No memory access

Which flavor?

http://en.wikipedia.org/wiki/List_of_assemblers

- GAS (http://en.wikipedia.org/wiki/GNU_Assembler)
- NASM (http://www.nasm.us)
- FASM (http://flatassembler.net)
- MASM (http://en.wikipedia.org/wiki/Microsoft_Macro_Assembler)
- **YASM (**http://yasm.tortall.net**)**

IDE

- Eclipse (www.eclipse.org)
  - Good for live writing and debugging of assembly
  - Write code in C/C++ perspective, debug in Debug perspective
  - Code Tab group, Memory Tab group, Debug Tab group, Console Tab group
  - Eclipse Script (eclipsescript.org) very complicated
- Rubymine (or other text editor)
  - No direct asm debugging

Hello World

```
.section .data
message: .string "Hello World!\n"

.section .text

# this directive allows the linker to see the "main" label
# which is our entry point
.globl main
```

Hello World Part 2

```
.func main
main:
  mov $0x4,%eax
  mov $0x1,%ebx
  mov $message,%ecx
  mov $0xe,%edx
  int $0x80
  mov $0x0,%eax
```

How to test with Ruby?

- Rubug (https://github.com/mcarpenter/rubug)
  - GAS only
  - Uses MI2 interpreter (http://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI.html#GDB_002fMI)
    - Ecplise CDT uses MI1
  - Uses Treetop (https://github.com/nathansobo/treetop.git) grammar to interpret GDB output
  - Doesn't handle all MI2 output possibilities, but close enough
    - Submitted some fixes to mcarpenter and he merged within a week or so

RSpec Spec Helper

```
require 'rubug/gdb'

ROOT = RSpec::Core::RubyProject.root.to_s
ASM_DIR = File.join(ROOT, 'asm')
BIN_DIR = File.join(ROOT, 'bin')
```

RSpec Spec Helper Part 2

```
RSpec.configure do |config|
  config.before(:suite) do
    `/usr/bin/as -g --gstabs -o
        "#{BIN_DIR}/2P-uFTR.o"
        "#{ASM_DIR}/2P-uFTR.s"`
    `/usr/bin/g++ -o "#{BIN_DIR}/PokerStats"
        "#{BIN_DIR}/2P-uFTR.o"`
  end
end
```

RSpec Spec Helper Part 3

```
def connect_gdb(file)
  gdb = Rubug::Gdb.new
  gdb.file File.join(BIN_DIR, file)
  gdb
end

def set_register(reg, value)
  @gdb.command("-interpreter-exec console \"set $#{reg} =
0x#{sprintf("%02x", value)}\"")
end
```

RSpec Spec

```
require 'spec_helper'

describe do
  before(:each) do
    @gdb = connect_gdb("PokerStats")
  end

  after(:each) do
    @gdb.quit
  end
end
```

RSpec Spec Part 2

```
it "should set r8 to not zero" do
  @gdb.break(:main)
  @gdb.run

  set_register("r8", 0)
  @gdb.register(:r8).should == 0
  @gdb.break(:initialize)
  @gdb.continue
  @gdb.register(:r8).should_not == 0
end
```

Inserting and deleting breakpoints

http://sourceware.org/gdb/onlinedocs/gdb/GDB_002fMI-Breakpoint-Commands.html#GDB_002fMI-Breakpoint-Commands

@gdb.command("-break-insert main")
@gdb.command("-break-delete 2")
    Use -break-list to determine numeric of breakpoint to delete

Also should be able jump to specific instruction
@gdb.command("-exec-jump foo.c:10")

Using Rubug

https://github.com/mcarpenter/rubug/blob/master/lib/rubug/gdb/cli_map.rb
https://github.com/mcarpenter/rubug/tree/master/examples

```
require 'rubug/gdb'

gdb = Rubug::GDB.new
gdb.file(object_file_name)
gdb.run
gdb.break(:main)
gdb.continue
gdb.registers
gdb.quit
```

Using Rubug Part 2

```
git clone https://github.com/mcarpenter/rubug.
git
cd rubug
bundle install
rake grammar
```

put in Gemfile

```
gem 'rubug', :path => '../rubug'
```

OS install gcc-c++ and gdb

In summary

- Assembly can be made more readable and maintainable by a decent test suite
- Assembly now much easier to refactor with testing
- Must easier to write tests in ruby than alternatives
- Can fiddle with register data on the fly for test initialization
- Can add, remove, and jump to specific breakpoints for testing deep functionality
- Now assembly programming can be fun