Research Notes- 10/19- 10/26

To compile my Java files to native machine code, I tried a few things:

* Mono Framework + IKVM: this method was mentioned as an alternative to GCJ and GCC because GCJ is not really supported anymore. However, I was not able to get Mono to install correctly on Mac OSX
* Next, I tried installing GCC directly, which worked, but I was not able to find any way to add GCJ on Mac.
* I also tried installing GCC and GCJ on ArchLinux, but the GCJ installation failed.
* Next, I read about another alternative to GCJ: Excelsior JET, which is meant to allow you to easily compile Java to native binaries. I was able to download a 60-day trial from <http://www.excelsior-usa.com/jet.html>
  + <http://www.excelsiorjet.com/evaluate> 🡪 quick guide

JET

* To install:
  + $ chmod a+x ~/Downloads/jet-1000-eval-en-osx-amd64.bin
  + $ sudo ~/Downloads/jet-1000-eval-en-osx-amd64.bin
* Then had to edit path in ~/.bash\_profile:
  + vim ~/.bash\_profile
  + export PATH=”*JET-home*/bin:$PATH”
  + source ~/.bash\_profile
* To launch, within eclipse src folder, ran:
  + ~/PATH/profile1.7.0\_55/jre/bin.java –jar BangBangController.jar, where the jar file was a Runnable jar with library handling set to “Copy required libraries into a sub-folder next to the generated JAR"
* Running
  + Able to successfully test run BangBangController in Jet
  + Able to successfully run BangBangController executable compiled by JET
* To run outside original build directory:
  + On JET Evaluation, on Compile click Package to open up JetPackII which will create a package so that you can run your executable anywhere else

GDB

* gdb BangBangController

To run BangBangController in Linux

* go to control\_research/native\_code/src and run java –jar BangBangController.jar (non-native version)
* to access the binary files, go into control\_research/native\_code/src/BangBangController-native
  + having trouble running the BangBangController on Clic machines
  + both the exe file and the linux system are 64bit but Clic still won’t run the executable BangBangController
* Tried automatically compiling the native package using ‘xpack “BangBangController.jpn” but that created the exact same files as JetPackII did so that didn’t fix the problem (this also necessitated that the BangBangController-native2 directory is empty)

Since I couldn’t figure out how to get the native code to compile on the Linux machines, I decided to just download and use GDB on my local OSX machine.

To download GDB on OSX, I used the following tutorial (after installing Homebrew for OSX)

* <http://ntraft.com/installing-gdb-on-os-x-mavericks/>

This installed GDB on my local laptop and allowed me to run the binary executable BangBangController using gdb.

In order to use gdb macros, had to first experiment with gdb and learn about breakpoints and debugging techniques. Used this gdb tutorial

* <http://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf>
* <http://faculty.kutztown.edu/spiegel/Debugging/DebugPrimer.htm>

Created simple gdb script files

* commands: <http://stackoverflow.com/questions/10748501/automating-gdb-sessions>
* disassemble: <http://blog.flip-edesign.com/?p=60>
* <http://madapvamshi.blogspot.com/2011/09/writing-gdb-scripts.html>
* <http://www.adacore.com/adaanswers/gems/gem-119-gdb-scripting-part-1/>

However, when I try to run gdb on BangBangController, it cannot find a few OSO archive files.

Debugging Java with gdb tutorial (NEEDS GCJ)

* <http://stackoverflow.com/questions/5096707/how-is-gdb-used-to-debug-java-programs>
* <http://stackoverflow.com/questions/138511/what-are-java-command-line-options-to-set-to-allow-jvm-to-be-remotely-debugged>
* <https://gcc.gnu.org/java/gdb.html> (GCJ tutorial)

Need to add debug command when compile Java program for gdb to be able to be used with it

* created debug\_java\_controller/
* might not work without GCJ

Convert Java files to C++

* <https://code.google.com/a/eclipselabs.org/p/j2c/>

10/27 Meeting Notes

* rewrite in c instead of c++
* run macros with a simple hello world \*\* create a function foo and test each routine individually first and then combine them and run them
* stuck-at-one and other methods where I change the address
  + read out to file, change just one char in the array, and then write back so will need to write to a outside file and then read from it
* /pc or /ip or /eip gives you the pointer for the current instruction while you are stepping
  + $pc prints out the pc (program counter) so can then change the bits and such (might or might not be allowed to change the program counter value itself so try it out and see)
* rand() will not work, so try to generate a random number with gdb
  + can call libc functions in gdb so maybe can call libm functions as well?
* For a simple function (add 2 numbers)
  + Change instruction
  + Change instruction pointer (it’s just a register so change it to another register)
  + Change register addresses

Global a = 0;

Int b, c =1;

main()

a = b+c;

change the instruction

set breakpoint to a = b+c

p $pc b

$pc = $pc & (mask)

change register values:

$eax or $elx

change change $eax to 0 or 99 or add mask to it

Do first for simple function and if works, do for bangbangcontroller

Don’t have to meet on the 3rd, but send progress notes