CSE2312-002/003, Fall 2014, Homework 5: QEMU Installation

Due Date via Blackboard: October 16, 2014 (at 2:00pm for 002, 3:30pm for 003)

This homework is to introduce you to the Quick Emulator (QEMU), ARM assembly programming, assembling ARM programs, and running ARM programs.

This homework consists of two problems, A and B.

Problem A: Take a screenshot of a running QEMU example output, and

Problem B: Take a screenshot of the ARM registers using gdb,

then submit the screenshots together in one file via Blackboard (so you don't have to print the images).

Your name must be visible on the screenshots (type it somewhere visibly if your username doesn't display). Example screenshots are shown below. If your name is not visible, ***you will not receive credit***. If the output of the program is not visible, ***you will not receive credit***.

If you want to run QEMU on the laboratory computers, they are available in the following rooms:

ERB 124, ERB 125, and ERB 132

That said, we encourage you to run QEMU and ARM on your own computer, and the following tutorial steps 0 through 3 will show you how to do that. If you run QEMU in one of the labs, skip to step 4, which shows you how to execute an example program.

Lab Computer Instructions

Login to the laboratory computer with your netid and password. Launch the Virtual Box virtual machine and select the cse2312 image. You will be automatically logged into an Ubuntu virtual machine. Open a terminal window and start the next set of instructions at step 4.

QEMU Installation and Execution Instructions

Step 0.

These installation instructions are for ***32-bit Ubuntu (we've tested 12.04 LTS)*** and may need to be modified for other distributions (including 64-bit Ubuntu). ***We will not support any distributions other than Ubuntu 32-bit, so if you do not use the supported distribution, you must solve any installation problems on your own***. If you want to run Ubuntu as a virtual machine on your computer, check out:

- VMWare Player (PC, free): https://my.vmware.com/web/vmware/free#desktop_end_user_computing/vmware_player/6_0
- Virtual Box (PC, free): http://www.oracle.com/technetwork/server-storage/virtualbox/downloads/index.html
- Parallels (Mac): http://www.parallels.com/

You can download an ISO for Ubuntu for free from here: http://www.ubuntu.com/download/desktop

Once you have Ubuntu running, execute the following sequence of commands from a terminal command line. ***Be careful about line breaks in the instructions below***. These instructions available as a text file (README.txt) which will not have any potential line break problems.

Step 1. Install the QEMU system.

```
sudo apt-get install qemu qemu-system qemu-user qemu-utils
```

Step 1 (optional, if running ***64-bit*** Ubuntu may be necessary):

```
sudo apt-get install ia32-libs
```

Step 2. Install the GCC compiler tools for ARM processors. We need a special version of GCC since we are not compiling for an x86/x86-64 architecture. This is known as ***cross-compilation***.

```
wget https://launchpad.net/gcc-arm-embedded/4.7/4.7-2013-q3-
update/+download/gcc-arm-none-eabi-4_7-2013q3-20130916-
linux.tar.bz2

tar xjvf gcc-arm-none-eabi-4_7-2013q3-20130916-linux.tar.bz2

sudo mv gcc-arm-none-eabi-4_7-2013q3 /opt/ARM

echo "PATH=$PATH:/opt/ARM/bin" >> /home/"$(whoami)"/.bashrc

source /home/"$(whoami)"/.bashrc

arm-none-eabi-gcc --version
```

Step 3. Download and assemble (like compile) a test application. You should look at the files extracted from hw05.tar, as it contains a sample makefile (filename: Makefile), a sample ARM assembly program (filename: hw05.s), and some other necessary files for specifying where in memory the program will be located.

```
wget http://www.taylortjohnson.com/class/cse2312/f14/hw/hw05.tar

tar xvf hw05.tar

cd hw05

make
```

You should also look at each of the downloaded files, such as the Makefile, hw05.s, and hw05_memmap to see what they look like and start to understand the assembly process. At this point in the course, you should basically be able to start understanding these. You can see help for the commands used in the Makefile (like the arm-none-eabi-as assembler call) by typing, e.g. (and similarly for the other commands):

```
arm-none-eabi-as –help
```

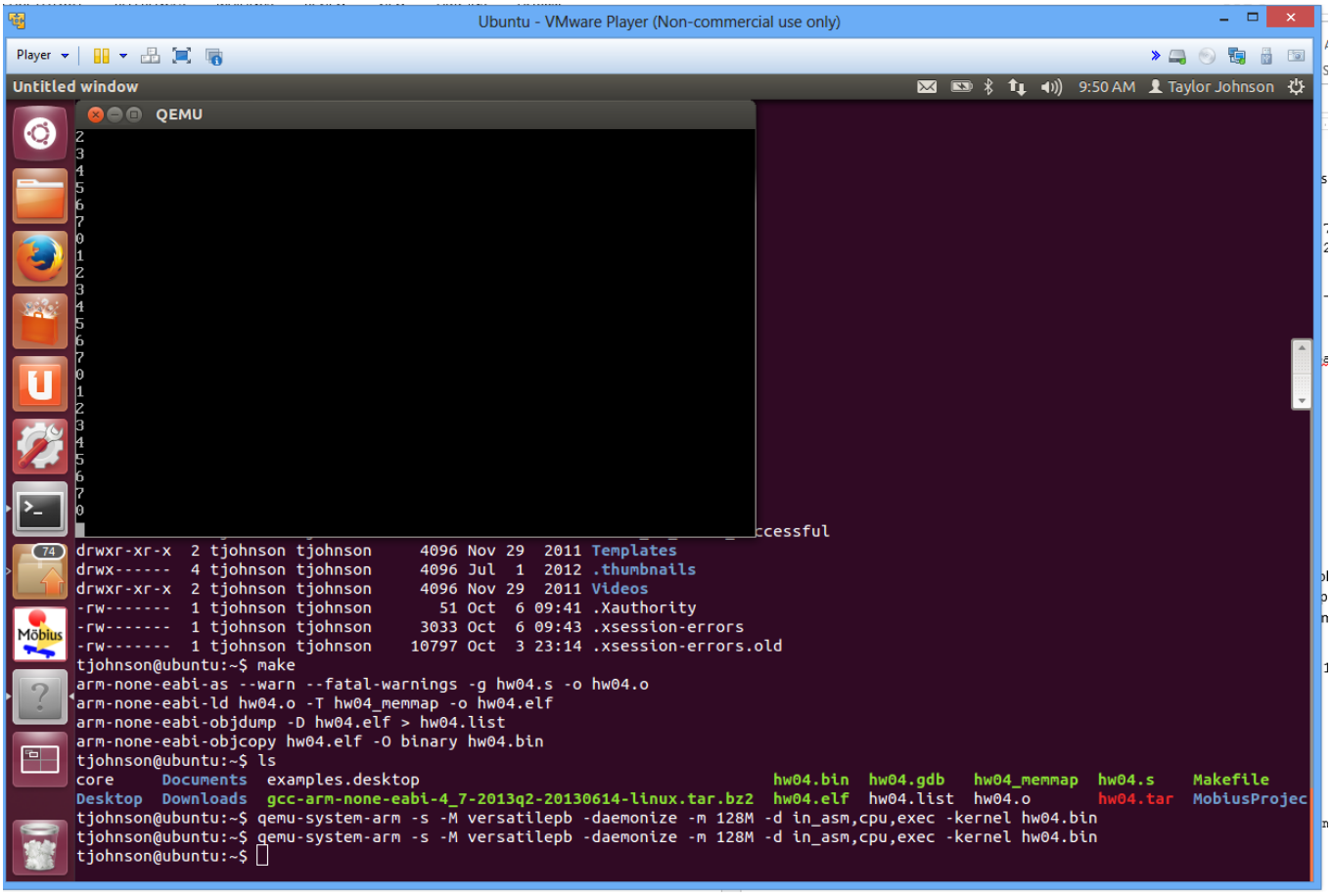
Step 4. Execute QEMU on the test application.

```
qemu-system-arm -s -M versatilepb -daemonize -m 128M -d
in_asm,cpu,exec -kernel hw05.bin
```

This should open a QEMU console window. Once started, press ctrl+alt+3 to change to QEMU's UART (serial) display output, it should be printing 0 through 7 repeatedly.

What this is doing is calling a program named "qemu-system-arm", which is the QEMU virtual machine, and using the binary (machine language) program from the file hw05.bin following the –kernel flag. This is in effect the "loading" part of linking and loading we've discussed in class. The flags –M versatilepb specifies a specific ARM board to use (which has additional hardware, like RAM, serial ports, etc., and is not just an ARM processor in isolation, it really represents a full virtual computer), the –daemonize starts the task as a daemon process (so we can reuse the console for other commands), the –m 128M specifies to use 128 MB RAM, the –d and flags allow for debugging.

Problem A. Create a screenshot of the QEMU output console window. The following is an example screenshot of what you should submit for the QEMU setup problem. Note that my name is visible on the upper right and the QEMU output window is displaying the sample application's output (printing 0 through 7 repeatedly).



Step 5. Execute the test program using QEMU and try out the GNU debugger (gdb) after installing it.

```
sudo apt-get install gdb-multiarch

qemu-system-arm -s -M versatilepb -daemonize -m 128M -d
in_asm,cpu,exec -kernel hw05.bin


gdb-multiarch
```

You can use GDB to look at memory values of particular addresses, register values, etc. Once you execute gdb-multiarch from the command line, you will be inside the gdb own console window (i.e., shell). Type the following:

```
target remote :1234

set architecture arm

symbol-file hw05.elf
```

The first line connects to the QEMU debugging port (1234), the next line sets the architecture of the process being debugged to ARM, and the third line sets the symbol file to be hw05.elf (which has the symbol table and lets you use, e.g., label names when debugging).

Your program starts paused. Type:

```
break _start

break loop
```

These commands add break points to the addresses at labels _start and iloop.

Next, type:

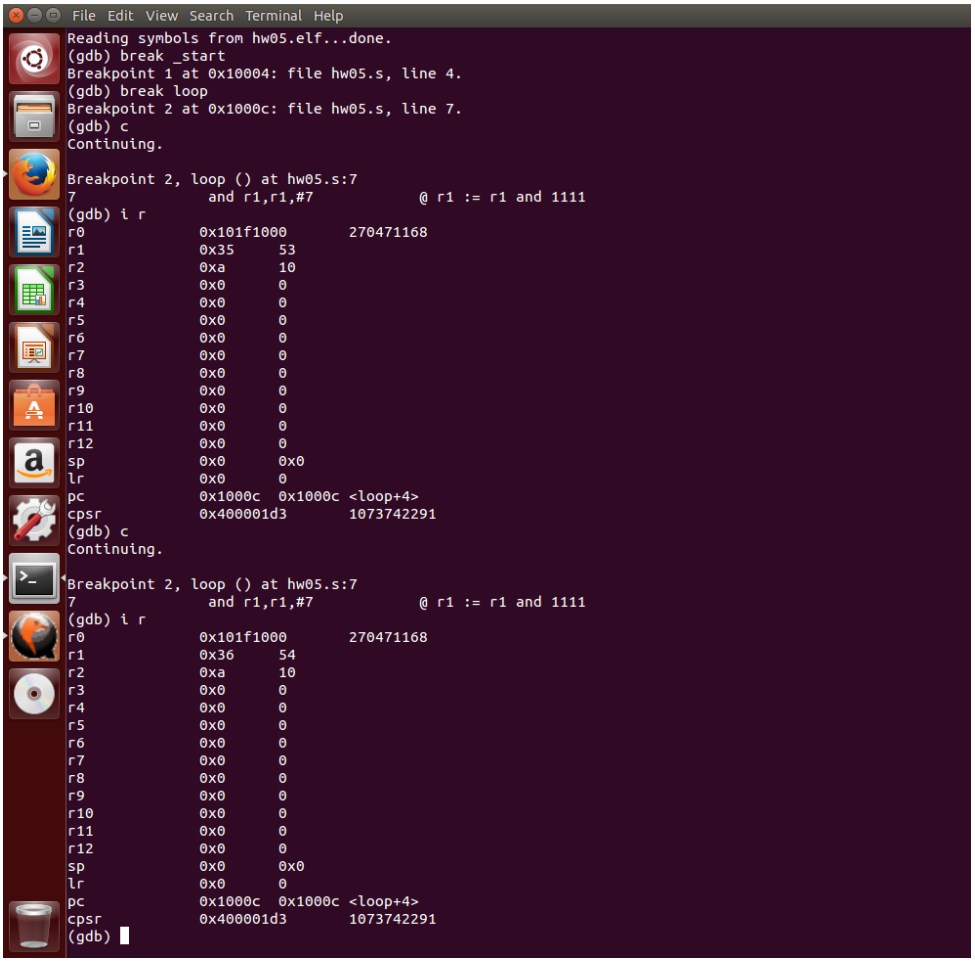
```
c

i r
```

Problem B: Create a screenshot of the GDB output, like what is shown in the next screenshot.

This causes the execution to resume c (for continue) and i r (for info registers) displays all the register values. Type c, then i r again to see updated register values computed until the next breakpoint is hit.

We will learn more about gdb later in the course. For more details for now, you can see a tutorial here:

GDB Tutorial: http://vlm1.uta.edu/~athitsos/courses/cse2312_summer2014/resources/gdb.html



Step 6. Here are some more details and other references. You should start to learn how to use gdb. We need to use gdb-multiarch since we're doing a cross-compilation (and cross-execution) from the x86/x86-64 Ubuntu to ARM on QEMU, as the normal gdb command will just know how to interpret x86/x86-64 machine language (you should understand why based on what we've seen in class with regard to machine language instructions).

- Debugger References
  - GDB Setup for QEMU: http://www.droid-developers.org/wiki/QEMU
  - GDB on QEMU: http://www.cs.utexas.edu/~dahlin/Classes/439/ref/qemu-gdb-reference.html
  - QEMU Monitor Commands: http://wiki.qemu.org/download/qemu-doc.html#pcsys_005fmonitor
- Other QEMU/ARM setup tutorials
  - ARM Toolchain: http://eliaselectronics.com/stm32f4-tutorials/setting-up-the-stm32f4-arm-development-toolchain/
  - QEMU Setup: http://www.contrib.andrew.cmu.edu/~acrichto/qemu.html
- GCC ARM Tools: https://launchpad.net/gcc-arm-embedded
- Make missing separator error: http://www.delorie.com/djgpp/v2faq/faq22_17.html

- ARM Hello World Program: http://blogs.arm.com/software-enablement/139-hello-world-in-assembly/
- ARM Assembly tutorial: http://www.coranac.com/tonc/text/asm.htm
- Other classes that are using QEMU
    - http://www.cs.sunysb.edu/~prade/Teaching/Spring13/lab1.html
    - http://www.cs.stonybrook.edu/~porter/courses/cse506/f11/lab1.html