CPSC 313: Computer Hardware and Operating Systems
Assignment #1, due Saturday January 23 at 11:59PM.

1 Objectives

NOTE: Late penalty of 33.333% per day. Late assignments not accepted after 2 days.

This assignment may be done in pairs or alone. Groups of 3 or more are not permitted. You are strongly encouraged to work with a partner.

After completing this assignment, you should be able to

- read programs written using the x86-64 ISA, and explain what each instruction does,
- identify the correspondences between assembly and C code fragments,
- translate programs written in C or x86-64 assembly language into Y86-64 assembly language.

2 Introduction

In CPSC 213, you learned how a C compiler translates constructs into assembly language using SM213, a fictional ISA developed specifically for that course. The goals of this lab are to allow you to refresh your understanding of assembly language, deepen your knowledge of the IA32/x86-64 instruction set architectures (which you may have touched upon in CPSC 213, but without going into any depth), and help you become more familiar with the Y86-64 subset of the IA32 ISA that we will be using in class for the first part of the course.

3 Setting up your team, the Simulator and getting your repo

We will use version control (in the guise of Atlassian Stash, which uses git) to distribute the files that you will be working on. It will also be the way that you handin your assignment. The only exceptions will be two files simulator.jar and antlr-3.4-complete.jar that you can download from the Assignment page on Connect.

As indicated above you will be allowed to do this assignment either working by yourself, or with a partner; teams of three or more are not allowed. Before you can access the git repo with your copy of the assignment, you will need to register your team, or yourself if you aren't working with a partner, using the *Register your team link* available on the Assignment page of Connect. You must do this even if you are working by yourself in this case you leave the Partner field blank. When you are working with a partner, both of you must register your team before the assignment repo will be created. Upon creation, the URI of the repo will be displayed at the bottom of the form; it will be something like https://stash.ugrad.cs.ubc.ca:8443/git/CPSC313_2015W2/ass5_r2d2_c3p0.git

(make sure it has https: at that front and isn't the ssh uri)

To create a local copy of the repo using Eclipse, go to File –¿ Import and select Projects from Git under Git. You can see a step by step description of the procedure by checking out the Youtube video link on the assignment page on Connect. You will need to fill in the URI of your repo in the box instead of whatever URI the speaker used. If you are doing this on a machine where your login ID is not your undergraduate ID (say, on your laptop or a computer at home), replace:

```
https://stash.ugrad...
```

with

```
https://LOGIN@stash.ugrad...
```

where LOGIN is your undergraduate login ID. Depending upon the version of Eclipse you are using the options in the above video might be a bit different especially when you get to the "Select a wizard to use for importing projects".At this point follow the following steps:

- Select the "Use the New Project Wizard" since you are not importing an Eclipse project. Make sure to to note the name of the working directory as you will need it later.
- Click Finish
- On the Select a Wizard screen that now now appears make sure that Java–> Java Project is selected.
- Click Next
- Give the project a name
- Unclick "Use default location"
- In the location option browse to the name of the working directory noted above.
- Click Finish.

Once your Java project has been created, you will need to add **simulator.jar** and **antlr-3.4-complete.jar** to it. Right-click on the name of your new project, select **Properties**, then **Java Build Path** and **Libraries**. Add both **.jar** files as External JARs. Finally, click on the Run menu item and then select **Java Application**. The main class you want is called SimpleMachine$Y86SeqStudent.

If you are not using Eclipse, which is fine, you will need to use either a command line or graphical version of git. Fortunately, the department Linux machines all have a command line version of git installed. Before creating your copy of the repo you need to know its URI which was displayed when all partners completed registration. You can also determine the repo name by logging into Stash.

Determing your repo URI via Stash

When you login to the Stash server, you are presented with a list of projects you have access to. With respect to this course, you should see a project called *"CS313_2015W2_201"* and its associated key *CS313_2015W2_201.* If you click on that project you will be taken to a list of repositories. A repository is essentially a collection of code objects that form some sort of cohesive unit. In this case it is assignment 1. The repository name will be `cs313w2_a1_` followed by the undergrad ID of you and your partner or just your ID if you are working along. (Working alone is discouraged.) You have permission to copy from and write to this repository. You are expected to keep the repository in sync with your code so that you can track the changes you make and to simplify recovery if you accidentally delete files or code. In addition this helps simplify the process of keeping copies of the code you are working on in sync across the various machines you and your partner might be using. (e.g. a department machine, your laptop, a home PC etc.) You can also use the tools within Stash to compare different versions of files to see what changes have been made.

Once in your repo, click on **...** towards the top left. One of the menu items will be clone. Click on clone, and then select http from the pulldown menu if it shows ssh. Copy the URI shown in the box so that it can be pasted to the appropriate git command. The URI will look something like:

https://r2d2@stash.ugrad.cs.ubc.ca:8443/scm/cs313_2015w2_201/cs313w2_a1_r2d2_c3p0.git

3.0.1 Working with git

If you are not using git through Eclipse then, to retrieve the assignment code, you will need to have a version of git installed on each machine where you want a copy of the code. The department has installed a command line version of `git` on the undergrad Linux machines. `git` is a complicated and powerful system, but we will try to keep things simple. If you need help installing git on your own machine, or want to learn more, a good starting point is book you can access at *http://git-scm.com/doc* . The following instructions should help you get going and assume you are using a department Linux machine and any URIs correspond to the ones provided when you registered your partner or determined via Stash.

```
git clone https://r2d2@stash.ugrad...c3p0.git
```

will create a copy of your project in the current directory.

As you work on your code you will want to periodically "push" changes to the Stash repository. It is highly recommended that you do this frequently. You might want to push these changes every time you stop working on the code and when you get to important implementation milestones. To start developing an understanding of how to use git read chapter 2 of the above book. To get you started here are some commands that may be useful.

- `git add fname` - to stage a file with the name `fname` for committing. You need to do this for every file you change or to files you want to start tracking in Stash.

- `git commit -a` - to commit the files. This is a local operation and does not result in any updates to the files in Stash, but it is an important step. (pay attention to the list of files shown to make sure you are not leaving out files whose changes should be committed.)
- `git push` to push all the commits to the Stash server.
- `git pull` This pulls all the changes from the Stash server to your local copy and merges them. This will be very useful if you are developing on more than one machine. First you would push the commits from one machine and then pull the changes from the other.

You will probably want to use tags to track important checkpoints and milestones in the progress of your assignment. (See section 2.6 of the book) Don't forget to push the tags to the Stash server.

If you have never used git on the machine you are going to use to track the code, you might first need to configure git. To do this follow the directions in chapter 1.5 of the above reference. The only part you really need to do is the section called *Your Identity*.

All of the work for this assignment will be done in the code directory of the repo you have cloned.

3.1 Repo Files

The repo for assignment 1 contains the following files:

- a directory **examples** that contains three examples sum_64.s, max_64.s and sort_64.s. Note that the max_64.s file doesn't work but can be easily fixed so you can use it to practice with.
- a directory **heapsort-C** that contains a C implementation of heapsort split between two source files, as well as a Makefile.
- a directory **heapsort-y86**that contains an incomplete Y86-64 implementation of heapsort.

You can use the sample program to re-familiarize yourself with the running of the simulator. Important do not use a simulator from 213 or any previous offering of 313. None of them will work.

4 Understanding x86-64 Assembly language

Your first task is to comment the the IA64 assembly version of the heapsort function to be produced by compiling the file `heapsort.c`. It implements a function that returns the position of the largest element of a subset of an integer array. Use the command `gcc -O2 -S heapsort.c` (do not forget any of the arguments) to get the gcc compiler to produce assembly code for this procedure. You must use a department machine (i.e. your own home machine, laptop etc. is not to be used. You can do it as an academic exercise to see the differences produced by different compiler and operating system combinations but what you commit must be from a department machine. All of the machines in ICCS 005 meet these requirements and you can ssh login to them from remote.ugrad.cs.ubc.ca using linX.ugrad.cs.ubc.ca where X is 01, or 03, or ... or 25.

```
void heapsort(int last)
 {
int i;
    heapify_array(last);
    for (i = last; i >= 0; i--) {
            heap[i] = extract_max(i);
}
}
```

(Compile the version of the file in the heapsort-C directory of the repo.) Once you have generated the .s file you should add and commit this starting point to your repo. Now, comment every line in the assembly code to explain what it does. Be sure to indicate the correspondence between C and Assembly code, i.e., which assembly instructions implement which lines of the C program. Ignore the setup and the tear-down code, that is, only comment the lines after

```
.cfi_startproc
```

and before

```
.cfi_endproc
```

and ignore all assembler directives such as `.cfi_def_cfa_offset` or `.p2align`.

You will want to read the textbook starting at section 3.4. Procedures and argument passing are explained starting in section 3.7

Finally a repeat of the word of caution about using your own machine. Different compiler/OS combinations can result in different code being produced by the compiler. Even a slight change between compiler versions can result in significantly different code. As a result, for consistency and ease of marking, the output being commented, and hence marked, must match the output as produced by the compilers on the department's undergraduate Linux machines.

Don't forget to commit the changes.

5 The simulator

If you followed the assignment directions your simulator should already be setup. If you like, you can use the sample assembly program provided sum_64.s (or any of the other pieces of code in the examples directory of the repo) to re-familiarize yourself with the execution of the simulator. Most of you have used the simulator before in CPSC 213, but if you are having problems do not hesitate to post a question on piazza. **IMPORTANT:** do not use an older version of the simulator from CPSC 213, or an earlier offering of CPSC 313 if you are unlucky enough to be retaking the course, or from this terms offering of CPSC 213.

6 Writing Y86 Assembly language programs

Your next task is to translate the code from the file heapsort.c into Y86 assembly language. One way to achieve this is to start from the file heapsort.s you worked with in Section 4, and translate it more or less line by line. Alternatively, you can write the Y86 program from scratch if you prefer. Your implementation must follow the calling conventions indicated at the top of file heapsort-y86/heapsort-student.s. Failure to follow these calling conventions will result in a function that looks like it should work, but fails once it is combined with the code we wrote.

Do not forget that the Xth element of array heap is at address $address(heap) + 8x$, not at address $address(heap) + x$. Your code must be well-commented. We have provided source code for the contents of file heapsort-main.c in file heapsort-y86/heapsort-student.s. You are to to test your solution by inserting your code for heapsort at the location indicated by

```
###
### THIS PART TO BE COMPLETED BY THE STUDENT.
###
```

in file *heapsort-y86/heapsort-student.s*, and then running it through the simulator to verify that it sorts the array heap correctly. Make sure to test your code with data other than that provided. You might want to test with things like duplicate values and special edge cases where the largest or smallest value are either the first of last elements in the array.

7 Deliverables

You are to use stash to submit your assignments by simply making sure that your final version is pushed to the stash repo. Of course you should be regularly pushing your code to track the changes you are making. It is good idea to make sure that every time you get something working you commit and push it.

For your final commit you will want to make sure that you commit the appropriate versions of the following files:

- The commented version of **heapsort.s**

- A copy of the file **heapsort-student.s** that includes your Y86 implementation of the heapsort function (with comments!).
- The filled in version of coverpage.txt. Follow the directions in the coverpage.txt file on how to fill-in and submit the file.

You are to commit and push your code regularly. Your grade will be based on your latest pushed version, and any late penalty, if applicable, will be based on the date that you committed that version. Any commits pushed more than 48 hours after the submission deadline may be ignored. The marking scheme will be broken down roughly as follows: 10 marks for commenting the heapsort.s file generated by the gcc compiler, 14 marks for the Y86 version of the heapsort.

When you have pushed your final version of the assignment you should sign-in to stash to verify that you have committed and pushed the versions of the files you want marked. You might want to clone a new version of your repo and test it to make sure everything is as you expect.