2014 ICS Lab4: Y86 Simulator

Hand out: Thu. May 8
Final Deadline: Thu. May 22 15:59 No Extension!

1. Introduction

The purpose of this lab is to have a deep insight into Y86 instruction function. You will do this by implementing a simulator which simulates the behavior of a machine running Y86 binary codes. A skeleton code of simulator is already prepared and you are required to make it support all Y86 instructions step by step. (You also could do it from scratch, we only check final output)

2. Logistics

You should work **individually** in solving the problems in this lab. Any clarifications and revisions to the lab will be posted on the News webpage of ICS course website (http://ipads.se.sjtu.edu.cn/courses/ics).

3. Materials

You should use *svn* tools to get lab4 from server like lab1. The URL of svn repository is still "svn://ipads.se.sjtu.edu.cn/ics-se13/[account]". You can see a directory named "lab4" under your path. The lab4 directory contains 3 sub-directories and 5 files:

```
y86-base  y86-ins-bin  y86-app-bin

Makefile  y86sim.h  y86sim.c  yat.c  yat
```

The only 2 files you need to modify and submission is **y86sim.c** and **y86sim.h** (don't add new files or directory to svn, TA would only check out the above 2 files for grading)**.**

The executable file **yat** allows you to evaluate the functional correctness of your implementation. You could use it to evaluate your implementation by yourself. The usage of **yat** will be introduced later.

The sub-directory **y86-base** contains a correct implementation of simulator, named as **y86sim-base**, and a correct implementation of assembler, named as **y86asm-base**, and all the **.ys** files from which the **.bin** files are generated. It is

used as the baseline by **yat** to testify your implementation. Your goal is to make your y86 simulator output equal to it.

The sub-directory **y86-ins-bin** contains 32 test .bin files. Each one corresponds to a kind of y86 **instruction**. **Attention** that some of the test .bin files in **y86-ins-bin** directory are **incorrect.** Your simulator should output the error information as well as the output of y86sim-base.

The sub-directory **y86-app-bin** contains 20 test .bin files. Each one corresponds to a simple assembly **program**.

4. Implement Y86 Simulator

Your job is to implement an Y86 simulator. A skeleton code of implementation is provided in y86sim.c. You can either implement functions and procedures in this skeleton or rewrite the whole program from scratch, since the evaluation is only based on the output generated by assembler.

If you choose to implement the simulator based on skeleton codes, we recommend you to go through the y86sim files first.

During the process of implementation, you can testify your implementation of any instruction at any time you want by using the following command:

```
$./yat -s <ins_name> (e.g. ./yat -s rrmovl)
```

(More functions of yat will be introduced later)

This command will set specific <ins_name>.bin file in y86-ins-bin directory as input file of your simulator and check the **.sim** file generated by your simulator by comparing to files generated by standard Y86 simulator(y86sim-base). You could see the result (Pass or Fail) and score for specific instruction.

5. Overview of y86sim.c

This program reads .bin file and parse binary codes according to the Y86 instruction structure. After an instruction is distinguished and confirmed, the program will change the values of registers and memory. Some key data structure are defined in y86sim.h and used in y86sim.c

The key function of simulator is **`stat_t nexti(y86sim_t *sim)`**. You need to fill in switch case blocks to implement the simulate function. And you also need to implement other support functions such as **`compute_alu`**, **`compute_cc`** and so on. Hints of implementation are also given in y86sim.c as comments. It is recommended to implement functions according to comments.

Attention, y86sim could set a specific number of running steps (one step

means distinguish and execute one instruction). If your simulator gets a wrong output, you can set a specific number of steps to check the program. You can use following command:

```
$./yat -s <instruction> [max_steps](e.g. ./yat -s call 1)
```

It means the program will stop after 1 step and print out the result. We **only** check the **final output** (without setting max steps), but we recommend you to use this option to debug the program.

6. Test and Evaluation

Your implementation will be evaluated by using **yat**. You can evaluate your implementation by yourself.

If the binary version of **yat** not work on your platform (e.g., MacOS), you could type "**make yat**" in lab5 directory to generate it from source code (yat.c). The usage of **yat** is as follows:

```
yat   -c <name>  [max_steps]  get the correct status of registers and
                              memory
                              e.g. yat -c prog9 4
yat   -s <ins_name>  [max_steps] test .bin file of single instruction
                              in y86-ins-bin directory
                              e.g. yat -s rrmovl
yat   -S                      test all instructions
yat   -a <app_name> [max_steps] test single application in y86-app-bin
                              directory.
                              e.g. yat -a asum 1
yat   -A                      test all applications
yat   -F                      test all instructions and applications,
                              and print out the final score.
yat   -h                      print this message
```

<name> should be the file name of one file in y86-base directory (.ys suffix is not included). **<ins_name>** should be the file name of one file in y86-ins-bin directory (.bin suffix is not included). **<app_name>** should be the file name of one file in y86-app-bin directory (.bin suffix is not included).

The **yat** program will compare the output generated by your implementation to those of y86asm-base. If difference is found, you will see information indicating correct result and your result.

32 instruction tests and 20 program tests are provided in corresponding directories. **Each instruction test values 1 point and each program test values 2 points. There are 72 points totally (32 * 1 + 20 * 2)**.

You can also write your own test files and put them in directories above. In this case you can only use these files with "-s" or "-a" options. **Test files written by yourself will not be evaluated if you use "-A" or "-F" options.**

The final score of your implementation is given by command "`./yat -F`".

P.S.
If you modify y86sim.c and the output of yat does not change as you expect, you can try to type "make clean", and then execute yat again.

7. Hand-In

You only need to commit the y86sim.c and y86sim.h files to svn server if you modify them. We strongly recommend you to multiple commit your code to svn during implementation.