2019 ICS Lab5: Y86-64 Assembler

Hand out: Fri. July. 12
Final Deadline: Fri. July. 26 15:59 No Extension!

1. Introduction

The purpose of this lab is to have a deep insight into Y86-64 instruction architecture. You will do this by implementing an assembler which transforms Y86-64 assembly codes into Y86-64 binary codes. A skeleton code of assembler is already prepared and you are required to make it support all Y86-64 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 lab5 from server like lab4. The URL of svn repository is still "svn://ipads.se.sjtu.edu.cn/ics-se17/[account]". You can see a directory named "lab5" under your path. The lab5 directory contains 4 sub-directories and 5 files:

```
Y64-base  y64-ins   y64-err   y64-app

Makefile  y64asm.h  y64asm.c  yat.c  yat
```

The only 2 files you need to modify and submission is **y64asm.c** and **y64asm.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 **y64-base** contains a version of correct implementation of assembler, named as **y64asm-base**, and all test files. It is used as a benchmark by **yat** to testify your implementation. Your goal is to make your y86-64 assembler output equal to it.

The sub-directory **y64-ins** contains 33 test files. Each one corresponds to a

kind of y86-64 **instruction**.

The sub-directory **y64-err** contains 8 test files. Each one corresponds to a kind of **error** we tested.

The sub-directory **y64-app** contains 20 test files. Each one is an simple assembly **program**.

4. Implement Y86-64 Assembler

Your job is to implement an Y86-64 assembler. A skeleton code of implementation is provided in y64asm.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 files (.yo and .bin files) generated by assembler.

If you choose to implement the assembler based on skeleton codes, we recommend you to go through the y64asm.c 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 <instruction> (e.g. ./yat -s rrmovq)
```

This command will set specific <incstuction>.ys file in y64-ins directory as input file of your assembler and check the .yo file and .bin file generated by your assembler by comparing to files generated by standard Y86-64 assembler. You could see the result (Pass or Fail) and score for specific instruction.

5. Overview of y64asm.c

This program reads .ys file and parse assembly codes line by line. After the first scan of assembly codes, relocate function will be invoked to fill in addresses that are needed in jump instructions according to corresponding labels. Some key data structure are defined in y64asm.h and used in y64asm.c. Contents of .yo and .bin files that are maintained in specific data structures are also generated in the process of parsing.

The key function of assembler is **`parse_line`**. You need to fill in switch case blocks to implement the parsing of each instruction. Other functions will be invoked during the process of parsing instructions, such as **`parse_reg`**, **`parse_mem`**, **`parse_label`** and so on and you need to implement them to parse instructions correctly as well.

Hints of implementation are also given in y64asm.c as comments. It is recommended to implement functions according to comments.

6. Test and Evaluation

Your implementation will be evaluated 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 "rm yat" and then "make yat" in lab5 directory to generate it from source code (yat.c). The usage of **yat** is as follows:

```
yat -h                  Show help information of yat
yat -s <instruction>    Test correctness of single instruction in
                        y64-ins directory
yat -s <error>          Test correctness of processing with
                        specific error in y64-err directory
yat -S                  Test correctness of processing all
                        instructions and errors.
yat -a <program>        Test correctness of processing single .ys
                        file in y64-app directory
yat -A                  Test correctness of processing all .ys
                        files provided in y64-app directory
yat -F                  Test correctness of processing all
                        instructions, error types and programs.
```

<instruction> should be the file name of one file in y64-ins directory (.ys suffix is not included). <error> should be the file name of one file in y64-err directory and <program> should be the file name of one file in y64-app directory.

The **yat** program will compare .yo files and .bin files generated by your implementation to those of y64asm-base. If difference is found, you will see information indicating correct result and your result.

E.g. Information below shows evaluation result of ***pushq*** instruction, which indicates difference between correct results and wrong results. Binary files of them are also different.

```
$./yat -s pushq
[ Testing instruction: pushq ]
<   0x000: a05f          |          pushq %rbp
---
>   0x000: a060          |          pushq %rbp
Binary files pushl.bin.base and pushl.bin differ
Scores for instruction:       0.00/ 1.00
```

33 instruction tests, 8 error tests and 20 program tests are provided in corresponding directories. **Each instruction test and error test values 1 point and each program test values 2 points. There are 81 points**

totally (33 * 1 + 8 * 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 y64asm.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 y64asm.c and y64asm.h files to svn server if you modify them. We strongly recommend you to multiple commit your code to svn during implementation.