CS 349, Summer 2002
Architecture Lab (Part A): Writing and Simulating Y86 Code
Assigned: Tue May 21, Due: Monday May 27, 11:59PM

Dave O'Hallaron (`droh@cs.cmu.edu`) is the lead person for this assignment.

1 Introduction

Part A of the Architecture Lab is a warm up exercise. You will learn how to write simple Y86 programs, translate them into Y86 object files using YAS (the Y86 assembler), and execute the object files using YIS (the Y86 instruction simulator).

2 Logistics

- You will work on this lab alone. Any clarifications and revisions to the assignment will be posted on the course Web page.
- In the following, `CLASSDIR` refers to

  ```
  /afs/cs/academic/class/15349-s02
  ```

- Linux binaries for the YAS and YIS tools are available at

  ```
  CLASSDIR/sim/misc/{yas,yis}
  ```

  In order to avoid typing a lengthy pathname each time you run a tool, you will need need to add its pathname to your shell's search path. You can do easily by sourcing one of the following files:

  ```
  CLASSDIR/Y86path.{bash,csh,ksh,sh,tcsh}
  ```

  For example, if you are running the `tcsh` shell, then you would type:

  ```
  unix>  source Y86path.tcsh
  ```

3 Handout Instructions

All files you need are in the directory

```
CLASSDIR/archlaba
```

Start by copying the file `archlaba-handout.tar` from that directory to a (protected) directory in which you plan to do your work. Then give the command: `tar xvf archlaba.tar`. This will cause the following files to be unpacked into the directory: `README`, `Makefile`, `examples.c`, `archlaba.ps`, and `archlaba.pdf`.

You will be writing and turning in three Y86 programs: `sum.ys`, `rsum.ys`, and `copy.ys`. The required behavior of these programs is defined by the example C functions in `examples.c`. The `Makefile` contains rules for assembling, executing, and submitting your solutions. The `archlaba.{ps,pdf}` files contain the postscript and pdf versions of the Lab writeup, respectively.

4 Your Task

Your task is to write and simulate the following three Y86 programs. Be sure to put your name and Andrew ID in a comment at the beginning of each program.

`sum.ys`: Iteratively sum linked list elements

Write a Y86 program (`sum.ys`) that iteratively sums the elements of a linked list. Your program should consist of a main routine that invokes a Y86 function (`sum_list`) that is functionally equivalent to the C `sum_list` function in Figure 1. Test your program using the following three-element list:

```
# Sample linked list
.align 4
ele1:
        .long 0x00a
        .long ele2
ele2:
        .long 0x0b0
        .long ele3
ele3:
        .long 0xc00
        .long 0
```

`rsum.ys`: Recursively sum linked list elements

Write a recursive version of `sum.ys` (`rsum.ys`) that recursively sums the elements of a linked list.

```
/* linked list element */
typedef struct ELE {
    int val;
    struct ELE *next;
} *list_ptr;

/* sum_list - Sum the elements of a linked list */
int sum_list(list_ptr ls)
{
    int val = 0;
    while (ls) {
        val += ls->val;
        ls = ls->next;
    }
    return val;
}

/* rsum_list - Recursive version of sum_list */
int rsum_list(list_ptr ls)
{
    if (!ls)
        return 0;
    else {
        int val = ls->val;
        int rest = rsum_list(ls->next);
        return val + rest;
    }
}

/* copy_block - Copy src to dest and return xor checksum of src */
int copy_block(int *src, int *dest, int len)
{
    int result = 0;
    while (len > 0) {
        int val = *src++;
        *dest++ = val;
        result ^= val;
        len--;
    }
    return result;
}
```

Figure 1: **C versions of the Y86 solution functions.** See `CLASSDIR/archlaba/examples.c`

Your program should consist of a main routine that invokes a recursive Y86 function (`rsum_list`) that is functionally equivalent to the `rsum_list` function in Figure 1. Test your program using the same three-element list you used for testing `list.ys`.

`copy.ys`: Copy a source block to a destination block

Write a program (`copy.ys`) that copies a block of words from one part of memory to another (non-overlapping area) area of memory, computing the checksum (Xor) of all the words copied.

Your program should consist of a main routine that calls a Y86 function (`copy_block`) that is functionally equivalent to the `copy_block` function in Figure 1. Test your program using the following three-element source and destination blocks:

```
.align 4
# Source block
src:
        .long 0x00a
        .long 0x0b0
        .long 0xc00

# Destination block
dest:
        .long 0x111
        .long 0x222
        .long 0x333
```

5 Evaluation

This part of the lab is worth 30 points, 10 points for each Y86 solution program. Each solution program will be evaluated for correctness, including proper handling of the `%ebp` stack frame register and functional equivalence with the example C functions in `examples.c`.

The programs `sum.ys` and `rsum.ys` will be considered correct if their respective `sum_list` and `rsum_list` functions return the sum `0xcba` in register `%eax`.

The program `copy.ys` will be considered correct if its `copy_block` function returns the sum `0xcba` in register `%eax`, and copies the three words `0x00a`, `0x0b`, and `0xc` to the 12 contiguous memory locations beginning at address `dest`.

6 Hints

- You can let GCC do most of the work for you by compiling the three C functions with the options `-O2 -S`. The `-O2` compiles at optimization level 2, producing decent assembly code, as opposed to no optimization, which produces very bad code. The `-S` option tells the compiler to emit an IA32 assembly language file, which you can then convert fairly easily into Y86 code.

- Recall that the Y86 `andl S2, S1` instruction is equivalent to the IA32 `testl S2, S1` instruction.

7 Handin Instructions

- You will be handing in the following three Y86 files: `sum.ys`, `rsum.ys`, and `copy.ys`.
- Make sure you have included your name and Andrew ID in a comment at the top of each of your handin files.
- To handin your three Y86 files, type:

  ```
  make handin TEAM=teamname
  ```

  where `teamname` is your Andrew ID.
- After the handin, if you discover a mistake and want to submit a revised copy, type

  ```
  make handin TEAM=teamname VERSION=2
  ```

  Keep incrementing the version number with each submission.
- You can verify your handin by looking in

  ```
  CLASSDIR/archlaba/handin
  ```

  You have list and insert permissions in this directory, but no read or write permissions.