CS 429, Spring 2012
Y86 Assembly Exercises
Assigned: Feb 9, Due: Feb 23, 11:59PM

Tyler Smith (`tms@cs.utexas.edu`) is the lead person for this assignment.

1 Introduction

In this lab, you will transform three simple functions from C into Y86 assembly and test them against a simulator. The purpose of this is to give you practice with assembly level programming in general, and with the Y86 instruction set and tools in particular.

2 Logistics

You will work on this lab individually.

Any clarifications and revisions to the assignment will be posted on the course Web page.

3 Handout Instructions

You can get a copy of this handout and the assignment code from the course website. You should download the `asmlab-handout_429.tar` file.

1. Start by copying the file `archlab-handout.tar` to a directory in which you plan to do your work.
2. Then give the command: `tar xvf asmlab-handout_429.tar`.
   This will cause the following files to be unpacked into the directory:
   `README`, `Makefile`, `sim.tar`, `archlab.ps`, `archlab.pdf`, and `simguide.pdf`.
3. Next, give the command `tar xvf sim.tar`. This will create the directory `sim`, which contains your personal copy of the Y86 tools. You will be doing all of your work inside this directory.
4. Finally, change to the `sim` directory and build the Y86 tools:

```
unix>  cd sim
unix>  make clean; make
```

4 Part A

You will be working in directory `sim/misc` in this part.

Your task is to write and simulate the following three Y86 programs. The required behavior of these programs is defined by the example C functions in `examples.c`. Be sure to put your name and ID in a comment at the beginning of each program. You can test your programs by first assemblying them with the program YAS and then running them with the instruction set simulator YIS.

In all of your Y86 functions, you should follow the IA32 conventions for the structure of the stack frame and for register usage instructions, including saving and restoring any callee-save registers that you use.

`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 some code that sets up the stack structure, invokes a function, and then halts. In this case, the function should be Y86 code for a 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 Y86 program `rsum.ys` that recursively sums the elements of a linked list. This code should be similar to the code in `sum.ys`, except that it should use a function `rsum_list` that recursively sums a list of numbers, as shown with the C function `rsum_list` in Figure 1. Test your program using the same three-element list you used for testing `list.ys`.

```c
/* 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 `sim/misc/examples.c`

`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 code that sets up a stack frame, invokes a function `copy_block`, and then halts. The function should be functionally equivalent to the C function `copy_block` shown in Figure 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

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 stack and registers, as well as functional equivalence with the example C functions in `examples.c`.

The programs `sum.ys` and `rsum.ys` will be considered correct if the graders do not spot any errors in them, and their respective `sum_list` and `rsum_list` functions return the sum `0xcba` in register `%eax`.

The program `copy.ys` will be considered correct if the graders do not spot any errors in them, and the `copy_block` function returns the sum `0xcba` in register `%eax`, copies the three words `0x00a`, `0x0b`, and `0xc` to the 12 contiguous memory locations beginning at address `dest`, and does not corrupt other memory locations.

6 Handin Instructions

To submit your code, use the following command:

```
turnin --submit tms asmlab sum.ys rsum.ys copy.ys
```

Make sure you have included your name and ID in a comment at the top of each of your handin files.