cs281: Introduction to Computer Systems

HW6 – archlab

Assigned: November 8, Due: Mon., November 14, 11:59pm

1 Introduction

In this lab, you will continue to learn about the design and implementation of a Y86 central processing unit (CPU).

The lab is organized into two parts. In Part A you will write some basic Y86 programs and become familiar with the Y86 tools, learning in more detail the Y86 instruction set architecture (ISA). In part B (and in conjunction with Lab 11), you will implement the control unit to complete the given datapath for the Y86.

2 Logistics

You will work on Part A alone, but will work on Part B in teams of at most two.

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

Although all you require for Part A is the specification of the three assembly language programs, I have included on the course's user directory (`~cs281`) the textbook's distribution giving you the source for the Y86 assembler and simulator, along with many example assembly language source files (with the `.ys` extension), and also the HCL implementation of the Y86 CPU. If you want to peruse these resources, you should be able to simply copy the contents of the `~cs281/sim` directory to a directory of your choosing under your home directory in the Olin 219 Linux lab. Also in the `~cs281` directory is the program `yo2banks`, along with its source, which takes as input a

Relative to the `sim` directory, you will find the following:

- `misc` subdirectory – In this directory, you will find the code and created executables for yas and yis and also the `examples.c` source file given in the description below.
- `y86-code` subdirectory – This directory contains many example Y86 assembly language source files and their assembled counterparts.
- `seq` subdirectory – This directory contains the HCL defined SEQ implementation of the Y86 architecture.
- `pipe` subdirectory – This directory contains the HCL defined PIPE implementation of the Y86 architecture.
- `ptest` subdirectory – In this directory, the authors include a set of Perl scripts for testing the HCL implementations of the Y86 architectures.

Since we are building an alternate CPU using Logisim, the last three directories will have limited use for you, but I include them for those that might want to experiment and investigate this alternative method of defining a CPU.

3 Part A

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.

`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
```

```
/* 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`

`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.

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
```

4 Evaluation

Part A

Part A 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`.

5 Handin Instructions

- You will be handing in three files:
  - Part A: `sum.ys`, `rsum.ys`, and `copy.ys`.
- Make sure you have included your name at the top of each of your handin files.
- Simply email me with your three text files listed above.