CS 429, Fall, 2014, Laboratory 2(A) and 2(B)
Writing Programs for the Y86

Assigned: Wednesday, October 8, 2014
Part 1 Due: Wednesday, November 5, 2014, 9:00 am
Part 2 Due: Wednesday, November 19, 2014, 9:00 am

1 Introduction

In this lab, you will learn about the class Y86 ISA specification. When you have completed the lab, you will have a better appreciation for the interactions between code and hardware that affect the performance of your programs.

The lab is organized into two parts, each with its own handin. In Part A you will write some simple Y86 programs and become familiar with the Y86 tools. In Part B, you will use the class simulator to optimize a Y86 program.

2 Logistics

You are to work on this lab alone. Any clarifications and revisions to the assignment will be posted on the course Web page.

3 Handout Instructions

You will find the file assembler/simulator referenced on the homework page of the class website. When you want to run the assembler/simulator, you should execute this file directly. Please don't copy this executable to your home directory – it is about 4.5 GBytes in size. In addition, there is a file with some examples – this file may downloaded and used as you see fit.

4 Evaluation

The lab is worth 100 points: 50 points for Part A, 50 points for Part B.

5 Part A

Your task is to write and simulate the following two 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 CS UserID in a comment at the beginning of any submitted file. The programs `sum.lisp` and `rsum.lisp` will be considered correct if their respective `sum_list` and `rsum_list` functions return with 36 in register `%eax` – and no, just writing a program that puts 36 into register `%eax` isn't sufficient. Remember, your program must follow caller/callee register save conventions.

Iteratively sum linked list elements

Write a Y86 program (`sum.lisp`) 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
    (dword 7)
    (dword ele2)

    ele2
    (dword 12)
    (dword ele3)

    ele3
    (dword 17)
    (dword 0)
    end
```

Recursively sum linked list elements

Write a recursive version of `sum.lisp` (`rsum.lisp`) 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 `sum.lisp`. Remember to submit both files (see submission instructions below).

6 Part B

Copy a source block to a destination block

Write a program (`copy.lisp`) that copies a block of words from one part of memory to another (non-overlapping) area of memory, computing the checksum (XOR) of all the words copied. Your task is to write and simulate the following Y86 program. The required behavior of this program is defined by the example C function in `examples.c`.

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 eight-element source and destination blocks:

```
(align 4)
; Source block
src
(dword 1)
(dword 2)
(dword 3)
(dword 4)
(dword 5)
(dword 6)
(dword 7)
(dword 8)
; Destination block
dest
(dword 111)
(dword 222)
(dword 333)
(dword 444)
(dword 555)
(dword 666)
(dword 777)
(dword 888)
end
```

Optimize Block Copy and Checksum

Your task here is to optimize the block copy program to minimize the number of cycles required to achieve a copy. You are to test your program on two 1024-byte (256 element, 32-bit word) arrays. You may use loop unrolling and other optimizations as they seem appropriate. You may assume that the number of 32-bit words to copy is a multiple of four. You should include the number of steps that it takes the `y86` simulator to exactly finish execution.

For Part B, you need to turn in two programs: the copy (`copy.lisp`) program and the optimized copy (`ocopy.lisp`) program. Again, remember to submit two separate solution files (see below).

Hand In Instructions

We will use the Canvas electronic submission system – stay tuned to the class Webpage for the actual submission details.

Before submitting your solutions, please follow the instructions below.

- Make sure you have included your identifying information in your files.
- Remove any extraneous statements.

- For part A, submit your Y86 program solutions in two files named "`sum.lisp`" (iterative program) and "`rsum.lisp`" (recursive program) are the files containing your solution.
- After turning something in, if you discover a mistake and want to submit a revised copy, please resubmit your two files.

- For part B, handin your Y86 program solutions in the two files named "`copy.lisp`" (basic version) and "`ocopy.lisp`" (optimized version) that contain your solution.
- After turning something in, if you discover a mistake and want to submit a revised copy, please resubmit your two files.

```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 `examples.c`