# CMSC 313, Spring 2011
# Writing Y86 Assembly Code
# Assigned: Wednesday April-27
# Due: Wednesday May 11, 11:59PM

## 1 Introduction

In this lab, you will write three programs in Y86 assembly language and then assemble and simulate execution of your code using the Y86 tools provided with the project handout. A description of each program is given below.

## 2 Logistics

This is an individual assignment.

Any clarifications and revisions to the assignment will be posted on the project description page. A discussion board for this project is also available for your questions.

## 3 Handout Instructions

1. Start by copying the file `archlab-handout.tar` from the course's public directory `/afs/umbc.edu/users/c/m/cmsc313/pub/` to a (protected) directory in which you plan to do your work.
2. Then give the command: `tar xvf archlab-handout.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 The Y86 programs you will write

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 UMBC email ID in a comment at the beginning of each program.

### `sum.ys`: Iteratively sum linked list elements

Write a Y86 program named `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` named `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 named `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:

```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`

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

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

### Assembling and running your code

Once you have written one or more of your `.ys` files, assemble them using the command Y86 assembler. For example,

```
unix>  yas copy.ys
```

When your code assembles without erorrs, a `.yo` file will be created. Execute your program using the Y86 simulator. For example,

```
unix>  yis copy.yo
```

## 5 Evaluation

This project is worth 45 points, 15 points for each Y86 program. Ten (10) points are awarded for correctness, including proper handling of the `%ebp` stack frame register and functional equivalence with the example C functions in `examples.c`. Five (5) points are reserved for our subjective evaluation of your code and comments.

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 Handin Instructions

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

- To handin your files go to your `archlab-handout` directory and modify the `Makefile` by replacing the default student ID with your UMBC email ID. Then simply type the command

  ```
  Unix>  make handin
  ```

- After the handin, if you discover a mistake and want to submit a revised copy, type

  ```
  unix  make handin VERSION=2
  ```

  Keep incrementing the version number with each submission.

- You can verify your handin by looking in

  ```
  /afs/umbc.edu/users/c/m/cmsc313/pub/Proj6/
  ```

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

# 7 Hints

- Begin by carefully reading and understanding the C versions of the functions provided.
- Compile `/sim/misc/examples.c` and view the IA32 versions of these functions.
- Use the IA32 assembly code as a starting point and modify it to Y86 assembly as necessary