CPSC 313, Winter 2006 - Term 2
Understanding the Y86 Architecture
and its Sequential Implementation

Assigned: March 2, Due: Sunday, March 11, 11:59PM

For this problem set, you may either work by yourself or in a group of two. If you choose to work in a group of two, then please make sure that you clearly identify both group members on each portion of your solution.

1. Textbook problem 4.32. Hand this one in on paper.
2. Textbook problem 4.33. Hand this one in on paper.
3. The two architecture lab questions contained in this handout. Hand these two in electronically.

1 The Y86 Architecture Lab

In this problem, you will learn about the Y86 instruction set architecture, become familiar with its execution on the simulator, and modify the simulator to implement 2 new instructions.

When you have completed the lab, you will have a deeper understanding of the architecture and its sequential implementation.

2 Getting started

1. Start by copying the file `archlab-handout.tar`, available from the link on the course home page, 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`, `sim.tar`, 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.

```
 1 /* copy_block - Copy src to dest and return xor checksum of src */
 2 int copy_block(int *src, int *dest, int len)
 3 {
 4     int result = 0;
 5     while (len > 0) {
 6         int val = *src++;
 7         *dest++ = val;
 8         result ^= val;
 9         len--;
10     }
11     return result;
12 }
```

Figure 1: A C version of the Y86 function copy_block

4. Finally, change to the `sim` directory and build the Y86 tools:

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

3 Writing a Y86 program

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

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
```

3.1 Running your program

You may run your program on the instruction level simulator (yis) to see what it does. However, you must run the program using the graphical version of the sequential HCL simulator to help strengthen your understanding of how the processor works.

4 Modifying the Processor

In this problem, you will extend the Y86 processor to implement the two new instructions `iaddl` and `leave`. You will be working in directory `sim/seq` in this part.

Your task is to extend the SEQ processor to support two new instructions: `iaddl` (described in homework problems 4.32 and 4.34) and `leave` (described in homework problems 4.33 and 4.35). To add these instructions, you will modify the file `seq-full.hcl`, which implements the version of SEQ described in the CS:APP textbook. In addition, it contains declarations of some constants that you will need for your solution.

Your HCL file must begin with a header comment containing the following information:

- Your name and student ID.
- A description of the computations required for the `iaddl` instruction. Use the descriptions of `irmovl` and `OPl` in Figure 4.16 in the CS:APP text as a guide.
- A description of the computations required for the `leave` instruction. Use the description of `popl` in Figure 4.18 in the CS:APP text as a guide.

Building and Testing Your Solution

Once you have finished modifying the `seq-full.hcl` file, then you will need to build a new instance of the SEQ simulator (`ssim`) based on this HCL file, and then test it:

- *Building a new simulator.* You can use `make` to build a new SEQ simulator:

  ```
  unix>  make VERSION=full
  ```

  This builds a version of `ssim` that uses the control login you specified in `seq-full.hcl`. To save typing, you can assign `VERSION=full` in the Makefile.

- *Testing your solution on a simple Y86 program.* For your initial testing, we recommend running a simple program such as `asum.yo` in TTY mode, comparing the results against the ISA simulation:

  ```
  unix>  ./ssim -t asum.yo
  ```

  If the ISA test fails, then you should debug your implementation by single stepping the simulator in GUI mode:

```
unix>  ./ssim -g asum.yo
```

- *Testing your solution using the benchmark programs.* Once your simulator is able to correctly execute small programs, then you can automatically test it on the Y86 benchmark programs in `../y86-code`:

  ```
  unix>  (cd ../y86-code; make testssim)
  ```

  This will run `ssim` on the benchmark programs and check for correctness by comparing the resulting processor state with the state from a high-level ISA simulation. See file `../y86-code/README` file for more details.

- *Performing regression tests.* Once you can execute the benchmark programs correctly, then you should run the extensive set of regression tests in `../ptest`. To test everything except `iaddl` and `leave`:

  ```
  unix>  (cd ../ptest; make SIM=../seq/ssim)
  ```

  To test your implementation of `iaddl`:

  ```
  unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-i)
  ```

  To test your implementation of `leave`:

  ```
  unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-l)
  ```

  To test both `iaddl` and `leave`:

  ```
  unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-il)
  ```

For more information on the SEQ simulator refer to the handout *CS:APP Guide to Y86 Processor Simulators* (`simguide.pdf`).

5 Handing it in

Hand in your copy.ys file and your modified seq-full.hcl file using the handin program. The name of the assignment is: `problemset6`, and its due date is Sunday at midnight with the usual grace period.