Architecture of Computer and Network, 2022 Fall
Lab Assignment L3: Pipelined Processor
Due: Sun., Dec. 18, 11:59PM

1 Introduction

In this lab, you will learn about the design and implementation of a pipelined Y86 processor, optimizing both it and a benchmark program to maximize performance. You are allowed to make any semantics preserving transformations to the benchmark program, or to make enhancements to the pipelined processor, or both. When you have completed the lab, you will have a keen appreciation for the interactions between code and hardware that affect the performance of your programs.

The lab is organized into three 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 extend the SEQ and PIPE simulator with two new instructions. And in part C, you will modify the simulator source code to support more instructions.

2 Logistics

You will work on this lab alone.

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

3 Handout Instructions

You can download `archlab-handout.tar` from `http://learn.tsinghua.edu.cn/`, on course page.

1. Start by copying the file `archlab-handout.tar` 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 Part A (Score 20)

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 assembling 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
```

```
/* 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 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`.

`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 Part B (Score 80)

You will be working in directory `sim/seq` and `sim/pipe` in this part.

Your task in Part B is to extend the SEQ and PIPE processor to support two new instructions: `iaddl` (described in Homework problems 4.47 and 4.49)[1] (archlab2.html#fn1x0). `leave` (described in Homework problems 4.48 and 4.50)[2] (archlab3.html#fn2x0). To add these instructions, you will modify the file `seq-full.hcl` and `pipe-full.hcl`, which implement the version of SEQ and PIPE described in the CS:APP2e 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 ID.
- A description of the computations required for the `iaddl` instruction. Use the descriptions of `irmovl` and `OPl` in Figure 4.18 in the CS:APP2e text as a guide.
- A description of the computations required for the `leave` instruction. Use the description of `popl` in Figure 4.20 in the CS:APP2e text as a guide.

Building and Testing Your Solution

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

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

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

  This builds a version of `ssim` that uses the control logic you specified in `*-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 simple programs such as `asumi.yo` (testing `iaddl`) and `asuml.yo` (testing `leave`) in TTY mode, comparing the results against the ISA simulation (use `ssim` as example):

  ```
  unix>    ./ssim -t ../y86-code/asumi.yo
  unix>    ./ssim -t ../y86-code/asuml.yo
  ```

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

  ```
  unix>    ./ssim -g ../y86-code/asumi.yo
  unix>    ./ssim -g ../y86-code/asuml.yo
  ```
- *Retesting 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. Note that none of these programs test the added instructions. You are simply making sure that your solution did not inject errors for the original instructions. 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 simulators refer to the handout *CS:APP2e Guide to Y86 Processor Simulators* (`simguide.pdf`).

6 Part C (Optional; Score 5)

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

In this part, your task is to support more instructions in the PIPE simulator. Unlike Part B, you will modify not only `pipe-full.hcl`, but also `psim.c` and assembler in `misc` directory. To finish the task, you will need to read all codes of the simulator.

Here is one instruction that you are suggested to implement:

- `rmxchg` Exchange value in a register with a value in memory.

7 Evaluation

The lab is worth 100+ points: 20points for Part A, 80 points for Part B, and 5 points for Part C.

Part B

This part of the lab is worth 100 points:

- 15 points for your description of the computations required for the `iaddl` instruction. (in report)
- 15 points for your description of the computations required for the `leave` instruction. (in report)
- 10 points for passing the benchmark regression tests in `y86-code`, to verify that your simulator still correctly executes the benchmark suite.
- 30 points for passing the regression tests in `ptest` for `iaddl`.
- 30 points for passing the regression tests in `ptest` for `leave`.

8 Handin Instructions

You will be handing in these files:

- Part B: `seq-full.hcl` and `pipe-full.hcl`.
- Part B: Reports (.pdf or .docx or .doc), which includes the description of the instructions you implemented, the difficulties you faced, what you learned in the lab, etc.
- Part C: `sim.tar`/`sim.zip`, which contains your modified simulator sources.
- Part C: `test.ys`, an Y86 program that uses the instrcutions you added.
- Part C: Reports (.pdf or .docx or .doc) about how you modified the simulator.

You should submit `seq-full.hcl` and `pipe-full.hcl` to the online judge system http://acn.thucloud.com/. The other files should be packed in a <ID>.zip archive and submitted to http://learn.tsinghua.edu.cn.
