4190.308 Computer Architecture, Fall 2015
Arch Lab: Optimizing the Performance of a Pipelined Processor
Assigned: Thu., Oct 29, Due: Sat., Nov 14, 23:59PM

Surim Oh (`comparch@csap.snu.ac.kr`) is the lead person for this assignment.

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 In Part A, you will extend the SEQ simulator with two new instructions. This part will prepare you for Part B, the heart of the lab, where you will optimize the Y86 benchmark program and the processor design.

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

Download the `archlab-handout.tar` file from eTL.

1. Start by copying the file `archlab-handout.tar` to a 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

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

Your task in Part A is to extend the SEQ processor to support one new instruction: `iaddl` (described in CS:APP textbook,practice problems 4.48 and 4.50). To add this instruction, you will modify the file `seq-full.hcl`, which implements the version of SEQ described in the textbook. In addition, it contains declarations of some constants that you will need for your solution.

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 logic 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 simple programs such as `asumi.yo` (testing `iaddl`) in TTY mode, comparing the results against the ISA simulation:

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

- *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)
  ```

```
/*
 * ncopy - copy src to dst, returning number of positive ints
 * contained in src array.
 */
int ncopy(int *src, int *dst, int len)
{
    int count = 0;
    int val;

    while (len > 0) {
        val = *src++;
        *dst++ = val;
        if (val > 0)
            count++;
        len--;
    }
    return count;
}
```

Figure 1: **C version of the `ncopy` function.** See `sim/pipe/ncopy.c`.

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`:

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

To test your implementation of `iaddl`:

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

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

5 Part B

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

The `ncopy` function in Figure 1 copies a `len`-element integer array `src` to a non-overlapping `dst`, returning a count of the number of positive integers contained in `src`. Figure 2 shows the baseline Y86 version of `ncopy`. The file `pipe-full.hcl` contains a copy of the HCL code for PIPE, along with a declaration of the constant value `IIADDL`.

```
##################################################################
# ncopy.ys - Copy a src block of len ints to dst.
# Return the number of positive ints (>0) contained in src.
#
# Include your name and ID here.
#
# Describe how and why you modified the baseline code.
#
##################################################################
# Do not modify this portion
# Function prologue.
ncopy:  pushl %ebp              # Save old frame pointer
        rrmovl %esp,%ebp        # Set up new frame pointer
        pushl %esi              # Save callee-save regs
        pushl %ebx
        pushl %edi
        mrmovl 8(%ebp),%ebx     # src
        mrmovl 16(%ebp),%edx    # len
        mrmovl 12(%ebp),%ecx    # dst

##################################################################
# You can modify this portion
        # Loop header
        xorl %eax,%eax          # count = 0;
        andl %edx,%edx          # len <= 0?
        jle Done                # if so, goto Done:

Loop:   mrmovl (%ebx), %esi     # read val from src...
        rmmovl %esi, (%ecx)     # ...and store it to dst
        andl %esi, %esi         # val <= 0?
        jle Npos                # if so, goto Npos:
        irmovl $1, %edi
        addl %edi, %eax         # count++
Npos:   irmovl $1, %edi
        subl %edi, %edx         # len--
        irmovl $4, %edi
        addl %edi, %ebx         # src++
        addl %edi, %ecx         # dst++
        andl %edx,%edx          # len > 0?
        jg Loop                 # if so, goto Loop:
##################################################################
# Do not modify the following section of code
# Function epilogue.
Done:
        popl %edi               # Restore callee-save registers
        popl %ebx
        popl %esi
        rrmovl %ebp, %esp
        popl %ebp
        ret
##################################################################
# Keep the following label at the end of your function
End:
```


Figure 2: **Baseline Y86 version of the `ncopy` function.** See `sim/pipe/ncopy.ys`.

Your task in Part B is to modify `ncopy.ys` and `pipe-full.hcl` with the goal of making `ncopy.ys` run as fast as possible.

Coding Rules

You are free to make any modifications you wish, with the following constraints:

- Your `ncopy.ys` function must work for arbitrary array sizes. You might be tempted to hardwire your solution for 64-element arrays by simply coding 64 copy instructions, but this would be a bad idea because we will be grading your solution based on its performance on arbitrary arrays.
- Your `ncopy.ys` function must run correctly with YIS. By correctly, we mean that it must correctly copy the `src` block *and* return (in `%eax`) the correct number of positive integers.
- Your `pipe-full.hcl` implementation must pass the regression tests in `../y86-code` and `../ptest` (without the `-i` flags that test `iaddl`).

Other than that, you are free to implement the `iaddl` instruction if you think that will help. You are free to alter the branch prediction behavior or to implement techniques such as load bypassing. You may make any semantics preserving transformations to the `ncopy.ys` function, such as reordering instructions, replacing groups of instructions with single instructions, deleting some instructions, and adding other instructions.

Building and Running Your Solution

In order to test your solution, you will need to build a driver program that calls your `ncopy` function. We have provided you with the `gen-driver.pl` program that generates a driver program for arbitrary sized input arrays. For example, typing

```
unix>  make drivers
```

will construct the following two useful driver programs:

- `sdriver.yo`: A *small driver program* that tests an `ncopy` function on small arrays with 4 elements. If your solution is correct, then this program will halt with a value of 2 in register `%eax` after copying the `src` array.
- `ldriver.yo`: A *large driver program* that tests an `ncopy` function on larger arrays with 63 elements. If your solution is correct, then this program will halt with a value of 31 (`0x1f`) in register `%eax` after copying the `src` array.

Each time you modify your `ncopy.ys` program, you can rebuild the driver programs by typing

```
unix>  make drivers
```

Each time you modify your `pipe-full.hcl` file, you can rebuild the simulator by typing

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

If you want to rebuild the simulator and the driver programs, type

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

To test your solution on a small 4-element array, type

```
unix> ./psim sdriver.yo
```

To test your solution on a larger 63-element array, type

```
unix> ./psim ldriver.yo
```

To test your solution in GUI mode, type with `-g` flag

```
unix> ./psim -g ldriver.yo
```

Once your simulator correctly runs your version of `ncopy.ys` on these two block lengths, you will want to perform the following additional tests:

- *Testing your driver files on the ISA simulator.* Make sure that your `ncopy.ys` function works properly with YIS:

  ```
  unix> make drivers
  unix> ../misc/yis sdriver.yo
  ```

- *Testing your code on a range of block lengths with the ISA simulator.* The Perl script `correctness.pl` generates driver files with block lengths from 0 up to some limit (default 65), plus some larger sizes. It simulates them (by default with YIS), and checks the results. It generates a report showing the status for each block length:

  ```
  unix> ./correctness.pl
  ```

  If you get incorrect results for some length K, you can generate a driver file for that length that includes checking code, and where the result varies randomly:

  ```
  unix> ./gen-driver.pl -f ncopy.ys -n K -rc > driver.ys
  unix> make driver.yo
  unix> ../misc/yis driver.yo
  ```

  The program will end with register `%eax` having the following value:

  `0xaaaa` : All tests pass.

  `0xbbbb` : Incorrect count

0xcccc : Function ncopy is more than 1000 bytes long.

0xdddd : Some of the source data was not copied to its destination.

0xeeee : Some word just before or just after the destination region was corrupted.

- *Testing your pipeline simulator on the benchmark programs.* Once your simulator is able to correctly execute `sdriver.ys` and `ldriver.ys`, you should test it against the Y86 benchmark programs in `../y86-code`:

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

  This will run `psim` on the benchmark programs and compare results with YIS.

- *Testing your pipeline simulator with extensive regression tests.* Once you can execute the benchmark programs correctly, then you should check it with the regression tests in `../ptest`. For example, if your solution implements the `iaddl` instruction, then

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

6 Evaluation

Part A

This part of the lab is worth 35 points:

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

Part B

This part of the Lab is worth 80 points:

- 20 points each for your descriptions in the headers of `ncopy.ys` and `pipe-full.hcl`.
- 60 points for performance. To receive credit here, your solution must be correct, as defined earlier. That is, `ncopy` runs correctly with YIS, and `pipe-full.hcl` passes all tests in `y86-code` and `ptest`.

  We will express the performance of your function in units of *cycles per element* (CPE). That is, if the simulated code requires C cycles to copy a block of N elements, then the CPE is C/N. The PIPE simulator displays the total number of cycles required to complete the program. The baseline version of the `ncopy` function running on the standard PIPE simulator with a large 63-element array requires 914 cycles to copy 63 elements, for a CPE of $914/63 = 14.51$.

Since some cycles are used to set up the call to `ncopy` and to set up the loop within `ncopy`, you will find that you will get different values of the CPE for different block lengths (generally the CPE will drop as N increases). We will therefore evaluate the performance of your function by computing the average of the CPEs for blocks ranging from 1 to 64 elements. You can use the Perl script `benchmark.pl` in the `pipe` directory to run simulations of your `ncopy.ys` code over a range of block lengths and compute the average CPE. Simply run the command

```
unix>  ./benchmark.pl
```

to see what happens. For example, the baseline version of the `ncopy` function has CPE values ranging between 46.0 and 14.51, with an average of 16.44. Note that this Perl script does not check for the correctness of the answer. Use the script `correctness.pl` for this.

You should be able to achieve an average CPE of less than 10.0. Our best version averages 9.27.

By default, `benchmark.pl` and `correctness.pl` compile and test `ncopy.ys`. Use the `-f` argument to specify a different file name. The `-h` flag gives a complete list of the command line arguments.

7 Handin

- You will be handing in three files:
  - Part A: `seq-full.hcl`.
  - Part B: `ncopy.ys` and `pipe-full.hcl`.
- Make a tarball file `StudentID.tar` containing the above files

```
ex. 2015-12345.tar
```

- Send the tar file in an email (`comparch@csap.snu.ac.kr`)
  with the Subject Line **[Archlab]StudentID Name**

```
ex. [ArchLab]2015-12345 Surim Oh
```