CS 349, Summer 2002
Architecture Lab (Part C):
Optimizing the Performance of a Pipelined Processor
Assigned: Thu June 6, Due: Friday June 21, 11:59PM

Dave O'Hallaron (`droh@cs.cmu.edu`) is the lead person for this assignment.

1 Introduction

In Part C of the Architecture Lab, you will learn about the design and implementation of a pipelined Y86 processor, optimizing its performance on a benchmark program. 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.

2 Logistics

- You will work on this lab alone. Any clarifications and revisions to the assignment will be posted on the course Web page.
- In the following, `CLASSDIR` refers to

  ```
  /afs/cs/academic/class/15349-s02
  ```

3 Handout Instructions

As usual, all files you need are in the directory

```
CLASSDIR/archlabc
```

1. Start by copying the file `archlabc-handout.tar` from that directory to a (protected) directory in which you plan to do your work.

```
1 /*
2  * ncopy - copy src to dst, returning number of positive ints
3  * contained in src array.
4  */
5 int ncopy(int *src, int *dst, int len)
6 {
7     int count = 0;
8     int val;
9
10    while (len > 0) {
11        val = *src++;
12        *dst++ = val;
13        if (val > 0)
14            count++;
15        len--;
16    }
17    return count;
18 }
```

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

2. Then give the command: `tar xvf archlabc-handout.tar`. This will cause the following files to be unpacked into the directory: `README`, `Makefile`, `y86.tar`, `archlabc.ps`, and `archlabc.pdf`.

3. Next, give the command `tar xvf y86.tar`. This will create the directory `./sim`, which contains your personal copy of the Y86 simulators for this part of the Lab. (Make sure you do this step in order to get the latest version of the simulator code. Do **not** use your old version of the `./sim` directory from part B.)

4. Finally, change to the `./sim` directory and build the Y86 simulators and utility routines:

```
unix>  cd ./sim
unix>  make clean
unix>  make
```

You are now ready to start the Lab. Note that this is simpler than the build process for part B. You no longer need to explicitly make the GUI simulators.

4 Your Task

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 `./sim/pipe/pipe-full.hcl` contains a copy of the HCL code for PIPE, along with a declaration of the constant value IIADDL.

```
1  ##################################################################
2  # ncopy.ys - Copy a src block of len ints to dst.
3  # Return the number of positive ints (>0) contained in src.
4  #
5  # Include your name and ID here.
6  #
7  # Describe how and why you modified the baseline code.
8  #
9  ##################################################################
10         # Function prologue. Do not modify.
11 ncopy:  pushl %ebp              # Save old frame pointer
12         rrmovl %esp,%ebp        # Set up new frame pointer
13         pushl %esi              # Save callee-save regs
14         pushl %ebx
15         mrmovl 8(%ebp),%ebx     # src
16         mrmovl 12(%ebp),%ecx    # dst
17         mrmovl 16(%ebp),%edx    # len
18
19         # Loop header
20         xorl %esi,%esi          # count = 0;
21         andl %edx,%edx          # len <= 0?
22         jle Done                # if so, goto Done:
23
24         # Loop body.
25 Loop:   mrmovl (%ebx), %eax     # read val from src...
26         rmmovl %eax, (%ecx)     # ...and store it to dst
27         andl %eax, %eax         # val <= 0?
28         jle Npos                # if so, goto Npos:
29         irmovl $1, %edi
30         addl %edi, %esi         # count++
31 Npos:   irmovl $1, %edi
32         subl %edi, %edx         # len--
33         irmovl $4, %edi
34         addl %edi, %ebx         # src++
35         addl %edi, %ecx         # dst++
36         andl %edx,%edx          # len > 0?
37         jg Loop                 # if so, goto Loop:
38
39         # Function epilogue. Do not modify.
40 Done:   rrmovl %esi, %eax
41         popl %ebx
42         popl %esi
43         rrmovl %ebp, %esp
44         popl %ebp
45         ret
```

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

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

You will be handing in two files: `pipe-full.hcl` and `ncopy.ys`. Each file should begin with a header comment with the following information:

- Your name and Andrew ID.
- A high-level description of your code. In each case, describe how and why you modified your code.

5 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 `./sim/y86-code` and `./sim/ptest` (without the `-il` flags that test `iaddl` and `leave`).

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 swapping instructions, replacing groups of instructions with single instructions, deleting some instructions, and adding other instructions.

6 Building and Running Your Solution

You will be working entirely in the `./sim/pipe` directory.

In order to test your solution, you will need to build a driver program that calls your `ncopy` function. Two drivers are provided for your convenience:

- `sdriver.yo`: A *small driver program* that tests your `ncopy` function on small arrays with 4 elements. If your solution is correct, then this program will halt with a value of 3 in register `%eax` after copying the `src` array.
- `ldriver.yo`: A *large driver program* that tests your `ncopy` function on larger arrays with 63 elements. If your solution is correct, then this program will halt with a value of 62 (`0x3e`) in register `%eax` after copying the `src` array.

Each time you modify your `ncopy.ys` or `pipe-full.hcl` files, you can rebuild the driver programs and the GUI and TTY PIPE simulators by typing:

```
unix> make
```

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

```
unix> ./pipe_tk sdriver.yo
```

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

```
unix> ./pipe_tk 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> cd ./sim/pipe
  unix> make
  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 1 up to some limit (default 64), simulates them 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:

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

  The program will end with register `%eax` having value `0xaaaa` if the correctness check passes, `0xeeee` if the count is wrong, and `0xffff` if the count is correct, but the words are not all copied correctly.

- *Testing your 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 `./sim/y86-code`:

  ```
  unix> cd ./sim/y86-code
  unix> make pipe
  ```

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

- *Testing your simulator with extensive regression tests.* Once you can execute the benchmark programs correctly, then you should check it with the regression tests in `./sim/ptest`:

  ```
  unix>  cd ./sim/ptest
  unix>  make
  ```

  Notice that we've already set the appropriate `Makefile` variables so that you don't have to type any arguments to `make`.

7 Evaluation

This part of the Lab is worth 100 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 `./sim/y86-code` and `./sim/ptest`.

  We will express the performance of the function in *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. Both the GUI and TTY versions of the PIPE simulator display 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 1037 cycles to copy 63 elements, for a CPE of $1037/63 = 16.46$.

  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` to run simulations of your 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 45.0 and 16.45, with an average of 18.15. 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 12.0. Our best version averages 8.38.

8 Handin Instructions

- You will be handing in the files `ncopy.ys` and `pipe-full.hcl`.
- Make sure you have included your name and Andrew ID in a comment at the top of each of your handin files.

- To handin your solution, go to your protected directory that contains your `./sim` directory (i.e., the directory in which you unpacked the `archlabc-handout.tar` file), and type:

  ```
  make handin TEAM=teamname
  ```

  where `teamname` is your Andrew ID.

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

  ```
  make handin TEAM=teamname VERSION=2
  ```

  Keep incrementing the version number with each submission.

- You can verify your handin by looking in

  ```
  CLASSDIR/archlabc/handin
  ```

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

9 Hints

- By design, both `sdriver.yo` and `ldriver.yo` are small enough to debug with the GUI PIPE simulator. We find it easiest to use this program, and suggest that you use it as well.

- If the makefile suddenly stops working for you, then you've probably accidently deleted some intermediate file. You can fix this by going to the `./sim` directory and rebuilding the simulators:

  ```
  unix>  cd ./sim
  unix>  make clean
  unix>  make
  ```

- We're still working out a few minor kinks in the simulators. Here are a few that you should know about:

  – The `pipe_tk` program seg faults if you ask it to execute a file that is not a valid Y86 object file.

  – On some windowing systems, the "Program Code" window begins life as a closed icon when you `pipe_tk`. Simply click on the icon to expand the window.