CS 429H, Spring 2011
Pipelined Y86 processor optimization
Assigned: Fri Mar 25, Due: Friday Apr 8, 11:59PM

Christian Miller (`ckm@cs.utexas.edu`) is the lead person for this assignment.

1   Introduction

In this lab, you will optimize a Y86 benchmark program and a pipelined processor design at the same time. The objective is to find the best combination of the two, and minimize the average number of instruction cycles per array element on the benchmark program.

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 get a copy of this handout and the assignment code from the CS429H labs webpage:

`http://www.cs.utexas.edu/~fussell/courses/cs429h/labs/labs.shtml`

Start by copying the file `pipelab-handout.tar` to a (protected) directory in which you plan to do your work. Then give the command:

`tar xvf pipelab-handout.tar`

In the newly-created directory, run `make` to build the distribution.

```
 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.

4 Assignment

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, return-ing 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.

Your task in Part C 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 UTCS ID.

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

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.

```
 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 `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 `-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.

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 3 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 62 (`0x3e`) 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 your modify your `pipe-full.hcl` file, you can rebuild the simulator by typing

```
unix>  make psim
```

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

```
unix>  make
```

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

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

4

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

```
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 prop-
  erly 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 exe-
  cute `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 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)
  ```

5

5 Evaluation

This 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 `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 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` 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 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 7.43.

  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.

6 Handin Instructions

To submit your code, use the following command:

```
turnin --submit ckm pipelab ncopy.ys pipe-full.hcl
```

Make sure you have included your name and UTCS ID in a comment at the top of each of your files.