4190.308.002, Fall 2018
Y86-64 Pipelined Processor
Assigned: November 8, Due: November 22, 11:59PM

1 Introduction

In this lab, you will learn about the design and implementation of a pipelined Y86-64 processor, optimizing both it and a benchmark program to maximize performance. You are allowed to make any semantics-preserving transformation to the benchmark program, or to make enhancements to the sequential 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 latest lab was organized into two parts. In Part A you will write some simple Y86-64 programs and become familiar with the Y86-64 tools. In Part B, you will extend the SEQ simulator with a new instruction. These two parts prepare you for Part C, where you will optimize the Y86-64 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 eTL.

3 Handout Instructions

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.pdf`, and `simguide.pdf`.

```
1 /*
2  * ncopy - copy src to dst, returning number of positive ints
3  * contained in src array.
4  */
5 word_t ncopy(word_t *src, word_t *dst, word_t len)
6 {
7     word_t count = 0;
8     word_t 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`.

3. Next, give the command `tar xvf sim.tar`. This will create the directory `sim`, which contains your personal copy of the Y86-64 tools. You will be doing all of your work inside this directory.
4. Finally, change to the `sim` directory and build the Y86-64 tools:

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

4 Part C

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 Y86-64 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 `IIADDQ`.

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

```
1  ##################################################################
2  # ncopy.ys - Copy a src block of len words to dst.
3  # Return the number of positive words (>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 # Do not modify this portion
11 # Function prologue.
12 # %rdi = src, %rsi = dst, %rdx = len
13 ncopy:
14
15 ##################################################################
16 # You can modify this portion
17         # Loop header
18         xorq %rax,%rax          # count = 0;
19         andq %rdx,%rdx          # len <= 0?
20         jle Done                # if so, goto Done:
21
22 Loop:   mrmovq (%rdi), %r10     # read val from src...
23         rmmovq %r10, (%rsi)     # ...and store it to dst
24         andq %r10, %r10         # val <= 0?
25         jle Npos                # if so, goto Npos:
26         irmovq $1, %r10
27         addq %r10, %rax         # count++
28 Npos:   irmovq $1, %r10
29         subq %r10, %rdx         # len--
30         irmovq $8, %r10
31         addq %r10, %rdi         # src++
32         addq %r10, %rsi         # dst++
33         andq %rdx,%rdx          # len > 0?
34         jg Loop                 # if so, goto Loop:
35 ##################################################################
36 # Do not modify the following section of code
37 # Function epilogue.
38 Done:
39         ret
40 ##################################################################
41 # Keep the following label at the end of your function
42 End:
```

Figure 2: **Baseline Y86-64 version of the `ncopy` function.** See `sim/pipe/ncopy.ys`.

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 `%rax`) the correct number of positive integers.
- The assembled version of your `ncopy` file must not be more than 1000 bytes long. You can check the length of any program with the `ncopy` function embedded using the provided script `check-len.pl`:

  ```
  unix>  ./check-len.pl < ncopy.yo
  ```

- Your `pipe-full.hcl` implementation must pass the regression tests in `../y86-code` and `../ptest` (without the `-i` flag that tests `iaddq`).

Other than that, you are free to implement the `iaddq` instruction if you think that will help. 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. You may find it useful to read about loop unrolling in Section 5.8 of CS:APP3e.

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 `%rax` 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 `%rax` 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 in GUI mode on a small 4-element array, type

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

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 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
  ```

  This script generates test programs where the result count varies randomly from one run to another, and so it provides a more stringent test than the standard drivers.

  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 `%rax` 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 Y86-64 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 `iaddq` instruction, then

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

- *Testing your code on a range of block lengths with the pipeline simulator.* Finally, you can run the same code tests on the pipeline simulator that you did earlier with the ISA simulator

  ```
  unix>  ./correctness.pl -p
  ```

5 Evaluation

Part C

This part of the Lab is worth 100 points: **You will not receive any credit if either your code for** `ncopy.ys` **or your modified simulator fails any of the tests described earlier.**

- 20 points each for your descriptions in the headers of `ncopy.ys` and `pipe-full.hcl` and the quality of these implementations.

- 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 897 cycles to copy 63 elements, for a CPE of $897/63 = 14.24$.

  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 29.00 and 14.27, with an average of 15.18. 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 9.00. Our best version averages 7.48. If your average CPE is c, then your score S for this portion of the lab will be:

$$S = \begin{cases} 0\,, & c > 10.5 \\ 20 \cdot (10.5 - c)\,, & 7.50 \leq c \leq 10.50 \\ 60\,, & c < 7.50 \end{cases}$$

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

- You will be handing in set of files:
  - Part C: `pipe-full.hcl` and `ncopy.ys`.
- You zip this file as Lab4.tar as the command `tar cvf Lab4.tar sim/pipe/pipe-full.hcl sim/pip` and submit Lab4.tar on the eTL.