Optimizing the Performance of a Pipelined Processor
优化流水线处理器的性能

1 Introduction

1 简介

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 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.

在本实验中，您将了解流水线 Y86-64 处理器的设计和实现，优化它和基准程序以最大限度地提高性能。您可以对基准程序进行任何语义保持转换，或者对流水线处理器进行增强，或者两者兼而有之。当你完成实验后，你会对影响程序性能的代码和硬件之间的交互产生强烈的兴趣。

The lab is organized into three parts, each with its own handin. 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 will prepare you for Part C, the heart of the lab, where you will optimize the Y86-64 benchmark program and the processor design.

实验分为三个部分，每个部分都有自己的上交成果。在 A 部分中，您将编写一些简单的 Y86-64 程序，并熟悉 Y86-64 工具。在 B 部分中，您将使用新的指令扩展 SEQ 模拟器。这两个部分将为您完成 C 部分，即实验的核心做好准备，在这里您将优化 Y86-64 基准程序和处理器设计。

2 Logistics

2 组织工作

You will work on this lab alone. Any clarifications and revisions to the assignment will be posted on the course Web page.

你将独立完成本实验。作业的任何澄清和修改都将发布在课程网页上。

3 Handout Instructions

3 讲义说明

SITE-SPECIFIC: Insert a paragraph here that explains how students should download the archlab-handout.tar file.

特定说明：在此处插入一段，解释学生应如何下载 archlab-handout.tar 文件。

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

1.首先复制文件 archlab-handout.tar 到您计划在其中执行工作的（受保护的）目录。

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.

2.然后给出命令：tar xvf archlab-handout.tar。这将导致以下文件解压缩到目录中：README、Makefile、sim.tar、archlab.pdf 和 simguide.pdf。

3. Next, give the command tar xvf sim.tar. This will create the directory sim, which contains your personal copy of the Y86-64 tools. You will be doing all of your work inside this directory.

3.接下来，给出命令 tar xvf sim.tar。这将创建目录 sim，其中包含 Y86-64 工具的个人副本。您将在该目录中完成所有工作。

4. Finally, change to the sim directory and build the Y86-64 tools:

4.最后，切换到 sim 目录并构建 Y86-64 工具：

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

4 Part A

4 A 部分

You will be working in directory sim/misc in this part.

在本部分中，您将在目录 sim/misc 中工作。

Your task is to write and simulate the following three Y86-64 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 assemblying them with the program YAS and then running them with the instruction set simulator YIS.

您的任务是编写并模拟以下三个 Y86-64 程序。这些程序所需的行为由 examples.c 中的示例 C 函数定义。请确保在每个程序开头的注释中输入您的姓名和 ID。您可以先用程序 YAS 汇编这些程序，然后用指令集模拟器 YIS 运行它们，从而测试您的程序。

In all of your Y86-64 functions, you should follow the x86-64 conventions for passing function arguments, using registers, and using the stack. This includes saving and restoring any callee-save registers that you use.

在所有 Y86-64 函数中，应该遵循 x86-64 惯例来传递函数参数、使用寄存器和使用栈。这包括保存和恢复您使用的任何被调用方保存的寄存器。

sum.ys: Iteratively sum linked list elements

sum.ys:对链表元素进行迭代求和

Write a Y86-64 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 Y86-64 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:

编写 Y86-64 程序 sum.ys 对链表元素进行迭代求和。您的程序应该由一些代码组成，这些代码设置栈结构，调用函数，然后停机。在这种情况下，函数应该是 Y86-64 代码（sum_list 函数），功能等同于图 1 中 C sum_list 函数。使用以下三元素列表测试您的程序：

```
# Sample linked list
.align 8
ele1:
	.quad 0x00a
	.quad ele2
ele2:
	.quad 0x0b0
	.quad ele3
ele3:
	.quad 0xc00
	.quad 0
```

```
1 /* linked list element */
2 typedef struct ELE {
3 	long val;
4 	struct ELE *next;
5 } *list_ptr;
```

```

/* sum_list - Sum the elements of a linked list */
long sum_list(list_ptr ls)
{
    long val = 0;
    while (ls) {
        val += ls->val;
        ls = ls->next;
    }
    return val;
}

/* rsum_list - Recursive version of sum_list */
long rsum_list(list_ptr ls)
{
    if (!ls)
        return 0;
    else {
        long val = ls->val;
        long rest = rsum_list(ls->next);
        return val + rest;
    }
}

/* copy_block - Copy src to dest and return xor checksum of src */
long copy_block(long *src, long *dest, long len)
{
    long result = 0;
    while (len > 0) {
        long val = *src++;
        *dest++ = val;
        result ˆ= val;
        len--;
    }
    return result;
}
```

Figure 1: C versions of the Y86-64 solution functions. See sim/misc/examples.c

图 1: C 语言版本 Y86-64 解决方案函数。请参阅 sim/misc/examples.c

rsum.ys: Recursively sum linked list elements

rsum.ys：递归求和链表元素

Write a Y86-64 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.

编写一个 Y86-64 程序 rsum.ys 对链表元素进行递归求和。该代码应与 sum.ys 的代码类似，但它使用一个函数 rsum_list 递归地对一系列数字求和，如图 1 中的 C 语言函数 rsum_list 所示。使用与测试 list.ys 相同的三元素列表测试您的程序。

copy.ys: Copy a source block to a destination block

copy.ys：将源块复制到目标块

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.

编写一个程序（copy.ys），将一个若干字构成的数据块从内存的一部分复制到另一个（非重叠区）内存区域，计算所有复制字的校验和（Xor）。

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:

您的程序应该由设置栈帧、调用函数复制块然后停机的代码组成。该函数在功能上应等同于图 1 所示的 C 复制块函数。使用以下三元素源块和目标块测试程序：

```
.align 8
# Source block
src:
	.quad 0x00a
	.quad 0x0b0
	.quad 0xc00
# Destination block
dest:
	.quad 0x111
	.quad 0x222
	.quad 0x333
```

5 Part B

5 B 部分

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

在本部分中，您将在目录 sim/seq 中工作。

Your task in Part B is to extend the SEQ processor to support the iaddq, described in Homework problems 4.51 and 4.52. To add this instructions, you will modify the file seq-full.hcl, which implements the version of SEQ described in the CS:APP3e textbook. In addition, it contains declarations of some constants that you will need for your solution.

您在 B 部分的任务是扩展 SEQ 处理器以支持 iaddq 指令，如家庭作业问题 4.51 和 4.52 所述。要添加此指令，您将修改文件 seq-full.hcl，它实现了 CS:APP3e 教科书中描述的 SEQ 版本。此外，它还包含解决方案所需的一些常量的声明。

Your HCL file must begin with a header comment containing the following information:

HCL 文件必须以注释头开始，其中包含以下信息：

• Your name and ID.

•您的姓名和 ID。

• A description of the computations required for the iaddq instruction. Use the descriptions of irmovq and OPq in Figure 4.18 in the CS:APP3e text as a guide.

•iaddq 指令所需计算的描述。参考 CS:APP3e 教材中图 4.18 中的 irmovq 和 OPq 描述。

Building and Testing Your Solution

构建和测试您的解决方案

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:

完成对 seq-full.hcl 文件的修改后，则需要基于此 HCL 文件构建 SEQ 模拟器（ssim）的新实例，然后对其进行测试：

• Building a new simulator. You can use make to build a new SEQ simulator:

•构建新的模拟器。您可以使用 make 构建新的 SEQ 模拟器：

```
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.

这将构建一个使用 seq-full.hcl 中指定的控制逻辑的 ssim 版本。要保存键入，可以在 Makefile 中指定 VERSION=full。

• Testing your solution on a simple Y86-64 program. For your initial testing, we recommend running simple programs such as asumi.yo (testing iaddq) in TTY mode, comparing the results against the ISA simulation:

•在一个简单的 Y86-64 程序上测试您的解决方案。对于初始测试，我们推荐在 TTY 模式下运行简单的程序例如 asumi.yo（测试 iaddq），将结果与 ISA 模拟进行比较：

```
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:

如果 ISA 测试失败，则应在 GUI 模式下单步执行模拟器来调试实现：

```
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 Y86-64 benchmark programs in ../y86-code:

•使用基准程序重新测试解决方案。一旦模拟器能够正确执行小程序，您就可以用../Y86-code 中的 Y86-64 基准程序自动测试它：

```
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.

这将在基准程序上运行 ssim，并通过将生成的处理器状态与高级 ISA 模拟的状态进行比较来检查正确性。请注意，这些程序都没有测试添加的指令。您只是确保您的解决方案没有为原始指令注入错误。有关更多详细信息，请参阅文件../y86 code/README。

• 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 iaddq and leave:

•执行回归测试。一旦您能够正确执行基准程序，那么您应该在../ptest 中运行大量的回归测试。要测试除 iaddq 之外的所有内容，请执行以下操作：

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

To test your implementation of iaddq:

要测试 iaddq 的实现，请执行以下操作：

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

For more information on the SEQ simulator refer to the handout CS:APP3e Guide to Y86-64 Processor Simulators (simguide.pdf).
有关 SEQ 模拟器的更多信息，请参阅讲义 CS:APP3e Y86-64 处理器模拟器指南（simguide.pdf）。

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

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

Figure 2: C version of the ncopy function. See sim/pipe/ncopy.c.
图 2：ncopy 函数的 C 语言版本。参见 sim/pipe/ncopy.c

6 Part C
6 C 部分

You will be working in directory sim/pipe in this part.
在本部分中，您将在目录 sim/pipe 中工作。

The ncopy function in Figure 2 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 3 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.
图 2 中的 ncopy 函数将 len 个元素整数数组 src 复制到非重叠的 dst，返回 src 中包含的正整数的数量。图 3 显示了 ncopy 的基线 Y86-64 版本。文件 pipe-full.hcl 包含 PIPE 的 HCL 代码副本，以及常量值 IIADDQ 的声明。

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.
在 C 部分中，您的任务是修改 ncopy.ys 和 pipe-full.hcl，目标是使 ncopy.ys 尽可能快地运行。

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:
您将提交两个文件：pipe-full.hcl 和 ncopy.ys。每个文件应以注释头开始，其中包含以下信息：

• Your name and ID.
•您的姓名和 ID。
• A high-level description of your code. In each case, describe how and why you modified your code.

•代码的高级描述。在每种情况下，描述您修改代码的方式和原因。

Coding Rules

编码规则

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

您可以根据以下限制自由进行任何修改：

• 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.

•您的 ncopy.ys 函数必须适用于任意数组大小。您可能会试图通过编写 64 个复制指令来硬连接 64 元素数组的解决方案，但这不是一个好主意，因为我们将根据解决方案在任意数组上的性能对其进行分级。

```
##################################################################
# ncopy.ys - Copy a src block of len words to dst.
# Return the number of positive words (>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.
# %rdi = src, %rsi = dst, %rdx = len
ncopy:

##################################################################
# You can modify this portion
# Loop header
        xorq %rax,%rax          # count = 0;
        andq %rdx,%rdx          # len <= 0?
        jle Done                # if so, goto Done:

Loop:   mrmovq (%rdi), %r10     # read val from src...
        rmmovq %r10, (%rsi)     # ...and store it to dst
        andq %r10, %r10         # val <= 0?
        jle Npos                # if so, goto Npos:
        irmovq $1, %r10
        addq %r10, %rax         # count++
Npos:   irmovq $1, %r10
        subq %r10, %rdx         # len—
        irmovq $8, %r10
        addq %r10, %rdi         # src++
        addq %r10, %rsi         # dst++
        andq %rdx,%rdx          # len > 0?
        jg Loop                 # if so, goto Loop:
######################################################################
# Do not modify the following section of code
# Function epilogue.
Done:
        ret
######################################################################
# Keep the following label at the end of your function
End:
```

Figure 3: Baseline Y86-64 version of the ncopy function. See sim/pipe/ncopy.ys.
图 3:ncopy 函数的基线 Y86-64 版本。请参见 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 %rax) the correct number of positive integers.
•您的 ncopy.ys 函数必须与 YIS 一起正确运行。正确地说，我们的意思是它必须正确地复制 src 块并正确返回（在%rax 中）其中正整数的数量。

• 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:
•ncocy 文件的汇编版本长度不得超过 1000 字节。您可以使用提供的脚本 check-len.pl 来检查嵌入 ncopy 函数的任何程序的长度。

```
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).
•您的 pipe-full.hcl 实现必须通过../y86-code 和../ptest 中的回归测试（不要使用-i 标志，那是测试 iaddq 指令用的）。

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.
除此之外，如果您认为 iaddq 指令有帮助，您可以自由实现它。您可以在保持语义不变的情况下对 ncopy.ys 函数进行任何转换，例如重新排序指令、用单个指令替换指令组、删除一些指令以及添加其他指令。您可能会发现阅读 CS:APP3e 第 5.8 节中关于循环展开的内容很有用。

Building and Running Your Solution
构建和运行解决方案

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
为了测试您的解决方案，您需要构建一个调用 ncopy 函数的驱动程序。我们为您提供了 gen-driver.pl 程序，它可以为任意大小的输入数组生成驱动程序。例如，键入

```
unix> make drivers
```

will construct the following two useful driver programs:
将构建以下两个有用的驱动程序：

• 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.

•sdriver.yo：一个小的驱动程序，在 4 个元素的小数组上测试 ncopy 函数。如果您的解决方案是正确的，那么在复制 src 数组后停机，在寄存器%rax 中返回值 2。

• 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.

•ldriver.yo：一个大型驱动程序，在 63 个元素的大型数组上测试 ncopy 函数。如果您的解决方案是正确的，那么在复制 src 数组后停机，在寄存器%rax 中返回值 31（0x1f）。

Each time you modify your ncopy.ys program, you can rebuild the driver programs by typing
每次修改 ncopy.ys 程序，您可以通过键入以下命令重新构建驱动程序

```
unix> make drivers
```

Each time you modify your pipe-full.hcl file, you can rebuild the simulator by typing
每次修改 pipe-full.hcl 文件，您可以通过键入以下命令重新构建模拟器

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

If you want to rebuild the simulator and the driver programs, type
如果要重新构建模拟器和驱动程序，请键入

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

To test your solution in GUI mode on a small 4-element array, type
要在小型 4 元素数组上以 GUI 模式测试解决方案，请键入

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

To test your solution on a larger 63-element array, type
要在更大的 63 元素数组上测试解决方案，请键入

```
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:
一旦你的模拟器在这两个块长度上正确运行你的 ncopy 版本，您将需要执行以下附加测试：

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

•在 ISA 模拟器上测试驱动程序文件。确保您的 ncopy.ys 函数与 YIS 一起正常工作：

```
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:

•使用 ISA 模拟器在一系列块长度上测试代码。Perl 脚本 correctness.pl 生成的驱动程序文件的块长度从 0 到一定限界（默认值 65），再加上一些较大的大小。该脚本模拟不同大小块复制程序（默认情况下使用 YIS），并检查结果。该脚本生成一个报告，显示每个块长度的状态：

```
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:
此脚本生成测试程序，其中结果计数随着每次运行而随机变化，因此它提供了比标准驱动程序更严格的测试。如果某个长度 K 的结果不正确，可以生成该长度的驱动程序文件，其中包含检查代码，并且结果随机变化：

```
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:
程序将以寄存器%rax 具有以下值结束：
0xaaaa : All tests pass.
0xaaaa：所有测试均通过。
0xbbbb : Incorrect count
0xbbbb:计数不正确
0xcccc : Function ncopy is more than 1000 bytes long.
0xcccc:函数 ncopy 的长度超过 1000 字节。
0xdddd : Some of the source data was not copied to its destination.
0xdddd:某些源数据未复制到其目的地。
0xeeee : Some word just before or just after the destination region was corrupted.
0xeeee:目的区域之前或之后的某个字已损坏。

• 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:
•在基准程序上测试流水线模拟器，一旦模拟器能够正确执行 sdriver.ys 和 ldriver.ys，您应该使用../Y86-code 中的 Y86-64 基准程序进行测试：

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

This will run psim on the benchmark programs and compare results with YIS.
这将在基准程序上运行 psim，并将结果与 YIS 进行比较。

• 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
•使用广泛的回归测试测试流水线模拟器，一旦能够正确执行基准程序，就应该使用../ptest 中的回归测试进行检查。例如，如果您的解决方案实现了 iaddq 指令，那么

```
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
•使用流水线模拟器在一系列块长度上测试代码，最后，您可以在流水线模拟器上运行与之前使用 ISA 模拟器相同的代码测试

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

7 Evaluation
7 评估

The lab is worth 190 points: 30 points for Part A, 60 points for Part B, and 100 points for Part C.
实验评估值 190 分：A 部分 30 分，B 部分 60 分，C 部分 100 分。

Part A
A 部分

Part A is worth 30 points, 10 points for each Y86-64 solution program. Each solution program will be evaluated for correctness, including proper handling of the stack and registers, as well as functional equivalence with the example C functions in examples.c.
A 部分值 30 分，每个 Y86-64 解决方案程序得 10 分。将评估每个解决方案程序的正确性，包括栈和寄存器的正确处理，以及与 examples.c 中的示例 C 函数的功能等效性。
The programs sum.ys and rsum.ys will be considered correct if the graders do not spot any errors in them, and their respective sum list and rsum list functions return the sum 0xcba in register %rax.
如果评分器在程序 sum.ys 和 rsum.ys 中没有发现任何错误，并且它们各自的 sum_list 和 rsum_list 函数在寄存器%rax 中返回总和为 0xcba，则认为这些程序是正确的。
The program copy.ys will be considered correct if the graders do not spot any errors in them, and the copy block function returns the sum 0xcba in register %rax, copies the three 64-bit values 0x00a, 0x0b, and 0xc to the 24 bytes beginning at address dest, and does not corrupt other memory locations.
如果评分器在程序 copy.ys 中没有发现任何错误，并且复制块函数在寄存器%rax 中返回总和为 0xcba，将三个 64 位值 0x00a、0x0b 和 0xc 复制到从地址 dest 开始的 24 个字节，并且不会损坏其他内存位置，则认为 copy.ys 是正确的。
Part B
B 部分
This part of the lab is worth 35 points:
这部分实验评估值 35 分：
• 10 points for your description of the computations required for the iaddq instruction.
•您对 iaddq 指令所需计算的描述得 10 分。
• 10 points for passing the benchmark regression tests in y86-code, to verify that your simulator still correctly executes the benchmark suite.
•通过 y86-code 中的基准回归测试得 10 分，以验证模拟器仍然正确执行基准测试套件。
• 15 points for passing the regression tests in ptest for iaddq.
•通过 iaddq 的 ptest 回归测试 15 分。
Part C
C 部分
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.
这部分实验评估值 100 分：如果你的代码 ncopy.ys 或者您修改的模拟器未能通过前面描述的任何测试，你将不会得到任何分数。
• 20 points each for your descriptions in the headers of ncopy.ys and pipe-full.hcl and the quality of these implementations.
•ncopy.ys 和 pipe-full.hcl 头中的每个描述，以及这些实现的质量，各值 20 分。
• 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.
•性能值 60 分。要在这方面获得分数，您的解决方案必须是正确的，如前面所定义的。也就是说，ncopy 使用 YIS 正确运行，并且 pipe-full.hcl 通过了 y86-code 和 ptest 中的所有测试。
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.
我们将以每个元素的周期数（CPE）为单位表示您的函数性能。也就是说，如果模拟代码需要 C 个周期来复制 N 个元素的块，那么 CPE 就是 C/N。PIPE 模拟器显示完成程序所需的总周期数。在标准 PIPE 模拟器上运行的 ncopy 函数的基线版本（带有大型 63 个元素数组）需要 897 个周期才能复制 63 个元素，相当于 CPE 为 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).
由于一些周期用于设置对 ncopy 的调用和在 ncopy 中设置循环，您会发现对于不同的块长度，您会得到不同的 CPE 值（通常 CPE 会随着 N 的增加而下降）。
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
因此，我们将通过计算从 1 到 64 个元素的块的平均 CPE 来评估您的函数性能。您可以使用 Perl 脚本 benchmark.pl 在 pipe 目录中运行一系列块长度的 ncopy.ys 模拟代码，并计算平均 CPE。只需运行命令

```
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.
看看会发生什么。例如，ncopy 函数的基线版本的 CPE 值介于 29.00 和 14.27 之间，平均值为 15.18。请注意，此 Perl 脚本并不检查答案的正确性。使用脚本 correctness.pl 检查程序正确性。
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:
您应该能够取得低于 9.00 的平均 CPE，我们的最佳版本平均为 7.48。如果您的平均 CPE 为 c，那么您在这部分实验中的分数 S 为：

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

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.
默认情况下，benchmark.pl 和 correctness.pl 编译并测试 ncopy.ys。使用-f 参数指定不同文件名，-h 标志给出了命令行参数的完整列表。

8 Handin Instructions

8 上交说明

SITE-SPECIFIC: Insert a description that explains how students should hand in the three parts of the lab. Here is the description we use at CMU.
特定说明：插入说明解释学生应如何提交实验的三个部分。这是我们在 CMU 使用的说明。
• You will be handing in three sets of files:
•您将提交三套文件：

– Part A: sum.ys, rsum.ys, and copy.ys.
－A 部分：sum.ys，rsum.ys 和 copy.ys。
– Part B: seq-full.hcl.
－B 部分：seq-full.hcl。
– Part C: ncopy.ys and pipe-full.hcl.
－C 部分：ncopy.ys 和 pipe-full.hcl。
• Make sure you have included your name and ID in a comment at the top of each of your handin files.
•确保在每个 handin 文件顶部的注释中包含您的姓名和 ID。
• To handin your files for part X, go to your archlab-handout directory and type:
•要提交第 X 部分的文件，请转到 archlab-handout 目录并键入：
unix> make handin-partX TEAM=teamname
where X is a, b, or c, and where teamname is your ID. For example, to handin Part A:
其中 X 是 a、b 或 c，teamname 是您的 ID。例如，要提交 Part a:
unix> make handin-parta TEAM=teamname
• After the handin, if you discover a mistake and want to submit a revised copy, type
•在提交之后，如果发现错误并想提交修订后的副本，请键入
unix make handin-partX TEAM=teamname VERSION=2
Keep incrementing the version number with each submission.
每次提交时不断增加版本号。
• You can verify your handin by looking in CLASSDIR/archlab/handin-partX You have list and insert permissions in this directory, but no read or write permissions.
•您可以通过在 CLASSDIR/archlab/handin-partX 中查找来验证您的提交。您拥有此目录列表和插入权限，但没有读或写权限。
9 Hints
9 个提示
• By design, both sdriver.yo and ldriver.yo are small enough to debug with in GUI mode. We find it easiest to debug in GUI mode, and suggest that you use it.
•根据设计，两个驱动程序 sdriver.yo 和 ldriver.yo 都足够小，可以在 GUI 模式下进行调试。我们发现在 GUI 模式下调试最容易，建议您使用它。
• If you running in GUI mode on a Unix server, make sure that you have initialized the DISPLAY environment variable:
•如果在 Unix 服务器上以 GUI 模式运行，请确保已初始化 DISPLAY 环境变量：
unix> setenv DISPLAY myhost.edu:0
• With some X servers, the “Program Code” window begins life as a closed icon when you run psim or ssim in GUI mode. Simply click on the icon to expand the window.
•对于某些 X 服务器，当您在 GUI 模式下运行 psim 或 ssim 时，“程序代码”窗口开始以关闭图标的形式出现。
• With some Microsoft Windows-based X servers, the “Memory Contents” window will not automatically resize itself. You’ll need to resize the window by hand.
•对于一些基于 Microsoft Windows 的 X 服务器，“内存内容”窗口不会自动调整大小。您需要手动调整窗口大小。
• The psim and ssim simulators terminate with a segmentation fault if you ask them to execute a file that is not a valid Y86-64 object file.

•如果您要求 psim 和 ssim 模拟器执行一个不是有效 Y86-64 目标代码的文件，则它们会因段错误而终止。