CS 349, Summer 2015
Cs349,2015 年夏天
Optimizing the Performance of a Pipelined Processor
优化流水线处理器的性能

Assigned: June 6, Due: June 21, 11:59PM
分配时间: 6 月 6 日，截止日期: 6 月 21 日，晚上 11:59

Harry Bovik (bovik@cs.cmu.edu) is the lead person for this assignment.
Harry Bovik (Bovik@cs.cmu.edu)是这项任务的负责人。

1 Introduction
引言

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.
在本实验室中，您将了解 y86-64 流水线处理器的设计和实现，优化它和一个基准程序，以最大限度地提高性能。你可以对基准程序进行任何保留语义的转换，或者对流水线处理器进行增强，或者两者兼而有之。当你完成了实验，你会对代码和硬件之间的交互作用产生强烈的兴趣，这些交互作用会影响你的程序的性能。

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.
实验室分为三个部分，每个部分都有自己的实验室。在 a 部分，你将编写一些简单的 y86-64 程序，并熟悉 y86-64 工具。在 b 部分，你将用一个新的指令来扩展 SEQ 模拟器。这两个部分将为您准备 c 部分，即实验室的核心部分，在那里您将优化 y86-64 基准测试程序和处理器设计。

2 Logistics
物流

You will work on this lab alone.
你一个人在这个实验室工作。

Any clarifications and revisions to the assignment will be posted on the course Web page.
任何对作业的澄清和修改都会在课程网页上公布。

3 Handout Instructions
讲义

SITE-SPECIFIC: Insert a paragraph here that explains how students should download the **archlab-handout.tar** file.
SITE-SPECIFIC: 在这里插入一段，解释学生应该如何下载 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.
首先，将 archlab-handout.tar 文件复制到一个(受保护的)目录，您计划在其中完成工作。

2. Then give the command: tar xvf archlab-handout.tar. This will cause the following
然后给出命令: tar xvf archlab-handout.tar
   files to be unpacked into the directory: README, Makefile, sim.tar, archlab.pdf, and simguide.pdf.
   要解压缩到目录中的文件: 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.
下一步，命令 tar xvf sim.tar。这将创建目录 sim，其中包含你的 y86-64 工具的个人副本。你将在这个目录中完成所有的工作。

4. Finally, change to the sim directory and build the Y86-64 tools:
最后，切换到 sim 目录，构建 y86-64 工具:

```
unix>  cd sim
Unix > cd sim 卡
unix>  make clean; make
使干净; 使
```

4 Part A
A 部分

You will be working in directory sim/misc in this part.
在这一部分中，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.
你的任务是编写和模拟以下三个 y86-64 程序。这些程序的必要行为是由 example.c 中的示例 c 函数定义的。一定要在每个程序开始的注释中写上你的名字和 ID。您可以先用程序 YAS 组装程序，然后用指令集仿真器 YIS 运行程序，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，应遵循 x86-64 约定传递函数参数、使用寄存器和使用堆栈。这包括保存和恢复你使用的所有调用方保存寄存器。

sum.ys: Iteratively sum linked list elements
Ys: 迭代求和链表元素

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:

编写一个 y86-64 程序 sum.ys，迭代求和链表的元素。你的程序应该包含一些代码来建立堆栈结构，调用一个函数，然后停止。在这种情况下，函数应该是 y86-64 代码，用于函数(sum list)，该函数在功能上等同于图 1 中的 c sum list 函数。使用下面的三元素列表测试你的程序:

```
# Sample linked list
示例链接列表
.align 8
ele1:
. 校准 8.
ele1:
        .quad 0x00a
        四季，第 0x00a 集
        .quad ele2
        2. quad ele2
ele2:
电线 2:
        .quad 0x0b0
        四季第 0x0b0 集
        .quad ele3
        第三季，第 3 集
ele3:
第三部分:
        .quad 0xc00
        . quad 0xc00
        .quad 0
        . quad 0
```

```
1  /* linked list element */
1/* 链表元素
2  typedef struct ELE {
2 typedef struct ELE {
3          long val;
长 val;

4          struct ELE *next; 5 } *list_ptr;
下一步构造 ELE * ;

6
7  /* sum_list - Sum the elements of a linked list */
7/* Sum _ list-Sum the elements of a linked list */
8  long sum_list(list_ptr ls)
8 个长和列表(list _ ptr ls)
9  {
10         long val = 0;
长 val = 0;
11         while (ls) {
而(ls){
12                 val += ls->val;
Val + = ls-> val;
13                 ls = ls->next;
Ls = ls-> next;
14         }
15         return val;
返回 val;
16  }
17
17
18  /* rsum_list - Recursive version of sum_list */
18/* rsum _ list-sum _ list 的递归版本
19  long rsum_list(list_ptr ls)
19 个长的简历列表(list _ ptr ls)
20  {
21         if (!ls)
如果(! ls)
22                 return 0;
返回 0;
23         else {
否则
24                 long val = ls->val;
长 val = ls-> val;
25                 long rest = rsum_list(ls->next);
Long rest = rsum _ list (ls-> next) ;
26                 return val + rest;
返回 val + rest;
27         }
28  }
29
二十九

30  /* copy_block - Copy src to dest and return xor checksum of src */ 31  long copy_block(long *src, long *dest, long len)
30/* Copy _ block-将 src 复制到 dest 并返回 src */31 long Copy _ block (long * src，long * dest，long len)的 xor 校验和
32  {
    long result = 0;
长结果 = 0;
    while (len > 0) {
而(len > 0){
        long val = *src++;
长 val = * src + + ;
        *dest++ = val;
* dest + + + = val;
        result ˆ= val;
结果 = val;
        len--;
Len -- ;
    }
    return result;
返回结果;
}
```

Figure 1: **C versions of the Y86-64 solution functions.** See sim/misc/examples.c
图 1: y86-64 解决方案函数的 c 版本

rsum.ys: Recursively sum linked list elements
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，递归地求和链表的元素。这段代码应该类似于 sum.ys 中的代码，只是它应该使用一个递归求和数字列表的函数 rsum 列表，如图 1 中的 c 函数 rsum 列表所示。使用测试 list.ys 时使用的三元素列表来测试你的程序。

copy.ys: Copy a source block to a destination block
Copy.ys: 将源代码块复制到目标代码块

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) ，将一个单词块从内存的一个部分复制到另一个(非重叠区域)内存区域，计算复制的所有单词的校验和(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:
你的程序应该包含建立一个堆栈框架，调用一个函数拷贝块，然后停止的代码。函数的功能应该等同于如图 1 所示的 c 函数拷贝块。使用以下三个元素的源代码和目标代码块测试你的程序:

```
.align 8
. 对齐 8
# Source block
src:
源块 src:
        .quad 0x00a
        四季，第 0x00a 集
        .quad 0x0b0
        四季第 0x0b0 集
        .quad 0xc00
        . quad 0xc00

# Destination block dest:
目的地块 dest:

        .quad 0x111
        四人行，第 0x111 集
        .quad 0x222
        四人行，第 0x222 集
        .quad 0x333
        四人行，第 0x333 集
```

5 Part B
第二部分

You will be working in directory sim/seq in this part.
在这一部分中，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.
你在 b 部分的任务是扩展 SEQ 处理器来支持 iaddq，在家庭作业问题 4.51 和 4.52 中有描述。要添加这些指令，你需要修改文件 seq-full。Hcl，它实现了 CS: APP3e 教科书中描述的 SEQ 版本。此外，它还包含了一些常量的声明，这些常量是你的解决方案所需要的。

Your HCL file must begin with a header comment containing the following information:
HCL 文件必须以包含以下信息的头注释开头:

Your name and ID.
你的名字和身份证。

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.
对 iaddq 指令所需计算的描述。使用 CS: APP3e 文本中图 4.18 中的 irmovq 和 OPq 的描述作为指导。

Building and Testing Your Solution
构建和测试你的解决方案

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:
一旦完成了 seq-full 的修改。HCL 文件，然后你需要基于这个 HCL 文件构建一个 SEQ 模拟器(ssim)的新实例，然后测试它:

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

simulator: unix> make VERSION=full

构建一个新的模拟器。你可以使用 make 来构建一个新的 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 中指定的控制逻辑的 ssim 版本。Hcl.为了保存输入，你可以在 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 程序上测试你的解决方案。对于您的初始测试，我们推荐您在 TTY 模式下运行简单的程序，比如 asumi.yo (testing iaddq)，将测试结果与 ISA 模拟进行比较:

```
unix>  ./ssim -t ../y86-code/asumi.yo
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，那么你应该在 GUI 模式下单步调试模拟器来调试你的实现:

```
unix>  ./ssim -g ../y86-code/asumi.yo
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
使用基准程序重新测试你的解决方案。一旦你的模拟器能够正确地执行小程序，你就可以在 y86-64 基准程序上自动测试它

../y86-code:
. ./y86- 编号:

```
unix>  (cd ../y86-code; make testssim)
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，并通过将得到的处理器状态与来自高级 ISA 模拟的状态进行比较来检查正确性。注意，这些程序都没有测试添加的指令。你只需要确保你的解决方案没有为原始指令注入错误。参见文件。./y86-code/README 文件了解更多细节。

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。一旦你可以正确地执行基准测试程序，那么你应该在。./ptest.测试除了 iaddq 和 leave 之外的所有东西:

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

To test your implementation of iaddq:
为了测试 iaddq 的实现:

```
unix>  (cd ../ptest; make SIM=../seq/ssim TFLAGS=-i)
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).
关于 SEQ 模拟器的更多信息请参考讲义 CS: APP3e Guide to Y86-64 Processor Simulators (simguide.pdf)。

```
/*
/*
 * ncopy - copy src to dst, returning number of positive ints
* ncopy-copy src to dst，返回正整型数
 * contained in src array.
* 包含在 src 数组中。
 */
*/
word_t ncopy(word_t *src, word_t *dst, word_t len)
word _ t ncopy (word _ t * src， word _ t * dst， word _ t len)
{
        word_t count = 0;
Word _ t 计数 = 0;
word_t val;
个单词

        while (len > 0) {
而(len > 0){
                val = *src++;
Val = * src + + ;
                *dst++ = val;
* dst + + = val;
                if (val > 0)
如果(val > 0)
                        count++;
计数 + + ;
                len--;
Len -- ;
        }
        return count;
返回计数;
}
```

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

6 Part C
C 部

You will be working in directory sim/pipe in this part.
您将在目录 sim/管道工作在这一部分。

The ncopy function in Figure 2 copies a len-element integer array src to a non-overlapping dst, re-turning 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 元素整数数组 src 复制到一个不重叠的 dst，返回 src 中包含的正整数数目的计数。图 3 显示了 ncopy 的基线 y86-64 版本。文件管道-已满。HCL 包含 PIPE 的 HCL 代码副本，以及常量 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 的目标是让 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:
你将提交两个文件：管道满了。Hcl 和 ncopy.ys。每个文件都应该以头注释开头，注释中包含以下信息:

Your name and ID.
你的名字和身份证。

A high-level description of your code. In each case, describe how and why you modified your code.
高层次的代码描述。在每种情况下，描述你如何以及为什么修改你的代码。

Coding Rules
编码规则

You are free to make any modifications you wish, with the following constraints:
您可以在以下限制条件下自由地进行任何修改:

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.
Ys 函数必须适用于任意数组大小。您可能想通过简单地编写 64 个拷贝指令来硬连接 64 个元素数组的解决方案，但这将是一个坏主意，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。

1 ##
##

2 # ncopy.ys - Copy a src block of len words to dst.
2 # ncopy.ys-将 len 单词的 src 块复制到 dst。

3 # Return the number of positive words (>0) contained in src.
3 # 返回 src 中包含的正面词数(> 0)。

4 #

5 # Include your name and ID here.
5 # 包括你的姓名和身份证在这里。

6 #

7 # Describe how and why you modified the baseline code.
描述你是如何以及为什么修改基准代码的。

8 #

9 ##
##

10 # Do not modify this portion
10 # 不要修改此部分

11 # Function prologue.
11 # 函数开场白。

12 # %rdi = src, %rsi = dst, %rdx = len
12 #% rdi = src,% rsi = dst,% rdx = len

13 ncopy:
13 ncopy:

14
14

15 ##
##

16 # You can modify this portion
# 你可以修改这部分

17 # Loop header
# Loop header

```
          xorq %rax,%rax          # count = 0;
18        Xorq% rax,% rax         # count = 0;
          andq %rdx,%rdx          # len <=    0?
19        Andq% rdx,% rdx         # len < =   0？
                                  # if so,
          jle Done                # 如果是    goto Done:
20        结束了                  这样,       完成:
21
   Loop:  mrmovq (%rdi), %r10     # read val from src...
22 循环:  Mrmovq (% rdi) ,% r10   从 src 读取 val..。
          rmmovq %r10, (%rsi)     # ...and    store it to dst
23        Rmmovq% r10，(% rsi)    # ... 和    储存到 dst
          andq %r10, %r10         # val <=    0?
24        然后是百分之十          # val < =   0？
                                  # if so,
          jle Npos                # 如果是    goto Npos:
25        Jle npo                 这样,       转到非营利组织:
26        irmovq $1, %r10
Irmovq $1,% r10
27        addq %r10, %rax         # count++
                Addq% r10% rax              计数 + +
    Npos:
    非营利      irmovq $1,    %r10
    组织:       Irmovq $1,    % r10
                subq %r10,    %rdx          # len--
                Subq% r10,    % rdx         # len --
                irmovq $8, %r10
Irmovq $8,% r10
                addq %r10, %rdi             # src++
                Addq% r10,% rdi             # src + +
                addq %r10, %rsi             # dst++
                Addq% r10,% rsi             # dst + +
                andq %rdx,%rdx              # len > 0?
                Andq% rdx,% rdx             # len > 0 ?
                jg Loop                     # if so, goto Loop:
                Jg Loop                     # 如果是这样，去环路:
    ##################################################################
##################################################################
    # Do not modify the following section of code
# 不要修改以下代码部分
    # Function epilogue.
函数尾声。
    Done:
完成:
                ret
悔恨
    ##################################################################
##################################################################
    # Keep the following label at the end of your function
# 在函数结束时使用以下标签
    End:
完:
```

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。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.
必须使用 YIS 正确运行 ncopy.ys 函数。正确的意思是，它必须正确地复制 src 块并返回(% rax)正确数目的正整数。

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:
你的 ncopy 文件的组装版本不能超过 1000 字节。你可以使用提供的 check-len 脚本检查任何嵌入了 ncopy 函数的程序的长度。Pl:

```
unix>  ./check-len.pl < ncopy.yo
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).
您的管道完整的.hcl 实现必须通过./y86-code 和. ./ptest 中的回归测试(没有测试 iaddq 的 -i 标志)。

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，如果您认为有帮助，您可以自由地实现 iaddq 指令。您可以对 ncopy.ys 函数进行任何保留语义的转换，例如重新排序指令、用单个指令替换指令组、删除某些指令以及添加其他指令。你可能会发现阅读 CS: APP3e 的第 5.8 节中关于循环展开的内容很有用。

Building and Running Your Solution
构建和运行你的解决方案

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。我们已经提供了 gen-driver.pl 程序，可以为任意大小的输入数组生成一个驱动程序。例如，输入

```
unix>      make drivers
制造驱动程序
```

will construct the following two useful driver programs:
将构造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.
Yo: 一个小的驱动程序，用于测试包含 4 个元素的小数组上的 ncopy 函数。如果你的解决方案是正确的，那么在复制 src 数组后，这个程序会在寄存器% rax 的值为 2 时停止。

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.
Yo: 一个大型驱动程序，用于在包含 63 个元素的较大数组上测试 ncopy 函数。如果你的解决方案是正确的，那么这个程序在复制 src 数组后会在寄存器% rax 中停止，值为 31(0x1f)。

Each time you modify your ncopy.ys program, you can rebuild the driver programs by typing
每次你修改你的 ncopy.ys 程序，你可以通过输入

```
unix>      make drivers
```
制造驱动程序

Each time you modify your pipe-full.hcl file, you can rebuild the simulator by typing
每次修改管道已满的.hcl 文件时，都可以通过键入

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

If you want to rebuild the simulator and the driver programs, type
如果你想重建模拟器和驱动程序，输入

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

To test your solution in GUI mode on a small 4-element array, type
要在一个小的 4 元素数组上测试 GUI 模式下的解决方案，输入

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

To test your solution on a larger 63-element array, type
要在一个更大的 63 元素数组上测试你的解决方案，输入

```
unix>     ./psim -g ldriver.yo
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:
一旦你的模拟器正确地在这两个块长度上运行你的 ncopy.ys 版本，你将需要执行以下额外的测试:

Testing your driver files on the ISA simulator. Make sure that your ncopy.ys function works prop-erly with YIS:
在 ISA 模拟器上测试驱动程序文件。确保 ncopy.ys 函数与 YIS 正常工作:

```
unix>  make drivers
Unix > make 驱动程序
unix>  ../misc/yis sdriver.yo
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.
使用 ISA 模拟器在一定的块长度范围内测试代码。Pl 脚本生成的驱动程序文件的块长度从 0 到一定限制(默认为 65)，外加一些更大的大小。
It simulates them (by default with YIS), and checks the results. It generates a report showing the status for each block length:
它模拟它们(默认情况下使用 YIS)，并检查结果。它会生成一个报告，显示每个块长度的状态:

```
unix>  ./correctness.pl
正确性.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.
这个脚本生成的测试程序，其结果计数在不同的运行之间随机变化，因此它提供了比标准驱动程序更严格的测试。

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:
如果得到某个长度 k 的错误结果，可以生成该长度的驱动程序文件，其中包括检查代码，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 > ./gen-driver.pl-f ncopy.ys-n k-rc > driver.ys unix > make driver.yo
```

```
unix>  ../misc/yis 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 结束，其值如下:

0xaaaa : All tests pass.
0xaaaa: 所有测试通过。

0xbbbb : Incorrect count
计数错误

0xcccc : Function ncopy is more than 1000 bytes long.
0xcccc: Function ncopy 长度超过 1000 字节。

0xdddd : Some of the source data was not copied to its destination.
0xdddd: 一些源数据没有复制到其目的地。

0xeeee : Some word just before or just after the destination region was corrupted.
0xeeee: 在目标区域被破坏之前或之后的一些单词。

Testing your pipeline simulator on the benchmark programs. Once your simulator is able to cor-rectly 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。一旦你的模拟器能够正确地执行 sdriver.ys 和 ldriver.ys，你应该在 y86-64 的基准程序中进行测试。./y86-code:

```
unix>  (cd ../y86-code; make testpsim)
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
用广泛的回归测试来测试你的流水线模拟器。一旦你可以正确地执行基准测试程序，那么你应该用。./ptest.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)
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
使用管道模拟器在一定的块长度范围内测试代码。最后，你可以在管道模拟器上运行和之前在 ISA 模拟器上一样的代码测试

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

7 Evaluation
评估

The lab is worth 190 points: 30 points for Part A, 60 points for Part B, and 100 points for Part C.
这个实验室值 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 eval-uated 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 10 分。将评估每个解决方案程序的正确性，包括正确处理堆栈和寄存器，以及与 example.c 中的示例 c 函数的功能等价性。

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.
如果评分员没有发现程序 sum.ys 和 rsum.ys 中的任何错误，它们各自的 sum list 和 rsum list 函数将返回寄存器% rax 中的 sum 0xcba，则这两个程序被认为是正确的。

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.
如果分级器没有发现任何错误，则 copy.ys 程序将被认为是正确的，并且 copy 块函数返回寄存器% rax 中的 sum 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。

Part B
第二部分

This part of the lab is worth 35 points:
实验室的这部分值 35 分:

10 points for your description of the computations required for the iaddq instruction.
10 分，用于描述 iaddq 指令所需的计算。

10 points for passing the benchmark regression tests in y86-code, to verify that your simulator still correctly executes the benchmark suite.
以 y86 代码通过基准回归测试的 10 分，以验证您的模拟器仍然正确地执行基准套件。

15 points for passing the regression tests in ptest for iaddq.
在 iaddq 的 ptest 中通过回归测试 15 分。

Part C
C 部

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.
实验室的这一部分值 100 分: 如果 ncopy.ys 的代码或修改后的模拟器没有通过前面描述的任何测试，您将不会得到任何学分。

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 标题中的描述和这些实现的质量各得 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.
表现得分 60 分。要在这里获得学分，你的解决方案必须是正确的，正如之前定义的那样。也就是说，ncopy 在 YIS 中正确运行，并且管道已满。Hcl 通过了 y86 代码和 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单位(CPE)来表示函数的性能。也就是说，如果模拟代码需要 c 循环来复制一块 n 元素，那么 CPE 就是 c = n。PIPE 模拟器显示完成程序所需的循环总数。在标准 PIPE 模拟器上运行的具有 63 个元素的大型数组的 ncopy 函数的基线版本需要 897 个循环来复制 63 个元素，对于 CPE 为 897 = 63 = 14:24 的情况来说。

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
由于使用了一些循环来设置对 ncopy 的调用和在 ncopy 内部设置循环，因此您会发现，对于不同的块长度，您将得到不同的 CPE 值(通常随着 n 的增加，CPE 会下降)。因此，我们将通过计算从 1 到 64 个元素的块的 cpe 的平均值来评估函数的性能。你可以使用管道目录中的 Perl 脚本 benchmark.pl 在一定的块长度范围内运行 ncopy.ys 代码的模拟，并计算平均 CPE。只需运行命令

```
unix>  ./benchmark.pl
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。注意，这个 Perl 脚本没有检查答案的正确性。使用脚本 correctness.pl。

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:
你应该能够达到平均 CPE 小于 9:00。我们最好的版本平均 7 分 48 秒。如果你的平均 CPE 是 c，那么你在实验室这部分的得分 s 将是:

```
S  =        20  (10:5        c) ;  7:50   c    10:50
S =       8 20(10:5            C) ; 7:50 c    10:50
          > 0 ;                      c > 10:5
```

0;	C > 10:5
< 60;	C < 7:50
:	

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 参数指定一个不同的文件名。H 标志给出了命令行参数的完整列表。

8 Handin Instructions

Handin Instructions Handin 指令

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.
特定网站: 插入一个描述，说明学生应该如何提交实验室的三个部分。以下是我们在卡内基梅隆大学使用的描述。

You will be handing in three sets of files:
你将交上三组文件:

– 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。

11

– 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. To handin your files for part X, go to your archlab-handout directory and type:

请确保在每个 handin 文件顶部的注释中都包含了您的姓名和 ID。要递交第 x 部分的文件，请进入 archlab-handout 目录，输入:

```
unix>      make handin-partX TEAM=teamname
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，其中 teamname 是您的 ID。例如，交 a 部分:

```
unix>      make handin-parta TEAM=teamname
Unix > make handin-parta TEAM = teamname
```

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

copy, type `unix make handin-partX TEAM=teamname VERSION=2`

在 handin 之后，如果发现错误并希望提交修改后的副本，输入 unix make

handin-partX TEAM = teamname VERSION = 2

Keep incrementing the version number with each submission.
保持每次提交的版本号递增。

You can verify your handin by looking in
你可以通过查看来核实你的手语

```
CLASSDIR/archlab/handin-partX
CLASSDIR/archlab/handin-partX
```

You have list and insert permissions in this directory, but no read or write permissions.
您在此目录中有列表和插入权限，但没有读或写权限。

9 Hints
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.
按照设计，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:
如果你在 Unix 服务器上运行 GUI 模式，确保你已经初始化了 DISPLAY 环境变量:

```
unix>     setenv DISPLAY myhost.edu:0
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，当你在 GUI 模式下运行 psim 或 ssim 时，“ Program Code”窗口开始时是一个关闭的图标。简单的点击图标来展开窗口。

With some Microsoft Windows-based X servers, the “Memory Contents” window will not automati-cally resize itself. You’ll need to resize the window by hand.
对于一些基于 microsoftwindows 的 x 服务器，“内存内容”窗口不会自动调整自己的大小。你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 目标文件的文件，它们将以内存区段错误终止。