## 1 介绍

在这个实验中，您将学习关于流水线Y86-64处理器的设计和实现，优化它和基准程序以最大化性能。您可以对基准程序进行任何保留语义的转换，或对流水线处理器进行增强，或者两者兼而有之。当您完成实验时，您将对影响程序性能的代码和硬件之间的相互作用有深刻的认识。

实验分为三个部分，每个部分都有自己的提交要求。在第一部分中，您将编写一些简单的Y86-64程序，并熟悉Y86-64工具。在第二部分中，您将向SEQ模拟器添加两个新指令。这两个部分将为您准备第三部分，实验的核心，您将优化Y86-64基准程序和处理器设计。

## 2 实验安排

您将独自完成此实验。

任何关于作业的澄清和修订将发布在课程网页上。

## 3 实验指南

您可以从ICS课程服务器（svn://ipads.se.sjtu.edu.cn/ics-se21/[account]）中使用svn获取Architecture Lab。您可以在路径下看到一个名为“lab6”的目录。它包含多个文件：sim/是您在此实验中的工作目录，simguide.pdf是此实验提供的Y86-64模拟器的用户指南。

您可能会发现与之前的实验相比，这次提供了更多的文件和工具。幸运的是，每个目录都包含一个README文件，详细描述了布局。强烈建议在继续下一步之前阅读这些文件。

## Part A第四部分

您将在此部分的sim/misc目录中工作。

您的任务是编写三个Y86-64程序。这些程序的所需行为由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写上您的姓名和ID。您可以通过使用程序YAS汇编它们，然后使用指令集模拟器YIS运行它们来测试您的程序。例如：

unix> ./yas sum.ys

unix> ./yis sum.yo

如果您在目录中没有YAS和YIS，请先制作它们（您可能需要安装flex和bison来构建这些工具）：

unix> make yas yis

在所有Y86-64函数中，您应遵循传递函数参数、使用寄存器和使用堆栈的x86-64约定。这包括保存和恢复您使用的任何被调用者保存寄存器。

sum.ys：迭代地求和链表元素

编写一个名为sum.ys的Y86-64程序，该程序迭代地求和链表的元素。您的程序应该包括一些设置堆栈结构、调用函数然后停止的代码。在这种情况下，该函数应该是Y86-64代码的函数（sum list），其功能等同于图1中的C sum list函数。使用以下三个元素列表测试您的程序：

#示例链表

.align 8

ele1:

.quad 0x00a

.quad ele2

ele2:

.quad 0x0b0

.quad ele3

ele3:

.quad 0xc00

.quad 0

rsum.ys：递归求和链表元素

编写一个名为rsum.ys的Y86-64程序，该程序递归求和链表的元素。这段代码应该类似于sum.ys中的代码，但它应该使用一个rsum list函数，该函数递归求和一个数字列表，如图1所示的C函数rsum list。使用您用于测试sum.ys的相同三个元素列表进行测试。

1/链表元素/

2typedef struct ELE {

3int val;

4struct ELE \* next;

5}\* list\_ptr;

6

7/sum\_list-求数字链表元素的总和/

8int sum\_list(list\_ptr ls)

9{

10int val = 0;

11while(ls){

12val += ls->val;

13ls = ls->next;

14}

15return val;

16}

17

18/rsum\_list-求数字链表元素的总和，递归版本/

19int rsum\_list(list\_ptr ls)

20{

21if(!ls)

22return 0;

23else{

24int val = ls->val;

25int rest = rsum\_list(ls->next);

26return val+rest;

27}

28}

29

30/copy\_block-将src复制到dest并返回src的异或校验和/

31int copy\_block(int \* src, int \* dest, int len)

32{

33int result = 0;

34while(len > 0){

35int val

36 \*dest++ = val;

37 result ˆ= val;

38 len--;

39 }

40 return result;

41 }

Figure 1中展示了Y86-64解决方案函数的C版本。请参见sim / misc / examples.c。

copy.ys：将源块复制到目标块

编写一个程序（copy.ys），将一块字从内存的一部分复制到另一部分（非重叠区域）内存，并计算复制的所有字的异或校验和。

您的程序应包括设置堆栈帧的代码，调用函数copy block，然后停止。该函数应当与图1中所示的C函数copy block的功能等效。使用以下三个元素的源和目标块测试您的程序：

.align 8

# Source block

src:

.quad 0x00a

.quad 0x0b0

.quad 0xc00

# Destination block

dest:

.quad 0x111

.quad 0x222

.quad 0x333

## Part B:

你将在sim/seq目录中进行操作。

你在Part B的任务是扩展SEQ处理器以支持iaddq指令（在作业问题4.51和4.52中描述）。为了添加这些指令，你将修改seq-full.hcl文件，该文件实现了CS:APP3e教材中描述的SEQ版本。此外，它还包含了一些常量的声明，这些常量在你的解决方案中需要使用。

你的HCL文件必须以包含以下信息的头注释开始：

• 你的姓名和ID。

• 描述iaddq指令所需的计算。使用CS:APP3e文本中图4.18中的irmovq和OPq的描述作为指南。

构建和测试你的解决方案

一旦你完成了修改seq-full.hcl文件，你将需要基于此HCL文件构建一个新的SEQ模拟器（ssim），然后对其进行测试：

• 构建一个新的模拟器。你可以使用make构建一个新的SEQ模拟器：

unix> make VERSION=full

这将构建一个使用你在seq-full.hcl中指定的控制逻辑的ssim版本。为了节省打字，你可以在Makefile中指定VERSION=full。

• 在简单的Y86-64程序上测试你的解决方案。对于你的初始测试，我们建议运行像asumi.yo（测试iaddq）这样的简单程序，以TTY模式运行，将结果与ISA模拟进行比较（如果你还没有构建asumi.yo，请先构建它）：

unix> ./ssim -t ../y86-code/asumi.yo

如果ISA测试失败，那么你可以通过单步调试GUI模式下的模拟器来尝试调试你的实现（有关如何构建GUI模式模拟器的详细信息，请参见第10节）：

unix> ./ssim -g ../y86-code/asumi.yo

• 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型程序，那么你就可以自动在../y86-code中的Y86-64基准程序上测试它：

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

这将在基准程序上运行ssim，并通过将处理器状态与高级ISA模拟的状态进行比较来检查正确性。请注意，这些程序中没有一个测试添加的指令。你只需确保你的解决方案未为原始指令注入错误。有关更多详细信息，请参见../y86-code/README文件。

• 执行回归测试。一旦你能够正确地执行基准程序，那么你应该在../ptest中运行广泛的回归测试集。要测试除iaddq之外的所有内容：

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

要测试你的iaddq实现：

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

有关SEQ模拟器的更多信息，请参阅CS：APP3e指南Y86-64处理器模拟器（simguide.pdf）。

## 6 第C部分

您将在此部分的sim / pipe目录中工作。

图2中的ncopy函数将长度为len的整数数组src复制到不重叠的dst，并返回src中包含的正整数数量。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。

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 }

图2：ncopy函数的C版本。请参见sim / pipe / ncopy.c。

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

您将提交两个文件：pipe-full.hcl和ncopy.ys。每个文件都应该以一个带有以下信息的头注释开始：

•您的姓名和ID。

•您的代码的高级描述。在每种情况下，描述您如何以及为什么修改了您的代码。

编码规则

您可以自由地进行任何修改，但有以下限制：

•您的ncopy.ys函数必须适用于任意数组大小。您可能会想为64个元素的数组硬编码您的解决方案，只需编写64个复制指令，但这是一个坏主意，因为我们将根据其在任意数组上的性能对您的解决方案进行评分。

•您的ncopy.ys函数必须在YIS上正确运行。正确的意思是它必须正确复制src块并（在％rax中）返回正确数量的正整数。

•您的ncopy文件的汇编版本不能超过1000个字节。您可以使用提供的check-len.pl脚本检查任何嵌入了ncopy函数的程序的长度：

unix> ./check-len.pl < ncopy.y

1 ##################################################################

2 # ncopy.ys - 将长度为len的src块复制到dst中。

3 # 返回src中包含的正整数（>0）数量。

4 #

5 # 在此处包含您的姓名和ID。

6 #

7 # 描述您如何以及为什么修改了基线代码。

8 #

9 ##################################################################

10 # 请勿修改此部分

11 # 函数序言。

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

13 ncopy:

14

15 ##################################################################

16 # 您可以修改此部分

17 # 循环头

18 xorq %rax,%rax # count = 0;

19 andq %rdx,%rdx # len <= 0?

20 jle Done # 如果是，则跳转到Done：

21

22 Loop: mrmovq (%rdi), %r10 # 从src读取val...

23 rmmovq %r10, (%rsi) # ...并将其存储到dst中

24 andq %r10, %r10 # val <= 0?

25 jle Npos # 如果是，则跳转到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 # 如果是，则跳转到Loop：

35 ##################################################################

36 # 请勿修改以下代码部分

37 # 函数结尾。

38 Done:

39 ret

40 ##################################################################

41 # 将以下标签保留在函数末尾

42 End:

好的，以下是严格的翻译：

图3：ncopy函数的基准Y86-64版本。请参见sim/pipe/ncopy.ys。

• 您的pipe-full.hcl实现必须正确。它应该通过../y86-code和../ptest中的回归测试（见下文）。

除此以外，您可以自由地实现iaddq指令，如果您认为这会有所帮助。您可以进行任何保留语义的转换，例如重新排序指令，用单一指令替换一组指令，删除某些指令以及添加其他指令。您可能会发现在CS:APP3e的第5.8节中了解有关循环展开的内容非常有用。

构建和运行您的解决方案

为了测试您的解决方案，您需要构建一个调用您的ncopy函数的驱动程序。我们已经为您提供了gen-driver.pl程序，该程序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

将构建以下两个有用的驱动程序：

• sdriver.yo：一个小型驱动程序，用于在具有4个元素的小数组上测试ncopy函数。如果您的解决方案正确，则在复制src数组后，此程序将以寄存器%rax中的值2停止。

• ldriver.yo：一个大型驱动程序，用于在具有63个元素的更大数组上测试ncopy函数。如果您的解决方案正确，则在复制src数组后，此程序将以寄存器%rax中的值31（0x1f）停止。

每次修改您的ncopy.ys程序时，您都可以键入

unix> make drivers

以重新构建驱动程序。

每次修改pipe-full.hcl文件时，您都可以键入

unix> make psim VERSION=full

以重新构建模拟器。

要在GUI模式下在小型4元素数组上测试您的解决方案，请键入

unix> ./psim -g sdriver.yo

要在更大的63元素数组上测试您的解决方案，请键入

unix> ./psim -g ldriver.yo

一旦您的模拟器在这两个块长度上正确运行您的ncopy.ys版本，您将想要执行以下附加测试：

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

unix> make drivers

unix> ../misc/yis sdriver.yo

• 在ISA模拟器上测试各种块长度的代码。Perl脚本correctness.pl会生成具有从0到某个限制（默认为65）的块长度以及一些更大大小的驱动程序文件。它模拟它们（默认情况下使用YIS），并检查结果。它生成一个报告，显示每个块长度的状态：.

unix> ./correctness.pl

这个脚本会生成测试程序，其中结果数量会随机地从一次运行到另一次运行而变化，因此它提供了比标准驱动程序更严格的测试。如果您在某个长度 K 上获得了不正确的结果，则可以为该长度生成一个包括检查代码和结果随机变化的驱动程序文件：

unix> ./gen-driver.pl -f ncopy.ys -n K -rc > driver.ys

unix> make driver.yo

unix> ../misc/yis driver.yo

程序将以寄存器 %rax 具有以下值结束：

0xaaaa：所有测试都通过。

0xbbbb：结果不正确

0xcccc：函数 ncopy 长度超过 1000 字节。

0xdddd：某些源数据未被复制到其目的地。

0xeeee：目标区域前面或后面的某个字被损坏。

• 使用基准程序测试流水线模拟器。一旦您的模拟器能够正确地执行 sdriver.ys 和 ldriver.ys，您应该测试它对于 ../y86-code 中的 Y86-64 基准程序：

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

这将在基准程序上运行 psim 并将结果与 YIS 进行比较。

• 使用广泛的回归测试测试您的流水线模拟器。一旦您能够正确地执行基准程序，那么您应该使用 ../ptest 中的回归测试检查它。例如，如果您的解决方案实现了 iaddq 指令（不是必须的），那么：

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

• 使用流水线模拟器在一系列块长度上测试您的代码。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 评估

本实验总共190分：A部分30分，B部分60分，C部分100分。

A部分

A部分总共30分，每个 Y86-64 解决方案程序10分。每个解决方案程序都将被评估其正确性，包括对堆栈和寄存器的正确处理以及与 examples.c 中的示例 C 函数的功能等价性。

如果评分人员没有发现错误，并且它们各自的 sum list 和 rsum list 函数返回寄存器 %rax 中的和 0xcba，则认为程序 sum.ys 和 rsum.ys 是正确的。

如果评分人员没有发现错误，并且复制块函数返回寄存器 %rax 中的校验和 0xcba，将三个 64 位值 0x00a、0x0b0 和 0xc00 复制到以 dest 开头的 24 个字节中

部分 A：

1.如果评分人员未发现sum.ys和rsum.ys中存在任何错误，且它们各自的sum list和rsum list函数都在%rax寄存器中返回值为0xcba的总和，则认为这两个程序是正确的。

2.如果评分人员未发现copy.ys中存在任何错误，且copy block函数在%rax寄存器中返回检验和0xcba，将三个64位值0x00a、0x0b0和0xc00复制到从地址dest开始的24个字节中，并且不破坏其他内存位置，则认为该程序是正确的。

部分 B：

1.用于iaddq指令所需计算的描述值得10分。

2.通过在y86-code中运行基准回归测试，以验证您的模拟器仍能正确执行基准测试套件，值得20分。

3.通过在ptest中运行回归测试来获得iaddq的30个分数。

部分 C：

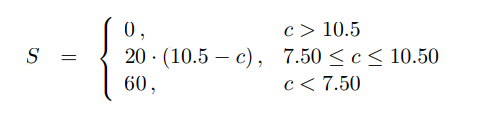
1.如果您的ncopy.ys代码或修改后的模拟器未能通过先前描述的任何测试，则不会获得任何学分。

2.ncopy.ys和pipe-full.hcl头部的描述值得20分，以及这些实现的质量。

3.性能值得60分。要获得此处的学分，您的解决方案必须是正确的，如前所定义。也就是说，ncopy在您的管道实现以及汇编版本的YIS中都可以正确运行，ncopy文件的汇编版本长度不得超过1000字节，pipe-full.hcl在y86-code和ptest中通过所有测试。

4.我们将以每个元素的周期数（CPE）为单位来表示您的函数的性能。也就是说，如果模拟的代码需要C个周期来复制N个元素的块，则CPE为C/N。PIPE模拟器显示完成程序所需的总周期数。在具有63个元素的大数组上运行标准PIPE模拟器上的ncopy函数的基线版本需要897个周期来复制63个元素，因此CPE为897/63 = 14.24。

5.由于某些周期用于调用ncopy和在ncopy内设置循环，因此您会发现对于不同的块长度，您会获得不同的CPE值（通常CPE随N的增加而下降）。因此，我们将通过计算从1到64个元素的块的CPE的平均值来评估您的函数的性能。您可以使用pipe目录中的Perl脚本benchmark.pl运行您的ncopy.ys代码在一系列块长度上的模拟，并计算平均CPE。只需运行命令unix> ./benchmark.pl，就可以看到发生了什么。举例来说，基准版本的ncopy函数的CPE值范围在29.00至14.27之间，平均值为15.18。请注意，这个Perl脚本不检查答案的正确性，请使用correctness.pl脚本进行检查。你应该能够达到平均CPE小于9.00的水平。我们最好的版本平均值为7.48。如果你的平均CPE为c，则你在此部分实验中的得分将为:



默认情况下，benchmark.pl和correctness.pl会编译并测试ncopy.ys文件。使用-f参数可以指定其他文件名。-h标志可列出命令行参数的完整列表。

第8部分C的评分基于TA的计算机。如果您想了解最终得分，请提交ncopy.ys和pipe-full.hcl到svn。得分将发布在以下网址上：http://ipads.se.sjtu.edu.cn/courses/ics/labs/archlab/arch-score.html 注意：网页不会每天更新。

第9部分提交说明： 您将提交三组文件：- 部分A：sum.ys、rsum.ys和copy.ys（不要忘记使用svn add将它们放入版本控制）。

第B部分：seq-full.hcl。

第C部分：ncopy.ys和pipe-full.hcl。 确保在每个提交文件的顶部的注释中包含了您的姓名和ID。

第10部分提示：

如果要执行的文件不是有效的Y86-64对象文件，则psim和ssim模拟器会以分段错误终止。

您可能需要安装flex和bison。sudo apt-get install flex bison

安装Tcl/Tk

在GUI模式下运行simulator可以可视化地显示程序在运行时的执行情况。它不仅有助于调试您在本实验中的实现，而且还增强了您对我们在本章中学到的内容的理解。要构建GUI模式模拟器，可以参考构建模拟器的Makefile的详细信息。您的系统应该已安装Tcl/Tk 8.5及其头文件和库。

运行以下命令以安装Tcl/Tk 8.5及其头文件和库文件：

sudo apt-get install tcl8.5-dev tk8.5-dev

Debian 10从apt源中删除了tcl8.5-dev和tk8.5-dev软件包，因此上述命令将失败。

如果您使用的是Debian 10，则需要以root身份执行以下命令：

apt\_source="deb http://mirrors.ustc.edu.cn/debian stretch main contrib"

echo $apt\_source > /etc/apt/sources.list.d/stretch.list

echo "APT::Default-Release "buster";" > /etc/apt/apt.conf.d/99stretch

apt update

apt -qqy install libx11-dev libxext-dev libxft-dev libxss-dev dpkg-dev

apt -qqy -t stretch install tcl8.5-dev tk8.5-dev

安装完Tcl/Tk后，您需要取消注释Makefile中与GUI模式相关的行以构建GUI模式仿真器。

GUIMODE=-DHAS\_GUI

TKLIBS=-L/usr/lib -ltk8.5 -ltcl8.5

TKINC=-isystem /usr/include/tcl8.5

如果您有任何问题，请发送电子邮件至Ruizhe Tong（tongruizhe@163.com）。