CS349,2015 年夏天
优化管道处理器的性能
分配日期：6 月 6 日，到期日期：6 月 21 日，晚上 11：59

HarryBovik(bovik@cs.cmu.edu)是这个任务的负责人。

1 介绍

在这个实验室中，您将了解一个流水线化的 Y86-64 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。您可以对基准测试程序进行任何语义保留转换，或者对流水线处理器进行增强，或者两者都可以。当您完成实验室后，您将对影响程序性能的代码和硬件之间的交互感到非常感谢。

实验室分为三个部分，每个部分都有自己的把手。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，您将使用一个新的指令来扩展 SEQ 模拟器。这两部分将为您为实验室的核心 C 部分做准备，在那里您将优化 Y86-64 基准程序和处理器设计。

2 物流

你将独自在这个实验室里工作。

对作业的任何澄清和修改都将张贴在课程网页上。

3 处理说明

特定于网站：在这里插入一段，解释学生应该如何下载 archlab-handout.tar 文件。

1. 首先，将文件 archlab-handout.tar 复制到您计划在其中执行工作的（受保护的）目录中。
2. 然后发出命令：tarxvfarchlab-handout.tar。这将导致以下文件被解压缩到目录中：自述文件、Make 文件、sim.tar、archlab.pdf 和 simguide.pdf。
3. 接下来，给出命令 tarxvfsim.tar。这将创建目录 sim，其中包含你的 Y86-64 工具的个人副本。您将在这个目录内完成您所有的工作。

4. 最后，更改到 sim 目录，并构建 Y86-64 工具：

Unix>cd 模拟
Unix>使清洁；使

4 A 部分

您将在本部分的目录 sim/misc 中工作。

您的任务是编写和模拟以下三个 Y86-64 程序。这些程序的所需行为由示例中的 C 函数定义。一定要在每个程序的开头把你的名字和 ID 放在一个评论中。您可以通过首先用程序 YAS 组装程序，然后使用指令集模拟器 YIS 来测试程序。

在所有的 Y86-64 函数中，您应该遵循 x86-64 约定来传递函数参数、使用寄存器和使用 6 堆栈。这包括保存和恢复您使用的任何调用保存寄存器。

交替和链接列表元素

编写一个 Y86-64 程序和。ys 迭代和链接列表的元素。您的程序应该包含一些设置堆栈结构、调用一个函数，然后停止的代码。在这种情况下，函数应该是函数（64）的 Y86 代码，该函数等同于图 1 中的 C 函数。使用以下三元素列表测试程序：

```
#示例链接列表
，对齐 8
ele1：
            。quad0x00a
            。四等位基因 2
ele2：
            。quad0x0b0
            。四等位基因 3
ele3：
            。quad0xc00
            。四分之一
```

```
/*链接列表            元素*/
类型定义的结构 ELE;
      结构 ELE*下一个; }*list_ptr;

/*sum_list 一和链接列表*/的元素
长 sum_list(list_ptrls)

      长 val=0; 而(ls){
            +=->; =->;
      }
      返回 val;


/*rsum_list 一递归版本的 sum_list*/longrsum_list(list_ptrl)
{
      如果(! ls)返回 0;
      其他{
            长 val=->val;
            长休息=rsum_list(ls->next); 返回 val+休息;
      }
}

/*copy_block 一复制 src 到 dest，并返回 src*/长 copy_block（长、长、长伦）的 xor 校验和
{
      长结果=0;
      而(len>0){
            长 val=*src++;
            *ᵡ++=val;
            结果 "=val;
            len——;
      }
      返回结果;
}
```

图 1：Y86-64 解决方案函数的 C 版本。参见 sim/misc/示例。c

递归求和链接列表元素

编写一个 Y86-64 程序 rsum.ys，递归地和链接列表的元素。这个代码应该类似于代码 sum.ys，除了它应该使用一个递归求和数字列表，如图 1 中的 C 函数 rsum_list 所示。使用用于测试列表的相同的三元素列表来测试程序。

复制。：将源块复制到目标块

编写一个程序（copy.ys），该程序将单词块从内存的一部分复制到另一个（非重叠区域）区域，计算复制的所有单词的校验和(Xor)。

您的程序应该由设置堆栈框架、调用函数 copy 上 lock，然后停止的代码组成。该函数应该在功能上等同于图 1 所示的 C 函数 copy 上 lock。使用以下三元素源块和目标块测试程序：

```
，对齐 8
#   源块
src：
            。quad0x00a
            。quad0x0b0
            。quad0xc00

#   目标块
dest：
            。四 x111
            。四 x222
            。四 x333
```

5 B 部分

您将在本部分的目录 sim/seq 中工作。

您在 B 部分中的任务是扩展 SEQ 处理器以支持 iaddq，在家庭作业问题 4.51 和 4.52 中描述。要添加此说明，您将修改文件 seq-full.hcl，该文件实现了 CS：APP3e 教科书中描述的 SEQ 版本。此外，它还包含了您的解决方案所需要的一些常量的声明。

HCL 文件必须以包含以下信息的标题注释开头：

- 您的名称和 ID。
- 对 iaddq 指令所需的计算方法的描述。使用 CS：APP3e 文本中图 4.18 中对 irmovq 和 OPq 的描述作为指南。

构建和测试您的解决方案

一旦您完成了对 seq-full.hcl 文件的修改，那么您将需要基于这个 HCL 文件构建一个 SEQ 模拟器(ssim)的新实例，然后进行测试：

- 建立一个新的模拟器。您可以使用 make 来构建一个新的 SEQ 模拟器：

  Unix>使版本=完整

  这将构建一个使用您在 seq-full.hcl 中指定的控制逻辑的 ssim 版本。要保存键字，您可以在=文件中完整地分配版本=。

- 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，我们建议在 TTY 模式下运行简单的程序，如 asumi.yo(测试 iaddq)，并将结果与 ISA 模拟进行比较：

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

如果 ISA 测试失败，那么您应该通过在 GUI 模式下单步使用模拟器来调试实现：

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

- 使用基准测试程序重新测试解决方案。一旦你的模拟器能够正确地执行小程序，那么你就可以自动在 Y86-64 基准测试程序上进行测试。/y86 代码：

  Unix>(cd../y86-代码；进行测试)

  这将在基准测试程序上运行 ssim，并通过将结果的处理器状态与来自高级 ISA 模拟的状态进行比较来检查其正确性。请注意，这些程序都没有测试所添加的指令。您只需确保解决方案没有为原始指令注入错误。详情请参阅文件../y86-代码/自述文件。

- 进行回归检验。一旦您可以正确地执行基准测试程序，那么您就应该在../ptest 中运行广泛的回归测试集。要测试除 iaddq 之外的所有内容并离开：

  Unix>(cd../ptest；制造 SIM=../seq/ssim)

  要测试 iaddq 的实现：

  unix>(cd../ptest；使 SIM=../seq/ssimTFLAGS=-i)

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

```
1 /*
    * ncopy 一拷贝 src          到 dst，返回正整数数组的数量。
    * 包含在 src*/中
  word_tncopy(word_t{           *src，word_t*dst，word_t 伦)

        word_t 计数=0;
        word_tval;
9
10       而(len>0){val=*src++;
11      *dst++=val；如果(val>0)
12                      计数++;
13            len——;
14
15
16      }
17      返回计数;
18
```

图 2：ncopy 函数的 C 版本。请参阅 sim/管道/ncope.c。

6C 部分

您将在本部分的目录 sim/管道中工作。

图 2 中的 ncopy 函数将一个 len 元素整数数组 src 复制到一个不重叠的 dst 中，返回 src 中包含的正整数数的计数。图 3 显示了 ncopy 的基线 Y86-64 版本。文件 pipe-full.hcl 包含 PIPE 的 HCL 代码的副本，以及常量值 IIADDQ 的声明。

您在 C 部分中的任务是修改 ncopy.ys 和 pipe-full.hcl，目的是使 ncopy.ys 尽可能快地运行。

你将提交两个文件：pipe-full.hcl 和 ncopy.ys。每个文件都应该以包含以下信息的标题注释开头：

- 您的名称和 ID。
- 对代码的高级描述。在每种情况下，都要描述如何以及为什么要修改代码。

编码规则

您可以自由地做任何您想要的修改，但有以下限制：

ncopy.ys 函数必须适用于任意数组大小。您可能会试图通过简单地编码 64 个复制指令来硬连接 64 个元素阵列的解决方案，但这将是一个坏主意，因为我们将根据它在任意数组上的性能对您的解决方案进行分级。

```
 # 复制一个返回的数字       src 块的伦的单词对 dst。
 #                          src 中包含的阳性单词（0）。
 #
 #  包括您的名称            和 ID 在这里。
 #
 ########################################################################
#  描述您如何以及为什么要修改基线代码。
#
########################################################################
#  不要修改此部分
#  函数序言。
#  %rdi=标准，%rsi=，%rdx==
ncopy：

########################################################################
#您可以修改此部分
          #循环头
                    xorq%rax，%rax                # 计数=0；
                    andq%rdx，%rdx                # Len<=0?
                    杰尔干了                      # 如果是，goto     完成：

             循环：     mrmovq(%rdi)，%r10         # 从 src 阅读 val..。
                    rmmovq%r10，(%rsi)            # …和商店          它到 dst
                    andq%r10，%r10                # val<=0?
                    jle Npos                      # 如果是，goto     Npos：
                    irmovq$1,%r10addq%r10,%rax
                                                  # count++
             Npos：
                    irmovq$1,%r10subq%r10,%rdx    # len——
                    irmovq$8，%r10
                    addq%r10，%rdi                # src++
                    addq%r10，%rsi                # dst++
                    andq%rdx，%rdx                # 伦>0?
                    jg 循环                       # 如果是，goto     循环：
########################################################################
#  不要修改以下代码的某些部分
#  函数后语。
完成：
          ret
########################################################################
#  在功能结束时保留以下标签
结束：
```

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

- ncopy.ys 函数必须使用 YIS 正确运行。通过正确地说，我们的意思是它必须正确地复制 src 块，并(以%rax 为单位)返回正确数量的正整数。
- ncopy 文件的组装版本的长度不能超过 1000 字节。您可以使用提供的脚本 check-len.pl 检查嵌入 ncopy 函数的任何程序的长度：

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

- pipe-full.hcl 实现必须通过../y86-code 和../ptest（没有测试 iaddq 的-i 标志）中的回归测试。

除此之外，如果您认为这有帮助，您可以免费实现 iaddq 指令。您可以对 ncopy.ys 函数进行任何语义保留转换，例如重新排序指令、用单个指令替换指令组、删除一些指令和添加其他指令。您可能会发现在 CS：APP3e 的 5.8 节中阅读关于循环展开很有用。

构建和运行您的解决方案

为了测试解决方案，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，它为任意大小的输入数组生成一个驱动程序。例如，键入

Unix>制造驱动程序

将构建以下两个有用的驱动程序和程序：

- sdriver.yo：一个小型驱动程序，它在包含 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 程序数组后，此程序将以注册器%rax 中的值为 2 停止。
- ldriver.yo：一个大型驱动程序，它在包含 63 个元素的较大数组上测试 ncopy 函数。如果您的解决方案是正确的，那么在复制该程序数组后，该程序将在注册器%rax 中停止，值为 31(0x1f)。

每次修改 ncopy.ys 程序时，都可以通过键入重新构建驱动程序

Unix>制造驱动程序

每次修改 pipe-full.hcl 文件时，您都可以通过键入来重新构建模拟器

Unix>使 psim 版本=完整

如果您想重新构建模拟器和驱动程序程序，请键入

Unix>使版本=完整

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

要在较大的 63 元素数组上测试解决方案，请键入

unix>./psim-g ldriver.yo

一旦模拟器在这两个块长度上正确运行 ncopy.ys 版本，您将需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 函数与 YIS 正常工作：*

  Unix>制造驱动程序
  unix>../misc/yis sdriver.yo

- 使用 ISA 模拟器在块长度范围内测试代码。Perl 脚本 correctness.pl 生成的驱动程序文件的块长度从 0 到某些限制（默认为 65），以及一些更大的大小。
  它会模拟它们（默认情况下使用 YIS），并检查结果。它将生成一个报告，显示每个块长度的状态：

  unix>./correctness.pl

  这个脚本生成的测试程序中，结果计数因不同运行而随机变化，因此它提供了比标准驱动程序更严格的测试。

  如果您对某些长度为 K 得到不正确的结果，您可以生成一个驱动程序文件，其中包括检查代码，其中结果随机变化：

  unix>./gen-driver.pl-f ncopy.ys-n K-rc>driver.ys
  Unix>制作驱动程序。yo
  unix>../misc/yis driver.yo

  该程序将以具有以下值的注册器%rax 结束：

  0xaaaa：所有测试均能通过。

  0xbbbb：计数不正确

  0xcccc：函数 ncopy 的长度超过 1000 字节。

  0xdddd：一些源数据没有复制到其目标。

  0xeeee：就在目的地区域损坏之前或之后的一些词。

- 在基准测试程序上测试管道模拟器。一旦您的模拟器能够正确地执行 sdriver.ys 和 Idriver.ys，您就应该对..中的 Y86-64 基准测试程序进行测试。/y86-代码：

  Unix>(cd../y86-代码；使测试 psim)

  这将在基准程序上运行 psim，并与 YIS 进行比较。

- 使用广泛的回归测试来测试管道模拟器。一旦您能够正确地执行基准测试程序，那么您就应该使用../ptest 中的回归测试来检查它。例如，如果您的解决方案实现了 iaddq 指令，那么

  unix>(cd../ptest；使 SIM=../pipe/psimTFLAGS=-i)

- 最后，使用管道模拟器在块长度范围上测试代码，您可以在管道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 评价

实验室值 190 分：A 部分 30 分，B 部分 60 分，C 部分 100 分。

合作伙伴

A 部分值 30 分，每个 Y86-64 解决方案程序值 10 分。将评估每个解决方案程序的正确性，包括对堆栈和寄存器的正确处理，以及与示例中的 C 函数的函数等价性。

如果评分程序没有发现 ys 和 ys 和 ys，则被认为是正确的，并且它们各自的和列表和 rsum_list 函数返回寄存器%rax 中的 0xcba。

如果程序没有发现任何错误，并且 copy 上 lock 函数以%rax 中的函数返回 0xcba，将三个 64 位值 0x00a、0x0b 和 0xc 复制到从地址 dest 开始的 24 字节，并且不会损坏其他内存位置。

合作伙伴

这部分实验室值 35 分：

- 描述 iaddq 指令所需计算的 10 点。
- 通过 y86-代码中的基准测试回归测试的 10 分，以验证模拟器仍然正确地执行基准测试套件。
- 通过 iaddq 检验回归检验的 15 分。

C 部分

实验室的这部分值 100 分：如果您的 ncopy.ys 或修改后的任何测试失败，您将不会获得任何学分。

- 在 ncopy.ys 和 pipe-full.hcl 的标题以及这些实现的质量中各得到 20 分。
- 性能 60 分。要在这里获得信用，您的解决方案必须是正确的，如前面定义的。也就是说，ncopy 使用 YIS 正确运行，pipe-full.hcl 通过了 y86 代码和 ptest 中的所有测试。

  我们将以每个元素的循环(CPE)为单位来表示您的功能性能。Thatis，如果模拟代码需要 C 循环来复制 N 个元素块，那么 CPE 是 C/N。管道模拟器显示完成程序所需的循环总数。运行在标准 PIPE 模拟器上的标准 POPE 函数的基线版本需要 897 个周期来复制 63 个元素，而 CPE 为 897/63=14.24。

  由于某些循环用于设置 ncopy 调用并在 ncopy 中设置循环,您会发现不同块长度获得不同的 CPE 值(通常 CPE 会随着 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。您可以使用管道目录中的 Perl 脚本 benchmark.pl 在一个块长度范围内运行 ncopy.ys 代码的模拟，并计算平均 CPE。只需运行该命令即可

  unix>./*benchmark.pl*

  看看会发生什么。例如，ncopy 函数的基线版本的 CPE 值在 29.00 到 14.27 之间，平均为 15.18。注意，这

个 Perl 脚本不检查答案的正确性。为此，请使用脚本 correctness.pl。

你应该能够达到一个平均 CPE 小于 9.00。我们的最佳版本平均是 7.48。如果你的平均 CPE 是 c，那么你在这部分实验室的分数 S 将是：

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

默认情况下，benchmark.pl 和 correctness.pl 会编译和测试 ncopy.ys。使用-f 参数可以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 处理程序说明

具体地点：插入一个说明，说明学生应该如何处理实验室的三个部分。以下是我们在 CMU 上使用的描述。

- 您将提交三套文件：
  - 一 A 部分：sum.ys，rsum.ys，和复制。ys。
  - 一 PartB：seq-full.hcl.
  - 一 C 部分：ncopy.ys 和 pipe-full.hcl。
- 确保您已经在每个手动文件顶部的注释中包含了您的名称和 ID。
- 要上交第 X 部分的文件，请转到您的原始实验室施舍目录并键入：

  unix>制作手工零件团队=团队名称

  其中 X 是 a、b 或 c，其中团队名是您的 ID。例如，要交入第 A 部分：

  unix>制作了合作团队=团队名称

- 插入操作后，如果发现错误并希望提交修改后的副本，请键入

  unix 制作手工零件团队=团队名版本=2

  在每次提交时，请不断增加版本号。

- 你可以通过查看来验证你的手机

  CLASSDIR/archlab/handin-partX

  您在此目录中有列表和插入权限，但没有读写权限。

9 铰链

- 根据设计，sdriver.yo 和 ldriver.yo 都小到可以在 GUI 模式下进行调试。我们发现在 GUI 模式下调试最容易，并建议您使用它。
- 如果您在 Unix 服务器上以 GUI 模式运行，请确保您已经初始化了显示环境变量：

    unix>天线显示 myhost.edu：0

- 对于一些 X 服务器，当您在 GUI 模式下运行 psim 或 ssim 时，“程序代码”窗口将作为一个关闭的图标开始使用。只需单击该图标即可展开该窗口。
- 对于一些基于微软 windows 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的大小。
- 如果您要求 psim 和 ssim 模拟器执行一个不是有效的 Y86-64 对象文件，那么这些模拟器就会以分割故障终止。