

# archlab报告

誓死和这个实验斗争到底。

## Table of Contents

前言：何出此言？	-----
partA	-----
sum.ys	-----
rsum.ys	-----
copy.ys	-----
partB	-----
partC	-----
基础修改	-----
加入iaddq指令	-----
使用iaddq指令进行改进	-----
根据分支预测改进	-----
循环展开	-----

## 前言：何出此言？

为什么说斗争到底这句话呢？因为一看到这个实验的文档和框架，就大致觉得这不是个简单的实验，涉及到晦涩的汇编代码、编译相关的文法检查、cpu模拟器等等，想全部搞懂要花费不少精力。

要命的是，我拿到手的handout还有bug。比如刚拿到手编译Y86-64 tools，不成功，原因并不是网上说的缺少tk库等，我碰到的问题主要是源码中缺少的 `extern` 标志。经过几番查询，发现解决[方案](#)。

在需要的gcc的子文件夹，在makefile的flags中加入 `-fcommon`。

此后回到sim文件夹，make成功。

关于图形界面：seq比较简单不需要，流水线版本调试还是需要GUI的。实验框架使用的tk库版本较老，为8.5版本，在最新的8.6版本中，有一个result域已经被弃用，还是要使用匹配版本的tk库。参考[文章](#)安装好tk8.5，再将头文件 `<tk.h>` 更改为 `<tk8.5/tk.h>`，然后编译，出现 `undefined reference to 'matherr'` 问题，去 `psim.c` 中手动注释掉matherr的两行代码就好。至此环境配置结束。

## partA

## sum.ys

链表求和，仿照中文版教材(3e)第252页，配置内存、堆栈和主函数，再完成sum函数，源文件内容：

```
.pos 0
irmovq stack, %rsp
call main
halt
.align 8
ele1:
.quad 0x00a
.quad ele2
ele2:
.quad 0x0b0
.quad ele3
ele3:
.quad 0xc00
.quad 0
main:
irmovq ele1, %rdi
call sum
ret
sum:
# init
irmovq $0, %rax
LH:
andq %rdi, %rdi
je E
mrmovq $0(%rdi), %rdx
addq %rdx, %rax
mrmovq $8(%rdi), %rdi
jmp LH
E:
ret
.pos 0x200
stack:
```

结果：

```
[lixiaiqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/misc] - [2023-03-17 04:06:11]
[0] <git:(main 69904e8+ )> ./yis sum.ys
Stopped in 28 steps at PC = 0x13. Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax: 0x0000000000000000      0x0000000000000000cba
%rdx: 0x0000000000000000      0x0000000000000000c00
%rsp: 0x0000000000000000      0x0000000000000000200

Changes to memory:
0x01f0: 0x0000000000000000      0x00000000000000005b
0x01f8: 0x0000000000000000      0x00000000000000000013
[lixiaiqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/misc] - [2023-03-17 04:06:12]
[0] <git:(main 69904e8+ )> 
```

16:06

Friday, March 17

返回值 %rax 是 0cba , 正确。

## rsum.ys

类似上述，需要注意递归调用时寄存器的备份。

检查寄存器 %rax 的值，正确。

注意到，因为递归调用，所以堆栈变化比循环实现更多。

```
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/misc] - [2023-03-17 04:08:47]
[0] <git:(main 69904e8* ) > ./ys rsum.ys
Stopped in 40 steps at PC = 0x13. Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rax: 0x0000000000000000      0x00000000000000ba
%rbx: 0x0000000000000000      0x000000000000000a
%rsp: 0x0000000000000000      0x0000000000000040
%rdi: 0x0000000000000000      0x0000000000000018

Changes to memory:
0x03c0: 0x0000000000000000      0x000000000000007c
0x03c8: 0x0000000000000000      0x0000000000000038
0x03d0: 0x0000000000000000      0x0000000000000007c
0x03d8: 0x0000000000000000      0x0000000000000028
0x03e0: 0x0000000000000000      0x000000000000007c
0x03e8: 0x0000000000000000      0x0000000000000018
0x03f0: 0x0000000000000000      0x000000000000005b
0x03f8: 0x0000000000000000      0x0000000000000013

[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/misc] - [2023-03-17 04:08:54]
[0] <git:(main 69904e8* ) > 
```

16:08

Friday, March 17

## copy.ys

需要实现数组拷贝、返回所有元素的异或。

目标数组正确赋值、返回结果正确。

```
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/misc] - [2023-03-17 04:09:37]
[0] <git:(main 69904e8* ) > ./ys copy.ys
Stopped in 44 steps at PC = 0x13. Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax: 0x0000000000000000      0x00000000000000ba
%rcx: 0x0000000000000000      0x0000000000000001
%rsp: 0x0000000000000000      0x0000000000000040
%rsi: 0x0000000000000000      0x0000000000000048
%rdi: 0x0000000000000000      0x0000000000000030

Changes to memory:
0x0030: 0x0000000000000011      0x00000000000000a
0x0038: 0x0000000000000022      0x00000000000000b0
0x0040: 0x0000000000000033      0x00000000000000c0
0x03f0: 0x0000000000000000      0x000000000000006f
0x03f8: 0x0000000000000000      0x0000000000000013

[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/misc] - [2023-03-17 04:09:41]
[0] <git:(main 69904e8* ) > 
```

16:09

Friday, March 17

## partB

这一部分要求加入指令iaddq，这条指令可以在习题4.3看到详细介绍。

添加指令，需要修改文件 sim/seq/seq-full.hcl ，主要是在每个阶段为iaddq指令选择合适的控制信号。

执行测试结果如下：

- 基准测试

You are simply making sure that your solution did not inject errors for the original instructions.

```
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/y86-code] - [2023-03-18 10:06:04]
[0] <git:(main 0bf4c8c*) > make testsim
Makefile:42: warning: ignoring prerequisites on suffix rule definition
Makefile:45: warning: ignoring prerequisites on suffix rule definition
Makefile:48: warning: ignoring prerequisites on suffix rule definition
Makefile:51: warning: ignoring prerequisites on suffix rule definition
./seq/ssim -t asum.yo > asum.seq
./seq/ssim -t asumr.yo > asumr.seq
./seq/ssim -t cjr.yo > cjr.seq
./seq/ssim -t j-cc.yo > j-cc.seq
./seq/ssim -t poptest.yo > poptest.seq
./seq/ssim -t pushquestion.yo > pushquestion.seq
./seq/ssim -t pushtest.yo > pushtest.seq
./seq/ssim -t prog1.yo > prog1.seq
./seq/ssim -t prog2.yo > prog2.seq
./seq/ssim -t prog3.yo > prog3.seq
./seq/ssim -t prog4.yo > prog4.seq
./seq/ssim -t prog5.yo > prog5.seq
./seq/ssim -t prog6.yo > prog6.seq
./seq/ssim -t prog7.yo > prog7.seq
./seq/ssim -t prog8.yo > prog8.seq
./seq/ssim -t ret-hazard.yo > ret-hazard.seq
grep "ISA Check" *.seq
asumr.seq:ISA Check Succeeds
asum.seq:ISA Check Succeeds
cjr.seq:ISA Check Succeeds
j-cc.seq:ISA Check Succeeds
poptest.seq:ISA Check Succeeds
prog1.seq:ISA Check Succeeds
prog2.seq:ISA Check Succeeds
prog3.seq:ISA Check Succeeds
prog4.seq:ISA Check Succeeds
prog5.seq:ISA Check Succeeds
prog6.seq:ISA Check Succeeds
prog7.seq:ISA Check Succeeds
prog8.seq:ISA Check Succeeds
pushquestion.seq:ISA Check Succeeds
pushtest.seq:ISA Check Succeeds
ret-hazard.seq:ISA Check Succeeds
rm asum.seq asumr.seq cjr.seq j-cc.seq poptest.seq pushquestion.seq pushtest.seq prog1.seq prog2.seq prog3.seq prog4.seq prog5.seq prog6.seq prog7.seq prog8.seq ret-hazard.seq
q
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/y86-code] - [2023-03-18 10:06:05]
[0] <git:(main 0bf4c8c*) > 
```

- 回归测试

- *make SIM=../seq/ssim*: test everything except iaddq.
- *make SIM=../seq/ssim TFLAGS=-i*: To test your implementation of iaddq

```
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/ptest] - [2023-03-18 10:02:52]
[0] <git:(main 0bf4c8c*) > make SIM=../seq/ssim
./optest.pl -s ..seq/ssim
Simulating with ..seq/ssim
All 49 ISA Checks Succeed
./jtest.pl -s ..seq/ssim
Simulating with ..seq/ssim
All 64 ISA Checks Succeed
./ctest.pl -s ..seq/ssim
Simulating with ..seq/ssim
All 22 ISA Checks Succeed
./htest.pl -s ..seq/ssim
Simulating with ..seq/ssim
All 600 ISA Checks Succeed
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/ptest] - [2023-03-18 10:03:14]
[0] <git:(main 0bf4c8c*) > make SIM=../seq/ssim TFLAGS=-i
./optest.pl -s ..seq/ssim -i
Simulating with ..seq/ssim
All 58 ISA Checks Succeed
./jtest.pl -s ..seq/ssim -i
Simulating with ..seq/ssim
All 96 ISA Checks Succeed
./ctest.pl -s ..seq/ssim -i
Simulating with ..seq/ssim
All 22 ISA Checks Succeed
./htest.pl -s ..seq/ssim -i
Simulating with ..seq/ssim
[lixiaoqi highlight] - [/Documents/Courses/CSAPP/lab/archlab/sim/ptest] - [2023-03-18 10:04:26]
[0] <git:(main 0bf4c8c*) > 
```

## partC

这部分要求“软硬兼施”，既可以修改软件实现，又可以修改硬件实现，使得程序得到最大的性能提升，性能通过CPE衡量。

### 基础修改

在基准程序上修改如下：

- 将常量1外提出循环
- `mrmovq (%rdi), %r10` 和 `rmmovq %r10, (%rsi)` 分离开，避免load/use冒险
- 循环测试len的部分，可以在修改 `%rdx` 后立刻使用jne判断，减少测试语句 `andq %rdx %rdx`

这一步做完，平均CPE达到了10.84。

## 加入iaddq指令

因为iaddq指令并不涉及控制冒险等繁琐的点，所以修改逻辑和seq版本几乎相同

### 基准测试

```
[lixiaoxi highlight] - [-/Documents/Courses/CSAPP/lab/archlab/sim/y86-code] - [2023-03-20 09:28:43]
[0] <git:(main b5d7996) > make testpsim
Makefile:42: warning: ignoring prerequisites on suffix rule definition
Makefile:45: warning: ignoring prerequisites on suffix rule definition
Makefile:48: warning: ignoring prerequisites on suffix rule definition
Makefile:51: warning: ignoring prerequisites on suffix rule definition
../pipe/psim -t asumr.yo > asumr.pipe
../pipe/psim -t asumr.yo > asumr.pipe
../pipe/psim -t cjr.yo > cjr.pipe
../pipe/psim -t j-cc.yo > j-cc.pipe
../pipe/psim -t poptest.yo > poptest.pipe
../pipe/psim -t pushquestion.yo > pushquestion.pipe
../pipe/psim -t pushtest.yo > pushtest.pipe
../pipe/psim -t prog1.yo > prog1.pipe
../pipe/psim -t prog2.yo > prog2.pipe
../pipe/psim -t prog3.yo > prog3.pipe
../pipe/psim -t prog4.yo > prog4.pipe
../pipe/psim -t prog5.yo > prog5.pipe
../pipe/psim -t prog6.yo > prog6.pipe
../pipe/psim -t prog7.yo > prog7.pipe
../pipe/psim -t prog8.yo > prog8.pipe
../pipe/psim -t ret-hazard.yo > ret-hazard.pipe
grep "ISA Check" *.pipe
asmr.pipe:ISA Check Succeeds
asumr.pipe:ISA Check Succeeds
cjr.pipe:ISA Check Succeeds
j-cc.pipe:ISA Check Succeeds
poptest.pipe:ISA Check Succeeds
prog1.pipe:ISA Check Succeeds
prog2.pipe:ISA Check Succeeds
prog3.pipe:ISA Check Succeeds
prog4.pipe:ISA Check Succeeds
prog5.pipe:ISA Check Succeeds
prog6.pipe:ISA Check Succeeds
prog7.pipe:ISA Check Succeeds
prog8.pipe:ISA Check Succeeds
pushquestion.pipe:ISA Check Succeeds
pushtest.pipe:ISA Check Succeeds
ret-hazard.pipe:ISA Check Succeeds
rm asumr.pipe asumr.pipe cjr.pipe j-cc.pipe poptest.pipe pushquestion.pipe pushtest.pipe prog1.pipe prog2.pipe prog3.pipe prog4.pipe prog5.pipe prog6.pipe prog7.pipe prog8.pipe
e ret-hazard.pipe
09:28 Monday, March 20
```

### 扩展测试

```
[lixiaoqi@highlight] - [~/Documents/Courses/CSAPP/lab/archlab/sim/ptest] - [2023-03-20 09:25:33]
[0] <git:(main b5d7996*)> make SIM=../pipe/psim
./optest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 49 ISA Checks Succeed
./jtest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 64 ISA Checks Succeed
./ctest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 22 ISA Checks Succeed
./htest.pl -s ../pipe/psim
Simulating with ../pipe/psim
  All 600 ISA Checks Succeed
[lixiaoqi@highlight] - [~/Documents/Courses/CSAPP/lab/archlab/sim/ptest] - [2023-03-20 09:25:45]
[0] <git:(main b5d7996*)> make SIM=../pipe/psim TFLAGS=-i
./optest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
  All 58 ISA Checks Succeed
./jtest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
  All 96 ISA Checks Succeed
./ctest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
  All 22 ISA Checks Succeed
./htest.pl -s ../pipe/psim -i
Simulating with ../pipe/psim
  All 756 ISA Checks Succeed
[lixiaoqi@highlight] - [~/Documents/Courses/CSAPP/lab/archlab/sim/ptest] - [2023-03-20 09:26:01]
[0] <git:(main b5d7996*)> []
```

## 使用iaddq指令进行改进

将加减常数的指令改成iaddq，CPE达到10.70，提升微弱是因为常数已经经过代码外提，每次只有循环外会额外用到两次(常数1和常数8)。

## 根据分支预测改进

因为处理器每次面对条件分支，都首先预测跳转地址，所以为了降低CPE，应该将分支跳转的地址写成常用的地址。

循环之前的判断：

```
andq %rdx,%rdx    # len <= 0?
jle Done        # if so, goto Done:
Loop:
...
Done:
...
```

这里分支跳转可以选择loop，减少预测错误：

```

andq %rdx,%rdx      # len <= 0?
jg  Loop
jmp Done       # if so, goto Done:
Loop:
...
Done:
...

```

此处修改使CPE降到10.55

## 循环展开

参考教材(3e)5.8节的循环展开技术。

类似如下叙述修改 ncopy.ys，实现kx1展开：

```

word_t ncopy(word_t *src, word_t *dst, word_t len)
{
    word_t count = 0;
    word_t val;

    len -= k
    while (len > 0) {
        val1 = src[0];
        // ...
        valk = src[k-1];

        dst[0] = val1;
        // ...
        dst[k-1] = valk;

        src += k, dst += k;

        if (val1 > 0)
            count++;
        // ...
        if (valk > 0)
            count++;
    }
    len += k;
    if (len != 0)
        // 按照之前的逻辑遍历
    return count;
}

```

采用7x1展开，平均CPE降到了8.04，然后尝试8x1展开，平均CPE为8.08。

观察发现，8x1性能有降低的主要原因是，在元素数量为7的倍数但不为8的倍数时，8x1性能较差（相较于7x1），而显然7x1下，性能提高数量要多于8x1。由此考虑再按照8x1结束后再按照4x1展开，最终平均CPE降到了8。