lab4

2024/4/20-2024/4/27

PB22111702 李岱峰

严重警告：该程序需要lhdf5库，以及tk库(可不需要)，我在我的vlab平台上修改makefile，没有使用这两个库

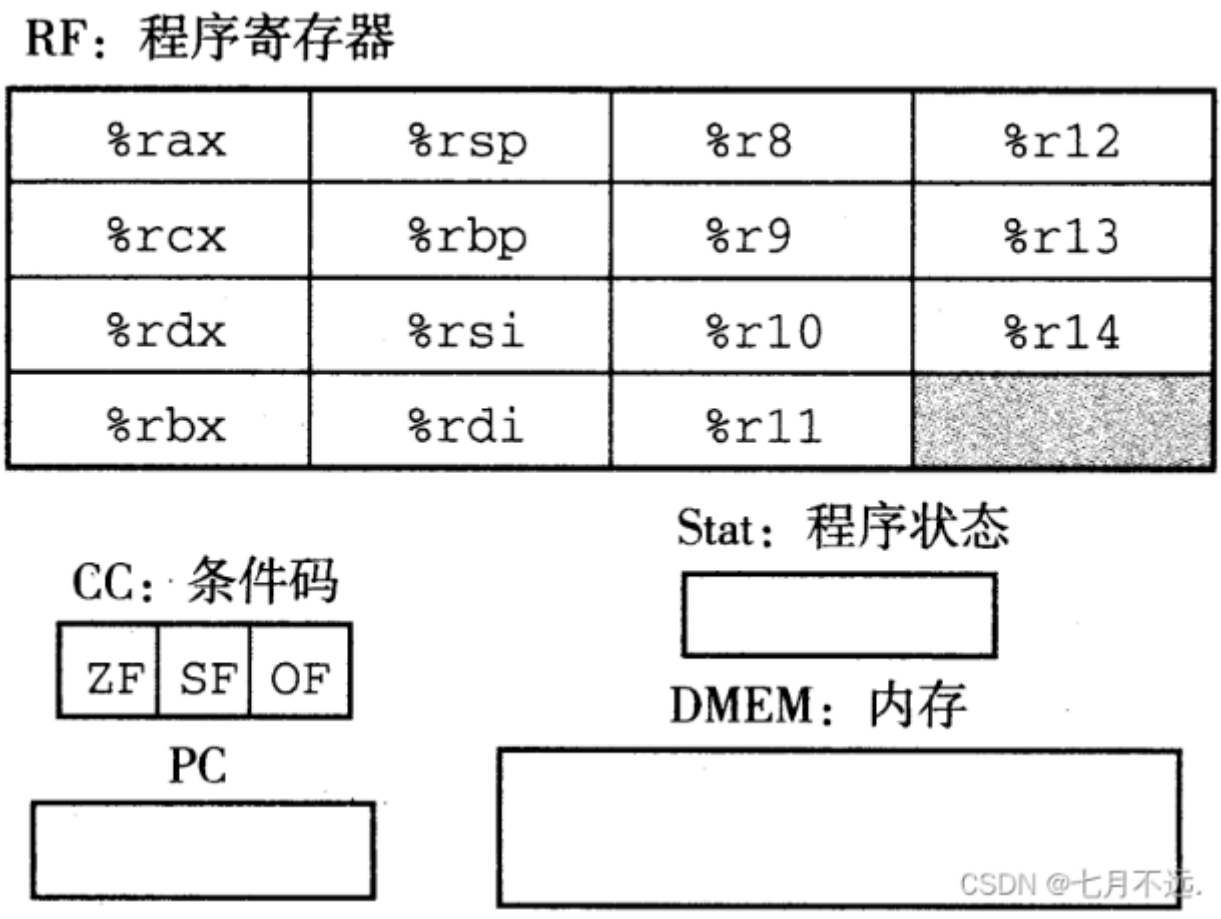
一. 实验过程

阅读实验文档，作者设计了一款"新"指令集Y86，是X86的改版，阅读中文版深入理解计算机系统P256页，熟悉Y86.

准备阶段

4种mov：irmovq\rr\rm\mr:

> irmovq指令表示将一个立即数存进一个寄存器中 rrmovq指令表示将一个寄存器中的值存进一个寄存器中 rmmovq指令表示将一个寄存器中的值存进一个内存地址所对应的内存中 mrmovq指令表示将一个内存中的值存进一个寄存器中



> jmp 直接跳转 jle 小于或等于 (有符号 <=) jl 小于 (有符号 <) je 相等 / 零 jne 不相等 / 非零 jge 大于或等于 (有符号 >=) jg 大于(有符号 >)

传送指令：cmovle，cmovl，cmove，cmovne，cmovge，cmovg

栈指令，call，ret等同x86

PartA

1.链表求和

文件目录：./sim/misc

```
long sum_list(list_ptr ls)
{
    long val = 0;
    while (ls) {
    val += ls->val;
    ls = ls->next;
    }
    return val;
}
```

取结点地址内容，add，判断链表是否遍历到末尾即可。

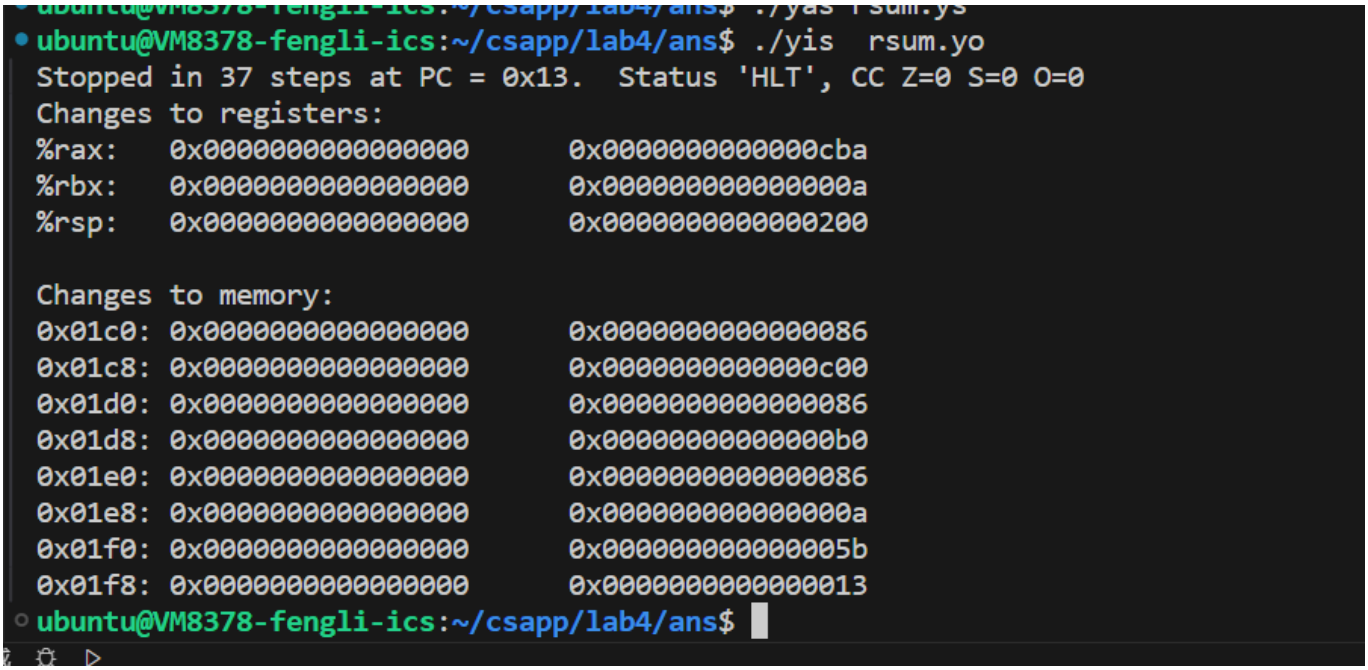
```
ubuntu@VM8378-fengli-ics:~/csapp/lab4$ ./sim/misc/yis sum.yo
Stopped in 26 steps at PC = 0x13.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000cba
%rsp:   0x0000000000000000      0x0000000000000200
%rsi:   0x0000000000000000      0x0000000000000c00

Changes to memory:
0x01f0: 0x0000000000000000      0x000000000000005b
0x01f8: 0x0000000000000000      0x0000000000000013
ubuntu@VM8378-fengli-ics:~/csapp/lab4$
```

2.递归链表求和

```
long rsum_list(list_ptr ls)
{
    if (!ls)
        return 0;
    else {
        long val = ls->val;
        long rest = rsum_list(ls->next);
        return val + rest;
```

```
    }
}
```

代码统一放在ans文件夹里。



3.复制数组，计算每个元素的异或

```
long copy_block(long *src, long *dest, long len)
{
    long result = 0;
    while (len > 0) {
        long val = *src++;
        *dest++ = val;
        result ^= val;
        len--;
    }
    return result;
}
```

```
ubuntu@VM8378-fengli-ics:~/csapp/lab4/ans$ ./yas copy.ys
ubuntu@VM8378-fengli-ics:~/csapp/lab4/ans$ ./yis copy.yo
Stopped in 36 steps at PC = 0x13.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000cba
%rsp:   0x0000000000000000      0x0000000000000200
%rsi:   0x0000000000000000      0x0000000000000048
%rdi:   0x0000000000000000      0x0000000000000030
%r8:    0x0000000000000000      0x0000000000000008
%r9:    0x0000000000000000      0x0000000000000001
%r10:   0x0000000000000000      0x0000000000000c00

Changes to memory:
0x0030: 0x0000000000000111      0x000000000000000a
0x0038: 0x0000000000000222      0x00000000000000b0
0x0040: 0x0000000000000333      0x0000000000000c00
0x01f0: 0x0000000000000000      0x000000000000006f
0x01f8: 0x0000000000000000      0x0000000000000013
ubuntu@VM8378-fengli-ics:~/csapp/lab4/ans$
```

PartB

1.添加修改SEQ指令

> o 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.

要求：实现iaddq指令（将立即数与寄存器相加）

iaddq执行流程：

> 1.取指：mem[pc]->code,pc+=10

> 2.译码：Reg[rb],imm

> 3.执行：val_<-Reg[rb]+imm

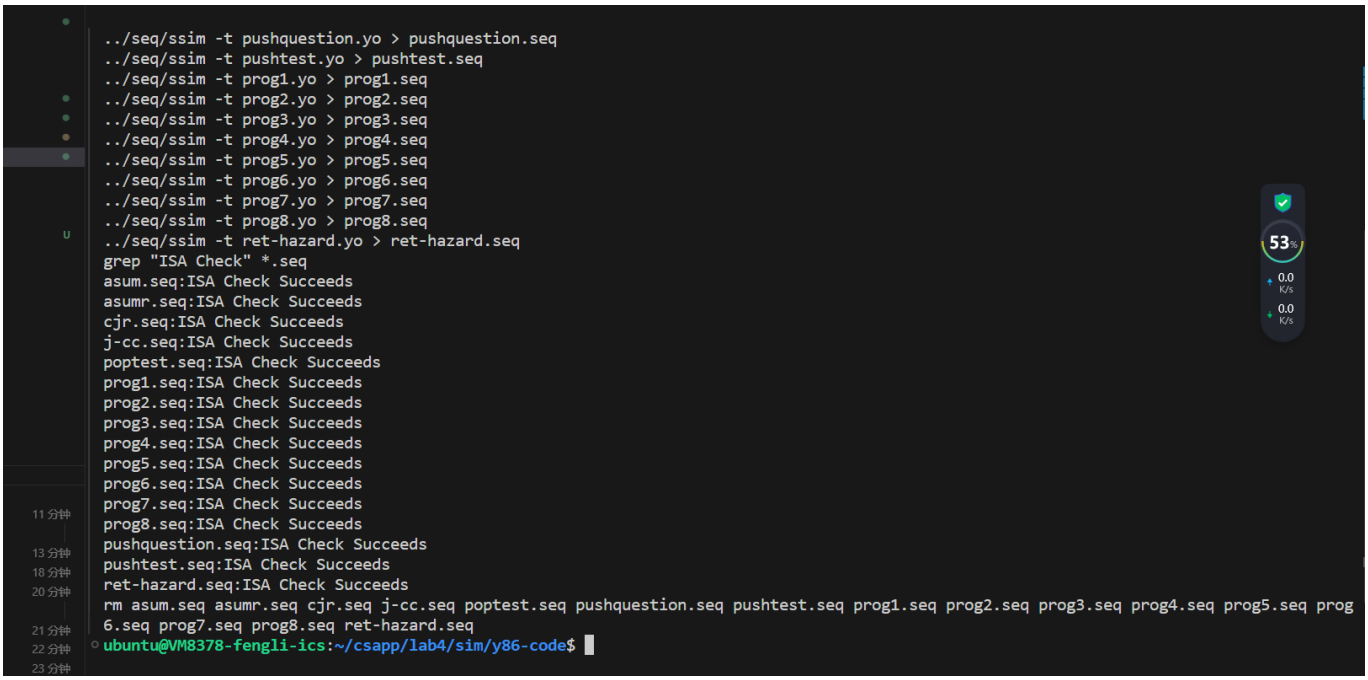
> 4.访存：

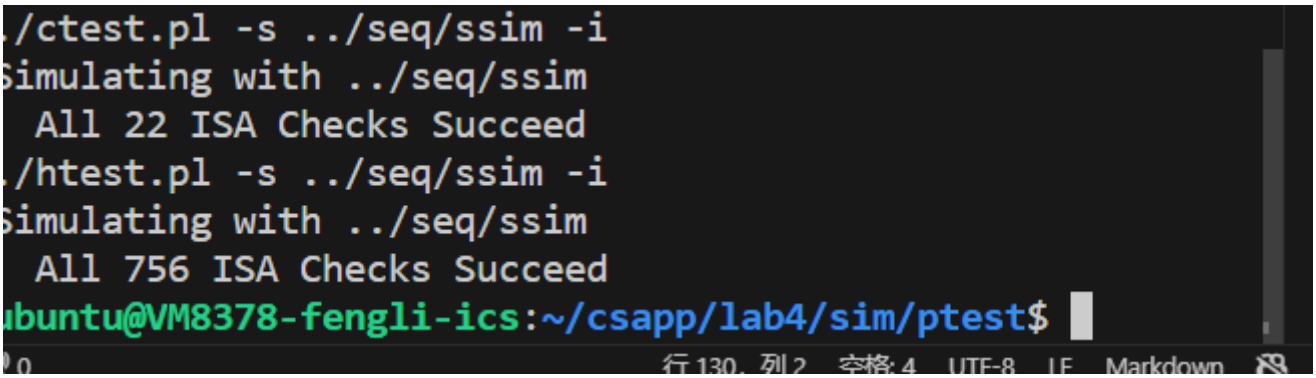
> 5.写回：Reg[rb]<-val_

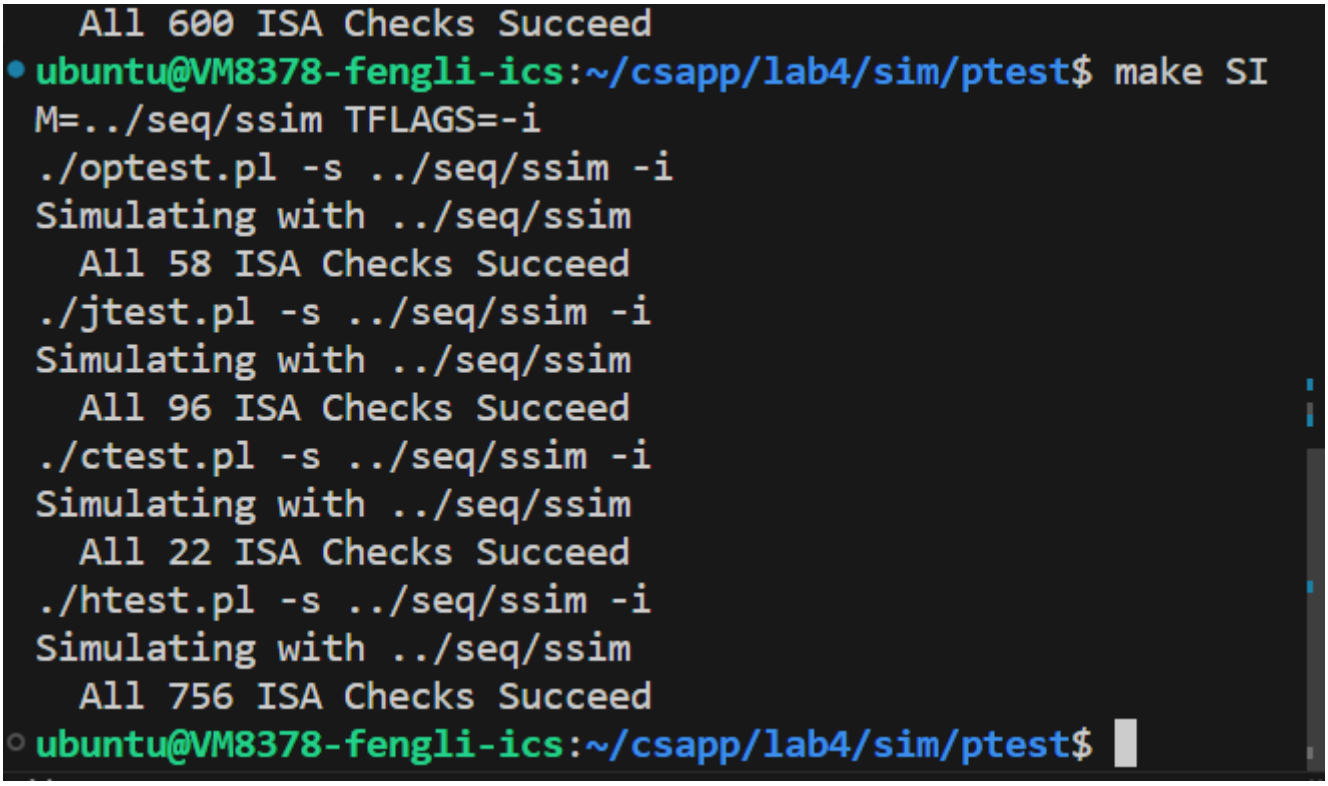
按照上述流程修改./sim/seq/seq-full.hcl

该程序设计很精巧，简化为询问每一个阶段这些指令作什么：Fetch阶段，向该程序中填写instr_valid中加入IIADDQ，表示该指令合法，need_regids中加入，表示该指令需要寄存器，need_valc中加入，表示该指令需要立即数，同时命名为valC。

运行指令make testssim，得到结果







如图，partb部分通过

PartC

working in directory sim/pipe in this part

1.修改 pipe-full.hcl ，使得该代码能够尽可能运行的更快

1.添加指令iaddq；

在pipe-full.hcl中找到了：

> Your task is to implement the iaddq instruction The file contains a declaration of the icodes for iaddq (IIADDQ) Your job is to add the rest of the logic to make it work

修改pipe-full.hcl，添加iaddq指令，添加方法同partb

```
CPI: 44 cycles/32 instructions = 1.38
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/pipe$ cd ../ptest; 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
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/ptest$ cd ../ptest; 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
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/ptest$
```

2.加载转发

在重整完ncopy.ys后，我发现成绩跑不满，决定完成加载转发的设计。

正常来讲，在遇到load-use时，唯一的可能就是bubble一个周期，否则从内存load上的数据无法到达临近下一条指令的执行阶段(差一个周期，load访存时，下一条指令已经在执行).

但书中讲述了一个情况，就是当load的临近下一条是st时，st指令不会在执行阶段写数据，而是在访存阶段写数据，这就有了load上数据分出来一份转发给st指令的可能。无需bubble，节省一个周期。例如：

```
ld r1,r2
st r2,r1
```

在这里，只需将ld在访存后读出的数据，放到exmem阶段寄存器中就好，无需暂停一个周期。

由此，对pipe-full的load-use机制进行修改。268行，execute阶段后寄存器添加另一个路径。326、337、342行，更改stall、bubble机制，去掉irmmove和ipush的stall和bubble

```
48      115     2.40
49      107     2.18
50      115     2.30
51      107     2.10
52      107     2.06
53      117     2.21
54      117     2.17
55      109     1.98
56      115     2.05
57      117     2.05
58      119     2.05
59      113     1.92
60      111     1.85
61      111     1.82
62      115     1.85
63      115     1.83
64      115     1.80
Average CPE     7.97
Score   50.6/60.0
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/pipe$
```

最好成绩50.9分（还没满！？）

2.重整ncopy.ys代码逻辑 ，使运行效果更快

代码C语言分析如下：

```c
int ncopy(int *src, int *dst, int len)
{
    int count = 0;  //%rax
    int val;        //%r10

    while (len > 0) { //%rdx
        val = *src++;    //mem[%rdi++]
        *dst++ = val;    //mem[%rsi++]
        if (val > 0){
            count++;
            }
        len--;
    }
    return count;
}
```

1.我先进行循环展开

尽可能减少循环发生的次数，因为branch类指令会导致流水线cpu停止一个周期，同时减少了条件判断的次数。

这里我实验6路展开，展开的意思是将一次循环复制一个数据变为一次循环复制6个数据。

```
54      OK
55      OK
56      OK
57      OK
58      OK
59      OK
60      OK
61      OK
62      OK
63      OK
64      OK
128     OK
192     OK
256     OK
68/68 pass correctness test
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/pipe$ ./benchmark.pl
        ncopy
0       16
1       25      25.00
2       37      18.50
3       46      15.33
4       58      14.50
5       65      13.00
6       58      9.67
7       67      9.57
8       79      9.88
9       88      9.78
10      100     10.00
11      107     9.73
12      96      8.00
13      105     8.08
14      117     8.36
15      126     8.40
16      138     8.62
17      145     8.53
18      134     7.44
19      143     7.53
20      155     7.75
```

```
41      297     7.24
42      286     6.81
43      295     6.86
44      307     6.98
45      316     7.02
46      328     7.13
47      335     7.13
```

```
48      324     6.75
49      333     6.80
50      345     6.90
51      354     6.94
52      366     7.04
53      373     7.04
54      362     6.70
55      371     6.75
56      383     6.84
57      392     6.88
58      404     6.97
59      411     6.97
60      400     6.67
61      409     6.70
62      421     6.79
63      430     6.83
64      442     6.91
Average CPE     8.29
Score   44.3/60.0
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/pipe$
```

如图，展开后成果如上。这个版本答案记录为partc_try1，保存在./ans中。

score 44.3/60 .

2.然后根据新ncopy.ys，发现循环减少后，跳转次数可以明显减少，这样就可以将分支预测改为不跳转。

```
9
0   # Predict next value of PC
1   word f_predPC = [
2       f_icode in { ICALL } : f_valC;
3       1 : f_valP;
4   ];
5
```

如图，保留ICALL(call的跳转预测)，去掉IJXX，即取消原先的jump预测

在更改一些细节优化后（如减少跳转，减少气泡的出现概率）

```
55      107     1.95
56      105     1.88
57      109     1.91
58      113     1.95
59      113     1.92
60      111     1.85
61      113     1.85
62      103     1.66
63      113     1.79
64      109     1.70
Average CPE     8.05
Score   49.1/60.0
ubuntu@VM8378-fengli-ics:~/csapp/lab4/sim/pipe$
```

此时运行最好成绩49.1，如图

二.总结

实验最终完成，parta，partb通过，但partc只拿到了50.9分(截至4.27)，实在找不到可以优化的点，或许运行程序的微调可以提高运行效率。