# 实验4 优化Y86-64流水线处理器性能

## 实验目的

通过本次实验，熟悉书中第四章的Y86-64处理器和相关的HCL等编程语言，熟练编写Y86-64的汇编代码。熟悉Y86-64处理器仿真器并对其进行性能优化。

## 实验内容

第一部分（Part A）：编写Y86-64简单程序，熟悉Y86-64工具。

第二部分（Part B）：扩展SEQ模拟器支持新的指令。

第三部分（Part C）：在前两部分的基础上，优化Y86-64基准测试程序和流水线处理器设计。

## 实验环境及配置

本实验使用了软件VMware 16 pro，虚拟机配置为Ubuntu22.04 LTS版本。

按照乐学上的补充说明，在终端输入如下指令：

1. sudo apt install make
2. sudo apt install make-guile
3. sudo apt install tcl tcl-dev tk tk-dev
4. sudo apt install flex
5. sudo apt install bison

## 实验过程及截图

### 3.1 Part A

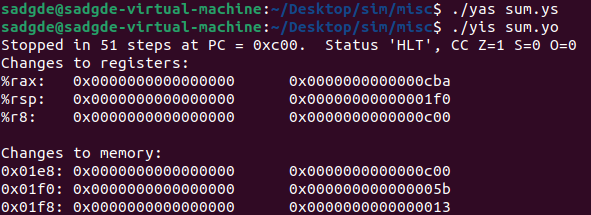
这部分需要编写3个函数，分别为sum.ys, rsum.ys, copy.ys。

#### 3.1.1 sum函数

结合书P251-P252中的图4-6和图4-7进行Y86-64代码编写：

1. # Execution begins at address 0
2. .pos 0
3. irmovq stack, %rsp
4. call main
5. halt
6. # Sample linked list
7. .align 8
8. ele1:
9. .quad 0x00a
10. .quad ele2
11. ele2:
12. .quad 0x0b0
13. .quad ele3
14. ele3:
15. .quad 0xc00
16. .quad 0
17. main:
18. irmovq ele1, %rdi
19. call sum\_list
20. ret
21. # long sum\_list(list\_ptr ls)
22. # ls in %rdi
23. sum\_list:
24. pushq %r8
25. xorq %rax, %rax # clean %rax
26. andq %rdi, %rdi
27. jmp test
28. loop:
29. mrmovq (%rdi), %r8
30. addq %r8, %rax
31. mrmovq 8(%rdi), %rdi  # because align 8
32. andq %rdi, %rdi
33. test:
34. jne loop   # stop when 0
35. popq %r8
36. ret
37. .pos 0x200
38. stack:

运行结果如下：

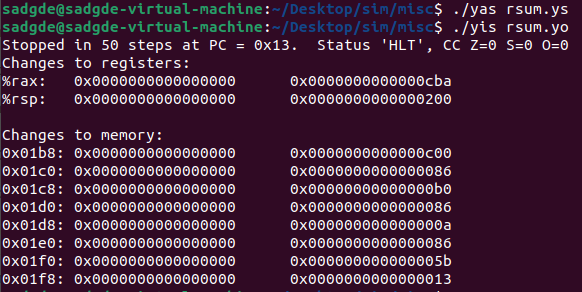


#### 3.1.2 rsum函数

相比上面的函数，这是一个递归求和，代码如下：

1. # Execution begins at address 0
2. .pos 0
3. irmovq stack,%rsp
4. call main
5. halt
6. # Sample linked list
7. .align 8
8. ele1:
9. .quad 0x00a
10. .quad ele2
11. ele2:
12. .quad 0x0b0
13. .quad ele3
14. ele3:
15. .quad 0xc00
16. .quad 0
17. main:
18. irmovq ele1, %rdi
19. call rsum\_list
20. ret
21. # long rsum\_list(list\_ptr ls)
22. # ls in %rdi
23. rsum\_list:
24. pushq %r8
25. xorq %rax, %rax  # clean %rax
26. jmp test
27. loop:
28. mrmovq (%rdi), %r8 # val = ls -> val
29. mrmovq 8(%rdi), %rdi  # ls -> next
30. call rsum\_list
31. addq %r8, %rax  # val + rest
32. jmp end
34. test:
35. andq %rdi, %rdi
36. jne loop
37. xorq %rax, %rax   # if !ls return 0
38. end:
39. popq %r8
40. ret
41. # Stack starts here and grows to lower addresses
42. .pos 0x200
43. stack:

运行结果如下：

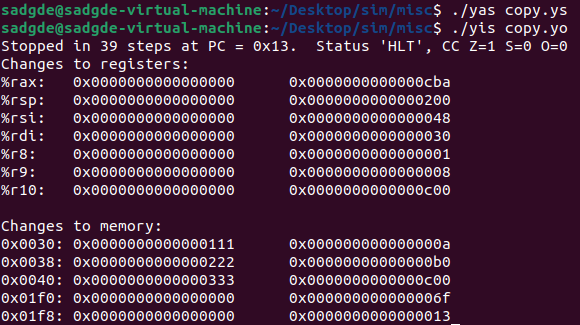


#### 3.1.3 copy函数

这个函数较为简单，但是要传入的参数较多，代码如下：

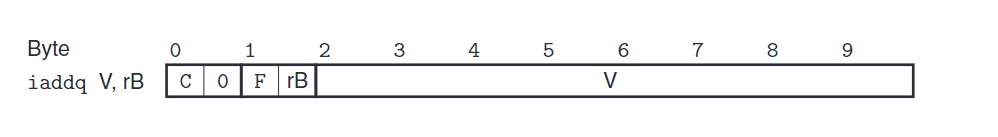
1. # Execution begins at address 0
2. .pos 0
3. irmovq stack, %rsp
4. call main
5. halt
6. # Sample linked list
7. .align 8
8. # Source block
9. src:
10. .quad 0x00a
11. .quad 0x0b0
12. .quad 0xc00
13. # Destination block
14. dest:
15. .quad 0x111
16. .quad 0x222
17. .quad 0x333
19. main:
20. irmovq src, %rdi
21. irmovq dest, %rsi
22. irmovq $3, %rdx
23. call copy\_block
24. ret
25. # long copy\_block(long \*src, long \*dest, long len)
26. # src in %rdi
27. # dest in %rsi
28. # len in %rdx
29. copy\_block:
30. irmovq $1, %r8
31. irmovq $8, %r9
32. xorq %rax, %rax  # clean %rax
33. jmp test
34. loop:
35. mrmovq (%rdi), %r10
36. rmmovq %r10, (%rsi)
37. addq %r9, %rdi
38. addq %r9, %rsi
39. xorq %r10, %rax
40. subq %r8, %rdx
41. test:
42. andq %rdx, %rdx
43. jne loop
44. ret
45. .pos 0x200
46. stack:

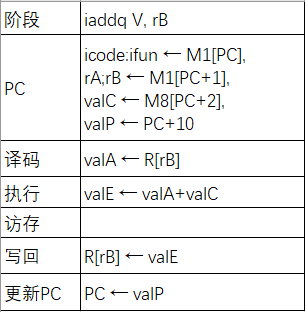
运行结果如下：



### 3.2 Part B

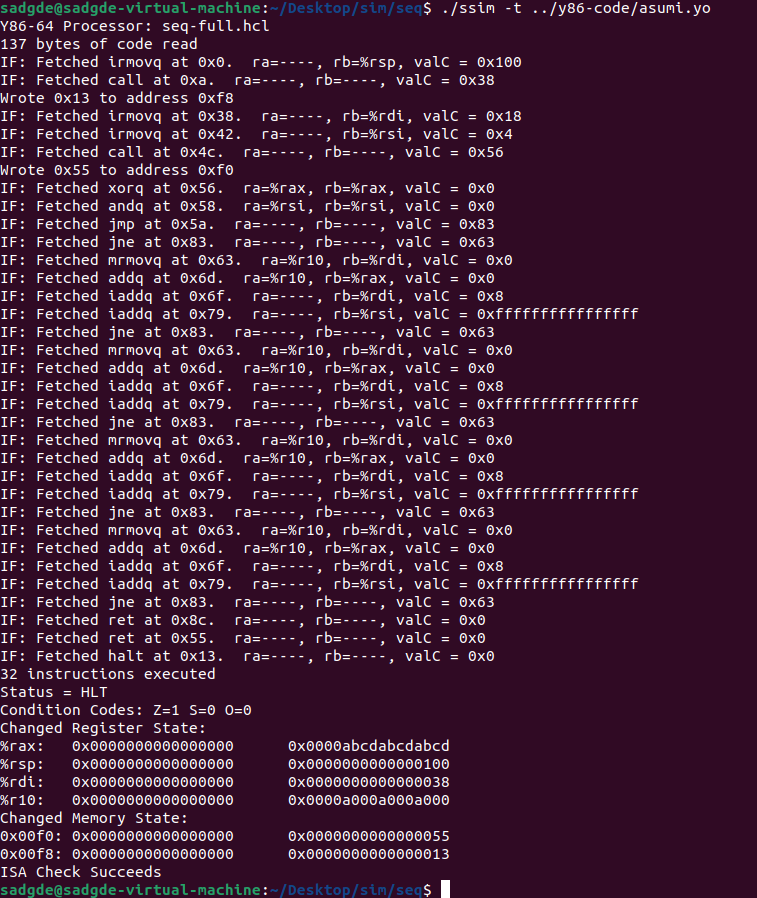
按照乐学上的指导对makefile文件和两个.c文件进行改写，然后仿照书中图4.18画出iaddq指令在顺序实现中的计算，如下：



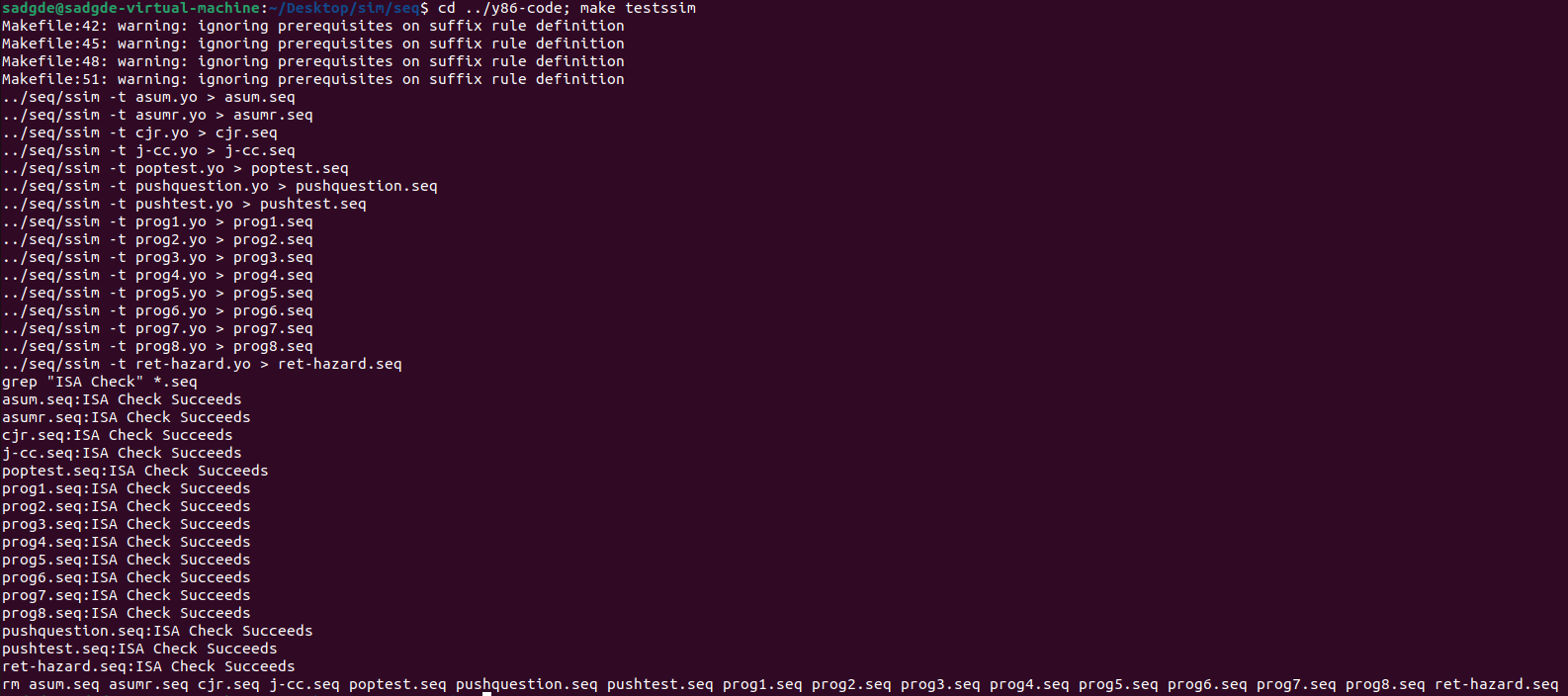


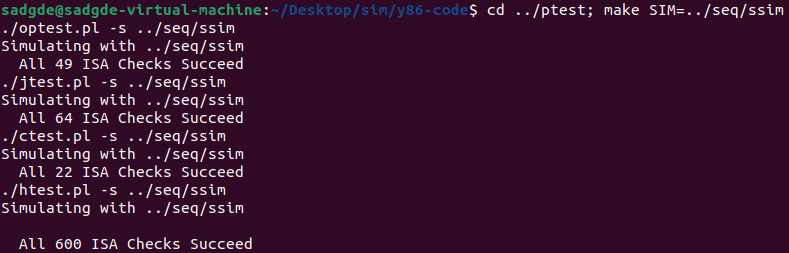
根据上图对seq-full.hcl文件进行更改，具体主要为在一些word中增添iaddq指令的判断分支。

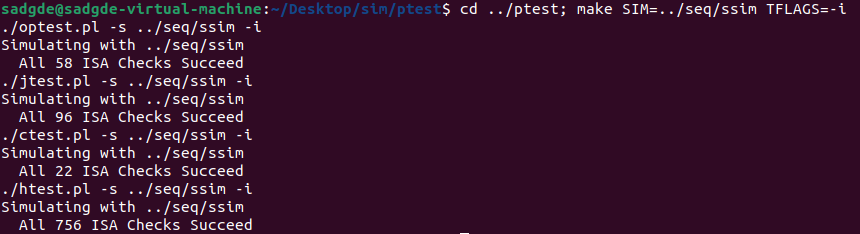
y86小程序测试运行结果如下：



用基准程序测试检查正确性结果如下：



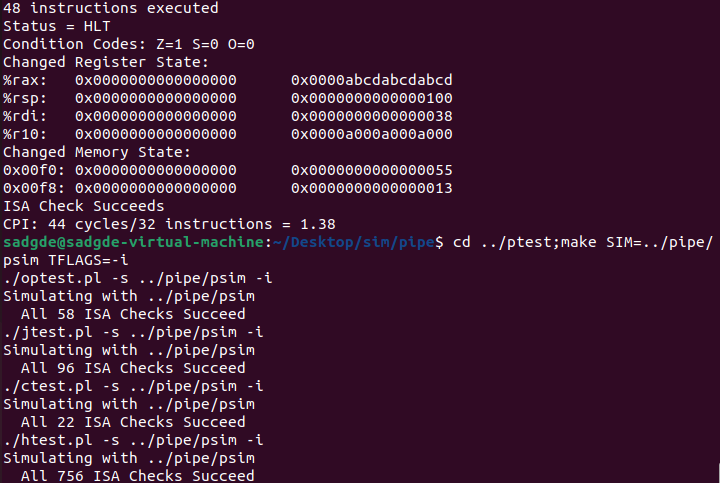
进行除iaddq指令的回归测试结果：

进行iaddq指令的回归测试结果：

### 3.3 Part C

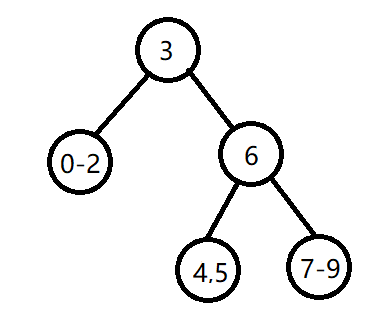
根据实验文档及乐学上的提示，对sim/pipe中的环境进行配置，本实验需要更改pipe-full.hcl和ncopy.ys文件。

对于pipe-full.hcl文件，我们仿照Part B，为模拟器增添iaddq指令，具体实现和Part B中的seq-full.hcl基本相同，在此不再赘述。保存更改后的pipe-full.hcl文件，然后用同Part B的方式进行小程序测试和回归测试，结果如下图：



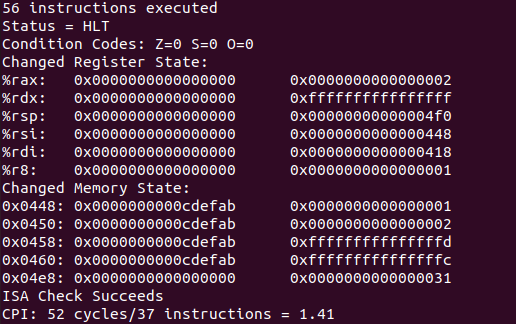
下面将对ncopy.ys文件进行更改，我从以下几个方面对其进行了优化：

1. 将一些指令改为iaddq，这样比两个寄存器之间的操作要更快。
2. 注意到Loop中的两个movq指令之间是有数据依赖的，会导致等待一个时钟周期，所以我们可以将Npos中的两个iaddq分别插入到两个movq指令之间。
3. 因为iaddq会设置条件码，所以不需要andq %rdx, %rdx
4. 循环展开。经过尝试，这里使用了10\*1的循环展开。
5. 对于余下数的操作，当不满10个数时，考虑使用二分法判断剩余的长度并执行对应的挨个复制操作。具体的二分图如下：

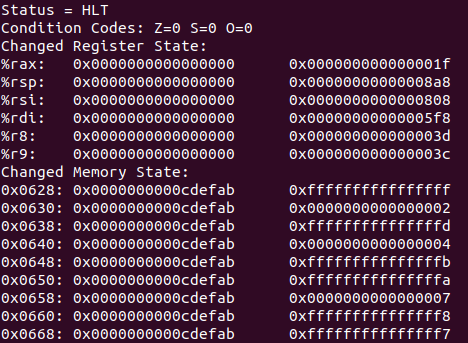


由于代码篇幅过长，故不在此展示。接下来展示运行结果。

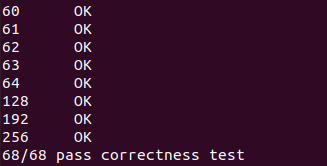
执行指令./psim -t sdriver.yo的结果如下：



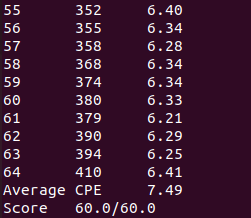
执行指令./psim -t ldriver.yo的结果如下：



执行指令./correctness.pl的结果如下（节选）：



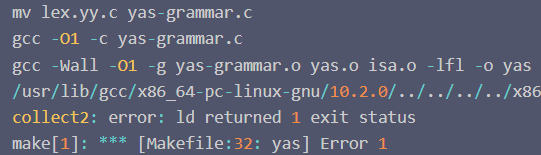
执行指令./benchmark.pl，计算CPE值和得分，结果如下（节选）：

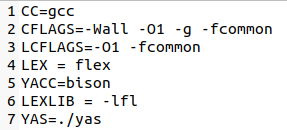


## 遇到的问题及解决方案

1. 问题：在sudo apt install时报错Failed to fetch <http://mirrors.tencentyun.com/ubuntu/pool/main/l/linux>

解决方法：需更换Ubuntu镜像源，我这里更换为了清华源。

1. 在make时报错如下  
   

解决方案：gcc-10将默认值从"-fcommon“更改为"-fno-common”。需要在Makefiles中添加“-fcommon”标志，具体如下  


每个文件夹中的makefile文件都需要做此更改，另外sim文件夹下的makefile文件要进行乐学中相同的修改操作。

1. 在Part C中，按照文档中执行指令./gen-driver.pl -f ncopy.ys -n K -rc > driver.ys和make driver.yo时提示driver.yo is out of date，问题尚未解决，上面的报告也没有执行此指令。

## 实验总结与心得

通过这次实验，我对于第四章中所讲的Y86-64处理器的结构，以及其各个指令的顺序计算流程有了更清晰的认识，并且结合第五章的有关程序性能优化的内容对Part C中的ncopy程序进行了优化实践，加深了对于CPE的理解和流水线的理解。