

## 实验-优化 Y86-64 流水线处理器性能

班级： 07812201    学号： 1820221053    姓名： 曾迦隼

### 一、实验目的

在本实验中，将了解流水线 Y86-64 处理器的设计和实现，优化它和基准程序以最大限度地提高性能。

### 二、实验内容

实验分为三个部分，每个部分都有自己的上交成果。

第一部分（Part A）：您将编写一些简单的 Y86 64 程序，并熟悉 Y86-64 工具。

第二部分（Part B）：您将使用新的指令扩展 SEQ 模拟器。

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

### 三、实验步骤

环境准备：本次实验使用 VMWARE WORKSTATION PRO 17 的 Ubuntu 18.04.6 系统上进行的。

在乐学平台上下载 sim.tar，并把文件拷贝到虚拟机中。并使用 `tar -xvf sim.tar` 进行解压。然后再 sim 目录执行 `make clean;make` 指令构建工具。进行 make 时提示缺少依赖，需安装以下依赖：

```
sudo apt install tcl tcl-dev tk tk-dev
```

```
sudo apt install flex
```

```
sudo apt install bison
```

第一部分（Part A）：

在这一部分中的任务将在 sim/misc 目录下工作，在此需要完成 3 个汇编程序（sum.y, rsum.y, copy.y），分别完成 examples.c 的 3 个函数。

编写完三个汇编程序后，执行 `./yas sum.y` 生成 sum.yo 文件，同理对其余两个一样。之后执行 `./yis sum.yo` 查看运行结果。

## 第二部分 (Part B) :

在这一部分中的任务将在/sim/seq 目录下工作,任务是扩展 SEQ 处理器以支持 iaddq 指令,在此我们只需要修改 seq-full.hcl 文件。在该文件中可以看到

```
# Instruction code for iaddq instruction
```

```
wordsig IIADDQ    'I_IADDQ'
```

符号表已经添加了 iaddq,这时我们需要修改 Fetch 阶段的以下部分,增加 IADDQ 指令:

```
bool instr_valid = icode in
```

```
{ INOP, IHALT, IRRMOVQ, IIRMOVQ, IRMMOVQ, IMRMVQ,
  IOPQ, IJXX, ICALL, IRET, IPUSHQ, IPOPQ, IIADDQ };
```

```
bool need_regids =
```

```
  icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,
    IIRMOVQ, IRMMOVQ, IMRMVQ, IIADDQ };
```

```
bool need_valC =
```

```
  icode in { IIRMOVQ, IRMMOVQ, IMRMVQ, IJXX, ICALL, IIADDQ };
```

修改 Decode 阶段的部分:

```
word srcB = [
```

```
  icode in { IOPQ, IRMMOVQ, IMRMVQ, IIADDQ } : rB;
  icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
  1 : RNONE; # Don't need register
```

```
];
```

```
word dstE = [
```

```
  icode in { IRRMOVQ } && Cnd : rB;
  icode in { IIRMOVQ, IOPQ, IIADDQ } : rB;
  icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
  1 : RNONE; # Don't write any register
```

```
];
```

修改 Execute 阶段的部分:

```
word aluA = [  
    icode in { IRRMOVQ, IOPQ } : valA;  
    icode in { IIRMOVQ, IRMMOVQ, IMRMVQ, IIADDQ } : valC;  
    icode in { ICALL, IPUSHQ } : -8;  
    icode in { IRET, IPOPQ } : 8;  
    # Other instructions don't need ALU  
];  
  
word aluB = [  
    icode in { IRMMOVQ, IMRMVQ, IOPQ, ICALL,  
        IPUSHQ, IRET, IPOPQ, IIADDQ } : valB;  
    icode in { IRRMOVQ, IIRMOVQ } : 0;  
    # Other instructions don't need ALU  
];  
  
bool set_cc = icode in { IOPQ, IIADDQ };
```

修改完成后, 需要基于此 HCL 文件构建 SEQ 模拟器 (ssim) 的新实例, 然后对其进行测试:

执行 `make VERSION=full`;

出现问题报告找不到 `tk.h` 头文件, 解决方法是把 Makefile 中第 20 行的 `tcl` 版本改为 8.6、第 26 行改为 `CFLAGS=-Wall -O2 -DUSE_INTERP_RESULT`。再次执行还是报错 `undefined reference to `matherr'`, 此时需要修改 `/sim/pipe/psim.c` 的 806、807 行和 `/sim/seq/ssim.c` 的 844、845 行, 将其注释或删除, 即含有 `matherr` 的一行和下一行。然后便能编译成功, 忽略 Warning 信息。

执行

```
./ssim -t ../y86-code/asumi.yo  
(cd ../y86-code; make testssim)  
(cd ../ptest; make SIM=../seq/ssim)  
(cd ../ptest; make SIM=../seq/ssim TFLAGS=-i)
```

来进行测试, 如果所有测试都能顺利通过, 那么此部分就完成了。

### 第三部分 (Part C) :

这部分的任务在/sim/pipe 目录下工作，任务是修改 pipe-full.hcl 和 ncopy.ys 文件，使 ncopy.ys 尽可能快速地运行。

首先对 pipe-full.hcl 进行修改，如 Part B 部分，扩展 iaddq 指令功能。

然后优化 ncopy.ys 的代码，主要思想是采用循环展开进行优化，把循环展开成 8x8，然后最后不足 8 个时利用二分查找，判断剩余多少个，然后跳转到剩余不同数的展开。

```
ncopy.ys:
iaddq $-8, %rdx
jl Test
```

Loop8x8:

```
#取地址
mrmovq 0(%rdi), %r8
mrmovq 8(%rdi), %r9
mrmovq 16(%rdi), %r10
mrmovq 24(%rdi), %r11
mrmovq 32(%rdi), %r12
mrmovq 40(%rdi), %r13
mrmovq 48(%rdi), %r14
mrmovq 56(%rdi), %rbx
```

```
#赋值
rmmovq %r8, 0(%rsi)
rmmovq %r9, 8(%rsi)
rmmovq %r10, 16(%rsi)
rmmovq %r11, 24(%rsi)
rmmovq %r12, 32(%rsi)
rmmovq %r13, 40(%rsi)
rmmovq %r14, 48(%rsi)
rmmovq %rbx, 56(%rsi)
```

#判断是否可以 count+1

judge0:

andq %r8, %r8

jle judge1

iaddq \$1, %rax

judge1:

andq %r9, %r9

jle judge2

iaddq \$1, %rax

judge2:

andq %r10, %r10

jle judge3

iaddq \$1, %rax

judge3:

andq %r11, %r11

jle judge4

iaddq \$1, %rax

judge4:

andq %r12, %r12

jle judge5

iaddq \$1, %rax

judge5:

andq %r13, %r13

jle judge6

iaddq \$1, %rax

judge6:

andq %r14, %r14

jle judge7

iaddq \$1, %rax

judge7:

```
andq %rbx, %rbx
jle Loop
iaddq $1, %rax
```

Loop:

```
iaddq $64, %rdi
iaddq $64, %rsi
iaddq $-8, %rdx
jge Loop8x8
```

Test:

```
iaddq $5, %rdx
jl LeftChild
je Remain3
```

RightChild:

```
iaddq $-2, %rdx
jg RightChild2
jl Remain4
je Remain5
```

LeftChild:

```
iaddq $2, %rdx
je Remain1
jl Done
iaddq $-1, %rdx
jmp Remain2
```

RightChild2:

```
iaddq $-1, %rdx
je Remain6
```

Remain7:

```
    mrmovq 48(%rdi), %r8
    rmmovq %r8, 48(%rsi)
    andq %r8, %r8
```

Remain6:

```
    mrmovq 40(%rdi), %r8
    jle Extra6
    iaddq $1, %rax
```

Extra6:

```
    rmmovq %r8, 40(%rsi)
    andq %r8, %r8
```

Remain5:

```
    mrmovq 32(%rdi), %r8
    jle Extra5
    iaddq $1, %rax
```

Extra5:

```
    rmmovq %r8, 32(%rsi)
    andq %r8, %r8
```

Remain4:

```
    mrmovq 24(%rdi), %r8
    jle Extra4
    iaddq $1, %rax
```

Extra4:

```
    rmmovq %r8, 24(%rsi)
    andq %r8, %r8
```

Remain3:

```
    mrmovq 16(%rdi), %r8
    jle Extra3
    iaddq $1, %rax
```

Extra3:

```
    rmmovq %r8, 16(%rsi)
    andq %r8, %r8
```

Remain2:

```
    mrmovq 8(%rdi), %r8
    jle Extra2
    iaddq $1, %rax
```

Extra2:

```
    rmmovq %r8, 8(%rsi)
    andq %r8, %r8
```

Remain1:

```
    mrmovq (%rdi), %r8
    jle Extra1
    iaddq $1, %rax
```

Extra1:

```
    rmmovq %r8, (%rsi)
    andq %r8, %r8
    jle Done
    iaddq $1, %rax
```

修改完成后

执行

make drivers （每次修改 ncopy.ys 时执行）

make psim VERSION=full （每次修改 pipe-full.hcl 时执行）

../misc/yis sdriver.yo （检测 ncopy.ys 与 YIS 一起正常工作）

./correctness.pl （检测正确率）

./benchmark.pl （检测代码性能得分）



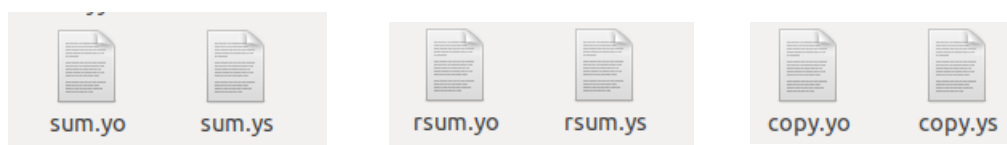
## 四、实验结果及分析

第一部分（Part A）：

编写完三个汇编程序后，进行编译：

```
master@ubuntu:~/sim/misc$ ./yas sum.y
master@ubuntu:~/sim/misc$ ./yas rsum.y
master@ubuntu:~/sim/misc$ ./yas copy.y
```

生成对应的.yo 文件：



执行./yis 对应文件查看运行结果：

```
master@ubuntu:~/sim/misc$ ./yis sum.yo
Stopped in 25 steps at PC = 0x1d. Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax: 0x0000000000000000      0x00000000000000c
%rsp: 0x0000000000000000      0x000000000000020
%r8: 0x0000000000000000      0x000000000000008
%r9: 0x0000000000000000      0x0000000000000c0

Changes to memory:
0x01f8: 0x0000000000000000      0x00000000000001d

master@ubuntu:~/sim/misc$ ./yis rsum.yo
Stopped in 39 steps at PC = 0x27. Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rax: 0x0000000000000000      0x00000000000000c
%rsp: 0x0000000000000000      0x000000000000020

Changes to memory:
0x01c8: 0x0000000000000000      0x000000000000082
0x01d0: 0x0000000000000000      0x0000000000000b0
0x01d8: 0x0000000000000000      0x000000000000082
0x01e0: 0x0000000000000000      0x00000000000000a
0x01e8: 0x0000000000000000      0x000000000000082
0x01f8: 0x0000000000000000      0x000000000000027

master@ubuntu:~/sim/misc$ ./yis copy.yo
Stopped in 35 steps at PC = 0x31. Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax: 0x0000000000000000      0x00000000000000c
%rsp: 0x0000000000000000      0x000000000000020
%rsi: 0x0000000000000000      0x000000000000068
%rdi: 0x0000000000000000      0x000000000000050
%r9: 0x0000000000000000      0x000000000000001
%r10: 0x0000000000000000      0x000000000000008
%r11: 0x0000000000000000      0x0000000000000c0

Changes to memory:
0x0050: 0x0000000000000111      0x00000000000000a
0x0058: 0x0000000000000222      0x0000000000000b0
0x0060: 0x0000000000000333      0x0000000000000c0
0x01f8: 0x0000000000000000      0x000000000000031
```

可以看到%rax 的值是 0xcba，程序运行结果正确，并且各个栈变化也可以看出结果的正确性。

第二部分（Part B）：

在修改完 seq-full.hcl 文件后，进行检测的结果：

执行./ssim -t ../y86-code/asumi.yo

```

master@ubuntu:~/sim/seq$ ./ssim -t ../y86-code/asumi.yo
Y86-64 Processor: seq-full.hcl
137 bytes of code read
IF: Fetched irmovq at 0x0. ra=----, rb=%rsp, valC = 0x100
IF: Fetched call at 0xa. ra=----, rb=----, valC = 0x38
Wrote 0x13 to address 0xf8
IF: Fetched irmovq at 0x38. ra=----, rb=%rdi, valC = 0x18
IF: Fetched irmovq at 0x42. ra=----, rb=%rsi, valC = 0x4
IF: Fetched call at 0x4c. ra=----, rb=----, valC = 0x56
Wrote 0x55 to address 0xf0
IF: Fetched xorq at 0x56. ra=%rax, rb=%rax, valC = 0x0
IF: Fetched andq at 0x58. ra=%rsi, rb=%rsi, valC = 0x0
IF: Fetched jmp at 0x5a. ra=----, rb=----, valC = 0x83
IF: Fetched jne at 0x83. ra=----, rb=----, valC = 0x63
IF: Fetched mrmovq at 0x63. ra=%r10, rb=%rdi, valC = 0x0
IF: Fetched addq at 0x6d. ra=%r10, rb=%rax, valC = 0x0
IF: Fetched iaddq at 0x6f. ra=----, rb=%rdi, valC = 0x8
IF: Fetched iaddq at 0x79. ra=----, rb=%rsi, valC = 0xffffffffffffffff
IF: Fetched jne at 0x83. ra=----, rb=----, valC = 0x63
IF: Fetched mrmovq at 0x63. ra=%r10, rb=%rdi, valC = 0x0
IF: Fetched addq at 0x6d. ra=%r10, rb=%rax, valC = 0x0
IF: Fetched iaddq at 0x6f. ra=----, rb=%rdi, valC = 0x8
IF: Fetched iaddq at 0x79. ra=----, rb=%rsi, valC = 0xffffffffffffffff
IF: Fetched jne at 0x83. ra=----, rb=----, valC = 0x63
IF: Fetched mrmovq at 0x63. ra=%r10, rb=%rdi, valC = 0x0
IF: Fetched addq at 0x6d. ra=%r10, rb=%rax, valC = 0x0
IF: Fetched iaddq at 0x6f. ra=----, rb=%rdi, valC = 0x8
IF: Fetched iaddq at 0x79. ra=----, rb=%rsi, valC = 0xffffffffffffffff
IF: Fetched jne at 0x83. ra=----, rb=----, valC = 0x63
IF: Fetched ret at 0x8c. ra=----, rb=----, valC = 0x0
IF: Fetched ret at 0x55. ra=----, rb=----, valC = 0x0
IF: Fetched halt at 0x13. ra=----, rb=----, valC = 0x0
32 instructions executed
Status = HLT
Condition Codes: Z=1 S=0 O=0
Changed Register State:
%rax: 0x0000000000000000 0x0000abcdabcdabcd
%rsp: 0x0000000000000000 0x0000000000000100
%rdi: 0x0000000000000000 0x0000000000000038
%r10: 0x0000000000000000 0x0000a000a000a000
Changed Memory State:
0x00f0: 0x0000000000000000 0x0000000000000055
0x00f8: 0x0000000000000000 0x0000000000000013
ISA Check Succeeds

```

执行(cd ../y86-code; make testssim)

```
master@ubuntu:~/sim/y86-code$ make testssim
../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
asum.seq:ISA Check Succeeds
asumr.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
```

执行(cd ../ptest; make SIM=../seq/ssim)

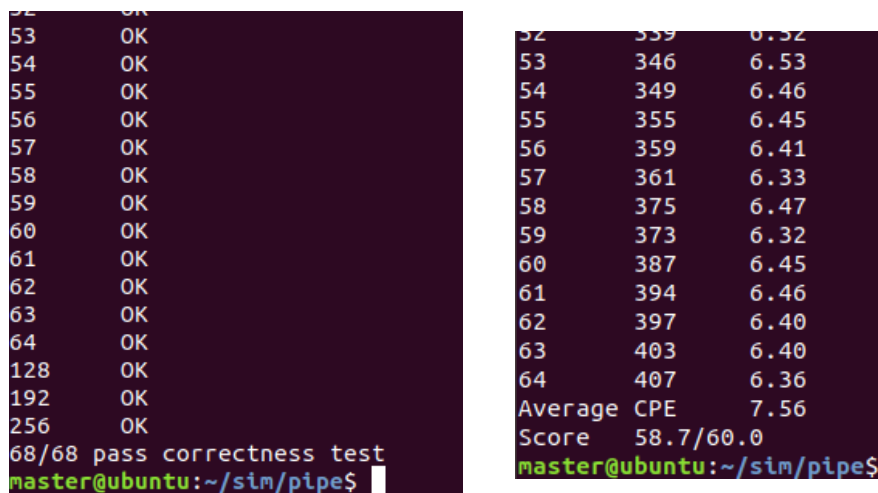
```
master@ubuntu:~/sim/ptest$ make SIM=../seq/ssim
./optest.pl -s ../seq/ssim
Simulating with ../seq/ssim
All 49 ISA Checks Succeed
./jtest.pl -s ../seq/ssim
Simulating with ../seq/ssim
All 64 ISA Checks Succeed
./ctest.pl -s ../seq/ssim
Simulating with ../seq/ssim
All 22 ISA Checks Succeed
./htest.pl -s ../seq/ssim
Simulating with ../seq/ssim
All 600 ISA Checks Succeed
```

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

```
master@ubuntu:~/sim/ptest$ 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
All 756 ISA Checks Succeed
```

### 第三部分 (Part C) :

修改完 pipe-full.hcl 和 ncopy.py 文件后,进行代码的正确性和性能进行检测。



Iteration	CPE
52	339
53	346
54	349
55	355
56	359
57	361
58	375
59	373
60	387
61	394
62	397
63	403
64	407
Average	7.56

Score 58.7/60.0

可以看到

正确性为百分百正确

性能 Average CPE 为 7.56, 分数为 58.7/60

## 五、实验收获与体会

通过本次实验,我更深入了解流水线 Y86-64 处理器的设计和实现,并对优化程序代码有了一定的了解。通过编写一系列汇编程序,让我更加掌握熟练运用汇编代码,在优化方面我对循环展开的方式有了很好的了解,并明白循环展开对代码的效率优化有着很大的帮助。

## 附录：程序清单及说明

sum.ys  
rsum.ys  
copy.ys  
seq-full.hcl  
ncopy.ys  
pipe-full.hcl