实验-优化Y86-64流水线处理器性能报告

一、实验目的

该实验主要是学习流水线`Y86-64`处理器的设计和实现，同时对处理器和基准测试程序进行优化以使性能最大化。通过这次试验来深入的理解处理器的体系结构，了解cpu的构成。

二、实验内容

本次作业作为第4章处理器体系结构的配套实验，以`Y86-64`顺序和流水线处理器仿真器为基础，

对Y86-64处理器进行性能优化。具体优化内容包括以下几个方面：

- Part A：编写`Y86-64`简单程序，熟悉`Y86-64`工具。
- Part B：扩展SEQ模拟器支持新的指令
- 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、实验环境

- Linux系统：Ubuntu 18.04
- `Y86-64`处理器模拟器

四、实验步骤与结果检验

Part A

这部分，要用`Y86-64`汇编代码实现`examples.c`中的三个函数。这三个函数都是与链表有关的操作，链表结点定义如下：

```
/* linked list element */
typedef struct ELE {
    long val;
    struct ELE *next;
} *list_ptr;
```

总共需要实现三个函数：`sum_list`，`rsum_list`，`copy_block`。

sum_list

简单的链表求和。

```
        #name:张子硕
        #ID:1120213300
        .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_list
        ret
sum_list:
        pushq %rbx
        xorq %rax,%rax
        jmp test
loop:
        mrmovq (%rdi),%rbx
        addq %rbx,%rax
        mrmovq 8(%rdi),%rdi
test:
        andq %rdi,%rdi
        jne loop
        popq %rbx
        ret

        .pos 0x200
stack:
```

结果验证：

```
user@user-virtual-machine:~/archlab/sim/misc$ ./yis sum.yo
Stopped in 28 steps at PC = 0x13.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000cba
%rsp:   0x0000000000000000      0x0000000000000200

Changes to memory:
0x01f0: 0x0000000000000000      0x000000000000005b
0x01f8: 0x0000000000000000      0x0000000000000013
```

rsum_list

链表求和的递归实现。

```
  #name:张子硕
  #ID:1120213300

  .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 rsum_list
  ret

rsum_list:
  pushq %rbx
  xorq %rax,%rax
  andq %rdi,%rdi
  je finish
  mrmovq (%rdi),%rbx
  mrmovq 8(%rdi),%rdi
  call rsum_list
  addq %rbx,%rax

finish:
  popq %rbx
  ret

  .pos 0x200
stack:
~
```

结果验证：

```
user@user-virtual-machine:~/archlab/sim/misc$ ./yis rsum.yo
Stopped in 42 steps at PC = 0x13.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%rax:   0x0000000000000000      0x0000000000000cba
%rsp:   0x0000000000000000      0x0000000000000200

Changes to memory:
0x01b8: 0x0000000000000000      0x0000000000000c00
0x01c0: 0x0000000000000000      0x0000000000000088
0x01c8: 0x0000000000000000      0x00000000000000b0
0x01d0: 0x0000000000000000      0x0000000000000088
0x01d8: 0x0000000000000000      0x000000000000000a
0x01e0: 0x0000000000000000      0x0000000000000088
0x01f0: 0x0000000000000000      0x000000000000005b
0x01f8: 0x0000000000000000      0x0000000000000013
```

copy_block

数组赋值操作。

```
 #name:张子硕
  #ID:1120213300

  .pos 0
  irmovq stack,%rsp
  call main
  halt

  .align 8
src:
  .quad 0x00a
  .quad 0x0b0
  .quad 0xc00

dest:
  .quad 0x111
  .quad 0x222
  .quad 0x333

main:
  irmovq src,%rdi
  irmovq dest,%rsi
  irmovq $3,%rdx
  call copy_block
  ret

copy_block:
  pushq %rbx
  pushq %r12
  pushq %r13
  xorq %rax,%rax
  irmovq $8,%r12
  irmovq $1,%r13

loop:
  andq %rdx,%rdx
  jle finish
  mrmovq (%rdi),%rbx
  rmmovq %rbx,(%rsi)
  xorq %rbx,%rax
  addq %r12,%rdi
  addq %r12,%rsi
  subq %r13,%rdx
  jmp loop

finish:
```

```
  popq %r13
  popq %r12
  popq %rbx
  ret

  .pos 0x200
stack:
```

结果验证：

```
user@user-virtual-machine:~/archlab/sim/misc$ ./yis copy.yo
Stopped in 47 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

Changes to memory:
0x0030: 0x0000000000000111      0x000000000000000a
0x0038: 0x0000000000000222      0x00000000000000b0
0x0040: 0x0000000000000333      0x0000000000000c00
0x01f0: 0x0000000000000000      0x000000000000006f
0x01f8: 0x0000000000000000      0x0000000000000013
```

Part B

在这个部分里，我们要修改`seq-full.hcl`文件中的代码来实现`iaddq`操作进行直接的立即数赋值给寄存器。我们对每个阶段分析修改信号。

取指阶段

`instr_valid`：判断指令是否合法，当然应该加上。修改后为

```
bool instr_valid = icode in
    { INOP, IHALT, IRRMOVQ, IIRMOVQ, IRMMOVQ, IMRMOVQ,
           IOPQ, IJXX, ICALL, IRET, IPUSHQ, IPOPQ, IIADDQ };
```

`need_regids`：判断指令是否包括寄存器指示符字节，当然也应该加上

```
bool need_regids =
    icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,
               IIRMOVQ, IRMMOVQ, IMRMOVQ, IIADDQ };
```

`need_valC`：判断指令是否包括常数字，还是要加上

```
bool need_valC =
    icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL, IIADDQ };
```

译码阶段和写回阶段

`srcB`：赋为产生`valB`的寄存器。译码阶段要从`rA`，`rB`指明的寄存器读为`valA`，`valB`，而`iaddq`有一个`rB`，于是有以下修改

```
word srcB = [
    icode in { IOPQ, IRMMOVQ, IMRMOVQ, IIADDQ  } : rB;
    icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;
    1 : RNONE;  # Don't need register
];
```

`dst_E`：表明写端口E的目的寄存器，计算出来的值`valE`将放在那里。最终结果要存放在`rB`中，所以要修改

```
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
];
```

执行阶段

执行阶段`ALU`要对`aluA`和`aluB`进行计算，计算格式为：`aluB OP aluA`。所以`aluA`可以是`valA`和`valC`或者`+-8`，`aluB`只能是`valB`。而`iaddq`执行阶段进行的运算是`valB + valC`，于是可知修改

```
## Select input A to ALU
word aluA = [
    icode in { IRRMOVQ, IOPQ } : valA;
    icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IIADDQ } : valC;
    icode in { ICALL, IPUSHQ } : -8;
    icode in { IRET, IPOPQ } : 8;
    # Other instructions don't need ALU
];

## Select input B to ALU
word aluB = [
    icode in { IRMMOVQ, IMRMOVQ, IOPQ, ICALL,
               IPUSHQ, IRET, IPOPQ, IIADDQ } : valB;
    icode in { IRRMOVQ, IIRMOVQ } : 0;
    # Other instructions don't need ALU
];
```

`set_cc`：判断是否应该更新条件码寄存器

```
bool set_cc = icode in { IOPQ, IIADDQ };
```

访存阶段

无需更改

更新PC阶段

无需更改

结果检验：

- 在`Y86-64`程序中

```
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 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
```

- 基准程序

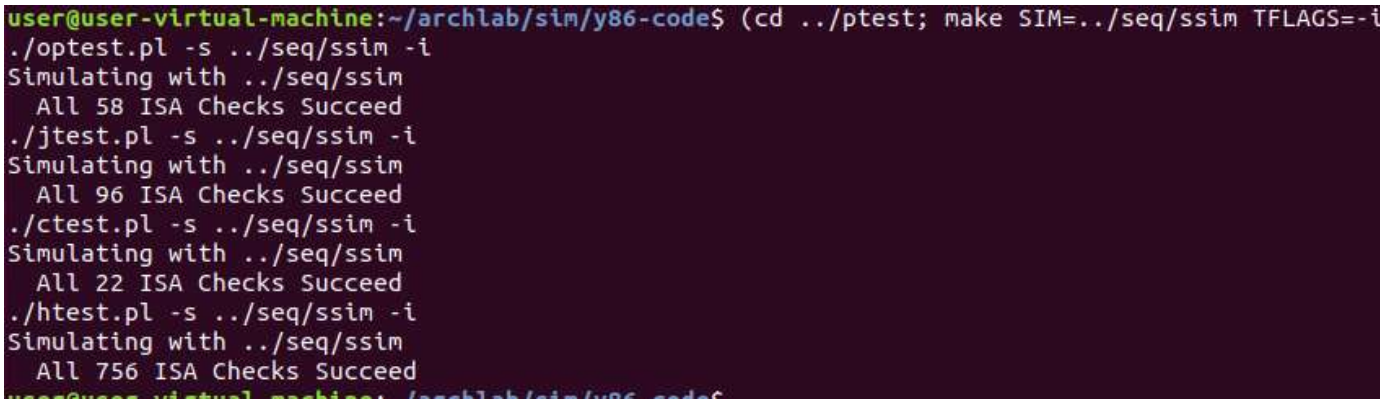
```
user@user-virtual-machine:~/archlab/sim/seq$ cd ../y86-code; make testssim
../seq/ssim -t asum.yo > asum.seq
../seq/ssim -t asumr.yo > asumr.seq
../seq/ssim -t cjr.yo > cjr.seq
../seq/ssim -t j-cc.yo > j-cc.seq
../seq/ssim -t poptest.yo > poptest.seq
../seq/ssim -t pushquestion.yo > pushquestion.seq
../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 pro
.seq prog6.seq prog7.seq prog8.seq ret-hazard.seq
user@user-virtual-machine:~/archlab/sim/y86-code$
```

- 大量回归测试

  除 `iaddq` 之外的所有指令

```
user@user-virtual-machine:~/archlab/sim/y86-code$ (cd ../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
user@user-virtual-machine:~/archlab/sim/y86-code$
```

  专门测试 `iaddq`



Part C

该部分在`sim/pipe`中进行，需要我们修改`ncopy.ys`使得`ncopy`函数尽可能块，也可以修改`pipe-full.hcl`文件来增加`iaddq`指令。
先增加`iaddq`指令，过程同Part B，省略。
整体思路：

- 我们使用`iaddq`来替代代码里所有立即数加减法指令。
- 将循环六路展开，加快处理大批量数据的速度。
- 添加小型的循环三路展开，更适配数据量小的情况。
- 我们多使用一个寄存器，两次复制来消除气泡。

代码如下：

```
#/* $begin ncopy-ys */
#
#
#name:张子硕
#ID:1120213300
#
##################################################################
# ncopy.ys - Copy a src block of len words to dst.
# Return the number of positive words (>0) contained in src.
#
# Include your name and ID here.
#
# Describe how and why you modified the baseline code.
#
##################################################################
# Do not modify this portion
# Function prologue.
# %rdi = src, %rsi = dst, %rdx = len
ncopy:

##################################################################
# You can modify this portion
        # Loop header
            andq %rdx,%rdx      # len <= 0?
    jmp test
Loop:
    mrmovq (%rdi),%r8
    mrmovq 8(%rdi),%r9
    andq %r8,%r8
    rmmovq %r8,(%rsi)
    rmmovq %r9,8(%rsi)
    jle Loop1
    iaddq $1,%rax
Loop1:
    andq %r9,%r9
    jle Loop2
    iaddq $1,%rax
Loop2:
    mrmovq 16(%rdi),%r8
    mrmovq 24(%rdi),%r9
    andq %r8,%r8
    rmmovq %r8,16(%rsi)
    rmmovq %r9,24(%rsi)
    jle Loop3
    iaddq $1,%rax
Loop3:
```

```
    andq %r9,%r9
    jle Loop4
    iaddq $1,%rax
Loop4:
    mrmovq 32(%rdi),%r8
    mrmovq 40(%rdi),%r9
    andq %r8,%r8
    rmmovq %r8,32(%rsi)
    rmmovq %r9,40(%rsi)
    jle Loop5
    iaddq $1,%rax
Loop5:
    iaddq $48,%rdi
    iaddq $48,%rsi
    andq %r9,%r9
    jle test
    iaddq $1,%rax
test:
    iaddq $-6, %rdx          # 先减，判断够不够6个
    jge Loop                 # 6路展开
    iaddq $6, %rdx
    jmp test2                #剩下的

L:
    mrmovq (%rdi),%r8
    andq %r8,%r8
    rmmovq %r8,(%rsi)
    jle L1
    iaddq $1,%rax
L1:
    mrmovq 8(%rdi),%r8
    andq %r8,%r8
    rmmovq %r8,8(%rsi)
    jle L2
    iaddq $1,%rax
L2:
    mrmovq 16(%rdi),%r8
    iaddq $24,%rdi
    rmmovq %r8,16(%rsi)
    iaddq $24,%rsi
    andq %r8,%r8
    jle test2
    iaddq $1,%rax
test2:
    iaddq $-3, %rdx          # 先减，判断够不够3个
    jge L
```

```
    iaddq $2, %rdx          # -1则不剩了，直接Done,0 剩一个, 1剩2个
    je R0
    jl Done
    mrmovq (%rdi),%r8
    mrmovq 8(%rdi),%r9
    rmmovq %r8,(%rsi)
    rmmovq %r9,8(%rsi)
    andq %r8,%r8
    jle R2
    iaddq $1,%rax
R2:
    andq %r9,%r9
    jle Done
    iaddq $1,%rax
    jmp Done
R0:
    mrmovq (%rdi),%r8
    andq %r8,%r8
    rmmovq %r8,(%rsi)
    jle Done
    iaddq $1,%rax
##################################################################
# Do not modify the following section of code
# Function epilogue.
Done:
        ret
##################################################################
# Keep the following label at the end of your function
End:
#/* $end ncopy-ys */
```

结果检验：

```
22      OK
23      OK
24      OK
25      OK
26      OK
27      OK
28      OK
29      OK
30      OK
31      OK
32      OK
33      OK
34      OK
35      OK
36      OK
37      OK
38      OK
39      OK
40      OK
41      OK
42      OK
43      OK
44      OK
45      OK
46      OK
47      OK
48      OK
49      OK
50      OK
51      OK
52      OK
53      OK
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
user@user-virtual-machine:~/archlab/sim/pipe$
```

```
26      189     7.27
27      196     7.26
28      201     7.18
29      209     7.21
30      212     7.07
31      214     6.90
32      225     7.03
33      233     7.06
34      238     7.00
35      247     7.06
36      249     6.92
37      251     6.78
38      263     6.92
39      270     6.92
40      275     6.88
41      284     6.93
42      286     6.81
43      288     6.70
44      299     6.80
45      307     6.82
46      312     6.78
47      320     6.81
48      323     6.73
49      325     6.63
50      336     6.72
51      344     6.75
52      349     6.71
53      357     6.74
54      360     6.67
55      362     6.58
56      373     6.66
57      381     6.68
58      386     6.66
59      394     6.68
60      397     6.62
61      399     6.54
62      411     6.63
63      418     6.63
64      423     6.61
Average CPE     8.16
Score   46.8/60.0
user@user-virtual-machine:~/archlab/sim/pipe$
```