此lab主要是针对CSAPP第四章的内容，主要是学习设计和优化Y86-64处理器，分成了3个部分，第一部分熟悉Y86-64指令，第二部分熟悉pipeline的设计，第三部分结合前两部分综合优化，获得尽可能低的CPE。

**优化建议**

我相信大部分人都是来看partC的优化的，所以先说说优化思路，按着思路再自己试试，实在想不出来再看答案。

1. 必须添加iaddq指令；
2. 此lab，%rax默认为0，不需要xor；
3. 循环展开；
4. 有数据冒险，可以通过psim -g调试发现冒险的地方，插入其他指令规避；
5. 有控制冒险，要合理安排跳转；
6. 一个测试条件可以同时跟jl、je、jg；
7. 着重优化非循环部分/数据少的部分；
8. 需要用到三叉树；
9. 剧透一下，hcl文件只改iaddq，通过优化ncopy代码是可以得满分的。

进阶优化：

1. 修改hcl文件，解决掉数据冒险，可以看书的家庭作业4.57。

**配置问题**

或者是环境配置问题，毕竟lab很老了，很多东西都更新了，可以根据出现的错误提示一步步解决。

1、可能需要安装tcl、tk、flex、bison

sudo apt-get install tcl-dev

sudo apt-get install tk-dev

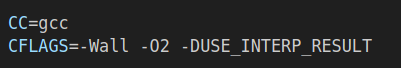
sudo apt-get install flex bison

2、可能要修改Makefile

Makefile中的TKINC的tcl头文件路径是tcl8.5，目前系统下载的是8.6，以后可能还会更新，改成对应的版本号。

3、可能代码发生了改变

tcl的代码结构发生了变化，改变了使用方法，不过可以添加-DUSE\_INTERP\_RESULT到Makefile中解决，忽略相关的警告。



​

4、可能提示找不到变量matherr

把它注释掉就好，



​

**第一部分**

将examples.c中的三个函数改写成Y86-64汇编程序，这个不难，没啥好说的，直接上代码吧。

sum.ys

#Execution begin at address 0

.pos 0

irmovq stack,%rsp

call main

halt

# Sample linked list

.align 8

ele1:

.quad 0x00a

.quad ele2

ele2:

.quad 0x0b1

.quad ele3

ele3:

.quad 0xc00

.quad 0

main:

irmovq ele1,%rdi

call sum\_list

ret

sum\_list:

xorq %rax,%rax

jmp test

loop:

mrmovq (%rdi),%rsi

addq %rsi,%rax

mrmovq $8(%rdi),%rdi

test:

andq %rdi,%rdi

jne loop

ret

#stack starts here and grows to lower address

.pos 0x200

stack:

rsum.ys

#Execution begin at address 0

.pos 0

irmovq stack,%rsp

call main

halt

# Sample linked list

.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 %rsi

xorq %rax,%rax

andq %rdi,%rdi

je retn

mrmovq (%rdi),%rsi

mrmovq $8(%rdi),%rdi

call rsum\_list

addq %rsi,%rax

retn:

popq %rsi

ret

#stack starts here and grows to lower address

.pos 0x200

stack:

copy\_block.ys

#Execution begin at address 0

.pos 0

irmovq stack,%rsp

call main

halt

.align 8

# Source block

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 copyblock

ret

copyblock:

xorq %rax,%rax

irmovq $8,%r8

irmovq $1,%r9

andq %rdx,%rdx

jmp test

loop:

mrmovq (%rdi),%rcx

rmmovq %rcx,(%rsi)

addq %r8,%rdi

addq %r8,%rsi

xorq %rcx,%rax

subq %r9,%rdx

test:

jne loop

ret

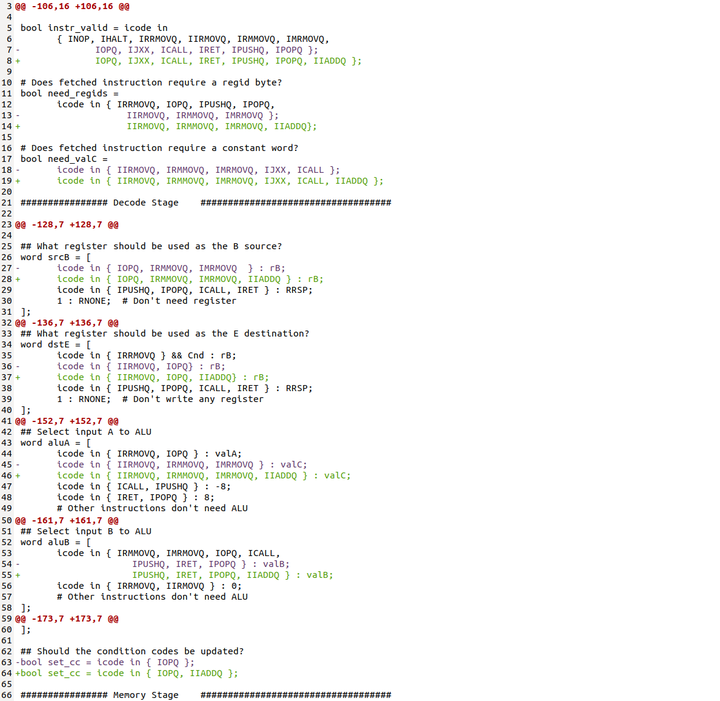
#stack starts here and grows to lower address

.pos 0x200

stack:

**第二部分**

添加iaddq指令，目的是熟悉hcl语言，后续要用到iaddq。具体改动看下面的diff，这个也不算难，理解了hcl可以很容易搞定。



​

**第三部分**

这部分才是这个lab的重头戏，通过修改ncopy.ys和pipe-full.hcl让ncopy.ys跑的尽可能快，几个要求是：

1. ncopy要适用任意数组大小；
2. ncopy要能过YIS；
3. ncopy的目标文件不能大于1000字节；
4. pipe-full.hcl要能过../y86-code 和 ../ptest的测试，尤其是这点，benchmark不检查对错，如果psim有问题，benchmark可能会得到很低的CPE，误以为自己得了满分。

**3.1、代码和模拟器测试：**

1、保证代码正确：

./correctness.pl

2、保证模拟器正确

(cd ../y86-code; make testpsim)

(cd ../ptest; make SIM=../pipe/psim TFLAGS=-i)

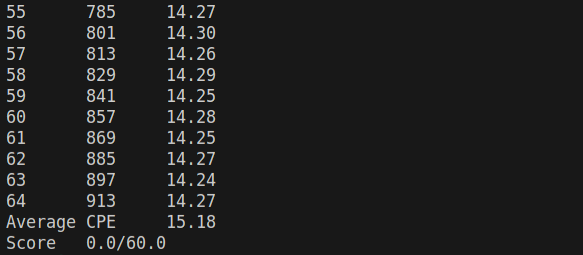
3、保证二者结合正确

./correctness.pl -p

通过了上面的测试，才能保证代码和模拟器都OK。

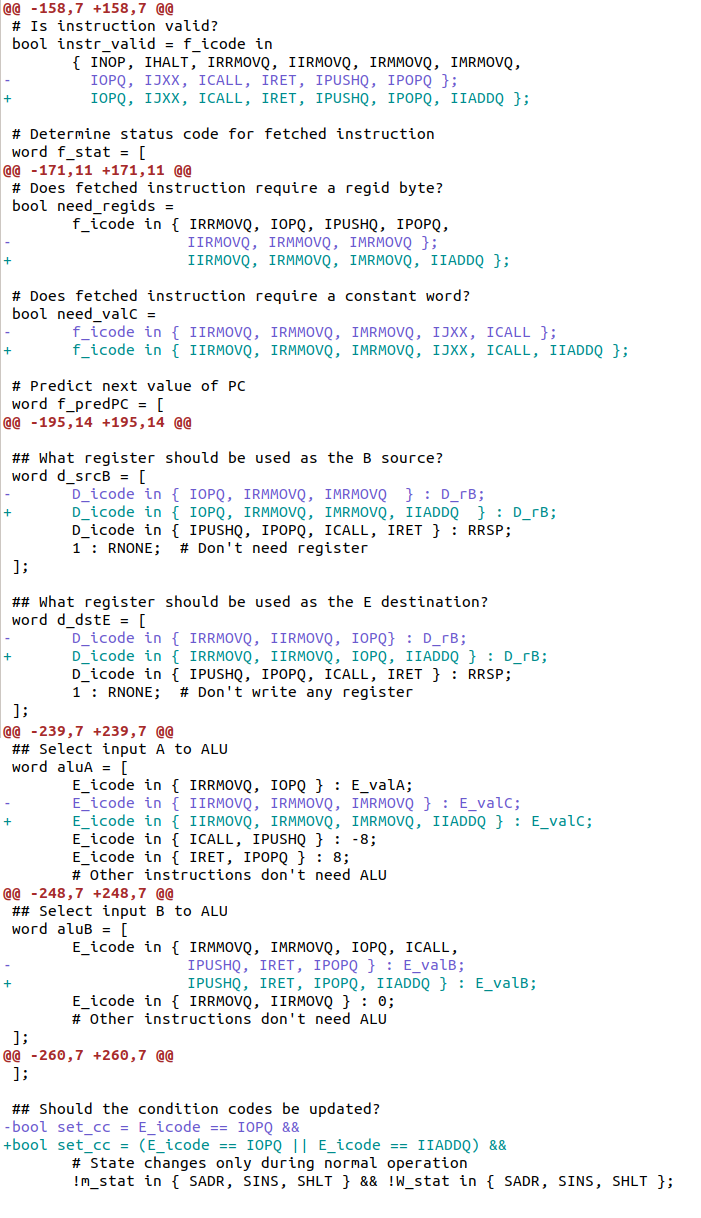
运行./benchmark.pl查看最终的CPE和得分。

**3.2、原始ncopy.ys和pipe-full.hcl得分**

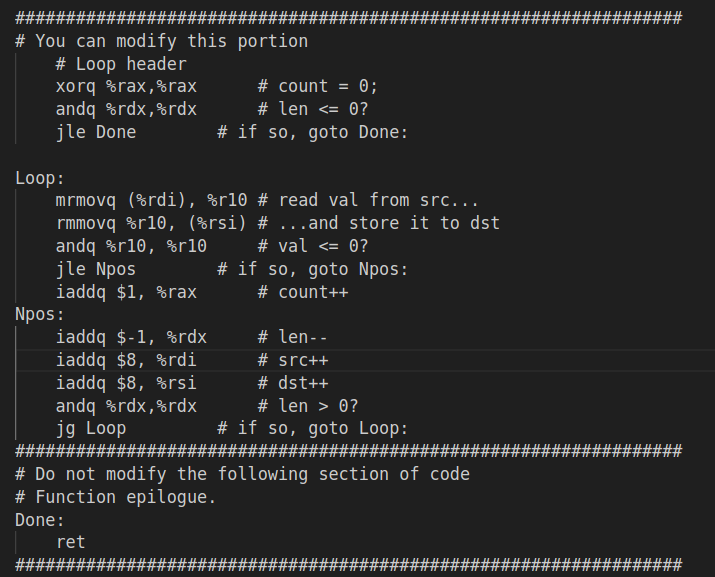


​

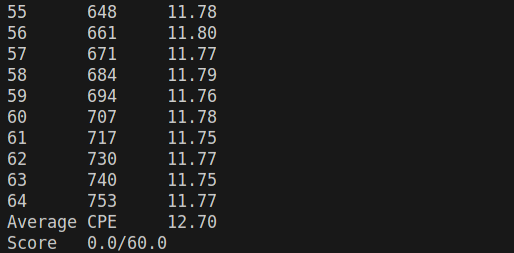
**3.3、hcl添加iaddq指令，ncopy.ys改为iaddq指令，速度明显提升；**



​



​

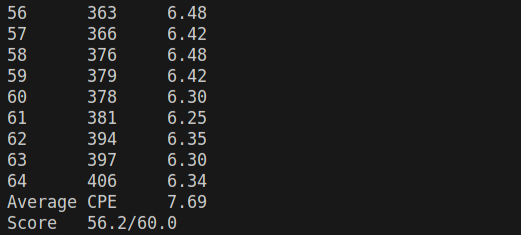


​

**3.4、循环展开**

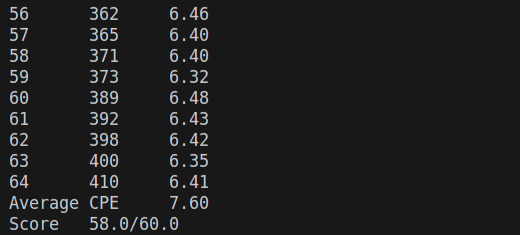
到此，需要进一步对代码动手了，方法是第五章的循环展开，由于目标代码长度1000bytes的限制，最多只能展开到10。另外，针对此lab，%rax默认为0，可以去掉xorq %rax,%rax这句，不过实际应用不能这样。

刚开始的思路是先展开（8-10）次循环，剩下的展开4次循环，之后2次，最后1次，（简称n421，以后都这么叫）通过判断rdx选择执行的路径组合，展开10次（10421）的CPE能到7.69，提升明显。代码见附1。



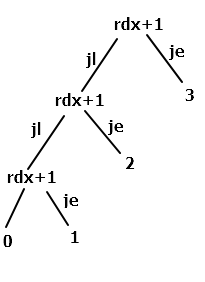
​

对于以上结果不满意，复制3个数据会分成2+1，这比较慢，想想改为n4321，3也独立出来，这次展开8次最快，CPE到7.60。代码见附2。



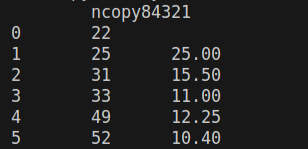
​

仍然不满意，4以下的路径是3-2-1-0，



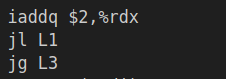
​

下图可知1的权重很大，减少它的CPE很关键。



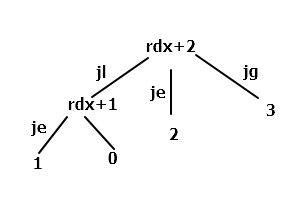
​

汇编可以一个条件执行多个跳转，而跳转有小于（jl）、等于（je）和大于（jg），



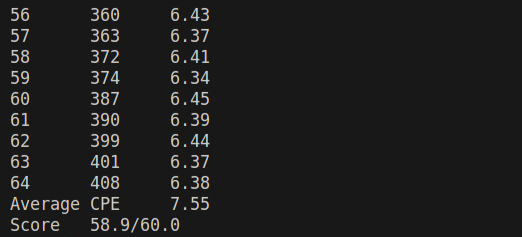
​

所以可以用三叉树构建跳转路径，减少指令数。



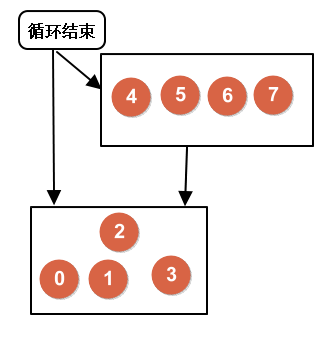
​

因为处理器有分支预测，并且默认是要跳转的，所以要把概率大的或者紧要的部分紧挨着测试条件，因为一旦预测错误，就会多出两条bubble。按照上面的树，构造出n4r213（r2表示root2，2是根，左在前，右在后，0第一个不算CPE，永远放到分支最后，而且可能会跟10冲突，故省略），我测试84r213最快，得到下面的结果，可见优化有效，CPE已经非常接近7.5了。这里还可以以1为根，感兴趣的小伙伴可以自己测试，看看是快是慢。代码见附3.



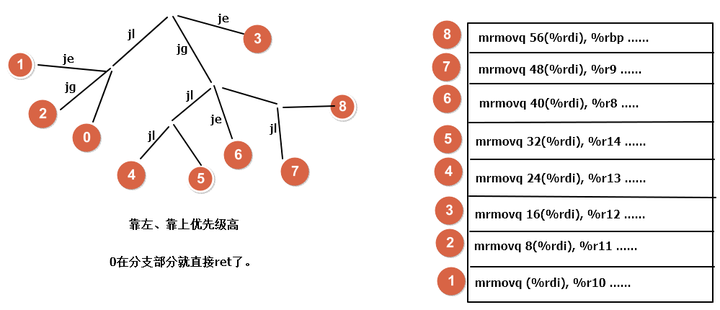
​

就差1.1就满分了，继续努力。我们发现循环展开部分已经无法优化了（最多是8、9、10的差别），后边1、2、3也到头了，问题就在中间的4了。之前的代码都是小于4的直接执行0、1、2、3的部分，大于等于4的先执行4次展开，再根据结果执行0、1、2、3，这里就拖慢了性能，我们换个思路进一步优化。



​

这次我们把非循环部分都独立出来，用上面树的思想得到具体的数，然后直接跳到对应的地方执行，如下图所示。这里以3为根，原因还是要尽量靠近循环次数少的，尽可能减少其指令数。（CPE=指令数/数据个数，数据越少，非复制部分占比越大。这里的定义好像跟第五章的CPE有点差别，我们知道这里啥意思就可以了）。



​

切换为居中

分支部分的代码如下所示（9r31264578）：

BR:

iaddq $6,%rdx

jl Left

jg Right

jeL3

L45:

iaddq $1,%rdx

jl L4

jmp L5

Left:

iaddq $2,%rdx

jeL1

jgL2

ret

Right:

iaddq $-3,%rdx

jlL45

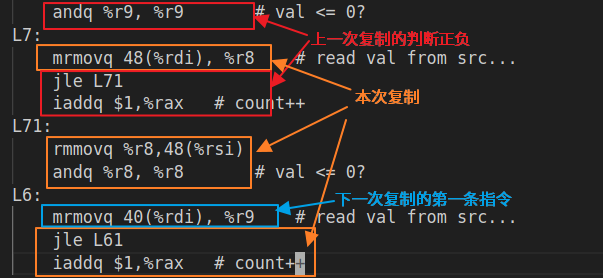
jeL6

iaddq $-2,%rdx

jl L7

mrmovq 56(%rdi), %r9#L8直接落下来执行

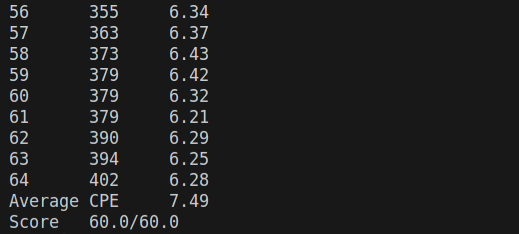
上图右边只是示意图，具体实现还需要优化，因为mrmovq和rmmovq 如果前后挨着，会有装载/使用冒险，第二条rmmovq指令会产生一个暂停，这会降低性能，可以使用加载转发技术，修改hcl文件解决这个问题，不过我们先不动hcl文件，想办法在它们之间插入一条其他指令，取代这个暂停就好，具体办法是把上一条指令的判断正负插入他们中间，如下：（可读性非常差，一不小心就混乱了，一定要仔细）



​

切换为居中

通过以上努力，我们得到了最终的版本，实测展开9次和10次CPE=7.49，满分啦。代码见附4。

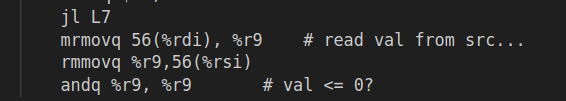


​

到此，只增加iaddq指令的优化就做到这里了，或许还能提高，就交给感兴趣的小伙伴自行尝试吧。

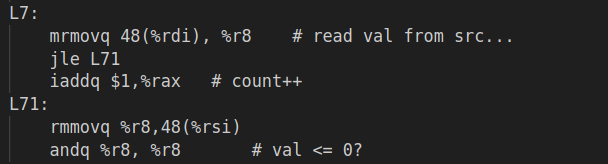
**3.5、加载转发**

之前的版本虽然满分但还是有缺陷，或者是不可避免的会有一个暂停（如下图，rdx为8），



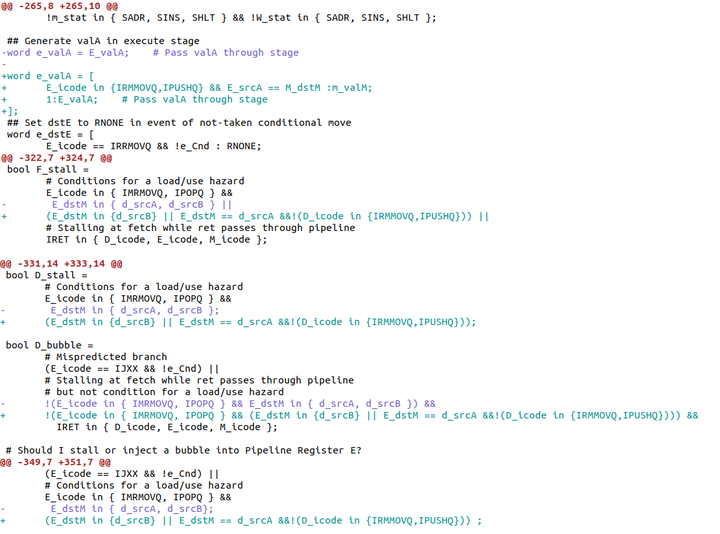
​

或者是执行一条没用的跳转（如下图，rdx为7，跳到这里），



​

这都是装载/使用冒险带来的副作用，看来必须把它解决掉才能得到完美的版本。加载转发具体原理可以看原书家庭作业4.57，我们在添加iaddq指令的pipe-ful.hcl文件中进一步做以下修改：

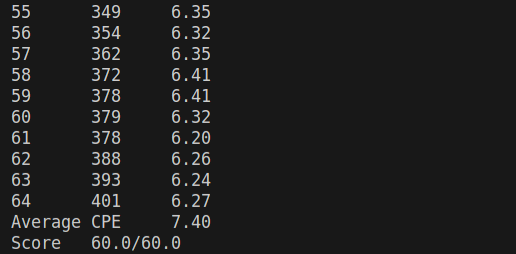


​

重新make并测试，都OK就得到了改进版的psim。

这样在ncopy里写代码就没有之前的限制了，mrmovq和rmmovq可以挨着，没有暂停。这里我人为加了个要求，就是寄存器只用调用者保存的，因为实际写程序，用被调用者保存寄存器是需要入栈和出栈的，这里虽没有要求，但规范还是要的。

此时的性能瓶颈就只在分支的安排上了，展开越多，分支的可能性越多，脑子想哪个更快难度很大，具体效果还需要实验，我实测展开10次最好，分支是10r312645789，CPE=7.40，这已经超过文档中老师的最好7.48，其他分支情况可能是7.44、7.45、7.46，差距不是太大。我都怀疑自己是不是弄错了，希望小伙伴们也试试，看能否复现。



​

**最优版本ncopy代码如下：**

#/\* $begin ncopy-ys \*/

##################################################################

# 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

#xorq %rax,%rax# count = 0;

iaddq $-10,%rdx

jl BR

Loop10:

mrmovq (%rdi), %r10# read val from src...

rmmovq %r10, (%rsi)# ...and store it to dst

andq %r10, %r10# val <= 0?

jle n1

iaddq $1,%rax # count++

n1:

mrmovq 8(%rdi), %r11# read val from src...

rmmovq %r11, 8(%rsi)# ...and store it to dst

andq %r11, %r11# val <= 0?

jle n2

iaddq $1,%rax # count++

n2:

mrmovq 16(%rdi), %r10# read val from src...

rmmovq %r10, 16(%rsi)# ...and store it to dst

andq %r10, %r10# val <= 0?

jle n3

iaddq $1,%rax # count++

n3:

mrmovq 24(%rdi), %r11# read val from src...

rmmovq %r11, 24(%rsi)# ...and store it to dst

andq %r11, %r11# val <= 0?

jle n4

iaddq $1,%rax # count++

n4:

mrmovq 32(%rdi), %r10# read val from src...

rmmovq %r10, 32(%rsi)# ...and store it to dst

andq %r10, %r10# val <= 0?

jle n5

iaddq $1,%rax # count++

n5:

mrmovq 40(%rdi), %r11# read val from src...

rmmovq %r11, 40(%rsi)# ...and store it to dst

andq %r11, %r11# val <= 0?

jle n6

iaddq $1,%rax # count++

n6:

mrmovq 48(%rdi), %r10# read val from src...

rmmovq %r10, 48(%rsi)# ...and store it to dst

andq %r10, %r10# val <= 0?

jle n7

iaddq $1,%rax # count++

n7:

mrmovq 56(%rdi), %r11# read val from src...

rmmovq %r11, 56(%rsi)# ...and store it to dst

andq %r11, %r11# val <= 0?

jle n8

iaddq $1,%rax # count++

n8:

mrmovq 64(%rdi), %r10# read val from src...

rmmovq %r10, 64(%rsi)# ...and store it to dst

andq %r10, %r10# val <= 0?

jle n9

iaddq $1,%rax # count++

n9:

mrmovq 72(%rdi), %r10# read val from src...

rmmovq %r10, 72(%rsi)# ...and store it to dst

andq %r10, %r10# val <= 0?

jle n10

iaddq $1,%rax # count++

n10:

iaddq $80,%rdi

iaddq $80,%rsi

iaddq $-10,%rdx

jge Loop10#

BR:

iaddq $7,%rdx

jl Left

jg Right

jeL3

L45:

iaddq $1,%rdx

jl L4

jmp L5

Left:

iaddq $2,%rdx

jeL1

jgL2

ret

Right:

iaddq $-3,%rdx

jlL45

jeL6

iaddq $-2,%rdx

jl L7

je L8

L9:

mrmovq 64(%rdi), %r10# read val from src...

rmmovq %r10,64(%rsi)

andq %r10, %r10# val <= 0?

jle L8

iaddq $1,%rax # count++

L8:

mrmovq 56(%rdi), %r10# read val from src...

rmmovq %r10,56(%rsi)

andq %r10, %r10# val <= 0?

jle L7

iaddq $1,%rax # count++

L7:

mrmovq 48(%rdi), %r10# read val from src...

rmmovq %r10,48(%rsi)

andq %r10, %r10# val <= 0?

jle L6

iaddq $1,%rax # count++

L6:

mrmovq 40(%rdi), %r10# read val from src...

rmmovq %r10,40(%rsi)

andq %r10, %r10# val <= 0?

jle L5

iaddq $1,%rax # count++

L5:

mrmovq 32(%rdi), %r10# read val from src...

rmmovq %r10,32(%rsi)

andq %r10, %r10# val <= 0?

jle L4

iaddq $1,%rax # count++

L4:

mrmovq 24(%rdi), %r10# read val from src...

rmmovq %r10,24(%rsi)

andq %r10, %r10# val <= 0?

jle L3

iaddq $1,%rax # count++

L3:

mrmovq 16(%rdi), %r10# read val from src...

rmmovq %r10,16(%rsi)

andq %r10, %r10# val <= 0?

jle L2

iaddq $1,%rax # count++

L2:

mrmovq 8(%rdi), %r10# read val from src...

rmmovq %r10,8(%rsi)

andq %r10, %r10# val <= 0?

jle L1

iaddq $1,%rax # count++

L1:

mrmovq (%rdi), %r10# read val from src...

rmmovq %r10,(%rsi)

andq %r10, %r10# val <= 0?

jle Done

iaddq $1,%rax # count++

##################################################################

# 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 \*/

**最优版本pipe-full.hcl如下：**

#/\* $begin pipe-all-hcl \*/

####################################################################

# HCL Description of Control for Pipelined Y86-64 Processor #

# Copyright (C) Randal E. Bryant, David R. O'Hallaron, 2014 #

####################################################################

## 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

####################################################################

# C Include's. Don't alter these #

####################################################################

quote '#include <stdio.h>'

quote '#include "isa.h"'

quote '#include "pipeline.h"'

quote '#include "stages.h"'

quote '#include "sim.h"'

quote 'int sim\_main(int argc, char \*argv[]);'

quote 'int main(int argc, char \*argv[]){return sim\_main(argc,argv);}'

####################################################################

# Declarations. Do not change/remove/delete any of these #

####################################################################

##### Symbolic representation of Y86-64 Instruction Codes #############

wordsig INOP 'I\_NOP'

wordsig IHALT'I\_HALT'

wordsig IRRMOVQ'I\_RRMOVQ'

wordsig IIRMOVQ'I\_IRMOVQ'

wordsig IRMMOVQ'I\_RMMOVQ'

wordsig IMRMOVQ'I\_MRMOVQ'

wordsig IOPQ'I\_ALU'

wordsig IJXX'I\_JMP'

wordsig ICALL'I\_CALL'

wordsig IRET'I\_RET'

wordsig IPUSHQ'I\_PUSHQ'

wordsig IPOPQ'I\_POPQ'

# Instruction code for iaddq instruction

wordsig IIADDQ'I\_IADDQ'

##### Symbolic represenations of Y86-64 function codes #####

wordsig FNONE 'F\_NONE' # Default function code

##### Symbolic representation of Y86-64 Registers referenced #####

wordsig RRSP 'REG\_RSP' # Stack Pointer

wordsig RNONE 'REG\_NONE' # Special value indicating "no register"

##### ALU Functions referenced explicitly ##########################

wordsig ALUADD'A\_ADD' # ALU should add its arguments

##### Possible instruction status values #####

wordsig SBUB'STAT\_BUB'# Bubble in stage

wordsig SAOK'STAT\_AOK'# Normal execution

wordsig SADR'STAT\_ADR'# Invalid memory address

wordsig SINS'STAT\_INS'# Invalid instruction

wordsig SHLT'STAT\_HLT'# Halt instruction encountered

##### Signals that can be referenced by control logic ##############

##### Pipeline Register F ##########################################

wordsig F\_predPC 'pc\_curr->pc' # Predicted value of PC

##### Intermediate Values in Fetch Stage ###########################

wordsig imem\_icode 'imem\_icode' # icode field from instruction memory

wordsig imem\_ifun 'imem\_ifun' # ifun field from instruction memory

wordsig f\_icode'if\_id\_next->icode' # (Possibly modified) instruction code

wordsig f\_ifun'if\_id\_next->ifun' # Fetched instruction function

wordsig f\_valC'if\_id\_next->valc' # Constant data of fetched instruction

wordsig f\_valP'if\_id\_next->valp' # Address of following instruction

boolsig imem\_error 'imem\_error' # Error signal from instruction memory

boolsig instr\_valid 'instr\_valid' # Is fetched instruction valid?

##### Pipeline Register D ##########################################

wordsig D\_icode 'if\_id\_curr->icode' # Instruction code

wordsig D\_rA 'if\_id\_curr->ra' # rA field from instruction

wordsig D\_rB 'if\_id\_curr->rb' # rB field from instruction

wordsig D\_valP 'if\_id\_curr->valp' # Incremented PC

##### Intermediate Values in Decode Stage #########################

wordsig d\_srcA 'id\_ex\_next->srca' # srcA from decoded instruction

wordsig d\_srcB 'id\_ex\_next->srcb' # srcB from decoded instruction

wordsig d\_rvalA 'd\_regvala' # valA read from register file

wordsig d\_rvalB 'd\_regvalb' # valB read from register file

##### Pipeline Register E ##########################################

wordsig E\_icode 'id\_ex\_curr->icode' # Instruction code

wordsig E\_ifun 'id\_ex\_curr->ifun' # Instruction function

wordsig E\_valC 'id\_ex\_curr->valc' # Constant data

wordsig E\_srcA 'id\_ex\_curr->srca' # Source A register ID

wordsig E\_valA 'id\_ex\_curr->vala' # Source A value

wordsig E\_srcB 'id\_ex\_curr->srcb' # Source B register ID

wordsig E\_valB 'id\_ex\_curr->valb' # Source B value

wordsig E\_dstE 'id\_ex\_curr->deste' # Destination E register ID

wordsig E\_dstM 'id\_ex\_curr->destm' # Destination M register ID

##### Intermediate Values in Execute Stage #########################

wordsig e\_valE 'ex\_mem\_next->vale'# valE generated by ALU

boolsig e\_Cnd 'ex\_mem\_next->takebranch' # Does condition hold?

wordsig e\_dstE 'ex\_mem\_next->deste' # dstE (possibly modified to be RNONE)

##### Pipeline Register M #########################

wordsig M\_stat 'ex\_mem\_curr->status' # Instruction status

wordsig M\_icode 'ex\_mem\_curr->icode'# Instruction code

wordsig M\_ifun 'ex\_mem\_curr->ifun'# Instruction function

wordsig M\_valA 'ex\_mem\_curr->vala' # Source A value

wordsig M\_dstE 'ex\_mem\_curr->deste'# Destination E register ID

wordsig M\_valE 'ex\_mem\_curr->vale' # ALU E value

wordsig M\_dstM 'ex\_mem\_curr->destm'# Destination M register ID

boolsig M\_Cnd 'ex\_mem\_curr->takebranch'# Condition flag

boolsig dmem\_error 'dmem\_error' # Error signal from instruction memory

##### Intermediate Values in Memory Stage ##########################

wordsig m\_valM 'mem\_wb\_next->valm'# valM generated by memory

wordsig m\_stat 'mem\_wb\_next->status'# stat (possibly modified to be SADR)

##### Pipeline Register W ##########################################

wordsig W\_stat 'mem\_wb\_curr->status' # Instruction status

wordsig W\_icode 'mem\_wb\_curr->icode'# Instruction code

wordsig W\_dstE 'mem\_wb\_curr->deste'# Destination E register ID

wordsig W\_valE 'mem\_wb\_curr->vale' # ALU E value

wordsig W\_dstM 'mem\_wb\_curr->destm'# Destination M register ID

wordsig W\_valM 'mem\_wb\_curr->valm'# Memory M value

####################################################################

# Control Signal Definitions. #

####################################################################

################ Fetch Stage ###################################

## What address should instruction be fetched at

word f\_pc = [

# Mispredicted branch. Fetch at incremented PC

M\_icode == IJXX && !M\_Cnd : M\_valA;

# Completion of RET instruction

W\_icode == IRET : W\_valM;

# Default: Use predicted value of PC

1 : F\_predPC;

];

## Determine icode of fetched instruction

word f\_icode = [

imem\_error : INOP;

1: imem\_icode;

];

# Determine ifun

word f\_ifun = [

imem\_error : FNONE;

1: imem\_ifun;

];

# Is instruction valid?

bool instr\_valid = f\_icode in

{ INOP, IHALT, IRRMOVQ, IIRMOVQ, IRMMOVQ, IMRMOVQ,

IOPQ, IJXX, ICALL, IRET, IPUSHQ, IPOPQ, IIADDQ };

# Determine status code for fetched instruction

word f\_stat = [

imem\_error: SADR;

!instr\_valid : SINS;

f\_icode == IHALT : SHLT;

1 : SAOK;

];

# Does fetched instruction require a regid byte?

bool need\_regids =

f\_icode in { IRRMOVQ, IOPQ, IPUSHQ, IPOPQ,

IIRMOVQ, IRMMOVQ, IMRMOVQ, IIADDQ};

# Does fetched instruction require a constant word?

bool need\_valC =

f\_icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IJXX, ICALL, IIADDQ };

# Predict next value of PC

word f\_predPC = [

f\_icode in { IJXX, ICALL } : f\_valC;

1 : f\_valP;

];

################ Decode Stage ######################################

## What register should be used as the A source?

word d\_srcA = [

D\_icode in { IRRMOVQ, IRMMOVQ, IOPQ, IPUSHQ } : D\_rA;

D\_icode in { IPOPQ, IRET } : RRSP;

1 : RNONE; # Don't need register

];

## What register should be used as the B source?

word d\_srcB = [

D\_icode in { IOPQ, IRMMOVQ, IMRMOVQ, IIADDQ } : D\_rB;

D\_icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;

1 : RNONE; # Don't need register

];

## What register should be used as the E destination?

word d\_dstE = [

D\_icode in { IRRMOVQ, IIRMOVQ, IOPQ, IIADDQ} : D\_rB;

D\_icode in { IPUSHQ, IPOPQ, ICALL, IRET } : RRSP;

1 : RNONE; # Don't write any register

];

## What register should be used as the M destination?

word d\_dstM = [

D\_icode in { IMRMOVQ, IPOPQ } : D\_rA;

1 : RNONE; # Don't write any register

];

## What should be the A value?

## Forward into decode stage for valA

word d\_valA = [

D\_icode in { ICALL, IJXX } : D\_valP; # Use incremented PC

d\_srcA == e\_dstE : e\_valE; # Forward valE from execute

d\_srcA == M\_dstM : m\_valM; # Forward valM from memory

d\_srcA == M\_dstE : M\_valE; # Forward valE from memory

d\_srcA == W\_dstM : W\_valM; # Forward valM from write back

d\_srcA == W\_dstE : W\_valE; # Forward valE from write back

1 : d\_rvalA; # Use value read from register file

];

word d\_valB = [

d\_srcB == e\_dstE : e\_valE; # Forward valE from execute

d\_srcB == M\_dstM : m\_valM; # Forward valM from memory

d\_srcB == M\_dstE : M\_valE; # Forward valE from memory

d\_srcB == W\_dstM : W\_valM; # Forward valM from write back

d\_srcB == W\_dstE : W\_valE; # Forward valE from write back

1 : d\_rvalB; # Use value read from register file

];

################ Execute Stage #####################################

## Select input A to ALU

word aluA = [

E\_icode in { IRRMOVQ, IOPQ } : E\_valA;

E\_icode in { IIRMOVQ, IRMMOVQ, IMRMOVQ, IIADDQ} : E\_valC;

E\_icode in { ICALL, IPUSHQ } : -8;

E\_icode in { IRET, IPOPQ } : 8;

# Other instructions don't need ALU

];

## Select input B to ALU

word aluB = [

E\_icode in { IRMMOVQ, IMRMOVQ, IOPQ, ICALL,

IPUSHQ, IRET, IPOPQ, IIADDQ} : E\_valB;

E\_icode in { IRRMOVQ, IIRMOVQ } : 0;

# Other instructions don't need ALU

];

## Set the ALU function

word alufun = [

E\_icode == IOPQ : E\_ifun;

1 : ALUADD;

];

## Should the condition codes be updated?

bool set\_cc = (E\_icode == IOPQ || E\_icode == IIADDQ) &&

# State changes only during normal operation

!m\_stat in { SADR, SINS, SHLT } && !W\_stat in { SADR, SINS, SHLT };

## Generate valA in execute stage

word e\_valA = [

E\_icode in {IRMMOVQ,IPUSHQ} && E\_srcA == M\_dstM :m\_valM;

1:E\_valA; # Pass valA through stage

];

## Set dstE to RNONE in event of not-taken conditional move

word e\_dstE = [

E\_icode == IRRMOVQ && !e\_Cnd : RNONE;

1 : E\_dstE;

];

################ Memory Stage ######################################

## Select memory address

word mem\_addr = [

M\_icode in { IRMMOVQ, IPUSHQ, ICALL, IMRMOVQ } : M\_valE;

M\_icode in { IPOPQ, IRET } : M\_valA;

# Other instructions don't need address

];

## Set read control signal

bool mem\_read = M\_icode in { IMRMOVQ, IPOPQ, IRET };

## Set write control signal

bool mem\_write = M\_icode in { IRMMOVQ, IPUSHQ, ICALL };

#/\* $begin pipe-m\_stat-hcl \*/

## Update the status

word m\_stat = [

dmem\_error : SADR;

1 : M\_stat;

];

#/\* $end pipe-m\_stat-hcl \*/

## Set E port register ID

word w\_dstE = W\_dstE;

## Set E port value

word w\_valE = W\_valE;

## Set M port register ID

word w\_dstM = W\_dstM;

## Set M port value

word w\_valM = W\_valM;

## Update processor status

word Stat = [

W\_stat == SBUB : SAOK;

1 : W\_stat;

];

################ Pipeline Register Control #########################

# Should I stall or inject a bubble into Pipeline Register F?

# At most one of these can be true.

bool F\_bubble = 0;

bool F\_stall =

# Conditions for a load/use hazard

E\_icode in { IMRMOVQ, IPOPQ } &&

(E\_dstM in {d\_srcB} || E\_dstM == d\_srcA &&!(D\_icode in {IRMMOVQ,IPUSHQ})) ||

# Stalling at fetch while ret passes through pipeline

IRET in { D\_icode, E\_icode, M\_icode };

# Should I stall or inject a bubble into Pipeline Register D?

# At most one of these can be true.

bool D\_stall =

# Conditions for a load/use hazard

E\_icode in { IMRMOVQ, IPOPQ } &&

(E\_dstM in {d\_srcB} || E\_dstM == d\_srcA &&!(D\_icode in {IRMMOVQ,IPUSHQ}));

bool D\_bubble =

# Mispredicted branch

(E\_icode == IJXX && !e\_Cnd) ||

# Stalling at fetch while ret passes through pipeline

# but not condition for a load/use hazard

!(E\_icode in { IMRMOVQ, IPOPQ } && (E\_dstM in {d\_srcB} || E\_dstM == d\_srcA &&!(D\_icode in {IRMMOVQ,IPUSHQ}))) &&

IRET in { D\_icode, E\_icode, M\_icode };

# Should I stall or inject a bubble into Pipeline Register E?

# At most one of these can be true.

bool E\_stall = 0;

bool E\_bubble =

# Mispredicted branch

(E\_icode == IJXX && !e\_Cnd) ||

# Conditions for a load/use hazard

E\_icode in { IMRMOVQ, IPOPQ } &&

(E\_dstM in {d\_srcB} || E\_dstM == d\_srcA &&!(D\_icode in {IRMMOVQ,IPUSHQ})) ;

# Should I stall or inject a bubble into Pipeline Register M?

# At most one of these can be true.

bool M\_stall = 0;

# Start injecting bubbles as soon as exception passes through memory stage

bool M\_bubble = m\_stat in { SADR, SINS, SHLT } || W\_stat in { SADR, SINS, SHLT };

# Should I stall or inject a bubble into Pipeline Register W?

bool W\_stall = W\_stat in { SADR, SINS, SHLT };

bool W\_bubble = 0;

#/\* $end pipe-all-hcl \*/

这里或许没人看，我就说点悄悄话，我们通过上面的过程可以看到，修改指令和循环展开获取的性能提升非常明显，这些优化是显而易见的，也是比较容易的，性价比很高。而后续的优化包括分支优化、加载转发，提升不算大，但是难度却不小，花了大量的时间，CPE的提升只有0.1，性价比不高。

不过，容易的谁都能做到，而能把不容易的做到，才能体现出自己产品的优势。很多好的系统和处理器或许就是在这些细节上有精益求精的要求，日积月累下让这些小的优势积累成了大的壁垒，成为了自己产品的核心竞争力。个人认为不应该忽视这些细枝末节的优化，累积的多了就会成为突变。

俗话说：不积跬步无以至千里，或许就是这个道理。

**附1：10421**

#/\* $begin ncopy-ys \*/

##################################################################

# 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

#xorq %rax,%rax # count = 0;

iaddq $-10,%rdx

jl L4

Loop10:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

mrmovq 24(%rdi), %r13 # read val from src...

mrmovq 32(%rdi), %r14 # read val from src...

mrmovq 40(%rdi), %r8 # read val from src...

mrmovq 48(%rdi), %r9 # read val from src...

mrmovq 56(%rdi), %rbp # read val from src...

mrmovq 64(%rdi), %rcx # read val from src...

mrmovq 72(%rdi), %rbx # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

rmmovq %r13, 24(%rsi) # ...and store it to dst

rmmovq %r14, 32(%rsi) # ...and store it to dst

rmmovq %r8, 40(%rsi) # ...and store it to dst

rmmovq %r9, 48(%rsi) # ...and store it to dst

rmmovq %rbp, 56(%rsi) # ...and store it to dst

rmmovq %rcx, 64(%rsi) # ...and store it to dst

rmmovq %rbx, 72(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n1

iaddq $1,%rax # count++

n1:

andq %r11, %r11 # val <= 0?

jle n2

iaddq $1,%rax # count++

n2:

andq %r12, %r12 # val <= 0?

jle n3

iaddq $1,%rax # count++

n3:

andq %r13, %r13 # val <= 0?

jle n4

iaddq $1,%rax # count++

n4:

andq %r14, %r14 # val <= 0?

jle n5

iaddq $1,%rax # count++

n5:

andq %r8, %r8 # val <= 0?

jle n6

iaddq $1,%rax # count++

n6:

andq %r9, %r9 # val <= 0?

jle n7

iaddq $1,%rax # count++

n7:

andq %rbp, %rbp # val <= 0?

jle n8

iaddq $1,%rax # count++

n8:

andq %rcx, %rcx # val <= 0?

jle n9

iaddq $1,%rax # count++

n9:

andq %rbx, %rbx # val <= 0?

jle n10

iaddq $1,%rax # count++

n10:

iaddq $80,%rdi

iaddq $80,%rsi

iaddq $-10,%rdx

jge Loop10 #

L4:

iaddq $6,%rdx

jl L2 # if so, goto Done:

Loop4:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

mrmovq 24(%rdi), %r13 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

rmmovq %r13, 24(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n41

iaddq $1,%rax # count++

n41:

andq %r11, %r11 # val <= 0?

jle n42

iaddq $1,%rax # count++

n42:

andq %r12, %r12 # val <= 1?

jle n43

iaddq $1,%rax # count++

n43:

andq %r13, %r13 # val <= 0?

jle n44

iaddq $1,%rax # count++

n44:

iaddq $32,%rdi

iaddq $32,%rsi

iaddq $-4,%rdx

jl L2

jge Loop4

L2:

iaddq $2,%rdx

jl L1

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

iaddq $16,%rdi

iaddq $16, %rsi # dst++

andq %r10, %r10 # val <= 0?

jle N21 # if so, goto Npos:

iaddq $1,%rax # count++

N21:

iaddq $-2,%rdx

andq %r11,%r11

jle L1

iaddq $1,%rax

L1:

iaddq $1,%rdx

je N11 # if so, goto Done:

ret

N11:

mrmovq (%rdi), %r10 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle Done # if so, goto Npos:

iaddq $1,%rax # count++

##################################################################

# 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 \*/

**附2：84321**

#/\* $begin ncopy-ys \*/

##################################################################

# 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

#xorq %rax,%rax # count = 0;

iaddq $-8,%rdx

jl L4

Loop8:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

mrmovq 24(%rdi), %r13 # read val from src...

mrmovq 32(%rdi), %r14 # read val from src...

mrmovq 40(%rdi), %r8 # read val from src...

mrmovq 48(%rdi), %r9 # read val from src...

mrmovq 56(%rdi), %rbp # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

rmmovq %r13, 24(%rsi) # ...and store it to dst

rmmovq %r14, 32(%rsi) # ...and store it to dst

rmmovq %r8, 40(%rsi) # ...and store it to dst

rmmovq %r9, 48(%rsi) # ...and store it to dst

rmmovq %rbp, 56(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n1

iaddq $1,%rax # count++

n1:

andq %r11, %r11 # val <= 0?

jle n2

iaddq $1,%rax # count++

n2:

andq %r12, %r12 # val <= 0?

jle n3

iaddq $1,%rax # count++

n3:

andq %r13, %r13 # val <= 0?

jle n4

iaddq $1,%rax # count++

n4:

andq %r14, %r14 # val <= 0?

jle n5

iaddq $1,%rax # count++

n5:

andq %r8, %r8 # val <= 0?

jle n6

iaddq $1,%rax # count++

n6:

andq %r9, %r9 # val <= 0?

jle n7

iaddq $1,%rax # count++

n7:

andq %rbp, %rbp # val <= 0?

jle n8

iaddq $1,%rax # count++

n8:

iaddq $64,%rdi

iaddq $64,%rsi

iaddq $-8,%rdx

jge Loop8 #

L4:

iaddq $4,%rdx

jl L3 # if so, goto Done:

Loop4:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

mrmovq 24(%rdi), %r13 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

rmmovq %r13, 24(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n41

iaddq $1,%rax # count++

n41:

andq %r11, %r11 # val <= 0?

jle n42

iaddq $1,%rax # count++

n42:

andq %r12, %r12 # val <= 0?

jle n43

iaddq $1,%rax # count++

n43:

andq %r13, %r13 # val <= 0?

jle n44

iaddq $1,%rax # count++

n44:

iaddq $32,%rdi

iaddq $32,%rsi

iaddq $-4,%rdx

L3:

iaddq $1,%rdx

jl L2

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle N31 # if so, goto Npos:

iaddq $1,%rax # count++

N31:

andq %r11,%r11

jle N32

iaddq $1,%rax

N32:

andq %r12,%r12

jle Done

iaddq $1,%rax

ret

L2:

iaddq $1,%rdx

jl L1

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle N21 # if so, goto Npos:

iaddq $1,%rax # count++

N21:

andq %r11,%r11

jle Done

iaddq $1,%rax

ret

L1:

iaddq $1,%rdx

je N11

ret

N11:

mrmovq (%rdi), %r10 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle Done # if so, goto Npos:

iaddq $1,%rax # count++

##################################################################

# 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 \*/

**附3：84r213**

#/\* $begin ncopy-ys \*/

##################################################################

# 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

#xorq %rax,%rax # count = 0;

iaddq $-8,%rdx

jl L4

Loop8:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

mrmovq 24(%rdi), %r13 # read val from src...

mrmovq 32(%rdi), %r14 # read val from src...

mrmovq 40(%rdi), %r8 # read val from src...

mrmovq 48(%rdi), %r9 # read val from src...

mrmovq 56(%rdi), %rbp # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

rmmovq %r13, 24(%rsi) # ...and store it to dst

rmmovq %r14, 32(%rsi) # ...and store it to dst

rmmovq %r8, 40(%rsi) # ...and store it to dst

rmmovq %r9, 48(%rsi) # ...and store it to dst

rmmovq %rbp, 56(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n1

iaddq $1,%rax # count++

n1:

andq %r11, %r11 # val <= 0?

jle n2

iaddq $1,%rax # count++

n2:

andq %r12, %r12 # val <= 0?

jle n3

iaddq $1,%rax # count++

n3:

andq %r13, %r13 # val <= 0?

jle n4

iaddq $1,%rax # count++

n4:

andq %r14, %r14 # val <= 0?

jle n5

iaddq $1,%rax # count++

n5:

andq %r8, %r8 # val <= 0?

jle n6

iaddq $1,%rax # count++

n6:

andq %r9, %r9 # val <= 0?

jle n7

iaddq $1,%rax # count++

n7:

andq %rbp, %rbp # val <= 0?

jle n8

iaddq $1,%rax # count++

n8:

iaddq $64,%rdi

iaddq $64,%rsi

iaddq $-8,%rdx

jge Loop8 #

L4:

iaddq $4,%rdx

jl L2 # if so, goto Done:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

mrmovq 24(%rdi), %r13 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

rmmovq %r13, 24(%rsi) # ...and store it to dst

iaddq $32,%rdi

iaddq $32,%rsi

iaddq $-4,%rdx

andq %r10, %r10 # val <= 0?

jle n41

iaddq $1,%rax # count++

n41:

andq %r11, %r11 # val <= 0?

jle n42

iaddq $1,%rax # count++

n42:

andq %r12, %r12 # val <= 0?

jle n43

iaddq $1,%rax # count++

n43:

andq %r13, %r13 # val <= 0?

jle L2

iaddq $1,%rax # count++

L2:

iaddq $2,%rdx

jl L1

jg L3

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle N21 # if so, goto Npos:

iaddq $1,%rax # count++

N21:

andq %r11,%r11

jle Done

iaddq $1,%rax

ret

L3:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

mrmovq 16(%rdi), %r12 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

rmmovq %r11, 8(%rsi) # ...and store it to dst

rmmovq %r12, 16(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle N31 # if so, goto Npos:

iaddq $1,%rax # count++

N31:

andq %r11,%r11

jle N32

iaddq $1,%rax

N32:

andq %r12,%r12

jle Done

iaddq $1,%rax

ret

L1:

iaddq $1,%rdx

je L11

ret

L11:

mrmovq (%rdi), %r10 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle Done # if so, goto Npos:

iaddq $1,%rax # count++

##################################################################

# 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 \*/

**附4：满分-只改iaddq指令**

#/\* $begin ncopy-ys \*/

##################################################################

# 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

#xorq %rax,%rax # count = 0;

iaddq $-9,%rdx

jl BR

Loop9:

mrmovq (%rdi), %r10 # read val from src...

mrmovq 8(%rdi), %r11 # read val from src...

rmmovq %r10, (%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n1

iaddq $1,%rax # count++

n1:

mrmovq 16(%rdi), %r10 # read val from src...

rmmovq %r11, 8(%rsi) # ...and store it to dst

andq %r11, %r11 # val <= 0?

jle n2

iaddq $1,%rax # count++

n2:

mrmovq 24(%rdi), %r11 # read val from src...

rmmovq %r10, 16(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n3

iaddq $1,%rax # count++

n3:

mrmovq 32(%rdi), %r10 # read val from src...

rmmovq %r11, 24(%rsi) # ...and store it to dst

andq %r11, %r11 # val <= 0?

jle n4

iaddq $1,%rax # count++

n4:

mrmovq 40(%rdi), %r11 # read val from src...

rmmovq %r10, 32(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n5

iaddq $1,%rax # count++

n5:

mrmovq 48(%rdi), %r10 # read val from src...

rmmovq %r11, 40(%rsi) # ...and store it to dst

andq %r11, %r11 # val <= 0?

jle n6

iaddq $1,%rax # count++

n6:

mrmovq 56(%rdi), %r11 # read val from src...

rmmovq %r10, 48(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n7

iaddq $1,%rax # count++

n7:

mrmovq 64(%rdi), %r10 # read val from src...

rmmovq %r11, 56(%rsi) # ...and store it to dst

andq %r11, %r11 # val <= 0?

jle n8

iaddq $1,%rax # count++

n8:

rmmovq %r10, 64(%rsi) # ...and store it to dst

andq %r10, %r10 # val <= 0?

jle n9

iaddq $1,%rax # count++

n9:

iaddq $72,%rdi

iaddq $72,%rsi

iaddq $-9,%rdx

jge Loop9 #

BR:

iaddq $6,%rdx

jl Left

jg Right

je L3

L45:

iaddq $1,%rdx

jl L4

jmp L5

Left:

iaddq $2,%rdx

je L1

jg L2

ret

Right:

iaddq $-3,%rdx

jl L45

je L6

iaddq $-2,%rdx

jl L7

mrmovq 56(%rdi), %r9 # read val from src...

rmmovq %r9,56(%rsi)

andq %r9, %r9 # val <= 0?

L7:

mrmovq 48(%rdi), %r8 # read val from src...

jle L71

iaddq $1,%rax # count++

L71:

rmmovq %r8,48(%rsi)

andq %r8, %r8 # val <= 0?

L6:

mrmovq 40(%rdi), %r9 # read val from src...

jle L61

iaddq $1,%rax # count++

L61:

rmmovq %r9,40(%rsi)

andq %r9, %r9 # val <= 0?

L5:

mrmovq 32(%rdi), %r8 # read val from src...

jle L51

iaddq $1,%rax # count++

L51:

rmmovq %r8,32(%rsi)

andq %r8, %r8 # val <= 0?

L4:

mrmovq 24(%rdi), %r9 # read val from src...

jle L41

iaddq $1,%rax # count++

L41:

rmmovq %r9,24(%rsi)

andq %r9, %r9 # val <= 0?

L3:

mrmovq 16(%rdi), %r8 # read val from src...

jle L31

iaddq $1,%rax # count++

L31:

rmmovq %r8,16(%rsi)

L2:

mrmovq 8(%rdi), %r9 # read val from src...

andq %r8, %r8 # val <= 0?

jle L21

iaddq $1,%rax # count++

L21:

rmmovq %r9,8(%rsi)

andq %r9, %r9 # val <= 0?

L1:

mrmovq (%rdi), %r8 # read val from src...

jle L11

iaddq $1,%rax # count++

L11:

rmmovq %r8,(%rsi)

andq %r8, %r8 # val <= 0?

jle Done

iaddq $1,%rax # count++

##################################################################

# 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 \*/