Project 1: Optimizing the Performance of a Pipelined Processor

516030910125 Yining Liu jadyn24@outlook.com
516030910421 Chenge Sun sunchenge1997@sjtu.edu.cn

1. Introduction

This lab is organized in 3 parts. In part A, we implement **sum_list**, **rsum_list** and **copy** functions which are functionally equivalent to corresponding functions in C, aiming to get familiar with Y86 tools. In part B, we implement new instruction **iaddl** in SEQ and learn about syntax in HCL. On the base of Part A and B, we modify **ncopy.ys** and **pipe-full.hcl**, aiming to optimize the Y86 benchmark program and the processor design.

Chenge Sun mainly worked in Y86 programs and finished Part A, while Yining Liu mainly works in instruction extensions and finished Part B. By exchanging information and working together, we finished Part C and this report.

2. Experiments

2.1 Part A

2.1.1 Analysis

Part A can be divided into three parts.

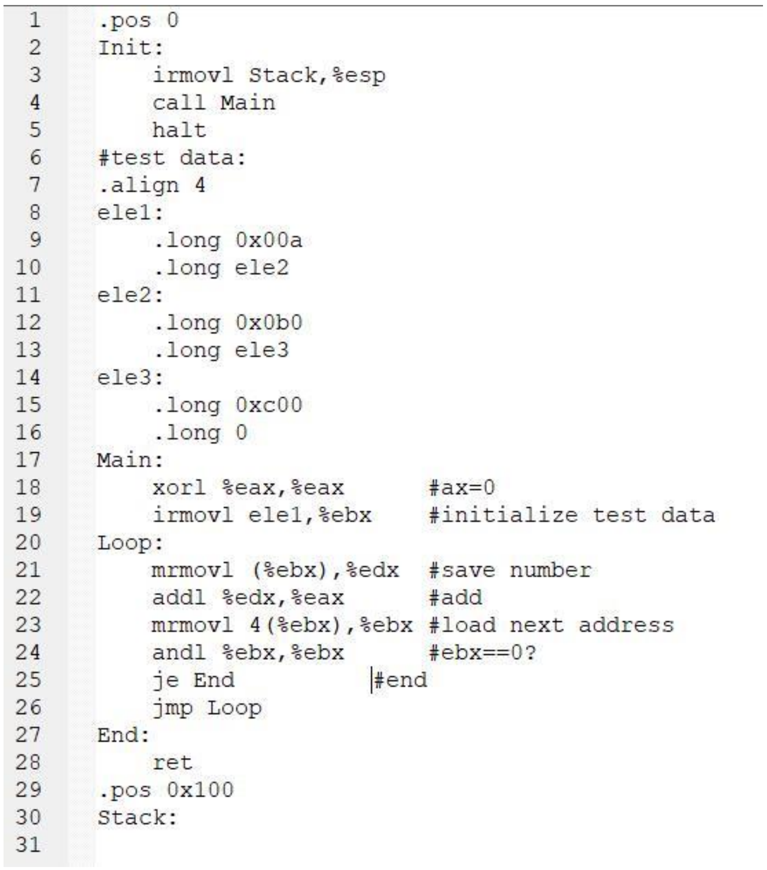
1. Simulate the **sum_list** function in a Y86 program **sum.ys**. Firstly, use **xor** operation to set the 'sum' in %eax as zero, then execute a loop to load a number in the list, add the number to sum, load next address and judge whether it goes to the end of the list.

2. Simulate the **rsum_list** function in a Y86 program **rsum.ys**. It is similar to the sum function. The core technique is using call operation to calculate the number in next address instead of executing a loop.

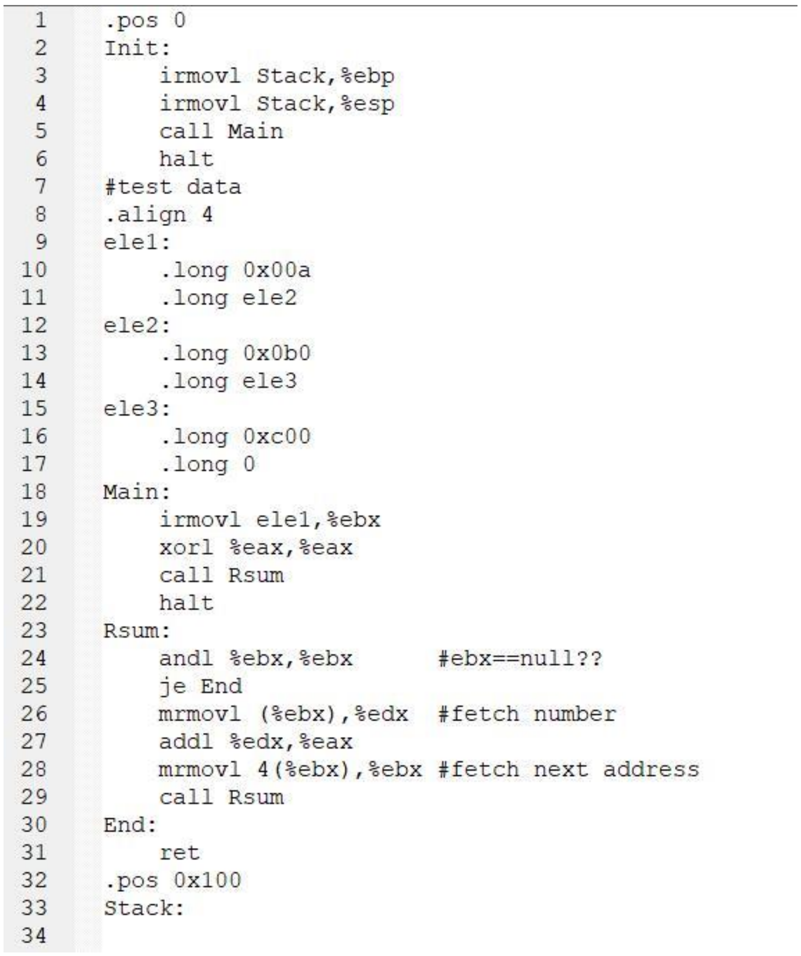
3. Simulate the **copy_block** function in a Y86 program **copy.ys**. Firstly, move the address of source block and destination block into register and define the length of data to be transfer. Then execute a loop to read and write data. We can do a **xor** operation to sum the 3 numbers since their significant bits are different.

2.1.2 Code

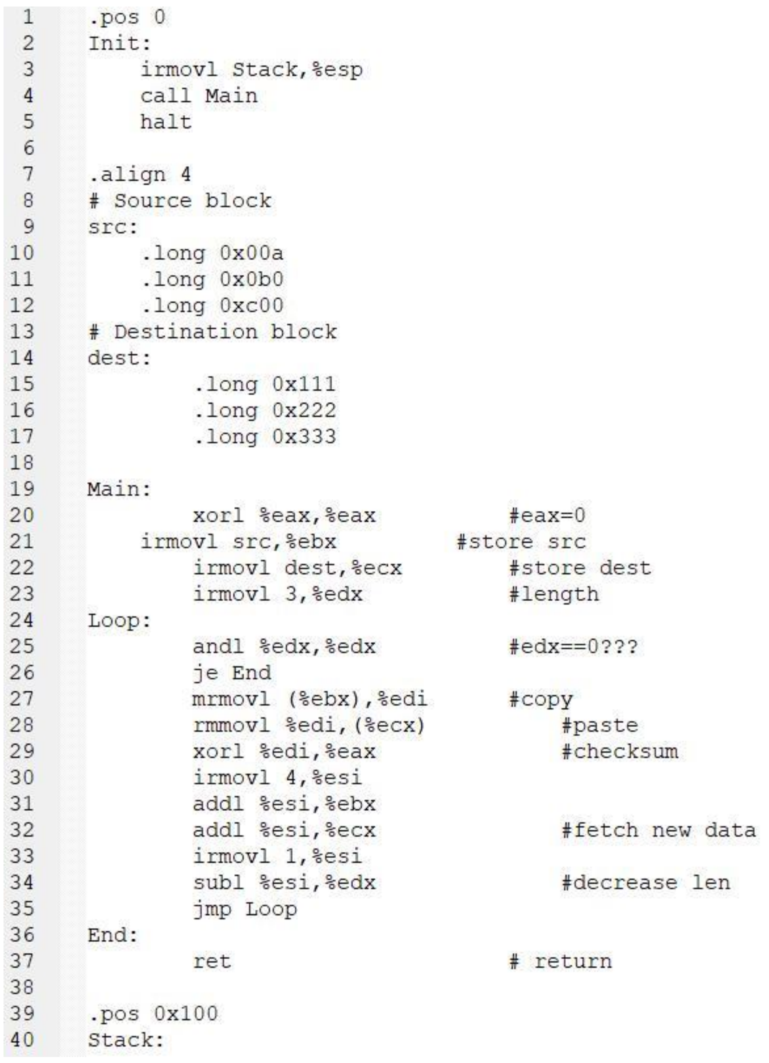
Code of **sum.ys**, **rsum.ys** and **copy.ys**



```
.pos 0
Init:
    irmovl Stack,%esp
    call Main
    halt
#test data:
.align 4
ele1:
    .long 0x00a
    .long ele2
ele2:
    .long 0x0b0
    .long ele3
ele3:
    .long 0xc00
    .long 0
Main:
    xorl %eax,%eax      #ax=0
    irmovl ele1,%ebx    #initialize test data
Loop:
    mrmovl (%ebx),%edx  #save number
    addl %edx,%eax      #add
    mrmovl 4(%ebx),%ebx #load next address
    andl %ebx,%ebx      #ebx==0?
    je End             #end
    jmp Loop
End:
    ret
.pos 0x100
Stack:
```

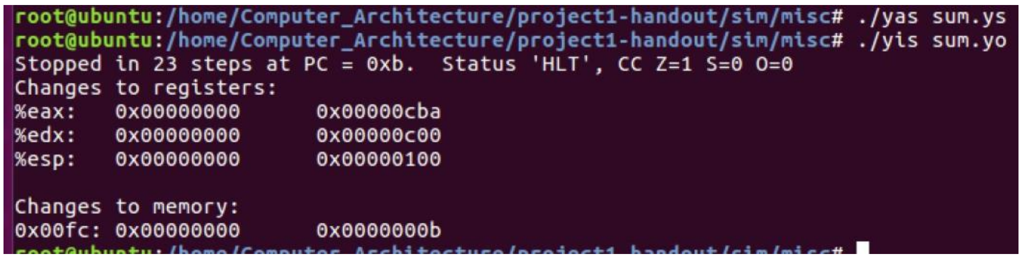


```
.pos 0
Init:
    irmovl Stack,%ebp
    irmovl Stack,%esp
    call Main
    halt
#test data
.align 4
ele1:
    .long 0x00a
    .long ele2
ele2:
    .long 0x0b0
    .long ele3
ele3:
    .long 0xc00
    .long 0
Main:
    irmovl ele1,%ebx
    xorl %eax,%eax
    call Rsum
    halt
Rsum:
    andl %ebx,%ebx      #ebx==null??
    je End
    mrmovl (%ebx),%edx  #fetch number
    addl %edx,%eax
    mrmovl 4(%ebx),%ebx #fetch next address
    call Rsum
End:
    ret
.pos 0x100
Stack:
```



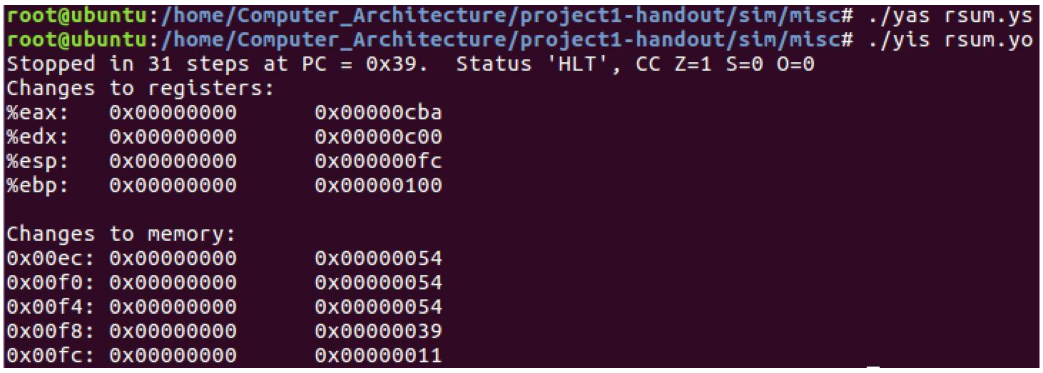
```
.pos 0
Init:
    irmovl Stack,%esp
    call Main
    halt

.align 4
# Source block
src:
    .long 0x00a
    .long 0x0b0
    .long 0xc00
# Destination block
dest:
        .long 0x111
        .long 0x222
        .long 0x333

Main:
        xorl %eax,%eax          #eax=0
    irmovl src,%ebx         #store src
        irmovl dest,%ecx        #store dest
        irmovl 3,%edx           #length
Loop:
        andl %edx,%edx          #edx==0???
        je End
        mrmovl (%ebx),%edi      #copy
        rmmovl %edi,(%ecx)          #paste
        xorl %edi,%eax              #checksum
        irmovl 4,%esi
        addl %esi,%ebx
        addl %esi,%ecx              #fetch new data
        irmovl 1,%esi
        subl %esi,%edx              #decrease len
        jmp Loop
End:
        ret                     # return

.pos 0x100
Stack:
```

2.1.3 Evaluation



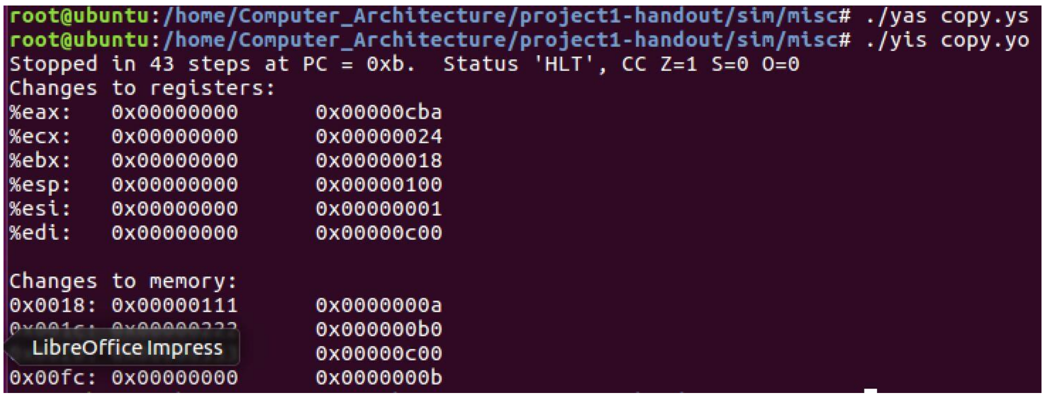
```
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc# ./yas sum.ys
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc# ./yis sum.yo
Stopped in 23 steps at PC = 0xb.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%eax:   0x00000000      0x00000cba
%edx:   0x00000000      0x00000c00
%esp:   0x00000000      0x00000100

Changes to memory:
0x00fc: 0x00000000      0x0000000b
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc#
```

sum.ys: sum stored in %eax is 0x00000cba which is the sum of ele1, ele2 and ele3.



```
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc# ./yas rsum.ys
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc# ./yis rsum.yo
Stopped in 31 steps at PC = 0x39.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%eax:   0x00000000      0x00000cba
%edx:   0x00000000      0x00000c00
%esp:   0x00000000      0x000000fc
%ebp:   0x00000000      0x00000100

Changes to memory:
0x00ec: 0x00000000      0x00000054
0x00f0: 0x00000000      0x00000054
0x00f4: 0x00000000      0x00000054
0x00f8: 0x00000000      0x00000039
0x00fc: 0x00000000      0x00000011
```

rsum.ys: sum stored in %eax is 0x00000cba which is the sum of ele1, ele2 and ele3.



```
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc# ./yas copy.ys
root@ubuntu:/home/Computer_Architecture/project1-handout/sim/misc# ./yis copy.yo
Stopped in 43 steps at PC = 0xb.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:
%eax:   0x00000000      0x00000cba
%ecx:   0x00000000      0x00000024
%ebx:   0x00000000      0x00000018
%esp:   0x00000000      0x00000100
%esi:   0x00000000      0x00000001
%edi:   0x00000000      0x00000c00

Changes to memory:
0x0018: 0x00000111      0x0000000a
0x001c: 0x00000222      0x000000b0
[illegible]             0x00000c00
0x00fc: 0x00000000      0x0000000b
```

copy.ys: According to the Changes to memory, we move 0x0000000a to the address with original content 0x0000111 and so on as we expect. And the checksum stored in %eax is 0x00000cba which is the exactly the sum of those three numbers in src block.

2.2 Part B

2.2.1 Analysis

Our task in Part B is to extend SEQ processor with a new instruction **iaddl**, which adds a constant value to a register and stores the sum in the original register.

The difficult point, or to say, the main task of this part, is to analyze computations required for each stage. A description is attached to the header of our code.

The modification of **iaddl** are as follows: making iaddl valid, stating that iaddl needs both constant and register operand, stating which register to fetch and the two inputs of ALU, and setting condition code for it.

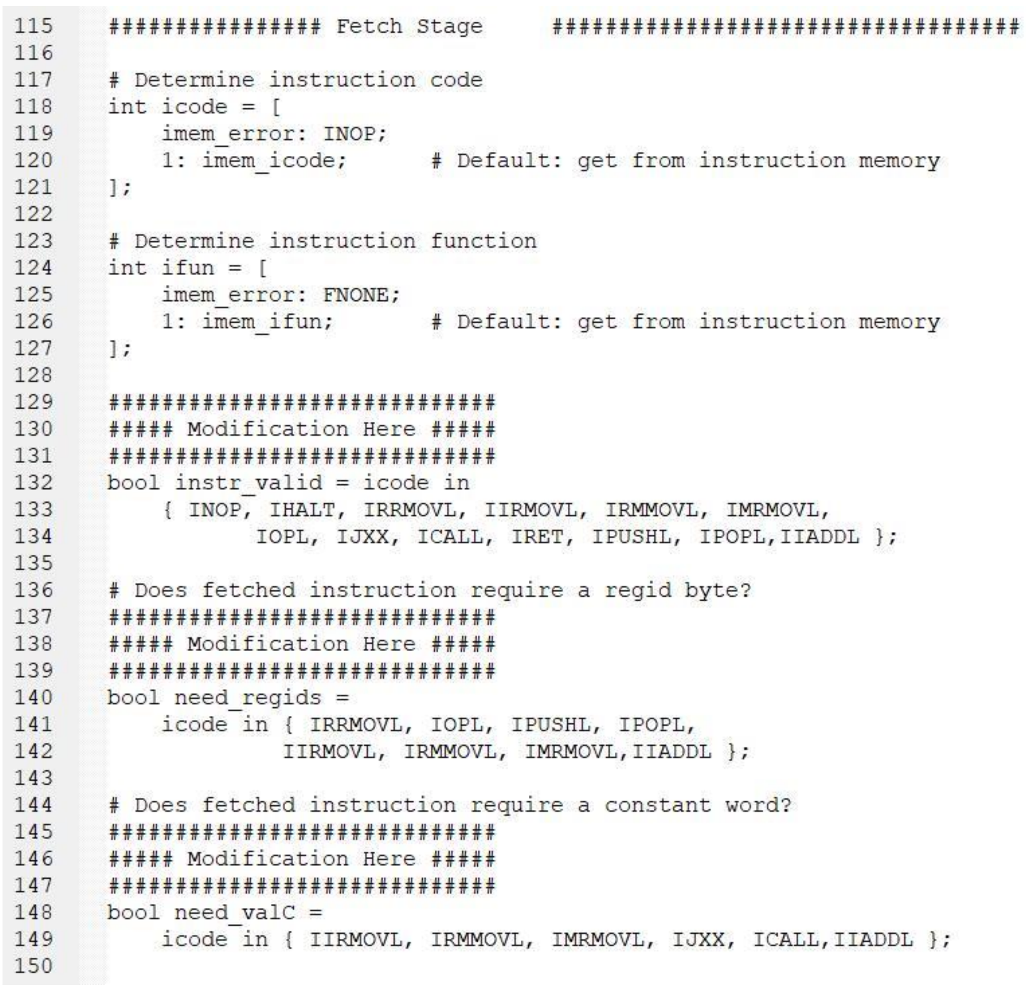
For the implementation, we add **IIADDL** to corresponding stage in **.hcl** file to make **iaddl** functional. Fetch, decode and execute stage need to be modified. Syntax of **.hcl** file should be core technique of this task.

2.2.2 Code

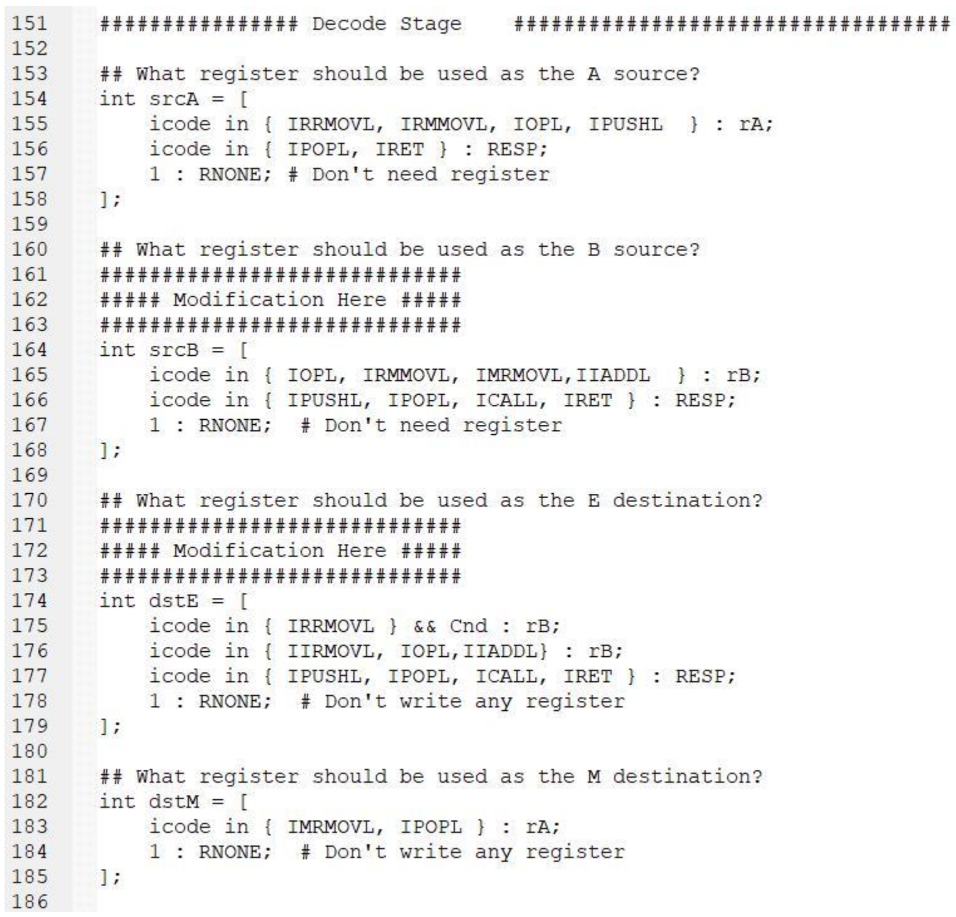
In fetch stage: add it to valid tuple, both regid byte and constant word are needed.

In decode stage: both the B source and the E destination are rB.

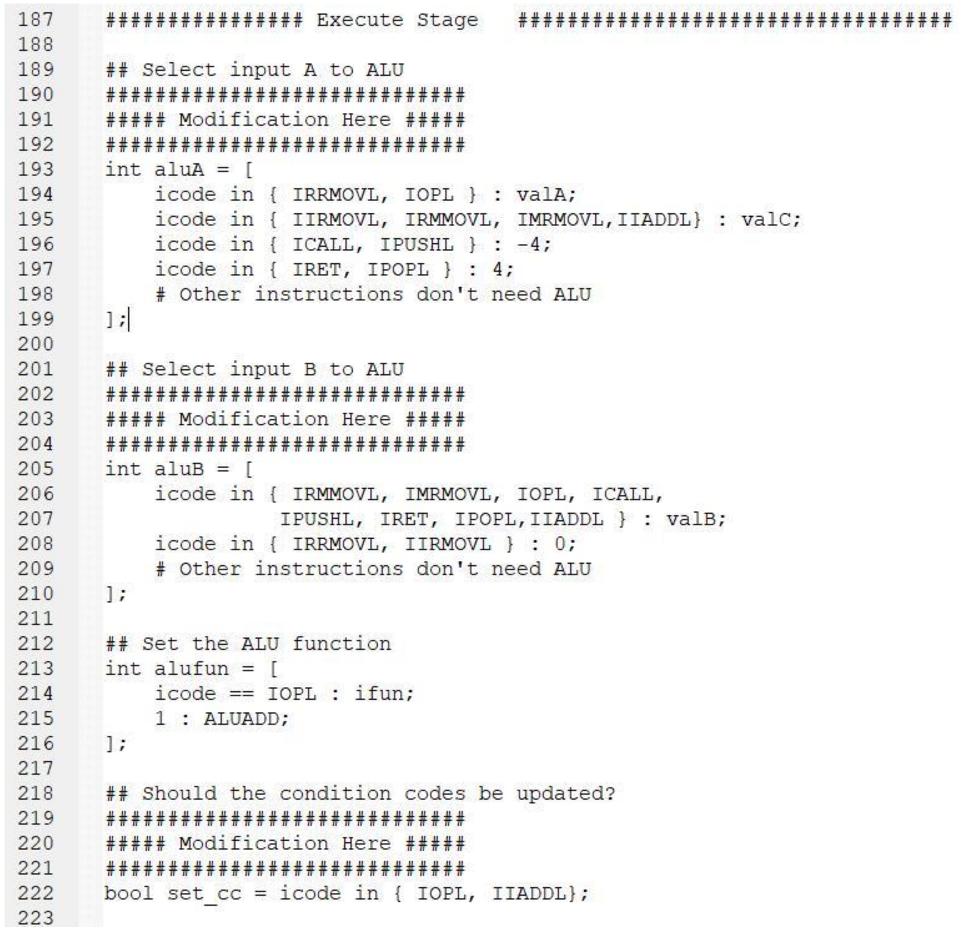
In execute stage: constant number saved in valC and register value in valB, add it to condition code as well.



```
################ Fetch Stage     ##################################

# Determine instruction code
int icode = [
    imem_error: INOP;
    1: imem_icode;      # Default: get from instruction memory
];

# Determine instruction function
int ifun = [
    imem_error: FNONE;
    1: imem_ifun;       # Default: get from instruction memory
];

#############################
##### Modification Here #####
#############################
bool instr_valid = icode in
    { INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
           IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL,IIADDL };

# Does fetched instruction require a regid byte?
#############################
##### Modification Here #####
#############################
bool need_regids =
    icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,
             IIRMOVL, IRMMOVL, IMRMOVL,IIADDL };

# Does fetched instruction require a constant word?
#############################
##### Modification Here #####
#############################
bool need_valC =
    icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL,IIADDL };
```



```
################ Decode Stage    ###################################

## What register should be used as the A source?
int srcA = [
    icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL  } : rA;
    icode in { IPOPL, IRET } : RESP;
    1 : RNONE; # Don't need register
];

## What register should be used as the B source?
#############################
##### Modification Here #####
#############################
int srcB = [
    icode in { IOPL, IRMMOVL, IMRMOVL,IIADDL  } : rB;
    icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
    1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
#############################
##### Modification Here #####
#############################
int dstE = [
    icode in { IRRMOVL } && Cnd : rB;
    icode in { IIRMOVL, IOPL,IIADDL} : rB;
    icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
    1 : RNONE;  # Don't write any register
];

## What register should be used as the M destination?
int dstM = [
    icode in { IMRMOVL, IPOPL } : rA;
    1 : RNONE;  # Don't write any register
];

```

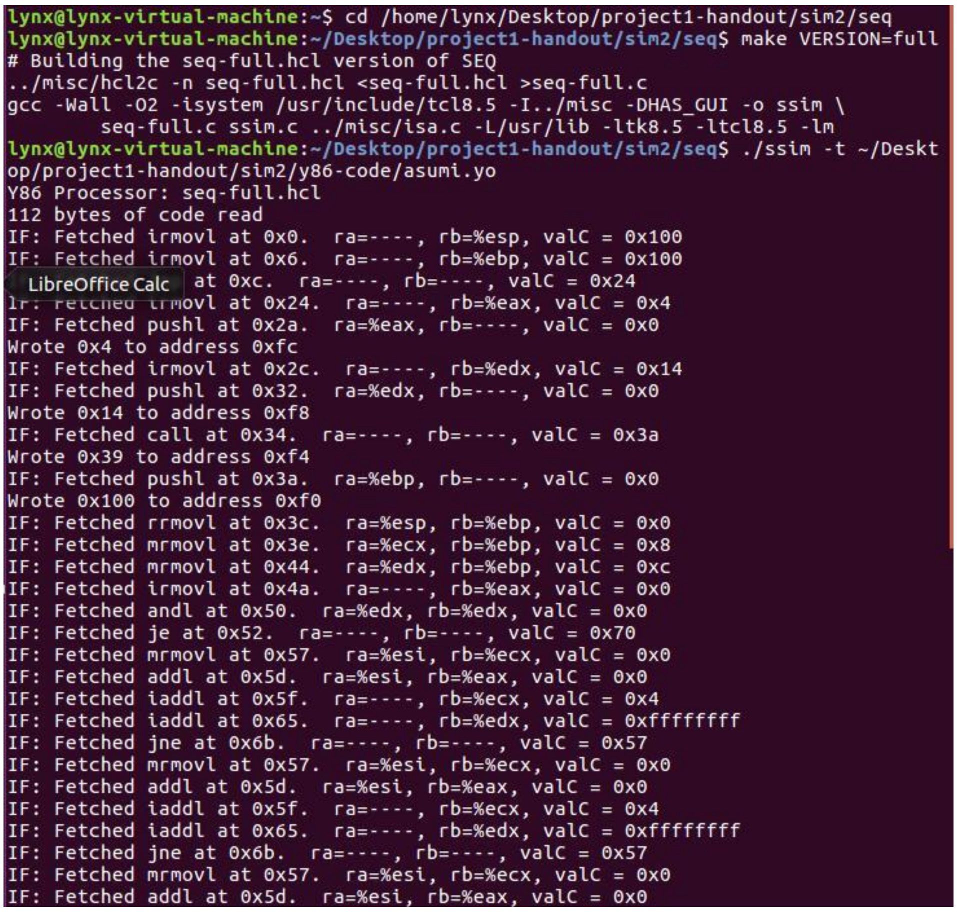


```
################ Execute Stage   ###################################

## Select input A to ALU
#############################
##### Modification Here #####
#############################
int aluA = [
    icode in { IRRMOVL, IOPL } : valA;
    icode in { IIRMOVL, IRMMOVL, IMRMOVL,IIADDL} : valC;
    icode in { ICALL, IPUSHL } : -4;
    icode in { IRET, IPOPL } : 4;
    # Other instructions don't need ALU
];

## Select input B to ALU
#############################
##### Modification Here #####
#############################
int aluB = [
    icode in { IRMMOVL, IMRMOVL, IOPL, ICALL,
              IPUSHL, IRET, IPOPL,IIADDL } : valB;
    icode in { IRRMOVL, IIRMOVL } : 0;
    # Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
    icode == IOPL : ifun;
    1 : ALUADD;
];

## Should the condition codes be updated?
#############################
##### Modification Here #####
#############################
bool set_cc = icode in { IOPL, IIADDL};

```

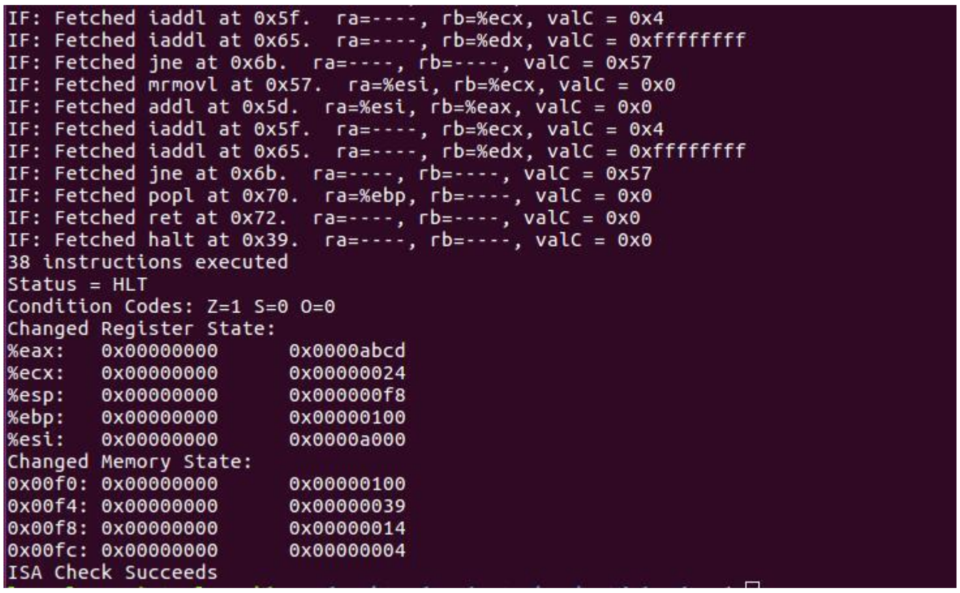
2.2.3 Evaluation

Run **asumi.yo**: check results against ISA.

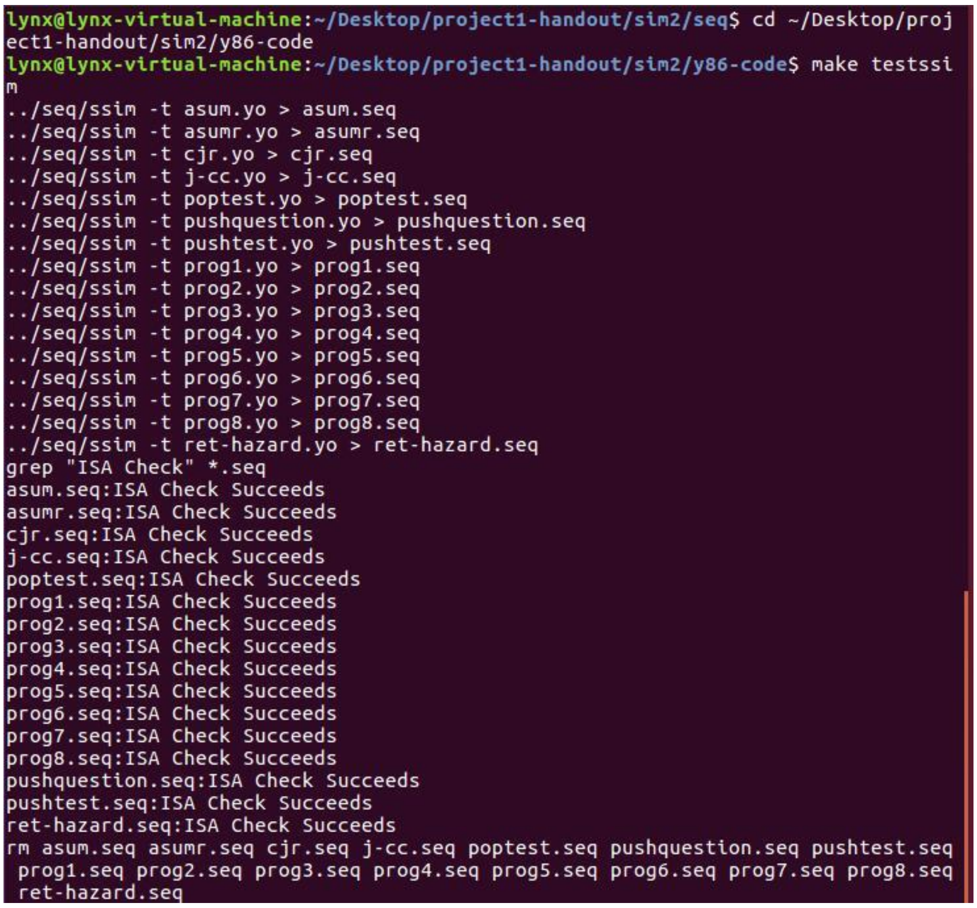
Run y86 benckmark: no error injected for original instructions.



```
lynx@lynx-virtual-machine:~$ cd /home/lynx/Desktop/project1-handout/sim2/seq
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/seq$ make VERSION=full
# Building the seq-full.hcl version of SEQ
../misc/hcl2c -n seq-full.hcl <seq-full.hcl >seq-full.c
gcc -Wall -O2 -isystem /usr/include/tcl8.5 -I../misc -DHAS_GUI -o ssim \
        seq-full.c ssim.c ../misc/isa.c -L/usr/lib -ltk8.5 -ltcl8.5 -lm
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/seq$ ./ssim -t ~/Deskt
op/project1-handout/sim2/y86-code/asumi.yo
Y86 Processor: seq-full.hcl
112 bytes of code read
IF: Fetched irmovl at 0x0.  ra=----, rb=%esp, valC = 0x100
IF: Fetched irmovl at 0x6.  ra=----, rb=%ebp, valC = 0x100
 LibreOffice Calc  at 0xc.  ra=----, rb=----, valC = 0x24
IF: Fetched irmovl at 0x24.  ra=----, rb=%eax, valC = 0x4
IF: Fetched pushl at 0x2a.  ra=%eax, rb=----, valC = 0x0
Wrote 0x4 to address 0xfc
IF: Fetched irmovl at 0x2c.  ra=----, rb=%edx, valC = 0x14
IF: Fetched pushl at 0x32.  ra=%edx, rb=----, valC = 0x0
Wrote 0x14 to address 0xf8
IF: Fetched call at 0x34.  ra=----, rb=----, valC = 0x3a
Wrote 0x39 to address 0xf4
IF: Fetched pushl at 0x3a.  ra=%ebp, rb=----, valC = 0x0
Wrote 0x100 to address 0xf0
IF: Fetched rrmovl at 0x3c.  ra=%esp, rb=%ebp, valC = 0x0
IF: Fetched mrmovl at 0x3e.  ra=%ecx, rb=%ebp, valC = 0x8
IF: Fetched mrmovl at 0x44.  ra=%edx, rb=%ebp, valC = 0xc
IF: Fetched irmovl at 0x4a.  ra=----, rb=%eax, valC = 0x0
IF: Fetched andl at 0x50.  ra=%edx, rb=%edx, valC = 0x0
IF: Fetched je at 0x52.  ra=----, rb=----, valC = 0x70
IF: Fetched mrmovl at 0x57.  ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d.  ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f.  ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65.  ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b.  ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57.  ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d.  ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f.  ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65.  ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b.  ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57.  ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d.  ra=%esi, rb=%eax, valC = 0x0
```



```
IF: Fetched iaddl at 0x5f.  ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65.  ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b.  ra=----, rb=----, valC = 0x57
IF: Fetched mrmovl at 0x57.  ra=%esi, rb=%ecx, valC = 0x0
IF: Fetched addl at 0x5d.  ra=%esi, rb=%eax, valC = 0x0
IF: Fetched iaddl at 0x5f.  ra=----, rb=%ecx, valC = 0x4
IF: Fetched iaddl at 0x65.  ra=----, rb=%edx, valC = 0xffffffff
IF: Fetched jne at 0x6b.  ra=----, rb=----, valC = 0x57
IF: Fetched popl at 0x70.  ra=%ebp, rb=----, valC = 0x0
IF: Fetched ret at 0x72.  ra=----, rb=----, valC = 0x0
IF: Fetched halt at 0x39.  ra=----, rb=----, valC = 0x0
38 instructions executed
Status = HLT
Condition Codes: Z=1 S=0 O=0
Changed Register State:
%eax:   0x00000000      0x0000abcd
%ecx:   0x00000000      0x00000024
%esp:   0x00000000      0x000000f8
%ebp:   0x00000000      0x00000100
%esi:   0x00000000      0x0000a000
Changed Memory State:
0x00f0: 0x00000000      0x00000100
0x00f4: 0x00000000      0x00000039
0x00f8: 0x00000000      0x00000014
0x00fc: 0x00000000      0x00000004
ISA Check Succeeds
```



```
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/seq$ cd ~/Desktop/proj
ect1-handout/sim2/y86-code
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/y86-code$ make testssi
m
../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
```

2.3 Part C

2.3.1 Analysis

In **pipe-full.hcl**, situations are almost the same as that in PartB, we believe no further explanation are needed. Modification list in details is attached to the code.

In **ncopy.ys** file, to decrease the CPI, we mainly focus on decreasing the number of wrongly-predicted branches and bubbles.

1. Branch: To decrease the number of wrong-predicted branches, the only way is to decrease the total number of branches, which needs us to avoid unnecessary comparisons.

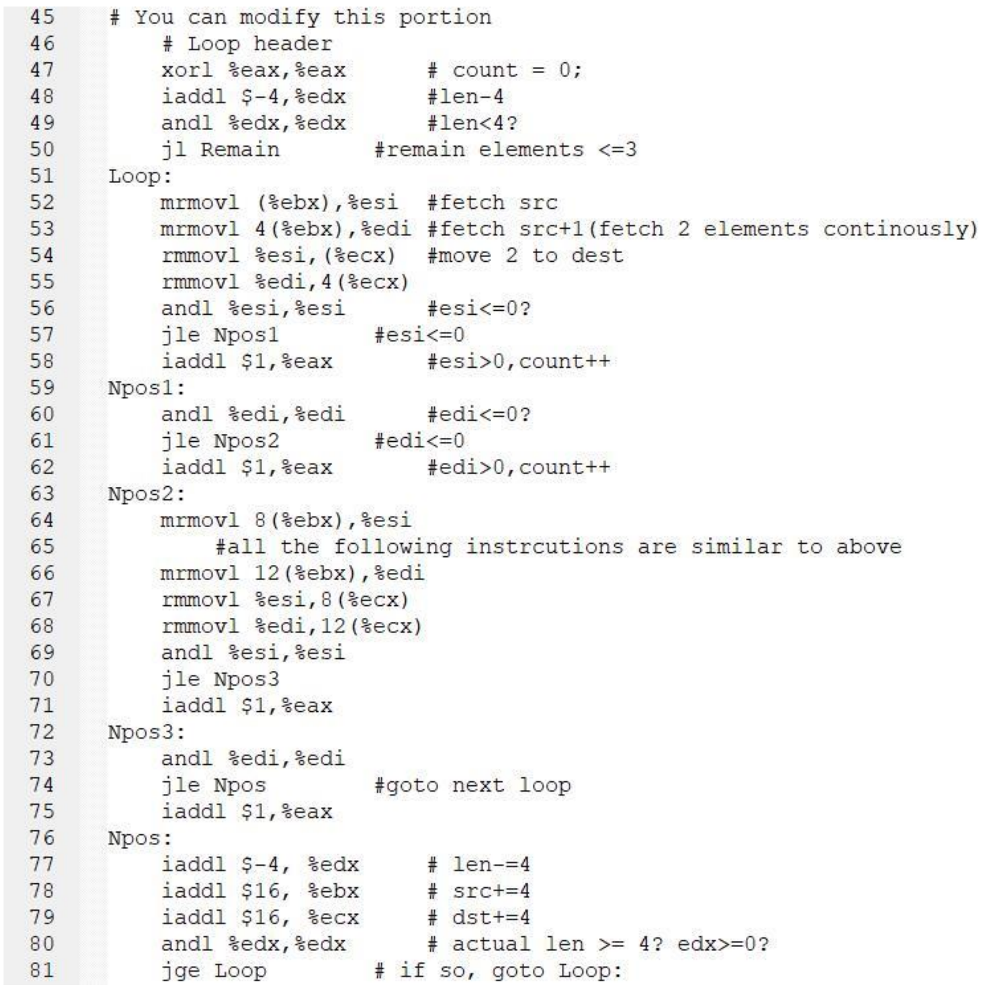
The method in this project to decrease comparisons is to decrease the judgement. There are two mainly kinds of comparison: a.judging whether the copy-content(stored in %esi,%edi) is positive; b.judging whether the current copy-length(stored in %edx) is positive. a. It is no efficient way to decrease the comparison caused by judging copy-content since the copy-contents are random. b. We copy 4 elements in a cycle of Loop, so that we can only judge the length once with 4 copy operations while the original code judge the length once with# one copy operations. And we can further reduce the comparisons by copying more elements in a cycle of Loop. In this project we choose 4 to make the code clearer.

2. Bubble: The bubbles are mainly caused by adjacent memory operations. In this project, 'mrmovl' following by 'rmmovl' operation which operates on the same register is the main source of bubbles in the original code. The bubble will be# formed because only after the former instruction passes its Memory stage will the latter operation enter the Execution stage. In this project, we put another 'rmmovl' operating on different register after a 'rmmovl'. Two seperate write-after-read operations are executed in pairs, which eliminate bubbles.

3.Other modifications: We also did some other tiny work to make it faster. Remove the judgement of whether the source block is empty at the beginning of the function, since every time we will enter the Remain part and we have to judge whether the length is zero. Move two elements in pair if the remain part of block to move to make the Remain part faster.

2.3.2 Code

ncopy.ys



```
# You can modify this portion
    # Loop header
    xorl %eax,%eax      # count = 0;
    iaddl $-4,%edx      #len-4
    andl %edx,%edx      #len<4?
    jl Remain       #remain elements <=3
Loop:
    mrmovl (%ebx),%esi  #fetch src
    mrmovl 4(%ebx),%edi #fetch src+1(fetch 2 elements continously)
    rmmovl %esi,(%ecx)  #move 2 to dest
    rmmovl %edi,4(%ecx)
    andl %esi,%esi      #esi<=0?
    jle Npos1       #esi<=0
    iaddl $1,%eax       #esi>0,count++
Npos1:
    andl %edi,%edi      #edi<=0?
    jle Npos2       #edi<=0
    iaddl $1,%eax       #edi>0,count++
Npos2:
    mrmovl 8(%ebx),%esi
        #all the following instrcutions are similar to above
    mrmovl 12(%ebx),%edi
    rmmovl %esi,8(%ecx)
    rmmovl %edi,12(%ecx)
    andl %esi,%esi
    jle Npos3
    iaddl $1,%eax
Npos3:
    andl %edi,%edi
    jle Npos        #goto next loop
    iaddl $1,%eax
Npos:
    iaddl $-4, %edx     # len-=4
    iaddl $16, %ebx     # src+=4
    iaddl $16, %ecx     # dst+=4
    andl %edx,%edx      # actual len >= 4? edx>=0?
    jge Loop        # if so, goto Loop:
```

```
Remain:
    iaddl $4,%edx       # Restore the true len
        iaddl $-1,%edx      #len=0?
        jl Done
        mrmovl (%ebx),%esi  #move 2 elements
        mrmovl 4(%ebx),%edi
        rmmovl %esi,(%ecx)
        andl %esi ,%esi     #similar to above loop
        jle rNpos
        iaddl $1,%eax
rNpos:
        iaddl $-1,%edx      #only 1 element
        jl Done
        rmmovl %edi,4(%ecx)
        andl %edi,%edi
        jle rNpos1
        iaddl $1,%eax
rNpos1:
        iaddl $-1,%edx      #2 elements
        jl Done
        mrmovl 8(%ebx),%esi
        rmmovl %esi,8(%ecx)
        andl %esi,%esi
        jle Done
        iaddl $1,%eax
```

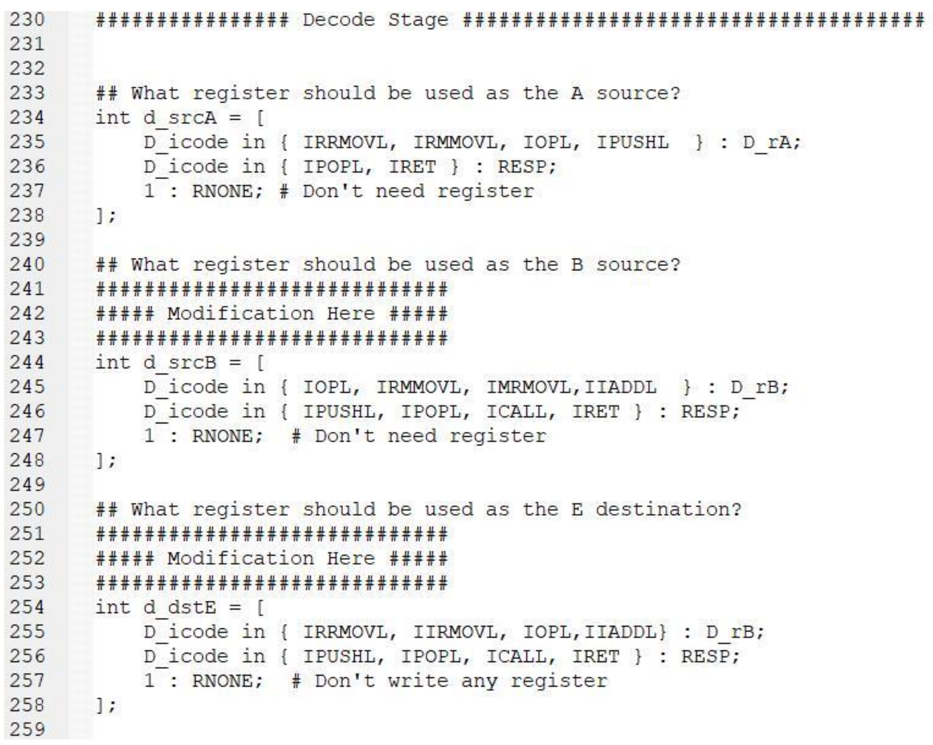
pipe-full.hcl

In fetch stage: add it to valid tuple, both regid byte and constant word are needed.

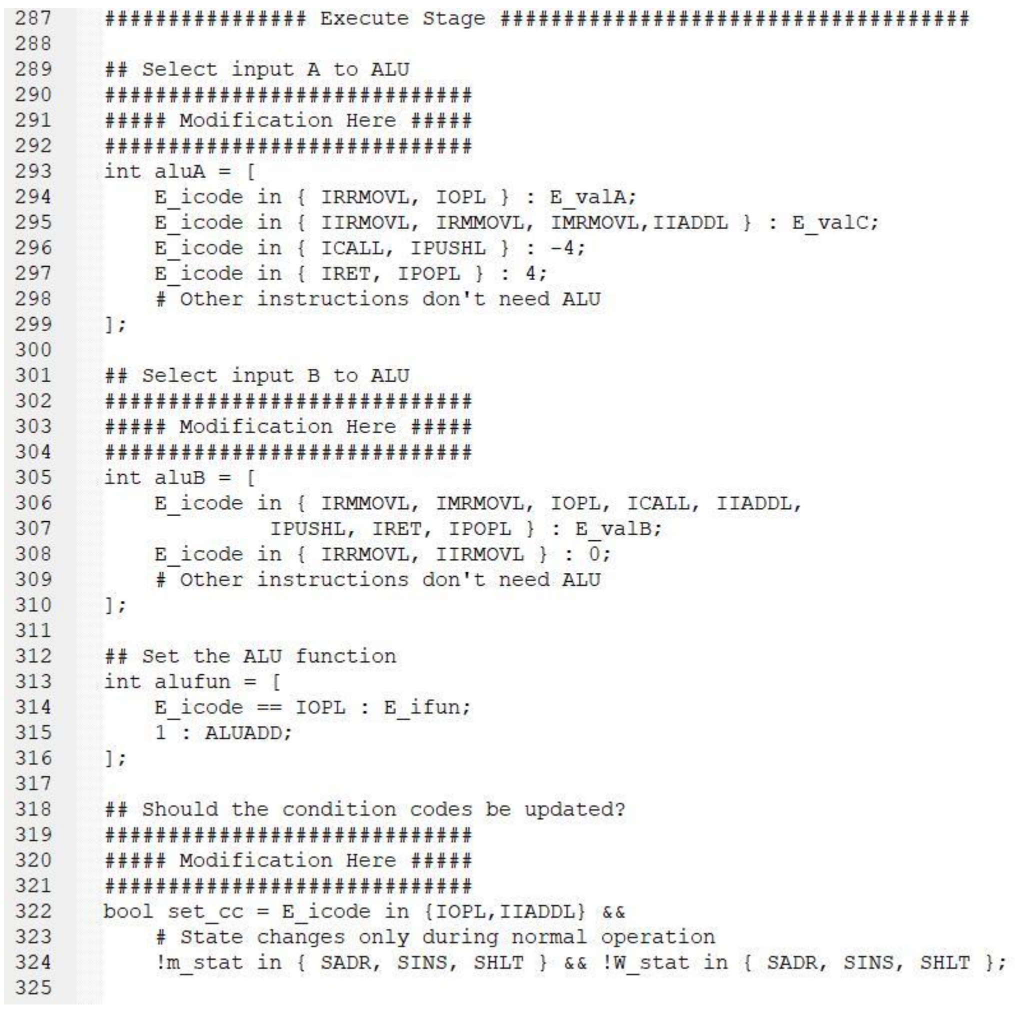
In decode stage: both the B source and the E destination are D_rB.

In execute stage: constant number saved in E_valC and register value in E_valB, add it to condition code as well.

```
# Is instruction valid?
#############################
##### Modification Here #####
#############################
bool instr_valid = f_icode in
    { INOP, IHALT, IRRMOVL, IIRMOVL, IRMMOVL, IMRMOVL,
      IOPL, IJXX, ICALL, IRET, IPUSHL, IPOPL,IIADDL };

# Determine status code for fetched instruction
int f_stat = [
    imem_error: SADR;
    !instr_valid : SINS;
    f_icode == IHALT : SHLT;
    1 : SAOK;
];

# Does fetched instruction require a regid byte?
#############################
##### Modification Here #####
#############################
bool need_regids =
    f_icode in { IRRMOVL, IOPL, IPUSHL, IPOPL,IIADDL,
             IIRMOVL, IRMMOVL, IMRMOVL };

# Does fetched instruction require a constant word?
#############################
##### Modification Here #####
#############################
bool need_valC =
    f_icode in { IIRMOVL, IRMMOVL, IMRMOVL, IJXX, ICALL,IIADDL };

# Predict next value of PC
int f_predPC = [
    f_icode in { IJXX, ICALL } : f_valC;
    1 : f_valP;
];
```



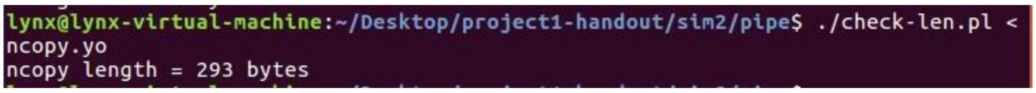
```
################ Decode Stage ######################################


## What register should be used as the A source?
int d_srcA = [
    D_icode in { IRRMOVL, IRMMOVL, IOPL, IPUSHL  } : D_rA;
    D_icode in { IPOPL, IRET } : RESP;
    1 : RNONE; # Don't need register
];

## What register should be used as the B source?
#############################
##### Modification Here #####
#############################
int d_srcB = [
    D_icode in { IOPL, IRMMOVL, IMRMOVL,IIADDL  } : D_rB;
    D_icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
    1 : RNONE;  # Don't need register
];

## What register should be used as the E destination?
#############################
##### Modification Here #####
#############################
int d_dstE = [
    D_icode in { IRRMOVL, IIRMOVL, IOPL,IIADDL} : D_rB;
    D_icode in { IPUSHL, IPOPL, ICALL, IRET } : RESP;
    1 : RNONE;  # Don't write any register
];

```



```
################ Execute Stage #####################################

## Select input A to ALU
#############################
##### Modification Here #####
#############################
int aluA = [
    E_icode in { IRRMOVL, IOPL } : E_valA;
    E_icode in { IIRMOVL, IRMMOVL, IMRMOVL,IIADDL } : E_valC;
    E_icode in { ICALL, IPUSHL } : -4;
    E_icode in { IRET, IPOPL } : 4;
    # Other instructions don't need ALU
];

## Select input B to ALU
#############################
##### Modification Here #####
#############################
int aluB = [
    E_icode in { IRMMOVL, IMRMOVL, IOPL, ICALL, IIADDL,
                 IPUSHL, IRET, IPOPL } : E_valB;
    E_icode in { IRRMOVL, IIRMOVL } : 0;
    # Other instructions don't need ALU
];

## Set the ALU function
int alufun = [
    E_icode == IOPL : E_ifun;
    1 : ALUADD;
];

## Should the condition codes be updated?
#############################
##### Modification Here #####
#############################
bool set_cc = E_icode in {IOPL,IIADDL} &&
    # State changes only during normal operation
    !m_stat in { SADR, SINS, SHLT } && !W_stat in { SADR, SINS, SHLT };

```

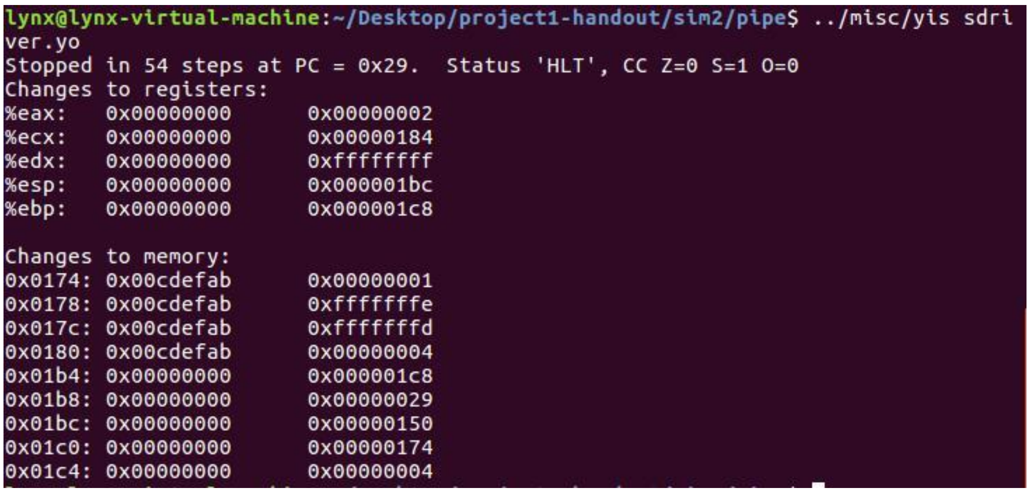
2.3.3 Evaluation

Check-len: 293 bytes < 1000 bytes

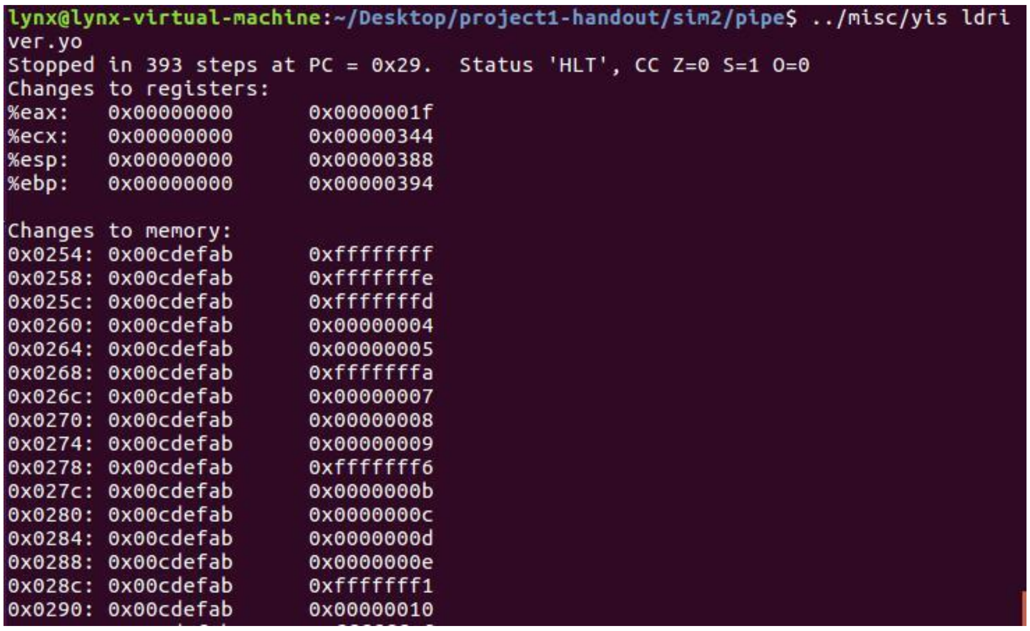


```
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/pipe$ ./check-len.pl <
ncopy.yo
ncopy length = 293 bytes
```

Run **sdriver.yo** and **ldriver.yo** in YIS, value in register %eax is 0x00000002 and 0x0000001f, which means **ncopy.ys** works correctly

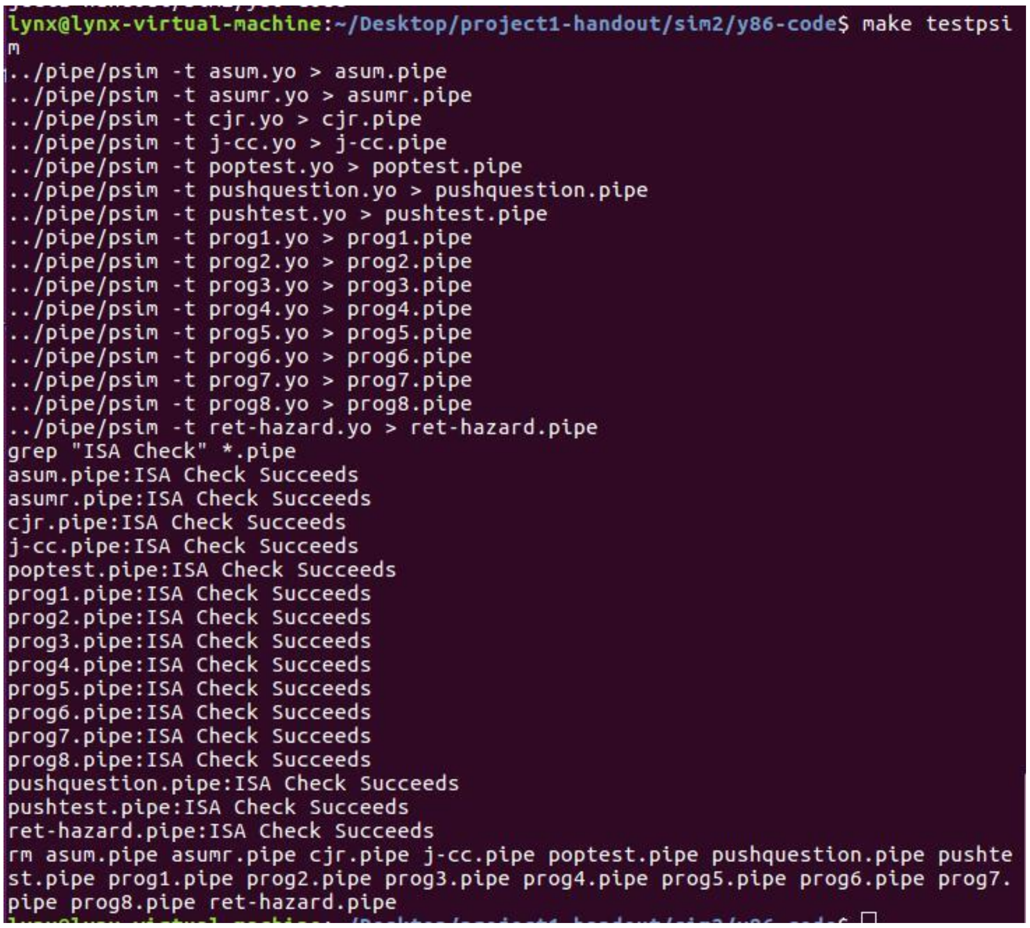


```
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/pipe$ ../misc/yis sdri
ver.yo
Stopped in 54 steps at PC = 0x29.  Status 'HLT', CC Z=0 S=1 O=0
Changes to registers:
%eax:   0x00000000      0x00000002
%ecx:   0x00000000      0x00000184
%edx:   0x00000000      0xffffffff
%esp:   0x00000000      0x000001bc
%ebp:   0x00000000      0x000001c8

Changes to memory:
0x0174: 0x00cdefab      0x00000001
0x0178: 0x00cdefab      0xfffffffe
0x017c: 0x00cdefab      0xfffffffd
0x0180: 0x00cdefab      0x00000004
0x01b4: 0x00000000      0x000001c8
0x01b8: 0x00000000      0x00000029
0x01bc: 0x00000000      0x00000150
0x01c0: 0x00000000      0x00000174
0x01c4: 0x00000000      0x00000004
```



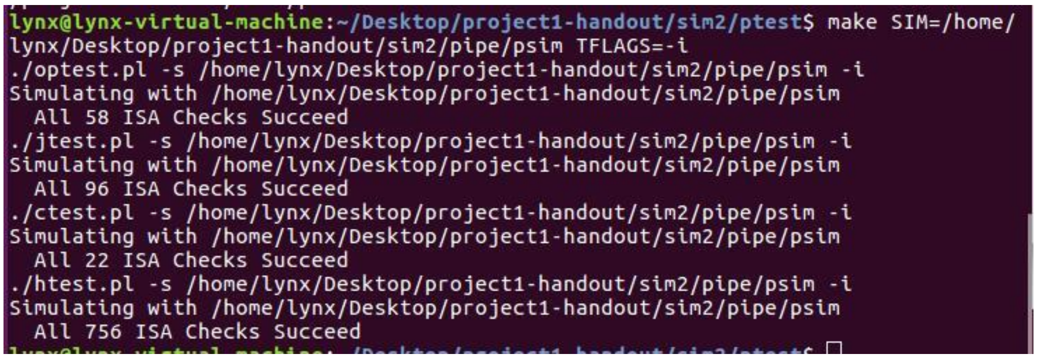
```
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/pipe$ ../misc/yis ldri
ver.yo
Stopped in 393 steps at PC = 0x29.  Status 'HLT', CC Z=0 S=1 O=0
Changes to registers:
%eax:   0x00000000      0x0000001f
%ecx:   0x00000000      0x00000344
%esp:   0x00000000      0x00000388
%ebp:   0x00000000      0x00000394

Changes to memory:
0x0254: 0x00cdefab      0xffffffff
0x0258: 0x00cdefab      0xfffffffe
0x025c: 0x00cdefab      0xfffffffd
0x0260: 0x00cdefab      0x00000004
0x0264: 0x00cdefab      0x00000005
0x0268: 0x00cdefab      0xfffffffa
0x026c: 0x00cdefab      0x00000007
0x0270: 0x00cdefab      0x00000008
0x0274: 0x00cdefab      0x00000009
0x0278: 0x00cdefab      0xfffffff6
0x027c: 0x00cdefab      0x0000000b
0x0280: 0x00cdefab      0x0000000c
0x0284: 0x00cdefab      0x0000000d
0x0288: 0x00cdefab      0x0000000e
0x028c: 0x00cdefab      0xfffffff1
0x0290: 0x00cdefab      0x00000010
```

Test pipeline simulator on the benchmark & with **iaddl**



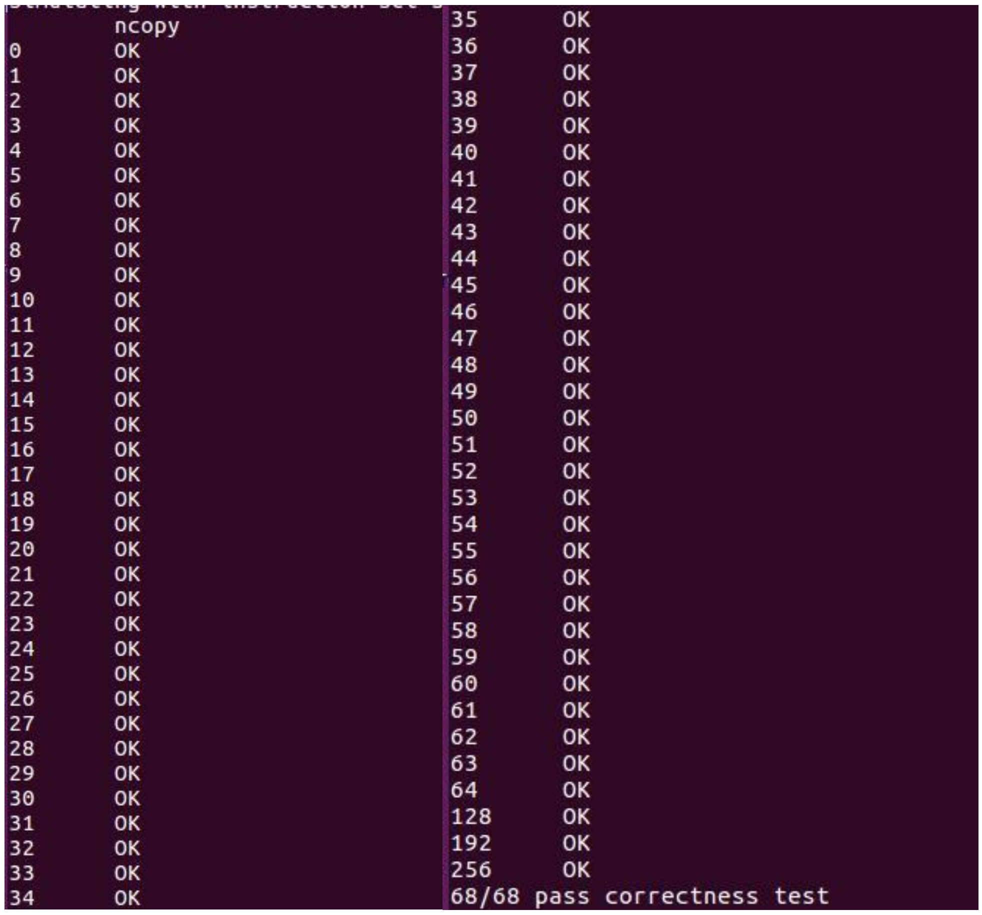
```
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/y86-code$ make testpsi
m
../pipe/psim -t asum.yo > asum.pipe
../pipe/psim -t asumr.yo > asumr.pipe
../pipe/psim -t cjr.yo > cjr.pipe
../pipe/psim -t j-cc.yo > j-cc.pipe
../pipe/psim -t poptest.yo > poptest.pipe
../pipe/psim -t pushquestion.yo > pushquestion.pipe
../pipe/psim -t pushtest.yo > pushtest.pipe
../pipe/psim -t prog1.yo > prog1.pipe
../pipe/psim -t prog2.yo > prog2.pipe
../pipe/psim -t prog3.yo > prog3.pipe
../pipe/psim -t prog4.yo > prog4.pipe
../pipe/psim -t prog5.yo > prog5.pipe
../pipe/psim -t prog6.yo > prog6.pipe
../pipe/psim -t prog7.yo > prog7.pipe
../pipe/psim -t prog8.yo > prog8.pipe
../pipe/psim -t ret-hazard.yo > ret-hazard.pipe
grep "ISA Check" *.pipe
asum.pipe:ISA Check Succeeds
asumr.pipe:ISA Check Succeeds
cjr.pipe:ISA Check Succeeds
j-cc.pipe:ISA Check Succeeds
poptest.pipe:ISA Check Succeeds
prog1.pipe:ISA Check Succeeds
prog2.pipe:ISA Check Succeeds
prog3.pipe:ISA Check Succeeds
prog4.pipe:ISA Check Succeeds
prog5.pipe:ISA Check Succeeds
prog6.pipe:ISA Check Succeeds
prog7.pipe:ISA Check Succeeds
prog8.pipe:ISA Check Succeeds
pushquestion.pipe:ISA Check Succeeds
pushtest.pipe:ISA Check Succeeds
ret-hazard.pipe:ISA Check Succeeds
rm asum.pipe asumr.pipe cjr.pipe j-cc.pipe poptest.pipe pushquestion.pipe pushte
st.pipe prog1.pipe prog2.pipe prog3.pipe prog4.pipe prog5.pipe prog6.pipe prog7.
pipe prog8.pipe ret-hazard.pipe
```

Test pipeline simulator on with extensive regression **iaddl**

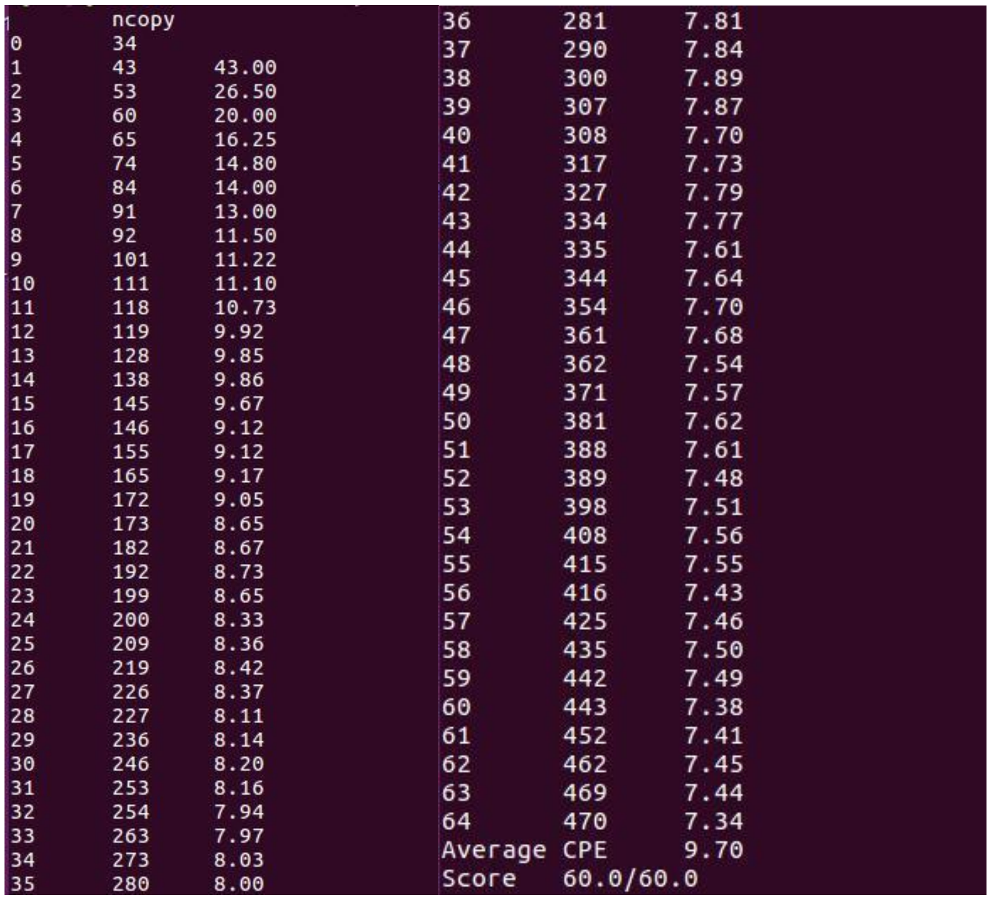


```
lynx@lynx-virtual-machine:~/Desktop/project1-handout/sim2/ptest$ make SIM=/home/
lynx/Desktop/project1-handout/sim2/pipe/psim TFLAGS=-i
./optest.pl -s /home/lynx/Desktop/project1-handout/sim2/pipe/psim -i
Simulating with /home/lynx/Desktop/project1-handout/sim2/pipe/psim
  All 58 ISA Checks Succeed
./jtest.pl -s /home/lynx/Desktop/project1-handout/sim2/pipe/psim -i
Simulating with /home/lynx/Desktop/project1-handout/sim2/pipe/psim
  All 96 ISA Checks Succeed
./ctest.pl -s /home/lynx/Desktop/project1-handout/sim2/pipe/psim -i
Simulating with /home/lynx/Desktop/project1-handout/sim2/pipe/psim
  All 22 ISA Checks Succeed
./htest.pl -s /home/lynx/Desktop/project1-handout/sim2/pipe/psim -i
Simulating with /home/lynx/Desktop/project1-handout/sim2/pipe/psim
  All 756 ISA Checks Succeed
```

Check correctness: all passed, benchmark: Average CPE = 9.70



```
        ncopy
0       OK
1       OK
2       OK
3       OK
4       OK
5       OK
6       OK
7       OK
8       OK
9       OK
10      OK
11      OK
12      OK
13      OK
14      OK
15      OK
16      OK
17      OK
18      OK
19      OK
20      OK
21      OK
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
```



```
        ncopy
0       34
1       43      43.00
2       53      26.50
3       60      20.00
4       65      16.25
5       74      14.80
6       84      14.00
7       91      13.00
8       92      11.50
9       101     11.22
10      111     11.10
11      118     10.73
12      119     9.92
13      128     9.85
14      138     9.86
15      145     9.67
16      146     9.12
17      155     9.12
18      165     9.17
19      172     9.05
20      173     8.65
21      182     8.67
22      192     8.73
23      199     8.65
24      200     8.33
25      209     8.36
26      219     8.42
27      226     8.37
28      227     8.11
29      236     8.14
30      246     8.20
31      253     8.16
32      254     7.94
33      263     7.97
34      273     8.03
35      280     8.00
36      281     7.81
37      290     7.84
38      300     7.89
39      307     7.87
40      308     7.70
41      317     7.73
42      327     7.79
43      334     7.77
44      335     7.61
45      344     7.64
46      354     7.70
47      361     7.68
48      362     7.54
49      371     7.57
50      381     7.62
51      388     7.61
52      389     7.48
53      398     7.51
54      408     7.56
55      415     7.55
56      416     7.43
57      425     7.46
58      435     7.50
59      442     7.49
60      443     7.38
61      452     7.41
62      462     7.45
63      469     7.44
64      470     7.34
Average CPE     9.70
Score   60.0/60.0
```

3. Conclusion

3.1 Problems

1. At first we thought the program in Part A could not use stack operation, so we didn't set up a stack frame. However, we could not successfully makefile, thus we realize it is necessary to set up a stack frame.

2. We didn't change the original code before Loop at first. When we figured out the ncopy operation for the first time, we failed to make the average CPI < 10. Then we found out if the source block was empty, it would judge twice, both ahead of Loop and in the Remain. Thus, we removed the judgement before Loop and successfully made the average CPI < 10.

3. At the beginning, we wrote a ncopy program with the Loop with 8 read and write operations in a cycle. But we found it made the Remain part so long and hard for reader to understand. We decided to execute 4 read and write operations in a cycle to make the code clearer.

3.2 Achievements

1. We use an ingenious way in which we execute read-and-write operations in pair to eliminate bubbles and we decrease the number of comparisons by conducting copy operations in batch of size 4.

2. To make the code easy to understand, we add abundant comments to explain the function.

3. In this project, we have deepened our understanding of pipeline and some of the basic concepts of computer architecture.

4. Through learning and completing this project together, we have deepened our understanding of each other and established a profound friendship that is as important as the knowledge we have gained from this project.