Philip Geramian

CSE-671

Lab 3-1

Homework #3

For this lab we had to take some provided C code and compile it to assembly using GCC. As the lab did not call for a specific architecture to run this on I ran it on a machine running the Intel instruction set, thus my assembly file is in that language. Looking through what the compiler did, it seems that it was able to translate the provided C code to assembly quite efficiently in my opinion. As this is a fairly simple program to being with, there is not much that can be done in a simplification process unless we wish to remove unneeded parts of code. For example our simple program sets three variables to non-changeable values, thus we can remove parts of code if we preprocess the data, so since a = 1, a will always be larger than 0, thus the need to have a case where we do a step o addition when a is less than 0 is unneeded as our code should never have this case, unless someone sets a to a value below 0. I am not sure however if this is the objective of the assignment in terms of simplifying the assembly code, as this only results in code that would produce the same values, but at the cost of using hard coded values in place of the programs logic flow. As the program provided produces no data output, there are no test results or images to be seen. The original C code and commented assembly code follows and should be included in this folder too.

Lab3\_1.c

int a=1, b=2, c=3;

int proc1(int a, int b, int c)

{

if (a > 0)

return b-c;

else

return b+c;

}

int main()

{

int w, x, y;

w=proc1(a,b,c);

switch (w)

{

case 0: x=3; break;

case 1: x=4; break;

case 2: x=7; break;

default: x=9; break;

}

y=w+x;

return 0;

}

Lab3\_1.s

# Compiled from Lab3\_1.c using GCC-5.3.0 on an Intel powered machine

# Declaration of global variable a, b, c

.globl \_a

.data

.align 2

\_a:

.long 1 # a=1

.globl \_b

.align 2

\_b:

.long 2 # b=2

.globl \_c

.align 2

\_c:

.long 3 # c=3

# The function we have (in C proc1, asm \_proc1)

.text

.globl \_proc1

\_proc1:

LFB0:

pushq %rbp # "protect" rbp from being overwritten

LCFI0:

movq %rsp, %rbp

LCFI1:

movl %edi, -4(%rbp) # move in a

movl %esi, -8(%rbp) # move in b

movl %edx, -12(%rbp) # move in c

cmpl $0, -4(%rbp) # is a > 0?

jle L2 # Jump to L2 if false

movl -8(%rbp), %eax # move b to eax

subl -12(%rbp), %eax # subtract b from c store in eax

jmp L3 # jump to L3

L2:

movl -8(%rbp), %edx # move b into edx

movl -12(%rbp), %eax # move c into eax

addl %edx, %eax # add b and c

L3:

popq %rbp # restore rbp

LCFI2:

ret # return

# The Main function

LFE0:

.globl \_main

\_main:

LFB1:

pushq %rbp # protect rbp

LCFI3:

movq %rsp, %rbp

LCFI4:

subq $16, %rsp

movl \_c(%rip), %edx # move c into edx

movl \_b(%rip), %ecx # move b into ecx

movl \_a(%rip), %eax # move a into eax

movl %ecx, %esi

movl %eax, %edi

call \_proc1 # call our function

movl %eax, -8(%rbp)

movl -8(%rbp), %eax

cmpl $1, %eax # are we case 1?

je L6 # jump to case 1

cmpl $2, %eax # are we case 2?

je L7 # jump to case 2

testl %eax, %eax

jne L11 # jump to default

# case 0

movl $3, -4(%rbp) # x=3

jmp L9

L6: # case 1

movl $4, -4(%rbp) # x=4

jmp L9

L7: # case 2

movl $7, -4(%rbp) # x=9

jmp L9

L11: # default

movl $9, -4(%rbp) # x=9

nop

L9:

movl -8(%rbp), %edx

movl -4(%rbp), %eax

addl %edx, %eax # y=w+x

movl %eax, -12(%rbp)

movl $0, %eax

leave

LCFI5:

ret # end of main

# Not sure what the rest of this code does. Perhaps unneeded but placed by GCC?

LFE1:

.section \_\_TEXT,\_\_eh\_frame,coalesced,no\_toc+strip\_static\_syms+live\_support

EH\_frame1:

.set L$set$0,LECIE1-LSCIE1

.long L$set$0

LSCIE1:

.long 0

.byte 0x1

.ascii "zR\0"

.byte 0x1

.byte 0x78

.byte 0x10

.byte 0x1

.byte 0x10

.byte 0xc

.byte 0x7

.byte 0x8

.byte 0x90

.byte 0x1

.align 3

LECIE1:

LSFDE1:

.set L$set$1,LEFDE1-LASFDE1

.long L$set$1

LASFDE1:

.long LASFDE1-EH\_frame1

.quad LFB0-.

.set L$set$2,LFE0-LFB0

.quad L$set$2

.byte 0

.byte 0x4

.set L$set$3,LCFI0-LFB0

.long L$set$3

.byte 0xe

.byte 0x10

.byte 0x86

.byte 0x2

.byte 0x4

.set L$set$4,LCFI1-LCFI0

.long L$set$4

.byte 0xd

.byte 0x6

.byte 0x4

.set L$set$5,LCFI2-LCFI1

.long L$set$5

.byte 0xc

.byte 0x7

.byte 0x8

.align 3

LEFDE1:

LSFDE3:

.set L$set$6,LEFDE3-LASFDE3

.long L$set$6

LASFDE3:

.long LASFDE3-EH\_frame1

.quad LFB1-.

.set L$set$7,LFE1-LFB1

.quad L$set$7

.byte 0

.byte 0x4

.set L$set$8,LCFI3-LFB1

.long L$set$8

.byte 0xe

.byte 0x10

.byte 0x86

.byte 0x2

.byte 0x4

.set L$set$9,LCFI4-LCFI3

.long L$set$9

.byte 0xd

.byte 0x6

.byte 0x4

.set L$set$10,LCFI5-LCFI4

.long L$set$10

.byte 0xc

.byte 0x7

.byte 0x8

.align 3

LEFDE3:

.subsections\_via\_symbols