**Course**: ENCM 369

**Lab Section:** B03

**Lab 3**

**Student Name**: Mitchell Sawatzky

**Date Submitted**: Feb 5, 2016

Exercise A

|  |  |
| --- | --- |
| **File** | **Message** |
| bad-align.asm | Error in /Users/Mitchell/Desktop/School/Y2T2 ENCM 369/Labs ENCM/3/exA/bad-align.asm line 12: Runtime exception at 0x00400010: fetch address not aligned on word boundary 0x10010002 |
| null-ptr.asm | Error in /Users/Mitchell/Desktop/School/Y2T2 ENCM 369/Labs ENCM/3/exA/null-ptr.asm line 16: Runtime exception at 0x00400004: address out of range 0x00000000 |
| overflow.asm | Error in /Users/Mitchell/Desktop/School/Y2T2 ENCM 369/Labs ENCM/3/exA/overflow.asm line 6: Runtime exception at 0x00400008: arithmetic overflow |

Exercise C

functions.asm

# stub2.asm

# ENCM 369 Winter 2016 Lab 3

# This program has complete start-up and clean-up code, and a "stub"

# main function. It's exactly the same as stub1.asm from Lab 2, except

# that comments have been added to help with the organization of main.

# BEGINNING of start-up & clean-up code. Do NOT edit this code.

.data

exit\_msg\_1:

.asciiz "\*\*\*About to exit. main returned "

exit\_msg\_2:

.asciiz ".\*\*\*\n"

main\_rv:

.word 0

.text

# adjust $sp, then call main

addi $t0, $zero, -32 # $t0 = 0xffffffe0

and $sp, $sp, $t0 # round $sp down to multiple of 32

jal main

nop

# when main is done, print its return value, then halt the program

sw $v0, main\_rv

la $a0, exit\_msg\_1

addi $v0, $zero, 4

syscall

nop

lw $a0, main\_rv

addi $v0, $zero, 1

syscall

nop

la $a0, exit\_msg\_2

addi $v0, $zero, 4

syscall

nop

addi $v0, $zero, 10

syscall

nop

# END of start-up & clean-up code.

# Below is the stub for main. Edit it to give main the desired behaviour.

.data

earth: .word 0x30000

.globl earth

.text

.globl main

main:

# PROLOGUE

addi $sp, $sp, -12 # allocate 3 stack slots

sw $ra, 8($sp) # save $ra

sw $s0, 4($sp) # save $s0

sw $s1, 0($sp) # save #s1

# BODY

addi $s0, $zero, 0x7000 # car = 0x7000

addi $s1, $zero, 0x3000 # truck = 0x3000

# set up a registers for call to murcury

addi $a0, $zero, 2 # $a0 = 2

addi $a1, $zero, 3 # $a1 = 3

addi $a2, $zero, 4 # $a2 = 4

addi $a3, $zero, 6 # $a3 = 6

jal mercury # $v0 = murcury(2,3,4,6)

add $s1, $s1, $v0 # truck += $v0

la $t0, earth # $t0 = earth

add $t0, $t0, $s1 # $t0 += truck

add $s0, $s0, $t0 # car += $t0

add $v0, $zero, $zero # return value from main = 0

# EPILOGUE

lw $s1, 0($sp) # recover $s1

lw $s0, 4($sp) # recover $s0

lw $ra, 8($sp) # recover $ra

addi $sp, $sp, 12 # decallocate 3 stack slots

jr $ra # return

mercury:

# PROLOGUE

addi $sp, $sp, -32 # allocate 8 stack slots

sw $ra, 28($sp) # save $ra

sw $s0, 24($sp) # save $s0

sw $s1, 20($sp) # save $s1

sw $s2, 16($sp) # save $s2

sw $s3, 12($sp) # save $s3

sw $s4, 8($sp) # save $s4

sw $s5, 4($sp) # save $s5

sw $s6, 0($sp) # save $s6

add $s0, $zero, $a0 # save $a0 in $s0

add $s1, $zero, $a1 # save $a1 in $s1

add $s2, $zero, $a2 # save $a2 in $s2

add $s3, $zero, $a3 # save $a3 in $s3

# BODY

# beta = venus(third, fourth)

add $a0, $zero, $s2 # $a0 = third

add $a1, $zero, $s3 # $a1 = fourth

jal venus # $v0 = venus(third, fourth)

add $s5, $zero, $v0 # beta = $v0

# gamma = venus(second, third)

add $a0, $zero, $s1 # $a0 = second

add $a1, $zero, $s2 # $a1 = third

jal venus # $v0 = venus(second, third)

add $s6, $zero, $v0 # gamma = $v0

# alpha = venus(fourth, first)

add $a0, $zero, $s3 # $a0 = fourth

add $a1, $zero, $s0 # $a1 = first

jal venus # $v0 = venus(fourth, first)

add $s4, $zero, $v0 # alpha = $v0

# setup return value

add $v0, $s4, $s5 # r.v. = alpha + beta

add $v0, $v0, $s6 # r.v. += gamma

# EPILOGUE

lw $s6, 0($sp) # recover $s6

lw $s5, 4($sp) # recover $s5

lw $s4, 8($sp) # recover $s4

lw $s3, 12($sp) # recover $s3

lw $s2, 16($sp) # recover $s2

lw $s1, 20($sp) # recvoer $s1

lw $s0, 24($sp) # recover $s0

lw $ra, 28($sp) # recover $ra

addi $sp, $sp, 32 # deallocate 8 stack slots

jr $ra # return

venus:

# BODY

# setup return value

sll $t0, $a1, 7 # $t0 = 128 \* $a1

add $v0, $a0, $t0 # r.v. = $a0 + $t0

jr $ra # return

Exercise E

# BEGINNING of start-up & clean-up code. Do NOT edit this code.

.data

exit\_msg\_1:

.asciiz "\*\*\*About to exit. main returned "

exit\_msg\_2:

.asciiz ".\*\*\*\n"

main\_rv:

.word 0

.text

# adjust $sp, then call main

addi $t0, $zero, -32 # $t0 = 0xffffffe0

and $sp, $sp, $t0 # round $sp down to multiple of 32

jal main

nop

# when main is done, print its return value, then halt the program

sw $v0, main\_rv

la $a0, exit\_msg\_1

addi $v0, $zero, 4

syscall

nop

lw $a0, main\_rv

addi $v0, $zero, 1

syscall

nop

la $a0, exit\_msg\_2

addi $v0, $zero, 4

syscall

nop

addi $v0, $zero, 10

syscall

nop

# END of start-up & clean-up code.

.data

aaa: .word 11, 11, 3, -11

.globl aaa

bbb: .word 200, -300, 400, 500

.globl bbb

ccc: .word -2, -3, 2, 1, 2, 3

.globl ccc

.text

.globl main

main:

# PROLOGUE

addi $sp, $sp, -16 # allocate 4 stack slots

sw $ra, 12($sp) # save $ra

sw $s0, 8($sp) # save $s0

sw $s1, 4($sp) # save $s1

sw $s2, 0($sp) # save $s2

# BODY

addi $s2, $zero, 1000 # blue = 1000

# red = special\_sum(aaa, 4, 10)

la $a0, aaa # $a0 = aaa

addi $a1, $zero, 4 # $a1 = 4

addi $a2, $zero, 10 # $a2 = 10

jal special\_sum # $v0 = special\_sum(aaa, 4, 10)

add $s0, $zero, $v0 # red = $v0

# green = special\_sum(bbb, 4, 200)

la $a0, bbb # $a0 = bbb

addi $a1, $zero, 4 # $a1 = 4

addi $a2, $zero, 200 # $a2 = 200

jal special\_sum # $v0 = special\_sum(aaa, 4, 200)

add $s1, $zero, $v0 # green = $v0

# blue += special\_sum(ccc, 6, 500) - red + green

la $a0, ccc # $a0 = ccc

addi $a1, $zero, 6 # $a1 = 6

addi $a2, $zero, 500 # $a2 = 500

jal special\_sum # $v0 = special\_sum(ccc, 6, 500)

add $s2, $s2, $v0 # blue += $v0

add $s2, $s2, $s1 # blue += green

sub $s2, $s2, $s0 # blue -= red

# setup main r.v.

add $v0, $zero, $zero # r.v. = 0

# EPILOGUE

lw $s2, 0($sp) # recover $s2

lw $s1, 4($sp) # recover $s1

lw $s0, 8($sp) # recover $s0

lw $ra, 12($sp) # recover $ra

addi $sp, $sp, 16 # decallocate 4 stack slots

jr $ra # return

special\_sum:

# PROLOGUE

addi $sp, $sp, -24 # allocate 6 stack slots

sw $ra, 20($sp) # save $ra

sw $s0, 16($sp) # save $s0

sw $s1, 12($sp) # save $s1

sw $s2, 8($sp) # save $s2

sw $s3, 4($sp) # save $s3

sw $s4, 0($sp) # save $s0

add $s0, $zero, $a2 # $s0 = b

add $s1, $zero, $a0 # $s1 = x

add $s2, $zero, $a1 # $s2 = n

# BODY

add $s3, $zero, $zero # result = 0

add $s4, $zero, $zero # i = 0

L0: sll $t0, $s4, 2 # $t0 = i \* 4

add $t0, $t0, $s1 # $t0 += x

lw $a0, ($t0) # $a0 = \*x

add $a1, $zero, $s0 # $a1 = b

jal saturate # $v0 = saturate(x[i], b)

add $s3, $s3, $v0 # result += $v0

addi $s4, $s4, 1 # i++

slt $t0, $s4, $s2 # $t0 = (i < n)

beq $t0, $zero, L1 # if ($t0 == 0) goto L1

j L0 # goto L0

L1: add $v0, $zero, $s3 # r.v. = result

# EPILOGUE

lw $s4, 0($sp) # recover $s4

lw $s3, 4($sp) # recover $s3

lw $s2, 8($sp) # recover $s2

lw $s1, 12($sp) # recover $s1

lw $s0, 16($sp) # recover $s0

lw $ra, 20($sp) # recover $ra

addi $sp, $sp, 24 # deallocate 6 stack slots

jr $ra # return

saturate:

# BODY

add $v0, $zero, $a0 # r.v. = x

slt $t0, $a1, $a0 # $t0 = (bound < x)

beq $t0, $zero, L2 # if ($t0 == 0) goto L2

add $v0, $zero, $a1 # r.v. = bound

L2: sub $a1, $zero, $a1 # bound = 0 - bound

slt $t0, $a0, $a1 # $t0 = (x < bound)

beq $t0, $zero, L3 # if ($t0 == 0) goto L3

add $v0, $zero, $a1 # r.v. = bound

L3: jr $ra # return

Exercise F

# swap.asm

# ENCM 369 Winter 2016 Lab 3 Exercise F

# BEGINNING of start-up & clean-up code. Do NOT edit this code.

.data

exit\_msg\_1:

.asciiz "\*\*\*About to exit. main returned "

exit\_msg\_2:

.asciiz ".\*\*\*\n"

main\_rv:

.word 0

.text

# adjust $sp, then call main

addi $t0, $zero, -32 # $t0 = 0xffffffe0

and $sp, $sp, $t0 # round $sp down to multiple of 32

jal main

nop

# when main is done, print its return value, then halt the program

sw $v0, main\_rv

la $a0, exit\_msg\_1

addi $v0, $zero, 4

syscall

nop

lw $a0, main\_rv

addi $v0, $zero, 1

syscall

nop

la $a0, exit\_msg\_2

addi $v0, $zero, 4

syscall

nop

addi $v0, $zero, 10

syscall

nop

# END of start-up & clean-up code.

# int foo[] = { 0x700, 0x600, 0x500, 0x400, 0x300, 0x200, 0x100 }

.data

.globl foo

foo: .word 0x700, 0x600, 0x500, 0x400, 0x300, 0x200, 0x100

# int main(void)

#

.text

.globl main

main:

addi $sp, $sp, -32

sw $ra, 0($sp)

la $t0, foo # $t0 = &foo[0]

addi $a0, $t0, 0 # $a0 = &foo[0]

addi $a1, $t0, 24 # $a1 = &foo[6]

jal swap

la $t0, foo # $t0 = &foo[0]

addi $a0, $t0, 4 # $a0 = &foo[1]

addi $a1, $t0, 20 # $a1 = &foo[5]

jal swap

la $t0, foo # $t0 = &foo[0]

addi $a0, $t0, 8 # $a0 = &foo[2]

addi $a1, $t0, 16 # $a1 = &foo[4]

jal swap

add $v0, $zero, $zero

lw $ra, 0($sp)

addi $sp, $sp, 32

jr $ra

# void swap(int \*left, int \*right)

#

.text

.globl swap

swap:

lw $t0, ($a1) # $t0 = \*right

lw $t1, ($a0) # $t1 = \*left

sw $t1, ($a1) # \*right = $t1

sw $t0, ($a0) # \*left = $t0

jr $ra