Name: Aarsh Shah

Student Name: Aarsh Shah

Lab Section: B02

Course: Computer Organization – ENCM 369

Lab #: 3

Exercise A: Instructions that try to do bad things

Messages:

Bad-align.asm:

“Assemble: operation completed successfully.

Go: running bad-align.asm

Error in A:\All_Files_Organized\Year 2\OneDrive - University of Calgary\Second Year\Semester 2\ENCM 369\Labs\Lab3\exA\bad-align.asm line 12: Runtime exception at 0x00400010: Load address not aligned to word boundary 0x10010002

Go: execution terminated with errors.”

null-ptr.asm:

“Assemble: operation completed successfully.

Go: running null-ptr.asm

Error in A:\All_Files_Organized\Year 2\OneDrive - University of Calgary\Second Year\Semester 2\ENCM 369\Labs\Lab3\exA\null-ptr.asm line 16: Runtime exception at 0x00400004: address out of range 0x00000000

Go: execution terminated with errors.”

write-to-text.asm:

“Assemble: operation completed successfully.

Go: running write-to-text.asm

Error in A:\All_Files_Organized\Year 2\OneDrive - University of Calgary\Second Year\Semester 2\ENCM 369\Labs\Lab3\exA\write-to-text.asm line 10: Runtime exception at 0x00400008: Cannot write directly to text segment!0x00400000

Go: execution terminated with errors.”

Exercise C: Translating a simple program with procedure calls

functions.asm

```
1  # stub1.asm
2  # ENCM 369 Winter 2023
3  # This program has complete start-up and clean-up code, and a "stub"
4  # main function.
5
6  # BEGINNING of start-up & clean-up code.  Do NOT edit this code.
7          .data
8  exit_msg_1:
9          .asciz  "***About to exit. main returned "
10 exit_msg_2:
11         .asciz  ".***\n"
12 main_rv:
13         .word   0
14
15         .text
16         # adjust sp, then call main
17         andi    sp, sp, -32             # round sp down to multiple of 32
18         jal     main
19
20         # when main is done, print its return value, then halt the program
21         sw      a0, main_rv, t0
22         la      a0, exit_msg_1
23         li      a7, 4
24         ecall
25         lw      a0, main_rv
26         li      a7, 1
27         ecall
28         la      a0, exit_msg_2
29         li      a7, 4
30         ecall
31         lw      a0, main_rv
32         addi    a7, zero, 93    # call for program exit with exit status that is in a0
33         ecall
34 # END of start-up & clean-up code.
35
36 # Global variables.
37         .data
38         .globl  train
39 train:  .word   0x20000
40
41 # Below is the stub for main. Edit it to give main the desired behaviour.
42         .text
43         .globl  main
44
45 main:
46         # prologue
47         addi    sp, sp, -12     # Increments the stack pointer down 12 bytes
48         sw      ra, 8(sp)       # Stores return address to stack pointer
```

```
49          sw      s1, 4(sp)
50          sw      s0, 0(sp)
51
52          # body
53          li      s1, 0xa000      # boat = 40960
54          li      s0, 0x3000      # plane = 12288
55
56          # Passing 4 constant arguments to procA
57          addi    a0, zero, 6     # a0 = 6
58          addi    a1, zero, 4     # a1 = 4
59          addi    a2, zero, 3     # a2 = 3
60          addi    a3, zero, 2     # a3 = 2
61          jal     procA           # calls procA function
62          add     s1, s1, a0      # s1 (boat) += a0 (return value from procA)
63
64          sub     t0, s1, s0      # t0 = boat - plane (s1 - s0)
65
66          la      t1, train       # load address of train
67          lw      t2, (t1)        # load value from t1 address
68
69          add     t3, t2, t0      # t1 = t2 + t0
70          sw      t3, (t1)        # train = t1
71
72
73          # epilogue
74          lw      s0, 0(sp)       # reload the values from the stacl
75          lw      s1, 4(sp)
76          lw      ra, 8(sp)
77          addi    sp, sp, 12
78
79          li      a0, 0   # return value from main = 0
80          jr      ra
81
82          .globl  procA
83  procA:
84          # prologue
85          addi    sp, sp, -32     # Increments stack pointer down by 32 bytes
86          sw      ra, 28(sp)      # saves ra for return to caller
87          sw      s6, 24(sp)      # saves s6 for main
88          sw      s5, 20(sp)      # saves s5 for main
89          sw      s4, 16(sp)      # saves s4 for main
90          sw      s3, 12(sp)      # saves s3 for main
91          sw      s2, 8(sp)       # saves s2 for main
92          sw      s1, 4(sp)       # saves s1 for main
93          sw      s0, 0(sp)       # saves s0 for main
94
95          add     s0, a0, zero    # copies first from a0 to s0
96          add     s1, a1, zero    # copies second from a1 to s1
```

```
97          add     s2, a2, zero    # copies third from a2 to s2
98          add     s3, a3, zero    # copies fourth from a3 to s3
99
100         # body
101         add     a0, s3, zero    # sets a0 = fourth
102         add     a1, s2, zero    # sets a1 = third
103         jal     procB           # calls procB function
104         add     s5, a0, zero    # gets return value from procB (s5 = a0)
105
106         add     a0, s1, zero    # sets a0 = second
107         add     a1, s0, zero    # sets a1 = first
108         jal     procB           # calls procB function
109         add     s6, a0, zero    # gets return value from procB (s6 = a0)
110
111         add     a0, s2, zero    # sets a0 = third
112         add     a1, s3, zero    # sets a1 = fourth
113         jal     procB           # calls procB function
114         add     s4, a0, zero    # gets return value from procB (s4 = a0)
115
116         add     t0, s5, s6      # t0 = s5 + s6
117         add     a0, t0, s4      # a0 = t0 + s4
118
119         # Prologue
120
121         lw      s0, 0(sp)       # loads s0 for main
122         lw      s1, 4(sp)       # loads s1 for main
123         lw      s2, 8(sp)       # loads s2 for main
124         lw      s3, 12(sp)      # loads s3 for main
125         lw      s4, 16(sp)      # loads s4 for main
126         lw      s5, 20(sp)      # loads s5 for main
127         lw      s6, 24(sp)      # loads s6 for main
128         lw      ra, 28(sp)      # copy backed-up ra for the correct return location
129         addi    sp, sp, 32      # Increments stack pointer back up by 32 bytes
130
131         jr      ra              # Jump back to the return address
132
133         .globl  procB
134
135 procB:
136         slli    t0, a0, 8       # t0 = 2^8 (256) * a0
137         add     a0, t0, a1      # a0 = t0 + a1
138
139         jr      ra
140
```

Exercise E: More practice with functions

```
# stub1.asm
# ENCM 369 Winter 2023
# This program has complete start-up and clean-up code, and a "stub"
# main function.

# BEGINNING of start-up & clean-up code.  Do NOT edit this code.
        .data
exit_msg_1:
        .asciz  "***About to exit. main returned "
exit_msg_2:
        .asciz  ".***\n"
main_rv:
        .word   0

        .text
        # adjust sp, then call main
        andi    sp, sp, -32             # round sp down to multiple of 32
        jal     main

        # when main is done, print its return value, then halt the program
        sw      a0, main_rv, t0
        la      a0, exit_msg_1
        li      a7, 4
        ecall
        lw      a0, main_rv
        li      a7, 1
        ecall
        la      a0, exit_msg_2
        li      a7, 4
        ecall
        lw      a0, main_rv
        addi    a7, zero, 93    # call for program exit with exit status that is in a0
        ecall
# END of start-up & clean-up code.


# Global variables
        .data
        # int aaa[] = { 11, 11, 3, -11}
        .globl  aaa
aaa:    .word   11, 11, 3, -11

        # bbb[] = { 200, -300, 400, 500 }
        .globl  bbb
bbb:    .word   200, -300, 400, 500

        # int ccc[] = { -2, -3, 2, 1, 2, 3 }
        .globl  ccc
```

```
49  ccc:    .word   -2, -3, 2, 1, 2, 3
50
51  # Below is the stub for main. Edit it to give main the desired behaviour.
52          .text
53          .globl  main
54  main:
55          # prologue
56          addi    sp, sp, -16     # Increments the stack pointer down 16 bytes
57          sw      ra, 12(sp)      # Stores RA and s-registers on stack
58          sw      s2, 8(sp)
59          sw      s1, 4(sp)
60          sw      s0, 0(sp)
61
62          # body
63          li      s2, 1000        # s2 = 1000
64
65          addi    a0, zero, 10    # a0 = 10
66          la      a1, aaa         # a1 = aaa
67          addi    a2, zero, 4     # a2 = 4
68          jal     special_sum
69          add     s0, zero, a0
70
71          addi    a0, zero, 200   # a0 = 200
72          la      a1, bbb         # a1 = bbb
73          addi    a2, zero, 4     # a2 = 4
74          jal     special_sum
75          add     s1, zero, a0
76
77          addi    a0, zero, 500   # a0 = 500
78          la      a1, ccc         # a1 = ccc
79          addi    a2, zero, 6     # a2 = 6
80          jal     special_sum
81          add     t0, a0, s0      # t0 = a0 + s0
82          add     t1, t0, s1      # t1 = t0 + s1
83          add     s2, s2, t1      # s2 += t1
84
85
86          # epilogue
87          lw      s0, 0(sp)       # loads registers back for prodecure call
88          lw      s1, 4(sp)
89          lw      s2, 8(sp)
90          lw      ra, 12(sp)      # loads return address to stack pointer
91          addi    sp, sp, 16      # Increments the stack pointer up 16 bytes
92
93
94          li      a0, 0           # return value from main = 0
95          jr      ra
96
```

```
97          .globl  clamp
98  clamp:
99          sub     t0, zero, a0            # t0 = -(a0)
100         bge     a1, t0, else_if         # if (a1 >= t0) goto else_if
101         add     a0, zero, t0            # a0 = t0
102         j       end_func
103
104 else_if:
105         ble     a1, a0, end_if          # if (a1 <= a0) goto end_if
106         j       end_func
107 end_if:
108         add     a0, zero, a1            # a0 = a1
109
110 end_func:
111         jr ra
112
113
114         .globl  special_sum
115
116 special_sum:
117
118         # prologue
119         addi    sp, sp, -24     # Increments the stack pointer down 20 bytes
120         sw      ra, 20(sp)      # Stores return address to stack pointer
121         sw      s4, 16(sp)
122         sw      s3, 12(sp)
123         sw      s2, 8(sp)
124         sw      s1, 4(sp)
125         sw      s0, 0(sp)
126
127         add     s0, zero, a0    # s0 = a0 (bound)
128         add     s1, zero, a1    # s1 = a1 (x)
129         add     s2, zero, a2    # s2 = a2 (n)
130
131         # body
132         add     s3, zero, zero  # s3 (result) = 0
133         add     s4, zero, zero  # s4 (i) = 0
134
135 for_start:
136         bge     s4, s2, for_end
137
138         add     a0, zero, s0
139
140         slli    t1, s4, 2       # t1 = i << 2
141         add     t2, s1, t1      # t2 = &x[k]
```

```
142         lw      a1, (t2)        # a1 = x[k]
143         jal     clamp
144         add     s3, s3, a0      # s3 += a0 (clamp return value)
145
146         addi    s4, s4, 1       # i++
147         j       for_start
148 for_end:
149         add     a0, s3, zero    # a0 = s3 (result)
150
151         # epilogue
152         lw      s0, 0(sp)       # loads registers back for prodecure call
153         lw      s1, 4(sp)
154         lw      s2, 8(sp)
155         lw      s3, 12(sp)
156         lw      s4, 16(sp)
157         lw      ra, 20(sp)      # loads return address to stack pointer
158         addi    sp, sp, 24      # Increments the stack pointer up 20 bytes
159
160         jr      ra
161
```

Exercise F: A function to swap contents of integer variables

```
# swap.asm
# ENCM 369 Winter 2023 Lab 3 Exercise F

# BEGINNING of start-up & clean-up code.  Do NOT edit this code.
        .data
exit_msg_1:
        .asciz  "***About to exit. main returned "
exit_msg_2:
        .asciz  ".***\n"
main_rv:
        .word   0

        .text
        # adjust sp, then call main
        andi    sp, sp, -32             # round sp down to multiple of 32
        jal     main

        # when main is done, print its return value, then halt the program
        sw      a0, main_rv, t0
        la      a0, exit_msg_1
        li      a7, 4
        ecall
        lw      a0, main_rv
        li      a7, 1
        ecall
        la      a0, exit_msg_2
        li      a7, 4
        ecall
        lw      a0, main_rv
        addi    a7, zero, 93    # call for program exit with exit status that is in a0
        ecall
# END of start-up & clean-up code.

# int foo[] =  { 0x600, 0x500, 0x400, 0x300, 0x200, 0x100 };
        .data
        .globl  foo
foo:    .word   0x600, 0x500, 0x400, 0x300, 0x200, 0x100

# int main(void)
#
        .text
        .globl  main
main:
        addi    sp, sp, -32
        sw      ra, 0(sp)
```

```
46
47          la      t0, foo         # t0 = &foo[0]
48          addi    a0, t0, 20      # a0 = &foo[5]
49          addi    a1, t0, 0       # a1 = &foo[0]
50          jal     swap
51
52          # Students: Replace this comment with code to correctly
53          # implement the next two calls to swap in main in swap.c.
54
55          la      t0, foo         # t0 = &foo[0]
56          addi    a0, t0, 16      # a0 = &foo[4]
57          addi    a1, t0, 4       # a1 = &foo[1]
58          jal     swap
59
60          la      t0, foo         # t0 = &foo[0]
61          addi    a0, t0, 12      # a0 = &foo[3]
62          addi    a1, t0, 8       # a1 = &foo[2]
63          jal     swap
64
65          add     a0, zero, zero
66          lw      ra, 0(sp)
67          addi    sp, sp, 32
68          jr      ra
69
70  # void swap(int *left, int *right)
71  #
72          .text
73          .globl  swap
74  swap:
75          # Students: Replace this comment with code to make swap
76          # do its job correctly.
77
78          # Prologue
79          addi    sp, sp, -12     # Increments the stack pointer down 12 bytes
80          sw      s2, 8(sp)
81          sw      s1, 4(sp)
82          sw      s0, 0(sp)
83
84          add     s0, zero, a0    # s0 = a0 (left)
85          add     s1, zero, a1    # s1 = a1 (right)
86
87          # Body
88          lw      t0, (s0)        # t0 = *left
89          lw      t1, (s1)        # t1 = *right
90
```

```
91          add     s2, zero, t0
92          sw      t1, (s0)
93          sw      s2, (s1)
94
95          # prologue
96          lw      s0, 0(sp)
97          lw      s1, 4(sp)
98          lw      s2, 8(sp)
99          addi    sp, sp, 12
100
101         jr      ra
102
```