**Course: Computer Organization – ENCM 369**
**Lab #:** Lab 3
**Instructor Name:** Norm Bartley
**Student Name:** Stephen Ravelo
**Lab Section:** B03
**Date submitted:** February 7, 2025

# Exercise A

| Program | Message |
|---|---|
| bad-align.asm | Error in ...\bad-align.asm line 12: Runtime exception at 0x00400010: Load address not aligned to word boundary 0x10010002 |
| null-ptr.asm | Error in ...\null-ptr.asm line 16: Runtime exception at 0x00400004: address out of range 0x00000000 |
| write-to-text.asm | Error in ...\write-to-text.asm line 10: Runtime exception at 0x00400008: Cannot write directly to text segment!0x00400000 |

# Exercise C

## functions.asm

```
# stub1.asm
# ENCM 369 Winter 2025
# 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.
```

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

    .data
    .globl    banana
banana:
    .word     0x20000

# int main(void)
#
# local variable register
# int apple s0
# int orange    s1
    .text
    .globl  main
main:
    # prologue
    addi    sp, sp, -32
    sw  ra, 8(sp)
    sw  s1, 4(sp)
    sw  s0, 0(sp)

    # body
    addi    s0, zero, 0x600 # apple = 0x600
    addi    s1, zero, 0x700 # orange = 0x700

    addi    a0, zero, 5 # a0 = 5
    addi    a1, zero, 4 # a1 = 4
    addi    a2, zero, 3 # a2 = 3
    addi    a3, zero, 2 # a3 = 2
    jal funcA
    add s1, s1, a0  # orange += r.v. of funcA

    la  t0, banana  # t0 = &banana[0]
    lw  t1, (t0)    # t1 = *t0
    sub t2, s0, s1  # t2 = apple - orange
    add t1, t1, t2  # t1 += t2
    sw  t1, (t0)

    # epilogue
    lw  s0, 0(sp)
    lw  s1, 4(sp)
    lw  ra, 8(sp)
    addi    sp, sp, 32
    li      a0, 0   # return value from main = 0
    jr  ra
```

```
# int funcA(int first, int second, int third, int fourth)
#
# local variable register
# int car   s4
# int truck s5
# int bus   s6
    .text
    .globl  funcA
funcA:
    # prologue
    addi    sp, sp, -32
    sw  ra, 28(sp)
    sw  s6, 24(sp)
    sw  s5, 20(sp)
    sw  s4, 16(sp)
    sw  s3, 12(sp)
    sw  s2, 8(sp)
    sw  s1, 4(sp)
    sw  s0, 0(sp)

    # body
    mv  s0, a0      # s0 = first
    mv  s1, a1      # s1 = second
    mv  s3, a3      # s3 = fourth
    mv  s2, a2      # s2 = third

    add a0, zero, s3    # a0 = s3
    add a1, zero, s2    # a1 = s2
    jal     funcB
    add s4, zero, a0    # car = r.v. of funcB(fourth, third)

    add a0, zero, s1    # a0 = s1
    add a1, zero, s0    # a1 = s0
    jal     funcB
    add s6, zero, a0    # bus = r.v. of funcB(second, first)

    add a0, zero, s2    # a0 = s2
    add a1, zero, s3    # a1 = s3
    jal     funcB
    add s5, zero, a0    # truck = r.v. of funcB(third, fourth)

    add t0, s4, s5  # t0 = car + truck
    add a0, t0, s6  # a0 = t0 + bus
```

```
    # epilogue
    lw  s0, 0(sp)
    lw  s1, 4(sp)
    lw  s2, 8(sp)
    lw  s3, 12(sp)
    lw  s4, 16(sp)
    lw  s5, 20(sp)
    lw  s6, 24(sp)
    lw  ra, 28(sp)
    addi    sp, sp, 32
    jr  ra

# int funcB(int y, int z)
    .text
    .globl  funcB
funcB:
    slli    t0, a1, 6   # t0 = z * 64
    add a0, a0, t0  # t1 = y + t0
    jr  ra
```

# Exercise E

## stub1.asm

```
# stub1.asm
# ENCM 369 Winter 2025
# 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.
```

```
    .data
    .globl  aaa
aaa:    .word   11, 11, 3, -11, 11
    .globl  bbb
bbb:    .word   200, -300, 400, 500
    .globl  ccc
ccc:    .word   -3, -4, 3, 2, 3, 4

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

# int main(void)
#
# local variable register
# int alpha s0
# int beta  s1
# int gamma s2
    .text
    .globl  main
main:
    # prologue
    addi    sp, sp, -32
    sw  ra, 12(sp)
    sw  s2, 8(sp)
    sw  s1, 4(sp)
    sw  s0, 0(sp)

    # body
    addi    s2, zero, 2000

    la  a0, aaa     # a0 = &aaa[0]
    addi    a1, zero, 5 # a1 = 5
    addi    a2, zero, 10    # a2 = 10
    jal sum_of_sats
    add s0, zero, a0    # alpha = r.v. of sum_of_sats

    la  a0, bbb     # a0 = &bbb[0]
    addi    a1, zero, 4 # a1 = 4
    addi    a2, zero, 300   # a2 = 300
    jal sum_of_sats
    add s1, zero, a0    # beta = r.v. of sum_of_sats
```

```
    la  a0, ccc     # a0 = &ccc[0]
    addi    a1, zero, 6 # a1 = 6
    addi    a2, zero, 3 # a2 = 3
    jal sum_of_sats
    add t0, s0, s1  # t0 = alpha + beta
    add s2, a0, t0  # gamma += r.v. of sum_of_sats + t0

    # epilogue
    lw  s0, 0(sp)
    lw  s1, 4(sp)
    lw  s2, 8(sp)
    lw  ra, 12(sp)
    addi    sp, sp, 32
    li      a0, 0   # return value from main = 0
    jr  ra

# int sat(int b, int x)
    .text
    .globl  sat
sat:
    sub t0, zero, a0    # t0 = -b
    bge a1, t0, L3  # if (x >= -b) goto L3
    add a0, t0, zero    # a0 = -b
    j   L4
L3:
    ble a1, a0, L4  # if (x > b) goto L4
    add a0, a1, zero    # a0 = x
L4:
    jr  ra
```

```
# int sum_of_sats(const int *a, int n, int max_mag)
#
# local variable register
# int result    s3
    .text
    .globl  sum_of_sats
sum_of_sats:
    # prologue
    addi    sp, sp, -32
    sw  ra, 16(sp)
    sw  s3, 12(sp)
    sw  s2, 8(sp)
    sw  s1, 4(sp)
    sw  s0, 0(sp)
    mv  s0, a0      # s0 = &a[0]
    mv  s1, a1      # s1 = n
    mv  s2, a2      # s2 = max_mag

    # body
    add s3, zero, zero  # result = 0
    ble s1, zero, L1    # if (n <= 0) goto L1
    addi    s1, s1, -1  # n--
L2:
    add a0, zero, s2    # a0 = max_mag
    slli    t0, s1, 2   # t0 = n << 2
    add t1, s0, t0  # t1 = &a[n]
    lw  a1, (t1)    # a1 = a[n]
    jal sat
    add s3, s3, a0  # result += r.v. of sat
    addi    s1, s1, -1  # n--
    bge s1, zero, L2    # if (n >= 0) goto L2
    mv  a0, s3      # a0 = result

L1:
    # epilogue
    lw  s0, 0(sp)
    lw  s1, 4(sp)
    lw  s2, 8(sp)
    lw  s3, 12(sp)
    lw  ra, 16(sp)
    addi    sp, sp, 32
    jr  ra
```

# Exercise F

## swap.asm

```
# swap.asm
# ENCM 369 Winter 2025 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)

    la  t0, foo         # t0 = &foo[0]
    mv      a0, t0      # a0 = &foo[0]
    addi    a1, t0, 20  # a1 = &foo[5]
    jal swap

    la  t0, foo         # t0 = &foo[0]
    addi    a0, t0, 4   # a0 = &foo[1]
    addi    a1, t0, 16  # a1 = &foo[4]
    jal swap

    la  t0, foo         # t0 = &foo[0]
    addi    a0, t0, 8   # a0 = &foo[2]
    addi    a1, t0, 12  # a1 = &foo[3]
    jal swap

    add a0, zero, zero
    lw  ra, 0(sp)
    addi    sp, sp, 32
    jr  ra

# void swap(int *p, int *q)
#
    .text
    .globl  swap
swap:
    lw  s0, (a1)
    lw  t0, (a0)
    sw  t0, (a1)
    sw  s0, (a0)
    jr  ra
```