Name: Aarsh Shah

Student Name: Aarsh Shah

Lab Section: B02

Course: Computer Organization – ENCM 369

Lab #: 2

Exercise A: Finding machine code for instructions.

sub s1, s1, t5:

First, I converted the registers to their actual register number from chart 6.1(s1 = x9 = 01001, t5 = x30 = 11110). Then I read the op, funct7 and funct3 values from pg. 333. Then using the format funct7_rs2_rs1_funct3_rd_op I converted it machine code.

Machine Code: 0100000_11110_01001_000_01001_0110011

sw s4, (t3):

First I went to appendix B and got the op and funct3 values for sw. Then I got the values for s4 and t3 from chart 6.1 (s4 = x20 = 10100, t3 = x28 = 11100). Then simply used the format: imm(11:5)_rs2_rs1_funct3_imm(4:0)_op to convert to machine code.

Machine Code:

0000000_11100_10100_010_00000_0100011

lw t6, 72(s3):

First I went to appendix B and got the op and funct3 values for lw. Then I got the values for t6 and s3 from chart 6.1 (s3 = x19 = 10011, t6 = x31 = 11111). Then I converted 72 to two's complement binary (72 = 000001001000). Then simply used the format: imm_rs1_funct3_rd_op to convert to machine code.

Machine Code:

000001001000_10011_010_11111_0000011

addi s7, s6, -16 # Hint: Involves 12-bit two's-complement:

First, I went to appendix B and got the op and funct3 values for addi. Then I got the values for s7 and s6 from chart 6.1 (s6 = x22 = 10110, t7 = x23 = 10111). Then I converted -16 to two's complement binary (-16 = 111111110000). Then simply used the format: imm_rs1_funct3_rd_op to convert to machine code.

Machine Code:

111111110000_10110_000_10111_0010011

Exercise C: Translating C code that has a main function

```
# array-sum2C.asm
# ENCM 369 Winter 2023 Lab 2 Exercise C Part 3

# Start-up and clean-up code copied from stub1.asm

# 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 abc[ ] = {-32, -8, -4, -16, -128, -64}
        .globl  abc
abc:    .word   -32, -8, -4, -16, -128, -64

# Hint for checking that the original program works:
# The sum of the six array elements is -252, which will be represented
# as 0xffffff04 in a RISC-V GPR.

# Hint for checking that your final version of the program works:
# The maximum of the four array elements is -4, which will be represented
# as 0xfffffffc in a RISC-V GPR.
```

```
49
50
51  # int main(void)
52  #
53  # local variable        register
54  #   int *p              s0
55  #   int *end            s1
56  #   int sum             s2
57  #   int max             s3  (to be used when students enhance the program)
58
59          .text
60          .globl  main
61  main:
62          la      s0, abc                 # p = abc
63          addi    s1, s0, 24              # end = p + 6
64          add     s2, zero, zero          # sum = 0
65          lw      s3, (s0)                # max = *p
66  L1:
67          beq     s0, s1, L3              # if (p == end) goto L3
68          lw      t0, (s0)                # t0 = *p
69          add     s2, s2, t0              # sum += t0
70          addi    s0, s0, 4               # p++
71          bgt     t0, s3, L2              # if(max < t0) goto L2
72          j       L1
73  L2:
74          addi    s3, t0, 0               # max = *p + 0
75          j       L1
76
77  L3:
78          add     a0, zero, zero          # return value from main = 0
79          jr      ra
80
```

Exercise D: Practice with arrays, loops, and if statements

```
# stub1.asm
# ENCM 369 Winter 2023 Lab 2
# 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
        .globl  alpha
alpha:  .word   0xb1, 0xe1, 0x91, 0xc1, 0x81, 0xa1, 0xf1, 0xd1
.globl  beta
beta:   .word   0x0, 0x10, 0x20, 0x30, 0x40, 0x50, 0x60, 0x70


# int main(void)
#
# local variable        register
#   int *p              s0
#   int *guard          s1
```

```
#   int min             s2
#   int j               s3
#   int k               s4



# Below is the stub for main. Edit it to give main the desired behaviour.
        .text
        .globl  main
main:
        la      s0, alpha       # p = alpha
        addi    s1, s0, 32      # guard = p + 8
        lw      s2, (s0)        # min = *p
        addi    s0, s0, 4       # p++
L1:
        beq     s0, s1, L3      # if(p == guard) goto L3
        lw      t0, (s0)        # t0 = *p
        bge     t0, s2, L2      # if (t0 >= min) goto L2
        addi    s2, t0, 0       # min = t0 + 0
L2:
        addi    s0, s0, 4       # p++
        j       L1
L3:
        la      t5, alpha       # t0 = alpha
        la      t6, beta        # t6 = beta
        add     s3, zero, zero  # j = 0
        addi    s4, zero, 7     # k = 7
        addi    t0, zero, 8     # t0 = 8
L4:
        bge     s3, t0, L5      # if(j >= 8) goto L5

        slli    t1, s4, 2       # t1 = k << 2
        add     t2, t6, t1      # t2 = &beta[k]
        lw      t3, (t2)        # t3 = beta[k]

        slli    t4, s3, 2       # t4 = j << 2
        add     t2, t5, t4      # t5 = &alpha[j]
        sw      t3, (t2)        # alpha[j] = t3

        addi    s3, s3, 1       # j++
        addi    s4, s4, -1      # k--
        j       L4
L5:
        li      a0, 0           # return value from main = 0
        jr      ra
```