EECS 2021 F23 LAB 03 REPORT

BY: Rayhaan Yaser Mohammed
Student ID: 219739887

Introduction: This lab focuses on exploring more on the topics of Stacks, Procedures and recursions using the Assembly Language instructions.

C1: Input and Otput (ecall, DC)

ecall: this is call is used to input an integer value in rvs

1.1 c3a.asm

```
ecall   x6, x0, 5
ecall   x7, x0, 5
add     x5, x6, x7
```

This source codes uses ecall to input two integers and adds the two integers and saves the result in x5.

1.2 c3b.asm

```
ecall   x6,x0,5
ecall   x7,x0,5
add     x5,x6,x7
ecall   x0,x5,0
```

This source code performs the same intrictuction but now it outputs the value in register x5.

1.3 c3c.asm

```
loop:   ecall   x6, x0, 5
        ecall   x7, x0, 5
        add     x5, x6, x7
        ecall   x0, x5, 0
        beq     x0, x0, loop
```

This source code uses a loop to keep running the instructions of getting 2 integers and outputting the sum stored i register x5.

1.4 c3d.asm

```
c1:     DC    "integer:"
c2:     DC    "sum:"
s1:     DC    "Inputs two integers\nand prints the sum. \0"
        ld    x28,   c1(x0)
        ld    x29,   c2(x0)
        addi  x30,   x0,    s1
        ecall x0,    x30,   4      ;info String
loop:   ecall x6,    x28,   5      ;integer1
        ecall x7,    x28,   5      ;integer2
        add   x5,    x6,    x7
        ecall x1,    x29,   3      ;"sum"
        ecall x0,    x5,    0      ;the result
        beq   x0,    x0,    loop
```

This source code prints the characters using the read_integer ecall and the uses print_character ecall . The first 2 sequence of characters are each lesser than 8 bytes of storage whereas the lasy Sequence of characters are larger than 8 bytes thus we need to print this usig the print_string ecall. This all is achieved using the loop and assigning the characters their specific labelling.

C2: The STACK

2.1 c2a05.asm

```
s1:           DC    "No of ints:\0"
s2:           DC    "Int"
s3:           DC    ":"
s4:           DC    " "
STACK:        EQU   0x100000
              lui   sp,    STACK>>12 ;
              add   x5,    x0,    s1     ;
              ecall x1,    x5,    4      ;out question
              ecall x5,    x0,    5      ;inp No. of inputs
              addi  x6,    x0,    1      ;counter
loop1:        ld    x7,    s2(x0)
              ecall x1,    x7,    3      ;out Int
              ecall x1,    x6,    0      ;out Index
              ld    x7,    s3(x0)        ;
```

```
            ecall   x8,     x7,     5       ;out :, in #
            sd      x8,     0(sp)           ;push
            addi    sp,     sp,     -8      ;push
            addi    x6,     x6,      1      ;counter
            bge     x5,     x6,     loop1 ;
            addi    x6,     x0,     1       ;counter
loop2:      ld      x7,     s2(x0)
            ecall   x1,     x7,3            ;out Int
            ecall   x1,     x6,     0       ;out Index
            ld      x7,     s3(x0)          ;
            ecall   x1,     x7,     3       ;out :, in #
            addi    sp,     sp,     8       ;pop
            ld      x8,     0(sp)            ;pop
            ecall   x0,     x8,     0 ; out :, in #
            addi    x6,     x6,     1
            bge     x5,     x6,     loop2
```

This source code is used to inputs a sequence of integers numbers and are then pushed onto a stack in order of the input then the code reverses the order by popping the stack and then outputs it in the reversed order.

C3: Leaf and Non-Leaf Procedures (jalr, jal)

3.1 c3a05.asm

```
str1:   DC          "sampled text\0"
        addi        x6,     x0,     str1    ;output
        ecall       x0,     x6,     4
        ebreak      x0,     x0,     0       ;finish
```

This source code defines a given text as a string using the DC assembler command and then it outputs it then finishes it by executing ebreak.\

3.2 c3b05.asm

```
str1:       DC "sampled text\0"
            addi a2,x0,str1+6     ;character addr
            jal x1, delch1
            addi x6,x0,str1       ;output
            ecall x0,x6,4
            ebreak x0,x0,0        ; finish
delch1:     lb x5,0(a2)
loop1:      beq x5,x0,end1        ;
            lb x5,1(a2)
            sb x5,0(a2)
            addi a2,a2,1
            jal x0,loop1
end1:       jalr x0,0(x1)         ;return
```

This source code first defines the test string str1, then loads the address of the desired character in register a2 and then finally employees the jal(jump and link) instruction to store and return the address in register x1 and jump to the first instruction of delch1 procedure.

3.3 c3c05.asm

```
str1:
            DC "sampled text\0"
STACK:
            EQU 0x100000              ; stack
            lui sp, STACK >> 12
            addi a2, x0, str1 + 6     ; chaddr
            addi a3, x0, 6            ; #ch
            jal x1, delch
            addi x6, x0, str1         ; output
            ecall x0, x6, 4
            ebreak x0, x0, 0          ; finish

delch1:
            lb x5,0(a2)
```

```
Loop1:
                beq x5,x0,end1          ;
                lb x5,1(a2)
                sb x5,0(a2)
                addi a2,a2,1
                jal x0,loop1
end1:           jalr x0,0(x1)           ;return

delch:
                sd x1, 0(sp)            ; push
                sd s0, -8(sp)           ; push
                sd s1, -16(sp)          ; push
                addi sp, sp, -24        ; push
                addi s0, a2, 0          ;
                addi s1, a3, 0          ;
                bge x0, s1, end2
loop2:
                jal x1, delch1          ; jal delch1
                addi a2, s0, 0
                addi s1, s1, -1
                bne s1, x0, loop2
end2:
                addi sp, sp, 24         ; pop
                ld x1, 0(sp)            ; pop
                ld s0, -8(sp)           ; pop
                ld s1, -16(sp)          ; pop
                jalr x0, 0(x1)          ; return
```

This source code uses the stored s1 and s2 registers to hold the values of parameters a2 and a3. As s0 ans 1 are saved register theu preserve their values thus the procedure delch1 preserves the values in s0 and s0 but delch procedure itself also preserves the values so the values stay intact. And the the return address in stored in x1. Then following the convention the values of x1,s0 and s1 are pushed onto the stack at the beginning od delch and then popped back in the end. Then the given examples is then carried out where it deletes the 6 characters starting from the letter d of the first word thus outputting just “sampled”.

3.4 c4a05.asm

```
addi sp,x0,0        ;sp initialization
addi a0,x0,5        ;n=5
jal x1,fact         ;call fcat
```

```
        ebreak x0,x0,0
fact:   blt x0,a0,recu      ;if(0<a0) recursion
        addi a0,x0,1        ;if(a0<=0) return 1
        jalr x0,0(x1)       ;return
recu:   sd x1,-8(sp)        ;push ra
        sd a0,-16(sp)       ;push a0
        addi sp,sp,-16      ;adjust sp
        addi a0,a0,-1       ; a0=a0-1
        jal x1,fact         ;recursive call
        addi sp,sp,16       ;adjust sp
        ld x1,-8(sp)        ;pop ra
        ld x5,-16(sp)       ;pop a0
        mul a0,x5,a0        ;fact(a0)=a0*fact(a0-1)
        jalr    x0, 0(x1)   ;return
```

The fact procedure in the source cpde either return the value of 1 fo a0 <= 0 or calls itself recursively with value of a0-1 each time until the base case is true. The return value in a0 is calculated by multiplying the current value of a0 restored from the stack to x5 with the returned from the recursive call value in a0.

3.5 c4b05.asm

```
        addi a1,x0,1        ;accumalator
        addi a0,x0,5        ;n=5
        jal x1,fact
        ebreak x0,x0,0
fact:   blt x0,a0,recu      ;if(0<a0) recursion
        add a0,x0,a1        ;if(a0<=0) return accumulator
        jalr x0,0(x1)       ;return
recu:   mul a1,a1,a0        ;fact(a0)=a0*fatc(a0-1)
        addi a0,a0,-1       ;a0=a0-1
        jal x0, fact        ; jump instead of a recursive call
```

The above source code performs the same function as the c4.a05.asm but this uses tail recursive version which eliminates the recursion which has very minimal usage of stack and we need to only store it in return address.

Conclusion:

The lab helped in understanding input ad output methods and how to construct simple procedures and also the usage of stack like how to intialize ,push and pop a stack. I also learned more about leaf and non leaf procedures like recursion and tail recursive code.