Exercise 3 : MIPS Programming 2 (Subroutine)

This time, I will learn about subroutines in assembler. In Ex 3, you can use the jal (jump and link) and jr (jump register) instructions in addition to the instructions listed in the instruction table (assemble.html) of assignment 1.

Ex 3.1 - Matrix Multiplication

As a MIPS programming we will create a program to find the product of matrices. Complete a program to calculate the product of two 4 × 4 matrices stored in the memory and check the operation with xspim just as before. Please use the following two matrices as input data.

```
  0  0  0  1       0  1  2  3
  0  2  0  0       4  5  6  7
  0  0  3  0       8  9 10 11
  4  0  0  0      12 13 14 15
```

Hint

a sample program written in C (matmul.c)

Procedure

1. Make the program of Ex.2 using subroutine.
   Remove the part that initializes the non-multiplier and multiplier, and adds the return to the end.
2. Add a main routine that calls the above subroutines to perform matrix multiplication.
   Please use $16 ~ $23 ($s0 ~ $s7) for temporary register. Using $8 to $15 may cause the value to be corrupted by subroutines.

Ex 3.2 - Calculation of factorial by recursive call (THIS EXERCISE IS REMOVED. For report 3, please submit only Ex 3.1)

In the matrix multiplication program of Ex 3.1, by using different sets of temporary registers in the main routine and subroutine, register conflicts were prevented. However, since general procedure calls have a deeper nest structure, it is impossible to avoid all register conflicts with this method alone. Therefore, it is necessary to save the contents of the register to the memory and restore the register contents from the memory by using the stack. The data to be saved also includes the return address stored in $31 (= $ra).

In this exercise, as an example of using the stack, understand the execution of recursive procedures to calculate factorials in textbook p.98. The following program takes N as input data and stores the result in FN.

link of code (ex_3_sample.s)

```
.data
N:  .word 5
FN: .word 0

    ... Change N if necessary ....

.text
main:
    lui $sp, 32767         # Initialization of upper 16bit of $sp(32767 = 16`h7fff of 32`h7fffef
    ori $sp, $sp, 61436         # Initialization of lower 16bit of $$sp(61436 = 16`heffc of 32`h
    lw $a0, N         # $a0(= N) = 5 Modification
    jal fact          # $v0 = fact($a0)
    sw $v0, FN         # FN = $v0
    exit: j exit

fact:
    addi $sp, $sp, -8 # $sp = $sp - 8
    sw $ra, 4($sp)
    sw $a0, 0($sp)

    slti $t0, $a0,1   # $t1 = ($a0 < 1) 1:0
    beq $t0, $0, L1   # if $t1 = 0 then L1

    addi $v0,$0,1     # $v0 = 1
    addi $sp,$sp,8    # $sp = $sp + 8
    jr $ra            # return FN = 1

L1:
    addi $a0,$a0,-1
    jal fact          # fact($a0 - 1)
    lw $a0, 0($sp)
    lw $ra, 4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

In the above program, the mul instruction to execute multiplication is used. In this exercise, this mul is implemented as a subroutine.

Please proceed with the following procedure.

1. Add a subroutine for the product to the above program.
2. Delete the mul instruction of the program and change it to call the subroutine for the product. At that time, an instruction to move the register is inserted as appropriate so that the arguments are passed correctly. Also, calling a subroutine will destroy the value of $ ra, so it will be necessary to restore that value from the stack.
3. Execute with xspim and confirm execution progress of function call. Check the contents of the running stack.
4. Change the value of N to execute, and confirm the execution progress.