# Cmpe344 Fall 2021 FF67
# Experiment #3: Writing a non-leaf procedure

In this experiment, you will write a non-leaf procedure in MIPS to complete a given program.

Once complete, the program will read elements $a_i$ and $b_i$ in pairs from the two input arrays `A` and `B` (given in the data segment) until they end. For each pair, it will write the result of $made\_up(a_i, b_i)$ (defined below) to the output array `M`.

- The arrays `A` and `B` are of equal length.
- Arrays end with a zero, which are not considered as members.
- All the input and output values will be **unsigned words**.
- We guarantee you that $a_i > b_i$ for all $i$.

The following two procedures are already written and **must be left as they are:**

`fibonacci:` Clears all the temporary registers as a challenge.
Takes an argument in `$a0` and returns the Fibonacci number of that order in `$v0`.

`main:` Stores values into saved registers as a challenge.
Reads elements from `A` and `B` pairwise, calls the `made_up` procedure with each pair, and writes the result into the array `M`.

Your task is to write the `made_up` procedure by the following constraints:

1. It should take two arguments via `$a0` and `$a1`, and return the result of

$$made\_up(\$a0, \$a1) = truncate\left(\frac{F(\$a0)\ \cdot\ F(\$a1)}{F(\$a0\ -\ \$a1)}\right)$$

   in `$v0`**,** where $F(n)$ denotes the $n$-th Fibonacci number, and $truncate(n)$ is the integer part of $n$.

2. Each of the three Fibonacci numbers must be obtained using the `fibonacci` procedure.

3. Registers that should be preserved[1] must be preserved.

4. Registers that do not need preserving[1] must be assumed to have been altered after each time calling the `fibonacci` procedure.

[1] See the 7th note in the *MIPS assembly language overview* section of the tutorial.

```
.text
.globl main
main:
    li $s1,1;li $s2,2;li $s3,3  # every time the made_up procedure ends,
    li $s4,4;li $s5,5  # these registers’ original values must be restored
    li $s6,6;li $s7,7  # VALUES WILL CHANGE, use stack to store-restore

    addi $sp, $sp, -4
    sw   $ra, 0($sp)     # push($ra)
    li   $s0, 0          # $s0 = array indexer
main_loop:
    lw   $a0, A($s0)     # $a0 = A[$s0]
    beqz $a0, main_done  # end loop if A[$s0] == 0
    lw   $a1, B($s0)     # $a1 = B[$s0]
    jal  made_up         # $v0 = made_up($a0, $a1)
    sw   $v0, M($s0)     # M[$s0] = $v0
    addi $s0, $s0, 4     # $s0 += 4
    b    main_loop       # loop
main_done:
    sw   $zero, M($s0)   # mark the end of the array M
    lw   $ra, 0($sp)     # pop($ra)
    addi $sp, $sp, 4
    jr   $ra

# DO NOT CHANGE THE CODE ABOVE
made_up:
    # WRITE YOUR CODE HERE
    jr   $ra
# DO NOT CHANGE THE CODE BELOW (you can change the data arrays)

fibonacci:
    li $a1,0;li $a2,0;li $a3,0;li $v1,0 # made_up procedure
    li $t1,0;li $t2,0;li $t3,0          # should not rely on these
    li $t4,0;li $t5,0;li $t6,0          # being preserved, because...
    li $t7,0;li $t8,0;li $t9,0  # all the temporaries are cleared!

    beqz $a0, fibonacci_zero  # fibonacci(0) = 0
    li   $t0, 0
    li   $v0, 1
fibonacci_loop:
    addi $a0, $a0, -1         # $a0 -= 1
    beqz $a0, fibonacci_done  # end loop if $a0 == 0
    add  $v0, $v0, $t0        # $v0 += $t0
    sub  $t0, $v0, $t0        # $t0 = $v0 - $t0
    b    fibonacci_loop       # loop
fibonacci_zero:
    li   $v0, 0
fibonacci_done:
    jr   $ra
.data
A: .word  6,  8, 22,  9, 19,  0
B: .word  5,  1, 17,  8, 10,  0
M: .word  0
```

Running the completed program should write the following array beginning at M:

```
40, 1, 5656893, 714, 6763, 0
```

## Question

In MIPS, assume that the **`or R1, R2, R3`** was not a real instruction, but instead a pseudo-instruction. The assembler would then have to convert it into a sequence of one or more real instructions. Provide such a sequence below. Use **`R1`**, **`R2`**, and **`R3`** to denote the operands passed to **`or`**. Use **`R4`**, **`R5`**, **`R6`**, … to denote any additional register you use.

We have to take two NORS like this:

nor R4, R2, R3

nor R1, R4, R4

a or b = (a nor b) nor (a nor b)