A(2, 2) | = | A(1, A(2, 1))

| = | A(1, A(1, A(2, 0)))

| = | A(1, A(1, A(1, 1)))

| = | A(1, A(1, A(0, A(1, 0))))

| = | A(1, A(1, A(0, A(0, 1))))

| = | A(1, A(1, A(0, 2)))

| = | A(1, A(1, 3))

| = | A(1, A(0, A(1, 2)))

| = | A(1, A(0, A(0, A(1, 1))))

| = | A(1, A(0, A(0, A(0, A(1, 0)))))

| = | A(1, A(0, A(0, A(0, A(0, 1)))))

| = | A(1, A(0, A(0, A(0, 2))))

| = | A(1, A(0, A(0, 3)))

| = | A(1, A(0, 4))

| = | A(1, 5)

| = | A(0, A(1, 4))

| = | A(0, A(0, A(1, 3)))

| = | A(0, A(0, A(0, A(1, 2))))

| = | A(0, A(0, A(0, A(0, A(1, 1)))))

| = | A(0, A(0, A(0, A(0, A(0, A(1, 0))))))

| = | A(0, A(0, A(0, A(0, A(0, A(0, 1))))))

| = | A(0, A(0, A(0, A(0, A(0, 2)))))

| = | A(0, A(0, A(0, A(0, 3))))

| = | A(0, A(0, A(0, 4)))

| = | A(0, A(0, 5))

| = | A(0, 6)

| = | 7

```
# Sources:
# http://stackoverflow.com/questions/7831834/mips-ackermann-function-in-floating-point
# https://github.com/tandasima/Ackermann-Function-MIPS-Assembly/blob/master/Ackermann.s
# http://en.wikipedia.org/wiki/Ackermann_function


                .data
progSummary:  .asciiz   "Welcome to ACKERMANN program\nEnter 2 postive integers to calculate the Ackermann value\nEnter a NEGATIVE number to terminate the program.\n\n"
mPrompt:      .asciiz   "Enter the m value: "
nPrompt:      .asciiz   "Enter the n value: "
result0:      .asciiz   "Ackermann("
result1: .asciiz   ", "
result2: .asciiz   ") is "
newLine:      .asciiz   "\n\n"
errResult:    .asciiz   " * Negative value entered. Program will terminate** "
                .text
main:

        li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a string
        la      $a0, progSummary # loading address for the string to be printed
        syscall                 # perform system call to print the prompt for m value

programInstructions:
        li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a string
        la      $a0, mPrompt    # loading address for the string to be printed
        syscall                 # perform system call to print the prompt for m value
```

```
        li      $v0, 5          # loading system call code 5 for reading an integer from the user
input into register $v0
        syscall                 # calling the OS to read an integer for the m value

        move    $s0, $v0        # copy m value from $v0 to $s0
        move    $t0, $v0        # store m value in temporary register $t0

        bltz    $s0, exit       # exit program if entered value is negative

        li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a
string
        la      $a0, nPrompt    # loading address for the string to be printed
        syscall                 # calling OS to print the prompt for n value

        li      $v0, 5          # loading system call code 5 for reading an integer from the user
input into register $v0
        syscall                 # calling the OS to read an interger for the n value

        move    $s1, $v0        # copy n value from $v0 to $s1
        move    $t1, $v0        # store m value in temporary register $t1

        bltz    $s1, exit       # exit program if entered value is negative

        move    $a0, $s0        # assigning m ($s0) to $a0
        move    $a1, $s1        # assigning n ($s1) to $a1

        jal Ackermann           # calling the Ackermann function

        addi    $sp, $sp, -16   # making stack with a room to store 4 registers
        sw      $s0, 4($sp)     # store $s0 in the stack -- value of m
```

```
        sw      $s1, 8($sp)     # store $s1 in the stack -- value of n
        sw      $s2, 12($sp)    # store $s2 in the stack
        sw      $ra, 0($sp)     # store return address register $ra

        move    $a2, $v0        # move the value in $v0 to $a2
        addi    $sp, $sp, -4    # make stack with room to store one register
        sw      $a0, 0($sp)     # store $a0 in the stack

        la      $a0, result0    # loading address for the string to be printed
        li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a
string
        syscall                 # calling OS to print the prompt for n value

        move    $a0, $t0        # loading address for the string to be printed
        li      $v0, 1          # loading system call code 4 to $v0 appropriate for printing a
string
        syscall                 # calling OS to print the prompt for n value

        la      $a0, result1    # loading address for the string to be printed
        li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a
string
        syscall                 # calling OS to print the prompt for n value

        move    $a0, $t1        # loading address for the string to be printed
        li      $v0, 1          # loading system call code 4 to $v0 appropriate for printing a
string
        syscall                 # calling OS to print the prompt for n value

        la      $a0, result2    # loading address for the string to be printed
```

```
li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a
string
syscall                 # calling OS to print the prompt for n value

move    $a0, $a2        # Print value
li      $v0, 1          # loading system call code 1 to $v0 appropriate for printing an
integer value
syscall                 # calling OS to print the integer value in $v0

la      $a0, newLine    # loading address for the string to be printed
li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a
string
syscall                 # calling OS to print the prompt for n value

lw      $a0, 0($sp)     # restore $a0 from stack
addi    $sp, $sp, 4     # restore $sp, the stack pointer

lw      $ra, 0($sp)     # restore $ra from stack
lw      $s0, 4($sp)     # restore $s0 from stack
lw      $s1, 8($sp)     # restore $s1 from stack
lw      $s2, 12($sp)    # restore $s2 from stack
addi    $sp, $sp, 16    # restore $sp, the stack pointer

j programInstructions   # calling the programInstructions to restart the program
instructions again
```

Ackermann:

```
addi $sp, $sp, -8       # making room in the stack to store 2 registers
sw $s0, 4($sp)          # store $s0 (m) in stack
sw $ra, 0($sp)          # storing return address register $ra in stack
```

```
mZero:
        bne $a0, $zero, nZero    # branch if m is not equal to 0 else m = 0 by default
        addi $v0, $a1,   1       # n + 1 -- to be returned if m == 0
        j done                   # completing the function call

nZero:
        bne $a1, $zero, bothAreGreater # if (n==0) do below else if branch to bothAreGreater
        addi $a0, $a0, -1        # $a0 = m -1
        addi $a1, $zero, 1       # $a1 = 1
        jal   Ackermann          # if (n == 0) return Ackermann(m-1, 1)
        j done                   # completing the function call

bothAreGreater:
        add $s0, $a0, $zero      # Saving m ($s0) for second call
        addi $a1, $a1, -1        # $a1 = n- 1
        jal Ackermann            # Ackermann(m, (n -1))

        addi $a0, $s0, -1        # $a0 = m - 1
        add $a1, $v0, $zero
        jal Ackermann            # Ackermann(m-1,A(m, (n - 1)))
        j   done            # complete the function call

done:
        lw $s0, 4($sp)           # restoring m
        lw $ra, 0($sp)           # restoring the return address
        addi $sp, $sp, 8 # restoring $sp, the stack pointer
        jr $ra                   # return to caller
```

```
exit:
        li      $v0, 4          # loading system call code 4 to $v0 appropriate for printing a
string
        la      $a0, errResult  # loading address for the string to be printed
        syscall                 # calling OS to perform the print operation

        li $v0, 10              # loading system call code 10 to $v0 appropriate for exiting
        syscall                 # calling OS to exit the program
```