# A program that inputs two 6x6 matrix of single-digit integers one row at a time

# and then multiply the integer matrices and print out the results by row

# Inner loop subroutine called matrixCalLoop unrolled 2 times with three iterations of the unrolled loop

.data

matrixArray1: .space 256

matrixArray2: .space 256

matrixResult: .space 256

matrix1Str: .asciiz "6 x 6 Matrix one:"

matrix2Str: .asciiz "6 x 6 Matrix two:"

inputRow: .asciiz "Input Row "

outputRow: .asciiz "Output Row "

space: .asciiz " "

colon: .asciiz ":"

newLine: .asciiz "\n"

inputIntString: .space 64

inputIntLength: .word 64

.text

.globl main

#Main

main:

li $v0,4 #Print string "Matrix one"

la $a0, matrix1Str

syscall

li $v0,4

la $a0, newLine

syscall

jal initializeVariables #Initialize Variables

la $s1, matrixArray1 # $s1 = &matrixArray1[i]

add $t4, $t4, $s1 #$t4 = address of $s1

jal beginInput #Jump to subroutine to take user input for matrix one

li $v0,4 #Print string "Matrix two"

la $a0, matrix2Str

syscall

li $v0,4

la $a0, newLine

syscall

jal initializeVariables #Initialize Variables

la $s2, matrixArray2 #$s2 = &matrixArray2[j]

add $t4, $t4, $s2 #$t4 = address of $s2

jal beginInput

jal initializeVariables #Initialize Variables

la $s0, matrixResult #$s0 gets address matrixResult[k]

jal matrxiMultiply #Jump to subroutine that does matrix multiplication

jal initializeVariables

la $s0, matrixResult

jal printOutput #Jump to subroutine that prints out result

exit:

li $v0, 10

syscall

#Call subroutine to accept userInput and call subroutine to load into array

beginInput:

addi $sp, $sp, -4 #Make room on stack for 1 register

sw $ra, 0($sp) #Save $ra on stack

loadInputLoop:

beq $t6, $t7, endBeginInput

jal userInput #Jump to subroutine that takes user input

jal loadArray #Jump to subroutine that loads user input into array

addi $t6, $t6, 1 #$t6 += 1 (Increment branch index for loopInputLoop)

j loadInputLoop

endBeginInput:

lw $ra, 0($sp) #Restore $ra from stack

addi $sp, $sp, 4 #Restore stack Pointer

jr $ra

#subroutine to take userInput

userInput:

addi $sp, $sp, -8 #Make room on stack for 2 registers

sw $a0, 0($sp) #Save $a0 on stack

sw $ra, 4($sp) #Save $ra on stack

li $v0,4

la $a0,inputRow #Print user inout prompt

syscall

li $v0,1

move $a0,$t8 #Print row number

syscall

li $v0,4

la $a0,colon

syscall

li $v0,4

la $a0,space

syscall

addi $t8, $t8, 1 #Increment row number

lw $a0, 0($sp) #Restore $a0 from stack

lw $ra, 4($sp) #Restore $ra from stack

addi $sp, $sp, 8 #Restore stack Pointer

jr $ra

#End subroutine userInput

#subroutine to load userInput into an array

loadArray:

addi $sp, $sp, -8 #Make room on stack for 2 registers

sw $a0, 0($sp) #save $a0 on stack

sw $ra, 4($sp) #save $ra on stack

li $v0, 8

la $a0, inputIntString #Load user input string

la $a1, inputIntLength

syscall

li $t2, 0

la $s0, inputIntString #$s0 = Address input string array

add $t2, $t2, $s0 #t2 = address of $s0

# Reads one byte at a time, skips when it finds a space char (32)

# Loop exits when found a new line char (10)

while:

lbu $t3, ($t2) #load one byte from string in $t3

beq $t3, 10, endwhile #If $t3 = newline, go to endwhile

beq $t3, 32, foundSpace #If $t3 = space, go to foundSpace to store int in MatrixArray

addi $t3, $t3, -48 #Convert Byte char to int

move $t0, $t3

addi $t2, $t2, 1 #Increment to next byte in user input string

j while #Loop back till space/newline found

foundSpace:

sw $t0, ($t4) #If found space, store int in MatrixArray

li $t0, 0

addi $t4, $t4, 4 #Increment to next word to store the next int

addi $t2, $t2, 1 #Increment to next byte

j while #Loop back to get the next byte in string

endwhile:

sw $t0, ($t4) #Store last element of row

li $t0, 0

addi $t4, $t4, 4

addi $t2, $t2, 1

lw $a0, 0($sp) #Restore $a0 from stack

lw $ra, 4($sp) #Restore $ra from stack

addi $sp, $sp, 8 #Restore stack Pointer

jr $ra

#End subroutine loadArray

matrxiMultiply:

addi $sp, $sp, -4 #Make room on stack for 1 register

sw $ra, 0($sp) #save $ra on stack

li $t8, 0

li $s5, 0

add $t1, $t1, $s1 #$t1 gets address of matrixArray1[i]

add $t2, $t2, $s2 #$t2 gets address of matrixArray2[j]

add $t8, $t8, $s0 #$t8 gets address of matrixResult[k]

move $s6, $t1 #$s6 = address of matrixArray1[i]

move $s7, $t2 #$s7 = address of matrixArray2[j]

rowIncrementLoop: #Loop increments row of matrixArray1[i]

beq $s5, 6, endRowIncrement

rowIndexIncrement:

beq $t0, 6, IndexEnd

li $t7, 0

move $t1, $s6 #$t1 = &matrixArray1[i]

move $t2, $s7 #$t2 = &matrixArray2[j]

li $t6, 0

matrixCalLoop: #Matrix Calculation

beq $t6, 3, matrixCalEnd

lw $t3, 0($t1) #$t3 = matrixArray1[i]

lw $t4, 4($t1) #$t3 = matrixArray1[1+4] / next row element

lw $k0, 0($t2) #$k0 = matrixArray2[j]

lw $k1, 24($t2) #$k0 = matrixArray2[j+24] / next column element

mul $t5, $t3, $k0 #$t5 = matrixArray1[i] \* matrixArray2[j]

mul $t9, $t4, $k1 #$t9 = matrixArray1[i+4] \* matrixArray2[j+24]

addi $t6, $t6, 1 #Increment branch index for matrixCalLoop (stall removed by adding instruction here)

add $s3, $t5, $t9 #$s3 = $t5 + $t9

addi $t1, $t1, 8 #Increment row element (i = 8) (stall removed by adding instruction here)

addi $t2, $t2, 48 #Increment to next column element (j = 24) (stall removed by adding instruction here)

add $t7, $t7, $s3 #$t7 = $t7 + $s3($t5+$t9)

j matrixCalLoop

matrixCalEnd:

move $t1, $s6 #Restore address of $t1 to the address of $s6

move $t2, $s7 #Restore address of $t2 to the address of $s7

sw $t7, ($t8) #Store result of matrixArray1 \* matrixArray2 = matrixResult[k]

addi $t8, $t8, 4 #Increment to store next result at next location in matrix resultArray[k+4]

addi $t2, $t2, 4 #Restore column element [j+4] to begining to perform next set of calculation for [k+4]

move $s7, $t2

addi $t0, $t0, 1 #Increment branch index for rowIndexIncrement

j rowIndexIncrement

IndexEnd:

move $t1, $s6

addi $t1, $t1, 24 #Increment to next row (matriaxArray1[i+24]) to perform next set of row column multiplication

move $s6, $t1

addi $s5, $s5, 1 #Increment branch index for rowIncrementLoop

li $t0, 0

li $t2, 0

add $t2, $t2, $s2

move $s7, $t2

j rowIncrementLoop

endRowIncrement:

lw $ra, 0($sp) #Restore $ra from stack

addi $sp, $sp, 4 #Restore stack Pointer

jr $ra

#Initialize Variables and call subroutine to display the array

printOutput:

addi $sp, $sp, -4 #Make room on stack for 1 register

sw $ra, 0($sp) #save $ra on stack

loopPrint:

beq $t6, $t7, outLoopPrint

jal outputString #Jump to subroutine that prints result string

jal printRow #Jump to routine that prints result in row

addi $t6, $t6, 1 #Increment branch index

j loopPrint

outLoopPrint:

lw $ra, 0($sp) #Restore $ra from stack

addi $sp, $sp, 4 #Restore stack pointer

jr $ra

outputString:

addi $sp, $sp, -8 #Make room on stack for 2 registers

sw $a0, 0($sp) #save $ra on stack

sw $ra, 4($sp) #save $a0 on stack

li $v0,4

la $a0,outputRow #Print output row string

syscall

li $v0,1

move $a0,$t8 #Print output row number

syscall

li $v0,4

la $a0,colon

syscall

li $v0,4

la $a0,space

syscall

addi $t8, $t8, 1 #Increment output row number

lw $a0, 0($sp) #Restore $a0 from stack

lw $ra, 4($sp) #Restore $ra from stack

addi $sp, $sp, 8 #Restore stack Pointer

jr $ra

#subroutine to print row

printRow:

addi $sp, $sp, -8 #Make room on stack for 2 registers

sw $a0, 0($sp) #save $a0 on stack

sw $ra, 4($sp) #save $ra on stack

loopPrintRow:

beq $t0, $t5, endRow

lw $a0, ($s0)

li $v0, 1

syscall

li $v0,4

la $a0,space

syscall

addi $s0, $s0, 4

addi $t0, $t0, 1

j loopPrintRow

endRow:

addi $t5, $t5, 6

li $v0,4

la $a0,newLine

syscall

lw $a0, 0($sp) #Restore $a0 from stack

lw $ra, 4($sp) #Restore $ra from stack

addi $sp, $sp, 8 #Restore stack Pointer

jr $ra

#End subroutine printRow

#subroutine to initializeVariables used in most subroutines

initializeVariables:

addi $sp, $sp, -4 #Make room on stack for 1 register

sw $ra, 0($sp) #save $ra on stack

li $t0, 0

li $t1, 0

li $t2, 0

li $t3, 0

li $t4, 0

li $t5, 6

li $t6, 0

li $t7, 6

li $t8, 1

lw $ra, 0($sp) #Restore $ra from stack

addi $sp, $sp, 4 #Restore stack Pointer

jr $ra

#End subroutine initializeVariables