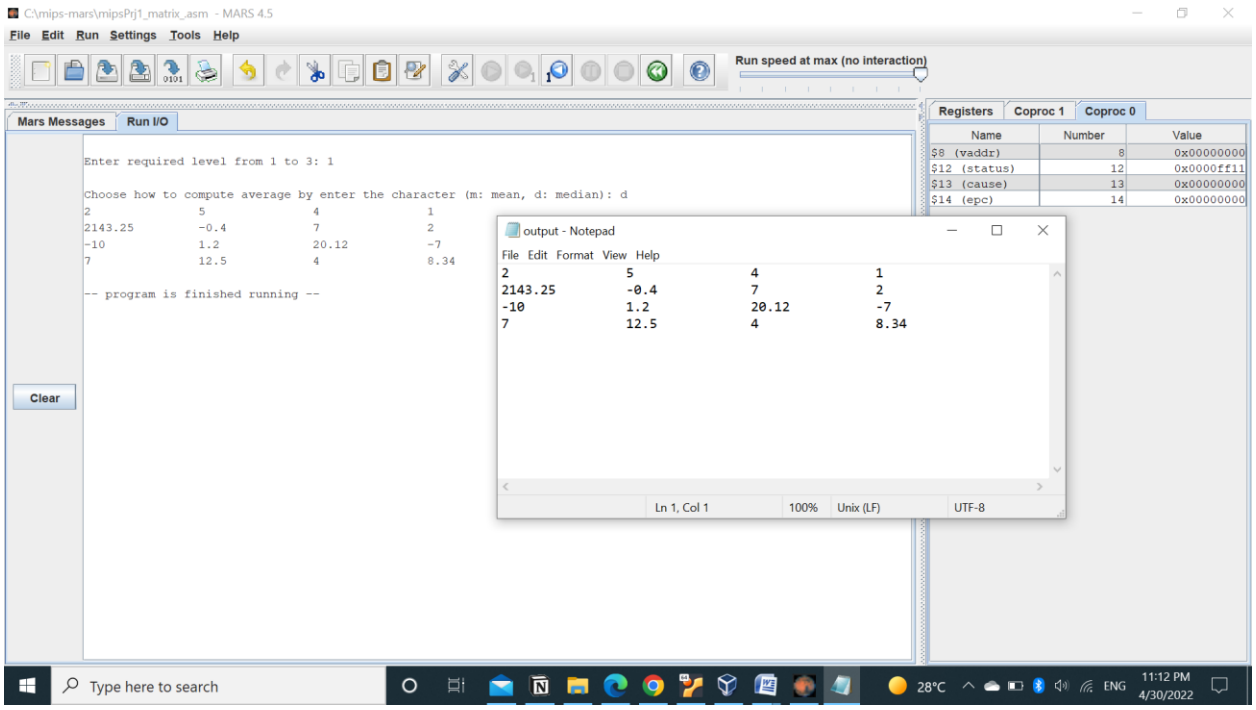
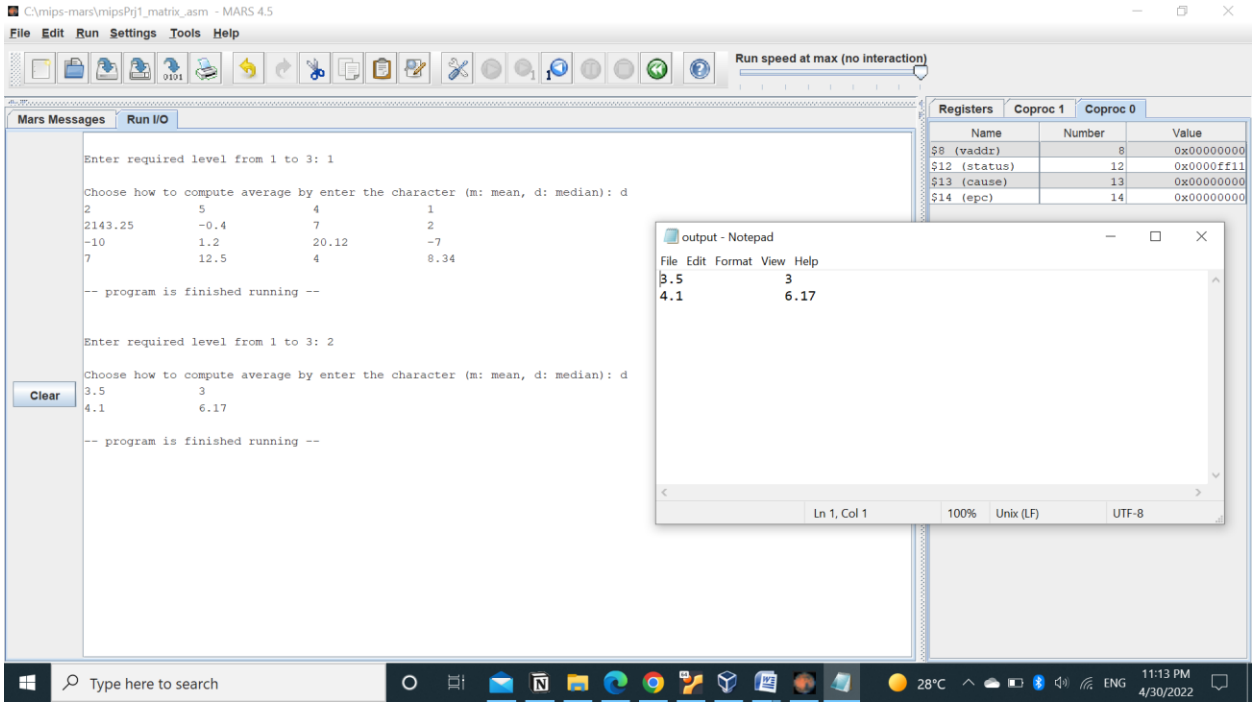
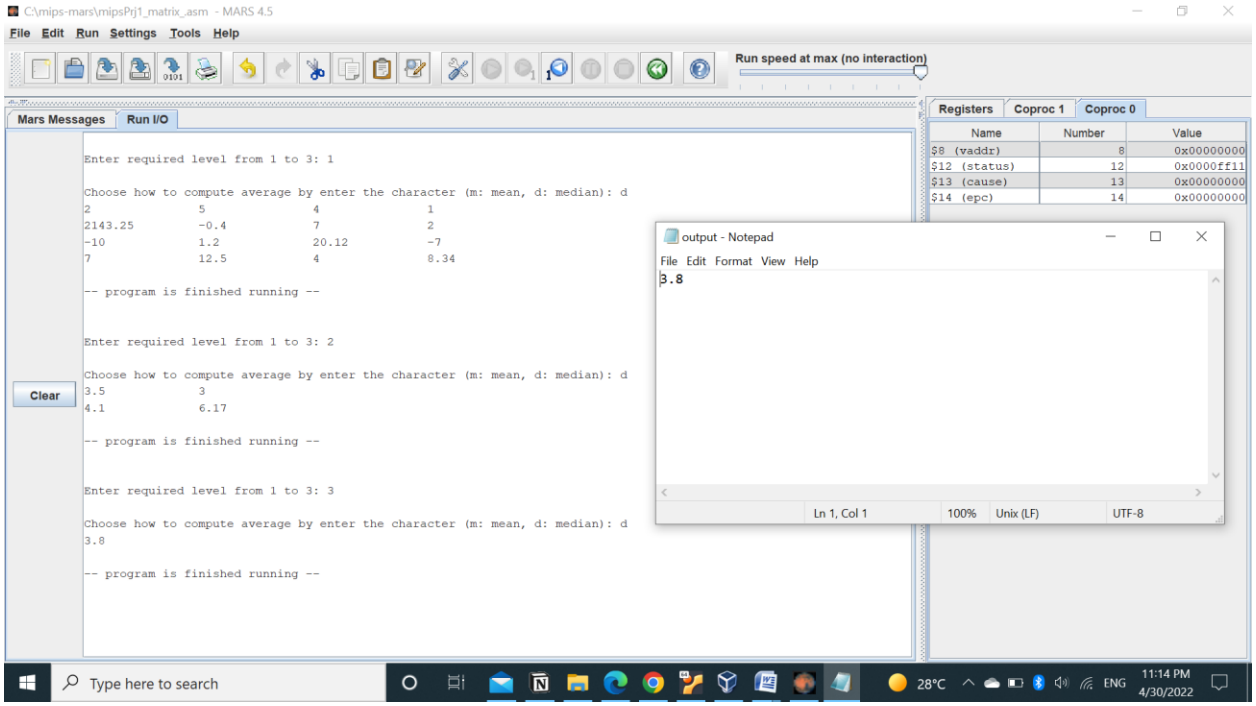
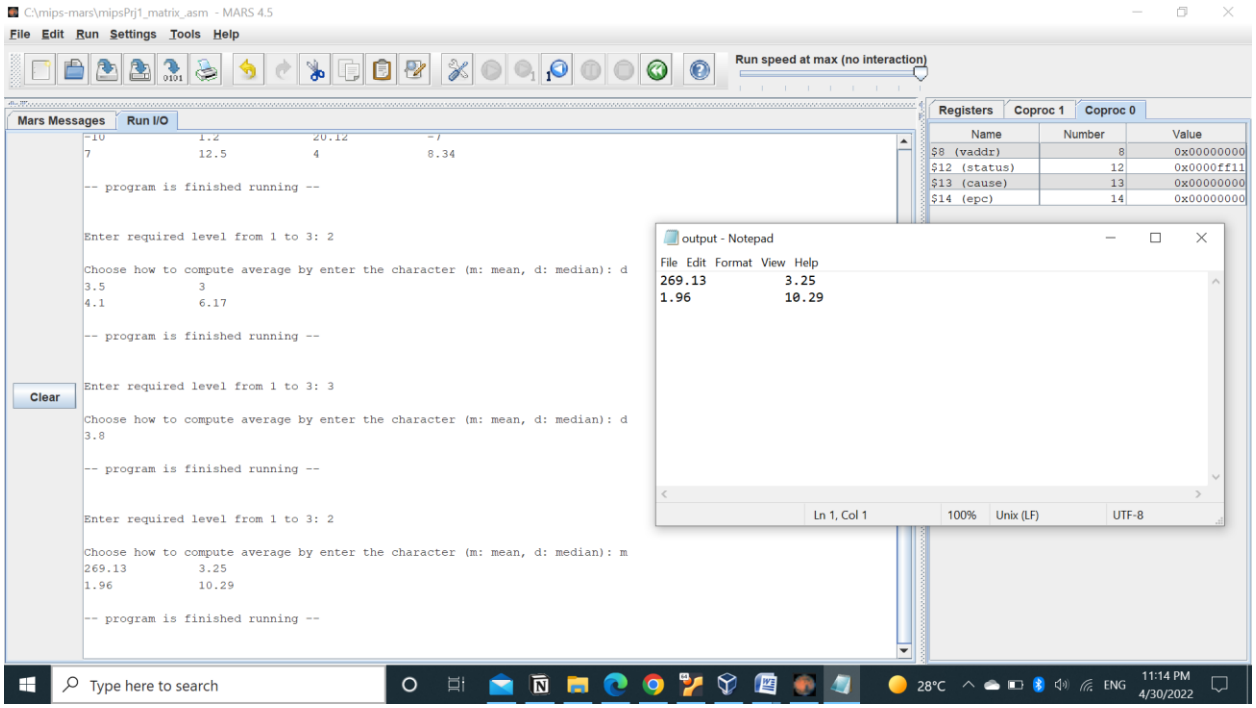
Abdallah Daoud Mohammad 1190515

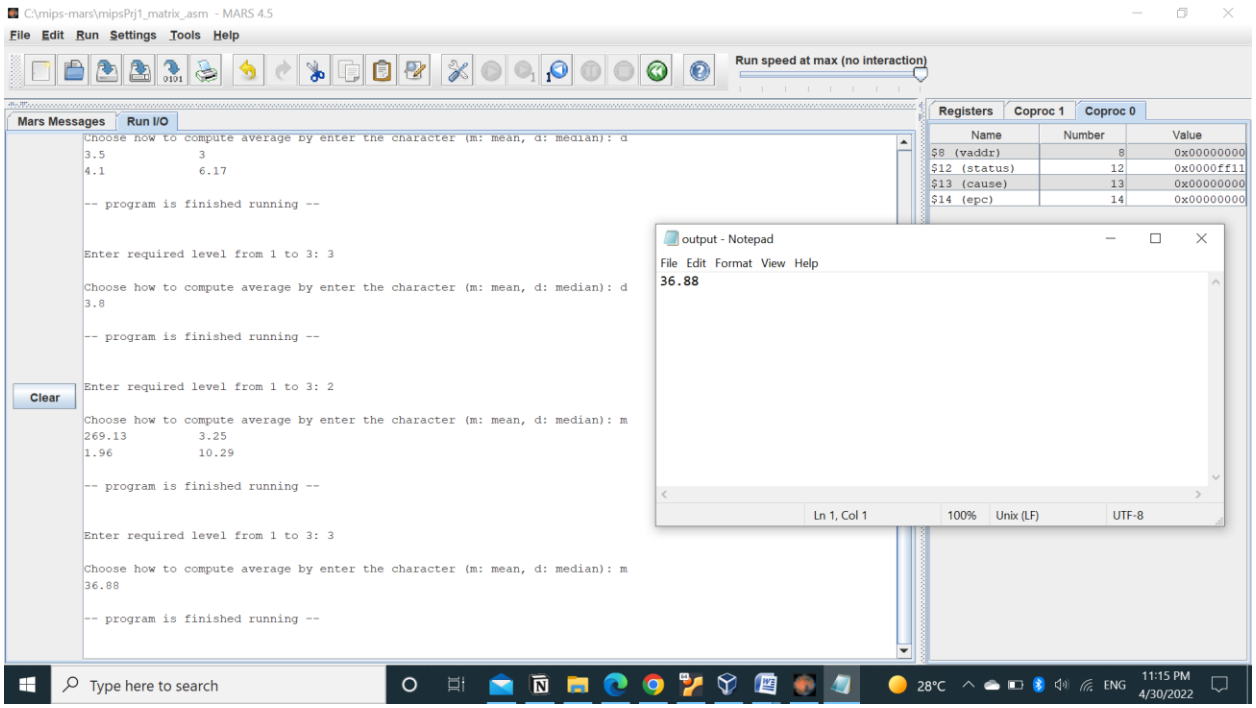
Test cases:-

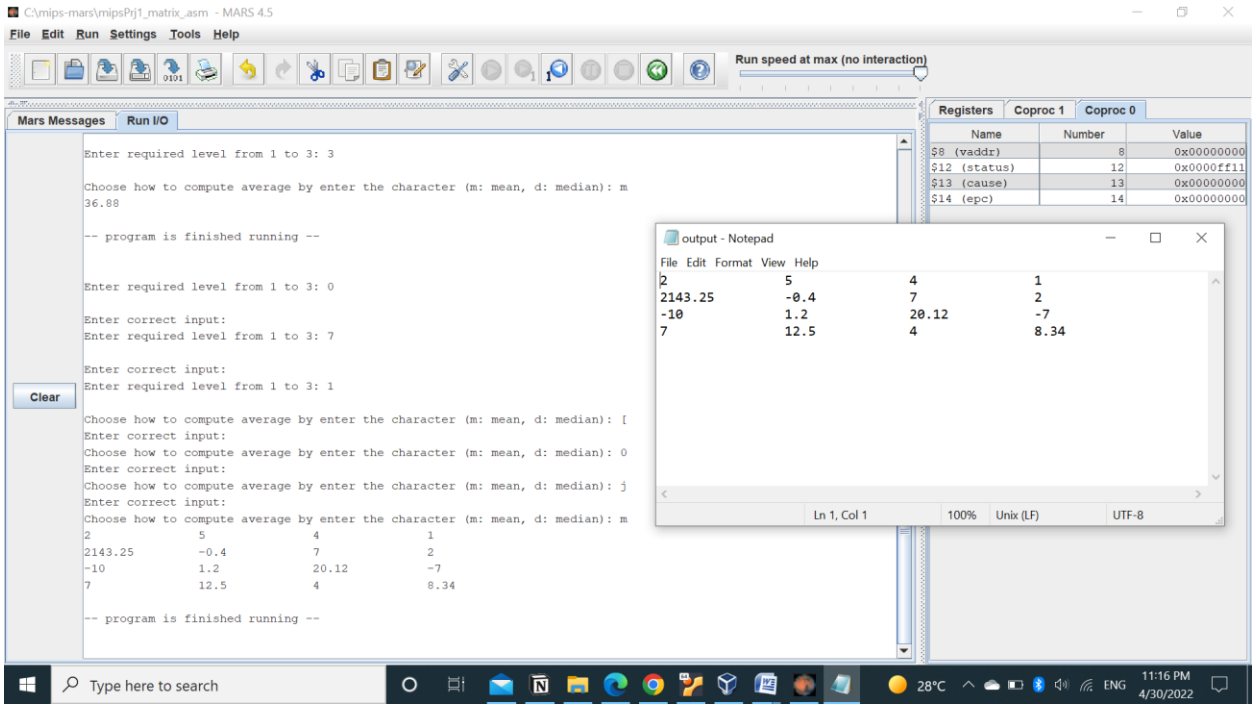





C:\mips-mars\mipsPrj1_matrix_asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Mars Messages
Run I/O
Enter required level from 1 to 3: 1
Choose how to compute average by enter the character (m: mean, d: median): d
2 5 4 1
2143.25 -0.4 7 2
-10 1.2 20.12 -7
7 12.5 4 8.34
-- program is finished running --
Enter required level from 1 to 3: 2
Choose how to compute average by enter the character (m: mean, d: median): d
3.5 3
4.1 6.17
-- program is finished running --
Enter required level from 1 to 3: 3
Choose how to compute average by enter the character (m: mean, d: median): d
3.8
-- program is finished running --
Clear
Registers
Coproc 1
Coproc 0
Name Number Value
$8 (vaddr) 8 0x00000000
$12 (status) 12 0x0000ff11
$13 (cause) 13 0x00000000
$14 (epc) 14 0x00000000
output - Notepad
File Edit Format View Help
3.8
Ln 1, Col 1
100%
Unix (LF)
UTF-8
Type here to search
28°C
ENG
11:14 PM
4/30/2022


C:\mips-mars\mipsPrj1_matrix_asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Mars Messages
Run I/O
-10 1.2 20.12 -7
7 12.5 4 8.34
-- program is finished running --
Enter required level from 1 to 3: 2
Choose how to compute average by enter the character (m: mean, d: median): d
3.5 3
4.1 6.17
-- program is finished running --
Enter required level from 1 to 3: 3
Choose how to compute average by enter the character (m: mean, d: median): d
3.8
-- program is finished running --
Enter required level from 1 to 3: 2
Choose how to compute average by enter the character (m: mean, d: median): m
269.13 3.25
1.96 10.29
-- program is finished running --
Clear
Registers
Coproc 1
Coproc 0
Name Number Value
$8 (vaddr) 8 0x00000000
$12 (status) 12 0x0000ff11
$13 (cause) 13 0x00000000
$14 (epc) 14 0x00000000
output - Notepad
File Edit Format View Help
269.13 3.25
1.96 10.29
Ln 1, Col 1
100%
Unix (LF)
UTF-8
Type here to search
28°C
ENG
11:14 PM
4/30/2022


C:\mips-mars\mipsPrj1_matrix_asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Mars Messages
Run I/O
Registers
Coproc 1
Coproc 0
Name Number Value
$8 (vaddr) 8 0x00000000
$12 (status) 12 0x0000ff11
$13 (cause) 13 0x00000000
$14 (epc) 14 0x00000000
Choose how to compute average by enter the character (m: mean, d: median): d
3.5 3
4.1 6.17
-- program is finished running --
Enter required level from 1 to 3: 3
Choose how to compute average by enter the character (m: mean, d: median): d
3.8
-- program is finished running --
Clear
Enter required level from 1 to 3: 2
Choose how to compute average by enter the character (m: mean, d: median): m
269.13 3.25
1.96 10.29
-- program is finished running --
Enter required level from 1 to 3: 3
Choose how to compute average by enter the character (m: mean, d: median): m
36.88
-- program is finished running --
output - Notepad
File Edit Format View Help
36.88
Ln 1, Col 1
100%
Unix (LF)
UTF-8
Type here to search
28°C
ENG
11:15 PM
4/30/2022


C:\mips-mars\mipsPrj1_matrix_asm - MARS 4.5
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Mars Messages
Run I/O
Registers
Coproc 1
Coproc 0
Name Number Value
$8 (vaddr) 8 0x00000000
$12 (status) 12 0x0000ff11
$13 (cause) 13 0x00000000
$14 (epc) 14 0x00000000
Enter required level from 1 to 3: 3
Choose how to compute average by enter the character (m: mean, d: median): m
36.88
-- program is finished running --
Enter required level from 1 to 3: 0
Enter correct input:
Enter required level from 1 to 3: 7
Enter correct input:
Enter required level from 1 to 3: 1
Clear
Choose how to compute average by enter the character (m: mean, d: median): [
Enter correct input:
Choose how to compute average by enter the character (m: mean, d: median): 0
Enter correct input:
Choose how to compute average by enter the character (m: mean, d: median): j
Enter correct input:
Choose how to compute average by enter the character (m: mean, d: median): m
2 5 4 1
2143.25 -0.4 7 2
-10 1.2 20.12 -7
7 12.5 4 8.34
-- program is finished running --
output - Notepad
File Edit Format View Help
2 5 4 1
2143.25 -0.4 7 2
-10 1.2 20.12 -7
7 12.5 4 8.34
Ln 1, Col 1
100%
Unix (LF)
UTF-8
Type here to search
28°C
ENG
11:16 PM
4/30/2022

Code:-

```
# Title: Matrix down sampling                    # File Name: mipsPrj1_matrix_.asm

# Author: Abdallah Mohammad 1190515      # Date: 2022/4/1

# Description: Computer Architecture first project (Matrix down sampling project)

# Input: “input” file includ a matrix, also get from user the required level and average computing method

# Output: Print in “output” file The computed matrix of required level if exists

################## Data segment ####################

.data

getMethod:    .asciiz        "\nChoose how to compute average by enter the character (m: mean, d: median): "

getLevel1:      .asciiz        "\nEnter required level from 1 to "

getLevel2:      .asciiz        ": "

correctInput:   .asciiz        "\nEnter correct input: "

inputFile:      .asciiz        "input"

outputFile:     .asciiz        "output"

dataFromFile:  .byte          '\0':3000

dataToFile:     .byte          '\0':3000

matrix:         .float         0:3000 # array with 4096 float elements with 0 value, note: matrix is nxn, length=4^m

                                      # I expect that there no need for more than 4096 number in the matrix

matrixErrorMessage:  .asciiz        "\nmatrix isn't nxn and 4^m"

convertNum:   .word          0

space:          .asciiz        " "

tab:            .asciiz        "\t"
```

```
newLine:        .asciiz         "\n"
dot:            .asciiz         "."
windows:        .float          1.5, .5, .5, 1.5
four:           .float          4.0
tow:            .float          2.0
zero:           .float          0.0
minus:          .float          -1.0
hundred:        .float          100.0
################## Code segment ######################
.text
.globl main
main:                   # main program entry

 ### read data from the file
 li $v0, 13 ## open file
 la $a0, inputFile
 li $a1, 0 # 0 = read-only
 li $a2, 0 # mode is ignored
 syscall
 move $t0, $v0 # save the file descriptor temporary
 li $v0, 14 ## write to file
 move $a0, $t0
 la $a1, dataFromFile # $a1 = the address for the data from the file as a string
 li $a2, 30000   # I expect that there no need for more than 30000 char in the file
```

```
syscall
move $a0, $t0
li $v0, 16 ## close the file
syscall

### stor input numbers from the file to the matrix mul $t4,                ($t1)
la $s0, matrix # pointes to the matrix (first element)
la $t0, matrix # pointes to matrix elements (any element)
la $t2, convertNum # $t2 = the address of the int number to convert into flout
lb $t3, 0($a1) # load first char in the string
lwc1 $f4, minus
li $t5, 1 # indicates the sign of the number
strToFloat:
  li $t6, 0 # use as a flag to know if we read integer part or fractional part
  li $t7, 1 # to get (fractional part) / 10^(number of digit in the part)
  ##removeNondigit
    beq $t3, 0, finishMatrix
    bne $t3, '-', isDigit
    li $t5, 0
    isDigit:
      blt $t3, '0', nextChar
      bgt $t3, '9', nextChar
  charToDigit:
  li $t4, 0 # reset
```

```
charToDigitLoop:
  addiu $t3, $t3, -48 # convert character to digit
  mul $t4, $t4, 10 #$t1 # $t4 = sum * 10
  addu $t4, $t4, $t3 # $t4 = sum * 10 + digit
  addiu $a1, $a1, 1 # $a1 = address of next char in the string got from the file
  lb $t3, 0($a1) # load $t3 = str[i]
  blt $t3, '0', notDigit # exit loop if the char isn't a digit, note: ' ' = 32 so this line before
  bgt $t3, '9', notDigit
  j charToDigitLoop # loop back to get the next digit
  notDigit:
  ##convert int to float
  beq $t6, 0, integerOrFractionalPart
    ##mov.s $f2, $f4
    mtc1 $t4, $f2
    cvt.s.w $f2, $f2
    calculateFractionalPart:
      mul $t7, $t7, 10
      addiu $t8, $t8, 1
    blt $t8, $a1, calculateFractionalPart
    mtc1 $t7, $f3
    cvt.s.w $f3, $f3
    div.s $f2, $f2, $f3 # (fractional part) / 10^(number of digit in the part)
    add.s $f1, $f1, $f2 # finally calculat the number
    j storeInMatrix
```

```
    integerOrFractionalPart:
      mtc1 $t4, $f1
      cvt.s.w $f1, $f1
    bne $t3, '.', storeInMatrix # check if dot
      li $t6, 1
      addiu $t8, $a1, 1 # to get number of digits in fractional part
      addiu $a1, $a1, 1 # $a1 = address of next char in the string got from the file
      lb $t3, 0($a1) # load $t3 = str[i]
      j charToDigit
    storeInMatrix:
    beq $t5, 1, isMinus
      mul.s $f1, $f1, $f4
      li $t5, 1
    isMinus:


    swc1 $f1, 0($t0) # store the number in the matrix
    addiu $t0, $t0, 4 # $t0 = address of next element in the matrix (float = 4 bytes)
    nextChar:
      addiu $a1, $a1, 1 # $a1 = address of next char in the string got from the file
      lb $t3, 0($a1) # load $t3 = str[i]
    j strToFloat # loop back to get the next number
finishMatrix:
move $s2, $t0 # save the value of last element in the matrix
```

```
### to see if the matrix are avalible and get the last level can reach ($t0)
subu $t2, $s2, $s0 # get the the number of elements in the matrix*4 where (float = 4 bytes)
div $s5, $t2, 16 # get the length of the matrix side
mflo $s5 # the lo register contains the quotient, and hi contains the remainder
li $t5, 4 # useed to divide on 4
li $t4, 0 # initialize $t4 to get remainder values
li $t1, 0 # counter for number of levels that can be obtained
loop:
  addiu $t1, $t1, 1 # level number + 1
  div $t2, $t5 # the lo register contains the quotient, and hi contains the remainder
  move $t4, $t2
  mflo $t2 # $t2 = quotient
  mfhi $t4 # $t4 = remainder
  bgt $t4, $zero, matrixError
  bne $t2, 1, loop
  j matrixFine
matrixError:
  la $a0, matrixErrorMessage
  li $v0, 4
  syscall
  li $v0, 10 # exit
  syscall
matrixFine:
```

```
### get the number of level that user want to get the valuse in from 1 to last level can reach
getCorrect1:
  la $a0, getLevel1 # tell user to enter the input
  li $v0, 4
  syscall
  move $a0, $t1
  li $v0, 1
  syscall
  la $a0, getLevel2
  li $v0, 4
  syscall
  #la $a0, getLevel1 # tell user to enter the input
  li $v0, 5 # Read integer  ############ change to read string then convert to read int to prevent exiption-error
  syscall
  bgt $v0, $t1, toExit
  bge $v0, 1, exitGetCorrect1
  toExit:
  la $a0, correctInput # print error message
  li $v0, 4
  syscall
  j getCorrect1
exitGetCorrect1:
```

```
move $s1, $v0 # save the value of the required level


### get the method that the user whant to use to compute the average
getCorrect2:
  la $a0, getMethod # tell user to enter the input
  li $v0, 4
  syscall
  li $v0, 12 # Read char
  syscall
  beq $v0, 'm', exitGetCorrect2 # check if the input is correct
  beq $v0, 'd', exitGetCorrect2
  la $a0, correctInput # print error message
  li $v0, 4
  syscall
  j getCorrect2
exitGetCorrect2:
move $s3, $v0 # save the value of the input 'm' or 'd'


################################################################################
############


### comput the mean or meadean
## calculat the pointers for first four elements in the matrix
li $t6, 1 # comput matrix side length
```

```
powTo2:
  beq $t1, 1, exitPow
  mul $t6, $t6, 2
  addiu $t1, $t1, -1
  j powTo2
exitPow: # $t6 = matrix side length
mul $t6, $t6, 4 # $t6 = matrix side length * (elements size = 4 byte)
li $t3, 0 # use to know if we finish reading these tow rows
move $t4, $s0 # write to
#li $t5, 2
lwc1 $f10, four # use to divide by 4.0
lwc1 $f11, tow # for median
lwc1 $f12, zero # to reset som values

move $t0, $s0 # read from these tow rows, $t0+4 and $t2+4 pointes to another tow elements
addu $t2, $t6, $s0
move $t4, $s0 # write to
subu $t7, $s2, $s0 # get the the number of elements in the matrix*4 where (float = 4 bytes)

beq $s3, 'd', comput # lode window if method is mean
  lwc1 $f6, windows
  lwc1 $f7, windows+4
  lwc1 $f8, windows+8
  lwc1 $f9, windows+12
```

```
comput:
 beq $s1, 1, finish

 ## load four elements
 lwc1 $f2, 0($t0)
 lwc1 $f3, 4($t0)
 lwc1 $f4, 0($t2)
 lwc1 $f5, 4($t2)

 beq $s3, 'm', isMean # comput the require method
  jal median
  j isNotMean
 isMean:
  jal mean
 isNotMean:

 swc1 $f1, ($t4) # stor
 addiu $t4, $t4, 4 # next element to stor in array
 addiu $t0, $t0, 8 # next four elements in array to reade
 addiu $t2, $t2, 8
 addiu $t3, $t3, 8 # increment the counter to see if we have to read from next tow rows
 blt $t3, $t6, nextTwoElements
  addu $t0, $t0, $t6 # go to next rows
```

```
  addu $t2, $t2, $t6
  li $t3, 0 # reset the counter
nextTwoElements:
blt $t2, $s2, comput

beq $s3, 'd', isMedian
  mov.s $f13, $f6
  mov.s $f6, $f7
  mov.s $f7, $f13
  mov.s $f13, $f8
  mov.s $f8, $f9
  mov.s $f9, $f13
isMedian:

div $t6, $t6, 2 # get side length of the computed level
mflo $t6
addiu $s1, $s1, -1 # increment level number to know how many times still need to calculate required level
div $t7, $t7, 4 # get the the number of elements in the matrix*4 where (float = 4 bytes)
mflo $t7
addu $s2, $s0, $t7 # calculat the pointer to last element

move $t0, $s0 # read from these tow rows, $t0+4 and $t2+4 pointes to another tow elements
addu $t2, $t6, $s0
```

```
  move $t4, $s0 # write to

  j comput

finish:

### convert from flout to string
la $a0, newLine # $a0 = value to print
li $v0, 4
syscall
lwc1 $f10, hundred
la $a0, tab # $a0 = value to print
la $t8, dataToFile
addiu $t8, $t8, 12
li $s4, '\t'
li $s5, '-'
li $s6, '.'
li $s7, '\n'

floatToString:
  ## get integer and fractional parts
  lwc1 $f1, 0($t0) # load the flout number
  mul.s $f1, $f1, $f10 # then mul with 100
  cvt.w.s $f1, $f1 # convert to int
```

```
mfc1 $t1, $f1
# write '-' if minus then deal with as possitive
bge $t1, 0, isNegative
  sb $s5, -12($t8)
  abs $t1, $t1
  addiu $t8, $t8, 1
isNegative:
div $t1, $t1, 100
move $t2, $t1
mfhi $t1 # fractition part
mflo $t2 # integer part

move $t7, $t8 #

## get fractition part
getFractitionPart:
  div $t1, $t1, 10
  move $t5, $t1
  mflo $t1
  mfhi $t5
  bne $t8, $t7, write
  beq $t5, 0, doNotWrite
  write:
```

```
    addiu $t5, $t5, 48
    sb $t5, ($t7)
    addiu $t7, $t7, -1
  doNotWrite:
bne $t1, 0, getFractitionPart
beq $t8, $t7, isInteger
  sb $s6, ($t7) # write dot
  addiu $t7, $t7, -1
isInteger:
## get integer part
getIntegerPart:
  div $t2, $t2, 10
  move $t5, $t2
  mflo $t2
  mfhi $t5
  addiu $t5, $t5, 48
  sb $t5, ($t7)
  addiu $t7, $t7, -1
bne $t2, 0, getIntegerPart

#addiu $t8, $t8, 12
move $t9, $t8
align:
  lb $t5, 1($t7)
```

```
    sb $t5, -12($t8)
    addiu $t7, $t7, 1
    addiu $t8, $t8, 1
  blt $t7, $t9, align

  addiu $t0, $t0, 4
  addiu $t3, $t3, 4 # increment the counter to see if we have to read from next row
  blt $t3, $t6, nextElement
    sb $s7, -12($t8)
    li $t3, 0 # reset the counter
    j nextRow
  nextElement:
    sb $s4, -12($t8)
    addiu $t8, $t8, 1
    sb $s4, -12($t8)
  nextRow:
  addiu $t8, $t8, 1
  blt $t0, $s2, floatToString

addiu $t8, $t8, -12
li $a0, '\0'
sb $a0, ($t8)

la $t9, dataToFile
```

```
subu $t7, $t8, $t9 # $a0 = length of output string


la $a0, dataToFile #############
li $v0, 4
syscall


### write data to the file
li $v0, 13 ## open file
la $a0, outputFile
li $a1, 1 # 1 = write-only
li $a2, 0 # mode is ignored
syscall
move $t0, $v0 # save the file descriptor temporary
li $v0, 15 ## write to file
move $a0, $t0
#move $a1, $s0 # $a1 = the address of the matrix
la $a1, dataToFile
move $a2, $t7 # $a0 = length of output string
syscall
li $v0, 16 ## close the file
move $a0, $t0
syscall
```

```
li $v0, 10                  # exit program
syscall



mean:
   mul.s $f2, $f2, $f6 # mul
   mul.s $f3, $f3, $f7
   mul.s $f4, $f4, $f8
   mul.s $f5, $f5, $f9
   add.s $f2, $f2, $f3 # add
   add.s $f4, $f4, $f5
   add.s $f1, $f12, $f12 # reset $f1
   add.s $f1, $f2, $f4
   div.s $f1, $f1, $f10 # sum/4
   jr $ra # return

 median:
 ## sort
 li $t1, 0 # flag or couter to know what is the number of compare and swape we do
 sort:
  c.lt.s $f2, $f3
  bc1t compareAndSwap1
    mov.s $f1, $f2
```

```
  mov.s $f2, $f3
  mov.s $f3, $f1
 compareAndSwap1:
 c.lt.s $f4, $f5
 bc1t compareAndSwap2
  mov.s $f1, $f4
  mov.s $f4, $f5
  mov.s $f5, $f1
 compareAndSwap2:
 bne $t1, $zero, endSort
 li $t1, 1
 c.lt.s $f3, $f4
 bc1t compareAndSwap3
  mov.s $f1, $f3
  mov.s $f3, $f4
  mov.s $f4, $f1
 compareAndSwap3:
 j sort
endSort:
add.s $f1, $f3, $f4
div.s $f1, $f1, $f11 # (midile elements sum)/2
jr $ra # return
```