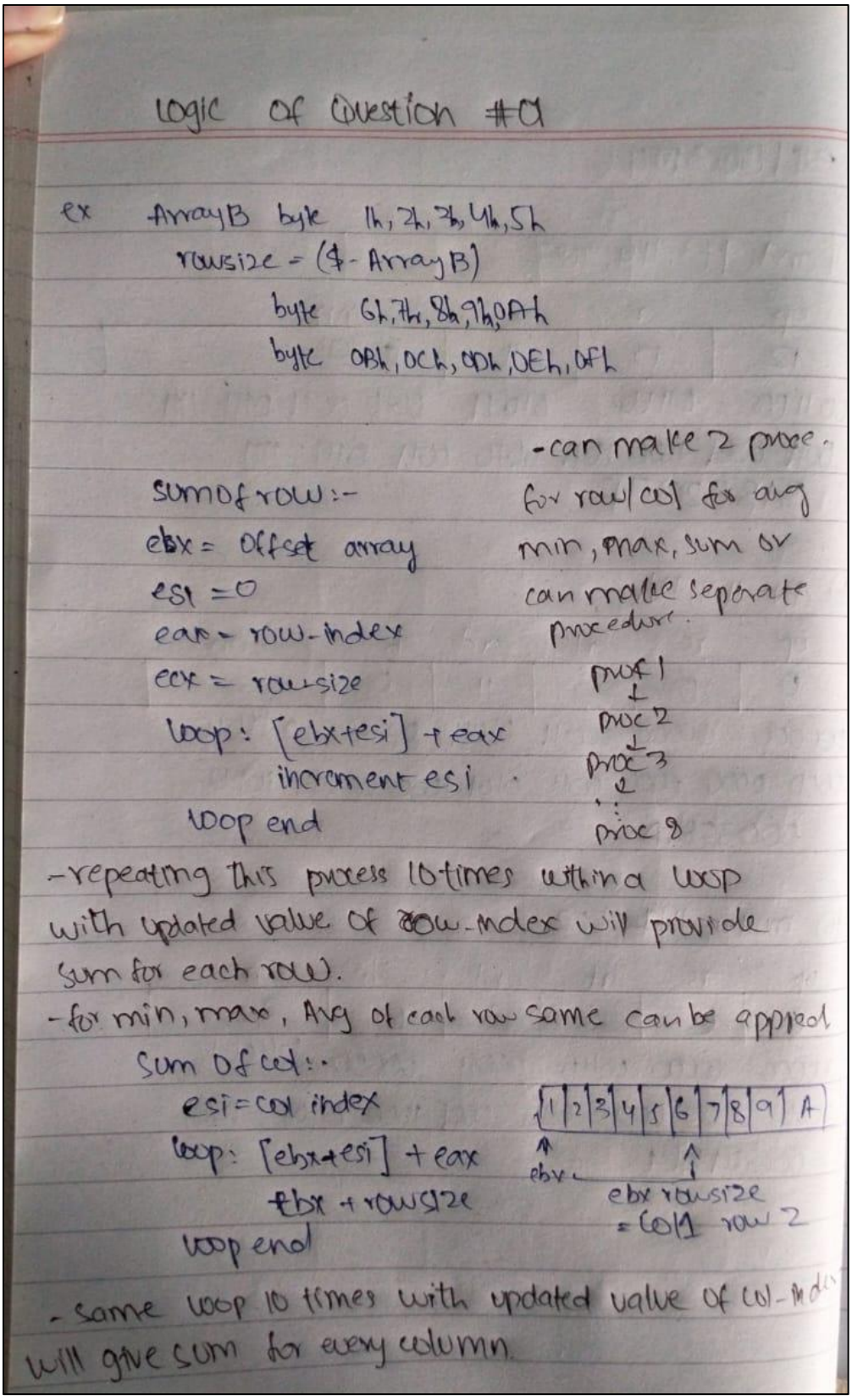
COAL ASSIGNMENT # 04

TASK# 01:

Logic:



Logic of Question #01

ex ArrayB byte 1h, 2h, 3h, 4h, 5h
rowsize = ($ - ArrayB)
byte 6h, 7h, 8h, 9h, 0Ah
byte 0Bh, 0Ch, 0Dh, 0Eh, 0Fh

sumofrow :-
ebx = Offset array
esi = 0
eax = row-index
ecx = rowsize
loop: [ebx+esi] + eax
increment esi
loop end

- can make 2 proce. for row/col for avg min, max, sum or can make separate procedur.

proc 1
proc 2
proc 3
proc 8

- repeating this process 10 times within a loop with updated value of row-index will provide sum for each row.
- for min, max, Avg of each row same can be applied

Sum of col :-
esi = col index
loop: [ebx+esi] + eax
ebx + rowsize
loop end

1	2	3	4	5	6	7	8	9	A

ebx
ebx rowsize = col 1 row 2

- same loop 10 times with updated value of col-index will give sum for every column.

Code:

```
TITLE Assignment4 (Test.asm)
Include irvine32.inc

.data
tableB byte 100 dup(?)
RowSize byte 10
RowIndex byte ?
ColIndex byte ?
counter dword ?

sum dword ?
TotalSum dword ?
sumColumn dword ?
maximumRow dword ?
maxTable dword 0
maximumColumn dword ?
minimumRow dword ?
minTable dword 100
minimumCol dword ?
averageRow dword ?
averageCol dword ?
averageTable dword ?

message byte "Row# ",0
message2 byte "Sum of Overall Table: ",0
colon byte ": ",0

message3 byte "Col# ",0
message5 byte "Maximum of Overall Table : ",0
message7 byte "Minimum of Overall Table : ",0
message9 byte "The Average of Overall Table : ",0

MessageSumRow byte "THE SUM OF EACH ROW!",0
MessageSumCol byte "THE SUM OF EACH COLUMN!",0
MessageMaxRow byte "THE MAXIUMUM OF EACH ROW!",0
MessageMaxCol byte "THE MAXIUMUM OF EACH COLUMN!",0
MessageMinRow byte "THE MINIMUM OF EACH ROW!",0
MessageMinCol byte "THE MINIMUM OF EACH COLUMN!",0
MessageAvgRow byte "THE AVERAGE OF EACH ROW!",0
MessageAvgCol byte "THE AVERAGE OF EACH COLUMN!",0

;prototypes of all the functions
;Note: made different procedures for them for easy understanding

sumRow proto array: ptr byte, rSize: byte, rIndex: byte
sumCol proto array: ptr byte, rSize: byte, cIndex: byte
MaxRow proto array: ptr byte, rSize: byte, rIndex: byte
MaxCol proto array: ptr byte, rSize: byte, cIndex: byte
```

```
MinRow proto array: ptr byte, rSize: byte, rIndex: byte
MinCol proto array: ptr byte, rSize: byte, cIndex: byte
AvgRow proto array: ptr byte, rSize: byte, rIndex: byte
AvgCol proto array: ptr byte, rSize: byte, cIndex: byte

.code
main PROC

; initializing array from 1-100
mov esi, offset tableB
mov eax, 1
mov ecx, 100
l1: mov [esi], al
        inc al
        inc esi
loop l1

mov RowIndex, 0
mov ecx, 10


; sum of rows
mov edx, offset MessageSumRow
call WriteString
call crlf

l2: mov counter, ecx
        mov edx, offset message
        call WriteString
        movzx eax, RowIndex
        call WriteDec
        mov edx, offset colon
        call WriteString
        invoke sumRow, addr tableB, RowSize, RowIndex

        mov eax, sum
        add TotalSum, eax               ; calculating the sum of overall table
        inc RowIndex
        mov ecx, counter
loop l2


; sum of columns
call crlf
call crlf
mov edx, offset MessageSumCol
call WriteString
call crlf

mov ecx, 10
mov ColIndex, 0
```

```
l3: mov counter, ecx
        mov edx, offset message3
        call WriteString
        movzx eax, ColIndex
        call WriteDec
        mov edx, offset colon
        call WriteString
        invoke sumCol, addr tableB, RowSize, ColIndex
        mov ecx, counter
        inc ColIndex
        call crlf
loop l3


; maximum of Rows
call crlf
call crlf
Mov edx, offset MessageMaxRow
call writeString
call crlf

mov RowIndex, 0
mov ecx, 10
l4: mov counter, ecx
        mov edx, offset message
        call WriteString
        movzx eax, RowIndex
        call WriteDec
        mov edx, offset colon
        call WriteString

        invoke maxRow, addr tableB, RowSize, RowIndex
        mov eax, maxTable
        cmp maximumRow, eax

        JLE next
        mov eax, maximumRow
        mov maxTable, eax
        next:
                inc RowIndex
                mov ecx, counter
loop l4


; maximum of columns
call crlf
call crlf
mov edx, offset MessageMaxCol
call writeString
call crlf
```

```
mov ecx, 10
mov ColIndex, 0
l5: mov counter, ecx
        mov edx, offset message3
        call WriteString
        movzx eax, ColIndex
        call WriteDec
        mov edx, offset colon
        call WriteString

        invoke maxCol, addr tableB, RowSize, ColIndex
        mov ecx, counter
        inc ColIndex
        call crlf
loop l5



;minimum of row
call crlf
call crlf
Mov edx, offset MessageMinRow
call writeString
call crlf
mov RowIndex, 0
mov ecx, 10
l6: mov counter, ecx
        mov edx, offset message
        call WriteString
        movzx eax, RowIndex
        call WriteDec
        mov edx, offset colon
        call WriteString

        invoke minRow, addr tableB, RowSize, RowIndex
        mov eax, minTable
        cmp minimumRow, eax

        JGE next3
        mov eax, minimumRow
        mov minTable, eax                       ; min of overall table
        next3:
                inc RowIndex
                mov ecx, counter
loop l6


;minimum of columns
call crlf
call crlf
mov edx, offset MessageMinCol
```

```
call writeString
call crlf

mov ecx, 10
mov ColIndex, 0
l7: mov counter, ecx
        mov edx, offset message3
        call WriteString
        movzx eax, ColIndex
        call WriteDec
        mov edx, offset colon
        call WriteString

        invoke MinCol, addr tableB, RowSize, ColIndex

        mov ecx, counter
        inc ColIndex
        call crlf
loop l7


; Average of Rows
call crlf
call crlf
mov edx, offset MessageAvgRow
call WriteString
call crlf

mov RowIndex, 0
mov ecx, 10
l8: mov counter, ecx
        mov edx, offset message
        call WriteString
        movzx eax, RowIndex
        call WriteDec
        mov edx, offset colon
        call WriteString
        invoke AvgRow, addr tableB, RowSize, RowIndex
        inc RowIndex
        mov ecx, counter
loop l8


; Average of Columns
call crlf
call crlf
mov edx, offset MessageAvgCol
call WriteString
call crlf

mov ecx, 10
```

```
mov ColIndex, 0
l9: mov counter, ecx
        mov edx, offset message3
        call WriteString
        movzx eax, ColIndex
        call WriteDec
        mov edx, offset colon
        call WriteString
        invoke AvgCol, addr tableB, RowSize, ColIndex
        mov ecx, counter
        inc ColIndex
loop l9


call crlf
mov edx, offset message2
call WriteString
mov eax, TotalSum
call WriteDec
call crlf

mov edx, offset message9
call WriteString
mov eax, TotalSum
mov edx, 0
mov bx, 100
div bx
mov averageTable, eax
call WriteDec
call crlf


mov edx, offset message7
call WriteString
mov eax, minTable
call WriteDec
call crlf


mov edx, offset message5
call WriteString
mov eax, maxTable
call WriteDec
call crlf



call dumpregs
exit
main ENDP
```

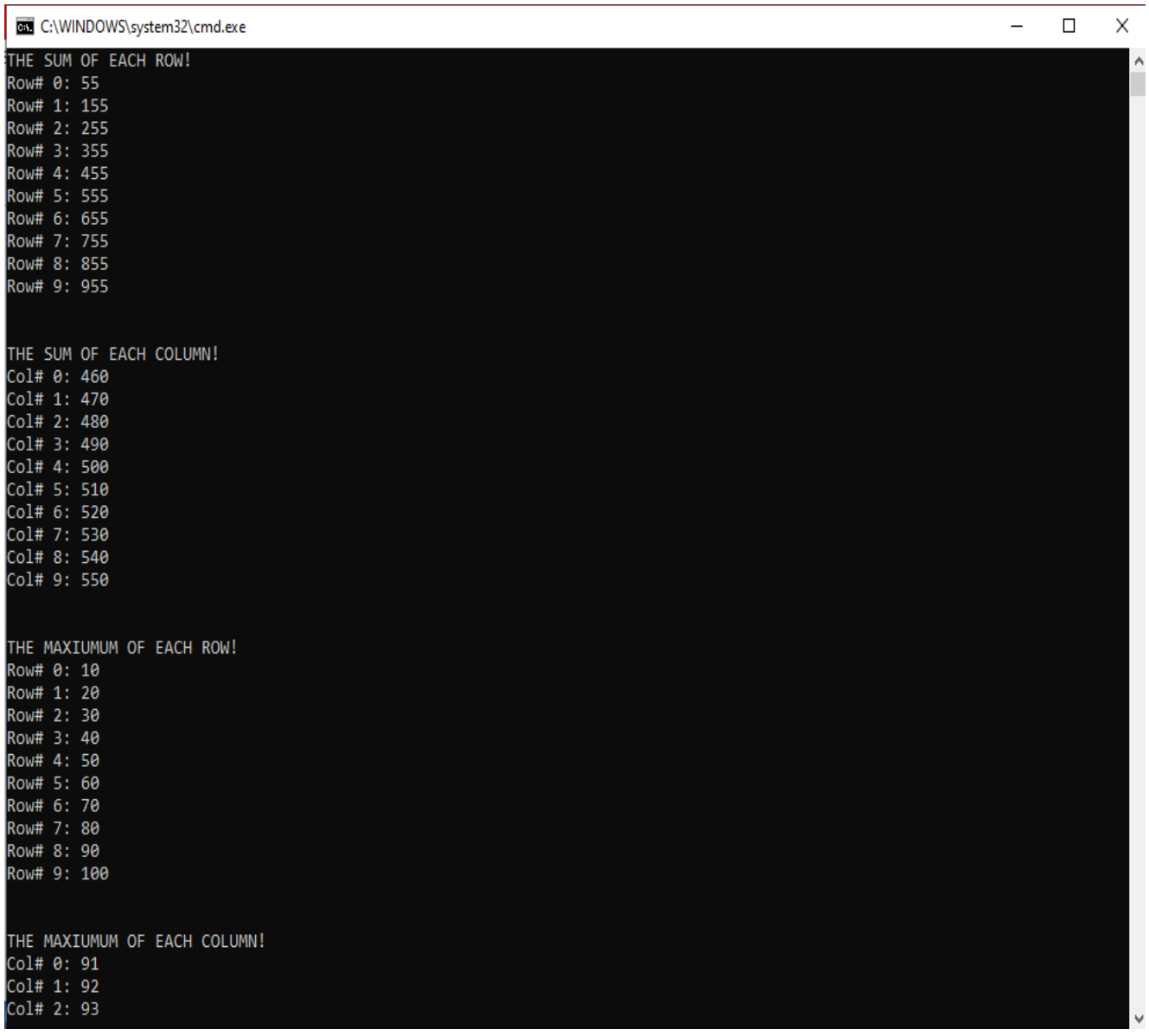
```
sumRow proc, array: ptr byte, rSize: byte, rIndex: byte

mov ebx, array
movzx ecx, rsize
movzx eax, rIndex
mul ecx
add ebx, eax

mov eax, 0
mov esi, 0

l1: movzx edx, byte ptr[ebx+esi]
        add eax, edx
        inc esi
loop l1

mov sum, eax
call WriteDec
call crlf

ret
sumRow endp



sumCol proc, array: ptr byte, rSize: byte, cIndex: byte

mov ebx, array
mov eax, 0
movzx esi, cIndex
movzx edi, rSize

movzx ecx, rsize
l1: movzx edx, byte ptr[ebx+esi]
        add eax, edx
        add ebx, edi
loop l1

mov sumColumn, eax
call WriteDec

ret
sumCol endp


MaxRow proc, array: ptr byte, rSize: byte, rIndex: byte

mov ebx, array
movzx ecx, rsize
movzx eax, rIndex
```

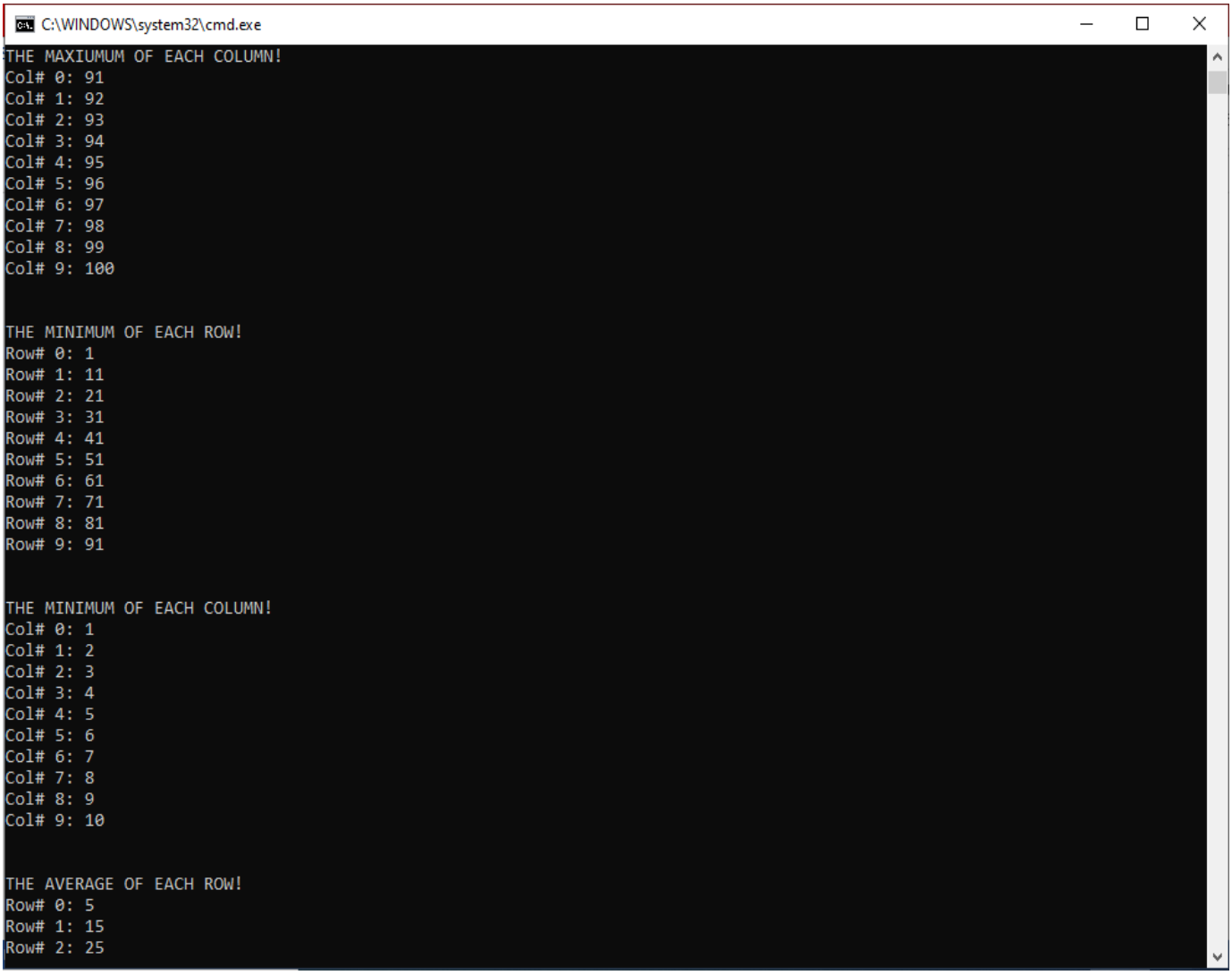
```
mul ecx
add ebx, eax

mov eax, 0
mov al, [ebx]
mov esi, 0

l1: movzx edx, byte ptr[ebx+esi]
        cmp edx, eax
        JLE next
        mov eax, edx
        next:
                inc esi
loop l1

mov maximumRow, eax
call WriteDec
call crlf

ret
MaxRow endp



MaxCol proc, array: ptr byte, rSize: byte, cIndex: byte

mov ebx, array
mov eax, 0
movzx esi, cIndex
movzx edi, rSize

movzx ecx, rsize
l1: movzx edx, byte ptr[ebx+esi]
        cmp edx, eax
        JLE next
        mov eax, edx
        next:
                add ebx, edi
loop l1

mov maximumColumn, eax
call WriteDec

ret
MaxCol endp


MinRow proc, array: ptr byte, rSize: byte, rIndex: byte

mov ebx, array
movzx ecx, rsize
```

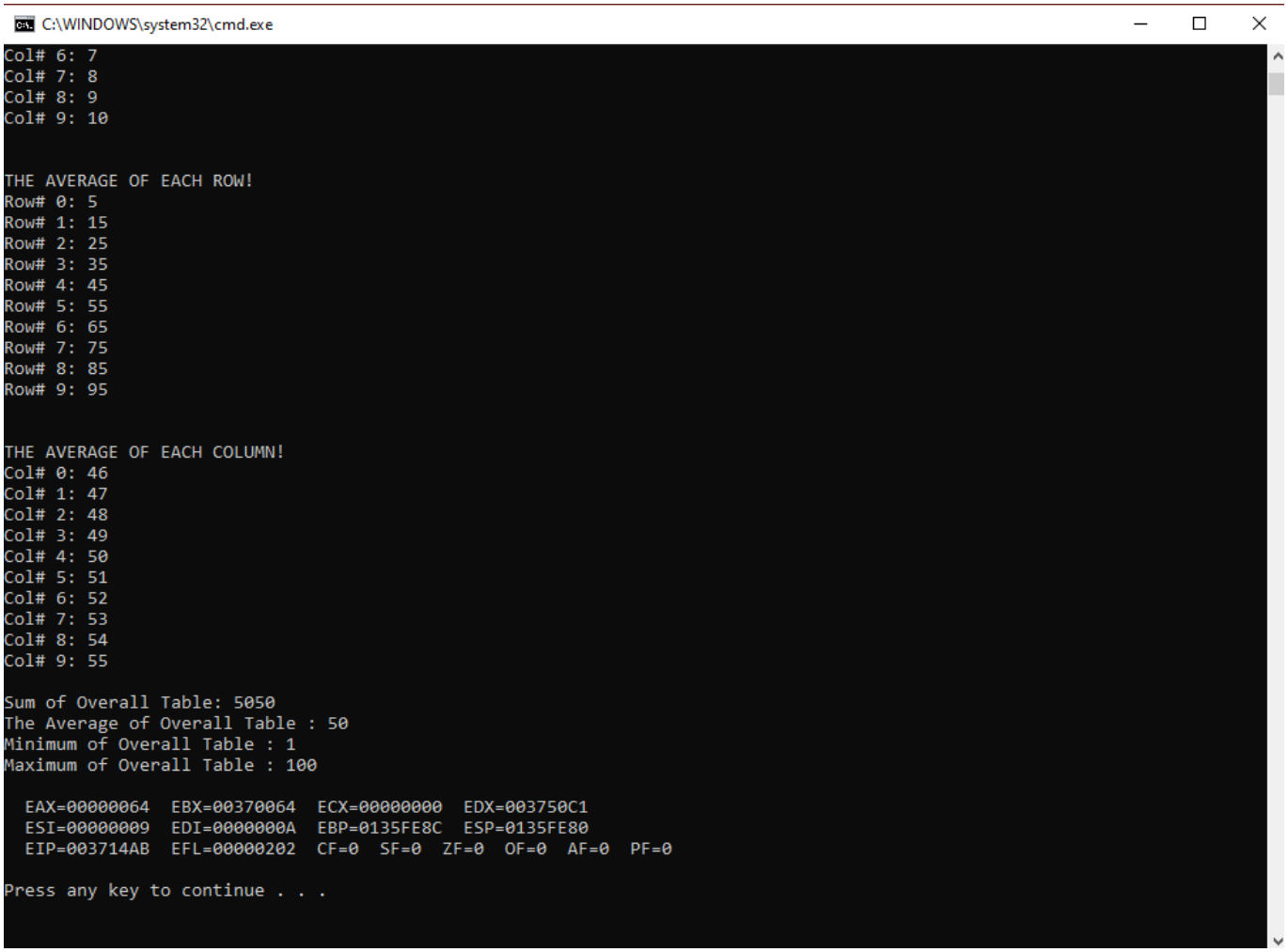
```
movzx eax, rIndex
mul ecx
add ebx, eax

mov eax, 0
mov al, [ebx]
mov esi, 0

l1: movzx edx, byte ptr[ebx+esi]
        cmp edx, eax
        JGE next
        mov eax, edx
        next:
                inc esi
loop l1

mov minimumRow, eax
call WriteDec
call crlf

ret
MinRow endp


MinCol proc, array: ptr byte, rSize: byte, cIndex: byte
mov ebx, array
mov eax, 0ffffffh
movzx esi, cIndex
movzx edi, rSize

movzx ecx, rsize
l1: movzx edx, byte ptr[ebx+esi]
        cmp edx, eax
        JGE next
        mov eax, edx
        next:
                add ebx, edi
loop l1

mov minimumCol, eax
call WriteDec
ret
MinCol endp


AvgRow proc, array: ptr byte, rSize: byte, rIndex: byte

mov ebx, array
movzx ecx, rsize
movzx eax, rIndex
mul ecx
```

```
add ebx, eax

mov eax, 0
mov esi, 0

l1: movzx edx, byte ptr[ebx+esi]
        add eax, edx
        inc esi
loop l1

mov edx, 0
mov bx, 10
div bx
mov averageRow, eax
call WriteDec
call crlf

ret
AvgRow endp


AvgCol proc, array: ptr byte, rSize: byte, cIndex: byte

mov ebx, array
mov eax, 0
movzx esi, cIndex
movzx edi, rSize

movzx ecx, rsize
l1: movzx edx, byte ptr[ebx+esi]
        add eax, edx
        add ebx, edi
loop l1

mov edx, 0
mov bx, 10
div bx
mov averageCol, eax
call WriteDec
call crlf

ret
AvgCol endp
END main
```

Output:


C:\WINDOWS\system32\cmd.exe
THE SUM OF EACH ROW!
Row# 0: 55
Row# 1: 155
Row# 2: 255
Row# 3: 355
Row# 4: 455
Row# 5: 555
Row# 6: 655
Row# 7: 755
Row# 8: 855
Row# 9: 955
THE SUM OF EACH COLUMN!
Col# 0: 460
Col# 1: 470
Col# 2: 480
Col# 3: 490
Col# 4: 500
Col# 5: 510
Col# 6: 520
Col# 7: 530
Col# 8: 540
Col# 9: 550
THE MAXIUMUM OF EACH ROW!
Row# 0: 10
Row# 1: 20
Row# 2: 30
Row# 3: 40
Row# 4: 50
Row# 5: 60
Row# 6: 70
Row# 7: 80
Row# 8: 90
Row# 9: 100
THE MAXIUMUM OF EACH COLUMN!
Col# 0: 91
Col# 1: 92
Col# 2: 93


C:\WINDOWS\system32\cmd.exe
THE MAXIUMUM OF EACH COLUMN!
Col# 0: 91
Col# 1: 92
Col# 2: 93
Col# 3: 94
Col# 4: 95
Col# 5: 96
Col# 6: 97
Col# 7: 98
Col# 8: 99
Col# 9: 100
THE MINIMUM OF EACH ROW!
Row# 0: 1
Row# 1: 11
Row# 2: 21
Row# 3: 31
Row# 4: 41
Row# 5: 51
Row# 6: 61
Row# 7: 71
Row# 8: 81
Row# 9: 91
THE MINIMUM OF EACH COLUMN!
Col# 0: 1
Col# 1: 2
Col# 2: 3
Col# 3: 4
Col# 4: 5
Col# 5: 6
Col# 6: 7
Col# 7: 8
Col# 8: 9
Col# 9: 10
THE AVERAGE OF EACH ROW!
Row# 0: 5
Row# 1: 15
Row# 2: 25


C:\WINDOWS\system32\cmd.exe
Col# 6: 7
Col# 7: 8
Col# 8: 9
Col# 9: 10
THE AVERAGE OF EACH ROW!
Row# 0: 5
Row# 1: 15
Row# 2: 25
Row# 3: 35
Row# 4: 45
Row# 5: 55
Row# 6: 65
Row# 7: 75
Row# 8: 85
Row# 9: 95
THE AVERAGE OF EACH COLUMN!
Col# 0: 46
Col# 1: 47
Col# 2: 48
Col# 3: 49
Col# 4: 50
Col# 5: 51
Col# 6: 52
Col# 7: 53
Col# 8: 54
Col# 9: 55
Sum of Overall Table: 5050
The Average of Overall Table : 50
Minimum of Overall Table : 1
Maximum of Overall Table : 100
EAX=00000064 EBX=00370064 ECX=00000000 EDX=003750C1
ESI=00000009 EDI=0000000A EBP=0135FE8C ESP=0135FE80
EIP=003714AB EFL=00000202 CF=0 SF=0 ZF=0 OF=0 AF=0 PF=0
Press any key to continue . . .

Ashmal Anis 19K-0305

TASK#02:

Machine	Number of General Purpose Registers	Architectural Style	Year	Difference
Arm64/A64	31 (including stack pointer and zero register)	Register-Register	2011	The Arm64/A64 is a 64 bit architecture that is different than previous architectures before 2004 because: all of its addresses are assumed to be 64 bits, and set of conditional instructions have been reduced down to cover branches. Dedicated zero register is available for most instructions.
AVR32	32 (16 on reduced architecture)	Register-Register	2006	It's an updated version of rev 2 and allows operations on up to 3 operands which was not possible in previous architectures before 2005 it was just up to 1 or 2 operands before RISC was introduced.
Elbrus (native VLIW)	8-64	Register-Register	2014	It's a 64 bit, updated version of Elbrus-4S version that supports the architecture type of Register – Register. Previously the machines and architectures such as RX (2000) consisted memory-memory type which is a more complex architecture of CISC than RISC.
ESI- RISC	8/16/32/72	Register-Register	2009	It's a 16/32 bit RISC architectural type that supports up to general purpose registers. The way it's different from previous machines and architectures is because of its increased general purpose

				registers, previously there exists 1, 3, 7, 8, 15, 32 general purpose registers initially which reduces the computational and calculation time.
Mico32	32	Register-Register	2006	It's a 32 bit machine architecture that follows register- register architectural type. It's different than previous because before 2004 instruction encoding styles were mostly variable not fixed. Such as MIPS (1981), but Misco32 entails fixed instruction encoding styles
OpenRisc	16 or 32	Register-Register	2010	OpenRisc architecture is designed in such a way that it supports pipelining, can execute multiple instructions in less clock cycle, it saves time exponentially and increases the processor's performance allowing multiple instructions to be executed at the same time without collisions.
RISC-V	32 (including "zero")	Register-Register	2010	It's a 2.2 version of RISC architecture, it supports little endian-ness which differentiates it from the previous architectures in a way that they used to support bi-endian-ness or big endian-ness which had their limitations, but since RISC was made to make instructions simpler, its little endian-ness adds on to its compatibility.