ELE 338 – Preliminary Work 2

Anıl Karaca – 21728405

Q1)

Source Code:

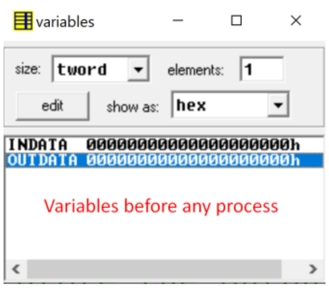
```
org 100h

;ELE338 - Preliminary Work 2 - Question 1

;Anil Karaca - 21728405


;Get input

MOV BX, 0d ;Assign 0 to the counter BX

indata DB 10 DUP (?) ;Initialize the input array


input:

;Read input from screen, wait until a key is pressed

MOV AH, 1d

INT 21h

;Store the input in array indata

MOV AH, 0d

MOV indata[BX], AL

;Increment counter, check if we are done taking input

INC BX

CMP BX, 10d

JNZ input


;Swap case

MOV BX, 0d ;Assign 0 to the counter BX

outdata DB 10 DUP (?) ;Initialize the output array


swap:

XOR indata[BX], 20h ;Swap cases by XOR'ing a character
with 20h

MOV CL, indata[BX] ;Store the output in an intermediate
value

MOV outdata[BX], CL ;Store the output in output array


;Increment counter, check if we are done swapping cases

INC BX

CMP BX, 10d

JNZ swap
```

```
;Display output

MOV BX, 0d ;Assign 0 to the counter BX


;New line

MOV DL, 0Dh ;Assign carriage return to DL

MOV AH, 2d

INT 21h


MOV DL, 0Ah ;Assign line feed to DL

MOV AH, 2d

INT 21h


output:

MOV DL, outdata[BX] ;Assign line feed to DL

;Print the value of DL on the screen

MOV AH, 2d

INT 21h

;Increment counter, check if we are done printing

INC BX

CMP BX, 10d

JNZ output


;Terminate the program

MOV AH, 4CH

INT 21H

ret
```

Screenshots:

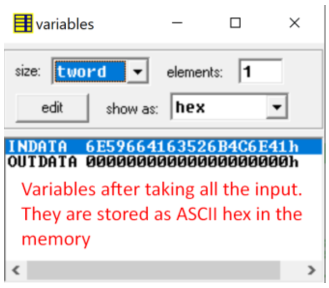
I've used "AnLkRcAfYn", which is the short for "Anıl Karaca Afyon".

Variables:



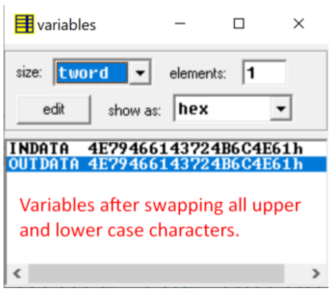
```
INDATA  00000000000000000000h
OUTDATA 00000000000000000000h
```

Variables before any process



```
INDATA  6E59664163526B4C6E41h
OUTDATA 00000000000000000000h
```

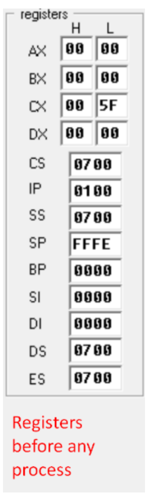
Variables after taking all the input. They are stored as ASCII hex in the memory



```
INDATA  4E79466143724B6C4E61h
OUTDATA 4E79466143724B6C4E61h
```

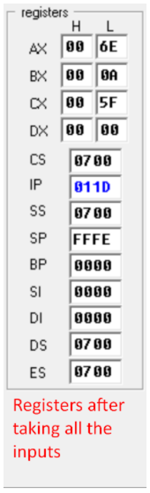
Variables after swapping all upper and lower case characters.

Registers:



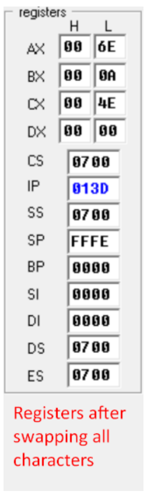
registers	H	L
AX	00	00
BX	00	00
CX	00	5F
DX	00	00
CS	0700	
IP	0100	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

Registers before any process



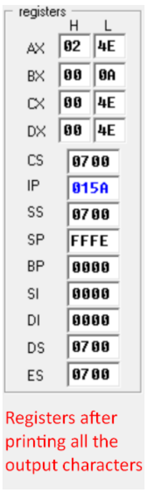
registers	H	L
AX	00	6E
BX	00	0A
CX	00	5F
DX	00	00
CS	0700	
IP	011D	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

Registers after taking all the inputs



registers	H	L
AX	00	6E
BX	00	0A
CX	00	4E
DX	00	00
CS	0700	
IP	013D	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

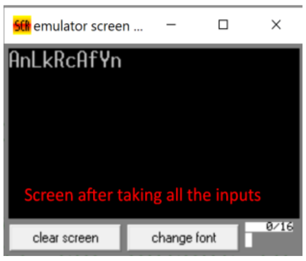
Registers after swapping all characters



registers	H	L
AX	02	4E
BX	00	0A
CX	00	4E
DX	00	4E
CS	0700	
IP	015A	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

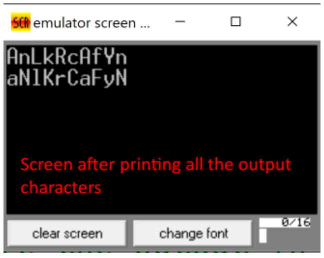
Registers after printing all the output characters

Emulator Screen:



```
AnLkRcAfYn
```

Screen after taking all the inputs



```
AnLkRcAfYn
aNlKrCaFyN
```

Screen after printing all the output characters

Comments:

For this question my approach was; taking the input from the user, storing the input in an array called "indata", swapping the upper and lower case letters, storing the result in an array called "outdata" and then, finally printing the output on the emulator screen.

I get the input(10 letters) from the user by making use of **`MOV AH, 1d`** and **`INT 21h`**. I could also use **`INT 16h`** to detect the keyboard inputs but using **`INT 21h`** allowed me to show user's input on the emulator screen more clearly. This taking input from the user loop continues until the counter BX is 10d. It is 10d because we are dealing with 10 letters.

For swapping the upper and lower case letters, at first I was planning on using some sort of control mechanism to detect whether the letter is upper or lower case, and then I could manipulate it's value. But then I realised XOR'ing a ASCII letter with 100000b(20h) is sufficient to change it's case, so that's the method I applied in the end.

In order to swap cases of all letters, I XOR'ed these letters one by one with 20h, and then assigned their new values to CL temporarily, and finally stored these new values in an array called "outdata".

To print the output, firstly I used 0Dh and 0Ah to move to the beginning of a new line. And then, I printed all the characters by using a loop similar to the one on the input part of the code.

Q2) I've used "transistor inductor capacitor" as an input.

Source Code:

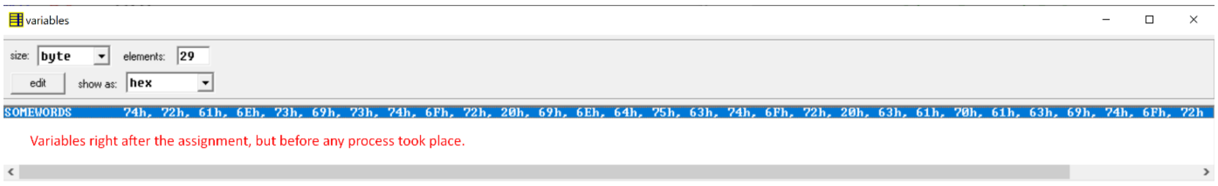
```
org 100h

;ELE338 - Preliminary Work 2 - Question 2

;Anil Karaca - 21728405


JMP start

somewords DB "transistor inductor capacitor",'$'

start:

MOV BX, 0d ;Initialize the counter BX

;Make the first letter uppercase

XOR somewords[BX], 20h


loopOver:

;Check if we encounter the space(20h) character

CMP somewords[BX], 20h

JZ convertNext

;If the character is not space keep looking

INC BX

CMP BX, 29d

JNZ loopOver

;Jump to display if have traversed the whole string

JMP display
```

```
convertNext:

;If we have encountered the space(20h) character make the next character uppercase

INC BX

XOR somewords[BX], 20h ;XOR the character with 20h so we can make it uppercase

;Go back to the loop which traverses the string

JMP loopOver


display:

MOV BX, 0d ;Reset the counter BX


loopDisplay:

MOV DL, somewords[BX] ;Assign the character to DL so we can print it

MOV AH, 2d ;Assign AH to 2d so we can print characters

INT 21h ;Open the command prompt

INC BX

;Check if we have done printing

CMP BX, 29d

JNZ loopDisplay


ret
```

Screenshots:

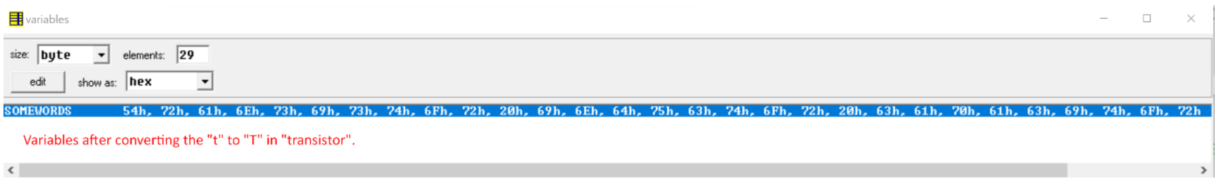
I've used "transistor inductor capacitor" as an input.

Variables:



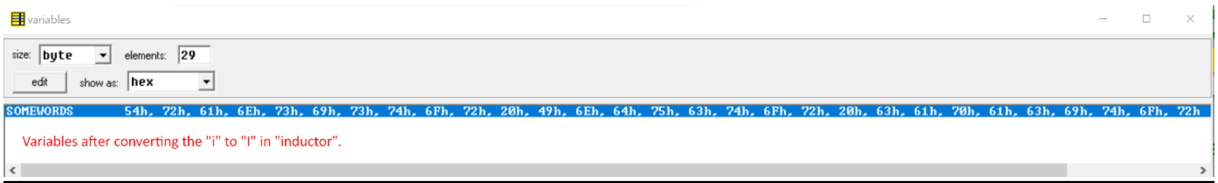
```
SOMEWORDS      74h, 72h, 61h, 6Eh, 73h, 69h, 73h, 74h, 6Fh, 72h, 20h, 69h, 6Eh, 64h, 75h, 63h, 74h, 6Fh, 72h, 20h, 63h, 61h, 70h, 61h, 63h, 69h, 74h, 6Fh, 72h
```

Variables right after the assignment, but before any process took place.



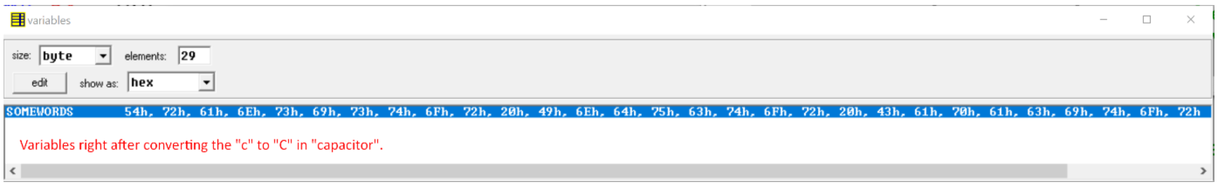
```
SOMEWORDS      54h, 72h, 61h, 6Eh, 73h, 69h, 73h, 74h, 6Fh, 72h, 20h, 69h, 6Eh, 64h, 75h, 63h, 74h, 6Fh, 72h, 20h, 63h, 61h, 70h, 61h, 63h, 69h, 74h, 6Fh, 72h
```

Variables after converting the "t" to "T" in "transistor".



```
SOMEWORDS      54h, 72h, 61h, 6Eh, 73h, 69h, 73h, 74h, 6Fh, 72h, 20h, 49h, 6Eh, 64h, 75h, 63h, 74h, 6Fh, 72h, 20h, 63h, 61h, 70h, 61h, 63h, 69h, 74h, 6Fh, 72h
```

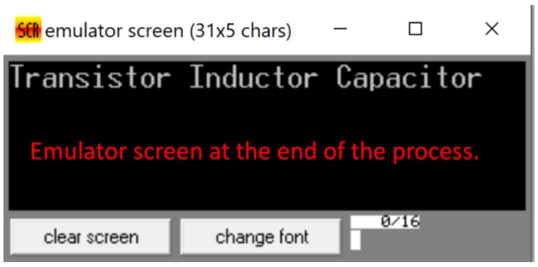
Variables after converting the "i" to "I" in "inductor".



```
SOMEWORDS      54h, 72h, 61h, 6Eh, 73h, 69h, 73h, 74h, 6Fh, 72h, 20h, 49h, 6Eh, 64h, 75h, 63h, 74h, 6Fh, 72h, 20h, 43h, 61h, 70h, 61h, 63h, 69h, 74h, 6Fh, 72h
```

Variables right after converting the "c" to "C" in "capacitor".

Emulator Screen:



```
Transistor Inductor Capacitor
```

Emulator screen at the end of the process.

Comments:

Here my approach was converting the first letter we encounter to uppercase, and then for the rest of the characters I checked whether they were a "space(20h)" character or not. If we encountered a "space(20h)" character the program converts the upcoming letter to uppercase in the "converNext" part of the code.

This algorithm is pretty reasonable because "space(20h)" character indicates that there is a letter we need to convert to uppercase is coming next.

I used BX register as a counter, "loopOver" loop checks whether we have found a "space(20h)" character or not, "convertNext" part of the code converts the upcoming letter to uppercase, and then it directs the program back to "loopOver". After all the conversions are done in order do display the result on the emulator screen I used a loop called "loopDisplay".

Q3) I’ve used “transistor inductor capacitor” as an input.

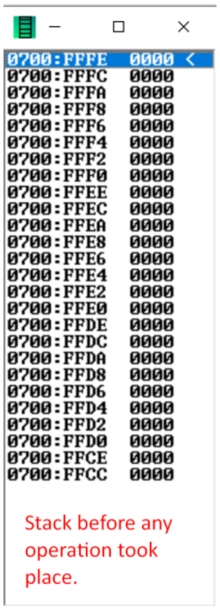
Source Code:

```
org 100h

;ELE338 - Preliminary Work 2 - Question 3

;Anil Karaca - 21728405


JMP start

somewords DB "transistor inductor capacitor",'$'

start:

MOV BX, 0d ;Initialize the counter BX


inStack:

MOV CL, somewords[BX] ;Assign the character to CL so we can push it

PUSH CX ;Push the character to the stack

INC BX

CMP BX, 29d ;Check if we have pushed all the characters

JNZ inStack ;Loop until we push all the characters


MOV BX, 0d ;Reset the counter BX

outStack:

POP CX ;Pop the character out of the stack

MOV somewords[BX], CL ;Assign the characters back to array

INC BX

CMP BX, 29d ;Check if we have popped all the characters

JNZ outStack ;Loop until we pop all the characters

MOV BX, 0d ;Reset the counter BX

display:

MOV DL, somewords[BX] ;Assign the character to DL so we can print it

MOV AH, 2d ;Assign AH to 2d so we can print characters

INT 21h ;Open the command prompt

INC BX

;Check if we have done printing

CMP BX, 29d

JNZ display


ret
```

<u>Screenshots:</u>

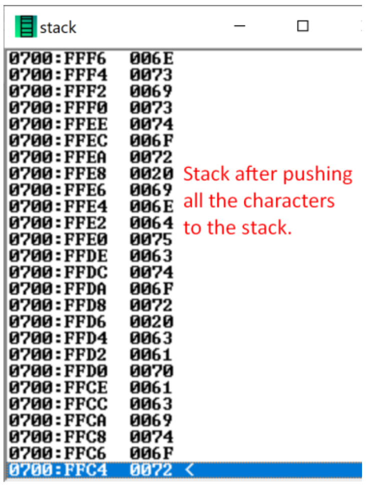
I’ve used “transistor inductor capacitor” as an input.

<u>Stack:</u>



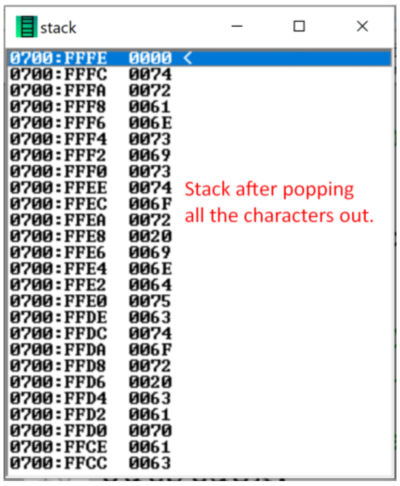
```
0700:FFFE  0000 <
0700:FFFC  0000
0700:FFFA  0000
0700:FFF8  0000
0700:FFF6  0000
0700:FFF4  0000
0700:FFF2  0000
0700:FFF0  0000
0700:FFEE  0000
0700:FFEC  0000
0700:FFEA  0000
0700:FFE8  0000
0700:FFE6  0000
0700:FFE4  0000
0700:FFE2  0000
0700:FFE0  0000
0700:FFDE  0000
0700:FFDC  0000
0700:FFDA  0000
0700:FFD8  0000
0700:FFD6  0000
0700:FFD4  0000
0700:FFD2  0000
0700:FFD0  0000
0700:FFCE  0000
0700:FFCC  0000
```

Stack before any operation took place.



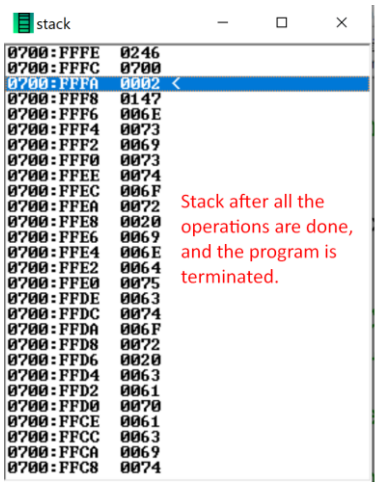
```
0700:FFF6  006E
0700:FFF4  0073
0700:FFF2  0069
0700:FFF0  0073
0700:FFEE  0074
0700:FFEC  006F
0700:FFEA  0072
0700:FFE8  0020
0700:FFE6  0069
0700:FFE4  006E
0700:FFE2  0064
0700:FFE0  0075
0700:FFDE  0063
0700:FFDC  0074
0700:FFDA  006F
0700:FFD8  0072
0700:FFD6  0020
0700:FFD4  0063
0700:FFD2  0061
0700:FFD0  0070
0700:FFCE  0061
0700:FFCC  0063
0700:FFCA  0069
0700:FFC8  0074
0700:FFC6  006F
0700:FFC4  0072 <
```

Stack after pushing all the characters to the stack.



```
0700:FFFE  0000 <
0700:FFFC  0074
0700:FFFA  0072
0700:FFF8  0061
0700:FFF6  006E
0700:FFF4  0073
0700:FFF2  0069
0700:FFF0  0073
0700:FFEE  0074
0700:FFEC  006F
0700:FFEA  0072
0700:FFE8  0020
0700:FFE6  0069
0700:FFE4  006E
0700:FFE2  0064
0700:FFE0  0075
0700:FFDE  0063
0700:FFDC  0074
0700:FFDA  006F
0700:FFD8  0072
0700:FFD6  0020
0700:FFD4  0063
0700:FFD2  0061
0700:FFD0  0070
0700:FFCE  0061
0700:FFCC  0063
```

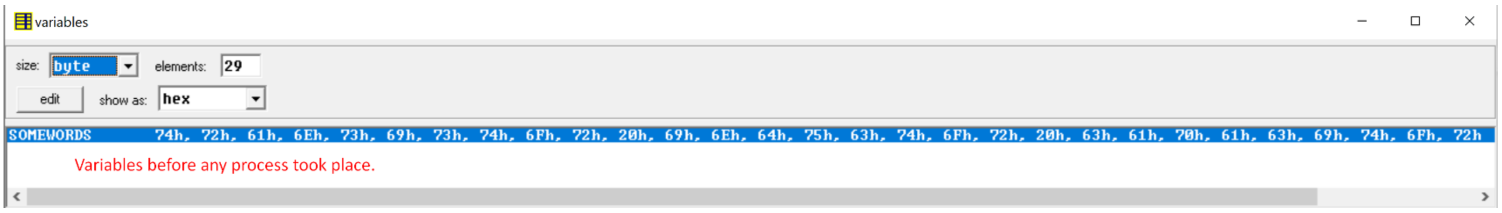
Stack after popping all the characters out.



```
0700:FFFE  0246
0700:FFFC  0700
0700:FFFA  0002 <
0700:FFF8  0147
0700:FFF6  006E
0700:FFF4  0073
0700:FFF2  0069
0700:FFF0  0073
0700:FFEE  0074
0700:FFEC  006F
0700:FFEA  0072
0700:FFE8  0020
0700:FFE6  0069
0700:FFE4  006E
0700:FFE2  0064
0700:FFE0  0075
0700:FFDE  0063
0700:FFDC  0074
0700:FFDA  006F
0700:FFD8  0072
0700:FFD6  0020
0700:FFD4  0063
0700:FFD2  0061
0700:FFD0  0070
0700:FFCE  0061
0700:FFCC  0063
0700:FFCA  0069
0700:FFC8  0074
```

Stack after all the operations are done, and the program is terminated.

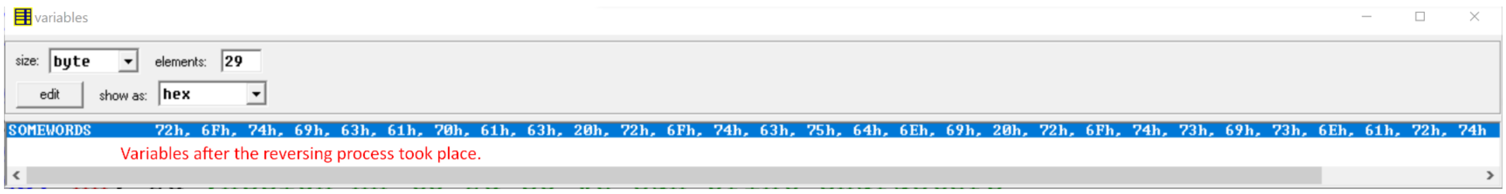
<u>Variables:</u>



size: byte elements: 29
show as: hex

```
SOMEWORDS      74h, 72h, 61h, 6Eh, 73h, 69h, 73h, 74h, 6Fh, 72h, 20h, 69h, 6Eh, 64h, 75h, 63h, 74h, 6Fh, 72h, 20h, 63h, 61h, 70h, 61h, 63h, 69h, 74h, 6Fh, 72h
```

Variables before any process took place.

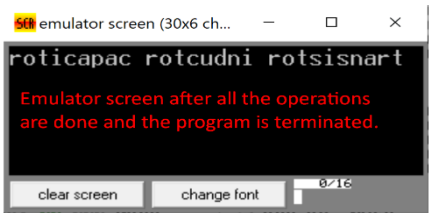


size: byte elements: 29
show as: hex

```
SOMEWORDS      72h, 6Fh, 74h, 69h, 63h, 61h, 70h, 61h, 63h, 20h, 72h, 6Fh, 74h, 63h, 75h, 64h, 6Eh, 69h, 20h, 72h, 6Fh, 74h, 73h, 69h, 73h, 6Eh, 61h, 72h, 74h
```

Variables after the reversing process took place.

<u>Emulator Screen:</u>



```
roticapac rotcudni rotsisnart
```

Emulator screen after all the operations are done and the program is terminated.

<u>Comments:</u>

Here I used the “first in last out” property of stack data structure to reverse a string.

In the “inStack” part of the code I pushed all the characters in the string to a stack by using CX register as an intermediate value holder and BX register as a counter. In the ”outStack” part of the code we pop all the characters back to our variable in a reverse order. Finally, in the “display” part of the code we print the characters stored in the variable one by one on the emulator screen by making use of interrupts.