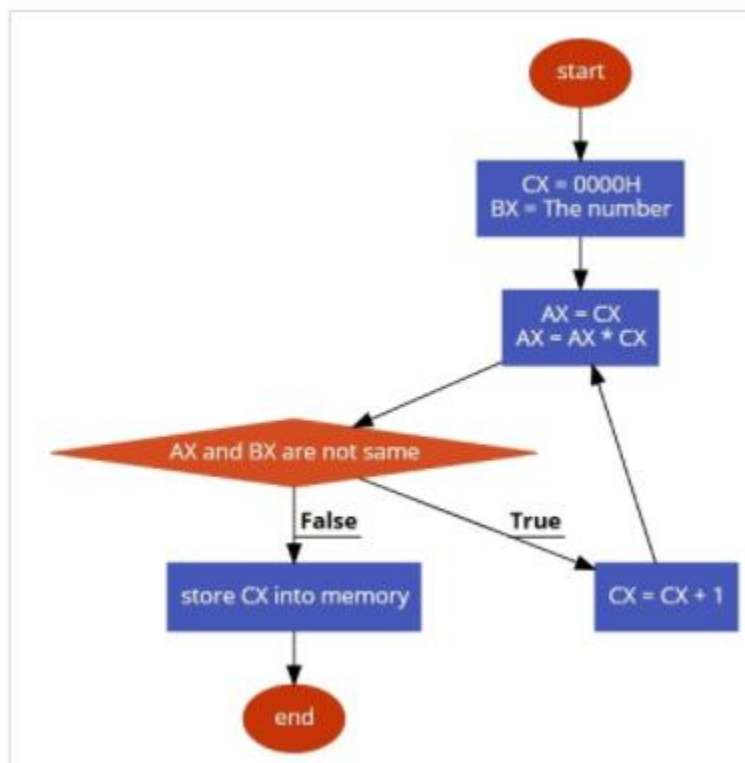


## Important Topics

- ✓ Verify the square root of a perfect squared number among number
- ✓ Generate and verify GP series
- ✓ To generate AP series
- ✓ Addition and average of a series
- ✓ To arrange a given series of hexadecimal bytes (67, 25, 14, and 02) in descending order
- ✓ To arrange a given series of hexadecimal bytes (67, 25, 14, and 02) in ascending order
- ✓ Assembly language program to generate Fibonacci sequence
- ✓ To convert into the following number system.
  - Binary to 2s Complement
  - BCD to binary
  - Binary to BCD
- ✓ Program to find and display the factorial of a number stored in memory offset
- ✓ Largest and smallest number in an array
- ✓ Write and verify 8086 assembly language program that will perform following string operation for a string: "This is a microprocessor and interfacing lab exam"
  - i. Calculate length of a string
  - ii. Count number of spaces in a string
  - iii. Reverse the given string and print the reversed string

Activate Windows  
Go to Settings to activate Windows.

## 1) SQRT:



# Code

Assume CS: Code DS: Data

**DATA SEGMENT**

**NUM DW 211H**

**SQRT DW ?**

**DATA ENDS**

**CODE SEGMENT**

**START:**

**MOV AX, @DATA**

**MOV DS, AX**

**MOV CX,0000H**

**MOV BX, Num**

**L1:MOV AX,CX**

**MUL CX**

**CMP AX, BX**

**JZ STORE**

**INC CX**

**JNZ L1**

**STORE:MOV DX,CX**

**MOV SQRT,DX**

**HLT**

**CODE ENDS**

# END START

```
01 Assume CS: Code DS: Data
02 DATA SEGMENT
03
04     NUM DW 211H
05     SQRT DW ?
06
07 DATA ENDS
08 CODE SEGMENT
09     START:
10     MOV AX, @DATA
11     MOV DS, AX
12     MOV CX, 0000H
13     MOV BX, Num
14     L1: MOV AX, CX
15         MUL CX
16     CMP AX, BX
17     JZ STORE
18     INC CX
19     JNZ L1
20 STORE: MOV DX, CX
21         MOV SQRT, DX
22     HLT
23     CODE ENDS
24 END START
25
```

## Before Exe

variables	
size: word	elements: 1
edit	show as: hex
NUM	0211h
SQRT	0000h

# OUTPUT:

The screenshot shows an x86 emulator window titled "emulator: noname.exe\_". The interface includes a menu bar (file, math, debug, view, external, virtual devices, virtual drive, help), a toolbar with buttons for Load, reload, step back, single step, and run, and a step delay slider set to 0 ms.

On the left, there are two panels:

- variables:** Shows a list of variables. "NUM" is at address 0211h and "SQRT" is at address 0017h. Both are of size "word" and contain 1 element. The "show as" is set to "hex".
- original source code:** Displays assembly code. Line 22, "HLT", is highlighted in yellow.

The main area shows the state of registers and memory:

- registers:** A table showing the high (H) and low (L) bytes of various registers. AX, BX, CX, and DX all have H=02 and L=11. CS is 0711, IP is 001D, SS is 0710, SP is 0000, BP is 0000, SI is 0000, DI is 0000, DS is 0710, and ES is 0700.
- Memory:** Two windows show memory contents. The first window, titled "0711:001D", shows a list of memory addresses from 0712D to 07146. The second window, titled "0711:0019", shows a list of instructions starting with "MOV AX, 00710h" and ending with "HLT".

# After Exe

The screenshot shows the "variables" window after execution. The "NUM" variable at address 0211h now contains the value 0211h, and the "SQRT" variable at address 0017h now contains the value 0017h. The "show as" is still set to "hex".

## 2)GP

### Code:

**Assume CS: Code DS: Data**

**DATA SEGMENT**

**Input DW 0500H**

**Output DW 0600H**

**DATA ENDS**

**CODE SEGMENT**

**START:**

**MOV AX, @DATA**

**MOV DS, AX**

**MOV SI, Input**

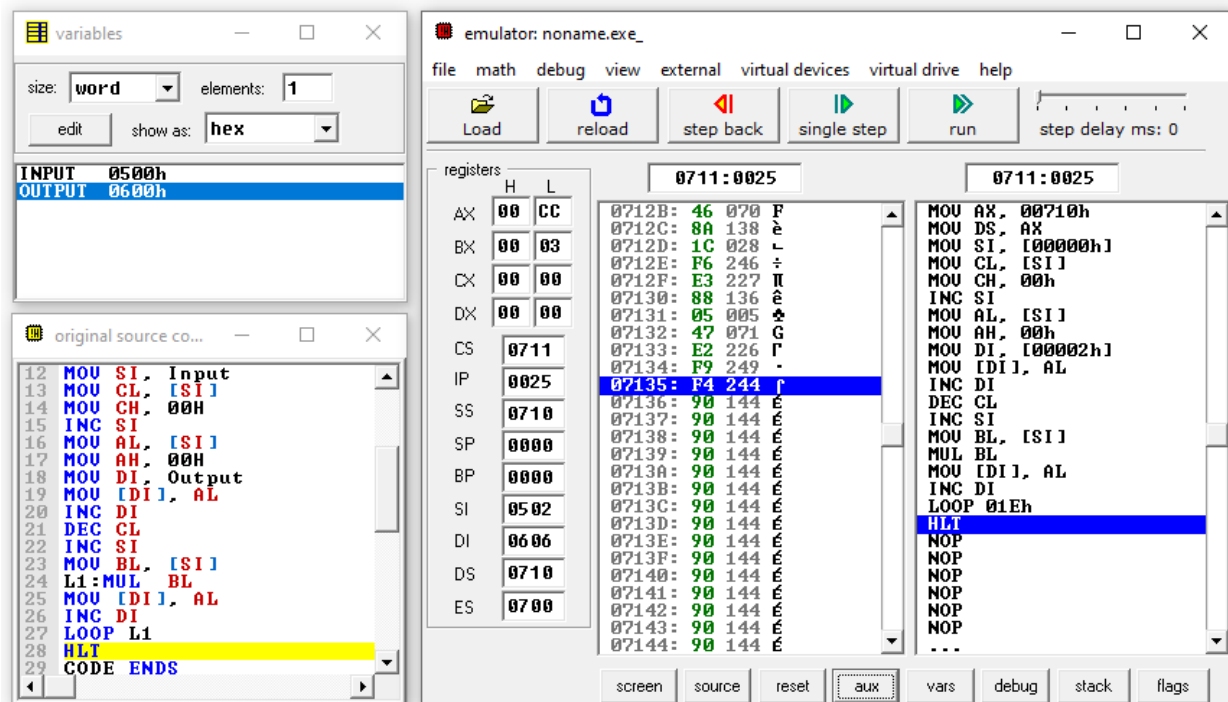
**MOV CL, [SI]**

```
MOV CH, 00H  
INC SI  
MOV AL, [SI]  
MOV AH, 00H  
MOV DI, Output  
MOV [DI], AL  
INC DI  
DEC CL  
INC SI  
MOV BL, [SI]  
L1:MUL BL  
MOV [DI], AL  
INC DI  
LOOP L1  
HLT  
CODE ENDS  
END START
```

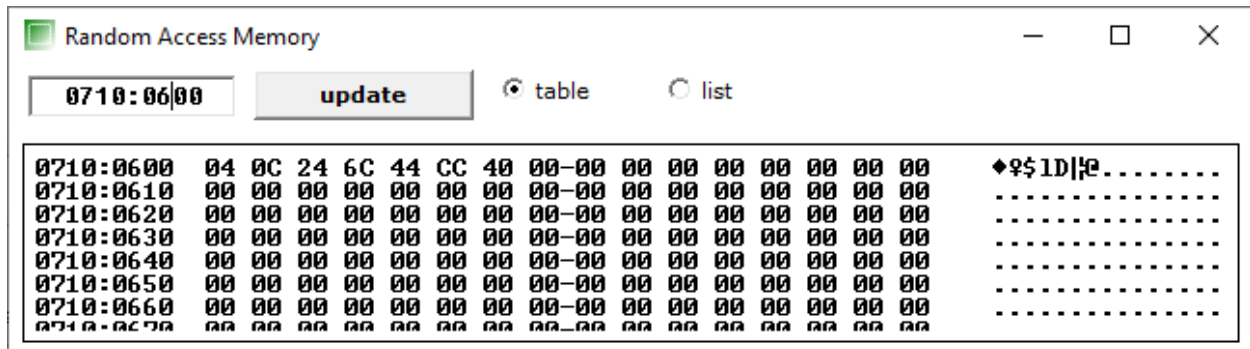


Given input (Number of terms) n=06h in 0710:0500h and first term a=04h in 0710:0501h and common ratio r =03h in 0710:0502h

Output:



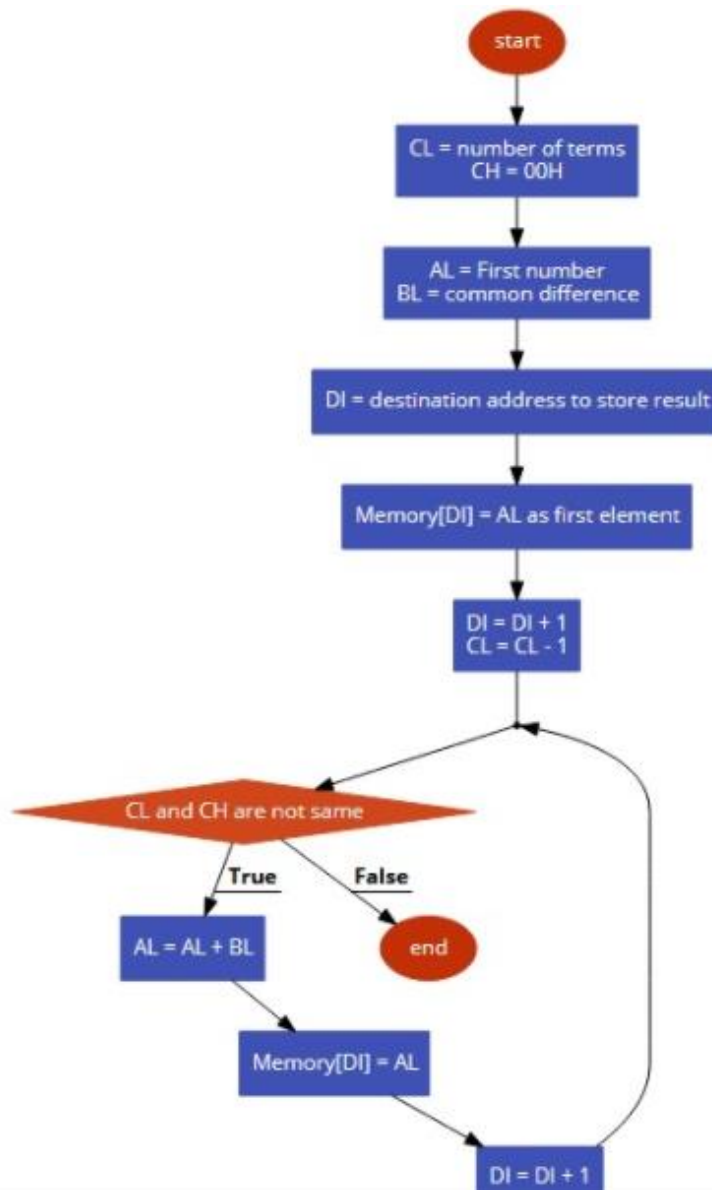
After Execution:





**GP Series obtained starting from 600  
location**

### 3) AP



### CODE:

Assume CS: Code DS: Data

DATA SEGMENT

**Input DW 0500H**

**Output DW 0600H**

**DATA ENDS**

**CODE SEGMENT**

**START:**

**MOV AX, @DATA**

**MOV DS, AX**

**MOV SI, Input**

**MOV CL, [SI]**

**MOV CH, 00H**

**INC SI**

**MOV AL, [SI]**

**MOV AH, 00H**

**MOV DI, Output**

**MOV [DI], AL**

**INC DI**

**DEC CL**

```
INC SI
MOV BL, [SI]
L1:ADD AL, BL
MOV [DI], AL
INC DI
LOOP L1
HLT
CODE ENDS
END START
```

```
01 Assume CS: Code DS: Data
02 DATA SEGMENT
03
04     Input DW 0500H
05     Output DW 0600H
06
07 DATA ENDS
08 CODE SEGMENT
09     START:
10     MOV AX, @DATA
11     MOV DS, AX
12     MOV SI, Input
13     MOV CL, [SI]
14     MOV CH, 00H
15     INC SI
16     MOV AL, [SI]
17     MOV AH, 00H
18     MOV DI, Output
19     MOV [DI], AL
20     INC DI
21     DEC CL
22     INC SI
23     MOV BL, [SI]
24     L1:ADD AL, BL
25     MOV [DI], AL
26     INC DI
27     LOOP L1
28     HLT
29 CODE ENDS
30 END START
31
32
```

Input:

Before Execution

Random Access Memory																07603	
0710:0500				update												table	
0710:0500				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00												list	
0710:0501				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													
0710:0502				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													
0710:0503				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													
0710:0504				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													
0710:0505				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													
0710:0506				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													
0710:0507				00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00													

Given input Number of Terms  $n=10h$  in  
0710:0500h and first term  $a=02h$  in  
0710:0501h and common difference  $d=03h$   
in 0710:0502h

Output:

variables

size: word elements: 1

edit show as: hex

INPUT	0500h
OUTPUT	0600h

original source co...

```
12 MOV SI, Input
13 MOV CL, [SI]
14 MOV CH, 00h
15 INC SI
16 MOV AL, [SI]
17 MOV AH, 00h
18 MOV DI, Output
19 MOV [DI], AL
20 INC DI
21 DEC CL
22 INC SI
23 MOV BL, [SI]
24 L1: ADD AL, BL
25 MOV [DI], AL
26 INC DI
27 LOOP L1
28 HLT
29 CODE ENDS
```

emulator: noname.exe

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

AX	00	2F
BX	00	03
CX	00	00
DX	00	00
CS	0711	
IP	0025	
SS	0710	
SP	0000	
BP	0000	
SI	0502	
DI	0610	
DS	0710	
ES	0700	

0711:0025

0712B:	46	070	F
0712C:	8A	138	è
0712D:	1C	028	L
0712E:	02	002	0
0712F:	C3	195	J
07130:	88	136	è
07131:	05	005	è
07132:	47	071	G
07133:	E2	226	r
07134:	F9	249	-
07135:	F4	244	f
07136:	90	144	E
07137:	90	144	E
07138:	90	144	E
07139:	90	144	E
0713A:	90	144	E
0713B:	90	144	E
0713C:	90	144	E
0713D:	90	144	E
0713E:	90	144	E
0713F:	90	144	E
07140:	90	144	E
07141:	90	144	E
07142:	90	144	E
07143:	90	144	E
07144:	90	144	E

0711:0025

```
MOV AX, 00710h
MOV DS, AX
MOV SI, [00000h]
MOV CL, [SI]
MOV CH, 00h
INC SI
MOV AL, [SI]
MOV AH, 00h
MOV DI, [00002h]
MOV [DI], AL
INC DI
DEC CL
INC SI
MOV BL, [SI]
ADD AL, BL
MOV [DI], AL
INC DI
LOOP 01Eh
HLT
NOP
NOP
NOP
NOP
NOP
NOP
...
```

## After Execution:

Random Access Memory																	
0710:0600		update		<input checked="" type="radio"/> table		<input type="radio"/> list											
0710:0600	02	05	08	0B	0E	11	14	17-1A	1D	20	23	26	29	2C	2F	0A.0B4911→+ #8>,,	
0710:0610	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	
0710:0620	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	
0710:0630	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	
0710:0640	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	
0710:0650	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	
0710:0660	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	
0710:0670	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	

**AP Series obtain starting from 600 location.**

## 4) ADDITION AND AVERAGE OF A SERIES

### Code

**Assume CS: Code DS: Data**

**DATA SEGMENT**

**InputLoc DW 0500H**

**SIZE DB 6h**

**SUM DW ?**

**AVG DW ?**

**DATA ENDS**

**CODE SEGMENT**

**START:**

**MOV AX, @DATA**

**MOV DS, AX**

**MOV SI, InputLoc**

```
MOV CL, size
MOV CH, 00H
MOV AL, [SI]
MOV AH, 00H
DEC CL
INC SI
L1:MOV BL,[SI]
MOV BH,00H
ADD AX,BX
INC SI
LOOP L1
MOV SUM,AX
MOV SUM,AX
MOV BL, SIZE
DIV BL
MOV AVG,AX
HLT
CODE ENDS
```



# END START

```
edit: C:\emu8086\MySource\Avg and Sum of series.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  options  help

01  Assume CS: Code DS: Data
02  DATA SEGMENT
03
04      InputLoc DW 0500H
05      SIZE DB 6h
06      SUM DW ?
07      AUG DW ?
08  DATA ENDS
09  CODE SEGMENT
10      START:
11      MOV AX, @DATA
12      MOV DS, AX
13      MOV SI, InputLoc
14      MOV CL, size
15      MOV CH, 00H
16      MOV AL, [SI]
17      MOV AH, 00H
18      DEC CL
19      INC SI
20  L1: MOV BL, [SI]
21      MOV BH, 00H
22      ADD AX, BX
23      INC SI
24      LOOP L1
25      MOV SUM, AX
26      MOV SUM, AX
27      MOV BL, SIZE
28      DIV BL
29      MOV AUG, AX
30      HLT
31      CODE ENDS
32  END START
33
34
```

Input:

Before Execution

Memory

Random Access Memory																			
0710:0500					update					table					list				
0710:0500	10	30	40	18	38	50	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0510	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0520	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0530	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0540	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0550	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0560	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0710:0570	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

## Variables

variables	
size: word	elements: 1
edit	show as: hex
INPUTLOC 0500h	
SIZE 06h	
SUM 0000h	
AUG 0000h	

## OUTPUT

variables	
size: word	elements: 1
edit	show as: hex
INPUTLOC 0500h	
SIZE 06h	
SUM 0120h	
AUG 0030h	


original source code	
15	MOV CH, 00h
16	MOV AL, [SI]
17	MOV AH, 00h
18	DEC CL
19	INC SI
20	LI: MOV BL, [SI]
21	MOV BH, 00h
22	ADD AX, BX
23	INC SI
24	LOOP LI
25	MOV SUM, AX
26	MOV SUM, AX
27	MOV BL, SIZE
28	DIV BL
29	MOV AUG, AX
30	HLT
31	CODE ENDS

emulator: Avg and Sum of series.exe_	
file math debug view external virtual devices virtual drive help	
Load	reload
step back	single step
run	step delay ms: 0

registers	
AX	00 30
BX	00 06
CX	00 00
DX	00 00
CS	0711
IP	002E
SS	0710
SP	0000
BP	0000
SI	0506
DI	0000
DS	0710
ES	0700

0711:002E	
07126: 8A 138	MOV AX, 00710h
07127: 1C 028	MOV DS, AX
07128: B7 183	MOV SI, [00000h]
07129: 00 000	MOV CL, [00002h]
0712A: 03 003	MOV CH, 00h
0712B: C3 195	MOV AL, [SI]
0712C: 46 070	MOV AH, 00h
0712D: E2 226	DEC CL
0712E: F7 247	INC SI
0712F: A3 163	MOV BL, [SI]
07130: 03 003	MOV BH, 00h
07131: 00 000	ADD AX, BX
07132: A3 163	INC SI
07133: 03 003	LOOP 016h
07134: 00 000	MOV [00003h], AX
07135: 8A 138	MOV [00003h], AX
07136: 1E 030	MOV BL, [00002h]
07137: 02 002	DIV BL
07138: 00 000	MOV [00005h], AX
07139: F6 246	HLT
0713A: F3 243	NOP
0713B: A3 163	NOP
0713C: 05 005	NOP
0713D: 00 000	NOP
0713E: F4 244	NOP
0713F: 9B 144	...

# Variables

 variables

size: word

elements: 1

edit

show as: hex

INPUTLOC	0500h
SIZE	06h
SUM	0120h
AUG	0030h

## **5)DESCENDING**

→ We can follow the same Procedure as in Ascending order

**Only 1 line of code will change i.e.**

In Ascending code

**CMP AL,BL**

In Descending code

**CMP BL,AL**

**CODE**

**DATA SEGMENT**

**Array DB 57H,10H,56H,25H,32H,35H**

**SIZE DB 6H**

**DATA ENDS**

**CODE SEGMENT**

**ASSUME CS:CODE,DS:DATA**

**START: MOV AX,DATA  
MOV DS,AX**

**MOV CH,SIZE  
DEC CH**

**UP2: MOV CL,SIZE  
DEC CL  
LEA SI,Array**

**UP1: MOV AL,[SI]  
MOV BL,[SI+1]  
CMP BL,AL  
JC DOWN  
MOV DL,[SI+1]  
XCHG [SI],DL  
MOV [SI+1],DL**

**DOWN: INC SI**

**DEC CL**

**JNZ UP1**

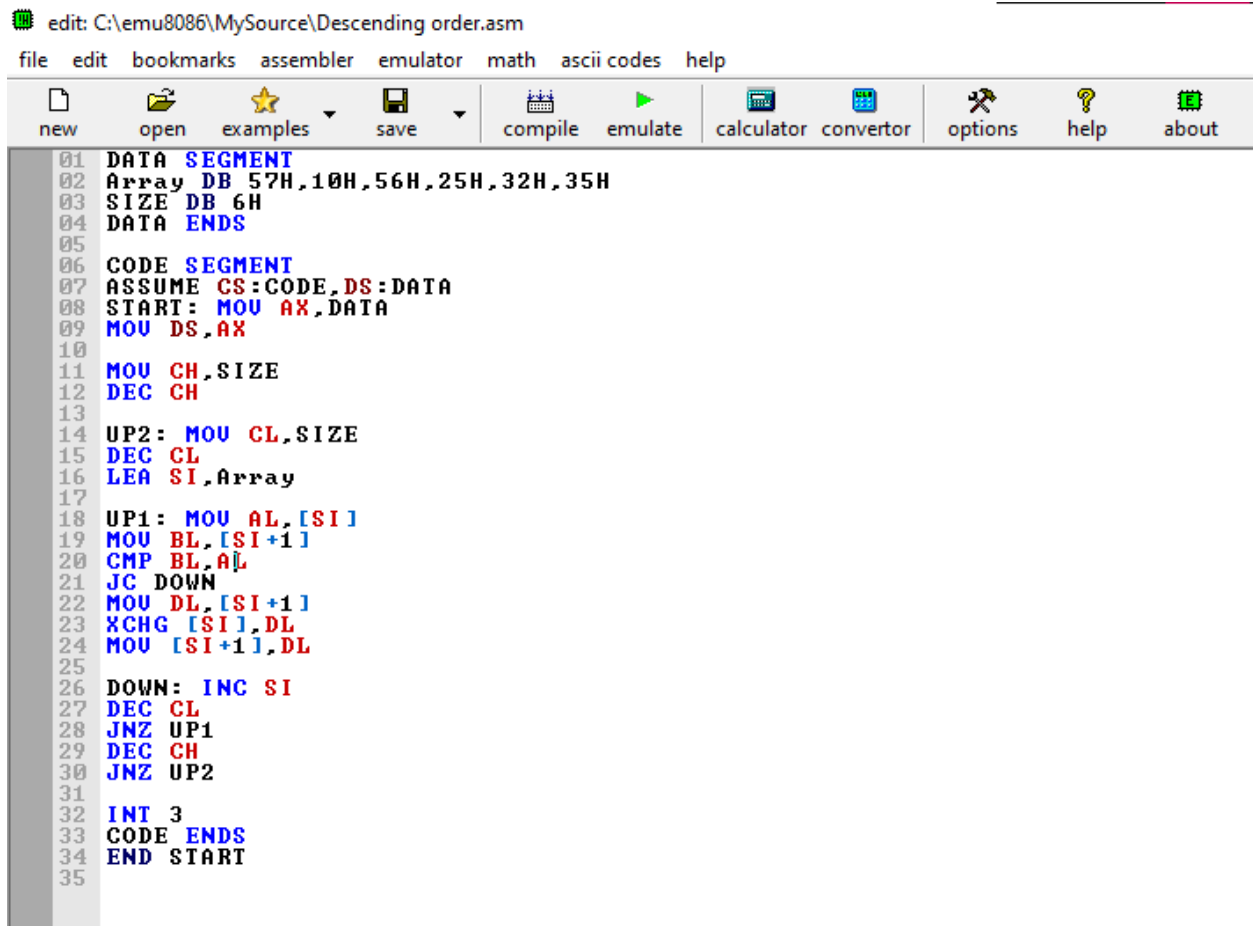
**DEC CH**

**JNZ UP2**

**INT 3**

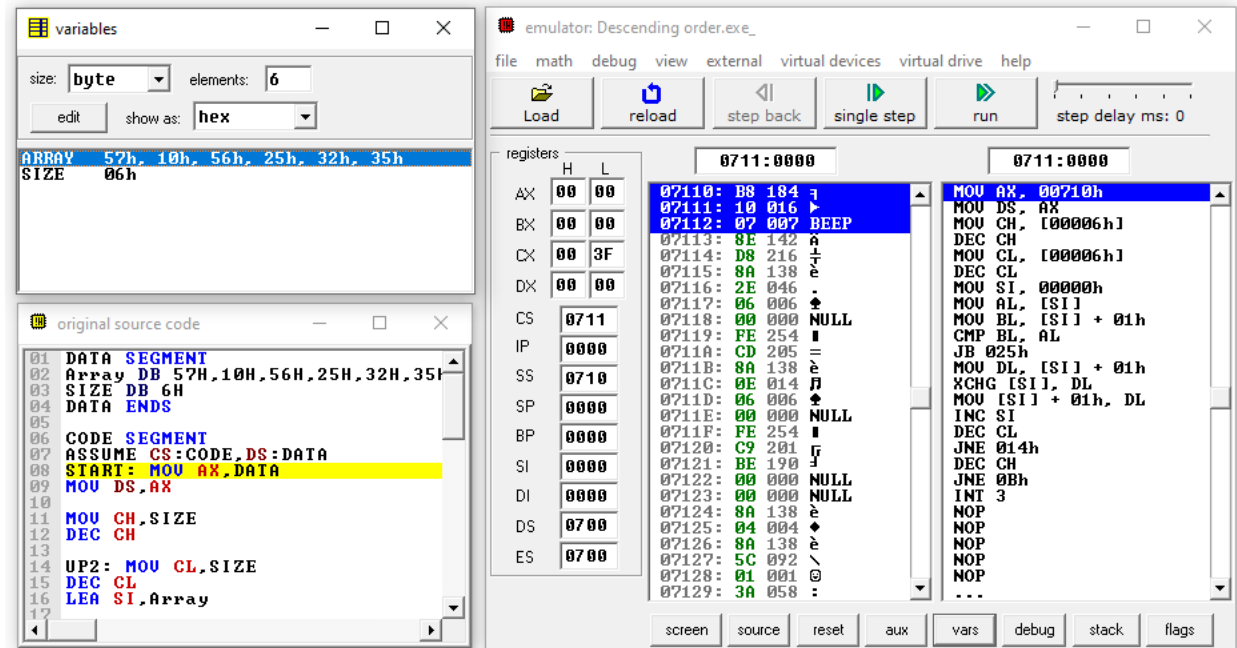
**CODE ENDS**

**END START**

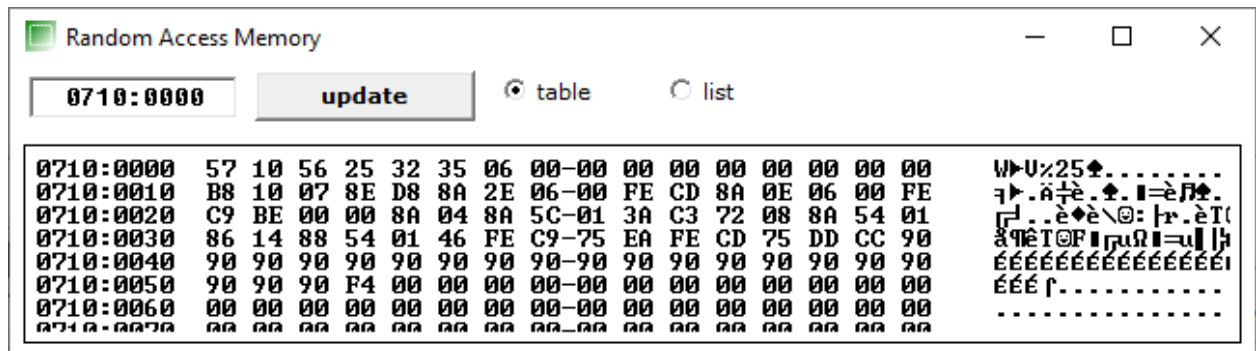


```
edit: C:\emu8086\MySource\Descending order.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  options  help  about
01  DATA SEGMENT
02  Array DB 57H,10H,56H,25H,32H,35H
03  SIZE DB 6H
04  DATA ENDS
05
06  CODE SEGMENT
07  ASSUME CS:CODE,DS:DATA
08  START: MOV AX,DATA
09  MOV DS,AX
10
11  MOV CH,SIZE
12  DEC CH
13
14  UP2: MOV CL,SIZE
15  DEC CL
16  LEA SI,Array
17
18  UP1: MOV AL,[SI]
19  MOV BL,[SI+1]
20  CMP BL,AL
21  JC DOWN
22  MOV DL,[SI+1]
23  XCHG [SI],DL
24  MOV [SI+1],DL
25
26  DOWN: INC SI
27  DEC CL
28  JNZ UP1
29  DEC CH
30  JNZ UP2
31
32  INT 3
33  CODE ENDS
34  END START
35
```

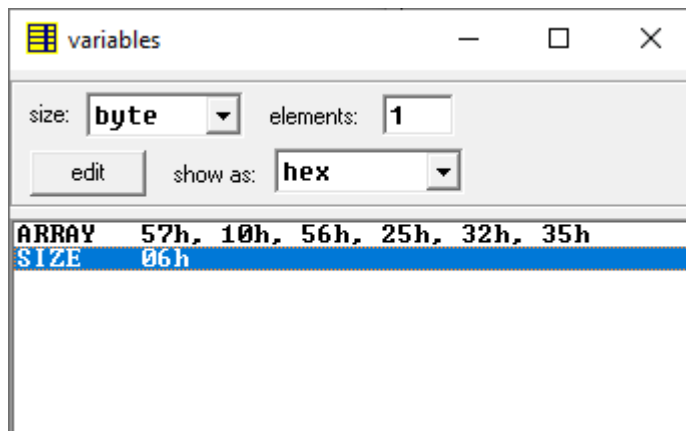
**Before Execution:**



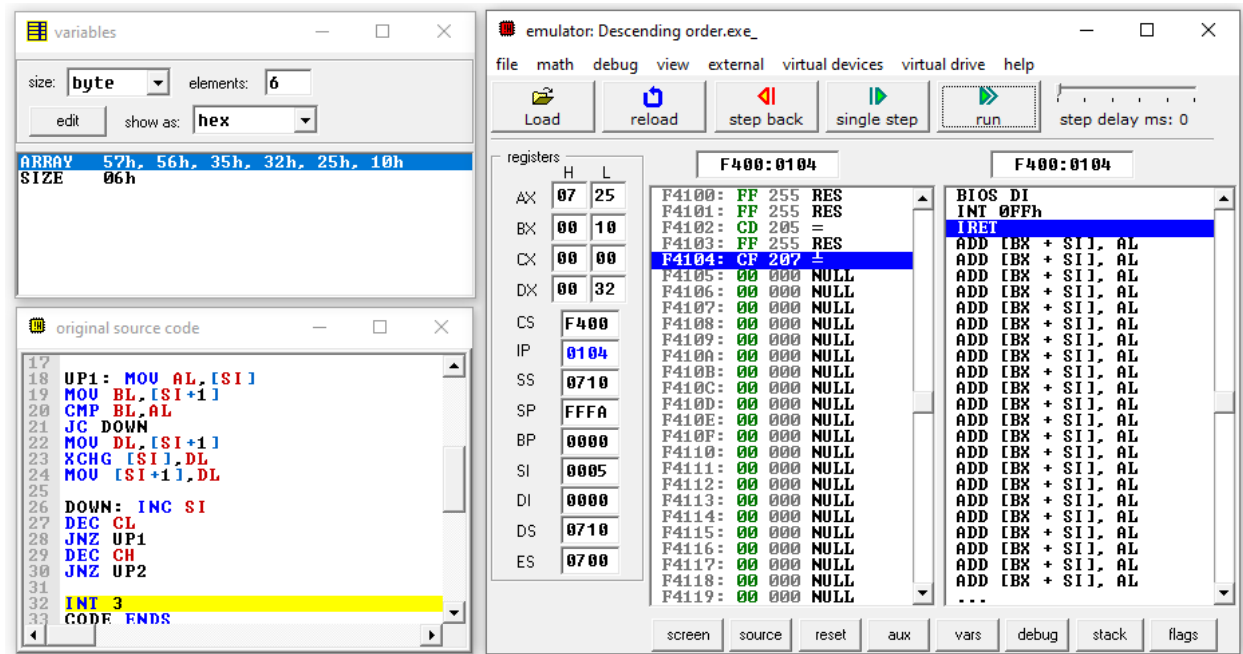
## Memory Location



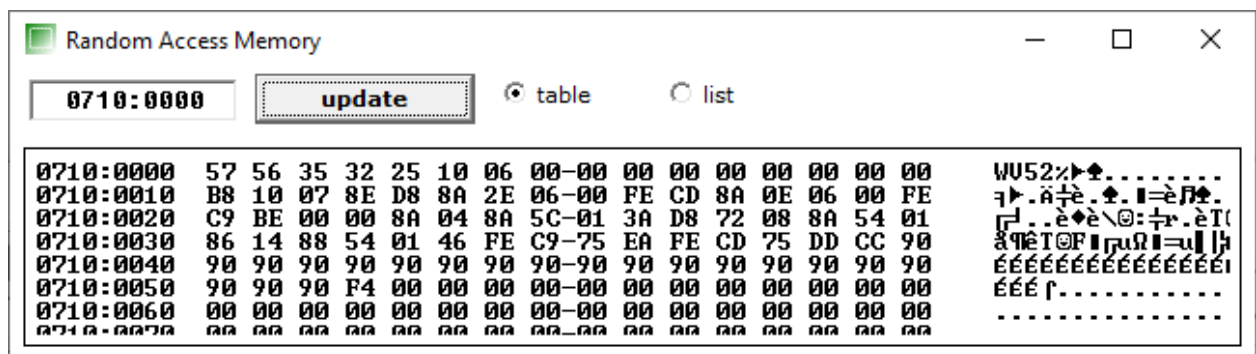
## Variables



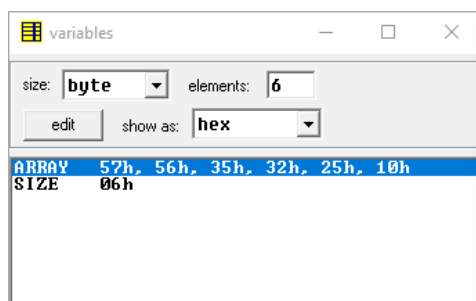
## After Execution:



## Memory Location



# Variables





## **6) ASCENDING**

**Copy Algo, Flowchart, design from the lab doc  
change Calculations an below SS**

**Lab DA6 → Experiment 8.1 → Question 1**

### **CODE**

**DATA SEGMENT**

**Array DB 57H,10H,56H,25H,32H,35H**

**SIZE DB 6H**

**DATA ENDS**

**CODE SEGMENT**

**ASSUME CS:CODE,DS:DATA**

**START: MOV AX,DATA**

**MOV DS,AX**

**MOV CH,SIZE**

**DEC CH**

**UP2: MOV CL,SIZE**

**DEC CL**

**LEA SI,Array**

**UP1: MOV AL,[SI]**

**MOV BL,[SI+1]**

**CMP AL,BL**

**JC DOWN**

**MOV DL,[SI+1]**

**XCHG [SI],DL**

**MOV [SI+1],DL**

**DOWN: INC SI**

**DEC CL**

**JNZ UP1**

**DEC CH**

**JNZ UP2**

**INT 3**

# CODE ENDS

# END START

```
01 DATA SEGMENT
02 Array DB 57H,10H,56H,25H,32H,35H
03 SIZE DB 6H
04 DATA ENDS
05
06 CODE SEGMENT
07 ASSUME CS:CODE,DS:DATA
08 START: MOV AX,DATA
09 MOV DS,AX
10
11 MOV CH,SIZE
12 DEC CH
13
14 UP2: MOV CL,SIZE
15 DEC CL
16 LEA SI,Array
17
18 UP1: MOV AL,[SI]
19 MOV BL,[SI+1]
20 CMP AL,BL
21 JC DOWN
22 MOV DL,[SI+1]
23 XCHG [SI],DL
24 MOV [SI+1],DL
25
26 DOWN: INC SI
27 DEC CL
28 JNZ UP1
29 DEC CH
30 JNZ UP2
31
32 INT 3
33 CODE ENDS
34 END START
35
```

## Before Execution:

The screenshot displays the state of an x86 emulator before execution. It includes three main windows:

- variables**: Shows the 'SIZE' variable at address 00710h with a value of 06h.
- original source code**: Shows the assembly code with the 'START' label highlighted.
- emulator: Exp8 Q1.exe**: Shows the registers and memory. The registers window shows CS:0711, IP:0000, SS:0710, SP:0000, BP:0000, SI:0000, DI:0000, DS:0700, ES:0700. The memory window shows the instruction at 0710:0006: MOV AX, 00710h.

## Memory Location

Random Access Memory																	
0710:0000		update		<input checked="" type="radio"/> table		<input type="radio"/> list											
0710:0000	57	10	56	25	32	35	06	00-00	00	00	00	00	00	00	00	WU%25	.....
0710:0010	B8	10	07	8E	D8	8A	2E	06-00	FE	CD	8A	0E	06	00	FE	7P.A+e.+=eP.	
0710:0020	C9	BE	00	00	8A	04	8A	5C-01	3A	C3	72	08	8A	54	01	7P..e+e\@: P.eT	
0710:0030	86	14	88	54	01	46	FE	C9-75	EA	FE	CD	75	DD	CC	90	89eT@F P u u P	
0710:0040	90	90	90	90	90	90	90	90-90	90	90	90	90	90	90	90	éééééééééééééééé	
0710:0050	90	90	90	F4	00	00	00	00-00	00	00	00	00	00	00	00	éééééééééééééééé	
0710:0060	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	éééééééééééééééé	
0710:0070	00	00	00	00	00	00	00	00-00	00	00	00	00	00	00	00	.....	

## Variables

variables

size: 

byte

elements: 

1

edit

show as: 

hex

ARRAY

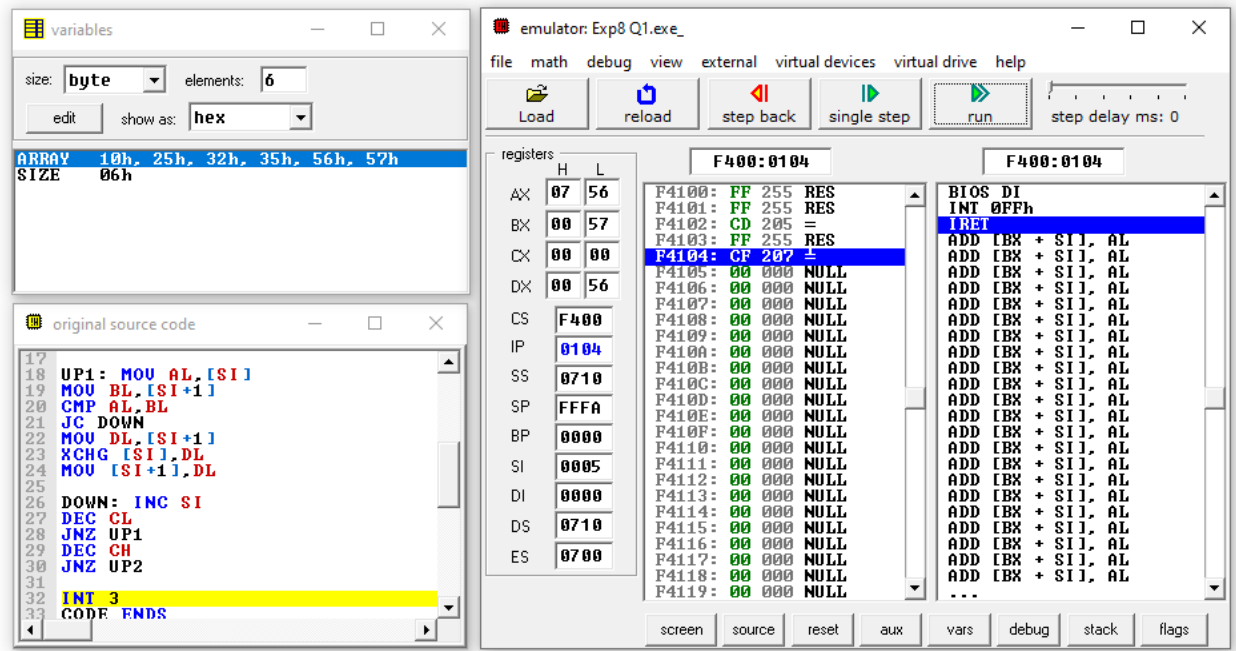
57h, 10h, 56h, 25h, 32h, 35h

SIZE

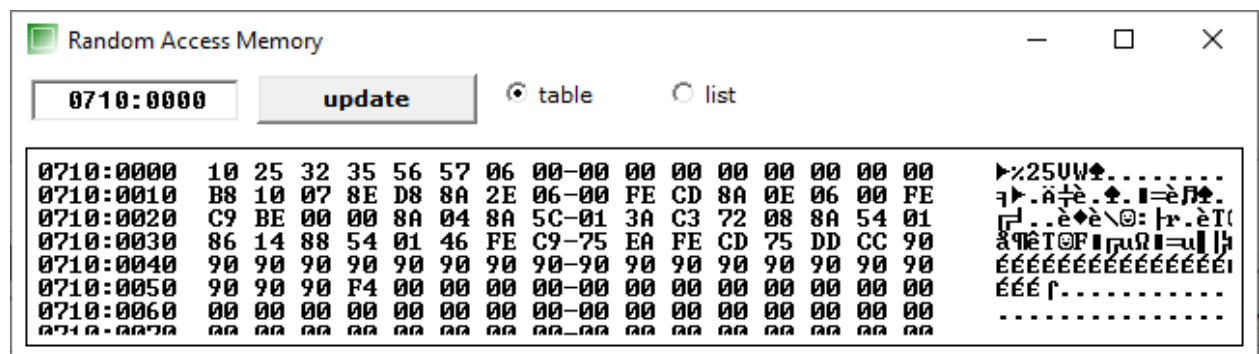
06h

## OUTPUT:

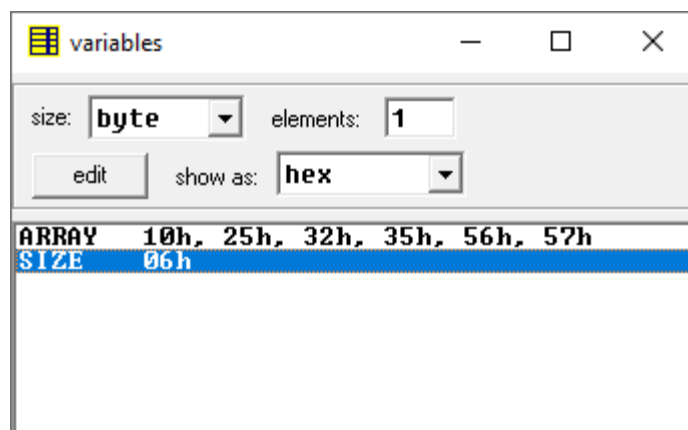
After Execution:



## Memory Location



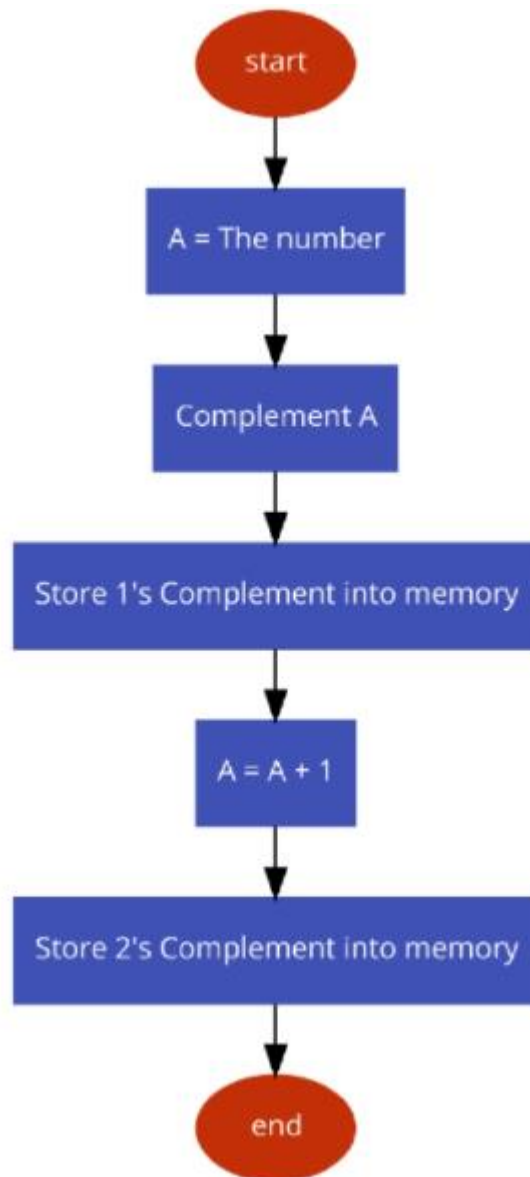
## Variables



## 7) FIBANOCCHI SERIES

Lab DA5 → Experiment 6 → Question 2 → 2<sup>nd</sup> part

## 8 A) BINARY TO 2'S COMP



## Code

### Data Segment

**bin db 00000010B**

**2comp db ?**

## **Data Ends**

## **Code Segment**

**Assume cs:code, ds:data**

### **Begin:**

**mov ax, data**

**mov ds, ax**

**mov ah, 0000h**

**mov al, bin**

**NOT al**

**mov bl, al**

**adc al, 00000001B**

**mov bl, al**

### **Exit:**

**mov 2comp,al**



**mov ax, 4c00h**

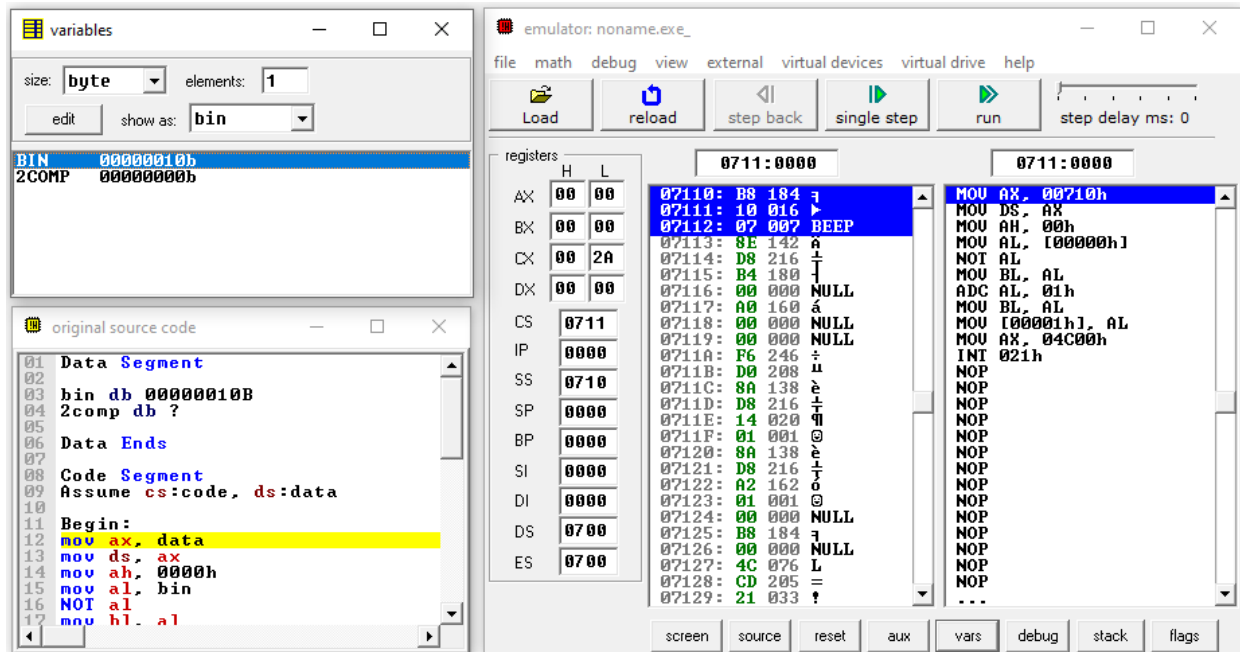
**int 21h**

**Code Ends**

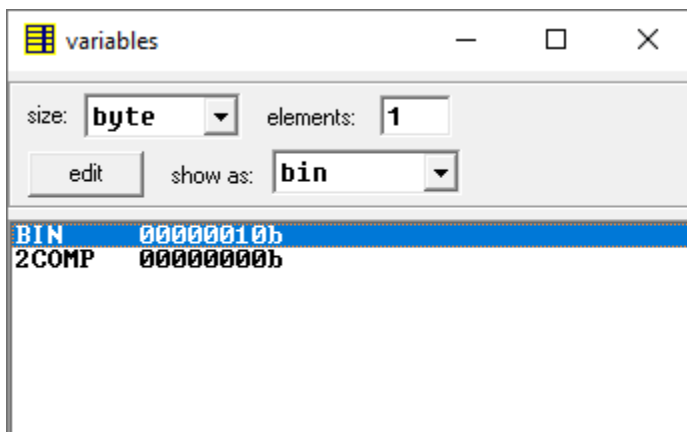
**End Begin**

```
01 Data Segment
02
03     bin db 00000010B
04     2comp db ?
05
06 Data Ends
07
08 Code Segment
09     Assume cs:code, ds:data
10
11     Begin:
12         mov ax, data
13         mov ds, ax
14         mov ah, 0000h
15         mov al, bin
16         NOT al
17         mov bl, al
18         adc al, 00000001B
19         mov bl, al
20
21     Exit:
22         mov 2comp, al
23         mov ax, 4c00h
24         int 21h
25 Code Ends
26 End Begin
```

**Before Exe**

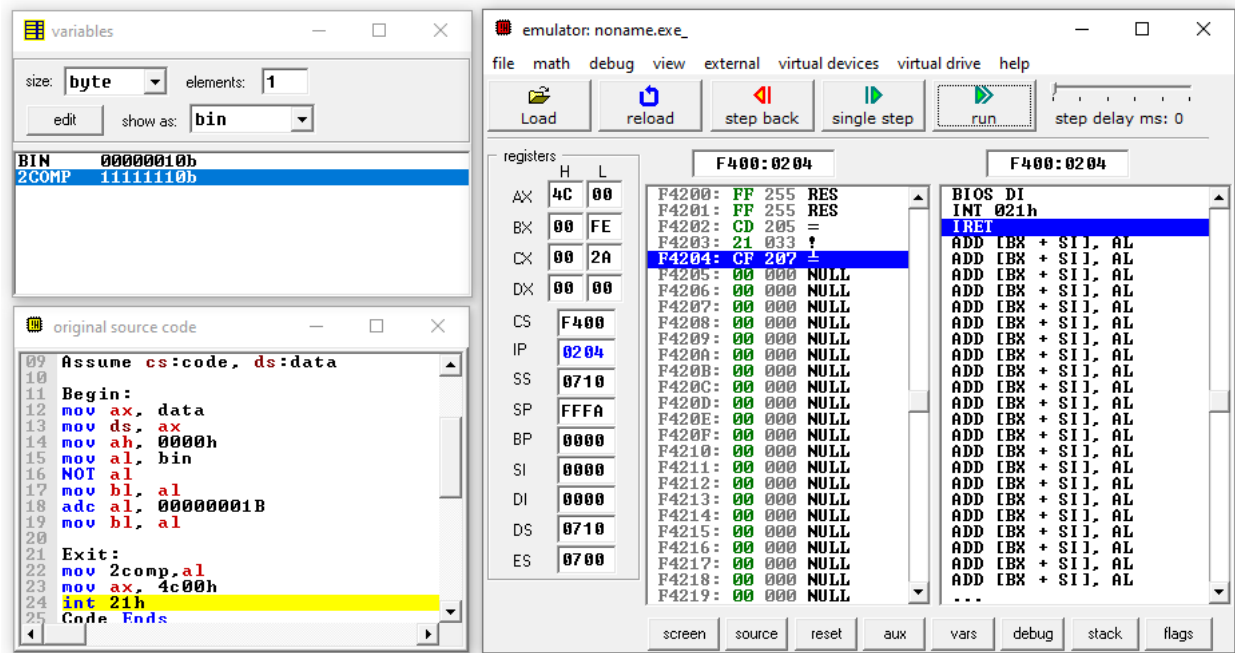


## Variables

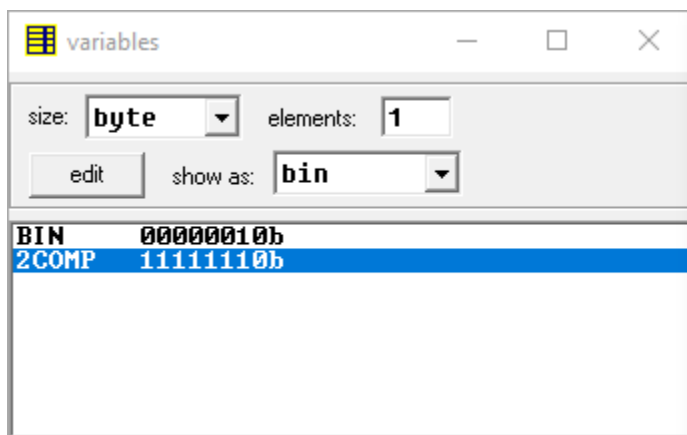


## OUTPUT

After Exe



Variables:



8 B)BCD TO BINARY

Lab DA6 → Experiment7 → Question2

## 8 C)BINARY TO BCD

**Lab DA6 →Experiment7 → Question1**

## 9) FACTORIAL OF NUMBER WHOSE INPUT IS IN MEMORY

**Lab DA5 →Exp 6→ Question2 → Part1**

**(Change Memory Location according to  
Question)**

## **10) LARGEST AND SMALLEST NUMBERS IN ARRAY**

### **CODE**

**data segment**

**array db 67h,25h,14h,02h**

**size db 04h**

**small db 0h**

**large db 0h**

**data ends**

**code segment**

**start:**

**mov ax,data**

**mov ds,ax**

**mov cl, size**

**mov si , offset array**

**mov al,[si]**

**dec cl**

```
up1:inc si
      cmp al,[si]
      jc dn1
      mov al,[si]
      jc dn1
dn1:dec cl
      jnz up1
      mov cl, size
      mov si , offset array
      mov bl,[si]
      dec cl
up2:inc si
      cmp [si], bl
      jc dn2
      mov bl,[si]
      jc dn2
dn2:dec cl
      jnz up2
```

**mov small ,al**

**mov large, bl**

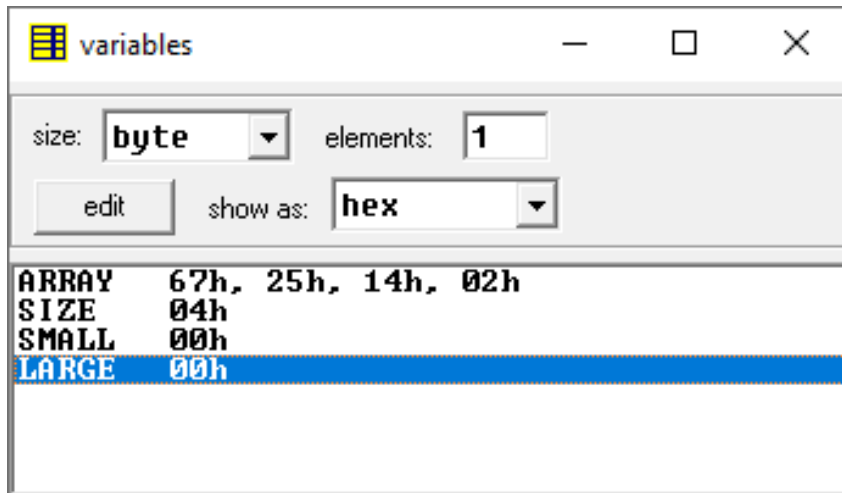
**HLT**

**code ends**

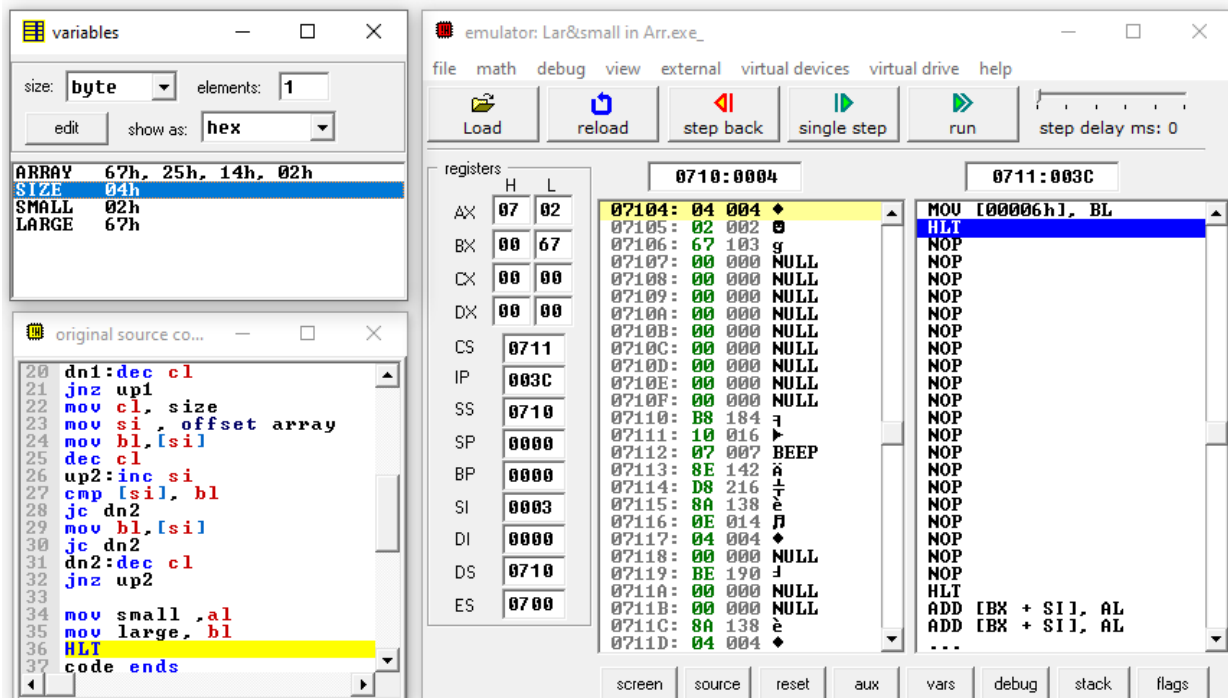
**end start**

```
01 data segment
02 array db 67h,25h,14h,02h
03 size db 04h
04 small db 0h
05 large db 0h
06 data ends
07 code segment
08 start:
09     mov ax,data
10     mov ds,ax
11     mov cl, size
12     mov si, offset array
13     mov al,[si]
14     dec cl
15 up1:inc si
16     cmp al,[si]
17     jc dn1
18     mov al,[si]
19     jc dn1
20 dn1:dec cl
21     jnz up1
22     mov cl, size
23     mov si, offset array
24     mov bl,[si]
25     dec cl
26 up2:inc si
27     cmp [si], bl
28     jc dn2
29     mov bl,[si]
30     jc dn2
31 dn2:dec cl
32     jnz up2
33
34     mov small ,al
35     mov large, bl
36     HLT
37 code ends
38 end start
```

# Variables Before Execution

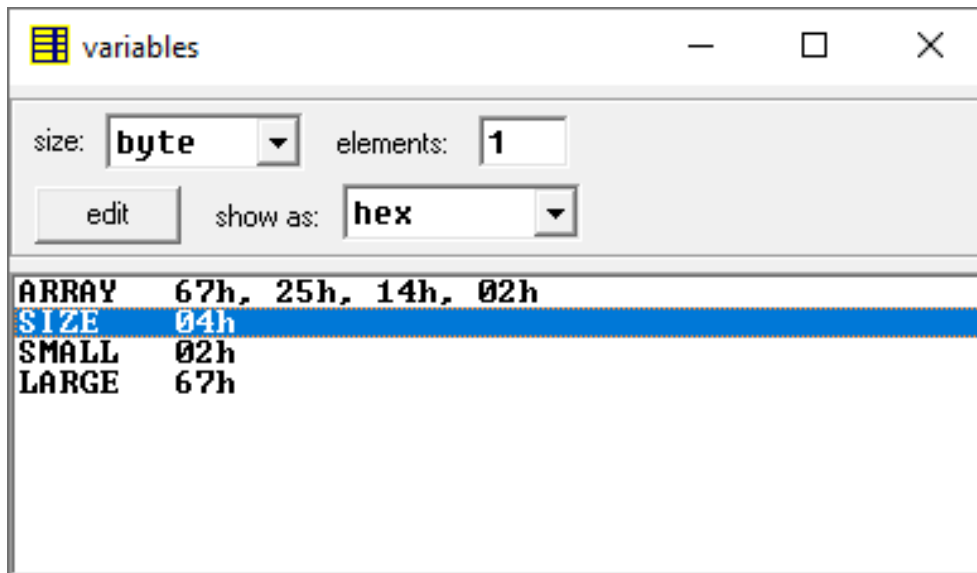


# OUTPUT



# Variables After Execution





11)

## A) CALCULATE LENGTH OF STRING CODE

**;macro for printing a string**

**print macro m**

**mov ah,09h**

**mov dx,offset m**

**int 21h**

**endm**

**.model small**

**.data**

**msg db 10,13, "Exiting the program \$"**

**empty db 10,13, " \$"**

**str1 db 25,?,25 dup('\$')**

**len db ?**

**mstring db 10,13, "Enter the string: \$"**

**mlength db 10,13, "Length is: \$"**

**;\*\*\*\*\* Code Segment \*\*\*\*\***

**.code**

**start:**

**mov ax,@data**

**mov ds,ax**

**print mstring**

**call accept\_string ;function call to accept a string**

**mov cl,str1+1 ;storing length in cl from first byte of the array**

**mov bl,cl ;copying in bl for displaying**

**print mlength**

**call display1 ;printing the length**

**exit:**

**mov ah,4ch ;exit the program**

**int 21h**

**;accept procedure**

**accept proc near**

**mov ah,01**

**int 21h**

**ret**

**accept endp**

**display1 proc near**

**mov al,bl**

**mov bl,al**

**and al,0f0h**

**mov cl,04**

**rol al,cl**

**cmp al,09**

**jbe number**

**add al,07**

**number: add al,30h**

**mov dl,al**

**mov ah,02**

**int 21h**

**mov al,bl**

**and al,00fh**

**cmp al,09**

**jbe number2**

**add al,07**

**number2: add al,30h**

**mov dl,al**

**mov ah,02**

**int 21h**

**ret**

**display1 endp**

**accept\_string proc near**

**mov ah,0ah ;accept string from user function**

**mov dx,offset str1 ; store the string in memory  
pointed by "DX"**

**int 21h**

**ret**

**accept\_string endp**

**end start**

**end**

```

01  %macro for printing a string
02  print %macro m
03  mov ah,09h
04  mov dx,offset m
05  int 21h
06  endm
07
08  .model small
09
10  .data
11
12  msg db 10,13, "Exiting the program $"
13
14  empty db 10,13, " $"
15  str1 db 25,?,25 dup('$')
16
17  len db ?
18  mstring db 10,13, "Enter the string: $"
19  mlength db 10,13, "Length is: $"
20
21
22
23  ;***** Code Segment *****
24
25  .code
26
27  start:
28      mov ax,@data
29      mov ds,ax
30
31      print mstring
32      call accept_string    ;function call to accept a string
33
34      mov cl,str1+1         ;storing length in cl from first byte of the array
35      mov bl,cl             ;copying in bl for displaying
36      print mlength
37      call display1         ;printing the length
38
39  exit:
40  mov ah,4ch               ;exit the program
41  int 21h
42
43
44  ;accept procedure
45
46  accept proc near
47
48  mov ah,01
49  int 21h
50  ret
51  accept endp
52
53  display1 proc near
54
55      mov al,bl
56      mov bl,al
57      and al,0f0h
58      mov cl,04
59      rol al,cl
60
61      cmp al,09
62      jbe number
63      add al,07
64  number:  add al,30h
65           mov dl,al
66           mov ah,02
67           int 21h
68
69           mov al,bl
70           and al,00fh
71           cmp al,09
72           jbe number2
73           add al,07
74  number2: add al,30h
75           mov dl,al
76           mov ah,02
77           int 21h
78  ret
79  display1 endp
80
81
82  accept_string proc near
83
84
85  mov ah,0ah               ;accept string from user function
86  mov dx,offset str1       ; store the string in memory pointed by "DX"
87  int 21h
88  ret
89  accept_string endp
90
91  end start
92  end

```

## Variables before Execution

output



## **B) COUNT OF SPACES IN STRING**

### **CODE**

**;macro for printing a string**

**print macro m**

**mov ah,09h**

**mov dx,offset m**

**int 21h**

**endm**

**.model small**

**;\*\*\*\*\* Data Segment \*\*\*\*\***

**.data**



```
empty db 10,13, "  $"
str1 db 25,?,25 dup('$')
mstring db 10,13, "Enter the string: $"
mscount db 10,13, "Number of spaces: $"
mlength db 10,13, "Length is: $"
scount db ?
```

```
;***** Code Segment *****
```

```
.code
```

```
start:
```

```
mov ax,@data
```

```
mov ds,ax
```

```
    print mstring
```

```
    call accept_string
```

**mov si,offset str1+2 ;position si to start of the string**

**mov cl,str1+1 ;copy length in cl**

**mov dh,00 ;counter to store number of spaces**

**cmpagain1: mov al,[si] ;copy content at memory location "si" in "al"**

**cmp al,' ' ;compare "al" with space**

**jne below ;if not equal jump to label "below"**

**inc dh**

**below: inc si ;move to next character**

**dec cl ;decrement string length counter**

**jnz cmpagain1 ;if not zero check again**

**mov scout,dh ;save the count in memory location "scout"**

**mov bl,scount ;copy count to "bl" for  
printing**

**print mscount**

**call display1**

**exit:**

**mov ah,4ch ;exit the program**

**int 21h**

**;accept procedure**

**accept proc near**

**mov ah,01**

**int 21h**

**ret**

**accept endp**

**display1 proc near**

**mov al,bl  
mov bl,al  
and al,0f0h  
mov cl,04  
rol al,cl**

**cmp al,09  
jbe number  
add al,07**

**number: add al,30h  
mov dl,al  
mov ah,02  
int 21h**

**mov al,bl  
and al,00fh  
cmp al,09  
jbe number2**

```
    add al,07
number2: add al,30h
    mov dl,al
    mov ah,02
    int 21h
ret
display1 endp
```

```
accept_string proc near
```

```
mov ah,0ah      ;accept string from user
function
mov dx,offset str1 ; store the string in memory
pointed by "DX"
int 21h
ret
accept_string endp
```

end start

end

```
0001 ;macro for printing a string
0002 print macro m
0003     mov ah,09h
0004     mov dx,offset m
0005     int 21h
0006     endm
0007
0008 .model small
0009
0010 ;***** Data Segment *****
0011 .data
0012
0013 empty db 10,13, "  $"
0014 str1 db 25,?,25 dup('<')
0015 mstring db 10,13, "Enter the string: $"
0016 mscount db 10,13, "Number of spaces: $"
0017 mlength db 10,13, "Length is: $"
0018 scount db ?
0019
0020
0021 ;***** Code Segment *****
0022
0023 .code
0024
0025 start:
0026     mov ax,@data
0027     mov ds,ax
0028
0029     print mstring
0030     call accept_string
0031     mov si,offset str1+2 ;position si to start of the string
0032
0033     mov cl,str1+1 ;copy length in cl
0034     mov dh,00 ;counter to store number of spaces
0035     cmpagain1: mov al,[si] ;copy content at memory location "si" in "al"
0036                 cmp al,' ' ;compare "al" with space
0037                 jne below ;if not equal jump to label "below"
0038                 inc dh
0039
0040     below: inc si ;move to next character
0041            dec cl ;decrement string length counter
0042            jnz cmpagain1 ;if not zero check again
0043
0044     mov scount,dh ;save the count in memory location "scount"
0045     mov bl,scount ;copy count to "bl" for printing
0046     print mscount
0047     call display1
0048
0049
0050
```

```

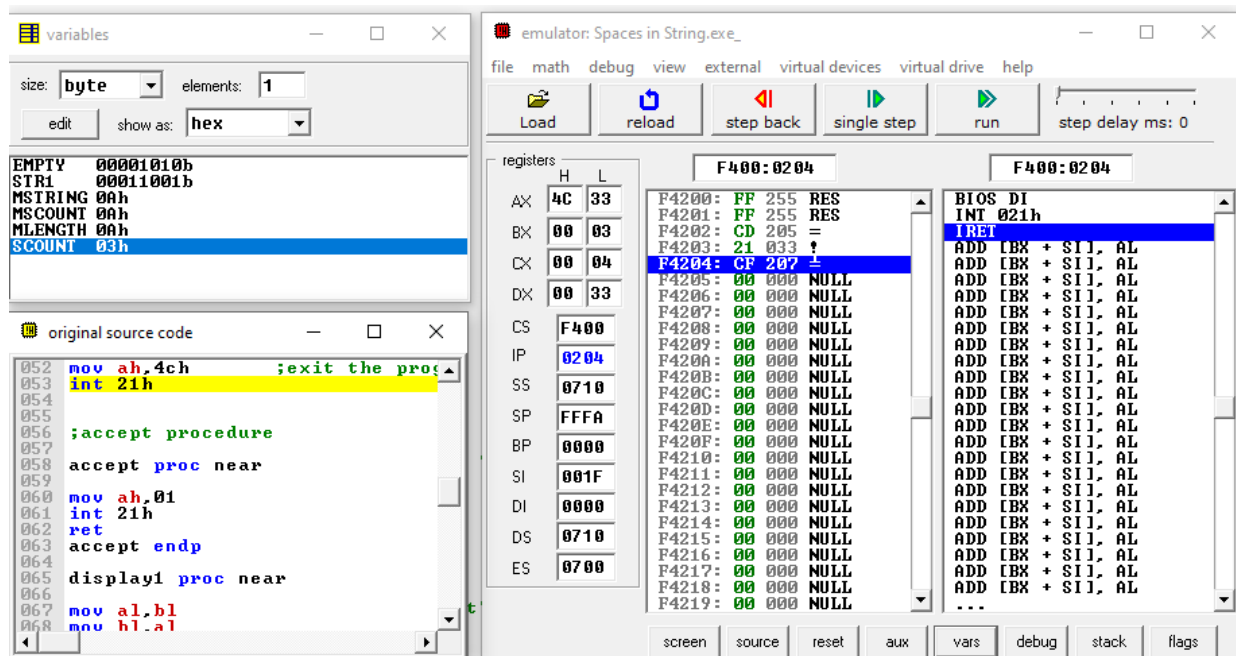
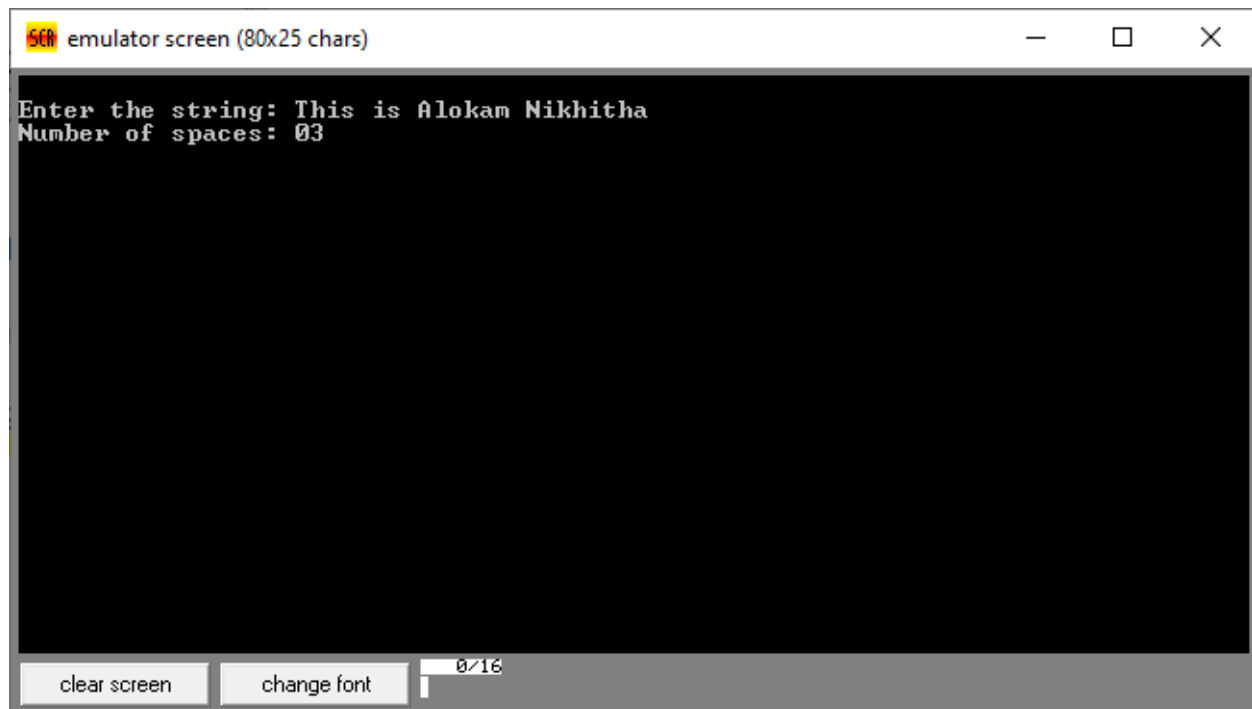
051 exit:
052 mov ah,4ch          ;exit the program
053 int 21h
054 ;accept procedure
055
056 accept proc near
057
058 mov ah,01
059 int 21h
060 ret
061 accept endp
062
063 display1 proc near
064
065     mov al,b1
066     mov bl,al
067     and al,0f0h
068     mov cl,04
069     rol al,cl
070
071     cmp al,09
072     jbe number
073     add al,07
074 number: add al,30h
075         mov dl,al
076         mov ah,02
077         int 21h
078
079         mov al,bl
080         and al,00fh
081         cmp al,09
082         jbe number2
083         add al,07
084 number2: add al,30h
085         mov dl,al
086         mov ah,02
087         int 21h
088 ret
089 display1 endp
090
091 ;accept_string proc near
092
093 mov ah,0ah          ;accept string from user function
094 mov dx,offset str1  ; store the string in memory pointed by "DX"
095 int 21h
096 ret
097 accept_string endp
098
099 end start
100 end

```

## Variables before Execution

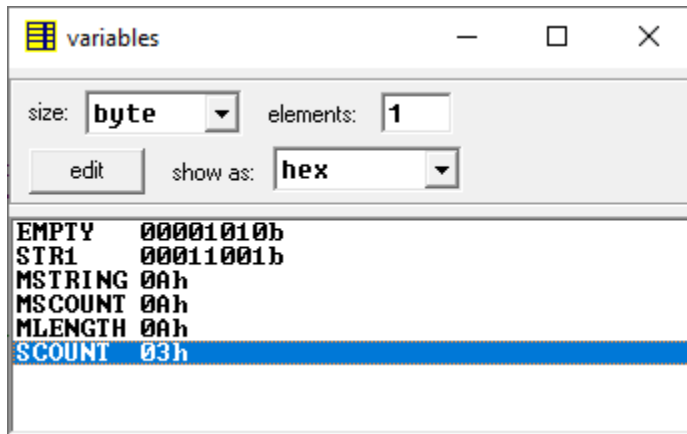
variables	
size:	byte
elements:	1
edit	show as: hex
EMPTY	00001010b
STR1	00011001b
MSTRING	0Ah
MSCOUNT	0Ah
MLENGTH	0Ah
SCOUNT	00h

## Output





## Variables after Execution



## Result in Scount

### C) REVERSE STRING AND PRINT IT

**;macro for printing a string**

**print macro m**

**mov ah,09h**

**mov dx,offset m**

**int 21h**

**endm**

**.model small**

**;\*\*\*\*\* Data Segment \*\*\*\*\***

**.data**

**empty db 10,13, " \$"**

**str1 db 25,?,25 dup('\$')**

**str2 db 25,?,25 dup('\$')**

**mstring db 10,13, "Enter the string: \$"**

**mreverse db 10,13, "Reversed string: \$"**

**;\*\*\*\*\* Code Segment \*\*\*\*\***

**.code**

**start:**

**mov ax,@data**

**mov ds,ax**

**print mstring**

**call accept\_string**

**mov si,offset str1     ;point si to start of  
string1**

**mov di,offset str2     ;point di to start of  
string2**

**mov al,[si]           ;copy first two locations of  
string1 to string2**

**mov [di],al           ;since these contain the size  
and length of the string**

**inc si               ;which are same in reverse  
string also**

**inc di**

**mov al,[si]**

**mov [di],al**

**inc si**

**inc di**

**mov cl,str1+1 ; copy length in cl**

**mov ch,00**

**add si,cx ;add length of string1 to si  
to move it to last location**

**dec si ;si at last location of string1**

**move\_more: mov al,[si] ;copying character  
one by one from string1 pointed by si**

**mov [di],al ; to string2 pointed by "di" in  
reverse order as si moves**

**dec si ; from last character to first  
character**

**inc di**

**dec cl**

**jnz move\_more**

**print mreverse**

**print str2+2 ; printing the reversed string**  
**print empty**

**exit:**

**mov ah,4ch ;exit the program**

**int 21h**

**;accept procedure**

**accept proc near**

**mov ah,01**

**int 21h**

**ret**

**accept endp**

**display1 proc near**

**mov al,bl  
mov bl,al  
and al,0f0h  
mov cl,04  
rol al,cl**

**cmp al,09  
jbe number  
add al,07**

**number: add al,30h  
mov dl,al  
mov ah,02  
int 21h**

**mov al,bl  
and al,00fh  
cmp al,09  
jbe number2  
add al,07**

**number2: add al,30h**

**mov dl,al**

**mov ah,02**

**int 21h**

**ret**

**display1 endp**

**accept\_string proc near**

**mov ah,0ah ;accept string from user  
function**

**mov dx,offset str1 ; store the string in memory  
pointed by "DX"**

**int 21h**

**ret**

**accept\_string endp**

**end start**

# end

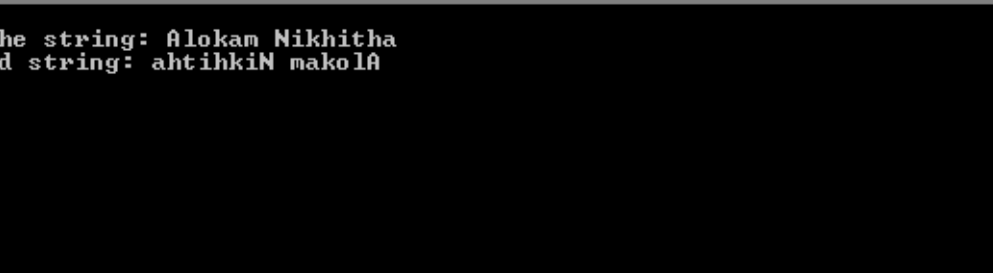
```
0001 ;macro for printing a string
0002 print macro m
0003     mov ah,09h
0004     mov dx,offset m
0005     int 21h
0006 endm
0007
0008 .model small
0009
0010 ;***** Data Segment *****
0011 .data
0012
0013 empty db 10,13, "  $"
0014 str1 db 25,?,25 dup('$')
0015 str2 db 25,?,25 dup('$')
0016 mstring db 10,13, "Enter the string: $"
0017 mreverse db 10,13, "Reversed string: $"
0018
0019 ;***** Code Segment *****
0020 .code
0021
0022 start:
0023     mov ax,@data
0024     mov ds,ax
0025
0026     print mstring
0027     call accept_string
0028
0029     mov si,offset str1      ;point si to start of string1
0030     mov di,offset str2      ;point di to start of string2
0031
0032     mov al,[si]             ;copy first two locations of string1 to string2
0033     mov [di],al             ;since these contain the size and length of the string
0034     inc si                  ;which are same in reverse string also
0035     inc di
0036
0037     mov al,[si]
0038     mov [di],al
0039     inc si
0040     inc di
0041
0042     mov al,[si]
0043     mov [di],al
0044     inc si
0045     inc di
0046
0047     mov cl,str1+1 ;      copy length in cl
0048     mov ch,00h
0049     add si,cx      ;add length of string1 to si to move it to last location
0050     dec si         ;at last location of string1
0051     ; from last character to first character
0052     dec si
0053     inc di
0054     dec cl
0055     jnz move_more
0056     print mreverse
0057     print str2+2    ; printing the reversed string
0058     print empty
0059
0060 exit:
0061     mov ah,4ch      ;exit the program
0062     int 21h
0063
0064 ;accept procedure
0065 accept proc near
0066     mov ah,01
0067     int 21h
0068     ret
0069
0070 accept endp
0071 display1 proc near
0072
0073     mov al,b1
0074     mov bl,al
0075     and al,0f0h
0076     mov cl,04
0077     rol al,cl
0078
0079     cmp al,09
0080     jbe number
0081     add al,07
0082
0083 number: add al,30h
0084         mov dl,al
0085         mov ah,02
0086         int 21h
0087         mov al,b1
0088         and al,00fh
0089         cmp al,09
0090         jbe number2
0091         add al,07
0092
0093 number2: add al,30h
0094         mov dl,al
0095         mov ah,02
0096         int 21h
0097     ret
0098 display1 endp
0099 accept_string proc near
0100     mov ah,0ah      ;accept string from user function
0101     mov dx,offset str1 ; store the string in memory pointed by "DX"
0102     int 21h
0103     ret
0104 accept_string endp
0105
```



## Output

The screenshot shows the Immunity Debugger interface with the following components:

- Variables Window:** Shows a single variable 'byte' with 1 element, displayed in hex.
- Assembly View:** Displays the assembly code for the 'mreverse' function. The function starts with 'add si, cx' and 'dec si', followed by a loop 'move\_more' that moves bytes from 'si' to 'di' until 'cl' reaches zero. It then prints the reversed string and exits with 'mov ah, 4ch' and 'int 21h'.
- Registers Window:** Shows the current state of registers. The 'F400:0204' register is selected, showing its value as 'CF 207 1'.
- Memory View (F400:0204):** Shows the memory contents at the selected address, including 'BIOS DI' and 'INT 021h'.



emulator screen (80x25 chars)

Enter the string: Alokam Nikhitha  
Reversed string: ahtihkiN makola

clear screen change font 0/16