FEBRUARY 18, 2018

LAB EXERCISE 2

PROF. JIMMY MATHEW | CSE 2006 MPI B1 SLOT | 16BCE0783 DAKSH

Question 1:

Find smallest number and largest number from an array of hexadecimal numbers. Store the results in DL and DH registers respectively.

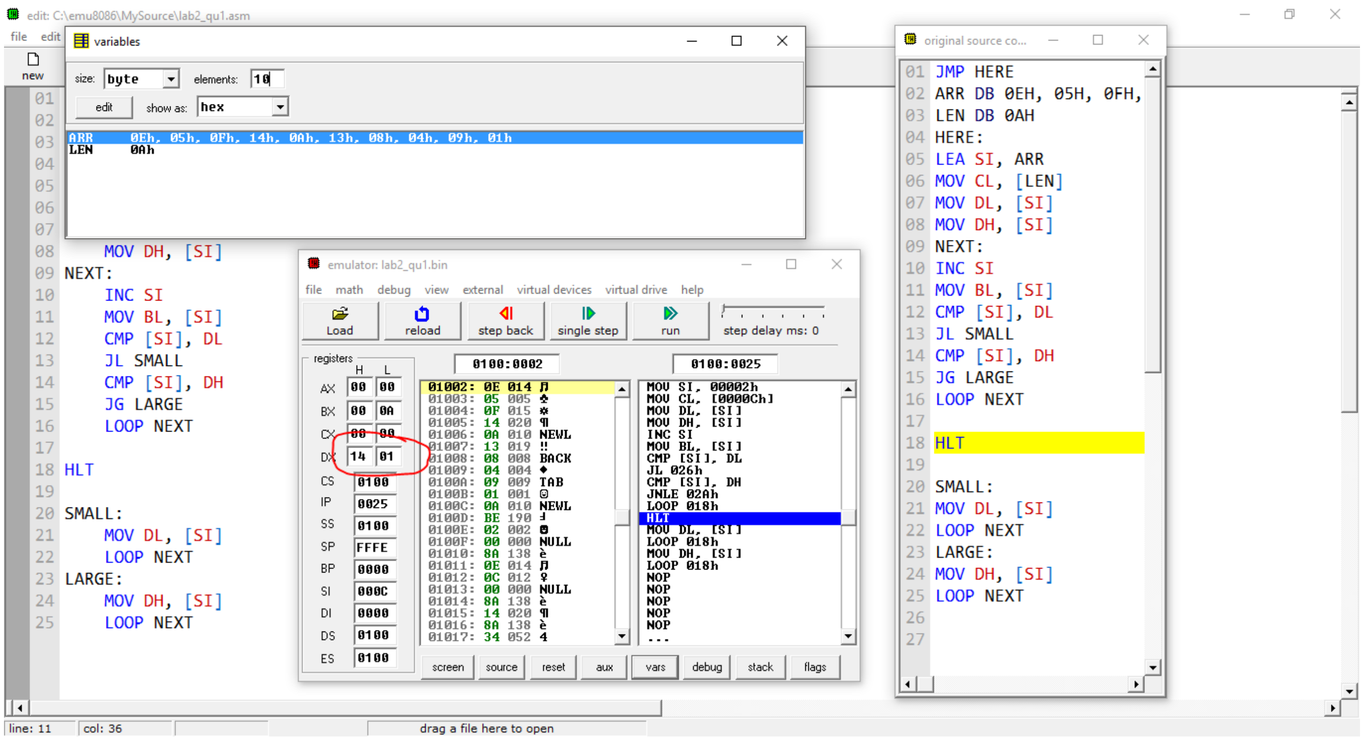
Algorithm:

The basic algorithm for finding largest and smallest. Making the first element as both largest and smallest. Then traversing along the array and comparing with both the numbers in the smallest and largest variable. If found smallest or largest, then changing the current element to be the smallest or largest respectively.

Code:

```
JMP HERE
    ARR DB 0EH, 05H, 0FH, 014H, 0AH, 013H, 08H, 04H, 09H, 01H
    LEN DB 0AH
HERE:
    LEA SI, ARR
    MOV CL, [LEN]
    MOV DL, [SI]
    MOV DH, [SI]
NEXT:
    INC SI
    MOV BL, [SI]

    CMP [SI], DL

    JL SMALL

    CMP [SI], DH

    JG LARGE

    LOOP NEXT


HLT


SMALL:

    MOV DL, [SI]

    LOOP NEXT

LARGE:

    MOV DH, [SI]

    LOOP NEXT
```

Screenshots (DH and DL Registers are Circled in Red):



edit: C:\emu8086\MySource\lab2_qu1.asm

```
    MOV DH, [SI]
NEXT:
    INC SI
    MOV BL, [SI]
    CMP [SI], DL
    JL SMALL
    CMP [SI], DH
    JG LARGE
    LOOP NEXT

HLT

SMALL:
    MOV DL, [SI]
    LOOP NEXT
LARGE:
    MOV DH, [SI]
    LOOP NEXT
```

variables — size: byte, elements: 10, show as: hex

```
ARR     0Eh, 05h, 0Fh, 14h, 0Ah, 13h, 08h, 04h, 09h, 01h
LEN     0Ah
```

emulator: lab2_qu1.bin — step delay ms: 0

registers	H	L
AX	00	00
BX	00	0A
CX	00	00
DX	14	01
CS	0100	
IP	0025	
SS	0100	
SP	FFFE	
BP	0000	
SI	000C	
DI	0000	
DS	0100	
ES	0100	

```
0100:0002
01002: 0E 014 ♫
01003: 05 005 ♣
01004: 0F 015 ☼
01005: 14 020 ¶
01006: 0A 010 NEWL
01007: 13 019 ‼
01008: 08 008 BACK
01009: 04 004 ♦
0100A: 09 009 TAB
0100B: 01 001 ☺
0100C: 0A 010 NEWL
0100D: BE 190 ╛
0100E: 02 002 ☻
0100F: 00 000 NULL
01010: 8A 138 è
01011: 0E 014 ♫
01012: 0C 012 ♀
01013: 00 000 NULL
01014: 8A 138 è
01015: 14 020 ¶
01016: 8A 138 è
01017: 34 052 4
```

```
0100:0025
MOV SI, 00002h
MOV CL, [0000Ch]
MOV DL, [SI]
MOV DH, [SI]
INC SI
MOV BL, [SI]
CMP [SI], DL
JL 026h
CMP [SI], DH
JNLE 02Ah
LOOP 018h
HLT
MOV DL, [SI]
LOOP 018h
MOV DH, [SI]
LOOP 018h
NOP
NOP
NOP
NOP
NOP
...
```

original source code

```
JMP HERE
ARR DB 0EH, 05H, 0FH,
LEN DB 0AH
HERE:
LEA SI, ARR
MOV CL, [LEN]
MOV DL, [SI]
MOV DH, [SI]
NEXT:
INC SI
MOV BL, [SI]
CMP [SI], DL
JL SMALL
CMP [SI], DH
JG LARGE
LOOP NEXT

HLT

SMALL:
MOV DL, [SI]
LOOP NEXT
LARGE:
MOV DH, [SI]
LOOP NEXT
```

line: 11 col: 36 drag a file here to open

Question 2:

Extract vowels from a given string, and store it in a separate array.

Algorithm:

Traversing the string till you reach the '$' sign (marking the end of the string) and comparing each character with all the vowels (both capital and small). If the character matches the vowels, moving it to another array (using the SI Pointer).

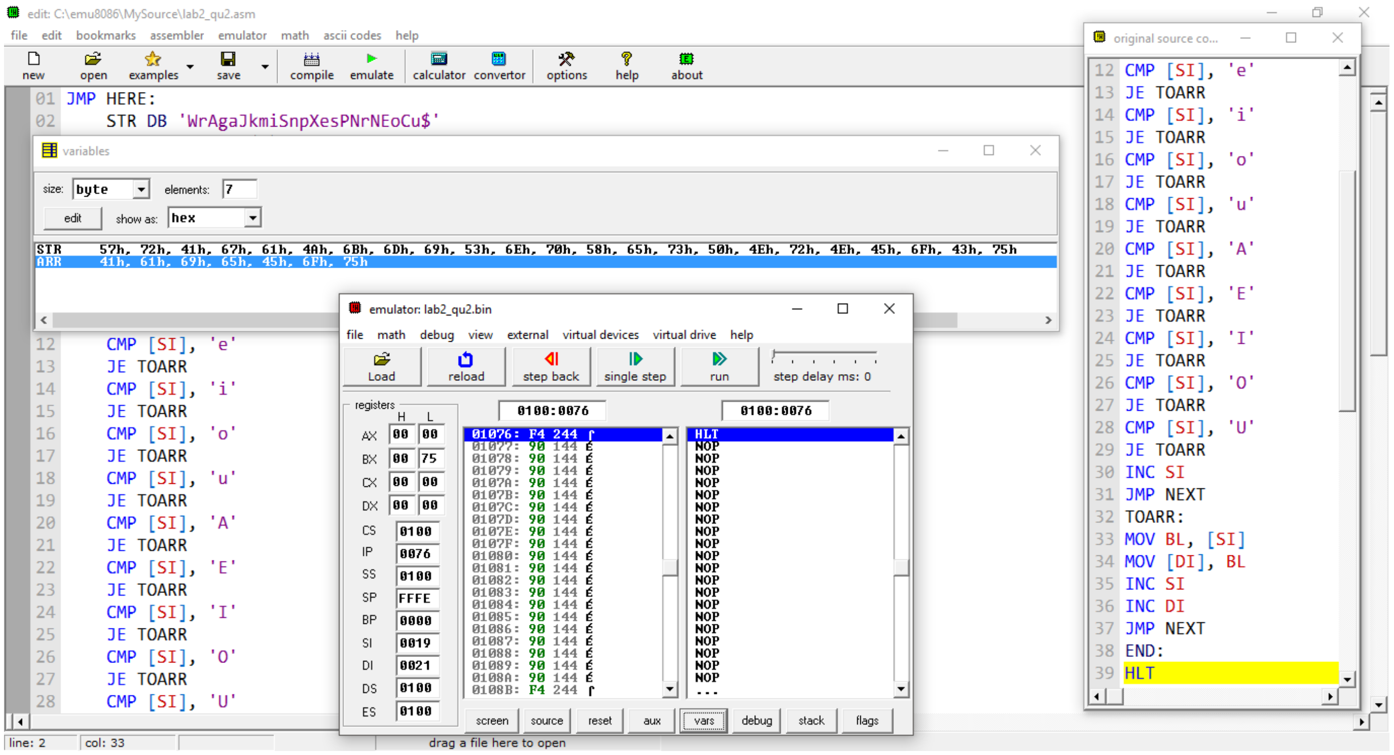
Code:

```
JMP HERE:

    STR DB 'WrAgaJkmiSnpXesPNrNEoCu$'

    ARR DB 20 dup (?)
```

```
HERE:

    LEA SI, STR

    LEA DI, ARR

NEXT:

    CMP [SI], '$'

    JE END

    CMP [SI], 'a'

    JE TOARR

    CMP [SI], 'e'

    JE TOARR

    CMP [SI], 'i'

    JE TOARR

    CMP [SI], 'o'

    JE TOARR

    CMP [SI], 'u'

    JE TOARR

    CMP [SI], 'A'

    JE TOARR

    CMP [SI], 'E'

    JE TOARR

    CMP [SI], 'I'

    JE TOARR

    CMP [SI], 'O'

    JE TOARR

    CMP [SI], 'U'

    JE TOARR
```

```
        INC SI
        JMP NEXT
TOARR:
        MOV BL, [SI]
        MOV [DI], BL
        INC SI
        INC DI
        JMP NEXT
END:
        HLT
```

Screenshots (STR is the original string and ARR consists of extracted vowels):



edit: C:\emu8086\MySource\lab2_qu2.asm

```
JMP HERE:
    STR DB 'WrAgaJkmiSnpXesPNrNEoCu$'
```

variables — size: byte, elements: 7, show as: hex

```
STR     57h, 72h, 41h, 67h, 61h, 4Ah, 6Bh, 6Dh, 69h, 53h, 6Eh, 70h, 58h, 65h, 73h, 50h, 4Eh, 72h, 4Eh, 45h, 6Fh, 43h, 75h
ARR     41h, 61h, 69h, 65h, 45h, 6Fh, 75h
```

```
    CMP [SI], 'e'
    JE TOARR
    CMP [SI], 'i'
    JE TOARR
    CMP [SI], 'o'
    JE TOARR
    CMP [SI], 'u'
    JE TOARR
    CMP [SI], 'A'
    JE TOARR
    CMP [SI], 'E'
    JE TOARR
    CMP [SI], 'I'
    JE TOARR
    CMP [SI], 'O'
    JE TOARR
    CMP [SI], 'U'
```

emulator: lab2_qu2.bin

```
registers   H   L
AX         00  00
BX         00  75
CX         00  00
DX         00  00
CS         0100
IP         0076
SS         0100
SP         FFFE
BP         0000
SI         0019
DI         0021
DS         0100
ES         0100
```

```
0100:0076                    0100:0076
01076: F4 244 ⌠              HLT
01077: 90 144 É              NOP
01078: 90 144 É              NOP
01079: 90 144 É              NOP
0107A: 90 144 É              NOP
0107B: 90 144 É              NOP
0107C: 90 144 É              NOP
0107D: 90 144 É              NOP
0107E: 90 144 É              NOP
0107F: 90 144 É              NOP
01080: 90 144 É              NOP
01081: 90 144 É              NOP
01082: 90 144 É              NOP
01083: 90 144 É              NOP
01084: 90 144 É              NOP
01085: 90 144 É              NOP
01086: 90 144 É              NOP
01087: 90 144 É              NOP
01088: 90 144 É              NOP
01089: 90 144 É              NOP
0108A: 90 144 É              NOP
0108B: F4 244 ⌠              ...
```

original source code

```
CMP [SI], 'e'
JE TOARR
CMP [SI], 'i'
JE TOARR
CMP [SI], 'o'
JE TOARR
CMP [SI], 'u'
JE TOARR
CMP [SI], 'A'
JE TOARR
CMP [SI], 'E'
JE TOARR
CMP [SI], 'I'
JE TOARR
CMP [SI], 'O'
JE TOARR
CMP [SI], 'U'
JE TOARR
INC SI
JMP NEXT
TOARR:
MOV BL, [SI]
MOV [DI], BL
INC SI
INC DI
JMP NEXT
END:
HLT
```

line: 2   col: 33   drag a file here to open

Question 3:

Sort the given number array in ascending order.

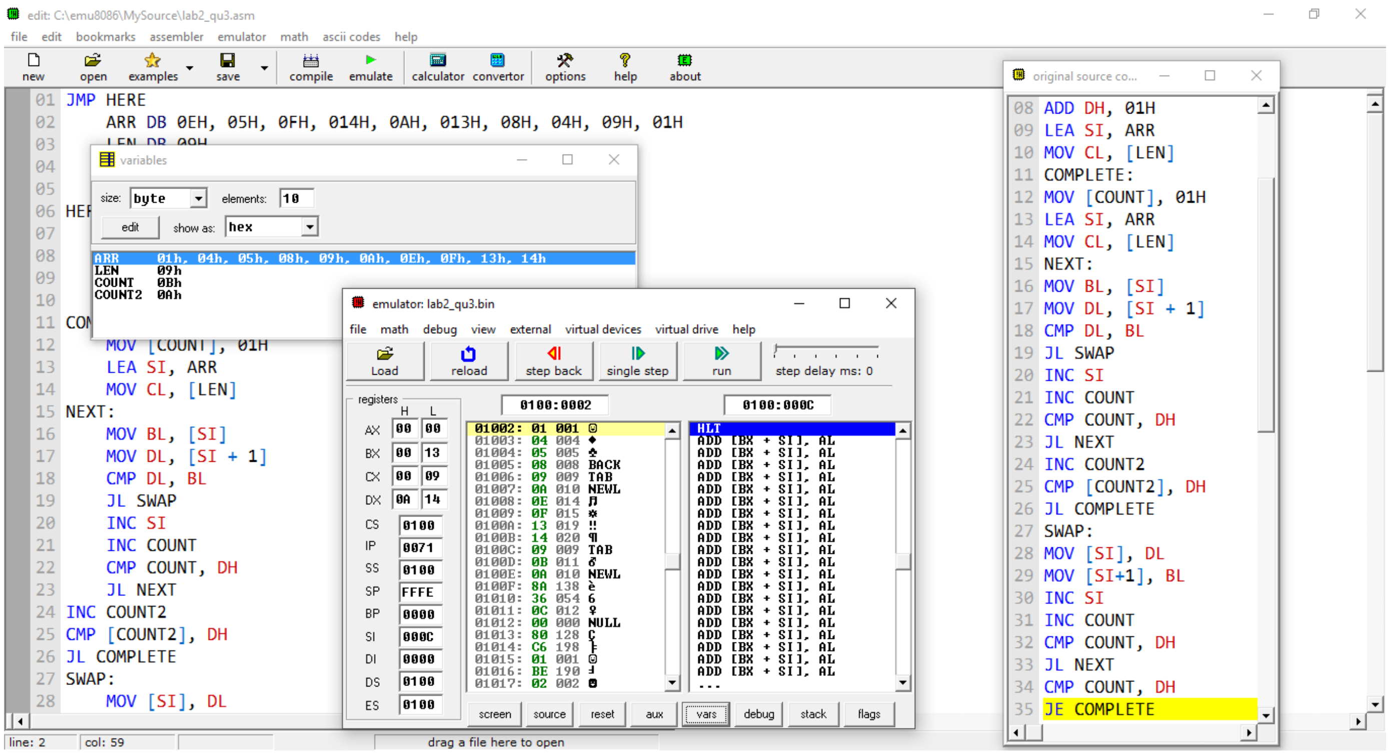
Algorithm:

I am not using another array. Sorting algorithm used is **Bubble Sort**. Two Pointers on adjacent elements. Traversing both the pointers through the array simultaneously multiple times (the length of the array - 1) and swapping if the number on the right is less than the number on the left.

Code:

```
JMP HERE

    ARR DB 0EH, 05H, 0FH, 014H, 0AH, 013H, 08H, 04H, 09H, 01H

    LEN DB 09H

    COUNT DB 01H

    COUNT2 DB 01H

HERE:

    MOV DH, [LEN]

    ADD DH, 01H

    LEA SI, ARR

    MOV CL, [LEN]

COMPLETE:

    MOV [COUNT], 01H

    LEA SI, ARR

    MOV CL, [LEN]

NEXT:
    MOV BL, [SI]

    MOV DL, [SI + 1]

    CMP DL, BL

    JL SWAP

    INC SI

    INC COUNT

    CMP COUNT, DH

    JL NEXT

INC COUNT2

CMP [COUNT2], DH

JL COMPLETE

SWAP:

    MOV [SI], DL

    MOV [SI+1], BL

    INC SI

    INC COUNT

    CMP COUNT, DH

    JL NEXT

    CMP COUNT, DH

    JE COMPLETE
```

Screenshots (The initial ARR and after sort ARR can be seen in the screenshot):


edit: C:\emu8086\MySource\lab2_qu3.asm

```
JMP HERE
    ARR DB 0EH, 05H, 0FH, 014H, 0AH, 013H, 08H, 04H, 09H, 01H
    LEN DB 09H

HEF

CON
    MOV [COUNT], 01H
    LEA SI, ARR
    MOV CL, [LEN]
NEXT:
    MOV BL, [SI]
    MOV DL, [SI + 1]
    CMP DL, BL
    JL SWAP
    INC SI
    INC COUNT
    CMP COUNT, DH
    JL NEXT
INC COUNT2
CMP [COUNT2], DH
JL COMPLETE
SWAP:
    MOV [SI], DL
```

line: 2   col: 59

variables — size: byte   elements: 10   show as: hex

```
ARR     01h, 04h, 05h, 08h, 09h, 0Ah, 0Eh, 0Fh, 13h, 14h
LEN     09h
COUNT   0Bh
COUNT2  0Ah
```

emulator: lab2_qu3.bin — step delay ms: 0

```
registers
      H   L
AX   00  00
BX   00  13
CX   00  09
DX   0A  14
CS   0100
IP   0071
SS   0100
SP   FFFE
BP   0000
SI   000C
DI   0000
DS   0100
ES   0100
```

```
0100:0002
01002: 01 001 ☺
01003: 04 004 ♦
01004: 05 005 ♣
01005: 08 008 BACK
01006: 09 009 TAB
01007: 0A 010 NEWL
01008: 0E 014 ♫
01009: 0F 015 ☼
0100A: 13 019 ‼
0100B: 14 020 ¶
0100C: 09 009 TAB
0100D: 0B 011 ♂
0100E: 0A 010 NEWL
0100F: 8A 138 è
01010: 36 054 6
01011: 0C 012 ♀
01012: 00 000 NULL
01013: 80 128 Ç
01014: C6 198 ╞
01015: 01 001 ☺
01016: BE 190 ╛
01017: 02 002 ☻
```

```
0100:000C
HLT
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

original source co...

```
ADD DH, 01H
LEA SI, ARR
MOV CL, [LEN]
COMPLETE:
MOV [COUNT], 01H
LEA SI, ARR
MOV CL, [LEN]
NEXT:
MOV BL, [SI]
MOV DL, [SI + 1]
CMP DL, BL
JL SWAP
INC SI
INC COUNT
CMP COUNT, DH
JL NEXT
INC COUNT2
CMP [COUNT2], DH
JL COMPLETE
SWAP:
MOV [SI], DL
MOV [SI+1], BL
INC SI
INC COUNT
CMP COUNT, DH
JL NEXT
CMP COUNT, DH
JE COMPLETE
```

Question 4:

Check is a given string is palindrome or not.

Algorithm:

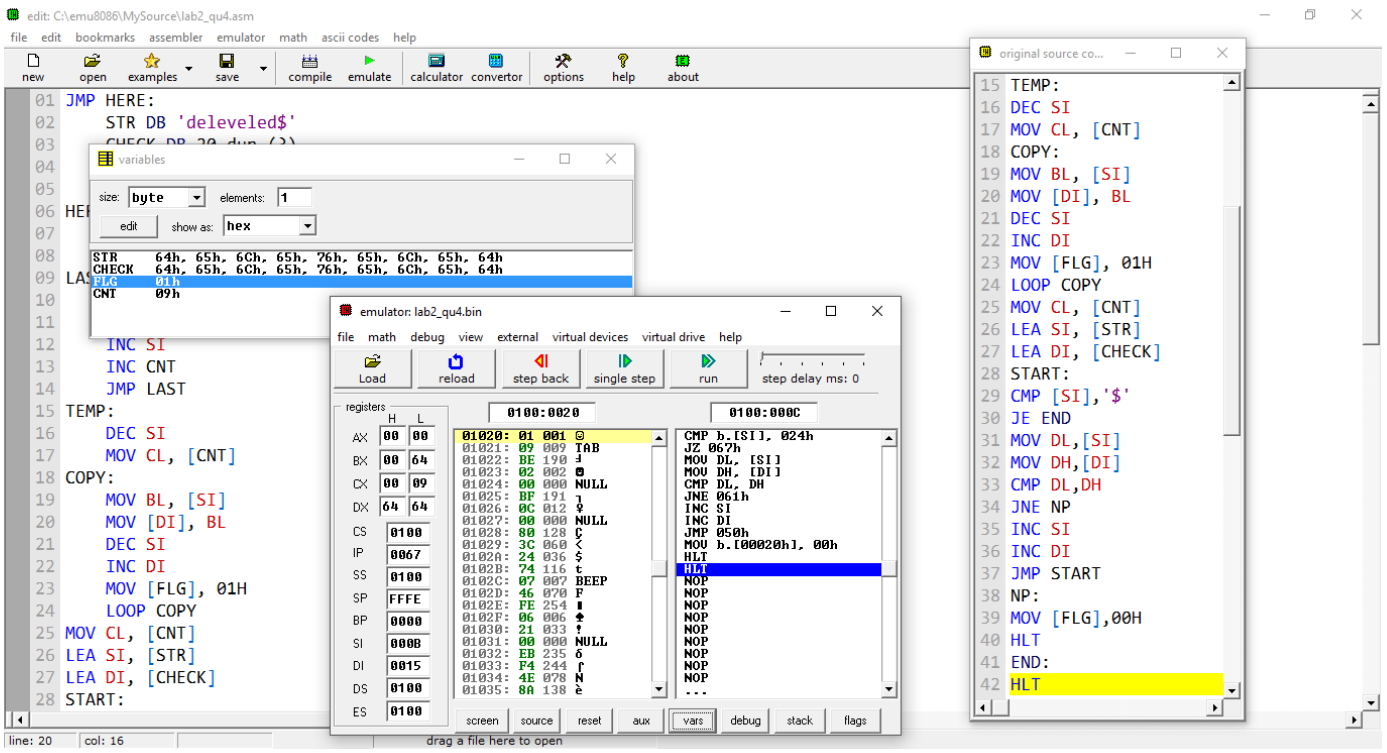
Taking the string and storing it in the reverse order in another array using two pointers (SI and DI). Now comparing both the strings. If comparison (character by character) results in non-equal condition, make the flag 0.

Code:

```
JMP HERE:
    STR DB 'deleveled$'
    CHECK DB 20 dup (?)
    FLG DB 01H
    CNT DB 00H
HERE:
    LEA SI,STR
    LEA DI,CHECK
LAST:
    CMP [SI],'$'
    JE TEMP
    INC SI
    INC CNT
    JMP LAST
TEMP:
    DEC SI
    MOV CL, [CNT]
COPY:
    MOV BL, [SI]
    MOV [DI], BL
    DEC SI
    INC DI
    MOV [FLG], 01H
    LOOP COPY
MOV CL, [CNT]
LEA SI, [STR]
LEA DI, [CHECK]
```

```
START:

    CMP [SI],'$'

    JE END

    MOV DL,[SI]

    MOV DH,[DI]

    CMP DL,DH

    JNE NP

    INC SI

    INC DI

    JMP START

NP:

    MOV [FLG],00H

    HLT

END:

    HLT
```

Screenshots (FLG Variable: 01 – Yes and 00 - No):


edit: C:\emu8086\MySource\lab2_qu4.asm

```
JMP HERE:
    STR DB 'deleveled$'
    CHECK DB 20 dup (?)


HE
    
    
LA


    INC SI
    INC CNT
    JMP LAST
TEMP:
    DEC SI
    MOV CL, [CNT]
COPY:
    MOV BL, [SI]
    MOV [DI], BL
    DEC SI
    INC DI
    MOV [FLG], 01H
    LOOP COPY
MOV CL, [CNT]
LEA SI, [STR]
LEA DI, [CHECK]
START:
```

line: 20   col: 16

variables — size: byte, elements: 1, show as: hex

```
STR     64h, 65h, 6Ch, 65h, 76h, 65h, 6Ch, 65h, 64h
CHECK   64h, 65h, 6Ch, 65h, 76h, 65h, 6Ch, 65h, 64h
FLG     01h
CNT     09h
```

emulator: lab2_qu4.bin — step delay ms: 0

```
registers
     H   L
AX   00  00
BX   00  64
CX   00  09
DX   64  64
CS   0100
IP   0067
SS   0100
SP   FFFE
BP   0000
SI   000B
DI   0015
DS   0100
ES   0100
```

```
0100:0020
01020: 01 001 ☺
01021: 09 009 TAB
01022: BE 190 ╛
01023: 02 002 ☻
01024: 00 000 NULL
01025: BF 191 ┐
01026: 0C 012 ♀
01027: 00 000 NULL
01028: 80 128 Ç
01029: 3C 060 <
0102A: 24 036 $
0102B: 74 116 t
0102C: 07 007 BEEP
0102D: 46 070 F
0102E: FE 254 ■
0102F: 06 006 ♠
01030: 21 033 !
01031: 00 000 NULL
01032: EB 235 δ
01033: F4 244 ⌠
01034: 4E 078 N
01035: 8A 138 è
```

```
0100:000C
CMP b.[SI], 024h
JZ 067h
MOV DL, [SI]
MOV DH, [DI]
CMP DL, DH
JNE 061h
INC SI
INC DI
JMP 050h
MOV b.[00020h], 00h
HLT
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

original source co...

```
TEMP:
DEC SI
MOV CL, [CNT]
COPY:
MOV BL, [SI]
MOV [DI], BL
DEC SI
INC DI
MOV [FLG], 01H
LOOP COPY
MOV CL, [CNT]
LEA SI, [STR]
LEA DI, [CHECK]
START:
CMP [SI],'$'
JE END
MOV DL,[SI]
MOV DH,[DI]
CMP DL,DH
JNE NP
INC SI
INC DI
JMP START
NP:
MOV [FLG],00H
HLT
END:
HLT
```