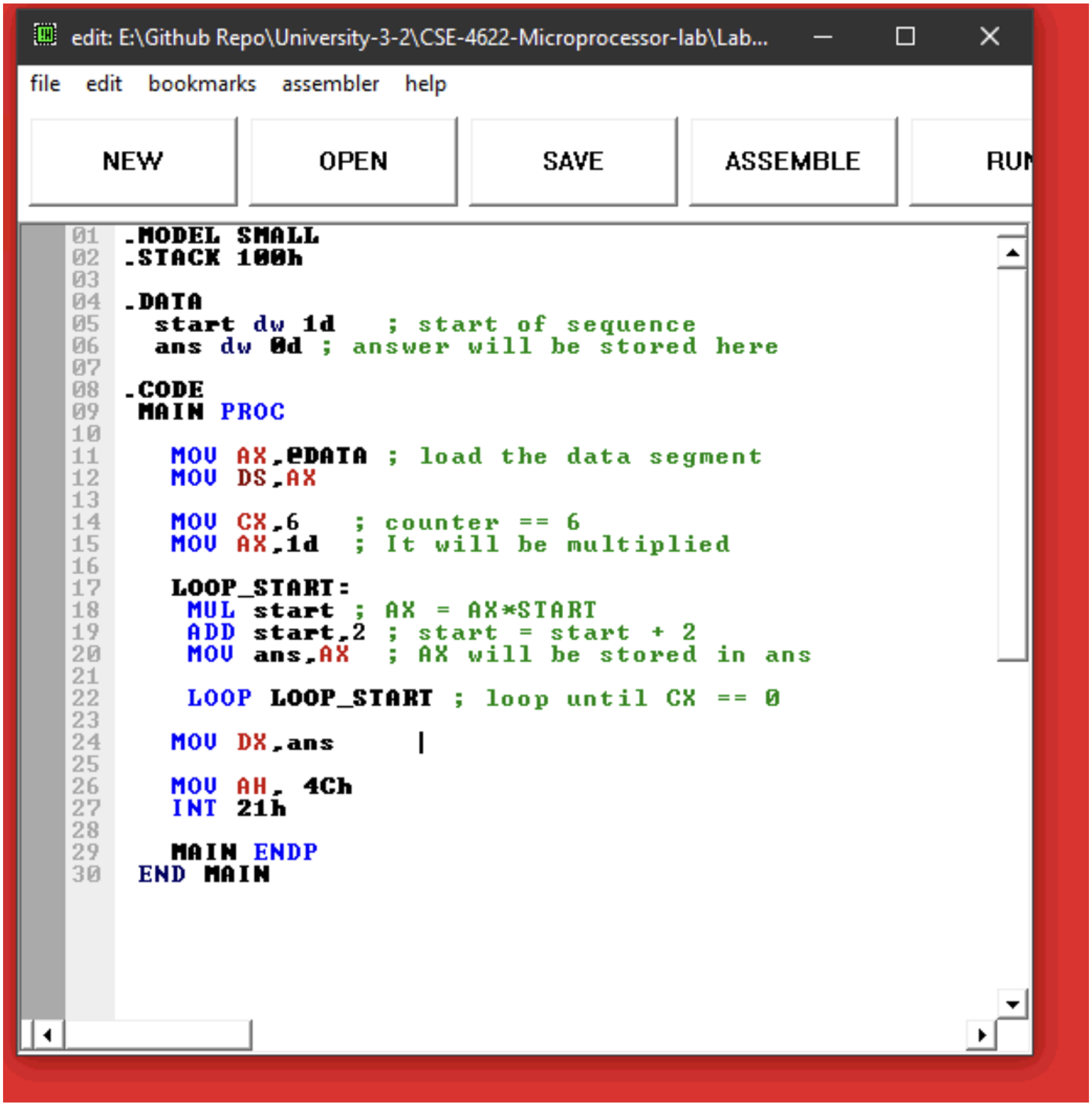
TASK-1

In the first task we had to get the product of the series 1,3,5,7,9,11 .


edit: E:\Github Repo\University-3-2\CSE-4622-Microprocessor-lab\Lab...
file
edit
bookmarks
assembler
help
NEW
OPEN
SAVE
ASSEMBLE
RUN

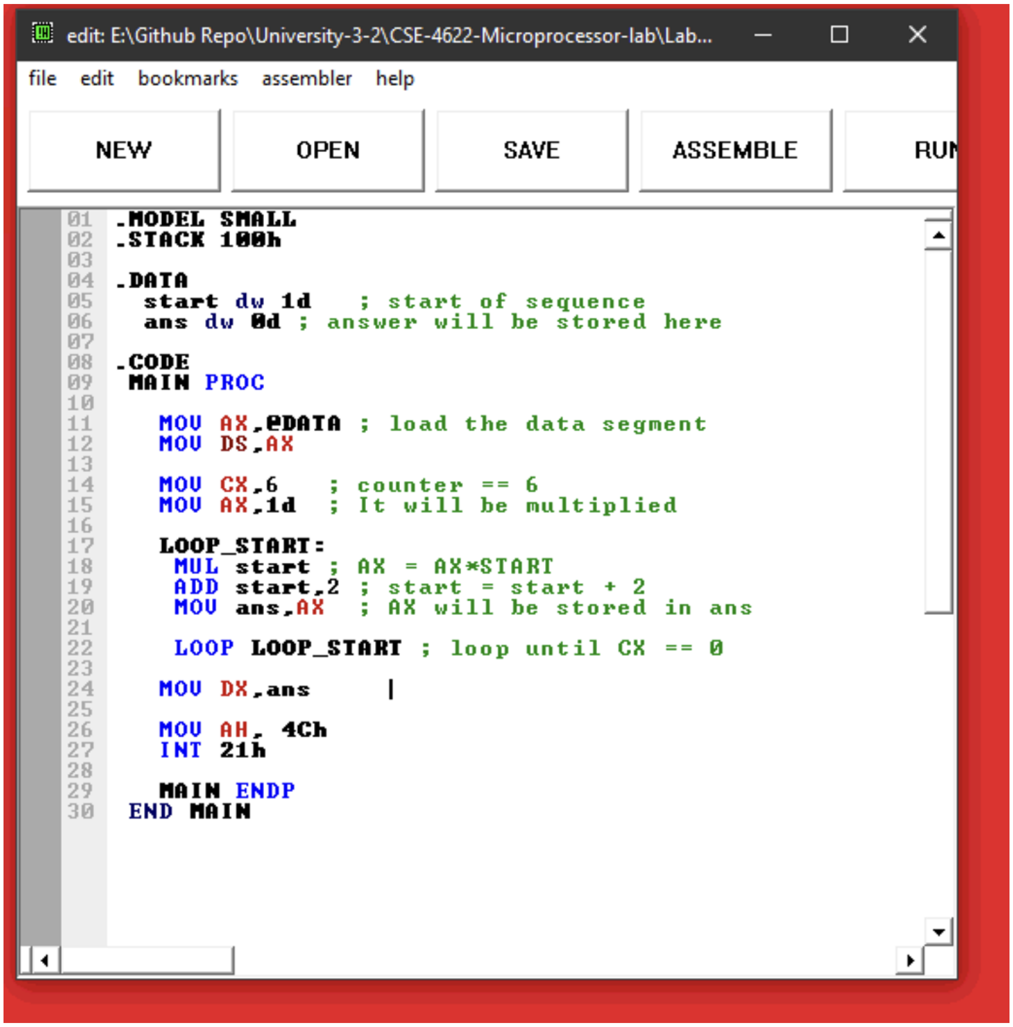
```
.MODEL SMALL
.STACK 100h

.DATA
  start dw 1d    ; start of sequence
  ans dw 0d ; answer will be stored here

.CODE
 MAIN PROC

   MOV AX,@DATA ; load the data segment
   MOV DS,AX

   MOV CX,6    ; counter == 6
   MOV AX,1d   ; It will be multiplied

   LOOP_START:
    MUL start ; AX = AX*START
    ADD start,2 ; start = start + 2
    MOV ans,AX  ; AX will be stored in ans

    LOOP LOOP_START ; loop until CX == 0

   MOV DX,ans

   MOV AH, 4Ch
   INT 21h

   MAIN ENDP
 END MAIN
```

Here I took a variable named “start”, initialized to 1. This is the start of the sequence. We took a counter for the loop and initialized it to 6. Then just kept on looping until the counter gets to zero.

Logic was:

```
    Start = 1
    Ans = 0
    AX = 1

    LOOP
        AX = AX*START
        START=START+2
        ANS = AX
```

Final ans is;

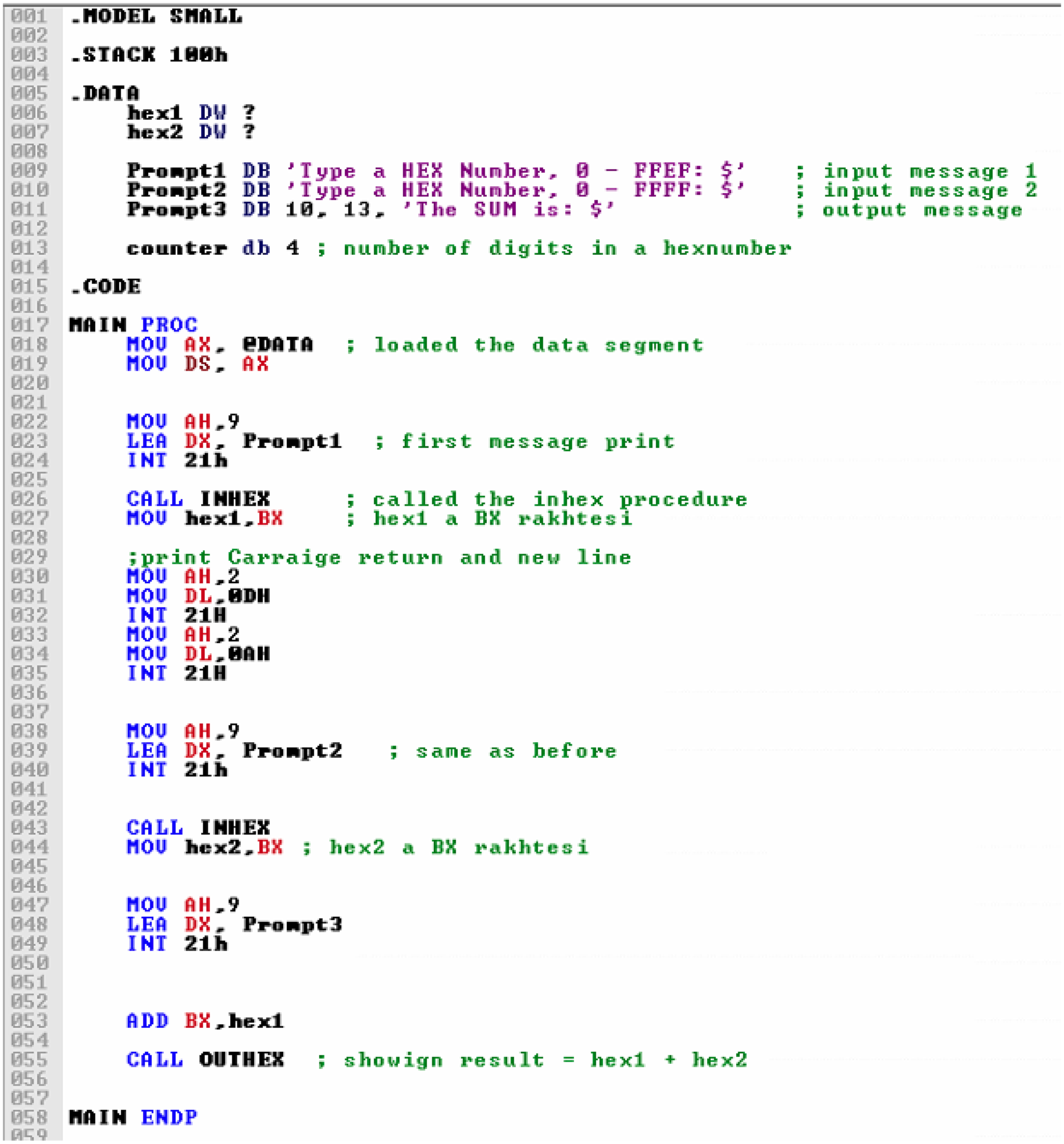

edit: E:\Github Repo\University-3-2\CSE-4622-Microprocessor-lab\Lab...
file
edit
bookmarks
assembler
help
NEW
OPEN
SAVE
ASSEMBLE
RUN

```
.MODEL SMALL
.STACK 100h

.DATA
  start dw 1d    ; start of sequence
  ans dw 0d ; answer will be stored here

.CODE
 MAIN PROC

    MOV AX,@DATA ; load the data segment
    MOV DS,AX

    MOV CX,6    ; counter == 6
    MOV AX,1d   ; It will be multiplied

    LOOP_START:
     MUL start ; AX = AX*START
     ADD start,2 ; start = start + 2
     MOV ans,AX  ; AX will be stored in ans

     LOOP LOOP_START ; loop until CX == 0

    MOV DX,ans

    MOV AH, 4Ch
    INT 21h

    MAIN ENDP
 END MAIN
```

TASK-2

So on this problem I had to write two procedures that were called from the main. One to take input and the other to display output.

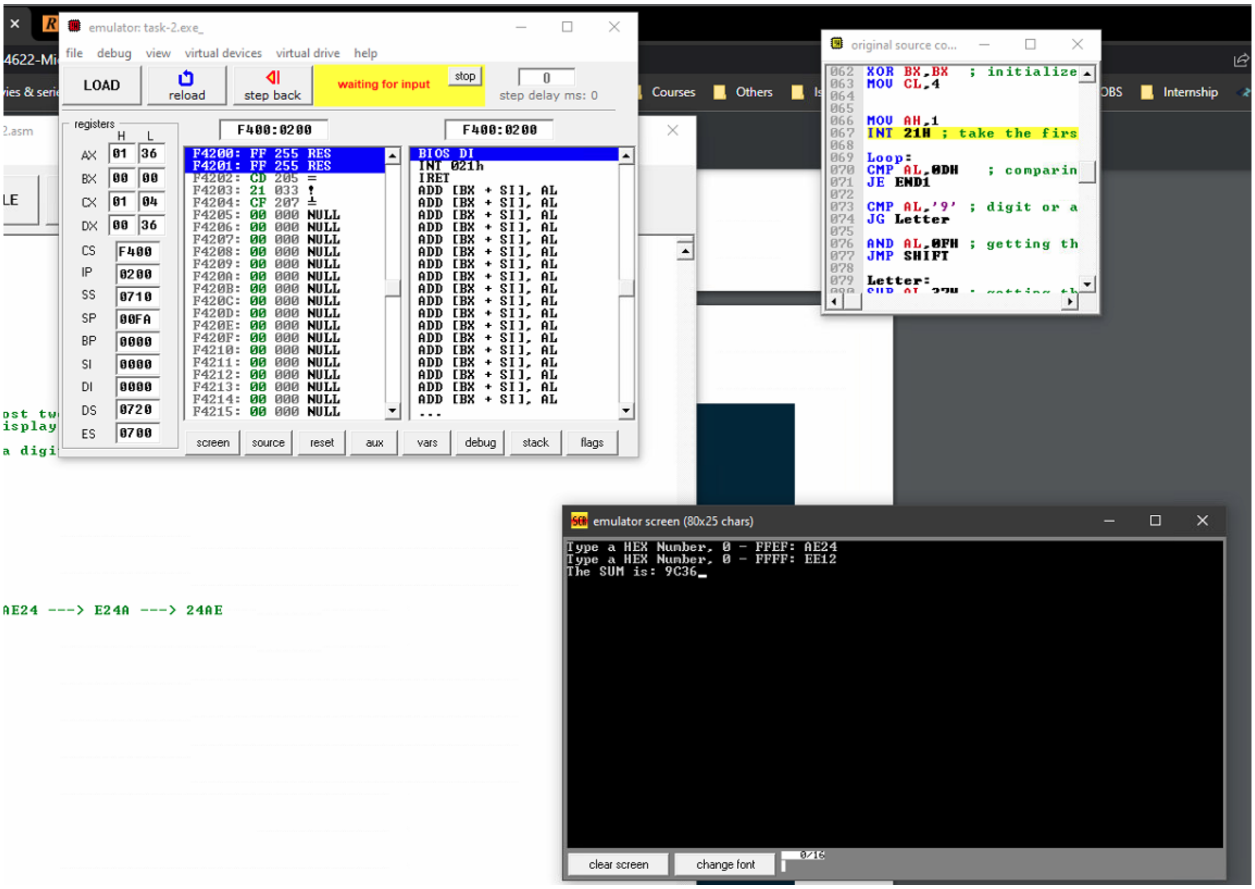
First the MAIN PROCEDURE:



```
.MODEL SMALL

.STACK 100h

.DATA
    hex1 DW ?
    hex2 DW ?

    Prompt1 DB 'Type a HEX Number, 0 - FFEF: $'    ; input message 1
    Prompt2 DB 'Type a HEX Number, 0 - FFFF: $'    ; input message 2
    Prompt3 DB 10, 13, 'The SUM is: $'             ; output message

    counter db 4 ; number of digits in a hexnumber

.CODE

MAIN PROC
    MOV AX, @DATA  ; loaded the data segment
    MOV DS, AX


    MOV AH,9
    LEA DX, Prompt1  ; first message print
    INT 21h

    CALL INHEX     ; called the inhex procedure
    MOV hex1,BX    ; hex1 a BX rakhtesi

    ;print Carraige return and new line
    MOV AH,2
    MOV DL,0DH
    INT 21H
    MOV AH,2
    MOV DL,0AH
    INT 21H


    MOV AH,9
    LEA DX, Prompt2   ; same as before
    INT 21h


    CALL INHEX
    MOV hex2,BX ; hex2 a BX rakhtesi


    MOV AH,9
    LEA DX, Prompt3
    INT 21h



    ADD BX,hex1

    CALL OUTHEX  ; showign result = hex1 + hex2


MAIN ENDP
```

Here we just displayed two prompts and called the inhex procedure to take input. After calculating the sum we called the outhex procedure to show the output.

RESULT:



emulator: task-2.exe_
file debug view virtual devices virtual drive help
LOAD
reload
step back
waiting for input
stop
step delay ms: 0
screen source reset aux vars debug stack flags
original source co...
emulator screen (80x25 chars)
clear screen
change font

registers

	H	L
AX	01	36
BX	00	00
CX	01	04
DX	00	36

CS F400
IP 0200
SS 0710
SP 00FA
BP 0000
SI 0000
DI 0000
DS 0720
ES 0700

```
F400:0200
F4200: FF 255 RES
F4201: FF 255 RES
F4202: CD 205 =
F4203: 21 033 !
F4204: CF 207 ╧
F4205: 00 000 NULL
F4206: 00 000 NULL
F4207: 00 000 NULL
F4208: 00 000 NULL
F4209: 00 000 NULL
F420A: 00 000 NULL
F420B: 00 000 NULL
F420C: 00 000 NULL
F420D: 00 000 NULL
F420E: 00 000 NULL
F420F: 00 000 NULL
F4210: 00 000 NULL
F4211: 00 000 NULL
F4212: 00 000 NULL
F4213: 00 000 NULL
F4214: 00 000 NULL
F4215: 00 000 NULL
```

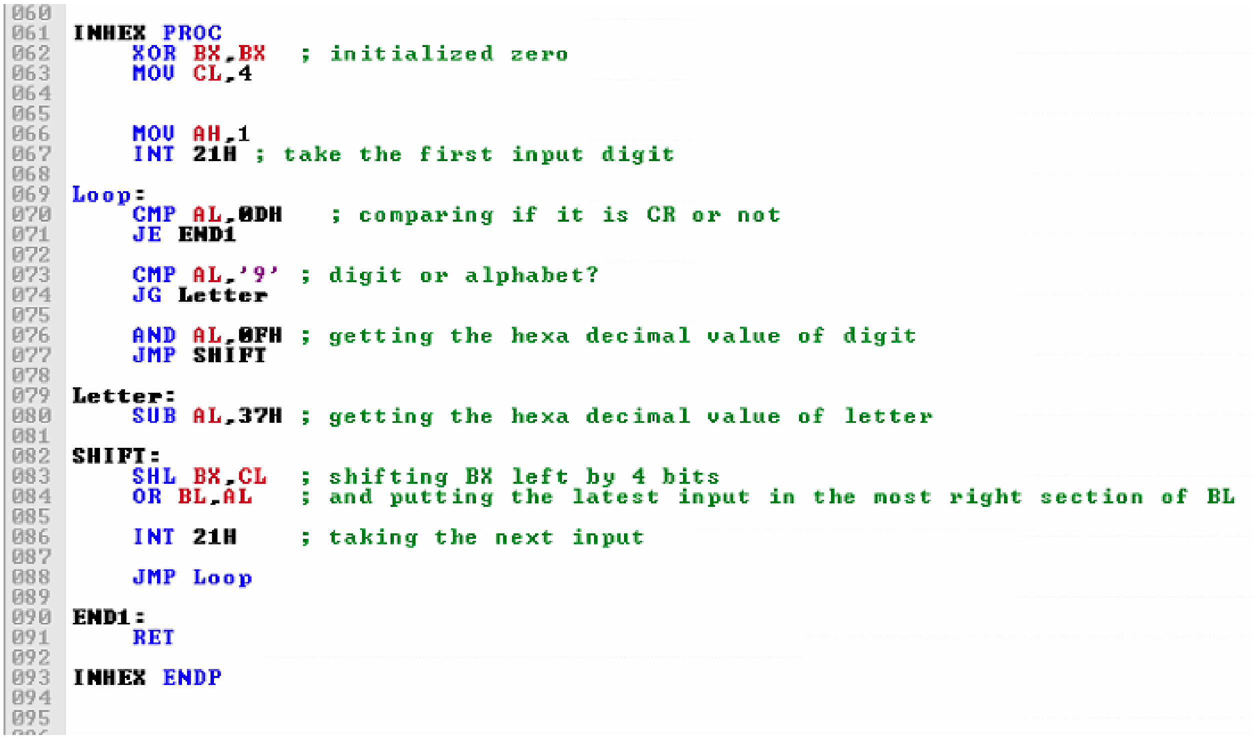
```
F400:0200
BIOS DI
INT 021h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
...
```

```
XOR BX,BX   ; initialize
MOV CL,4


MOV AH,1
INT 21H ; take the firs

Loop:
CMP AL,0DH    ; comparin
JE END1

CMP AL,'9'  ; digit or a
JG Letter

AND AL,0FH ; getting th
JMP SHIFT

Letter:
```

```
Type a HEX Number, 0 - FFEF: AE24
Type a HEX Number, 0 - FFFF: EE12
The SUM is: 9C36_
```

```
AE24 ---> E24A ---> 24AE
```

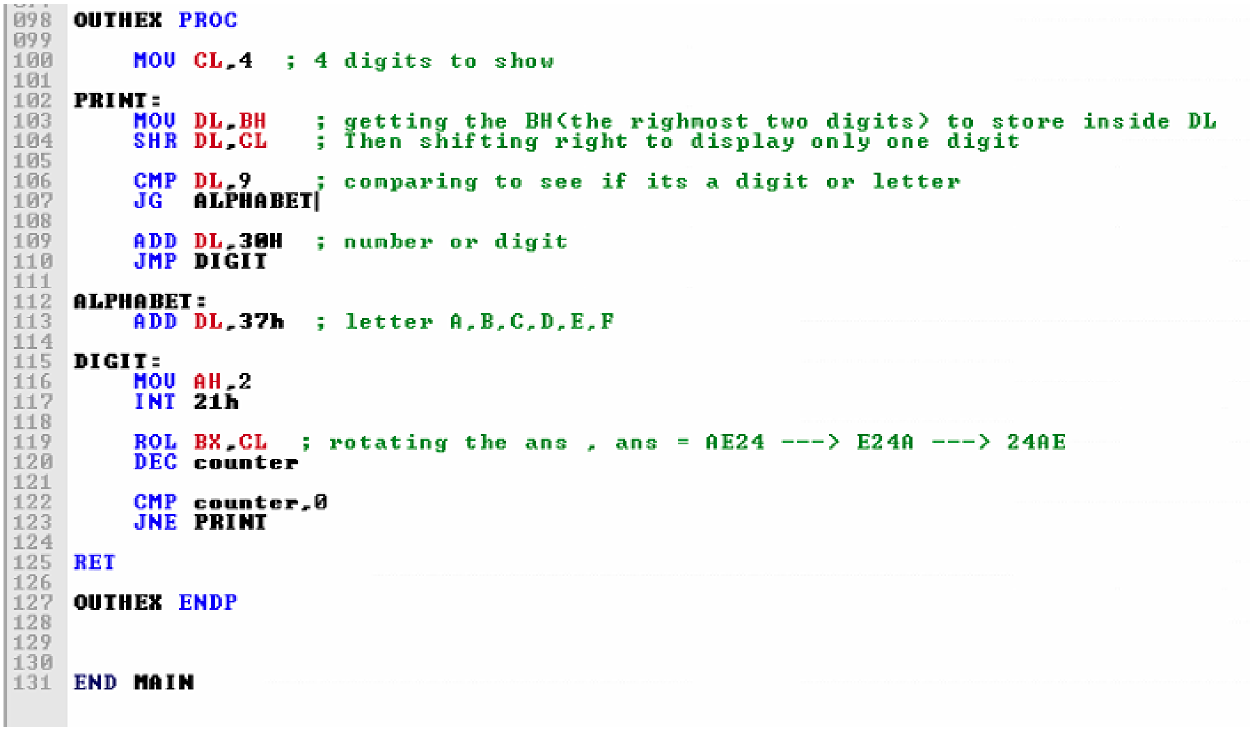
NOW THE INHEX



```
INHEX PROC
    XOR BX,BX   ; initialized zero
    MOV CL,4


    MOV AH,1
    INT 21H ; take the first input digit

Loop:
    CMP AL,0DH    ; comparing if it is CR or not
    JE END1

    CMP AL,'9' ; digit or alphabet?
    JG Letter

    AND AL,0FH ; getting the hexa decimal value of digit
    JMP SHIFT

Letter:
    SUB AL,37H ; getting the hexa decimal value of letter

SHIFT:
    SHL BX,CL  ; shifting BX left by 4 bits
    OR BL,AL   ; and putting the latest input in the most right section of BL

    INT 21H    ; taking the next input

    JMP Loop

END1:
    RET

INHEX ENDP
```

Here we used the BX to store the input. We had to keep track of two types of input: one is digit and the other one is alphabet. And we took the input using a loop and converted it to their respective hexadecimal value and stored it inside BX. We also had to left-shift the BX by 4 bits to take the input one after another.

NOW THE OUTHEX:



```
OUTHEX PROC

    MOV CL,4  ; 4 digits to show

PRINT:
    MOV DL,BH   ; getting the BH(the righmost two digits) to store inside DL
    SHR DL,CL   ; Then shifting right to display only one digit

    CMP DL,9    ; comparing to see if its a digit or letter
    JG  ALPHABET

    ADD DL,30H  ; number or digit
    JMP DIGIT

ALPHABET:
    ADD DL,37h  ; letter A,B,C,D,E,F

DIGIT:
    MOV AH,2
    INT 21h

    ROL BX,CL  ; rotating the ans , ans = AE24 ---> E24A ---> 24AE
    DEC counter

    CMP counter,0
    JNE PRINT

RET

OUTHEX ENDP


END MAIN
```

OUTHEX is basically just taking one by one digits/alphabet from the BH and displaying it to the console. The process is almost similar to the INHEX. Here we had to convert the hexadecimal value to ASCII code.