Indian Institute of Technology Patna

Mechatronics, Instrumentation And Controls Laboratory

Lab 7 Report

Topic:

Introduction to Microprocessor

Submitted by: Aditya Shah(2011mt02)
(MTech-Mechatronics)

Lab In-charge:

Dr. Atul Thakur
(Assistant Professor)
Department of Mechanical Engineering
IIT Patna

1. Aim of the Experiments.

1.Write a program to ADD and subtract two 16 bit number and also store carry for addition and borrow for subtraction.
2.Write a program to compute factorial of a number.
3.Write a program to multiply to 16 bit no using repetitive addition.
4.Write a program to divide to 16 bit no using repetitive subtraction.

2. Pre-Requisites/Components Required

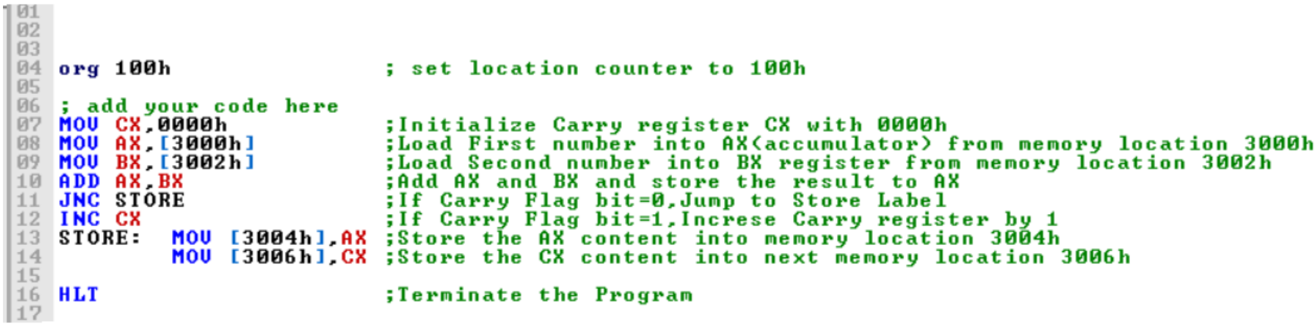
In the Simulation Implementation I have used the following Software:

- EMU8086 - THE MICROPROCESSOR EMULATOR

1.1) Write a program to ADD and subtract two 16 bit number and also store carry for addition and borrow for subtraction.

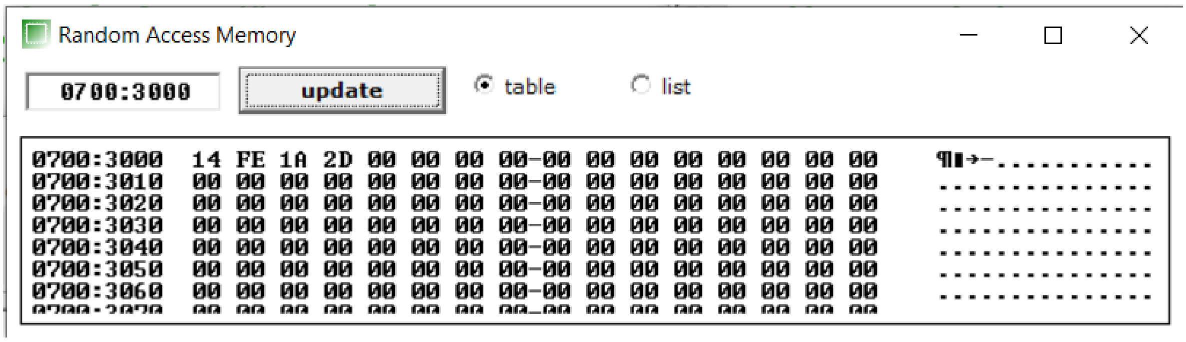
Addition: -

- Program for ADD



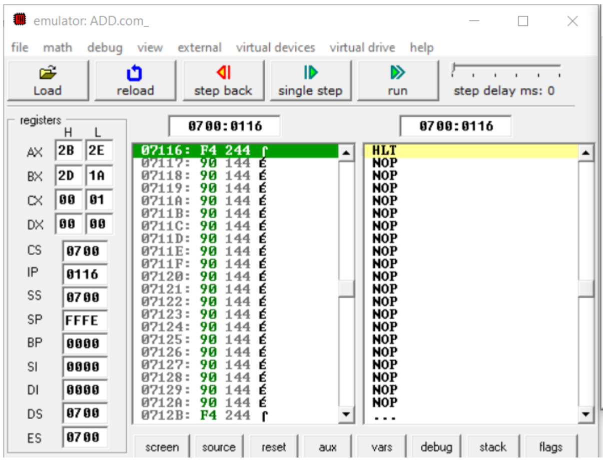
```



org 100h                        ; set location counter to 100h

; add your code here
MOV CX,0000h                    ;Initialize Carry register CX with 0000h
MOV AX,[3000h]                  ;Load First number into AX(accumulator) from memory location 3000h
MOV BX,[3002h]                  ;Load Second number into BX register from memory location 3002h
ADD AX,BX                       ;Add AX and BX and store the result to AX
JNC STORE                       ;If Carry Flag bit=0,Jump to Store Label
INC CX                          ;If Carry Flag bit=1,Increse Carry register by 1
STORE:    MOV [3004h],AX        ;Store the AX content into memory location 3004h
          MOV [3006h],CX        ;Store the CX content into next memory location 3006h

HLT                             ;Terminate the Program

```

- Data stored in memory location 3000h(First Number) and 3002h(Second Number) respectively



```
0700:3000  14 FE 1A 2D 00 00 00 00-00 00 00 00 00 00 00 00   ¶■→-............
0700:3010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- ➢ Execution



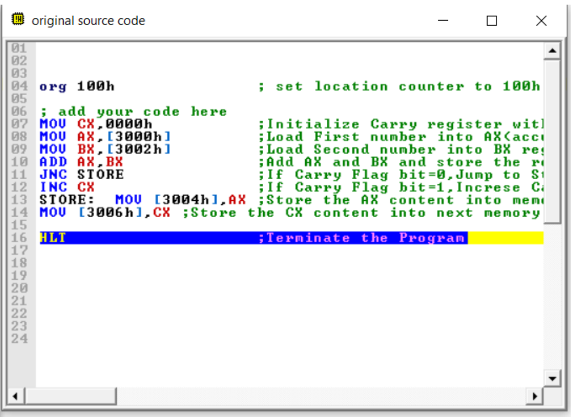
registers

	H	L
AX	2B	2E
BX	2D	1A
CX	00	01
DX	00	00

CS	0700
IP	0116
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

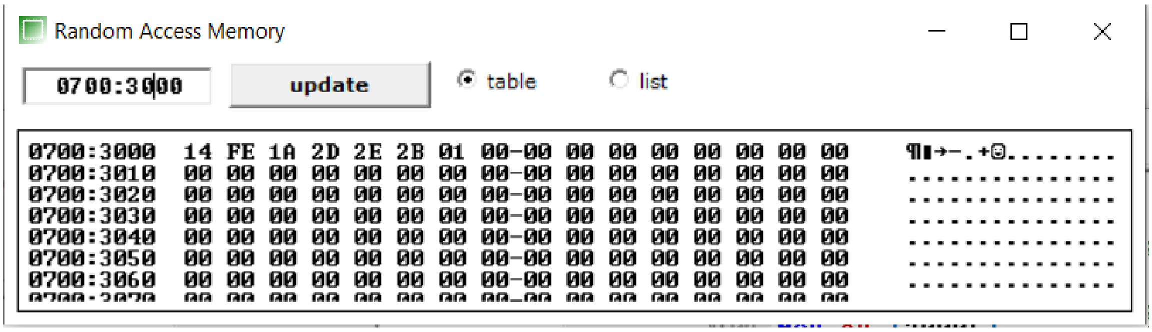
```
0700:0116
07116: F4 244 ſ
07117: 90 144 É
07118: 90 144 É
07119: 90 144 É
0711A: 90 144 É
0711B: 90 144 É
0711C: 90 144 É
0711D: 90 144 É
0711E: 90 144 É
0711F: 90 144 É
07120: 90 144 É
07121: 90 144 É
07122: 90 144 É
07123: 90 144 É
07124: 90 144 É
07125: 90 144 É
07126: 90 144 É
07127: 90 144 É
07128: 90 144 É
07129: 90 144 É
0712A: 90 144 É
0712B: F4 244 ſ
```

```
0700:0116
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



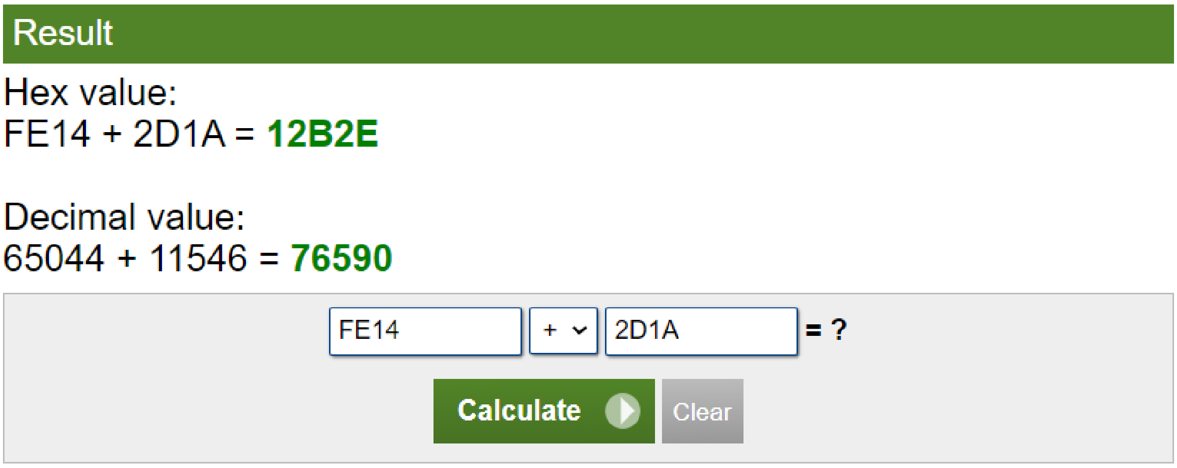
```



org 100h                         ; set location counter to 100h

; add your code here
MOV CX,0000h                     ;Initialize Carry register wit
MOV AX,[3000h]                   ;Load First number into AX(acc
MOV BX,[3002h]                   ;Load Second number into BX re
ADD AX,BX                        ;Add AX and BX and store the r
JNC STORE                        ;If Carry Flag bit=0,Jump to S
INC CX                           ;If Carry Flag bit=1,Increse C
STORE:  MOV [3004h],AX ;Store the AX content into mem
MOV [3006h],CX ;Store the CX content into next memory

HLT                              ;Terminate the Program
```

- ➢ Result



```
0700:3000
0700:3000  14 FE 1A 2D 2E 2B 01 00-00 00 00 00 00 00 00 00   ¶▮→-.+☺.........
0700:3010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- ➢ Verification



Result

Hex value:
FE14 + 2D1A = **12B2E**

Decimal value:
65044 + 11546 = **76590**

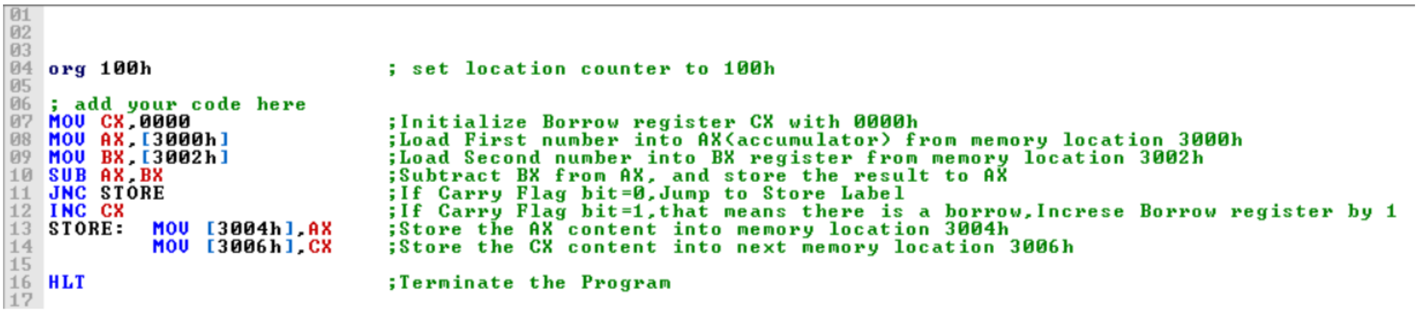
FE14 + 2D1A = ?

- ❖ Indeed my result stored in memory location 3004h(**SUM**) and 3006h(**CARRY**),matches with calculated values as shown above.

Name: Aditya Shah Roll: 2011mt02

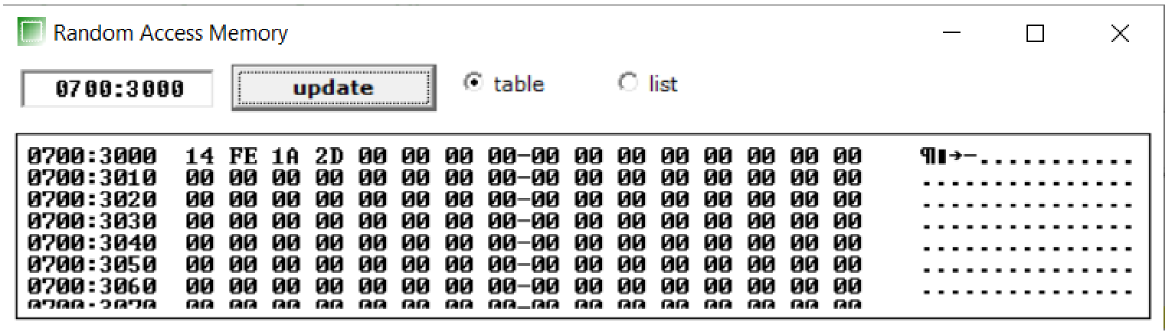
Subtraction: -

- Program for Subtraction



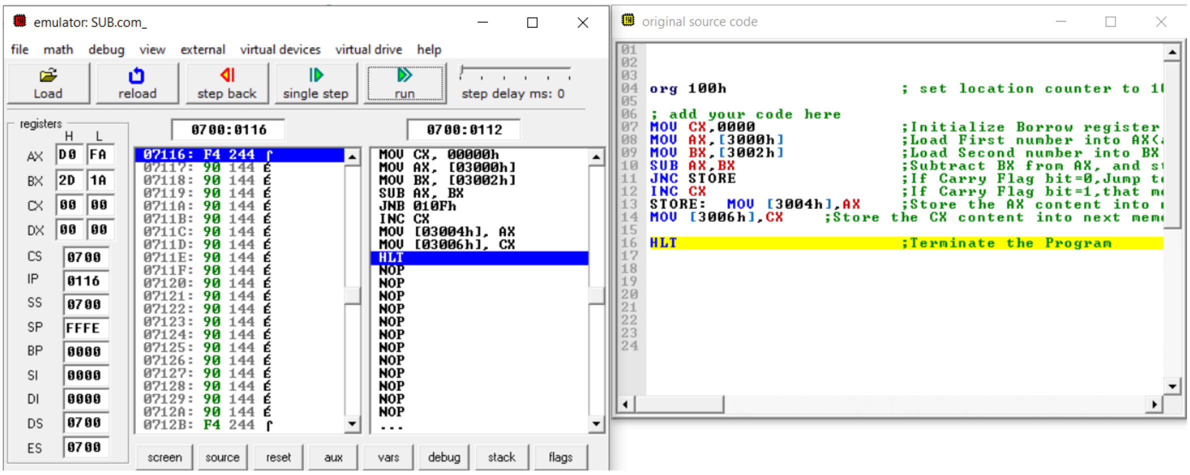
```
org 100h                                  ; set location counter to 100h

; add your code here
MOV CX,0000                               ;Initialize Borrow register CX with 0000h
MOV AX,[3000h]                            ;Load First number into AX(accumulator) from memory location 3000h
MOV BX,[3002h]                            ;Load Second number into BX register from memory location 3002h
SUB AX,BX                                 ;Subtract BX from AX, and store the result to AX
JNC STORE                                 ;If Carry Flag bit=0,Jump to Store Label
INC CX                                    ;If Carry Flag bit=1,that means there is a borrow,Increse Borrow register by 1
STORE:   MOV [3004h],AX                   ;Store the AX content into memory location 3004h
         MOV [3006h],CX                   ;Store the CX content into next memory location 3006h

HLT                                       ;Terminate the Program
```

- Data stored in memory location 3000h(First Number) and 3002h(Second Number) respectively



```
0700:3000

0700:3000   14 FE 1A 2D 00 00 00 00-00 00 00 00 00 00 00 00   ¶▮→-............
0700:3010   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3020   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3030   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3040   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3050   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3060   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3070   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- Execution

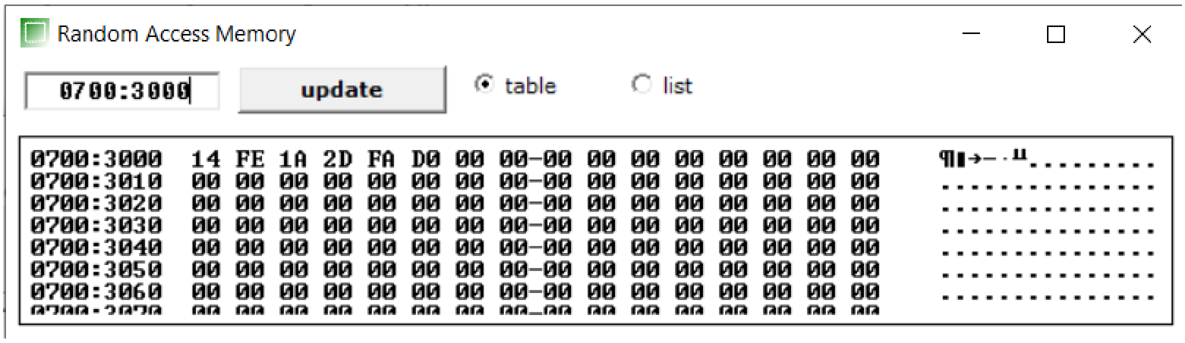


```
registers   H   L
AX          D0  FA
BX          2D  1A
CX          00  00
DX          00  00
CS          0700
IP          0116
SS          0700
SP          FFFE
BP          0000
SI          0000
DI          0000
DS          0700
ES          0700
```

```
0700:0116                       0700:0112

07116:  F4 244 ſ                MOV CX, 00000h
07117:  90 144 É                MOV AX, [03000h]
07118:  90 144 É                MOV BX, [03002h]
07119:  90 144 É                SUB AX, BX
0711A:  90 144 É                JNB 010Fh
0711B:  90 144 É                INC CX
0711C:  90 144 É                MOV [03004h], AX
0711D:  90 144 É                MOV [03006h], CX
0711E:  90 144 É                HLT
0711F:  90 144 É                NOP
07120:  90 144 É                NOP
07121:  90 144 É                NOP
07122:  90 144 É                NOP
07123:  90 144 É                NOP
07124:  90 144 É                NOP
07125:  90 144 É                NOP
07126:  90 144 É                NOP
07127:  90 144 É                NOP
07128:  90 144 É                NOP
07129:  90 144 É                NOP
0712A:  90 144 É                NOP
0712B:  F4 244 ſ                ...
```

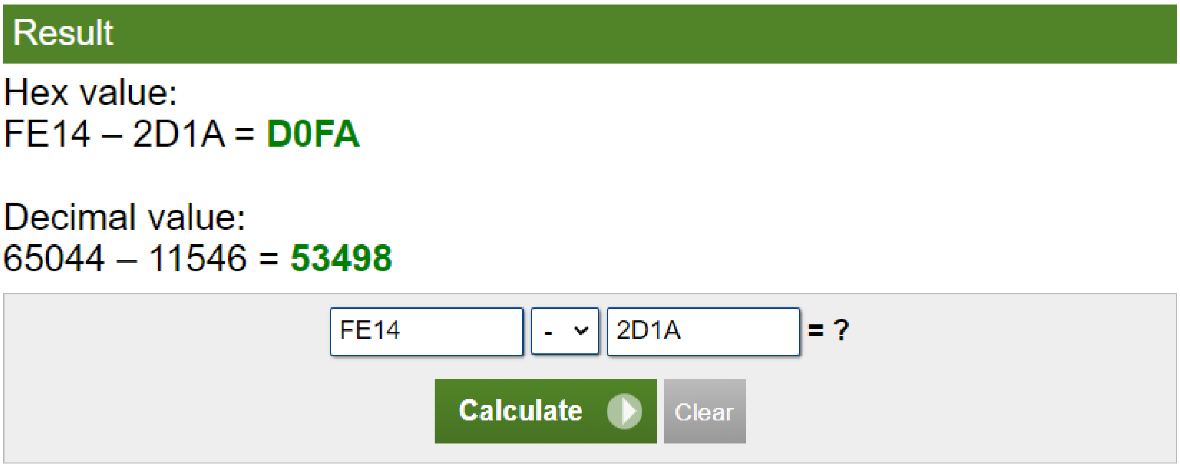
```
org 100h                        ; set location counter to 1(

; add your code here
MOV CX,0000                     ;Initialize Borrow register
MOV AX,[3000h]                  ;Load First number into AX(
MOV BX,[3002h]                  ;Load Second number into BX
SUB AX,BX                       ;Subtract BX from AX, and s
JNC STORE                       ;If Carry Flag bit=0,Jump to
INC CX                          ;If Carry Flag bit=1,that m
STORE:   MOV [3004h],AX         ;Store the AX content into
MOV [3006h],CX        ;Store the CX content into next mem

HLT                             ;Terminate the Program
```

- Result



```
0700:3000

0700:3000   14 FE 1A 2D FA D0 00 00-00 00 00 00 00 00 00 00   ¶▮→-·╨..........
0700:3010   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3020   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3030   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3040   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3050   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3060   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3070   00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- Verification



Result

Hex value:
FE14 – 2D1A = **D0FA**

Decimal value:
65044 – 11546 = **53498**

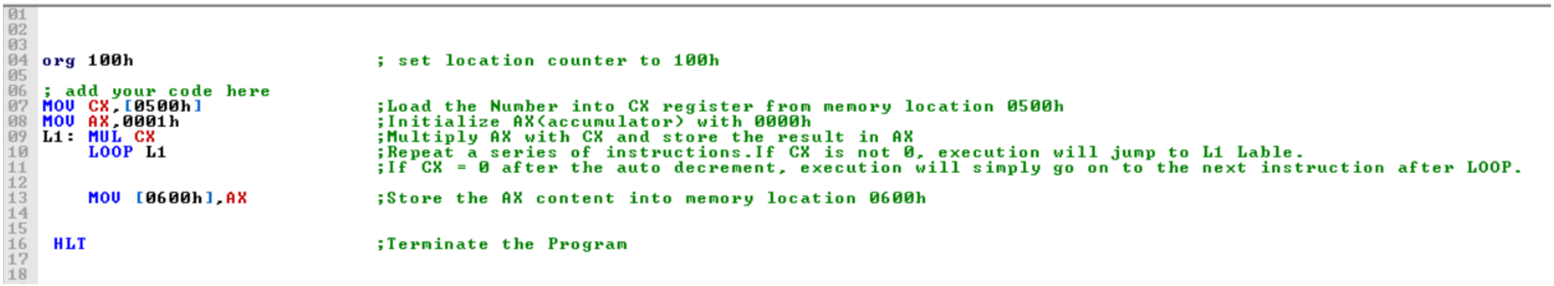
FE14 - 2D1A = ?

- Indeed my result stored in memory location 3004h(**Difference**) and 3006h(**Borrow**),matches with calculated values as shown above.

1.2) Write a program to compute factorial of a number.

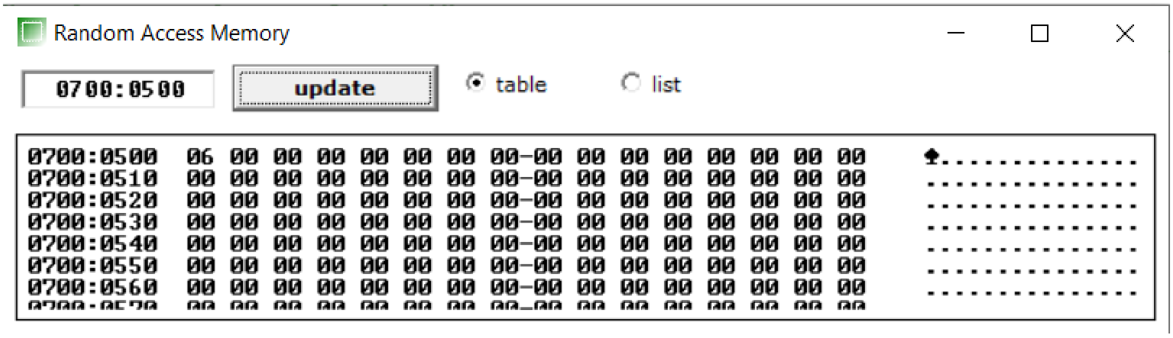
Factorial: -

- Program



```



org 100h                        ; set location counter to 100h

; add your code here
MOV CX,[0500h]                  ;Load the Number into CX register from memory location 0500h
MOV AX,0001h                    ;Initialize AX(accumulator) with 0000h
L1: MUL CX                      ;Multiply AX with CX and store the result in AX
    LOOP L1                     ;Repeat a series of instructions.If CX is not 0, execution will jump to L1 Lable.
                                ;If CX = 0 after the auto decrement, execution will simply go on to the next instruction after LOOP.

    MOV [0600h],AX              ;Store the AX content into memory location 0600h


 HLT                            ;Terminate the Program


```

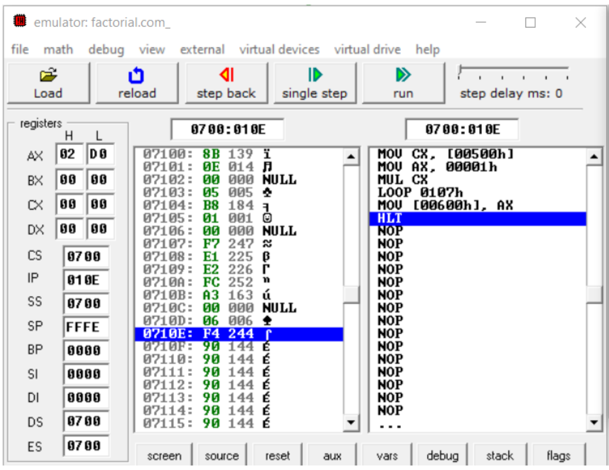
- Data stored in memory location 0500h(Number)



Random Access Memory

```
0700:0500  06 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ♠...............
0700:0510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0570  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- ➢ Execution

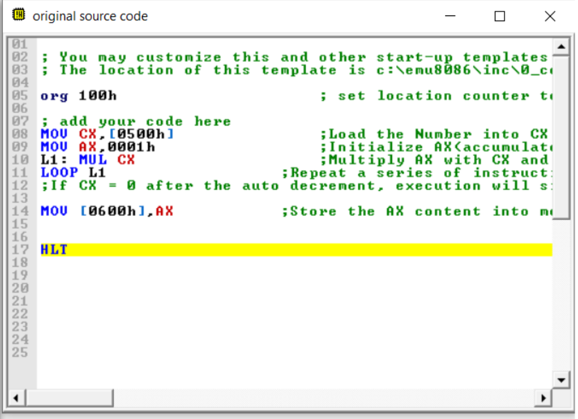


emulator: factorial.com_

```
      H  L
AX   02 D0
BX   00 00
CX   00 00
DX   00 00
CS   0700
IP   010E
SS   0700
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

```
0700:010E

07100: 8B 139 ï
07101: 0E 014 ♫
07102: 00 000 NULL
07103: 05 005 ♣
07104: B8 184 ╕
07105: 01 001 ☺
07106: 00 000 NULL
07107: F7 247 ≈
07108: E1 225 ß
07109: E2 226 Γ
0710A: FC 252 ⁿ
0710B: A3 163 ú
0710C: 00 000 NULL
0710D: 06 006 ♠
0710E: F4 244 ⌠
0710F: 90 144 É
07110: 90 144 É
07111: 90 144 É
07112: 90 144 É
07113: 90 144 É
07114: 90 144 É
07115: 90 144 É
```

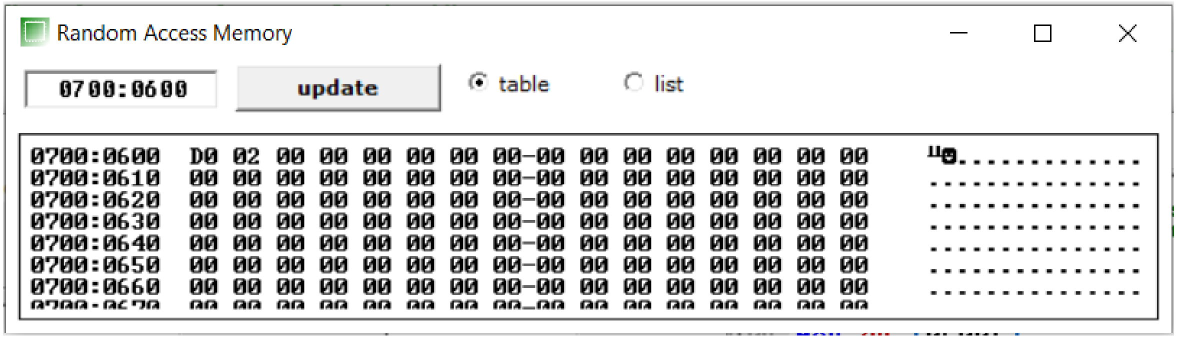
```
0700:010E

MOV CX, [00500h]
MOV AX, 00001h
MUL CX
LOOP 0107h
MOV [00600h], AX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



original source code

```

; You may customize this and other start-up templates
; The location of this template is c:\emu8086\inc\0_c

org 100h                       ; set location counter t

; add your code here
MOV CX,[0500h]                 ;Load the Number into CX
MOV AX,0001h                   ;Initialize AX(accumulat
L1: MUL CX                     ;Multiply AX with CX and
LOOP L1                     ;Repeat a series of instruct
;If CX = 0 after the auto decrement, execution will s

MOV [0600h],AX              ;Store the AX content into m


HLT
```

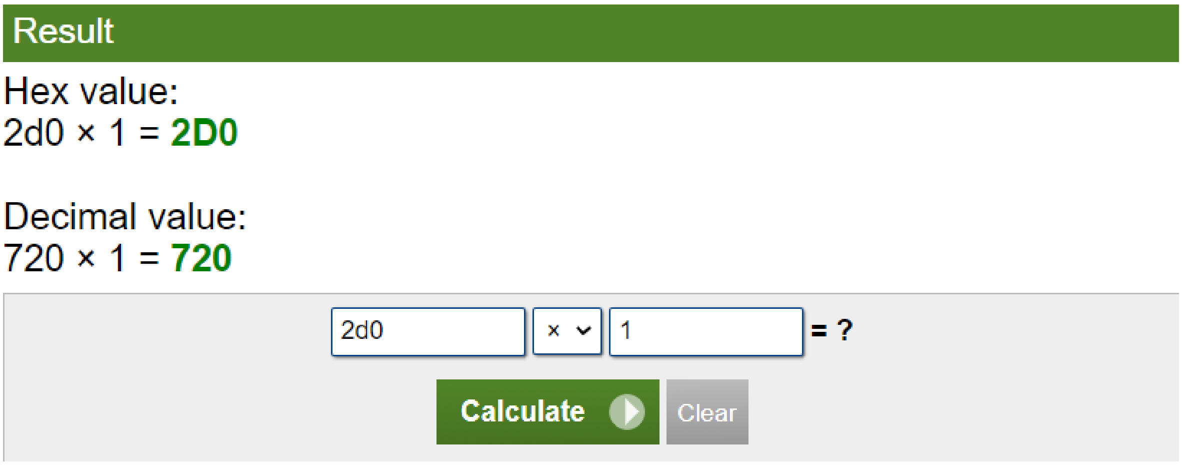
- ➢ Result



Random Access Memory

```
0700:0600  D0 02 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ╨☻..............
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- ➢ Verification



Result

Hex value:
2d0 × 1 = **2D0**

Decimal value:
720 × 1 = **720**

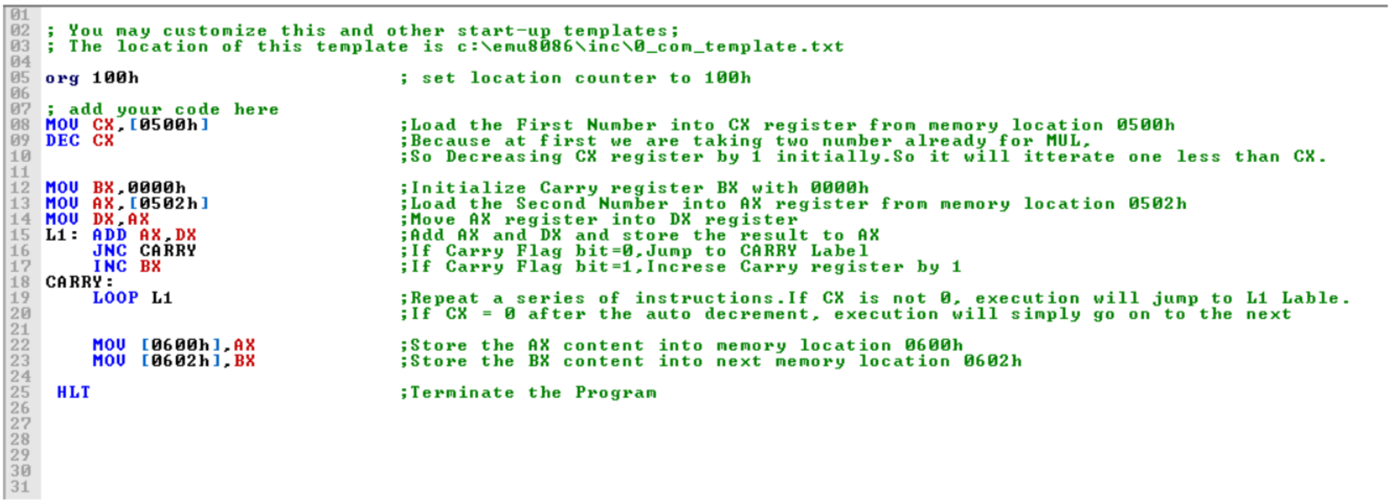
2d0 × 1 = ?

- ❖ Indeed my result stored in memory location 0600h,matches with calculated values as shown above.

1.3) Write a program to multiply to 16 bit no using repetitive addition.

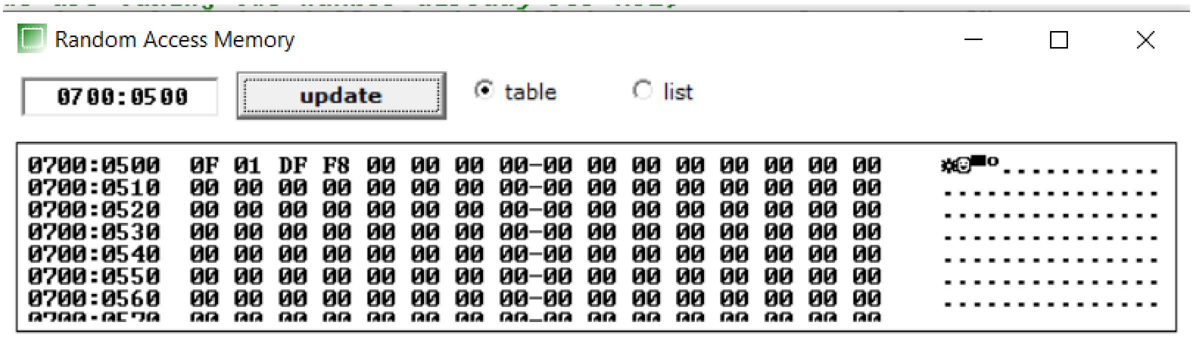
Multiplication using repetitive addition: -

- ➢ Program



```
; You may customize this and other start-up templates;
; The location of this template is c:\emu8086\inc\0_com_template.txt

org 100h                       ; set location counter to 100h

; add your code here
MOV CX,[0500h]                 ;Load the First Number into CX register from memory location 0500h
DEC CX                         ;Because at first we are taking two number already for MUL,
                               ;So Decreasing CX register by 1 initially.So it will itterate one less than CX.

MOV BX,0000h                   ;Initialize Carry register BX with 0000h
MOV AX,[0502h]                 ;Load the Second Number into AX register from memory location 0502h
MOV DX,AX                      ;Move AX register into DX register
L1: ADD AX,DX                  ;Add AX and DX and store the result to AX
    JNC CARRY                  ;If Carry Flag bit=0,Jump to CARRY Label
    INC BX                     ;If Carry Flag bit=1,Increse Carry register by 1
CARRY:
    LOOP L1                    ;Repeat a series of instructions.If CX is not 0, execution will jump to L1 Lable.
                               ;If CX = 0 after the auto decrement, execution will simply go on to the next

    MOV [0600h],AX             ;Store the AX content into memory location 0600h
    MOV [0602h],BX             ;Store the BX content into next memory location 0602h

 HLT                           ;Terminate the Program
```

- ➢ Data stored in memory location 0500h(First Number) and 0502h(Second Number) respectively

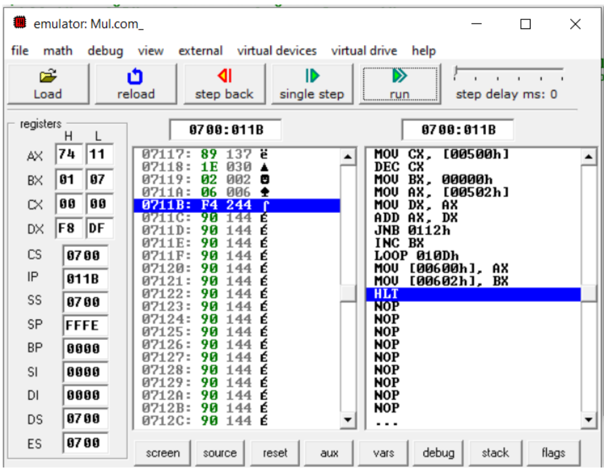


Random Access Memory

0700:0500 update ● table ○ list

```
0700:0500  0F 01 DF F8 00 00 00 00-00 00 00 00 00 00 00 00   ☼☺▀°............
0700:0510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0570  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- ➢ Execution



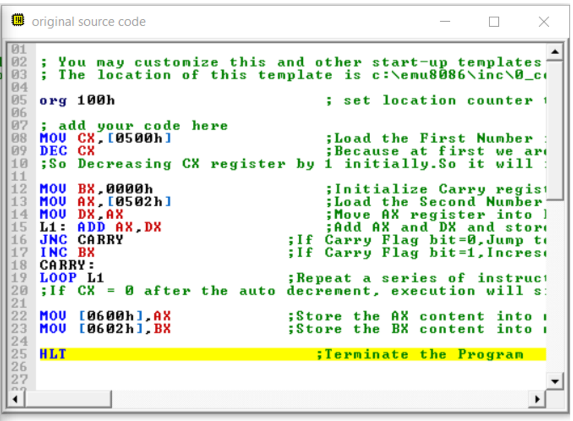
step delay ms: 0

registers

	H	L
AX	74	11
BX	01	07
CX	00	00
DX	F8	DF
CS	0700	
IP	011B	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
0700:011B

07117: 89 137 ë
07118: 1E 030 ▲
07119: 02 002 ☻
0711A: 06 006 ♠
0711B: F4 244 ſ
0711C: 90 144 É
0711D: 90 144 É
0711E: 90 144 É
0711F: 90 144 É
07120: 90 144 É
07121: 90 144 É
07122: 90 144 É
07123: 90 144 É
07124: 90 144 É
07125: 90 144 É
07126: 90 144 É
07127: 90 144 É
07128: 90 144 É
07129: 90 144 É
0712A: 90 144 É
0712B: 90 144 É
0712C: 90 144 É
```

```
0700:011B

MOV CX, [00500h]
DEC CX
MOV BX, 00000h
MOV AX, [00502h]
MOV DX, AX
ADD AX, DX
JNB 0112h
INC BX
LOOP 010Dh
MOV [00600h], AX
MOV [00602h], BX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

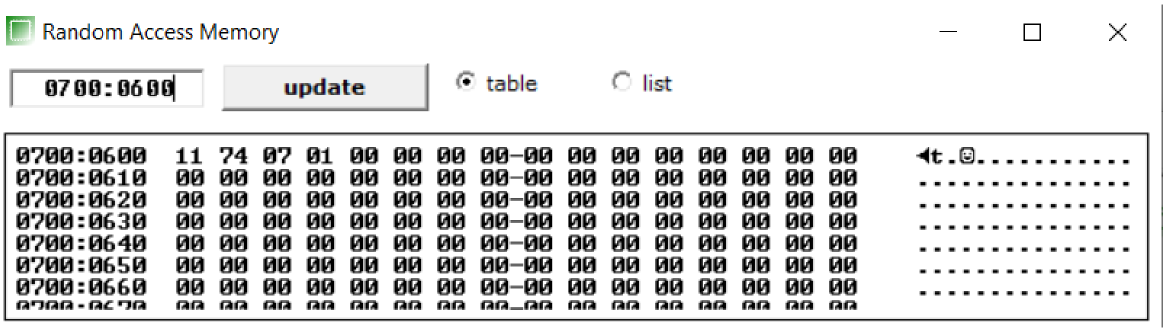


original source code

```
; You may customize this and other start-up templates
; The location of this template is c:\emu8086\inc\0_c

org 100h                        ; set location counter t

; add your code here
MOV CX,[0500h]                  ;Load the First Number :
DEC CX                          ;Because at first we are
;So Decreasing CX register by 1 initially.So it will :

MOV BX,0000h                    ;Initialize Carry regist
MOV AX,[0502h]                  ;Load the Second Number
MOV DX,AX                       ;Move AX register into ]
L1: ADD AX,DX                   ;Add AX and DX and store
JNC CARRY                   ;If Carry Flag bit=0,Jump to
INC BX                      ;If Carry Flag bit=1,Increse
CARRY:
LOOP L1                     ;Repeat a series of instruct
;If CX = 0 after the auto decrement, execution will s:

MOV [0600h],AX              ;Store the AX content into ı
MOV [0602h],BX              ;Store the BX content into ı

HLT                            ;Terminate the Program
```

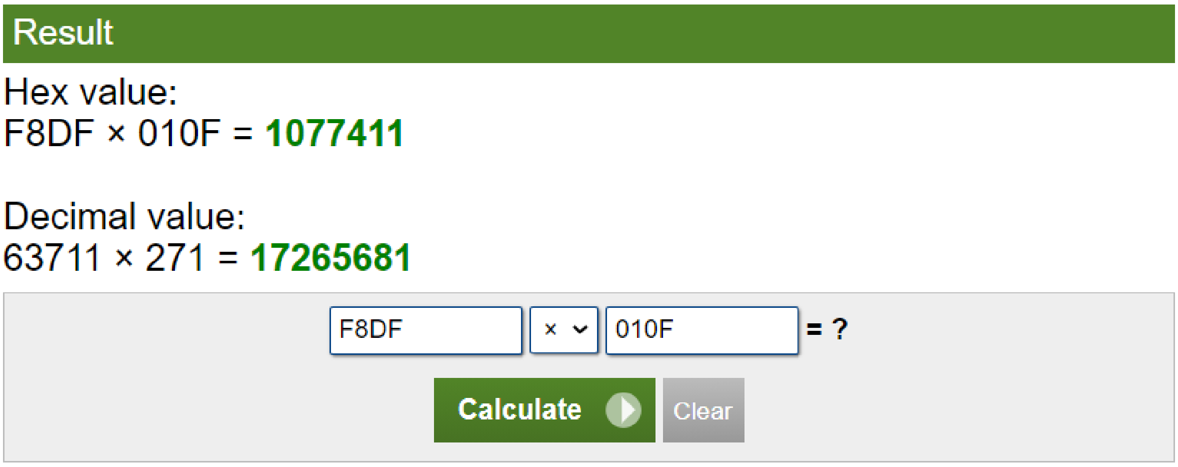
Name: Aditya Shah Roll: 2011mt02

- Result



```
0700:0600
0700:0600  11 74 07 01 00 00 00 00-00 00 00 00 00 00 00 00   ◄t.☺............
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- Verification



Result

Hex value:
F8DF × 010F = 1077411

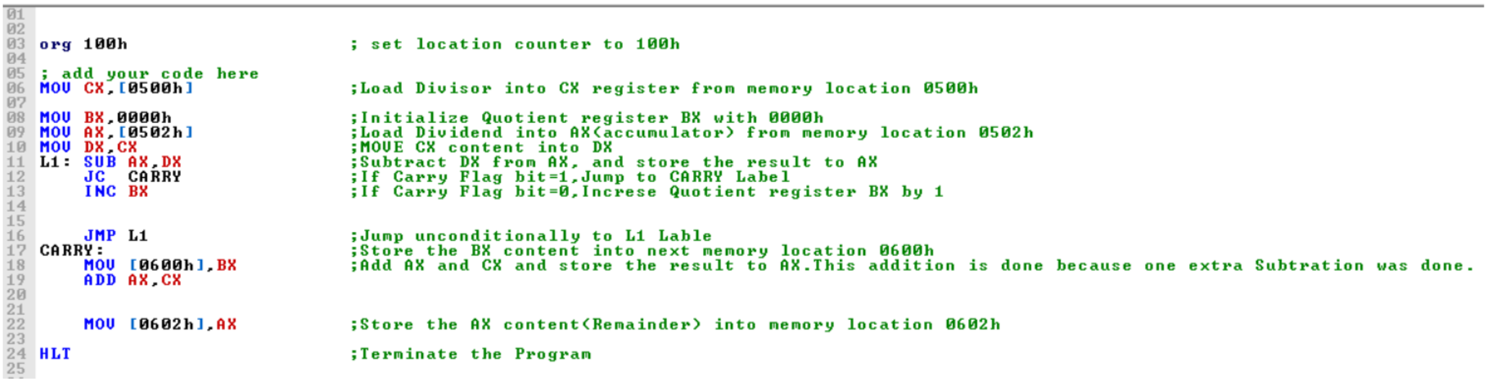
Decimal value:
63711 × 271 = 17265681

❖ Indeed my result stored in memory location 0600h(**Lower 16 bit Result**) and 0602h(**Higher 16 bit Result**),matches with calculated values as shown above.

1.4) Write a program to divide to 16 bit no using repetitive subtraction.

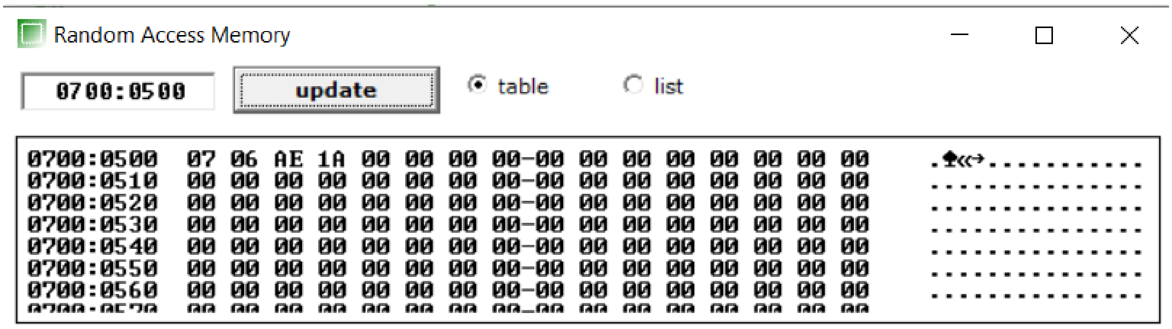
Division using repetitive subtraction: -

- Program



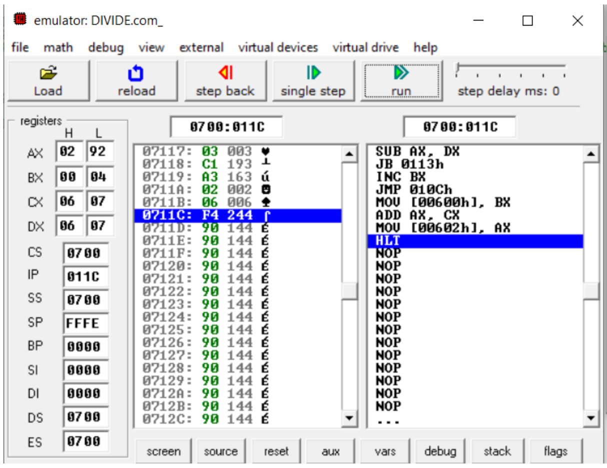
```


org 100h                        ; set location counter to 100h

; add your code here
MOV CX,[0500h]                  ;Load Divisor into CX register from memory location 0500h

MOV BX,0000h                    ;Initialize Quotient register BX with 0000h
MOV AX,[0502h]                  ;Load Dividend into AX(accumulator) from memory location 0502h
MOV DX,CX                       ;MOVE CX content into DX
L1: SUB AX,DX                   ;Subtract DX from AX, and store the result to AX
    JC  CARRY                   ;If Carry Flag bit=1,Jump to CARRY Label
    INC BX                      ;If Carry Flag bit=0,Increse Quotient register BX by 1


    JMP L1                      ;Jump unconditionally to L1 Lable
CARRY:                          ;Store the BX content into next memory location 0600h
    MOV [0600h],BX              ;Add AX and CX and store the result to AX.This addition is done because one extra Subtration was done.
    ADD AX,CX


    MOV [0602h],AX              ;Store the AX content(Remainder) into memory location 0602h

HLT                             ;Terminate the Program

```

- Data stored in memory location 0500h(Divisor) and 0502h(Dividend) respectively



```
0700:0500

0700:0500  07 06 AE 1A 00 00 00 00-00 00 00 00 00 00 00 00   .♠«→............
0700:0510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0520  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0530  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0540  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0550  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0560  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0570  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

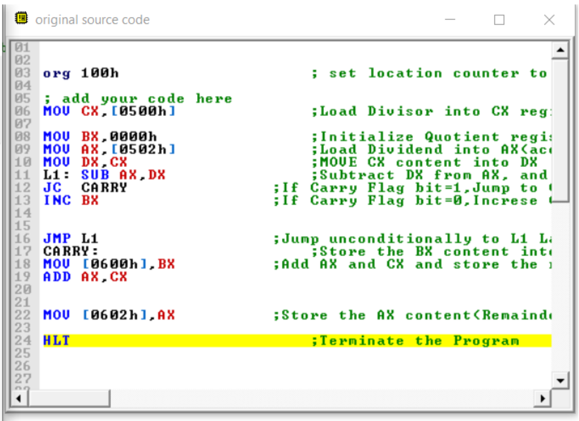
- Execution



```
registers
     H   L
AX  02  92
BX  00  04
CX  06  07
DX  06  07
CS  0700
IP  011C
SS  0700
SP  FFFE
BP  0000
SI  0000
DI  0000
DS  0700
ES  0700
```

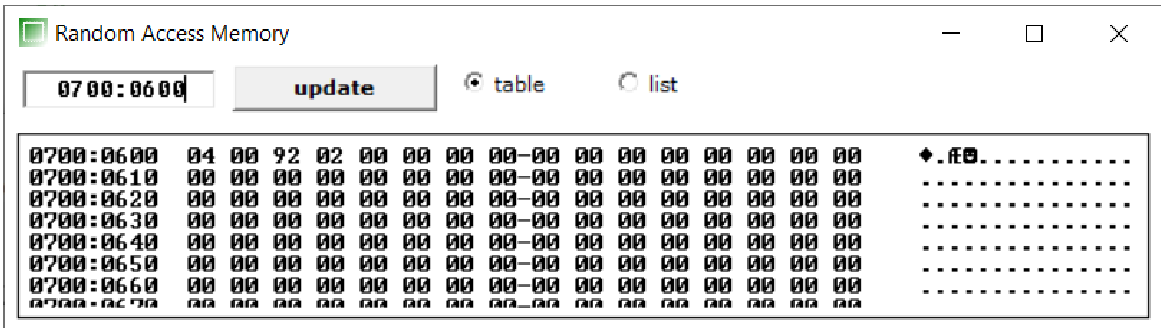
```
0700:011C
07117: 03 003 ♥
07118: C1 193 ┴
07119: A3 163 ú
0711A: 02 002 ☻
0711B: 06 006 ♠
0711C: F4 244 ⌠
0711D: 90 144 É
0711E: 90 144 É
0711F: 90 144 É
07120: 90 144 É
07121: 90 144 É
07122: 90 144 É
07123: 90 144 É
07124: 90 144 É
07125: 90 144 É
07126: 90 144 É
07127: 90 144 É
07128: 90 144 É
07129: 90 144 É
0712A: 90 144 É
0712B: 90 144 É
0712C: 90 144 É
```

```
0700:011C
SUB AX, DX
JB 0113h
INC BX
JMP 010Ch
MOV [00600h], BX
ADD AX, CX
MOV [00602h], AX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



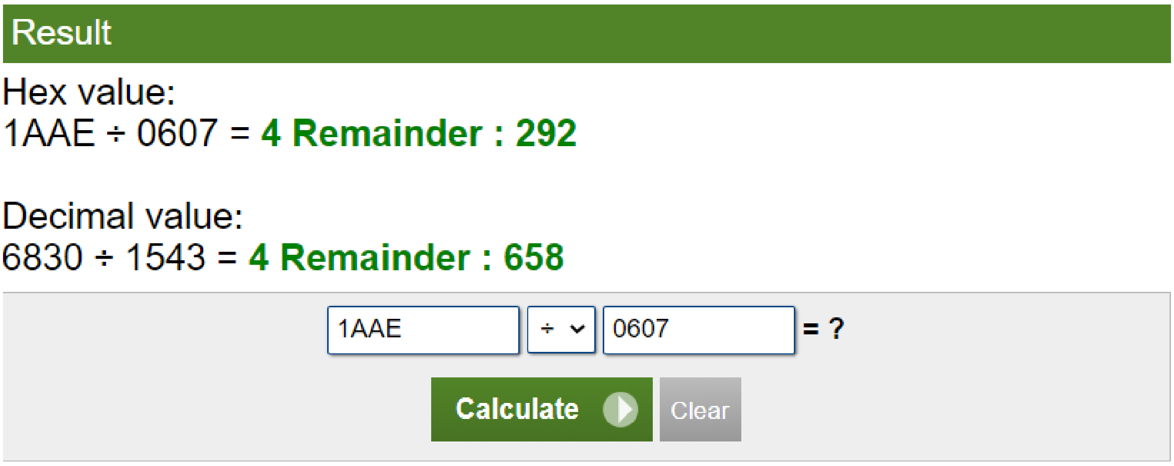
```


org 100h                              ; set location counter to

; add your code here
MOV CX,[0500h]                        ;Load Divisor into CX reg

MOV BX,0000h                          ;Initialize Quotient regis
MOV AX,[0502h]                        ;Load Dividend into AX(acc
MOV DX,CX                             ;MOVE CX content into DX
L1: SUB AX,DX                         ;Subtract DX from AX, and
JC   CARRY                      ;If Carry Flag bit=1,Jump to (
INC BX                          ;If Carry Flag bit=0,Increse (


JMP L1                          ;Jump unconditionally to L1 La
CARRY:                                ;Store the BX content into
MOV [0600h],BX                  ;Add AX and CX and store the r
ADD AX,CX


MOV [0602h],AX                  ;Store the AX content(Remainde

HLT                                   ;Terminate the Program
```

- Result



```
0700:0600

0700:0600  04 00 92 02 00 00 00 00-00 00 00 00 00 00 00 00   ♦.Æ☻............
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- Verification



Result

Hex value:
1AAE ÷ 0607 = 4 Remainder : 292

Decimal value:
6830 ÷ 1543 = 4 Remainder : 658

1AAE ÷ 0607 = ?

- Indeed my result stored in memory location 0600h(**Quotient**) and 0602h(**Remainder**),matches with calculated values as shown above.

Question: -

1.)What you have understood of processors in short, one para ?

Ans. A processor is the logic circuitry that responds to and processes the basic instructions that drive a computer. Thus it’s basic job is to receive input and provide the appropriate output. Nowadays, Processors are found in all modern electronic devices, including PCs, smartphones, tablets, etc. Their purpose is to receive input in the form of program instructions and execute trillions of calculations to provide the output that the user will interface with.
A processor is made of four basic elements:

- The arithmetic logic unit (ALU),
- The floating point unit (FPU),
- Registers
- Cache memories.

The **ALU and FPU** carry basic and advanced arithmetic and logic operations on numbers, and then results are sent to the **registers**, which also store instructions. **Caches** are small and fast memories that store copies of data for frequent use, and act similarly to a random-access memory (RAM).

Basically, a processor carries out his operations through the three main steps of the instruction cycle: fetch, decode, and execute.

- Fetch: The processor retrieves instructions, usually from a RAM.

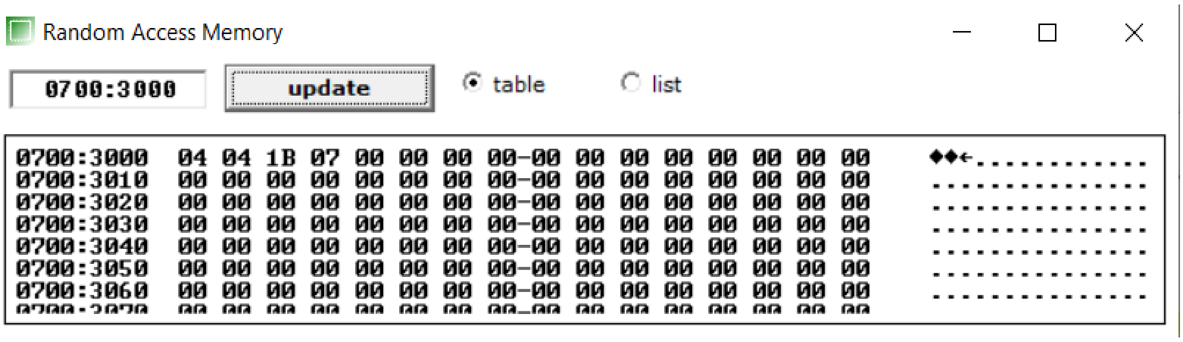
- Decode: A decoder converts the instruction into signals to the other components of processing unit .

- ➢ Execute: The now decoded instructions are sent to each component so that the desired operation can be performed.

2.)What happens when you subtract a-b where b>a and a+bwhen (a+b)> 16bit ? What would be stored in the accumulator

Ans)

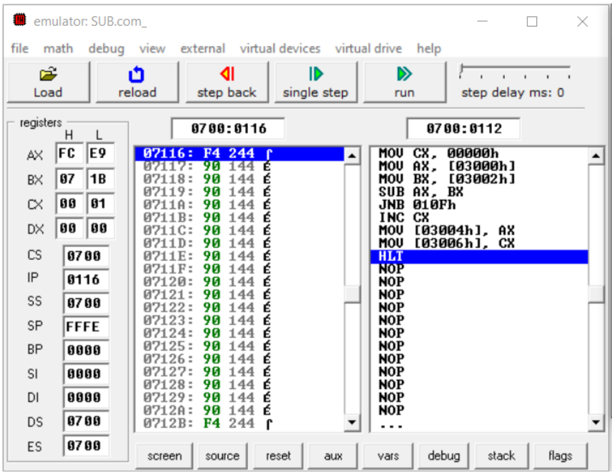
I have taken a=0404h and b as 071Bh,where b>a.



```
0700:3000
0700:3000  04 04 1B 07 00 00 00 00-00 00 00 00 00 00 00 00  ♦♦←.............
0700:3010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0700:3020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0700:3030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0700:3040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0700:3050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0700:3060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0700:3070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Inference: -

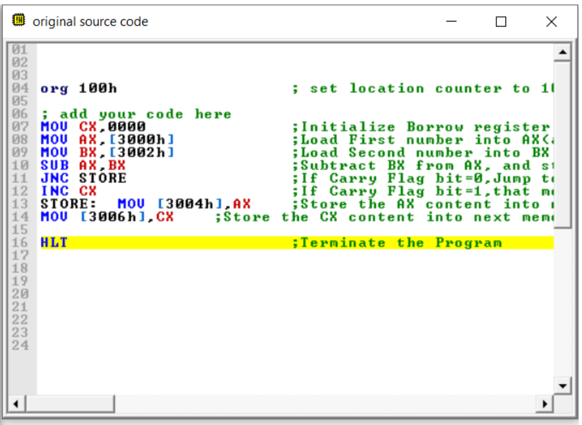
- ➢ After Subtraction Operation, result being stored in Accumulator contains FCE9 i.e, AX= FCE9 ,which shows the difference between the two , taking the borrow and the borrow being saved in CX. We can see borrow=1/CX=1



registers	H	L
AX	FC	E9
BX	07	1B
CX	00	01
DX	00	00
CS	0700	
IP	0116	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

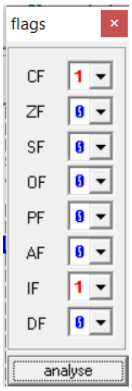
```
0700:0116
07116: F4 244 ſ
07117: 90 144 É
07118: 90 144 É
07119: 90 144 É
0711A: 90 144 É
0711B: 90 144 É
0711C: 90 144 É
0711D: 90 144 É
0711E: 90 144 É
0711F: 90 144 É
07120: 90 144 É
07121: 90 144 É
07122: 90 144 É
07123: 90 144 É
07124: 90 144 É
07125: 90 144 É
07126: 90 144 É
07127: 90 144 É
07128: 90 144 É
07129: 90 144 É
0712A: 90 144 É
0712B: F4 244 ſ
```

```
0700:0112
MOV CX, 00000h
MOV AX, [03000h]
MOV BX, [03002h]
SUB AX, BX
JNB 010Fh
INC CX
MOV [03004h], AX
MOV [03006h], CX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



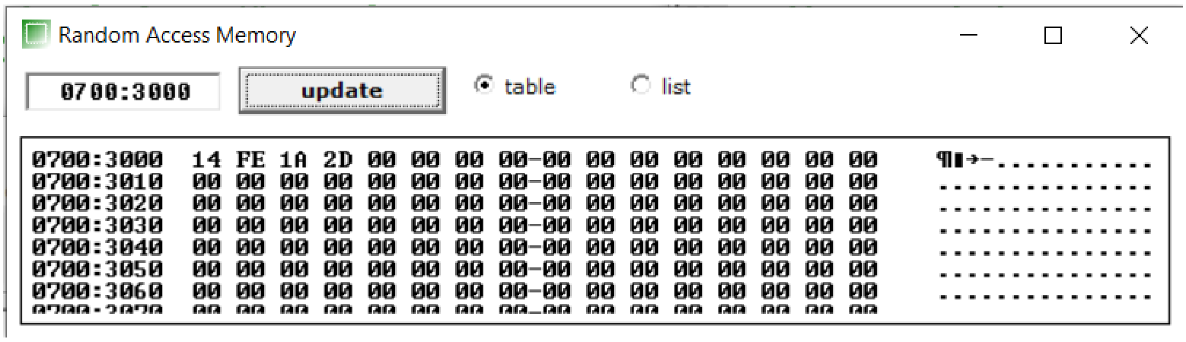
```


org 100h                          ; set location counter to 1(

; add your code here
MOV CX,0000                       ;Initialize Borrow register
MOV AX,[3000h]                    ;Load First number into AX(
MOV BX,[3002h]                    ;Load Second number into BX
SUB AX,BX                         ;Subtract BX from AX, and s
JNC STORE                         ;If Carry Flag bit=0,Jump t
INC CX                            ;If Carry Flag bit=1,that m
STORE:  MOV [3004h],AX            ;Store the AX content into 
MOV [3006h],CX        ;Store the CX content into next mem

HLT                               ;Terminate the Program
```

- ➢ The same can be referred by seeing the FLAG Status. We will observe that the Carry Flag bit is set to 1. Concluding the 1st no saved in AX is smaller than 2nd number saved in BX,and thus an borrow has occurred.



flags	
CF	1
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

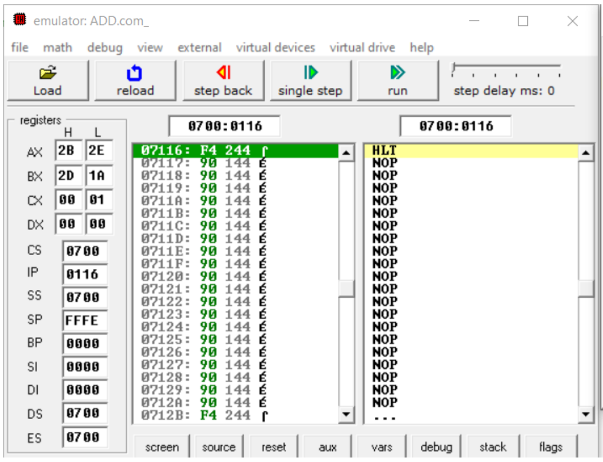
I have taken a = FE14 and b =2D1A, when (a+b)> 16bit,



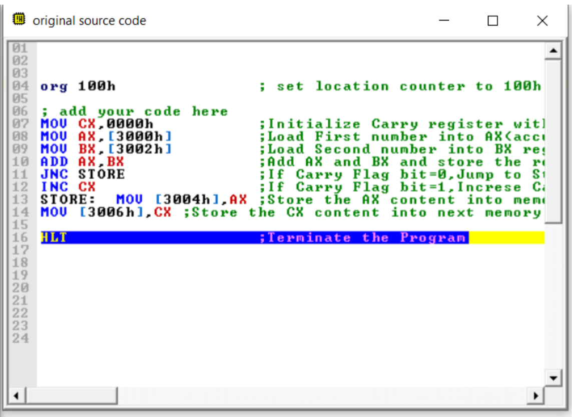
```
0700:3000

0700:3000  14 FE 1A 2D 00 00 00 00-00 00 00 00 00 00 00 00   ¶▮→–............
0700:3010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:3060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Inference: -

- The result shows addition of two number ,being stored in AX with carry,and for carry we have to use another register to store higher bit results.

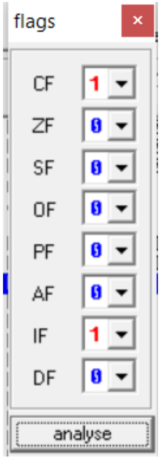


```
      H   L
AX   2B  2E
BX   2D  1A
CX   00  01
DX   00  00
CS   0700
IP   0116
SS   0700
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700

0700:0116                    0700:0116
07116:  F4 244 ſ             HLT
07117:  90 144 É             NOP
07118:  90 144 É             NOP
07119:  90 144 É             NOP
0711A:  90 144 É             NOP
0711B:  90 144 É             NOP
0711C:  90 144 É             NOP
0711D:  90 144 É             NOP
0711E:  90 144 É             NOP
0711F:  90 144 É             NOP
07120:  90 144 É             NOP
07121:  90 144 É             NOP
07122:  90 144 É             NOP
07123:  90 144 É             NOP
07124:  90 144 É             NOP
07125:  90 144 É             NOP
07126:  90 144 É             NOP
07127:  90 144 É             NOP
07128:  90 144 É             NOP
07129:  90 144 É             NOP
0712A:  90 144 É             NOP
0712B:  F4 244 ſ             ...
```



```



org 100h                       ; set location counter to 100h

; add your code here
MOV CX,0000h                   ;Initialize Carry register wit
MOV AX,[3000h]                 ;Load First number into AX(acc
MOV BX,[3002h]                 ;Load Second number into BX re
ADD AX,BX                      ;Add AX and BX and store the r
JNC STORE                      ;If Carry Flag bit=0,Jump to S
INC CX                         ;If Carry Flag bit=1,Increse C
STORE:  MOV [3004h],AX ;Store the AX content into mem
MOV [3006h],CX ;Store the CX content into next memory

HLT                            ;Terminate the Program
```

- Result stored in AX=2B2E and carry is saved in CX.
- The same can be inferred from carry flag, as CF bit is set to 1,which infer that there is overflow.



```
CF  1
ZF  0
SF  0
OF  0
PF  0
AF  0
IF  1
DF  0
```

3.) What are instruction sets ?
•You can make perform a instruction like multiplication using two methods like in a single a go or using repeated addition ?
•How would the size of instruction set affect the performance and price ?
•Look at https://www.computerhope.com/jargon/i/instset.htm

Ans)

An instruction set, is a list of all the commands (instructions), with all their variations, that a processor can execute. It is more or less machine language. The instruction set provides commands to the processor, to tell it what it needs to do.

Instructions include:

- Arithmetic such as add and subtract, etc.
- Logic instructions such as and, or, and not, etc.
- Data instructions such as move, input, output, load, and store, etc.
- Control flow instructions such as goto, if ... goto, call, and others.

3.2)Yes, It has been shown in exp-3 and also single go multiplication instruction is also used in factorial experiment.
3.3) The Size of instruction set greatly affect the performance and price: -

- Like in our experiment, we have used 8086, which is enhanced version of 8085 microprocessor.
- There are many difference between the two, starting from architectural difference's.
- From Instruction set view point,there is also a difference.
  - No multiplication and division instruction in 8085 Microprocessor.
  - Multiplication and Division operations are present
- Also 8086 supports Instruction Queue, Pipelining, Memory Segmentation and many more advantages, which we don't have in 8085.
- ❖ Thus, with enhanced Instruction set, it ease various task and performance is also better, but this comes at higher cost, which we see in the case of 8085 and 8086 too. (The cost of 8085 is low whereas that of 8086 is high)

4.) Can we add 32bit number in a 16bit processor, how, example ?
Ans)
Yes,we can add very easily two 32bit number in a 16bit processor.
The following algorithm shows ,how it can be done: -

<u>Algorithm to Add Two 32 Bit Numbers: -</u>

- Initialize the data segment.
- Load the LSB of first number into AX register.
- Load the MSB of first number into BX register.
- Load the LSB of the second number into CX register.
- Load the MSB of the second number into DX register.
- Add the LSBs of two number.

- Add the MSBs of two numbers along with carry.
- Display the result.
- Stop.

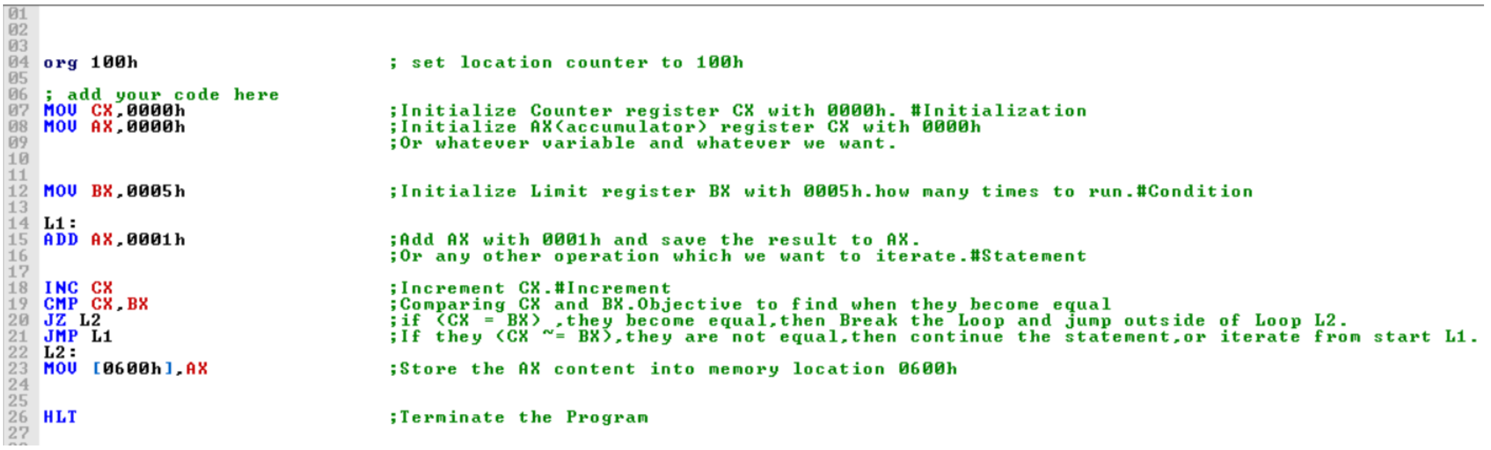
5.) PAE is useful when we have more than one application running, and consuming memory. Like if we have 16GB of RAM on a 32bit machine, without PAE, we'd be able to use only 4GB for all applications, and the OS itself. But with PAE, we could use all the 16GB of RAM, or more.
Physical Address Extension (PAE) is a processor feature that enables x86 processors to access more than 4 GB of physical memory on capable versions of Windows. Certain 32-bit versions of Windows Server running on x86-based systems can use PAE to access up to 64 GB or 128 GB of physical memory, depending on the physical address size of the processor.
With PAE, the operating system moves from **two-level linear address translation to three-level address translation**.
Instead of a linear address ,it is being split into three separate fields for indexing into memory tables, it is split into four separate fields: a 2-bit bitfield, two 9-bit bitfields, and a 12-bit bitfield that corresponds to the page size implemented by Intel architecture (4 KB). The size of page table entries (PTEs) and page directory entries (PDEs) in PAE mode is increased from 32 to 64 bits. The additional bits allow an operating system PTE or PDE to reference physical memory above 4 GB.

Thus, x86 processor(32bit processor) hardware-architecture being augmented with additional address lines, which are used to select the additional memory, so physical address size increases from 32 bits to higher bits. This, theoretically, increases maximum physical memory size above 4 GB.

6.) Following two questions gives a hint how would you make a compiler for a high level language like “C” from assembly language.
•Can you make a custom “For” loop and “while” loop show any example if possible
•Can you make a function using assembly language from using jump statements and other operations.

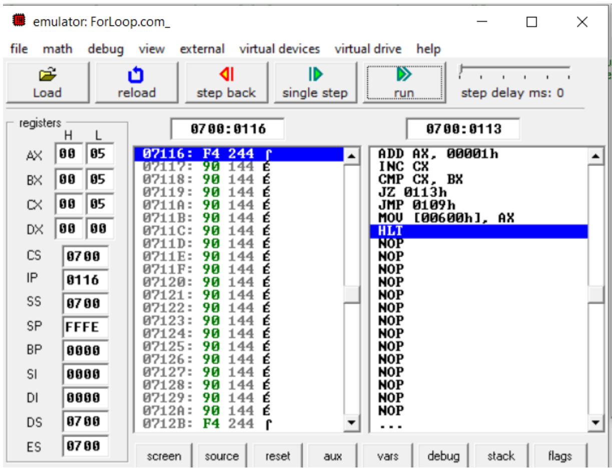
Ans.)
6.1)
Yes we can make a custom “For” loop and “While” loop: -

- For Loop: -
    - Syntax: -
        - for (init; condition; increment) {
        -   statement(s);
        - }
    - Program



```



org 100h                        ; set location counter to 100h

; add your code here
MOV CX,0000h                    ;Initialize Counter register CX with 0000h. #Initialization
MOV AX,0000h                    ;Initialize AX(accumulator) register CX with 0000h
                                ;Or whatever variable and whatever we want.


MOV BX,0005h                    ;Initialize Limit register BX with 0005h.how many times to run.#Condition

L1:
ADD AX,0001h                    ;Add AX with 0001h and save the result to AX.
                                ;Or any other operation which we want to iterate.#Statement

INC CX                          ;Increment CX.#Increment
CMP CX,BX                       ;Comparing CX and BX.Objective to find when they become equal
JZ L2                           ;if (CX = BX) ,they become equal,then Break the Loop and jump outside of Loop L2.
JMP L1                          ;If they (CX ~= BX),they are not equal,then continue the statement,or iterate from start L1.
L2:
MOV [0600h],AX                  ;Store the AX content into memory location 0600h


HLT                             ;Terminate the Program

```

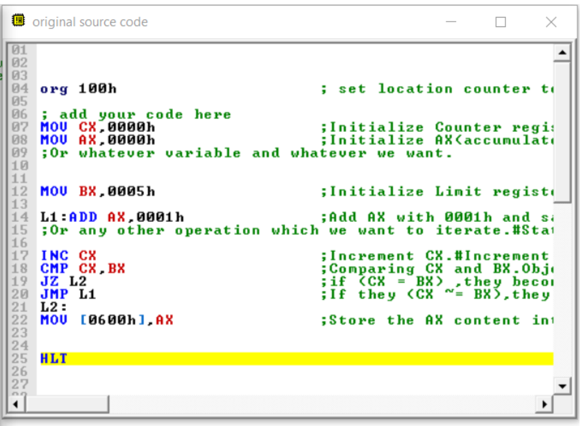
- Result



```
     H  L
AX   00 05
BX   00 05
CX   00 05
DX   00 00
CS   0700
IP   0116
SS   0700
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

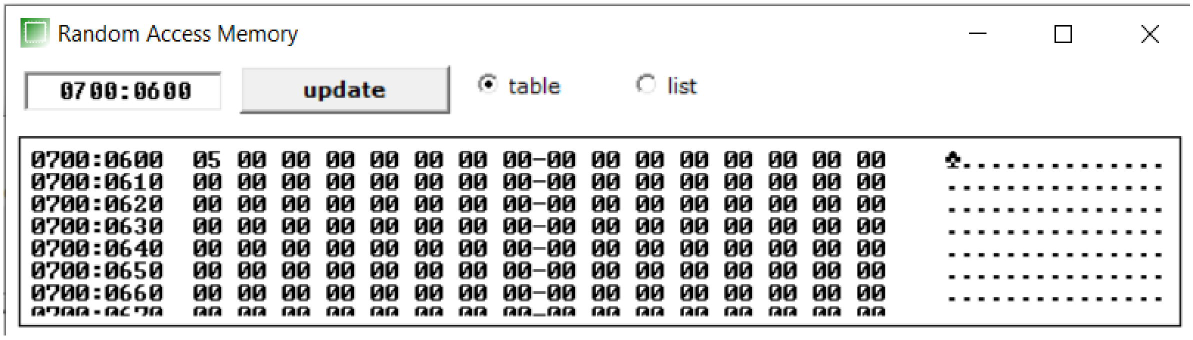
```
0700:0116
07116: F4 244 ⌠
07117: 90 144 É
07118: 90 144 É
07119: 90 144 É
0711A: 90 144 É
0711B: 90 144 É
0711C: 90 144 É
0711D: 90 144 É
0711E: 90 144 É
0711F: 90 144 É
07120: 90 144 É
07121: 90 144 É
07122: 90 144 É
07123: 90 144 É
07124: 90 144 É
07125: 90 144 É
07126: 90 144 É
07127: 90 144 É
07128: 90 144 É
07129: 90 144 É
0712A: 90 144 É
0712B: F4 244 ⌠
```

```
0700:0113
ADD AX, 00001h
INC CX
CMP CX, BX
JZ 0113h
JMP 0109h
MOV [00600h], AX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



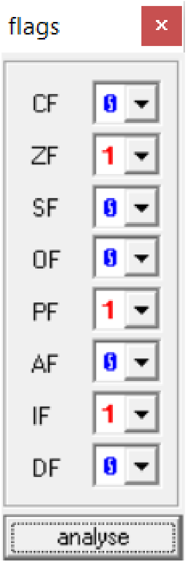
```


org 100h                                 ; set location counter t

; add your code here
MOV CX,0000h                             ;Initialize Counter regi
MOV AX,0000h                             ;Initialize AX(accumulat
;Or whatever variable and whatever we want.


MOV BX,0005h                             ;Initialize Limit regist

L1:ADD AX,0001h                          ;Add AX with 0001h and s
;Or any other operation which we want to iterate.#Sta

INC CX                                   ;Increment CX.#Increment
CMP CX,BX                                ;Comparing CX and BX.Obj
JZ L2                                    ;if (CX = BX) ,they beco
JMP L1                                   ;If they (CX ~= BX),they
L2:
MOV [0600h],AX                           ;Store the AX content in


HLT
```

- Verification



0700:0600 ● table ○ list

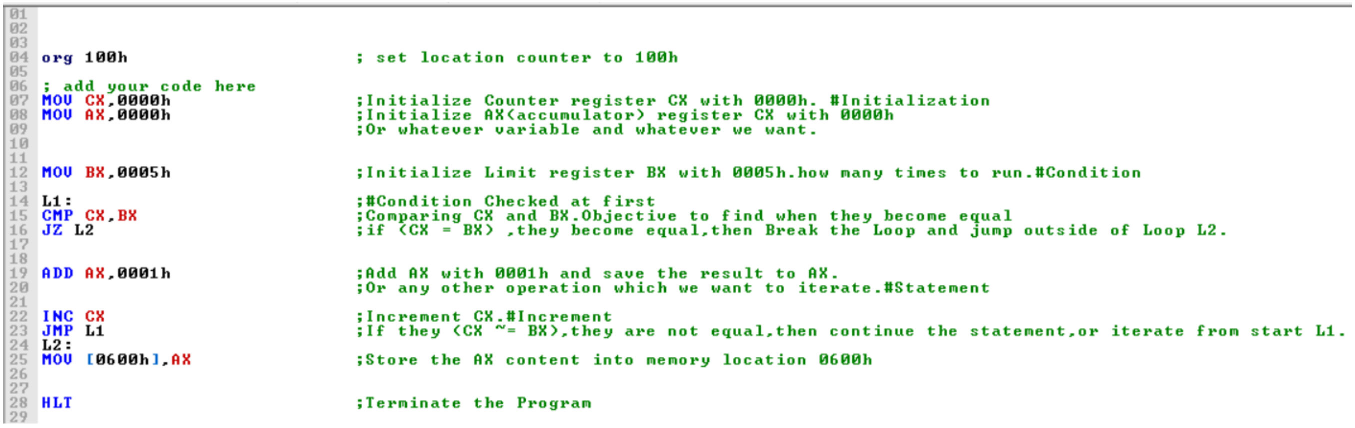
```
0700:0600  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ♣...............
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```



```
CF  0
ZF  1
SF  0
OF  0
PF  1
AF  0
IF  1
DF  0
```

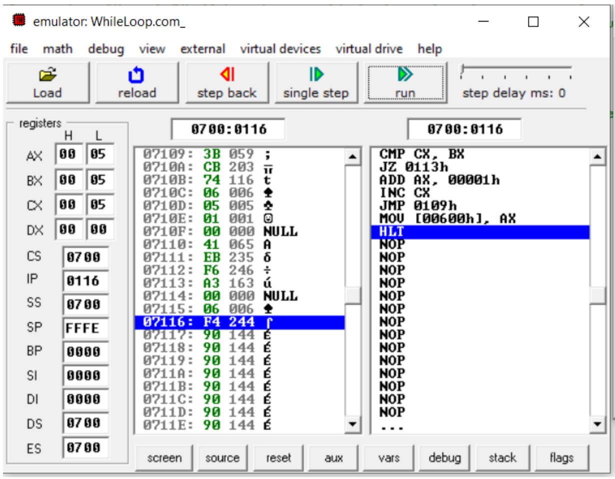
Thus, we can infer from above, that when the condition is met,i.e., when ZF bit is set to high, the loops ends. Through the above program, we added 1 to AX iteratively 5 times and then when condition is met, the program counter went out of loop and then consequent instruction are executed.

- While Loop
    - Syntax
        - while(condition) {
        - statement(s);
        - }
        -
    - Program



```
org 100h                       ; set location counter to 100h

; add your code here
MOV CX,0000h                   ;Initialize Counter register CX with 0000h. #Initialization
MOV AX,0000h                   ;Initialize AX(accumulator) register CX with 0000h
                               ;Or whatever variable and whatever we want.


MOV BX,0005h                   ;Initialize Limit register BX with 0005h.how many times to run.#Condition

L1:                            ;#Condition Checked at first
CMP CX,BX                      ;Comparing CX and BX.Objective to find when they become equal
JZ L2                          ;if (CX = BX) ,they become equal,then Break the Loop and jump outside of Loop L2.


ADD AX,0001h                   ;Add AX with 0001h and save the result to AX.
                               ;Or any other operation which we want to iterate.#Statement

INC CX                         ;Increment CX.#Increment
JMP L1                         ;If they (CX ~= BX),they are not equal,then continue the statement,or iterate from start L1.
L2:
MOV [0600h],AX                 ;Store the AX content into memory location 0600h


HLT                            ;Terminate the Program
```

- Result



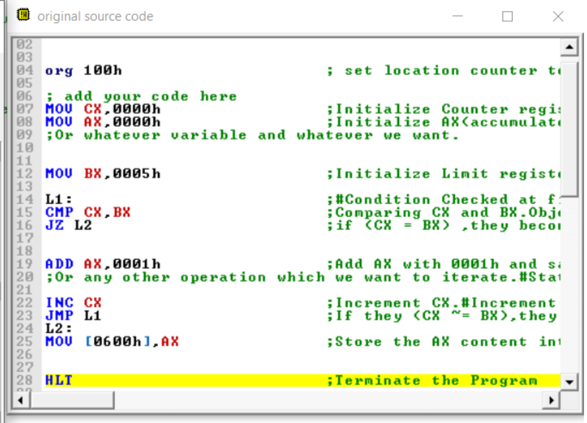
registers

	H	L
AX	00	05
BX	00	05
CX	00	05
DX	00	00

CS	0700
IP	0116
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

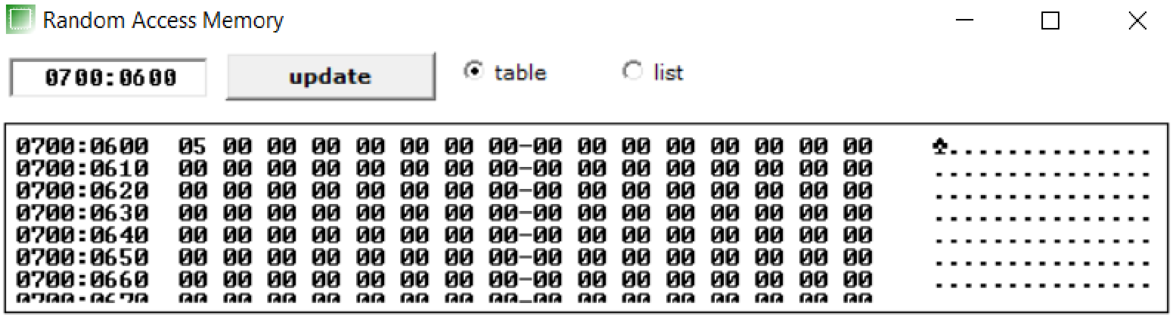
```
0700:0116
07109: 3B 059 ;
0710A: CB 203 ╦
0710B: 74 116 t
0710C: 06 006 ♠
0710D: 05 005 ♣
0710E: 01 001 ☺
0710F: 00 000 NULL
07110: 41 065 A
07111: EB 235 δ
07112: F6 246 ÷
07113: A3 163 ú
07114: 00 000 NULL
07115: 06 006 ♠
07116: F4 244 ⌠
07117: 90 144 É
07118: 90 144 É
07119: 90 144 É
0711A: 90 144 É
0711B: 90 144 É
0711C: 90 144 É
0711D: 90 144 É
0711E: 90 144 É
```

```
0700:0116
CMP CX, BX
JZ 0113h
ADD AX, 00001h
INC CX
JMP 0109h
MOV [00600h], AX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

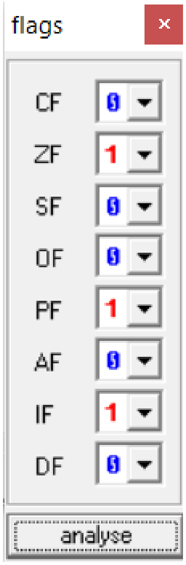


```
org 100h                       ; set location counter t

; add your code here
MOV CX,0000h                   ;Initialize Counter regi
MOV AX,0000h                   ;Initialize AX(accumulat
;Or whatever variable and whatever we want.


MOV BX,0005h                   ;Initialize Limit regist

L1:                            ;#Condition Checked at f
CMP CX,BX                      ;Comparing CX and BX.Obj
JZ L2                          ;if (CX = BX) ,they beco


ADD AX,0001h                   ;Add AX with 0001h and s
;Or any other operation which we want to iterate.#Sta

INC CX                         ;Increment CX.#Increment
JMP L1                         ;If they (CX ~= BX),they
L2:
MOV [0600h],AX                 ;Store the AX content in


HLT                            ;Terminate the Program
```

- Verification



```
0700:0600

0700:0600  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ♣...............
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```



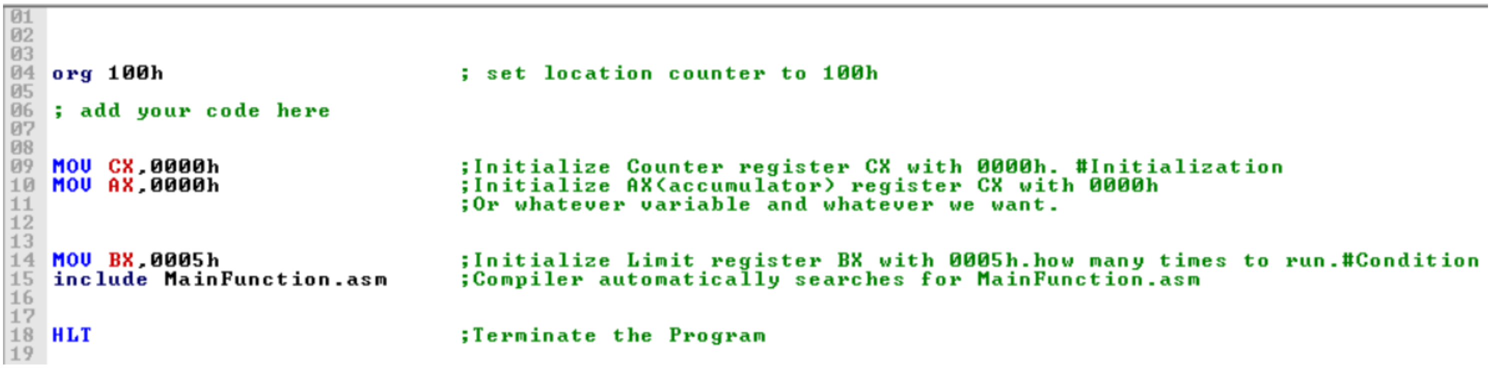
CF 0
ZF 1
SF 0
OF 0
PF 1
AF 0
IF 1
DF 0

Thus, we can infer from above, that the condition is checked at entry of loop and when the condition is met,i.e., when ZF bit is set to high, the loops ends. Through the above program, we added 1 to AX iteratively 5 times and then when condition is met, the program counter went out of loop and then consequent instruction are executed.

6.2)

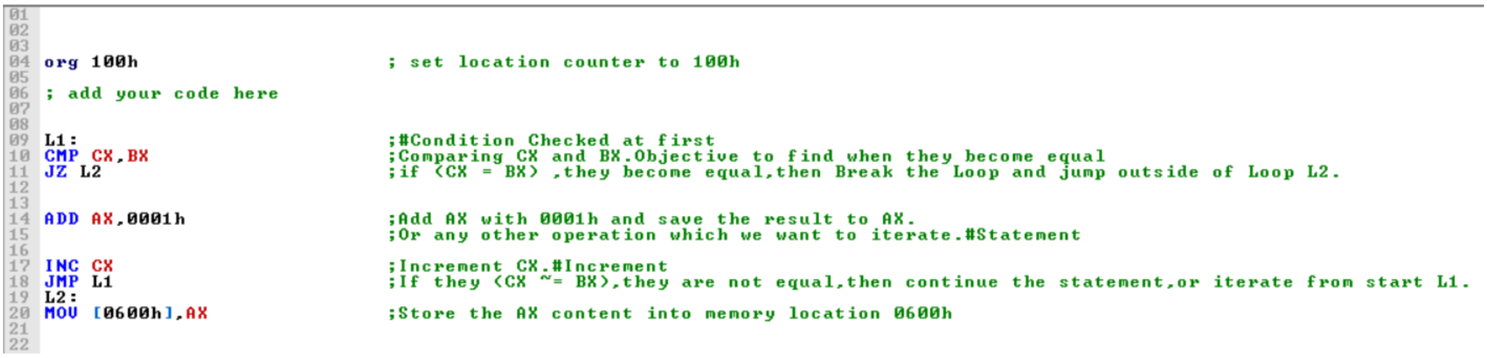
Yes we can make a function in assembly language,as shown below.

- Program(where we call MainFunction.asm) : -



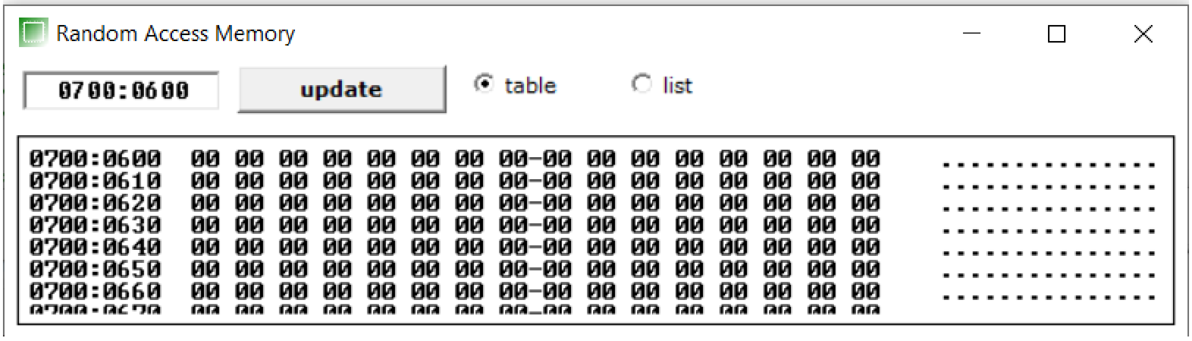
```



org 100h                          ; set location counter to 100h

; add your code here


MOV CX,0000h                      ;Initialize Counter register CX with 0000h. #Initialization
MOV AX,0000h                      ;Initialize AX(accumulator) register CX with 0000h
                                  ;Or whatever variable and whatever we want.


MOV BX,0005h                      ;Initialize Limit register BX with 0005h.how many times to run.#Condition
include MainFunction.asm          ;Compiler automatically searches for MainFunction.asm


HLT                               ;Terminate the Program

```

- Program(MainFunction.asm): -



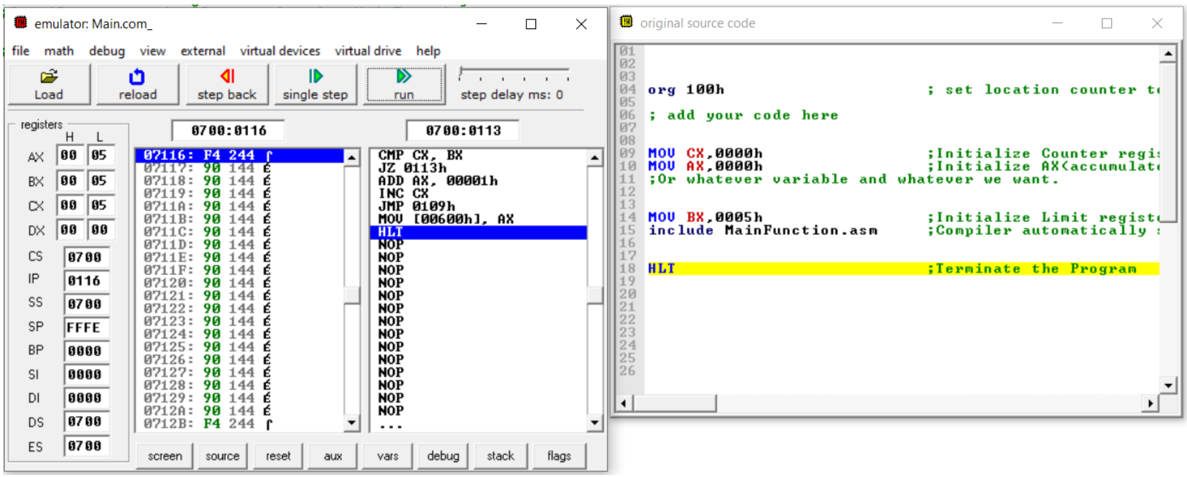
```



org 100h                          ; set location counter to 100h

; add your code here


L1:                               ;#Condition Checked at first
CMP CX,BX                         ;Comparing CX and BX.Objective to find when they become equal
JZ L2                             ;if (CX = BX) ,they become equal,then Break the Loop and jump outside of Loop L2.


ADD AX,0001h                      ;Add AX with 0001h and save the result to AX.
                                  ;Or any other operation which we want to iterate.#Statement

INC CX                            ;Increment CX.#Increment
JMP L1                            ;If they (CX ~= BX),they are not equal,then continue the statement,or iterate from start L1.
L2:
MOV [0600h],AX                    ;Store the AX content into memory location 0600h


```

- Before running Main.asm, content of Memory location 0600h



```
0700:0600

0700:0600  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

- Execution



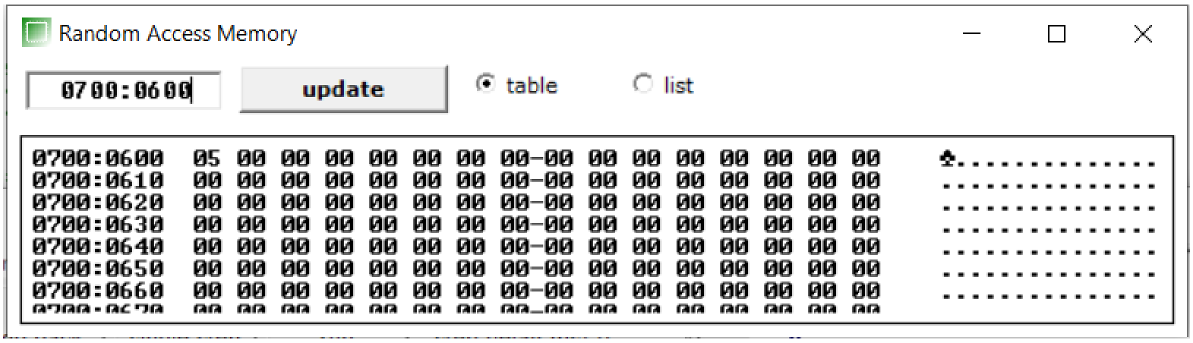
```
registers   H   L
AX          00  05
BX          00  05
CX          00  05
DX          00  00
CS          0700
IP          0116
SS          0700
SP          FFFE
BP          0000
SI          0000
DI          0000
DS          0700
ES          0700
```

```
0700:0116

07116: F4 244 ſ
07117: 90 144 É
07118: 90 144 É
07119: 90 144 É
0711A: 90 144 É
0711B: 90 144 É
0711C: 90 144 É
0711D: 90 144 É
0711E: 90 144 É
0711F: 90 144 É
07120: 90 144 É
07121: 90 144 É
07122: 90 144 É
07123: 90 144 É
07124: 90 144 É
07125: 90 144 É
07126: 90 144 É
07127: 90 144 É
07128: 90 144 É
07129: 90 144 É
0712A: 90 144 É
0712B: F4 244 ſ
```

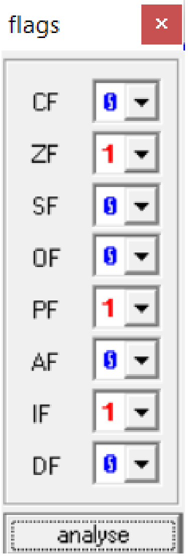
```
0700:0113

CMP CX, BX
JZ 0113h
ADD AX, 00001h
INC CX
JMP 0109h
MOV [00600h], AX
HLT
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

```



org 100h                           ; set location counter t

; add your code here


MOV CX,0000h                       ;Initialize Counter regi
MOV AX,0000h                       ;Initialize AX(accumulat
;Or whatever variable and whatever we want.


MOV BX,0005h                       ;Initialize Limit regist
include MainFunction.asm           ;Compiler automatically 


HLT                                ;Terminate the Program
```

- Result



```
0700:0600

0700:0600  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ♣...............
0700:0610  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0620  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0630  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0640  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0650  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0660  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0700:0670  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```



CF 0
ZF 1
SF 0
OF 0
PF 1
AF 0
IF 1
DF 0

- Inference: -
  - We run the Main.asm file, where we just initialized the three registers, namely AX, BX, CX and after initialization, we called the MainFunction.asm file, where we wrote the FOR-loop operation, using jump statements and other operations and thus we see the changes in memory location 0600h and other registers and the no of times the loop has been run is inferred from CX content. After executing this file and it then returned back to Main.asm file ,where it halted the processor.

- ✓ To view the program, click on below link: -
  - Program Files