Practical-1

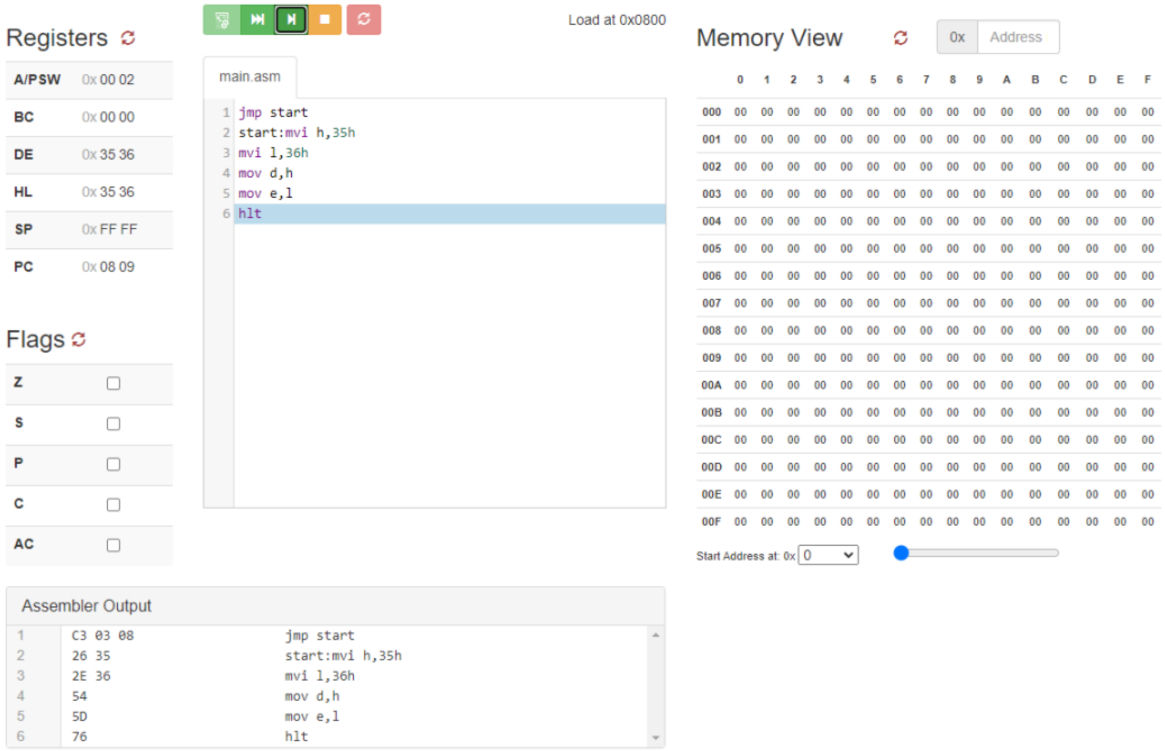
❖ Transfer operation for 8085

1. Write a program to load values 35H and 36H in register H and L then copy in register D and E.

INPUT:

```
jmp start
Start:mvi h,35h
Mvi l,36h
Mov d,h
Mov e,l
hlt
```

OUTPUT:



Registers

A/PSW	0x 00 02
BC	0x 00 00
DE	0x 35 36
HL	0x 35 36
SP	0x FF FF
PC	0x 08 09

Flags

Z	□
S	□
P	□
C	□
AC	□

Load at 0x0800

main.asm

```
jmp start
start:mvi h,35h
mvi l,36h
mov d,h
mov e,l
hlt
```

Memory View

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

Assembler Output

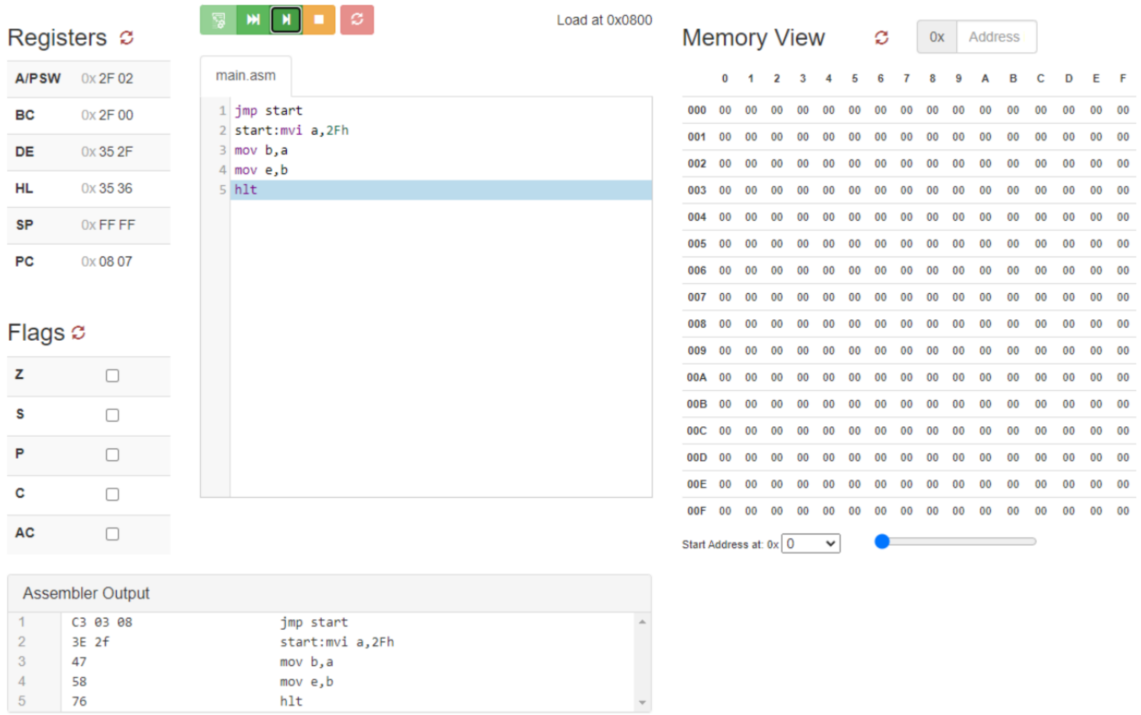
```
C3 03 08        jmp start
26 35           start:mvi h,35h
2E 36           mvi l,36h
54              mov d,h
5D              mov e,l
76              hlt
```

2. write a program to copy register with value 2FH then copy register A to B and B to E.

INPUT:

jmp start
Start:mvi a,2Fh
Mov b,a
Mov e,b
Hlt

OUTPUT:



Registers

A/PSW	0x 2F 02
BC	0x 2F 00
DE	0x 35 2F
HL	0x 35 36
SP	0x FF FF
PC	0x 08 07

Flags

Z	□
S	□
P	□
C	□
AC	□

Load at 0x0800

main.asm

```
jmp start
start:mvi a,2Fh
mov b,a
mov e,b
hlt
```

Memory View

0x Address

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

Assembler Output

```
C3 03 08                  jmp start
3E 2f                     start:mvi a,2Fh
47                        mov b,a
58                        mov e,b
76                        hlt
```

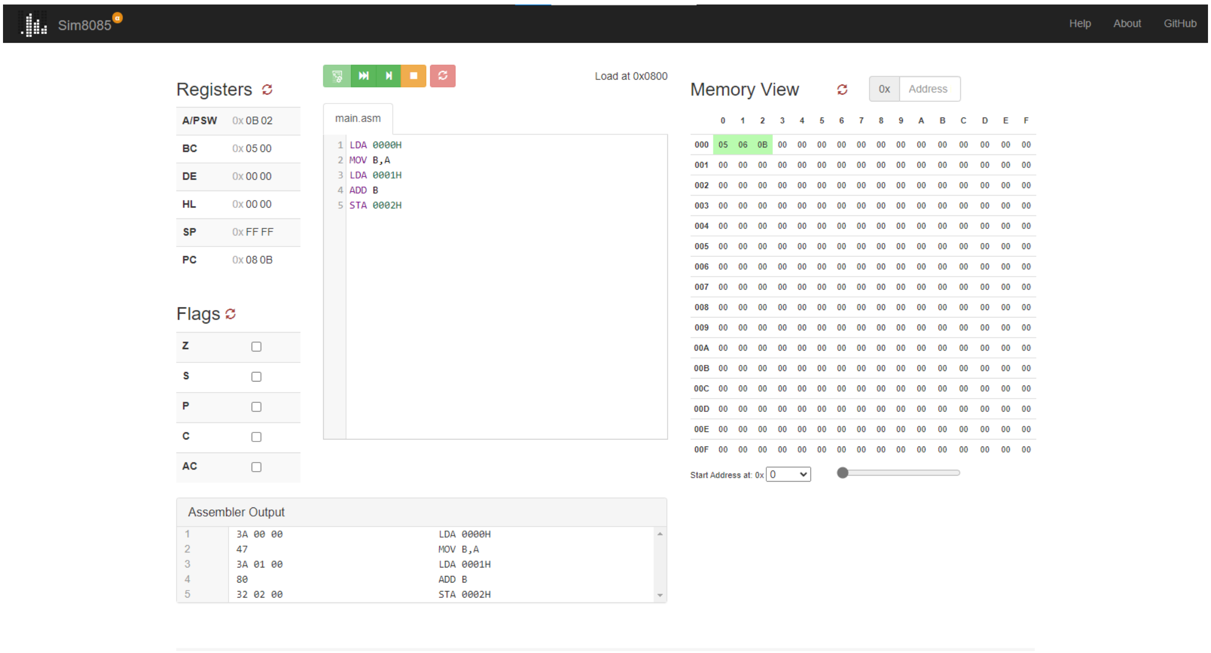
Practical-2

1. Objective : Addition of two 8-bit numbers.

INPUT:

```
LDA 0000H
MOV B,A
LDA 0001H
ADD B
STA 0002H
```

OUTPUT:



Registers

A/PSW	0x 0B 02
BC	0x 05 00
DE	0x 00 00
HL	0x 00 00
SP	0x FF FF
PC	0x 08 0B

Flags

Z	□
S	□
P	□
C	□
AC	□

```
LDA 0000H
MOV B,A
LDA 0001H
ADD B
STA 0002H
```

Memory View

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	05	06	0B	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

Assembler Output

```
3A 00 00                    LDA 0000H
47                          MOV B,A
3A 01 00                    LDA 0001H
80                          ADD B
32 02 00                    STA 0002H
```

Conclusion:

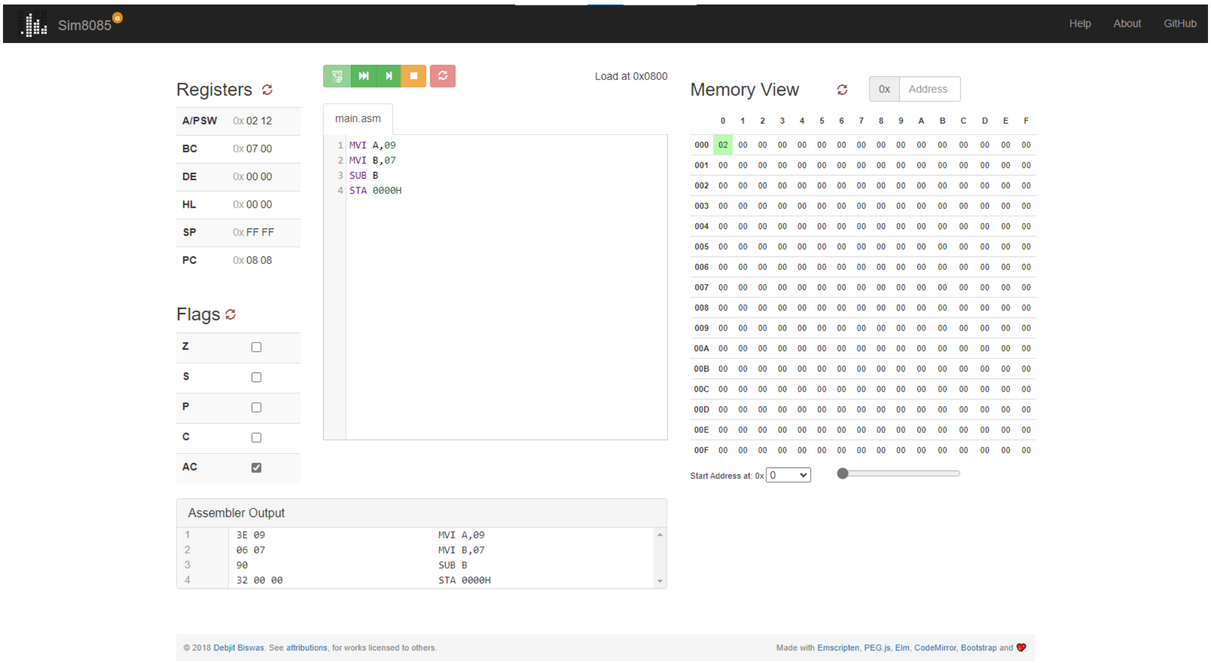
➔ In this practical, we learn about the LDA,MOV,ADD and STA instruction.Moreover how to add two numbers and store it in a different memory location.

2. Objective: Subtraction of two 8-bit numbers.

INPUT:

```
MVI A,09
MVI B,07
SUB B
STA 0000H
```

OUTPUT:



Registers

A/PSW	0x 02 12
BC	0x 07 00
DE	0x 00 00
HL	0x 00 00
SP	0x FF FF
PC	0x 08 08

Flags

Z	□
S	□
P	□
C	□
AC	☑

```
MVI A,09
MVI B,07
SUB B
STA 0000H
```

Memory View

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

Assembler Output

```
3E 09                          MVI A,09
06 07                          MVI B,07
90                             SUB B
32 00 00                       STA 0000H
```

© 2018 Debjit Biswas. See attributions, for works licensed to others.

Made with Emscripten, PEG.js, Elm, CodeMirror, Bootstrap and ♥

Conclusion:

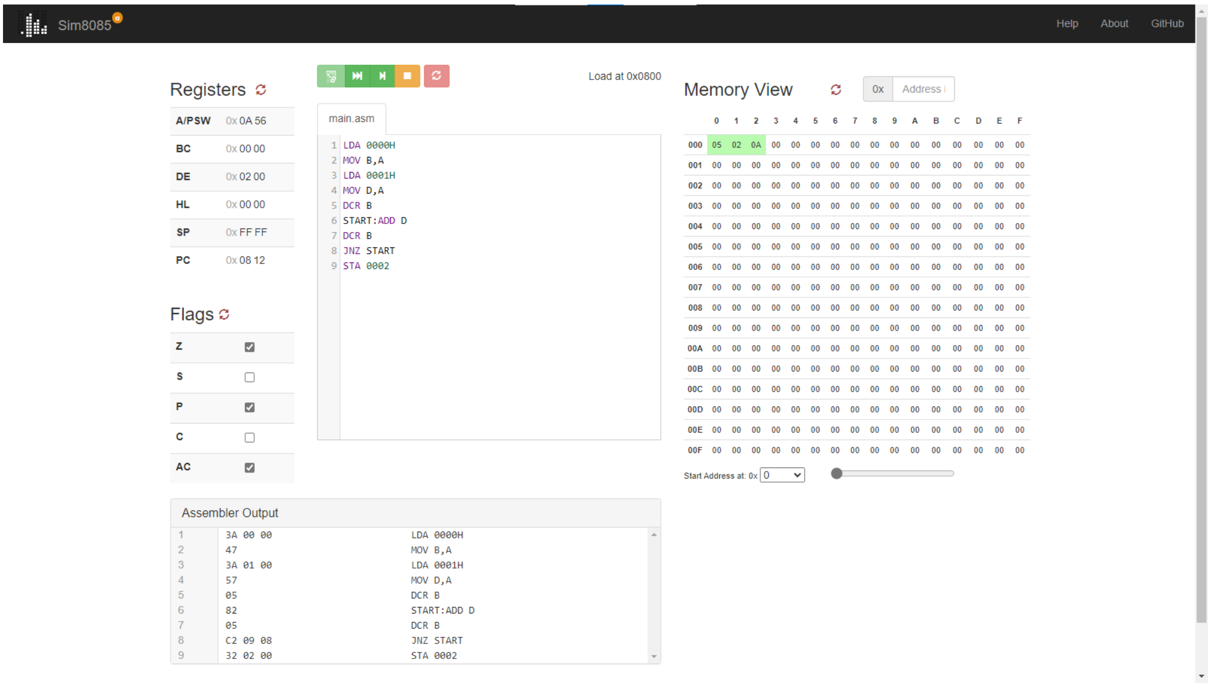
➔ In this practical, we learn how to subtract two numbers stored in memory location. We also learn how to store the result at a new memory location.

3. Objective: Multiplication of two 8- bit nos. using repeated Addition.

INPUT:

```
LDA 0000H
MOV B,A
LDA 0001H
MOV D,A
DCR B
START:ADD D
DCR B
JNZ START
STA 0002
```

OUTPUT:



Registers	
A/PSW	0x 0A 56
BC	0x 00 00
DE	0x 02 00
HL	0x 00 00
SP	0x FF FF
PC	0x 08 12

Flags	
Z	☑
S	☐
P	☑
C	☐
AC	☑

```
LDA 0000H
MOV B,A
LDA 0001H
MOV D,A
DCR B
START:ADD D
DCR B
JNZ START
STA 0002
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	05	02	0A	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

Assembler Output

```
1    3A 00 00        LDA 0000H
2    47              MOV B,A
3    3A 01 00        LDA 0001H
4    57              MOV D,A
5    05              DCR B
6    82              START:ADD D
7    05              DCR B
8    C2 09 08        JNZ START
9    32 02 00        STA 0002
```

Conclusion:

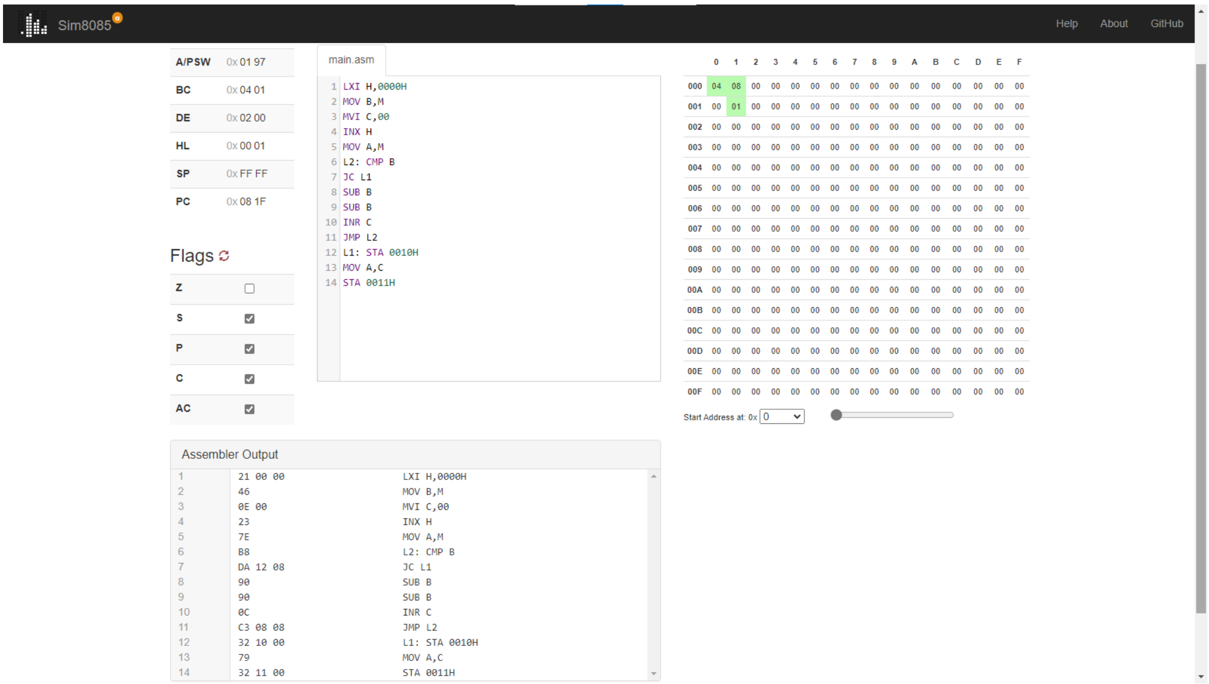
In this practical we learn multiplication of two 8-bit numbers using LDA and MOV instruction.

4. Objective: Division of two 8- bit nos. using repeated Subtraction

INPUT:

```
LXI H,0000H
MOV B,M
MVI C,00
INX H
MOV A,M
L2: CMP B
JC L1
SUB B
SUB B
INR C
JMP L2
L1: STA 0010H
MOV A,C
STA 0011H
```

OUTPUT:



A/PSW	0x 01 97
BC	0x 04 01
DE	0x 02 00
HL	0x 00 01
SP	0x FF FF
PC	0x 08 1F

Flags

Z	☐
S	☑
P	☑
C	☑
AC	☑

main.asm

```
LXI H,0000H
MOV B,M
MVI C,00
INX H
MOV A,M
L2: CMP B
JC L1
SUB B
SUB B
INR C
JMP L2
L1: STA 0010H
MOV A,C
STA 0011H
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	04	08	00	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

Assembler Output

```
21 00 00                LXI H,0000H
46                      MOV B,M
0E 00                   MVI C,00
23                      INX H
7E                      MOV A,M
B8                      L2: CMP B
DA 12 08                JC L1
90                      SUB B
90                      SUB B
0C                      INR C
C3 08 08                JMP L2
32 10 00                L1: STA 0010H
79                      MOV A,C
32 11 00                STA 0011H
```

Conclusion:

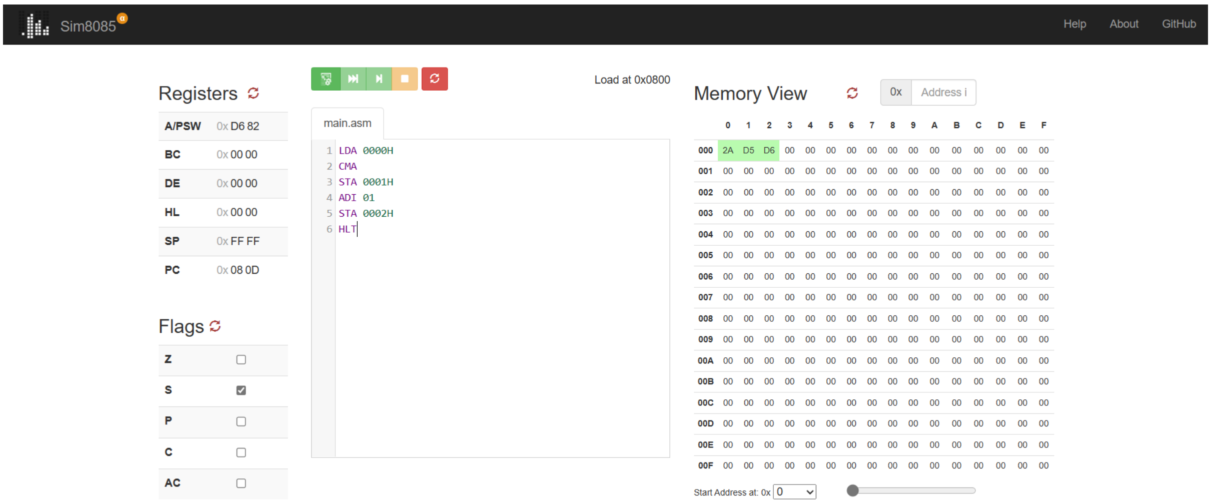
➔ In this practical, we learn how to do the division operation using multiple subtraction.

5. Objective: Find 1's & 2's complement of an 8 bit number.

INPUT:

```
LDA 0000H
CMA
STA 0001H
ADI 01
STA 0002H
HLT
```

OUTPUT:



Registers

A/PSW	0x D6 82
BC	0x 00 00
DE	0x 00 00
HL	0x 00 00
SP	0x FF FF
PC	0x 08 0D

```
LDA 0000H
CMA
STA 0001H
ADI 01
STA 0002H
HLT
```

Flags

Z	□
S	☑
P	□
C	□
AC	□

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	2A	D5	D6	00	00	00	00	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

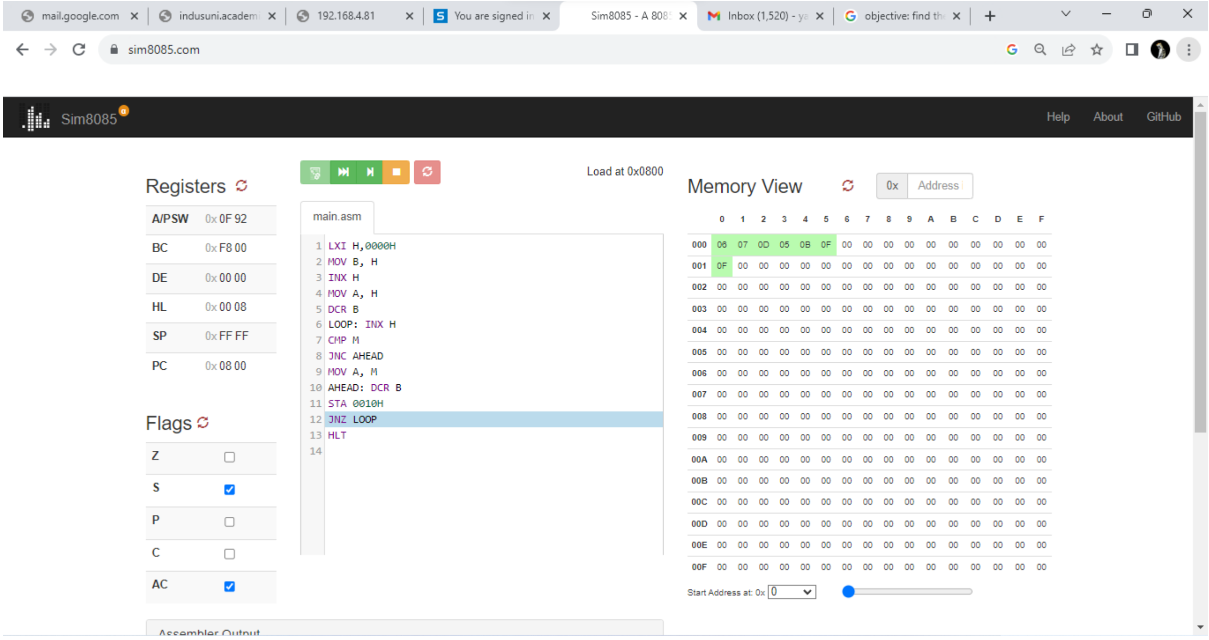
Start Address at: 0x 0

6. Objective: Find largest Number From an array.

INPUT:

LXI H,0000H
MOV B, H
INX H
MOV A, H
DCR B
LOOP: INX H
CMP M
JNC AHEAD
MOV A, M
AHEAD: DCR B
STA 0010H
JNZ LOOP
HLT

OUTPUT:



Registers

A/PSW	0x 0F 92
BC	0x F8 00
DE	0x 00 00
HL	0x 00 08
SP	0x FF FF
PC	0x 08 00

Flags

Z	□
S	☑
P	□
C	□
AC	☑

```
LXI H,0000H
MOV B, H
INX H
MOV A, H
DCR B
LOOP: INX H
CMP M
JNC AHEAD
MOV A, M
AHEAD: DCR B
STA 0010H
JNZ LOOP
HLT
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	06	07	0D	05	0B	0F	00	00	00	00	00	00	00	00	00	00
001	0F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

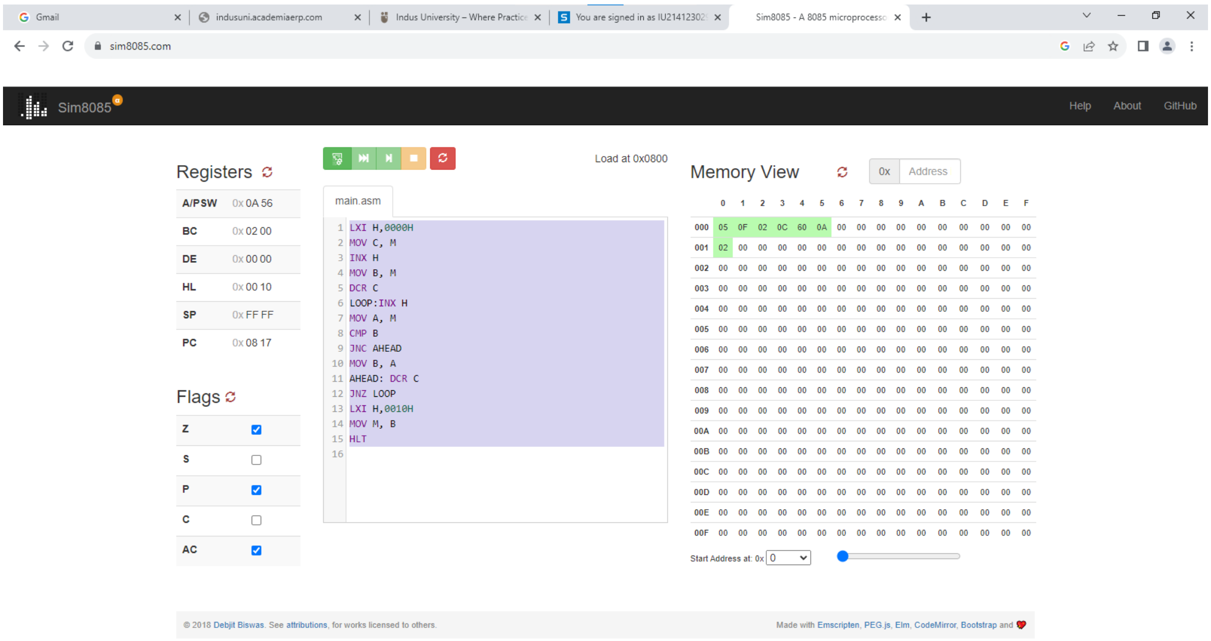
Start Address at: 0x 0

7. Objective: Find smallest No. from an array.

INPUT:

```
LXI H,0000H
MOV C, M
INX H
MOV B, M
DCR C
LOOP:INX H
MOV A, M
CMP B
JNC AHEAD
MOV B, A
AHEAD: DCR C
JNZ LOOP
LXI H,0010H
MOV M, B
HLT
```

OUTPUT:



Registers

A/PSW	0x 0A 56
BC	0x 02 00
DE	0x 00 00
HL	0x 00 10
SP	0x FF FF
PC	0x 08 17

Flags

Z	☑
S	☐
P	☑
C	☐
AC	☑

```
LXI H,0000H
MOV C, M
INX H
MOV B, M
DCR C
LOOP:INX H
MOV A, M
CMP B
JNC AHEAD
MOV B, A
AHEAD: DCR C
JNZ LOOP
LXI H,0010H
MOV M, B
HLT
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	05	0F	02	0C	60	0A	00	00	00	00	00	00	00	00	00	00
001	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

© 2018 Debjit Biswas. See attributions, for works licensed to others.

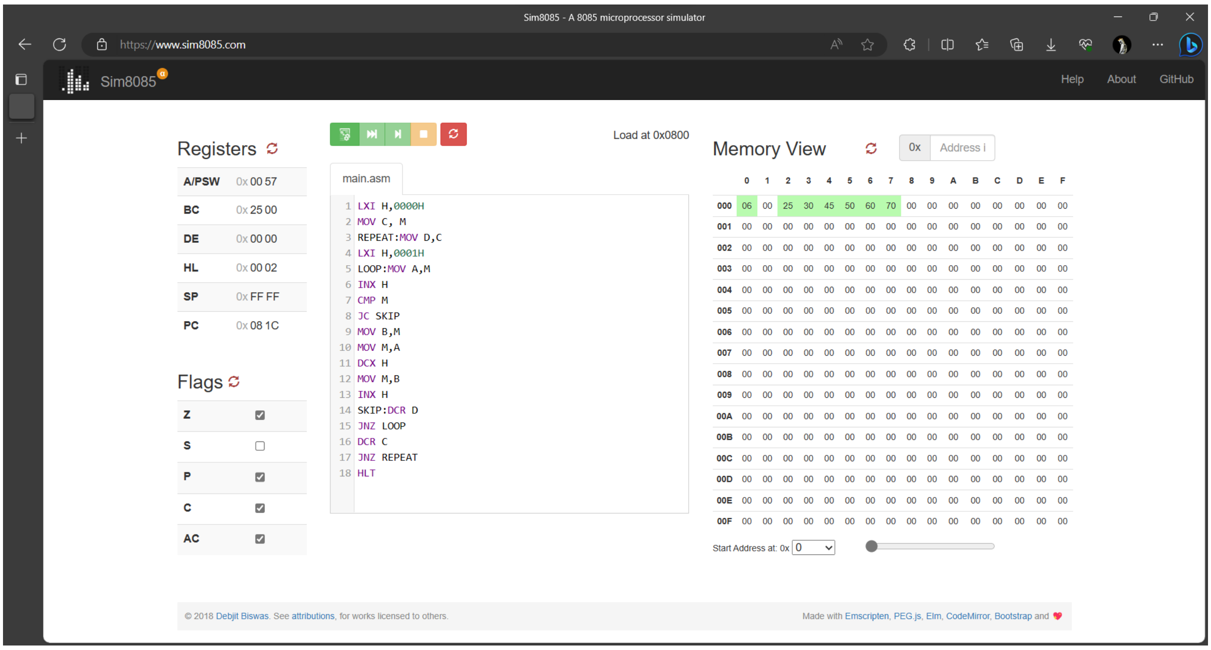
Made with Emscripten, PEG.js, Elm, CodeMirror, Bootstrap and ❤

8. Objective: Arrange data bytes in ascending order.

INPUT:

```
LXI H,0000H
MOV C, M
REPEAT:MOV D,C
LXI H,0001H
LOOP:MOV A,M
INX H
CMP M
JC SKIP
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H
SKIP:DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT
```

OUTPUT:



Registers

A/PSW	0x 00 57
BC	0x 25 00
DE	0x 00 00
HL	0x 00 02
SP	0x FF FF
PC	0x 08 1C

Flags

Z	☑
S	□
P	☑
C	☑
AC	☑

```
LXI H,0000H
MOV C, M
REPEAT:MOV D,C
LXI H,0001H
LOOP:MOV A,M
INX H
CMP M
JC SKIP
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H
SKIP:DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	06	00	25	30	45	50	60	70	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

© 2018 Debjit Biswas. See attributions, for works licensed to others.

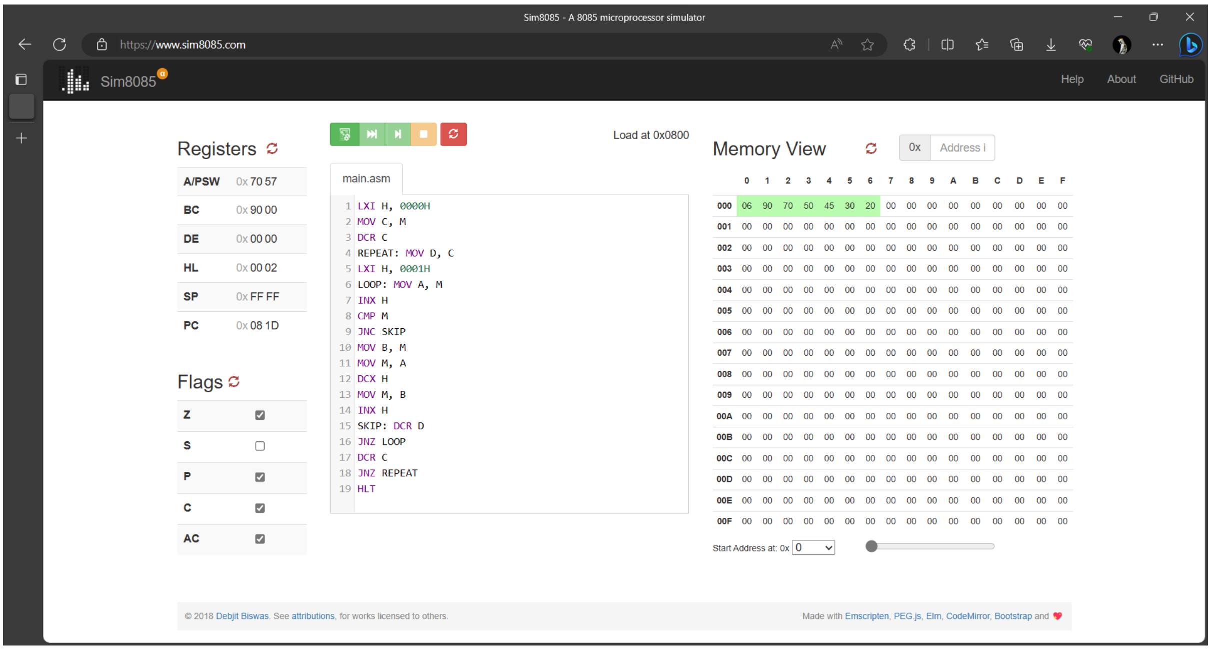
Made with Emscripten, PEG.js, Elm, CodeMirror, Bootstrap and ❤

9. Objective: Arrange data bytes in descending order.

INPUT:

```
LXI H,0000H
MOV C, M
REPEAT:MOV D,C
LXI H,0001H
LOOP:MOV A,M
INX H
CMP M
JC SKIP
MOV B,M
MOV M,A
DCX H
MOV M,B
INX H
SKIP:DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT
```

OUTPUT:



Registers

A/PSW	0x 70 57
BC	0x 90 00
DE	0x 00 00
HL	0x 00 02
SP	0x FF FF
PC	0x 08 1D

Flags

Z	☑
S	□
P	☑
C	☑
AC	☑

Load at 0x0800

main.asm

```
LXI H, 0000H
MOV C, M
DCR C
REPEAT: MOV D, C
LXI H, 0001H
LOOP: MOV A, M
INX H
CMP M
JNC SKIP
MOV B, M
MOV M, A
DCX H
MOV M, B
INX H
SKIP: DCR D
JNZ LOOP
DCR C
JNZ REPEAT
HLT
```

Memory View

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	06	90	70	50	45	30	20	00	00	00	00	00	00	00	00	00
001	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
002	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
003	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
004	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
005	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
006	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
007	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
008	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
009	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00A	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00B	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00D	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00F	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Start Address at: 0x 0

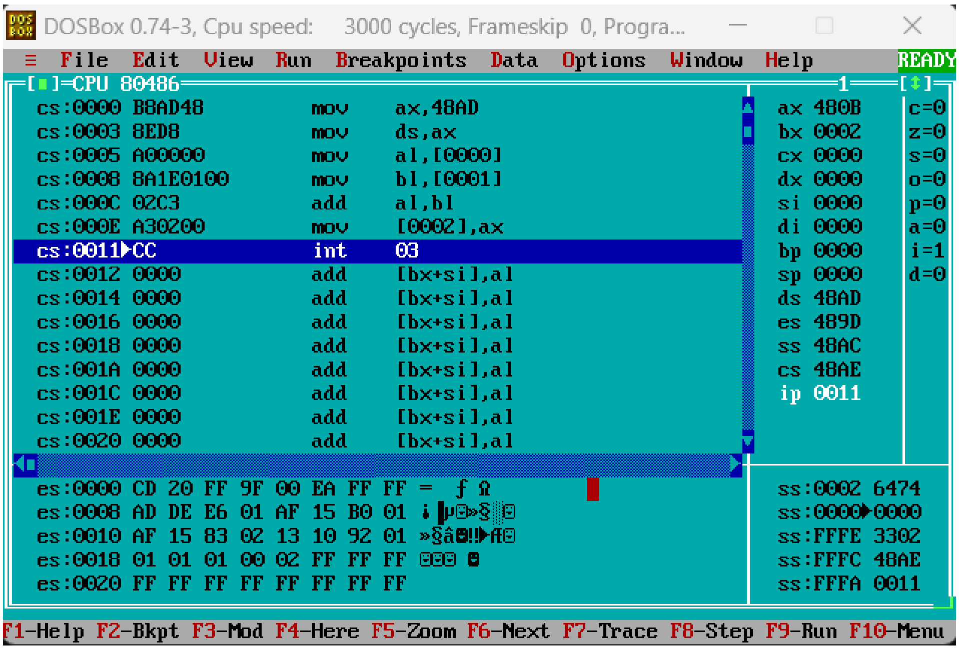
❖ Introduction to MASM:

1. A) TO PERFORM ADDITION OPERATION ON 8-BIT DATA

INPUT:

```
data segment
a db 09h
b db 02h
c dw ?
data ends

code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
mov al,a
mov bl,b
add al,bl
mov c,ax
int 3
code ends
end start
```

OUTPUT:


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
≡ File Edit View Run Breakpoints Data Options Window Help READY
[■]=CPU 80486
1
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

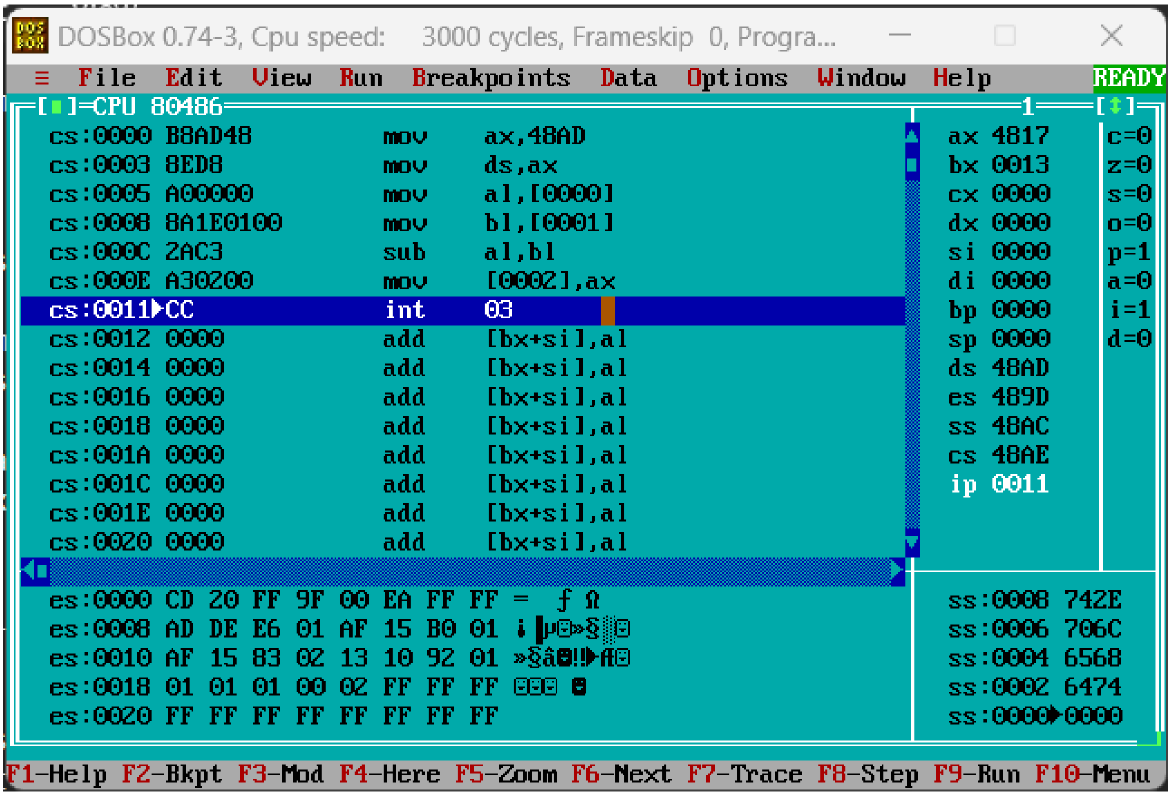
```
cs:0000 B8AD48        mov   ax,48AD         ax 480B   c=0
cs:0003 8ED8          mov   ds,ax           bx 0002   z=0
cs:0005 A00000        mov   al,[0000]       cx 0000   s=0
cs:0008 8A1E0100      mov   bl,[0001]       dx 0000   o=0
cs:000C 02C3          add   al,bl           si 0000   p=0
cs:000E A30200        mov   [0002],ax       di 0000   a=0
cs:0011►CC            int   03              bp 0000   i=1
cs:0012 0000          add   [bx+si],al      sp 0000   d=0
cs:0014 0000          add   [bx+si],al      ds 48AD
cs:0016 0000          add   [bx+si],al      es 489D
cs:0018 0000          add   [bx+si],al      ss 48AC
cs:001A 0000          add   [bx+si],al      cs 48AE
cs:001C 0000          add   [bx+si],al      ip 0011
cs:001E 0000          add   [bx+si],al
cs:0020 0000          add   [bx+si],al

es:0000 CD 20 FF 9F 00 EA FF FF             ss:0002 6474
es:0008 AD DE E6 01 AF 15 B0 01             ss:0000►0000
es:0010 AF 15 83 02 13 10 92 01             ss:FFFE 3302
es:0018 01 01 01 00 02 FF FF FF             ss:FFFC 48AE
es:0020 FF FF FF FF FF FF FF FF             ss:FFFA 0011
```

1. B) TO PERFORM SUBTRACTION OPERATION ON 8-BIT DATA.

INPUT:

```
data segment
a db 2Ah
b db 13h
c dw ?
data ends

code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax
mov al,a
mov bl,b
sub al,bl
mov c,ax
int 3
code ends
end start
```

OUTPUT:



DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
≡ File Edit View Run Breakpoints Data Options Window Help READY
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

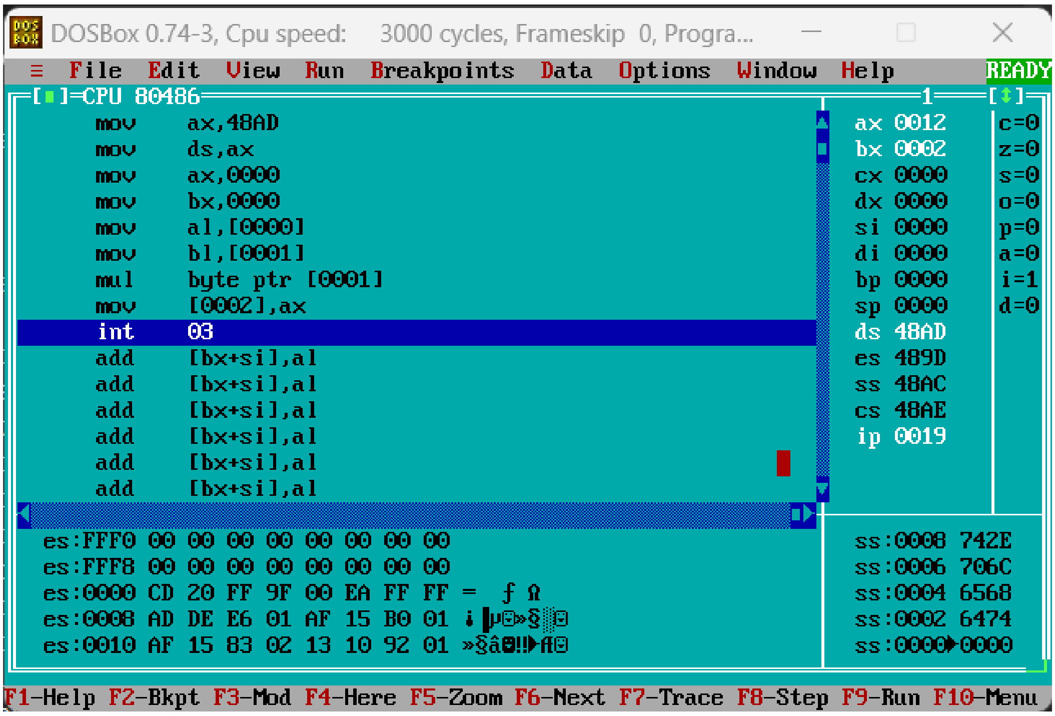
```
[■]=CPU 80486=========================================1====[↕]=
 cs:0000 B8AD48          mov    ax,48AD           ax 4817   c=0
 cs:0003 8ED8            mov    ds,ax             bx 0013   z=0
 cs:0005 A00000          mov    al,[0000]         cx 0000   s=0
 cs:0008 8A1E0100        mov    bl,[0001]         dx 0000   o=0
 cs:000C 2AC3            sub    al,bl             si 0000   p=1
 cs:000E A30200          mov    [0002],ax         di 0000   a=0
 cs:0011►CC              int    03                bp 0000   i=1
 cs:0012 0000            add    [bx+si],al        sp 0000   d=0
 cs:0014 0000            add    [bx+si],al        ds 48AD
 cs:0016 0000            add    [bx+si],al        es 489D
 cs:0018 0000            add    [bx+si],al        ss 48AC
 cs:001A 0000            add    [bx+si],al        cs 48AE
 cs:001C 0000            add    [bx+si],al        ip 0011
 cs:001E 0000            add    [bx+si],al
 cs:0020 0000            add    [bx+si],al

 es:0000 CD 20 FF 9F 00 EA FF FF = ƒ Ω            ss:0008 742E
 es:0008 AD DE E6 01 AF 15 B0 01 ¡▐µ☺»§░☺          ss:0006 706C
 es:0010 AF 15 83 02 13 10 92 01 »§â☻‼►Æ☺          ss:0004 6568
 es:0018 01 01 01 00 02 FF FF FF ☺☺☺ ☻            ss:0002 6474
 es:0020 FF FF FF FF FF FF FF FF                  ss:0000►0000
```

1. C) TO PERFORM MULTIPLICATION OPERATION ON 8-BIT DATA.

INPUT:

```
data segment
a db 09h
b db 02h
c dw ?
data ends

code segment
assume cs:code, ds:data
start:
mov ax,data
mov ds,ax
mov ax,0000h
mov bx,0000h
mov al,a
mov bl,b
mul b
mov c,ax
int 3
code ends
end start
```

OUTPUT:


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
≡ File Edit View Run Breakpoints Data Options Window Help READY
[■]=CPU 80486
1
[↕]
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

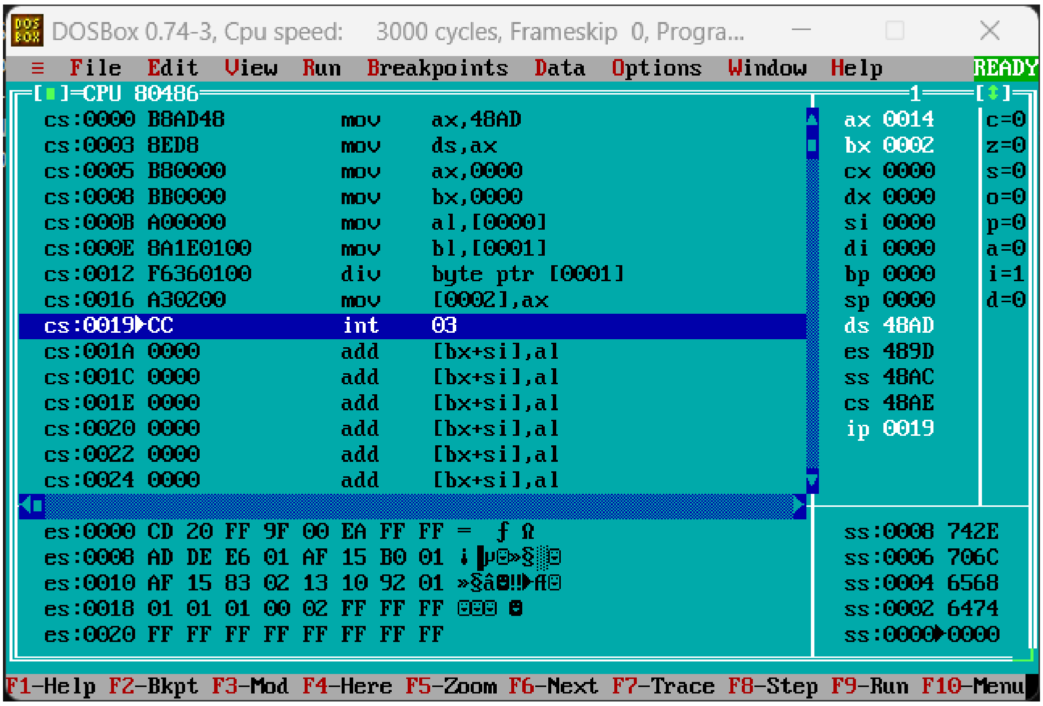
```
    mov     ax,48AD                 ax 0012   c=0
    mov     ds,ax                   bx 0002   z=0
    mov     ax,0000                 cx 0000   s=0
    mov     bx,0000                 dx 0000   o=0
    mov     al,[0000]               si 0000   p=0
    mov     bl,[0001]               di 0000   a=0
    mul     byte ptr [0001]         bp 0000   i=1
    mov     [0002],ax               sp 0000   d=0
    int     03                      ds 48AD
    add     [bx+si],al              es 489D
    add     [bx+si],al              ss 48AC
    add     [bx+si],al              cs 48AE
    add     [bx+si],al              ip 0019
    add     [bx+si],al
    add     [bx+si],al

 es:FFF0 00 00 00 00 00 00 00 00                ss:0008 742E
 es:FFF8 00 00 00 00 00 00 00 00                ss:0006 706C
 es:0000 CD 20 FF 9F 00 EA FF FF =  ƒ Ω          ss:0004 6568
 es:0008 AD DE E6 01 AF 15 B0 01 ¡▌µ☺»§░☺        ss:0002 6474
 es:0010 AF 15 83 02 13 10 92 01 »§â☻‼►Æ☺        ss:0000►0000
```

1. D) TO PERFORM DIVISION OPERATION ON 8-BIT DATA.

INPUT:

```
data segment
a db 28h
b db 02h
c dw ?
data ends

code segment
assume cs:code, ds:data
start:
mov ax,data
mov ds,ax
mov ax,0000h
mov bx,0000h
mov al,a
mov bl,b
div b
mov c,ax
int 3
code ends
end start
```

OUTPUT:


DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
≡ File Edit View Run Breakpoints Data Options Window Help
READY
[■]═CPU 80486
1
[↕]

```
cs:0000 B8AD48        mov   ax,48AD               ax 0014   c=0
cs:0003 8ED8          mov   ds,ax                 bx 0002   z=0
cs:0005 B80000        mov   ax,0000               cx 0000   s=0
cs:0008 BB0000        mov   bx,0000               dx 0000   o=0
cs:000B A00000        mov   al,[0000]             si 0000   p=0
cs:000E 8A1E0100      mov   bl,[0001]             di 0000   a=0
cs:0012 F6360100      div   byte ptr [0001]       bp 0000   i=1
cs:0016 A30200        mov   [0002],ax             sp 0000   d=0
cs:0019►CC            int   03                    ds 48AD
cs:001A 0000          add   [bx+si],al            es 489D
cs:001C 0000          add   [bx+si],al            ss 48AC
cs:001E 0000          add   [bx+si],al            cs 48AE
cs:0020 0000          add   [bx+si],al            ip 0019
cs:0022 0000          add   [bx+si],al
cs:0024 0000          add   [bx+si],al

es:0000 CD 20 FF 9F 00 EA FF FF =  ƒ Ω            ss:0008 742E
es:0008 AD DE E6 01 AF 15 B0 01 ¡▐µ☺»§░☺          ss:0006 706C
es:0010 AF 15 83 02 13 10 92 01 »§â☻‼►Æ☺          ss:0004 6568
es:0018 01 01 01 00 02 FF FF FF ☺☺☺ ☻             ss:0002 6474
es:0020 FF FF FF FF FF FF FF FF                   ss:0000►0000
```

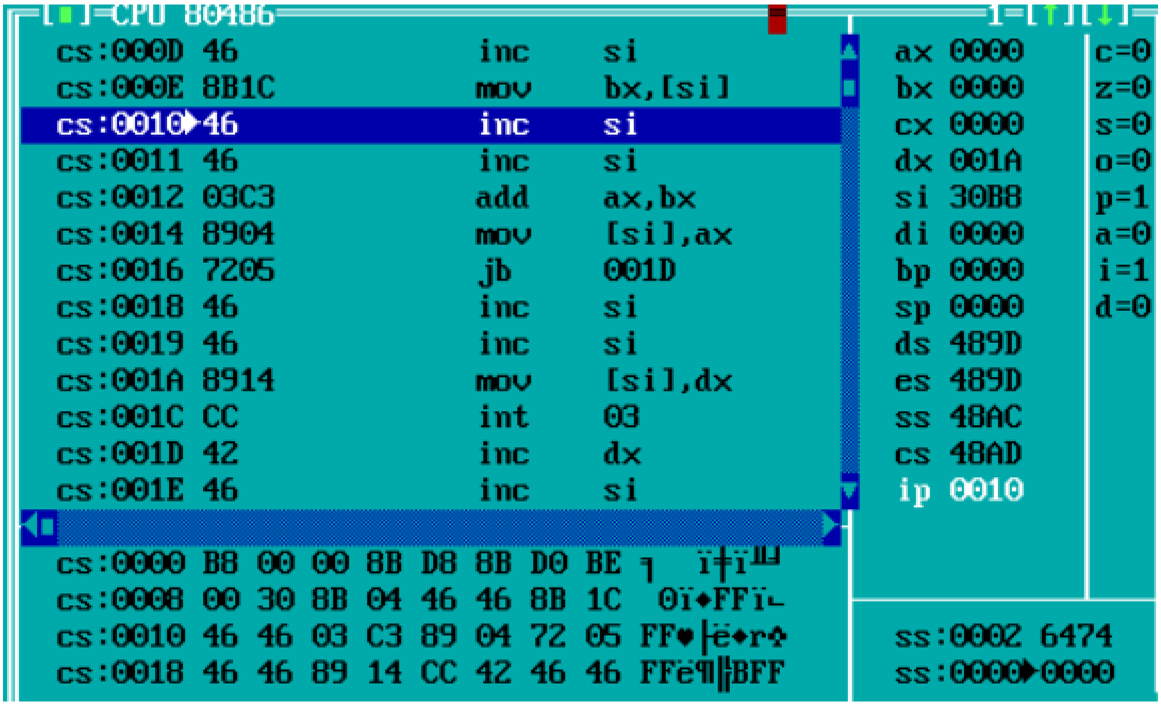
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu

2. A) TO PERFORM AN ADDITION OPERATION AN ASSEMBLY LANGUAGE ON 16-BIT NUMBER.

INPUT:

```
Code segment
Assume CS: code
Start :
Mov ax,0000h
Mov bx, ax
Mov dx, ax
Mov si,3000h
Mov ax ,[si]
Inc si
Inc si
Mov bx, [si]
Inc si
Inc si
Add ax,bx
Mov [si], ax
Jc l1
Inc si
Inc si
Mov [si], dx
Int 3
L1: inc dx
Inc si
Inc si
Mov [si], dx
Int 3
Code ends
End start
```

OUPUT:



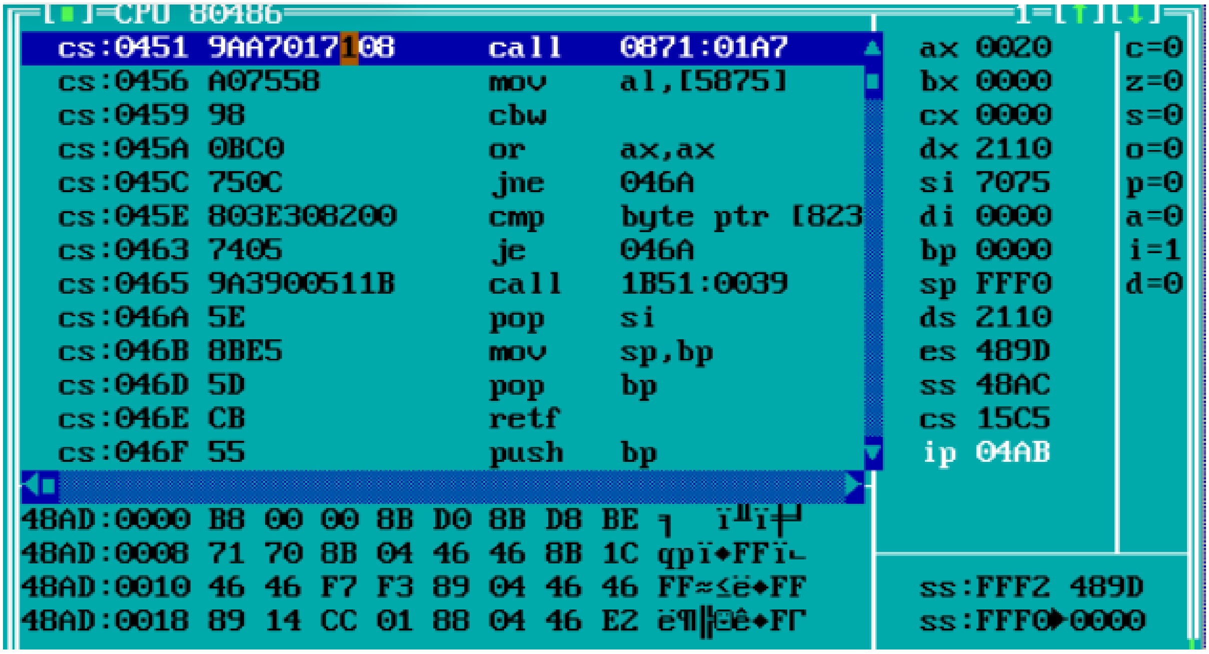
```
[■]=CPU 80486=========================================1=[↑][↓]=
 cs:000D 46              inc     si          ax 0000   c=0
 cs:000E 8B1C            mov     bx,[si]     bx 0000   z=0
 cs:0010►46              inc     si          cx 0000   s=0
 cs:0011 46              inc     si          dx 001A   o=0
 cs:0012 03C3            add     ax,bx       si 30B8   p=1
 cs:0014 8904            mov     [si],ax     di 0000   a=0
 cs:0016 7205            jb      001D        bp 0000   i=1
 cs:0018 46              inc     si          sp 0000   d=0
 cs:0019 46              inc     si          ds 489D
 cs:001A 8914            mov     [si],dx     es 489D
 cs:001C CC              int     03          ss 48AC
 cs:001D 42              inc     dx          cs 48AD
 cs:001E 46              inc     si          ip 0010

 cs:0000 B8 00 00 8B D8 8B D0 BE ┐  ï╪ï╨
 cs:0008 00 30 8B 04 46 46 8B 1C  0ï♦FFï∟
 cs:0010 46 46 03 C3 89 04 72 05 FF♥├ë♦r♣    ss:0002 6474
 cs:0018 46 46 89 14 CC 42 46 46 FFë¶╠BFF    ss:0000►0000
```

2. B) TO PERFORM A SUBTRACTION OPERATION ASSEMBLY LANGUAGE ON 16-BIT NUMBER.

INPUT:

```
Code segment
Assume CS: code
Start :
Mov ax, 0000h
Mov bx, ax
Mov dx, ax
Mov si, 3000h
Mov ax, [si]
Inc si
Inc si
Mov bx, [si]
Inc si
Inc si
Sub ax, bx
Mov [si], ax
Jc l1
Inc si
Inc si
Mov [si], dx
Int 3
L1: inc dx
Inc si
Inc si
Mov [si], dx
Int 3
Code ends
End start
```

OUTPUT:



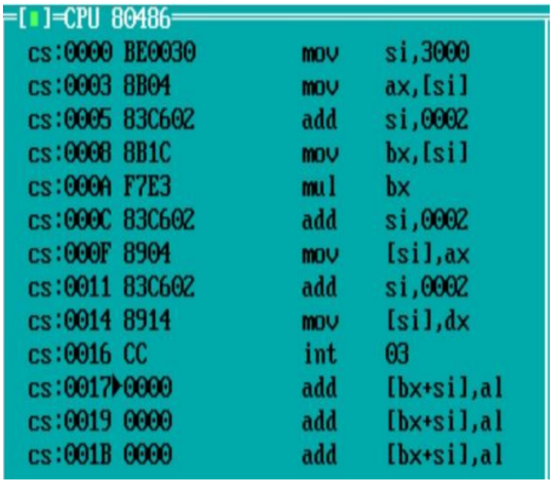
```
[■]=CPU 80486=========================================================1=[↑][↓]
 cs:0451 9AA7017108     call   0871:01A7        ax 0020    c=0
 cs:0456 A07558         mov    al,[5875]        bx 0000    z=0
 cs:0459 98             cbw                     cx 0000    s=0
 cs:045A 0BC0           or     ax,ax            dx 2110    o=0
 cs:045C 750C           jne    046A             si 7075    p=0
 cs:045E 803E308200     cmp    byte ptr [823    di 0000    a=0
 cs:0463 7405           je     046A             bp 0000    i=1
 cs:0465 9A3900511B     call   1B51:0039        sp FFF0    d=0
 cs:046A 5E             pop    si               ds 2110
 cs:046B 8BE5           mov    sp,bp            es 489D
 cs:046D 5D             pop    bp               ss 48AC
 cs:046E CB             retf                    cs 15C5
 cs:046F 55             push   bp               ip 04AB
48AD:0000 B8 00 00 8B D0 8B D8 BE ╕  ï╨ï╪╛
48AD:0008 71 70 8B 04 46 46 8B 1C qpï♦FFï∟
48AD:0010 46 46 F7 F3 89 04 46 46 FF≈≤ë♦FF      ss:FFF2 489D
48AD:0018 89 14 CC 01 88 04 46 E2 ë¶╠☺ê♦FΓ      ss:FFF0►0000
```

2. C) TO PERFORM A MULTIPLICATION OPERATION ASSEMBLY LANGUAGE ON 16-BIT NUMBER.

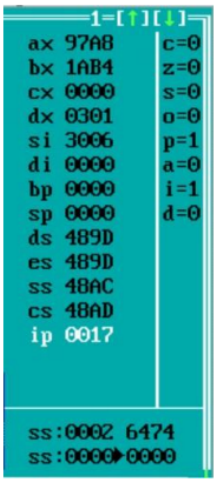
INPUT:

```
Code segment
Assume CS:code
Start :
Mov ah, 0000h
Mov si, 3000h
Mov ax, [si]
Add si , 02h
Mov bx , [si]
Mul bx
Add si , 02h
Mov [si] , ax
Add si , 02h
Add [si] , dx
Int 3
Code ends
End start
```

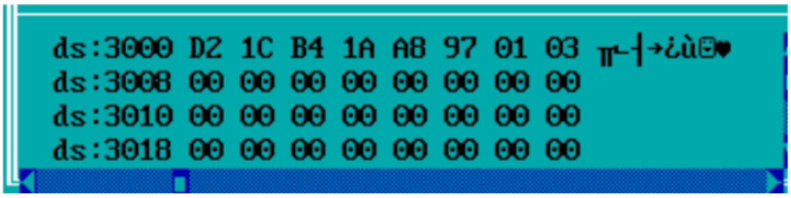
OUTPUT:



```
[■]=CPU 80486
cs:0000 BE0030      mov    si,3000
cs:0003 8B04        mov    ax,[si]
cs:0005 83C602      add    si,0002
cs:0008 8B1C        mov    bx,[si]
cs:000A F7E3        mul    bx
cs:000C 83C602      add    si,0002
cs:000F 8904        mov    [si],ax
cs:0011 83C602      add    si,0002
cs:0014 8914        mov    [si],dx
cs:0016 CC          int    03
cs:0017▸0000        add    [bx+si],al
cs:0019 0000        add    [bx+si],al
cs:001B 0000        add    [bx+si],al
```



```
1=[↑][↓]
ax 97A8    c=0
bx 1AB4    z=0
cx 0000    s=0
dx 0301    o=0
si 3006    p=1
di 0000    a=0
bp 0000    i=1
sp 0000    d=0
ds 489D
es 489D
ss 48AC
cs 48AD
ip 0017

ss:0002 6474
ss:0000▸0000
```



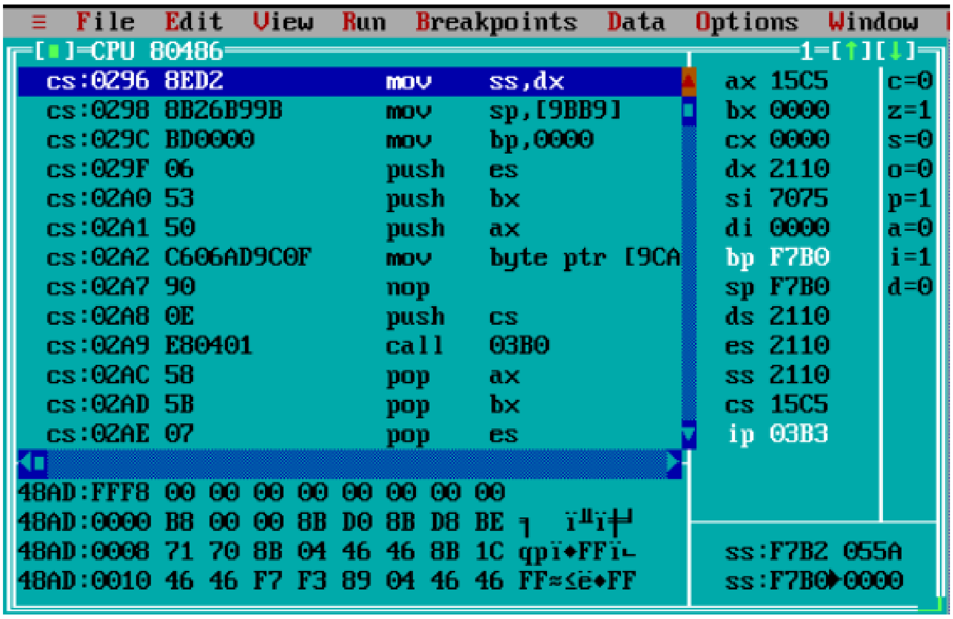
```
ds:3000 D2 1C B4 1A A8 97 01 03  ╥└┤→¿ù☺♥
ds:3008 00 00 00 00 00 00 00 00
ds:3010 00 00 00 00 00 00 00 00
ds:3018 00 00 00 00 00 00 00 00
```

2. D) TO PERFORM A DIVISION OPERATION ASSEMBLY LANGUAGE ON 16-BIT NUMBER.

INPUT:

```
Code segment
assume CS: code
start: mov ax,0000h
mov dx, ax
mov bx, ax
mov si, 7071h
mov ax, [si]
inc si
inc si
mov bx, [si]
inc si
inc si
div bx
mov [si], ax
inc si
inc si
mov [si], dx
int 3
code ends
end start
```

OUTPUT:



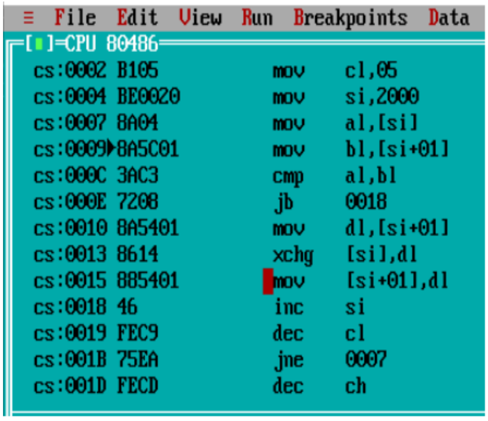
```
cs:0296 8ED2           mov    ss,dx                 ax 15C5    c=0
cs:0298 8B26B99B       mov    sp,[9BB9]             bx 0000    z=1
cs:029C BD0000         mov    bp,0000               cx 0000    s=0
cs:029F 06             push   es                    dx 2110    o=0
cs:02A0 53             push   bx                    si 7075    p=1
cs:02A1 50             push   ax                    di 0000    a=0
cs:02A2 C606AD9C0F     mov    byte ptr [9CA         bp F7B0    i=1
cs:02A7 90             nop                          sp F7B0    d=0
cs:02A8 0E             push   cs                    ds 2110
cs:02A9 E80401         call   03B0                  es 2110
cs:02AC 58             pop    ax                    ss 2110
cs:02AD 5B             pop    bx                    cs 15C5
cs:02AE 07             pop    es                    ip 03B3

48AD:FFF8 00 00 00 00 00 00 00 00
48AD:0000 B8 00 00 8B D0 8B D8 BE ╕  ï╨ï╪╛
48AD:0008 71 70 8B 04 46 46 8B 1C qpï♦FFï∟     ss:F7B2 055A
48AD:0010 46 46 F7 F3 89 04 46 46 FF≈≤ë♦FF     ss:F7B0►0000
```

3. WRITE A PROGRAM TO ARRANGE GIVEN NUMBERS IN ASCENDING ORDER.

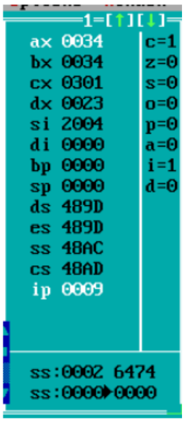
INPUT:

```
Code segment
Assume CS:code
Start: mov ch, 05h
L1: mov cl, 05h
Mov si , 2000h
L2: mov al , [si]
Mov bl , [si +1]
Cmp al , bl
Jc l3
Mov dl , [si + 1]
Xchg [si], dl
L3: inc si
Dec cl
Jnz l2
Dec ch
Jnz l
Int 3
Code ends
End start
```

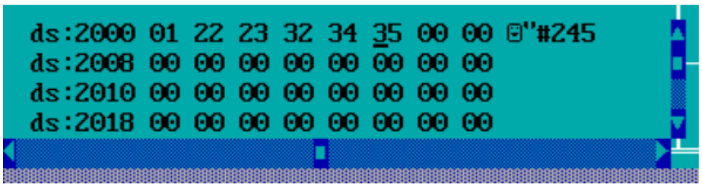
OUTPUT:



```
cs:0002 B105        mov    cl,05
cs:0004 BE0020      mov    si,2000
cs:0007 8A04        mov    al,[si]
cs:0009▸8A5C01      mov    bl,[si+01]
cs:000C 3AC3        cmp    al,bl
cs:000E 7208        jb     0018
cs:0010 8A5401      mov    dl,[si+01]
cs:0013 8614        xchg   [si],dl
cs:0015 885401      mov    [si+01],dl
cs:0018 46          inc    si
cs:0019 FEC9        dec    cl
cs:001B 75EA        jne    0007
cs:001D FECD        dec    ch
```



```
ax 0034   c=1
bx 0034   z=0
cx 0301   s=0
dx 0023   o=0
si 2004   p=0
di 0000   a=0
bp 0000   i=1
sp 0000   d=0
ds 489D
es 489D
ss 48AC
cs 48AD
ip 0009

ss:0002 6474
ss:0000▸0000
```



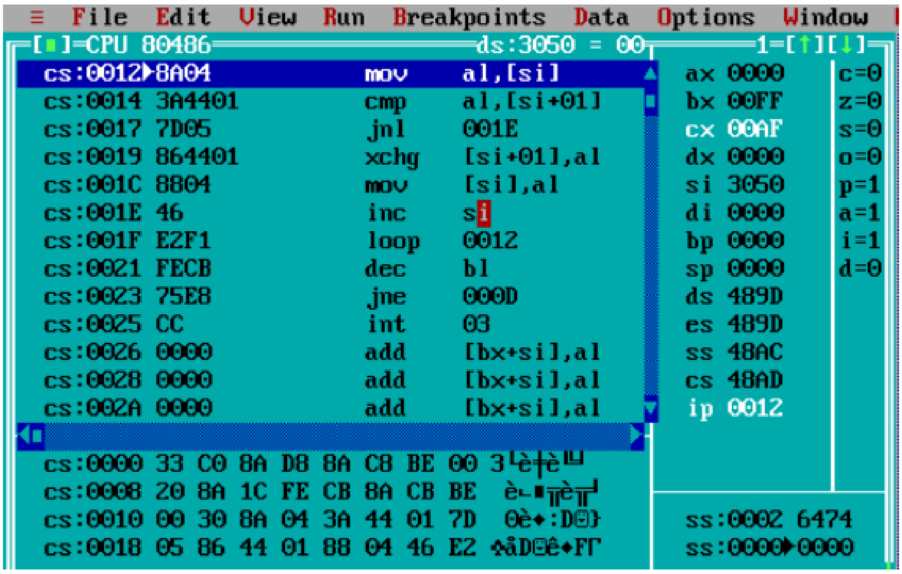
```
ds:2000 01 22 23 32 34 35 00 00 ☺"#245
ds:2008 00 00 00 00 00 00 00 00
ds:2010 00 00 00 00 00 00 00 00
ds:2018 00 00 00 00 00 00 00 00
```

4. WRITE A PROGRAM TO ARRANGE GIVEN NUMBERS IN DESCENDING ORDER.

INPUT:

```
Code segment
Assume CS: code
Start: xor ax , ax
Mov bl, al
Mov cl, al
Mov si, 2000h
Mov bl, [si]
Dec bl
L3: mov cl, bl
Mov si, 3000h
L2: mov al, [si]
Cmp al, [si +1]
Jge l1
Xchg al, [si +1]
Mov [si], al
L1: inc si
Loop L2
Dec bl
Jnz L3
Int 3
Code ends
End start
```

OUTPUT:



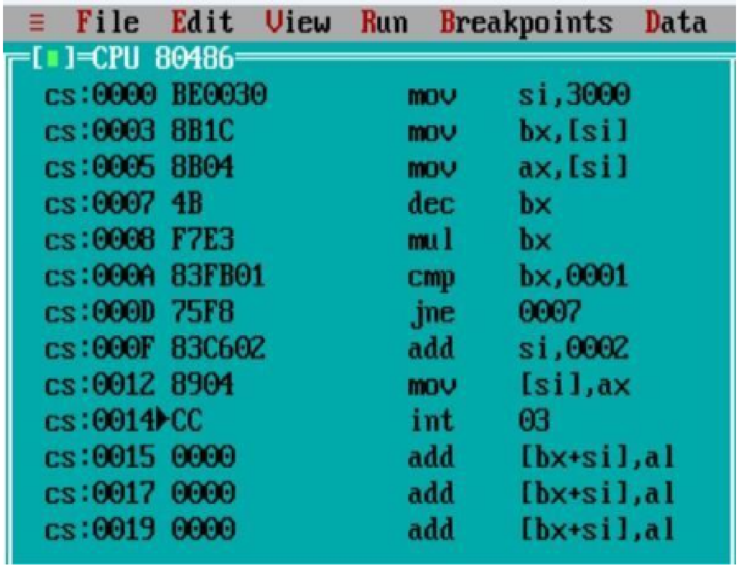
```
ds:3050 = 00
cs:0012▸8A04        mov    al,[si]         ax 0000   c=0
cs:0014 3A4401      cmp    al,[si+01]      bx 00FF   z=0
cs:0017 7D05        jnl    001E            cx 00AF   s=0
cs:0019 864401      xchg   [si+01],al      dx 0000   o=0
cs:001C 8804        mov    [si],al         si 3050   p=1
cs:001E 46          inc    si              di 0000   a=1
cs:001F E2F1        loop   0012            bp 0000   i=1
cs:0021 FECB        dec    bl              sp 0000   d=0
cs:0023 75E8        jne    000D            ds 489D
cs:0025 CC          int    03              es 489D
cs:0026 0000        add    [bx+si],al      ss 48AC
cs:0028 0000        add    [bx+si],al      cs 48AD
cs:002A 0000        add    [bx+si],al      ip 0012

cs:0000 33 C0 8A D8 8A C8 BE 00 3└è╪è╚
cs:0008 20 8A 1C FE CB 8A CB BE  è∟■╦è╦╛
cs:0010 00 30 8A 04 3A 44 01 7D  0è♦:D☺}   ss:0002 6474
cs:0018 05 86 44 01 88 04 46 E2 ♣åD☺ê♦FΓ   ss:0000▸0000
```

5. WRITE AN ASSEMBLY LANGUAGE PROGRAM TO FIND THE FACTORIAL OF GIVEN NUMBER.

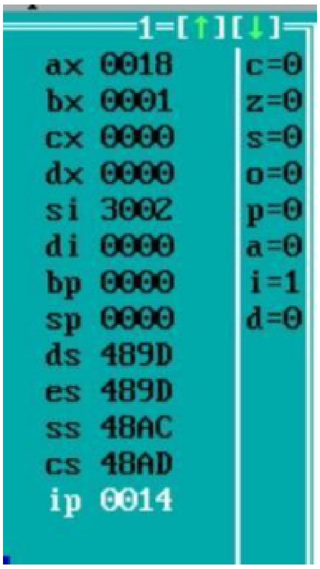
INPUT:

```
Code segment
Assume CS: code
Start:
Mov si, 3000h
Mov bx, [si]
Mov ax, [si]
L1: dec bx
Mul bx
Cmp bx, 01h
Jnz l1
Add si, ax
Int 3
Code ends
End start
```

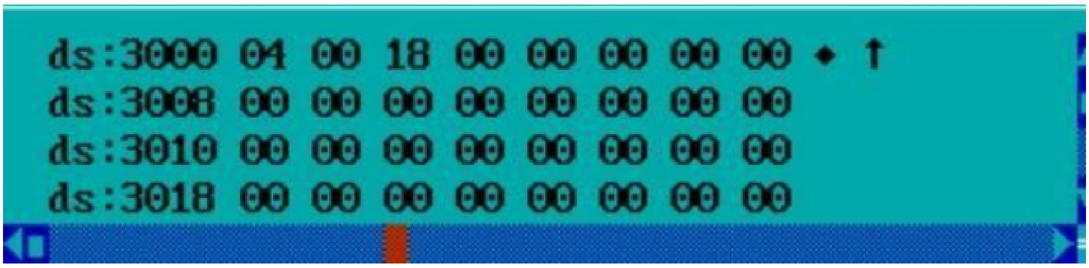
OUTPUT:



```
cs:0000 BE0030      mov     si,3000
cs:0003 8B1C        mov     bx,[si]
cs:0005 8B04        mov     ax,[si]
cs:0007 4B          dec     bx
cs:0008 F7E3        mul     bx
cs:000A 83FB01      cmp     bx,0001
cs:000D 75F8        jne     0007
cs:000F 83C602      add     si,0002
cs:0012 8904        mov     [si],ax
cs:0014►CC          int     03
cs:0015 0000        add     [bx+si],al
cs:0017 0000        add     [bx+si],al
cs:0019 0000        add     [bx+si],al
```



```
ax 0018   c=0
bx 0001   z=0
cx 0000   s=0
dx 0000   o=0
si 3002   p=0
di 0000   a=0
bp 0000   i=1
sp 0000   d=0
ds 489D
es 489D
ss 48AC
cs 48AD
ip 0014
```



```
ds:3000 04 00 18 00 00 00 00 00 ♦ ↑
ds:3008 00 00 00 00 00 00 00 00
ds:3010 00 00 00 00 00 00 00 00
ds:3018 00 00 00 00 00 00 00 00
```