AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH (AIUB)
Faculty of Engineering
Department of EEE and CoE
Undergraduate Program

American International University-Bangladesh (AIUB)

Course: MICROPROCESSOR AND EMBEDDED SYSTEMS **Summer 2021-22, MID**

Experiment 5: Familiarization of assembly language programs

Submitted by:

Name	ID
MD SHAMIM SIDDIKY	20-42649-1
WASIUDDIN	15-30843-3
TANJIM HOSSAIN	18-36964-1
MD. SAKIB HOSSAIN RIJON	19-39460-1
SADIA AFRIN ETY	19-39659-1
FERDOUS SUNY	19-40485-1
DIPONKAR SUTRA DHAR	19-41004-2

Section: O

Group: 6

Submitted To

Md Ali Noor
Lecturer
Faculty of Computer Engineering
American International University-Bangladesh

Objective:

To learn how to write assembly programs using 8086 instructions and arrays.

Theory and methodology:

To understand the working principle of assembly language programs 8086 Microprocessor and familiarize emulator EMU8086 is done in this experiment. The main objective of this experiment is to test its different uses, introduction to segmented memory technology used by Microprocessor 8086 and work with the advanced 8086 instructions and learn how to write assembly programs using the 8086 instructions. The microprocessor 8086 can be considered to be the basic processor for the Intel X-86 family. With the knowledge of this 16-bit processor, one can study the further versions of this processor 80386, 80406 and Pentium. Through assembly language we can able to handle direct processor registers because assembly language is low level language which is convenient for hardware system controlling. In computers, there is an assembler that helps in converting the assembly code into machine code executable.

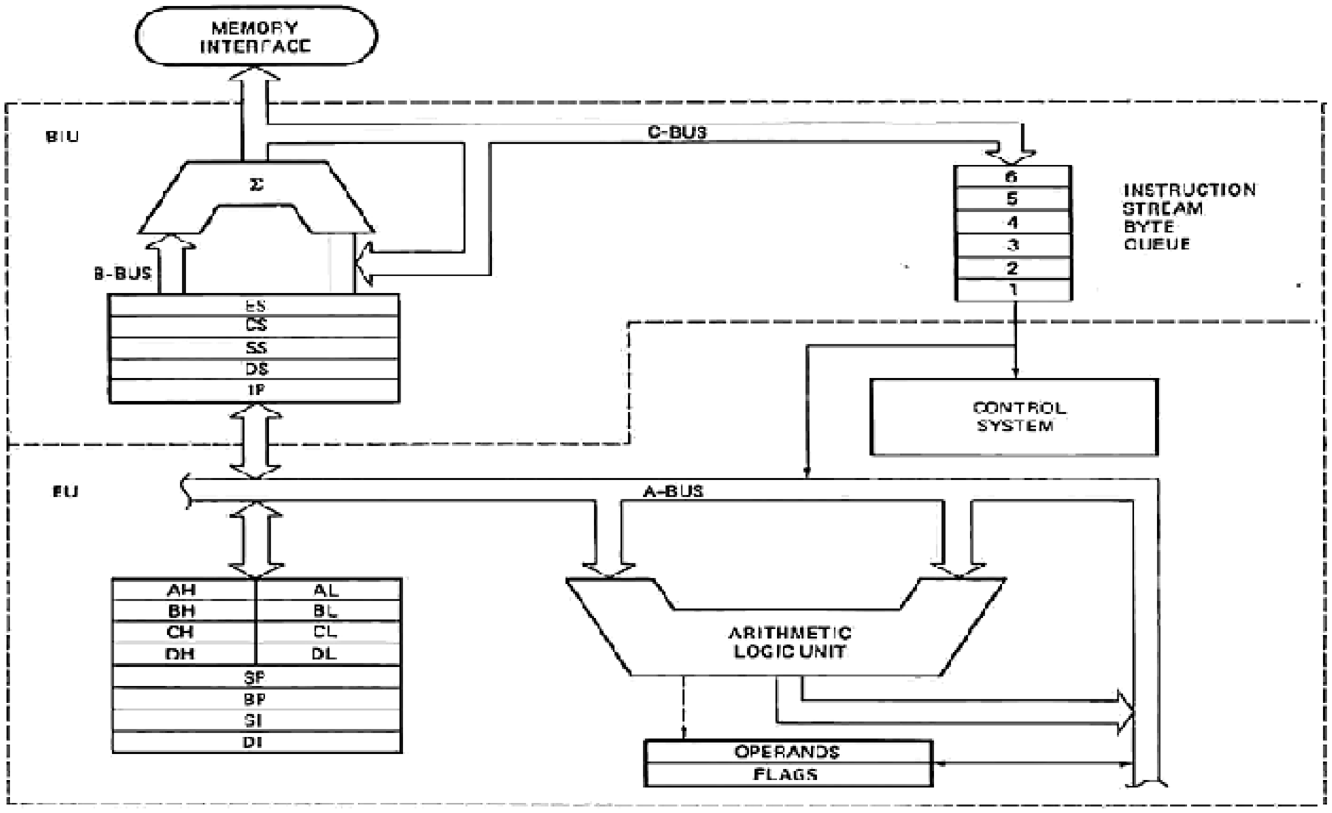


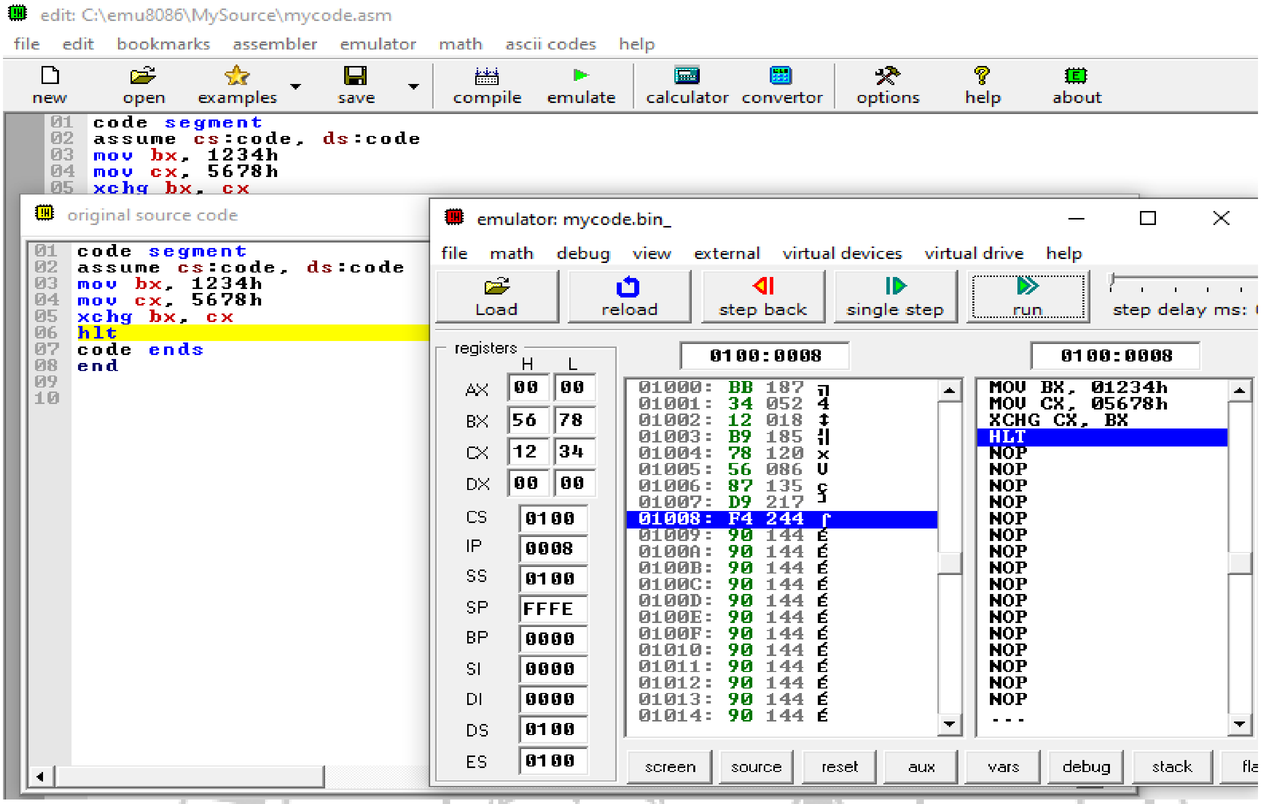
Fig 1: Intel 8086 internal architecture

Equipment List:

- EMU8086 [ver.408 (32-bit WINOS compatible)]
- PC having Intel Microprocessor

Code and Output:

1. Program to exchange the contents of two register



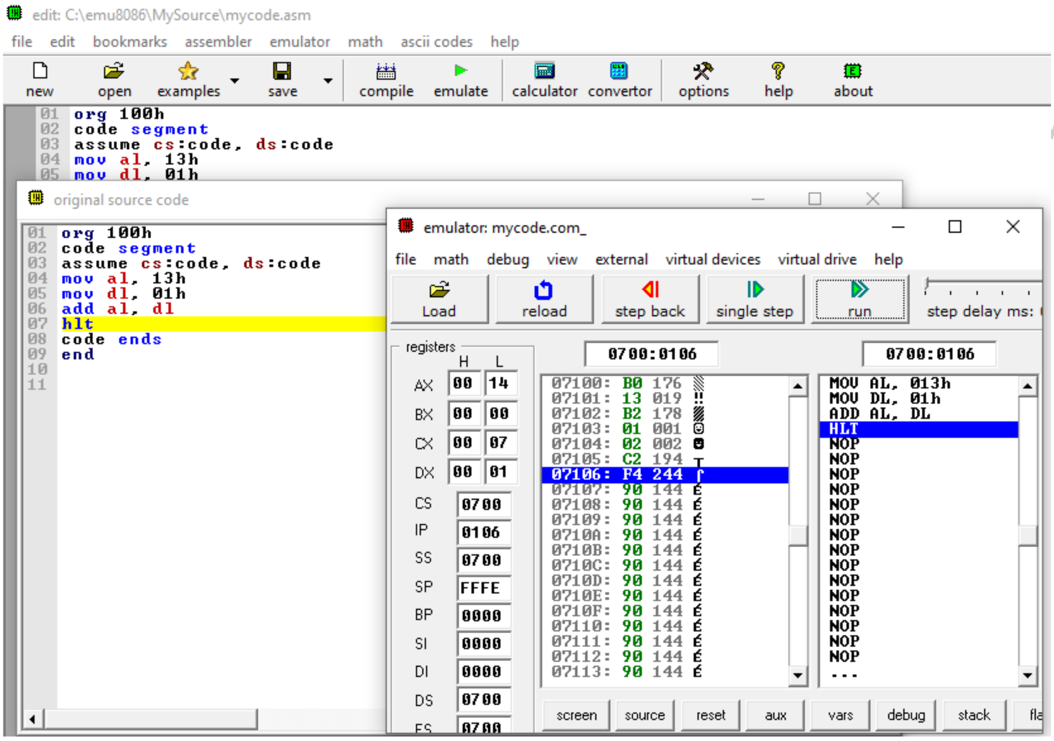
```
code segment
assume cs:code, ds:code
mov bx, 1234h
mov cx, 5678h
xchg bx, cx
hlt
code ends
end
```

```
registers   H   L
AX          00  00
BX          56  78
CX          12  34
DX          00  00
CS          0100
IP          0008
SS          0100
SP          FFFE
BP          0000
SI          0000
DI          0000
DS          0100
ES          0100
```

```
0100:0008                      0100:0008
01000: BB 187 ╗                MOV BX, 01234h
01001: 34 052 4                MOV CX, 05678h
01002: 12 018 ↕                XCHG CX, BX
01003: B9 185 ╣                HLT
01004: 78 120 x                NOP
01005: 56 086 V                NOP
01006: 87 135 ç                NOP
01007: D9 217 ┘                NOP
01008: F4 244 ⌠                NOP
01009: 90 144 É                NOP
0100A: 90 144 É                NOP
0100B: 90 144 É                NOP
0100C: 90 144 É                NOP
0100D: 90 144 É                NOP
0100E: 90 144 É                NOP
0100F: 90 144 É                NOP
01010: 90 144 É                NOP
01011: 90 144 É                NOP
01012: 90 144 É                NOP
01013: 90 144 É                NOP
01014: 90 144 É                ...
```

Explanation: XCHG instruction exchanges the content of a register with the content of another register or with the content of memory location. It cannot directly exchange the content of two memory locations. The source and destination must both be of the same type (bytes or words).

2. Program for adding two numbers.



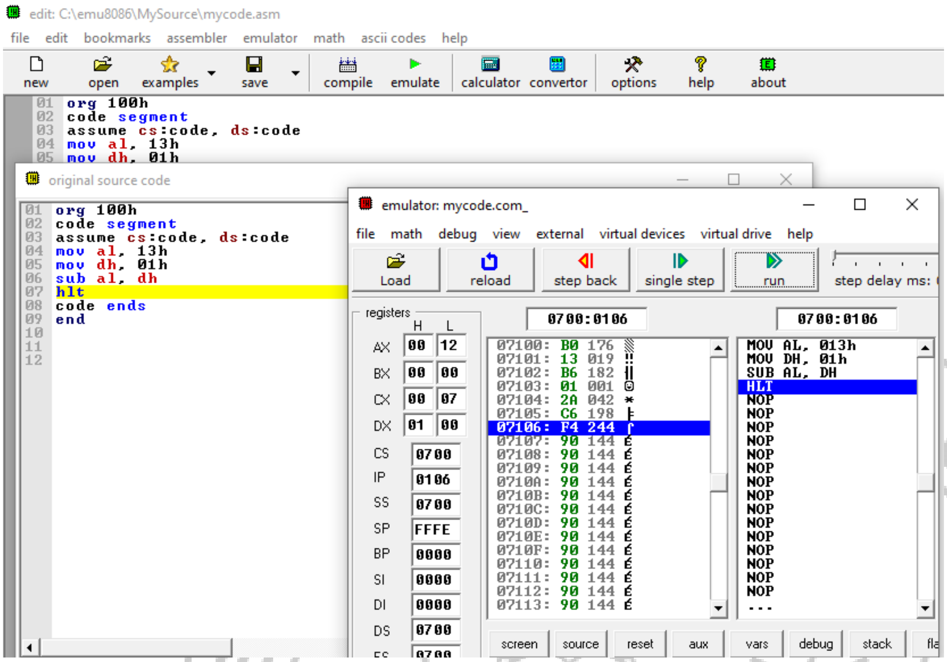
```
org 100h
code segment
assume cs:code, ds:code
mov al, 13h
mov dl, 01h
add al, dl
hlt
code ends
end
```

```
registers   H   L
AX          00  14
BX          00  00
CX          00  07
DX          00  01
CS          0700
IP          0106
SS          0700
SP          FFFE
BP          0000
SI          0000
DI          0000
DS          0700
ES          0700
```

```
0700:0106                      0700:0106
07100: B0 176 ░                MOV AL, 013h
07101: 13 019 ‼                MOV DL, 01h
07102: B2 178 ▓                ADD AL, DL
07103: 01 001 ☺                HLT
07104: 02 002 ☻                NOP
07105: C2 194 ┬                NOP
07106: F4 244 ⌠                NOP
07107: 90 144 É                NOP
07108: 90 144 É                NOP
07109: 90 144 É                NOP
0710A: 90 144 É                NOP
0710B: 90 144 É                NOP
0710C: 90 144 É                NOP
0710D: 90 144 É                NOP
0710E: 90 144 É                NOP
0710F: 90 144 É                NOP
07110: 90 144 É                NOP
07111: 90 144 É                NOP
07112: 90 144 É                NOP
07113: 90 144 É                ...
```

Explanation: The ADD instruction adds the data of destination and source operand and stores the result in destination. Both operands should be of same type i.e. words or bytes otherwise assembler will generate an error.

3. Program for subtraction between two numbers


edit: C:\emu8086\MySource\mycode.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
original source code
emulator: mycode.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms:
screen source reset aux vars debug stack fla

```
org 100h
code segment
assume cs:code, ds:code
mov al, 13h
mov dh, 01h
```

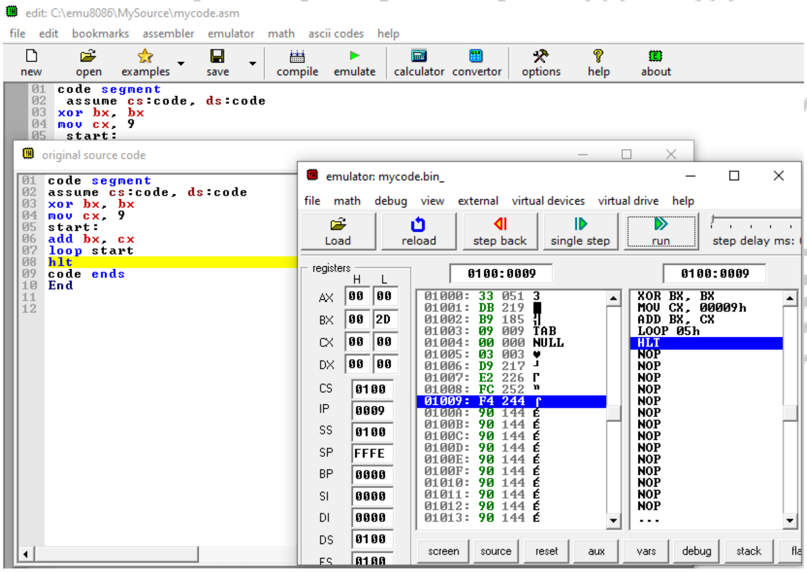
```
org 100h
code segment
assume cs:code, ds:code
mov al, 13h
mov dh, 01h
sub al, dh
hlt
code ends
end
```

```
registers
     H   L
AX   00  12
BX   00  00
CX   00  07
DX   01  00
CS   0700
IP   0106
SS   0700
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

```
0700:0106                     0700:0106
07100: B0 176 ░               MOV AL, 013h
07101: 13 019 ‼               MOV DH, 01h
07102: B6 182 ╢               SUB AL, DH
07103: 01 001 ☺               HLT
07104: 2A 042 *               NOP
07105: C6 198 ╞               NOP
07106: F4 244 ⌠               NOP
07107: 90 144 É               NOP
07108: 90 144 É               NOP
07109: 90 144 É               NOP
0710A: 90 144 É               NOP
0710B: 90 144 É               NOP
0710C: 90 144 É               NOP
0710D: 90 144 É               NOP
0710E: 90 144 É               NOP
0710F: 90 144 É               NOP
07110: 90 144 É               NOP
07111: 90 144 É               NOP
07112: 90 144 É               NOP
07113: 90 144 É               ...
```

Explanation: Subtraction (sub) instruction takes two operands. Subtract the data in the source operand from the data of destination operand and then store the result back to the destination operand.

4. Program for Summation of a series: [1+2+3+4+………+N] = BX . The value of N is stored in C


edit: C:\emu8086\MySource\mycode.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
original source code
emulator: mycode.bin_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms:
screen source reset aux vars debug stack fla

```
code segment
 assume cs:code, ds:code
xor bx, bx
mov cx, 9
 start:
```

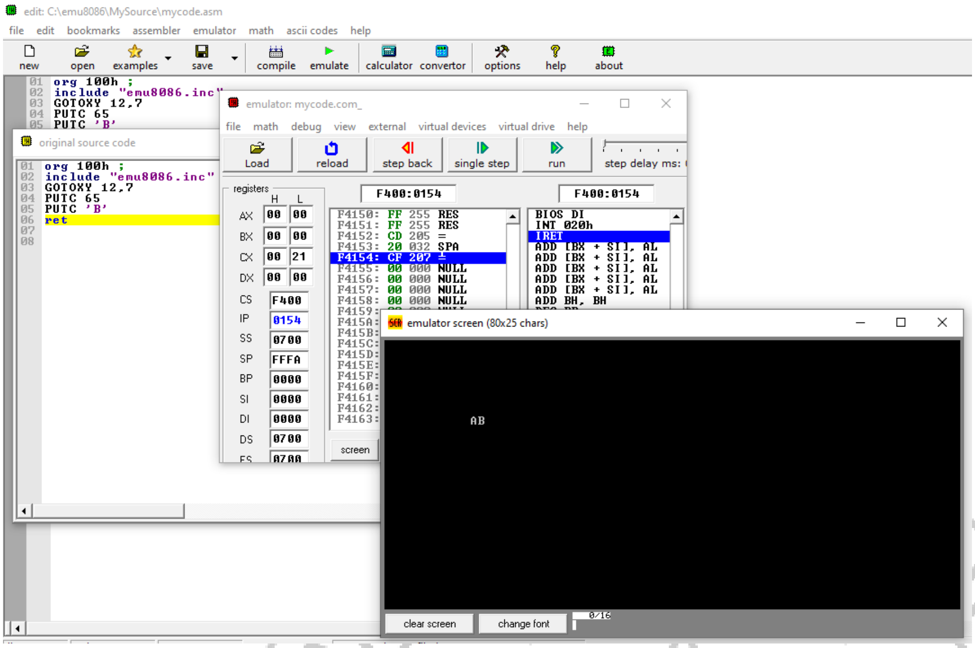
```
code segment
assume cs:code, ds:code
xor bx, bx
mov cx, 9
start:
add bx, cx
loop start
hlt
code ends
End
```

```
registers
     H   L
AX   00  00
BX   00  2D
CX   00  00
DX   00  00
CS   0100
IP   0009
SS   0100
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0100
ES   0100
```

```
0100:0009                     0100:0009
01000: 33 051 3               XOR BX, BX
01001: DB 219 █               MOV CX, 00009h
01002: B9 185 ╣               ADD BX, CX
01003: 09 009 TAB             LOOP 05h
01004: 00 000 NULL            HLT
01005: 03 003 ♥               NOP
01006: D9 217 ┘               NOP
01007: E2 226 Γ               NOP
01008: FC 252 ⁿ               NOP
01009: F4 244 ⌠               NOP
0100A: 90 144 É               NOP
0100B: 90 144 É               NOP
0100C: 90 144 É               NOP
0100D: 90 144 É               NOP
0100E: 90 144 É               NOP
0100F: 90 144 É               NOP
01010: 90 144 É               NOP
01011: 90 144 É               NOP
01012: 90 144 É               NOP
01013: 90 144 É               ...
```

Explanation: Loop is used to do the same tasks multiple times. Here add operation performed multiple times using loop.

5. Program which display two characters at column#12 and row#7 at emulator screen.


edit: C:\emu8086\MySource\mycode.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
original source code
emulator: mycode.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms:
registers
screen
emulator screen (80x25 chars)
clear screen change font
0/16

```
org 100h ;
include "emu8086.inc"
GOTOXY 12,7
PUTC 65
PUTC 'B'
ret
```

```
     H  L
AX   00 00
BX   00 00
CX   00 21
DX   00 00
CS   F400
IP   0154
SS   0700
SP   FFFA
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

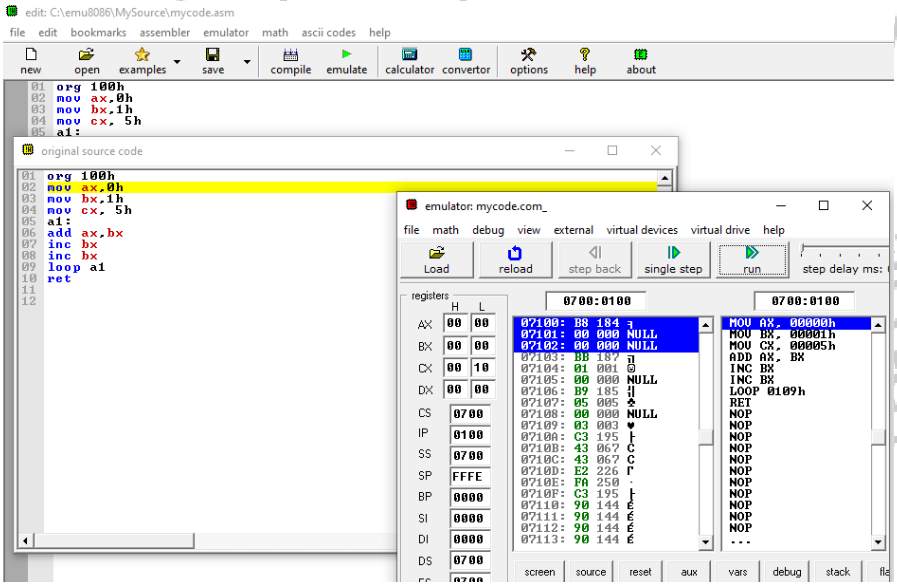
```
F400:0154

F4150:  FF 255 RES
F4151:  FF 255 RES
F4152:  CD 205 =
F4153:  20 032 SPA
F4154:  CF 207 ±
F4155:  00 000 NULL
F4156:  00 000 NULL
F4157:  00 000 NULL
F4158:  00 000 NULL
F4159:
F415A:
F415B:
F415C:
F415D:
F415E:
F415F:
F4160:
F4161:
F4162:
F4163:
```

```
F400:0154

BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
```

```
AB
```

Explanation: Here, PUTC char is macro with 1 parameter, and prints out an ASCII char at the current cursor position. And GOTOXY is macro with 2 parameters and sets cursor position.

6. Assembly code for the following sequence 1+3+5+7…..+N. Where N = 5 using a loop


edit: C:\emu8086\MySource\mycode.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
original source code
emulator: mycode.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms:
registers
screen source reset aux vars debug stack fla

```
org 100h
mov ax,0h
mov bx,1h
mov cx, 5h
a1:
add ax,bx
inc bx
inc bx
loop a1
ret
```

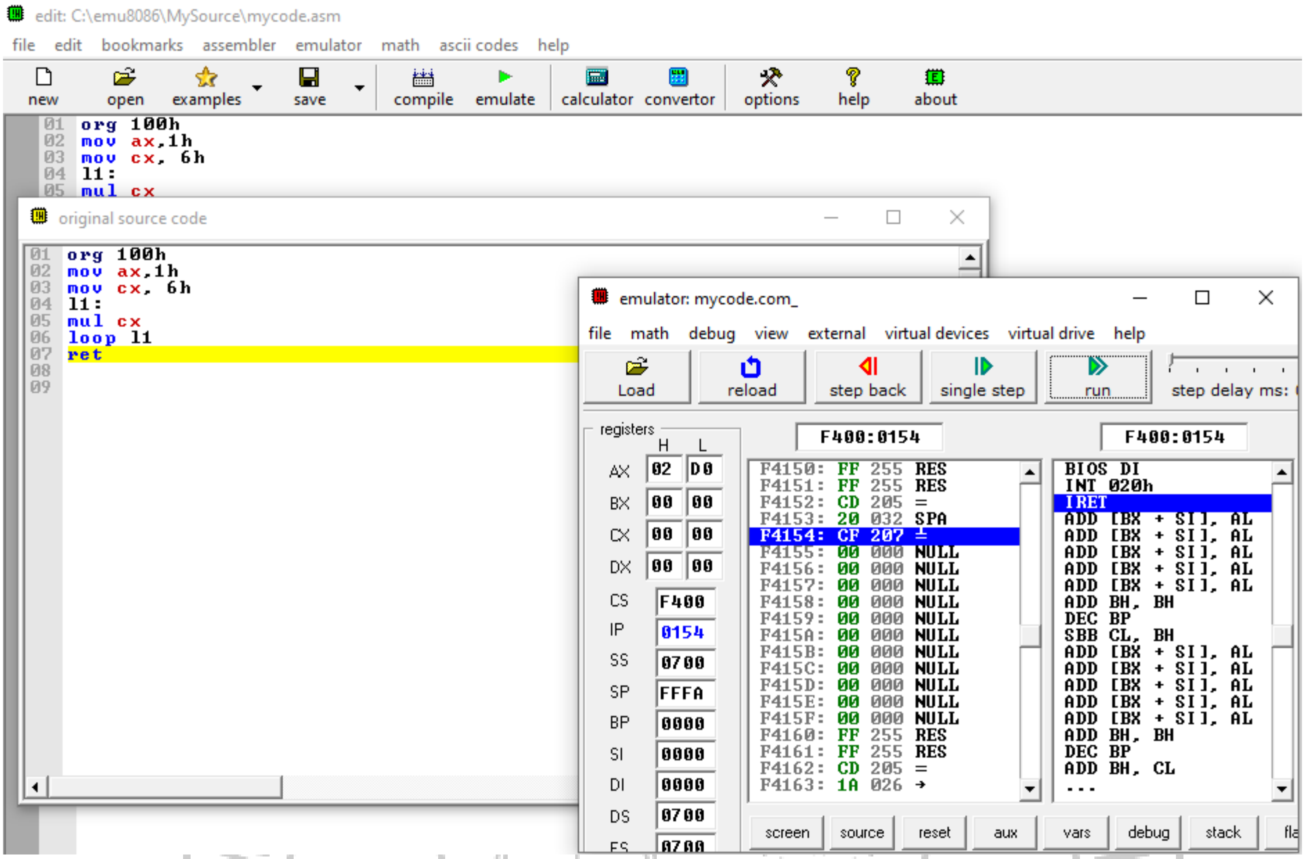
```
     H  L
AX   00 00
BX   00 00
CX   00 10
DX   00 00
CS   0700
IP   0100
SS   0700
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

```
0700:0100

07100:  B8 184 ╕
07101:  00 000 NULL
07102:  00 000 NULL
07103:  BB 187 ╗
07104:  01 001 ☺
07105:  00 000 NULL
07106:  B9 185 ╣
07107:  05 005 ♣
07108:  00 000 NULL
07109:  03 003 ♥
0710A:  C3 195 ├
0710B:  43 067 C
0710C:  43 067 C
0710D:  E2 226 Γ
0710E:  FA 250 ·
0710F:  C3 195 ├
07110:  90 144 É
07111:  90 144 É
07112:  90 144 É
07113:  90 144 É
```

```
0700:0100

MOV AX, 00000h
MOV BX, 00001h
MOV CX, 00005h
ADD AX, BX
INC BX
INC BX
LOOP 0109h
RET
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

Explanation: Loop is used to do the same tasks multiple times. Here add operation performed multiple times using loop.

7. Write a code for finding the value of 6!


edit: C:\emu8086\MySource\mycode.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
original source code
emulator: mycode.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms:
screen source reset aux vars debug stack

```
org 100h
mov ax,1h
mov cx, 6h
l1:
mul cx
loop l1
ret
```

registers

	H	L
AX	02	D0
BX	00	00
CX	00	00
DX	00	00

CS	F400
IP	0154
SS	0700
SP	FFFA
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

```
F400:0154
F4150: FF 255 RES
F4151: FF 255 RES
F4152: CD 205 =
F4153: 20 032 SPA
F4154: CF 207 ╧
F4155: 00 000 NULL
F4156: 00 000 NULL
F4157: 00 000 NULL
F4158: 00 000 NULL
F4159: 00 000 NULL
F415A: 00 000 NULL
F415B: 00 000 NULL
F415C: 00 000 NULL
F415D: 00 000 NULL
F415E: 00 000 NULL
F415F: 00 000 NULL
F4160: FF 255 RES
F4161: FF 255 RES
F4162: CD 205 =
F4163: 1A 026 →
```

```
F400:0154
BIOS DI
INT 020h
IRET
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
SBB CL, BH
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD BH, BH
DEC BP
ADD BH, CL
...
```

Explanation: Loop is used to do the same tasks multiple times. Here by using the loop we have performed the mul instruction to find the factorial.

Discussion:

The codes were implemented and run them on emu8086. The output was same as instructors' output. First, we did exchange the contents of two registers. So, we a write an assembly code and exchange the content from BX register to CX register. Then, in addition of two number is done by assembly language using both AX register and BX register. After performing the operation result has been stored in AX. In subtraction program we stored two values in AX register and in BX register. After performing the operation result has been stored in AX. Similarly following codes and other equation solving is done by assembly language using hardware's register. In this experiment there may be an only scope of error is typing mistake while typing the codes in the emu8086.