We use compiler emulator 8086 which is not case sensitive, that means if we write a keyword with mixing lowercase and uppercase letter it can identify easily without any trouble.

In assembly language every syntax has four fields

First field is label name. It follow the rules of valid identifier rules and any label name will be highest 35 characters. End of the label name there will be semicolon symbol (:) , label name generally used for control a block of statement execution specially in loop control.

Second field is called mnemonic or operation code (abbreviation Opcode) , it indicates microprocessor which operation should be performed by CPU. Example of Opcode are ADD, SUB, DIV, MUL, MOV , INT (interrupt), INC ,DEC etc.

Third field is operands , it may be single or double. Two operands separated by comma symbol.
As we use compiler emulator 8086, 8086 is a 16 bit microprocessor. It follows the syntax format
Opcode <space> destinationOopreand , SourceOperand

destination operand generally copy the value of source operand in case of MOV operation.
In arithmetic operation destination operand value generally override.

Fourth field is Comment field and it begins with semicolon(;) symbol

Every field must be separated with at least one space.
We most of the time use two fields in our code. One is Opcode and other is operands.

Every assembly language program separated in three segments.

First is Stack segment (which we write as .stack) after declare stack segment it requires to specify memory space, for this reason we generally write .stack 100h

Second is data segment (which we write as .data). Variable declaration , may be memory variable, string variable etc we write here.

Third is code segment (which we write as .code) .

These three segments are especially by default 64KB in size and are allocated RAM memory space.

Every assembly program there must be a main procedure and one or more than one sub-ordinate procedure which are operated under the main procedure.

But it's not mandatory to declare a procedure name like "main" as C/C++ compiler.

Procedure name also follow the rules of identifier rules, the format of write procedure name is:

PROCEDDURE_NAME <space> PROC   ; procedure begin

PROCEDDURE_NAME <space> ENDP ;procedure end

END <space>   PROCEDDURE_NAME ;exit from program

Here PROC is a keyword which means procedure
ENDP is also a keyword which means end procedure

END also a keyword normally written to exit from program.

Registers that are available in compiler emulator 8086
AX (Accumulator register)
BX (Base register)
CX (Counter register)
DX (Data register)

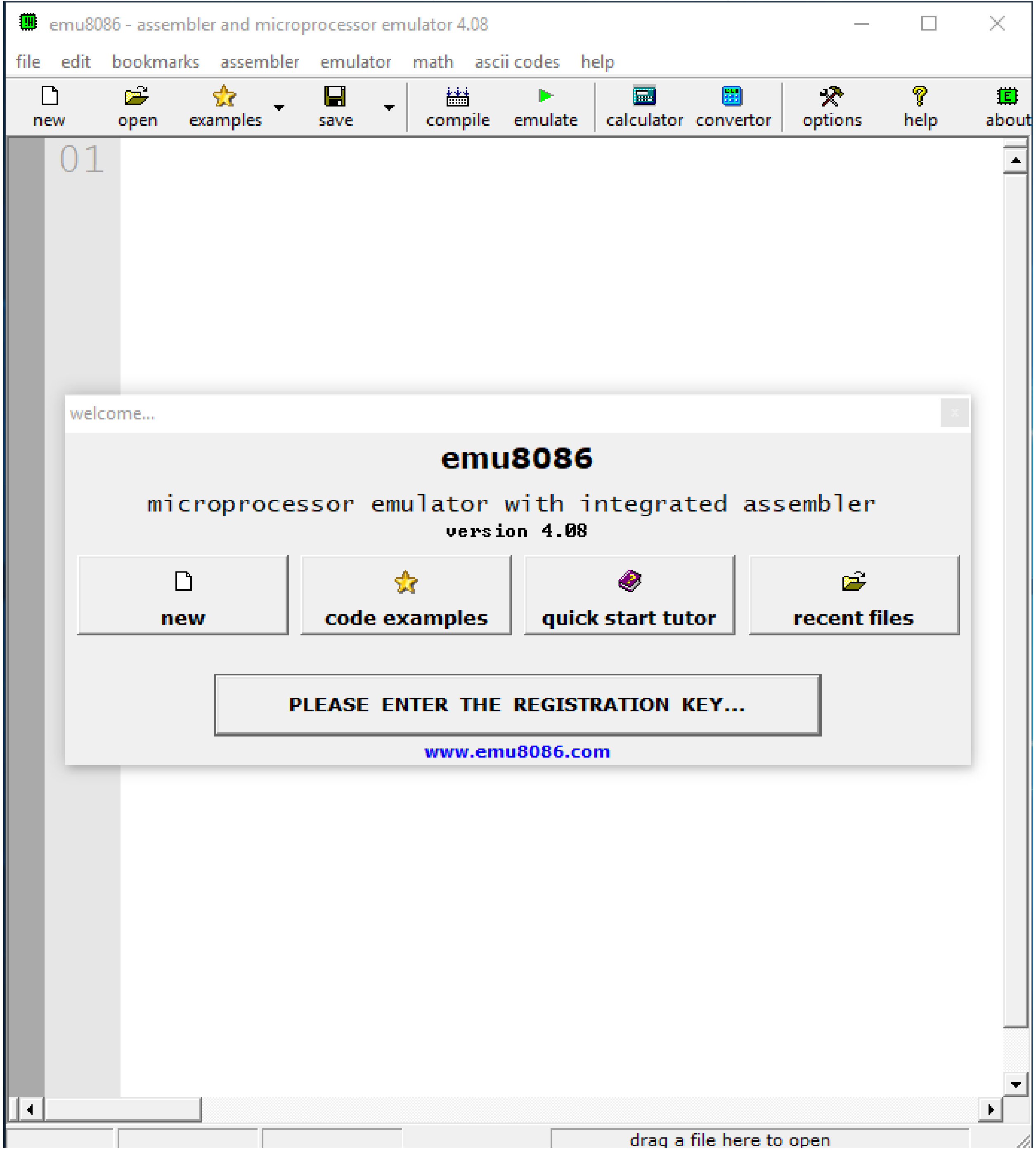
They all are 16 bit in size because here uses a symbol "X" after every register name which indicate registers are in word size (16 bit)

Every registers are divided into two separate bytes High('H') byte and Low('L') byte (8 bit in size)
AH, AL
BH, BL
CH, CL
DH,DL

These 8 registers variables are available when we write code in this compiler.
If we need more variables then we must declare it in data segment before used.

When we write code in a new file we select the new file as an "empty workspace"

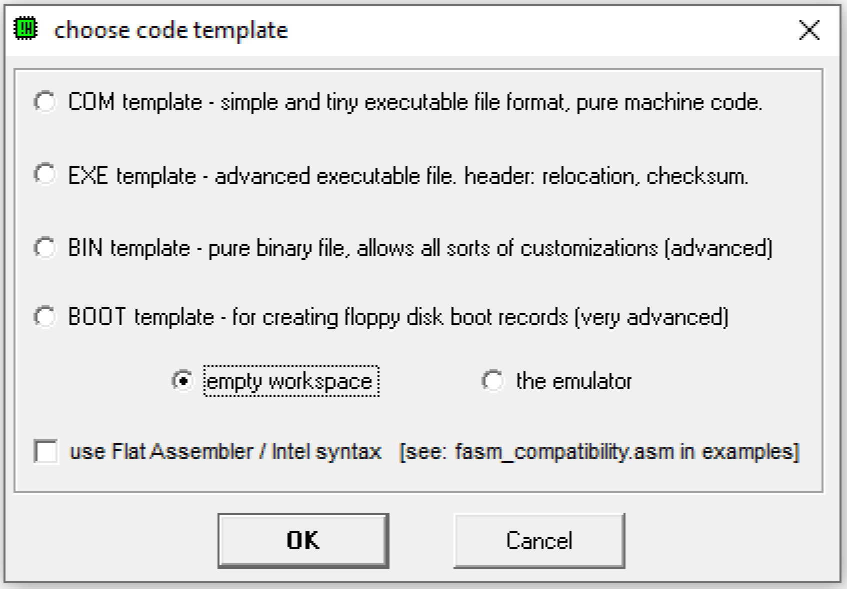
At first we open the emulator software by double click, the following window will show:



emu8086

microprocessor emulator with integrated assembler
version 4.08

Then we click NEW option, after that the following window will show:



choose code template

○ COM template - simple and tiny executable file format, pure machine code.

○ EXE template - advanced executable file. header: relocation, checksum.

○ BIN template - pure binary file, allows all sorts of customizations (advanced)

○ BOOT template - for creating floppy disk boot records (very advanced)

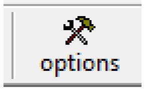
● empty workspace ○ the emulator

□ use Flat Assembler / Intel syntax [see: fasm_compatibility.asm in examples]

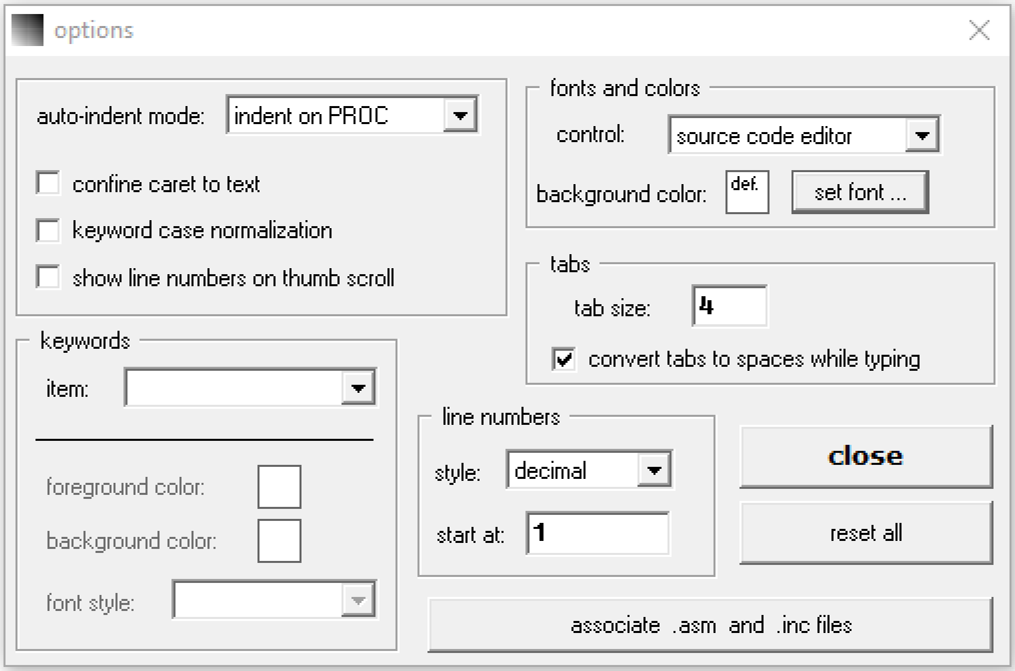
Here we select "empty workspace" and finally press OK button.

If you want to change the font size of this compiler you choose the following options:

At first press



Then the following window will come as follows



options

auto-indent mode: indent on PROC

□ confine caret to text

□ keyword case normalization

□ show line numbers on thumb scroll

keywords

item:

foreground color:

background color:

font style:

fonts and colors

control: source code editor

background color: def.

tabs

tab size: 4

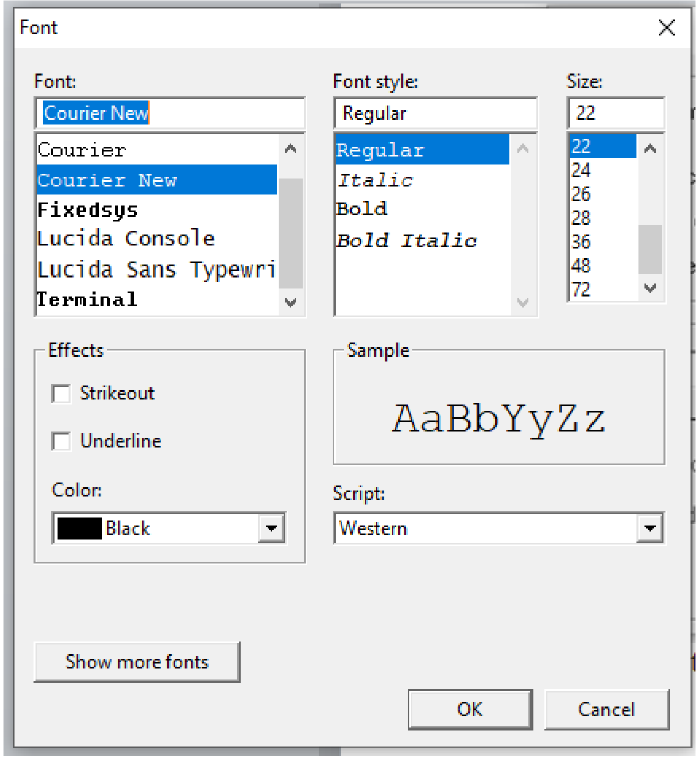
☑ convert tabs to spaces while typing

line numbers

style: decimal

start at: 1

You can select font size as you wish by pressing “set font” button as follows



Font

Font: Courier New

Courier
Courier New
Fixedsys
Lucida Console
Lucida Sans Typewri
Terminal

Font style: Regular

Regular
Italic
Bold
Bold Italic

Size: 22

22
24
26
28
36
48
72

Effects

□ Strikeout

□ Underline

Color: ■ Black

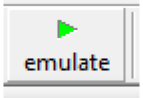
Sample

AaBbYyZz

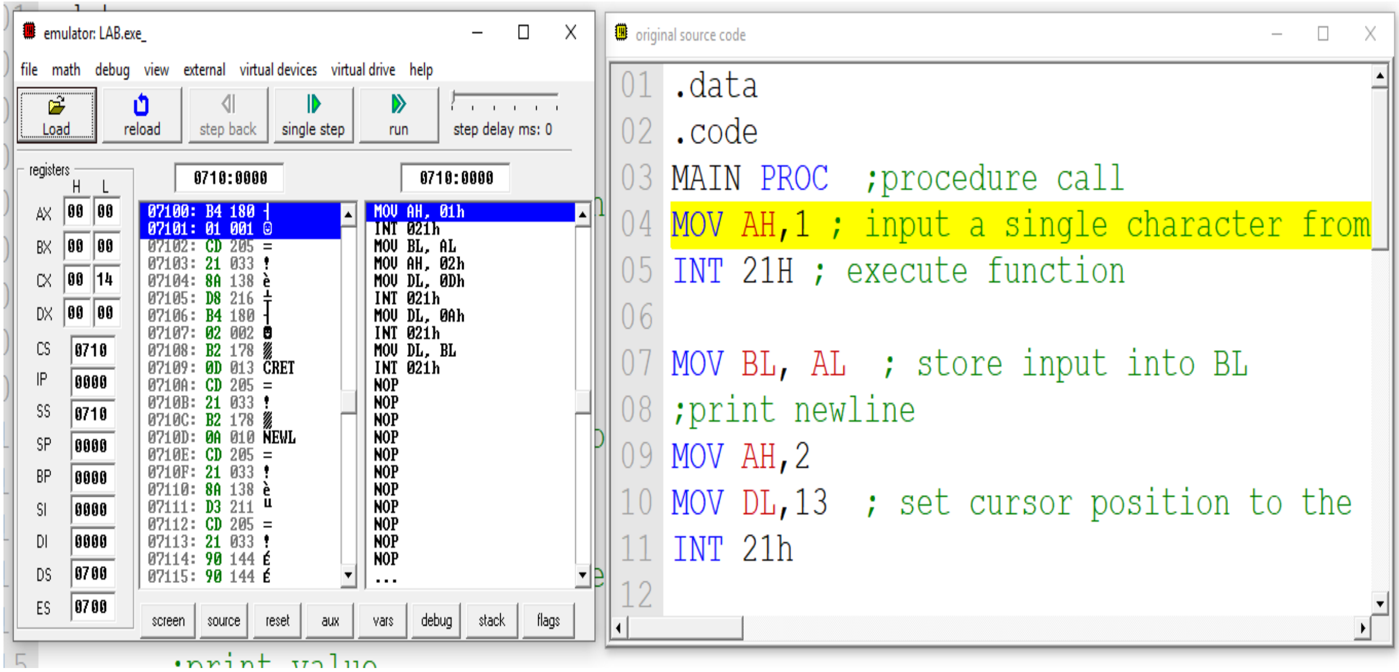
Script: Western

When we complete to write our code we must save the file with extension name assembly (.ASM) or (.asm).

When we want to run a program we press the following button



After pressing the above button you will show two window one is for debugging and other window contains your original source codes that indicates you which line compiler now compiling.


emulator: LAB.exe_
file math debug view external virtual devices virtual drive help
Load
reload
step back
single step
run
step delay ms: 0
registers
screen source reset aux vars debug stack flags
original source code

	H	L
AX	00	00
BX	00	00
CX	00	14
DX	00	00
CS	0710	
IP	0000	
SS	0710	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
0710:0000
07100: B4 180 ┤
07101: 01 001 ☺
07102: CD 205 ═
07103: 21 033 !
07104: 8A 138 è
07105: D8 216 ╪
07106: B4 180 ┤
07107: 02 002 ☻
07108: B2 178 ▓
07109: 0D 013 CRET
0710A: CD 205 ═
0710B: 21 033 !
0710C: B2 178 ▓
0710D: 0A 010 NEWL
0710E: CD 205 ═
0710F: 21 033 !
07110: 8A 138 è
07111: D3 211 ╙
07112: CD 205 ═
07113: 21 033 !
07114: 90 144 É
07115: 90 144 É
```

```
0710:0000
MOV AH, 01h
INT 021h
MOV BL, AL
MOV AH, 02h
MOV DL, 0Dh
INT 021h
MOV DL, 0Ah
INT 021h
MOV DL, BL
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```

```
.data
.code
MAIN PROC  ;procedure call
MOV AH,1 ; input a single character from
INT 21H ; execute function

MOV BL, AL  ; store input into BL
;print newline
MOV AH,2
MOV DL,13  ; set cursor position to the
INT 21h

```

The blue color indicates that this line compiler is now compiling.

In this software every input that we are taking from keyboard are stored in accumulator register especially in AL part.

When we want to show any value in monitor console window we have to use data register DL or DH it is built in this compiler software.

In every compiler there is some built in function for input and output as like C compiler for input we declare scanf() and for output we declare printf().

In emulator 8086 compiler,
When we declare a function which is I/O related we must use AH register of the accumulator register, it also built in.

For input a single character we have to store value 1 in AH, for this we write,
MOV   AH, 1

For print a single character we have to store value 2 in AH, for this we write,
MOV   AH, 2

For print string we have to store value 9 in AH, for this we write,
MOV   AH, 9

These are the built in function declaration. To operate these functions we must have to execute them that's why we must call this function.
To perform this operation we write

INT 21H ; interrupt routine which number is 33 in decimal or 21H in hexadecimal

If we don't write this line after function declaration actual operation will not perform.

Why we call this as interrupt:
In general CPU or microprocessor perform execution may be arithmetic or logical, we call them ALU (arithmetic and logic unit) operation.

To get input from keyboard or to show output in monitor CPU must have to connect with peripheral devices and these operations are not part of the microprocessor. That's why we call these operation as interrupt because here cpu must have to connect with peripheral devices to perform the required operation. To connect peripheral devices there is an interrupt routine which number is 33 in decimal or 21h in hexadecimal, so if we call this routine this operation will perform successfully. That's why we write   INT 21H or INT   33.

Here in this software if we want to write decimal number we don't need to write any symbol letter after the number that means if we write any number it will consider as decimal
For example

INT   33 ; 33 is decimal
MOV AH, 2 ; 2 is decimal

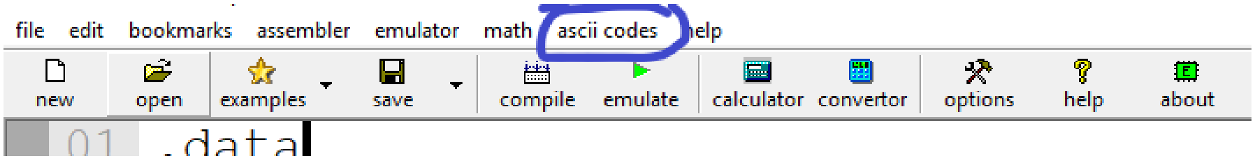
if we want to write hexadecimal number we need to write symbol 'h' or 'H' letter after the number then it will consider as hexadecimal
For example
INT   21h ; 21 is hexadecimal
MOV AH, 20h ; 20 is hexadecimal

if we want to write binary number we need to write symbol ‘B’ (or ‘Y’ some of the old compiler/assembler support) letter after the number then it will consider as binary
For example

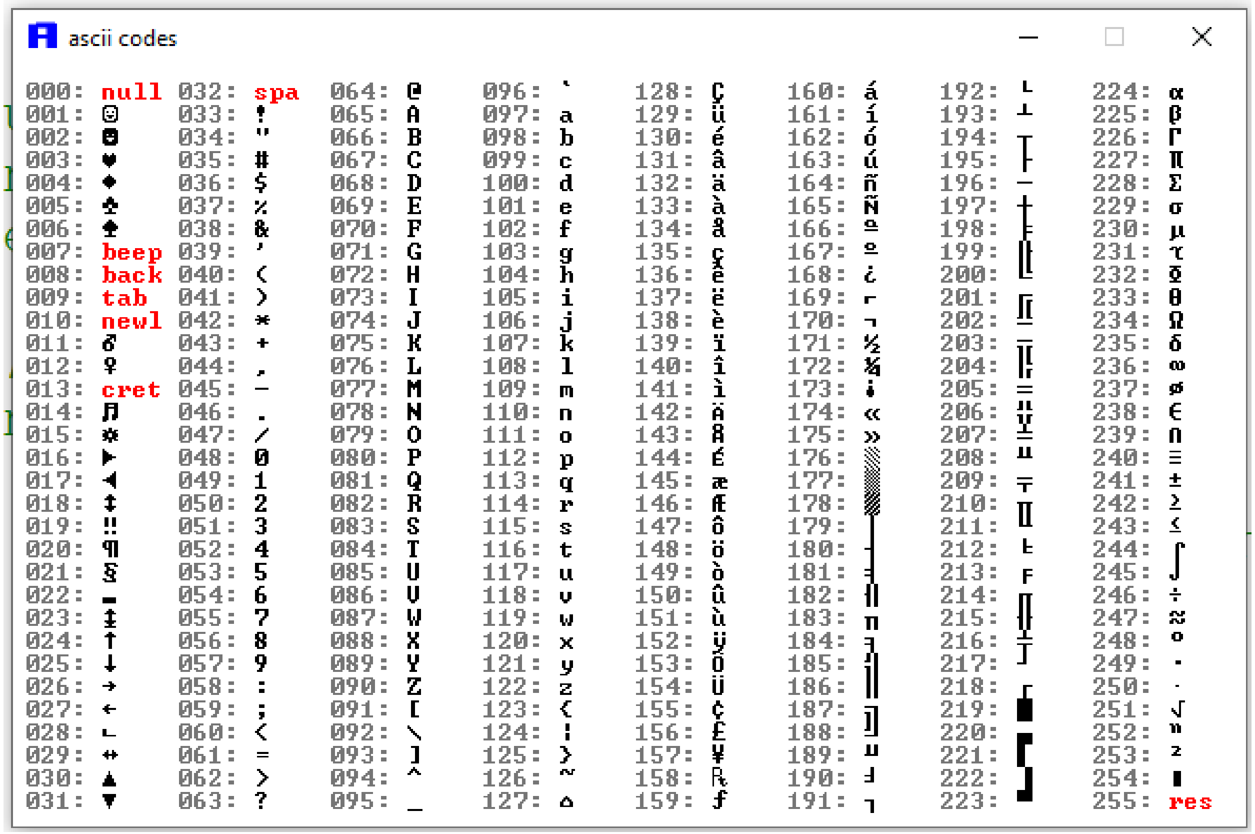
MOV BH, 10110011B ; 10110011 is binary

Here one important thing we must remember whatever we input from keyboard in this compiler it will take the corresponding ASCII value (in hexadecimal) of that character.

If you want to show the details of ASCII characters , you simply press the “ascii codes”

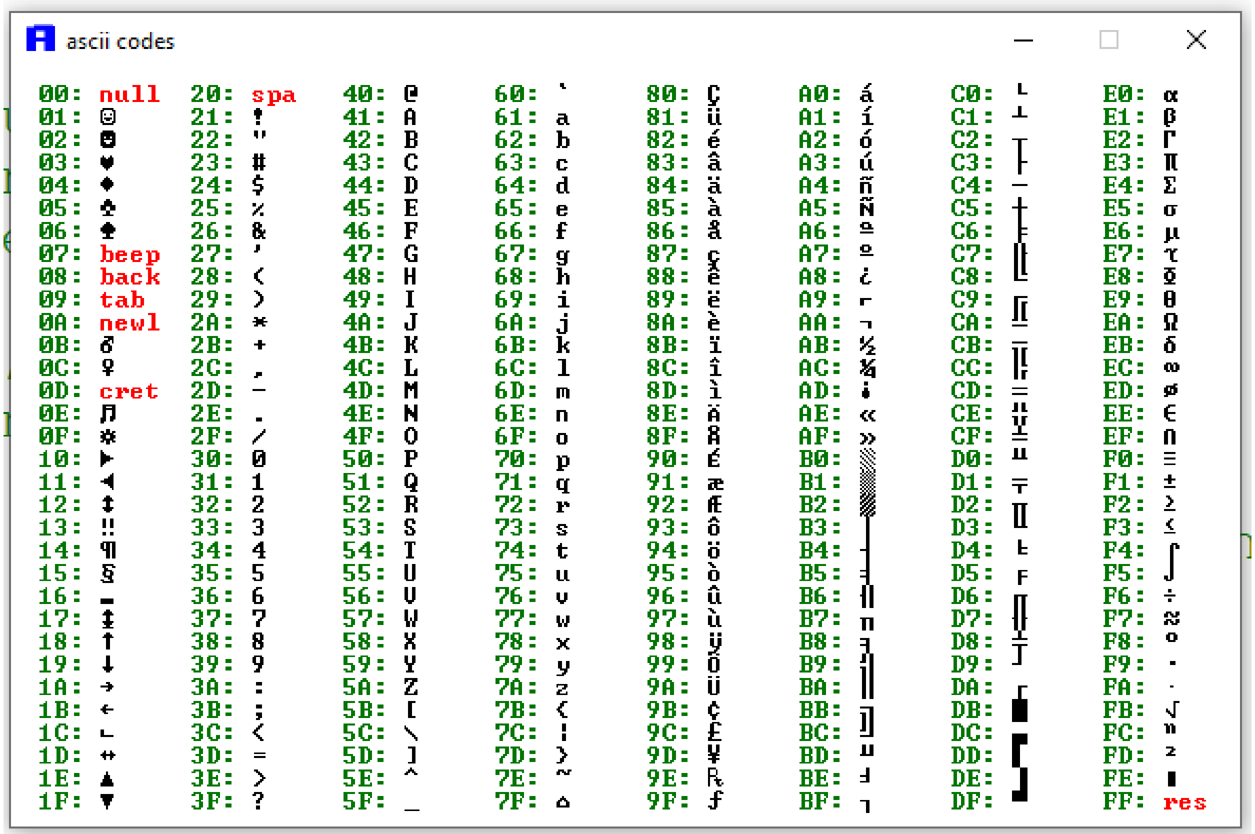


After press the “ascii codes” it will show the following window



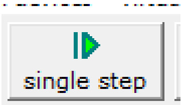
```
000: null 032: spa 064: @  096: `  128: Ç  160: á  192: └  224: α
001: ☺    033: !   065: A  097: a  129: ü  161: í  193: ┴  225: ß
002: ☻    034: "   066: B  098: b  130: é  162: ó  194: ┬  226: Γ
003: ♥    035: #   067: C  099: c  131: â  163: ú  195: ├  227: π
004: ♦    036: $   068: D  100: d  132: ä  164: ñ  196: ─  228: Σ
005: ♣    037: %   069: E  101: e  133: à  165: Ñ  197: ┼  229: σ
006: ♠    038: &   070: F  102: f  134: å  166: ª  198: ╞  230: µ
007: beep 039: '   071: G  103: g  135: ç  167: º  199: ╟  231: τ
008: back 040: (   072: H  104: h  136: ê  168: ¿  200: ╚  232: Φ
009: tab  041: )   073: I  105: i  137: ë  169: ⌐  201: ╔  233: Θ
010: newl 042: *   074: J  106: j  138: è  170: ¬  202: ╩  234: Ω
011: ♂    043: +   075: K  107: k  139: ï  171: ½  203: ╦  235: δ
012: ♀    044: ,   076: L  108: l  140: î  172: ¼  204: ╠  236: ∞
013: cret 045: -   077: M  109: m  141: ì  173: ¡  205: ═  237: φ
014: ♫    046: .   078: N  110: n  142: Ä  174: «  206: ╬  238: ε
015: ☼    047: /   079: O  111: o  143: Å  175: »  207: ╧  239: ∩
016: ►    048: 0   080: P  112: p  144: É  176: ░  208: ╨  240: ≡
017: ◄    049: 1   081: Q  113: q  145: æ  177: ▒  209: ╤  241: ±
018: ↕    050: 2   082: R  114: r  146: Æ  178: ▓  210: ╥  242: ≥
019: ‼    051: 3   083: S  115: s  147: ô  179: │  211: ╙  243: ≤
020: ¶    052: 4   084: T  116: t  148: ö  180: ┤  212: ╘  244: ⌠
021: §    053: 5   085: U  117: u  149: ò  181: ╡  213: ╒  245: ⌡
022: ▬    054: 6   086: V  118: v  150: û  182: ╢  214: ╓  246: ÷
023: ↨    055: 7   087: W  119: w  151: ù  183: ╖  215: ╫  247: ≈
024: ↑    056: 8   088: X  120: x  152: ÿ  184: ╕  216: ╪  248: °
025: ↓    057: 9   089: Y  121: y  153: Ö  185: ╣  217: ┘  249: ∙
026: →    058: :   090: Z  122: z  154: Ü  186: ║  218: ┌  250: ·
027: ←    059: ;   091: [  123: {  155: ¢  187: ╗  219: █  251: √
028: ∟    060: <   092: \  124: |  156: £  188: ╝  220: ▄  252: ⁿ
029: ↔    061: =   093: ]  125: }  157: ¥  189: ╜  221: ▌  253: ²
030: ▲    062: >   094: ^  126: ~  158: ₧  190: ╛  222: ▐  254: ■
031: ▼    063: ?   095: _  127: ⌂  159: ƒ  191: ┐  223: ▀  255: res
```

It shows decimal values of every ascii characters, you can also show the hexadecimal value of every ascii characters (support emulator8086 version 4.0 or higher but not lower version), for this you have to place the mouse cursor to the any one of the decimal value of the above chart window and then press the mouse left button , it will automatically show the following window



```
00: null   20: spa   40: @   60: `   80: Ç   A0: á   C0: └   E0: α
01: ☺      21: !     41: A   61: a   81: ü   A1: í   C1: ┴   E1: ß
02: ☻      22: "     42: B   62: b   82: é   A2: ó   C2: ┬   E2: Γ
03: ♥      23: #     43: C   63: c   83: â   A3: ú   C3: ├   E3: π
04: ♦      24: $     44: D   64: d   84: ä   A4: ñ   C4: ─   E4: Σ
05: ♣      25: %     45: E   65: e   85: à   A5: Ñ   C5: ┼   E5: σ
06: ♠      26: &     46: F   66: f   86: å   A6: ª   C6: ╞   E6: µ
07: beep   27: '     47: G   67: g   87: ç   A7: º   C7: ╟   E7: τ
08: back   28: (     48: H   68: h   88: ê   A8: ¿   C8: ╚   E8: Φ
09: tab    29: )     49: I   69: i   89: ë   A9: ⌐   C9: ╔   E9: Θ
0A: newl   2A: *     4A: J   6A: j   8A: è   AA: ¬   CA: ╩   EA: Ω
0B: ♂      2B: +     4B: K   6B: k   8B: ï   AB: ½   CB: ╦   EB: δ
0C: ♀      2C: ,     4C: L   6C: l   8C: î   AC: ¼   CC: ╠   EC: ∞
0D: cret   2D: -     4D: M   6D: m   8D: ì   AD: ¡   CD: ═   ED: φ
0E: ♫      2E: .     4E: N   6E: n   8E: Ä   AE: «   CE: ╬   EE: ε
0F: ☼      2F: /     4F: O   6F: o   8F: Å   AF: »   CF: ╧   EF: ∩
10: ►      30: 0     50: P   70: p   90: É   B0: ░   D0: ╨   F0: ≡
11: ◄      31: 1     51: Q   71: q   91: æ   B1: ▒   D1: ╤   F1: ±
12: ↕      32: 2     52: R   72: r   92: Æ   B2: ▓   D2: ╥   F2: ≥
13: ‼      33: 3     53: S   73: s   93: ô   B3: │   D3: ╙   F3: ≤
14: ¶      34: 4     54: T   74: t   94: ö   B4: ┤   D4: ╘   F4: ⌠
15: §      35: 5     55: U   75: u   95: ò   B5: ╡   D5: ╒   F5: ⌡
16: ▬      36: 6     56: V   76: v   96: û   B6: ╢   D6: ╓   F6: ÷
17: ↨      37: 7     57: W   77: w   97: ù   B7: ╖   D7: ╫   F7: ≈
18: ↑      38: 8     58: X   78: x   98: ÿ   B8: ╕   D8: ╪   F8: °
19: ↓      39: 9     59: Y   79: y   99: Ö   B9: ╣   D9: ┘   F9: ∙
1A: →      3A: :     5A: Z   7A: z   9A: Ü   BA: ║   DA: ┌   FA: ·
1B: ←      3B: ;     5B: [   7B: {   9B: ¢   BB: ╗   DB: █   FB: √
1C: ∟      3C: <     5C: \   7C: |   9C: £   BC: ╝   DC: ▄   FC: ⁿ
1D: ↔      3D: =     5D: ]   7D: }   9D: ¥   BD: ╜   DD: ▌   FD: ²
1E: ▲      3E: >     5E: ^   7E: ~   9E: ₧   BE: ╛   DE: ▐   FE: ■
1F: ▼      3F: ?     5F: _   7F: ⌂   9F: ƒ   BF: ┐   DF: ▀   FF: res
```

DEBUG

If we want to debug any program we have to press “single step” 

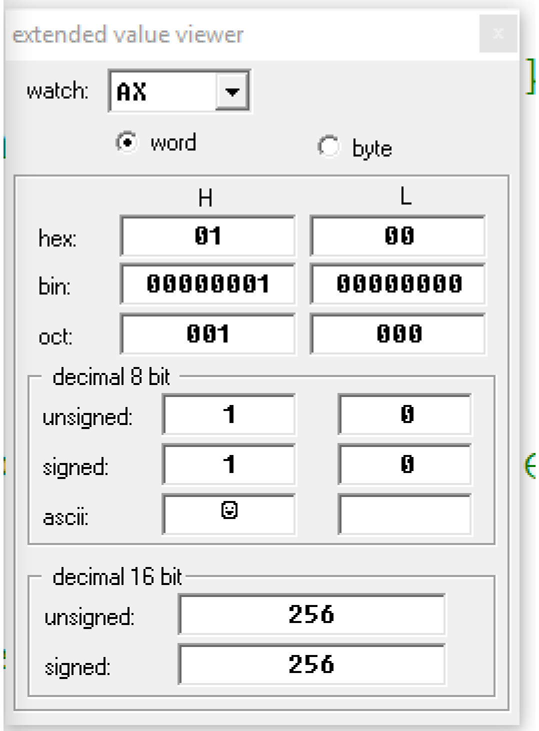
button. It will compile next line sequentially.

And if we want to go back one step back of the compiling we need to press “step back”

button.

You can show the specific values of every registers such as AL or AH or BL or BH or CL or CH or DL or DH by simply double click of the mouse left button. For this first you have to place the mouse left button on any one of the register and then press double click, you will show the folllowing window:

Here I press double click of the mouse left button on the AL part of the AX register.



extended value viewer

watch: AX

● word ○ byte

	H	L
hex:	01	00
bin:	00000001	00000000
oct:	001	000

decimal 8 bit

	H	L
unsigned:	1	0
signed:	1	0
ascii:	☺	

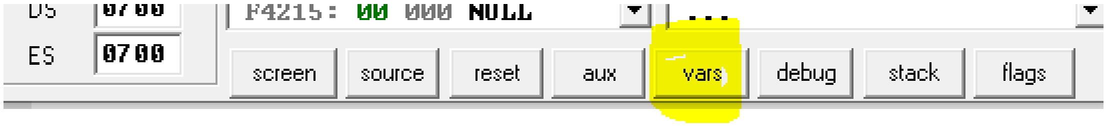
decimal 16 bit

unsigned: 256

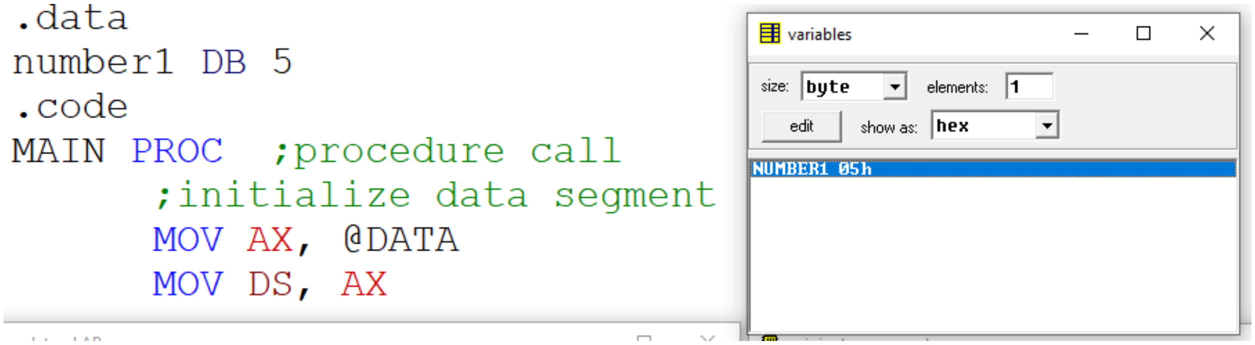
signed: 256

The above window will show you value of the register in hexadecimal, decimal, binary and octal of the corresponding ASCII character.

If we want to show the memory variables values when needs to execute program we simply click the button “vars”



The following window will show after press the button “vars”. As we declare variable name number1 and set default value 5, the window shows the variable name and it’s value in hexadecimal.

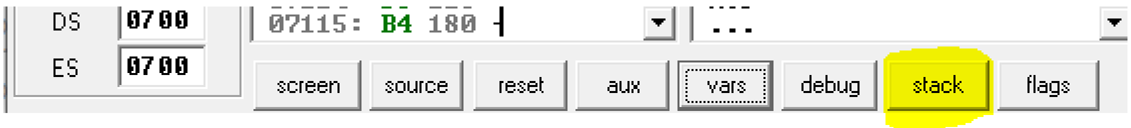


```
.data
number1 DB 5
.code
MAIN PROC  ;procedure call
       ;initialize data segment
       MOV AX, @DATA
       MOV DS, AX
```

size: byte   elements: 1
show as: hex

```
NUMBER1 05h
```

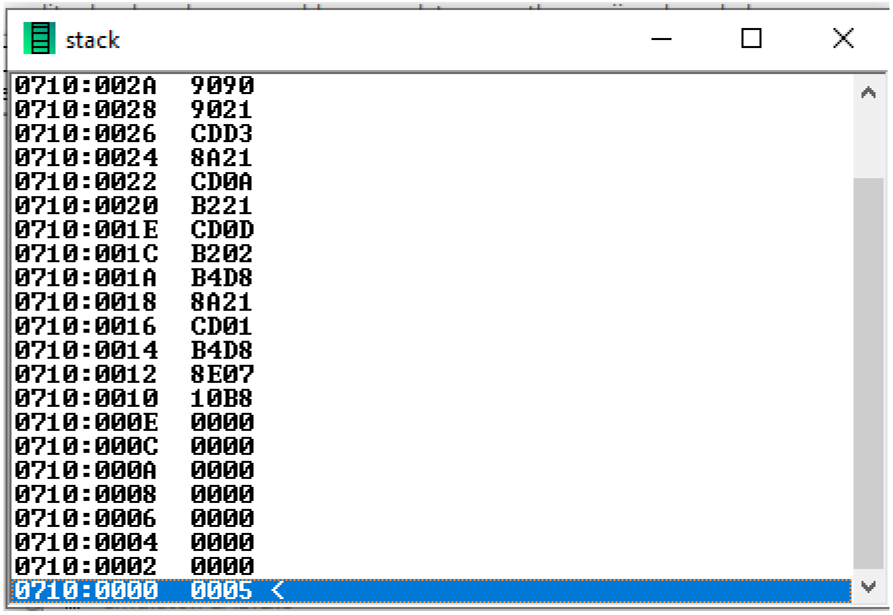
If we want to show the STACK memory values when needs to execute program we simply click the button “stack”



DS 0700
ES 0700

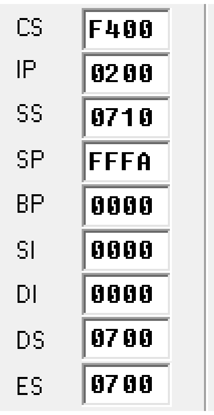
```
07115: B4 180 ┤
```

The following window will show after press the button “stack”. If we use stack value will show in third column, as we don’t use stack the third column is not show here. Blue color highlighting line indicates the starting position of the stack memory address or we can simply say the TOP of the stack location.

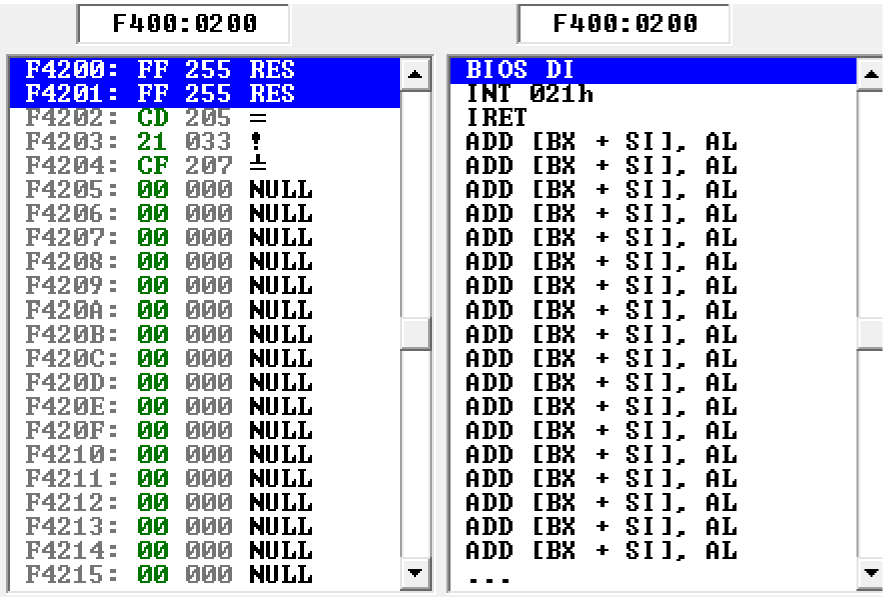


```
0710:002A  9090
0710:0028  9021
0710:0026  CDD3
0710:0024  8A21
0710:0022  CD0A
0710:0020  B221
0710:001E  CD0D
0710:001C  B202
0710:001A  B4D8
0710:0018  8A21
0710:0016  CD01
0710:0014  B4D8
0710:0012  8E07
0710:0010  10B8
0710:000E  0000
0710:000C  0000
0710:000A  0000
0710:0008  0000
0710:0006  0000
0710:0004  0000
0710:0002  0000
0710:0000  0005 <
```

The following two windows value is not mandatory to understand clearly. It's mention the physical address of the memory RAM portion, that means actually which RAM portion now used this compiler to perform the programming task.



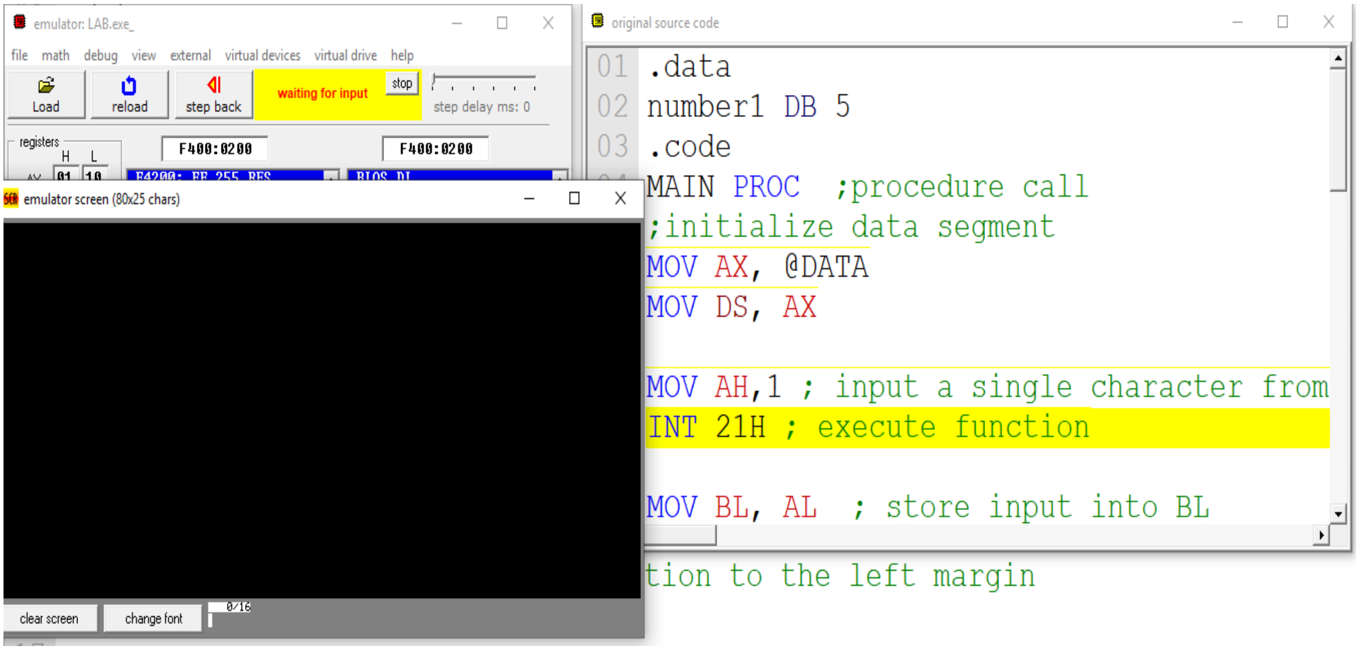
CS	F400
IP	0200
SS	0710
SP	FFFA
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700



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

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

At the time of debugging when it needs to take input from keyboard a console window will show as follows:

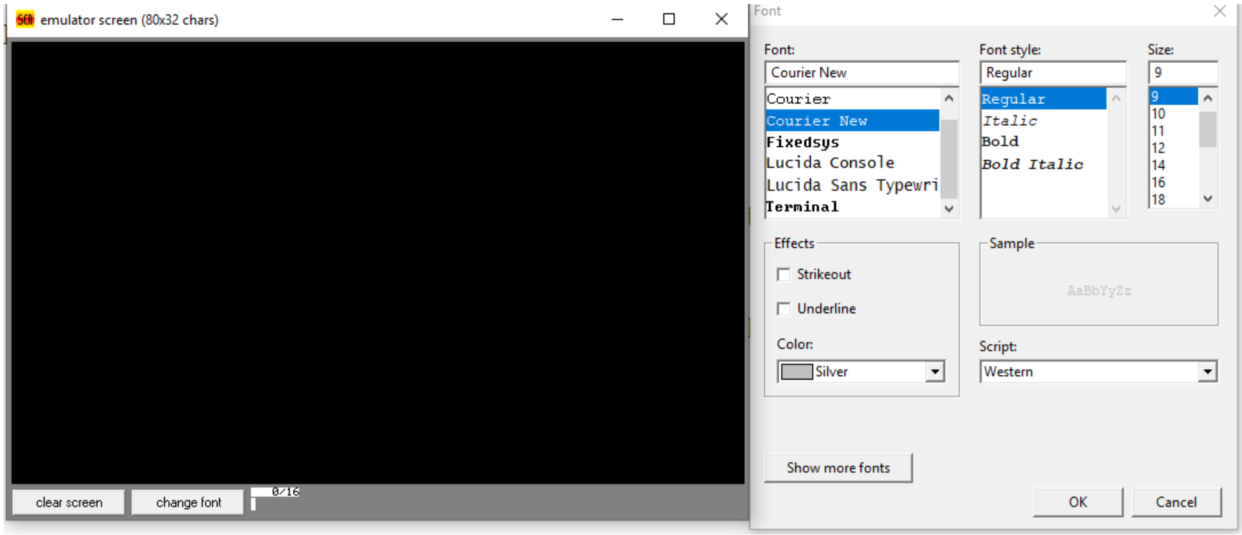


```
.data
number1 DB 5
.code
MAIN PROC   ;procedure call
;initialize data segment
MOV AX, @DATA
MOV DS, AX

MOV AH,1 ; input a single character from
INT 21H ; execute function

MOV BL, AL  ; store input into BL
tion to the left margin
```

You can also change font size of the console window, for this you have to press “change font” button of the console window which shows down side of the console window. When press “change font” button the following window will show as follows:



Font: Courier New

Font style: Regular

Size: 9

Color: Silver

Script: Western

If you select font “Courier New” you can see the font size range up to 72. You can choose as you want.

If you want to debug from the beginning you can press the “reload” reload button, it will go back to you the first line of your code.

In this emulator8086 we use most of the time data type “DB” which means define byte. There are also some other data types such as DW (define word), DQ (define quad word –four consecutive words) , DT (define ten bytes), DD (define double word –two consecutive words).