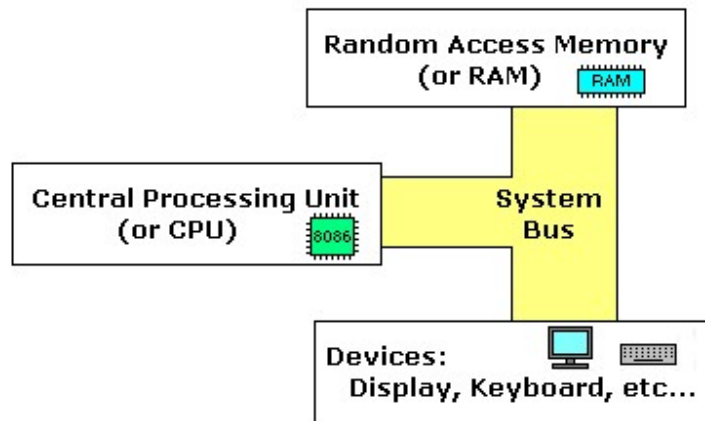


Lab#1: Introduction to Assembly Language and How to use the emulator.

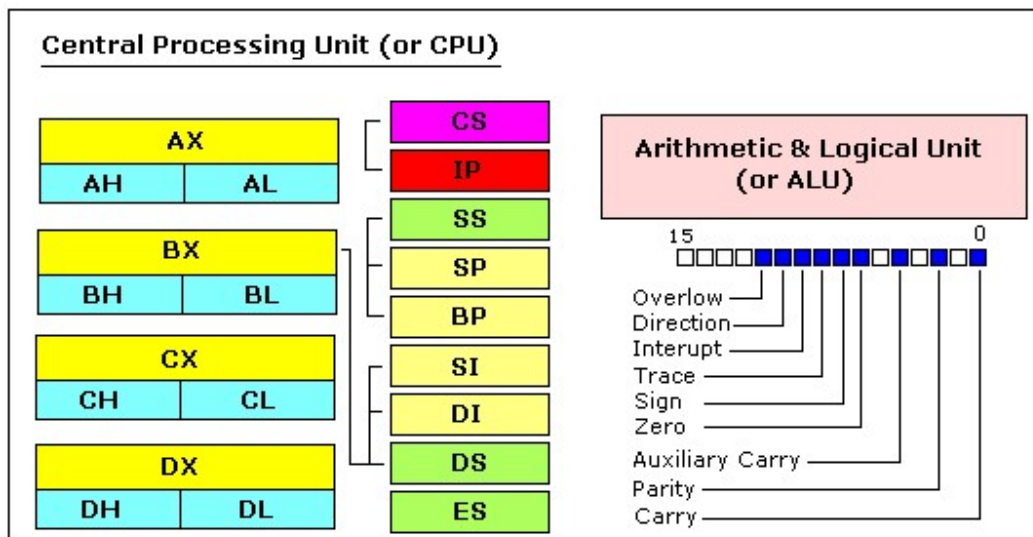
What is assembly language?

Assembly language is a low level programming language. you need to get some knowledge about computer structure in order to understand anything. The following diagram shows a simple computer model:



The **system bus** (shown in yellow) connects the various components of a computer. The **CPU** is the heart of the computer, most of computations occurs inside the **CPU**. **RAM** is a place to where the programs are loaded in order to be executed.

Inside the CPU



General purpose registers

8086 CPU has 8 general purpose registers, each register has its own name:

- **AX** - the accumulator register (divided into **AH** / **AL**).

- **BX** - the base address register (divided into **BH** / **BL**).
- **CX** - the count register (divided into **CH** / **CL**).
- **DX** - the data register (divided into **DH** / **DL**).
- **SI** - source index register.
- **DI** - destination index register.
- **BP** - base pointer.
- **SP** - stack pointer.

Despite the name of a register, it's the programmer who determines the usage for each general purpose register. The main purpose of a register is to keep a number (variable). The size of the above registers is 16 bit, it's something like: **0011000000111001b** (in binary form), or **12345** in decimal (human) form.

4 general purpose registers (AX, BX, CX, DX) are made of two separate 8 bit registers, for example if AX= **0011000000111001b**, then AH=**00110000b** and AL=**00111001b**. Therefore, when you modify any of the 8 bit registers 16 bit register is also updated, and vice-versa. The same is for other 3 registers, "H" is for high and "L" is for low part.

Segment registers

- **CS** - points at the segment containing the current program.
- **DS** - generally points at segment where variables are defined.
- **ES** - extra segment register, it's up to a coder to define its usage.
- **SS** - points at the segment containing the stack.

Although it is possible to store any data in the segment registers, this is never a good idea. The segment registers have a very special purpose - pointing at accessible blocks of memory.

Segment registers work together with general purpose register to access any memory value. For example if we would like to access memory at the physical address **12345h** (hexadecimal), we should set the **DS = 1230h** and **SI = 0045h**. This is good, since this way we can access much more memory than with a single register that is limited to 16 bit values.

CPU makes a calculation of physical address by multiplying the segment register by 10h and adding general purpose register to it($1230h * 10h + 45h = 12345h$):

$$\begin{array}{r} 12300 \\ + 0045 \\ \hline 12345 \end{array}$$


special purpose registers

- **IP** - the instruction pointer.
- **flags register** - determines the current state of the microprocessor.

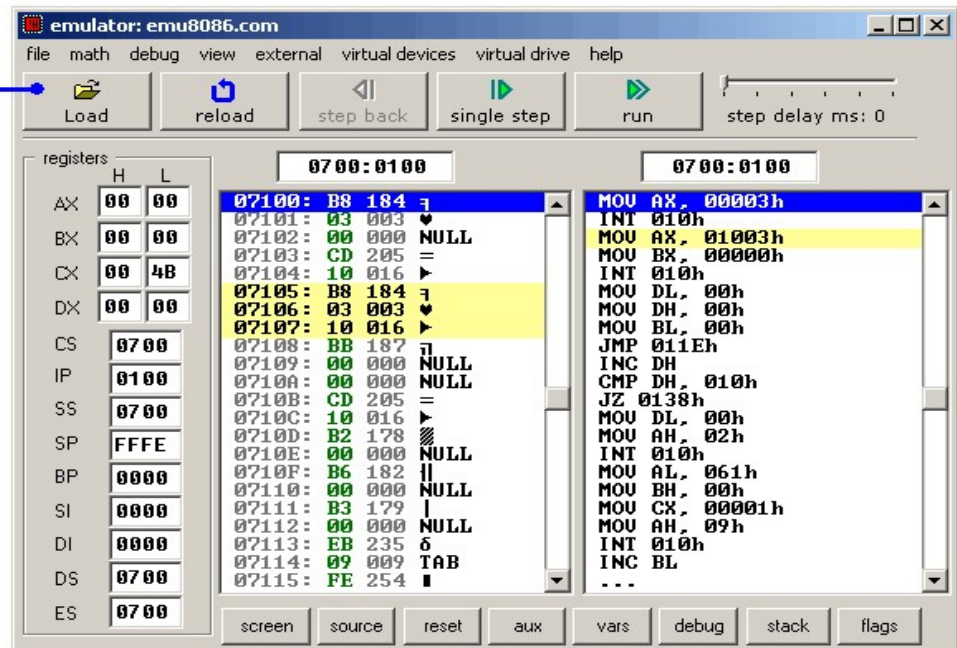
IP register always works together with **CS** segment register and it points to currently executing instruction.

flags register is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.

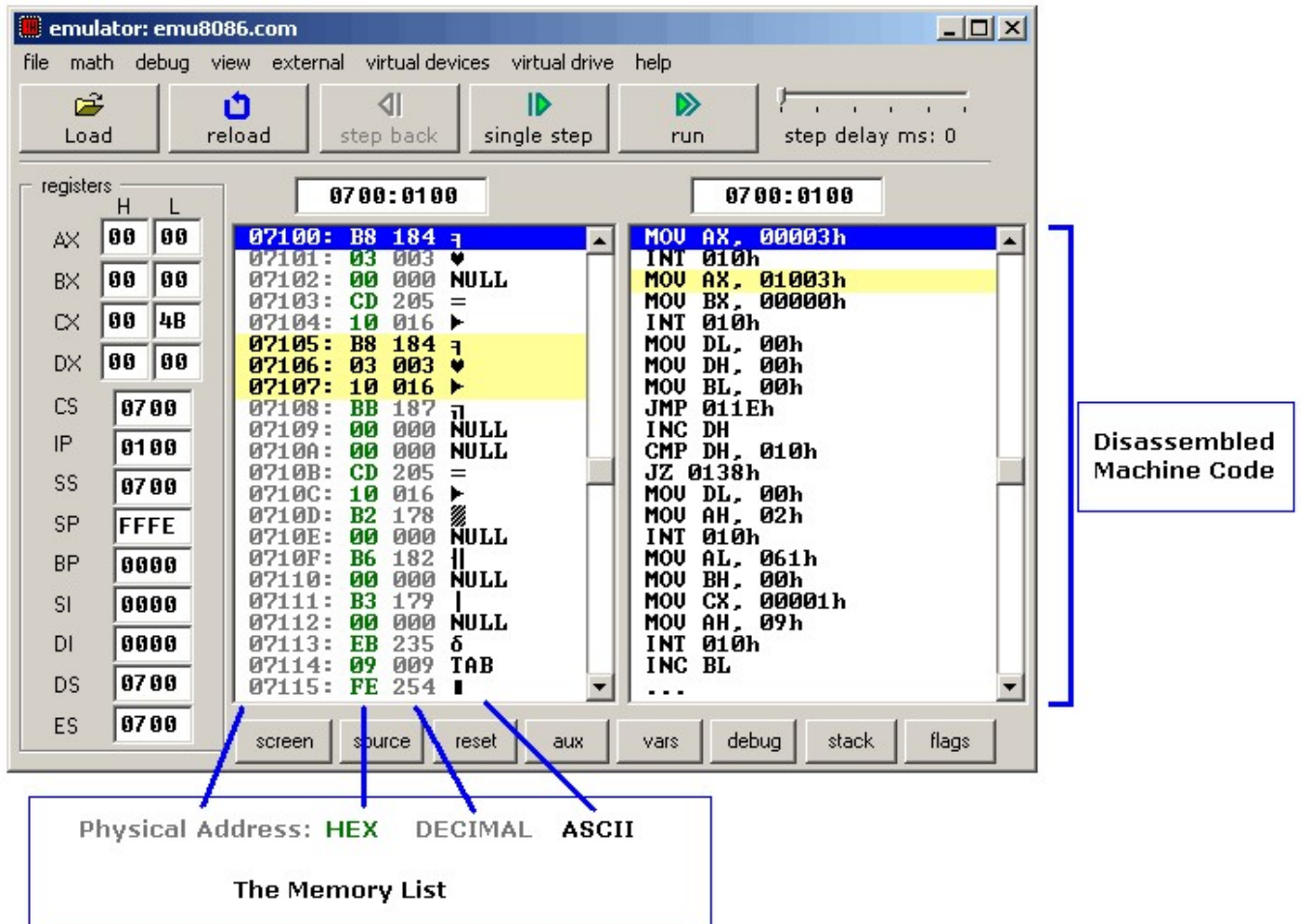
Using Emulator

If you want to load your code into the emulator, just click "Emulate" button . But you can also use emulator to load executables even if you don't have the original source code. Select **Show emulator** from the **Emulator** menu.

it's possible to load executables that do not have source codes



Try loading files from "MyBuild" folder. If there are no files in "MyBuild" folder return to source editor, select *Examples* from *File* menu, load any sample, compile it and then load into the emulator:



[Single Step] button executes instructions one by one stopping after each instruction.
 [Run] button executes instructions one by one with delay set by **step delay** between instructions.

Double click on register text-boxes opens "**Extended Viewer**" window with value of that register converted to all possible forms. You can modify the value of the register directly in this window.

Double click on memory list item opens "**Extended Viewer**" with WORD value loaded from memory list at selected location. Less significant byte is at lower address: LOW BYTE is loaded from selected position and HIGH BYTE from next memory address. You can modify the value of the memory word directly in the "**Extended Viewer**" window, you can modify the values of registers on runtime by typing over the existing values.

[Flags] button allows you to view and modify flags on runtime.

Lab#2: Data Definition and Addressing Modes.

Data Definition:

Assignment #1: Show the memory after defining the following variables:

```
ORG 100H
X1 DB ?
X2 DB 'ABC'
X3 DB 32
X4 DB 20H
X5 DB 01011001B
X6 DB 01, 'JAN'
X7 DB '32654'
X8 DB 3 DUP(0)

Y1 DW OFFF0H
Y2 DW 01011001B
Y3 DW X7
Y4 DW 3, 4, 17
Y5 DW 2 DUP(0)
Y6 DW X8-X7
HLT
```

Address	Offset	Value	Comment
0700:0100:	00	000	NULL
0700:0101:	41	065	A
0700:0102:	42	066	B
0700:0103:	43	067	C
0700:0104:	20	032	SPA
0700:0105:	20	032	SPA
0700:0106:	59	089	Y
0700:0107:	01	001	⊕
0700:0108:	4A	074	J
0700:0109:	41	065	A
0700:010A:	4E	078	N
0700:010B:	33	051	3
0700:010C:	32	050	2
0700:010D:	36	054	6
0700:010E:	35	053	5
0700:010F:	34	052	4
0700:0110:	00	000	NULL
0700:0111:	00	000	NULL
0700:0112:	00	000	NULL
0700:0113:	F0	240	≡
0700:0114:	FF	255	RES
0700:0115:	59	089	Y
0700:0116:	00	000	NULL
0700:0117:	0B	011	♂
0700:0118:	01	001	⊕
0700:0119:	03	003	♥
0700:011A:	00	000	NULL
0700:011B:	04	004	♦
0700:011C:	00	000	NULL
0700:011D:	11	017	◀
0700:011E:	00	000	NULL
0700:011F:	00	000	NULL
0700:0120:	00	000	NULL
0700:0121:	00	000	NULL
0700:0122:	00	000	NULL
0700:0123:	05	005	♠
0700:0124:	00	000	NULL

X1	00
X2	41
	42
	43
X3	20
X4	20
X5	59
X6	01
	4A
	41
	4E
X7	33
	32
	36
	35
	34
X8	00
	00
	00
Y1	F0
	FF
Y2	59
	00
Y3	0B
	01
Y4	03
	00
	04
	00
	11
	00
Y5	00
	00
	00
	00
Y6	05
	00

Addressing Modes:

Assignment #2: Initializing the internal registers of the 80x86 as follows:

(AX) = 0000H
(BX) = 0001H
(CX) = 0002H
(DX) = 0003H
(SI) = 0010H
(DI) = 0020H
(BP) = 0030H
(DS) = 0B60H

Fill all memory locations in the range DS:00 through DS:1F with 00H and then initialize the following storage locations:

(DS:0001H) = BBBBH
(DS:0004H) = CCCCH
(DS:0011H) = DDDDH
(DS:0014H) = EEEEH
(DS:0016H) = FFFFH

Using emu8086 to assemble the instructions

- (a) MOV AX, BX
- (b) MOV AX, 0AAAAh
- (c) MOV AX, [BX]
- (d) MOV AX, [4]
- (e) MOV AX, [BX+SI]
- (f) MOV AX, [SI+4]
- (g) MOV AX, [BX+SI+4]

Trace the execution of the instructions (a) through (g). Explain the execution of **each** instruction, including addressing mode, physical address for memory addressing mode, value in AX. Fill the table below.

Instruction	Addressing Mode	Physical Address	AX
a	Register A. M.		
b	Immediate A. M.		
c	Indirect A. M.		
d	Direct A. M.		
e	Based Indexed A. M.		
f	Indexed A. M.		
g	Based Indexed A. M.		

Solution:

ORG 100H

;;; Initializing Registers

;;; Fill the specified memory locations

;;; Instructions

HLT

Lab#3: Arithmetic Instructions.

Assignment #1: Complete the following codes to find the summation of the array elements

A. Array DB 20h, 0Ah, 32, 9

Sum DB ?

B. X DW 9, 120

Sum DW ?

Solution A:

ORG 100H

HLT

Array DB 20h, 0Ah, 32, 9

Sum DB ?

Solution B:

ORG 100H

HLT

X DW 9, 120

Sum DW ?

Assignment #2: Write an assembly code to add the following 2 numbers

Number 1: 123AB46H

Number 2: 14F2321H

Solution:

ORG 100H

HLT

X DD 123AB46H

Y DD 14F2321H

Z DD ?

Assignment #3: Determine the values of the flags (CF, AF, PF, SF, ZF, and OF) after executing the following code.

No.	Instructions		AF	CF	ZF	SF	PF	OF
1.	MOV AX, 0FFFFH INC AL XCHG AH, AL INC AX	AX =						
2.	MOV AX, 0FFFFH MOV BX, 0FFFFH ADD AX, BX	AX =						
3.	MOV AH, 0B7H MOV BH, 0EH ADD AH, BH	AH =						
4.	MOV AL, 80H MOV BL, 80H ADD AL, BL	AL =						
5.	MOV AX, 8000H MOV BX, 0001H SUB AX, BX	AX =						
6.	MOV AX, -5 MOV BL, -12 MOV BH, -12H	AX = BL = BH =						

Lab#4: Array Manipulations (Control Transfer Instructions).

Assignment #1: Write an assembly code to store the numbers (from 1 to 10) in the array x.

Solution:

```
ORG 100H
```

```
HLT
X DB 10 DUP(0)
```

Assignment #2: Write an assembly code to find the maximum value in the array N.
N DW 100H, 50H, 250H, 30H

Solution:

ORG 100H

HLT

N DW 100H, 50H, 250H, 30H

MAX DW ?

Assignment #3: Write an assembly code to exchange the values in the two arrays

Solution:

ORG 100H

HLT

X DB 1, 2, 3, 4

Y DB 5, 6, 7, 8

Assignment #4: Write an assembly code to exchange the values in the two arrays using stack (push and pop instructions)

Solution:

ORG 100H

HLT

X DB 1, 2, 3, 4

Y DB 5, 6, 7, 8

Assignment #5: Write an assembly code to perform division without using DIV or IDIV instructions.

X = 25, Y = 3

AL = X / Y

AH = X % Y

Solution:

ORG 100H

HLT

X DB 25

Y DB 3

Lab#5: Logical, Bit Manipulation and String instructions

Assignment #1: Write an assembly code to find number of ones in each number in the array X and store the result in the array Y

```
X DB 12H, 0F5H, 0A7H
Y DB 3 DUP(0)
```

Solution:

```
ORG 100H
```

```
HLT
X DB 12H, 0F5H, 0A7H
Y DB 3 DUP(0)
```

Assignment #2: Write an assembly code to replace the even numbers in the array X with zero (Note: without using DIV or IDIV instructions):

```
X DB 10, 3, 1, 2
```

Solution:

```
ORG 100H
```

```
HLT
X DB 10, 3, 1, 2
```

Assignment #3: Write an assembly code to count number of a's in the following string (Note: using SCASB instruction and REPNE prefix):

```
STR DB 'aabcadxyaaAa'
```

Solution:

```
ORG 100H
```

```
HLT
STR DB 'aabcadxyaaAa'
L EQU $-STR
```

Assignment #4: Write an assembly code to reverse the string STR1 in STR2

```
STR1 DB 'Assembly'  
LENGTH EQU ($-STR1)  
STR2 DB LENGTH DUP(0)
```

Solution:

ORG 100H

HLT

```
STR1 DB 'Assembly'  
LENGTH EQU ($-STR1)  
STR2 DB LENGTH DUP(0)
```

Lab#6: Interrupt Instructions (Part A)

INT 10H			
AH	Input Parameters	Output Parameter	Description
02	BH: page # DH: y coordinate (Row#) DL: x coordinate (Col#)		Position cursor to location (x,y) on the screen. Generally you would specify page zero. BIOS maintains a separate cursor for each page.
06	AL: Number of lines to scroll. BH: Screen attribute for cleared area. CL: x coordinate UL CH: y coordinate UL DL: x coordinate LR DH: y coordinate LR		Clear or scroll up. If AL contains zero, this function clears the rectangular portion of the screen specified by CL/CH (the upper left hand corner) and DL/DH (the lower right hand corner). If AL contains any other value, this service will scroll that rectangular window up the number of lines specified in AL.
07	AL: Number of lines to scroll. BH: Screen attribute for cleared area. CL: x coordinate UL CH: y coordinate UL DL: x coordinate LR DH: y coordinate LR		Clear or scroll up. If AL contains zero, this function clears the rectangular portion of the screen specified by CL/CH (the upper left hand corner) and DL/DH (the lower right hand corner). If AL contains any other value, this service will scroll that rectangular window down the number of lines specified in AL.
09	AL: character to display BH: page# BL: attribute CX: # of times to replicate Character		This call writes CX copies of the character and attribute in AL/BL starting at the current cursor position on the screen. It does not change the cursor's position.
INT 21H			
AH	Input Parameters	Output Parameter	Description
01		AL: char read	Reads a single character from the keyboard with echo (displays typed character on screen). if there is no character in the keyboard buffer, the function waits until any key is pressed.
02	DL: character to display		Writes a single character to the display.
09	DS:DX- pointer to string (terminated with "\$").		Display String: This function displays the characters from location DS:DX up to (but not including) a terminating "\$" character.
0A	DS:DX- pointer to input buffer.		Buffered Keyboard Input: This function reads a line of text from the keyboard and stores it into the input buffer pointed at by DS:DX. The first byte of the buffer must contain a count between one and 255 that contains the maximum number of allowable characters in the input buffer. This routine stores the actual number of characters read in the second byte. The actual input characters begin at the third byte of the buffer.

Assignment #1: Write an assembly code to do the following:

1. Clear the screen and change the background color to light gray and the foreground color to blue.
2. Display a message "enter a number between 0 and 9" in the screen.
3. Accept an input from the keyboard then clear the screen and test the number

- If the number is divisible by 3 then draw the following shape at the center

```
*
 *
  *
   *
    *
```

- Otherwise, draw the following shape (diagonal shape)

```
#
 #
  #
   #
    #
```

Solution:

ORG 100H

;;;;;;;;;;;;;;Clear the screen

;;;;;;;;;;;;;;Display a message

;;;;;;;;;;;;;;Input one character

;;;;;;;;;;;;;;Clear the screen

HLT

STR1 DB 'ENTER A NUMBER BETWEEN 0 AND 9:.'

S1 DB '*'

Lab#7: Interrupt Instructions (Part B)

Assignment #1: Write an assembly code to display 'Assembly' message, then moving down the message until reach the end of the screen

Solution:

```
ORG 100H
```

```
;;;;;;;;;;;;;Display a message
```

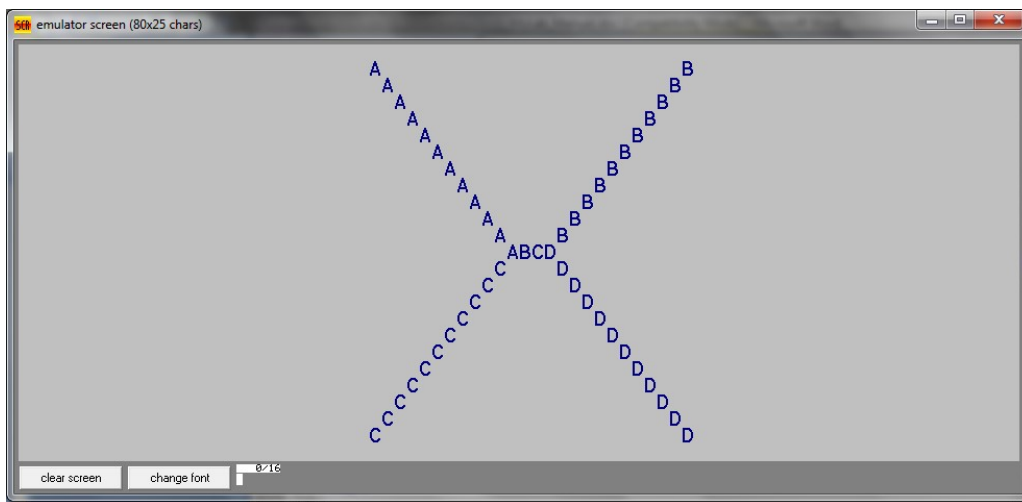
```
;;;;;;;;;;;;;Moving a message down (using scroll down interrupt function)
```

```
HLT
```

```
MSG DB 'Assembly$'
```

Assignment #2: Write an assembly code to do the following:

1. Display a message "enter 4 characters: ' in the screen.
2. Accept the 4 characters from the keyboard.
3. Clear the screen.
4. Display the 4 characters at the center of the screen, then display the characters as follows:
Example: if the user input: ABCD, the display must be as the following figure



Solution:**ORG 100H**

ENDM

;;;;;;;;;;Display a message

;;;;;;;;;;Accept an input (4 characters) from the keyboard

HLT

DATA DB 5, ?, 5 DUP(?)

MSG DB 'ENTER 4 CHARACTERS ONLY: \$'

R1 DB 11

C1 DB 38

R2 DB 13

C2 DB 43

Assignment #3: This example prints out "hello world!" by writing directly to video memory.

In VGA memory: first byte is ASCII character, byte that follows is character attribute. If you change the second byte, you can change the color of the character even after it is printed. The character attribute is 8 bit value; high 4 bits set background color and low 4 bits set foreground color.

HEX	BIN	COLOR
0	0000	black
1	0001	blue
2	0010	green
3	0011	cyan
4	0100	red
5	0101	magenta
6	0110	brown
7	0111	light gray
8	1000	dark gray
9	1001	light blue
A	1010	light green
B	1011	light cyan
C	1100	light red
D	1101	light magenta
E	1110	yellow
F	1111	white

Solution:

```
ORG 100H
;;;;;;;;;;;;; set video mode
MOV AX, 3  ;;;;;;;;;; text mode 80x25, 16 colors, 8 pages (ah=0, al=3)
INT 10H    ;;;;;;;;;; do it!
;;;;;;;;;;;;; set segment register:
MOV  AX, 0B800H
MOV  DS, AX
;;;;;;;;;;;;; print "hello world"
;;;;;;;;;;;;; first byte is ascii code, second byte is color code.
MOV [02h], 'H'
MOV [04h], 'e'
MOV [06h], 'l'
MOV [08h], 'l'
MOV [0ah], 'o'
MOV [0ch], ','
MOV [0eh], 'W'
MOV [10h], 'o'
MOV [12h], 'r'
MOV [14h], 'l'
MOV [16h], 'd'
MOV [18h], '!'
;;;;;;;;;;;;; color all characters:
MOV CX, 12 ;;;;;;;;;; number of characters.
MOV DI, 03H ;;;;;;;;;; start from byte after 'h'
C:
MOV [DI], 11101100B ; light red(1100) on yellow(1110)
ADD DI, 2 ;;;;;;;;;; skip over next ascii code in vga memory.
LOOP C
; ;;;;;;;;;; wait for any key press:
MOV AH, 0
INT 16H
HLT
```