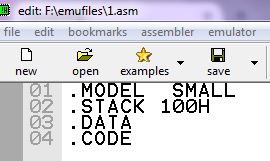
<http://www.rasmurtech.com/free-online-video-classes/programming/assembly-language-programming-tutorial/>

ASSEMBLY LANGUAGE

**ASCII VALUE :** Green **hexa value**

Black **decimal value**

**নিচের ৪টা লাইন সব কোড এই থাকবে। এই ৪ টার sequence ও maintain করতে হবে।**



Next line হল :

PROC হল একটা keyword.

MAIN PROC

এখানে MAIN হল procedure এর নাম।Procedure থেকে execution start হবে।

**এখন আমরা দেখব code এর structure : ১ টা লাইন এ এই চার ধরনের component থাকতে পারে।**

Name: Opcode Operand(s); Comment

❶ **(optional)** ❷ ❸ ❹ **(optional)**

❶ NAME :

1. যেকোনো variable এর নামে allowed. c/c++ এ যেমন তেমন।
2. এটা optional. না দিলেও problem nai.
3. NAME এর পর colon ( : ) দিতেই হবে।

❷ OPCODE : ( Operation code/ command )

1. এটা হল mainly operator.
2. একটা single line এ ১টাই opcode লিখা যায়।
3. যদি একাধিক operator নিয়ে কাজ করতেই হয় তাহলে ১ লাইন এর কাজ একাধিক লাইন এ ভেঙ্গে ভেঙ্গে করতে হবে।

বিভিন্ন operation এর জন্য বিভিন্ন opcode আছে। যেমন :

1.MOV- Assignment operator variable এর ক্ষেত্রে যেই কাজ করে। এক্ষেত্রে MOV operator register এর ক্ষেত্রে same কাজ করবে। MOV হল move করা।

EXAMPLE: MOV AH , 6 ( এক্ষেত্রে AH নামক Register এ 6 store হবে।

এক্ষেত্রে

2.ADD- এর কাজ হল 2 টা operand কে যোগ করা।

EXAMPLE : BH=2 ,CH=5

ADD CH,BH ; CH=CH+BH (CH এই যোগফল টা store হয়। )

Syntax : ADD destination ,source

3.SUB –

4. ++/-- : MOV BL,2

MOV BH,4

5. XCHG : swapping .

Example : XCHG BL,CH

But XCHG BL,2 -> this is an error.

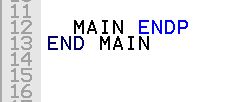
❸ OPERAND(s) :

1. Multiple operand থাকলে তার জন্য comma( ,) use করতে হয়।

❹ COMMENT :

1. Semicolon ( ;) এর পর যা লিখব তাই comment .

Last part হল :



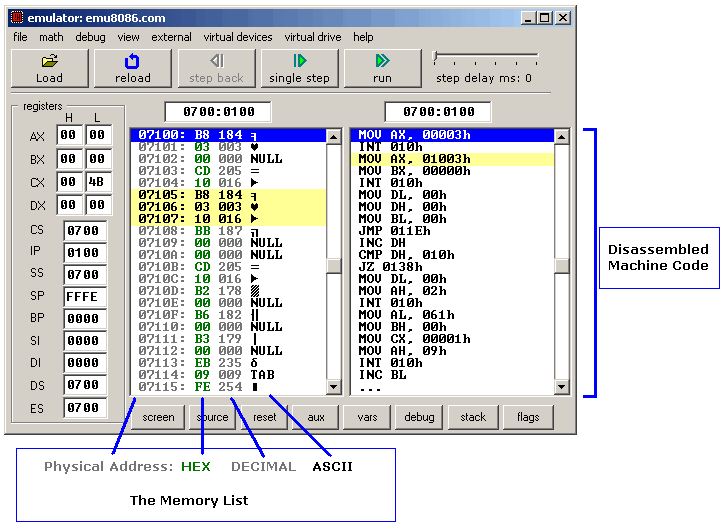
MAIN ENDP : আগে C/C++ এ যখন function use করতাম তখন scope দিতে হত। but এখন সেটা নাই। ENDP মানে হল end of procedure.

END MAIN : এটা দিয়ে পুরা program এর end বুঝান হয়।

**EMU8086 এর নিয়ে কিছু বকরবকর :**

EMU8086 নিয়ে যানতে আপনার আদিক অদিক করা লাগবেনা। just simulator টা open করেন। অতপর একদম top bar এ গিয়ে help button এ click করেন। অতপর DOCUMENTATION AND TUTORIAL option টাতে click করেন।

১। এটা একটা emulator.

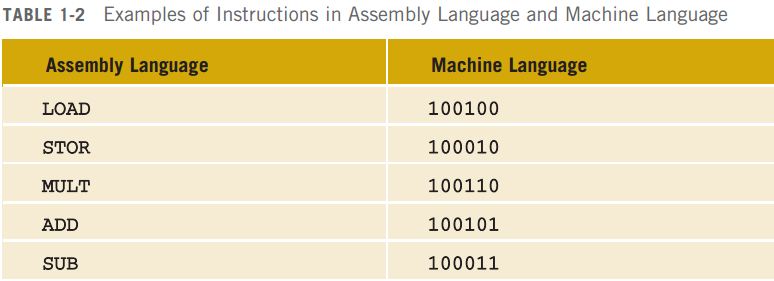


 এটা হল CS:IP . 1st এরটা code segment আর তার পরেরটা IP. একটা instructiobn execute হলে IP এর value “update” হয়।পরের instruction কে point করে রাখে।

**ASSEMBLY LANGUAGE কি?**

Machinye language Assembly Language High Level Language

( low level language) (mid level language) (high level language)



The most basic language of a computer, the machine language, provides program instructions in bits. Even though most computers perform the same kinds of operations ,the designers of the computer may have chosen different sets of binary codes to perform the operations. Therefore, the machine language of one machine is not necessarily the same as the machine language of another machine. The only consistency among computers is that in any modern computer, all data is stored and manipulated as binary codes.

Early computers were programmed in machine language. To see how instructions are written in machine language, suppose you want to use the equation:

wages = rate\*hours

to calculate weekly wages. Further, suppose that the binary code 100100 stands for load, 100110 stands for multiplication, and 100010 stands for store. In machine language, you might need the following sequence of instructions to calculate weekly wages:

100100 010001

100110 010010

100010 010011

To represent the weekly wages equation in machine language, the programmer had to remember the machine language codes for various operations. Also, to manipulate data, the programmer had to remember the locations of the data in the main memory.

This need to remember specific codes made programming not only very difficult, but also error prone.

Assembly languages were developed to make the programmer’s job easier. In assembly language, an instruction is an easy-to-remember form called a mnemonic. Table 1-2 shows some examples of instructions in assembly language and their corresponding machine language code.

It is much easier to write instructions in assembly language. However ,a computer cannot execute assembly language instructions directly. The instructions first have to be translated into machine language. A program called an assembler translates the assembly language instructions into machine language.

Assembler: A program that translates a program written in assembly language into an equivalent program in machine language.

Moving from machine language to assembly language made programming easier, but a programmer was still forced to think in terms of individual machine instructions. The next step toward making programming easier was to devise high-level languages that were closer to natural languages, such as English, French, German, and Spanish. Basic, FORTRAN, COBOL, Pascal, C, C++, C#, and Java are all high-level languages.

The instruction written in C++ is much easier to understand and is self-explanatory to a novice user who is familiar with basic arithmetic. As in the case of assembly language, however, the computer cannot directly execute instructions written in a high-level language. To run on a computer, these C++ instructions first need to be translated into machine language. A program called a compiler translates instructions written in highlevel languages into machine code.

Compiler: A program that translates instructions written in a high-level language into the equivalent machine language.

**INTEL 8086 PROCESSOR:**

16 bit microprocessor. এটার architecture এর উপর base করে নতুন processor গুলো তৈরি।

Instruction in emu8086 Intel 8086 এর মত করে দেখানো।

(emulates)

**SIMULATOR VS EMULATOR :**

Simulation = For analysis and study

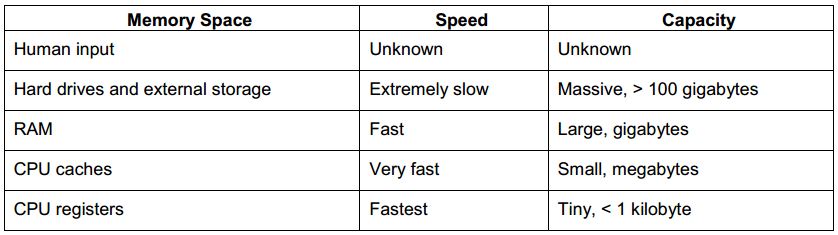
Emulation = For usage as a substitute

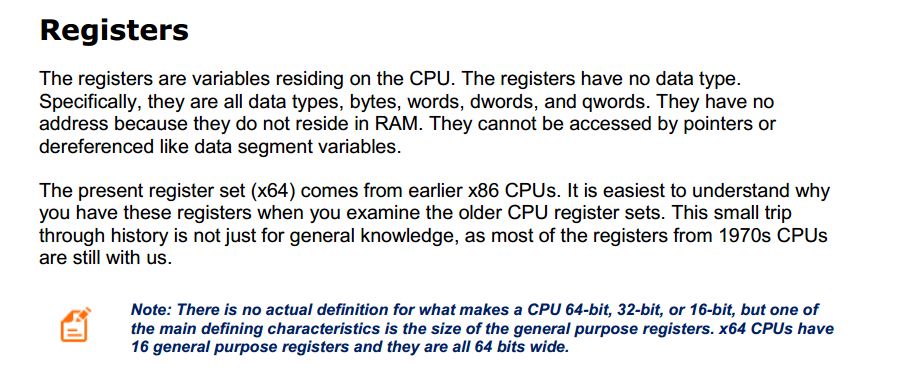
A simulator is an environment which models but an emulator is one that replicates the usage as on the original device or system.

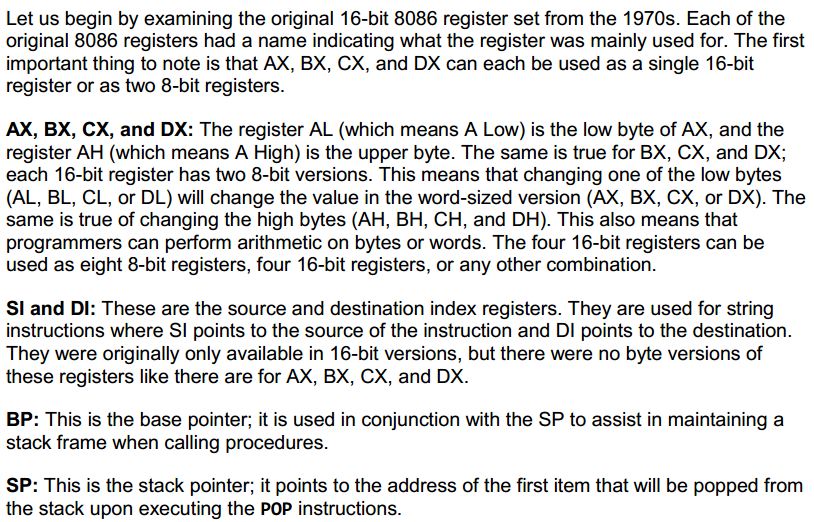
Simulator mimics the activity of something that it is simulating. It "appears"(a lot can go with this "appears", depending on the context) to be the same as the thing being simulated. For example the flight simulator "appears" to be a real flight to the user, although it does transport you from one place to another.

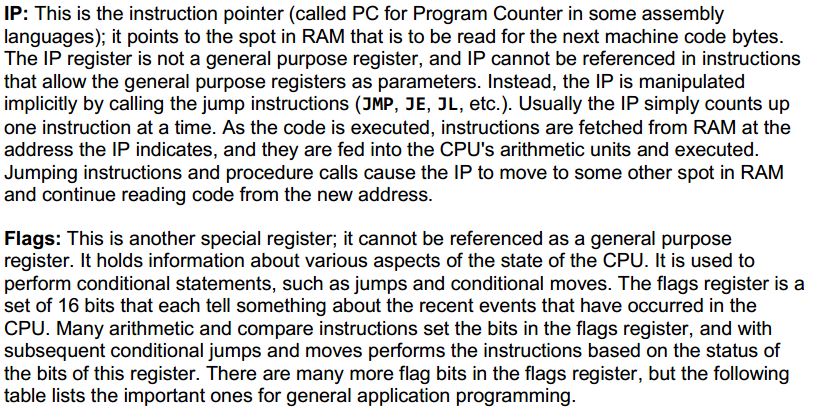
Emulator, on the other hand, actually "does" what the thing being emulated does, and in doing so it too "appears to be doing the same thing". An emulator may use different set of protocols for mimicking the thing being emulated, but the result/outcome is always the same as the original object. For example, EMU8086 emulates the 8086 microprocessor on your computer, which obviously is not running on 8086(=different protocols), but the output it gives is what a real 8086 would give.

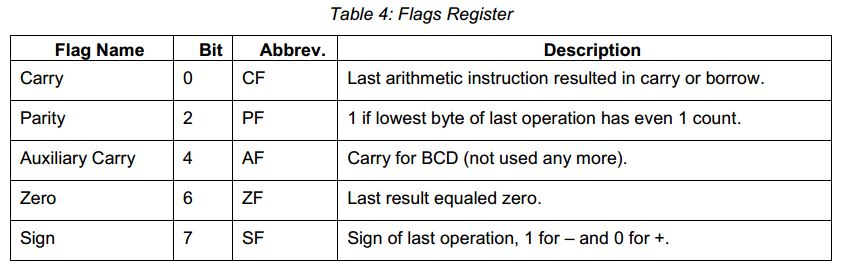
**REGISTER :**

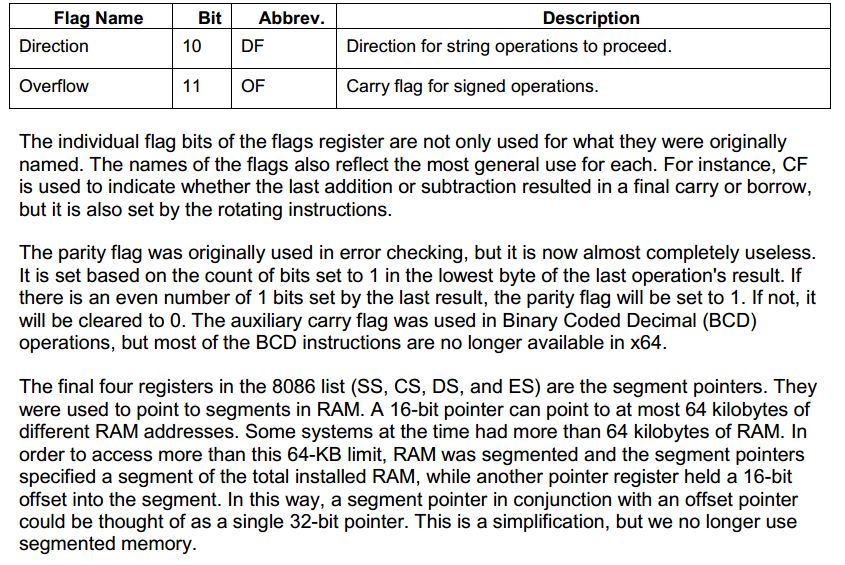


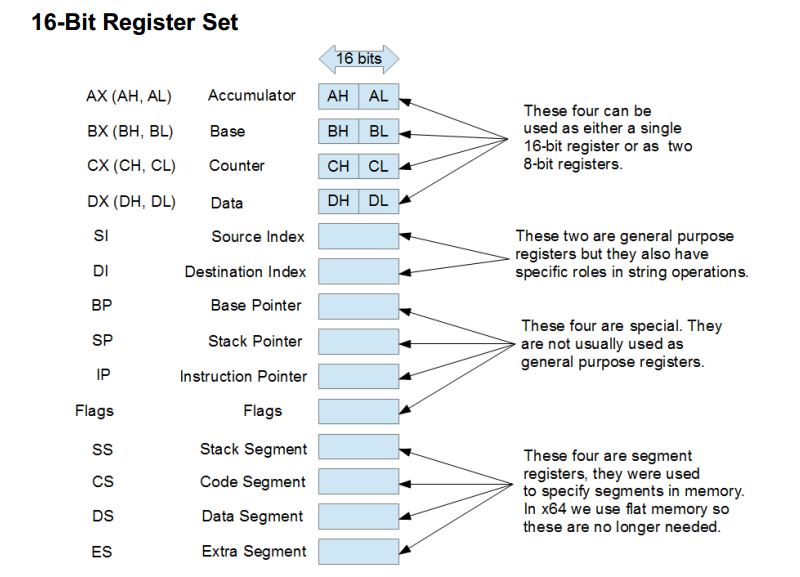








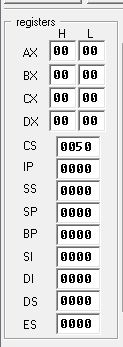




* CPU এর সাথেই থাকে।
* RAM থেকে কিছু নিয়ে আসলে টা register এ store করে রাখে।
* 8086 microprocessor এ total 14 টা register আছে।
* এদের বলে word register.এদের প্রত্যেকের size 16 bit/2 byte/1word.

Register গুলো হল :

1. AX
2. BX এই চারটা register কে দুই ভাগে ভাগ করা যায়।তখন
3. CX ওদের বলে byte register যেহেত half হয়ে যাই তাই তখন size হয়
4. DX ১ byte = ৮ bit. একটাকে বলে upper half আরেকটাকে বলে
5. CS lower half. নিচে pic দেখ। আর আমদের কোড এ এই চারটা register
6. IP নিয়েই কাজ করব।
7. SS
8. SP
9. BP
10. SI
11. DI
12. DS
13. ES



**SECTIONS:**

❶ .MODEL SMALL

❷ .STACK 100H

Stack segment এর address এর syntax হল : SS:SP

❸ .DATA ( Data section always code section এর আগে বসবে )

DATA segment এর address এর syntax হল : DS:

❹ .CODE (opcode এর section এটা)

Code segment এর address এর syntax হল : CS:IP

❶ .MODEL SMALL :

❷ .STACK 100H :

❸ .DATA :

এই sector এর নিচে data variable define করা হয়।

Syntax : name(c এর নিয়ম ) define\_type initial value

a DB 65 (value initialize করলে)

অথবা a DB ? (কোন value initialize না করলে)

বিভিন্ন ধরনের define type নিম্নে দেয়া হল :

• DB = Define Byte (size : 8bit/1byte) example : ‘y’

• DW=Define Word (size : 16bit/2byte) example : 12345/-12345

• DD= Define Double Word (size : 32bit /4byte) example : 1.234

• DQ= Define Quad Word (size : 64bit/8 byte) example : 12345678910

• DT=Define Ten byte (size : 80 bit/10byte) example: 10^10

String এর ক্ষেত্রে :

Syntax : Name DB ‘…..$’

String এর ক্ষেত্রে প্রত্যেক character এর জন্য ৮ বিট জাইগা লাগে। তাই ১ টা করে DB লাগে প্রত্যেকটা character এর জন্য।

এক্ষেত্রে MAIN PROC এর পর নিম্নোক্ত লাইন দুটি দিখতে হবে :

MOV AX/BX/CX/DX , @DATA (.DATA segment এর address টাকে AX এ রাখলাম )

MOV DS , AX/BX/CX/DX (সাথে সাথে AX/BX/CX/DX এর address টা DS এ কপি করে রাখবে। )

**Rules and syntax for string type data :**

* **String কে double quation/single quation এর মধ্যে রাখতে হবে।**
* **Assembly Language এ string লিখার পর একটা ending character ($) লিখে দিতে হবে quation এর ভেতরে।**

**Example : “I am kawser$”**

* **String declare করলে নিম্নোক্ত code লিখতে হয় :**

**MOV AH,9**

**LEA DX , variable name (LEA= load effective address)**

**INT 21H**

**এক্ষেত্রে register হিসেবে DX ই রাখতে হবে। coz address 16 bit. আমরা আগেই বলেছিলাম যে string type এর ক্ষেত্রে ($) sign দিতে হয়। এটার সুবিধা পায়া যাবে যখন আমরা array তে লুপ use করব string input নেয়ার জন্য। উপরের variable name টা Dx থেকে 1st location টা খুজে বের করে প্রিন্ট করবে। অতঃপর ২য় character খুজে বের করবে অর্থাৎ loop এর কাজ করবে। সব character নেয়া হয়ে গেলে last এ যখন ($) sign পাবে তখন stop করবে।**

❹ .CODE:

**INPUT –OUTPUT OPERATION:**

Register এ যেই character ইনপুট নিব তার always ASCII VALUE টা store হয়।আর output দেখাই ওই ascii value এর জন্য যেই CHARACTER টা আছে সেটা।

আর যদি আমরা যা ইনপুট দিব তাই machine কে নেয়াতে চাই তাহলে ‘ ‘ দিতে হবে। আর যদি ‘ ‘ না দেই তাহলে ascii value দেখাবে।

এখন আমরা কিছু function দেখব :

* AH,1 মানে হল input নেয়া হবে এই command দেয়া।

CODE: MOV AH,1

* AH,2 মানে হল output দেখানো হবে এই command দেয়া।

CODE : MOV AH,2

* AH ,9 মানে হল যখন string input নিবে সেই command নেয়া।

CODE : MOV AH,9

এখানে 1,2,9 হল function number.

MOV AH,1-read character function

MOV AH,2-display character function

MOV AH,9-display string function

INPUT নিয়ে কিছু কথাঃ

কোন কিছু ইনপুট নিয়ে যখন interrupt টা কল করা হয় তখন ওইটার ascii value টা **automatic**  AL নামক register এ store হয়। অর্থাৎ

MOV AH,1 - input key function

INT 21H - ASCII code in AL

OUTPUT নিয়ে কিছু কথা :

OUTPUT হিসেবে কিছু দেখতে চাইলে তা DL এ রাখতে হয়।DL এর বিশিষ্টও হল এতে যাই রাখা হয় তাই print করে দেখায়। এতে ভালু না রেখে output দেখানো যাবেনা। তাই c/c++ এর মত printf / cout না লিখে just value টা DL register এ রেখে দাও কাজ হয়ে যাবে। :D তারপর কমা তারপর যা output হিসেবে দেখাতে চাই তা single quotation এর মধ্যে রাখতে হবে।

Example : DL,’B’ ;

কোন কিছু print করাতে চাইলে নিম্নোক্ত লাইন লিখতে হবে :

MOV DL , যা প্রিন্ট করতে চাই তা (constant/variable/register)

যা প্রিন্ট করতে চাই/ input নিতে চাই তা লিখার পর একটা interrupt call করতে হয়। যার syntax হল :

INT 21H

এখানে INT মানে হল interrupt. Interrupt এর শাব্দিক অর্থ হল বাধা দেয়া। CPU যেই কাজ করতেসিল সেটা stop করে user যেই টাস্ক টা দেয় সেটা করাই এক্ষেত্রে interrupt. অতঃপর কাজ শেষ হয়ে গেলে নিজে যা করতেসিল সেটা করে। interrupt হল একটা subroutine . subroutine হল ছোটো ছোটো CODE.subroutine গুলর identifying number থাকে। 21H হল তাই। এক্ষেত্রে 21 মানে হল ২১ নম্বর interrupt call করা হয়েছে। আর H মানে হল hexadecimal. INT 21H দিয়ে input/output command execute করা হয়।

**যতবার ইনপুট নিবা যতবার output নিবা ততবার interrupt call করতে হবে।**

Example

**MOV AH,2**

**MOV DL,43H**

**INT 21H**

MOV AH,2 এই line এর মাধ্যমে screen এর সাথে communication হল যে output দেখাতে হবে।

MOV DL,43H এর মানে হল DL নামক register এ ৪৩ এর hexadecimal value store করে রাখবে।

INT 21H কে call করলে final কাজ টা complete করে. অর্থাৎ output / input দেখাই / নেয়।

**একটা common প্রশ্ন :Input নেয়ার ক্ষেত্রে MOV AH,1 লিখার পরই interrupt call করা হল। যেটা logical. But Output দেখানর ক্ষেত্রে MOV AH,2 লিখার পর কেন interrupt call করা হল না?**



এর কারন হল ইনপুট নেয়ার জন্য ইনপুট screen এর সাথে যেই communication টা করা লাগবে সেটা করানর কাজ টাই interrupt করে থাকে। অর্থাৎ input নেয়ার জন্য যেই prompt টা আসে সেটা আনাই interrupt এর কাজ। আর output এর ক্ষেত্রে output দেখানই interrupt এর কাজ। তাই কি output দেখাতে হবে সেটাই যদি বলে না দেই তাহলে কি output দেখাবে ! তাই

Printing null/space :

MOV DL, ‘ ‘

MOV DL ,’0 ‘

Printing new line:

ENTER/NEW LINE= CARRIAGE RETURN+LINE FEED

Enter button এর hexa ascii value 2 টা হল

CR->0DH (CR=Carriage retun)

LF - > 0AH (line feed)

Here is the code :



Enter button এর decimal ascii value 2ta হল

CR-> 13

LF->10

Here is the code :



**THE EMULATOR IS HALTED:**

Halted মানে হল paused হয়ে থাকা। operating system থেকে control নিয়ে নেয়। কন্ট্রোল অপারেটিং সিস্টেম কে দেয় না। কিন্তু নিম্নোক্ত দুই লাইন লিখলে আর এম্ন হয় না ।

MOV AH,4(CH/L) (অর্থাৎ CH/L এর যেকোনো ১ টা use করা যাবে। )

INT 21H

**ASCII VALUE :**

**JUMP**

Syntax :

JMP (লেভেল এর নাম as operand)

লেভেল এর নাম use করার উপায় :

LEVEL এর নাম টা লিখলে ওই নাম টা খুজে exact ওই যায়গায় jump করবে।

ENTER কে STRING আকারে লিখার উপায়। সো এখন আর আলাদা ভাবে line feed + carriage return লিখার দরকার পরতেসেনা।

N1 db odh,oah,’$’

**CMP (compare)**

Compare er sysntax টা হল ঃ

Cmp register (variable/register/contant)

Common সমস্যা :

Aa

**STACK :**

Best book এর page – 33 dekho.

Stack code :

.MODEL SMALL

.STACK 100H

.DATA

var1 db '9'

var2 dw 'A'

var3 dw ?

.CODE

MAIN PROC

MOV AH,9

PUSH AX

MOV BH,'5'

PUSH BX

;SUB SP,2

MOV CX,'3'

PUSH CX

MOV DX,5

PUSH DX

MOV BX,'W'

PUSH BX

;PUSH var1 ;it wont work because of db (1 byte)

PUSH VAR2 ; it will work because of dw (2 byte)

POP AX

POP BX

POP DX

INT 21H

MAIN ENDP

END MAIN

PROCEDURE:

The main purpose of CALL is to transfer the control to a procedure. The CALL instruction differs from the jump instruction because a CALL saves a return address on the stack. CALL নিজেও একটা procedure. Call এর সাথে যেই RET থাকে শেতা অন্য CALL এর সাথে যেই RET থাকে সেটা operating system কে control return করে। আর procedure এর সাথে যেই RET থাকে সেটা

BASIC CODE FOR PROCEDURE:

.MODEL SMALL

.STACK 100H

.DATA

CODE TO PRACTISE STEP BY STEP :

1.Basic single character input output:

.MODEL SMALL

.STACK 100H

.DATA

.CODE

MOV AH,1

INT 21H ;AUTOMATIC stores the value in AL

MOV AH,2

MOV DL,AL ;AL er valueta output dekhanor jonno DX e rakhlo.

INT 21H

MAIN PROC

MAIN ENDP

২। addition subtraction :