Assembly Language Programming

Tuesday, June 9, 2015

8086 Assembly Language Programming

Assembly Language Programming is a low level programming language which is processor specific. It means it will run only on the processor architecture for which it was written.

Pros:

1. ***Faster***- Basically assembly language program are executed in much less time as compared to the high-level programing language like c,c+.
2. ***Low memory usage*** - As assembly is processor specific it consumes less memory and are compiled in low memory space.
3. ***Real Time Systems*** - Real time applications use assembly because they have a deadline for their output. (i.e system should response or generate output within a specific period of time.)

Cons:

1. ***Portability***- Assembly language is processor specific so it cannot run on multiple platforms. It is machine specific language.
2. ***Difficult to program***- The programmer should have a keen knowledge about the architecture of the processor as different processors will have different register set and different combinations to use them.
3. ***Debugging***- Debugging becomes very difficult for assembly language if program has some error.

So why to use Assembly Language Programming?

If you are programming for a specific processor or for real time applications assembly language programming can be more useful to you in terms of processing speed, performance and in low memory systems.

Where to write the Code?

The code can be written in Notepad and saved with an extension of **asm**. i.e **Filename.asm**
This file can be made to run on various assembler packages like TASM, MASM etc.

There are also different Emulators (a software which simulates a hardware) available for various processors for compiling and running the code.

I will be using TASM to run few of my codes written for 8086 processor.

Things to know before writing an Assembly Language Program (ALP)

About Me

Rahul Setpal

Follow 39

View my complete profile

Blog Archive

▼ 2015 (1)
▼ June (1)
8086 Assembly Language Programming

Assembler Directives or Pseudo Codes

These are the ***Statements or Instructions that Direct the assembler*** to perform a task.

The inform the processor about the start/end of segment, procedure or program and reserve a appropriate space for data storage etc.

1. Basic Assembler Directives(Pseudo Codes) Used in Programming

1) ASSUME

Assume CS: CODE, DS: DATA

It is used to inform the complier that **CS** (**CODE SEGMENT**) contains the **CODE** and **DS** (**DATA SEGMENT**) contains **DATA**
*****The above Directive can also be written as:
(***Not Recommended as STD. Coding***)

Assume CS: DATA, DS: CODE
Here CODE is written in DATA SEGMENT and DATA in CODE SEGMENT

2) DUP()

Declaring an array with garbage

Eg. A DB 04H DUP (?)
A = Variable
DB = Data Type
04H = Length of Array
? = Element to be DUPLICATED (DUP)

Declaring an array with Same value

Eg. A DB 04H DUP (33H)

Defines the array with variable name A of length 04H having values 33H
FOUR locations of array are having value 33H

Declaring an array with Different Elements

Eg. 1) A DB 03H, 04H, 05H
Eg. 2) A DB 'R','A','H','U','L'

3) START

It indicates the start of Program.

4) END

It indicates end of Program.

5) ENDS

Indicates End of **S**egment.

6) PROC

Used to indicate the beginning of Procedure.

7) ENDP

Used to indicate the end of Procedure.

8) EQU

EQU (Equates) it is used for declaring variables having constants

values.

Eg. A EQU 13H
Variable A is a constant having value 13H

2. SOFTWARE INTERRUPTS

1) INT 03H

INT 03H (3) Breakpoint

INT 3 is the breakpoint interrupt.

Debuggers use this interrupt to establish breakpoints in a program that is being debugged. This is normally done by substituting an **INT 3** instruction, which is one byte long, for a byte in the actual program. The original byte from the program is restored by the debugger after it receives control through **INT 3**

2) KEYBOARD INTERRUPTS

Taking Input from USER

i) MOV AH,0AH
INT 21H

Keeps on taking input from user until terminated by '**$**'.
The input is taken in reg. AL

ii) MOV AH,01H
INT 21H

Takes only one character from user.
The input is taken in reg. AL

Display Messages

i) MOV AH,09H
INT 21H

Displays a message terminated by '**$**'.
The Characters are taken in DX reg. (for word) or DL reg. (for byte) and Displayed.

ii) MOV AH,02H
INT 21H

Displays only single Character whose **ASCII** value is in DL reg.

3) INT 10H

INT 10h / AH = 0 - set **video mode**.

Input:
AL = desired video mode.

These video modes are supported:
00h - text mode. 40x25. 16 colors. 8 pages
03h - text mode. 80x25. 16 colors. 8 pages
13h - graphical mode. 40x25. 256 colors. 320x200 pixels. 1 page.

Example:
```
MOV AL, 13H
MOV AH, 0
INT 10H
```

***NOTE:** This Interrupt is used for clearing the **DOS** screen.

3. Macros and Procedure

1) MACRO

Definition of the macro

A macro is a group of repetitive instructions in a program which are coded only once and can be used as many times as necessary.

The main difference between a macro and a procedure is that in the macro the passage of parameters is possible and in the procedure it is not, this is only applicable for the TASM - there are other programming languages which do allow it. At the moment the macro is executed each parameter is substituted by the name or value specified at the
time of the call.

Syntax of a Macro

The parts which make a macro are:

i) Declaration of the macro**.**
ii) Code of the macro
iii) Macro termination directive

The declaration of the macro is done the following way:
NameMacro MACRO [parameter1, parameter2...]

Eg. **To Display a message**

```
DSPLY  MACRO MSG
  MOV AH,09H
  LEA DX,MSG
  INT 21H
ENDM
```

To use a macro it is only necessary to call it by its name, as if it were another assembler instruction, since directives are no longer necessary as in the case of the procedures.

Example:

DSPLY MSG1

2) PROC

Procedure
Definition of procedure

A procedure is a collection of instructions to which we can direct the flow of our program, and once the execution of these instructions is over control is given back to the next line to process of the code which called on the procedure.

At the time of invoking a procedure the address of the next instruction of the program is kept on the stack so that, once the flow of the program has been transferred and the procedure is done, one can return to the next line. of the original program, the one which

called the procedure.

Syntax of a Procedure

There are two types of procedures, the **INTRA-SEGMENTS**, which are found on the same segment of instructions, and the **INTER-SEGMENTS** which can be stored on different memory segments.

When the intra-segment procedures are used, the value of IP is stored on the stack and when the intra-segments are used the value of CS:IP is stored.

The part which make a procedure are:

i) Declaration of the procedure
ii) Code of the procedure
iii) Return directive
iv) Termination of the procedure

Eg.
```
ADD PROC NEAR
MOV AX,30H
MOV BX,30H
ADD AX,BX
RET
ADD ENDP
```

To divert the flow of a procedure (calling it), the following directive is used:
CALL Name of the Procedure, Example
CALL ADD

************************NOTE*********************

The **LEA** Instruction
LOAD EFFECTIVE (OFFSET) **ADDRESS**

```
LEA SI, A        ; Loads effective address of A in
                 ; SI reg.
```

The above instruction can also be written as
MOV SI, OFFSET A

Eg. **A DB 01H,20H,30H,40H,50H**

To load the effective address of 50H in SI:
LEA SI, A+04H
This is because by Default **LEA SI,A** points at location **01H** to make it point at location **50H** we add **+04H**

To Initialize the address of **DATA SEGMENT** and **EXTRA SEGMENT** in **DS** and **ES** respectively

Getting address **of DATA SEGMENT:**

```
MOV AX,DATA
MOV DS,AX
```

*****Similarly it can be done for extra segment.**

Why can't we write MOV DS,DATA?

DS is a **SEGMENT REGISTER**. In 8086 only registers that can give the value to **SEGMENT REGISTERS** are the **GENERAL PURPOSE REGISTERS.**
i.e. registers **AX,BX,CX,DX**

**********IMP**********

CODE SEGMENT can never initialize by a programmer.
It is automatically initialized by assembler.

How to use TASM ?

Download TASM.
you can use the following link to download.

https://drive.google.com/file/d/0B2UREG3dWedjVU4tZ1RlQ3ltM0k/view?usp=sharing

Compile and run a code in TASM

1) Save the file in **C: \Tasm\Bin**

2) Open **command prompt**.

3) Change the path to that of installation to **\tasm\bin**

if your installation directory is c then type this

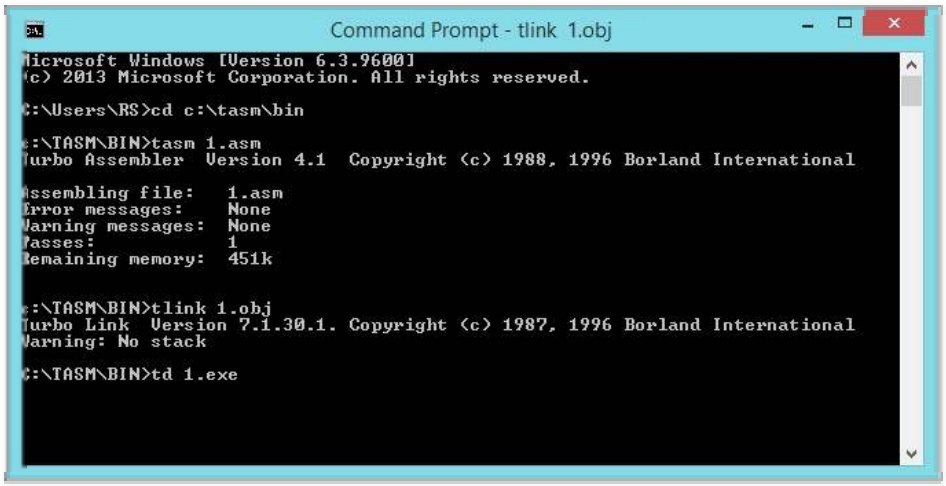
cd c:\tasm\bin

4) Checking for errors- type this
tasm filename.asm

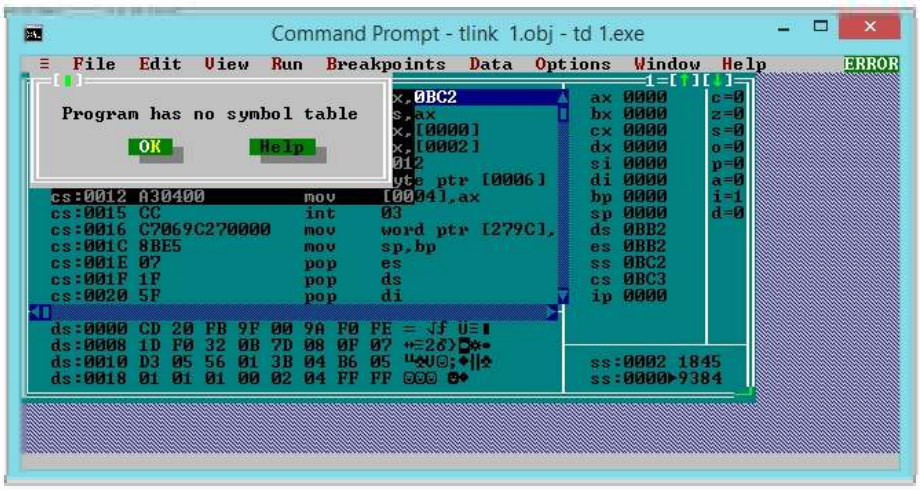
Here my filename is 1

5) Create a object file - type this
tlink filename.obj

6) Now creating the .exe file of your code -type
td 1.exe



Now press **"Enter"**



you will be returned to above screen with the message **"Program has no symbol table"**
click ok.

7) Run the code

go to MENU->Run -> Run

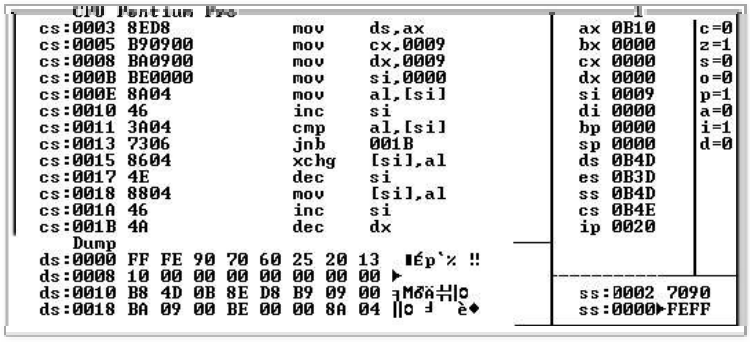
or press F9

to view the Dump **goto**

MENU ->View -> Dump

Dump contains your Stored data.

Now let us move towards programming



NOTE: Assembly language is not case sensitive.

I'll be covering few programs on **8086** processor

List of Programs

1) Addition of two 16-bit nos
2) Adding two 16-bit BCD nos
3) To sort the nos. in ascending order
4) To sort the nos. in descending order
5) To find largest of 10 nos
6) To find smallest of 10 nos
7) To find the no of even & odd nos. from series of 10 nos
8) To find the no. of positive,negative & zeros from series of 10 nos
9) To take String from user find its length and reverse the string
10) To take a string from user & find its length (using Macro and Procedure)
11) Palindrome (single word)------Programmer Defined Input/ Input by programmer
12) Palindrome (single word)-------User Defined Input/ Input by User
13) Palindrome (palindrome string/sentence) ---User Defined Input (using Macro and Procedure)
14) Palindrome (palindrome string/sentence) ---User Defined Input (without using Macro and Procedure)
15) Multiplication of 32 bit nos
16) 3x3 Matrix Multiplication

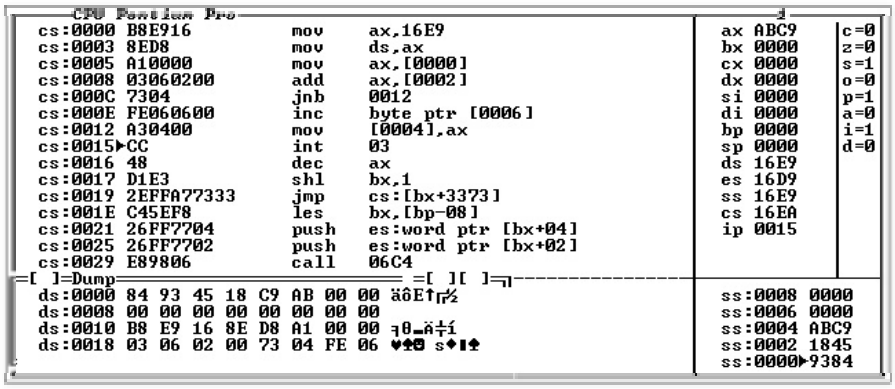
PROGRAMS

1) Addition of two 16-bit nos

Program:

```
ASSUME CS: CODE, DS: DATA

DATA SEGMENT
   A DW 9384H
   B DW 1845H
   SUM DW ?
   CARRY DB 00H
DATA ENDS

CODE SEGMENT
START:   MOV AX, DATA
   MOV DS, AX

   MOV AX, A
   ADD AX, B
   JNC SKIP
   INC CARRY
   SKIP:   MOV SUM, AX
   INT 03H
CODE  ENDS
END START
```

Output:

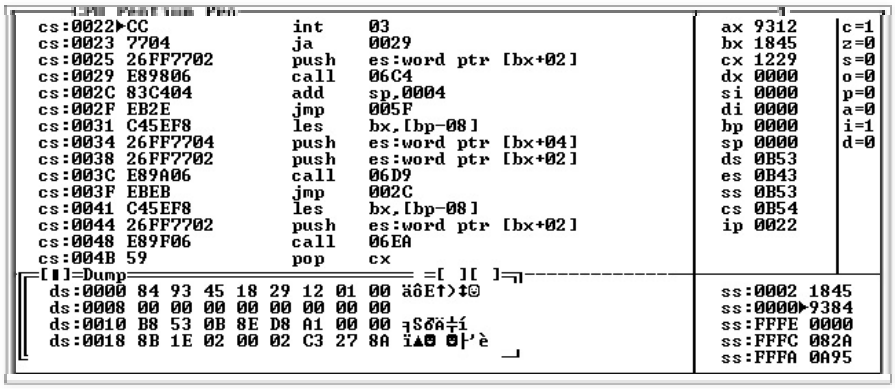


2) Adding two 16-bit BCD nos

Program:

```
ASSUME CS: CODE, DS: DATA
DATA SEGMENT
   A DW 9384H
   B DW 1845H
   SUM DW ?
   CARRY DB 00H
DATA ENDS
CODE SEGMENT
START:   MOV AX, DATA
   MOV DS, AX
   MOV AX, A
   MOV BX, B
   ADD AL, BL
   DAA
   MOV CL, AL
   MOV AL, AH
   ADC AL, BH
   DAA
   MOV CH, AL
   JNC SKIP
   INC CARRY
SKIP:   MOV SUM, CX
   INT 03H
CODE   ENDS
END START
```

Output:

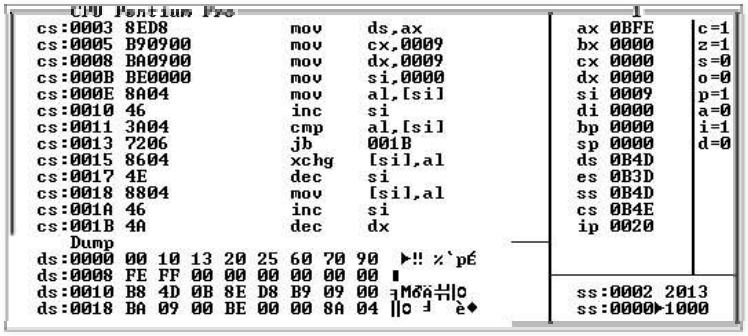


3) To sort the nos. in ascending order

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
  A DB 0FFH,70H,90H,60H,0FEH,20H,10H,13H,25H,00H
DATA ENDS
CODE SEGMENT
   START :MOV AX,DATA
     MOV DS,AX
     MOV CX,0009H
  BACK:   MOV DX,0009H
     LEA SI,A
  BACK1:   MOV AL,[SI]
     INC SI
     CMP AL,[SI]
     JC SKIP
     XCHG AL,[SI]
     DEC SI
     MOV [SI],AL
     INC SI
  SKIP:   DEC DX
     JNZ BACK1
     LOOP BACK
     INT 03H
CODE ENDS
END START
```

Output:



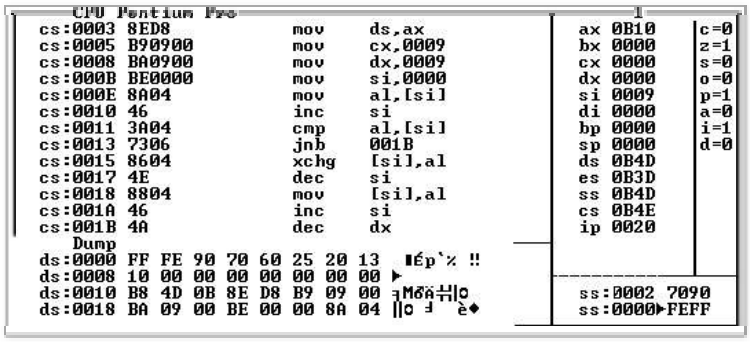
4) To sort the nos. in descending order

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
  A DB 0FFH,70H,90H,60H,0FEH,20H,10H,13H,25H,00H
DATA ENDS
CODE SEGMENT
START :MOV AX,DATA
  MOV DS,AX
  MOV CX,0009H
BACK:   MOV DX,0009H
  LEA SI,A
BACK1:   MOV AL,[SI]
  INC SI
  CMP AL,[SI]
  JNC SKIP
  XCHG AL,[SI]
```

```
   DEC SI
   MOV [SI],AL
   INC SI
SKIP:   DEC DX
   JNZ BACK1
   LOOP BACK
   INT 03H
CODE ENDS
END START
```

Output:



5) To find largest of 10 nos

Program:

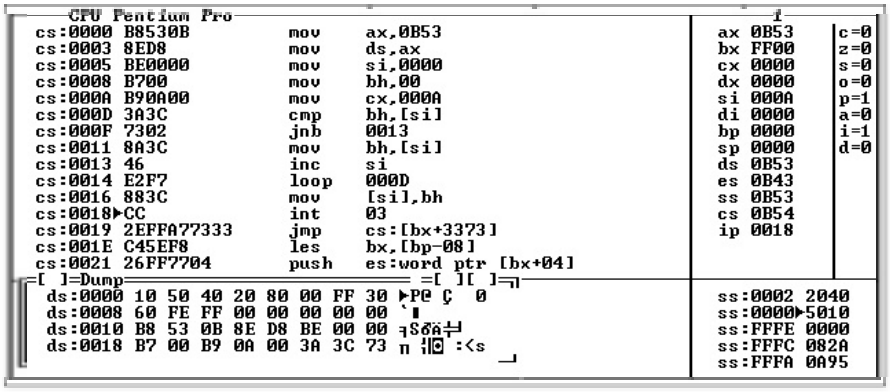
```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
   A DB 10H,50H,40H,20H,80H,00H,00FFH,30H,60H,00FEH
DATA ENDS
CODE SEGMENT
   START:   MOV AX,DATA
      MOV DS,AX

      LEA SI,A
      MOV BH,00H

      MOV CX,000AH
   BACK:   CMP BH,[SI]
      JNC SKIP
      MOV BH,[SI]
   SKIP:   INC SI
      LOOP BACK
      MOV [SI],BH
      INT 03H
CODE ENDS
END START
```

Output:

6) To find smallest of 10 nos

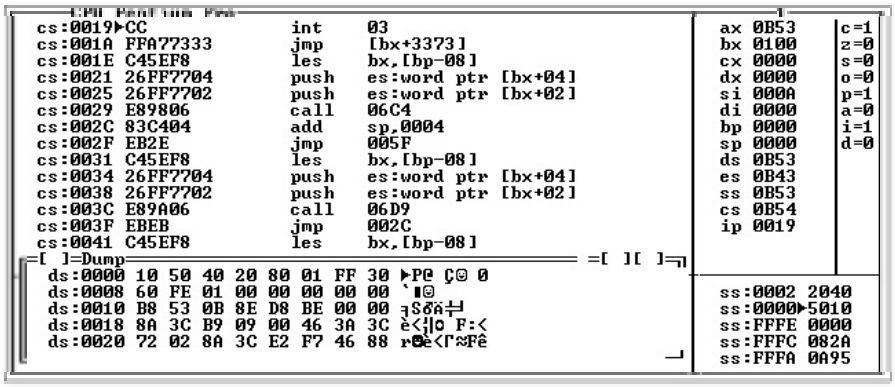
Program:

```
ASSUME CS:CODE,DS:DATA
```




```
DATA                                                    SEGMENT

   A DB 10H,50H,40H,20H,80H,01H,00FFH,30H,60H,00FEH
DATA ENDS
CODE SEGMENT
   START:    MOV AX,DATA
      MOV DS,AX

      LEA SI,A
      MOV BH,[SI]

      MOV CX,0009H
   BACK:    INC SI
      CMP BH,[SI]
      JC SKIP
      MOV BH,[SI]
   SKIP:    LOOP BACK
      INC SI
      MOV [SI],BH
      INT 03H
CODE ENDS
END START
```

Output:



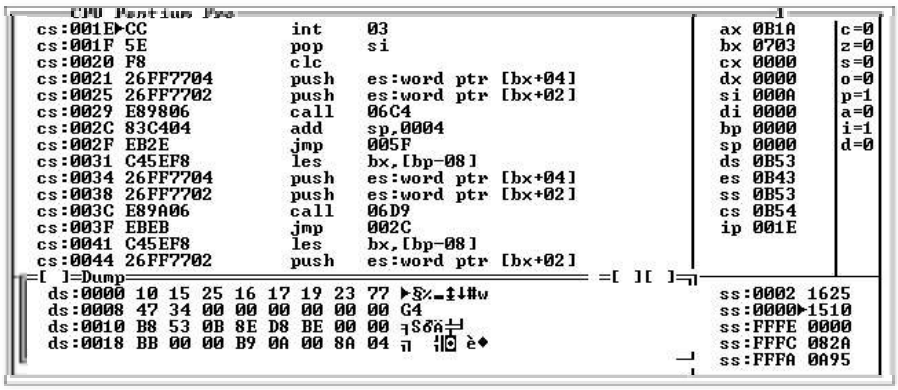
7) To find the no of even & odd nos. from series of 10 nos

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
   A DB 10H,15H,25H,16H,17H,19H,23H,77H,47H,34H
DATA ENDS
CODE SEGMENT
   START:    MOV AX,DATA
      MOV DS,AX

      LEA SI,A
      MOV BX,0000H
      MOV CX,000AH
   BACK:    MOV AL,[SI]
      ROR AL,1
```

```
    JC ODD
    INC BL
    JMP NEXT
  ODD:  INC BH
  NEXT:  INC SI
    LOOP BACK
    INT 03H
CODE ENDS
END START
```

Output:

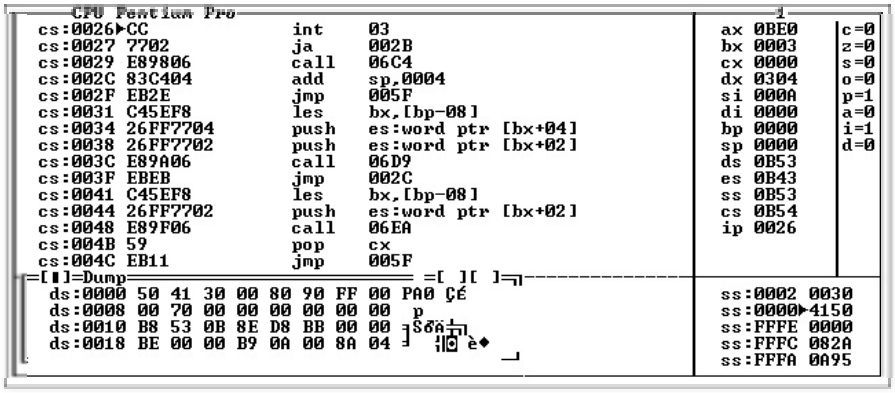


8) To find the no. of positive,negative & zeros from series of 10 nos

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
  A DB 50H,41H,30H,00H,80H,90H,00FFH,00H,00H,70H
DATA ENDS
CODE SEGMENT
  START:  MOV AX,DATA
    MOV DS,AX

    MOV BX,0000H
    LEA SI,A
    MOV CX,000AH
  BACK:  MOV AL,[SI]
    CMP AL,00H
    JZ ZERO
    ROL AL,1
    JC NEGAT
    INC DL
    JMP SKIP
  ZERO:  INC BX
    JMP SKIP
  NEGAT:  INC DH
  SKIP:  INC SI
    LOOP BACK
    INT 03H

CODE ENDS
END START
```

Output:



9) To take String from user find its length and reverse the string

Program:

```
ASSUME DS:DATA,CS:CODE
DATA SEGMENT
  CR EQU 13D          ; EQU defines constant, CR and LF are constants
  LF EQU 10D          ; CARRIAGE RETURN and LINE FEED initialize with
            ; ASCII VALUES
  ER DB CR,LF,'NO STRING ENTERED PRESS ANY KEY TO EXIT.....$'
  LEN DB CR,LF,'THE LENGTH OF STRING IS->$'
  REV DB CR,LF,'REVERSE OF YOUR STRING->$'
  INPUT DB 'ENTER A STRING->$'
  TEMP DB 00FFH DUP (?)
DATA ENDS
CODE SEGMENT
  START:   MOV AX,DATA    ; Initialize DATA SEGMENT
       MOV DS,AX

       MOV AL,03H          ; CLEAR the DOS SCREEN
       MOV AH,0
       INT 10H

       MOV CX,0000H       ; CLEAR the COUNT reg.

       MOV DX,OFFSET INPUT    ; Print the INPUT message
       MOV AH,09H
       INT 21H

       LEA DI,TEMP        ; CHECKING whether STRING is
       MOV AH,01H       ; PROVIDED
       MOV [DI],AL
       INC CX
       INC DI
       INT 21H
       CMP AL,13D
       JE EXIT

  BACK:   MOV AH,01H       ; KEEP ON taking CHARACTERS
       MOV [DI],AL          ; until press ENTER
       INT 21H
       INC DI
       INC CX
       CMP AL,13D
       JNZ BACK

MOV AH,09H       ; Print the LEN message
       LEA DX,LEN
       INT 21H

       DEC CL
       CMP CL,64H          ; CHECK for STRING LENGTH  greater
            ; than 100D (64H)
PUSHF          ; CLEAR the OVERFLOW flag
       POP BX
       AND BH,00F7H
```

```
        PUSH BX
        POPF

JGE PRINT1
        MOV BX,CX
        CMP CL,0AH              ; CHECK for STRING LENGTH  greater
        JGE SKIP        ; than 10D (0AH)
        MOV BX,CX
        ADD BL,30H
        MOV AH,02H      ; PRINT the LENGTH for SINGLE
        MOV DL,BL       ; DIGIT (FROM 1-9)
        INT 21H
        JMP SKIP1

        PRINT1:MOV AH,02H   ; PRINT 1 as MSB when length is greater
                ; than 99D
        MOV DL,31H
        INT 21H

  SKIP:     MOV BL,CL       ; CONVERT the COUNT in BCD format
                ; for 2-DIGIT
        MOV AL,00H      ; COUNT
  BACK0:  ADD AL,01H
        DAA
        DEC BL
        JNZ BACK0
        MOV BL,AL
        ROL AL,01H      ; MASK the LOWER NIBBLE & PRINT
        ROL AL,01H
        ROL AL,01H
        ROL AL,01H
        AND AL,0FH
        ADD AL,30H
        MOV AH,02H
        MOV DL,AL
        INT 21H
AND BL,0FH      ; MASK the UPPER NIBBLE & PRINT
        ADD BL,30H
        MOV AH,02H
        MOV DL,BL
        INT 21H

  SKIP1:    MOV AH,09H      ; Print the REV message
        MOV DX,OFFSET REV
        INT 21H

        MOV DI,OFFSET TEMP   ;  Print the REVERSE STRING
        MOV BX,CX
        MOV AH,02H
  BACK1:  MOV DL,[BX+DI]
        INT 21H
        DEC BX
        JNZ BACK1
        JMP LAST

  EXIT:     MOV AH,09H      ; PRINT the ERROR message
                ; when no string is given
        LEA DX,ER
        INT 21H

  LAST:     MOV AH,01H      ; HOLD the O/P SCREEN
        INT 21H

        INT 03H
CODE ENDS
END START
```

Output:

(This program can give a maximum count of **C7H** i.e **199D**)

```
ENTER A STRING->zZyYxXwWvVuUtTsSrRqQpPoOnNmMlLkKjJiIhHgGfFeEdDcCbBaAABCDEFGHIJKL
MnopqrstuvwxyzzyxwvutsrqponMLKJIHGFEDCBA
THE LENGTH OF STRING IS->104
REVERSE OF YOUR STRING->ABCDEFGHIJKLMnopqrstuvwxyzzyxwvutsrqponMLKJIHGFEDCBAAaBb
CcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
```

10) To take a string from user & find its length (using Macro and Procedure)

Program:

```
ASSUME CS:CODE , DS:DATA
DATA  SEGMENT
   CR EQU 0DH
   LF EQU 0AH
   LEN DB 04 DUP(0)
   MSG1 DB CR,LF,'ENTER THE STRING=','$'
   MSG2 DB CR,LF,'THE LENGTH OF STRING=','$'
DATA ENDS

DISP MACRO MSG
MOV AH,09H
MOV DX,OFFSET MSG
INT 21H
ENDM

CODE SEGMENT
   START:    MOV AX,DATA
      MOV DS,AX
      DISP MSG1
      MOV CX,00H
      READ:    MOV AH,01H
      INT 21H
      CMP AL,CR
      JZ AHEAD
      INC CX
      JMP READ


   AHEAD:    DISP MSG2
      MOV AX,CX
      CALL HEX2ASC
      MOV BX,AX
      MOV DL,BH
      MOV AH,02H
      INT 21H
      MOV DL,BL
      MOV AH,02H
      INT 21H
      MOV AH,4CH
      INT 21H

HEX2ASC  PROC NEAR
    MOV BL,01H
   MUL BL
   AAM
   OR AX,3030H
   RET
HEX2ASC ENDP
CODE ENDS
END START
```

Output:

```
C:\tasm\BIN>C.EXE

ENTER THE STRING=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
THE LENGTH OF STRING=36
C:\tasm\BIN>_
```

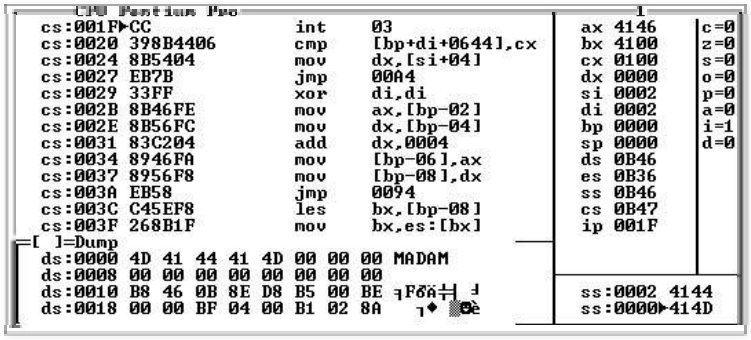
(This program gives a maximum count of **63H** i.e. **99D**)

11) Palindrome (single word)------Programmer Defined Input/ Input by programmer

Program:

```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT
     A DB 'M','A','D','A','M'
DATA ENDS
CODE SEGMENT
  START:   MOV AX,DATA
     MOV DS,AX
     MOV CH,00H

     LEA SI,A
     LEA DI,A+04H
     MOV CL,02H
  BACK:   MOV AH,[SI]
     MOV BH,[DI]
     CMP AH,BH
     JNZ SKIP
     INC SI
     DEC DI
     DEC CL
     JNZ BACK

     INC CH
  SKIP:   INT 03H
CODE ENDS
END START
```

Output:



(After execution CH=01H indicates string is palindrome, CH=00H indicates not a palindrome. Comparison is done Length of string divided by 02H)

12) Palindrome (single word)-------User Defined Input/ Input by User

Program:

```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT
   A DB 13D,10D,'THE GIVEN STRING IS PALINDROME $'
   B DB 13D,10D,'THE GIVEN STRING IS NOT PALINDROME $'
   C DB 'ENTER THE STRING- $'
   TEMP DB 00FFH DUP(?)
DATA ENDS

CODE SEGMENT
START:    MOV AX,DATA
   MOV DS,AX

   MOV AL,03H          ; CLEAR THE DOS SCREEN
   MOV AH,0
   INT 10H

   MOV AH,09H
   LEA DX,C
   INT 21H

   MOV CX,0000H          ; CLEAR THE COUNTER
   LEA SI,TEMP
BACK:    MOV AH,01H          ; TAKE STRING FROM USER AND SAVE IT IN "TEMP"
   MOV [SI],AL
   INT 21H
   INC SI
   INC CX
   CMP AL,13D
   JNZ BACK
   DEC CX

   MOV DX,CX
   MOV AX,CX          ; MOVE COUNT IN AX
   MOV BL,02H
   DIV BL             ; COMPARISION SHOULD BE DONE HALF THE NO. OF CHARACTERS
   MOV CL,AL

   LEA SI,TEMP          ; SETTING THE POINTER SI TO FIRST CHARACTER OF STRING
   INC SI
   LEA DI,TEMP
   ADD DI,DX          ; SETTING THE POINTER DI TO LAST CHARACTER OF STRING

BACK1:    MOV AL,[SI]          ; MOVING THE CHARACTER POINTED BY SI IN AL
   MOV BL,[DI]          ; MOVING THE CHARACTER POINTED BY DI IN BL
   INC SI
   DEC DI
   CMP AL,BL          ; COMPARING AL AND BL
   JNZ SKIP
   DEC CL
   JNZ BACK1
   JMP SKIP2

SKIP:     MOV AH,09H
   LEA DX,B
   INT 21H
   JMP EXIT

SKIP2:     MOV AH,09H
   LEA DX,A
   INT 21H

EXIT:    MOV AH,01H          ; HOLDING THE OUTPUT SCREEN
   INT 21H             ; GIVE ANY KEYBOARD INTERRUPT TO EXIT
   INT 03H

CODE ENDS
END START
```

Output:

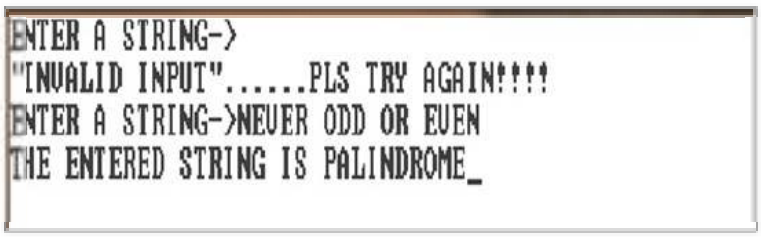
13) Palindrome (palindrome string/sentence) ---User Defined Input (using Macro and Procedure)

Program:

```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT
  ER DB 13D,10D,'"INVALID INPUT".....PLS TRY AGAIN!!!! $'
  A DB 13D,10D, 'THE ENTERED STRING IS PALINDROME$'
  B DB 13D,10D,'THE ENTERED STRING IS NOT A PALINDROME$'
  INPUT DB 'ENTER A STRING->$'
  TEMP DB 00FFH DUP (?)
DATA ENDS

DSPLY   MACRO MSG          ; MACRO function for DISPLAY
     MOV AH,09H
     LEA DX,MSG
     INT 21H
ENDM

CODE SEGMENT
  START:   MOV AX,DATA
       MOV DS,AX

       MOV AL,03H          ; CLEAR the DOS Screen
       MOV AH,0
       INT 10H

  STRT:      MOV CX,0000H
       DSPLY INPUT       ; PRINT INPUT msg

       LEA SI,TEMP
       MOV AH,01H
       MOV [SI],AL
       INT 21H
       INC CX
       INC SI
       CMP AL,13D       ; CHECK whether STRING PROVIDED
       JNE BACK
       DSPLY ER       ; PRINT ERROR msg on SCREEN

       MOV AH,02H       ; LINE FEED and CARRIAGE RETURN
       MOV DL,13D
       INT 21H
       MOV AH,02H
       MOV DL,10D
       INT 21H
       JMP STRT

  BACK:      MOV AH,01H       ; TAKE INPUT from user and STORE
       MOV [SI],AL
       INT 21H
       INC SI
       INC CX
       CMP AL,13D
       JNZ BACK

       DEC CX
       MOV BX,CX
       CALL COUNT          ; CALL sub-routine to CALCULATE NO. of
       LEA SI,TEMP         ; COMPARISION
       INC SI
```

```
        LEA DI,TEMP
        ADD DI,BX

  BACK1:    MOV AH,[SI]
        MOV DH,[DI]
        CMP AH,20H          ; CHECK IF SPACE
        JE PLUS

  BAAK:     INC SI
        CMP DH,20H
        JE PLUSS
  BAKK:     DEC DI
        CMP AH,DH
        JNZ SKIP
        DEC CL
        JNZ BACK1
        JMP LAST

  PLUS:     INC SI
        MOV AH,[SI]
        JMP BAAK
  PLUSS:    DEC DI
        MOV DH,[DI]
        JMP BAKK

  LAST:     DSPLY A
        JMP EXIT
  SKIP:     DSPLY B

  EXIT:     MOV AH,01H
        INT 21H
        INT 03H

COUNT   PROC NEAR       ; CALCULATE NO. OF COMPARISION
        MOV AX,CX
        MOV CL,02H
        DIV CL
        MOV CL,AL
        RET
  COUNT ENDP

CODE ENDS
END START
```

Output:



****

If enter is given as first character it will show an error----------

"INVALID INPUT".....PLS TRY AGAIN

And in next line will again ask for Input
****

14) Palindrome (palindrome string/sentence) ---User Defined Input (without using Macro and Procedure)

Program:

```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT
  A DB 13D,10D,'THE GIVEN STRING IS PALINDROME $'
  B DB 13D,10D,'THE GIVEN STRING IS NOT PALINDROME $'
  C DB 'ENTER THE STRING- $'
  TEMP DB 00FFH DUP(?)
DATA ENDS

CODE SEGMENT
START:  MOV AX,DATA
  MOV DS,AX

  MOV AL,03H        ; CLEAR THE DOS SCREEN
  MOV AH,0
  INT 10H

  MOV AH,09H
  LEA DX,C
  INT 21H

  MOV CX,0000H        ; CLEAR THE COUNTER
  LEA SI,TEMP
BACK:   MOV AH,01H          ; TAKE STRING FROM USER AND SAVE IT IN
"TEMP"
  MOV [SI],AL
  INT 21H
  INC SI
  INC CX
  CMP AL,13D
  JNZ BACK
  DEC CX

  MOV DX,CX
  MOV AX,CX        ; MOVE COUNT IN AX
  MOV BL,02H
   DIV BL             ; COMPARISION SHOULD BE DONE HALF THE NO. OF
CHARACTERS
  MOV CL,AL

  LEA SI,TEMP         ; SETTING THE POINTER SI TO FIRST CHARACTER OF
STRING
  INC SI
  LEA DI,TEMP
   ADD DI,DX          ; SETTING THE POINTER DI TO LAST CHARACTER OF
STRING

BACK1:  MOV AL,[SI]         ; MOVING THE CHARACTER POINTED BY SI IN
AL
  MOV BL,[DI]       ; MOVING THE CHARACTER POINTED BY DI IN BL
  CMP AL,20H        ; CHECK FOR "SPACE" AT SI
  JE SKIIP
BAAK:   INC SI
  CMP BL,20H        ; CHECK FOR "SPACE" AT DI
  JE SKIPP

BAKK:   DEC DI
  CMP AL,BL        ; COMPARING AL AND BL
  JNZ SKIP
  DEC CL
  JNZ BACK1
  JMP SKIP2

SKIIP:    INC SI          ; IF "SPACE" AT "SI" THEN INCREMENT SI AND MOVE
ITS CONTENT TO AL
  MOV AL,[SI]
  JMP BAAK

SKIPP:    DEC DI            ; IF "SPACE" AT "DI" THEN DECREMENT DI AND
```

```
MOVE ITS CONTENT TO BL
  MOV BL,[DI]
  JMP BAKK

SKIP:    MOV AH,09H
  LEA DX,B
  INT 21H
  JMP EXIT

SKIP2:    MOV AH,09H
  LEA DX,A
  INT 21H

EXIT:   MOV AH,01H          ; HOLDING THE OUTPUT SCREEN
  INT 21H             ; GIVE ANY KEYBOARD INTERRUPT TO EXIT
  INT 03H

CODE ENDS
END START
```

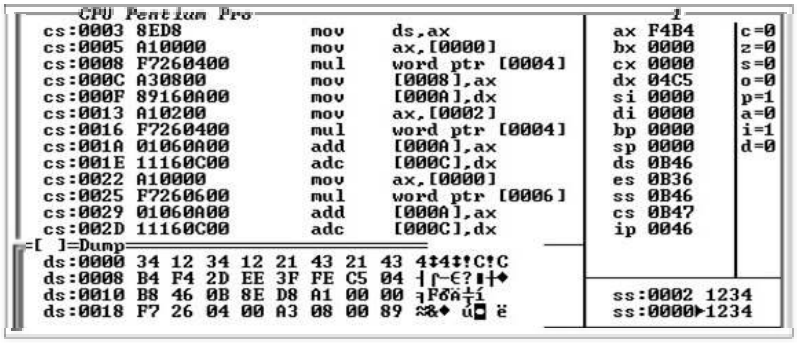
Output:

15) Multiplication of 32 bit nos

Program:

```
ASSUME CS:CODE,DS:DATA
DATA SEGMENT
            MULD DW 1234H, 1234H
            MULR DW 4321H, 4321H
            RES DW 04H DUP(?)
DATA ENDS
CODE SEGMENT
  START:    MOV AX,DATA
            MOV DS,AX

            MOV AX, MULD
                MUL MULR
            MOV RES,AX
              MOV RES+2,DX
                    MOV AX, MULD+2
                    MUL MULR
              ADD RES+2,AX
              ADC RES+4, DX
              MOV AX, MULD
              MUL MULR+2
              ADD RES+2,AX
              ADC RES+4,DX
              JNC SKIP
              INC RES+6
  SKIP:     MOV AX,MULD+2
              MUL MULR+2
              ADD RES+4,AX
              ADC RES+6,DX
              INT 03H
CODE ENDS
END START
```

Output:



16) 3x3 Matrix Multiplication

Note: In this program all entered elements should be single digit and space should be given after each element.

Program:

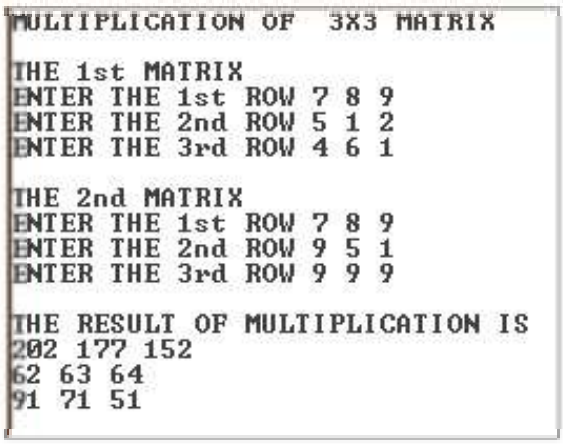
```
ASSUME CS:CODE,DS:DATA

DATA SEGMENT
  A DB 'MULTIPLICATION OF  3X3 MATRIX$'
  B DB 13D,10D,10D,'THE 1st MATRIX$'
  C DB 13D,10D,10D,'THE 2nd MATRIX$'
  D DB 13D,10D,'ENTER THE 1st ROW $'
  E DB 13D,10D,'ENTER THE 2nd ROW $'
  F DB 13D,10D,'ENTER THE 3rd ROW $'
  M1 DB 20H DUP (?)
  M2 DB 20H DUP (?)
  ANS DB 20H DUP(?)
  G DB 13D,10D,10D,'THE RESULT OF MULTIPLICATION IS $'
  I DB 13D,10D,'$'
  K DB 20H,'$'
DATA ENDS

CODE SEGMENT
  DSPLY   MACRO MSG
     MOV AH,09H
     LEA DX,MSG
     INT 21H
     ENDM

  START:   MOV AX,DATA
     MOV DS,AX

     MOV AL,03H
     MOV AH,0
     INT 10H

     DSPLY A

     DSPLY B
     LEA SI,M1
     CALL INPUT

     DSPLY C
     LEA SI,M2
     CALL INPUT

     DSPLY G
     DSPLY I

     LEA SI,M1+01H
     LEA DI,M2+01H
     CALL AD
     DSPLY K
```

```
    LEA SI,M1+01H
    LEA DI,M2+03H
    CALL AD
    DSPLY K

    LEA SI,M1+01H
    LEA DI,M2+05H
    CALL AD

    DSPLY I

    LEA SI,M1+07H
    LEA DI,M2+01H
    CALL AD
    DSPLY K

    LEA SI,M1+07H
    LEA DI,M2+03H
    CALL AD
    DSPLY K

    LEA SI,M1+07H
    LEA DI,M2+05H
    CALL AD

    DSPLY I

    LEA SI,M1+0DH
    LEA DI,M2+01H
    CALL AD
    DSPLY K

    LEA SI,M1+0DH
    LEA DI,M2+03H
    CALL AD
    DSPLY K

    LEA SI,M1+0DH
    LEA DI,M2+05H
    CALL AD

    MOV AH,01H
    INT 21H
    INT 03H


INPUT PROC NEAR
    DSPLY D
BACK0:  MOV AH,01H
    AND AL,0FH
    MOV [SI],AL
    INT 21H
    INC SI
    CMP AL,13D
    JNE BACK0

    DSPLY E
BACK1:  MOV AH,01H
    AND AL,0FH
    MOV [SI],AL
    INT 21H
    INC SI
    CMP AL,13D
    JNE BACK1

    DSPLY F
BACK2:  MOV AH,01H
    AND AL,0FH
    MOV [SI],AL
    INT 21H
    INC SI
```

```
   CMP AL,13D
   JNE BACK2
   RET
INPUT ENDP


AD PROC NEAR
   MOV AX,0000H
   MOV CX,0000H
   MOV DL,0003H

   LEA BX,ANS
BAAK:   MOV AL,[SI]
   MOV CL,[DI]
   MUL CL
   MOV [BX],AX
   ADD SI,02H
   ADD DI,06H
   INC BX
   DEC DL
   JNZ BAAK

   MOV AX,0000H
   LEA SI,ANS
   MOV AL,[SI]
   INC SI
   MOV CL,[SI]
   ADD AL,CL
   INC SI
   MOV CL,[SI]
   ADC AL,CL

   MOV BL,AL
   ROL BL,01H
   JNC SKIP0
   SUB AL,64H
   CMP AL,64H
   PUSHF
   POP BX
   AND BX,00F7H
   PUSH BX
   POPF
   JL SKIIP
   SUB AL,64H
   MOV BL,AL
   MOV AH,02H
   MOV DL,32H
   INT 21H
   JMP SKIP1

SKIP0:   CMP AL,64H
   PUSHF
   POP BX
   AND BH,00F7H
   PUSH BX
   POPF
   JL SKIP
   SUB AL,64H
SKIIP:   MOV BL,AL
   MOV AH,02H
   MOV DL,31H
   INT 21H

SKIP1:   MOV AL,BL
SKIP:   MOV BL,01H
   MUL BL
   AAM
   OR AX,3030H
   MOV BX,AX
   MOV DL,BH
   MOV AH,02H
```

```
        INT 21H
        MOV DL,BL
        MOV AH,02H
        INT 21H
        RET
    AD ENDP
CODE ENDS
END START
```

Output:



That's all about ALP.....

Posted by Rahul Setpal at 4:51 AM

Labels: 8086, ALP, ASM, Assembly language programming

2 comments:

Anupama July 11, 2016 at 3:02 AM

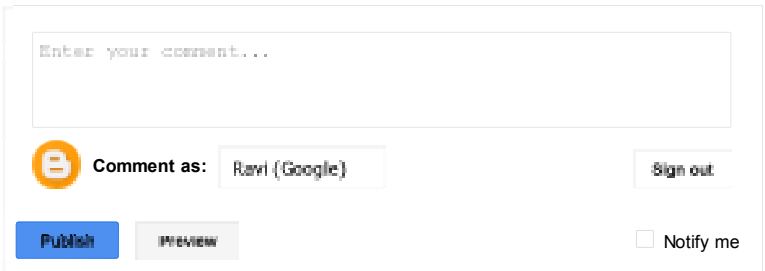
Thanku sir its help me a lot

Reply

Anupama July 11, 2016 at 3:03 AM

Thanku sir its help me a lot

Reply



Home

Subscribe to: Post Comments (Atom)

Simple theme. Powered by Blogger.