

# 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 programming 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.

### 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 Programmin](#)

### Assembler Directives or Pseudo Codes

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

They inform the processor about the start/end of segment, procedure or program and reserve an 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 compiler 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 Segment.

### 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 inter-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/0B2UREG3dWedjVU4tZ1RIQ3ltM0k/view?usp=sharing>

#### Compile and run a code in TASM

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

2) Open command prompt.



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\RS>_
```

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

if your installation directory is c then type this

**cd c:\tasm\bin**



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\RS>cd c:\tasm\bin
C:\TASM\BIN>
```

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

Here my filename is 1



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\RS>cd c:\tasm\bin
e:\TASM\BIN>tasm 1.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file: 1.asm
Error messages:  None
Warning messages: None
Passes:          1
Remaining memory: 451k

e:\TASM\BIN>
```

None means no errors and warnings

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



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.


C:\Users\RS>cd c:\tasm\bin
e:\TASM\BIN>tasm 1.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file: 1.asm
Error messages:  None
Warning messages: None
Passes:          1
Remaining memory: 451k

e:\TASM\BIN>tlink 1.obj
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
Warning: No stack

e:\TASM\BIN>_
```

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



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

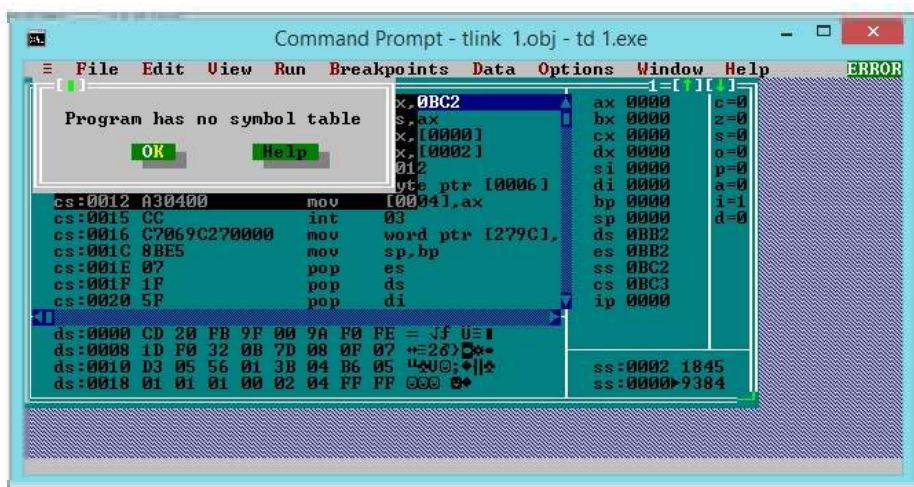
C:\Users\RS>cd c:\tasm\bin
e:\TASM\BIN>tasm 1.asm
Turbo Assembler Version 4.1 Copyright (c) 1988, 1996 Borland International

Assembling file: 1.asm
Error messages:  None
Warning messages: None
Passes:          1
Remaining memory: 451k

e:\TASM\BIN>tlink 1.obj
Turbo Link Version 7.1.30.1. Copyright (c) 1987, 1996 Borland International
Warning: No stack

e:\TASM\BIN>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

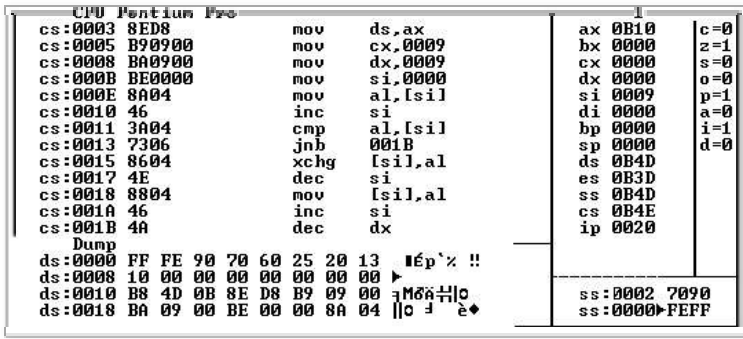


to view the Dump goto

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:**

CS	IP	Instruction	Operation	AX	BX	CX	DX	SI	DI	BP	SP	IP	CS	DS	ES	SS	SI	DI	BP	SP	IP	CS	DS	ES	SS		
cs:0000	B8E916	mov	ax,16E9	ax ABC9	bx 0000	cx 0000	dx 0000	si 0000	di 0000	bp 0000	sp 0000	ip 0015	cs 16E9	ds 16E9	es 16D9	ss 16E9	si 16EA	di 16E9	bp 0000	sp 0000	ip 0015	cs 16E9	ds 16E9	es 16D9	ss 16E9		
cs:0003	8ED8	mov	ds,ax																								
cs:0005	A10000	mov	ax,[0000]																								
cs:0008	03060200	add	ax,[0002]																								
cs:000C	7304	jnb	0012																								
cs:000E	FE060600	inc	byte ptr [0006]																								
cs:0012	A30400	mov	[0004],ax																								
cs:0015	CC	int	03																								
cs:0016	48	dec	ax																								
cs:0017	D1E3	shl	bx,1																								
cs:0019	2EFFA77333	jmp	cs:[bx+3373]																								
cs:001E	C45EF8	les	bx,[bp-08]																								
cs:0021	26FF7704	push	es:word ptr [bx+04]																								
cs:0025	26FF7702	push	es:word ptr [bx+02]																								
cs:0029	E89806	call	06C4																								
[I]=Dump																											
ds:0000	84 93 45 18 C9 AB 00 00	äðE↑↑															ss:0008	0000									
ds:0008	00 00 00 00 00 00 00 00																ss:0006	0000									
ds:0010	B8 E9 16 8E D8 A1 00 00	↑Sðä↑↑															ss:0004	ABC9									
ds:0018	03 06 02 00 73 04 FE 06	↑↑ð s↑↑															ss:0002	1845									
																										ss:0000	9384

**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:**

cs:0022	CC	int	03	ax 9312	bx 1845	cx 1229	dx 0000	si 0000	di 0000	bp 0000	sp 0000	ip 0022	cs 0B53	ds 0B53	es 0B43	ss 0B53	si 0B54	di 0B54	bp 0000	sp 0000	ip 0022	cs 0B54	ds 0B54	es 0B43	ss 0B53		
cs:0023	7704	ja	0029																								
cs:0025	26FF7702	push	es:word ptr [bx+02]																								
cs:0029	E89806	call	06C4																								
cs:002C	83C404	add	sp,0004																								
cs:002F	EB2E	jmp	005F																								
cs:0031	C45EF8	les	bx,[bp-08]																								
cs:0034	26FF7704	push	es:word ptr [bx+04]																								
cs:0038	26FF7702	push	es:word ptr [bx+02]																								
cs:003C	E89A06	call	06D9																								
cs:003F	EBEB	jmp	002C																								
cs:0041	C45EF8	les	bx,[bp-08]																								
cs:0044	26FF7702	push	es:word ptr [bx+02]																								
cs:0048	E89F06	call	06EA																								
cs:004B	59	pop	cx																								
[I]=Dump																											
ds:0000	84 93 45 18 29 12 01 00	äðE↑↑ð															ss:0002	1845									
ds:0008	00 00 00 00 00 00 00 00																ss:0000	9384									
ds:0010	B8 53 0B 8E D8 A1 00 00	↑Sðä↑↑															ss:FFFE	0000									
ds:0018	0B 1E 02 00 02 C3 27 8A	↑äð 0↑'è															ss:FFFC	082A									
																										ss:FFFA	0A95

**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:**

CPU Pentium Pro		I				
cs:0003	8ED8	mov	ds,ax	ax	0BF8	c=1
cs:0005	B90900	mov	cx,0009	bx	0000	z=1
cs:0008	BA0900	mov	dx,0009	cx	0000	s=0
cs:000B	BE0000	mov	si,0000	dx	0000	o=0
cs:000E	8A04	mov	al,[si]	si	0009	p=1
cs:0010	46	inc	si	di	0000	a=0
cs:0011	3A04	cmp	al,[si]	bp	0000	i=1
cs:0013	7206	jb	001B	sp	0000	d=0
cs:0015	8604	xchg	[si],al	ds	0B4D	
cs:0017	4E	dec	si	es	0B3D	
cs:0018	8804	mov	[si],al	ss	0B4D	
cs:001A	46	inc	si	cs	0B4E	
cs:001B	4A	dec	dx	ip	0020	
Dump						
ds:0000	00 10 13 20 25 60 70 90	>!! z'pé				
ds:0008	FE FF 00 00 00 00 00 00					
ds:0010	B8 4D 0B 8E D8 B9 09 00	Mδā+ c				
ds:0018	BA 09 00 BE 00 00 8A 04	c d e				
				ss:0002	2013	
				ss:0000	1000	

**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:**

CPU Pentium Pro						
cs:0003	8ED8	mov	ds,ax	ax	0B10	c=0
cs:0005	B90900	mov	cx,0009	bx	0000	z=1
cs:0008	BA0900	mov	dx,0009	cx	0000	s=0
cs:000B	BE0000	mov	si,0000	dx	0000	o=0
cs:000E	8A04	mov	al,[si]	si	0009	p=1
cs:0010	46	inc	si	di	0000	a=0
cs:0011	3A04	cmp	al,[si]	bp	0000	i=1
cs:0013	7306	jnb	001B	sp	0000	d=0
cs:0015	8604	xchg	[si],al	ds	0B4D	
cs:0017	4E	dec	si	es	0B3D	
cs:0018	8804	mov	[si],al	ss	0B4D	
cs:001A	46	inc	si	cs	0B4E	
cs:001B	4A	dec	dx	ip	0020	
Dump						
ds:0000	FF FE 90 70 60 25 20 13	Ép`x !!				
ds:0008	10 00 00 00 00 00 00 00					
ds:0010	B8 4D 0B 8E D8 B9 09 00	Möñ+ c		ss:0002 7090		
ds:0018	BA 09 00 BE 00 00 8A 04	c d e		ss:0000 FEFF		

**5) To find largest of 10 nos****Program:**

```

A&SUME 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:**

```

A&SUME CS:CODE,DS:DATA

```

CPU Pentium Pro				F		
cs:0000	B8530B	mov	ax,0B53	ax	0B53	c=0
cs:0003	8ED8	mov	ds,ax	bx	FF00	z=0
cs:0005	BE0000	mov	si,0000	cx	0000	s=0
cs:0008	B700	mov	bh,00	dx	0000	o=0
cs:000A	B90A00	mov	cx,000A	si	000A	p=1
cs:000D	3A3C	mov	bh,[sil]	di	0000	a=0
cs:000F	7302	cmp	0013	bp	0000	i=1
cs:0011	8A3C	mov	bh,[sil]	sp	0000	d=0
cs:0013	46	inc	si	ds	0B53	
cs:0014	E2F7	loop	000D	es	0B43	
cs:0016	883C	mov	[sil],bh	ss	0B53	
cs:0018	CC	int	03	cs	0B54	
cs:0019	2EFA77333	jmp	cs:[bx+3373]	ip	0018	
cs:001E	C45EF8	les	bx,[bp-08]			
cs:0021	26FF7704	push	es:word ptr [bx+04]			
[ ]=Dump						
ds:0000	10 50 40 20 80 00 FF 30	PPE C 0		ss:0002	2040	
ds:0008	60 FE FF 00 00 00 00 00	00 00 00 00		ss:0000	5010	
ds:0010	B8 53 0B 8E D8 BE 00 00	750A 1		ss:FFFE	0000	
ds:0018	B7 00 B9 0A 00 3A 3C 73	n 10 :<s		ss:FFFC	082A	
				ss:FFFA	0A95	

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:

CPU Pentium Pro				F		
cs:0019	CC	int	03	ax	0B53	c=1
cs:001A	FFA77333	jmp	[bx+3373]	bx	0100	z=0
cs:001E	C45EF8	les	bx,[bp-08]	cx	0000	s=0
cs:0021	26FF7704	push	es:word ptr [bx+04]	dx	0000	o=0
cs:0025	26FF7702	push	es:word ptr [bx+02]	si	000A	p=1
cs:0029	E89806	call	06C4	di	0000	a=0
cs:002C	83C404	add	sp,0004	bp	0000	i=1
cs:002F	EB2E	jmp	005F	sp	0000	d=0
cs:0031	C45EF8	les	bx,[bp-08]	ds	0B53	
cs:0034	26FF7704	push	es:word ptr [bx+04]	es	0B43	
cs:0038	26FF7702	push	es:word ptr [bx+02]	ss	0B53	
cs:003C	E89A06	call	06D9	cs	0B54	
cs:003F	EBEB	jmp	002C	ip	0019	
cs:0041	C45EF8	les	bx,[bp-08]			
[ ]=Dump						
ds:0000	10 50 40 20 80 01 FF 30	PPE C 0		ss:0002	2040	
ds:0008	60 FE 01 00 00 00 00 00	00 00 00 00		ss:0000	5010	
ds:0010	B8 53 0B 8E D8 BE 00 00	750A 1		ss:FFFE	0000	
ds:0018	8A 3C B9 09 00 46 3A 3C	E<10 F:<		ss:FFFC	082A	
ds:0020	72 02 8A 3C E2 F7 46 88	r0E<F&F&		ss:FFFA	0A95	

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

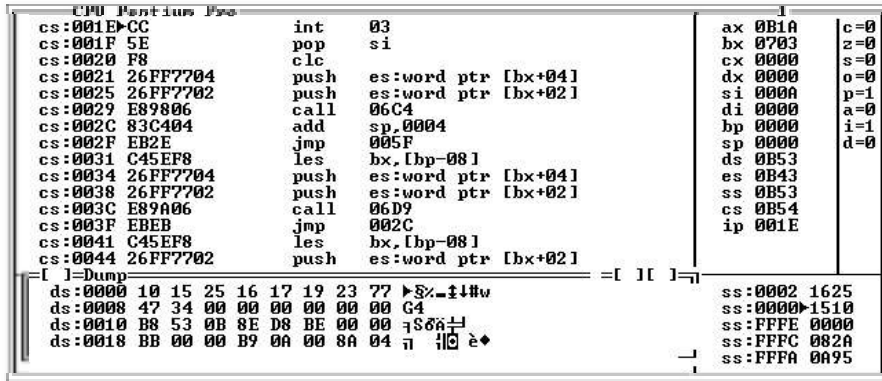
START: MOV AX,DATA  
MOV DS,AX

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:**

CPU Function Pro						
cs:0026	CC	int	03	ax	0BE0	c=0
cs:0027	7702	ja	002B	bx	0003	z=0
cs:0029	E89006	call	06C4	cx	0000	s=0
cs:002C	83C404	add	sp,0004	dx	0304	o=0
cs:002F	EB2E	jmp	005F	si	0000	p=1
cs:0031	C45EF8	les	bx,[bp-08]	di	0000	a=0
cs:0034	26FF7704	push	es:word ptr [bx+04]	bp	0000	i=1
cs:0038	26FF7702	push	es:word ptr [bx+02]	sp	0000	d=0
cs:003C	E89A06	call	06D9	ds	0B53	
cs:003F	E8EB	jmp	002C	es	0B43	
cs:0041	C45EF8	les	bx,[bp-08]	ss	0B53	
cs:0044	26FF7702	push	es:word ptr [bx+02]	cs	0B54	
cs:0048	E89F06	call	06EA	ip	0026	
cs:004B	59	pop	cx			
cs:004C	EB11	jmp	005F			
[ ]=Dump						
ds:0000	50 41 30 00 80 90 FF 00	PA0 C6		ss:0002	0030	
ds:0008	00 70 00 00 00 00 00 00	p		ss:0000	4150	
ds:0010	B8 53 0B 8E D8 BB 00 00	3800		ss:FFFF	0000	
ds:0018	BE 00 00 B9 0A 00 8A 04	3800		ss:FFFC	082A	
				ss:FFFA	0A95	

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

;JE 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->ZzYyXxWwUuTtSsRrQqPpOoNnMmLlKkJjIiHhGgFfEeDdCcBbAaABCDEF GHIJKL
mnopqrstuvwxyzzyxwvutsrqponMLKJIHG FEDCBA
THE LENGTH OF STRING IS->104
REVERSE OF YOUR STRING->ABCDEF GHIJKLmnopqrstuvwxyzzyxwvutsrqponMLKJIHG FEDCBAaAbB
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=ABCDEFGHIJKLMNPOQRSTUVWXYZ0123456789
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:**

CPU Position Prev				I			
cs:001F	CC	int	03	ax	4146	c=0	
cs:0020	398B4406	cmp	[bp+di+0644],cx	bx	4100	z=0	
cs:0024	8B5404	mov	dx,[si+04]	cx	0100	s=0	
cs:0027	EB7B	jmp	00A4	dx	0000	o=0	
cs:0029	33FF	xor	di,di	si	0002	p=0	
cs:002B	8B46FE	mov	ax,[bp-02]	di	0002	a=0	
cs:002E	8B56FC	mov	dx,[bp-04]	bp	0000	i=1	
cs:0031	83C204	add	dx,0004	sp	0000	d=0	
cs:0034	8946F0	mov	[bp-06],ax	ds	0B46		
cs:0037	8956F8	mov	[bp-08],dx	es	0B36		
cs:003A	EB58	jmp	0094	ss	0B46		
cs:003C	C45EF8	les	bx,[bp-08]	cs	0B47		
cs:003F	268B1F	mov	bx,es:[bx]	ip	001F		
I=Dump							
ds:0000	4D 41 44 41 4D 00 00 00	MADAM					
ds:0008	00 00 00 00 00 00 00 00						
ds:0010	B8 46 0B 8E D8 B5 00 BE	F00+					
ds:0018	00 00 BF 04 00 B1 02 8A						
				ss:0002	4144		
				ss:0000	414D		

(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        ; COMPARISON 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:**

```

ENTER THE STRING- RADAR
THE GIVEN STRING IS PALINDROME _

```

### 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!!!! '$'
A DB 13D,10D,"THE ENTERED STRING IS PALINDROMES$"
B DB 13D,10D,"THE ENTERED STRING IS NOT A PALINDROMES$"
INPUT DB 'ENTER A STRING->','$'
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 PLUS
BAKK:   DEC DI
        CMP AH,DH
        JNZ SKIP
        DEC CL
        JNZ BACK1
        JMP LAST

PLUS:   INC SI
        MOV AH,[SI]
        JMP BAAK
PLUS:   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:**

```

ENTER A STRING->
"INVALID INPUT".....PLS TRY AGAIN!!!
ENTER A STRING->NEUER ODD OR EUEN
THE ENTERED STRING IS PALINDROME_

```

\*\*\*\*\*

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 ; COMPARISON 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 ; CHECK FOR "SPACE" AT DI
CMP BL,20H
JE SKIPP
```

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

```
SKIP: 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:**

```

ENTER THE STRING- NO LEMON NO MELON
THE GIVEN STRING IS PALINDROME

```

**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:**

CPU Pentium Pro			I		
cs:0003	8ED8	mov ds,ax	ax	F4B4	c=0
cs:0005	A10000	mov ax,[0000]	bx	0000	z=0
cs:0008	F7260400	mul word ptr [0004]	cx	0000	s=0
cs:000C	A30800	mov [0008],ax	dx	04C5	o=0
cs:000F	89160A00	mov [000A],dx	si	0000	p=1
cs:0013	A10200	mov ax,[0002]	di	0000	a=0
cs:0016	F7260400	mul word ptr [0004]	bp	0000	i=1
cs:001A	01060A00	add [000A],ax	sp	0000	d=0
cs:001E	11160C00	adc [000C],dx	ds	0B46	
cs:0022	A10000	mov ax,[0000]	es	0B36	
cs:0025	F7260600	mul word ptr [0006]	ss	0B46	
cs:0029	01060A00	add [000A],ax	cs	0B47	
cs:002D	11160C00	adc [000C],dx	ip	0046	
[ ]=Dump					
ds:0000	34 12 34 12 21 43 21 43	4444C!C			
ds:0008	B4 F4 2D EE 3F FE C5 04	1F-C?11+			
ds:0010	B8 46 0B 8E D8 A1 00 00	1F0A+1			
ds:0018	F7 26 04 00 A3 08 00 89	3&+ u e			
			ss:0002	1234	
			ss:0000	1234	

### 10) 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 MATRICES'
B DB 13D,10D,10D,'THE 1st MATRICES'
C DB 13D,10D,10D,'THE 2nd MATRICES'
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:**

```
MULTIPLICATION OF 3X3 MATRIX

THE 1st MATRIX
ENTER THE 1st ROW 7 8 9
ENTER THE 2nd ROW 5 1 2
ENTER THE 3rd ROW 4 6 1

THE 2nd MATRIX
ENTER THE 1st ROW 7 8 9
ENTER THE 2nd ROW 9 5 1
ENTER THE 3rd ROW 9 9 9

THE RESULT OF MULTIPLICATION IS
202 177 152
62 63 64
91 71 51
```

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](#)

Enter your comment...



Comment as: [Ravi \(Google\)](#)

[Sign out](#)

[Publish](#)

[PREVIEW](#)

[Notify me](#)

[Home](#)

Subscribe to: [Post Comments \(Atom\)](#)

