

Aim :- Use of programming tools to perform basic arithmetic operations on 8 and 16 bit data

Theory :-

Assembling the program:-

To assemble the program two assemblers are available for the IBM-PC. They are : Micro Macro Assembler and Borland Turbo Assembler. Besides doing the tedious task of producing binary codes for the instruction statements, an assembler also allows the user to refer to data items by name rather than by numerical address. This makes the program much more readable. In addition to program instructions, the source program contains directives to the assembler.

~~MASM < file name. ASM >~~

The < file name. ASM > file that contains the assembly language program is assembled. The assembler generates error messages if there are any errors.

## Test and Debug :-

The executable program can be run under DOS or DEBUG. As a thumb rule a program under DOS only when there is no error or it produces some not visible or audible result. If the program result is stored in registers or in memory, the result is visible. Hence it should be run using DEBUG or TD or code-view only. EXE file can be loaded into memory using DEBUG.

## Commands :-

1) a (Assembler)

Format : -a starting address <enter>

It is used to enter the assembly language program in the code segment from the given offset address.

2) e (Enter)

Format : -e memory-address <enter>

It is used to edit (modify) or enter data segment from the given offset address. To enter data in consecutive memory locations use space bar for next memory location.



3) u (Un assemble)

Format: -u starting-address ending-address  
<enter>

It is used to unassemble program or part of the program. It displays Program or part of the program in following format.

Address op code mnemonic operand

XXXX : YYYY

It displays program or part of the program along with its operation codes from the starting-address to ending address.

4) g (go)

Format: -g = starting-address ending address  
<enter>

It is used to execute program or part of the program which is present in code segment from start-address to end-address. It displays the result if it is present in the registers of the 8086 Microprocessor.

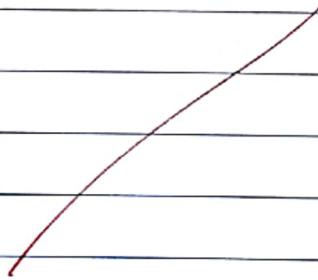
5) d (Dump or display)

Format: -d start-address end address <enter>

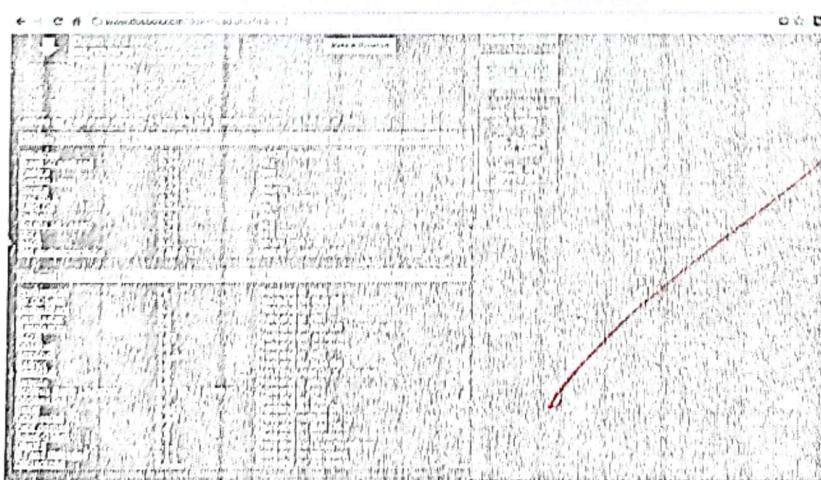
It is used to display the contents of the memory locations from the given address range. Hence it can be used to display result of the program if it is stored in the memory.

Conclusion :

All above debug commands of debug are studied by entering test program , data, executing program and displaying the contents of 8086 register as well as contents of memory locations

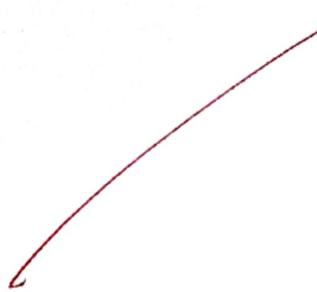


A screenshot of a DOSBox window titled "DOSBox 0.74, CPU speed: 3000 cycles, Framekip 0, Program: DOSEBOX". The window contains a large amount of text that is severely corrupted by noise, appearing as a dense, illegible pattern of black and white pixels. The text is completely unrecognizable.



edit: C:\emu\i8086\Mysource\mycode.asm

```
01 DATA SEGMENT
02     A      DB      0001H
03     B      DB      0002H
04     SUM   DB      ?
05     CARRY DB      00H
06
07 DATA ENDS
08
09 CODE SEGMENT
10     ASSUME CS: CODE, DS: DATA
11 START:
12     MOV AX, DATA
13     MOV DS, AX
14
15     MOV AL, A
16     ADD AL, B
17     JNC SKIP
18     INC CARRY
19 SKIP: MOV SUM, AL
20     MOV AH, 4CH
21     INT 21H
22 CODE ENDS
23
24 END START
```



24 col: 24 drag a file here to open

emulator mycode.exe

file math debug view external virtual devices virtual drive help

This image shows a vertical strip of paper, likely a page from a ledger or account book. It features four circular punch holes along its right edge, suggesting it was part of a binder or folder. The paper is oriented vertically, and the text is arranged in columns, though the specific details of the text are not clearly legible.

registers E400-0304

AX | 4C | 03 | F4200: FF255 RES

BX  
100 100  
F4202: CD 205 i

Dx 100 00

CS F400 F4206: 00 000 NULL

F4208: 00 000 NULL

SP EEEA F420A:00 000 NULL

F420C: 00 000 NULL

F420E: 00 000 NULL

DS 0710 F4210:00 000 NULL

3 0

Name : Vivek Manik Chaudhari

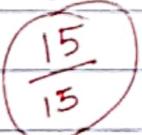
Roll No : 150

Batch : A2

DOP :

DOS :

Sign : 

Marks : 

Aim : WAP to convert BCD to Hexadecimal,  
hexadecimal to BCD.

Theory :

# 8086 program to convert an 8 bit  
BCD number into hexadecimal number.

Assumption :-

Initial value of segment register is  
00000. Calculation of physical memory  
address:

~~Memory Address = Segment Register \* 10H + offset~~  
~~where Segment Register is 00000 and Offset~~  
~~is given in the program.~~

Algorithm :-

1. Assign value 500 in SI and 600 in DI
2. Move the contents of [SI] in BL.
3. Use AND instruction to calculate AND  
between OF and contents of [BL]
4. Move the contents of [SI] in AL
5. USE AND instruction to calculate AND  
between Fo and contents of AL
6. Move 04 in CL
7. Use ROR instruction on AL
8. Move 0A in DL
9. Use MUL instruction to multiply AL  
with DI



10. Use ADD instruction to ADD AL with BL

11. Move the contents of AL in EDI

12. Halt the program.

# 8086 program to convert 8 bit BCD number into ASCII Code.

Assumption :-

Starting address of program : 400

Input memory location : 2000

Output memory location : 3000

Input :

DATA : 98H in memory location 2000

Output :

DATA : 38H in memory location 3000 and

39H in memory location 3001

Algorithm :

1. Load contents of memory location 2000 in register AL

2. Copy contents of register AL in register AH.

3. Perform AND operation register AL with 0F

4. Assign 04 to CL Register.

5. Shift the contents of AH by executing SHR instruction using CL

6. Perform OR operation on register AX with 3030
7. Store the content of AX in memory location 3000.

Problem : Write a program to convert ASCII to BCD 8-bit number where starting address is 2000 and number is stored at 2050 memory address and store result into 3050 memory address.

Input : Location - 2050

Data - 37

Output : Location - 3050

Data - 07

Algorithm :

1. Move value at [2050] into AL
2. Perform AND operation on AL with 0F
3. Move contents of accumulator AL into 3050
4. Stop

Explanation :

Register AL is used for general purposes.

# 8086 program to convert 8 bit ASCII to BCD number.

Problem : Write a program to convert ASCII to BCD 8-bit number where starting address is 2000 and number is stored at 2050 memory address and store result into 3050 memory address.

Input : Location - 2050

Data - 37

Output : Location - 3050

Data - 07

Algorithm :

1. Move value of [2050] into AL
2. Perform AND operation on AL with 0F
3. Move content of accumulator AL into 3050
4. Stop

Explanation :

Register AL is used for general purpose

1. MOV is used to transfer Data

2. AND is used for Multiplication (logically)

3. HLT is used to Halt program.

Conclusion :

We successfully implemented program to convert Dec BCD to Hexadecimal and Hexadecimal to BCD

file edit document settings change

new open examples save compile emulate calculator converter options help about

SEGMENT

```
ASSUME CS:Code
MOV DS,2X
MOV AL,[4000H]
AND AL,0FFH
MOV CL,0EH
SHR AL,CL
MOV BX,00E
MOV BL,0AH
MUL BL
MOV CL,[4000H]
AND CL,0EEH
ADD AL,CL
MOV [4001H],AL
INT 3H
ENDS
END
```

drag a file here to open

line: 15 | col: 5

file math debug view external virtual devices virtual drive help

Load

reload

step back

single step

step delay ms: 0

registers	H	L
AX	00	00
BX	00	0A
CX	00	00
DX	00	00
CS	0100	
IP	001D	
SS	0100	
SP	FFFF	
BP	0000	
SI	0000	
DI	0000	
DS	0000	
ES	0100	

0100:001D

010111:8A 138 S  
010121:0E 014 E  
010131:00 000 NULL  
010141:40 064 @  
010151:80 128 E  
010161:E1 225 E  
010171:0F 045 D  
010181:02 002 D  
010191:C1 193 A  
0101A1:A2 162 E  
0101B1:01 001 D  
0101C1:40 064 @  
0101D1:00 264 D

MOV DS, AX  
MOV AL, [04000h]  
AND AL, 0F0h  
MOV CL, 04h  
SHR AL, CL  
MOV AH, 00h  
MOV BL, 0Ah  
MUL BL  
MOV CL, [04000h]  
AND CL, DFh  
ADD AL, CL  
MOV [04001h], AL

NOP  
NOP  
NOP  
...  
INT3

0101E1:90 144 D

0101F1:90 144 D

010201:90 144 D

010211:90 144 D

source  
screen

reset  
aux

debug

stack  
flags

new open examples save compile emulate converter options help about

CODE SEGMENT  
ASSUME CS:Code  
MOV AX, 2000H  
MOV DS, BX  
MOV AL, [4000H]  
MOV AH, 0EH  
MOV BL, OAH  
DIV BL, BX+BX  
MOV CH, AX  
MOV AH, 0EH  
DIV BX  
MOV [4002E], AL  
MOV AL, AH  
MOV CL, 04H  
SLL BX, CL  
ADD BX, CH  
MOV [4001E], AL  
INT 3H

CODE ENDS



file math debug view external virtual devices virtual drive help

Load  
reload

step back  
single step

step delay ms: 0

registers

	H	L	
AX	00	00	01012: F6 246 0
BX	00	0A	01013: F3 243 5
CX	00	04	01014: A2 162 \$
DX	00	00	01015: 02 002 =
GS	010C		01016: 40 084 @
IP	0022		01017: 8A 438 S
SS	0100		01018: C4 196 A
SP	FFFE		01019: B1 177 ±
BP	0000		0101A: 04 004 =
SI	0000		0101B: D2 210 O
DI	0000		0101C: E0 224 à
DS	2000		0101D: 02 002 =
ES	0100		0101E: C5 197 A

0100:00022

0100:00022

0100:00022

MOV AX, 02000h  
MOV DS, AX  
MOV AL, [04000h]  
MOV AH, 09h  
MOV BL, 0Ah  
DIV BL  
MOV CH, AH  
MOV AH, 09h  
DIV BL  
MOV [04002h], AL  
MOV AL, AH  
MOV CL, 04h  
SHL AL, CL  
ADD AL, CH  
MOV [04001h], AL  
INT 3  
NOP

screen source reset aux vars debug stack flags