Exp No: 2 **Date:** 05/09/2020

16-bit Arithmetic Operations

Name: Swetha Saseendran

Register Number: 185001183

Aim:

To write and execute 8086 programs for arithmetic operations of 16 bit numbers like addition, subtraction, multiplication and division.

Procedure for executing MASM:

I. Open Dosbox and mount the
II. masm folder to the required drive using the command - ("mount drive-name location-of-masm-file")
III. Go to the mounted drive ("Drive-name:")
IV. Save the 8086 program with extension .asm in the same folder using command "edit" in Dosbox or use your desired editor and write your program and save in the same location where the masm file is located with extension asm.
V. Assemble it using the command ("masm filename.asm")
VI. Link the file using the command ("link filename.obj")
VII. Debug the file to execute and analyse the memory contents,
VIII. ("debug filename.exe").
IX. Now use command "u" to display the unassembled code.
X. Use command ("d segment:offset") to see the content of memory locations starting from segment:offset address
XI. Execute using the command "g" and check the outputs by repeating the previous step.
XII. Use command ("e segment:offset") to edit the variables.
XIII. Command "q" to exit from debug and command "exit" from command prompt to close dosbox.

Programs:

(i) 16-bit Addition:

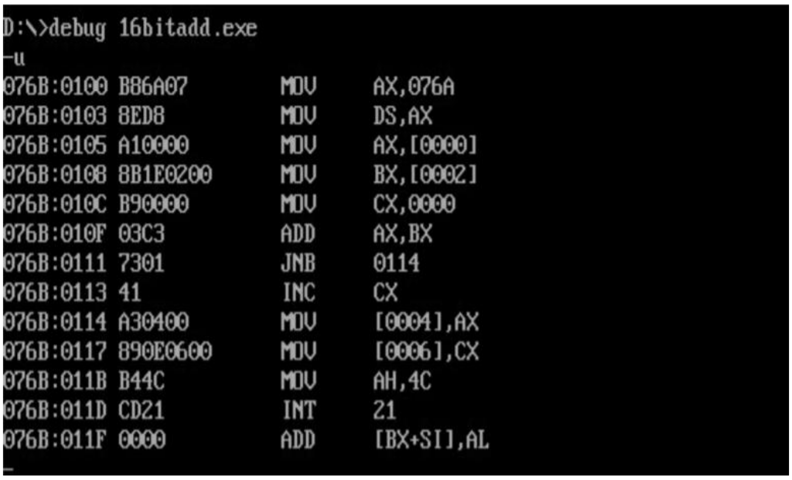
Algorithm:

- Move the data segment to the AX register and then move it to the DS register.
- Move the first operand to AX register.
- Move the second operand to the BX register
- Initially set the CX register to 0000h.
- Then add using ADD AX, BX.
- Using JNC instruction check for carry and if there is no carry, no need to increment CX.

- Else, increment CX by 1.
- The result and carry stored in AX and CX should be moved to RESULT and CARRY respectively.
- Terminate the program

Program	Comments
assume cs:code, ds:data	Using assume directive to declare data and code segment
data segment	Declaring and initialising variables in data segment
opr1 dw 1234h	
opr2 dw 5140h	
result dw 0000h	
carry db 00h	
data ends	
code segment	
org 0100h	Set location for code segment at 0100h
***start:** mov ax,data*	Transferring address of data segment to ds
mov ds,ax	
mov ax,opr1	Value of opr1 is loaded to ax
mov bx,opr2	Value of opr2 is loaded to bx
mov ch,0000h	Initializing the value of ch
add ax,bx	ah=ax+bx
jnc here	Jump to “here” segment if no carry is generated
inc cx	cx=cx+1
***here:** mov result,ah*	Load register value of ah to result
mov carry,ch	Load ch value to carry
mov ah,4ch	Moves the hexadecimal value 4c to ah.
int 21h	When Software interrupt 21 is called with AH=4C, then current process terminates
code ends	Ending the segment with the segment name
end start	

Unassembled Code:



```
D:\>debug 16bitadd.exe
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A10000        MOV     AX,[0000]
076B:0108 8B1E0200      MOV     BX,[0002]
076B:010C B90000        MOV     CX,0000
076B:010F 03C3          ADD     AX,BX
076B:0111 7301          JNB     0114
076B:0113 41            INC     CX
076B:0114 A30400        MOV     [0004],AX
076B:0117 890E0600      MOV     [0006],CX
076B:011B B44C          MOV     AH,4C
076B:011D CD21          INT     21
076B:011F 0000          ADD     [BX+SI],AL
-
```

Snapshot of sample input and output:

Input:

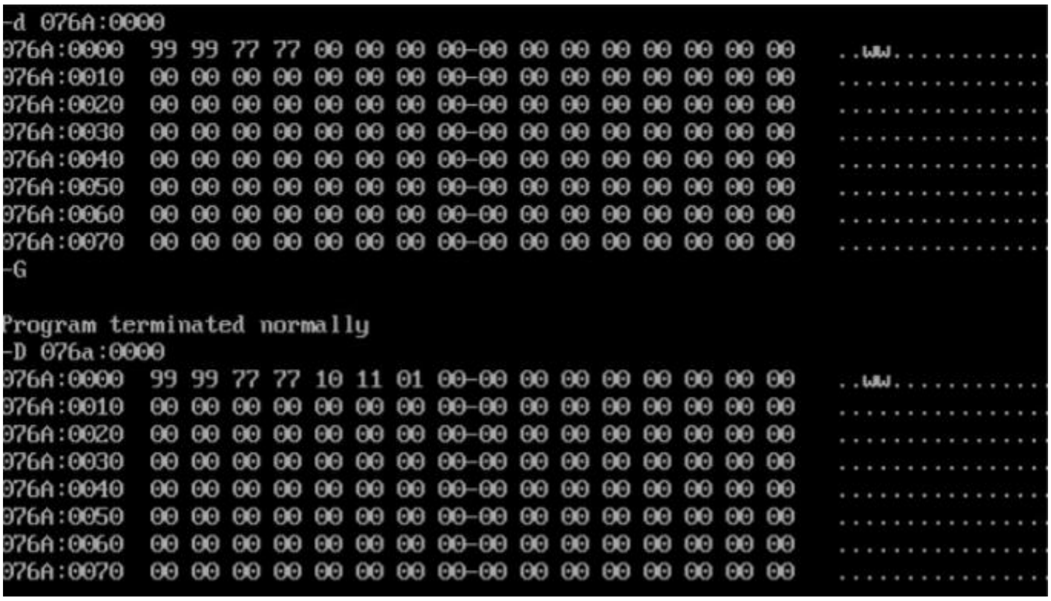
opr1=9999

opr2=7777

Output:

Result:1110

Carry: 0001



```
-d 076A:0000
076A:0000  99 99 77 77 00 00 00 00-00 00 00 00 00 00 00 00   ..ww............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G

Program terminated normally
-D 076a:0000
076A:0000  99 99 77 77 10 11 01 00-00 00 00 00 00 00 00 00   ..ww............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

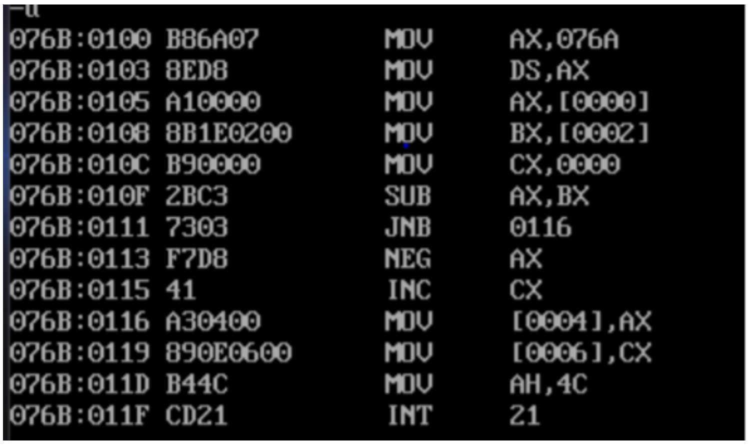
(ii) 16-bit Subtraction

Algorithm:

- Initialize the data segment
- Move data segment address to ds
- Load operand-1 to ax and operand-2 to bx
- Load 00h to ch register
- Subtract ax and bx
- If ax is greater than bx, goto here segment else, increment cx by 1 and find the 2’s complement of ah
- In here segment,
    - Load ax to result
    - Load cx to borrow
- Terminate the program

Program	Comments
assume cs:code, ds:data	Using assume directive to declare data and code segment
data segment	Declaring and initialising variables in data segment
opr1 dw 11h	
opr2 dw 99h	
result dw 00H	
borrow db 00H	
data ends	
code segment	
org 0100h	Set location for code segment at 0100h
start: *mov ax,data*	Transferring address of data segment to ds
mov ds,ax	
mov ax,opr1	Value of opr1 is loaded to ax
mov bx,opr2	Value of opr2 is loaded to bx
mov cx,0000h	Initializing the value of cx
sub ax,bx	ax=ax-bx
jnc here	Jump to "here" segment if ah>bh
inc cx	Increments ch by 1 (cx=cx+1)
neg a~~h~~X	2's complement of ah
here: *mov result,ax*	Load register value of ah to result
mov borrow,cx	Load ch value to borrow
mov ah,4ch	Moves the hexadecimal value 4c to ah.
int 21h	When Software interrupt 21 is called with AH=4C, then current process terminates
code ends	Ending the segment with the segment name

Unassembled Code:



```
-u
076B:0100 B86A07          MOV     AX,076A
076B:0103 8ED8            MOV     DS,AX
076B:0105 A10000          MOV     AX,[0000]
076B:0108 8B1E0200        MOV     BX,[0002]
076B:010C B90000          MOV     CX,0000
076B:010F 2BC3            SUB     AX,BX
076B:0111 7303            JNB     0116
076B:0113 F7D8            NEG     AX
076B:0115 41              INC     CX
076B:0116 A30400          MOV     [0004],AX
076B:0119 890E0600        MOV     [0006],CX
076B:011D B44C            MOV     AH,4C
076B:011F CD21            INT     21
```

Snapshot of sample input and output:

(i) **Input:**

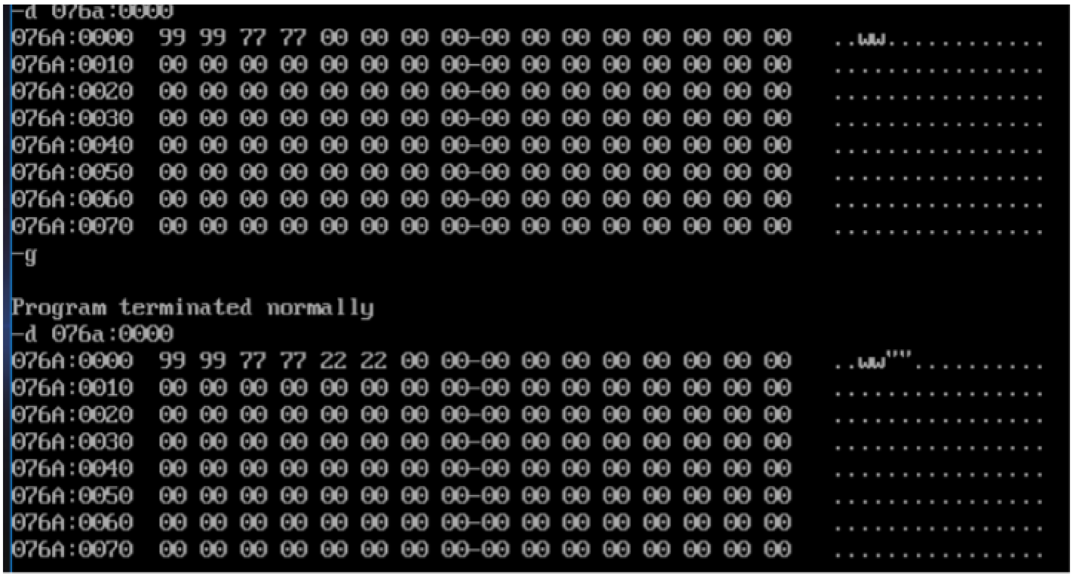
opr1=9999

opr2=7777

Output:

Result:2222

Carry:0000



```
-d 076a:0000
076A:0000  99 99 77 77 00 00 00 00-00 00 00 00 00 00 00 00   ..ww............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-g

Program terminated normally
-d 076a:0000
076A:0000  99 99 77 77 22 22 00 00-00 00 00 00 00 00 00 00   ..ww""..........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

(ii) **Input:**

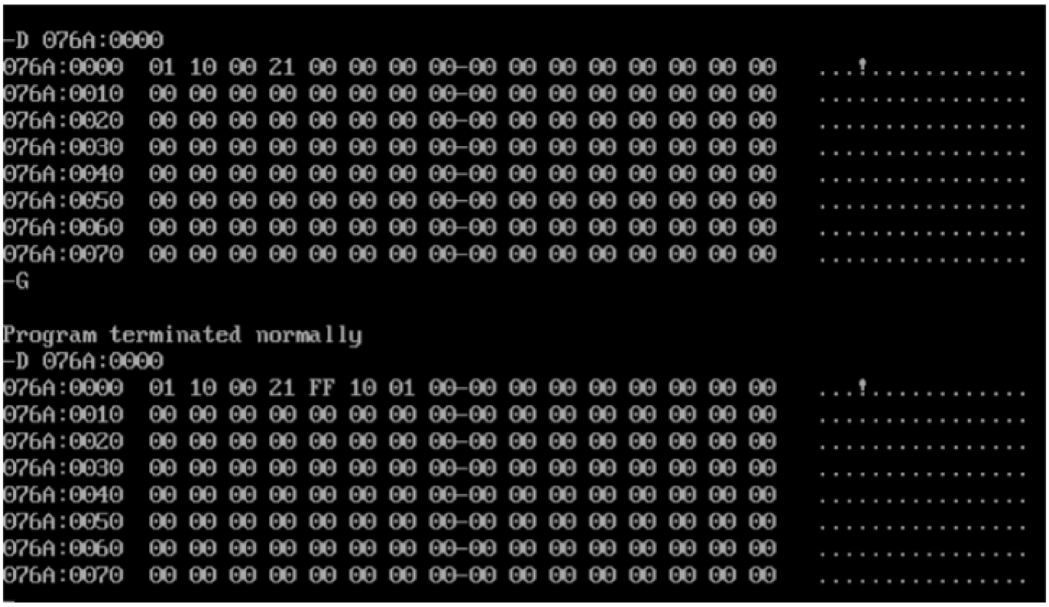
opr1=1001

opr2=2100

Output:

Result: 10FF

Carry: 0001



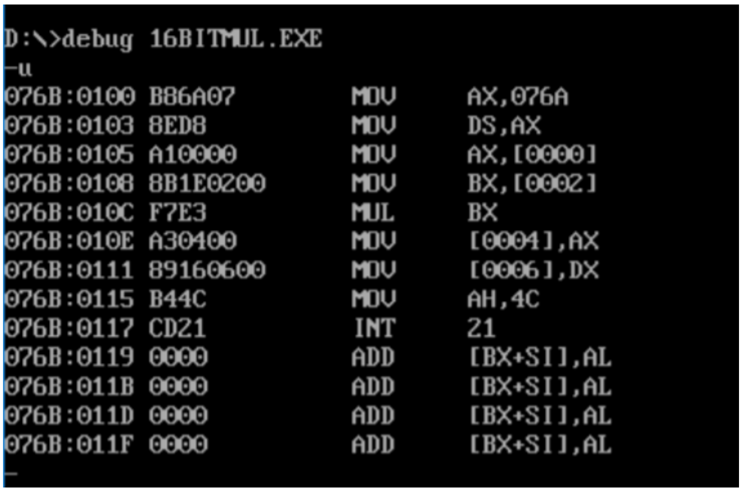
```
-D 076A:0000
076A:0000  01 10 00 21 00 00 00 00-00 00 00 00 00 00 00 00   ...!............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G

Program terminated normally
-D 076A:0000
076A:0000  01 10 00 21 FF 10 01 00-00 00 00 00 00 00 00 00   ...!............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

(iii) 16-bit Multiplication:

Algorithm:

- Initialize the data segment
- Move data segment address to ds
- Move the first operand to AX register.
- Move the second operand to the BX register.
- Multiply using MUL BX. (Since AX is default operand register for MUL instruction we only need to specify the other operand register.)
- The lower order and higher order result bits stored in AX and DX are now to be transferred to lsb and msb respectively

Unassembled Code:



```
D:\>debug 16BITMUL.EXE
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 A10000        MOV     AX,[0000]
076B:0108 8B1E0200      MOV     BX,[0002]
076B:010C F7E3          MUL     BX
076B:010E A30400        MOV     [0004],AX
076B:0111 89160600      MOV     [0006],DX
076B:0115 B44C          MOV     AH,4C
076B:0117 CD21          INT     21
076B:0119 0000          ADD     [BX+SI],AL
076B:011B 0000          ADD     [BX+SI],AL
076B:011D 0000          ADD     [BX+SI],AL
076B:011F 0000          ADD     [BX+SI],AL
-
```

Program	Comments
assume cs:code, ds:data	Using assume directive to declare data and code segment
data segment	Declaring and initialising variables in data segment
opr1 dw 1111h	
opr2 dw 9999h	
lsb dw 0000H	
msb dw 0000H	
data ends	
code segment	
org 0100h	Set location for code segment at 0100h
start: *mov ax,data*	Transferring address of data segment to ds
mov ds,ax	
mov al,opr1	Value of opr1 is loaded to al
mov bl,opr2	Value of opr2 is loaded to bl
mul bx	dx ax=ax x bx
mov result1,ax	Load register value of ax to result1
mov result2,dx	Load register value of dx to result2
mov ah,4ch	Moves the hexadecimal value 4c to ah.
int 21h	When Software interrupt 21 is called with AH=4C, then current process terminates
code ends	Ending the segment with the segment name

Snapshot of sample input and output:

Input :

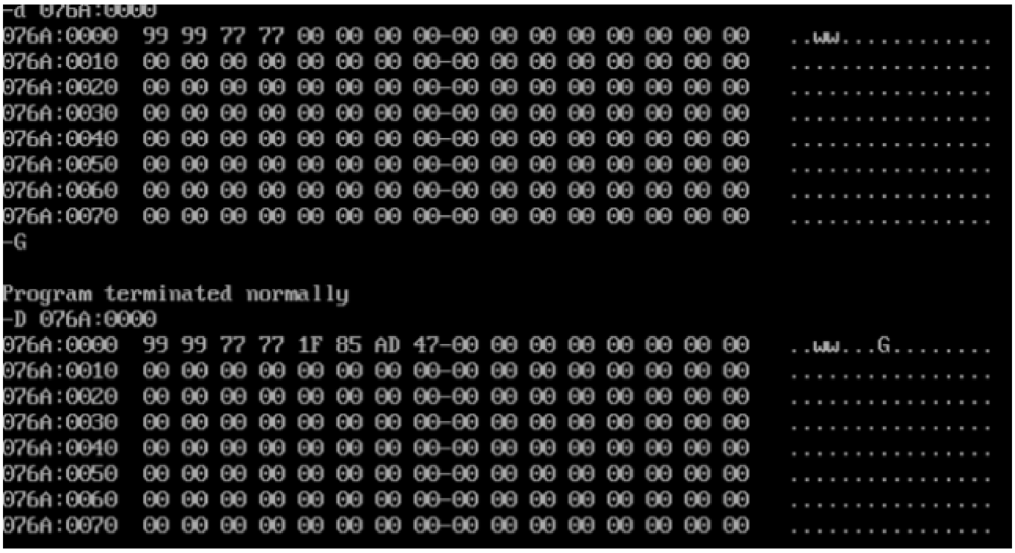
opr1=9999

opr2=7777

Output:

Result: 47AD851F

[Result2: 47AD Result1: 851F]



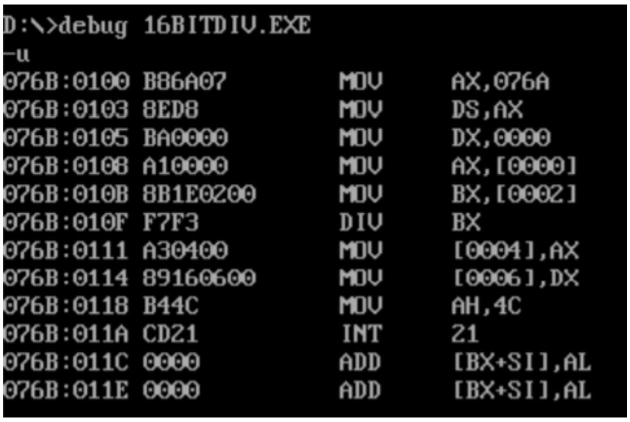
```
-d 076A:0000
076A:0000  99 99 77 77 00 00 00 00-00 00 00 00 00 00 00 00   ..ww............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G

Program terminated normally
-D 076A:0000
076A:0000  99 99 77 77 1F 85 AD 47-00 00 00 00 00 00 00 00   ..ww...G........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

(iv) 16-bit division:

Algorithm:

- Initialize the data segment
- Move data segment address to ds
- Move the data segment to the AX register and then move it to the DS register.
- Now, set DX register to 0000h and move first operand to AX register.(Since we can't directly divide a 16 bit number by 16 bit number in 8086, we now make our dividend 32 bit by storing 0000h in DX register and the 16-bit operand 1 in AX register).
- Move the second operand to the BX register.
- Now divide using DIV BX. (It will perform DXAX / BX. Because DX is 0000h, what actually happens is the division of a 32 bit number by a 16 bit number.)
- The quotient and remainder stored in AX and DX should be moved to QUOTIENT and REMAINDER respectively.

Unassembled Code:



```
D:\>debug 16BITDIV.EXE
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 BA0000        MOV     DX,0000
076B:0108 A10000        MOV     AX,[0000]
076B:010B 8B1E0200      MOV     BX,[0002]
076B:010F F7F3          DIV     BX
076B:0111 A30400        MOV     [0004],AX
076B:0114 89160600      MOV     [0006],DX
076B:0118 B44C          MOV     AH,4C
076B:011A CD21          INT     21
076B:011C 0000          ADD     [BX+SI],AL
076B:011E 0000          ADD     [BX+SI],AL
```

Program	Comments
assume cs:code, ds:data	Using assume directive to declare data and code segment
data segment	Declaring and initialising variables in data segment
opr1 dw 11h	
opr2 dw 99h	
quo dw 00h	
rem dw 00h	
data ends	
code segment	
org 0100h	Set location for code segment at 0100h
start: *mov ax,data*	Transferring address of data segment to ds
mov ds,ax	
mov ax,opr1	Register ah is loaded with 00
mov bl,opr2	Value of opr1 is loaded to ax
div bx	Value of opr2 is loaded to bl
mov quo,al	dx ax / bx
mov rem,ah	Load register value of al to result
mov ah,4ch	Moves the hexadecimal value 4c to ah.
int 21h	When Software interrupt 21 is called with AH=4C, then current process terminates
code ends	Ending the segment with the segment name

Snapshot of sample input and output:

Input:

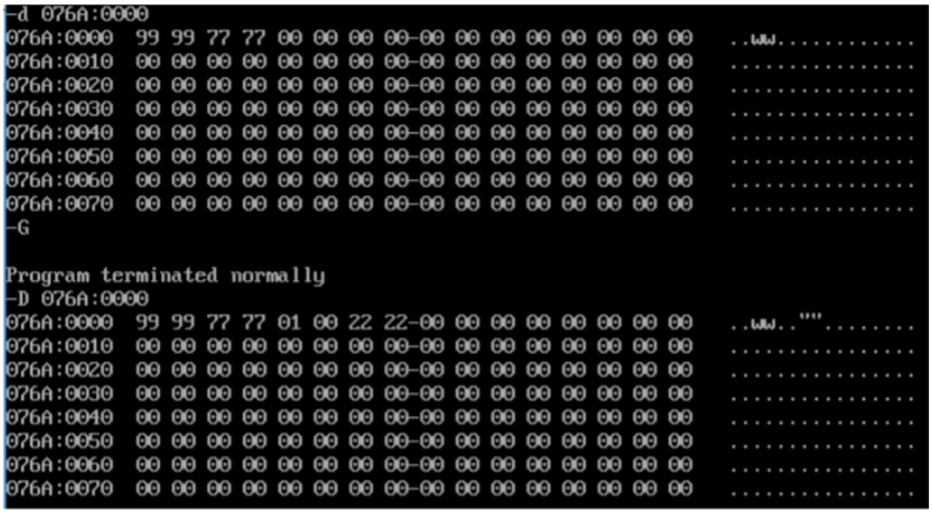
opr1=9999

opr2=7777

Output:

Quotient:0001

Remainder:2222



```
-d 076A:0000
076A:0000  99 99 77 77 00 00 00 00-00 00 00 00 00 00 00 00   ..ww............
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G

Program terminated normally
-D 076A:0000
076A:0000  99 99 77 77 01 00 22 22-00 00 00 00 00 00 00 00   ..ww..""........
076A:0010  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
076A:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
```

Result:

The assembly level program to perform basic arithmetic operation of two 16-bit numbers using an 8086 microprocessor has been implemented.