Aravind Kannan Rathinasabapathi
195001022
Date: 16-07-2021
Ex. No. 1

Exercise 1 – 8-bit arithmetic operations
1A – 8-bit addition

Aim:

To add two 8-bit numbers

Procedure for executing MASM:

1. Mount the local folder in the DOS-BOX using a temp disk name: `mount <disk-name> <folder-location>`
2. Change directory into the mounted disk: `<disk-name>: `
3. Assemble the instructions: `masm <file-name>.asm`
4. Link the object file(s) to produce an executable file(.exe): `link <file-name>.obj; ` Note that removal of semi-colon will make linking process interactive.
5. Debug the executable file to read the memory map and execute the program: `debug <file-name>.exe`. After entering debug mode,
    a. `d <segment:offset> ` - dump(read) memory map from the given location
    b. `e <segment:offset> ` - edit memory values from the given location. Use 'White space' to continue editing and 'new line' to exit editing.
    c. `u ` - unassemble code (with or without <segment:offset>)
    d. `g ` - execute the program
    e. `? ` - display command list
    f. `q` - quit the debugger

Algorithm:

1. Declare and initialize the data segment.
2. Begin code segment, where actual assembler instructions are present.
3. Change the origin at *100H*, offset where code segment starts storing instructions.
4. Move the starting address of data segment into ds register.
5. Store the addend in ah and augend in bh.
6. Initialize ch to be *00H*, to process carry.
7. Add ah and bh: ah = ah + bh
8. If carry generated, increment ch, else jump ahead of it.
9. Store ah in sum, ch in carry.
10. Terminate program and code segment.

Aravind Kannan Rathinasabapathi
195001022

Date: 16-07-2021
Ex. No. 1

Program:

Program	Comments
`;Program to add two 8-bit numbers`	Comment after ‘;’
`assume cs:code, ds:data`	Map CS to code segment and DS to data segment

`data segment`
`    addend db 0f0H`
`    augend db 1fH`
`    sum db 00H`
`    carry db 00H`
`data ends`
	Initialise data segment
	db = define a byte
	Initialise addend = 0f0H, augend = 1fH, sum = 00H, carry = 00H

`code segment` `        org 100H`	Initialise code segment Origin 100H = Offset set to 100H, to begin storing code instructions
`start:  mov ax, data` `        mov ds, ax`	Move the starting address of data segment in ax, then move ax to ds. Since in 8086, only code segment register is loaded automatically, the remaining segment register can be assigned using general purpose registers.

`        mov ah, addend`
`        mov bh, augend`
`        mov ch, 00H`
	Move addend to ah, augend to bh
	Initialise ch = 00H using direct addressing mode

`        add ah, bh`
`        jnc noCarry`
`        inc ch`
	Add ah and bh: ah = ah + bh
	Jump if no carry to ‘*noCarry*’ label
	Increment ch

`noCarry:mov sum, ah` `        mov carry, ch`	Move ah to sum Move ch to carry
`        mov ah, 4cH` `        int 21H`	Set ah = 4cH Call interrupt routine 21H for DOS, which terminates if ah = 4cH
`code ends` `end start`	

Aravind Kannan Rathinasabapathi
195001022

Unassembled code:

```
D:\>debug 8BITADD.EXE
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 8A260000      MOV     AH,[0000]
076B:0109 8A3E0100      MOV     BH,[0001]
076B:010D B500          MOV     CH,00
076B:010F 02E7          ADD     AH,BH
076B:0111 7302          JNB     0115
076B:0113 FEC5          INC     CH
076B:0115 88260200      MOV     [0002],AH
076B:0119 882E0300      MOV     [0003],CH
076B:011D B44C          MOV     AH,4C
076B:011F CD21          INT     21
```

Aravind Kannan Rathinasabapathi
195001022

Snapshot of sample input and output:

Case i: Without Carry

Hexadecimal addition - F0 + 0F = FF (Sum: FF, Carry: 00)

```
-e 076a:0000
076A:0000  F0.F0   1F.0F

-d 076a:0000
076A:0000  F0 0F 00 00 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  F0 0F FF 00 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   ................
```

Case ii: With Carry

Hexadecimal addition - F0 + 1F = 10F (Sum: 0F, Carry: 01)

```
-e 076a:0000
076A:0000  F0.F0   1F.1F

-d 076a:0000
076A:0000  F0 1F 00 00 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  F0 1F 0F 01 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   ................
```

Result:

Program to add two 8-bit numbers assembled, executed and verified.

Aravind Kannan Rathinasabapathi
195001022

Date: 16-07-2021
Ex. No. 1

1B – 8-bit subtraction

Aim:

To subtract two 8-bit numbers

Algorithm:

1. Declare and initialize the data segment.
2. Begin code segment, where actual assembler instructions are present.
3. Change the origin at *50H*, offset where code segment starts storing instructions.
4. Move the starting address of data segment into ds register.
5. Store the minuend ah and subtrahend in bh.
6. Initialize ch to be *00H*, to process diff.
7. Subtract ah and bh: ah = ah - bh
8. If carry generated, increment sign to denote negative and take 2’s complement of ah, else jump ahead of it.
9. Store ah in diff, ch in sign.
10. Terminate program and code segment.

Aravind Kannan Rathinasabapathi
195001022
Date: 16-07-2021
Ex. No. 1

Program:

<table>
<tr><th>Program</th><th>Comments</th></tr>
<tr><td><pre>;Program to subtract two 8-
bit numbers

assume cs:code, ds:data

data segment
    minuend db 0ffH
    subtrahend db 0feH
    diff db 00H
    sign db 00H
data ends

code segment
        org 50H

start: mov ax, data
        mov ds, ax

        mov ah, minuend
        mov bh, subtrahend
        mov ch, 00H

        sub ah, bh
        jnc trueForm

        inc ch
        neg ah

trueForm:
        mov diff, ah
        mov sign, ch

        mov ah, 4cH
        int 21H
code ends
end start</pre></td><td>Comment after ‘;’

Map CS to code segment and DS to data segment

Initialise data segment
db = define a byte
Initialise minuend = 0ffH, subtrahend = 0feH, diff = 00H, sign = 00H

Initialise code segment
Origin 50H = Offset set to 50H, to begin storing code instructions

Move the starting address of data segment in ax, then move ax to ds.

Move minuend to ah, subtrahend to bh

Initialise ch = 00H using direct addressing mode

Sub ah and bh: ah = ah - bh
Jump if no carry to ‘trueForm’ label

Increment ch
Negate – Take 2’s complement of ah

Move ah to diff
Move ch to sign

Set ah = 4cH
Call interrupt routine 21H for DOS, which terminates if ah = 4cH</td></tr>
</table>

Aravind Kannan Rathinasabapathi
195001022

Date: 16-07-2021
Ex. No. 1

Unassembled code:

```
D:\>debug 8BITSUB.EXE
-u
076B:0050 B86A07          MOV     AX,076A
076B:0053 8ED8            MOV     DS,AX
076B:0055 8A260000        MOV     AH,[0000]
076B:0059 8A3E0100        MOV     BH,[0001]
076B:005D B500            MOV     CH,00
076B:005F 2AE7            SUB     AH,BH
076B:0061 7304            JNB     0067
076B:0063 FEC5            INC     CH
076B:0065 F6DC            NEG     AH
076B:0067 88260200        MOV     [0002],AH
076B:006B 882E0300        MOV     [0003],CH
076B:006F B44C            MOV     AH,4C
076B:0071 CD21            INT     21
```

Aravind Kannan Rathinasabapathi Date: 16-07-2021
195001022 Ex. No. 1

Snapshot of sample input and output:

Case i: Minuend > Subtrahend = Positive difference
Hexadecimal subtraction - FF - FE = (Difference: 01, Sign: 00)

```
-e 076a:0000
076A:0000  FF.FF   FE.FE

-d 076a:0000
076A:0000  FF FE 00 00 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  B8 6A 07 8E D8 8A 26 00-00 8A 3E 01 00 B5 00 2A   .j....&...>....*
076A:0070  E7 73 04 FE C5 F6 DC 88-26 02 00 88 2E 03 00 B4   .s......&.......
-g

Program terminated normally
-d 076a:0000
076A:0000  FF FE 01 00 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  B8 6A 07 8E D8 8A 26 00-00 8A 3E 01 00 B5 00 2A   .j....&...>....*
076A:0070  E7 73 04 FE C5 F6 DC 88-26 02 00 88 2E 03 00 B4   .s......&.......
```

Case ii: Minuend < Subtrahend = Negative difference
Hexadecimal subtraction - FE - FF = (Difference: 01, Sign: 01)

```
-e 076a:0000
076A:0000  FF.FE   FE.FF

-d 076a:0000
076A:0000  FE FF 00 00 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  B8 6A 07 8E D8 8A 26 00-00 8A 3E 01 00 B5 00 2A   .j....&...>....*
076A:0070  E7 73 04 FE C5 F6 DC 88-26 02 00 88 2E 03 00 B4   .s......&.......
-g

Program terminated normally
-d 076a:0000
076A:0000  FE FF 01 01 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  B8 6A 07 8E D8 8A 26 00-00 8A 3E 01 00 B5 00 2A   .j....&...>....*
076A:0070  E7 73 04 FE C5 F6 DC 88-26 02 00 88 2E 03 00 B4   .s......&.......
```

Result:

Program to subtract two 8-bit numbers assembled, executed and verified.

Aravind Kannan Rathinasabapathi
195001022

1C – 8-bit multiplication

Aim:

To multiply two 8-bit numbers

Algorithm:

1. Declare and initialize the data segment.
2. Begin code segment, where actual assembler instructions are present.
3. Change the origin at *50H*, offset where code segment starts storing instructions.
4. Move the starting address of data segment into ds register.
5. Store the multiplicand in al and multiplier in bl.
6. Multiply al and bl: ax = al * bl
7. Store ax in prod.
8. Terminate program and code segment.

Aravind Kannan Rathinasabapathi
195001022

Date: 16-07-2021
Ex. No. 1

Program:

Program	Comments
`;Program to multiply two 8-bit numbers`	Comment after ‘;’
`assume cs: code, ds: data`	Map CS to code segment and DS to data segment

`data segment`
`    multiplicand db 10H`
`    multiplier db 10H`
`    prod dw 0000H`
`data ends`
	Initialise data segment
	db = define a byte
	dw = define a word
	Initialise multiplicand = 10H, multiplier = 0feH, prod = 0000H

`code segment` `        org 50H`	Initialise code segment Origin 50H = Offset set to 50H, to begin storing code instructions
`start:  mov ax, data` `        mov ds, ax`	Move the starting address of data segment in ax, then move ax to ds.
`        mov al, multiplicand` `        mov bl, multiplier`	Move multiplicand to al, multiplier to bl
`        mul bl`	Multiply al and bl: ax = al * bl (Fixed instruction)
`        mov prod, ax`	Move ax to prod

`        mov ah, 4cH`
`        int 21H`
`code ends`
`end start`
	Set ah = 4cH
	Call interrupt routine 21H for DOS, which terminates if ah = 4cH

Aravind Kannan Rathinasabapathi Date: 16-07-2021
195001022 Ex. No. 1

Unassembled code:

```
D:\>debug 8BITMUL.EXE
-u
076B:0050 B86A07          MOV     AX,076A
076B:0053 8ED8            MOV     DS,AX
076B:0055 A00000          MOV     AL,[0000]
076B:0058 8A1E0100        MOV     BL,[0001]
076B:005C F6E3            MUL     BL
076B:005E A30200          MOV     [0002],AX
076B:0061 B44C            MOV     AH,4C
076B:0063 CD21            INT     21
```

Snapshot of sample input and output:

Lower half of result stored in lower address value (here, 0001)
Higher half of result stored in higher address value (here, 0002)
That's lower half and higher half is mixed

Case i: Multiplication with zero
Hexadecimal multiplication - 10 x 00 = 00 00

```
-e 076a:0000
076A:0000  10.10   10.00

-d 076a:0000
076A:0000  10 00 00 00 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  B8 6A 07 8E D8 A0 00 00-8A 1E 01 00 F6 E3 A3 02   .j..............
076A:0070  00 B4 4C CD 21 F6 DC 88-26 02 00 88 2E 03 00 B4   ..L.!...&.......
-g

Program terminated normally
-d 076a:0000
076A:0000  10 00 00 00 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  B8 6A 07 8E D8 A0 00 00-8A 1E 01 00 F6 E3 A3 02   .j..............
076A:0070  00 B4 4C CD 21 F6 DC 88-26 02 00 88 2E 03 00 B4   ..L.!...&.......
```

Aravind Kannan Rathinasabapathi
195001022

Case ii: Multiplication producing only al
Hexadecimal multiplication – 0F x 01 = 00 0F

```
-e 076a:0000
076A:0000  10.0F   10.01

-d 076a:0000
076A:0000  0F 01 00 00 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  B8 6A 07 8E D8 A0 00 00-8A 1E 01 00 F6 E3 A3 02   .j..............
076A:0070  00 B4 4C CD 21 F6 DC 88-26 02 00 88 2E 03 00 B4   ..L.!...&.......
-g

Program terminated normally
-d 076a:0000
076A:0000  0F 01 0F 00 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  B8 6A 07 8E D8 A0 00 00-8A 1E 01 00 F6 E3 A3 02   .j..............
076A:0070  00 B4 4C CD 21 F6 DC 88-26 02 00 88 2E 03 00 B4   ..L.!...&.......
```

Case iii: Multiplication producing al and ah
Hexadecimal multiplication - 10 x 10 = 01 00

```
-e 076a:0000
076A:0000  10.10   10.10

-d 076a:0000
076A:0000  10 10 00 00 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  B8 6A 07 8E D8 A0 00 00-8A 1E 01 00 F6 E3 A3 02   .j..............
076A:0070  00 B4 4C CD 21 F6 DC 88-26 02 00 88 2E 03 00 B4   ..L.!...&.......
-g

Program terminated normally
-d 076a:0000
076A:0000  10 10 00 01 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  B8 6A 07 8E D8 A0 00 00-8A 1E 01 00 F6 E3 A3 02   .j..............
076A:0070  00 B4 4C CD 21 F6 DC 88-26 02 00 88 2E 03 00 B4   ..L.!...&.......
```

Result:

Program to multiply two 8-bit numbers assembled, executed and verified.

Aravind Kannan Rathinasabapathi
195001022

1D – 8-bit division

Aim:

To divide two 8-bit numbers

Algorithm:

1. Declare and initialize the data segment.
2. Begin code segment, where actual assembler instructions are present.
3. Change the origin at *100H*, offset where code segment starts storing instructions.
4. Move the starting address of data segment into ds register.
5. Store the dividend in al and divisor in bl.
6. Divide al and bl: ax = al / bl
7. Store quotient generated at al and remainder at ah, by default property of the instruction.
8. Terminate program and code segment.

Aravind Kannan Rathinasabapathi
195001022
Date: 16-07-2021
Ex. No. 1

Program:

<table>
<tr><th>Program</th><th>Comments</th></tr>
<tr><td><pre>
;Program to divide two 8-
bit numbers

assume cs:code, ds:data

data segment
    dividend db 05H
    divisor db 03H
    quotient db 00H
    remainder db 00H
data ends

code segment
        org 100H

start: mov ax, data
        mov ds, ax

        mov ah, 00h
        mov al, dividend
        mov bl, divisor

        div bl

        mov quotient, al
        mov remainder, ah

        mov ah, 4cH
        int 21H
code ends
end start
</pre></td><td>Comment after ‘;’

Map CS to code segment and DS to data segment

Initialise data segment
db = define a byte
Initialise dividend = 05H, divisor = 03H, quotient = 00H, remainder = 00H

Initialise code segment
Origin 100H = Offset set to 100H, to begin storing code instructions

Move the starting address of data segment in ax, then move ax to ds.

Move dividend to al, divisor to bl

Divide al and bl: ax = al / bl (Fixed instruction)

Move al to quotient
Move ah to remainder

Set ah = 4cH
Call interrupt routine 21H for DOS, which terminates if ah = 4cH</td></tr>
</table>

Unassembled code:

```
D:\>debug 8BITDIV.EXE
-u
076B:0100 B86A07        MOV     AX,076A
076B:0103 8ED8          MOV     DS,AX
076B:0105 B400          MOV     AH,00
076B:0107 A00000        MOV     AL,[0000]
076B:010A 8A1E0100      MOV     BL,[0001]
076B:010E F6F3          DIV     BL
076B:0110 A20200        MOV     [0002],AL
076B:0113 88260300      MOV     [0003],AH
076B:0117 B44C          MOV     AH,4C
076B:0119 CD21          INT     21
```

Snapshot of sample input and output:

Case i: With remainder

Hexadecimal Division: 05 / 03 = (Quotient: 01, Remainder: 02)

```
-e 076a:0000
076A:0000  05.05   03.03

-d 076a:0000
076A:0000  05 03 00 00 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  05 03 01 02 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   ................
```

Aravind Kannan Rathinasabapathi
195001022

Case ii: Without remainder
Hexadecimal Division: 06 / 03 = (Quotient: 02, Remainder: 00)

```
-e 076a:0000
076A:0000  05.06   03.03

-d 076a:0000
076A:0000  06 03 00 00 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  06 03 02 00 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   ................
```

Result:

Program to divide two 8-bit numbers assembled, executed and verified.