Name:	BARI ANKIT VINOD
Roll No:	65
Class/Sem:	SE/IV
Experiment No.:	1
Title:	To perform basic arithmetic operations on 16-bit data.
Date of Performance:	10/10/24
Date of Submission:	24/01/24
Marks:	
Sign of Faculty:	

Aim: Assembly Language Program to perform basic arithmetic operations (addition, subtraction, multiplication, and division) on 16-bit data.

Theory:

MOV: MOV Destination, Source.

The MOV instruction copies data from a specified destination. word or byte of data from a specified destination.

Source: Register, Memory Location, Immediate Number
Destination: Register, Memory Location

MOV CX, 037AH; Put immediate number 037AH to CX.

ADD: ADD Destination, Source.

These instructions add a number source to a number from some destination and put the result in the specified destination.

Source: Register, Memory Location, Immediate Number

Destination: Register, Memory Location

The source and the destination in an instruction cannot both be memory locations.

ADD AL, 74H; add the immediate number to 74H to the content of AL. Result in AL.

SUB: SUB Destination, Source.

These instructions subtract the number in some source from the number in some destination and put the result in the destination.

Source: Immediate Number, Register, or Memory Location.

Destination: Register or a Memory Location.

The source and the destination in an instruction cannot both be memory locations.

SUB AX, 3427H; Subtract immediate number 3427H from AX.

MUL: MUL Source.

This instruction multiplies an unsigned byte from some source times an unsigned byte in the AL register or an unsigned word from some source times an unsigned word in the AX register.

Source: Register, Memory Location.

MUL CX; Multiply AX with CX; result in high word in DX, low word in AX.

DIV: DIV Source.

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word (32 bits) by a word.

Source: Register, Memory Location.

If the divisor is 8-bit, then the dividend is in AX register. After division, the quotient is in AL and the remainder in AH.

If the divisor is 16-bit, then the dividend is in DX-AX register. After division, the quotient is in AX and the remainder in DX.

DIV CX; divide double word in DX and AX by word in CX; Quotient in AX; and remainder in DX.

<u>Algorithm to add two 16-bit numbers</u>

1. Load the first number in AX

2. Load the second number in BX

   3 Add the second number to AX

   4. Store the result in AX.

   <u>Algorithm to subtract two 16-bit numbers</u>

1. Load the first number in AX.

2. Load the second number. in BX 3. Subtract the second number to

   AX

   4. Store the result in AX.

   <u>Algorithm to multiply a 16-bit number by an 8-bit number</u>

1. Load the first number in AX.

2. Load the second number. in BL

3. Multiply DX and AX.

4. The result is in DX and AX.

   <u>Algorithm to divide a 16-bit number by an 8-bit number</u>

1. Load the first number in AX.

2. Load the second number. in BL

3. Divide AX by BL.

4. After division, the quotient is in AL and the remainder is in AH.
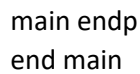
Code :

```
.org 100h

.data
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

  mov ax, @data
  mov ds, ax
```

```asm
    mov ax, num1
    add ax, num2
    mov result, ax

    mov ax, num1
    sub ax, num2
    mov result, ax

    mov ax, num1
    mov bx, num2
    mul bx
    mov result, ax

    mov ax, num1
    mov bx, num2
    div bx
    mov result, ax

    mov ax, 4ch
    int 21h

main endp
end main
```

Output :

```
.org 100h

.data
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

    mov ax, @data
    mov ds, ax

    mov ax, num1
    add ax, num2
    mov result, ax

    mov ax, num1
    sub ax, num2
    mov result, ax

    mov ax, num1
    mov bx, num2
    mul bx
    mov result, ax

    mov ax, num1
    mov bx, num2
    div bx
    mov result, ax

    mov ax, 4ch
    int 21h

main endp
end main
```

original source code

```
.code
main proc

mov ax, @data
mov ds, ax

mov ax, num1
add ax, num2
mov result, ax

mov ax, num1
sub ax, num2
mov result, ax

mov ax, num1
mov bx, num2
mul bx
mov result, ax
```

emulator: exp1.com

file math debug view external virtual devices virtual drive help

Load   reload   step back   single step   run   step delay ms: 0

registers   0700:0132   0700:0132

H	L
AX	00	00
BX	F4	00
CX	00	46
DX	00	F4
CS	0700
IP	0132
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0010
ES	0700

```
07130: F7 247 ≈     XOR AL, 012h
07131: E3 227 π     DB 67h
07132: A3 163 ú     INC BP
07133: 04 004 ◆     ADD [BX + SI], AL
07134: 00 000 NULL  ADD [BX + SI], AL
07135: A1 161 í     ADD [BX + SI], AL
07136: 00 000 NULL  ADD [BX + SI], AL
07137: 00 000 NULL  ADD [BX + SI], AL
07138: 8B 139 ï     ADD [BX + SI], AL
07139: 1E 030 ▲     MOV AX, 00010h
0713A: 02 002 ☻     MOV DS, AX
0713B: 00 000 NULL  MOV AX, [00000h]
0713C: F7 247 ≈     ADD AX, [00002h]
0713D: F3 243 ≤     MOV [00004h], AX
0713E: A3 163 ú     MOV AX, [00000h]
0713F: 04 004 ◆     SUB AX, [00002h]
07140: 00 000 NULL  MOV [00004h], AX
07141: B8 184 ╕     MOV AX, [00000h]
07142: 4C 076 L     MOV BX, [00002h]
07143: 00 000 NULL  MUL BX
07144: CD 205 =     MOV [00004h], AX
07145: 21 033 !     ...
```

screen source reset aux vars debug stack flags

```
.org 100h

.data
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

    mov ax, @data
    mov ds, ax

    mov ax, num1
    add ax, num2
    mov result, ax

    mov ax, num1
    sub ax, num2
    mov result, ax

    mov ax, num1
    mov bx, num2
    mul bx
    mov result, ax

    mov ax, num1
    mov bx, num2
    div bx
    mov result, ax

    mov ax, 4ch
    int 21h

main endp
end main
```

original source code

```
num1 dw 1234h
num2 dw 4567h
result dw ?

.code
main proc

mov ax, @data
mov ds, ax

mov ax, num1
add ax, num2
mov result, ax

mov ax, num1
sub ax, num2
mov result, ax
```

emulator: exp1.com

file math debug view external virtual devices virtual drive help

Load   reload   step back   single step   run   step delay ms: 0

registers   0700:0126   0700:0126

H	L
AX	0D	00
BX	00	00
CX	00	46
DX	00	00
CS	0700
IP	0126
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0010
ES	0700

```
07118: 03 003 ♥     XOR AL, 012h
07119: 06 006 ♠     DB 67h
0711A: 02 002 ☻     INC BP
0711B: 00 000 NULL  ADD [BX + SI], AL
0711C: A3 163 ú     ADD [BX + SI], AL
0711D: 04 004 ◆     ADD [BX + SI], AL
0711E: 00 000 NULL  ADD [BX + SI], AL
0711F: A1 161 í     ADD [BX + SI], AL
07120: 00 000 NULL  MOV AX, 00010h
07121: 00 000 NULL  MOV DS, AX
07122: 2B 043 +     MOV AX, [00000h]
07123: 06 006 ♠     ADD AX, [00002h]
07124: 02 002 ☻     MOV [00004h], AX
07125: 00 000 NULL  MOV AX, [00000h]
07126: A3 163 ú     SUB AX, [00002h]
07127: 04 004 ◆     MOV [00004h], AX
07128: 00 000 NULL  MOV AX, [00000h]
07129: A1 161 í     MOV BX, [00002h]
0712A: 00 000 NULL  MUL BX
0712B: 00 000 NULL  MOV [00004h], AX
0712C: 8B 139 ï     ...
0712D: 1E 030 ▲
```

screen source reset aux vars debug stack flags

```
01  .org 100h
02
03  .data
04  num1 dw 1234h
05  num2 dw 4567h
06  result dw ?
07
08  .code
09  main proc
10
11      mov ax, @data
12      mov ds, ax
13
14      mov ax, num1
15      add ax, num2
16      mov result, ax
17
18      mov ax, num1
19      sub ax, num2
20      mov result, ax
21
22      mov ax, num1
23      mov bx, num2
24      mul bx
25      mov result, ax
26
27      mov ax, num1
28      mov bx, num2
29      div bx
30      mov result, ax
31
32      mov ax, 4ch
33      int 21h
34
35  main endp
36  end main
37
38
```
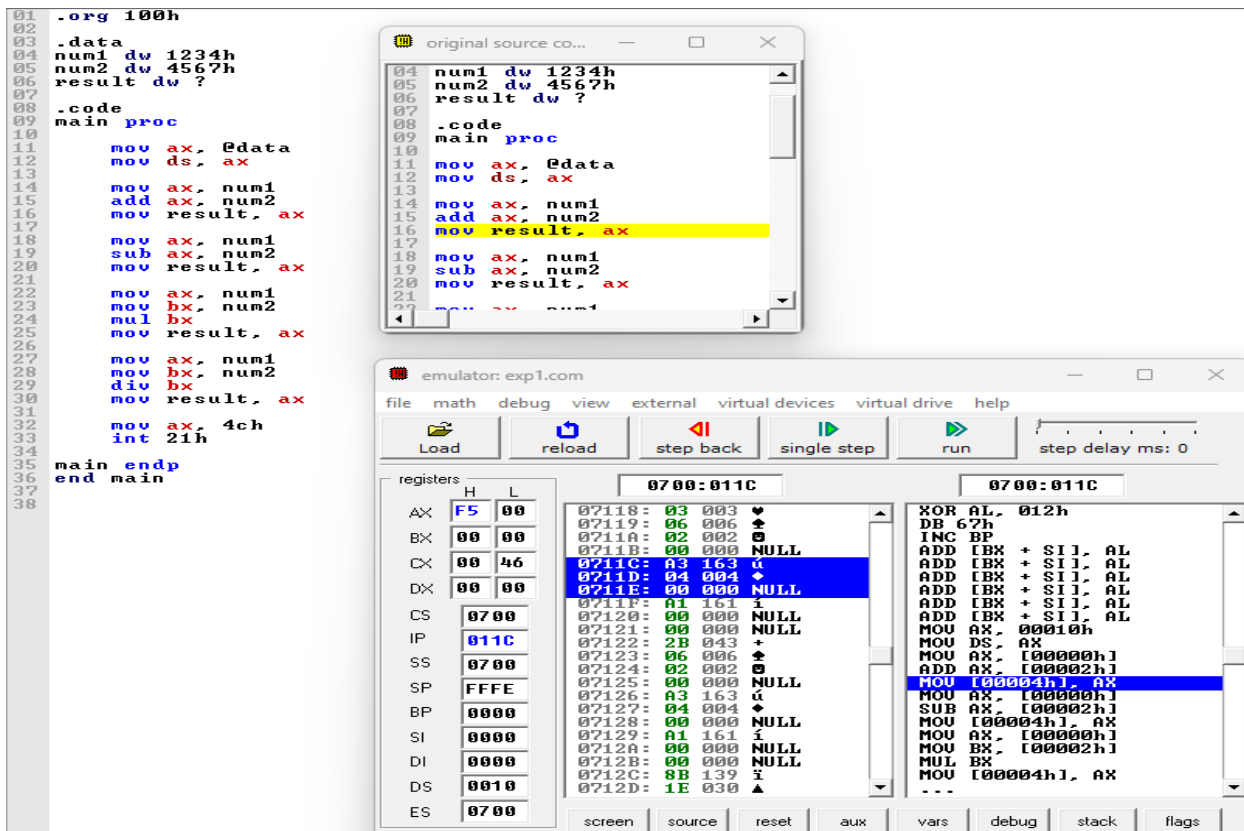

Conclusion:

In conclusion, the ability to perform basic arithmetic operations on 16-bit data is fundamental in various fields such as computer science, engineering, and mathematics. By efficiently manipulating 16-bit data, we can solve complex problems, process large datasets, and design intricate algorithms. Whether it's addition, subtraction, multiplication, or division, mastering these operations enables us to build robust systems, develop advanced technologies, and push the boundaries of innovation. As we continue to advance in the digital age, a solid understanding of arithmetic operations on 16-bit data remains an indispensable skill for professionals across diverse disciplines.