



Vidyavardhini's College of Engineering & Technology
Department of Artificial Intelligence and Data Science (AI&DS)

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:	24/01/24
Date of Submission:	24/01/24
Marks:	
Sign of Faculty:	



Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

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.



Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science (AI&DS)

Algorithm to add two 16-bit numbers

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.

Algorithm to subtract two 16-bit numbers

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.

Algorithm to multiply a 16-bit number by an 8-bit number

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.

Algorithm to divide a 16-bit number by an 8-bit number

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

```

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

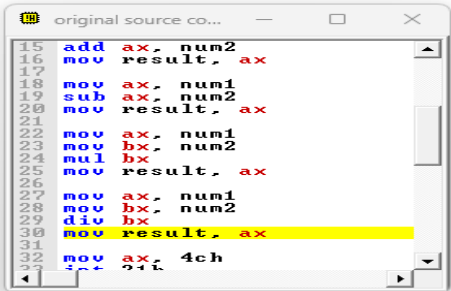
```

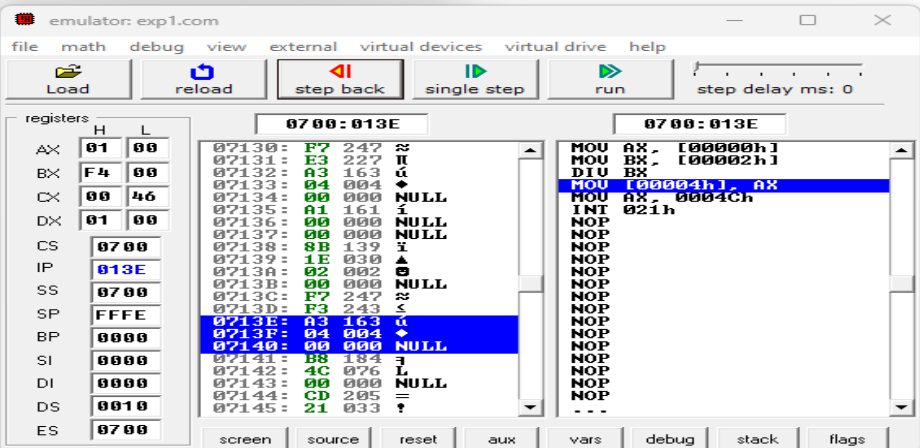
Output :

```

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

```





```

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, num2
25     mov result, ax
26
27     mov ax, num1
28     mov bx, num2
29     div bx, num2
30     mov result, ax
31
32     mov ax, 4ch
33     int 21h
34
35 main endp
36 end main
37
38

```

```

original source co...
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, num2
25     mov result, ax
26

```

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

	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	

0700:0132

07130:	E7	247	S
07131:	E3	222	I
07132:	A3	163	u
07133:	04	004	+
07134:	00	000	NULL
07135:	A1	161	i
07136:	00	000	NULL
07137:	00	000	NULL
07138:	8B	139	i
07139:	1E	030	+
0713A:	02	002	+
0713B:	00	000	NULL
0713C:	F7	247	S
0713D:	F3	243	S
0713E:	A3	163	u
0713F:	04	004	+
07140:	00	000	NULL
07141:	B8	184	+
07142:	4C	076	L
07143:	00	000	NULL
07144:	CD	205	+
07145:	21	033	!

0700:0132

```

XOR AL, 012h
DB 67h
INC BP
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
MOV DS, AX
MOV AX, [00000h]
ADD AX, [00002h]
MOV [00004h], AX
MOV AX, [00000h]
SUB AX, [00002h]
MOV [00004h], AX
MOV AX, [00000h]
MOV BX, [00002h]
MUL BX
MOV [00004h], AX

```

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, num2
25     mov result, ax
26
27     mov ax, num1
28     mov bx, num2
29     div bx, num2
30     mov result, ax
31
32     mov ax, 4ch
33     int 21h
34
35 main endp
36 end main
37
38

```

```

original source co...
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, num2
25     mov result, ax
26

```

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

	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	

0700:0126

07118:	03	003	+
07119:	06	006	+
0711A:	02	002	+
0711B:	00	000	NULL
0711C:	A3	163	u
0711D:	04	004	+
0711E:	00	000	NULL
0711F:	A1	161	i
07120:	00	000	NULL
07121:	00	000	NULL
07122:	2B	043	+
07123:	06	006	+
07124:	02	002	+
07125:	00	000	NULL
07126:	A3	163	u
07127:	04	004	+
07128:	00	000	NULL
07129:	A1	161	i
0712A:	00	000	NULL
0712B:	00	000	NULL
0712C:	8B	139	i
0712D:	1E	030	+

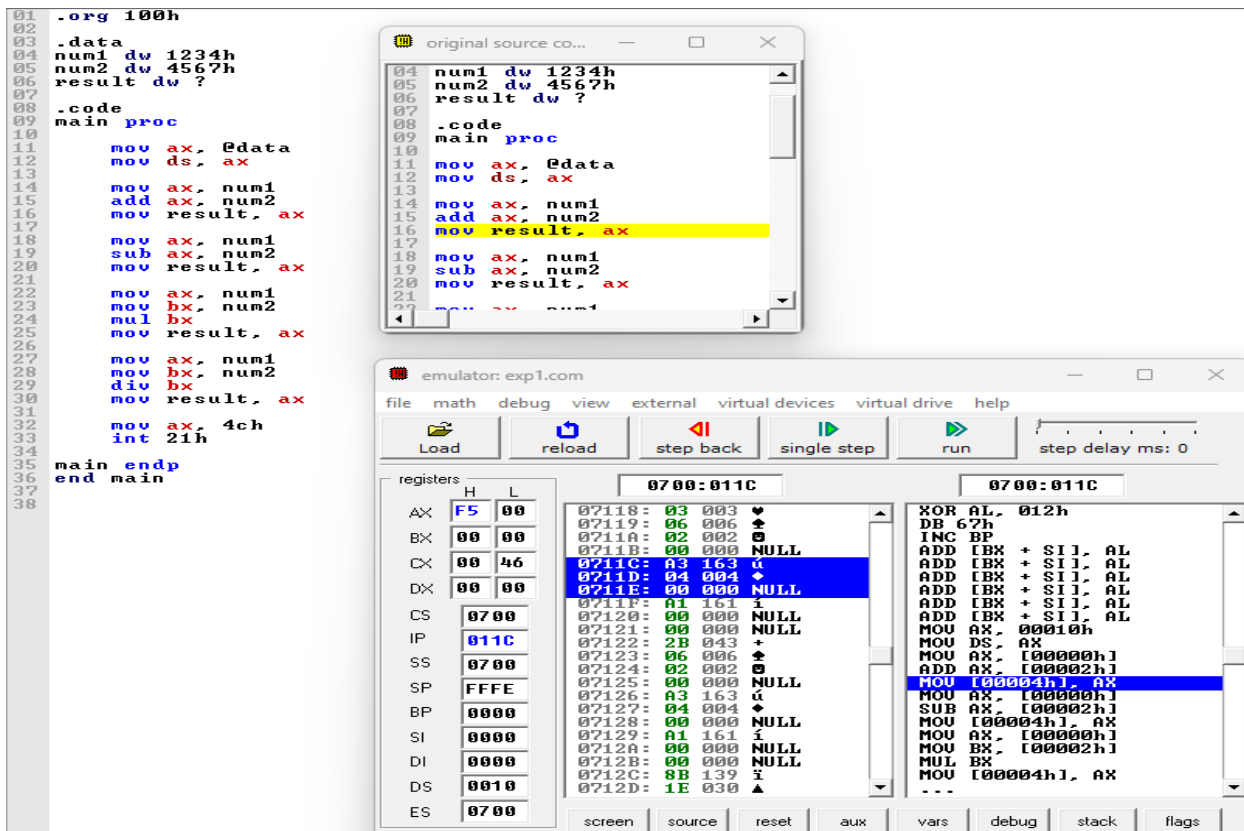
0700:0126

```

XOR AL, 012h
DB 67h
INC BP
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
MOV DS, AX
MOV AX, [00000h]
ADD AX, [00002h]
MOV [00004h], AX
MOV AX, [00000h]
SUB AX, [00002h]
MOV [00004h], AX
MOV AX, [00000h]
MOV BX, [00002h]
MUL BX
MOV [00004h], AX

```

screen source reset aux vars debug stack flags



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.