Name: **Tushar Nankani** Roll No: **1902112** Batch: **C23**

**Microprocessor: Experiment 2**

**Aim:** Write an assembly language menu driven program for 16-bit addition, subtraction, multiplication and division.

**THEORY:-**

**MOV** is used to load and store data.

**ADD** is used to add two numbers where their one number is in accumulator or not.

**JNC** is a 2-bit command which is used to check whether the carry is generated from accumulator or not.

**INC** is used to increment an register by 1.

**HLT** is used to stop the program.

**AX** is an accumulator which is used to load and store the data.

**BX**, **CX** are general purpose registers where BX is used for storing second number and CX is used to store carry.

The **CALL** instruction is used whenever we need to make a call to some procedure or a subprogram

The **RET** instruction stands for return. This instruction is used at the end of the procedures or the subprograms. This instruction transfers the execution to the caller program.

The **ADD** and **SUB** Instructions- The ADD and SUB instructions are used for performing simple addition/subtraction of binary data in byte, word and double word size, i.e., for adding or subtracting 8-bit, 16-bit or 32-bit operands, respectively.

The ADD and SUB instructions have the following syntax:- ADD/SUB destination, source

The MUL/IMUL Instruction- There are two instructions for multiplying binary data. The MUL (Multiply) instruction handles unsigned data and the IMUL (Integer Multiply) handles signed data. Both instructions affect the Carry and Overflow flag. Multiplicand in both cases will be in an accumulator, depending upon the size of the multiplicand and the multiplier and the generated product is also stored in two registers depending upon the size of the operands.

The syntax for the **MUL/IMUL** instructions is as follows:- MUL/IMUL multiplier

The **DIV/IDIV** InstructionsThe division operation generates two elements - a quotient and a remainder. In case of multiplication, overflow does not occur because double-length registers are used to keep the product. However, in case of division, overflow may occur. The processor generates an interrupt if overflow occurs. The DIV (Divide) instruction is used for unsigned data and the IDIV (Integer Divide) is used for signed data. The dividend is in an accumulator. Both the instructions can work with 8-bit, 16-bit or 32-bit operands.

The syntax for the DIV/IDIV instruction − DIV/IDIV divisor

**ALGORITHM: -**

1. Start

2. Display the message as “Enter first 16 bit number”

3. Read first digit in AL register through keyboard (e.g. AL=32h)

4. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=02h

5. Move contents of AH with 00h. (AHß 00h so AX=0002h)

6. Rotate AX contents in left directions by 12 bits. (AX=2000h)

7. Move the contents of AX to BX(BXßAX so BX=2000h)

8. Read a second digit in AL register through keyboard AL=35h

9. Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h

10.Move contents of AH with 00h. (AHß 00h so AX=0005h)

11.Rotate AX contents in left directions by 8 bits. (AX=0500h)

12.Add the contents of AX and BX (BX=BX+AX so BX=2500h)

13.Read a third digit in AL register through keyboard AL=31h

14.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=01h

15.Move contents of AH with 00h. (AHß 00h so AX=0001h)

16.Rotate AX contents in left directions by 4 bits. (AX=0010h)

17.Add the contents of AX and BX (BX=BX+AX so BX=2510h)

18.Read a fourth digit in AL register through keyboard AL=30h

19.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=00h 20.Move contents of AH with 00h. (AHß 00h so AX=0000h)

21.Add the contents of AX and BX (BX=BX+AX so BX=2510h)

22.Display the message as “Enter second 16 bit number”

23.Read first digit in AL register through keyboard (e.g. AL=37h)

24.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number.AL=07h

25.Move contents of AH with 00h. (AHß 00h so AX=0007h)

26.Rotate AX contents in left directions by 12 bits. (AX=7000h)

27.Move the contents of AX to CX(CXßAX so CX=7000h)

28. Read a second digit in AL register through keyboard AL=35h

29.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=05h

30.Move contents of AH with 00h. (AHß 00h so AX=0005h)

31.Rotate AX contents in left directions by 8 bits. (AX=0500h)

32.Add the contents of AX and CX (CX=CX+AX so CX=7500h)

33.Read a third digit in AL register through keyboard AL=31h

34.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=01h

35.Move contents of AH with 00h. (AHß 00h so AX=0001h)

36.Rotate AX contents in left directions by 4 bits. (AX=0010h)

37.Add the contents of AX and CX (CX=CX+AX so CX=7510h)

38.Read a fourth digit in AL register through keyboard AL=34h

39.Call Input procedure to make a number from ASCII hexadecimal to a normal hexadecimal number. AL=04h

40.Move contents of AH with 00h. (AHß 00h so AX=0004h)

41.Add the contents of AX and BX (CX=CX+AX so CX=7514h)

42.Compare accepted choice from AL with 02h

43.If zero flag is set then goto step no 69 otherwise goto step no 49

44.Add the contents of BX and CX(BX=BX+CX so BX=9A24h)

45.Preserve the result in temporary variable as t of 16 bit (so t=9A24h)

46.Mask the first nibble by AND operation with number F000h (AND BX,f000h so BX=9000h)

47.Rotate the BX contents right by 12(in decimal so BX=0009h)

48.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0039h)

49.Move the contents of BL to DL and display it on the screen

50.Move result from temporary variable t to BX again (Now BX=9A24h)

51.Mask the second nibble by AND operation with number 0F00h (AND BX,0F00h so BX=0A00h)

52.Rotate the contents of BX to right by 8(in decimal)

53.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0041h(ASCII hex value of ‘A’))

54.Move the contents of BL to DL and display it on the screen.

55.Move result from temporary variable to BX again (Now AX=9A24h)

56.Mask the third nibble by AND operation with number 00F0h (AND BX,00F0h so BX=0020h)

57.Rotate the contents of BX to right by 4(in decimal)

58.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=00032h) 59.Move the contents of BL to DL and display it on the screen

60.Move result back from temporary variable to BX again (Now BX=9A24h)

61.Mask the fourth nibble by AND operation with number 000Fh (AND BX,000fh so BX=0004h)

62.Call Output procedure with BL register to make a digit

back in ASCII hexadecimal range (BX=0004h) 63.Move the contents of BL to DL and display it on the screen.

64.Subtract the contents of CX from BX(BXßBX-CX so BX=AFFCh)

65.Preserve the result in temporary variable as t of 16 bit (so t=AFFCh)

66.Mask the first nibble by AND operation with number F000h (AND BX,F000h so BX=A000h)

67.Rotate the BX contents right by 12(in decimal so BX=000Ah)

68.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0041h (i.e. ASCII hex value for ‘A’ )) 69.Move the contents of BL to DL and display it on the screen

70.Move result from temporary variable t to BX again (Now BX=AFFCh)

71.Mask the second nibble by AND operation with number 0f00h (AND BX,0F00h so BX=0F00h)

72.Rotate the contents of BX to right by 8(in decimal)

73.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0046h(i.e ASCII hex value of ‘F’))

74.Move the contents of BL to DL and display it on the screen.

75.Move result from temporary variable to BX again (Now BX=AFFCh)

76.Mask the third nibble by AND operation with number 00F0h (AND BX,00F0h so BX=00F0h)

77.Rotate the contents of BX to right by 4(in decimal)

78.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0046h (i.e ASCII hex value of ‘F’))

79.Move the contents of BL to DL and display it on the screen

80.Move result back from temporary variable to BX again (Now BX=AFFCh)

81.Mask the fourth nibble by AND operation with number 000fh (AND AX,000Fh so AX=000Ch)

82.Call Output procedure with BL register to make a digit back in ASCII hexadecimal range (BX=0043h(i.e ASCII hex value of ‘C))

83.Move the contents of BL to DL and display it on the screen.

84.Stop

**Algorithm for Input procedure :( To accept input from 0 to f)**

Compare the contents of AL with 41h(Small case)

2. Jump to step no 4 if carry flag is set

3. Sub 07h to AL register

4. Sub 30h to AL register

5. Return.

**Algorithm for Output procedure:**

1. Compare the contents of BL with 0Ah

2. Jump to step no 4 if carry flag is set

3. Add 07h to AL register

4. Add 30h to AL register

5. Return.

**Code: -**

data Segment

msg db 0dh, 0ah, "Enter a 16-bit number: $"

result db 0dh, 0ah, "The Result is: $"

newl db 0dh, 0ah, " $"

menu db 0dh, 0ah, "1: add", 0dh, 0ah, "2: sub", 0dh, 0ah, "3: mul", 0dh, 0ah, "4: div", 0dh, 0ah, "$"

data ends

code Segment

assume cs:code, ds:data

Start:

mov ax, data

mov ds, ax

mov dx, offset msg

mov ah, 09h

int 21h

call AcceptNum

mov bh, bl

call AcceptNum

mov cx, bx

mov dx, offset msg

mov ah, 09h

int 21h

call AcceptNum

mov bh, bl

call AcceptNum

mov dx, offset menu

mov ah, 09h

int 21h

mov ah, 01h

int 21h

cmp al, '1'

jz Addition

cmp al, '2'

jz Subtraction

cmp al, '3'

jz Multiplication

cmp al, '4'

jz Division

Addition:

add cx, bx

jmp EndSwitch

Subtraction:

sub cx, bx

jmp EndSwitch

Multiplication:

mov ax, cx

mul bl

mov cx, ax

jmp EndSwitch

Division:

mov ah, 0

mov al, cl

div bl

mov ch, 0

mov cl, al

jmp EndSwitch

EndSwitch:

mov dx, offset result

mov ah, 09h

int 21h

mov bl, ch

call DispNum

mov bl, cl

call DispNum

mov dx, offset newl

mov ah, 09h

int 21h

mov ah, 4ch

int 21h

AcceptNum proc

mov ah, 01h

int 21h

call HexAccept

mov bl, al

rol bl, 4

mov ah, 01h

int 21h

call HexAccept

add bl, al

ret

endp

DispNum proc

mov al, bl

and al, 0f0h

ror al, 4

mov dl, al

call HexDisp

mov ah, 02h

int 21h

mov al, bl

and al, 0fh

mov dl, al

call HexDisp

mov ah, 02h

int 21h

endp

HexAccept proc ; Compare to 41 if it is less than A then we need to sub only 30

; If it is greater than or equal to 41 then we also need to sub 07

cmp al, 41h

jc norm

sub al, 07h

norm: sub al, 30h

ret

endp

HexDisp proc ; Compare to 0a if it is less than A then we need to add only 30

; If it is greater than or equal to 0a then we also need to add 07

cmp dl, 0ah

jc nothex

add dl, 07h

nothex: add dl, 30h

ret

endp

end Start

Code ends

**OUTPUT: -**







