NAME : Hael Larenz Y. Manalo **DATE** : November 28, 2024
SUBJ/SEC: Computer Architecture and Organization/BSCOE4-6 **PROFESSOR** : Engr. Rolito L. Mahaguay

EXPERIMENT NO. 6
WORKING WITH NUMBERS

OBJECTIVE (S) :

1. Understand how math works in assembly language programming.
2. Create a program that can perform numeric operations.

REQUIREMENTS :

Personal Computer
System disk
Data disk

DISCUSSION:

An assembly program consists of a set of statements. The two types of statements are ***instructions*** and ***directives.***

Instruction ***:*** such as MOV and ADD, which the assembler translates to object code
Directives ***:*** These tell the assembler to perform a specific action, such as defining a data item.
: An assembly language supports a number of statements that enable you to control the way in which a source program assembles and lists.

An operator provides a facility for changing or analyzing operands during assembly. Operators are divided into various categories:

- ***Calculation Operators***: Arithmetic, index, logic, shift, and structure field name.
- ***Record Operators***: MASK and WIDTH
- ***Relational Operators***: EQ, GE, GT, LE, LT, and NE
- ***Segment Operators***: OffSET, SEG, and segment override
- ***Type (or attribute) operators***: HIGH, HIGHWORD, LENGTH, LOW, LOWWORD, PTR, SHORT, SIZE THIS, and TYPE.

Arithmetic Operators

These operators include familiar arithmetic signs and perform arithmetic during the assembly. In most cases, you could perform the calculation yourself, although the advantage of using these operators is that every time you change the program and reassemble it, the assembler automatically recalculates with an example of their use.

PROCEDURE:

1. Encode the given program.

```
        .model small
        .code
        org 100h

start:  jmp main
        x db “INPUT A SINGLE DIGIT NUMBER : $”
        y db “INPUT ANOTHER SINGLE DIGIT NUMBER : $”
        z db “THEIR SUM IS : $”

main    proc near
        mov dx, offset x
        call print
        call input_ok
        mov cl, al
        call down
        mov dx, offset y
        call print
        call input_ok
        mov ch, al
        call down
        mov dx, offset z
        call print
        add ch, cl
        mov ah, 2
        mov dl, ch
        add dl, ch
        mov ah, 2
        mov dl, ch
        add dl, ‘0’
        int 21h
        int 20h
main    endp
```

```
down    proc near
        mov ah, 2
        mov dl, 13
        int 21h
        mov dl, 10
        int 21h
        ret


down    endp

print   proc near
        mov ah, 9
        int 21h
        ret

print   endp

input _ok  proc near
                mov ah,1
                int 21h
                sub al, '0'
                ret

input_ok  endp

end start
```

2. Execute the given program, then input 2 and 6 respectively.

```
C:\TASM>xp6samp
INPUT A SINGLE DIGIT NUMBER : 2
INPUT ANOTHER SINGLE DIGIT NUMBER : 6
THEIR SUM IS : 8
C:\TASM>
```

3. Does the expected output appear?
   Yes, the sum is correctly displayed.

4. Execute the program again, then input 5 and 7.

```
C:\TASM>xp6samp
INPUT A SINGLE DIGIT NUMBER : 5
INPUT ANOTHER SINGLE DIGIT NUMBER : 7
THEIR SUM IS : <
C:\TASM>_
```

5. Does the expected sum appear?
   No, the sum of 5 and 7 weren't displayed and instead it showed the < symbol.

6. Input another set of single-digit numbers which will give a sum above nine. What do you observe?

```
INPUT A SINGLE DIGIT NUMBER : 4
INPUT ANOTHER SINGLE DIGIT NUMBER : 6
THEIR SUM IS : :
C:\TASM>xp6samp
INPUT A SINGLE DIGIT NUMBER : 7
INPUT ANOTHER SINGLE DIGIT NUMBER : 8
THEIR SUM IS : ?
```

48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

   Based on the outputs, I observed that the sum of the numbers greater than 9 shows symbols corresponding to their respective ASCII count.

7. Modify the program so that it will be able to display the sum of two input numbers even if their sum is double-digit.

```
INPUT A SINGLE DIGIT NUMBER: 1
INPUT ANOTHER SINGLE DIGIT NUMBER: 2
THEIR SUM IS: 03
C:\TASM>xp6mod

INPUT A SINGLE DIGIT NUMBER: 5
INPUT ANOTHER SINGLE DIGIT NUMBER: 5
THEIR SUM IS: 10
```

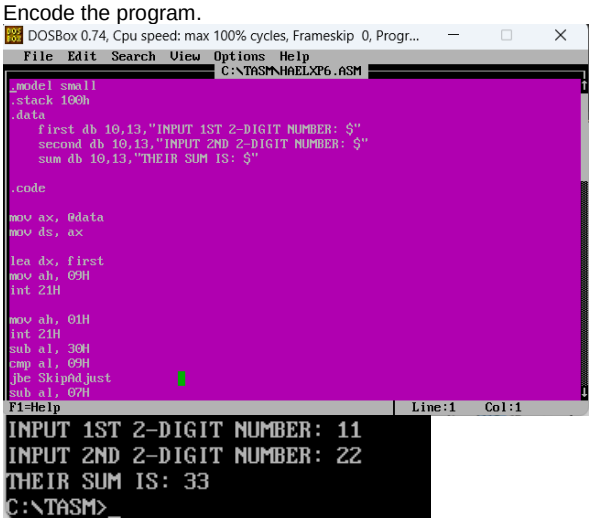
8. Write your new program in the space below.

```
.model small
.code
org 100h

start:  jmp main
        x db 13,10,'INPUT A SINGLE DIGIT NUMBER: $'
        y db 13,10, 'INPUT ANOTHER SINGLE DIGIT NUMBER: $'
        z db 13,10, 'THEIR SUM IS: $'


main:
    mov dx,offset x
    mov ah,9
    int 21h

    mov ah,1
    int 21h

    mov cl, al

    sub cl, 30h

    mov dx,offset y
    mov ah,9
    int 21h
    mov ah,1
    int 21h

    sub al, 30h

    xor ah,ah
    add al,cl
    aaa

    mov cx, ax
    add cx,3030h
    mov dx, offset z
    mov ah,9
    int 21h

    mov ah,2
    mov dl, ch
    int 21h

    mov dl, cl
    int 21h

exit:
    mov ah, 4ch
    int 21h


end start
```

EXERCISES:

1. Write down the task given by your instructor.
   Write a program that adds two two-digit numbers.

2. Encode the program.



3. Ask your instructor to check your work.

ANSWER THE FOLLOWING QUESTIONS:

1. From your ASCII table, write the ASCII code for the following characters.

CHARACTER	DECIMAL	HEXADECIMAL
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
+	43	2B
-	45	2D
*	42	2A
/	47	2F

2. Compare the arrangement of the ASCII codes of numbers and letters.
   The ASCII codes for numbers (0-9) are arranged sequentially from 48 to 57, where each character's code represents the respective digit. In contrast, the ASCII codes for uppercase letters (A-Z) range from 65 to 90, and lowercase letters (a-z) range from 97 to 122, with each character assigned a unique code that is not sequentially related to the numeric codes. Therefore, the ASCII code for numbers is placed before the codes for letters, reflecting their respective ranges and order in the ASCII table.

3. What method is used in getting the actual value of the input number?
   To convert an input character to its numeric value, subtract the ASCII code for '0' (48) from the character's ASCII code. This gives the digit's numeric value.

SUMMARY:
Assembly programs include instructions and directives, which control the creation of object code. Operators like arithmetic and relational operators modify operands during assembly, automating calculations and ensuring accuracy. The conversion of ASCII codes for numbers and letters helps in processing user input.

CONCLUSION:
Mastering assembly language, including operators and ASCII encoding, is key for efficient low-level programming. Arithmetic operations and converting ASCII codes to numbers enable precise handling of user input and effective data manipulation.