Hands-On Activity No. 5	
LOGICAL INSTRUCTIONS	
Course Code: CPE021	**Program:** Computer Engineering
Course Title: Computer Architecture and Organization	**Date Performed:** March 10, 2025
Section: CPE22S2	**Date Submitted:** March 10, 2025
Name: Adia, James Russel E.	**Instructor:** Engr. Maria Rizette H. Sayo

A. Procedure: Output(s) and Observation(s)

Sample Problem 1:

1. Type the following program in Notepad.

```
TITLE logic.asm
.model small
.stack 100h
.data
myStringdb "Proud to be TIPians","$"
.code
main proc
movax,@data
movds,ax
movbx,offset myString
LP1: mov dl,[bx]
Cmp dl, '$'
Je exit
Inc bx
;insert code here
mov ah,02
int 21h
jmp lp1
Exit: Mov ax, 4c00h
Int 21h
Main endp
End main
```

2. Save the program as **logic.asm**
3. Assemble and execute the program.

```
C:\HOA_5.1>tasm logic.asm
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International

Assembling file:   logic.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  491k


C:\HOA_5.1>tlink logic.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
```

4. Analyze the output and record the output in Table 5.1

Table 5.1 – Output of logic.asm

```
C:\HOA_5.1>logic
Proud to be TIPians
```

Sample Problem 2:

1. Modify program logic.asm.
2. Replace the line "**; insert code here**", with "**and dl, 11011111B**" .
3. Save the program as **and.asm**.
4. Assemble and execute the program.

```
C:\HOA_5.1>tasm and.asm
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International

Assembling file:   and.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  491k


C:\HOA_5.1>tlink and.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
```

5. Observe and record the output in Table 5.2

Table 5.2 – Output of and.asm

```
C:\HOA_5.1>and
PROUD TO BE TIPIANS
```

6. How is your output different from before? Why?

- My output is different from before in a way that the string "Proud to be TIPians" in logic.asm is now outputted as all uppercase letters in and.asm. This is because of the added code which is used to convert any lowercase letter in the register dl to its uppercase equivalent.

Sample Problem 3:

1. Modify logic.asm again, this time replace the line "**;insert code here**", with "**xor dl, 00100000B**".
2. Save the program as **xor.asm**.
3. Assemble and execute the program.

```
C:\HOA_5.1>tasm xor.asm
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International

Assembling file:   xor.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  491k


C:\HOA_5.1>tlink xor.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
```

4. Observe and record the output in Table 5.3.

Table 5.3 – Output of xor.asm

```
C:\HOA_5.1>xor
pROUD TO BE tipIANS
```

How is your output different from before? Why?

- The output here differs from the output of logic.asm since the case of each alphabetic character is toggled. Uppercase letters become lowercase, and lowercase letters become uppercase. This is because the xor instruction was used to toggle the case of each alphabetic character.

Sample Problem 4:

1. Modify logic.asm once again, this timeplace the line "**;insert code here**", with "**or dl, 00100000B**".
2. Save the program as **or.asm.**
3. Assemble and execute the program.

```
C:\HOA_5.1>tasm or.asm
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990 Borland International

Assembling file:   or.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  491k


C:\HOA_5.1>tlink or.obj
Turbo Link  Version 3.0 Copyright (c) 1987, 1990 Borland International
```

4. Observe and record the output in Table 5.4.

Table 5.4 – Output of or.asm

```
C:\HOA_5.1>or
proud to be tipians
```

5. How is your output different from before? Why?

- The output is different from before because this time the string output is all in lowercase which is opposite of what was outputted when using 'and' in Sample Problem 2. This is because the or instruction was used with 00100000B

B. Supplementary Activity: Output(s) and Observation(s)

1. Write an assembly program that will simulate the given Boolean expression using assembly programming.

   AL = (AH· BH + AL· BL)' xor (CL+(CH· DH)' · DL)'

Assembly Program (.asm was screenshotted on VSCode for better readability)

HOA 5.1 - Logical Instructions > suppAct1.asm

```
.model small
.stack 100h

.data
    ; initial test values (these can be changed)
    ah_val  db  10101010b   ; example value for ah
    bh_val  db  11001100b   ; example value for bh
    al_val  db  11110000b   ; example value for al
    bl_val  db  00001111b   ; example value for bl
    ch_val  db  10101010b   ; example value for ch
    dh_val  db  01010101b   ; example value for dh
    cl_val  db  11110000b   ; example value for cl
    dl_val  db  00001111b   ; example value for dl
    result  db  ?           ; variable to store final result

.code
main proc
    mov ax, @data
    mov ds, ax

    ; load test values into registers
    mov ah, ah_val
    mov bh, bh_val
    mov al, al_val
    mov bl, bl_val
    mov ch, ch_val
    mov dh, dh_val
    mov cl, cl_val
    mov dl, dl_val

    ; part 1: calculate (ah·bh + al·bl)'
    ; store ah value temporarily
    push ax

    ; calculate ah·bh
    mov al, ah      ; move ah to al for and operation
    and al, bh      ; al = ah·bh
    mov bh, al      ; store ah·bh in bh temporarily

    ; restore ah and calculate al·bl
    pop ax
    push ax         ; save ax again
    and al, bl      ; al = al·bl

    ; perform or operation: al = ah·bh + al·bl
    or al, bh       ; al = (ah·bh) + (al·bl)

    ; perform not operation: al = (ah·bh + al·bl)'
```

```
49      not al          ; al = (ah·bh + al·bl)'
50
51      ; store the result temporarily
52      mov bl, al
53
54      ; part 2: calculate (cl+(ch·dh)'·dl)'
55      mov al, ch
56      and al, dh      ; al = ch·dh
57      not al          ; al = (ch·dh)'
58
59      ; calculate (ch·dh)'·dl
60      and al, dl      ; al = (ch·dh)'·dl
61
62      ; perform or operation with cl
63      or al, cl       ; al = cl+(ch·dh)'·dl
64
65      ; perform not operation
66      not al          ; al = (cl+(ch·dh)'·dl)'
67
68      ; part 3: xor the two results
69      xor al, bl      ; al = (ah·bh + al·bl)' xor (cl+(ch·dh)'·dl)'
70
71      ; store the final result
72      mov result, al
73
74      ; exit program
75      mov ax, 4c00h
76      int 21h
77  main endp
78  end main
```

* Note: The program outputs nothing since the final result was only stored in 'result' variable and there was not code implemented that lets it output the result of the sample values indicated *

2. Give a sample problem where the logical instructions can be applied.

- In bitmap graphics processing, logical instructions form the foundation of pixel manipulation. For instance, when modifying an image, you can use AND operations with specific masks to isolate color components while preserving others. By applying OR operations with new color values, you can blend colors without affecting preserved bits. XOR operations are particularly useful for creating visual effects or toggling specific pixel attributes. This technique is fundamental in game development, image processing applications, and UI rendering where individual bits represent color channels or transparency values.
- Link for more information on bitmap graphics processing: https://pixinsight.com/developer/pcl/doc/html/group__bitmap__bitwise__ops.html

C. Conclusion & Lessons Learned

Upon completing the hands-on activity, I have developed a better understanding of various logical instructions and their applications. I was able to effectively compare different logical instructions, identifying their unique characteristics and

appropriate use cases by doing the procedures in the activity. Additionally, I have gained practical experience by creating a program in assembly that incorporate these logical instructions when given a Boolean expression. In conclusion, I was able to perform the tasks required with the help of the procedures as my guide especially for the part wherein I needed to write an assembly program to perform the given Boolean expression.