Experiment N° 5
Arithmetic and Logical Instructions

Introduction:
In this experiment, you will be introduced to the logic instructions of the 8086 family of processors. You will also deal with the conversion of numbers from one radix to another.
Objectives:

1- Logic Instructions
2- Base conversion

References:

- Textbook:
- Lecture notes.

Arithmetic Instructions:
The following table (Table 5. 1) summarizes the arithmetic instructions used with the 8086 microprocessor. It also shows the effect of each instruction, a brief example, and the flags affected by the instruction. The "*" in the table means that the corresponding flag may change as a result of executing the instruction. The "-" means that the corresponding flag is not affected by the instruction, whereas the "?" means that the flag is undefined after executing the instruction.

Type	Instruction	Example	Meaning	Flags Affected					
				OF	SF	ZF	AF	PF	CF
Addition	ADD	ADD AX,7BH	AX ← AX + 7B	*	*	*	*	*	*
	ADC	ADC AX,7BH	AX ← AX + 7B +CF	*	*	*	*	*	*
	INC	INC [BX]	[BX]←[BX]+1	*	*	*	*	*	-
	DAA	DAA		?	*	*	*	*	*
Subtraction	SUB	SUB CL,AH	CL ← CL – AH	*	*	*	*	*	*
	SBB	SBB CL,AH	CL ← CL – AH – CF	*	*	*	*	*	*
	DEC	DEC DAT	[DAT] ← [DAT] – 1	*	*	*	*	*	-
	DAS	DAS		?	*	*	*	*	*
	NEG	NEG CX	CX ← 0 – CX	*	*	*	*	*	*
Multiplication	MUL	MUL CL MUL CX	AX ← AL * CL (DX,AX) ← AX* CX	*	?	?	?	?	*
	IMUL	IMUL BYTE PTR X IMUL WORD PTR X	AX ← AL * [X] (DX,AX) ← AX* [X]	*	?	?	?	?	*
Division	DIV	DIV WORD PTR X	AX ← Q(([DX,AX])/[X]) DX ←R(([DX,AX])/[X])	?	?	?	?	?	?
	IDIV	IDIV BH	AL ← Q(AX/BH) AH ←R(AX/BH)	?	?	?	?	?	?
Sign Extension	CBW	CBW	AH ← MSB(AL)	-	-	-	-	-	-
	CWD	CWD	DX ← MSB(AX)	-	-	-	-	-	-

Table 5. 1:: Summary of Arithmetic Instructions of the 8086 microprocessor

Notes:

The DAA (Decimal Adjust after Addition) instruction allows addition of numbers represented in 8-bit packed BCD code. It is used immediately after normal addition instruction operating on BCD codes. This instruction assumes the AL register as the source and the destination, and hence it requires no operand. The effect of DAS (Decimal Adjust after Subtraction) instruction is similar to that of DAA, except the fact that it is used after performing a subtraction.

CBW and CWD are two instructions used to facilitate division of 8 and 16 bit signed numbers. Since division requires a double-width dividend, CBW converts an 8-bit signed number (in AL), to a word, where the MSB of AL register is copied to AH register. Similarly, CWD converts a 16-bit signed number to a 32-bit signed number (DX,AX).

Logical Instructions:

Logic shift and rotate instructions are called bit manipulation operations. These operations are designed for low-level operations, and are commonly used for low-level control of input/output devices. The list of the logic operations of the 8086 is given in Table 5.1, along with examples, and the effect of these operations on the flags. The "*" in the table means that the corresponding flag may change as a result of executing the instruction. The "-" means that the corresponding flag is not affected by the instruction, whereas the "?" means that the flag is undefined after executing the instruction.

Instruction	Example	Meaning	Flags					
			OF	SF	ZF	AF	PF	
AND	AND AX, FFDFH	AX ← AX AND FFDFH	0	*	*	?	*	
OR	OR AL, 20H	AL ← AL OR 20H	0	*	*	?	*	
XOR	XOR NUM1, FF00	[NUM1]←[NUM1]XOR FF00	0	*	*	?	*	
NOT	NOT NUM2	[NUM2]←$\overline{[\text{NUM2}]}$	-	-	-	-	-	

Table 5.2: Summary of the Logic Instructions of the 8086 Microprocessor

The logic operations are the software analogy of logic gates. They are commonly used to separate a bit or a group of bits in a register or in a memory location, for the purpose of testing, resetting or complementing. For example, if b is the value of a certain bit in a number. The related effects of the basic operations on a single bit are indicated in Table 5.3:

Operation	Effect
b AND 0 = 0	Reset the bit
b OR 1 = 1	Set the bit
b XOR 1 = b	Complement the bit
b XOR 0 = b	-

Table 5.3: Effects on bits of the basic logic instructions

Byte manipulations for reading and displaying purposes:

1 / To put two decimal digits into the same byte use the following:

```
MOV AH, 01H
INT 21H
SUB AL, 30H
MOV CH, AL          ; Read high digit       e.g. 8

MOV AH, 01H
INT 21H
SUB AL, 30H
MOV CL, AL          ; Read low digit        e.g. 3

MOV AL, 10000B      ; Use MUL by 10000B to shift left by 4 bits
MUL CH              ; Shift AL 4 bits to the left
XOR AH, AH          ; Clear AH
OR AL, CL           ; Result in AL ← 83
```

If we want to perform addition:

```
; If AL  contains the first number in BCD format
; and CL  contains the second number in BCD format
ADD AL, CL
DAA                 ; Decimal adjust
; New result in AL in BCD format
MOV CL, AL
; Number in CL register. See next how to display it as decimal number.
```

2 / To display a number in BCD format use the following:

```
; The number is in the CL register:
MOV AL, CL          ; Move CL to AL
XOR AH, AH          ; Clear AH
MOV BL, 10000B
DIV BL              ; Shift  AX 4 bits to the right
AND AL, 0FH         ; Clear 4 high nibbles of AL
ADD AL, 30H         ; Convert to character

; Now Display AL as high digit first
MOV AL, CL          ; Read number again
AND AL, 0FH         ; Clear 4 high nibbles of AL
ADD AL, 30H         ; Convert to character
; Now Display AL as low digit second
```

Displaying Data in any Number Base r:

The basic idea behind displaying data in any number base is division. If a binary number is divided by 10, and the remainder of the division is saved as a significant digit in the result, the remainder must be a number between zero and nine. On the other hand, if a number is divided by the radix r, the remainder must be a number between zero and (r-1). Because of this, the resultant remainder will be a different number base than the input which is base 2. To convert from binary to any other base, use the following algorithm.

Algorithm:

1. Divide the number to be converted by the desired radix (number base r).
2. Save the remainder as a significant digit of the result.
3. Repeat steps 1 and 2 until the resulting quotient is zero.
4. Display the remainders as digits of the result.
   Note that the first remainder is the least significant digit, while the last remainder is the most significant one.

Pre Lab Work:

1. Study program 5.2, and explain how base conversion is performed?
2. Write, assemble and link program 5.1. You will run it in the lab using CodeView.
3. Write, assemble, link and run program 5.2.
4. Modify the program so that it prompts the user for the RADIX and the number NUM to be converted. Call the new program prog-5.3.
5. Write a program that converts from decimal to hexadecimal. Name it Prog-5.4.
6. Bring your work to the lab.

Lab Work:

1- Use CodeView to trace program 5.1. Fill in table 5.3. Notice any changes in the status flags, and explain them.
2- Run program 5.2, and see what value is displayed.
3- Change the value of the variable NUM and see the output value.
4- Now change the value of RADIX and see the value displayed.
5- Write a program that prompts the user to enter two numbers of 4 digits each. Converts these numbers to hexadecimal. Then calculates the sum, the difference of the two numbers, and finally displays the result in decimal format. Name it program 5.5.
6- Show all your work to the instructor.
7- Submit all your work at the end of the lab session.

Lab Assignment:

Write a program that reads two binary numbers of 8 digits each, stores them inside the internal registers. Multiply the two numbers using a simple MUL operation, and display the result in decimal format.

To ease bit manipulation and shifting, use division and multiplication by 2, to perform right shift and left shift.

TITLE "Program 5.1: Logic Instructions"

```
; This program shows the effect of the logic instructions

.MODEL SMALL
.STACK 200
.DATA
            NUM1 DW     0FA62H
            NUM2 DB     94H
.CODE
.STARTUP

            MOV   AX,  NUM1          ;load AX with number NUM1
            AND   AX,  0FFDFH        ;Reset 6th bit of AX
            OR    AL,  20H           ;Set 6th bit of AL
            XOR   NUM1, 0FF00H       ;Complement the high order byte of
                                  ; NUM1
            NOT NUM2                 ;Complement NUM2
            XOR AX, AX               ;Clear AX
            MOV AX, NUM1
            AND AX, 0008H            ; Isolate bit 4 of NUM1
            XOR AX, 0080H            ;Complement 4th bit of AX
.EXIT
END
```

Fill in table 5.3 while running the above program using CodeView.

Statement	**Destination Content**		**Status Flags**							
	Before	**After**	OF	DF	IF	SF	ZF	AF	PF	CF
1. MOV AX, NUM1										
2. AND AX, 0FFDFH										
3. OR AL, 20H										
4. XOR NUM1, 0FF00H										
5. NOT NUM2										
6. XOR AX, AX										
7. MOV AX, NUM1										
8. AND AX, 0008H										
9. XOR AX, 0080H										

Table 5.4: Effects of Executing Program 5.1

```
TITLE "Lab Exp. # 5 Program # 5.2"
; This program converts a number NUM from Hexadecimal,
; to a new numbering base (RADIX).

.MODEL SMALL
.STACK 200
.DATA
                RADIX DB 10                ;radix: 10 for decimal
                NUM   DW 0EFE4H            ;the number to be converted
                                           ;put here any other number.
                ;Note that: 0EFE4H = 61412 (base 10)
                TEMP  DB    10 DUP(?)      ;Used to simulate a stack

.CODE
.STARTUP
        MOV   AX, NUM              ;load AX with number NUM
                                   ;display AX in decimal
        MOV   CX, 0                ;clear digit counter
        XOR   BH, BH               ;clear BH
        MOV   BL, RADIX            ;set for decimal
        XOR   SI, SI               ;Clear SI register
DISPX1:
        MOV   DX, 0000             ;clear DX
        DIV   BX                   ;divide DX:AX by 10
        MOV   TEMP[SI], DL         ;save remainder
        INC   SI
        INC   CX                   ;count remainder
        OR    AX, AX               ;test for quotient of zero
        JNZ   DISPX1               ;if quotient is not zero
        DEC   SI

DISPX2:
        MOV   DL, TEMP[SI]         ;get remainder
        MOV   AH, 06H              ;select function 06H
        ADD   DL, 30H              ;convert to ASCII
        INT   21H                  ;display digit
        DEC   SI
        DEC   CX                   ;repeat for all digits
        JNZ   DISPX2
.EXIT                              ;exit to dos
END
```