

EXPERIMENT 3

Aim : Assembly program to display the contents of the flag register.

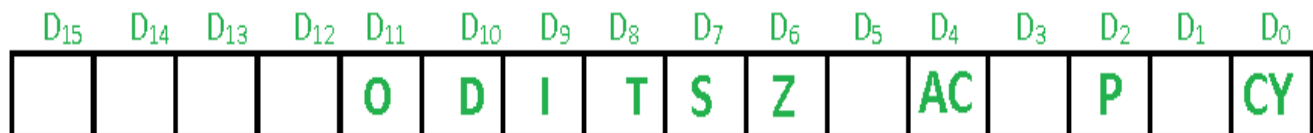
Theory :

➤ Flag Registers of 8086

The **flag register in 8086** is a 16-bit register that indicates the current state of the processor and provides control over its operation. It is divided into two categories: **status flags** and **control flags**.

Structure of the Flag Register

The 16-bit flag register is structured as follows:



1. **Status Flags:** Reflect the outcomes of operations.
2. **Control Flags:** Control the operation of the processor.

Status Flags

1. **Carry Flag (CF) (Bit 0):**
 - Set if there is a carry out from the MSB (Most Significant Bit) after an arithmetic operation.
 - Used for unsigned arithmetic.
 - Example: $255(1111\ 1111) + 1(0000\ 0001) = 256\ (1\ 0000\ 0000)$, sets CF.
2. **Parity Flag (PF) (Bit 2):**
 - Set if the number of set bits in the result is even.
 - Indicates even or odd parity.
 - Example: $127(0111\ 1111) + 2(0000\ 0010) = 129\ (1000\ 0001)$, sets PF.[unsigned]

3. **Auxiliary Carry Flag (AF)** (Bit 4):

- Set if there is a carry out from the lower nibble (4 bits) to the upper nibble.
- Used in Binary-Coded Decimal (BCD) arithmetic.
- Example: $127(0111\ 1111) + 2(0000\ 0010) = 129(1000\ 0001)$, sets AF.[unsigned]

4. **Zero Flag (ZF)** (Bit 6):

- Set if the result of an operation is zero.
- Example: $255(1111\ 1111) + 1(0000\ 0001) = 256(1\ 0000\ 0000)$, sets ZF.

5. **Sign Flag (SF)** (Bit 7):

- Set if the result of an operation is negative (MSB is 1).
- Indicates the sign of a result (0 = positive, 1 = negative).
- Example: $127(0111\ 1111) + 2(0000\ 0010) = 129(1000\ 0001)$, sets SF.[unsigned]

6. **Overflow Flag (OF)** (Bit 11):

- Set if there is a signed overflow (the result exceeds the range for signed numbers).
- Range for 16 bit numbers is [-128 to 127] or [-80 to 7F] (in hexadecimal).
- Example: $127(0111\ 1111) + 1(0000\ 0001) = -128(1000\ 0000)$, sets OF.[signed]

Control Flags

1. **Direction Flag (DF)** (Bit 10):

- Controls string operations (increment or decrement).
- **Set (1):** Strings are processed from high memory to low memory.
- **Clear (0):** Strings are processed from low memory to high memory.

2. **Interrupt Flag (IF)** (Bit 9):

- Controls the processor's response to maskable interrupts.
- **Set (1):** Enables interrupts.
- **Clear (0):** Disables interrupts.

3. **Trap Flag (TF)** (Bit 8):

- Used for debugging purposes.
- **Set (1):** Enables single-step mode.
- Processor generates an interrupt after executing each instruction.

Unused and Reserved Bits

- Bits 1,3, 5, 12–15 are unused or reserved in the 8086.
-

➤ Instructions used in the program :

Data Segment :

1. **msg db 0dh,0ah,"-- -- -- -- OF DF IF TF SF ZF -- AF -- PF -- CF \$"**
 - Defines a string msg to display a labeled layout of the flags.
 - 0dh, 0ah are carriage return (CR) and line feed (LF) for a new line.
2. **newl db 0dh,0ah,"\$"**
 - Defines a string for a new line (\$ marks the end of the string).
3. **flag dw ?**
 - Defines a word (16 bits) to temporarily store the flag register's value.

Code Segment :

Initialization

1. **assume CS:Code, DS:Data**
 - Tells the assembler to assume that CS (Code Segment) is labeled Code and DS (Data Segment) is labeled Data.
2. **mov ax, Data**
 - Loads the starting address of the Data segment into the AX register.
3. **mov DS, ax**
 - Loads the AX value into the DS register, establishing the Data segment for use.

Displaying Messages

4. **mov dx, offset msg**
 - Loads the address of the msg string into the DX register.
5. **mov ah, 09h**
 - Sets up the DOS interrupt to display a string.

6. int 21h

- Triggers the DOS interrupt to display the string at the address in DX.

Manipulating Flags

7. cli

- Clears the Interrupt Flag (IF), disabling interrupts.

8. stc

- Sets the Carry Flag (CF).

9. std

- Sets the Direction Flag (DF) to decrement mode.

10. pushf

- Pushes the current flag register onto the stack.

11. pop bx

- Pops the flag register value from the stack into the BX register.

12. mov flag, bx

- Moves the value of BX (flag register) into the variable flag.

Processing and Displaying Flag Values

13. mov cx, 16

- Loads the loop counter CX with 16 (number of flags).

14. mov bx, 8000h

- Initializes BX with the MSB (Most Significant Bit) mask for flags.

15. loops:

- Label marking the start of the loop.

16. mov ax, flag

- Loads the flag register value into AX.

17. and ax, bx

- Performs a bitwise AND between AX and BX to isolate a specific flag bit.

18. jz zero

- Jumps to the zero label if the result of the AND operation is zero (flag bit not set).

19. mov dl, 31h

- Loads DL with ASCII code for 1 (flag set).

20. mov ah, 02h

- Sets up DOS interrupt for displaying a character.

21. int 21h

- Triggers the interrupt to display the character in DL.

22. jmp space

- Jumps to the space label to add spacing.

23. zero:

- Label for handling a zero bit.

24. mov dl, 30h

- Loads DL with ASCII code for 0 (flag not set).

25. space:

- Adds space between the displayed bits.

26. ror bx, 1

- Rotates the bits in BX one position to the right to check the next flag.

27. loop loops

- Decrements CX and jumps back to loops if CX is not zero.

Program Termination

28. mov ah, 4ch

- Sets up DOS interrupt for program termination.

29. int 21h

- Ends the program execution.

➤ Interrupts used in the program :

Interrupt 21h

int 21h is a software interrupt in DOS that provides access to many system services, including input/output operations, memory management, and process control. The behavior of int 21h is determined by the value stored in the **AH** register, which specifies the function to be performed. Below, we focus on the **AH values** used in the code.

1. int 21h with AH = 09h (Display String)

- **Purpose:** This interrupt function displays a string to the console.
- **Input:** The address of the string (terminated by a \$ character) must be placed in the **DX** register.
- **Behavior:**
 - The string is printed character by character, and the process stops when a \$ character is encountered.

Code in the program:

```
mov dx, offset msg    ; Load address of msg into DX
mov ah, 09h           ; Set function code to 09h (Display String)
int 21h               ; Call DOS interrupt to display string
```

- **Explanation:** In the program, msg contains a string that lists the flags' names. The program loads the address of msg into the **DX** register and then calls int 21h with AH = 09h. This causes the string to be displayed on the screen.

2. int 21h with AH = 02h (Display Character)

- **Purpose:** This interrupt function is used to display a single character to the console.
- **Input:** The character to be displayed is placed in the **DL** register.
- **Behavior:**
 - The character specified in **DL** is printed to the screen.

Code in the program:

```
mov dl, '1'           ; Load character '1' into DL (flag is set)
mov ah, 02h           ; Set function code to 02h (Display Character)
int 21h               ; Call DOS interrupt to display character
```

- **Explanation:** This instruction is used throughout the program to display 1 or 0 for each flag. The character is stored in **DL**, and then int 21h with AH = 02h is called to print the character on the screen.

3. int 21h with AH = 4Ch (Terminate Program)

- **Purpose:** This interrupt function is used to terminate the program and return control to the operating system.
- **Input:** The **AL** register contains the exit code. If **AL** is set to 00h, the program terminates normally.
- **Behavior:**
 - The program is terminated, and the DOS environment regains control. If any value other than 00h is in **AL**, it is returned as the exit code.

Code in the program:

```
mov ah, 4Ch      ; Set function code to 4Ch (Terminate Program)
int 21h          ; Call DOS interrupt to terminate the program
```

- **Explanation:** This instruction is used at the end of the program to terminate the execution after displaying the flags. It ensures that control is returned to DOS.

➤ Before implementing Implied Instructions (STD/STC/CLI)

Code :

Data Segment

```
msg db 0dh,0ah,&quot;-- -- -- -- OF DF IF TF SF ZF -- AF -- PF -- CF $&quot;;
newl db 0dh,0ah,&quot;$&quot;;
flag dw ?
```

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
    mov dx,offset newl
    mov ah,09h
    int 21h
```


➤ After implementing Implied Instructions (STD/STC/CLI)

Data Segment

```
msg db 0dh,0ah,&quot;-- -- -- -- OF DF IF TF SF ZF -- AF -- PF -- CF $&quot;;  
newl db 0dh,0ah,&quot;$&quot;;  
flag dw ?
```

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  
    mov dx,offset newl  
    mov ah,09h  
    int 21h  
    cli  
    stc  
    std  
    pushf  
    pop bx  
    mov flag,bx  
    mov cx,16  
    mov bx,8000h  
    loops:  
        mov ax,flag  
        and ax,bx  
        jz zero  
        mov dl,31h  
        mov ah,02h  
        int 21h  
        jmp space  
    zero: mov dl,30h  
        mov ah,02h  
        int 21h  
    space: mov dl,&#39; &#39;  
        mov ah,02h  
        int 21h  
        mov ah,02h
```

```
int 21h
ror bx,1
loop loops
mov ah,4ch
int 21h
Code ends
end start
```

Output :



```
D:\TASM>viraj.exe
```

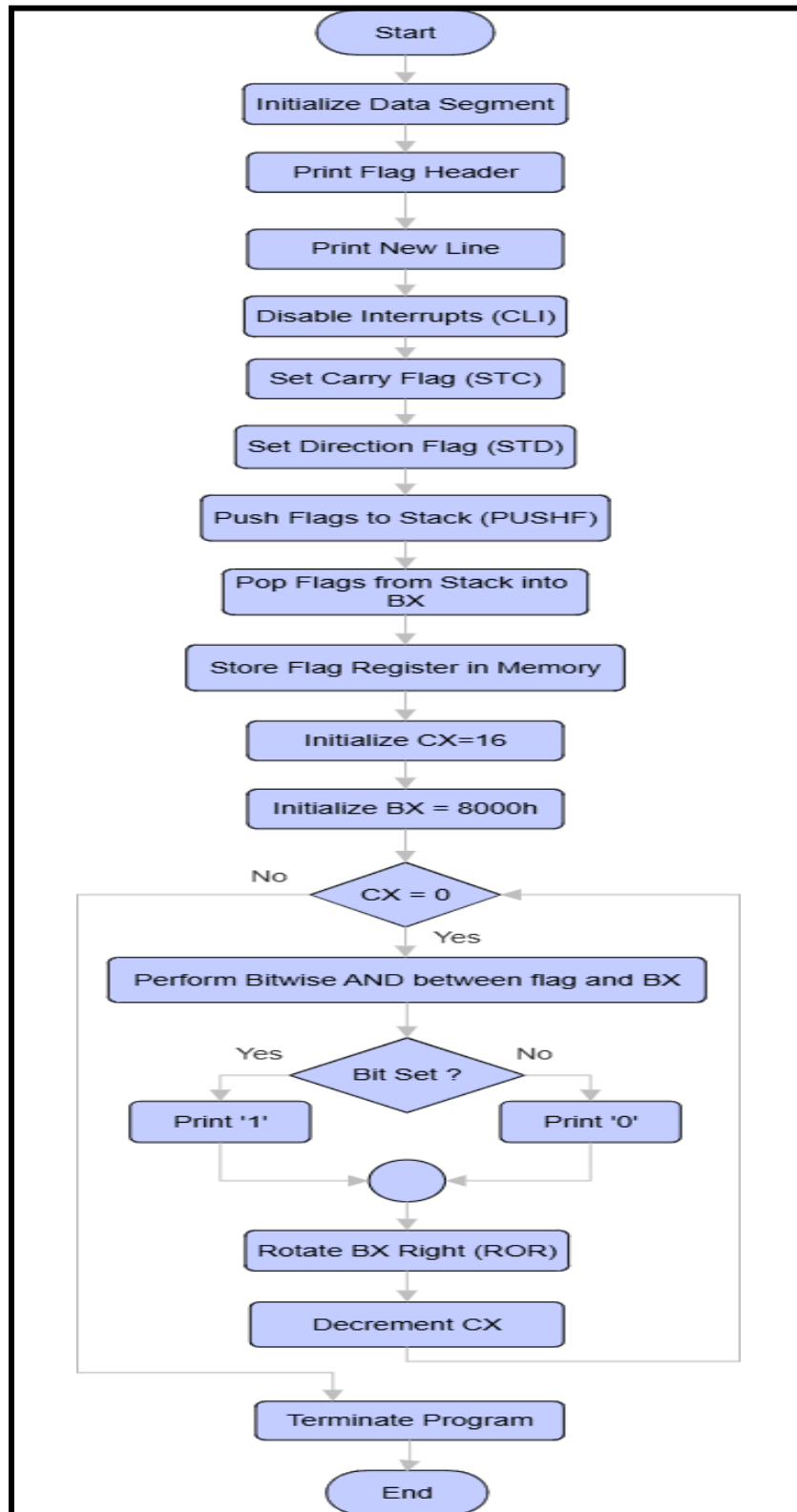
OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	1	0	0	0	0	0	0	1

➤ Algorithm :

1. Initialize the data segment by loading the 'Data' segment address into 'AX' and moving it into 'DS'.
2. Load the address of the 'msg' string into 'DX' and use 'INT 21h' with 'AH = 09h' to print the flag header.
3. Load the address of 'newl' into 'DX' and use 'INT 21h' with 'AH = 09h' to print a newline for formatting.
4. Disable interrupts using the 'CLI' instruction.
5. Set the carry flag using the 'STC' instruction.
6. Set the direction flag using the 'STD' instruction.
7. Push the flag register onto the stack using 'PUSHF'.
8. Pop the flag register value from the stack into 'BX'.
9. Store the flag register value in the memory variable 'flag'.
10. Initialize 'CX' with 16 to loop through all flag bits.
11. Initialize 'BX' with '8000h' to check the most significant bit first.
12. Perform a bitwise AND operation between 'flag' and 'BX' to check if the current flag bit is set.
13. If the result is zero, print '0'; otherwise, print '1'.

14. Rotate 'BX' right by one position using 'ROR' to move to the next flag bit.
15. Repeat the loop until all 16 bits of the flag register are processed.
16. Terminate the program using 'INT 21h' with 'AH = 4Ch' to return control to DOS.

Flowchart :



➤ **Conclusion :**

The provided assembly program demonstrates the functionality of the **8086 flag register** by displaying its 16 bits as a binary sequence, where each bit corresponds to a specific flag (e.g., Carry Flag, Zero Flag, etc.).

Key points observed:

1. The program uses **bitwise operations** to isolate and process individual flags, showcasing efficient manipulation of binary data.
2. It employs **DOS interrupts** for output operations, demonstrating basic I/O techniques in assembly language.
3. By explicitly setting and clearing flags like **CF** and **DF**, the program illustrates how the 8086 CPU modifies and uses flags during execution.
4. The use of a loop and mask to traverse all 16 bits of the flag register highlights the systematic approach for binary flag processing.

Overall, this program provides a practical example of low-level flag manipulation and bitwise computation, laying the foundation for understanding CPU status registers and their role in decision-making and program control in assembly programming.

➤ **Post Experiment :**

(Take 03*05+06-04 expression and compare the flag registers before and after the execution of the expression)

Code :

Data Segment

```
msg db 0dh,0ah,&quot;-- -- -- -- OF DF IF TF SF ZF -- AF -- PF -- CF $&quot;;  
newl db 0dh,0ah,&quot;$&quot;;  
flag dw ?
```

Data ends

Code Segment

```
assume CS:Code,DS:Data  
start:  
    mov ax,Data
```

NAME : VIRAJ KRISHNAJI PRABHU
MICROPROCESSOR

CLASS/SECTION :SE-CMPN-C
ROLL NO :16

```
mov DS,ax
mov dx,offset msg
mov ah,09h
int 21h
mov dx,offset newl
mov ah,09h
int 21h
mov al,03h
mov bl,05h
mul bl
add ax,06h
sub ax,04h
mov flag,ax
pushf
pop bx
mov flag,bx
mov cx,16
mov bx,8000h
loops:
    mov ax,flag
    and ax,bx
    jz zero
    mov dl,31h
    mov ah,02h
    int 21h
    jmp space
zero: mov dl,30h
    mov ah,02h
    int 21h
space: mov dl,' ';
    mov ah,02h
    int 21h
    mov ah,02h
    int 21h
    ror bx,1
loop loops
mov ah,4ch
int 21h
Code ends
end start
```

Output :

Before the execution :

```
D:\TASM>viraj.exe

--  --  --  --  OF  DF  IF  TF  SF  ZF  --  AF  --  PF  --  CF
0   1   1   1   0   0   1   0   0   0   0   0   0   0   1   0
```

After the execution :

```
D:\TASM>viraj.exe

--  --  --  --  OF  DF  IF  TF  SF  ZF  --  AF  --  PF  --  CF
0   1   1   1   0   0   1   0   0   0   0   0   0   1   1   0
```

➤ Inference :

Expression Execution:

- The arithmetic expression $(03 * 05) + 06 - 04$ evaluates to 11h (17 in decimal), represented in binary as 0001 0001b.

Flag Status Analysis:

- **Parity Flag (PF):**
 - **Set (1)** because the least significant byte (0001 0001) has an even number of 1s (2 ones).
- **Zero Flag (ZF):**
 - **Cleared (0)** because the result is non-zero.
- **Sign Flag (SF):**
 - **Cleared (0)** because the result is positive (most significant bit is 0).
- **Carry Flag (CF):**
 - **Cleared (0)** because no carry was generated during the arithmetic operations.
- **Overflow Flag (OF):**
 - **Cleared (0)** as there was no signed overflow.