

Processor Status & FLAGS Register

Course Code: COE 3205

Course Title: Computer Organization & Architecture



Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	05	Week No:	05	Semester:	Fall 24-25
Lecturer:	<i>Noboranjana Dey; noboranjana@aiub.edu</i>				

Lecture Outline



1. **Overview**
2. **Learning Objective**
3. **The FLAGS Register**
4. **The Status Flags**
5. **Overflow**
6. **How Instructions Affect the Flags**
7. **DEBUG Program**

Overview



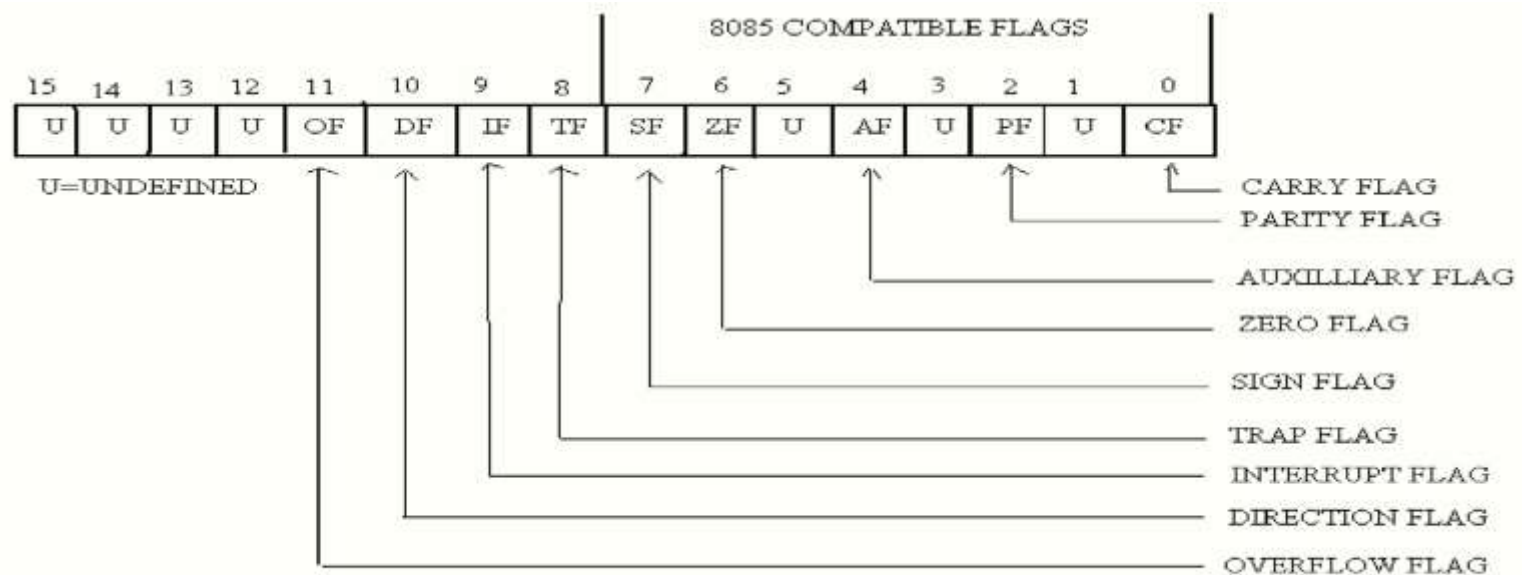
- **Computer's decision-making ability makes it unique than other devices.**
- **The CPU circuits perform decision making based on the processor's current state.**
- **The 8086 processor's state is represented with nine individual bits or flags.**
- **The 8086 takes decision based on the flags value.**
- **The flags are placed in the FLAGS register.**
- **Status flags: Reflects the result of computation.**
- **Control flags: used to enable or disable certain operations of processor**

Learning Objective



- **How flags are effected by the machine instructions.**
- **DOS program DEBUG**
- **Display registers, flags and memory locations using DEBUG.**

The FLAGS Register



Status Flags: bit 0, 2, 4, 6, 7 and 11

Control Flags: bit 8, 9 and 10

***** bit 1,3,5,12,13,14,15 has no significance**



The Status Flags (1/2)

Flags	Descriptions
Carry Flag	<ul style="list-style-type: none">• CF=1 If there is a carry out from the most significant bit (MSB) on addition.• CF=1 If there is a borrow into the most significant bit MSB on subtraction.• Otherwise, CF=0
Parity Flag	<ul style="list-style-type: none">• Even Parity: PF=1, if low byte of a result has even number of one bits [e.g., 1111000011]• Odd Parity: PF=0, if low byte of a result has an odd number of one bits [e.g., 1111000111]• e.g FFFF is odd parity and FFFE is even parity
Auxiliary carry Flag	<ul style="list-style-type: none">• AF=1, if there is a carry out from bit 3 on addition or• AF=1, if there is a borrow from bit 3 on subtraction.

The Status Flags (2/2)



Zero Flag	<ul style="list-style-type: none">• ZF=1 for a zero result. e.g. for $AX-AX= \text{Zero}$, thus, ZF=1<ul style="list-style-type: none">• ZF=0 for a non-zero result
Sign Flag	<ul style="list-style-type: none">• SF=1 if MSB of a result is 1. that is the result is negative<ul style="list-style-type: none">• SF=0 if the MSB is zero
Overflow Flag	<ul style="list-style-type: none">• OF=1 if signed overflow occurred otherwise it is 0 (Zero)

Overflow



- The range of numbers can be represented is limited!
 - Range of **signed** numbers can be represented in 16-bit word is **-32768 to 32767**
 - For 8 bit byte it is **-128 to 127**
 - For unsigned number, the range for **word** is **0 to 65535**.
 - For byte it is 0 to 255

- If an Operations falls **outside these range**, overflow occurs and truncated result that is saved and will be incorrect.

Overflow

Possible Overflows



- If we perform an arithmetic operation such as addition, there are four possible consequences:
 - **No overflow**
 - **Signed overflow only**
 - **Unsigned overflows only**
 - **Both signed and unsigned overflows**

Overflow

Example: Unsigned but not Signed Overflow



AX , FFFFh = 1111111111111111 [-1]

BX , 0001h = 0000000000000001 [+1]

=====

ADD AX,BX 1 0000 0000 0000 0000

- For an unsigned interpretation, the correct answer is 10000h or 65536. but this is out of range and 1 is carried out of **msb** and finally AX= 0000h[WRONG!] thus **UNSIGNED overflow occurred**.
- However, for a signed number the answer is correct FFFFh+0001h or -1+1=0. So **no signed overflow occurred**.

Overflow

Example: Signed but not Unsigned Overflow



AX 7FFFh = 0111111111111111

BX 7FFFh = 0111111111111111

=====

ADD AX,BX 1111111111111110 [FFFEh = -2]

- For signed and unsigned interpretation, 7FFFh= 32767 thus 7FFFh+7FFFh= 32767+32767 = 65534 [out of range for signed numbers!]. **So signed overflow occurred.**
- However, an **unsigned** interpretation the answer is correct FFFEh or 65534

Overflow

Processor Overflow Indication



- **OF = 1** for a **signed** overflow
- **CF = 1** for **unsigned** overflow

Unsigned Overflow:

- **On addition:** the unsigned overflow occurs when there is a carry out of the MSB.
Meaning the correct answer is bigger than largest unsigned number i.e. FFFFh or FFh
- **On Subtraction:** If there is a borrow into the MSB
Meaning the correct answer is Smaller than 0.

Overflow

Signed Overflow:



- **On addition:** On addition with the numbers with the same sign, signed overflow occurs while **SUM** has different sign. However, overflow is impossible in addition of numbers with different sign.
- **On subtraction:** Subtraction of numbers with different signs is like adding numbers of the same sign. Signed overflow occurs if the result has different sign than expected.
- **If there is a carry into msb but no carry out or if there is a carry out but no carry in then the signed overflow occurs and OF is set to 1.**

Overflow

Overflow Summary on Addition:



- Note that the overflow bit was set whenever we had a carry from bit 6 to bit 7, but no carry from bit 7 to C.
- It was also set when we had a carry from bit 7 to C, but no carry from bit 6 to bit 7.
- Upshot: The overflow bit is the EXCLUSIVE-OR of a carry from bit 6 to bit 7 and a carry from bit 7 to C.

Carry And Overflow

Example - 1

Dec	Hex
53	35
+25	+19
<u>78</u>	<u>4E</u>

Binary	
0011	0101
0001	1001
<u>01001110</u>	

bit 6 to 7 No carry
 bit 7 to C 1
 $\therefore C=0, O=0$

Example - 2

Dec	Hex
53	35
+91	+5B
<u>144</u>	<u>90</u>

Binary	
0011	0101
0101	1011
<u>10010000</u>	

bit 6 to 7 carry yes
 bit 7 to C carry NO carry
 $\therefore C=0, O=1$

Example - 3

Dec	Hex
53	35
-45	+D3
<u>8</u>	<u>108</u>

Binary	
0011	0101
1101	0011
<u>00001000</u>	

bit 6 to 7 carry yes
 bit 7 to C carry yes
 $\therefore C=1, O=0$

Example - 4

Dec	Hex
-98	0E
-58	+D3
<u>-143</u>	<u>171</u>

Binary	
1001	1110
1101	0011
<u>01110001</u>	

bit 6 to 7 No carry
 bit 7 to C carry
 $\therefore C=1, O=1$

How Instructions Affect the Flags



INSTRUCTION	AFFECT FLAGS
MOV/ XCHG	NONE
ADD/SUB	ALL
INC/DEC	ALL EXCEPT CARRY FLAG
NEG	ALL(CF =1 UNLESS RESULT 0 OF = 1 IF WORD OPERAND IS 8000H OR BYTE OPERAND IS 80H)

How Instructions Affect the Flags

Examples



1. **ADD AX,BX , WHERE AX CONTAINS FFFFh AND BX CONTAINS FFFFh.**

$$\begin{array}{r} \text{➤} \quad \text{FFFFh} \\ + \quad \text{FFFFh} \\ \hline \text{=====} \\ \text{1FFFEh} \end{array}$$

The result stored in AX is FFFEh = 1111 1111 1111 1110

- **SF = 1** because msb is 1
- **PF = 0** because there are 7 of 1 bits in the low byte of the result
- **ZF = 0** because the result is non zero
- **CF = 1** because there is a carry out of the msb on addition
- **OF = 0** because the sign of the stored result is the same as that of the numbers being added(as a binary addition, there is a carry into the msb and also a carry out)

How Instructions Affect the Flags

Examples



2. ADD AL, BL ,WHERE AL CONTAINS 80H AND BL CONTAINS 80H.

➤ 80h
 + 80h
 =====

1 00h

The result stored in AL is 00h.

- **SF = 0** because the msb is 0
- **PF = 1** because all the bits in the result are 0
- **ZF = 1** because the result is 0
- **CF = 1** because there is a carry out of the msb on the addition
- **OF = 1** because the numbers being added are both negative, but the result is 0(as a binary addition, there is no carry out into the msb but there is a carry out.)

How Instructions Affect the Flags

Examples



3. NEG AX, WHERE AX CONTAINS 8000H.



$$\begin{array}{rcl} 8000h & = & 1000\ 0000\ 0000\ 0000 \\ \text{one's complement} & = & 0111\ 1111\ 1111\ 1111 \\ \text{two's complement} & = & + 1 \\ & & \text{=====} \\ & & 1000\ 0000\ 0000\ 0000 = 8000h \end{array}$$

The result is stored in AX is 8000h

- SF = 1
- PF = 1
- ZF = 0
- CF = 1 because for NEG CF is always 1 unless the result is 0
- OF = ? because the result is 8000h; when a number is negated, we would expect a sign change, but because 8000h is its own two's complement, there is no sign change.

HW

Task : Solve the Following and Show the Effects on Flag Registers



- **SUB AX,BX , WHERE AX CONTAINS 8000H AND BX CONTAINS 0001H.**
- **INC AL, WHERE AL CONTAINS FFH.**
- **MOV AX, -5 .**

DEBUGING Program



- Debug is used to step through a program, display and change the registers and memory.
- It is possible to enter assembly code directly. DEBUG then converts it to machine code and stores it in memory.

DEBUGGING Program

Program 1: Checking Flags



```
.MODEL    SMALL
.STACK    100H
.DATA
.CODE
    MAIN PROC
    MOV AX,4000H    ;AX=4000h
    ADD AX,AX        ;AX=8000h
    SUB AX,0FFFFH    ;AX=8001h

    NEG AX            ;AX=7FFFh

    INC AX            ;AX=8000h

    MOV AH,4CH

    INT 21H

    MAIN ENDP

END MAIN
```

DEBUGGING Program

DEBUG Flag Symbols



Status Flag		Set (1) Symbol	Clear (0) Symbol
CF		CY (carry)	NC (no carry)
Pf		PE (even parity)	PO (odd parity)
AF		AC (auxiliary carry)	NA (No Auxiliary carry)
ZF		ZR (zero)	NZ (nonzero)
SF		NG (negative)	PL (plus)
OF		OV (overflow)	NV (no overflow}
Control Flag			
DF	DN (down)		UP (up)
IF	EI (enable interrupts)		DI (Disable interrupt)



Books

- Assembly Language Programing and Organization of the IBM PC

Ytha Yu
Charles Marut



References

- Flag Register Details
 - <https://www.youtube.com/watch?v=-LXc6lvtsfl>
 - https://www.youtube.com/watch?v=vkpGK5bZbSY&list=RDCMUCCU6xxwO9uJuFieylWL2ISA&start_radio=1&t=27
- Carry and Overflow Details
 - https://www.youtube.com/watch?v=9cXe_T99nL4