Computer Organization & Assembly Language
(Lecture 06)

Omar Bin Samin
Lecturer
Institute of Management Sciences, Peshawar.

Flags

Many instructions involve arithmetic and logical operations that changes the status of the flags, after which conditional instructions test the value of these status flags to take the control/ flow to other location

- Common flag bits are:
  - Overflow Flag - OF
  - Direction Flag- DF
  - Interrupt Flag- IF
  - Sign Flag - SF
  - Zero Flag - ZF
  - Auxiliary Flag - AF
  - Parity Flag - PF
  - Carry Flag - CF
  - Trap Flag - TF

Overflow Flag (OF)

- It indicates the overflow of a high-order bit (leftmost bit) of data after a signed arithmetic operation

- This flag is set to '1', when there is a **signed overflow**
  - **Example**

$$(1000\ 0000)_2 + (1000\ 0000)_2 = (0000\ 0000)_2$$

- This flag is set to '0', there is **no overflow**
  - **Example**

$$(1111\ 0011)_2 + (0000\ 0001)_2 = (1111\ 0100)_2$$

OF: Examples

$$\begin{array}{r} {\scriptstyle 1} \\ (1000\ 0000)_2 \\ +\ (1000\ 0001)_2 \\ \hline (0000\ 0001)_2 \end{array}$$

OF = 1

$$\begin{array}{r} (1000\ 0000)_2 \\ +(0000\ 0001)_2 \\ \hline (1000\ 0001)_2 \end{array}$$

OF = 0

OF: Example

$$
\begin{array}{rl}
 & {}^{2}(0001\ 0100\ 0000\ 1000)_2 \\
- & (1000\ 0100\ 0000\ 0000)_2 \\
\hline
 & (1001\ 0000\ 0000\ 1000)_2
\end{array}
$$

OF = 1

Direction Flag

- This flag is specifically used in string instructions
- If directional flag is set (1), then access the string data from higher memory location towards lower memory location (Reverse Direction)
  - **Syntax:** STD
- If directional flag is reset (0), then access the string data from lower memory location towards higher memory location (Forward Direction)
  - **Syntax:** CLD

Interrupt Flag

- It determines whether the external interrupts like keyboard entry, etc., are to be ignored or processed

- It disables the external interrupt when the value is reset (0) and enables interrupts when set (1)

Sign Flag (SF)

- It shows the sign of the result of an arithmetic or logical operation
- The sign is indicated by the leftmost bit
- A positive result clears the value of SF to ‘0’ and negative result sets it to ‘1’

SF: Examples

$$\begin{array}{r} (1000\ 0000)_2 \\ +\ (0000\ 0001)_2 \\ \hline (1000\ 0001)_2 \end{array}$$

SF = 1

$$\begin{array}{r} (0000\ 0000)_2 \\ +(0000\ 0001)_2 \\ \hline (0000\ 0001)_2 \end{array}$$

SF = 0

SF: Example

$$
\begin{array}{rl}
 & {}^{2}(0001\ 0100\ 0000\ 1000)_2 \\
- & (1000\ 0100\ 0000\ 0000)_2 \\
\hline
 & (1001\ 0000\ 0000\ 1000)_2
\end{array}
$$

SF = 1

Zero Flag (ZF)

- Indicates zero result
- This flag is set to ‘1’, when result is **zero**
  - **Example:** 2-2 = 0
- This flag is set to ‘0’, when result is **non- zero**
  - **Example**: 3-2 = 1

ZF: Examples

$$\begin{array}{r} (1000\ 0000)_2 \\ +\ (1000\ 0000)_2 \\ \hline (0000\ 0000)_2 \end{array}$$

ZF = 1

$$\begin{array}{r} (0000\ 0000)_2 \\ +(0000\ 0001)_2 \\ \hline (0000\ 0001)_2 \end{array}$$

ZF = 0

ZF: Example

$$
\begin{array}{rr}
 & (1000\ 0100\ 0000\ 1000)_2 \\
- & (1000\ 0100\ 0000\ 1000)_2 \\
\hline
 & (0000\ 0000\ 0000\ 0000)_2
\end{array}
$$

ZF = 1

Auxiliary Flag (AF)

- Indicates whether an operation produced a carry or borrow in the low-order 4 bits (nibble)
- For 16-bit and 32-bit values, only the least significant 8-bits are considered for computing auxiliary flag value

AF: Examples

$$\begin{array}{r} (0000\ 1000)_2 \\ +\ (0000\ 1001)_2 \\ \hline (0001\ 0001)_2 \end{array}$$

AF = 1

$$\begin{array}{r} (0000\ 0000)_2 \\ +(0000\ 0001)_2 \\ \hline (0000\ 0001)_2 \end{array}$$

AF = 0

AF: Example

$$
\begin{array}{rl}
 & \overset{1}{} \\
 & (0000\ 1000\ 0000\ 1000)_2 \\
+ & (1000\ 1000\ 0000\ 0000)_2 \\
\hline
 & (1001\ 0000\ 0000\ 1000)_2
\end{array}
$$

AF = 0

Parity Flag (PF)

- Indicates even parity of the low 8-bits of the result
- PF is set to ‘1’, if the lower 8-bits contain even number of 1
- For 16-bit and 32-bit values, only the least significant 8-bits are considered for computing parity value

PF: Examples

$$\begin{array}{r} (0000\ \overset{1}{0}000)_2 \\ +\ (0000\ 1001)_2 \\ \hline (0001\ 0001)_2 \end{array}$$

PF = 1

$$\begin{array}{r} (0000\ 0000)_2 \\ +(0000\ 0001)_2 \\ \hline (0000\ 0001)_2 \end{array}$$

PF = 0

PF: Example

$$
\begin{array}{rl}
 & \overset{1}{} \\
 & (0000\ 1000\ 0001\ 1000)_2 \\
+ & (0000\ 1000\ 0000\ 0000)_2 \\
\hline
 & (0001\ 0000\ 0001\ 1000)_2
\end{array}
$$

PF = 1

Carry Flag (CF)

- Records the fact that the result of an arithmetic operation on unsigned numbers is in range or out of range

- This flag is set to '1', when result is **out of range**
  - **Example**

$$(1111\ 1111)_2 + (0000\ 0001)_2 = (1\ 0000\ 0000)_2$$

- This flag is set to '0', when result is **in range**
  - **Example**

$$(1111\ 0011)_2 + (0000\ 0001)_2 = (1111\ 0100)_2$$

CF: Examples

$$\begin{array}{r} {\scriptstyle 1}\ (1100\ {\scriptstyle 1}1000)_2 \\ +\ (1100\ 1001)_2 \\ \hline (1001\ 0001)_2 \end{array}$$

CF = 1

$$\begin{array}{r} (0000\ 0000)_2 \\ +(0000\ 0001)_2 \\ \hline (0000\ 0001)_2 \end{array}$$

CF = 0

CF: Example

$$
\begin{array}{rl}
 & {}^{2}(0000\ 1000\ 0001\ 1000)_2 \\
- & (1000\ 1000\ 0000\ 0000)_2 \\
\hline
 & (1000\ 0000\ 0001\ 1000)_2
\end{array}
$$

CF = 1

Trap Flag (TF)

- It allows setting the operation of the processor in single step mode

- **Example**
  - The DEBUG program we used, sets the trap flag to ‘1’, so we could step through executing one instruction at a time

Example

- Consider the following piece of code and show the values of ax, bx, OF, IF, SF, ZF, AF, PF and CF.

```
[org 0x0100]
mov al,10
mov bl,1
sub ah, bl
add al,ah
Mov ax,0xffff
xor ax,ax
mov ax,0x4c00 ; exit
int 0x21
```

Instructions	AX	BX	OF	IF	SF	ZF	AF	PF	CF
[org 0x0100]									
mov al,10	00 0A	00 00	0	1	0	0	0	0	0
mov bl,1	00 0A	00 01	0	1	0	0	0	0	0
sub ah, bl	FF 0A	00 01	0	1	1	0	1	1	1
add al,ah	FF 09	00 01	0	1	0	0	1	1	1
Mov ax,oxffff	FF FF	00 01	0	1	0	0	1	1	1
xor ax,ax	00 00	00 01	0	1	0	1	0	1	0
mov ax,0x4c00									
int 0x21									

Task

- Consider the following piece of code and show the values of ax, bx, OF, IF, SF, ZF, AF, PF and CF.

```
[org 0x0100]
xor ax,ax
mov al,12
mov bl,5
sub ah, bl
add ah,ah
mov ah,al
add al,bl
mov ax,0xffff
xor ax,ax
mov ax,0x4c00    ; exit
int 0x21
```