Fall 2015

CSE 231 (Microprocessor and Assembly Language)

Week - 4

Books Referred

1. Chapter 5

   Reference Books II

   Assembly Language Programming and Organization of the IBM PC

   Ytha Yu and Charles Marut

   McGraw-Hill

Topic Contents

- The FLAGS Register
- Overflow
- How Instructions Affect the Flags

The Processor Status and the FLAGS Register

The FLAGS Register

- The flags are placed in the FLAGS register and they arc classified as either status flags or control flags.
- The status flags reflect the result of a computation.
- They are affected by the machine instructions.
- They are used to implement jump instructions that allow programs to have multiple branches and loops.
- The control flags are used to enable or disable certain operations of the processor

The FLAGS Register

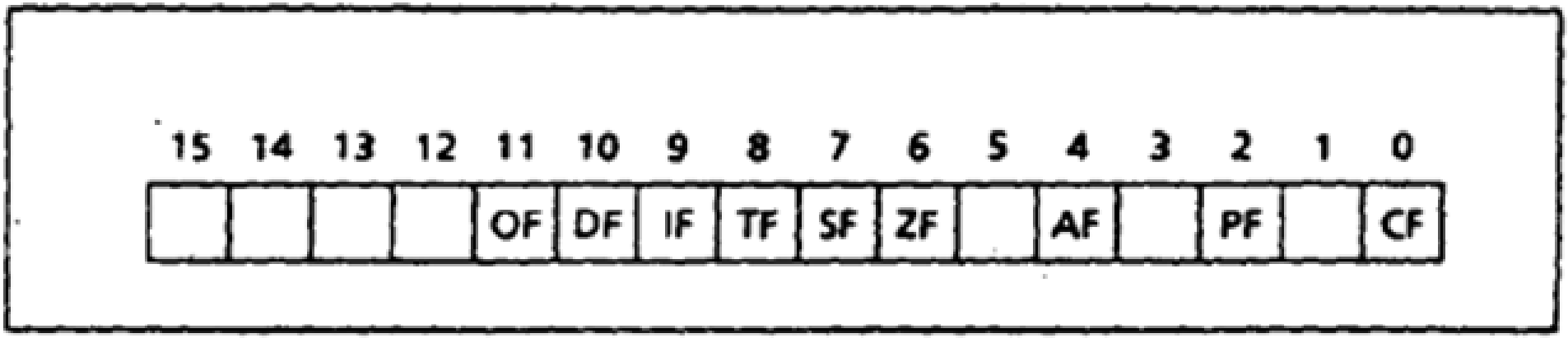

Figure 5.1 The FLAGS Register

- Figure 5.1 shows the FLAGS register.
- The status flags are located in bits 0, 2, 4, 6, 7, and 11 and the control flags are located in bits 8, 9, and 10.
- The other bits have no significance.
- In this topic, we concentrate on the status flags.

The FLAGS Register

- **msb**: The **most significant bit**, or **msb**, is the leftmost bit in an 8-bit byte or a 16-bit word. In a word, the msb is bit 15; in a byte, it is bit 7.
- **lsb**: The **least significant bit**, or **lsb**, is the rightmost bit in an 8-bit byte or a 16-bit word. In a byte or word, the lsb is bit 0.

The FLAGS Register

- ***Carry Flag (CF)*** *:* CF = 1 if there is a carry out from the most significant bit (msb) on addition, or there Is a borrow into the msb on subtraction; otherwise, it is 0. CF is also affected by shift and rotate Instructions.
- ***Parity Flag (PF)*** *:* PF = 1 if the low byte of a result has an even number of one bits (even parity). It is O if the low byte has, odd parity. For example, if the result of a word addition is FFFEh, then the low byte contains 7 one bits, so PF=0.
- ***Auxiliary Carry Flag (AF)*** : AF = 1 if there is a carry out from bit 3 on addition, or a borrow into bit 3 on subtraction.

The FLAGS Register

- ***Sign Flag (SF) :*** SF = 1 if the msb of a result is 1; it means the result is negative if you are giving a signed interpretation. SF = 0 if the msb is 0 .
- ***Overflow Flag (OF) :*** OF = 1 if signed overflow occurred, otherwise it is 0.
- ***Zero Flag (ZF) :*** ZF = 1 for a zero result, and ZF = 0 for a nonzero result.

Overflow

- Signed and unsigned overflows are independent phenomena. When we perform an arithmetic operation such as addition, there are four possible outcomes:

  (1) no overflow,

  (2) signed overflow only,

  (3) unsigned overflow only, and

  (4) both signed and unsigned overflows.

Unsigned Overflow

- As an example of unsigned overflow but not signed overflow, suppose AX contains FFFFh, BX contains 0001 h, and ADD AX,BX is executed. The binary result is-

```
  1111 1111 1111 1111
+ 0000 0000 0000 0001
---------------------
1 0000 0000 0000 0000
```

- If we are giving an unsigned interpretation, the correct answer is 1OOOOh = 65536, but this is out of range for a word operation. A 1 is carried out of the msb and the answer stored in AX, OOOOh, is wrong, so unsigned overflow occurred. But the stored answer is correct as a signed number, for FFFFh = -1. OOO1h = 1, and FFFFh + OOO1h *= -1 + 1 = 0, so signed overflow* did not occur.

Signed Overflow

- As an example of signed but not unsigned overflow, suppose AX and BX both contain 7FFFh, *and we execute ADD AX,BX. The binary result is*

```
  0111 1111 1111 1111
+ 0111 1111 1111 1111
-------------------
  1111 1111 1111 1110 = FFFEh
```

- The signed and unsigned decimal interpretation of 7FFFh is 32767. Thus for both signed and unsigned addition, 7FFFh + 7FFFh = 32767 + 32767 = 65534.
- This is out of range for signed numbers; the signed interpretation of the stored answer FFFEh is 2. so signed overflow occurred. However, the unsigned interpretation of FFFEh is 65534, which is the right answer, so there no unsigned overflow.

Unsigned Overflow

- On addition, unsigned overflow occurs when there Is a carry out of the msb. This means that the correct answer is larger than the biggest unsigned number; that is, FFFFh for a word and FFh for a byte .
- On subtraction, unsigned overflow occurs when there is a borrow into the msb. This means that the correct answer is smaller than 0.

Signed Overflow

- On addition of numbers with the same sign, signed overflow occurs when the sum has a different sign. This happened in the preceding example when we were adding 7FFFh. and 7FFFh (two positive numbers), but got FFFEh (a negative result).
- Subtraction of numbers with different signs is like adding numbers of the same sign. For example, A -(-B) = A + B and -A -(+B) = -A + -B. Signed overflow occurs if the result has a different sign than expected.

Signed Overflow

- In addition of numbers with different signs, overflow is impossible, because a sum like A + (-B) is really A - B, and because A and B are small enough to fit in the destination, so is A - B. For exactly the same reason, subtraction of numbers with the' same sign cannot give overflow.
- Actually, the processor. uses the following method to set the OF: If the carries into and out of the msb don't match-that is, there *is a carry into* the msb but no carry out, ·or if there is a carry out but no *carry in-then* signed overflow has occurred, and OF is set to 1.

Signed Overflow

- Unsigned overflow conditions for addition and subtraction:

Operation	Result indicating overflow
$A + B$	There is a carry out of the msb
$A - B$	There is a borrow into the msb

- Signed overflow conditions for addition and subtraction:

Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

How Instructions Effect the Flags

Instruction	*Affects flags*
MOV/XCHG	none
ADD/SUB	all
INC/DEC	all except CF
NEG	all (CF = 1 unless result is 0, OF = 1 if word operand is 8000h, or byte operand is 80h)

Examples

Example 5.1 ADD AX,BX, where AX contains FFFFh, BX contains FFFFh.

Solution:

```
  FFFFh
+ FFFFh
--------
1 FFFEh
```

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

SF = 1 because the msb is 1.

PF = 0 because there are 7 (odd number) of 1 bits in the low byte of the result.

ZF = 0 because the result is nonzero.

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).

Examples

Example 5.2 ADD AL,BL, where AL contains 80h, BL contains 80h.

Solution:

```
  80h
+ 80h
-----
 100h
```

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 addition.

OF = 1 because the numbers being added are both negative, but the result is 0 (as a binary addition, there is no carry into the msb but there is a carry out).

Examples

Example 5.3 SUB AX,BX, where AX contains 8000h and BX contains 0001h.

Solution:

```
  8000h
- 0001h
-------
  7FFFh = 0111 1111 1111 1111
```

The result stored in AX is 7FFFh.

SF = 0 because the msb is 0.

PF = 1 because there are 8 (even number) one bits in the low byte of the result.

ZF = 0 because the result is nonzero.

CF = 0 because a smaller unsigned number is being subtracted from a larger one.

Now for OF. In a signed sense, we are subtracting a positive number from a negative one, which is like adding two negatives. Because the result is positive (the wrong sign), OF = 1.

Examples

Example 5.4 INC AL, where AL contains FFh.

Solution:

$$\begin{array}{r} \text{FFh} \\ +\ \text{1h} \\ \hline 1\ 00\text{h} \end{array}$$

The result stored in AL is 00h. SF = 0, PF = 1, ZF = 1. Even though there is a carry out, CF is unaffected by INC. This means that if CF = 0 before the execution of the instruction, CF will still be 0 afterward.

OF = 0 because numbers of unlike sign are being added (there is a carry into the msb and also a carry out).

Example 5.5 MOV AX, –5

Solution: The result stored in AX is –5 = FFFBh.

None of the flags are affected by MOV.