FLAG REGISTER
AND
ADD INSTRUCTION

Flag register

The flag register is a 16-bit register sometimes referred to as the *status register*. Although the register is 16 bits wide, only some of the bits are used. The rest are either undefined or reserved by Intel. Six of the flags are called conditional flags, meaning that they indicate some condition that resulted after an instruction was executed. These six are CF, PF, AF, ZF, SF, and OF. The three remaining flags are sometimes called control flags since they are used to control the operation of instructions before they are executed. A diagram of the flag register is shown in Figure 1-5.

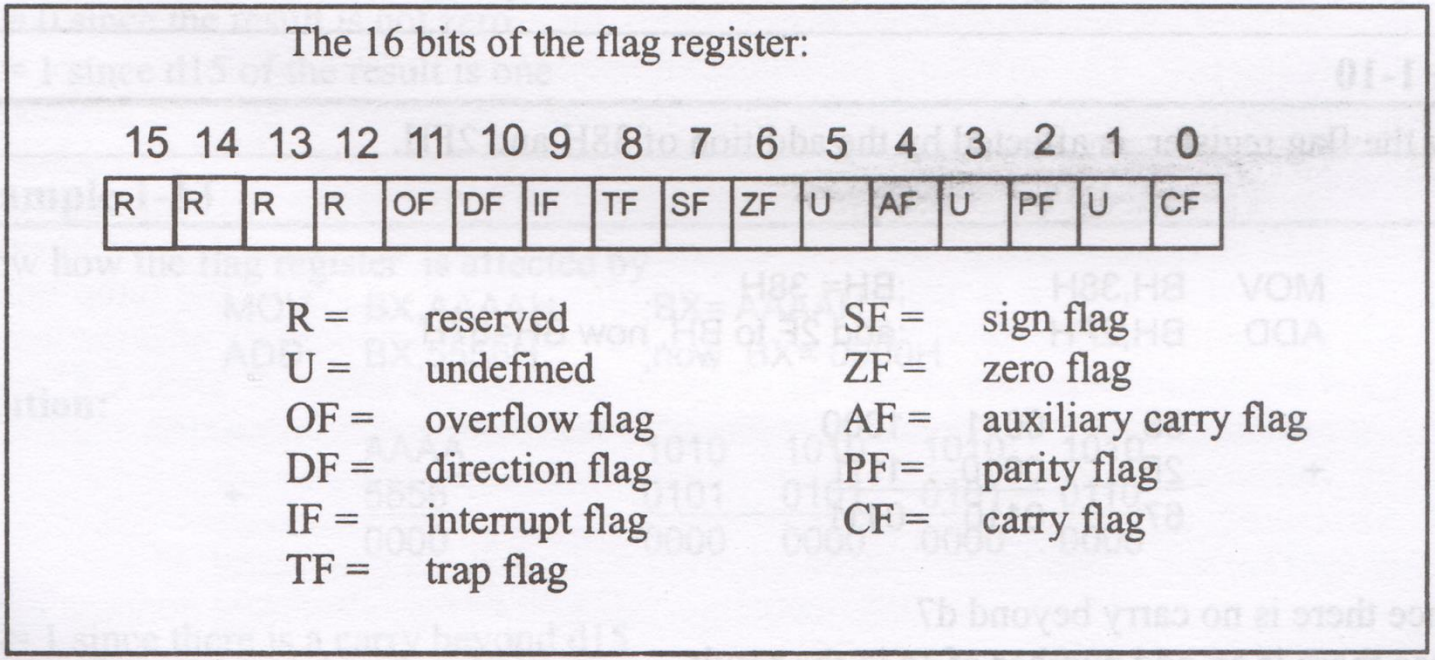

Figure 1-5. Flag Register

Bits of the flag register

Below are listed the bits of the flag register that are used in 80x86 Assembly language programming. A brief explanation of each bit is given. How these flag bits are used will be seen in programming examples throughout the textbook.

CF, the Carry Flag . This flag is set whenever there is a carry out, either from d7 after an 8-bit operation, or from d15 after a 16-bit data operation.

PF, the Parity Flag. After certain operations, the parity of the result's low-order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise, it is cleared.

AF, Auxiliary Carry Flag. If there is a carry from d3 to d4 of an operation, this bit is set; otherwise, it is cleared (set equal to zero). This flag is used by the instructions that perform BCD (binary coded decimal) arithmetic.

ZF, the Zero Flag. The zero flag is set to 1 if the result of an arithmetic or logical operation is zero; otherwise, it is cleared.

SF, the Sign Flag. Binary representation of signed numbers uses the most significant bit as the sign bit. After arithmetic or logic operations, the status of this sign bit is copied into the SF, thereby indicating the sign of the result.

TF, the Trap Flag. When this flag is set it allows the program to single-step, meaning to execute one instruction at a time. Single-stepping is used for debugging purposes.

IF, Interrupt Enable Flag. This bit is set or cleared to enable or disable only the external maskable interrupt requests.

DF, the Direction Flag. This bit is used to control the direction of string operations, which are described in Chapter 6.

OF, the Overflow Flag. This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit. In general, the carry flag is used to detect errors in unsigned arithmetic operations. The overflow flag is only used to detect errors in signed arithmetic operations.

Flag register and ADD instruction

In this section we examine the impact of the ADD instruction on the flag register as an example of the use of the flag bits. The flag bits affected by the ADD instruction are CF (carry flag), PF (parity flag), AF (auxiliary carry flag), ZF (zero flag), SF (sign flag), and OF (overflow flag). The overflow flag will be covered in Chapter 6, since it relates only to signed number arithmetic. To understand how each of these flag bits is affected, look at Examples 1-10 and 1-11.

Example 1-10

Show how the flag register is affected by the addition of 38H and 2FH.

Solution:

```
MOV   BH,38H        ;BH= 38H
ADD   BH,2FH        ;add 2F to BH, now BH=67H
```

```
      38    0011  1000
+     2F    0010  1111
      67    0110  0111
```

CF = 0 since there is no carry beyond d7

PF = 0 since there is an odd number of 1s in the result

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 0 since d7 of the result is zero

Example 1-11

Show how the flag register is affected by

```
MOV   AL,9CH      ;AL=9CH
MOV   DH,64H      ;DH=64H
ADD   AL,DH       ;now AL=0
```

Solution:

```
      9C    1001  1100
+     64    0110  0100
      00    0000  0000
```

CF=1 since there is a carry beyond d7

PF=1 since there is an even number of 1s in the result

AF=1 since there is a carry from d3 to d4

ZF=1 since the result is zero

SF=0 since d7 of the result is zero

Example 1-12

Show how the flag register is affected by

```
MOV   AX,34F5H     ;AX= 34F5H
ADD   AX,95EBH     ;now AX= CAE0H
```

Solution:

```
      34F5     0011  0100  1111  0101
+     95EB     1001  0101  1110  1011
      CAE0     1100  1010  1110  0000
```

CF = 0 since there is no carry beyond d15

PF = 0 since there is an odd number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 1 since d15 of the result is one

Example 1-13

Show how the flag register is affected by

```
MOV   BX,AAAAH    ;BX= AAAAH
ADD   BX,5556H    ;now BX= 0000H
```

Solution:

	AAAA	1010	1010	1010	1010
+	5556	0101	0101	0101	0110
	0000	0000	0000	0000	0000

CF = 1 since there is a carry beyond d15

PF = 1 since there is an even number of 1s in the lower byte

AF = 1 since there is a carry from d3 to d4

ZF = 1 since the result is zero

SF = 0 since d15 of the result is zero

ADD Signed or Unsigned ADD

Flags: Affected: OF, SF, ZF, AF, PF, CF.
Format: ADD dest,source ;dest = dest + source
Function: Adds source operand to destination operand and places the result in destination. Both source and destination operands must match (e.g., both byte size or word size) and only one of them can be in memory.

ADC Add with Carry

Flags: Affected: OF, SF, ZF, AF, PF, CF.
Format: ADC dest,source ;dest = dest + source + CF
Function: If CF=1 prior to this instruction, then after execution of this instruction, source is added to destination plus 1. If CF = 0, source is added to destination plus 0. Used widely in multibyte and multiword additions.

INC Increment

Flags: Affected: OF, SF, ZF, AF, PF. Unchanged: CF.
Format: INC destination ;dest = dest + 1
Function: Adds 1 to the register or memory location specified by the operand. Note that CF is not affected even if a value FFFF is incremented to 0000.

DEC Decrement

Flags: Affected: OF, SF, ZF, AF, PF. Unchanged: CF.
Format: DEC dest ;dest = dest - 1
Function: Subtracts 1 from the destination operand. Note that CF (carry/ borrow) is unchanged even if a value 0000 is decremented and becomes FFFF.

Use of the zero flag for looping

One of the most widely used applications of the flag register is the use of the zero flag to implement program loops. The term *loop* refers to a set of instructions that is repeated a number of times. For example, to add 5 bytes of data, a counter can be used to keep track of how many times the loop needs to be repeated. Each time the addition is performed the counter is decremented and the zero flag is checked. When the counter becomes zero, the zero flag is set (ZF = 1) and the loop is stopped. The following shows the implementation of the looping concept in the program, which adds 5 bytes of data. Register CX is used to hold the counter and BX is the offset pointer (SI or DI could have been used instead). AL is initialized before the start of the loop. In each iteration, ZF is checked by the JNZ instruction. JNZ stands for "Jump Not Zero" meaning that if ZF = 0, jump to a new address. If ZF = 1, the jump is not performed and the instruction below the jump will be executed. Notice that the JNZ instruction must come immediately after the instruction that decrements CX since JNZ needs to check the affect of "DEC CX" on the zero flag. If any instruction were placed between them, that instruction might affect the zero flag.

```
        MOV   CX,05        ;CX holds the loop count
        MOV   BX,0200H     ;BX holds the offset data address
        MOV   AL,00        ;initialize AL
ADD_LP: ADD   AL,[BX]      ;add the next byte to AL
        INC   BX           ;increment the data pointer
        DEC   CX           ;decrement the loop counter
        JNZ   ADD_LP       ;jump to next iteration if counter not zero
```

Thank you