5

The Processor Status and the FLAGS Register

Overview

One important feature that distinguishes a computer from other machines is the computer's ability to make decisions. The circuits in the CPU can perform simple decision making based on the current state of the processor. For the 8086 processor, the processor state is implemented as nine individual bits called **flags**. Each decision made by the 8086 is based on the values of these flags.

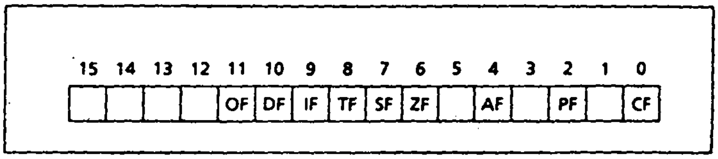
The flags are placed in the FLAGS register and they are classified as either status flags or control flags. The status flags reflect the result of a computation. In this chapter, you will see how they are affected by the machine instructions. In Chapter 6, you will see how they are used to implement 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; they are covered in later chapters.

In section 5.4 we introduce the DOS program DEBUG. We'll show how to use DEBUG to trace through a user program and to display registers, flags, and memory locations.

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. *Note:* it's not important to remember

Figure 5.1 The FLAGS Register



which bit is which flag—Table 5.1 gives the names of the flags and their symbols. In this chapter, we concentrate on the status flags.

The Status Flags

As stated earlier, the processor uses the status flags to reflect the result of an operation. For example, if SUB AX,AX is executed, the zero flag becomes 1, thereby indicating that a zero result was produced. Now let's get to know the status flags.

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 (Chapter 7).

Parity Flag (PF)

PF = 1 if the low byte of a result has an even number of one bits (even parity). It is 0 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.

Table 5.1 Flag Names and Symbols

Status Flags

Bit	***Name***	***Symbol***
0	Carry flag	CF
2	Parity flag	PF
4	Auxiliary carry flag	AF
6	Zero flag	ZF
7	Sign flag	SF
11	Overflow flag	OF

Control Flags

Bit	***Name***	***Symbol***
8	Trap flag	TF
9	Interrupt flag	IF
10	Direction flag	DF

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. AF is used in binary-coded decimal (BCD) operations (Chapter 18).

Zero Flag (ZF)

ZF = 1 for a zero result, and ZF = 0 for a nonzero result.

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. The meaning of overflow is discussed next.

5.2 Overflow

The phenomenon of overflow is associated with the fact that the range of numbers that can be represented in a computer is limited.

Chapter 2 explained that the (decimal) range of signed numbers that can be represented by a 16-bit word is -32768 to 32767; for an 8-bit byte the range is -128 to 127. For unsigned numbers, the range for a word is 0 to 65535; for a byte, it is 0 to 255. If the result of an operation falls outside these ranges, overflow occurs and the truncated result that is saved will be incorrect.

Examples of 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.

As an example of unsigned overflow but not signed overflow, suppose AX contains FFFFh, BX contains 0001h, 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 10000h = 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, 0000h, is wrong, so unsigned overflow occurred. But the stored answer is correct as a signed number, for FFFFh = -1, 0001h = 1, and FFFFh + 0001h = -1 + 1 = 0, so signed overflow did not occur.

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 is no unsigned overflow.

There are two questions to be answered in connection with overflow: (1) how does the CPU indicate overflow, and (2) how does it know that overflow occurred?

How the Processor Indicates Overflow

The processor sets OF = 1 for signed overflow and CF = 1 for unsigned overflow. It is then up to the program to take appropriate action, and if nothing is done immediately the result of a subsequent instruction may cause the overflow flag to be turned off.

In determining overflow, the processor does not interpret the result as either signed or unsigned. The action it takes is to use both interpretations for each operation and to turn on CF or OF for unsigned overflow or signed overflow, respectively.

It is the programmer who is interpreting the results. If a signed interpretation is being given, then only OF is of interest and CF can be ignored; conversely, for an unsigned interpretation CF is important but not OF.

How the Processor Determines that Overflow Occurred

Many instructions can cause overflow; for simplicity, we'll limit the discussion to addition and subtraction.

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. See example 5.3, in the next section.

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. See example 5.2, in the next section.

5.3 How Instructions Affect the Flags

In general, each time the processor executes an instruction, the flags are altered to reflect the result. However, some instructions don't affect any of the flags, affect only some of them, or may leave them undefined. Because the jump instructions studied in Chapter 6 depend on the flag settings, it's important to know what each instruction does to the flags. Let's return to the seven basic instructions introduced in Chapter 4. They affect the flags as follows:

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)

To get you used to seeing how these instructions affect the flags, we will do several examples. In each example, we give an instruction, the contents of the operands, and predict the result and the settings of CF, PF, ZF, SF, and OF (we ignore AF because it is used only for BCD arithmetic).

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).

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

Solution:

```
   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 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).

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.

Example 5.4 INC AL, where AL contains FFh.

Solution:

```
   FFh
 +  1h
 1 00h
```

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.

Example 5.6 NEG AX, where AX contains 8000h.

Solution:

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

The result 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 = 1, 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.

In the next section, we introduce a program that lets us see the actual setting of the flags.

5.4 The DEBUG Program

The DEBUG program provides an environment in which a program may be tested. The user can step through a program, and display and change the registers and memory. It is also possible to enter assembly code directly, which DEBUG converts to machine code and stores in memory. A tutorial for DEBUG and CODEVIEW, a more sophisticated debugger, may be found in Appendix E.

We use DEBUG to demonstrate the way instructions affect the flags. To that end, the following program has been created.

Program Listing PGM5_1.ASM

```
TITLE PGM5_1:CHECK FLAGS
;used in DEBUG to check flag settings
.MODEL  SMALL
.STACK  100H
.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           ;DOS exit
MAIN   ENDP
       END    MAIN
```

We assemble and link the program, producing the run file PGM5_1.EXE, which is on a disk in drive A. In the following, the user's responses are in boldface.

The DEBUG program is on the DOS disk, which is in drive C. To enter DEBUG with our demonstration program, we type

```
C>DEBUG A:PGM5_1.EXE
```

DEBUG responds by its prompt, " -", and waits for a command to be entered. First, we can view the registers by typing "R".

```
-R
AX=0000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0000 NV UP DI PL NZ NA PO NC
0EE6:0000 B80040     MOV    AX,4000
```

The display shows the contents of the registers in hex. On the third line of the display, we see

```
0EE6:0000 B80040     MOV    AX,4000
```

0EE6:0000 is the address of the next instruction to be executed, in segment:offset form. B80040h is the machine code of that instruction. Segment 0EE6h is where DOS decided to load the program; if you try this demonstration, you will probably see a different segment number.

The eight pairs of letters appearing on the second line at the right are the current flag settings. The flags appear in this order: OF, DF, IF, SF, ZF, AF, PF, and CF. Table 5.2 gives the symbols DEBUG uses for the flags. You can see that they have been cleared by DEBUG. The meaning of the control flag symbols are explained in Chapters 11 and 15.

To step through our program, we use the "T" (trace) command. Before doing so, let's display the registers again.

```
-R
AX=0000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0000 NV UP DI PL NZ NA PO NC
0EE6:0000 B80040     MOV    AX,4000
```

The first instruction is MOV AX,4000h.

```
-T
AX=4000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0003 NV UP DI PL NZ NA PO NC
0EE6:0003 03C0       ADD    AX,AX
```

Execution of MOV AX,4000h puts 4000h in AX. The flags are unchanged since a MOV doesn't affect them. Now let's execute ADD AX,AX:

```
-T
AX=8000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0005 OV UP DI NG NZ NA PE NC
0EE6:0005 2DFFFF     SUB    AX,FFFF
```

Table 5.2 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 interrupts)

AX now contains 4000h + 4000h = 8000h. SF becomes 1 (NG) to indicate a negative result. Signed overflow is indicated by OF = 1 (OV) because we added two positive numbers and got a negative result. PF = 1 (PE) because the low byte of AX contains no 1's.

Next we trace SUB AX,0FFFFh:

```
-T
AX=8001 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=0008 NV UP DI NG NZ AC PO CY
0EE6:0008 F7D8     NEG   AX
```

AX gets 8000h – FFFFh = 8001h. OF changes back to 0 (NV), because we are subtracting numbers of like sign, so signed overflow is impossible. However, CF = 1 (CY) indicates that we got unsigned overflow, because we have subtracted a bigger unsigned number from a smaller one, which requires a borrow into the msb. PF = 0 (PO) because the low byte of AX has a single 1.

Now let's trace NEG AX:

```
-T
AX=7FFF BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=000A NV UP DI PL NZ AC PE CY
0EE6:000A 40       INC   AX
```

AX gets the two's complement of 8001h = 7FFFh. For NEG, CF = 1 (CY) unless the result is 0, which is not the case here. OF = 0 (NV) because the result is not 8000h.

Finally, we execute INC AX:

```
-T
AX=8000 BX=0000 CX=001F DX=0000 SP=000A BP=0000 SI=0000 DI=0000
DS=0ED5 ES=0ED5 SS=0EE5 CS=0EE6 IP=000B OV UP DI NG NZ AC PE CY
0EE6:000B B44C      MOV   AH,4C
```

OF changes back to 1 (OV) because we added two positives (7FFFh and 1), and got a negative result. Even though there was no carry out of the msb, CF stays 1 because INC doesn't affect this flag.
To complete execution of the program, we can type "G" (go):

```
-G
Program terminated normally
```

and to exit DEBUG, type "Q" (quit)

```
-Q
C>
```

Summary

- The FLAGS register is one of the registers in the 8086 microprocessor. Six of the bits are called status flags, and three are control flags.
- The status flags reflect the result of an operation. They are the carry flag (CF), parity flag (PF), auxiliary carry flag (AF), zero flag (ZF), sign flag (SF), and overflow flag (OF).
- CF is 1 if an add or subtract operation generates a carry out or borrow into the most significant bit position; otherwise, it is 0.
- PF is 1 if there is an even number of 1 bits in the result; otherwise, it is 0.
- AF is 1 if there is a carry out or borrow into bit 3 in the result; otherwise, it is 0.
- ZF is 1 if the result is 0; otherwise, it is 0.
- SF is 1 if the most significant bit of the result is 1; otherwise, it is 0.
- OF is 1 if the correct signed result is too big to fit in the destination; otherwise, it is 0.

- Overflow occurs when the correct result is outside the range of values represented by the computer. Unsigned overflow occurs if an unsigned interpretation is being given to the result, and signed overflow happens if a signed interpretation is being given.
- The processor uses CF and OF to indicate overflow: CF = 1 means that unsigned overflow occurred, and OF = 1 indicates signed overflow.
- The processor sets CF if there is a carry out of the msb on addition, or a borrow into the msb on subtraction. In the latter case, this means that a larger unsigned number is being subtracted from a smaller one.
- The processor sets OF if there is a carry into the msb but no carry out, or if there is a carry out of the msb but no carry in.
- There is another way to tell whether signed overflow occurred on addition and subtraction. On addition of numbers of like sign, signed overflow occurs if the result has a different sign; subtraction of numbers of different sign is like adding numbers of the same sign, and signed overflow occurs if the result has a different sign.
- On addition of numbers of different sign, or subtraction of numbers of the same sign, signed overflow is impossible.
- Generally the execution of each instruction affects the flags, but some instructions don't affect any of the flags, and some affect only some of the flags.
- The settings of the flags is part of the DEBUG display.
- The DEBUG program may be used to trace a program. Some of its commands are "R", to display registers; "T", to trace an instruction; and "G", to execute a program.

Glossary

control flags	Flags that are used to enable or disable certain operations of the CPU
flags	Bits of the FLAGS register that represent a condition of the CPU
FLAGS register	Register in the CPU whose bits are flags
status flags	Flags that reflect the result of an instruction executed by the CPU

Exercises

1. For each of the following instructions, give the new destination contents and the new settings of CF, SF, ZF, PF, and OF. Suppose that the flags are initially 0 in each part of this question.

a.	`ADD AX,BX`	where AX contains 7FFFh and BX contains 0001h
b.	`SUB AL,BL`	where AL contains 01h and BL contains FFh
c.	`DEC AL`	where AL contains 00h

d.	NEG AL	where AL contains 7Fh
e.	XCHG AX,BX	where AX contains 1ABCh and BX contains 712Ah
f.	ADD AL,BL	where AL contains 80h and BL contains FFh
g.	SUB AX,BX	where AX contains 0000h and BX contains 8000h
h.	NEG AX	where AX contains 0001h

2. a. Suppose that AX and BX both contain positive numbers, and ADD AX,BX is executed. Show that there is a carry into the msb but no carry out of the msb if, and only if, signed overflow occurs.

   b. Suppose AX and BX both contain negative numbers, and ADD AX,BX is executed. Show that there is a carry out of the msb but no carry into the msb if, and only if, signed overflow occurs.

3. Suppose ADD AX,BX is executed. In each of the following parts, the first number being added is the contents of AX, and the second number is the contents of BX. Give the resulting value of AX and tell whether signed or unsigned overflow occurred.

```
a.  512Ch
  + 4185h

b.  FE12h
  + 1ACBh

c.  E1E4h
  + DAB3h

d.  7132h
  + 7000h

e.  6389h
  + 1176h
```

4. Suppose SUB AX,BX is executed. In each of the following parts, the first number is the initial contents of AX and the second number is the contents of BX. Give the resulting value of AX and tell whether signed or unsigned overflow occurred.

```
a.  2143h
  - 1986h

b.  81FEh
  - 1986h

c.  19BCh
  - 81FEh

d.  0002h
  - FE0Fh

e.  8BCDh
  - 71ABh
```