# The Processor Status Register (Flags)

The flags register maintains the current operating mode of the CPU and some instruction state information. The following figure shows the layout of the flags register.

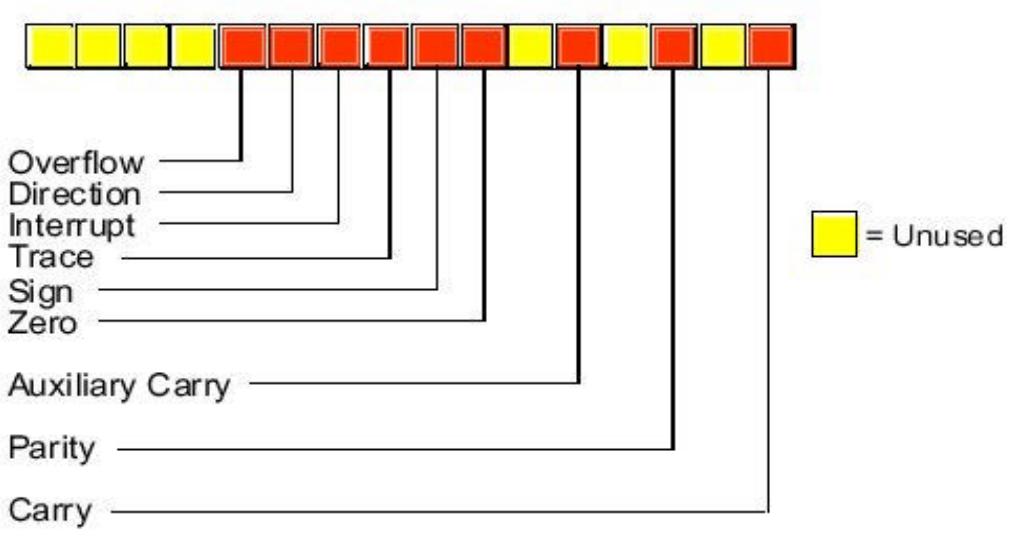


**Figure 8.1: 80x86 Flags Register**

| Flag | Meaning | Set | Cleared | Position | Function |
|---|---|---|---|---|---|
| OF | Overflow Flag | OV | NV | 11 | Status |
| DF | Direction Flag | DN | UP | 10 | Control |
| IF | Interrupt Flag | EI | DI | 9 | Control |
| SF | Sign Flag | NG | PL | 7 | Status |
| ZF | Zero Flag | ZR | NZ | 6 | Status |
| AF | Auxiliary Carry Flag | AC | NA | 4 | Status |
| PF | Parity Flag | PE | PO | 2 | Status |
| CF | Carry Flag | CY | NC | 0 | Status |
| TF | Trap Flag | NA. | | 8 | Control |

**Table 8.1: 80x86 Flags Register**

**Note 1**: The carry, parity, zero, sign, and overflow flags can be set or cleared with the set clear and conditional jump instructions. The 80x86 uses these bits, to make decisions during program execution.

**Note 2**: There are no instructions to set or clear the trace flag.

**The Overflow Flag:**

Various arithmetic, logical, and miscellaneous instructions affect the overflow flag. After an arithmetic operation, this flag contains one if the result does not fit in the signed destination operand.

Example:

Add two 16-bit signed numbers 7FFFh and 0001h

```
      7FFFh
+     0001h
=   1 0000h  ✍  OF = OV = 1
```

the result is too large to fit in one word, the overflow flag is set. If the result of the arithmetic operation does not produce a signed overflow, the CPU clears this flag.

Since the logical operations generally apply to unsigned values, the 80x86 logical instructions simply clear the overflow flag. Other 80x86 instructions leave the overflow flag containing an arbitrary value.

**The Sign Flag:**

If the result of some computation is negative, the 80x86 sets the sign flag. This flag can be testes after an arithmetic operation to check for a negative result. Note that, a value is negative if its H.O. bit is one. Therefore, operations on unsigned values will set the sign flag if the result has a one in the H.O. position.

**The Zero Flag:**

Various instructions set the zero flag when they generate a zero result. You will often use this flag to see if two values are equal (e.g., after subtracting two numbers, they are equal if the result is zero). This flag is also useful after various logical operations to see if a specific bit in a register or memory location contains zero or one.

**The Auxiliary Carry Flag:**

The auxiliary carry flag supports special binary coded decimal (BCD) operations. Since most programs do not deal with BCD numbers, you will rarely use this flag and even then you will not access it directly. The 80x86 CPUs do not provide any instructions that let you directly test, set, or clear this flag. Only the add, adc, sub, sbb, mul, imul, div, idiv, and BCD instructions manipulate this flag.

**The Parity Flag:**

The parity flag is set according to the parity of the **L.O. eight bits of any data operation**. If an operation produces an **even** number of **one** (**1**) bits, the CPU sets this flag. It clears this flag if the operation yields an odd number of **ones**. This flag is useful in certain data communication programs. However, Intel provided it mainly to provide some compatibility with the older 8080 microprocessor.

**The Carry Flag:**

The carry flag has several purposes. First, it denotes an unsigned overflow (much like the overflow flag detects a signed overflow). You will also use it during precision arithmetic and logical operations. Certain bit test set clear and invert instructions on the 80386 directly affect this flag.
Finally, since you can easily clear, set, invert, and test it, it is useful for various boolean operations. The carry flag has many purposes and knowing when to use it, and for what purpose, can confuse beginning assembly language programmers. Fortunately, for any given instruction, the meaning of the carry flag is clear.

**The Direction Flag:**

The 80x86 string instructions use the direction flag. When the direction flag is clear, the 80x86 processes string elements from low addresses to high addresses; when set, the CPU processes strings in the opposite direction.

**The Interrupt Enable Flag:**

The interrupt enable/disable flag controls the 80x86 ability to respond to external events known as interrupt requests. Some programs contain certain instruction sequences that the CPU must not interrupt. The interrupt enable/disable flag turns interrupts on or off to guarantee that the CPU does not interrupt those critical sections of code.

**The Trace Flag:**

The trace flag enables or disables the 80x86 trace mode. Debuggers (such as CodeView) use this bit to enable or disable the single step/trace operation. When set, the CPU interrupts each instruction and passes control to the debugger software, allowing the debugger to single step through the application. If the trace bit is clear, then the 80x86 executes instructions without the interruption. The 80x86 CPUs do not provide any instructions that directly manipulate this flag. To set or clear the trace flag, you must:

| | |
|---|---|
| ✍ Push the flags onto the 80x86 stack | PUSHF |
| ✍ Pop the value into another register | POP AX |
| ✍ Change the trace flag value | XOR AX, 0100H |
| ✍ Push the result onto the stack, and then | POP AX |
| ✍ Pop the flags off the stack | POPF |