# Carry flag

In computer processors the **carry flag** (usually indicated as the **C flag**) is a single bit in a system status register/flag register used to indicate when an arithmetic carry or borrow has been generated out of the most significant arithmetic logic unit (ALU) bit position. The carry flag enables numbers larger than a single ALU width to be added/subtracted by carrying (adding) a binary digit from a partial addition/subtraction to the least significant bit position of a more significant word. It is also used to extend bit shifts and rotates in a similar manner on many processors (sometimes done via a dedicated **X** flag). For subtractive operations, two (opposite) conventions are employed as most machines set the carry flag on borrow while some machines (such as the 6502 and the PIC) instead reset the carry flag on borrow (and vice versa).

## Contents



## Uses

The carry flag is affected by the result of most arithmetic (and typically several bit wise) instructions and is also used as an input to many of them. Several of these instructions have two forms which either read or ignore the carry. In assembly languages these instructions are represented by mnemonics such as `ADD/SUB`, `ADC/SBC` (`ADD/SUB` including carry), `SHL/SHR` (bit shifts), `ROL/ROR` (bit rotates), `RCR/RCL` (rotate through carry), and so on.[1] The use of the carry flag in this manner enables multi-word add, subtract, shift, and rotate operations.

An example is what happens if one were to add 255 and 255 using 8-bit registers. The result should be 510 which is the 9-bit value `111111110` in binary. The 8 least significant bits always stored in the register would be `11111110` binary (254 decimal) but since there is carry out of bit 7 (the eight bit), the carry is set, indicating that the result needs 9 bits. The valid 9-bit result is the concatenation of the carry flag with the result.

For x86 ALU size of 8 bits, an 8-bit two's complement interpretation, the addition operation `11111111` + `11111111` results in `111111110`, `Carry_Flag` set, `Sign_Flag` set, and `Overflow_Flag` clear.

If `11111111` represents two's complement signed integer −1 (`ADD al,-1`), then the interpretation of the result is `11111110` because `Overflow_Flag` is clear, and `Carry_Flag` is ignored. The sign of the result is negative, because `Sign_Flag` is set. `11111110` is the two's complement form of signed integer −2.

If `11111111` represents unsigned integer binary number 255 (`ADD al,255`), then the interpretation of the result that the `Carry_Flag` cannot be ignored. The `Overflow_Flag` and the `Sign_Flag` are ignored.

Another example may be an 8-bit register with the bit pattern `01010101` and the carry flag set; if we execute a *rotate left through carry* instruction, the result would be `10101011` with the carry flag cleared because the most significant bit (bit 7) was rotated into the carry while the carry was rotated into the least significant bit (bit 0).

The early microprocessors Intel 4004 and Intel 8008 had specific instructions to set as well as reset the carry flag explicitly. However, the later Intel 8080 (and Z80) did not include an explicit reset carry opcode as this could be done equally fast via one of the bitwise AND, OR or XOR instructions (which do not use the carry flag).

The carry flag is also often used following comparison instructions, which are typically implemented by subtractive operations, to allow a decision to be made about which of the two compared values is lower than (or greater or equal to) the other. Branch instructions which examine the carry flag are often represented by mnemonics such as `BCC` and `BCS` to branch if the carry is clear, or branch if the carry is set respectively. When used in this way the carry flag provides a mechanism for comparing the values as unsigned integers. This is in contrast to the overflow flag which provides a mechanism for comparing the values as signed integer values.

# Carry flag vs. borrow flag

While the carry flag is well-defined for addition, there are two ways in common use to use the carry flag for subtraction operations.

The first uses the bit as a borrow flag, setting it if $a<b$ when computing $a-b$, and a borrow must be performed. If $a \geq b$, the bit is cleared. A **subtract with borrow** (`SBB`) instruction will compute $a-b-C = a-(b+C)$, while a subtract without borrow (`SUB`) acts as if the borrow bit were clear. The 8080, Z80, 8051, x86[1] and 68k families (among others) use a borrow bit.

The second takes advantage of the identity that $-x = \text{not}(x)+1$ and computes $a-b$ as $a+\text{not}(b)+1$. The carry flag is set according to this addition, and **subtract with carry** computes $a+\text{not}(b)+C$, while subtract without carry acts as if the carry bit were set. The result is that the carry bit is set if $a \geq b$, and clear if $a<b$. The System/360,[2] 6502, MSP430, ARM and PowerPC processors use this convention. The 6502 is a particularly well-known example because it does not have a subtract *without* carry operation, so programmers must ensure that the carry flag is set before every subtract operation where a borrow is not required.

Summary of different uses of carry flag in subtraction

| Carry or borrow bit | Subtract without carry/borrow | Subtract with borrow | Subtract with carry |
|---|---|---|---|
| ***C* = 0** | $a - b$ $= a + \text{not}(b) + \mathbf{1}$ | $a - b - \mathbf{0}$ $= a + \text{not}(b) + \mathbf{1}$ | $a + \text{not}(b) + \mathbf{0}$ $= a - b - \mathbf{1}$ |
| ***C* = 1** | | $a - b - \mathbf{1}$ $= a + \text{not}(b) + \mathbf{0}$ | $a + \text{not}(b) + \mathbf{1}$ $= a - b - \mathbf{0}$ |

Most commonly, the first alternative is referred to as a "subtract with borrow", while the second is called a "subtract with carry". However, there are exceptions in both directions; the VAX, NS320xx, and Atmel AVR architectures use the borrow bit convention, but call their $a-b-C$ operation "subtract with carry" (`SBWC`, `SUBC` and `SBC`). The PA-RISC and PICmicro architectures use the carry bit convention, but call their $a+\text{not}(b)+C$ operation "subtract with borrow" (`SUBB` and `SUBWFB`).

The ST6/ST7 8-bit microcontrollers are perhaps the most confusing of all. Although they do not have any sort of "subtract with carry" instruction, they do have a carry bit which is set by a subtract instruction, and the convention depends on the processor model. The ST60 processor uses the "carry" convention, while the ST62 and ST63 processors use the "borrow" convention.[3]

## See also

- Binary arithmetic
- Half-carry flag
- Status register

## References


1. "Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual" (http://download.intel.com/design/PentiumII/manuals/24319102.PDF) (PDF). Retrieved 2007-10-25.
2. *IBM System/360 Principles of Operation* (http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/360/princOps/A22-6821-0_360PrincOps.pdf) (PDF). p. 28. IBM Form A22-6821-0.
3. "ST6 Family Programming Manual" (http://www.st.com/content/ccc/resource/technical/document/programming_manual/4d/05/d1/a5/a0/9e/40/8b/CD00004606.pdf/files/CD00004606.pdf/jcr:content/translations/en.CD00004606.pdf#page=42) (PDF). Revision 2.0. STMicroelectronics. October 2004. p. 42. Retrieved 2017-02-28.


## External links

- Carry Flag and Overflow Flag in binary arithmetic (http://teaching.idallen.com/dat2343/10f/notes/040_overflow.txt)
- Carry Bit: How does it work? (https://brodowsky.it-sky.net/2013/12/22/carry-bit-how-does-it-work/)