Two's complement system

A two's-complement system or two's-complement arithmetic is a system in which negative numbers are represented by the two's complement of the absolute value;[1] this system is the most common method of representing signed integers on computers. In such a system, a number is negated (converted from positive to negative or vice versa) by computing its two's complement.

The importance of this system is that it helps us represent negative numbers, while addition (ADC) and substraction (SBC) operations can be used exactly the same way as with unsigned numbers.

In finding the two's complement of a binary number, the bits are inverted, or “flipped”, by using the EOR #$FF operation; the value of 1 is then added (CLC ADC #$01) to the resulting value. Bit overflow is ignored, which is the normal case with zero.

in c64 assembly:

```
EOR #$FF
CLC
ADC #$01
```

For example, beginning with the signed 8-bit binary representation of the decimal value 5

```
%00000101 = 5
```

The most significant bit is 0, so the pattern represents a non-negative value. To convert to −5 in two's complement notation, the bits are inverted; 0 becomes 1, and 1 becomes 0:

```
%11111010
```

At this point, the numeral is the ones' complement of the decimal value 5. To obtain the two's complement, 1 is added to the result, giving:

```
11111011 = − 5
```

The result is a signed binary number representing the decimal value −5 in two's complement form. The most significant bit is 1, so the value represented is negative. The most significant bit acts like the sign itself. If it is 1 the number is negative, and 0 means positive. This is what the instructions BPL and BMI were intended for. They are doing nothing but checking the most significant bit of the ACC, and branching according to that.

The two's complement of a negative number is the corresponding positive value. For example, inverting the bits of −5 (above) gives:

```
00000100
```

And adding one gives the final value:

```
00000101 = 5
```

The value of a two's complement binary number can be calculated by adding up the power-of-two weights of the “one” bits, but with a negative weight for the most significant (sign) bit; for example:

```
    1111 1011_2 = -128 + 64 + 32 + 16 + 8 + 0 + 2 + 1 = (-2^7 + 2^6 + ...) =
-5
```

Note that the two's complement of zero is zero: inverting gives all ones, and adding one changes the ones back to zeros (the overflow is ignored). Also the two's complement of the most negative number representable (e.g. a one as the sign bit and all other bits zero) is itself. Hence, there appears to be an 'extra' negative number.

Bitwise Shifting

When a two's complement negative number is shifted to the right (which is the equal as dividing with two), the sign bit must be maintained. However when shifted to the left, a zero is shifted in.

```
    11101100
```

shifting to the right (LSR A):

```
    01110110  notice how the sign bit becomes suddenly positive giving a
false result.
```

To keep the sign of the number the following code has to be applied:

```
    CMP #$80    ;this one will magically copy the most significant bit
(sign) into Carry
    ROR A       ;ROR shifts in Carry from the left, thus keeping our sign
the same
```