

Binary coded decimal addition and subtraction

Addition

Addition is performed digit by digit (not bit by bit), in 4-bit groups, from right to left.

Example:

Consider adding $(+255)_{10}$ and $(+63)_{10}$ in BCD

$$\begin{array}{ccccccc} & & 0 & & 1 & & 0 & & 0 & \leftarrow \text{Carries} \\ & & \underline{0000} & & \underline{0010} & & \underline{0101} & & \underline{0101} & (+255)_{10} \\ & & (0)_{10} & & (2)_{10} & & (5)_{10} & & (5)_{10} & \\ + & & \underline{0000} & & \underline{0000} & & \underline{0110} & & \underline{0011} & (+63)_{10} \\ & & (0)_{10} & & (0)_{10} & & (6)_{10} & & (3)_{10} & \\ \hline & & \underline{0000} & & \underline{0011} & & \underline{0001} & & \underline{1000} & (+318)_{10} \\ & & (0)_{10} & & (3)_{10} & & (1)_{10} & & (8)_{10} & \end{array}$$

Addition is performed on a BCD digit by BCD digit basis (not bit by bit) from right to left.

Subtraction Using BCD

Subtraction is carried out by adding the ten's complement negative of the subtrahend to the minuend.

Ten's complement negative of subtrahend is obtained by adding 1 to the nine's complement negative of the subtrahend.

Example

Consider performing the subtraction operation $(255 - 63 = 192)_{10}$:

This can be considered as

$$(255 + (-63) = 192)_{10}$$

Start by forming the nine's complement of 63:

$$\begin{array}{r} 9999 \\ - 0063 \\ \hline 9936 \end{array}$$

Then add 1 to form the ten's complement

$$\begin{array}{r} 9936 \\ + 0001 \\ \hline 9937 \end{array}$$

	1	1	0	1	0	← Carries
		0000	0010	0101	0101	(+255) ₁₀
+		1001	1001	0011	0111	(-63) ₁₀
	1	0000	0001	1001	0010	(+192) ₁₀
	↑					
	Discard carry					

In ten's complement the number is positive if the left most digit is between 0 and 4 and negative otherwise.

The BCD of 4 and 5 are 0100 and 0101 both have 0 in leftmost, 4 indicate positive number and 5 indicate negative number

In an excess 3 encoding for each digit the left most bit will indicate the sign

BCD Bit Pattern	Normal BCD value	Excess 3 value
0 0 0 0	0	d
0 0 0 1	1	d
0 0 1 0	2	d
0 0 1 1	3	0
0 1 0 0	4	1
0 1 0 1	5	2
0 1 1 0	6	3
0 1 1 1	7	4
1 0 0 0	8	5
1 0 0 1	9	6
1 0 1 0	d	7
1 0 1 1	d	8
1 1 0 0	d	9
1 1 0 1	d	d
1 1 1 0	d	d
1 1 1 1	d	d

Positive numbers

Negative numbers

BCD full adder

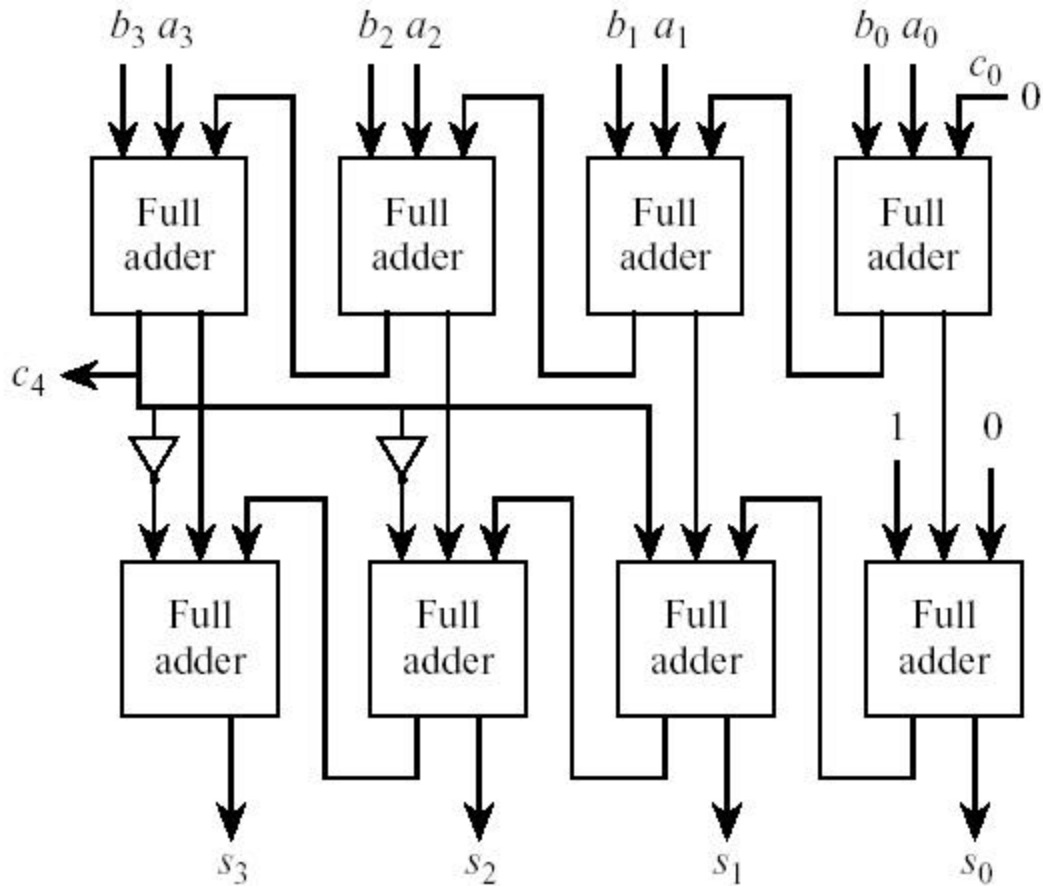
The BCD full adder should sum two BCD digits and a carry in and produce a sum BCD and carry out all using excess 3.

Following figure is a BCD full adder

Circuit adds two base 10 digits represented in BCD.

Example:

Adding 5 and 7 (0101 and 0111) results in 12 (0010 with a carry of 1, and not 1100, which is the binary representation of 12_{10}).



Excess 3 BCD are added in the upper four 2's complement full adder

Each operand is represented in excess 3, the result in excess 6

To restore the result to excess 3, we should subtract 3 from the result, or

We could add 13 to the result and discard the carry

Remember $16 - 3 = 16 + 13$ in four-bit representation (13 and 13 give 29 take 16 out to the next location, left with 13, the 16 taken are only 1 carry)

Two situations:

No carry i.e. $C_4 = 0$

$(13)_{10} = 1101$ is added to the result

If there is a carry $C_4 = 1$

We need to subtracts 10 (equivalent to add 6) from the result beside subtracting 3 (adding 13), So total of subtraction 13 (adding 3) $_{10} = 0011$

See the figure when $C_4 = 1$ then we are adding 0011

BCD subtraction

Ten's Complement Subtraction

The following compares the traditional signed magnitude approach for adding decimal numbers vs. the ten's complement approach, for $(21 - 34 = -13)_{10}$:

$$\begin{array}{r} 0021 \\ + 9966 \\ \hline 9987 \end{array}$$

Ten's Complement

$$\begin{array}{r} 0021 \\ - 0034 \\ \hline - 0013 \end{array}$$

Signed Magnitude

Here the tens's complement negative of 34 is added to 21 result in 9987 in ten's complement, this is $(-34)_{10}$ in signed magnitude.

BCD Floating Point Representation

Consider a base 10 floating point representation with a two digit signed magnitude exponent and an eight digit signed magnitude fraction. On a calculator, a sample entry might look like:

$$-.37100000 \times 10^{-12}$$

We use a ten's complement representation for the exponent, and a base 10 signed magnitude representation for the fraction.

Using ten's complement would be 88 for the exponent and 62900000 for the fraction

A separate sign bit is maintained for the fraction, so that each digit can take on any of the 10 values 0–9 (except for the first digit, which cannot be zero).

We should also represent the exponent in excess 50 (placing the representation for 0 in the middle of the exponents, which range from -50 to +49) to make comparisons easier.

The example above now looks like this

Sign bit: 1

Exponent: 0110 1011

Fraction: 0110 1010 0100 0011 0011 0011 0011 0011

Note that the representation is still in excess 3 binary forms, with two-digit excess 50 exponent.

To add two numbers in this representation, as for a base 2 floating-point representations, we start by adjusting the exponent and fraction of the smaller operand until the exponents of both operands are the same.

After adjusting the smaller fraction, we convert either or both operands from signed magnitude to ten's complement according to whether we are adding or subtracting, and whether the operands are positive or negative, and then perform the addition or subtraction operation.