

# BINARY-CODED DECIMAL (BCD)

## Definition

The **binary-coded decimal (BCD)** is an encoding for decimal numbers in which each digit is represented by its own binary sequence.

## Basics

In computing and electronic systems, **binary-coded decimal (BCD)** is an encoding for decimal numbers in which each digit is represented by its own binary sequence. Its main virtue is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. Its drawbacks are the increased complexity of circuits needed to implement mathematical operations and a relatively inefficient encoding – it occupies more space than a pure binary representation. Even though the importance of BCD has diminished, it is still widely used in financial, commercial, and industrial applications.

In BCD, a digit is usually represented by four bits which, in general, represent the values/digits/characters 0-9. Other bit combinations are sometimes used for sign or other indications.

To BCD-encode a decimal number using the common encoding, each decimal digit is stored in a four-bit nibble.

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Thus, the BCD encoding for the number 127 would be:

0001 0010 0111

Since most computers store data in eight-bit bytes, there are two common ways of storing four-bit BCD digits in those bytes:

- each digit is stored in one byte, and the other four bits are then set to all zeros, all ones (as in the EBCDIC code), or to 0011 (as in the ASCII code)
- two digits are stored in each byte.

Unlike binary encoded numbers, BCD encoded numbers can easily be displayed by mapping each of the nibbles to a different character. Converting a binary encoded number to decimal for display is much harder involving integer multiplication or divide operations.

## BCD in electronics

BCD is very common in electronic systems where a numeric value is to be displayed, especially in systems consisting solely of digital logic, and not containing a microprocessor. By utilising BCD, the manipulation of numerical data for display can be greatly simplified by treating each digit as a separate single sub-circuit. This matches much more closely the physical reality of display hardware—a designer might choose to use a series of separate identical 7-segment displays to build a metering circuit, for example. If the numeric quantity were stored and manipulated as pure binary, interfacing to such a display would require complex circuitry. Therefore, in cases where the calculations are relatively simple working throughout with BCD can lead to a simpler overall system than converting to 'pure' binary.

The same argument applies when hardware of this type uses an embedded microcontroller or other small processor. Often, smaller code results when representing numbers internally in BCD format, since a conversion from or to binary representation can be expensive on such limited processors. For these applications, some small processors feature BCD arithmetic modes, which assist when writing routines that manipulate BCD quantities.

## Packed BCD

A widely used variation of the two-digits-per-byte encoding is called **packed BCD** (or simply **packed decimal**), where numbers are stored with two decimal digits "packed" into one byte each, and the last digit (or nibble) is used as a sign indicator. The preferred sign values are 1100 (hex C) for positive (+) and 1101 (hex D) for negative (−); other allowed signs are 1010 (A) and 1110 (E) for positive and 1011 (B) for negative. Some implementations also provide unsigned BCD values with a sign nibble of 1111 (hex F). In packed BCD, the number +127 is represented as the bytes 00010010 01111100 (hex 12 7C), and −127 as 00010010 01111101 (hex 12 7D).

<b>Sign Digit</b>	<b>BCD</b> <b>8 4 2 1</b>	<b>Sign</b>
<b>A</b>	1 0 1 0	+
<b>B</b>	1 0 1 1	-
<b>C</b>	1 1 0 0	+ (preferred)
<b>D</b>	1 1 0 1	- (preferred)
<b>E</b>	1 1 1 0	+
<b>F</b>	1 1 1 1	+ (unsigned)

Packing four-bit digits and a sign into eight-bit bytes means that an  $n$ -byte packed decimal value (where  $n$  typically ranges from 1 to 15) contains  $2n-1$  decimal digits (which is always an odd number of digits). In other words,  $d$  decimal digits require a packed decimal representation that is  $\frac{1}{2}(d+1)$  bytes wide. For example, a four-byte packed decimal number holds seven decimal digits plus a sign, and can represent values from  $\pm 0,000,000$  to  $\pm 9,999,999$ . In contrast, a four-byte binary two's complement integer can represent values from  $-2,147,483,648$  to  $+2,147,483,647$ .

While packed BCD does not make optimal use of storage (about  $\frac{1}{6}$  of the required memory is wasted), conversion to ASCII, EBCDIC, or the various encodings of Unicode is still trivial, as no arithmetic operations are required. The extra storage requirements are usually offset by the need for the accuracy that fixed-point decimal arithmetic provides. More dense packings of BCD exist which avoid the storage penalty and also need no arithmetic operations for common conversions.

### **Fixed-point packed decimal**

Fixed-point decimal numbers are supported by some programming languages (such as COBOL and PL/1), and provide an implicit decimal point in front of one of the digits. For example, a packed decimal value encoded with the bytes 12 34 56 7C represents the fixed-point value  $+1,234.567$  when the implied decimal point is located between the 4th and 5th digits.

## Higher-density encodings

If a decimal digit requires four bits, then three decimal digits require 12 bits. However, since  $2^{10} > 10^3$ , if three decimal digits are encoded together then only 10 bits are needed. Two such encodings are *Chen-Ho encoding* and *Densely Packed Decimal*. The latter has the advantage that subsets of the encoding encode two digits in the optimal 7 bits and one digit in 4 bits, as in regular BCD.

## Zoned decimal

Some implementations (notably IBM mainframe systems) support **zoned decimal** numeric representations. Each decimal digit is stored in one byte, with the lower four bits encoding the digit in BCD form. The upper four bits, called the "zone" bits, are usually set to a fixed value so that the byte holds a character value corresponding to the digit. EBCDIC systems use a zone value of 1111 (hex F); this yields bytes in the range F0 to F9 (hex), which are the EBCDIC codes for the characters "0" through "9". Similarly, ASCII systems use a zone value of 0011 (hex 3), giving character codes 30 to 39 (hex).

For signed zoned decimal values, the rightmost (least significant) zone nibble holds the sign digit, which is the same set of values that are used for signed packed decimal numbers (see above). Thus a zoned decimal value encoded as the hex bytes F1 F2 D3 represents the signed decimal value −123.

## Fixed-point zone decimal

Some languages (such as COBOL and PL/1) directly support fixed-point zoned decimal values, assigning an implicit decimal point at some location between the decimal digits of a number. For example, given a six-byte signed zoned decimal value with an implied decimal point to the right of the 4th digit, the hex bytes F1 F2 F7 F9 F5 C0 represent the value +1,279.50.

## IBM and BCD

IBM used the terms **binary-coded decimal** and **BCD** for six-bit *alphanumeric* codes that represented numbers, upper-case letters and special characters. Some variation of BCD was used in most early IBM computers, including the IBM 1620, IBM 1400 series, and non-Decimal Architecture members of the IBM 700/7000 series. With the introduction of System/360, IBM replaced BCD with 8-bit EBCDIC.

Bit positions in BCD were usually labelled *B*, *A*, *8*, *4*, *2* and *1*. For encoding digits, *B* and *A* were zero. The letter **A** was encoded (*B,A,I*).

In the 1620 BCD *alphamerics* were encoded using digit pairs, with the "zone" in the even digit and the "digit" in the odd digit. Input/Output translation hardware converted between the internal digit pairs and the external standard six-bit BCD codes.

In the Decimal Architecture IBM 7070, IBM 7072, and IBM 7074 *alphamerics* were encoded using digit pairs (using two-out-of-five code in the digits, **not** BCD) of the 10-digit word, with the "zone" in the left digit and the "digit" in the right digit. Input/Output translation hardware converted between the internal digit pairs and the external standard six-bit BCD codes.

Today, BCD is still heavily used in IBM processors and databases, such as IBM DB2, mainframes and Power6. In these products, the BCD is usually zoned BCD (as in EBCDIC or ASCII), Packed BCD, or 'pure' BCD encoding. All of these are used in within hardware registers and processing units and in software.

## Addition with BCD

To perform addition in BCD, you can first add-up in binary format, and then perform the conversion to BCD afterwards. This conversion involves adding 6 to each group of four digits that has a value of greater-than 9. For example:

- $9+5=14 = [1001] + [0101] = [1110]$  in binary.

However, in BCD, we cannot have a value greater-than 9 (1001) per-nibble. To correct this, one adds 6 to that group:

- $[0000\ 1110] + [0000\ 0110] = [0001\ 0100]$

which gives us two nibbles, [0001] and [0100] which correspond to "1" and "4" respectively. This gives us the 14 in BCD which is the correct result.

## Background

The binary-coded decimal scheme described in this article is the most common encoding, but there are many others.

The method here can be referred to as *Simple Binary-Coded Decimal (SBCD)* or *BCD 8421*. In the headers to the table, the '8 4 2 1' indicates the four bit weights; note that in the 5<sup>th</sup> column two of the weights are negative.

The following table represents decimal digits from 0 to 9 in various BCD systems:

Digit	BCD	Excess-3 or Stibitz Code	BCD 2 4 2 1 or Aiken Code	BCD	IBM 702 IBM 7080	IBM 705 IBM 1401
	8 4 2 1			8 4 -2 -1		
0	0000	0011	0000	0000		1010
1	0001	0100	0001	0111		0001
2	0010	0101	0010	0110		0010
3	0011	0110	0011	0101		0011
4	0100	0111	0100	0100		0100
5	0101	1000	1011	1011		0101
6	0110	1001	1100	1010		0110
7	0111	1010	1101	1001		0111
8	1000	1011	1110	1000		1000
9	1001	1100	1111	1111		1001

## Legal history

In 1972, the U.S. Supreme Court overturned a lower court decision which had allowed a patent for converting BCD encoded numbers to binary on a computer. This was an important case in determining the patentability of software and algorithms.

# Comparison with pure binary

## Advantages

- Scaling by a factor of 10 (or a power of 10) is simple; this is useful when a decimal scaling factor is needed to represent a non-integer quantity (e.g., in financial calculations)
- Rounding at a decimal digit boundary is simpler.
- Alignment of two decimal numbers (for example  $1.3 + 27.08$ ) is a simple, exact, shift
- Conversion to a character form or for display (e.g., to a text-based format such as XML, or to drive signals for a seven-segment display) is a simple per-digit mapping, and can be done in linear ( $O(n)$ ) time. Conversion from pure binary involves relatively complex logic that spans digits, and for large numbers no linear-time conversion algorithm is known (see Binary numeral system#Conversion\_to\_and\_from\_other\_numeral\_systems).
- Some non-integral values, such as 0.2, have a finite place-value representation in decimal but not in binary; consequently a system based on binary place-value representations would introduce a small error representing such a value, which may be compounded by further computation if careful numerical considerations are not made. Note that if computation is not performed on the value this is not an issue, since it suffices to represent it using enough bits that when rounded to the original number of decimal digits the original value is correctly recovered.

## Disadvantages

- Some operations are more complex to implement. Adders require extra logic to cause them to wrap and generate a carry early. 15%–20% more circuitry is needed for BCD add compared to pure binary. Multiplication requires the use of algorithms that are somewhat more complex than shift-mask-add (a binary multiplication, requiring binary shifts and adds or the equivalent, per-digit or group of digits is required)
- Standard BCD requires four bits per digit, roughly 20% more space than a binary encoding. When packed so that three digits are encoded in ten bits, the storage overhead is reduced to about 0.34%, at the expense of an encoding that is unaligned with the 8-bit byte boundaries common on existing hardware, resulting in slower implementations on these systems.

- Practical existing implementations of BCD are typically slower than operations on binary representations, especially on embedded systems, due to limited processor support for native BCD operations.

## Applications

The BIOS in many PCs keeps the date and time in BCD format, probably for historical reasons (it avoided the need for binary to ASCII conversion).

## Representational variations

Various BCD implementations exist that employ other representations for numbers. Programmable calculators manufactured by Texas Instruments, Hewlett-Packard, and others typically employ a floating-point BCD format, typically with two or three digits for the (decimal) exponent. The extra bits of the sign digit may be used to indicate special numeric values, such as infinity, underflow/overflow, and error (a blinking display).

## Alternative encodings

If error in representation and computation is the primary concern, rather than efficiency of conversion to and from display form, a scaled binary representation may be used, which stores a decimal number as a binary-encoded integer and a binary-encoded signed decimal exponent. For example, 0.2 can be represented as  $2 \times 10^{-1}$ . This representation allows rapid multiplication and division, but may require multiplication by a power of 10 during addition and subtraction to align the decimals. It is particularly appropriate for applications with a fixed number of decimal places, which do not require adjustment during addition and subtraction and need not store the exponent explicitly.

Chen-Ho encoding provides a boolean transformation for converting groups of three BCD-encoded digits to and from 10-bit values that can be efficiently encoded in hardware with only 2 or 3 gate delays. Densely Packed Decimal is a similar scheme that deals more efficiently and conveniently with the case where the number of digits is not a multiple of 3.

Source: <http://www.juliantrubin.com/encyclopedia/electronics/bcd.html>