This is a logic circuit that add two decimal numbers to produce a decimal result.

Each breadboard in the circuit serves a different purpose. The two idential boards on the left are the input (with digit display). The numbers are set using the dip switches, with up = 1 and down = 0:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1** | **2** | **3** | **4** | **Value** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |

The wide board at the bottom is a 4-digit binary full adder. It takes the two 4 digit binary numbers and creates a 5-bit binary result.

The middle board on the right converts the binary result into a BCD (Binary Coded Decimal) value. It works like this. If the result is less than 10, then it basically does nothing. When the result is 10 or more, then it marks that the tens digit is 1, and subtracts 10 from the value.

For example, suppose the numbers 7 and 7 are to be added. In binary, this is binary 0111 + binary 0111 = binary 1110. Unfortunately, this produces a number that is not on the table above. Since binary 1110 is greater than 10 (binary 1010), we subtract 10 to produce the smaller digit: 1110 - 1010 = 0100. According to the table, this is 4. So the display shows "14", which is the correct decimal result.

Finally, the top board on the right displays the two-digit result.

2

**V22.0436 - Prof. Grishman**

**Assignment 2: Combinational Circuit -- Decimal Adder**

In this assignment, you are to design a two-digit decimal adder, using a binary-coded-decimal representation, and verify the design by simulating it using a popular procedural language such as C or Pascal.

In a binary-coded-decimal (BCD) representation, each decimal digit is represented by four signals, D[0], D[1], D[2], and D[3]. These are interpreted as a binary number, with D[0] being the least significant digit, so that the digit 5 is represented by D[3]=0, D[2]=1, D[1]=0, and D[0]=1. Bit combinations representing values of ten or greater are not valid BCD patterns.

A two-digit decimal number is represented by two sets of four signals: ONES and TENS. A two-digit decimal adder takes two decimal numbers: A-ONES and A-TENS, and B-ONES and B-TENS (16 bits of input in total); it produces a two-digit sum, SUM-ONES and SUM-TENS, and a single CARRY-OUT bit (9 bits total).

The tricky part is to create a decimal adder circuit for a single decimal digit. You can create a circuit directly from the truth table, but there are more straightforward ways. Specifically, you can perform decimal addition of inputs A and B in three steps:

1. add A and B in a 4-bit binary adder, producing a binary sum BSUM and carry-out COUT1
2. generate a correction CORR as follows: if BSUM>9, CORR=6 else CORR=0 (remember, you have to do this with a Boolean circuit!); this value will be used to *correct* digits which are greater than 9
3. use a second 4-bit binary adder to add BSUM and CORR, producing SUM and a second carry-out COUT2

Finally, you have to produce a carry-out from the decimal digit.

**What to Submit:**

1. A design for the circuit, starting from basic logic gates. Don't draw one big circuit --- show the design at several levels. First, create a full-adder from gates; then, a 4-bit adder from full adders; then, a one-digit decimal adder from 4-bit adders and individual gates; finally, a two-digit decimal adder from one-digit adders. This emphasizes the structure of the design and means you have much less to draw.
2. Determine the propagation delay of the circuit as a multiple of the propagation delay of an individual gate.
3. Create a simulation of the circuit using a program. Use the same hierarchical approach you used in designing the circuit. Submit a listing of this program.
4. Write a program to test this circuit. Submit a listing of this program, and the test output. Can you exhaustively test this circuit?

All this should be bundled together and submitted not later than ***September 29***. Late assignments are penalized 10% for each weekday late.

[[Logo](https://meta.wikimedia.org/wiki/Special:MyLanguage/Terms_of_use/Creative_Commons_4.0)](https://meta.wikimedia.org/wiki/Special:MyLanguage/Terms_of_use/Creative_Commons_4.0)

[Should Wikimedia content be licensed under the latest Creative Commons license?](https://meta.wikimedia.org/wiki/Special:MyLanguage/Terms_of_use/Creative_Commons_4.0)

[Participate in the discussion between October 5 and November 8!](https://meta.wikimedia.org/wiki/Special:MyLanguage/Terms_of_use/Creative_Commons_4.0)

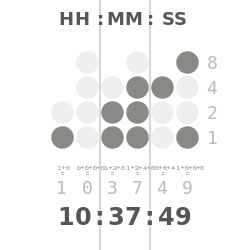
[[](https://meta.wikimedia.org/wiki/Special:MyLanguage/Terms_of_use/Creative_Commons_4.0)](https://meta.wikimedia.org/wiki/Special:MyLanguage/Terms_of_use/Creative_Commons_4.0)

[⧼centralnotice-shared-close⧽](https://en.wikipedia.org/wiki/Binary-coded_decimal)

# Binary-coded decimal

From Wikipedia, the free encyclopedia

"BCD code" redirects here. For BCD character sets, see [BCD (character encoding)](https://en.wikipedia.org/wiki/BCD_%28character_encoding%29).

[](https://en.wikipedia.org/wiki/File:Binary_clock.svg)

A [binary clock](https://en.wikipedia.org/wiki/Binary_clock) might use [LEDs](https://en.wikipedia.org/wiki/Light-emitting_diode) to express binary values. In this clock, each column of LEDs shows a binary-coded decimal numeral of the traditional [sexagesimal](https://en.wikipedia.org/wiki/Sexagesimal) time.

In [computing](https://en.wikipedia.org/wiki/Computing) and [electronic](https://en.wikipedia.org/wiki/Electronics) systems, **binary-coded decimal** (**BCD**) is a class of [binary](https://en.wikipedia.org/wiki/Binary_numeral_system) encodings of [decimal](https://en.wikipedia.org/wiki/Decimal) numbers where each decimal [digit](https://en.wikipedia.org/wiki/Numerical_digit) is represented by a fixed number of [bits](https://en.wikipedia.org/wiki/Bit), usually four or eight. Special bit patterns are sometimes used for a [sign](https://en.wikipedia.org/wiki/Sign_%28mathematics%29) or for other indications (e.g., error or overflow).

In byte-oriented systems (i.e. most modern computers), the term *unpacked* BCD[[1]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-1) usually implies a full [byte](https://en.wikipedia.org/wiki/Byte) for each digit (often including a sign), whereas *packed* BCD typically encodes two decimal digits within a single byte by taking advantage of the fact that four bits are enough to represent the range 0 to 9. The precise 4-bit encoding may vary however, for technical reasons, see [Excess-3](https://en.wikipedia.org/wiki/Excess-3) for instance. The ten states representing a BCD decimal digit are sometimes called *tetrades* (for the [nibble](https://en.wikipedia.org/wiki/Nibble) typically needed to hold them also known as [tetrade](https://en.wikipedia.org/wiki/Tetrade_%28computing%29)) with those [don't care](https://en.wikipedia.org/wiki/Don%27t_care)-states unused named [*pseudo-tetrad(e)s*](https://en.wikipedia.org/wiki/Pseudo-tetrade) ([de](https://de.wikipedia.org/wiki/Pseudotetrade))[[2]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Schneider_1986-2)[[3]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Steinbuch-Weber_1974-3)[[4]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Tietze-Schenk_2013-4)[[5]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Kowalski_1070-5) or *pseudo-decimal digit*[[6]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Ferretti_2013-6)).[[nb 1]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Pseudo-tetrades-7)

BCD's main virtue is its more accurate representation and rounding of decimal quantities as well as an ease of conversion into human-readable representations, in comparison to binary [positional systems](https://en.wikipedia.org/wiki/Positional_system). BCD's principal drawbacks are a small increase in the complexity of the circuits needed to implement basic arithmetics and a slightly less dense storage.

BCD was used in many early [decimal computers](https://en.wikipedia.org/wiki/Decimal_computer), and is implemented in the instruction set of machines such as the [IBM System/360](https://en.wikipedia.org/wiki/IBM_System/360) series and its descendants and [Digital's](https://en.wikipedia.org/wiki/Digital_Equipment_Corporation) [VAX](https://en.wikipedia.org/wiki/VAX). Although BCD *per se* is not as widely used as in the past and is no longer implemented in computers' instruction sets[[*dubious*](https://en.wikipedia.org/wiki/Wikipedia:Disputed_statement) *–* [*discuss*](https://en.wikipedia.org/wiki/Talk:Binary-coded_decimal#Dubious)], decimal [fixed-point](https://en.wikipedia.org/wiki/Fixed-point_arithmetic) and [floating-point](https://en.wikipedia.org/wiki/Floating-point) formats are still important and continue to be used in financial, commercial, and industrial computing, where subtle conversion and [fractional](https://en.wikipedia.org/wiki/Fraction_%28mathematics%29) [rounding](https://en.wikipedia.org/wiki/Rounding) [errors](https://en.wikipedia.org/wiki/Round-off_error) that are inherent in floating point binary representations cannot be tolerated.[[7]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Cowlishaw_GDA-8)

## Contents

* [1 Basics](https://en.wikipedia.org/wiki/Binary-coded_decimal#Basics)
* [2 BCD in electronics](https://en.wikipedia.org/wiki/Binary-coded_decimal#BCD_in_electronics)
* [3 Packed BCD](https://en.wikipedia.org/wiki/Binary-coded_decimal#Packed_BCD)
  + [3.1 Fixed-point packed decimal](https://en.wikipedia.org/wiki/Binary-coded_decimal#Fixed-point_packed_decimal)
  + [3.2 Higher-density encodings](https://en.wikipedia.org/wiki/Binary-coded_decimal#Higher-density_encodings)
* [4 Zoned decimal](https://en.wikipedia.org/wiki/Binary-coded_decimal#Zoned_decimal)
  + [4.1 EBCDIC zoned decimal conversion table](https://en.wikipedia.org/wiki/Binary-coded_decimal#EBCDIC_zoned_decimal_conversion_table)
  + [4.2 Fixed-point zoned decimal](https://en.wikipedia.org/wiki/Binary-coded_decimal#Fixed-point_zoned_decimal)
* [5 IBM and BCD](https://en.wikipedia.org/wiki/Binary-coded_decimal#IBM_and_BCD)
* [6 Other computers and BCD](https://en.wikipedia.org/wiki/Binary-coded_decimal#Other_computers_and_BCD)
* [7 Addition with BCD](https://en.wikipedia.org/wiki/Binary-coded_decimal#Addition_with_BCD)
* [8 Subtraction with BCD](https://en.wikipedia.org/wiki/Binary-coded_decimal#Subtraction_with_BCD)
* [9 Background](https://en.wikipedia.org/wiki/Binary-coded_decimal#Background)
* [10 Legal history](https://en.wikipedia.org/wiki/Binary-coded_decimal#Legal_history)
* [11 Comparison with pure binary](https://en.wikipedia.org/wiki/Binary-coded_decimal#Comparison_with_pure_binary)
  + [11.1 Advantages](https://en.wikipedia.org/wiki/Binary-coded_decimal#Advantages)
  + [11.2 Disadvantages](https://en.wikipedia.org/wiki/Binary-coded_decimal#Disadvantages)
* [12 Application](https://en.wikipedia.org/wiki/Binary-coded_decimal#Application)
* [13 Representational variations](https://en.wikipedia.org/wiki/Binary-coded_decimal#Representational_variations)
  + [13.1 Signed variations](https://en.wikipedia.org/wiki/Binary-coded_decimal#Signed_variations)
  + [13.2 Telephony Binary Coded Decimal (TBCD)](https://en.wikipedia.org/wiki/Binary-coded_decimal#Telephony_Binary_Coded_Decimal_.28TBCD.29)
* [14 Alternative encodings](https://en.wikipedia.org/wiki/Binary-coded_decimal#Alternative_encodings)
* [15 See also](https://en.wikipedia.org/wiki/Binary-coded_decimal#See_also)
* [16 Notes](https://en.wikipedia.org/wiki/Binary-coded_decimal#Notes)
* [17 References](https://en.wikipedia.org/wiki/Binary-coded_decimal#References)
* [18 Further reading](https://en.wikipedia.org/wiki/Binary-coded_decimal#Further_reading)
* [19 External links](https://en.wikipedia.org/wiki/Binary-coded_decimal#External_links)

## Basics

BCD takes advantage of the fact that any one decimal numeral can be represented by a four bit pattern. The most obvious way of encoding digits is "natural BCD" (NBCD), where each decimal digit is represented by its corresponding four-bit binary value, as shown in the following table. This is also called "8421" encoding.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Decimal digit** | **BCD** | | | |
| **8** | **4** | **2** | **1** |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 |
| 6 | 0 | 1 | 1 | 0 |
| 7 | 0 | 1 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 |
| 9 | 1 | 0 | 0 | 1 |

Other encodings are also used, including so-called "4221" and "7421" — named after the weighting used for the bits — and "[excess-3](https://en.wikipedia.org/wiki/Excess-3)".[[8]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-9) For example, the BCD digit 6, '0110'b in 8421 notation, is '1100'b in 4221 (two encodings are possible), '0110'b in 7421, and '1001'b (6+3=9) in excess-3.

As most computers deal with data in 8-bit [bytes](https://en.wikipedia.org/wiki/Byte), it is possible to use one of the following methods to encode a BCD number:

* **Unpacked**: each numeral is encoded into one byte, with four bits representing the numeral and the remaining bits having no significance.
* **Packed**: two numerals are encoded into a single byte, with one numeral in the least significant [nibble](https://en.wikipedia.org/wiki/Nibble) (bits 0 through 3) and the other numeral in the most significant nibble (bits 4 through 7).

As an example, encoding the decimal number **91** using unpacked BCD results in the following binary pattern of two bytes:

Decimal: 9 1

Binary : 0000 1001 0000 0001

In packed BCD, the same number would fit into a single byte:

Decimal: 9 1

Binary : 1001 0001

Hence the numerical range for one unpacked BCD byte is zero through nine inclusive, whereas the range for one packed BCD is zero through ninety-nine inclusive.

To represent numbers larger than the range of a single byte any number of contiguous bytes may be used.  For example, to represent the decimal number **12345** in packed BCD, using [big-endian](https://en.wikipedia.org/wiki/Big-endian) format, a program would encode as follows:

Decimal: 1 2 3 4 5

Binary : 0000 0001 0010 0011 0100 0101

Note that the most significant nibble of the most significant byte is zero, implying that the number is in actuality **012345**.  Also note how packed BCD is more efficient in storage usage as compared to unpacked BCD; encoding the same number (with the leading zero) in unpacked format would consume twice the storage.

[Shifting](https://en.wikipedia.org/wiki/Logical_shift) and [masking](https://en.wikipedia.org/wiki/Mask_%28computing%29) operations are used to pack or unpack a packed BCD digit.  Other [logical operations](https://en.wikipedia.org/wiki/Bitwise_operation) are used to convert a numeral to its equivalent bit pattern or reverse the process.

## 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 employing 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 [seven-segment displays](https://en.wikipedia.org/wiki/Seven-segment_display) 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 and from binary. Most pocket calculators do all their calculations in BCD.

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.[[9]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-10)[[10]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-11)

## Packed BCD

In **Packed BCD** (or simply **packed decimal**), each of the two [nibbles](https://en.wikipedia.org/wiki/Nibble) of each byte represent a decimal digit. Packed BCD has been in use since at least the 1960s and is implemented in all IBM mainframe hardware since then. Most implementations are [big endian](https://en.wikipedia.org/wiki/Big_endian), i.e. with the more significant digit in the upper half of each byte, and with the leftmost byte (residing at the lowest memory address) containing the most significant digits of the packed decimal value. The lower nibble of the rightmost byte is usually used as the sign flag, although some unsigned representations lack a sign flag. As an example, a 4-byte value consists of 8 nibbles, wherein the upper 7 nibbles store the digits of a 7-digit decimal value and the lowest nibble indicates the sign of the decimal integer value.

Standard sign values are 1100 ([hex](https://en.wikipedia.org/wiki/Hexadecimal) C) for positive (+) and 1101 (D) for negative (−). This convention comes from the zone field for [EBCDIC](https://en.wikipedia.org/wiki/EBCDIC) characters and the [signed overpunch](https://en.wikipedia.org/wiki/Signed_overpunch) representation. Other allowed signs are 1010 (A) and 1110 (E) for positive and 1011 (B) for negative. IBM System/360 processors will use the 1010 (A) and 1011 (B) signs if the A bit is set in the PSW, for the ASCII-8 standard that never passed. Most implementations also provide unsigned BCD values with a sign nibble of 1111 (F).[[11]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-IBM_POP-12)[[12]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-DEC_PDP11-13)[[13]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-DEC_VAX11-14) ILE RPG uses 1111 (F) for positive and 1101 (D) for negative.[[14]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-15) These match the EBCDIC zone for digits without a sign overpunch. In packed BCD, the number 127 is represented by 0001 0010 0111 1100 (127C) and −127 is represented by 0001 0010 0111 1101 (127D). Burroughs systems used 1101 (D) for negative, and any other value is considered a positive sign value (the processors will normalize a positive sign to 1100 (C)).

|  |  |  |  |
| --- | --- | --- | --- |
| **Sign Digit** | **BCD 8 4 2 1** | **Sign** | **Notes** |
| **A** | 1 0 1 0 | **+** |  |
| **B** | 1 0 1 1 | **−** |  |
| **C** | 1 1 0 0 | **+** | Preferred |
| **D** | 1 1 0 1 | **−** | Preferred |
| **E** | 1 1 1 0 | **+** |  |
| **F** | 1 1 1 1 | **+** | Unsigned |

No matter how many bytes wide a [word](https://en.wikipedia.org/wiki/Word_%28data_type%29) is, there are always an even number of nibbles because each byte has two of them. Therefore, a word of *n* bytes can contain up to (2*n*)−1 decimal digits, which is always an odd number of digits. A decimal number with *d* digits requires ½(*d*+1) bytes of storage space.

For example, a 4-byte (32-bit) word can hold seven decimal digits plus a sign, and can represent values ranging from ±9,999,999. Thus the number −1,234,567 is 7 digits wide and is encoded as:

0001 0010 0011 0100 0101 0110 0111 1101

*1 2 3 4 5 6 7 −*

(Note that, like character strings, the first byte of the packed decimal – with the most significant two digits – is usually stored in the lowest address in memory, independent of the [endianness](https://en.wikipedia.org/wiki/Endianness) of the machine.)

In contrast, a 4-byte binary [two's complement](https://en.wikipedia.org/wiki/Two%27s_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 1/6 of the memory used is wasted), conversion to [ASCII](https://en.wikipedia.org/wiki/ASCII), [EBCDIC](https://en.wikipedia.org/wiki/EBCDIC), or the various encodings of [Unicode](https://en.wikipedia.org/wiki/Unicode) is still trivial, as no arithmetic operations are required. The extra storage requirements are usually offset by the need for the accuracy and compatibility with calculator or hand calculation that fixed-point decimal arithmetic provides. Denser packings of [BCD](https://en.wikipedia.org/wiki/BCD_%28character_encoding%29) exist which avoid the storage penalty and also need no arithmetic operations for common conversions.

Packed BCD is supported in the [COBOL](https://en.wikipedia.org/wiki/COBOL) programming language as the "COMPUTATIONAL-3" (an IBM extension adopted by many other compiler vendors) or "PACKED-DECIMAL" (part of the 1985 COBOL standard) data type. It is supported in [PL/I](https://en.wikipedia.org/wiki/PL/I) as "FIXED DECIMAL". Besides the IBM System/360 and later compatible mainframes, packed BCD is implemented in the native instruction set of the original [VAX](https://en.wikipedia.org/wiki/VAX) processors from [Digital Equipment Corporation](https://en.wikipedia.org/wiki/Digital_Equipment_Corporation) and some models of the [SDS Sigma series](https://en.wikipedia.org/wiki/SDS_Sigma_series) mainframes, and is the native format for the [Burroughs Corporation](https://en.wikipedia.org/wiki/Burroughs_Corporation) Medium Systems line of mainframes (descended from the 1950s Electrodata 200 series).

[Ten's complement](https://en.wikipedia.org/wiki/Ten%27s_complement) representations for negative numbers offer an alternative approach to encoding the sign of packed (and other) BCD numbers. In this case, positive numbers always have a most significant digit between 0 and 4 (inclusive), while negative numbers are represented by the 10's complement of the corresponding positive number. As a result, this system allows for, a 32-bit packed BCD numbers to range from -50,000,000 to 49,999,999, and -1 is represented as 99999999. (As with [two's complement](https://en.wikipedia.org/wiki/Two%27s_complement) binary numbers, the range is not symmetric about zero.)

### Fixed-point packed decimal

[Fixed-point](https://en.wikipedia.org/wiki/Fixed-point_arithmetic) decimal numbers are supported by some programming languages (such as [COBOL](https://en.wikipedia.org/wiki/COBOL) and [PL/I](https://en.wikipedia.org/wiki/PL/I)). These languages allow the programmer to specify 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:

12 34 56 7C

*12 34.56 7+*

The decimal point is not actually stored in memory, as the packed BCD storage format does not provide for it. Its location is simply known to the compiler and the generated code acts accordingly for the various arithmetic operations.

### Higher-density encodings

If a decimal digit requires four bits, then three decimal digits require 12 bits. However, since 210 (1,024) is greater than 103 (1,000), if three decimal digits are encoded together, only 10 bits are needed. Two such encodings are [*Chen-Ho encoding*](https://en.wikipedia.org/wiki/Chen-Ho_encoding) and [*Densely Packed Decimal*](https://en.wikipedia.org/wiki/Densely_Packed_Decimal) (DPD). The latter has the advantage that subsets of the encoding encode two digits in the optimal seven bits and one digit in four bits, as in regular BCD.

## Zoned decimal

Some implementations, for example [IBM](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/EBCDIC) codes for the characters "0" through "9". Similarly, [ASCII](https://en.wikipedia.org/wiki/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:

F1 F2 D3

*1 2 −3*

### EBCDIC zoned decimal conversion table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BCD Digit** | **Hexadecimal** | | | | **EBCDIC Character** | | | |
| 0+ | C0 | A0 | E0 | F0 | { (\*) |  | \ (\*) | 0 |
| 1+ | C1 | A1 | E1 | F1 | A | ~ (\*) |  | 1 |
| 2+ | C2 | A2 | E2 | F2 | B | s | S | 2 |
| 3+ | C3 | A3 | E3 | F3 | C | t | T | 3 |
| 4+ | C4 | A4 | E4 | F4 | D | u | U | 4 |
| 5+ | C5 | A5 | E5 | F5 | E | v | V | 5 |
| 6+ | C6 | A6 | E6 | F6 | F | w | W | 6 |
| 7+ | C7 | A7 | E7 | F7 | G | x | X | 7 |
| 8+ | C8 | A8 | E8 | F8 | H | y | Y | 8 |
| 9+ | C9 | A9 | E9 | F9 | I | z | Z | 9 |
| 0− | D0 | B0 |  |  | }  (\*) | ^  (\*) |  |  |
| 1− | D1 | B1 |  |  | J |  |  |  |
| 2− | D2 | B2 |  |  | K |  |  |  |
| 3− | D3 | B3 |  |  | L |  |  |  |
| 4− | D4 | B4 |  |  | M |  |  |  |
| 5− | D5 | B5 |  |  | N |  |  |  |
| 6− | D6 | B6 |  |  | O |  |  |  |
| 7− | D7 | B7 |  |  | P |  |  |  |
| 8− | D8 | B8 |  |  | Q |  |  |  |
| 9− | D9 | B9 |  |  | R |  |  |  |

(\*) *Note: These characters vary depending on the local character* [*code page*](https://en.wikipedia.org/wiki/Code_page) *setting.*

### Fixed-point zoned decimal

Some languages (such as [COBOL](https://en.wikipedia.org/wiki/COBOL) and [PL/I](https://en.wikipedia.org/wiki/PL/I)) 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 fourth digit, the hex bytes F1 F2 F7 F9 F5 C0 represent the value +1,279.50:

F1 F2 F7 F9 F5 C0

*1 2 7 9. 5 +0*

## IBM and BCD

Main article: [BCDIC](https://en.wikipedia.org/wiki/BCDIC)

[IBM](https://en.wikipedia.org/wiki/IBM) used the terms [*Binary-Coded Decimal Interchange Code*](https://en.wikipedia.org/wiki/Binary-Coded_Decimal_Interchange_Code) (BCDIC, sometimes just called BCD), for 6-bit [*alphanumeric*](https://en.wikipedia.org/wiki/Alphanumeric) codes that represented numbers, upper-case letters and special characters. Some variation of BCDIC *alphamerics* is used in most early IBM computers, including the [IBM 1620](https://en.wikipedia.org/wiki/IBM_1620), [IBM 1400 series](https://en.wikipedia.org/wiki/IBM_1400_series), and non-[Decimal Architecture](https://en.wikipedia.org/wiki/IBM_700/7000_series#Decimal_architecture_.287070.2F7072.2F7074.29) members of the [IBM 700/7000 series](https://en.wikipedia.org/wiki/IBM_700/7000_series).

The [IBM 1400 series](https://en.wikipedia.org/wiki/IBM_1400_series) are character-addressable machines, each location being six bits labeled *B, A, 8, 4, 2* and *1,* plus an odd parity check bit (*C*) and a word mark bit (*M*). For encoding digits *1* through *9*, *B* and *A* are zero and the digit value represented by standard 4-bit BCD in bits *8* through *1*. For most other characters bits *B* and *A* are derived simply from the "12", "11", and "0" "zone punches" in the punched card character code, and bits *8* through *1* from the *1* through *9* punches. A "12 zone" punch set both *B* and *A*, an "11 zone" set *B*, and a "0 zone" (a 0 punch combined with any others) set *A*. Thus the letter **A**, which is *(12,1)* in the punched card format, is encoded *(B,A,1)*. The currency symbol **$**, *(11,8,3)* in the punched card, was encoded in memory as *(B,8,2,1)*. This allowed the circuitry to convert between the punched card format and the internal storage format to be very simple with only a few special cases. One important special case is digit *0*, represented by a lone *0* punch in the card, and *(8,2)* in core memory. [[15]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-16)

The memory of the [IBM 1620](https://en.wikipedia.org/wiki/IBM_1620) is organized into 6-bit addressable digits, the usual *8, 4, 2, 1* plus *F*, used as a flag bit and *C*, an odd parity check bit. BCD *alphamerics* are encoded using digit pairs, with the "zone" in the even-addressed digit and the "digit" in the odd-addressed digit, the "zone" being related to the *12*, *11*, and *0* "zone punches" as in the 1400 series. Input/Output translation hardware converted between the internal digit pairs and the external standard 6-bit BCD codes.

In the Decimal Architecture [IBM 7070](https://en.wikipedia.org/wiki/IBM_7070), [IBM 7072](https://en.wikipedia.org/wiki/IBM_7072), and [IBM 7074](https://en.wikipedia.org/wiki/IBM_7074) *alphamerics* are encoded using digit pairs (using [two-out-of-five code](https://en.wikipedia.org/wiki/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 6-bit BCD codes.

With the introduction of [System/360](https://en.wikipedia.org/wiki/System/360), IBM expanded 6-bit BCD *alphamerics* to 8-bit [EBCDIC](https://en.wikipedia.org/wiki/EBCDIC), allowing the addition of many more characters (e.g., lowercase letters). A variable length Packed BCD *numeric* data type is also implemented, providing machine instructions that perform arithmetic directly on packed decimal data.

On the [IBM 1130](https://en.wikipedia.org/wiki/IBM_1130) and [1800](https://en.wikipedia.org/wiki/IBM_1800), packed BCD is supported in software by IBM's Commercial Subroutine Package.

Today, BCD data is still heavily used in IBM processors and databases, such as [IBM DB2](https://en.wikipedia.org/wiki/IBM_DB2), mainframes, and [Power6](https://en.wikipedia.org/wiki/Power6). In these products, the BCD is usually zoned BCD (as in EBCDIC or ASCII), Packed BCD (two decimal digits per [byte](https://en.wikipedia.org/wiki/Byte)), or "pure" BCD encoding (one decimal digit stored as BCD in the low four bits of each byte). All of these are used within hardware registers and processing units, and in software. To convert packed decimals in EBCDIC table unloads to readable numbers, you can use the OUTREC FIELDS mask of the JCL utility DFSORT.[[16]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-17)

## Other computers and BCD

The [Digital Equipment Corporation](https://en.wikipedia.org/wiki/Digital_Equipment_Corporation) [VAX-11](https://en.wikipedia.org/wiki/VAX) series includes [instructions](https://en.wikipedia.org/wiki/Instruction_set) that can perform arithmetic directly on packed BCD data and convert between packed BCD data and other integer representations.[[13]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-DEC_VAX11-14) The VAX's packed BCD format is compatible with that on IBM System/360 and IBM's later compatible processors. The MicroVAX and later VAX implementations dropped this ability from the CPU but retained code compatibility with earlier machines by implementing the missing instructions in an operating system-supplied software library. This is invoked automatically via exception handling when the no longer implemented instructions are encountered, so that programs using them can execute without modification on the newer machines.

The [Intel](https://en.wikipedia.org/wiki/Intel) [x86](https://en.wikipedia.org/wiki/X86) architecture found on Intel systems supports a unique 18-digit (ten-byte) BCD format that can be loaded into and stored from the floating point registers, and computations can be performed there.

In more recent computers such capabilities are almost always implemented in software rather than the CPU's instruction set, but BCD numeric data is still extremely common in commercial and financial applications. There are tricks for implementing packed BCD and zoned decimal add or subtract operations using short but difficult to understand sequences of word-parallel logic and binary arithmetic operations.[[17]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-Jones_AT-18) For example, the following code (written in [C](https://en.wikipedia.org/wiki/C_%28programming_language%29)) computes an unsigned 8-digit packed BCD add using 32-bit binary operations:

uint32\_t BCDadd(uint32\_t a,uint32\_t b)

{

uint32\_t t1, t2; // unsigned 32-bit intermediate values

t1 = a + 0x06666666;

t2 = t1 ^ b; // sum without carry propagation

t1 = t1 + b; // provisional sum

t2 = t1 ^ t2; // all the binary carry bits

t2 = ~t2 & 0x11111110; // just the BCD carry bits

t2 = (t2 >> 2) | (t2 >> 3); // correction

return t1 - t2; // corrected BCD sum

}

## Addition with BCD

It is possible to perform [addition](https://en.wikipedia.org/wiki/Addition) in BCD by first adding in binary, and then converting to BCD afterwards. Conversion of the simple sum of two digits can be done by adding 6 (that is, 16 – 10) when the five-bit result of adding a pair of digits has a value greater than 9. For example:

1001 + 1000 = 10001

9 + 8 = 17

Note that 10001 is the binary, not decimal, representation of the desired result. Also note that it cannot fit in a 4-bit number. In BCD as in decimal, there cannot exist a value greater than 9 (1001) per digit. To correct this, 6 (0110) is added to that sum and then the result is treated as two [nibbles](https://en.wikipedia.org/wiki/Nibble):

10001 + 0110 = 00010111 => 0001 0111

17 + 6 = 23 1 7

The two nibbles of the result, 0001 and 0111, correspond to the digits "1" and "7". This yields "17" in BCD, which is the correct result.

This technique can be extended to adding multiple digits by adding in groups from right to left, propagating the second digit as a carry, always comparing the 5-bit result of each digit-pair sum to 9. Some CPUs provide a [half-carry flag](https://en.wikipedia.org/wiki/Half-carry_flag) to facilitate BCD arithmetic adjustments following binary addition and subtraction operations.

## Subtraction with BCD

Subtraction is done by adding the [ten's complement](https://en.wikipedia.org/wiki/Ten%27s_complement) of the [subtrahend](https://en.wikipedia.org/wiki/Subtrahend). To represent the sign of a number in BCD, the number 0000 is used to represent a [positive number](https://en.wikipedia.org/wiki/Positive_number), and 1001 is used to represent a [negative number](https://en.wikipedia.org/wiki/Negative_number). The remaining 14 combinations are invalid signs. To illustrate signed BCD subtraction, consider the following problem: 357 − 432.

In signed BCD, 357 is 0000 0011 0101 0111. The [ten's complement](https://en.wikipedia.org/wiki/Ten%27s_complement) of 432 can be obtained by taking the [nine's complement](https://en.wikipedia.org/wiki/Nine%27s_complement) of 432, and then adding one. So, 999 − 432 = 567, and 567 + 1 = 568. By preceding 568 in BCD by the negative sign code, the number −432 can be represented. So, −432 in signed BCD is 1001 0101 0110 1000.

Now that both numbers are represented in signed BCD, they can be added together:

0000 0011 0101 0111

0 3 5 7

+ 1001 0101 0110 1000

9 5 6 8

= 1001 1000 1011 1111

9 8 11 15

Since BCD is a form of decimal representation, several of the digit sums above are invalid. In the event that an invalid entry (any BCD digit greater than 1001) exists, 6 is added to generate a carry bit and cause the sum to become a valid entry. The reason for adding 6 is that there are 16 possible 4-bit BCD values (since 24 = 16), but only 10 values are valid (0000 through 1001). So adding 6 to the invalid entries results in the following:

1001 1000 1011 1111

9 8 11 15

+ 0000 0000 0110 0110

0 0 6 6

= 1001 1001 0010 0101

9 9 2 5

Thus the result of the subtraction is 1001 1001 0010 0101 (-925). To check the answer, note that the first digit is 9, which means negative. This seems to be correct, since 357 − 432 should result in a negative number. To check the rest of the digits, represent them in decimal. 1001 0010 0101 is 925. The [ten's complement](https://en.wikipedia.org/wiki/Ten%27s_complement) of 925 is 1000 − 925 = 999 − 925 + 1 = 074 + 1 = 75, so the calculated answer is −75. To check, perform standard subtraction to verify that 357 − 432 is −75.

Note that in the event that there are a different number of nibbles being added together (such as 1053 − 122), the number with the fewest number of digits must first be padded with zeros before taking the [ten's complement](https://en.wikipedia.org/wiki/Ten%27s_complement) or subtracting. So, with 1053 − 122, 122 would have to first be represented as 0122, and the [ten's complement](https://en.wikipedia.org/wiki/Ten%27s_complement) of 0122 would have to be calculated.

## 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 weight of each bit shown; note that in the fifth column, "BCD 8 4 −2 −1", two of the weights are negative. Both ASCII and EBCDIC character codes for the digits are examples of zoned BCD, and are also shown in the table.

The following table represents [decimal](https://en.wikipedia.org/wiki/Decimal) digits from 0 to 9 in various BCD systems:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Digit** | **BCD 8 4 2 1** | [**Stibitz**](https://en.wikipedia.org/wiki/George_Stibitz)**Code or** [**Excess-3**](https://en.wikipedia.org/wiki/Excess-3) | [**Aiken**](https://en.wikipedia.org/wiki/Howard_Aiken)**Code or BCD 2 4 2 1** | **BCD 8 4 −2 −1** | [**IBM 702**](https://en.wikipedia.org/wiki/IBM_702)**,** [**IBM 705**](https://en.wikipedia.org/wiki/IBM_705)**,** [**IBM 7080**](https://en.wikipedia.org/wiki/IBM_7080)**,** [**IBM 1401**](https://en.wikipedia.org/wiki/IBM_1401) **8 4 2 1** | [**ASCII**](https://en.wikipedia.org/wiki/ASCII) **0000 8421** | [**EBCDIC**](https://en.wikipedia.org/wiki/EBCDIC) **0000 8421** |
| **0** | 0000 | 0011 | 0000 | 0000 | 1010 | 0011 0000 | 1111 0000 |
| **1** | 0001 | 0100 | 0001 | 0111 | 0001 | 0011 0001 | 1111 0001 |
| **2** | 0010 | 0101 | 0010 | 0110 | 0010 | 0011 0010 | 1111 0010 |
| **3** | 0011 | 0110 | 0011 | 0101 | 0011 | 0011 0011 | 1111 0011 |
| **4** | 0100 | 0111 | 0100 | 0100 | 0100 | 0011 0100 | 1111 0100 |
| **5** | 0101 | 1000 | 1011 | 1011 | 0101 | 0011 0101 | 1111 0101 |
| **6** | 0110 | 1001 | 1100 | 1010 | 0110 | 0011 0110 | 1111 0110 |
| **7** | 0111 | 1010 | 1101 | 1001 | 0111 | 0011 0111 | 1111 0111 |
| **8** | 1000 | 1011 | 1110 | 1000 | 1000 | 0011 1000 | 1111 1000 |
| **9** | 1001 | 1100 | 1111 | 1111 | 1001 | 0011 1001 | 1111 1001 |

## Legal history

In the 1972 case [Gottschalk v. Benson](https://en.wikipedia.org/wiki/Gottschalk_v._Benson), 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

* Many non-integral values, such as decimal 0.2, have an infinite place-value representation in binary (.001100110011...) but have a finite place-value in binary-coded decimal (0.0010). Consequently, a system based on binary-coded decimal representations of decimal fractions avoids errors representing and calculating such values.
* 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](https://en.wikipedia.org/wiki/Rounding) at a decimal digit boundary is simpler. Addition and subtraction in decimal does not require rounding.
* 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](https://en.wikipedia.org/wiki/XML), or to drive signals for a [seven-segment display](https://en.wikipedia.org/wiki/Seven-segment_display)) is a simple per-digit mapping, and can be done in linear ([O](https://en.wikipedia.org/wiki/Big-O_notation)(*n*)) time. Conversion from pure [binary](https://en.wikipedia.org/wiki/Binary_numeral_system) involves relatively complex logic that spans digits, and for large numbers no linear-time conversion algorithm is known (see [Binary numeral system](https://en.wikipedia.org/wiki/Binary_numeral_system#Conversion_to_and_from_other_numeral_systems)).

### Disadvantages

* Some operations are more complex to implement. [Adders](https://en.wikipedia.org/wiki/Adder_%28electronics%29) require extra logic to cause them to wrap and generate a carry early. 15–20 percent more circuitry is needed for BCD add compared to pure binary.[[*citation needed*](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed)] Multiplication requires the use of algorithms that are somewhat more complex than shift-mask-add (a [binary multiplication](https://en.wikipedia.org/wiki/Binary_numeral_system#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 percent more space than a binary encoding (the ratio of 4 bits to log210 bits is 1.204). When packed so that three digits are encoded in ten bits, the storage overhead is greatly reduced, 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,[*[citation needed](https://en.wikipedia.org/wiki/Wikipedia:Citation_needed" \o "Wikipedia:Citation needed)*] due to limited processor support for native BCD operations.

## Application

The [BIOS](https://en.wikipedia.org/wiki/BIOS) in many [personal computers](https://en.wikipedia.org/wiki/Personal_computer) stores the date and time in BCD because the [MC6818](https://en.wikipedia.org/wiki/MC6818) real-time clock chip used in the original [IBM PC AT](https://en.wikipedia.org/wiki/IBM_PC_AT) motherboard provided the time encoded in BCD. This form is easily converted into ASCII for display.[[18]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-19)

The [Atari 8-bit family](https://en.wikipedia.org/wiki/Atari_8-bit_family) of computers used BCD to implement floating-point algorithms. The [MOS 6502](https://en.wikipedia.org/wiki/MOS_Technology_6502) processor used has a BCD mode that affects the addition and subtraction instructions.

Early models of the [PlayStation 3](https://en.wikipedia.org/wiki/PlayStation_3) store the date and time in BCD. This led to a worldwide outage of the console on 1 March 2010. The last two digits of the year stored as BCD [were misinterpreted](https://en.wikipedia.org/wiki/Time_formatting_and_storage_bugs) as 16 causing an error in the unit's date, rendering most functions inoperable. This has been referred to as the [Year 2010 Problem](https://en.wikipedia.org/wiki/Year_2000_problem#Year_2010_problem).

## Representational variations

Various BCD implementations exist that employ other representations for numbers. [Programmable calculators](https://en.wikipedia.org/wiki/Programmable_calculator) manufactured by [Texas Instruments](https://en.wikipedia.org/wiki/Texas_Instruments), [Hewlett-Packard](https://en.wikipedia.org/wiki/Hewlett-Packard), and others typically employ a [floating-point](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Infinity), [underflow](https://en.wikipedia.org/wiki/Arithmetic_underflow)/[overflow](https://en.wikipedia.org/wiki/Arithmetic_overflow), and [error](https://en.wikipedia.org/wiki/Defined_and_undefined) (a blinking display).

### Signed variations

Signed decimal values may be represented in several ways. The [COBOL](https://en.wikipedia.org/wiki/COBOL) programming language, for example, supports a total of five zoned decimal formats, each one encoding the numeric sign in a different way:

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example** |
| Unsigned | No sign nibble | F1 F2 F3 |
| Signed trailing *(canonical format)* | Sign nibble in the last (least significant) byte | F1 F2 C3 |
| Signed leading *(overpunch)* | Sign nibble in the first (most significant) byte | C1 F2 F3 |
| Signed trailing separate | Separate sign character byte ('+' or '−') following the digit bytes | F1 F2 F3 2B |
| Signed leading separate | Separate sign character byte ('+' or '−') preceding the digit bytes | 2B F1 F2 F3 |

### Telephony Binary Coded Decimal (TBCD)

[3GPP](https://en.wikipedia.org/wiki/3GPP) developed **TBCD**,[[19]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-20) an expansion to BCD where the remaining (unused) bit combinations are used to add specific [telephony](https://en.wikipedia.org/wiki/Telephony) characters,[[20]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-21)[[21]](https://en.wikipedia.org/wiki/Binary-coded_decimal#cite_note-22) with digits similar to those found in [telephone keypads](https://en.wikipedia.org/wiki/Dual-tone_multi-frequency_signaling) original design. It is [backward compatible](https://en.wikipedia.org/wiki/Backward_compatible) with BCD.

|  |  |
| --- | --- |
| **Decimal Digit** | **TBCD 8 4 2 1** |
| **\*** | 1 0 1 0 |
| **#** | 1 0 1 1 |
| **a** | 1 1 0 0 |
| **b** | 1 1 0 1 |
| **c** | 1 1 1 0 |
| **Used as filler when there is an odd number of digits** | 1 1 1 1 |

## Alternative encodings

If errors in representation and computation are more important than the speed of conversion to and from display, 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 2×10−1.

This representation allows rapid multiplication and division, but may require shifting by a power of 10 during addition and subtraction to align the decimal points. It is appropriate for applications with a fixed number of decimal places that do not then require this adjustment— particularly financial applications where 2 or 4 digits after the decimal point are usually enough. Indeed, this is almost a form of [fixed point arithmetic](https://en.wikipedia.org/wiki/Fixed_point_arithmetic) since the position of the [radix point](https://en.wikipedia.org/wiki/Radix_point) is implied.

[Chen-Ho encoding](https://en.wikipedia.org/wiki/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](https://en.wikipedia.org/wiki/Densely_Packed_Decimal) is a similar scheme that is used for most of the [significand](https://en.wikipedia.org/wiki/Significand), except the lead digit, for one of the two alternative decimal encodings specified in the [IEEE 754-2008](https://en.wikipedia.org/wiki/IEEE_754-2008) standard.

## See also

* [Bi-quinary coded decimal](https://en.wikipedia.org/wiki/Bi-quinary_coded_decimal)
* [Chen-Ho encoding](https://en.wikipedia.org/wiki/Chen-Ho_encoding)
* [Densely packed decimal](https://en.wikipedia.org/wiki/Densely_packed_decimal)
* [Double dabble](https://en.wikipedia.org/wiki/Double_dabble), an algorithm for converting binary numbers to BCD
* [Gray code](https://en.wikipedia.org/wiki/Gray_code)
* [Year 2000 problem](https://en.wikipedia.org/wiki/Year_2000_problem)
* [Decimal computer](https://en.wikipedia.org/wiki/Decimal_computer)

## Notes

1.  That is, in a standard packed 4-bit representation, there are 16 states (4 bits for 1 digit) with 10 tetrades and 6 pseudo-tetrades, whereas in more densely packed schemes such as [Chen-Ho](https://en.wikipedia.org/wiki/Chen-Ho_encoding) or [DPD](https://en.wikipedia.org/wiki/Densely_Packed_Decimal) coding there are less, f.e. only 24 pseudo-tetrades in 1024 states (10 bits for 3 digits).
   1. (PDF). p. 93.