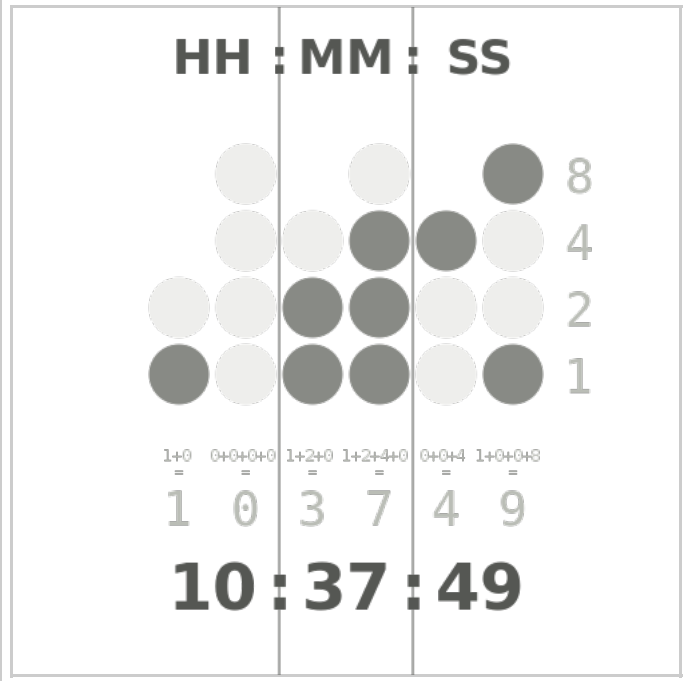
Binary-coded decimal

From Wikipedia, the free encyclopedia

In computing and electronic systems, **binary-coded decimal** (**BCD**) is a class of binary encodings of decimal numbers where each decimal digit is represented by a fixed number of bits, usually four or eight. Special bit patterns are sometimes used for a sign or for other indications (e.g., error or overflow).



A binary clock might use LEDs to express binary values. In this clock, each column of LEDs shows a binary-coded decimal numeral of the traditional sexagesimal time.

In byte-oriented systems (i.e. most modern computers), the term *uncompressed* BCD usually implies a full 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 for instance. The ten states representing a BCD decimal digit are sometimes called *tetrades* (for the nibble typically needed to hold them also known as tetrade) with those don't care-states unused named *pseudo-tetrades*).[nb 1]

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. 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, and implemented in the instruction set of machines such as the IBM System/360 series and its descendants and Digital's VAX. Although BCD *per se* is not as widely used as in the past and is no longer implemented in computers' instruction sets, decimal fixed-point and floating-point formats are still important and continue to be used in financial, commercial, and industrial computing, where subtle conversion and fractional rounding errors that are inherent in floating point binary representations cannot be tolerated.[1]

Contents

- 1 Basics
- 2 BCD in electronics
- 3 Packed BCD
  - 3.1 Fixed-point packed decimal
  - 3.2 Higher-density encodings
- 4 Zoned decimal
  - 4.1 EBCDIC zoned decimal conversion table
  - 4.2 Fixed-point zoned decimal
- 5 IBM and BCD

- 6 Other computers and BCD
- 7 Addition with BCD
- 8 Subtraction with BCD
- 9 Background
- 10 Legal history
- 11 Comparison with pure binary
  - 11.1 Advantages
  - 11.2 Disadvantages
- 12 Application
- 13 Representational variations
  - 13.1 Signed variations
  - 13.2 Telephony Binary Coded Decimal (TBCD)
- 14 Alternative encodings
- 15 See also
- 16 Notes
- 17 References
- 18 Further reading
- 19 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".[2] 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, it is possible to use one of the following methods to encode a BCD number:

- **Uncompressed**: 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 (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 uncompressed 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 uncompressed 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 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 uncompressed BCD; encoding the same number (with the leading zero) in uncompressed format would consume twice the storage.

Shifting and masking operations are used to pack or unpack a packed BCD digit. Other logical operations 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 utilizing 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 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 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

In **Packed BCD** (or simply **packed decimal**), each of the two nibbles of each byte represent a decimal digit. Packed BCD has been in use since at least the 1960s and implemented in all IBM mainframe hardware since then. Most implementations are 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 C) for positive (+) and 1101 (D) for negative (–). This convention was derived from abbreviations for accounting terms (Credit and Debit), as packed decimal coding was widely used in accounting systems. Other allowed signs are 1010 (A) and 1110 (E) for positive and 1011 (B) for negative. Most implementations also provide unsigned BCD values with a sign nibble of 1111 (F).[3][4][5]

ILE RPG uses 1111 (F) for positive and 1101 (D) for negative.[6] 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 was considered a positive sign value (the processors would 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 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 $(2n)-1$ decimal digits, which is always an odd number of digits. A decimal number with d digits requires $\frac{1}{2}(d+1)$ 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 of the machine.)

In contrast, a 4-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 $^1/_6$ of the memory used 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 and compatibility with calculator or hand calculation that fixed-point decimal arithmetic provides. Denser packings of BCD exist which avoid the storage penalty and also need no arithmetic operations for common conversions.

Packed BCD is supported in the 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. Besides the IBM System/360 and later compatible mainframes, packed BCD was implemented in the native instruction set of the original VAX processors from Digital Equipment Corporation and was the native format for the Burroughs Corporation Medium Systems line of mainframes (descended from the 1950s Electrodata 200 series).

Ten's 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 binary numbers, the range not symetric about zero.)

Fixed-point packed decimal

Fixed-point decimal numbers are supported by some programming languages (such as COBOL and 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 2^{10} (1,024) is greater than 10^3 (1,000), if three decimal digits are encoded together, only 10 bits are needed. Two such encodings are *Chen-Ho encoding* and *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 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:

```
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 setting*.

Fixed-point zoned decimal

Some languages (such as COBOL and 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

IBM used the terms **binary-coded decimal** and **BCD** for 6-bit *alphanumeric* codes that represented numbers, upper-case letters and special characters. Some variation of BCD *alphamerics* 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.

The IBM 1400 series were 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* were zero and the digit value represented by standard 4-bit BCD in bits *8* through *1*. For most other characters bits *B* and *A* were 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 was *(12,1)* in the punched card format, was 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 was digit *0*, represented by a lone *0* punch in the card, and *(8,2)* in core memory. [7]

The memory of the IBM 1620 was 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* were 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, 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 6-bit BCD codes.

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

On the IBM 1130 and 1800, packed BCD was 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, mainframes, and Power6. In these products, the BCD is usually zoned BCD (as in EBCDIC or ASCII), Packed BCD (two decimal digits per 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.[8]

Other computers and BCD

The Digital Equipment Corporation VAX-11 series included instructions that could perform arithmetic directly on packed BCD data and convert between packed BCD data and other integer representations.[5] The VAX's packed BCD format was 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 was invoked automatically via exception handling when the no longer implemented instructions were encountered, so that programs using them could execute without modification on the newer machines.

The Intel x86 architecture found on Intel 32-bit 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.[9] For example, the following code (written in C) 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 = a + 0x06666666;
        uint32_t t2 = t1 + b; /* provisional sum */
        uint32_t t3 = t1 ^ b; /* sum without carry propagation */
        uint32_t t4 = t2 ^ t3; /* all the binary carry bits */
        uint32_t t5 = ~t4 & 0x11111110; /* just BCD carry bits */
        uint32_t t6 = (t5 >> 2) | (t5 >> 3); /* correction */
        return t2 - t6; /* corrected sum */
}
```

Addition with BCD

It is possible to perform 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. 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:

```
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 to facilitate BCD arithmetic adjustments following binary addition and subtraction operations.

Subtraction with BCD

Subtraction is done by adding the ten's complement of the subtrahend. To represent the sign of a number in BCD, the number 0000 is used to represent a positive number, and 1001 is used to represent a 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 of 432 can be obtained by taking the nine's 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 $2^4 = 16$), 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 bit is the sign bit, which is 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 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 or subtracting. So, with 1053 – 122, 122 would have to first be represented as 0122, and the ten's 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', *etc*., indicates the weight of each bit shown; note that in the fifth column 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 digits from 0 to 9 in various BCD systems:

Digit	BCD 8 4 2 1	Excess-3 or Stibitz Code	BCD 2 4 2 1 or Aiken Code	BCD 8 4 −2 −1	IBM 702 IBM 705 IBM 7080 IBM 1401 8 4 2 1	ASCII 0000 8421	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, 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 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, 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).

Disadvantages

- Some operations are more complex to implement. Adders 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. 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 percent more space than a binary encoding (the ratio of 4 bits to $\log_2 10$ 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, due to limited processor support for native BCD operations.

Application

The BIOS in many personal computers stores the date and time in BCD because the MC6818 real-time clock chip used in the original IBM PC AT motherboard provided the time encoded in BCD. This form is easily converted into ASCII for display.[10]

The Atari 8-bit family of computers used BCD to implement floating-point algorithms. The MOS 6502 processor used has a BCD mode that affects the addition and subtraction instructions.

Early models of the 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 as 16 causing an error in the unit's date, rendering most functions inoperable. This has been referred to as the Year 2010 Problem.

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).

Signed variations

Signed decimal values may be represented in several ways. The 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 developed **TBCD**,[11] an expansion to BCD where the remaining (unused) bit combinations are used to add specific telephony characters,[12][13] with digits similar to those found in telephone keypads original design. It is backward compatible to 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 since the position of the radix point is implied.

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 is used for most of the significand, except the lead digit, for one of the two alternative decimal encodings specified in the IEEE 754-2008 standard.

See also

- Bi-quinary coded decimal
- Chen-Ho encoding
- Densely packed decimal
- Double dabble, an algorithm for converting binary numbers to BCD
- Gray code
- Year 2000 problem
- 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 or DPD coding there are less, f.e. only 24 pseudo-tetrades in 1024 states (10 bits for 3 digits).

References

1. "General Decimal Arithmetic" (http://speleotrove.com/decimal/).
2. Parag K., Lala (2007). *Principles of Modern Digital Design* (http://books.google.com/books?id=doNGOrHUyCoC&lpg=PA20&dq=binary%20coded%20decimal%204221&pg=PA20#v=onepage&q=binary%20coded%20decimal%204221&f=false). John Wiley & Sons. pp. 20–25. ISBN 978-0-470-07296-7.
3. "Chapter 8: Decimal Instructions", *IBM System/370 Principles of Operation*, IBM, March 1980
4. "Chapter 3: Data Representation", *PDP-11 Architecture Handbook*, Digital Equipment Corporation, 1983
5. *VAX-11 Architecture Handbook*, Digital Equipment Corporation, 1985
6. "ILE RPG Reference" (http://publib.boulder.ibm.com/iseries/v5r2/ic2924/books/c0925083170.htm).
7. IBM BM 1401/1440/1460/1410/7010 Character Code Chart in BCD Order (http://ed-thelen.org/1401Project/Van1401-CodeChart.pdf)
8. http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.iceg200%2Fenf.htm
9. Douglas W. Jones, BCD Arithmetic, a tutorial (http://homepage.cs.uiowa.edu/~jones/bcd/bcd.html), 2002.
10. http://www.se.ecu.edu.au/units/ens1242/lectures/ens_Notes_08.pdf
11. *3GPP TS 29.002: Mobile Application Part (MAP) specification* (http://www.3gpp.org/ftp/Specs/html-info/29002.htm) (Technical report). 2013. sec. 17.7.8 Common data types.
12. "Signalling Protocols and Switching (SPS) Guidelines for using Abstract Syntax Notation One (ASN.1) in telecommunication application protocols" (http://www.etsi.org/deliver/etsi_etr/001_099/060/02_60/etr_060e02p.pdf). p. 15.
13. "XOM Mobile Application Part (XMAP) Specification" (http://www.openss7.org/specs/xmap.pdf). p. 93.

Further reading

- Mackenzie, Charles E. (1980). *Coded Character Sets: History and Development*. Addison-Wesley. ISBN 0-201-14460-3.
- *Arithmetic Operations in Digital Computers*, R. K. Richards, 397pp, D. Van Nostrand Co., NY, 1955
- Schmid, Hermann, *Decimal computation*, ISBN 0-471-76180-X, 266pp, Wiley, 1974
- *Superoptimizer: A Look at the Smallest Program*, Henry Massalin, ACM Sigplan Notices, Vol. 22 #10 (Proceedings of the Second International Conference on Architectural support for Programming Languages and Operating Systems), pp122–126, ACM, also IEEE Computer Society Press #87CH2440-6, October 1987
- *VLSI designs for redundant binary-coded decimal addition*, Behrooz Shirazi, David Y. Y. Yun, and Chang N. Zhang, IEEE Seventh Annual International Phoenix Conference on Computers and Communications, 1988, pp52–56, IEEE, March 1988
- *Fundamentals of Digital Logic* by Brown and Vranesic, 2003
- *Modified Carry Look Ahead BCD Adder With CMOS and Reversible Logic Implementation*, Himanshu Thapliyal and Hamid R. Arabnia, Proceedings of the 2006 International Conference on Computer Design (CDES'06), ISBN 1-60132-009-4, pp64–69, CSREA Press, November 2006
- *Reversible Implementation of Densely-Packed-Decimal Converter to and from Binary-Coded-Decimal Format Using in IEEE-754R*, A. Kaivani, A. Zaker Alhosseini, S. Gorgin, and M. Fazlali, 9th International Conference on Information Technology (ICIT'06), pp273–276, IEEE, December 2006.
- Decimal Arithmetic Bibliography (http://speleotrove.com/decimal/decbibindex.html)

External links

- IBM: Chen-Ho encoding (http://speleotrove.com/decimal/chen-ho.html)
- IBM: Densely Packed Decimal (http://speleotrove.com/decimal/DPDecimal.html)
- Convert BCD to decimal, binary and hexadecimal and vice versa (http://www.unitjuggler.com/convert-numbersystems-from-decimal-to-bcd.html)
- BCD for Java (http://github.com/c-rack/bcd4j)

Retrieved from "http://en.wikipedia.org/w/index.php?title=Binary-coded_decimal&oldid=653660582"

Categories: Computer arithmetic | Numeral systems

- This page was last modified on 26 March 2015, at 21:29.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.