(12) UK Patent Application (19) GB (11) 2 376 773 (13) A

(43) Date of A Publication 24.12.2002

(21) Application No 0205775.0

(22) Date of Filing 12.03.2002

(30) Priority Data
(31) 60275323 (32) 12.03.2001 (33) US
(31) 60275324 (32) 12.03.2001

(51) INT CL^7
G06F 5/00

(52) UK CL (Edition T)
G4A ACC

(56) Documents Cited
JP 050289846 A JP 020034037 A
US 6396921 B1

(58) Field of Search
UK CL (Edition T) G4A AAF AAU ACC ACX
INT CL^7 G06F 5/00 5/01 17/21 17/22, H03M
Other: Online: WPI, EPODOC, PAJ, TXTE, INSPEC, ELSEVIER, IBM TDB, IEEE Xplore, Internet

(71) Applicant(s)
Touch Technologies Inc
(Incorporated in USA - California)
Suite 310, 9988 Hibert Street, San Diego, California 92131, United States of America

(72) Inventor(s)
Daniel Esbensen

(74) Agent and/or Address for Service
Mewburn Ellis
York House, 23 Kingsway, LONDON, WC2B 6HP, United Kingdom

(54) Abstract Title

Display and/or precision operations of numerical values in binary systems

(57) A method and/or apparatus determining character codes such as ASCII and EBCDIC for a numerical value retrieves two or more character codes for each iteration of the method and/or representing and operating on numerical values in binary systems.

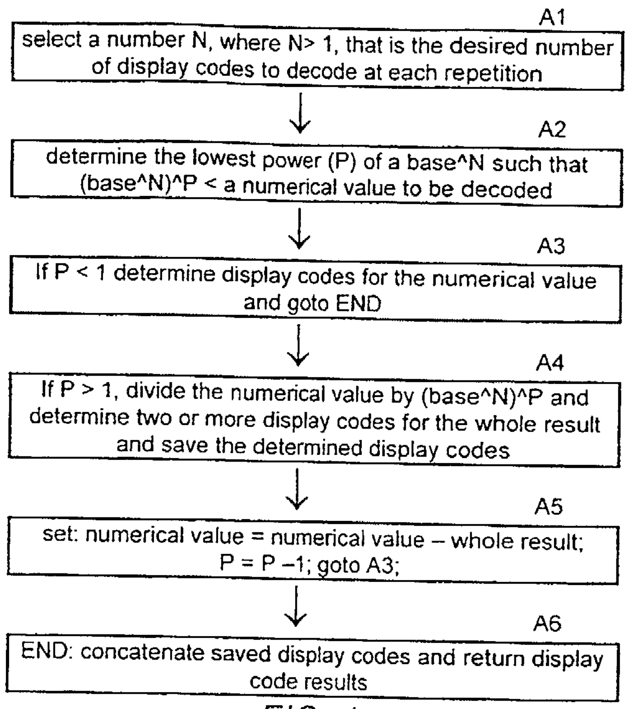

A1
select a number N, where N> 1, that is the desired number of display codes to decode at each repetition
A2
determine the lowest power (P) of a base^N such that (base^N)^P < a numerical value to be decoded
A3
If P < 1 determine display codes for the numerical value and goto END
A4
If P > 1, divide the numerical value by (base^N)^P and determine two or more display codes for the whole result and save the determined display codes
A5
set: numerical value = numerical value – whole result; P = P –1; goto A3;
A6
END: concatenate saved display codes and return display code results

FIG. 1

GB 2 376 773 A

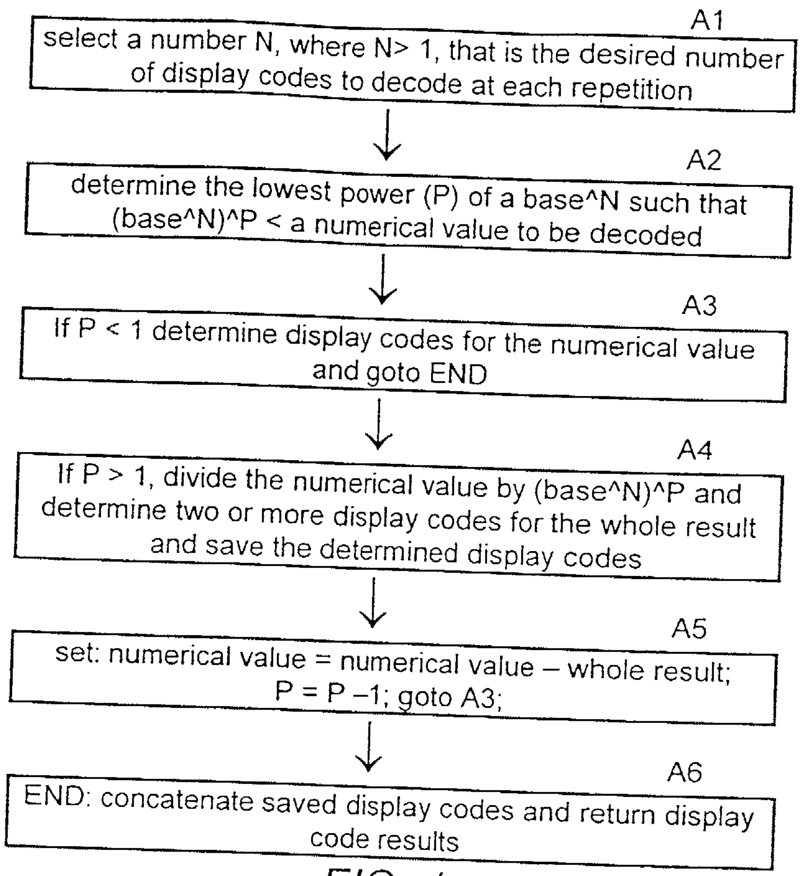

A1
select a number N, where N> 1, that is the desired number of display codes to decode at each repetition
A2
determine the lowest power (P) of a base^N such that (base^N)^P < a numerical value to be decoded
A3
If P < 1 determine display codes for the numerical value and goto END
A4
If P > 1, divide the numerical value by (base^N)^P and determine two or more display codes for the whole result and save the determined display codes
A5
set: numerical value = numerical value – whole result; P = P –1; goto A3;
A6
END: concatenate saved display codes and return display code results

FIG. 1

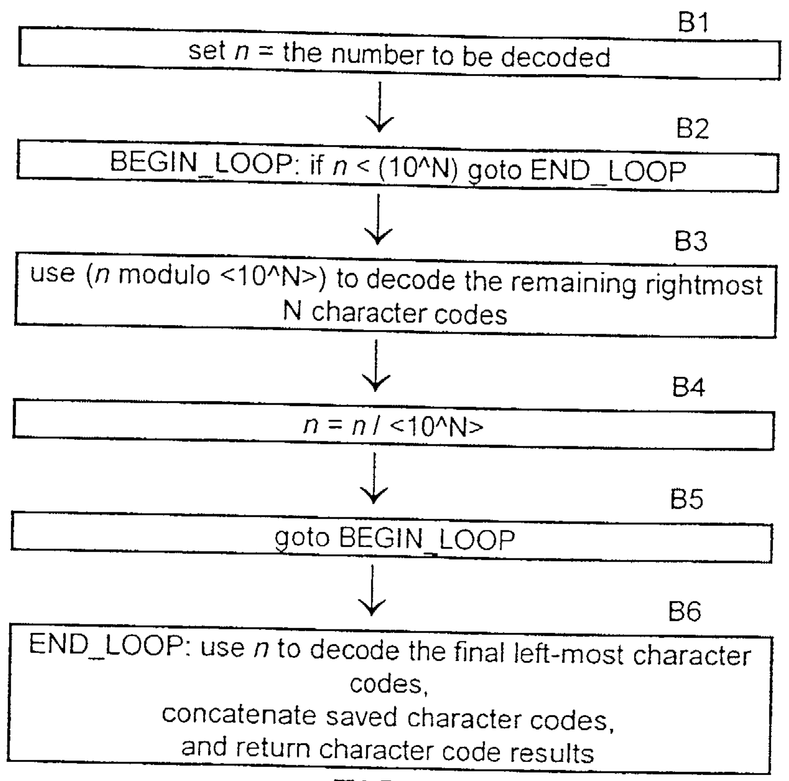

B1
set n = the number to be decoded
B2
BEGIN_LOOP: if n < (10^N) goto END_LOOP
B3
use (n modulo <10^N>) to decode the remaining rightmost N character codes
B4
n = n / <10^N>
B5
goto BEGIN_LOOP
B6
END_LOOP: use n to decode the final left-most character codes, concatenate saved character codes, and return character code results

FIG. 2

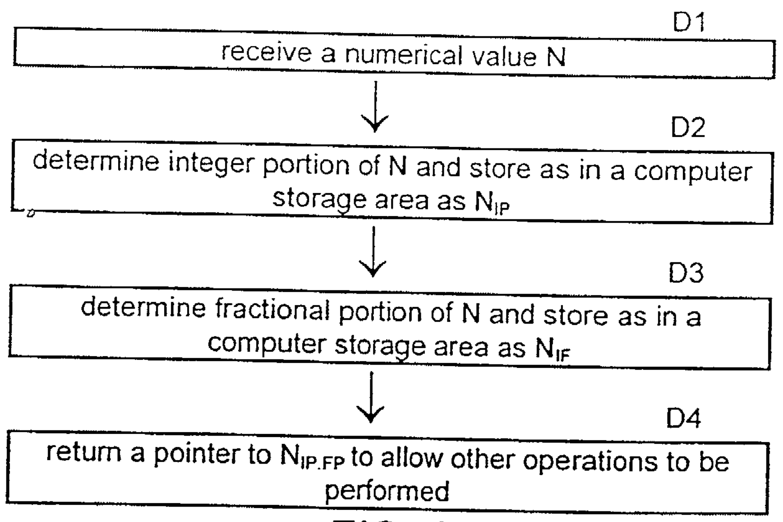
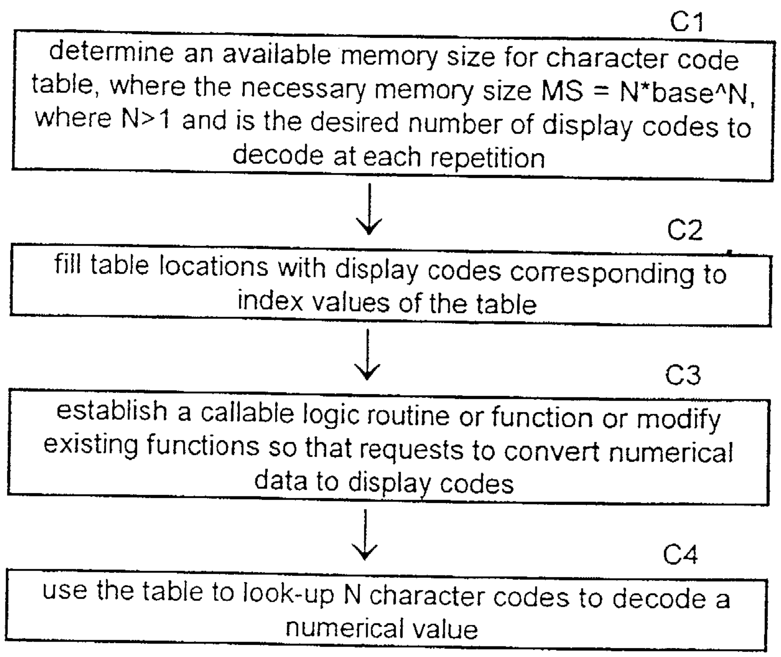

D1
receive a numerical value N
D2
determine integer portion of N and store as in a computer storage area as N_{IP}
D3
determine fractional portion of N and store as in a computer storage area as N_{IF}
D4
return a pointer to $N_{IP.FP}$ to allow other operations to be performed

FIG. 6


C1
determine an available memory size for character code table, where the necessary memory size MS = N*base^N, where N>1 and is the desired number of display codes to decode at each repetition

C2
fill table locations with display codes corresponding to index values of the table

C3
establish a callable logic routine or function or modify existing functions so that requests to convert numerical data to display codes

C4
use the table to look-up N character codes to decode a numerical value

FIG. 3

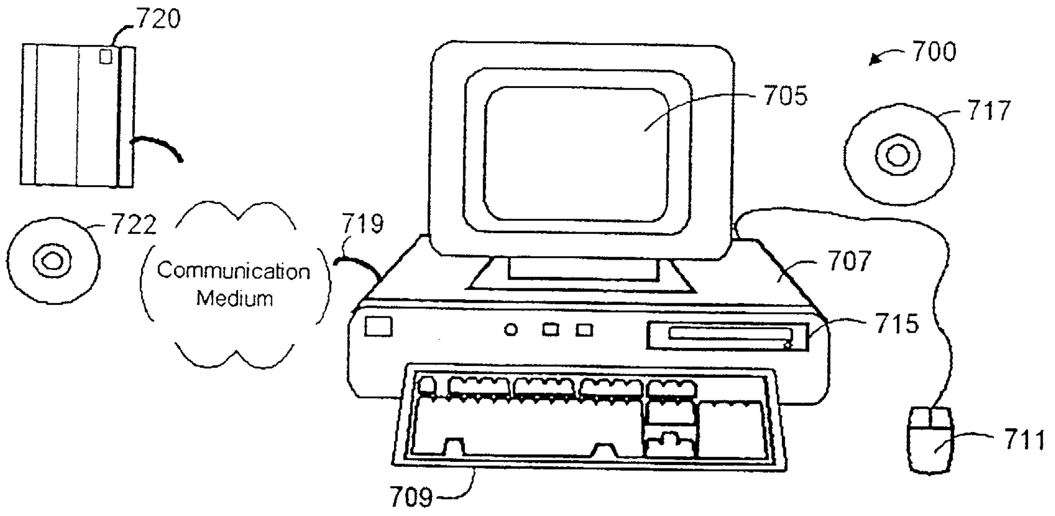


FIG. 8

```
                                  ////////////////////////////////////////////////////////////
// GET_ASCII_FRACTIONAL_DIGITS EXAMPLE CODE
/////////////////////////////////////////////////////////////////////////
// Brief description:    Get ascii fractional digits.
// Expected:
//             a       =   address of a character array for output ascii digits
//             i       =   index into above character array
//             n       =   binary integer to convert
//             p       =   scale fractional digits NEVER USED?
//             digits  =   number of precision digits
// Result  :  i new index into a
// Other variables:
//              TABLE_SCALE = number of entries in looked-up table; also 10^N
//              index = index value used to access ntab
//              ntab = indexed table of character codes
//              TABLE_DIGITS = number of digits in each table entry
//              table_digit = position of digit (1st (0) through TABLE_DIGITSth
//              TABLE_DIGITS -1) ) read in a
//              particular table entry; DAN, I AM A LITTLE CONFUSED BY THIS
/////////////////////////////////////////////////////////////////////////
inline int routine get_ascii_fractional_digits (char *a, int i, int n,
                         int fract_digits, int digits)
{
  int table_digit, index;
  if (n < 0)                             /* calculate -1 */
    n = -n;                              /* as 1 */
  while (fract_digits > 0)
    {
      if (n < TABLE_SCALE)
      {                                  /* already less no need for a divide */
        index = (int) n;                 /* use the number as the index */
        n = 0;                           /* no more number */
      }
      else
      {
        index = n % TABLE_SCALE;         /* set index to remainder (using modulo)
*/
        n /= TABLE_SCALE;                /* new n = n/TABLE_SCALE
      }
      for (table_digit = 0; table_digit < TABLE_DIGITS;
         table_digit++, --fract_digits)
        if (fract_digits > 0 && fract_digits <= digits)
          a[i++] = ntab[index][table_digit]; /* In this example, each character
is
                                              /* separately fetched from its
table
                                         /* entry and the table is treated as a
                                         /* 2-dimensional table.
    }
  return i;
}                                       /* END get ascii fractional digits */
```

FIG. 4

```
//////////////////////////////////////////////////////////////////////
// GET_ASCII_WHOLE_DIGITS
//////////////////////////////////////////////////////////////////////
// Brief description:  Get ascii whole integer digits.
// Expected:
//            a       =   address a character array for ascii digits
//            i       =   index into above character array
//            n       =   binary 64-bit integer to convert
//
// Result  :  i new index into a
//
//////////////////////////////////////////////////////////////////////
inline int routine get_ascii_whole_digits (char *a, int i, real * _n)
{
  int index, table_digit, skip_leading;
  int64 n = _n->ip;                        /* integer portion */
  do
    {
      if (n < TABLE_SCALE)
      {                                    /* already less, no need for a divide */
        index = (int) n;                   /* use the number as the index */
        n = 0;                             /* no more number */
      }
      else
      {
        index = (int) (n % TABLE_SCALE);  /* remainder */
        n /= TABLE_SCALE;                  /* divide by TABLE_SCALE */
      }

      a[i++] = ntab[index][0];             /* always at least 1 character */
      if (n)                               /* n does not equal zero */
      {                                    /* more to come */
        for (table_digit = 1; table_digit < TABLE_DIGITS;
           table_digit++)
          a[i++] = ntab[index][table_digit];/* convert all digits for this
entry*/
      }
      else
      {                                      /* skip leading '0's on last divide
*/
        for (skip_leading = TABLE_DIGITS - 1; skip_leading;
           --skip_leading)
          if (ntab[index][skip_leading] != '0')
            {
              for (table_digit = 1; table_digit <= skip_leading;
                 table_digit++) a[i++] = ntab[index][table_digit];
              break;
            }
      }
    }
  while (n);
  return i;
}                                 /* get ascii whole digits */
```

FIG. 5

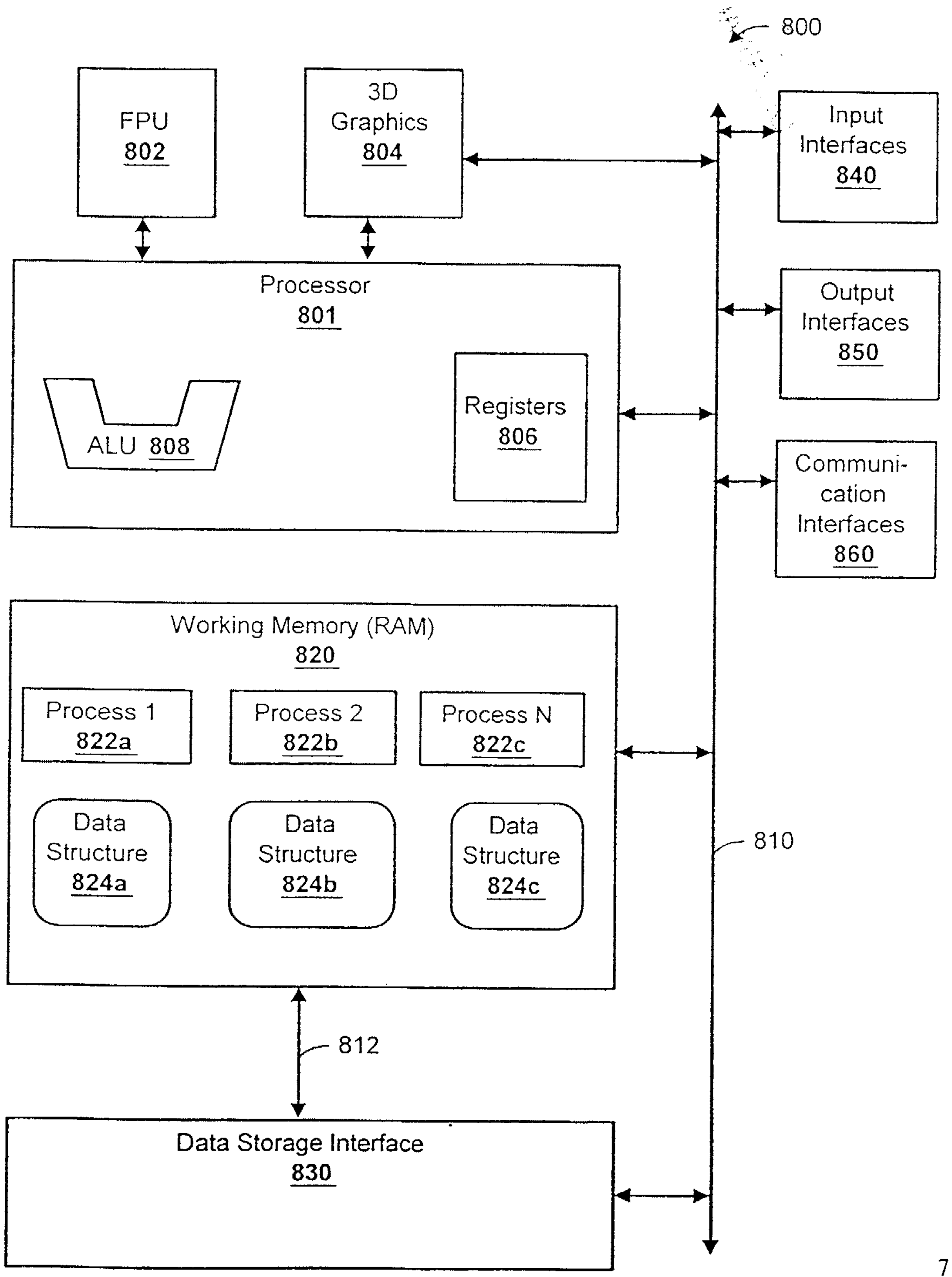

800
FPU
802
3D Graphics
804
Input Interfaces
840
Processor
801
Output Interfaces
850
ALU 808
Registers
806
Communi-cation Interfaces
860
Working Memory (RAM)
820
Process 1
822a
Process 2
822b
Process N
822c
Data Structure
824a
Data Structure
824b
Data Structure
824c
810
812
Data Storage Interface
830

FIG. 7

APPARATUS AND METHOD FOR DISPLAY AND/OR PRECISION OPERATIONS OF NUMERICAL VALUES IN BINARY SYSTEMS

The present invention relates to information processing systems and methods.

The discussion of any work, publications, sales, or activity anywhere in this submission, including in any documents submitted with this application and in any documents incorporated herein by reference, shall not be taken as an admission by the inventors that any such work constitutes prior art. The discussion of any activity, work, or publication herein is not an admission that such activity, work, or publication existed or was known in any particular jurisdiction.

A task that faced the early designers of computer architecture was how to quickly and efficiently display numerical data. Numerical data is typically stored in computer systems as a base-2 representation of the number. For example, the two byte integer 63119**d** (herein **d** indicates decimal notation, **h** indicates hexadecimal, **b** indicates binary notation) may be represented in a computer's binary memory as the binary number 1100000110001011**b**. Many variations are known regarding representing numerical values in binary systems, such as 2's complement notation.

A difficulty arises when converting a binary number to a form for display or operations in another base system, such as decimal. In presentation applications, a routine must convert the binary-stored number to a series of character codes. Known and commonly used character codes include ASCII and EBCDIC. The Baudot Code is another example code that was used extensively in telegraph systems, with five bits coding 32 characters. While various presentation codes present different issues in converting between binary, the overall problem is the same. For simplicity, the present discussion will concentrate on conversion of binary encoded numbers to ASCII, though the present invention can be used to convert between other appropriate encoding schemes.

Such conversion in most computer systems is a processor-intensive task. One method is to translate each decimal digit separately from the left, dividing the original number by the largest power of 10 less than it. The whole number result (also referred to as the quotient) is then looked up in a table or combined with a value to give an ASCII code. The process is repeated in the remainders until character codes for all the digits are determined.

An alternative method is to translate each decimal digit starting from the right, by dividing the original number by 10, and using the modulo operator to look up in a table or combine with a value to give an ASCII code and repeat. While this form may be shorter to express, the computer operations necessary to perform the modulo are generally similar, if not longer and more complex. The binary arithmetic calculations necessary to perform these operations are very processor intensive. In particular, divisions of binary numbers, particularly floating point numbers, is often very slow and processor intensive.

Decoding a signed and/or a floating-point number requires additional steps. For signed numbers, in one example prior art method, the sign of negative numbers is first stored, then the number is converted to a positive value (which may involve a 2's complement conversion) and the general procedure described above is performed. For floating point numbers, further steps may be needed to determine the correct decimal point location and to convert the floating point number to the correct integer value before converting the binary integer value to character codes. These steps vary depending on the particular floating point encoding scheme used in a particular system.

Calculations of Decimal Numbers in Binary Systems

In order for non-binary numbers to be manipulated in binary systems, the numbers must be converted or encoded in a form compatible with the base-2 representation of binary systems. For example, the two-byte integer 63119**d** may be represented in a computer's binary memory as the binary number 1100000110001011**b**. Many variations are known for representing numbers in binary systems. Floating point numbers present additional issues. A variety of binary floating-point formats have been defined for computers; one of the most popular is that defined by IEEE (Institute of Electrical & Electronic Engineers) known as IEEE 754.

The IEEE 754 specification defines 64 bit floating-point format with three parts:

(1) An 11-bit binary exponent, using "excess-1023" format. In this format, the exponent is represented as an unsigned binary integer from 0 to 2047, and one subtracts 1023 to get the signed value of the exponent.

(2) A 52-bit mantissa, also an unsigned binary number, defining a fractional value with a leading implied "1".

(3) A sign bit, giving the sign of the mantissa.

A variation of this scheme uses 32-bits, such as a 23-bit mantissa with a sign bit and an 8-bit exponent (in excess-127 format), giving 7 valid decimal digits. Such floating-point numbers are sometimes referred to as "reals" or "floats": a 32-bit float value is sometimes called a "real32" or a "single" (indicating "single-precision floating-point value") while a 64-bit float is sometimes called a "real64" or a "double" (indicating "double-precision floating-point value").

With these floating-point numbers, precision problems can be encountered. As with integers, there is only a finite range of values, though it is a larger range, allowing for "overflow" or "underflow." A maximum real value allowed in a system is sometimes referred to as "machine infinity." A further problem is that there is limited precision to computer-encoded real numbers. As a result, in many floating-point computations, there can be a small error in the result because some lower digits have been dropped. This may be unnoticeable in most cases, but in math analysis that requires a lot of computations, the errors tend to build up and can affect the results. Another error that can arise in floating-point numbers is due to the fact that the mantissa is expressed as a binary fraction that may not perfectly match a desired decimal fraction.

An alternative format used in some systems is to create fixed decimal point representations for real values (sometimes referred to as ***scaled values***). As an example, the encoding scheme **FOUR** assumes for all encoded numbers that four decimal digits are present after the decimal. All computations are done on binary whole numbers, without a loss of significance. However, this means that on a system with 64-bit integers (19 decimal digits of data), in the given example, this would allow only the representation of 15 digits left of the decimal point and four to the right of the decimal point. Another format used to handle decimal numbers and address some of these issues is **Binary Coded Decimal** (BCD). In this notation groups of 4 bits are used to represent each decimal digit from 0 to 9.

In prior art systems, floating point processing of floating point numbers that are defined by a known standard is often handled by a Floating Point Unit (FPU), typically an integrated circuit module or area designed to handle floating-point numbers. In systems without a "hardware" FPU, floating point operations are generally handled by software.

SUMMARY OF THE INVENTION

The present invention in a first of its aspects may provide a method and/or apparatus and/or digital logic circuit for more quickly determining the character codes (such as

ASCII or EBSIDIC character codes) for a binary-represented numerical value and that in specific embodiments, avoiding many of the divisions that can be necessary when converting to character codes.

In further aspects, the invention may provide a method and/or apparatus and/or digital logic circuit for more effectively handling numerical values in a binary information handling system, such as using two non-contiguous (or "separated") bit areas to store real numbers. These areas are referred to as non-contiguous because preferably there is no automatic binary carry in either direction with regard to the bit areas and because preferably in a given number there is no dependency on either part of the number to determine the value of the other part. In other words, the integer portion of a decimal is preferably dependent only on the integer portion of the stored binary value. Likewise, the fractional portion of a decimal is preferably dependent only on the fractional portion of the stored binary value. In terms of their storage in a computer's memory, the different portions of the numbers may be stored in memory locations next to each other. Thus, preferably a real number as discussed herein is stored as an integer part (IP) and a fractional part (FP).

Preferably, the "independence" of the IP and FP of a number extend into the sign bits and both the IP and the FP carry a sign bit. Preferably, the sign bits on the IP and the FP are always identical for both portions. As used herein, such a number may be designated as *N*.IP.FP.

The IP according to preferred embodiments of the invention is represented as a conventional binary integer. In preferred embodiments, a sign bit is included as the first bit, with 2's complement or other notation used to represent negative numbers. The FP preferably is also represented as a conventional binary integer. Preferably, a sign is also included as the first bit, with 2's complement or other notation used to represent negative numbers. However the fractional portion of the decimal number may be multiplied by 10^{scale_factor} to insure all bits of precision of interest are within the integer. For example, in a scale_factor=9 system, the fractional part of a decimal number is multiplied by 10^9 before it is stored in the FP, as indicated in the following table:

Decimal Part	FP
.4	400 000 000
.03456	034 560 000
.1415926	141 592 600

Preferably, the ***scale_factor*** is a selected power of 10 and indicates the number of fractional decimal digits supported by the system. Scale is sometimes used to herein to indicate 10^{scale_factor}. In an example embodiment, 64-bits are used for the integer portion and 32 bits for the decimal portion, thus a single FP requires 96-bits of storage. As used herein, such a number may be designated as an r96 (for real-96 bit). Preferably, a numbering representation according to the present invention uses 18 decimal digits in the whole part of the number and 8 digits in the decimal portion, which allows some implementations to use built-in math functions of various CPUs.

Preferably, carries between the IP and FP portions of a number are not handled automatically and different numerical operators handle carries between the IP and FP explicitly and with appropriate variations for each operation.

Preferably, the binary storage space for an FP will be able to hold a number that is larger that the maximum FP allowed. In a system with 8 decimal digits in the FP portion, for example, the largest FP that can be represented is .99999999d. To store this number requires 27 bits for the binary value (101111101011110000011111111), plus a sign bit, for a total of 28 bits. However, the largest decimal number that can be stored in 27 bits is 2.18103807. Thus, for many mathematical operations, the resultant FP portion (S.FP) will be compared to the MAX.FP (e.g. .99999999d) and if over will cause a carry bit to be active and will be decremented by the MAX.FP.

The invention and various specific aspects and embodiments will be better understood with reference to the following drawings and detailed descriptions. For purposes of clarity, this discussion refers to devices, methods, and concepts in terms of specific examples. However, the invention and aspects thereof may have applications to a variety of types of devices and systems. It is therefore intended that the invention not be limited except as provided in the attached claims. It is well known in the art that logic systems and methods such as described herein can include a variety of different components and different functions in a modular fashion. Different embodiments of the invention can include different mixtures of elements and functions and may group various functions as parts of various elements.

The functional aspects of the invention that are implemented on a computer, as will be understood from the teachings herein, may be implemented or accomplished using any appropriate implementation environment or programming language, such as C, C++, Cobol, Pascal, Java, Java-script, assembly or machine code programming, custom logic circuits, etc. All references, publications, patents, and patent applications cited herein are hereby incorporated by reference in their entirety for all purposes.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates an example method according to specific embodiments of the invention.

FIG. 2 illustrates an alternative method in some details according to specific embodiments of the invention.

FIG. 3 illustrates an example system deployment method details according to specific embodiments of the invention.

FIG. 4 illustrates an example program code fragment implementation according to specific embodiments of the present invention.

FIG. 5 illustrates an example program code fragment implementation according to specific embodiments of the present invention.

FIG. 6 illustrates an example method according to specific embodiments of the invention.

FIG. 7 illustrates an example architecture of an example information handling system relevant to various specific embodiments of the present invention.

FIG. 8 is a block diagram showing a representative example logic device in which various aspects of the present invention may be embodied.

In one embodiment, the invention can be understood as a computer implemented method that converts binary integer values into decimal values by using a divisor smaller than the original value, but that satisfies the formula **(10^N)^P**, where N is an integer greater than 1, and P is an integer greater than or equal to 1. **N**, as further described below, according to specific embodiments of the present invention, is a number preselected or predetermined and represents the number of decimal digits that will be determined during each iteration of the routine. P is determined when a numerical value is being decoded and is selected as the maximum number that provides a **(10^N)^P** value less than the absolute value of the original number. While **10** is used as an example base value throughout this discussion other embodiments can use a different base value. The equation above can be understood as **(base^N)^P**.

A specific example of a general method, determining two digit character codes **(N=2)** after each division, using 63179 is provided below:

1. Determine the largest **(10^N)^P** less than the number = 10000.
2. 63179 / 10000, has a whole number result 06 (lookup this numerical value) ◊ 30h 36h.
3. Multiply the whole number result (06) and the divisor (10000) = 6000.
4. Subtract that number from the original number (63179 - 60000) = 3179.
5. Repeat steps 1 through 4 until all digits are decoded, as follows:
6. Determine the largest **(10^N)^P** less than the number (or set P=P-1) = 100.
7. 3179 / 100 has a whole number result 31 ◊ 33h 31h.
8. Subtract that number from the original number (3179 - 3100 = 79).
9. STOP when the subtraction result is less than **(10^N)** (or when P=1) and lookup the final result: 79◊ 37h 39h.
10. Return the resulting character codes (deleting leading zeros at some point if desired): 30h 36h 33h 31h 37h 39h.

Alternatively, this method can be performed or expressed as follows, with N=2:

1. Set *n* = 63179.
2. *n* modulo **(10^N)** = 79 ◊ 37h 39h.
3. set *n* = *n* / 100 = 631.
4. Repeat steps 2 and 3 until *n* < **(10^N)**.
5. Lookup final result: 6◊ 36h.
6. Concatenate the resulting character codes: **36h 33h 31h 37h 39h.**

FIG. 1 illustrates an example method according to specific embodiments of the invention. FIG. 2 illustrates an alternative method in some details according to specific embodiments of the invention. As shown in the figures, the invention can be embodied in a logical method suitable for implementation by any appropriately configured information handling system, including general purpose computing systems and information handling displays and subsystems.

This aspect of the invention has described in terms of general embodiments that are believed to be a full and complete description sufficient to allow a practitioner in the art to make and use the invention. What follows are descriptions of example systems and methods that are involved with or may embody various aspects of the present invention. The following discussion may also include independent innovative embodiments of the invention.

Look-up Tables

One method or mechanism for performing the above described steps uses an appropriately sized look-up table associating all numerical values less than **10^N** with the appropriate character codes (generally **N** codes in each table entry). An example table can have the form:

Binary Encoded Number (Index Value)		Character Codes (in this example, ASCII codes expressed in hexadecimal)
Decimal Notation	Binary Notation	
0	000000	30h 30h
1	000001	30h 31h
2	000010	30h 32h
*		
16	010000	31h 36h
17	010001	31h 37h
18	010010	31h 38h
*		
79	1001111	37h 39h
80	1010000	38h 30h
81	1010001	38h 31h
*		*
98	1100010	39h 38h
99	1100011	39h 39h

TABLE 1

Note that the uncompressed size of this table is 100 locations times two bytes or 200 bytes. More generally, the uncompressed size of a **base-10** table as discussed above is **N*(10^N)**. Thus, using a similar method, three of five character codes could be retrieved for each table lookup, requiring tables of 3000 (approximately 3K) or 500000 (approximately .5M) bytes respectively. A single representative entry for an N=3 table is:

Binary Encoded Number (Index Value)		Character Codes (in this example, ASCII codes expressed in hexadecimal)
Decimal	Binary	
174	10101110	31h 37h 34h

TABLE 2

Note that in each of these example tables, it is indicated that the tables are one-dimensional tables the numerical value being decoded used as the index for the table entries. While this represents one embodiment of the present invention, the invention can use other table formats, where desired, including table formats wherein the index is not simply the numerical value being decoded. When decoding larger numbers for display, large tables dramatically reduce the number of binary mathematical steps required. It will be understood from the teachings herein that such tables will have a large amount of redundancy and thus may be compressed in specific embodiments according to various compression methods. However, in a common embodiment, the present invention is used to speed up conversion of numerical values to display codes and thus uses an uncompressed look-up table.

FIG. 3 illustrates an example system deployment method details according to specific embodiments of the invention. As shown in the figure, in this method a table is allocated in memory according to available or desired memory size for the table, and the table is then used to decode N character codes of a numerical value.

Negative and Conventional Floating Point Numbers

According to specific embodiments, the present invention can handle negative numbers in various ways according to the underlying scheme for storing negative numbers. A most straightforward method for handling negative numbers involves remembering the sign of the number and converting the number to a positive value before performing a table look-up using the number. According to further specific embodiments of the present invention, floating point numbers are decoded by first converting the floating point number to an integer number before converting to display codes.

Separately Stored Floating Point Numbers

According to further specific embodiments of the present invention, display code conversion can be beneficially used with the innovative floating point encoding scheme as described below. In such a decoding scheme, conversion to display codes is simplified because every stored floating point number is stored separately, with separate storage areas for the integer part and fractional part of the number. With this scheme, the integer portion and the fractional portion can separately be converted to display codes as described above.

As a very specific example of a method according to specific embodiments of the present invention for converting and IP and FP separately to character codes, example C++ code listings are provided in FIG. 4 and FIG. 5.

MATHEMATICAL AND LOGICAL OPERATIONS

In one embodiment, the invention can be understood as a computer-implemented method for storing and performing operations on numerical values. FIG. 6 illustrates an example method according to specific embodiments of the invention.

According to specific embodiments, the present invention provides specific routines for various math operators, including optional shortcuts according to specific embodiments. As used herein, a notation for the portions of a real number A is A.IP and A.FP. The results of a function F(A,B) is noted as S.IP and S.FP or S.IP.FP

Addition Operations

Addition of two numbers, for example A and B, proceeds generally as follows. First, if one number is negative, but not another, then jump to a subtraction operation, as described below. Then, if both of the numbers are negative, store that fact, convert the numbers to positive (i.e. use the absolute values of the numbers) and perform the addition as follows:

```
s.fp = a.fp + b.fp;
carry = 0
if s.fp > max.fp then
 carry = 1
 s.fp = s.fp - max.fp - 1 ;max.fp is a scaled
number e.g. 99999999 and "1" is unscaled
endif
s.ip = a.ip + b.ip + carry
if both a and b were negative then
 s.ip =  - s.ip
 s.fp = - s.fp
return
```

Subtraction Operations

Subtraction according to specific embodiments of the invention is defined as combining two numbers having different signs. With the positive number set to A and the negative number set to B, the absolute value of B is subsequently used for all compare operations and the procedure performs as indicated below. According to further specific embodiments of the invention, a number of shortcuts can be provided as indicated.

```
;S=A-B
;shortcuts according to specific embodiments
if a = 0.0
 s.fp = -b.fp
 s.ip = -b.ip
 return
if b = 0.0
 s.fp =  a.fp
 s.ip =  a.ip
 return
```

```
;main procedure
if (b.ip < a.ip) and (a.fp < b.fp) then
 a.ip = a.ip-1; reverse carry
 a.fp = a.fp + max.fp + 1 ;max.fp is a scaled
number e.g. 99999999 and "1" is unscaled
endif
if (a.ip < b.ip) and (b.fp < a.fp) then
 b.ip = b.ip-1; reverse carry
 b.fp = b.fp + max.fp + 1;
endif
s.fp = a.fp - b.fp
s.ip = a.ip - b.ip
```

<u>**Multiplication And Division Operations**</u>

Multiplication and division, according to specific embodiments of the present invention, are handled by converting the operands into scaled numbers, performing the desired function F() on the scaled numbers using scaled arithmetic, and then converting the scaled result back into the N.ip.fp format. According to specific embodiments of the present invention, the scale factor used for the scaled number can be the same scale factor used for the FP portion of the R.ip.fp number, which simplifies a number of conversion operations. (According to alternate embodiments of the invention, a different scale factor can be used, thus requiring a scaling of both the IP and FP parts to compute an A_{scaled} number.)

The procedure generally works as follows (with the scale factor of S_{scaled}, A_{scaled}, and B_{scaled} is set as the same scale factor used in the FP portions r.ip.fp number.

1. $A_{scaled} = A.IP*10^{scale\_factor} + A.FP$
2. $B_{scaled} = B.IP*10^{scale\_factor} + B.FP$
3. $S_{scaled} = F(A_{scaled}, B_{scaled})$
4. $S.IP = integer(S_{scaled}/10^{scale\_factor})$
5. $S.FP = S_{scaled} - S.IP*10^{scale\_factor}$.

According to specific embodiments of the present invention, a number of shortcuts can be used to improve speed. These shortcuts in fact will significantly speed up many real-world applications.

Rounding according to specific embodiments of the invention is very easy. When performing rounding, if the value of FP is greater than ½ of 10^{scale_factor}, set IP = IP + 1. Otherwise, do not change IP. Set FP to zero.

As discussed above, this aspect of the invention is beneficially used with the method for converting numerical data to display codes as described above because with IP and FP stored separately, the integer portion and the fractional portion can separately be converted to display codes.

Other Implementation Details

It will be understood from the teachings provided herein, that a method according to the present invention can be variously implemented in computing systems. In one implementation, computer-understandable logic instructions related to the present invention can be included in an application program and/or can be invoked by an application program during initiation and/or execution. Note that according to further specific embodiments of the present invention, a numerical encoding scheme or display scheme as described herein can be implemented in an operating system (OS) of a computing device and thereby be made available to any application programs running in the operating system. Thus, according to specific implementations of the present invention, an OS can create a data template and operator routines during OS initiation. It will further be understood from the teachings herein, that logic routines according to the present invention can be included in a logic instruction compiler or logic instruction interpreter and/or include or other files associated with such a programming environment.

Embodiments in an Information Processing Architecture

As discussed herein, according to specific embodiments, the present invention can be embodied in various kinds of information handling system which can include personal digital assistants (PDAs), cellular telephones, television set top systems or cable systems interfaces, toys, home appliances with information handling abilities, scientific and diagnostic systems, and machinery or industrial systems with information handling abilities. Typically, information handling in such systems is performed by binary logic circuits. According to further specific embodiments, the present invention can be embodied in either an information handling system or circuitry or components of an information handling system performing according to the description herein.

According to further specific embodiments, the invention can be embodied as one or more sets of instructions and/or data that are used to program or guide or affect the operation of an information handling system. As is known in the art, these sets of instructions and/or data can be distributed to users stored or recorded on a storage medium, such as a disk, diskette, hard-drive, CD-ROM, tape, ROM, EPROM, ASIC, PLD, etc., and according to specific embodiments, the invention can be embodied as such a medium storing data and/or instructions that when loaded into an appropriately configured information system will cause the system to performing according to the description herein.

As is further known in the art, sets of instructions and/or data can be transmitted to an information handling system over a communication medium (such as the internet, a local area network, a wireless network, a telephone line, a cable-television system, etc.) from a remote data holding location (such as a server) and thereby when loaded into an appropriately configured information system will cause the system to performing according to the description herein.

FIG. 7 illustrates an example architecture of an example information handling system relevant to various specific embodiments of the present invention. As will be understood to those of skill in the art and from the teachings provided herein, the general organization of a system 800 as shown in FIG. 7 is representative of various information systems ranging from computer-on-a-chip type circuits in a household appliance or toy to super computer systems and distributed systems. In some information handling systems, the various components shown in FIG. 7 may be separable computer chips or separable circuit areas on a computer chip, whereas in other information handling systems, some or all of the functions shown in FIG. 7 will be performed by shared circuitry or implemented in software. Some systems will not have all of the components shown in FIG. 7, and other systems will have additional core components. FIG. 7 does not represent the only device architecture on which the present invention can be performed and it will be understood that the present invention is applicable to a variety of types of information processing devices.

An information handling device typically includes one or more processors, such as 801. Processor 801 is generally characterized as being able to perform different logic operations on data, where logic operations are selected or specified by one or more instructions. In the example of a personal computer system or workstation, processor 801 can represent any of the number of well-known microprocessors manufactured by companies such as Intel, AMD, Zilog, and Motorola. Processor 801 can also represent a subset of circuitry configured as a processor in an integrated circuit such as an ASIC or PLD.

A processor 801 can at times work in cooperation with other information handling circuits (which may or may not also be processors) that may have special-purpose abilities. These circuits may be external from the processor or internal with the processor. As an example, FIG. 7 shows a floating point unit (FPU) 802 and a 3D graphics module 804. A processor 801 may also have a number of structures to facilitate its operation, such as, for example, a set of internal registers

806 and/or an arithmetic logic unit (ALU) 808. In some processors, these structures are internal to the processor circuitry.

In most information handling systems, various modules communicate with other modules over one or more communication paths or buses. FIG. 7 shows a representative system bus 810 and a separate auxiliary bus 812. The illustrated buses can represent signal channels on an integrated circuit, communication connections on a printed circuit board, connection between two or more printed circuit board or a back-plane, or any other channels used by the modules to exchange data or control signals.

In various information processing systems, separable modules can include such things as working memory 820, one or more storage systems 830, one or more input interfaces 840, one or more output interfaces 850. Some information systems also include a communication interface (such as a network interface or a modem) 860 for communicating with other computer systems, such as over a network. These modules are shown in FIG. 7 as broadly representative of aspects of a computing system.

In typical information processing systems, working memory 820 is some type of random access memory (RAM) that can be quickly accessed by processor 801 and possibly by other processors. In general purpose computers and other computer systems, during operation, such a working memory contains the data and instructions for one or more processes 822, including operating system processes. Each process generally represents an executing program or program thread. Working memory 820 can also include one or more data structures 824, which may be associated with particular processes or may be shared or system-wide. These data structures can include data tables or any other data structures that can be represented in digital memory. Therefore, in many general purpose information processing systems (such as personal computers) working memory

820 will be understood in the art as containing resident parts of an operating system and/or of various application systems and/or data files and/or other logic modules or digital data.

As is familiar to those skilled in the art, an information processing system that is a general purpose type computer system further generally includes an operating system and at least one application program. The operating system is a set of logic instructions that control the computer system's operation and the allocation of resources. The application program is a set of logic instructions (possibly also including data) to perform tasks desired by the user. During operation, both may be resident in a memory system such as 820.

Storage 830 is illustrated to represent other, usually more long-term (also called non-volatile) data storage. In general purpose computers, this typically includes one or more disk-type systems (such as hard-disk drives, floppy drives, CD-ROMs, etc.) and can also include a variety of other storage devices. Storage 830 can be used to supplement working memory 820 through a variety of known paging techniques. Storage 830 can also include remote storage systems available over a network. In hand-held devices especially, storage 830 may consist sole of read-only-memory (ROM) used to store executable components of the system. Depending on particular implementations, 830 can represent either storage systems that are part of computer system 800 or an interface to external storage systems.

Input interfaces 840 can represent circuits, devices, and/logic or instructions that can provide for video, audio, keyboard, pointer, other input to a computer system. Typical input devices include such things as a keyboard or keypad, touch-screen, mouse, microphone, camera, environmental sensors (e.g. a thermostat or a motion detection), etc. Input interfaces 840, along with possibly other modules in the computer system, handle tasks involved in translating external data (such as key strokes) to the appropriate encoded data (typically binary data). These translation tasks can involve multiple steps, performed in various parts of a computer system. Depending on particular implementations, 840 can represent input devices and associated interface logic or only interface logic to particular input devices.

Output interfaces 850 represents circuits, devices, and/or instructions that can provide for video, audio, print or other output from a computer system and can also represent actual output devices. Typical output devices include a display screen, a printer, a speaker, etc. Output can also be in the form of control signals to an external machine such as an engine, manufacturing robot or other computer-controlled device. Output interfaces 850, along with possibly other modules in the computer system, handle tasks involved in translating computer encoded data (typically binary data) to the appropriate form for output. These translation tasks can involve multiple steps, performed in various parts of a computer system. A display of numerical data, for example, typically requires a conversion from binary encoded numerical values to a series of character codes. These character codes are then further translated by display driver circuits to produce the electrical signals needed to excite various pixels on a CRT or LCD type display.

Communication interfaces 860 represents circuits, devices, and/or instructions that allow a computer system to communicate with other information handling systems, such as over a telephone dial-up connection or over the world-wide internet.

In accordance with the practices of persons skilled in the art of computer programming, the invention according to specific embodiments is described herein with reference to symbolic representations of operations that are performed by an information processing system. Such operations are sometimes referred to as being computer-executed or processorexecuted. It will be appreciated that the operations that are symbolically represented include the manipulation by a CPU or other logic circuitry of electrical signals representing data bits and the maintenance of data bits at memory locations in a memory system, as well as other processing of signals. The memory locations where data bits are maintained are physical locations that have particular electrical, magnetic, optical, or organic properties corresponding to the data bits.

Thus, it will be understood from the teachings herein that the present invention can, according to specific embodiments, be embodied into an information handling system and/or into different separable components of an information handling system.

Embodiments in a Programmed System

FIG. 8 is a block diagram showing a representative example logic device in which various aspects of the present invention may be embodied. The invention can be implemented in hardware and/or software. The invention may be embodied in a fixed media or transmissible program component containing logic instructions and/or data that when loaded into an appropriately configured computing device cause that device to perform according to the invention. FIG. 8 shows digital device 700 that may be understood as a logical apparatus that can read instructions from media 717 and/or network port 719. Apparatus 700 can thereafter use those instructions to direct a server or client application as is known in the art and that further includes the components of the invention. One type of logical apparatus that may embody the invention is a computer system as illustrated in 700, containing CPU 707, optional input devices 709 and 711, disk drives 715 and optional monitor 705. Fixed media 717 may be used to program such a system and may represent a disk-type optical or magnetic media or a memory. The invention may be embodies in whole or in part as software recorded on this fixed media. Communication port 719 may also be used to program such a system and may represent any type of communication connection.

The invention also may be embodied in whole or in part within the circuitry of an application specific integrated circuit (ASIC) or a programmable logic device (PLD). In such a case, the invention may be embodied in a computer understandable descriptor language which may be used to create an ASIC or PLD that operates as herein described.

The invention has now been explained with reference to specific embodiments. Other embodiments will be apparent to those of skill in the art. It is understood that the examples and embodiments described herein are for illustrative purposes only and that various modifications or changes in light thereof will be suggested to persons skilled in the art and are to be included within the spirit and purview of this application and scope of the appended claims. All publications, patents, and patent applications cited herein are hereby incorporated by reference in their entirety for all purposes.

WHAT IS CLAIMED IS:

1. A method of determining character codes for a binary encoded numerical original value using an information processing apparatus comprising:
   a. dividing said original value by a divisor that will produce a two or more digit integer result;
   b. using said two or more digit result to determine two or more display codes;
   c. determining a secondary original value;
   d. repeating steps a through c until said secondary original value represents less than a predetermined number of digits; and
   e. using a final secondary original value to determine a final two or more display codes.

2. A method according to claim 1 wherein said integer result is an integer remainder and said determining comprises dividing said original number by a base raised to a power indicating the number of digits decoded.

3. A method according to claim 1 wherein said integer result is an integer quotient and said determining comprises:
   subtracting said integer result multiplied by said divisor from said original value to get a secondary original value.

4. A method according to claim 1 wherein said divisor is an integer exponential power of 10 greater than 10.

5. A method according to claim 1 wherein said divisor is an integer exponential power of 10 greater than 1000.

6. A method according to claim 1 wherein said using comprises:
   performing a table lookup using said two or more digit result as an index to a table.

7. A method of determining character codes for a binary encoded numerical original value using an information processing apparatus comprising:

a. dividing said original value by a divisor to produce a two or more digit remainder integer result;

b. using said two or more digit result to determine two or more display codes;

c. subtracting said integer result multiplied by said divisor from said original value to get a secondary original value;

d. repeating steps a through c until said secondary original value is less than a predetermined number of digits; and

e. using a final secondary original value to determine a final two or more display codes.

8. A method according to claim 7 wherein said divisor is an integer exponential power of 10 greater than 10.

9. A method according to claim 7 wherein said divisor is an integer exponential power of 10 greater than 1000.

10. A method according to claim 7 wherein said using comprises:

performing a table lookup using said two or more digit result as an index to a table.

11. An apparatus in a computing system converting a binary encoded number to a set of display codes comprising:

a table having a plurality of entries, each entry providing two or more display codes for two or more digits; and

a processor able to divide a binary encoded number by a divisor and use results therefrom to look-up two or more display codes in said static table.

12. An apparatus according to claim 11 further wherein said result is used as an index to said table.

13. An apparatus according to claim 11 further wherein said result is a remainder result of said division.

14. An apparatus according to claim 11 further wherein said result is an integer quotient result of said division.

15. An apparatus according to claim 11 wherein each indexed entry in said table has a number N of digit display codes and wherein said table has 10^N indexed entries and wherein N is an integer greater than one.

16. A method allowing an information handling system to more quickly execute programs requiring conversion of binary encoded numbers to character codes comprising:

constructing a lookup table in a memory of said information handling system wherein said lookup table is indexed by a value and wherein entries in said lookup table represent two or more display codes corresponding to said value; and

establishing a logic routine that accepts a binary encoded numerical value and uses said lookup table to determine display codes for said binary encoded numerical value.

17. A method according to claim 16 wherein said constructing comprises:

constructing a static lookup table in an operating system memory space of said information handling system.

18. A method according to claim 16 further comprising:

selecting an integer greater than one indicating the number of digit display codes in each entry in a look-up table.

19. A method of speeding up operation of a computer system comprising:

establishing a logic route for displaying binary encoded numbers wherein said logic routine determines two or more display code representations of a binary encoded number at each iteration through a conversion routine.

20. A method of storing a numerical value in an information processing apparatus comprising:

a. reserving a first storage area and storing an integer portion of said numerical value in said first storage area;

b. reserving a second storage area and storing a fractional portion of said numerical value in said second storage area;

c. wherein said first storage area and said second storage area are non-contiguous in that there is not automatic carry or bit shifting between said first and second storage area and wherein an integer portion of a decimal value can be fully determined by reference to an integer portion of said stored value and wherein a fractional portion of a decimal value can be fully determined by reference to a fractional portion of said stored value.

21. A method according to claim 20 wherein said integer portion is stored in said first storage area according to a standard binary integer format.

22. A method according to claim 21 wherein said integer portion is stored in said first storage area as a signed 2's complement binary integer.

23. A method according to claim 20 wherein said fractional portion is stored in said second storage area according to a standard binary integer format.

24. A method according to claim 23 wherein said fractional portion is multiplied by a scale value and then stored in said second storage area according to a standard binary integer format.

25. A method according to claim 24 wherein said scale value is an integer power of 10.

26. A method according to claim 23 wherein said fractional portion is stored in said first storage area as a signed 2's complement binary integer.

27. A method according to claim 20 wherein said integer portion and said fractional portion are each stored with a separate sign bit.

28. A method according to claim 20 further comprising determining character codes for a numerical stored as an integer portion and an fractional portion comprising separately for said integer portion and said fractional portion determining said character codes.

29. An apparatus in a computing system for handling floating point numbers comprising logic modules to perform the method as recited in claim 20.

30. A method allowing an information handling system to handle a range of floating-point numbers comprising:
  establishing a IP.FP data template in a memory of said information handling system wherein said data template provides non-contiguous storage areas for the decimal part and fractional part of numerical values;
  establishing a plurality of logic routines for performing numerical and logic operations on numerical values stored in IP.FP format.

31. A method of speeding up operation of a computer system comprising operating said system in accordance with claim 30 when handling floating point numbers.

INVESTOR IN PEOPLE

Application No:	GB 0205775.0	**Examiner:**	Michael Powell Waters
Claims searched:	1 to 10	**Date of search:**	16 October 2002

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:

UK Cl (Ed.T): G4A (AAF, AAU, ACC, ACX)

Int Cl (Ed.7): G06F (5/00, 5/01, 17/21, 17/22) H03M

Other: ONLINE: WPI, EPODOC, PAJ, TXTE, INSPEC, ELSEVIER, IBM TDB, IEEE Xplore, Internet

Documents considered to be relevant:

Category	Identity of document and relevant passage		Relevant to claims
A	US 6396921 B1	(LONGSTER) see figures 3 to 5	
A	JP 050289846 A	(MATSUSHITA) see WPI and PAJ abstracts	
A	JP 020034037 A	(HITACHI) see WPI and PAJ abstracts	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
		E	Patent document published on or after, but with priority date earlier than, the filing date of this application.
&	Member of the same patent family		