Storage-Efficient Representation of Decimal Data

Tien Chi Chen
IBM San Jose Research Laboratory
and
Irving T. Ho
IBM Systems Products Division

Usually n decimal digits are represented by $4n$ bits in computers. Actually, two BCD digits can be compressed optimally and reversibly into 7 bits, and three digits into 10 bits, by a very simple algorithm based on the fixed-length combination of two variable field-length encodings. In over half of the cases the compressed code results from the conventional BCD code by simple removal of redundant 0 bits. A long decimal message can be subdivided into three-digit blocks, and separately compressed; the result differs from the asymptotic minimum length by only 0.34 percent. The hardware requirement is small, and the mappings can be done manually.

Key Words and Phrases: binary-coded decimal digits, decimal data storage
CR Categories: 6.32, 6.34

Authors' addresses: T.C. Chen, IBM San Jose Research Laboratory, Dept. K54-282, Montery & Cottle Roads, San Jose, CA 95193; I.T. Ho, Dept. 55A IBM Systems Products Division, Hopewell Junction, NY 12533.

1. Introduction

Since present-day digital electronics is based on binary signals, it is straightforward to treat computer numbers in the binary notation. Throughout the past quarter-century, binary arithmetic has been widely promoted as the wave of the future.

Nevertheless, the decimal habit, deeply ingrained in the human society for millennia, has proved unshakable. The predominant use of decimal notation for machine input-output is a clear indication that users strongly prefer to stay with decimal arithmetic.

As the cost of hardware continues to decrease, and the cost of programming and debugging continues to increase, the productivity of the human computer user is becoming the overriding concern.

Rather than trying to convert users to think in the binary mode, it now appears vastly more rewarding to equip machines with decimal arithmetic.

The current practice is to deal with users in decimal numbers, yet employ binary arithmetic inside the machines. This is becoming increasingly untenable. First, the radix conversion for nonintegers introduces errors; repeated two-way conversion may even decrease accuracy without limit, as shown by Matula [1].

More alarming, binary algorithms create roundoff situations quite intractable from the viewpoint of decimal arithmetic. The quantity "one-tenth" is easy to express in decimal, yet it is a recurrent binary fraction. The converse, incidentally, does not occur; all finite binary fractions map into finite decimal fractions. This is because the radix 10 contains 2 as a factor.

The question of decimal arithmetic in computers should therefore be reopened. Design convenience should no longer be an issue, nor cost of hardware. The significant hardware issues are operation speed and storage efficiency. This note addresses the latter problem. For a discussion of decimal arithmetic performance see Schmookler and Weinberger [2].

We shall show that three-decimal digits in the standard BCD format can be mapped easily and reversibly into 10 bits, within 0.34% of the maximum allowable efficiency. Likewise two-decimal digits can be mapped into 7 bits, deviating only 5.1% from optimum. The scheme is based on minimum alteration of the BCD bit patterns, and requires no arithmetic, only simple logic, deletions, displacements, and insertions.

2. Storage Efficiency

Four bits are needed to represent a single decimal digit in binary technology; the BCD (binary-coded decimal) scheme is most commonly used, in which the digit A is rendered as a string of 4 bits $(abcd)$, such that $A = 8a + 4b + 2c + d$. Similarly, a pair of decimal digits A,B is rendered as $(abcd)(efgh)$, and a triplet $A,B,C,$ as $(abcd)(efgh)(ijkl)$, and so on.

The 4-bit-per-digit approach does not economize storage space, as 4 bits can already cover 16 states instead of 10. The relative inefficiency of BCD encoding versus the binary notation is measured by the ratio

$$r \equiv 4 \log 2/\log 10 = 1.20411\ 99826\ 55924\ 78085.$$

Thus a space saving of up to 20.4% can be expected with a binary representation.

For n decimal digits, the BCD notation would require $N_n = 4n$ bits. Let M_n be the minimum number of bits to convey the same information. Being an integer, M_n is not just N_n/r, but is its "ceiling," that is, the smallest integer no less than N_n/r, symbolized by $M_n = \lceil N_n/r \rceil$. The actual improvement in storage economy is measured by the compression ratio $r_n \equiv N_n/M_n = 4n/\lceil 4n/r \rceil$, which fluctuates with n, but tends to r_∞ as the limit for large n.

Table I gives the successively higher values of r_n as n varies from 1 to 100,000. We see that $r_2 = 8/7$ is already within 5.1% of r, and the next case $r_3 = 6/5$ is even better, within 0.34%. After these comes r_{31} within 0.02%, then r_{59} within 0.003%. Cases beyond 59 are probably only of academic interest.

A long decimal message can be subdivided into blocks of n bits each, and be "compressed" independently. The compression ratio of 8/7 and 6/5 are already quite attractive. The mapping can be done using radix conversion, but the scheme to be discussed is simpler, faster, and takes less hardware. It can even be done visually by human operators.

Table I. Compression of Strings of Decimal Digits

No. decimal digits (n)	No. BCD bits (N_n)	Min. no. encoded bits (M_n)	Compression ratio ($r_n = N_n/M_n$)	Relative deviation from asymptotic limit (percent)
1	4	4	1	16.95
2	8	7	1.14285 71429	5.088
3	12	10	1.2	.3422
31	124	103	1.20388 34951	.01964
59	236	196	1.20408 16327	.003185
205	820	681	1.20411 16001	7.0×10^{-6}
351	1404	1166	1.20411 66381	2.7×10^{-6}
497	1988	1651	1.20411 87159	1.1×10^{-6}
643	2572	2136	1.20411 98502	1.1×10^{-7}
4647	18588	15437	1.20411 99715	9.3×10^{-9}
8651	34604	28738	1.20411 99805	1.8×10^{-9}
21306	85224	70777	1.20411 99824 8	1.5×10^{-10}
97879	391516	325147	1.20411 99826 54	1.6×10^{-12}
∞	∞	∞	1.20411 99826 55925	0

3. Fixed-Length Code from Two Variable-Length Encodings

In the BCD notation for a single digit ($abcd$), 80% of the states (0–7) can be represented by 3 bits, namely (bcd). For the remaining two states (8, 9), 4 bits are commonly employed, but two of these (bc) are predictably 0, and can actually be omitted. We can view the BCD digit to have two parts; the leading bit (a) is a magnitude indicator, $a = 1$ if and only if the digit is "large," with a value of 8 or higher. The second part, (bcd) for small digits and (d) alone for large ones, gives the detailed value.

By simply dropping the redundant 0 bits in a BCD message, a condensed code naturally results. The magnitude indicator for each digit serves also as length indicator for each digit. This code is unambiguous and reversible, but only moderately efficient; the compression ratio is $10/9 = 1.11$.

Variable-length messages are difficult to decode, are highly sensitive to errors, and have an uncertain storage requirement. The scheme above is not really recommended. Further, we wish to use fewer than 3.6 bits per digit.

A more desirable encoding can result from the combination of two fields. One of these is an "indicator" field, showing the location of the large digits, if any. It is a recoding of the collection of n indicator bits. The other is a "detail" field, which in conjunction with the indicator field will give the actual value of the digits. To reduce the encoding effort, the detail field is just an arrangement of the "detail value" bits of the individual BCD digits.

There are 2^n possible indicator states, ranging from "all large" to "all small." The corresponding detail fields vary in length, from n bits to $3n$ bits. To produce a combined fixed length encoding, the indicator field must have compensating variable length, with shorter encoding for the more common smaller magnitudes.

This suggests Huffman encoding [3] for the indicator field, which also guarantees the shortest average length. As is well known, the bit pattern in Huffman encoding is not unique; given one Huffman code, one can obtain another by interchanging 0's and 1's in any column. The extra freedom available will be exploited to simplify the encoding process and preserve salient features of the BCD notation.

4. Compressing Two BCD Digits into 7 Bits

For a pair of decimal BCD digits $A,B = (abcd)(efgh)$, the indicator bits (a,e) provide four indicator states, namely:

(1) $a,e = 0,0$:	both digits small; occurrence 64%, detail part has 6 bits ($bcdfgh$).
(2) $a,e = 1,0$:	A alone is large; occurrence 16%; detail part has 4 bits ($dfgh$).
(3) $a,e = 0,1$:	B alone is large; occurrence 16%; detail part has 4 bits ($bcdh$).
(4) $a,e = 1,1$:	both digits large; occurrence 4%; detail part has 2 bits (dh).

Applying the principles of Huffman encoding, the indicator field lengths are 1, 2, 3, and 3, respectively. The combination with the proper detail field, with suitable fill-ins, yields a code with the fixed length of 7 bits, which is the best possible from two-decimal digits.

A reasonable scheme is given in Figure 1, where the result bits are called $(pqrstuv)$.

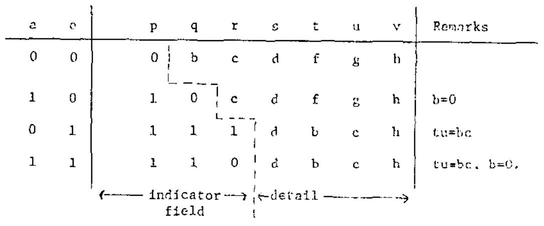
The scheme is easily mastered by human users, as follows:

Encoding: $pqrstuvw = abcdfgh$	if $e = 0$	(80%)
$= 11\bar{a}dbch$	otherwise.	
Decoding: $abcdefgh = pqrs0tuv$	if $p \cdot q = 0$	(80%)
$= \bar{r}tus100v$	otherwise.	

Other 7-bit mappings can be similarly constructed; but the ones here are distinguished by the following features. (1) There is no arithmetic action, only permutation, deletions, and insertions. (2) Only 2-bits (a,e) are examined to determine the encoding. (3) For 80% of the cases (namely small B), there is no significant change in the BCD bit pattern, only the omission of a 0 bit. (4) The bits (d, h) are mapped into fixed positions. (5) The parity of the original BCD digits is preserved, and need not be generated separately.

The expansion back to the 8-bit BCD notation is easily done, as seen in Figure 2. The Boolean logic for the mapping and remapping is given in Table II.

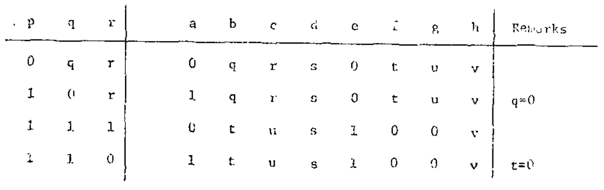
Fig. 1. The compression of two BCD digits $(abcd)(efgh)$ into 7 bits $(pqrstuv)$. The parity is conserved. The scheme works even if $c = 1$ in all cases.



a	e	p	q	r	s	t	u	v	Remarks
0	0	0	b	c	d	f	g	h	
1	0	1	0	c	d	f	g	h	b=0
0	1	1	1	1	d	b	c	h	tu=bc
1	1	1	1	0	d	b	c	h	tu=bc, b=0.

←— indicator —→ field
←detail ——→

Fig. 2. The expansion back to two BCD digits



p	q	r	a	b	c	d	e	f	g	h	Remarks
0	q	r	0	q	r	s	0	t	u	v	
1	0	r	1	q	r	s	0	t	u	v	q=0
1	1	1	0	t	u	s	1	0	0	v	
1	1	0	1	t	u	s	1	0	0	v	t=0

Table II. Boolean Logic in the two-digit, 7-bit Mapping

(*a*) Compression into 7 bits $(pqrstuv)$	(*b*) Decompression into 8 bits in BCD format $(abcd)(efgh)$
$p = a + e$	$a = p(\bar{q} + \bar{r})$
$q = b + e$	$b = q(\bar{p} + t)$
$r = c\bar{e} + \bar{a}e$	$c = r(\bar{p} + \bar{q}) + pqu$
$s = d$	$d = s$
$t = f\bar{e} + be$	$e = pq$
$u = g\bar{e} + ce$	$f = t(\bar{p} + \bar{q})$
$v = h$	$g = u(\bar{p} + \bar{q})$
	$h = v$

Incidentally, one can allow A to count up to 11, and our mapping scheme remains valid. The bit (c) is allowed to be 0 or 1 in all four magnitude cases, and the same 7-bit encoding can cover 120 "BCD" states correctly and reversibly.

5. Compressing Three BCD Digits into 10 Bits

The same principle of Huffman encoding applies to the three-digit mapping problem. There are $2^3 = 8$ cases to consider, subdivided into four indicator groups; Huffman encoding is again used, based on the 3 indicator bits (a,e,i).

(1) All digits are small (51.2%);
encoded by (0).
(2) One digit is large (38.4%);
encoded by (100), (101), (110).
(3) Two digits are large (9.6%);
encoded by (11100), (11101), (11110).
(4) All digits are large (0.8%);
encoded by (11111).

The first three groups, with indicator field lengths of 1, 3, and 5 bits, respectively, mesh neatly with the corresponding detail field lengths (9, 7, 5) to yield a total code length of 10 bits. Only the last group shows a slight redundancy, where a total of $5 + 3 = 8$ bits are padded into 10 bits also.

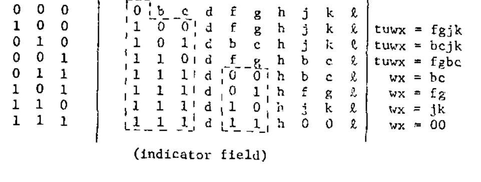
Again the encoding is not unique. The scheme shown in Figure 3 breaks the indicator field into two parts, to emphasize similarity to the original BCD encoding. The result bits are $(pqrstuvwxy)$, with (s,v,y) always standing for (d,h,l) of the BCD digits.

We note again, there is no arithmetic, only shifts, deletions, and insertions, based on the testing of 3 bits (a,e,i). The two cases with $e,i = 0,0$ (combined probability 64%) are obtained from the BCD notation by simple deletion. Unlike the two-digit case above, there is no uniform parity preservation. The compression ratio is $12/10 = 6/5 = 1.2$, within 0.34% of the asymptotic limit r.

The reverse mapping is shown in Figure 4. The Boolean logic for the transformations is given in Table III. The mapping algorithm suitable for human use is given in Table IV.

Incidentally, there is an earlier three-digit to 10-bit compression scheme by Hertz [4], which subdivides a

Fig. 3. The compression of three BCD digits $(abcd)(efgh)(ikjl)$ into 10 bits $(pqrstuvwxy)$



a	e	i	p	q	r	s	t	u	v	w	x	y	Remarks
0	0	0	0	b	c	d	f	g	h	j	k	ℓ	
1	0	0	1	0	0	d	f	g	h	j	k	ℓ	tuwx = fgjk
0	1	0	1	0	1	d	b	c	h	j	k	ℓ	tuwx = bcjk
0	0	1	1	1	0	d	f	g	h	b	c	ℓ	tuwx = fgbc
0	1	1	1	1	1	d	0	0	h	b	c	ℓ	wx = bc
1	0	1	1	1	1	d	0	1	h	f	g	ℓ	wx = fg
1	1	0	1	1	1	d	1	0	h	j	k	ℓ	wx = jk
1	1	1	1	1	1	d	1	1	h	0	0	ℓ	wx = 00

(indicator field)

Table III. Boolean Logic in the three-digit, 10-bit Mapping

(a) Compression into bits $(pqrstuvwxy)$

$$
\begin{aligned}
p &= a + e + i\\
q &= b\bar{e} + i + ae\\
r &= c\bar{i} + e + ai\\
s &= d\\
t &= ae + f(\bar{a} + \bar{i}) + be\bar{i}\\
u &= ai + ce\bar{i} + g\bar{e}\\
v &= h\\
w &= j + bi + fai\\
x &= k + ci + gai\\
y &= l
\end{aligned}
$$

(b) Decompression into 12 bits in BCD format $(abcd)(efgh)$-$(ijkl)$

$$
\begin{aligned}
a &= p\bar{q}\bar{r} + pqr(t + u)\\
b &= q\bar{p} + t\,p\bar{q}r + wq(\bar{r} + \bar{t}\bar{u})\\
c &= r(\bar{p} + \bar{q}u) + xpq(\bar{r} + \bar{t}\bar{u})\\
d &= s\\
e &= pr(\bar{q} + \bar{u} + t)\\
f &= t(\bar{p} + \bar{r}) + pqr\bar{t}uw\\
g &= u(\bar{p} + \bar{r}) + pqr\bar{t}ux\\
h &= v\\
i &= pq(\bar{r} + \bar{t} + u)\\
j &= w(\bar{p} + \bar{q} + rt)\\
k &= x(\bar{p} + \bar{q} + rt)\\
l &= y
\end{aligned}
$$

Table IV. Algorithms for Mapping Between Three BCD Digits and 10 Bits

(a) Encoding of $(abcd)(efgh)(ijkl)$ into $(pqrstuvwxy)$

```
If e,i = 0,0 (64% occurrence)

    then                  p q r s t u v w x y = a b c d f g h j k ℓ

    else if a = 0 (28.8%)

              then        p q r s t u v w x y = 1 i e d f g h j k ℓ

                  and if i = 1 then add         . . . . . . . b c .

                               else add         . . . . b c . . . .

              else (7.2%) p q r s t u v w x y = 1 1 1 d e i h j k ℓ

                               add              . . . . . . . b c .

                               and add          . . . . . . . f g .
```

(b) Decoding back into three BCD digits

```
If p = 0, or q,r = 0, 0 (64% occurrence)

    then                 a b c d e f g h i j k ℓ = p q r s 0 t u v 0 w x y

    else if q·r = 0, or q,r,t,u = 1 1 0 0 (28.8%)

              then       a b c d e f g h i j k ℓ = 0 0 0 s r 0 0 v q 0 0 y

                  and if q,r = 0,1 then add        . t u . . . . . . w x .

                                   else add        . w x . . t u . . . . .

              else       a b c d e f g h i j k ℓ = 1 0 0 s t 0 0 v u 0 0 y

                  and if t = 0     then add        . . . . . w x . . . . .

                                   else add        . . . . . . . . . w x .
```

digit into three categories (<8, $=8$, $=9$), and the three-digit combination receives $3^3 = 27$ subdivisions. The mappings are correspondingly more complicated.

6. Conclusion

We have demonstrated that the conventional BCD digits can be compressed by simple algorithms into fixed-length codes, two or three digits at a time, to achieve a storage efficiency up to within 0.34% of the asymptotic maximum. The scheme, based on the meshing of two variable-length fields, involves minimum alteration of the original BCD code; in over half of the cases, the condensed code is obtained by dropping redundant 0 bits from the BCD counterpart.

The mapping hardware is simple and fast; its cost, in this era of large scale integration, is small. It is feasible to put the compression/decompression mechanism at the interface between the arithmetic unit, which may require the expanded BCD format, and the memories, especially the low-speed file storage, where information tends to remain for long durations. The compressed storage will mean a saving of 20%. The transformation can even be done visually, with a little practice; this fact should be useful in debugging.

Fig. 4. The decompression back to three BCD digits.

p	q	r	t	u	a	b	c	d	e	f	g	h	i	j	k	ℓ	Remarks
0	q	r	t	u	0	q	r	s	0	t	u	v	0	w	x	y	aei = 000
1	0	0	t	u	1	0	0	s	0	t	u	v	0	w	x	y	aei = 100, fgjk = tuwx
1	0	1	t	u	0	t	u	s	1	o	o	v	0	w	x	y	aei = 010, bcjk = tuwx
1	1	0	t	u	0	w	x	s	0	t	u	v	1	0	0	y	aei = 001, fgbc = tuwx
1	1	1	0	0	0	w	x	s	1	0	0	v	1	0	0	y	aei = 011, bc = wx
1	1	1	0	1	1	0	0	s	0	w	x	v	1	0	0	y	aei = 101, fg = wx
1	1	1	1	0	1	0	0	s	1	0	0	v	0	w	x	y	aei = 110, jk = wx
1	1	1	1	1	1	0	0	s	1	0	0	v	1	0	0	y	aei = 111

We have thus shown that storage efficiency is no longer a serious disadvantage of decimal machines. Schmookler and Weinberger [2], on the other hand, had already shown that efficient decimal adder units can be constructed at a small added cost in hardware. Perhaps the final hurdle confronting decimal machines is not hardware, but the dearth of error analyses of decimal algorithms; this should interest numerical analysts.

Acknowledgments. One of us (T.C. Chen) is indebted to Joseph D. Rutledge and John C. McPherson for stimulating discussions.

Received April 1974

References

1. Matula, D.W. A formalization of floating-point numeric base conversion. *IEEE Trans. Computers C-19* (Aug. 1970), 681–692.
2. Schmookler, M.S., and Weinberger, A.W. High speed decimal addition. *IEEE Trans. Computers C-20* (Aug. 1971), 862–867.
3. Huffman, D.A. A method for the construction of minimum redundancy codes. *Proc. IRE 40* (Sept. 1952), 1098–1101.
4. Hertz, T.M. System for the compact storage of decimal numbers. U.S. Patent 3,618,047, Nov. 2, 1971.