Date: March 29, 1971
From (location or U.S. mail address): San Jose Research Laboratory
Dept. & Bldg: K08/028
Tieline & Tel. Ext.: 6804

IBM

Subject: Decimal Number Compression

Reference:

To: Dr. Irving T. Ho

The fact that four bits can represent 16 different states, but a decimal digit exploits only 10 of them (0 to 9) has been a valid criticism against decimal arithmetic.

On the other hand, it is well-known that a number with several decimal digits can be reexpressed into binary, leading to a 20% gain in the number of bits used. Examples are, two decimal digits (8 bits) reexpressed as a seven-bit number, and three decimal digits (twelve bits) reexpressed as a ten-bit number. Fast conversion from decimal to binary is the subject of a previous memo [Reference 1]. I have studied the reverse conversion and hope to have interesting results shortly.

1. Bit Pattern Preservation

There is no question that the equivalent binary number represents the closest a priori packaging scheme of the number. Further compression may be possible, but would require a detailed knowledge of the bit patterns.

However, the binary mapping produces a garbled result which is not immediately readable, and the forward and backward mapping technique call for hardware arithmetic. The question therefore arises, do there exist techniques of equivalent compression power which preserve the bit patterns to a degree?

The answer is in the affirmative. There exist reversible schemes mapping 2 decimal digits into 7 bits or 3 decimal digits into 10 bits, which preserve the non-zero bit patterns.

2. Mapping of Two Digits - Method A

Eighty percent of the decimal digits can be written in three bits. The exceptions are 8 ("1000") and 9 ("1001"). We note that the second and third bits are zeros here.

We could, therefore, conceive of a flag indicating whether the digits exceed 7, and if so, which ones. The flag could be so constructed that the usual case (three bits per digit) has a minimum flag (one bit). The next usual case has a longer flag (two bits) and so on.

A possibility is as follows: the decimal digit pair is assumed to have the binary encoding of (abcd) (efgh). We have

0bcdfgh	for	a=0, e=0	
10Xdfgh	for	a=1, e=0	X means "don't care"
110hbcd	for	a=0, e=1	
111dXXh	for	a=1, e=1	

We shall assume X to be 0 by convention.

To implement the transformation, we need a pair of 4-bit registers, P,Q, whose bit positions are labelled rstu,vwxy, respectively.

Put abcd in rstu, efgh in vwxy, then

if v = 0, read out rstuwxy whatever the state of r.
if v = 1, r = 0 read out v1xystu (note the change of order)
if v = 1, r = 1 read out v11uwxy.

the result will be the encoding above

To implement the decoding, we use the same registers. Let the encoded bits be ijkℓmno, we start by putting ijkℓ in rstu, 0mno in vwxy, then

if r = 0, read out rstuvwxy,
if $r\bar{s}$ = 1, read out rstuvwxy also
if rst = 1, read out vwxyr0tu (note change of order)
else, (rst = 1) read out r00u100y

The result will be abcdefgh.

<u>3. Method B for Mapping Two Digits</u>

The following scheme is simpler in controls, and <u>vastly more readable.</u>

0bcdfgh	for	a=0, e=0
10defgh	for	a=1,
11habcd	for	e=1.

This scheme has an ambiguity. When a = 1, e = 1, which encoding should be used? The tie breaking (if necessary) should favor the use of 10defgh because of the readability and ease of transformation.

To encode, we use the same registers P = rstu, Q = vwxy and start by abcd going to P, efgh going to Q. Then

if r = 0, v = 0, read out rstuwxy
if r = 1, regardless of v, read out r0uvwxy
else, read out 11yrstu.

To decode, we put ijkℓ in rstu, ℓmno in vwxy, then

if r = 0 read out rstu0wxy (note zeroing of v position)
if r$\bar{s}$ = 1, read out rs0tvwxy (note insertion of 0)
else (rs = 1) read out vwxyrs0t (note exchange)

And the result will be correct. We note that the data handling is harder, but the simplification of decoding and the readability far exceeds this slight disadvantage.

4. Mapping of 3 Digits into 10 Bits

Assuming the digits are (abcd) (efgh) (ijkℓ), then we can encode as

0bcdfghjkℓ	if a = 0, e = 0, i = 0	
100dfghjkℓ	if a = 1, e = 0, i = 0	
101hjkℓbcd	if a = 0, e = 1, i = 0	
110ℓbcdfgh	if a = 0, e = 0, i = 1	
11100hℓbcd	if a = 0, e = 1, i = 1	
11101ℓdfgh	if a = 1, e = 0, i = 1	
11110dhjkℓ	if a = 1, e = 1, i = 0	
11111dhℓXX	if a = 1, e = 1, i = 1	(X = "don't care")

The efficiency is equal to the binary conversion (12 bits mapped into 10 bits) and the encoding/decoding requires no arithmetic. The uninitiated reader (including myself) will need a table to decipher the bits, but he is spared the need to do multiplications or divisions.

The hardware implementation requires 3 registers each of 4 bits. The read out is either straightforward, or calls for shifts / and/or bit selection.

5. Conclusion

We have demonstrated that decimal data compression to "binary conversion efficiency" is achieved without any arithmetic, using essentially the same hardware for forward and backward compression.

I benefited from a conversation with Dr. Frank Tung to verify the results.

March 29, 1971

6. References

1. "Decimal-binary integer conversion scheme" memo by T. C. Chen to Dr. Irving T. Ho, March 12, 1971. Related references are also found therein.

T. C. Chen/pjc

T. C. Chen

/pjc

cc: Dr. G. C. Bacon, San Jose Research
Mr. C. J. Conti, SDD Poughkeepsie
Mr. C. M. Davis, SDD Poughkeepsie
Dr. H. Fleisher, SDD Poughkeepsie
Mr. A. A. Magdall, SDD Endicott
Mr. J. C. McPherson, CHQ Armonk
Mr. A. Peacock, SDD Poughkeepsie
Mr. N. Rochester, SDD Poughkeepsie
Dr. M. S. Schmookler, SDD Poughkeepsie
Dr. W. S. Worley, Jr., SDD Poughkeepsie