Griffith

COMPANY CONFIDENTIAL

PROJECT STRETCH

FILE MEMO #46

SUBJECT: Binary-Decimal and Decimal-Binary Conversions
BY: H. C. Montgomery and W. Wolensky
DATE: September 21, 1956

Summary

A number of conversion techniques are mentioned and given an approximate evaluation using criteria of interest in computer applications. In some detail a modified table look-up method is described which appears to have comparative advantages of speed, flexibility, and logical simplicity.

The Methods Considered (see references 2 and 3 for detailed procedures)

1. Repeated Division. In this method the number to be converted, the model number N_m, is divided (or multiplied if N_m is a fraction) by the image radix. The partial remainder is the least significant bit of the image number N_i. The process is repeated until the partial quotient is zero.

2. Repeated Subtraction. Using this method one subtracts from N_m the highest power of the image radix which is less than or equal to N_m. A 1 placed in the appropriate bit position of the conversion image N_i of N_m indicates that a subtraction has been made. Taking the partial difference, successively decreasing powers of the image radix are tested until a subtraction yields a non-negative difference. Powers of the radix not subtracted appear as zeros in the corresponding bit positions of N_i. The process continues until a partial difference equaling a power of the image radix is obtained.

3. Boehm Shift Method. Described by Miss E. Boehm and Mr. J. Griffith in the 704 literature, this method combines features of a number of other methods in a fast but complex shifting-adding-testing procedure.

4. Summation Method. The most significant bit of N_m is multiplied by the model radix expressed in the image radix form. The next bit is added to this partial product and the multiplication repeated. The procedure continues until the least significant bit has been added to the preceding partial product.

5. Modified Table Look-up. N_m is converted a byte at a time. The conversion images of the bytes are extracted from the machine memory and added together to form N_i. The images in the machine memory represent previously computed conversions for all possible combinations of the byte arrays.

   An approximate evaluation of these methods according to six criteria is summarized in the table on page 6.

In each of the methods listed above, provision must be made for converting floating point numbers. The method described in some detail below seems to be the most favorable of those with which we are familiar at the present time. It is a combination of a table look-up scheme and a method described by Dr. W. W. Peterson in his paper "Editor and Translator PK EDIT".

Notation

The following notation proves to be convenient:

1. n = the number of bits in one word.

2. N^f = the fraction part of the floating point number N.

3. N^e = the exponent part of the floating point number N.

4. S = the address of the first cell of the block of memory in which column three of the look-up table is stored. (see appendix 1.)

5. d = the number of bits used to express exponents of floating point numbers.

6. k = the highest power of 10 which can be expressed in binary form using n bits. $n = 3(k-1) + h/3$, where h is the multiple of 3 nearest to k.

The Method

Fixed point to fixed point for integers or fractions.

1. Binary to BCD.

   a. Restriction: using the 4 bit BCD code, $10^{-n/4} \leq N_m < 10^{n/4}$.

b. Arithmetic units required: both the binary and BCD units. (the BCD unit may be used alone if the byte size is restricted to be fewer than 4 bits.)

c. Tables: a sample table is given in appendix 1-a.

d. The procedure:

1. To the number S add binary-wise byte one of N_m.

2. To the contents of some register or memory cell A (whose contents are initially zero) add BCD-wise the contents of the memory cell whose address is the sum found in step one.

3. Repeat steps one and two for the remaining bytes of N_m, each time incrementing the constant S of step one by 2^p, where p is the number of bits in a byte. Upon completion of step two for the last byte, C (A) = N_i.

2. BCD to Binary.

a. Restrictions: if the 4 bit BCD code is used, then the byte length should be an integral multiple of four. For fractions, an error of one-half is possible in the last place for each byte. If N_m does not exceed one word in length, then no overflow occurs.

b. Arithmetic units required: only the binary unit need be used.

c. A sample look-up table is given as appendix 1-b.

d. The procedure: the procedure is the same as given above for the inverse conversion, except:

1. Binary addition is used throughout.

2. In step three, S is incremented by $10^{p/4}$ rather than by 2^p.

Floating Point to Floating Point

1. Binary to BCD.

a. Restriction: depending upon the word length, overflow may occur.

b. Arithmetic units required: both the binary and BCD units are used.

c. The procedure:

1. Divide or multiply N_m by 10^k in floating binary as many times r as necessary to get N_m^1 such that $1 \leq N_m^1 < 10^k$. (Retain the number rk for later use in developing N_1^e.)

2. Convert N_m^1 to a fixed point binary number N_m^{11} of length 2_n bits with the binary point in the middle.

3. Divide N_m^{11} by 10^q, where q is such that $10^{q-1} \leq N_m^{11} < 10^q$. Let $\bar{N}_m^{111} = N_m^{11} \cdot 10^{-q}$.

4. Round $\bar{N}_m^{111}$ by dividing one-half by 10^{n-d} and adding it to $\bar{N}_m^{111}$ to get N_m^{111}.

5. Convert N_m^{111} as a fixed point fraction to get N_1^f.

6. Convert the quantity (+ kr + q) as a binary fixed point integer. This gives $\bar{N}_1^e$ (use + if $|N_m| > 1$ and - if $|N_m| < 1$.)

2. BCD to Binary.

This is analogous to the procedure just described.

Other Comments

1. At the cost of increased complexity in fixed point conversions, a zero look-ahead feature could be used to enable one to skip bytes of N_m having all zero bits.

2. The floating point procedure can be readily adapted to conversions of the floating to fixed point type.

3. Means for automatically extracting successive bytes of a word would increase the speed of fixed point conversions and, moreover, would simplify their programming.

4. A facility for incrementing by some constant a register not related to an operation, while the operation is being executed, would eliminate the delay, in step three of the fixed point conversions, in which S is increased successively by 2^P (or $10^{P/4}$).

References

1. Editor and Translator PK EDIT, a paper by Dr. W. W. Peterson.

2. Manual of Operation, IBM Type 704. Appendix A.

3. 704 Manual. Section N, "Conversion".

Method	Same Method Used for Integ. & Fractions	Special Requirements	Relative Flexibility	Relative Speed	Relative Logical Complexity	Equipment Required
1	no	none	good	very slow	simple	Moderate controls binary & decimal adders.
2	yes	Powers of the radices are needed.	good	slow	simple	Moderate controls binary & decimal adders.
3	no	Fraction conversion not treated.	fair	very fast	complex	Elaborate controls binary & decimal adders.
4	no	Some fractions require awkward scaling.	poor	slow	simple	Moderate controls binary & decimal adders.
5	yes	Prepared tables & memory allocation.	good	very fast	simple	Moderate controls binary & decimal adders.

Appendix 1.

a. This table assumes n = 12 and p = 4. Notice that we must restrict $N_m < 10^3$. An analogous table would be used for fractions.

Binary to BCD (Integers)

Col. 1	Col. 2	Col. 3			Col. 4
Byte No.	Byte Array	Byte Images			Address where column 3 entry is stored
1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	S+0
	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 1	S+1
	0 0 1 0	0 0 0 0	0 0 0 0	0 0 1 0	S+2
	⋮	⋮	⋮	⋮	⋮
	1 1 1 1	0 0 0 0	0 0 0 1	0 1 0 1	S+15
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	(S+16)+0
	0 0 0 1	0 0 0 0	0 0 0 1	0 1 1 0	(S+16)+1
	0 0 1 0	0 0 0 0	0 0 1 1	0 0 1 0	(S+16)+2
	⋮	⋮	⋮	⋮	⋮ ⋮
	1 1 1 1	0 0 1 0	0 1 0 1	0 1 0 1	(S+16)+15
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	(S+32)+0
	0 0 0 1	0 0 1 0	0 1 0 1	0 1 1 0	(S+32)+1
	0 0 1 0	0 1 0 1	0 0 0 1	0 0 1 0	(S+32)+2
	0 0 1 1	0 1 1 1	0 1 1 0	1 0 0 0	(S+32)+3

b. Comments on "a" above apply. (But $N_m < 10^3$ not for the same reason)

BCD to Binary (Integers)

1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	R+0
	0 0 0 1	0 0 0 0	0 0 0 0	0 0 0 1	R+1
	0 0 1 0	0 0 0 0	0 0 0 0	0 0 1 0	R+2
	⋮	⋮	⋮	⋮	⋮
	1 0 0 1	0 0 0 0	0 0 0 0	1 0 0 1	R+9
2	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	(R+10)+0
	0 0 0 1	0 0 0 0	0 0 0 0	1 0 1 0	(R+10)+1
	0 0 1 0	0 0 0 0	0 0 0 1	0 1 0 0	(R+10)+2
	⋮	⋮	⋮	⋮	⋮
	1 0 0 1	0 0 0 0	0 1 0 1	1 0 1 0	(R+10)+9
3	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	(R+20)+0
	0 0 0 1	0 0 0 0	0 1 1 0	0 1 0 0	(R+20)+1
	0 0 1 0	0 0 0 0	1 1 0 0	1 0 0 0	(R+20)+2
	⋮	⋮	⋮	⋮	⋮
	1 0 0 1	0 0 1 1	1 0 0 0	0 1 0 0	(R+20)+9