Conversion of 42_{10} from binary (101010) to BCD.

David F. Brailsford

The Double Dabble technique

The algorithm starts with the appropriate number of BCD 4-bit nibbles on the left (initialised to zero) and the binary value on the right. The whole binary representation (two 4-bit nibbles and a 6-bit binary string in the present case, where 42_{10} i.e. 101010_2 is the value to be converted) is regarded as a single shift register. Whenever shift left operations take place these have the effect of multiplying the overall representation by 2 (hence the 'double' part of the name). If a prospective BCD digit is 5 or larger, then 3 is added before the next shift left. This is the 'dabble' part of the name — using the dictionary definition of this word as being: '... to make a small adjustment'.

<BCD NIBBLES>			
TENS	UNITS	BINARY	Operation
0000	0000	101010	Start
0000	0001	01010	Shift 1
0000	0010	1010	Shift 2
0000	0101	010	Shift 3
0000	1000	010	ADD-3 to UNITS
0001	0000	10	Shift 4
0010	0001	0	Shift 5
0100	0010		

The result is the BCD encoded value for 42.

Each of the 4-bit nibbles for holding the BCD digits could potentially hold a value of 15 (F in hexadecimal), with any further shifted-in bit, on the right, causing a carry on the left However, this must not be allowed to happen! The range of BCD-nibble values must be restricted to 0–9. In other words we need to cause each 4-bit nibble to do an 'early carry' at a value of 10, rather than at a value of 16.

The rationale for the ADD-3 rule is that whenever the shifted value is 10 or more, the weight of that shifted-out bit needs to be reduced from 16 to 10 — a difference of 6. To compensate for this we add 1/2 of that value, i.e. 3, before doubling the

answer, once again, by performing the next shift.

We detect that a 10-or-greater value will appear in a BCD-nibble, after shifting, by the fact that the value *before* shifting is 5 or greater. Notice that this rule needs to be applied to *all* BCD-nibbles within the bit string. Each nibble is regarded as a localised, unweighted, 0-9 entity when applying the double-dabble rule and this ensures that the various BCD-digits can never hold any non-BCD values.

Further Processing

The double-dabble technique terminates when the final binary digit of the input has been shifted left into the BCD part of the overall shift register. The shift operations are easiest if the entire bit pattern (14 bits in the case where 42 is being converted) is regarded as one unified shift register. However, this leaves the problem that the various BCD nibbles in the result have to be extracted out of a long bit string by the use of shift and mask operations.

The article in reference [1], below, discusses extensions of the double-dabble technique that enable least significant BCD digits to be delivered, in turn, by a shift-based method, which implements division by 10.

References

[1] Charles B. "Chuck" Falconer,"An Explanation of the Double-Dabble Bin-BCD Conversion Algorithm"
https://web.archive.org/web/20090325002523/http://cbfalconer.home.att.net/download/dubldabl.txt

[2] See also, Wikipedia articles on "Binary coded decimal" and "Double Dabble"