Calculator chart for hex displacements

Probably the biggest chore when you are programming a microcomputer in hexadecimal code is calculating relative address displacements. Here is an improved displacement lookup table which the author claims is easier and more rapidly used than any method yet published.

by COLIN KEAY

Associate Professor of Physics, The University of Newcastle NSW 2308

Unless you happen to be proficient at subtraction in hexadecimal arithmetic, the determination of relative address displacements is the most irksome task faced by microcomputer programmers who attempt to assemble programs manually. Even when using a microcomputer with a built-in displacement calculator routine the required button-pushing takes time, especially if you forget or mistake the addresses in memory where the origin and destination bytes must be lodged for processing. Furthermore each microcomputer with this facility has a different procedure to follow.

Alternatively, one may work from sequential hexadecimal number tables which require tedious counting, buy a Texas Instruments "Programmer" calculator or use a Relative Jump Ruler as described by John S. MacDougall (Interface Age, March 1978). This requires that the program be coded on a line-per-byte basis, which is a considerable nuisance for the majority of microcomputers.

There had to be a better way, and it was found by analogy with the unique way the old IBM 1620 computer performed its decimal arithmetic with binary coded numbers. This is the reverse problem: getting decimal-oriented humans to work in hexadecimal. The solution is a simple but universal look-up table for finding hexadecimal differences one digit at a time, with provision for borrowing when necessary between digits. The resulting Microcomputer Program Displacement Table shown in Fig. 1 has been used to assist the manual assembly of programs for a variety of different microprocessors and has proved very successful as a programming teaching aid.

Using the Displacement Table is almost self-explanatory. A few examples will make the procedure clear. First, a program branch back from A7 to 54 as demonstrated in Fig. 2. Take the low order digit first, descending column 7 to the row containing the destination digit 4 which also contains the digit D in the column under X1. Similarly, descend from A to 5 and in the same row as 5 the digit A is found in the column under X2. The required displacement byte is therefore AD.

Had the first destination digit been located within the dotted triangle the second digit of the displacement byte would have been obtained from the column under X1. This takes into account the absence of a "borrow" from the subtraction of the first digit. So for our second example consider a branch from A7 to 8D as shown in Fig. 3. Descending from 7 we find D is within the dotted triangle. In the row containing D we find the digit 6 under X1, but for the next digit we must this time find it again under X1 and not X2.

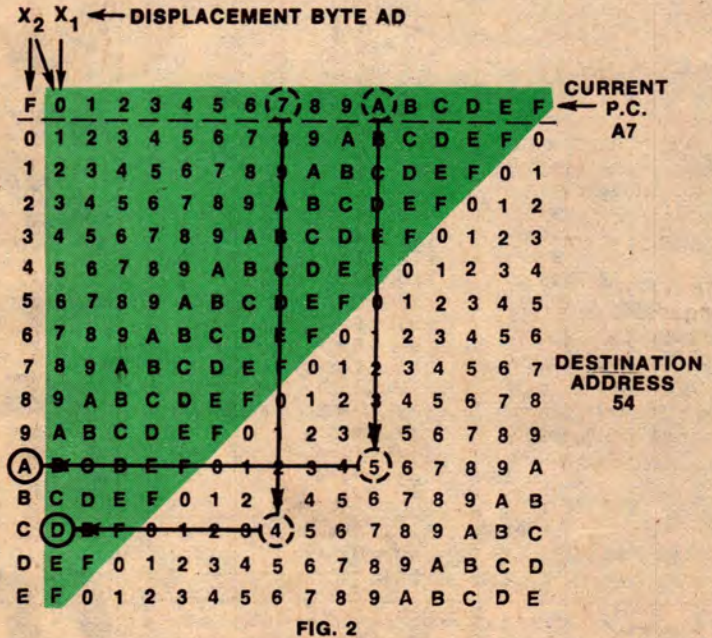


FIG. 2

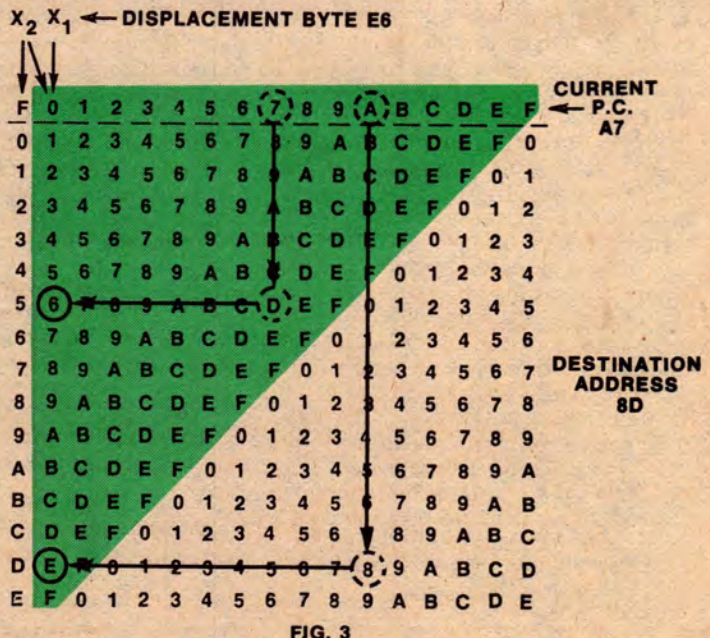
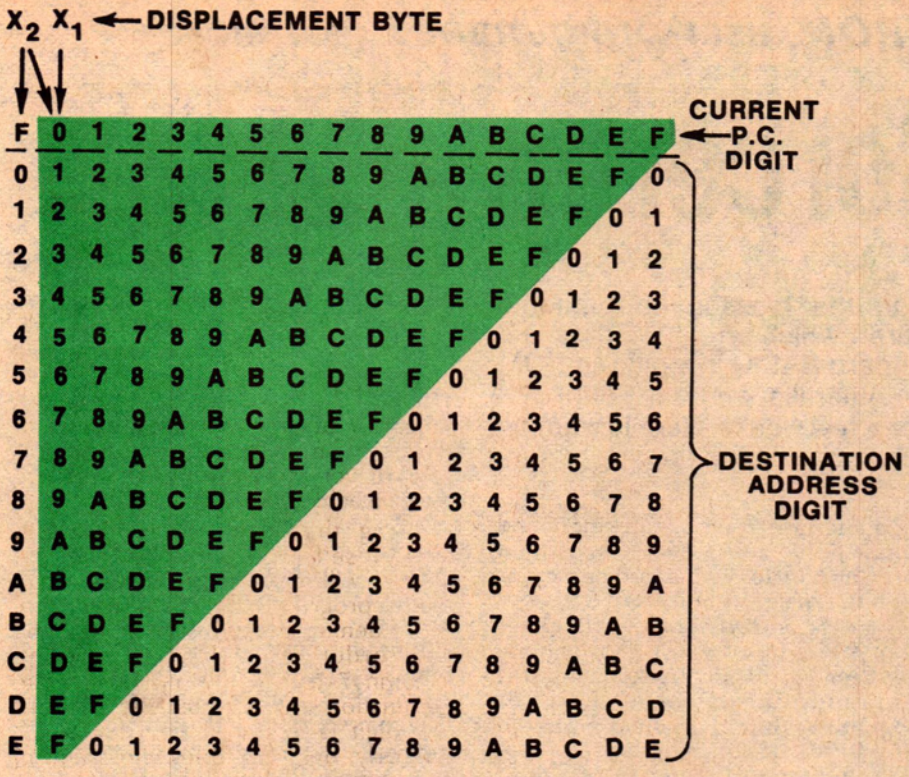


FIG. 3

These two examples show how the displacement table is used. The example in Fig. 2 shows a jump from A7 to 54, while that in Fig. 3 shows a jump from A7 to 8D.



F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1
2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2
3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3
4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4
5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5
6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6
7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8
9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A
B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B
C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C
D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D
E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E

RULE: IF FIRST DESTINATION IS WITHIN SHADED TRIANGLE X2 WILL BE ALSO

FIG. 1 MICROCOMPUTER PROGRAM DISPLACEMENT TABLE

Here is Professor Keay's table, inspired by the computational method used by an early computer. Using it you can find relative address and branch displacements very rapidly and conveniently.

Descending from A to 8 we find in the row containing 8 the digit E under X1. The required displacement byte is therefore E6.

We have assumed that the high-order byte is identical for the current address and the destination. Whenever the high order bytes differ, a single byte may be insufficient for the displacement. In such a case the Displacement Table can be employed to obtain X3 and X4 in exactly the same way as for X2. One must simply obey the rule that whenever a destination digit falls within the dotted triangle the subsequent displacement digit must also be within the dotted triangle. For this reason the dotted region of the table has been nicknamed "The Bermuda Triangle" — when landing in it one stays in it, but only for the next turn!

Besides calculating program displacements the table is useful for determining the length of program segments, subroutines, etc. For example, can a program extending from 27B9 to 3C63 be relocated in a gap between 0FA2 and 1455? Using the table we find that the program is 4AA bytes in length and the gap is found to be of 4B3 bytes, so it will fit with 4B3-4AA = 19 (hex) bytes to spare.

This technique uses hexadecimal numbers exclusively and thereby avoids the confusion inherent in the Address Calculator Tables (such as Ray Boaz's table in Byte Magazine, April 1978) and Multi-base Conversion Charts (wall-size or otherwise) where the actual arithmetic is performed in decimal notation.

Finally, when time permits, it is a good idea to check the results obtained using the Displacement Table by attempting to perform an actual hexadecimal subtraction and if necessary, counting digits, in hexadecimal to verify the result. After a few weeks (or months?) of practice the terrors of hexadecimal arithmetic will vanish and the Displacement Table can then be thrown away — even when calculating fearsome reverse jump displacements!

NOTE: The foregoing article is reproduced by arrangement with the author and copyright holder, C.S.L. Keay. Reproduction by educational bodies for tuitional purposes is permitted, providing due acknowledgement is given. Reproduction for any other purpose by negotiation only.

On sale now
"ELECTRONICS AUSTRALIA" PROJECTS & CIRCUITS NO. 2
$3.00 plus 60c p & p
Electronics Australia,
Box 163, Beaconsfield, NSW 2014