Forth Extensions

A Note on Bitmaps

Contributed by Howard Goodell, Jr., Sr. Systems Analyst, Periphonics, Bohemia, NY.

Bit maps are an invaluable tool for handling 2-valued data such as disk free block lists and compressed database indices. They reduce index storage space by a factor of 8 over bytes, and can reduce the iterations for searching the indices by a factor of the word length of the computer. For example, an index of free blocks for a 320-block floppy needs just 40 words and can be searched in 20 iterations on a 16-bit computer; the index for a 5 megabyte hard disk occupies 625 bytes and takes 313 iterations. The routines presented minimize the number of multiplications, divisions, and shifts (or for you BASIC fans, exponentiations) required to manipulate bitmaps by the use of 2 table lookups.

Block 1317 contains the set of bit manipulation routines used for block allocation in the SQUIRT database, which is based on work presented at the 1982 Rochester Conference [GOO82]. The basic word is BIT , which converts an address and an offset in bits to the actual byte address and mask that are used to SET , RUB (clear) , or TEST it. The mask is generated by an 8-byte lookup table (the byte ordering shown is suitable for byte-swapped families such as the PDP-11 and 8080: non-byte-swapped computer such as the TI 990 replace line 2 with the alternate on line 15). -BIT is the basic bitmap searcher; it searches by byte until it finds 1 nonzero; then it calls ABIT to find the first bit set in this byte. The word &OR ORs 2 multi-byte data structures together; e.g., all the blocks assigned to a file can be removed by &ORing its block list with the free space bitmap.

Block 1318 contains a routine that counts the bits set up to and including the parameter given. It does this via another table lookup; this time the byte indexed by each nybble (4 bits) of the offset contains the number of bits in that nybble; for example there are 2 bits set in 3, 5, and 9; so all these numbers index to a 2. Bit counting is a possible strategy for sparse matrix expansion; for each row, a bitmap indicates the occupied columns, and the values for the occupied columns follow the bitmap. After a quick TEST operation verifies that a column is occupied, the bits to the left of it are counted and the address of the data is calculated. The routine shown has 4 times less iterations than scanning for the column number; the speed of the counting could be doubled by using a 256-byte lookup table that found the number of bits in a byte rather than a nybble. An added advantage of compressing matrices this way is that sometimes one only cares whether a value is nonzero; so only the TEST is required.

Block 1319 contains the bitmap equivalent of DUMP , BITS . Note that it takes an arbitrary first and last bit after the starting address. The number of bits on a line should be set to whatever value is most useful for the application; for instance if one cares about word alignment it would be better to use 64 bits per line.

Character operators and a factor of 8 have been used in this example because the result is transportable (except for switching byte ordering in the 2 lookup tables) and especially simple. Speed may in general be increased at the cost of a little space by expanding the lookup tables and operations to a full word.

References

[GOO82] H. Goodell, "Elements of a Forth Relational Database", *1982 Rochester Forth Conference on Data Bases and Process Control*, Institute for Applied Forth Research, 1982.

```
1317 LIST

 0 ( B+      bit mapping )
 1 CREATE MASKS   HEX
 2 0201 , 0804 , 2010 , 8040 , DECIMAL ( lsb,msb )
 3 : BIT ( base address, offset in #bits - byte address, mask )
 4       8 /MOD ROT + SWAP MASKS + C@ ;
 5 : SET ( a,n )  BIT OVER C@ OR SWAP C! ;
 6 : RUB ( a,n )  BIT 255 XOR OVER C@ AND SWAP C! ;
 7 : TEST ( a,n )  BIT SWAP C@ AND ;
 8 : ABIT ( a - n )  8 0 DO DUP I TEST IF
 9       DROP I LEAVE THEN LOOP ;
10 : -BIT ( a,n - #bit found,0 / true )  8 / 0 DO DUP I + C@ IF
11       I + ABIT I 8 * + 0 LEAVE THEN LOOP ;
12 : &OR ( source, destination, #bits )  0 DO OVER I + C@
13       OVER I + DUP C@ ROT OR SWAP C! LOOP 2DROP ;
14 1318 1319 THRU
15 EXIT 0102 , 0408 , 1020 , 4080 , DECIMAL ( msb, lsb )

1318 LIST

 0 ( bit counting )
 1 CREATE BUCKET   HEX
 2 ( 1 0  3 2  5 4  7 6  9 8  B A  D C  F E lsb, msb )
 3 0100 , 0201 , 0201 , 0302 , 0201 , 0302 , 0302 , 0403 ,
 4 DECIMAL
 5 : #B ( n - #bits )  16 /MOD BUCKET + C@ SWAP BUCKET + C@ + ;
 6
 7 : #BITS ( base address , # to search - #found )  1+ SWAP R
 8       8 /MOD DUP DUP IF 0 SWAP 0 DO
 9       J I + C@ #B + LOOP THEN SWAP R + R SWAP ?DUP IF
10       0 DO J I TEST 0= NOT + LOOP THEN R DROP ;
11
12
13 EXIT
14 ( 0 1  2 3  4 5  6 7  8 9  A B  C D  E F msb, lsb
15  0001 , 0102 , 0102 , 0203 , 0102 , 0203 , 0203 , 0304 , )
```

```
1319 LIST

 0 ( Bit display )
 1
 2 : BITS ( base address , first bit , last bit )  CR 6 SPACES
 3       50 0 DO OVER I + 10 MOD 1 U.R
 4       LOOP OVER - 0 DO I 50 MOD NOT IF
 5       CR DUP I + 5 U.R SPACE THEN
 6       2DUP I + TEST IF ." X" ELSE SPACE THEN LOOP 2DROP ;
 7
 8
 9
10
11
12
13
14
15
```