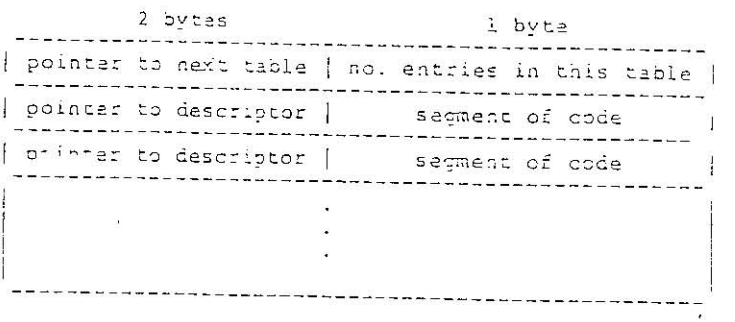
1. Extension Statements

BASIC keeps in page 0 a linked chain of pointers to keyword descriptors, so that extra keywords can be linked in as required by linking a new table of pointers to the previous table in the chain. It is also possible to change the pointers in the default table to point to new descriptors in order to change or enhance the action of the default statements.

The tables are set out as follows:



A pointer value of 0 for the pointer to the next table in the chain indicates the end of the chain. A segment number of 0 indicates that the keyword code is in page 0.

The descriptor itself (described below) must always be in page 0, although the actual code can be in another segment. The only exception to this is BASIC's default keywords, which are in page 3 but are known by BASIC to be paged in anyway when the table is accessed.

The actual token values for the keywords (see section 2.2 (Program) for more details) are obtained from the position of the pointer to the descriptor in the chain of tables. The first pointer (which is to one of BASICs default keywords) has the lowest token value (0), the token values incrementing for each pointer.

Copyright (C) 1985 Intelligent Software Limited

It is recommended that new keyword tables link themselves in by changing the pointer to the next table in the default table to point to the new table, and then setting the pointer in the new table to whatever value was contained in the pointer in the default table. Thus new keywords should not assume that they have a fixed token value, but should calculate it by following the list of keyword tables.

It should be noted that many of BASICs block-orientated keywords do not do this, but assume that they are at a fixed position within the keyword table list. Thus it not wise to cause the default keyword table to be other than first in the list. This can be avoided by following the above method.

The keyword descriptors which are pointed to by the above pointers (and are always in page 0) take the following form:

2 bytes	2 bytes	1 byte	n bytes
execution address	tokenising address	flags	name text

The execution address is a page 3 pointer to the address which BASIC calls when the keyword is executed. The routine must be in the segment indicated by the segment byte of the keyword pointer table described above.

he tokenising address is a pointer (following the same conventions as the execution address pointer above) which points to a routine which BASIC calls when the line is first typed in, but after the line has beem tokenised.

This will not often be used by extension statements, but may be useful for checking the syntax of a line before it is run, for avoiding the confusion that can occur over colons used in statements and colons used for multi-statement line separators, and for indicating that a number in a statement is in fact a line number. The latter is used to ensure that RENUMBER will renumber it correctly. See section 2.2 (Program) for details of the internal representation of a line number.

Copyright (C) 1985 Intelligent Software Limited

For example, suppose that an extension keyword's syntax included a line number immediately after the keyword itself (eg. GOTO n, GOSUB m, RESTORE p). The tokenisation routine would first call the system routine GETITEM, so that the pointer SYMB pointed to the data of the line number just got by GETITEM. The item descriptor byte at SYMB-1 is then changed from being C2 hex. (integer number) to A2 hex. (line number). The tokenisation routine would then return. Subsequent uses of RENUMBER will then renumber the line number.

The actual code for the above would be:

```
RST     10H         ; Call GETITEM.
DEFB    20H         ; (code for GETITEM)
DEFB    0
LD      HL,(SYMB)   ; Point to number.
DEC     HL          ; Point to descriptor.
LD      A,(HL)      ; Get descriptor byte for error
LD      (HL),0A2H   ; check, and replace with new.
CP      0C2H        ; Check that number was given.
RET     Z           ; Return if OK.
LD      HL,20001    ; Else give an error ('Invalid
RST     20H         ;   line number').
```

The flags byte contains single bit flags describing the keyword and under what context it may be used. The bit within the byte have the following meanings:

bit 0 - Set to indicate that the keyword may be used in immediate mode. Keywords such as DEF are not allowed in immediate mode.

bit 1 - Set to indicate that the keyword may be used in a program. Keywords such as CONTINUE are not allowed in a program.

bit 2 - Set to indicate the end of a block. Causes the indentation byte to be set to one less than that of the previous line. See section 2.2 (Program). Can be used so that the statement at the start of the block can find and verify the correct end of block.

bit 3 - Set to indicate the start of a block. Causes the indentation byte of the next line to be set to one more than that of the previous (and current) line. Can be used to match with the end of block.

bit 4 - Set to indicate that the keyword is allowed on a multi-statement line and after a THEN statement.

Copyright (C) 1985 Intelligent Software Limited

bit 5 - Set to cause the symbol table and stack to be cleared before execution in immediate mode. Unlikely to be used by an extension.

bit 6 - Set to indicate that the remainder of the statement after the keyword must be tokenised ie. the usual condition. DATA statements and comments to not require the rest of the line to be tokenised, so do not set this bit.

If both the start of block and end of block bits are set, then BASIC will assume that the keyword is a 'temporary' end of block, and will reduce the indentation count for that line only. This is used for example in the CASE statement.

The name text part of the keyword descriptor contains the name of the keyword in upper case ASCII, preceded by a byte containing the number of characters in the name.

Note that when the keyword execution routine is called, the GETITEM routine (see section 6 (System Calls) and section 2 (Program)) will already have been called to get the item following the keyword in the program line. This is not the case, however, for the tokenising routine.

 Copyright (C) 1985 Intelligent Software Limited