7 Adding New Commands

When the BASIC interpreter discovers anything which it does' t recognise, it generates an error (usually 'Mistake'), to stop processing of the program or command and go back to command mode. This section describes how new statements and commands can be added to BASIC by intercepting this error.

7.1 Trapping BRK

The method that BASIC uses to generate an error, is to execute a BRK instruction, which is followed by a number of bytes in a standard error format. This format is:

BRK instruction to generate the error
Single byte error number (ERR)
Error message (like 'Mistake')
A zero byte to terminate the message

This is the standard method of generating errors on the Acorn BBC system, and it allows errors to be 'trapped' by intercepting the BRK vector (at &202). By trapping the errors generated by BASIC, it is possible to add new commands, overlay procedures, etc., and continue where it left off. Other errors which are generated by BASIC are described in chapter 11.

When a BRK instruction is executed, the Machine Operating System will JMP to the BRK handler whose address is in the BRK vector at &202,&203. On entry to the BRK handler the following conditions prevail:

(a) The A, X and Y registers are unchanged from when the BRK instruction was executed.

(b) The 6502 stack is prepared ready for an RTI to the instruction following the BRK instruction (i.e. with the 6502 flag byte on the top of the stack, and the return address underneath it). This will return control to the instruction 2 bytes after the BRK instruction.

(c) The pointer in locations &FD,&FE points to the 'error number' byte after the BRK instruction.

Although a return from a BRK intruction is possible (it can be used as a breakpoint in a machine code program), BASIC does not expect such a return; executing an RTI after a BRK instruction has been executed by BASIC (or any other program using it as an error generating mechanism) will probably have fatal results.

The small program below illustrates how the BRK vector can be intercepted, to cause a bleep (CHR$7) each time an error is generated. If you get fed up with this, pressing BREAK or typing BASIC' will re-set the BRK vector to point to the default BRK handler in BASIC, missing out this routine.

The code assembles into the user defined character area from &()C00 onwards. If any user defined characters are to be used while the routine is 'linked in' to the BRK vector, it could be assembled somewhere else, by changing line 900. Space could be allocated at PAGE for it by adding 256 to PAGE before the routine is loaded (or typed in), and assembling the code to the old location of PAGE, underneath the BASIC program.

```
  10 REM Routine to print a bleep on an error
  20 REM
 400 brkv   = &0202              :REM BRK vector location
 410 oldbrk = !brkv AND &FFFF    :REM Get default BRK handler
 420
 500 oswrch = &FFEE              :REM OSWRCH (to print bteep)
 505
 900 start% = &0C00              :REM User char area
 905
 910 FOR opt% = 0 TO 3 STEP 3
 915 P%=start%
 920 [OPT opt%
 925
1000 .newbrk
1005          PHA                \ Save A
1007
1010          LDA #&7            \ Print a bteep
1015          JSR oswrch
1017
1020          PLA                \ Retrieve A, and continue
1025          JMP oldbrk         \  with default BRK handter.
9000 ]
9010 NEXT
9020 IF newbrk=oldbrk PRINT"Already set up":END
9030 brkv?0 = newbrk MOD &100 :REM Set up BRK vector to
9040 brkv?1 = newbrk DIV &100 :REM point to this routine.
9050 END
```

When the program is assembled, the address of the default BRK handler is retrieved at line 410. This is where the new routine will JMP to when it has printed its bleep. This means that the error message will still be printed by the BASIC BRK handler, as though nothing had happened.

After the program has been assembled, its start address is poked into the BRK vector at lines 9030 and 9040 (the BRK vector is stored low byte first). Line 9020 checks to see if the program has already been set up. If it has, the new BRK handler would jump back to itself when it has finished. This means that if any error occurs, it will continue printing bleeps until BREAK is pressed − not very useful (try assembling it twice, and see what happens). This is something to look out for with most error trapping routines; if they fail to clear the error which called them, it will be generated again, and they will be called again in exactly the same situation.

The error trap routine saves A by pushing it on the stack, while it prints the bleep. This is not necessary if the BASIC error handler will be JMPed to immediately afterwards, as it does not use it; but it would be important if a different routine, which relies on A being correct on entry, had intercepted the BRK vector before this program was entered. If this other routine had been linked in to the BRK vector in a similar way, the 'JMP oldbrk' on the end of this routine will jump into that routine when it is finished, rather than the BASIC BRK handler.

It is usually a good idea to save any registers you are going to use, if control will be returned to another routine which may need them. If the 'No room' error is being trapped, for example (chapter 11, BASIC2 only), all of the 6502 registers (A, X, Y) must be intact so that the source of the error can be determined.

7.2 The 'Mistake' error

If you type in a word that BASIC doesn't recognise, it generates a 'Mistake' error (error number 4). However, it leaves its statement pointer, PTRA, pointing one character after the start of the name (PTRA was advanced one byte by the action of reading in the first character). This means that the word which caused the error to be generated can be checked, and action taken if it corresponds to a new, 'home-made' statement.

The ‘Mistake’ error is actually generated when BASIC fails to find an ‘=’ character, often due to a mistyped keyword (such as ‘PRIT’ instead of ‘PRINT’). When this happens, the sequence of actions is as follows:

1 The statement interpreter reads the character at PTRA, advancing PTRA to point to the next character.

2 The character is not a keyword token. It is alphabetic, however, so it looks like the start of a variable name; and the statement interpreter jumps into the variable assignment handler.

3 The assignment handler scans what it thinks is a variable name, using PTRB. This means that PTRA still points one byte after the first character of the name. If the name is of a variable which doesn’t already exist, it will create it; but only after it has checked that there is an ‘=’ following it.

4 The assignment routine checks for an ‘=’ after the variable name. If it doesn’t find one (which it won’t, if it was a mistyped keyword), it generates a ‘Mistake’ error. If it does find one, it continues with the assignment.

In fact there are 5 slightly different causes of a ‘Mistake’:

(a) A non-existent variable name was found, without an ‘=’ following it. This error is generated before the variable is created, by a sort of ‘pre-check’ before the main assignment routine is entered.

(b) An existing variable name was found, without an ‘=’ following it. This is not quite the same as (a), above, but the only difference is the return address left on the 6502 stack.

(c) A ‘LET’ statement, followed by a valid variable, was found, but there was no ‘=’ following the name. If the variable did not exist before this statement, it would have been created before the error was generated (unlike (a) above).

(d) A psuedo-variable name, like ‘HIMEM’, was found, but no ‘=’ followed it.

(e) A ‘FOR’ statement was found, followed by a valid variable, but no ‘=’ followed the name.

All of these leave PTRA pointing 1 byte after the start of the statement, but (c), (d), and (e) leave the 6502 stack in different states. Fortunately, this only happens if the first character of the statement is a keyword token; so if new statements are to be introduced, they should not be allowed to start with one of the tokens mentioned above (so ‘FORAGE’ cannot be a new statement keyword).

Note that new keywords cannot begin with any other tokens either (like the ‘TO’ in ‘TOTAL’) as these will cause a ‘Syntax error’ rather than a ‘Mistake’. However, some of the BASIC keywords are not tokenised if followed by an alphanumeric character (see section 2.3.1), so ‘TIMER’ could be used as a new statement (the ‘TIME’ part would not be tokenised).

For (a) and (b), the prevailing conditions on entry to the BRK handler are:

&FD,&FE	points to the error number (4)	
Stack contents:	RTI information Return	3 bytes
	Return address	2 bytes
PTRA	points 1 after the first byte of the name	

Other conditions are not so important (see chapter 11, error number 4).

When a new statement has been recognised, the 3 bytes of RTI information (pushed by the BRK instruction) and the 2 bytes of return address (the ‘=’ was checked by a subroutine called by the assignment handler) must be pulled from the stack before execution is continued. If this is not done, any FNs or PROCs will not return properly, as they expect their return address to be on the top of the stack (see section 5.3).

7.3 A single character statement

The routine in this section shows a simple example of adding a ew statement, by just checking the first character of the tatement; the one just before PTRA. If it is a 'B' ,it pulls the 5 bytes to be discarded from the stack, checks that the 'B' is the only thing (apart from spaces) in the statement, and produces a bleep. Finally, it JMPs to the BASIC entry point to continue executing the following statements.

Instead of being initialised when the program is assembled, this program links in to the BRK vector when the small routine at the start is CALLed (lines 1000 to 1115). Any programs which are initialised in this way don't need to be reassembled each time they are used.

Note that the EQUB and EQUS assembler directives are used in this program (lines 1025 to 1040), as they are much clearer than the equivalent in BASIC. However, the EQU directive is not implemented in BASIC 1, and should be replaced with its equivalent using indirection operators.

```
10 REM *** Program to add single character command ***
12 REM
14 REM         M D Ptumbtey       1984
16 REM
18 REM This program traps the BRK vector. On an error,
20 REM  if ERR (the error number) is 4 ("Mistake")
22 REM  and the unrecognised statement is the singte
24 REM  character "B", then a bteep will be produced.
26 REM
28 REM If the error number is not 4, or the first char
30 REM  of the statement is not a "B" , then controll will
32 REM  be passed to the defautt error handl-er.
34 REM
36 REM When setting up, the program tests for BASIC 1
38 REM  or BASIC 2, and uses the corresponding ROM
40 REM  entry points.
42 REM
44 REM Before using on BASIC I, all EQU directives
46 REM  should be reptaced with indirections:
48 REM  "EQUB X"  => "]?P%=X:P%=P%+1:[OPTopt%"
50 REM  "EQUS A$" => "]$P%=A$:P%=P%+LEN$P%:[OPTopt%"
52 REM
54 REM The code is assembled into the user defined
56 REM  character space: alternatively, space coutd
58 REM  be reserved at PAGE for it.
```

```
  60 REM
  99
 100 PROCsetup :REM Set up correct ROM entry points
 490
 495 REM *** OS routines and vectors ***
 500 OSWRCH = &FFEE
 550 BRKV   = &0202
 799
 900     start% = &0C00 :REM Assemble into user char space
 905
 910     FOR opt% = 0 TO 3 STEP 3
 920     P% = start%
 950     [OPT opt%
1000 .init
1005     LDA &8015              \Test that the correct
1010     CMP #baschr            \ version of BASIC is
1015     BEQ basok              \ in the ROM.
1016
1020     BRK                    \If it isn't, print an
1025     EQUB 60                \ error message.
1030     EQUS "Not BASIC "      \ (baschr set by PROCsetup)
1035     EQUB baschr
1040     EQUB 0
1041
1045 .basok
1050     LDA BRKV               \Load the current BRK vector
1055     LDX BRKV+1             \ into A and X.
1056
1060     CMP #newbrk MOD &100   \If this routine is already
1065     BNE ntsavd             \ set up, don't change BRKV.
1070     CPX #newbrk DIV &100
1075     BEQ saved
1076
1078 .ntsavd
1080     STA svbrkv             \It has not been set up
1085     STX svbrkv+1           \ already, so save old
1090     LDA #newbrk MOD &100   \ BRKV, and set up the neu
1095     STA BRKV               \ one.
1100     LDA #newbrk DIV &100
1105     STA BRKV+1
1106
1110 .saved
1115     RTS
1190
1192 \ *** This is the new BRK handting routine ***
1200 .newbrk
1205      PHA           \Save A and Y on 6502 stack
1210      TYA
1215      PHA
1216
1220      LDY #0        \Get error number
1225      LDA (&FD),Y
```

```
1226
1280      CMP #4        \If "Mistake", check for a "B"
1285      BEQ mistak
1286
1400 .giveup
1410      PLA           \Restore A and Y from 6502 stack
1420      TAY
1430      PLA
1431
1440      JMP (svbrkv)  \Go to old BRK handl-er
1441
1490 \ *** If we get here, an error 4 ("Mistake") has  ***
1492 \ *** ocurred, so see if the charcter is a "B".   ***
1500 .mistak
1510      LDY &A        \Get character at start of statement
1520      DEY
1530      LDA (&B),Y
1531
1540      CMP #ASC"B"   \If it is not a "B" , go to the old
1550      BNE giveup    \ BRK handler
1551
1560      PLA           \Discard saved A and Y from stack
1570      PLA
1571
1580      PLA           \Discard RTI information from the
1590      PLA           \ 6502 stack. This is automaticatty
1600      PLA           \ pushed by the BRK instruction.
1601
1610      PLA           \Discard return addr (of routine
1620      PLA           \ to check for "=") from stack
1621
1630      JSR chksda    \Check for end of statement
1631
1640      LDA #7        \Print a beep
1650      JSR OSWRCH    \ (action at last!)
1651
1660      JMP cont      \Continue execution
1661
6899
6990 \ ***          Routine variabtes area         ***
6991
7000 .svbrkv EQUW !BRKV \Space to save old BRK vector
7010
8000 ]
8010 NEXT
8015 @%=0
8020 PRINT'"Code length =&"~P%-start%
8190
8200 PRINT'''''"** WARNING: Once assembled, the code"
8210 PRINT"generated by this program is not"
8220 PRINT"transferable between different BASICS"
8230 PRINT
```

```
8300 PRINT"Execute ""CALL &"~init""" to initialise."
8310 END
8990
8992 REM *** Set up ROM entry points, allowing for ***
8993 REM ***  BASIC 1 and BASIC 2.                  ***
9000 DEFPROCsetup
9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
9030 IF $&8009=basic1$ THEN PROCset1 :ENDPROC
9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
9050 PRINT "NOT BASIC I OR II"
9060 END
9290
9292 REM *** Set up BASIC 1 entry points ***
9300 DEFPROCset1
9310 baschr = ASC"1":REM Used by init routine
9320 chksda = &9810 :REM Check for statement delimiter
9330 cont   = &8B0C :REM Cont execution at next statement
9490
9492 REM *** Set up BASIC 2 entry points ***
9500 DEFPROCset2
9505 baschr = ASC"2":REM Used by init routine
9530 chksda = &9857 :REM Check for statement delimiter
9540 cont   = &8B9B :REM Cont execution at next statement
9550 ENDPROC
```

The general operation of the program is as follows:

PROCsetup is called to set up the correct ROM entry points required by the routine ('Check for statement delimiter' and 'Continue execution' in this case). This uses the copyright string to check for the version type, and calls PROCset1 or PROCset2 depending on the year (1981 or 1982). Alternatively, the paged ROM version number, held in location &8008, could be used. This is &00 for BASIC1, and &01 for BASIC2.

When the assembled code is initialised by CALLing the start, the initialisation routine first checks that the year of the ROM is the same as the one it was assembled for; if it isn't, it won't link itself in (as the ROM entry points will be wrong). Note that this check will only work if the BASIC ROM is paged in when the initialisation routine checks the year; and not if the DFS, say, is paged in (if the routine has just been RUN'). See chapter 10 for more on this.

If the ROM is correct, the initialisation routine saves the contents of the BRK vector at 'svbrkv', and sets the BRK vector to point to the new BRK handling routine.

When an error is generated, and 'newbrk' is entered, it checks that the error number pointed to by &FD ,&FE is 4, if it isn't, the or was not a 'Mistake', and a JMP is made to the default BRK handler to deal with it.

If the error is a 'Mistake', the character before PTRA is tested to see if it is a 'B' (the base of PTRA is stored in &B,&C with the offset in &A). If it isn't the old BRK handler is JMPed to to print the 'Mistake' message.

If it is a 'B', then the 5 bytes on the 6502 stack are pulled from it (together with the 2 saved registers from the BRK handler). Then the ROM routine is called which checks for the end of the statement at PTRA (which still points just after the 'B'). This will produce a 'Syntax error' (error number 16) if it doesn't find a ':', an ELSE token, or the end of the line.

Finally, a bleep is printed, and a JMP is made to the ROM routine which continues with the execution of the program. Note that this routine expects the 'Check for statement delimiter' routine to be called before it, so that PTRA is set up to actually point 1 byte after the statement terminator. These ROM routines are detailed in chapter 10.

7.4 Recognising keywords

Just using single character statements is not very versatile: most of the time it would be much more useful to give the new statements keywords which reflect the action that they perform, like 'DUMP' to dump the variables, or 'REN' to renumber a program. The program in this section shows how to implement a command line interpreter to recognise keywords from a table.

The keywords implemented in the program are 'BEEP', which beeps (again), and 'DUMP', which lists the current active dynamic variables (see section 3.1.2). Neither of them take any arguments.

Note that the EQU assembler directive has been used again (lines 1025 to 1040 as before, and lines 2500 to 2580 in the keyword table).

```
  10 REM *** Program to add new BASIC commands ***
  12 REM
  14 REM     M D Plumbley     1984
  16 REM
  18 REM This program traps the BRK vector. On an error,
  20 REM  if ERR (the error number) is 4 ("Mistake")
  22 REM  then a command line interpreter with test the
  24 REM  statement for a keywrd to recognise. If it is
  26 REM  recognised, the keyword's action is performed.
  28 REM  Otherwise, controt is passed on to the default
  30 REM  BRK handler.
  32 REM
  34 REM The code is assembled into the user key/char
  36 REM  space: alternatively, space could be reserved
  38 REM  at PAGE for it.
  40 REM
  42 REM Before using with BASIC 1, the EQUs should be
  44 REM reptaced with their equivatent:
  46 REM "EQUB X"  => "]?P%=X:P%=P%+1:[OPTopt%"
  48 REM "EQUW X"  => "]!P%=X:P%=P%+2:[OPTopt%"
  50 REM "EQUS A$" => "]$P%=A$:P%=P%+LEN$P%:[OPTopt%"
  52 REM
  99
 100 PROCsetup :REM Set up correct ROM entry points
 490
 495 REM *** OS routines and vectors ***
 500 OSWRCH = &FFEE
 550 BRKV   = &0202
 590
 600 svbrkv = &0070 :REM Space to save old BRK vector
 690
 900 start% = &0B00 :REM User keylchar area
 905
 910 FOR opt% = 0 TO 3 STEP 3
 920 P% = start%
 950 [OPT opt%
1000 .init
1005     LDA &8015             \Test that the correct
1010     CMP #baschr           \ version of BASIC is
1015     BEQ basok             \ in the ROM.
1016
1020     BRK                   \If it isn't, print an
1025     EQUB 60               \ error message.
1030     EQUS "Not BASIC "     \ (baschr set by PROCsetup)
1035     EQUB baschr
1040     EQUB 0
1041
1045 .basok
1050     LDA BRKV              \Load the current BRK vector
1055     LDX BRKV+1            \ into A and X.
1056
1060     CMP #newbrk MOD &100 \If this routine is already
```

```
1065     BNE ntsavd           \ set up, don't change BRKV.
1070     CPX #newbrk DIV &100
1075     BEQ saved
1076
1078 .ntsavd
1080     STA svbrkv           \It has not been set up
1085     STX svbrkv+1         \ atready, so save old
1090     LDA #newbrk MOD &100 \ BRKV, and set up the new
1095     STA BRKV             \ one.
1100     LDA #newbrk DIV &100
1105     STA BRKV+1
1106
1110 .saved
1115     RTS
1190
1192     \ *** This is the new BRK handting routine ***
1200 .newbrk
1205      PHA           \Save A and Y on 6502 stack
1210      TYA
1215      PHA
1216
1220      LDY #0        \Get error number
1225      LDA (&FD),Y
1226
1280      CMP #4        \If "Mistake", try new keytwsrds
1285      BEQ mistak
1286
1400 .giveup
1410      PLA           \Restore A and Y from 6502 stack
1420      TAY 1430 PLA
1431
1440      JMP (svbrkv)  \Go to old BRK handter
1441
1490 \ *** If we get here, an error 4 ("Mistake") has  ***
1492 \ *** ocurred, so attempt to recognise one of the ***
1494 \ *** command keywords in the table.              ***
1500 .mistak
1510      LDA #keytab MOD &100\Get start of keytusrd tabte
1520      STA &39       \ into (&39)
1530      LDA #keytab DIV &100
1540      STA &3A
1541
1550      LDY &A        \Set (&37) to point to character
1560      DEY           \ before PTRA. It will then point
1570      TYA           \ to the first non-space character
1580      CLC           \ of the statement.
1590      ADC &B
1600      STA &37
1610      LDA &C
1620      ADC #0
1630      STA &38
```

```
1631
1640      JSR nxtwrd    \Call the command line interpreter
1641
1650      BCS giveup    \Exit if no match
1651
1660      DEY           \Adjust the offset of PTRA so that
1665      TYA           \ it points to the first charcter
1670      CLC           \ after the keytwrd just recognised.
1675      ADC &A
1680      STA &A
1681
1685      PLA           \Discard saved A and Y from stack
1690      PLA
1691
1695      PLA           \Discard RTI information from the
1700      PLA           \ 6502 stack. This is automatically
1705      PLA           \ pushed by the BRK instruction.
1706
1710      PLA           \Discard return addr (of routine
1715      PLA           \ to check for "=") from stack
1716
1720      JMP (&0037)   \Execute the command
1721
1900 \ ***           Command Line Interpreter              ***
1902 \ ***  On entry, (837) shoutd point to the first     ***
1904 \ ***     char of the HELLO!rd in the program to be  ***
1906 \ ***     recognised. (&39) should point to the      ***
1908 \ ***     start of the keyword table.                ***
1910 \ ***  On exit;                                      ***
1912 \ ***     if C is set, a match was not made          ***
1914 \ ***     if C is ctear, the action addr is in       ***
1916 \ ***     &37,38, so that JMP (&37) will call it.    ***
1917 \ ***     Y contains the length of the word.         ***
1918 \ ***                                                ***
1920 \ ***  No abbreviations are allowed.                  ***
1922
2135 .nxtwrd
2140      LDY #0        \Beginning of words
2141
2150      LDA (&39),Y   \If no word, this is the end of the
2160      BEQ nomtch    \ table, so no match was made.
2161
2170      CMP (&37),Y   \If the chars do not match,
2180      BNE difrnt    \ try the next keyword.
2181
2190 .nextch
2200      INY           \Get the next character:
2210      LDA (&39),Y   \ if it is the end of the keyword,
2220      BEQ getadr    \ then get its addr, and jump there.
2221
2230      CMP (&37),Y   \If the chars match,
2240      BEQ nextch    \ try the next one.
```

```
2241
2250 .difrnt
2260      INY           \This keywrd is not the right one,
2270      LDA (&39),Y   \ so look for the end of it.
2280      BNE difrnt
2281
2290      INY           \Set the base pointer at (&39) to
2300      INY           \ the start of the next keyword in
2310      TYA           \ the tabte (i.e. 3 bytes past the
2320      SEC           \ end of this keyword, to allow
2330      ADC &39       \ for the address).
2340      STA &39
2350      LDA &3A
2360      ADC #0
2370      STA &3A
2371
2380      JMP nxtwrd    \Try the next keyword in the table
2381
2400 .getadr
2410      INY           \The correct keyword has been
2415      LDA (&39),Y   \ matched, so put its execution
2420      STA &37       \ addr in (&37).
2425      INY
2430      LDA (&39),Y
2435      STA &38
2436
2440      DEY           \Adjust Y so it contains the length
2445      DEY           \ of the recognised word.
2446
2450      CLC           \Flag "Match OK" , and exit
2455      RTS
2456
2460 .nomtch
2465      SEC           \Flag "No match", and exit
2470      RTS
2490
2494 \ *** Keywrd tabte. The format of this table ***
2496 \ *** is; Keywrd, zero byte, action addr     ***
2498 \ *** A keyword entry marks end of table.    ***
2499
2500 .keytab
2505      EQUS "BEEP"   \Keyword,
2510      EQUB 0        \ zero byte,
2515      EQUW beep     \ action addr
2516
2520      EQUS "DUMP"
2525      EQUB 0
2530      EQUW dump
2531
2580      EQUB 0        \End of keywrd tabte
2990
2992 \ *** BEEP - This command makes a beep by    ***
```

```
2994 \ *** printing a BEL character (CHR$7)       ***
3000 .beep
3010      JSR chksda    \Ensure end of statement
3011
3020      LDA #7        \Print a beep
3030      JSR OSWRCH
3031
3035 .alldne
3040      JMP cont      \Continue execution
3090
3092     \ *** DUMP - This command lists the names of ***
3094     \ *** all of the current active variables.   ***
3100 .dump
3105      JSR chksda    \Ensure end of statement
3106
3110      LDA #ASC"A"-1 \Set first initial letter for
3120      STA &39       \ variable (allow for first INC)
3121
3125 .newltr
3130      INC &39       \Use the next initial. Letter
3131
3140      LDA &39       \If all the letters have been
3150      CMP #ASC"z"+1 \ used up, go to next statement
3160      BCS alldne
3161
3170      ASL A         \Point (&3A) at the right ptace
3180      STA &3A       \ in the variabte link table
3190      LDA #4        \ in the top hatf of page 4
3200      STA &3B
3201
3205 .newptr
3210      LDY #1        \Get the MSB of the pointer to the
3220      LDA (&3A),Y   \ next variabte in the linked list.
3221
3230      BEQ newltr    \If it is 0, we have found the end,
3231                    \ so try another initial letter.
3232
3240      TAX           \Using X as a temp for the MSB,
3245      DEY           \ get the LSB of the pointer to the
3250      LDA (&3A),Y   \ next variabte in the list, and
3255      STA &3A       \ set (83A) to point to this
3260      STX &3B       \ variable.
3261
3262      LDA &39       \Print initiat letter of variabte
3264      JSR pchar     \ name (not stored in the list)
3265
3266      LDY #2        \Point at 1st stored char
3267
3268 .nxtchr
3270      LDA (&3A),Y   \Get the char in the name. If it
3275      BEQ namend    \ is the end of the name, exit.
3280      JSR pchar     \ Otherwise, print the char, and
```

```
3285      INY           \ go to the next one.
3290      BNE nxtchr    \ (Y never 0 here, so branch atways)
3291
3295 .namend
3300      JSR pnewl     \Print a new line after the end of
3305      JMP newptr    \ the name, and try the next link.
8000 ]
8010 NEXT
8015 @%=0
8020 PRINT'"Code length =&"~P%-start%
8190
8200 PRINT''''"** WARNING: Once assembled, the code"
8210 PRINT"generated by this program is not"
8220 PRINT"transferable between different BASICS"
8230 PRINT
8300 PRINT"Execute ""CALL &"~init""" to initialise."'
8310 END
8990
8992 REM *** Set up ROM entry points, allowing for ***
8993 REM ***  BASIC 1 and BASIC 2.                 ***
9000 DEFPROCsetup
9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
9030 IF $&8009=basic1$ THEN PROCset1 :ENDPROC
9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
9050 PRINT "NOT BASIC 1 OR 2"
9060 END
9290
9292 REM *** Set up BASIC 1 entry points          ***
9300 DEFPROCset1
9310 baschr = ASC"1":REM Used by init routi ne
9320 pchar  = &B571 :REM Print char in A: handle COUNT
9330 pnewl  = &BC42 :REM Print a CRLF, and zero COUNT
9340 chksda = &9810 :REM Check for statement detimiter
9350 cont   = &8B0C :REM Cont execution at next statement
9360 ENDPROC
9490
9492 REM *** Set up BASIC 2 entry points          ***
9500 DEFPROCset2
9505 baschr = ASC"2":REM Used by init routine
9520 pchar  = &B558 :REM Print char in A: handle COUNT
9525 pnewl  = &BC25 :REM Print a CRLF, and zero COUNT
9530 chksda = &9857 :REM Check for statement detimiter
9540 cont   = &8B9B :REM Cont execution at next statenemt
9550 ENDPROC
```

Note that the initialisation and setup routines are substantially the same as for the program in section 7.3 (although there are a few extra ROM routines). The program is longer than the last One, so it destroys the user defined function key area (this means that

funny things might happen if you press BREAK, as it is function key 10). The comrrfand line interpreter in this program (lines 1500 on) replaces the simple check for a 'B' in the last one.

The keyword recogniser (lines 1900 to 2470) is a subroutine all by itself. It uses a keyword table (lines 2500 to 2580) with each entry in the following format:

keyword characters
a zero byte to terminate the keyword
the action address of the keyword (2 bytes)

The end of the table is marked by the first character of the keyword being a zero byte.

The keyword recogniser is entered with the address of the table in &38,&39 and the address of the keyword to be recognised in &37,&38. If the keyword is recognised, the action address is put into &37,&38, the length of the recognised word is left in Y, and the carry flag cleared. If the keyword is not recognised, the carry flag is set.

Sending the address of the table in this manner allows more than one routine to use the same recogniser, with different tables. This means that it could also be used if new functions are being added as well.

The general operation of the keyword recogniser is as follows:

1 If the first byte of the name is a zero, the end of the table has been reached without a match, so exit with the carry flag set.

2 Compare the keyword in the table against the word in the program. If they both match until the zero at the end of the word in the table is found, get the action address of the keyword.

3 If any characters did not match, move the table pointer up to point to the next entry, and go back to stage 1 to try to match the next one.

When the keyword recogniser has returned, PTRA is updated to point to the first character after the keyword (lines 1660 to 1680). This allows the routine for the keyword to continue from there, to

get anything it needs from the text (or to just check for the end of the statment).

The variable dump routine works in a similar way to the BASIC one in section 3.1.2, but it doesn't print out their values.

7.5 A renumber utility

The RENUMBER command in BASIC is very limited; it only allows you to renumber the whole of your program. This is OK for small programs, but larger programs usually consist of a number of PROC and FN definitions, and it is very easy to loose track of these if they don't start on, say, 1000 boundaries. Using BASIC's blanket renumber on programs such as these will lose this structure completely.

This section describes how to add a new command to allow selected areas of the program to be renumbered. It is less than 512 bytes long, and so will fit in any 2 spare pages in memory (the user defined character and function key pages, perhaps).

Once the program has been assembled, and initialised by CALLing the start address, the new statement 'REN' has been added.

```
REN L, U; S, I
```

will renumber the lines in the program between L and U (inclusive) starting at S with an increment of I. All line numbers outside this range will be left unaltered. The GOTO and GOSUB line number references will be dealt with, in the same way as the BASIC RENUMBER command (in fact, the program JMPs into the RENUMBER code to do this!).

For example, if the following program was in memory:

```
  10 REM PROGRAM
 100 A=0
 101 B=0
 110 PROCthing
1000 DEFPROCthing
1010 ENDPROC
```

typing ‘REN 100,110;500,20’ would leave the program as:

```
  10 REM PROGRAM
 500 A=0
 520 B=0
 540 PROCthing
1000 DEFPROCthing
1010 ENDPROC
```

The following errors will be produced if the REN statement is misused:

REN syntax

This error is generated if the REN statement fails to find a comrm or a semicolon separating its arguments where expected.

REN space

This error is generated if there is not enough room for the pile of old line numbers the REN statment needs to put on the TOP of the program. This is similar to the ‘RENUMBER space’ error (a fatal error).

REN range

An attempt was made to renumber the program such that the new lines would be out of sequence. In the above example, if ‘REN 1000,1010.,1,2’ was typed this error would be generated.

REN type

A string was used as the argument to the REN statement (floating point numbers will be converted to integer if necessary).

EQU has not been used in this program, so it will work without modification with either BASIC 1 or BASIC 2 (although it looks a bit messy).

```
10 REM ***     Selective renumber utility ***
12 REM
14 REM           M D Plumbley 1984
16 REM
18 REM This program traps the BRK vector. If the error
20 REM  number is 4 ("Mistake") then the command line
22 REM  interpreter will test for the new command "REN",
```

```
 24 REM  and execute it if it is.
 26 REM
 28 REM   REN L, U; S, I   will renumber lines L to U of a
 30 REM  program, starting at S, tfith an increment of I.
 32 REM
 34 REM The code is assembted into the user key char
 36 REM  space. This can be changed by changing line 900
 38 REM
 40 REM The EQU directive is not used in this program, and
 42 REM  it will work without modification on either
 44 REM  BASIC1 or BASIC2 machines.
 46 REM
 99
100 PROCsetup             :REM Set up correct ROM entry points
490
495 REM *** OS routines and vectors ***
550 BRKV   = &0202
590
600 worksp = &0070        :REM Workspace area
605 svbrkv = worksp       :REM BRK vector save slot
610 lower  = worksp+&2    :REM Lower renumber limit
615 upper  = worksp+&4    :REM Upper renumber limit
620 start  = worksp+&6    :REM Start line number
625 number = worksp+&8    :REM Next renumber number
630 line   = worksp+&A    :REM Pointer to line in prog.
635 pile   = worksp+&C    :REM Ptr. to line no. pile
640 newnum = worksp+&E    :REM line no. to be used
690
695 REM *** BASIC system variables ***
700 himem  = &0006
705 top    = &0012
710 page   = &0018
715 count  = &001E
720 inta   = &002A        :REM Integer accumutator
725
750 renum  = 0            :REM To stop "No such var."
799
900 start% = &0B00        :REM User key/char
905
910 FOR opt%= 0 TO 3 STEP 3
920 P% = start%
950 [OPT opt%
1000 .init
1005        LDA &8015     \Test that the correct
1010        CMP #baschr   \ version of BASIC is
1015        BEQ basok     \ in the ROM.
1020
1025        BRK           \If it isn't, print an
1030 ]?P%=60:P%=P%+1      :REM error message
1035 $P%="Not BASIC ":P%=P%+LEN$P%
1040 ?P%=baschr:P%=P%+1
1045 ?P%=0:P%=P%+1:[OPTopt%
```

```
1050
1055 .basok
1060         LDA BRKV       \Load the current BRK vector
1065         LDX BRKV+1     \ into A and X.
1070
1075         CMP #newbrk MOD &100\If this routine is atready
1080         BNE ntsavd     \ set up, don't change BRKV.
1085         CPX #newbrk DIV &100
1090         BEQ saved
1095
1100 .ntsavd
1105         STA svbrkv     \It has not been set up
1110         STX svbrkv+1   \ atready, so save old
1115         LDA #newbrk MOD &100\ BRKV, and set up the new
1120         STA BRKV       \ one.
1125         LDA #newbrk DIV &100
1130         STA BRKV+1
1135
1140 .saved
1145         RTS
1190
1192    \ *** This is the new BRK handling routine ***
1200 .newbrk
1205     PHA            \Save A and Y on 6502 stack
1210     TYA
1215     PHA
1220
1225     LDY #0         \Get error number
1230     LDA (&FD),Y
1235
1240     CMP #4         \If "Mistake" , try new keywords
1245     BEQ mistak
1250
1400 .giveup
1405     PLA            \Restore A and Y from 6502 stack
1410     TAY
1415     PLA
1420
1425     JMP (svbrkv)   \Go to old BRK handler
1430
1490 \ *** If we get here, an error 4 ("Mistake") has   ***
1492 \ *** occurred, so attempt to recognise one of the ***
1494 \ *** command keywords in the table.               ***
1500 .mistak
1505     LDA #keytab MOD &100 \Get start of keyword table
1510     STA &39              \ into (&39)
1515     LDA #keytab DIV &100
1520     STA &3A
1525
1530     LDY &A         \Set (&37) to point to character
1535     DEY            \ before PTRA. It will then point
1540     TYA            \ to the first non-space character
```

```
1545     CLC            \ of the statement.
1550     ADC &B
1555     STA &37
1560     LDA &C
1565     ADC #0
1570     STA &38
1575
1580     JSR nxtwrd     \Call the command line interpreter
1585
1590     BCS giveup     \Exit if no match
1595
1600     DEY            \Adjust the offset of PTRA so that
1605     TYA            \ it points to the first charcter
1610     CLC            \ after the keyword just recognised.
1615     ADC &A
1620     STA &A
1625
1630     PLA            \Discard saved A and Y from stack
1635     PLA
1640
1645     PLA            \Discard RTI information from the
1650     PLA            \ 6502 stack. This is automatically
1655     PLA            \ pushed by the BRK instruction.
1660
1665     PLA            \Discard return addr (of routine
1670     PLA            \ to check for "=") from stack
1675
1680     JMP (&0037)    \Execute the command
1685
1690
1990 \ *** This is the command line interpreter bit ***
1992
2000 .nxtwrd
2005     LDY #0         \Beginning of wrds
2010
2015     LDA (&39),Y    \If no word, this is the end of the
2020     BEQ nomtch     \ table, so no match was made.
2025
2030     CMP (&37),Y    \If the chars do not match,
2035     BNE difrnt     \ try the next keyword.
2040
2045 .nextch
2050     INY            \Get the next character:
2055     LDA (&39),Y    \ if it is the end of the keyuord,
2060     BEQ getadr     \ then get its addr, and jump there.
2065
2070     CMP (&37),Y    \If the chars match,
2075     BEQ nextch     \ try the next one.
2080
2085 .difrnt
2090     INY            \This keyword is not the right one,
2095     LDA (&39),Y    \ so look for the end of it.
```

```
2100     BNE difrnt
2105
2110     INY            \Set the base pointer at (&39) to
2115     INY            \ the start of the next keywrd in
2120     TYA            \ the table (i.e. 3 bytes past the
2125     SEC            \ end of this keyword, to allow
2130     ADC &39        \ for the address).
2135     STA &39
2140     LDA &3A
2145     ADC #0
2150     STA &3A
2155
2160     JMP nxtwrd     \Try the next keyword in the table
2165
2170 .getadr
2175     INY            \The correct keywrd has been
2180     LDA (&39),Y    \ matched, so put its execution
2185     STA &37        \ addr in (&37).
2190     INY
2195     LDA (&39),Y
2200     STA &38
2205
2210     DEY            \Adjust Y so it contains the length
2215     DEY            \ of the recognised word.
2220
2225     CLC            \Flag "Match OK", and exit
2230     RTS
2235
2240 .nomtch
2245     SEC            \Flag "No match", and exit
2250     RTS
2490
2494 \ *** Keyword table. The format of this table ***
2496 \ ***  is; Keyword, zero byte, action addr    ***
2498 \ *** A 0 keyword entry marks end of tabte.   ***
2499
2500 ]
2505 keytab  = P%
2510 $P%     = "REN" :P%=P%+LEN$P%
2515 ?P%     = 0  :P%=P%+1
2520 !P%     = renum :P%=P%+2
2525 ?P%     = 0  :P%=P%+1:REM end of table
2600 [OPT opt%
2790
2792 \ *** This prints a REN syntax error ***
2800 .nocom                   \ If "," missing, or ";"
2805 .noscol                  \ missing, generate a
2810     BRK                 \ "REN syntax" error
2815 ]
2820 ?P%=&60:P%=P%+1
2825 $P%="REN syntax":P%=P%+LEN$P%
2830 ?P%=0:P%=P%+1
```

```
2835 [OPT opt%
2990
2992 \ *** REN - This command renumbers a selected ***
2994 \ *** part of a program                        ***
3000 .renum
3005     JSR gtinta           \Get the lower limit line
3010     LDA inta             \ number from the text at
3015     STA lower            \ PTRA, and save it in
3020     LDA inta+1           \ "lower". PTRB points to
3025     STA lower+1          \ the next item.
3030
3035     JSR getchb           \Check for a comma at PTRB,
3040     CMP #ASC","          \ and error if it isn't.
3045     BNE nocom
3050
3055     JSR gtintb           \Get the upper limit line
3060     LDA inta             \ number from the text at
3065     STA upper            \ PTRB, and save it in
3070     LDA inta+1           \ "upper".
3075     STA upper+1
3080
3085     JSR getchb           \Check for a semicolon at
3090     CMP #ASC";"          \ PTRB, and error if it
3095     BNE noscol           \ isn't.
3100
3105     JSR gtintb           \Get the start number for
3110     LDA inta             \ the renumbered section,
3115     STA start            \ and save it in "start".
3120     LDA inta+1
3125     STA start+1
3130
3135     JSR getchb           \Check for a comma, and
3140     CMP #ASC","          \ error if it isn't.
3145     BNE nocom
3150
3155     JSR gtintb           \Get the increment, leaving
3157                          \ leaving it in IntA.
3160
3165     JSR chksdb           \Check for end of statement
3170
3200     JSR settop           \ Set TOP to the top of the
3202                          \ program, and set up the
3205     JSR setup            \ initiat ptrs and numbers
3210
3490 \ ** Go through all the lines, piting up the    ***
3492 \ ** numbers, and checking for range.           ***
3500 .chklns
3505     LDY #0               \If we're at the end of the
3510     LDA (line),Y         \ program, go on to renumber
3515     BMI renlns           \ the lines
3520
3525     STA (pile),Y         \Othewise, add the line
```

```
3530     INY                    \ number to the pile on the
3535     lDA (line),Y           \ TOP of the program.
3540     STA (pile),Y
3545
3550     CLC                    \Add 2 to the pile pointer,
3555     LDA #2                 \ to cover the new line just
3560     ADC pile               \ added to it. Save the LSB
3565     STA pile               \ of the pile pointer in X,
3570     TAX                    \ as it will be needed to
3575     LDA pile+1             \ check against HIMEM.
3580     ADC #0
3585     STA pile+1
3590
3595     CPX himem              \If the pile pointer is now
3600     SBC himem+1            \ above HIMEM, give a
3605     BCS noroom             \ "REN space" error.
3610
3615     JSR rngchk             \Check the line range, and
3620     JSR nextln             \ move the pointer to the
3621                            \ next one, and go back to
3625     JMP chklns             \ do another.
3630
3635 .noroom                    \Generate a "REN space"
3640     BRK                    \ error.
3645 ]?P%=&61:P%=P%+1
3650 $P%="REN space":P%=P%+LEN$P%
3655 ?P%=0:P%=P%+1
3660 [OPT opt%
3990
3992 \ ** Once the line range has been checked, and the **
3994 \ **  pile set up, come here to renumber the lines **
3996
4000 .renlns                    \Re-set the line pointer and
4005     JSR setup              \ numbers.
4010
4015 .rnline                   \ If we're at the end of the
4020     LDY #0                 \ program, go on to resolve
4025     LDA (line),Y           \ the GOTO line references.
4030     BMI rsolve
4035
4040     JSR rngchk             \Set up "newnum" to be the
4045                            \ new line number to be
4050     LDA newnum+1           \ used, and set the line
4055     STA (line),Y           \ number of the current line
4060     INY                    \ to it.
4065     LDA newnum
4070     STA (line),Y
4075
4080     JSR nextln             \Move the line pointer to
4085                            \ point to the next line,
4090     JMP rnline             \ and jump back to renumber
4095                            \ the next one.
```

```
4100
4500 .rsolve                    \Jump into RENUMBER to fix
4505     JMP rsvgot             \ the GOTO refeFences.
4510
5989
5990 \ ** Set up current number to first,
5992 \  line pointer to PAGE+1,
5994 \  pile pointer to TOP
6000 .setup
6005     LDA start              \Set the next number in the
6010     STA number             \ renumbered section to the
6015     LDA start+1            \ start number in the
6020     STA number+1           \ renumbered section.
6025
6030     LDA #1                 \Set the line pointer to
6035     STA line               \ point to the first line
6040     LDA page               \ at PAGE+1
6045     STA line+1
6050
6055     LDA top                \Set the pile pointer to
6060     STA pile               \ the TOP of the program
6065     LDA top+1
6070     STA pile+1
6075
6080     LDA #0                 \Set the last number used to
6085     STA newnum             \ zero
6090     STA newnum+1
6092
6095     RTS                    \ Exit
6189
6190 \ ** Set "line" to point to next line        **
6200 .nextln
6205     LDY #2                 \Get the length byte of the
6210     LDA (line),Y           \ current line.
6212
6215     CLC                    \ Add the length of the line
6220     ADC line               \ to the line pointer.
6225     STA line
6230     BCC lineok
6235     INC line+1
6240 .lineok
6245     RTS                    \ Exit
6489
6490 \ ** Check range and set up newnum     **
6500 .rngchk
6505     LDY #1                 \Get the current line number
6510     LDA (line),Y           \ into X (LSB) and A (MSB)
6515     TAX
6520     DEY
6525     LDA (line),Y
6530
6535     CPX lower              \If the current line is not
```

```
6540     SBC lower+1          \ under the lower limit, go
6545     BCS notund           \ to "notund"
6550
6555     LDA (line),Y         \If it is, check that the
6560     CPX start            \ start line for the REN
6565     SBC start+1          \ section is above this
6570     BCC thistn           \ line. Otherwise, ...
6575
6580 .rngerr                  \Generate a "REN range"
6585     BRK                  \ error
6590 ]?P%=&62:P%=P%+1
6595 $P%="REN range":P%=P%+LEN$P%
6600 ?P%=0:P%=P%+1
6605 [OPT opt%
6610
6615 .notund                  \Check to see if the current
6620     LDA (line),Y         \ line number, which is
6625     CMP upper+1          \ not under the lcwer limit,
6630     BCC notovr           \ is also not over the upper
6635     BNE over             \ limit. If it is inside
6640     CPX upper            \ both these limits, go to
6645     BCC notovr           \ "notovr" to generate a new
6650     BEQ notovr           \ line number.
6655
6660 .over                    \If the current line number
6665     CMP newnum+1         \ is over the upper limit,
6670     BCC rngerr           \ check that the last line
6675     BNE thistn           \ used was not above this
6680     CPX newnum           \ one. If it was, the last
6685     BCC rngerr           \ renumbered line number was
6690     BEQ rngerr           \ too big, so error.
6695
6700 .thistn                  \If the current line number
6705     LDA (line),Y         \ is outside the REN limits,
6710     STA newnum+1         \ use the current line
6715     STX newnum           \ number as the new one, and
6720     RTS                  \ exit.
6725
6730 .notovr                  \If the current line number
6735     CLC                  \ is inside the REN limits,
6740     LDA number           \ use "number" as the new
6745     STA newnum           \ line number, and add the
6750     ADC inta             \ increment to "number".
6755     STA number
6760
6765     LDA number+1         \The AND is to make sure
6767     AND #&7F             \ that the line number never
6770     STA newnum+1         \ exceeds 32768. If it does,
6775     ADC inta+1           \ it will be lost off the
6780     STA number+1         \ end of the program.
6782
6785     RTS                  \ Exit
```

```
6790
6990 \ ** Get an integer from the text at PTRA **
7000 .gtinta
7005     JSR getnsa            \Get a <numeric> or <string>
7010     JMP typchk            \ at PTRA, and check type.
7015
7017 \ ** Get an integer from the text at PTRB **
7020 .gtintb                   \Get a <numeric> or <string>
7025     JSR getnsb            \ at PTRB.
7027
7030 .typchk                   \If it was a string, give a
7035     BEQ msmtch            \ "REN type" error
7040
7045     BPL noconv            \If it was real (type -ve),
7050     JSR cftoi             \ convert it to integer.
7052
7055 .noconv
7060     RTS                   \ Exit.
7065
7070 .msmtch                   \Generate a "REN type"
7075     BRK                   \ error.
7080 ]?P%=&63:P%=P%+1
7085 $P%="REN type":P%=P%+LEN$P%
7090 ?P%=0:P%=P%+1
8000
8010 NEXT
8015 @%=0
8020 PRINT'"Code length=&"~P%-start%
8190
8200 PRINT''''"** WARNING: Once assembled, the code"
8210 PRINT"generated by this program is not"
8220 PRINT"transferable between different BASICS"
8230 PRINT
8300 PRINT"Execute ""CALL &"~init""" to initialise."'
8310 END
8990
8992 REM *** Set up ROM entry points, allowing for ***
8993 REM *** BASIC 1 and BASIC 2. ***
9000 DEFPROCsetup
9010 basic1$ = "BASIC"+CHR$0+"(C)1981 Acorn"+CHR$&A
9020 basic2$ = "BASIC"+CHR$0+"(C)1982 Acorn"+CHR$&A
9030 IF $&8009=basic1$ THEN PROCset1 :ENDPROC
9040 IF $&8009=basic2$ THEN PROCset2 :ENDPROC
9050 PRINT "NOT BASIC 1 OR 2"
9060 END
9290
9292 REM *** Set up BASIC 1 entry points ***
9300 DEFPROCset1
9305 baschr  = ASC"1":REM Used by init routine
9310 cftoi   = &A3F2:REM Convert ftoating point to integer
9315 chksdb  = &980B:REM Check statement delimiter at PTRB
9320 getchb  = &8A13:REM Get character at PTRB
```

```
9325 getnsb  = &9B03:REM Get <numeric> or <string> at PTRB
9330 getnsa  = &9AF7:REM Get <numeric> or <string> at PTRA
9340 rsvgot  = &8FAD:REM Resolve RENUMBERed G0TOs
9345 ENDPROC
9490
9492 REM *** Set up BASIC 2 entry points ***
9500 DEFPROCset2
9505 baschr  = ASC"2": REM Used by i ni t routi ne
9510 cftoi   =  &A3E4:REM Convert ftoating point to integer
9515 chksdb  = &9852:REM Check statement delimiter at PTRB
9520 getchb  = &8A8C:REM Get character at PTRB
9525 getnsb  = &9B29:REM Get <numeric> or <string> at PTRB
9530 getnsa  = &9B1D:REM Get <numeric> or <string> at PTRA
9535 settop  = &BE6F:REM Set up TOP, check "Bad program"
9540 rsvgot  = &900D:REM Resotve RENUMBERed GOTOs
9545 ENDPROC
```

The initialisation routine, BRK handler, and keyword recogniser used by this program (lines 1000 to 2250) are the same as used in the program in section 7.4. The keyword table (lines 2500 to 2525) contains the single entry ‘REN’.

The general operation of the renumber utility, once recognised, is as follows:

1 The rest of the line after the ‘REN’ is decoded (lines 3000 to 3165). The keyword recogniser leaves PTRA pointing to the first character after the keyword, so this is used to get the first integer. The succeeding characters and integers are read in from PTRB, as this is advanced leaving PTRA still pointing to the first character after the ‘REN’.

2 The old line numbers are piled up above the program, from TOP onwards (lines 3500 to 3625). Also, each line is checked to make sure that the range of the renumbered lines does not overlap with the lines which will not be renumbered. This check is carried out by the routine ‘rngchk’ (which also calculates the new line number, but that is not used at this stage).

3 The lines are then renumbered (lines 4000 to 4095), using the routine ‘rngchk’ to calculate the new line number. This is not done at stage 2, in case there was not enough room 4 for the pile of line numbers; otherwise, the program would be left half-renumbered, with no GOTO references resolved.

4 The GOTO and GOSUB references are resolved. This part is in fact done by the routine in the ROM which is used by the BASIC RENUMBER command. It scans through the program, looking for line number tokens (section 2.3.2). If it finds one, it searches through the pile of old line numbers on top of the program, at the same time keeping track of the corresponding new line number in the program. When it matches the line numbers, it changes the tokenised line number to the new one. If it couldn't match them, it prints the 'Failed at xxx' message, before continuing.

The 'rngchk' routine is used both in stages 2 and 3. It decides whether the current line number is inside the range to be renumbered or not, and generates 'newnum' to be either the current line number, or a new renumbered line number accordingly. If it finds that the renumbering would cause a line number overlap, it generates a 'REN range' error.

The 'getinta' and 'getintb' routines get an integer from the line of text, leaving it in IntA (&2A to &2D). If the argument is in fact a string, a 'REN type' error will be generated. If the argument is a floating point number, it will convert it to an integer. The routine to get a <numeric> or <string> at PTRA will first copy PTRA into PTRB, and then get the <numeric> or <string> at PTRB (thus leaving PTRA unchanged). See chapter 10 for more details of these expression evaluation routines.

With the mechanisms described in this chapter, any number of new statements can be added (provided there is enough memory to keep them all in). The next chapters describe how other errors can be trapped, as well as the 'Mistake' error.