code as they are encountered while the program is running (they do not do it particularly efficiently, either). Writing in machine code avoids this translation process. Unfortunately, writing Assembly language programs is much more difficult than writing BASIC, and the cost in time and effort may not be worth the eventual saving. However, some programs — those using animated graphics, for instance — would not work as intended if they were written in BASIC alone.

There are many other ways of making smaller savings in processing speed. Use a variable instead of an actual number (e.g. MAX rather than 267.5) for faster access to values, especially in loops. Use different letters to start variable names, and spread these initial letters evenly throughout the alphabet. Use multiple statement lines (if that is possible) and create a sizeable interval between line numbers (such as 10). With FOR... NEXT loops, if the interpreter permits, leave off the loop counter variable (for example, use NEXT rather than NEXT LOOP). Inside a loop, try to avoid calculating the same value over and over again. Instead, calculate it outside the loop and incorporate it as a variable.

SAVING SPACE

Integer arithmetic not only saves time, it also saves space. Where it may take four or five bytes to store a real number, it need take only two to store an integer. This represents a major saving, especially where large arrays are involved. Other improvements to the speed of a program will also save space: using inbuilt or user-defined functions saves code, as does writing in Assembly language and using multiple statement lines. Compiling tends to *increase* the size of smaller programs and only saves space for large ones.

Removing REM statements is an obvious space-saver, and using shorter strings of text for prompts also helps. Putting large blocks of text into files that are stored outside the program keeps them out of the way when they are not needed (instructions and 'help' files are the biggest burdens). Remove as many spaces as is legal within a line, and use shorter line numbers and shorter variable names. If an array needs to be dimensioned but its exact size is not known, don't just guess a convenient round number. Instead, leave it until the information needed is on hand and then dimension it with a variable, like this:

```
10 INPUT"How many instances are in this
   category?";INSTANCES%
20 DIM ARRAY%(INSTANCES%)
```

This is called 'dynamic dimensioning' and it is something that BASIC offers and most other languages don't — so make the most of it!

Another technique involves increasing BASIC's memory allocation in RAM. This can be done by using commands like HIMEM. What these commands usually do is to change the area in RAM that is available to BASIC programs and variables. The normal use for this is to store machine code programs in a safe place where they won't be overwritten, but the same command can be used to access extra space from that normally reserved for the screen memory. If it does not matter what is appearing on the screen, then this is a good way to get an extra kilobyte of RAM. If it is not possible to change HIMEM, the screen memory can still often be used by PEEKing and POKEing directly to the memory locations reserved for it.

If all else fails and the program simply will not fit in the space available, many BASICs have a CHAIN command that allows one program to pass control to another. Some BASICs allow use of the COMMON command; this passes particular variables and their current values to the next program. CHAIN on home micros (if it exists at all) is usually a very simple command that enables all or none of the variables from the first program to be passed to the second.

If programs are written in a structured way, the individual subroutines should be capable of being written and tested independently. Their execution can also be individually timed. Write a simple timer like this one:

```
100 REM Use this first section to set any variables
105 REM that the routine will need (don't forget
110 REM to dimension arrays and fill them with
115 REM realistic data too if the routine uses any).
120 REM This program is in BBC BASIC and TIME
125 REM is a pseudo-variable that holds a value in
130 REM hundredths of a second, generated by the
135 REM system clock
200 START=TIME
210 GOSUB 2000:REM The routine being timed is
    called here.
220 FINISH=TIME
230 PRINT "Execution took"; (FINISH-START)/100;
    "seconds."
240 END
```

With this routine it is possible to experiment with different algorithms and other ways of increasing speed.

How To Be Quick

- Weigh carefully the demands of good style against the need for speedy, sometimes incomprehensible, code.
- Compile when you can; define functions and procedures if you can't.
- Avoid file accesses.
- Avoid absolute real numbers. Initialise variables and use integer arithmetic, if your micro allows it.
- Design your algorithms carefully and learn from the example of others.
- Consider the advantages and disadvantages of machine code. While it may be fast, it takes longer to write and to debug.
- Condense your code, and remove your REMs once you have a working version.