SCOPE

Computer Graphics Language

INSTRUCTION MANUAL

SCOPE
Computer Graphics Language

Instruction Manual

by
Allen E. Pendle

Published by ISP Marketing Ltd
Crown House, 38b High Street, Godalming, Surrey

First published October 1983

© 1983 ISP Marketing Ltd
"SCOPE: computer graphics language" is a
registered trade mark.

"SCOPE" stands for:
Simple Compilation of Plain English

Publishers: ISP Marketing Ltd., Crown House,
38b High Street, Godalming, Surrey.

No part of this publication or the "SCOPE" cassette
program may be reproduced, stored in a retrievable
system or transmitted in any form by any means
without prior permission from the publishers.

Whilst every effort is made to ensure that the
contents of this Instruction Manual are accurate, the
publishers and the printer can accept no liability for
errors and omissions.

Cover design by The Graphics Workshop, Godalming.
Book design by Medimedia Ltd., Godalming, Surrey.
Printed by Willow Press Ltd., Godalming, Surrey.

Contents

Introduction

SCOPE is a multitask fully structured language for the 48K Spectrum. From 31 command words one can build up a program within BASIC which is then compiled or re-written into an area of memory reserved for it.

The compiled program is then run by a USR call. Because the compiled code uses routines which are part of SCOPE, a SCOPE program will only run if SCOPE is present in high memory. SCOPE is primarily intended for high speed handling of graphics, colour, sound and animation.

Most words only handle integer numbers in the range Ø to 255. Plotting and printing graphics do not need numbers outside of this range.

To write games however, you will want the facility of calculating and printing integer numbers to the screen in the range Ø to 65535 - scores for example - and several words support this facility. The handling of colour and sound is approached in a different manner than in BASIC. See relevant sections of this manual for comprehensive details. In the following pages the language and its operation will be examined step by step.

Throughout this manual we have assumed that you have an understanding of the operation of your computer and will have read the BASIC MANUAL.

SCOPE does not cover the generation of user defined graphics which will be found at page 93 of your BASIC MANUAL.

Loading instructions

Type LOAD "scope"
   Start tape and press ENTER, when loading is complete it will be indicated on the screen and after a short delay the screen and low memory will be cleared ready for you to start.

To save SCOPE programs on tape:

a) UNCOMPILED PROGRAM; FOLLOW BASIC Procedure.
b) COMPILED PROGRAM; use the "SAVE BYTES" facility in BASIC procedure.
Note: It is usually preferable to save the uncompiled program and compile it when loaded.

Memory organisation

Before you start writing a SCOPE program you must nominate an area where the compiled SCOPE code is to be written. You must also nominate an area where SCOPE routines should be written, because SCOPE ensures that any routines are written in a special area.

After SCOPE has been loaded, the memory addresses from 23755 to 59999 are available to the user for programming. You will be typing SCOPE words in BASIC REM statements and therefore a SCOPE program must start with the word ORG.

```
        10 REM Org;40000,50000:
```

The above program line actually means:

Start compiling SCOPE at memory location 40000 and any SCOPE routines at 50000. Therefore, you will have allowed yourself 16K area for BASIC (from 23755 to 39999), 10K for SCOPE statements (40,000 to 49,999) and 10K for SCOPE routines (50,000 to 59,999).

These parameters are totally under your control and at any time you can change these parameters.

Sometimes it is useful to know how much of any area you have used, this can easily be done as follows: to find how much of the BASIC area has been used type:- as a direct statement:

```
      PRINT  PEEK  23645 + 256*  PEEK  23646
```

press ENTER

and the last address used will be printed on the screen.

To find out how far you have progressed in the SCOPE program area, type in as a direct statement:

```
      PRINT  PEEK  60316 + 256*  PEEK  60317
```

press ENTER

this will print to the screen the next location where any SCOPE programs will be written.

To find out how far you have progressed in the SCOPE routine program area enter as a direct statement

```
      PRINT  PEEK  60318 + 256*  PEEK  60319
```

this will print to the screen the next location where any SCOPE routines will be written.

It is suggested to start off with that you use 40000 for the SCOPE compiled program and 50000 for the SCOPE routines.

This word ORG must always be the first word of any SCOPE program, and the word EXIT must always be the last word to enable a successful return to BASIC.

It is therefore recommended as good practise to type the following 2 lines first when starting to write any SCOPE program.

```
    1Ø  REM  Org;4ØØØØ,5ØØØØ:
    2ØØ  REM Exit;
```

Syntax

There are certain simple rules about the grammatical construction of SCOPE words. Any transgression of these rules will be greeted with the error report "Parameter Error". It is therefore worthwhile spending a little time talking about those rules:

1 All command words must be entered in BASIC REM statements.
2 All words must be typed in full, BASIC KEYWORDS cannot be used.
3 All command words must start with a capital letter. All other letters should be in lower case. The word should be typed exactly as shown in the dictionary.
4 All command words must be followed immediately by a semi colon, there are no exceptions to this rule.
5 Any operands should be separated by a comma.
6 To make structured programming easier only one statement or word can be entered in a line.
7 Apart from the words which have no operands or text following them all words MUST end in a colon.

Compiling and running a SCOPE program

When a series of command words has been written as a SCOPE program it has to be compiled into machine code and rewritten into the SCOPE program area. To do this type as a direct statement:

```
PRINT USR 6Ø45Ø
```

press ENTER

you will either receive an error report if you have made a mistake or a number will appear in the top lefthand corner of the screen. This number is the next address for any further SCOPE program. There is a very good reason for this being the next rather than the last memory location used.

You may wish to use SCOPE to write some machine code routines which you wish to use within a BASIC program. You will therefore need to know the start number of each routine so that you may use a USR call from BASIC.

Used in this fashion SCOPE is a unique assembler using plain word mnemonics and enables you to write powerful machine code routines without any knowledge of the complexities of machine code.

To run your program when it has been compiled you can use either of the following direct statements - RAND USR 4ØØØØ (presuming that you have organised to this address) or LET X = USR 4ØØØØ. If you have used the word RND; in your SCOPE program always use the second statement so that the random number remains truly random.

Let us now look at the dictionary.

Dictionary

ADD

Format
Add; *a Bvar;, a number:*

What it does
Increases the Bvar; specified by the amount specified.

Example
```
   1Ø REM Add;A,25:
          or
   2Ø REM Add;C,5Ø:
```

Special comments
Must only be used to increase a Bvar; not a Var;

ATTR (attribute)

Format

Attr; *a variable, line number, column number:*
or or
a variable, a variable:

What it does

It puts into the variable specified the value of the colour attribute at the co-ordinates specified.

Example

```
   1Ø REM Attr;a,1Ø,1Ø:
             or
   2Ø REM Attr;a,b,c:
```

Special comments

The colour attribute number will be the paper and ink number as in the colour table (see colour section).

BDR (border)

Format

Bdr; *a number:*
or
a variable:

What it does

Changes the colour of the border. The colour numbers are the paper only colours in the range Ø to 7.

Example
```
   1Ø REM Bdr;6:
        or
   2Ø REM Bdr;a:
```

BVAR (big variable)

Format
Bvar; *a letter, a number:*

What it does
Tells the computer to note that until changed the specified Bvar; will contain the specified value.

Example
```
   1Ø REM Bvar;A,25ØØØ:
           or
   2Ø REM Bvar;C,5ØØØØ:
```

Special comments
A big variable can contain a number in the range Ø to 65535. There are 52 possible Bvar; totally separate from Var; but using once again the letters A to Z or a to z.

CALL

Format

Call; *a letter*

What it does

Instructs the computer to jump to the instructions following the specified label in the routine area, carry out the routine and then return and carry on with the next statement.

Example

```
   10 REM Call;A:
        or
   20 REM Call;C:
```

Special comments

This must only be used for labels in the routine area.

CHG (change)

Format

Chg; *a number:*

        or

    a variable:

What it does

Changes the permanent paper and ink colours and clears the screen.

Example

```
   1Ø REM Chg;16:
          or
   2Ø REM Chg;a:
```

Special comments

The colour numbers are the combined paper and ink colours - see section on colour.

DEC (decrease)

Format

Dec; *a variable, a number:*

What it does

Decreases the variable specified by the amount specified.

Example

```
   1Ø REM Dec;A,5:
            or
   2Ø REM Dec;C,1Ø:
```

DRAW

Format

```
Draw; colour number,
          +  a number,   + a number:
          or    or       or    or
          -  a variable, - a variable:
```

What it does

Draws a line from the last pixel plotted in the colour specified, the number of pixels specified to the right if the sign is plus, or left if the sign is minus, and the number of pixels specified up if the sign is plus or down if the sign is minus.

Example

(note: let us assume that Var; A = 5Ø and Var; B = 5Ø)

```
   1Ø REM Plot;58,5Ø,5Ø:
   2Ø REM Draw;58,+A,-B,
```

In effect this has drawn a line from co-ordinate 5Ø , 5Ø to co-ordinates 1ØØ , Ø.

END

Format

End;

What it does

Tells the computer that a routine must end here and that any following lines must be written in the SCOPE program area.

FSCR (fine scroll)

Format

Fscr; *a number:*

What it does

Instructs the computer to scroll the whole screen pixel by pixel in the direction specified. The number following the word indicates the direction and equates with the 4 cursor keys

5 =	scroll left
6 =	scroll down
7 =	scroll up
8 =	scroll right

Any other number will produce an error report.

Example

```
   1Ø REM Fscr;5:
          or
   2Ø REM Fscr;7:
```

Special comments

It is worth remembering that Fscr; 5 : immediately followed by FSCR; 6 : will give you a left downward diagonal scroll.

EXIT

Format

Exit;

What it does

Ends a SCOPE program and returns the user to BASIC.

GET

Format
Get; *a variable:*

What it does
Instructs the computer to put the code value of the last key pressed on the keyboard into the specified variable.

Example
```
   1Ø REM Get;k
          or
   2Ø REM Get;K
```

Special comments
If no key has been pressed since the last time that the keyboard was read no action is taken. It is good programming practice to use the Var; k or K as it is self-explanatory. (i.e. K = keyboard).

HALT

Format
Halt; *a number:*
or
a variable:

What it does
Tells the computer to stop processing and display the picture for n frames of the tele-

vision at 50 frames per second. n can have a value between 1 and 255. 255 would give you a delay of 5 seconds. Do not use Ø as that means Halt; for ever.

Example
```
   1Ø REM Halt;25:
         or
   2Ø REM Halt;5:
```

INC (increase)

Format
Inc; *a variable, a number:*

What it does
Increases the variable specified by the amount specified.

Example
```
   1Ø REM Inc;A,5:
         or
   2Ø REM Inc;C,1Ø:
```

JUMP

Format
Jump; *a letter*

What it does
Instructs the computer to jump to the

instructions following the specified label in the program area.

Example

```
   10 REM Jump;A:
        or
   10 REM Jump;C:
```

Special comments

This must not be used to jump to a label in a routine. (the word Call; caters for that).

LABEL

Format

Label; *a letter*

What it does

Makes a note of the next address so that you may jump to it later in the program.

Examples

```
   10 REM Label;A:
          or
   10 REM Label;C:
```

Special comments

You have 52 labels available to you, 26 capital letters and 26 lower case. Labels can be used either in the program area or the routine area. No label of course can be used more than once.

LIM (limit)

Format
Lim; *a Bvar;, a number, a label:*

What it does
A special word enabling you to jump to a label specified when a Bvar; reaches a specified value. For example if you are using a Bvar; to register a score you may wish to jump to a specified point in the program when the score reaches a certain level.

Example
```
   1Ø REM Lim;a,Ø,B:
          or
   2Ø REM Lim;C,5ØØØØ,A:
```

MINUS

Format
Minus; *a Bvar;, a number:*

What it does
Decreases the Bvar; specified by the amount specified.

Example
```
   1Ø REM Minus;A,1ØØ:
          or
   2Ø REM Minus;C,5ØØ:
```

Special comments
Must only be used to decrease a Bvar; not a Var;

NOTE

Format
Note; text:

What it does
Tells the computer to ignore anything on that line, the same as a BASIC REM statement. Enables the user to make comments for his guidance.

Example
```
   1Ø REM Note; this is a square:
```

NUM (number)

Format
Num; *colour number, a Bvar;, a line number, a column number:*

What it does
Prints the Bvar; specified in the colour specified at the co-ordinates specified.

Example
```
   1Ø REM Num;56,A,1Ø,1Ø:
           or
   2Ø REM Num;4Ø,C,5,1Ø:
```

ORG (organise)

Format

ORG;a number,a number:

What it does

Sets the address pointers for the SCOPE program area and routine area, the first number being the start of the SCOPE program area, the second number being the start of the SCOPE routine area.

Examples

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
               or
   1Ø REM Org;35ØØØ,45ØØØ:
```

OVER

Format

Over; *number Ø:*
      or
      number 1:

What it does

Over; followed by 1 tells the computer to put any future graphic display over the top of what was there previously without obliterating it. Over; followed by Ø restores things to normal where anything printed obliterates anything printed there before.

Example

```
   1Ø REM Over;Ø:
          or
   2Ø REM Over;1:
```

Special comments

If the same character is printed twice this has the effect of printing a blank space.

PLOT

Format

Plot;

   colour number, line number, column number:
          or   or   or
   a variable,   a variable,   a variable:

What it does

This illuminates a single pixel on the screen in the colour specified and at the co-ordinates specified.

Example

```
   1Ø REM Plot;6,1ØØ,1ØØ:
             or
   2Ø REM Plot;a,5Ø,b:
             or
   3Ø REM Plot;a,b,C:
```

Special comments

See section on colour for details of colour numbers.

PUT

Format

Put; *colour number,*
   line number, column number, text:
        or   or
   a variable, a variable:

What it does

Puts to the screen in the colours specified and the co-ordinates specified the text specified.

Example

```
   1Ø REM Put;32,1Ø,1Ø,SCOPE:
                 or
   2Ø REM Put;56,b,c,COMPUTER:
                 or
   3Ø REM Put;7,b,1Ø,LANGUAGE:
```

Special comments

See table in colour section for colour numbers. Text may consist of any characters including quotation marks. Except, of course, a colon which will end a word.

RND (random)

Format

Rnd; *a variable, a number, a number:*

What it does

This puts into the variable specified, a specified random number using the following simple rules: the number will be between Ø and up to and including the first number specified, plus the second number specified.

Example

```
   1Ø REM Rnd;a,1Ø,Ø:
      (this will give a random number between
      Ø and 1Ø)
   2Ø REM Rnd;a,1Ø,3:
      (this will give a random number between
      3 and 13)
   3Ø REM Rnd;a,7,56:
      (this will give a random number between
      56 and 63)
```

ROUTINE

Format

Routine;

What it does

Tells the computer that any lines following must be written in the routine area until it encounters an END; statement.

SCR (scroll)

Format

Scr; *a number:*
       or
     a variable:

What it does

This scrolls a specified number of lines of the display. The lines are counted from the bottom. The range is from 3 to 24 where 24 scrolls the whole screen.

Example

```
   1Ø REM Scr;24:
        or
   2Ø REM Scr;12:
        or
   3Ø REM Scr;a:
```

Special comments

Care must be taken to remain in the range of numbers or the system will crash.

SOUND

Format

Sound; *pitch number, delay number:*
            or   or
     a variable,   a variable:

What it does

Sounds the beeper at the specified pitch for the specified time.

Example

```
   1Ø REM Sound;1Ø,1Ø:
          or
   2Ø REM Sound;a,1:
```

Special comments

This word is not the same as BEEP in BASIC. See sound table in section on sound for details.

TEST

Format

Test; *a test number, a variable, a number, a label:*

What it does

The most complex word in SCOPE. It compares a specified variable with a specified number and conditionally jumps to a specified label. The condition is the first number in the word and is known as the test number. The details of the test numbers are given below.

Test number	The condition
194	Jump if the number tested is not equal to the variable tested
202	Jump if the number tested is equal to the variable tested
210	Jump if the number tested is greater than or equal to the variable tested
218	Jump if the number tested is less than the variable tested
196	Call the routine specified if the number tested does not equal the variable tested

204	<u>Call</u> the routine specified if the number tested does equal the variable tested
212	<u>Call</u> the routine specified if the number tested is greater than or equal to the variable tested
220	<u>Call</u> the routine specified if the number tested is less than the variable tested

<u>Example</u>

```
   1Ø REM Test;2Ø2,a,1Ø,A
               or
   2Ø REM Test;22Ø,a,1Ø,A
```

<u>Special comments</u>

Note that 4 tests make a Jump to a label and that 4 tests call a routine. If you should attempt to jump to a routine instead of a call you will either return to BASIC during the program or the system might crash.

VAR (variable)

<u>Format</u>

Var; *a letter, a number:*

<u>What it does</u>

Tells the computer to note that until changed the specified variable will contain the specified value.

Example

```
   1Ø REM Var;A,1Ø:
        or
   2Ø REM Var;c,5:
```

Special comments

A variable can only contain an integer in the range Ø to 255. For numbers above this see word Bvar; . There are 52 possible variables A to Z or a to z.

WIPE

Format

Wipe; *a number:*
or
a variable:

What it does

This clears a specified number of lines of the screen display. The lines are counted from the bottom. The numbers used must be in the range 1 to 24 where 24 wipes the whole screen.

Example

```
   1Ø REM Wipe;1:
        or
   2Ø REM Wipe;24:
        or
   3Ø REM Wipe;a:
```

Special comments

Care must be taken to remain in the range of numbers or the system will crash.

Handling colour with SCOPE

As you will know from your BASIC manual colours are designated by the numbers Ø to 7 used in separate paper and ink statements.

In SCOPE the paper and ink colour and the instructions for bright and flash are all contained in a single colour number. This colour number is calculated as follows: Multiply the paper number (Ø to 7) by 8 and add to the ink number (Ø to 7). Then if bright is required add 64 and if flash is required add 128. The table below shows the paper and ink combinations to which must be added 64 for bright, 128 for flash or 192 for both bright and flash.

Colour Table

	INK							
	Black	Blue	Red	Magenta	Green	Cyan	Yellow	White
PAPER								
Black	Ø	1	2	3	4	5	6	7
Blue	8	9	10	11	12	13	14	15
Red	16	17	18	19	20	21	22	23
Magenta	24	25	26	27	28	29	30	31
Green	32	33	34	35	36	37	38	39
Cyan	40	41	42	43	44	45	46	47
Yellow	48	49	50	51	52	53	54	55
White	56	57	58	59	60	61	62	63

Note: There is one word Bdr; in which the colour number must be in the range Ø to 7.

Handling sound with SCOPE

We have deliberately used the word sound as distinct from the BASIC word BEEP because it is used to make a sound as distinct from a musical note. Because of its speed SCOPE can handle repetitive sound statements and produce results where the pitch of the note alters so quickly that it produces a glissando sound.

The command word Sound; should not be used to attempt to write musical sequences, it is really intended for production of action type sounds.

The following table gives an indication of what note will be obtained by a particular pitch number and the delay numbers which you should use with the pitch number chosen.

Do not use delay numbers outside of the range given or you could find yourself waiting a quarter of an hour for the sound to finish.

Sound Table

Pitch No.	Sound Octave	Delay Numbers
225	C - 2	1 - 10
127	C - 1	1 - 20
63	Middle C	1 - 40
31	C + 1	1 - 80
15	C + 2	1 - 160
7	C + 3	1 - 255

Note: The higher the pitch number the lower the sound. The higher the delay number the longer the sound.

Examples of the use of SCOPE words

We have already established that a program must start with Org; and end with Exit; .

We will now give some examples of how to program with SCOPE. We recommend that you type these in and try them for yourself. Do not forget; when you have typed in the program enter as a direct statement PRINT USR 6Ø45Ø and press ENTER to compile it and type RAND USR 4ØØØØ and enter to run it.

The use of sound

```
10 REM Org ;4ØØØØ,5ØØØØ:
15 REM Var;a,l:
20 REM Label;A:
25 REM Sound;a,l:
30 REM Inc;a,l:
35 REM Test;194,a,95,A:
 2ØØ REM Exit;
```

This will give a laser type sound. In plain English this program means:- Arrange to compile the program starting at address 4ØØØØ and any routines starting at address 5ØØØØ.

Set the variable named a to the value 1. Note the address after the label and store it in the label called A. Make a sound with the pitch at the value of a and the duration of 1. Increase the value of the variable a by 1. Test whether the value of variable a has reached 95, and if it has not, jump back to make another sound. When the value of variable a is 95 then stop and return to BASIC.

Unlock your imagination, try making some sounds of your own. You could for instance, by adding to the above program, make 2 sounds one sliding upwards and the other sliding downwards concurrently.

Lets try it: Add to the above program:

```
17 REM Var;b,63:
26 REM Sound;b,1:
28 REM Dec;b,1:
```

Putting characters to the screen

Type in the following:-

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
   15 REM Var;a,Ø:
   2Ø REM Routine;
   25 REM Label;A:
   3Ø REM Put;59,1Ø,a,"SCOPE":
   35 REM End;
   4Ø REM Call;A:
  2ØØ REM Exit;
```

Experiment with different characters, words and positions on the screen. You will want to move characters. Add the following lines to the above program:

```
   36 REM Label;B:
   38 REM Over;Ø:
   45 REM Halt;3:
   5Ø REM Over;1:
   55 REM Call;A:
   6Ø REM Inc;a,1:
   7Ø REM Test;194,a,3Ø,B:
```

Now compile and run it.

Animation is also quite easy to do, the following program, although very simple shows the ability to animate in colour:

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
   15 REM Var;a,Ø:
   2Ø REM Routine;
   25 REM Label;A:
   3Ø REM Put;49,1Ø,a, ▚▞
   31 REM Halt;3:
   32 REM Put;59,1Ø,a, ▞▚
   33 REM Halt;3:
   34 REM Put;57,1Ø,a,  : (2 spaces)
   35 REM End;
   36 REM Label;B:
   4Ø REM Call;A:
   45 REM Halt;1:
   6Ø REM Inc;a,1:
   65 REM Test;194,a,24,B:
   7Ø REM Call;A:
  2ØØ REM Exit;
```

Plotting and drawing

Enter the following program which is a simple repetitive plot instruction:

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
   15 REM Var;a,Ø:
   2Ø REM Label;A:
   25 REM Plot;58,a,a:
   3Ø REM Inc;a,1:
   35 REM Test;194,a,174,A:
   4Ø REM Exit;
```

Now substitute the following line:

```
   25 REM Plot;58,a,1:
```

Now compile and run it again.

To draw properly, however, add the following:

```
   12 REM Var;b,174:
   28 REM Draw;58,+b,+b:
```

Now compile and run it again.

Now try this:

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
   15 REM Bdr;Ø:
```

```
 20 REM Chg;0:
 22 REM Var;d,100:
 25 REM Var;a,5:
 30 REM Var;b,0:
 35 REM Routine;
 40 REM Label;A:
 45 REM Note;SQUARE:
 50 REM Plot;6,d,50:
 55 REM Draw;5,+a,+b:
 60 REM Draw;4,+b,-a:
 65 REM Draw;3,-a,+b:
 70 REM Draw;2,+b,+a:
 75 REM End;
 80 REM Call;A:
 85 REM Halt;100:
200 REM Exit;
```

This will give you a square with each side a different colour. The size is dictated by variable a . Simply by altering that variable and the plot position you can draw different size squares at different positions.

Handling and printing numbers

In this example we show how to set an initial score at 25ØØØ, increase it by 25, and when it reaches 27ØØØ return to BASIC.

```
 1Ø REM Org;4ØØØØ,5ØØØØ:
 15 REM Bvar;a,25ØØØ:
 2Ø REM Label;B:
 25 REM Put;49,1Ø,4,SCORE:
 3Ø REM Num;49,a,1Ø,1Ø:
 35 REM Lim;a,27ØØØ,C:
 4Ø REM Add;a,25:
 45 REM Halt;3:
 5Ø REM Jump;B:
 55 REM Label;C:
2ØØ REM Exit;
```

Moving the whole display

Scr; can only move upwards.
Type in the following:

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
   15 REM Label;A:
   2Ø REM Put;56,21,1Ø, ▞▞▞▞▞▞▞:
   25 REM Scr;24:
   3Ø REM Put;56,21,1Ø, ▌▌▌▌▌▌▌:
   35 REM Scr;24:
   4Ø REM Var;k,Ø:
   45 REM Get;k:
   5Ø REM Test;194,k,11Ø,A:
  2ØØ REM Exit;
```

Now compile and run. To stop this PRESS the key N (the code for N is 11Ø see line 5Ø).

Fscr; can move the whole display pixel by pixel up, down, left or right.
Type in the following:

```
   1Ø REM Org;4ØØØØ,5ØØØØ:
   15 REM Put;56,21,Ø,SCOPE:
   2Ø REM Label;A:
   25 REM Fscr;7:
   3Ø REM Fscr;8:
```

```
 35 REM Var;K,Ø:
 4Ø REM Get;K:
 45 REM Test;194,k,11Ø,A:
2ØØ REM Exit;
```

This will print the word SCOPE in the bottom lefthand corner and it will glide diagonally up and to the right until it vanishes off the screen. To stop press N.

All of the examples you have just tried are very simple and are intended to show you the use of SCOPE words.

Structured programming with SCOPE

SCOPE gives you the ability to write your programs in small separate sections. You can test each section as you go and build up other sections which will use the original sections until perhaps one routine puts into motion the whole program.

Let us examine how this is done. We will pretend that you wish to write a very simple game in which an alien spaceship passes across the screen. You have a moveable gun which appears at the bottom of the screen which can fire at the alien.

In logical terms the program could be written as follows:

```
ROUTINE A

Prints Alien at
specific position on
screen and erases last print

ROUTINE B

Prints gun at
specified position
on screen and erases last print
```

```
ROUTINE C

Prints missile
at specified position
on screen and erases last print
```

We now have the three main routines. Now follow the operating routines:

```
ROUTINE D

Test if Key A pressed
if so update position and call
Routine B

ROUTINE E

Test if Key B pressed
if so update position and call
Routine C

ROUTINE F

CALL ROUTINE D
CALL ROUTINE E
```

The final program is now a simple loop.

```
LABEL G
CALL F
JUMP G
```

We have deliberately not written this as a program. As an exercise we suggest you write it yourself.

This is obviously a simplistic example but we hope it shows clearly how programs can be built up.

SCOPE

Computer Graphics Language

ISP Marketing Ltd, Crown House, 38b High Street, Godalming, Surrey