```
10 SID=54272
20 POKESID+23,0
30 POKESID+24,15
40 POKESID+5,40
50 POKESID+6,201
60 FOR N=1TO5
70 READ FH,FL,D
80 POKESID+1,FH:
   POKESID,FL:
   REM*PLAY
   NOTE*
90 POKESID+4,33
100 FORI=1TO300*
   D:NEXT I
110 POKESID+4,32
120 FORI=1TO100:
   NEXT I
130 NEXT N
140 FORI=1TO2000:
   NEXT I
150 POKESID+24,0
160 REM**FH FL D**
170 DATA 57,172,1
180 DATA 64,188,1
190 DATA 51,97,1
200 DATA 25,177,1
210 DATA 38,126,2
220 END
```

out frequencies lower than a specified one; and variable resonance can be applied to all the above filters to emphasise the frequencies around the cut-off points. Envelope filtering is a special case: it has a different effect from the others in that digitised ADSR values set for envelope 3 can be read from the SID chip and applied to a signal in such a way that the harmonic structure changes throughout the course of a note. It works like a variable filter.

These sophisticated features enable you to build highly complex sounds into interesting effects and convincing emulations of conventional instruments. The daunting aspect of SID is that CBM BASIC V2, the dialect supplied with the 64, provides no commands dedicated to sound at all. Control is exercised by PEEKing from and POKEing into the 29 SID control registers. A lot of BASIC code is therefore needed to generate even simple effects, and in some cases BASIC isn't fast enough to do full justice to the full range of SID's possibilities.

A full description of the SID control registers would require more space than a complete issue of THE HOME COMPUTER COURSE, but it is possible to play notes with pleasing tones as shown in the program on the left.

Although the program is 22 lines long, it plays merely five notes of a simple tune on one oscillator. Line 20 disconnects the filter from the oscillators; line 30 sets the master volume at its maximum; and lines 40 and 50 specify a piano-like envelope. Line 80 sets the note frequency; 90 and 100 start and stop the ADSR cycle and select a sawtooth wave for voice 1; and timing is achieved with FOR . . . NEXT loops in lines 100, 120 and 140.

Programming sound on the Commodore 64 in BASIC is a major effort in terms of both learning and writing code. Moreover, it can be a very frustrating exercise, as the only way to discover if a more complex set of BASIC statements will run in an acceptable time is by trial and error. If you want simpler methods of sound generation it is worth investigating the many sound editing programs that are commercially available. These are usually written in machine code and make the most of the marvellous features of the Commodore 64.

its own 256 or 128 bytes of memory.

Each of the four players has a missile figure associated with it that is two bits wide. To create players and missiles it is necessary to POKE the bit patterns that define their shape into a certain area of memory. The area of RAM used can be chosen by the programmer but the computer must be informed by setting a pointer to the beginning of the area.

If the programmer elects to use single-pixel vertical resolution then twice as much memory is required than for two-pixel vertical resolution. The following program designs player 0 in two-pixel vertical resolution as a space ship:

```
10 REM *** DEFINE A PLAYER ***
20 P=PEEK(106) -8:REM SETS P TO 2K BELOW
   TOP OF RAM
30 POKE 54279,P:REM SETS POINTER TO PM
   AREA
40 BASE = 256*P:REM SETS PM AREA BASE
   ADDRESS
50 FOR I = BASE+512 TO BASE+640
60 POKE I,0:REM CLEAR PLAYER 0 AREA
70 NEXT I
80 FOR I = BASE+512+50 TO BASE+530+50
90 READ A:POKE I,A:REM DEFINE FIGURE
100 NEXT I
110 DATA 16,16,16,56,40,56,40,56,40
120 DATA 56,56,186,186 146,186,254,186,146
```

Each player figure has several registers associated with it. These registers control colour, horizontal position and size. The last of these enables the programmer to increase the width of a player by a factor of two or four. Further registers control player-to-background priority. Missiles take on the colour of their parent player but missile size can be changed independently. For games applications a series of registers is set aside to detect on-screen collisions between players, missiles and background. However, there is no vertical position register for missiles or players. Vertical movement must be achieved by moving the contents of each location that holds the bit patterns for the figure up through the area of memory set aside for that player. This is a fairly straightforward task in Assembly language but would be relatively slow in BASIC. It is a good idea to try to make characters that move vertically as short and stubby as possible.

Player-Missile graphics considerably extend the Atari's graphics potential, although they are not as versatile or as easy to use as the Commodore 64's sprites. Here is a continuation of the program started earlier, to colour the space ship and move it from left to right across the screen.

```
130 POKE 559,46:REM ENABLE PM 2 LINE
   DISPLAY
140 POKE 53277,3:REM ENABLE PM DISPLAY
150 POKE 704,88:REM COLOUR PLAYER 0 PINK
160 GRAPHICS 0
170 SETCOLOUR 2,8,2:REM SET BACKGROUND TO
   DARK BLUE
180 FOR I = 0 TO 320
190 POKE 53248,I:REM SET HORIZONTAL
   POSITION
200 NEXT I
210 END
```

Rocket PM

Before a player object can be defined, it must first be drawn out and the decimal values for each row of pixels calculated