AMSTRAD AMSTRAD AMSTRAD AMSTRAD

Computing with the AMSTRAD

INC. AMTIX!

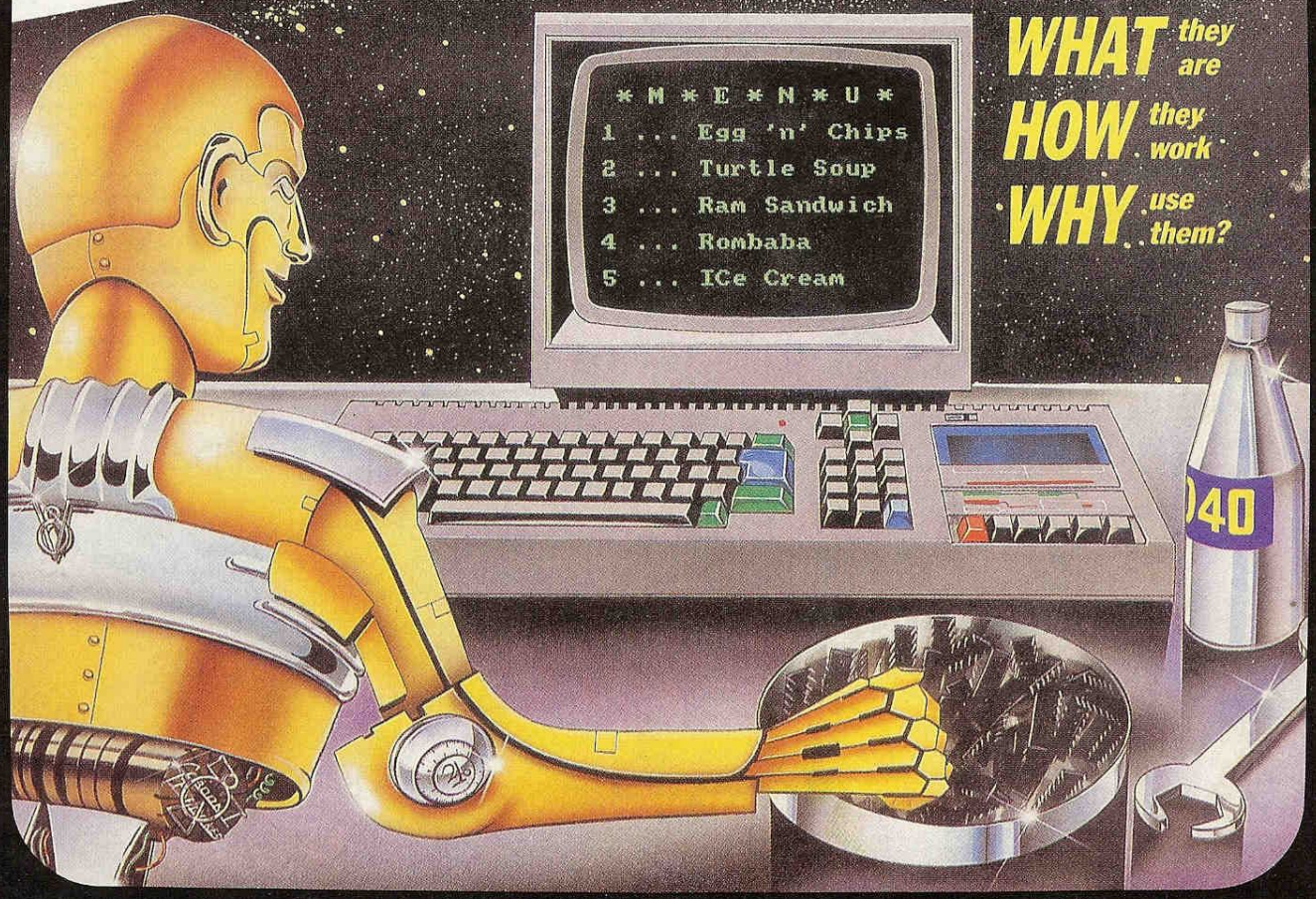
$4.50

The independent AUSTRALIAN magazine for Amstrad users
Registered by Australia Post – Publication No. TBP 1053

OCTOBER, 1987
A Database Publication

SAVE OVER $90.00
ON VOICE SYNTHESIZER/ LIGHTPEN PACKAGE
SEE CENTRE PAGE COUPON

Rom special



WHAT they are
HOW they work
WHY use them?

- **HOWZAT – EXCITING CRICKET GAME**
- **SOUND – SQ WILL HELP BUDDING BACHS**
- **NEWS • GRAPHICS – RUBBER BANDING**
- **DISC MAINTENANCE – AT A STROKE**

$0 % $AVINGS ON $OFTWARE

AMSTRAD – AMSTRAD –

Contents

Published monthly by Planet Publishing Pty Ltd
under licence from Database Publications Ltd.
Mail: P.O. Box 11, Blackmans Bay, Tasmania 7052.
Telex: AA58134, Attn. HT163 General enquiries, phone (002) 29 4377

Annual subscription (12 issues)
Australia $ 45
South Pacific (inc NZ) $A65
Subscriptions – see centre page

SQ will help budding BACHs get their cantatas together

Nigel Peters brings his series on Amstrad sound to a grand finale with an examination of the SQ function.

It seems a long time since we started out together on this exploration of the Amstrad's Basic sound commands. In the past few months we've covered the channel, pitch, duration and volume parameters and licked both volume and pitch envelopes.

After all that we had another look at the channel parameter and saw it was bit significant, learning how to hold, release and rendezvous notes on different channels.

And now, in this final article, we'll be taking a look at the SQ function.

At first sight the SQ command looks like it should be a mathematical function, squaring a number. However, what it does is test the state of the Sound Queue, hence SQ. If you're not too sure about what a channel's sound queue is, take a look at the articles in the May and July issues of *Computing with the Amstrad*.

SQ is used to gather information about the state of a channel. It can tell you if a note is playing — not always so obvious if you're using three channels at once — how many free places there are on the queue and if the note at the front of the queue has been held or rendezvoused with another note.

This is extremely useful when you're trying to figure out why your masterpiece doesn't work like it was supposed to.

When you want to know about a channel you just use SQ followed by that channel's parameter in brackets. So if you wanted to know about Channel A you'd use:

```
PRINT SQ(1)
```

and, of course, press the Enter key. Provided that you haven't got a note playing, you should get the figure 4 returned. You'll find that:

```
PRINT SQ(2)
```

and:

```
PRINT SQ(4)
```

will give the same result. Well, it's consistent, but what does it mean?

The answer is that it means:

* That there's no note playing.

* That there's no note held at the start of that channel's sound queue.

* The leading note in the queue, if any, is not rendezvoused with a note on any other channel.

* There are four free places in the sound queue.

That's a lot of information to get from one number, isn't it? The reason a single figure can throw so much light on the state of the channel queue is because the number returned from SQ is bit significant. Table I shows the significance of each bit.

number	bit	if set it show:
1	0	spaces in queue
2	1	spaces in queue
4	2	spaces in queue
8	3	rendezvous with A
16	4	rendezvous with B
32	5	rendezvous with C
64	6	first note held
128	7	channel playing

Table I: Bit sygnificant values of SQ

So, let's have a bit of a look at the significance of the 4 we get from:

```
PRINT SQ(1)
```

when there's no note playing on channel A. However, since we're dealing with bit significance it makes sense to use BIN$ to put the number into 8 bit binary form. So, with no note playing on channel A when we use:

```
PRINT BIN$(SQ(1),8)
```

we should get:

```
00000100
```

which is the binary version of decimal 4, returned. Figure I shows decimal 4 as a binary number along with the numbers of the bit positions.

bit number	7	6	5	4	3	2	1	0
binary byte	0	0	0	0	0	1	0	0

Figure I: Bit positions of binary 4

Now by combining the data in Figure I with the information in Table I we should be able to see how we learnt so much about the state of channel A from a single 4.

Let's look at bit 7. This is a 0, so we know that there is no note playing. If there was the binary bit would be set and the decimal value would be at least 128, and maybe more depending on the state of the other bits in the byte.

Bit 6 is also 0, and this shows that the front note on the queue is not held. If that bit was set it would mean that the first note on the queue is waiting for a RELEASE commands before it sounds.

There are still more 0s in bit positions 5, 4 and 3. These show that the first note in the sound queue, always supposing there is one, is not rendezvoused with any other.

If bit 5 was set then it would mean that the front note on that channel was rendezvoused with one on channel C. Bit 4 being set means that the note is rendezvoused with one on channel B.

As you might guess, a 1 in bit 3 show that the leading note is waiting for one on channel A.

The final three bits are just used to show the number of three places in the sound queue. In this case bits 2, 1 and 0 hold the binary number 100, which is decimal 4. This shows, as you might expect since there is no note playing, that there are four places free in that particular channel's sound queue.

If the last three bits are 011 it shows that there are 3 places free in the queue, if there are 010,2. I leave it to you to figure out how many unused places there are when the last three bits are 001 and 000.

As you can see, the figure returned by SQ already contains a powerful lot of information even though the Amstrad hasn't made a sound yet.

Now we'll move on th showing what SQ can do when the micro is actually making noises, but first you might like to set up the small Enter key with:

```
KEY 139,"SOUND135,0,0,0"+CHR(13)
```

Use this to clear the channels when you get stuck!

And now to making noises. Let's play a long note on channel A with:

```
SOUND 1,200,2000,7
```

and see what SQ can tell us about it.

Provided you do it while the note is playing, you'll find that:

```
PRINT SQ(1)
```

will give you the number 132 for your pains. I think you'll agree that the binary form gives more information. You'll find a quick:

```
PRINT BIN$(SQ(1),8)
```

will do the job, returning the binary value:

```
10000100
```

Figure II shows this number

bit number	7	6	5	4	3	2	1	0
binary byte	1	0	0	0	0	1	0	0

Figure II: Bit positions of binary 132

along with its bit numbers.

From this and Table I you should be able to see that since bit 7 is set the channel is active. Also as bits 2, 1 and 0 make the binary number 100, it should come as no surprise that there are four spaces left in the sound queue.

Now let's have two notes in succession on channel A and see what SQ can tell us about them.

Enter:

```
SOUND 1,200,2000,7
```

followed by

```
SOUND 1,300,1000,7
```

which gives us two long notes, the second one lower in pitch.

While the first note is playing:

```
PRINT SQ(1)
```

gives us the figure 131. Using:

```
PRINT BIN$(SQ(1),8)
```

gives us the more useful binary form of 131, returning

```
10000011
```

This shows that the channel is active (bit 7 is set) and that there are only three places left in the sound queue (bits 2-0). This tallies with what we know as we can hear the note and, since the second one hasn't started yet it must be on the queue. There are only three of the original four spaces left.

number	bit set	result
1	0	uses channel A
2	1	uses channel B
4	2	uses channel C
8	3	rendezvous with A
16	4	rendezvous with B
32	5	rendezvous with C
64	6	hold until RELEASed
128	7	flush the channel

Table II: Channel parameter values and action

As soon as the second note has started playing, enter:

```
PRINT BIN$(SQ(1),8)
```

again and you'll see that:

```
10000100
```

is the result.

Again bit 7, being set to a 1, shows that the channel is still active. However now that the first note is finished and the second one started, there are four places left in the queue (bits 2-0). I leave it to you to put two, three and four notes on the queue and see what happens to the results of:

```
PRINT BIN$(SQ(1),8)
```

as the notes come off the queue and sound.

What about bits 3, 4 and 5? These show whether or not the leading note of the queue is rendezvoused with one on another channel. Clear the channel A queue by pressing the small Enter key and type in:

```
SOUND 17,200,1000,7
```

This puts a note of pitch 200, duration 10 seconds and volume 7 on the channel A queue, making it rendezvous with one on channel B. Table II should refresh your memory about channel parameters.

Now using:

```
PRINT BIN$(SQ(1),8)
```

produces:

```
00010011
```

Bit 7 is 0, there is no note playing. Nor is the leading note held, so bit 6 is also clear. Bit 5 is 0 as the note isn't rendezvoused with channel C. However it is rendezvoused with one on channel B, so bit 4 is set. For obvious reasons bit 3, which marks a rendezvous with a note on channel A is clear.

The binary number formed by bits 2, 1 and 0 is 011, which means there are three spare places on the queue. The fourth space is held up by our note waiting forlornly for a date with one from channel B. Put it out of its misery with:

```
SOUND 10,1,1,1
```

and let's see about putting a note on channel B to rendezvous with one on channel A. We do this with:

```
SOUND 10,200,1000,7
```

Now:

```
PRINT BIN$(SQ(2),8)
```

gives us:

```
00001011
```

Notice the 2 in the brackets after the SQ. We're dealing with channel B.

By now you should have no trouble reconciling the fact that because the note is rendezvoused with one on channel A, bit 3 is set. As before, the last three bits show that there are three places left in the queue.

```
SOUND 17,1,1,1
```

provides a suitable partner for that note.

In order to show the significance of bit 5, enter:

```
SOUND 34,200,1000,7
```

which puts a note on the channel B queue, rendezvoused with one on channel C. Now:

```
PRINT BIN$(SQ(2),8)
```

gives:

```
00100011
```

showing that when a note is rendezvoused with one on channel C, bit 5 is set to 1.

Now let's look at bit 6. This is set when the leading note on the queue is held. Let's put a note on the channel A queue and hold it with:

```
SOUND 65,200,1000,7
```

and interrogate it with:

```
PRINT BIN$(SQ(1),8)
```

giving:

```
01000011
```

As you can see, bit 6 is set, showing that the leading note on channel A is held. If you now enter:

```
RELEASE 1
```

you'll see that:

```
PRINT BIN$(SQ(1),8)
```

returns:

```
10000100
```

while the note is playing and:

```
00000100
```

when it's finished.

We've already seen that bit 7 is set when the channel is active, that is, the note is playing. If you really want to test your understanding of SQ, try using it on channels whose leading notes are not only held but also rendezvoused. Then see what happens as you free these notes from their double restraints.

SQ isn't just used for fact finding and debugging channel queues. It can also be used to sort out some of the drawbacks we've come across when using sound with programs. Program I shows what I mean.

```
10  REM Program I
20  FOR noise=1 TO 10
30  PRINT "Loop number ";noise
40  SOUND  1,10+noise,100,7
50  NEXT noise
```

Program I

As you'll see when you run the program, the PRINT and SOUND commands are out of step. The numbers rush through to six, then appear one at intervals of a second. Meanwhile the notes are playing and carry on after the "Ready" message has appeared. You'll remember from before that this is a result of the sound queue getting full and holding up the Basic.

Well SQ can remedy this, by making the micro enter a delaying WHILE . . . WEND which holds up the loop while a note is playing. You do this wiht a line such as:

```
45 WHILE SQ(1)>=127:WEND
```

While a note is playing the WHILE . . . WEND loop cycles use-

lessly, holding up the FOR . . . NEXT loop. Of course when that note finishes so does the delay loop, and the program carries on. This keeps the messages and the notes in step, but it does slow the Basic program down.

Program II shows another problem.

```
10  REM Program II
20  FOR noise=1 TO 10
30  READ pitch
40  SOUND  1,Pitch,100,7
50  NEXT noise
60  DATA  100,200,300,400,500
70  DATA  600,400,300,200,100
```

Program II

Here you'll hear 10 notes, one after the other. Now suppose I wanted this wonderful melody to be playing while the micro was doing some maths? The aim might be to keep you entertained during a boring set of sums. How can I do it? If it was the two times tale I wanted to accompany with my tune I might try a line like:

```
25 PRINT 2*noise
```

but it doesn't do much good. What we want is a method that has the Amstrad doing two things at once, playing a tune and running a Basic program. This isn't as impossible as it seems. All it needs is the ON SQ( ) GOSUB command. Program III shows how it is done.

```
10 REM program III
20 ON SQ(1) GOSUB 70
30 WHILE NOT true
40 PRINT "The Amstrad is doing two things at
   once!"
50 WEND
60 REM Sound producing routine
70 READ pitch
80 IF pitch=0 THEN RESTORE:GOTO 70
90 SOUND 1,pitch,100,7
100 ON SQ(1) GOSUB 70:RETURN
110 DATA 10,30,50,70,90
120 DATA 80,70,60,50,40,0
```

Program III

'You'll learn a lot more by trying the sound commands out on your micro'

Here the messages are being printed out while a "tune" is being played and it's all due to the ON SQ GOSUB. Let's take a closer look at the program.

Line 20 is the first occurrence of the new command. All it does is to tell the Amstrad that when there is a free place on the channel A sound queue it is to momentarily interrupt what it is doing and go to the subroutine at the specified line number.

The figure in brackets after the SQ determines which channel is used to determine the interruption. As soon as it has performed the subroutine it goes back to where it left off.

In other words, the ON SQ GOSUB sets an interrupt. When the conditions are ripe — a space on the appropriate channel queue — the micro stops what it is doing while it goes to the specified subroutine, obeys the code it finds there, and goes back to where it left off. So long as the subroutine is short and the interruptions few and far between, the user hardly notices the interrupt.

So line 20 has set up an interrupt and, since there is nothing in the channel A sound queue at present, it immediately goes to the subroutine at line 70. This just reads a value for *pitch* from the data of lines 110 and 120 and uses the SOUND command of line 90 to play a note.

Then, while the note is playing the RETURN of line 100 sends the micro back to the main program, but only after setting up another interrupt with ON SQ GOSUB. This now waits for the next space to be free. In fact the first few times round the subroutine the interrupt will work straightaway as, at the beginning, there are four empty places to be filled. After that it has to wait until one of the notes has finished and a space appeared on the queue.

When an interrupt is over the micro goes back to the endless WHILE . . . WEND formed by lines 30 and 50. This prints out the message while a note is playing. As soon as one stops and a place becomes free an the queue the interrupt is triggered and the subroutine at line 70 is performed again, providing another note.

One important thing to notice is that as soon as an ON SQ GOSUB has sent the micro to the subroutine the interrupt is disabled. It only works once.

If you want to use it again you've got to issue another ON SQ GOSUB enabling the interrupt again. Every

time the subroutine has been called the interrupt is switched off. Line 100 just makes sure that it is switched back on again before it returns to the main program.

Now try leaving out line 80 of Program III and see if you can explain what happens. The micro tells you that the data is exhausted long before all the 11 *pitch* values have been used. Why?

The answer lies in line 100. This keeps on looking for more notes to put on the queue when spaces become vacant. After it's shoved 11 notes on the queue it still looks for more. Of course there aren't any now that the RESTORE of line 80 has gone, so the:

DATA exhausted in 70

message appears. The result is that the program crashes before all the notes have reached the front of the queue and been played.

And that's where I run out of data as well. I've finished sounding off, and there's nothing left in my queue except Program IV which just shows SQ and ON SQ GOSUB in action again.

Try figuring out how that works, and then try using all we've covered to write your own music.

You'll learn a lot more by trying the sound commands out at your micro than just by reading about them. And you'll have a lot of fun, which is what computing is all about.

If you come up with anything nice, let's see it at *Computing with the Amstrad*. And on that note I bid farewell.

```
10  REM Program IV
20  ON SQ(1) GOSUB 70
30  WHILE NOT true
40  PRINT "The Amstrad is doing two things
    at once!"
50  WEND
60  REM Sound producing routine
70  READ pitch
80  count=count+1
90  SOUND 1, pitch,100,7
100 IF count <10 THEN ON SQ(1) GOSUB 70
110 RETURN
120 DATA 10,30,50,70,90
130 DATA 80,70,60,50,40
```

Program IV

On second thoughts...

Have you ever written a program that ran first time and you never wanted to alter? I haven't either. So if you are going to start blowing roms, you've got to bear in mind the possibility of wanting to change their contents.

A 16k chip makes bugs expensive, so you really need a means of erasing the chip. This is done using ultra-violet light, and with a suitable source you can easily recover from mistakes without scrapping it.

The Watford eraser is a flourescent UV tube housed in a steel case. The de-luxe version has a safety switch that only turns on the light when the case is closed. Inside is a strip of conducting foam on which you mount the chips. Toast them for 15-20 minutes and you have clean eproms ready for reuse.

It's a robust unit which is not specific to any particular micro and should give years of trouble-free service. Replacement tubes are available and it will pay for itself many times over.

Wrapping it all up with a neat bit of rubber banding

Geof Turner and Michael Noels conclude their series on basic graphics

We've already seen how useful the EOR function can be when applied to our graphics routines. With its ability to draw and erase shapes on the screen without disturbing the background, it's useful in many application.

Let's just recap on how it works:

In Program I we're drawing a solid red rectangle. When a key is pressed, a blue line cuts through the centre of the rectangle to divide it equally into two pieces.

Press a key again, the line is removed and we end up with our original rectangle. This will be repeated over and over again if we continue to press a key.

All we are doing is EORing the line on and off again. Line 80 has asked the micro to draw in EOR mode while line 90 waits for a key to be pressed. Then the line is drawn over again.

As you'll remember, EORing something twice returns things to the original state.

Suppose we wish to cut the rectangle into two smaller rectangles,

```
10 REM PROGRAM I
20 MODE 0
30 FOR strip=100 TO 500 STEP 4
40 MOVE strip,0
50 DRAW strip,300,3
60 NEXT
70 WHILE -1
80 PRINT CHR$(23);CHR$(1);
90  WHILE INKEY$="";WEND
100 MOVE 100,150
110 DRAW 500,150,3
120 WEND
```

Program I

```
10 REM PROGRAM II
20 MODE 0
30 FOR strip=100 TO 500 STEP 4
40 MOVE strip,0
50 DRAW strip,300,3
60 NEXT
70 y=150
80 GOSUB 130
90 WHILE -1
100 IF INKEY(0)=0 THEN GOSUB 130:y=y+
2:GOSUB 130
110 IF INKEY(2)=0 THEN GOSUB 130:y=y-
2:GOSUB 130
120 WEND
130 IF y<0 THEN y=0 :RETURN
140 IF y>300 THEN y=300 :RETURN
150 PRINT CHR$(23);CHR$(1);
160 MOVE 100,y
170 DRAW 500,y,3
180 RETURN
```

Program II

but we're not quite sure where we wish to place the cut.

Program II is similar to the previous program, but this time we're given the opportunity to alter the position of the dividing line.

The line is originally placed in the centre of the rectangle as before, but by pressing either of the up or down cursor keys, the lines may be moved into another position.

The program works like this. Lines 10 to 60 are used to draw the rectangle while line 70 fixes the original value of y for the line. Line 80 calls the subroutine used to draw the blue line.

When it detects that one of the cursor keys has been pressed at lines 100 or 110, the program jumps to the subroutine at line 130, and draws the line again, which has the effect of erasing it.

The value of the variable *y* is either increased or decreased, depending on which key has been pressed, and then the subroutine is used yet again to redraw the line in its new position.

```
10 REM PROGRAM III
20 MODE 0
30 INK 15,18
40 FOR strip=100 TO 500 STEP 4
50 MOVE strip,0
60 DRAW strip,300,3
70 NEXT
80 PRINT CHR$(23);CHR$(1);
90 FOR strip=0 TO 48 STEP 4
100 MOVE strip,0
110 DRAW strip,150,12
120 NEXT
130 x=0
140 WHILE -1
150 IF INKEY(1)=0 THEN GOSUB 180:x=x+
4
160 IF INKEY(8)=0 THEN x=x-4:GOSUB 18
0
170 WEND
180 MOVE x,0
190 DRAW x,150,12
200 MOVE x+52,0
210 DRAW x+52,150,12
220 RETURN
```

Program III

The process is repeated every time a cursor key is pressed. Lines 130 and 140 simply prevent the line going past the top or bottom edges of the rectangle.

In Program II we've used the computer to help us to place the dividing line in the position that we want it. This is in fact the first step to computer aided design — usually referred to as CAD.

We can take this a stage further now. Imagine you are an architect and you wish to decide on the position of the front door of a house.

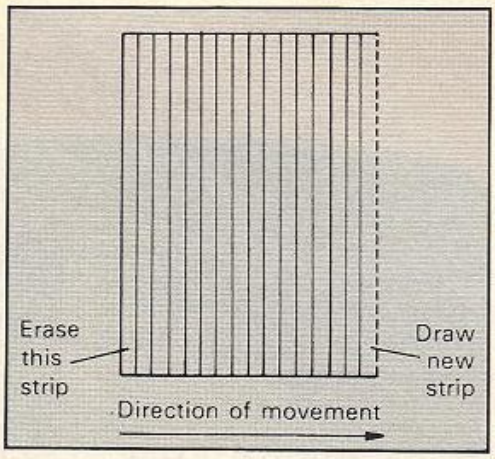


Figure I: How the door moves

```
10 REM PROGRAM IV
20 MODE 0
30 REM draw house
40 MOVE 0,50
50 DRAW 639,50
60 MOVE 200,50
70 DRAW 200,300
80 DRAW 600,300
90 DRAW 600,50
100 MOVE 200,300
110 DRAW 250,350
120 DRAW 550,350
130 DRAW 600,300
140 WHILE -1
150 WHILE shape=0
160 IF INKEY(59)=0 THEN shape=1
170 IF INKEY(61)=0 THEN shape=2
180 WEND
190 s=100
200 code=1
210 x=0:y=399
220 GOSUB 330
230 WHILE code=1
240 IF INKEY(39)=0 THEN GOSUB 330:s=s
-2:GOSUB 330
250 IF INKEY(31)=0 THEN GOSUB 330:s=s
+2:GOSUB 330
260 IF INKEY(0)=0 THEN GOSUB 330:y=y+
2:GOSUB 330
270 IF INKEY(2)=0 THEN GOSUB 330:y=y-
2:GOSUB 330
280 IF INKEY(8)=0 THEN GOSUB 330:x=x-
4:GOSUB 330
290 IF INKEY(1)=0 THEN GOSUB 330:x=x+
4:GOSUB 330
300 IF INKEY(9)=0 THEN code=0:GOSUB 3
30:shape=0
310 WEND
320 WEND
330 PRINT CHR$(23);CHR$(code);
340 MOVE x,y
350 IF shape=2 THEN 450
360 DRAWR s,0
370 DRAWR 0,-s
380 DRAWR -s,0
390 DRAWR 0,s
400 MOVER 0.5*s,0
410 DRAWR 0,-s
420 MOVER -0.5*s,0.5*s
430 DRAWR s,0
440 RETURN
450 DRAWR 0.75*s,0
460 DRAWR 0,-1.5*s
470 DRAWR -0.75*s,0
480 DRAWR 0,1.5*s
490 MOVER 0.15*s,-0.2*s
500 DRAWR 0.45*s,0
510 DRAWR 0,-0.45*s
520 DRAWR -0.45*s,0
530 DRAWR 0,0.45*s
540 RETURN
```

Program IV

Program III draws your house, represented by the red rectangle, and a green door is placed on the screen to the left of it. Using the left and right cursor keys, the door can be moved across the house until a suitable position is found.

Be wary of one thing when you run this month's programs. The graphics logic used in previous programs my affect their working.

You might have to resort to pressing Ctrl, Shift and Escape to reset the micro. It may be heavy handed, but it works.

Program III is very similar to the previous program, but notice that we are now moving a solid object rather than just a single line.

However it's not necessary to repeatedly draw the complete door. Instead, we are simply EORing the leftmost and rightmost edges of the door. This makes it look like we are moving the complete door across the screen.

Figure I shows how the door consists of a number of vertical strips. To move it to the right, all we need to do is to erase the leftmost strip and draw a new strip on the right edge of the door.

Notice that lines 150 and 160 which are used to detect left or right movement, are different.

One of them changes the variable *x* before calling the subroutine, the other alters it after calling the subroutine. This allows us to use the one subroutine for both left and right movement.

Although these programs are very simple examples, the same basic principles can be used in more complex design programs.

```
10 REM PROGRAM V
20 MODE 1
30 DEG
40 FOR angle=0 TO 360
50 PLOT 100*SIN(angle)+320,100*COS(an
gle)+200
60 NEXT
70 angle=1
80 GOSUB 130
90 WHILE -1
100 IF INKEY(1)=0 THEN GOSUB 130:angl
e=angle+1:GOSUB 130
110 IF INKEY(8)=0 THEN GOSUB 130:angl
e=angle-1:GOSUB 130
120 WEND
130 PRINT CHR$(23);CHR$(1);
140 IF angle=361 THEN angle=1
150 IF angle=0 THEN angle=360
160 MOVE 320,200
170 DRAW 100*SIN(angle)+320,100*COS(a
ngle)+200
180 RETURN
```

Program V

Program IV is a rather more sophisticated house design program.

```
10 REM PROGRAM VI
20 MODE 1
30 DEG
40 FOR angle=0 TO 360
50 PLOT 100*SIN(angle)+320,100*COS(an
gle)+200,1
60 NEXT
70 a=1:s=1
80 GOSUB 170
90 GOSUB 210
100 EVERY 50,0 GOSUB 130
110 EVERY 3000,1 GOSUB 150
120 WHILE INKEY$="":WEND
130 GOSUB 170:s=s+b:GOSUB 170
140 RETURN
150 GOSUB 210:a=a+b:GOSUB 210
160 RETURN
170 PRINT CHR$(23);CHR$(1);
180 MOVE 320,200
190 DRAW 100*SIN(s)+320,100*COS(s)+20
0,2
200 RETURN
210 MOVE 320,200
220 DRAW 75*SIN(a)+320,75*COS(a)+200,
3
230 RETURN
```

Program VI

Here we're dispensing with the use of colour and simply using line or wire frame graphics.

The outline of a house is drawn on the screen, and we are given the opportunity to add the doors and windows in any position we like. After the house is drawn — lines 40 to 130 — the program waits for either key D or W to be pressed. Pressing D will result in a door being drawn, W a window.

When the shape appears, it can be moved around the screen by using the cursor keys.

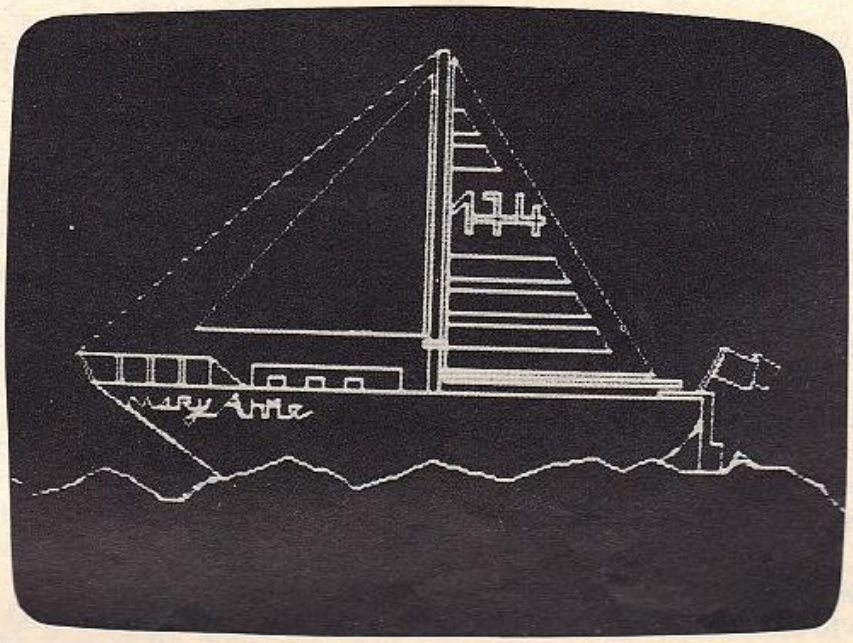
When the shape appears, it can be moved around the screen by using the cursor keys.

```
10 REM PROGRAM VII
20 MODE 1
30 INK 3,24
40 code=1
50 x=320
60 y=200
70 a=400
80 a=300
90 GOSUB 170
100 WHILE -1
110 IF INKEY(0)=0 THEN GOSUB 170:b=b+
4:GOSUB 170
120 IF INKEY(2)=0 THEN GOSUB 170:b=b-
4:GOSUB 170
130 IF INKEY(1)=0 THEN GOSUB 170:a=a+
4:GOSUB 170
140 IF INKEY(8)=0 THEN GOSUB 170:a=a-
4:GOSUB 170
160 WEND
170 PRINT CHR$(23);CHR$(code);
180 MOVE x,y
190 DRAW a,b,code+1
210 MOVE a-10,b
220 DRAW a+10,b
230 MOVE a,b-10
240 DRAW a,b+10
250 RETURN
```

Program VII

Parameter	Function
0	Normal
1	EOR
2	AND
3	OR

Table I



Rubber banding in action using Program VIII

The technique used is similar to the previous programs. When a cursor key is detected (lines 240 — 300), the subroutine at 330 is called to erase the shape before its new position is calculated.

The subroutine is called again, and the shape is redrawn in its new position.

As well as being able to move the objects around the screen, we can also vary their size.

Pressing the < or > keys will make the shape smaller or larger. We do this by altering the variable *s* so that this time the shape is redrawn in a different size.

Having placed our door or window in position it can then be fixed by pressing the Copy key.

This changes the value of the variable *code* to zero, which results in the shape being redrawn with normal logic. This gives us a permanent drawing.

After using the Copy key, the cursor returns to the top left hand corner of the screen ready to collect another shape when W or D is pressed.

Perhaps you might like to add other shapes of your own design to the program. These should be added to the subroutine from line 540, and the RETURN statement at line 540 will need to be repositioned at the new end of the subroutine.

Your new object should make the value of the variable *shape* equal to 3 when the appropriate key is pressed.

Finally you will need to redirect the program to the last part of the subroutine at line 350.

The EOR function is quite useful in making objects appear to move around the screen. In reality we are of course simply erasing the object in its old position and redrawing it in a new position.

This technique is often used in games applications. Program V draws a rotating dial which could be used as a scanner or radar screen in you space games.

Lines 40 to 60 draw a circle. Then a line representing the pointer is drawn from the centre to the perimeter of the circle.

The pointer will now rotate under the control of the left and right cursor keys.

Using the familiar method, the subroutine at line 130 repeatedly EORs the pointer off and on . Lines 100 and 110 decide which way the pointer should go by altering the value of angle.

Although in this program the movement is controlled by the cursor keys, we could use variables within the program to move the pointer.

If you've ever used a flight simulator program, you will probably

KEY	ACTION
CURSORS	MOVE PEN
U	PEN UP
D	PEN DOWN
DEL	DELETE LINE
COPY	FIX LINE

Table II

have seen this technique used to display some of the aircraft's instruments.

In Program VI we have made use of the Amstrad's built in timers to produce a real time clock. The second hand is moved every second using timer number 1.

Line 100 tells the program to jump to the subroutine at line 130 every second and go on to move the blue seconds hand.

Similarly line 110 tells the program to divert to the subroutine at line 150 every minute and so move the red minute hand. Timer number 2 is used to move the minute hand.

I haven't included an hour hand — perhaps you would like to add this, using timer number 3. The only problem is, you'll have to hang around for one hour to see if it works.

We'll move on now to the technique known as rubber banding. With this it is possible to produce line drawings on the screen using the cursor keys — a sort of computerised Etch-a-Sketch.

Program VII draws a line with one end fixed to point at the centre of the screen. The other end of the line is the free end and is marked with a crosshair cursor.

This cursor can be moved around the screen using, unsurprisingly, the Amstrad's cursor keys.

You will see how the free end of the line appears to stretch and shrink as it attempts to follow the cursor. It appears to act like a piece of elastic, hence the name rubber banding.

The line is movable because we are once again using the EOR function which repeatedly erases and redraws the line in its new position.

The program uses two sets of variables to mark the ends of the line *x* and *y* are the coordinates of the fixed end, while *a* and *b* are the coordinates of the free end. The subroutine at line 170 simply draws a line between *x,y* and *a,b* using EOR logic.

We can arrange to fix the line in any position by adding the following lines to Program VII:

```
150 IF INKEY(9)=0  THEN GOSUB 170:
code=0:GOSUB170:code=1:GOSUB 170
200 IF code=0  THEN RETURN
```

Continues page 65

```
10 REM PROGRAM VIII
20 REM INITIALISE VARIABLES
30 MODE 1
40 INK 3,24
50 code=1
60 n1b=1
70 x=320
80 y=200
90 a=320
100 b=200
110 REM DRAW CURSOR
120 GOSUB 240
130 REM DETECT KEYS
140 WHILE-1
150 IF INKEY(0)=0 THEN GOSUB 370:b=b+
4:GOSUB 370
160 IF INKEY(2)=0 THEN GOSUB 370:b=b-
4:GOSUB 370
170 IF INKEY(1)=0 THEN GOSUB 370:a=a+
4:GOSUB 370
180 IF INKEY(8)=0 THEN GOSUB 370:a=a-
4:GOSUB 370
190 IF INKEY(9)=0 AND n1b=1 THEN GOSU
B 440
200 IF INKEY(42)=0 AND n1b=1 THEN GOS
UB 500
210 IF INKEY(61)=0 AND n1b=0 THEN GOS
UB 500
220 IF INKEY(27)=0 THEN GOSUB 560
230 WEND
240 REM DRAW CURSOR
250 PRINT CHR$(23);CHR$(1);
260 MOVE A,B
270 DRAWR 0,10,2
280 DRAWR 20,20
290 DRAWR 10,-10
300 DRAWR -20,-20
310 DRAWR -10,0
320 MOVER 0,10
330 DRAWR 10,-10
340 MOVER -5,5
350 DRAWR 20,20
360 RETURN
370 REM DRAW LINE
380 IF n1b=0 THEN GOTO 420
390 PRINT CHR$(23);CHR$(1);
400 MOVE x,y
410 DRAW a,b
420 GOSUB 240
430 RETURN
440 REM FIX LINE
450 PRINT CHR$(23);CHR$(2);
460 MOVE x,y
470 DRAW a,b,1
480 x=a:y=b
490 RETURN
500 REM RAISE / LOWER PEN
510 GOSUB 370
520 IF n1b=1 THEN n1b=0 ELSE n1b=1
530 x=a:y=b
540 GOSUB 370
550 RETURN
560 REM DELETE LINE
570 PRINT CHR$(23);CHR$(0);
580 MOVE a,b
590 DRAW x,y,0
600 x=a:y=b
610 RETURN
```

Program VIII

Step aside, Rembrandt!

JON REVIS picks up his palette and brush and paints a pretty picture of the Advanced Art Studio

When releaed, the Art Studio, from Rainbird Software was, in my own words "the best artistic package I've seen for the Amstrad computer". Not being a company to rest on its laurels, Rainbird has now released The Advanced Art Studio, a disk-based program for the 6128, which makes the original pale into significance.

For the purposes of this review I'll run through many of the features available, but concentrate upon those peculiar to the advanced version.

The screen layout remains unchanged: A large drawing area with a two-line command section at the top of the screen. Pull-down menus are selected by moving a pointer over the required option and clicking the mouse/joystick/keyboard. All the original menus are present, but many have been modified or extended.

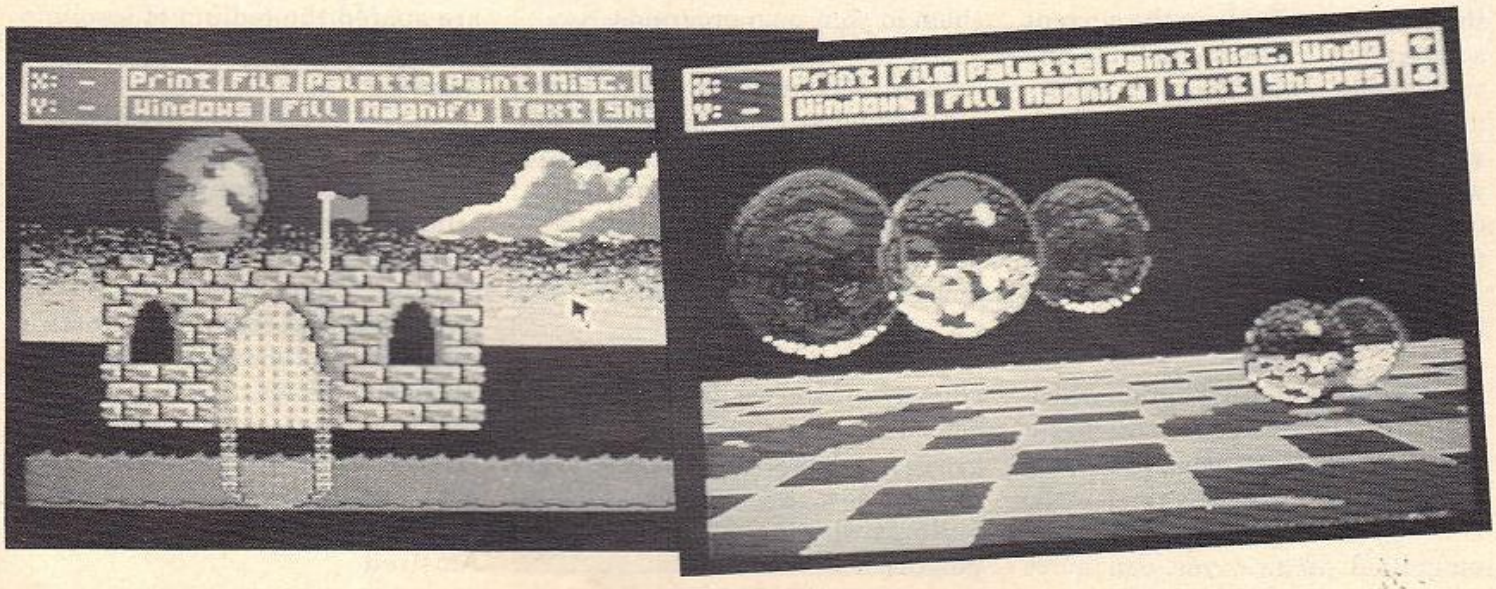
One of the new version's major differences is its ability to operate in Mode 0. No more trying to create a design using a maximum of four colours – you've now got a total of 16 to play with.

Calling up the Paint menu offers the same five options as before. However modifications have been made to both spray and brush facilities — the spray can still give eight different widths, but you can now adjust the speed at which it is applied.

The Brush mode asks you to select one of 15 multi-coloured brushes, a brush being a user-defined character which is applied to the screen using the pointer and button. The 15 on offer consist of eyes, lips, bricks, marbles and numerous abstract shapes. And if this was not enough, there are another seven files of brushes on the disk which can be accessed via the Pattern Editor.

The Pattern Editor is a much improved utility — the previous patterns were all based on a 11x11 grid. The new version allows patterns ranging in size from 1x1 up to 16x16, and these can be saved to disk in blocks of 15.

In addition to the standard inks, there are three pseudo-inks, labelled foreground, background and transparent. A brush designed using these will take on the currently



used. By using transparent as a background colour, the brush can be placed on a design without its own background over-writing the one currently there.

One outstanding enhancement to the Pattern Editor is the Capture feature. While on the main drawing screen you can define a window around the section of the design in which you are interested. On entering the Pattern Editor select Capture, and the program will take your piece of screen and place it in the editing grid where you can edit the design and re-apply it to the screen with the brush.

You will encounter the Pattern Editor when using the Fill menu. All the brushes can be used as fill patterns and as you edit a brush, a small window at the bottom of the screen shows how the characters appear when placed together in a group.

Manipulation of the colour palette is another feature which is now infinitely more powerful than before. You have the power to create animation effects by causing one or more colours to cycle through a user-defined sequence of inks. Across the top of the screen are all the colours available in the current screen mode. Select one of these and then, using a row of twelve sliders at the bottom of the screen, specify what sequence of coloured inks you want that colour to cycle through. One of the demo screens displays the effect beautifully by causing the words Art Studio to ripple across the screen.

Magnify is another option which has undergone some cosmetic changes, the most extensive being the View facility. At any time, while editing your two, four or eight times magnified picture, you can move the pointer over the box labelled View.

Clicking the button causes the full-size drawing to be displayed: Release it and the screen reverts to the magnified display. A small square highlights the area being edited whenever the view display is in use. This feature is a great time saver, and an improvement on the original program.

The text menu enables you to annotate your designs. All the usual features are present, printing text of different sizes and in different directions. The main improvement in the advanced version is the addition of a Font Editor. This superb software allows you to edit any or all of the 96 characters that are held in each font.

The full font is displayed at the bottom of the screen. Above this are three 8x8 grids containing three adjacent characters from the font.

As you edit one of the large characters its real-size counterpart is instantly updated. Rainbird have designed and supplied 10 different fonts on the disk and using a short Basic program, supplied in the manual, you can incorporate any of them in your own programs.

The Windows menu is as comprehensive as ever, but it too has been improved. By selecting the Multiple facility you can paste numerous copies of a window without having to pull down the Windows menu for each copy.

Any or all of the inks can be excluded from the window that is to be copied. This is used to stop background colours, encompassed by the window, from obscuring any features when pasted on to its new position.

The final option will save windows as disk files, and these can be used to build up a library of your favourite designs for use in future drawings. There are nine such windows already stored on the Art Studio disk.

One very useful facility, hidden away on the Miscellaneous menu, lets you Protect one or more colours of ink.

If you have just drawn one section of your masterpiece using red and blue, and you are worried about getting a little too close with a green spray can — worry no more! Merely protect red and blue, you can then splash about with the green to your heart's content, and there is no way that you can alter any red or blue pixels on the screen. If you need to make minor adjustments at a later time, just un-protect the inks.

The program uses an intelligent filing system. The disk drive can be accessed from several of the menus, yet the catalogue will only display the files relevant to that area of the package. Complete screens are available from the main menu, pattern files from the Pattern Editor, font files from the Font Editor, and so on. Using such a system you are spared the tedium of scrolling through dozens of irrelevant filenames.

The Advanced Art Studio is supplied with a 65-page manual which contains comprehensive chapters covering all the program's many features. There is even a step-by-step guide that lets you re-create a medieval castle scene, using predefined windows.

Until Rainbird release a mega-advanced version of Art Studio this one is definitely the tops for the Amstrad.

For the more technically minded

ROMs galore

Mike Cook gets to the heart of matters in his micro

The read only memory (rom) is at the heart of all micro computers. It comes in all sorts of guises, the most common being rom and eprom.

The story of the rom as a circuit element has a much longer history than its use in microcomputers, so let's look and see exactly what it is composed of.

First of all we need to define our terms. There are two classes of logic circuit; combination logic and sequential logic. Combination logic consists of an arbitrary number of logic gates or switches with outputs connected to inputs.

If you were to draw the circuit of a piece of sequential logic the inputs would be on one side, and the outputs on the other, and the signals would flow from left to right.

If you consider the whole circuit as a black box the signals coming out would depend upon what signals were going in. With the same input conditions you would always get the same output.

Counters, shift registers and rams are all examples of sequential logic; whereas decoders, multiplexers and roms are all examples of combination logic.

The most usual form of combination logic is the rom and it is basically built from a multiplexer and de-multiplexer. These are fancy names for switches. If you look at Figure I you will see an example of each with the switch equivalent.

The position of the switch is controlled by the logic levels on the select lines. If you were to make this out of a real switch you would need only one device but because in logic gates, signals can only flow in one direction, we need two.

To make these into a rom we need to take the outputs from a de-multiplexer and lay them over the inputs of a multiplexer. This forms what is known as a matrix. Note these do not make connection where they cross.

At some matrix intersections we place diodes — devices that will allow a current to flow through in only one direction, a sort of non-return valve. Figure II shows a simple rom. At the input of the multiplexer we place a logic zero. This means that we can place the logic zero on any column by the logic levels on the multiplexer select lines.

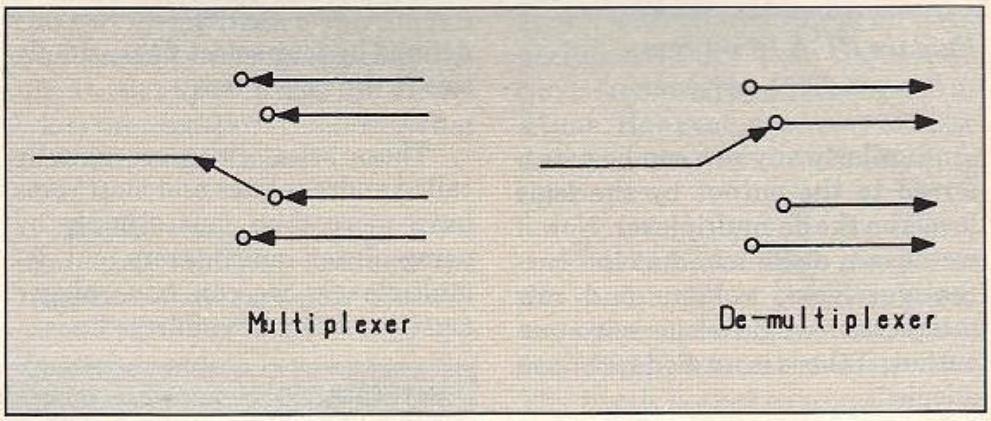


Figure I: Switch equivalent circuit

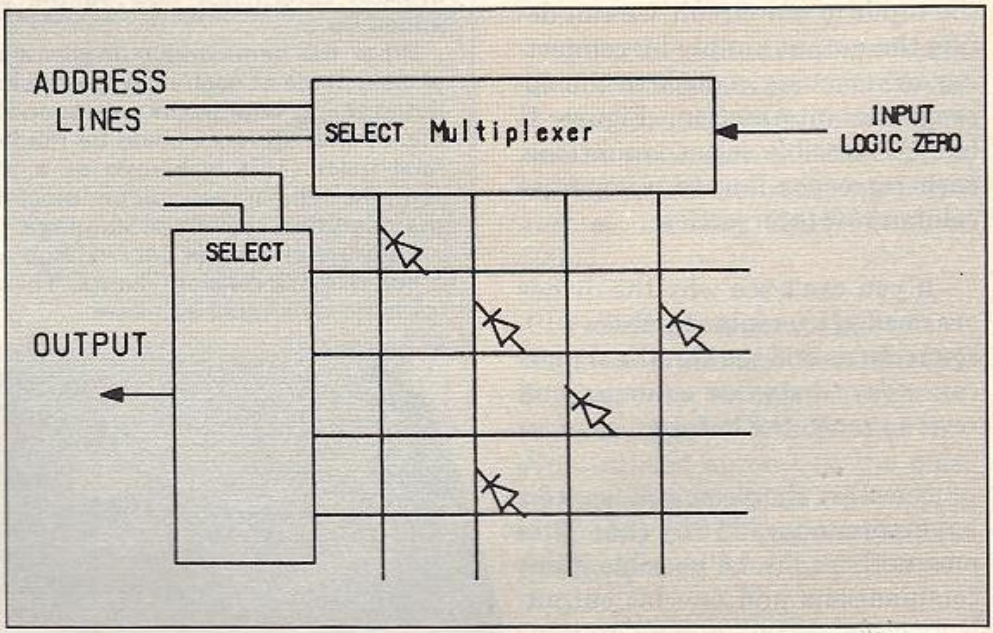
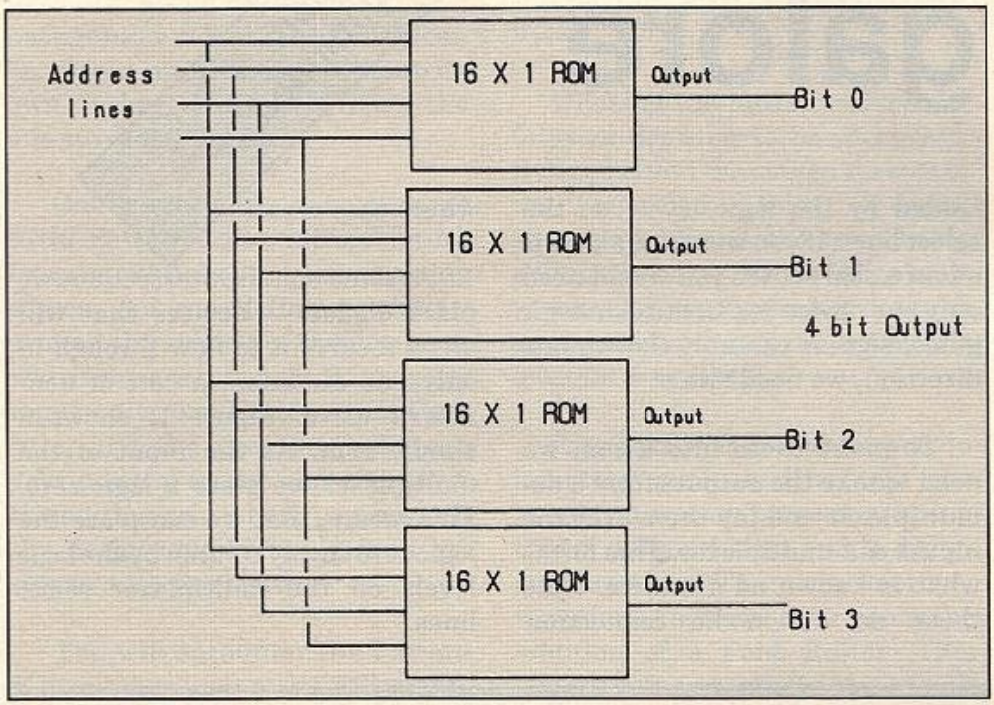


Figure II: 16 x 1 diode matrix rom



Fuigure III: A 16 x 4 rom

Similarly any row can be transferred to the output by the logic levels on the de-multiplexer. Now if we have a diode wired at the currently selected column and row intersection we get a logic zero as an output; if there is no diode we have a logic 1.

If we consider our select lines as the input to our circuit we can define the output simply by connecting, or removing, a diode at the appropriate intersection. Figure II has 16 possible input conditions, each corresponding to a separate column/row intersection.

If you can't see why the diodes are needed try replacing them with connections and see how the signals can weave between columns and rows to reach the output.

Now this simple circuit is a 16 x 1 rom (pronounced 16 by one). That means there are 16 possible input combinations and one bit output. Note we have sixteen combinations defined by four select lines, simply two to the power four.

These select lines are usually called address lines and most roms have more than four. Simply by using bigger multiplexers and de-multiplexers we can have bigger matrices and more address lines.

One bit per address, however, is not usually much use so several of these circuits are used with all the address lines connected together and the outputs separate. Figure III shows a 16x 4 rom constructed from the four 16 x 1 roms that were shown in Figure II.

You can see as we get bigger roms we need more and more diodes. Early diode matrix roms had these wired on large cards, the placing of the diodes determining the contents of the rom. Roms were often used simply to replace a collection of logic gates or perform signal decoding. With the advent of integrated circuits, roms could be made very small and with a large number of address lines. There were some standard roms like character generators of VDUs, but in the main, each customer wanted a different pattern for his own product.

INTEGRATED circuits are made by photo-etching a series of patterns in silicon, with each pattern produced by a photographic mask. The last mask used in the process determines how all the circuit elements are joined up.

What the semiconductor manufacturers did was to place a diode at each matrix intersection, the final mask determining whether it was connected up or not. This meant that all roms could be made the same until the final mask. This is known as a mask programmable rom, the type of rom you are most likely to find holding the operating system or language.

They are cheap but only in large quantities as there is a heavy mask-making charge. Therefore they are ideal for large debugged software.

When you are developing a rom you obviously don't want the heavy cost of making a mask every time you have a new version so there are a few cheaper alternatives. The first of these to be made was the fused-link prom or programmable read only memory.

In this each intersection would have a diode which was connected up by a thin link of nichrome metal, thus the whole rom consisted of logic zero outputs.

To program the device to give a logic one the appropriate intersections were selected on the address lines and a large surge of current sent through the links. This current caused the metal to heat up and melt, and once molten the surface tension pulled the link apart — you had literally blown the fuse.

This is where the term blowing a rom is derived although the way it is used to day is really quite wrong.

The major problem with a fuse link prom is that once the link is blown it can't be replaced. If you want a different pattern you have to throw it away and get a replacement. With the advent of placing software on roms mistakes frequently occurred so the eprom was developed. Eprom stands for erasable programmable read only memory. In this the fuse link is replaced by an FET (Field Effect Transistor) which is simply an electronic switch. The switch is on if there is a voltage on the gate connection and it is off if there is no voltage.

This type of transistor can be made to use virtually no current at all, so, if you connect a charged capacitor to its gate you will have a permenantly closed switch as shown in Figure IV.

Initially all the capacitors are discharged and you can induce a charge on any one by selection the appropriate address and pulsing a special programming line. This places a small charge on the capacitor and turns on the switch.

In practice the charge will slowly leak away so anything placed on a eprom will only stay there for about 30 years — long enough for it to become obsolete several times over.

In order to reprogram eproms you have to remove the charge on all the capacitors. This is done by shining ultra violet light on the chips through a small quartz window. I call this giving them a sun tan but what really happens is that the ultra violet light makes the dielectric of the capacitors conduct so that the charge leaks away.

The dielectric, by the way, is the insulating material between the two plates of the capacitor. The way the ultra violet light makes the dielectric conduct is by kicking the electrons in the silicon into a more energetic orbit. There they are less closely bound to the atom and can wander off and form an electric current.

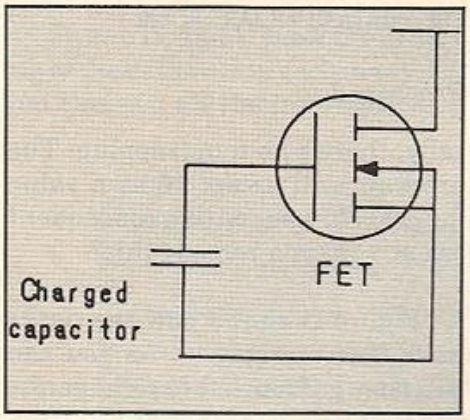


Figure IV: An eprom switch

The point about an eprom is that it all has to be erased at once, and to make the exposure to ultra violet light it has to be removed from the circuit. Also the programming is normally done out of the final circuit.

It would be useful in some circumstances if individual address locations could be reprogrammed without removal from the circuit. This has been achieved in the eeprom or electrically erasable programmable read only memory, which acts like normal read write memory only it is non-volatile. This means that when the power is removed it still retains its bit pattern. This is achieved like the eprom with charge stored on a capacitor only this time the charge can be removed by FETs one cell at a time. However, not many of these devices have found their way into computers.

The reason for this is that due to the complexity of the circuit you can't pack as much data on a given size chip as you can with eproms. The result is that eeproms are expensive compared to conventional eproms and have a small capacity.

A further disadvantage is that when writing data into eeproms the devices are slow. It takes about 1mS to write data but they can be read at speeds approaching that of normal memory that is about 500ns.

The future seems to be towards bigger and cheaper eproms, that require less stringent programming voltages. Clive Sinclair has even utilised eproms as the mass storage device in his new laptop computer, but despite that eproms will be around for a long time yet.

Consistency counts when organising information

by Pete Bibby

Last month we continued our exploration of the world of arrays. We saw how to create them using a DIM statement and then went on to ways of using them in our programs.

We made use of the fact that the elements of an array are in a numerical order to allow arrays to store ordered lists of information.

We then could get at, or access, different parts of the list using a numeric pointer. And, in addition, we saw how one pointer value could pick out several pieces of linked information from a set of parallel arrays.

Finally we ended up with a program which used the strange looking command:

```
DIM result$(3,3)
```

What's the second number in the brackets for? How does this differ from the arrays we've been used to? These are the questions we attempt to answer this month.

To do this let's start by taking a look at Figure I. This is a map of the desk positions of a class of nine pupils, showing who sits where.

Now suppose that you wanted to use your Amstrad to keep a record of this. One way of doing it would be to number the desks and use an array of nine elements to hold the names.

You could then find out who was sitting at, say, desk 1 by accessing the first element of the array. Program I shows this idea in action.

The mechanics of the program should hold no difficulties for you by now. Line 20 sets up an array *deskName$()*.

The following FOR . . . NEXT loop reads the names from the data list of lines 90 to 110 into the array.

Lines 60 to 80 allow you to "interrogate" the array to find out who sits at what desk number. Incidentally, there's no mugtrapping. The program will accept 10 as a value for *deskNumber* with a consequent crash. Can you remedy this?

```
10  REM Program I
20  DIM deskNames$(9)
30  FOR pointer=1 TO 9
40  READ deskName$(pointer)
50  NEXT pointer
60  PRINT "Enter a desk number"
70  INPUT deskNumber
80  PRINT deskNames$(deskNumber)
    "sits at desk number"deskNumber
90  DATA TOM,IAN,SUE
100 DATA LESLIE,BRIAN,ERIC
110 DATA CLIFF,JANE,ROBIN
```

Program I

Figure II shows how the desks have been numbered. As you can see, they go from left to right across the first row, 1-3, then the second, 4-6, and the third, 7-9. The data in the data statements of lines 90 to 100 reflect this arrangement.

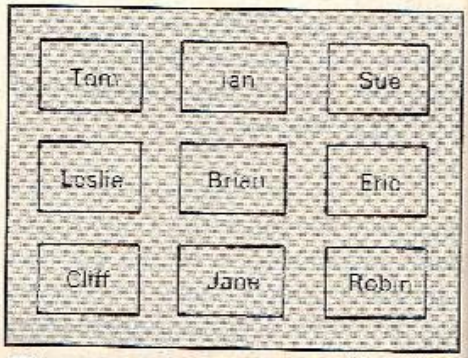


Figure I: A class of kids and their

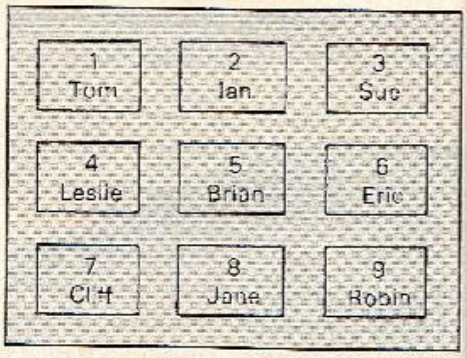


Figure II: Names, desks and numbers

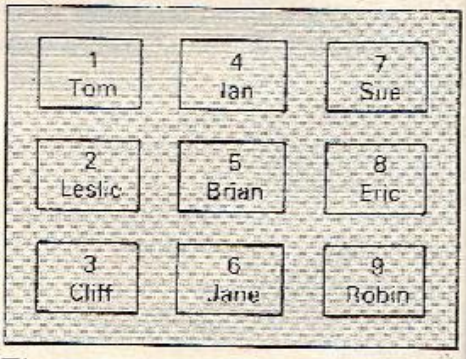
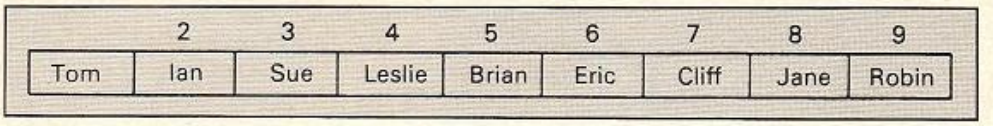
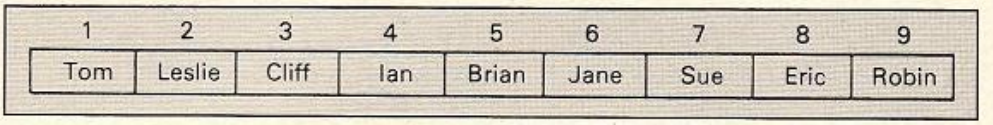


Figure III: Names, desks and other numbers



2	3	4	5	6	7	8	9
Tom	Ian	Sue	Leslie	Brian	Eric	Cliff	Jane	Robin

Figure IV: The order of array elements used in program I



1	2	3	4	5	6	7	8	9
Tom	Leslie	Cliff	Ian	Brian	Jane	Sue	Eric	Robin

Figure V: The order of array elements used in program II

So when the FOR... NEXT loop has done its job the first element of the array, *deskNames$(1)*, will contain TOM, the fourth LESLIE and the eighth JANE.

However we don't have to have the desks numbered in that order. Figure III shows a different way of numbering them that is perfectly acceptable.

If we use this alternative numbering for our array *deskNames$*, the first three elements will correspond to the left hand column, the next three to the middle one and the last three to the righthand column. Program II uses this alternative form of numbering.

The first five lines are more or less the same as Program I's and do the same job of creating and filling the array *deskName$()*. Notice, however, that the data in the data lines is in a different order, reflecting the new desk numbering system.

Rather than just having you enter a desk number and be rewarded with a name, as in Program I, this program does the opposite. It takes a name, forcing it to uppercase letters, and gives the number of the desk that person sits at.

Again the program needs mugtrapping. Look what happens if you enter a name such as HILDA. Obviously the INPUT of line 70 has to be trapped. Can you do it? You might have to use *deskName$()*.

Notice that in the above programs the date in the data statements, and subsequently the arrays, are exactly the same. Each set of data contains the names of all the children in the class.

```
10  REM Program II
20  DIM deskName$(9)
30  FOR pointer=1 TO 9
40  READ deskName$(pointer)
50  NEXT pointer
60  PRINT "Enter a name"
70  INPUT nameQuery$
80  nameQuerY$=UPPER$(nameQuery$)
90  elementNumber=0
100 WHILE found=0
110 elementNumber=elementNumber+1
120 IF nameQuery$=deskName$(element
    Number) THEN found=elementNumber
130 WEND
140 PRINT nameQuery$" sits at desk num
    ber" found
150 DATA TOM,LESLIE,CLIFF
160 DATA IAN,BRIAN,JANE
170 DATA SUE,ERIC,ROBIN
```

Program II

What does differ, however, is the order of the data within the array. In the first program the sixth element of *deskName$()* is ERIC. In the second program the array is arranged so that the sixth element is JANE.

Figure IV and V show the ordering of the data within each array. As you can see, all the information is there, it's just organised differently.

		Column 1	Column 2	Column 3
Row	1	Tom	Ian	Sue
	2	Leslie	Brian	Eric
	3	Cliff	Jane	Robin

Figure VI: The class as a two dimensional array

From all this there are two points to grasp. The first is that there is more than one way of organising a set of data. Although in this case it doesn't make much difference, in larger listings a little thought as to how you structure the information used by the program can save a lot of time and trouble.

The second point is that although there may be more than one way of organising information, when you've decided on how you're going to do it, stick to it. Suppose along with the array *deskNames$()* I had an array *age* containing the

deskName(1,1)	deskName(1,2)	deskName(1,3)
deskName(2,1)	deskName(2,1)	deskName(2,3)
deskName(3,1)	deskName(3,2)	deskName(3,3)

Figure VII: Elements of a 3 x 3 array

child's age. There'd be all sorts of problems if I used the first way of ordering the data for the names and the second for the ages.

If I used a pointer value of 2 to retrieve the name and age of the child at desk 2 I'd end up with Ian's name but Leslie's age. Be consistent in your methods.

As it's an important point, I'll repeat myself — you can structure your data in different ways. In Programs I and II we used two fairly obvious methods but there are others, some of which are a lot more realistic.

Suppose you were asked which desk Jane sat in? I doubt very much if you'd answer "Desk number 8 where the desks are numbered from left to right starting at the first row and working backwards". You'd probably say something like "the third one back in the middle row" or "the middle of the back row" or, from Figure I, "the third desk in the second column".

Instead of giving a number, you'd use two numbers to pinpoint the desk you're talking about, rather like map references or coordinates in maths.

This is a much more natural way of talking about a set of data like the class. It's also a lot more useful a method of ordering, as we'll see later.

Figure VI shows how the class can be arrange into rows and columns. Notice how easy it is to say that Eric is in the desk at row 2, column 3, or Robin is lurking at column 3, row 3. We can even just use numbers and say that Ian is at 1,2 where the first figure is the row number, the second the column number. So if we talk about the child at desk 3, 1 we mean the child at the desk that is both in row 3 and column 1. A glance at Figure VI shows this to be Cliff.

As there are three desks in each row and three desks in each column this way of structuring the data is called a 3 by 3 grid or array. You can refer to each desk (and each child) by two numbers, each between 1 and 3, which is a useful technique. Happily, Locomotive Basic allows you to structure your arrays in a way that reflects this grid-like organisation of data.

The secret lies in the DIM command. Up until now we've just used something like:

```
DIM deskName$(9)
```

to give an array of nine elements, one after the other. This form of array is known as a one dimensional or linear array. If you look at the way the elements are lined up in Figures IV and V you'll see how apt the name is.

We're not, however, struck with just linear arrays, we can use:

```
DIM deskNames$(3,3)
```

to set up what is known as a two dimensional array. Here instead of nine elements numbered 1 to 9, there are still nine elements but they are defined using two numbers, each of which can take values between 1 and 3.

Now, rather than just being a list of nine elements, the array is organised into a block of elements as shown in Figure VII. Notice how the numbers of the elements in the same rows and columns form a pattern.

It takes no great intuitive leap to see how the elements of *deskNames$()* shown in Figure VII can correspond to the raw data in Figure VI. If we used *deskName$(1,1)* to hold the string TOM, *deskNames$(1,2)* to hold IAN and so on, the two dimensional array will come to mirror the data. Program III shows this technique in action.

In this line 20 dimensions the array while the nested FOR . . . NEXT loops read the data from the data statements into it. The first time round the outer loop *row* is 1

```
10 REM Program III
20 DIM deskName$(3,3)
30 FOR row=1 TO 3
40 FOR column=1 TO 3
50 READ deskName$(row,column)
60 NEXT column
70 NEXT row
80 PRINT "Enter a row number"
90 INPUT rowNumber
100 PRINT "Enter a column number"
110 INPUT columnNumber
120 PRINT  deskName $(rowNumber,
    columnNuber)" sits at the desk on
    row number"rowNumber", column
    number" columnNumber
130 DATA TOM,IAN,SUE
140 DATA LESLIE,BRIAN,ERIC
150 DATA CLIFF,JANE,ROBIN
```

Program III

and while this is the case the inner loop cycles three times with *column* going from 1 to 3.

Each time round a string from the data list is read into *deskName$(row,column)*. The result is that *deskName$(1,1)* holds TOM, *deskName$(1,2)* holds IAN and *deskName$(1,3)* holds SUE. Try following the outer loop as *row* takes the values 2 and 3 and see what happens to the array.

(1,1) Tom	(2,1) Ian	(3,1) Sue
(1,2) Leslie	(2,2) Brian	(3,2) Eric
(1,3) Cliff	(2,3) Jane	(3,3) Robin

Figure IX: Column, row array

(1,1) Tom	(1,2) Ian	(1,3) Sue
(2,1) Leslie	(2,2) Brian	(2,3) Eric
(3,1) Cliff	(3,2) Jane	(3,3) Robin

Figure VII: Row, column array format

Now that the array has been given values, the program goes on to let you retrieve information from it. Unlike our previous programs, however, you have to give it two numbers to locate a particular element, not one. These are then stored in *rowNumber* and *columnNumber* and used to pinpoint the required array element in line 120.

Notice that in this case we've used the row number then the column number to pick out the element we're talking about. Figure VIII shows how the elements are identified in this method.

We could, if we wanted, locate the elements using the column number first, then the row number. Figure IX shows how this specifies the elements.

Program IV uses this method to structure the array. Now the column number comes first, then the row number.

If you can't see how the array subscripts vary for each particular name, try adding:

```
55 PRINT row,column, deskName
$(column,row)
```

to Program III and:

```
55 PRINT column,row,desk Name$
(row, column)
```

to Program IV to make things clearer.

Notice that the order of the names in the data lines has to be altered if you use this method. Again it's a case of the data staying the same but the pattern that holds the data differing. You pick the pattern that suits your purpose best.

```
10  REM Program IV
20  Dim desk Names$(3,3)
30  FOR column=1 TO 3
40  FOR row=1 TO 3
50  READ deskName$(column,row)
60  NEXT row
70  NEXT column
80  PRINT "Enter a column number"
90  INPUT columnMumber
100 PRINT "Enter a row number"
110 INPUT rowNumber
120 PRINT deskName$(column
    Number,rowNumber)" sits at the
    desk in column number" column
    Number" on row number "row
    Number
130 DATA TOM, LESLIE,CLIFF
140 DATA IAN,BRIAN,JANE
150 DATA SUE,ERIC,ROBIN
```

Program IV

But more of that next time. For the moment let's look at Program V, last month's Program VI.

Here a two dimensional array *result$(3,3)* is set up to hold the names of the first three pupils in each of the three subjects English, Maths and Computing. The nested FOR... NEXT loops of lines 30 to 70 are used to read the names from the data lists at the end of the program into *result$(subject,place)*. As the loops cycle, so each name in turn is read into one of the nine array elements.

You'll see that a code number is used for the subject. All the elements of *result$()* that have a 1 as the first figure inside the brackets refer to the English results. Similarly a 2 in that position signifies the Maths results and 3 shows that the element refers to the

Continues page 66

Call in BDOS to access files

Part X of Colin Foster's exploration of CP/M 2.2

Having already seen how CP/M stores files physically on a disc we can now go on to look at the BDOS function calls available to us for disc operations.

In the August issue of Computing with the Amstrad I described the non-disc BDOS functions and explained how they are used. The disc calls, function numbers 13 to 36, are used in a similar way and are listed in Figure I.

CP/M always reads and writes a file on disc, one record — 128 bytes — at a time. It is impossible to use the BDOS to get a particular physical track or sector.

If you want to do this sort of thing you must use the functions provided by the BIOS. I'll be discussing the BIOS in detail in future articles.

There are two ways of reading from and writing to a file using CP/M. The first, and more common, is called sequential access.

This simply means that we start at the beginning of the file and read or write records one after the other — sequentially — until we have either read all we want to or written the entire file.

No.	Function	Input parameters	Output parameters
13	Reset disc system	None	None
14	Select drive	E = drive no.	None
15	Open file	DE = .FCB	A = Dir code
16	Close file	DE = .FCB	A = Dir code
17	Search for first	DE = .FCB	A = Dir code
18	Search for next	None	A = Dir code
19	Delete file	DE = .FCB	A = Dir code
20	Read sequential	DE = .FCB	A = Error code
21	Write sequential	DE = .FCB	A = Error code
22	Make file	DE = .FCB	A = Dir code
23	Rename file	DE = .FCB	A = Dir code
24	Return login vector	None	HL = Login vector
25	Return current drive	None	A = Current drive
26	Set DMA address	DE = .DMA	None
27	Get alloc vector	None	HL = .AllocVect
28	Write protect disc	None	None
29	Get R/O vector	None	HL = R/OVect
30	Set file attributes	DE = .FCB	A = Dir code
31	Get DPB address	None	HL = .DPB
32	Set/get user code	E = &FF (get)	A = Current User (get)
		E = User Code (set)	None (set)
33	Read random	DE = .FCB	A = Error code
34	Write random	DE = .FCB	A = Error code
35	Compute file size	DE = .FCB	r0,r1,r2
36	Set random record	DE = .FCB	r0,r1,r2

Figure I: Disc BDOS function call

The second access method is called random record access. Here we tell CP/M which record of a file we want to read or write — we don't have to start at the beginning of it.

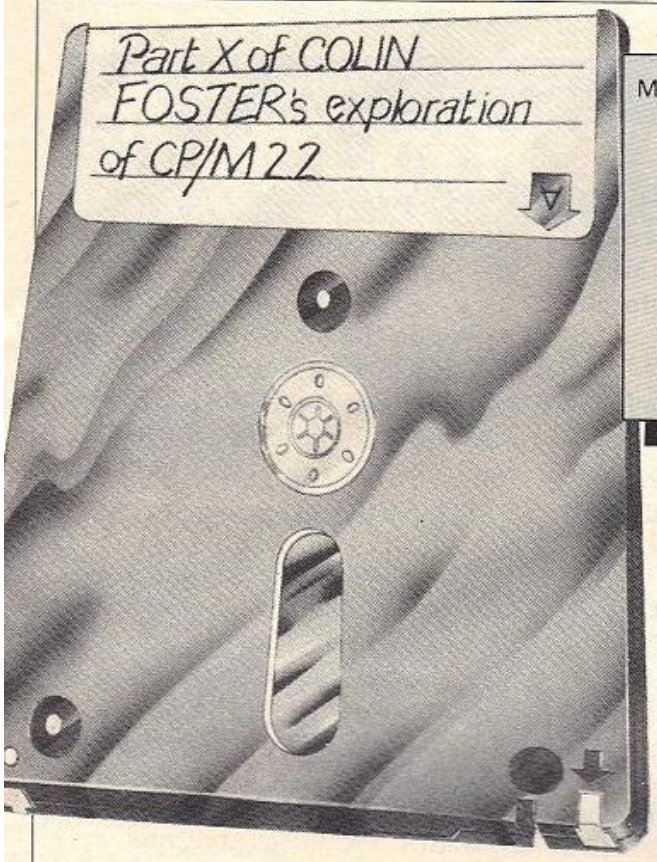
If we want to add information to the end of an existing file we must use this method. It is also much faster if we have, say, a database which has to change a single entry in the middle of a large data file.

All functions which perform disc read or write operation of any sort require us to pass the address of a memory file control block (MFCB) which we have set up for the file we are working with. Last month we saw how a directory FCB is made up on the disc and a memory FCB is an exact copy of this with the following exceptions:

Firstly byte zero of the MFCB — user number in the directory FCB — specifies which disc drive we are using. A value of 0 means the currently logged-in drive, 1 means drive A and 2 means drive B.

Secondly four extra bytes — 32 to 35 — are added to the end of the MFCB which are not present in the directory FCB on disc. Byte 32 is the current record byte for sequential file operations, and should initially be zeroed.

Bytes 33 to 35 are the random




```
MFCB:   defb    drive_num       Drive code – 0 = current logged, 1 = A, 2 = B
        defb    'FILENAME'      Filename and type in upper case
        defb    'TYP'           Ascii.
        defb    00              Extent no.
        defw    00              Reserved
        defb    00              Record count
        defb    00,00,00,00     Data allocation blocks d0 . . d15
        defb    00,00,00,00
        defb    00,00,00,00
        defb    00,00,00,00
        defb    00              Current record byte
        defb    00,00,00        Random record number
```

Byte:	0	1	2	3	4	5	6	7	8	
Contents:	0	"F"	"I"	"L"	"E"	"1"	" "	" "	" "	" "
Byte:	9	10	11	12	13	14	15			
Contents:	"C"	"O"	"M"	00	00	00	00			
Byte:	16	17	18	19	20	21	22	23	24	
Contents:	2	"F"	"I"	"L"	"E"	"2"	" "	" "	" "	" "
Byte:	25	26	27	28	29	30	31	32	. . .	35
Contents:	"C"	"O"	"M"	00	00	00	00		00	

Figure III: Contents of default FCB for an example command tail

record count bytes which are used in random access operations. For sequential file access they should always be zeroed.

Figure II shows a typical MFCB as we would set it up before using it to perform operation on a file using sequential access. It is the programmer's responsibility to fill in the lower 16 bytes of the MFCB for a file and initialise the current record byte.

Normally bytes 1 to 11 contain the filename and type in Ascii and the rest are zeroed. CP/M updates the other entries as it goes along before using this MFCB to update the disc directory if necessary when we have finished.

When we looked at the contents of the System Parameter Area (SPA) in a previous article I said that the area of memory from &5C to &7F made up the default FCB. CP/M reserves this for our use as an MFCB should we want to use it, and defaults to using it unless told differently.

Also the 128 bytes from &80 up to the top of the SPA and &FF are the default DMA buffer. A DMA, or Disc Memory Access, buffer is simply the area of memory which contains the record we want to write to disc, or where CP/M puts a record which we read from disc.

Again CP/M provides us with this default area which it uses unless told otherwise. However with both the default FCB and DMA buffer we are quite free to use different blocks of memory if we want.

When we ask the CCP to load a program for us we also often type command line parameters, usually one or more other filenames — for example:

```
A>filecopy ourtext.doc
```

These command parameters — OURTEXT.DOC, in this example — are also referred to as the command tail. The CCP then loads the file we have requested into memory from disc, but before starting to execute it the CCP takes the command tail and copies it into the DMA buffer at addresses &82 onwards.

&81 is always set to an Ascii space and the byte at address &80 contains the number of characters in the command tail, including the space at address &81.

If our program wants to read its command tail and we will be using the default DMA buffer for disc access it must either read and use the command tail, or copy it elsewhere for safekeeping, before we first access the disc. Otherwise it will be overwritten by the first disc record we read or write.

Obviously this does not apply if we are using our own DMA buffer and not the default one supplied for us.

At the same time as it copies the command tail into the DMA buffer the CCP also constructs an MFCB in the default FCB and &5C for any files which were specified in the tail.

So for our example above bytes &5D through &63 will be initialised to OURTEXT .DOC and the other bytes zeroed. If we have specified a second filename in the command

tail, for example:

```
A>compare file1.com b:file2.com
```

then the CCP also sets up the first 16 bytes of an MFCB with byte 0 set to 2 to indicate drive B FILE2 COM in the next 11 bytes and the remaining four set to zero.

It puts these 16 bytes into the top 16 bytes of the first FCB, which are used later by CP/M to hold the block allocation data for the file. Figure III shows exactly how the default FCB is laid out for this example.

Again if we expect to use this second filename in our program we must move these 16 bytes elsewhere to our own MFCB for the second file or they will be overwritten and lost when we start accessing the disc.

Also if we don't want to use the default FCB at all it is up to us to copy the contents to somewhere useful if we want them.

That's enough theory for this month. Let's look at the most simple operation possible, a sequential read, and see how it's done.

First we must tell CP/M that we want to work on a particular file. To do this we call function 15 — open file — with register pair DE containing the address of the MFCB for the file, which we must previously have set up.

When the function returns register A contains the directory code. This can be either 0,1, 2 or 3 if CP/M found the file in the directory successfully, and is 255 (&FF) if an error occurred. That is, the file whose name we put in our MFCB could not be found on the disc.

If the operation was successful bytes 16 through 31 of our MFCB are filled with directory information for the file.

Once we have opened the file in this way we can start to read it. If we call function 20 (read sequential) again with DE pointing to the MFCB for the file, CP/M will return the first record of the file in the default DMA buffer starting at address &80.

An error code is returned in register A. This is 0 if the read was successful, non-zero if we have reached the end of file and no more data exists. To read the next record we just call function 20 again.

'Note that it is entirely up to us to ensure that we do not create a file with the same name as an existing one. If we do this by mistake CP/M will get very confused and unpleasant things will start to happen.'

Every time we do this the next record of the file will be loaded into the DMA buffer, overwriting the previous one. If we want to keep all the records we load we must copy the contents of the buffer elsewhere each time, before we load the next record.

Alternatively to save ourselves the work of copying the contents of the buffer each time we could make CP/M load every record into a different area of memory, each 128 bytes long, so that we would get a contiguous copy of the file in memory.

We do this by making a call to function 26 (set DMA) before each read. All this requires is the start address of the area of memory we want to use as the DMA buffer in register pair DE.

Therefore if we increase this address by 128 before each call to read then CP/M will load the file into memory, one record after another.

And that's all there is to reading a disc file. Writing sequentially is only slightly more complicated.

Firstly if the file we wish to write to already exists and we wish to completely overwrite it with new information we simply open the file with function 15 exactly as we did before for a read. Remember, however, that a sequential write cannot add information to the end of an existing file.

To do this we must write randomly, using a different set of function calls. We'll look at this more closely next month. The way we'll write data to the disc for the moment is into a completely new file which we will create.

To do this we call function 22 (make file) a instead of opening the file with function 15 as before. This is used in exactly the same way as the other.

We pass it the address of our MFCB in register pair DE, and CP/M creates a new directory entry on disc from it with the name of our new file in it. Register A contains the directory code on return as before.

Continues page 64

Metrocross

US Gold Cass, joystick and keys

Remember all the sports fads of the 60s and 70s? Well, I don't either, but the one that stands out most in my mind is the skateboard, the much touted successor to roller skates.

Well, how the mighty are fallen, and where can you find the skateboard today? Only in the new Metrocross game from US Gold, an arcade licence from Namco.

The object seems at first quite simple — skateboard your way across 24 similar levels of chequerboard floor riddled with potholes, a barrage of obstacles that hinder your progress, and grey no-go areas that zap your speed.

Initially you play the game on foot, dodging rolling coke cans, jumping over hurdles, avoiding pits that swallow you whole, until you leap on to your very first skateboard.

And all this in a race against the clock to reach the end of the first section.

Springboards are waiting to catapult you forward, and your skill can be gauged by just how well you can fly from one to the next.

Probe Software, responsible for the conversion, has done a reasonable job. It is let down slightly by jerky scrolling, but still maintains the fast overall speed of the arcade original.

After four hours play I almost managed to reach the end of the eighth level, but not before I'd been burnt alive, attacked by rampaging rats, and crashed into hurdles galore.

I found the only successful method of shaking off the rodent attackers was to take them down a hole in the floor with me, with losing time as the main penalty.

I was convinced that after level four the main obstacle to beating the game was the relentless time factor.

But complex floor designs, ever-increasing numbers of hazards, coupled with a touch of over confidence leads me to conclude that level eight would do better with four or five extra seconds to allow you to win.

Good use of spot sound effects, and a pleasant background tune make the gameplay absorbing, and I love the way the little fellow pants and gasps on finally reaching the bridge at the end of each level.

Victor Laszlo

Presentation 85%

Great joystick control, with a choice of one or two players.

Graphics 85%

Good version of the arcade original, let down slightly by the jerky scroll.

Sound 90%

Good sound effects and a snazzy tune.

Playability 90%

Couldn't fault the gameplay at all.

Addictive qualities 87%

Kept me busy for many hours.

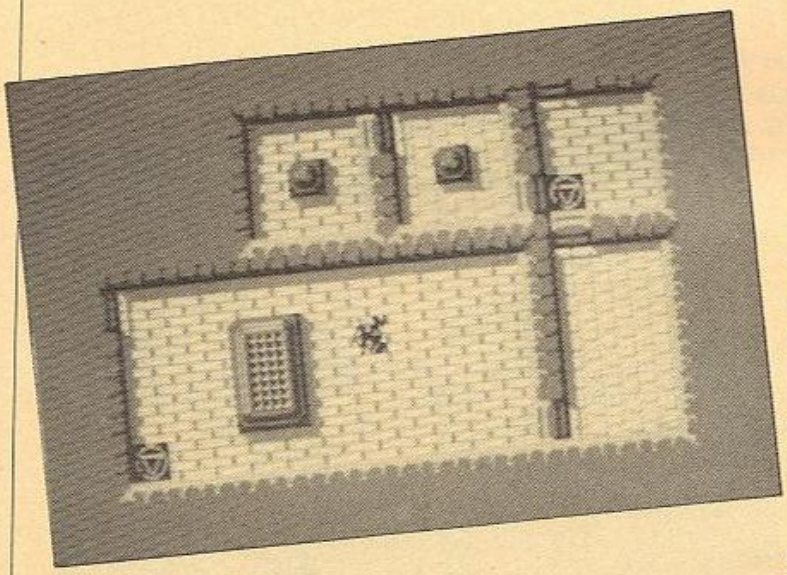
Value for money 85%

Worth every penny.

Overall 88%

A scrolling game with a real difference

Ranarama



Cass/disk, joystick and keys

One day Mervyn decides to turn himself into the most handsome sorcerer's apprentice in town with disastrous results!

Now he's a warty frog and only a magic potion can restore him to his former self.

This is not going to be as easy as you think. Following an invasion of warlocks, Mervyn is trapped in a multi-level dungeon, patrolled by a variety of unfriendly creatures. Their job is to protect the warlocks and splat any foolish froggies that try to escape.

But it's the warlocks that pose the real threat. To have any chance of success, you must guide Mervyn and engage them in ritual combat. If he succeeds in defeating a warlock, he wins valuable runes which he can exchange for more powerful spells.

There are eight levels and 12 warlocks on each. With 50 to a 100 rooms per level (about five or so to a screen) you have plenty to explore. You are given an aerial view of the action, with each room lighting up as you enter and remaining illuminated till the end of the game.

The nasties come in various guises and are successively more deadly. Most numerous initially are the dwarf-warriors who, despite their numbers, are fairly easily disposed of. A little more of a challenge is the Fire-Gollum, a ghost-like figure who can singe your whiskers (not t mention anything else).

There's also a host of other warlock minions which yo must either avoid or obliterate — serpent, ghouls, spider insects and gargoyles.

Some are more deadly than others but all will sap you strength should they catch up with you. Along the wa you'll also encounter four types of weapon: Spinnin knives, munching mouths, rotating balls (ooch! aargh! and a whirling sphere.

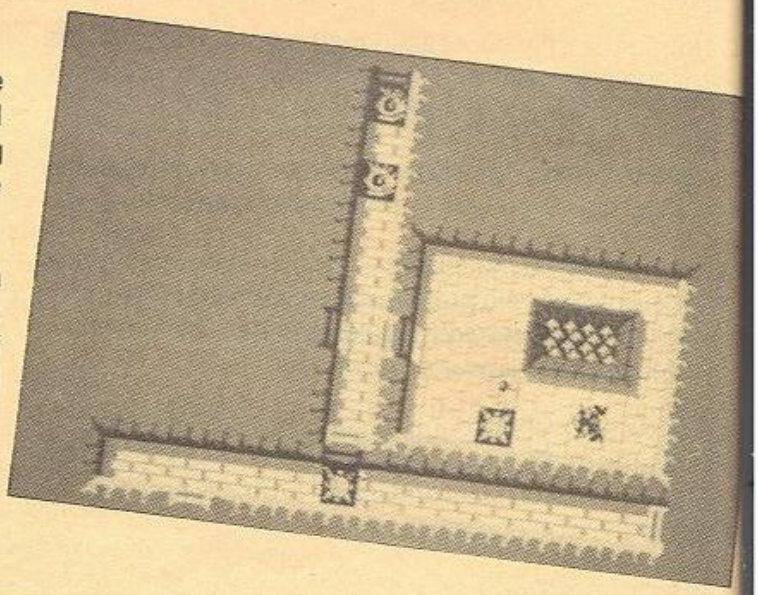
These are produced by the weapon generators whic you'll find in most rooms. Destroy them and the roo becomes a lot safer, and there's also the bonus points.

Fortunately, you'll also encounter energy crystal which look like rotating diamonds. These will give yo added strength and keep you going in your quest for th magic potion.

It is essential to accumulate spells, as without thes you cannot proceed to higher levels or expect to live ver long. You start the game relatively weak in four varieties o spell: Power, Offence, Defence and Effect.

Having two Power spells — Psychic and Mortal — means you have two lives. Should you lose power you fall from the psychic to the mortal level, and should yo lose too much energy, from there to oblivion.

Neither your Offence spell (Zap) nor your Defence spe are particularly strong, so it's vital that you exchange an



acquired runes for stronger spells. Though also weak, your initial Effect spell (Find) enables you to locate hidden doors which can prove invaluable when you're cornered in the dungeon's maze.

On the floor of the dungeon there are four kinds of magical symbol (or glyph) which can be activated if you step on them and press fire. Proper use of these is advisable if you are to complete the game.

The Glyph of Sorcery allows you to use available spells and cast them. The Glyph of Power enables you to destroy any nasties in the vicinity, but it has no effect on weapons.

By using the Glyph of Travel you can teleport between levels at whim. Once you have activated this facility the screen changes and you are given a vertical demonstration of movement from one level to the next.

The Glyph of Seeing reveals how much of a level you have covered. Used with a more powerful Effect spell, it is possible to locate any remaining warlocks you have not yet encountered.

Once you've discovered a warlock you suddenly find yourself in a sub-game. All you have to do is unscramble the jumbled letters of Ranarma and put them in their proper order. At first, this is quite easy — once you get the knack — but on later levels the time restriction becomes increasingly more demanding.

If you succeed you'll find yourself back in the dungeon surrounded by spinning Rs. You must gather these up as quickly as you can — they don't hang round for very long.

Programmed by Steve Turner, who wrote Dragontorc and more recently Quazatron, Ranarama is both addictive and immensely playable. Essentially a shoot'em up, there is also an element of strategy which gives the game an exciting new dimension. For its playability alone, this is definitely worth adding to your collection.

Tony Flanagan

Presentation 80%

Comprehensive playing instructions, though how you acquire and use spells should have been made clearer.

Graphics 85%

The sprites are nicely done, and good use is made of colour to make each level interesting.

Sound 80%

A good variety of spot effects and lively jingles.

Addictive qualities 90%

Difficult to turn off.

Value for money 90%

A very enjoyable game with a great deal to achieve and explore.

Overall 90%

As a shoot-em-up with elements of strategy this game should give you a few sleepless nights.

Thrust 2

Firebird Software Cassette, joystick and keys

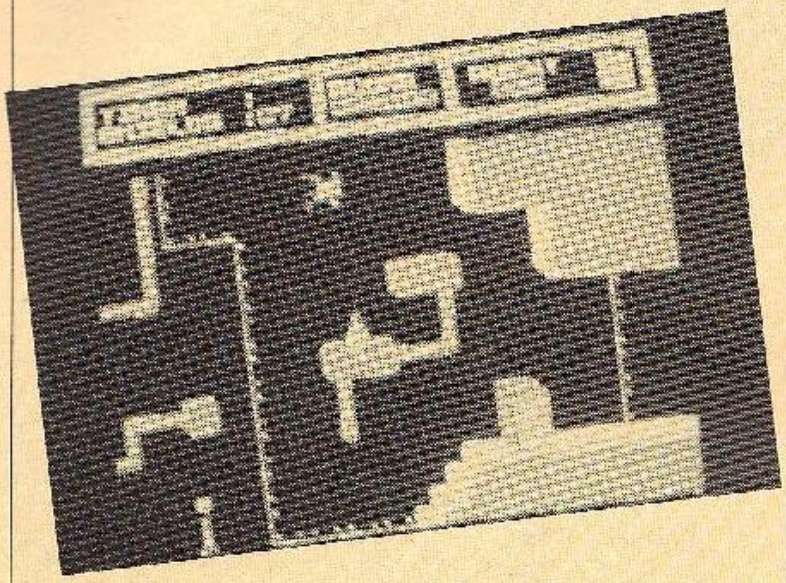
The resistance again needs your help in their battle against the Galactic Empire.

They have captured a small artificial planet as a forward base for the next offensive.

Unfortunately it is still under construction, and as yet has no atmosphere. Your task is to find and retrieve the 16 sections of the atmospheric processor, so making the planet habitable for resistance troops.

The original Thrust involved flying a simple dart-shaped spaceship through long, winding tunnels in search of drive units. Every time you escape the planet's surface, with a drive unit in tow, you were transported to the next planet and a more testing tunnel system.

In some ways Thrust 2 is similar. Most of the action occurs underground and the 16 sections of the atmospheric processor are towed behind the ship using a tractor beam.



You can rotate and thrust the ship as before, but the lasers have gone. This is a great disappointment to all zap and blast fans.

You can use a joystick though I found the keyboard gave the best control, and the keys are user-definable.

The first difference you will notice is the replacement of the simple line drawings with very smart, high-res multi-coloured characters.

Attention to detail is demonstrated beautifully in the way the highlighting of the ship alters as its position changes relative to the sun.

It's a great pity that this effort is marred by the appalling horizontal and vertical scrolling when you near a screen boundary.

The second difference is in the gameplay. In Thrust there was a single route you could follow through a tunnel, but in the sequel the tunnel is replaced by a giant maze of a single planet.

Gone are the deadly security lasers, and in their pla are the planet's defence droids which roam the ma following specific routes.

Thrust 2 is a very good game and were it not for t intensely irritating scrolling action it would have been great one. This said, it is still worth anybody's money.

Jon Revis

Presentation 84%

User-definable keys and a demo mode.

Graphics 85%

Highly detailed and colourful but let down by poor scrolling.

Sound 84%

Good sound effects.

Playability 80%

Responds well when the screen is not scrolling.

Addictive qualities 82%

A moments lack of concentration and you're an ex-pilot!

Value for money 89%

Excellent.

Overall 85%

A good game which could have been great.

Army Moves

Imagine, Cass/disk, joystick and keys

Information which could affect the outcome of the war is being held at the enemy HQ.

An all-out attack is far too risky, so the Special Operations Corps (SOC) has been called in to infiltrate the enemy base and obtain the secret plans.

As you are their best operative — well, that's what they told you — you have been volunteered for the assignment.

Stage one is set on a mock-up of the Forth Bridg Racing along in your Army issue jeep you are buzzed helicopter gunships. Ground-to-air missiles are very effe tive against these aerial attackers, but make sure you f the first shot, or you may not get a second chance.

Just when you think you've got the measure of t

helicopters, the ground forces arrive. Enemy jeeps scream towards you, and unless you act quickly you'll be rammed.

Two courses of action are open: Bang the joystick skywards at the right moment and you'll jump over the jeep; get it wrong and you won't.

A far better manoeuvre is to loose a ground-to-ground missile. It's more effective, but takes some handling.

The terrain is against you too. Saboteurs have blown holes in the bridge so your jumping skills will be severely tested.

You have five lives for your foray through stage one, which may seem generous but isn't. Apart from taking a great deal of mastering, the game returns you to the start every time you lose a life.

This means you must have a clean run through the complete stage if you are to progress to level two! Those lucky enough to do so will be awarded a further nine lives.

Screeching to a halt, you leap from your jeep and climb aboard a helicopter — your transport for the next three stages.

The final three stages are tackled on foot as with a hefty supply of grenades and bullets you set out into the swamp. If you can avoid the guerrillas and their grenades, you'll make it to the barracks. Now find the bunker, get the plans and head for the hills.

Army Moves is not the easiest of games to play, but once you've mastered it you'll love it!

Nev Astly

Presentation 84%
Comprehensive opening menu gives a choice of joystick or keyboard and controls sound effects

Graphics 88%
Smoothly animated and colourful.

Sound 86%
Can be played with music or sound effects only.

Playability 85%
Simultaneous use of joystick and keyboard can be tricky.

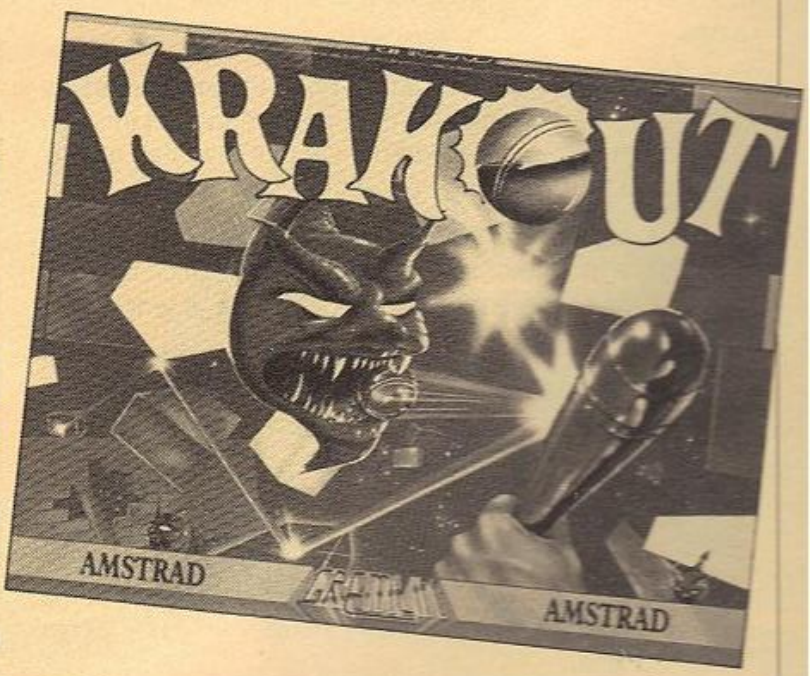
Addictive qualities 88%
Difficult at first, but satisfying once mastered.

Value for money 87%
Seven different stages which should keep you amused for a long time.

Overall 88%
A good all-rounder.

Krakout

Gremlin Graphics cassette/disk, joystick and keys



Not to be outdone by CRL's Ball Breaker and Imagine's Arkanoid, Gremlin Graphics has released its own Breakout game.

Krakout is a bang up-to-date version of that brick-busting rave from the grave.

It is played in a three-sided arena with an open end which you defend. In the arena are numerous coloured bricks, and their pattern changes as you progress through the levels.

You control the vertical movement of a small bat, and using a ball you must destroy every brick in the arena before attempting the next level.

Each time you fail to stop the ball leaving the arena you lose one of your three lives.

Having battled through to level nine, I can say with some certainty that there are only three different types of brick.

The first, in a variety of colours, is destroyed with a single blow; the second is grey and requires two direct hits before disintegrating.

The third looks very similar to the grey one, except for an occasional glint in the light. This type is indestructible and is there purely for nuisance value.

Variation is provided by rollover bricks — indistinguishable from any other until struck with the ball, when they rotate to reveal a letter or symbol. When hit a second time they will have an effect upon the bat, the ball, or some other aspect of the game.

These special bricks have a limited lifespan and disintegrate if you take too long to guide the ball in their direction. If you strike a second rollover brick its effects will replace the ones currently in operation.

Bricks bearing the letter G make the ball stick to your bat until you press the fire button.

Having caught the ball in this way you can carry it to the ideal position and fire it through a gap in the wall.

Bomb bricks cause substantial damage to the surrounding area when struck by the ball.

The shield bricks are a new idea, causing the arena to become enclosed by a fourth wall. Just sit back and watch the ball do all the work — a bell sounds before the effects wear off and the wall disappears.

There are bricks which arm your bat with a single missile. The thought of having a solitary one to play with didn't impress me a great deal — until I fired one of the things.

There is nothing which halts one of these beauties as they clear a path from one side of the arena right through to the other — real fire power! It's a pity that they are so rarely available.

The pretty flashing brick has the effect of slightly slowing the movement of the ball, while the brick labelled x2 doubles your rate of scoring, for a while at least.

A second bat is provided when you hit the double-bat brick, and it appears in front of the first. This may seem a pointless feature — but wait until you meet the wasps.

The X brick provides an extra life and is well worth having if you intend scoring more than a few thousand points.

Finally you have the E brick, which causes your bat to expand, allowing you a greater margin for error when hitting the ball.

Constantly crawling out of the brickwork are alien li forms. The heads of frowning gremlins cause nothir more than the occasional deflection and can be destroye with bat or ball.

A spaceship makes increasingly frequent appea ances as you move up through the levels, but I'm not su what effect it has as I have usually blasted it before it's g too close.

One creature which must be avoided, or destroyed, the wasp which flies in a straight line from the back of t arena to the front.

Contact between it and your bat results in paralysis the controls and the inevitable loss of a life. I never got chance to try it myself, but I suspect the double bat is some use in these situations.

The game options menu is very comprehensive. As right-handed player I prefer to have the bat on the rig side of the screen but the lefties among you can rever this set-up.

The bat can be controlled using one of two modes operation. In Inertia Mode the bat's initial rate of mov ment is slow, but speeds up if you keep the key/joysti held down.

The second option is Fast/Slow Mode, the bat tra elling slowly unless the fire button is depressed, when i rate of travel increases dramatically.

Once again it has been proved that the oldest ar simplest of ideas can still be modified and polished produce a stunning arcade game such as Krakout.

James Riddell

Presentation 89%
Comprehensive options menu.
Graphics 89%
Well detailed and colourful.
Sound 85%
Nice title tune and good sound effects.
Playability 89%
Response of controls can be tailored to your requirements.
Addictive qualities 94%
You never know what's going to happen next!
Value for money 90%
Worth every penny.
Overall 92%
Yet another excellent breakout game.

Little computer people

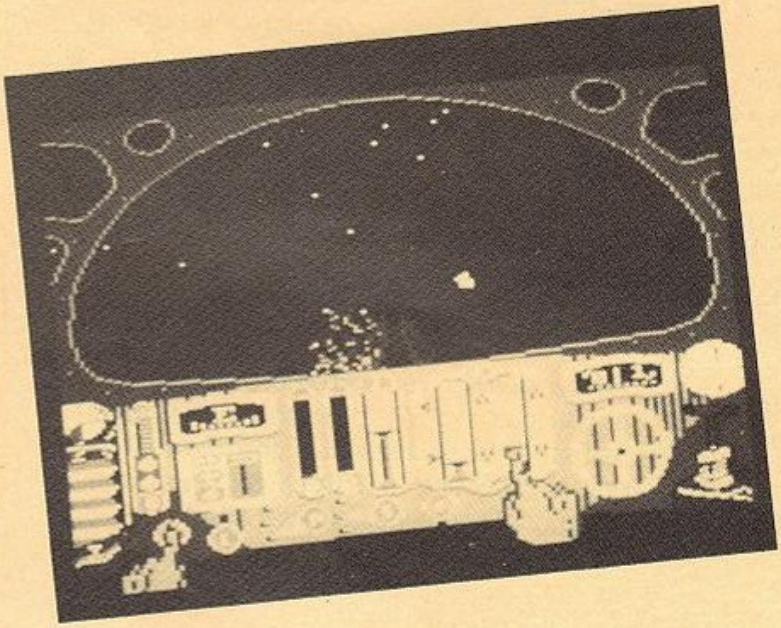
(Activision)

Shame on you Kenny Moorhead of Seaforth. This monster has committed mass murder. Thousands of LCPs across the nation are quivering in their armchair and dying just at the sight of the poke below. Type it in and your LCP will be evicted and a new one will move in.

This is terrible, horrifying and downright nasty. In fact I think that I won't even print the poke Kenny sent in. Instead we have a different one that will do lots of nice things to your LCP (nudge nudge!), such as wash the dog (wink wink!) and make the dinner (manic gibber!!). Just type it is as shown below, insert your LCP disc and type RUN.

```
  1 REM LCP
  2  REM by Kenny Moorhead
  3 REM CWTA/CPC
 20 MEMORY &1FFF
 30 LOAD "BOOT"
 40 POKE &8020,&FB
 50 POKE &8021,&C9
 60 POKE &8000
 70 POKE &20C3,0
 80 POKE &251A,0
 90 POKE &2DD2,0
100 CALL &2000
```

Terror of the deep



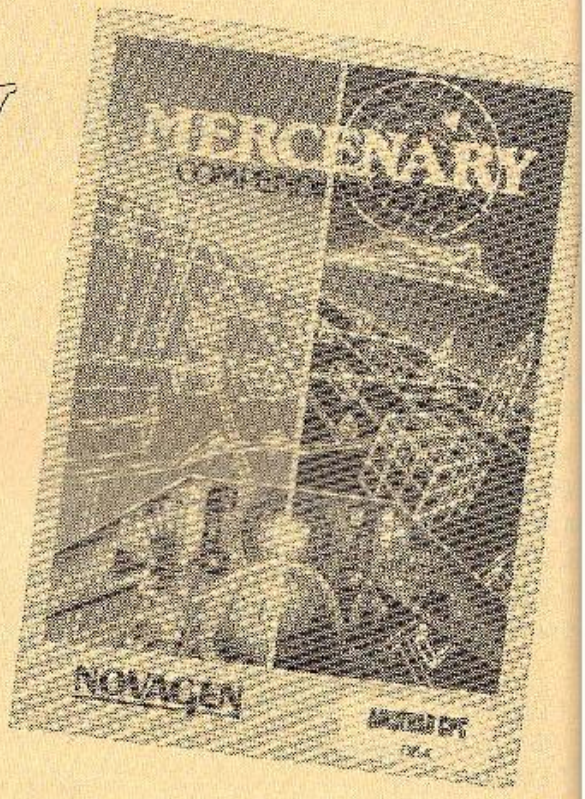
(Mirrorsoft)

Just to get you started, here are a few tips from my own Tipster's File.

As soon as the main game begings, start moving forward to cut the number of creatures clinging to your diving bell. When you hear a squelching sound, start rotating to find which window a critter has hung on to, then fry it with the electrified hull.

Shoot the frog-like creatures as quickly as possible, as they will eventuall cling to the diving bell. To find the crystal, look at the direction in which the fish are swimming, then travel in the opposite direction.

Mercenary



(Novagen)

Over the past few hours I have been trying to complete Mercenary with as many credits as possible. Budding Mercenaries among you may like to know that a sum of more than two million credits can be collected by giving the Palyars and Mechanoids what they want. There is even a way to give an object to both parties, so look out for big profits.

The first problem is getting a better ship. The obvious solution that can be gained from the manual is to find a Concorde or Cheese. But the better way is to collect a power amp because once this is attached to your ship you will have completed a sequence of events that allows you to complete the game.

Take a training flight to LOCREF 81-35 to collect the bar of gold and the key. Fly back to LOCREF 09-06 and enter the complex.

A search here will reveal a second key, a large box, sights, energy crystal, medial supplies and catering provisions. Go back to the hangar and go out of the North door.

Turn left and enter the first door on the right. The power amp should now be in view. Take it and return to your dart. If, while you are searching this complex, make sure you don't go along a very long corridor. This corridor leads to another hangar at 09-05 and can be avoided at this stage of the game. Leave the hangar and fly up to the Palyars' colony craft. Descend to its second level and collect the kitchen sink. This will enable you to pick up any object no matter how heavy it is.

While you are in the colony craft you may wish to deposit some of the objects you have collected, for large sums of money. Fly back to location 09-06 and go out of the hangar door on the east wall. Turn left and you will see a web. Pick it up and get rid of all your keys. The web is a cheat key and will open any door.

Fly to location 09-05 and enter the hangar. Inside will be two traingular-shaped doors. Going through the one on the west wall will reveal a photon emitter. Collect it and leave the hangar. Fly to location **-** which is the Targ star base. Inside you will find the Novadrive and the Winchester.

Take them and return to the room inside this complex which has three teleport doors. Travel through the one in the center and you will be transported to a new underground complex.

A search here should put you in a room with a sk crossbones on the floor. Go straight in. Voilà, one in lar ship for you to steal. Board it and press E. Wher press the Y key, do so. Congratulations on your from Targ.

There is another way to escape which involves communications room, but I will leave you to disco to do this. For a little fun, try driving a hover ca Palyars' colony craft.

I am sorry that I can't print the map, but Bruce atNovagen made me promise. If you do want maps, but the Targ survival kit. Even if you know finish the game, the kit is still a worth addition.

Hyperbow

(Mastertronic)

Always position yourself before accelerat as you may lose control. Hit the puck at ab half speed for best results. For some stra reason shooting the puck towards y home goal sometines slows it down.

Arkanoid

(Image)

Never go for bonuses if it means losing a ball. Always (if possible) go for the Breakout — very useful on those difficult levels.

On the easy levels, don't overuse the blaster because you'll miss out on the bonus capsules. Use it instead to destroy individual blocks. This way you will not lose any valuable extra lives, and so on.

Level tips

1 Get the lasers and save them for the end of the level. Only get more bonuses if they are extra lives or breakouts.

2 Try to knock out the red block on the right-hand side and send the ball through the hole. Lasers make this level easier.

3 Try to collect a Slow bonus to give you more of a chance of finishing this screen and collect as many other bonuses as possible.

4 Shoot the ball up the right-hand side, but watch for its quick return. Look out for Lasers.

5 This is a very difficult level without the laser so go for the Collect bonus. Again, shoot the ball up the right-hand side.

A word on pokes

(Tricks and Tips)

So many people have asked "How do you poke a game?", that I just had to answer them. First, learn machine code! This is not as daunting as you may think. There are plenty of books on the subject and most are very helpful. It should take you no more than two weeks to pick up enough to start some decent hacking.

There are so many different loading routines for Amstrad games and they are going to be your first problem. Try to intercept them and use them to load the program so that control is returned to your machine code monitor. The biggest problem is stopping the monitor from being over written. Once the code is in the machine and the monitor is working happily, disassemble the code and try to find pieces of machine code that look like they handle a lives routine. For example:

```
LD A,(xxxx)
DEC A
CP 1
JR Z,yyyy
```

and insert a value of 201 at the location which stores instruction JR Z,yyyy.

STARGLIDER

(Rainbird)

There were some minor niggles about the Starglider pokes I printed. The real problem is not with the program, but that some CPCs cannot handle a change of HIMEM without a disc/tape buffer being installed first.

If you have such a machine don't despair, simply type in this little addition to the main poke and all will be fine. Remember to have the original program in memory before typing it in, and also set the write protect tab to OFF(UP):

```
145 OPENOUT "D"
155 CLOSEOUT
```

Chuckie egg II

(A'n'F)

When you get to the first junction go down into the ice and milk kingdom to fill the vat (a full sign will appear under the milk). Make your way to the cocoa and fill that up with the cocoa found on the green screens.

Travel through the black screens and drop on to the railway. To stay there you should keep your finger down on the left or right button but keep will clear of the blue train. Go down a stairway which is defended by a rat. Then travel downwards, but not into Despatch. You should soon reach the generator after dodging the drops of water. Jump to flick the switch to the right when a red light will come on. Go up the stairs to collect the stairway. This will enable you to reach the Sugar Kingdom. Fill the sugar vat. When you have completed all the above problems go to the red brick toy kingdom and drop all the toy parts into the toy machine.

All lifts except the main one should now work. The completed toy should be a motorbike. Travel to the top of the railway and climb up the ladder you came down originally. Keep going up until you reach the screens with piping on them.

Go left and drop the toy where instructed. Collect the egg and return to the despatch screen. Jump in the van and drive away.

Do the whole thing again another eight times and you will have completed the game.

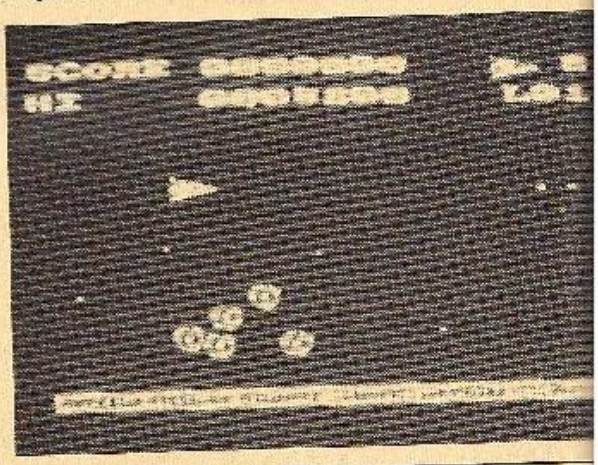
NEMESIS

(Konami)

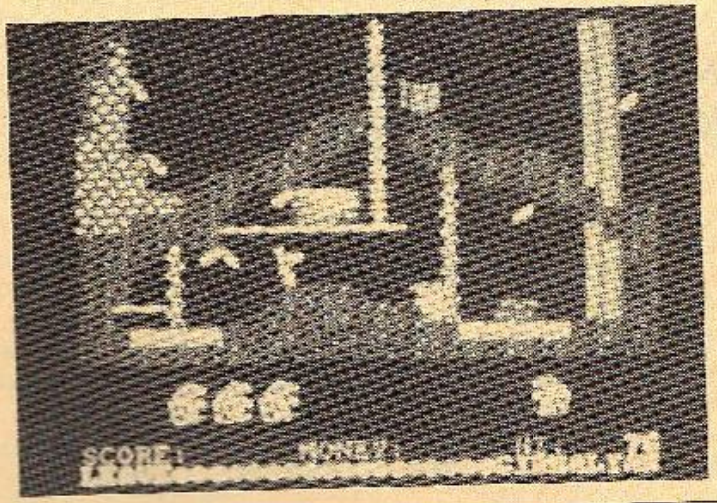
Here's some tips for a great coin-op hit. On the firs wave of aliens, alternate between the bottom-left c the screen. Shoot all that come up from that poir (top or bottom) and collect the pods. Switch to th laser as soon as you can. Try to equip your ship wit a laser, a missile chamber, as many option pods a possible and finally a shield.

When flying over the pink landscape, dodg around near the top of the screen, shooting rapidl When a small yellow blob zooms on screen, bom it and pick up the pod. Try to stay near the top of th screen to avoid the mountains.

If you're simply trying to complete the game a are not trying for a high score, fly low at the far rig of the screen. When a mountain comes into vie dodge and return to the bottom of the screen.



Auf Wiedersehen Mont



(Gremlin Graphics)

Try these infinite lives poke for this brilliant gam

Just type in the line below (remember to ins second comma, it is very important) and the mach return with Syntax Error. Fear not. Load the ga normal and pick up the machine gun on the first scre Monty will become a daring muscle mole, invin almost everything.

Da pok'iz (German speak) POKE 800,1,

There's room for you, too

Go adventuring with Gandalf

Is it my imagination, or is there a shortage of quality adventures at the moment? Looking back to this time last year, we had most of the Infocom games being released (but little else) and this makes me wonder if spring is the time when adventure programmers take their holidays.

New adventures are due and some might even be out by the time you read this.

Infocom have brought out Moonmist, and Rainbird are rumoured to be releasing something exciting soon — as good as if not better than, The Pawn. Watch this space for more details.

From the Infocom range I have only seen Hitchhikers, Leather Goddesses, Planetfall and Zork, and I would very much like to see the rest.

Others that come to mind include The Wild Bunch, Subsunk, Planet of Death, Inca Curse, Espionage Island, Shrewsbury Key, Spy Trek, Hot Runestone, Aftershock, Ship of Doom, Theseus parts I & II, Nythyd parts I & II , The Lost Phirious part II, Seas of Blood, Twice Shy, Miami Mice, The Hermitage, Teacher Trouble, Neccis Dome, Warlock, The Vera Cruz Affair, The Fantastic Four, Sphinx Adventure, Spellbound, Very Big Cave Adventure and Kentilla.

As you can see from my list of unseen adventures, not all software houses send their games in for review. This does them a disservice and doesn't help me much either.

I have been told that one of the reasons that my reviews tend to appear later than anyone else's. This is generally true, but I don't believe adventures can be reviewed in the same way as arcade games. They take time to play, and to give a fair review I hope to get near to completing one before I give an opinion of it.

This isn't always possible, as regular readers will know — even we so-called experts sometimes get stuck.

You can be assured of one thing though when you read my reviews: I will have spent about three weeks playing a game before I write a single word about it. This way you get the fairest review I can offer.

Apparently many excellent adventures fail to get the publicity they deserve because the writers feel they won't be treated fairly alongside the larger software houses.

One of my regulars, Pat Winstanley, recently brought this fact to my attention. Frankly, I am appalled. The thought that a wealth of talent exists and is not being brought to other adventurers because of a mistaken belief, is a tragedy. Let me assure everyone, I review an adventure on its merits, nothing else.

Now to those teething problems. Jonathon Ketley asks for help with Bored of the Rings. To get out of the willow tree, cry for help.

Can you keep going South from the black rider location without hiding? No.

Are there any other objects you can collect apart from the ale before being suffocated by the tree? Yes, wait at Fag End for the ring.

Are there any useful things to do in the big party tent before Bilbo disappears? No.

What is the sceptical? I assume you mean the program at the end of the tape/disk, and I'm afraid you'll have to complete the game before you see it. It's a send up of teletext, and very funny.

Julia Addison and Jane Darby are having problems with Mordon's Quest, but most of them can be solved by reading Pat Winstanley's solution (page 42). The aq-

Reader's Hall of Fame
Imagination — A solution by Sandra Sharkey

Get disk, insert disk, type one, examine time lord, pull lever, view, press switch, open hatch, S, S, get bucket, E, examine bed, get cord, E, get map, pinch arm, type two, N, examine bull, W, N, W, W, enter tower, examine woman, give map.

Pinch arm, type four, out, get spade, dig earth, get gloves, enter plane, aim gun at bull, fire gun, out, get paint, paint cord, pinch arm, type three, E, E, E, E, get poker, examine poker, fill bucket with coals, pinch arm.

Type two, N, E, N, drop bucket, get tuft, S, W, W, examine cow, feed cow, milk cow, get icicle, E, E, give icicle, get yoyo, examine yoyo, pinch arm, type three, E, E, up, N, E, N, E, examine pit, play yoyo, pinch arm, type two, N, enter citadel, down, unlock cell, get bread, pinch arm.

Type three, E, E, up, N, give bread, pinch arm, type one, pull lever, view, press switch, open hatch, S, S, E, E, N, press button, get chips, pinch arm, type four, out, drop chips, E, enter airbase, down, get pickaxe, pinch arm, type three, E, E, up, N, E, N, tiptoe bridge, get oilcan, smash cracks, fill oilcan, pinch arm, type two, N, enter citadel, up, read plaque, down, examine font, pinch arm.

Type four, oil gun, fire gun, pinch arm, type two, N, enter citadel, drink elixir, pinch arm, type three, E, E, up, N, E, N, tiptoe bridge, enter mole.

Pull lever, out, get ramboard, pinch arm, type one, pull lever, view, open hatch, S, E, insert ramboard in console, W, N, enter pod, view, press switch, grab spacesuit, open hatch, get spacesuit, open hatch ... Adventure completed.

ualung can be filled nine moves from the location it is found in; the cigar is in the future and trying to take the pyrites does cause an avalanche unless you break them up first.

Yes, Julia, Mordon is meant to appear the way he does. Simply say yes to him and get the torch and transporter from the cupboard. The newspaper is not a red herring, though I think the headline found on it is.

Julia has problems with two further adventures. In Message from Andromeda she has been trapped by a clinging vine in the botanic garden. Cut the view with the knife found in the workshop to get free.

In Return to Eden, to avoid being blasted by the Snowball go into the storeroom, wear the radsuit and watch and get the compass and geiger counter.

Then leave the ship, go East, dig and keep going do until you find an underground cave complex. Then East, South and wait. When the coast is clear you come out and start exploring.

Allan Fell has written in with an excellent map and h on problems raised about Mordon's Quest in the Ma column.

He says that he has completed the game but doens't know what the blue directions are; also the d in the telescript shop is a red herring. There is no nee surrender the Cretan coins to get out of the labyri Answers to all the other problems mentioned can be fo in Pat Winstanley's solution.

Reader's Hall of Fame
Mordon's Quest — A solution by Pat Winstanley

Start in the master bedroom – Take blanket, S, W, N, take newspaper, E, N, climb drainpipe, N, S, W, S. Mordon appears and explains your quest. N, E, take transporter, take torch, light torch, N, N, N.

The jungle:

E, drop blanket, E, take tusks, take bamboo, NW, NW, take berries, NE, SE, E, take thorns, make blowpipe, take pygmy, drop bamboo, SW, E, give pygmy to plant, E, E, FROG, W, W, W, NE, N, E, S, E, E, N, E, S, SW, SE, S, E, NE, SE, SW, take dagger, SW, E, E, take metallic device, transport metallic device, W, W, NE, NE, NW, SW, W, N, NE, NW, N, W, sacrifice frog, take jade frog, transport jade frog, take gems, transport gems, N, SW, NW, NE, take piece of machine, transport piece of machine, SW, SE, go rubble, drop transporter, W, W, down, take iron pyurites, brak iron pyrites, take diamond, take pyrites, up, E, E, take transporter, transport diamond, down, E, N.

This area is now complete and there is no need for you to return. You are in the time machine and should answer the telephone when it rings. (Dial 1611 for a password). To your left is a plate which you can press to progress to other time zones, the one you arrive at being selected randomly.

There are two exits north: If you emerge on a shingle beach you are in the undersea zone but if you are in a chalk hollow you have arrived in the Roman area.

There are also two exits south: If you find yourself in a large cave, you have returned to the jungle area, but if you arrive in an ante-room then you have gone to the futuristic zone.

If you arrive in an area that you do not wish to visit, return to the time machine and keep pressing the plate until you get to the one you want.

The undersea are:

From the single beach – N, N, NW, climb in to boat, down, take aqualung, N, N, N, SE, up, SE, down, N, E, fill aqualung, W, S, NW, take black pearl, NW, N, N, N, off lamp, N, N, on lamp.

Take glowing metal object, transport glowing metal ob transport black pearl, E, take doubloons, transport doubloo E, up, wait, wait, E, N, N, give newspaper to spiderman, (he you some spray paint), take remote control, S, S, W, dow W, W, off lamp, S, S, on lamp, S, S, S, S, S, S, up, S, S, aqualung, S and back into the time machine,.

The area is now complete and also need not be revisited

The adventure development area: From the time machin while carrying the remote control and iron pyrites, push b Give pyrites to jester, W, E, (You are given a piece machine), E, S, up, transport piece.

The futuristic era: Press the plate until you emerge in the room. E, take geiger counter, W, N, (push the plate un emerge in the Roman era), NE, E, move churns, take transport ring, W, W, examine straw, take battery, E, S,

Push the plate until you emerge in the future again, E geiger counter, E, SE, SW, SW, spray paint, 8875, S device, transport device, N, NW, SW, S, S, S.

Take ingot, transport ingot, N, N, N, NW, W, W, N, E battery, W, S, E, E, touch plate, S, press 3, press 1, press 2 2, press 4, press 1, N, W, touch plate, S, S, S, touch plate, crystal orb, transport crystal orb, NW, NW, take cigar, NW perseverance, N.

Take unit, transport unit, S, SE, SE, SE, S, touch pla N, N, E, SE, NE, NW, NE, NW, W, W, S, SE, SE, take coins, transport roman coins, take cretan coins, transpor coins, NW, NW, N, N, (back to the time machine).

e) The Roman era: From the time machine (ensure you h cigar with you), press the plate until you emerge in the era, N, N, N, N, N, N, N.

Take sword, take shield, smoke cigar, transport laure S, W, W, N, NE, kill minotaur, skin minotaur, take machine, transport piece of machine. Adventure compl

Waving the flag to good effect

Part VIII of MIKE BIBBY's introduction to machine code

Do you remember what a flag is? Well, a flag is an indicator that lets us know the state of play in our programs. It's as vital a guide to us as radar to a pilot, or the pulse and heart rate to a doctor.

Despite their importance, flags are essentially quite simple. You can think of them as a primitive numeric variable that's only allowed to have two values — 1 when we say the flag is set, and 0 when we say it's clear.

Another way to think of it is as a single bit register, since a single bit can only take the values 0 or 1.

Depending on what's happening in a program, the micro sets or clears the various flags. We first met flags when we were adding numbers. Because the largest number we could store in a byte was 255, when we have two numbers together to give an answer larger than 255 we hit problems.

As we've seen, going past 255 in a byte — register or memory — is rather like going round the block on a car's odometer: You start again at zero.

So far as the A register (the register ADD and SUB work on) is concerned:

255 + 1 = 0
255 + 2 = 1
254 + 3 = 1

and so on. Much the same thing happens when you try to go below zero in a subtraction.

Try Programs I and II if you don't believe me. The first does:

254 + 3 (&FE + &03)

and the second:

3 – 4 (&03 – &04)

In both cases the answer is stored at &2FF8, the first byte of

address	hex code	mnemonics
3000	3E FE	LD A, &FE
3002	C6 03	ADD A,&03
3004	32 F8 2F	LD (&2FF8),A
3007	C9	RET

Program I

address	hex code	mnemonics
3000	3E 03	LD A, &03
3002	D6 04	SUB &03
3004	32 F8 2F	LD (&2FF8) ,A
3007	C9	RET

Program II

address	hex code	mnemonics
3000	3E ?1	LD A, ?1
3002	C6 ?2	ADD A, ?2
3004	32 F8 2F	LD (&2FF8),A
3007	D2 0F 30	JP NC,&300F
300A	3E 07	LD A,&07
300C	CD 5A BB	CALL CharOut
300F	C9	RET

Program III

Hexer's workspace.

Of course, the Z80 doesn't just ignore aberrant results when things go round the clock: It sets what's known as the Carry flag. That is, the Carry flag becomes 1. Conversely, if we haven't gone over (or under) the limit, the flag is cleared, to 0.

So, as far as the micro is concerned:

255 + 1 =	Carry flag set
254 + 1 = 255	Carry flag clear
0 - 0 = 0	Carry flag clear
3 - 4 = 255	Carry flag set

The carry flag lets us know when things go wrong!

Now there isn't any way that we as programmers can directly examine a particular flag, but the micro is clever enough to take the value of a flag into account.

As we've seen in earlier articles depending on whether the Carry flag is set or clear, we can jump to different parts of our programs with instructions such as:

```
JP C  ( opcode &DA )
```

which is JumP with Carry set and:

```
JP NC ( opcode &D2 )
```

which is JumP if the Carry flag is Not set.

Program III is a nice little bleeper! What it does is to add the two numbers ?1 and ?2. Of course, ?1, ?2 are only labels — you're meant to insert the two bytes you want to add here — memory locations &3001, &3003, respectively, if you want to POKE them into memory.

The first two lines of Program III do the addition. The ADD instruction will set or clear the Carry flag appropriately to warn us if we've exceeded 255.

```
LD (&2FF8) ,A
```

then puts the answer at the beginning of Hexer's workspace, for later examination.

Now hear this — LD instructions do not affect any flags ever. (There are only two minor exceptions to this, LD A,R and LD A,I which we're never likely to meet.) They leave the flags completely alone. This is a very useful piece of information to bear in mind.

So even after this LD, the state of the Carry will depend on the result of the ADD.

We then test this with:

```
JP NC, &300F
```

This checks to see if the Carry flag is set. If not — that is, if our answer didn't exceed the byte limit — the program jumps to memory location &300F, which, since it contains &C9, RET, simply ends the machine code routine.

Notice that for the first ti
we're jumping forward and
actually skip some code if Carr
not set.

If, however, Carry is set — t
is, we have exceeded 255 in
answer — we carry on directly w
the next bit of code after the
which is:

```
LD A,&07
CALL CharOut
```

which, I'm sure you'll recall, cau
a bleep. We then encounter RET

The outcome of all this is th
our program will bleep for su
that cause the Carry flag to be set
the ones that gave us "wrong" a
swers— yet let the other sums pa
without question.

All right, a warning bleep is
all that useful, but it does illustra
that we can detect theCarry fla
signal and act on it. In practice w
use this branching technique to
some more arithmetical work b
fore we get the right answer.

Experiment by changing th
values to be added — memory loc
tions &3001, &3003 — and s
whether you get the expected lea
or not.

Next replace our:

```
JP NC, &300F
```

with a

```
JP C, &300F
```

by changing the fourth line of Pr

gram III to:

```
3007 DA 0F 30 JP C, &300F
```

Notice how neatly these three bytes replace the old ones.

This time you jump when the

address	hex code	mnemonics
3000	3E ?1	LD A, ?1
3002	D6 ?2	SUB ?2
3004	32 F8 2F	LD (&2FF8),A
3007	D2 0F 30	JP NC,&300F
300A	3E 07	LD A, &07
300C	CD 5A BB	CALL CharOut
300F	C9	RET

Program IV

Carry is set. This only happens when an addition gives a bigger than byte-sized answer, so there's no beep for "out of limits" answers.

If the answer stays within bounds, and Carry is cleared, you don't take the jump, and the beep is heard.

So by changing just one byte, the opcode for JP NC — &D2 — to the opcode for JP C — &DA — we have completely reversed the program's effect.

However the main point of the demonstration is that when we ADD two single bytes, if the answer exceeds 255 the Carry flag is set to warn us.

You may be wondering why we call it the Carry flag. Well if you can remember that far back, at school you used to mark the tens and units columns when you did sums — and those columns were very important!

If you were doing the sum:

```
  t u
  16
+ 9
  —
  —
```

your thought processes went *9 add 6 is 15, which is too big for the units' column, so I'll put 5 units down and carry one ten to the tens' column.*

We work in much the same way in machine code. This time our "units' column" — a single byte — can hold up to the number 255, after which we have to carry, or add one, to a byte representing the second column of our sum.

Hence it's very useful that our flag is set to one automatically when we go past 255. Just when we need to carry one our flag is conveniently set to one, so we call it the Carry flag. We'll explain exactly how the whole process works later in the series, so don't worry if you don't follow it too well.

As we've mentioned, this flag is also set when the result of a calculation goes below zero. Program IV is a variant of Program III, adapted for subtraction.

Once again, ?1 and ?2 are merely labels — you're meant to put your own numbers here at memory locations &3001, &3003.

The program works in much the same way as Program III. If there's No Carry generated, the JP NC simply jumps to the terminating RET. If there is a Carry, the bleeping code is performed.

As you'll discover when you experiment, the bleep occurs when the second number is larger than the first. That is, when you try to subtract a number bigger than the one already in the A register.

Think about it. If the number you're taking away is bigger than the number you're taking away from, you must cross that important zero boundary, cycling round the clock again and triggering the carry flag.

The following shows what happens as we subtract increasingly greater numbers from 2, say:

```
2-0= 2   Carry clear
2-1= 1   Carry clear
2-2= 0   Carry clear
2-3= 255 Carry set
2-4= 254 Carry set
```

So the Carry flag is set when the second number is greater than the first. If you like, it's doing a sort of comparison, comparing the number in the A register with the one you're taking from it. If the latter is greater, the Carry flag is set.

You'd be amazed how often you actually want to compare bytes in machine code, but doing it as we did above — using SUB — isn't too useful. You see, although you can

use the Carry flag with SUB in this manner to do a rudimentary comparison between a byte and what's in the A register, the contents of the A register will almost certainly be

address	hex code	mnemonics
3000	3E ?1	LD A,?1
3002	FE ?2	CP ?2
3004	32 F8 2F	LD (&2FF8) ,A
3007	D2 OF 30	JP NC,&300F
300A	3E 07	LD A,&07
300C	CD 5A BB	CALL CharOut
300F	C9	RET

Program V

changed in the process. After all, you are taking a number away from it.

SUB n takes the number n from what is in the A register and stores the answer back in the A register. We represent this as:

A (- - - A-n

However the Z80 designers have provided us with a comparison instruction that gets around this problem — CP n. This takes the

address	hex code	mnemonics
3000	3E 20	LD A,&20
3002	CD 5A BB	CALL CharOut
3005	C6 01	ADD A,1
3007	D2 02 30	JP NC,&3002
300A	C9	RET

Program VI

number in the A register, subtracts n from it, sets the flags accordingly but does not put the answer back in A. It simply discards it, leaving the number in A unchanged.

So memory bytes and the registers we've met are unaffected by CP n — only the flags are set or cleared as appropriate. That is:

- If the number compared with A is greater than that in A, the Carry flag is set.

- If the number compared with A is less than or equal to A, the

address	hex code	mnemonics
3000	3E 1F	LD A,&1F
3002	C6 01	ADD A,1
3004	CD 5A BB	CALL CharOut
3007	FE FF	CP &FF
3009	DA 02 30	JP C,&3002
300C	C9	RET

Program VII

Carry flag is cleared.

Mathematically:

A < n ------> Carry set

A >= n ------> Carry cleared

The effect on the flags is the same as for SUB, but the number in A is unaltered.

Program V is essentially Program IV with the SUB replaced by CP. Again, insert your own n bers into memory locati &3001,&3003 and see if you predict the bleeps accurately.

Take a good look at &2FF8 a you've run it, though. This sh prove to you that the contents have indeed been unchanged by

To illustrate how we can use to effect, have a look at Program We've met it before — in fact it's first loop we met.

As you'll see, it prints out all characters with codes from &2 &FF by loading A with &20, pr ing the character with that A code, increasing what's in A by jumping back with a JP NC to p A once again, then increase it, so on...

Finally the character co

address	hex code	mnemonics
3000	3E 5A	LD A,&5A
3002	CD 5A BB	CALL CharOu
3005	D6 01	SUB 1
3007	FE 41	CP &41
3009	D2 02 30	JP NC,&3002
300C	C9	RET

Program VIII

sponding to &FF is printed and value in A increased by one, tak it "round the clock" to zero, sett the Carry flag and dropping ou the loop.

What we've done is to take vantage of the fact that Carry is

when we go round the clock. Alternatively, we can use CP to check for the last character of the loop, and it has some advantages as well. Program VII shows the idea.

Notice that this time we're adding one to the A register *before* we print it out. We then compare it directly with &FF, since this is the last character we want printing. This means we load the A register with &1F initially, which is immediately increased to &20 by the Add A,1.

Now until the A register gets to &FF the number we're comparing with A is greater than that in A, so the Carry flag is set, so we jump back to the top of the loop again with JP C.

Once the A register gets to &FF, the numbers we're comparing are equal and the Carry flag is cleared.

You might wonder why we didn't just add a CP 0 after the ADD A,1 of Program VI. Well, if you think about it, you can't go round the block by taking zero away from a number, so Carry will never be set.

The nice thing about using CP in loops is that we can easily vary the number we end with. For instance, if we just wanted to print up to Z — Ascii &5A — we'd change the CP &FF to CP &5A. And, if we wanted to start with A — Ascii &41 — we load our accumulator initially be replacing LD A,&1F with LD A,&40. Remember, we're going to add one to it straight away.

Another advantage of using CP to set Carry rather than just going round the clock is that it allows us to do things such as print out the alphabet backwards, as in Program VIII.

As a final example of the use of CP, let's use it to filter out a range of inputs. I'd better explain what I mean by that.

Often in a program you give people a choice of, say, five options from a menu. They respond by pressing a number in the range 1 to 5, corresponding to their choice. If they press a wrong key, it's ignored and the program waits until a valid choice is made.

So let's write a program that waits until a 1, 2, 3, 4 or 5 is input and then prints it out.

We'll use CharIn to input a key, and then check it's in range with two comparisons. Remember that CharIn puts the Ascii value of the character pressed in to the A register, so we'll be wanting to check for values in the range &31 to &35, the Ascii codes for 1 to 5.

In other words, the number in the A register must be greater than or equal to &31, our first comparison. It must also be less than &36, our second comparison.

Program IX puts this into practice.

address	hex code	mnemonics
3000	CD 18 BB	CALL CharIn
3003	FE 31	CP &31
3005	DA 00 30	JP C,&3000
3008	FE 36	CP &36
300A	D2 00 30	JP NC,&3000
300D	CD 5A BB	CALL CharOut
3010	C9	RET

Program IX

The CP &31 checks that the number in the A register is &31 or greater. If not, Carry is set and we jump back to the start to get another key since the character entered was "too low".

If we get past the check though, we then compare the number with&36. We need the number in the A register to be less than that, so this time we want Carry to be set. If it isn't, the key is "too high" so we jump back to get another key.

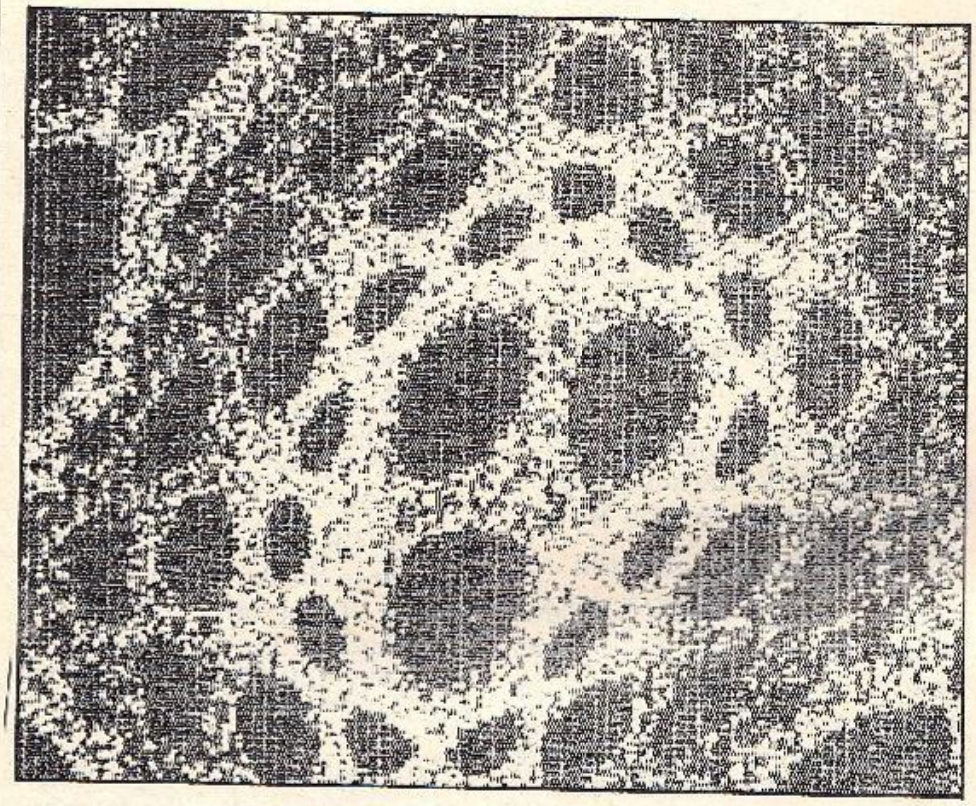
If, however, Carry is set, we simply continue. If the program has got this far the number in A must be in the range we want so we CALL CharOut and RET.

Once you've seen how it works, why not try to write it to accept only the upper case letters of the alphabet, &41 to &5A? It shouldn't be too hard.

Well that's all for now. There's lots more you can do with CP and loops in general, as we'll see next month.

10 LINERS

Ten liners is certainly proving popular — in fact we've been swamped with them. Here are two of the best. Keep them coming and remember to send a stamped adressed envelope if you want your tape or disc returned to you.



Aliens

by Alan Grosvenor

Next we have a fast action arcade game complete with space ship, laser bolts and little green aliens.

Your mission is to destroy 20 of the blighters as quickly as possible. Can you beat 55 seconds?

```
10 t=0:sec=TIME:INK 0,0:INK 1,15:INK
2,18:INK 3,6:PAPER 0:BORDER 0
20 MODE 1:FOR stars=1 TO 50:PLOT INT(
RND*(640)),INT(RND*(400)),1:NEXT star
s:j=INT(RND*(25))+1:x=8:y=12:FOR i=37
 TO x STEP -1.5:LOCATE i,j:PEN 2:PRIN
T CHR$(225);
30 IF t=20 THEN 80
40 IF INKEY(69)=0 AND y>1 THEN y=y-1:
LOCATE x,y+1:PRINT " ":LOCATE x,y:PEN
 3:PRINT CHR$(246):SOUND 1,y*5,10,3
50 IF INKEY(71)=0 AND y<24 THEN y=y+1
:LOCATE x,y-1:PRINT " ":LOCATE x,y:PEN
 3:PRINT CHR$(246):SOUND 1,y*5,10,3
60 IF INKEY(21)=32 THEN FOR a=x+1 TO
i-1:LOCATE a,y:PEN 1:PRINT " ";CHR$(2
6):NEXT:LOCATE a,y:PRINT " ":IF y=j T
HEN SOUND 1,50:LOCATE i,j:PRINT CHR$(
238):FOR c=1 TO 99:NEXT:LOCATE i,j:PR
INT CHR$(227):FOR c=1 TO 99:NEXT:LOCA
TE i,j:PRINT " ":t=t+1:GOTO 20
70 LOCATE x,y:PEN 3:PRINT CHR$(246):N
EXT i:SOUND 1,998:GOTO 20
80 PEN 3:sec=INT((TIME-sec)/300):LOCA
TE 1,1:PRINT"You destroyed 20 aliens"
:PRINT:PRINT"Well done":PRINT:PRINT"I
t took you";sec;"seconds":PRINT:PRINT
"Press 'P' to play again":WHILE INKEY
(27)<>0:WEND:RUN
```

Fractual Nebulae

by Stan Grapley

This intriguing progra
grows glorious technicolour fr
tal nebulae. The pattern starts
the centre of the screen a
grows outwards in spurts. Ju
when you think it's finished
new layer is formed at the ed
Try altering the variables in li
30 for different effects.

```
10 BORDER 0:INK 0,0:INK 1,22
20 MODE 0:ORIGIN 320,200
30 a=4:b=3:c=7
40 WHILE 1
50 FOR col=1 TO 15
60 FOR i=1 TO 200
70 xx=y-SGN(x)*SQR(ABS(b*x-c))
80 yy=a-x:x=xx:y=yy
90 PLOT x*8,y*8,col
100 NEXT:NEXT:WEND
```

Whoops!

Sorry folks but a couple of err
crept into last month's rountin
The editor's promised not to
any more programming a
things should be OK now we
using your entries.

In Fandango the GOSUB
line 30 should be to 110 and t
RUN in 100 should be on a line
its own at 110. Even though t
breaks the ten-line rule, deleti
line 20 will put that right.

In the circles routine the l
four numbers were chopped fr
the end of line 100. The last nu
ber shown should be 180 not 1
then add 30,5,1.

HOWZAT!

By Jason Chown translated for the CPC by Ken Goodman

Howzat is a two-player game which simulates a full innings between England and Australia. It can be played solo, but competition and interaction are vital parts of the game.

You first decide who is going to take charge of which team and at the prompt elect to bat either first or second.

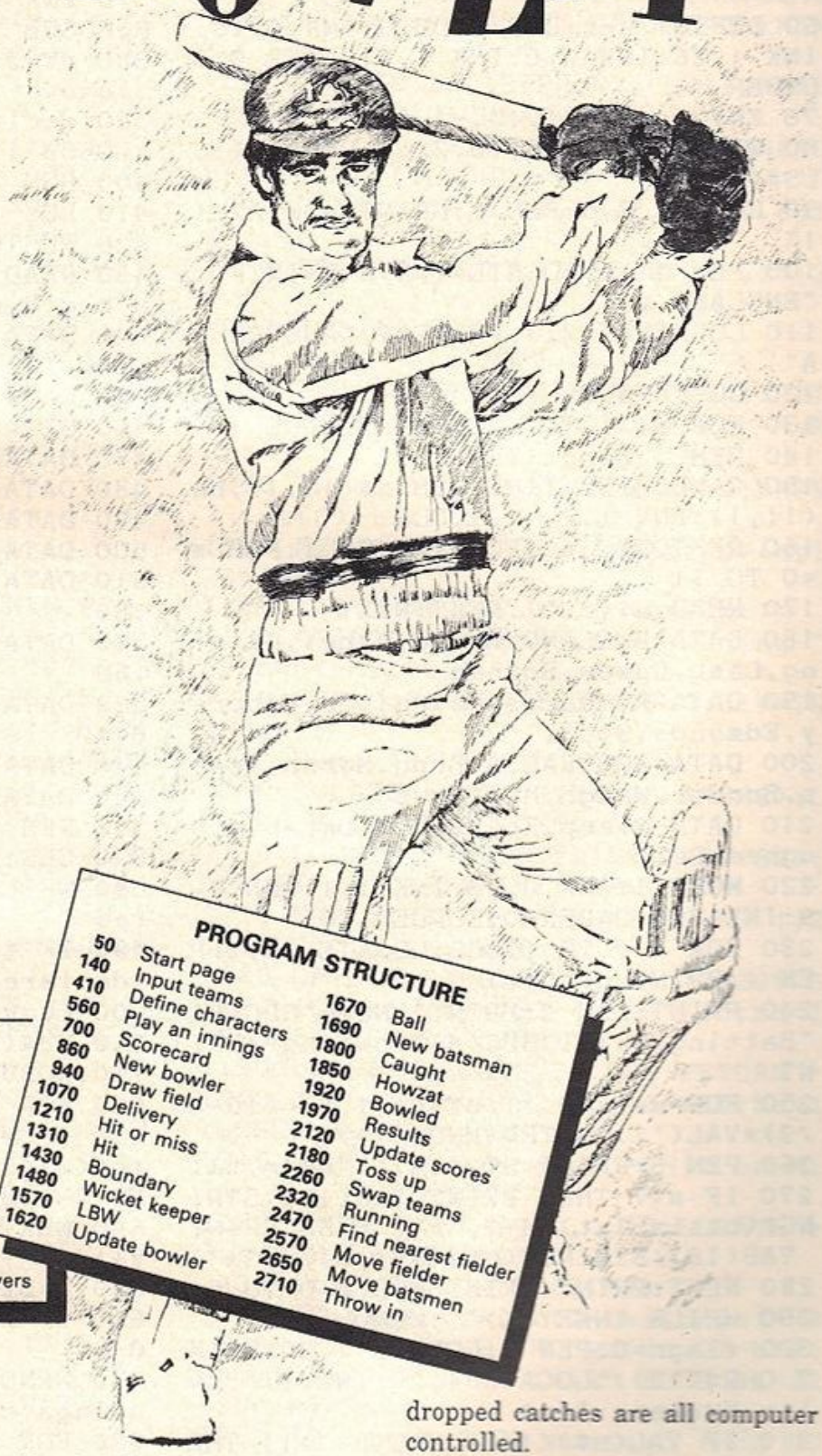
Each player then inputs the numbers of six bowlers in order of skill, and once you have done this the game begins.

First you are shown the score card which depicts the current state of both teams. Both players will use this; the batting team to select its next batsman when a wicket has fallen, or to declare the innings closed; the fielding team to select a new bowler at the end of each over.

On the card the current batsman and the last bowler are highlighted.

The main screen is a graphic representation of the pitch. At the top is the scoreboard showing the teams' and batsmen's scores, wickets lost, and the bowler's name.

The computer will take charge of the bowling and all attempts at striking the ball. The results will depend on the respective skills of both batsman and bowler.



MAJOR VARIABLES

tt	Current team
o	Other team
ov	Number of overs
bx,by	Ball's coordinates
bt1	Batsman facing
w	Wickets lost
bwn()	Bowler number
bws()	Bowler skill
bts()	Batsman skill
n$()	Name of team and players

PROGRAM STRUCTURE

50	Start page
140	Input teams
410	Define characters
560	Play an innings
700	Scorecard
860	New bowler
940	Draw field
1070	Delivery
1210	Hit or miss
1310	Hit
1430	Boundary
1480	Wicket keeper
1570	LBW
1620	Update bowler
1670	Ball
1690	New batsman
1800	Caught
1850	Howzat
1920	Bowled
1970	Results
2120	Update scores
2180	Toss up
2260	Swap teams
2320	Running
2470	Find nearest fielder
2570	Move fielder
2650	Move batsmen
2710	Throw in

If the ball is struck the player controlling the batsmen will be asked whether he wishes to chance a run or not.

If he does, the batsmen will dash across the wicket while the nearest fielder will attempt to run them out by collecting the ball and hurling it at the wickets.

LBWs, catches, bowled-outs and dropped catches are all computer controlled.

If you prefer a limited overs game, simply declare after an agreed number of overs.

```
10 REM              HOWZAT!
20 REM           By Jason Chown
30 REM       adapted by Ken Goodman
40 REM (c) Computing With The Amst
rad
50 REM ------------ CPC ---------
----
60 DEFINT a-z:DEG:MODE 0:INK 0,15:
INK 1,26:INK 2,0:INK 8,7:PAPER 0:B
ORDER 11
70 ENV 1,1,15,1,15,-1,2
80 WINDOW#1,3,18,10,22:PAPER#1,1:C
LS#1
90 LOCATE 7,5:PEN 1:PRINT"HOWZAT !
!"
100 PEN #1,8:LOCATE#1,2,6:PRINT#1,
"ENGLAND v "
110 LOCATE#1,7,8:PRINT#1,"AUSTRALI
A"
120 GOSUB 420
130 FOR d1=1 TO 4000:NEXT:MODE 1
140 REM   Input Teams
150 RANDOMIZE TIME:DIM n$(11,1),rn
(11,1):ENV 1,7,-1,1:CLS
160 RESTORE 180:FOR t=0 TO 1:FOR m
=0 TO 11
170 READ n$(m,t):NEXT:NEXT
180 DATA ENGLAND,Broad,Athey,Gatti
ng,Lamb,Gower,Botham
190 DATA Richards,Defreitas,Embure
y,Edmonds,Small
200 DATA AUSTRALIA,Boon,Marsh,Jone
s,Border,Waugh,Matthews
210 DATA Sleep,Zoehrer,McDermott,H
ughes,Reid
220 MODE 1:INK 0,23:INK 1,1:INK 2,
9:INK 3,6:PAPER 0:BORDER 10
230 FOR t=0 TO 1:CLS:LOCATE 16,2:P
EN 1:PRINT n$(0,t)
240 PRINT:PEN 2:PRINT" Name"SPC(8)
"Batting Skill"SPC(4)"Bowling":PRI
NT
250 FOR m=1 TO 11:bts(m-1,t)=(10-m
/2)*VAL("1."+STR$(RND*9+1))
260 PEN 3:PRINT STR$(m)")"n$(m,t);
270 IF m<7 THEN PRINT TAB(18);STRI
NG$(bts(m-1,t)/4+1,"*") ELSE PRINT
 TAB(18);STRING$(bts(m-1,t)/4,"*")
280 NEXT:PRINT:PRINT:FOR b=0 TO 5
290 WHILE INKEY$<>"":WEND
300 flagn=0:PEN 1:LOCATE 1,20:PRIN
T CHR$(18);:LOCATE 1,20:INPUT;"Bow
ler Number: ";n$
310 IF VAL(n$)<1 OR VAL(n$)>11 THE
N 300
320 n=VAL(n$):FOR s=0 TO b:IF bwn(
s,t)=n THEN flagn=1
330 NEXT:IF flagn THEN 300
340 LOCATE 27,n+5:PRINT"Bowler
"b+1:bwn(b,t)=n
350 bws(b,t)=(8-b*1.5)+INT(RND*
1
360 NEXT:LOCATE 1,25:PRINT"Pres
ny key to go on..":CALL &BB18:N
370 INK 0,10:INK 1,26:BORDER 15
S:GOSUB 2190
380 ov=0:declared=0:tt=0:GOSUB
:lo=ov
390 declared=0:ov=0:tt=1:GOSUB
:GOSUB 1980
400 RUN
410 REM      Set Characters
420 RESTORE 440:FOR i=240 TO 25
430 READ a,b,c,d,e,f,g,h:SYMBOL
a,b,c,d,e,f,g,h:NEXT:RETURN
440 DATA 0,108,146,146,146,146,
,146
450 DATA 38,16,98,98,82,82,76,76
460 DATA 0,0,0,0,56,56,56,16
470 DATA 60,121,105,56,56,40,68,
480 DATA 60,121,185,56,56,40,40,
490 DATA 156,92,60,30,29,40,68,6
500 DATA 0,0,0,0,112,112,112,32
510 DATA 112,120,116,116,116,84,
,80
520 DATA 112,120,116,116,114,84,
,80
530 DATA 56,120,184,184,184,168,
8,40
540 DATA 0,0,0,0,0,129,153,90
550 DATA 90,126,60,60,60,36,36,1
560 REM      One Innings
570 CLS:bt1=1:bt2=2:bn=6
580 WHILE wk(tt)<>10 AND NOT dec
red
590 IF tt=1 AND tot(1)>tot(0) TH
 declared =-1
600 runs=0:ho$(bt1-1,tt)="not ou
610 ho$(bt2-1,tt)="not out":GOSU
710:GOSUB 870
620 FOR bowls=1 TO 6
630 IF wk(tt)<10 AND NOT(tt=1 AN
tot(0)<tot(1)) AND NOT declared
EN GOSUB 950:GOSUB 1080
640 NEXT:IF NOT declared THEN ov
v+1
650 a=bt1:bt1=bt2:bt2=a
660 IF NOT declared THEN GOSUB 1
0
670 WEND:PEN 2:LOCATE 1,25:PRINT
nnings closed. Press any key..."
680 FOR i=1 TO 12:SOUND 17,50,15,
,1:SOUND 10,51,15,0,1:NEXT
690 CALL &BB18:RETURN
700 REM Scoreboard
```

```
710 INK 0,23:INK 1,1:INK 2,9:INK 3
,6:PAPER 0:BORDER 13:CLS
720 PEN 1:LOCATE 10,1:PRINT n$(0,t
t)" after"ov"overs":tot(tt)=0
730 FOR m=1 TO 11:IF m=bt1 THEN PE
N 0:PAPER 3 ELSE PEN 3:PAPER 0
740 LOCATE 1,m+1:PRINT:PRINT MID$(
STR$(m),2,2)TAB(3)n$(m,tt)TAB(14)h
o$(m-1,tt)TAB(39)MID$(STR$(rn(m,tt
)),2,2);:tot(tt)=tot(tt)+rn(m,tt):
NEXT
750 PEN 3:PAPER 0
760 PRINT:PRINT SPC(13)"Extras:"TA
B(39)MID$(STR$(rn(0,tt)),2,2);
770 tot(tt)=tot(tt)+rn(0,tt)
780 PRINT SPC(14)"total:"TAB(22)MI
D$(STR$(tot(tt)),2,2)" for "MID$(S
TR$(wk(tt)),2,2);
790 IF ov>0 AND tt=0 THEN PRINT:PR
INT SPC(11)"(Average"INT(tot(0)/ov
*100)/100"an over)"
800 IF tt=1 THEN IF ov-lo>0 THEN P
RINT:PRINT SPC(11)"(Average"INT(to
t(1)/ov*100)/100"an over)"
810 PEN 2:PRINT:PRINT"   Bowler"SP
C(9)"Sk   Ov Mdns Runs Wkts";
820 o=-NOT tt:IF o=2 THEN o=0
830 FOR m=0 TO 5:IF bn=m THEN PEN
0:PAPER 1 ELSE PEN 1:PAPER 0
840 PRINT MID$(STR$(m+1),2,1)TAB(3
)n$(bwn(m,o),o)TAB(18)STRING$(bws(
m,o)/3+1,"*")TAB(23)bwf(m,o,0)TAB(
27)bwf(m,o,1)TAB(32)bwf(m,o,2)TAB(
37)bwf(m,o,3)
850 NEXT:PEN 1:PAPER 0:RETURN
860 REM       new Bowler
870 WHILE INKEY$<>"":WEND
880 PEN 2:LOCATE 1,25:PRINT CHR$(1
8);:LOCATE 1,25:PRINT"Bowler Numbe
r or 'd' to declare..";
890 a$=INKEY$:IF a$="" THEN 890
900 a$=LOWER$(a$):IF a$="d" THEN d
eclared=-1:PRINT a$:FOR dl=1 TO 50
0:NEXT:RETURN
910 IF VAL(a$)<1 OR VAL(a$)>6 OR V
AL(a$)=(bn+1)THEN 870
920 bwf(VAL(a$)-1,o,0)=bwf(VAL(a$)
-1,o,0)+1
930 bn=VAL(a$)-1:PRINT a$:FOR dl=1
 TO 500:NEXT:RETURN
940 REM     draw the field
950 INK 0,0:INK 1,22:INK 2,17:INK
3,20:PAPER 0:BORDER 10:CLS
960 WINDOW#1,2,39,4,24:PAPER#1,1:C
LS#1
970 LOCATE#1,1,1:PEN#1,0:PRINT#1,C
HR$(212):LOCATE#1,38,1:PRINT#1,CHR
$(213)
980 LOCATE#1,1,21:PRINT#1,CHR$(215
):LOCATE#1,38,21:PRINT#1,CHR$(214)
;
990 LOCATE 11,14:PEN 0:PAPER 1:PRI
NT CHR$(240)SPC(18)CHR$(240)
1000 RESTORE 1040:FOR i=1 TO 11:RE
AD xx,yy
1010 LOCATE xx,yy-1:PRINT CHR$(242
):LOCATE xx,yy:PRINT CHR$(243):NEX
T
1020 LOCATE 28,13:PRINT CHR$(242)C
HR$(8)CHR$(10)CHR$(249)
1030 LOCATE 13,13:PRINT CHR$(246)C
HR$(8)CHR$(10)CHR$(247)
1040 DATA 14,6,31,6,19,8,28,10,8,1
4,8,16
1050 DATA 9,17,11,19,20,21,31,23,3
6,16
1060 RETURN
1070 REM        Delivery
1080 PEN 3:PAPER 0:LOCATE 10,1:PRI
NT n$(0,tt)" Are "tot(tt)" for "wk
(tt)
1090 IF tt=1 THEN LOCATE 28,2:PRIN
T tot(0)-tot(1)+1"to win-."
1100 PEN 2:LOCATE 1,2:PRINT n$(bt1
,tt)rn(bt1,tt)
1110 PRINT n$(bt2,tt)rn(bt2,tt);
1120 PRINT" " n$(bt1,tt)" facing "
n$(bwn(bn,o),o)
1130 PEN 0:PAPER 1:LOCATE 3,5:PRIN
T bowls
1140 FOR z=36 TO 30 STEP-1
1150 LOCATE z,15:PRINT CHR$(242):L
OCATE z,16:PRINT CHR$(243)
1160 LOCATE z+1,15:PRINT CHR$(32):
LOCATE z+1,16:PRINT CHR$(32)
1170 FOR dl=1 TO 50:NEXT:SOUND 1,2
0,10,7,1,0,15:NEXT
1180 ORIGIN 0,0:FOR b=464 TO 208 S
TEP -16:bx=b:by=192-(b+16)/15:bc=1
:GOSUB 1680:bc=o:NEXT
1190 bx=0:by=0:GOSUB 1230:RETURN
1200 bx=b+16:by=192-(b+16)/15:bc=1
:GOSUB 1680:bc=o:NEXT
1210 REM        Choice - Hit or Mi
ss
1220 SOUND 1,0,4,0,1,0,15:LOCATE 1
3,14:PRINT CHR$(248)
1230 ot=0:RANDOMIZE(TIME)
1240 IF bts(bt1-1,tt)-bws(bn,o))=8
 THEN tz=27 ELSE IF bws(bn,0)-bts(
bt1-1,tt))=1 THEN tz=15 ELSE tz=22
1250 rz=RND*tz+1
1260 IF rz=tz-2 OR rz=tz-3 THEN GO
SUB 1490:RETURN
1270 IF rz=tz-1 THEN GOSUB 1860:GO
SUB 1580:GOSUB 1700:RETURN
```

```
1280 IF rz=tz THEN GOSUB 1860:GOSU
B 1930:GOSUB 1700:RETURN
1290 GOSUB 1320:IF ot THEN GOSUB 1
700
1300 RETURN
1310 REM        Hit
1320 flagb=0:flagc=0:a=RND*360+1:x
=12*COS(a):y=12*SIN(a)
1330 a=RND*360+1:x=12*COS(a):y=12*
SIN(a)
1340 ORIGIN 208,176:bx=0:by=0:bc=1
1350 dz=RND*14+4:FOR i=1 TO dz:GOS
UB 1680:bx=bx+x:by=by+y:bc=0
1360 IF TEST(bx,by)<>1 THEN flagb=
-1
1370 IF (bx)-96)*(bx(48)*(by)-80)*
(by(-16)+(bx)16)*(bx(64)*(by)96)*(
by(144)+(bx)224)*(bx(288)*(by)64)*
(by(144)THEN flagc=-1:i=dz:GOTO 13
90
1380 GOSUB 1680:bc=1:FOR dl=1 TO 5
0:NEXT
1390 IF ((bx)384)+(bx(-160)+(by)14
4)+(by(-128)) THEN i=dz:GOSUB 1680
:GOSUB 1440:RETURN
1400 NEXT:IF ot THEN RETURN
1410 IF flagc THEN GOSUB 1520:IF N
OT flagd THEN RETURN
1420 GOSUB 2330:RETURN
1430 REM        Boundary
1440 FOR i=1 TO 2:SOUND 1,50,15,0,
1:NEXT
1450 IF RND(0.8 THEN add2=4 ELSE a
dd2=6
1460 GOSUB 2130:LOCATE 14,4:IF add
2=4 THEN PRINT"Four runs" ELSE PRI
NT"Six runs"
1470 GOSUB 2800:RETURN
1480 REM        Wicket keeper
1490 LOCATE 10,4:PRINT"Wicket keep
er"
1500 GOSUB 2800:RETURN
1510 REM        Choice - caught or
 dropped
1520 flagd=0:IF RND(0.4 THEN GOSUB
 1810:RETURN
1530 dn=INT(RND*11)+1:LOCATE 10,4:
PRINT"Dropped by "n$(dn,o)
1540 flagd=-1:FOR i=20 TO 50 STEP
2:SOUND 17,i,2,15:SOUND 10,i,2,15:
NEXT
1550 FOR dl=1 TO 1500:NEXT:LOCATE
10,4:PRINT SPC(28);
1560 bc=0:GOSUB 1680:RETURN
1570 REM        L.B.W.
1580 ot=-1:LOCATE 3,23
1590 PRINT"Howzat! lbw. (b."n$(bwn
(bn,o),o)")"
1600 ho$(bt1-1,tt)="L.B.W. (b."
(bwn(bn,o),o)+")"
1610 GOSUB 2800:RETURN
1620 REM      Update Bowlers sta
tics
1630 bwf(bn,o,2)=bwf(bn,o,2)+ru
1640 IF runs=0 THEN bwf(bn,o,1)=
f(bn,o,1)+1
1650 bws(bn,o)=bws(bn,o)-2
1660 RETURN
1670 REM        Ball
1680 PLOT bx,by,bc:PLOT bx,by+2:
TURN
1690 REM        New batsman
1700 wk(tt)=wk(tt)+1
1710 IF ho$(bt1-1,tt)<>"Run Out"
HEN bwf(bn,o,3)=bwf(bn,o,3)+1
1720 GOSUB 710:IF wk(tt)=10 THEN
ETURN
1730 LOCATE 1,25:PRINT CHR$(18);
1740 PEN 2:LOCATE 1,25:PRINT"New
atsman Number:";
1750 WHILE INKEY$<>"":WEND
1760 INPUT a:IF a(1 OR a)11 THEN
730
1770 IF ho$(a-1,tt)<>"" THEN 173
1780 bt1=a:ho$(a-1,tt)="not out"
1790 RETURN
1800 REM        caught
1810 cc=INT(RND*11)+1:IF n$(cc,o
n$(bwn(bn,o),o) THEN a$="c.& b.
n$(cc,o) ELSE a$="c."+n$(cc,o)+"
."+n$(bwn(bn,o),o)
1820 GOSUB 1860:LOCATE 3,23:PRIN
Howzat!("a$")"
1830 ot=-1:ho$(bt1-1,tt)=a$
1840 GOSUB 2800:RETURN
1850 REM        Howzat!!
1860 RESTORE 1040:FOR s=0 TO 10:
AD xx,yy
1870 LOCATE xx,yy-1:PRINT CHR$(2
)
1880 LOCATE xx,yy:PRINT CHR$(251
1890 FOR dl=1 TO 100:NEXT:NEXT
1900 FOR i=100 TO 20 STEP -5:SOU
 1,i,2,15:NEXT
1910 FOR dl=1 TO 50:NEXT:RETURN
1920 REM        Bowled
1930 ot=-1:ho$(bt1-1,tt)="b. "+n
bwn(bn,o),o)
1940 LOCATE 11,14:PRINT CHR$(241
1950 LOCATE 3,23:PRINT"HOWZAT! (
wled "n$(bwn(bn,o),o)")"
1960 GOSUB 2800:RETURN
1970 REM        Results
1980 CLS:LOCATE 16,4:PEN 2:PRINT
esults"
1990 IF wk(0)=10 THEN w$="All Ou
```

```
 ELSE w$=" for "+STR$(wk(0))
2000 LOCATE 4,9:PEN 3:PRINT n$(0,0
)TAB(15)tot(0)w$
2010 IF wk(1)=10 THEN w$="All Out"
 ELSE w$=" for "+STR$(wk(1))+"dec.
"
2020 LOCATE 4,12:PEN 1:PRINT n$(0,
1)TAB(15)tot(1)w$
2030 IF tot(0)>tot(1) THEN LOCATE
4,15:PEN 2:PRINT n$(0,0)" Win by"t
ot(0)-tot(1)"runs"
2040 IF tot(0)<tot(1) THEN LOCATE
4,15:PEN 2:PRINT n$(0,1)" Win by"1
0-wk(1)"wickets"
2050 IF tot(0)=tot(1) THEN LOCATE
14,15:PEN 2:PRINT"Match Drawn"
2060 LOCATE 9,23:PRINT"Press P to
play again "
2070 WHILE INKEY$<>"":WEND
2080 a$=INKEY$
2090 IF a$="" THEN 2080 ELSE a$=UP
PER$(a$)
2100 IF a$<>"P"THEN CLS:END
2110 RETURN
2120 REM       Update Scores
2130 IF bx<=-160 AND ABS(by)<24 TH
EN LOCATE 25,4:PRINT"(Extras)":rn(
0,tt)=rn(0,tt)+add2:RETURN
2140 rn(bt1,tt)=rn(bt1,tt)+add2
2150 IF add2 MOD 2=1 THEN dum=bt1:
bt1=bt2:bt2=dum
2160 tot(tt)=tot(tt)+add2
2170 runs=runs+add2:RETURN
2180 REM       Toss up
2190 RANDOMIZE TIME:a=INT(RND*2)+1
2200 LOCATE 6,12:PRINT n$(0,a-1)"
wins the toss..."
2210 LOCATE 6,14:PRINT"Bat first o
r second (1/2)?";
2220 WHILE INKEY$<>"":WEND
2230 b$=INKEY$:IF NOT(b$="1")+(b$=
"2") THEN 2230
2240 b=VAL(b$):PRINT" "b:IF a<>b T
HEN GOSUB 2270
2250 FOR d1=1 TO 750:NEXT:RETURN
2260 REM       Swap teams
2270 FOR t=0 TO 11:a$=n$(t,0):n$(t
,0)=n$(t,1):n$(t,1)=a$:NEXT
2280 FOR t=0 TO 10:a=bts(t,0):bts(
t,0)=bts(t,1):bts(t,1)=a:NEXT
2290 FOR t=0 TO 5:a=bws(t,0):bws(t
,0)=bws(t,1):bws(t,1)=a
2300 a=bwn(t,0):bwn(t,0)=bwn(t,1):
bwn(t,1)=a:NEXT
2310 RETURN
2320 REM   Running
2330 LOCATE 3,24:PRINT"Run (Y/N) ?
";
2340 s=0:g=0:WHILE INKEY$<>"":WEND
2350 a$=INKEY$:IF a$="" THEN 2350
2360 a$=UPPER$(a$):IF a$="N" THEN
RETURN ELSE IF a$<> "Y" THEN 2350
2370 ORIGIN 0,0:bx=bx+208:by=by+17
6:LOCATE 3,24:PRINT SPC(12);:GOSUB
 2480
2380 bx=bx/16:by=(400-by)/16
2390 s=s+1:GOSUB 2520
2400 IF NOT ot THEN LOCATE 3,24:PR
INT"Run (Y/N) ?"; ELSE 2450
2410 WHILE INKEY$<>"":WEND
2420 a$=INKEY$:IF a$="" THEN 2420
2430 a$=UPPER$(a$)
2440 IF NOT ot AND a$="Y" THEN 239
0 ELSE IF a$<>"N" THEN 2420
2450 IF NOT ot THEN add2=s:GOSUB 2
130 ELSE SOUND 1,0,4,0,1,0,15:GOSU
B 1860:LOCATE 14,23:PRINT "Run Out
":GOSUB 2800:ot=-1:ho$(bt1-1,tt)="
Run Out"
2460 RETURN
2470 REM   Find nearest fielder
2480 RESTORE 1040:ns=1000:n=0
2490 FOR w=1 TO 11:READ a,b
2500 IF ABS(a-bx/16)+ABS(b-(400-by
)/16)<ns THEN ns=ABS(a-bx/16)+ABS(
b-(400-by)/16):mx=a:my=b:n=w
2510 NEXT:RETURN
2520 n=13
2530 IF g THEN GOSUB 2720 ELSE GOS
UB 2580
2540 LOCATE mx,my-1:PRINT CHR$(242
):LOCATE mx,my:PRINT CHR$(243):GOS
UB 2660
2550 IF NOT ot AND n<>28 THEN 2530
2560 RETURN
2570 REM Move fielder
2580 sw=0:LOCATE mx,my:PRINT" ":LO
CATE mx,my-1:PRINT" ":LOCATE 11,14
:PRINT CHR$(240)
2590 IF mx>bx AND n MOD 2=1 THEN m
x=mx-1
2600 IF mx<bx AND n MOD 2=1 THEN m
x=mx+1
2610 IF my<by AND n MOD 2=1 THEN m
y=my+1
2620 IF my>by AND n MOD 2=1 THEN m
y=my-1
2630 IF mx=bx AND my=by THEN g=-1:
sw=-1:bx=bx*16:by=400-by*16
2640 RETURN
2650 REM   Move Batsman
2660 LOCATE n,13:PRINT" ":LOCATE n
,14:PRINT" "
2670 LOCATE 41-n,13:PRINT" ":LOCAT
E 41-n,14:PRINT" "
```

Game of the Month listing continues page 63

DISC MAINTENANCE AT A STROKE

Robin Nixon provides a routine to take the tedium out of file management

Catalogue is a versatile program which lets you carry out disc file maintenance from single key presses.

The functions supported are:

LOAD program
RUN program
CHANGE user
CHANGE drive
ERASE file
ERASE all backup files
TITLE disc
RENAME file

These are normally long-winded and tiresome to use, particularly on a CPC464 — it's a lot easier to select an item from a menu and press a key.

You'll be familiar with most of these with the exception of Title. This lets you give a name to each side of your disc up to a mximum of 25 characters. The title is saved in User15 so the file is normally invisible.

The various ON ERROR statements will prevent the CPC6128/664 stopping should a disc error occur. They make debugging a little difficult so it's a good idea to make sure you've typed Catalogue in properly by using the checksum program. Unfortunately it isn't possible to trap disc errors from Basic on the CPC464, so if you make a mistake such as trying to run a text file Catalogue will stop with an error report.

To use Catalogue simply save it using the filename DISC then type:

```
RUN "DISC
```

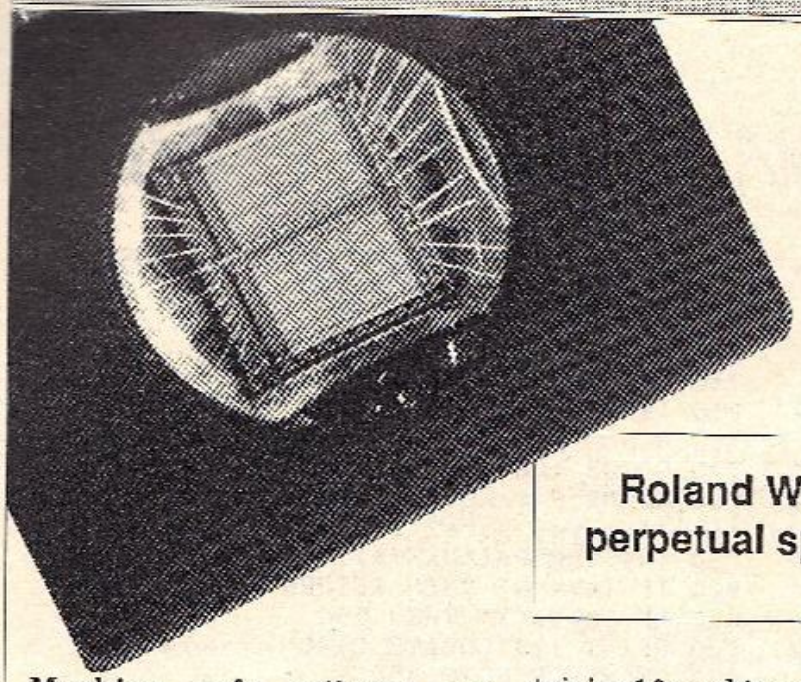
or for those or you with Arnor's Utopia rom, press Ctrl+Enter. You will be presented with a catalo of all the files in User 0 on the d Use the cursor keys to select a and press the key correspondin the operation required. These shown at the bottom of the scre

From time to time you will error messages returned Amsdos such as ????????.BAK found. Don't worry, this sim means you tried to erase bac files when there wern't any. other message you will get TITLE.DSC not found which occ when you are titling a disc wh does not already have a title.

I suggest you save a copy of program on each side of all y discs and get into the habit of r ning it whenever you put a dis the drive. It certainly makes us discs more convenient and yo soon wonder how you mana without it.

```
5 REM 40 characters wide
10 REM    DISC CATALOG
20 REM    BY ROBIN NIXON
30 REM    (C) COMPUTING WITH THE AMSTRAD
40 REM    ----------CPC----------
50 CALL &BB4E
60 MODE 2:INK 0,0:INK 1,20:DEFINT a-g,i-
o,q-z:DIM A$(100)
70 BORDER 13:WINDOW 2,79,1,24:PEN 1:PAPE
R 0:MEMORY &8FFF:ON ERROR GOTO 90
80 |USER,15:INK 1,0:CAT:h=HIMEM:MEMORY h
-&800:GOSUB 200:MEMORY h
90 IF j=0 THEN 110 ELSE OPENIN "TITLE.DS
C"
100 INPUT#9,title$
110 CLS:INK 1,26:PEN 1:CLOSEIN:CLS:|USER
,0
120 PLOT 0,0,1:DRAW 638,0:DRAW 638,398:D
RAW 0,398:DRAW 0,0
130 LOCATE 35,2:PRINT"CATALOG":LOCATE 79
-LEN(title$),2:PRINT title$;
140 LOCATE 1,1:CAT:h=HIMEM:MEMORY h-&
:GOSUB 200:MEMORY h
150 IF j=0 THEN a$(1)="<<No files>>"
160 BORDER 13:WINDOW 2,79,2,24:PEN 1:
ER 0
170 num=(j\4)-((j MOD 4)>0)
180 GOSUB 510:x=1:y=1:PEN 0: PAPER 1:
SUB 250
190 GOSUB 270:CLS:WINDOW 2,79,1,24:|E
E a$:DIM a$(100):GOTO 120
200 j=0:p=h-&7FF
210 IF PEEK(p)=&FF THEN j=j+1:p=p+1 E
 RETURN
220 a$(j)="":FOR k=1 TO 8:a$(j)=a$(j)+
R$(PEEK(p) AND &7F):p=p+1:NEXT
230 a$(j)=a$(j)+".":FOR k=1 TO 3:a$(j)
$(j)+CHR$(PEEK(p) AND &7F):p=p+1:NEXT
240 p=p+2:GOTO 210
250 LOCATE (x-1)*20+1,y+2:GOSUB 1060
260 PRINT a$(num1);:RETURN
270 WHILE INKEY$<>"":WEND
```

```
280 ik$=UPPER$(INKEY$):i1=INKEY(1):i2=IN
KEY(8):i3=INKEY(0):i4=INKEY(2)
290 IF ik$="" THEN 280
300 inf=0:PEN 1:PAPER 0:GOSUB 250
310 IF j=0 THEN 420
320 IF i1>-1 THEN x=x+1:GOSUB 1130:IF x>
4 OR x>j THEN x=1:y=y+1
330 IF i2>-1 THEN x=x-1:GOSUB 1150:IF x<
1 THEN x=4:y=y-1:IF y>(j-num*3) THEN x=x
-1
340 IF i3>-1 THEN y=y-1:inf=1:GOSUB 1170
:IF y<1 THEN y=num:IF x=4 THEN y=(j-num*
3)
350 IF inf=1 THEN GOSUB 1060:IF a$(num1)
="" THEN y=y-1:GOTO 350
360 IF i4>-1 THEN y=y+1:GOSUB 1190:IF y>
num OR x=4 AND y>(j-num*3) THEN y=1
370 IF x=1 AND y>num THEN y=1
380 IF y<1 THEN x=4:y=num:IF y>(j-num*3)
 THEN x=x-1
390 IF x=4 AND y>(j-num*3) THEN x=1:y=y+
1:IF y>num THEN y=1
400 IF x>j THEN x=x-1:y=1:GOTO 400
410 GOSUB 1060:IF num1>j THEN x=x-1
420 IF ik$="T" THEN 550
430 IF ik$="U" THEN GOSUB 650:RETURN
440 IF ik$="R" AND j>0 THEN 680
450 IF ik$="X" THEN 710
460 IF ik$="E" AND j>0 THEN GOSUB 770:RE
TURN
470 IF ik$="L" AND j>0 THEN 860
480 IF ik$="D" THEN 910
490 IF ik$="N" AND j>0 THEN 990
500 PEN 0:PAPER 1: GOSUB 250:GOTO 270
510 PEN 0:PAPER 1:LOCATE 2,23
520 PRINT " R=RUN L=LOAD U=USER D=DRIVE
";
530 PRINT "E=ERASE X=ERASE *.BAK T=TITLE
 ";
540 PRINT "N=RENAME ";:PEN 1:PAPER 0:RET
URN
550 GOSUB 640:LOCATE 2,23:INPUT "ENTER T
ITLE? ",title$
560 title$=LEFT$(title$,25)
570 |USER,15
580 ON ERROR GOTO 630
590 LOCATE 2,20
600 e$="TITLE.DSC":|ERA,@e$
610 OPENOUT "TITLE.DSC"
620 PRINT#9,title$
630 CLOSEOUT:|USER,0:PEN 1:RUN
640 PEN 1:PAPER 0:LOCATE 2,23:PRINT STRI
NG$(76,32);:RETURN
650 GOSUB 640:LOCATE 2,23:INPUT "ENTER U
SER NUMBER (0-15)?",user
660 IF user>15 OR user<0 THEN 650
670 |USER,user:RETURN
680 GOSUB 640:LOCATE 2,23:PRINT "RUNNING
";
690 GOSUB 1060
700 PRINT r$:RUN r$
710 GOSUB 640:LOCATE 2,23:PRINT "ERASING
 *.BAK";
720 LOCATE 2,20
730 ON ERROR GOTO 760
740 e$="*.bak"
750 |ERA,@e$
760 RUN
770 GOSUB 1060:e$=r$
780 GOSUB 640:LOCATE 2,23:PRINT "ERASE "
;e$;"Are you sure?";
790 ik$=UPPER$(INKEY$):IF ik$=""THEN 790
800 IF ik$="N" THEN RETURN
810 IF ik$<>"Y" THEN 790
820 PRINT ik$:LOCATE 2,20
830 ON ERROR GOTO 850
840 |ERA,@e$
850 RUN
860 GOSUB 1060
870 MODE 2:PEN 1:PAPER 0:PRINT "LOADING
";r$
880 ON ERROR GOTO 900
890 LOAD r$:END
900 FOR x=1 TO 5000:NEXT:RUN
910 GOSUB 640:LOCATE 2,23:PRINT "ENTER D
RIVE (A-B)?";
920 ik$=UPPER$(INKEY$):IF ik$=""THEN 920
930 IF ik$<"A" OR ik$>"B" THEN 920
940 PRINT ik$:LOCATE 2,20
950 ON ERROR GOTO 980
960 |DRIVE,@ik$
970 RUN
980 FOR j=1 TO 5000:NEXT:RUN
990 GOSUB 1060
1000 GOSUB 640:LOCATE 2,23:PRINT "RENAME
 ";r$;
1010 INPUT " as?",nn$
1020 LOCATE 2,20
1030 ON ERROR GOTO 980
1040 |REN,@nn$,@r$
1050 RUN
1060 num1=(x-1)*(j\4)+y-2+x-((j MOD 4)>0
)
1070 IF ((j MOD 4)>0) THEN 1090
1080 num1=num1-(x=1)+(x=3)+((x=4)*2)
1090 r$=a$(num1)
1100 is=INSTR(z$," ")
1110 IF is THEN r$=LEFT$(r$,is-1)+RIGHT$
(r$,LEN(z$)-is):GOTO 1100
1120 RETURN
1130 GOSUB 1060:IF a$(num1)="" THEN x=x+
1
1140 RETURN
1150 IF x=0 the RETURN ELSE GOSUB 1060:
IF a$(num1)="" THEN x=x-1
1160 RETURN
1170 IF y=0 THEN RETURN ELSE GOSUB 1060:
IF a$(num1)="" THEN y=y-1:GOTO 1170
1180 RETURN
1190 GOSUB 1060:IF a$(num1)="" THEN y=1
1200 RETURN
```



Writing a ROM

Roland Waddilove offers a way out of the perpetual space problem — put your utilities on rom

Machine code utilities are among the most popular features in *Computing with Amstrad*. They are short, easy to enter and genuinely useful.

However, there is a problem: If all the utilities we've published were loaded at the same time there wouldn't be any room left for your programs. And this isn't the only problem, as most utilities are designed to run at or around &A000. So, once you've loaded one you can't switch off or reset the micro.

The situation isn't quite as gloomy as it might at first appear and there is a solution. The way round the problem is to put the utilities in rom.

Even CPC has at least two roms and many of you will have more. The operating system (os) rom controls all input from the keyboard and cassette, and output to the screen, sound chip and cassette. The Basic rom interprets your Basic programs and translates them into a form understandable by the Z80 processor.

All roms contain either machine code programs or pure data (for instance the character set in the os rom). But the advantage of a rom is that it doesn't use up much of your micro's memory. So it's quite possible to have seven roms each containing 10 machine code utilities — seventy programs — all taking up very little memory, and all instantly available, like Basic.

Each rom, apart from the os starts at &C000 in memory. In other words they all appear at the same address. Now, so that the Amstrad doesn't get double vision and become confused with all these roms it is able to switch each one in and out of the memory map. Effectively, each rom can be turned on or off by your micro as and when it feels like it. Each rom is given a unique number, for instance Basic is 0 and the disc rom is 7. The machine code routines in this are then located from&C000 onwards and can quite easily be called.

After calling a routine the micro switches the disc rom out and switches in whatever rom it was using before the interruption.

When the operating system comes across an RSX command it automatically searches through each of the roms one by one for the corresponding machine code routine. So, when we use the :DISC command it switches in rom 0 (and all the others out) and looks for the machine code routine. If it can't find it, it will look in rom 1, then rom 2, rom 3 and so on until it finds it in rom 7, the disc rom.

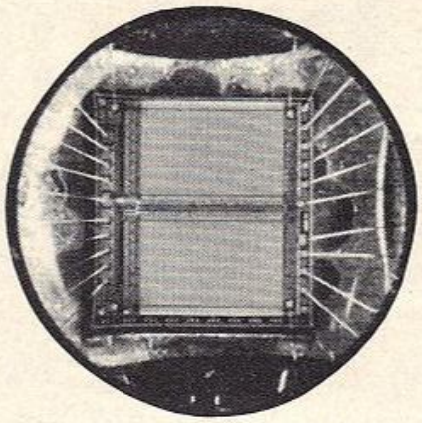
So, how can we take advant of this facility? Can we put our utilities in a rom? Yes, we can our RSXs in rom and it's not difficult as we'll see.

The listing accompanying article is our *Computing with Amstrad* utility rom. You can semble it with Maxam and b your own rom using the rom blo reviewed on page 59 of this iss

Two RSX commands are se — :DOUBLE, a double height routine (use :DOUBLE,@a$ wh a$ contains the text), and :HE which prints a list of the comma accepted by this rom. Not trem dously useful you might think, that isn't my intention. They two simple commands that sh how to implement RSX comma in rom. We'll see how these are up.

All roms start at &C000 and first few bytes form the rom pre or header. You can see this at start of the listing. This essen information is required by the op ating system and if it's not there rom will be ignored or the mi may even crash and you'll have switch off.

The listing shows the layout the rom header. The first byte is rom type — 0 for a foreground r such as a language, 1 for a ba



ground rom (containing utilities), and 2 for an extension rom containing all the bits that won't fit into a foreground rom. As roms are 16k long it's unlikely that you'll see an extension rom, this would be a very large chunk of code.

Bytes one, two and three of the header aren't that important. They contain the mark, version and modification level of the rom and is for your own use. As you modify and improve your rom you can increase these numbers so that you can tell which is the most recent.

Byte four marks the start of a standard RSX command table. The first two bytes of this — 4 and 5 in the rom header — point to the list of RSX names and the third byte — 6 in the rom header — is the start of the jump table.

All the command names appear twice in the command table. The second time each appears it is prefixed with CWTA. This is in case there is a clash of names with any other roms you may have. For instance, Arnor's Utopia also has the command :HELP so use the alternative :CWTAHELP.

The first entry in the jump table is rather special and is only called when the Amstrad is reset or powered up. It should also be given a unique name so that it can't be called as an RSX. This can even be an illegal name such as :MY ROM — it's impossible to enter, from the keyboard, an RSX name containing a space.

When this first entry is called by the operating system the lowest and highest memory locations available are passed in the DE and HL registers. If your utilities require some workspace to temporarily store variables you can grab a chink of ram by increasing DE or decreasing HL. All the currently available roms take ram from the top of memory by lowering HL so it's probably best to do likewise.

The CWTA rom subtracts 32 from HL so the os reserves 32 bytes for our own private use. However, we don't know the address of this 32 byte block.

During this call the title string is printed and you'll see the message "Computing with the Amstrad ROM" printed when you reset or switch the micro on.

Having set up the rom header, command and jump tables and power-up entry, the only remaining task is to write the themselves. The :HELP command is straightforward and merely prints a string of text.

The :DOUBLE command is more complex though you've probably seen or written something similar yourself. When writing a rom RSX the difficulty lies in not knowing where your workspace is. In the case of the CWTA rom there are 32 bytes somewhere in memory.

Every RSX command in rom is entered with the IY register pointing to the start of this workspace. You must write your code so that no matter where the workspace is located the routine will work providing the IY register points to it. It's a bit of a nuisance but, it can be done.

As you can see form the listing you don't need to write a lot of code to create a rom. Providing the header is correct you can have just a few simple commands and pad out the rest of the rom with spaces or zeros.

Using the CWTA rom as an example, why not write your own? Or, using the CWTA rom header replace the commands with some of the others we've published in the past. They'll need some modification, but not too much and it's easier than starting from scratch.

If you think you've written something special why not send it in? You never know, you might just have a potential top selling rom like Arnor's Utopia or Maxam!

```
;-------------------------------
;          Amstrad ROM
;       By R.A.Waddilove
;-------------------------------
WRITE "CWTAROM.BIN"
NOLIST
LIMIT &FFFF
ORG &C000
;-------------------------------
;  Reserves 32 bytes for workspace:
;      0-17 used by DOUBLE
;-------------------------------
;-------------------------------
;          ROM Prefix
;-------------------------------
DEFB 1          ;ROM type...background
DEFB 1          ;mark 1
DEFB 1          ;version 1
DEFB 1          ;modification 1
;-------------------------------
;         Command table
;-------------------------------
DEFW name_table
JP initialise_rom          ;power-up/
reset entry
JP help_cwta               ;help screen
JP help_cwta
JP double                  ;double height
print
JP double
;-------------------------------
;          Name table
;-------------------------------
.name_table
DEFB "CWTA RO","M"+&80          ;bad name
-> can't power-up with command
DEFB "HEL","P"+&80              ;HELP
DEFB "CWTAHEL","P"+&80          ;CWTA
prefix
DEFB "DOUBL","E"+&80
;DOUBLE,@a$
DEFB "CWTADOUBL","E"+&80        ;CWTA
prefix
DEFB 0
;-------------------------------
;    Power-up/Reset intialisation
;         AF,BC corrupted
;-------------------------------
.initialise_rom                 ;save
PUSH DE:PUSH HL
DE/HL
CALL print_string               ;print
power-up message
DEFB 13,10
DEFB "Computing with the Amstrad ROM"
DEFB 13,10,10
DEFB 0
POP HL:POP DE                   ;restore
HL/DE
AND A
LD BC,32:SBC HL,BC              ;grab 32
bytes from top of memory
SCF
RET
;-------------------------------
;   General string print subroutine
;         AF,HL corrupted
;-------------------------------
.print_string                   ;get string
POP HL
address
.sp_loop
LD A,(HL):CALL &BB5A            ;print cha-
racter
INC HL
OR A                            ;done?
JR NZ,sp_loop
JP (HL)
;-------------------------------
;             HELP
; Print syntax and function of commands
;         AF,HL corrupted
;-------------------------------
.help_cwta
CALL print_string
DEFB "Computing with the Amstrad
ROM",1fl,13
DEFB "HELP          - list all com-
mands",1fl,13
DEFB "DOUBLE,@a$ - double height pri-
nt",1fl,13
DEFB "Names can be prefixed with
CWTA",1fl,13
DEFB 7,0
RET
;-------------------------------
;          DOUBLE
;       Double height print
;       AF,BC,DE,HL corrupted
;-------------------------------
.double                                 ;one
DEC A:JP NZ,help_cwta                   ;HL
paramter?
LD L,(IX+0):LD H,(IX+1)
points to string descriptor
LD A,(HL)
;A=length
AND A:RET Z
;abort if 0
LD (IY+17),A
;workspace+17=length
INC HL:LD E,(HL):INC HL:LD D,(HL)
;DE=address
.loop1                                  ;get
PUSH DE
LD A,(DE)
character
CALL &BBA5
;HL=data address
CALL &B906:PUSH AF                      ;DE=-
;enable lower ROM
PUSH IY:POP DE                          ;copy
workspace
LD B,8
data to workspace
.loop2
LD A,(HL):LD (DE),A
INC DE:LD (DE),A
INC DE:INC HL                           ;res-
DJNZ loop2
POP AF:CALL &B90C
tore lower ROM state
LD A,255
;define top half of character
PUSH IY:POP HL
CALL &BBA8
LD A,254
;define bottom half of character
PUSH IY:POP HL
LD DE,8:ADD HL,DE
CALL &BBA8
CALL print_string
;print big character
DEFB 255,8,10,254,11,0                  ;next
POP DE:INC DE                           ;all
character
DEC (IY+17):JR NZ,loop1
done?
RET
list
END
**** END ****
```

All aboard for Memory Lane

IAN SHARPE evaluates two aids for the serious machine code programmer

Putting software in a read only memory chip is a good way of having programs permanently and instantly available and programs that run in rom do not take up large areas of ram.

Taking Arnor's Protext word processor as an example, the disk version leaves around 23k free for text whereas the rom version gives you approximately 40k and is ready to use at a moment's notice.

The process of transferring software to a blank rom chip is not difficult provided you have access to the necessary, usually expensive, equipment.

The hobbyist does not need an industrial quality blower designed to program multiple roms so there's scope for a no frills unit. This is the concept behind John Morrison's budget rom blower reviewed here.

It's housed in a white plastic case which plugs into your Amstrad's expansion port. An external socket takes the eprom to be programmed which can be either 8k or 16k.

Great care must be taken when inserting or removing a rom because the chip is a push fit and it's easy to end up with bent legs (on the rom that is). A ZIF (Zero Insertion Force) socket, where the chip is held in place by a clamp actuated by a lever, would have been more practical.

Software written in unprotected Basic is supplied on tape and is easily transferred to disk. There are actually two programs. The first blows the rom and expects your program to be present as a binary file on disk or tape.

This is loaded into a buffer and the rom programmed with the buffer's contents. In addition to programming, the software offers various facilities such as copying the contents of rom into the buffer, saving and loading binary files, veryifying a rom for erasure and correct programming, and editing single locations.

The other program supplied is a utility that converts Basic programs to a suitable form for blowing. It isn't possible to put a Basic program in rom and run it.

Firstly, it would be underneath the basic interpreter which is rather like trying to read a book while standing on it at the same time. Secondly, Basic stores numeric variables at the end of the program which is impossible when the program is in read only memory.

Basic installed in a rom is called up by an RSX command put in by the utility. Your program is then copied down to its normal position in ram and run. In this case the rom is being treated as a filing system rather than the usual method of writing programs to run in rom.

As you will read on page 43 in this issue, writing a machine code program to run in rom is not the same as writing the equivalent for ram. If you have existing machine code that you want to transfer to rom, you may find it preferable to write a routine that downloads the code from rom to ram.

Of course, the size of program you can store is limited to the capacity of a 16k eprom.

It's easy to criticise the blower. There's no through connector, which is annoying for 464 owners because they can't use the disk drive. The DIY kit which comes without a case does have the con-

Continued on page 65

Show ace that's the CPC

The CPC range refused to take a back seat to its more well publicised cousins – the PCWs and PCs – at the seventh Amstrad Computer Show.

It once again proved to be a major drawing card for visitors to the three-day event and was strongly represented on the exhibition front itself.

Despite the fact that temperatures were in the eighties in London at the time, attendances broke all previous records. With flow through the Alexandra Pavilion's turnstiles on the first day almost double that for any previous event, it more than justified organiser Database's move to a 50 per cent larger venue.

So important has the event become in the computer calendar that it attracted journalists from around the world.

All g in Toky

A Japanese magazine received instant updates of what was happening at the Amstrad Computer Show – thanks to MicroLink.

Jeff Gearing, the UK correspondent of the publication known as 4X4, regularly uses the electronic mail service to file his Tokyo-bound copy from his office in Bristol.

So he couldn't resist the opportunity to use the on-the-spot world wide facility available on the MicroLink stand in the Alexandra Pavilion.

"There is a great deal of interest in Japan abut Amstrad products and they want to know all the news as fast as possible", he told Compu with the Amstrad CPC.

"And MicroLink just happen be the most effective and quic way of getting the information to them.

"Even more so this time as I di even have to go back to my offi use it.".

Jeff first switched to Microl after several of his stories fell of first the British then the Ja nese postal service.

"MicroLink has proved to bc ideal answer to all of the probl we have faced in the past", he s

MEET THE PLONKER

Some of the world's best ideas have often been the simplest like the corkscrew, ring pull cans, sliced bread – and now the Plonker box from Merit Computers (0942 495821).

The product is one of those novelties that makes people wonder why no one has thought of it before.

It is a small plastic disc holder that sticks to the side of the computer keyboard or monitor and was designed to prevent discs from being covered from the odd cup of tea or coffee that sometimes gets split on the desk.

Plonker's designer is Merit's technical director Bill Edgar. He came up with the idea while stuck on a cold train last January.

Ray Seavers, the marketing director, initially said no to the Plonker box as he thought it wasn't viable.

He couldn't have been more wrong.

Unveiled for the first time at show, it was to sell more than 1,(units within three days.

And such was the interest gen ated among overseas visitors t Merit reached agreement for dis bution with representatives companies from France, Belgiu Switzerland Scandinavia, Aust lia and America.

It's the BIG show

Britain's most impressive exhibition centre outside London – G-Mex in Manchester – is the setting for the next Amstrad Computer Show.

The 100,000 sq ft Greater Manchester Exhibitions and Events Centre took £20 million and three years to develop from the shell of a listed former railway station in the city. From October 23 to 25 it will house the country's biggest computer specific show ever to be held outside London. The venue was chosen for its ability to accommodate and reflect Amstrad's current dominance of the UK micro scene. There will be no shortage of attractions for CPC enthusiasts – including all the latest hardware and software and lots of bargain buys.

Software firms help stop child abuse

CPC users are being asked to help a new national campaign against child abuse. The UK software industry, which raised over £300,000 for the worldwide Live Aid movement through Softaid in 1986, is behind the plan.

This year the software house have decided ot concentrate their efforts within this country and offer a helping hand to the NSPCC/RSSPCC.

The campaign is called BACK – Battle Against Cruelty to Kids – and it has its own super hero spearheading the fight against abuse of children.

The aim is to raise money through sales of donated entertainment software for a network of special child protection teams being set up by the NSPCC/RSSPCC.

Each team, providing 24-hour specialist services, costs £266,000 to create.

Focal point of the fundraising scheme is the BackPack series – compilations of games selling for under £10 which will go on sale this autumn.

The Amstrad CPC version contains Xeno, Deactivators, Night Gunner, Tempest, Marsport, Monty on the Run, Starion, Nomad, Starstrike and Knightshare.

"The software industry is a unique vehicle for bridging the gap between the fortunate and unfortunate withing our younger generations", said BACK spokesman Mal Thomas. "We need the support of CPC users to make BACK a success".

CPC for the USA

Leading UK computer retailer Dixons will take the CPC along when it enters the US market this autumn.

The High Street chain will become America's second largest electrical goods retailer when it takes over 144 Silo stores.

Managing director Mark Souhami has said he intends to market the entire range of Amstrad micros in the US and expects to place orders "significantly in excess of £10 million" in the first year.

Dixons will get its stocks from Video, the American distributor Amstrad appointed at the beginning of this year to boost its flagging US presence. This company is said to be currently selling about 9,000 Amstrad computers a month in the US through its 20,000 retailers.

Dixons, which has close ties with Amstrad dating back to 1985, is expected to take 20,000 units initially.

Amstrad chairman Alan Sugar described the new US venture as "a marvellous deal".

He said it could lead to Dixons becoming "our number one force" in the US in the next twelve months.

Dash for the CPC

Prism Leisure has re-released the maze game Boulder Dash for the CPC, to coincide with the launch of a coin-operated game in the arcades based on the game's star Rockford.

Maze "physics" apply except that unlike most this one has jewels which tumble with the rocks.

Boulder Dash contains guards, fireflies, amoebae, enchanted walls, titanium walls and boulders.

The tape versions and disc versions are being prepared.

Have a heart

First find your grave-robbing equipment. That's the opening instruction for the new CPC horror game Bride of Frankenstein on Ariolasoft's 39 Steps labels.

The idea is to revive Frankenstein before his wedding.

The player must disturb gra and crypts to find a pair of lu kidneys, a liver, a brain and a g heart.

The game has 60 rooms, se keys to find, lost souls, hide ghouls, zombies and a cardiac rest feature.

Games under fire

The quality of new games for the CPC has been criticised by Amstrad's software subsidary.

Amsoft has just released its first three titles for the machine this year as sales manager Mike Mordecai bemoans the lack of quality material.

"We didn't have what we felt was good enough for long enough – and then it took quite some time to convert it", he said. "Nobody's coming up with anything good any m – just the same old rubbish".

Mike also said that because price of 3in discs had dropped fi should be cutting the price of th disc games, as Amsoft had done

Game of the Month — **continued from page 53**

```
2680 n=n+1
2690 LOCATE n,13:PRINT CHR$(242):L
OCATE n,14:PRINT CHR$(243)
2700 LOCATE 41-n,13:PRINT CHR$(242
):LOCATE 41-n,14:PRINT CHR$(243):R
ETURN
2710 REM        Throw-in
2720 IF sw=-1 THEN LOCATE mx+1,my:
PRINT" ":LOCATE mx+1,my+1:PRINT" "
:sw=0
2730 dbx=10*COS(ATN(ABS(by-192)/AB
S(bx-168))):dby=10*SIN(ATN(ABS(by-
192)/ABS(bx-168)))
2740 bc=1:GOSUB 1680
2750 IF bx>168 THEN bx=bx-dbx ELSE
 bx=bx+dbx
2760 IF by<192 THEN by=by+dby ELSE
 by=by-dby
2770 bc=0:GOSUB 1680
2780 IF ABS(bx-168)<10 AND ABS(by-
192)<10 THEN ot=-1:LOCATE 11,14:PR
INT CHR$(241):LOCATE mx,my:PRINT"
":LOCATE mx,my-1:PRINT" ";
2790 RETURN
2800 FOR delay=1 TO 5000:NEXT:RETU
RN
```

CP/M – continued from page 26

If CP/M finds there is no more room in the directory on the disc we will get back a value of 255-&FF. If it all went well we'll get back 0, 1, 2 or 3.

If there is any doubt we can call function 19 (delete file) before we make our new file, passing it the same address of our MFCB in register pair DE to remove any old file with the same name first.

To write a record — 128 bytes — we can either put the information we want to write into the default DMA buffer at &80, or use function 26 again to set the DMA address to the start of our data in memory. Calling function 21 (write sequential) will then write the record to disc.

We must repeat this process for every record we wish to write. As with reading we must supply the address of our MFCB in register pair DE — CP/M returns an error code in register A.

A value of 0 means the write operation was successful, while a non-zero value means that the disc is full.

When we have written all the information out we must carry out an additional operation to close the file, or we will find that CP/M has incorrectly updated the directory information on disc which describes where our file is to be found.

We didn't need to bother this when reading because didn't alter anything on the though closing a file after we read it is quite allowable and w no harm. To close one we call f tion 16.

In a similar way to opening pass the address of the MF which CP/M had been upda with block allocation data as have been writing, and we get a directory code as before. A val 255 means that an error occu and our file will be incomplete

Next month we'll look closely at random record access see how to search the disc direc for files.

First Steps – continued from page 23

computing results.

The second figure in the brackets following *result$()* gives the position in the test. So *result$(1,2)* contains the name of the person who came second in English, Sue. In the same manner *result$(1,2)* contains the name of the person who came second in Computing, old clever dick Ian.

In you can't quite see how the array is built up don't worry too much. Just try working your way through the nested loops with a pencil and paper, noting what is read into *result$(subject,place)* as *subject* and *place* vary. You'll soon get the hang of it.

```
10  REM Program V
15  REM old Program VI
20  DIM result$(3,3)
30  FOR subject=1 TO 3
40  FOR place=1 TO 3
50  READ result$(subject,place)
60  NEXT place
70  NEXT subject
80  PRINT "Press1, 2 or 3"
90  INPUT choice
100 CLS
110 IF choice=1 THEN PRINT "English
    results:"
120 IF choice=2 THEN PRINT "Maths
    results:"
130 IF choice=3 THEN PRINT "Computing
    results:"
140 PRINT
150 FOR place=1 TO 3
160 PRINT result$(choice,place)
170 NEXT place
180 PRINT
190 DATA Ian,Sue,Tom
200 DATA Sue,Ian,Tom
210 DATA Ian,Tom,Sue
```

Program V

Finally, take a look at the thir FOR . . . NEXT loop in Progra . Neat isn't it? You get three re sults from one input pointer, *choice*. That should give you s idea of the power of two dimen sional loops.

•We'll be exploring them ther next month.

Now when we press the Copy key, the line is first erased then redrawn in normal logic, and becomes permanently fixed on the screen.

Line 200 prevents the cursor from being permanently fixed. The line is finally redrawn in EOR logic so that it will continue to respond to the cursor keys.

We can therefore fix as many lines as we like with the Copy key. Table 1 shows the values used in graphics logic selection.

With Program VII, all the lines are drawn from a fixed point at the centre of the screen. However we may wish to move the fixed end of the line as well as the free end.

We can modify it once again by changing line 150 to :

```
150 IF INKEY(9)=0 THEN code=0:

GOSUB 170 :x=a :y=b :code=1
```

Now, when the line is fixed, the position of the fixed end is moved to the last position of the free end. In this way continuous lines can be drawn, and we can build up pictures.

In an earlier program we drew the outline of a house using data in the program. We can now draw the same house or, indeed, any other shape simply by positioning the cursor and drawing the appropriate lines.

The drawback with Program VII is that we have to draw a continuous line. There are no facilities to lift the pen from the paper and move it to another position. Also we can't erase a line once it has been fixed.

Program VIII is similar to the previous program, but we have now added the missing facilities which provide us with a complete rubber banding design program. There are several keys used in this program and these are shown in Table II.

This program is a little more complex than the previous one. You might like to spend some time working through it to understand fully how it works.

You should have plenty of time as this is the last in the series on basic graphics techniques.

We hope that you've enjoyed it and learnt a lot. If you want to learn more, then you'll have to delve into the world of machine code. The excellent article by Mike Bibby and Roland Waddilove should help here.

And, in the meantime, if you come up with any graphics masterpieces let us have a look at them here at *Computing with the Amstrad*.

REVIEW

Continued from page 59

nector, which is fine if you can use a soldering iron. There's also the matter of the ZIF socket, and on a CPC6128 the unit lifts the back of the keyboard by a few millimetres.

For a peripheral that was permanently connected these would be major faults, but the blower is only going to be plugged in when needed.

The documentation was in the form of stapled photocopied sheets. Machine code programmers armed with the firmware manual and our rom article will have few problems in writing their own roms. Experienced Basic programmers should manage to put their programs on rom without too much difficulty though the instructions leave something to be desired for less seasoned users.

If you want a blower, don't let the negative aspects put you off. As long as you aren't planning to blow large numbers of roms, it does what it's supposed to do efficiently enough, but lacks finish on the details.

Postbag . . .

★ On page 61 of the September issue of *Computing With The Amstrad* a cou of words were left off the end of line 1600 – it should read:

```
1600 score%=0:lives=3:chase=1:burrow=0:bonus%=0
```

★ On the Contents page we put in by mistake a section for Page 46 – **Feat Story**. This will appear in *Computing With The Amstrad* in the near futu

Computing With The Amstrad

Computing With The Amstrad welcomes program listings and articles for publication. Material should be typed or computer printed, and must be double spaced. Program listings should be accompanied by cassette tape or disk. Please enclose a stamped addressed envelope or the return of material cannot be guaranteed. Contributions accepted for publication by Database Publication or its licensee will be on an all-rights basis.

While every care is taken, the publishers cannot be held responsible for any errors in articles, listings or advertisements. *Computing With The Amstrad* is an independent publication and neither *Computing With The Amstrad* and neither Amstrad plc or Amsoft or their distributors are responsible for any of the articles in this issue or for any of the opinions expressed.