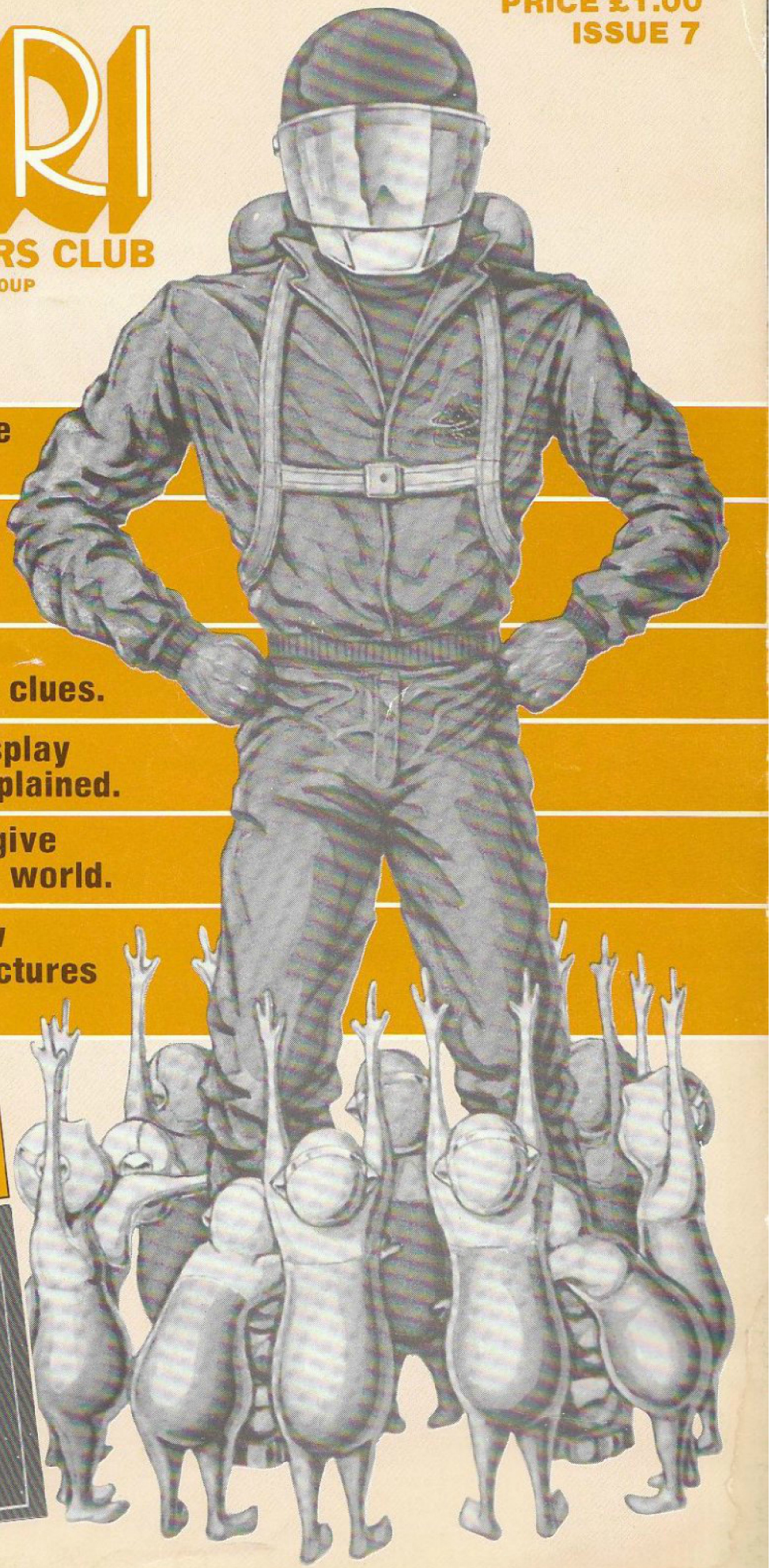
THE U.K.

ATARI

COMPUTER OWNERS CLUB

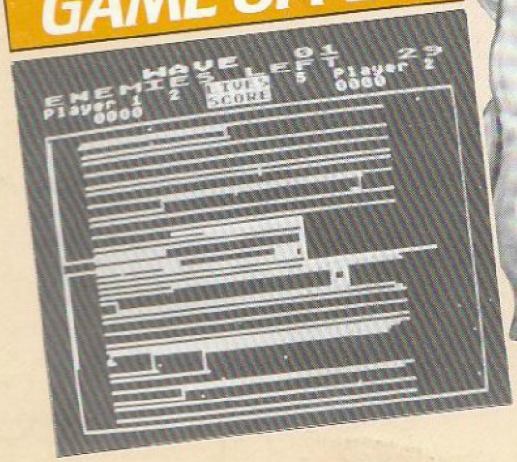
INDEPENDENT USER GROUP

PRICE £1.00
ISSUE 7



IN THIS ISSUE

EXCLUSIVE GAME OFFER



THE U.K.
ATARI
COMPUTER OWNERS CLUB
INDEPENDENT USER GROUP

CLUB CHANGES

There have been a few changes in the club officers since the last newsletter, unfortunately Chris Barlow has left us and moved on to other things, but his position as President has been taken over by Ron Levy and the editorship of the newsletter is now the responsibility of Roy Smith. The other major change that has occured is that the membership fee has been increased to £4.00 this is to cover increased costs of producing the newsletter and the recent increase in postal charges. However, we still feel that this is good value for the service we provide to members and we hope that you feel the same and will continue to support the club. Remember it is your club and you are welcome to contribute at any time to the contents of the newsletter, be your contribution a mega five page article, or just a few line programming tip!

For those of you whose interest in a Light Pen and the RTTY terminal unit was kindled by the last issue, perhaps you would like to know that both are now available in kit form from Maplin Electronic Supplies. The Light Pen includes a printed circuit board, but not the pen holder, and the TU1000 unit has everything in the kit except the case. The Light Pen is £10.95 and the TU1000 is £49.95.

The main article in this issue is all about machine code interrupts, both the Display List and the Vertical Blank types, with useful routines which can be tacked on to your programs. Another useful article gives an ultra fast routine to search a string of characters. Following on from last issue's article about printing out Graphics 8 pictures, the Compressor is a program to compress the picture data so that each picture takes up less room. Our teach yourself machine code tutorial continues and several adventures are reviewed in our regular adventure column. Plus software reviews, new library software, 'Did you Knows' and other regulars.

Finally, don't forget that the higher the membership the better the club will become, the newsletter can expand futher (some may have noted that issue 6 was increased to 32 pages) and other benefits could be considered (maybe buying disks in bulk so that members can get them cheaply). So if you have friends who read your copy of this newsletter why not talk them into joining, then they, as well as you, can benefit.

CREDITS

Editor	Roy Smith
Technical Editor	Ron Levy
Technical Editor	Keith Mayhew
Art Editor	Peter Blackmore
Adventure Editor	Steve Hillen
Photography	John Attfield

CONTENTS

SUBSCRIPTION RATES

U.K. and Ireland	£4.00
Europe	£7.00
Outside Europe (Surface)	£7.00
Outside Europe (Airmail)	£10.50

The above prices are in English Pounds Sterling and include postage and packing.

A subscription/membership fee to join the U.K. Atari Computer Owners Club is just £4.00 for four issues of the club magazine. All cheques/postal orders are to be made payable to the 'U.K. Atari Computer Owners Club'. Overseas membership is also available at slightly higher rates. Overseas members who use the Library service should include enough extra monies to cover return postage.

ADVERTISEMENTS

Please note that the club cannot be held legally responsible for claims made by advertisers.

COVER: Planetfall is a trademark and is copyright Infocom Inc., 55 Wheeler Street, Cambridge, MA 02138, U.S.A.

ULTRA FAST STRING SEARCH

By Keith Mayhew

This ultra fast, short machine code routine, just over 100 bytes long, will search and compare for a set of given characters anywhere in memory. If a match is found then the relative position from the start of the search is returned. This routine could be used in a program for many purposes, such as searching for a correct spelling in a word-processing package or spelling checker type program. Although this program is capable of searching anywhere in memory, it is shown in the demo BASIC listing searching within a BASIC string for a specified string of characters, and this is probably the most likely way it would be implemented. This program can also search for literally any character at all i.e. space, control, and graphic characters anywhere in memory including the O.S. ROM.

How to use it

To find the first occurence of the set of characters to be found, called the sub-string, in the specified area of memory, called the main string, you simply have, as a BASIC statement, the following line:

X=USR(ADR(SEARCH$),A1,L1,A2,L2)

Where A1 and L1 are the address and length of the main string and A2 and L2 are the address and length of the sub-string. The address of the machine code is found by 'ADR(SEARCH$)' in the above line, where the machine code is held within a string called 'SEARCH$'. The machine code could equally be fixed in memory e.g. page six (1536) onwards. Please note that the length of the sub-string (L2) must be less than 256 as this greatly simplifies and speeds up the routine, and it is extremely unlikely that the sub-string would need to be greater than this anyway.

Upon the first call to the routine where the four parameters are passed (A1,L1,A2,L2) then the program will return to BASIC with the variable ('X') containing the relative position in the string of the first occurence. Note that if, for instance, the sub-string was found at the beginning of the main string then '1' would be returned, i.e. it is in the first position of the string. This convention was used so that it is directly compatible with BASIC's strings, where the first character is also designated the position '1'. If you are searching outside of BASIC's strings then to compute the address of the match you would add the returned position to the main string's address minus ONE, as the first position is actually zero.

If a string search fails, i.e. it does not find an exact match for the sub-string, then the value of zero is returned in the variable (X). If a match is found, i.e. X is greater than zero, then a second search can be implemented to continue from the last position by a second entry point in the routine, to see if a second match can be found. This can be repeated for as long as X is greater than zero to find as many matches as there actually are in the main string. The Second call is of the form:

X=USR(ADR(SEARCH$)+71)

Listing 1 is a demonstration of a string search incorporating the two 'USR' calls to find all the matches in the main string. Type in the program double checking the DATA statements. When you are satisfied you have made no errors, save the program to cassette or disk in the usual way i.e. CSAVE or SAVE"D:SEARCH.BAS". Run the program and take note of how fast it finds all the occurences of the specified sub-string. If you want to use the machine code in your own programs then 'LIST' the appropriate line numbers to cassette or disk using: LIST"C:",31000,32130 or LIST"D:SEARCH.LST",31000,32130. This can then be ENTERed over another program. Experiment with the demo listing to find different sections or lengths of the string, so that you are familiar with how to change the parameters.

Listing 1

```
10 DIM M$(500),S$(255)
20 ? CHR$(125);"Please wait while machine code"
30 ? "is read into 'SEARCH$'..."
40 GOSUB 31000
50 M$="THIS IS THE STRING THAT WILL BE SEARCHED FOR A MA
TCH BY THE MACHINE CODE."
60 ? :? :? "Please enter the characters you wish"
70 ? "to find in the following string:"
80 ? :? M$
90 INPUT S$
100 L1=LEN(M$):L2=LEN(S$)
110 A1=ADR(M$):A2=ADR(S$)
120 X=USR(ADR(SEARCH$),A1,L1,A2,L2)
130 IF X=0 THEN 200
140 GOSUB 1000
150 X=USR(ADR(SEARCH$)+71)
160 IF X=0 THEN 200
170 GOSUB 1000
180 GOTO 150
200 ? :? :? "End of search...":? :? :? :GOTO 60
1000 REM Print results.
1010 ? :? :? "Match found at position ";X
1020 ? :? "String from there reads:"
1030 ? M$(X,L1)
1040 RETURN
31000 REM Load 'SEARCH$' with machine code.
31010 DIM SEARCH$(109)
31020 FOR I=1 TO 109
31030 READ X
31040 SEARCH$(I,I)=CHR$(X)
31050 NEXT I
31060 RETURN
32000 DATA 104,104,133,204,104,133,203,104
32010 DATA 141,241,6,104,141,240,6,104
32020 DATA 133,206,104,133,205,104,104,141
32030 DATA 242,6,240,74,169,1,141,243
32040 DATA 6,169,0,141,244,6,238,240
32050 DATA 6,208,3,238,241,6,160,0
32060 DATA 177,203,209,205,208,18,200,204
32070 DATA 242,6,208,244,173,243,6,133
32080 DATA 212,173,244,6,133,213,96,104
32090 DATA 238,243,6,208,3,238,244,6
32100 DATA 230,203,208,2,230,204,173,244
32110 DATA 6,205,241,6,208,208,173,243
32120 DATA 6,205,240,6,208,200,169,0
32130 DATA 133,212,133,213,96
```

Listing 2

```
0100 ;Written by Keith Mayhew.
0110 ;Relocatable machine code.
0120 ;Search for a sub-string,
0130 ;in a main string.
0140 ;Called from BASIC by:
0150 ;X=USR(ADR(SEARCH$),A1,L1,A2,L2)
0160 ;Where A1 is the address of the
0170 ;main string and L1 is its
0180 ;length, similarly for the
0190 ;sub-string; A2 and L2.
0200 ;If 0 is returned string was
0210 ;not found.  Second entry point
0220 ;continues search for the next
0230 ;occurence of the string called by:
0240 ;X=USR(ADR(SEARCH$)+71)
0250         X=      $CB
0260 STRING  X=      X+2       Indirect pointers
0270 SUBSTR  X=      X+2       to the two strings.
0280         X=      $06F0
0290 STRNGL  X=      X+2       Main string length.
0300 LEN     X=      X+1       Sub-string length.
0310 PNTR    X=      X+2       Position in string.
0320         X=      $0600     Locate in page 6.
0330         PLA                Clear stack.
0340         PLA                String pointer HI,
0350         STA     STRING+1   store.
0360         PLA                String pointer LO,
0370         STA     STRING     store.
0380         PLA                String length HI,
0390         STA     STRNGL+1   store.
0400         PLA                String length LO,
0410         STA     STRNGL     store.
0420         PLA                Sub-string pointer HI,
0430         STA     SUBSTR+1   store.
0440         PLA                Sub-string pointer LO,
0450         STA     SUBSTR     store.
0460         PLA                Sub-string length HI,
0470         PLA                Sub-string length LO,
0480         STA     LEN        Store LO byte.
0490         BEQ     NOTFND     If zero then not found.
0500         LDA     #$01       Set pointer
0510         STA     PNTR       to '1'
0520         LDA     #$00       and HI
0530         STA     PNTR+1     byte to zero.
0540         INC     STRNGL     Increase string
0550         BNE     SEARCH     length by
0560         INC     STRNGL+1   one.
0570 ;Search for sub-string.
0580 SEARCH  LDY     #$00       Set index to zero.
0590 LOOP1   LDA     (STRING),Y Get main character.
0600         CMP     (SUBSTR),Y Compare to other.
0610         BNE     NXTCHR     If not equal try next.
0620         INY                Increment index
0630         CPY     LEN        and compare to length,
0640         BNE     LOOP1      go back if not at end.
0650 ;Found sub-string.
0660         LDA     PNTR       Set BASIC
0670         STA     212        variable
0680         LDA     PNTR+1     equal to
0690         STA     213        pointer
0700         RTS                and return to BASIC.
0710 ;Entry for continuing search.
0720 CONT    PLA                Just clean stack.
0730 ;Point to next position in
0740 ;the main string.
0750 NXTCHR  INC     PNTR       Increment
0760         BNE     SKIP1      pointer
0770         INC     PNTR+1     by one.
0780 SKIP1   INC     STRING     Increment
0790         BNE     SKIP2      string pointer
0800         INC     STRING+1   by one.
0810 SKIP2   LDA     PNTR+1     Compare pointer HI
0820         CMP     STRNGL+1   to string HI.
0830         BNE     SEARCH     If not equal go back.
0840         LDA     PNTR       Compare pointer LO
0850         CMP     STRNGL     to string LO.
0860         BNE     SEARCH     If not equal go back.
0870 ;String not found.
0880 NOTFND  LDA     #$00       Store zero
0890         STA     212        in BASIC
0900         STA     213        variable.
0910         RTS                Return to BASIC.
```

You may notice that when the program is run it takes a few seconds to load the machine code search routine (held in the DATA statements), but once this is achieved the actual use of the routine is extremely fast as you will see. The program will then display the main string (M$) and will ask you to enter a group of characters for it to find. First enter 'THEM', the program will report 'End of search' as no match was encountered, note that the first three characters i.e. 'THE' do appear in the main string and so does 'THE M' but these are not detected because the space is a significant part of the string. Now try entering a character or a group of characters which are in the main string, the program will tell you that a match was found and what position in the string it occured and it will also print the rest of the string from that position. If there was more than one occurence of your chosen character(s) then all of these will be printed out in a similar manner. You may find it useful to know that when the program finds a large number of matches, you can use CONTROL 1 to pause the listing of the results and press CONTROL 1 again to resume.

The program DIMensions M$ and S$ to 500 and 255 respectively. This as far as S$ is concerned is its maximum value, but M$ can be as large as the free memory in your machine will allow. To give you an idea of the speed of the machine code routine, you could place more characters into M$ on line 50 or you could dimension M$ to a larger number (a few thousand) and then set the last few characters to something, say, your name! For example try the following changes:

```
10 DIM M$(2000),S$(255)
55 M$(1196,2000)="KEITH"
```

When this is RUN the program will print M$, unfortunately, now very long! It will contain a series of dummy characters from the full stop after 'CODE' to the name 'KEITH' this is because you have not told BASIC to clear these characters. When you are asked to enter the characters you wish to find, type 'KEITH', and it should find it very quickly, giving you an indication of its speed searching through two thousand characters.

Listing 2 is the machine code which includes comments for those of you who wish to follow the flow of the routine.

If this article seems familiar, then it's probably because you are a reader of 'Practical Computing' magazine. This article was published in the October 1984 edition in the 'Open File' section. Our thanks to Mr. Jack Schofield for publishing it and for supporting Atari computers in general through the pages of Practical Computing, there aren't many editors who bother with things Atari these days.

THE LITTLE SILVER BOX

by Paul Griffey - Bath

Have you ever been asked, while showing a non-computer addict your beloved system, "Yes, but what do you use your computer for?" This question often causes the loading of your latest game, with mega graphics, to be aborted. But now with this Relay Box you need no longer be ashamed. It will enable your computer to communicate with the outside world, and here are a few ideas to start you off. You could use it as a programmable alarm clock to operate your radio, TV etc., or you could have a security system to control the lights in your house when you go out in the evening. Maybe you could make a system with floats and micro switches fitted to the 'TRIGGER' inputs, and a pump to control the level of water in a tank, or a system for the disabled that uses their TV and a joystick to operate curtains, open the front door or call for assistance.

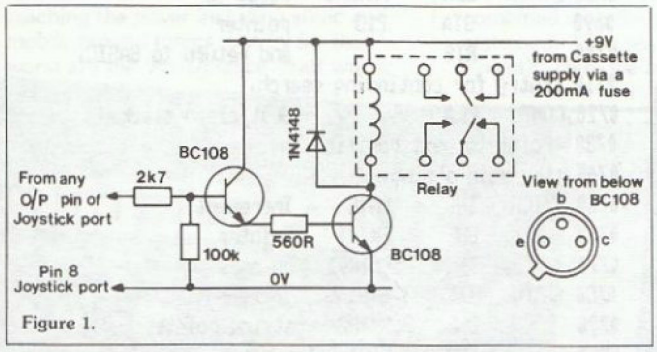


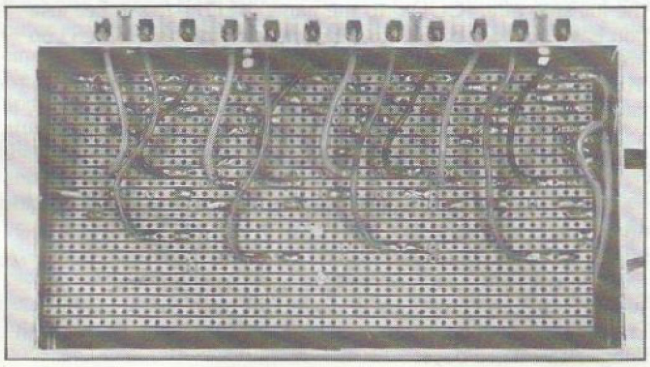
Figure 1.

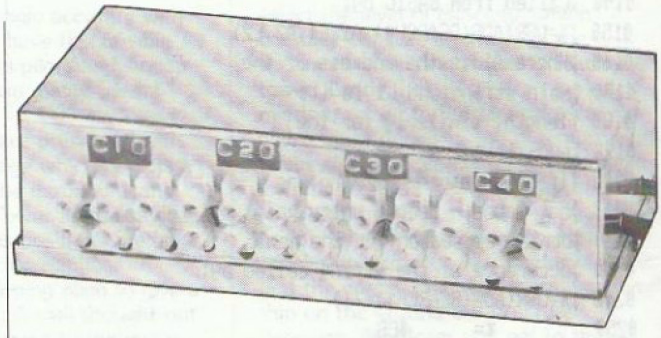
Design

The Relay Box was designed to be as simple as possible using readily available components. As shown in figure 1, the supply is taken from the cassette output on the power pack and provides 9V dc. The reason for using this supply and not the computer's own 5V supply is two-fold. Firstly for isolation there is less chance that a high voltage spike from the relays will find its way into your computer. The second reason is because of components. The majority of 5 volt relays will not handle voltages at adequate currents. The 9 volt supply enables a 6-12 volt relay to be used. MAPLIN can supply a suitable relay (YX98G) which is rated at 5A,240V AC, 30V DC.

Construction

When a program 'POKES' a number to a joystick port the corresponding pin switches to logic 1 (5V). The relevant transistors amplify the signal and drive the relay, switching the external circuit. Figure 1 shows the circuit required to drive one relay. The prototype had four relays, which is the maximum that can be driven from one joystick port. It was built on veroboard and housed in an aluminium box. Construction should be straightforward but care should be taken on the following points. When making the lead for the 9 volt supply, ensure the negative





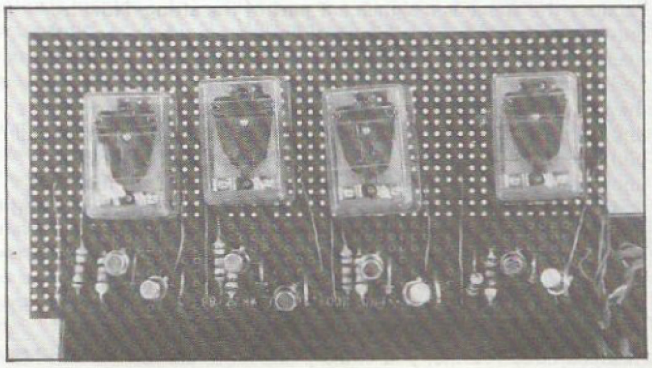

```
100 REM SET UP PACTL
110 DIM A$(1)
120 PORTA=54016
130 PORTB=54017
140 PACTL=54018
150 PECTL=54019
160 REM SET BIT 2 OF PORT A CONTROL REGISTER TO '0' & AL
LOW ACCESS TO DATA DIRECTION REGISTER
170 POKE PACTL,56
180 REM SET PORT A TO ALL OUTPUTS
190 POKE PORTA,255
200 REM SET BIT 2 OF PORT A CONTROL REGISTER TO '1' & AL
LOW ACCESS TO DATA REGISTER
210 POKE PACTL,60
220 REM OPERATE RELAYS IN A BINARY COUNT SEQUENCE
230 FOR I=1 TO 15
240 POKE PORTA,I
250 ? "      ";I
260 REM PRESS RETURN FOR NEXT NUMBER
270 INPUT A$
280 NEXT I
290 POKE PORTA,0
300 STOP
```

Listing 1.

side is connected to the 0V rail, otherwise if the cassette is used at the same time, the transformer will be shorted out! Ensure diodes across the relay are inserted correctly as these suppress the high voltage spikes from the relays when they change state (these spikes can be 1000s of volts!). Figure 2 shows how to connect the board to the port.

Testing

Connect the circuit to the cassette supply but don't insert the 'D' plug into the front port. The relays should all be de-energised. If all is well, connect the +9V rail to one of the inputs



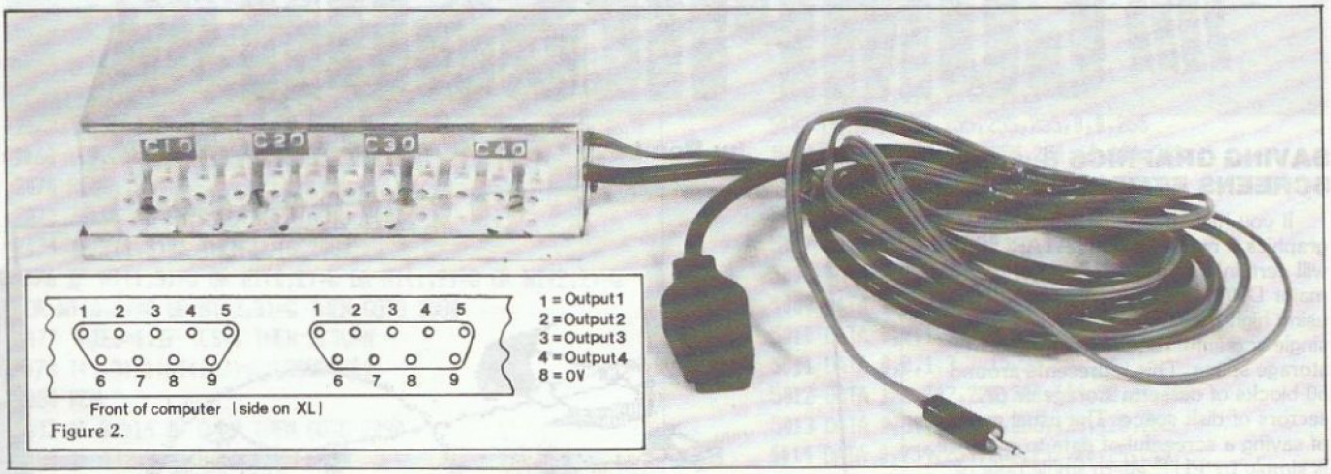


Figure 2.

on the plug (see fig 2) and the associated relay should now operate. Repeat this test on all four inputs. Now insert the plug into the port and boot your computer with BASIC. If the blue screen does not appear on your TV switch off IMMEDIATELY and check the circuit. All of the relays should be energised and now you are ready to enter the programs.

Software

The setting up of the ports has been covered in a previous article (Issue 3), but to start you off I have shown a couple of simple programs. The first will check the operation of the four relays connected to port 1, and the second will show you how to control individual relays selected from a menu.

Warning

Please exercise caution if you intend to switch Mains appliances. Do not try to control large inductive loads. Suppression will be necessary in the form of a 0.047 microfarads 250V AC capacitor (MAPLIN FF55K), across the contacts of the relay.

NOTE: In this program, anything which is underlined, should be entered in "INVERSE".

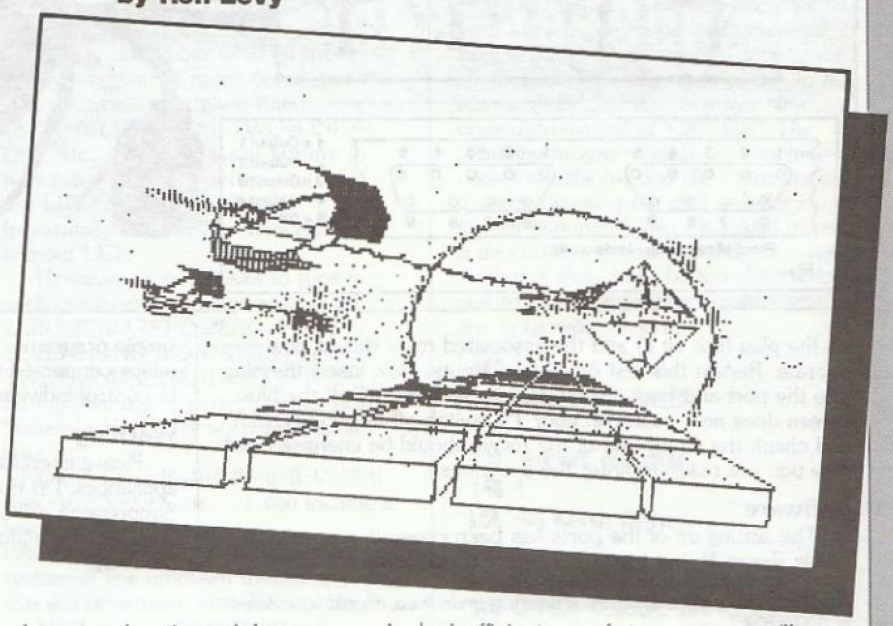
Listing 2.

```
110 REM *PORT CONTROL*
130 REM RESET PORTS
140 POKE 54018,56:POKE 54019,56:POKE 54016,0:POKE 54017,
0:POKE 54018,60:POKE 54019,60
150 REM WHICH PORTS???
160 GRAPHICS 0:SETCOLOR 2,5,3:DIM A$(5)
170 TRAP 170
180 ? :? "Which joystick port do you wish to usefor the
relay box (1-4)";
190 INPUT RELAY
200 IF RELAY<1 OR RELAY>4 THEN 180
210 TRAP 210
220 ? :? "Which port do you wish to use for the joystick
 (1-4)";
230 INPUT JOYSTICK
240 IF JOYSTICK<1 OR JOYSTICK>4 THEN 220
250 IF JOYSTICK=RELAY THEN ? " The joystick & relays can
't operate from the same port !":GOTO 180
260 TRAP 0
270 REM SET UP PORT FOR OUTPUT
280 IF RELAY=1 OR RELAY=2 THEN PORT=54016
290 IF RELAY=3 OR RELAY=4 THEN PORT=54017
300 POKE PORT+2,56
310 IF RELAY=1 OR RELAY=3 THEN POKE PORT,15
320 IF RELAY=2 OR RELAY=4 THEN POKE PORT,240
330 POKE PORT+2,60:POKE PORT,0
340 ? :? "THE PORTS HAVE BEEN SET,"
350 ? "Plug the relays into port ";RELAY
360 ? "Plug the joystick into port ";JOYSTICK
370 ? "  Then press any key"
380 POKE 764,255
390 IF PEEK(764)=255 THEN 390
400 REM DISPLAY STATUS
410 GRAPHICS 18
420 ? #6;" move joystick left and right to select the re
lay and press fire to change state"
430 POSITION 2,6:? #6;"off off off off"
440 POSITION 2,7:? #6;"RL1 RL2 RL3 RL4"
450 REM SELECT RELAY
460 RELAY=1
470 IF STRIG(JOYSTICK-1)=0 THEN 580
480 A=STICK(JOYSTICK-1)
490 IF A<>7 AND A<>11 THEN 470
500 IF A=7 THEN 530
510 RELAY=RELAY-1:IF RELAY<1 THEN RELAY=4
520 GOTO 540
530 RELAY=RELAY+1:IF RELAY>4 THEN RELAY=1
540 POSITION 2,7:? #6;"RL1 RL2 RL3 RL4"
550 POSITION RELAY*4-2,7:? #6;"RL";RELAY
560 IF STICK(JOYSTICK-1)=15 THEN 470
570 GOTO 560
580 REM CHANGE STATE   1=ON 0=OFF
590 POSITION RELAY*4-2,6
600 ON RELAY GOTO 610,630,650,670
610 RL1=1-RL1:IF RL1 THEN 680
620 GOTO 690
630 RL2=1-RL2:IF RL2 THEN 680
640 GOTO 690
650 RL3=1-RL3:IF RL3 THEN 680
660 GOTO 690
670 RL4=1-RL4:IF  NOT RL4 THEN 690
680 ? #6;"ON!":GOTO 710
690 ? #6;"off"
700 REM FIND VALUE FOR PORT
710 VALUE=RL1+RL2*2+RL3*4+RL4*8+RL1*16+RL2*32+RL3*64+RL4
*128
720 POKE PORT,VALUE
730 IF STRIG(JOYSTICK-1)=0 THEN 730
740 GOTO 470
```

»» COMPRESSOR ««

SAVING GRAPHICS 8 SCREENS EFFICIENTLY

by Ron Levy



If you have been working with graphics 8 mode on your ATARI you will certainly be aware of one of the major DRAWbacks (pun intended!) of using high resolution graphics, to save a single screenful requires around 8K of storage space. This represents around 60 blocks of cassette storage or 62 sectors of disk space. The usual method of saving a screenful of data to a device, is simply to 'PUT' every single byte of the screen data to the device exactly as it exists in memory, as shown in Program 1.

```
1 REM ........PROGRAM 1........
2 REM Save Screen with no compression
10 DIM F$(15)
20 PRINT "Which File->";:INPUT F$
100 GRAPHICS 8+16:COLOR 1
110 PLOT 0,0:DRAWTO 319,191
120 PLOT 0,191:DRAWTO 319,0
130 GOSUB 26000:STOP
26000 REM .....Dump Screen Data.....
26100 OPEN #1,8,0,F$
26200 SS=PEEK(88)+256*PEEK(89)
26210 ES=SS+7680
26300 FOR I=SS TO ES
26310 X=PEEK(I)
26320 PUT #1,X
26400 NEXT I
26500 CLOSE #1:RETURN
```

This demo routine simply opens the screen in mode 8 and then draws an X across the width of the picture. It then asks you for the filename to save this artistic masterpiece. Those of you who are fortunate enough to be blessed with the luxury of a disk drive, can enter the usual D:FILENAME, but those who use a cassette only, should type C:. The program then determines the start and end memory locations of the screen and then proceeds to dump the data, one byte at time, to your output file. This is the principle used by virtually all graphics eight or drawing or graph storage systems, and is pretty straightforward. To reload a particular file Program 2 can be used.

```
1 REM ........PROGRAM 2........
2 REM Load Screen with no compression
10 DIM F$(15)
20 PRINT "Which File->";:INPUT F$
100 GRAPHICS 8+16:COLOR 1
110 GOSUB 27000
1000 GOTO 1000
27000 REM .....Load Screen Data.....
27100 OPEN #1,4,0,F$
27200 SS=PEEK(88)+256*PEEK(89)
27210 ES=SS+7680
27300 FOR I=SS TO ES
27310 GET #1,X:POKE I,X
27400 NEXT I
27500 CLOSE #1:RETURN
```

If you examine the file you have just created you will see that it is nearly all empty spaces, or zero bytes. Of the 7680 bytes saved only about 5% in this case are actually non-zero values! One improvement which has been made by some GRAPHICS 8 utilities is to incorporate a machine code routine to save and reload the data. This relieves the time factor to a certain extent on disk systems, but the problem of inefficient use of storage space still persists.

```
1 REM ........PROGRAM 3........
2 REM Save Screen with compression V1
10 DIM F$(15)
20 PRINT "Which File->";:INPUT F$
100 GRAPHICS 8+16:COLOR 1
110 PLOT 0,0:DRAWTO 319,191
120 PLOT 0,191:DRAWTO 319,0
130 GOSUB 26000:STOP
26000 REM .....Dump Screen Data.....
26100 OPEN #1,8,0,F$
26200 SS=PEEK(88)+256*PEEK(89)
26210 ES=SS+7680:C=0
26300 FOR I=SS TO ES
26310 X=PEEK(I)
26320 IF X>0 OR C>254 THEN PUT #1,C:PU
T #1,X:C=1:GOTO 26400
26330 C=C+1
26400 NEXT I
26500 CLOSE #1:RETURN
```

The solution is to have a routine which, when it comes across a series of blank bytes on the screen simply counts these zero bytes and puts one byte to the output file whose value represents this count. Now type in and RUN Program 3.

```
1 REM ........PROGRAM 4........
2 REM Load Screen with compression V1
10 DIM F$(15)
20 PRINT "Which File->";:INPUT F$
100 GRAPHICS 8+16:COLOR 1
110 GOSUB 27000
1000 GOTO 1000
27000 REM .....Load Screen Data.....
27100 OPEN #1,4,0,F$
27200 SS=PEEK(88)+256*PEEK(89)
27210 P=SS:TRAP 27500
27300 GET #1,C:GET #1,X
27310 P=P+C:POKE P,X
27400 GOTO 27300
27500 CLOSE #1:RETURN
```

You should notice that, in comparison to the first example, very few blocks or sectors of data were saved. In fact, for this particular shape on the screen, only 7! You may at this stage doubt that this will actually work, but just type in Program 4, rewind your tape if you are using cassette, and RUN it.

If all has gone well, you should now have the cross back on your screen. Two advantages of this system should now have become apparent. Firstly, I have reduced the storage file size to an

incredible 10% of its former size, and secondly, because we are only restoring the bytes with data in them, the picure is redrawn at incredible speed!

In practice, however, you will not usually achieve such amazing results since our example picture was a very simple one. The more 'dense' the picture, the less efficient the new system will become, for it relies on compacting the contiguous block of zero bytes which represent blank areas on the screen.

So How Does It Work?

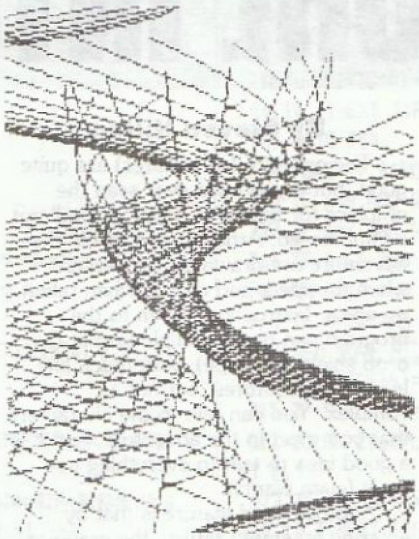
The computer stores a graphics eight screen in a continuous block of memory, roughly 8K of bytes somewhere near the top of memory (the locations 88 and 89 tell us exactly where). When a graphics 8 screen is first called, it starts as a blank screen, none of the pixels are lit, and this is achieved by making all the screen bytes zero in value. When you plot a point on the screen the computer determines which of its memory locations corresponds to the particular pixel you want to be lit, and alters its value accordingly. Each byte of screen data contains the information for eight pixels of the display, one per Bit. I won't go into the precise way in which this happens since this has been covered in previous articles, but it is sufficient to remember that if a screen data memory location is greater than zero then one or more of the corresponding pixels will be lit, but if it is zero then that small part of the screen (eight dots wide) will be blank.

Our compressor Program 3 looks at each of the screen memory locations in succession, and tests it to see if it contains zero. If it does, then it increments the zero byte count (variable C) and then loops back to look at the next byte. Once a non-zero byte has been found the program takes the value in C (the zero byte counter), and sends it to the output device, followed immediately by the non-zero screen byte which it found. It then resets the zero-bytes found variable (C) and repeats the operation, stopping when the end of the screen data has been reached.

The result is an output file which is a block of pairs of numbers, the first of each corresponding to the number (if any) of contiguous zero bytes, and the second being an actual screen data byte. In practice, because of a short-cut I have taken to speed up the loading of this file, if there were no zero bytes on a particular pair then the first byte would be equal to 1.

As you may appreciate, since this program relies upon blocks of blank locations, the more dense the picture is, the less efficient it will be. Take the extreme example where there are little or no zero bytes, we would be storing two bytes for every one screen byte! In practice, however, most line drawings or graphs use only a tiny percentage of the surface area of the screen, and so will benefit greatly from this method of compression, typically being reduced to around 10% to 30% of their original storage space. The most remarkable point about this method is its sheer simplicity - just a couple of extra commands more than the usual way of dumping a graphics eight screen to a storage file.

```
1 REM ........PROGRAM 5........
2 REM Save Screen with compression V2
10 DIM F$(15)
20 PRINT "Which File->";:INPUT F$
100 GRAPHICS 8+16:COLOR 1
110 PLOT 0,0:DRAWTO 319,191
120 PLOT 0,191:DRAWTO 319,0
130 GOSUB 26000:STOP
26000 REM .....Dump Screen Data.....
26100 OPEN #1,8,0,F$
26200 SS=PEEK(88)+256*PEEK(89)
26210 ES=SS+7680:C=0
26300 FOR I=SS TO ES
26310 X=PEEK(I)
26320 IF X=0 THEN C=C+1:IF C=255 THEN
PUT #1,0:PUT #1,255:C=0:GOTO 26400
26330 IF X>0 AND C>0 THEN PUT #1,0:PUT
 #1,C:C=0
26340 IF X>0 THEN PUT #1,X
26400 NEXT I
26500 CLOSE #1:RETURN
```



There is however another way of compressing this type of data, you could simply place a zero into our output file to indicate that the next number is one which tells how many successive zeros were found. Program 5 will save a picture using this method, and Program 6 will restore the picture.

Although at first sight this second method seems to be much more efficient, in reality this is a more complex matter to decide. If you used Program 5 and Program 6 on very rarified pictures such as graphs or line drawings, then you would find that you were storing three bytes for almost every non-zero screen byte you came across. A zero flag, a byte indicating how many zero bytes were counted and then the actual screen data byte. On pictures with very dense portions, you will find that this second method really comes into its own, with a considerable advantage over the first (Programs 3 & 4). You will need to experiment in order to develop a 'feel' for which method to use in particular situations.

Using the Programs

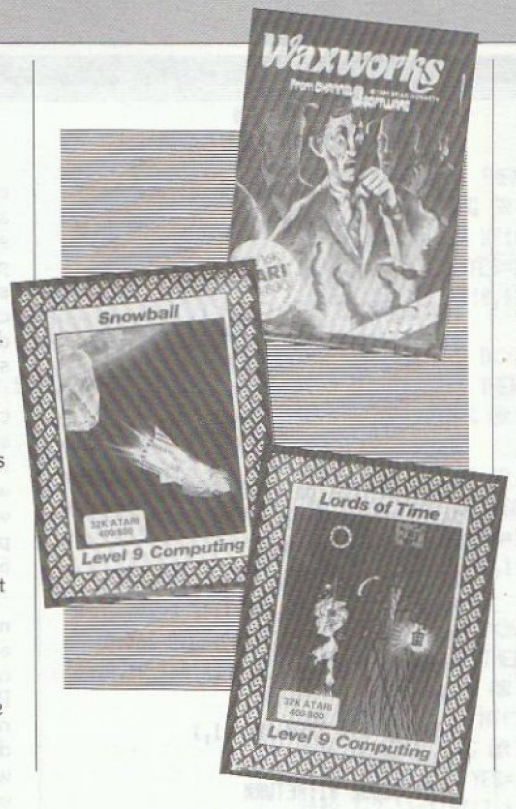
You will notice that in my examples I have kept the working parts of the programs as subroutines with high line numbers. Only the line numbers 26000 and above need be placed into your own programs, and you simply equate F$ to the storage device and file name (if applicable) such as "C:" or "D:filename.ext." Any device which can read and write data could be used, even the RS232 ports. Using the 850 interface you could transfer the pictures between any two Ataris, even via telephone if you have a modem!

Use of the compressor programs need not be restricted to graphics 8 screens either, since any data which has such intermittent blocks of zeros could benefit from it's application.

I am quite sure that these cannot be the only good methods of compressing graphics 8 data - you may know of some yourself. One way which might work is to store some form of table of numbers first, which perhaps could itself be a bit-map of the function (zero counters or data bytes) of each of the stored bytes. The possibilities are many and varied, and I would be very interested to hear of any ideas you may have. Any novel or useful alternatives will be printed in future issues. So put your thinking caps on!! I would also like to hear of any uses you have for data compression apart from graphics data.

```
1 REM ........PROGRAM 6........
2 REM Load Screen with compression V2
10 DIM F$(15)
20 PRINT "Which File->";:INPUT F$
100 GRAPHICS 8+16:COLOR 1
110 GOSUB 27000
1000 GOTO 1000
27000 REM .....Load Screen Data.....
27100 OPEN #1,4,0,F$
27200 SS=PEEK(88)+256*PEEK(89)
27210 P=SS-1:TRAP 27500
27300 GET #1,X
27310 IF X=0 THEN GET #1,X:P=P+X:GOTO
27400
27320 P=P+1:POKE P,X
27400 GOTO 27300
27500 CLOSE #1:RETURN
```

ADVENTURE INTO ATARI



Last issue I suggested that this column could be a mix of reviews and hints. Since I received no contrary submonitions, that is what it's going to be. So, on with the reviews . . .

The Incredible Hulk

by Scott Adams. (24K tape £9.95, 48K disk with graphics £19.95)

Unlike the original SAGA's, this adventure has Micro-Painter quality pictures that occupy most of the screen. There is no annoying flicker, and the artwork just has to be seen to be believed. The graphics display is toggled with the text display by just touching the keyboard, so the last few commands can be reviewed, as can the place descriptions.

The object of the game is that you, either as Bruce Banner or his alter ego, the Hulk, must retrieve 13 gems and put them in the correct place. You start as Bruce Banner, tied hand and foot to a chair. Once you escape, you find you are in a spherical dome, where to venture outside gets you crushed by the high gravity. Of course, there is a way round this problem, as there is around the killer bees, the underground room and eventually even the Chief Examiner who sits critically watching your progress (or lack of it). Interestingly enough, when you are killed, you end up in a limbo state whereby you can descend to your starting place without penalty. This feature is much appreciated!

As you would expect, this is by no means an easy adventure. However, the first six or so gems are very easily retrieved – then the problems start! The only grouse about this otherwise excellent program is that Scott Adams has used the same miserable parser as was in his other 13 adventures. Response time is of course slow in the disk version because each picture is loaded separately.

Blade of Blackpoole

by Sirius (48K Diskette)

The Blade of Blackpoole is semi-graphic, fantasy-type adventure published by Sirius Software in the States, and costs around £25. The object of the game is to recover the magical sword, MYRAGLYM, and return it to the altar from whence it was stolen. When I purchased this adventure, I was assured by the shop assistant that it was one of the hardest of its kind available. As it happened I was stumped by one particular problem in the early stages of the game for a couple of months. Once that was solved I completed it within a few days.

By Steven Hillen

The graphics (or pictures) are quite good, but if you have ever seen the graphics for Sands of Egypt you will not be impressed. Articles used within the adventure are drawn on the screen when dropped, and erased when taken up again. Instructions given to the program can be in multiple format (i.e. drop shield and rope), very much like Infocom adventures, but not as complex. You can talk to the characters that you meet in the adventure, and it is a good idea to talk to everything you meet (even plants).

Another good feature is that by entering an empty return, the graphics disappears leaving a screen full of your previous locations. This is a good idea if you have forgotten to map your course and wish to look back. Also, it can be played as a text only game, and pressing return again gets the picture back.

For anyone starting out in adventures, this is probably the one for you as a second or third game. But for anyone who has just battled their way through the Zork trilogy or something similar, it will be a disappointment.

I would like to thank Mr. A. Lusher for sending me the review of Blade of Blackpoole which I have printed above.

Waxworks

by Channel 8 Software. (16K cassette £9.95)

Channel 8 now supply a large series of 16K adventures, all on cassette. They all share a similar screen display, which is very "Scott Adams-ish" in that the descriptions of rooms are limited, in the main, to names and objects present. However, the parsers on these adventures are very fast, as is the updating of the screen display.

Waxworks is one of the their more recent titles, and the scenario is that you have dozed off unnoticed by the Waxworks attendant, who has inadvertantly locked you in. Upon waking, you realise that it would be safer for you to escape as soon as possible! A cursory glance around the leisure lounge reveals a few useful objects that help in your exploration. Obviously, this is quite a challenge as certain exhibits are out to stop you.

As a summary then, this is a very novel setting for an adventure and the game is written with professional pride. If you are limited to a 16K cassette system, then these are without doubt the best adventures out, beating the Other Venture series hollow. Lastly thanks go to Channel 8 for allowing me to use a review copy.

Snowball

by Level 9. (32K cassette £9.90).

If you are into large adventures, this is the one for you. Data compression techniques have enabled Level 9 to create an adventure with over 7000 rooms! Don't worry though – you do not need to visit every single one of them, but mapping this game out could take some time! The especially good thing about this adventure is the detail of description for each location – it is easy to imagine yourself as the hero, Kim Kimberley. The adventure booklet supplied with the game gives the background galactic scene. Snowball 9 is the inter-stellar transport ship carrying thousands of human colonists from Earth – you are on the ship to ensure that all goes well.

The game starts with your being awoken from cryogenic suspension. Obviously something is wrong! Once you have escaped from the coffin, and avoided the now homicidal robots, you must endeavour to switch Snowball 9 back to autopilot, as someone has aimed the ship at a star. The solution to this adventure is devious indeed, and Level 9 are to be congratulated on firstly writing such an excellent adventure, and secondly for including a "Help" envelope. If you are really stuck somewhere, then you can mail a letter in this envelope for a free clue. Also, Level 9 supply excellent Hint Sheets for those of you who, like me, have little perseverance.

If this sort of Sci-fi is not your scene then perhaps you should try "Lords of Time", their latest release. In this adventure, you are required to obtain objects from 9 different time-zones, ranging from pre-history to the far future, in order to defeat the evil Timelords. Your time-travel machine is a cavernous grandfather clock which has an amusing method of operation.

Level 9 cassettes have separate versions on either side, and if you still have any problems loading, then they ask to be notified so that the problem may be rectified. For value for money and excellent after-sale support, Level 9 adventures have yet to be bettered. Again, a word of thanks to Level 9 for allowing me to have review copies.

Planetfall

by Infocom. (32K disk £35.95)

This is quite simply, the best adventure I have ever played. The superb parser eliminates most rephrasing problems, and is also very fast in responding. Once again, the room descriptions are exceptionally graphic, and there is a great deal of humour within the program. Attention to detail is unusually meticulous and overall the game is very enjoyable.

You are a lowly Ensign 7th Class in the Stellar Patrol, on board the Feinstein under the command of Ensign Blather. As you note from the diary supplied in the game package, you and Blather do not get on. Blather is in fact intent on de-meriting you. Luckily, if you can avoid being thrown into the ship's brig for punching Blather, you alone can escape from the destruction of both Blather and the Feinstein.

As you attempt to move out of the escape pod once you have landed on one of the two visible islands on a nearby planet, the pod topples into the sea. If you survive this, you go down with flu before being able to explore the deserted complex. To do this, you must collect access cards to permit you to use elevators, underground railways, and teleport booths. Floyd, a robot you discover is a good friend and eventually gives his life for you – so take good care of him!

Infocom have collected a superb series of logical puzzles together, along with humorous and realistic characters and events in producing this excellent package. Even though it is rather expensive, it is worth every penny, for even non-adventuring friends of mine have commented that it is an extremely playable and enjoyable game.

Adventure Call

Now to the section of this column that is reserved for your queries and solutions to adventures. The number of replies I received since the last issue comfortably failed to reach double figures. Understandably, this was not too pleasing, but thanks go out to those who did write in. Remember that I would like to hear from you if you have an adventure problem or if you've solved an adventure, or even if you've just got suggestions for the column, e.g. you might like to see how to write adventures. But please let's have a little more response!

Right then, I've come up with a little coding routine that will prevent you from finding out answers to problems unless you really want to know. Hopefully this will prevent unwanted clues from spoiling games you have not yet attempted.

All you have to do is type in the 3 lines listed below, save a version out to cassette or disk, and then look through to the clue you want, typing this in as line 30 TEXT$=" characters . . . " The Adventurer who supplied a clue is named alongside. Finally, there is a list of questions that as of now are unanswered. If you can answer any, please write in so I can print solutions in the next issue. The decoding routine is as follows:

```
10 DIM TEXT$(100),MC$(21):FOR A=1
TO 21:READ D:MC$(A)=CHR$(D):
NEXT A
20 DATA 104,104,104,168,136,104,133,
204,104,133,203,177,203,73,6,145,203,
136,16,247,96
40 X=USR(ADR(MC$),LEN(TEXT$),
ADR(TEXT$)):?CHR$(125):?:?TEXT$
```

Answered Questions

Pyramid of Doom.
Where is the iron glove? (Solution by P. Lister).
30 TEXT$="OH&RNC&DI ^ &QORN &RNC&DIHCU"

Earthquake San Francisco 1906
How do you free yourself after the first earthquake?
30 TEXT$="SUC&ETIQDGT&@TIK &SHBCT&DCB"

Suspended
Can you get over the first step?
30 TEXT$="SUC&TGKV&@TIK&USD &USVVJ_&TIIK"

Mystery Fun House
What are spectacles for?
30 TEXT$="JIMM&GR&KOTTIT"
Stuck with mermaid?
30 TEXT$="JIMM&GR&NCT&NGOT"
Stuck in Pit?
30 TEXT$="KIPC&RTGKVIJOHC"

Unanswered Questions

Starcross
How do you remove the rod from the sphere without using the gun?

Golden Voyage
Can't find enough small stones to make second stone tablet.

Savage Island Pt2
Can't kill dinosaurs nor change back from neanderthal.

Escape from Pulsar 7
Can't remove cups from Captain's cabin ceiling.

Sands of Egypt
What is the command that gets you out of the boat once you have the ladder?

Earthquake 1906
How do you get past the opera house?

Feasibility Experiment
Can't get past guard in the mine.

Ten Little Indians
Can't do anything with couch or wall safe.

CRACKING THE CODE

By Keith Mayhew and Roy Smith Part 3

Hopefully, the first two parts of this series have given you a good grounding of the basics of the machine and a brief insight into the mathematics of binary and hex numbers. From this point we can move on to some examples and explanations of Assembly Language programs. We assume that you are the proud owner of an Assembler package such as the ATARI Assembler/Editor cartridge for which all of our listings will be written, but if you own a different one then we shall give a brief idea on how to convert the few offending commands. If you do not have an Assembler, you really do need one if you intend to write seriously in Assembly Language. As an alternative to buying a commercially available Assembler, there are two available from the Club's software library through the donation scheme at £1.00 per copy ('USERCOMP' is highly recommended).

All examples given in this and future articles are only intended to be fully working programs if they are listed with line numbers. Programs without these are purely to aid learning and will generally not work as they stand.

Mnemonics.

All of the 6502's instructions are given three-letter labels called mnemonics. The reason for this is simply as an aid to your memory so that you do not have to remember a string of numbers but only a short letter sequence. When using an Assembler you are in fact writing in mnemonics and when the program is 'assembled' it is converted into its numerical equivalents or the actual 'machine code'. So, Assembly Language is Machine Code written in an easy to remember format.

Getting Started!

The first mnemonic we will discuss is the instruction 'LDA'. This means LoaD the Accumulator. In other words when this instruction is executed a number between 0 and 255 will be deposited into the A register of the 6502. Where the number actually comes from is determined by the operand byte(s), remember from Part 2 that the instruction 'LDA' is an op-code and the bytes following it are it's operand bytes. The op-code 'LDA' can be used in many different addressing modes, the simplest is 'immediate' addressing:

```
LDA #40
```

Here we have loaded 'A' with the decimal number 40. The '#' symbol informs the assembler to produce an op-code which will mean 'load the accumulator in immediate mode'. It would be more usual to write the instruction as:

```
LDA #$28
```

This has exactly the same effect as the previous example, only here we have written 40 decimal as 28 hex i.e. $28, where the '$' means hex.

The lowest number that can be loaded into 'A' is 0 and the largest is 255 or $FF (remember the 'A' register is an eight bit register). When this line is assembled by the Assembler it converts our mnemonic code into a single hexadecimal number, so that our line would look like this:

```
$39 $28
```

Where $39 is the op-code and $28 is the operand byte. Note most Assemblers omit the '$' symbol. These two numbers, $39 and $28, would be stored into sequential memory locations ready to be executed. It is a point worth remembering that commercial Assemblers usually produce the machine code listing in hex format, so becoming familiar with hex notation is essential. Consider the following:

```
LDA $28
```

You've probably spotted that the '#' symbol is missing, this is because we are no longer using immediate mode but we are using zero-page or short addressing. In this mode the data to be loaded into 'A' is found at the memory location of $28, and as this is a number below 255 or $FF it will fall in the first page of memory (page 0).

For example if the contents of memory location $28 was $6B, then $6B would be read from location $28 and placed in 'A' (once again the number read can only be between 0 and 255). When our line is now assembled the operand byte remains the same but as a different addressing mode is being used then a different form of the op-code is produced thus:

```
$A5 $28
```

Supposing the data, $6B, was not to be found in page 0 but was located somewhere else in memory, we would need to use absolute addressing to access it. In this case two operand bytes would be needed to cover the complete memory from 0 to $FFFF. Note that the programmer does not need to identify between these modes as once the specified location exceeds page 0 then the Assembler automatically uses absolute addressing. Let's suppose location $24E5 contains our data, $6B, then to load this into the 'A' register we would write the following:

```
LDA $24E5
```

Once again, when assembled, the op-code will be different because of the mode change and also an extra byte will be added for the extended addressing area, as shown below:

```
$AD $E5 $24
```

As you can see the op-code is $AD but the address has been assembled back to front, so that the low byte is placed first then the high byte next. This is done because the 6502 expects them in this form, but note that the friendly Assembler allows you to write them the normal way round.

It is worth noting that the Assembler will always produce a pair of hex digits for each byte, so that although you are allowed to write a hex number in an odd number of digits the Assembler will pad out the leading zero for you. For example:

LDA $5 will be assembled as $A5 $05.

LDA $B3C will be assembled as $AD $3C $0B.

We have shown the accumulator being loaded in three addressing modes, as you know there are more than this but we will come to these later. The other two registers, 'X' and 'Y', are loaded in exactly the same manner as the 'A' register except that different mnemonics are used i.e. 'LDX' and 'LDY'. For example:

LDX #$E4 (Loads 'X' in immediate mode).

LDY $3B (Loads 'Y' in page-0 mode).

LDX $6E43 (Loads 'X' in absolute mode).

Store it away

Now we have seen how to load some data into these three registers, we will now show the instruction to store that data into memory. Once again, all three registers have their own mnemonic and they are: 'STA', 'STX' and 'STY'. Note that unlike 'load', the data must be stored in a memory location, so obviously immediate mode cannot be used. The following examples show these instructions in the other two addressing modes we have covered so far:

STA $12 (Store 'A' in location $12).

STX $4C5F (Store 'X' in location $4C5F).

STY $10EB (Store 'Y' in location $10EB).

Here we have stored 'A' in page-0 mode and 'X' and 'Y' in absolute mode. Suppose we wish to store a number into a specific location, we would go about it by loading the number into a register and then storing that register into the desired location, as shown below:

```
LDA #$60
STA $0400
```

We have loaded 'A' with our number ($60) in immediate mode and then stored that information into a location ($0400).

Now suppose that we wish to move some data from one location to another. It would look like this:

```
LDX $7E43
STX $030B
```

Here we have loaded 'X' with the data in location $7E43 (in absolute mode) and stored it at $030B. In these two examples, where we have stored some data and moved some data, the registers used could just as well have been any of the three ('A', 'X' or 'Y').

Logically speaking

Included in the instruction set of the 6502 are three logical operators with which numbers can be manipulated.

These instructions are 'And', 'Or' and 'Exclusive-Or'. Their mnemonics are 'AND', 'ORA' and 'EOR' respectively.

These three op-codes only operate with the 'A' register and not with the 'X' and 'Y' registers. Note, due to the standard that all 6502 mnemonics have three letters, the 'Or' function is designated 'ORA' for convenience.

The purpose of the 'AND' instruction is to reset specific bits in the 'A' register to a '0'. The 'AND' function has four possible states, these are:

```
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1
```

As you can see the result of ANDing two bits together is only '1' if both the bits are '1'. If this principle is extended to cover 8-bits (1 byte) then we can use the results to obtain the bit pattern we desire.

```
        10110010    (byte 1).
AND     01110100    (byte 2).
        00110000    (result).
```

Here we have ANDed byte 2 with byte 1 to get our result. Referring to the table, and working our way from right to left; taking bit zero of byte 1 and byte 2 we have 0 AND 0 which results in 0, next we have bit 1 from both bytes, giving 1 AND 0 = 0, then 0 AND 1 =0, etc. until we have completed all eight bits.

Byte 2 is often referred to as a 'mask' which acts like a sieve, where a '1' in the mask is like a hole through which the bits of byte 1 pass. Looking at our example you can see that the bits in byte 1 above a '1' in byte 2 are copied into the result, also where a '0' is in the mask a '0' is in the result.

Suppose we wish to ensure that bits 5 and 7 in the 'A' register are reset to zero, then we would use the following mask:

```
01011111
```

ANDing this mask onto the 'A' register, no matter what bit pattern was there, because of the zero's in bits 5 and 7 of the mask, the result will also have zero in bits 5 and 7, and the other bits will remain the same.

The result of the ANDing is automatically returned into the 'A' register. Suppose the 'A' register was loaded with the binary pattern: 10011010 which in hex is 9A, and was ANDed with our mask of: 01011111 (5F hex) so that bits 5 and 7 are zeroed to give the result of: 00011010 (1A hex).

```
LDA #$9A
AND #$5F
STA $067D
```

Here we have loaded 'A' with $9A which is the hex equivalent of our original bit pattern. We have ANDed it with our mask, $5F, so the result is $1A which is now in the 'A' register.

Then we store our new bit pattern into a memory location at $067D. We have used immediate mode to load the 'A' register and to 'AND' the mask, but we could have used any of the three addressing modes described so far.

The second logical operator 'OR', is used to set specific bits in the result to a '1'. The 'OR' function has four possible states:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

From this table you can see that when two bits are ORed together, the result is always one except in the case of both bits being zero's. Here is an example showing two bytes being ORed together:

```
        01011100    (byte 1).
OR      01101001    (byte 2).
        01111101    (result).
```

You can see that only bits one and seven of bytes 1 and 2 contain matching zero's thus giving '0' in the result. Suppose the accumulator contains the byte: 10100011 (A3 hex) and we wish to ensure that bits 6, 5, 2 and 0 are set to a '1' then we would need to OR it with the following byte: 01100101 (65 hex) where bits 6, 5, 2 and 0 have a '1' in them. Conversely the other bits must be '0's so that the result will be a copy of those bits from the first byte. The result of ORing these two bytes is: 11100111 (E7 hex), this is automatically returned into the 'A' register.

We would write this in assembler thus:

```
LDA #$A3
ORA #$65
STA $4B0D
```

This loads the 'A' register with $A3, ORs it with $65, giving the result $E7 in the 'A' register, which is then stored in location $4B0D.

The final logical operator is 'Exclusive-Or' which is used to toggle specific bits of the accumulator between the two possible states i.e. '0' and '1'. In other words if the bit was a '0' and it was toggled then it would become a '1', if it was toggled again then it would revert back to a '0'. The four possible states of the Ex-OR function are:

```
0 Ex-OR 0 = 0
0 Ex-OR 1 = 1
1 Ex-OR 0 = 1
1 Ex-OR 1 = 0
```

As you can see the result is '0' when both bits are the same, i.e. '0 Ex-OR 0' and '1 Ex-OR 1'. The bits to be toggled are Ex-ORed by a '1' in the appropriate bit of the second byte. Thus if we have 11111111 (FF hex) as the second byte then Ex-ORing any byte will invert every bit in it, this is called the complement, here is an example of a complement:

```
        00101110    (byte 1).
EOR     11111111    (byte 2).
        11010001    (result).
```

Note that the result is the exact opposite of the first byte, and also that if it was Ex-ORed with $FF again then the result would lead us back to our original byte. Similarly if we only wish to toggle specific bits then we only set those bits in the second byte, here is an example:

```
        00011101    (byte 1).
EOR     10000000    (byte 2).
        10011101    (result).
```

In the above example only bit 7 has been toggled with all the other bits remaining as in byte 1. If this result was Ex-ORed with byte 2 again then bit 7 would be toggled back to a zero giving the original byte 1. In assembly language:

```
LDA #$1D
EOR #$80
STA $35E0
EOR #$80
```

We have loaded the accumulator with $1D, toggled bit 7, then stored the result ($9D) in location $35E0, and finally toggled bit 7 back to give $1D in the 'A' register again.

More or Less?

In Part 2, we showed some examples of adding unsigned binary numbers and we mentioned that when the result of adding two 8 bit numbers together is greater than 8 bits, then the ninth bit is placed into the carry flag.

The 6502 does not have an instruction which will add together 2 bytes directly, ignoring the carry flag. Instead, the 6502 has the instruction 'add with carry', whose mnemonic is 'ADC', which will add 2 bytes together and also add the contents of the carry flag to the result, i,e. adds nothing if C=0 and adds one if C=1. Suppose we wish to add 28 hex to 14 hex, then the first instruction would be to load the accumulator with $14 and then add with carry $28, note we could have loaded with $28 and then added $14, as it would be the same result.

```
LDA #$14
ADC #$28
```

Notice that 'ADC' has been used in the immediate mode and the result is stored back into the accumulator, as are all operations on the accumulator.

The result of adding these two numbers together could be one of two answers, the reason for this is that the state of the carry flag is taken into account and we did not know if it was cleared or set i.e. '0' or '1'. Therefore to ensure we end up with the correct result we must clear the carry flag before adding the numbers together, the instruction to do this is called 'clear the carry', whose mnemonic is 'CLC'. So, the proper version of this program now reads!

```
CLC
LDA #$14
ADC #$28
```

If the result of an addition is less than 256 (decimal) then only eight bits are used and the carry flag, the ninth bit, will be set to zero again.

With a result of greater than 255 the carry flag will be set to a one. If we are doing an addition of more than eight bits then this previous state of the carry flag must be taken into account.

Now we will write a program which will take two sixteen bit numbers and add them together, storing the result as another sixteen bit number, note that the result could actually be a seventeen bit number (including the carry) but we shall ignore this in our example as the two numbers will be reasonably small and therefore not generate a seventeenth bit.

In our example the first sixteen bit number is stored in two eight bit locations, $0600 and $0601 with the low byte stored first, and the second number is in locations $0602 and $0603.

The sixteen bits of the result will be stored in locations $0604 and $0605 with the seventeenth bit still in the carry. The program first clears the carry so that it is zero and then adds the two low bytes of each sixteen bit number together storing the accumulator into the low part of the sixteen bit result, as follows:

```
CLC
LDA $0600
ADC $0602
STA $0604
```

Now we must do the same for the two high bytes storing them back into the high byte of the result. We will not clear the carry this time because we know that whatever the result, whether it is more or less than 255 the correct carry will be transfered into the sum of the high bytes, so the rest of the program would read:

```
LDA $0601
ADC $0603
STA $0605
```

At this point the carry flag would reflect the status of the seventeenth bit, normally zero if the two sixteen bit numbers are small enough.

Let us assume that the first number is $14F9 and the second number is $311A, the contents of the first four locations will now be:

```
$0600 = $F9
$0601 = $14
$0602 = $1A
$0603 = $31
```

Adding the two low bytes together we have $F9+$1A = $113. The $13 will be in the accumulator and the '1' is in the carry flag. The two high bytes added together give $14+$31 = $45, but as the carry flag is now set another one is added to this byte to give $46 in the result with no carry in the seventeenth bit (the carry flag). So, locations $0604 and $0605 will contain $13 and $46 respectively and so the result will be $4613.

The same technique can be used for even larger additions if necessary, e.g. 24, 32 bits etc. If when adding two numbers a carry is generated for the highest bit, i.e. the result exceeds the number of bits being used, then your program could check for this state and indicate an overflow error, using an instruction which will be covered later.

As with addition there is no instruction that will subtract two bytes directly, but there is an instruction that will 'subtract with carry', whose mnemonic is 'SBC' which will subtract two bytes and also subtract the complement, or opposite, of the carry flag. If we wish to subtract $05 from $2A we would load the accumulator with $2A and then subtract with carry $05, but as in the case of addition where we clear the carry first, we need to set the carry to one. With the carry set, the complement of this is subtracted from our result, i.e. zero is subtracted leaving us with the desired answer.

The following program subtracts $05 from $2A and first sets the carry with the 'SEC' command to ensure the correct result:

```
SEC
LDA #$2A
SBC #$05
```

You may be asking yourself why you do not clear the carry and let the 6502 subtract that directly rather than setting the carry to one and then subtracting the complement, which is also zero? The answer lies in the fact that the 6502 can only perform addition, and therefore has to use the two's complement of the number to be subtracted i.e. the negative of that number, and adds it to the contents of the accumulator. The following shows the 'subtraction' of the two numbers:

```
  00101010 ($2A)
(+) 11111011 (-$05) two's complement of $05.
  ----------
(1) 00100101 ($25 ignoring carry of one).
```

We can see that in a non-borrow subtraction, i.e. a small number from a large number, the carry is left set as a consequence and is interpreted as 'subtract zero' by any further subtractions. So remember that in subtractions when the carry flag is set to one it means 'subtract zero' and when the carry is cleared to zero it means 'subtract one'.

Let's subtract two sixteen bit numbers and see how the carry flag performs. We will subtract $32CA from $6B12. The main number is stored in locations $0400 and $0401 and the number to be subtracted from this is stored in locations $0402 and $0403, the result will be left in locations $0404 and $0405. The first part of our program sets the carry then subtracts the two low bytes and stores the number in the low part of the result:

```
SEC
LDA $0400
SBC $0402
STA $0404
```

Thus the accumulator was loaded with $12 from location $0400 then $CA from location $0402 was subtracted from the accumulator and the result, which is $48 is stored in location $0404. Because the number we subtracted was larger than the accumulator's contents, a borrow of one was made from the high byte so that the $12 became $112 (the one that was borrowed is worth 256 decimal or $100), thus when $CA is subtracted from $112 the answer is $48 and to indicate that a borrow occured the carry flag would be cleared. The second part of the program will subtract the two high bytes in locations $0401 and $0403 leaving the result in location $0405, again we do not have to worry about the condition of the carry in the program as it will be treated correctly by the 6502 in the next subtraction:

```
LDA $0401
SBC $0403
STA $0405
```

Here the accumulator is loaded with $6B from location $0401 and $32 is subtracted (location $0403) from the accumulator leaving an answer of $39, but a further one is subtracted due to the carry being zero (one is the complement of zero), so the result is $38 which is stored in location $0405. The final result of subtracting $32CA from $6B12 is therefore $3848.

Decimal Mode

As mentioned before the 6502 has a decimal mode and is activated when the 'D' flag of the processor status byte is set to one. This is achieved by using the instruction whose mnemonic is 'SED' or 'set the decimal mode', to revert back to the normal binary mode the 'CLD' instruction is used which 'clears the decimal mode'. The 'CLD' instruction should be used if you are unsure which mode you are currently in, thus ensuring that any additions or subtractions do occur in binary mode. Once the decimal mode has been cleared there is no need to use the 'CLD' command again unless you have used decimal arithmetic in your program. Here is an example of decimal mode addition:

```
SED
CLC
LDA #$48
ADC #$25
STA $0580
CLD
```

The program starts by ensuring decimal mode is set and that the carry is cleared for the addition. The accumulator is loaded with $48 and $25 is added to it and the result is stored in location $0580, and finally we clear the decimal mode so we are back to binary for further arithmetic. Because the decimal mode has been implemented the two numbers $48 and $25 are added together just like a normal sum. So 5 is added to 8 giving 13 where the 3 is placed in the low half of the accumulator and the one is carried automatically into the high part of the sum, so we now have 4 plus 2 plus our carry of one equalling 7. So the result is $73. This internal carry between the high and the low parts has no effect on the carry flag. But any external carry generated from a decimal addition is treated in exactly the same way as for binary addition and is added into the next calculation. Subtraction follows the same rules as for binary subtraction and once again an internal carry is made automatically.

Up and Down

If we wish to only add or subtract one at a time, such as a counter, then instead of loading the accumulator with the previous number, either setting or clearing the carry then performing the addition or subtraction of one, then storing the result back into the counter, we could use one of the increment or decrement instructions. These instructions apply to the 'X' and 'Y' registers and any memory location. The increment instructions will add one to the contents of the register or location and the decrement instructions subtract one. If when using the increment instruction the result exceeds $FF then the result will 'wrap around' to zero and start again. Conversely, when using the decrement instruction it wraps around from zero to $FF.

The three increment instructions are 'INX', 'INY' and 'INC' which are increment the 'X' register, increment the 'Y' register and increment a memory location respectively. The three decrement instructions are 'DEX', 'DEY' and 'DEC' which are decrement the 'X' register, decrement the 'Y' register and decrement a memory location respectively. The state of the carry flag is not important before one of these instructions is implemented.

Move It

There are six instructions that allow us to transfer data quickly between the 'A', 'X', 'Y' and stack pointer (SP) registers. As we know, only the accumulator can be used for operations such as addition, subtraction and logical operations, so it is useful to be able to transfer the contents of either the 'X' or 'Y' registers directly into the accumulator to perform the necessary operations rather than storing it in memory and then loading it back into the 'A' register. The two instructions that perform this function are 'TXA' and 'TYA', which mean transfer the 'X' register into the 'A' register or transfer the 'Y' register into the 'A' register respectively. There are another two instructions to return the data back into the 'X' or 'Y' registers. These are 'TAX' and 'TAY', which mean transfer the 'A' register into the 'X' register or transfer the 'A' register into the 'Y' register respectively.

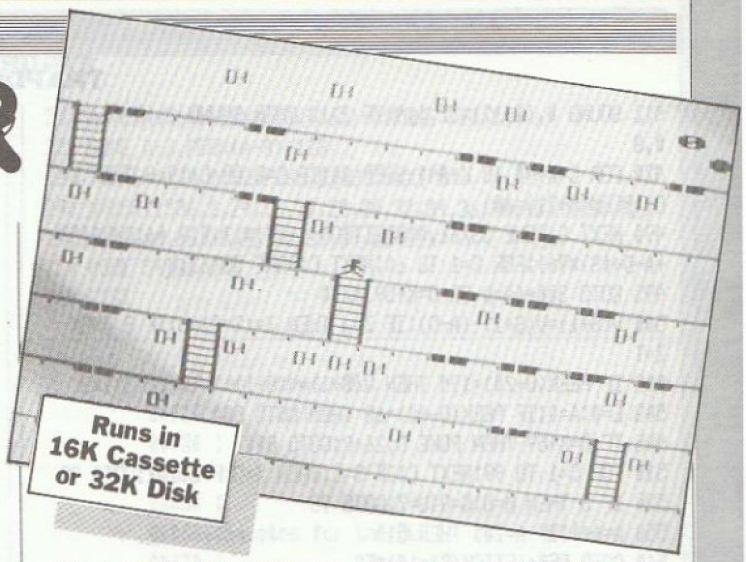
Lastly there are two specialised commands that transfer to and from the 'X' and 'SP' registers. The 'TSX' instruction transfers the stack pointer into the 'X' register so that it can be examined and the 'TXS' instruction is used to transfer the contents of the 'X' register into the stack pointer to set it to a specific value. However these two instructions are not used very often if at all in a program because the computer uses the 'TXS' instruction to set the stack pointer to $FF at power up, and should therefore not need to be altered.

Nothing To Do

The most unusual command in the 6502 instruction set is the 'NOP' instruction which actually means 'no operation' and the 6502 puts it's feet up for a while when this command is executed! No, seriously though, this command can be used as a 'debugging aid' to replace an instruction held in memory so that when the 'NOP' instruction(s) are encountered the program will 'skip' over them without crashing or corrupting the program.

In the next issue we will complete the instruction set for the 6502 and hopefully start on some programming techniques. Finally we suggest after reading this far, you perform a 'NOP' operation and go and put your feet up!

TRAPDOOR



by Bob Askew — Northampton

Every so often a game appears that on first glance looks like many other games you've seen before, but looks can be deceptive. Trapdoor is such a game. When the screen is drawn and you see the ladders leading to different levels, you think, "Oh! Yes, this is familiar". But once you and your partner start playing (it's a two player game), you can see some inovative thought has been put into the game.

Each player has his own little man who must be moved from the bottom left hand side of the screen, up the series of ladders to the top right hand side, to retrieve a flashing red ball which can then be bounced back down the ladders to try and squash your opponent. Each player is allowed to move up to 6 spaces at a time, which is decided by a push of the fire button which stops a rotating dice at the top of the screen. Once you have made your move however, you may have landed on a trapdoor. This is no worry unless your opponent, on his throw lands on a square above which is situated a "Keyswitch", if he does he can jump up and activate the switch, the result of this is that the trapdoor you are standing on opens, and you fall through. If you are unfortunate enough to be on a trapdoor which has other trapdoors in line with it on lower levels, all trapdoors will open and you will fall even further. If you or your opponent lands below a keyswitch, just by jumping up and activating the keyswitch gives a second throw of the dice, even if nobody is standing on a trapdoor.

Trapdoor is not just a race to the red ball, but a game of strategy too, a game of wits and cunning which will keep the whole family entertained for hours.

NOTE: In this program, anything which is underlined, should be entered in "INVERSE".

```
1 REM *TRAP DOOR* is a 2 player 2 joystick 'board' game.
                    Player 1-WHITE     Player 2-BLACK.
2 REM OBJECT-To squash opponent,by moving from START(bot
tom left)to FINISH(top right)and throwing ball,hitting h
im
3 REM MOVES-Rotating 'clock'(1-8)indicates players turn.
(white or black)          Stop clock with trig'
4 REM Clock will show No.of permitted moves.Move in any
direction(no backtracking,unless by edge)in same move.
5 REM SWITCHBOXES-If(at end of move),you're under switch
box,you may turn switch off(push stick up).
6 REM If opponent on a TRAPDOOR he falls through!Switchb
ox will turn off.  The more trap doors you dare land on
,
7 REM the slower your 'clock'.            BALL-may be th
rown(press trig'),if player can reach it.
8 REM EXTA MOVES-Awarded by turning switch off,or,having
 reached ball.      'Missed'balls reappear on moving
9 GRAPHICS 17:K=PEEK(88)+PEEK(89)*256:L=400:CB=PEEK(742)
*256-1280
10 DIM V(2):DIM A$(12):A$="TRAP DOOR   ":FOR A=0 TO 511:
POKE CB+A,PEEK(57344+A):NEXT A:POKE 712,6
15 FOR A=8 TO 127:READ B:POKE CB+A,B:NEXT A:FOR A=0 TO 3
1:READ B:POKE CB+208+A,B:NEXT A:POKE 709,196:POKE 710,18
20 POKE 711,0:POKE 708,140:FOR C=1 TO 12:FOR B=6 TO 7:PO
KE K-26,B:POSITION 5,1:? #6;A$(1,C):FOR G=1 TO 5:READ F
30 SOUND 0,F,10,9:SOUND 1,(F>0)*(F-5),10,6:FOR A=1 TO 7:
NEXT A:NEXT G:NEXT B:NEXT C:POSITION 8,6:? #6;"BY"
40 POSITION 5,9:? #6;"bob askew":FOR A=1 TO 39:READ B:SO
UND 0,B,10,9:SOUND 1,(B>0)*(B-5),10,6:FOR C=1 TO 9:NEXT
C
60 NEXT A:? #6;"":POKE K-26,6:POKE 756,CB/256:POKE 559,
0:FOR A=3 TO 23 STEP 4:COLOR 1:PLOT 0,A:DRAWTO 19,A:NEXT
 A
70 POKE K+39,140:POKE K+78,194:FOR C=1 TO 2:D=20:FOR X=1
 TO 5:D=D-4:A=INT((RND(0)*9)+1)*2:B=INT((RND(0)*X)+1)*4-
1
80 FOR B=B TO B+D STEP 4:COLOR 130:PLOT A,B:NEXT B:NEXT
X:NEXT C:COLOR 163:PLOT 1,3:DRAWTO 1,6:X=K+440:M=X
90 FOR B=7 TO 19 STEP 4:A=INT((RND(0)*9)+1)*2-1:COLOR 16
3:PLOT A,B:DRAWTO A,B+3:NEXT B:A=K+3:FOR B=0 TO 13
91 POKE A+B,(PEEK(A+B+60)<>194 AND RND(0)<0.3)*68:NEXT B
95 FOR B=4 TO 20 STEP 4:FOR A=0 TO 19:LOCATE A,B-1,H:LOC
ATE A,B+3,C
96 IF H=1 AND C<>130 AND RND(0)<0.3 THEN COLOR 4:PLOT A,
B
97 NEXT A:NEXT B:POKE K+400,0:POKE 559,34:POKE X,137:U=0
:T=0:F=6:G=6:V(1)=12:V(2)=12:GOTO INT(RND(0)*2)+98
98 S=0:O=X:J=G+192:W=M:Q=F:Z=T:GOSUB 250:GOTO 300
99 S=1:O=M:J=F:W=X:Q=G:Z=U:GOSUB 250:GOTO 300
250 POKE 77,0:R=(S)*192:D=0:A=INT(RND(0)*8)+1
255 A=A+1:A=A-(A=9)*8:POSITION 1+(S)*17,0:? #6;CHR$(A+48
+(S)*96)
260 SOUND 0,30+V(S+1),10,2:FOR C=1 TO V(S+1):NEXT C:POKE
 53761,0:IF STRIG(S)=0 THEN RETURN
280 GOTO 255
300 I=STICK(S)
305 IF I=7 AND Q<>8 AND PEEK(O+21)>0 AND O+1<>E THEN E=O
:O=O+1:Q=6:GOTO L
306 IF I=7 AND Q=8 AND (PEEK(O+21)=65 OR PEEK(O+21)=194)
 THEN Q=6:GOTO 305
310 IF I=11 AND Q<>8 AND PEEK(O+19)>0 AND O-1<>E THEN E=
O:O=O-1:Q=7:GOTO L
315 IF I=11 AND Q=8 AND (PEEK(O+19)=65 OR PEEK(O+19)=194
) THEN Q=7:GOTO 310
320 IF I=14 AND Q=8 AND O-20<>E AND (PEEK(O+1)=194 OR PE
EK(O+1)=65) THEN E=O:O=O-20:Q=6:GOTO L
330 IF I=14 AND PEEK(O-20)>0 AND O-20<>E AND O<>K+59 THE
N E=O:O=O-20:Q=8:GOTO L
340 IF I=13 AND O+20<>E AND (PEEK(O+20)=131 OR PEEK(O+20
)=J) THEN E=O:O=O+20:Q=8:Q=Q-(PEEK(O+20)=65)*2:GOTO L
350 GOTO 300
400 POKE E,Z:Z=PEEK(O):IF PEEK(W)=131 OR PEEK(W)=0 THEN
POKE W,J
405 O=O+1:POKE O,Q+R:POKE (O=W)*O,137
410 IF PEEK(K+39)=0 THEN POKE K+39,140:FOR C=9 TO 0 STEP
 -0.5:SOUND 0,15,2,C:NEXT C
```

```
412 SOUND 0,(I<12)*32,1,4:IF I>12 THEN SOUND 1,(O-K)/2,1
0,8
420 FOR C=K+40 TO K+440 STEP 80:IF C=O OR C+19=O THEN E=
O+(PEEK(O+21)=68)
490 NEXT C:POKE 53761,0:POSITION 1+(S)*17,0:? #6;CHR$(98
+A-D+(S)*96):FOR C=1 TO 60:NEXT C:POKE 53763,0
495 GOTO 300+(A=D OR O=K+59)*200
500 V(S+1)=V(S+1)-(A-D):IF Z=J THEN Z=(S=0)*U:IF S THEN
Z=T
502 IF PEEK(O+20)=194 THEN V(S+1)=V(S+1)+(V(S+1)<37)*3
503 E=0:A=0:IF PEEK(O-40)=68 THEN GOTO 550
505 IF O=K+59 THEN POKE O,26+R:GOTO 800
510 FOR C=1 TO 99:NEXT C:IF S=0 THEN X=O:F=Q:T=Z:GOTO 99
520 IF S THEN M=O:G=Q:U=Z:GOTO 98
550 A=A+1:IF A=200 THEN 510
560 GOTO 550+(STICK(S)=14)*50
580 IF S=0 THEN X=O:M=W:F=Q:T=Z:GOTO 98
590 IF S THEN M=O:X=W:G=Q:U=Z:GOTO 99
600 B=8:POKE O-20,B+R:POKE O,0:SOUND 0,150,12,8:FOR C=1
TO 30:NEXT C:POKE O-20,0:POKE O,B+R:POKE O-40,197
610 FOR C=1 TO 20:NEXT C:POKE 53761,0:POKE O,Q+R:GOTO 58
0+(PEEK(W+20)=194)*120
700 FOR C=1 TO 99:NEXT C:J=8+(S=0)*192:A=W+20
705 FOR A=A TO A+(PEEK(A+80)=194)*80 STEP 80:NEXT A
706 GOTO 705+(PEEK(A)=65)*2
707 A=A-80:FOR A=A TO W-20 STEP -80:POKE A,221:FOR C=15
TO 0 STEP -0.5:SOUND 0,90,12,C:NEXT C:NEXT A
710 A=0:FOR W=W TO W+60 STEP 20:A=A+1:POKE W,J:POKE W-20
,(A=3)*221:SOUND 0,(W-K)/2,10,8:FOR C=1 TO 9:NEXT C:NEXT
 W
715 IF PEEK(W+20)<>65 THEN 710
720 SOUND 0,100,2,6:J=J+2:POKE W,J:POKE W-20,0:FOR C=1 T
O 20:NEXT C:POKE 53761,0:A=W-60:FOR C=1 TO 99:NEXT C
730 FOR A=A TO A-(PEEK(A-80)=221)*80 STEP -80:POKE A,194
:FOR C=50 TO 100:SOUND 0,C,6,6:NEXT C
740 NEXT A:GOTO 730+(PEEK(A)<>221)*20
750 POKE 53761,0:GOTO 580
800 A=A+1:IF A=40 THEN 580
801 IF INT(A/2)=A/2 THEN POKE K+39,12:FOR C=15 TO 0 STEP
 -1:SOUND 0,15,10,C:NEXT C
803 POKE K+39,140:GOTO (STRIG(S)=0)*5+800
805 FOR A=26 TO 28:POKE K+59,A+R:FOR C=1 TO 20:NEXT C:NE
XT A:POKE K+39,0:E=0:H=18
810 D=INT(RND(0)*2)+1:A=K+38+(D=1)*20:I=(D=1)
820 POKE A,140:D=D+(D=1)+(D=3)-(D=2)-(D=4):IF A=K+36 OR
A=K+37 THEN POKE O,26+R
822 IF A=K+41 OR (PEEK(A+20)=131 OR PEEK(A+40)=131) AND
(RND(0)<0.7 AND I=0) THEN D=5
825 IF A=W THEN 900
826 IF H=0 AND A>W THEN POKE A,0:FOR C=60 TO 20 STEP -1:
SOUND 0,C,10,(C>20)*4:NEXT C:GOTO 580
830 IF D=5 AND PEEK(A+20)=65 THEN D=(INT(RND(0)*2)+1)*2
831 IF D=5 AND PEEK(A-40)=131 AND (PEEK(A+21)=65 OR PEEK
(A+21)=194) AND RND(0)<0.3 THEN D=(INT(RND(0)*2)+1)*2
833 IF H=0 AND I THEN I=0:D=3:POKE A,0:A=A-20:POKE A,140
:GOTO 880
835 IF H=0 OR H=19 THEN D=D+(D<3)*2-(D=3 OR D=4)*2:IF H
AND RND(0)<0.6 THEN POKE A,0:A=A-20:D=1:I=1:POKE A,140
880 FOR C=44 TO 28 STEP -8:SOUND 0,C,12,(D=2 OR D=4)*4:N
EXT C
890 H=H-(D<3)+(D=3 OR D=4):POKE A,E:A=A+(D=1)*19+(D=3)*2
1+(D=5)*20-(D=2)*21-(D=4)*19:E=PEEK(A):GOTO 820
900 FOR B=11 TO 15:POKE A,B+128:SOUND 0,B*10,4,8:FOR C=1
 TO 30:NEXT C:NEXT B:POKE A,0
910 FOR B=8 TO 0 STEP -1:SOUND 0,B*10,4,B:FOR C=1 TO 5:N
EXT C:NEXT B:FOR C=1 TO 500:NEXT C
920 FOR A=150 TO 40 STEP -10:POSITION 1,0:? #6;"  PLAYER
   WINS   ":FOR B=1 TO 10:NEXT B:POSITION 3,0
930 ? #6;"PLAYER ";CHR$(49+S+(S)*96);" WINS":SOUND 0,A,1
0,8:SOUND 1,A-10,12,4:FOR B=1 TO 20:NEXT B:SOUND 0,0,0,0
940 NEXT A:SOUND 1,0,0,0:FOR A=1 TO 3000:NEXT A:RESTORE
1050:? #6;CHR$(125):GOTO 20
1000 DATA 255,255,1,1,0,0,0,0,239,239,239,239,0,0,0,0,12
9,255,129,129,129,255,129,129,116,84,84,92,84,84,112,0
1010 DATA 112,112,116,124,116,116,116,0,24,24,16,56,84,1
6,40,68,24,24,8,28,42,8,20,34,24,90,74,126,24,24,36,36
1020 DATA 108,108,68,238,213,68,170,170,0,0,12,12,4,14,2
1,60,60,60,126,126,126,126,60,60,0,0,56,124,124,124,124
1030 DATA 56,0,0,24,60,60,60,24,0,0,0,0,16,56,56,16,0,0,
0,0,0,16,16,0,0,24,24,8,28,42,42,8,8,4,12,12,4,4,8,16,8
1040 DATA 24,88,40,24,8,8,8,8,195,195,195,195,195,195,19
5,195
1050 DATA 90,90,90,0,68,68,68,0,55,55,0,68,68,68,0,0,0,0
,90,90,90,0,68,68,68,0,55,55,0,68,68,68,0,0,0
1060 DATA 0,73,73,0,68,68,0,73,73,0,85,85,85,0,0,0,0,78,
78,0,73,73,0,78,78,0,90,90,90,0,0,0
1070 DATA 90,90,90,0,68,68,68,0,55,55,0,68,68,68,0,0,0,0
,90,90,90,0,68,68,68,0,55,55,0,68,68,68,0,0,0
1080 DATA 67,67,0,62,62,0,67,67,0,72,72,72,0,0,95,95,0,6
8,68,68,68,68
1090 DATA 0,0,0,0,0,0,0,0,0,0,57,57,0,0,90,90,0,0,0,0,0,
0,0,0,0,0,0,57,57,0,0,90,90,90,0
```

SPECIAL OFFERS

By popular request we have arranged a further selection of top quality software titles for members to purchase from Maplin Electronics. The prices shown are extremely low, and so we have had to rule that total purchases must be at least £7.00 to be eligible for the club's special prices. Address your orders to the club and we will pass them on to Maplin for you.

Educational Software

Three software packages that are all intended specifically for the 4-10 age group. Each comprises three or four games which, although simple, are nonetheless excellently produced. The instructions are written in the form of a comic based around the robot/alien character Prototype, and each package should keep many a youngster happy for hours. All three work on the XL range as well as the old 800 & 400.

PROTO'S FAVOURITE GAMES
16K Cassette or 32 K Disk for 400, 800 and XL range.
Normally £9.95 **Club Price £3.00**

PROTO'S FUN DAY
16K Cassette or 32K Disk for 400, 800 and XL range.
Normally £9.95 **Club Price £3.00**

THE ADVENTURES OF PROTO
16K Cassette or 32K Disk for 400, 800 and XL range.
Normally £9.95 **Club Price £3.00**

MUSIC MAJOR by Charles Parker is, as the instructions state, a lighthearted approach to learning the basics of music for the young student. Produced in a similar but 'slightly' more mature style to the prototype series, but featuring Professor Von Chip. Excellent entertainment for those also wishing to learn something about music and includes a quiz program about Beethoven that can be modified to suit other topics.

MUSIC MAJOR
32K Cassette or Disk for 400, 800 and XL range.
Normally £14.95 **Club Price £4.00**

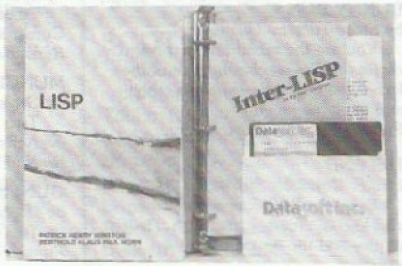
Games

BUG ATTACK
This high speed colourful graphics game is an excellent version of the popular CENTIPEDE game, and it is guaranteed to wear you to exhaustion! 32K Cassette or 48K Disk for 400, 800 and XL range.
Normally £13.95 **Club Price £3.00**

CYPHER BOWL
Very clever graphics allow the participants in this game to move their teams in changing formations against their opponents. Excellent sound effects and crowd-roar give this game a marvellous 'feel'. An added bonus is that both cassette and disk are supplied together in each package, together with comprehensive instructions and strategy plan charts.
16K Cassette or Disk for 400, 800 and XL range.
Normally £12.95 **Club Price £4.00**

JAWBREAKER
A brilliant verson of PAC-MAN, but with much higher resolution, smoother graphics and better sound effects. An amusing aspect of this game is the toothbrush which scrubs the jaws after a complete screenful is munched.
32K disk only, for 400, 800 and XL range.
Normally £9.95 **Club Price £3.00**

Languages



INTER-LISP
LISP is a computer language which is used extensively in artificial intelligence research. This is an ideal purchase for the Atari owner who wants to experiment with a language which offers a completely different directon in programming, it is certainly not suitable for the beginner! A manual is provided, together with a substantial tutorial book.
48K Disk only for 400, 800 and XL range.
Normally £59.95 **Club Price £29.95**

Cheques or Postal Orders are to be made payable to the club and the prices quoted are valid until the end of February 1985. Stocks are limited so please order promptly and remember to state cassette or disk format.

USER GROUP SOFTWARE

Software Librarian - Roy Smith

Due to demand from members there are now two ways to get programs from the library. The original scheme of exchanging '3 for 1' will still apply, but now with an added bonus. So the library rules have been extended to enable those members who cannot write their own programs to gain access, and those that can to have a possibility of some reward for their efforts. The extended library rules are as follows:

3 FOR 1 EXCHANGE

1. Every program you donate to the library entitles you to three programs in return.
2. The program you donate must be your original and not copied.
3. Your donated program must be submitted on a cassette or a disk, programs in the form of print-outs will not be processed.
4. If your program requires any special instructions they should be added in the form of REM statements within the program (or you may present them as instructions when the program is actually run).
5. BONUS. Every program donated per quarter (between issues of the newsletter) will be eligible to be judged 'STAR PROGRAM' for that quarter. This carries a prize of £10 which will be paid to the author from the club funds. The programs will be judged by the Editorial Team and their decision will be final. The Editorial Team are not eligible for the prize.
6. The '3 FOR 1' exchange is only open to club members.

DONATION SCHEME

1. Every club member will be entitled to ask for up to 3 programs per quarter from the library by donating to the club funds.
2. If a member does not take his/her entitlement for a particular quarter, it cannot be carried forward to the next quarter.
3. A member can have more than one quarter's entitlement at one time (up to a maximum of 12 programs (1 year)), but then will be unable to ask for more until his/her credit quarters have been used. Note that odd numbers of programs will be counted in quarters, i.e. if a member asks for 5 programs, the first 3 will be that quarter's entitlement, the next 2 will be the second quarter's entitlement and he/she will have to wait until the third quarter before he/she is entitiled to any more. Also note that having programs in advance will only be allowed if that member's membership covers the advance quarters.
4. The donation fee will be £1 per program and is not refundable. Cheques and Postal Orders are to be made out to the 'U.K. Atari Computer Owners Club'.
5. Members must send in a blank cassette or diskette for the chosen programs to be recorded on.
6. The 'DONATION SCHEME' is only open to club members.

Finally I would like to point out that some people omit to include return postage when donating to the library, so please do not forget to include 30p worth of stamps to cover this.

THE LIBRARY SOFTWARE SERVICE IS FOR MEMBERS ONLY

LIBRARY SOFTWARE TITLES

Listed below are the software titles received by members for inclusion in the library since issue six was published. As the library now contains over 200 programs it is getting a bit too large to keep on reprinting the entire list. Eventually it would probably take over the whole magazine and there would be no room left for the articles and program listings. For those of you who are new members and do not know what is available from the library then you can either purchase a back copy of issue six or send for a photocopy of the complete list which is available from the Librarian. There is a small charge for this service to cover photocopying costs. If you would like a list please send 50p and a SAE for return.

Games

BUNNY RUN
by Steve Tullett - Dalkeith.
You are the rabbit and you need carrots to survive, but beware of the ferret, the fox, the evil brown rabbit and the farmer's gun!
Runs in 16K min. Cassette only.

DEFEND
by J.P. Connell - Stoke-on-Trent.
Excellent BASIC version of the well known arcade game, with 5 levels of play and the option of 1 to 4 players.
Runs in 16K Cassette or 32K Disk min.

FLIP 2
by Stephen Taylor - London.
The object of the game is to flip all of the tiles to a new colour, but you can only use a colour so many times and the same colours cannot be adjacent. An exceedingly frustrating game.
Runs in 32K Cassette or Disk min.

JACKPOT
by J.P. Connell - Stoke-on-Trent.
A fruit machine simulation including nudge and hold.
Runs in 16K min. Cassette only.
XL machines only.

MATCHBOX
by Peter Cunningham - Chester.
This is a concentration type game in which you have to match the hidden shapes in the boxes.
Runs in 16K Cassette or 32K Disk min.

MILKY WAY
by Grahame Fairall - Oxon.
Pinball game created using Pinball Constructor.
Runs in 48K min. Disk only.

MUNCH
by Grahame Fairall - Oxon.
Two player game in which you eat everything except the mushrooms.
Runs in 16K Cassette or 32K Disk min.

GUNFIGHT 2
by Grahame Fairall - Oxon.
Two player cowboy gunfight.
Runs in 16K Cassette or Disk min.

TIMEWORD
by Steve & Gwenda Tullett - Dalkeith.
A two player game based on scrabble and TV's Countdown program.
Runs in 16K Cassette or Disk min.

Adventure Games

★★★★ STAR PROGRAM ★★★★

★ IDOL ISLAND ★
by J.P. Crackett - Choppington.
Extensive word only adventure incorporating its own text style. Your object is to search the island for treasure and live to tell the tale.
Runs in 48K Disk only.

SIMPLE ADVENTURE
by Alex Kells - Liverpool.
An adventure with graphics, the object is to find the hidden gem. This program has an Autorun file and requires one complete side of a disk.
Runs in 32K min. Disk only.

Home Entertainment

BIORHYTHM 2
by J. Bennett - Newcastle.
This program gives a graphic representation of your future well-being.
Runs in 16K Cassette or 32K Disk min.

CHANGE GIVER
by G. Berry - Wakefield.
A program that works out what coins you should expect in your change.
Runs in 16K Cassette or Disk min.

CHESS SET
by Steve Tullett - Dalkeith.
A chess set drawn in GR.8 which can be used to play the game of chess.
Runs in 16K Cassette or 32K Disk min.

TEXT GOLF
by Steve Tullett - Dalkeith.
Make your way round the course by selecting the right club and the right shot.
Runs in 16K Cassette or Disk min.

Demos

COLOR SELECTOR
by Ian Leonard - Chelmsford.
Use this program to find the colours you require.
Runs in 16K Cassette or Disk min.

FIREWORK
by Mark Christian - Wirral.
Randomly generated fireworks display with sound effects.
Runs in 16K Cassette or Disk min.

SHAPES 8
by Mark and Brian Christian - Wirral.
Interesting little demo that has some pretty nice shapes.
Runs in 16K Cassette or Disk min.

Utilities

ARTIST 2
by Martin Byfield - Birmingham.
Artistic sketch pad program in graphics 7+.
Runs in 32K Cassette or Disk min.

ASSEMBLER 2
by M.C. Barnard - Guisborough.
Improved version of Chris Rutter's assembler with disk and print out options.
Runs in 32K Disk min. only.

BUBBLESORT
by M. Iremonger - Dublin.
Sorts a list and then dumps it to a printer.
Runs in 16K Cassette or Disk min.

CHARACTER GENERATOR 5
by Stephen Bylo - Harpenden.
Design characters and save them to cassette.
Runs in 48K min. Cassette only.

DISK CONSISTENCY CHECKER
by Paul McAlinden - Airdrie.
Verifies that the directory, bit map, etc. on a DOS 3 disk are intact and in the event of corruption may help to

identify the problem.
Runs in 16K min. Disk only.

ERROR
by Stephen Taylor – London.
This program gives written error messages in place of the normal error number.
Runs in 16K Cassette or Disk min.

ERROR 2
by David Pink – Swanley.
This program supplies error messages instead of code numbers.
Runs in 16K Cassette or Disk min.

GRAPHY
by M. Iremonger – Dublin.
This program will graph sets of numbers using auto-scaling, includes a print option.
Runs in 32K Cassette or Disk min.

LISTALL
by J.P. Crackett – Choppington.
This m/c program will list any ATASCII file to an Epson printer (tested on RX80 F/T) including inverse and graphics characters. Source code is also available for Atari Macro Assembler.
Runs in 32K min. Disk only.

MEMORY DUMP
by Jeff Davies – Llandeilo.
A memory dump routine to the screen.
Runs in 16K Cassette or Disk min.

MENU
by M. Iremonger – Dublin.
Lists programs on disk, also allows them to be RUN and files can be locked, unlocked, deleted, etc.
Runs in 16K min. Disk only.

TOP TEN

1 (4)	FOLLY OF E. KKHANN	ALEX KELLS
2 (2)	SYNTHESISER	CHRIS PAYNE
3 (3)	STONEVILLE MANOR	NIGEL HASLOCK
4 (1)	THE VALLEY	STEVE CALKIN
5 (8)	OUTPOST	ANTHONY BALL
6 (5)	ASSEMBLER	CHRIS RUTTER
7 (-)	BIORYTHM	EZIO BOTTARELLI
8 (-)	CATALOG	H.M. HOFFMAN
9 (-)	FRUIT MACHINE	MIKE NASH
10 (-)	USERCOMP	TREVOR SKEGGS

MC MONITOR
by Paul McAlinden – Airdrie.
A simple m/c monitor which permits display, modification, saving and loading of blocks of memory using hex addresses and data.
Runs in 32K min. Disk only.

MENU MAKER
by Mark & Brian Christian – Wirral.
Add this program to your disks to give you a menu of files on that disk.
Runs in 16K min. Disk only.

MINIDOS 2
by Linda Tinkler – Wirral.
This program gives Directory, Rename, Delete, Lock, Unlock and Disk format options directly from BASIC.
Runs in 16K min. Disk only.

OLDTEXT
by Martin Byfield – Birmingham.
Change Atari text into old fashioned lettering.
Runs in 32K Cassette or Disk min.

REMOVE
by Jeff Davies – Llandeilo
Deletes lines using forced read mode.
Runs in 16K Cassette or Disk min.

TEXT FILE MANAGER
by Ian Scott – Tyne & Wear.
Disk based BASIC program to create and edit text files.
Runs in 16K min. Disk only.

VATCALC
by Ian Leonard – Chelmsford.
Useful little program for the businessman which will add or subtract VAT from figures input by the user.
Runs in 16K Cassette or Disk min.

WORLD CLOCK
by David Pink – Swanley.
On input of the present time this program will give a list of the times in many places around the world.
Runs in 16K Cassette or Disk min.

YEARLY BARGRAPH
by M. Iremonger – Dublin.
Graphs monthly income up to 2 years, includes print option.
Runs in 16K Cassette or 32K Disk min.

Education

ELECTRONIC DICTIONARY
by Paolo Fragapane – Bristol.
Create your own foreign language dictionary, this program uses the forced read mode to create its own data. Originally devised for French/English it could be modified to suit other languages.
Runs in 16K min. Cassette only.

MATHS PROBLEMS
by Brian Christian – Wirral.
Three programs to solve Simultaneous Equations, Indices and Triangles. Very useful for school homework.
Runs in 32K Cassette or Disk min.

Music

MUSIC 3
by Grahame Fairall – Oxon.
Six tunes: Isle of Capri, Hawaii 5-0, Match of the Day, A Whiter Shade of Pale, Lion Sleeps Tonight, Steptoe and Son. All for use with the Atari Music Composer cartridge.
Runs in 16K Cassette or Disk min.

MUSIC 4
by Grahame Fairall – Oxon.
Six tunes: Cinderella Rockafella, Diamonds are a Girls Best Friend, Tie a Yellow Ribbon, Star Wars, Milord, Organ Waltz. All for use with the Atari Music Composer cartridge.
Runs in 16K Cassette or Disk min.

THREE TUNES
by Grahame Fairall – Oxon.
The Entertainer, Run Rabbit and Camptown Races all from BASIC.
Runs in 16K Cassette or 32K Disk min.

CONTACT

ESSEX

I am interested in starting a local user's group, would anybody in the Chelmsford/ Braintree area who wants to know more contact me. Ian Leonard, telephone Chelmsford 440512.

Is there anyone in the Club who might be interested in practical applications of the Atari "Graph It" programs? Is it only for drawing designs or has it practical, business uses? I would be pleased to make contact with someone who has a similar interest. Chris Cheatle, 28 Park Rd, Leigh-on Sea, Essex or phone: Southend-on-Sea 79043

YORKSHIRE

Do you know the procedure for altering the GR.1 and 2 Characterset in the "Magic Window" utility from Quicksilva? Contact John Proud, Police Station Flat, Grove Square, Leyburn, North Yorkshire.

DERBYSHIRE

Anyone out there got any info on dumping a screen, generated by a video camera, onto a printer. It can be done, I'd love to know how (A similar system is available for the Beeb!?). Bryan Cox, 20, Somerset Close, Buxton, Derbyshire SK17 9XB.

YORKSHIRE

Is there anyone in my local area who would like to share ideas and interests with an Atari enthusiast who's friends are not "into computing". Preferably you should have a disk drive and an interest in Assembly Language. P.S. Do you know of any local user groups I could contact? Steve Hill, 5 Broadacres, Durkar, Wakefield, West Yorks, WF4 3BE.

NORTHERN IRELAND

New Atari user group to be set up in Belfast, plans to produce a newsletter and a book/software library. If you would like to help or are just plain interested, please write (including SAE) to Frankie Smyth, 62 Orchardville Ave., Belfast, BT10 0JH.

I would like to contact owners of NEC 8023 printers who can help me to get the best out of my printer, especially the graphics handling capabilities. G.M. Hutchinson, 1 Hollymount, Finaghy, Belfast BT10 0GL.

AVON

New Bulletin Board System (BBS) for Atari owners has started up. On line from 21.00 to 7.00, ring Bath (0225) 23276.

NOTTS/DERBYS

Does anyone have a program that allows creation of files with an editing facility and then be able to print the files out on a 1020. I would also like to contact any other ATARI enthusiasts in my local area. Tony Morris, 2 Potter Street, Sutton in Ashfield, Notts.

BEDFORDSHIRE

The Atari does not seem to allow mixing of coloured text in normal (GR.0) size. Many other machines now do this, and although the Technical Notes suggest that the hardware can cope, no-one seems to have written a routine to cover it. If someone somewhere has done it or knows how to do it, please contact me. Ian Favell, 34 Ruffs Furze, Oakley, Beds.

SCOTLAND

Does anyone know if there is a program which would allow Touch Tablet disk files to be loaded into a BASIC program? Do you know what sort of format is used to save the pictures? Any help would be gratefully received. A.S. Darling, 2 Salters Terrace, Dalkeith, Midlothian.

INTERRUPTS Part 1.

By Steven Hillen

This article is intended as a general discussion on how to use interrupts to improve the graphics of your BASIC programs. It does not pretend to fully explain interrupts, it merely provides guidelines for using them. By their very nature, interrupts can only be accessed by machine-code, but don't worry, if you have not been keeping up with "Cracking the Code", there are listings for you to add to your BASIC programs which require no knowledge of machine-code!

Before we start, I would recommend that you make sure you've read Keith Mayhew and Roy Smith's excellent article on display lists (Issue 4) as it contains much material that I do not wish to repeat here.

There are two questions which must be answered. Firstly, what are interrupts? Imagine a bank clerk busy totting up figures one afternoon. Suddenly, the telephone rings, so he makes a note of where his calculations are, then answers the 'phone. After his wife has finished speaking, he returns to his sums at the exact point where he left off. The bank clerk is analogous to the 6502 chip in your Atari, the telephone call is an interrupt which halts processing for a while. The two types of interrupt that are useful for graphics are known as the Vertical Blank Interrupt (VBI), and the Display List Interrupt (DLI).

Secondly, why are interrupts useful to us? The answer to this is that they are synchronized to the TV display. Translated, this means that by using interrupts, we can have many colours on the screen, scroll smoothly without flicker, and move objects on the screen continually without stopping BASIC or even seeing them being redrawn.

Setting Up Interrupts

This section is aimed at the novice assembly programmer, and is not essential to those of you who would rather just use the listed interrupt routines. DLI's occur at the end of one scan-line on your TV set. They allow you to ensure that when you change colours half-way down the screen, the exact position of this change will not alter, i.e. the boundary of the colour change will be fixed. DLI's must be enabled by the following operations:

1. Adjust your display list by adding 128 to each mode line that you wish an interrupt to occur on, remembering that the interrupt will occur at the END of that line.
2. Write your DLI, remembering that you must save any registers you use on the hardware stack and that you must address hardware colour registers, not the shadows at locations 704-712.
3. Point to your DLI routine by poking the low and high bytes of its address into VDSLST (locations 512,513).
4. Enable DLI's by poking NMIEN (54286) with 192. (You can disable DLI's with a poke of 64 to NMIEN.)

Here is the source code for a simple display list interrupt:-

```
WSYNC   .EQ  $D40A      This is the latch that synchronizes
                        your program with the TV display.
COLPH2  .EQ  $D018      This is the hardware equivalent of
                        colour register 710.
DLI     PHA             Save Accumulator on stack.
        LDA  #$34       Load A with red colour.
        STA  WSYNC      Just addressing WSYNC causes
                        processing to halt until synchronized.
        STA  COLPH2     Save red into hardware.
        PLA             Restore the accumulator.
        RTI             Return to main processing.
```

As you can see, DLI's are very simple. However, they are best kept fairly short, you only really have enough time to change a few colours. Note also that the red colour is stored into hardware immediately AFTER the WSYNC instruction. If you stored it before the WSYNC then red would appear somewhere along the current TV line rather than at the end; and worse, on successive TV frames, red could appear anywhere along that line, causing flicker. It is also a good idea to make use of page 0 locations for variables used in DLI's, as these locations take less time to access.

While DLI's usually consist of only a few instructions, VBI's can be very large. Usually, however, it is again best to keep VBI's reasonably short as excessively long interrupts can cause bad flicker on the screen. As you probably know, VBI occurs 50 times per second in between consecutive TV frames. It occurs when the electron beam in the TV set is returning from bottom right to top left, thus any scrolling or redrawing of the screen implemented during this time will not cause any flicker. The Atari Operating System has its own VBI routine which updates the internal clock, copies shadow colour registers into hardware, copies joystick port data into RAM, and also deals with the keyboard auto-repeat.

Let's examine exactly what happens when the 6502 jumps to a VBI. The A,X,Y registers are saved on the stack. A jump is made through VVBLKI which normally points to the Stage 1 Vblank Processor. The system timers are updated (locations 18-20). If the code interrupted was non-critical, then a jump is made to Stage 2. This now deals with all the colour registers, etc. However, if the interrupted code was critical, the system leaves the interrupt immediately, without passing through either Stage 2 processing or the vector in VVBLKD. Otherwise, the OS now jumps through VVBLKD where the registers are restored and main processing recommences. If all this sounds a little complicated, don't worry, using a VBI is simplicity itself. First you must decide whether you wish your routine to take place before (immediate) or after (deferred) that of the OS. If any time-critical operation is likely to occur whilst your VBI is in operation, then you should place your VBI before the OS routine as EVERY VBI will then jump through your routine. Note that if I/O is being used, then your immediate VBI should be short, or else I/O may be messed up. If you place your routine after the OS one, then only the non-critical VBI's will vector through your routine, which means that your interrupt routine might not always be called. If any I/O is likely to go on, it is better to place your VBI after the OS routine.

Suppose we want to place our VBI routine after the OS routine. We must therefore alter VVBLKD to point to our deferred Vblank processing. To avoid the possibility of having changed only one byte of VVBLKD when an interrupt occurred, we use the following routine to point to our program:-

```
        LDA  #$07       Specifies deferred vector.
        LDX  /VBI       High byte of your program address.
        LDY  #VBI       Low byte of your program address.
        JSR  $E45C      Let the OS install our new vector.
        RTS
VBI     LDA  #0         Our actual program.
        STA  COUNT
        JMP  $E462      This instruction exits us from the
                        VBI.
```

Suppose we had wanted to install our program before the OS routine. We use a similar method to change VVBLKI. In this case the following routine is used:-

```
        LDA  #$06       Specify immediate vector.
        LDX  /VBI       MSB of address of program.
        LDY  #VBI       LSB of address of program.
        JSR  $E45C      Let OS install vector.
        RTS
VBI     LDA  #0         Our program.
        STA  COUNT
        JMP  $E45F      Into Stage 1 Vblank.
```

Note that the JMP instruction which is executed at the end of your VBI program is different for the immediate and deferred types. Note also that in calling the routine that sets up our VBI, it automatically disables any DLI's you may have enabled. These

DLI's can be re-enabled with a poke of 192 to NMIEN.

That is all you need to know to set up and write your own interrupt procedures. I hope that you will find the following examples useful, both as illustrations of DLI's and VBI's and as complete routines that anyone can 'tack' on to their BASIC programs.

Display List Interrupts

There are two listings for each DLI, but you only need to type in the BASIC listing for each. The source code for the DLI routines are included for those who wish to see further examples before writing their own.

Listing 1 is a program that will allow you to change one colour down the screen as many times as you like. It will work in any graphics mode, and with colours that you specify. Two demos are included in the listing starting at lines 100 and 200 respectively. The actual DLI handler is installed by lines 10 to 40. Don't forget to save this listing before running the program! So the procedure to use this DLI in your own program is as follows:-

1. Install machine code on page 6.
2. Locate your display list and adjust the interrupt lines by adding 128.
   Be careful not to alter LMS operand bytes.
3. Call the initialising routine:
   X=USR(1536,REGISTER,NUMBER, COL1,COL2....) Where REGISTER is the colour you want to change, e.g. for Color 0 (708) REGISTER=0. For Color 4 (712) REGISTER=4. NUMBER is the total number of interrupts that you have installed into the modified display list, and COL1,COL2... are the colours that you want to display in the order that they will appear down the screen. Note that it is essential that the number of COL1,COL2, etc. that you pass to this routine MUST equal NUMBER and the total number of DLI's that you have installed. If they are not equal, you will probably cause a lock-up.
4. Enable DLI's with a POKE to NMIEN (54286) with a number larger than 192.

NOTE: In this program, anything which is underlined, should be entered in "INVERSE".

Listing 1.

```
5 REM xxxMachine code followsxxx
10 DATA 104,104,104,216,24,105,22,141,67,6,104,104,170,1
60,0,104,104,153,77,6,200,202,208,247,142,76,6,169
20 DATA 6,141,1,2,169,54,141,0,2,169,6,162,6,160,46,76,9
2,228,169,0,141,76,6,76,95,228,72,152,72,172,76,6
30 DATA 185,77,6,141,10,212,141,22,208,238,76,6,104,168,
104,64
35 REM xxxInstall m-c in page 6xxx
40 FOR A=0 TO 75:READ D:POKE 1536+A,D:NEXT A
50 ? "DEMO 1 OR 2":INPUT A:IF A=1 THEN GOTO 100
60 IF A=2 THEN GOTO 200
70 GOTO 50
80 REM xxxSunset display in Gr.8xxx
100 GRAPHICS 24:POKE 710,132
110 REM xxxFind Display Listxxx
120 A=PEEK(560)+256*PEEK(561)+5:RESTORE 140
125 REM xxxAlter Display Listxxx
130 FOR B=1 TO 13:READ LINE:POKE A+LINE,PEEK(A+LINE)+128
:NEXT B
140 DATA 50,70,75,80,110,112,114,117,120,122,126,130,140
145 REM xxxX=USR(1536,color register (0-4),no.of interru
pts,color data....)xxx
150 X=USR(1536,2,13,148,133,117,133,117,101,85,69,53,0,1
94,196,198)
160 POKE 54286,192:REM xxxEnablexxx
170 GOTO 170
200 REM xxxColours in Gr.0xxx
210 A=PEEK(560)+256*PEEK(561)+6:POKE 710,16
220 POKE A-3,PEEK(A-3)+128
230 FOR B=A TO A+22:POKE B,130:NEXT B
240 X=USR(1536,2,24,18,20,22,24,32,34,36,38,40,48,50,52,
54,56,64,66,68,70,72,80,82,84,86,88)
250 POKE 54286,192
260 LIST
```

Listing 2.

```
00010 ;An example DLI
00020 ;by S.Hillen.
00030 ;
00040         .LI OFF
00050 ;
00060 ;Equates for the DLI
00070 ;
00080 ;
00090 VDSLST  .EQ $200    DLI vector
00100 COLPH0  .EQ $D016   Colour 0
00110 WSYNC   .EQ $D40A   Wait for
00120 ;                   sync
00130 SETVBV  .EQ $E45C   Set Vblank
00140 SYSVBV  .EQ $E45F   Stage 1
00150 ;
00160         .OR $600    Page 6
00170 ;
00180         PLA         Discard
00190 ;                   #arguments
00200         PLA         Discard hi
00210         PLA         Colour
00220         CLD         No decimal
00230         CLC         Adjust DLI
00240         ADC #$16    to correct
00250         STA COLBYTE+1  colour.
00260 ;
00270         PLA         Discard hi
00280         PLA         No.of cols
00290         TAX         as index
00300         LDY #0      Init Y
00310 GET.DATA PLA        Discard hi
00320         PLA         Store in
00330         STA COLOR.DATA,Y table
00340         INY         and get
00350         DEX         another
00360         BNE GET.DATA byte
00370 ;
00380         STX COUNT   Zero it
00390         LDA /DLI    Point to
00400         STA VDSLST+1  our
00410         LDA #DLI    DLI and
00420         STA VDSLST  then the
00430         LDA #6      immediate
00440         LDX /VBI    Vblank
00450         LDY #VBI    routine
00460         JMP SETVBV  Into O.S.
00470 ;
00480 ;The vertical blank just resets
00490 ;the variable COUNT to ensure
00500 ;that the same colour starts
00510 ;at the top of the display.
00520 ;
```

```
00530 VBI       LDA #0          Reset it
00540           STA COUNT       & jump to
00550           JMP SYSVBV      OS routine
00560 ;
00570 ;The DLI save A and Y on the
00580 ;stack, finds out which line
00590 ;it is on from COUNT,loads the
00600 ;related colour for that line,
00610 ;and stores it in the hardware
00620 ;colour register selected by
00630 ;the user-it then restores A
00640 ;and Y and returns.
00650 ;
00660 DLI
00670           PHA             Save A
00680           TYA
00690           PHA             Save Y
00700           LDY COUNT       Get line,
00710           LDA COLOR,DATA,Y color
00720           STA WSYNC       Wait and
00730 COLBYTE   STA COLPH0      save color
00740           INC COUNT       Next time
00750           PLA             Restore Y
00760           TAY
00770           PLA             Restore A
00780           RTI             & return
00790 ;
00800 COUNT     .BS 1
00810 COLOR,DATA    .BS 1
```

Listing 3 is another DLI that you can use from BASIC. This one allows you to rotate colours through a row of characters, which could be useful for intro pages on your games etc. Lines 10 to 50 install the DLI routine into page 6, and lines 100 to 170 are the demo-program. The procedure for using this DLI in your own program is as follows:-

1. Install machine code as per lines 10 to 50.
2. Modify your display list so that the line PRECEDING the line you want the colours on, is the interrupt line (add 128).
3. Call the routine with X=USR (1664,REGISTER,HEIGHT) where REGISTER is the colour you wish to be rotated, e.g. capitals in Gr.2 are Color 0 so REGISTER=0, and HEIGHT is the height of the characters that you want to colour rotate, e.g Gr.0 and Gr.1 are 8 high so HEIGHT=8, Gr.2 is 16 high so HEIGHT=16.
4. Enable the DLI with a poke of 192 to NMIEN.

There is no limit to the number of lines that you can have colour rotating provided that you remember to have the interrupt on the preceding line. The only other thing you must not do is to try rotating colours through characters of differing heights on the same screen, i.e. don't expect this DLI to successfully rotate colours through Gr. 2 and GR.1 characters on the same screen, although Gr.0 and Gr.1 characters would be all right as they are the same height.

Hopefully, these examples will have shown you how easy it is to write DLI's. Other possible DLI's could be written to alter the rate of scrolling down the screen, the best example of which is "Frogger", change screen width down the screen by addressing DMACTL, multi-task players and missiles, and to turn on and off GTIA so you can mix Gr.9, 10 or 11 with the normal modes.

Listing 3.

```
5 REM xxxMachine-code data followsxxx
10 DATA 104,104,104,216,141,222,6,24,105,22,141,201,6,14
1,216,6,173,222,6,24,105,196,141,210,6,104,104,141
20 DATA 193,6,169,6,141,1,2,169,189,141,0,2,169,6,162,6,
160,177,76,92,228,238,223,6,173,223,6,141
30 DATA 222,6,76,95,228,72,138,72,162,8,173,222,6,141,10
,212,141,22,208,238,222,6,202,208,241,173,196,2
40 DATA 141,10,212,141,22,208,104,170,104,64
45 REM xxxInstall m-c in page 6xxx
50 FOR A=1664 TO 1664+93:READ D:POKE A,D:NEXT A
60 ? "DEMO 1 OR 2":INPUT B:IF B<1 OR B>2 THEN 60
100 REM xxxFancy font 1xxx
110 GRAPHICS 0:A=PEEK(560)+256*PEEK(561)
115 REM xxxNote that interrupt is on the preceding line!
xxx
120 POKE A+7,130:POKE A+8,5+B
130 POKE 87,1:POSITION 2,6
140 ? #6;"ROTATING COLOURS"
145 REM xxxX=USR(1664,Colour,8 or 16 depending on charac
ter height)xxx
150 X=USR(1664,0,8*B)
160 POKE 54286,192
170 GOTO 170
```

Listing 4.

```
00010 ;Second DLI
00020 ;by S.Hillen
00030 ;
00040 VDSLST    .EQ $200    DLI vector
00050 COLPH0    .EQ $D016
00060 WSYNC     .EQ $D40A
00070 SETVBV    .EQ $E45C
00080 SYSVBV    .EQ $E45F   Stage 1
00090 COLOR0    .EQ 708     Shadow
00100 ;
00110           .OR $680
00120 ;
00130           PLA         Discard
00140           PLA         Discard hi
00150           PLA         colour reg
00151           CLD         No decimal
00160           STA ACTCOL  Need later
00170           CLC
00180           ADC #$16    Adjust DLI
00190           STA ADJUST1+1  so that
00200           STA ADJUST3+1  correct
00210           LDA ACTCOL  colours.
00220           CLC         Also need
00230           ADC #$C4    shadow
00240           STA ADJUST2+1  colour
00250 ;
00260           PLA         Discard hi
00270           PLA         Get height
00280           STA ADJUST+1 and save
00290 ;
00300           LDA /DLI    Vector the
00310           STA VDSLST+1   DLI
00320           LDA #DLI
00330           STA VDSLST
00340           LDA #6      Immediate
00350           LDX /VBI    VBI
00360           LDY #VBI    Let OS do
00370           JMP SETVBV  the rest!
00371 ;
00372 ;This VBI adds one to a control
00373 ;colour shadow and copies it
00374 ;into the DLI colour shadow-
```

```
00375 ;gives rotating effect.
00376 ;
00380 VBI
00390         INC CTLCOL  Rotate the
00400         LDA CTLCOL  colour
00410         STA ACTCOL  by one and
00420         JMP SYSVBV  go to OS!
00430 ;
00431 ;This DLI is unusual in that it
00432 ;addresses WSYNC up to 17 times
00433 ;for one interrupt; a lot of
00434 ;processing time is wasted, but
00435 ;you cannot create a DLI in
00436 ;the middle of a character
00437 ;without more complex control-
00438 ;e.g. using VCOUNT.
00439 ;
00440 DLI       PHA         Save A
00450         TXA
00460         PHA         Save X
00470 ADJUST  LDX #8      8 or 16
00480 AGAIN   LDA ACTCOL  Colour
00490         STA WSYNC   Wait
00500 ADJUST1 STA COLPH0  hardware
00510         INC ACTCOL  Next color
00520         DEX         Done all?
00530         BNE AGAIN   No
00540 ADJUST2 LDA COLOR0  Restore
00550         STA WSYNC   old colour
00560 ADJUST3 STA COLPH0  from shad.
00570         PLA         Restore X
00580         TAX
00590         PLA         Restore A
00600         RTI         & return
00610 ;
00620 ACTCOL  .BS 1
00630 CTLCOL  .BS 1
```

Vertical Blank Interrupts

As you have probably already noticed, all but the very simplest of DLI's need to have counters reset after the last DLI has taken place. The VBI, occuring at the end of the TV frame is an excellent time to reset such variables.

Another good use for a VBI is to move players around on the screen. In this way, fast but smooth movement can be achieved, without holding up your BASIC program. However, before we look at the listed routine to do just this, a quick summary of Player-Missile Graphics (PMG) is in order.

Normally, when you want to move, say a spaceship over a graphics screen, you would have to copy out a region of the screen, copy in your spaceship shape, and then restore the screen in the space where the spaceship had moved from. This process is made especially tedious by the fact that your spaceship shape is non-sequential in memory; i.e. consecutive sections could be spaced apart by 40 bytes due to the screen RAM arrangement.

By using players and missiles, we have a method of moving shapes over the graphics screen without the bother of redrawing the screen behind them. Another advantage is that players map vertically down the screen, i.e. consecutive bytes in your PM table form a one byte wide vertical band down the screen, making movement very simple to produce. Two other very useful features of PMG are that they can be assigned priority, ie. different players can appear to move in front of, or behind other objects. This gives a good pseudo-3D effect, e.g. in Basketball.

The second advantage is that collisions between each player, missile, and objects on the screen are automatically detected by the hardware, and your programs can thus detect collisions just by peeking these locations.

The Atari has 4 players and 4 missiles, which can be grouped together to form a 5th player. There are several methods for putting shapes into the PM table, but only one is practical for use with BASIC. This is known as DMA (Direct Memory Access). This method requires you to reserve some RAM which is your player-missile table. The size of this table depends on whether you want single line resolution players, meaning that there are 256 possible vertical positions, or double line resolution, which means that there are only 128 plottable vertical positions. Rather than write yet another description of setting up PMG, I am limiting this discussion to the use of listing 5 for single line resolution players. Full instructions on how to set this routine are listed below, after the introduction.

Listing 5 is a program which runs in VBI and hence does not interfere with your BASIC program. It works with single line resolution players, but no missiles. By using this routine, players can be moved anywhere on the screen by a single poke, or alternatively, you can select one or two players to be controlled by joystick ports 1 and 2. No other work need be done by BASIC except to poke in desired X and Y coordinates (X and Y as in the normal graphics screens). The procedure for using single line players and this VBI program is as follows:

1. Reserve 2K of memory, starting on a 2K boundary. First make the graphics call and then find a suitable area of memory thus: AREA=INT ((PEEK(106)-16)/8)*8 then reserve it: POKE 106,AREA
2. Tell the system where your PM table is with a poke to PMBASE thus: POKE 54279,AREA
3. Enable P/M DMA with pokes to SDMCTL and GRACTL: POKE 559,62 and POKE 53277,3
4. Set the desired colours by addressing the locations 704-707 for players 0 to 3, and 711 for the fifth player if enabled.
5. Create your player shapes in a similar fashion as you would redefine a character, see diagram 1, and install this data into strings PM0$, PM1$, PM2$ and PM3$ as shown in listing 5.
6. Put the VBI data into VBI$, and call it once with X=USR(ADR (VBI$), JOYSTICKS,SPEED,NUMBER,ADR (PM0$),LEN(PM0$),ADR(PM1$),LEN(PM1$),ADR(PM2$), LEN(PM2$),ADR(PM3$),LEN(PM3$),ADR(VBI$)+56) where JOYSTICKS is 0 if you want no players to be controlled, 1 if joystick 1 is to control player 0, and 2 if joysticks 1 and 2 are to control players 0 and 1 respectively. SPEED is the amount you want the players, if they are being controlled by joysticks, to be moved, e.g. 1 is slow, 3 is quite fast. You should put 0 in as SPEED if no joysticks are being used. NUMBER is the number of players you have defined. Note that if this number is not 4, then you need only include the ADR and LEN of those players you have defined, and this routine assumes that however many players you select, the first is always player 0, the second player 1, etc.

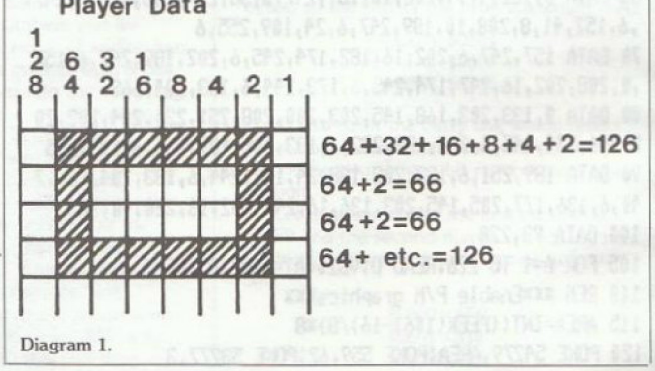


Diagram 1.

7. Enable the VBI with a POKE of less than 128 to 1767; any number larger than 128 will disable the VBI. If you want to change any parameters while the VBI is running, just re-call the USR function with the new parameters.

Now your program is free to alter the four X and Y positions:

Xpos player 0 = 1783
Xpos player 1 = 1784
Xpos player 2 = 1785
Xpos player 3 = 1786
Ypos player 0 = 1787
Ypos player 1 = 1788
Ypos player 2 = 1789
Ypos player 3 = 1790

Note that if you select the joystick option, players 0 and/or 1 will be moved automatically. Remember that only values of Xpos between 41 and 200 will be on the screen, and only values of 32 to 223 will be on screen for Ypos. Remember also, that Ypos refers to the top of your player, see diagram 2.

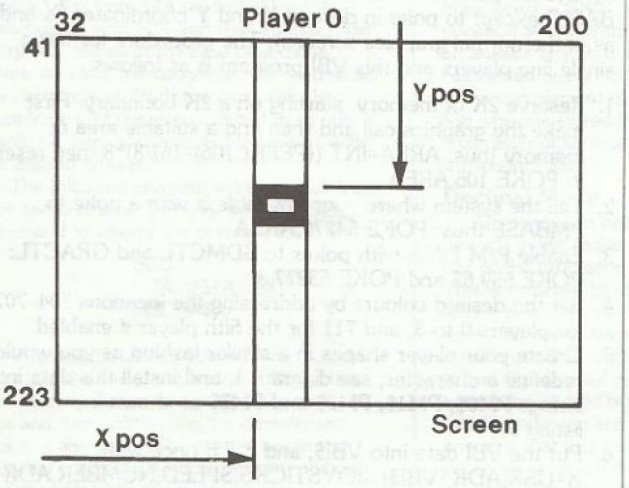


Diagram 2.

Listing 5.

```
10 DIM VBI$(226),PM0$(4),PM1$(6)
15 REM xxx Machine code datxxx
20 DATA 216,104,104,104,141,246,6,104,104,141,255,6,104,
104,141,245,6,170,202,160,0,104,153,232,6,104,153,236
30 DATA 6,104,104,153,240,6,200,202,16,239,104,104,24,10
5,4,141,244,6,104,170,104,168,169,7,32,92,228,96
40 DATA 173,231,6,16,3,76,98,228,8,216,174,246,6,240,75,
202,173,0,211,224,0,240,4,74,74,74,74,168
50 DATA 41,1,208,10,189,251,6,56,237,255,6,157,251,6,152
,41,2,208,10,189,251,6,24,109,255,6,157,251
60 DATA 6,152,41,4,208,10,189,247,6,56,237,255,6,157,247
,6,152,41,8,208,10,189,247,6,24,109,255,6
70 DATA 157,247,6,202,16,182,174,245,6,202,189,247,6,157
,0,208,202,16,247,174,245,6,173,244,6,133,204,169
80 DATA 0,133,203,168,145,203,200,208,251,230,204,202,20
8,246,174,245,6,202,189,232,6,133,206,189,236,6,133,205
90 DATA 189,251,6,133,203,138,24,109,244,6,133,204,188,2
40,6,136,177,205,145,203,136,16,249,202,16,220,40,76
100 DATA 98,228
105 FOR A=1 TO 226:READ D:VBI$(A)=CHR$(D):NEXT A
110 REM xxxEnable P/M graphicsxxx
115 AREA=INT((PEEK(106)-16)/8)x8
120 POKE 54279,AREA:POKE 559,62:POKE 53277,3
125 REM xxxSet up player dataxxx
130 FOR A=1 TO 4:READ D:PM0$(A)=CHR$(D):NEXT A
135 FOR A=1 TO 6:READ D:PM1$(A)=CHR$(D):NEXT A
140 DATA 255,129,129,255,255,129,129,129,129,255
145 POKE 704,52:POKE 705,70:REM xxxSet up colourxxx
150 X=USR(ADR(VBI$),1,3,2,ADR(PM0$),LEN(PM0$),ADR(PM1$),
LEN(PM1$),AREA,ADR(VBI$)+56)
160 POKE 1767,0:REM xxxEnable VBIxxx
170 FOR A=1 TO 255 STEP 2:POKE 1784,A:POKE 1788,A:NEXT A
:GOTO 170
```

Listing 6.

```
00010 ;Player mover
00020 ;by S. Hillen
00030 ;
00040 ;called with
00050 ;X=USR(ML,Joysticks,Speed,
00060 ;Number of players,addresses
00070 ;and lengths of players' data,
00080 ;Pmarea,ML+56)
00090 ;
00100          .LI OFF
00110 ;Equates
00120 ;
00130 SETVBV   .EQ $E45C   Set VBI
00140 XITVBV   .EQ $E462   Leave VBI
00150 PORTA    .EQ $D300   Joysticks
00160 HPOSP0   .EQ $D000   Xpos PM0
00170 ;
00180          .OR $6E7
00190 ;
00200 ENABLE   .BS 1       Flag
00210 HI       .BS 4       Address of
00220 LO       .BS 4       PM strings
00230 LENGTH   .BS 4       Lengths
00240 PMAREA   .BS 1       Pmbase
00250 NUMBER   .BS 1       Of players
00260 JOYST    .BS 1       No.of joys
00270 XPOS     .BS 4       Your work
00280 YPOS     .BS 4       positions
00290 SPEED    .BS 1       How fast
00300 ;
00310 TO       .EQ $CB     Page 0
00320 FROM     .EQ $CD     usage
00330 ;
00340 ;
00350          .OR $4000   Anywhere
00360 ;
00370          CLD         No decimal
00380          PLA         Discard
00390 ;
00400          PLA         Discard hi
00410          PLA         No.of
00420          STA JOYST   joysticks
00430          PLA         Discard hi
00440          PLA         How fast?
00450          STA SPEED
00460          PLA         Discard hi
00470          PLA         How many
00480          STA NUMBER  players?
00490          TAX
00500          DEX         Index
```

```
00510          LDY #0       Init ptr
00520 GET.ADD  PLA          Save hi
00530          STA HI,Y     and lo
00540          PLA          of string
00550          STA LO,Y     addresses
00560          PLA          and then
00570          PLA          save their
00580          STA LENGTH,Y lengths.
00590          INY
00600          DEX          For all of
00610          BPL GET.ADD  them.
00620 ;
00630          PLA          Discard hi
00640          PLA          Add 4 to
00650          CLC          lo byte to
00660          ADC #4       point to
00670          STA PMAREA   player 0.
00680          PLA          Hi of VBI
00690          TAX          into X
00700          PLA          Lo of VBI
00710          TAY          into Y
00720          LDA #7       Deferred
00730          JSR SETVBV   Vblank
00740 ;
00750          RTS          That's all
00760 ;
00770 VBI
00780          LDA ENABLE   O.k.?
00790          BPL .1       Yes
00800          JMP XITVBV   Leave VBI
00810 .1       CLD          Binary
00820          LDX JOYST    No.of joys
00830          BEQ NOJOY    None!
00840          DEX
00850 GETJOY   LDA PORTA    Joysticks
00860          CPX #0
00870          BEQ .1
00880          LSR          Move l.h.
00890          LSR          nibble to
00900          LSR          rh. nibble
00910          LSR          if player2
00920 .1       TAY          Save it
00930          AND #1       Up?
00940          BNE .2       No
00950          LDA YPOS,X   Move up
00960          SEC          by value
00970          SBC SPEED    in SPEED
00980          STA YPOS,X   and save
00990 .2       TYA          Restore A
01000          AND #2       Down?
01010          BNE .3       No
01020          LDA YPOS,X
01030          CLC
01040          ADC SPEED
01050          STA YPOS,X
01060 .3       TYA
01070          AND #4       Left?
01080          BNE .4       No
01090          LDA XPOS,X
01100          SEC
01110          SBC SPEED
01120          STA XPOS,X
01130 .4       TYA
01140          AND #8       Right?
01150          BNE .5       No
01160          LDA XPOS,X
01170          CLC
01180          ADC SPEED
01190          STA XPOS,X
01200 .5       DEX          For both
01210          BPL GETJOY   joysticks
01220 ;
01230 NOJOY    LDX NUMBER   Copy the
01240          DEX          X-pos's to
01250 HPOS     LDA XPOS,X   hardware.
01260          STA HPOSP0,X
01270          DEX          More left?
01280          BPL HPOS     Yes
01290 ;
01300          LDX NUMBER   Now set up
01310          LDA PMAREA   page 0 ptr
01320          STA TO+1     so that 4
01330          LDA #0       pages from
01340          STA TO       player0 to
01350          TAY          player3
01360 ERASE    STA (TO),Y   are erased
01370          INY          Clear
01380          BNE ERASE    PM table.
01390          INC TO+1
01400          DEX
01410          BNE ERASE
01420 ;
01430          LDX NUMBER   Now set up
01440          DEX          pointer to
01450 AGAIN    LDA HI,X     the PM
01460          STA FROM+1   string and
01470          LDA LO,X
01480          STA FROM
01490          LDA YPOS,X   then set
01500          STA TO       another
01510          TXA          pointer to
01520          CLC          the PM
01530          ADC PMAREA   table.
01540          STA TO+1
01550          LDY LENGTH,X
01560          DEY
01570 FILL     LDA (FROM),Y Transfer
01580          STA (TO),Y   string
01590          DEY          into PM
01600          BPL FILL     table.
01610          DEX
01620          BPL AGAIN    Do all
01630          JMP XITVBV   Leave VBI
```

I am certain that any points you are unsure about are illustrated in listing 5, so now you can write programs in BASIC using either of the DLI's and also have fast player movement. The source code (Listing 6) is commented for those of you who would like to see a slightly longer VBI routine.

Three last points need to be mentioned in this section on players. Firstly, priority can be selected by addressing GPRIOR (623). See table 1 for a list of possible priorities. Secondly,

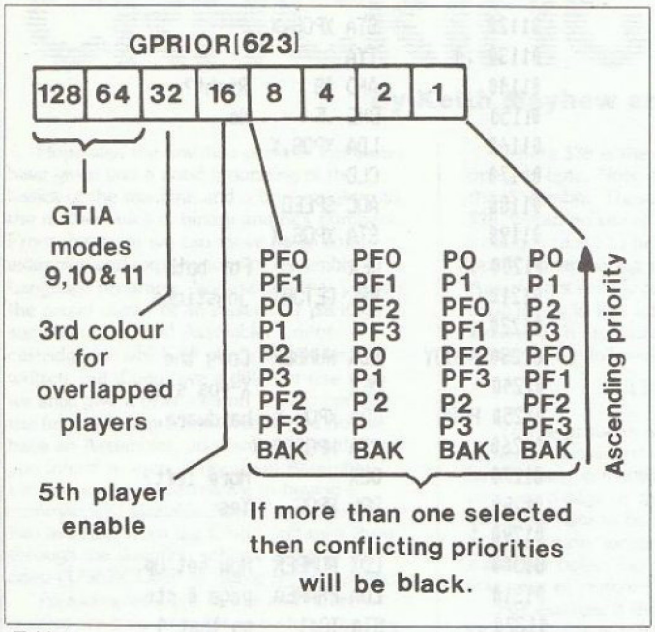


Table 1.

collision detection is effected by reading the appropriate register as shown in table 2. Thirdly, the width of the players can be altered by poking values into locations shown in table 3.

One final point, as you may realise, typing in immediate mode commands in BASIC will cause string locations to alter, and hence the VBI may well cause a lock-up. The moral is don't use any immediate mode commands after RUNNING this program.

In Part 2 I will continue with Horizontal and Vertical Scrolling techniques

Collision Registers

Name	Location	Description
M0PF	53248	Missile 0 to Playfield
M1PF	53249	Missile 1 to Playfield
M2PF	53250	Missile 2 to Playfield
M3PF	53251	Missile 3 to Playfield
P0PF	53252	Player 0 to Playfield
P1PF	53253	Player 1 to Playfield
P2PF	53254	Player 2 to Playfield
P3PF	53255	Player 3 to Playfield
M0PL	53256	Missile 0 to Players
M1PL	53257	Missile 1 to Players
M2PL	53258	Missile 2 to Players
M3PL	53259	Missile 3 to Players
P0PL	53260	Player 0 to Players
P1PL	53261	Player 1 to Players
P2PL	53262	Player 2 to Players
P3PL	53263	Player 3 to Players
HITCLR	53278	Poke to clear all registers

Collisions

Value Read	Collision with Player or Playfield Colour Number since last HITCLR
1	0
2	1
4	2
8	3

Table 2.

Player Sizes	Poke with . . .
Size of Player 0 - Location 53256	0 or 2 for Normal Size
Size of Player 1 - Location 53257	1 for Double Width
Size of Player 2 - Location 53258	3 for Quad Width
Size of Player 3 - Location 53259	

Table 3.

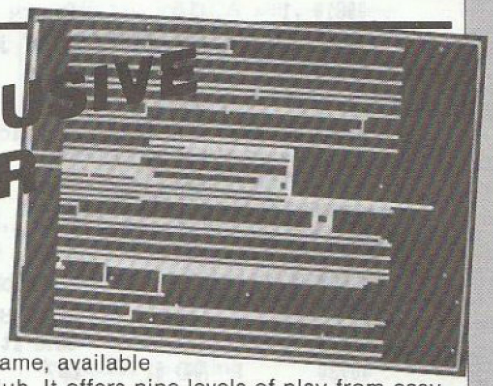
Button Blaster is a new device designed by the User Group Founders to relieve that agonising cramp of the thumb experienced by ardent games fanatics. Imagine playing those graphic games which allow more than one shot on the screen simultaneously and being able to press the trigger and have a continuous high speed burst of shots ring out! Well now you can with the BUTTON BLASTER.

Button Blaster has two leads, one roughly two metres long, the other about one metre (this also allows you to sit back in comfort whilst playing!). The longer lead plugs into your computer, in place of your joystick, and your joystick simply plugs into the socket on the shorter lead, now you are ready to blast. You can vary the auto-trigger rate using the control knob, and a flashing indicator (LED) shows the rate selected.

There is no need for batteries, since Button Blaster draws its power from the computer itself, and since it uses the lastest IC technology its current consumption is negligible!

Normal retail price for Button Blaster is £14.95, but we are offering it to user group members at the reduced rate of only £11.95 including P&P. Send a cheque/P.O. made out to the U.K. Atari Computer Owners Club, and please mark your envelope 'Button Blaster Offer'.

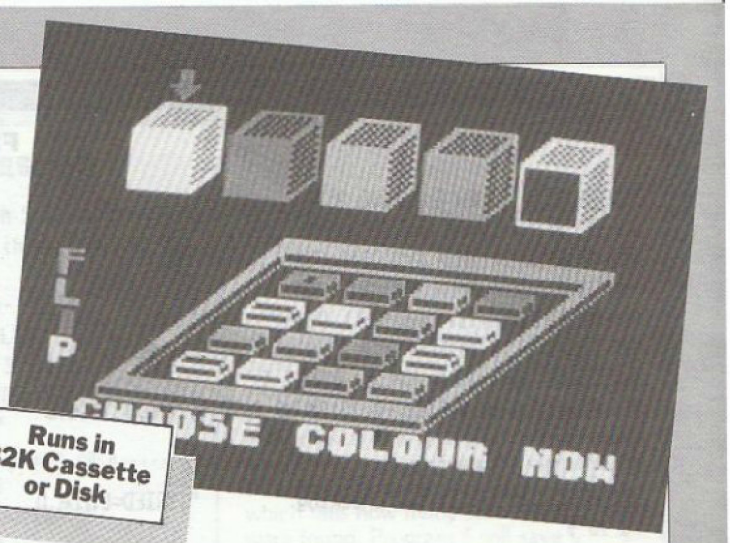
Buy BUTTON BLASTER now and perhaps double, or even treble your scores on your favourite games.



TRAP is a new, 100% machine code game, available only from this club. It offers nine levels of play from easy to impossible and is for one or two players. You and the 'enemy' must fill in all the open spaces with your trailing tails, avoiding the mines left in your way. If you out last all of the enemies then you are awarded a point for every enemy ranged against you on that screen, but if only one enemy out lives you then you get nothing. In the two player option, if all the enemies are destroyed then the player who lasts the longest gets the points. For every 100 points you collect you are given an extra life.

TRAP is obtainable by sending a cheque/postal order for £3.95, made out to the club, to P.O. Box 3, Rayleigh, Essex together with a blank cassette or disk. An auto-boot copy of TRAP will be recorded onto your cassette or disk and then returned to you. By marketing TRAP along similar lines to the Library Donation Scheme the overall costs can be kept low and many more people will feel the benefit. The price includes postage (U.K. & Eire) and packing but Overseas members should add 50p (Europe) or £1.00 (Outside Europe). Please mark your package 'TRAP OFFER' and remember it is best to send a good quality cassette or disk and also ensure adequate wrapping is provided.

FLIP



Runs in 32K Cassette or Disk

By Stephen Taylor - London

Flip is one of the most frustrating games I have come across. You are presented with a 4 x 4 grid of coloured tiles composed of 5 different colours, your object is to change the colour of every tile and flip it onto its back, but the colour you change it to must not conflict with adjacent tiles in any direction including diagonals. This is no easy task and so far all my attempts have failed! At the start of the program Stephen assures us that there is a solution, but I am not so sure. Apart from the frustrating nature of this game, the game does have redeeming features, such as the graphic display of the tiles which is in psuedo 3D, excellent sound effects and a lovely routine showing the tiles rotating or flipping over into place.

NOTE: In this program, anything which is underlined, should be entered in "INVERSE".

```
10 N=0:DIM NT(3,3),T(3,3)
20 MEMTOP=PEEK(106)*256
30 CHBASE=MEMTOP-2048
35 POKE 756,CHBASE/256
40 ? CHR$(125):POKE 752,1:GOSUB 8000
50 FOR I=CHBASE TO CHBASE+1024
60 POKE I,0
70 NEXT I
90 READ Z
100 IF Z=999 THEN 137
110 POKE CHBASE+N,Z:SOUND 0,Z,10,12
120 N=N+1
130 GOTO 90
137 SOUND 0,0,0,0:GOSUB 8012
140 POSITION 2,21:FOR I=0 TO 300:NEXT I:? "PRESS ANY KEY"
141 POSITION 2,21:FOR I=0 TO 300:NEXT I:? "              "
142 IF PEEK(764)=255 THEN GOTO 140
145 FOR I=0 TO 23:POSITION 0,0:? CHR$(157):SOUND 0,I+10,
10,6:NEXT I:SOUND 0,0,0,0
290 REM DRAW BOARD
300 GRAPHICS 18:FOR I=10 TO 0 STEP -0.25:SETCOLOR 2,0,I:
SETCOLOR 4,3,I:NEXT I
301 BLACK=3:RED=3:BLUE=3:GREEN=3:WHITE=4:A=4:F=0:G=0:H=0
303 NT(0,0)=2:NT(1,0)=1:NT(2,0)=2:NT(3,0)=1
304 NT(0,1)=4:NT(1,1)=0:NT(2,1)=3:NT(3,1)=0
305 NT(0,2)=3:NT(1,2)=2:NT(2,2)=1:NT(3,2)=4
306 NT(0,3)=4:NT(1,3)=0:NT(2,3)=3:NT(3,3)=2
307 FOR I=0 TO 3:FOR II=0 TO 3:T(I,II)=0:NEXT II:NEXT I
320 POKE 756,CHBASE/256
530 SETCOLOR 0,0,14:SETCOLOR 1,3,6:SETCOLOR 2,8,8:SETCOL
OR 3,11,8:SETCOLOR 4,4,0
540 RESTORE 6000
550 FOR ROW=0 TO 10
560 FOR COLUMN=0 TO 19
570 READ CHAR
580 COLOR CHAR
590 PLOT COLUMN,ROW
600 NEXT COLUMN
610 NEXT ROW
700 REM  MOVE TOP CURSOR
701 POSITION 2,11:? #6;"CHOOSE COLOUR NOW":C=18:D=6:B=10
702 POKE (77),0:REM **STOPS ATTRACT MODE
705 IF STICK(0)=11 THEN A=A-3:IF A<4 THEN A=4
710 IF STICK(0)=7 THEN A=A+3:IF A>16 THEN A=16
715 COLOR 122:PLOT A,0:FOR I=0 TO 100:NEXT I:COLOR 0:PLO
T A,0
720 IF A=4 THEN G=0:H=32
725 IF A=7 THEN G=1:H=0
730 IF A=10 THEN G=2:H=160
735 IF A=13 THEN G=3:H=128
740 IF A=16 THEN G=4:H=32
745 IF STRIG(0)=0 THEN FOR I=10 TO 0 STEP -0.25:SOUND 0,
40,10,I:NEXT I:POSITION 2,11:GOTO 748
746 GOTO 700
748 POKE 77,0:POSITION 2,11:? #6;"SELECT TILE      "
755 IF A=4 AND WHITE=0 THEN GOSUB 7000:GOTO 700
759 IF A=7 AND RED=0 THEN GOSUB 7000:GOTO 700
764 IF A=10 AND BLUE=0 THEN GOSUB 7000:GOTO 700
769 IF A=13 AND GREEN=0 THEN GOSUB 7000:GOTO 700
774 IF A=16 AND BLACK=0 THEN GOSUB 7000:GOTO 700
800 REM MOVE MIDDLE CURSOR
810 IF STICK(0)=11 THEN B=B-2:IF B<10 THEN B=10
815 IF STICK(0)=7 THEN B=B+2:IF B>16 THEN B=16
822 COLOR 216:PLOT B,4:FOR I=0 TO 100:NEXT I:COLOR 0:PLO
T B,4
823 IF STRIG(0)=0 THEN FOR I=10 TO 0 STEP -0.25:SOUND 0,
50,10,I:NEXT I:GOTO 900
825 GOTO 800
900 REM MOVE BOTTOM CURSOR
905 COLOR 224:PLOT C,D:FOR I=0 TO 100:NEXT I:COLOR 0:PLO
T C,D
910 IF STICK(0)=14 THEN D=D-1:C=C+1:IF D<6 THEN D=6:IF C
>18 THEN C=18
915 IF STICK(0)=13 THEN D=D+1:C=C-1:IF D>9 THEN D=9:IF C
<15 THEN C=15
920 IF STRIG(0)=0 THEN FOR I=10 TO 0 STEP -0.25:SOUND 0,
60,10,I:NEXT I:GOTO 930
922 GOTO 900
929 REM CAN YOU GO?
930 TRIED=0:YES=0:GOSUB 1000
933 IF TRIED=0 AND YES=0 THEN POSITION 2,11:? #6;"YOU CA
NT GO THERE":FOR I=0 TO 255:SOUND 0,255,10,10:SOUND 0,0,
0,0:NEXT I:GOTO 700
935 IF WHITE>0 THEN G=0:GOTO 950
```

```
937 IF RED>0 THEN G=1:GOTO 950
939 IF BLUE>0 THEN G=2:GOTO 950
941 IF GREEN>0 THEN G=3:GOTO 950
943 IF BLACK>0 THEN G=4:GOTO 950
944 POSITION 2,11:? #6;"WELL DONE      ":CX=0
945 FOR I=0 TO 10:FOR J=10 TO 0 STEP -0.5:SOUND 0,47+I,1
0,I:SOUND 1,40-I,10,J:NEXT J:NEXT I
946 FOR I=14 TO 0 STEP -0.25:SOUND 0,15,8,I:NEXT I:GOTO
290
950 IF CX=16 THEN GOTO 944
951 POSITION 2,11:? #6;"checking         ":TRIED=0:FOR D
=6 TO 9
952 FOR B=10 TO 16 STEP 2
953 YES=1
955 GOSUB 1000
957 NEXT B:NEXT D
961 IF TRIED=1 THEN GOTO 700
963 IF TRIED=0 AND G=0 THEN GOTO 937
965 IF TRIED=0 AND G=1 THEN GOTO 939
967 IF TRIED=0 AND G=2 THEN GOTO 941
969 IF TRIED=0 AND G=3 THEN GOTO 943
971 POSITION 2,11:? #6;"YOU CANT GO"
973 FOR I=100 TO 120:FOR J=14 TO 0 STEP -1:SOUND 0,I,10,
J:NEXT J:NEXT I:FOR I=119 TO 100 STEP -1:FOR J=0 TO 14:S
OUND 0,I,10,J:NEXT J:NEXT I
979 SOUND 0,0,0,0:POKE 764,255
980 POSITION 2,11:? #6;"PRESS ANY KEY":FOR I=1 TO 300:NE
XT I:POSITION 2,11:? #6;"              "
982 IF PEEK(764)=255 THEN FOR I=0 TO 300:NEXT I:GOTO 980
984 GOTO 290
1000 REM TURN OVER TILE, CHECK LEGAL MOVE
1005 REM 0,0
1010 IF B<>10 OR D<>6 THEN GOTO 1036
1015 IF T(0,0)=1 THEN GOTO 1036
1020 IF NT(0,0)=G OR NT(0,1)=G OR NT(0,1)=G OR NT(1,1)=G
 THEN GOTO 1036
1024 TRIED=1:IF YES=1 THEN RETURN
1025 T(0,0)=1:NT(0,0)=G:GOSUB 3000
1036 REM 1,0
1040 IF B<>12 OR D<>6 THEN GOTO 1060
1045 IF T(1,0)=1 THEN GOTO 1060
1049 IF NT(0,0)=G OR NT(1,0)=G OR NT(2,0)=G OR NT(0,1)=G
 OR NT(1,1)=G OR NT(2,1)=G THEN GOTO 1060
1050 TRIED=1:IF YES=1 THEN RETURN
1051 T(1,0)=1:NT(1,0)=G:GOSUB 3000
1060 REM 2,0
1062 IF B<>14 OR D<>6 THEN GOTO 1070
1064 IF T(2,0)=1 THEN GOTO 1070
1066 IF NT(1,0)=G OR NT(2,0)=G OR NT(3,0)=G OR NT(1,1)=G
 OR NT(2,1)=G OR NT(3,1)=G THEN GOTO 1070
1067 TRIED=1:IF YES=1 THEN RETURN
1068 T(2,0)=1:NT(2,0)=G:GOSUB 3000
1070 REM 3,0
1072 IF B<>16 OR D<>6 THEN GOTO 1080
1074 IF T(3,0)=1 THEN GOTO 1080
1076 IF NT(3,0)=G OR NT(2,0)=G OR NT(2,1)=G OR NT(3,1)=G
 THEN GOTO 1080
1077 TRIED=1:IF YES=1 THEN RETURN
1078 T(3,0)=1:NT(3,0)=G:GOSUB 3000
1080 REM 0,1
1082 IF B<>10 OR D<>7 THEN GOTO 1090
1084 IF T(0,1)=1 THEN GOTO 1090
1086 IF NT(0,1)=G OR NT(0,0)=G OR NT(1,0)=G OR NT(1,1)=G
 OR NT(1,2)=G OR NT(0,2)=G THEN GOTO 1090
1087 TRIED=1:IF YES=1087 TRIED=1:IF YES=1 THEN RETURN
1088 T(0,1)=1:NT(0,1)=G:GOSUB 3000
1090 REM 1,1
1092 IF B<>12 OR D<>7 THEN GOTO 2000
1094 IF T(1,1)=1 THEN GOTO 2000
1095 IF NT(1,1)=G OR NT(0,0)=G OR NT(1,0)=G OR NT(2,0)=G
 THEN GOTO 2000
1096 IF NT(0,1)=G OR NT(2,1)=G OR NT(0,2)=G OR NT(1,2)=G
 OR NT(2,2)=G THEN GOTO 2000
1097 TRIED=1:IF YES=1 THEN RETURN
1098 T(1,1)=1:NT(1,1)=G:GOSUB 3000
2000 REM 2,1
2002 IF B<>14 OR D<>7 THEN GOTO 2010
2004 IF T(2,1)=1 THEN GOTO 2010
2005 IF NT(2,1)=G OR NT(1,0)=G OR NT(2,0)=G OR NT(3,0)=G
 OR NT(3,1)=G OR NT(3,2)=G OR NT(2,2)=G THEN GOTO 2010
2006 IF NT(1,2)=G OR NT(1,1)=G THEN GOTO 2010
2007 TRIED=1:IF YES=1 THEN RETURN
2008 T(2,1)=1:NT(2,1)=G:GOSUB 3000
2010 REM 3,1
2012 IF B<>16 OR D<>7 THEN GOTO 2020
2014 IF T(3,1)=1 THEN GOTO 2020
2016 IF NT(3,1)=G OR NT(3,0)=G OR NT(2,0)=G OR NT(2,1)=G
 OR NT(2,2)=G OR NT(3,2)=G THEN GOTO 2020
2017 TRIED=1:IF YES=1 THEN RETURN
2018 T(3,1)=1:NT(3,1)=G:GOSUB 3000
2020 REM 0,2
2022 IF B<>10 OR D<>8 THEN GOTO 2030
2024 IF T(0,2)=1 THEN GOTO 2030
2026 IF NT(0,2)=G OR NT(0,1)=G OR NT(1,1)=G OR NT(1,2)=G
 OR NT(0,3)=G OR NT(1,3)=G THEN GOTO 2030
2027 TRIED=1:IF YES=1 THEN RETURN
2028 T(0,2)=1:NT(0,2)=G:GOSUB 3000
2030 REM 1,2
2032 IF B<>12 OR D<>8 THEN GOTO 2040
2034 IF T(1,2)=1 THEN GOSUB 2040
2035 IF NT(1,2)=G OR NT(0,1)=G OR NT(1,1)=G OR NT(2,1)=G
 OR NT(0,2)=G OR NT(2,2)=G OR NT(0,3)=G THEN GOTO 2040
2036 IF NT(1,3)=G OR NT(2,3)=G THEN GOTO 2040
2037 TRIED=1:IF YES=1 THEN RETURN
2038 T(1,2)=1:NT(1,2)=G:GOSUB 3000
2040 REM 2,2
2042 IF B<>14 OR D<>8 THEN GOTO 2050
2044 IF T(2,2)=1 THEN GOTO 2050
2045 IF NT(2,2)=G OR NT(1,1)=G OR NT(2,1)=G OR NT(3,1)=G
 OR NT(1,2)=G OR NT(3,2)=G OR NT(1,3)=G THEN GOTO 2050
2046 IF NT(2,3)=G OR NT(3,3)=G THEN GOTO 2050
2047 TRIED=1:IF YES=1 THEN RETURN
2048 T(2,2)=1:NT(2,2)=G:GOSUB 3000
2050 REM 3,2
2052 IF B<>16 OR D<>8 THEN GOTO 2060
2054 IF T(3,2)=1 THEN GOTO 2060
2056 IF NT(3,2)=G OR NT(2,1)=G OR NT(3,1)=G OR NT(2,2)=G
 OR NT(2,3)=G OR NT(3,3)=G THEN GOTO 2060
2057 TRIED=1:IF YES=1 THEN RETURN
2058 T(3,2)=1:NT(3,2)=G:GOSUB 3000
2060 REM 0,3
2062 IF B<>10 OR D<>9 THEN GOTO 2070
```

```
2064 IF T(0,3)=1 THEN GOTO 2070
2066 IF NT(0,3)=G OR NT(0,2)=G OR NT(1,2)=G OR NT(1,3)=G
 THEN GOTO 2070
2067 TRIED=1:IF YES=1 THEN RETURN
2068 T(0,3)=1:NT(0,3)=G:GOSUB 3000
2070 REM 1,3
2072 IF B<>12 OR D<>9 THEN GOTO 2080
2074 IF T(1,3)=1 THEN GOTO 2080
2076 IF NT(1,3)=G OR NT(0,2)=G OR NT(1,2)=G OR NT(2,2)=G
 OR NT(0,3)=G OR NT(2,3)=G THEN GOTO 2080
2077 TRIED=1:IF YES=1 THEN RETURN
2078 T(1,3)=1:NT(1,3)=G:GOSUB 3000
2080 REM 2,3
2082 IF B<>14 OR D<>9 THEN GOTO 2090
2084 IF T(2,3)=1 THEN GOTO 2090
2086 IF NT(2,3)=G OR NT(1,2)=G OR NT(2,2)=G OR NT(3,2)=G
 OR NT(1,3)=G OR NT(3,3)=G THEN GOTO 2090
2087 TRIED=1:IF YES=1 THEN RETURN
2088 T(2,3)=1:NT(2,3)=G:GOSUB 3000
2090 REM 3,3
2092 IF B<>16 OR D<>9 THEN GOTO 2100
2093 IF T(3,3)=1 THEN GOTO 2100
2094 IF NT(3,3)=G OR NT(2,2)=G OR NT(3,2)=G OR NT(2,3)=G
 THEN GOTO 2100
2095 IF NT(3,3)=G OR NT(2,2)=G OR NT(3,2)=G OR NT(2,3)=G
 THEN GOTO 2100
2096 TRIED=1:IF YES=1 THEN RETURN
2097 T(3,3)=1:NT(3,3)=G:GOSUB 3000
2100 RETURN
3000 REM  FLIP OVER TILE SUB.
3010 POSITION 2,11:? #6;"              ":CX=CX+1
3020 COLOR 15+H:PLOT B-D+4,D
3025 COLOR 16+H:PLOT B-D+5,D:FOR I=10 TO 19:SOUND 0,I,8,
10:SOUND 0,0,0,0:NEXT I
3026 COLOR 17+H:PLOT B-D+4,D
3027 COLOR 18+H:PLOT B-D+5,D:FOR I=40 TO 49:SOUND 1,I,8,
10:SOUND 1,0,0,0:NEXT I
3028 COLOR 19+H:PLOT B-D+4,D
3029 COLOR 20+H:PLOT B-D+5,D:FOR I=50 TO 60:SOUND 2,I,8,
10:SOUND 2,0,0,0:NEXT I
3030 IF G=4 THEN COLOR 23+H:PLOT B-D+4,D:COLOR 24+H:PLOT
 B-D+5,D:RETURN
3035 IF G<4 THEN COLOR 21+H:PLOT B-D+4,D:COLOR 22+H:PLOT
 B-D+5,D
4000 REM RUN OUT OF COLOUR?
4050 IF A=4 THEN WHITE=WHITE-1:IF WHITE=0 THEN GOTO 4070
4052 IF A=7 THEN RED=RED-1:IF RED=0 THEN GOTO 4070
4054 IF A=10 THEN BLUE=BLUE-1:IF BLUE=0 THEN GOTO 4070
4056 IF A=13 THEN GREEN=GREEN-1:IF GREEN=0 THEN GOTO 407
0
4058 IF A=16 THEN BLACK=BLACK-1:IF BLACK=0 THEN GOTO 407
0
4060 RETURN
4070 POSITION 1,11:? #6;"COLOUR NOW RUN OUT"
4071 COLOR 0:PLOT (G+1)*3,1:DRAWTO (G+1)*3+2,1
4072 COLOR 0:PLOT (G+1)*3,2:DRAWTO (G+1)*3+2,2
4074 COLOR 0:PLOT (G+1)*3,3:DRAWTO (G+1)*3+2,3
4075 FOR I=0 TO 300:NEXT I:POSITION 0,11:? #6;"
    "
4080 RETURN
4999 REM NEW IMPROVED CHARACTER SET !!
5000 DATA 0,0,0,0,0,0,0,0
5001 DATA 0,255,255,255,255,128,128,255
5002 DATA 0,0,0,0,0,0,0,0
5003 DATA 0,255,255,255,255,0,0,255
5004 DATA 0,255,255,255,254,60,121,242
5005 DATA 0,192,192,64,64,128,0,0
5006 DATA 228,200,144,32,64,128,0,0
5007 DATA 1,3,7,15,30,60,121,242
5008 DATA 1,255,255,255,254,2,3,254
5009 DATA 121,255,255,255,255,0,0,255
5010 DATA 0,0,1,3,7,4,4,7
5011 DATA 0,0,1,3,7,15,30,60
5012 DATA 121,242,228,200,144,32,64,128
5013 DATA 0,31,63,127,255,128,128,255
5014 DATA 0,254,254,250,242,20,24,240
5015 DATA 0,0,255,128,128,255,0,0
5016 DATA 0,0,254,18,18,254,0,0
5017 DATA 0,255,128,128,255,127,63,31
5018 DATA 0,240,24,20,242,250,254,254
5019 DATA 63,64,255,255,255,255,255,255
5020 DATA 252,4,244,244,244,244,248,240
5021 DATA 0,31,62,124,255,128,128,255
5022 DATA 0,254,126,250,242,20,24,240
5023 DATA 0,31,33,67,255,128,128,255
5024 DATA 0,254,134,10,242,20,24,240
5025 DATA 0,31,32,64,255,128,128,255
5026 DATA 0,254,6,10,242,20,24,240
5027 DATA 255,128,128,128,128,128,128,128
5028 DATA 255,1,1,1,1,1,1,1
5029 DATA 0,1,3,7,14,29,58,117
5030 DATA 0,255,170,85,170,85,170,85
5031 DATA 0,254,174,94,186,118,234,214
5032 DATA 16,48,127,255,255,127,48,16
5033 DATA 0,124,238,238,238,254,238,238
5034 DATA 0,252,238,252,238,238,254,252
5035 DATA 0,124,230,224,224,230,254,124
5036 DATA 0,248,236,230,230,238,254,252
5037 DATA 0,254,224,254,224,224,254,254
5038 DATA 0,254,224,254,224,224,224,224
5039 DATA 0,124,230,224,238,230,254,126
5040 DATA 0,238,238,254,254,238,238,238
5041 DATA 0,60,60,60,60,60,60,60
5042 DATA 0,28,28,28,28,220,252,120
5043 DATA 0,238,236,248,252,238,238,238
5044 DATA 0,224,224,224,224,224,254,254
5045 DATA 0,230,254,254,218,230,230,230
5046 DATA 0,238,246,254,254,238,238,238
5047 DATA 0,124,238,238,238,238,254,124
5048 DATA 0,252,230,230,254,252,224,224
5049 DATA 0,124,238,238,238,238,252,126
5050 DATA 0,252,238,252,238,238,238,238
5051 DATA 0,124,224,124,14,14,254,252
5052 DATA 0,254,56,56,56,56,56,56
5053 DATA 0,238,238,238,238,238,254,124
5054 DATA 255,255,255,255,255,255,255,255
5055 DATA 0,230,230,230,218,254,254,230
5056 DATA 4,142,223,254,252,248,252,254
5057 DATA 0,238,238,124,56,56,56,56
5058 DATA 60,60,60,60,255,126,60,24
5059 DATA 128,128,128,128,128,128,128,255
5060 DATA 1,1,1,1,1,1,1,255
```

Continued on Page 29

INTERFACE

MEMORY SHRINKER!

Dear Sirs,

My reason for writing to you is that I have typed in numerous magazine listings, but have been unable to run a fair percentage of them. I think this is because my machine is a 48K 800 and most of the programs were for a 16K machine and these programs probably have a line in them that is controlled by the machine's memory capacity that prevents them from running properly. Is there any way that I can POKE a lower memory value in order to fool my computer into thinking it is only a 16K version?

I don't mind the typing in (it's all good practice isn't it?), but it is very frustrating to see four or five (or even eight) days typing turn into a blank screen when run.

A. Pullinger - Aylesbury.

COMMENT

Yes, it is frustrating typing in pages of listing only to find they don't work. We wish those national glossy mags would at least say what memory is required, and the author of the piece write it so that it would work in any size machine, which is not so much extra work.

The ATARI computers keep a series of pointers which keep track of such things as the screen memory, which moves about depending on how much memory is free. Programs should access things via these pointers and not directly because they do not keep to one fixed location. The reason why you are getting blank screens is that your screen memory is up near 48K, but the program is assuming that your screen is in the 16K region.

There is one location in the computer that keeps a record of how many 'pages' of RAM there are in your machine. So, if you PEEK location 106 you can see how many pages you have, and multiplying it by 256 gives you the number of bytes. For a 48K machine you get returned 160 pages or 40K bytes (remember the other 8K is BASIC).

To make the machine think it has less memory, you need to POKE a lower value into location 106, this will not change anything instantly, but a call to GRAPHICS 0 will make the machine re-calculate its memory pointers for the smaller memory available. If you add as the first line of your program: POKE 106,64:GRAPHICS 0 then this will set your memory to 16K, and similarly for 32K you would POKE 106 with 128.

DESCRIPTIVE PROGRAMMING

Once you are used to looking at program listings you begin to notice that most programmers never seem to write anything in a straightforward manner. They always seem to be redefining the alphabet so that their programs end up with lines like: A=1:B=A+A:C=A+B etc... Halfway through a program, can they really remember what all those symbols meant? A more descriptive approach could be taken thus: ZERO=0:LET ONE=1:TWO=ONE+ ONE etc... a much easier system to remember! Or even better you could use LET ONE =NOT ZERO!! Incidentally ONE will be rejected without LET.

However, nobody likes to type in such cumbersome things as SOUND ZERO,ZERO,ZERO,ZERO:POKE SEVENHUNDREDANDSIXTYFOUR, TWOHUNDREDANDFIFTYFIVE. So my favourite approach is the 'Roman System' which gives such wonderful lines as:
I=NOT Z:II=I+I:III=II+I:IV=II+II:V=III+II: X=V+V:XX=X+X etc... I also include a few specials such as PAGE=256:FF=PAGE-I. With this system at line umpteen thousand you can still remember what XX equals.

Most of us are familiar with a statement such as TRAP 1000 to cause line 1000 to execute in the event of an error, but the statement TRAP 40000 usually makes us look twice (40000 is used as a deliberate error to end the special trapping arrangements). A more elegant and descriptive way would be to use TRAP END (provided there is no line zero).

What is needed is a common standard to make programs understandable to all, say the Skeggs Atari Standard Comprehensible Information Interchange, not a B.S spec. more a SASCII standard!!

Trevor Skeggs - Milton Keynes.

Editor's Note: This is a joke, isn't it Trevor?

MICROSOFT FORMAT

Dear Sir,

I am writing to you to ask for help with a software problem that I have encountered. My problem concerns ATARI Microsoft BASIC V1.0 on diskette. I have written a large database suite of programs for my personal video library. The amount of data involved is so large that one diskette is not enough. Therefore my program asks for new diskettes when the current diskette is full. The problem occurs, because when this new diskette is called for, I have to leave BASIC to format a new diskette. As you can imagine this takes quite a bit of time and is very inconvenient. The solution to my dilemma would be a routine that could format the diskette within BASIC. Can you help?

A. Lusher - Erith, Kent.

COMMENT

If you had been using ATARI BASIC, formatting the diskette is easily achieved using the flexible XIO command: XIO 254,#1,0,0,"D:". Microsoft BASIC however, does not have such a command and you need to access the Central Input/Output routines via machine code calls. On the Microsoft BASIC disk is a machine code data file called 'CIOUSR'. The following program reads the machine code from the file and then formats a diskette by using the disk handler's format command (hex FE). Also given is a description of how the program works so that it should not be too difficult for you to incorporate it as a subroutine into your main program.

```
100 !
110 !Microsoft BASIC.
120 !
130 !Format diskette.
140 !
150 !By Keith Mayhew.
160 !
170 DIM IOCB%(10)
180 OPTION RESERVE 160
190 ADDR=VARPTR(RESERVE)
200 OPEN#1,"D:CIOUSR" INPUT
210 FOR I=0 TO 159
220 GET#1,X:POKE ADDR+I,X
230 NEXT:CLOSE#1
240 DISK$="D:"
250 IOCB%(0)=1
260 IOCB%(1)=&FE
270 ICBADDR=VARPTR(DISK$)
280 ELEMENT=VARPTR(IOCB%(3))
290 POKE ELEMENT,PEEK(ICBADDR+2)
300 POKE ELEMENT+1,PEEK(ICBADDR+1)
310 PUTIOCB=ADDR
320 CALLCIO=ADDR+61
330 ARRAY=VARPTR(IOCB%(0))
340 INPUT "Press [RETURN] to format
 disk.";DUMMY$
350 X=USR(PUTIOCB,ARRAY)
360 X=USR(CALLCIO,ARRAY)
370 PRINT "Disk formatted."
380 END
```

The program first dimensions IOCB% to 10, which will contain the information to format the disk. The OPTION RESERVE command reserves 160 bytes of memory into which the data of the 'CIOUSR' file is read, byte by byte. Disk$ is set to the file spec. of the device to be accessed (D:), and then the array IOCB% is set up for the format command. Element 0 is set to 1 and is the channel number, the next element is the command number (&FE - format). ICBADDR is set to the address of the string DISK$ and ELEMENT is set to the address of the third element of IOCB%. The two byte integer ICBADDR is then POKEd into IOCB%(3), PUTIOCB is the address of

the machine code to transfer the IOCB% array into CIO and CALLCIO is the address for executing the command. The two routines are then called in sequence, both being passed the address of the array, and then the disk is formatted! Simple isn't it?

HARDWARE ADVICE

Dear UKACOC,

I am writing for a couple of reasons. One is I think that your magazine has improved considerably since the first one. Keep up the good work. I am impressed with the ability of some of the subscribing people out there and yours as well!

The small program that I submitted and was printed in Issue 5 about artifacting was probably a big disappointment to most people. I have just seen it run on a UK spec. machine and it's a real dud. From my investigations, I have found out that with the PAL/UK TV interface there is more "resolution" in the PAL system (more lines) and that artifacting does not come out as good as compared to the US NTSC system. Some of the people I know have also complained about games that use artifacting for the colours. The result on a UK system is not what you would call vivid Atari Graphics!! I am using a US spec. Atari 800 here in the UK with an NTSC colour monitor and of course, the games (and my demo) look much better. Sorry about that!!!

Which brings me to the last point. I noticed several people have asked about hardware additions to their Ataris. I have several modifications/additions to my computer and they are welcome to get in contact with me if they could use what advice or help that I might be able to give. My set-up consists of an Atari 800 (USA model) with a Newell Industries Ramrod MMOS board in the first slot. This board has Omnimon! and the Fastchip floating point chip installed in it. Also I have a different operating system which is on EPROMS also on the board. The board allows EPROMS or the standard ROMS. Also, it is possible to remove the OMNIMON! EPROM and install 4K of RAM which fills up the "hole" that Atari left in the system. The second slot has the standard Atari 16K RAM board. The third slot has an Intec 32K board for a total of 48K. The last slot has a Bit-3 80 column card installed which is in turn hooked up to a green screen Zenith monitor. This works great with the 80 column versions of LJK's Data Perfect and Letter Perfect.

I also have SWP's ATR8000 peripheral which handles my printer (Epson MX-80 w/Graftrax +) and disk drives (two slimline double sided Shugarts and a Percom RFD40-S2). I can also run CP/M 2.2, but this is my latest addition and I can't say much about it other than it works with no problems so far. In the "Atari" mode, the ATR is also a 60K printer buffer which I have found to be useful. Finally the video output from my 800 runs to a Panasonic colour monitor with sound. I have had no problem with the system here in the UK. I run the entire set-up through a step down transformer. Oh, and I also have a 410 recorder.

For anybody who wishes to contact me my name and address is Kirby Schrader, 13 Earlswells View, Cults, Aberdeen, Scotland. My telephone number is 0224-861226 and if I'm not in, leave your number and I'll call you back when I get in.

COMMENT

My, what a big system you have Kirby! We are sure many people will be interested to know more about your equipment. Stand-by for lots of letters and phone calls.

Continued from Page 27.

FLIP

```
5061 DATA 255,255,255,255,255,255,255,255
5062 DATA 170,86,170,86,170,86,170,86
5063 DATA 170,86,172,88,176,96,192,128,999
5999 REM DRAW PLAYFIELD DATA
6000 DATA 0,0,0,0,122,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6002 DATA 0,0,0,61,62,63,29,30,31,189,190,191,157,158,15
9,61,62,63,0,0
6004 DATA 0,0,0,86,86,94,118,118,126,214,214,222,246,246
,254,59,60,94,0,0
6006 DATA 0,0,0,86,86,95,118,118,127,214,214,223,246,246
,255,91,92,95,0,0
6008 DATA 0,0,0,0,0,0,0,0,0,216,0,0,0,0,0,0,0,0
6010 DATA 0,0,0,0,0,0,0,171,161,163,163,163,163,163,163,
163,163,164,165,0,0
6012 DATA 230,0,0,0,0,171,172,173,174,13,14,173,174,13,1
4,167,166,224,0,0
6014 DATA 204,0,0,0,171,172,57,58,45,46,141,142,45,46,16
7,166,0,0,0,0
6016 DATA 105,0,0,171,172,141,142,173,174,13,14,57,58,16
7,166,0,0,0,0,0
6018 DATA 80,0,171,172,57,58,45,46,141,142,173,174,167,1
66,0,0,0,0,0,0
6020 DATA 0,170,169,163,163,163,163,163,163,163,163,168,
166,0,0,0,0,0,0,0
7000 REM MESSAGE
7005 POSITION 2,11:? #6;"TRY AGAIN PLEASE"
7010 FOR J=230 TO 250:SOUND 0,J,10,10:FOR I=0 TO 3:NEXT
I:SOUND 0,0,0,0:FOR I=0 TO 3:NEXT I:NEXT J
7020 RETURN
7999 REM INSTRUCTIONS
8000 SETCOLOR 2,0,10:SETCOLOR 1,0,0:SETCOLOR 4,3,10:? CH
R$(125)
8001 POSITION 10,7:? "FFFFF LL    II PPPP "
8002 POSITION 10,8:? "FFFFF LL    II PPPPP"
8003 POSITION 10,9:? "FF    LL    II PP PP"
8004 POSITION 10,10:? "FFFFF LL    II PPPPP"
8005 POSITION 10,11:? "FFFFF LL    II PPPP"
8006 POSITION 10,12:? "FF    LLLLL II PP"
8007 POSITION 10,13:? "FF    LLLLL II PP"
8008 POSITION 7,16:? "DESIGNED BY STEPHEN TAYLOR "
8010 RETURN
8012 FOR I=0 TO 23:POSITION 0,0:? CHR$(156):NEXT I
8019 ? "HOW TO PLAY FLIP":? :?
8020 ? "YOU MUST REPLACE ALL THE TILES WITH   ONE ";
8030 ? "THE COLOUR OF YOUR OWN CHOOSING":?
8040 ? "USE THE JOYSTICK AND BUTTON":?
8050 ? "YOU CAN ONLY USE EACH COLOUR SO MANY"
8060 ? "TIMES":?
8070 ? "PICK THE TILE YOU WANT TO CHANGE"
8080 ? "IT MUST NOT BE THE SAME COLOUR AND"
8090 ? "IT MUST NOT BE AJACENT TO THE SAME "
8100 ? "COLOUR TILE IN ANY DIRECTION":? :?
8110 ? "GOOD LUCK"
8115 ? "THERE IS A SOLUTION"
8900 RETURN
```

REVIEWS*REVIEWS*REVIEWS

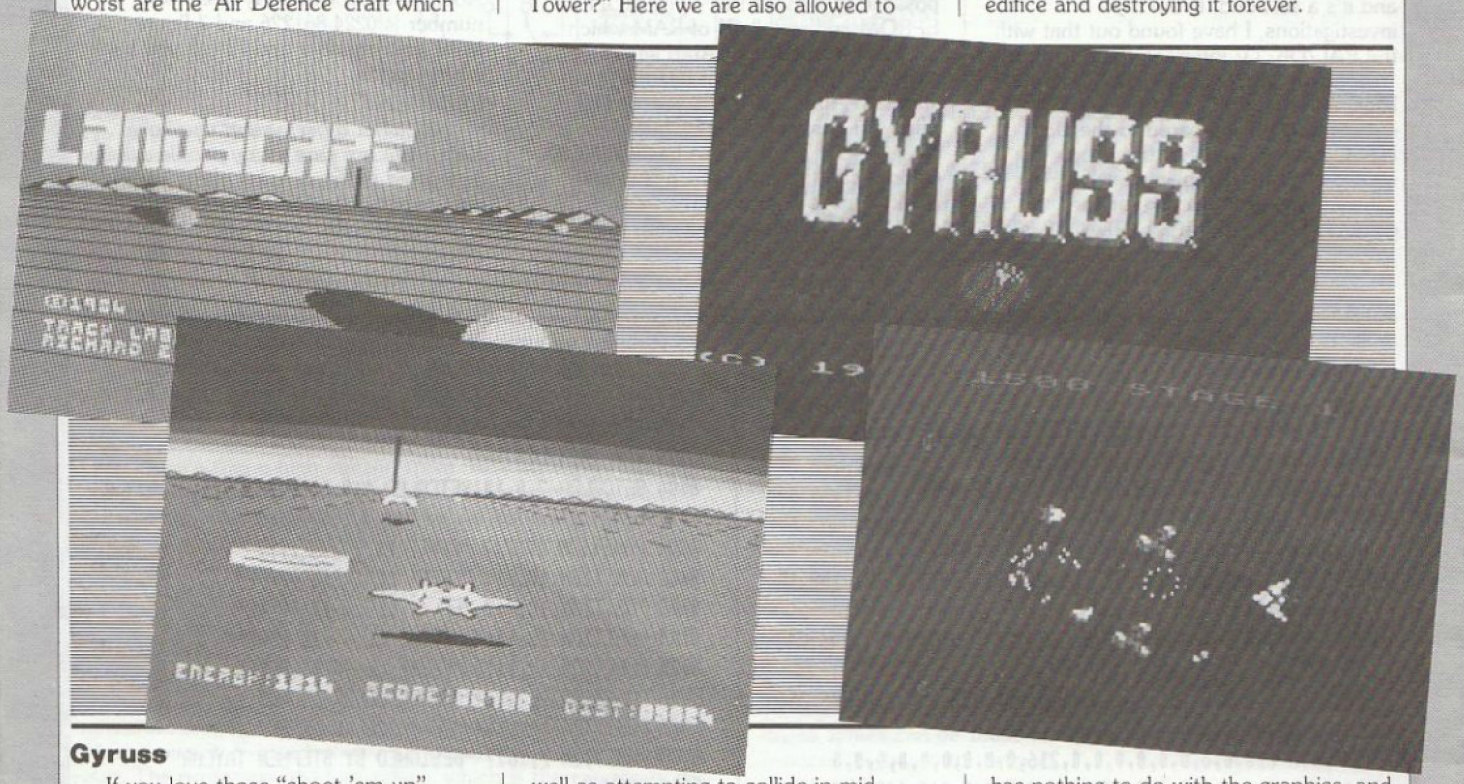
by Ralph Kingsley

Landscape

Landscape is a cross between 'Buck Rogers' and 'Encounter' in so far as we are treated to the rear view of our spaceship as it twists and turns across the scrolling landscape avoiding enemy fire as well as giving a good account of itself in the return fire department. The object of the exercise is to traverse the landscape to reach the 'Dark Tower' which can be seen in the distance, and when you finally achieve your goal, to destroy the tower by shooting out its base to make it come crashing down, chimney stack fashion. Various enemy formations are out to stop you from reaching the tower including static and mobile ground forces, but by far the worst are the 'Air Defence' craft which are not only exceedingly accurate with their shots, but also have this terrible desire to be kamikazi pilots and swoop down straight into you, nasty beasts indeed!

At the start, we are presented with an interesting title page which depicts a 3D landscape with the mountains and the ominous Dark Tower as the skyline, and dotted in various positions are large coloured balls that cast elliptical shadows onto the sloping plain to give a rather 'op-art' effect. A well thought out opener and also stunning to the eye in its own way. Next we are given a list of the enemy forces ranged against us, combined with the provocative statement "Can you destroy the Dark Tower?" Here we are also allowed to select the level of difficulty, which basically consists of changing the distance we need to travel to reach the tower. Lastly, before we start on our mission, we go to the tactical screen, which gives details of the optimum route to the tower. Then it's full speed ahead into the inferno.

Your ship appears at the edge of the grid, you push forward on your stick to accelerate and you are on your way. At first the enemy defences seem weak and thin on the ground but don't be deceived, the closer you get to the tower the more active the enemy gets. You only have three ships to play with so you must last as long as possible to stand any chance of reaching the dark edifice and destroying it forever.



Gyruss

If you love those "shoot 'em up" type games in the tradition of Defender and Space Invaders then you'll just love Gyruss. The concept of the game is that you are spiralling your spaceship into the Solar System from somewhere in outer space, coming in on a trajectory aimed at good old Earth, passing by Neptune, Uranus, Saturn et al. As you approach each planet defending craft attack you, usually spiralling out from the centre, although they can be sneaky and come at you from behind. Your ship rotates in a circle as you move your joystick left and right, together with a background starfield which radiates out from the centre, giving a realistic impression off travelling down into a tunnel. Each atacking wave consists of fighters which fire at you as well as attempting to collide in mid-space. Molecular satellites also attack you and are usually to be found in threes, if you destroy the middle satellite you receive the added bonus of firing double lasers. Other hazards include meteorites and 'electromagnetic' satellites which line up in pairs and produce a lethal energy wave between them which must be avoided.

As you pass each wave of attackers, you warp to the next stage and every time you pass a planet you are given a bonus screen of enemy ships who can't fire back, this is the ideal situation for those who love to gather lots of points. Speaking of points, you are awarded an extra ship at 60,000 points and at every 100,000 points thereafter.

One of the best features of the game has nothing to do with the graphics, and that's the sound. The music that accompanies the game all the time you are playing is Bach's Toccatta and Fugue in D Minor, which is probably familiar to you if you are a John Williams fan. Far from annoying after several minutes like some music does on some games (the constant music on Astro Chase and Shamus to name but two) it actually enhances the game and makes it more enjoyable.

Parker Brothers who produce Gyruss are to be congratulated on developing an old theme into an original and entertaining program, this surely has to be one of their best so far. I put it into my personal number two spot (Boulder Dash is number one of course, what else?).

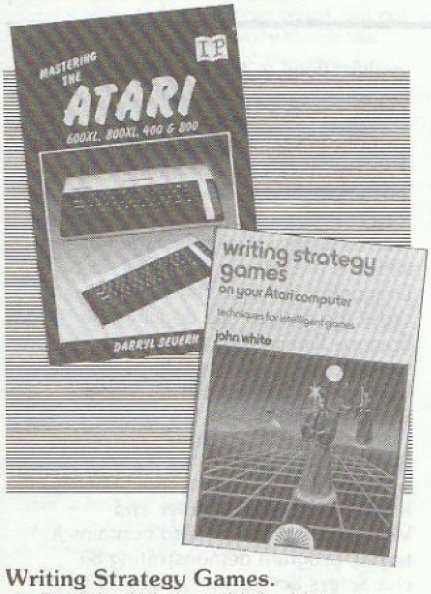
READ ALL ABOUT IT

Some Book Reviews by Bradley Mountjoy

Mastering the Atari.

By Daryl Severn, published by Interface Publications, 9-11 Kensington High Street, London W8 5NP.

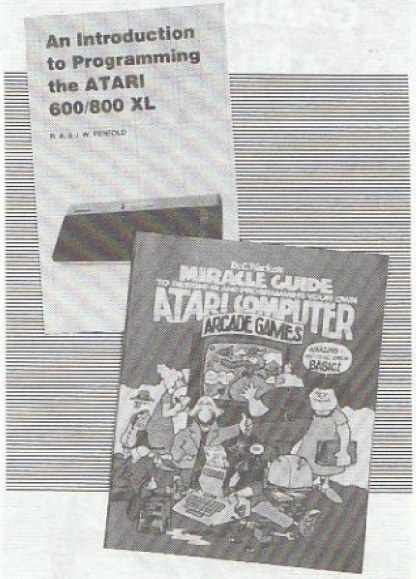
This book has clearly been written with the intention of providing a good working knowledge of the Atari computers and as such it starts by recommending that the best way to get the most out of the book is to be at your keyboard as you read so that you can enter the programs as they come up. Four main topics are covered and these are Strings, Input/Output, Logic and Graphics and each section is generously filled with example programs. Of the four sections, the graphics section is the largest and includes much information on Display Lists, Player-Missile Graphics and Scrolling Techniques. The I/O chapters include interesting tid-bits on the Screen Editor, the Keyboard, Joystick Ports, as well as explanations of how the Cassette and Disk handling systems work. This book is not for the very beginner but assumes a reasonable knowledge of Basic and is aimed at the enthusiast who wants to know more about his machine.



Writing Strategy Games.

By John White, published by Sunshine Books, 12-13 Little Newport Street, London WC2R 3LD.

To write games of strategy requires not only a knowledge of programming but also of certain mathematical and coding techniques. This book looks at the programming theory behind intelligent games before moving on to practical examples of how to set up a board, move pieces, standard openings and endgame moves. Included are sample games for draughts, chess and other board games, and the author has also provided his own board game called 'Warp Trog', which illustrates most of the techniques discussed in the book.



An Introduction to Programming the Atari 600/800XL.

By R.A. & J.W. Penfold, published by Babani, The Grampians, Shepherds Bush Road, London W6 7NF.

The step-by-step approach adopted by this book starts with the fundamentals of Basic and then moves on to more advanced topics such as animated graphics. There are chapters on Variables and Arrays, Strings, Inputting Data, Decision Making, Sound Generation, Graphics and Ins & Outs such as Paddles, Lightpen, Interfacing, and Timers. There are plenty of program listings given to illustrate various points. All in all a useful little volume especially as it is priced at under £2.00.

Dr. C. Wacko's Miracle Guide to Designing and Programming your own Atari Computer Arcade Games.

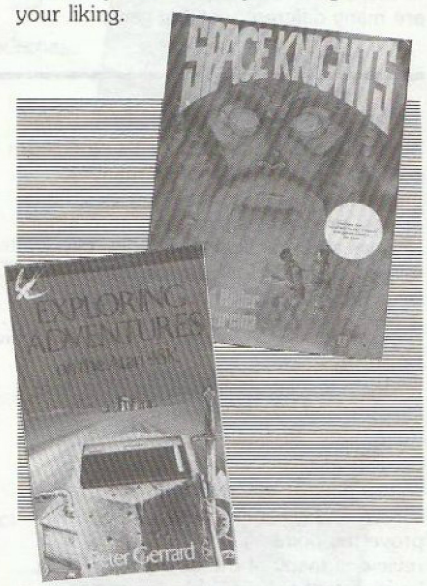
By David Heller, John F. Johnson & Robert Kurcina, published by Addison-Wesley, Finchampstead Road, Wokingham, Berks RG11 2NZ.

As well as having the longest book title I have ever come across, this volume also manages to be the most entertaining programming book I have read. It is packed with cartoons and jokes in the true tradition of West Coast of America wackiness. Dr. Wacko guides us through all the necessary details with such fun and style that you hardly notice that you're learning as you go. Subjects covered include Graphics modes, Character graphics, Flip-flop animation, Joystick control, PMG, and 'Zounds' (Sounds?). In Dr. Wacko's opening speech, I quote: "Holy Zanzibar! Are you going to be glad that you bought this book!" Well, all I can say is: "I was Doc."

Space Knights.

By David Heller & Robert Kurcina, published by Reston Publishing, Reston, Virginia 22090, U.S.A.

Space Knights is a book and a disk combined to give an entertaining read coupled with ready to run programs which slot into various action sequences in the novel. What you might call a novel concept! The idea is that you can read through the story of the adventures of Jack and Lisa and then at certain points load up the appropriate game and try out your skills or, if you are like me, trust to luck to win through. About two thirds of the book is taken up by the story, which is nicely illustrated, the rest of the book being devoted to game descriptions and rules. There are nine games with titles such as Bug Buster, Weomby, War Room, Navigate and Mind Demons. Some of the games require the use of paddle controllers but most are joystick controlled. If you are a Sci-Fi fan or just looking for something different, then I am sure you will find Space Knights to your liking.



Exploring Adventures on the Atari 48K.

By Peter Gerrard, published by Duckworth, The Old Piano Factory, 43 Gloucester Crescent, London NW1.

This book is a thoroughly comprehensive guide to writing your own adventures in BASIC. It assumes only a very limited knowledge of BASIC, as one chapter provides a useful summary of the BASIC commands used in writing adventures. Another chapter breaks down a complete adventure into small sections, and then discusses exactly how each works. This is a particularly good way of showing how an adventure can be written. Everything you need to know is explained, from creating the initial map of the game to

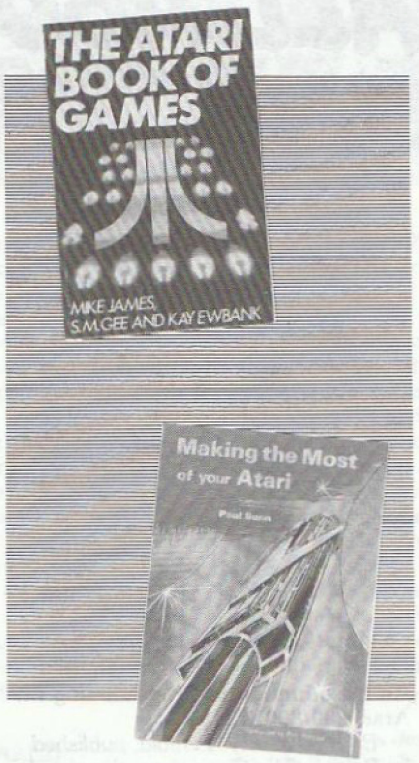
READ ALL ABOUT IT

bringing in random elements, such as small dwarves attacking you. Although the book limits itself to text only types of adventure, there is an overview of graphic adventures and fantasy role playing games. Also, if you're lost for ideas, there is a large selection of themes for possible adventures. Perhaps best of all, there are three complete adventure listings provided, but if this amount of typing is too daunting, you can send off for a cassette of these games. In summary then, this book is well worth the price, giving clear instructions on producing a high quality game with minimum effort.

The Atari Book of Games.

By Mike James, S.M. Gee & Kay Ewbank, published by Granada Publishing, 8 Grafton Street, London W1X 3LA.

Twenty-one listings of games programs are presented in this book together with a description of the object of the game and details of how to play. All the games are written in Basic but show how much can be achieved without resorting to machine code. Each program also has tips and hints provided so that you can modify them to make your own improvements. There are many different types of game



included such as: Bobsleigh, Capture the Quark, Treasure Island, Smalltalker, Sheepdog Trials, Save the Whale, to name but a few. The listings are well printed and easy to read and in addition 'typing tips' are given if there are any special control codes required.

Making the Most of Your Atari.

By Paul Bunn, published by Interface Publications, 9-11 Kensington High Street, London W8 5NP.

Aimed at the computer owner who has had his machine only a short time, this book provides details of much of the information that a beginner needs to improve his programming attempts. It does this by providing just enough on each topic to make it usable without going into lengthy explanations of why and how. It includes information on all the Graphics modes and associated Basic commands such as Plot, Drawto, Position, XIO (fill), Setcolor, etc. The Error message system, PMG including Collision Detection and the fifth player, Redefining characters, and making programs more efficient are also discussed. Finally, Paul has added fourteen games programs for your enjoyment which include Frog Jump, Astro Blast, Sales Analysis and Grand Prix.

BACK ISSUES

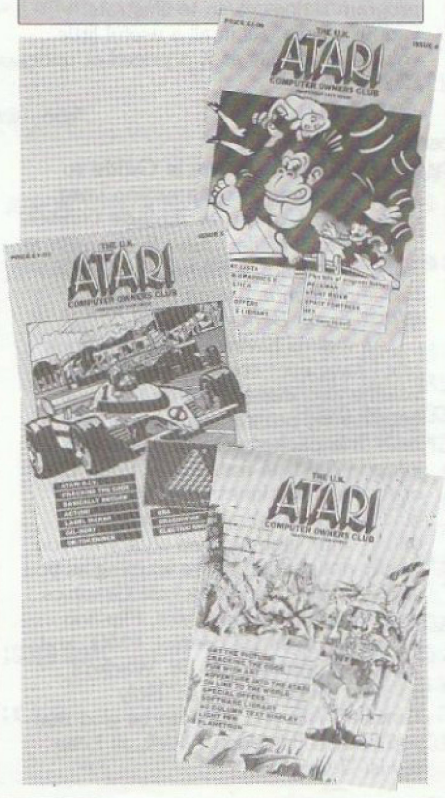
Previous issues of this magazine are obtainable from the club for £1 plus 30p. postage each. They contain many interesting and informative articles, hints & tips, program listings for you to input, reviews and practical advice. If you have missed out send for your copies of back issues today! Please note that issues 1, 2, and 3 are already sold out.

Issue 4.

Includes a complete in-depth look at Display Lists, what they are, how to use them, LMS explained, horizontal and vertical scrolling, etc. Another article shows how to get text on a Graphics 8 screen and gives an example graph to prove the point. A comprehensive review of many of the different types of joystick that are available gives ratings for comfort, action, looks and value. Program listings are aplenty and include Peckman, a Basic version of a well-known arcade game, Stunt Rider, in which you must jump your motorbike over the buses, Hex is a two player board game with excellent graphics, and for the more serious minded, you can enjoy designing your own shapes with CAD (computer assisted design).

Issue 5.

The first part of the series on 'Cracking the Code' starts in this issue and covers Binary, Hexadecimal and Decimal mathematics. There is an article on protecting your Basic programs from prying eyes and an interesting article on hardware



modifications to the 800/400 machines to give improved sound and picture quality, a cold start key and a busy light for your cassette player. Also included is a review of the new programming language. 'Action!' showing its potential for creating exciting fast action games. Games listings shown include Gil-bert, which is a 'Q-bert' type game, also Dragonfire in which the player must cross the drawbridge dodging the dragons flaming breath to reach the treasure room. Other listings include a label maker and a QRA locator for Radio Amateurs.

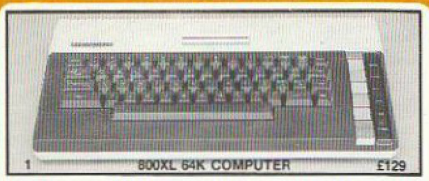
Issue 6.

Includes a useful tutorial showing how to print Micropainter and Versawriter pictures, also contains a terrific program demonstrating 80 characters across the screen. A new regular column for adventure enthusiasts is started to give reviews of adventure games and give hints and tips on how to play them. Part two of Cracking the Code continues with addressing modes and binary sums. The hardware design for a Light Pen is shown together with some simple programs to use with it once you have built it. Fun with Art from Epyx is reviewed and some of the excellent results of using this package are shown. Programs include Planetron and an RTTY listing for use with a short wave band radio, the Atari 850 interface and a signal terminal unit (such as the Maplin TU1000).

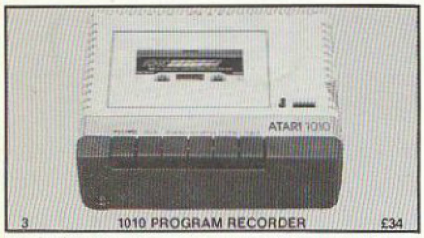
THE FIRST 64K COMPUTER FOR ONLY £129!

ATARI XL

THE NEW ATARI 64K 800XL £129

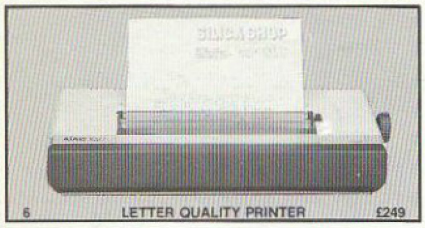

1 800XL 64K COMPUTER £129


2 400 16K GAMES MACHINE £29

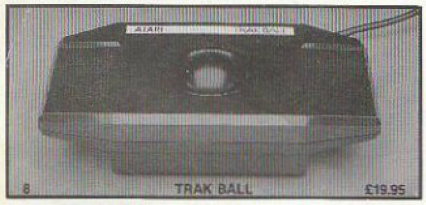

3 1010 PROGRAM RECORDER £34


4 1050 127K DISK DRIVE £199


5 1020 4 COLOUR PRINTER £99


6 LETTER QUALITY PRINTER £249


7 TOUCH TABLET PEN + CART £49


8 TRAK BALL £19.95


9 SUPER CONTROLLER £9.95

EVERYTHING YOU WANT FROM A HOME COMPUTER

1. **ATARI 64K 800XL - £129:** The Atari 800XL has many facilities and includes such advanced specifications that you will be amazed by its performance. At the new reduced price of only £129 inc VAT for a full specification 64K computer with a proper full stroke keyboard, we believe that the 800XL cannot be beaten. Compare Atari with the competition, just look at these specifications:-
   COLOUR CAPABILITIES: 16 colours and 16 intensities giving 256 different colours (all of the 256 colours can be displayed at the same time).
   OPERATING SYSTEM: 24K ROM including Atari Basic programming language and a self diagnostic test program.
   KEYBOARD: Full stroke design with 62 keys including help key and 4 special function keys, international character set and 29 graphics keys.
   SOUND: 4 independent sound synthesisers each capable of producing music across a 3½ octave range or a wide variety of special sound effects. (Additional programming can achieve an octave range of up to nine octaves!)
   DISPLAY: 11 graphic modes and 5 text modes. Up to 320×192 resolution. Maximum text display 24 lines by 40 columns.
   SPECIAL ATARI INTEGRATED CIRCUITS: GTIA for graphics display, Pokey for sound and controller ports, Antic for screen control and I/O (Input/Output).
   CPU: 6502C microprocessor - 0.56 microsecond cycle and a clock speed of 1.79 MHz.
   EXTENDED GRAPHICS FUNCTIONS: High resolution graphics. Multi-coloured character set. Software screen switching. Multiple redifined character sets. Player missile (sprite) graphics. Fine screen scrolling. Changeable colour registers. Smooth character movement. Simple colour animation facilities.
   PROGRAMMING FEATURES: Built in Atari Basic programming language supporting peek, poke and USR plus at least 8 other languages available. The help key will provide additional information and menu screens with certain software. Full on-screen editing is available as well as syntax checking on entry.
   INPUT/OUTPUT: External processor bus for expansion with memory and peripherals. Composite video monitor output. Peripheral port for direct connection to Atari standard peripherals. Software cartridge slot is included as well as 2 joystick controller ports.
   SOFTWARE: Over 1,500 items of software are available including self teaching programs with unique voice over. The range of programs includes Education, Home Management & Programming aids. There is also APX (Atari Program Exchange) and of course Atari's famous entertainment software now at only £9.95. In addition there is a host of support and help available from specialist Atari magazines like Antic and Analog and from over 75 Atari books/manuals.
2. **ATARI 400 16K GAMES MACHINE - £29:** We have several Atari 400 games consoles/computers with 16K RAM. The price is £29 (for a reconditioned model) or £39 for a new machine. Both come with 12 months guarantee. The Atari 400 can play all 800XL ROM cartridge games and is expandable up to 48K RAM. Computer upgrade with Basic Programming Kit (£30) optional extra.
3. **ATARI 1010 PROGRAM RECORDER - £34:** For low cost storage and retrieval capability. Data transmission 600 baud. Storage capability 100K bytes on a sixty minute cassette. Track configuration four track, two channels (digital and audio). Auto record/playback/pause control/unique soundthrough facility. Built in accidental erasure prevention, automatic shutoff and 3 digit tape counter.
4. **ATARI 1050 DUAL DENSITY DISK DRIVE - £199:** 5¼" disks holding 127K randomly accessible bytes provide both expansion and flexibility for your 400/800 or XL system with new 'helpful' DOS 3. All customers who purchase a Disk Drive from Silica Shop will be automatically given a FREE set of 100 programs on 3 Disks recorded on both sides.
5. **ATARI 1020 COLOUR PRINTER - £99:** Printer and Plotter with four colour graphic print capability. 40 column width printing at 10 characters per second. Can print 5, 10 and 20 characters per inch. 64 character sizes. Prints text in 4 directions. Choice of line types.
6. **ATARI 1027 LETTER QUALITY PRINTER - £249:** For word processing letters in professional type. Print speed of 20 chars per second.
7. **ATARI TOUCH TABLET - £49:** Enables you to draw and paint pictures on your T.V. screen, with the touch of a stylus.
8. **ATARI TRAK BALL CONTROLLER - £19.95:** Enables cursor movement in any direction and adds arcade realism to your games.
9. **ATARI SUPER CONTROLLER - £9.95:** The ultimate joystick with double fire button to give you a greater competitive edge in your games.

SILICA SHOP ARE THE No1 ATARI SPECIALIST

Silica Shop are now firmly established as the No 1 Atari retail/mail order and wholesale specialist in the U.K. We already offer our service to over 120,000 customers, 10,000 of whom have purchased Atari Home Computers. Because we specialise (and with a turnover of £1.5 million), we are able to keep prices low by bulk purchases. Ring one of our 45 staff and we will be glad to be of service to you. Complete the coupon below and we will send you our Atari pack with our 16 page price list and XL colour catalogue.

EXTENDED TWO YEAR GUARANTEE: We are an Atari Service Centre, able to service and repair Atari equipment and have added a 12 month guarantee to the year offered by Atari, giving you a full 2 year guarantee on your new XL computer.

SPECIALIST SUPPORT: Our technical staff are always available on the telephone to help and advise you. We endeavour to hold stocks of every Atari compatible item available in the U.K. and we stock over 75 Atari books and manuals.

AFTER SALES SERVICE: When you purchase your equipment from Silica, your name will be automatically added to our mailing list. You will then receive price lists, newsletters and details of new releases and developments, as well as special offers which are exclusive to Silica Atari Computer Owners.

LOW PRICES: Our prices include VAT and are extremely competitive. We will normally match any lower price offered by our competitors.

FREE COMPUTER OWNERS CLUB: This is open to all Atari computer owners irrespective of where you purchased your equipment. Membership is FREE and entitles you to receive bulletins giving details of new releases and developments. Send now for your FREE information pack, price list & colour catalogue.

PAYMENT: We accept cash, cheques, postal orders and all Credit Cards. We also offer credit facilities over 1, 2 or 3 years, please write for a written quotation.

NEXT DAY DELIVERY - FREE: All goods despatched from Silica Shop are normally sent by first class post or parcel post FREE OF CHARGE. As a special introductory offer for a limited period only we will be sending all Computers and Disk Drives by a next day Securicor delivery service at our own expense.

So fill in the coupon below with a literature enquiry or order and begin to experience a specialist Atari service that is second to none.

SILICA SHOP LTD, 1-4 The Mews, Hatherley Road, Sidcup, Kent, DA14 4DX Tel: 01-309 1111

ORDER NOW-OR SEND FOR A FREE COLOUR BROCHURE

To: SILICA SHOP LTD, Dept ATCOC 0185, 1-4 The Mews, Hatherley Road, Sidcup, Kent, DA14 4DX Telephone: 01-309 1111

LITERATURE REQUEST:

☐ Please send me your FREE colour brochures and 16 page price list on Atari Computers.

☐ I own a ..... Videogame ☐ I own a ..... Computer

Mr/Mrs/Ms: ..... Initials: ..... Surname .....

Address: .....

..... Postcode: .....

ORDER REQUEST:

PLEASE SEND ME:

☐ 800XL 64K Computer	£129	☐ 1020 4 Colour Printer	£99
☐ 400 16K Games Machine	£29/£39	☐ Letter Quality Printer	£249
☐ 1010 Program Recorder	£34	☐ Touch Tablet + Cartridge	£49
☐ 1050 127K Disk Drive	£199	☐ Trak Ball	£19.95
		☐ Super Controller	£9.95

ALL PRICES QUOTED ARE INCLUSIVE OF VAT - POSTAGE & PACKING IS FREE OF CHARGE

☐ I enclose Cheque/P.O. payable to Silica Shop Limited for the following amount £.....

☐ CREDIT CARD - Please debit my:
Access/Barclaycard/Visa/American Express/Diners Club Card Number .....