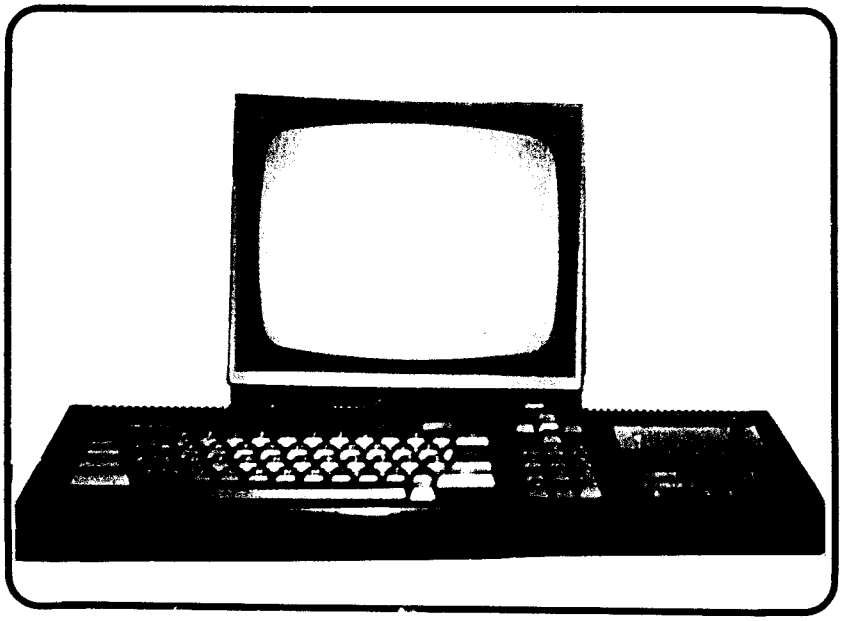
PRINT-OUT

Price 70p

ISSUE FOUR



BY Thomas Defoe, Mark Gearing and Jonathan Haddock

Contributor :- Bob Taylor

Including:-

STEP RSX - DEBUGGING AID

SEABATTLE-PROGRAM

BASIC and M/C tutorials

Homebrew Software

& many other articles

INDEX

Miscellaneous

Features

Reviews

Programming

We would like to express our thanks to Mr Gearing and Black Horse Agencies, Januarys, for the continued use of their photocopier in the production of this issue of Print-Out.

Editorial

WELCOME TO ISSUE FOUR OF PRINT-OUT !!
We hope that you enjoy reading it !!!

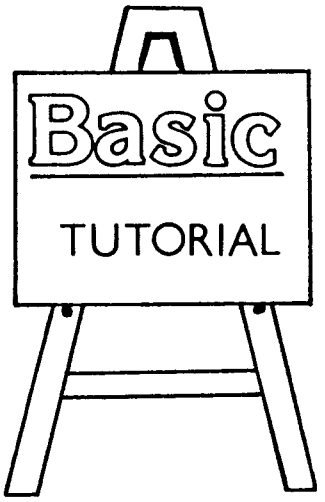
If this is your first time for buying Print-Out we hope you enjoy it enough to buy back issues and if you are now a 'regular' we hope that you continue to buy the magazine which 'caters for all serious users of the Amstrad CPC' If everything goes according to plan, (we managed it this time !!!) we hope to have completed Issue Five by about May 30th. If you wish to order a copy of Issue Five in advance, please send one of the following :-

a) 70p + A4 SAE (with a 28p stamp)
b) £1.10 (which includes p+p)

Of course you can also order previous issues, further future issues or even have a subscription (see 'Offers' for more details).

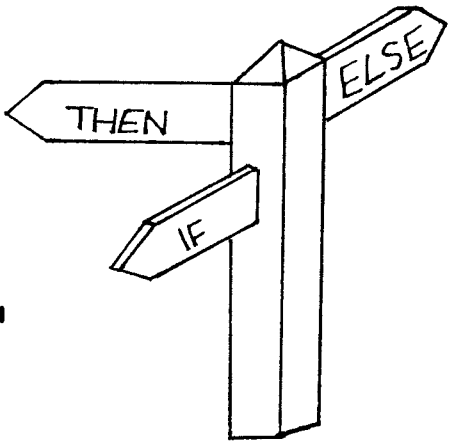
We would like to thank Bob Taylor for contributing to Print-Out and also to the other people who have helped with the production of this issue. If anybody would like to contribute pieces of writing, programs or even artwork we'd be most interested and grateful. It can be on virtually any CPC related matter so we look forward to hearing from you ! (see address below) Of course, if you've any questions or problems concerning the CPC please do not hesitate to write to us and we'll do our best to try and solve it.

The address to write to is :-
PRINT-OUT, 8 Maze Green Road,
Bishop's Stortford,
Herts CM23 2PJ.



BEGINNER'S BASIC

DECISION MAKING



We have now looked at printing information, inputting instructions or information from the user, using this 'data' in our own calculations and also, most recently, repeating instructions. However, there is one very important concept that we have not covered and that is decision making. Very few programs except the simplest of them, will not contain some example of the computer having to make some decisions. If we take a small program, such as a game of noughts and crosses between two human players, it is obvious that numerous decisions will have to be made. Not only will the computer need to determine where a O or a X must go on the grid and even whether it can be placed in that position, but it will also have to judge who has won and lost.

The most common decision making instruction is known as the IF...THEN...ELSE command. This allows the computer to compare one piece of information with another and IF some condition(s) are met, THEN the program will do one thing, but IF the conditions are not met, it will do something ELSE. Before we look at the command in great detail, type in and RUN the program shown below that will give you some idea of what the command can do.

```
10 INPUT "What is your name ";name$
20 a$=UPPER$(name$)
30 IF a$="ARNOLD" THEN PRINT "That's my name!" ELSE PRINT "Hello ";name$
40 END
```

We've already discussed line 10 in previous issues. As a reminder, it gets you to type in a word (ie your name) which is stored in the variable 'name$'. Line 20 converts whatever was in name$ into capital letters and then stores that in a$. So suppose you enter 'Bill' as your name in line 10. After the execution of line 20, name$ would still hold Bill, but a$ would hold BILL. Line 30, however, is the important line for us and it requires some careful investigation.

What this line does is to check and see if the contents of a$ match the word ARNOLD, if they do it decides to print "That's my name!" and then ignores everything upto the end of the line. If the contents of a$ don't match ARNOLD then the computer chooses to print whatever follows the ELSE statement ("Hello whatever you entered"). Of course you can do other things as well as printing in an IF...THEN...ELSE command. Look at the example, on the next page, which uses the commands GOTO and FOR...NEXT which we looked at last time.

The program is :-

```
10 INPUT "Do you want me to clear the screen (yes/no) ";a$
20 b$=UPPER$(a$)
30 IF b$="YES" THEN CLS ELSE GOTO 50
40 END
50 FOR i=1 TO 1000
60 PRINT "*";
70 NEXT i
80 END
```

This program asks you to reply either 'yes' or 'no' to its initial question. Line 30 then checks to see what you typed and if you did type 'yes' the program clears the screen (CLS which follows the THEN), ignores the rest of line 30 and executes line 40 which ends the program. However if you typed 'no' (or anything other than yes), it executes the command after the ELSE instruction which sends it to line 50. Lines 50 - 80 simply fill the screen with stars and then end the program. For a full explanation of these lines see Issue 3 - Beginner's Basic.

IF...THEN...ELSE instructions are very flexible and we can use other types of variables in the command. For example we can say whether a variable is greater than or smaller than a particular number or not, whether a variable is equal to a certain number or not, or even whether a variable multiplied by PI is larger than the logarithm of another variable or not !! We can also have more than one condition which needs to be met before a certain action can take place. Look at this program :-

```
10 INPUT "Enter a number between 1 and 10 ",num
20 IF num>10 OR num<1 THEN PRINT "Your number's not between 1 & 10":GOTO 10
30 PRINT "Thank you for your number"
40 END
```

This short program illustrates several new points. Line 10 is simple enough and just asks you to enter a number which is stored in the variable 'num'. Line 20 checks to see if 'num' is greater than 10 or smaller than 1. If either of these conditions is met then the message will be printed and the program will go back to line 10. Notice how the ELSE part of the statement has been ommitted & this is because it is an optional part - the THEN part must always be present.

The listing shown below just asks you for a number and then decides whether it is equal to one or not. It then tells you its incredible conclusion and asks for another number.

```
10 INPUT num
20 IF num=1 THEN PRINT "It's one":GOTO 10:ELSE PRINT "It isn't one":GOTO 10
30 END
```

In line 20, if the number that is stored in 'num' is 1, then everything upto the word ELSE will be executed. If the number in 'num' is not 1 then everything from the ELSE to the end of the line will be executed. In either case line 30 will be executed. If we wished, we could even have the 'ELSE' part of the statement on a separate line except that it would not need the ELSE in front of it. Although it makes no difference in this example, this may not be possible with some programs.

```
10 INPUT num
20 IF num=1 THEN PRINT "It is one":GOTO 10
30 PRINT "It isn't one":GOTO 10
40 END
```

It works for this example because line 30 would be executed only when 'num' did not equal one. This is the one of the reasons why ELSE can be omitted from this type of statement. At some time you may wish to test to see if something is different (or not equal) from something else. The way to do this is to use <>. Look at the example which is a revised version of an earlier listing :-

```
10 INPUT "What is your name ";name$
20 a$=UPPER$(name$)
30 IF a$<>"ARNOLD" THEN PRINT "Your name is different from mine!"
40 END
```

Now the message is printed only if the name that you entered (stored in 'name$' and then in 'a$') is different from 'ARNOLD'.

That concludes this issue's section on 'DECISION MAKING' in Beginner's BASIC but from now on, more and more of your programs in BASIC, especially as they become longer and of greater complexity, will involve IF...THEN...ELSE commands as this is the most common of the decision making commands. At the bottom of the page is a list of all the commands that we have looked at so far for your reference, and following them is the issue in which they were mentioned.

COMMANDS LOOKED AT SO FAR :-

AUTO	-	ISSUE 2	GOTO	-	ISSUE 3	RENUM	-	ISSUE 2
CLS	-	ISSUE 1	LIST	-	ISSUE 1	RUN	-	ISSUE 1
EDIT	-	ISSUE 1	LOWER$	-	ISSUE 2	THEN	-	ISSUE 4
ELSE	-	ISSUE 4	NEW	-	ISSUE 2	TROFF	-	ISSUE 3
FOR	-	ISSUE 3	NEXT	-	ISSUE 3	TRON	-	ISSUE 3
IF	-	ISSUE 4	PRINT	-	ISSUE 1	UPPER$	-	ISSUE 2
INPUT	-	ISSUE 2						

MACHINE CODE

the final installment in 'PRINTING'

PART 4

In Issue Three. we wrote a short program that accepted two numbers and then calculated and printed their total. However this routine was able to deal only with sums that had an answer that fell between 0 and 9. In this issue. we will try to write a program that will print much larger numbers upto 65535 (the reason for this number is explained later). In the last part of this tutorial we touched on 'conditional' instructions and these play a major part in all machine code programs. The routines in this tutorial include many examples of conditional instructions and in order to understand how they work it will be beneficial if you have read the section on 'FLAGS AND CONDITIONS' first.

Those of you who have looked at other machine code tutorials before may have realised that we have used only six calls to the firmware (see Issue One for an explanation) so far. Normally, by the fourth part of a series on machine code. you would have been introduced to about 25 different firmware calls! The reason for this is that when you understand the basics of machine code, you can apply your knowledge to all areas of programming. However, if you feel that you wish to explore machine code further by yourself, the best way of doing this is to disassemble some of the machine code listings in this magazine (eg. ISTEP) and try to work out what is happening in them. If you do not have a disassembler, we supply copies of all the programs in the magazine. complete with comments on what the program is doing, at the cost of £1, including postage and packing.

As we have already mentioned, in this article we are going to write a short routine that will print any number from 0 to 65535 (giving 65536 possiblilities in total) on the screen. The reason why only 65536 numbers can be printed using this method is relatively simple to explain. In the computer's memory, hundreds of numbers are stored and each of these numbers is called a byte and these can hold a number from between 0 and 255, this is the largest number which can be represented by an 8 digit binary number (see Issue One - 'What is my Amstrad'). Therefore one byte can store 256 possible numbers and when two bytes are used together we can store 256 x 256 (=65536) different numbers. Often two bytes are utilised in this way either in memory or stored in a REGISTER PAIR.

BASIC POKER

Next to all the routines contained in this issue (in the leftmost column) there are rows of numbers. These numbers should not be typed into an ASSEMBLER, but they are intended for use by those who do not possess one. These numbers can be typed into the 'BASIC Poker' which was printed in both Issues One and Two. This program is included on every issue's program cassette and includes instructions for its correct use.

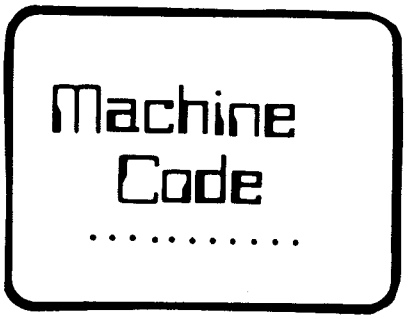
The listing itself contains many fairly self explanatory comments and extra notes on some of the trickier parts are printed after the program. However, the basic idea is that the program is subtracting multiples of 10 (in a decreasing order) from the number that is stored in HL (the number to be printed). Every time this happens it increases the A register by one and this acts as a counter. It then repeats this subtraction until HL holds a negative number and when this occurs the carry flag is set. The number in HL then has the multiple of 10 ADDED to it in order to make it a positive number for future calculations. The number in A (a record of the number of times that the multiple of 10 was subtracted) is converted into its ASCII code by adding &30 to it and printed. This sequence is now repeated with smaller and smaller multiples of 10, until only the units are left and these are then printed without the need for an adjustment.

```
                ORG &4000
21 39 30        LD HL,12345     ; the number to print is stored in HL
01 10 27        LD BC,10000     ; BC=10000
CD 22 40        CALL prt_dig    ; calculate and print the 10000's digit
01 E8 03        LD BC,1000      ; BC=1000
CD 22 40        CALL prt_dig    ; calculate and print the 1000's digit
01 64 00        LD BC,100       ; BC=100
CD 22 40        CALL prt_dig    ; calculate and print the 100's digit
01 0A 00        LD BC,10        ; BC=10
CD 22 40        CALL prt_dig    ; calculate and print the 10's digit
7D              LD A,L          ; the value left is less than 10
C6 30           ADD A,&30       ; add &30 to convert this digit into one that
                                ; can be printed (see note above)
CD 5A BB        CALL &BB5A      ; print the units digit
C9              RET             ; return

                .prt_dig        ; the calculating routine
3E FF           LD A,&FF        ; the digit counter =-1 (see below)
                .loop
3C              INC A           ; increase the counter by 1
ED 42           SBC HL,BC       ; subtract the number that is stored in BC
                                ; (ie 10000,1000,100,10) from the number to
                                ; be printed that is in HL
30 FB           JR NC,loop      ; if HL becomes negative the carry will be set
                                ; otherwise jump back to the label .loop
09              ADD HL,BC       ; add the value in BC to HL, so that HL will
                                ; contain a positive number for future use
C6 30           ADD A,&30       ; convert the value in A to a printable digit
CD 5A BB        CALL &BB5A      ; and then print the digit
C9              RET             ; return from subroutine
```

When we CALL &4000 we get 12345 printed on the screen as we would expect. Some of the commands used above are new and are explained on the next page.

Before we look at the new commands, it is worthwhile to study exactly what the program is doing during its execution. Below is shown a list of what really happens when the routine is called.



1. HL is loaded with 12345
2. BC is loaded with 10000
3. The routine 'prt_dig' is called
4. In this routine the following occurs :-
   a) A is loaded with &FF.
   b) A is increased by one.
   c) BC is subtracted from HL and the carry flag is set if BC becomes negative.
   d) As long as the carry flag is NOT set, jump to instruction 4b otherwise go to instruction 4e.
   e) BC will now contain a negative number (eg. -7655), so add BC to HL so that a positive number (eg. 2345) will be produced.
   f) A will now contain the number of times BC was subtracted from HL without a negative number being produced (eg. 1). Add &30 to this number in order to convert it into its ASCII equivalent.
   g) Print this number and then return to wherever this routine was called.
5. Now load BC with 1000 and go to instruction 4.
6. Then load BC with 100 and go to instruction 4.
7. BC is loaded with 10 and the routine goes to instruction 4.
8. The remainder will be less than 10, so add &30 (to convert it to its ASCII equivalent) to it and then print the final digit

At one point, the notes say that A is loaded with -1, but the instruction says LD A,&FF, which may be a bit confusing. The reason for this is that &FF is the largest number that can be stored in a single register. When A is increased by one there is no room for the overflow and so A contains 0 and the overflow is discarded. If you now go back the other way, if you subtract 1 from 0, you get -1 and so it would appear that &FF equals -1. This may sound confusing to you, but if you think about it IS logical.

In the routine there are several new commands. The first is SBC which is similar to SUB except that the carry is then subtracted from the number. This will make some sense to you if you have looked at FLAGS. Likewise JR is another form of a jump and stands for JUMP RELATIVE and is used to jump small distances, but at the moment you can treat it in the same way as you would a normal jump. The NC,loop following it makes this a conditional jump (in this particular case, it will jump only to .loop if there is No Carry) and this is explained in the part on FLAGS in this issue.

That concludes this issue's section on Machine Code and in our next issue we will look further at the flags and will start writing some useful routines. In Issue Five we hope to start of section on advanced M/C which will include many unusual and complicated programming techniques.

HOMEBREW COMPANY PROFILE

This article was written by Tony Kingsmill who runs a small company that produces adventure games. Two of his programs are reviewed elsewhere in the magazine, but we felt it would be interesting for others to see why he started a homebrew company & how. It seems that homebrew software is an important, and often neglected, part of the computer market & is thriving especially on the CPC. Although the software industry for the CPC is strong, at the moment, it will eventually decrease. It is then that users will turn to the homebrew sector to supply their games and utilities. By printing this article, we hope that some potential software writer may be stirred into producing their own games. For have no doubt, that the day of the homebrew industry will come!!

I am 16 years of age and still at school at Verulam, St.Albans. My first taste of computers was with the Atari console, way back around 1980, but didn't get a proper computer until about 1983. We bought an Acorn Electron, on which I mainly played games at first. I was about eleven when I first became interested in programming. I didn't really have much idea where to start and so I bought a book with games to type in. I found this enjoyable and quickly became familiar with many commands. From there I began to write my own programs in BASIC, varying in quality but none were good enough to sell.

Considering how old the CPC is now, I suppose I bought mine fairly late, around February 1988 after I felt my Electron had seen better days! I chose the CPC 6128 mainly because it is so versatile - it can be used for just about anything. 16-Bit software is far more expensive and apart from a few extras it all does the same thing.

I started producing adventure games towards the end of last summer, my first release being 'Lords of Magic', which is available now. I did not finish the game properly until December, constantly adding to the game until I ran out of memory. Lords of Magic involves you being captured in a strange land by the Lords of Magic, & you must find a way to escape. My second & most recent release is the Island of Chaos based on the island of Brael Ti. The game is about an evil ruler named Baktron who takes over the island, & it is your mission to defeat him. Both games are £3.95 each which I feel is very reasonably priced compared to what most professional companies charge and when you consider the price of the blank disks, you're left with very little profit.

For a full review of both 'Lords of Magic' and also 'Island of Chaos' look at the article on 'HOMEBREW SOFTWARE' - in this issue we also feature a selection of educational programs from Matthew Pinder.

Lords of Magic and Island of Chaos were both programmed with Gilsoft's Quill Adventure Writing System. The system gives you (excluding graphics) about 30k to use. I found this was enough for Island of Chaos but I almost ran out of bytes with Lords of Magic. Considering it's age, the Quill is very good, allowing games to run reasonably fast, and of course, merge graphics designed on the Illustrator, Gilsoft's graphic designer for the Quill. The major set back with the Quill is probably the parser - limited to understanding two words per sentence. Nevertheless, I manage to get round this by using lots of small commands rather than one larger one. For instance, instead of 'GET CANDLE AND LIGHT IT', I would have to use 'GET CANDLE' then 'LIGHT CANDLE' as two separate sentences.

Looking ahead, I will probably update to a system like the Professional Adventure Writer (PAW), Gilsoft's more powerful system. This will enable me to produce games with a better parser. At the moment I'm just producing adventures and cannot see any reason to update to a more powerful computer or change the style of games.

There is a lot of life in the CPC's yet, especially with the new models arriving later in the year. With most major companies like Level 9 switching to more arcade style 16 Bit games, more people will turn to homebrew companies for new adventures, and most of them are cheaper too.

As for the near future, I hope to continue releasing adventures throughout the year. My next game will probably have less of a magical fantasy element, possibly even futuristic. I also have plans for a sequel to Island of Chaos, but that will depend on how successful Island of Chaos is.

If you run a small company that produces anything (games, hardware, programs, etc.) for the Amstrad range then we would be delighted to hear from you. Not only would we give your product a full review in Print-Out, but we would also like to write a company profile on you because we believe that Homebrew programs & small companies will be the future of the CPC. If you are interested in having your product reviewed or in writing an article on your company then please write to us at the address which is printed at the front of the magazine.

FALLING LETTERS

BITS AND PIECES

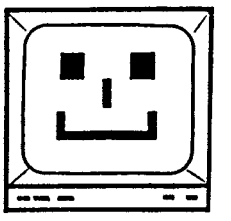
Listed on this page is a program that provides you with a rather unusual means of displaying any message of upto 80 characters. The actual printing section of the program is written in M/Code but a BASIC 'driver' pokes the message in to the relevant places in the computer's memory. Therefore, by altering lines 245 onwards you can change the pattern in which the letters fall on the screen. If you wish the letters to fall more slowly, just change line 60 to read :-

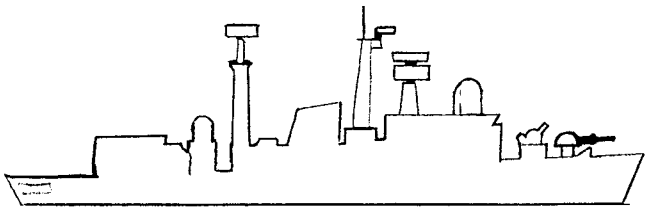
DATA CD,19,BD,CD,19,BD,00,00

```
[D8]  10 DATA 3E,02,CD,0E,BC,21,5E,40
[A9]  20 DATA 4E,79,FE,00,C8,23,56,E5
[39]  30 DATA 06,18,3E,19,90,CD,72,BB
[02]  40 DATA 7A,CD,6F,BB,3E,01,CD,90
[F8]  50 DATA BB,79,CD,5A,BB,00,00,00
[DA]  60 DATA 00,00,00,00,00,00,00,00
[F1]  70 DATA 00,3E,19,90,CD,72,BB,7A
[A1]  80 DATA CD,6F,BB,3E,00,CD,90,BB
[3D]  90 DATA 79,CD,5A,BB,10,CC,3E,19
[4E] 100 DATA CD,72,BB,7A,CD,6F,BB,3E
[50] 110 DATA 01,CD,90,BB,79,CD,5A,BB
[38] 120 DATA E1,23,C3,08,40,C9,00,00
[A7] 130 '
[33] 140 DATA end
[6A] 150 add=&4000
[03] 160 READ a$:IF a$="end" THEN GOTO 190
[5A] 170 POKE add,VAL("&"+a$):add=add+1
[2E] 180 GOTO 160
[5F] 190 add=&405E
[D2] 200 PRINT "Please enter the string you wish to be printed of no more than
         80 letters :-"
[71] 210 LINE INPUT a$
[07] 220 IF LEN(a$)>80 THEN GOTO 200
[3F] 230 IF LEN(a$)<80 THEN a$=a$+" ":GOTO 230
[2A] 240 DIM d(80,2)
[E3] 245 FOR i=1 TO 80
[63] 250 b$=LEFT$(a$,1)
[23] 260 c=ASC(b$):d(i,1)=c
[5A] 270 d(i,2)=i
[2F] 280 a$=RIGHT$(a$,80-i)
[BA] 290 NEXT i
[5B] 295 FOR j=0 TO 4
[CD] 300 FOR i=1+j TO 40+j STEP 5
[54] 310 POKE add,d(80-i,1):add=add+1:POKE add,d(80-i,2):add=add+1
[89] 320 POKE add,d(i,1):add=add+1:POKE add,d(i,2):add=add+1
[BC] 325 NEXT i
[C9] 326 NEXT j
[2A] 330 POKE add,0
[3C] 340 CALL &4000
```

Linechecker

A PROGRAM TYPING AID
for use with PRINT-OUT

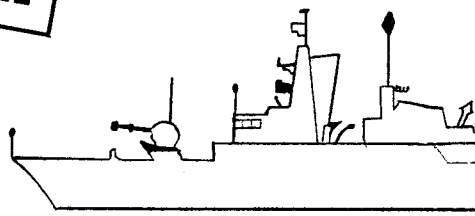




SEABATTLE

by

S. MESSINA



This program was written by S. Messina from Heywood in Lancashire and was his entry for the Maxam competition that we ran in Issue Two of our magazine. This program earned him first place and a copy of Maxam from Arnor on tape.

The program is like a computerised 'Battleships' for your CPC. However, it also features some excellent graphics and sound effects. When you have selected the position you wish to bomb (in a 5 x 5 grid), you can see the missiles being launched and if they hit the target, the ship explodes and sinks into the sea. However, although the program is very good, it is quite long and you may feel that it would be advantageous to buy the program cassette/disc which contains this and all the other programs in the magazine.

When you have finished typing in the program and you run it, you are asked for both players' names. Player Two is then asked to leave the room while Player One positions his ships in the grid. Each position is identified by a two character coordinate which should be entered in lower-case in the form 'letter then number' (eg. a1 or d4). When Player One has positioned five targets, he is asked to leave the room whilst Player Two enters his coordinates. When this is done, the players take it in turns to try and locate the others ships by entering coordinates in the same manner as before. A grid is shown to remind players where they have already tried - a dot means a hit & a cross means a miss. The game ends when one of the players has hit all five of the opposition's ships & you are then given the opportunity to play again.

To aid typing in, we have printed Linechecker numbers (see Issue Three for a full explanation) and we wish you good luck in the game.

PROGRAM

```
[CB] 10 SYMBOL AFTER 256:MEMORY 40959:SYMBOL AFTER 236:add=40960:PEN 10:GOSUB 1830:GOSUB
     1810
[D3] 20 WINDOW 3,18,8,25
[4A] 30 ORIGIN 0,0,0,640,400,0:CLG 0:LOCATE 4,4:PRINT"PLAYER(1)":LOCATE 5,7:INPUT " ",f$:
     f$=UPPER$(f$):CLS
[64] 40 LOCATE 4,4:PRINT"PLAYER(2)":LOCATE 5,7:INPUT" ",q$:q$=UPPER$(q$):CLS
[00] 50 LOCATE 7,4:PRINT q$:LOCATE 2,7:PRINT" Please leave":FOR a=1 TO 820:NEXT:CLS
[0D] 60 LOCATE 7,4:PRINT f$:LOCATE 2,7:PRINT"Press any key":k$=INKEY$:IF k$="" THEN 60:
     ELSE CLS#2:FOR a=0 TO 15:INK a,0:NEXT:GOSUB 730:GOSUB 720
[EA] 70 FOR p=&A9EC TO &A9FF:POKE p,&0:NEXT
```

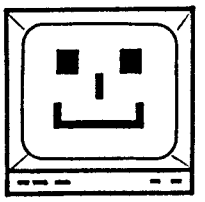
```
[CE] 80 LOCATE 9,4:PRINT f$:LOCATE 4,9:PEN 10:PRINT"Choose 5 target";:WINDOW#4,12,16,19,
     19:PAPER#4,0:PEN#4,10:CLS#4
[B1] 90 INPUT#4," ",c$:GOSUB 1780:IF ic=0 THEN GOTO 90:ELSE GOSUB 870:GOSUB 1190
[C4] 100 INPUT#4," ",c$:GOSUB 290:IF ok=1 THEN GOTO 100:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     100:ELSE GOSUB 880:GOSUB 1190
[B7] 110 INPUT#4," ",c$:GOSUB 290:IF ok=1 THEN GOTO 110:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     110:ELSE GOSUB 890:GOSUB 1190
[0A] 120 INPUT#4," ",c$:GOSUB 290:IF ok=1 THEN GOTO 120:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     120:ELSE GOSUB 900:GOSUB 1190
[FD] 130 INPUT#4," ",c$:GOSUB 290:IF ok=1 THEN GOTO 130:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     130:ELSE GOSUB 910:GOSUB 1190
[D8] 140 CLS:LOCATE 7,12:PRINT f$:LOCATE 4,17:PRINT" Please leave";:FOR a=1 TO 820:NEXT:
     CLS:LOCATE 7,12:PRINT q$
[E2] 150 LOCATE 5,17:PRINT"Press any key":IF k$=""THEN 150:ELSE CLS:FOR a=0 TO 15:INK a,0
     :NEXT:GOSUB 730:GOSUB 720
[F4] 160 LOCATE 8,4:PRINT q$:LOCATE 4,9:PRINT"Choose 5 target";:WINDOW#4,12,16,19,19:
     PAPER #4,0:PEN#4,10:CLS#4
[2D] 170 LOCATE#4,1,1:INPUT#4," ",c$:GOSUB 1780:IF ic=0 THEN SOUND 1,500:GOTO 170:ELSE
     GOSUB 920:GOSUB 1190
[A6] 180 INPUT#4," ",c$:GOSUB 300:IF ok=1 THEN GOTO 180:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     180:ELSE GOSUB 930:GOSUB 1190
[99] 190 INPUT#4," ",c$:GOSUB 300:IF ok=1 THEN GOTO 190:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     190:ELSE GOSUB 940:GOSUB 1190
[9D] 200 INPUT#4," ",c$:GOSUB 300:IF ok=1 THEN GOTO 200:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     200:ELSE GOSUB 950:GOSUB 1190
[90] 210 INPUT#4," ",c$:GOSUB 300:IF ok=1 THEN GOTO 210:ELSE GOSUB 1780:IF ic=0 THEN GOTO
     210:ELSE GOSUB 960:GOSUB 1190
[32] 220 cd=0:sd=0:sm=0:MODE 0:LOCATE 4,15:PRINT"Please Wait..":FOR a=1 TO 1100:NEXT:FOR
     a=0 TO 15:INK a,0:NEXT:CLS:GOSUB 730:|SCR2:CLS:GOSUB 730:GOSUB 790
[03] 230 |SCR1:CLS:PEN 1:GOSUB 730:GOSUB 790
[CF] 240 BORDER 0:GOSUB 720:PRINT z$:LOCATE 12,16:PRINT f$:GOSUB 1510
[D7] 250 CLS#2:PEN#2,10:GOSUB 1460:GOSUB 1490:INPUT#2," ",c$:GOSUB 1500:GOSUB 1780:IF ic=
     0 THEN GOTO 250:ELSE PRINT x$:GOSUB 630:GOSUB 1140:GOSUB 1470
[7F] 260 |SCR2:BORDER 15:PRINT z$;:LOCATE 12,16:PRINT q$:WINDOW#2,13,16,19,19:PAPER#2,3
[90] 270 CLS#2:PEN#2,10:GOSUB 1460:GOSUB 1490:LOCATE#2,1,1:INPUT#2," ",c$:GOSUB 1500:
     GOSUB 1780:IF ic=0 THEN GOTO 260:ELSE PRINT x$;:GOSUB 310:GOSUB 1150:GOSUB 1470
[35] 280 |SCR1:BORDER 0:GOSUB 1510:CLS#2:PEN#2,10:GOSUB 1460:GOSUB 1490:INPUT#2," ",c$:
     GOSUB 1500:GOSUB 1780:IF ic=0 THEN GOTO 280:ELSE PRINT x$;:GOSUB 630:GOSUB 1140:
     GOSUB 1470:GOTO 260
[20] 290 IF VAL("&"+c$)=PEEK(43500) OR VAL("&"+c$)=PEEK(43501)OR VAL("&"+c$)=PEEK(43502)
     OR VAL("&"+c$)=PEEK(43503)OR VAL("&"+c$)=PEEK(43504) THEN ok=1:GOSUB 1480:RETURN:
     ELSE ok=0:RETURN
[20] 300 IF VAL("&"+c$)=PEEK(43505) OR VAL("&"+c$)=PEEK(43506)OR VAL("&"+c$)=PEEK(43507)
     OR VAL("&"+c$)=PEEK(43508)OR VAL("&"+c$)=PEEK(43509) THEN ok=1:GOSUB 1480:RETURN:
     ELSE ok=0:RETURN
[94] 310 IF VAL("&"+c$)=PEEK(43500)AND PEEK(43515)=0 THEN cd=1:pop=0:POKE 43515,1:GOSUB
     980:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43500)AND PEEK(43515)=1 THEN pop=1:
     GOTO 260:ELSE pop=0
[72] 320 IF VAL("&"+c$)=PEEK(43501)AND PEEK(43516)=0 THEN cd=1:pop=0:POKE 43516,1:GOSUB
     1000:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43501)AND PEEK(43516)=1 THEN pop=1:
     GOTO 260:ELSE pop=0
[7E] 330 IF VAL("&"+c$)=PEEK(43502)AND PEEK(43517)=0 THEN cd=1:pop=0:POKE 43517,1:GOSUB
     1010:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43502)AND PEEK(43517)=1 THEN pop=1:
     GOTO 260:ELSE pop=0
[8A] 340 IF VAL("&"+c$)=PEEK(43503)AND PEEK(43518)=0 THEN cd=1:pop=0:POKE 43518,1:GOSUB
     1020:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43503)AND PEEK(43518)=1 THEN pop=1:
     GOTO 260:ELSE pop=0
```

```
[BE] 350 IF VAL("&"+c$)=PEEK(43504)AND PEEK(43519)=0 THEN cd=1:pop=0:poke 45319,1:GOSUB
     1030:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43504)AND PEEK(43519)=1 THEN pop=1:
     GOTO 260:ELSE IF pop=0 THEN GOSUB 370
[C9] 360 RETURN
[AA] 370 IF c$="b5"THEN cd=0:GOSUB 690:RETURN
[6C] 380 IF c$="a1"THEN cd=0:GOSUB 690:RETURN
[7B] 390 IF c$="a2"THEN cd=0:GOSUB 690:RETURN
[77] 400 IF c$="a3"THEN cd=0:GOSUB 690:RETURN
[85] 410 IF c$="a4"THEN cd=0:GOSUB 700:RETURN
[94] 420 IF c$="a5"THEN cd=0:GOSUB 700:RETURN
[6E] 430 IF c$="b1"THEN cd=0:GOSUB 700:RETURN
[7D] 440 IF c$="b2"THEN cd=0:GOSUB 700:RETURN
[8C] 450 IF c$="b3"THEN cd=0:GOSUB 700:RETURN
[BB] 460 IF c$="b4"THEN cd=0:GOSUB 710:RETURN
[A2] 470 IF c$="c1"THEN cd=0:GOSUB 710:RETURN
[B1] 480 IF c$="c2"THEN cd=0:GOSUB 710:RETURN
[C0] 490 IF c$="c3"THEN cd=0:GOSUB 710:RETURN
[BC] 500 IF c$="c4"THEN cd=0:GOSUB 710:RETURN
[AC] 510 IF c$="c5"THEN cd=0:GOSUB 690:RETURN
[86] 520 IF c$="d1"THEN cd=0:GOSUB 690:RETURN
[95] 530 IF c$="d2"THEN cd=0:GOSUB 690:RETURN
[A3] 540 IF c$="d3"THEN cd=0:GOSUB 700:RETURN
[B2] 550 IF c$="d4"THEN cd=0:GOSUB 700:RETURN
[C1] 560 IF c$="d5"THEN cd=0:GOSUB 700:RETURN
[9B] 570 IF c$="e1"THEN cd=0:GOSUB 700:RETURN
[CA] 580 IF c$="e2"THEN cd=0:GOSUB 710:RETURN
[D9] 590 IF c$="e3"THEN cd=0:GOSUB 710:RETURN
[B5] 600 IF c$="e4"THEN cd=0:GOSUB 700:RETURN
[C5] 610 IF c$="e5"THEN cd=0:GOSUB 690:RETURN
[C4] 620 RETURN
[19] 630 IF VAL("&"+c$)=PEEK(43505)AND PEEK(43510)=0 THEN cd=1:pop=0:POKE 43510,1:GOSUB
     1030:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43505)AND PEEK(43510)=1 THEN pop=1:
     GOTO 280:ELSE pop=0
[9D] 640 IF VAL("&"+c$)=PEEK(43506)AND PEEK(43511)=0 THEN cd=1:pop=0:POKE 43511,1:GOSUB
     980:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43506)AND PEEK(43511)=1 THEN pop=1:
     GOTO 280:ELSE pop=0
[D9] 650 IF VAL("&"+c$)=PEEK(43507)AND PEEK(43512)=0 THEN cd=1:pop=0:POKE 43512,1:GOSUB
     1010:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43507)AND PEEK(43512)=1 THEN pop=1:
     GOTO 280:ELSE pop=0
[39] 660 IF VAL("&"+c$)=PEEK(43508)AND PEEK(43513)=0 THEN cd=1:pop=0:POKE 43513,1:GOSUB
     1000:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43508)AND PEEK(43513)=1 THEN pop=1:
     GOTO 280:ELSE pop=0
[C3] 670 IF VAL("&"+c$)=PEEK(43509)AND PEEK(43514)=0 THEN cd=1:pop=0:POKE 43514,1:GOSUB
     1020:GOSUB 1190:RETURN:ELSE IF VAL("&"+c$)=PEEK(43509)AND PEEK(43514)=1 THEN pop=1:
     GOTO 280:ELSE IF pop=0 THEN GOSUB 370
[D0] 680 RETURN
[BE] 690 GOSUB 1060:GOSUB 1530:RETURN
[BA] 700 GOSUB 1070:GOSUB 1530:RETURN
[C9] 710 GOSUB 1080:GOSUB 1530:RETURN
[C3] 720 INK 0,2:INK 1,13:INK 2,10:INK 3,3:INK 4,11:INK 5,24:INK 6,19:INK 7,23,6:INK 8,27
     :INK 9,5:INK 10,26:INK 11,21:INK 12,23:INK 13,10:INK 14,0:INK 15,26,9:RETURN
[A0] 730 WINDOW#4,1,20,12,25:PAPER#4,0:CLS#4:WINDOW 1,20,1,25:WINDOW#2,1,20,12,25:PAPER#2
     ,3:CLS#2
[22] 740 PLOT 58,211,10:MOVE 58,210:TAG:PRINT"1";:MOVER 9,0:PRINT"2";:MOVER 7,0:PRINT"3";
     :MOVER 8,0::PRINT"4";:MOVER 9,0:PRINT"5";
[5D] 750 MOVE 19,185:PRINT"A";:MOVER -33,-35:PRINT"B";:MOVER -33,-35:PRINT"C";:MOVER -33,
     -35:PRINT"D";:MOVER -33,-35:PRINT"E";:TAGOFF
[BF] 760 MOVE 315,90:DRAWR 205,0:DRAWR 0,35:DRAWR -205,0:DRAWR 0,-35:FOR y=20 TO 192 STEP
```

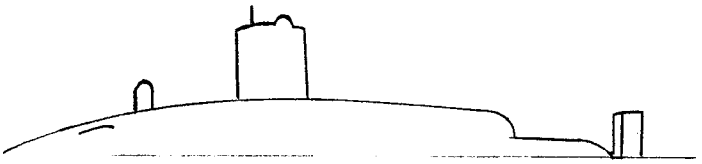
In Issue Three of Print-Out, we included a listing called 'Line-Checker' that produced codes for every line of a program in order to help eliminate typing errors.

The numbers that are printed in square brackets at the beginning of each line are the codes. They should not be typed in & all the programs will run correctly even if 'Linechecker' is not used.

For full instructions on how to use it, please refer to the original article in Issue Three.



```
     1:MOVE 50,y:DRAW 245,y,14:NEXT
[61] 770 FOR y=20 TO 200 STEP 35:MOVE 50,y:DRAW 245,y,10:NEXT
[44] 780 FOR x=50 TO 250 STEP 40:y=195:MOVE x,20:DRAW x,y,10:NEXT:RETURN
[13] 790 PEN 4:WINDOW#1,1,20,1,3:PAPER#1,4:CLS#1
[FB] 800 PLOT -10,0,14:INK 1,26:TAG:MOVE 255,238:PRINT g$;:MOVE 255,250:PRINT g$;:TAGOFF:
     PEN 10:LOCATE 3,4:PRINT h$;:PEN 4:LOCATE 2,5:PRINT v$;:PLOT 32,333,3:DRAWR 158,0
[DF] 810 PEN 10:LOCATE 9,4:PRINT h$;:PEN 4:LOCATE 8,5:PRINT v$;:PLOT 227,332,3:DRAWR 148,
     0:PEN 10:LOCATE 16,4:PRINT h$;:PEN 4:LOCATE 15,5:PRINT v$;:PLOT 452,330,3:DRAWR 148,0
[4D] 820 PEN 10:LOCATE 5,6:PRINT h$;:PEN 4:LOCATE 4,7:PRINT v$;:PLOT 96,300,3:DRAWR 154,0
[1D] 830 PEN 10:LOCATE 15,7:PRINT h$;:LOCATE 14,8:PEN 4:PRINT v$;:PLOT 416,284,3:DRAWR
     155,0:FOR A=1 TO 15:PLOT 447,370:DRAWR -25*SIN(A)-20,-17,14:NEXT:PLOT 400,440
[CF] 840 FOR A=1 TO 10:PLOT 397,370:DRAWR 25*SIN(A)-20,-17,14:NEXT:PLOT 400,440
[21] 850 FOR A=1 TO 15:PLOT 47,370:DRAWR 25*SIN(A)-20,-17,14:NEXT:PLOT 400,440,14
[CE] 860 RETURN
[6C] 870 POKE 43500,VAL("&"+c$):RETURN
[7C] 880 POKE 43501,VAL("&"+c$):RETURN
[8C] 890 POKE 43502,VAL("&"+c$):RETURN
[89] 900 POKE 43503,VAL("&"+c$):RETURN
[99] 910 POKE 43504,VAL("&"+c$):RETURN
[A9] 920 POKE 43505,VAL("&"+c$):RETURN
[B9] 930 POKE 43506,VAL("&"+c$):RETURN
[C9] 940 POKE 43507,VAL("&"+c$):RETURN
[D9] 950 POKE 43508,VAL("&"+c$):RETURN
[E9] 960 POKE 43509,VAL("&"+c$):RETURN
```



SEABATTLE

```
[4F] 970 ENV 1,15,-1,20:OUT &BC00,8:OUT &BD00,1:SPEED INK 1,1:BORDER 0,26:SOUND 1,0,150,
     15,1,,31:WHILE (SQ(1) AND &80)<>0:WEND:OUT &BC00,8:OUT &BD00,0:BORDER 0:RETURN
[6F] 980 GOSUB 1050:INK 0,15:FOR a=1 TO 100:NEXT:INK 0,2:SPEED INK 1,5:LOCATE 3,4:PEN 7:
     PRINT h$;:FOR a=1 TO 250:NEXT:LOCATE 3,4:PEN 14:PRINT h$;:GOSUB 1040:FOR s=1 TO 16
[3A] 990 !SINK,2,6,4,5:FOR i=1 TO 250:NEXT:NEXT:RETURN
[27] 1000 GOSUB 1050:INK 0,15:FOR a=1 TO 100:NEXT:INK 0,2:SPEED INK 1,5:LOCATE 9,4:PEN 7:
     PRINT h$;:FOR a=1 TO 250:NEXT:LOCATE 9,4:PEN 14:PRINT h$;:GOSUB 1040:PEN 1:FOR s=1
     TO 16:!SINK,8,13,4,5:FOR i=1 TO 250:NEXT:NEXT:RETURN
[31] 1010 GOSUB 1050:INK 0,15:FOR a=1 TO 100:NEXT:INK 0,2:SPEED INK 1,5:LOCATE 16,4:PEN 7
     :PRINT h$;:FOR a=1 TO 150:NEXT:LOCATE 16,4:PEN 14:PRINT h$;:GOSUB 1040:PEN 1:FOR s=1
     TO 16:!SINK,15,19,4,5:FOR i=1 TO 250:NEXT:NEXT:RETURN
[65] 1020 GOSUB 1050:INK 0,15:FOR a=1 TO 100:NEXT:INK 0,2:SPEED INK 1,5:LOCATE 5,6:PEN 7:
     PRINT h$;:FOR a=1 TO 250:NEXT:LOCATE 5,6:PEN 14:PRINT h$;:GOSUB 1040:PEN 1:FOR s=1
     TO 16:!SINK,4,8,6,7:FOR i=1 TO 250:NEXT:NEXT:RETURN
[F5] 1030 GOSUB 1050:INK 0,15:FOR a=1 TO 100:NEXT:INK 0,2:SPEED INK 1,5:LOCATE 15,7:PEN 7
     :PRINT h$;:FOR a=1 TO 250:NEXT:LOCATE 15,7:PEN 14:PRINT h$;:GOSUB 1040:PEN 1:FOR s=1
     TO 16:!SINK,14,19,7,8:FOR i=1 TO 250:NEXT:NEXT:RETURN
[F7] 1040 ENV 1,15,-1,20:OUT &BC00,8:OUT &BD00,1:SPEED INK 1,4:BORDER 0,26:SOUND 1,0,90,
     15,1,,31:WHILE (SQ(1) AND &80)<>0:WEND:OUT &BC00,8:OUT &BD00,0:BORDER 0:RETURN
[59] 1050 GOSUB 1090:GOSUB 1100:RETURN
[1D] 1060 GOSUB 1090:GOSUB 1100:GOSUB 1110:RETURN
[44] 1070 GOSUB 1090:GOSUB 1100:GOSUB 1120:RETURN
[6B] 1080 GOSUB 1090:GOSUB 1100:GOSUB 1130:RETURN
[15] 1090 SOUND 1,0,15,15,1,,31:TAG:PLOT 269,270,7:SPEED INK 1,1:PRINT j$;:FOR a=1 TO 100
     :NEXT:PLOT 269,270,0:PRINT j$;:FOR a=1 TO 200:NEXT:TAGOFF:RETURN
[7E] 1100 SOUND 4,0,15,15,1,,31:TAG:PLOT 293,270,7:SPEED INK 1,1:PRINT j$;:FOR a=1 TO 100
     :NEXT:PLOT 293,270,0:PRINT j$;:FOR a=1 TO 350:NEXT:TAGOFF:RETURN
[EF] 1110 GOSUB 1800:TAG:PLOT 5,350,15:SPEED INK 1,2:PRINT s$;:PLOT 15,300,15:PRINT s$;:
     TAGOFF:FOR t=1 TO 15:!SINK,1,2,7,7:!SINK,1,2,4,4:FOR a=1 TO 180:NEXT:NEXT:RETURN
[13] 1120 GOSUB 1800:TAG:PLOT 359,304,15:SPEED INK 1,2:PRINT s$;:PLOT 269,316,15:PRINT s$
     ;:TAGOFF:FOR t=1 TO 15:!SINK,9,10,6,6:!SINK,12,13,6,7:FOR a=1 TO 180:NEXT:NEXT:RETURN
[A9] 1130 GOSUB 1800:TAG:PLOT 200,255,15:SPEED INK 1,2:PRINT s$;:PLOT 290,285,15:PRINT s$
     ;:TAGOFF:FOR t=1 TO 15:!SINK,7,8,9,10:!SINK,10,10,7,8:FOR a=1 TO 180:NEXT:NEXT:RETURN
```

cont. on p.33

HOMEBREW REVIEWS

Lords of Magic

– BY TONY KINGSMILL
(Price – £3.95 on disc)

This is an adventure game which was written using The Quill with graphics designed by the Illustrator from Gilsoft.

The plot is a simple one :– You have been sent to a distant land by powerful magic from where no–one has ever escaped. Many creatures will attack you at first sight but a few have avoided the magical lords and live away from their enemies. Most of these are humans and they know ways of defeating the magic but are too old to do so. Your task is straightforward – find a way to defeat the magic and then return home.

Now onto actually playing the game. The loading screen was particularly good though there were no graphics but it was well laid out & looked good to the eye. I was pleased to find that there were graphics included in 'Lords Of Magic' but some of them were not very good and I had to rely totally on the description to know where or what the location was & the graphics were the game's worst point by far.

When I started playing the game, I was quite impressed with the text and the overall descriptions (although some were rather short) of the various locations but I thought that there could have been a larger vocabulary for the the player to use and I would have liked to have seen a command to 'TAKE ALL' and to 'DROP ALL', which would have made for less typing.

But in general the game was by no means bad but at the same time was nothing special. However, let us not forget that it is a homebrew game and the price is reasonable at £3.95 (including postage and packing). Unfortunately it is available only on disc but apparently there is the possibility of the game being released on tape at a later date. Be warned, though, that on a CPC 464 (even with a disc drive) the loading screen program doesn't work correctly but this can be overcome by simply running the game directly.

If you do buy the game but get stuck and would like some help (& believe me, the game is hard!), Tony Kingsmill has produced some free clue sheets which are available, providing that you send an SAE. The sheets include a list of all the objects in the game, the most interesting locations & the solutions to the many puzzles.

LORDS OF MAGIC costs £3.95 on disc and can be obtained from:
Tony Kingsmill, 202 Park Street Lane, Park Street, St Albans
Hertfordshire AL2 2AQ
PLEASE MAKE ALL CHEQUES/POSTAL ORDERS PAYABLE TO T.KINGSMILL

Island of Chaos – BY TONY KINGSMILL

(Price – £3.95 on disc)

This is the second of the two adventure games by Tony Kingsmill. Again it is Quill based with the graphics designed on the Illustrator.

For some reason I liked the background to the game & the plot is that a long time ago, there was a prosperous island called Brael Ti in the sea of Karzania. Trading had always been an important feature of the island and Kansith, who was the island's leader, was rich and believed to have possessed magic but despite his fortune he looked after his people well. However soon Brael Ti was attacked & taken over by a man called Baktron. Kansith was murdered & Brael Ti was isolated from the outside world. Trading was forced to stop and no-one dared to set foot on the island. But now, you have to explore the island & defeat the mighty Baktron himself.

As I loaded the Island of Chaos I wondered whether it would be upto the same standard as the previous game, Lords of Magic. I wasn't disappointed. The loading screen was again good as were the imaginative descriptions & puzzles (some of which were quite tricky!).

Like Lords of Magic this game included graphics but unfortunately they were not very good (the ones I saw at least) at all and this rather spoilt the whole impression of the game and lowered its standard somewhat (as did a large number of spelling mistakes!)

That apart the game was a good one and there were a number of small features I particularly liked. These were notably the SAVE and LOAD postion features and the SCORE command which gave your present score and health at the touch of a button and the game was made even more interesting by an enjoyable, if slightly limited, interaction with other characters. Another incidental that I found impressive was the presentation of both of his games; it was obvious that a great deal of effort had gone into making neat, clear and informative covers for his games. It's little things like this that make a homebrew game more professional and a tip that other homebrew software producers should take up.

Like the previous game, it failed to load on a CPC 464 but again this could easily be got around either by altering a line or two or running the main game directly. On the 6128, however, the game loaded without any hitches.

So assuming that this small problem can be sorted out, I thought the game was appealing enought, of a good enough standard (though by no means excellent) and if you especially enjoy adventure games I would say that it's good value at £3.95 on disc.

Like Lords Of Magic it is available only on disc at the moment & again there are free clue sheets available providing you send an SAE.

ISLAND OF CHAOS costs £3.95 on disc and can be obtained from:
Tony Kingsmill, 202 Park Street Lane, Park Street, St Albans
Hertfordshire AL2 2AQ
PLEASE MAKE ALL CHEQUES/POSTAL ORDERS PAYABLE TO T.KINGSMILL

Educational Pack One

– BY MATTHEW PINDER, MIP SOFTWARE
(Price – £5.95 on tape or disc)

Back in Issue Two of Print-Out we reviewed Maths Master Plus, a mathematical aid by the small software company, MiP Software, and it was highly recommended. Now, MiP have produced an education pack which includes Maths Master (the predecessor of Maths Master Plus) to suit ages from 9 to 14.

The pack includes a large number of mathematics programs that cover a wide range of subjects. They include :– addition and subtraction (with 3 different skill levels), multiplication and division, simultaneous equations, fractions & percentages, areas, ratios, number series, indices and number guess. The last program in the collection allows you to practice your weighing and balancing skills.

With the disc (a tape version is available) comes a well presented and very comprehensive instruction sheet of all the programs which explains each of them in a clear and useful way. I won't bother to go through each of the 10 programs in turn but I will give my general view on the whole pack.

The programs were well presented and organised and there was no confusion as to what was going on at any time. Having said that, I thought that some of the programs could have done with clearing the screen a bit more often, as when the wrong type of input was entered the screen quickly turned into a mess of words and numbers.

I had the disc version to review and there was no problem with waiting for the games to load but I would imagine that there could be a long wait on the tape version (as I experienced when reviewing Maths Master Plus a while ago).

The Educational Pack is probably aimed at the less advanced mathematicians (more for 8–11 year olds) whereas the Maths Master program is designed for the more advanced user. Maths Master is not an educational program but is rather an advanced calculator where you enter the numbers and the computer gives you the answer whereas the other programs in the Educational Pack are purely learning aids. The only difference between Maths Master and Maths Master Plus is that the latter is a more upto date version of the other and features more functions. If you find maths hard, this must surely be a certain way to improve although the game provides you with no real incentive to continue playing it.

The price is a very reasonable £5.95 on disc or double tape pack and with it comes the extremely thorough instruction sheets. At that price, although not a bargain, it certainly represents good value for money.

I also hear (on the grapevine!) that MiP are planning an Educational Pack 2 which will have an 'Englishy theme' (?) I shall wait in suspense and keep you informed in further issues of Print-Out.

EDUCATIONAL PACK ONE costs £5.95 on disc or double tape pack and is available from MiP Software, 4 Wham Hey, New Longton, Preston, Lancashire PR4 4XU.
PLEASE MAKE ALL CHEQUES/POSTAL ORDERS PAYABLE TO MATTHEW PINDER.

PUBLIC DOMAIN ~ CPD!

A short while ago, Print-Out heard of a new Public Domain library - CPD - which was tape based. There are very few such libraries & so we asked the man behind it, Alan Scully, to write an article on his interesting venture and here it is.

Ever since I got my CPC, I have been programming. I chose the CPC because the BASIC was like that of the BBC & was simple to learn. Within days I was writing graphics programs, & within weeks I started on my first game, SUGAR DISC, which I worked on for months because I would keep adding bits into it to try and make it look more professional.

I first got the idea for distributing Public Domain Software through Amstrad Action when I realised that there was a vast shortage of cassette based Public Domain libraries for the Amstrad and so CPD, Cassette Public Domain, was born. At first, getting programs was difficult. I had a large number of games that I had written myself but more serious software like a Word Processor & an assembler were more difficult to find. I hunted around other Public Domain libraries to get software, and the 'Type-ins' page of Amstrad Action proved a great help. Then I had the problem of publicity. I wrote to Print-Out, and they invited me to write this article.

CPD has every kind of program you could possibly want for your Amstrad, including Utilities, Educational Programs, Artificial Intelligence programs, serious software & applications software. If all you want are games, then you need look no further than CPD. We've three games packs that promise to entertain you with every type of game you could possibly imagine:- Maze Games,Strategy Games,Snake Games, Card Games, Shoot-em Games, even a Break-Out clone & a Cricket Game. CPD also caters for the BASIC programmer with the AA Sprite Editor, the AA Sprite Driver, & subroutines to help with the more complicated tasks. Such subroutines include Split Modes (from AA), 40 Characters in Mode 0, Multi-Height characters, scrolling messages and many more. As a first order I strongly recommend the CPD starter pack. It contains a selection from all our PD packs and is a great introduction to the world of Public Domain software. If of course you would rather choose a specific pack, then the packs CPD has are :-

CPD 1 : GAMES 1
CPD 2 : GAMES 2
CPD 3 : (BIG) GAMES 3
CPD 4 : SUBROUTINES/ROUTINES
CPD 5 : AI/EDUCATIONAL
CPD 6 : APPLICATIONS
CPD 7 : SERIOUS
CPD 8 : STARTER PACK

So, finally, we come to the cost. For the starter pack send :-

A blank cassette. 25p to cover duplication, and a Stamped Addressed Envelope.

For two selections from the above list, send :-

A blank cassette, 50p to cover duplication, and a Stamped Addressed Envelope.

The address is :-

ALAN SCULLY, 119 LAUREL DRIVE, GREENHILLS, EAST KILBRIDE, GLASGOW G75 9JG.

UTILITY

A MACHINE CODE RSX

STEP

—A Single Stepping Routine

by BOB TAYLOR

Many people have found out the hard way that Machine Code is very difficult to program correctly. One of the reasons for this is that machine code does not possess a thorough and helpful error trapping system such as that used in BASIC - often the first the programmer knows of an error is when the computer resets! For this reason it is useful to be able to keep track of the various registers and to see what effect the program has on them whilst running. In order to help solve this problem, we have devised an RSX called |STEP. It can be used to look at the values stored in the registers and to see how they are altered, step by step, whilst the program is running. |STEP is a relocatable RSX that is used to 'single step' through Machine Code instructions.

When entering the RSX, a hexadecimal display similar to the following will be shown:

```
SP BF00 0000 0000 0000 0000
IY 0000 01 89 7F ED 49 C3 00 00
IX 0000 01 89 7F ED 49 C3
HL 0000 01 89 7F ED 49 C3
DE 0000 01 89 7F ED 49 C3
BC 0000 01 89 7F ED 49 C3
AF 00 FF SZ h PnC
PC 0000 01 89 7F
```

The top line shows the Stack Pointer address (shown moved to &BF00 for use with |STEP) and the next four entries on the stack. This address is the low byte of the first 'word' (ie the second half of that word). In practice, this low byte is lower in memory than the high byte - however when displaying a word, the high byte is written first. The display of any words above &BEFF does not necessarily show that they have actually been PUSHed there. A word PUSHED onto this Stack will be sent to the two bytes below the existing address which will, itself, then be altered to point to the low byte that has just been PUSHed. The stack has no limits, so care must be taken not to PUSH below &BDF4 (&BE80 if a Disc Drive is fitted), or above about &BF40 else the computer might crash.

The next five lines give the contents of the registers IY, IX, HL, DE and BC as a word. In each case this is assumed to be an address and there follow the contents of the eight bytes starting from this address and upwards. In practice however, the register contents may not be an address or even a word (the HL, DE and BC registers can normally be split into halves - so can the IY and IX registers). In the display, the H, D & B registers occupy the high byte of the word (ie the first half) while the L,E & C registers are the low bytes. The CPC's do not readily support use of the alternate register set, and therefore no attempt has been made to display their contents.

The seventh line displays the Accumulator and Flag registers:- the first byte is A, the second F. The Flag register is then repeated in binary form (from bit 7 to 0) with the relevant bits identified by the initial of their name when set ie. Sign, Zero, half carry, Parity (=overflow), n (=add/subtract), Carry. Those signs, shown by lower case, cannot be tested and don't affect conditional Jumps etc. The two bits that aren't used are present as spaces to provide proportion.

The final line of the display is the Program Counter. The address shown is that of the next Machine Code instruction to be performed. This is followed by the full complement of bytes which go to make up that instruction - either 1,2, 3 or 4. The bytes are in memory sequence, so addresses and words are presented with the low byte first.

How to access STEP

This RSX is fairly simple and flexible to use but it has several different ways of utilising it.

1. Sometimes a Program Counter address parameter will be entered & this should immediately follow |STEP. This then becomes the address of the next instruction on the PC line. If this parameter is omitted, then the previous Program Counter address will be retained (or &0000 will be used if this is the first time that the RSX has been entered).
2. It is possible to enter values or addresses into any or all of the registers via parameters. Each such entry must have two parameters :- The first must be a value from 1 to 7 to identify the register required (1=AF, 2=BC etc up to 6=IY, 7=Stack Pointer). The second parameter will be the value or address required to be loaded - remember that the H, D and B registers require values multiplied by 256 (whereas L, E and C do not). Theoretically this should also apply to the A register but since this is much more frequently used than the F, the input for this pair of registers has been reversed to normal, and only Flag bits must be multiplied by 256.
3. A convenient way to utilise the register indentifying parameter would be to assign values of 1 to 7 to the BASIC variables A, B, D, H, X, Y and S - instead of entering a value as the first of the pair, the variable could be substituted in and thus making the sequence of parameters much clearer;

eg |STEP,&8000,a,65,h,&C000,c,&4000,d,&4000

Note that, apart from the PC address parameter, the order of any other register entries does not matter. The full syntax of the RSX is:

|STEP [,<Program Counter address>] [,<list of: [<Register identifier value>, <Register value>]]

Once !STEP has been entered, only three keys will be effective and they are:

J JUMP over the instruction at the Program Counter without performing it.

: PERFORM the instruction and update any registers affected by it. However, some instructions cannot be carried out due to the CPC firmware and these will be jumped over as if 'J' had been used. Of those allowed, most instructions (including all the 'illegal' ones involving halves of the IX and IY registers) are performed normally, but a few are simulated - CALL, JP, JR etc. CALL when the conditions allow it, is expected to RETurn from the CALLed routine to the instruction which follows; if it does not, the computer may lock-up and then have to be reset.

ESC to return to BASIC. All registers etc are preserved for the next entry to !STEP unless parameters are entered at that time.

ALL other keys presed will be displayed but no action will be taken.

It is often useful to precede a !STEP command with a CLS to give an uncluttered display. This was not incorporated in the RSX itself, so that any existing data elsewhere on the screen could be retained if needed. Some instructions are not performed at all and they are:- DI,EI,EX AF,AF',EXX,HALT,IM 0,1 OR 2,IND,INDR, INI,INIR,LD with Interrupt or Refresh registers,OUTD,OTDR,OUTI,OTIR,RETI,RETN.

LISTING

```
[F1] 10 'STEP-LOADER copyright R Taylor 1989
[98] 20 MEMORY HIMEM-&440:RESTORE:PRINT:PRINT"Please wait a few seconds"
[E0] 30 FOR lin=0 TO &440/8-1:total=0:FOR n=1 TO 8:READ a$
[54] 40 byte=VAL("&"+a$):POKE HIMEM+lin*8+n,byte
[4B] 50 total=total+byte:NEXT n
[21] 60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT"Error in line"lin*10+110:
        END
[44] 70 NEXT lin:IF PEEK(6)=&80 THEN POKE HIMEM+&30,&93:POKE HIMEM+&31,&CA
[E6] 80 PRINT:PRINT"All M/C loaded":PRINT:PRINT"Press 'S' to save M/C as STEP.BI
        N":WHILE INKEY$="":WEND:IF INKEY(60)<>-1 THEN a=HIMEM+1:SAVE "STEP.BIN",
        B,a,&440
[87] 90 PRINT:PRINT"To Load and Initialise !STEP RSX with a program present just
        Enter:":PRINT"MEMORY HIMEM-&440:a=HIMEM+1:LOAD"CHR$(34)"STEP.BIN"CHR$(34
        )",a:CALL a":PRINT"in Direct Command Mode with the Disc or Tape inserted
         at the correct place"
[EA] 100 END
[65] 110 DATA 21,3F,03,19,3E,10,4E,23,13B
[A9] 120 DATA 46,23,E5,D5,EB,09,4E,23,388
[A5] 130 DATA 46,CB,B8,EB,E1,E5,09,EB,56E
[13] 140 DATA 72,2B,73,D1,E1,3D,20,E6,405
[39] 150 DATA 01,34,80,EB,36,C9,23,C3,385
[D1] 160 DATA D1,BC,3E,05,0E,00,21,55,254
```

```
[DE] 170 DATA CB,C3,1B,00,07,83,FD,21,351
[E2] 180 DATA 2F,83,B7,28,34,47,DD,56,33F
[DD] 190 DATA 01,DD,5E,00,05,28,30,3E,1D7
[A0] 200 DATA 07,DD,96,02,FE,06,38,05,2BD
[F7] 210 DATA 20,D8,4A,53,59,3C,FD,E5,40C
[8C] 220 DATA FD,23,FD,23,3D,20,F9,FD,493
[AE] 230 DATA 72,FF,FD,73,FE,FD,E1,DD,69A
[1E] 240 DATA 23,DD,23,DD,23,DD,23,10,333
[2B] 250 DATA CD,FD,56,0F,FD,5E,0E,1A,3B2
[47] 260 DATA 13,FE,DD,28,41,FE,FD,28,47A
[E5] 270 DATA 3D,FE,ED,28,5E,21,5F,83,3B1
[E6] 280 DATA 01,4A,00,ED,B1,79,0E,01,271
[12] 290 DATA 20,6E,04,B7,28,6A,04,FE,2DD
[72] 300 DATA 06,38,65,04,FE,0A,38,60,247
[44] 310 DATA 04,FE,13,38,5B,0E,03,FE,2B7
[70] 320 DATA 25,38,55,04,FE,29,38,50,265
[FF] 330 DATA FE,31,38,06,0D,04,FE,37,2B3
[9C] 340 DATA 38,46,06,00,18,42,21,A9,1A8
[69] 350 DATA 83,01,55,00,1A,ED,B1,79,30A
[8F] 360 DATA 01,01,02,20,33,01,02,07,061
[0A] 370 DATA B7,28,2D,06,00,FE,36,38,27E
[8E] 380 DATA 27,0C,FE,51,38,22,0C,18,200
[9C] 390 DATA 1F,18,94,21,FE,83,01,3C,2AA
[FC] 400 DATA 00,1A,ED,B1,79,01,01,02,235
[9E] 410 DATA 20,0E,0E,02,FE,05,38,08,181
[9F] 420 DATA 06,00,FE,34,38,02,0E,04,184
[1C] 430 DATA 1B,FD,72,0F,FD,73,0E,3E,355
[D7] 440 DATA 1E,CD,5A,BB,D5,FD,E5,C5,57C
[94] 450 DATA 21,0D,83,0E,08,06,08,C5,19A
[E4] 460 DATA 7E,23,B7,C4,5A,BB,20,F8,449
[E3] 470 DATA 06,02,3E,20,CD,5A,BB,C5,30D
[B3] 480 DATA 06,02,FD,7E,01,5F,0F,0F,201
[DE] 490 DATA 0F,0F,E6,0F,C6,90,27,CE,35E
[98] 500 DATA 40,27,CD,5A,BB,7B,10,F2,3C6
[22] 510 DATA C1,05,28,09,FD,2B,79,FE,396
[DD] 520 DATA 02,20,DC,18,D5,FD,23,C1,3CC
[84] 530 DATA 79,FE,02,20,16,3E,20,CD,2DA
[9D] 540 DATA 5A,BB,7E,23,CB,03,38,02,2BE
[31] 550 DATA 3E,20,CD,5A,BB,10,F3,18,35B
[8F] 560 DATA 3F,18,AA,FD,56,01,FE,01,354
[38] 570 DATA 20,03,E3,45,E3,FE,08,20,354
[E1] 580 DATA 01,13,3E,20,CD,5A,BB,C5,319
[49] 590 DATA 06,02,1A,0F,0F,0F,0F,E6,144
[1E] 600 DATA 0F,C6,90,27,CE,40,27,CD,38E
[44] 610 DATA 5A,BB,1A,10,F2,C1,05,28,31F
[7F] 620 DATA 0F,79,FE,08,20,DB,1B,CB,36F
[47] 630 DATA 40,20,DC,13,13,13,18,D1,25E
[80] 640 DATA 3E,12,CD,5A,BB,3E,0A,CD,347
[10] 650 DATA 5A,BB,3E,0D,CD,5A,BB,FD,43F
[38] 660 DATA 23,FD,23,0D,20,AB,C1,FD,3D9
[D5] 670 DATA E1,D1,3E,8F,CD,5A,BB,3E,49F
[D2] 680 DATA 12,CD,5A,BB,3E,0D,CD,5A,366
[A4] 690 DATA BB,CD,06,BB,FE,FC,C8,F6,601
[30] 700 DATA 20,CD,5A,BB,D6,3A,28,0A,344
[A4] 710 DATA D6,30,20,DE,47,EB,09,EB,42A
```

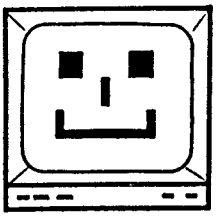
Linechecker

A PROGRAM TYPING AID
for use with PRINT-OUT

In Issue Three of Print-Out, we included a listing called 'Line-Checker' that produced codes for every line of a program in order to help eliminate typing errors.

The numbers that are printed in square brackets at the beginning of each line are the codes. They should not be typed in & all the programs will run correctly even if 'Linechecker' is not used.

For full instructions on how to use it, please refer to the original article in Issue Three.



```
[FE] 720 DATA 18,64,B8,1A,CA,C2,82,05,361
[24] 730 DATA 28,23,05,28,EF,05,28,5B,1EF
[22] 740 DATA 05,28,70,05,28,53,05,28,14A
[AF] 750 DATA 1C,FE,DD,20,08,FD,56,05,377
[68] 760 DATA FD,5E,04,18,41,FD,56,03,30E
[16] 770 DATA FD,5E,02,18,39,FD,56,07,308
[03] 780 DATA FD,5E,06,18,31,FE,18,28,2E8
[F2] 790 DATA 20,30,05,FD,35,0B,18,0F,1B9
[0C] 800 DATA FD,4E,0C,FE,30,CB,71,38,3F9
```

```
[23] 810 DATA 02,CB,41,28,08,CB,5F,20,288
[1C] 820 DATA 08,13,13,18,11,CB,5F,20,1A1
[6D] 830 DATA F8,13,1A,13,FE,80,38,01,2EF
[BD] 840 DATA 15,83,5F,30,01,14,C3,77,276
[AF] 850 DATA 80,13,13,13,FD,66,01,FD,31A
[25] 860 DATA 6E,00,2B,72,2B,73,16,00,1BF
[BA] 870 DATA E6,3F,5F,FD,74,01,FD,75,468
[72] 880 DATA 00,18,E3,CB,4F,20,04,CB,304
[A5] 890 DATA 57,28,02,13,13,13,CB,47,1CC
[CA] 900 DATA 20,23,FD,4E,0C,FE,D0,CB,433
[59] 910 DATA 71,38,0E,FE,E0,CB,41,38,3D9
[76] 920 DATA 08,FE,F0,CB,51,38,02,CB,417
[EA] 930 DATA 79,28,06,CB,5F,20,06,18,20F
[AB] 940 DATA B5,CB,5F,20,B1,FD,66,01,414
[5F] 950 DATA FD,6E,00,CB,4F,20,0E,CB,37E
[4B] 960 DATA 57,28,11,2B,72,2B,73,FD,2C8
[C1] 970 DATA 74,01,FD,75,00,E8,2B,56,353
[B6] 980 DATA 2B,5E,18,92,5E,23,56,23,22D
[69] 990 DATA 18,A1,21,EA,82,EB,3E,05,374
[DE] 1000 DATA 91,ED,B0,EB,3D,28,04,70,3F2
[11] 1010 DATA 23,18,F9,D5,FD,E5,F3,ED,5CB
[F7] 1020 DATA 73,2D,83,31,31,83,FD,E1,3E6
[35] 1030 DATA DD,E1,E1,D1,C1,F1,ED,7B,68A
[70] 1040 DATA 2F,83,00,00,00,00,ED,73,212
[EF] 1050 DATA 2F,83,31,3D,83,F5,C5,D5,432
[40] 1060 DATA E5,DD,E5,FD,E5,ED,7B,2D,61E
[E0] 1070 DATA 83,FB,FD,E1,D1,18,90,53,528
[98] 1080 DATA 54,45,D0,00,00,53,50,00,20C
[C6] 1090 DATA 49,59,00,49,58,00,48,4C,1D7
[94] 1100 DATA 00,44,45,00,42,43,00,41,14F
[D3] 1110 DATA 46,00,53,5A,20,68,20,50,1EB
[6F] 1120 DATA 6E,43,50,43,00,00,00,00,144
[A8] 1130 DATA BF,00,00,00,00,00,00,00,0BF
[F5] 1140 DATA 00,00,00,00,00,00,00,21,021
[6C] 1150 DATA 00,34,00,38,00,86,00,BF,1B1
[7C] 1160 DATA 00,E4,00,11,01,ED,01,C3,2A7
[D6] 1170 DATA 02,D9,02,DC,02,E8,02,F0,395
[26] 1180 DATA 02,F3,02,FF,02,4F,02,06,24F
[C7] 1190 DATA 0E,16,1E,26,2E,36,3E,C6,1D0
[79] 1200 DATA CE,D6,DE,E6,EE,F6,FE,D3,71D
[1D] 1210 DATA DB,CB,10,18,20,28,30,38,27E
[EB] 1220 DATA 01,11,21,22,2A,31,32,3A,11C
[49] 1230 DATA CF,D7,DF,EF,C3,C2,CA,D2,695
[70] 1240 DATA DA,E2,EA,F2,FA,CD,C4,CC,6EF
[90] 1250 DATA D4,DC,E4,EC,F4,FC,C9,C0,6F9
[5D] 1260 DATA C8,D0,D8,E0,E8,F0,F8,C7,6E7
[89] 1270 DATA E7,F7,FF,08,D9,F3,FB,76,622
[53] 1280 DATA E9,21,22,2A,36,26,2E,46,226
[AE] 1290 DATA 4E,56,5E,66,6E,7E,70,71,335
[31] 1300 DATA 72,73,74,75,77,86,8E,96,3EF
[75] 1310 DATA 9E,A6,AE,B6,BE,34,35,23,3F2
[DB] 1320 DATA 2B,E5,E1,09,19,29,39,E3,358
[41] 1330 DATA F9,24,25,2C,2D,84,85,8C,330
[02] 1340 DATA 8D,94,95,9C,9D,A4,A5,AC,4E4
[BB] 1350 DATA AD,B4,B5,BC,BD,44,45,4C,464
[D1] 1360 DATA 4D,54,55,5C,5D,64,65,6C,2E4
[8B] 1370 DATA 6D,7C,7D,60,61,62,63,67,353
[B5] 1380 DATA 68,69,6A,6B,6F,E9,43,4B,38C
[19] 1390 DATA 53,5B,63,6B,73,7B,42,4A,2F6
[D6] 1400 DATA 52,5A,62,6A,72,7A,40,41,2E5
[32] 1410 DATA 48,49,50,51,58,59,60,61,2A4
[66] 1420 DATA 68,69,70,71,78,79,4F,5F,351
[1C] 1430 DATA 47,57,BB,AB,BA,AA,B3,A3,4BE
[7B] 1440 DATA B2,A2,67,6F,44,B9,A9,B8,488
[81] 1450 DATA A8,B1,A1,B0,A0,45,46,4D,422
[AF] 1460 DATA 56,5E,00,00,00,00,00,00,0B4
```

Advanced Basic !!

Why errors can be FUN

Every programmer in BASIC, unless they're very lucky or extremely talented, will have seen at some time an error message such as the common 'Syntax Error' or, even worse, the 'Unexpected RETURN'.

Seriously though, when any error is encountered in a BASIC program a complex and precise procedure is followed by the BASIC interpreter. There are four main error commands and they are ERR, ERL, ERROR and ON ERROR GOTO. However on a 664 or 6128 there's an additional command, DERR, which handles all disc errors. For now, we will be concerned only with the four instructions that exist on all the CPCs. Before we look at any examples of 'error trapping', we need to know what each of the above commands does. In fact, both ERR and ERL are not commands but actually variables that are looked after by the computer itself. When an error occurs, the number of that error is stored in the variable ERR - you can find a list of all the error numbers and also what they represent at the back of your manual. For example, type in this line :-

```
10 READ a$
```

Now run this short program and the message 'DATA exhausted in 10' is printed in order to inform you that no new data is available to be read. If you now enter:

```
PRINT ERR
```

The number '4' should appear and if we look this up in our manual we find that this signifies that the DATA has been exhausted. ERL is also a variable and it holds the line number in which the last error occured. So type :-

```
PRINT ERL
```

The number '10' now appears & this tells us that the error happened in line 10.

If you were writing a program that involved setting both the PEN & PAPER to bright magenta, it would be sensible to add a few lines to the program so that whenever an error occured, the screen would be set to a some sensible colours & an indication of the error & its location would be given. The program below is an example of this :-

```
   10 ON ERROR GOTO 1000
   20 INK 1,8:INK 0,8
   30 FOR i=1 TO 3
  999 END
 1000 REM Error trapping routine
 1010 MODE 2:INK 1,24:INK 0,0:PEN 1:PAPER 0
 1020 PRINT "ERROR";ERR;"in line";ERL
 1030 PRINT:LIST
```

When run, the program tells you that there is ERROR 26 in line 30 - ERROR 26 refers to a NEXT instruction being missing. The actual error trapping routine is contained in lines 1000-1030 and line 10 sets this up by saying that as soon an error is encountered the program must go to line 1000.

Another use of the error commands is so that the user can customise his own error messages. One way to do this is to send the program off to an error trapping routine by using the ON ERROR GOTO command, see if ERR equals a particular value & then take the appropriate action. This short program demonstrates this point.

```
  10 ON ERROR GOTO 1000
  20 MODE 2
  30 INPUT "Please enter a number :- ",a
  40 PRINT "The number is";a
  50 GOTO 30
1000 REM Error trapping routine
1010 IF ERR=6 THEN PRINT "The number is too long"
1020 RESUME
```

When any error occurs the program automaticlly goes to line 1000 where the ERR value is compared with 6 and if they are equal (ie ERROR 6 has taken place) the message is printed & the program then RESUMEs execution. If there was any error other than 6 (overflow), the program would have RESUMEd execution as if nothing had happened. In order to force an error to occur in this program, you need to enter a number of more than about 80 digits !! Of course, as your program grows bigger and more complex you may need further error messages to be incorporated into the 'Error trapping routine'.

In the above program, we met yet another new command, RESUME. This command tells the computer where to restart the program from after it has dealt with the error. There are three forms that RESUME may take and they are :-

a) RESUME - tells the computer to restart from the same line that caused the error. Often this will be result in the error recurring but in the above example it does not.
b) RESUME NEXT - tells the computer to restart from the line following that which caused the error (eg. in the above program RESUME NEXT would send the computer to line 40 as it was line 30 that caused the error).
c) RESUME <line number> - tells the computer to restart from the line number which follows the RESUME (eg. RESUME 100 restarts at line 100).

So far we have looked at how to work out which error occured and where & how to alter an existing error message. However, there is another thing that we can do using the various error commands that are provided by BASIC - we can design our own errors !!! You may well think that you have enough constraints in BASIC without inventing more yourself. Rest assure that these 'home-made' errors are meant only to be helpful and not restrictive. In BASIC, the errors from 1 to 30 are reserved by the Operating System. However all the errors from 31 to 255 are

available for our own use. The way to operate your own 'error messages' is to have some instruction that decides whether your own personal error has occured and if it has then it should invoke the error using the command, ERROR. (To see ERROR in action, just type ERROR 8 and the computer will react exactly as if a line really did not exist.) When an error is invoked, it is picked up by an ON ERROR GOTO command which immediately sends the computer to the 'Error trapping routine'. Here, the program checks to see if the error that occured matches any of the ones that it has been programmed to interpret. As soon as any necessary actions have been carried out the program restarts execution at the point which you tell it to, using the RESUME command.

The program listed below is a demonstration of a simple parsing routine. In it three error messages are defined (errors 50-52) and these are executed when certain conditions concerning the entering of words are met.

```
10 REM Simple Parser Demo
20 ON ERROR GOTO 1050
30 PRINT "Please enter your next command."
40 INPUT ">>> ",word$
50 know$="CLIMJUMPPULLPUSHLIFT"
60 where=0
70 word$=LEFT$(UPPER$(word$))
80 IF word$="" THEN ERROR 50
90 IF LEN(word$)>10 THEN ERROR 51
100 where=INSTR(know$,word$)
110 IF where=0 THEN ERROR 52
120 where=INT(where/4)+1
130 ON where GOSUB 1000,1010,1020,1030,1040
140 GOTO 30
1000 PRINT "You start to climb the rope but fall.":RETURN
1010 PRINT "You try and jump onto the ledge but you can't reach it.":RETURN
1020 PRINT "You pull at the rope and it unties itself.":RETURN
1030 PRINT "You push the rock towards the cliff face.":RETURN
1040 PRINT "The rock is too heavy to lift.":RETURN
1050 REM Error trapping routine
1060 IF ERR=50 THEN PRINT "I'm afraid that you didn't enter anything."
1070 IF ERR=51 THEN PRINT "I'm very sorry but that word is too long."
1080 IF ERR=52 THEN PRINT "I don't know that word. Please use another one."
1090 RESUME 30
```

In this program, whenever word$ is empty ERROR 50 occurs (line 80) and this causes the ON ERROR GOTO command (line 20) to make the computer go to line 1050 where the ERR number is checked and the appropriate message is printed before execution is returned to the main program at line 30 (RESUME 30). The other two errors (invoked in lines 90 & 110) also follow the same path & they too rejoin the main program at line 30. That concludes this section on error messages and their uses which can be of the greatest assistance whilst programming.

Listing

BUBBLE SORT

A Sorting Routine for use in your PROGRAMS

by BOB TAYLOR

The sort routine that is the subject of this article is intended as a module to be used in a Machine Code database program. Before embarking on such a listing, it is necessary to have decided on the form which the Database will take & especially on the way that data will be stored.

The unit of data is said to be the 'record' (equivalent to a card in a card index system) which will contain a number of related details such as the Name, Address, Phone No etc.

In a database, each of these details is stored in a separate 'field' & these will be linked together by the program. Although each record's fields could be stored with other like fields, it is more common to have all the fields for one record grouped together. Further to this, all the records are usually arranged sequentially in the same block of memory. To best utilise the somewhat limited memory in the CPC, variable length of fields (and therefore of records too) is needed in order to eliminate the unfilled bytes that result from using a system of fixed length fields which, of necessity, must be made large enough to cope with any eventuality.

Since the length of a field cannot be predicted, some way is needed to indicate its start and finish. To acheive this, a byte of low value (which would never occur in a field proper) is used at the start of each field; if the bytes are identical for the same field in each record then they can also act as field indicators. Thus for each record a series of bytes, which start at &00 for the first field, &01 for the next etc, is utilised as field separators. As can be seen the byte value is one less than the field number, with the first separator (&00) doubling as a start of record indicator also. To further assist with record management, double bytes of &00 are present both before the first record in the list and after the last so far entered. Thus a record starts at a byte of &00 and consists of the required fields with separator bytes between; it ends just before the byte of &00 which starts the following record.

Since a byte of &20 (32 decimal) represents a space (which may appear in any field), the maximum number of fields is limited to 32 (0 to 31), although if it is required to use Line Feed and Carriage Return characters (&0A and &0D respectively) in a field, the number drops to 10.

There many different methods of performing a sort, some faster than others. Unusually, the simplest methods do not work out to be the fastest. However, the more sophisticated ones require to work on specific records throughout the list - since we have elected to use variable length records, the positions of these records cannot be predicted and we are forced to go for one of the simpler routines usually called 'bubble sort' which works on adjacent records. It is given its name because records sort of 'bubble' to the required position in the list.

The version given here has been streamlined to eliminate the usual intermediate swapping of records and multiple passes through the list. It works as follows:

1. Starting with the first two, adjacent records are compared and if in the correct sequence, the higher of the pair plus the one above that are then compared.
2. This comparison of pairs is continued either until the top of the list is reached when the sort is complete, or until the upper one of a pair should actually be lower than it is.
3. When this latter occurs, this upper record is compared with each of those below it either until the bottom of the list is met or until a record is found which should not be above the upper one.
4. At this point, the upper record is sent to a buffer, the block of records which it should be below is moved up to where it was and the 'upper' record is then transferred from the buffer to just below the moved-up block.
5. The comparison of adjacent pairs is resumed with the highest one moved up to take the old upper one's place and the one above that.
6. At any point reached in the list, all records below those being compared will be in the correct order.

It is possible to sort using any field in the records (eg by Name or Town) by loading the sort variable FLDNUM with the number (minus 1) of the field in the sequence of fields in the record. Of course sorting one field causes the other fields to become unsorted as they are moved about with the record in the sort.

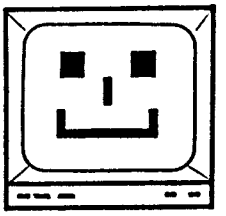
It is also possible to perform a reverse sort by loading the sort variable DIRFLG with anything other than 0. A reverse sort is used to put numbers into descending order. Whenever it is required to sort numbers, care must be taken to pad small numbers with the requisite number of leading spaces - failure to do so could lead to a number like 1000 appearing to be less than 99 because 1 is less than 9.

The relative slowness of the bubble sort can be alleviated to some extent by performing a sort after each new record is added to the list - a list which is nearly in the correct order is quicker to sort. To this end there is usually a sorted sequence (eg by Names alphabetically) which is most desirable and which will be retained most of the time. Other sequences will be required much less frequently.

After entering and running the Sort Loader program, a simple demonstration is given with a list of ten records, each having three fields, being generated and displayed. You will be prompted for a field number (1 to 3) and the list sorted and displayed again. This can be repeated as many times as you like. If &A578 is POKEd with a value other than 0, reverse sorts can also be demonstrated on re-RUNning from line 210. As should be seen, even empty fields can be handled; they are treated as less than fields containing characters. If no empty fields appear as the program is running, try ESCaping and RUN 210 until they do.

LISTING

```
[F1] 10 'SORT-LOADER by R Taylor for PRINT-OUT (copyright 1990)
[2C] 20 MEMORY &8FFF:RESTORE:PRINT:PRINT"Please wait a few seconds"
[2B] 30 FOR lin=0 TO &90/8-1:total=0:FOR n=1 TO 8:READ a$
[1E] 40 byte=VAL("&"+a$):POKE &A577+lin*8+n,byte
[4B] 50 total=total+byte:NEXT n
[98] 60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT"Error in line"lin*10+160
        :END
[C4] 70 NEXT lin
[47] 80 PRINT:PRINT"All M/C loaded":PRINT:PRINT"POKE &A577,<Field Number-1>":
        PRINT"POKE &A578,0 (for Normal Sort) or 1 (for Reverse Sort)"
[EF] 90 PRINT:PRINT"Code is located at &A577 and is &8B bytes long with entry
        at &A580"
[A1] 100 '** DEMONSTRATION **
[C4] 110 POKE &9000,0:ADDR=&9001:FOR r=1 TO 10:FOR f=0 TO 2:POKE addr,f:addr=
         addr+1:FOR c=1 TO RND*9:POKE addr,65+25*RND:addr=addr+1:NEXT c:NEXT f:
         NEXT r:POKE addr,0:POKE addr+1,0
[C8] 120 addr=&9001:r=11
[22] 130 c=PEEK(addr):addr=addr+1:IF c=0 THEN PRINT:r=r-1 ELSE IF c<10 THEN
         PRINT,ELSE PRINT CHR$(c);
[C7] 140 IF r GOTO 130
[DA] 150 PRINT:INPUT"Sort by Field No ";n:IF n<1 OR n>3 GOTO 150 ELSE POKE
         &A577,n-1:CALL &A580:GOTO 120
[6B] 160 DATA 00,00,00,00,00,00,00,00, 000
[AC] 170 DATA 00,21,01,90,54,5D,AF,06, 218
[28] 180 DATA FF,23,ED,B1,BE,C8,2B,22, 493
[4D] 190 DATA 7D,A5,EB,3C,32,7F,A5,22, 3C1
[54] 200 DATA 79,A5,3A,77,A5,ED,B1,EB, 4FD
[4C] 210 DATA 2A,7D,A5,ED,B1,3A,78,A5, 441
[E0] 220 DATA B7,28,01,EB,1A,FE,20,38, 33B
[23] 230 DATA 1E,CB,EF,4E,CB,E9,B9,38, 4CB
[FB] 240 DATA 16,20,04,23,13,18,ED,2A, 19F
[C5] 250 DATA 79,A5,22,7B,A5,2B,AF,BE, 3F8
[32] 260 DATA 28,09,ED,B9,23,18,C5,3A, 311
[E2] 270 DATA 7F,A5,B7,2A,7D,A5,20,AC, 3F3
[29] 280 DATA 23,E5,ED,B1,D1,ED,52,44, 4FA
[9C] 290 DATA 4D,19,2B,E5,11,76,A5,C5, 367
[57] 300 DATA D5,E5,ED,B8,ED,5B,7B,A5, 5C7
[CD] 310 DATA ED,52,44,4D,19,D1,ED,B8, 45F
[9E] 320 DATA E1,C1,ED,B8,E1,2B,ED,B9, 5F9
[11] 330 DATA 23,18,81,00,00,00,00,00, 0BC
```



Over the page there is the disassembly of the machine code that is contained in this routine and it shows the workings of the program in greater depth.

```
.botstt EQU  &9001  ;start of bottom record.
.buftop EQU  &A576  ;top byte of buffer.

ORG  &A577
.fldnum DEFB &00    ;field number - 1.
.dirflg DEFB &00    ;direction flag: 0=Normal Sort; <>0=Reverse.
.lwrstt DEFS 2      ;start of lower of 2 records being compared.
.blkstt DEFS 2      ;start of block to be moved upwards.
.uprstt DEFS 2      ;start of upper of 2 records being compared.
.movflg DEFS 1      ;move flag: 1=no move; 0=move required.

.sort
LD   HL,botstt      ;start at bottom of records.
.nxtup
LD   D,H
LD   E,L            ;DE -> lower record start (LWRSTT)
XOR  A              ;0 to look for start of next record upwards.
LD   B,&FF          ;make BC large enough for all CPIRs etc.
INC  HL             ;HL -> 1st char of lower record.
CPIR                ;HL -> next record 1st char or Top of
                    ;Records byte (&00).
CP   (HL)           ;Zero if at top of records.
RET  Z              ;finished if at top of records.
DEC  HL             ;HL -> start of next/upper record (UPRSTT).
LD   (uprstt),HL    ;store.
EX   DE,HL          ;HL -> LWRSTT.
INC  A              ;signal 'no move yet'.
.nxtdwn
LD   (movflg),A     ;store move flag.
LD   (lwrstt),HL    ;store.
LD   A,(fldnum)     ;get field number - 1.
CPIR                ;HL -> 1st char in lower required field.
EX   DE,HL          ;DE -> "  "        "      "    "
LD   HL,(uprstt)
CPIR                ;HL -> 1st char in upper required field.
LD   A,(dirflg)     ;get direction flag.
OR   A              ;Zero if Normal Sort.
JR   Z,nxtchr       ;if Normal then lower record should be
                    ;inferior (to upper).
EX   DE,HL          ;otherwise Reverse so upper record should be
                    ;inferior (to lower).
.nxtchr
LD   A,(DE)         ;get 1st/next char from inferior record.
CP   &20            ;field separators have values < &20.

JR   C,chkmov       ;if field separator then inferior field has
                    ;ended; ie is less or equal so no move.
SET  5,A            ;convert inferior's char to lower case.
LD   C,(HL)         ;get 1st next superior record char.
SET  5,C            ;convert this to lower case.
CP   C              ;compare both record's chars.
JR   C,chkmov       ;if inferior is lesser then don't move this
                    ;lower record, but check whether others need
                    ;moving.
JR   NZ,setmov      ;if inferior is greater then it will require
                    ;moving.
INC  HL             ;otherwise if equal then step on -
INC  DE             ;to next chars of both records.
JR   nxtchr
.setmov
LD   HL,(lwrstt)    ;include lower record -
LD   (blkstt),HL    ;in those to be moved.
DEC  HL             ;step down to last byte of next lower record
                    ;or to Start of Records byte (&00).
XOR  A              ;0 to look for &00 byte.
CP   (HL)
JR   Z,move         ;if Start of Records byte then no more to
                    ;check downwards.
CPDR                ;otherwise find start of record below.
INC  HL             ;HL -> start of new lower record.
JR   nxtdwn         ;to store 0 in move flag (= move).
.chkmov
LD   A,(movflg)
OR   A              ;Zero if move flag already set.
.move
LD   HL,(uprstt)    ;collect UPRSTT for NXT UP or MOVE proper.
JR   NZ,nxtup       ;if no move required.
INC  HL             ;HL -> 1st char of upper record.
PUSH HL             ;-> (1) UPRSTT + 1
CPIR                ;HL -> next higher record start + 1.
POP  DE             ;UPRSTT + 1 (0)
SBC  HL,DE          ;HL = UPRLEN (upper record length).
LD   B,H
LD   C,L            ;BC = UPRLEN.
ADD  HL,DE          ;HL -> next higher record start + 1.
DEC  HL             ;HL -> next higher record start (NXTSTT).
PUSH HL             ;-> (1) NXTSTT
LD   DE,buftop      ;DE -> buffer top byte.
PUSH BC             ;-> (2) UPRLEN
PUSH DE             ;-> (3) BUFTOP
PUSH HL             ;-> (4) NXTSTT
LDDR                ;transfer upper record to buffer; HL ->
                    ;upper record start (UPRSTT).
LD   DE,(blkstt)    ;DE -> start of lowest record to be moved.
SBC  HL,DE          ;HL = BLKLEN (length of block to be moved).
LD   B,H
LD   C,L            ;BC = BLKLEN.
ADD  HL,DE          ;HL -> UPRSTT.
POP  DE             ;NXTSTT (3)
LDDR                ;transfer block of record(s) upwards to
                    ;cover old upper record; HL -> new BLKSTT
                    ;(=old BLKSTT + UPRLEN).
POP  HL             ;BUFTOP (2)
POP  BC             ;UPRLEN (1)
LDDR                ;transfer upper record from buffer to below
                    ;new BLKSTT.
POP  HL             ;NXTSTT (0)
DEC  HL             ;HL -> last byte of new upper record.
CPDR                ;find start of record which has taken the
                    ;place of the old upper record.
INC  HL             ;HL -> start of new upper record.
JR   nxtup          ;every record below this point is sorted so
                    ;go on to compare this new upper record with
                    ;the one above.
```

SEA BATTLE

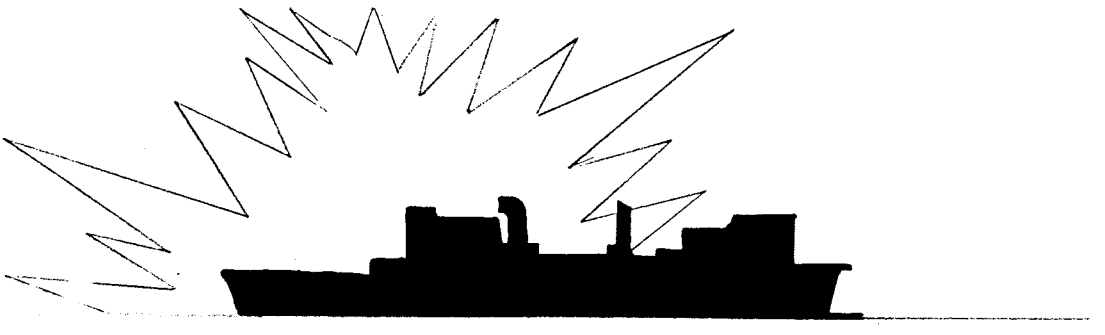
PROGRAM

cont. from page 16

```
[4B] 1140 IF cd=0 THEN RETURN ELSE IF cd=1 THEN sd=sd+1:FOR a=1 TO 1130:NEXT:IF sd<5 THEN
     RETURN ELSE IF sd=5 THEN GOSUB 1160
[52] 1150 IF cd=0 THEN RETURN ELSE IF cd=1 THEN sm=sm+1:FOR a=1 TO 1000:NEXT:IF sm<5 THEN
     RETURN ELSE IF sm=5 THEN GOSUB 1170
[99] 1160 LOCATE 5,5:PEN 7:PRINT f$;" WIN";:GOSUB 1520:LOCATE 5,8:PEN 8:PRINT"Press any
     key":GOTO 1180
[DB] 1170 LOCATE 5,5:PEN 7:PRINT q$;" WIN";:GOSUB 1520:LOCATE 5,8:PEN 8:PRINT"Press any
     key":GOTO 1180
[60] 1180 k$=INKEY$:IF k$=""THEN 1180:ELSE MODE 0:WINDOW 3,17,8,25:PEN 10:GOTO 30
[A5] 1190 IF c$="a1"AND cd=1 THEN TAG:GOSUB 1440:PLOT 49,193,10:MOVER 6,0:PRINT 1$;:MOVER
     -32,-15:PRINT i$;:TAGOFF
[7C] 1200 IF c$="a2"AND cd=1 THEN TAG:GOSUB 1440:PLOT 104,179,10:MOVE 93,192:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[27] 1210 IF c$="a3"AND cd=1 THEN TAG:GOSUB 1440:PLOT 144,179,10:MOVE 134,192:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[AE] 1220 IF c$="a4"AND cd=1 THEN TAG:GOSUB 1440:PLOT 169,179,10:MOVE 174,192:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[86] 1230 IF c$="a5"AND cd=1 THEN TAG:GOSUB 1440:PLOT 215,179,10:MOVE 215,192:PRINT 1$;:
     MOVER -35,-16:PRINT i$;:TAGOFF
[A9] 1240 IF c$="b1"AND cd=1 THEN TAG:GOSUB 1440:PLOT 49,157,10:MOVER 6,0:PRINT 1$;:MOVER
     -32,-15:PRINT i$;:TAGOFF
[3A] 1250 IF c$="b2"AND cd=1 THEN TAG:GOSUB 1440:PLOT 49,157,10:MOVE 93,158:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[F9] 1260 IF c$="b3"AND cd=1 THEN TAG:GOSUB 1440:PLOT 144,157,10:MOVE 134,158:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[80] 1270 IF c$="b4"AND cd=1 THEN TAG:GOSUB 1440:PLOT 169,157,10:MOVE 174,158:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[58] 1280 IF c$="b5"AND cd=1 THEN TAG:GOSUB 1440:PLOT 215,157,10:MOVE 215,158:PRINT 1$;:
     MOVER -35,-16:PRINT i$;:TAGOFF
[10] 1290 IF c$="c1"AND cd=1 THEN TAG:GOSUB 1440:PLOT 49,122,10:MOVER 6,0:PRINT 1$;:MOVER
     -32,-15:PRINT i$;:TAGOFF
[13] 1300 IF c$="c2"AND cd=1 THEN TAG:GOSUB 1440:PLOT 104,122,10:MOVE 93,122:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[B7] 1310 IF c$="c3"AND cd=1 THEN TAG:GOSUB 1440:PLOT 144,122,10:MOVE 134,122:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[3E] 1320 IF c$="c4"AND cd=1 THEN TAG:GOSUB 1440:PLOT 169,122,10:MOVE 174,122:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[16] 1330 IF c$="c5"AND cd=1 THEN TAG:GOSUB 1440:PLOT 215,122,10:MOVE 215,122:PRINT 1$;:
     MOVER -35,-16:PRINT i$;:TAGOFF
[EB] 1340 IF c$="d1"AND cd=1 THEN TAG:GOSUB 1440:PLOT 49,87,10:MOVER 6,0:PRINT 1$;:MOVER
     -32,-15:PRINT i$;:TAGOFF
[FE] 1350 IF c$="d2"AND cd=1 THEN TAG:GOSUB 1440:PLOT 104,87,10:MOVE 93,87:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[4A] 1360 IF c$="d3"AND cd=1 THEN TAG:GOSUB 1440:PLOT 144,89,10:MOVE 134,89:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[CD] 1370 IF c$="d4"AND cd=1 THEN TAG:GOSUB 1440:PLOT 169,89,10:MOVE 174,89:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[A1] 1380 IF c$="d5"AND cd=1 THEN TAG:GOSUB 1440:PLOT 215,89,10:MOVE 215,89:PRINT 1$;:
     MOVER -35,-16:PRINT i$;:TAGOFF
[90] 1390 IF c$="e1"AND cd=1 THEN TAG:GOSUB 1440:PLOT 49,53,10:MOVER 6,0:PRINT 1$;:MOVER
     -32,-15:PRINT i$;:TAGOFF
[A0] 1400 IF c$="e2"AND cd=1 THEN TAG:GOSUB 1440:PLOT 104,53,10:MOVE 93,53:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[EB] 1410 IF c$="e3"AND cd=1 THEN TAG:GOSUB 1440:PLOT 144,53,10:MOVE 134,53:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
[6E] 1420 IF c$="e4"AND cd=1 THEN TAG:GOSUB 1440:PLOT 169,53,10:MOVE 174,53:PRINT 1$;:
     MOVER -31,-16:PRINT i$;:TAGOFF
```

```
[BC] 1430 IF c$="e5"AND cd=1 THEN TAG:GOSUB 1440:PLOT 215,53,10:MOVE 215,53:PRINT 1$;:
     MOVER -35,-16:PRINT i$;:TAGOFF:RETURN
[0C] 1440 FOR a=250 TO 2 STEP -5:SOUND 130,a+10,1,15:NEXT:RETURN
[74] 1450 FOR l=1 TO 1 STEP -3:FOR f=800 TO 160 STEP -25:SOUND 1,f,1,15,1,1:NEXT:NEXT:
     RETURN
[87] 1460 FOR a=200 TO 50 STEP-8:SOUND 130,a+10,5,15:NEXT:RETURN
[35] 1470 FOR a=1 TO 1000:NEXT:RETURN
[0E] 1480 SOUND 1,400:RETURN
[4D] 1490 SPEED INK 8,15:LOCATE 12,19:PEN 7:PRINT CHR$(22)+CHR$(1);:PRINT"*";:PEN 1:RETURN
[F7] 1500 LOCATE 12,19:PRINT"*":WINDOW#2,13,16,19,19:PAPER#2,3:RETURN
[FA] 1510 LOCATE 12,19:PRINT"*":WINDOW#2,13,16,19,19:PAPER#2,3:RETURN
[A2] 1520 FOR a=1 TO 200:SPEED INK 5,15:SOUND 130,a,10,15:BORDER 26:BORDER 14:NEXT:FOR l=
     1 TO 1 STEP -3:FOR f=100 TO 10 STEP -1:SOUND 1,f,1*2,15,1,1:NEXT:NEXT:RETURN
[11] 1530 IF c$="a1"AND cd=0 THEN TAG:GOSUB 1450:PLOT 49,193,10:MOVER 6,0::PRINT o$;:
     MOVER -32,-15:PRINT m$;:TAGOFF
[2A] 1540 IF c$="a2"AND cd=0 THEN TAG:GOSUB 1450:PLOT 104,179,10:MOVE 93,192:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[DC] 1550 IF c$="a3"AND cd=0 THEN TAG:GOSUB 1450:PLOT 144,179,10:MOVE 134,192:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[63] 1560 IF c$="a4"AND cd=0 THEN TAG:GOSUB 1450:PLOT 169,179,10:MOVE 174,192:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[3B] 1570 IF c$="a5"AND cd=0 THEN TAG:GOSUB 1450:PLOT 215,179,10:MOVE 215,192:PRINT o$;:
     MOVER -35,-16:PRINT m$;:TAGOFF
[42] 1580 IF c$="b1"AND cd=0 THEN TAG:GOSUB 1450:PLOT 49,157,10:MOVER 6,0:PRINT o$;:MOVER
     -32,-15:PRINT m$;:TAGOFF
[E1] 1590 IF c$="b2"AND cd=0 THEN TAG:GOSUB 1450:PLOT 49,157,10:MOVE 93,158:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[92] 1600 IF c$="b3"AND cd=0 THEN TAG:GOSUB 1450:PLOT 144,157,10:MOVE 134,158:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[19] 1610 IF c$="b4"AND cd=0 THEN TAG:GOSUB 1450:PLOT 169,157,10:MOVE 174,158:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[F1] 1620 IF c$="b5"AND cd=0 THEN TAG:GOSUB 1450:PLOT 215,157,10:MOVE 215,158:PRINT o$;:
     MOVER -35,-16:PRINT m$;:TAGOFF
[8D] 1630 IF c$="c1"AND cd=0 THEN TAG:GOSUB 1450:PLOT 49,122,10:MOVER 6,0:PRINT o$;:MOVER
     -32,-15:PRINT m$;:TAGOFF
[C1] 1640 IF c$="c2"AND cd=0 THEN TAG:GOSUB 1450:PLOT 104,122,10:MOVE 93,122:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[6C] 1650 IF c$="c3"AND cd=0 THEN TAG:GOSUB 1450:PLOT 144,122,10:MOVE 134,122:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[F3] 1660 IF c$="c4"AND cd=0 THEN TAG:GOSUB 1450:PLOT 169,122,10:MOVE 174,122:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[CB] 1670 IF c$="c5"AND cd=0 THEN TAG:GOSUB 1450:PLOT 215,122,10:MOVE 215,122:PRINT o$;:
     MOVER -35,-16:PRINT m$;:TAGOFF
[7D] 1680 IF c$="d1"AND cd=0 THEN TAG:GOSUB 1450:PLOT 49,87,10:MOVER 6,0:PRINT o$;:MOVER
     -32,-15:PRINT m$;:TAGOFF
[9E] 1690 IF c$="d2"AND cd=0 THEN TAG:GOSUB 1450:PLOT 104,87,10:MOVE 93,87:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[D5] 1700 IF c$="d3"AND cd=0 THEN TAG:GOSUB 1450:PLOT 144,89,10:MOVE 134,89:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[58] 1710 IF c$="d4"AND cd=0 THEN TAG:GOSUB 1450:PLOT 169,89,10:MOVE 174,89:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[2C] 1720 IF c$="d5"AND cd=0 THEN TAG:GOSUB 1450:PLOT 215,89,10:MOVE 215,89:PRINT o$;:
     MOVER -35,-16:PRINT m$;:TAGOFF
[06] 1730 IF c$="e1"AND cd=0 THEN TAG:GOSUB 1450:PLOT 49,53,10:MOVER 6,0:PRINT o$;:MOVER
     -32,-15:PRINT m$;:TAGOFF
[40] 1740 IF c$="e2"AND cd=0 THEN TAG:GOSUB 1450:PLOT 104,53,10:MOVE 93,53:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
```

```
[92] 1750 IF c$="e3"AND cd=0 THEN TAG:GOSUB 1450:PLOT 144,53,10:MOVE 134,53:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[15] 1760 IF c$="e4"AND cd=0 THEN TAG:GOSUB 1450:PLOT 169,53,10:MOVE 174,53:PRINT o$;:
     MOVER -31,-16:PRINT m$;:TAGOFF
[63] 1770 IF c$="e5"AND cd=0 THEN TAG:GOSUB 1450:PLOT 215,53,10:MOVE 215,53:PRINT o$;:
     MOVER -35,-16:PRINT m$;:TAGOFF:RETURN
[EB] 1780 IF c$="a1"OR c$="a2"OR c$="a3"OR c$="a4"OR c$="a5"OR c$="b1"OR c$="b2"OR c$=
     "b3"OR c$="b4"OR c$="b5"OR c$="c1"OR c$="c2"OR c$="c3"THEN ic=1:RETURN:ELSE
[F2] 1790 IF c$="c4"OR c$="c5"OR c$="d1"OR c$="d2"OR c$="d3"OR c$="d4"OR c$="d5"OR c$=
     "e1"OR c$="e2"OR c$="e3"OR c$="e4"OR c$="e5" THEN ic=1 :RETURN ELSE ic=0:GOSUB 1480:
     RETURN
[45] 1800 SOUND 1,0,150,15,1,,31:SOUND 4,0,40,15,0,0,2:RETURN
[DA] 1810 DEFINT a-z:cd=1:MODE 0:BORDER 0:GOSUB 720:CLS:GOSUB 790:WINDOW#2,1,20,12,25:
     PAPER#2,14:CLS#2:PEN#2,5:LOCATE#2,4,5:PRINT#2,"* SEABATTLE *":PAPER 0
[FD] 1820 LOCATE#2,9,8:PRINT#2,"By":LOCATE#2,3,11:PRINT#2,"S.Messina @ 1987":TAG:GOSUB 80
     0:GOSUB 1070:GOSUB 1000:RETURN
[59] 1830 SYMBOL 240,0,0,0,0,0,17,8,102,255:SYMBOL 241,1,0,0,4,77,159,252,223:SYMBOL 242,64
     ,128,128,128,129,194,236,255
[F6] 1840 SYMBOL 243,0,0,0,0,4,8,48,120:SYMBOL 244,1,255,127,63,31,15,7,3:SYMBOL 245,223,
     255,255,215,255,255,255,255
[FD] 1850 SYMBOL 246,255,255,255,255,255,255,255,255:SYMBOL 247,255,255,255,253,255,255,
     255,255:SYMBOL 248,239,254,252,220,248,240,224,192
[1C] 1860 SYMBOL 249,42,126,255,126,58,24,16,0:SYMBOL 250,0,0,0,28,62,127,127,127:SYMBOL
     251,127,127,127,62,28,0,0,0
[D5] 1870 SYMBOL 252,0,60,126,255,255,127,62,28:SYMBOL 253,2,32,141,30,63,31,78,5:SYMBOL
     254,0,99,99,99,54,54,28,28
[A7] 1880 SYMBOL 255,28,28,54,54,99,99,99,0
[EF] 1890 s$=CHR$(249):t$=CHR$(23)+CHR$(1):u$=CHR$(23)+CHR$(0):z$=CHR$(22)+CHR$(1):x$=
     CHR$(22)+CHR$(0):g$=CHR$(138)+CHR$(138):e$=CHR$(253):p$=CHR$(231)
[4F] 1900 h$=CHR$(240)+CHR$(241)+CHR$(242)+CHR$(243):l$=CHR$(250):i$=CHR$(251):o$=CHR$
     (254):m$=CHR$(255):j$=CHR$(111)
[B8] 1910 v$=CHR$(244)+CHR$(245)+CHR$(246)+CHR$(247)+CHR$(248)
[1E] 1920 DATA 1,10,160,33,21,160,205,209,188,201,25,160,195,38,160,195,199,160,195,205,
     160,0,0,10,160,83,73,78,203,83
[47] 1930 DATA 67,82,177,83,67,82,178,0,195,163,160,213,67,93,124,214,56,87,125,214,80,
     111,48,10,124,37,230,7,32,4
[69] 1940 DATA 124,198,8,103,229,126,18,28,32,11,20,122,230,7,32,5,122,205,211,160,87,44,
     32,11,36,124,230,7,32,5
[35] 1950 DATA 124,205,211,160,103,16,224,225,209,213,229,75,229,6,7,93,84,124,205,211,
     160,103,126,18,16,245,225,44,32,11
[6B] 1960 DATA 36,124,230,7,32,5,124,205,211,160,103,13,32,224,225,209,21,32,158,124,214,
     56,103,67,54,0,44,32,11,36
[B7] 1970 DATA 124,230,7,32,5,124,205,211,160,103,16,238,201,221,110,0,221,102,6,45,37,
     221,126,4,148,79,125,198,2,221
[A5] 1980 DATA 150,2,87,205,26,188,124,198,56,103,175,129,16,253,95,24,158,24,98,62,192,
     205,8,188,201,62,64,205,8,188
[39] 1990 DATA 201,214,8,201,end
[F9] 2000 READ a$:IF a$="end" THEN CALL 40960:RETURN:ELSE POKE add,VAL(a$):add=add+1:GOTO
     2000
```



Conditional Jumps and the Flags

Machine Code

It is very important in Machine Code, as it is in BASIC, to be able to make decisions within a program. In BASIC, the main decision making command is IF... THEN...ELSE (see this issue's Beginner's BASIC), & this is relatively simple to use and is extremely versatile. In Machine Code, however, we do not have such a universal and easy command available to us. Instead we have to use three things in conjunction and they are conditional jumps, comparing and the FLAG REGISTER. The first two are M/C instructions for use in programs whilst the third is the actual means of making decisions, and this is the first thing that we will look at.

THE FLAG REGISTER - Before looking at the Flag Register it will be necessary to remind ourselves of one or two facts concerning registers in general. There are several registers available for use in M/C (eg B,C,D,E,H,L) & these are general purpose registers in which we can store, or load, information. These registers can be used to hold any 8-bit number for any reason (but there are some conventions by which certain registers hold certain values) & they can be grouped together in register pairs (eg. BC,DE,HL). Each of these register pairs can hold a 16-bit number & can be used in instructions as register pairs. An 8-bit number is a binary number which has 8 digits (for a full explanation see Issue One - 'What is my Amstrad?') & an example is the number 11010010 which can also be represented by 210 (decimal) or even D2 (hexadecimal). The means of conversion is unimportant but the thing to remember is that each Binary digIT can only be either a 1 or a 0 (ie. on or off). As well as these general registers there are several specific ones which have a special use, one example is the accumulator (A) register which is mainly used in arithmetic and which we have already met. The Flag (F) register is another specific register and its use is to give information about the result of the last executed instruction. Together the A and F registers can also form a 'register pair' (AF) and are both 8-bit registers. However, AF has no specific purpose and is not used to allow a 16-bit number to be stored. Some instructions (eg. PUSH and POP which we haven't discussed yet) need to operate on a register pair and this is where AF is used. Normally, with an ordinary register, it is the value as a whole that is used. However with the Flag register, it is more important to know which bits are set (ie. equal to 1) or reset (ie. equal to 0). The reason for this is that each of the bits in the Flag register tells us something about the result of the previous command to be executed. In fact, two of the 8 bits in the Flag register are not used but the others signify the following :-

Bit	7	6	5	4	3	2	1	0
Signifies	S	Z	-	H	-	P/V	N	C

S = SIGN Flag
Z = ZERO Flag
H = HALF-CARRY Flag
\- = unused bits
P/V = PARITY/OVERFLOW Flag
N = ADD/SUBTRACT Flag
C = CARRY Flag

The 'half-carry' and 'add/subtract' flags cannot be tested and so we'll ignore them and 'parity/overflow' is also rather tricky and uncommon so we'll leave it until a later issue. That leaves the 'sign', 'zero' and 'carry' flags which are all explained below :-

THE SIGN FLAG is set (made equal) to one if the result of the last instruction was negative. If the result was not negative, this flag would be reset to zero. Therefore after the following instructions, the Sign Flag would be set to 1 :-

```
LD A,201        ; A = 201
SUB 232         ; A now equals -31 and the SIGN FLAG is set to 1
```

THE ZERO FLAG is set to one if the result of the previous instruction was equal to zero. Any other result would reset this flag to zero. After these two instructions, the zero flag would be set to 1 :-

```
LD A,-201       ; A = -201
ADD A,201       ; A now equals 0 and the ZERO FLAG is set to 1
```

THE CARRY FLAG is set to one if a 'carry' occurs in addition or if a number is 'borrowed' in subtraction. Both of the sets of instructions below will cause the carry flag ti be set to 1 :-

(i)
```
LD A,60         ; A = 60
ADD A,230       ; A = 290, cannot be stored in a single register
                ; so A = 290-256=34 and the CARRY FLAG is set
```

(ii)
```
LD A,60         ; A = 60
SUB A,230       ; A = -170, cannot be stored as an 8-bit number
                ; because of something known as 2's complement
                ; so A = -170+256=86 and the CARRY FLAG is set
```

The reason for this can be seen if we print the sums in their binary form :-

(i)
```
   0 0 1 1 1 1 0 0      =    60  (decimal)
+  1 1 1 0 0 1 1 0      =    230 (decimal)
-------------------------------------------
[1] 0 0 1 0 0 0 10      =    34  (decimal)
```

Cannot be held in an 8-bit number so this sets the CARRY Flag and is then discarded.

(ii)

```
    [1] 0 0 1 1 1 1 0 0      =      60  (decimal)
  -     1 1 1 0 0 1 1 0      =     230  (decimal)
  ---------------------------------------------
        0 1 0 1 0 1 1 0      =      86  (decimal)
```

Has to be borrowed from outside the 8-bit number and so it sets the CARRY Flag and is then ignored.

CONDITIONAL JUMPS

These are Machine Code's equivalents to the BASIC IF...THEN...ELSE command but are greatly limited. The main reason for this is that they can only test & act on the flags (2 of which cannot be tested). However, these problems can be got round by careful programming. We've already met the command JP (JumP) before & this was an example of an unconditional jump. JumP also has a conditional form which is used in much the same way. Following the acutal JP command there is a letter (or couple of letters) which stands for the various flags there is also a form which jumps when a flag is NOT set. The jumps that we will look at are:-

JP Z,zero	-	jump if the ZERO flag is set (the result was zero)
JP NZ,notzero	-	jump if the ZERO flag is not set (the result was not 0)
JP M,minus	-	jump if the SIGN flag is set (the result was negative)
JP P,pos	-	jump if the SIGN flag is not set (the result was positive)
JP C,carry	-	jump if the CARRY flag is set (the previous calculation produced a 'carry' or a 'borrow')
JP NC,nocarry	-	jump if the CARRY flag is not set (the previous sum did not produce a 'carry' or a 'borrow')

There are also two more conditional jumps (JP PO,label and JP PE,label) and they refer to the P/V flag which we are not going to discuss in this issue. We have briefly mentioned another jump command called JR (Jump Relative) which is used only for small jumps. JR can only act on four conditions and they are :-

JR Z,zero	-	see above
JR NZ,notzero	-	see above
JR C,carry	-	see above
JR NC,nocarry	-	see above

For the sake of completeness it is worth mentioning conditional returns (RET) and CALL commands which can both use all the conditions that JP can :-

RET Z	CALL Z,zero	RET NZ	CALL NZ,notzero
RET M	CALL M,minus	RET P	CALL P,positive
RET C	CALL C,carry	RET NC	RET NC.nocarry
RET PO	CALL PO,label	RET PE	RET PE,label

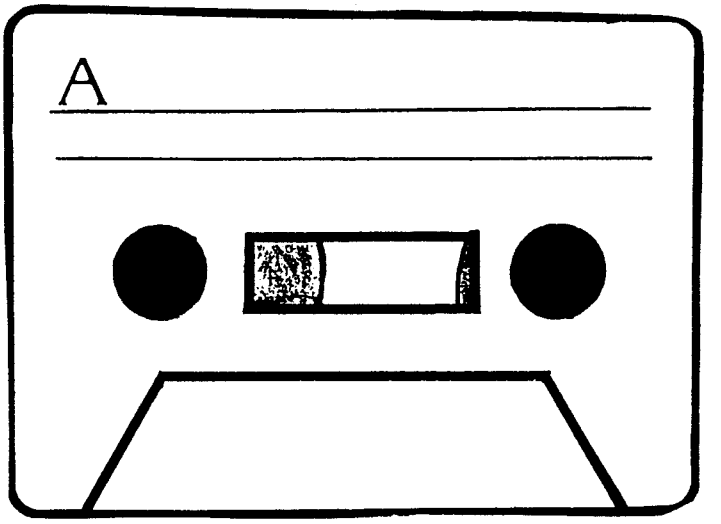
COMPARE (CP)

There will be times when you want to make a decision without actually changing any of the registers except the Flag. An example of this would be if you wanted to see if A held a particular value and if it did, the program should jump to a certain address but otherwise the value in A should be printed. It's obvious in this example, that there's no use in changing the value of A if you may need to use it again later and this is where the compare (CP) instruction comes in useful. CP actually subtracts the value following it from the accumulator and then sets the flags according to the result, without storing the result anywhere or altering the accumulator. To end this section on flags and conditions here is a program which asks a question, gets the user's reply and then makes a decision based on that input.

```
   org &4000
   ld a,2            ; A = 2
   call &bc0e        ; set mode to mode 2
   ld hl,text        ; HL now holds the address at which the text is stored
   call print        ; call the printing routine
   call &bb06        ; get the user to press a key and store its value in A
   cp 121            ; compare the value in A with 121 = 'Y' (ie. set flags
                     ; as if 121 had been subtracted from the value in A)
   call z,clear      ; and jump to .clear if the zero flag is set
   cp 89             ; compare the value in A with 89 = 'y' (ie. set flags
                     ; as if 89 had been subtracted from the value in A)
   call z,clear      ; and jump to .clear if the zero flag is set
   ret               ; return to BASIC
.clear
   ld a,2            ; A = 2
   call &bc0e        ; set mode to mode 2
   ret               ; return from subroutine
.print
   ld a,(hl)         ; A holds the contents of the address point to by HL
   cp 0              ; compare the value in A with 0 (ie subtract 0 from A)
   ret z             ; and return if the zero flag is set
   call &bb5a        ; print the character in A
   inc hl            ; HL = HL + 1
   jp print          ; jump to the label .print
.text
   db "Do you want me to clear the screen (y/n)",0
```

Although this is a very simple and short example of how conditional jumps and flags work, I hope that this has shown you that it is possible to make all the decisions that you need to in M/C with a little effort and skill. In Issue Five we'll look at some more conditional jumps of greater complexity and their uses.

Offers



Please make all cheques payable to Print-Out but any postal orders should be made out to T J Defoe as this saves the Post Office a great deal of time and effort. Unless it cannot be avoided, it is advisable not to send cash through the post.
All orders should be sent to :- PRINT-OUT, Special Offers, 8 Maze Green Road, Bishop's Stortford, Hertfordshire CM23 2PJ.

Issue 5

If you wish to order a copy of Issue Five in advance, you may do so by sending a cheque / postal order for £1.10 (or 70p + an A4 SAE with a 28p stamp) to the usual address. We hope to have it published by about the 30th May & as soon as it is printed it will be forwarded to you.

Program tapes and discs

We now supply both program tapes and discs for ALL issues and the prices given below also include a booklet to explain how the programs work plus postage and packing. Tapes and discs are available for Issues One, Two, Three and Four.
The cost for a program tape is as follows :-

a) A blank tape (at least 15 minutes) and 50p (p+p)
or b) £1.00 (which also includes the price of a tape)

The cost for a program disc is :-

a) A blank formatted disc and 50p (p+p)
or b) £3.00 (which also includes the cost of a MAXELL/AMSOFT disc) *

* When ordering using this particular method, please allow about 14 days for delivery as we must rely on outside suppliers for the discs.

Back issues

We still have some copies of Issues One, Two and Three available and the price is £1.10 which includes postage and packing. Alternatively, you can order both a back issue and its corresponding tape or disc by sending :-

a) £1.75 - includes the tape, the required issue and postage and packing
b) £3.75 - includes the disc (genuine MAXELL/AMSOFT disc) & the required issue and postage and packing.