KOBRAHSOFT SOFTWARE

SP7 ADVANCED TAPE TO +3 DRIVE UTILITY

OWERS MANUAL

INDEX

SP7 ADVANCED TAPE TO +3 DRIVE UTILITY

INSTRUCTIONS FOR USE

NOTE :- THIS UTILITY IS SUPPLIED ON THE UNDERSTANDING THAT YOU USE IT TO TRANSFER YOUR OWN SOFTWARE TO +3 DRIVE - AND DO NOT USE IT TO MAKE COPIES TO DISTRIBUTE OR ILLEGALLY SELL. THIS IS PIRACY, AND WE DO NOT CONDONE PIRACY!

GENERAL INTRODUCTION.

This SP7 Advanced Tape to +3 Utility is the second part of the SP7 package, and is most suitable for those people with some knowledge of machine code programming techniques, which will enable the transfer of even the LATEST protected programs to disc. It gives much useful information in the Technical Section below; and also gives examples of the transfer of some machine code protected programs. It also comes with our KD1 disassembler which lists machine code mnemonics to the screen.

NOTE:- AS MENTIONED IN THE BEGINNERS MANUAL, ALWAYS ENSURE THAT THE PROGRAM YOU ARE GOING TO TRANSFER WILL LOAD IN +3 BASIC, AS SOME EARLIER PROGRAMS WERE INCOMPATIBLE WITH THE +3 IN 128K MODE.

TECHNICAL SECTION.

When you load a program, you usually see at first a burst of RED/CYAN THICK STRIPES - this is called a LEADER (L), and is around 5sec. in length for a "HEADER", but only 2sec. for a "CODE BLOCK". (These terms will be explained shortly). After, comes a burst of BLUE/YELLOW NARROWER STRIPES - these are BYTES (B) i.e. bits of code. Thus, for a typical BASIC program you get:-

HEADER		CODE BLOCK	
------		----------	
L---B		L---B	
^ ^		^ ^	
5secs	V.short	2secs	Any length
RED/CYAN	BLUE/YELLOW	RED/CYAN	BLUE/YELLOW
THICK BANDS	THINNER BANDS	THICK BANDS	THINNER BANDS

The HEADER must always come first, since it tells the Spectrum where the following code must go in memory. The burst of BYTES for a header is always very short, since it always contains only 17 bytes. These 17 bytes give the following information:-

BYTE NUMBER	INFORMATION
1	This gives the TYPE of program i.e. 0 for BASIC, 1 for a NUMERIC ARRAY, 2 for a STRING ARRAY, 3 for MACHINE CODE.
2-11	These 10 bytes give the PROGRAM NAME, in Spectrum character codes.
12,13	These 2 bytes give, for a block of code, the CODE LENGTH; or, for a Basic program, the length of the program area PLUS its variables.
14,15	These 2 bytes give, for a block of code, the Start Address of the block in memory; or, for a Basic program, the AUTO-RUN line number.
16,17	These 2 bytes give, for a block of code, a repeat of the CODE LENGTH; or, for a Basic program, the length of the program area MINUS its variables.

The information in these 17 bytes can be obtained using the Header Reader we have supplied. After the HEADER, usually comes a CODE BLOCK. This contains a leader of around 2sec. and then CODE which can be of any length. Thus, we can have:-

```
L---B       L---B       L---B       L---B
  ^           ^           ^           ^
Header 1    Code 1      Header 2    Code 2
```

In every case, (L+B) = 1 BLOCK. e.g. for so-called HEADERLESS BLOCKS, we get:-

```
    L---B       L---B       L---B       L---B
      ^           ^           ^           ^
  Header 1    Code 1      Code 2      Code 3
```

i.e. here we have 4 BLOCKS but only 1 HEADER - this can be identified by only having ONE leader of 5sec. - the rest will be 2sec. The length of Headerless Code Blocks can be obtained using our Headerless Block Length Reader (see later).

SPECTRUM MEMORY MAP.

The Spectrum contains a total memory capacity of 65535 Bytes or locations (i.e. 64K). This consists of 16K of ROM (Read Only Memory) - which contains the operating and Basic systems and cannot be written to or changed. Also, it has 48K of RAM (Random Access Memory) - which CAN be written to or changed. For a detailed diagram - see P.182 of your Plus 3 Manual. Consider the following main memory areas (48K Mode only):-

DISPLAY FILE AND ATTRIBUTES.

This area, (16384-23295), is the memory mapped area which forms the "high resolution" display of the Spectrum. Anything POKED into these locations will appear on your screen i.e. type POKE 16384,255 - and a small line will appear in the top L.H. corner of your screen. Note:- POKE - puts a value into a memory location, whereas PEEK - reads a value from a memory location. i.e. type POKE 16384,0 - the line disappears. Typing PRINT PEEK 0 gives the number 243 - the first byte in ROM.

PRINTER BUFFER.

The 256 locations from 23296 to 23551 stores any numbers destined for your printer. Without a printer, it is free. NOTE:- This is ONLY in 48K Mode - in 128K Mode this area is occupied by extra System Variables (see below).

THE EXTENDED SYSTEM VARIABLES.

In 48K Mode the 182 locations from 23552 to 23733 contain the System Variables, whereas in 128K Mode an extended area of 437 bytes from 23296 to 23733 is used. Several of the variables are used in program protection (see later).

THE BASIC PROGRAM AREA.

This area of the memory holds the current Basic program lines, if any. The size of the area depends on just how many Basic lines exist. The start of the program area is always given by the value held in the System Variable PROG, which itself occupies the locations 23635 and 23636. Note that in the normal +3 Spectrum, PROG will indicate that the BASIC program starts at address 23755, and this will always be true unless the program has been written with a microdrive fitted (see later).

In the program area, BASIC lines are stored in the following format:- The first 2 bytes of any line hold the line number, with the first byte being the "high" byte and the second byte being the "low" byte. The third and fourth bytes of a line hold the "remaining length". This time the "low" byte comes before the "high" byte. The "remaining length" is the number of bytes from the fifth byte to the final ENTER character inclusively. Next, comes the BASIC line itself. Sinclair codes are used for the tokens and some characters. The last byte of a line is always an ENTER character (13). If a decimal number occurs in a BASIC line, it is stored as its ASCII characters and followed by the NUMBER character (14), plus 5 more bytes containing the Floating-Point form of the number. Type in the following:-

```
10: FOR A=23755 TO 24000:PRINT A,:PRINT PEEK A:NEXT A
```

This is a very useful program for examining a range of memory locations - we shall call it PEEK-LINE in the future.
now RUN it - it will print the addresses and characters contained in its own line i.e.:-

ADDRESS	NUMBER	TOKEN / CHARACTER
23755	0	) Line No. = (256*0)
23756	10	) +10 = 10.
23757	38	) Line Length=(256 *
23758	0	) 0)+38 = 38.
23759	235	FOR
23760	65	A
23761	61	=
23762	50	2
23763	51	3
23764	55	7
23765	53	5
23766	53	5
23767	14	) 6 byte F.P. No.
23768	0	) = (92*256)+ 203
23769	0	) = 23755
23770	203	)
23771	92	)
23772	0	)
23773	204	TO
23774	50	2
23775	52	4
23776	48	0
23777	48	0
(	)	(
23796	13	ENTER

NUMBERS.

We have talked above about 2 byte numbers and "high" and "low" bytes. The biggest number your Spectrum can contain in a single byte (location) is 255. So how can it handle bigger numbers? The answer is that it splits the number into 2 parts (for numbers from 0 to 65535). To calculate the number, you add the FIRST (or low order byte - L.O.B) to 256 x the SECOND (or high order byte - H.O.B). i.e. above:- LINE NUMBER = 10 + (256*0) = 10. Also, for the Floating Point number:- NUMBER = 203 + (92*256) = 23755. The L.O.B and H.O.B are usually stated, so you should not get them the "wrong way round".

RAMTOP AND CLEAR.

The main purpose in typing, say, CLEAR 29999, is that all the memory locations from 30000 up to the top of memory (65535), will be reserved - say for some machine code. The number here, 30000, is called RAMTOP - it is always 1 above the current CLEAR statement - it is the upper limit for your Basic program and its variables. NOTE:- Typing NEW will clear the memory up to RAMTOP but NOT above - this area will now only be cleared by:- (1) Disconnecting the mains supply at the back of your Spectrum, or (2) A better way (and less wearing on the Spectrum) is to type:- RANDOMIZE USR 0. This has the same effect - it is said to RESET the computer. Any machine code is best placed above RAMTOP, since it is then out of the dynamic Basic area, which can otherwise be moved around in memory by various peripherals e.g. microdrive etc.

MACHINE CODE CALLS.

Machine code routines are usually executed using RANDOMIZE USR number (R.USR n); or perhaps; PRINT USR number e.g. R.USR 0 above. This RESETS the computer by calling the machine code routines in ROM which start at address 0.

MICRODRIVES.

The main point to remember, is that when a microdrive is fitted, it can move up the start of the Basic program area (usually 23755). This is because the drive needs 58 more system variables than normal i.e. the value of PROG will be 23755 + 58 = 23813. However, since these extra variables are only invoked if a microdrive is fitted, this problem does not affect us when using the +3 Drive i.e. the value of PROG remains the same at 23755, irrespective of whether the drive has been used or not. This in itself helps make transferring programs to +3 Drive easier than transferring them to microdrive, since this problem does not occur.

NOTE:- Here, again, PROG is a 2 byte number, so we have:- PROG = Value at 23635 + 256 X Value at 23636. Where 23635 is the L.O.B; 23636 is the H.O.B.

DECIMAL AND HEXADECIMAL NUMBERS.

When we count in the usual DECIMAL way we say:-

0,1,2,3,4,5,6,7,8,9

Then:- 9+1 = 10 or:-

10x10x10 (Thousands)	10x10 (Hundreds)	10x1 (Tens)	10x0 (Units)
0	0	0	9
			+1
		1	0

i.e. the Decimal system is based on blocks of TEN. Similarly, the HEXADECIMAL system is based on blocks of SIXTEEN. Here:-

```
HEXADECIMAL:-  0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F
  DECIMAL  :-  0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
```

Then:-

16x16x16 (4096)	16x16 (256)	16x1 (16)	16x0 1	
0	0	0	F	(15 Dec.)
			+ 1	+1
==========	==========	==========	==========	====
		1	0	16

As you can see, the main difference are the letters A to F representing the numbers 10 to 15. Consider the decimal number FIVE THOUSAND ONE HUNDRED AND TWO - how is this represented numerically? First, we divide the number by the largest possible of the numeric blocks - here 1000. This gives 5x1000 = 5000, remainder = 102. Next, divide by 100 - this gives 1x100 = 100, remainder =2 i.e 0 tens and 2 units. Thus:- 5102 = 5x1000 + 1x100 + 0x10 + 2x1. Now, what is this in Hexadecimal? Here, the blocks are:-

16x16x16x16=65536 16x16x16=4096 16x16=256 16x1=16 16x0= 0 - F

Here:- 5102 / 4096 = 1 remainder = 1006
1006 / 256 = 3 remainder = 238
238 / 16 = 14 (= E) remainder = 14 (=E).

Thus, 5102 D = 13EE H. D and H represent Decimal and Hexadecimal.

Why do we need Hex. (for short) numbers, you may well ask? The reason is that most disassemblers (our KD1 too!) use Hex notation in their disassemblies. Also, Hex numbers are often used in Machine Code routines. We suggest you practice converting Hex to Decimal numbers and vice-versa until you feel you can understand them thoroughly. If in doubt as to what is a correct answer - use the number converter in our KD1 disassembler

THE MACHINE STACK.

This is the area of the computer's memory which is used to manipulate numbers. It is MOST IMPORTANT to always be aware of the location of the stack in memory, since, if it should get overwritten, the computer will almost certainly "lock-up" or crash. (See RAMTOP and CLEAR). A CLEAR instruction e.g. CLEAR 29999, ensures here that the stack is set below RAMTOP - thus any machine code above this address will NOT overwrite the stack. (See also machine code instructions, later).

INTRODUCTION TO MACHINE CODE.

Don't worry! It's not as difficult as you may think! In order to transfer the latest programs, a rudimentary knowledge of machine code is really required, since they are protected by various machine code routines.

WHAT IS MACHINE CODE?

Machine code is simply a sequence of numbers which the Z80 processor in your Spectrum can "understand", i.e. it translates the numbers into instructions which it then performs. The main reasons why machine code is hard to use are that (1) Any error in the number sequence will send a completely different instruction from the one you intended - usually crashing the computer. (2) There are no obvious error warnings. (3) There is no simple relation between the numbers and English i.e. as in Basic. However, we intend explaining just a few of the MAIN instructions - enough for you to tackle transfers to your +3 Drive!

A FEW SIMPLE MACHINE CODE INSTRUCTIONS.

The Z80 processor has THREE main sets of storage locations called REGISTERS. These are the HL, DE and BC registers. Each register can store a number from 0 to 65535. The letters chosen are arbritrary - but HL will help you remember how the number is stored. Remember, a number from 0 to 255 can be stored in ONE BYTE (location). A number from 256 to 65535 needs TWO BYTES - a H.O.B and a L.O.B. In the HL register:- the H register stores the H.O.B; the L register stores the L.O.B. Similarly for DE and BC. Each register pair, HL, DE or BC can be used singly i.e. D and E, H and L, or B and C. Another register pair also exists - AF. Here, A is called the ACCUMULATOR. F is called the FLAGS register. There is also an INDEX REGISTER - IX. This is used as a double register only i.e. you can't have I and X separately. The machine code instructions consist of numbers which are loaded into these registers.

NUMBER STORAGE IN THE Z80.

Another complication is that in machine code, numbers are stored in a "REVERSE" order! i.e. not as H.O.B first, L.O.B second, but L.O.B FIRST; H.O.B SECOND! This is just a convention, but it is important to remember it. Thus, suppose we have a number whose H.O.B = 82, and L.O.B = 100. As we showed you earlier, the actual number is:- 100 + 256*82 = 21092. This, however, is stored in machine code "IN REVERSE" i.e.:- ----100,(L.O.B)82,(H.O.B)----.

INSTRUCTIONS.

Below are the main instructions you will need, together with the machine code numbers which represent them. NOTE:- A SINGLE BYTE number = n, a TWO BYTE number = nn, or nH,nL (H.O.B, L.O.B).

MNEMONIC	INSTRUCTION	NUMBERS.
LD HL,nn	LOAD HL register with 2 byte number	33,nL,nH
LD DE,nn	LOAD DE register with 2 byte number	17,nL,nH
LD BC,nn	LOAD BC register with 2 byte number	1,nL,nH
LD IX,nn	LOAD IX register with 2 byte number	221,33,nL,nH
LD A,n	LOAD ACCUMULATOR with 1 byte number	62,n
SCF	SET CARRY FLAG	55
CALL nn	CALL routine at nn	205,nL,nH
RET	RETurn from routine	201
JP nn	JUMP to address nn	195,nL,nH
RST 0	RESTART ZERO	199
ADD HL,DE	ADD contents of DE to contents of HL	25
CP n	COMPARE number n with number in A	254,n
DEC HL	DECREMENT contents of HL register by 1	43
DEC BC	DECREMENT contents of BC register by 1	11
DEC DE	DECREMENT contents of DE register by 1	27
INC HL	INCREMENT contents of HL register by 1	35
INC BC	INCREMENT contents of BC register by 1	3
INC DE	INCREMENT contents of DE register by 1	19
LD (DE),A	LOAD contents of A into address pointed to by DE	18
LD A,(DE)	LOAD contents of address pointed to by DE into A	26
LDIR	LOAD, INCREMENT AND REPEAT	237,176
SBC HL,DE	SUBTRACT contents of DE from contents of HL	237,82

These are the main instructions. In total, for the Z80, there are over 600! NOTE:- MNEMONIC is the "shorthand" version of the instruction as shown in books, assemblers etc. (Pronounce it "NEEMONIC"). The LDIR instruction is special - it is used to move blocks of memory around i.e.:- HL contains the "start" address; DE contains the "destination" address; BC contains the "number of bytes to move". What it does is - load the contents of HL into address DE, then decrement BC - if this is not zero, increment HL and DE and repeat - hence "LOAD INCREMENT AND REPEAT". So whenever you see an LDIR instruction (or the numbers 237,176) - you know a block of memory is being moved. This is used as a method of protection in some programs - see later.

MACHINE CODE "LOAD" TECHNIQUES.

To help with +3 drive transfer, you must learn to recognise the sequence of instructions (numbers) frequently used in programs to LOAD in blocks of code. These are:- (1) SCF. (2) LD A,n. (3) LD IX,nn. (4) LD DE,nn. (5) CALL nn. (6) RET. Thus, to load a block of code using machine code, the sequence is:- (1) Set the carry flag - this signals "LOAD" (55). (2) Load the accumulator with a number - this is either 0 (to load a Header), or 255 (to load a code block). i.e. we can have:- 62,0 OR 62,255. (3) Load IX

with a number - this indicates the START ADDRESS (S.A.) of the block in memory i.e. 221,33,nL,nH. (4) Load DE with a number - this is the CODE LENGTH (C.L.) of the block i.e. 17,nL,nH. (5) Call address nn. This is USUALLY (but not always) the address of the LOAD routine in ROM, and = 1366 i.e.:- 205,86,5. (address = 86 + 5*256 = 1366). (6) Return from this routine i.e. 201. So, a typical sequence to LOAD would be:- 55,62,n,221,33,nL,nH,17,nL,nH,205,86,5,201, or any combination of these numbers. A typical "load game" structure is:- (1) BASIC LOADER. (2) LOAD SCREEN$ (picture). (3) LOAD GAME CODE. (4) JUMP TO START ADDRESS. In machine code, this would appear as:-

```
SCF         - signal "LOAD".
LD A,0      - signal load HEADER.
LD IX,nn    - START address
LD DE,17    - LENGTH = 17 bytes for a HEADER.
CALL 1366   - LOAD it.
SCF         - signal "LOAD".
LD A,255    - signal load "CODE".
LD IX,16384 - start address = start of screen RAM.
LD DE,6912  - length = length of screen RAM.
CALL 1366   - LOAD it.
SCF         - signal "LOAD".
LD A,255    - signal load "CODE".
LD IX,nn    - START address of code.
LD DE,nn    - LENGTH of code.
CALL 1366   - LOAD it.
JP nn       - JUMP to start address.
```

Now we know "what to look for", let's use this in conjunction with the methods used to protect modern programs. NOTE:- Use 48K Mode except where 128K Mode is specified.

PROGRAM PROTECTION METHODS.

Hints and Tips, Tricks of the Trade.

(1). Auto-Run in Basic.

Early Basic programs were protected by making them AUTO-RUN when reloaded with the LOAD "" command. This was done by saving them with:- SAVE"name" LINE n. Where n was the line number to start from. However, this is simply stopped by using:- MERGE "" instead of LOAD "". When the Basic loads, the auto-run is stopped and it can be listed as usual. NOTE:- You can similarly save a program to +3 drive with the command SAVE"name"LINE n. This will auto-run on reloading with LOAD "". This is stopped from auto running by reloading thus:- RESET the +3. At the initial Menu select +3 Basic by pressing the "down" cursor (arrow) key then ENTER. Next type load"t:" and press ENTER. This tells the +3 to expect a load from TAPE. Next, type merge"" then press ENTER and PLAY in the Basic - it will NOT auto-run and can then be LISTed.

Invisible Lines, etc.

A better protection for Basic programs is to insert the line:- POKE 23659,0 - near the start of your program. Any attempt to BREAK into the program will cause a crash - try it! Type:-

```
10 REM protected program
20 POKE 23659,0
30 PRINT "PROTECTED"
40 GOTO 40
```

Now, save to auto-run with:- SAVE "test" LINE 10. Next, load back, using LOAD "". The word PROTECTED appears. Now press BREAK - the computer crashes! Reset - by pressing the RESET button on the left hand side. The reason is that location 23659 is a System Variable called DFSZ - it tells the Spectrum the number of lines available for printing messages at the bottom of the screen. This is usually = 2. Try:- PRINT PEEK 23659 - you should get 2. If you change this to zero, no lines are available and the computer crashes. However, using MERGE, the program can still be stopped and listed! NOTE:- Line 40 stops message print out and stops an immediate crash. So now we will make our program lines INVISIBLE! Type:-

```
 5 POKE 23659,0
10 PRINT "This"
20 PRINT "is"
30 PRINT "Invisible"
40 GOTO 40
```

Enter EDIT mode (Caps Shift/1); edit each line. While in EDIT mode, press CAPS SHIFT and SYMBOL SHIFT to enter extended mode. Then, PRESS (CAPS SHIFT and 7),ENTER - the line disappears! Repeat with each line. A LIST now shows just the number 5 - the first line number! To see how this works, type:- PRINT PEEK 23635+256*PEEK 23636. This gives the start of Basic as 23755 as we explained earlier. Now type:-

```
50 FOR A=23755 TO 65535:PRINT A,:PRINT PEEK A:NEXT A
```

Now type:- RUN 50. This shows the above Basic lines. NOTE:- at 23759,23760 and 23786,23787 and 23800,23801 and 23812,23813 and 23831,23832 we see 16,7 i.e. INK 7. This makes PAPER and INK = 7 and makes the line invisible. Another method is to put a copyright line in your program (usually line 1) then POKE it to zero. Thus, type:- 1: REM This is MY Copyright. Now, find PROG as above. The first line number will occur at 23755+1= 23756. Thus, type:- POKE 23756,0 Now LIST the program. Your copyright is now in line 0 - and it can't be edited out - try it! This program could now be saved to auto-run. It can still be stopped and listed using MERGE, but the lines will be invisible! NOTE:- By checking PROG and listing the program lines - you could also POKE all the other line numbers to zero! NOTE:- Keep this program for (3) below.

(3). Changing Numerical Values.

Another useful ploy is to change any numerical values, so they read differently to what they really are! In the above program, find PROG. Type in the PEEK-LINE program to see the Basic lines. Suppose we see:-

ADDRESS	CODE	TOKEN
23843	16	INK
23844	7	7
23845	244	POKE
23846	50	2
23847	51	3
23848	54	6
23849	53	5
23850	57	9
23851	14	) Floating-Point
23852	0	) form of number
23853	0	) = 107 + 92*256
23854	107	) = 23659.
23855	92	)
23856	0	)

We can change locations 23846-23850 by POKEing in different numeric codes. Try:-

```
POKE 23846,54  (=6)
POKE 23847,53  (=5)
POKE 23848,50  (=2)
POKE 23849,49  (=1)
POKE 23850,48  (=0)
```

Also, POKE 23844,0 - restores INK 0 value. Now LIST it:- Line 5 has become:- POKE 65210,0! But the computer still uses only the Floating-Point number - so it in fact DOES POKE 23659,0! - remember this when hacking! Try it - type RUN. Press BREAK - the computer still crashes!

(4). Saving BASIC as Auto-Running CODE.

A Basic program can be saved as an apparent piece of auto-running CODE by the following method. Suppose we have the program:-

```
10 REM Auto-running Basic Code.
20 CLS: PRINT "Saved as CODE!":STOP
30 SAVE "auto-code" CODE 23552,370
40 GOTO 20
50 FOR A=23755 TO 65535: PRINT A,: PRINT PEEK A: NEXT A
```

Firstly, check where LINE 40 ends - (type:- RUN 50). Remember, look for an ENTER code (13) - this is the end of a Basic line. Subtract 23552 (S.A. of System Variables) - this gives 370 for the length of code to save. Now, type:- RUN 30 or GOTO 30. This executes line 30 and saves the System Variables and Basic, as CODE, of length 370 bytes. When reloaded using LOAD "" CODE, the program will auto-run starting at line 40 - the line AFTER the last one executed. The important point now is that you can't stop the auto-run with MERGE, since you can't MERGE CODE! NOTE:- In a proper Basic program, you would include the POKE 23659,0 to stop any use of "BREAK". Also, the "BASIC" is now saved as "BYTES"!

(5). Putting Machine Code in REMS.

A good place to put machine code is in a REM statement in a BASIC program. Thus, you could have:-

```
10 REM Copyright Statement
20 REM hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
```

The "h's" in the REM can then be POKED with machine code numbers. Another place for the code is in the printer buffer (23296-23551), however this only applies in 48K mode, since in +3 Basic this area is occupied by extra System Variables - see later.
A sure indication of the presence of machine code in a REM statement as above is that the program loads a first Basic part and then a HEADERLESS block. The machine code to load this headerless block MUST be in the first Basic part and will probably be in a REM statement as detailed above. The line numbers will also usually probably be poked to cause a crash. For this reason these sorts of programs are best loaded using the "MCbasldr" and "BASCONV" programs as detailed later. The code can then be examined with the KD1 disassembler.

(6). Loading and Stopping 'Unmergeable' Basic Loaders.

As detailed above, with most Basic Loaders which contain machine code - if you try to stop them auto-running by loading using the "MERGE" command, they usually crash. They CAN be made to load using the "MCbasldr" and "BASCONV" programs supplied and the code then examined using the KD1 disassembler.

(7). Other Protection Methods.

(a). Altering System Variables.

Some programs load different values than normal into the System Variables. Popular locations are DFSZ (see earlier), and also ERR-SP. This is the return address when an error occurs. It usually contains the numbers 84 (L.O.B) and 255 (H.O.B). These are stored at 23613,23614 - the address of ERR-SP. This gives the address 65364 (i.e.84 + 256*255) and this gives the error return address as 4867 - the normal return to Basic. Check with:- PRINT PEEK 23613 + 256*PEEK 23614 - this should give 65364. Also, PRINT PEEK 65364 + 256*PEEK 65365 - this should give 4867. (i.e. 3 + 256*19 = 4867). The usual ruse is to POKE ERR-SP with the address 23728 - where 23728,23729 are 2 unused System Variable locations, normally containing 0. Thus, POKE 23728,176:POKE 23729,92 is usually used i.e. the address is 176+256*92 = 23728. Since these locations are 0, any error which would normally print an error message i.e. D-BREAK CONT REPEATS, Out of Memory, etc, would direct the Z80 to address 0 i.e. the computer would reset (as with R.USR 0 - remember?). Another System Variable used is FRAMES. This is 3 locations i.e. 23672,23673,23674. It acts as a sort of "clock" - its value is incremented every 20 milliseconds by the Z80. Some programs load values into FRAMES, then recheck that they are within certain limits when the program has loaded. If not it usually resets the computer. Many of the latest programs overwrite the System Variables - take care if this is the case - since trying to return to Basic will usually crash the computer.

(b). Checking the Screen.

Several programs check the SCREEN$ picture loaded, and if it has been altered, reset the computer. These are in the minority however, and in most examples the picture can be left out to save disc space.

(c). Moving Code Around.

Many programs, after loading, move parts of the code to different areas of memory - using the LDIR command (see earlier).

(d). Scramblers.

Some programs, notably the U.S Gold series, employ SCRAMBLERS i.e. the main code, when loaded in, if checked would seem to be rubbish; but there is usually a small block of code somewhere which rearranges or UNSCRAMBLES the whole code - transforming it back to normal code. The problem is finding it! The Speedlock protection system as used by many software houses including U.S. Gold, Ocean etc employ a very complex scrambler. These programs are transferred to disc using the Speedlock decoders in the SP7 beginner book. Also, the Alkatraz protected programs use scramblers and are also best transferred using the Alkatraz decoders in the SP7 Beginner book, as these scramblers may employ up to ONE HUNDRED separate pieces of code all of which must be deciphered to reveal the main program code, which clearly would take many hours.

(e). Changing Line Numbers.

The +3 can only accomodate line numbers up to 9999, so POKEing a line number to a value greater than this will cause the computer to crash when MERGE"" or "BREAK" is used, especially when used in conjunction with POKE 23659,0 (see earlier). Reset the +3 and type in:-

```
10 POKE 23659,0:LOAD"T:":LOAD""CODE
20 REM
30 FOR A=23755 TO 65535:PRINT A,:PRINT PEEK A:NEXT A
```

Type RUN 30 - you will see that the L.O.B of line 20 occurs at 23791. Delete line 30 - type PRINT PEEK 23797, this should give 225. This is the usual terminator for a Basic program - if this is also POKEd to 0, a crash will result as above. Try it - type POKE 23791,255:POKE 23797,0. This alters line 20 to 65300! Now save to tape by typing save "t:":save"test"line 0. Reset and load in the usual way - the program goes into tape load mode. Try pressing "BREAK" - the computer crashes! Also try loading using merge"" where the same effect occurs! NOTE:- The REM can be used to hold some machine code as described earlier.

THE PROGRAMS SUPPLIED AND WHAT THEY DO.

We have supplied you with a large suite of programs on your SP7 disc to help you with your program investigation and transfer. These are listed below:-

LOADING INSTRUCTIONS FOR PROGRAMS.

To load a program from your SP7 disc, proceed as follows. RESET the +3 by pressing the reset button on the left side of the computer. Place the SP7 disc in the drive and, at the Main Menu, press the ENTER key. A special Menu type program will load displaying the program names and numbers. The message "Enter your choice:- " also appears. To load a specific program, it is then only necessary to press the number key of the program required i.e. 1 to 27, followed by pressing ENTER. The selected program will load and run automatically. The programs supplied are:-

HEADER READER.

Probably the most useful program. To load it, type 8 then ENTER. This tells you the composition of the program you wish to transfer. (see later for fuller details). It gives the Program Name, Program Type, Start Address, Length, etc.

HEADERLESS BLOCK LENGTH READER.

To load, type 7 then (ENTER). This will tell you the length of any program without a Header i.e. the length of any headerless block e.g. for a Header, it will find 17 bytes. (useful for detecting false Headers, which are often not 17 bytes).

MCbasldr.

To load, type 11 then (ENTER). It is a "Machine Coded Basic Loader" i.e. it loads (via machine code) those "Basic Loaders" which contain machine code and crash when loaded via MERGE "". The program puts the code (Basic and machine code) at address 32000, where it can then be examined with your next program:-

BASCONV.

To load, press key 3 then ENTER. It is a BASIC CONVERTER, it reads code from (3) above - "lists" any Basic and gives the address of the 1st byte after any REMS (which may contain machine code). It is extremely versatile in that it will also make visible any Basic lines which have been rendered invisible (see earlier); it also gives the TRUE value for any numeric values found, since these may have been altered to give false values. The TRUE numeric value is the one in brackets. NOTE:- Pressing "p" gives a listing to a printer but in +3 mode (i.e.128K) only. Any machine code in the REMS may be examined with:-

KD1 DISASSEMBLER.

To load, press key 9 then ENTER. It is an important addition to your SP7, since you now have the facility to load in machine code, and get a full disassembly of all the instructions - see later for full instructions.

CODE-STOP.

To load, press key 5 then ENTER. It is a machine code program which, when added to the end of an auto-running code block, stops it from running and returns to Basic, leaving the code in memory.

MOVEUP, MOVEDOWN, BLOCK MOVE.

To load, press keys 13 or 12 or 4 then ENTER. As suggested - these programs move blocks of code around in memory.

SPEEDLOCK DECODER SD1, SD1 Trans.

To load press key 14 or 15 then ENTER. SD1 is a useful Speedlock decoder program which enables the EASY transfer to +3 Drive (and a backup to tape, if required) of the "Pulsed Leader" type of program. SD1 Trans transfers the tape from SD1 to your drive.

CONVERSION UTILITY CU6; TRANS128; TRANS48; CU7.

To load, press keys 24, 25, 26 or 27 then ENTER. CU6 and CU7 are new additions to SP7, which allow the transfer of FULL single load 128K programs to disc. They make a special tape which is then transferred to disc using Trans128 or Trans48K. Thus, programs such as "OPERATION WOLF 128K", STARGLIDER 2 128K", "HIGH STEEL 128K" and "JAWS 128K" may now be readily transferred to disc.

SPEEDLOCK DECODER SD234; SLOCKLDR.

To load, press key 6 or 16 then ENTER. SD234 stands for "Speedlock Decoder for types 2,3 and 4"; SLOCKLDR stands for "Speedlock Loader" - it is an automatic loader for SD234 transferred programs.

SPEEDLOCK DECODER SD5; SD5LDR.

To load, press 17 or 18 (ENTER). This decodes the latest type 5 Speedlock programs using a "software emulator" for the decoding. SD5LDR is a loader for SD5 transferred programs.

ALKATRAZ DECODER AD1; AD1 TRANS; LOADER.

To load, press key 1 or 2 or 10 then ENTER. With AD1 you can transfer programs protected by the Alkatraz system. AD1 TRANS effects simple transfer of the tape obtained from AD1; LOADER is an auto loader program for same.

ALKATRAZ DECODERS AD2; AD3; TRANSFER.

To load press keys 19, 22 or 20 then ENTER. AD2 and AD3 are our NEW Alkatraz decoders, similar to AD1. TRANSFER transfers the tapes from these AND FB1 (see below) to disc.

FIREBIRD DECODER FB1.

To load press keys 23 then ENTER. This NEW program will transfer the multi-block Firebird programs which load with many very small blocks, to disc.

DISKCAT - A FULL DISC CATALOGUE PROGRAM.

To load press keys 21 then ENTER. With this superb program you can now get a FULL catalogue VERY QUICKLY of ALL the files on your discs.

OBJECTS AND PROBLEMS IN +3 DRIVE TRANSFERS.

The programs encountered fall into 3 main types:-

(1). BASIC PROGRAMS.

If unprotected - these are simply transferred to your +3 drive using:- SAVE "name", OR:-Also use SAVE"NAME" Line Number to auto-run. If protected, e.g. auto-running; stop with MERGE (see earlier), and transfer as above.

(2). UNPROTECTED MACHINE CODE PROGRAMS.

To test for protection, proceed as follows:- Firstly, try loading the Basic Loader (the FIRST part) using MERGE "". If unprotected, it will stop any auto-run, allowing the program to be LISTed as normal. If this causes a crash, the program is quite probably protected. Also, if the game code can be loaded and run separately, using the Execution Address (from the Basic Loader), it is probably unprotected. With these programs, the code can usually be easily relocated if required, and transferred to +3 Drive.

(3). PROTECTED MACHINE CODE PROGRAMS.

These are in the majority nowadays and are of course the most difficult - needing a small knowledge of machine code to transfer them. Here, the Basic Loader(s) MUST be loaded checked and listed with MCBasldr and BASCONV. Remember, EVERY program, protected or not, MUST start with a normal speed Basic Loader.

With types (2) and (3), the main object is to get the code into memory (i.e. it must be prevented from auto-running), where it can be relocated as required, then transferred to +3 Drive.

PLEASE NOTE:- All transfer operations may be done in +3 Basic, EXCEPT those which may involve putting any code in the Printer Buffer area (see earlier). Any such transfers MUST be done in 48K Mode - if attempted in +3 Basic a "crash" will almost invariably occur, since in +3 Basic this area is occupied by extra System Variables which are used by the computer. This is not the case in 48K Mode. Also, it is wise to try and load a program in +3 Basic BEFORE attempting to transfer it to disc, to ensure it is compatible with +3 Basic - some of the older programs are NOT compatible with the +3!

OBJECTIVES AND GENERAL TRANSFER METHOD.

The object is to isolate any code blocks from the program, and to find their Start Address and Code Length, and if possible the Execute Address to activate the program. The blocks may then be saved to disc together with a simple Basic program to reload them A general method would be:-

(1). PLAY in the game tape in +3 Basic - make sure it loads and runs. This may seem rather obvious, but if a game will not load in +3 Basic it is unlikely you will succeed in transferring it to disc.

(2). Rewind the tape. Load the Header Reader program and PLAY the tape through again - note all the details of each block and if there are any headerless blocks. NOTE:- Headerless blocks will, of course, NOT be read by the Header Reader program.

(3). Reset the +3. Try to load the first Basic part on the tape and stop any autorun by typing load"t:":merge"" then press ENTER and PLAY in the Basic. If it loads O.K. list the Basic and note the details such as BORDER, INK and PAPER colours, any CLEAR statements and in particular any LOAD""CODE and RANDOMIZE USR statements. However, if the program crashes:-

(4). Reset the +3 and load the Basic using the MCbasldr and BASCONV programs on your SP7 disc as follows. Press ENTER to load the SP7 Menu, and load the MCbasldr program by pressing key 11 and ENTER. For "Number of bytes to enter" use the value obtained earlier when using the Header Reader on the first Basic part. Press any key then PLAY in the first Basic part. Press any key to reset the +3, then load the BASCONV program by pressing ENTER then key 3 and ENTER.. For Screen or Printout select Screen by pressing the "s" key; for "Start Address of Code" type in 23755 and press ENTER. For "Number of bytes to Analyse" type in the same value as used for the MCbasldr program. BASCONV will list the Basic. NOTE:- If the listing seems "odd", the original program may have been compiled on a Spectrum with a microdrive attached, in which case PROG will be 23813 (see earlier). In this case, at the end of the listing, press the "down" cursor key, then

ENTER and type GOTO 30 to rerun the BASCONV program. This time for "Start Address of Code" type in 23813. The listing should now be more sensible. Note the values of any CLEAR's, POKE's, LOAD""CODE's and RANDOMIZE USR statements. If there is a REM which contains machine code disassemble with KD1 as follows:- Type CLEAR 30000 then press ENTER then type NEW and press ENTER. This resets the +3 but leaves the code intact. Load the SP7 Menu by pressing ENTER, then load KD1 by pressing key "9" then ENTER. Follow the instructions for using the KD1 - start the disassembly from 32000. Remember, you are looking for any or all of the following:-

```
SCF
LD A,n
LD IX,nn
LD DE,nn
CALL LOAD ROUTINE
JUMP TO START
```

The number in IX gives the Start Address of the code, that in DE gives the code length.

EXAMPLES OF PROGRAM INVESTIGATION.

We shall now further illustrate the above program transfer methods using the following programs:-

MANIC MINER. (SOFTWARE PROJECTS).

We will start with this program which is totally unprotected. Using the Header Reader program gave the following details:-

Tape Count	File Name	Type	Details
10 - 13	ManicMiner	BASIC	T.L.=69, P.L.=69, A.R. Line 1
14 - 18	MM1	M/C	S.A.=22528, C.L.=768
19 - 87	MM2	M/C	S.A.=32768, C.L.=32768

Where T.L.=Total Length; P.L.=Program Length; S.A.=Start Address; C.L.=Code Length; A.R. =Auto-Run Line No; M/C=Machine Code.

Following the method described earlier, rewind the tape. Reset the +3 then select +3 Basic and type load"t:":merge"" press ENTER and PLAY in the first Basic part. List the Basic. This gave:-

```
10 CLEAR 30000
20 PAPER 0:INK 0:CLS:LOAD""CODE:LOAD""CODE
30 RANDOMIZE USR 33792
```

All we need to do is rewrite the Basic thus:-

```
10 CLEAR 30000
20 PAPER 0:INK 0:CLS:LOAD"MM2"CODE
30 PAUSE 100
40 RANDOMIZE USR 33792
```

We can ignore the first LOAD""CODE since this simply gives the screen picture. We MUST name the second block to load it from +3 disc - hence the LOAD"MM2"CODE. The pause instruction in line 30 gives the disc drive time to stop! Save to disc by typing save"miner"line 10. Next, we must load in the LAST block of code, as this is the main game code. Type clear 30000:load""code and PLAY in the last block. When loaded, save to disc by typing save"MM2"code 32768,32767. The game reloads using load"miner".
This is a typical example of an unprotected program - the Basic could be MERGEd and listed, and all the necessary information was present. Our next example is:-

WAY OF THE EXPLODING FIST. (MELBOURNE HOUSE).

Press RESET and load the SP7 Menu. Press 8 (ENTER) to load the HReader program. PLAY in the "FIST" tape from the start until all the headers have been read. This gave:-

Tape Count	File Name	Type	Details
6 - 9	FIST	BASIC	T.L.=319,P.L.=319,A.R. Line 10
10 - 23	t	M/C	S.A.=32768, C.L.=6912 (SCREEN$)
24 - 187	k	M/C	S.A.=24576, C.L.=38912

Where:- T.L.= Total Length, P.L.= Program Length, S.A.= Start Address, C.L.= Code Length, A.R.= Auto-Run Line No., M/C = Machine Code.
Reset. Select +3 Basic and type load"t:":merge"" and PLAY in the first Basic part. The program crashed i.e. it was probably protected and we must use MCbasldr and BASCONV. Press RESET; load the SP7 Menu program; load "MCbasldr" by pressing 11 (ENTER). Type in 319 (ENTER) for code length to load. Following the screen instructions, press any key, then PLAY in the FIST Basic Loader. After, press any key (NOT ENTER) to NEW the MCbasldr the FIST loader is still intact at address 32000. To examine this, reload the SP7 Menu; type 3 (ENTER) to load "BASCONV". "S" gives screen output. Press "P" for printer output which is automatic here in +3 Basic (a printout is NOT available in 48K Mode). Next, type 23755 (ENTER) for the code Start Address, then type 319 (ENTER) for length of code to analyse. A Basic listing is given which seemed odd. When STOP appears at the bottom of the screen, retry by pressing the "down arrow" key then (ENTER). Type GOTO 30 (ENTER) - press "s" for screen and then type in 23813 (ENTER) for the Start Address, then type 319 (ENTER) for the code length to analyse. The Basic listing is now more sensible! This is because the program originated on a computer with a microdrive,

which moved PROG up to 23813. Note the following, BORDER 6: PAPER 6: INK 6: CLEAR 24576: PRINT USR 34816. NOTE:- The values shown for CLEAR and PRINT USR were FALSE - the true values were those listed by BASCONV (in brackets). Check for protection thus:- Press RESET and select +3 Basic - type load"t:"(ENTER). Type BORDER 6: PAPER 6: INK 6: CLEAR 24575 (ENTER). Load code "k" using load""code(ENTER). When loaded, type RANDOMIZE USR 34816 (ENTER) - the game ran i.e the code was NOT protected. Conversion to +3 drive is thus just a matter of loading the "k" code with a suitable Basic Loader. Press the RESET button. Select +3 Basic, and type in the following Basic Loader:-

```
10 BORDER 6:PAPER 6:INK 6:CLEAR 24575:LOAD"fist"CODE:PAUSE 100:RANDOMIZE USR 34816
```

Press ENTER. Save to disc with:- SAVE"FISTLDR" LINE 10 (ENTER). Press RESET. We must now load the game code "k" into the Spectrum's memory from tape, prior to saving it to disc. Select +3 Basic. Type CLEAR 24575 (ENTER) - keeps the code above RAMTOP. For a load from TAPE not disc, type load"t:" then press ENTER. Type load""code, press ENTER, and PLAY in the "k" game code. When loaded, save the game code to disc by typing:-
save"fist"code 24576,38912 (ENTER). The game can be reloaded with:- load"FISTLDR".

A different type of difficulty is encountered in:-

JETPAC. (ULTIMATE).

Following the usual method, PLAYing the Jetpac tape using the Header Reader program gave the following details:-

Tape Count	Filename	Type	Details
10 - 13	JETPAC	Basic	T.L.=376; A.R. Line 1
14 - 30	SCREEN	M/C	S.A.=16384; C.L.=6912
31 - 52	0	M/C	S.A.=24576; C.L.=8192
53 - 55	1	M/C	S.A.=23424; C.L.=15
56 - 59	2	M/C	S.A.=23728; C.L.=1
60 - 64	3	M/C	S.A.=23672; C.L.=2

As can be seen, blocks 1,2 and 3 will cause problems since they load in the Printer Buffer and System Variables areas. Following the usual method, RESET the +3, select +3 Basic and type load"t:":merge"" and PLAY in the first Basic part. This gave Line 1 which contained a CLEAR 24575 and a RANDOMIZE USR 24576. Delete Line 1. Type clear 24575:load""code 40000 and PLAY in the SCREEN block. Save to disc with save"SCR"code 40000,6912. Type load""code and PLAY in block 0. Save to disc with save"0"code 24576, 8192. Type load""code 40000 and PLAY in block 1. Save to disc with save"1"code 40000,15.

The problem now are blocks 2 and 3. However, these can be POKEd in as they are only 1 and 2 bytes each. The technique for block 3 is to load it to a convenient address e.g. 36864, then add a small piece of LDIR "mover" code to the end, which will move the block to its correct address then jump to the execute address AFTER all disc drive operations are over. For this we shall employ another short but very useful program which we shall call "INPUT". This is a Basic line of the form:-

```
10 FOR A=40000 TO 40030:INPUT B:POKE A,B:NEXT A
```

Type CLEAR 65535 then load"a:":load"1"code 36864 and load the block from disc. Type in the above INPUT line for the range 36880 to 36893 and enter the following numbers:- 33, 0,144,17,128,91,1,16,0,237,176,195,0,96. Save to disc with save"new1cd"code 36864,30. Reset the +3. Select +3 Basic. Rewind the Jetpac tape, type load"t:":merge"" and PLAY in the first Basic part. Edit Line 1 to read:-

```
(First part)
1 ---------- :LOAD"SCR"SCREEN$:INK 0:PAPER 0:PRINT AT 5,0;"LOAD"0"CODE:POKE 23728, 233
 :POKE 23672,67:POKE 23673,131:LOAD"NEW1CD"CODE 36864:PAUSE 100:RANDOMIZE USR 36880
```

Save to disc with save"JETPAC"LINE 1. Reload with LOAD"JETPAC".
NOTE:- The code at 36880 is:-

```
LD HL,36864
LD DE,23424
LD BC,16
LDIR
JP 24576
```

i.e. the 16 bytes at 36864 are moved to 23424. The program then jumps to the start address of 24576.
NOTE:- Wherever a Basic loader crashes on trying to load it using the merge"" command, this indicates the program is probably protected, and the Basic must be loaded using MCbasldr and BASCONV as detailed earlier.

HORACE AND THE SPIDERS. (MELBOURNE HOUSE).

Again, using the Header Reader to analyse the program we find:-

Tape Count	Filename	Type	Details
11 - 13	spiders	Basic	T.L.=15; A.R. Line 10
14 - 18	SPIDERS	Basic	T.L.=256;A.R. Line 5
19 - 32	s	SCREEN$	S.A.=16384; C.L.=6912
33 - 52	h	M/C	S.A.=24576; C.L.=8191
53 - 56	hi	M/C	S.A.=32768; C.L.=816

Type load"t:":merge"" and PLAY in the first Basic. This showed:-

```
10 LOAD"SPIDERS"
```

i.e. it simply loads in the second Basic. Reset. Type load"t:":merge"" and PLAY in t
second Basic. Listing showed a CLEAR 24575, R.USR 24576, + ONE LOAD"h"CODE command. Th
seemed to infer that the last code block must be loaded by the "h" code block, which
indeed the case. The header in block "h1" was thus false. First we load the "s" code
typing clear 24575:load""code 40000 and PLAY in the "s" block. Save to disc wi
save"s"code 40000,6912. (This CAN be omitted if desired, as the screen is not usual.
checked in these old programs). Type load""code and PLAY in the "h" block. Save to di
with save"h"code 24576,8191. Delete the Basic lines and use the PEEK LINE progra
examine the range 24576 to 65535. This showed:-

Address	Contents	Mnemonic	
24576	195	JP	
24577	224	L.O.B	=(127X256)+224
24578	127	H.O.B	= 32736

i.e. the instruction is JUMP to address 32736.
Use PEEK LINE now to examine the area from 32736 to 32764. This gave:-

Location	Number	Mnemonic
32736	221	
32737	33	LD IX, (0 X 256)+0 = 0 = S.A.
32738	0	
32739	0	
32740	17	
32741	16	LD DE, (0 X 256)+16 = 16 = C.L.
32742	0	
32743	175	XOR A = LD A,0 = Signal Header Loading
32744	55	SCF = Signal LOAD
32745	205	
32746	86	CALL ROM Load Routine
32747	5	
32748	221	
32749	33	
32750	208	LD IX, (92X256)+208 = 23760 = S.A.
32751	92	

```
32752      17
32753      13        LD DE, (3X256)+13 = 781 = C.L.
32754       3

32755      62        LD A,255 = Signal Code Block Loading
32756     255

32757      55        SCF = Signal LOAD

32758     205
32759      86        CALL ROM Load Routine
32760       5

32761     243        Disable Interrupts (not important)

32762     195
32763     199        JP (95X256)+199 = 24519 = Execute Address
32764      95
```

We have covered this program in detail to clarify the clues to be looked for. Thus, this code loads the last block "h1" to address 23760 the length being 781 bytes. It then jumps to the Execute Address of 24519 to start the program. The problem here is that the loading address is so low (23760, whereas PROG is only 23755) that the code would overwrite any Basic we use to load it. To overcome this, we load "h1" to its usual address of 32768 then add a piece of mover code to the h1 code which will move it back to the correct address, then jump to the execute address. Type load""code and PLAY in the h1 block. Use the INPUT line for the range 33550 to 33563 and enter the following numbers:- 33,8,129,17,208,92,1,13,3,237,176,195,199,95. This mover code is:-

```
LD HL,32768 - Start of block to move.
LD DE,23760 - Destination of block.
LD BC,781   - No. of bytes to move.
LDIR        - Move the block.
JP 24519    - Jump to Execute Address.
```

Save the new code to disc with save"h1"code 32768,796. Reset the +3. Select +3 Basic and type load"t:"merge"" and PLAY in the second Basic part. Edit the Basic to read:-

```
                Rest of program
50 LOAD"h"CODE:LOAD"h1"CODE
55 PAUSE 100
60 RANDOMIZE USR 33550
```

Save to disc with save"horace"line 5. Reload with load"horace".

ZOLYX. (Y/S).

The protection on this program is a special type used by Players. It is indicated b short first Basic, followed by a short code block. The rest of the program then lo with blue/black loading stripes. Using the Header Reader gave:-

Tape Count	Filename	Type	Details
6 - 9	Zolyx	Basic	T.L.=142; Auto Run Line 0.
10 - 14	loader	M/C	S.A.=65024; C.L.=512.
15 -	Special Loader		

Reset. Type load"t:":merge"" and PLAY in the first Basic part. This gave:-

```
10 PAPER 0:BORDER 0:INK 0:CLEAR 32767
20 LOAD""CODE 65024
30 RANDOMIZE USR 65024
40 POKE 23418,84
50 SAVE"Zolyx"LINE 0
60 LOAD"Mast0"
```

Please note this Basic - wherever you see a similar Basic you will know the Play protection system is being used. Type clear 27000:load""code and PLAY in the "load code block. Delete the Basic, and use PEEK LINE in the range 65024 to 65535 to exam the "loader" code. At 65024 we see a DI instruction; then a scrambler with HL contain the start which = (254X256)+128 = 65152. Also, at 65041 we see a JP FE30 instruct i.e. jump to 65152. Thus, by POKEing 65041,251 and 65042,201 then RANDOMIZE USR 65 this activates the scrambler and decodes the code. Use PEEK LINE in the range 65152 65535 to examine the game loader code. The important points are the LD SP,24576 at 65 and the JP 45056 at 65213. Thus, type POKE 65153,0:POKE 65154,0:POKE 65155,0:POKE 652 251:POKE 65214,201. This removes the LD SP command and forces a return to Basic when game has loaded. Type RANDOMIZE USR 65152 and PLAY in the rest of the tape. When load the program will return to Basic. We now need to know where to save the code. At 65 there is a block move (LDDR) which moves the code up from 65023 to 65535, the len being 20481. Thus, the code to save is from 65535-20481 = 45054 to 65535. Save to d with save"z"code 45000,20535. Reset and write the following Basic loader:-

```
10 CLEAR 24500:LOAD"Z"CODE:PAUSE 100:RANDOMIZE USR 45056
```

Save to disc with save"Zolyx"Line 10. Reload with load"Zolyx".

NOTE:- It is necessary to remove the LD SP (load stack pointer) command since using t as well as a CLEAR can cause a crash when returning to Basic. The 251 and 201 put commands EI and RET or Enable Interrupts and Return at the end of the code forcin return to Basic. Most other Players protected programs employ a similar method and be transferred to disc in a similar way.

APACHE GOLD. (CRASH FREE TAPE - FEB '90).
Using the Header Reader program gave:-

Tape Count	Filename	Type	Details
3 - 5	APACHE	Basic	T.L.=143; A.Run Line 0
6 - 20	screen	M/C	S.A.=32768; C.L.=6912
21 - 22	apache	Basic	T.L.=54; A.Run Line 1.
23 - 106	Headerless block.		

Typing load"t:":merge"" and PLAYing the first Basic gave:-

```
  10 REM ag
  15 BORDER 2:INK 2:PAPER 2:BRIGHT 0:CLS
  20 LOAD""SCREEN$
  30 PRINT AT 0,0;
  40 INK 1:PAPER 1
  50 BORDER 7:LOAD""
9999 SAVE"APACHE"LINE 0
```

The second block is the screen picture which can be omitted. RESET and type load"t:":merge"" then press ENTER and PLAY in the second Basic part. The program crashed. It was thus reloaded using MCbasldr and BASCONV in the usual way. This showed:-

```
10 RANDOMIZE USR 23778
20 REM (Machine Code)
```

i.e. the machine code to load the last main headerless block was contained in the REM statement as described earlier. Using PEEK LINE showed this to be:-

```
DI
LD SP,23973
LD HL,23778
PUSH HL
LD (23613),SP
LD HL,34104
PUSH HL
LD DE,41536
LD IX,24000
LD A,255
SCF
JP 1366
```

From this we see that the S.A. = 24000; the C.L. = 41536 - the E.A. is obtained from LAST value PUSHed on the stacl, i.e. 34104. We now need to use a piece of machine c to load the final headerless block, then save to disc. Reset the +3. Use INPUT in range 40000 to 40015, RUN it and enter the following numbers:- 243,62,255,55,221, 192,93,17,64,162,205,86,5,251,201. Save to disc with save"apldr"code 40000,16. Reset type CLEAR 23999 then type in AS A DIRECT COMMAND:-

LOAD"APLDR"CODE 16384:PAUSE 100:RANDOMIZE USR 16384

This loads the loader code to the screen, where it will not be overwritten when the code block is loaded. Press ENTER and PLAY in the last headerless block. Save to with save"indian"code 24000,41536. Reset and type in this Basic loader:-

10 CLEAR VAL"23999":LOAD"INDIAN"CODE:PAUSE VAL"100":RANDOMIZE USR VAL"34104"

Note the use of VAL, this is because the CLEAR statement is so low, allowing little for any Basic. Save to disc with save"gold"line 10. Reload with load"gold". This proc illustrates the use of headerless blocks as a means of protection.

BALLBREAKER 2. (Y/S FREE TAPE).

This program also illustrates the use of machine code in a REM statement - here causes the Basic to crash when loaded using merge"". Using the Header Reader proc showed:-

Tape Count	Filename	Type	Details
5 - 10	BB2	Basic	T.L.=1107; A.Run Line 0.
11 - end	Headerless block.		

Again, the headerless block AFTER the first Basic MUST mean machine code in a REM in first Basic part. Load the Basic using MCbasldr and BASCONV in the usual for BASCONV enter 23813 for the Start Address of the Code, and 1107 for the Length code to analyse. This showed the machine code in a REM statement, starting at 32 Press BREAK then the down cursor key and ENTER to go into +3 Basic. Type clear 30 new then load the SP7 Menu and load the KD1 disassembler. Disassemble from address 7 Hex or 32084 Decimal. This showed the following important points:-

Address (Decimal)	Address (Hex)	Mnemonics	Details
32084	7D54	JR 7D6C	Jump to 7D6C
32108	7D6C	LD SP,0	Set Stack Pointer
32111	7D6F	LD BC,0197	BC holds no. bytes to move

32114	7D72	LD HL,(5C53)	Set HL to PROG.
32117	7D75	LD DE,0081	DE holds offset = 129
32120	7D78	ADD HL,DE	Add 129 to PROG
32121	7D79	LD DE,FE2D	Destination = 65069
32124	7D7C	LDIR	Move bytes to FE2D
32126	7D7E	JP FE2D	Jump to Start Address

The net result is thus to move the 407 bytes from PROG + 129 which here = 32000 + 129 =32129 = 7D81 Hex to FE2D = 65069, then jump to the E.A. of 65069.

The main points from 32129 are:-

Address (Decimal)	Address (Hex)	Mnemonics	Details
32131	7D83	LD DE,1161	C.L.=4449
32134	7D86	LD IX,8000	S.A.=32768
32189	7D8D	LD DE,86ED	C.L.=34541
32192	7DC0	LD IX,6DD6	S.A.=28118
32215	7DD7	JP 8000	E.A.=32768

i.e. this gives the Start Address and Code Lengths of the last 2 headerless blocks, and the Execute Address. Next, type in the following POKES:-

```
POKE 32108,0
POKE 32109,0    - Removes the LD SP instruction.
POKE 32110,0

POKE 32114,33
POKE 32115,0    - Gives effective PROG = 32000
POKE 32116,125

POKE 32122,0    - Moves code to 36864
POKE 32123,144

POKE 32126,251  - Enable interrupts
POKE 32127,201  - RETurn to Basic
```

Type RANDOMIZE USR 32111, press ENTER - this moves the code to 36864. Save to disc with save"ld"code 36864,407. We now need to reload this code to the correct address of 65069. Reset. Type CLEAR 27000:load"ld"code 65069. Type POKE 65155,251:POKE 65156,201. This makes the program return to Basic after loading the rest of the code. Type RANDOMIZE USR 65069 and PLAY in the rest of the tape. Save to disc with save"bb"code 28118,34541. Reset and type in the following Basic:-

```
10 CLEAR 26998:LOAD"BB"CODE:PAUSE 100:RANDOMIZE USR 32768
```

Save to disc with save"ballbr2"line 10. Reload with load"ballbr2".

JET SKI SIMULATOR. (CODEMASTERS).

This program illustrates the splitting of one large code block into two parts, and transfer of the lower part to the screen from where it is moved back to its normal p using an LDIR block move command. From the Header Reader we get:-

Tape Count	Filename	Type	Details
7 - 10	Jet	Basic	T.L.=329; A.Run Line 10
11 - end	Headerless blocks		

Again the presence of code in a Basic loader is indicated. Type load"t:":merge"" PLAY in the first Basic part. This gave:-

```
10 CLEAR
20 FOR J=24576 TO 24594
30 READ A
40 POKE J,A
50 NEXT J
60 RANDOMIZE USR 24576
70 DATA 17,0,1,221,33,0,255,62,255,55,205,86,5,212,0,0,195,0,255
```

This example is different in that the machine code is POKEd into place by the B program. Change the last 3 data items so that Line 70 reads:-

```
70 DATA 17,0,1,221,33,0,255,62,255,55,205,86,5,212,0,0,251,201,0
```

This will force a return to Basic at the end of the program. RUN the program and r the tape. Delete the Basic. Note that from the above data the block loaded will hav S.A.=65280 and C.L.=256. Use PEEK LINE in the range 65280 to 65535 and we find:-

Address	Mnemonics
65302	LD DE,6912
65305	LD IX,16384
65318	LD DE,41529
65321	LD IX,23750
65334	JP 23750

We will now load the main code in 256 bytes higher than normal, then split it into 2 parts - the lower part being moved down to its correct place by a piece of mover code which we add to the end of the lower piece and load to the screen area. Reset the +3. Type in:-

```
10 CLS:FOR A=16384 TO 16398:INPUT B:POKE A,B:NEXT A
20 SAVE"jsld"CODE 16384,15
```

RUN the program and enter the following numbers:-

62,255,55,221,33,198,93,17,57,162,205,86,5,251,201. When the last number is entered, the program "jsld" is saved to disc. Delete the Basic, place the tape at the start of the main block. Type in AS A DIRECT COMMAND:- CLEAR 24005:LOAD"JSLD"CODE:PAUSE 100:RANDOMIZE USR 16384 and PLAY in the rest of the tape. Save to disc in 2 parts i.e. save"juplow"code 24006,2000 then save"juphi"code 26006,39529. We now need to tag the mover code onto the start of the "juplow" code. Do this as follows:- Reset the +3. Use INPUT for the range 40000 to 40013. RUN it and enter the following numbers:- 33,14,64, 17,198,92,1,208,7,237,176,195,198,92. This is in machine code:-

```
LD HL,16398
LD DE,23750
LD BC,2000
LDIR
JP 23750
```

Reload the code from disc with load"juplow"code 40014. Save this including the mover code with save"jetmvd"code 40000,2014. Reset the +3 and write the following Basic loader

```
10 CLEAR VAL"65535":LOAD"JUPHI"CODE VAL"25750":LOAD"JETMVD"CODE VAL"16384":PAUSE VAL
   "100":RANDOMIZE USR VAL"16384"
```

Save to disc with save"jetsim"line 10. Reload with load"jetsim".

FRED 48K. (ORIGINAL).

This is another unusual program in that it loads as CODE and as such it is impossible to stop it running using the merge"" command. The technique is to load the code to a higher address, then use PEEK LINE to look for the required RANDOMIZE USR number in the Basic part of the code. From the Header Reader we find:-

Tape Count	Filename	Type	Details
1 - 2	FRED	Basic	T.L.=426; A.Run Line 10
3 - 38	scr	M/C	S.A.=16384; C.L.=30766

We will load the 2nd (code) block 10000 bytes higher than normal to stop it running. Reset the +3. Type load"t:":load""code 26384 then PLAY in the 2nd block. PEEK LINE to examine the range from 33755 to 65535. NOTE now the start of the Basic is now = 23755+10000 = 33755. Looking at the Basic tokens gives the RANDOMIZE USR 3 command - this is the Execute Address. The code now occupies 26384 - 57150 in memory must now add a piece of mover code to the end of the main code in the usual way delete the PEEK LINE, and use INPUT in the range 57151-57164. RUN it and enter following numbers:- 33,16,103,17,0,64,1,46,120,237,176,195,77,118. In code this is:-

```
LD HL,26384
LD DE,16384
LD BC,30766
LDIR
JP 30285
```

Save the whole code to disc with save"fmv"code 26384,30781. NOTE:- This cannot be reloaded in +3 Basic since it will crash the computer by overloading System Variables area (see earlier). We must also add another piece of code which change the +3 into 48K Basic mode. The code will then load and auto run as normal. is done thus:- Reset the +3. Write the following Basic:-

```
10 REM 48 SPACES
20 LOAD"FMV"CODE:PAUSE 100:RANDOMIZE USR 23762
30 RANDOMIZE USR 57151
40 FOR A=23760 TO 23797:INPUT B:POKE A,B:NEXT A
```

Type RUN 40 and enter the following numbers:- 13,13,243,42,61,92,35,54,19,43,54,3 54,27,43,54,118,43,54,0,43,54,31,249,1,253,127,62,48,0,0,253,203,1,166,201,56. This POKE the machine code into the REM statement in Line 10. Delete Line 40. Save to with save"fred"line 10. Reload with load"fred".

We hope the above examples will clearly illustrate the methods to transfer a variety your programs to disc, and you should now be able to use these with some modificati to transfer most other programs.

"SP7 TAPE TO +3 UTILITY 128K"

INSTRUCTIONS FOR USE

MEMORY MANAGEMENT IN THE SPECTRUM 128K

The Z80 processor in the Spectrum can only address 64K of memory at once. However, in the new 128K Spectrums, the computer contains 128K of RAM and 32K of ROM. The extra memory can be slotted in and out of the normal 64K at will, by a system called PAGING. This is achieved by setting the right bits in an Input/Output port - address 32765 decimal or 7FFDH was chosen for the new Spectrums. The memory map is shown below:-

```
DECIMAL                          HEX
 65535--------------------------FFFFH
                RAM 0 - 7

 49152--------------------------C000H
                RAM 2

 32768--------------------------8000H
                RAM 5

 16384--------------------------4000H
                RAM 0 - 1

     0--------------------------0
```

RAMs 2 and 5 are ALWAYS in the positions shown. The RAM banks are of two types; RAMS 4 to 7 are contended (i.e. they share time with the video circuitry) and RAM 0 to 3 are uncontended (i.a. the processor has exclusive use). Any machine code which has critical timing loops (such as music or communications programs) should keep all such routines in the uncontended banks. The bit field for the hardware switch at 32765 is:-

Bit No.	Function
0	)
1	) RAM Select
2	)
3	) SCREEN Select
4	) ROM Select
5	) 48K Lock

It is important to note that the paging in and out of RAMS and ROMS using this port ONLY BE PERFORMED USING MACHINE CODE ROUTINES, AND CANNOT BE DONE FROM BASIC.
Bits 0,1 and 2 make a three bit number which selects which RAM goes into the C000H FFFFH memory slot. In Basic, RAM 0 is normally in situ, and when editing, RAM 7 is u for various buffers and scratchpads. Bit 3 switches screens; screen 0 is held in RA (beginning at 4000H) and is the one that Basic uses, screen 1 is held in RA (beginning at C000H) and can only be used by machine code programs. Bit 4 determ whether ROM 0 (the editor ROM) or ROM 1 (the Basic ROM) is paged into 0 to 3FFFH. Bi is a safety feature; once this bit has been set, the machine assumes a standard Spectrum configuration and all the memory paging circuitry is locked out. It can only returned to 128K by pressing RESET, or by switching off.

The main points to be aware of when transferring 128K programs are:-

(1). ALWAYS consider where the Stack is - if you put it in RAM 7 then page it out, will produce an immediate crash!

(2). ALWAYS be aware which RAM and ROM you have paged or are going to page - check appropriate bits.

ALL the programs on your SP7 disc will run on the +3 in +3 Basic.
NOTE:- Remember, to load any program:- At the +3 Menu, insert the SP7 disc and pr ENTER - select the program to load by pressing the appropriate number.
We must thus look for the following or similar piece of code somewhere in the Ba Loader which will perform the memory switching i.e.:-

```
LD BC, 7FFDH  - Loads BC register with I/O address.
LD A, 13H     - Load A register with data for switch - here it pages in Basic RO
                and RAM 3.
OUT (C),A     - Perform the switch.
LD IX,C000H   - Load IX register with Start Address of code to be loaded.
LD DE,4000H   - Load DE register with length of code.
LD A,FFH      - Load A register with 255 i.e. code NOT Basic is to be loaded.
SCF           - Set carry flag - signal LOAD.
CALL 556H     - Call the ROM loading routine.
RET           - Return to Basic.
```

We will now illustrate the method by looking at:-

THE NEVER ENDING STORY - 128K.

Load the Header Reader as detailed earlier. From this we find:-

Tape Count	Filename	Type	Details
5 - 8	NEVER	Basic	T.L.=230, A.R Line 10
9 - 25	NEVTITLE	M/C	S.A.=32768; C.L.=6912
26 - 29	NEVLOADER	M/C	S.A.=42752; C.L.=512
30 -->	Headerless Blocks		

The first task is to examine the Basic Loader "NEVER". All the initial operations are best performed in +3 Basic. Press RESET. Select +3 Basic, type load"t:", and load the file using MERGE"". This gave, on LISTING, as the more important points:-

```
10: CLEAR 24575: LOAD""CODE 16384
30: LOAD""CODE 42752
40: PRINT USR 42752
```

We see that Line 10 sets the stack at 24575, and loads the file "NEVTITLE" to the screen. Line 30 loads the file "NEVLOADER" to 42752, and line 40 starts the game. Thus "NEVLOADER" is the code to load and run the game. To examine this file, press RESET. Select +3 Basic, and type load"t:". Type CLEAR 24575, and load the file "NEVLOADER" by typing load "" code. Load the KD1 disassembler, and examine the area starting at 42752 (A700 Hex). This shows the start of the machine code to load the rest of the game. The object now is to use this suitably modified code to load the rest of the code blocks and save them to disc, then write a suitable Basic Loader to reload these blocks from the disc. Looking at the code we see the following code blocks:-

RAM Paged in	Start Address	Code Length	Title
4	24576	18176	NEV1
4	43520	10496	NEV2
4	23552	800	NEV3
0	49152	16384	NEV4
1	49152	16384	NEV5
3	49152	16384	NEV6

Remember, IX holds the Start Address, DE holds the code length. At A787 we see a JP 5800H - the E.A. = 24576. We must now load each code block from tape using a modified "NEVLOADER" code, then save each one to disc. Return to Basic from KD1. Type CLEAR 42750 then NEW. This clears the KD1 Basic but leaves the "NEVLOADER" code intact. Select +3 Basic. Type load"t:" then CLEAR 24575. We are now ready to load the code blocks from tape. Type in the following POKES:-

POKE	DETAILS
42753,0) 42754,0) - 42755,0)	These 3 pokes remove the LD SP,5FFE, this avoids stack problem
42761,0) - 42762,0)	These 2 pokes remove the OUT instruction, which avoids the paging in and out of any RAMS.
42776,251) - 42777,201)	These 2 pokes insert an EI (enable interrupts) and a RET (return) to give a return to Basic after loading.

Type R.USR 42752 and PLAY in the block from tape. Type save"a:" and save to disc w save"NEV1"code 24576,18176. The names NEV1 etc are just convenient titles for e block. Type load"t:", then POKE 42778,243; POKE 42792,251; POKE 42793,201. Type R. 42778 and PLAY in the next block. Type save"a:" and save to disc with save"NEV2"c 43520,10496. We can ignore the block NEV3 since this merely overwrites the Sys Variables. Type load"t:" then POKE 42817,243; POKE 42831,251; POKE 42832,201. Type R. 42817 and PLAY in the next block. Type save"a:" and save to disc with save"NEV4"c 49152,16384. Type load"t:" then POKE 42826,2 and R.USR 42817 and PLAY in the next blo Type save"a:" and save to disc with save"NEV5"code 49152,16384. Finally, type load" then POKE 42826,3; R.USR 42817 and PLAY in the last block. Type save"a:" and save disc with save"NEV6"code 49152,16384. We now have the required blocks of code on di it only remains to write a suitable Basic Loader to reload them. Care must be taken w the order of reloading of the blocks, and the positioning of the stack. The follow Basic Loader achieves both these points. Press RESET, select +3 Basic and type in:-

```
10: CLEAR VAL"43518"
20: LOAD"NEVSCR"CODE VAL"16384",VAL"6912"
30: GOSUB VAL"200"
40: LET A=VAL"0":GOSUB VAL"400"
50: LET A=VAL"1":GOSUB VAL"410"
60: LET A=VAL"3":GOSUB VAL"420"
90: POKE VAL"42765",VAL"170":POKE VAL"42768",VAL"41":LET A=VAL"4":GOSUB 430
98: RESTORE 310:GOSUB 200
95: LOAD"NEV1"CODE VAL"24576"
100: PAUSE VAL"100":RANDOMIZE USR VAL"42752"
200: LET A=VAL"42752"
210: READ A$:IF A$="*" THEN RETURN
```

```
220: POKE A,VAL A$:LET A=A+1:GOTO 210
300: DATA "243","62","1","1","253","127","237","121","33","144","101","17","0",
       "192","1","0","64","237","176","62","16","1","253","127","237","121",
       "251","201","*"
310: DATA "243","62","4","1","253","127","237","121","49","254","95","195","0",
       "96","*"
400: POKE VAL"42754",A:LOAD"NEV4"CODE VAL"26000":RANDOMIZE USR VAL"42752":RETURN
410: POKE VAL"42754",A:LOAD"NEV5"CODE VAL"26000":RANDOMIZE USR VAL"42752":RETURN
420: POKE VAL"42754",A:LOAD"NEV6"CODE VAL"26000":RANDOMIZE USR VAL"42752":RETURN
430: POKE VAL"42754",A:LOAD"NEV2"CODE VAL"26000":RANDOMIZE USR VAL"42752":RETURN
```

Save this to disc to auto run from line 10 with save"NES"line 10. The program will reload with load"NES".

Points to Note:-

(1). Line 10 sets the stack to 43518 which is a free area and allows plenty of room for the Basic Loader.
(2). Line 20 loads the screen picture.
(3). Line 30 POKES the following routine to address 42752:-

```
DI          - Disable interrupts
LD A,1      - Load A with 1;changed by POKES
LD BC,32765 - Load BC with I/O address
OUT (C),A   - Page in correct RAM = 1 here
LD HL,26000 - Load HL=26000,start of code
LD DE,49152 - Load DE=49152,move code here
LD BC,16384 - Load BC=16384,length to move
LDIR        - Move code from 26000 to 49152
LD A,16     - Load A with 16
LD BC,32765 - Load BC with I/O address
OUT (C),A   - Page in RAM 0 (Basic RAM)
EI          - Enable interrupts
RET         - Return to Basic
```

(4). Lines 40,50 and 60 POKE the correct value into 42754 (the value to go into the A register). The routine at lines 400,410,420 and 430 loads the block of code from tape, calls the machine code to page in the correct RAM, moves the code into place using the LDIR, pages back in the Basic RAM 0, and returns to Basic.
(5). Line 80 alters the routine to move "NEV2" code of length 10496 to 43520.
(6). Line 90 pokes the following routine into address 42752:-

```
DI          - Disable interrupts
LD A,4      - Load A with 4; page in RAM 4
LD BC,32765 - Load BC with I/O address
OUT (C),A   - Page in RAM 4
LD SP,24574 - Set stack to 24574
JP 24576    - Jump to start of game
```

(7). Line 95 loads the code "NEV1" into place. Line 100 pauses, then jumps to the routine and starts the game.

Using similar methods and considerations, other 128K games may be transferred to +3 Drive.

"KOBRAHSOFT KD1 DISASSEMBLER"

INSTRUCTIONS FOR USE

Introduction.

The KD1 disassembler is an efficient and easy-to-use disassembler for your ZX Spectrum. It can be used to list machine code instructions on your T.V. Screen. NOTE:- The KD1 was originally written to give a hard copy on a ZX Printer - this is THE ONLY type of printer on which KD1 can give a printout. Since the ZX Printer is not compatible with the Spectrum +3 the PRINTOUT OPTION IS NOT AVAILABLE. We have still given you KD1 since even without a printout option it is still an indispensable utility.
The KD1 will also allow easy interchange to and from Basic, and also contains a Number Converter to convert (a) Hexadecimal to Decimal numbers, (b) Decimal to Hexadecimal numbers.

Loading.

To load KD1, first ensure the Spectrum is reset, by pressing the RESET switch on the left side.

Insert the SP7 disc, then press ENTER. The SP7 Menu program will load and run automatically.
At the SP7 Menu, to load the KD1 disassembler, press key "9" then press ENTER - KD1 will load and run automatically.

When KD1 has loaded the following message will appear:- "To Start: Press BREAK". On pressing BREAK, KD1 goes into the Command Mode - with a flashing cursor at the bottom of the screen. At this stage, the following commands are available:-

Commands.

(1). Number Converter. (N).

To access the Number Converter, press "N". The message:- HEX or DEC? is displayed. To convert a Hexadecimal to a Decimal number, press "H". Now, enter your Hex. number. NOTE:- Any leading zeros MUST be included i.e. enter 38 Hex. as 0038, etc. Any normal Hex. letters (A-F) may also be entered. On pressing ENTER, the result is shown.
To convert Decimal to Hex., press "N", then "D" for Decimal. Now, enter your Decimal number WITHOUT leading zeros. Typing ENTER gives the result.

(2). Disassembler. (D).

NOTE:- Any section of memory may be disassembled, but NOT that occupied by KD1. attempt to do so will produce the message:- INVALID ADDRESS. To enter the disasse mode, press "D". Next, enter, IN 4 HEX. DIGITS, your required start address. You ca also specify similarly a 4 HEX. DIGIT end address. If you don't, the only differen that KD1 will continue its disassembly up to 65535 (FFFFH), which is pro inconvenient because of the long period of time needed! In the case of no specifie address - press ENTER. You are then asked if you require a printout to the ZX Pri as this does not apply here, press N, and a listing will appear on the screen. To more - press ENTER. To end the listing - press F.
When an end address is specified - ENTER is not needed - the enquiry PRINT? appear pressing the last digit.

Return to Basic. (R).

For a NORMAL return to Basic, press "R" while in Command Mode - you will be returne Basic. To re-enter KD1 from Basic, type:- RANDOMISE USR 59625, then press "BREAK". will take you into the Command Mode.

"HEADER READER"

INSTRUCTIONS FOR USE.

LOADING:-
Proceed as for KD1 above, but at the SP7 Menu, press key "8" then ENTER - Header Reader program will load and run.

USING:-
The HEADER READER will read the data from the header section at the start of each block in a program. It will display details such as:-

FILENAME:- The program name. This may sometimes be printed vertically due to presence of certain control codes in the header e.g. CHR$(13), etc.

PROGRAM TYPE:- i.e. Basic, Machine Code, SCREEN$, Numeric Array, Character Array, etc

PROGRAM LENGTH:- The HEADER READER will give, for a BASIC program, the total prog length (Basic program length + Variables), and the normal program length. It will a give the length of a machine code block.

START ADDRESS:- For a machine code block, this is the start of the block in memory.

AUTO-RUN LINE NUMBER:- For Basic programs only.

To obtain this information, load the HEADER READER as described above, then load your desired cassette and press "PLAY". For each header, the screen will clear, and the data read will be displayed.
It is usually best to "STOP" the tape when each header is read, so that the data can be written down. Press "PLAY" to continue. Repeat until no more data loads in i.e. the program has finished. This can then be repeated with any other tape which you wish to investigate.
We recommend you use the HEADER READER before copying a program, since this will tell you how many data blocks you must copy and hence, when the program has ended.

NOTE:- DO NOT, AGAIN, PRESS "BREAK" AT ANY TIME!

"HEADERLESS BLOCK LENGTH READER"

INSTRUCTIONS FOR USE

TO LOAD:-
Proceed as for KD1 earlier but at the SP7 Menu press key "7" then ENTER. The HBReader program will load and run displaying the message "PLAY the Tape". In order to determine the length of a Headerless Block, position your tape at the start of the block, then press "PLAY" on your recorder. The program will read in the bytes, count them, and print out the number of bytes in the block. To read in another block, press "r", then repeat as above.

With these two utilities, the composition of most programs can be determined (except fast loaders and pulsing programs).