KOBRAHSOFT SOFTWARE

SD5 ADVANCED TAPE TO MICRODRIVE UTILITY

OWNERS MANUAL

INDEX

SD5 ADVANCED TAPE TO MICRODRIVE UTILITY

INSTRUCTIONS FOR USE

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

GENERAL INTRODUCTION.

SD5 is our latest utility package to help you transfer the MAJORITY of your software to Microdrive. Unlike most similar packages, SD5 will help you transfer EVEN the latest PROTECTED programs. Its advanced features now include:-

(1). Tackles even the latest protected machine code programs.
(2). Now comes complete with our KD1 Disassembler.
(3). Comes complete with our "SD1 SPEEDLOCK DECODER" program. which makes transfer of "Pulsed Leader" programs to microdrive much easier. SD1 has now been improved and is in TWO parts. Part 1 allows transfer of the old type programs, whereas Part 2 now transfers the later type programs - for full details see later.
(4). SD5 now ALSO contains our latest SD2 Speedlock decoder which allows the transfer of the most up to date programs to microdrive. Even programs such as "ARKANOID 2", "BATMAN 2" and "AFTERBURNER" can now be EASILY transferred.
(5). Also included now is our "AD1 ALKATRAZ DECODER" program, which allows the easy transfer of programs which load with an unusual screen and a counter to zero.
(6). SD5 also now contains our NEW AD2 and AD3 Alkatraz decoders which enable the easy transfer of programs such as "THUNDERBLADE" and "H.A.T.E" to your microdrive.
(7). The latest addition to SD5 is our FB1 Firebird Decoder which enables the easy transfer of those "many small block" loaders by Firebird, including programs such as "FLYING SHARK" and "GOTHIK" etc.
(8). Now also contains instructions on how to transfer true 128K programs.

NOTE:- From here onwards, (ENTER) will mean "Press the ENTER Key".

To get the most out of your SD5 utility - particularly if you are a relative novice to programming - we suggest you first use the SD1,SD2,AD1,AD2,AD3 and FB1 transfer programs (see Index), as these will allow you to transfer your Speedlock, Alkatraz and Firebird programs to the microdrive with a minimum of previous knowledge.

For the more experienced programmer, the Technical Section below is a recommended good place to start, as it provides a good background knowledge for microdrive transfers.

Anyhow, this utility, unlike most, will try and show you the general techniques involved and needed to transfer even the more complex programs to your microdrive. We recommend that you start with your older games, as these should be easier to transfer.

TECHNICAL SECTION.

When you load a program, you usually see at first a burst of RED/CYAN THICK STRIPES -

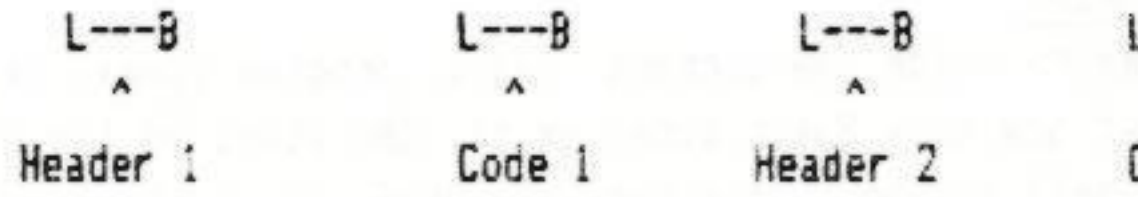
this is called a LEADER (L), and is around 5sec. in length for a "HEADER", but only 2sec. for a "CODE BLOCK". (These terms will be explained shortly). After, comes a burst of BLUE/YELLOW NARROWER STRIPES - these are BYTES (B) i.e. bits of code. Thus, for a typical BASIC program you get:-

```
        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:-



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

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.163 of your Spectrum Manual. We will now discuss the more important memory areas:-

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. Unless a printer is in use, it is usually free and as such can be used as a "work-space" e.g. to put machine code for example.

THE SYSTEM VARIABLES.

The 182 locations from 23552 to 23733 contains the many variables which tell the Spectrum the exact state of its memory at any one time. Several of the variables have important uses e.g. for program protection (see later).

THE MICRODRIVE MAPS.

This is a dynamic area of RAM i.e. it varies in length e.g. when a microdrive operation is called, the area expands by up to 600 bytes to accomodate variables etc. (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 standard Spectrum, PROG will indicate that the BASIC program starts at address 23755, and this will always be true unless a microdrive is 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
```

Now RUN it - it will print the addresses and characters as 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	)

ADDRESS	NUMBER	TOKEN / CHARACTER
23772	0	)
23773	204	TO
23774	50	2
23775	52	4
23776	48	0
23777	48	0
)	)	)
(	)	(
23796	13	ENTER

The above program (PEEK-LINE) is useful for examining a range of memory locations.

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:- RANDOMISE 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 RANDOMISE 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. These extra variables are only invoked when:- (1) A microdrive operation is performed; (2) A syntax error occurs. To check this, type:- PRINT PEEK 23635+256*PEEK 23636 (ENTER). (Incidentally, we shall assume the (ENTER) after a line from here onwards). This should give 23755 - the value of PROG. If not, RESET your computer with:- R.USR 0, then recheck. Now use your drive e.g. by typing:- CAT 1 for a cartridge catalogue. Now retype the above line - PROG will now be 23813! A similar situation occurs with (2) - try it - generate a syntax error (e.g. type:- CLS g). Now recheck PROG. 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 supplied.

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. Having frightened you thoroughly, let us say that we intend explaining just a few of the MAIN instructions - enough for you to tackle microdrive transfers!

YOUR PROGRAMS AND WHAT THEY DO.

For your assistance, we have supplied a large suite of programs. Mainly, they are in Basic - these will POKE some machine code into a particular location, this is then executed. A few are pure machine code - mainly Header and Block length readers - you need not understand how these work! NOTE:- These programs are ALL our copyright, so please don't sell them or give them to friends. We have deliberately left the programs visible, so that you can, if you wish, examine them. However, we HAVE included other copyright protections which, of course, we cannot disclose!

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

MNEMONIC	INSTRUCTION	NUMBERS
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 "MEEMONIC"). 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 microdrive 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.

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 microdrive with:- SAVE * "m";1;"name" LINE n. This will auto-run on reloading with LOAD "". However, it CANNOT be stopped by using MERGE "" - as an error results.

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 disconnecting then reconnecting the power supply. 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
```

Now, go into EDIT mode, and edit each line. While in the 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 either 23755 or 23813 - see earlier. Now type:-

```
50: FOR A=23755(23813) 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 23756 or 23814. Thus, type:- POKE 23756(23814),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! - very confusing to a hacker! Try it - type RUN. Press BREAK - the computer still crashes! NOTE:- Always check PROG before doing these protections - since it can change from 23755 to 23813 as explained earlier.

(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
```

Check PROG, and 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 is that now, a hacker 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 hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
```

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).

(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 and stop thus:-

(1). Play in the Header - then stop the tape - press "BREAK".
(2). Save the Header to tape with:- SAVE "name".
(3). Type RANDOMISE USR 0 - resets the computer.
(4). Re-load the Header "name" with LOAD "" - then stop the tape.
(5). Position the Basic Loader PAST its Header and PLAY it in - when it will load but NOT run NOR crash!

The Spectrum has done the work for us by calculating the space needed. Reloading removes the auto-run which means it loads and doesn't crash!

(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. 0-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 microdrive 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 usually, is finding the unscrambler! However, many are near the end of the main code.

OBJECTS AND PROBLEMS IN CONVERSIONS.

The programs encountered fall into 3 main types:-

(1). BASIC PROGRAMS.

If unprotected - these are simply transferred to microdrive with:- SAVE *"m";1;"name", OR:- SAVE *"m";1;"name" LINE No. - 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 microdrive.

(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 microdrive. The main point to note is that while your microdrive is running, it raises the start of Basic by around 650 bytes. Thus, any code loading below about 24500 (depending, of course, if any Basic is present also), will overwrite the microdrive information area - leading to a crash with the drive running! Thus, to allow for a few lines of Basic, the lowest safe address to load any code is around 25000.

THE PROGRAMS SUPPLIED AND WHAT THEY DO.

We have supplied you with a large set of programs to help you with your conversions. These are listed below.

LOADING INSTRUCTIONS FOR PROGRAMS.

FOR THE 48K SPECTRUM:- Type LOAD "" ; press ENTER and PLAY in the program.

FOR THE 128K SPECTRUM:- At the opening Menu, select 48K Basic, then proceed as for the 48K method above.

The programs are:-

(1). HEADER READER.

Probably the most important program. To load it, type:- LOAD ""(ENTER). It tells you the composition of the program you wish to transfer. (see separate instruction sheet). It gives the Program Name, Program Type, Start Address, Length, etc.

(2). HEADERLESS BLOCK LENGTH READER.

To load, type:- LOAD "". This program tells 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).

(3). MCbasldr.

To load, type:- LOAD "". This stands for "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:-

(4). BASCONV.

To load, type:- LOAD "". This is a BASIC CONVERTER. It "reads" the 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. Any machine code in the REMS may be examined with:-

(5). KD1 DISASSEMBLER.

To load, type:- LOAD "". This is an important addition to the utility, since you now have the facility to load in machine code, and get a full disassembly of all the instructions - see separate instruction sheet.

(6). CODE-STOP.

To load, type:- LOAD "". This is a small 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.

(7),(8),(9). MOVEUP, MOVEDOWN, BLOCK MOVE.

To load, type:- LOAD "" in each case. As the names suggest - these programs move blocks of code around in memory.

(10). SPEEDLOCK DECODER SD1.

This is a new and invaluable addition to this utility. It is a program which enables the EASY transfer to microdrive (and a backup to tape, if required) of the "Pulsed Leader" type of program. Now, with the new improved SD1, older type Speedlock programs e.g."LEADERBOARD"; "ARKANOID", are easily transferred; as also are the newer types e.g. "WIZBALL". See later for FULL instructions.

(11). SPEEDLOCK DECODER SD2.

This new program enables the easy transfer of the most up to date Speedlock programs to your microdrive. Programs such as "ARKANOID 2", "BATMAN 2" and "AFTERBURNER" can now be easily transferred. NOTE:- Neither SD1 nor SD2 can transfer ALL parts of the modern "Multi-Load" games - neither can hardware devices! This is because once loaded, the main part contains the code to load subsequent parts, and this cannot be changed. SD1 and SD2 WILL transfer the LONG main block - the final blocks must then be loaded from tape in the normal way.

(12). ALKATRAZ DECODER AD1.

Yet another very important addition to SD5, AD1 enables the easy transfer to microdrive of programs of the "Countdown" type; i.e. those which load with no loading border and feature a counter which reduces to zero. For FULL instructions, see later.

(13),(14). ALKATRAZ DECODERS AD2 AND AD3.

SD5 now contains the NEW Alkatraz decoders AD2 and AD3 which enable the latest Countdown programs to be transferred e.g. "THUNDERBLADE" and "H.A.T.E".

(15). FIREBIRD DECODER FB1.

This latest addition to the utility enables the easy transfer to microdrive of the "many small blocks" Firebird programs such as "FLYING SHARK" etc.

GENERAL METHODS OF TRANSFER.

The main point to remember is that in ANY game, the first part to load (usually a Basic program), MUST be normal speed, otherwise it could not be loaded with the usual LOAD "" command. The programs encountered will be of four main types i.e.:-

(1). UNPROTECTED BASIC PROGRAMS.

Save these to microdrive with:- SAVE*"M";1;"NAME" OR:- SAVE*"M";1;"NAME" Line No.

(2). PROTECTED BASIC PROGRAMS.

Stop any auto run by loading using MERGE "" instead of LOAD "". Then save to microdrive as in (1) above.

(3). UNPROTECTED MACHINE CODE PROGRAMS.

These are in the minority nowadays. The only programs which we know of which have unprotected machine code today are those by Melbourne House, and some of the budget priced games.

(4). PROTECTED MACHINE CODE PROGRAMS.

This covers the vast majority of modern games. Unfortunately, they can also be very difficult to transfer to microdrive, mainly because of the many and varied methods used to protect the machine code which comprises the game. We hope you will appreciate that any utility like SD5 CANNOT transfer ALL programs; but it can give general methods to transfer MOST. However, SD5 now contains our SPEEDLOCK DECODERS SD1 + SD2 and ALKATRAZ DECODERS AD1 and AD2, which now enable the transfer of "Pulsed Leader" and "Countdown" programs with the minimum of effort (see later). SD5 also now contains the NEW Firebird Decoder FB1 which should also prove an invaluable addition.

We will now deal with the two main types (3) and (4) in more detail:-

TRANSFER OF UNPROTECTED MACHINE CODE PROGRAMS.

A general (but necessarily simplified) method of transfer for BOTH unprotected AND protected programs is as follows:-

(1). Load the Basic Loader, using MERGE "" for unprotected programs. For protected programs, we advise you to load it thus:- First, load in the "Header Reader" program, then PLAY in the Basic Loader. Note the total length. This gives the number of bytes to load in using the MCbasldr program. Reset the Spectrum by pressing BREAK - this will remove the Header Reader program and clear the computer. Next, load the MCbasldr program. Type in the figure you have noted for "Number of bytes to Load = ". Follow the on screen instructions, then PLAY in the Basic Loader. The MCbasldr program will load ANY Basic Loader to address 32000, where it can then be examined using the BASCONV program. As instructed on screen, press any key to NEW the Spectrum - this removes the MCbasldr program but the Basic is still intact at address 32000. Now load the BASCONV program. This versatile program will "LIST" the Basic Loader, making any invisible lines visible. It also gives the address of any REM statements found, since these can be used to hold machine code (see earlier). Also, it gives the TRUE values for any numeric values found - the true value is the one in brackets. NOTE:- The initial screen asks if you require output to the screen (press "S"), or to a printer (press "P"). To output to a printer, you must have your own printer driver routine installed, usually high up in memory. It then asks you the Start Address of the Code - for a normal Basic Loader this is usually 23755, but it can be 23813 if a microdrive is connected (see earlier).

Next, enter the number of bytes to analyse - this is the value you noted using the Header Reader program. The Basic is then "listed" for you. Note any important items such as CLEAR statements, or POKEs or RANDOMISE USR values; also any PAPER or INK or BORDER colours. If you see any REM statements followed by a series of incomprehensible codes e.g. STEP ^ , / FOR INVERSE ! etc, this is probably machine code - note the address of the first character after the REM (given in the program) - this will be the start of the machine code which could possibly load the game.

To investigate any such code, type CLEAR 30000, then NEW. This will clear the BASCONV program, but leave the code intact. Load the KD1 disassembler program, and disassemble from the start of the code. Look for any hints as to the start and length and methods of loading the game code blocks (see earlier).

We have covered this part in much detail, since it will be frequently used. Next:-

(2). Look for the method of protection used in part (1) above.

(3). Decode the protection, and isolate the code to load the game.

(4). Stop the code from running after it has loaded the game so it returns to Basic.

(5). Load the game code, ensuring a return to Basic so it can be saved to microdrive.

(6). Move and/or save the game code to microdrive as required.

(7). Save a Basic Loader to reload the game code and start the game.

NOTE:- For unprotected programs, steps (2), (3) and (4) will not be needed. The whole object is to isolate the code which loads the game, make it return to Basic when it has loaded the game; load the game code, then save it to microdrive.

As an example to illustrate the method, consider:-

F1 SIMULATOR. (48K).

From the Header Reader, we find:-

Tape Count	File Name	Type	Details
5 - 8	F1	BASIC	T.L.=411,P.L.=411,A.R. Line 10
9 - 24	F1$	M/C	S.A.=32768, C.L.=6912 (SCREEN$)
25 - 94	F1CODE	M/C	S.A.=28655, C.L.=33700

Where:- T.L.= Total Length, P.L.= Program Length, S.A.= Start Address, C.L.= Code Length, A.R.= Auto-Run Line No. Load the Basic loader using MCBasldr thus:- Load MCBasldr, enter 411 for the Code Length (C.L.). Next, use BASCONV to "LIST" the Basic thus:- Load BASCONV, enter S.A. = 23813 (start of PROG), C.L.=411. This gave:-

```
10: CLEAR 28600: POKE 23296,237: POKE 23297,176: POKE 23298,195: POKE 23299,0: POKE
    23300,128: BEEP .1,1: BEEP .1, 2: BEEP .1,3: BEEP .1,4: BEEP .1,5
20: PAPER 0: BORDER 0: INK 7: BRIGHT 1: CLS: PRINT BRIGHT 1; INK 7; AT 9,5;"FORMULA £1
IS LOADING"; AT 12,10;"PLEASE WAIT": LOAD "" SCREEN$: INK 0: PRINT AT 5,0;:
LOAD "" CODE : R.USR 28655
```

NOTE:- It is always best to save parts such as:- CLEAR 28600; and the POKES. Also, any INK, PAPER etc colours. NOTE:- the E.A. (Execute Address) is R.USR 28655. Now, type R.USR 0 - resets the computer. Then type:-

```
10: CLEAR 28600: POKE 23296,237: POKE 23297,176: POKE 23298,195: POKE 23299,0: POKE
    23300,128: PAPER 0: BORDER 0: INK 0: BRIGHT 1: CLS: LOAD "" CODE
```

Position the tape at the start of F1CODE - type RUN - press PLAY. This will load the game code. Type R.USR 28655 - the game ran O.K. i.e the code was NOT protected. NOTE:- we have omitted F1$ (the SCREEN$), since this is unlikely to be checked and saves on microdrive space. We should have no problems here since, for the main code, S.A.= 28655, C.L.= 33700 (i.e. code loads from 28655-61355). Next, type in this Basic Loader:-

```
10: CLEAR 28600: POKE 23296,237: POKE 23297,176: POKE 23298,195: POKE 23299,0 : POKE
    23300,128: PAPER 0: BORDER 0: INK 0: BRIGHT 1: CLS: LOAD*"M";1;"f1code" CODE:
    RANDOMIZE USR 28655
```

Save to microdrive with:- SAVE*"m";1;"F1"LINE 10 - this will auto-run, on reloading, from line 10. Next, we must save the main code. Load it with:- Type R.USR 0 (clear computer): CLEAR 28600: LOAD "" CODE. Next, save with:- SAVE*"m";1;"f1code"CODE 28655,33700. The game can then be loaded with:- LOAD*"m";1;"F1". NOTE:- We usually compile a small menu program of the games on each cartridge thus:-

```
10: BORDER 0: PAPER 0: INK 7: CLS
20: PRINT AT 0,11;"GAMES 1"
30: PRINT AT 1,11;"=======":PRINT:PRINT:PRINT:PRINT
40: PRINT "(1) MANIC MINER.":PRINT:PRINT
50: PRINT "(2) JETSET WILLY.":PRINT:PRINT
60: PRINT "(3) JETPAC.":PRINT:PRINT:PRINT:PRINT
70: PRINT "PRESS (1),(2)OR (3) - (0) TO QUIT."
80: IF INKEY$="1" THEN CLS: PRINT AT 10,10;"MANIC MINER":LOAD*"m";1;"MANICMINER"
90: IF INKEY$="2" THEN CLS: PRINT AT 10,10;"JETSET WILLY":LOAD*"m";1;"J.WILLY"
100: IF INKEY$="3" THEN CLS: PRINT AT 10,13;"JETPAC":LOAD*"m";1;"JETPAC"
110: IF INKEY$="0" OR INKEY$="q" THEN RANDOMISE USR 0
120: GOTO 80
```

Save this to auto-run with:- SAVE*"m";1;"run" LINE 10. Calling the file "run" means that typing RUN (a single entry keyword) will load the file with only 1 key press. Thus, loading a game is reduced to only THREE key presses i.e. RUN, (ENTER), and 1,2, or 3!

TRANSFERRING PROTECTED MACHINE CODE PROGRAMS.

These are, of course, more difficult, since not only do we have to break the protection, but we have to contend with other complications such as long code blocks, which overwrite the System Variables. We can divide the programs likely to be encountered into 3 main groups e.g.:-

(1). CODE BLOCKS STARTING AT 25000 OR ABOVE.

(2). SHORT CODE BLOCKS LOADING BELOW 25000.

(3). LONG CODE BLOCKS LOADING BELOW 25000.

As stated earlier, we must remember that when a microdrive is used, the start of the Basic program area can be moved up by as much as 600 bytes. This must be considered when saving code at low addresses i.e. around 25000, since it could be corrupted by this moving of the Basic area. This is why the usual lowest safe address to save code is around 25000.

We will now consider examples of the 3 cases above to illustrate this. As an example of type (1) above, consider:-

ZYNAPS. (48K).

Using the Header Reader program to analyse the Basic Loader, we find:-

Tape Count	Filename	Type	Details
7 - 9	zynaps	Basic	T.L.=81. A.R Line 0.
10 - 14	loader	M/C	S.A.=64512; C.L.=768.
15 - 25	picy	M/C	S.A.=32768; C.L.=4096.
26 - 105	code	M/C	S.A.=23976; C.L.=40535.

NOTE:- A simple (but not infallible) indication of a program being protected is to try and load the Basic Loader using MERGE "" (this stops any auto-run). If the computer crashes or locks up - the program is usually protected. Try using MERGE "" in this case - the computer locks up!

We must thus load the Basic "zynaps" using the 'MCbasldr' and 'BASCONV' programs as detailed earlier. This gave the following essential details:-

10: BORDER 0: PAPER 0: INK 7: CLEAR 32767: LOAD "" CODE: R. USR 64512

This showed that the next part (the machine code program "loader") was loaded, then called by R.USR 64512. i.e. it was loaded and called normally. We now know that the "loader" code loads to 64512, is of length 768, and its Execute Address (E.A.) = 64512.

We now need to examine this code. However, if we load it to its normal address of 64512, it will overwrite the KD1 disassembler! We thus need to load it to a lower EQUIVALENT address, where it can be examined with KD1. By an equivalent address we mean this:- the S.A. of "loader" = 64512 Decimal = FC00 Hex. An equivalent address here would be 7C00 Hex or 31744 Decimal i.e. the first digit is altered to give a lower address, but the rest stay the same. Thus, load the "loader" code to 31744 by typing LOAD "" CODE 31744 - press ENTER then PLAY in the code. Next, type CLEAR 30000, to leave the code above RAMTOP, then load the KD1 disassembler. Examine 7C00 Hex onwards. We can see an unscrambler! This unscrambles the code starting at FC13 Hex. We can use this piece of code to unscramble the code at FC13 by inserting a RETurn instruction at 7C13 (= 31763 Decimal). Do this by typing POKE 31763,201 ENTER. Return to Basic from KD1, type CLEAR 30000, then NEW to clear the KD1 Basic. Now reload the "loader" code to its normal address with LOAD "" CODE ENTER, then PLAY it in. Now unscramble the code at 64512 by typing R.USR 31744. Save the code temporarily out to tape with:- SAVE"zynaps"CODE 64531, 653. The start address being 64531 = FC13 Hex. - this is the number in the HL register. The code length 653 = 28C is the number in the BC register. Type R.USR 0 to reset the computer. Type CLEAR 30000. The equivalent address here is FC13 = 7C13 = 31763. Reload the zynaps code with LOAD "" CODE 31763. Load KD1, and examine the equivalent start address = 7C13 - it shows JP FE3A - the start address. Examine the area from 7E3A onwards. This shows the following areas of interest:-

At the E.A. of 7E3A Hex (or equivalent address FE3A = 65082) we see a LD SP,FFFF Hex instruction. This sets the stack to the top of memory i.e. 65535. This is similar to a Basic CLEAR 65535. Next, we see DI, or DISABLE INTERRUPTS. This just makes the program run faster. At 7E3E we have a set of BLOCK MOVE instructions, where HL holds the start address; DE holds the destination address; BC holds the number of bytes to move. The LD (HL),0 instruction fills each byte with 0. In short, the effect is to fill the attributes with zero, or make the whole screen black. LDIR at 7E49 executes the block move. The CALL FE74 at 7E4B loads the next block of code (the screen); the CALL 8001 performs a check, the CALL FE74 at 7E51 loads the next code block i.e. "code". The important part starts at address 7E54; this is another block move which moves the 18 bytes from FE62 to FE73 into the printer buffer from 5B00 to 5B12; then jumps to the start at 5B00. At the end is the JP 8000 instruction (at 7E71) - this is the E.A. of the game! Thus, if we POKE addresses FE71, FE72 and FE73 (65137, 65138 and 65139) = 0, the printer buffer addresses 23311, 23312 and 23313 will be = 0. We can then put a piece of "stop" code starting at, say, 23320 to make the program return to Basic after it has loaded the game code, so that this game code can then be saved to microdrive. This piece of "stop" code is only 10 bytes long, and can be POKed in. Thus, start by typing R.USR 0, which clears the computer. Next, type CLEAR 50000 - this will ensure the code you are going to load is safe above RAMTOP. Type LOAD "" CODE, and play in the "zynaps" code you saved on tape. Now type POKE 65137,0; POKE 65138,0; POKE 65139,0 - this removes the "jump" instruction as stated earlier. We now put the "stop" code at address 23320 - type in the following:-

```
10: FOR A = 23320 TO 23329: INPUT B: POKE A,B: NEXT A
```

RUN it and enter the following numbers i.e. 253,33,58,92,33,3,19,229,251,201. For machine code programmers this gives:-

Numbers	Mnemonics
253,33,58,92	LD IY,5C3A Hex
33,3,19	LD HL,1303 Hex
229	PUSH HL
251	EI
201	RET

NOTE:- These 10 bytes will stop ANY piece of code running, provided the System Variables are not overwritten when the game code is loaded. Next, position the tape at the start of the "picy" file, and type R.USR 65082 (the execute address) and PLAY to load in the game code. After loading, the program will return to Basic - note the usual O.K. message at the bottom of the screen. We now have the game code in memory. Type CLEAR 24998, then save to microdrive with SAVE*"M";1;"zynaps" CODE 25000, 40536.

How do we know the start address and the length of the code to save? Note that at 7E62 Hex (equivalent to FE62 Hex) there is a block move, where HL=FBFE; DE=FFFE; and BC=9E57. It is an LDDR instruction i.e. the end of the code will be at FFFE, and the start will be FFFE - 9E57 = 61A7 = 24999. The code thus runs from 25000 to 65534. We now need a Basic loader to reload the code:- Type in:-

```
10: CLEAR 24998: LOAD*"M";1;"zynaps" CODE 25000: R.USR 32768
```

The CLEAR statement ensures the code is safe above RAMTOP, it is loaded then executed by the R.USR statement. Save this with:- SAVE*"M";1;"ZYNAPS" LINE 10. Saving this to microdrive means the line will auto run on reloading. The game is reloaded with LOAD*"m";1;"ZYNAPS", when it will load then run.
We have gone into much detail with this game so as to show you the usual procedure to follow. This can be adapted for other games.

Another example of this type is:-

GUNRUNNER. 48K.

Again, using the Header Reader, we find:-

Tape Count	Filename	Type	Details
12 - 15	Gunrunner	Basic	T.L.=81;P.L.=81; A.R. Line 0
16 - 19	loader	M/C	S.A.=64512; C.L.=768.
20 - 35	picy	M/C	S.A.=32768; C.L.=6912.
36 - 92	code	M/C	S.A.=25319; C.L.=38937.

As in the previous example, trying to load the Basic with MERGE "" gives a crash. So we must load it with MCbasldr and BASCONV as shown earlier. This gave:-

```
10: BORDER 0: PAPER 0: INK 7: CLEAR 25317: LOAD "" CODE: R.USR 64512
```

- does this look familiar? (See "ZYNAPS" method earlier!). The 2 sets of data are virtually identical! This illustrates an important point - the protection methods used by software houses often tend to be the same for many games - it is very expensive to alter! Here, both games are marketed by HEWSON; so if you can master the method, you should find it very similar for other HEWSON games featuring the "countdown" type of loaders. NOTE:- Do not confuse this with the usual countdown games which feature the "ALKATRAZ" protection system - for full details of these, see our AD1 program later.

Using a similar method as for "ZYNAPS", decode the loader, and reload it to address 31744. Examination with the KD1 disassembler shows a similar pattern; at the execute address of 7E3A Hex (equivalent to FE3A Hex = 65082 Decimal) we see a DI, followed by the attribute block move (7E4B - 7E55). This is followed by the CALL FE76 at 7E48; this

loads the screen display (the "picy" file) to address 32768; it is then moved into the display area by the block move at 7E4B. The CALL FE76 at 7E56 loads the game code and the block move at 7E59 moves 15 bytes into the printer buffer. The E.A. to start the game is the JP BBC6 instruction at 7E73 i.e. the E.A. = BBC6 Hex = 48070 Decimal. Follow a similar method as for "ZYNAPS", but POKE 65139,0; POKE 65140,0; POKE 65141,0; then put the "stop" code at 23320. Type CLEAR 24995; R.USR 65082 and PLAY in the game code. When the program returns to Basic, type CLEAR 26500; then save the game code to microdrive with:- SAVE*"M";1;"gun" CODE 26597, 38938. Again, the start and length of the code is obtained from the block move at 7E67 i.e. DE = FFFF; BC = 9818; thus the code ends at FFFF Hex, and starts at (FFFF - 9818) = 67E7 Hex = 26599 i.e. the code runs from 26599 to 65535. Reset the computer, and type in the following Basic loader:-

```
10: BORDER 0: PAPER 0: INK 7: CLEAR 26500: LOAD*"M";1;"gun" CODE 26597: R.USR 48070
```

Save to auto-run from line 10 with:- SAVE*"M";1;"GR" LINE 10. The game will reload with:- LOAD*"M";1;"GR".

An example of type (2) i.e. a short block loading below 25000 is:-

THE LIVING DAYLIGHTS. (48K).

From the Header Reader we get:-

Tape Count	Filename	Type	Details
0 - 6	Bond	Basic	T.L.=380; P.L.=380; A.R. Line 0.
7 -->	Headerless blocks		

Loading the Basic loader "Bond" using "MCbasldr" and "BASCONV" gave for the important points:-

```
0: R.USR 23768
```

- the E.A.! Note that sometimes you must enter the start of the code when analysing a Basic program as 23755; other times you must enter 23813; this is dependent on whether or not a microdrive was present when the program was originally saved. Here, enter 23813

Note the REM statement followed by all the apparently meaningless tokens - this is machine code contained in the REM statement, as shown earlier! This is the machine code we are after to load the game. To examine this in detail, type CLEAR 31000; NEW. This removes the BASCONV Basic but leaves the machine code safe above RAMTOP. Load the KD1 disassembler. The R.USR value of 23768 = PROG + 13 for an unexpanded Spectrum. Thus, examine the area from 32013 (i.e. 7D0D Hex) onwards with KD1. This is because the MCbasldr program loads the Basic to address 32000.

Using KD1, you will see at 7D0D a block move which effectively moves the whole screen along one byte. This seems to suggest some sort of screen check may be employed, so it will be wise to save the screen area as well as the main code! At 7D1B there is another block move which moves the area from 7D29 on up to FE00 i.e. 65024, then jumps to this address. The code from 7D29 on is thus the code to load the game. At 7D29 we see the instruction LD IX,4000 - this is the start address of the code = 16384 Decimal. At 7D2D we see LD DE,9600 - this is the code length = 38400 Decimal. The game code thus runs from 16384 to 54784. This gives us another problem - it will overwrite all the System Variables and Basic areas, making a simple return to Basic impossible! The only solution is the one normally employed in this case; load the code to a higher address originally, then save it out with a piece of 'mover' code tagged on the end which will enable the code to be reloaded to its high position, then moved back down to its normal position, and jump to the Execute Address; WITHOUT RETURNING TO BASIC. To do this, we must alter the value in IX. A convenient value here would be, say, 26000. The code would then load from 26000 to 64400. This will then let us return to Basic to save it. This can be done by typing POKE 32043,144; POKE 32044,101; this puts the address (101*256)+144 = 26000 in IX at 7D2B. We must also change the area moved at 7D1B (which = 5CF4), since this relates to the NORMAL position of the Basic. The address 5CF4 must be changed to 7D29. This can be done by typing POKE 32028,41; POKE 32029,125. We now need to find the E.A. Looking at address 7D98 we see JP 9100 i.e. 37120 - the E.A.! To force a return to Basic here instead of the jump to start the game, type POKE 32152,251; followed by POKE 32153, 201. This gives EI (enable interrupts), then RET (return to Basic after loading the game code). Position the tape after the Basic loader "Bond", type CLEAR 25990 (keeps the code above RAMTOP), then R.USR 32027 and PLAY in the game code. Now we must add the piece of "mover" code to the end of the game code. To do this, enter the following:-

```
10: FOR A = 64405 TO 64418: INPUT B: POKE A,B: NEXT A
```

RUN it and type in these numbers:- 33,144,101,17,0,64,1,20,150,237,176,195,0,145.
For machine code programmers, this gives:-

```
LD HL, 26000
LD DE, 16384
LD BC, 38420
LDIR
JP 37120
```

or "move the 38420 bytes starting at 26000 to 16384, then jump to the E.A. of 37120. Save this together with the game code with:- SAVE*"M";1;"bond" CODE 26000, 38420. Type in the following Basic loader:-

10: CLEAR 65535: LOAD*"M";1;"bond" CODE 26000: R.USR 64405

Save to auto-run from line 10 with:- SAVE*"M";1;"LD" LINE 10. The game reloads with:- LOAD*"M";1;"LD". Note:- The CLEAR 65535 must be included to avoid the stack being overwritten when the code is moved back down. The R.USR is 64405 since this is the E.A. to start the move down, and jump to the start of the game.

An example of type (3) is:-

DEATH WISH 3. (48K).

These long blocks loading below 25000 are by far the most difficult to transfer to microdrive. This is because they overwrite the System Variables, PLUS the fact that they usually leave very little space for the programmer. Applying the normal method, from the Header Reader, we find:-

Tape Count	Filename	Type	Details
5 - 8	DEATHWISH	Basic	T.L.=84; P.L.=84; A.R. Line 0.
9 - 14	c	M/C	S.A.=32768; C.L.=768.
15 ---> Fast Loading Code			

Following the usual method, load the Basic "DEATHWISH" using MCbasldr and BASCONV. This gave, as the most important points:-

10: PAPER 0: INK 0: BORDER 0: CLEAR 24575: LOAD "" CODE: R.USR 32768

i.e. the file "c" is the machine code to load the game. Load the code, then load KD1 and examine the start at 8000 Hex (32768). There are 2 versions of the game, 48K and 128K. The code at the start is concerned with the 128K version. Note the JP 8118 instruction at 8031. This is the start of the 48K routine. Let us examine the code blocks to be loaded. At 811B we see IX=4000; DE=1B00; i.e. S.A.=16384, C.L.=6912. (16384 - 23296). At 8125 we see IX=5B00; DE=2500; i.e. S.A.=23296, C.L.=9472 (23296 - 32768). At 8137 we see IX=8400; DE=7C00; i.e. S.A.=33792, C.L.=31744 (33792 - 65535). Finally, at 814E we see IX=4000; DE=400; i.e. S.A.=16384, C.L.=1024 (16384 - 17408). The program then follows the JR 80F9 at 8158, and jumps to 80F9, where a block move instruction moves 100 bytes from 8107 onto the screen at 4400 Hex (17408 Decimal), then jumps to that address. There, the stack is set to 5FFF (24575), and the 1024 bytes previously loaded onto the screen are moved to address 32768, followed by a jump to 6C40 Hex (27712 Decimal), which is the E.A. to start the game. The problem is, where to put the "stop" code, as there is little room left! The answer is to POKE it into addresses 8141 - 8149 (33089 - 33098). This will load the first 3 blocks, then return to Basic. Note that the stack is set at 24575 just before the game is started - this gives an indication that any code just

below this area can be ignored, since it is unlikely to be used in this area. Start afresh, reset the computer, then reload the loader code file "c". Enter the following Basic line:-

```
10: FOR A = 33089 TO 33098: INPUT B: POKE A,B: NEXT A
```

- then RUN it and enter the following numbers; 253,33,58,92,33,3,19,229,251,201. This puts the "stop" code in position. Now type R.USR 33048, and PLAY in the fast loading blocks after the "c" loader file. On returning to Basic, we have code which we must save which starts at 24576. This cannot be saved to microdrive without it being overwritten (see earlier). We must thus save, say, the first 2000 bytes temporarily to tape. This will allow enough room to use the microdrive. Save this code to tape with SAVE "a" CODE 24576, 2000. The code from 24576 to 26576 is now safe on tape. Type CLEAR 26500, then LIST. You will see two lines of Basic from when the programmer was writing the program! Delete these, by pressing 10 ENTER and 20 ENTER. We now need to continue loading the rest of the game, then return to Basic to save it. To do this, where can we put the "stop" code? The answer is ON THE SCREEN, after the code which is loaded there i.e. the block starting at 814E which starts at 16384 and is 1024 bytes long i.e. the code loads from 16384 to 17408. Also, at 80F9 we see that 100 bytes of code from 8107 onwards are also loaded to the screen after the first block. Thus the total code runs from 16384 to 17508. We can thus put the "stop" code at, say, 17510. Enter the following:-

```
10: FOR A = 17510 TO 17519: INPUT B: POKE A,B: NEXT A
```

RUN it, then enter these numbers; 253,33,58,92,33,3,19,229,251,201. They will appear on the screen as a series of lines. NOTE:- Don't use any clear screen instructions, or you will erase the code! Also, type POKE 33046,102; POKE 33047,68. This makes the program jump to the "stop" code and return to Basic after loading the rest of the game code. Type R.USR 33102 and PLAY in the last block of code. Type CLEAR 26500. We can now save the code to microdrive. To do this, type SAVE*"M";1;"b" CODE 26576,38959 Then SAVE*"M";1;"c" CODE 23296,256. This saves the main code and the small block in the printer buffer. Reset the computer. We now need to reload the "a" code to a suitable location, say, 32000 and then tag some "mover" code on the end which will move it from the screen area (where we reload it) to its normal position i.e. 24576 AFTER we have finished using the microdrive. Load it with LOAD "" CODE 32000 and PLAY it in. The code now runs from 32000 to 34000. To tag the "move" code on the end, enter:-

```
10: FOR A = 34001 TO 34017: INPUT B: POKE A,B: NEXT A
```

RUN it, then enter:- 49,255,95,33,0,64,17,0,96,1,208,7,237,176,195,64,108. In machine code, this is:-

```
(1) LD SP, 24575
(2) LD HL, 16384
(3) LD DE, 24576
(4) LD BC, 2000
(5) LDIR
(6) JP 27712
```

(1) sets the stack to 24575, below the code.
(2) loads HL with 16384, the start of the code to be moved.
(3) loads DE with 24576, the destination of the code.
(4) loads BC with 2000, the number of bytes to move.
(5) performs the move.
(6) jumps to the execute address and starts the game.

Save this to microdrive with SAVE*"M";1;"a" CODE 32000, 2020. NOTE:- the code length is now 2020 to include the "mover" code. This will be reloaded to 16384 (on the screen), with a start address of 18385 (i.e. 16384+2001). Reset the computer, and enter the following Basic loader:-

```
10: CLEAR 26500: LOAD*"M";1;"b" CODE 26576: LOAD*"M";1;"c" CODE 23296: LOAD*"M";1;"a"
    CODE 16384: R.USR 18385.
```

Save to auto-run from line 10 with SAVE*"M";1;"DW3" LINE 10. The program will reload with LOAD*"M";1;"DW3".

Using modifications on the above methods, other programs may be transferred to your microdrive.

"SPEEDLOCK DECODER SD1"

INSTRUCTIONS

INTRODUCTION.

One form of software protection which has recently been developed is the famous (or infamous) SPEEDLOCK system - which uses the dreaded "Pulsed Leaders" to load the program. These have lead to many loading problems, and because of this we have produced our "SD1 SPEEDLOCK DECODER" system. We did this since these programs are now so widespread - being used by several software houses including Ocean, US Gold, and Imagine to name but a few. SD1 converts the pulsed leaders to normal ones, and saves the program out at normal speed. It also saves the program code in two well defined blocks, with NORMAL headers, and gives the start address for the machine code, as well as producing a Basic loader to reload the program for Tape, (if required), and Microdrive conversions. This means that these programs can now be converted to your Microdrive quite easily. We have improved SD1 and it is now in TWO distinct parts. Part 1 converts the "old" type Speedlocks programs, whereas Part 2 converts the later types. Please note that any utility like SD1 (and AD1) can NEVER transfer ALL programs, but it will transfer most. SD5 also now contains our new Speedlock Decoder SD2 which will transfer the VERY LATEST Speedlock programs - as we promised in SD4!

HOW TO TELL WHICH SPEEDLOCK SYSTEM A PROGRAM CONTAINS.

The OLDER type Speedlock games had 1 (sometimes 2) Basic parts, these were usually fairly short, and were followed by the "Pulsing Leaders". Here, instead of seeing the usual RED and CYAN (light blue) loading stripes in the border, we see "jerking" stripes which load with a "clicking" sound. The colour of the stripes was later changed to RED and BLACK on games such as "ENDURO RACER" etc. All these are classed as OLD type Speedlocks, and SD1 Part 1 will convert these. The NEWER Speedlock games are completely different. These have a very short Basic section, then a long Basic section. This is followed by the border turning red/black and a series of random musical "beeps" is heard. Following this is a very short red/black part; then the start of a long block loading with a blue/black border. Also, there appears on the screen a counter which starts around 2m 30s and counts down to 0. Part 2 of SD1 will convert these programs. NOTE:- As mentioned earlier, the newest Speedlock e.g. in "ARKANOID 2" has similar features, but has clicks instead of the random "beeps" - SD2 must be used for these.

LOADING SD1.

Firstly, clear your Spectrum by either disconnecting then reconnecting the power supply, OR pressing RESET, whichever is applicable.

For the 48K Spectrum:- Type:- LOAD""; press ENTER then PLAY on your recorder.

For the 128K Spectrum:- Select 48K Basic at the Menu; proceed as for 48K above.

SD1 will load and run, starting with the message:- "Please Choose: (1) OLD Type Speedlocks; (2) NEW Type Speedlock."

(1). TRANSFERRING OLD TYPE SPEEDLOCK PROGRAMS.

Press key "1" at the above menu, when the message "Press ENTER then PLAY Tape" will be displayed. Remove the SD1 tape, and insert your Speedlock program tape, and rewind it to the start. SD1 is now ready to use. Some old Speedlock programs have ONE Basic section, others have TWO, before the pulsing sections. Where TWO Basic sections are present, SD1 will ignore the first. Press ENTER, then PLAY on your recorder. When SD1 meets the main Basic, it will display its name, which will be used when saving Basic and code blocks. Watch your screen, and when the main Basic has loaded you will hear a warning beep and see a message saying "Stop the Tape". STOP your recorder - there will be a short pause while SD1 decodes the Speedlock Basic, then the message "Start Address of Code = *****" will appear. This is the E.A. (R.USR number) of the converted program for microdrive transfer - write it down. NOTE:- You may sometimes see the message:- "Decoding Error - Press ENTER". If this happens, rewind your Speedlock tape to its start, press ENTER, then PLAY to try again. If the error reoccurs, try again using a slightly different volume, if it still happens, assume the program isn't Speedlock, or SD1 can't convert it. The Menu available is:-

Key	Function
"1"	- start loading a NEW Speedlock program.
"t"	- Save Basic loader for TAPE use.
"m"	- Save Basic and code for microdrive transfer.
"n"	- Next stage; loading pulsing parts.
BREAK	- Performs a reset or NEW.

Now, if you wish to go back to the start, simply press the "1" key, otherwise place a new tape into your recorder, start recording, and according to which system you are converting for, press the appropriate key i.e.:- the "t" or the "m" key. Press "t" for a copy that is to be reloaded from tape. Press "m" if you are transferring to microdrive. ENSURE you press RECORD and PLAY on your recorder BEFORE pressing a key, since pressing a key saves out to tape immediately. On pressing the appropriate key, you will save a Basic program that will load in the rest of the program's machine code i.e. a Basic Loader. This will contain the equivalent of LOAD"a"CODE for tape; LOAD#"m";1;"a"CODE for microdrive. Also, for microdrive transfer you will save an additional block of machine code which is needed to make your program run.

Having now saved a program to tape, we go on to the next section:-

Having saved out to tape, replace the Speedlock tape in your recorder, and press the "n" key. This will show the FINAL Menu, i.e.:-

(1) Save Code (Tape or Microdrive).
(5) Start Game.
BREAK = NEW

NOTE:- This is the Menu available AFTER all pulsing sections have been loaded. Start loading all the pulsing sections by pressing ENTER, followed by PLAY on your recorder. Wait until SD1 has loaded in ALL the program - this is shown by the border going YELLOW. If the screen display is corrupted, this indicates a loading error - this means you must NEW the Spectrum (press BREAK) and reload SD1. The keys available now are as described above i.e. (1), (5) or BREAK. Replace your saving tape into your recorder and set ready to record. For Tape, and / or Microdrive transfer, press RECORD and PLAY, followed by key (1). Wait until the program has been saved (border YELLOW). For tape backups the process is now complete. The program reloads with the usual LOAD "" ENTER command. When the program has loaded, (screen loaded), Press ENTER to start the game. NOTE:- SD1 always saves 2 blocks of code with normal headers, followed by a single headerless block. The last block is used for tape transfer only and is in fact the screen display. The first 2 blocks have the same name as the original except the tenth character is set to 1 and 2 respectively. These 2 blocks are always of length 4096 and 38144 bytes respectively e.g. for a program "test", SD1 would save these 2 blocks:-

"test 1", start address = 23296, length = 4096.
"test 2", start address = 27392, length = 38144.

In addition, for Microdrive, an extra block of code i.e. "test 3", will be saved, whose Start Address = Start of Code address given on screen. This is saved AFTER the Basic loader. NOTE:- After saving, you can either reset (clear) the Spectrum by pressing BREAK, or play the game by pressing key (5).

FINAL TRANSFER TO MICRODRIVE.

We now need to transfer the Basic and machine code to your drive. Ignore the last block on the tape. For Microdrive conversions, the Basic Loader (the 1st on the tape) must be transferred to microdrive. To transfer it, type MERGE "", press ENTER and PLAY it in from tape. This stops any auto-run. Save to Microdrive with:- SAVE*"m";1;"test"LINE 0 - press ENTER. We now need to transfer the 3 blocks of machine code to the Microdrive. This is done thus:- Reset your Spectrum. We now need to know the start address and length of the first code block which is "test 3". Do this using our Header Reader program HReader. Load this by typing LOAD "" then press ENTER. PLAY in the code block. The Start Address will be 34025. Let us assume the code length is 55 bytes. Reset your Spectrum. Type CLEAR 27390. Load the block from tape with LOAD "" CODE - press ENTER, then PLAY. Save to microdrive with:- SAVE*"m";1;"test 3" CODE 34025, 55 - press ENTER. The next block on tape is "test 1". Load this to a convenient address e.g. 40000 with LOAD "" CODE 40000 - ENTER. Save to microdrive with:- SAVE*"m";1;"test 1" CODE 40000,4096 - ENTER. Load the third and final block with LOAD "" CODE 27392 - ENTER. Save to microdrive with SAVE*"m";1;"test 2" CODE 27392,38144 - ENTER. The program will reload from microdrive with:- LOAD*"m";1;"test" - ENTER.

(2).TRANSFERRING NEW TYPE SPEEDLOCK PROGRAMS.

Press key "2" at the opening Menu, and follow the "old" method.

"SPEEDLOCK DECODER SD2"

INSTRUCTIONS

INTRODUCTION.

Our new SD2 enables the EASY transfer of the VERY LATEST Speedlock programs using the following procedures:-

(1). Load SD2, then PLAY in the Speedlock tape.

(2). When loaded, press ENTER which saves 3 code blocks to a new tape.

(3). Load the transfer program "Transfer"; put a formatted cartridge in the drive then PLAY in the saved tape - after loading the program is automatically transferred to your cartridge, and reloads by pressing key "R" for RUN.

The FULL details are now given below:-

LOADING SD2.

For the 48K Spectrum:- RESET. Then type LOAD "" and press ENTER and PLAY in SD2.

For the 128K Spectrum:- RESET. Select 48K Basic and proceed as for the 48K method.

PLEASE NOTE:- BEFORE loading SD2 you must ALWAYS RESET your Spectrum to ensure all the memory is free and clear.
When SD2 has loaded, the message "PLAY THE TAPE" appears. Remove your SD5 tape, and insert the tape you wish to transfer into your recorder. Rewind it to the start and press PLAY. When the first long part has loaded, the border will flash as SD2 decodes the Speedlock - leave the tape PLAYing. The program should load as normal. When loaded - this is usually when the on screen counter reaches zero - remove your game tape and insert a new tape. Press REC + PLAY to save then press ENTER - 3 headerless blocks will be saved to your tape. On completion of saving, the game will run as normal - play it if you wish. Rewind the saving tape. RESET the Spectrum, and load the "Transfer" program from your SD5 tape using the same method as for SD2 above. When loaded, the screen goes BLACK and the Spectrum goes into tape loading mode. Put a formatted cartridge in the microdrive. Insert your saved tape and press PLAY. "Transfer" will load the 3 blocks. NOTE:- When the blocks are loading some distortion of the upper part of the screen will appear, but this is normal. When the tape has loaded, "Transfer" will AUTOMATICALLY transfer the program to microdrive together with a file called "run". The computer will then reset. To reload the program from cartridge, simply press the "R" key to give the RUN keyword and then press ENTER. The game will load and run as normal.

"ALKATRAZ DECODER AD1"

INSTRUCTIONS

INTRODUCTION.

Another popular form of protection system which has recently been evolved is the "ALKATRAZ PROTECTION SYSTEM". It is used by software houses such as Ocean, US Gold and The Edge. Its presence is shown by a loading sequence which has no changing border colours; a screen display which loads in an unusual way; and a 3 digit counter which reduces to zero. Its presence can be confirmed by loading the Basic loader using MERGE "" - ENTER. The message "ALKATRAZ PROTECTION SYSTEM" will be displayed.

Again, the problem with these programs are they are VERY sensitive to loading volume changes. Thus, it is a good idea to ensure before trying to convert these programs, that the correct volume has been selected to load the original!

We have thus evolved our "AD1 ALKATRAZ DECODER", which converts the programs to standard format regarding speed and headers. This enables easy transfer to tape, (if required), and Microdrive.

LOADING AD1.

Firstly, ensure your Spectrum is cleared by either disconnecting then reconnecting the power supply, OR pressing the RESET switch, whichever is applicable.

For the 48K Spectrum:- Type LOAD "", press ENTER, then PLAY on your recorder.

For the 128K Spectrum:- Select 48K Basic from the opening Menu, then type LOAD "", press ENTER, then PLAY on your recorder.

AD1 will load and run, starting with the message:- "Press ENTER then PLAY Tape". Remove the AD1 tape, and insert your Alkatraz tape, and rewind it to the start. AD1 is now ready to use.

Press ENTER, then PLAY on your recorder. Watch your screen, when the main Basic has loaded you will hear a warning beep and see a message saying "Stop the Tape. STOP your recorder - there will be a short pause while AD1 decodes the Basic, then you can select a Basic loader for M/d or Tape, by pressing key "m" or "t". You may see the message:- "Tape Error - Please press ENTER". If this occurs, press ENTER, rewind the tape, and repeat, trying a slightly different volume. If this persists, either the program is not an Alkatraz program, or AD1 can't convert it. Remove the Alkatraz tape, and insert a new tape, press RECORD and PLAY, and press "t" to save Tape Basic, "m" to save Microdrive Basic.

Press "n" to go to the next stage. Replace the Alkatraz tape, press ENTER, then PLAY. When the next part has loaded, STOP the tape and AD1 will decode this part. Press ENTER then PLAY again. AD1 will load in the rest of the program as normal. When successfully loaded, the screen will appear jumbled with a YELLOW border. If an error occurs, (no jumbled screen or yellow border), reload AD1 and start again.

On successful loading, reinsert your saving tape, press RECORD and PLAY, then press key (1). This applies irrespective of whether you are saving for tape or for M/d.

AD1 will then save 3 blocks of code with the original name, but with the tenth character altered to 1,2 and 3. Repeat copies can be obtained by pressing key (1), or the Spectrum can be cleared by pressing BREAK. Alternatively, you can play the game by pressing key (5), then ENTER. A TAPE conversion is completed at this point. You now have a Basic loader and three pieces of code on tape.

The three pieces of code can now, if required, be transferred to microdrive as detailed below. They should be saved with names like "test 1", test 2", and "test 3", as shown below. Remember, the lengths are always 20200 for block 1, 20536 for block 2, and 6912 for block 3. To transfer block 1, type CLEAR 26999, then LOAD "" CODE 27000, press PLAY on your recorder, and load in the block. Save to your M/D using SAVE*---"1" CODE 27000, 20200. Repeat this for loading and saving the other two pieces of code, using SAVE*---"2" CODE 27000, 20536 for the second and SAVE*---"3" CODE 27000, 6912 for the third. The Basic can be saved thus:- Load it using MERGE "" then save with SAVE*"m";1;"test" LINE 0.

"ALKATRAZ DECODER AD2"

INSTRUCTIONS

INTRODUCTION.

Our new AD2 program enables the EASY transfer of the VERY LATEST Alkatraz programs using the following procedures:-

(1). Load AD2, the PLAY in the Alkatraz tape.

(2). When loaded, press ENTER, which saves 3 code blocks to a new tape.

(3). Load the transfer program "Transfer"; put a formatted cartridge in the drive then PLAY in the saved tape - after loading, the program is automatically transferred to your cartridge, and reloads by pressing key "R" for RUN.

The FULL details are now given below:-

LOADING AD2.

For the 48K Spectrum:- RESET. Then type LOAD "" and press ENTER and PLAY in AD2.

For the 128K Spectrum:- RESET. Select 48K Basic and proceed as for the 48K method.

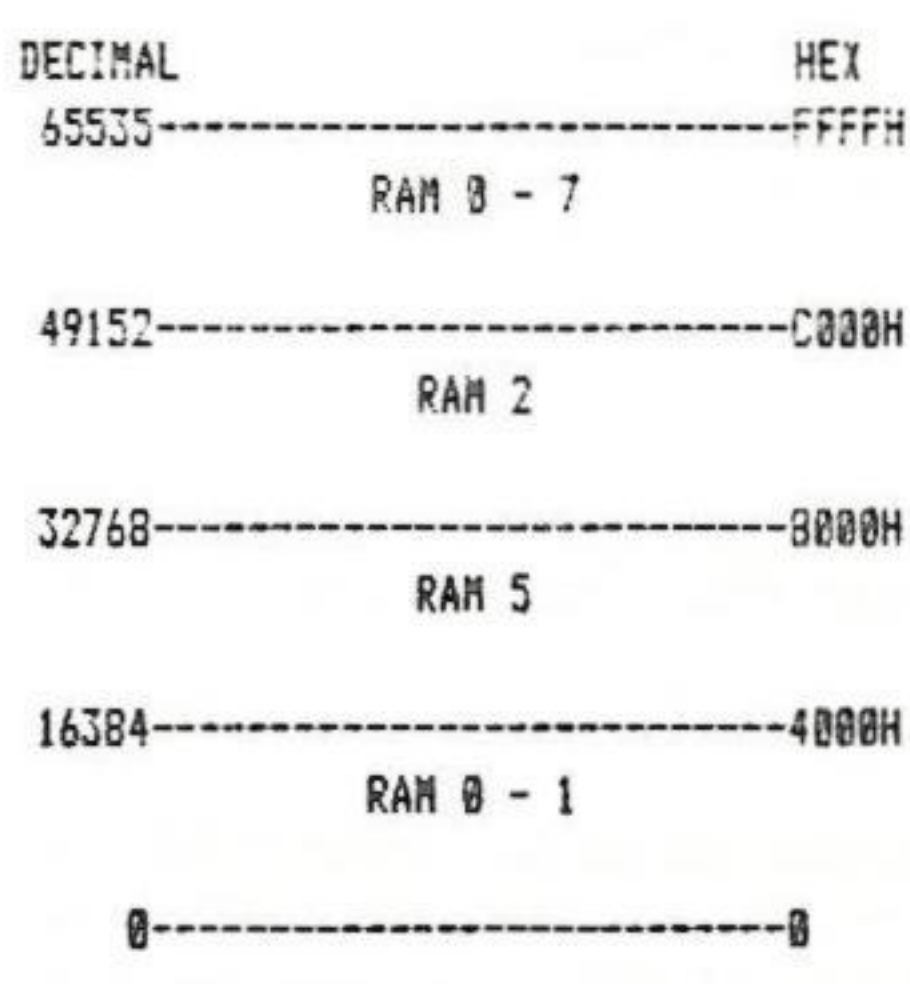
PLEASE NOTE:- BEFORE loading AD2 you must ALWAYS RESET your Spectrum to clear the memory When AD2 has loaded, the message "PLAY THE TAPE" appears. Remove your SD5 tape, and insert the tape you wish to transfer into your recorder. Rewind it to the start and press PLAY. When the first long part has loaded, the border will flash as AD2 decodes the Alkatraz, and a few coloured squares appear toward the top of the screen, but this is normal. Now leave the tape PLAYing. The program should load as normal. When loaded - this is usually when the on screen counter reaches zero - remove your game tape and insert a new tape. Press REC + PLAY to save then press ENTER - 3 headerless blocks will be saved to your tape. On completion of saving, the game will run as normal - play it if you wish. Rewind the saving tape. RESET the Spectrum, and load the "Transfer" program from your SD5 tape using the same method as for AD2 above. When loaded, the screen goes BLACK and the Spectrum goes into tape loading mode. Put a formatted cartridge in the microdrive. Insert your saved tape and press PLAY. "Transfer" will load the 3 blocks. NOTE:- When the blocks are loading some distortion of the upper part of the screen will appear, but this is normal. When the tape has loaded, "Transfer" will AUTOMATICALLY transfer the program to microdrive together with a file called "run". The computer will then reset. To reload the program from cartridge, simply press the "R" key to give the RUN keyword and then press ENTER. The game will load and run as normal.

"SD5 TAPE TO M/D 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:-



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.e. 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 CAN 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 to FFFFH memory slot. In Basic, RAM 0 is normally in situ, and when editing, RAM 7 is used for various buffers and scratchpads. Bit 3 switches screens; screen 0 is held in RAM 5 (beginning at 4000H) and is the one that Basic uses, screen 1 is held in RAM 7 (beginning at C000H) and can only be used by machine code programs. Bit 4 determines whether ROM 0 (the editor ROM) or ROM 1 (the Basic ROM) is paged into 0 to 3FFFH. Bit 5 is a safety feature; once this bit has been set, the machine assumes a standard 48K Spectrum configuration and all the memory paging circuitry is locked out. It can only be 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, this will produce an immediate crash!

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

MOST programs on your SD5 cassette will run on the 128K Spectrum (in 128K mode). The exceptions are:- "MOVEUP" (alter Line 270 to: LET code=18560); and "MOVEDOWN" (alter Line 130 to: LET code=18560).

NOTE:- The SD5 programs can simply be loaded in +3 Basic by pressing ENTER at the Initial Menu, when they will load and run.

We must thus look for the following or similar piece of code somewhere in the Basic 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 ROM
                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:-

KNIGHTYME - 128K.

Using the Header Reader we find:-

Tape Count	File Name	Type	Details
7 - 8	KNIGHT	Basic	P.L.=88; T.L.=88; A.R. Line 10.
9 - 18	SCREEN	M/C	S.A.=16384; C.L.=6912.
19 - 35	XXX	M/C	S.A.=32768; C.L.=16384.
36 - 43	XXX	M/C	S.A.=32768; C.L.=6912.
44 - 63	XXX	M/C	S.A.=32768; C.L.=16384.
64 - 83	XXX	M/C	S.A.=32768; C.L.=16384.
84 - 103	XXX	M/C	S.A.=32768; C.L.=16384.
104 - 152	XXX	M/C	S.A.=26240; C.L.=39296.

Let us consider the Basic loader in more detail. Use MERGE "" then LIST shows:-

```
 10: CLEAR VAL "26200":INK NOT PI:PAPER NOT PI:BORDER NOT PI:CLS
 20: LOAD "" SCREENS
 30: GOSUB 200
 40: LET A=3:GOSUB 400
 50: LET A=7:GOSUB 400
 60: LET A=6:GOSUB 400
 70: LET A=4:GOSUB 400
 80: LET A=1:GOSUB 400
 90: PRINT AT 5,NOT PI:LOAD "" CODE
100: RANDOMIZE USR VAL "26624"
200: LET A=VAL "26210"
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","0","128","17","0","192",
     "1","0","64","237","176","62","16","1","253","127","237","121","251", "201",
     "*"
400: POKE VAL "26212",A:PRINT AT 5,NOT PI:LOAD "" CODE:RANDOMIZE USR VAL  "26210":
     RETURN
```

Let us further examine each line:-

LINE 10:- Sets stack to 26200; sets colours to 0 (black).

LINE 20:- Loads the screen display.

LINE 30:- Pokes a machine code routine starting at 26210 to load data and page in RAMS. This is:-

```
DI          - Disable interrupts
LD A,1      - Load A register with 1 - changed by POKING
LD BC,32765 - Load BC with I/O address
OUT (C),A   - Switch in appropriate RAM - here = RAM 1
LD HL,32768 - Load HL with 32768 - start of loaded code
```

```
LD DE,49152 - Load DE with 49152 - where code will be moved to
LD BC,16384 - Load BC with no. of bytes to move
LDIR        - Move code from 32768-49151 to 49152-65535
LD A,16     - Load A register with 16
LD BC,32765 - Load BC register with I/O address
OUT (C),A   - Switch in ROM 1, RAM 0
EI          - Enable interrupts
RET         - Return to Basic
```

Lines 40,50,60,70 and 80 POKE the correct value into 26212 (the value to go into the A register). The routine at line 400 also uses the PRINT to print the program name so as not to corrupt the screen picture. It then loads the block of code from tape, and calls the machine code routine which pages in the correct RAM, moves the code into place using LDIR, pages back in ROM 1, RAM 0, then returns to Basic.
Line 90 loads the final code block, then starts the game at E.A.=26624.
Since the program code is unprotected, the main difference in the Basic loader needed is to change the tape load commands to the appropriate microdrive commands. As you can see, the total length of all the code blocks is well over 100K i.e. it is impossible to get the program on a single cartridge. We must use a second cartridge also. We can get all the blocks but the last on the first cartridge. To allow a change of cartridge, we must put a PAUSE 0 command in before loading this final block. This means that the computer will load the initial blocks, then wait for you to press any key. We can thus change to the second cartridge, then press any key and the final block will be loaded and the game will run.
As you will see from the Start Addresses and Code Lengths of the various blocks, we should have no problems with overwriting the Microdrive Channel, since the lowest main block starts at 26240. We must also put in more subroutines at the end of the program to specify names for each code block so they can be reloaded from microdrive. The new Basic Loader is:-

```
 10: CLEAR VAL "26200":INK NOT PI: PAPER NOT PI: BORDER NOT PI: CLS
 20: LOAD*"M";1;"SCREEN"CODE 16384,6912
 30: GOSUB 200
 40: LET A=3: GOSUB 400
 50: LET A=7: GOSUB 410
 60: LET A=6: GOSUB 420
 70: LET A=4: GOSUB 430
 80: LET A=1: GOSUB 440
 85: PAUSE 0
 90: PRINT AT 5,NOT PI: LOAD*"M";1;"KTYME6"CODE
100: RANDOMIZE USR VAL "26624"
200: LET A=VAL "26210"
```

```
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","0","128","17","0","192","1" ,
          "0","64","237","176","62","16","1","253","127","237","121","251","201","*"
400: POKE VAL "26212",A: PRINT  AT  5, NOT PI: LOAD*"M";1;"KTYME1"CODE:RANDOMIZE USR VAL
     "26210": RETURN
410: POKE VAL "26212",A: PRINT  AT  5, NOT PI: LOAD*"M";1;"KTYME2"CODE:RANDOMIZE USR VAL
     "26210": RETURN
420: POKE VAL "26212",A: PRINT  AT  5, NOT PI: LOAD*"M";1;"KTYME3"CODE:RANDOMIZE USR VAL
     "26210": RETURN
430: POKE VAL "26212",A: PRINT  AT  5, NOT PI: LOAD*"M";1;"KTYME4"CODE:RANDOMIZE USR VAL
     "26210": RETURN
440: POKE VAL "26212",A: PRINT  AT  5, NOT PI: LOAD*"M";1;"KTYME5"CODE:RANDOMIZE USR VAL
     "26210": RETURN
```

Save this to microdrive with:- SAVE*"M";1;"KNIGHTYME" LINE 10. We must now load each code block into memory from the tape, and save them to microdrive. Start by resetting the computer. Select 48K mode. Type CLEAR 26200 to keep the loaded code safe above RAMTOP. Next, load the SCREEN code block by typing LOAD "" CODE 32768,6912. PLAY in the block. Save to microdrive with:- SAVE*"M";1;"SCREEN"CODE 32768,6912. Next, load in the next block with:- LOAD "" CODE. PLAY it in. Save to microdrive with:- SAVE*"M";1;"KTYME1"CODE 32768,16384. Load in the next block with LOAD "" CODE. Again, save to microdrive with:- SAVE*"M";1;"KTYME2"CODE 32768,6912. Load in the next block with LOAD "" CODE. Save with:- SAVE*"M";1;"KTYME3"CODE 32768,16384. Load in the next block with LOAD "" CODE. Save with:- SAVE*"M";1;"KTYME4"CODE 32768,16384. Load in the next block, again using LOAD "" CODE. Save with:- SAVE*"M";1;"KTYME5"CODE 32768,16384. We now come to the last block. Reset the computer. Insert the second microdrive cartridge, type CLEAR 26200. Load the final block with LOAD "" CODE. Save with:- SAVE*"M";1;"KTYME6"CODE 26240,39296. That's it! The game will reload by selecting 128K Basic, and type:- LOAD*"M";1;"KNIGHTYME". After the first five blocks have loaded, the microdrive will stop - during the pause insert the second cartridge, and press any key to load the final block of code.

Using a similar approach, other 128K games may be transferred to microdrive.

"KOBRAHSOFT KD1 DISASSEMBLER"

INSTRUCTIONS FOR USE

Introduction.

The KD1 disassembler is an efficient and easy-to-use disassembler for your 48K Spectrum. It can be used to list machine code instructions on your T.V. Screen, or output them to a ZX Printer. It allows 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, either by disconnecting then reconnecting the power supply, or by pressing the RESET switch - whichever is applicable. Next:-

FOR THE 48K SPECTRUM:- Type LOAD "" ; press ENTER, then PLAY on your recorder.

FOR THE 128K SPECTRUM:- Select 48K Mode from the opening Menu; proceed as above.

KD1 will then load and run, displaying the message:- "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. Any attempt to do so will produce the message:- INVALID ADDRESS. To enter the disassembler mode, press "D". Next, enter, IN 4 HEX. DIGITS, your required start address. You can now also specify similarly a 4 HEX. DIGIT end address. If you don't, the only difference is that KD1 will continue its disassembly up to 65535 (FFFFH), which is probably inconvenient especially when outputting to your printer! In the case of no specified end address - press ENTER. You are then asked if you require a printout to the ZX Printer. If yes - press "\", if no - press N, when the listing will appear on the screen. To list more - press ENTER. To end the listing - press F.
When an end address is specified - ENTER is not needed - the enquiry PRINT? appears on pressing the last digit. NOTE:- To get "\": press Symbol Shift and "V" keys.
If you do NOT specify an end address, you can stop a continual printout to a ZX Printer by pressing "BREAK". This will return you to Basic, with the usual error message.

Return to Basic. (R).

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

"HEADER READER"

INSTRUCTIONS FOR USE.

LOADING:-

Proceed as for KD1 above. The program will load, then auto-run.

USING:-

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

FILENAME:- The program name. This may sometimes be printed vertically due to the 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 program length (Basic program length + Variables), and the normal program length. It will also 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:-

FOR THE 48K SPECTRUM:- Type LOAD ""; press ENTER, then PLAY on your recorder.
FOR THE 128K SPECTRUM:- Select 48K Basic at the opening Menu, load as for 48K.

The program will run, then load the machine code. 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).

"ALKATRAZ DECODER AD3"

INSTRUCTIONS

INTRODUCTION.

Our new AD3 program enables the EASY transfer of the VERY LATEST Alkatraz programs using the following procedures:-

(1). Load AD3, the PLAY in the Alkatraz tape.

(2). When loaded, press BREAK, which saves 3 code blocks to a new tape.

(3). Load the transfer program "Transfer"; put a formatted cartridge in the drive then PLAY in the saved tape - after loading, the program is automatically transferred to your cartridge, and reloads by pressing key "R" for RUN.

The FULL details are now given below:-

LOADING AD3.

For the 48K Spectrum:- RESET. Then type LOAD "" and press ENTER and PLAY in AD3.

For the 128K Spectrum:- RESET. Select 48K Basic and proceed as for the 48K method.

PLEASE NOTE:- BEFORE loading AD3 you must ALWAYS RESET your Spectrum to clear the memory When AD3 has loaded, the message "PLAY THE TAPE" appears. Remove your SD5 tape, and insert the tape you wish to transfer into your recorder. Rewind it to the start and press PLAY. When the first long part has loaded, the border will flash as AD3 decodes the Alkatraz, and a few coloured squares appear toward the top of the screen, but this is normal. Now leave the tape PLAYing. The program should load as normal. When loaded - this is usually when the on screen counter reaches zero - remove your game tape and insert a new tape. Press REC + PLAY to save then press BREAK - 3 headerless blocks will be saved to your tape. On completion of saving, the game will run as normal - play it if you wish. Rewind the saving tape. RESET the Spectrum, and load the "Transfer" program from your SD5 tape using the same method as for AD3 above. When loaded, the screen goes BLACK and the Spectrum goes into tape loading mode. Put a formatted cartridge in the microdrive. Insert your saved tape and press PLAY. "Transfer" will load the 3 blocks. NOTE:- When the blocks are loading some distortion of the upper part of the screen will appear, but this is normal. When the tape has loaded, "Transfer" will AUTOMATICALLY transfer the program to microdrive together with a file called "run". The computer will then reset. To reload the program from cartridge, simply press the "R" key to give the RUN keyword and then press ENTER. The game will load and run as normal.

"FIREBIRD DECODER FB1"

INSTRUCTIONS

INTRODUCTION.

Our new FB1 program enables the EASY transfer of those slow loading "multiple small block" programs from "FIREBIRD", using the following procedure:-

(1). Load FB1, then PLAY in the Firebird tape.

(2). When loaded, press BREAK, which saves 3 code blocks to a new tape.

(3). Load the transfer program "Transfer"; put a formatted cartridge in your M/D then PLAY in the saved tape - after loading, the program is automatically transferred to your cartridge and reloads by pressing key "R" then ENTER.

The FULL details are now given below:-

LOADING FB1.

For the 48K Spectrum:- RESET. Type LOAD "" then press ENTER and PLAY in FB1.

For the 128K Spectrum:- RESET. Select 48K Basic - proceed as for 48K above.

PLEASE NOTE:- BEFORE loading FB1 you must ALWAYS RESET your Spectrum to clear the memory When FB1 has loaded, the message "PLAY THE TAPE" will appear. FB1 is now ready to use. Put the tape for transfer into your tape recorder. Rewind it to the start and press PLAY. Continue PLAYing the tape until it has completely loaded. When the loading has finished, (Loading counter reaches a maximum then stops), remove the Firebird tape and insert a new one. Press REC+PLAY to save then press BREAK - 3 headerless blocks will be saved to your tape. On completion of saving, the game will run as normal - play it if you wish. Rewind the saving tape. RESET the Spectrum, and load the "Transfer" program When loaded, the screen goes BLACK and the Spectrum goes into tape loading mode. Put a formatted cartridge in the microdrive. Insert your saved tape and press PLAY. "Transfer" will load the 3 blocks. NOTE:- When the blocks are loading some distortion of the upper part of the screen will appear, but this is normal. When the tape has loaded, "Transfer" will AUTOMATICALLY transfer the program to microdrive together with a file called "run". The computer will then reset. To reload the program from cartridge, simply press the "R" key to give the RUN keyword and then press ENTER. The game will load and run as normal .