SRAM HI*RES
AN EXTENDED BASIC PACKAGE
FOR ZX81/TS1000 with STATIC RAM from $2000 to $3FFF
or
TS1500 with STATIC RAM plus 32K or more of RAM

by

Gregory C. Harder
and
Fred Nachbaur

WRX16 core by Wilf Rigter

Distributed by:
SILICON MOUNTAIN COMPUTERS
C-12, Mtn. Stn. Group Box

Nelson, BC VIL SP1
Canada

ACKNOWLEDGEMENTS

If you have ever written a long program, especially one in machine-code, then you know that not everything you write can be 100% original. Unfortunately, it is impossible to give credit to everyone who helped along the way. However, a few names do come to mind, including W. Rigter for the WRXI6 core routines, R, Kingsley for HOT 2, H. Doakes for the line and circle algorithms, T. Baker for her book, D. Woods for his book Cambridge for the 2X ROM, and KMART for selling us our first TS1000.

INTRODUCTION

Welcome to SRAM HI*RES. This is a compact package of machine-language routines designed to maximize the usefulness of your ZX81/TS1000/TS1500 with STATIC RAM in the $2000 to $3FFF area (8-16K). It gives you 38 new commands (sone with several variants) that you can call from BASIC. An extensive DEMO is included to help you get the feel of the package, and to demonstrate some of the techniques you can use. You don't have to know the first thing about machine-code to use SRAM HI*RES to the fullest. If you know how to program in Sinclair ZX BASIC, you will very quickly learn how to use the new SRAM HI*RES commands.

A single USR entry point accesses all of the routines, and a unique method of passing parameters lets you use virtually any expression that is valid in standard Sinclair BASIC. No POKEing or PEEKing around to print variables, etc.!

This manual fully details the operation of SRAM HI*RES, Study it carefully, as there is a great deal of information here.

BACK-UP POLICY

Since you're buying this package as a programmer's utility, you have some duplicating rights not normally granted for applications software. You may include the SRAM HI*RES core in your applications program for sale or other distribution, under the following conditions:

1: A copy of your program must be sent to Silicon Mountain Computers.

2: Instructions for extracting or using SRAM HI*RES must not be included in your documentation. No part of this manual or the DEMO program may be included without written authorization.

This only applies to material you'll be circulating, whether free or for payment. Provision #1 does not apply, of course, to software that you write strictly for your own use. The point is, that while SRAM HI*RES is sold as a programmer's utility, it is NOT to be considered "public domain." Exercise the honesty and fairness that has become the hallmark of (most) ZX/TS computerists, and all will be well.

If the tape you received is defective, we will of course replace it at no charge. However, you are expected to make sufficient backup copies of the programs to insure that you will not lose them. Tapes CAN and DO go bad, so don't put it off.

SRAM BASIC SYNTAX

We've gone to considerable effort to make you feel right at home when programming in SRAM BASIC. All the commands, which have a direct Sinclair analog (PRINT, PLOT, CLS, etc.) are used almost exactly Like their Sinclair counterparts. The new Commands are defined, so as to be mnemonically 'sensible', so you won't be forever referring to the command summary, Best of all, most commands support ANY expression (including ANY variables) allowed in the Sinclair convention.

All the SRAM BASIC commands use a single USR entry point, of the form: IF USR HR THEN ... (command)

At the start of your program, you must assign the correct value to HR using a LET statement: this value is 19400. Instead of HR you could use any other variable name; I chose HR because any line containing HR will obviously be a HI*RES operation. There's no reason why you couldn't use a single-letter variable instead, to slightly reduce your program's 'overhead'. However, use caution when typing in your lines, a typo error could be potential disaster. For example, if you miss-type the USR call and the resultant variable has a different valve than 19400 then you could have a runaway USR which more than likely will crash.

If the syntax, "IF USR HR THEN ..." seems add, rest assured that there were very good reasons for choosing it. Most of you have used RAND USR .., but may not realize that it has one big drawback - it can make the computer's randomizing function virtually useless. If the reason is obscure, remember that USR is a function, returning a value. When using RAND, this value is placed into the SEED variable of the random number generator. As a result, games and simulators that use the random number generator can get "stuck" since SEED is continually reset to the same value by RAND USR.

Another significant advantage to IF USR...THEN is that the entire command, including any parameters, can be expressed in a single BASIC line. In SRAM BASIC, there are no extra REM lines needed to pass parameters. This considerably reduces program 'overhead', while making your HR command lines easy to read.

I should note that SRAM HI*RES always sets the BC register to zero before returning to BASIC, so the logical result of the USR will ALWAYS be false. In other words, the part after the THEN is NEVER executed! It is only used to tell SRAM HI*RES what to do, and is thus completely transparent to BASIC.

Finally, this method of USR access makes the best use of your ROM's resident syntax-checking features with most commands. For example, if you say,

```
IF USR HR THEN PLOT A$, 10    or
IF USR HR THEN PRINT "HI
```

your line will automatically be rejected as invalid before you can even get it up into the listing, The inverse "S" syntax cursor will mark the error (trying to plot a string in the first case, missing quote mark in the second).

This philosophy of 'letting the ROM do the dirty work' extends throughout SRAM HI*RES, and is largely responsible for its flexibility, speed and low overhead. ROM routines are used to scan your SRAM BASIC lines, evaluate expressions, report on errors, and perform sundry other housekeeping tasks. Why re-invent the ZX?

SRAM HR*RES MEMORY MAP

Besides the memory used by the core routines, almost all of the 8K from $2000 to $3FFF is used for the HR-DFILE, various DATA buffers, and HR SYSTEM VARIABLES.

FROM	TO	NAME	COMMENT
$2000	$37FF	HR-DFILE	The high resolution display file. (8192-14335)
$3800	$3BFF	BUF1	The UDG (MODE 2) buffer. (14336 - 15359)
$3C00	$3DFF	BUF2	The MODE 1 character buffer. (15360 - 15871)
$3E00	$3EFF	BUF3	Sprite pattern buffer. (15872-16127)
$3F00	$3FBF	SPARE	Unused area of memory. (16128 - 16319)
$3FCO	$3FFF	HRVAR	The HR system variables. (16320 - 16383)
$407D	$4D20	CORE	The core routines.

The SPARE area of memory, 191 bytes, is an excellent place to put some of your own M.C. routines, as this area will be SAVEd and LOADed by the SAVE S and LOAD S commands,

The core consists of three REM statements that must be at the beginning of your program. These are 0 REN (main core routines), 1 REM (F-save package} and 2 REM (64-column PRINT). Lines 1 and 2 may optionally be deleted if you don't need these features; see appendix 5.

Besides the above, there is one other area of memory used, 45 bytes long, which is part of SRAM HI*RES. This is line G000 REM which must always be present when running the core routines.

HR SYSTEM VARIABLES

Here is how the HR SYSTEM VARIABLES are organized. Some of then (MODE, CRD3, XPYP) can be usefully POKEd.

ADDRESS___NAME____COMMENTS:

$3FC0 MODE The character mode currently in force. (16320)
$3FC1 CRD3 The coordinates of the next print pos.(y=16321/x=16322)
$3FC3 XPYP The last point plotted, drawn to, or
to which a SPRITE was moved. (y=16323,x=16324)
$3FC5 DFLG DRAW/UNDRAW flag. (16325)

$3FC6 to $3FCE SPARE*1 Unused area. (16326 - 16334)

$3FCF to $3FDE TEMS Temporary variables. (16335 - 16350)
$3FDF to $3FE2 SPARE*2 Unused area. (16351-16354)

$3FE3 to $3FF2 BKG Background pattern, (16355-16370)
$3FF3 to $3FFF SPARE*3 Unused area, (16371 - 16383)

$4078 HRFLAG Various flags. Oo not POKE. (16504)

COMMAND GROUPS

SRAM HI*RES BASIC commands fall into two basic catagories, or "groups".

GROUP 1 commands are all of the form,
IF USR HR THEN (Keyword) (parameters if needed)
The keyword will be a standard Sinclair token; many of the commands are directly analogous to the Sinclair commands, The rest have a different meaning.

GROUP 2 commands are of the form,
IF USR HR THEN LPRINT (command);(parameters if needed)

Why LPRINT? Though it may seem that we're doing a lot of printer operations, this is not the case. LPRINT is used because it supports syntax constructs exactly like the PRINT command; as a result, it is easy to pass as many parameters as we need for a particular command. Furthermore, numbers after LPRINT are stored in binary form, for better run-time operating speed. Also, commas and semi-colons are permitted as separators, making the parameters easy to read. Finally, a great deal of syntax-checking is done on entering such a line, reducing the amount of syntax checking that SRAM HI*RES has to do. You won't get reams of garbage sent to your printer by the group 2 commands, since the LPRINT is never actually executed by BASIC.

A few of the Group 1 commands can be used to advantage with reversal of the logic; e.g., IF NOT USR HR THEN CLS, in such a case, the command after the THEN is executed. The commands that could be of such use will be pointed out as we go along.
* Note that SRAM HR*RES commands can be combined with other IF...THEN statements, For example, IF N=5 THEN IF USR HR THEN CLS will only clear the HR-DFILE if N=5. Similarly, IF NOT F THEN IF USR HR THEN CLS will only clear the HR-DFILE if F=0.

THE GROUP 1 COMMANDS
1: RUN
GENERAL FORM: IF USR HR THEN RUN
This command will display the contents of the HR-DFILE. Resolution in this mode is 192 rows by 256 columns. Keep in mind, when in this mode you will not be able to see any of the normal run-time error reports. If your program appears to stop, there is probably an error contained in your BASIC listing, such as an undefined variable, nonsensical VAL etc. In such a case, just press "edit" (shift 1) followed by ENTER to get back to the normal display. (WARNING: Do not leave the program cursor on one of the m/c REM lines, or using EDIT will ruin it.) If you have the "Bent ROM" upgrade, simply press ENTER to get to the normal display. Since the HR-DFILE can't be seen if in FAST mode, this routine will first call SLOW before producing the display.

2: CLEAR
GENERAL FORM: IF USR HR THEN CLEAR
This command is like RUN, except that only 176 lines are displayed. Think of this as RUNning with the bottom rows CLEARed. The purpose is to open a small "window" into the normal display file. Therefore, run-time error reports will be displayed, making it very useful for program debugging. Also, the INPUT cursor and your inputs with be visible.

Even though you can't see the bottom of the HR-DFILE when in CLEAR mode, most commands will still operate on a full size display file, The exceptions are PRINT and COPY.

3: CLS

GENERAL FORM: IF USR HR THEN CLS

Clears the HR-DFILE and sets the HR PRINT position to the top left-hand corner of the screen (0,0).

Inverted logic can be useful with this command, should you want to also clear the normal display file at the same time.

4: RETURN

GENERAL FORM: IF USR HR THEN RETURN

Returns to the normal display mode. You would normally call this at the end of your program, or any time you want to view the contents of the normal display file.

5: PLOT

GENERAL FORM: IF USR HR THEN PLOT x,y

This command sets the pixel at the specified x,y coordinate. The "x" coordinate may be in the range of 0-255, and the "y" coordinate in the range of 0-191. As with the Sinclair command, the origin (0,0) is taken as the lower left-hand corner of the screen. If the "x" coordinate is out of range, you'll get error report H, if "y" is out of range you'll get V.

6: UNPLOT

GENERAL FORM: IF USR HR THEN UNPLOT x,y

This is the exact inverse of the PLOT command. It unsets the pixel at coordinate x,y.

7: COPY
GENERAL FORM: IF USR HR THEN COPY

This command copies the displayed portion of the HR-DFILE to the TS2040 printer.

IF NOT USR HR THEN COPY will dump first the HR-DFILE, then the normal display file.

8: SAVE
GENERAL FORM: IF USR HR THEN SAVE name,SCREEN
IF USR HR THEN SAVE name,S
IF USR HR THEN SAVE name,VARS
IF USR HR THEN SAVE name,V
IF USR HR THEN SAVE name,PROGRAM
IF USR HR THEN SAVE name,P
This command saves the specified area of memory, at a greatly increased speed. This fast-save routine is quite probably the most reliable such program ever written; a lot of research went into making it work on any machine and any decent tape recorder while being fast, compact, and relatively bomb-proof. So if you've had dubious or no results when using other fast-load tape packages, don't automatically assume that this one won't work either. try it! We think you'll like it.

The SAVE name may be a literal string (e.g. "TESTFILE") or any string expression (e.g., N$). A name must be given; maximum length is 32 characters.

The suffix (S, V or P) specifies the type of save. S (or SCREEN) saves the screen data (including all buffers) from $2000-$3FFF. V (or VARS) saves the entire variables area (only), and P (or PROGRAM) saves the program + variables area (analogous to the normal SAVE). There is no way to break from any of the SAVE commands; if you change your mind in mid-stream, you'll have to wait until the SAVE is complete before you can regain control.

Since the tape routines are in the 16-32k region of memory specifically, in the 1 REM statement), you can only load a program if an SRAM HI*RES application is already loaded. To make it possible for you to fast-load your applications, a "BOOT" program is included on the tape. This installs only the bare minimum of code required to fast-load a program, and therefore loads in under a minute. So you will probably want to start each of your applications tapes with a copy of BOOT, so that you can rapidly get your applications up and running after powering up.

9: LOAD
GENERAL FORM: IF USR HR THEN LOAD name,SCREEN (or S)
IF USR HR THEN LOAD name,VARS (or V)
IF USR HR THEN LOAD name,PROGRAM
This command is the inverse of the SAVE command. Unlike SAVE, however, the LOAD command monitors the BREAK key until the named file is found, so if you are trying to load a nonexistant file, you can abort the attempted LOAD. The BREAK Key only works if a signal is being received. If you press BREAK and nothing happens, Turn on your recorder and play something - anything. The first pulse that is received will activate the BREAK key.

WARNING: Be careful to specify the proper LOAD type (S, V or P). If you try to load a screen file as a program, for instance, you will almost certainly crash.

Further information on the tape routines are in appendices 1 and 5.

10: PAUSE
GENERAL FORM: IF USR HR THEN PAUSE nnnn

When in HR mode you can not use the normal PAUSE command, since it causes a screen update which will automatically return you back to the normal display mode. This command is a replacement which is nearly analogous to the normal PAUSE. This routine pauses the program for the specified length, nnnn. Also, like the normal Sinclair command, the pause is terminated if you press a Key.

11: LIST

GENERAL FORM: IF USR HR THEN LIST
IF USR HR THEN LIST n

This command is used to transfer all or part of the normal display file to the HR-DFILE, using the current character mode in force. The first form will transfer the entire normal display file, all 24 lines, to the HR-DFILE. The second form will transfer n lines (0 to 23) to the top part of the HR-DFILE.

12: LLIST

GENERAL FORM: IF USR HR THEN LLIST
IF USR HR THEN LLIST n

This is similar to LIST except that the lines transferred will be to the bottom part of the HR-DFILE.

13: PRINT

GENERAL FORM: IF USR HR THEN PRINT (print items)

Like most of the other SRAM HI*RES commands, anything that goes in the normal ZX/TS PRINT syntax is perfectly valid here. It should therefore not prove too difficult to upgrade existing BASIC software to HR format.

Printing to the HR-DFILE follows the normal Sinclair convention almost exactly. You can PRINT string variables, numeric variables, arrays, etc. Anything that is permissible in the normal SINCLAIR PRINT SYNTAX is usable with HI*RES PRINT, even tokens. You can use AT and TAB as before, even with expressions. COMMA can be used to advance the print position to the next screen quarter. In other words, comma now advances 8 spaces of normal characters, or 16 spaces of 64-column characters. (This change was made because of the addition of the 64-column mode.)

There is some added versatility. The PRINT AT r,c coordinates can range from r=0 to r=184 and c=0 to c=63, so you can start printing at any of the 184 lines. Print formatting would be identical to the normal Sinclair format by multiplying the normal y coordinate by eight, and the normal x coordinate by 2. PRINT AT 3,5 in normal mode is the same as IF USR HR THEN PRINT AT 24,10 in the HR mode. The horizontal coordinate was doubled to allow full control in the 64-column mode. Note, however, that in the 32-column modes only even-numbered positions are permitted; an odd number will cause 32-column characters to be printed at the next even position instead.

The SEMI-COLON is an important construct within the PRINT line, as it is used to toggle between the various PRINT MODEs.
You have four different PRINT MODE options available to use. MODE 0 will print all characters to the screen in the normal SINCLAIR fashion, that is, all upper case or inverse characters,

MODE 1 will use the alternate character set stored in the BUF2 area of RAM. These are meant to be a semi-permanent character set, such as lower case letters and special symbols. These would normally be loaded into the buffer before you start your own programming. There are no commands from BASIC to change these patterns.

MODE 2 is used to print your User Defined Graphics, which are stored in the BUF1 area of RAM. These patterns can be changed by using the LPRINT UDG; command.

MODE 4 is used to switch to the 64-column character mode. This mode is otherwise similar to mode 1, i.e. you can print capitals (inverse video) and lower-case (normal video) characters.
You can toggle between the four print MODEs by using the SEMI-COLON; and COMMA:
; maintains print position
;; switches to print MODE 0
;;; switches to print MODE 1
;;;; Switches to print MODE 2
;, switches to print MODE 4

For example:
IF USR HR THEN PRINT;;”ABCD”;;;”EFGH”;;;;”IJKL”;,”MNOP”;;

In this example, ABCD would be printed in MODE 0, EFGH in MODE 1, IJKL in MODE 2, and MNOP in MODE 4. The final two semi-colons restore print MODE 0 before exiting the line. The last print mode used will stay in force until changed. Note that the 64-column mode (MODE 4) uses a slightly different syntax; this is is mark the considerable difference between this mode and the other three.

Note that if you use commas to advance your print position, any MODE-change characters should be the LAST ones in the sequence. For instance, if you want to advance 3 screen quarters and switch to mode 2, you would say IF USR HR THEN PRINT,,,;;;;”text”. Tf you said IF USR HR THEN PRINT;;;;,,,”text”, this would be interpreted as a command to select mode 4, followed by an advance of two screen quarters.

There is a}so a second way of changing the print MODE, this is by POKEing the HR system variable MODE with the mode wanted. MODE is located at 16320. Specifically,

POKE MODE,0
POKE MODE,1
POKE MODE,2
POKE MODE,4

will change the print mode as indicated. Any other values (e.g. 3) will give an error report when subsequently trying to PRINT.

Another nice feature of the new PRINT command is that it automatically scrolls the display upward if you run out of room at the bottom, In this respect, the PRINT command operates as it does on the Spectrum and 2048, (Unlike these machines, however, we still have the SCROLL command, so don’t have to use PRINT to accomplish this.) The auto-scroll is especially useful when printing from within a loop (see the demo). However, an out-of-range vertical AT coordinate will produce an error report.

14: RAND
GENERAL FORM: IF USR HR THEN RAND
IF USR HR THEN RAND n

This is a screen-reverse command. The first general form is used to restore the screen to the default black characters an white background, The second general form reverses from column 0 to column n. Note that the column reference for RAND is in the traditional 32-column format, i.e. 0-31, not the 64-column format used by PRINT.

You should know that this command is not the same as the INVERSE command described later. Unlike INVERSE, RAND doesn't actually invert the contents of the HR display file. Instead, it modifies how the screen data is displayed. Consider it a software-only "video reverse switch." Another way to explain the difference is with an example. Let's say you used CLS followed by INVERT to create a black screen. Any printing, plotting, etc, that you do subsequently will not be visible. However, if you used RAND to create the black screen, printing and plotting would show in white.

Successive uses of RAND can be employed to make "panels" in opposite ink/background colors. See the demo for an example. Since RAND is very fast, it can also be used as a FLASH command; again, refer to the demo for one possible (and rather startling) display of this nature.

RAND n works by reversing the "dummy display file" (REM line G000), so the present RAND status will be saved to tape with your program. RAND restores the "dummy display file" to zero (normal video).

Do not use RANO from normal video mode after making changes to your program. To be safe, call RAND only when in high-res mode (RUN or CLEAR).

Reverse logic (IF NOT USR HR THEN RAND) will execute the RAND (to set the random number seed) after reversing or restoring the high-res screen.

15: SCROLL
GENERAL FORM: IF USR HR THEN SCROLL

Works just like the comparable Sinclair command, Scrolls up 1 row (8 lines) and sets PRINT position to the last printable row (184 in RUN mode, 174 in CLEAR mode).

THE GROUP 2 COMMANDS

As noted previously, these commands are all of the form;
IF USR HR THEN LPRINT (command);(parameters as required)
A nice feature of these commands is that all the GROUP 2 commands can be used within MULTIPLE STATEMENT LINES!

USING THE MULTIPLE STATEMENT LINES
The LPRINT method of parameter passing lends itself readily to implementing multiple statement lines. As long as each of the commands are set up properly, SRAM HI*RES will process each command sequentially until the end of the line is found.

The SEMI-COLON is used as the item separator. A COLON would have been ideal, but is not allowed in Sinclair BASIC.

Here is an example of the DRAW demo used in the demonstration program in long form:

```
10 FOR Y=0 TO 191 STEP 8
20 IF USR HR THEN LPRINT 0;0,Y,255,191-Y
30 NEXT Y
40 FOR Y=0 TO 191 STEP 6
50 IF USR HR THEN LPRINT D;Y,0
60 IF USR HR THEN LERINT D;255,Y
70 IF USR HR THEN LPRINT D;255-Y,191
80 IF USR HR THEN LPRINT D;255-Y,191
90 IF USR HR THEN LPRINT D;255-Y,191,0,191-Y
100 NEXT Y
```

Now the same routine using multiple statement lines;

```
10 FOR Y=0 TO 191 STEP 8
20 IF USR HR THEN LPRINT D;0,Y,255,191-Y
30 NEXT Y
40 FOR Y=0 TO 191 STEP 6
50 IF USR HR THEN LPRINT D;Y,0;D;255,Y;D;255-Y,191;D;255-
   Y,191,0,191-Y
60 NEXT Y
```

Each of the LPRINT commands can be abbreviated to one or two character "word", but may also be "real" words. As long as the command characters are in their right position SRAM HI*RES will just ignore the extra characters. For instance:

```
LPRINT D;0,0,0,255,0
LPRINT DR;0,0,255,0
LPRINT DRA;0,0,255,0
LPRINT DRAW;0,0,255,0
```

or even
LPRINT DOGGONEITWHATTHEHECK;0,0,255,0

will all do the same thing, in this case, it's only the first character "D" which is being evaluated. In the following command summary I have included, in parenthesis the recommended long form for each command, the short form is shown in the GENERAL FORM lines.

It is important that all of your command words, long or short, end in a SEMI-COLON, as this tells SRAM HI*RES when the command ends and to start processing the parameters.

16: POINT (POINT)
GENERAL FORM: IF USR HR THEN LPRINT P;x,y,n

If the pixel at coordinate x,y is set, then POINT will return a value of 1 to the variable n. If the pixel is unset then n will equal 0.

There are three restrictions to Keep in mind when using POINT;
one, the designated variable n must have been previously assigned;
two, the variable n can only be a single character variable; and
three, the variable n can not have been used as an earlier FOR-NEXT variable.

17: LOCATE (LOCATE)
GENERAL FORM: IF USR HR THEN LPRINT L;x,y,n

LOCATE will return in the n variable the value of the byte contained at the HR-DFILE address controlled by coordinates x,y. The sane restrictions apply to n as noted in POINT.

18: ADDRESS (ADDRESS)
GENERAL FORM: IF USR HR THEN LPRINT A;x,y,n
ADDRESS will return to variable n the address within the HR-DFILE corresponding to the x,y coordinates.
The same restrictions on n apply here as well.

19: BINARY (BINARY)
GENERAL FORM: IF USR HR THEN LPRINT B;r,c,d
BINARY will print in binary form the decimal value d, 0 to 255, at row r and column c.
* Note that the r and c values are PRINT coordinates, NOT PIXEL coordinates, so r would range from 0 to 191, and c would range from 0 to 31. A value of r=0,c=0 would print the binary character at the top left side of the screen. This command,

PRINT and RAND are the only ones that use this type of coordinate system, all the others are referenced by PIXEL coordinates.

20; DRAW (DRAW)
GENERAL FORM: IF USR HR THEN LPRINT D;x1,y1,x2,y2
IF USR HR THEN LPRINT D;x2,y2

This quickly draws an unbroken line between coordinates x1,y1 and x2,y2. Note that this uses ABSOLUTE coordinates, which are generally easier to use in graphing situations than the relative coordinates employed by some other machines. (Keep this in mind when translating 2068 or Spectrum programs.)

After the line has been drawn, the x2,y2 values become the new x1,y1 coordinates. This makes possible the second form, employing only two parameters. In this form, the line is drawn from the current PLOT position (or the previous x2,y2, as the case may be) to the new x2,y2. This makes it very easy to connect the dots on a graph without having to do the “variables shuffle".

21: UNDRAW (XDRAW)
GENERAL FORM: IF USR HR THEN LPRINT XD;x1,y1,x2,y2
IF USR HR THEN LPRINT XD;x2,y2

This is the inverse of DRAW, and erases the line between x1,y1 and x2,y2 or the current PLOT position and x2,y2. As with DRAW, either the full or abbreviated form may be used.

22: CIRCLE (CIRCLE)
GENERAL FORM: IF USR HR THEN LPRINT C;x,y,r

This command draws a circle of radius r with its center at x,y (in PLOT positions).

If a circle spills over the left or right edge, it wraps around to the other side. If it spills off the top or bottoa, it simply disappears.

23: UNCIRCLE (XCIRCLE)
GENERAL FORM: IF USR HR THEN LPRINT XC;x,y,r

This is the same as CIRCLE except that it "undraws” the circle of radius r, with center at x,y.

24: RECTANGLE (RECTANGLE)
GENERAL FORM: IF USR HR THEN LPRINT R;r1,r2,c1,c2

This will draw a rectangle from r1 to r2 and c1 to c2, where r1 = row 1, r2 = row 2, c1 = column 1, and c2 = column 2, all in PLOT coordinates.

25: UNRECTANGLE (XRECTANGLE)
GENERAL FORM: IF USR HR THEN XR;r1,r2,c1,c2
Undraws the rectangle from r1 to r2 and c1 to c2.

26: TRIANGLE (TRIANGLE)
GENERAL FORM: IF USR HR THEN LPRINT T;x1,y1,x2,y2,x3,y3
Draws a triangle defined by the three vertices.

27: UNTRIANGLE (XTRIANGLE)
GENERAL FORM: IF USR HR THEN LPRINT XT;x1,y1,x2,y2,x3,y3
Undraws the triangle.

28: INVERT (INVERT)
GENERAL FORM: IF USR HR THEN LPRINT I;
IF USR HR THEN LPRINT I;n
IF USR HR THEN LPRINT I;n,r1,r2,c1,c2

These are used to "invert" all or part of the HR-DFILE. Form one inverts the entire screen. Form two inverts the entire screen n times. Form three inverts the rectangular ares defined by r1,r2,c1,c2, n times.

29: WINDOW UP (WINDQWU)
GENERAL FORM: IF USR HR THEN LPRINT WU;
IF USR HR THEN LPRINT WU;n
IE USR HR THEN LPRINT WU;n,r1,r2,c1,c2

These routines are used to scroll all or part of the HR-DFILE up. Form one scrolls the entire HR-DFILE up by one line. Form two scrolls the entire HR-OFILE up by n lines. Form three scrolls the rectangular window defined by r1,r2,c1,and c2 up by n lines.

30: WINDOW DOWN (WINDOWD)
GENERAL FORM: IF USR HR THEN LPRINT WD;
If USR HR THEN LPRINT WD;n
IF USR HR THEN LPRINT WD;n,r1,r2,c1,c2

This is the same as WINDOM UP except that the scrolling action is down.

31: WINDOW RIGHT (WINDOWR)
GENERAL FORM: IF USR HR THEN LPRINT WR;
IF USR HR THEN LPRINT WR;n
IF USR HR THEN LPRINT WR;n,r1,r2,c1,c2

These will scroll all or part of the HR-DFILE right,

32: WINDOW LEFT (WINDOWL)
GENERAL FORM: IF USR HR THEN WL;
IF USR HR THEN LPRINT WL;n
IF USR HR THEN LPRINT WL;n,r1,r2,c1,c2
You guessed it, these will scroll all or part of the HR-DFILE left.

33; UDG (UDG)
GENERAL FORM: IF USR HR THEN LPRINT U; “c”,”hb,hh,,,,,hh”

This command is used to define UDG (User-defined graphic) characters. The first argument is the character being defined. This can be any of the normal printable Sinclair characters, followed by a string of hex numbers. For example, if you want to define a UDG that corresponds to the character “X”, you might enter:

IF USR HR THEN LPRINT U; ”X”,"00,42,24,18,24,42,42,00"

which will result in a rather strange-looking letter "X" when printed.

This command will continue processing hex numbers until it runs out of them. So, if you U;”A”; followed by 40 hex numbers, it will define UDG’s A through E. So the first argument is, more precisely, which UDG to start at. In principle, you could define your entice UDG set, a total of 128 characters, with a single WDG command, starting with U;”(space)"; followed by 1024 hex pairs. Such a mondo line would be a real pain to edit, however, and can’t be recommended. A better approach is to define, say, five or six at a time.

Note that the hex sequence must be enclosed in quotes. The separators between hex numbers can be anything you like except quote marks).

If your UDG set abruptly "shifts" up or down, you probably are missing a hex number somewhere, or inserted an extra one. Remember you must have eight, and only eight, numbers for each pattern.

The reason hex is used instead of decimal in this command is that it's EASIER. (Binary is easier yet, but would be terribly memory inefficient.) UDG (and sprite) patterns consist of eight numbers, one for each row within the character, starting at the top. Each number represents the binary bit pattern of the corresponding row, Black pixels represent "1's" and blank pixels represent "0's". So you would usually start by drawing out your character on an eight by eight grid on a piece of graph paper. Each row will eventually reduce to one number. Now draw a line (real of imaginary) straight down the middle of your picture, such that there are four columns on the left, and four on the right. The four pixels of each row on the left (half a byte, or 'nybble') represent the first hex digit, and the four pixels on the right represent the second hex digit. Simply convert each nybble to a hex digit according to the following table:

0000 - 0 0001 - 1
0010 - 2 0011 - 3
0100 - 4 0101 - 5
0110 - 6 0111 - 7
1000 - 8 1001 - 9
1010 - A 1011 - B
1100 - C 1101 - D
1110 - E 1111 - F

Once your UDG's are defined, you can print them by using PRINT;;;; or by POKE MODE,2.

SPRITES
Like UDG's, sprites are user-defined characters that can be placed on the screen. They are even defined like UDG's in SRAM HI*RES, using hex strings to specify their bit patterns. One major difference between the two is that sprites can be smoothly moved, a pixel at a time. The other is that sprites do not erase what is already on the screen; they are in a sense "transparent."
To best utilize the sprites, as implemented in SRAM HI*RES, you should visualize the video display as consisting of two 'planes', the background plane and the sprite plane. Up until now, you have been dealing only with one plane, that is, there was no way of moving or printing something to the screen without affecting what was already there. Sprites, however, live on a "higher" plane and can be moved independently of the background plane. In this sense, the sprite covers or shadows the background plane, As a result, you can get the illusion of 3-dimensional depth, in a manner much like old fashioned black and white animated cartoons.

SRAM HI*RES offers five commands to define and control sprites.
34: SPRITE DEFINE (SPRITED)
GENERAL FORM: IF USR HR THEN LPRINT SD;n,"hh,hh,...,hh"

This command defines a sprite pattern, much along the lines of the LPRINT UDG; command. The first parameter indicates which sprite number, 0 to 31, you wish to define. As with UDG, you can define as many of the sprites as you want within the hex string, you could, for example, define all 32 sprites at one time by letting n=0 followed by 256 hex pairs. As you may gather, you can define a maximum of 32 sprites.

35: SPRITE PRINT (SPRITEP)
GENERAL FORM: IF USR HR THEN LPRINT SP;n,x,y
This command will print sprite n at the specified x,y coordinates. The upper left corner of the sprite is taken as the reference point. I should note that the x and y values can be larger than the plot limits of the screen. This may seem odd, but there is a very good reason for this, as you will see when you read the next command description.

36: SPRITE MOVE (SPRITEM)
GENERAL FORM: IF USR HR THEN LPRINT SM;n,x1,y1,x2,y2
IF USR HR THEN LPRINT SM;n,x2,y2
This command can be used to smoothly move a sprite (# n) between any two screen positions. This is very similar to DRAW, as a matter of fact, they both use the same M.C. routine to calculate the move positions. As with DRAW, you can specify either two or four parameters for moving the sprite. If you specify only two

parameters, the last PLOT (or MOVE) position will be taken as the starting point. Note that you can change the sprite number at any time, thus changing the sprite character in “mid-stream”. As noted above, y values are not limited to the permissible plot positions. The reason for this is that it allows you to move the sprites off the screen (top or bottom) and wrap them around to the other side.

37: SPRITE SPEED (SPRITES)
GENERAL FORM: IF USR HR THEN LPRINT SS;
IF USR HR THEN LPRINT SS;n
This command controls the speed at which the sprite is moved. The first form automatically chooses the highest speed. (Consider it equivalent to SS;0.) The second form allows you to select the speed your most comfortable with. However, start with a low number, as very high values can be very slow,
The rate at which the sprite moves is 60/(n+1) bits per second. For example, a speed of 9 would cause the sprite to move at the rate of 6 bits per second, and would take 41.39 seconds to traverse the screen horizontal by (248 positions). As you might have guessed, the FRAMES system variable is used to mark time.

38: SPRITE ERASE (SPRITEE)
GENERAL FORM: IF USR HR THEN LPRINT SE;
Any time a sprite becomes stationary on the screen, as with SPRITEP or the last SPRITEM position, it will become part of the background plane if another sprite is printed or moved. This command will allow you to remove the last sprite printed or moved. If you do not erase a sprite immediately, it will become, permanently, part of the background plane.

Another point to keep in mind is that you shouldn’t PRINT or PLOT on top of a sprite, or immediately to the right. If you do, it will be erased when the sprite it removed, See the Earth-Moon-Sun demo; here we erase the moon sprite before plotting its position in the orbit.
When using this command in a multiple statement line, be sure to do it correctly, for example:
IF USR HR THEN LPRINT SE;D;0,255
is incorrect! This is the correct way to use it:

IF USR HR THEN LPRINT SE;;D;0,255

Note the two SEMI-COLONS after the command word, the first one signifies the end of the command, the second marks the start of a new command. The same also holds true for those commands with optional parameters, such as LPRINT I;, LPRINT WU;, LPRINT SS; . . . etc.

This ends the summary of the SRAM HI*RES commands. The main core routines (not including the removable tape and 64-column routines) take up a total of 3152 bytes, which averages out to just 88 bytes per routine! The total length of all routines is 4196 bytes, averaging to 110 bytes per routine.

ERROR REPORTS

SRAM HI*RES generates its own error codes, in addition to the ones you'd normally expect from the Sinclair operating system. All of the SRAM REPORTS will automatically return you to the normal display mode. The Sinclair report will not, and may or may not be visible, depending on which HR mode you are using.

REPORT ERROR CONDITION

D/ BREAK key pressed during printer operation or tape LOAD.

H/ Horizontal parameter out of range, PRINT, PLOT, DRAW, etc.

I/ Invalid command, or improper command syntax.

M/ Multiple statement line improperly formatted,
SEMI-COLON not used as a command separator.

P/ Parameter missing or COMA not used as a data separator.

S/ Sprite error, sprite number out of range.

U/ Expression in a UDG or SPRITED definition is not a hex number.

V/ Vertical parameter out of range, PRINT, PLOT, DRAW, etc.

2/ Variable undefined, POINT, LOCATE, or ADDRESS.

T/ Tape command has improper syntax, name missing or too long, or invalid or missing type suffix.

TECHNICAL INFORMATION
Presumably, if you purchased this program, you already have STATIC RAM available in the $2000 to $3FFF area of memory. If not, you can use either a modified “HUNTER BOARD" or some other type of SRAM memory board, see SYNCWARE NEWS volume 4 number 1 for details on construction of one such board for the ZX81/TS1000. Alternately, contact Silicon Mountain Computers for our SRAM NVM.
If using a ZX81 or TS1000, you will also need at least a 16K rampack. For hardware reasons, when using these routines on a TS1500, you will also need to add another memory pack, 16K or more, in addition to the SRAM (this would have to be a modified “HUNTER BOARD” or a stock Silicon Mountain board).

I should mention here something about using SRAM HI*RES with 64K rampaks. You may have been wondering what the GO00 REM line does, In fact, this line would not be necessary if all users of SRAM HI*RES were using a maximum of 48K of memory. If this was the case we could put the data stored there at a fixed location within the core routines with no problems. But as some users will, undoubtedly, be using 64K of ram, we want the data here to float in memory, that is we want it to be pushed further up into memory as our BASIC program gets longer.

Without going into too much detail about the display system, which could fi11 up another manual, the Sinclair ULA always expects to see an "ECHO" of the display file in high memory. The data contained within line G000 REM is our HI*RES "ECHO" of the display file. If you are using less than 64K of ram, this "ECHO” would be reflected into the high ram, automatically, by the lack of decoding. However, if using 64K, then we have to put the “ECHO" into high ram manually, in order for the routines to work. SRAM HI*RES does this automatically whenever RUM or CLEAR is called.
This means you should be aware of this data, 37 bytes, which is residing in upper ram when using 64K. Caution should be taken not to overwrite it by data or by your program, once in HR mode. Similarly, you should be aware of it so that it won’t overwrite important data you might have stored up there. The address of the start of this data can be found by;
PEEK 16396+256*PEEK 16397+32730

The reason we want the data to be floating in memory may be obvious now. This allows us to Keep the data transferred into high ram “ahead” of the actual program, thus removing some restrictions on 64K ram use.

In either case, 64K or less than 64K, line GO00 REM must always be present when using SRAM HI*RES.

PROGRAMS ON TAPE

The tape you received with this manual contains four separate programs.

HR*DEMO*1 This program should be LOADed prior to the HR*DEMO*2 program. This program just contains a full set of character patterns for defining the MODE 1 and MODE 2 characters. I would suggest you LOAD these character sets prior to starting any of your own programs to use the MODE 1 characters.

HR*DEMO*2 This is an extensive demonstration program which utilizes most of the HR commands. I suggest you make a hard copy of this and study it.

HR*CORE The SRAM HI*RES core routines, remove the extraneous lines and start programming.

BOOT This is a boiled-down version of the fast-load routine. Preface your own tapes with a copy of this program, and you'll be able to immediately fast-load your applications. Note that this will NOT work with programs that do not contain the SRAM HI*RES core.

If you get stuck with an automatic listing of line 0 each time you enter a program line, you will have to POKE S-TOP with a different value. First enter a line such as 10 REM, then LIST 10, then POKE 16419,10.

If you have any questions, complaints, comments or interesting programs you'd like to share, I can be reached at:

Gregory C. Harder
P.O. BOX 6493
DENVER, COLQRADO
80206

APPENDIX I
THE F-SAVE PACKAGE
The SAVE and LOAD routines used in the core are unlike any other fast-load tape package to date. Without getting into great detail, here's why.

1: F-SAVE saves a completely symmetrical signal to the tape This makes it usable with either the TS1000 or the TS1500, regardless of recorder playback phase or the presence of inverting pre-conditioners.

2: F-SAVE'S save routines were written with careful attention to EXACT timing between bits, and between bytes. This means that there is no fluctuation in the signal's average (DC) level, which can cause unreliable operation with some recorders.

3: A high degree of noise immunity was designed into the system, making it virtually immune to false names, falsely trying to load tiles saved by other routines, etc.
Files are saved to tape as follows:

1: 2-second silence.
2: Header 1 (1024 "0" bits).
3: ID bytes 1: a 01 (start bit) followed by FF=255d (all bits set) followed by AA=176 (every other bit set) are sent to tape.
4: Length-of-name bytes (2)
5: Text of name
6: Header 2 (1024 "0" bits)
7: ID bytes 2 (01, FF, AA)
8: Length-of-file bytes
9: Text of file.

When loading, the routine must find at least 64 consecutive 00 bytes, terminated by a start bit and followed by an FF and an AA. As a result, it is very unlikely that it will falsely try to load files by starting somewhere in the middle.

It is possible to modify the speed at which these routines operate. As supplied, it is approximately 8.5 times as fast as the standard tape routines. Speed is changed by POKEs to five different locations:

No. ADDRESS DESCRIPTION
1: 19713 Bit 0 "on" time
2: 19729 Bit 0 "off" time
3: 19733 Bit 1 "on" time
4: 19740 Bit 1 "off" time
5: 19919 LOAD bit comparison value

Here are a few different speeds that you can try out. "FACTOR" is the relative speed improvement over the standard tape routines.

FACTOR	19713 #1	19729 #2	19733 #3	19740 #4	19919 #5
4-5	30	22	91	82	225
8.5	26	18	41	32	231
13.5	18	10	32	23	240

Here are the "rules." The contents of #2 must always be 8 less than #1, and #4 must be 9 less than #3. The lower the numbers, the faster it goes. The ratio of (3+4) to (1+2) is the approximate ratio of “1” pulse-width to “0” pulse-width; higher ratios give better discrimination between bits. To determine the value for #5, average the values of #2 and #4, and subtract from 256.

The TS1500 performs best at higher rates, because of the high-pass filter in its tape-port hardware. For this machine, I wouldn’t recommend trying to go much slower than “stock” (second entry in table above). ZX81’s and TS1000’s can go with the slower rates, however. So if your tape recorder just isn’t up to snuff for the higher rates, try slowing it down.

] have successfully used these routines at a 15x speed increase. However, the last entry above is about as fast as you'd dare to go, unless you have a REALLY excellent tape recorder.

APPENDIX II
EASY DEC-HEX CONVERSION

If you can not seem to get the hang of converting decimal to hex, I offer this simple, but effective method of converting and entering decimal data into SRAM HI*RES.

Many published programs have lists of data which contain information for defining USER DEFINED GRAPHICS. For example, a recent issue of ZX-COMPUTING has a program which defines a new character set, The data for defining the A and B characters was given as:

```
500 DATA 30,34,66,126,66,66,66,0
510 DATA 92,98,66,92,66,66,124,0
```

The problem therefore is how to easily convert the decimal values to the hexi-decimal format required by SRAM HI*RES, and then enter the hex data into a program line. I have found the following method the easiest to use. Let's assume that you want to define the A and B characters using the above data, and that you will do this in line number 100. Therefore, you should enter into your program a line which looks like this:

```
100 If USR HR THEN LPRINT U;"A",""
```

Leave the quotes empty, we'll fill them in later.
Now enter the short subroutine given below:

```
9000 INPUT D
9010 LET HI=INT(D/16)
9020 LET LO=D-HI*16
9030 LET H$=CHR$(HI+28)+CHR$(LO+28)+","
9040 PRINT H$;
9050 GOTO 9000
```

Before running the subroutine, put the line cursor onto the line you want to enter the data into, in our example this would be line 100. This is done by entering the command:

```
LIST 100
```

Now GOTO 9000. Enter the decimal data, which should be printed to the screen in hex format.

When you've entered all the data you want in the specified line, enter STOP (shift A). The program will stop with report 9/9000, but the hex data will still be on the screen. Now comes the nice part. Press EDIT (shift 1), which will pull line 100 down to the edit line. Now all you have to do is fill in the quotes with the data displayed on the screen (using the right-cursor shift 8), and then put the line back into the program.

APPENDIX III

UDG PATTERN UTILITY

The following program is offered to help you design your own WDG's easily. First, LOAD the CORE program, then enter the program listing as shown, it is pretty well self explanatory. Sample output is shown. These patterns can form the basis of an entire library of neat fill patterns, space monsters and other apparitions (not to mention custom character sets).

APPENDIX 4

REMOVING 64-COLUMN PRINT

The 64-column character patterns (512 bytes) and the 64-column PRINT driver routine are located in the 2 REM statement.

If you will not be needing the 64-column PRINT option, you can delete this line and reduce the length of the core by 616 bytes.

Simply delete line 2, then LIST 10 and POKE 14419,0. Then, to prevent the possibility of accidentally calling the deleted routine (and crashing), enter the following POKEs:

```
POKE 18080,208
POKE 18081,75
```

This will force an error I report, if you try to PRINT something in 64-column mode (mode 4).

APPENDIX 5

DELETING THE F-SAVE ROUTINES

If you won't be using the fast-load package, you can save 430 bytes by removing it.

These routines are housed in line 1 REM. Simply delete line 1 to remove the line, then enter the following POKEs:

```
POKE 19433,208
POKE 19434,75
POKE 19463,208
POKE 19464,75
```

These will cause an error I report if you attempt to use the (deleted) LOAD or SAVE commands.

If you also deleted the 64-column PRINT routine, that will be all you need to do. However, if you wish to keep the é4-column PRINT routine but delete the F-SAVE package, you will have to add a few more POKEs AFTER deleting line 1 and before attempting to use the 64-column PRINT routine:

```
POKE 18080,235
POKE 18081,78
POKE 20205,235
POKE 20206,46
```

That will do it. The PRINT command will now work as before, but the F-SAVE package will be gone.

WARNING: Don’t try to use BOOT to fast-load a program from which the F-SAVE core has been deleted!

WARNING: Tf you change the speed of the F-SAVE routines, be sure to change your version of BOOT also. Subtract the number shown in the REM line within the BOOT program from the addresses shown in Appendix I, and POKE the new address with the appropriate value. (This offset is needed because the F-SAVE routines in BOOT are in the 0 REM, and are transferred from there to their actual run-time locations, after BOOT has loaded.)