SHARP MZ 700 Series

USER NOTES No 1

SHARPSOFT MZ-700 USER NOTES

FIRST EDITION

I N D E X

Sections

SHARPSOFT MZ-700 USER NOTES

FIRST EDITION

Thank you for subscribing to the SHARPSOFT MZ 700 USER NOTES. In our opinion the MZ 700 computer is one of the finest personal colour computers currently available in its price range. This publication will be produced three times a year. Its content will include useful programming tips, articles on all aspects of the MZ 700 hardware and software, and also your letters, comments and programs. If you have a programming idea or a short program which you would wish to pass on to other MZ 700 users then do send it to us at SHARPSOFT.

In each Edition we will publish at least one program and often more. Included in this our first edition is a Z80 dissassembler program written in BASIC. The listing is given, with instructions and notes, allowing you to expand or change the program.

The major articles in this Edition are firstly an extensive presentation of undocumented features contained in the MZ 700 SHARP BASIC and secondly a background article on the SHARPSOFT FORTH for the MZ 700. SHARPSOFT FORTH will be released towards the end of November. We are offering our FORTH package to MZ 700 User Notes subscribers at reduced cost.

In the future we will include information on the current and very exciting, software developments for the MZ 700 - including PASCAL and assembly language programming.

The Second Edition of the User Notes will be published towards the end of February 1984.

MIKE BRINSON

EDITOR

SHARP MZ-700 BASIC V1.0A

The MZ-700 BASIC is the latest interpreter to be released by SHARP for their MZ series of computers. It is upward compatible with the earlier MZ-80K and MZ-80A interpreters.

Storage of BASIC Programs

SHARP BASIC stores lines of source code in compacted form using the "token" concept and ASCII characters. This technique has become a standard procedure in most of the popular versions of the BASIC language.

In SHARP BASIC all reserved words, for example, FOR, GOTO, END, = are stored as one or two byte tokens. These tokens have their highest bit set to logic 1, which implies they are greater than 7FH or 127. A few examples are:-

97	REM	8D	FOR	F4	=
E0	TO	8F	PRINT	FF82	SIN
F7	+	F8	-	96	DIM

The token table is stored at memory locations

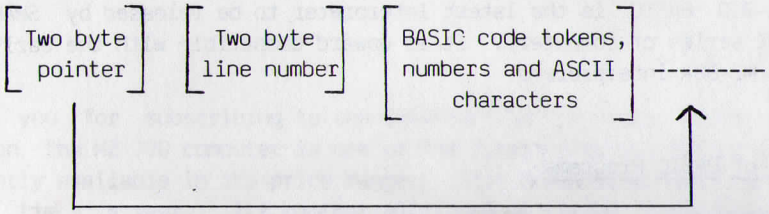
$2AF5	to	$2CA8	and continues between
$30A8	to	$3146	

Numbers are held in floating point format (6 bytes), for example:

1_{10}	=	15 81 00 00 00 00
2_{10}	=	15 82 00 00 00 00
3_{10}	=	15 82 40 00 00 00

Identifier names and text strings are stored as ASCII characters and graphics characters.

The line number at the start of each BASIC line is stored as a two byte binary number. Each stored BASIC line also holds a two byte pointer containing the relative address in memory of the next line of BASIC code. By using this linked pointer the BASIC interpreter can search rapidly for a given line number. The format for each line is:



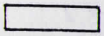
A typical storage pattern in memory is shown below:

```
10   REM   FIRST LINE
11   FOR I=1 TO 10
12   PRINT
13   Y = SIN(I) : REM LINE 13
14   Y = COS(I)
15   Y = TAN(I) : REM LINE 15
16   Y = ATAN(I)
17   NEXT I
```

Hex address									
6BC8	00	00	00	00	00	00	00	11	
6BD0	00	0A	00	97	20	46	49	52	Line 10
6BD8	53	54	20	4C	49	4E	45	00	
6BE0	1A	00	0B	00	8D	20	49	20	Line 11
6BE8	F4	20	15	81	00	00	00	00	
6BF0	20	E0	20	15	84	20	00	00	
6BF8	00	00	06	00	0C	00	8F	00	Line 12
6C00	18	00	0D	00	59	F4	FF	82	Line 13
6C08	28	49	29	20	3A	20	97	20	
6C10	4C	49	4E	45	20	31	33	00	
6C18	0C	00	0E	00	59	F4	FF	83	Line 14
6C20	28	49	29	40	18	00	0F	00	Line 15
6C28	59	F4	FF	84	28	49	29	20	
6C30	3A	20	97	20	4C	49	4E	45	
6C38	20	31	35	00	0C	00	10	00	Line 16
6C40	59	F4	FF	8A	28	49	29	00	
6C48	08	00	11	00	8E	20	49	00	Line 17
6C50	14	00	12	00	59	---	---	---	

Tokens 

97	REM	8D	FOR	F4	=
E0	TO	8F	PRINT	FF82	SIN
FF83	COS	FF84	TAN	FF8A	ATAN
8E	NEXT				

Line numbers  , pointers _ _ _

Floating point numbers _____

Programs are stored in RAM starting at address:

$6BCE

The token table includes a number of BASIC keywords which are NOT describeo in the MZ-700 owners manual - these are:

TRON	TROFF		
ERASE	KILL	OR	AND
BOOT	EOF	HEX$	CLS
JOY			

TRON and TROFF

Entering TRON from the keyboard switches trace on. Trace outputs each BASIC line number as it is interpreted during program execution. TROFF turns the trace off. These commands are useful when debugging programs.

JOY

BASIC statement for controlling the SHARP joysticks. (Complete details are given in the joystick handbook).

JOY(0)	Horizontal movement	0-255	Joystick 1
JOY(1)	Vertical movement	0-255	Joystick 1
JOY(2)	Horizontal movement	0-255	Joystick 2
JOY(3)	Vertical movement	0-255	Joystick 2
JOY(4)	Left hand button	off=0, on=-1	Joystick 1
JOY(5)	Right hand button	off=0, on=-1	Joystick 1
JOY(6)	Left hand button	off=0, on=-1	Joystick 2
JOY(7)	Right hand button	off=0, on=-1	Joystick 2

HEX$

Converts decimal number (or numeric variables) into hex. This function is used in the PRINT statement - for example:

```
10  FOR V=1 TO 255
20  PRINT "DECIMAL = "; V,"HEX = "; HEX$(V)
30  NEXT V
```

CLS

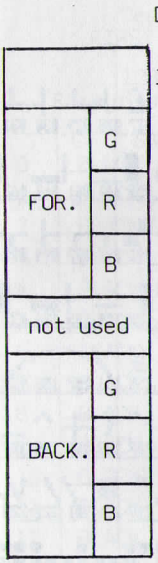
Clears the screen.

BOOT

Same function as pushing the reset key.

The other keywords either return a syntax error or have an unknown effect!

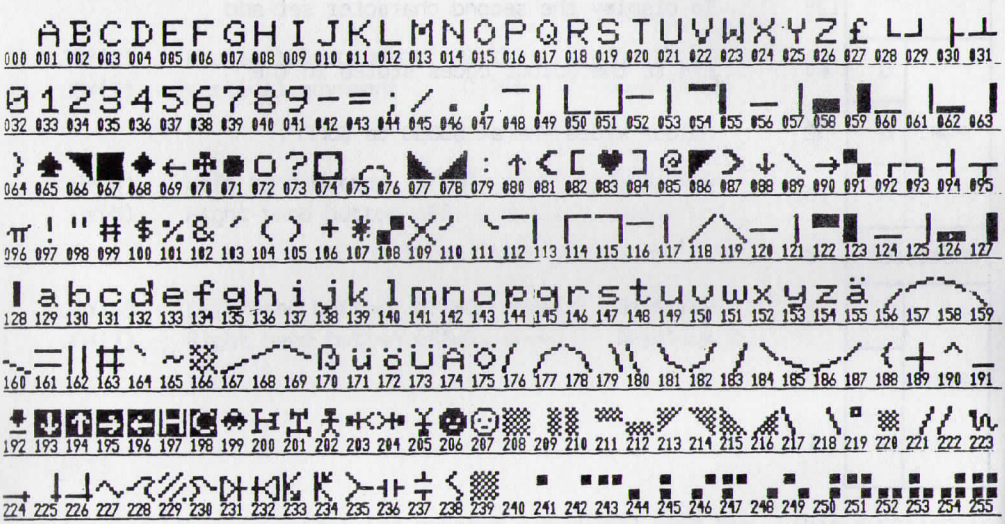
The Second Characters Set



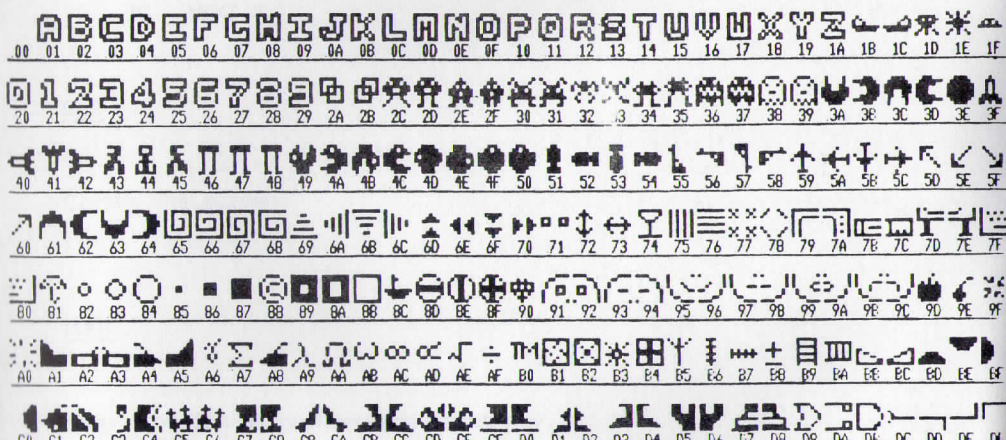
To display the second character set **add** 128 to the colour codes stored in the colour Video RAM at $D800 to $DFFF.

ALL 512 CHARACTER SET

1st CHARACTER SET



2nd. CHARACTER SET



COLOUR CODES FOR COL. VRAM

FIRST CHARACTER SET			SECOND CHARACTER SET		
	Foreground	Background		Foreground	Background
0	BLACK	BLACK	128	BLACK	BLACK
1	BLACK	BLUE	129	BLACK	BLUE
2	BLACK	RED	130	BLACK	RED
3	BLACK	PURPLE	131	BLACK	PURPLE
4	BLACK	GREEN	132	BLACK	GREEN
5	BLACK	L. BLUE	133	BLACK	L. BLUE
6	BLACK	YELLOW	134	BLACK	YELLOW
7	BLACK	WHITE	135	BLACK	WHITE
8	BLACK	BLACK	136	BLACK	BLACK
9	BLACK	BLUE	137	BLACK	BLUE
10	BLACK	RED	138	BLACK	RED
11	BLACK	PURPLE	139	BLACK	PURPLE
12	BLACK	GREEN	140	BLACK	GREEN
13	BLACK	L. BLUE	141	BLACK	L. BLUE
14	BLACK	YELLOW	142	BLACK	YELLOW
15	BLACK	WHITE	143	BLACK	WHITE
16	BLUE	BLACK	144	BLUE	BLACK
17	BLUE	BLUE	145	BLUE	BLUE
18	BLUE	RED	146	BLUE	RED
19	BLUE	PURPLE	147	BLUE	PURPLE
20	BLUE	GREEN	148	BLUE	GREEN
21	BLUE	L. BLUE	149	BLUE	L. BLUE
22	BLUE	YELLOW	150	BLUE	YELLOW
23	BLUE	WHITE	151	BLUE	WHITE
24	BLUE	BLACK	152	BLQE	BLACK
25	BLUE	BLUE	153	BLUE	BLUE
26	BLUE	RED	154	BLUE	RED
27	BLUE	PURPLE	155	BLUE	PURPLE
28	BLUE	GREEN	156	BLUE	GREEN
29	BLUE	L. BLUE	157	BLUE	L. BLUE
30	BLUE	YELLOW	158	BLUE	YELLOW
31	BLUE	WHITE	159	BLUE	WHITE
32	RED	BLACK	160	RED	BLACK
33	RED	BLUE	161	RED	BLUE

FIRST CHARACTER SET			SECOND CHARACTER SET		
	Foreground	Background		Foreground	Background
34	RED	RED	162	RED	RED
35	RED	PURPLE	163	RED	PURPLE
36	RED	GREEN	164	RED	GREEN
37	RED	L. BLUE	165	RED	L. BLUE
38	RED	YELLOW	166	RED	YELLOW
39	RED	WHITE	167	RED	WHITE
40	RED	BLACK	168	RED	BLACK
41	RED	BLUE	169	RED	BLUE
42	RED	RED	170	RED	RED
43	RED	PURPLE	171	RED	PURPLE
44	RED	GREEN	172	RED	GREEN
45	RED	L. BLUE	173	RED	L. BLUE
46	RED	YELLOW	174	RED	YELLOW
47	RED	WHITE	175	RED	WHITE
48	PURPLE	BLACK	176	PURPLE	BLACK
49	PURPLE	BLUE	177	PURPLE	BLUE
50	PURPLE	RED	178	PURPLE	RED
51	PURPLE	PURPLE	179	PURPLE	PURPLE
52	PURPLE	GREEN	180	PURPLE	GREEN
53	PURPLE	L. BLUE	181	PURPLE	L. BLUE
54	PURPLE	YELLOW	182	PURPLE	YELLOW
55	PURPLE	WHITE	183	PURPLE	WHITE
56	PURPLE	BLACK	184	PURPLE	BLACK
57	PURPLE	BLUE	185	PURPLE	BLUE
58	PURPLE	RED	186	PURPLE	RED
59	PURPLE	PURPLE	187	PURPLE	PURPLE
60	PURPLE	GREEN	188	PURPLE	GREEN
61	PURPLE	L. BLUE	189	PURPLE	L. BLUE
62	PURPLE	YELLOW	190	PURPLE	YELLOW
63	PURPLE	WHITE	191	PURPLE	WHITE
64	GREEN	BLACK	192	GREEN	BLACK
65	GREEN	BLUE	193	GREEN	BLUE
66	GREEN	RED	194	GREEN	RED
67	GREEN	PURPLE	195	GREEN	PURPLE
68	GREEN	GREEN	196	GREEN	GREEN
69	GREEN	L. BLUE	197	GREEN	L. BLUE
70	GREEN	YELLOW	198	GREEN	YELLOW

FIRST CHARACTER SET			SECOND CHARACTER SET		
	Foreground	Background		Foreground	Background
71	GREEN	WHITE	199	GREEN	WHITE
72	GREEN	BLACK	200	GREEN	BLACK
73	GREEN	BLUE	201	GREEN	BLUE
74	GREEN	RED	201	GRAEN	RED
75	GREEN	PURPLE	203	GREEN	PURPLE
76	GREEN	GREEN	204	GREEN	GREEN
77	GREEN	L. BLUE	205	GREEN	L. BLUE
78	GREEN	YELLOW	206	GREEN	YELLOW
79	GREEN	WHITE	207	GREEN	WHITE
80	L. BLUE	BLACK	208	L. BLUE	BLACK
81	L. BLUE	BLUE	209	L. BLUE	BLUE
82	L. BLUE	RED	210	L. BLUE	RED
83	L. BLUE	PURPLE	211	L. BLUE	PURPLE
84	L. BLUE	GREEN	212	L. BLUE	GREEN
85	L. BLUE	L. BLUE	213	L. BLUE	L. BLUE
86	L. BLUE	YELLOW	214	L. BLUE	YELLOW
87	L. BLUE	WHITE	215	L. BLUE	WHITE
88	L. BLUE	BLACK	216	L. BLUE	BLACK
89	L. BLUE	BLUE	217	L. BLUE	BLUE
90	L. BLUE	RED	218	L. BLUE	RED
91	L. BLUE	PURPLE	219	L. BLUE	PURPLE
92	L. BLUE	GREEN	220	L. BLUE	GREEN
93	L. BLUE	L. BLUE	221	L. BLUE	L. BLUE
94	L. BLUE	YELLOW	222	L. BLUE	YELLOW
95	L. BLUE	WHITE	223	L. BLUE	WHITE
96	YELLOW	BLACK	224	YELLOW	BLACK
97	YELLOW	BLUE	225	YELLOW	BLUE
98	YELLOW	RED	226	YELLOW	RED
99	YELLOW	PURPLE	227	YELLOW	PURPLE
100	YELLOW	GREEN	228	YELLOW	GREEN
101	YELLOW	L. BLUE	229	YELLOW	L. BLUE
102	TELLOW	YELLOW	230	TELLOW	YELLOW
103	YELLOW	WHITE	231	YELLOW	WHITE
104	YELLOW	BLACK	232	YELLOW	BLACK
105	YELLOW	BLUE	233	YELLOW	BLUE
106	YELLOW	RED	234	YELLOW	RED
107	YELLOW	PURPLE	235	YELLOW	PURPLE

FIRST CHARACTER SET			SECOND CHARACTER SET		
	Foreground	Background		Foreground	Background
108	YELLOW	GREEN	236	YELLOW	GREEN
109	YELLOW	L. BLUE	237	YELLOW	L. BLUE
110	YELLOW	YELLOW	238	YELLOW	YELLOW
111	YELLOW	WHITE	239	YELLOW	WHITE
112	WHITE	BLACK	240	WHITE	BLACK
113	WHITE	BLUE	241	WHITA	BLUE
114	WHITE	RED	242	WHITE	RED
115	WHITE	PURPLE	243	WHITE	PURPLE
116	WHITE	GREEN	244	WHITE	GREEN
117	WHITE	L. BLUE	245	WHITE	L. BLUE
118	WHITE	YELLOW	246	WHITE	YELLOW
119	WHITE	WHITE	247	WHITE	WHITE
120	WHITE	BLACK	248	WHITE	BLACK
121	WHITE	BLUE	249	WHITE	BLUE
122	WHITE	RED	250	WHITE	RED
123	WHITE	PURPLE	251	WHITE	PURPLE
124	WHITE	GREEN	252	WHITE	GREEN
125	WHITE	L. BLUE	253	WHITE	L. BLUE
126	WHITE	YELLOW	254	WHITE	YELLOW
127	WHITE	WHITE	255	WHITE	WHITE

How to obtain the Second Character Set on the MZ-711

The screen on the MZ-700 is memory mapped, this means that every position of the screen has it's own position in memory. This section of memory runs from 53248 ($D000) for the top left of the screen to 54247 ($D3E7) for the bottom right. (See page 128 of the owner's manual).

So for example, position 2,1 has a memory position of 53290 (53238 + No. of rows x 40 + No. of columns), and position 10,20 has a memory position of 54058 (assuming the top left hand corner of the screen has co-ordinates of 0,0).

This allows you to display a character on the screen by POKEing the relevant part of the memory with the display code of a character (See page 155). For example, POKE 53248,1 will display a letter A in the top left hand corner of the screen. The sample program below will fill the screen with space ship characters.

```
10   FOR X=53248 TO 54247  ;Screen addresses.
20   POKE X,$C7            ;Poke space ship character onto the screen.
30   NEXT X                ;Next address.
```

The colour information for the screen is also memory mapped. The position in memory of this being from 55296 ($D800) for the top left of the screen, onwards. To alter the colour of a character on the screen you can simply POKE the relevant position with colour data.

The simplest way of calculating the colour data is to use hexadecimal values. This allows you to enter the codes directly as a two digit number as follows:-

$(C1) (C2) where C1 is the character colour and C2 is the background colour.

For example, $70 would be white (7) on black (0), and $62 would be yellow (6) on red (2). So POKE 53248,1 : POKE 55296, $62 would produce a yellow letter A on a red background, in the top left hand side of the screen.

Below is a simple program which fills the screen with cursor symbols ($D0) of all the possible colour combinations.

```
 10 PRINT "[C]"                ;Clear screen.
 20 C1=0:C2=0                  ;Initialise colours.
 30 FOR X=53248 TO 54247       ;Screen addresses.
 40 POKE X,$D0                 ;Poke cursor character onto the screen.
 50 CL=C1*16+C2                ;Calculate colour code.
 60 POKE X+2048,CL             ;Poke colour data onto the screen.
 70 C1=C1+1:IF C1=8 THEN 100   ;Increment foreground colour.
 80 NEXT X                     ;Next screen address.
 90 END
100 C1=0                       ;Reset foreground colour.
110 C2=C2+1:IF C2=8 THEN C2=0  ;Increment background colour.
120 GOTO 80
```

To access the second character set is now quite simple. You simply add 128 ($80) onto the colour data. For example, POKE 53248,1:POKE 55296,$62 + 128 would produce a reverse A character from the second character set in yellow on red. The samdple program below will produce all the characters of the second set.

```
10 COLOR,,7,1:PRINT " C "
20 N=1                          ;Initialise character number.
30 FOR Y=0 TO 25 STEP 2         ;Row of screen.
40 FOR X=1 TO 40 STEP 2         ;Colume of screen.
50 P=53248 + X + (Y*40)         ;Calculate memory location of screen
                                 position.
60 P1=P+2048                    ;Calculate memory location of colour
                                 info.
70 POKE P,N                     ;Poke the character onto the screen.
80 POKE P1,$71 + 128            ;Poke the colour data (white on blue)
                                 + 128.
90 N=N+1: IF N=256 THEN 110     ;Increment character number and test for
                                 the end.
100 NEXT X,Y                    ;Next position.
110 GOTO 110
```

The sample program below will produce a changing face.

```
10 COLOR,,7,0,:PRINT " C "
20 P1 = 53708                   ;Position of lower part of face.
30 POKE 55716,$70+128:POKE 55717,
   $70+128                      ;Initialise colour part of screen used.
40 POKE 55756,$70+128:POKE 55757
   $70+128                      ;Initialise colour part of screen used.
50 POKE 53668,$91:POKE 53669, $92 ;Poke top of face.
60 FOR CH=$95 TO $9C STEP 2     ;Character codes for lower part of face.
70 POKE P1,CH:POKE P1+1, CH+1   ;Poke the two characters onto screen.
80 FOR TM=1 TO 300:NEXT TM      ;Time delay
90 NEXT CH                      ;Next character
100 GOTO 60
```

A fast routine for displaying graphics characters and colour changes on the V.D.U.

BASIC interpreter memory location $5D is used to store a code (0-255) which determines the colour of the VDU screen.

To fill the screen with character X and the colour held in $5D:

```
POKE $0731,X : USR($0724)
```

To return to normal:

```
POKE $0731,0 : USR($0724)
```

Routine USR($0724) clears the screen and sets the cursor back to the top left hand corner of the VDU screen.

The following routine will quickly change the colour of the VDU screen without changing the character display:

```
1. POKE $730,$18 : POKE $731,2 : REM DISABLES THE CLEAR SCREEN
2. FOR  Z = 1 TO 255
3. POKE $5D,Z                  : REM SETS COLOUR ACCORDING TO
                                   COLOUR CODE Z
4. USR($724)                   : REM PAINTS SCREEN WITH COLOUR
                                   SET IN LINE NO. 3
5. NEXT Z
6. POKE $730,$36 : POKE $731,0 : ENABLES CLEAR SCREEN.
```

Accessing the Second Character Set in BASIC PRINT statements

An easy way to use the second charcter set in the PRINT (or PRINT USING) statement is to POKE location $5D with appropriate colour code plus 128, for example:

```
100 POKE $5D,160          : REM SECOND CHARACTER SET
                            - RED FOR./BLACK BACK.

200 PRINT "SHARPSOFT"     : REM OUTPUT USING SECOND
                            CHARACTER SET/ OUTLINED
                            LETTERS.
```

BASIC Cursor Control

Location $54 contains the horizontal position of cursor (0-39)

Location $55 contains the vertical position of cursor (0-24)

Typical use:-

```
10 POKE $54,10 : POKE $55,10
11 REM sets cursor at X = 10, Y = 10
```

Also note:

```
X = PEEK ($54) : Y = PEEK ($55)
```

will tell you the current cursor position.

Sound control

```
POKE $0A3A, V : REM COURSE CONTROL FOR SOUND
POKE $0A39, Z : REM FINE CONTROL FOR SOUND
```

Both V and Z must be in the range 0-255

User functions:

```
USR(62)   :  REM SOUND BELL
USR(68)   :  REM START SOUND
USR(71)   :  REM STOP SOUND
```

```
POKE $124,1  causes a bleep on entry
POKE $124,4  restores to normal
```

Key control

1. During program execution holding the BREAK key down stops processing. Release the BREAK key and processing continues - useful when using the trace function.

2. Repeat key location - S BASIC Monitor.

```
GET A$ : A = PEEK(95)
```

A then contains the ASCII code of the key being pressed.

3. BREAK key protection.

```
POKE $1935,0 : POKE $1934,0 : POKE $1933,0
```
Inhibits break key during program execution.

```
POKE $1933,$CA : POKE $1934,$71 : POKE $1935,$20
```
Restores to normal

Please NOTE - you must enter the pokes in the order given - otherwise you will have to reload BASIC.

```
POKE $03D1,$EA    :  REM PROTECTS BREAK KEY ON INPUT
POKE $03D1,$9A    :  REM RESTORES TO NORMAL

POKE $0134,$26    :  REM PROTECTS BREAK KEY ON INPUT
                     AND SOUNDS BELL WHEN IT IS PUSHED

POKE $0134,$AD    :  REM RESTORES TO NORMAL
```

4. Restrictive key input routine

```
1  REM RESTRICTIVE KEY INPUT ROUTINE
2  REM
3  REM SHARP MZ700 BASIC
4  REM
5  REM Input is restricted to the characters in string ZZ$
6  REM WW - if 0 tests ZZ$, if 1 tests if ASCII of key is >31 or <32
7  REM QQ - string length of INP$ is restricted to <= QQ
8  REM INP$ - contains string of the user's input
9  REM
10 INP$ = ""
11 II = 0
```

```
12 USR($25B) : KY = PEEK(95)
13 IF II = 0 THEM 15
14 IF KY = 16 THEN INP$ = LEFT$(INP$,II-1):PRINT CHR$(16):GOTO 23
15 IF KY = 13 THEN RETURN
16 IF WW = 1 THEN GOTO 19
17 FOR I = 1 TO LEN(ZZ$) : IF MID$(ZZ$,I,1)=CHR$(KY) THEN 21
18 NEXT I : GOTO 12
19 IF (KY>31) * (KY<92) THEN 21
20 GOTO 12
21 II = II+1 : IF II>QQ THEN II=QQ : GOTO 12
22 PRINT CHR$(KY) : INP$=INP$+CHR$(KY): GOTO 12
23 II=II-1 : IF II<1 THEN 11
24 GOTO12
```

NOTES:-

LINE 12 : USR($25B) blinks the cursor and waits for one key to be pressed. It then stores the ASCII code in location 95($5F).

Exit from this subroutine is in line 15. String ZZ$ must be set in the main calling program To use the routine RENUMBER it to fit in a suitable place in your program.

Please note - this routine produces a string of characters input from the keyboard and stores them in INP$. The length of the string is controlled by line 21. QQ must be set by the calling program. Input characters may be checked for inclusion in ZZ$ by setting W=0 (in the calling program) or for inclusion in the ASCII character set by setting W=1.

More PEEK and POKE notes

There are times when you may wish to inhibit the use of the BREAK key during program execution. This can be achieved by adding the following line to the start of your program:-

```
POKE 6453,0:POKE 6452,0:POKE 6451,0
```

To return the function the following line must be executed:-

```
POKE 6451,202:POKE 6452,113:POKE 6453,32.
```

The display of the MZ-711 can be scrolled in a number of different ways. The simplest way is to press the SHIFT and UP or DOWN cursor keys together. It is possible to prevent this facility being used as follows:-

POKE 78,1 - will inhibit the scrolling facility of the display.
POKE 78,0 - will return this facility.

If you have used a program which uses the GET statement, you will probably have realised that the repeat key function does not work. To get around this you can PEEk (95) which contains the ASCII value of the last key pressed. For example try holding a letter key down in each of the programs below:-

```
10  A$="":GETA$
20  PRINT A$,ASC(A$)
30  GOTO 10
```

```
10  A$="":GETA$
20  A=PEEK(95)
30  PRINT CHR$(A),A
40  GOTO 10
```

It is possible to alter the speed of the repeat of the keyboard as follows: POKE 648,v where v is a value between 1 and 255. The lower the number you use the faster the keyboard response will be, the initial setting is 16.

You can change the character input mode in a number of different ways, for example to puyt the machine into GRAPH mode you can press CTRL+W (see page 27 of the Owner's Manual), or you can PRINT CHR$(23). This may also be achieved with pokes.

POKE 96,255 - Graph mode.
POKE 96,67 - Lower case mode.
POKE 96,239 - Upper case mode.

BASIC USR routines

USR($0C) Displays a space at the current cursor position.
USR($12) Outputs to VDU contents of the Z80 accumulator (A reg.) Prints at the current cursor position.
USR($3E) Sounds bell.
USR($44) Starts sound.
USR($47) Stops sound.
USR($51) Prints message at current cursor position. Z80 DE registers holds address of message Start on entry. Message must end with 00.

FUNCTION keys

BASIC storage locations for FUNCTION keys

Key	HEX	DEC	- MAX 16 CHARACTERS -
F1	$1323	4899	RUN <CR>
F2	$1333	4915	LIST
F3	$1343	4931	AUTO
F4	$1353	4947	RENUM
F5	$1363	4963	COLOR
SHIFT F1	$1373	4979	CHR$(
" F2	$1383	4995	DEF KEY(
" F3	$1393	5011	CONT
" F4	$13A3	5027	SAVE
" F5	$13B3	5043	LOAD

BASIC messages

"READY" message is stored at $219B

"BREAK" message is stored at $21A1

Error messages are stored from $4B73 to $4D4E.

PRINTER/PLOTTER

To self test the printer/plotter push the paper feed key and the reset button at the same time.

Programming tips

There is no need to use a semicolon when printing a number between two strings in a PRINT statement - for example:

```
 5   A = 100
10   PRINT "SHARPSOFT" A "HELLOS"
20   END

RUN
SHARPSOFT 100 HELLOS
```

Also in a PRINT statement variables can be separated by a space - for example:

```
10    A = 100 : B = 20
20    PRINT A B
30    END

RUN
10 20
```

Please NOTE: PRINT AB (without a space) will print variable AB.

COMPATIBILITY WITH OTHER SHARP MACHINES

The BASIC of the MZ-711 is totally compatible with the MZ-80K. This means that you can load an MZ-80K program (written in BASIC SP-5025) directly into the MZ-711 (loaded with S-BASIC). A message will then appear on the screen saying "CONVERTING TEXT". this means that the MZ-711 is converting your MZ-80K program into a form which it can understand.

You should now be able to run this program. It will not of course contain any colour commands, or any other statements unique to the MZ-711. there are one or two differences which you should make a careful note of.

The sound generation locations which are used in conjunction with USR(68) and USR(71) are at locations 4513 and 4514 ($11A1 and $11A2) on the MZ-80K. On the MZ-711 these are at locations 2617 and 2618 ($0A39 and $0A3A).

The cursor co-ordinates on the MZ-80K can be poked in at locations 4465 and 4466 ($1171 and $1172). On the MZ-711 these are at locations 84 and 85 ($0054 and $0055).

There is no direct compatibility between the MZ-711 and the MZ-80A. Although an MZ-80A program will load, if you list it you will see that it is full of rubbish. You can if you wish load the MZ-80A BASIC (SA-5510) into the MZ-711. This will allow you to load and run MZ-80A programs, although you will not be able to use the MZ-711 special features, i.e. colour etc.

HOW TO COPY THE BASIC TAPE OF THE MZ-711

The BASIC tape of the MZ-711 can be duplicated as follows:-

Immediately after switching the machine one, use the M command of the monitor to enter the following data:-

A000 CD
A001 27
A002 00
A003 CD
A004 2A
A005 00
A006 C3
A007 08
A008 11

This is done by entering MA000 followed by the carriage return. You now enter the first number (i.e. CD) and so on. When you have entered all the data press the SHIFT and BREAK keys to return to the * prompt.

Put your BASIC tape into the tape recorder and rewind it if necessary. Now enter JA000 (CR) and then press the PLAY key. After a short time the following will appear on the screen:-

S-BASIC SAVER
HIT ANY KEY?

Remove the BASIC tape, then put in a blank tape. Hit any key then press the RECORD and PLAY keys. When the recording is complete you can use this tape and store your master tape in a safe place.

****

If you find any useful programming hints or undocumented features of SHARP BASIC write to us at SHARPSOFT and we will publish them in the next Edition of the MZ-700 User Notes. Good hunting!

Z80 DISASSEMBLER in BASIC

A machine code disassembler is an important systems utility which allows binary machine code to be converted to a more readable list of assembly language statements.

If you write BASIC programs which include machine code routines that are accessed via the USR() function then checking that these routines are correct using the BASIC monitor DUMP command can be tedious! This process is made much easier with a Z80 disassembler written in BASIC. The following Z80 disassembler will disassemble any location in RAM. The program can be loaded separately or MERGED with your BASIC programs. Remember when merging if line numbers overlap - then renumber the disassembler before use.

Example output.

```
0080 00          NOP
0081 8D          ADC A,L
0082 07          RLCA
0083 E9          JP (HL)
0084 00          NOP
0085 E9          JP (HL)
0086 00          NOP
0087 04          INC B
0088 08          EX AF,AF'
0089 B1          XOR A,C
008A 07          RLCA
008B D6 07       SUB A,07
```

```
50000 PRINT"▯"
50010 PRINT↗P "▯"
50020 PRINT"M≥700 DEVELOPMENT SYSTEM"
50030 PRINT"≥80 DISASSEMBLER"
50040 DIM A$(255),B$(7),C$(7),D$(7)
50050 DIM E$(3),N$(15),U1$(255)
50060 REM READ DATA
50070 FOR I = 0 TO 15 : READ N$(I) : NEX
T
50080 FOR I = 0 TO 7  : READ B$(I) : NEX
T
50090 FOR I = 0 TO 7  : READ C$(I) : NEX
T
50100 FOR I = 0 TO 7  : READ D$(I) : NEX
T
50110 FOR I = 0 TO 3  : READ E$(I) : NEX
T
50120 FOR I = 0 TO 255
50130 IF (I<192)*(I>63) THEN GOSUB 50960
 : GOTO 50160
50140 IF (I=203)+(I=221)+(I=237)+(I=253)
 THEN 50160
50150 READ A$(I)
50160 NEXT
50170 FOR I = 0 TO 255 : IF (I>123)*(I<1
60) THEN U1$(I)="ERROR":GOTO 50200
50180 IF (I<64)+(I>187) THEN U1$(I)="ERR
OR" : GOTO 50200
50190 READ U1$(I)
50200 NEXT
50210 PRINT : PR = 0 : INPUT "LIST from
";A$
50220 GOSUB 51020 : A = A1
50230 INPUT "     to    ";A$ : GOSUB 5102
0
50240 PRINT : INPUT"OUTPUT TO PRINTER ?
";PR$
50250 IF (PR$="YES")+(PR$="Y") THEN PR =
1
50260 IF PR = 0 THEN 50280
```

```
50270 PRINT/P "■"
50280 REM SINGLE BYTE
50290 IF A > A1 THEN 51640
50300 REM MAIN ROUTINE
50310 X1=INT(A/4096):X3=A-X1*4096:X2=INT
(X3/256):X4=X3-X2*256:X3=INT(X4/16)
50320 IF PR = 0 THEN 50340
50330 PRINT/P N$(X1);N$(X2);N$(X3);N$(X4
-16*X3);" ";
50340 PRINT N$(X1);N$(X2);N$(X3);N$(X4-1
6*X3);" ";
50350 GOSUB 51190
50360 IF D=203 THEN GOSUB 51080 : GOTO 5
0280
50370 IF D=221 THEN GOSUB 51190 : IX$="I
X":ID$="IX+K" : GOTO 50770
50380 IF D=237 THEN GOSUB 51190 : GOTO 5
0730
50390 IF D=253 THEN GOSUB 51190 : IX$="I
Y":ID$="IY+K" : GOTO50770
50400 C$=A$(D)+" "
50410 REM
50420 IF LEFT$(C$,1)="2" THEN 50480
50430 IF LEFT$(C$,1)="3" THEN 50600
50440 REM

50450 IF PR = 0 THEN 50470
50460 PRINT/P TAB(18);C$
50470 PRINTTAB(18);C$ : GOTO 50280
50480 REM
50490 FOR I=4 TO LEN(C$) : D$=MID$(C$,I,
1)
50500 IF D$="K" THEN XX = 0 : GOTO 50550
50510 IF D$="Q" THEN XX = 2 : GOTO 50550
50520 NEXT
50530 IF PR=1 THEN PRINT/P TAB()18);"ERR
OR"
50540 PRINT TAB(18);"ERROR" : GOTO 50280
50550 GOSUB 51190
50560 IF PR=1 THEN PRINT/P TAB(18);MID$(
C$,2,I-2); : X=D+XX
```

```
50570 PRINT TAB(18);MID$(C$,2,I-2); : X=
D+XX : GOSUB 50920
50580 IF PR=1 THENPRINT/P MID$(C$,I+2,LE
N(C$))
50590 PRINT MID$(C$,I+2,LEN(C$)) : GOTO
50280
50600 REM
50610 FOR I = 4 TO LEN(C$)-1 : D$=MID$(C
$,I,2)
50620 IF D$="NN" THEN 50660
50630 NEXT
50640 IF PR = 1 THEN PRINT/P TAB(18);"ER
ROR"
50650 PRINT TAB(18);"ERROR" : GOTO 50280
50660 GOSUB 51190 : D1 = D : GOSUB 51190
50670 IF PR = 1 THEN PRINT/P TAB(18);MID
$(C$,2,I-2); : X=D
50680 PRINT TAB(18);MID$(C$,2,I-2); : X=
D : GOSUB 50920
50690 IF (D=237)*(D1=5) THEN USR(62) : S
TOP
50700 X=D1 : GOSUB 50920
50710 IF PR=1 THEN PRINT/P MID$(C$,I+2,L
EN(C$))
50720 PRINT "⊡";MID$(C$,I+2,LEN(C$)) : G
OTO 50280
50730 REM
50740 IF LEFT$(U1$(D),1)="4" THEN C$=U1$
(D)+" " : GOTO 50600
50750 IF PR=1 THEN PRINT/P TAB(18);U1$(D
)
50760 PRINT TAB(18);U1$(D) : GOTO 50280
50770 IF D <> 54 THEN 50800
50780 IF D <> 54 THEN 50800
50790 GOSUB 51190:M=INT(D/16):C$="LD ("+
IX$+"+"+N$(M)+N$(D-16*M)+"),K ":GOTO 504
80
```

```
50800 IF D <> 203 THEN 50850
50810 GOSUB 51190 : F=D : GOSUB 51190
50820 L =INT(D/64):M=INT(D/8-L*8):M1=INT
(F/16):N1=F-M1*16
50830 IF L =0 THEN C$=D$((D-6)/8)+" ("+I
X$+"+"+N$(M1)+N$(N1)+")" : GOTO 50440
50840 C$=E$(L)+" "+N$(M)+" ("+IX$+"+"+N$
(M1)+N$(N1)+")" : GOTO 50440
50850 IF (D=57)+(D<44)+(D>224) THEN C$=A
$(D)+" " : GOTO 50870
50860 IX$=ID$ : C$="2"+A$(D)+" "

50870 FOR I=4 TO LEN(C$)-1 : D$=MID$(C$,
I,2)
50880 IF D$="HL" THEN C$=LEFT$(C$,I-1)+I
X$+MID$(C$,I+2-I-1) : GOTO 50410
50890 NEXT
50900 IF PR=1 THEN PRINT/P TAB(18);"ERRO
R"
50910 PRINT TAB(18);"ERROR" : GOTO 50280
50920 REM
50930 M = INT(X/16) : N=X-M*16
50940 IF PR=1 THEN PRINT/P N$(M);N$(N);"
 ";
50950 PRINT N$(M);N$(N);" "; : RETURN
50960 REM
50970 L =INT(I/64) : N=I-L*64 : M=INT(N/
8) : N=N-M*8
50980 IF L=2 THEN A$(I)=C$(M)+" A,"+B$(N
) : RETURN
50990 IF I=118 THEN A$(I)="HALT" : RETUR
N
51000 A$(I)="LD "+B$(M)+","+B$(N)
51010 RETURN
51020 REM
51030 A1=0 : FOR I=1 TO LEN(A$) : Z=ASC(
MID$(A$,I,1))
```

```
51040 IF (Z>47)*(Z<58) THEN A1=A1*16+Z-4
8 : GOTO 51070
51050 IF (Z>64)*(Z<71) THEN A1=A1*16+Z-5
5 : GOTO 51070
51060 PRINT "CHARACTER ERROR"
51070 NEXT : RETURN
51080 REM
51090 GOSUB 51190 : REM RETURN D
51100 IF D<64 THEN 51150
51110 L =INT(D/64) : N=D-L*64 : M=INT(N/
8) : N=N-M*8
51120 IF PR=1 THEN PRINT/P TAB(18);E$(L)
;" ";N$(M);",";B$(N)
51130 PRINT TAB(18);E$(L);" ";N$(M);",";
B$(N)
51140 RETURN
51150 M=INT(D/8) : N=D-M*8
51160 IF PR=1 THEN PRINT/P TAB(18);D$(M)
+" "+B$(N)
51170 PRINT TAB(18);D$(M)+" "+B$(N)
51180 RETURN
51190 REM
51200 D=PEEK(A) : X=D : GOSUB 50920
51210 A=A+1 : RETURN
51220 REM DATA
51230 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E
,F
51240 DATA B,C,D,E,H,L,"(HL)",A
51250 DATA ADD,ADC,SUB,SBC,AND,OR,XOR,CP
51260 DATA RLC,RRC,RL,RR,SLA,SRA,ERROR,S
RL
51270 DATA ERROR,BIT,RES,SET
51280 DATA NOP,"3LD BC,NN","LD (BC),A","
INC BC","INC B","DEC B","2LD B,K",RLCA
51290 DATA "EX AF,AF'","ADD HL,BC","LD A
,(BC)","DEC BC","INC C","DEC C"
```

```
51300 DATA "2LD C,K"
51310 DATA RRCA,"2DJNZ Q","3LD DE,NN","L
D (DE),A","INC DE","INC D","DEC D"
51320 DATA "2LD D,K",RLA,"2JR Q","ADD HL
,DE","LD A,(DE)","DEC DE","INC E"

51330 DATA "DEC E","2LD E,K",RRA,"2JR NZ
,Q","3LD HL,NN","3LD (NN),HL","INC HL"
51340 DATA "INC H","DEC H","2LD H,K",DAA
,"2JR Z,Q","ADD HL,HL","3LD HL,(NN)"
51350 DATA "DEC HL","INC L","DEC L","2LD
 L,K",CPL,"2JR NC,Q","3LD SP,NN"
51360 DATA "3LD (NN),A","INC SP","INC (H
L)","DEC (HL)","2LD (HL),K",SCF
51370 DATA "2JR C,Q"
51380 DATA "ADD HL,SP","3LD A,(NN)","DEC
 SP","INC A","DEC A","2LD A,K",CCF
51390 REM
51400 DATA "RET NZ","POP BC","3JP NZ,NN"
,"3JP NN","3CALL NZ,NN","PUSH BC"
51410 DATA "2ADD A,K","RST 0","RET Z",RE
T,"3JP Z,NN","3CALL Z,NN","3CALL NN"
51420 DATA "2ADC A,K","RST 1","RET NC","
POP DE","3JP NC,NN","2OUT (K),A"
51430 DATA "3CALL NC,NN","PUSH DE","2SUB
 A,K","RST 2","RET C",EXX,"3JP C,NN"
51440 DATA "2IN A,(K)","3CALL C,NN","2SB
C A,K","RST 3","RET PO","POP HL"
51450 DATA "3JP PO,NN","EX (SP),HL","3CA
LL PO,NN","PUSH HL","2AND A,K","RST 4"
51460 DATA "RET PE","JP (HL)","3JP PE,NN
","EX DE,HL","3CALL PE,NN","2OR A,K"
51470 DATA "RST 5","RET P","POP AF","3JP
 P,NN",DI,"3CALL P,NN","PUSH AF"
51480 DATA "2XOR A,K","RST 6","RET M","L
D SP,HL","3JP M,NN",EI,"3CALL M,NN"
```

```
51490 DATA "2CP A,K","RST 7"
51500 REM
51510 DATA "IN B,(C)","OUT (C),B","SBC H
L,BC","4LD (NN),BC",NEG,RETN,IM0
51520 DATA "LD I,A","IN C,(C)","OUT (C),
C","ADC HL,BC","4LD BC,(NN)",ERROR
51530 DATA RETI,ERROR,"LD,R,A","INC (D)"
,"OUT (C),D","SBC HL,DE","4LD (NN),DE"
51540 DATA ERROR,ERROR,"IM 1","LD A,I","
INC (E)","OUT (C),E","ADC HL,DE"
51550 DATA "4LD DE,(NN)",ERROR,ERROR,"IM
 2","LD A,R","IN C,(H)","OUT (C),H"
51560 DATA "SBC HL,HL","4LD (NN),HL",ERR
OR,ERROR,ERROR,RRD,"IN C,(L)"
51570 DATA "OUT (C),HL","ADC HL,HL","4LD
 HL,(NN)",ERROR,ERROR,ERROR,RLD,ERROR
51580 DATA ERROR,"SBC HL,SP","4LD (NN),S
P",ERROR,ERROR,ERROR,ERROR,"IN C,(A)"
51590 DATA "OUT (C),A","ADC HL,SP","4LD
SP,(NN)"
51600 REM
51610 DATA LDI,CPI,INI,OUTI,ERROR,ERROR,
ERROR,ERROR,LDD,CPD,IND,OUTD,ERROR
51620 DATA ERROR,ERROR,ERROR,LDIR,CPIR,I
NIR,OTIR,ERROR,ERROR,ERROR,ERROR,LDDR
51630 DATA CPDR,INDR,OTDR
51640 INPUT "Continue ? ";Z$
51650 IF (Z$="YES")+(Z$="yes")+(Z$="Y")+
(Z$="y") THEN 50210
51660 PRINT "@"
51670 PRINT/P "@"
51680 END
```

MZ 700 FORTH

FORTH is an unusual computer language, originally invented over ten years ago by an American, Charles Moore, as an alternative to the then existing high-level computer languages. Today, FORTH has evolved into a powerful high-level language with a rapidly growing group of hobbyist enthusiasts both in this country and America. Although FORTH was developed for system applications, its vocabulary can easily be expanded to include additional program structures and new data types. Well known examples are PASCAL - like CASE statements and array and string data structures.

FORTH is both an interpreter and a compiler, merged within its structure are the best features of an interactive interpreter, an assembler, an editor and a computer operating system. These features combine together to produce program execution times similar to that achieved by a Z80 native code compiler.

The statements which compose a FORTH program are called words. One of the most striking features of FORTH is its central data structure, called the dictionary. This dictionary is a list of words linked together. This dictionary accounts for roughly ninety percent of the language, where the fundamental components consist of approximately 40 words. These words are called primatives and are written in machine code. New words are defined in terms of the primatives. In FORTH, as soon as a new word is defined it may be executed immediately or used to create additional new words. FORTH does not use argument lists. Words communicate via a parameter stack. FORTH does allow programmers to define variables but these variables are themselves operations which, for example, place the address in memory of a variable on the top of the parameter stack. This implies that FORTH programmers can not only extend the language by adding new WORDS to the dictionary, but can also create their own data structures and types. Hence, when writing FORTH programs the process used to develop a new program is straightforward but very different to programming in BASIC.

It is difficult to describe FORTH in detail because its extensible features allos a programmer to extend the language to solve the application which is being programmed. However, every FORTH implementation is characterised by a number of common features; there are (1) the dictionary, (2) two stacks, (3) an inner and an outer interpreter, an assembler and a virtual memory management system.

The dictionary is an extensible list of words. Two push-down stacks, the parameter and return stacks, are maintained by FORTH. These stacks are used to communicate arguments between words in a FORTH program. The outer interpreter is a conventional program for passing text strings from a keyboard, and looking up each decoded word in the FORTH dictionary. If a word is found in the dictionary it is executed by calling the FORTH inner interpreter.

Unlike a BASIC interpreter the FORTH inner interpreter is very small, often 25 or less machine code instructions when run on an 8-bit microprocessor, which results in fast execution.

Most FORTH systems include a resident machine code assembler. This assembler is used to create words which are constructed from in-line machine code or machine code subroutines. FORTH assemblers use the reverse polish notation with the assembly mnemonics following their arguments. Fully structured assembly programming is achieved in FORTH through the use of statements which include IF... ELSE... THEN.

In common with other computer languages, some form of mass storage device is necessary to store FORTH programs and test data. FORTH uses a virtual memory system based on blocks which are fixed-length segments of disk or tape space. These blocks may be used to store programs or test data. A number of buffers are also held in RAM, so that the blocks can be read into the buffers automatically from the mass storage device. If a block is modified in memory it is automatically replaced on disk or tape by the new version. Associated with the FORTH virtual memory scheme is a text editor which provides a means to edit program source text, usually both line and character editing facilities are supported.

SHARPSOFT MZ 700 FORTH

SHARPSOFT MZ 700 FORTH is a very low cost tape based implementation of the FORTH language.

This pacakge is based on the "fig-FORTH" dialect of the language with extensive new features that support the MZ 700 advanced hardware. The package provides:-

1. Virtual memory management with blocks stored on cassette tape. Special utilities for formatting blank tapes and viewing the current system configuration.

2. 8080 assembler vocabulary. Z80 machine code extensions may be programmed using FORTH's zero level assembler features.

3. Text editor - which includes line and character editing features.

4. Colour extensions for output to the V.D.U. Both MZ 700 character sets are supported.

5. Printer/plotter extensions for output to the SHARP hard copy device. Both text and graphics modes are supported.

6. Complete "fig-FORTH" dictionary excluding disk words.

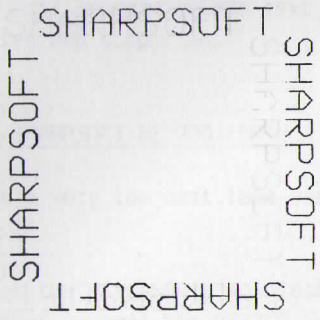
7. The source code for all MZ 700 extensions is included in the package.

This software will be released by SHARPSOFT towards the end of November 1983. We are offering it to MZ 700 User Notes subscribers at the following reduced prices.

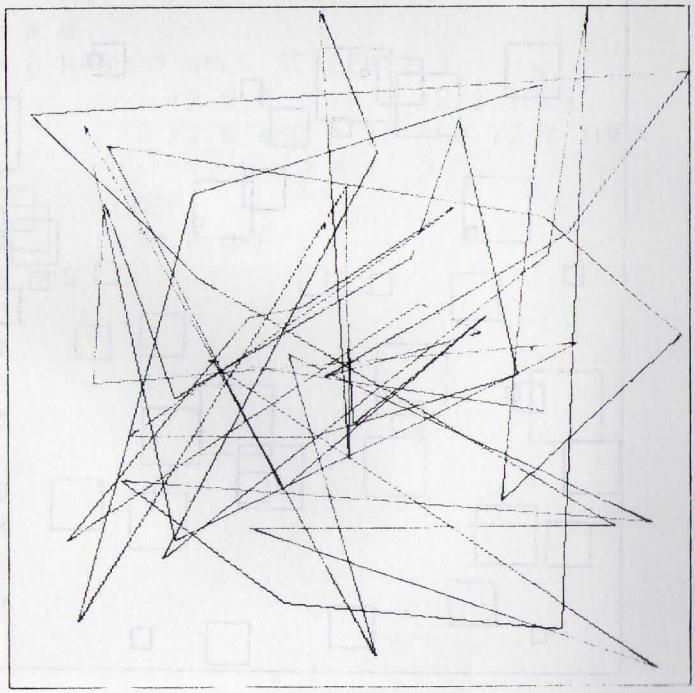
		Pounds Sterling	
(A)	U.K.	8.95	(includes VAT, postage and packing)
(B)	Overseas	13.95	(includes airmail postage and packing)

A twenty page booklet is supplied with the language cassette. This booklet gives helpful tips on how to start programming your MZ 700 in FORTH.

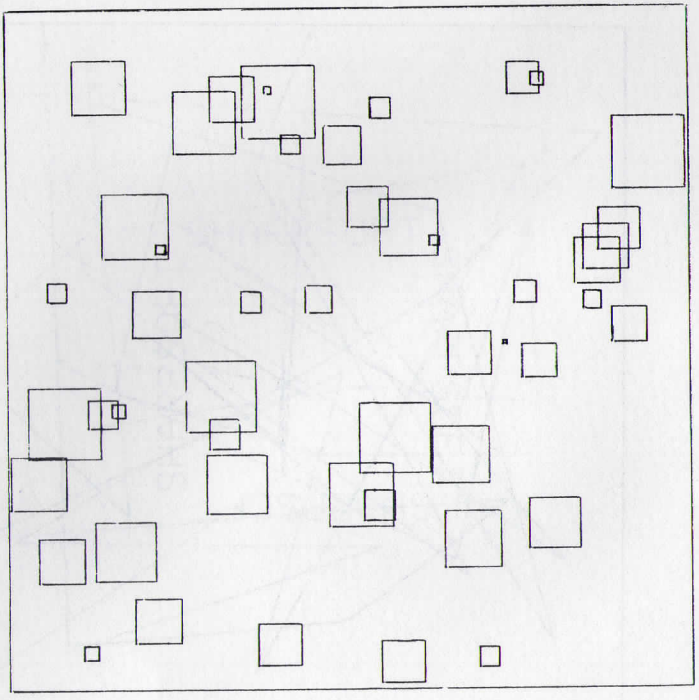
```
SCR # 4
  0 ( PLOTTER/PRINTER EXAMPLE 1 )
  1 FORTH DEFINITIONS DECIMAL
  2 : STEXT .P ." SHARPSOFT" .CR ;
  3 : CRLF  .CR .LF ;
  4 : PEXAMPLE1 P ON
  5                  .GM -100 100 .M
  6                2 .SS
  7                0 .CC
  8                 0 .AR STEXT
  9                 1 .AR STEXT
 10                 2 .AR STEXT
 11                 3 .AR STEXT
 12                 .H .TM P.OFF ;
 13 ;S
 14
 15
```



```
SCR # 5
  0 ( RANDOM LINES )
  1 FORTH DEFINITIONS DECIMAL
  2 0 VARIABLE SEED
  3 : (RAND) SEED @ 259 * 3 + 32767
  4          AND DUP SEED ! ;
  5 : RANDOM (RAND) 32767 */ ;
  6 : GRID 0 0 460 0 460 460 0 460 4
  7          .D ;
  8 : R.LINES P.ON .GM .I GRID
  9           230 230 .M
 10           50 0 DO
 11           459 RANDOM 459 RANDOM 1
 12           .D
 13            LOOP .TM P.OFF ;
 14 ;S
 15
```

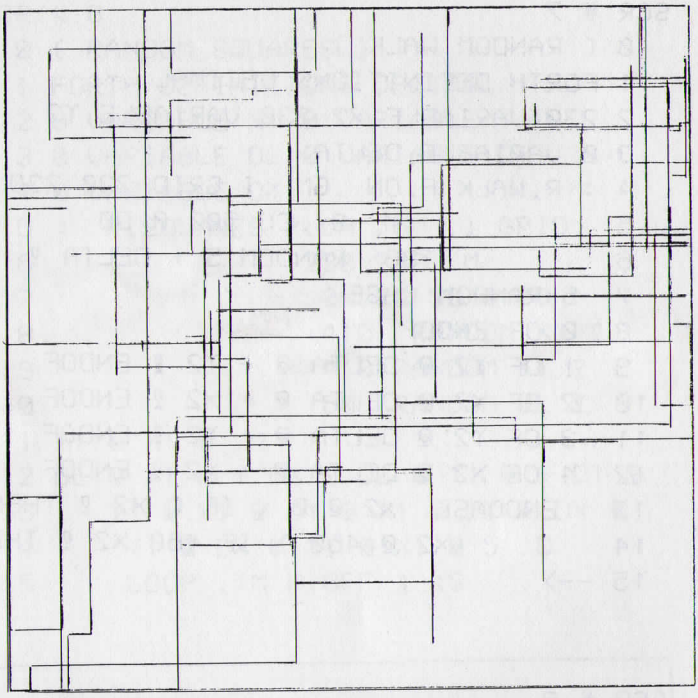


```
SCR # 6
  0 ( RANDOM SQUARES )
  1 FORTH DEFINITIONS DECIMAL
  2 0 VARIABLE X1  0 VARIABLE Y1
  3 0 VARIABLE DL  0 VARIABLE DX
  4 0 VARIABLE DY
  5 :  R.SQUARES P.ON .GM .I GRID
  6                    230 230 .M
  7                     0 .CC 50 0 DO
  8                     410  RANDOM X1 !
  9                     410 RANDOM Y1 !
 10                     50  RANDOM DL !
 11 Y1 @ X1 @ .M
 12 DL @ X1 @ + DX ! DL @ Y1 @ + DY !
 13    Y1 @ X1 @ DY @ X1 @ DY @ DX @
 14     Y1 @ DX @ Y1 @ X1 @ 5 .D
 15     LOOP .TM P.OFF ; ;S
```



```
SCR # 7
  0 ( RANDOM WALK )
  1 FORTH DEFINITIONS DECIMAL
  2 230 VARIABLE X2 230 VARIABLE Y2
  3 0 VARIABLE DELTA
  4 : R.WALK P.ON .GM .I GRID 230 230
  5              .M  0 .CC 500 0 DO
  6            100  RANDOM 5 + DELTA !
  7   5 RANDOM CASE
  8   0 OF ENDOF
  9   1 OF Y2 @ DELTA @ - Y2 ! ENDOF
 10   2 OF X2 @ DELTA @ - X2 ! ENDOF
 11   3 OF Y2 @ DELTA @ + Y2 ! ENDOF
 12   4 OF X2 @ DELTA @ + X2 ! ENDOF
 13    ENDCASE  X2 @ 0 < IF 0 X2 ! THEN
 14           X2 @ 460 > IF 460 X2 ! THEN
 15 -->


SCR # 8
  0 ( RANDOM WALK SCREEN 2 )
  1          Y2 @ 0 < IF 0 Y2 ! THEN
  2          Y2 @ 460 > IF 460 Y2 ! THEN
  3           Y2 @ X2 @ 1 .D
  4       LOOP
  5       .TM P.OFF ;
  6   ;S
  7
  8
  9
 10
 11
 12
 13
 14
 15
```



NOTES TO AUTHORS

HINTS & TIPS

SHARPSOFT are always looking for Tips, Hints, Peeks and Pokes etc. that you find whilst working on your **MZ 700**, so why not drop us a line and we will include them in future issues of the **SHARPSOFT 700 USER NOTES.**

ARTICLES & PROGRAM LISTINGS

Up to £10.00 worth of HARDWARE/SOFTWARE vouchers will be paid for hardware/software articles or program listings for **MZ 700** or its peripherals. Why not send your "Pride & Joy" for publication?

SHARPSOFT LTD.,
CRISALLEN HOUSE,
2nd FLOOR,
86/90, PAUL STREET,
LONDON, EC2A 4NE,
ENGLAND.

STOP PRESS

MZ700 HIGH RESOLUTION SPECIFICATIONS

The system consists of a high quality double sided, through plated printed circuit board. 80.5 mm X 222.5 mm. The board is supplied fully assembled and tested. Installation consists of removal of the Monitor ROM character generator ROM and attribute RAM from the main PCB. These are then plugged into the High Resolution board. The High Resolution board is then plugged into the sockets vacated by these chips. In addition to this four solded connections are made to the main PCB.

The High Resolution includes an additional 8K of RAM. This RAM is page moded with the attribute RAM all decoding is done by a single port. The facilities given are:

1. Full High Resolution graphics, 320 X 200.
2. All 64000available pixels can be displayed simultaneously.
3. The High Resolution can be switched in/out under software control.
4. Text and Graphics can be mixed.
5. A single command re-configures the High Resolution RAM into four programmable generators, ie: 1024 user programmable graphic characters.
6. Colour Attributes - these are the same as the standard MZ700 ie: foreground and background can be specified within a single cell.
7. Screenflash supression is ensured as updating can take place during fly-back time.
8. A selection pattern can be built into the attribute RAM allowing a rapid scan to be made of the four PCGs, thus giving animation capability.
9. A modified basic is supplied giving full set, re-set, line-draw, and line-wipe facilities.
10. When the High Resolution PCG system is not in the display mode, the RAM can be used by the user as work space.

SHARPSOFT

Sharpsoft Ltd., 86-90 Paul Street, London EC2A 4NE
Printed by Oldham Press (T.U.), Chatham, Kent.

£1.50