SHARPSOFT

SHARPSOFT USER NOTES

Issue No 1 January 1981

Thank you for subscribing to the SHARPSOFT-USER NOTES. In our opinion the MZ-80K computer is one of the finest personal computers currently available in its price range. It was unfortunate that the MZ-80K was released by SHARP after its competitors' machines were established in the U.K. market. This fact is obvious if one looks through the popular magazines and counts the number of published MZ-80K programs or articles. Only in the last six months are users beginning to get their programs and ideas into print. We believe that it does not matter how good a manufacturer's hardware is, without software back-up it is difficult for the user to obtain the best from a computer. Frustration on our part at the lack of software for the MZ-80K and general programming information led to the idea that we should produce these notes. We hope that through this medium we can pass onto you useful programming tips and in return through your letters we can publish your ideas to all our subscribers.

Initially we intend to produce three or four newsletters a year. However, if the venture is successful and demand is high enough we will increase the number of issues to six a year. In some respects this depends on you. If you have a programming idea or a short program which you would wish to pass on to other MZ-80K users write to us. If we like it then we will publish it in a future edition of the SHARPSOFT-USER NOTES. As an added incentive all authors who have an item published in these User Notes will receive a software voucher valued at £5.00. This voucher may be used to help purchase software from the SHARPSOFT Catalogue.

In each issue we will publish at least one program and often more. Included in this our first issue is a program for the generation and maintenance of a contents directory on cassette tape. The listing is given, with instructions and notes, allowing you to expand or change the program.

The major article in this issue is an extensive reference document for the SHARP tape and disk versions of the SHARP BASIC language. We chose this for our first issue because the majority of our readers use these versions of BASIC and indeed often find the manuals provided with their machines inadequate.

Also included, for machine code programmers, is a list and short description of the important MZ-80K monitor routines and their entry addresses.

In the future we will be including information on the current, and very exciting, software developments for the MZ-80K, for example CP/M, FORTH/STOIC, PASCAL, ALGOL, FORTRAN and machine code development programs.

SHARP BASIC reference notes

These notes are intended for users of the SHARP SP-5025 Tape and SP-6015 disk BASIC. The material is presented as a reference guide. The complete set of commands, statements, functions and operators are listed, together with examples, detailed explanations and their application and function.

BASIC operates in two modes, the direct mode and the program execution or run mode. After loading BASIC to RAM, the prompter READY appears on the V.D.U. screen to indicate that the computer is in the direct mode and ready to accept keyboard input. With the computer in the direct mode, we can enter a line of text from the keyboard. The entered text is echoed to the V.D.U. screen and stored in a line buffer in RAM. On pressing the yellow CR key one of two possibilities take place. Firstly, if the line of text starts with a line number, the line of text is stored in the computer RAM and becomes part of a BASIC source program. Lines are stored in correct numerical sequence. Secondly, if the line of text did not start with a line number, BASIC executes the text from the line buffer treating the text as if it were a one line program. Provided BASIC is in the direct mode the line of text displayed on the V.D.U. screen may be edited using the yellow special function keys described on page 15 of the SHARP MZ-80K BASIC manual.

Direct Mode Commands (T = Tape, D = Disk)

RUN (T and D)
Enters run mode and executes a stored BASIC program. BASIC starts interpretation and execution of the source code starting at the first line, lowest line number, of the program.
Discards any previous values of variables and constructs a new variable table.

RUN "Line number" (T and D)
Starts execution of the stored program at "line number", for example RUN 200 starts at line 200.
If an unused line number is assigned in the Run command an error occurs.
All real variables in the source program are set to zero and all string variables are set to a null string.

GOTO "Line number" (T and D)
Starts execution of the program from "Line number". Keeps the old Variable table and retains previous values of variables.

```
GOTO 1000
```

LIST (T and D)
Lists the stored source program on the V.D.U. screen. May be stopped with (SHIFT/BREAK)

SHARP SP-5025 Tape and SHARP SP-6015 Disk BASIC are copyright by SHARP.

Examples.

List 70	Lists line 70 only.
List 70 -	Lists lines 70 to the end of the source program.
List 70-100	List lines 70 to 100.
List 100	List all lines from the beginning of the source program to line 100

LIST/P (T and D)

Lists the stored source program on the printer. Similar facilities to the LIST command. An error results if a printer is not connected to the MZ-80K.

(SHIFT/BREAK) (T and D)

Interrupts execution of the source program, LISTing, LOADing, and SAVE-ing commands and returns to direct mode.

CONT (T and D)

Continues any operation that has been interrupted by a (SHIFT/BREAK) or a STOP or an END command, except the LIST command. The CONT command cannot be used in the following cases.

1. Before a program has been executed by the RUN command.
2. When a program has been edited after execution of the program has been stopped.
3. If an error occurs during execution. In this case the computer returns to the direct mode and outputs to the V.D.U. screen the READY prompt.
4. When the MUSIC command is interrupted using (SHIFT/BREAK).

(BREAK) (T and D)

Pressing the yellow BREAK key during LOADing or SAVE-ing source programs or data on cassette tape will stop the read or write operation. NOTE, the CONT command cannot be used to restart the read or write sequence.

LOAD (T only)

Reads a BASIC source program, stored on a cassette tape, and loads it into RAM. Since no file name is specified, the first program found on the tape is read.

LOAD "File name" (T only)

Reads the BASIC source program, stored on a cassette tape, with the file name "File name" and loads it into RAM. File names with not more than 16 characters are valid. NOTE, the commands LOAD and LOAD "File name" clear any previously stored programs in RAM. These commands cannot be used to merge or append programs.

NEW (T and DO	Delete the source program currently stored in RAM. NOTE, this command can be included in a stored program source! for example 100 NEW, use with care.
SAVE (T only)	Write the program currently stored in RAM to cassette tape.
SAVE "File name" (T only)	Similar to the SAVE commands except this command writes a header "File name" at the start of the tape program file.
BYE (T and D)	Exits from the BASIC interpreter and returns control to the SHARP machine code MONITOR program in ROM. NOTE, this command can be included in a BASIC program, for example, 10 BYE
RUN 'File name" (D only)	Loads the program names "File name" from the disk to RAM and executes the program. "File name" must be the name of a BASIC program file with a BTX extension or (an Object code program with extension OBJ). The disk drive number and disk volume number may be entered before the "File name", for example: RUN FD3@7,"File name" NOTE, all real variables are set to zero and string variables to a null string before the execution of the loaded program.
CLR (D only)	This command clears all the real variables in a source program to zero and all string variables to a null string.
DIR n (D only)	Display on the V.D.U. screen the directory for the disk in drive n. If the number n is omitted (i.e. DIR) then the directory for drive 1 is listed. The SHARP Disk BASIC stores the number of the drive for which the last DIR n command was requested. This means that in any subsequent direct mode disk commands the drive number may be omitted if the command applies to the last DIR accessed disk.
DIRn/P (D only)	Similar command to DIR n but lists the directory on the printer. An error results in a printer is not connected to the MZ-80K.
LOAD "File name" (D only)	Similar command to Run "File name" except the program is loaded from disk and NOT run.
LOAD/T (D only)	Similar to tape BASIC LOAD command.

LOAD/T "File name" (D only)
Similar to tape BASIC LOAD "File name" command.

SAVE "File name" (D only)
The BASIC program in RAM is given the name "File name" and written to disk as a BTX extension File. The disk drive number and disk volume number may be entered before the "File name", for example:

```
SAVE FD1@3,"TEXT C"
```

SAVE/T "File name" (D only)
Similar to tape BASIC SAVE "File Name" command

RUN MODE COMMANDS (statements)

= (T and D)
The replacement statement. Note LET is not used in SHARP BASIC. Examples

```
 20   Y=10
100 A1$="TEST"
```

REM (T and D)
This statement allows comments to be included in a source program. Examples

```
110  REM THIS IS A COMMENT
200  M=20+B: REM ADD 20 to B.
```

NOTE; unlike some BASIC interpreters or compilers, SHARP BASIC does not pack repeated characters into a compact form. Every ASCII character in a REM statement takes one byte of memory.

GOTO (T and D)
Example; 200 GOTO 2000
On execution of a GOTO statement a jump to the line number following the GOTO statement occurs. A line number must following the GOTO statement, for example the following is not allowed; 200 GOTO n, where n is a variable. This applies to all control statements.

GOSUB (T and D)
Example; 300 GOSUB 2500
Subroutine call command. In the above example, the program branches to the subroutine which starts at line 2500. A complete program could be:-

```
  10 Z=1
  20 GOSUB 1000
  30 PRINT Z
  40 END
1000 Z=Z+1: REM SUBROUTINE
1010 RETURN
```

Where the statements are executed in the following order 10, 20, 1000, 1010, 30, 40

Note, all variables are "GLOBAL" in BASIC

ON----GOTO (T and D)

Example, 250 ON I GOTO 1000, 2000, 3000.
This statement is called a computed GOTO statement. In the above example the program jumps to statement 1000 if the value of variable I=1, to statement 2000 if I=2 or to statement 3000 if I=3.
NOTE; the program executes the next statement after the ON----GOTO statement if I=0 or I =4. Also if the control variables (I in the example) is a real quantity, for example I=2.7, the value of the control variable is truncated to an integer. In the example I is set equal to 2 if I=2.7 on entry to statement 250.

ON----GOSUB (T and D)

Example, 100 ON I GOSUB 700, 800.
This statement is called a computed GOSUB statement. In the above example the program calls the subroutine starting at line 700 (i.e. executes the statements at line 700) if I=1 or calls the subroutine starting at line 800 if I=2.
NOTE; the program executes the next statement after the ON----GOSUB statement if I=0 or I =3. Also truncation of the control variable is undertaken, see the ON---GOTO statement. There is no limit (except line length) to the number of line numbers after the GOSUB. Upon return, from the subroutine called, the next statement after the ON---GOSUB statement is executed.

IF----THEN (T and D)

Examples,
```
10 IF A  20 THEN 100
20 IF A   10 THEM 200
30 IF B  3 THEN B=B+3
40 IF B  7 THEN PRINT B
```

If the expression after the IF is true, then all the statements after the THEN are executed. If NOT true, then the next line following the IF---THEN statement is executed.

NOTE the following example:-

```
100 IF A=B THEN IF C=D THEN 1000
110 U = P
```

In cases where multiple IF statements are present on the same line the final statement after the last THEN is only executed if all the conditions are true. If one or more conditions are false BASIC executes the next statement following the IF---THEN statement.

IF----GOTO
(T and D)

Example, 100 IF I = B GOTO 10.
If the expression after the IF is true, then the statement indicated by the line number following the GOTO is executed. If the expression after the IF is NOT true, then the statement following the IF----GOTO statement is executed.

IF----GOSUB
(T and D)

Example, 100 IF M = B + 2 GOSUB 700.
If the expression after the IF is true, then the subroutine indicated by the line number following the GOSUB is called. If the expression after the IF is not true, then the statement following the IF----GOSUB is executed.

FOR---=---TO
|
|
NEXT---

BASIC loops.
There are several subtle and very inportant facts which must be remember for trouble free use of loops. Consider the following example:-

```
100 FOR M = 1 TO 4
110 PRINT M
120 NEXT M
150 PRINT "THE FINAL VALUE OF M IS";M
RUN

 1
 2
 3
 4
THE FINAL VALUE OF M IS 5
```

Note, the loop is always executed at least once because the test for exit occurs at the NEXT statement after the loop control variable, M in the above example, has been incremented. On entry to the FOR----NEXT statement, the initial value of the loop variable M is calculated, then the value which determines the exit condition is calculated. The increment size, one in the above example, is also determined. These values do not change during the loop calculations, because the statements in the body of the loop will be repeatedly executed but the FOR statement will not again be interpreted.

Consider the following example:-

```
1000 FOR M = 1 TO 100
1100 M = 20
1200 NEXT
```

RUN - causes the program to loop forever. In the body of a loop, the value of the loop control variable may be redefined - but use with care.

Also, note that the NEXT statement can be used without the name of the control variable. After entering a loop, it is possible to jump out of the loop before the normal exit at the NEXT statement. On exit the loop control variable retains its current value. This is demonstrated by the following example:

```
200 FOR M = 1 TO 4
300 IF M = 2 THEN 6000
400 NEXT
500 PRINT "EXIT AT LOOP NEXT";:END
6000 PRINT "EXIT AT JUMP";M
6100 END

RUN
EXIT A JUMP 2
```

It is also possible to re-enter a loop that has previously been exited from by a jump. However, a program error will occur if a jump to an uninitialised loop is attempted. Executing a NEXT statement without first going through a FOR statement will also cause an error.
Loops may be nested, consider the following example:

```
20 FOR M = 1 TO 3
50 FOR J = 1 TO 10
60 NEXT J
70 NEXT I
```

In this example, the control variable J could have been left of line 60 since a NEXT without a control variable name is assumed to apply to the last FOR---TO statement executed. NOTE, in nested statements the order of the NEXT statements is important. If in the last example lines 60 and 70 were written as

```
60 NEXT I
70 NEXT J
```

an error occurs.

If the nested loops end together, a shorter form of the NEXT statement can be used, for example

```
400 NEXT A, B, C, D, E
```

Again the order of the control variable is important.

STEP
(T and D)

Loop increments, other than 1, in the FOR---TO statement are formed using the STEP statement for example

```
100 FOR X = 19.7 TO 30.2 STEP 0.1
200 FOR M = -100 TO -1000 STEP -10
300 FOR Y = 0 TO 10 STEP 0.1  Y
```

DATA
(T and D)

The DATA statement is used to store blocks of either numeric and/or string data with a BASIC source program. DATA statements are reasonably economical in terms of storage space in RAM and in general it is wasteful to transfer data from a DATA statement to a dimensional array.
Example, 200 DATA 5, b, 17.5, "A", BIG
Only the order in which the date is stored in the program is important, not the number of DATA statements used for example:

```
100 DATA 10, 20, 30, 40, 50
```

and

```
200 DATA 10, 20, 30
210 DATA 40, 50
```

are the same. However, note the latter form takes up more room in memory.
In a DATA statement where a string includes a , or a : the , or : characters must be enclosed by double quotation marks, for example:

```
1000 DATA HEART, "E,F,G," SPADE
```

READ
(T and D)

The contents of a DATA statement must be transferred to other BASIC statements for use, for example:

```
10 DATA 1, 2, 3, 4, 5
20 FOR I = 1 TO 5
30 READ M(I)
40 NEXT
```

or

```
10 DATA 1, 2, 3, 4, 5
20 FOR I = 1 TO 5
30 PRINT READ M
40 NEXT
```

If simultaneous use of the data items stored in a DATA statement is NOT required then storage space can be saved, as in the second form in the above examples, by NOT transferred the data to a dimensional array. As a READ statement "reads" data sequentially from a DATA statement or a set of DATA statements, a pointer is set which points to the next available data item. The DATA statements are used in the numerical order in which they occur in the source program, no matter where the READ statements are located.

RESTORE (T and D)

This BASIC statement restores the READ/DATA pointer to the first entry in the first DATA statement in the program.

RESTORE "Line number" (D only)

The RESTORE "Line number" statement restores the READ/DATA pointer to the first entry in the DATA statement on line "line number" of the source program, for example:

```
500 DATA 1, 2, 3, 4, 5
510 DATA A, B, C, D, C
520 DATA 6, 7, 8, 9, 10
530 DATA E, F, G, H, I
     |   |        |
     |   |        |

1000 RESTORE 510
```

CLR (T and D)

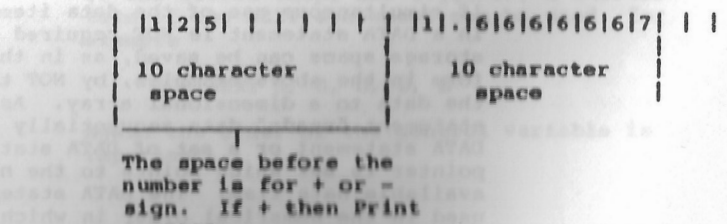
This statement zeros all nubmeric variables and clears all string variables to a null string. It also possibly has the same effect as a RESTORE command on the READ/DATA pointer - needs testing.

PRINT (T and D)

The numeric variable, string variable and expression values given after the BASIC word PRINT are displayed on the V.D.U. Screen at locations following the current cursor position. When writing a BASIC source program the symbol "?" can be used as a short form for the word PRINT. PRINT without any expressions prints a blank line. There are two kinds of separator between the items in the list of items to be printed following the word PRINT, these are comma (",") and semicolon (";").
If the PRINT list is terminated with a semicolon, the linefeed/carriage return at the end of the line will be suppressed.
The command generates tabulated output with the items in the PRINT list separated into 10 space zones, for example

```
100  PRINT A, B  )where A = 125
                          B = 1.6666667
```

yields



If the item in a given column is longer than 10 spaces, in for example text, or overlaps into the next 10 character field the next 10 character field is skipped. If there are more than four items in the list to be printed, then more than one line is used. The semicolon causes the print fields to be adjacent to each other. Thus strings are printed without spaces between them. However, remember positive numbers have a leading space.

Hence:

```
200 PRINT  A ; B
```

yeilds

```
|1|2|5| |1|.|6|6|6|6|6|6|7| | | | | | | | |
```

Comma and semicolon separators may be used in the same list.

PRINT/T
(T only)

Executes identical function to PRINT statement but sends output to cassette tape.

PRINT/P
(T and D)

Executes identical function to PRINT statement but sends output to optional printer. Error results if the printer is not connected.

TAB
(T and D)

The TAB function is used in the PRINT statement to cause data to be printed in exact locations. TAB indicates to the BASIC interpreter which position to begin printing the next value in the print list. The argument of TAB may be an expression, for example:

```
300 TAB (X);A$
```

Shifts the cursor by X character spaces from the left hand side of the V.D.U. Screen, and displays the character string A$.

SPC
(T and D)

This function is used in the PRINT statements to add spaces between items in the print list. The argument of the function is a numerical constant or an expression that has values between 0 and 255. If it is not an integer value, the argument is translated to an integer. Note large argument values will cause printing to continue on the next or later lines.

INPUT
(T and D)

The input statement allows users to enter data from the keyboard during program execution. The statement can be preceded by a comment, for example:

```
20 INPUT "NEXT"; A, B, C
30 INPUT "NEXT ?" D, E, F
40 PRINT A; B; C
50 PRINT D; E; F
60 END

RUN
NEXT 1, 2, 3
NEXT ? 4, 5, 6
1 2 3
4 5 6
```

Note strings can also be entered using the INPUT statement.

Caution: for INPUT A$, B$, C$ - a null response would create three null variables. Only constants can be given in response to an INPUT statement.

INPUT/T
(T only)

Executes identical function TO INPUT statement but reads data from cassette tape. If however, there is no data file opened by a ROPEN statement, an error occurs.

POKE
(T and D)

This operator statement stores an integer N in a location M of memory, for example

```
200 I = 2
210 X = 53248
220 POKE X + 10 or I, I + 1
```

Special POKE commands

1. POKE 53248 + X, Y — Displays a character corresponding to display code Y at picture element X.

2. POKE 4509, 0 — Entry bell sounds when a key pressed.

3. POKE 4509, 1 — Stops bell sound when a key pressed.

An error is reported if the number to be stored is not in the range 0 N 255.

PEEK
(T and D)

This is a function not a command. PEEK returns the value (as a decimal integer between 0 and 255) of the contents of an address W (in decimal), for example:

```
20 W = 53248
30 PRINT PEEK (W)
40 END
```

NOTE, if an address is specified which is at a memory location in the BASIC interpreter or BASIC source code region of RAM then 32_{10} is returned.

The following POKES allow users to "PEEK" at all memory locations:

1. Tape BASIC — POKE 10167,1
2. Disk BASIC — POKE 8048,1

These POKES must take place before the use of a PEEK if the return 32_{10} is to be disabled.

LIMIT
(T and D)

This command specifies the upper limit of RAM which may be used by a BASIC program. It is used to reserve memory at the top of RAM for USER machine code program segments, for example:

```
10 MAX = 32000
20 LIMIT = MAX
```

NOTE, when the MZ-80K is first turned on and the BASIC interpreter loaded LIMIT is set automatically to the highest RAM location available.

DEF FN
(T and D)

This statement defines a user defined function, for example

```
10 DEF  FNA(X) = X * X + X * X
```

User defined functions must be contained on one line.
The variable in parentheses following the variable name is called a dummy variable. A function may be defined to be any expression but it may ONLY have one argument. Other variables used in the expression are considered to be global, and their current values are used in the evaluation.

Following execution of a function definition statement, a user defined function may be used in a similar fashion to the standard BASIC functions, for example in the above case

```
10 DEF FNA(A) = X * X + X * X
20 FOR I = 1 TO 10
30 Y = FNA(I)
40 PRINT I , Y
50 NEXT I
60 END
```

Note, string functions cannot be defined.

Other user defined functions may be used to define a new function. However, a limit of 6 levels of nesting is all that is allowed for example

```
100 DEF FNA(X) = exp (-x)
200 DEF FNB(Y) = Y * Y * FNA(I)
300 DEF FNC(Z) = SIN(Z) * FNB(M)
```

The following user defined mathematical functions are often defined

1. ARCSIN or SIN^{-1}

```
DEF FNA(X) = ATN (X/SQR(1-X*X))
Valid for ABS(X)<1
```

2. ARCCOS or COS^{-1}

```
DEF FNB(X) = ↑↑/2-ATN(X/SQR(1-X*X))
Valid for ABS(X) 1
```

3. SINH

```
DEF FNC(X) = (EXP(X)-EXP(-X))/2
Valid for any X.
```

4. COSH

```
DEF FND(X) = (EXP(X) + EXP(-X))/2
Valid for any X
```

5. TANH

```
DEF FNE(X) = 1-EXP(-X)/(EXP(X)+EXP(-X))*2
Valid for any X
```

6. ARCSINH or $SINH^{-1}$

```
DEF FNF(X) = LN (X + SQR(X*X+1))
Valid for any X
```

7. ARC TANH or $TANH^{-1}$

```
DEF FNG(X) = LN ((1+X)/(1-X))/2
Valid for ABS (X) <1
```

DIM
(T and D)

This statement is used to inform the BASIC interpreter to allocate space for matrices. In SHARP BASIC only one or two dimension matrices are allowed, with a maximum of 256 elements in each dimension, for example

```
10 DIM A(3), B(10)
200 DIM R3(5,5), A$(10,10)
```

Subscripts can be used with values in the range 0 to 255, or can be declared by a variable or equation, for example

```
50 DIM AA(X), AB(X*2)
```

Note, all subscripts start at 0, which implies that DIM A(100) allocates 101 elements to variable A.

A variable must be dimensioned before it is referenced as a matrix element.

If an attempt is made to redimension a matrix during program execution an error will result

Numerical Functions

RND(X)
(T and D)

This is a pseudorandom number generator function. If the argument is 0 or a negative number the function returns the same number as the previous call gave. If the argument is a positive number a random number in the range 0.00000001 to 0.999999999 is generated.

If your require random numbers other than between 0 and 1, then use

```
INT ((B-A+1) * RND (1) + A)
```

which will generate integer random numbers ranging between A and B.

INT(X) (T and D)

The function INT(X) returns the greatest integer less than X, for example

```
INT (94.354) = 94
```
and
```
INT (-94.354) = -95
```

ABS(X) (T and D)

The function ABS(X) returns the "absolute" value of x, i.e. $|x|$., for example

```
ABS (3.44) = 3.44
```
and
```
ABS (-3.44) = 3.44
```

SGN(X) (T and D)

The function SGN(X) returns the "sign" (+ or -) of x, for example

```
SGN (4.5)  = 1
SGN (-4.5) = -1
SGN (0)    = 0
SGN (-0)   = 0
```

SIN(X) (T and D)

This function returns the SINE of x. (x in radians).

COS(X) (T and D)

This function returns the COSINE of x. (x in radians).

TAN(X) (T and D)

This function returns the TANGENT of x (x in radians).

ATAN (T and D)

This function returns the angle, in radians, that is the arc tangent of x. Calculation results in the value between $-\pi/2$ and $\pi/2$ radians.

LN(X) (T and D)

This function returns the natural logarithm of x.

LOG(X) (T and D)

This function returns the common logarithm ($\log_{10}$) of x.

EXP(X) (T and D)

This function returns the natural logarithm raised to the xth power (e^x) and is the inverse of LN(x)

SQR(X) (T and D)

This function returns the square root of x.

SIZE (T and D)

This function (when used in a PRINT statement) returns the number of unused memory bytes.

String Functions

ASC (T and D)

The function ASC (string or string variable) returns the decimal numeric value of the first ASCII character within the string, for example

```
ASC ("?") = 63
ASC ("A") = 65
ASC ("5") = 53

10 B$ = "5"
20 Y  = ASC (B$)
```

LEN (T and D)

The function LEN (string or string variable) returns the number of characters contained in the string or string variable, for example

```
LEN ("TEST")     =  4
LEN ("TEST ONE") =  8
```

In the last example the space is significant
NOTE; LEN (STR$(Y)) = number of print positions required to print the number Y.

VAL (T and D)

The function VAL (string or string variable) returns a numeric constant equivalent to the value in the string or string variable, for example

```
VAL("12.3")  =  12.3
VAL("5E4")   =  5000
VAL("-12.3") = -12.3
```

NOTE: VAL("TWO") generates an error.

LEFT$ (T and D)

The function LEFT$ (string or string variable, N) returns a string of characters from the leftmost to the Nth character in the string or string variable, for example

```
10  X$ = "ONE TWO THREE"
20  A$ = LEFT$(X$,6)
```

After execution of line 20 A$ contains 'ONE TW

RIGHT$ (T and D)

The function RIGHT$ (string or string variables, N) returns a string of characters from the Nth position to the rightmost character in the string or string variable, for example

```
200 X$ = "ONE TWO THREE"
300 A$ = RIGHT$(X$,7)
```

After execution of line 200 A$ contains "O THREE".

MID$
(T and D)

The function MID$ (string or string variable, X, Y) returns a string of characters from the string or string variable beginning with the Xth character from the left, and continuing for Y characters from that point, for example

```
400 X$ = "ONE TWO THREE"
500 A$ = MID$(X$,3,10)
```

After execution of line 500 A$ contains "E TWO THREE"

CHR$
(T and D)

The function CHR$(X) returns a single character string equivalent to the decimal ASCII numeric value of X. This is the inverse of the ASC function, for example

```
CHR$(63) = "?"
CHR$(65) = "A"
CHR$(53) = "5"
```

STR$
(T and D)

The function STR$(X) returns the string value of a numeric value. This is the inverse of the VAL function, for example

```
10  A  = 34567
20  A$ = STR$(A)
```

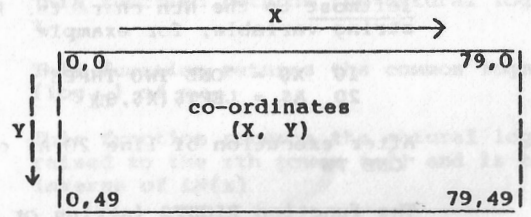
After execution of line 20 A$ contains "34567"

Special Statements

SET

(T and D)

This statement allows a pixal given by the co-ordinates X, Y to be turned on (set to white spot). The X, Y grid on the V.D.U. display is as follows



Examples

```
10 SET 0,0
20 SET A,B
If X>79, then X←X-80 calculated
If Y>49, then Y←Y-50 calculated
```

RESET (T and D)

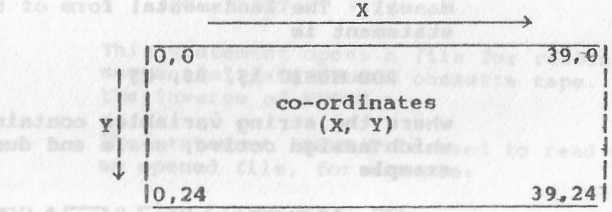
This statement is the reverse of SET and allows a pixel at co-ordinates X, Y to be turned off (reset to black spot) for example

```
 50 RESET 0,0
100 RESET M,49
```

NOTE; in both the SET and RESET statements co-ordinate assignment with a negative value, or X or Y values exceeding 255 may cause a run time error.

CURSOR (D only)

The CURSOR statement is used to move the cursor to any desired point on the V.D.U. screen. The co-ordinate position grid on the V.D.U. is as follows



Examples

```
 80 CURSOR 25,15
 90 PRINT "ABC"
```

In this example the string "ABC" is printed starting at the 26th position from the left of the V.D.U. Screen and the 16th position from the top of the V.D.U. Screen.

GET (T and D)

Inputs one numeral or character from the keyboard. When a key is not pressed a 0 is returned if a numeric variable or a space if a string variable for example

```
10 GET A : IF A<>-1 THEN 10
```

or

```
20 GET A$: IF A$ = "G" THEN 100
30 GOTO 20
```

In both these cases the program will loop until the correct key is pressed:-
1 in the first case and the letter G in the second case.

TEMPO
(T and D)

This statement assigns a tempo to a music sequence. Seven tempo levels are allowed, these are for example

Tempo

1	slowest	- Lento and Adagio
4	moderate	- Moderate, 4 times as fast as TEMPO 1
7	fastest	- Molto Allegro and Presto, 7 times as fast as TEMPO 1

NOTE, if a TEMPO is not assigned, BASIC assumes tempo 4.

MUSIC
(T and D)

A detailed description of this BASIC statement is given on pages 85 to 87 of the SHARP BASIC Manual. The fundamental form of this statement is

```
200 MUSIC A$, B$, C$
```

where the string variables contain characters which assign octive, scale and duration, for example

```
70  A$ = "D□ D□ F□ ♯ CDFA 5R5"
```

denotes lower octive — denotes sharp — denotes rest

INP
(T and D)

This statement reads data from an external device using the universal inteface card. Two forms have been implemented.

a. tape BASIC, 10 INP # port, variable
b. disk BASIC, 10 INP @ port, variable

where "port" is an input number assigned on the innerface card, range 0 to 255_{10}, and "Variable: is a BASIc variable which takes a value in the range 0 to 255_{10}

OUT
(T and D)

This statement outputs data to an external device using the universal interface card Two forms have been inplemented.

a. tape BASIC, 20 OUT # port, variable
b. disk BASIC, 20 OUT @ port, variable

where "port" is an output port number assigned on the interface card, range 0 to 255_{10}, and "Variable" is a BASIC variable which takes a value in the range 0 to 255_{10}

Storing data on Cassette Tape
(Cassette BASIC only)

WOPEN

This command opens a file for writing sequential data to cassette tape. The data may consisst of a sequence of numerical and/or string items. An optional file name may be included after the WOPEN key-word, for example:

```
100  WOPEN "TEST"
```

A maximum of 16 characters are allowed in a filename. NOTE spaces are significant.

PRINT/T

The statement PRINT/T is used to write data to an opened file, for example:

```
200  PRINT/T  Y,X$
```

ROPEN

This statement opens a file for reading sequential data from a cassette tape. It is the inverse of WOPEN.

INPUT/T

The statement INPUT/T is used to read data to an opened file, for example:

```
300  INPUT/T  Y, X$
```

NOTE, when using INPUT/T if a file is NOT opened for reading an error occurs.

CLOSE

This statement is used to CLOSE a file after it has been opened with either the WOPEN or ROPEN statements. Note after the WOPEN and ROPEN statements are used the file must be closed following write or read operations - if not an error occurs.

The following examples indicate the use of this group of statements.

1. Writing data to cassette

```
10  WOPEN "DATA"        ) Program writes a
20  FOR X = 1 to 99     ) sequence of numbers
30  PRINT/T X           ) from 1 to 99 in
40  NEXT X              ) increasing order to
50  CLOSE               ) tape and calls the
60  END                 ) file "DATA"
```

2. Reading data from cassette

```
10  ROPEN "DATA       ) Program reads a
20  FOR X = 1 to 99   ) sequence of numbers
30  INPUT/T DATA      ) from 1 to 99 in the
40  PRINT DATA        ) order read from tape
50  NEXT X            ) file "DATA"
60  CLOSE             )
70  END               )
```

A more complex example of the use of the data storage commands is given in the program at the end of this issue of these user notes.

Numerical Variables

Numeric variables may be one or two alphanumeric characters in length, but the first character must be alphabetic. Longer variable names are identified by the first two characters only, for example:

```
MONDAY
MO
MO123
```

are all indistinguishable to the BASIC interpreter. Also BASIC keywords, such as FOR, IF, TO etc., may not be used as variable names, nor may non-alphanumeric graphic characters.

Range and accuracy

Numeric variables are allowed values between approximately 10^{-38} and 10^{38} with 7 to 8 figures of accuracy. Strings may be from 0 to 255 characters in length.

Arrays

Arrays may be used with both numeric and string variables. One or two dimensions is allowed with 0 to 255 elements in each dimension.

Numerical operations

1. \- negation
2. ↑ exponention (raise to a power ie x^3 written as x↑3)
3. * multiplication
4. / division
5. + addition
6. - subtraction

These numerical operators have their usual meaning and may be used in expressions which including parenthesis to make explicit the order of evaluation. Note, parenthesis may be nested.

Operators

1. > greater than
2. < less than
3. <> or >< not equal to
4 <= or =< less than or equal to
5. >= or => greater than or equal to

In IF statements a symbol + is used to represent the local OR function and a * to represent the logical AND function, for example:

```
10 IF (A>X) + (B Y) THEM 1000
```

executes statement 1000 if A> X OR B> Y;

```
20 IF (A>X) * (B Y) THEN 1000
```

executes statement 1000 if A>X AND B>Y.

Operator evaluation order

Expressions are evaluated in the following order, scanning from left to right:

Brackets first, then

1.
2. negation
3. * / from left to right
4. + - from left to right
5. = from left to right

Two separate numbers or variables may not be placed next to each other, similarly two operators, unless the second is + or -, for example

1. P+-25 is equivalent to P-25
2. M*-5 = -5*M but M-*5 is illegal
3. 3 2*7 + 5/10*2 is calculated as

a. 3 2 = 9, then
b. 9*7 = 63, then
c. 5/10 = 0.5, then
d. 0.5*2 = 1 , then
e. 63 + 1 = 64

String operators

There is only one, concatenation which is denoted using a + sign, for example

```
1000 A$ = "A" : B$ = "B"
2000 C$ = A$ + B$ ? C$
RUN
AB
```

Strings may also be compared for equality in IF statements, for example

```
20 IF A$ = "DATA" THEN 1000
```

executes statement 1000 if A$ = "DATA"

Real Time Clock

The string variable TI$ is used to represent a digital clock function in SHARP BASIC. To set the MZ 80K internal clock equate TI$ to a string containing hours, minutes and seconds, for example:

```
10 TI$ = "102030"
```

Sets the internal clock to 10 hours, 20 minutes and 30 seconds. The string must be six characters long between quotation marks. NOTE, on switching on the MZ-80K TI$ is set to 00 hours, 00 minutes and 00 seconds and must be set by the user to the correct time before use.

A possible use of the clock is for timing program execution, for example:

```
10 A$ = "123456" : TI$ = A$
20 PRINT A$
30 FOR T=1 to 5000:NEXT T
40 B$ = TI$ : PRINT B$
50 END
RUN

123456 ) 4 seconds taken to run program-
123500 ) mainly loop at line 30
```

Error Control Statements (Disk BASIC only)

ON ERROR GOTO "Line number"

This statement is used to inform the BASIC interpreter which statement to continue with if an error occurs during execution of a program. "Line number" is the statement number of a block of statements which deals with the error. Note, the ON ERROR GOTO statement must have been executed by BASIC before an error occurs. Hence, this statement is normally placed near the start of a program.

ERN and ERL

ERN and ERL are two variables which hold the "error number" and "line number" of the statement at which the error occurred. They may be tested using IF statements to determine the action the program needs to take to correct the error. These variables are normally used in the block of statements which control is transferred to in the ON ERROR GOTO "Line number" statement.

RESUME

This instruction informs the BAISC interpreter to return control to the main program where the error occurred.

Four modes are possible, these are

RESUME	start program execution at the statement where the error occurred.
RESUME NEXT	start program at the statement after the statement where the error occurred.
RESUME "Line number"	return control to the main program at line number"
RESUME 0	transfer control to the start of the program i.e. statement with the lowest line number.

To be continued in issue 2 of the SHARPSOFT-USER NOTES

In issue 2 we will include notes on disk BASIC file handling and the USR statement.

Storage of BASIC programs

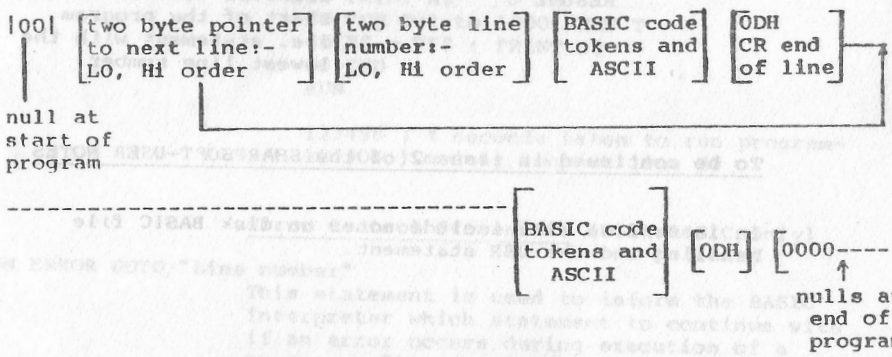
SHARP BASIC stores lines of source code in a compacted form using the "token" concept and ASCII characters. This technique has become a standard procedure in most of the popular versions of the BASIC language.

In SHARP BASIC all reserved words, for example, FOR, GOTO, END, = are stored as a one-byte "token". These tokens have their highest bit set to logic 1, which implies they are greater than 7FH or 127_{10}. A few examples are

141_{10} = NEXT, 135_{10} = FOR and 203_{10} = SIZE

A complete list of tokens for the SHARP cassette BASIC, with extensions for the disk BASIC, is given at the end of these notes.

The remainder of the letters and numbers in a BASIC statement are stored as the ASCII characters, and graphics characters, entered from the keyboard. The line number at the start of a line is stored as a two byte binary number. Each stored BASIC line also holds a two byte pointer containing the absolute address in memory of the next line of BASIC code. By using this linked pointed the BASIC interpreter can search rapidly for a given line number. The format for each line of stored BASIC is:



The position in RAM of the initial null at the beginning of a program is :

Tape BASIC	4805 H	(18437_{10})
Disk BASIC	652B H	(25899_{10})

TOKENS

1. TAPE BASIC

token	keyword	token	keyword	token	keyword
128	REM	157	WOPEN	190	#
129	DATA	158	ROPEN	191	/
130	LIST	159	CLOSE	192	LEFT$(
131	RUN	160	BYE	193	RIGHT$(
132	NEW	161	LIMIT	194	MID$
133	PRINT	162	CONT	195	LEN(
134	LET	163	SET	196	CNR$(
135	FOR	164	RESET	197	STR$
136	IF	165	GET	198	ASC(
137	GOTO	166	INP#	199	VAL(
138	READ	167	OUT#	200	PEEK(
139	GOSUB	168/172	unused	201	TAB(
140	RETURN	173	THEN	202	SPC(
141	NEXT	174	TO	203	SIZE
142	STOP	175	STEP	204/206	unused
143	END	176		207	
144	ON	177		208	RND(
145	LOAD	178	=	209	SIN(
146	SAVE	179	=	210	COS(
147	VEIFY	180	=	211	TAN(
148	POKE	181	=	212	ATN(
149	DIM	182	=	213	EXP(
150	DEF FN	183		214	INT(
151	INPUT	184		215	LOG(
152	RESTORE	185	AND	216	LN(
153	CLR	186	OR	217	ABS(
154	MUSIC	187	NOT	218	SGN(
155	TEMPO	188	+	219	SQR(
156	USR(	189	-	220/255	unused

2. Disk BASIC extensions

Not all the keywords have been decided - more word needed!
The following list gives some of the extensions

token	keyword
224	ERROR
240	RESUME

A typical storage pattern in memory for a BASIc program is given below; consider the following segment of the start of a tape BASIC program:

```
60000  REM Dump utility
60010  REM for SHARP cassette BASIC
60020  REM change line number 60050
60025  REM to POKE 10167,1
60030 PRINT "C":H$="0123456789ABCDEF":K1=4096:K2=256:K3=16
```

HEX address	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
											REM	sp	d	u	m	p
4800	EA	E9	49	02	02	00	19	48	60	EA	80	20	44	A5	B3	9E
	sp	u	t	i	l	i	t	y	CR					REM	sp	f
4810	20	A5	96	A6	B8	A6	96	BD	0D	38	48	EA	EA	80	20	46
	o	r	sp	S	H	A	R	P	sp	c	a	s	s	e	t	t
4820	B7	9D	20	53	48	41	52	50	20	9F	A1	A4	A4	92	96	96
	e	sp	B	A	S	I	C	CR					REM	sp	c	h
4830	92	20	42	41	53	49	43	0D	57	48	74	EA	80	20	9F	98
	a	n	g	e	sp	l	i	n	e	sp	n	u	m	b	e	r
4840	A1	B0	97	92	20	B8	A6	B0	92	20	B0	A5	B3	9A	92	9D
	sp	6	0	0	5	0	CR					REM	sp	t	o	sp
4850	20	36	30	30	35	30	0D	6D	48	79	EA	80	20	96	B7	20
	P	O	K	E	sp	1	0	1	6	7	)	1	CR			
4860	50	4F	4B	45	20	31	30	31	36	37	2C	31	0D	A1	48	7E
		PRINT	"	C	"	:	H	$	=	"	0	1	2	3	4	5
4870	EA	85	22	16	22	3A	48	24	B6	22	30	31	32	33	34	34
	6	7	8	9	A	B	C	D	E	F	'	:	K	1	=	4
4880	36	37	38	39	41	42	43	44	45	46	22	3A	4B	31	B6	34
	0	9	6	:	K	2	=	2	5	6	:	K	3	=	1	6
4890	30	39	36	3A	4B	32	B6	32	35	36	3A	4B	33	B6	31	36
	CR															
48A0	0D	B5	48	-	-	-	-	-	-	-	-	-	-	-	-	-

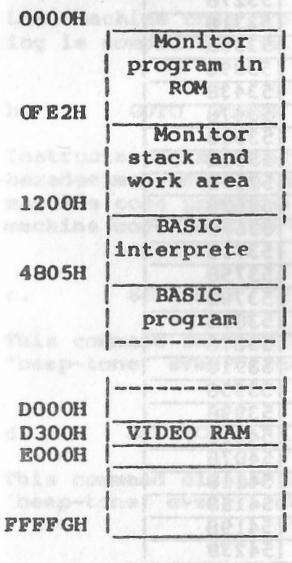
KEY:

1. ( ) denotes tokens
2. Lo Hi pointer to start of next line
3. Lo Hi BASIC line number (in hexadecimal)
4. CR carriage return at end of the line.

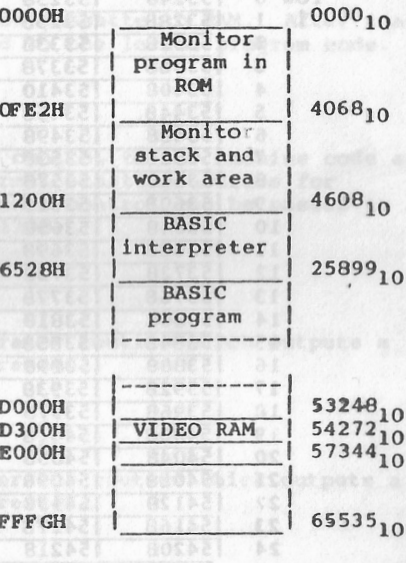
A knowledge of the BASIC keyword tokens and the storage format allows a number of interesting utility programs to be written, for example BASIC line renumbering routines, programs to list identifiers and the line numbers at which they occur and packing and unpacking routines. This is an area of programming where you can write useful routines which will help in developing other software.

MZ-80K Memory Maps

1. TAPE BASIC



2. DISK BASIC



2. VIDEO MEMORY MAP

column-	0 - 9	10 - 19	20 - 29	30 - 39
row 0	53248	53258	53268	53278
1	53288	53298	53308	53318
2	53328	53338	53348	53358
3	53368	53378	53388	53398
4	53408	53410	53428	53438
5	53448	53458	53468	53478
6	53488	53498	53508	53518
7	53528	53538	53548	53558
8	53568	53578	53588	53598
9	53608	53618	53628	53638
10	53648	53658	53668	53678
11	53688	53698	53708	53718
12	53728	53738	53748	53758
13	53768	53778	53788	53798
14	53808	53818	53828	53838
15	53848	53858	53868	53878
16	53888	53898	53908	53918
17	53928	53938	53948	53958
18	53968	53978	53988	53998
19	54008	54018	54028	54038
20	54048	54058	54068	54078
21	54088	54098	54108	54118
22	54128	54138	54148	54158
23	54168	54178	54188	54198
24	54208	54218	54228	54238

Monitor Machine Code Commands and Routine entry addresses

1. Monitor Commands

Only five entry commands are allowed; these are

a. LOAD (CR)

Load machine code program stored on cassette to RAM. After loading is completed control is passed to the loaded program code.

b. GOTO $HHHH (CR)

Instructs the monitor to transfer control to the machine code at hexadecimal address HHHH. The normal starting address for machine code programs is 1200H, hence control can be passed to a machine code routine using GOTO $1200.

c. SG (CR)

This command enables a sound generator routine which outputs a "beep-tone" every time a key is pressed.

d. SS (CR)

This command disenables sound generator routine which outputs a "beep-tone" every time a key is pressed.

e. FD (CR)

Causes control to be transferred to F000H. This is the address of the floppy disk bootstrap routine (held in ROM on disk interface board).

2. Monitor Machine Code routine entry points and functions.

Address (hexadecimal)	Function
0000	Monitor cold start entry point.
0003	USER input routine.
0006	Output CRLF to V.D.U. and move cursor to start of the next line.
000C	Output a space on V.D.U. at current cursor position.
0012	Video driver.
0015	Output message to V.D.U.
001B	Keyboard driver.
0021	Output header to tape.
0024	Output data to cassette tape.
0027	Load header from cassette tape.
002A	Load data from cassette tape.
002D	Checks data on cassette tape BUT dos not store data in RAM - just compares data on tape with contents of RAM.
0030	Music driver.
0033	Set real time clock.
003B	Read time from real time clock.
003E	Keyboard beeper driver (Call Bell).
0041	Tempo driver routine.
0044	Sound driver routine.
0047	Stop sound.

Notes on Monitor ROM routines

These routines are of special interest to people writing applications in machine code. The listing of the complete monitor assembly code is available in the user publication

" A Commented Assembly Listing for the SHARP MZ-80K Monitor"

0003H

Inputs one line of text from the keyboard. Specify the start address of the RAM location for storing data in the Z80 registe pair DE before entry to the rouintne. The end of text string marker is a CR (ODH) character. The maximum number of characters in the input string is 80 including the CR at the end of the string. Note, key input is echoed to the V.D.U. screen. Cursor control characters are allowed in the string. When the SHIFT and BREAK keys are pressed the BREAK code is entered in RAM at the address specified by the contents of DE. A CR is also entered at address (DE)+1. Note, all registers are stored on entry to this routine and 15 stack levels are required.

0006H

Th routine outputs a CRLF to the V.D.U. screen an moves the cursor to the start of the next line. All registe s are stored except A and F. Also 8 stack levels are require by this routine.

000CH

This routine outputs to the V.D.U. sc een one space at the current cursor position. All regis ers are stored except A and F. Also 13 stack levels are equired by this routine.

0012H

This routine outputs the ASCII code for a character stored in accumulator A to the current cursor position on the V.D.U. screen. If the contents of A is a carriage return (ODH) character a carriage return is displayed on the V.D.U. Si l ly cursor control codes 11H to 16H cause movement of the curso , , , , H and C. All registerd are stored except A and F. Also 13 stack levels are required by this routine.

0015H

Outputs a message on the V.D.U. screen starting at the current cursor position. The address of the start of the message in RAM is entered in the Z80 DE register pair before entry to the routine. The message must be a string of ASCII characters terminated by an ODH character. NOTE, the carriage return character at the end of the message string is NOT executed. However, cursor control characters are executed. All registeres are stored. Also 13 stack levels are required by this routine.

001BH

Inputs to accumulator A the ASCII code of a key which has been pressed. The code for the pressed key is NOT echoed to the V.D.U. display drive. NOTE key bounce is not prevented by this routine. All registers other than A and F are stored. Also 9 stack levels are required by this routine.

Special Keys	ue set in A
DEL	60
INST	61
CAP	62
SML	63
BREAK	64
CR	66

0030H

This routine plays music, executing a string of ASCII data characters to generate the sound sequence. Specify the start address of the music data string in the Z80 register pair DE before entry to the routine. The music data is a string which is arranged as a sequence of pitch and duration ASCII control codes. The string sequence is the same as that shown on pages 84 and 85 of the SHARP BASIC manual. The end of string marker should be an ODH or C8 character. On returning from the subroutine Z80 register C is set to 0 if the music sequence was completed or C is set to 1 if the sequence was not completed, due to the BREAK key having been pressed while the music was playing. All registers other than A and F are stored. also 7 stack levels are required by this routine.

0033H

This routine sets the built-in real time clock, Note the clock is started by a call to this routine. The clock is set to the contents of the following registers.

If accumulator A = 0 then time is AM
If accumulator A = 1 then time is PM

DE contains the time in seconds (in binary - 2 bytes)

The contents of A and DE must be set before entry to the routine. All registered are stored other than A F and DE. Also 3 stack levels are required by this routine.

003BH

This routine reads the time from the real time clock. The Z80 internal registers hold the real time.

Accumulator A = 0 if the time is AM
Accumulator A = 1 if the time is PM

DE contains the time in seconds (in binary - 2 bytes)

All registers are stored other than Af. Also 1 stack level is required by this routine.

003EH

This routine simulates the ringing of a bell. A midle A note (roughly 440HZ) is generated for a few seconds. All registers are stored other than A F. Also 5 stack levels are required by this routine.

0041H

This routine sets the tempo of the music generator. The value (1 to 7) of the tempo must be entered into the Z80 accumualtor A before calling this routine.

A	Tempo	NOTE, the codes set into
01	slowest tempo	accumulator A are
04	medium tempo	in binary NOT ASCII
07	fastest tempo	characters.

0044H

This routine generates a sound specified by the relation

$$2/nn^1 \text{ MHZ}$$

where nn^1 is 16 bit binary number.

n^1 is stored at address 11A1H, and
n is stored at address 11A2H

Values for n^1 and n must be stored before calling this routine only registers B C and D E are stored. Also 3 stack levels are required by this routine.

0047H

This routine stops the operation of the sound generator. All registers are stored except A F. Only 1 stack level is required by this routine.

Other useful routines

03DAH

This routine takes the lower 4 bits of the contents of the accumulator and converts it into an ASCII code. The resulting code is entered into A before leaving the routine. All registers are stored except A and F. Only 1 stack level is required by this routine.

03F9H

This routine performs the reverse function of the routine at 03DAH. The 8 bits of accumulator A are assumed to represent an ASCII code. This code is converted into a hexadecimal number. The resulting number is stored in the accumulator before leaving the routine. If the carry flag is 0 on returning to the calling routine the number in the accumulator is a hexadecimal number. All registeres are stored except A and F and H L. Only 1 stack level is required by this routine.

0410H

This routine converts a four character ASCII string into a hexadecimal number. The hexadecimal number is stored in register HL. Before calling this routine the address in RAM of the start of the ASCII string must be entered in registered pair DE, for example:

```
"3", "1", "A", "5"     -- ASCII string
 ↑
 |_____ DE

HL  = 31A5H
```

If the carry flag is 0 on returning to the calling routine the number in HL is hexadecimal. All registers are stored except AF and HL. Only 2 stack levels are required by this routine.

041FH

Similar to the routine at 041FH. Converts two character ASCII string pointed to by DE to hexadecimal and sets result in accumulator A. If the carry flag is 0 on returning to the calling routine the number in A is hexadecimal. All registers except AF and DE are stored. Only 2 stack levels are required by this routine.

09B3H

While flickering the cursor this routine waits for a key to be pressed. When a key is pressed it sets the display code for the key in accumulator A and returns. All registers except AF are stored. Only 7 stack levels are required by this routine.

0BB9H

This routine converts an ASCII charactered string in accumulator A into its display code. All registeres except AF are stored. Only 3 stack levels are required by this routine.

0BCEH

This routine converts a display code stored accumulator A into its ASCII code. All registers except AF are stored. Only 3 stack levels are required by this routine.

0DDCH

This routine may be used to control the display on the V.D.U. screen. The value of accumulator A determines the controlling function. Controlling codes are:

A (hex)	Function
C0	scrolling
C1	same as ↓ key
C2	same as ↑ key
C3	same as ← key
C4	same as → key
C5	same as HOME key
C6	same as CLR key
C7	sames as DEL key
C8	same as INST key
C9	same as CAB key
CA	same as SML key
CD	same as CE key

All registeres are stored. Only 10 stack levels are required by this routine.

0FB1H

This routine determines the current position of cursor on the V.D.U. screen. The cursor position is stored as a binary number in Z80 register pair HL. Remember there are 25 lines of 40 characters, yielding 1000_{10} possible positions between 0_{10} to 999_{10}. All registers are stored except AF and HL only 5 stack levels are required by this routine.

Cassette tape directory program: Cat-file

"Cat-file" is a short BASIC program which allows a directory to be recorded and maintained at the start of a cassette tape. The program consists of the following sections:-

1. Initialisation of a new tape (i.e. blank tape)

An initialised directory of up to 20 file names is recorded at the start of a blank tape by this routine. The file format is:

```
"name", "type", "code", "counter", "counter",
                        "start", "finish".
```

"name"	is entered as a field of 16 * characters
"type"	is entered as INI to denote an initialised record.
"code"	is entered as 0.
"counter"	is entered as 0.
"start"	
"counter"	is entered as 0.
"finish"	

2. Enter file name in directory

Name, type, code, counter start and counter finish data are entered from the keyboard at the computers request. Suggested type and codes for different types of files are:

		type	code
1.	Object source	OBJ	01
2.	BASIC source	TXT	02
3.	BASIC sequential data	BSD	03
4.	Zen assembler source	ZEN	04

NOTE, "name": may be any sequence of up to 16 ASCII characters.

3. Delete file entry

Enter name of file on request.

4. List directory on V.D.U. screen

5. Finish using Cat-file program

At the start of the program the user is requested to enter a command - replies may be in short form, for example:

1.	Initialisation	I	or	INIT	
2.	Enter file name	E	or	ENTER	
3.	Delete file name	D	or	DEL	
4.	List directory	L	or	LIST	
5.	Finish using program	F	or	FIN	or FINISH

Suggested improvements to use program:

1. add a section which sorts directory by name, type or code or length of source or counter start value.

2. add statements for outputting information to a hard copy printer.

When using this program DO REMEMBER to take a note of the counter start and counter finish values when you record a program on tape. Also remember to initialise th tape position counter.

```
10 PRINT"■♦♦♦♦♦   Cat-file   ♦♦♦♦♦"
50 DIMNA$(20),TY$(20),CO(20),CS(20),CF(20)
60 S1$="****************":S2$="INI"
70 CO=0:CS=0:CF=0
100 INPUT"Command ? ";A$:C$=LEFT$(A$,1)
110 IFC$="I"THEN1000
130 IFC$="D"THEN3000
140 IFC$="E"THEN1500
150 IFC$="L"THEN4000
160 IFC$="F"THENEND
200 GOTO100
1000 PRINT"■Initialization of new tape"
1010 INPUT"Have you rewound the tape ? ";M$
1020 IFASC(M$)<>89THEN1000
1030 WOPEN"Cat-file"
1040 FORI=1TO20
1050 PRINT#TS1$,S2$,C0,CS,CF
1060 NEXTI
1070 CLOSE
1080 GOTO100
1500 PRINT"■Enter file name in directory"
1510 INPUT"Have you rewound the tape ? ";M$
1520 ROPEN"Cat-file"
1530 FORI=1TO20
1540 INPUT#TNA$(I),TY$(I),CO(I),CS(I),CF(I)
1550 NEXTI
1560 CLOSE
1570 PRINT"Name of file to be"
1575 INPUT"     entered in Cat-file ? ";M$
1580 IFM$=""THEN1570
1590 IFLEN(M$)>16THENPRINT"File name too long ":GOTO1570
1600 IFLEN(M$)<16THENM$=M$+" ":GOTO1600
1610 FORI=1TO20
1620 IFM$=NA$(I)THEN:GOSUB1910:GOTO100
1630 NEXTI
1650 FORI=1TO20
1660 IFNA$(I)=S1$THEN1700
1670 NEXTI
1680 PRINT"Cat-file  FULL  with 20 NAME   GOTO100
1700 INPUT"Type of file ? ";TY$
1710 INPUT"Code for file ? ";CD
1720 INPUT"Counter start value ? ";SV
1730 INPUT"Counter finish value ? ";FV
1740 PRINT"File parameters are :-  "
1750 PRINT"File     name      = ";M$
1760 PRINT"         type      = ";TY$
1770 PRINT"         code      = ";CD
1780 PRINT"         start     = ";SV
1790 PRINT"         finish    = ";FV
1795 PRINT:PRINT:PRINT
```

```
1800 INPUT"IS THIS OK ? ";A$
1810 IFASC(A$)<>89THEN100
1820 NA$(I)=M$:TY$(I)=TY$:CO(I)=CD
1830 CS(I)=SV:CF(I)=FV
1840 INPUT"Have you rewound the tape ? ";M$
1850 IFASC(M$)<>89THEN1840
1860 WOPEN"Cat-file"
1870 FORI=1TO20
1880 PRINT#TNA$(I),TY$(I),CO(I),CS(I),CF(I)
1890 NEXTI
1895 CLOSE
1900 GOTO100
1910 PRINT"File ";M$
1920 PRINT"     already exists on Cat-file"
1930 RETURN
3000 PRINT"@Delete file entry"
3010 INPUT"Have you rewound the tape ? ";M$
3020 IFASC(M$)<>89THEN3000
3030 ROPEN"Cat-file"
3040 FORI=1TO20
3050 INPUT#TNA$(I),TY$(I),CO(I),CS(I),CF(I)
3060 NEXTI
3070 CLOSE
3080 PRINT"Name of file to be"
3085 INPUT"        DELETED from Cat-file ";M$
3090 IFM$=""THEN3080
3100 IFLEN(M$)>16THENPRINT"File name too long ":GOTO3080
3110 IFLEN(M$)<16THENM$=M$+" ":GOTO3110
3120 FORI=1TO20
3130 IFM$=NA$(I)THENGOTO3160
3140 NEXTI
3150 PRINT"File ";M$
3155 PRINT"     does not exist":GOTO3080
3160 NA$(I)=S1$:TY$(I)=S2$:CO(I)=0:CS(I)=0:CF(I)=0
3170 PRINT"Please rewind the tape "
3180 PRINT"Press any key to restart when rewound"
3190 GETM$:IFM$=""THEN3190
3200 WOPEN"Cat-file"
3210 FORI=1TO20
3220 PRINT#TNA$(I),TY$(I),CO(I),CS(I),CF(I)
3230 NEXTI
3240 CLOSE
3250 GOTO100
4000 PRINT"@List directory"
4010 INPUT"Have you rewound the tape ? ";M$
4020 IFASC(M$)<>89THEN4000
4030 ROPEN"Cat-file"
4040 FORI=1TO20
4050 INPUT#TNA$(I),TY$(I),CO(I),CS(I),CF(I)
4060 PRINTNA$(I);" ";TY$(I);" ";CO(I);" ";CS(I);" ";CF(I)
4070 NEXTI
4080 CLOSE
4090 GOTO100
```

Mektronic Consultants Limited have been working on a new interface to be used for systems control etc. We have not yet seen one but hope too in the very near future. This is the press release they sent to us:-

"An interface unit for the Sharp MZ80K desk-top computer is now available from Mektronic and simply plugs into the 50-way User Port to give the MZK80K 8 input and 8 output channels for system control.

The output voltage can be within the range 8V to 24V at 300mA depending on the supply connected.

The input channels have a variable switching point which means that the input voltage switching level can be set at any point between -24 and +24.

Input/Output operations can be easily performed using PEEK and POKE commands. Programming is as simple at that!

The Sharp interface unit enables the computer to monitor and control systems and display results and procedures. Because the unit does not impair the use of other 1/O devices, results can be stored for later analysis or compared in real-time with known limits. The interface unit greatly enhances the capabilities of the Sharp MZ80K and is available from:- "

MEKTRONIC CONSULTANTS,
Linden House,
116 Rectory Lane,
Prestwich,
Manchester M25 5DB.

We hope that in the next issue of the User Notes to be able to give you more information like price, what it can control and how.

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