(4)

SHARPSOFT

SHARPSOFT USER NOTES

Issue No 4

With this issue of the SHARPSOFT USER NOTES you should have received two tapes. These tapes contain our MZ-80K implementation of the fig. FORTH 8080 package. For those readers with MZ-80B computers we will be releasing a MZ-80B version of our FORTH package sometime later this year. As soon as this is available we will let you know through these user notes.

We hope you enjoy FORTH – it's fun to play with and very fast. FORTH requires study and practice if you are to become proficient in this language. To help new FORTH programmers we have included in this issue a FORTH Tutorial with examples. Further support will follow during the year with more programs and tutorial examples. Please send us your comments, queries and "PROGRAMS" so that we can publish them in the next issue.

If you have a friend with an MZ-80K who wishes to have a copy of the FORTH tapes – PLEASE ask him or her to subscribe to our user notes *rather than COPY* your tapes. Remember good quality software takes time to write and is expensive to produce. Also we rely on subscriptions to keep publishing these user notes and to provide our readers with low cost software.

To date we have received very little feedback from our readers concerning the SHARP PASCAL and our Tiny C packages – please write to us if you are using these packages with your comments and programs for publication.

This issue is the largest we have produced so far. Within the pages we hope everyone will find something of interest. A number of readers have asked for information on the XTAL—BASIC package – this is included in the form of an article by the author of this software.

1982 promises to be an interesting year for SHARP computer users. Our future plans for this magazine include more articles on FORTH, Tiny C, CP/M and BASIC plus a new column on hardwre. Our first hardware project has been the design of a simple, but comprehensive, ROM burner add-on board to the MZ-80K and MZ-80B parallel ports. More details on this project in our next issue.

The first of our Z80 assembler articles appears in this issue. Your letters and comments are, of course, also featured.

Mike Brinson
Editor

SHARPSOFT fig-FORTH 8080 v1.1
for the
MZ80K
by
Mike Brinson
and
Alan Grey

Thank you for buying this copy of FORTH. If you have any questions regarding this package, please direct them, in writing, via the distributer.

SHARPSOFT FORTH is based on the 8080 fig-FORTH version 1.1. We have rewritten the CP/M mass storage routines as a cassette based virtual memory sytem. The package will run on a MZ80K computer with 48k of RAM. The FORTH package is provided through the courtesy of

THE FORTH INTEREST GROUP,P.O. BOX 1105,SAN CARLOS
CA 94070,USA.

SHARPSOFT FORTH is distributed on two cassette tapes with these instructions for their use. Before either of these tapes are run on your computer make sure that the record protect lugs are removed from the top of the cassettes. This will stop accidental erasure of your FORTH master tapes.

The first tape contains the fig-FORTH language and operating system. Side one of the second tape contains a text editor, for program preperation, and side two of the second tape a FORTH 8080 assembler. Both these programs are written in FORTH and may be loaded, when required, by the FORTH operating system.

Loading FORTH

Place the first FORTH tape in the MZ80K cassette recorder,rewind and type LOAD<CR>;in response to the SP-1002 monitor prompt. The symbols <CR> denote that the yellow carriage return key was pressed. NOTE this loading procedure is identical to the SHARP BASIC SP-5025 interpreter.

On completion of the load sequence the FORTH header appears on the display V.D.U.

8080 fig-FORTH 1.1

The user may now program the MZ80K in FORTH. A synopsis of the 8080 fig-FORTH words is given at the end of these notes.

The FORTH editor

A listing of the FORTH editor is given at the end of these notes. This editor is the standard FORTH text editor described in the fig-FORTH installation manual. A detailed description of its operation is presented in the book "A systems guide to fig-FORTH" by C.H. Ting.

To load the editor place the second tape in the cassette recorder and rewind side one of the tape. Next enter

DECIMAL 7 LOAD <CR>

Press the cassette recorder "PLAY" key when requested. To use the editor vocabulary after loading is completed enter

EDITOR <CR>

The editor commands are:-

1.	H	Hold numbered line at PAD.
2.	E	Erase line with blanks.
3.	S	Spread making line # blank
4.	D	Delete line,but hold in PAD.
5.	M	Move cursor by signed amount,print its line.
6.	T	Type line #,save also in PAD.
7.	L	Re-list screen.
8.	R	Replace line #,from PAD.
9.	P	Put following text on line #.
10.	I	Insert text from PAD into line #.

Screen editing commands:-

1.	CLEAR	Clear screen by number.
2.	FLUSH	Write all updated blocks to tape.
3.	COPY	Duplicate screen-2 to screen-1.

The 8080 assembler.

A listing of the fig-FORTH 8080 assembler is given at the end of these notes. A detailed description of the FORTH assembler is given in Dr Ting's book.

To load the FORTH assembler place the second tape in the MZ80K cassette recorder and rewind side two of the tape. On completion of this operation enter:-

DECIMAL 7 LOAD <CR>

Press the cassette recorder "PLAY" key when requested. To use the assembler vocabulary after loading is completed enter:-

ASSEMBLER <CR>

Screen storage.

In this version of fig-FORTH screens are stored on cassette tape. Before programs or data can be stored on a tape, the tape must be FORMATED. We have included the word P-HEADER in the FORTH dictionary to allow you to do this.

The format procedure ia as follows:-

1. Place the tape to be formated in the MZ80K cassette recorder and rewind the tape.
2. Enter from the keyboard

   FORTH DECIMAL <CR>

3. Then type:-

   : FORMAT 25 0 DO I P-HEADER LOOP ; <CR>

4. Typing

   FORMAT <CR>

   and pressing the "RECORD/PLAY" keys as requested formats the tape.

NOTES

1. Roughly 25 screens can be stored on a C90 tape.

2. The FORTH word P-HEADER writes a 1K buffer to the cassette tape with screen numbers as the file header.

3. Once formated a tape may be written to or read from using the FORTH virtual memory words LIST,FLUSH,LOAD and CLEAR etc.

Special features.

1. If you have a printer pressing the SHIFT and INSERT keys together,toggles the printer on/off.

2. Screens 4 and 5 on the editer side of tape two contain the fig-FORTH error messages. When present in RAM messages may be used,rather than numbers,by entering:-

   FORTH DECIMAL 1 WARNING !

Starting FORTH.

If you have never programmed a computer using the FORTH language then the previous instructions will probably make very little sense to you ! We suggest that you read a number of basic articles before attempting to program in FORTH on your MZ80K.

The following articles will help you learn FORTH.

1. Brinson M.E.,Not fith but FORTH,a computer language that builds itself,Computing Today,October 1981,p34-35.

2. James J.S.,FORTH for microcomputers,Dr Dobb's Journal of Calisthenics and Orthodontia,No. 26,p21-27.

3. James J.S.,FORTH a tutorial introduction,Byte,August 1980,p100-126.

4. Katzen H.,Invitation to FORTH.

More advanced reference works are listed below:-

1. Using FORTH,FORTH Inc.,1980.

2. Systems guide to fig-FORTH by Ting.

3. Threaded interpretive languages by Loeliger.

4. Installation manual for fig-FORTH.

5. Source listing of 8080 fig-FORTH.

The above books and the 8080 fig-FORTH source listing are available from

MOUNTAIN VIEW PRESS,USA.

fig-FORTH GLOSSARY

This glossary contains all of the word definitions in Release 1 of fig-FORTH. The definitions are presented in the order of their ASCII sort.

The first line of each entry shows a symbolic description of the action of the procedure on the parameter stack. The symbols indicate the order in which input parameters have been placed on the stack. Three dashes "---" indicate the execution point; any parameters left on the stack are listed. In this notation, the top of the stack is to the right.

The symbols include:

addr	memory address
b	8 bit byte (i.e. hi 8 bits zero)
c	7 bit ASCII character (hi 9 bits zero)
d	32 bit signed double integer, most significant portion with sign on top of stack
f	boolean flag. 0 = false, non-zero = true
ff	boolean false flag = 0
n	16 bit signed integer number
u	16 bit unsigned integer
tf	boolean true flag = non-zero

The capital letters on the right show definition characteristics:

C	May only be used within a colon definition. A digit indicates number of memory addresses used, is other than one.
E	Intended for execution only.

L0 Level Zero definition of FORTH-78
L1 Level One definition of FORTH-78
P Has precedence bit set. Will execute even when compiling.
U A user variable.

Unless otherwise noted, all references to numbers are for 16 bit signed integers. On 8 bit data bus computers, the high byte of a number is on top of the stack, with the sign in the leftmost bit. For 32 bit signed double numbers, the most significant part (with the sign) is on top.

All arithmetic is implicitly 16 bit signed integer math, with error and under-flow indication unspecified.

! n addr – – – LO
Store 16 bits of n at address. Pronounced "store".

!CSP
Save the stack position in CSP. Used as part of the compiler security.

d1 – – – d2 LO
Generate from a double number d1, the next ascii character which is placed in an output string. Result d2 is the quotient after division by BASE, and is maintained for further processing. Used between and . See # S.

d – – – addr count LO
Terminates numeric output conversion by dropping d, leaving the text address and character count suitable for TYPE.

 # S d1 – – – d2 LO
Generates ascii text in the text output buffer, by the use of #, until a zero double number n2 results. Used between #and # .

' ' – – – addr P,LO
Used in the form:
' nnnn
Leaves the parameter field address of dictionary word nnnn. As a compiler directive, executes in a colon-definition to compile the address as a literal. If the word is not found after a search of CONTEXT and CURRENT, an appropriate error message is given. Pronounced "tick".

(P,LO

Used in the form:

(cccc)

Ignore a comment that will be delimited by a right parenthesis on the same line. May occur during execution or in a colon-definition. a blank after the leading parenthesis is required.

(.'') C+

The run-time procedure, compiled by .'' which transmits the following in-line text to the selected output device. See .''

(;CODE) C

The run-time procedure, compiled by ;CODE, that rewrites the code field of the most recently defined word to point to the following machine code sequence. See ;CODE.

(+LOOP) n--- C2

The run-time procedure compiled by +LOOP, which increments the loop index by n and tests for loop completion. See +LOOP

(ABORT)

Executes after an error when WARNING is –1. This word normally executes ABORT, but may be altered (with care) to a user's alternative procedure.

(DO) C

The run-time procedure compiled by DO which moves the loop control parameters to the return stack. See DO.

(FIND) addr1 addr2 --- pfa b tf (ok)
addr1 addr2 --- ff (bad)

Searches the dictionary starting at the name field address addr2, matching to the text at addr1. Returns parameter field address, length byte of name field and boolean true for a good match. If no match is found, only a boolean false is left.

(LINE) n1 n2 --- addr count

Convert the line number n1 and the screen n2 to the disc buffer address containing the data. A count of 64 indicates the full line text length.

(LOOP) C2

The run-time procedure compiled by LOOP which increments the loop index and tests for loop completion. See LOOP.

(NUMBER) d1 addr1 – – – d2 addr2
Convert the ascii text beginning at addr1 +1 with regard to BASE. The new value is accumulated into double number d1, being left as d2. Addr2 is the address of the first unconvertable digit. Used by NUMBER.

* n1 n2 – – – prod LO
Leave the signed product of two signed numbers.

* n1 n2 n3 – – – n4 LO
Leave the ratio n4 = n1*n2/n3 where all are are signed numbers. Retention of an intermediate 31 bit product permits greater accuracy than would be available with the sequence:

n1 n2 * n3 /

*/MOD n1 n2 n3 – – – n4 n5 LO
Leave the quotient n5 and remainder n4 of the operation n1*n2/n3 A 31 bit intermediate product is used as for */.

\+ n1 n2 – – – sum LO
Leave the sum of n1 +n2.

+ ! n addr – – – LO
Add n to the value at the address. Pronounced "plus-store".

+ – n1 n2 – – – n3
Apply the sign of n2 to n1, which is left as n3.

+BUF addr1 – – – addr2 f
Advance the disc buffer address addr1 to the address of the next buffer addr2. Boolean f is false when addr2 is the buffer presently pointed to by variable PREV.

+LOOP n1 – – – (run)
addr n2 – – – (compile) P,C2,LO
Used in a colon-definition in the form:

DO . . . n1 +LOOP

At run-time, +LOOP selectively controls branching back to the corresponding DO based on n1, the loop index and the loop limit. The signed increment n1 is added to the index and the total compared to the limit. The branch back to DO occurs until the new index is equal to or greater than the limit (n1 >0), or until the new index is equal to or less than the limit (n1 < o). Upon exiting the loop, the parameters are discarded and execution continues ahead.

At compile time, +LOOP compiles the run-time word (+LOOP) and the branch offset computed from HERE to the address left on the stack by DO. n2 is used for compile time error checking.

+ORIGIN n – – – addr

Leave the memory address relative by n to the origin parameter area. n is the minimum address unit, either byte or word. This definition is used to access or modify the boot-up parameters at the origin area.

n – – – LO

Store n into the next available dictionary memory cell, advancing the dictionary pointer. (comma)

– / – n1 n2 – – – diff LO

Leave the difference of n1-n2.

– – → P,LO

Continue interpretation with the next disc screen. (Pronounced next-screen).

–DUP n1 – – – n1 (if zero) LO
n1 – – – n1 n1 (non-zero)

Reproduce n1 only if it is non-zero. This is usually used to copy a value just before IF, to eliminate the need for an ELSE part to drop it.

–FIND – – – pfa b tf (found)
– – – ff (not found)

Accepts the next text word (delimited by blanks) in the input stream to HERE, and searches the CONTEXT and then CURRENT vocabularies for a matching entry. If found, the dictionary entry's parameter field address, its length byte, and a boolean true is left. Otherwise, only a boolean false is left.

–TRAILING addr n1 – – – addr n2

Adjusts the character count n1 of a text string beginning address to suppress the output of trailing blanks. i.e. the characters at addr +n1 to addr +n2 are blanks.

n – – – LO

Print a number from a signed 16 bit two's complement value, converted according to the numeric BASE. A trailing blank follows. Pronounced "dot".

.'' P,LO

Used in the form:

'' cccc ''

Compiles an in-line string cccc (delimited by the trailing '') with an execution procedure to transmit the text to the selected output device. If executed outside a definition, .'' will immediately print the text until the final ''. The maximum number of characters may be an installation dependent value. See (.'').

.LINE line scr – – –
Print on the terminal device, a line of text from the disc by its line and screen number. Trailing blanks are suppressed.

.R n1 n2 – – –
Print the number n1 right aligned in a field whose width is n2. No following blank is printed.

/ n1 n2 – – – quot LO
Leave the signed quotient of n1/n2.

/MOD n1 n2 – – – rem quot LO
Leave the remainder and signed quotent of n1/n2. The remainder has the sign of the dividend.

0 1 2 3 – – – n
These small numbers are used so often that it is attractive to define them by name in the dictionary as constants.

0< n – – – f LO
Leave a true flag if the number is less than zero (negative), otherwise leave a false flag.

0= n – – – f LO
Leave a true flag if the number is equal to zero, otherwise leave a false flag.

0BRANCH f – – – C2
The run-time procedure is conditionally branch. If f is false (zero), the following in-line parameter is added to the interpretive pointer to branch ahead or back. Compiled by IF, UNTIL, and WHILE.

1+ n1 – – – n2 L1
Increment n1 by 1.

2+ n1 – – – n2
Leave n1 incremented by 2.

P,E,LO

Used in the form called a colon-definition:

: cccc ... ;

Creates a dictionary entry defining cccc as equivalent to the following sequence of Forth word definitions '...' until the next ';' or ';CODE'. The compiling process is done by the text interpreter as long as STATE is non-zero. Other details are that the CONTEXT vocabulary is set to the CURRENT vocabulary and that words with the precedence bit set (P) are executed rather than being compiled.

; P,C,LO

Terminate a colon-definition and stop further compilation. Compiles the run-time;S.

;CODE F,C,LO

Used in the form:

: cccc .... ;CODE
assembly mnemonics

Stop compilation and terminate a new defining word cccc by compiling (;CODE). Set the CONTEXT vocabulary to ASSEMBER, assembling to machine code the following mnemonics.

When cccc later executes in the form:

cccc nnnn

the word nnnn will be created with its execution procedure given by the machine code following cccc. That is, when nnnn is executed, it does so by jumping to the code after nnnn. An existing defining word must exist in cccc prior to ;CODE.

;S P,LO

Stop interpretation of a screen. ;S is also the run-time word compiled at the end of a colon-definition which returns execution to the calling procedure.

< n1 n2 – – – f LO

Leave a true flag if n1 is less than n2; otherwise leave a false flag.

<# LO

Setup for pictured numeric output formatting using the words:

<# # #S SIGN #>

The conversion is done on a double number producing text at PAD.

<BUILDS C,LO

Used within a colon-definition:

```
: cccc <BUILDS ...
     DOES> ... ;
```

Each time cccc is executed, <BUILDS defines a new word with a high-level execution procedure. Executing cccc in the form:

```
cccc nnnn
```

uses <BUILDS to create a dictionary entry for nnnn with a call to the DOES> part for nnnn. When nnnn is later executed, it has the address of its parameter area on the stack and executes the words after DOES> in cccc. <BUILDS and DOES> allow run-time procedures to be written in high-level rather than in assembler code (as required by ;CODE).

= n1 n2 – – – f LO

Leave a true flag if n1 = n2; otherwise leave a false flag.

> n1 n2 – – – f LO

Leave a true flag if n1 is greater than n2; otherwise a false flag.

>R n – – – C,LO

Remove a number from the computation stack and place as the most accessible on the return stack. Use should be balanced with R> in the same definition.

? addr – – – LO

Print the value contained at the address in free format according to the current base.

?COMP

Issue error message if not compiling.

?CSP

Issue error message if stack position differs from value saved in CSP.

?ERROR f n – – –

Issue an error message number n, if the boolean flag is true.

?EXEC

Issue an error message if not executing.

?LOADING

Issue an error message if not loading.

? PAIRS n1 n2 – – –
Issue an error message if n1 does not equal n2. The message indicates that compiled conditionals do not match.

? STACK
Issue an error message if the stack is out of bounds. This definition may be installation dependent.

? TERMINAL – – – f
Perform a test of the terminal keyboard for actuation of the break key. A true flag indicates actuation. This definition is installation dependent.

@ addr – – – n LO
Leave the 16 bit contents of address.

ABORT LO
Clear the stacks and enter the execution state. Return control to the operators terminal, printing a message appropriate to the installation.

ABS n – – – u LO
Leave the absolute value of n as u.

AGAIN addr n – – – (compiling) P,C2,LO
Used in a colon-definition in the form:
BEGIN . . . AGAIN
At run-time, AGAIN forces execution to return to corresponding BEGIN. There is no effect on the stack. Execution cannot leave this loop (unless R > DROP is executed one level below).

At compile time, AGAIN compiles BRANCH with an offset from HERE to addr. n is used for compile-time error checking.

ALLOT n – – – LO
Add the signed number to the dictionary pointer DP. May be used to reserve dictionary space or re-origin memory. n is with regard to computer address type (byte or word).

AND n1 n2 – – – n2 LO
Leave the bitwise logical and of n1 and n2 as n3.

B/BUF – – – n
This constant leaves the number of bytes per disc buffer, the byte count read from disc by BLOCK.

B/SCR ---n
This constant leaves the number of blocks per editing screen. By convention, an editing screen is 1024 bytes organized as 16 lines of 64 characters each.

BACK addr---
Calculate the backward branch offset from HERE to addr and compile into the next available dictionary memory address.

BASE ---addr U,LO
A user variable containing the current number base used for input and output conversion.

BEGIN ---addr n (compiling) P,LO
Occurs in a colon-definition in form:
BEGIN . . . UNTIL
BEGIN . . . AGAIN
BEGIN . . . WHILE . . . REPEAT
At run-time, BEGIN marks the start of a sequence that may be repetitively executed. It serves as a return point from the corresponding UNTIL, AGAIN or REPEAT. When executing UNTIL, a return to BEGIN will occur if the top of the stack is false; for AGAIN and REPEAT a return to BEGIN always occurs.

At compile time BEGIN leaves its return address and n for compiler error checking.

BL ---c
A constant that leaves the ascii value for "blank".

BLANKS addr count---
Fill an area of memory beginning at addr with blanks.

BLK ---addr U,LO
A user variable containing the block number being interpreted. If zero, input is being taken from the terminal input buffer.

BLOCK N---,addr LO
Leave the memory address of the block buffer containing block n. If the block is not already in memory, it is transferred from disc to which ever buffer was least recently written. If the block occupying that buffer has been marked as updated, it is rewritten to disc before block n is read into the buffer. See also BUFFER, R/W UPDATE FLUSH

BLOCK–READ
BLOCK–WRITE — These are the preferred names for the installation dependent code to read and write one block to the disc.

BRANCH — C2,LO
The run-time procedure to unconditionally branch. An in-line offset is added to the interpretive pointer IP to branch ahead or back. BRANCH is compiled by ELSE, AGAIN, REPEAT.

BUFFER — n – – – addr
Obtain the next memory buffer, assigning it to block n. If the contents of the buffer is marked as updated, it is written to the disc. The block is not read from the disc. The address left is the first cell within the buffer for data storage.

C! — b addr – – –
Store 8 bits at address. On word addressing computers, further specification is necessary regarding byte addressing.

C, — b – – –
Store 8 bits of b into the next available dictionary byte, advancing the dictionary pointer. This is only available on byte addressing computers, and should be used with caution on byte addressing mini-computer.

C@ — addr – – – b
Leave the 8 bit contents of memory address. On word addressing computers, further specification is needed regarding byte addressing.

CFA — pfa – – – cfa
Convert the parameter field address of a definition to its code field address.

CMOVE — from to count – – –
Move the specified quantity of bytes beginning at address from to address to. The contents of address from is moved first proceeding toward high memory. Further specification is necessary on word addressing computers.

COLD
The cold start procedure to adjust the dictionary pointer to the minimum standard and restart via ABORT. May be called from the terminal to remove application programs and restart.

COMPILE C2

When the word containing COMPILE executes, the execution address of the word following COMPILE is copied (compiled) into the dictionary. This allows specific compilation situations to be handled in addition to simply compiling an execution address (which the interpreter already does).

CONSTANT n – – – LO

A defining word used in the form:

n CONSTANT cccc

to create word cccc, with its parameter field containing n. When cccc is later executed, it will push the value of n to the stack.

CONTEXT – – – addr U,LO

A user variable containing a pointer to the vocabulary within which dictionary searches will first begin.

COUNT addr1 – – – addr2 n LO

Leave the byte address addr2 and byte count n of a message text beginning at address addr1. It is presumed that the first byte at addr1 contains the text byte count and the actual text starts with the second byte. Typically COUNT is followed by TYPE.

CR LO

Transmit a carriage return and line feed to the selected output device.

CREATE

A defining word used in the form:

CREATE cccc

by such words as CODE and CONSTANT to create a dictionary header for a Forth definition. The code field contains the address of the words parameter field. The new word is created in the CURRENT vocabulary.

CSP – – – addr U

A user variable temporarily storing the stack pointer position, for compilation error checking.

D+ d1 d2 – – – dsum

Leave the double number sum of two double numbers.

D+– d1 n – – – d2

Apply the sign of n to the double number d1, leaving it as d2.

D. d – – – L1
Print a signed double number from a 32 bit two's complement value. The high-order 16 bits are most accessible on the stack. Conversion is performed according to the current BASE. A blank follows. Pronounced D-dot.

D.R d n – – –
Print a signed double number d right aligned in a field n characters wide.

DABS d – – – ud
Leave the absolute value ud of a double number

DECIMAL LO
Set the numeric conversion BASE for decimal input-output.

DEFINITIONS L1
Used in the form:
cccc DEFINITIONS
Set the CURRENT vocabulary to the CONTEXT vocabulary. In the example, executing vocabulary name cccc made if the CONTEXT vocabulary and executing DEFINITIONS made both specify vocabulary cccc.

DIGIT c n1 – – – n2 tf (ok)
c n1 – – – ff (bad)
Converts the ascii character c (using base n1) to its binary equivalent n2, accompanied by a true flag. If the conversion is invalid, leaves only a false flag.

DLIST
List the names of the dictionary entries in the CONTEXT vocabulary.

DLITERAL d – – – d (executing)
d – – – (compiling) P
If compiling, compile a stack double number into a literal. Later execution of the definition containing the literal will push it to the stack. If executing, the number will remain on the stack.

DMINUS d1 – – – d2
Convert d1 to its double number two's complement.

DO n1 n2 – – – (execute)
addr n – – – (compile) P,C2,LO
Occurs in a colon-definition in form:

DO ... LOOP
DO ... +LOOP

At run time, DO begins a sequence with repetitive execution controlled by a loop limit n1 and an index with initial value n2. DO removes these from the stack. Upon reaching LOOP the index is incremented by one. Until the new index equals or exceeds the limit, execution loops back to just after DO; otherwise the loop parameters are discarded and execution continues ahead. Both n1 and n2 are determined at run-time and may be the result of other operations. Within a loop 'I' will copy the current value of the index to the stack. See I, LOOP, +LOOP, LEAVE.

When compiling within the colon-definition, DO compiles (DO), leaves the following address addr and n for later error checking.

DOES> LO

A word which defines the run-time action within a high-level defining word. DOES> alters the code field and first parameter of the new word to execute the sequence of compiled word addresses following DOES>. Used in combination with <BUILDS. When the DOES> part executes it begins with the address of the first parameter of the new word on the stack. This allows interpretation using this area or its contents. Typical uses include the Forth assembler, multi-dimensional arrays, and compiler generation.

DP ---addr U,L

A user variable, the dictionary pointer, which contains the address of the next free memory above the dictionary. The value may be read by HERE and altered by ALLOT.

DPL ---addr U,LO

A user variable containing the number of digits to the right of the decimal on double integer input. It may also be used hold output column location of a decimal point, in user generated formating. The default value on single number input is -1.

DR0
DR1

Installation dependent commands to select disc drives, by presetting OFFSET. The contents of OFFSET is added to the block number in BLOCK to allow for this

selection. Offset is suppressed for error text so that it may always originate from drive 0.

DROP n – – – LO

Drop the number from the stack.

DUMP addr n – – – LO

Print the contents of n memory locations beginning at addr. Both addresses and contents are shown in the current numeric base.

DUP n – – – n n LO

Duplicate the value on the stack.

ELSE addr1 n1 – – – addr2 n2 (compiling) P,C2,LO

Occurs within a colon-definition in the form:

IF . . . ELSE . . . ENDIF

At run-time, ELSE executes after the true part following IF. ELSE forces execution to skip over the following false part and resumes execution after the ENDIF. It has no stack effect.

At compile-time ELSE emplaces BRANCH reserving a branch offset, leaves the address addr2 and n2 for error testing. ELSE also resolves the pending forward branch from IF by calculating the offset from addr1 to HERE and storing at addr1.

EMIT c – – – LO

Transmit ascii character c to the selected output device. OUT is incremented for each character output.

EMPTY–BUFFERS LO

Mark all block-buffers as empty, not necessarily affecting the contents. Updated blocks are not written to the disc. This is also an initialization procedure before first use of the disc.

ENCLOSE addr1 c – – –

addr1 n1 n2 n3

The text scanning primitive used by WORD. From the text address addr1 and an ascii delimiting character c, is determined the byte offset to the first non-delimiter character n1, the offset to the first delimiter after the text n2, and the offset to the first character not included. This procedure will not process past an ascii 'null', treating it as an unconditional delimiter.

END P,C2,LO

This is an 'alias' or duplicate definition for UNTIL.

ENDIF addr n – – – (compile) P,CO,LO

Occurs in a colon-definition in form:

IF ... ENDIF

IF ... ELSE ... ENDIF

At run-time, ENDIF serves only as the destination of a forward branch from IF or ELSE. It marks the conclusion of the conditional structure. THEN is another name for ENDIF. Both names are supported in fig-FORTH. See also IF and ELSE.

At compile-time, ENDIF computes the forward branch offset from addr to HERE and stores it at addr. n is used for error tests.

ERASE addr n – – –

Clear a region of memory to zero from addr over n addresses.

ERROR line – – – in blk

Execute error notification and restart of system. WARNING is first examined. If 1, the text of line n, relative to screen 4 of drive 0 is printed. This line number may be positive or negative, and beyond just screen 4. If WARNING = 0, n is just printed as a message number (non disc installation). If WARNING is –1, the definition (ABORT) is executed, which executes the system ABORT. The user may cautiously modify this execution by altering (ABORT). fig-FORTH saves the contents of IN and BLK to assist in determining the location of the error. Final action is execution of QUIT.

EXECUTE addr – – –

Execute the definition whose code field address is on the stack. The code field address is also called the compilation address.

EXPECT addr count – – – LO

Transfer characters from the terminal to address, until a "return" or the count of characters have been received. One or more nulls are added at the end of the text.

FENCE – – – addr U

A user variable containing an address below which FORGETting is trapped. To forget below this point the user must alter the contents of FENCE.

FILL ADDR QUAN B – – –
Fill memory at the address with the specified quantity of bytes b.

FIRST – – – n
A constant that leaves the address of the first (lowest) block buffer.

FLD – – – addr U
A user variable for control of number output field width. Presently unused in fig-FORTH.

FORGET E,LO
Executed in the form:
FORGET cccc
Deletes definition named cccc from the dictionary with all entries physically following it. In fig-FORTH, an error message will occur if the CURRENT and CONTEXT vocabularies are not currently the same.

FORTH P,L1
The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. Until additional user vocabularies are defined, new user definitions become a part of FORTH. FORTH is immediate, so it will execute during the creation of a colon-definition, to select this vocabulary at compile time.

HERE – – – addr LO
Leave the address of the next available dictionary location.

HEX LO
Set the numeric conversion base to sixteen (hexa-decimal).

HLD – – – addr LO
A user variable that holds the address of the latest character of text during numeric output conversion.

HOLD c – – – LO
Used between < # and # > to insert an ascii character into a pictured numeric output string.
e.g. 2E HOLD will place a decimal point.

I – – – n C,LO
Used within a DO-LOOP to copy the loop index to the stack. Other use is implementation dependent. See R.

ID. addr – – –
Print a definition's name from its name field address.

IF f – – – (run-time)
– – – addr n (compile) P,C2,LO
Occurs if a colon-definition in form:
IF (tp) ... ENDIF
IF (tp) ... ELSE (fp) ... ENDIF
At run-time, IF selects execution based on a boolean flag. If f is true (non-zero), execution continues ahead thru the true part. If f is false (zero), execution skips till just after ELSE to execute the false part. After either part, execution resumes after ENDIF. ELSE and its false part are optional; if missing, false execution skips to just after ENDIF.

At compile-time IF compiles 0BRANCH and reserves space for an offset at addr. addr and n are used later for resolution of the offset and error testing.

IMMEDIATE

Mark the most recently made definition so that when encountered at compile time, it will be executed rather than being compiled. i.e. the precedence bit in its header is set. This method allows definitions to handle unusual compiling situations, rather than build them into the fundamental compiler. The user may force compilation of an immediate definition by preceeding it with [COMPILE].

IN – – – addr LO
A user variable containing the byte offset within the current input text buffer (terminal or disc) from which the next text will be accepted. WORD uses and moves the value of IN.

INDEX FROM TO – – –
Print the first line of each screen over the range from, to. This is used to view the comment lines of an area of text on disc screens.

INTERPRET

The outer text interpreter which sequentially executes or compiles text from the input stream (terminal or disc) depending on STATE. If the word name cannot be found after a search of CONTEXT and then CURRENT it is converted to a number according to the current base.

That also failing, an error message echoing the name with a ''?'' will be given. Text input will be taken according to the convention for WORD. If a decimal point is found as part of a number, a double number value will be left. The decimal point has no other purpose than to force this action. See NUMBER.

KEY ---c LO
Leave the ascii value of the next terminal key struck.

LATEST ---addr
Leave the name field address of the topmost word in the CURRENT vocabulary.

LEAVE C,LO
Force termination of a DO-LOOP at the next opportunity by setting the loop limit equal to the current value of the index. The index itself remains unchanged, and execution proceeds normally until LOOP or +LOOP is encountered.

LFA pfa---Lfa
Convert the parameter field address of a dictionary definition to its link field address.

LIMIT ---n
A constant leaving the address just above the highest memory available for a disc buffer. Usually this is the highest system memory.

LIST n--- LO
Display the ascii text of screen n on the selected output device. SCR contain the screen number during and after this process.

LIT ---n C2,LO
Within a colon-definition, LIT is automatically compiled before each 16 bit literal number encountered in input text. Later execution of LIT causes the contents of the next dictionary address to be pushed to the stack.

LITERAL n--- (compiling) P,C2,LO
If compiling, then compile the stack value n as a 16 bit literal. This definition is immediate so that it will execute during a colon definition. The intended use is:

: xxx [calculate] LITERAL ;

Compilation is suspended for the compile time calculation of a value, Compilation is resumed and LITERAL compiles this value.

LOAD n – – – LO
Begin interpretation of screen n. Loading will terminate at the end of the screen or at ;S. See ;S and —→.

LOOP addr n – – – (compiling) P,C2,LO
Occurs in a colon-definition in form:
DO ... LOOP
At run-time, LOOP selectively controls branching back to the corresponding DO based on the loop index and limit. The loop index is incremented by one and compared to the limit. The branch back to DO occurs until the index equals or exceeds the limit; at that time, the parameters are discarded and execution continues ahead.

At compile-time, LOOP compiles (LOOP) and uses addr to calculate an offset to DO. Zn is used for error testing.

M* n1 n2 – – – d
A mixed magnitude math operation which leaves the double number signed product of two signed number.

M/ d n1 – – – n2 n3
A mixed magnitude math operator which leaves the signed remainder n2 and signed quotient n3, from a double number dividend and divisor n1. The remainder takes its sign from the dividend.

M/MOD ud1 u2 – – – u3 ud4
An unsigned mixed magnitude math operation which leaves a double quotient ud4 and remainder u3, from a double dividend ud1 and single divisor u2.

MAX n1 n2 – – – max LO
Leave the greater of two numbers.

MESSAGE n – – –
Print on the selected output device the text of line n relative to screen 4 of drive 0. n may be positive or negative. MESSAGE may be used to print incidental text such as report headers. If WARNING is zero, the message will simply be printed as a number (disc unavailable).

MIN n1 n2 – – – min LO
Leave the smaller of two numbers.

MINUS n1 – – –n2 LO
Leave the two's complement of a number.

MOD n1 n2 – – – mod LO
Leave the remainder of n1/n2, with the same sign as n1.

MON
Exit to the system monitor, leaving a re-entry to Forth, if possible.

MOVE addr1 addr2 n – – –
Move the contents of n memory cells (16 bit contents) beginning at addr1 into n cells beginning at addr2. The contents of addr1 is moved first. This definition is appropriate on word addressing computers.

NEXT
This is the inner interpreter that uses the interpretive pointer IP to execute compiled Forth definitions. It is not directly executed but is the return point for all code procedures. It acts by fetching the address pointed by IP, storing this value in register W. It then jumps to the address pointed to by the address pointed to by W.W points to the code field of a definition which contains the address of the code which executes for that definition. This usage of indirect threaded code is a major contributor to the power, portability, and extensibility of Forth. Locations of IP and W are computer specific.

NFA pfa – – – nfa
Convert the parameter field address of a definition to its name field.

NUMBER addr – – – d
Convert a character string left at addr with a preceding count, to a signed double number, using the current numeric base. If a decimal point is encountered in the text, its position will be given in DPL, but no other effect occurs. If numeric conversion is not possible, an error message will be given.

OFFSET – – – addr U
A user variable which may contain a block offset to disc drives. The contents of OFFSET is added to the stack number by BLOCK. Messages by MESSAGE are independent of OFFSET. See BLOCK, DR0, DR1, MESSAGE.

OR n1 n2 – – – or LO
Leave the bit-wise logical or of two 16 bit values.

OUT ---addr U
A user variable that contains a value increment by EMIT. The user may alter and examine OUT to control display formating.

OVER n1 n2---n1 n2 n1 LO
Copy the second stack value, placing it as the new top.

PAD ---addr LO
Leave the address of the text output buffer, which is a fixed offset above HERE.

PFA nfa---pfa
Convert the name field address of a compiled definition to its parameter field address.

POP
The code sequence to remove a stack value and return to NEXT. POP is not directly executable, but is a Forth re-entry point after machine code.

PREV ---addr
A variable containing the address of the disc buffer most recently referenced. The UPDATE command marks this buffer to be later written to disc.

PUSH
This code sequence pushes machine registers to the computation stack and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

PUT
This code sequence stores machine register contents over the topmost computation stack value and returns to NEXT. It is not directly executable, but is a Forth re-entry point after machine code.

QUERY
Input 80 characters of text (or until a "return") from the operators terminal. Text is positioned at the address contained in TIB with IN set to zero.

QUIT L1
Clear the return stack, stop compilation, and return control to the operators terminal. No message is given.

R ---n
Copy the top of the return stack to the computation stack.

R# ---addr U
A user variable which may contain the location of an editing cursor, or other file related function.

R/W addr blk f---
The fig-FORTH standard disc read-write linkage. addr specifies the source or destination block buffer, blk is the sequential number of the referenced block; and f is a flag for f=0 write and f=1 read. R/W determines the location on mass storage, performs the read-write and performs any error checking.

R> ---n LO
Remove the top value from the return stack and leave it on the computation stack. See >R and R.

RO ---addr U
A user variable containing the initial location of the return stack. Pronounced R-zero. See RP!.

REPEAT addr n---(compiling) P,C2
Used within a colon-definition in the form:
BEGIN ... WHILE ... REPEAT
At run-time, REPEAT forces an unconditional branch back to just after the corresponding BEGIN.

At compile-time, REPEAT compiles BRANCH and the offset from HERE to addr. n is used for error testing.

ROT n1 n2 n3---n2 n3 n1 LO
Rotate the top three values on the stack, bringing the third to the top.

RP! A computer dependent procedure to initialize the return stack pointer from user variable R0.

S->D n---d
Sign extend a single number to form a double number.

SO ---addr U
A user variable that contains the initial value for the stack pointer. Pronounced S-zero. See SP!

SCR ---addr U
A user variable containing the screen number most recently reference by LIST.

SIGN n d---d LO
Stores an ascii "-" sign just before a converted numeric output sting in the text output buffer when n is negative.

n is discarded, but double number d is maintained. Must be used between <# and #>.

SMUDGE

Used during word definition to toggle the "smudge bit" in a definitions name field. This prevents an uncompleted definition from being found during dictionary searches, until compiling is completed without error.

SP!

A computer dependent procedure to initialize the stack pointer from S0.

SP@ ---addr

A computer dependent procedure to return the address of the stack position to the top of the stack, as it was before SP@ was executed. (e.g. 1 2 SP@ . . . would type 2 2 1)

SPACE LO

Transmit an ascii blank to the output device.

SPACES n--- LO

Transmit n ascii blanks to the output device.

STATE ---addr LO,U

A user variable containing the compilation state. A non-zero value indicates compilation. The value itself may be implementation dependent.

SWAP n1 n2---n2 n1 LO

Exchange the top two values on the stack.

TASK

A no-operation word which can mark the boundary between applications. By forgetting TASK and recompiling, an application can be discarded in its entirety.

THEN P,CO,LO

An alias for ENDIF.

TIB ---addr U

A user variable containing the address of the terminal input buffer.

TOGGLE addr b---

Complement the contents of addr by the bit pattern b.

TRAVERSE addr1 n---addr2

Move across the name field of a fig-FORTH variable

length name field. addr1 is the address of either the length byte or the last letter. If n=1, the motion is toward hi memory; if n=–1, the motion is toward low memory. The addr2 resulting is address of the other end of the name.

TRIAD

scr – – –
Display on the selected output device the three screens which include that numbered scr, beginning with a screen evenly divisible by three. Output is suitable for source text records, and includes a reference line at the bottom taken from line 15 of screen 4.

TYPE

addr count – – – LO
Transmit count characters from addr to the selected output device.

U*

u1 u2 – – – ud
Leave the unsigned double number product of two unsigned numbers.

U/

ud u1 – – – u2 u3
Leave the unsigned remainder u2 and unsigned quotient u3 from the unsigned double dividend ud and unsigned divisor u1.

UNTIL

f – – – (run-time)
addr n – – – (compile) P,C2,LO
Occurs within a colon-definition in the form:

BEGIN ... UNTIL

At run-time, UNTIL controls the conditional branch back to the corresponding BEGIN. If f is false, execution returns to just after BEGIN; if true, execution continues ahead.

At compile-time, UNTIL compiles (0BRANCH) and an offset from HERE to addr. n is used for error tests.

UPDATE LO

Marks the most recently referenced block (pointed to by PREV) as altered. The block will subsequently be transferred automatically to disc should its buffer be required for storage of a different block.

USE

– – – addr
A variable containing the address of the block buffer to use next, as the least recently written.

USER n--- LO

A defining word used in the form:

n USER cccc

which creates a user variable cccc. The parameter field of cccc contains n as a fixed offset relative to the user pointer register UP for this user variable. When cccc is later executed, it places the sum of its offset and the user area base address on the stack as the storage address of that particular variable.

VARIABLE E,LO

A defining word used in the form:

n VARIABLE cccc

When VARIABLE is executed, it creates the definition cccc with its parameter field initialized to n. When cccc is later executed, the address of its parameter field (containing n) is left on the stack, so that a fetch or store may access this location.

VOC–LINK ---addr U

A user variable containing the address of a field in the definition of the most recently created vocabulary. All vocabulary names are linked by these fields to allow control for FORGETing thru multiple vocabularys.

VOCABULARY E,L

A defining word used in the form:

VOCABULARY cccc

to create a vocabulary definition cccc. Subsequent use of cccc will make it the CONTEXT vocabulary which is searched first by INTERPRET. The sequence "cccc DEFINITIONS" will also make cccc the CURRENT vocabulary into which new definitions are placed.

In fig-FORTH, cccc will be so chained as to include all definitions of the vocabulary in which cccc is itself defined. All vocabularys ultimately chain to Forth. By convention, vocabulary names are to be declared IMMEDIATE. See VOC-LINK.

VLIST

List the names of the definitions in the context vocabulary. "Break" will terminate the listing.

WARNING ---addr U

A user variable containing a value controlling messages. If = 1 disc is present, and screen 4 of drive 0 is the base location for messages. If = 0, no disc is present

and messages will be presented by number. If = –1, execute (ABORT) for a user specified procedure. See MESSAGE, ERROR.

WHILE f – – – (run-time)
ad1 n1 – – – ad1 n1 ad2 n2 P,C2

Occurs in a colon-definition in the form:

BEGIN ... WHILE (tp) ... REPEAT

At run-time, WHILE selects conditional execution based on boolean flag f. If f is true (non-zero), WHILE continues execution of the true part thru to REPEAT, which then branches back to BEGIN. If if is false (zero), execution skips to just after REPEAT, exiting the structure.

At compile time, WHILE emplaces (0BRANCH) and leaves ad2 of the reserved offset. The stack values will be resolved by REPEAT.

ADDENDA to Glossary for 8080 fig-FORTH v1.1

.CPU	Prints the processor name.
2!	nlow nhigh addr ---- 32 bit store. nhigh is stored at addr; nlow is stored at addr+2;
2@	addr --- nlow nhigh 32 bit fetch. nhigh is fetched from addr; nlow is fetched from addr+2.
2DUP	n2 n1 --- n2 n1 n2 n1 Duplicates the top two values on the stack. Equivalent to OVER OVER
C/L	--- n Constant leaving the number of character per line;used by the editor.
NOOP	A FORTH "no operation'
P!	b port# 8080 or Z80 I/O port store. Outputs byte b to port#.
P@	port# --- b 8080 or Z80 I/O port fetch. Inputs byte b from port#.
RP@	--- addr Leaves the current value in the return stack pointer register.
U<	u1 u2 --- f Leave the boolean value of an unsigned less-than comparison. Leaves f=1 for u1 < u2;otherwise leaves 0. This function must be used when comparing memory addresses. u1 and u2 are unsigned 16 bit integers.

```
3 LIST
SCR # 3
  0 ( FORTH HEADER ) DECIMAL :S
  1 SHARPSOFT MZ-80K VERSION OF
  2 8080 FIG-FORTH 1.1
  3 FIG-FORTH MODEL THROUGH THE
  4 COURTESY OF THE
  5 FORTH INTEREST GROUP,
  6 P.O. BOX 1105
  7 SAN CARLOS, CA 94070
  8 RELEASE 1
  9 WITH COMPILER SECURITY
 10          AND
 11 VARIABLE LENGTH NAMES.
 12 ADDITIONAL MATERIAL WRITTEN
 13 BY MIKE BRINSON AND
 14 ALAN GREY ---- SHARPSOFT.
 15 SHARPSOFT RELEASE NO. 1.2
OK
```

```
4 LIST
SCR # 4
  0 ( ERROR MESSAGES )
  1 EMPTY STACK
  2 DICTIONARY FULL
  3 HAS INCORRECT ADDRESS MODE
  4 ISN'T UNIQUE
  5
  6 TAPE RANGE ?
  7 FULL STACK
  8 TAPE ERROR !
  9
 10
 11
 12
 13
 14 SHARPSOFT RELEASE 1.2
 15 FORTH INTEREST GROUP
OK
```

```
5 LIST
SCR # 5
  0 ( ERROR MESSAGES )
  1 COMPILATION ONLY, USE IN DEFINITION
  2 EXECUTION ONLY
  3 CONDITIONALS NOT PAIRED
  4 DEFINITION NOT FINISHED
  5 IN PROTECTED DICTIONARY
  6 USE ONLY WHEN LOADING
  7 OFF CURRENT EDITING SCREEN
  8 DECLARE VOCABULARY
  9
 10
 11
 12
 13
 14
 15 SHARPSOFT RELEASE 1.2
OK
```

```
6 LIST
SCR # 6
  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
OK
```

```
7 LIST
SCR # 7
  0 ( PPS - LINE EDITOR ED SCR1 )
  1 FORTH DEFINITIONS HEX
  2 : TEXT    HERE C/L 1+ BLANKS WORD
  3           HERE PAD C/L 1+ CMOVE ;
  4 : LINE    DUP FFF0 AND 17 ?ERROR
  5           SCR @ (LINE) DROP ;
  6 VOCABULARY EDITOR IMMEDIATE HEX
  7 : WHERE   DUP B/SCR / DUP SCR !
  8           ." SCR # " DECIMAL . SWAP
  9           C/L /MOD C/L * ROT BLOCK
 10         + CR C/L TYPE CR HERE C@
 11         - SPACES 5E EMIT [COMPILE]
 12           EDITOR QUIT ;
 13 EDITOR DEFINITIONS
 14 : #LOCATE R# @ C/L /MOD ;
 15 : #LEAD #LOCATE LINE SWAP ; -->
OK
```

```
8 LIST
SCR # 8
  0 ( PPS - LINE EDITOR ED SCR2 )
  1 : #LAG    #LEAD DUP >R + C/L R> - ;
  2 : -MOVE   LINE C/L CMOVE UPDATE ;
  3 : H       LINE PAD 1+ C/L DUP PAD
  4           C! CMOVE ;
  5 : E       LINE C/L BLANKS UPDATE ;
  6 : S       DUP 1 - ( LIMIT ) 0E
  7           ( FIRST TO MOVE )
  8           DO I LINE I 1+ -MOVE
  9           -1 +LOOP E ;
 10 : D       DUP H 0F DUP ROT
 11          DO I 1+ LINE I -MOVE
 12          LOOP E ;
 13 -->
 14
 15
OK
```

```
9 LIST
SCR # 9
  0 ( PPS - LINE EDITOR ED SCR4 )
  1 : M      R# +! CR SPACE #LEAD TYPE
  2          5F EMIT #LAG TYPE #LOCATE
  3          . DROP ;
  4 : T      DUP C/L * R# ! DUP H 0 M ;
  5 : L      SCR @ LIST 0 M ;
  6 : R      PAD 1+ SWAP -MOVE ;
  7 : P      1 TEXT R ;
  8 : I      DUP S R ;
  9 : TOP    0 R# ! ;
 10 : CLEAR  SCR ! 10 0 DO FORTH I
 11          EDITOR E LOOP ;
 12
 13
 14
 15 -->
OK
```

```
10 LIST
SCR # 10
  0 ( PPS - LINE EDITOR ED SCR4 )
  1 : COPY    B/SCR * OFFSET @ + SWAP
  2           B/SCR * B/SCR OVER + SWAP
  3           DO DUP FORTH I BLOCK 2 -
  4           ! 1+ UPDATE LOOP DROP
  5           FLUSH ;
  6 HEX 1 1A +ORIGIN !
  7 FORTH DEFINITIONS DECIMAL
  8 LATEST 12 +ORIGIN !
  9 HERE   28 +ORIGIN !
 10 HERE   30 +ORIGIN !
 11 ' EDITOR 6 +  32 +ORIGIN !
 12 HERE FENCE  !
 13 ;S
 14
 15 ( PPS - LINE EDITOR END ) ;S
OK
```

```
7 LIST
SCR # 7
  0 ( FIG-FORTH 8080 ASSEMBLER )
  1 HEX VOCABULARY ASSEMBLER IMMEDIATE
  2 ' ASSEMBLER CFA ' ;CODE 0A + !
  3 : CODE    ?EXEC CREATE [COMPILE]
  4           ASSEMBLER !CSP ; IMMEDIATE
  5 : C;      CURRENT @ CONTEXT ! ?EXEC
  6           ?CSP SMUDGE ; IMMEDIATE
  7 : LABEL   ?EXEC 0 VARIABLE SMUDGE -2
  8           ALLOT [COMPILE] ASSEMBLER
  9           !CSP ; IMMEDIATE
 10 : 8*      DUP + DUP + DUP + ;
 11 ASSEMBLER DEFINITIONS
 12 -->
 13
 14
 15
OK
```

```
8 LIST
SCR # 8
  0 ( FIG-FORTH ASSEMBLER 2 )
  1 4 CONSTANT H   5 CONSTANT L
  2 7 CONSTANT A   6 CONSTANT PSW
  3 2 CONSTANT D   3 CONSTANT E
  4 0 CONSTANT B   1 CONSTANT C
  5 6 CONSTANT M   6 CONSTANT SP
  6 1245 CONSTANT NEXT
  7 : 1MI <BUILDS C, DOES> C@ C, ;
  8 : 2MI <BUILDS C, DOES> C@ + C, ;
  9 : 3MI <BUILDS C, DOES> C@ SWAP 8* + C, ;
 10 : 4MI <BUILDS C, DOES> C@ C, C, ;
 11 : 5MI <BUILDS C, DOES> C@ C, , ;
 12 -->
 13
 14
 15
OK
```

```
9 LIST
SCR # 9
  0 ( FIG-FORTH ASSEMBLER 3 )
  1 00 1MI NOP        76 1MI HLT
  2 F3 1MI DI         FB 1MI EI
  3 07 1MI RLC        0F 1MI RRC
  4 17 1MI RAL        1F 1MI RAR
  5 E9 1MI PCHL       F9 1MI SPHL
  6 E3 1MI XTHL       EB 1MI XCHG
  7 27 1MI DAA        2F 1MI CMA
  8 37 1MI STC        3F 1MI CMC
  9 80 2MI ADD        88 2MI ADC
 10 90 2MI SUB        98 2MI SBB
 11 A0 2MI ANA        A8 2MI XRA
 12 B0 2MI ORA        B8 2MI CMP
 13 -->
 14
 15
OK
```

```
10 LIST
SCR # 10
  0 ( FIG-FORTH ASSEMBLER 4 )
  1 09 3MI DAD        C1 3MI POP
  2 C5 3MI PUSH       02 3MI STAX
  3 0A 3MI LDAX       04 3MI INR
  4 05 3MI DCR        03 3MI INX
  5 0B 3MI DCX        C7 3MI RST
  6 D3 4MI OUT        DB 4MI IN
  7 C6 4MI ADI        CE 4MI ACI
  8 D6 4MI SUI        DE 4MI SBI
  9 E6 4MI ANI        EE 4MI XRI
 10 F6 4MI ORI        FE 4MI CPI
 11 22 5MI SHLD       2A 5MI LHLD
 12 32 5MI STA        3A 5MI LDA
 13 C4 5MI CNZ        CC 5MI CZ
 14 D4 5MI CNC        DC 5MI CC
 15 -->
OK
```

Already used in FORTH

12

```
11 LIST
SCR # 11
  0 ( FIG-FORTH ASSEMBLER 4 )
  1 E4 5MI CPO     EC 5MI CPE
  2 F4 5MI CP      FC 5MI CM
  3 CD 5MI CALL    C0 1MI RNZ
  4 C8 1MI RZ      D0 1MI RNC
  5 D8 1MI RC      E0 1MI RPO
  6 E8 1MI RPE     F0 1MI RP
  7 F8 1MI RM      C9 1MI RET
  8 C3 5MI JMP
  9 C2 CONSTANT 0= D2 CONSTANT CS
 10 E2 CONSTANT PE F2 CONSTANT 0<
 11 : NOT 8 + ;
 12 : MOV 8* 40 + + C, ;
 13 : MVI 8* 6 + C, C, ;
 14 : LXI 8* 1+ C, , ;
 15 -->
OK
```

```
12 LIST
SCR # 12
  0 ( FIG-FORTH ASSEMBLER 5 )
  1 : ENDIF 2 ?PAIRS HERE SWAP ! ;
  2 : THEN [COMPILE] ENDIF ;
  3 : IF C, HERE 0 , 2 ;
  4 : ELSE 2 ?PAIRS C3 IF ROT SWAP ENDIF 2 ;
  5 : BEGIN HERE 1 ;
  6 : UNTIL SWAP 1 ?PAIRS C, , ;
  7 : AGAIN 1 ?PAIRS C3 C, , ;
  8 : WHILE IF 2+ ;
  9 : REPEAT >R >R AGAIN R> R> 2 - ENDIF ;
 10
 11
 12
 13
 14
 15 ;S
OK
```

STARTING FORTH ------- A TUTORIAL

by
Mike Brinson
and
Larry Parsons

1. Introduction

FORTH is a general purpose interactive programming environment which includes a language compiler and interpreter, editor, assembler and operating system. It is memory-efficient while retaining high running speed. The FORTH language is very flexible, permitting the user to develop a working vocabulary tailored to a specific application.

The most prominent feature of FORTH is its data structure, which is called the DICTIONARY. The dictionary is an ordered list of machine code segments and other entries where each entry is called a WORD. A word has an associated name. A legal name for a word is a string of ASCII characters excluding SPACE, TAB, CARRIAGE RETURN, FORM FEED, LINE FEED, RUBOUT and NULL.

Examples:-

```
TEST
,me
#56
IDENTIFIER.ONE
```

2. A literal is a sequence of characters which describe a constant or a string. FORTH supports 16-bit and 32-bit integer literals. An integer literal is a sequence of digits optionally preceded by a plus or minus sign. No space may be embedded within a literal.

Examples:-

```
-1234          Legal.
+-100          NOT legal --- note +- operators.
-AFC0          Legal -- if the system BASE is HEX.
```

3. FORTH syntax is simple. A legal command line consists of a sequence of literals and/or names of words seperated by spaces, and terminated by a carriage return.

4. Programming in FORTH consists of defining a set of new words based on words which have already been defined. An initial vocabulary called the KERNEL, see the fig-FORTH glossary, enables the user to start programming in FORTH.

5. In FORTH communication between words is done by passing parameters on a "STACK". This stack is called the PARAMETER STACK or simply the stack. Typically, the parameters operated on by a word are pushed on the parameter stack, the word pops these parameters from the stack and pushes its results back on the stack.

6. FORTH uses reverse POLISH notation for all operations. This means that all operands precede their operations. Parentheses are never necessary.

Examples:-

```
1 1 + 2 +       is equivalent to    ( 1 + 1 ) + 2
1 2 3 * -       is equivalent to    1 - ( 2*3 )
```

7. In contrast to the majority of other high level computer languages, FORTH enables the user to manipulate addresses as well as data. It is VERY important that you understand the distinction between an address and its contents.

There are three types of words which are often used to push numbers on the stack, these are

1. literals
2. constants
3. variables

A reference to a LITERAL or a CONSTANT causes its VALUE to be pushed on the stack.

A reference to a VARIABLE causes its ADDRESS to be pushed on the stack.

The two operators "@" and "!" are used in FORTH to obtain and modify the value of a variable. They are defined as follows:-

@ Replace the address on the top of the stack by the 16-bit contents of that address. This word is used to load the contents of a memory location onto the stack.

! Store at the address on the top of the stack the number next to the top of the stack. Both numbers, i.e. the address and the number, are removed from the stack.

Examples:-

```
A,B and C are constants
D,E and F are variables

100 D !              Store 100 in D, i.e. assign D=100
E @ F !              Store the value of E in F, i.e
                     set the value of F equal to the
                     value of E.
D E !                Set the value of E to the address
                     of D.
D @ E @ + F !        Add the values of D and E and store
                     the result in F.
D A + F !            Store (address of D)+A in F.
D A B + + @ F !      Store in F the contents of
                     address, i.e. memory location,
                     D+A+B.
```

8. Operators

FORTH includes an unusually large number of fixed point operators. The following are among the more often used examples:-

8.1 Unary operators

The following operators replace the top of the stack with their result. In the examples the number on the top of the stack is called A and unless otherwise stated, all numbers are 16-bit integers.

Word	operation	comment
MINUS	-A	Leave the two's compliment of a number on the stack.
ABS	|A|	Leave the absolute value of n as an unsigned integer U on the stack.
1+	A+1	Increment n by 1.

8.2 Binary operators

The following operators replace the top two numbers on the stack with their results. In the examples the number on the top of the stack is called A and the number next to the top is called B.

Word	operation	comment
+	B+A	
-	B-A	
*	B*A	
/	B/A	
MAX		Leave the greater of the two numbers on the stack.
MIN		Leave the smaller of the two numbers on the stack.
AND		Leave the bitwise logical AND of A and B on the stack.
OR		Leave the bitwise logical OR of A and B on the stack.

8.3 Stack operators

A number of words are provided in FORTH whose sole function is to reorganize the elements on the stack.

Word	stack before	stack after	comment
DUP	A	A A	Duplicates top of stack.
OVER	A B	B A B	Copy the second stack value, placing it as the new top of stack.
DROP	A B	B	Drop the number from the top of the stack.
SWAP	A B	B A	Exchange the top two values on the stack.
ROT	A B C	C A B	Rotate the top three values on the stack, bringing the third item to the top of the stack.

8.4 Input/output operators

In the following examples "n" is the number on the top of the stack.

Word	comment
CR	Transmit a carriage return and a line feed to the selected output device.
SPACE	Transmit an ASCII blank to the output device.
SPACES	Transmit n ASCII blanks to the output device.
.	Print the number n, converting to the current numeric BASE, a trailing blank follows.
?	Print the value contained at the address, on the top of the stack, in free format according to the current BASE.
TYPE	Transmit count characters from addr. to the selected output device. STACK ——— top = n (count) top-1 = addr.

8.6 More general operations

Word	stack	comment
DECIMAL		Set current BASE to decimal.
HEX		Set current BASE to HEXidecimal.
+!	addr. n	Add n to the value at address addr.
MOVE	n addr2 addr1	Move the contents of n 16-bit memory cells begining at addr1 into n cells begining at addr2.

Examples (FORTH responses are underlined)

```
1 MINUS . <CR>                    NOTE:- the symbols <CR>
-1 OK

-1 ABS . <CR>                           denote the yellow
1 OK

1 1 + . <CR>                            carriage return
2 OK

1 1 1 + + . <CR>                        pressed.
3 OK

1 2 3 . . . <CR>
3 2 1 OK

1 2 SWAP . . <CR>
2 1 OK
1 2 OK
1 2 DUP . . . <CR>
2 2 1 OK

0 VARIABLE X <CR>
OK

100 X ! X ? <CR>
100 OK

-1 5 MAX . <CR>
5 OK

-1 5 MIN . <CR>
-1 OK
```

9. Colon definitions

FORTH allows definition, i.e. compulation, of new words in terms of previously defined words by means of the COLON definition. Its syntax is as follows:-

```
: NEW.WORD WORD1 WORD2 WORD3 WORD4 ..... WORDn ;
```

This sequence creates a new dictionary entry called NEW.WORD which, when executed, will execute the FORTH words WORD1, WORD2, WORD3 ..... in sequence; terminating with WORDn.
NOTE:- each of the words WORD1 ..... must already exist in the FORTH dictionary, otherwise an error will result.

Examples:-

```
: AVERAGE + 2 / ; <CR>
OK
```

This defines the word AVERAGE to be the average of the top two numbers on the stack, for example

```
3 9 AVERAGE . <CR>
6 OK
```

```
: 2ADD 2 + ; <CR>
OK
```

THis word adds 2 to the number on the top of the stack, for example

```
6 2ADD . <CR>
8 OK
```

10. Loops

There are two basic types of loop operation , these are

(a) limit index DO 'FORTH words' LOOP

(b) limit index DO 'FORTH words' increment +LOOP

In (a) the loop performs the 'FORTH words' inside the loop for each count from index to limit, incrementing by 1.

In (b) the loop performs the 'FORTH words' inside the loop for each count from limit to index, incrementing (or decrementing) by the value increment.

The limit and the index are kept on a second stack, called the RETURN stack, during the loop. The FORTH word I copies the index from the return stack to the parameter stack. NOTE:- the words DO, LOOP and +LOOP can only be used in a COLON definition sequence.

Examples:-

```
: COUNT.UP 10 0 DO I . LOOP CR ; <CR>
OK
DECIMAL
COUNT.UP <CR>
0 1 2 3 4 5 6 7 8 9
OK

: COUNT.UP1 10 0 DO I . 3 +LOOP ; <CR>
OK

COUNT.UP1 <CR>
0 3 6 9 OK

: COUNT.DOWN -10 0 DO I . -1 +LOOP ; <CR>
OK

COUNT.DOWN <CR>
0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10 OK
```

<<<<<< TUTORIAL EXAMPLES FOR BEGINERS >>>>>>

The following examples demonstrate a number of the basic features of the FORTH language. Each example can be entered from the keyboard. In a number of cases we have presented a BASIC equivalent program. However, do remember comparing computer languages can be misleading and should only be done as a guide to help understanding. Study the examples, experiment and see if you can firstly understand the FORTH words, and secondly try to rewrite each example using a different technique.

HAVE FUN !!!!!!!!!!!!!!!!!!!

NOTES

1. The symbols <CR> indicate the yellow carriage return key is pressed.
2. In most of the examples the FORTH responses are underlined.
3. Remember SPACES between FORTH words are significant.

FORTH TUTORIAL--1 addition of two constants

FORTH	BASIC
1 CONSTANT A <CR> OK	10 A=1
2 CONSTANT B <CR> OK	20 B=1
A B + . <CR> 3 OK	30 PRINT A+B
	40 END

FORTH TUTORIAL--2 Reverse POLISH expressions

```
FORTH                                   BASIC

1 CONSTANT A <CR>                       10 A=1
OK

2 CONSTANT B <CR>                       20 B=2
OK

3 CONSTANT C <CR>                       30 C=3
OK

A C + B / . <CR>                        40 PRINT (A+C)/B
2 OK

A DUP * B + . <CR>                      50 PRINT A*A+B
3 OK

0 VARIABLE D <CR>
OK

A B / D ! <CR>
OK

A C * D @ - . <CR>                      60 PRINT A*C-A/B
3 OK

D @ . <CR>                              70 END
0 OK
```

NOTES :-

The ratio A/B evaluates to 1/2 ------ 0 due to FORTH's 16-bit two's compliment arithmatic.

FORTH TUTORIAL--3 Timing loops

FORTH

```
DECIMAL <CR>
OK

: ST ." START " ; <CR>
OK

: FI ." FINISH " ; <CR>
OK

: TL1 30000 0 DO LOOP ; <CR>
OK

: TIMER1 CR ST CR TL1 FI CR ; <CR>
OK

TIMER1 <CR>
START       ------- Takes roughly 4.3
FINISH              seconds.
OK
```

BASIC

```
10 PRINT "START"

20 FOR I=0 TO 30000

30 NEXT I

40 PRINT "FINISH"

50 END
```

Takes roughly
34.9 seconds.

```
DECIMAL <CR>
OK

0 VARIABLE VA <CR>
OK

: TL2 30000 0 DO I VA ! LOOP ; <CR>
OK

: TIMER2 CR ST CR TL2 FI CR ; <CR>
OK

TIMER2 <CR>
START      -------- Takes roughly
FINISH              9.2 seconds.
OK
```

```
10 PRINT "START"

20 FOR I=0 TO 30000

30 VA = I

40 NEXT I

50 PRINT "FINISH"

60 END
```

Takes roughly
81.5 seconds.

FORTH TUTORIAL--4 Calculations in different BASES

```
DECIMAL <CR>
OK
4 CONSTANT C <CR>
OK
8 CONSTANT D <CR>
OK
D C + . <CR>
12 OK
HEX <CR>
OK
D C + . <CR>
C OK
DECIMAL <CR>
OK
: OCTAL 8 BASE ! ; <CR>
OK
OCTAL <CR>
OK
D C + . <CR>
14 OK
DECIMAL <CR>
OK
: BINARY 2 BASE ! ; <CR>
OK
DECIMAL <CR>
OK
: DECIMAL-LOOP 10 1 DO I . LOOP ; <CR>
OK
DECIMAL-LOOP <CR>
1 2 3 4 5 6 7 8 9 OK
: DECIMAL-BINARY <CR>
                 DO DECIMAL CR I DUP . <CR>
                 ."      " BINARY . <CR>
                 LOOP ; <CR>
OK
DECIMAL <CR>
OK
10 0 DECIMAL-BINARY <CR>
0      0
1      1
2      10
3      11
4      100
5      101
6      110
7      111
8      1000
9      1001 OK
```

FORTH TUTORIAL--5 Fast screen clear programs

```
FORTH

: SCREEN-CLEAR1 ." C " ; <CR>
OK

NOTE:- The above definition clears the screen when you type
C ------ not very convenient, but very fast !

HEX <CR>
OK

: SCREEN-CLEAR2  <CR>
                 HEX <CR>
                 D000 800 0 FILL <CR>
                 DECIMAL <CR>
                 ; <CR>
OK

NOTE:- The MZ80K memory mapped screen starts at D000H and is
400H bytes long.

: SCREEN-CLEAR3  DECIMAL 12 EMIT ; <CR>
OK

NOTE:- The above definition is simple and very fast.


BASIC

10 PRINT " C "  ---------- Equivalent to SCREEN-CLEAR1.
20 END

20 FOR I = 53248 TO 54247
30 POKE I,0

40 NEXT I

50 END          ---------- Equivalent to SCREEN-CLEAR2
                                    SLOW !!!!!!!!!!!!
```

FORTH TUTORIAL--6 Character display program

This program fills the screen with each MZ80K V.D.U. character, starting with code 0 through to code 255. As each screen contains 1000 characters a total of 255000 characters are written to the V.D.U. NOTE, the FORTH run time. With practice this could lead to some interesting graphics programs !!!!!!!

```
FORTH

: SCREEN-DISPLAY <CR>
                    HEX <CR>
                    FF 00 DO D000 800 I FILL LOOP <CR>
                    DECIMAL <CR>
                    ; <CR>
OK

SCREEN-DISPLAY <CR>
OK                  ---------- RUN TIME roughly 14 seconds


BASIC

10 FOR K = 0 TO 255

20 FOR I = 53248 TO 54247

30 POKE I,K

40 NEXT I

50 NEXT K

60 END  ------------  RUN TIME roughly 20 to 30 minutes !!!!!!!
```

FORTH TUTORIAL--7 HEX dump program

```
HEX <CR>
OK

: PDUMP <CR>
                HEX 100 * DUP 100 + SWAP <CR>
                C EMIT <CR>
                ." ADDR 0 1 2 3 4 5 6 7 8 9 A B C D E F " <CR>
                DO CR I 5 .R <CR>
                I 10 + I <CR>
                DO I C@ 2 .R LOOP <CR>
                10 +LOOP ; <CR>
OK

2 PDUMP <CR>
```

NOTE:- PDUMP expects the page number to be on the top of the stack.

Program output

```
ADDR  0 1 2 3 4 5 6 7 8 9 A B C D E F
  200273237 2C345 181C9E8 1 D 22A28 1
  2105E2356D----------------------------
  ----------------------------------------
  ----------------------

                    ----------------------
  ----------------------------------------

  2F0D9 2F4 2C3B616893F5445524D494E40K
```

NOTE:- Compare this program with the BASIC version of HEX dump published in issure 3 of the User Notes.

FORTH TUTORIAL---8 IF--------THEN

Structure

```
        (false)
IF      (true)  'FORTH words'  THEN   'FORTH words'
```

The FORTH word IF tests for a 0 (false) or a non-zero (true) on the top of the stack. If false the program branches to the 'FORTH words' following THEN. NOTE:- that the 'FORTH words following THEN are always executed irrespective of true or false at IF.

Program to print the absolute value of a number:-

```
: ABSOLUTE-VALUE DUP 0< IF MINUS THEN . ; <CR>
OK
```

True if the top of the stack is negative.

If negative change sign.

```
56 ABSOLUTE-VALUE  <CR>
56 OK

-56 ABSOLUTE-VALUE <CR>
56 OK
```

Program to print the numbers between 1 and 100 which are divissible by a given integer:-

```
: DIVN          100 1 DO DUP      ( set loop limit ) <CR>
                I  SWAP  MOD      ( divide loop index by  <CR>
                                    integer and put remainder <CR>
                                    on stack ) <CR>
                0= IF I . CR      ( IF MODULO = 0 then print <CR>
                THEN                number and CR ) <CR>
                THEN DROP LOOP ;  ( remove MODULO and exit loop <CR>
                                    ) <CR>
OK

15 DIVN <CR>

15                          NOTE:- "(" followed by a space is the
30                                 FORTH word for defining the
45                                 start of a comment.  Comments
60                                 are terminated by a space and
75                                 a ")".
90
OK
```

FORTH TUTORIAL--9 IF-----ELSE-----THEN

Structure

```
     (false)
IF   (true)  'FORTH words'  ELSE  'FORTH words'  THEN 'FORTH words'
```

As with IF-----THEN the IF tests the top of the stack. If true the 'FORTH words' between IF and ELSE are executed. The program next branches to the 'FORTH words' following THEN. If false the 'FORTH words' between ELSE and THEN are executed.

Program to test whether a number, representing a temperature, is greater or less than 150. If less than 150 the program prints the message " TEMPERATURE SAFE ". If greater then 150 the program prints the message " DANGER---- TOO HOT ".

```
: TEMP-TEST  150 - 0< ( Test for greater or less than 150 ) <CR>
      IF ." TEMPERATURE SAFE " ( Print message TEMPERATURE <CR>
                                 SAFE if < 150 ) <CR>
      ELSE ." DANGER ---- TOO HOT " ( PRINT message DANGER  <CR>
                                      ---- TOO HOT if >150 ) <CR>
      THEN ; <CR>
OK

120 TEMP-TEST <CR>
TEMPERATURE SAFE OK

170 TEMP-TEST <CR>
DANGER ---- TOO HOT OK
```

FORTH TUTORIAL--10 BEGIN---------UNTIL

Structure

```
BEGIN          'FORTH words'          UNTIL
                                        I
                                        I
                                      Exit if true.
```

The top of the stack is tested at UNTIL. If false (0) the loop commencing at BEGIN is repeated. If true (#0) an exit from the loop occurs.

Program to print the numbers between 1 and 100.

```
: KOUNT 0 BEGIN 1 + DUP . CR        ( Add 1 to stack and <CR>
                                      print number ) <CR>
        100 = UNTIL ;               ( If # 100 repeat ) <CR>
OK

KOUNT
1
2
3
4
---
---
---
---
99
100
OK
```

NOTE:- Compare timings for a longer loop - say 2000 for the above program and a similar program which uses the DO------LOOP construction.

FORTH TUTORIAL---11 BEGIN---WHILE---UNTIL

Structure

```
BEGIN   'FORTH words'   WHILE      (true)   'FORTH words'   REPEAT
                                   (false)
```

The test in this loop is made at WHILE. If false the program exits from the loop. If true the program continues to REPEAT and then branches back to BEGIN.

Program to print all the numbers and their squares, starting from 1, stopping when the square >= 1500.

```
: SQUARE 1 BEGIN DUP DUP * * ( Number and square on stack ) <CR>
         DUP 1500 <          ( Is square < 1500 ) <CR>
         WHILE . DUP . CR <CR>
         REPEAT ;            ( If yes print number and <CR>
                             square ) <CR>
OK

      SQUARE <CR>
       1 1
       2 4
       3 9
       ---
       ---
       ----
       OK
```

FORTH TUTORIAL--12 Prime numbers

Program to print all the prime numbers between 1 and a given number.

```
1 VARIABLE PRIME <CR>
OK
1 VARIABLE FLAG <CR>
OK
: TEST 0 FLAG ! PRIME @       ( Set FLAG to zero put <CR>
                               prime on stack ) <CR>
  2 DO PRIME @ I MOD 0=       ( Loop from 2 to PRIME-1 <CR>
                               Divide prime by loop index. ) <CR>
  IF 1 FLAG ! LEAVE  <CR>
  THEN LOOP ;                 ( Modulo = 0, number is not a <CR>
                               PRIME. Set FLAG = 1 and EXIT <CR>
                               LOOP. ) <CR>
OK

: PPRIME 2 DO I PRIME ! TEST ( Store loop index in <CR>
                               variable PRIME. ) <CR>
         FLAG @ 0=           ( Test to see if PRIME is a <CR>
                               PRIME ) <CR>
         IF I . CR THEN      ( If number is a PRIME print ) <CR>
         LOOP ; <CR>
OK

1000 PPRIME <CR>
1
3
5
---
---
---
997
OK
```

More FORTH

Learning a new computer language can be difficult, time consuming and at worst frustrating. This is particularly true with FORTH where its syntax is 'foreign' to the reader. One technique which helps new FORTH programmers become proficient is to study published programs. At first constant reference to the fig-FORTH glossary is needed to decode FORTH words. However, you will find, with practice that it is possible to read and understand FORTH listings.

There are three major sources of example programs: firstly the popular american and U.K. computer magazines, secondly the american fig-FORTH Interest group publications and thirdly the U.K. FORTH Interest group publications. A bibliography of FORTH articles is presented at the end of these notes to help you get started in a search for example programs.

The fig-FORTH Interest and U.K. FORTH Interest groups are independent organizations dedicated to supporting the FORTH language. Both groups publish a magazine. These publications a full of helpful tips and example programs.

Further information about these groups activities can be obtained from:-

1. The Hon Secretary,
FORTH INTEREST GROUP U.K.,
C/O 38 Worsley Road,
Frimley,
Camberley,
Surrey, CU16 5AU. Tel. (02516) 6254

2. FORTH INTEREST GROUP,
P.O. BOX 1105,
San Carlos, CA 94070,
U.S.A.

The U.K. FORTH Interest group's Magazine is called **FORTHWRITE FIG UK.** Recent issues include articles on multi-dimensional arrays, Z80 FIG Assembler, INFIX to RPN, 6502 Assembler, discompiler, double precision arithmetic, and a CP/M FIG FORTH update and editor.

The american fig-FORTH group's magazine is called **FORTH DIMENSIONS.** All the information published in FORTH DIMENSIONS has been placed in the public domain by fig-FORTH. The following short programs are taken from FORTH DIMENSIONS. Our thanks to fig-FORTH and the authors of these programs for allowing their articles to be reprinted.

If you wish to keep up-to date with the FORTH world -------- then do join the FORTH interest groups.

FORTH BIBLIOGRAPHY

J.S. James, FORTH for microcomputers, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, No. 26, pp 21–27.

C.H. Moore, the evolution of FORTH, an unusual language, Byte, August 1980, pp 76–92.

K. Harris, FORTH extensibility – or how to write a compiler in 25 words or less, Byte, August 1980, pp 164–190.

S.M. Hicks, FORTH's forte is tighter programming, Electronics, March 1979, pp 114–118.

J.S. James, FORTH dump programs, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, No. 28, pp 26–27.

R.B. Main, FORTH versus assembly, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, No. 31, pp 45–47.

G. Filby, FORTH, Computer Age.
Dec. 1979, pp 58–59, Jan. 1980, pp 31–32,
Feb. 1980, pp 48–49, March 1980, pp 56–57,
Oct. 1980, pp 38–40, Nov. 1980, pp 42–43,
Jan. 1981, pp 25–26.

J.S. James, FORTH A tutorial introduction, Byte, August 1980, pp 100–126.

R. Deane, A proposal on strings for FORTH, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, No. 50, pp 40–45.

R. Deane, Adding arrays to FORTH, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, No. 46, pp 25–26.

H. Mannoni, FORTH – an extensible path to efficient programs, Electronic Design, July 19, 1980, pp 175–178.

J. Hopprich, A FORTH introduction, Liverpool Software Gazette, May 1980, pp 44–46.

R. Fritzen, Write your own FORTH interpreter, Microcomputing, Feb. 1981, pp 76–92.

R. Fritzen, Write your own Pseudo-FORTH compiler, Microcomputing, March 1981, pp 44–64.

J.O. Bumgarner, A coding sheet for FORTH, Byte, March 1981, pp 155–162.

U. Frei, KNIGHT: A knight's tour problem in MMSFORTH, Byte, Feb. 1981, p 325.

J.J. Cassady, Stacking strings in FORTH, Byte, Feb. 1981, pp 152–162.

R. Miller and J. Miller, Breakforth into FORTH !, Byte, August 1980, pp 150–163.

T. Ritter, Varieties of threaded code for language implementation, Byte, Sept. 1980, pp 206–227.

K. Harris, The FORTH philosophy, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, Sept. 1981, pp 6–11.

W.F. Ragsdale, A FORTH assembler for the 6502, Dr Dobb'e Journal of Computer Calisthenics and Orthodontia, Sept. 1981, pp 12–24.

R.G. Loeliger, Sallying FORTH into battle, Dr Dobb's Journal of Computer Calisthenis and Orthodontia, Sept. 1981, pp 25–26 and page 48.

H. Laxen, Screen-oriented editor in FORTH, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, Sept. 1981, pp 27–41.

G.B. Haydon, Elements of a FORTH data base design, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, Sept. 1981, pp 42–47 and page 56.

R. Duncan, FORTH decompiler, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, Sept. 1981, pp 49–53.

M.E. Timin, The FORTH alternative, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, Sept. 1981, pp 57–59.

M.E. Brinson, Not fifth but FORTH, a language that builds itself, Computing Today, Oct. 1981, pp 34–35.

S.B. Bassett, Algorithms in FORTH, Dr Dobb's Journal of Computer Calisthenics and Orthodontia, April 1981, pp 16–17 and page 20.

HELP

SCR No. 6

```
0       THE 'HELP' COMMAND IS PROBABLY THE MOST USEFUL OPTION FOR
1   A FORTH SYSTEM. IT ALLOWS YOU TO VIEW THE DICTIONARY WORDS
2   AND LOCATE THEM IN MEMORY. WHEN YOU ARE TESTING NEW
3   DEFINITIONS, IT WILL SHOW RE-DEFINITIONS. IT IS A WAY TO
4   LOCATE WHERE A MISSING WORD SHOULD BE, BUT ISN'T.
5
6       IF YOU MAKE A COMPILE ERROR FROM DISC, 'HELP' WILL SHOW
7   THE WORD IN WHICH THE ERROR OCCURED.
8
9       YOU SHOULD MODIFY THE FOLLOWING DEFINITIONS TO THE FORMAT
10  YOU WANT. FOR OBJECT CODE EXAMINATION, I LIKE THE CODE FIELD
11  ADDRESSES AS SHOWN, SINCE THIS IS WHAT RESULTS IN THE COMPILED
12  CODE. FOR A QUICK SNAP-SHOT OF THE DICTIONARY, I JUST PRINT
13  THE LENGTH AND NAMES.
14
15  JUST TYPE 'HELP' AND HIT THE 'BREAK' KEY TO STOP.
```

SCR NO. 7

```
0  ( HELP )                                HEX
1  00          CONSTANT LAST. LINK         (IS $8000 on MICRO-FORTH)
2   4          CONSTANT # /LINE            (WORDS PRINTED PER LINE)
3
4  : .NAME                    ( ENTER WITH ADDRESS OF LENGTH BYTE )
5    DUP   C@  7F   AND   DECIMAL   3   .R   SPACE   1+   3   TYPE SPACE ;
6
7  : .CODE-ADDRESS            ( ENTER WITH ADDRESS OF LENGTH BYTE )
8    6   +   HEX   5   .R   SPACE ;
9
10 : .HEADER                  ( ENTER WITH ADDRESS OF LENGTH BYTE )
11   DUP   .NAME   .CODE-ADDRESS ;
12
13 :?TERMINAL   0   ;              (USER'S MACHINE DEPENDENT TERMINAL BREAK)
14                ( RETURN '00' FOR NO BREAK, AND '01' FOR A BREAK )
15 8 LOAD   ; S   8/27/78   WFR
```

SCR No. 8

```
0  ( HELP, CONT. )
1
2  : .LINE                    ( PRINT A LINE OF NAMES AND CODE ADDRESSES )
3    #/LINE 0                 ( ENTER WITH ADDRESS OF LENGTH BYTE )
4    DO   DUP   .HEADER SPACE   4   +   @   DUP   LAST.LINK =
5         IF LEAVE THEN LOOP ;          ( EXIT WITH NEXT ADDRESS )
6
7  : HELP                     ( PRINT DICTIONARY FROM TOP CURRENT WORD DOWN )
8                             ( TO BOTTOM. FORMAT IS LENGTH COUNT, 3 LETTERS OF )
```

```
 9                                  ( NAME, AND CODE FIELD ADDRESS. WILL TERMINATE )
10                                    ( UPON LAST LINK VALUE OR A TERMINAL BREAK. )
11    BASE C@  >R   CURRENT   @  @
12    BEGIN CR   .LINE    DUP    LAST.LINK    =    ?TERMINAL    +
13      END    DROP   ( LAST LINK )   R>   BASE   C!;
14
15  DECIMAL   ;S   8/28/78   WFR

HELP
 4 HEL    1EBB        5 .LI     1E90        7 .HE     1E80        13.CO     1E68
 5 .NA    1E43        6 # /L    1E39        9 LAS     1E2F        4 TAS     1E 25
 4 BOO    1B8B        2 ->      1B6A        4 TUB     1B57        3 TTY     1B44 OK
```

ABOVE WE SEE AN EXAMPLE OF THE LOADING OF THE 'HELP' COMMAND FROM DISC. IT THEN IS TESTED, AND DUMPS THE DICTIONARY. WE SEE THE LISTING OF 'HELP' AND THE WORDS IT USES; LISTING CONTINUES INTO THE RESIDENT DICTIONARY.

GOOD LUCK, WFR

SCR No. 3

```
 0                GLOSSARY DOCUMENTATION – D.W. BORDEN
 1        AFTER READING W.F. RAGSDALE'S ARTICLE TO THE 'HELP' COMMAND
 2  IN FORTH DIMENSIONS NO.2 AND SOME PROPAGANDA FROM FORTH INC., I
 3  WROTE A SHORT PROGRAM WHICH PRINTED OUT EACH FORTH WORD AS IT
 4  IS DEFINED, THE ADDRESS OF THE LENGTH BYTE AND THE ADDRESS OF
 5  THE PARM BYTE. AFTER EACH ENTRY, I PRINTED ELLIPSES TO ALLOW
 6  A HANDWRITTEN ENTRY OF WHAT EACH COMMAND IS SUPPOSEDLY DOING.
 7  THE ADDRESS OF THE PARM BYTE IS USEFUL SINCE THAT ADDRESS
 8  APPEARS IN HIGH LEVEL COLON DEFINITIONS 'OBJECT CODE. THUS, IF
 9  YOU HAVE NOT WRITTEN YOUR OWN FORTH SYSTEM, YOU CAN HAND
10  DISASSEMBLE ( DISFORTH MIGHT BE MORE APPROPRIATE ) EACH COMMAND
11  AND SEE WHAT IT IS DOING.
12        OF COURSE A DISFORTH PROGRAM COULD BE WRITTEN AND I HAVE DONE
13  SO, BUT I AM NOT HAPPY WITH ITS OUTPUT FORMAT YET. I ALSO HAVE
14  WRITTEN A TRACE WHICH PUTS A TRAP IN 'NEXT' AND LISTS EACH FORTH
15  COMMAND EXECUTED. VERY USEFUL FOR DEBUGGING.          ;S      01/31/79
```

SCR No. 2

```
 0  ( GLOSSARY OF FORTH WORDS WITH HEAD AND PARM ADDRESSES )
 1  0 VARIABLE CMD ( TEMPORARY VARIABLE TO HOLD COMMAND )
 2  : TOPOFPAGE CR CR [ CMD    HEAD PARM ] CR ;
 3  :UNDERLINE[ ..............................................................] ;
 4  : GLOSSARY TOPOFPAGE CURRENT@@CMD ! ( GET TOP CMD ADDRESS )
 5  BEGIN
 6  CMD@ IF CMD@ C@DUP 80 AND – ( GET COMMAND LENGTH )
 7  DECIMAL . HEX 4 1 DO ( PRINT COMMAND LENGTH )
 8  CMD@ I + ( INDEX FETCHED IS 1–2–3 )
 9  C@ECHO ( PRINT COMMAND LETTER )
10  LOOP SPACE
```

```
11  CMD @.H SPACE ( PRINT COMMAND HEAD ADDRESS )          example fig-FORTH
12  CMD@6 + .H   UNDERLINE CR CR (PRINT CMD PARM ADDRESS)  [      ."
13  CMD@4 +@ CMD ! ( PICK UP LINK WITH NEXT CMD ADDRESS )  [      ."
14  ELSE QUIT THEN                                        ECHO      EMIT
15  AGAIN ;          ;S        1/30/79        DWB          .H        H. (hex
                                                                     output)
```

GLOSSARY

CMD	HEAD	PARM
8 GLO	1C2A	1C30
9 UND	1BEE	1BF4
9 TOP	1BC9	1BCF
3 CMD	1BBD	1BC3
9 ?TE	1BA1	1BA7

Recursion – The Eight Queens Problem

Jerry LeVan
Dept. of Math Science
Eastern Kentucky University
Richmond, KY 40475

The Eight Queens problem has been often used as a textbook problem in programming, particularly to illustrate recursion. I present here a solution in FORTH.

Recursion is the technique of allowing a procedure (a FORTH word definition) to call itself. This is normally blocked during FORTH compilation, to allow an old word name to be used in a new definition of the same name. For example:

```
: HELP  CR  CR  HELP  CR  CR ;
```

The new definition of HELP will execute a previous definition, but with two carriage returns before and after. This is a necessary and common capability.

How then to have a word call itself, if not by name? The answer is MYSELF. This word will compile a call to the word in which it is located:

```
: DEMO
-----
IF MYSELF ELSE — THEN ;
```

In this example, if the test is true at IF, at MYSELF a call to DEMO will occur. This is accomplished by defining MYSELF as IMMEDIATE. At compile time, MYSELF executes and compiles the execution (code field) address of the most recent (actually incomplete) definition in the CURRENT vocabulary. The fig-FORTH definition is:

```
: MYSELF
LATEST PFA CFA  ,  ; IMMEDIATE
```

The Four Queen Problem at hand finds the board row and column locations on which eight chess queens would be safe from mutual attack. This example doesn't check for board rotations or reflections, so more answers are printed than necessary.

The output gives the calculation number on which the answer was found and a list of the eight row numbers, column by column on which the queens are located. Now it's your turn to DO-IT.

```
SCR # 02
  0 (  8 queens   by Jerry LeVan                          WFR-79DEC02 )
  1 : 2*    DUP  +  ;  ( double a value )
  2
  3 : MYSELF    ( allow a word to call itself, by recursion )
  4       LATEST  PFA  CFA  ,  ;  IMMEDIATE
  5
  6 : IARRAY   ( makes an array of 1's, as given by input )
  7     <BUILDS  0  DO  1  ,  LOOP
  8     DOES>  SWAP  2*  +  ;  ( leave address within array )
  9
 10  8 IARRAY A   ( these form workspace for the solutions )
 11 16 IARRAY B
 12 16 IARRAY C
 13  8 IARRAY X   ( this contains trial solutions )
 14 -->
 15
```

```
SCR # 03
  0 (  more 8 queens                                      WFR-79DEC02 )
  1 : SAFE
  2     SWAP  OVER  OVER  OVER  OVER
  3     -  7  +  C  @  >R
  4     +  B  @  >R
  5   DROP  A  @  R>  R>  *  *  ;
  6 : MARK
  7     SWAP  OVER  OVER  OVER  OVER
  8     -  7  +  C    0  SWAP  !
  9     +  B  0  SWAP  !
 10     DROP  A  0  SWAP  !  ;
 11 : UNMARK
 12     SWAP  OVER  OVER  OVER  OVER
 13     -  7  +  C  1  SWAP  !
 14     +  B  1  SWAP  !
 15     DROP  A  1  SWAP  !  ;  -->
```

```
SCR # 04
  0 (  more 8 queens                                      WFR-79DEC02 )
  1 0 VARIABLE TRIES
  2 : PRINTSOL   ( print one solution )
  3       ." found on try "  TRIES  @  6 .R  8  0
  4   DO  I  X  @  1+  5  .R  LOOP  CR  ;
  5 : TRY        8  0       ( search for answers )
  6   DO 1 TRIES +!  ?TERMINAL  IF  QUIT  THEN  DUP  I  SAFE
  7     IF  DUP  I  MARK
  8         DUP  I  SWAP  X  !
  9         DUP  7  <
 10           IF  DUP  1+  ?STACK  MYSELF  ELSE  PRINTSOL  THEN
 11         DUP  I  UNMARK
 12       THEN
 13   LOOP  DROP  ;
 14
 15 : DO-IT    0 TRIES !  0  CR  TRY  ;   ( This runs the problem )
```

XTAL BASIC FOR THE MZ80K

A number of readers have written to us asking whether XTAL BASIC is available for the MZ80K. The answer is of course yes. A tape based version of this popular BASIC interpreter has been available for over a year. We understand from Mr T.F. Brownen of Crystal Electronics that a CP/M version, to run under XTAL CP/M, will also be on sale in the near future. However, please note the release date for the CP/M XTAL BASIC is not yet known.

XTAL BASIC is an important package for two major reasons, firstly it only requires roughly 9K of RAM to load the interpreter yielding more user space (this is particularly important on the basic 20K RAM MZ80K model), and secondly it allows users to add their own keywords to extend the BASIC interpreter. An excellent article, with examples, on how to extend XTAL BASIC was published in the October edition of Computing Today – see Basic Upgrade by Paul Kriwaczek, pp 57–59.

Recently Crystal Electronics have published a small book outlining the inner working of XTAL BASIC. This is an invaluable reference source for software 'hacks'. Please note this book is only available to purchasers of XTAL BASIC.

The background to XTAL BASIC is given in the following article written by the package's author. Our thanks to Mr T.F. Brownen for giving us permission to reprint Mr Andrew Cornish's article. The article was originally published in the August 1980 edition of the Liverpool Software Gazette.

XTAL BASIC – THE EXTENDABLE ONE

by
Andrew Cornish BSc.
Crystal Electronics Torquay Devon

Nowadays, in micro-computing circles, the term 'BASIC' seems to cover a multitude of sins! The number of variations of this most popular of high-level computer languages has grown to such an extent that standardisation is now virtually impossible and, frequently, several variations may be available for the same machine.

The NASCOM microcomputers are a case in point – here one has a choice of at least (and there may be more by the time this goes to press) TEN different versions of BASIC. These range from the 'Tiny' variety, which offer whole number arithmetic and have little or no character manipulation facilities, but are extremely useful for games programs. At the other end of the scale, we have the 'Standard' variety, offering floating-point numbers,

string handling, multi-dimensional arrays, and so on. There is even talk (plenty of that!) of producing 'Extended' BASICs occupying 12K or 16K of memory.

The perplexed NASCOM owner may well raise his hands in horror and as 'But where do I jump into this rat-race? If I add an 'X 'K BASIC to my system, the chances are that it will need to be replaced in a few months time by a bigger one, thus wasting my money. On the other hand, will MAFIA COMPUTERS INC. EVER bring out their promised 24K SUPER EXTENDED BASIC with added washing-machine driver software?' Moreover, the perplexed 'home-brew' system owner has virtually NO choice – as far as I can tell, there has (until now) been no Z80-based '8K' BASIC available to such a user.

I believe that there is a need for a BASIC Interpreter that contains the flexibility to grow with the needs of the user, and Xtal (pronounced 'Crystal') BASIC has been developed with just this aim in mind. It contains just about all of the features that one would expect to find in a 'Standard' 8K BASIC, i.e., floating-point numbers (to 6 significant figures accuracy), any-length variable names, strings, arrays, multi-dimensional arrays and string arrays, string and transcendental functions, and over 20 error messages – all in 7424 bytes, to be precise! Table 1 gives a list of the available commands and functions of Xtal BASIC 2.2 in its fundamental form. As you can see, it contains several 'extras' not usual in most implementations of BASIC, and some of these are explained in a little more detail in Table 1. The purpose of this article, however, is to explain the most important feature of Xtal BASIC, i.e., its ability for expansion.

Some readers may already know that most BASIC Interpreters do not actually use a line of BASIC as typed – instead, each reserved word (e.g., PRINT, FOR) is shortened into a unique single-byte code. This speeds up program execution and also saves storage space (sometimes quite dramatically – as one prominent advertiser puts it, 'It packs the RAM more tightly!').

To illustrate this point, consider the following one-line 'program':

```
FOR I = 0 to 9:PRINT SQR (I):NEXT:END
```

A normal text editor would store this line in memory in the form of ASCII codes, thus:

FOR I = 0 TO 9 : PRINT SQR (I) 46 4F 52 20 49 3D 30 20 54 4F 20 39 3A 20 50 49 4E 54 20 53 51 52 28 49 29

. . . and so on. This would be abbreviated by the BASIC Interpreter, making use of the fact that the Hex numbers 80H to FFH are not ASCII codes, and representing each reserved work by one of these codes:

FOR I = 0 TO 9 : PRINT SQR (I) :NEXT :END 81 20 49 B0 30 20 A2 20 39 3A 20 98 20 B8 28 49 29 3A 82 3A 80

Note that even the '=' is treated as a reserved word, although it has only one character. This is so that execution will be faster when '=' is used as a relational operator (along with '<' and '>').

The Interpreter contains a table of reserved words and an associated table of routine addresses so that, in execution, the reserved word code may be used as an index to this table, to jump straight to the routine in machine-code that represents that particular command or function. Note that the LIST command will 'blow up' a reserved word code back into the actual word that it represents, so that the user need never be aware that all of this is actually going on.

This much is not at all new – the point about Xtal BASIC is that it has TWO reserved word tables, one of which is empty, and may be expanded by the user (and by ourselves, of course!) to contain up to 64 extra reserved words of his/her own choosing. The user can write a machine-code routine for the command, enter the address of the start of that routine into the address table, and then the ASCII codes representing the letters of the reserved word in the auxiliary reserved word table. If Xtal BASIC is then run up and the new command typed as part of a line of BASIC, it will be executed as if it had always been a part of the Interpreter.

Fig. 1 shows a simplified memory map of a typical Z80 system incorporating Xtal BASIC, and serves to illustrate another aspect of its flexibility. Three important address pointers to note within the scratch-pad are:

TOPRAM – This points to the top address of the RAM space available to BASIC. This is normally the same as the physical topmost RAM address, but can be set to a lower address by means of the CLEAR command.

TXTUNF – This points to the start of the current BASIC program (i.e., the first memory location of the space for storing variables).

TEXT – This points to the start of the current BASIC program in memory, which is normally 2D00H. This address is stored within the Interpreter (at 1283H and 1284H, known as the 'hard' TEXT pointer) and copied into the scratch-area when BASIC is first entered (to OC8CH and OC8DH, or the 'soft' TEXT pointer – so called because although it may be changed, it reverts back to the value of 'hard' TEXT if BASIC is subsequently re-initialised. If 'soft' TEXT is changed to point to some higher location (e.g., 2E00H instead 2D00H), all BASIC programs will load at that new address, and the area under it will then be free for storing machine-code routines. One is then in the happy position of being able to store the entire Interpreter plus 'extras' onto a new tape, as on continuous block (from 0E80H to 2E00H, or as far up as your extra routines go). One should also change 'hard' TEXT to point to this new address before saving the Interpreter.

We can make further use of this idea, to append one BASIC program to another. This time, we load the first program, and then POKE 'soft' TEXT to point to TXTUNF-2. We next load the second program (with line numbers higher than those of the first), restore 'soft' TEXT to its original position, and, finally, save the resulting long program.

In addition to the above, there is also the facility of storing machine-code routines ABOVE the BASIC program and variable space, by utilising the CLEAR command to set the TOPRAM pointer to some lower value. This is not quite so useful for software exchange between systems, since some systems will not have RAM in that area, whereas others will. It is, however, useful for adding temporary routines to the system, or routines that are only likely to be used by, say, one BASIC program (e.g., a Chess program which may do a lot of its work in BASIC, and require certain routines in machine-code for maximum execution speed). We can use the CALL command to execute a machine-code routine at a specified memory address, if we don't want to assign a reserved word to it.

EXAMPLE – 'CLS'

As a simple example, let us define a command to clear the VDU screen on the computer. This is normally done by printing the 'Clear Screen' character (ASCII code 0CH, but is 1EH on the NASBUG T2, BBUG and T4 monitors, and is 16H for the SHARP MZ80K, for example). In machine-code therefore, we put the appropriate code into the Accumulator and jump to the address of the routine to print out the contents of the Accumulator to the screen:

```
2D00 3E 0C     LD  A,0CH
2D02 C3 F4 2B  JP  CRTVEC
```

Location 2BF4H marks the jump to the monitor output routine, whose address depends upon the monitor being used. Next, change location 1284H (the high byte of 'hard' TEXT) from 2DH to 2EH, so that BASIC programs will load starting from 2E00H and not destroy our new routine.

Now we must actually put the command word and address into the tables:

```
0E80  C3  4C  53  80
      C   L   S
0F80  00  2D          The address 2D00
```

The reason for putting C3 and not 43 for the letter 'C' is so that the first letter of each word can be identified by the state of the top 'bit' of the byte. Since CLS is the last (and the first in this case!) reserved word in the list, we put a code 80 after it, to indicate that there are no more words in the list – if we now define another word, it will start from 0E83H, where the 80 is.

FINE! We now have an extra command available to the Interpreter, and we can run it up, type CLS (followed by the 'Return' key!) and watch the screen disappear.

As well as commands, we may also have functions, and the procedure is quite similar, except that we must include a left bracket '(' with the function name, so that Xtal BASIC can tell it apart from a command when syntax checking. It does not matter in what order we define the new commands and functions – execution speed is exactly the same whether the command is the first or last in the list.

FUTURE DEVELOPMENTS

Xtal BASIC is being developed on several fronts at the present time. First we are making it available on other Z80 machines – as well as the NASCOM I/II, it is now available for the SHARP MZ80K. Although this machine already has a BASIC, it is generally considered somewhat limited for its size (14K). Xtal BASIC 2.2S implements all of the features of SHARP BASIC in about 5K LESS space (including the MUSIC, Real Time Clock and Tape Data File commands), although SHARP BASIC does have a slightly greater floating-point accuracy. The PRINT@feature of Xtal BASIC allows printing at given co-ordinates in the screen, and thus saves that enormous quantity of unsightly (and unprintable!) curser arrows in listings of programs using graphics.

The MZ80K version of Xtal BASIC 2.2 does not conform to the memory map of Fig. 1, since it has been relocated to start from 1200H, the scratch-pad starts at 2F00H, auxiliary reserved word tables start at 2F80H and TEXT is at 3500H. It comes supplied with a small machine-code monitor (since the SHARP monitor does not have debugging facilities) and a program for converting existing SHARP BASIC programs into Xtal BASIC. Most SHARP dealers should soon be able to offer it.

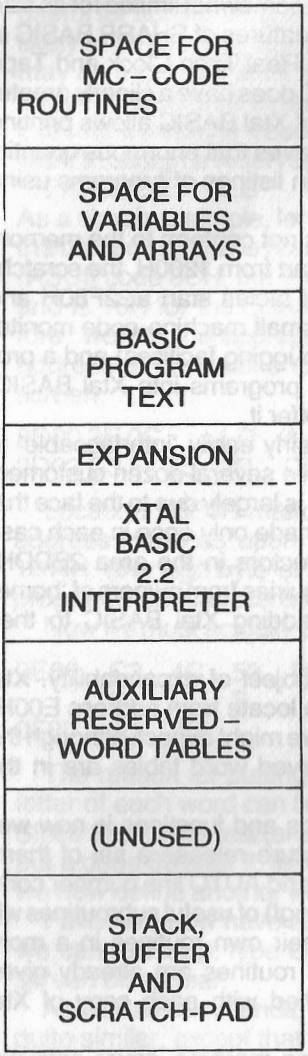
Xtal BASIC 2.2 has been designed to be fairly easily 'interfaceable' to most Z80 systems, and we do, at present, have several dozen customers who have adapted it to their own systems. This is largely due to the face that reference to the required monitor routines is made only once in each case within the Interpreter, using a set of jump vectors in the area 2BDDH-2BFFH. We should therefore welcome any enquiries from owners of 'home-brew' Z80 systems who are interested in adding Xtal BASIC to their machine.

Although I feel that it slightly defeats the object of expandability. Xtal BASIC is now available in EPROM (2708s) to locate from address E00H-FCFFH. In this case, TEXT is from 1000H, as we might expect, although the scratch-pad locations and the auxiliary reserved word tables are in the same places.

Finally, the development of extra commands and functions is now well under way (at last!), and shall in the near future release a set of them, including the much sought-after RENUMBER and AUTO line number commands. At the same time, a long list (it is very long!) of useful subroutines will be made available, to help users develop their own routines in a more compact and efficient manner. A few useful routines are already given within the Operating Manual which is supplied with each copy of Xtal BASIC.

Time to get back to that in-house washing-machine driver software (anyone interested in WASH, SPIN, RINSE AND SLOOSH?). . . .

FIGURE 1
SIMPLIFIED MEMORY MAP OF Z80 SYSTEM INCORPORATING XTAL BASIC 2.2



Limit of RAM Space

TOPRAM (loc stored at (0 C92H))

TXTUNF (loc stored at (0 CB7H))

TEXT (loc stored at (0 C8CH) and (1283H))

2D00H

1000H

0E80H

0E00H

0C80H

TABLE 1 – XTAL BASIC 2.2 SPECIFICATION

COMMANDS

CALL	CLEAR	CLOAD	CONT	CSAVE	READ	DATA	RESTORE	DEF	FN
DIM	EDIT	END	FOR	TO	STEP	NEXT	GOSUB	RETURN	GOTO
IF	THEN	INPUT	LIST	MON	NEW	ON	OUT	POKE	POP
PRINT	REM	RUN	SPEED	STOP	WAIT				

FUNCTIONS

ABS	ASC	ATN	CHR$	CMD$	COS	ERR	EXP	INCH	INP
KBD	LEN	LEFT$	LOG	MID$	PEEK	PI	POS	RND	RIGHT$
SGN	SIN	SIZE	SIZE$	SQR	SPC	STR$	TAB	TAN	VAL

VARIABLES:– Names must start with a letter, but can be up to any length. First two characters used to distinguish one variable from another. Strings of up to 255 characters, also multi-dimensional arrays and string arrays.

NUMBERS:– Floating-point accuracy of 6 sig. figs, with exponent range –38 to 38

OPERATORS:– Arithmetic: + – * / ** ('to the power of')

Relational: =>< <>> = < =

Arith-logical: AND OR NOT

String: + (concatenation)

CASSETTE COMMANDS:– CSAVE, CLOAD for saving and loading programs. Also CSAVE@and CLOAD@for saving and loading of arrays. All these utilise error and indication. In addition, CLOAD? and CLOAD?@are available for verification of saved programs and data.

SPECIAL COMMANDS: – EDIT – Powerful line editor.

CALL – Machine-code sub-routine call.

MON – Return to monitor under software control.

OUT, INP, WAIT – For control of I/O ports.

INCH, KBD – Single-character input functions.

PRINT@– Allows printing on the VDU screen at specified points.

ON ERR GOTO . . / ON ERR GOSUB . . – Handle errors from BASIC.

CMD$ – Display reserved word and error message names for given code.

Notes: SHARP MZ80K version uses LOAD, SAVE and VERIFY in place of CLOAD, CSAVE and CLOAD?. Full data file handling is supported instead of the CLOAD@and CSAVE@commands. The EDIT command is not supported in this version, since the MZ80K already has an on-screen editing system.

On the NASCOM versions, the power symbol ' ' is replaced by '**', since early NASBUG monitors did not allow typing of the '↑'. MON is replaced by NAS on the NASCOM versions.

PRICES (JUNE 1980):

Xtal BASIC 2.2S on tape for SHARP MZ80K	– 40 pounds.
Xtal BASIC 2.2 on tape for other Z80 systems	– 35 pounds.
EPROM version (8 x 2708)	– 100 pounds

All prices exclude VAT.

AN introduction to Z80 Assembly code programming using the SHARP "Systems" and "FDOS" programs.

After purchasing a computer for the first time most hobbyists will learn BASIC or PASCAL and program their computer entirely in a high level language. At a later stage learning Z80 machine code and Assembly language programming is the next natural progression along the road to mastering all the features of the MZ80K.

This article is the first in a series which will be included in the SHARPSOFT USER NOTES. These articles we believe will help those readers whose knowledge of assembly language is limited to experiment with Z80 machine code.

If you are unfamiliar with the basic principles of the operation of a microprocessor or the representation of data and machine code instructions the we recommend that you read selected sections from the following books :-

1. An Introduction to Microcomputers.

   Volume 0 - The beginners book.
   Volume 1 - Basic concepts.

   by Adam Osborne
   Osborne Associates Inc.

2. Programming the Z80.

   by Rodnay Zaks
   Sybex.

3. The Z80 Microcomputer Handbook.

   by William Barden
   Howard W. Sams and Co.

The series will be based on practical examples which you can run on your computer. Often we will include suggestions for extending the programs presented in the text. Do remember the best way to learn machine code programming is to experiment with simple examples.

We have chosen the MZ80K "Systems" and "FDOS" programs to demonstrate our examples. This choice was deliberate because these programs offer similar facilities for both tape and disc based MZ80K computers. If you have a different Z80 Assembler, for example ZEN, the ex oles can be programmed on your MZ80K but the program preparati and assembly instructions will be different.

We start the series with a few basic assembly language definitions.

1. Labels

Labels are used in assembly language programming to identify memory locations; for example specific locations in a program where an instruction sequence begins, or where a constant is located, or where a variable is to be stored. The use of a label in an assembly language program provides the means to refer to a location mnemonically, by name, rather than by a machine code address. The use of a label also permits an Assembler to automatically update all references to a memory location when its machine code address changes due to changes in the program. A label may also be used synonyomously for a numerical value which is NOT a memory location such a label is called a SYMBOL.

A LABEL consists of one or more characters where the first six characters must be unique. The first character is normally a letter. The characters comma (,), colon (:), semicolon (;), space ( ), plus sign (+) and minus sign (-) must NOT be used in a label.

The reserved names ; A,B,C,D,E,H,L,F,I,R
IX,IY,SP
AF,BC,DE,HL,AF',BC',DE',HL'
C,NC,Z,NZ,M,P,PE,PO

MUST NOT BE USED AS LABELS.

Label selection rules

1. Do not use labels that are the same as the Z80 instruction mnemonoics.
2. Do not use labels that are larger than the Assembler permits.
3. Avoid special characters, non-alphabetic and non-numeric.
4. Do not use labels that could be confused with each other, for example avoid the letters I,O and Z and the numbers 1,0 and 2.
5. When you are not sure if a label is legal, do not use it.

Constants

Constants are used in Z80 assembly language programs to represent numerical values. Although numerical constants are expressed internally in the Z80 microprocessor in binary, an Assembler permits such constants to be expressed in hexadecimal or decimal notation. The postfix symbol "H" is used to denote a hexadecimal number, for example

2AH or 0010H or +01H

The absence of a postfix symbol or the inclusion of the postfix symbol D indicates a decimal value, for example

00 or 0200 or -26 or 25D or +099D

It is good practice to start all hexadecimal numbers with a number, for example the number A0H could be considered by an Assembler to represent a label. However, -A0H will not confuse the Assembler.

Numerical constants are limited in magnitude to 255 if they are represented by a single byte or to 65535 if a double byte.

Expressions

An expression is a sequence of one or more labels and/or numerical constants separated by the arithmetic operations +,- (these symbols represent the operations add and subtract). An expression defines a value in terms of an algebraic expression that is evaluated at assembly time ----, NOT at program execution time. In general, Z80 Assemblers evaluate expressions strictly left to right; there is no operator precedence in expression evaluation and parenthesis may NOT be used to alter the order of operations.

It is good programming practice to avoid expressions within address fields whenever possible. If you must compute an address, comment any unclear expressions, and be sure that the evaluation of the expression never yields a result which is too large, i.e. outside the range 0 to 255 or 0 to 65535.

Assembly statements

Z80 Assemblers recognize three types of statement.

1. Comment statements.
2. Instruction statements.
3. Directive statements -- often called PSEUDO - OPERATIONS.

Each statement is ONE line of assembler code.

Comment statements

Comment statements are included in an assembly language program to provide explanatory information. These statements, in no way, affect the machine code generated by the Assembler.

A comment statement consists of a semicolon as the left most, non-blank character, followed by the commentary,for example

```
; My first program
;
;
; INPUT subroutine
; Written by John.
```

Instruction statements

Instruction statements specify a Z80 assembly language instruction mnemonic or operation code (OP-CODE). The OP-CODE may be preceded by a label and must be followed by an appropriate "OPERAND" corresponding to the instruction addressing mode. The operand may be followed by any commentary desired. Thus, the fields in an instruction statement are :-

(LABEL) (OP-CODE) (OPERAND) (COMMENT)

where the (LABEL) and (COMMENT) fields are optional and the (OPERAND) field is appropriate to the instruction addressing mode.

Most Assemblers use a special symbol, called a delimiter, at the beginning or end of each field. The most obvious delimiter is the space character. The following table lists the standard Z80 Assembler delimiters :-

:	After a label.
' '	Between operation code and addresses.
,	Between operands in the address field.
;	Before a comment.

Directive statements

Directive statements are included in an assembly code program to guide or instruct the Assembler during the assembly process. One obvious example is the instruction END which informs the Assembler that it has reached the end of a program.

Developing Z80 Assembly language programs

The SHARP Z80 assembly language development programs consist of a set of interconnecting programs, these are shown in the figure below.

The text editor is used to prepare an assembly language program. Remember, in BASIC the editor is usually included with the interpreter. Once a program is typed-in using the editor it can be assembled using the Z80 Assembler. The Assembler outputs an assembled form of the original program where the Z80 mnemonic instructions have been translated to machine code. This program can be located anywhere in the MZ80K RAM. We say that the program is relocatable because it does not contain, at this stage, absolute memory addresses. The final stage in the development cycle involves linking the Z80 program. This is done by a loader program which outputs from the loader phase the machine code program which is run as a stand-alone Z80 program. If the program contains errors and will not run correctly it can be tested and the errors removed by a fourth program called the symbolic debugger.

Each of these distinct stages in the development of a Z80 program will be considered in more detail in this and future articles in this series.

The text editor

Preparation of an assembly language program using the text editor is the first step to master. The SHARP text editors are based on a format developed by the Data General Corporation for the NOVA series of minicomputers. In general the tape based systems program editor and the new FDOS editor have the same commands.

To practice with the editor we require a program. Out of interest, and for comparison, we have chosen the graphics display problem given in the FORTH tutorial. This allows machine code run times to be compared with both the BASIC and FORTH times previously reported.

In assembly language there is no unique solution to this problem. Our objective is to see firstly how many different ways we can solve the graphics display problem and secondly which solution runs the fastest.

After loading the tape or disc text editor a program header is displayed on the V.D.U. followed by a *. The editor is now ready to accept commands. These commands are described on page 29 of the Systems Program manual and page Edit-2 of the FDOS manual.

At this stage typing T<CR>, where <CR> denotes the yellow carriage return key is pressed, shows that the edit buffer is empty.

To insert text type I<CR>. The system is now ready to accept text. Enter the following short programs -- each line is terminated with a <CR>. NOTE the layout -- the terminators are :, ' ', ',', and ';'. Use the editor correction commands to correct any typing errors. Assemble, link and run the example

programs. In each case make sure you understand the Z80 instructions. The notes at the end of each example are provided as a guide.

Assembly language programming -- EXAMPLE 1.
Graphics display using two loops and subtraction with carry.

```
; EXAMPLE 1
LD A,0
OUTER:LD BC,D000H
LOOP:LD HL,D3FFH
LD (BC),A
INC BC
SBC HL,BC
JP NZ,LOOP
INC A
JP NZ,OUTER
JP 1200H
END
```

Assembler output.

```
01    0000
02    0000 3E00                    LD    A,0
03    0002
04    0002 0100D0     OUTER:       LD    BC,D000H
05    0005
06    0005 21FFD3     LOOP:        LD    HL,D3FFH
07    0008 02                      LD    (BC),A
08    0009 03                      INC   BC
09    000A ED42                    SBC   HL,BC
10    000C C20500                  JP    NZ,LOOP
11    000F 3C                      INC   A
12    0010 C20200                  JP    NZ,OUTER
13    0013 C30012                  JP    1200H
14    0016                         END

LOOP      0005      OUTER          0002
```

Link loader output.

```
LINKING EXAMPLE1.RB
   TOP ASM.BIAS $4600
   END ASM.BIAS $4616
SAVE EXAMPLE1.OBJ
   LOADING ADDRESS $4600
   EXECUTE ADDRESS $4600
   BYTESIZE $0016
```

Notes

1. Assembled with FDOS.
2. Register A holds the display character code -- 0 to 255.
3. Register pair BC initially holds the start of the display RAM (D000H)--- this is incremented each time round the inner loop.

4. Register pair HL holds the address of the end of the display RAM (D3FFH). Hence the LOOP is terminated when HL-BC --> 0.
5. RUN time roughly 6.3 seconds.
6. JP 1200H - warm start to FDOS -- can be replaced with JP 0000H

Assembly language programming -- Example 2.
Graphics display using multiple register transfer.

```
; EXAMPLE 2
LD A,0
OUTER:LD HL,D000H
LD BC,D100H
LD E,0
LOOP1:LD (HL),A
LD (BC),A
INC HL
INC BC
INC E
JP NZ,LOOP1
LD HL,D200H
LD BC,D300H
LOOP2:LD (HL),A
LD (BC),A
INC HL
INC BC
INC E
JP NZ,LOOP2
INC A
JP NZ,OUTER
JP 1200
END
```

Assembler output.

```
01    0000
02    0000 3E00            LD    A,0
03    0002 2100D0  OUTER:  LD    HL,D000H
04    0005 0100D1          LD    BC,D100H
05    0008 1E00            LD    E,0
06    000A 77      LOOP1:  LD    (HL),A
07    000B 02              LD    (BC),A
08    000C 23              INC   HL
09    000D 03              INC   BC
10    000E 1C              INC   E
11    000F C20A00          JP    NZ,LOOP1
12    0012 2100D2          LD    HL,D200H
13    0015 0100D3          LD    BC,D300H
14    0018 77      LOOP2:  LD    (HL),A
15    0019 02              LD    (BC),A
16    001A 23              INC   HL
17    001B 03              INC   BC
18    001C 1C              INC   E
19    001D C21800          JP    NZ,LOOP2
20    0020 3C              INC   A
21    0021 C20200          JP    NZ,OUTER
```

```
22   0024 C30012              JP   1200H
 23   0027
 24   0027                      END

 LOOP1     000A LOOP2     0018 OUTER     0002

 Link loader output.

 LINKING EXAMPL2.RB
   TOP ASM,BIAS $4600
   END ASM,BIAS $4627
SAVE EXAMPLE2.OBJ
  LOADING ADDRESS $4600
  EXECUTE ADDRESS $4600
  BYTESIZE $0027
```

Notes.

1. Assembled using FDOS.
2. Two screen locations set per loop.
3. RUN time roughly 2.7 seconds.

Assembly language programming -- Example 3.
Graphics display using block move instructions.

```
; EXAMPLE 3
LD A,0
OUTER:LD C,0
LD HL,CFFFH
LOOP:INC HL
LD (HL),A
INC C
JP NZ,LOOP
LD HL,D000H
LD DE,D100H
LD BC,0100H
LDIR
LD HL,D000H
LD DE,D200H
LD BC,0100H
LDIR
LD HL,D000H
LD DE,D300H
LD BC,0100H
LDIR
INC A
JP NZ,OUTER
JP 1200H
END
```

Assembler output.

```
01    0000
02    0000
03    0000 3E00                      LD    A,0
04    0002
05    0002 0E00              OUTER:  LD    C,0
06    0004 21FFFC                    LD    HL,CFFFH
07    0007 23                LOOP:   INC   HL
08    0008 77                        LD    (HL),A
09    0009 0C                        INC   C
10    000A C20700                    JP    NZ,LOOP
11    000D 2100D0                    LD    HL,D000H
12    0010 1100D1                    LD    DE,D100H
13    0013 010001                    LD    BC,0100H
14    0016
15    0016 EDB0                      LDIR
16    0018 2100D0                    LD    HL,D000H
17    001B 1100D2                    LD    DE,D200H
19    0021
20    0021 EDB0                      LDIR
21    0023 2100D0                    LD    HL,D000H
22    0026 1100D3                    LD    DE,D300H
23    0029 010001                    LD    BC,0100H
24    002C
25    002C EDB0                      LDIR
26    002E 3C                        INC   A
27    002F C20200                    JP    NZ,OUTER
28    0032 C30012                    JP    1200H
29    0035                           END

LOOP 0007 OUTER      0002
```

Link loader output.

```
LINKING EXAMPLE3.RB
  TOP ASM.BIAS $4600
  END ASM.BIAS $4635
SAVE EXAMPLE3.OBJ
  LOADING ADDRESS $4600
  EXECUTE ADDRESS $4600
  BYTERSIZE $0035
```

Notes.

1. Assembled using FDOS.
2. Program fills first 256 bytes with display code then copies block to the last three pages of the display RAM using the block move instruction LDIR.
3. RUN time roughly 3 seconds.

Assembly language programming -- Example 4.
Graphics display using the stack pointer set to display RAM.

```
; EXAMPLE 4
LD A,0
OUTER:LD B,A
LD C,A
LD E,0
LD HL,D3FFH
LD SP,HL
LOOP1:PUSH BC
INC E
JP NZ,LOOP1
LOOP2:PUSH BC
INC E
JP NZ,LOOP2
LOOP3:PUSH BC
INC E
JP NZ,LOOP3
LOOP4:PUSH BC
INC E
JP NZ,LOOP4
INC A
JP NZ,OUTER
JP 0000H
END
```

Assembler output.

```
01    0000
02    0000
03    0000
04    0000
05    0000
06    0002 3E00                    LD    A,0
07    0004 47          OUTER:      LD    B,A
08    0005 4F                      LD    C,A
09    0006 1E00                    LD    E,0
10    0008 21FFD3                  LD    HL,D3FFH
11    000B F9                      LD    SP,HL
12    000C C5          LOOP1:      PUSH  BC
13    000D 1C                      INC   E
14    000E C20C00                  JP    NZ,LOOP1
15    0011 C5          LOOP2:      PUSH  BC
16    0012 1C                      INC   E
17    0013 C21100                  JP    NZ,LOOP2
18    0016
19    0016 C5          LOOP3:      PUSH  BC
20    0017 1C                      INC   E
21    0018 C21600                  JP    NZ,LOOP3
22    001B C5          LOOP4:      PUSH  BC
23    001C 1C                      INC   E
24    001D C21B00                  JP    NZ,LOOP4
```

```
25   0020 3C                          INC  A
26   0021 C20400                      JP   NZ,OUTER
27   0024
28   0024 F9                          LD   SP,HL
29   0025 C30000                      JP   0000H
30   0028                             END

LOOP1     000C LOOP2     0011 LOOP3     0016
      LOOP4     001B OUTER     0004
```

Link loader output.

```
LINKING EXAMPLE4.RB
  TOP ASM.BIAS $4600
  END ASM.BIAS $4629
SAVE EXAMPLE4.OBJ
  LOADING ADDRESS $4600
  EXECUTE ADDRESS $4600
  BYTESIZE $0029
```

Notes.

1. Assembled using FDOS.
2. Top of stack set to the end of the display RAM.
3. RUN time roughly 3 seconds.

Exercises

The four examples each demonstrate different techniques for solving the graphics display problem. Notice in all cases absolute jumps were used. Try to rewrite the examples using relative jumps. A second suggestion for an exercise could be, in example 1, to replace the SBC instruction with a sequence that compliments the contents of BC and adds it to HL. Do try some experiments -- this is the best way to learn Z80 machine code programming.

Run times

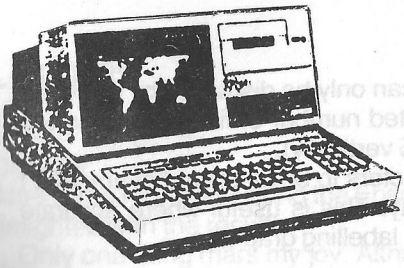
It is interesting to compare the run times obtained from the example exercises, these are:-

```
Example 1    =    6.3 seconds
Example 2    =    2.7 seconds
Example 3    =    3.0 seconds
Example 4    =    3.3 seconds
```

Also

```
FORTH        =    14 seconds
BASIC        =    20 to 30 minutes  !!!!!!!!!!!!!!!!
```

if you can improve on the fastest time write to us with your solution and we will publish it in our next issue.



MZ-80B NEWS

THE LOW-DOWN ON THE MZ–80B

In December, 1981, Personal Computer World Magazine reviewed the Sharp MZ–80B. In the opening paragraph the reviewer says, "My impression after a few hours with the MZ–80B is that this machine is the one that Sharp's design team wanted to make when they worked on the 80K".

In conclusion he says, "It's a versatile and likeable machine which shows evidence of much thought in its design, particularly in the area of user conveniences".

In between it says a great deal about the software, the keyboard, the operating system, the languages, the documentation, its expandibility and potential.

To read the report in full you can get a copy of the magazine from the Back Numbers Department, PCW, 14 Rathbone Place, London W.1. Price £1.25.

****GRAPHIC UTILITY PROGRAMS FOR THE MZ—80B****

CHARACTER STRING UTILITY

Using this facility, it is possible to draw both vertical and horizontal character strings into either of the graphic RAM pages.

This facility is useful for adding textual and symbolic information to graphs and diagrams, such that both graphical details and text can be copied simultaneously on the printer.

The full MZ—80B character set (including graphic and reverse symbols) is available with this facility, and the character size (8 x 8 dots) is the same size as for characters displayed using the character VRAM in 40 CHAR/LINE mode.

Without using this facility, characters can only be displayed by using the character VRAM, which results in a limited number of fixed positions (40 character positions across the screen, 25 vertically).

This additional facility allows for characters to be displayed at any (X,Y) position within the bounds of the display. This is useful where accurate positioning of characters is required (e.g. labelling graphs).

GRAPHIC ROTATE/PRINT UTILITY

This facility will enable users to print a copy of an entire frame of graphic VRAM rotated through 90 degrees, using the MZ-80 P5 printer.

The utility complements the existing 'COPY/P' facility in BASIC (which copies display frames across the width of the printer paper in the same orientation as they appear on the screen).

This additional facility causes the printer to copy display frames 'sideways' along the length of the paper.

There is an option for copying the display rotated either clockwise or anti-clockwise.

Another option allows for the user to specify which graphic page(s) are to be copied. The user may select either graphic page 1, graphic page 2, or both graphic pages together (i.e. pages are 'mixed' together).

The printed copy is centred across the range of printer head travel, so it is possible for the copy to appear centred on various widths of paper.

LETTERS

Dear Sir,
After three months of using the Epson Mx80 that you supplied me with, I am delighted with the results.

Only one thing mars my joy. Although the printer interfaces well with the SP5025 BASIC, I am unable to persuade it to talk to my XTAL BASIC 2.2s. My main problem is that while I understand that ports FF and FE are used for the printer output, I do not know what signals are expected there. I would be extremely grateful, therefore, if you could supply me with either a routine which will work with Xtal BASIC or even sufficient details of the SP5025 routine for me to 'cobble one up'.

By way of recompense, may I offer the attached BASIC routine for use in the next user notes. I hope it is reasonably self explanatory, being nothing more than a simple crash-proof entry system for validating character, numeric and date inputs. There is nothing startling clever about it but perhaps some readers may find it helpful.

Harvey Platter
Devon

CONTROLLED INPUT SUBROUTINE

BY Harvey Platter

```
32000 REM  'X' IS HORIZONTAL SCREEN POSITION,  'Y' IS VERTICAL SCREEN POSITION
32001 REM  'Z' IS INPUT STRING LENGTH,   'ZZ' IS COUNTER BASED ON 'Z'
32002 REM  'DD$','MM$','YY$', ARE DATE SUB-STRINGS, 'DD','MM','GY'ARE NUMERIC
32003 REM   EQUIVALENTS ††
32004 REM 'BD$' AND 'AD$' ARE NUMERIC PROMPTS, 'PD$' & 'ND$' ARE INPUT RESULTS
32005 REM  ALL INPUTS ARE RETURNED TO MAIN PROGRAM IN 'ST$' OR 'ST'
32010 POKE4465,X:POKE4466,Y:RETURN
32020 ST$="":GOSUB32010
32021 XX$="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
32022 BL$="                                        "
32025 REM       ††††       ALPHA-NUMERIC INPUT      †††††
32030 PR$=LEFT$(XX$,Z):PRINTPR$:GOSUB32570:GOSUB32010:FORZZ=1TOZ
32040 GETA$:IFA$=""GOTO32040
32050 IFA$="†"THENZZ=Z:GOTO32075
32060 IF(ASC(A$)<32)+(ASC(A$)>90)THEN32040
32070 PRINTA$;:ST$=ST$+A$:NEXT
32075 GOSUB32520:IFQ$="N"GOTO32020
```

```
32080 RETURN
32090 GETA$:IFA$=""THEN32090
32091 IF(A$="¥")¥(CH=0)THENCK=1:RETURN
32092 A=ASC(A$):IF(A<48)+(A>57)THEN32090
32095 CH=1:RETURN
32099 REM       ¥¥¥¥      DATE (DD/MM/YY) INPUT   ¥¥¥¥¥
32100 CP=0:ST$="":ST=0:GOSUB32010:PRINT"dd/mm/yy":GOSUB32570:GOSUB32010
32105 CK=0:CH=0:DD$="":MM$="":YY$=""
32110 FORZZ=1TO2:GOSUB32090:IFCK=1THENZZ=2:GOTO32225
32115 PRINTA$;:DD$=DD$+A$:NEXT
32120 XL=PEEK(4465):POKE4465,XL+1:FORZZ=1TO2:GOSUB32090:PRINTA$;:MM$=MM$+A$
32130 NEXT:XL=PEEK(4465):POKE4465,XL+1:FORZZ=1TO2:GOSUB32090
32140 PRINTA$;:YY$=YY$+A$:NEXT
32150 DD=VAL(DD$):MM=VAL(MM$):GY=VAL(YY$)
32160 IF(DD<1)+(DD>31)+(MM<1)+(MM>12)THENGOSUB32500:GOTO32100
32170 IF(MM=2)+(MM=4)+(MM=6)+(MM=9)+(MM=11)THENCP=1
32180 IF(CP=1)¥(DD>30)THENGOSUB32500:GOTO32100
32190 IF(MM=2)¥(DD>29)THENGOSUB32500:GOTO32100
32200 IF(GY/4<>INT(GY/4))¥(MM=2)¥(DD>28)THENGOSUB32500:GOTO32100
32210 ST$=DD$+"/"+MM$+"/"+YY$:ST=(DD¥10000)+(MM¥100)+GY:GOSUB32520
32220 IFQ$="N"GOTO32100
32225 IFCK=1THENST=0:ST$="00/00/00":GOSUB32010:PRINTST$
32230 RETURN
32235 REM       ¥¥¥¥      NUMERIC INPUT  ¥¥¥¥¥¥
32300 CH=0:CK=0:ST$="":ST=0:GOSUB32010:PRINTBD$;".";AD$:GOSUB32570
32305 GOSUB32010:FORZZ=1TOLEN(BD$)
32310 GOSUB32090:IFCK=1THENZZ=LEN(BD$):GOTO32355
32315 PRINTA$;:PD$=PD$+A$:NEXT:ST$=PD$:XL=PEEK(4465):POKE4465,XL+1
32320 ST$=ST$+"."
32330 FORZZ=1TOLEN(AD$):GOSUB32090:PRINTA$;:ND$=ND$+A$:NEXT
32340 ST$=ST$+ND$:ST=VAL(ST$):GOSUB32520
32350 IFQ$="N"GOTO32300
32355 IFCK=1THENST=0:ST$="000.00":GOSUB32010:PRINTST$
32360 RETURN
32499 END
32500 POKE4465,0:POKE4466,23:PRINT"    Invalid date : Please re-enter     "
32510 FORI=1TO5:MUSIC"A":NEXT:RETURN
32520 POKE4465,0:POKE4466,23:PRINT"  Is data correct?    Type 'Y' or 'N'  "
32530 GETQ$:IFQ$=""GOTO32530
32540 IFQ$="N"THENRETURN
32550 IFQ$="Y"THENRETURN
32560 GOTO32530
32570 POKE4465,0:POKE4466,23:PRINT" Enter data in format shown '¥' to end "
32580 RETURN
```

NOTES

Dear Sir,
I enclose my cheque for £7.50 in respect of the 1982 subscription.

I have read with interest the last issue of the user notes and with particular interest the letter and programmes etc from other users.

However whilst the various utilities and suggestions from other users are useful I have not yet seen a suggestion or a programme to overcome, what I consider to be a glaring omission in "Sharp Basic", ie the absence of a Using "xxx . xx" statement. It would be interesting to know for what purpose the "User" purchased his or her computer as from the user notes many of the users seem to be more interested in games or musical orientated programmes which are really not more than gimmicks and are very boring after one or two run throughs.

Is no other amateur home "user" utilising the facilities of the computer for personal business and financial affairs, for example. In many cases it is difficult for the average person to check whether their income tax deductions are correct, however by entering answers to questions, the computer can very quickly verify the position and in many cases with 100% accuracy if absolutely accurate figures are entered. Whilst by its very nature even a simple tax programme would be fairly long, would this be of interest for inclusion in a future edition of User Notes. If so I will be happy to forward such a programme. In addition to this providing there are no major changes in the basic tax law in the budget it is fairly easy to update the programme from year to year. I would mention that a very high proportion of the tax calculations made by the Revenue are incorrect and many of these are in their favour.

Has any "user" devised a sub-routine programme to overcome the absence of the Using "# # # . # # ") function in Sharp Basic? has any other "user" any of their own home use programmes which may be of interest other than games?

W. H. Fisher
Liverpool 12

Dear Sir,
I enclose a cheque for £7.50 against 1982 subscription to User Notes. I am looking forward to receiving the fig-FORTH tapes as I have already bought a Knights FORTH only to discover that it is completely non-standard. As the "manual" was also very poor it was necessary for me to look elsewhere for teaching material. (That's how I discovered fig-FORTH and other standard FORTHs and how I learnt an expensive lesson).

I have purchased STARTING FORTH by BRODIE and the following comments may prove useful to others.

1. It does teach a standard FORTH from the ground upwards.

2. It becomes quite advanced.
3. It teaches very few programmes and since the skill of programming in FORTH is not easily acquired, this is a serious deficiency.
4. I find normal flow diagrams difficult to translate into FORTH, but only 3 pages are devoted to this topic – another loss.
5. It is worth its cost – about £13.

J. R. Burton
Hampshire

Dear Sir,
I have recently received the copy of 'Sharpsoft User Notes' issue No. 3.

I must congratulate you on this excellent publication. I am a member of one other club and was a member of two other clubs associated with another microcomputer system, by comparison you are definitely top of the league.

I found your 'user notes' informative and well presented, I particularly appreciated your approach to the reader as someone who is interested in learning and not someone to be talked 'down to'. The software you presented was excellent, the right balance of informative and serious but always interesting.

I have loaded (by hand) the 'Utility programs' with the necessary adjustments for the MZ80B, these have been linked together which with 64K of RAM is acceptable and still leaves sufficient memory for programs to be developed. One error, although everybody else has probably already spotted it, is line 60030 of the 'Renumber/TP' program, AH should equal 72 not 721. I have enclosed my listing should this be of any benefit.

Could you please let me know what POKE 10167,1 does at it seems unnecessary on the MZ80B, indeed it may be unwise to do so, but may have an equivalent. I would be interested to know if there are any tricks yet discovered to do the following on the MZ80B;

1. Make a program load and run automatically from the BASIC common LOAD.
2. Make a program unlistable, before and after it has 'RUN'.
3. Link programs together using the 'Tape BASIC'.

I have the following information for MZ80B owners, if you have not yet managed to copy the 'Tape BASIC', you may do so by moving or rather copying the Monitor and BASIC from address 8000 hex and if you wish to enter BASIC directly change the bytes at 80AE to JP 1220 hex (C3, 20, 12) before saving the machine code file from 8000H to C920H. SHARP have conceded a jump to a routine which is overwritten by BASIC on initialisation, but the method above works perfectly (see enclosed listings). I have

enclosed some other information on the SB5510 BASIC, text start, tokens used etc.

You will find enclosed my personal cheque to the value £7.50 for my subscription to the user notes. As you are aware I would not be able to load the M80K FORTH tape on my machine, however if you have FORTH available for the MZ80B I would be most interested, and prepared to pay any extra.

Once again thank you for your superb 'user notes' and I wish you every success in the future.

R. A. Wymark
Birmingham

POKE 10167,1 is used in the MZ-80K to enable the PEEK statement allowing the contents of RAM, where BASIC is stored, to be read. This form of protection does not apply on the MZ-80B and hence POKE 10167,1 may be removed from our utilities.

Ed

BASIC COPY ROUTINE.

Code is position independent,I use $D000.

```
  $D000 21 00 00 ; LD HL,0000
  $D003 11 00 80 ; LD DE,8000
  $D006 01 20 49 ; LD BC,4920
  $D009 ED B0    ; LDIR
  $D00B C3 00 00 ; JP,0000

Now if you wish to jump directly into BASIC
change from $80AE to $80B0 to the following-
$80AE C3 20 12 ; JP,1220

Simply save from $8000 to $C920,no jump address is needed.
```

FUNCTIONS.
The following functions seem to be represented by the tokens shown.

```
128- 0 = RND(       128- 1 = SIN(       128- 2 = COS(       128- 3 = TAN(
128- 4 = ATN(       128- 5 = EXP(       128- 6 = INT(       128- 7 = LOG(
128- 8 = LN(        128- 9 = ABS(       128- 10 = SGN(      128- 11 = SQR(
128- 12 =
```

All the tokens have been shown as decimal values.

BASIC TEXT STARTS AT DECIMAL 20572.

BASIC uses the following 'TOKENS' in the source text.

```
128- 128 = REM      128- 160 = USR(     128- 192 = ↰        128- 224 = LEFT$(
128- 129 = DATA     128- 161 = WOPEN    128- 193 = ↰        128- 225 = RIGHT$(
128- 130 = DEF KEY( 128- 162 = ROPEN    128- 194 = ↰        128- 226 = MID$(
128- 131 = ↰        128- 163 = CLOSE    128- 195 = ><       128- 227 = LEN$(
128- 132 = READ     128- 164 = MON      128- 196 = <>       128- 228 = CHR$(
128- 133 = LIST     128- 165 = LIMIT    128- 197 = =<       128- 229 = STR$(
128- 134 = RUN      128- 166 = CONT     128- 198 = <=       128- 230 = ASC(
128- 135 = NEW      128- 167 = GET      128- 199 = =>       128- 231 = VAL(
128- 136 = PRINT    128- 168 = INP@     128- 200 = >=       128- 232 = PEEK(
128- 137 = LET      128- 169 = OUT@     128- 201 = ↰        128- 233 = TAB(
128- 138 = FOR      128- 170 = CURSOR   128- 202 = >        128- 234 = SPACE$(
128- 139 = IF       128- 171 = SET      128- 203 = <        128- 235 = SIZE
128- 140 = THEN     128- 172 = RESET    128- 204 = ↰        128- 236 = ↰
```

```
128- 141 = GOTO      128- 173 = LINE      128- 205 = ↴         128- 237 = ↴
128- 142 = GOSUB     128- 174 = BLINE     128- 206 = ↴         128- 238 = ↴
128- 143 = RETURN    128- 175 = CONSOLE   128- 207 = ↴         128- 239 = STRING$(
128- 144 = NEXT      128- 176 = GRAPH     128- 208 = ↴         128- 240 = POINT(
128- 145 = STOP      128- 177 = POSITION  128- 209 = ↴         128- 241 = CHARACTE
R$(

128- 146 = END       128- 178 = PATTERN   128- 210 = ↴         128- 242 = CSR
128- 147 = ↴         128- 179 = AUTO      128- 211 = ↴         128- 243 = POS
128- 148 = ON        128- 180 = ↴         128- 212 = ↴         128- 244 = ↴
128- 149 = LOAD      128- 181 = IMAGE/P   128- 213 = ↴         128- 245 = ↴
128- 150 = SAVE      128- 182 = COPY/P    128- 214 = ↴         128- 246 = ↴
128- 151 = VERIFY    128- 183 = PAGE/P    128- 215 = ↴         128- 247 = ↴
128- 152 = POKE      128- 184 = ↴         128- 216 = ↴         128- 248 = ↴
128- 153 = DIM       128- 185 = BOOT      128- 217 = ↴         128- 249 = ↴
128- 154 = DEF FN    128- 186 = FLIST     128- 218 = ↴         128- 250 = ↴
128- 155 = INPUT     128- 187 = CHANGE    128- 219 = ↴         128- 251 = ↴
128- 156 = RESTORE   128- 188 = ↴         128- 220 = ↴         128- 252 = ↴
128- 157 = CLR       128- 189 = ↴         128- 221 = ↴         128- 253 = ↴
128- 158 = MUSIC     128- 190 = REW       128- 222 = TO        128- 254 = ↴
128- 159 = TEMPO     128- 191 = FAST      128- 223 = STEP      128- 255 = ↴


The symbol '↴' is used as a carriage return abreviation, although this is
not necessarily it's meaning as a token.

The mathematic symbols ^ * / + - are represnted by their ASCII values.
The following are represented by a single byte:-
138 = <>      138 = >       139 = <       158 = TO      168 = PEEK(
An array variable parameter is prefixed with 165 ie AC(12) = 'A'C'165'12')'.
All lines are terminated by a carriage return decimal 13.

60000 REM LINKER FOR UTILITIES.
60010 CLR:PRINT CHR$(6)
60020 PRINT"Hexadecimal dump=1":PRINT"Variable table=2"
60030 PRINT"String search=3":PRINT"Renumber=4":PRINT"Exit=5":PRINT
60040 INPUT"Function required=";A$
60050 A1=VAL(A$):IF (A1>=5)+(A1=0) THEN END
60060 ON A1 GOTO 60100,60400,60900,61400
60070 END
60100 PRINT CHR$(6)
60120 L$="──────────────────────────────────────────"
60130 H$="0123456789ABCDEF":K1=4096:K2=256:K3=16
60140 DATA 4096,256,16,1
60145 DIM A(4):U=48:X=7:Y=9
60160 INPUT"   Hex dump address   ";A$:IF LEN(A$)<4 THEN 60160
60170 A$=LEFT$(A$,3)+"0":GOSUB 60330:PRINT:PRINT
60180 PRINT/P:PRINT/PL$;" Hex Dump ";L$
60190 PRINT"Addr = value in hex":PRINT
60200 PRINT/P"Addr = value in hex":PRINT/P
60210 AH=A
60220 FOR I=1 TO 16:A=AH:GOSUB 60290:PRINTA$+" ";
60230 PRINT/PA$+" ";
60240 FOR J=0 TO 15:A=AH+J:A=PEEK(A):GOSUB 60290:PRINT RIGHT$(A$,2);
60250 PRINT/P RIGHT$(A$,2)+" ";
60260 NEXT J:AH=AH+J:PRINT
60270 PRINT/P
60280 NEXT I
60285 GET KB$:IF KB$="" THEN 60285
60286 GOTO 60000
60290 RESTORE:A$="":M=4
60300 READ P:FOR Z=1 TO 16:IF A-Z*P<0 THEN 60310
60305 NEXT Z
60310 A$=A$+MID$(H$,Z,1):A=A-(Z-1)*P:M=M-1:IF M>0 THEN 60300
60320 RETURN
60330 FOR Z=1 TO 4:A(Z)=0:NEXT Z:FOR Z=1 TO 4
60340 A(Z)=ASC(MID$(A$,Z,1))-U:IF A(Z)>Y THEN A(Z)=A(Z)-X
60350 NEXT Z:A=K1*A(1)+K2*A(2)+K3*A(3)+A(4):RETURN
60400 PRINT CHR$(6)
```

```
60410 PRINT"————————Table/TP————————"
60420 PRINT/P"————————Table/TP————————"
60430 DIM US$(255):A=20572:C=0:LP=0:Y=60000
60440 FOR I=1 TO 100:US$(I)="":NEXT I:GOSUB 60760
60450 PRINT:PRINT "Line numbers processed":PRINT
60460 PRINT/P:PRINT/P"Line numbers processed":PRINT/P
60470 D=PEEK(A)+256*PEEK(A+1)
60480 LN$=STR$(PEEK(A+2)+PEEK(A+3)*256):A=A+3
60490 IF VAL(LN$)<>Y GOTO 60540
60500 PRINT:FOR I=1 TO 39:PRINT"-";:NEXT I:PRINT
60510 PRINT"Variables---Line numbers":PRINT
60520 PRINT/P:FOR I=1 TO 79:PRINT/P"-";:NEXT I:PRINT/P
60530 PRINT/P"Variables---Line numbers":PRINT/P:GOTO 60700
60540 IF LP>10 THEN PRINT:PRINT/P:LP=0:GOTO60570
60550 PRINT" ";LN$;:PRINT/P" ";LN$;
60560 LP=LP+1
60570 A$="":A=A+1:IF A>=D THEN 60470
60580 IF PEEK(A)=34 THEN 60720
60590 IF (PEEK(A)=128)*(PEEK(A+1)=128) THEN A=D:GOTO 60470
60600 IF (PEEK(A)>90)+(PEEK(A)<65) THEN 60570
60610 A$=A$+CHR$(PEEK(A)):A=A+1:IF A=D THEN 60660
60620 P=PEEK(A):IF (P>47)*(P<91)*(P<>61)*(P<>58) THEN 60610
60630 IF PEEK(A)=36 THEN A$=A$+"$"
60640 IF LEN(A$)<3 THEN A$=A$+" ":GOTO 60640
60650 Z$=MID$(A$,3,1):Z=ASC(Z$):IF Z<>36 THEN A$=LEFT$(A$,2)
60660 IF LEN(A$)<3 THEN A$=A$+" ":GOTO 60660
60680 FOR X=1 TO C:IF LEFT$(US$(X),3)=A$ THEN 60740
60690 NEXT X:C=C+1:US$(C)=A$+LN$:GOTO60570
60700 FOR X=1 TO C:PRINT US$(X)
60710 PRINT/P US$(X):NEXT X
60715 GET KB$:IF KB$="" THEN 60715
60717 GOTO 60000
60720 A=A+1:IF (PEEK(A)<>34)*(A<>D-1) THEN 60720
60730 GOTO 60600
60740 IF LN$=RIGHT$(US$(X),LEN(LN$)) THEN 60570
60750 US$(X)=US$(X)+" "+LN$:GOTO 60570
60760 INPUT"From? ";LN$:IF ASC(LN$)=65 THEN Y=60000:RETURN
60770 X=VAL(LN$):INPUT" To? ";LN$:IF ASC(LN$)=69 THEN Y=60000:GOTO 60790
60780 Y=VAL(LN$)
60790 LN$=STR$(PEEK(A+2)+PEEK(A+3)*256):IF VAL(LN$)=X THEN RETURN
60800 IF VAL(LN$)=Y THEN PRINT"Error from= ";X;" To= ";Y:GOTO 60715
60810 A=PEEK(A)+256*PEEK(A+1):GOTO 60790
60900 PRINT CHR$(6)
60910 PRINT"———— Search/TP ————"
60920 PRINT/P"————Search/TP————"
60930 OM=0:L2=20571:Y=60000:LP=0
60940 DIM L1(255):L=L2
60950 INPUT"Command= ";NL$:PRINT
60960 PRINT/P"Command= ";NL$
60970 IF ASC(NL$)=70 THEN GOSUB 61020
60980 IF ASC(NL$)=82 THEN GOSUB 61140
60990 IF ASC(NL$)=66 THEN GOSUB 61300
61000 IF ASC(NL$)=69 THEN GOTO 60000
61010 L=L2:OM=0:GOTO 60950
61020 FOR I=0 TO 255:L1(I)=0:NEXT I
61030 INPUT"String ?";P1$:IF P1$="" THEN 61030
61040 PRINT/P"String= ";P1$
61050 L=L+4:LN=256*PEEK(L)+PEEK(L-1):IF LN>Y THEN 61110
61060 GOSUB 61220:IF CH=13 THEN 61050
61070 L1(OM)=LN:OM=OM+1:IF OM>255 THEN 61090
61080 GOTO 61050
61090 PRINT" Too many first 256 output"
61100 PRINT/P"Too many first 256 output":PRINT/P
61110 FOR I=0 TO 255:IF L1(I)<>0 THENPRINT L1(I);:PRINT/P L1(I);:LP=LP+1
61120 IF LP>10 THEN PRINT:LP=0
61130 NEXT I:PRINT:PRINT/P:RETURN
61140 INPUT "String-1 ";:P1$:INPUT"String-2 ";P2$
61150 PRINT/P"String-1 =";:P1$;"String-2 =";P2$
61160 IF LEN(P1$)=LEN(P2$) THEN 61180
61170 PRINT"Unequal lengths ":PRINT/P"Unequal lengths ":GOTO 61140
```

```
61180 L=L+4:LN=256*PEEK(L)+PEEK(L-1):IF LN>Y THEN RETURN
61190 GOSUB 61220:IF CH=13 THEN 61180
61200 FOR I=1 TO LEN(P2$):POKE(L+I-1),ASC(MID$(P2$,I,1)):NEXT I
61210 L=L+LEN(P2$)-1:GOTO 61190
61220 J=-1:L=L+1:CH=PEEK(L):IF CH=13 THEN RETURN
61230 IF CHR$(CH)=LEFT$(P1$,1) THEN 61250
61240 GOTO 61220
61250 FOR I=1 TO LEN(P1$)
61260 IF CHR$(PEEK(L+I-1))=MID$(P1$,I,1) THEN 61280
61270 J=1
61280 NEXT I:IF J=1 THEN 61220
61290 RETURN
61300 INPUT"From ?";NL$:IF ASC(NL$)=65 THEN L2=20571:Y=60000:RETURN
61310 X=VAL(NL$)
61320 INPUT"To ?";NL$:IF ASC(NL$)=69 THENY=60000:GOTO 61340
61330 Y=VAL(NL$)
61340 A=20572
61350 NL$=STR$(PEEK(A+2)+PEEK(A+3)*256):IF VAL(NL$)=X THEN L2=A-1:RETURN
61360 IF VAL(NL$)=Y THEN PRINT"Err from= ";X;" to= ";Y:L2=20571:Y=60000:RETURN
61370 A=PEEK(A)+256*PEEK(A+1):GOTO 61350
61400 PRINTCHR$(6)
61410 PRINT"--------Renumber/TP--------":PRINT:PRINT
61420 PRINT/P:PRINT/P"--------Renumber/TP--------":PRINT/P
61430 DIM TA(255):AL=92:AH=80:M=0:Q$=""
61440 INPUT"Starting line number=";SL
61450 INPUT"Line number increment=";IN
61460 PRINT/P"Starting line number is ";SL
61470 PRINT/P"Line number increment is ";IN
61480 H=INT(SL/256):L=SL-H*256
61490 D=256*AH+AL:A=PEEK(D+2):B=PEEK(D+3):LN=256*B+A
61500 IF LN=60000 THEN 61560
61510 TA(M)=LN:M=M+1
61520 POKE D+2,L:POKE D+3,H:L=L+IN:IF L>255 THEN L=L-256:H=H+1
61530 IF M>255 THEN PRINT"Line table full!":GOTO61560
61540 AL=PEEK(D):AH=PEEK(D+1):GOTO 61490
61550 REM SECOND PASS
61560 M=M-1:D=20571:PRINT"Number of lines processed is ";M+1
61570 PRINT/P"Number of lines processed is ";M+1
61580 PRINT:PRINT:FOR I=1 TO 39:PRINT"=";:NEXT I:PRINT
61590 PRINT/P:PRINT/P:FORI=1 TO 79:PRINT/P"=";:NEXT I:PRINT/P
61600 PRINT:PRINT:PRINT"Errors":PRINT/P:PRINT/P"Errors":PRINT/P
61610 D=D+4:L=PEEK(D-1)+256*PEEK(D):IF L<59999 THEN 61620
61615 GET KB$:IF KB$="" THEN 61615
61617 GOTO 60000
61620 D=D+1:H=PEEK(D):IF H=13 THEN 61610
61630 IF (H<>141)*(H<>142)*(H<>128) THEN 61620
61640 IF (H=128)*(PEEK(D+1)=141) THEN D=D+1
61650 IF (H=128)*(PEEK(D+1)=142) THEN D=D+1
61660 A=D
61670 D=D+1:H=PEEK(D):IF H=32 THEN 61670
61680 IF (H>47)*(H<58) THEN Q=H-48:Q$=Q$+RIGHT$(STR$(Q),1):GOTO 61670
61690 IF Q$="" THEN 61620
61700 IF H=58 THEN GOSUB 61730:GOTO 61660
61710 GOSUB 61730:IF H=13 THEN 61610
61720 GOTO 61620
61730 Q=VAL(Q$):FOR J=0 TO M:IF TA(J)=Q THEN61770
61740 NEXT J
61750 PRINT"Old line number ";Q;" in line ";L:PRINT"                    does not ex
ist"
61760 PRINT/P"Old line number ";Q;" in line ";L;" does not exist":GOTO 61830
61770 QL=IN*J+SL:QL$=STR$(QL):QL$=RIGHT$(QL$,LEN(QL$))
61780 IF LEN(QL$)>(D-A-1) GOSUB 61840:GOTO 61830
61790 IF LEN(QL$)<(D-A-1) THEN QL$=QL$+" ":GOTO 61790
61800 FOR Z=A+1 TO D-1:Q$=MID$(QL$,Z-A,1):Q=VAL(Q$)+48:POKE Z,Q
61810 IF Q$=" " THEN POKE Z,32
61820 NEXT Z
61830 Q$="":RETURN
61840 PRINT"Change old line number ";Q;" to ";QL;" in new line ";L
61850 PRINT/P"Change old line number ";Q;" to ";QL;" in new line number ";L
61860 RETURN
```

Dear Sir,
Please find enclosed a cheque for £7.50, being the subscription to 82 Sharpsoft User Notes.

Also enclosed is a tape which has on it an article which you might like to include in the next issue. The article describes how to update a MZ-80K to 48K of memory. The article has been written on tape so as to use the Graphic's for figure purposes. I do not have a printer so the output is directed to the screen, so will require the /printer option inserted after the PRINT commands to enable a printout without line numbers.

The article describes the procedure I carried out to update my own MZ-80K, the information to do so coming from the Sharp Service Manual.

I would like to keep Copyright of the article if this is possible, please advise if this is possible, if you decide to use the article.

Now for a request, does anybody know of a supplier of plug/socket which is compatible with the Sharp Bus, as addition of Games Paddles is straight forward if the connector hardware can be found. If I can find a supplier I will write a further article about I/O.

P. E. Parker
GU14 7DT

```
70 PRINT"        SHARP MZ80K Memory Upgrade"
80 PRINT
90 PRINT"        by P.E.Parker
100 PRINT
110 PRINT"       When I first bought my MZ-80K,the 20K version was at the
120 PRINT"limit of my finance,anyway 6K user RAM compared to the 6800 D2,my
130 PRINT"previous micro,seemed more than adequate.So when articles about
140 PRINT"FORTH appear in the user notes,requiring 48K,the owners of SHARP
145 PRINT"MZ-80K computers with less than the maximum memory may like to
150 PRINT"Upgrade their memory to 48K of RAM,and so join in exploring new
160 PRINT"languages.The following article explains how to achieve the
170 PRINT"message '34680 BYTES' after loading SHARP Basic.
180 PRINT
190 PRINT"       This modification is straight forward and should be within
200 PRINT"the capabilities of anyone willing to open the MZ-80K case,but,it
210 PRINT"must be pointed out that any remaining guarantee will be revoked
220 PRINT"by doing the mod.The Author or Sharpsoft cannot be held liable
230 PRINT"for any claims as a result of this modification.Read the article
240 PRINT"carefully and ensure that you feel competant to carry out the job
250 PRINT"BEFORE you order the chips necessary.
260 PRINT
270 PRINT"        Memory Configuration
280 PRINT
290 PRINT"       The SHARP MZ-80K has been available with memory configured
300 PRINT"for 20K,24K,32K,36K and 48K of RAM.This apparent confusion of
310 PRINT"memory sizes has been made possible by designing the MZ-80K with
320 PRINT"three physical memory blocks,which we will call 'RAM A','RAM B'
330 PRINT"and 'RAM C','RAM A' always contains 16K bytes(8 bit word),of
340 PRINT"memory.RAM's 'B'&'C' can be fitted with either 4K or 16K RAM I.C's
350 PRINT"Make all the RAM plug in and it can be seen that 20K computer
360 PRINT"would have 16K in RAM'A' and 4K in RAM'B'with RAM'C'empty.A 36K
370 PRINT"computer would contain 16K in RAM's 'A'&'B' with 4K in 'C',and so
380 PRINT"on.Obviously fitting 'A','B'&'C' with 16K RAM I.C's will achieve
390 PRINT"a full 48K of memory.
400 PRINT
410 PRINT"        What we Need !!
```

```
420 PRINT
430 PRINT"      Owners of 32/36K computers will need to purchase 8 off 4116
440 PRINT"Dynamic RAM i.c's.Adverts for these i.c's show two types available
450 PRINT"200ns and 150ns,the slower and cheaper 200ns version is more than
470 PRINT"adequate for the mod.The memory I bought cost 76p each by Mail
480 PRINT"order.Owners with less than 32K of memory will require 16 of the
490 PRINT"memory i.c's.Also you will need a good soldering iron and some
500 PRINT"thin single wire for the addressing link changes.
510 PRINT
520 PRINT"        Identifying the Components
530 PRINT
540 PRINT"      Looking inside will allow you to get a better understanding
550 PRINT"of what you will be required to change.If after reading the
560 PRINT"text and looking,you have any doubts,close the computer and leave
570 PRINT"well alone.
580 PRINT
590 PRINT"        First TURN OFF and remove the power lead
600 PRINT"      To open the MZ-80K turn it on its side and look at the
610 PRINT"bottom,remove the four black headed Philips screws at the edges.
620 PRINT"Carefully place the computer on its feet and hinge the Top up.
630 PRINT"The support arm on the left of the case should be locked by the
640 PRINT"screw.The Computer board is the large board taking up most of the
650 PRINT"space in the base,and the front of the board is reproduced below.
660 PRINT
670 PRINT
680 PRINT
690 PRINT
700 PRINT"|      ▮▮▮ ▒▒|▮▮ ▮ ▮  ▮▮▮  ▮▮  ▮        |
710 PRINT"|      ▮▮▮ ▒▒|▮▮ ▮ ▮  ▮▮▮  ▮▮  ▮        |
720 PRINT"|      ▮▮▮ ▒▒|▮▮ ▮ ▮  ▮▮▮  ▮▮  ▮        |
730 PRINT"|      ▮▮▮ ▮ ▮        ▮▮▮                |
740 PRINT"|        CS1                              |
750 PRINT"|  ┌ ─ ─ ─ ─ ─ ─ ─ ─ ┐                    |
760 PRINT"|  RAM ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮  ▮▮  ▮          |
770 PRINT"|   'A' ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮▮  ▮          |
780 PRINT"|  16K  ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮  ▮▮  ▮          |
790 PRINT"|                 |                       |
800 PRINT"|  ┌ ─ ─ ─ ─ ─ ─ ─ ─ ┐                    |
810 PRINT"|  RAM  ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮  ▮▮  ▮▮          |
820 PRINT"|   'B' ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮▮  ▮▮          |
830 PRINT"| 4/16K ▮ ▮ ▮ ▮ ▮ ▮ ▮ ▮  ▮▮  ▮▮          |
840 PRINT"|                 |     ▮▮                |
850 PRINT"|  ┌ ─ ─ ─ ─ ─ ─ ─ ─ ┐                    |
860 PRINT"|  RAM  ▒ ▒ ▒ ▒ ▒ ▒ ▒ ▒  ▮▮               |
870 PRINT"|   'C' ▒ ▒ ▒ ▒ ▒ ▒ ▒ ▒ ▮▮  ▒▒▒▒▒▒       |
880 PRINT"| 4/16K ▒ ▒ ▒ ▒ ▒ ▒ ▒ ▒  ▮▮  ▒▒▒▒▒▒       |
890 PRINT"|                 |                       |
900 PRINT"| BITS  0 1 2 3 4 5 6 7     CS 2          |
910 PRINT"└─────────────────┘                       ┘
920 PRINT
930 PRINT
940 PRINT
950 PRINT"      The parts identified '▒' are the parts that will be changed
960 PRINT"or modified.RAM's 'B'&'C' will have new i.c's substituted and
970 PRINT"CS 1 & CS 2 will be modified to enable new Chip Select signals
980 PRINT"to address the extended memory.CS 1 enables 4K of memory for each
990 PRINT"link.Eight links enabling 32K on top of the 16K which would be
1000 PRINT"fitted to RAM'A'.Changes to CS 2,will identify to the addressing
1010 PRINT"logic that 16K RAM i.c's are fitted to RAM'B'&'C'.
1020 PRINT
1030 PRINT"        How to do the Modification
1040 PRINT
1050 PRINT"      Once again,do not do anything unless you feel confident
1060 PRINT"to remove and insert(right way round)the i.c's and solder in new
1070 PRINT"addressing links.Also normal precautions for handling MOS
1080 PRINT"devices should be observed.So here goes
1090 PRINT
1100 PRINT"      Open the case as decribed in previous paragraphs,identify
1110 PRINT"once more the components to be changed.
1120 PRINT"      Taking out any 4K RAM i.c's will be the first task,these
```

```
1130 PRINT"will occupy RAM's 'B', 'B'&'C'or'C' depending on your
1140 PRINT"present memory configuration,and are identified '4027'.Remove
1150 PRINT"them by gently easing them up and out,using an i.c removal tool
1160 PRINT"will make the job simple.
1170 PRINT"      Insertion of the new i.c's has to be done carefully,making
1180 PRINT"sure they are the correct way round,pins 1 and 16 are to the
1190 PRINT"FRONT of the board.When inserting the i.c's support the board
1200 PRINT"from underneath,I used the insulated handle of a pair of pliers
1210 PRINT"which could slide under the board,and gently push home the new
1220 PRINT"memory i.c's.
1230 PRINT
1240 PRINT"      CS 1 Addressing links are just added to until they fill
1250 PRINT"completely the available connections as shown below
1260 PRINT"       ┌──┐
1270 PRINT"       │ °‿° │  16-20K
1280 PRINT"       │ °‿° │  20-24K     CS 1 Links will
1290 PRINT"       │ °‿° │  24-28K     be all wired
1300 PRINT"       │ °‿° │  28-32K     as shown at the
1310 PRINT"       │ °‿° │  32-34K     end of the mod.
1320 PRINT"       │ °‿° │  36-40K
1330 PRINT"       │ °‿° │  40-44K
1340 PRINT"     16│ °‿° │1 44-48K
1350 PRINT"       └─┐┌─┘
1360 PRINT
1390 PRINT
1400 PRINT
1410 PRINT"      The remaining changes concern CS 2,at the front of the
1420 PRINT"board.Depending on your present memory configuration there will
1430 PRINT"two or three links already present.Remove links between pins
1440 PRINT"3 & 14 and 5 & 12,if fitted.
1450 PRINT"      Add links between pins 1 & 16,2 & 15,3 & 13 and 5 & 11,as
1460 PRINT"necessary.
1470 PRINT"   16              9
1480 PRINT"   ┌─────────────┐      The final link
1490 PRINT"   │+ + + + + + + +│     configuration
1500 PRINT"   ╲ ╲ ╲ ╱ ╱ ╱ ╱ ╱ │     will be as
1510 PRINT"   ╱ ╱ ╱ ╲ ╲ ╲ ╲ ╲ │     shown.
1520 PRINT"   │+ + + + + + + +│
1530 PRINT"   └─────────────┘      CS 2
1560 PRINT"    1              8
1570 PRINT
1580 PRINT
1590 PRINT"       That complete's the modification as far as changes are
1600 PRINT"concerned,it only remains to check that your handiwork is correct
1610 PRINT"and that solder has not reached the parts solder should not
1620 PRINT"have reached.
1630 PRINT"       When you are satisfied that all is correct,close the
1640 PRINT"computer reconnect the power lead and switch ON.The appearance
1650 PRINT"of the monitor prompt will assure you that no major fault exists.
1660 PRINT"       Load in the SP-5025 Basic,when it's loaded the message
1670 PRINT" 34680 BYTES will be displayed,indicating a 48K memory size.
1680 PRINT"Any response indicating less than 48K points to one of the new
1690 PRINT"i.c's being unserviceable,as the area of memory that Basic
1700 PRINT"resides in is RAM'A',the area that was not modified.
1710 PRINT
1720 PRINT"      Memory Diagnostic
1730 PRINT
1740 PRINT"       The following Diagnostic program,written in Basic,will
1750 PRINT"exercise the new memory and provide screen information to
1760 PRINT"assist you diagnose any faults that might occur
1770 PRINT
1780 PRINT"   10 I=20479:T1=170:T2=85:POKE10167,1:LIMIT I:EN=53247
1790 PRINT"   20 I=I+1:POKEI,T1:IFPEEK(I)<>T1GOTO60
1800 PRINT"   30 POKEI,T2:IFPEEK(I)<>T2GOTO60
1810 PRINT"   40 IFI>=ENTHENPRINT'Test Complete':LIMIT MAX:END
1820 PRINT"   50 GOTO20
1830 PRINT"   60 PRINT'Location':I;'is faulty':GOTO20
1840 PRINT
1850 PRINT"      The above program Limits itself to the RAM'A' memory,
1860 PRINT"if you have only changed the RAM'C' then set I to 36863.Each
```

```
1870 PRINT"memory location is sequentially loaded with a binary pattern,
1880 PRINT"which is then checked before continuing.The program will take
1890 PRINT"just under 9 minutes to test 32K.If any faults are reported,then
1900 PRINT"one or more of the new i.c's are faulty.Moving i.c's within
2000 PRINT"the RAM,or to another RAM will help pin down the fault.
2010 PRINT
2020 PRINT
2030 PRINT
2040 PRINT"      Having read the article and perhaps looked inside your
2050 PRINT"MZ-80K you will feel confident enough to carry out the 48K
2060 PRINT"Upgrade,giving you all those extra Bytes.
2070 PRINT"Perhaps in a few years we shall be discussing Megabyte Upgrades,
2080 PRINT"so until then Happy Computing!!!
2100 END
3000 END
10000 END
```

Dear Sir,

Thank you for the 'Beginners Guide to the MZ80K' and No 3 of the SHARPSOFT User Notes. (No 1 and 2 to follow soon I hope).

As requested I am writing to inform you of a difficulty experienced with the 'RENUMBER UTILITY' program on page 34 of No 3 SHARPSOFT Notes. I am attempting to use this listing for a MZ80K with 48K RAM casette (not disc), to printer. The program will not run. It fails, "DATA ERROR LINE 60090" every time. Can you plesase advise me what may be wrong. I have checked my listing several times and it is as per page 34. As a matter of interest, the short renumbering program from C. J. McD Wood on page 57 works perfectly on my machine.

M. J. G. Williams
Cheshire

Please see Mr. R.A. Wymark's letter for correction to our Renumber Utility.
Ed

Dear Sir,

Further to my telephone conversation with you recently regarding Ardensoft Toolpak, please find enclosed some notes on Sharp SP-6015 DOS, which were written following my disassembly of SP-6015 prior to writing TOOLPAK. Not having seen your user notes I don't know whether you have previously published any of this information. I have recently developed a (tape) conversion program to convert XTAL programs to SP-5025. Is there likely to be a market for it?

E. P. Fletcher
Leicester

SHARP MZ-80K DISC INFORMATION.

GENERAL.
Tracks 0,1,2,3 are the control tracks
used by Basic SP-6015- and hold the
following information:

Track 0: Sectors 1-14 is the bootstrap which is loaded by the Bootstrap ROM at $F000 when starting from the Monitor with FD.

This Bootstrap then pulls in the SP-6015 from track 4 onwards.

Track 0: Sectors 15 & 16 hold the disc identification and block availability map for tracks 4 to 69.

The information on these sectors is allocated as follows:-

BYTE 1 - No use yet found
BYTE 2 - Disc volume number
BYTES 3&4 - Number of sectors used
BYTE 5 ON - This is the block availability map organised as follows:-

Each 2 bytes represent the 16 sectors of a track, bytes 5&6 being track 4, 7&8 being track 5 etc. Each bit represents a sector, the first byte of the pair being sectors 1-8. The bits in each byte are read in the conventional manner, i.e. the least significant bit on the right. Thus if the bytes representing track 4 are

1 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1

then the following sectors have been used 1,2,3,4,5,8,9,10,11,12,13

All this information is generated by SP-6015 the T3444M disc controller only reading or writing the sector required.

Tracks 1,2,3
These 3 tracks hold the disc directory each sector containing two files. The 64 bytes of each file i.d. are organised as follows:-

Byte 1 - filetype 1,2,3,4
1 - OBJ machine code object program
2 - BTX basic source program
3 - BSD basic sequential data
4 - BRD basic random data
Bytes 2 -17 - filename
Byte 18 - lockflag, 1 for LOCK, 0 for UNLOCK
Bytes 19-20 - filesize
Bytes 21-22 - load address
Bytes 23-24 - run address
Bytes 25-62 - unused
Byte 63 - track, on which file starts
Byte 64 - sector, first sector of file

File structure

Types 1,2,3 - these are all stored as 126 byte records, bytes 127&128 being the track & sector addresses of the next sector of the file

Type 4 - the sector pointed to by the directory is the control sector for

```
BRD files. Bytes 1&2 of this sector
hold the number of 16 byte records that
have been written to the file, byte 3
onwards being the track on which data
is stored. BRD's allocate whole tracks
to data, the data being recorded as 8-
16 byte elements per sector without the
necessity for pointers to the next one in
the same track.
```

Dear Sir,
here are some hints which I hope can be useful for other readers:

1. A simple and unusual method to make a back-up copy of BASIC, Systems Program, machine code games, and other machine code programs is the following:

i. Load the program from the monitor.

ii. Return to the monitor.

iii. Clear the screen and, starting at the first row, first column, write the characters with the following display codes (see the SP-5025 manual):

175 212 33 0 212 36 0 195 130 0

(The third character from the right is the key 'cursor right' and can be written if you first press the INST key and thereafter the 'cursor right' key.)

iv. Move the cursor down one line with the cursor keys and press CR.

v. Put a tape in the recorder.

vi. Write GOTO $D000, press CR, and when the RECORD-PLAY buttons have been pressed the program will be saved to tape.

What you actually do when you enter the above characters on the screen is that you enter the following machine code program at the beginning of the video RAM:

```
XOR A; Reset carry flag
CALL NC, 0021H; Write file header to tape
CALL NC, 0024H; Write program to tape
JP 0082H; Return to monitor
```

(The CALL nn instruction can't be used because the code for that instruction is CD, hexadecimal, which is impossible to enter as a character from the keyboard).

2. In number 3 of the User Notes C. F. McD. Wood described how to protect a block of memory when loading BASIC SP-5025. This was achieved if the memory location 120BH was changed so it contained the high byte of the first address to be protected. For example if the contents is changed to A0H, the memory from A000H (inclusive) and upwards will be protected from overwriting. Here is a list of the equivalent memory locations for some other system software:

XTAL BASIC 2,25 : 2DEDH (11757 decimal)
Text Editor SP-2202 : 2223H
Relocatable Loader SP-2301 : 1222H
Symbolic Debugger SP-2401 : 1233H

3. If lines which are longer than 39 characters are printed on the screen, the MZ-80K will scroll 2 lines each time and this gives a very unsmooth scrolling. This can be avoided if the connection between the physical lines on the screen are cleared. The memory location 4468-4491 (decimal) contains these connections as follows:

   4468 is 0 if no connection between lines 0 and 1, and 1 otherwise
   4467 indicates connection between lines 1 and 2
   4491 indicates connection between lines 23 and 24

   This information can also be useful if you want to print out spaces to clear a part of the screen.

4. If you have used POKE 4465, X, to change the cursor position, together with the INPUT statement you may have recognized that it doesn't always work as it should. This can be corrected in the following way:

   Instead of POKE 4465, X write POKE 4465, X : POKE 4500, X (if the line is longer than 40 characters you have to write POKE 4465, X : POKE 4500, X + 40).
   Memory location 4500 contains the length of the actual line on the screen (0–79).

5. A simple way to print double quotes on the screen with SP-5025 is demonstrated by the following short program:

```
10 READ A$
20 A$ = RIGHTS (A$, 1)
30 PRINT A$
40 DATA A"
```

6. If you want to remove the write and read protection from SYMBOLIC DEBUGGER SP-2401, so you can examine and change locations outside the Link Area, the following modification will work:

   Change memory location 19E8H to 37H
   19E9H to 00H
   19EAH to 00H
   19F0H to A7H
   19F1H to 00H
   19F2H to 00H

I am not absolutely sure that this doesn't affect something else, but it has worked as long as I have used it.

Peter Andersson
Sweden

Dear Sir,
Readers of SHARPSOFT USER NOTES may sometimes, as I have, wished there was a "PRINT USING" facility in Sharp Basic. I have overcome this lack by using the following subroutine in my mathematical programs whenever I want figures rounded off to two decimal places, and formatted output with the decimal points in vertical line with each other.

```
20000  REM * SUBROUTINE TO ROUND OFF AND FORMAT A
NUMBER (XN)
20100  XN$ = STR$ (XN)
20200  FOR YX = 1 TO 8
20300       PT$ = MID$ (XN$, YX, 1)
20400       IF PT$ = "." GOTO 20700
20500  NEXT YX
20600  XN$ = XN$ + ".00" : GOTO 20200
20700  Z = VAL (MID$ (XN$, YX + 3, 1)) : RN= VAL (LEFT$ (XN$, YX +
2))
20800  IF Z = > 5 THEN RN = RN + 0.01
20900  RETURN
```

The routine works as follows:–

1. Line 20100: The number (XN) is converted to a string to enable string manipulations to be carried out.
2. The loop in lines 20200–20500 counts the position of the decimal point from the left of the string.
3. Line 20600 adds .00 to integers and returns to the loop to format integers.
4. Line 20600 returns the value (RN) of the string of length YX+2 (ie. two decimal places), and checks the value (Z) of the third decimal figure.
5. Line 20700 rounds up RN if Z is equal or greater than 5.
6. The loop counter YX is used to position the output. For example if the decimal points are required to fall at cursor position 25 across the screen then either:–

   PRINT TAB (25–YX); RN : REM FOR SP-5025 BASIC
   or PRINT @ 5, 25–YX; RN : REM FOR EXTENDED BASIC
7. Lines 20200–20500 can be entered as one line.

An example of the use of this subroutine is shown in the "MOLDAT" program listing which appears elsewhere in this issue of USER NOTES.

F. E. Woodward
CT12 6TH

Dear Sir,

A friend of mine – Larry Watkinson of 24 Sidney road, Gillingham, Kent kindly presented me with a list of Pokes and Peeks which he has compiled over the past two years. The ? indicates that he does not know what the number indicates, and the ref to Sharp Basic T. is with the addition of Knights Commander.

If it is of any interest you may print it in the next issue of Sharpsoft Notes.

O. W. Oldham
ME5 8PR

SHARP MZ-80K PEEKS & POKES

```
0 REM  -4095  Monitor SP-1002  (in ROM)
4096 REM ?-4151 ???
4152 REM  - 4154 Set to JP 914 by          monitor for RST 38H
4336 REM  Down to ? stack for monitor
4337 REM  - 4353  Buffer where program     name stored
4354 REM  & 4355  Sets byte length of      program to be saved
4356 REM  & 4357  Start address where      program to be copied from
4358 REM  & 4359  Sets execution           address.  ie  to auto run
4360 REM ?& 4361
4464 REM  Sets upper or lower case         0 = upper  1 = lower
4465 REM  Sets Cursor ↔  0-39
4466 REM  Sets Cursor ↑↓  0-24
4467 REM  - 4493  Buffer for screen        line length  0<40ch  1>40ch
4494 REM  Holds character under cursor
4495 REM  & 4496  Hold address at which    to flash cursor
4497 REM ? or 4499 Quote status            0 = no quote  128 = quote
4498 REM  Stores character for cursor      (normally 239)
4500 REM  Holds no. of characters in       current line
4501 REM ?& 4502
4503 REM ?& 4504
4505 REM ?& 4506
4507 REM  Clock.   0 = a.m.   1 = p.m.
4508 REM ? Set to 240 by monitor ? WHY
4509 REM  Sets tone on key press           0 = no tone  1 = tone
4510 REM  Sets TEMPO 1-7  Fast-Slow        opposite of tempo function
4511 REM  Sets LENGTH of note 0-9 as in    music function
4512 REM  Set to 1,2,or 3 according to     which octave to be played
4513 REM  Sets fine control of note        frequency  0-255
4514 REM  Sets coarse control of note      frequency  1-255
4515 REM  -4594  Input buffer for          monitor
4595 REM ?-4607 Spare ???
4608 REM  Start of BASIC                   All addresses + for SP-5025
6350 REM  Norm 34. if set to 0, allows     double quotes to be printed
10167 REM  Flag for PEEK PROTECT            0 if on  1 if off
10680 REM  Flag to STOP program saving      1 will ignore save & list
10681 REM ?
10682 REM  Flag to AUTO RUN  1 when         program saved will auto run
11516 REM  Port to input from
11587 REM  Port to output to
15478 REM  15= Form Feed, 13= Line Feed,    0 disables in LIST/P norm15
15638 REM ?
17408 REM  -17487  Input buffer for         BASIC
17493 REM ?& 17494
17495 REM ?& 17496
17497 REM ?
17751 REM  - 17756  Correct address for     TI$ (BASIC T)
17753 REM  - 17758  Hold ASCII of TI$
17761 REM  & 17762  Store highest byte      of memory in machine
17762 REM  Set 17761 & 2 as 17763 & 4       limit max will set to this
17763 REM  & 17764 · Store limit set        by LIMIT command
```

```
17765 REM ?
17772 REM ?& 17773
17774 REM ?& 17775
17776 REM ?& 17777
17778 REM ?& 17779
17780 REM ?
17781 REM  & 17782  Copy of 17784 & 5
17784 REM  & 17785  Address of next                    DATA item to be read
17786 REM  0 = No files open                           1 = Files open
17787 REM ?& 17788
17789 REM ?
17790 REM ?& 17791
17792 REM ?& 17793
17806 REM ?
17807 REM ?& 17808
17810 REM  0 = VERIFY                                  1 = LOAD
17826 REM  Stores ascii value of lust                  key pressed
17828 REM  Stores ascii value of key                   being pressed
17966 REM ?& 17967
17968 REM ?& 17969
17970 REM ?& 17971
17972 REM  & 17973  Stores address of                  start of DEF FN space
17974 REM  & 17975  Stores address of                  start of Multi Dimension $
17976 REM  & 17977  Stores address of                  start of Single Dimension $
17978 REM  & 17979  Stores address of                  start of String space
17980 REM  & 17981  Stores address of                  start of Multi Dimension V
17982 REM  & 17983  Stores address of                  start of Single Dimension V
17984 REM  & 17985  Stores address of                  start of Variable space
17986 REM ?& 17987
17988 REM ?& 17989
17990 REM ?& 17991
17992 REM  & 3 Set to 135 and set 18436                to 0 to clear FOR stack
17994 REM  & 5 Set to 240 and set 18437                to 0 to clear GOSUB stack
18006 REM  = 18311  FOR NEXT Stack
18311 REM  = 18416  GOSUB Stack
18413 REM ?
18429 REM ?& 18330
18431 REM  = 18432  Hold line no.                      being executed
18433 REM ?& 18334
18435 REM ?
18436 REM  Set to 0 and set 17992 to                   135 to clear FOR stack
18437 REM  Set to 0 and set 17994 to                   240 to clear GOSUB stack
18438 REM  Start of free RAM after                     BASIC
18440 REM  Set to 0 makes first line of                Program undeletable
18547 REM  Set to 4 turns light red                    Set to 5 turns light green
19555 REM  Set to 0 blanks screen                      Set to 1 turns on screen
```

Dear Sir,

Thank you for sending the July edition of your Sharpsoft Catalogue. I should be grateful if you would forward tapes of Asteroids and Block Kuzushi – I attach my cheque for £11.70.

Perhaps I could mention that I have found your User notes most helpful – perhaps one day I shall be knowledgeable enough to make a contribution myself! Meanwhile if you are short of material at any time perhaps you could suggest ways of getting round what appears to be a fault in the GET function on the MZ80K. I find that the machine quite often fails to read characters typed in, causing great frustration particularly when playing arcade-type games! The problem can be demonstrated by entering characters during the programme:

```
10 FOR J = 1 TO 20000
20 GET A$: PRINT A$
30 NEXT J
```

It will be found that quite a lot of "A"s do not get printed and presumably have not been picked up from the keyboard.

MK4 3AU

Digital Sound Generator

16 – – – 3906 Hz

The following program could be improved by the addition of an indication of the actual frequency generated. The frequency is always correct to within ± O.2%.

```
1   LIMIT 21504
2   FOR I = 23552 TO 23565
3   READ D : POKE I,D : NEXT I
4   DATA 229, 237, 75, 254, 91, 237, 67, 161, 17, 205, 68, 0, 225, 201
5   INPUT "FREQUENZ = " ; C
6   A = 1000000 / (C * 256)
7   B = (A – INT (A)) * 256 + 0.5
11   POKE 23550, B
22   POKE 23551, A
33   USR (23552)
44   GET A$
55   IF A$ = "S" THEN GOTO 5
66   GOTO 44
77   STOP
88   END
```

M. Hermann

Dear Sirs,
Perhaps the following two ideas might be of interest to your readers.

1. Poke 4687,62 will ring bell whenever READY is displayed on the screen.
2. Poke 4687,48 will switch bell off.

The following program might also prove of interest if your readers possess a KNIGHT COMMANDER toolkit. It can be joined to the Sharp SP-5025 BASIC so that instead of loading two cassettes at the start of a programming session only one is required.

1. Load SP-5025 BASIC
2. When <READY> enter BYE, press <CR>
3. Load KNIGHT COMMANDER
4. When <READY> enter the following BASIC program.

```
10  Poke 10167,1
20  DATA 66, 65, 83, 73, 67, 32, 83, 80, 45, 53, 48, 50, 53, 18, 0, 50, 0,
    18
30  FOR A = 0 TO 13 : READ B : POKE 4337 + A, B : NEXT
40  FOR A = 0 to 3: READ B: POKE 4354 + A, B: NEXT
50  USR (33) : USR (36)
```

5. Type <RUN>, press <CR>
6. Follow instructions shown on screen.
7. Save the program on a new cassette not on top of your KNIGHT COMMANDER or Sharp SP-5025 BASIC – just in case there's an error.

Hope this might be of interest.

T. A. Metcalfe
Norfolk

Dear Sir,

On looking through your catalogue I saw a manual "Using Forth" but could not find the price quoted on the lists provided. Can you please inform me as to the suitability of the manual for someone completely new to this language, and the price of the manual.

A suggestion for a future article in your user notes would be extra commands for Xtal Basic, these could include APPEND, REMEMBER, and a command to delete REM statements to form a program development package.

If someone is just starting to use Forth the following program may be useful to save a security copy of the language tape. S0024 S0021 USR USR

The program is entered in the run mode and on pressing 'CR' the standard recording pattern is followed.

W. Richardson
NG12 3NE

Dear Sir,

CP/M Software

I suspect that I am not alone in the category of personal computer owners who wants to run before knowing how to walk.

With the MZ-80K (48K), Dual floppy and an Epson MX-80ZF/T printer and a rather incomplete knowledge of Basic, I was making no progress in writing anything but variations on the standard student programmes presented in Basic manuals. I came to the conclusion that, if I ever want to run any useful programmes – bank statements, investment records, budget and cash flow projections, I should convert to CP/M – under the impression that this would give me access to a wide range of "off the shelf" business applications.

Having now installed CP/M I find myself no further forward.

I see from my "User Notes" that you intend discussing CP/M in a future edition and would like to make a few suggestions regarding the questions that are not clearly covered in any publications I have seen:

1. What are the advantages of having CP/M?
2. Under CP/M what software is necessary to load a better version of Basic than Sharp Disc Basic?
3. Is what is necessary available in the CP/M Users Group Software and, if so, where?
4. And, if available, what is the step by step procedure of loading it? (I am presently going crazy ringing the changes on a few volumes of UG including BASIC.COM, BASCOM.LIT, RUN.COM, BAS2–1.COM, BAS2–0. COM, etc.).
5. When will Sharpsoft give us a list of what is available from the Users Group?
6. How can one find out where and what business orientated applications are available to run under Sharp CP/M?

So much for CP/M but, while writing to you I would like to mention three other points.

Star Trek

In your issue No. 2 you listed this fine programme but I challenge anyone who is not familiar with this type of game to play it only on the basis of your very summary instructions. Could you not complete the job in future by printing a full version of the instructions?

Programmers

Again I am sure that I am not alone among your overseas readers living where Sharp MZ-80Ks are rare and competent programmers non-existent. I should imagine that you have many readers that would be glad to quote a fee for providing a programme on disk on the basis of a clear analysis of the input available and the output required.

Would it not be of interest to all to invite such people to provide you with

their names, addresses, hardware configurations and the types of applications they are experienced in – Scholastic – Business – Scientific – Technical etc. and then to publish this list?

Publicity

In view of the very sloppy standard of manuals produced by Sharp (U.K.) the initiative of Sharpsoft in producing the Users Notes and such excellent booklets ch as R. G. Meadows' "Beginners Guide" tends to fill an enormous gap. Surely it would be worthwhile advertising the existence of these publications in the hobby magazines in order to get a wider readership.

Keep up the good work!

Guy Norton
Rome

In the past we have not included a CP/M column in these user notes because so few CP/M based MZ-80K computers were being used by our readers. However, this situation is changing and hence we will in the future be including CP/M software notes for both the MZ-80K and MZ-80B computers.

Ed

GAMES & OTHER PROGRAMMES

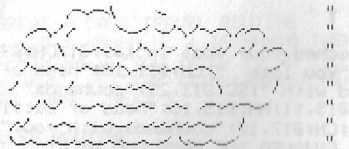
```
1000 REM*TRAIL BOSS--WRITTEN IN EXTENDED BASIC FOR THE SHARP MZ-80K
1010 REM*BY F.E.WOODWARD, 8 VIOLET AV. RAMSGATE KENT.*
1020 REM*INITIALISATION OF VARIABLES*
1030 HN=50*(INT(81*RND(1)+20)) :SH=HN :  REM*SIZE OF HERD
1040 CN=INT(HN/115) : SC=CN :REM*NUMBER OF COWHANDS
1050 LN=INT(401*RND(1)+800) : REM*LENGTH OF DRIVE
1060 RN=INT(LN/300) : REM*NUMBER OF RIVERS
1070 R1=INT(LN/(RN+1)) : REM*DIST TO FIRST RIVER
1080 R2=INT(2.35*R1):REM*DIST TO 2nd.RIV1070 R3=3*R1 : REM*DIST TO 3rd.RIVER
1090 R3=3*R1 : IF RN<3 GOTO 1110
1100 R4=INT(3.75*R1) : IF RN<4 GOTO 1120
1110 R3=LN+100 : R4=LN+100
1120 R4=LN+100
1130 DM=(INT(LN/2.5))+(INT(101*RND(1)+50)) : REM*DIST TO MOUNTAIN RANGE
1140 DN=0 : DT=0 : REM*NO.OF DAYS AND DIST.TRAVELLED SET TO ZERO
1150 REM*SET UP STARTING DISPLAY AND PRINT DETAILS AND INSTRUCTIONS
1160 GOSUB 3230
1170 PRINT@2,2;"You are a Trail-boss of a crew of"
1180 PRINT@4,1;"cowhands and a herd of steers which"
1190 PRINT@6,1;"you have to drive to the railroad"
1200 PRINT@8,1;"stockyard"
1210 PRINT@10,2;"There will be a mountain range to"
1220 PRINT@12,1;"go through by the pass.There will be"
1230 PRINT@14,1;"2 or more rivers to cross,and you"
1240 PRINT@16,1;"may pass near a town or towns which"
1250 PRINT@18,1;"could be either for or against your"
1260 PRINT@20,1;"advantage.You may also encounter"
1270 PRINT@22,1;"rustlers,indians,and storms."
1280 FOR T=1 TO 10000 : NEXT T : PRINT"@"
1290 PRINT@2,2;"The result of these events is partly"
1300 PRINT@4,1;"chance,and partly depends on your"
1310 PRINT@6,1;"situation at that time,ie.the size of"
1320 PRINT@8,1;"the herd and the number of hands you"
1330 PRINT@10,1;"have a that time."
1340 PRINT@14,2;"Also if you attempt to cover too"
1350 PRINT@16,1;"much ground in a day this will have"
1360 PRINT@18,1;"an adverse result"
1370 PRINT@22,2;"WHEN YOU ARE READY KEY 'S'"
1380 GET A$ : IF A$<>"S" GOTO 1380
1390 PRINT"@" :IF HN=<0 GOTO 3140: IF CN=<0 GOTO 3140
1400 CC=INT(HN/150) : REM*CRITICAL NO. OF COWHANDS
1410 DL=40+(CN-CC):REM*MAX DAYS TRAVEL
1420 PRINT@ 2,15;"SITUATION DETAILS": USR(62)
1430 PRINT@6,2;"TRAIL LENGTH=";@6,17;LN;@6,22;"miles"
1440 PRINT@8,2;"NUMBER OF STEERS IN HERD=";@8,28;HN
1450 PRINT@10,2;"MILES COVERED SO FAR=";@10,24;DT
1460 PRINT@12,2;"NUMBER OF DAYS GONE=";@12,23;DN
1470 PRINT@14,2;"NUMBER OF COWHANDS AT THE MOMENT=";@14,36;CN
1480 PRINT@ 20,2;"INPUT THE MILES YOU INTEND TO COVER"
1490 PRINT@22,2;"TODAY" : INPUT ML : PRINT"@" :GOTO 1510
1500 REM*RESULT OF DAYS TRAVEL INPUT
1510 DN=DN+1 : DT=DT+ML
1520 IF ML>DL GOSUB 2320: REM*DAYS TRAVEL TOO LONG
1530 IF DT=>LN GOTO 2990: REM*END OF TRAIL
1540 IF DT=>R1 THEN R1=LN+100 : GOTO 1680: REM*REACHED RIVER
1550 IF DT=>R2 RHEN R2=LN+100 : GOTO 1680: REM*REACHED RIVER
1560 IF DT=>R3 THEN R3=LN+100 : GOTO 1680: REM*REACHED RIVER
1570 IF DT=>R4 THEN R4=LN+100 : GOTO 1680: REM*REACHED RIVER
1580 IF DT=>DM THEN DM=LN+100 : GOTO 1970: REM*REACHED MOUNTAINS
1590 EV=INT(10*RND(1)+1) : REM*EVENT SELECTION
1600 IF EV=<5 GOTO 1390: REM*UNEVENTFUL DAY
```

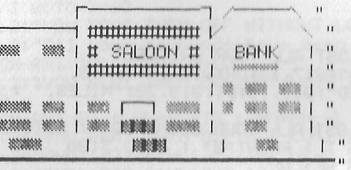
```
1610 IF EV=6 GOSUB 2390: REM*RUSTLERS
1620 IF EV=7 GOSUB 2560: REM*INDIANS
1630 IF EV=8 GOSUB 2700: REM*TOWN
1640 IF EV=9 GOSUB 2900: REM*STORM
1650 IF EV=10 GOTO 1390: REM*UNEVENTFUL DAY
1660 GOTO 1390
1670 REM*RIVER ROUTINE
1680 PRINT"E" :GOSUB 4280: PRINT@15,5:"YOU HAVE REACHED A RIVER"
1690 RV=INT(3*RND(1)+1)
1700 IF RF=3 PRINT@17,3:"The river is low.You can cross here."
1710 PRINT@16,2:"You must send a scout up or down"
1720 PRINT@18,1:"river to find a crossing point."
1730 PRINT@20,2:"Input 1 for down or 2 for up." : INPUT DS
1740 PRINT@20,2:"                              "
1750 IF DS=RV GOTO 1810: REM*RIGHT CHOICE
1760 PRINT@20,2:"YOU CHOSE THE WRONG WAY"
1770 PRINT@21,2:"You lost two days finding the."
1780 PRINT@22,1:"crossing and it will take you"
1790 PRINT@23,1:"one more day to cross over"
1800 DN=DN+3 : GOTO 1850
1810 PRINT@20,2:"YOU MADE THE RIGHT CHOICE"
1820 PRINT@21,2:"It will take you one day to get"
1830 PRINT@22,1:"to the crossing and cross over."
1840 DN=DN+1
1850 FOR T=1 TO 3000 : NEXT T : IF CN<CC GOTO 1900
1860 PRINT"E"
1870 PRINT@4,2:"YOU MADE A GOOD CROSSING WITH"
1880 PRINT@6,1:"NO LOSSES."
1890 FOR T=1 TO 4000 : NEXT T : GOTO1390
1900 HL=50*(CC-CN) : HN=HN-HL
1910 PRINT"E"
1920 PRINT@4,1:"YOU LOST";PRINT@4,10:HL;@4,14:"HEAD OF STEERS"
1930 PRINT@6,1:"BECAUSE YOU HAD TOO FEW COWHANDS"
1940 FOR T=1 TO 4000 : NEXT T
1950 GOTO 1390
1960 REM*MOUNTAIN ROUTINE
1970 PRINT"E":GOSUB 4150:PRINT@14,5:"YOU HAVE REACHED THE MOUNTAIN RANGE"
1980 PD=INT(2*RND(1)+1) : PL=50*INT(3*RND(1)+1)
1990 PRINT@18,2:"You must send out a scout to find"
2000 PRINT@20,1:"the pass,either N or S."
2010 PRINT@22,2:"Input 1 for N or 2 for S."
2020 INPUT DP:PRINT@22,2:SPC(35)
2030 PRINT@18,2:SPC(35):@20,1:SPC(35):@22,2:SPC(35)
2040 IF DP<>PD GOTO 2100
2050 PRINT"E":GOSUB 4150
2060 PRINT@18,2:"YOU CHOSE RIGHT."
2070 PRINT@20,1:"The pass is one days travel away"
2080 FOR T=1 TO 5000 : NEXT T
2090 DN=DN+1 : GOTO 2160
2100 PRINT@18,2:"YOU CHOSE THE WRONG WAY"
2110 PRINT@20,1:"You lost 3 days in finding the pass"
2120 PRINT@22,1:"and it is one days travel away."
2130 DN=DN+4
2140 FOR T=1 TO 5000 : NEXT T
2150 REM*PASS ROUTINE
2160 PRINT"E"
2170 PRINT@4,2:"YOU HAVE REACHED THE PASS" : MP=0
2180 PRINT@6,2:"The pass is";@6,14;PL;@6,18:"miles long"
2190 PRINT@9,2:"Input proposed days mileage." : INPUT ML
2200 MP=MP+ML : DT=DT+ML : DN=DN+1
2210 IF ML>DL GOSUB 2320
2220 IF MP<PL GOTO 2260
2230 PRINT"E" : PRINT@4,2:"YOU HAVE CLEARED THE MOUNTAINS"
2240 FOR T=1 TO 2500 : NEXT T
2250 GOTO 1390
2260 MV=INT(5*RND(1)+1) : REM*EVENT SELECTION
2270 IF MV=2 GOSUB 2560: REM*INDIANS
2280 IF MV=4 GOSUB 2390: REM*RUSTLERS
2290 PRINT"E":PRINT@6,2:"Youve still";@6,15;PL-MP:@6,19:"miles of pass to go"
2300 FOR T=1 TO 5000 : NEXT T
```

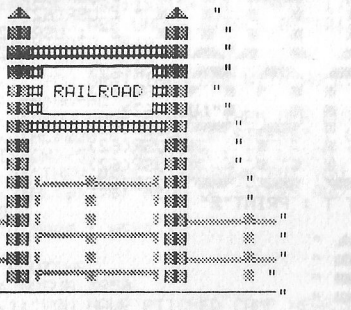
```
2310 GOTO 2190
2320 REM*TOO GREAT A DAYS TRAVEL ROUTINE
2330 HL=(ML-DL)*20 : HN=HN-HL : IF HN=<0 THEN HL=HN+HL
2340 PRINT"&":PRINT@4,2;"YOU TRIED TO GO TOO FAST."
2350 PRINT@8,2;"YOU LOST";@8,11;HL;@8,16;"HEAD OF STEERS"
2360 FOR T=1 TO 5000 : NEXT T
2370 RETURN
2380 REM*RUSTLERS ROUTINE
2390 PRINT"&" :GOSUB 3400: RX=2*INT(10*RND(1)+1): REM*NO OF RUSTLERS
2400 PRINT@18,2;"YOU ARE UNDER ATTACK BY";@18,26;RX;@18,30;"RUSTLERS"
2410 IF RX>CN GOTO 2450
2420 IF RX>CN/2 GOTO 2490
2430 IF RX>CN/3 GOTO 2460
2440 PRINT@20,2;"YOU BEAT THEM OFF WITHOUT LOSS" : GOTO 2530
2450 PRINT@20,2;"All your cowhands have been killed." : CN=0 : GOTO 2530
2460 LC=INT((CN-RX)/10) : CN=CN-LC
2470 PRINT@20,2;"YOU BEAT THEM OFF,BUT LOST";@20,29;LC;@20,32;"COWHANDS"
2480 GOTO 2530
2490 LC=INT((CN-RX)/8) : HL=RX*10 : HN=HN-HL : CN=CN-LC
2500 PRINT@20,2;"YOU BEAT THEM OFF,BUT LOST";@20,29;LC;@20,32;"COWHANDS"
2510 PRINT@22,2;"AND";@22,6;HL;@22,10;"HEAD OF STEERS" : GOTO 2530
2520 PRINT@22,2;"AND";@22,6;HL;@22,10;"HEAD OF STEERS" : GOTO 2530
2530 FOR T=1 TO 5000 : NEXT T
2540 RETURN
2550 REM*INDIANS ROUTINE
2560 PRINT"&":GOSUB 3580:IX=15*INT(10*RND(1)+1)  : REM*NO OF INDIANS
2570 PRINT@17,1;"YOU ARE UNDER ATTACK BY";@17,25;IX;@17,30;"INDIANS"
2580 IF IX>2.5*CN GOTO 2640
2590 IF IX>1.5*CN GOTO 2610
2600 PRINT@18,1;"YOU BEAT THEM OFF WITH NO LOSS" : GOTO 2680
2610 HL=IX*2 : HN=HN-HL
2620 PRINT@18,2;"YOU HAVE BEATEN THEM OFF BUT LOST"
2630 PRINT@20,2;HL;@20,6;"HEAD OF STEERS" : GOTO 2680
2640 LC=INT((IX-CN)/5) : HL=IX*5 : IF LC>CN THEN LC=CN : CN=CN-LC : HN=HN-HL
2650 PRINT@18,2;"THEY HAVE WITHDRAWN BUT YOU LOST"
2660 PRINT@20,2;HL;@12,8;"HEAD OF STEERS AND";@20,27;CL;@10,31;"COWHANDS"
2670 GOTO 2680
2680 FOR T=1 TO 5000 : NEXT T
2690 RETURN
2700 REM*TOWN ROUTINE
2710 PRINT"&": GOSUB 3850
2720 PRINT@15,1;"YOU HAVE PITCHED CAMP NEAR A TOWN"
2730 IF CN>CC GOTO 2820
2740 PRINT@17,1;"YOU GO TO TOWN TO HIRE MORE HANDS"
2750 Z=INT(3*RND(1)+1) : RC=INT(3*RND(1)+1)
2760 IF Z=1 GOTO 2800
2770 IF Z=2 GOTO 2810
2780 NC=RC*(CC-CN) : CN=CN+NC
2790 PRINT@19,1;"YOU MANAGED TO HIRE";@19,21;NC;@19,25;"NEW HANDS" : GOTO 2870
2800 PRINT@19,1;"HARD LUCK!! THERE WER NO MEN AVAILABLE." : GOTO 2870
2810 NC=INT(10*RND(1)+1) : CN=CN+NC : GOTO 2790
2820 PRINT@17,1;"SOME OF YOUR HANDS WENT TO TOWN"
2830 PRINT@19,1;"ON A SPREE AND GOT INTO A DRUNKEN BRAWL"
2840 PRINT@21,1;"YOU HAD TO LEAVE SOME OF THEM IN JAIL."
2850 Z=INT(3*RND(1)+1) : NC=INT((CN-CC)/Z) : CN=CN-NC
2860 PRINT@23,1;"YOU LOST";@23,10;NC;@23,13;"HANDS"
2870 FOR T=1 TO 4000 : NEXT T
2880 RETURN
2890 REM*STORM ROUTINE
2900 PRINT"&" : GOSUB 3720
2910 PRINT@14,1;"YOU HAVE BEEN CAUGHT IN A FLASH STORM"
2920 IF CN<CC GOTO 2940
2930 PRINT@18,1;"HOWEVER YOU WEATHERED THE STORM OK." : GOTO 2970
2940 HL=10*(CC-CN) : HN=HN-HL
2950 PRINT@16,1;"YOU WERE SHORT OF HANDS AND LOST";@16,33;HL;@16,36;"HEAD"
2960 PRINT@18,1;"OF STEERS IN THE STORM"
2970 FOR T=1 TO 4000:NEXT T
2980 RETURN
2990 REM*END OF TRAIL
3000 PRINT"&": GOSUB 3980
```

```
3010 PRINT@22,12;"END OF TRAIL"
3020 PRINT@23,12;"─────────"
3030 FOR T=1 TO 4000 : NEXT T
3040 PRINT"Ⓔ"
3050 PRINT@9,2;"You have reached your goal in";@9,31;DN;@9,34;"days"
3060 LH=SH-HN : PRINT@11,2;"You lost  ";LH;@11,18;"head of cattle."
3070 PRINT@13,2;"You started with ";SC;@13,25;"cowhands"
3080 PRINT@15,2;"You have";@15,11;HN;@15,16;"head of cattle left."
3090 PRINT@17,2;"and";@17,6;CN;@17,10;"cowhands with you."
3100 PRINT@20,2;"I HOPE YOU ENJOYED THE GAME,AND WILL"
3110 PRINT@22,2;"TRY AGAIN SOON. BYE FOR NOW."
3120 END
3130 REM * LOST ROUTINE
3140 IF HN=<0 GOTO 3160
3150 IF CN=<0 GOTO 3200
3160 PRINT"Ⓔ": GOSUB 4450
3170 PRINT@15,2;"YOU HAVE LOST ALL YOUR CATTLE."
3180 PRINT@17,2;"YOU LASTED";@17,13;DN;@17,17;"DAYS."
3190 PRINT@19,2;"YOU COVERED";@19,15;DT;@19,20;"MILES." : GOTO 3220
3200 PRINT"Ⓔ"
3210 PRINT@15,2;"YOU HAVE LOST ALL YOUR COWHANDS" : GOTO 3180
3220 FOR T=1 TO 4500 : NEXT T : PRINT"Ⓔ" : GOTO 3100
3230 PRINT"Ⓔ" : REM+DISPLAY SET UP
3240 PRINT@3,5"▀▀▀▀ ▀▀▀▀▀ ▄▄▄▄ ▀▀▀ ▀▀▀" :USR(62)
3250 PRINT@4,5"▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ " : USR(62)
3260 PRINT@5,5"▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ " : USR(62)
3270 PRINT@6,5"▀ ▀▀▀▀ ▀▀▀▀ ▀ ▀ ▀ " : USR(62)
3280 PRINT@7,5"▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ " : USR(62)
3290 PRINT@8,5"▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀ ▄" : USR(62)
3300 PRINT@9,5"▄▄▄ ▄▄▄ ▄▄▄ ▄▄▄ ▄▄▄ ▄▄▄ ▄▄▄▄▄" : USR(62)
3310 PRINT@14,9;"▀▀▀▀ ▄▄▄▄ ▄▄▄▄ ▄▄▄▄":USR(62)
3320 PRINT@15,9;"▀ ▀ ▀ ▀ ▀ ▀ ▀ ▀":USR(62)
3330 PRINT@16,9;"▀ ▄ ▀ ▀ ▀ ▀":USR(62)
3340 PRINT@17,9;"▀▀▀ ▀ ▀ ▀▀▀▀ ▀▀▀▀":USR(62)
3350 PRINT@18,9;"▀ ▄ ▀ ▀ ▀ ▀":USR(62)
3360 PRINT@19,9;"▀ ▀ ▀ ▀ ▄ ▀ ▄ ▀":USR(62)
3370 PRINT@20,9;"▄▄▄▄ ▄▄▄▄ ▄▄▄▄ ▄▄▄▄":USR(62)
3380 FOR T=1 TO 3000 : NEXT T : PRINT"Ⓔ"
3390 RETURN
3400 PRINT@1,8;"      ▄▄▄ "
3410 PRINT@2,8;"     ▄▄▄▄ "
3420 PRINT@3,8;"     ▄▄▄▄ "
3430 PRINT@4,8;"     ▄▄▄▄ "
3440 PRINT@5,8;"   ▄▄▄▄▄ "
3450 PRINT@6,8;"  ▄▄▄▄▄▄ "
3460 PRINT@7,8;"  ▄▄▄▄▄▀ "
3470 PRINT@8,9;"  ▌  ▌ "
3480 PRINT@9,9;"  ▙  ▟ "
3490 PRINT@10,9;"   ▚ ▞ "
3500 PRINT@11,9;"   ▛ ▙ "
3510 PRINT@12,9;"   ▌ ▟ "
3520 PRINT@13,9;"  ▄▀▄▄▀ "
3530 PRINT@14,9;"  ▌ ▀▄ ▌ "
3540 PRINT@15,9;" ▄▀ ▄▀ ▀▄ "
3550 PRINT@16,9;"▄▄▄▄▄▄▄▄▄▄ "
3560 PRINT@17,9;"▀▀▀▀▀▀▀▀▀▀ "
3570 RETURN
3580 PRINT@4,8; "  ▄▀ "
3590 PRINT@5,8; "  ▀▄      ▄▄▄▀▄▄  "
3600 PRINT@6,8; "   ▀     ▄▄▄▀▄▄▄▀ "
3610 PRINT@7,8; "  ▄ ▀    ▀▄▄▄▀▄▀ "
3620 PRINT@8,8; " ▄▀▄▄▀     ▀▄▄▀  "
3630 PRINT@9,8; " ▀▄▄▄▀    ▀▄▄▄▄▀ "
3640 PRINT@10,8;"▄▄▄▀▀    ▀ ▌▌▄▀  "
3650 PRINT@11,8;"▀▄▄▄▀    ▀ ▀▀▄▄▄ "
3660 PRINT@12,8;"  ▀▄▀▄    ▀▄▀ ▄▀ "
3670 PRINT@13,8;"  ▀  ▀▄     ▌  ▀▄ "
3680 PRINT@14,8;"       ▀▄   ▌▌ ▀▄ "
3690 PRINT@15,8;"         ▀▄ ▀  ▀▄ "
3700 PRINT@16,8;"           ▀▄▄▄▄ "
```

```
3710 RETURN
3720 REM STORM
3730 PRINT@2,5;"
3740 PRINT@3,5;"
3750 PRINT@4,5;"
3760 PRINT@5,5;"
3770 PRINT@6,5;"
3780 PRINT@7,5;"
3790 PRINT@8,5;"
3800 PRINT@9,5;"
3810 PRINT@10,5;"
3820 PRINT@11,5;"
3830 PRINT@12,5;"
3840 RETURN
3850 PRINT@3,5;"
3860 PRINT@4,5;"
3870 PRINT@5,5;"
3880 PRINT@6,5;"
3890 PRINT@7,5;"
3900 PRINT@8,5;"
3910 PRINT@9,5;"
3920 PRINT@10,5;"
3930 PRINT@11,5;"
3940 PRINT@12,5;"
3950 PRINT@13,5;"
3960 PRINT@14,5;"
3970 RETURN
3980 PRINT"
3990 PRINT"
4000 PRINT"
4010 PRINT"
4020 PRINT"
4030 PRINT"
4040 PRINT"
4050 PRINT"
4060 PRINT"
4070 PRINT"
4080 PRINT"
4090 PRINT"
4100 PRINT"
4110 PRINT"
4120 PRINT"
4130 PRINT"
```







```
4140 RETURN
4150 PRINT@1,5;"                                "
4160 PRINT@2,5;"                                "
4170 PRINT@3,5;"                                "
4180 PRINT@4,5;"                                "
4190 PRINT@5,5;"                                "
4200 PRINT@6,5;"                                "
4210 PRINT@7,5;"                                "
4220 PRINT@8,5;"                                "
4230 PRINT@9,5;"                                "
4240 PRINT@10,5;"                               "
4250 PRINT@11,5;"////\\\\\\\\\\\\\\\\\\\\\\\\\\\\"
4260 PRINT@12,5;"/\\\\\\\\\\\\\\\\\\\\\\\\\\\\////"
4270 RETURN
4280 PRINT@1,5;"                                "
4290 PRINT@2,5;"                                "
4300 PRINT@3,5;"                                "
4310 PRINT@4,5;"                                "
4320 PRINT@5,5;"                                "
4330 PRINT@6,5;"                                "
4340 PRINT@7,5;"                                "
4350 PRINT@8,5;"                                "
4360 PRINT@9,5;"////////////////////////////////"
4370 PRINT@10,5;"////////////////////////////////"
4380 PRINT@11,5;"////////////////////////////////"
4390 PRINT@12,5;"////////////////////////////////"
4400 PRINT@13,5;"////////////////////////////////"
4410 PRINT@14,5;"////////////////////////////////"
4420 PRINT@15,5;"////////////////////////////////"
4430 PRINT@16,5;"////////////////////////////////"
4440 RETURN
4450 PRINT@1,5;"                                "
4460 PRINT@2,5;"                                "
4470 PRINT@3,5;"                                "
4480 PRINT@4,5;"                                "
4490 PRINT@5,5;"                                "
4500 PRINT@6,5;"                                "
4510 PRINT@7,5;"                                "
4520 PRINT@8,5;"                                "
4530 PRINT@9,5;"                                "
4540 RETURN
```

```
1000 REM-MZ80-K COLONIES...PROG.USING EXTENDED SHARP BASIC
1010 REM BY F.E.WOODWARD. 8 VIOLET AV. RAMSGATE
1020 PRINT"@":PRINT@6,10;"MZ80-K COLONIES"
1030 PRINT@7,10;"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒"
1040 FOR N=1 TO4000:NEXT N:PRINT"@"
1050 PRINT@4,2;"BASED ON THE POPULAR COMPUTER GAME OF"
1060 PRINT@6,15;"LIFE":PRINT@7,15;"****"
1070 PRINT@9,2;"TAILORED FOR THE MZ80-K AND EXTENDED"
1080 PRINT@11,11;"SHARP BASIC"
1090 FOR X=1 TO5000:NEXT X:PRINT"@"
1100 PRINT@4,5;"WATCH YOUR COLONIES GROW OR PERISH"
1110 PRINT@7,2;"EACH COLONY TAKES FROM 15 TO60 SECS.TO"
1120 PRINT@9,1;"GO THROUGH ITS CHANGING STATE DEPENDING"
1130 PRINT@11,1;"ON THE SIZE OF THE COLONY"
1140 PRINT@15,2;"THERE WILL NOW BE A SHORT PAUSE WHILE"
1150 PRINT@17,2;"THE GROWING MEDIA IS PREPARED"
1160 FOR N=1 TO 3000:NEXT N
1170 REM INITIALISATION OF VARIABLES
1180 DIM C(16),A(20,20)
1190 FOR I=1 TO 20
1200 FOR J=1 TO 20
1210 A(I,J)=0
1220 NEXT J
1230 NEXT I
1240 FOR I=1 TO 16:C(I)=0:NEXT I
1250 DATA 1,0,0,1,-1,0,0,-1,1,1,1,-1,-1,1,-1,-1
1260 FOR I=1  TO 16 :READ C(I) : NEXT I: PRINT"@"
1270 FOR I=18 TO 39:PRINT@22,I;"%";@1,I;"%":NEXT I
1280 FOR J=1 TO 22:PRINT@J,18;"%";@J,39;"%":NEXT J
1290 REM INSTRUCTIONS
1300 PRINT@1,5;"KEY IN"
1310 PRINT@2,2;"CO-ÒRDINATES OF"
1320 PRINT@3,2;"YOUR INITIAL"
1330 PRINT@4,2;"CELLS AS :-"
1340 PRINT@5,4;"X,Y THEN 'CR'"
1350 PRINT@6,2;"(X&Y FROM 1→20)"
1360 PRINT@7,1;"END WITH 0,0 CR"
1370 PRINT@9,1;"WHEN EACH NEW"
1380 PRINT@10,1;"GENERATION OF *"
1390 PRINT@11,1;"IS READY THEN:-"
1400 PRINT@13,1;"KEY 'G' FOR A"
1410 PRINT@14,1;"NEW GENERATION"
1420 PRINT@15,5;"OR"
1430 PRINT@16,1;"KEY 'K' TO"
1440 PRINT@17,1;"END GROWTH"
1450 PRINT@22,1;"CELL CO-ORDS."
1460 INPUT X,Y
1470 IF (X=0)*(Y=0) GOTO 1520
1480 IF (X>20)+(Y>20) GOTO 1450
```

```
1490 A(X,Y)=10 : PRINT@X+1,Y+18;"*"
1500 PRINT@22,1;SPC(17):PRINT@23,1;SPC(17):GOTO 1450
1510 REM START OF GROWTH PATTERN
1520 REM SET UP MATRIX AND SEARCH AND SET UP COLONY DISPLAY
1530 FOR X=1 TO 20
1540 FOR Y=1 TO 20
1550 IF A(X,Y)<10 THEN 1610
1560 FOR M=1 TO 15 STEP 2
1570 XC=X+C(M):YC=Y+C(M+1)
1580 IF XC*(XC-21)*YC*(YC-21)=0                    THEN 1600
1590 A(XC,YC)=A(XC,YC)+1
1600 NEXT M
1610 NEXT Y
1620 NEXT X
1630 REM REFORMAT MATRIX
1640 FOR X=1 TO 20
1650 FOR Y=1 TO 20
1660 IF A(X,Y)=13 GOTO 1700
1670 IF A(X,Y)=12 GOTO 1700
1680 IF A(X,Y)=3 GOTO 1700
1690 A(X,Y)=0:GOTO 1710
1700 A(X,Y)=10
1710 NEXT Y
1720 NEXT X
1730 REM DISPLAY NEW COLONY
1740 FOR X=2 TO 21
1750 PRINT@X,19;SPC(20)
1760 NEXT X
1770 FOR X=1 TO 20
1780 FOR Y=1 TO 20
1790 IF A(X,Y)=10 THEN PRINT@X+1,Y+18;"*"
1800 NEXT Y
1810 NEXT X
1820 PRINT@22,1;SPC(17);@23,1;SPC(17)
1830 PRINT@22,2;"AGAIN  G/K"
1840 GET A$
1850 IF A$="G" GOTO 1530
1860 IF A$="K" GOTO 1880
1870 GOTO 1840
1880 PRINT"@":PRINT@7,1;"I HOPE YOU ENJOYED THE GENERATION GAME."
1890 PRINT@9,1;"WOULD YOU LIKE TO START ANOTHER COLONY?"
1900 PRINT@11,15;"KEY Y OR N"
1910 GET A$:IF A$="Y"GOTO 1935
1920 IF A$="N" GOTO 1940
1930 GOTO 1910
1935 CLR:RESTORE:GOTO 1140
1940 PRINT@14,4;"SO-LONG FOR NOW THEN"
1950 END
```

```
800 REM*PROGRAM WRITTEN IN'EXTENDED SHARP BASIC *
850 REM*BY F.E.WOODWARD,8 VIOLET AV.RAMSGATE,KENT
900 REM*USING BASIC SUBROUTINE EQUIV.TO 'PRINT USING'STATEMENT
950 PRINT"@"
1000 PRINT@3,15;"**********"
1010 PRINT@4;15;"* MOLDAT *"
1020 PRINT@5,15;"**********"
1030 PRINT@8,4;"THS PROGRAM GIVES THE PERCENTAGE"
1040 PRINT@10,3;"COMPOSITION AND MOLECULAR WEIGHT OF"
1050 PRINT@12,3;"COMPOUNDS CONTAINING ANY COMPINATION"
1060 PRINT@14,3;"OF UPTO 10 OF ANY OF THE 30 MOST'
1070 PRINT@16,3;"COMMONLY MET ELEMENTS"
1080 PRINT@21,5;"KEY 'S' WHEN READY TO START"
1090 GET AN$ : IF AN$<>"S" GOTO 1080
1100 PRINT"@"
1110 PRINT@2,2;"ENTER THE CHEMICAL SYMBOL FOR EACH"
1120 PRINT@3,1;"ELEMENT,AND THE NUMBER OF ATOMS OF THAT"
1130 PRINT@4,1;"ELEMENT SEPERATED BY A'COMMA'"
1140 PRINT@5,6;"eg. C,8 or BR,1
1150 PRINT@7,2;"IF THE COMPOUND CONTAINS LESS THAN 10"
1160 PRINT@8,1;"ELEMENTS FINISH WITH '0,0'"
1170 DIM SU$(11,1),CN$(30,1),W(10),P(10)
1175 REM*INITIALISATION OF ALL VARIABLES*
1180 X=0 : MW=0 : FOR N=1 TO 10 : W(N)=0 : NEXT N
1190 FOR N=1 TO 11 : SU$(N,0)=" " : SU$(N,1)=" " : NEXT N
1195 REM*FILLCONSTANTS ARRAY*
1200 FOR N=0 TO 30 : READ CN$(N,0) : READ CN$(N,1) : NEXT N
1205 REM*FILL INPUT ARRAY
1210 FOR N=1 TO 10
1220 X=X+1
1230 PRINT@9+X,1;"ELEMENT";@9+X,8;X;@9+X,13;",ATOMS";:INPUT SU$(N,0),SU$(N,1)
1240 IF SU$(N,0)="0" THEN GOTO 1260
1250 NEXT N
1255 REM*COMPARE INPUT AGAINST CONSTANTS ARRAY*
1260 FOR I=1 TO X-1
1270 FOR J=1 TO 30
1280 IF SU$(I,0)=CN$(J,0) GOTO 1310
1290 NEXT J
1300 PRINT@20,1;"I AM NOT PROGRAMMED TO ACCEPT ELEMENT. "
1305 PRINT@21,10;SU$(I,0) : GOTO 1400
1307 REM*DO CALCULATIONS AND FORMAT OUTPUT*
1310 W(I)=VAL(SU$(I,1))*VAL(CN$(J,1))
1320 MW=MW+W(I)
1330 NEXT I
1340 PRINT@21,23;"MOL.WT.= "
1350 XN=MW : GOSUB 20000 : PRINT@21,35-VX;RN
1360 FOR I=1 TO X-1
1370 PRINT@9+I,26;"%age";@9+I,31;SU$(I,0)
1380 XN=100*W(I)/MW : GOSUB 20000 : PRINT@9+I,36-VX;RN
1390 NEXT I
1400 PRINT@23,1;"ANY MORE FOR ME TO DO (KEY Y or N)": GET AN$
1410 IF AN$="Y" THEN PRINT"@" : RESTORE : GOTO 1180
1420 IF AN$="N" THEN GOTO 21000
1425 GOTO 1400
1430 DATA 0,0,C,12.011,H,1.008,N,14.0067,O,15.999,S,32.064,BR,79.909,CL,35.453
1440 DATA I,126.904,F,18.998,AL,26.981,BA,147.34,B,10.811,CD,112.40,CA,40.08
1450 DATA NI,58.71,P,30.974,K,39.102,SI,28.086,AG,107.870,NA,22.989,SN,118.69
1460 DATA CR,51.996,CO,58.933,CU,63.54,FE,55.847,PB,207.19,MG,24.312,MN,54.938
1470 DATA ZN,65.37,LI,6.939
20000 REM*SUBROUTINE TO ROUND OFF AND FORMAT A NUMBER(XN)
20100 XN$=STR$(XN)
20200 FOR VX=1 TO 8
20300 PT$=MID$(XN$,VX,1)
20400 IF PT$="." GOTO 20700
20500 NEXT VX
20600 XN$=XN$+".00" : GOTO 20200
20700 Z=VAL(MID$(XN$,VX+3,1)) : RN=VAL(LEFT$(XN$,VX+2))
20800 IF Z=>5 THEN RN=RN+0.01
20900 RETURN
21000 PRINT"@":PRINT@5,2;"SO-LONG FOR NOW."
?2000 PRINT@7,2;"I HOPE I CAN HELP YOU AGAIN SOMETIME."
  000 END
```

MORSE TUTOR/T

In this versatile program the MZ-80K becomes a morse code tutor. It enables the user to learn and practice receiving morse code for the full alpha-numeric set (A–Z & 0–9) at a variety of speeds.

Initially the user selects the group of characters he wishes to practice with. A choice of six groups, plus a personal selection, is available. This allows all possible group combinations.

Next transmission speed is selected. The transmission speed may be varied from single character (keyboard input) to 20 words/minute. The computer then transmits five "pips" or "dots" as a "get ready" signal after which the characters are first displayed on the VDU and are then transmitted in random groups of five characters at a time. The sequence is then repeated.

Pressing the "P" key during execution will cause the program to enter the "PAUSE" mode. (The search for "pause request" is made after the five characters have been transmitted. Consequently, and particularly at slow transmission speeds, the P key should be held down until the pause mode is entered). In the pause mode the following commands are available:

Tye C – to CONTINUE with program
Type E – to EXIT from program
Type T – to select a new TRANSMISSION SPEED
Type M – to enter a new CHARACTER MODE and select a new group of characters.

Due to program size Morse Tutor is held as two separate programs in the tape version. The first, "MORSE INS" gives the full set of instructions whilst the second, "MORSE TUTOR' is the program with abbreviated instructions. This allows the user to run the program in a machine where memory space available to programs is limited to 6K and also allows the user to enter the program directly once the instructions are known.

NOTES

```
1 PRINT"."
2 REM A PROGRAM FROM SHARPSOFT
3 REM COPYRIGHT IN EUROPE
5 DIM A$(5)
10 N$=".A":R$="R":TEMPO5
90 T$="3"
92 SR$=R$+T$:LR$=SR$+SR$+SR$
94 DT$=N$+T$+SR$
96 DS$=N$+T$+N$+T$+N$+T$+SR$
100 A$(1)=DS$
110 A$(2)=DT$
120 A$(3)=DT$+DT$
130 A$(4)=DT$+DT$+DT$
140 A$(5)=DT$+DT$+DT$+DT$
200 GOSUB 9000
8100 PRINT"."
8110 PRINT"In this versatile program the MZ-80K"
8120 PRINT"is your morse code trainer. It will"
8130 PRINT"enable you to practice receiving morse"
8140 PRINT"code for the full alpha-numeric set"
8150 PRINT"(A-Z & 0-9) at a variety of speeds.":PRINT
8160 PRINT"Initially you - the user - will select"
8170 PRINT"the group of characters you wish to"
8180 PRINT"practice with. You have a choice of"
8190 PRINT"six groups plus your own special"
8200 PRINT"selection. This allows you all"
8210 PRINT"possible character group combinations":PRINT
8220 PRINT"Next transmission speed is selected."
8230 PRINT"and, after a pause of five pips,"
8240 PRINT"the charaters are transmitted at"
8250 PRINT"random and displayed on the VDU."
8260 PRINT"The sequence is then repeated.":PRINT
8270 PRINT"(Note, however, that in the single"
8280 PRINT"character mode only one character"
8290 PRINT"is transmitted at a time.)"
8291 PRINT
8295 PRINT"PRESS ANY KEY TO CONTINUE"
8297 GETA$:IFA$=""GOTO 8297
8298 PRINT".":FORI=1TO5:MUSICA$(I):NEXTI
8310 PRINT"Pressing the P key during execution"
8320 PRINT"will cause the program to enter the"
8330 PRINT"pause mode. (Hold the P key down "
8340 PRINT"until the pause occurs. The search"
8350 PRINT"for a pause occurs just after the "
8360 PRINT"transmission period )."PRINT:PRINT
8370 PRINT"In the pause mode the following"
8380 PRINT"commands are available:-":PRINT:PRINT
```

```
8390 PRINT"Type C - To CONTINUE with program":PRINT
8400 PRINT"Type E - To EXIT from program":PRINT
8410 PRINT"Type T - To alter TRANSMISSION SPEED":PRINT
8420 PRINT"Type M - To enter a new CHARACTER MODE"
8430 PRINT:PRINT:PRINT"PRESS ANY KEY TO CONTINUE"
8440 GETA$:IFA$=""GOTO8440
8442 PRINT".":FORI=1TO5:MUSICA$(I):NEXTI
8450 PRINT"......Please clear memory (type NEW) and load"
8451 PRINT:PRINT
8452 PRINT"the program MORSETUT from tape."
8460 FORI=1TO2500:NEXT I
8465 PRINT:PRINT:PRINT:PRINT:PRINT
8466 PRINT:PRINT:PRINT:PRINT:PRINT
8470 END
8520 FORI=1TO5
8525 MUSIC A$(1),LR$,LR$,LR$
8530 NEXT I
8535 RETURN
9000 X=SIZE
9001 TEMPO 7
9005 MUSIC A$(2)
9010 FOR X=0TO79:SET X,0:NEXT X
9020 MUSIC A$(3)
9030 FOR Y=0 TO 49
9035 SET 79,Y
9040 NEXT Y
9045 MUSIC A$(4)
9050 FOR X=79 TO 0 STEP -1
9055 SET X,49
9060 NEXT X
9065 MUSIC A$(5)
9070 FOR Y=49 TO 0 STEP -1
9075 SET 0,Y
9080 NEXT Y
9150 PRINT".....";
9200 PRINT"HI THERE ! "
9206 PRINT:PRINT:PRINT:PRINT
9208 PRINT".....";
9210 PRINT"I'M YOUR MORSE CODE TUTOR"
9213 PRINT:PRINT:PRINT
9214 PRINT".....";
9215 MUSIC A$(1)
9217 FOR I=1TO5:MUSIC A$(I):NEXTI
9218 FOR I=1TO2500:NEXTI
9219 FOR I=1TO5:MUSIC A$(I):NEXTI
9220 RETURN
9500 FOR I=1 TO LEN(X$)
```

```
1 REM A PROGRAM FROM SHARPSOFT
2 REM COPYRIGHT IN EUROPE
10 DIM A$(40),B$(40),C$(40),D$(40),E(40)
20 N$=".A":R$="R":TEMPO7:PRINT"."
100 GOTO 2000
1000 SR$=R$+T$:LR$=SR$+SR$+SR$:DT$=N$+T$+SR$
1030 DS$=N$+T$+N$+T$+N$+T$+SR$
1101 A$(1)=DT$+DS$:A$(2)=DS$+DT$+DT$+DT$
1103 A$(3)=DS$+DT$+DS$+DT$:A$(4)=DS$+DT$+DT$
1105 A$(5)=DT$:A$(6)=DT$+DT$+DS$+DT$
1107 A$(7)=DS$+DS$+DT$:A$(8)=DT$+DT$+DT$+DT$
1109 A$(9)=DT$+DT$:A$(10)=DT$+DS$+DS$+DS$
1111 A$(11)=DS$+DT$+DS$:A$(12)=DT$+DS$+DT$+DT$
1113 A$(13)=DS$+DS$:A$(14)=DS$+DT$
1115 A$(15)=DS$+DS$+DS$:A$(16)=DT$+DS$+DS$+DT$
1117 A$(17)=DS$+DS$+DT$+DS$:A$(18)=DT$+DS$+DT$
1119 A$(19)=DT$+DT$+DT$:A$(20)=DS$
1121 A$(21)=DT$+DT$+DS$:A$(22)=DT$+DT$+DT$+DS$
1123 A$(23)=DT$+DS$+DS$:A$(24)=DS$+DT$+DT$+DS$
1125 A$(25)=DS$+DT$+DS$+DS$:A$(26)=DS$+DS$+DT$+DT$
1127 A$(27)=DT$+DS$+DS$+DS$+DS$:A$(28)=DT$+DT$+DS$+DS$+DS$
1129 A$(29)=DT$+DT$+DT$+DS$+DS$:A$(30)=DT$+DT$+DT$+DT$+DS$
1131 A$(31)=DT$+DT$+DT$+DT$+DT$:A$(32)=DS$+DT$+DT$+DT$+DT$
1133 A$(33)=DS$+DS$+DT$+DT$+DT$:A$(34)=DS$+DS$+DS$+DT$+DT$
1135 A$(35)=DS$+DS$+DS$+DS$+DT$:A$(36)=DS$+DS$+DS$+DS$+DS$
1200 FORI=1TO26:C$(I)=CHR$(I+64):NEXTI
1230 FORI=1TO 9:C$(26+I)=CHR$(48+I):NEXTI:C$(36)="0"
1296 X1=SIZE
1299 RETURN
1500 GOSUB 8500
1520 L1=36:L2=1:GOSUB1700
1530 IFA$="M"THEN2000
1540 GOTO 1520
1600 PRINTD$(1);D$(2);D$(3);D$(4);D$(5);"  ";
1605 MUSIC B$(1),P$,B$(2),P$,B$(3),P$,B$(4),P$,B$(5),P$,P$
1610 RETURN
1700 FORI=1TO5
1702 N=INT(L1*RND(1))+L2:B$(I)=A$(N)
1704 D$(I)=C$(N):NEXTI:GOSUB1600
1706 GETA$:IFA$="P"THENGOSUB8000
1708 RETURN
2000 PRINT"."
2001 W1=SIZE
2010 PRINT"Please select mode of operation":PRINT
2015 PRINT"(1) Full alpha-numeric set":PRINT"(2) Alphabet only"
2020 PRINT"(3) Numericals only":PRINT"(4) E,I,S & H"
2025 PRINT"(5) T,M & O":PRINT"(6) Your choice"
2030 PRINT"(7) Single character from keyboard"
2051 GETC:IFC=0GOTO2051
2056 IFC>7GOTO2051
2060 ON C GOTO 1500,2500,3000,4000,5000,6000,7000
2076 GOTO 2000
2400 PRINT",select transmission speed":PRINT
2410 PRINT"(1)  20 WORDS/MIN"
2412 PRINT"(2)  16 WORDS/MIN"
2414 PRINT"(3)  14 WORDS/MIN"
2416 PRINT"(4)  12 WORDS/MIN"
```

```
6220 E(L1)=36:GOTO 6400
6230 E(L1)=H-22
6240 GOTO 6400
6300 L1=L1+1
6310 E(L1)=H-64
6400 NEXT I
6405 FOR I=1 TO 5
6410 N=INT(L1*RND(1))+1
6420 N=E(N)
6430 B$(I)=A$(N):D$(I)=C$(N)
6440 NEXT I:GOSUB1600
6461 GETA$:IFA$="P"GOSUB8000
6462 IFA$="M"GOTO2000
6470 GOTO 6405
7000 GOSUB 2400
7010 GOSUB 1000
7020 PRINT".Enter single character from key
7025 PRINT"Type * to exit from program"
7041 PRINT:PRINT
7045 N1=0
7050 GETA$
7060 IF A$="*" THEN END
7070 IF A$="" GOTO 7050
7100 H=ASC(A$)
7110 IF H<47GOTO7050
7115 IF H>90GOTO7050
7120 IF H<58GOTO7200
7125 IF H>64GOTO7300
7130 GOTO 7050
7200 IFH<>48GOTO7220
7210 B$(1)=A$(36):D$(1)=C$(36):GOTO7400
7220 B$(1)=A$(H-22):D$(1)=C$(H-22)
7230 GOTO7400
7300 B$(1)=A$(H-64):D$(1)=C$(H-64)
7400 N1=N1+1
7410 PRINTD$(1);:MUSIC B$(1)
7420 IFN1/5=INT(N1/5)THEN PRINT"    ";
7430 GOTO7050
8000 GETA$:IFA$="C"THENRETURN
8001 IFA$="M"THENRETURN
8010 IFA$="E"THEN END
8020 IFA$="T"THEN GOTO8040
8030 GOTO 8000
8040 GOSUB 2400
8050 GOSUB 1000
8060 RETURN
8100 PRINT"."
8410 PRINT"Type T - To alter TRANSMISSION SPEED":PRINT
8500 GOSUB 2400
8505 GOSUB 1000
8520 FOR I=1TO5
8525 MUSIC A$(20),LR$,LR$,LR$
8530 NEXT I
8535 RETURN
9220 RETURN
```

```
2418 PRINT"(5)  10 WORDS/MIN"
2420 PRINT"(6)   8 WORDS/MIN"
2422 PRINT"(7)   6 WORDS/MIN"
2424 PRINT"(8)   4 WORDS/MIN"
2426 PRINT"(9)   2 WORDS/MIN"
2428 PRINT:PRINT"Input speed (1-9)
2430 GET T:IFT=0THEN2430
2433 PRINT"."
2434 ON T GOTO 2440,2442,2444,2446,2448,2450,2452,2454,2456
2440 T$="2":P$="R3":RETURN
2442 T$="3":P$="R5":RETURN
2444 T$="3":P$="R6":RETURN
2446 T$="3":P$="R8":RETURNR9":RETURN
2448 T$="3":P$="R9":RETURN
2450 T$="4":P$="R8R5":RETURN
2452 T$="5":P$="R9R6":RETURN
2454 T$="6":P$="R9R9R9R9R9":RETURN
2456 T$="6":P$="R9R9R9R9R9R9R9":RETURN
2500 GOSUB 8500
2520 L1=26:L2=1:GOSUB1700
2530 IFA$="M"THEN2000
2540 GOTO2520
3000 GOSUB 8500
3020 L1=10:L2=27:GOSUB1700
3030 IFA$="M"THEN2000
3040 GOTO 3020
4000 E(1)=5:E(2)=9:E(3)=19:E(4)=8:L1=4
4010 GOTO5020
5000 E(1)=13:E(2)=15:E(3)=20:L1=3
5020 GOSUB8500
5021 FORI=1TO5:N=INT(L1*RND(1))+1
5030 N=E(N):B$(I)=A$(N):D$(I)=C$(N)
5035 NEXT I
5040 GOSUB1600
5050 GETA$:IFA$="P"GOSUB8000
5060 IFA$="M"GOTO2000
5070 GOTO 5021
6000 PRINT".Enter your own character set in the"
6005 PRINT"form: ASD4W etc followed by carriage"
6010 PRINT"return"
6055 PRINT:PRINT
6060 INPUT X$
6065 GOSUB 8500
6070 PRINT"."
6110 L=LEN(X$)
6111 L1=0
6120 FOR I=1 TO L
6130 Y$=MID$(X$,I,1)
6140 H=ASC(Y$)
6150 IF H<47 GOTO 6400
6160 IF H>90 GOTO 6400
6170 IF H<58 GOTO 6200
6180 IF H>64 GOTO 6300
6190 GOTO 6400
6200 L1=L1+1
6210 IF H<>48 GOTO 6230
```

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