Download a plain-text copy of this document, as originally formatted for Courier

I T T Y B I T T Y C O M P U T E R S

TINY BASIC User Manual

Congratulations! You have received the first of what we hope is a long line of low cost software for hobby computers. We are operating on a low margin basis, and hope to make a profit on volume. Please help us to stay in business by respecting the Copyright notices on the software and documentation.

If you are in a hurry to try TINY BASIC, Appendix C will tell you how to get on the air. Then come back and read the rest of this manual --- most of it is useful information.

The TINY BASIC interpreter program has been extensively tested for errors ("bugs"), but it would be foolish to claim of any program that it is guaranteed bug-free. This program does come with a "Limited Warranty" in that any errors discovered will be corrected in the first 90 days. Catastrophic bugs will be corrected by automatically mailing out corrected versions to all direct mail customers and local dealers. Minor bugs will be corrected by request. In any case this warranty is limited to replacement of the Program Tape and/or documentation, and no liability for consequential damages is implied.

If you think you have found a bug, make a listing of the program that demonstrates the bug, together with the run input and output. Indicate on the listing what you think is wrong and what version number you are running and your serial number (on the tape leader). Mail this to:

ITTY BITTY COMPUTERS
P.0. Box 6539
San Jose, CA 95150

We will try to be responsive to your needs.

TINY BASIC was conceived by the dragons at the People's Computer Company (PCC), a non-profit corporation in Menlo Park CA. and its implementation defined by Dennis Allison and others in the PCC newspaper and an offshoot newsletter. The implementation of this program follows the philosophy defined there. The reader is referred to PCC v.4 Nos 1-3 for a discussion of the inner workings of this software.

In keeping with the "small is good" philosophy, TINY BASIC employs the two level interpreter approach (with its consequent speed cost) so that the whole system occupies only 2K of program memory (exclusive of user program; some versions are slightly larger). With 1K of additional RAM small but useful user programs (50 lines or less) may be accommodated. A system with 4K of RAM can contain the interpreter and about 100 lines of user program.

TINY BASIC is offered in several versions for each processor. One is designed to be used with an arbitrary operating system, and executes out of low memory (e.g. 0100-08FF for the 6800). The other versions are configured for unusual memory requirements of particular operating systems. All are "clean" programs, in that they will execute properly from protected memory (such as PROM). Direct addressing is used for interpreter variables as much as possible, so memory Page 00 is largely dedicated. In all cases the user programs are placed at the end of that part of lower memory used by TINY, and they may occupy all the remaining contiguous memory. Appendix D is a a summary of the important low-memory addresses.

TINY BASIC is designed to be I/O independent, with all input and output funneled through three jumps placed near the beginning of the program. In the non-standard versions these are preset for the particular operating system I/O, so the discussion to follow is primarily concerned with the standard versions. For this discussion, it is assumed that the interpreter begins at hex address 0100, though the remarks may be applied to other versions with an appropriate offset.

Location 0106 is a JMP to a subroutine to read one ASCII character from the console/terminal. Location 0109 is a JMP to a subroutine to type or display one ASCII character on the console/terminal. In both cases the character is in the A accumulator, but the subroutine need not preserve the contents of the other registers. It is assumed that the character input routine will simultaneously display each character as it is input; if this is not the case, the JMP instruction in location 0106 may be converted to a JSR, so that each character input flows through the output subroutine (which in this case must preserve A) before being fed to TINY. Users with terminals using Baudot or some other non-ASCII code should perform the character conversion in the Input and Output subroutines. If your console is a CRT and/or you have no need to output or display extra pad characters with each Carriage Return and Linefeed, you may intercept these in the output routine to bypass their display. Each input prompt by TINY is followed by an "X-ON" character (ASCII DC1) with the sign bit set to 1 (all other characters except rubout are output with the sign bit set to 0) so these are also readily detected and deleted from the output stream. Appendix C shows how to perform these tests.

A third subroutine provided by you is optional, and gives TINY a means to test for the BREAK condition in your system. Appendix C shows how this subroutine may be implemented for different types of I/O devices. If you choose to omit this subroutine, TINY will assume that a BREAK condition never happens; to include it, simply replace locations 010C-010E with a JMP to your subroutine, which returns with the break condition recorded in the Carry flag (1 = BREAK, 0 = no BREAK). The Break condition is used to interrupt program execution, or to prematurely terminate a LIST operation. Tiny responds to the Break condition any time in the LIST, or just before examining the next statement in program execution. If a LIST statement included within a program is aborted by the Break condition, the Break condition must be held over to the next statement fetch (or repeated) to stop program execution also.

All input to Tiny is buffered in a 72 character line, terminated by a Carriage Return ("CR"). Excess characters are ignored, as signaled by ringing the console/terminal bell. When the CR is typed in, Tiny will echo it with a Linefeed, then proceed to process the information in the line. If a typing error occurs during the input of either a program line or data for an INPUT statement, the erroneous characters may be deleted by "backspacing" over them and retyping. If the entire line is in error, it may be canceled (and thus ignored) by typing the "Cancel" key. The Backspace code is located near the beginning of the program (location 010F), and is set by default to "left-arrow" or ASCII Underline (shift-O on your Teletype). To change this to the ASCII Standard Backspace code (or anything else you choose), the contents of location 010F may be changed to the desired code. Similarly the Cancel code is located at memory address 0110, and is set by default to the ASCII Cancel code (Control-X). Four characters which may not be used for line edits (Backspace or Cancel) are DC3 (hex 13), LF (0A), NUL (00), and DEL (FF). These codes are trapped by the TINY BASIC input routines before line edits are tested.

When Tiny ends a line (either input or output), it types a CR, two pad characters, a Linefeed, and one more pad character. The pad character used is defined by the sign bit in location 0111, and is set by default to the "Rubout" or Delete code (hex FF; Location 0111 Bit 7 = 1) to minimize synchronization loss for bit-banger I/O routines. The pad character may be changed to a Null (hex 00) by setting the sign of location 0111 to 0. The remainder of this byte defines the number of Pad characters between the CR and linefeed. More than two pad characters may be required if large user programs are to be loaded from tape (see comments on

Tape Mode, below).

TINY BASIC has a provision for suppressing output (in particular line prompts) when using paper tape for loading a program or inputting data. This is activated by the occurrence of a Linefeed in the input stream (note that the user normally has no cause to type a Linefeed since it is echoed in response to each CR), and disables all output (including program output) until the tape mode is deactivated. This is especially useful in half-duplex I/O systems such as that supported by Mikbug, since any output would interfere with incoming tape data. The tape mode is turned off by the occurrence of an X-OFF character (ASCII DC3, or Control-S) in the input, by the termination of an executing program due to an error, or after the execution of any statement or command which leaves Tiny in the command mode. The tape mode may be disabled completely by replacing the contents of memory location 0112 with a 00.

Memory location 0113 is of interest to those 6800 users with extensive operating systems. Normally Tiny reserves 32 bytes of stack space for use by the interpreter and I/O routines (including interrupts). Up to half of these may be used by Tiny in normal operation, leaving not more than 16 bytes on the stack for I/O. If your system allows nested interrupts or uses much more than ten or twelve stack bytes for any purpose, additional space must be allocated on the stack. Location 0113 contains the reserve stack space parameter used by Tiny, and is normally set to 32 (hex 20). If your system requires more reserve, this value should be augmented accordingly before attempting to run the interpreter.

All of these memory locations are summarized in Appendix D. Note that there are no Input or Output instructions or interrupt disables in the interpreter itself; aside from the routines provided for your convenience (which you may connect or disconnect), your system has complete control over the I/O and interrupt structure of the TINY BASIC environment.

TINY BASIC is designed to use all of the memory available to it for user programs. This is done by scanning all the memory from the beginning of the user program space (e.g. 0900 for the standard 6800 version) for the end of contiguous memory. This then becomes the user program space, and any previous contents may be obliterated. If it is desired to preserve some part of this memory for machine language subroutines or I/O routines, it will be necessary to omit the memory scan initialization. This is facilitated in TINY BASIC by the definition of two starting addresses. Location 0100 (or the beginning of the interpreter) is the "Cold Start" entry point, and makes no assumptions about the contents of memory, except that it is available. Location 0103 is the "Warm Start" entry point, and assumes that the upper and lower bounds of the user program memory have been defined, and that the program space is correctly formatted. The Warm Start does not destroy any TINY BASIC programs in the program space, so it may be used to recover from catastrophic failures. The lower bound is stored in locations 0020-0021 and the upper bound is in locations 0022-0023. When using the Warm Start to preserve memory, you should be sure these locations contain the bounds of the user space. Also when using the Warm Start instead of the Cold Start, the first command typed into TINY should be "CLEAR" to properly format the program space.

STATEMENTS

TINY BASIC is a subset of Dartmouth BASIC, with a few extensions to adapt it to the microcomputer environment. Appendix B contains a BNF definition of the language; the discussion here is intended to enable you to use it. When TINY issues a line prompt (a colon on the left margin) you may type in a statement with or without a line number. If the line number is included, the entire line is inserted into the user program space in line number sequence, without further analysis. Any previously existing line with the same line number is deleted or replaced by the new line. If the new line consists of a line number only, it is considered a deletion, and nothing is inserted. Blanks are not significant to TINY, so blanks imbedded in the line number are ignored; however, after the first non-blank, non-numeric character in the line, all blanks are preserved in memory.

The following are valid lines with line numbers!

```
123 PRINT "HELLO"
456    G O T O 1 2 3
7 8 9 PRINT "THIS IS LINE # 789"
123
32767 PRINT "THIS IS THE LARGEST LINE #"
1PRINT"THIS, IS THE SMALLEST LINE #"
10000 TINY BASIC DOES NOT CHECK
10001 FOR EXECUTABLE STATEMENTS ON INSERTION.
```

0 Is not a valid line number.

If the input line does not begin with a line number it is executed directly, and must consist of one of the following statement types:

```
LET             GOTO            REM
IF...THEN       GOSUB           CLEAR
INPUT           RETURN          LIST
PRINT           END             RUN
```

These statement types are discussed in more detail in the pages to follow.

Note that all twelve statement types may be used in either the Direct Execution mode (without a line number) or in a program sequence (with a line number). Two of the statements (INPUT and RUN) behave slightly differently in these two operating modes, but otherwise each statement works the same in Direct Execution as within a program. Obviously there is not much point in including such statements as RUN or CLEAR in a program, but they are valid. Similarly, a GOSUB statement executed directly, though valid, is likely to result in an error stop when the corresponding RETURN statement is executed.

EXPRESSIONS

Many of these statement types involve the use of EXPRESSIONS. An Expression is the combination of one or more NUMBERS or VARIABLES, joined by OPERATORS, and possibly grouped by Parentheses. There are four Operators:

+ addition
- subtraction
* multiplication
/ division

These are hierarchical, so that in an expression without parentheses, multiplication and division are performed before addition and subtraction. Similarly, sub-expressions within parentheses are evaluated first. Otherwise evaluation proceeds from left to right. Unary operators (+ and -) are allowed in front of an expression to denote its sign.

A Number is any sequence of decimal digits (0, 1, 2, ... 9), denoting the decimal number so represented. Blanks have no significance and may be imbedded within the number for readability if desired, but commas are not allowed. All numbers are evaluated as 16-bit signed numbers, so numbers with five or more digits are truncated modulo 65536, with values greater than 32767 being considered negative. The following are some valid numbers (note that the last two are equivalent to the first two in TINY):

```
0
100
10 000
1 2 3 4
32767
65536
65 636
```

A Variable is any Capital letter (A, B, ... Z). This variable is assigned a fixed location in memory (two bytes, the address of which is twice the ASCII representation of the variable name). It may assume any value in the range, -32768 to +32767, as assigned to it by a LET or INPUT statement.

The following are some examples of valid expressions:

```
A
123
1+2-3
B-14*C
(A+B)/(C+D)
-128/(-32768+(I*1))
(((((Q)))))
```

All expressions are evaluated as integers modulo 65536. Thus an expression such as

N / P * P may not evaluate to the same value as (N), and in fact this may be put to use to determine if a variable is an exact multiple of some number. TINY BASIC also makes no attempt to discover arithmetic overflow conditions, except in the case of an attempt to divide by zero (which results in an error stop). Thus all of the following expressions evaluate to the same value:

```
-4096
15*4096
32768/8
30720+30720
```

TINY BASIC allows two intrinsic functions. These are:

`RND (range)`

`USR (address,Xreg,Areg)`

Either of these functions may be used anywhere an (expression) is appropriate.

FUNCTIONS

```
RND (range)
```

This function has as its value, a positive pseudo-random number between zero and range-1, inclusive. If the range argument is zero an error stop results.

```
USR (address)
USR (address,Xreg)
USR (address,Xreg,Areg)
```

This function is actually a machine-language subroutine call to the address in the first argument. If the second argument is included the index registers contain that value on entry to the subroutine, with the most significant part in X. If the third argument is included, the accumulators contain that value on entry to the subroutine, with the least significant part in A. On exit, the value in the Accumulators (for the 6800; A and Y for the 6502) becomes the value of the function, with the least significant part in A. All three arguments are evaluated as normal expressions.

It should be noted that machine language subroutine addresses are 16-bit Binary numbers. TINY BASIC evaluates all expressions to 16-bit binary numbers, so any valid expression may be used to define a subroutine address. However, most addresses are expressed in hexadecimal whereas TINY BASIC only accepts numerical constants in decimal. Thus to jump to a subroutine at hex address 40AF, you must code USR(16559). Hex address FFB5 is similarly 65461 in decimal, though the equivalent (-75) may be easier to use.

For your convenience two subroutines have been included in the TINY BASIC interpreter to access memory. If S contains the address of the beginning of the TINY BASIC interpreter (256 for standard 6800, 512 for standard 6502, etc.), then location S+20 (hex 0114) is the entry point of a subroutine to read one byte

from the memory address in the index register, and location S+24 (hex 0118) is the entry point of a subroutine to store one byte into memory.

Appendix E gives examples of the USR function.

STATEMENT TYPES

PRINT print-list
PR print-list

This statement prints on the console/terminal the values of the expressions and/or the contents of the strings in the print-list. The print-list has the general form,

item,item... or item;item...

The items may be expressions or alphanumeric strings enclosed in quotation marks (e.g. "STRING"). Expressions are evaluated and printed as signed numbers; strings are printed as they occur in the PRINT statement. When the items are separated by commas the printed values are justified in columns of 8 characters wide; when semicolons are used there is no separation between the printed items. Thus,

```
PRINT 1,2,3
```

prints as

```
1       2       3
```

and

```
PRINT 1;2;3
```

prints as

```
123
```

Commas and semicolons, strings and expressions may be mixed in one PRINT statement at will.

If a PRINT statement ends with a comma or semicolon TINY BASIC will not terminate the output line so that several PRINT statements may print on the same output line, or an output message may be printed on the same line as an input request (see INPUT). When the PRINT statement does not end with a comma or semicolon the output is terminated with a carriage return and linefeed (with their associated pad characters). To aid in preparing data tapes for input to other programs, a colon at the end of a print-list will output an "X-OFF" control character just before the Carriage Return.

Although the PRINT statement generates the output immediately while scanning the statement line, output lines are limited to 125 characters, with excess suppressed.

While the Break key will not interrupt a PRINT statement in progress, the Break condition will take effect at the end of the current PRINT statement.

The following are some examples of valid PRINT statements:

```
PRINT "A=";A,"B+C=";B+C
PR                      (one blank line)
PRI                      (prints the value of I)
PRINT 1,","Q*P;",",R/42:
```

INPUT input-list

This statement checks to see if the current input line is exhausted. If it is, a question mark is prompted with an X-ON control character, and a new line is read in. Then or otherwise, the input line is scanned for an expression which is evaluated. The value thus derived is stored in the first variable in the input-list. If there are more variables in the input-list the process is repeated. In an executing program, several values may be input on a single request by separating them with commas. If these values are not used up in the current

INPUT statement they are saved for subsequent INPUT statements. The question mark is prompted only when a new line of input values is required. Note that each line of input values must be terminated by a carriage return. Since expressions may be used as input values, any letter in the input line will be interpreted as the value of that variable. Thus if a program sets the value of A to 1, B to 2, and C to 3, and the following statement occurs during execution:

```
INPUT X,Y,Z
```

and the user types in

```
A,C,B
```

the values entered into X, Y, and Z will be 1, 3, and 2, respectively, just as if the numbers had been typed in. Note also that blanks on the input line are ignored by TINY, and the commas are required only for separation in cases of ambiguity. In the example above

```
ACB
```

could have been typed in with the same results. However an input line typed in as

```
+1  -3  +6   0
```

will be interpreted by TINY as a single value (=58) without commas for separators. There is one anomaly in the expression input capability: if in response to this INPUT, the user types,

```
RND+3
```

TINY will stop on a bad function syntax error (the RND function must be of the form, RND(x)); but if the user types,

```
RN,D+3
```

the values in the variables R, N, and the expression (D+3) will be input. This is because in the expression evaluator the intrinsic function names are recognized before variables, as long as they are correctly spelled.

Due to the way TINY BASIC buffers its input lines, the INPUT statement cannot be directly executed for more than one variable at a time, and if the following statement is typed in without a line number,

```
INPUT A,B,C
```

the value of B will be copied to A, and only one value (for C) will be requested from the console/terminal. Similarly, the statement,

```
INPUT X,1,Y,2,Z,3
```

will execute directly (loading X, Y, and Z with the values 1, 2, and 3), requesting no input, but with a line number in a program this statement will produce an error stop after requesting one value.

If the number of expressions in the input line does not match the number of variables in the INPUT statement, the excess input issaved for the next INPUT statement, or another prompt is issued for more data. The user should note that misalignment in these circumstances may result in incorrect program execution (the wrong data to the wrong variables). If this is suspected, data entry may be typed in one value at a time to observe its synchronization with PRINT statements in the program.

There is no defined escape from an input request, but if an invalid expression is typed (such as a period or a pair of commas) an invalid expression error stop will occur.

Because Tiny Basic does not allow arrays, about the only way to process large volumes of data is through paper tape files. Each input request prompt consists of a question mark followed by an X-ON (ASCII DC1) control character to turn on an automatic paper tape reader on the Teletype (if it is ready). A paper tape may be prepared in advance with data separated by commas, and an X-OFF (ASCII DC3 or Control-S) control character preceding the CR (a Teletype will generally read at least one more character after the X-OFF). In this way the tape will feed one line at a time, as requested by the succession of INPUT statements. This tape may also be prepared from a previous program output (see the PRINT statement).

```
LET var = expression
var = expression
```

This statement assigns the value of the expression to the variable (var). The long form of this statement (i.e. with the keyword LET) executes slightly faster than the short form. The following are valid LET statements:

```
LET A = B+C
I = 0
LET Q = RND (RND(33)+5)
```

GOTO expression

The GOTO statement permits changes in the sequence of program execution. Normally programs are executed in the numerical sequence of the program line numbers, but the next statement to be executed after a GOTO has the line number derived by the evaluation of the expression in the GOTO statement. Note that this permits you to compute the line number of the next statement on the basis of program parameters during program execution. An error stop occurs if the evaluation of the expression results in a number for which there is no line. If a GOTO statement is executed directly, it has the same effect as if it were the first line of a program, and the RUN statement were typed in, that is, program execution begins from that line number, even though it may not be the first in the program. Thus a program may be continued where it left off after correcting the cause of an error stop. The following are valid GOTO statements:

```
GOTO 100
GO TO 200+I*10
G 0 T 0 X
```

GOSUB expression

The GOSUB statement is like the GOTO statement, except that TINY remembers the line number of the GOSUB statement, so that the next occurrence of a RETURN statement will result in execution proceeding from the statement following the GOSUB. Subroutines called by GOSUB statements may be nested to any depth, limited only by the amount of user program memory remaining. Note that a GOSUB directly executed may result in an error stop at the corresponding RETURN. The following are some examples of valid GOSUB statements:

```
GOSUB 100
GO SUB 200+I*10
```

RETURN

The RETURN statement transfers execution control to the line following the most recent unRETURNed GOSUB. If there is no matching GOSUB an error stop occurs.

IF expression rel expression **THEN** statement
IF expression rel expression statement

The IF statement compares two expressions according to one of six relational operators. If the relationship is True, the statement is executed; if False, the associated statement is skipped. The six relational operators are:

=	equality
<	less than
>	greater than
<=	less or equal (not greater)
>=	greater or equal (not less)
<>, ><	not equal (greater or less)

The statement may be any valid TINY BASIC statement (including another IF statement). The following are valid IF statements:

```
IF I>25 THEN PRINT "ERROR"
IF N/P*P=N GOTO 100
```

```
IF 1=2 Then this is nonsense
IF RND (100) > 50 THEN IF I <> J INPUT Q,R
```

END

The END statement must be the last executable statement in a program. Failure to include an END statement will result in an error stop after the last line of the program is executed. The END statement may be used to terminate a program at any time, and there may be as many END statements in a program as needed. The END statement also clears out any saved GOSUB line numbers remaining, and may be used for that purpose in the direct execution mode.

REM comments

The REM statement permits comments to be interspersed in the program. Its execution has no effect on program operation, except for the time taken.

CLEAR

The CLEAR statement formats the user program space, deleting any previous programs. If included in a program (i.e. with a line number) the program becomes suicidal when the statement is executed, although no error results. If the Warm Start is used to initialize the interpreter, this must be the first command given.

RUN
RUN,expression-list

The RUN statement is used to begin program execution at the first (lowest) line number. If the RUN statement is directly executed, it may be followed by a comma, followed by values to be input when the program executes an INPUT statement.

If the RUN statement is included in a program with a line number, its execution works like a GO TO first statement of the program.

LIST
LIST expression
LIST expression,expression

The LIST statement causes part or all of the user program to be listed. If no parameters are given, the whole program is listed. A single expression parameter in evaluated to a line number which, if it exists, is listed. If both expression parameters are given, all of the lines with line numbers between the two values (inclusive) are listed. If the last expression in the LIST statement evaluates to a number for which there is no line, the next line above that number which does exist (if any) is listed as the last line. Zero is not a valid line number, and an error stop will occur if one of the expressions evaluates to zero. A LIST statement may be included as part of the program, which may be used for printing large text strings such as instructions to the operator. A listing may be terminated by the Break key.

If the terminal punch (or cassette recorder) is turned on for a LIST operation, the tape may be saved to reload the program into TINY at a later time.

The following are valid LIST statements:

```
LIST
LIST 75+25           (lists line 100)
LIST 100,200
LIST 500,400         (lists nothing)
```

Appendix A

ERROR MESSAGE SUMMARY

0 Break during execution
8 Memory overflow; line not inserted
9 Line number 0 not allowed
13 RUN with no program in memory
18 LET is missing a variable name
20 LET is missing an =
23 Improper syntax in LET
25 LET is not followed by END
34 Improper syntax in GOTO
37 No line to GO TO
39 Misspelled GOTO
40,41 Misspelled GOSUB
46 GOSUB subroutine does not exist
59 PRINT not followed by END
62 Missing close quote in PRINT string
73 Colon in PRINT is not at end of statement
75 PRINT not followed by END
95 IF not followed by END
104 INPUT syntax bad - expects variable name
123 INPUT syntax bad - expects comma
124 INPUT not followed by END
132 RETURN syntax bad
133 RETURN has no matching GOSUB
134 GOSUB not followed by END
139 END syntax bad
154 Can't LIST line number 0
164 LIST syntax error - expects comma
183 REM not followed by END
184 Missing statement type keyword
186 Misspelled statement type keyword
188 Memory overflow: too many GOSUB's ...
211 ... or expression too complex
224 Divide by 0
226 Memory overflow
232 Expression too complex ...
233 ... using RND ...
234 ... in direct evaluation;
253 ... simplify the expression
259 RND (0) not allowed
266 Expression too complex ...
267 ... for RND
275 USR expects "(" before arguments
284 USR expects ")" after arguments
287 Expression too complex ...
288 ... for USR

290 Expression too complex
293 Syntax error in expression - expects value
296 Syntax error - expects ")"
298 Memory overflow (in USR)
303 Expression too complex (in USR)
304 Memory overflow (in function evaluation)
306 Syntax error - expects "(" for function arguments
330 IF syntax error - expects relation operator

Other error message numbers may possibly occur if the interpreter is malfunctioning. If this happens, check the program in memory, or reload it, and try again.

Error number 184 may also occur if TINY BASIC is incorrectly interfaced to the keyboard input routines. A memory dump of the input line buffer may disclose this kind of irregularity.

Appendix B

FORMAL DEFINITION OF TINY BASIC

```
line ::= number statement CR
         statement CR
statement ::= PRINT printlist
              PR printlist
              INPUT varlist
              LET var = expression
              var = expression
              GOTO expression
              GOSUB expression
              RETURN
              IF expression relop expression THEN statement
              IF expression relop expression statement
              REM commentstring
              CLEAR
              RUN
              RUN exprlist
              LIST
              LIST exprlist
printlist ::=
              printitem
              printitem :
              printitem separator printlist
printitem ::= expression
              "characterstring"
varlist ::= var
           var , varlist
exprlist ::= expression
             expression , exprlist
expression ::= unsignedexpr
               + unsignedexpr
               - unsignedexpr
unsignedexpr ::= term
                 term + unsignedexpr
                 term - unsignedexpr
term ::= factor
         factor * term
         factor / term
```

```
factor ::= var
           number
           ( expression )
           function
function ::= RND ( expression )
             USR ( exprlist )
number ::= digit
           digit number
separator ::= , ! ;
var ::= A ! B ! ... ! Y ! Z
digit ::= 0 ! 1 2 ! ... ! 9
relop ::= < ! > ! = ! <= ! >= ! <> ! ><
```

Appendix C

IMPLEMENTING I/O ROUTINES

COSMAC

COSMAC TINY occupies the same space as 6800 TINY -- 0100-08FF. Similarly, the general parameters occupy 0020-00B7, as defined in the manual. However, COSMAC TINY also uses locations 0011-001F to contain copies of interpreter parameters and other run-time data; do not attempt to use these locations while running TINY.

Like all Itty Bitty Computer software, COSMAC TINY contains no I/O instructions (nor references to Q or EF1-4), no interrupt enables or disables, and no references to an operating system. The three jumps (LBR instructions) at 0106, 0109, and 010C provide all necessary I/O, as defined in the manual. If you are using UT3 or UT4, you may insert the following LBR instructions, which jump to the necessary interface routines:

```
                    ..  LINKS TO UT3/4
0106 C0076F                 LBR UTIN
0109 C00776                 LBR UTOUT
010C C00766                 LBR UTBRK
```

If you are not using the RCA monitor, you must write your own I/O routines. For this the standard subroutine call and return linkages are used, except that D is preserved through calls and returns by storing it in RF.1. Registers R2-RB and RD contain essential interpreter data, and if the I/O routines make any use of any of them they should be saved and restored. Note however, that R2-R6 are defined in the customary way and may be used to nest subroutine calls if needed. R0, R1, RC, RE and RF are available for use by the I/O routines, as is memory under R2. Both the call and return linkages modify X and the I/O data character is passed in the accumulator ("D", not RD).

After connecting TINY to the I/O routines, start the processor at 0100 (the Cold Start). Do not attempt to use the Warm Start without entering the Cold Start at least once to set up memory from 0011-0023. Any register may be serving as program counter when entering either the Cold Start or the Warm Start.

The USR function works the same way as described in the manual, except that the second argument in the call is loaded into R8, and the third argument is loaded into RA with the least significant byte also in the Accumulator. On return RA.1 and the accumulator contain the function value (RA.0 is ignored). The machine language subroutine must exit by a SEP R5 instruction. USR machine language subroutines may use R0, R1, R8, RA, RC-RF, so long as these do not conflict with I/O routine assignments. TINY BASIC makes no internal use of R0, R1, RC, or RE.

RCA Corporation funded the development of COSMAC TINY BASIC, and it is by RCA's permission that it is made available.

If you do not have access to a monitor in ROM with ASCII I/O built in, you will have to write your own I/O routines. Most likely you have something connected to a parallel port for the keyboard input; output may be some parallel port also, or you may want to use the 1861 video display for a gross dot-matrix kind of text display. For the moment, let's assume you have parallel ports, Port C (N=1100) for input, and port 4 (N=0100) for output. Assume also that EF4 controls both input and output. This is the situation you would have if you took an ordinary ELF and used the hex input and display with the single "input" button to step through the characters. You need for this configuration, two routines, which might look something like this:

```
0106 C0 00E0    LBR KEYIN
0109 C0 00E7    LBR DISPL
...
00E0 3FE0 KEYIN BN4 *
00E2 E2         SEX 2
00E3 6C         INP 4
00E4 37E4       B4  *
00E6 68         LSKP
00E7 3FE7 DISPL BN4 *
00E9 E2         SEX 2
00EA 73         STXD
00EB 52         STR 2
00EC 64         OUT 4
00ED 37ED       B4  *
00EF D5         SEP 5
```

Of course if you have a keyboard on Port F you will change the INP instruction to match; if the keyboard pulls EF3 down, then you must change the first pair of BN4/B4 instructions to BN3/B3 instructions and change the LSKP to a NOP (C4 or E2). If your input comes from some device that already displayed the character typed, then change the LSKP to a Return (D5).

Similarly, if the output is to a different port you must change the OUT instruction to fit it, and the second pair of BN4/B4 instructions to match the control line being used. Notice that the LSKP instruction is only there to prevent your waiting on the EF4 line twice for each keyin, and should be removed (changed to a NOP) as soon as you connect up real input and output ports.

Many 1802 systems come equipped with a video output using the 1861 chip. If you have this, you should get a copy of the February and April 1979 issues of KILOBAUD MICROCOMPUTING (formerly just KILOBAUD). I have a two-part article published in these two issues which explains how to put text on the 1861 graphics display, with particular emphasis on how to connect it to TINY BASIC.

So far I have not mentioned the Break test. If you leave that part unchanged, Tiny will work just fine, but you cannot stop a listing or a program that is getting too long. After you get your keyboard and display connected up and working, you may want to use EF4 (or some other flag) as a Break input. It is possible to use the same flag for Break as for "input ready", if you want Tiny to stop executing when you press a key on your keyboard (this does not affect the INPUT instruction, which is obviously waiting for that keyin). This code will do that:

```
010C C000F0     LBR BRKT
...
00F0 FC00  BRKT ADI 0
00F2 3FF6       BN4 EXIT
00F4 FF00       SMI 0
00F6 D5    EXIT SEP R5
```

Notice that the only function of this routine is to set the Carry (DF) when EF4 is true (low) and clear it otherwise.

KIM

The Teletype I/O routines in the MOS Technology KIM system may be used for the character input and

output requirements of TINY BASIC 6502. The following break routine is included in Tiny to test the serial data line at 1740; Since TINY BASIC 6502 does not use the lower part of memory page 01, the break test routine is ORG'ed to execute in that space:

```
            ;  BREAK TEST FOR KIM
0100 AD4017 KIMBT LDA KTTY      LOOK AT TTY
0103 18           CLC           C=0 IF IDLE
0104 300E         BMI KIMX      IDLE
0106 AD4017       LDA KTTY      WAIT FOR END
0109 10FB         BPL *-3
010B 200E01 KLDY  JSR *+3
010E A9FF         LDA #255      DELAY 2 RUBOUT TIMES
0110 20A01E       JSR OUTCH
0113 38           SEC           C=1 IF BREAK
0114 60     KIMX  RTS
```

To run TINY BASIC 6502 load the paper tape into your Teletype reader, type "L", and turn on the reader. Then key in the following Jumps:

```
            ;  JUMPS TO KIM
0206 4C5A1E     JMP GETCH       CHARACTER INPUT
0209 4CA01E     JMP OUTCH       CHARACTER OUTPUT
020C 4C0001     JMP KIMBT       BREAK TEST
```

It is recommended that you save a copy of memory on tape (0100-0114 and 0200-0AFF) before going any further. Or you may prefer to save it on audio cassette. Set up the starting address for Tiny at 0200, and type "G".

Because of the awkwardness of putting memory in the 4K gap left in the KIM-1 system, an alternate version is available which executes out of 2000-28FF. For this version the Cold Start is at 2000 and other addresses are at 200x instead of 020x (cf. 010x in Appendix D).

JOLT or TIM

JOLT systems may not always have memory loaded in the space from 0200 on up, so a special version has been prepared in which the interpreter resides in the space 1000-18FF. This is the only difference between the JOLT version and the KIM version, so if your JOLT or TIM system has contiguous memory from Page 00 you may prefer to use the KIM version to gain the extra memory space. Since the serial data in the JOLT/TIM systems is not the same as KIM, a special break test routine has also been provided for those systems:

```
        ; JOLT BREAK TEST
0115 A901   JOLBT LDA #1        LOOK AT TTY
0117 2C026E       BIT JTTY
011A 18           CLC           C=0 IF IDLE
011B F00E         BEQ JOLTX     IDLE
011D 2C026E       BIT JTTY      WAIT FOR END
0120 D0FB         BNE *-3
0122 202501       JSR *+3       DELAY TWO CH TIMES
0125 A9FF         LDA #255
0127 20C672       JSR WRT
012A 38           SEC           C=1 = BREAK
012B 60     JOLTX RTS
```

To run, load the paper tape into your Teletype reader and type "LH". Then key in the following Jumps:

```
            ;  JUMPS TO JOLT/TIM
1006 4CE972     JMP RDT         CHARACTER INPUT
1009 4CC672     JMP WRT         CHARACTER OUTPUT
100C 4C1501     JMP JOLBT       BREAK TEST
```

As with other versions, the Cold start is the beginning of the program (1000).

MIKBUG

Systems that use MIKBUG (TM Motorola) for console I/O may use the I/O routines in MIKBUG. The following break routine is provided in Tiny to test the PIA at 8004:

```
        *  BREAK TEST FOR MIKBUG

B68004  BREAK   LDA A PIAA      LOOK AT PIA
0C              CLC             C=0 IF NONE
2B0D            BMI EXIT
B68004          LDA A PIAA
2AFB            BPL *-3         WAIT FOR END
8D00            BSR *+2
86FF            LDA A #$FF      DELAY ONE
BD0109          JSR TYPE        CHARACTER TIME
0D              SEC             C=1 IF BREAK
39      EXIT    RTS
```

To run, load the paper tape into your Teletype reader and type "L". Then key in the following Jumps:

```
            *  JUMPS TO MIKBUG
                ORG $0106
0106 7EE1AC     JMP $E1AC       CHARACTER INPUT
0109 7EE1D1     JMP $E1D1       CHARACTER OUTPUT
010C 7E08FD     JMP $08FD       BREAK TEST
```

It is recommended that you save a copy of memory on tape (0100-08FF) before going any further. Set the starting address in A048-A049 to 0100 and type "G". For your convenience the Cold Start entry leaves the Warm start entry set up in the Mikbug stack, so that after a reset a simple "G" command will result in a Warm start and preserve the user programs.

OTHER

For standard systems (and for special systems with I/O other than that provided), subroutines must be supplied by the user to interface TINY to the operator. For ACIA input or output the following routines may be used, or they may serve as examples for your coding (6800 opcodes are shown). They should be assembled for your ACIA address, and in some memory location which is not contiguous with the TINY BASIC user program memory (which may be destroyed by the Cold Start). If nothing else is available, locations 00D8-00FF are not used by Tiny and may be used for this purpose.

```
        *
        *  ACIA I/O
        *
B6XXXX BREAK    LDA A ACIA
47              ASR A           CHECK FOR TYPEIN
2406            BCC BRX         NO, NOT BREAK
B6XXXY          LDA A ACIA+1    GET IT
2601            BNE BRX         NOT NULL IS BREAK
0C              CLC             IGNORE NULLS
39      BRX     RTS
B6XXXX INPUT    LDA A ACIA
47              ASR A
24FA            BCC INPUT       WAIT FOR A CHARACTER
B6XXXY          LDA A ACIA+1    GET IT
36      OUTPUT  PSH A           SAVE CHARACTER
```

```
B6XXXX          LDA A ACIA
8402            AND A #2        WAIT FOR READY
27F9            BEQ OUTPUT+1
32              PUL A
B7XXXY          STA A ACIA+1    OUTPUT CHARACTER
39              RTS
```

Note that this routine will accept any non-null character typein as a break. Alternatively we could look at the Framing Error status, but if a character has been input this status will not show up until that character is read in, rendering it ineffective in some cases. Nulls are excepted as break characters since one or more of them may follow the carriage return in an input tape, and still be pending. Note that for this to work properly, the pad character defined in location 0111 should be set to NULL (hex 00).

The 6800 "R" version of TINY BASIC includes these routines in the code, as shown here. Locations 08FA-08FC contain a JMP to the break test at the beginning of this block. You should alter the ACIA addresses to suit your system before using the subroutines.

CRT OR TVT

If a TV Typewriter is used for I/O it may be desirable to remove excess control characters from the output stream. All controls except Carriage Return may be removed by the following instructions at the beginning of the output subroutine (6800 opcodes shown):

```
39              RTS
810A    OUTPUT  CMP A #0A
2FFB            BLE OUTPUT-1
```

Only nulls, Rubouts, X-ON and X-OFF may be deleted by changing the CMP to a TST A. Nulls may be passed through by also changing the BLE to a BMI.

Some TV Typewriters do not scroll up when the cursor reaches the bottom of the screen, but rather wrap the cursor around to the top of the screen, writing over the previously displayed data. With this kind of display it is essential that the I/O routines (or the hardware) clear to the end of the line whenever a CR-LF is output, so that previous data does not interfere with the new. If your I/O routines are fixed in ROM, some sort of preprocessor may be required to recognize output CR's and convert them to the appropriate sequence of control functions. It may also be necessary to trap input CR's (suppressing their echo) since Tiny generally responds with both another CR and a linefeed.

Some users prefer to concatenate all output into one "line" of print, using the terminal comma or semicolon to suppress the line breaks. Since TINY was designed to limit line lengths to less than 128 characters, if this sort of concatenation is attempted it will appear that TINY has quit running. To eliminate the print suppression the most significant two bits of the print control byte (location 00BF in most versions) may be cleared to zero periodically with the USR function or in the output driver routine. The least significant three bits of this same byte are used for the "comma spacing" in the PRINT statement, and should be left unaltered.

CASSETTE I/O

Officially, TINY only speaks to one peripheral--the console. However a certain amount of file storage may be simulated by attaching these peripherals (such as cassette systems) to the character input and output routines. If the same electrical and software interface is used this is very easy. Otherwise the I/O drivers will

require special routines to recognize control characters in the input and output data for setting internal switches which select one of several peripherals. The USR function may also be used either to directly call I/O routines or to alter switches in memory.

Appendix D

LOW MEMORY MAP

LOCATION	SIGNIFICANCE
0000-000F	Not used by any version of TINY
0011-001F	COSMAC version temporaries
0020-0021	Lowest address of user program space
0022-0023	Highest address of program space
0024-0025	Program end + stack reserve
0026-0027	Top of GOSUB stack
0028-002F	Interpreter parameters
0030-007F	Input line buffer & Computation stack
0080-0081	Random Number Generator workspace
0082-0083	Variable "A"
0084-0085	Variable "B"
...	...
00B4-00B5	Variable "Z"
00B6-00C7	Interpreter temporaries
00B8	Start of User program (PROTO)
00C8-00D7	Sphere parameters (not 0020-002F)
00D8-00FF	Unused by standard version
0100	Cold Start entry point (6800)
0103	Warm Start entry point
0106-0108	JMP (or JSR) to character input
0109-010B	JMP to character output
010C-010E	JMP to Break test
010F	Backspace code
0110	Line Cancel code
0111	Pad character
0112	Tape Mode Enable flag (hex 80 = enabled)
0113	Spare stack size
0114	Subroutine to read one Byte from RAM to A (address in X)
0118	Subroutine to store A into RAM at address in X
0900	Beginning of User program (6800)

Note that some of these addresses apply to the standard 6800 version. For other versions addresses above

0100 should be read as addresses above their respective starting address.

Appendix E

AN EXAMPLE PROGRAM

```
10 REM DISPLAY 64 RANDOM NUMBERS < 100 ON 8 LINES
20 LET I=0
30 PRINT RND (100),
40 LET I=I+1
50 IF I/8*8=I THEN PRINT
60 IF I<64 THEN GOTO 30
70 END


100 REM PRINT HEX MEMORY DUMP
109 REM INITIALIZE
110 A=-10
120 B=-11
130 C=-12
140 D=-13
150 E=-14
160 F=-15
170 X = -1
175 O = 0
180 LET S = 256
190 REMARK: S IS BEGINNING OF TINY (IN DECIMAL)
200 REM GET (HEX) ADDRESSES
210 PRINT "DUMP: L,U";
215 REM INPUT STARTING ADDRESS IN HEX
220 GOSUB 500
230 L=N
235 REM INPUT ENDING ADDRESS IN HEX
240 GOSUB 500
250 U=N
275 REM TYPE OUT ADDRESS
280 GOSUB 450
290 REM GET MEMORY BYTE
300 LET N = USR (S+20,L)
305 REM CONVERT IT TO HEX
310 LET M = N/16
320 LET N = N-M*16
330 PRINT " ";
335 REM PRINT IT
340 GOSUB 400+M+M
350 GOSUB 400+N+N
355 REM END?
360 IF L=U GO TO 390
365 L=L+1
370 IF L/16*16 = L GOTO 280
375 REM DO 16 BYTES PER LINE
380 GO TO 300
390 PRINT
395 END
399 PRINT ONE HEX DIGIT
400 PRINT O;
401 RETURN
402 PRINT 1;
403 RETURN
```

```
404 PRINT 2;
405 RETURN
406 PRINT 3;
407 RETURN
408 PRINT 4;
409 RETURN
410 PRINT 5;
411 RETURN
412 PRINT 6;
413 RETURN
414 PRINT 7;
415 RETURN
416 PRINT 8;
417 RETURN
418 PRINT 9;
419 RETURN
420 PRINT "A";
421 RETURN
422 PRINT "B";
423 RETURN
424 PRINT "C";
425 RETURN
426 PRINT "D";
427 RETURN
428 PRINT "E";
429 RETURN
430 PRINT "F";
431 RETURN
440 REM PRINT HEX ADDRESS
450 PRINT
455 REM CONVERT IT TO HEX
460 N = L/4096
470 IF L<0 N=(L-32768)/4096+8
480 GOSUB 400+N+N
483 LET N=(L-N*4096)
486 GOSUB 400+N/256*2
490 GOSUB 400+(N-N/256*256)/16*2
495 GOTO 400+(N-N/16*16)*2
496 REM GOTO=GOSUB,RETURN
500 REM INPUT HEX NUMBER
501 REM FORMAT IS NNNNX
502 REM WHERE "N" IS ANY HEX DIGIT
505 N=0
509 REM INPUT LETTER OR STRING OF DIGITS
510 INPUT R
520 IF R=X RETURN
525 REM CHECK FOR ERROR
530 IF R>9999 THEN PRINT "BAD HEX ADDRESS
531 REM NOTE ERROR STOP ON LINE 530 (ON PURPOSE!)
535 REM CONVERT INPUT DECIMAL DIGITS TO HEX
540 IF R>999 THEN N=N*16
545 IF R>99 THEN N=N*16
550 IF R>9 THEN N=N*16
555 IF R>0 THEN R=R+R/1000*1536+R/100*96+R/10*6
559 REM PICK UP NON-DECIMAL DIGIT LETTERS
560 IF R<0 THEN LET R=-R
565 REM ADD NEW DIGIT TO PREVIOUS NUMBER
570 LET N=N*16+R
580 GOTO 510
590 NOTE: DON'T NEED END HERE

1000 TO RUN RANDOM NUMBER PROGRAM, TYPE "RUN"
1010 IT WILL TYPE 8 LINES THEN STOP.
1020 TO RUN HEX DUMP PROGRAM TYPE "GOTO 100"
1030 IT WILL ASK FOR INPUT, TYPE 2 HEX ADDRESSES
```

```
1040 EACH TERMINATED BY THE LETTER X,
1050 AND SEPARATED BY A COMMA
1055 (TYPE ALL ZEROS AS LETTER OH).
1060 THE PROGRAM WILL DUMP MEMORY BETWEEN
1070 THOSE TWO ADDRESSES, INCLUSIVE.
1080 EXAMPLE:
1090 :GOTO 100
1100 DUMP: L,U? AO3EX,AO46X
1110 A03E EE FF
1120 A040 00 11 22 33 44 55 66
1130 IF THE RANDOM NUMBER PROGRAM
1140 IS REMOVED, OR IF YOU TYPE IN
1150 :1 GOTO 100
1160 THEN YOU CAN GET THE SAME DUMP BY TYPING
1170 :RUN,AO3EX,AO46X
1180 .
1190 NOTE THAT THIS PROGRAM DEMONSTRATES NEARLY
1200 EVERY FEATURE AVAILABLE IN TINY BASIC.


REMARK: TO FIND OUT HOW MUCH PROGRAM SPACE
REM... YOU HAVE LEFT, TYPE:
LET I=0
1 LET I=I+2
2 GOSUB 1
RUN
REMARK: AFTER A FEW SECONDS, THIS WILL STOP
REM... WITH AN ERROR; THEN TYPE:
END
PRINT "THERE ARE ";I;" BYTES LEFT"

REM: TO EXIT FROM TINY BASIC TO YOUR MONITOR/DEBUGGER,
LET S=256
REM (S AS IN LINE 180 ABOVE)
LET B=0
IF P=6800 THEN LET B=63
REM: B IS SWI OR BRK INSTRUCTION
LET A = USR (S+24,0,B) + USR (0)
REM: THE FIRST CALL STORES A BREAK IN 0000
REM... THE SECOND CALL JUMPS TO IT.
```

Back to TinyBasic main page
Itty Bitty Computers home page