COSMAC TINY BASIC

Preface

TINY BASIC provides the most fundamental of those functions normally attributed to the high-level programming language called BASIC. It is specifically designed for a microcomputer with minimal memory. The TINY BASIC interpreter program requires only 2K bytes of storage. Thus, an Evaluation Kit with 4K of RAM can accomodate modest (about 100 statements in length) TINY BASIC programs.

TINY BASIC is perhaps the best language for the beginning microcomputer programmer. It is easily learned, and elementary application programs may be developed quickly. For the more experienced programmer, TINY forms the kernel of a system whose facilities may be extended indefinitely by the addition of machine-language subroutines (limited only by the amount of memory which is available).

TINY packs a significant amount of processing capability within 2K bytes. For example, it includes its own line editor, and it provides a rich assortment of error messages to the user. However, clearly one cannot expect certain features which are normally available only in 8K systems. For example, TINY does not do floating-point arithmetic. (Its numeric capability is limited to integers in the range -32768 to +32767.) It cannot directly handle arrays or alphanumeric strings. (On the other hand, each of these (and other) advanced facilities may be added via a machine-language extension). In addition, one must recognize that economies in memory space used were achieved at the expense of processing speed.

Generally, then, TINY BASIC may be considered as a good "budget" high-level language for a user with a comparable microcomputer setup. Although TINY is quite slow and is of limited capability, it can act as the nucleus of a system whose sophistication may be indefinitely extended.

COSMAC TINY BASIC

INTRODUCTION

We assume that you are already familiar with section III of the Evaluation Kit Manual which explains the functions available from the resident utility program UT4. UT4 permanently resides in memory locations 8000-81FF. After it is given control (via the RESET, RUN U, CR or LF sequence), it types its prompt character, an asterisk, indicating that it is awaiting your input. Each of your input lines (terminated with a CR) is interpreted and executed by UT4. After disposing of your input command, UT4 indicates that it is ready for new input by typing another * prompt.

One important function of UT4 is to permit you to load an arbitrary sequence of hexadecimal digits (a machine language program) into an arbitrary area in memory and then to invoke this program (transfer control to it; run it) via the appropriate $P command. When your program completes its computation, it may relinquish control back to UT4 by executing a C08039 instruction (a long branch to the location labeled START on p.3-16), provided all registers used by UT4 have the values they had when UT4 exited.† Under these conditions, a user program halt (or exit) would be signified by a new * UT4 prompt.

COSMAC 2K TINY BASIC is a program which must be loaded into the lowest 2K bytes of memory (locations 0000-07FF). A hexadecimal listing of the program and loading instructions for it appear in Appendix A. After TINY BASIC is made resident, control is transferred to it using the proper $P UT4 command (see Appendix A). Once it receives control, TINY BASIC delivers its prompt character, a colon, and awaits your input. Each time after it has properly disposed of an input line (terminated with a carriage return - CR), TINY BASIC again types its : prompt.

† In particular, P should be 5.

If an input line does not begin with a number, TINY BASIC immediately interprets it and executes it. (The line is called a statement.) If the line begins with a number (normally followed by a statement), then TINY BASIC merely stores it, in the proper position, in an area of memory where the user program (a sequence of statements ordered by statement number) is assembled. If the statement number is the same as one already existing in this area, then the new statement replaces the old one. Thus, you load a TINY BASIC program by entering a sequence of statements (one per line), each preceded by a unique statement number. The program must have at least one END statement in it.

After your program has been loaded, you can run it by typing a RUN command (equivalent to the $P command to UT4). TINY BASIC will then interpret and execute your program's statements, in order, following the rules discussed in subsequent sections. When an END statement is encountered during execution, control will be passed back to TINY BASIC's "enter" mode, and another : prompt will be issued.

Note that TINY BASIC assembles statements which begin with numbers into the program area in memory without any further analysis. Errors are detected only when execution is attempted. If an entered line consists only of a line number, it is considered a deletion. The previously inserted statement with the same line number is erased. Note also that 0 is not a valid line number. Blanks within a line have no significance to TINY. All spaces, until the first non-numeric character, are totally ignored. After that, however, blanks are preserved in the memory copy of the statement (i.e., each blank character occupies one byte).

NUMBERS

A number is any sequence of decimal digits optionally preceded by a sign. If no sign is present, the number is assumed positive. Since TINY BASIC stores all numbers internally as 16-bit signed integers, positive values may run from 0 to 32767 (2^{15}-1) and negative values may run from -1 to -32768 (-2^{15}).

VARIABLES

A variable is any single capital letter (A-Z). Each possible variable is assigned a unique two-byte location in memory. The value of the variable is the contents of that location -- i.e., a number in the range -32768 to +32767.

EXPRESSIONS

An expression is a combination of one or more numbers or variables, joined by operators and possibly grouped by parenthesis pairs. The permissible operators are:

+ addition
- subtraction
* multiplication
/ division

Whenever TINY BASIC encounters an expression within a statement (during its execution) it evaluates the expression -- combining the numbers and the values of the variables, using the indicated operations. The exact disposition of the final computed value depends on the type of statement. This is discussed further later.

Internal sub-expressions within parentheses are evaluated first. Usually parentheses make clear the order in which operations are to be performed. However, if there is ambiguity because parentheses are absent, TINY gives precedence to multiplication and division over addition and subtraction. Thus, in evaluating

B-14*C

the multiplication is performed first. In cases involving two operators of equal precedence, evaluation would proceed from left to right. An expression may be optionally preceded by a sign.

Note that during the evaluation of an expression, all intermediate values, and the final value, are truncated -- using the lowest 16 bits of the results. That is, expressions are evaluated modulo 2^{16}. TINY BASIC makes no attempt to discover arithmetic overflow conditions, except that an attempt to divide by zero results in an error stop.

The following are some examples of valid expressions:
(Note that a single variable or number is also an expression.)

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

The following are some examples of expressions which have the same value:

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

because any number in the range 32768 to 65535 (2^{15} to $2^{16}-1$) has a sign bit of 1(making it negative), so that it is actually treated by TINY BASIC as if 65536 (2^{16}) were subtracted from it.

THE RND FUNCTION

TINY BASIC includes the ability to generate a positive pseudo-random number in a specified range. Whenever it encounters the form

RND (expression 1, expression 2)

during execution of a statement, TINY generates a random number in the range from the value of expression 1 to the value of expression 2, inclusive. The resulting number may be used as would any other number. In particular, the above form may itself be used within another expression. If the arguments are invalid, an error stop may result.

THE RND FUNCTION (cont'd)

```
RND (1,100)
RND (A,B)
```

are valid RND functions (assuming 0<A<B).

STATEMENT TYPES

A statement normally begins with a keyword, such as PRINT or GOTO, indicating the type of statement. The interpretation of the remainder of the statement depends on this keyword. In some cases, a short form of the key word is also acceptable -- for example, PR instead of PRINT.

REM STATEMENT

Following the keyword REM (for remark or comment) any sequence of characters may appear. This statement is ignored by TINY BASIC. It is used to permit you to intersperse arbitrary comments or remarks within your program.

END STATEMENT

END must be the last statement executed in a program. It is used to halt execution and return to TINY BASIC's "enter" mode. There may be as many END statements in a program as needed.

LET STATEMENT

This statement has the form

```
LET variable = expression
```

Alternatively, the keyword LET may be omitted entirely. Execution of this statement assigns the value of the expression to the variable. The following are valid LET statements:

```
LET A = B+C
I = I+1
J = 0
LET Q = RND (5,33)
```

IF STATEMENT

This statement has the form

IF expression1 relation expression2 THEN statement

The keyword THEN may be omitted entirely. Execution of this statement evaluates the two expressions and compares them according to the relation specified. If the condition specified is TRUE, then the associated statement is executed. Otherwise, the associated statement is skipped. The permissible relational operators are as follows:

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

The associated statement may be any other valid TINY BASIC statement including, in particular, another IF statement. The following are some valid IF statements:

```
IF I>25  THEN END
IF A>B  IF B>C  I=I+1
```

(The last statement increments I only if B is between C and A.)

TRANSFERS OF CONTROL

TINY BASIC normally executes statements in a program in statement number order. The following statements may be used to alter this flow:

(a) GOTO expression

The subsequent statement executed is the one whose line number equals the value of the expression. Note that this permits you to compute the line number of the next statement on the basis of program parameters during execution. The following are some valid GOTO statements:

```
GOTO 100
GO TO 200 + I*10
```

(b) GOSUB expression

This statement executes exactly as does the GOTO statement, except that in addition TINY records (remembers) the statement number of the following statement (the one which would have been executed next, had the branch not taken place).

(c) RETURN

This statement (which also has the short form RET) executes by transferring control back to the statement whose number was last recorded as the result of the execution of a GOSUB. This last-recorded statement number is also forgotten.

SUBROUTINE NESTING

A subroutine is a sub-program which is normally evoked in two or more places within a main program. Rather than duplicate the statements of the sub-program in several places, it appears only once. It is written so that it exits with a RETURN statement. It is evoked at any point in a program by a GOSUB statement which transfers control to it.

Whenever one subroutine calls another subroutine (termed subroutine "nesting"), an additional "return-statement-number" is recorded. These are stored in order, so that every RETURN jumps back to the statement following the GOSUB which called it. Subroutines may be nested to any depth, limited only by the amount of user program memory remaining.

PRINT STATEMENT

This statement has the form

PRINT printlist

where printlist is a succession of one or more items to be printed separated by either commas or semicolons. The acceptable short form for PRINT is PR. Each print item may be either an expression or a character string enclosed in quotes. In the first case the value of the expression is typed. In the second case the character string is printed verbatim. No spaces are generated between the printouts of items separated by semicolons in the PRINT statement. On the other hand, the printout of an item, preceded by a comma in the PRINT statement, begins at the next "tab setting". Tabs are automatically set every eight character spaces. Thus,

```
PRINT 1,2,3  prints as
1       2       3
```

while PRINT 1;2;3 prints as

```
123
```

Commas and semicolons, character strings and expressions may be mixed in one PRINT statement in any manner.

Normally, the execution of a PRINT statement terminates with the generation of a carriage return and line feed to begin a new line. However, if the PRINT statement ends with a comma or semicolon, then the CR-LF sequence is suppressed, permitting subsequent PRINT statements to output on the same line or permitting an input message (see INPUT, next) to appear on the same line as previous output.

The following are valid PRINT statement examples:

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

INPUT STATEMENT

This statement has the form

```
INPUT inputlist
```

where inputlist is a succession of one or more variables separated by commas. The acceptable short form for INPUT is IN. Normally, execution of this statement begins with the typing of a question mark prompt indicating that TINY is expecting the user to type in data. The user should respond by typing in a line of one or more expressions separated by commas and terminated with a carriage return. Each input expression is evaluated and assigned to its associated variable in the INPUT statement.

If the number of requested variables in the inputlist is not satisfied by the number of expressions in the user's input line, a new ? prompt will be issued asking for more input information. If the number of expressions in the user's input line is greater than the number of requested variables, then those input expressions not requested are saved internally and used to satisfy subsequent INPUT requests. Thus, before a ? prompt is issued during execution of an INPUT instruction, TINY first checks to see if any saved expressions exist. If so, then these are used first - to satisfy some or all of the variables requesting values. Only when no saved data exists is the ? prompt issued. The user is cautioned to use the latter property of the INPUT statement with care.

Example: Suppose statement INPUT X,Y,Z is executed, and the user responds by typing A,C,B. The results are the same as if X=A, Y=C and Z=B had been executed. Note that commas are required in the user's input line only to avoid ambiguity. If he had entered ACB, the same results would have occurred. On the other hand, an input line of +1 -3 +6 0 in response to INPUT A,B,C,D will result in A being given the value 58 and a new ? prompt issued for values for B,C and D.

SYSTEM CONTROL STATEMENTS

The statements listed below are normally not included as part of a program. That is, they are normally entered without line numbers:

(a) NEW

Execution of this statement clears the program area in memory. It is used before entering a new program.

(b) RUN

Begin program execution at the first (lowest) line number. Note: If RUN is followed by a comma followed by a sequence of one or more expressions (separated with commas), then the expression list is treated as an initial input line -- which will be scanned first whenever INPUT statements are executed. (See discussion of INPUT statement.)

(c) LIST

LIST expression

LIST expression, expression

SYSTEM CONTROL STATEMENTS (cont'd)

(c) (cont'd)

The LIST statement causes part or all of a stored user program to be printed. If no parameters are given, the whole program is listed. A single expression parameter is evaluated to a line number. If the line exists, it is printed. If both parameters are given, all lines with numbers in the range specified are printed.

SUMMARY OF COSMAC TINY BASIC REPERTOIRE

The following should serve as your short form guide to the facilities offered by TINY BASIC. Characters enclosed in brackets [ ] are optional and may be omitted.

FORM OF STATEMENT	BRIEF EXPLANATION OF EXECUTION
REM any comment	Ignored.
END	Halt execution and return to "enter" mode.
[LET] variable = expression	Assign the value of the expression to the variable
IF expr rel expr [THEN] statement	If the relation between the values of the expressions is TRUE, execute the statement. Otherwise, skip it.
GOTO expression	Jump to the statement whose number is the expression's value.
GOSUB expression	Save the statement number of the next statement in sequence. Then execute a GOTO.
RET[URN]	Jump to the last saved statement number (see GOSUB) and "unsave" this number.
PR[INT] printlist	Type the items in the printlist. Type values of expressions. Type quoted strings verbatim. Horizontal TAB on comma.
IN[PUT] inputlist	Read and evaluate expressions from the keyboard and assign them in order to the variables specified in the inputlist.
NEW	Clear the program area.
RUN[,expression sequence]	Start execution at first statement. (Save the expression sequence to satisfy subsequent INPUT's.)
LIST[expression][,expression]	Print entire program, or one selected line, or a range of lines.

where:

number = -32768 to +32767; variable = single capital letter.

expression = one or more numbers or variables (possibly grouped by parentheses) joined by operators +,-,*, or /.

relations are =,>,<,<=,>=,<>, or >< .

printlist = one or more expressions or quoted strings separated by commas or semicolons.

inputlist = one or more variables separated by commas.

expression sequence = one or more expressions separated by commas.

NOTE: The RND(expr1,expr2) function generates a positive random number in the range between the values of the expressions. This function may be used anywhere in place of a number.

IMMEDIATE EXECUTION VS. PROGRAM MODE

One important use of the immediate execution mode (entering a statement without a line number) is to permit line-at-a-time testing. LET, IF and PRINT can be demonstrated this way. 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 1,2,3), requesting no input, but with a line number, in a program, this statement will produce an error stop after requesting one value.

Clearly there is no point to executing REM or END in the immediate mode. Furthermore, GOSUB and RETURN are normally meant for the program mode. On the other hand, an immediate GOTO has the same effect as if RUN were typed, but execution may begin at other than the program's first statement.

Similarly, the stored program should not contain a NEW statement (self destruct!), and a stored RUN statement will be equivalent to a GOTO to the first statement. On the other hand, a LIST statement may be included as part of a program and used for printing large text strings, such as instructions to the operator.

PROGRAMMING EXAMPLES

The following two simple programs are designed to give you examples of TINY BASIC in action. The first uses most of the statements in TINY's repertoire. The second demonstrates particularly the use of subroutines. REMarks are omitted from the listings to keep them short. Instead, each program is accompanied by a detailed explanation of its functioning. (It should be emphasized that omission of comments is generally bad documentation practice, but it suits our present objectives.) Each program can be entered in a few minutes. It is recommended that you run both of them to gain experience with the system.

I. Arithmetic Drill Program

This program generates a random sequence of arithmetic problems. After the program prints the problem, you respond with your solution. The program tells you whether your answer was correct or not (providing the right answer in the latter case) and then proceeds to generate a new problem, and so on.

Stepping through the program listed below: first, three random numbers are generated. The value of F (1 to 4) will be used to decide whether this will be an add, subtract, multiply or divide problem. The range of possible values for the arguments A and B was chosen to prevent the possiblity of overflow under two conditions: First, 181*181 is still less than 32767. Second, division by zero is prevented. Because TINY BASIC discards division remainders, the fourth statement is included to keep the division problems interesting. It says: If this is a division problem where the quotient would ordinarily come out as zero (true for many of the A,B combinations that might be generated), arbitrarily increase the size of the dividend (to a maximum of 18100 in this case) to make the problem non-trivial. Statement 50 begins the presentation of the problem to the user by printing an encouraging message followed by the value of the first argument. Notice that the final semicolon keeps the printer on the same line without advancing the carriage further.

Statement 60 does a four-way branch based on the value of F (the arithmetic function selected). Thus, control passes next to one of the following statement numbers: 70, 100, 130 or 160. Each of these statements begins a short sequence which prints the sign for the arithmetic operation and then computes the proper

I. Arithmetic Drill Program (cont'd)

function, placing the result in C. (Notice the final semicolons again, in the PRINT statements.) No matter which path is taken, control passes next to statement 180, which prints the second argument value followed by an = sign. The presentation of the problem to the user is now complete, and the INPUT statement at 190 delivers a ? prompt on the same print line and reads the user's answer into D. Statement 200 congratulates the user on a correct answer, while 210 points out that his answer was incorrect and provides him with the proper result. The commas at the end of both PRINT statements here again inhibit a new line from starting, but they space over to the next tab setting, where a new problem is posed as a result of the loop (at 220) back to the top.

```
10 A=RND(1,181)
20 B=RND(1,181)
30 F=RND(1,4)
40 IF F=4 IF A/B<1 A=A*100
50 PRINT "TRY THIS ONE: ";A;
60 GO TO 40+F*30
70 PRINT "+";
80 C=A+B
90 GO TO 180
100 PRINT "-";
110 C=A-B
120 GO TO 180
130 PRINT "*";
140 C=A*B
150 GO TO 180
160 PRINT"/";
170 C=A/B
180 PRINT B;"=";
190 INPUT D
200 IF D=C PRINT "RIGHT!",
210 IF D<>C PRINT "WRONG. CORRECT ANSWER IS ";C,
220 GO TO 10
```

Notice that an END statement is not present here -- contrary to earlier advice. The nature of this program is such that TINY will never go past the last statement. The program as written loops endlessly, and only under these conditions is the omission of an END permissible.

Running this program should give you some practice in learning how TINY divides.

II. Geometric Print Pattern Program

This program is designed to print three identical, trapezoidal patterns across the page, each filled with repeated imprints of the same numeric digit. The user can specify which digit is to fill each trapezoid and, for all three, the number of characters across its top, the slope of its sides (positive or negative) and its height. He can also specify the spacing between the patterns on the page.

Since the printer prints line-by-line, the program prints the pattern in a scanning mode. Every line consists of a sequence of three identical segments, and each segment contains D spaces followed by E identical digits followed by D spaces again. The values of D and E vary from line to line. For each new line, D is decremented by a value I (positive or negative) and E is incremented by 2*I (to keep the pattern symmetrical).

To analyze the program listed below, let us begin by identifying its subroutines. Reading from the bottom up, the subroutine from 250 to 280 prints the digit N, M times across (notice the semicolon). Similarly, the subroutine from 210 to 240 prints a sequence of M spaces. Finally, the subroutine from 140 to 200 prints D spaces followed by E digits (all N) followed again by D spaces. Notice that this subroutine calls the other two.

The main part of the program runs from 10 to 130. First, the program initializes a counter J for the number of lines which have been printed. Then it reads (from the user) initial values for A to E, I and L (the total number of lines to be printed). A,B and C should be single digits. D,E and L must be > 0. Each of the three sequences 30-40, 50-60, and 70-80 prints one segment of a line using the digit specified by the user. 85 starts a new line. 90 and 100 advance D and E as explained earlier, and 110-120 decide whether or not a sufficient number of lines have yet been printed. If not, a new line is started.

GEOMETRIC PRINT PATTERN PROGRAM

```
10 J=0
20 INPUT A,B,C,D,E,I,L
30 N=A
40 GOSUB 140
50 N=B
60 GOSUB 140
70 N=C
80 GOSUB 140
85 PRINT
90 D=D-I
100 E=E+2*I
110 J=J+1
120 IF J<>L GO TO 30
130 END
140 M=D
150 GOSUB 210
160 M=E
170 GOSUB 250
180 M=D
190 GOSUB 210
200 RETURN
210 PRINT " ";
220 M=M-1
230 IF M>0 GOTO 210
240 RETURN
250 PRINT N;
260 M=M-1
270 IF M>0 GOTO 250
280 RETURN
```

For this program to run properly the values of D and E should not become too small. Nor should they be so large as to requ ire excessive line length. The inital values should obey the following relations: $3(E+2D) <$ maximum line width; If $I<0$, $E>2|I|(L-1)$; If $I>0$, $D>I(L-1)$.

THE USR FUNCTION

TINY BASIC includes an important feature to permit you to extend its facilities via machine language subroutines. To use this feature, you must be familiar with many of the intricate details associated with machine language programming. Not only must you know the instruction set for the CPU (See MPM-201, User Manual for the CDP1802 Microprocessor), but you must also be aware of which CPU and memory registers are reserved for TINY, which are freely available for your use and which can act as an interface between your machine-language program and your TINY BASIC program. We assume here that you are familiar with the manual cited above and that you have some introductory machine language programming experience.

The form of the USR construct within a TINY BASIC statement is as follows:

USR (expression [,expression][,expression])

where the brackets indicate that either or both of the latter two expressions may be omitted. On encountering this form, TINY evaluates the first expression and transfers control to that address. (Remember that a desired hex address must be converted into its equivalent decimal expression value, and that addresses in the upper half of memory have negative equivalent decimal values.) If a second expression is included, it is evaluated and the resulting value is passed to the called program as the contents of CPU register 8. If a third expression is included, its value is passed in register A (with D also holding RA.0). The subroutine receives control with P=3 and X=2.

Your called program must return with a SEP 5 (D5) instruction. When it returns, its 16-bit function value is the final contents of RA.1 and D (lower 8 bits in D) just before the SEP 5 was executed. This is why USR is called a function. Whenever it is called, it returns a result - a number. Thus, the USR form can appear anywhere in a TINY BASIC statement where a number can normally appear. (Recall our previous discussion of the RND function. Exactly the same idea applies here.)

Thus, in addition to performing some machine-language function (for example, moving a block of data), your USR program will always return a value or result in RA.1 and D. In many cases, this is desirable -- for example, when your subroutine is given two arguments X and Y (in R8 and RA) and returns a number which is, say, the larger of the two. In other cases, however, your USR program will not need to return a value. In that case the value returned must be ignored in the TINY BASIC program which called it. There are several ways to do this. For example, if

```
+0*USR(.......)
```

were included in an expression, then the USR function would be executed but the returned value would be ignored.

For your convenience, TINY itself includes four built-in subroutines which you may want to make use of via the USR mechanism. They are as follows:

(1) USR(20,N)

Returns the decimal value of the byte at memory location N (decimal), where N is the value of the second expression. (Note that this machine language routine begins at location 14 hex.)

(2) USR(24,N,M)

Stores the value of the third expression, M (mod 256) into the byte at location N (decimal), the value of the second expression. Also returns the value M as the function's "value".

Examples: PRINT USR(20,3072) prints the decimal contents of memory location 0C00
A=USR(24,3072,254) loads memory location 0C00 with FE and also loads the"returned value", 254, into A.

(3) USR(6)

Reads one ASCII character from the keyboard and returns its decimal equivalent (including parity bit if any).

(4) USR(9,0,C)

Prints the ASCII character whose code is the right half of the (hex) value of expression C. (Note: The second expression, in this case 0, is ignored. The character to be typed must start out in a D register. Hence, the above format. The third expression is passed in RA with its lower half also in D.) This routine happens to return a "value" 251 in all cases -- which would normally be ignored, as explained earlier.

Examples: PRINT USR(6) will read a character and print its decimal equivalent. On the printer you would see, for example, A65 for a zero parity bit (where A was typed by you).

A=A+0*USR(9,0,66) will print the character B and ignore the returned result (251).

Register Usage and An Example USR Routine:

When you write your own USR routine, you must be careful not to modify the contents of those registers which are used by TINY BASIC. These include CPU registers and memory registers. Appendix B lists how the CPU registers are used by TINY. Machine language subroutines have the free use of

R0,R1,R8,RA,RD and RF.

In addition, R2 is pointing at a free byte on the control stack.

Clearly, the memory areas used by TINY should also not be modified, except with care. TINY uses most of the first page of the available RAM (beginning at 0800) for its own storage. A table of the allocation of this space is given in Appendix C. You probably will not want to bother with any part of this area except for that which includes the A to Z variable cells. These are located at 0882 to 08B5. Note also that, by reducing the address value stored in 0822, you can make space for your added program and data areas in upper memory.

Appendix D lists some key locations at the beginning of the TINY BASIC program itself. (Notice locations 6, 9, 14 and 18 which correspond to the entry points for the built-in subroutines discussed earlier.) TINY BASIC was written as a pure procedure (capable of execution out of ROM) -- not modified in any way as it runs. This area should not be altered except, conceivably, for modifications to the special character codes beginning at location F. This is discussed further later in this manual.

Consider now an example of a USR added routine. Assume we wish to add a logical AND operation to TINY's repertoire. The machine language routine given below will do the job, given that the two arguments are passed in R8 and RA, and that the computed result must be passed back in RA.1 and D.

```
98   GHI   R8
52   STR   R2
9A   GHI   RA
F2   AND
BA   PHI   RA
88   GLO   R8
52   STR   R2
8A   GLO   RA
F2   AND
D5   SEP   R5
```

Given two 16-bit arguments, this routine computes the 16-bit AND of these and returns that result. Note the use of the spare byte pointed to by R2 and the assumption that X=2 on entry. Notice also the SEP5 exit. This routine can be stored in any available memory area.

Assuming the above program is stored at location 0C00, then if L=3072, the statement T=USR(L,R,S) will assign to T the 16-bit AND of the values of variables R and S.

ERROR MESSAGES AND PROGRAM DEBUGGING

Error Messages:

Whenever TINY BASIC detects an error in a statement, it generates an error message consisting of an exclamation point followed by a decimal error number. A listing of error numbers and their corresponding meanings is given in Appendix E. If the error is detected during program execution, the error code is followed by the word AT followed by the offending statement's number.

Almost all of the errors detected by TINY are syntax errors. TINY was in the process of interpreting a statement and found it unacceptable for some reason. Only two of the errors in the error list are detected during execution of a statement (i.e., after its syntax has been accepted). These are errors 141 and 243.

Any other error number not listed in the table signifies a memory "full" condition -- probably due to too many nested GOSUB's or an excessively complex expression.

Program Debugging:

Most program execution errors are due to either incorrect flow or improper modification of variable values. To find an error of the first kind, you must determine whether your program is sequencing properly -- whether certain sections of code are indeed executed when expected. Often, the insertion of dummy PRINT statements within suspected code sections will reveal whether the flow within the program is proper.

The second type of error is most easily detected by inserting dummy program stops at key point. This procedure is also useful for diagnosing incorrect flow. A dummy stop is an inserted END, or some other inserted statement which is intentionally erroneous to cause an error stop. Once the stop occurs, you may examine the values of key variables (using the immediate execution mode - e.g., PRINT A,B,C) to see if they indeed have the expected behavior. In some cases, variable values may be corrected, in the immediate mode, while the program is still stopped. In this case, and in the case where the program behavior is proper so far, you will want to resume the program at the point where it last stopped. An immediate or direct GOTO, using the statement number after the stop, will permit the program to proceed as if it had not been interrupted.

APPENDIX A

LOADING AND STARTING TINY BASIC

The hexadecimal listing given below is the TINY BASIC object program (listed in UT4 semicolon format). Initially, you will have to load this file into memory by hand from the keyboard and then verify that it is a faithful copy. While this process is time consuming, it needs to be done only once. After memory is loaded, the contents of the first 2K bytes should be properly recorded on your peripheral file storage medium. Section III of your Evaluation Kit Manual contains instructions for recording a file from memory (using UT4's ?M command) onto a Teletype's paper tape or a TI terminal's magnetic tape cassette. If your terminal is different from either of these, you must develop equivalent procedures to those described in the manual. Once you have correctly recorded a copy of TINY BASIC on paper tape or tape cassette, it should be easily reloadable by preceding the tape read with a !M from the keyboard. This is discussed in the Evaluation Kit Manual.

Once TINY BASIC has been loaded, it may be started at one of two locations:

$P1 is the normal "cold" start. TINY BASIC initializes itself (sizes memory; copies a control block from 000F-001B to 0813-081F; and marks the user program space empty) and then delivers the : prompt.

$P3 is the "warm" start, which skips the initialization procedure and preserves the state of RAM. It is used as a restart, when there is already a useful program resident in RAM or when certain control parameters have been modified so that they are different from those which were first initialized. If, after a "warm" start, you wish to enter a new program, type the NEW command.

```
0000 0130 B0C0 00ED C006 6FC0 0676 C006 665F;
0010 1882 8020 3022 3020 58D5 0681 08C8 0008;
0020 4838 97BA 48D5 C006 51D3 BFE2 8673 9673;
0030 83A6 93B6 46B3 46A3 9F30 29D3 BFE2 96B3;
0040 86A3 1242 B602 A69F 303B D343 ADF8 08BD;
0050 4DED 304A 0198 01A0 021F 01DD 01F0 01D4;
0060 0481 0249 00ED 044E 0104 05A2 01D3 01D3;
0070 04AA 01D3 01D3 02C5 02D5 0303 0279 0318;
0080 053C 01D3 0429 036C 03CB 03A7 0398 039B;
0090 040E 0460 046D 0581 01B6 0267 0348 034B;
00A0 01D3 01D3 01C9 01C5 024E 0244 0241 01D3;
00B0 F8B3 A3F8 00B3 D3BA F81C AA4A B24A A24A;
00C0 BDF8 00AD 0DBF E212 F0AF FBFF 52F3 EDC6;
00D0 9FF3 FCFF 8F52 3BC6 220A BDF8 23AD 8273;
00E0 9273 2A2A 0A73 8DFB 123A E3F6 C8FF 00F8;
00F0 F2A3 F800 B3D3 B4B5 B7F8 2AA4 F83C A5F8;
0100 4BA7 331A D720 BB4D AB97 5B1B 5BD7 168B;
0110 F4BF D724 9F73 9B7C 0073 D722 B24D A2D7;
0120 2682 7392 73D4 02CC D71E B94D A9E2 49FF;
0130 3033 4BFD D733 85FE FCB0 A6F8 2D22 2273;
0140 9373 97B6 4652 46A6 F0B6 D5FF 103B 6AA6;
0150 FA1F 325C 5289 F473 997C 0038 7373 86F6;
0160 F6F6 F6FA FEFC 54A6 3042 FC08 FA07 B649;
0170 A633 7A89 7399 73D4 0237 D71E 86F4 A996;
0180 2D74 B930 2DFD 0752 D71A ADE2 F4A6 9DB6;
0190 0D52 065D 0256 302D 86FF 20A6 967F 0038;
01A0 96C2 027F B986 A930 2D1B 0BFF 2032 A9FF;
01B0 10C7 FD09 0BD5 D401 C54D AD9A 5D1D 8A5D;
01C0 30C9 D401 C5D4 01C9 BAD7 1A2D FC01 5DAD;
01D0 2D4D AAD5 D401 AAFB 0D32 2D30 A0D4 01AA;
01E0 FF41 3BA0 FF1A 33A0 1B9F FED4 0259 302D;
01F0 D401 AA3B A097 BAAA D402 544B FA0F AA97;
0200 BAF8 0AAF ED1D 8AF4 AA9A 2D74 BA2F 8F3A;
0210 059A 5D1D 8A73 D401 AAC3 01FB C001 2D9B;
0220 BA8B AAD4 01AA 1B52 49F3 3223 FB80 321C;
0230 9ABB 8AAB C001 A0D7 2482 F52D 9275 337F;
0240 D549 3059 49BA 4930 55D4 0525 3055 D401;
0250 C5D4 0254 8AD4 0259 9A52 D719 F733 7FF8;
0260 01F5 5DAD 025D D5D4 01C9 AD4D BA4D 3055;
0270 FB2F 3266 FB22 D402 F44B FB0D 3A70 29D7;
0280 18B8 D402 CCF8 21D4 02F4 D71E 89F7 AA99;
0290 2D77 BAD4 0315 9832 A9F8 BDA9 93B9 D402;
02A0 C5D7 28BA 4DAA D403 15F8 07D4 0009 D402;
02B0 D5D7 1A97 5DD7 26B2 4DA2 C001 2820 4154;
02C0 20A3 D402 F249 FC80 3BC2 30F2 D719 F880;
02D0 7397 7373 C8D7 1BFE 3366 D715 AAF8 0DD4;
02E0 0009 D71A 8AFE 32EF 2A97 C7F8 FF30 DF73;
02F0 F88A FF80 BFD7 1B2D FC81 FC80 3B66 5D9F;
0300 C000 09D7 1BFA 07FD 08AA 8A32 97F8 20D4;
0310 02F4 2A30 0AD4 0254 D71A ADD4 0413 3B25;
0320 F82D D402 F497 73BA F80A D402 551D D403;
0330 E38A F6F9 3073 1D4D EDF1 2D2D 3A2E 1202;
0340 C201 C2D4 02F4 303E D72E 389B FB08 3A5E;
0350 8B52 F0FF 8033 5ED7 2E8B 739B 5DD5 D72E;
0360 B80D A88B 739B 5D98 BB88 ABD5 D401 C59A;
0370 FB80 738A 73D4 01C9 AFD4 01C5 128A F7AA;
0380 129A FB80 7752 3B92 8AF1 328F 8FF6 388F;
0390 F638 8FF6 C7C4 19D5 D404 0ED4 01C5 ED1D;
03A0 8AF4 739A 745D D5D4 01C5 F810 AF4D B80D;
03B0 A80D FE5D 2D0D 7E5D D404 223B C5ED 1D88;
03C0 F473 9874 5D2F 8F1D 3AB1 D5D4 01C5 9A52;
03D0 8AF1 C202 7F0D F373 D404 132D 2DD4 0413;
03E0 1D97 C897 73AA BAF8 11AF ED8A F752 2D9A;
03F0 773B F6BA 02AA 1D1D 1DF0 7E73 F07E 738A;
```

2 K

TINY

BASIC

Cold start $P1

Warm start $P3

```
0400 7ED4 0424 2F8F CA03 EA12 02FE 3B21 D71A;
0410 AD30 18ED F0FE 3B21 1D97 F773 9777 5DFF;
0420 00D5 8AFE AA9A 7EBA D5D7 18C2 02B1 4BFB;
0430 0D3A 2ED4 0598 324B D400 0C33 46D7 1CB9;
0440 4DA9 D717 5DD5 D71E B94D A9C0 027F D720;
0450 BB4D ABD4 0598 324B D71C 8973 995D 3042;
0460 D404 FE32 38D7 288A 739A 5D30 4BD4 048B;
0470 42BA 02AA D726 8273 9273 D405 013A 6530;
0480 88D4 048B 42B9 02A9 C001 2DD7 2212 1282;
0490 FC02 F32D 3A9C 927C 00F3 324B 12D5 D716;
04A0 3897 FED7 1A97 765D 30B2 F830 ABD4 0254;
04B0 9DBB D400 06FA 7F32 B252 FB7F 32B2 FB75;
04C0 329E FB19 32A1 D713 02F3 32D7 2D02 F33A;
04D0 DD2B 8BFF 3033 B2F8 30AB F80D 3802 5BD7;
04E0 198B F73B ECF8 07D4 02F4 0B38 4BFB 0D3A;
04F0 B2D4 02D5 D718 8B5D F830 ABC0 01C5 D401;
0500 C58A 529A F1C2 027F D720 BB4D ABD4 0598;
0510 C68D D5ED 8AF5 529A 2D75 E2F1 3312 4BFB;
0520 0D3A 1E30 0DD4 0528 D401 C54D B84D A84D;
0530 B64D A68D 52D7 1902 5DAD 8AD5 D72C 8B73;
0540 9B5D D404 FED7 2A8B 739B 73D4 04FE 2B2B;
0550 D72A 8BF7 2D9B 7733 7B4B BA4B AA3A 629A;
0560 327B D403 15F8 2DFB 0DD4 02F4 D400 0C33;
0570 7B4B FB0D 3A67 D402 D530 50D7 2CBB 4DAB;
0580 D5D7 2682 7392 5DD7 182D CED7 28AA 4D12;
0590 12E2 738A 73C0 012D D727 4B5D 1D4B 73F1;
05A0 1DD5 D403 5ED4 04FE FCFF 97AF 33BA 9BBD;
05B0 8BAD 2F2F 2F4D FB0D 3AB4 2B2B D403 5ED7;
05C0 280B FB0D 735D 32D9 9A5D 1D8A 5D9B BA8B;
05D0 AA1F 1F1F 4AFB 0D3A D3D7 2EBA 4DAA D724;
05E0 8AF7 AA2D 9A77 BA1D 8FF4 BF8F FA80 CEF8;
05F0 FF2D 74E2 73B8 9F73 5282 F598 5292 75C3;
0600 027E 8F32 3052 FE3B 1ED7 2EBF 4DAF E2F7;
0610 A89F 7C00 B848 5F1F 1A9A 3A15 3030 9FAF;
0620 98BF D724 B84D A82A EF08 2873 1A9A 3A29;
0630 D724 1242 7302 5DD7 2EBA 4DAA D728 AFF1;
0640 324E 8F5A 1A4D 5A1A 4B5A FB0D 3A47 C002;
0650 B573 5297 BA2D 43D5 5D2D 88FA 0FF9 605D;
0660 FA08 CEC4 12DD FC00 376E FF00 3F6C D5D7;
0670 118D 73C0 8140 D712 327E DC17 2D5D C081;
0680 A424 3A91 2710 E159 C32A 562C 8A47 4F54;
0690 CF30 D010 11EB 6C8C 474F 5355 C230 D010;
06A0 11E0 1416 8B4C 45D4 A080 BD30 D0E0 131D;
06B0 8C50 D283 494E D4E1 6285 BA38 5338 5583;
06C0 A221 6330 D020 83AC 2262 84BB E167 4A83;
06D0 DE24 93E0 231D 9149 C630 D031 1F30 D084;
06E0 5448 45CE 1C1D 380B 9B49 CE83 5055 D4A0;
06F0 10E7 243F 2091 27E1 5981 AC30 D013 1182;
0700 AC4D E01D 8A52 45D4 8355 52CE E015 1D85;
0710 454E C4E0 2D87 5255 CE10 1138 0A84 4E45;
0720 D72B 9F4C 4953 D4E7 0A00 010A 7FFF 6530;
0730 D030 CBE0 2400 0000 0000 000A 801F 2493;
0740 231D 8452 45CD 1DA0 80BD 382A 82AC 620B;
0750 2F85 AD30 E617 6481 AB30 E685 AB30 E618;
0760 5A93 AD30 E619 5430 F585 AA30 F51A 5A85;
0770 AF30 F51B 542F 8852 4E44 A831 1539 448E;
0780 5553 52A8 30D0 30CB 30CB 311C 2E2F A212;
0790 2FC1 2F80 A865 30D0 0B80 AC30 D080 A92F;
07A0 84BD 0902 2F83 3CBE 7485 3CBD 0903 2F84;
07B0 BC09 012F 853E BD09 062F 853E BC09 052F;
07C0 80BE 0904 2F19 170A 0001 1809 8009 8012;
07D0 0A09 291A 0A1A 8518 0813 0980 1203 0102;
07E0 316A 3175 1B1A 1931 7518 2F0B 0105 0104;
07F0 0B01 0701 062F 0B09 060A 0000 1C17 2F00
```

APPENDIX B

REGISTER ALLOCATIONS

Registers R0 and R1 are not used by TINY BASIC in any way. In addition, the program makes no reference to Q or EF1,2,3 or 4. All character I/O is funnelled through a vector near the beginning of the program. The user may request the performance of INP or OUT instructions as part of the BASIC program, but these are up to the user's discretion.

The other registers used by TINY are as follows:

2	Control stack pointer.
3	Inner interpreter Program Counter.
4	Call linkage PC.
5	Return linkage PC.
6	Top of control stack; =address of caller. Also holds branch address.
7	Byte Fetch PC.
8	Temporary work register. Receives second argument in USR call.
9	Outer interpreter Program Counter. =address of next IL opcode.
A	16-bit accumulator and work register. Contains third argument of USR calls, and part of response from USR calls.
B	BASIC Pointer. Points to next token.
C	Timing subroutine in Terminal I/O.
D	Workspace memory pointer. =Expression Stack Pointer in USR calls.
E	Subroutine linkage temporary and Terminal timing constant.
F	Temporary work register.

Machine language subroutines called via the USR function have the free use of R0,R1,R8,RA,RD,RF.

APPENDIX C

USE OF FIRST PAGE OF USER RAM BY TINY BASIC

0812	UT3/UT4 output delay flag
0813	Copy of BACKSPACE code
0814	Copy of CANCEL code
0815	Copy of Pad code
0816	Copy of Tape Mode Enable
0817	Copy of Spare stack Space
0818	Execution mode flag
0819	End of input line
081A	Expression Stack pointer
081B	Output Control
081C-081D	Saved address for NX
081E-081F	Copy of IL base address
0820-0821	Lowest address of user program space
0822-0823	Highest address of user program space
0824-0825	End of user program + stack reserve
0826-0827	Top of GOSUB stack
0828-0829	Current Line number in BASIC
082A-082D	Temporary
082E-082F	Input line pointer
0830-087F	Input line buffer and expression computation stack
0880-0881	Random Number Generator seed
0882-08B5	BASIC variables A-Z

Note: Each variable occupies two bytes beginning at a displacement in the page which is twice its ASCII code.

Displacement	Variable
0082	A
0084	B
.	
.	
.	
00B4	Z

APPENDIX D

ALLOCATIONS IN LOW RAM

0001	Cold Start
0003	Warm Start
0006-0008	LBR to character input
0009-000B	LBR to character output
000C-000E	LBR to Break test
000F	Backspace code
0010	Line Cancel code
0011	Pad character
0012	Tape Mode enable flag (hex 80=enabled)
0013	Space stack size
0014	Byte fetch subroutine
0016	Double byte fetch entry vector
0018	Byte store Subroutine
001A-001B	Address of IL
001C-001D	User space start for scan
001E	Page for memory wrap test
001F	Page for workspace
0120	Entry vector for Hex input
0123	Entry vector for Hex print
0126	Entry vector for I/O
0129	Entry vector for AND
0800	Beginning of user RAM space

APPENDIX E

ERROR MESSAGE SUMMARY

0	Break during execution
8	Memory overflow; line not inserted
9	Line number 0 not allowed
11	RUN with no program in memory
33	Improper syntax in GOTO
35	No line to GO TO
40	LET is missing a variable name
42	LET is missing an =
45	Improper syntax in LET
47	LET is not followed by END
65	Missing close quote in PRINT string
83	Circumflex in PRINT is not at end of statement
85	PRINT not followed by END
101	IF not followed by END
111	INPUT syntax bad - expects variable name
130	INPUT syntax bad - expects comma
131	INPUT not followed by END
140	RETURN syntax bad
141	RETURN has no matching GOSUB
142	GOSUB not followed by END
147	END syntax bad
179	LIST syntax error - expects comma
189	Can't LIST line number 0
193	LIST not followed by END
198	REM not followed by END
199	Missing statement type keyword
201	Misspelled statement type keyword
243	Divide by zero
276	Syntax error in Expression - expects value
281	RND expects two arguments
286	Missing right parenthesis
321	IF expects relation operator
356	Invalid arguments in RND

All other error numbers signify memory overflow (too many nested GOSUBS) or an excessively complex expression.

APPENDIX F

SPECIAL KEYBOARD CONTROL CHARACTERS

You may erase (backspace over) an incorrectly-entered character by hitting the "erase previous character" key. Its hex code is stored in location 000F, and it is presently an ASCII Left-arrow or Underline (Shift O; hex 5F). Each occurrence of __ erases the last stored input character. Thus,

POINT _ _ _ _ RINT

corrects the erroneous second character. Similarly, you may erase the entire input line and start over by hitting the "cancel line" character. Its hex code is stored in location 0010, and it is presently an ASCII CANCEL (Control X; hex 18). You may change either of these edit control characters by changing its stored code - to any value except DC3, LF, NULL or DELETE (hex codes 13, 0A, 00 and FF, respectively). These special characters are trapped by TINY before its line edit code is entered.

The BREAK key may be used for two purposes: to interrupt a long LISTing or to interrupt the execution of a program (for example, one caught in an endless loop). While executing the LIST command, TINY checks BREAK at the beginning of every typed line. While executing a stored program, TINY checks BREAK between statements.

Each of your input lines from the keyboard is terminated with a carriage return (CR). Whenever TINY generates a new line (for example, when it echoes your CR), it generates CR PAD PAD LF PAD, where the pad character depends on the 2^7 bit of location 0011 (hex). If 0, it is the NULL character (hex 00). If 1, it is the RUBOUT/DELETE character (hex FF). The rest of the byte in location 0011 defines the count of the number of pads to be sent between each CR and LF. It is presently set to 2.

SUMMARY OF KEY CHARACTERS

CR	Terminates every entry line.
__	Backspace.
CAN	Cancel line.
BREAK	Interrupt long printout or execution.

Appendix G

Tape Control Characters

Whenever TINY generates the ? prompt character (during execution of an INPUT statement), it follows this by generating the XON (ASCII DC1) control character. If the input comes from tape, the user may elect to use this special control character to activate the tape reader.

Similarly, TINY generates the XOFF (ASCII DC3; hex 13; Control S) control character whenever an error stop or NEW or END occurs - under the assumption that the user may want to deactivate the reader with this character.

These control characters may be ignored if the user has found an alternative method for tape I/O.