TINY BASIC UNIVERSAL

This Tiny Basic tape contains three copies of Tiny Basic recorded in the following order:

#1 TBX-TVCOS-TV-ONLY
#2 TBX-BAUDOT-TV & BAUDOT TTY
#3 TBX-ASCII-TV & ASCII TTY

Functionally the three versions are identical. The only variable is the output print speed which is fastest for the TVCOS version and slowest for the Baudot version. Most users with TTY equipment find it best to develop their programs using the TVCOS version. They "SAVE" them to cassette tape and "LOAD" them into the appropriate TTY version for hardcopy execution.

Each version of TBX is capable of making a copy of itself. Space limitations did not permit room for the copy message to be in the "shopping list"; however, if it had it would have read

Ø COPY TINY BASIC
* * * * *

As soon as you receive your TBX-UNIVERSAL tape you should do the following:

1. Read in the TVCOS version and you will notice the following:
   a) There will be a pause at the end of the read-in while the numbers are still on the screen. During this pause TBX is adjusting itself to your memory size.
   b) The following message will appear on the screen

      SCREEN SIZE 32/64?
      REPLY 3 OR 6

      If the message appears all on one line, you have a 64 character screen so reply 6. If it appears on two lines, you have a 32 character screen so reply 3.

2. As soon as you have replied to the screen size message, the regular "shopping list" should appear on the screen. At this time insert a blank cassette in your recorder, start it recording and press the Ø (zero) key to make your own customized copy of TBX-TVCOS-UNIVERSAL.

po box 1086, arvada, colorado 80001
(303) 422-4566

The same procedure is followed to make your own copies of TBX-BAUDOT-UNIVERSAL and TBX-ASCII-UNIVERSAL.

***NOTE: The only method possible because of space limitations to adapt Tiny Basic to the 64 character screen was to insert a space after every normally output character. While this may not seem to be a very elegant solution, it does enable you to continue running your TBX programs with the displays as you are accustomed to seeing them.

digital group software systems inc.

Dear User:

Thank you for your order for TINY BASIC EXTENDED TV-CASSETTE OPERATED SYSTEMS.

The attached documentation should be more than sufficient to enable the user who is already familiar with the BASIC language family to effectively use TINY BASIC. For those users who have not worked with BASIC before, it is suggested that they procure one of the many fine tutorials on the use of the BASIC Language. There are a number of sources for BASIC manuals and one of the best is The People's Computer Company at P. O. Box 310, Menlo Park, California 94025.

Digital Group Software Systems, Inc. (DGSS) distributes a number of TINY BASIC related products. At the present moment, these products are a series of cassette tapes containing computer game programs. Each cassette contains five or more games or "fun" type programs. The cost of each cassette is $5.00 and documentation is provided if requested.

TINY BASIC users are encouraged to develop additional games and programs which run under the system. Users who submit games and/or programs to DGSS with appropriate documentation and who consent to permitting DGSS to distribute copies will be paid a small royalty for each copy sold. Submissions must include a cassette tape containing the game or program and typewritten documentation as required. All submissions will be acknowledged but cannot be returned unless accompanied by a self-addressed stamped mailer. Obviously, if many users send in submissions, there will be duplications. In this event, date of receipt and quality of work will determine which contributors' submission will be utilized. Royalties will be paid quarterly in cash or may be applied to the purchase of products at a discounted rate.

We hope that you will enjoy the use of TINY BASIC and will advise us of any problems which you encounter.

Sincerely,

Dianne Howerton

Dianne W. Howerton
Vice President, Operations

1 Attachment

po box 1086, arvada, colorado 80001

TINY BASIC-EXTENDED TVCOS VERSION

I. INTRODUCTION

TINY BASIC EXTENDED (TBX) was created by Dick Whipple and John Arnold, both of Tyler, TX, based upon the design criteria published in the September 1975 "People's Computer Company" (Vol. 4, No. 2). The Whipple and Arnold TBX design was published in the January 1976 issue of "TINY BASIC Calesthentics & Orthodontia". Dr. Robert T. Suding of The Digital Group, Inc., designed and developed the software interfaces between TBX and the TV-Cassette Operating System (TVCOS) used with The Digital Group, Inc. 8080 micro-computer. TBX-TVCOS is a super-set of TINY BASIC as originally proposed and is a limited and modified subset of the full BASIC language.

TBX-TVCOS is available in three standard versions dependent upon the amount of memory available on the machine. These three versions are for 10K, 18K and 26K machines, respectively. All versions are upward compatible in that they can run on larger machines. However, they do not take advantage of any additional memory. Upon request, and for a nominal additional charge, a customized version of TBX-TVCOS can be supplied for any specified memory size assuming that the available memory is contiguous.

II. FEATURES OF TBX-TVCOS

In order to make TBX-TVCOS as flexible and useful as possible without using an excessive amount of memory, all of the non-TBX related features of the operating system have been replaced by TBX-related features. The release version of TBX-TVCOS incorporates several features which will be useful to the user. These features are implimented as operating system options in lieu of the standard operating system options. The features available under TBX-TVCOS are displayed on the TV-screen at the end of reading in the TINY BASIC release cassette and are as follows:

1. READ BASIC Program
2. WRITE BASIC Program
3. Display Commands
4. Display Error Codes
5. Continue Programming
6. TINY BASIC

The uses of the TBX-TVCOS features are as follows:

1. READ BASIC Program (from cassette): Causes a program or program part written in the TINY BASIC language, which was written to a cassette under option 2 below, to be read from the cassette and made ready for execution under TBX-TVCOS. CAUTION!! Programs created on and written from systems larger than the one being read into will read in correctly BUT WILL NOT EXECUTE PROPERLY. As long as the machine on which a program was created and from which it was written is EQUAL IN SIZE OR SMALLER THAN THE MACHINE ON WHICH THE PROGRAM IS TO BE RUN, there should be no difficulty in execution.

2. WRITE BASIC Program (to cassette): Causes a copy of the program or program part written in the TINY BASIC language, which is currently resident in the machine, to be written to a cassette. Programs or program parts written to cassette(s) under this option are in the correct format for subsequent re-entry under option 1 above.

3. Display Commands: Causes an unannotated list of the TINY BASIC verbs/commands to be displayed on the TV-screen. Return to the operating system is accomplished by keying the space bar on the keyboard. When option 3 is selected, the following will appear on the screen:

TINY BASIC COMMANDS

DTA	LET
PR	GOTO
GOSUB	RET
IF	IN
LST	RUN
NEW	SZE
DIM	FOR
NXT	END

(space)

4. Display Error Codes: Causes an annotated list of TINY BASIC error codes to be displayed on the TV-screen. The error messages are fairly self-explanatory. However, they are covered in-depth in the section on TBX-TVCOS errors. Return to the operating system is accomplished by keying the space bar on the keyboard.

5. Continue Programming: Causes re-entry into TINY BASIC and displays the contents of the TV-screen as it was when exit was effected. The display is the only difference between options 5 and 6. Option 5 is particularly useful in the debugging stage when it can be used as the user goes from TINY BASIC to the operating system to list the error messages; then uses option 5 to display the screen as it was before exiting to the operating system.

6. TINY BASIC: Causes entry to TINY BASIC with a cleared TV-screen.

Another feature of TBX-TVCOS is the capability to exit TINY BASIC and enter the operating system at any time that it is permissable to enter a character by keying an ESCape (233 octal). Should the user desire to exit TINY BASIC at any other time, this can be effected by using the reset hardware feature.

At any time the user detects an error prior to keying a Carriage-Return (215 octal), the error can be corrected or eliminated by one of two procedures:

1. If the error is a single character or small group of characters and it is detected as soon as it is keyed, it can be deleted by keying one delete character (377 octal) for EACH character to be deleted (working BACKWARDS across the line), or

2. Should the error to too "messy" to correct by deleting an individual character or group of characters, it may be better to delete the entire line. This is accomplished by keying a CTL-L (Control L; 214 octal).

A thorough familiarity with the features of TBX-TVCOS and the TINY BASIC verb/command set should enable the user to work effectively with the system.

III. DIFFERENCES: TBX-TVCOS vs. BASIC

In the interest of operational speed and storage economy it became necessary to impose some restrictions in the design and implementation of TBX-TVCOS. As a result, programs written in Standard BASIC (in any of its' many forms) will require some modification and, in some instances, extensive alteration before they can be executed.

Several of the verbs/commands differ in forms, format, options and execution from BASIC. For the most part, these differences are not severely restrictive and, in several instances, represent an improvement over the original. In any event, almost all of the verb/command differences can be resolved by imaginative and creative programming. These verb/command differences, where they exist, are explained in some detail in the command descriptions section.

The biggest difference between TBX-TVCOS and BASIC is in the definition and treatment of variables. There are only twenty-six possible simple or dimensioned variables in TBX-TVCOS and they are designated or identified by the single letters A through Z, whereas most versions of BASIC permit dimensioned variables to be identified by single letters and simple variables to be identified by a single letter followed by a single digit. TBX-TVCOS requires that all array variables, whether one or two dimensioned, have the dimension(s) of the array be explicitly defined by the use of a DIM verb/command; whereas most versions of BASIC provide implied dimensions of 10 for subscripted variables for which no explicit dimension(s) are provided. In TBX-TVCOS the only limitation on the dimensions of an array is the amount of available memory space since each occurance uses two bytes.

Variables in TBX-TVCOS can ONLY be integers with values in the range +32767 to -32768. When aritmetic is performed on variables it is possible to exceed these limitations. When the limits are exceeded, no error message is given and wrap-around occurs (e.g., +32767 +1 = -32768).

IV. VERBS AND COMMANDS OF TBX-TVCOS

TBX-TVCOS supports sixteen verbs and commands; there are four commands which can only be executed in the immediate mode and twelve verbs/commands which can be entered either as program statements or immediate execution commands. All input lines MUST be terminated with a Carriage-Return (cr=215 octal).

* * * * * * * * * * * * * * *

The four immediate execution commands are used in the construction, debugging and execution of programs. These commands are LST, NEW, RUN and SZE.

1. LST - List Programming. The LST command is used to cause all or selected parts of a program to be listed on the TV-screen. The LST command has three possible formats:

A.	LST	List entire program
B.	LST nnnnn	List line nnnnn
C.	LST sssss,eeeee	List lines sssss through eeeee inclusive

(NOTE: Both sssss and eeeee must be existing line numbers).

2. NEW - Start new programming. The NEW command is used whever it is desired to begin the entry of a new program from the keyboard. The NEW command resets all of the internal constants and values of the TBX-TVCOS system and deletes all programming resident in the system.

3. RUN - Execute (run) resident program. The RUN command is used to cause the execution of the currently resident program beginning with the statement with the LOWEST line number. In most cases, when a fully developed program is read in from a cassette, the only command that is required is the RUN command once operating systems Options 5 or 6 have been selected.

4. SZE - Display program size. The SZE command is used to cause TINY BASIC to display the amount of storage required by the currently resident program. The value displayed by the SZE command will be different once a program has been executed because prior to program execution, no space has been reserved for the variables used in the program. SZE actually displays two values: The first value is the amount of storage required by the program at the current moment; the second value is the amount of storage space remaining and available for use.

* * * * * * * * * * * * * * *

The remaining twelve verbs/commands are the TINY BASIC instruction set which is used in TINY BASIC programs. Any of the twelve can be executed in the immediate mode by entering or keying the verb/command without a preceeding line number.

The format for entering a line of coding for inclusion in a TINY BASIC program is as follows:

nnnnn (statement)cr

Where: nnnnn is the line number of the statement. TINY BASIC automatically arranges all of the lines of a program in ascending order.

$$0 < nnnnn < 65535$$

(statement) is an optional statement in the TINY BASIC language consisting of one of the twelve verbs/commands followed by its associated operands.

cr represents a Carriage-Return (215 octal).

Whenever a line is entered which has a line number that is a duplicate of an exixting line number, one of two things occurs:

1. If the new line contains only a line number immediately followed by a Carriage-Return, the old line is deleted and the current line is NOT included in the program. Therefore, all reference to a line with the input line number is deleted from the program; or

2. If the new line contains anything except a line number immediately followed by a Carriage-Return, it replaces the old line. NOTE: The line-delete function CTL-L described in the "Features of TBX-TVCOS" Section will NOT cause a line which is part of the current program to be deleted; it causes the line which is being input to be deleted before it is accepted by TINY BASIC.

A line can be inserted in a TINY BASIC program between two other lines by giving it a line number which falls between the line numbers of the other two lines.

Each of the twelve verb/commands has a specific format which MUST BE FOLLOWED EXACTLY. Every attempt has been made to provide examples which include as many variants of the verb/command as possible. The twelve verb/commands are DIM, DTA, END, FOR, GOSUB, GOTO, IF, IN, LET, NXT, PR and RET.

1. DIM - Assign array dimension(s): The DIM verb/command is used to define the dimension(s) of array variable(s). Array variables may have one or two dimensions. The formats of the DIM statement are as follows:

```
nnnnn DIM var(dim)                  One dimension
nnnnn DIM var(dim1,dim2)            Two dimensions
```

Where: nnnnn = Line number
var = One of the letters of the alphabet
dim = Number of elements in the array
dim_1 = First dimension of two dimensional array (matrix)
dim_2 = Second dimension of two dimensional array (matrix)

$dim/dim_1/dim_2$ may be a numeric literal or an expression

Examples of DIM:

```
 50 DIM X(30)                       One dimension
100 DIM Y(15,27)                    Two dimensions
150 DIM Z(12,Q+Z)                   Expression used in dimension statement
200 DIM A(10),B(20,5),C(30)         Multiple array dimensions assigned by
                                      one DIM statement
```

2. DTA - Assign value(s) to simple or single dimensioned array variable.

The DTA statement is analogous to a combination of both the DATA and READ statements in Standard BASIC. It functions differently than either of the standard statements but does provide the capability of easily assigning literal values to multiple variables in one statement.

The format of the DTA statement is as follows:

```
nnnnn DTA var=literal
```

In its' simplest form, the DTA statement is equivalent to a LET statement.

For example:

```
50 DTA X=3 is the same as
50 LET X=3
```

However, the DTA statement permits multiple assignment of literals to variables. As many var=literal sets as can be accomodated on a single 72 character line may follow the DTA verb/command provided that they are separated by semi-colons (;)s. For example:

```
100 DTA X=3;Y=4;Z=7;A=4. . . .
```

In addition, the DTA statement can be used to assign a string of values to consecutive elements of an array beginning with some pre-specified element through the last element of the array or until all of the literals in the literal string have been used.

For example:

```
100 DIM B(10)
200 DTA B(1)=10,9,8,7,6,5,4,3,2,1
```

This will assign consecutive elements of the literal string to consecutive elements of the array B such that B(1) = 10, B(2) = 9 etc.

If the DTA statement had the following form of:

200 DTA B(3)=10,9,8,7,6,5,4,3,2,1 then B(3) = 10, B(4) = 9, B(5) = 8 etc., and the values of B(1) and B(2) would be undefined relative to the effects of the DTA statement. The subscripted variable can be subscripted by another variable and take the form of:

200 DTA B(X)=10,9,8,7,6,5,4,3,2,1 and the value of X at the time that the DTA statement is executed will determine which elements of the array B will be assigned which values from the literal string.

3. END - End of Program. The END statement is normally the LAST statement to be executed in a BASIC program. Its function is to terminate execution of the program and return control to the user. While the END statement need not be the last instruction in sequence by line number, it is a good practice to make it so. The format of the END statement is:

```
nnnnn END
```

For example:

```
9999 END
```

4. FOR - Beginning of a loop. The FOR verb/command is used to initiate and control the execution of a program loop. It is used to assign the initial and the terminal values for the loop control variable. In Standard BASIC, it is also possible to specify the value of the increment to be applied to the loop control variable; whereas in TINY BASIC the STEP or increment for the loop control variable is fixed and has a value of 1. The format of the FOR statement is as follows:

```
nnnnn For var=expression-1 TO expression-2
Where:  nnnnn is the statement line number
        var is the loop control variable and can be any valid variable
          whether simple of subscripted

        expression-1 is any valid expression which can be resolved to a
          starting value for the loop control variable and is less in
          value than expression-2

        expression-2 is any valid expression which can be resolved to a
          terminal value for the loop control variable and is greater in
          value than expression-1
```

expression -1/-2 can be a numeric literal, any valid variable or an arithmetic expression consisting of a number of numeric literals and/or valid variables connected by the arithmetic operators (+,-,*,/); the arithmetic expression may be as complex as necessary utilizing parentheses as required to insure clarity and eliminate ambiguities.

The FOR statement is used in conjunction with the NXT statement to control the loop. Typical examples of a FOR-NXT loop are as follows:

```
200 FOR Q=1 TO 1Ø
    ---
    --- (loop)
    --- (instructions)
    ---
300 NXT Q
```

Control variable used as a counter to control number of iterations of the loop.

```
500 FOR I=2 TO 23
    ---
    ---
    LET A(I)=Ø
    ---
650 NXT I
```

Control variable used as a combination counter and subscript

```
1010 FOR Z=(((1Ø*Y)+(6*Q))/M+7 TO (((15*Y)-(4*Q))*N)-26
     ---
     ---
     ---
1216 NXT Z
```

Complex expressions used as starting and terminal values of the loop control variable

The starting and terminal values of the loop control variable are computed and fixed when the loop is begun. Therefore, altering the values of any variables which are used to compute them while within the loop will not effect the values of the starting and terminal values. Caution must be exercised when permanently exiting a loop by means other than falling through the NXT statement because a loop is still in effect until the terminal condition is satisfied and a loop terminating NXT statement is exercised (see "Programming Techniques" Section).

5. GOTO- Alter program execution sequence:/-Jumping to a line other than the next one in sequence. The GOTO verb/command is used to control the sequence of execution in a program. The format of the GOTO statement is as follows:

nnnnn GOTO expression

Where: nnnnn is the statement line number

expression is any valid expression which can be resolved into a number which is equal to an existing line number in the currently resident program.

NOTE: The GOTO statement of TINY BASIC differs from the GOTO statement of Standard BASIC in that the destination line number can be expressed in the form of an expression which gives the capability of generating computed GOTO's; Standard BASIC requires that the destination be expressed as a numeric literal line number. The following are examples of the GOTO statement:

```
270 GOTO 490              Explicit destination line number

540 GOTO X                Destination line number equal to value of X

1327 GOTO ((Z*Y)+3)/Q     Computed line number equal to value of expression
```

6. GOSUB - Jump to subroutine and save address of next sequential line number for return using the RET statement. The GOSUB verb/command is used to cause TINY BASIC to alter the sequence of program execution by jumping to the statement whose line number follows the word GOSUB and to begin sequential execution at that point and to continue from there until a RET statement is encountered whereupon sequential execution resumes at the statement immediately following the GOSUB statement. The format of the GOSUB statement is as follows:

nnnnn GOSUB expression

Where: nnnnn is the statement line number

expression is any valid expression which can be resolved into a number which is equal to an existing line number in the currently resident program (see NOTE in GOTO statement description).

Examples of the GOSUB statement are as follows:

```
 135 GOSUB 1540              Explicit destination line number

1645 GOSUB Y                 Destination line number equal to value of Y

26455 GOSUB  ((P+Q)*R)-S/T   Computed line number equal to value of expression
```

7. IF - Conditional Statement. The IF statement is used to test to determine whether or not a given condition exists. The format of the IF statement is:

nnnnn IF expression-1 condition expression-2 imperative-statement

Where: nnnnn is the line number of the statement

expression-1 is any valid expression

expression-2 is any valid expression

condition is one of the following relational operators:

```
=  EQUAL
<  LESS THAN
>  GREATER THAN
=< EQUAL OR LESS THAN
=> EQUAL OR GREATER THAN
<> NOT EQUAL (LESS THAN or GREATER THAN)
```

imperative-statement is any valid TINY BASIC statement, including another IF statement which is to be executed if the required condition is true.

Should the required condition be false, the imperative-statement will not be executed and control will be passed to the statement immediately following the IF statement in sequence. Many versions of Standard BASIC require that the imperative-statement be an implied or explicit GOTO. TINY BASIC permits the imperative-statement to be any valid TINY BASIC statement. Should the imperative-statement be another IF statement, there exists an implied AND between the two (or more) IF statements and both must be true if the final imperative-statement is to be executed; if either (or any) is false, the imperative-statement will not be executed. Examples of the IF statement are as follows:

```
 260 IF A=B GOTO 460

 555 IF (C+D)*X>Y LET Z=1

1378 IF A<>B IF C<>D IF E=F GOSUB 1582
```

8. IN - Enter (input) a value from the keyboard. The IN verb/command causes TINY BASIC to display a "?" on the TV-screen requesting the user to enter a value. The format of the IN statement is as follows:

nnnnn IN var_1 ,var_2....,var_n

Where: nnnnn is the statement line-number

var_1-var_n are a series of one or more variables which are to be assigned values equal to the values entered.

If more than one variable is specified in the IN statement, the values are entered one at a time with a Carriage-Return required between each value entered. Standard BASIC usually permits multiple inputs to be separated by commas (,)s. This is NOT permitted by TINY BASIC. Examples of the IN statement are as follows:

```
390 IN A                    Enter single value

922 IN X,Y(3),Z(15,4)       Enter multiple values
```

NOTE: If the variables on a multiple variable IN statement are separated by semi-colons, they can all be entered as one line even though each is terminated by a Carriage-Return.

9. LET - Assign a value to a variable. The LET verb/command is the workhorse of BASIC. The LET statement causes TINY BASIC to assign a value to a variable. In some versions of BASIC, the word LET can be omitted from the LET statement. However, TINY BASIC requires that the word LET be present. The format of the LET statement is as follows:

nnnnn LET var=expression

Where: nnnnn is the statement line number

var is the simple or subscripted variable to which a value is being assigned

expression is any valid expression in TINY BASIC

Examples of the LET statement:

```
 227 LET A=1                    Assign numeric literal

 495 LET X(4)=B(7,9)            Assign value of another variable

1111 LET Z(Y)=(((100*A)+73)/M*2)
```

10. NXT - End of loop. The NXT statement is used to terminate the sequence of instructions that make up a loop. The function of the NXT command is to cause the loop control variable to be incremented and tested against the terminal condition specified in the logically preceeding FOR statement. If the terminal condition is not satisfied, control is transfered to the statement immediately FOLLOWING the FOR statement; if the terminal condition is satisfied, control is transfered to the statement following the NXT statement. The format of the NXT statement is as follows:

```
nnnnn NXT var
```

Where: nnnnn is the statement line number

var is the loop control variable

A typical NXT statement would appear as follows:

```
419 NXT X
```

11. PR - Print output data. The PR verb/command is used to cause TINY BASIC to print the data specified in the PR statement. As many print-items can be output by a single PR statement as can be accomodated on a 72 character command line. The print-items in a single print statement may be separated by either commas (,)s or semi-colons (;)s.

The PR statement was designed, initially, to be used with a teletype machine and, therefore, an output line is assumed to be 75 characters long. When print-items are separated by commas, the print-line is assumed to be divided into five 15 character fields; each print-item is left justified in the next sequential print field. When print-items are separated by semi-colons a single space is inserted between the print-items as they are printed.

Because the width of the TV-screen is 32 characters, the use of a comma as a print-item separator when columnar alignment is required presents some difficulties when more than two print-items are to be output by a PR statement. The reason for this is that the first two characters of the third field are on one line and the balance are on the following line.

A PR statement can be terminated in any of three ways. The first method of termination is to immediately follow the last print-item with the statement terminating Carriage-Return; this will cause the TV-screen scroll to be rolled up one line. The second method of termination is to follow the last print-item with a comma and then a Carriage-Return; this will cause the TINY BASIC print control routines to advance the print field pointer to be aligned to the next 15 character print-field but not cause a teletype Carr-Ret to be emitted (the TV-screen scroll will not necessarily be advanced one line). The third method

is to follow the last print-item with a semi-colon followed by a Carriage-Return; this will cause a single space to be emitted and suppress emission of a teletype Carriage-Return (the TV-screen scroll will not necessarily be advanced one line).

The format of a PR statement is as follows:

nnnnn PR [print-item][separator print-item...] [separator] cr

Where: nnnnn is the statement line number

print-item is an item to be printed; this can be a variable, an expression, a numeric-literal or a character literal. (A character literal is a string of characters which is preceeded and followed by a double-quote; e. g., "CHARACTER LITERAL").

separator is a comma or a semi-colon.

A PR statement does not require that any print-items or separators follow PR. A PR statement consisting of only a PR followed by a Carriage-Return will cause the current output line (even if it is empty) to be terminated by emitting a Carriage-Return.

The PR statement is probably the most useful of the verb/commands which can be executed immediately. This form of the PR statement causes TINY BASIC to act like a calculator if the print-item(s) are expressions. An example of the PR statement used in the immediate execution mode might be as follows:

```
PR (((3*5)+(21/7))*(42+117))     CR
```

would cause 2862 to be printed on the next line.

Examples of the PR statement as part of a program are as follows:

```
180 PR X                          Print the value of X

250 PR "THE VALUE OF X IS";X      Print the character-literal followed by
                                    the value of X

540 PR A,B,C                      Print the values of the variables A, B,
                                    and C aligned on print-field boundaries
```

The PR statement incorporates, for its exclusive use, one of the two functions available under TINY BASIC. This is the tab TB function. The format of the TB function as as follows:

TB(expression)

The expression in the parentheses following TB is evaluated and a number of spaces equal to the value of the expression (up to 255) is printed out. An expression which has a value of zero will emit 256 spaces and in TBX-TVCOS will

clear half of the TV-screen. The TB function can replace any print-item in a PR statement.

12. RET - Return from subroutine. The RET verb/command is used to terminate a subroutine. It causes program execution sequence to be returned to the statement following the GOSUB statement which initiated execution of the sub-routine.

* * * * * * * * * * * * * * *

RN There is one general function available under TBX-TVCOS. This is the function which generates a random number. The random number function is involked by inserting the letters RN wherever a variable might normally occur in an expression. Whenever RN is encountered, the random number generator generates a number in the range Ø<=RN<=10000.

V. PROGRAMMING TECHNIQUES

One of the best ways to begin learning how to program a computer in any specific language is to formulate a problem which can be solved by using the language. Think the problem through and write it down in normal English as specifically detailed as possible. After having done this, study the language and try to imagine how each of the verbs/commands could be used to solve a portion of the problem. Build your programs in sections with each section written to solve a specific part of the problem. Then link the sections together appropriately to create the program.

The program listed below is a fairly typical example of a simple program written to solve a simple problem. The program plays the number game TRAP. The object is to discover the value of an unknown number by trying to trap it between two known numbers. The program provides clues as to the relationship of the unknown number to the numbers guessed by the player.

As many variations as possible of the verbs/commands of TINY BASIC were used in constructing this program; some of them unnecessarily. Study of the program will reveal that the subroutine at line 8ØØØ could be inserted in place of the GOSUB and eliminate the RET. However, that would not have illustrated an example of a subroutine. Normally, subroutines are not coded separately unless they are executed from several places in the program by GOSUBs and where including the subroutine instruction in-line would lengthen the program.

Several lines in the program have more than one statement in them. The statements are separated by the $ character. A special feature of TINY BASIC permits multiple statement on one line provided they are separated by $'s. The use of this feature can be very handy when a program has grown too big for the available memory. To reduce the size of the program, examine it to find statements which can be combined on one line; every line which is eliminated by this method will save 3 bytes of memory.

Another memory saving technique is to eliminate all of the spaces in a program which are not part of character-literals in PR statements. The spaces

have no value to TINY BASIC except to make the program more readable.

* * * * * * * * * * * * * * *

Loops are one of the most common structures in programs. However, every now and again it is desirable to permanently exit a loop at a point other than the normal exit point. At the point where the exit is to be effected, insert coding similary to the following to avoid the problems caused by not clearing the loop in the normal manner:

```
        For X=1 TO 17
        ---
        ---
Other --> IF Z(X)=Ø LET X=17$NXT X$ GOTO nnnnn
  exit  ---
        ---
Normal--> NXT X
  exit
```

In situations where it is desired to have the increment of the loop control variable be other than 1 you can use combinations of other statements to accomplish the same effect as a FOR-NXT.

For example, you wish the starting value to be 27 and you wish the terminal value to be Ø (steping backwards) and you wish the increment to be -3. You could use the following coding to accomplish this:

```
100 LET V=27
110 ---
    ---
    ---
175 IF V>Ø LET V=V-3$ GOTO 11Ø
```

** SAMPLE TINY BASIC PROGRAM **

```
1                                                                    8
 1ØØ DIM X(2)                          Array variable X has two elements
 11Ø LET T=Ø                           Set variable T to zero
 12Ø LET R=RN/1Ø                       Set variable R to a value Ø<=R<=1ØØØ
 13Ø PR TB(Ø)$PR TB(Ø)                 Print 256 spaces twice to clear screen
 14Ø PR "OK I HAVE A NUMBER"           Print start message
 15Ø PR                                Print a blank line
 16Ø PR "ENTER FIRST NUMBER";          Print entry message inhibit carr-return
 17Ø IN X(1)                           Input variable X(1) from keyboard
 18Ø PR "ENTER SECOND NUMBER";         Print entry message inhibit carr-return
 19Ø IN X(2)                           Input variable X(2) from keyboard
 2ØØ LET T=T+1                         Add 1 to variable T used to count tries
 21Ø GOSUB 8ØØØ                        Go to subroutine which tests inputs
 22Ø IF Z=1 GOTO 25Ø                   If variable Z equals zero go to line 25Ø
 23Ø PR "GUESS AGAIN"                  Print message with carriage-return
 24Ø GOTO 15Ø                          Go to line 15Ø for next instruction
 25Ø PR "YOU GUESSED IT IN";T;"TRIES"  Print complex message with 3 print-items
 26Ø PR"AGAIN? Y=1, N=Ø";              Print entry message inhibit carr-return
 27Ø IN A                              Input variable A from keyboard
 28Ø IF A=Ø GOTO 9999                  If input value equal zero go to 9999
 29Ø GOTO 11Ø                          Otherwise go to line 11Ø
```

```
0                    4                    8
1                    0                    0
```

```
* TEST INPUTS SUBROUTINE

8000 LET Z=0                              Set variable Z to zero indicating no-hit
8100 FOR Y=1 TO 2                         Set loop control variable Y to a
                                               starting value of 1
                                               with a terminal value of 2
8200 PR X(Y);                             Print the Yth element of array X no c-r
8300 IF X(Y)<R PR "<NUMBER" $GOTO 8800    If the Yth element of array X is
                                               less than variable R
                                               print character-literal  NUMBER
                                               then go to line 8700
8400 IF X(Y)>R PR ">NUMBER" $GOTO 8800    See line 8300 test is greater than

* NOTE:  If tests in lines 8300 and 8400 fail execution falls through to here
*        and X(Y) must be equal to the generated number

8500 LET Z=1                              Set Z to 1 indicating a hit
8600 PR "=";                              Print character-literal = no carr-return
8700 PR R                                 Print the generated number
8800 NXT Y                                Increment control variable Y test
                                               against terminal value if not
                                               greater than go to line 8200
                                               else fall through to next line
8900 RET                                  Return to line following GOSUB line 220
9999 END                                  End of program return to entry mode
```

* * * * * * * * * * * * * * *

IV. TBX-TVCOS ERROR MESSAGES

The form of error messages in TBX-TVCOS is as follows:

ERR ee nnnnn

Where: ee is the error number

nnnnn is the number of the line on which the error occurred.

There are 15 known error messages generated by TINY BASIC. They are listed below along with some suggested actions to take to fix the problem.

1. Input line too long (72 maximum)
   Fix: Break input line into two or more lines and re-enter them.

2. Input numeric overflow. Number out of range +32767=>number =>-32768
   Fix: Re-enter a number in the correct range.

3. Illegal character
   Fix: Re-enter the line in error.

4. No ending quote
   Fix: Re-enter the line in error with an ending quote.

5. Arithmetic too complex
   Fix: Rearrange the expression and, if required, break it into several parts and re-enter.

6. Illegal Arithmetic
   Fix: Re-enter with correct arithmetic.

7. Label not found
   Fix: Check program and either enter a line with the required label or fix the line which references the non-existant label.

8. Division by zero
   Fix: Check arithmetic and re-enter the error line or test for zero before dividing and jump around the divide if division=Ø.

9. Subroutines nested too deep
   Fix: Rearrange program logic to use fewer nested subroutines. Maximum nesting is 8.

10. RET with no GOSUB
    Fix: Determine how the program got to the RET without having executed a GOSUB and fix logic.

11. Illegal Variable. Probably a subscripted variable with no DIM.
    Fix: Examine error statement and correct as required.

12. Unrecognizable statement
    Fix: Examine the statement and replace it with a legal statement.

13. Parentheses error. The quantity of left parens MUST be equal to the quantity of right parens.
    Fix: Examine the statement and re-enter with corrections.

14. Out of memory. Program too big.
    Fix: See "Programming Techniques" Section on how to reduce program size.

15. DIM too large. Variable storage exceeds available memory.
    Fix: See "Programming Techniques" Section on how to reduce program size. Remember each occurance of a dimensioned variable takes 2 bytes.

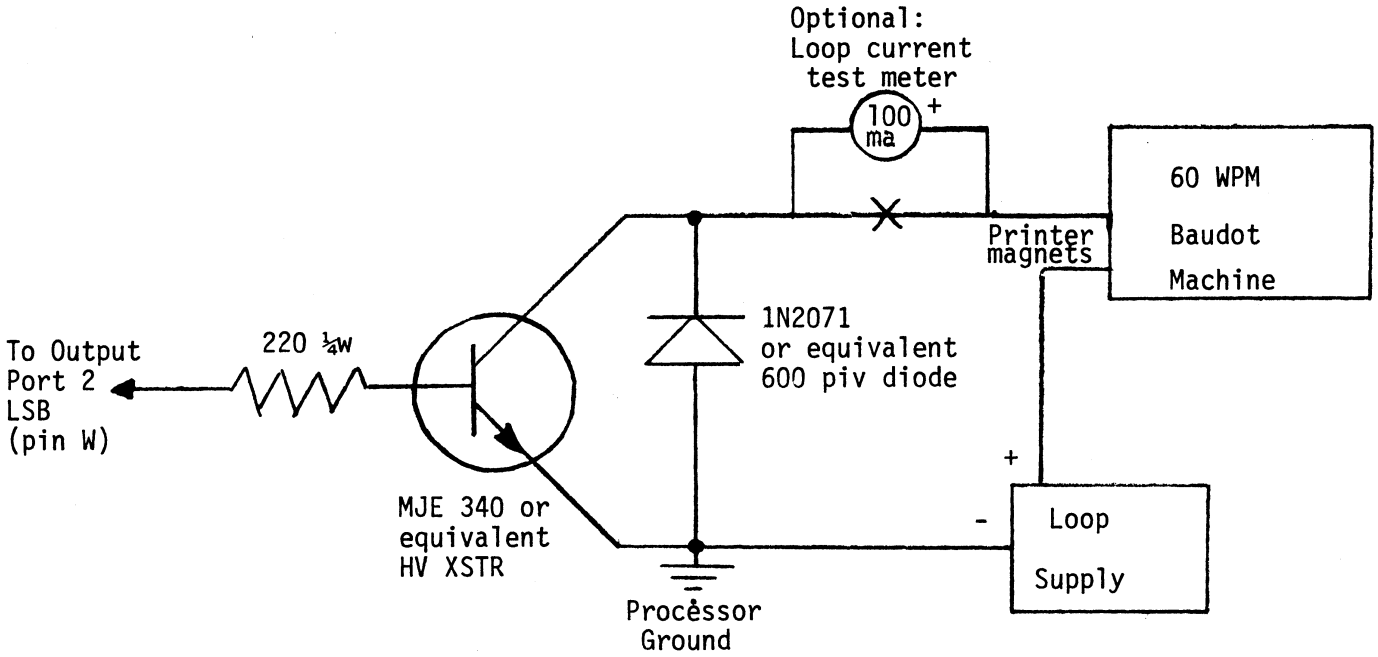
TINY BASIC -- BAUDOT

INTRODUCTION

TINY BASIC Extended of The Digital Group Software Systems, Inc. (DGSS) has been modified slightly to utilize a very low cost 5 level (Baudot) teletype machine for hardcopy. The hardcopy may be either a program listing for program maintenance or may show the actual program execution. Two non-interchangable versions are presently available; one for use with The Digital Group, Inc. (DGI) 8080 system and another for use on DGI Z-80 systems. Each system consists of the 1100 Baud audio cassette input, a 32 by 16 row standard TV cassette output, 10K or greater storage, and a 5 level (Baudot) hardcopy machine running at 60 WPM.

CONNECTIONS

The Baudot machine is driven through the LSB of Port 2 output. This is pin W of the 36 pin connector of the I/O board located between the CPU board and the TVC board. Since these hardcopy machines are typically driven by a high voltage loop supply, some means of interface between this loop and the output ports TTL level must be utilized. Several schemes have been used and yield similar results. The simple high voltage transistor scheme shown below certainly represents a low cost method:



INTERFACE THEORY OF OPERATION

The TTL to loop interface shown on the preceding page is simply an electronic open/closed switch to the 60 WPM Baudot machine.

The output ports voltage causes the emitter-collector connection to be "closed" when the TTL input is high (≃+3 volts) and "open" when the TTL input is low (≃ 0 volts). The diode prevents reverse voltage spikes from destroying the transistor or noise problems in the processor. The loop voltage is generally in the range of 100 - 150 volts and the loop current is adjusted to be around 60ma. Several excellent books are available which explain the basics about these machines and the loop supplies. Two books are as follows:

1. RTTY HANDBOOK published by TAB, Edited by Wayne Green; and

2. NEW RTTY HANDBOOK published by Cowan, Edited by Byron Kretzman.

These books are generally available at most "Ham Radio" supply stores.

The loop supply must be transformer isolated from the incoming 110 volts household AC.

TESTING THE CONNECTION

After assembling this little interface, connect it to the processor, machine, and loop supply with correct polarities and with all voltages off. Be very certain that the ground return on the processor is connected. Turn on the loop supplies and the machine. The machine should "run open". If not, the line to the diode/xstr is probably reversed and the diode is forward biased and closes the loop. Removing the I/O board (and others for convenience) should permit placing a temporary +5 volts on the output port without trouble. Turn on the processor supply and bring a +5 volt lead in series with another ≃ 220 ohm resistor over to the output port side of the infacing 220 ohm resistor. The printer should latch up (quiet down) when the voltage to the xstr's input is applied and run open (noisy) when the xstr's input voltage source is removed. If all tests OK, turn off everything and reassemble the processor. Be sure the interface input goes to Output Port 2, LSB. Remove any I/O devices and interfaces also connected to I/O Port 2 to avoid undesired commands to these devices.

LOADING TBX-Baudot

TBX-Baudot runs almost identical to the original DGSS TBX-TVCOS, so load the cassette exactly as if running TBX. When the Op Sys is displayed, you will notice that Options 4 and 5 are missing since the Baudot routines are located in the storage area formerly occupied by the Error Code listings. A "modified for hardcopy" BIORYTHM program is automatically loaded at the same time as the TBX-Baudot system. Typing LST should result in a hardcopy and a TV image of the programming in BIORYTHM. Typing RUN will result in a BIORYTHM listing for wives, neighbors, and friends but you run the risk of losing the use of your processor for a considerable amount of time!!

USING TBX-Baudot

TBX-Baudot can be interchangeably used for listing of running programs built under either TBX or TBX-Baudot on either 8080 or Z-80 systems. Since the output speed of the system is limited by the 60 WPM Baudot machine, it is generally preferable to build the program under TBX, make a cassette of the program, bring up TBX-Baudot, load in the program cassette, and list/correct. Refer to the TBX documentation for all the commands/error codes of TBX-Baudot (they are identical). Programs built under TBX-8080 will run on TBX-Z-80 and vice versa. However, TBX-80 itself will not---repeat not---work on a DGI Z-80 system nor vice versa. You will notice that the Z-80 TBX runs faster so programming loops will have to be reset. Also, some TBX programs may not have correct Carriage Return/Line Feed operation when running on the Baudot machine. Generally, simply reprogramming to add Carriage Return instead of "Video Wrap Around" will solve the problem.

FINE POINTS

Unfortunately, the Baudot code (≃ 64 characters possible) does not have all of the characters that the ASCII set has (256 characters possible). For this reason, several character substitutions have been used (such as (for < and ! for *, etc.). After a little practice, the users have found this to be no great problem.

Different character set options have been used on these older machines and, if possible, request one using "Standard Ham RTTY Character Set". Some companies (advertised in Ham magazines) offer character set conversions if required.

Often readjusting the machines' "Range" control and setting to the midpoint between faulty printing can improve the error rejection capabilities of the machine.

DGSS and DGI have been using an earlier version of this TBX-Baudot for several months and the ability to have low cost hardcopy has been greatly appreciated. "Even if it ain't fast, you have to admit it gets the job done cheap".

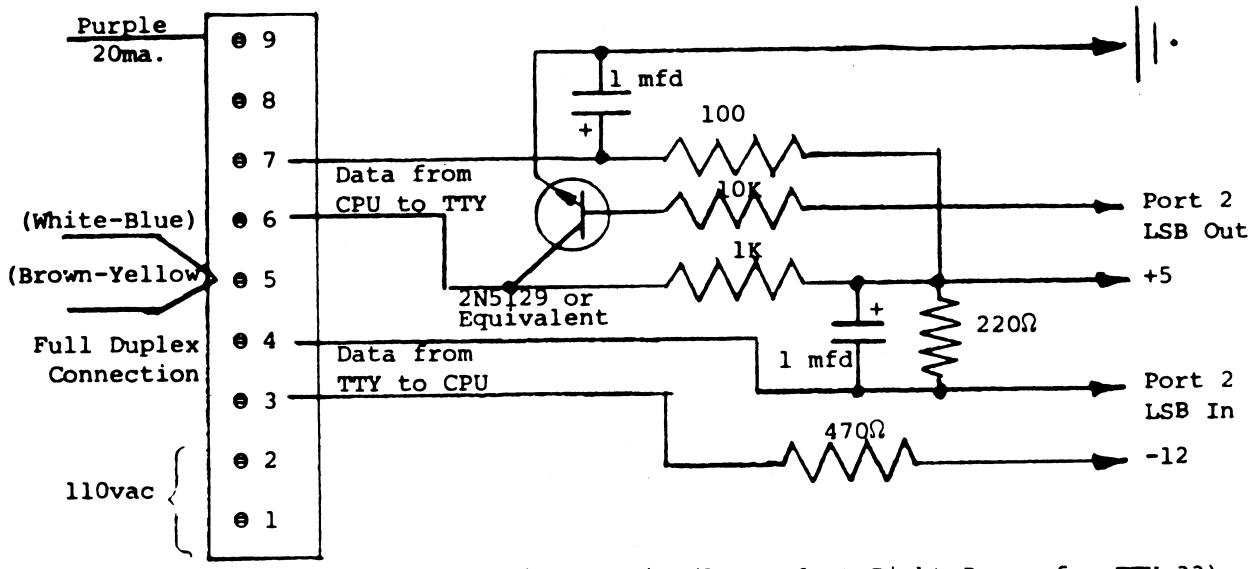
digital group software systems, inc. p.o. box 1086 arvada, colorado 80001

TBX Schematic ONLY

ASCII

ASCII

The ASCII machine is driven through the LSB of Port 2 output. This is pin W of the 36 pin connector of the I/O board located between the CPU board and the TVC board. The Teletype model 33 machine may be simply interfaced by using the following circuit. A few extra lines and parts are used to enable data entry from the model 33 if appropriate software were written to sample and deserialized the incoming data.



(Line Terminal Strip (located at Right Rear of a TTY 33)

Digital Group System Interface to TTY 33 20ma. loop

The Teletype 33 should be set up for Full Duplex operation with a 20ma. loop as per the manuals or the included circuit of the color coded leads.