$1.50

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

January, 1976 | Box 310, Menlo Park CA 94025 | Volume 1, Number 1

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Table of Contents for Volume, 1, Number 1 (20 pages) page

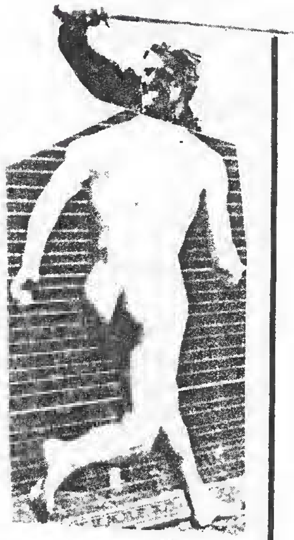
THIS IS A REPRINT OF THE ORIGINAL VOLUME 1, NUMBER 1

dr. dobb's journal of

Tiny BASIC Calisthenics & Orthodontia

Running Light Without Overbyte

Box 310, Menlo Park CA 94025 Volume 1, Number 1



STATUS LETTER

by Dennis Allison

The magic of a good language is the ease with which a particular idea may be expressed. The assembly language of most microcomputers is very complex, very powerful, and very hard to learn. The Tiny BASIC project at PCC represents our attempt to give the hobbyist a more human-oriented language or notation with which to encode his programs. This is done at some cost in space and/or time. As memory still is relatively expensive, we have chosen to trade features for space (and time for space) where we could.

Our own implementation of Tiny BASIC has been very slow. I have provided technical direction only on a sporadic basis. The real work has been done by a number of volunteers; Bernard Greening has left the project. As might be guessed, Tiny BASIC is a tiny part of what we do regularly. (And volunteer labor is not the way to run a software project with any kind of deadline!)

While we've been slow, several others have really been fast. In this issue we publish a version of Tiny BASIC done by Dick Whipple and John Arnold in Tyler, Texas. (And other versions can't be far behind.)

MY, HOW TINY BASIC GROWED!

Once upon a time, in *PCC*, Tiny BASIC started out to be:

† a BASIC-like language for tiny kids, to be used for games, recreations, and the stuff you find in elementary school math books.

† an exercise in getting people together to develop FREE software.

† portable-machine independent.

† open-ended--a toy for software tinkerers.

† small.

Then . . . (fanfare!) . . . along came Dick Whipple and John Arnold. They built Tiny BASIC Extended. It works. See pp 13-17 and 19 in this issue for more information. More next issue.

WANTED: More Tiny BASICs up and running.

WANTED: More articles for this newsletter.

WANTED: Tiny other languages. I might be able to live with Tiny FORTRAN but, I implore you, no Tiny COBOL! How about Tiny APL? Or Tiny PASCAL (whatever that is)?

WANTED: Entirely new, never before seen, Tiny Languages, imported from another planet or invented here on Earth. Especially languages for kids using home computers that talk to tvs or play music or run model trains or . . .

BASIC

BASIC, Beginners' All-purpose Symbolic Instruction Code, was initially developed in 1963 and 1964 by Professors John Kemeny and Thomas Kurtz of Dartmouth College, with partial lsupport from the National Science Foundation under the terms of Grant NSF GE 3864. For information on Dartmouth BASIC publications, get *Publications List* **(TM 086) from Documents Clerk, Kiewit Computation Center, Dartmouth College, Hanover NH 03755. Telephone 603-646-2643.**

Try these: **TM028** *BASIC: A Specification* **$3.15**
TM075 *BASIC* **$4.50**

****
It would help a lot if you would each send us a 3x5 card with your name, address (including zip), telephone number, and a rather complete description of your hardware.
****

DRAGON THOUGHTS

† We promised three issues. After these are done, shall we continue?

† If we do, we will change the name and include languages other than BASIC.

† This newsletter is meant to be a sharing experience, intended to disseminate FREE software. It's OK to charge a few bucks for tape cassettes or paper tape or otherwise recover the cost of sharing. But please make documentation essentially free, including annotated source code.

† If we do continue, we will have to charge about $1 per issue to recover our costs. In Xeroxed form, we can provide about 20-24 pages per issue of tiny eye-strain stuff. If we get big bunches of subscriptions, we'll print it and expand the number of pages, depending on the number of subscribers.

† So, let us know . . . shall we continue?

For our new readers, and those who have been following articles on Tiny BASIC as they appeared in *People's Computer Company*, **we have reprinted on pages 3-12 the best of Tiny BASIC from** *PCC* **as an introduction, and as a reference.**

TECHNIQUES & PRACNIQUES

by Dennis Allison, 12/1/75

(This will be a continuing column of tricks, algorithms, and other good stuff everyone needs when writing software. Contributions solicited.)

16-BIT BINARY TO DECIMAL CONVERSION ROUTINE

- † saves characters on stack
- † performs zero suppressed conversion
- † uses multiplication by 0.1 to obtain n/10 and n mod 10

```
define crutch = OFFH;
declare n, u, v, t; BIT (16)        These could be registers, or on the
if n < 0 then                       stack
      do;
         n = -n;
         call outch('-');
      end;
call push (crutch)                  The crutch marks the end of num-
                                    ber on the stack
repeat;
   v = shr (n,1);
   v = v + shr (v,1);               These all are 16 bit shifts
   v = v  + shr (v,4);              Computes [n/10] or [n/10] - 1 by
   v = v + shr (v,8);               multiplication
   v = shr (v,3);                   Call it x
   t = v + v;
   u = t + t;                       Computes 10 - x
   u = u + u + t
   u = n - u                        Byte only as high order must be
                                    equal
   if u ≥ 10 then                   Perhaps one could use a decimal
         do;                        feature here
            u = u - 10;
            n = v + 1;              Corrects for case where [n/10] - 1
         end                        is computed and creates [n/10] and
   else                             n mod 10
            n = v;
   call push (u);                   Saves result on stack
until n = 0;                        Loop at least once

ch = pop;
do while < > crutch;                Write result in reverse order
   call outch (ch + 030H);          Converts digits to ASCII
                                    0 = 030H  02 = 032H  etc.
   ch = pop;                        Pop takes one word off the stack
end
```

† Letters from readers are most welcome. Unless they note otherwise, we will assume we are free to publish and share them.

† We hereby assign reprint rights to all who wish to use *Tiny BASIC Calisthenics & Orthodontia* for non-commercial purposes.

† To facilitate connection between our subscribers, we will in subsequent issues publish our subscriber list (including addresses and equipment of access/interest).

PCC Tiny BASIC Reorganizes. . .

12-15-75

Bob Albrect
Dennis Allison

Tiny Basic feels like a dead Albitross around my neck. I do not feel like working on it any more.

Bernard

. . . and so we procede somewhat more slowly than some of our readers

BUILD YOUR OWN BASIC

by Dennis Allison & Others
(reprinted from *People's Computer Company* Vol. 3, No.4)

A DO IT YOURSELF KIT FOR BASIC??

Yes, available from PCC with this newspaper and a lot of your time. This is the beginning of a series of articles in which we will work our way through the design and implementation of a reasonable BASIC system for your brand X computer. We'll be working on computers based on the INTEL 8008 and 8080 microprocessors. But it doesn't make much difference – if your machine is the ZORT 9901 or ACME X you can still build a BASIC for it. But remember, it's a hard job and will take lots of time particularly if you haven't done it before. A good BASIC system could easily take one man six months!

We'd like everyone interested to participate in the design. While we could do it all ourselves, (we have done it before) your ideas may be better than ours. Maybe we can save you, or you can save us, a lot of work or problems. Write us and we'll publish your letter and comments.

WHICH BASIC?

There is not any one standard BASIC (yet). The question is which BASIC should we choose to implement. A smaller (fewer statements, fewer features) BASIC is easier to implement and (more important) takes less space in the computer. Memory is still expensive so the smaller the better. Yet maybe we can't give up some goodies like string variables, dynamic array allocation, and so on.

There is a standard version of BASIC which is to be the minimal language which can be called BASIC. It's a pretty big language with lots of goodies. Maybe too big. Is there any advantage to being compatible with, say, the EDU BASICS? We don't have to make any decision yet; but the time will come . . .

DATA STRUCTURES

Data structures are places to put things so you can find them or use them later. BASIC has at least three important ones: a symbol table which looks up a program name, A or Z9 or A$, with its value. If we had a big computer where space was not a huge problem, we could simply preallocate all storage since BASIC provides for only 312 different names excluding arrays. When memory is so costly this doesn't make much sense. Somewhere, also, we've got to store the names which BASIC is going to need to know, names like LET and GO TO and IF. This table gets pretty big when there are lots of statements.

Lastly, we need some information about what is a legal BASIC statement and which error to report when it isn't. These tables are called parsing tables since they control the decomposition of the program into its component parts.

STRATEGY

Divide and Conquer is the programmers maxim. BASIC will consist of a lot of smaller pieces which communicate with each other. These pieces themselves consist of smaller pieces which themselves consist of smaller pieces, and so forth down to the actual code. A large problem is made manageable by cutting it into pieces.

What are the pieces, the building blocks of BASIC? We see a bunch of them:

* a supervisor which determines what is to be done next. It receives control when BASIC is loaded.
* a program and line editor. This program collects lines as they are entered from the keyboard and puts them into a part of computer memory for later use.
* a line executor routine which executes a single BASIC statement, whatever that is.
* a line sequence which determines which line is to be executed next.
* a floating point package to provide floating point on a machine without the hardware.
* terminal I/O handler to input and output information from the Teletype and provide simple editing (backspace and line deletion).
* a function package to provide all the BASIC functions (RND, INT, TAB, etc.)
* an error handling routine (part of the supervisor).
* a memory management program which provides dynamic allocation data objects.

These are the major ones. As we get futher into the system we'll begin to see others and we'll begin to be able to more fully define the function of each of these modules.

COMPILER OR INTERPRETER?

We favor using an interpreter. An interpreter is a program which will execute the BASIC program from its textual representation. The program you write is the one which gets executed. A compiler converts the BASIC program into the machine code for the machine it is to run on. Compiled code is a lot faster, but requires more space and some kind of mass storage device (tape or disk). Interpretative BASIC is the most common on small machines.

DIRECT MODE?

Some kind of "desk calculator" mode of operation would be nice. At least, we would like to be able to look at and set different variables in a program and restart execution at any given point. This feature makes it easier to find and gently terminate the existence of "bugs."

HOW MUCH MEMORY? AND . . . WHAT KIND?

Can we make some guesses about how big the BASIC system will be? Only if you don't hold us to it. Suppose we want to be able to run a 50 line BASIC program. We need about 800 bytes to store the program, another 60 or so bytes for storing program values (all numeric) without leaving any space for the interpreter and its special data. Past experience has shown that something like 6 to 8 Kbytes are needed for a minimum BASIC interpreter and that at least 12K bytes are necessary for a comfortable system. That's a lot of memory, but not too much more than you need to run the assembler. A lot of BASIC could be put into ROM (Read Only Memory) once developed and checked out. ROM is a lot cheaper than RAM (Read and Write) memory, but you can't change it. It's lots better to make sure everything works first.

But . . . if we can agree on some chunks of code and get it properly checked out, some enterprising person out there might make a few thousand ROMs and save us all some $$$. Let's see now . . . how about ROMs for floating point arithmetic, integer arithmetic, Teletype I/O . . .

TINY BASIC

Pretend you are 7 years old and don't care much about floating point arithmetic (what's that?), logarithms, sines, matrix inversion, nuclear reactor calculations and stuff like that.

And . . . your home computer is kinda small, not too much memory. Maybe its a MARK-8 or an ALTAIR 8800 with less than 4K bytes and a TV typewriter for input and output.

You would like to use it for homework, math recreations and games like NUMBER, STARS, TRAP, HURKLE, SNARK, BAGELS, . . .

Consider then, TINY BASIC

- Integer arithmetic only – 8 bits? 16 bits?
- 26 variables: A, B, C, D, . . ., Z
- The RND function – of course!
- Seven BASIC statement types
  - INPUT
  - PRINT
  - LET
  - GO TO
  - IF
  - GOSUB
  - RETURN
- Strings? OK in PRINT statements, not OK otherwise.

BUILD YOUR OWN BASIC--REVIVED

(reprinted from *People's Computer Company* Vol. 4, No. 1)

WHAT IS TINY BASIC???

TINY BASIC is a very simplified form of BASIC which can be implemented easily on a microcomputer. Some of its features are:

Integer arithmetic 16 bits only

26 variables (A, B, . . ., Z)

Seven BASIC statements

INPUT	PRINT	LET	GOTO
IF	GOSUB	RETURN	

Strings only in PRINT statements

Only 256 line programs (if you've got that much memory)

Only a few functions including RND

It's not really BASIC but it looks and acts a lot like it. I'll be good to play with on your ALTAIR or whatever; better, you can change it to match your requirements and needs.

TINY BASIC LIVES!!!

We are working on a version of TINY BASIC to run on the INTEL 8080. It will be an interpretive system designed to be as conservative of memory as possible. The interpreter will be programmed in assembly language, but we'll try to provide adequate descriptions of our intent to allow the same system to be programmed for most any other machine. The next issue of PCC will devote a number of pages to this project.

✱ In the meantime, read one of these.

Compiler Construction For Digital Computers, David Gries, Wiley, 1971 493 pages, $14.95

Theory & Application of a Bottom-Up Syntax Directed Translator Harvey Abramson, Academic Press, 1973, 160 pages, $11.00

Compiling Techniques, F.R.A. Hopgood, American Elsevier, 126 pages $6.50

A BASIC Language Interpreter for the Intel 8008 Microprocessor A.C. Weaver, M.H. Tindall, R.L. Danielson. University of Illinois Computer Science Dept, Urbana IL 61801. June 1974. Report No. UIUCDCS-R-74-658. Distributed by National Technical Information Service, U.S. Commerce Dept, Springfield VA 22151. $4.25.

A BASIC language interpreter has been designed for use in a microprocessor environment. This report discusses the development of 1) an elaborate text editor and 2) a table-driven interpreter. The entire system, including text editor, interpreter, user text buffer, and full floating point arithmetic routines fits in 16K 8-bit words.

The TINY BASIC proposal for small home computers was of great interest to me. The lack of floating point arithmetic however, tends to limit its usefulness for my objectives.

As a matter of a suggestion, consideration should be given to the optional inclusion of floating point arithmetic, logarithm and trigonmetric calculation capability via a *scientific calculator chip interface.*†

The inclusion of such an option would tend to extend the interpreter to users who desire these complex calculation capabilities. A number of calculator chip proposals have been made, with the Suding unit being of the most interest.

Thank you for the note of 13 June, regarding my letter on the Tiny BASIC article (PCC Vol. 3 No. 4). It was with regret that I learned that the series was not continued in the next volume. Even though few responded to the article published, conceptually the knowledge and principles which would be disseminated regarding a limited lexicon, high level programming language are of importance to the *independent* avocational microcomputer community.

At this time, PCC may not have a wide distribution in the avocation microcomputer community. This could be possibly the cause for the low number of respondies. Never the less, this should not detract from the dissemination and importance of concepts and principles which are of significance.

The thrust of my letter of 15 April, 1975, was to suggest a mechanism for the inclusion of F.P. in a limited lexicon and memory consumptive BASIC. I hope that the implication that F.P. must be included was not read into my letter.

It is my interest that information, concepts and the principles of compiler/interpreter construction as it related to microcomputers be available to the limited budget avocational user. The MITS BASIC, which you brought up, appears from my viewpoint to be a *licensed*, blackbox program which is not currently available to: (a) 8008 users, (b) IMP-16 users, (c) independent 8080 users (except at a very large expense) or (d) MC6800 users who will shortly be on line.

Presently it appears that microcomputer compiler interpretor function langauges will be coming available from a number of sources (MITS, NITS, Processor Technology and etc.). However, few will probably deal in the conceptualizations which are the basis of the interpreter. Information which will fill the void in the interpreter construction knowledge held by the avocation builder, should be made available.

I strongly urge that the series started with Vol. 3 No. 4 article be continued. Possibly the hardware, peripheral, machine programming difficulties incurred by the microcomputer builder, is prohibiting a major contribution at this time. However, I would expect that by Autumn a number of builders should have their construction and peripheral difficulties far enough along to start thinking about higher level languages. The previous objective for the article series sounds reasonable. It was not my purpose in submitting the letter to detract from the objective of a very limited lexicon BASIC, ie., to be attractive and usable by the young and beginner due to its simplicity.

If wives, children, neighbors or anyone who is not machine language or programming oriented is expected to use a home-base unit created under a restrained budget a high level language will be a necessity. It is with this foresight that I encourage the continuance of the "Build Your Own BASIC" series.

This issue aside, I would like to encourage the PCC to continue the quite creditable activities which have been its order of business with regard to avocational computing.

Michael Christoffer
4139 12th NE No. 400
Seattle, Wash. 98105

† Please see Dr Robert Suding's article on p. 18

DESIGN NOTES FOR TINY BASIC

by Dennis Allison, happy Lady, & friends
(reprinted from *People's Computer Company* Vol. 4, No. 2)

SOME MOTIVATIONS

A lot of people have just gotten into having their own computer. Often they don't know too much about software and particularly systems software, but would like to be able to program in something other than machine language. The TINY BASIC project is aimed at you if you are one of these people. Our goals are very limited--to provide a minimal BASIC-like language for writing simple programs. Later we may make it more complicated, but now the name of the game is keep it simple. That translates to a limited language (no floating point, no sines and cosines, no arrays, etc.) and even this is a pretty difficult undertaking.

Originally we had planned to limit ourselves to the 8080, but with a variety of new machines appearing at very low prices, we have decided to try to make a portable TINY BASIC system even at the cost of some efficiency. Most of the language processor will be written in a pseudo language which is good for writing interpreters like TINY BASIC. This pseudo language (which interprets TINY BASIC) will then itself be implemented interpretively. To implement TINY BASIC on a new machine, one simply writes a simple interpreter for this pseudo language and not a whole interpreter for TINY BASIC.

We'd like this to be a participatory design project. This sequence of design notes follows the project which we are doing here at PCC. There may well be errors in content and concept. If you're making a BASIC along with us, we'd appreciate your help and your corrections.

Incidentally, were we building a production interpreter or compiler, we would probably structure the whole system quite differently. We chose this scheme because it is easy for people to change without access to specialized tools like parser generator programs.

THE TINY BASIC LANGUAGE

There isn't much to it. TINY BASIC looks like BASIC but all variables are integers There are no functions yet (we plan to add RND, TAB, and some others later). Statement numbers must be between 1 and 255 so we can store them in a single byte. LIST only works on the whole program. There is no FOR-NEXT statement. We've tried to simplify the language to the point where it will fit into a very small memory so impecunious, tyros can use the system.

The boxes shown define the language. The guide gives a quick reference to what we will include. The formal grammar defines, **exactly** what is a legal TINY BASIC statement. The grammar is important because our interpreter design will be based upon it.

IT'S ALL DONE WITH MIRRORS------ OR HOW TINY BASIC WORKS

All the variables in TINY BASIC: the control information as to which statement is presently being executed and how the next statement is to be found, the returnaddressesof active GOSUBS----all this information constitutes the state of the TINY BASIC interpreter.

There are several procedures which act upon this state. One procedure knows how to execute any TINY BASIC statement. Given the starting point in memory of a TINY BASIC statement, it will execute it changing the state of the machine as required. For example,

100 LET S = A+6 cr

would change the value of S to the sum of the contents of the variable A and the interger 6, and sets the next line counter to whatever line follows 100, if the line exists.

A second procedure really controls the interpretation process by telling the line interpreter what to do. When TINY BASIC is loaded, this control routine performs some initialization, and then attempts to read a line of information from the console. The characters typed in are saved in a buffer, LBUF. It first checks to see if there is a leading line number. If there is, it incorporates the line into the program by first deleting the line with the same line number (if it is present) then inserting the new line if it is of nonzero length. If there is no line number present, it attempts to execute the line directly. With this strategy, all possible commands, even LIST and CLEAR and RUN are possible inside programs. Suicidal' programs are also certainly possible.

TINY BASIC GRAMMAR

The things in **bold face** stand for themselves. The names in lower case represent classes of things. '::=' is read 'is defined as'. The asterisk denotes zero or more occurances of the object to its immediate left. Parenthesis group objects. ϵ is the empty set. | denotes the alternative (the exclusive-or).

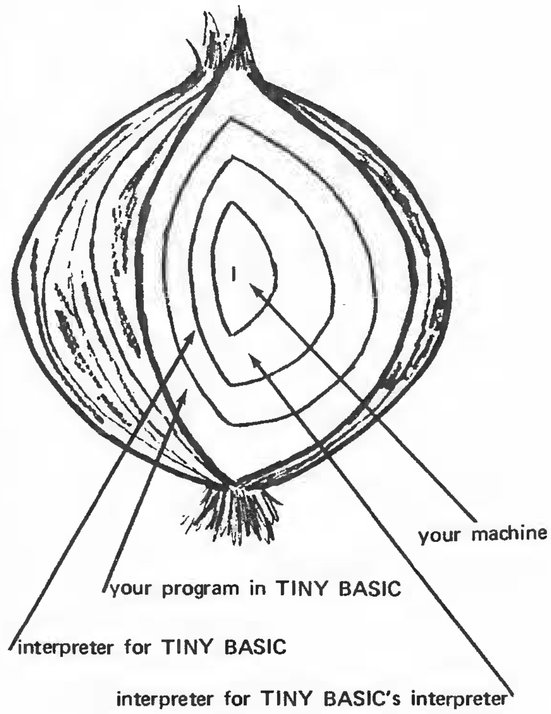
```
line::= number statement cr | statement cr
statement::= PRINT expr-list
             IF expression relop expression THEN statement
             GOTO expression
             INPUT var-list
             LET var = expression
             GOSUB expression
             RETURN
             CLEAR
             LIST
             RUN
             END
expr-list::= (string | expression) ( , (string | expression) *)
var-list::= var (, var)*
expression::= (+|-|ε) term ( (+|-) term)*
term::= factor ( (* | /) factor)*
factor::= var | number | (expression)
var::= A |B |C ... |Y |Z
number::= digit digit*
digit::= 0| 1 | 2 |... | 8 |9
relop::=< ( >| = | ε) | >(< | = | ε) | =
```

A BREAK from the console will interrupt execution of the program.

IMPLEMENTATION STRATGIES AND ONIONS

When you write a program in TINY BASIC there is an **abstract machine** which is necessary to execute it. If you had a compiler it would make in the machine language of your computer a program which emulates that abstract machine for your program. An **interpreter** implements the abstract machine for the entire language and rather than translating the program once to machine code it translates it dynamically as needed. Interpreters are programs and as such have their's as abstract machines. One can find a better instruction set than that of any general purpose computer for writing a particular interpreter. Then one can write an interpreter to interpret the instructions of the interpreter which is interpreting the TINY BASIC program. And if your machine is microprogrammed (like PACE), the machine which is interpreting the interpreter interpreting the interpreter interpreting BASIC is in fact interpreted.

This multilayered, onion-like approach gains two things: the interpreter for the interpreter is smaller and simpler to write than an interpreter for all of TINY BASIC, so the resultant system is fairly portable. Secondly, since the major part of the TINY BASIC is programmed in a highly memory efficient, tailored instruction set, the interpreted TINY BASIC will be smaller than direct coding would allow. The cost is in execution speed, but there is not such a thing as a free lunch.



LINE STORAGE

The TINY BASIC program is stored, except for line numbers, just as it is entered from the console. In some BASIC interpreters, the program is translated into an intermediate form which speeds execution and saves space. In the TINY BASIC environment, the code necessary to provide the

QUICK REFERENCE GUIDE FOR TINY BASIC

LINE FORMAT AND EDITING

- Lines without numbers executed immediately
- Lines with numbers appended to program
- Line numbers must be 1 to 255
- Line number alone (empty line) deletes line
- Blanks are not significant, but key words must contain no unneeded blanks
- '←' deletes last character
- X^c deletes the entire line

EXECUTION CONTROL

CLEAR delete all lines and data
RUN run program
LIST list program

EXPRESSIONS

Operators

Arithmetic		Relational	
+	−	>	>=
*	/	<	<=
		=	<>,><

Variables
A.....Z (26 only)

All arithmetic is modulo 2^{15} (±32762)

INPUT / OUTPUT

```
PRINT X,Y,Z
PRINT 'A STRING'
PRINT 'THE ANSWER IS'
INPUT X
INPUT X,Y,Z
```

ASSIGNMENT STATEMENTS

```
LET X=3
LET X= -3+5*Y
```

CONTROL STATEMENTS

```
GOTO X+10
GOTO 35
GOSUB X+35
GOSUB 50
RETURN
IF X>Y THEN GOTO 30
```

transformation would easily exceed the space saved.

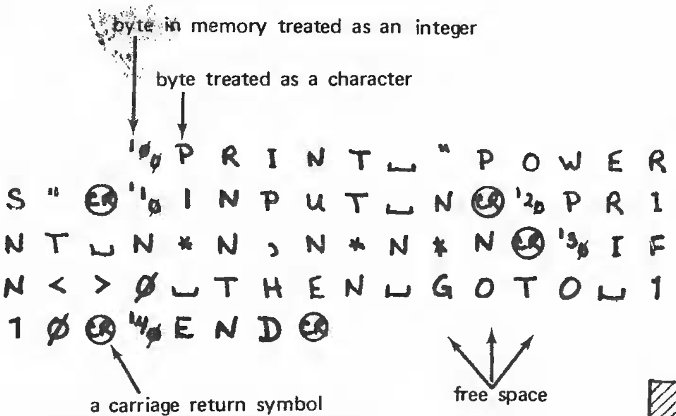
When a line is read in from the console device, it is saved in a 72-byte array called **LBUF** (Line BUFfer). At the same time, a pointer, **CP**, is maintained to indicate the next available space in LBUF. Indexing is, of course, from zero.

Delete the leading blanks. If the string matches the BASIC line, advance the cursor over the matched string and execute the next IL instruction. If the match fails, continue at the IL instruction labeled lbl.

The TINY BASIC program is stored as an array called **PGM** in order of increasing line numbers. A pointer, **PGP**, indicates the first free place in the array. PGP=0 indicates an empty program; PGP must be less than the dimension of the array PGM. The PGM array must be reorganized when new lines are added, lines replaced, or lines are deleted.

Insertion and deletion are carried on simultaneously. When a new line is to be entered, the PGM array searches for a line with a line number greater than or equal to that of the new line. Notice that lines begin at PGM (0) and at PGM (j+1) for every j such that PGM (j)=[carriage return]. If the line numbers are equal, then the length of the existing line is computed. A space equal to the length of the new line is created by moving all lines with line numbers greater than that of the line being inserted up or down as appropriate. The empty line is handled as a special case in that no insertion is made.

TINY BASIC AS STORED IN MEMORY



ERRORS AND ERROR RECOVERY

There are two places that errors can occur. If they occur in the TINY BASIC system, they must be captured and action taken to preserve the system. If the error occurs in the TINY BASIC program entered by the user, the system should report the error and allow the user to fix his problem. An error in TINY BASIC can result from a badly formed statement, an illegal action (attempt to divide by zero, for example), or the exhaustion of some resource such as memory space. In any case, the desired response is some kind of error message. We plan to provide a message of the form:

! mmm AT nnn

where mmm is the error number and nnn is the line number at which it occurs. For direct statements, the form will be:

! mmm

since there is no line number.

Some error indications we know we will need are:

1 Syntax error	5 RETURN without GOSUB
2 Missing line	6 Expression too complex
3 Line number too large	7 Too many lines
4 Too many GOSUBs	8 Division by zero

THE BASIC LINE EXECUTOR

The execution routine is written in the interpretive language, IL. It consists of a sequence of instructions which may call subroutines written in IL, or invoke special instructions which are really subroutines written in machine language.

Two different things are going on at the same time. The routines must determine if the TINY BASIC line is a legal one and determine its form according to the grammar; secondly, it must call appropriate action routines to execute the line. Consider the TINY BASIC statement:

GOTO 100

At the start of the line, the interpreter looks for BASIC key words (LET, GO, IF, RETURN, etc.) In this case, it finds GO, and then finds TO. By this time it knows that it has found a GOTO statement. It then calls the routine EXPR to obtain the destination line number of the GOTO. The expression routine calls a whole bunch of other routines, eventually leaving the number 100 (the value of the expression) in a special place, the top of the arithmetic expression stack. Since everything is legal, the XFER operator is invoked to arrange for the execution of line 100 (if it exists) as the next line to be executed.

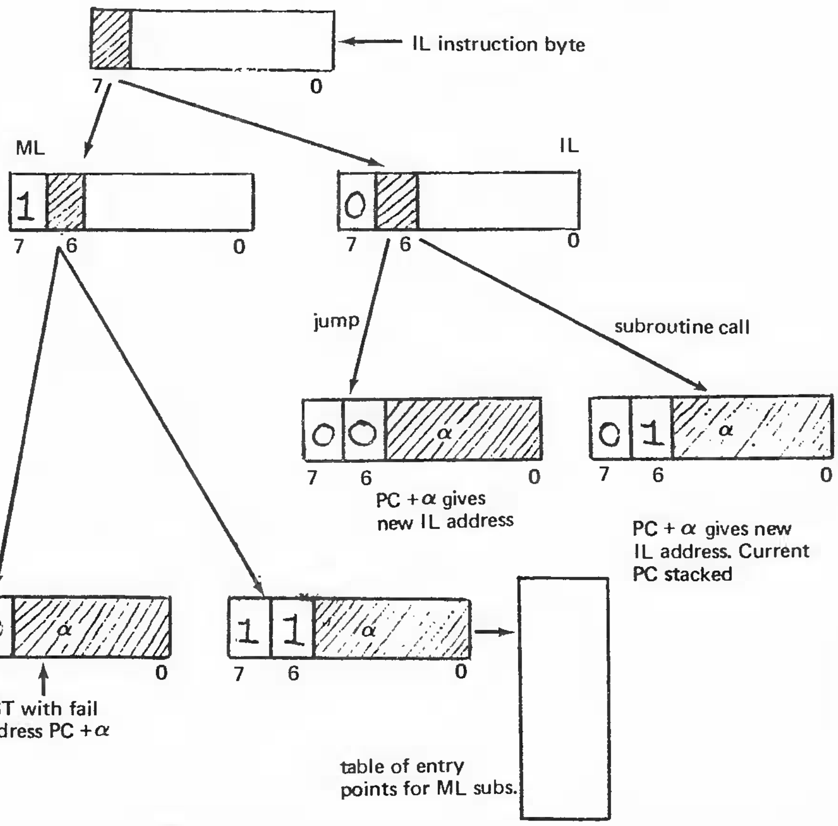
Each TINY BASIC statement is handled similarly. Some procedural section of an IL program corresponds to tests for the statement structure and acts to execute the statement.

ENCODING

There are a number of different considerations in the TINY BASIC design which fall in this general category. The problem is to make efficient use of the bits available to store information without loosing out by requiring a too complex decoding scheme.

In a number of places we have to indicate the end of a string of characters (or else we have to provide for its length somewhere). Commonly, one uses a special character (NUL = 00H for example) to indicate the end. This costs one byte per string but is easy to check. A better way depends upon the fact that ASCII code does not use the high order bit; normally it is used for parity

ONE POTENTIAL IL ENCODING



on transmission. We can use it to indicate the end (that is, last character) of a string. When we process the characters we must AND the character with 07FH to scrub off the flag bit.

The interpreter opcodes can be encoded into a single byte. Operations fall into two distinct classes---those which call machine language subroutines, and those which either call or transfer within the IL language itself. The diagram indicates one encoding scheme. The CALL operations have been subsumed into the IL instruction set. Addressing is shown to be relative to PC for IL operations. Given the current IL program size, this seems adequate. If it is not, the address could be used to index an array with the ML class instructions.

TINY BASIC INTERPRETIVE OPERATIONS

TST lbl, 'string'	delete leading blanks If string matches the BASIC line, advance cursor over the matched string and execute the next IL instruction. If a match fails, execute the IL instruction at the labeled lbl.
CALL lbl	Execute the IL subroutine starting at lbl. Save the IL address following the CALL on the control stack.
RTN	Return to the IL location specified by the top of the control stack.
DONE	Report a syntax error if after deletion leading blanks the cursor is not positioned to read a carriage return.
JMP lbl	Continue execution of IL at the label specified.
PRS	Print characters from the BASIC text up to but not including the closing quote mark. If a cr is found in the program text, report an error. Move the cursor to the point following the closing quote.
PRN	Print number obtained by popping the top of the expression stack.
SPC	Insert spaces to move the print head to next zone.
NLINE	Output CRLF to Printer.
NXT	If the present mode is direct (line number zero), then return to line collection. Otherwise, select the next sequential line and begin interpretation.
XFER	Test value at the top of the AE stack to be within range. If not, report an error. If so, attempt to position cursor at that line. If it exists, begin interpretation there; if not report an error.
SAV	Place present line number on SBRSTK. Report overflow as error.
RSTR	Replace current line number with value on SBRSTK. If stack is empty, report error.
CMPR	Compare AESTK(SP), the top of the stack, with AESTK(SP-2) as per the relation indicated by AESTK(SP-1). Delete all from stack. If condition specified did not match, then perform NXT action.
INNUM	Read a number from the terminal and push its value onto the AESTK.
FIN	Return to the line collect routine.
ERR	Report syntax error and return to line collect routine.
ADD	Replace top two elements of AESTK by their sum.
SUB	Replace top two elements of AESTK by their difference.
NEG	Replace top of AESTK with its negative.
MUL	Replace top two elements of AESTK by their product.
DIV	Replace top two elements of AESTK by their quotient.
STORE	Place the value at the top of the AESTK into the variable designated by the index specified by the value immediately below it. Delete both from the stack.
TSTV lbl	Test for variable (i.e. letter) if present. Place its index value onto the AESTK and continue execution at next suggested location. Otherwise, continue at lbl.
TSTN lbl	Test for number. If present, place its value onto the AESTK and continue execution at next suggested location. Otherwise, continue at lbl.
IND	Replace top of stack by variable value if indexes.
LST	list the contents of the program area.
INIT	Performs global initialization Clears program area, emptys GOSUB stack, etc.
GETLINE	Input a line to LBUF.
TSTL lbl	After editing leading blanks, look for a line number. Report error if invalid; transfer to lbl if not present.
INSRT	Insert line after deleting any line with same line number.
XINIT	Perform initialization for each stated execution. Empties AEXP stack.

A STATEMENT EXECUTOR WRITTEN IN IL

This program in IL will execute a TINY BASIC statement. The operators TST, TSTV, TSTN, and PRS all use a cursor to find characteristics of the TINY BASIC line. Other operations (NXT, XPER) move the cursor so it points to another TINY BASIC line.

THE IL CONTROL SECTION

```
START:  INIT                    ;INITIALIZE
        NLINE                   ;WRITE CR/LF
CO:     GET LINE                ;WRITE PROMPT & GET A LINE
        TSTL      XEC           ;TEST FOR LINE NUMBER
        INSRT                   ;INSERT IT (MAY BE DELETE)
        JMP       CO
STMT:   XINIT                   ;INITIALIZE FOR EXECUTION
```

STATEMENT EXECUTOR

```
STMT:   TST       S1,'LET'      ;IS STATEMENT A LET?
        TSTV      S16           ;YES: PLACE VAR ADDRESS ON AESTK.
        CALL      EXPR          ;PLACE EXPR VALUE ON AESTK.
        DONE                    ;REPORT ERROR IF cr NOT NEXT.
        STORE                   ;STORE RESULT:
        NXT                     ;AND SEQUENCE TO NEXT.
S1:     TST       S3,'GO'       ;GOTO OR GOSUB?
        TST       S2,'TO'       ;YES ...TO OR ...SUB.
        CALL      EXPR          ;GET LABEL.
        DONE                    ;ERROR IF cr NOT NEXT.
        XPER                    ;SET UP AND JUMP.
S2:     TST       S14,'SUB'     ;ERROR IF NO MATCH.
        CALL      EXPR          ;GET DESTINATION.
        DONE                    ;ERROR IF cr NOT NEXT.
        SAV                     ;SAVE RETURN LINE.
        XFER                    ;AND JUMP.
S3      TST       S8,'PRINT'    ;PRINT.
S4.     TST       S7,'"'        ;TEST FOR QUOTE.
        PRS                     ;PRINT STRING.
S5:     TST       S6,','        ;IS THERE MORE?
        SPC                     ;SPACE TO NEXT ZONE.
        JMP       S4            ;YES, JUMP BACK.
S6:     DONE                    ;NO, ERROR IF NO cr.
        NLINE
        NXT
S7:     CALL      EXPR          ;GET EXPR VALUE.
        PRN                     ;PRINT IT.
        JMP       S5            ;IS THERE MORE?
S8:     TST       S9,'IF'       ;IF STATEMENT.
        CALL      EXPR          ;GET EXPRESSION.
        CALL      RELOP         ;DETERMINE OPR AND PUT ON STK.
        CALL      EXPR          ;GET EXPRESSION.
        CMPR                    ;PERFORM COMPARISON-PERFORMS NEXT IF FALSE.
        JMP       STMT          ;GET NEXT STATEMENT.
S9:     TST       S12,'INPUT'   ;INPUT STATEMENT.
S10:    CALL      VAR           ;GET VAR ADDRESS.
        INNUM                   ;MOVE NUMBER FROM TTY TO AESTK.'
        STORE                   ;STORE IT.
        TST       S11,','       ;IS THERE MORE?
        JMP       S10           ;YES.
S11:    DONE                    ;MUST BE cr.
        NXT                     ;SEQUENCE TO NEXT.
S12:    TST       S13,'RETURN'  ;RETURN STATEMENT.
        DONE                    ;MUST BE cr.
        RSTR                    ;RESTORE LINE NUMBER OF CALL.[
        NXT                     ;SEQUENCE TO NEXT STATEMENT.
S13:    TST       S14,'END'
        FIN
S14:    TST       S15,'LIST'    ;LIST COMMAND.
        DONE
        LST
        NXT
S15:    TST       S16,'RUN'     ;RUN COMMAND.
        DONE
        NXT
S16:    TST S17,'CLEAR'         ;CLEAR COMMAND.
        DONE
        JMP START

S17:    ERR                     ;SYNTAX ERROR.

EXPR:   TST       E0,'-'
        CALL      TERM          ;TEST FOR UNARY -.
        NEG                     ;GET VALUE.
        JMP       E1            ;NEGATE IT.
E0:     TST       E1,'+'        ;LOOK FOR MORE.
        CALL      TERM          ;TEST FOR UNARY +.
E1:     TST       E2,'+'        ;LEADING TERM.
        CALL      TERM          ;SUM TERM.
        ADD
        JMP       E1
E2:     TST       E3,'-'        ;ANY MORE?
        CALL      TERM          ;DIFFERENCE TERM.
        SUB
        JMP       E3
E3: T2: RTN                     ;ANY MORE?

TERM:   CALL      FACT
TO:     TST       T1,'*'
        CALL      FACT          ;PRODUCT FACTOR.
        MPY
        JMP       TO
T1:     TST       T2,'/'        ;ANY MORE?
        CALL      FACT          ;QUOTIENT FACTOR.
        DIV
        JMP       TO

FACT:   TSTV      FO            ;VARIABLE.
        IND                     ;YES, GET THE VALUE.[
        RTN
F0:     TSTN      F1            ;NUMBER, GET ITS VALUE.
        RTN
F1:     TST       F2,'('        ;PARENTHESIZED EXPR.
        CALL      EXPR
        TST       F2,')'        ;MATCHING PARENTHESIS.
        RTN
F2:     ERR                     ;ERROR.

RELOP:  TST       RO,'='
        LIT       0             ;=
        RTN
RO:     TST       R4,'<'
        TST       R1,'='
        LIT       2             ;<=
        RTN
R1:     TST       R3,'>'
        LIT       3             ;<>
        RTN
R3:     LIT       1             ;<
        RTN

R4:     TST       S17,'>'
        TST       R5='
        LIT       5             ;>=
        RTN
R5:     TST R6,'<'
        LIT 3                   ;<>
R6:     LIT 4                   ;>
        RTN
```

TINY BASIC

by Dennis Allison, Bernard Greening, happy Lady, & lots of Friends
(reprinted from *People's Computer Company* Vol. 4, No. 3)

Dear People,

After a quick pique at TINY BASIC I have the following (possibly ill-considered) comments:
1. It looks useful for tiny computers, which is as intended.
2. Those accustomed to extended BASIC, or even the original Dartmouth BASIC, will be irked by its limitations. But then, that's how the bits byte!
3. How does the interpreter scan the word THEN in an IF statement?
4. Some of the comments for EXPR seem to be on the wrong line, or my reading is more biased than usual.
5. Users should note that arithmetic expressions are evaluated left-to-right unless subexpressions are parenthesized (i.e., there is no implicit operator procedure).
6. Real numbers would be nice, but would take up a lot more space. Probably too much. Ditto for arrays and string variables.
7. Please consider adding semicolon (i.e., unzoned) PRINT format with a trailing semicolon inhibiting the CRLF. This would be very useful and would be easy to add.
8. If INPUT prompts with a question mark, please print a blank character after the question mark (for readability).
9. I suggest allowing THEN as a separator in any multi-statement line, not just in IF statements. Since lines like

```
IF 5<X THEN IF X<10 THEN GOSUB 100
```

are already legal, why not allow lines like

```
LET A=B THEN PRINT A
```

or any other combination, including silly ones like

```
GOTO 200 THEN INPUT Z
```

the second statement of which would never be executed. If THEN works for IF, it should be possible to make it work for anything.
10. I also suggest allowing comments somehow. At present, comments must be held to a minimum
are possible via subterfuges such as

```
IF X<>X THEN PRINT "THIS IS A COMMENT"
```

but that seems kind of gauche. Naturally, comments must be held to a minimum in TINY BASIC, but sometimes they may be vital.
11. Doing a

```
PRINT "  "
```

seems to be the only way to print a blank line. Well, all right.
12. Exponentiation via ** would seem fairly easy to add, and might be worthwhile.
13. By the way, all of this will execute in 1K, won't it?

Jim Day
17042 Gunther St.
Granada Hills, CA 91344

Answering your Questions by number where appropriate:

3&4. Woops! There should be a TST instruction to scan the THEN. The comments are displaced a line. See the corrected IL listing in this issue.
5. Expressions are evaluated left-to-right with operator precedence. That is, 3+2*5 gives 13 and not 25. To see this, note that the routine EXPR which handles addition gets the operands onto the stack by calling TERM, and TERM will evaluate any product or quotient before returning.
7. Agreed, but this is intended as a minimal system.
9. One man's syntactic sugar is anothers poison. I don't like the idea. Incidentally, how would you interpret

```
LET A=B THEN GOSUB 200 THEN PRINT 'A'
```

The GOSUB then has to store a program address which botches up the line entry routine or one has to zap the GOSUB stack when an error is found. Both are solved only by kludges.
10-12. See 7.
13. Maybe. But 2K certainly. See below.

Dear PCC,

I am thrilled with your idea of an IL but I think that if you intend only to write a BASIC interpreter that a good symbolic assembler would be appropriate. With an assembler similar to DEC's PAL 3 or PAL B the necessary routines could be written and used in nearly the same way without having to write the associated run time material that would be necessary for its use as an interpreter. A command decoder, a text buffer, and a line editor would be necessary and all of this uses up a good amount of space in memory.

If you are aware of all these things and still plan to develop an IL interpreter, then I suggest you start as DEC did with a simple symbolic editor as the backbone of the interpreter. In this way you allow a 2800% increase in development and debugging speed (according to Datamation's comparison of interpreters and compilers whose fundamental difference is the on line editing capability). Once this has been implemented and IL is running on a particular system then the development of interpreters of all types is greatly simplified. By suggesting IL you have stumbled onto the most logical and easiest way to develop a complete library of interpreters. In addition to BASIC, it is very easy to write interpreters for: FOCAL, ALGOL, FORTRAN, PL 1, LISP, COBOL, SNOWBAL, PL/m, APL, and develop custom interpreters-with the ease with which one would write a long BASIC program!

As I pointed out earlier, all these features take up memory space and, as you have pointed out, run time is much slower. The way around this is to define the IL commands in assembly language subroutines then assemble the completed interpreter as calls to these subroutines. Thus the need for the IL interpreter as a run time space and time consumer is no longer necessary! (OK symbolic assembler haters, let's see you do this in machine language in less than ten man-years!)

In places where time and space are not so much of a problem, I suggest the addition of an interrupt handler and priority scheduler to allow IL to be used as a simplified and painless TIMESHARED system enabling many users to run in an interpreter and use more than one interpreter at once. Multi-lingual timeshare systems previously being available to those who have a highspeed swapping disk, drum, or virtual memory, are now avaliable to the user who has about 16K of memory and a method of equitably bringing interpreters in to main memory from the outside world (a paper tape reader or cassette system is the easiest to come by).

In short, IL as I suggested, in its minor stages would be a powerful software development aid; and in its final, most complex stages would provide a runtime system of unheard of inexpense.

I have heard from unofficial sources that ordinarily an interpreter or compiler requires ten man-years to write and debug to the point of use (if one man works the job would require 10 years, if 10 men work it would take one year). Since this is to be expected as the initial development of IL and since I have a general idea of the circulation of PCC, we should have IL up and running by the next issue of PCC!!

At this time I would like to request a few reprints of the article dealing with IL because I want to get some help from others in my school in getting a timeshared version working on our 16K PDP 8/m with DECTAPE. I seem to have lent my copy of that issue to one of the people I had been trying to get on this project and be has not returned it to me. Meanwhile, I need the article to begin initial work on the interpreter to insure compatibility with the version coming across through PCC. I will keep you posted as with regards to the development.

William Cattey
39 Pequet Road
Wallingford, Ct. 06492

The IL approach to implementation is quite standard and dates back to Schorre's META II, Gleenie's Syntax Machine, and numerous early compilers. It was widely used in the Digitek FORTRAN systems. We did not "stumble" on to the technique, we chose it with some deliberation.

You are right that a symbolic assembler can be used either to assemble the pseudocode into an appropriate form or to

expand the pseudocode into actual machine instructions with the attendant cost in space (and decrease in execution time). Our goal is a small, easily transportable system. The interpretive approach seems consistant with this primary goal. We are using the Intel 8080 assembler's macro facility to assemble our pseudocode.

I certainly agree that it is relatively easy (but not simple!) to implement other languages using the IL approach. From the users standpoint, provided he is not compute bound, there is little difference. Interpreters are often a bit more forgiving of errors and can give better diagnostics.

In my experience, your figure of 10 man-years is high for some languages and low for others. A figure of two to four man-years is probably more accurate, and that includes documentation at both the implementation and user level. Good luck on your implementation.

....I have found in my adaptation of it (TINY BASIC IL) for full use that certain commands need strengthening, while some might be dropped. I will hopefully be coming out with these possible modifications. Concerning my ideas on space trade-offs; I think an assembled version would take less space, since each command is treated as a subroutine call in a program made up of routines, while the interpreter needs a run time system in the background which, since it is interpretive in itself, takes up space.

P.S. You missed my allusion to assembler over strictly octal or hexidecimal op codes (my meaning was twofold). In DEC's PAL8 assembler the following syntax is needed to make the most efficient use of routine calling:

TSTN=JMSI (jump to subroutine indirectly via this location)
10: XTSTN

The assembler shows the binary as if TSTN were like a JMSI 100/ JMP to subroutine indirectly via 100 (requiring very very little extra space per routine—one word, to be exact).

I would be happy to resolve any questions regarding compilers vs. interpreters. (Datamation did an article on the writing of a standard program in several languages then documented development and run time.)

William Cattey

There are several different varieties of interpreters. One is simply a sequence of subroutine calls. Another is, as you suggest, a list of indirect references to subroutine calls. We are considering a different organization where the call address and some additional information is packed into a single byte. This is a good strategy vis a vis memory conservation only if the size of the code memory to decode the packed instruction plus the size of the encoded instructions is smaller than the size of a more straightforward encoding. This remains to be seen.

I guess I did miss your point on assemblers. However, let me assure you that I would never advocate making software by programming directly in hex or binary. Even an assembler seems cumbersome and difficult to me; I prefer a good systems language like PL/M!

Dear Dennis and other PCCers,

In my last crazily jumbled letter I made some comments about TINY BASIC. Here is the result of 2-3 days work and thinking about it. Instead of having an interpretive IL, I chose to set it up as detailed as possible, then have people with different machines code up subroutines to perform each IL instruction. I'm not convinced that this way would take more space, and I'm sure it would be faster.

There are a couple of changes in the syntax from your published version: separate commands from statements, add terminal comma to PRINT, and restrict IF-THEN to a line number (implied GOTO).

The semantics are separated out from the syntax in IL as much as possible. This should make it easier to be clear about what the results of any given syntatic structure. This is most apparent in the TST instructions, and the elimination of the NXT instruction. That one in particular was a confusion.

Please let me know how this fits with what you're doing. I don't have a micro yet—time, not money prevents it.

John Rible
51 Davenport St.
Cambridge, MA 02140

Because of space limitations, we have not been able to publish all of John Rible's version (dialect) of TINY BASIC. We'll probably include it in the first issue of the TINY BASIC NEWSLETTER. Limited space requires it to be in 2nd issue.

By seperating the syntax from the semantics he has produced a larger and possibly simpler to understand IL. There are more IL instructions so, I believe, the resultant system will be larger; further, the speed of execution is roughly proportional to the number of IL instructions (decoding IL is costly), it will be slower.

EXTENDABLE TINY BASIC

JOHN RIBLE

INTERMEDIATE LANGUAGE PHILOSOPHY

Instead of IL being interpreted, my goal has been to describe IL well enough that almost anyone will be able to code the instructions as either single machine language instructions or small subroutines. Besides speeding up TINY BASIC, this should decrease its size. Most of the instructions are similar to those of Dennis' (PCC V4 no. 2), but the syntactical has been seperated from the active routines. This would be useful if you want the syntax errors to be printed while inputting the line, rather than when RUNning the program.

Most subroutines (STMT, EXPR, etc.) are recursively called, so in addition to the return address being stacked, all the related data must be stacked. This can use up space quickly.

SYNTAX for John Rible's version of TINY BASIC

```
<PROGRAM>::=<PLINE>*1
<PLINE>::=<NUMBER><STATEMENT>
<ILINE>::=<COMMANO>|<STATEMENT>
<COMMANO>::= CLEAR CR|LIST CR|RUN CR
<STATEMENT>::=   CR |
                 LET<VAR>=<EXPR> CR|
                 GOTO<EXPR> CR|
                 GOSUB<EXPR> CR|
                 PRINT<EXPR-LIST>(,|ε) CR|
                 IF<EXPR><RELOP><EXPR>
                    THEN<STATEMENT> CR|
                 INPUT <VAR-LIST> CR|
                 RETURN CR|
                 ENO CR
<EXPR-LIST>::= (<STRING>|<EXPR>) (,(<STRING>|<EXPR>))*2
<STRING>::= "<ANY CHAR> *2"
<ANY-CHAR>::= any character except " or CR
<EXPR>::= (+|-|ε)<TERM>((+|-)<TERM>)*2
<TERM>::=<FACTOR>((*|/)<FACTOR>)*2
<FACTOR>::=<VAR>|<NUMBER>|(<EXPR>)
<VAR-LIST>::=<VAR>(,<VAR>)*2
<VAR>::= A|B|....|Y|Z
<NUMBER>::= <OIGIT><OIGIT>*3
<OIGIT>::= 0|1|....|8|9
<RELOP>::= <(=|>|ε)|>(=|<|ε)|=
```

notes: ε is null character
actual characters are in bold face
*1 repeat limited by size of program memory space
*2 repeat limited by length of line
*3 repeated 0 to 4 times

Dear Mr. Allison,

I was very interested in your Tiny BASIC article in PCC. Your ideas seem quite good. I have a few suggestions regarding your IL system. I hope I am not being presumptuous or premature with this. Unless I misunderstood you, your IL encoding scheme seems inadequate. For instance, IL JMPs must be capable of going up and down from the current PC. This means allotting one of the 6 remaining bits of the IL byte as a sign bit resulting in a maximum PC change of ±31 which is not adequate in some cases, ie. the JMP from just above S17 back to START. May I suggest the following scheme which is based on 2 bytes per IL instruction:

IL		ML		
JMP	CALL	TST	CALL	
$0XX_8$	$1XX_8$	$2XX_8$	$1XX_8$	(1st byte)
YYY_8	YYY_8	YYY_8	YYY_8	(2nd byte)

where XX= lower 6 bits of high part of address (assume upper 2 bits are 00)

YYY= all 8 hits of low part of address.

The complete address being $0XXYYY_8$. These addresses represent the locations associated with the IL and ML instructions. Note that if ϖ points to a table with a stored address, you have 3 bytes used— my scheme uses only 2 bytes with the same basic information.

I also wondered about the TST character string. In my implementation I am using the following technique: the string follows the TST byte pair immediately with a bit 7 set in the last character.

Example: 240, 006 } TST fail address in 040006_8

0 {L}

0 {E}

1 {T}

On the TSTL, TSTV, and TSTN IL's, it appears you need a ML address for the particular subroutine and 2 additional hytes for the fail address. At least this is how I am handling it.

I am looking forward to future articles in the series. Thanks again— keep up the good work!

P.S. I am co-owner of an Altair. We are writing our Tiny BASIC in Baudot to feed our Model 19's.

Richard Whipple
305 Clemson Dr.
Tyler, Tx. 75701

We found the same problem with the published IL interpreter. We solved it by doing a bit of rearranging and introducing a new operations code which does jumps relative to the start of the program, but has the same basic encoding. Your mechanization will, of course, work, but requires one more byte per IL instruction, may be harder to implement on some machines, and takes more code.

We are using the same scheme of string termination (i.e., using the parity bit) as you are. It's simple, easy to test, and difficult to get into the assembler.

There are a few errors and oversights in the IL language and in the interpreter you didn't mention. See the new listing in this issue.

Good luck. Keep us informed of your progress.

Dear People at PCC,

I have a couple of comments on Tiny BASIC:

S4 says TST S7, but S7 got left out. T1 says TST on my paper which I suppose should be TST T2.

What is LIT and all these "or 2000"? When are we going to start putting some of this into machine code?

Sincerely,

BOB BEARD
2530 Hillegass, No. 109
Berkeley CA 94704

Soon! Ed.

Dear Tiny BASIC Dragon,

Please scratch my name onto your list for Tiny BASIC Vol. 1. Enclosed is a coupon for 3 chunks of fire.

I am really enjoying my subscription to PCC, especially the article on Tiny BASIC.

Someday I am going to build an extended Tiny BASIC that will take over the world.

Basically yours,

RON YOUNG
2505 Wilburn, No. 144
Bethany OK 73008

Since the last issue came out, the IL code, macro definitions for each IL instruction, a subroutine address table for the assembly language routines that execute the IL functions, the assembly language code that executes the IL functions (all except the 16-bit arithmetic ones), and the IL processor have been punched on paper tape in source form.

HOP, TST, TSTN, and TSTL now do branches +32 relative to the current position counter. If the relative branch field has a zero in it, indicating a branch to "here", the IL processor prints out the syntax error message with the line number. The ERR instruction that was in the old IL code no longer exists.

IJMP and ICALL are used because the Intel 8080 assembler uses JMP and CALL as mnemonics for 8080 instructions. IJMP and ICALL are followed by one byte with an unsigned number from 0 to 255. This is added to START to do an indexed jump or call.

Bernard

corrected TINY BASIC IL

```
;
;  INTERPRETIVE LANGUAGE SUBROUTINES
;
EXPR:   TST     E0       ;TEST FOR UNARY '-'
        DB      '-' OR 2000
        ICALL   TERM     ;PUT TERM ON AESTK
        NEG              ;NEGATE VALUE ON AESTK
        HOP     E1       ;GO GET A TERM
;
E0:     TST     E01      ;TEST FOR UNARY '+'
        DB      '+' OR 2000
E01:    ICALL   TERM     ;PUT TERM ON AESTK
E1:     TST     E2       ;TEST FOR ADDITION
        DB      '+' OR 2000
        CALL    TERM     ;GET SECOND TERM
        ADD              ;PUT SUM OF TERMS ON AESTK
        HOP     E1       ;LOOP AROUND FOR MORE
;
E2:     TST     E3       ;TEST FOR SUBTRACTION
        DB      '-' OR 2000
        CALL    TERM     ;GET SECOND TERM
        SUB              ;PUT DIFFERENCE OF TERMS ON AESTK
        HOP     E1       ;LOOP AROUND FOR MORE
;
E3:     RTN              ;THIS CAN BE RECURSIVE
;
;
;
TERM:   ICALL   FACT     ;GET ONE FACTOR
T0:     TST     T1       ;TEST FOR MULTIPLICATION
        DB      '*' OR 2000
        ICALL   FACT     ;GET A FACTOR
        MPY              ;PUT THE PRODUCT ON AESTK
        HOP     T0       ;LOOP AROUND FOR MORE
;
T1      TST              ;TEST FOR DIVISION
        DB      '/' OR 2000
        ICALL   FACT     ;GET THE QUOTIENT
        DIV              ;PUT QUOTIENT ON AESTK
        HOP     T0       ;LOOP FOR MORE
;
T2      RTN              ;RETURN TO CALLER
;
;
;
FACT:   TSTV    F0       ;TEST FOR VARIABLE
        IND              ;GET INDES OF THE VARIABLE
        RTN
F0:     TSTN    F1       ;TEST FOR NUMBER
        RTN
F1:     TST     F1       ;ERROR IF ITS NOT A '('
        DB      '(' OR 2000
        ICALL   EXPR     ;THIS IS A RECURSIVE PROCESS
;
E1:     TST     FE1      ;EVERY '(' HAS TO HAVE A ')'
        DB      ')' OR 2000
        RTN
;
;
;
RELOP:  TST     R0       ;CHECK FOR '='
        DB      '=' OR 2000
        LIT     0
        RTN
;
R0:     TST     R4       ;CHECK FOR '<'
        DB      '<' OR 2000
        TST     R1
        DB      '=' OR 2000
        LIT     2
        RTN
;
R1:     TST     R3       ;CHECK FOR '>'
        DB      '>' OR 2000
        LIT     3
        RTN
;
R3:     LIT     1
        RTN
;
R4:     TST     R4
        DB      '>' OR 2000
        TST     R5
        DB      '=' OR 2000
        LIT     5
        RTN
;
R5:     TST     R6
        DB      '<'
        LIT     3
        RTN
;
R6:     LIT     4
        RTN
```

```
;
;
;  STATEMENT EXECUTOR WRITTEN IN IL (INTERPRETIVE LANGUAGE)
;  THIS IS WRITTEN IN MACROS FOR THE INTEL INTELEC 8/MOD 80
;  SYSTEM USING INTEL'S ASSEMBLER.
;
;  CONTROL SECTION
;
START:  INIT                      ;INITIALIZE
ERRENT: NLINE                     ;WRITE A CR-LF
CO:     GETLN                     ;WRITE PROMPT AND GET A LINE
        TSTL     XEC              ;IF NO LINE NUMBER GO EXECUTE IT
        INSRT                     ;INSERT OR DELETE THE LINE
        IJMP     CO               ;LOOP FOR ANOTHER LINE
XEC:    XINIT                     ;INITIALIZE FOR EXECUTION
;
;
;  STATEMENT EXECUTOR
;
STMT:   TST      S1       ;CHECK FOR 'LET'
        DB       'LE','T' OR 2000
SE1:    TSTV     SE1      ;ERROR IF NO VARIABLE!
SE2:    TST      SE2      ;ERROR IF NO "="
        DB       '=' OR 2000
        ICALL    EXPR     ;PUT EXPRESSION ON AESTK
        DONE              ;CHECK FOR CR LINE TERMINATOR
        STORE             ;PUT VALUE OF EXPRESSION IT ITS CELL
        NXT               ;CONTINUE NEXT LINE
;
;
S1:     TST      S3       ;CHECK FOR 'GO'
        DB       'G','O' OR 2000
        TST      S2       ;CHECK FOR 'GOTO'
        DB       'T','O' OR 2000
        ICALL    EXPR     ;GET THE LABEL
        DONE              ;CHECK FOR CR LINE TERMINATOR
        XFER              ;DO A 'GOTO' TO THE LABEL
;
;
S2:     TST      S2       ;CHECK FOR 'GOSUB', FAILURE IS AN ERROR!
        DB       'SU','B' OR 2000
        ICALL    XPER     ;PUT EXPRESSION ON AESTK
        DONE              ;CHECK FOR CR LINE TERMINATOR
        SAV               ;SAVE NEXT LINE NUMBER IN BASIC TEXT
        XFER              ;DO A 'GOSUB' TO THE LABEL
;
;
S3:     TST      S8       ;CHECK FOR 'PRINT'
        DB       'PRIN','T' OR 2000
S4:     TST      S7       ;CHECK FOR '"' TO BEGIN A STRING
        DB       '"' OR 2000
        PRS               ;PRINT THE DATA ENCLOSED IN QUOTES
S5:     TST      S6       ;',' MEANS MORE TO COME
        DB       ',' OR 2000
        SPC               ;SPACE TO NEXT ZONE
        HOP      S4       ;GO BACK FOR MORE
S6:     DONE              ;CHECK FOR CR LINE TERMINATOR
        NXT               ;CONTINUE NEXT LINE
;
;
S8:     TST      S9       ;CHECK FOR 'IF'
        DB       'I','F' OR 2000
        ICALL    EXPR     ;GET THE FIRST EXPRESSION
        ICALL    RELOP    ;GET THE RELATIONAL OPPERATOR
        ICALL    EXPR     ;GET THE SECOND EXPRESSION
S8A:    TST      S8A      ;CHECK FOR 'THEN'
        DB       'THE','N' OR 2000
        CMPR              ;IF NOT TRUE CONTINUE NEXT LINE
        IJMP     STMT     ;IF TRUE PROCESS THE REST OF THIS LINE
;
;
S9:     TST      S12      ;CHECK FOR 'INPUT'
        DB       'INPU','T' OR 2000
S10:    ICALL    VAR      ;GET THE VARIABLE'S INDEX
        INNUM             ;GET THE NUMBER FROM THE TELETYPE
        STORE             ;PUT THE VALUE OF THE VARIABLE IN ITS CELL
        TST      S11      ;',' MEANS MORE DATA
        DB       ',' OR 2000
S11:    DONE              ;CHECK FOR CR LINE TERMINATOR
        NXT               ;CONTINUE NEXT LINE
;
;
S12:    TST      S13      ;CHECK FOR 'RETURN'
        DB       'RETUR','N' OR 2000
        DONE              ;CHECK FOR CR  LINE TERMINATOR
        RSTR              ;RETURN TO CALLER
;
;
S13:    TST      S14      ;CHECK FOR 'END'
        DB       'EN','D' OR 2000
        FIN               ;GO BACK TO CONTROL MODE
;
;
S14:    TST      S15      ;CHECK FOR 'LIST'
        DB       'LIS','T' OR 2000
        DONE              ;CHECK FOR CR LINE TERMINATOR
        LST               ;TYPE OUT THE BASIC PROGRAM
        NXT               ;CONTINUE NEXT LINE
;
;
S15:    TST      S16      ;CHECK FOR 'RUN'
        DB       'RU','N' OR 2000
        DONE              ;CHECK FOR CR LINE TERMINATOR
        NXT               ;CONTINUE NEXT LINE
;
;
S16:    TST      S16      ;CHECK FOR 'CLEAR', FAILURE IS AN ERROR!
        DB       'CLEA','R' OR 2000
        IJMP     START    ;REINITIALIZE EVEHYTHING!
;
;
***
```

December 12, 1975

The Tyler Branch of the North Texas Computer Club is still having fun with Tiny BASIC as you can see by examining the print-out that follows. We are now calling it Tiny BASIC Extended after the addition of FOR-NXT loops, DIMension statements-arrays, and a few other goodies. The LIFE program was written by David Piper, a high school student of John's (he teaches at Robert E. Lee High School). David is working on KINGDOM now--we can hardly wait. Below are a few comments about our system and Tiny BASIC that may be of interest to your readers.

1. Our Altair 8800 is interfaced to a Model 19 Baudot Teletype at John's and via modems and a leased telephone line to a Model 15 Teletype at my house about 3/4 mile away. At present the system is strictly BAUDOT--no ASCII conversion whatsoever.

2. We use a Suding-type cassette interface that has been very reliable. 4K bytes load in about 1 minute 20 seconds.

3. The Tiny BASIC Extended takes about 2.9K bytes of memory.

4. The storage format for our Tiny BASIC is as follows: 2 byte statement label - 1 byte length of text - multibyte text - cr The statement label range is 1 to 65535. The "length of text byte" is used to speed up label searching in GOTO and other branching.

5. To conserve memory, we have shortened some commands to two or three letters (i.e., PR for PRINT, IN for INPUT, NXT for NEXT, etc.).

6. A "$" is used to write multi-statement lines. A "!" is used to suppress new line output in a PR statement. This allows continuing the next PR on the same line. The ";" provides one skipped space in a PR statement.

7. Functions currently on line are:

RN → generates random numbers between 0 and 10,000 decimal.

TB (exp) → TAB function in PR statement produces a number of skipped spaces equal to the value of "exp," an arithmetic expression.

8. Memory for arrays is allotted from the top of memory down while the program builds from the bottom up. If they cross, you get error message. Arrays may be 1 or 2 dimension. Max. size: 255 by 255.

9. Here are some BAUDOT equivalances:

:	= (equal to)
):	> = (greater than equal to)
(:	< = (less than equal to)
)(	<> (not equal to)
&	+ (plus)
#	* (times)

Parentheses are also used in arithmetic expressions. The system understands the difference by context.

10. FOR I=1,1000
    NXT I
    END   takes about 1.6 seconds to execute.

11. The colon is used as a Tiny BASIC prompt.

12. "?" is used as a rubout key and two LTR's keystrokes are used to begin a line over (LTR and FGS are keystrokes used to change case in Model 15/19 Teletypes)

13. Model 15/19 Teletypes are great machines and we have proved their worth to computer hobbyists!

Thanks again for your fine work at PCC, we remain
Yours Truly,

DICK WHIPPLE
305 Clemson Dr.
Tyler TX 75701

JOHN ARNOLD
Rt 4, Box 52A
Tyler TX 75701


```
00090 PR "LIFE WITH TINY BASIC EXTENDED"
00100 PR "SIZE";!
00105 LET F:0
00110 IN A
00112 PR""$ PR "THE BEGINNING-WAIT"$ PR""
00115 LET B:A&2
00120 DIM G(B,B),H(B,B)
00130 FOR J:1 TO B
00140 FOR I:1 TO B
00150 LET G(I,J):0$ LET H(I,J):0
00160 NXT I
00170 NXT J
00175 LET M:A&1
00180 FOR J:2 TO M
00190 FOR I:2 TO M
00200 IN K
00210 IF K (: 1 GO TO 220
00212 LET I:M
00214 GO TO 230
00220 LET G(I,J):K
00225 IF K : 1 LET F:F&1
00230 NXT I
00240 PR""
00250 NXT J
00260 PR "GENERATIONS";!
00270 IN D
00280 PR""
00285 PR""
00287 LET S:0
00290 FOR E:S TO D
00300 PR "GENERATION";E$PR""
00301 PR""
00302 IF F ) 0 GO TO 305
00303 PR "POPULATION IS ZERO"$PR""$END
00305 PR "POPULATION IS";F$PR""
00310 GO SUB 6000
00315 LET F:0
00320 GO SUB 5000
00330 NXT E
00335 PR "HOW MANY MORE";!$IN C$PR""
00336 PR""
00345 IF C : 0 END
00350 LET S:E$LET D:D&C
00355 GO TO 290
05000 FOR I:2 TO M
05010 FOR J:2 TO M
05020 LET N:0
05030 LET N:G(I-1,J-1)&G(I,J-1)&G(I&1,J-1)&G(I-1,J)&G(I&1,J)
05040 LETN:N&G(I-1,J&1)&G(I,J&1)&G(I&1,J&1)
05110 IF G(I,J) )( 1 GO TO 5180
05120 IF N ) 1 GO TO 5150
05130 LET H(I,J):0
05140 GO TO 5210
05150 IF N (: 3 GO TO 5200
05160 LET H(I,J):0
05170 GO TO 5210
05180 IF N )( 3 GO TO 5210
05200 LET H(I,J):1
05205 LET F:F&1
05210 NXT J
05220 NXT I
05230 FOR I:1 TO B
05240 FOR J:1TO B
05250 LET G(I,J):H(I,J)
05260 LET H(I,J):0
05270 NXT J
05280 NXT I
05290 RET
06000 FOR J:2 TO M
06010 LET R:0
06020 FOR Q:1 TO M
06030 IF G(Q,J) : 1 LET R:1
06040 NXT Q
06050 IF R:0 GO TO 6120
06060 FOR I:2 TO M
06070 IF G(I,J) : 1 GO TO 6100
06080 PR " ";!
06090 GO TO 6110
06100 PR "#";!
06110 NXT I
06120 PR ""
06130 NXT J
06140 PR""$PR""
06150 RET
:
```

```
:RUN

LIFE WITH TINY BASIC EXTENDED
SIZE ? 11
THE BEGINNING-WAIT

? 2
? 2
? 2
? 2
? 2
? 0 ? 0 ? 1 ? 1 ? 1 ? 1 ? 1 ? 1 ? 1 ? 2
? 2
? 2
? 2
? 2
? 2
GENERATIONS ? 3

GENERATION 0

POPULATION IS 7

   # # # # # # #

GENERATION 1

POPULATION IS 15

   # # # # #
   # # # # #
   # # # # #

GENERATION 2

POPULATION IS 12

     # # #
   #       #
 #           #
   #       #
     # # #

GENERATION 3

POPULATION IS 22

       #
     # # #
   # # # # #
 # #       # #
   # # # # #
     # # #
       #

HOW MANY MORE ? 1
GENERATION 4

POPULATION IS 16

     # # #
   #       #
 #           #
 #           #
 #           #
   #       #
     # # #

HOW MANY MORE ? 0
:
```

TINY BASIC, EXTENDED VERSION

by Dick Whipple (305 Clemson Dr., Tyler TX 75701)
& John Arnold (Route 4, Box 52-A, Tyler TX 75701)

INTRODUCTION

The version of TINY BASIC (TB) presented here is based on the design noted published in September 1975 *PCC* (Vol. 4, No. 2). The differences where they exist are noted below. In this issue we shall endeavor to present sufficient information to bring the system up on an Itel 8080-based computer such as the Altair 8800. Included is an octal listing of our ASCII version of TINY BASIC EXTENDED (TBX). In subsequent issues, structural details will be presented along with a source listing. A Suding-type cassette is now available from the authors (information to follow). We would greatly appreciate comments and suggestions from readers. Unlike some software people out there, we hope you *will* fiddle with TINY BASIC EXTENDED and make it *less Tiny*!

ABBREVIATED COMMAND SET

TB AND TBX			STANDARD BASIC
LET			LET
PR			PRINT
GOTO			GOTO
GOSUB			GOSUB
RET			RETURN
IF			IF
IN	In TB	In TBX	INPUT
LST			LIST
RUN			RUN
NEW *			NEW
SZE			SIZE
DIM			DIMENSION
FOR			FOR
NXT			NEXT

*CLEAR in original TB

TBX — HOW IT DIFFERS FROM TB

1. TBX system prompt is a colon ":".

2. Statement label values 1 to 65535.

3. Error correction during line entry:

   a) Rubout (ASCII 177_8) to delete a character. Prints a " ← ".

   b) Control L (Form Feed ASCII 014_8) to delete full line.

4. IN Statement: Termination of numeric input is accomplished by SPACE keystroke. All other terminations use CR (Carriage Return).

5. PR Statement: A comma is used for zone spacing while a semicolon produces a single space. A comma or semicolon at the end of a line surpresses CR and LF (Line Feed). To skip a line, use PR by itself.

6. DIM Statement: One or two dimensional arrays permitted. Array arguments can be expressions.

   Example:
   ```
   10   LET V = 10
   20   DIM A(10,10),B(2+V)
   . . .
   ```

   Array variables can be used in the same manner as ordinary variables.

7. FOR and NXT Statements: Step equal to 1 only. Iterative limits can be expressions. Nesting permitted. Care must be exercised when exiting a loop prior to completion of indexing. See Example.

   Example:
   ```
   10   LET X = 10
   20   FOR I = 1 to X
   30   LST Y = 2 * A+B
   40   IF Y=Z  I=X$NXT I$GOTO 60 *
   50   NXT I
   60   LET Y=3
   . . .
   ```

   * For explanation of "$" see no. 9.

8. Available Functions:
   a) RN: Random number generator. Range 0 ≤ RN ≤ 10,000. No argument permitted.
   b) TB(E): Tab function. In a PR statement, TB(E) prints a number of SPACE's equal to the value of expression "E".

9. The dollar sign can be used to write multiple statement lines.

   Example:
   ```
   10   IN B
   20   LET A=2*(B+1)$PRA$END
   ```

   When using an IF statement, a "false" condition transfers execution to the next numbered line. Thus in line 40 of the example of no. 7, the chained statements will not be executed unless a "true" condition is encountered.

10. LST Command: Can take anyone of three forms:

    a) LST CR— lists all statements in program
    b) LST a CR— lists only statement labelled a
    c) LST a,b CR— lists all statements between labels a and b inclusive.

11. SZE Command: Prints two decimal numbers equal to:

    a) Number of memory bytes used by current program.
    b) Number of memory bytes remaining.

    Note: Array storage included only after first execution of program.

12. Recording Programs on Cassette: Core dumps to cassette should begin at 033350 (split octal) and continue through address stored at

    033354 (low byte of address)
    033355 (high byte of address)

    Of course these cassette programs should be loaded back at 033350.

IMPLEMENTING TBX

Memory Allocation:

I. Misc. Storage (I/O Routines) 000000 to 000377*

II. TBX 020000 to 033377

III. TBX Programs 034000 to upper limit of memory.

* In our system we maintain a Monitor/Editor in the first 1K byte of memory. 3/4 K is protected and 1/4 K can be used for system RAM. Such a configuration is useful but not necessary.

External Program Requirements:

1. System Entry Routine —

```
ADRS                INST

000000              061 }
000001              377 }    LXI SP
000002              000 }
000003              303 }
000004              254 }    JMP TBX Entry Point
000005              021 }
```

The stack pointer (SP) must not be in protected memory. If you desire to relocate the SP change the following locations accordingly:

a) 000001 (SP low) and 000002 (SP high)
b) 026301 (SP low) and 026302 (SP high)

2. System Recovery Routine —

```
ADRS                INST

000070              303
000071              000
000072              000
```

3. Input Subroutine: Your input subroutine must begin at 000030. It should carry out the following functions:

a) Move an ASCII character from the input device to register A. The ASCII character should be right justified in A with Parity bit equal to zero. Example: "B" keystroke should set A to 102_8.

b) Test for ESC keystroke (ASCII 177_8) and jump if true to 000000. Suggested instructions

```
     . . .
     376 }    CPI 'ESC'
     177 }
     312 }
     000 }    JPZ System Entry Routine
     000 }
     . . .
```

c) Output an echo check of the inputed character.

d) No registers should be modified except A.

4. Output Subroutine: Your output subroutine should begin at 000050. It should move the ASCII character in register A to the output device. Parity bit is zero. No registers including A should be modified.

5. CR-LF Subroutine: At 000020 you must have a subroutine that will output a CR followed by a LF. Only register A may be modified.

LOADING TBX:

The octal listing of TBX is reproduced later in the text. Addressing is split octal and gives the address of the first byte of each line. An octal loader of some kind is almost a necessity. Loading by front panel switches would be a considerable chore. A Suding-type cassette is available for $5, postpaid, from the authors. Send check or money order to: TBX Tape c/o John Arnold, Route 4, Box 52-A, Tyler TX 75701. If you are interested in a Baudot version of TBX, please inquire at the same address.

Use of a cassette tape to store TBX is virtually a necessity. Every effort has been made to protect TBX against self-destruction byt nothing is 100% sure!

The highest address available in your system for program storage must be loaded as follows:

026115	XXX_8	low part
026116	XXX_8	high part

Example: Suppose you have one 4K board: 026115 377
026116 037

EXECUTING TBX:

Simply examine 000000 and place the computer in the RUN mode. A colon indicates the system is operative.

ERROR MESSAGES

The form of error messages is: ERR α β where α is error number, and β is statement number where error was detected. Label 00000 indicates error occurred in direct execution.

ERROR NUMBER

1 Input line too long--exceeds 72 characters.
2 Numeric overflow on input.
3 Illegal character detected during execution.
4 No ending quotation mark in PR literal.
5 Arithmetic expression too complex.
6 Illegal arithmetic expression.
7 Label does not exist.
8 Division by zero not permitted.
9 Subroutine nesting too deep.
10 RET executed with no prior GOSUB
11 Illegal variable.
12 unrecognizable statement or command.
13 Error in use of parentheses.
14 Memory depletion.

EXAMPLE PROGRAM OF TBX

One example program written in TBX follows. It might assist you in debugging. A TBX line is structured as follows:

1	2	3	4	5	...	n	n+1

Byte No.

1 & 2	Binary value of label; most significant part in 1.
3	Length of text plus 2 in octal.
4 thru n	Text of line.
n + 1	CR (015_8).

After the last line you should find two 377s. At the end of the example run is an octal dump of the program area of memory.

EXAMPLE PROGRAM IN TBX

```
:NEW

:10 IN A
:20 PR"   TEST A IS ";A
:30 PR
:40 GOTO 10
:LST

00010 IN A
00020 PR"   TEST A IS ";A
00030 PR
00040 GOTO 10
:LST 20

00020 PR"   TEST A IS ";A
:LST 20,30

00020 PR"   TEST A IS ";A
00030 PR
:RUN

? 12    TEST A IS  12

? 356    TEST A IS  356

?
:
!DPO:034000 007
034000  000 012 007 040 111 116 040 101
034010  015 000 024 025 040 120 122 042
034020  040 040 124 105 123 124 040 101
034030  040 111 123 040 042 073 101 015
034040  000 036 005 040 120 122 015 000
034050  050 012 040 107 117 124 117 040
034060  061 060 015 377 377 107 022 000
!
```

TINY BASIC EXTENDED

OCTAL LISTING

```
020000  041 111 020 006 110 337 376 015
020010  312 036 020 376 177 312 040 020
020020  376 014 312 067 020 167 043 005
020030  312 306 026 303 005 020 167 311
020040  053 004 076 077 357 303 005 020
020050  332 000 021 076 057 276 322 000
020060  021 303 371 020 000 000 000 327
020070  076 072 357 076 015 062 007 020
020100  303 000 020 000 000 000 000 000
020110  000 114 123 124 040 066 060 060
020120  054 066 062 060 015 015 042 124
020130  105 123 124 061 042 044 120 122
020140  040 042 105 116 104 042 015 106
020150  117 122 040 122 117 127 040 042
020160  073 111 015 015 111 124 040 116
020170  117 122 105 040 114 111 116 105
020200  123 042 015 015 042 015 057 067
020210  062 010 000 000 000 000 000 000
020220  000 032 376 060 330 376 072 320
020230  346 017 311 000 000 000 000 000
020240  000 000 000 000 000 000 000 000
020250  000 000 000 000 000 000 000 000
020260  000 000 000 000 300 021 111 020
020270  325 032 376 040 023 312 271 020
020300  033 041 000 000 376 100 332 320
020310  020 042 350 033 000 321 311 000
020320  315 331 020 042 350 033 067 321
020330  311 315 221 020 376 012 320 023
020340  104 115 051 051 011 051 332 311
020350  026 117 006 000 011 303 331 020
020360  325 052 350 033 104 115 041 111
020370  020 076 071 043 276 303 050 020
021000  345 026 001 076 015 276 312 016
021010  021 024 043 303 005 021 172 062
021020  356 033 321 052 352 033 176 270
021030  312 052 021 322 064 021 043 043
021040  175 206 157 322 026 021 044 303
021050  026 021 043 176 271 312 170 021
021060  332 037 021 053 053 325 353 052
021070  354 033 345 072 356 033 306 003
021100  205 322 105 021 044 157 315 340
021110  030 104 115 341 176 002 053 013
021120  174 272 302 114 021 175 273 302
021130  114 021 023 052 350 033 353 162
021140  043 163 043 072 356 033 074 167
021150  043 321 032 167 376 015 312 166
021160  021 043 023 303 152 021 321 311
021170  053 345 043 043 043 176 376 015

021200  312 207 021 043 303 175 021 043
021210  353 052 354 033 043 104 115 341
021220  032 167 043 023 172 270 302 220
021230  021 173 271 302 220 021 053 042
021240  354 033 072 356 033 376 001 302
021250  361 020 321 311 041 002 032 176
021260  376 200 322 314 021 376 100 322
021270  300 021 043 156 147 303 257 021
021300  346 077 107 043 116 043 345 140
021310  151 303 257 021 376 300 322 000
021320  022 346 077 107 043 116 043 032
021330  023 376 040 312 327 021 033 325
021340  353 032 376 200 322 363 021 276
021350  043 023 312 341 021 321 140 151
021360  303 257 021 346 177 276 302 355
021370  021 353 301 023 043 303 257 021
022000  346 077 043 116 043 345 041 015
022010  022 345 147 151 351 341 322 257
022020  021 043 043 303 257 021 041 357
022030  033 357 043 065 300 066 017 311
022040  000 000 000 000 000 000 000 000
022050  000 000 000 000 000 000 000 000
022060  000 000 000 000 000 000 000 000
022070  000 000 000 000 000 000 000 000
022100  000 345 325 305 353 016 000 041
022110  020 047 315 147 022 041 350 003
022120  315 147 022 041 144 000 315 147
022130  022 041 012 000 315 147 022 173
022140  315 201 022 301 321 341 311 006
022150  377 004 173 225 137 172 234 127
022160  322 151 022 173 205 137 172 214
022170  127 170 271 310 015 315 201 022
022200  311 000 000 000 000 306 060 315
022210  026 022 311 325 052 306 033 053
022220  104 115 052 304 033 353 033 023
022230  327 170 272 302 243 022 171 273
022240  312 275 022 032 147 023 032 157
022250  315 205 026 023 023 032 376 015
022260  312 227 022 305 345 315 026 022
022270  341 301 303 254 022 321 311 000
022300  341 301 345 311 032 023 376 040
022310  312 304 022 033 376 015 310 303
022320  022 030 032 023 376 042 310 376
022330  015 312 317 026 315 026 022 303
022340  322 022 041 360 033 076 040 357
022350  065 302 345 032 066 017 247 311
022360  041 360 033 066 017 000 076 012
022370  357 227 311 000 311 052 350 033

023000  227 274 302 021 023 275 302 021
023010  023 041 004 032 301 343 305 247
023020  311 023 032 147 023 032 157 042
023030  350 033 023 023 301 041 022 032
023040  343 305 247 311 305 104 115 052
023050  361 033 160 043 161 043 042 361
023060  033 301 175 376 177 330 303 322
023070  026 305 052 361 033 053 106 053
023100  042 361 033 146 175 376 100 150
023110  301 320 303 325 026 174 057 147
023120  175 057 157 043 311 315 071 023
023130  174 267 362 147 023 315 115 023
023140  076 055 345 315 026 022 341 315
023150  101 022 247 311 345 052 352 033
023160  104 115 341 012 274 312 174 023
023170  320 303 204 023 003 012 275 312
023200  220 023 320 013 003 003 012 201
023210  117 322 163 023 004 303 163 023
023220  013 140 151 311 315 071 023 315
023230  154 023 353 312 022 023 303 330
023240  026 325 076 077 315 026 022 076
023250  040 357 062 007 020 315 000 020
023260  021 111 020 032 376 055 041 000
023270  000 312 312 023 315 331 020 315
023300  044 023 076 015 062 007 020 321
023310  247 311 023 315 331 020 315 115
023320  023 303 277 023 032 376 040 023
023330  312 324 023 033 306 300 320 007
023340  157 046 024 315 044 023 067 023
023350  311 032 376 040 023 312 351 023
023360  033 376 100 322 310 023 376 050
023370  310 041 000 000 303 124 024 000
024000  000 023 055 050 007 056 073 025
024010  000 001 002 000 001 000 001 000
024020  002 000 001 000 013 000 010 000
024030  000 000 000 000 070 000 025 000
024040  000 000 000 000 000 000 002 000
024050  324 046 004 000 002 000 001 000
024060  000 000 003 000 126 053 000 023
024070  016 000 004 000 000 023 000 023
024100  032 023 376 040 312 100 024 033
024110  376 015 310 376 044 310 303 314
024120  026 023 076 001 315 331 020 315
024130  044 023 311 315 071 023 106 043
024140  146 150 315 044 023 247 311 315
024150  071 023 114 105 315 071 023 160
024160  043 161 247 311 035 372 034 125
024170  023 321 076 001 311 023 000 023

024200  315 071 023 104 115 315 071 023
024210  011 315 044 023 247 311 315 071
024220  023 315 115 023 104 115 315 071
024230  023 011 315 044 023 247 311 000
024240  325 006 000 315 071 023 174 267
024250  374 301 024 353 315 071 023 174
024260  267 374 301 024 315 306 024 005
024270  314 115 023 315 044 023 321 247
024300  311 004 315 115 023 311 305 104
024310  115 041 000 000 076 021 062 363
024320  033 170 037 107 171 037 117 322
024330  333 024 031 174 037 147 175 037
024340  157 072 363 033 075 312 356 024
024350  062 363 033 303 321 024 140 151
024360  301 311 325 006 000 315 071 023
024370  174 267 374 301 024 353 315 071
025000  023 174 267 374 301 024 353 227
025010  274 302 020 025 275 312 333 026
025020  315 026 025 303 267 024 305 006
025030  001 174 346 100 302 044 025 051
025040  004 303 031 025 170 062 363 033
025050  104 115 041 000 000 173 221 137
025060  172 230 127 322 117 025 173 201
025070  137 172 210 127 051 072 363 033
025100  075 312 115 025 062 363 033 353
025110  051 353 303 055 025 301 311 051
025120  043 072 363 033 075 312 115 025
025130  303 104 025 315 071 023 315 115
025140  023 315 044 023 247 311 000 000
025150  000 325 315 071 023 353 315 071
025160  023 345 315 071 023 174 346 200
025170  302 262 025 172 346 200 302 227
025200  025 174 272 312 214 025 322 227
025210  025 303 224 025 175 273 312 232
025220  025 322 227 025 076 001 041 076
025230  004 041 076 000 341 021 242 025
025240  325 351 312 260 025 321 032 376
025250  015 312 375 022 023 303 246 025
025260  321 311 172 346 200 302 201 025
025270  303 224 025 376 000 311 376 001
025300  311 376 000 310 376 001 311 376
025310  001 310 376 004 311 376 004 311
025320  376 000 310 376 004 311 056 273
025330  001 056 276 001 056 301 001 056
025340  307 001 056 315 001 056 320 046
025350  025 315 044 023 247 311 305 104
025360  115 052 364 033 160 043 161 043
025370  042 364 033 301 175 376 177 330
```

```
026000  303 336 026 305 052 364 033 053
026010  106 053 042 364 033 146 175 376
026020  164 150 301 320 303 341 026 142
026030  153 315 356 025 247 311 315 003
026040  026 353 247 311 076 040 315 026
026050  022 247 311 000 000 000 041 077
026060  026 001 350 033 176 002 175 376
026070  033 310 003 043 303 064 026 000
026100  000 000 034 001 034 000 040 017
026110  100 030 000 164 024 377 057 000
026120  000 056 241 051 321 377 057 377
026130  377 041 100 030 042 361 033 041
026140  164 024 042 364 033 315 020 027
026150  052 352 033 126 043 136 353 000
026160  042 350 033 023 023 247 311 076
026170  015 357 303 360 022 327 076 017
026200  062 360 033 247 311 345 325 305
026210  353 016 377 303 107 022 000 000
026220  327 000 000 000 076 105 357 076
026230  122 357 357 076 040 357 046 000
026240  000 000 000 315 101 022 052 350
026250  033 076 040 357 315 205 026 016
026260  010 041 357 033 021 106 026 032
026270  167 015 302 267 026 041 002 032
026300  061 377 000 303 257 021 056 001
026310  001 056 002 001 056 003 001 056
026320  004 001 056 005 001 056 006 001
026330  056 007 001 056 010 001 056 011
026340  001 056 012 001 056 013 001 056
026350  014 001 056 015 001 056 016 001
026360  056 017 001 056 020 303 216 026
026370  000 000 000 000 000 000 000 000
027000  000 000 000 000 000 000 000 000
027010  000 000 000 000 000 000 000 000
027020  076 012 357 052 115 026 042 366
027030  033 311 325 315 071 023 353 315
027040  071 023 104 115 315 044 023 353
027050  315 044 023 321 305 315 240 024
027060  315 071 023 303 072 027 315 071
027070  023 345 051 104 115 052 366 033
027100  175 221 117 174 230 107 013 052
027110  354 033 274 302 120 027 171 275
027120  332 360 026 140 151 301 160 053
027130  161 104 115 042 366 033 315 071
027140  023 161 043 160 247 311 315 071
027150  023 053 051 104 115 315 071 023
027160  011 315 044 023 247 311 315 071
027170  023 053 315 044 023 052 370 033
```

```
027200  315 044 023 315 240 024 315 200
027210  024 303 146 027 032 023 376 040
027220  312 214 027 033 306 300 320 007
027230  117 023 032 376 050 312 243 027
027240  033 247 311 151 046 024 116 043
027250  146 151 116 043 106 043 315 044
027260  023 140 151 042 370 033 067 311
027270  300 325 342 000 000 000 000 000
027300  000 000 000 000 000 325 023 032
027310  376 015 302 064 030 353 315 044
027320  023 321 247 311 325 315 071 023
027330  345 116 043 106 315 071 023 353
027340  315 071 023 003 172 270 302 361
027350  027 173 271 322 361 027 303 006
027360  030 345 315 044 023 341 353 315
027370  044 023 341 315 044 023 140 151
030000  315 044 023 341 247 311 341 315
030010  044 023 140 151 315 044 023 321
030020  247 311 376 044 302 314 026 303
030030  033 023 032 376 040 023 312 032
030040  030 033 306 300 320 325 023 032
030050  306 300 321 320 376 015 310 376
030060  040 310 067 311 376 044 312 315
030070  027 303 306 027 347 323 241 324
030100  000 120 000 036 000 006 000 007
030110  000 010 000 012 000 000 000 030
030120  000 000 000 030 326 036 322 375
030130  230 141 001 014 211 326 167 323
030140  011 230 155 022 006 005 220 322
030150  304 322 213 322 375 230 166 012
030160  007 214 322 304 027 320 230 073
030170  014 001 223 322 304 027 300 000
030200  204 232 146 015 041 375 033 006
030210  010 176 007 007 007 256 027 027
030220  055 055 055 176 027 167 054 176
030230  027 167 054 176 027 167 054 176
030240  027 167 005 302 211 030 052 374
030250  033 174 346 077 147 376 047 312
030260  272 030 322 204 030 315 044 023
030270  077 311 175 376 020 303 262 030
030300  315 071 023 105 076 040 315 026
030310  022 005 302 304 030 063 063 063
030320  063 063 063 301 341 043 043 345
030330  305 073 073 073 073 073 073 311
030340  072 367 033 274 312 360 030 332
030350  360 026 042 354 033 311 000 000
030360  072 366 033 326 000 275 322 352
030370  030 303 360 026 000 000 000 000
```

```
031000  052 354 033 053 104 115 052 376
031010  033 011 345 052 366 033 104 115
031020  052 352 033 011 301 315 060 031
031030  315 101 022 076 040 357 052 366
031040  033 104 115 052 354 033 053 315
031050  060 031 315 101 022 327 247 311
031060  171 225 157 170 234 147 311 052
031070  352 033 042 304 033 052 354 033
031100  042 306 033 247 311 315 165 031
031110  042 304 033 043 043 076 015 043
031120  276 302 117 031 043 043 042 306
031130  033 247 311 000 315 165 031 043
031140  043 076 015 043 276 302 143 031
031150  043 043 042 306 033 315 165 031
031160  042 304 033 247 311 315 071 023
031170  315 154 023 310 303 330 026 000
031200  000 000 000 000 000 000 000 000
031210  000 000 000 000 000 000 000 000
031220  000 000 000 000 000 000 000 000
031230  000 000 000 000 000 000 000 000
031240  000 000 000 000 000 000 000 000
031250  000 000 000 000 000 000 000 000
031260  000 000 000 000 000 000 000 000
031270  000 000 000 000 000 000 000 000
031300  231 310 122 316 330 204 322 300
031310  232 330 124 302 132 343 330 300
031320  322 300 231 331 215 326 175 322
031330  375 232 210 244 326 175 323 034
031340  231 351 215 331 067 322 213 322
031350  375 132 343 231 366 254 132 343
031360  331 134 322 213 032 216 331 105
031370  322 213 032 216 047 041 066 010
032000  326 053 326 167 320 070 322 360
032010  320 265 032 022 320 360 032 004
032020  326 131 232 041 114 105 324 133
032030  310 132 340 324 147 322 304 322
032040  375 232 074 107 317 232 057 124
032050  317 132 343 322 304 323 224 232
032060  275 123 125 302 132 343 324 100
032070  326 027 323 224 232 112 111 306
032100  132 343 133 114 132 343 325 151
032110  032 022 233 326 106 117 322 323
032120  324 326 363 132 340 324 147 226
032130  363 124 317 327 305 132 343 322
032140  304 322 375 075 046 062 004 032
032150  232 226 120 322 231 322 242 322
032160  322 232 173 254 322 342 232 332
032170  215 322 375 232 202 273 326 044
```

```
032200  032 166 326 175 322 304 322 375
032210  132 343 323 125 032 161 322 304
032220  322 375 000 000 000 000 232 251
032230  111 315 133 310 323 241 324 147
032240  232 245 254 032 232 322 304 322
032250  375 232 264 122 105 324 326 036
032260  322 304 322 375 233 200 105 116
032270  304 326 167 323 011 232 306 114
032300  123 324 031 340 322 375 232 317
032310  122 125 315 322 304 032 020 233
032320  101 116 105 327 322 304 032 000
032330  326 347 232 154 244 323 034 000
032340  232 343 275 232 354 255 133 003
032350  325 133 032 361 232 357 253 133
032360  003 232 372 253 133 003 324 200
032370  032 361 233 055 255 133 003 324
033000  216 032 361 133 027 233 016 252
033010  133 027 324 240 033 005 233 055
033020  257 133 027 324 362 033 005 330
033030  032 033 035 031 300 327 214 033
033040  047 133 254 324 133 322 300 323
033050  324 033 057 324 133 322 300 323
033060  351 033 065 322 300 233 077 250
033070  132 343 233 077 251 322 300 326
033100  352 232 330 123 132 305 331 000
033110  032 216 000 000 233 123 275 325
033120  326 322 300 233 150 274 233 135
033130  275 325 334 322 300 233 144 276
033140  325 337 322 300 325 331 322 300
033150  232 330 276 233 162 275 325 345
033160  322 300 233 171 274 325 337 322
033170  300 325 342 322 300 000 000 000
033200  232 275 104 111 315 323 324 326
033210  352 226 355 250 132 343 233 241
033220  254 132 343 226 355 251 327 032
033230  233 235 254 033 205 322 304 322
033240  375 226 355 251 327 066 033 230
033250  000 000 000 000 226 355 250 132
033260  343 233 275 254 132 343 226 355
033270  251 327 166 322 300 226 355 251
033300  327 146 322 300 044 034 054 034
033310  327 214 033 320 133 254 322 300
033320  323 324 326 344 322 300 232 150
033330  116 130 324 323 324 326 352 327
033340  324 324 147 322 304 322 375 072
033350  000 000 000 034 054 034 004 040
033360  017 100 030 000 164 024 377 057
033370  000 000 056 241 051 321 377 057
```

the digital group

po box 6528, denver, colorado 80206

December 14, 1975

Mr. Bob Albrecht & Bernard Greening
People's Computer Company
PO Box 310
Menlo Park, CA 94025

Dear Bob and Bernard,

I am very interested in helping out with your Tiny BASIC (perhaps Micro BASIC might be more appropriate). Since my specialty is Hardware and the lowest level Software to interface this hardware to a system, I would like to suggest a simple hardware subsystem.

A scientific calculator IC can be easily interfaced to a microprocessor to provide all of the various mathematical operations very accurately with minimal software overhead. I am including a copy of some of the scientific calculator documentation out by the Digital Group.

This scientific calculator has been interfaced to an 8008 (Mark-8 modified) and MOS Technology 6501/2 system. The software can be easily modified to support an 8080 or 6800, thereby providing an easy access to building "Tiny BASIC" for 8008, 8080, 6800, 6501 or 6502 systems.

The major drawback of a calculator chip for math routines is that it is very slow compared to specialized hardware and software systems. The major advantages are:

1. Low software overhead (about 300 bytes for interfacing)
2. Low cost (around $45 worth of parts & PC board)
3. Quick way to develop Math routines with high accuracy.

I would be happy to assist PCC in developing Tiny BASIC using these Scientific Calculator IC's.

Dr. Robert Suding

c/o The Digital Group

SCIENTIFIC CALCULATOR

Here is a calculator circuit designed to be used with any computer of 8 bits or more capacity. I am presently using it with an 8008 system, approximately 300 bytes of storage being required to basically interface this circuit to my TV readout and keyboard. Only one 8-bit input port and one 8-bit output port is required.

The heart of the circuit is the 2529-103 calculator IC from Mos Technology. This is a simple IC which gives trig, log, memory, square root, etc., functions. The display is normally a 12-digit LED 7-segment assembly. The segment drivers are built into the 2529. The 12-digit outputs are usually fed to a pair of 75492's which serially scan each of the 12 digits at about a 60Hz cycle rate from an internal clock. A matrixed keyboard is normally attached between the 12 digit outputs of the 2529 and 4 keyboard inputs of the 2529, giving a potential 48-key input capability, 41 of which are actually used.

The design required efficient handling of the 12-digit outputs. Since it was necessary to utilize the digit outputs for both data entry and digit segment output, the design was centered on a controlled accessing of the asyncronously scanning 12 digits. The computer has 4 bits of an output port assigned to the duty of selecting a given digit by sending its binary equivalent to the inputs of a 74150 sixteen input selector. When the selected digit becomes present the output at pin 10 of the 74150 goes low as long as the digit is present. By combining this input with three more bits from the computer, the desired "keyboard" input is sent to the 2529. The computer word should be held for at least 40 ms to be certain that the asyncronously scanning digit has been accessed.

Likewise, the digit output must identify the digit to which the current segment outputs apply. By using the same coding scheme for the four inputs to the 74150, a computer controlled sampling system is established. The MSB output from the computer informs the calculator interface that a digit/segment output is desired. When the desired digit finally ripples by, a strobed MSB+ pulse appears on the interface output. This pulse then interrupts the computer to inform it that the segment data for the desired digit is present and valid as long as the MSB stays +.

Several considerations: First, only 5 of the 7 segments are needed to decode 0 through 9, minus, blank, and the error signs. Each digit may also have a decimal point attached to it, so the output becomes 6 bits, plus the MSB strobe bit. Be aware that these calculator chips are quite slow. When entering a data item or especially a function, the display will go blank up to 1/3 second while internal processing takes place. The result can take on any number of digits, but digit 9 is always used. By sampling for "digit 9 not blank," the end of internal processing can be detected. When this occurs, either further entries, or sampling of all 12 digits may proceed without data loss. 8008 programs have been written to handle simple keyboard entry and tv result display, and interactive calculation operations involving messages anf formula building and reiteration. These are available through the Digital Group.

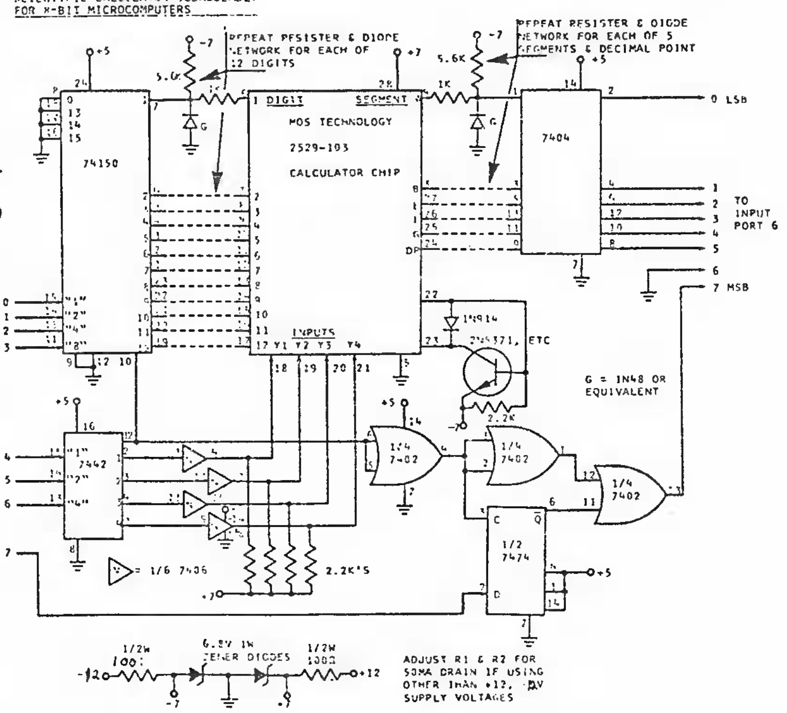
The 2529 is available from Mos Technology at $27.50 apiece. Some newer scientific calculator chips have been announced by Mos Technology and are being presently sampled.

Other calculator chips could be used in similar circuits. However, I would question the advisability of using these simpler chips with their much lower calculation power return. Mos Technology also makes an RPN format calculator IC, the 2529-106 for H.P. buffs. A metric conversion chip (2529-104) is also available from Mos Technology. These IC's have been tried in the circuit. They are directly usable in the enclosed circuit.

The basic functions are roughly equivalent to the TI SR-50, but the enhanced software version will be considerably better than the HP-65 programmable calculator due to its message display capacity and "almost" unlimited memory capacity.

–Dr Robert Suding WOLMD

SCIENTIFIC CALCULATOR SUBASSEMBLY FOR X-BIT MICROCOMPUTERS



INPUT CODES FOR FUNCTION ENTRY

FUNCTION	OCTAL	HEX	FUNCTION	OCTAL	HEX
0	021	11	ARC	033	1B
1	022	12	SIN	061	31
2	023	13	COS	062	32
3	024	14	TAN	063	33
4	025	15	LN	064	34
5	026	16	LOG	065	35
6	027	17	RCL	067	37
7	030	18	Σ	070	38
8	031	19	x↔y	071	39
9	032	1A	DGR	072	3A
.	041	21	STO	073	3B
+	042	22	CA/CE	074	3C
-	043	23	CHS	053	2B
x	044	24	EEX	054	2C
÷	045	25	y^x	046	26
=	047	27	(	050	28
π	052	2A	)	051	29
√	066	36	10^x	103	43
1/x	101	41	e^x	104	44
x^2	102	42	N!	105	45
No Op	000	00	Restore-Display	034	1C

INPUT CODES FOR DIGIT DATA REQUEST

DIGIT	OCTAL	HEX	DIGIT	OCTAL	HEX
1	201	81	7	207	87
2	202	82	8	210	88
3	203	83	9	211	89
4	204	84	10	212	8A
5	205	85	11	213	8B
6	206	86	12	214	8C

OUTPUT CODES FOR SEGMENT DECODE

DIGIT ALONE	OCTAL	HEX	DIGIT AND DECIMAL PT	OCTAL	HEX
0	260	B0	0.	220	90
1	275	BD	1.	235	9D
2	250	A8	2.	210	88
3	254	AC	3.	214	8C
4	245	A5	4.	205	85
5	246	A6	5.	206	86
6	243	A3	6.	203	83
7	274	8C	7.	234	9C
8	240	A0	8.	200	80
9	244	A4	9.	204	84
-	257	AF	-.	217	8F
ERROR*	262 or 242	B2 or A2	ERROR.*	222 or 202	92 or 82
Blank	277	BF	Blank.	237	9F

*(leftmost digit only)

- INPUT CODES SENT TO ENTER DIGITS AND FUNCTIONS. CODE MUST BE HELD MORE THAN 40MS.
- SEND DIGIT 9 DATA REQUEST (211 OR 89) AND WAIT FOR MSB FROM 7402 TO GO +. THIS INDICATES INTERNAL CALCULATIONS FINISHED.
- SEND DIGITS 12 THROUGH 1 DATA REQUESTS, DECODE EACH WITH SEGMENT DECODE DATA TABLE AS DATA AVAILABLE MSB LINE FROM 7402 GOES + FOR EACH DIGIT.
- LONGEST CALCULATION DELAY APPEARS TO BE 69! (ABOUT 1/3 SEC).
- RPN (2529-106) AND METRIC-CONVERSION (2529-104) CALCULATOR IC'S FROM MOS TECHNOLOGY WILL WORK WITH THE SAME CIRCUIT.

DR. ROBERT SUDING WØLMD

SNOBOL FOR THE ALTAIR

Dear Dragons,

Thanks for the great publication and other nice things--like dragon shirts!. What a way to learn.

I have a problem. Without considering any possible consequences, I have committed myself to writing a SNOBOL Compiler (interpreter?) for an Altair 8800. My officemate has built the Altair for the college at which he teaches, and after many months of promising some kind of assistance, I finally offered to write a compiler.

To get to the point: does anyone out there have any experience in compiler writing, particularly in SNOBOL compiler writing? I know that some of the sharpest people in this field read *PCC*, so I'm really hoping to hear from someone.

Of course, once I get the compiler working, I will make it available to other Altair owners and users (for a nominal fee and a lot of glory).

(I realize all you people are heavily into BASIC, but SNOBOL is a pretty neat language for things like compiler writing, natural language translation, and general string manipulation.)

Also, since my friend's Altair is 75 miles away from my home, donations of Altairs will be accepted.

MAUREEN SUPPLE
828 S. Irving St
Arlington VA 22204

(SNOBOL compilers are tough. An interpreter would be easier. A good place to start looking for information would be Griswold's book, *The Macro Implementation of SNOBOL*, W.H. Freeman, San Francisco, 1973; and Waite's book, *Implementing Software for Non-Numeric Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1971.)

FULL OF HOLES

I guess you know, Tiny BASIC as presented in its first chapter is full of holes. Look, for example, at what happens if you try to evaluate an expression without aunary plus or minus on the front. Ich. Also, I wonder if the interpreted interpreting interpreter interpreter executor is viable for a really small, slow system like an 8008 system. Talk about crunching! Anyway, I want to see more. I'm crazy, maybe? Who cares.

Sincerely,

FRITZ ROTH
Rt 7
Carbondale IL 62901

A HIGH ORDER

Dear Bob Albrecht, I am writing this letter about many things I've read about in *PCC*. The Tiny BASIC project looks like something *everyone* would like to tackle. The interpreter idea is a little costly on time and storage, unless you plan to use it on many systems. Otherwise, it's a good idea. I'm interested in simulating languages using BASIC or FORTRAN as the "machine," so this type of thing is interesting. If only someone had the plans for ALGOL in IL . . .

If anyone has done any projects simulating languages/computers in a high order language, would they please contact me?!

Thanks for everything, *PCC*!

Respectfully,

REED CHRISTIANSEN
2756 Fernwood No.
Roseville MN 55113

TB CODE SHEET

by Dick Whipple

You may be interested in knowing that John Arnold and I write our programs (like TB) in machine language. We have found it to be less restrictive and more versatile although not having a source file of some kind is a disadvantage. We do keep a hand-generated source listing on coding sheets for our reference. A major program like TB requires a two-pass development: the first pass ends up with lots of "fixes" and "patches" to get the program to work; the second pass is then used to clean-up the mess produced in pass one. The coding sheets from pass two represent the nearest thing to source code we have. For your reference I have included a copy of one of our coding sheets from TB. The addresses are split octal.

```
Program Name: Tiny BASIC (Rev 1)      PCC Dragon Copy
By RBW   Date 11/10/75   H = 020             Page 1 of 27

BUFFIN:
L=  000 | 040  LXI H BUFSTRT
     01 | 111
     02 | 020
     03 | 001  LXI  B: LNWDTH
     04 | 040       C: CASE
     05 | 110
CONT4: 006 | 337  RST IN
CONT3: 007 | 376  CPI <FGS>
     10 | 033
     11 | 312  JPZ FGS
     12 | 052
     13 | 020
     14 | 376  CPI <LTR>
     15 | 037
     16 | 312  JPZ LTR
     17 | 057
     20 | 020
     21 | 005  DCR B
     22 | 312  JPZ ERR306
     23 | 306
     24 | 026
     25 | 376  CPI <CR>
     26 | 010
     27 | 312  JPZ END
     30 | 107
     31 | 020
     32 | 376  CPI <SP>
     33 | 004
     34 | 312  JPZ SP
     35 | 045
     36 | 020
     37 | 201  ADR C

L = 040 | 376  CPI <?>+040_8
     41 | 071
     42 | 312  JPZ RUBOUT
     43 | 101
     44 | 020
SP:  045 | 167  A→M
     46 | 043  INX H
     47 | 303  JMP CONT4
     50 | 006
     51 | 020
FGS: 052 | 016  MVI C 040_8
     53 | 040
     54 | 303  JMP CONT4
     55 | 006
     56 | 020
LTR: 057 | 016  MVI C Ø_8
     60 | 000
     61 | 337  RST IN
     62 | 376  CPI <LTR>
     63 | 037
     64 | 302  JNZ CONT3
     65 | 007
     66 | 020
     67 | 327  RST CRLF
GETLINE: 070 | 076  MVI A <FGS>
     71 | 033
     72 | 357  RST OUT
     73 | 076  MVI A <:>
     74 | 016
     75 | 357  RST OUT
     76 | 303  JMP BUFFIN
     77 | 000
            cont.
```

Are you implementing Tiny BASIC or some other software. Let us know and we'll let others know. Let's stand on each others shoulders and not on each others toes (to paraphrase C. Strachey).