dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia*

Running Light Without Overbyte

$1.50

February 1976 Box 310, Menlo Park CA 94025 Volume 1, Number 2

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

IN THIS ISSUE . . . page

In Future Issues . . .

† DOCUMENTED SOURCE CODE FOR THE DENVER VERSION OF TINY BASIC
† A PUBLIC-DOMAIN FLOPPY DISC FILE SYSTEM
† SCHEMATICS & ARTICLES, REPRINTED FROM MANY COMPUTER CLUB NEWSLETTERS
† DIRECTORIES OF CLUBS & ORGANIZATIONS, COMPUTER HOBBYIST STORES & DISTRIBUTORS, PUBLICATIONS, ETC.
† LISTS OF COMPUTER HOBBYISTS & THEIR EQUIPMENT
† INDICES TO COMPUTER HOBBYIST ARTICLES IN OTHER PUBLICATIONS

*previously DR DOBB'S JOURNAL OF TINY BASIC CALISTHENICS & ORTHODONTIA

DR DOBB'S JOURNAL OF
COMPUTER CALISTHENICS & ORTHODONTIA
Volume 1, Number 2; February 1976
Box 310, Menlo Park CA 94025
[1010 Doyle, Menlo Park; (415) 323-6117]

POSTMASTER: *Please send Form 3579 to: Box 310, Menlo Park CA 94025. Return postage guaranteed. Application to mail at second-class postage rates is pending at Menlo Park CA. Published 10 times per year; monthly, excluding July and December.*

U.S. subscriptions:
(Subscription blank is inside of back cover)
$1.50 for a single issue.
$3 for the first three issues.
$10 per year.

Foreign subscriptions:
Add $4 per year to the above rates for surface mail.
Add $12 per year to the above rates for air mail.

Discounts available for bulk orders.

Disclaimer

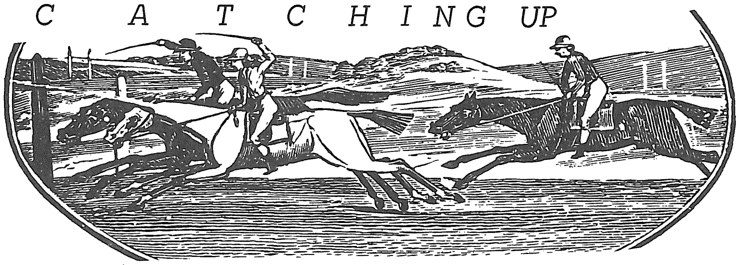
We serve as a communication medium for the exchange of information. We do not guarantee the validity of that information.

COMING in the next issue...

† Documentation & complete source code for a Denver version of Tiny BASIC
† Touchless sensing for under $100--proximity sensors that can "see" liquids and solids
† Quik Bits--short news articles concerning home computing
† Keyboard Loader for Octal Code via the TVT-2
† Details of a Software Contest for the TV Dazzler
† A center for software reproduction and distribution
† More details on the Votrax speech synthesizer kits
† Articles from the computer club newsletters
† Lots o' Letters, & A pointer to a 16K BASIC for the 8008

and much more . . .

W E ' R E C A T C H I N G UP



This started out to be a one-shot, three-issue quickie on Tiny BASIC. It was being put together on a sorta spare-time basis by the PCC mob. Once we became aware of the information gap that we are now focusing on filling, it took a coupla weeks or more to gather together a staff and organize a full-scale magazine production effort. Thus:

The first issue, "January, 1976," didn't get out until the end of February.

This second issue is being mailed April 12th, in spite of it's being dated "February."

Number 3 will go out less than two weeks thereafter, however, and the "April" issue should go out in the first week of May.

Finally . . . the May issue will go out about the third week of May, and (whew!) we'll be on schedule from there on.

What's DDJCC&O all about?

My gawd! Not *another* computer hobbyist magazine! That was my first reaction when People's Computer Company approached me about becoming Editor of their one-issue-old infant, **DR DOBB'S JOURNAL OF TINY BASIC CALISTHENICS & ORTHODONTIA.** PCC had originally planned on publishing three issues of the **JOURNAL**. The response to the first, patchquilt issue, however, convinced them (and me) that an area of badly-needed information is not being covered by the presently existing publications. Furthermore, it seems unlikely that the other publishers will choose to cover that area; they have their hands (and pages) full just covering hardware and small bits of software.

What is this area; this information vacuum? It's *free and very inexpensive software*. One of the primary thrusts of **DR DOBB'S JOURNAL** will be to present detailed information concerning low-cost systems software; interpreters, compilers, structured assemblers, graphics languages, floppy disc file systems, etc. This will include user documentation and examples, documentation on implementation including complete source code listings, updates giving errors and their fixes, explicit and detailed notes on the design and implementation of such systems software, and so on. This **JOURNAL** is explicitly available to serve as a communication medium concerning the design, development, and distribution of free and low-cost software for the home computer.

We encourage you to send in documented software, as you develop it. We hope that you will use the software that we publish in this **JOURNAL**; that you will study it and modify it to expand its capabilities, and that you will report any bugs you may note to us and to the authors.

We are also quite interested in publishing evaluations of *any* software and hardware that is being sold to the home computer user. We are supported by readers' subscriptions rather than advertising. We will not hesitate to publish positive *and* negative evaluations. We adamantly hold the position that, if a manufacturer of some hardware or software is going to peddle it to unsuspecting consumers for a healthy profit, their product damn well ought to perform as well as their advertisements and profit imply it will!

There are some other areas of information that we expect to cover, not seen in most of the other major computer hobbyist publications. These include complete indices to *all* of those publications, directories of computer stores and distributors, listings of computer clubs and organizations, listings of users and their equipment, etc. Another tidbit: as long as we can afford to, we will carry classified ads.

We also plan to begin reprinting articles and schematics from the club newsletters. We have heard the comment, over and over, "I wish I could see the stuff that's being printed by all the homebrew groups, but I just can't afford to subscribe to all of them." We expect to help with this desire.

Finally, we will be doing some fairly detailed "blue skying." Everyone is wondering where home computers are going, and what the potentials are. We have a number of ideas (with more rolling in, every day) about what can be done in the immediately foreseeable future. We will be presenting them and encouraging their realization. The Votrax articles on page 32 of this issue are one small example of this.

Thank you for reading. We want your suggestions. We want your contributions of software, hardware designs, evaluations, and anything else you're willing to share with other home computer enthusiasts. And, of course, we want your subscriptions. The more subscriptions we have; the more pages we can print; the more information we can pass along to you and your friends. If you like what you see here, we hope you will spread the word.

Nuf sed, for now. More in a coupla weeks. **--Jim C. Warren, Jr., Editor**

A CRITICAL LOOK AT BASIC

Dennis Allison, 169 Spruce Ave, Menlo Park CA 94025
Consultant (415) 325-2962

[This article appeared in *Timesharing: Past, Present, Future. Proceedings of the Second Annual Computer Communications Conference, San Jose, January 1973.*]

0. INTRODUCTION

BASIC is the dominate interactive programming language. It has been widely implemented since its introduction in 1965 as a component of the Dartmouth timesharing system. BASIC is presently widely used as an instructional language at both the high school and college level. Standardization efforts are now in progress, but are hampered by the proliferation of dialects and incompatible extensions.

The purpose of this paper is to evaluate the BASIC language as a problem solving tool. BASIC is not the language of choice for problem solving given our present understanding of the programming process. That is not to say that programs, even good programs, cannot be written in BASIC. There is overwhelming evidence which indicates they can be. Rather, it says that the language structure makes it difficult to write a clear, concise, well structured program.

The emerging discipline of software engineering has provided us with a pair of complementary methodologies which, when properly applied, help minimize the difficulty of developing error-free software systems both large and small.

One might say that BASIC is too simple, too easy to use. It is possible for a novice user to learn to program in a single day. It is also almost axiomatic that large programs written by BASIC programmers will be ridden with bugs. The language lacks the mechanisms to structure the problem's algorithm and data well. It breeds bad habits, habits which are difficult to unlearn.

1. MAKING A PROGRAM

The program development cycle can be summarized as repetative application of the following:

- o Problem definition
- o Algorithm development
- o Program entry (error prone, mechanical)
- o Testing to discover errors
- o Debugging (localizing errors)
- o Editing (mechanical)

Contemporary timesharing systems support the mechanical portion of program development and neglect the conceptual and definitional part. BASIC systems provide for program entry, syntax checking, editing, and the like, but don't really provide much help when it comes to deciding how to solve the problem at hand. The program is expected to blossom forth in full bloom from the gestalt mind of the user. A corollary to the above observation: BASIC programs written at the console usually look it.

While little support is given to testing and proofs of correctness, debugging is well supported within BASIC. BASIC is usually interpreted, so the state of the BASIC machine is available to construct diagnostics. Simple errors, array-bounds violations are checked and diagnostics reported at run-time. Many BASIC systems defer reporting structual errors (for example, a missing NEXT) to run-time as well, a practice not to be commended. The ease of finding errors in BASIC programs allows one to build programs on a pragmatic, experimental basis. That leads to a false sense of security. One had best remember Dijkstra's dictum: "Program testing can be used to show the presence of bugs, never to show their absence."

2. MODULAR PROGRAMMING

If any rule of the thumb as to how to construct good programs exists, it is: Divided and Conquer. Problems are best solved by decomposing them into smaller and smaller problems until the resultant problem can be solved in a simple, direct manner.

Dijkstra has pointed out that the process of dividing a problem into its natural fragments results in the introduction of levels of abstraction. At each level of abstraction primative functions are defined which manipulate primative data aggregates; the operations and the data structures mirror an abstract model of the problem being solved. At lower levels of abstraction these primative operations and data structures are themselves decomposed into still more primative units. For example, a sort-merge program may deal on one level with manipulations of files, and on another level with records and keys.

BASIC provides few mechanisms for modularity. There is a one-line arithmetic function capability and an unparametered subroutine (GOSUB/RETURN) facility. The first has limited usefulness; it provides a convenient shorthand for computation, nothing more. The subroutine facility is very primative. It requires the user to develop conventions for passing parameters and for the naming of local data. All variables are global in BASIC and are shared between all modules of the program. Subroutines are not distinct from the corpus of the main program (and other subroutines); transfers into and out of subroutines is unrestricted and often unmanageable. Subroutine reference is by line number rather than a mnemonic name, a convention which tends to obscure the functional purpose of the subroutine.

3. STRUCTURED PROGRAMMING

Structured programming is a technique which limits the control structures which interconnect the modules of a program to a few well-defined forms. Modules include procedures and collections of statements, either of which may be nested. The flow of control utilizes conditional and plex selection to select paths and provide for repetition unconditionally, under control of a boolean expression, or under control of an indexing variable. Recent systems provide escape statements which allow control to exit several nested modules. All systems forbid the use of unconditional branching.

The rationale behind structured programmin is the minimization of the connectivity within a program. Programs which have a well defined, nested structure tend to be clear. The logical flow is usually sufficiently clear that a flowchart is not necessary to tour all paths in the program. In addition, the conditions under which any given path is to be executed are clearly spelled out.

BASIC is the antithesis of a language for writing structured programs. The GOTO and the IF...THEN both result in unstructured transfers of control. No means is provided for collecting statements into a group to be executed as a module. And instead of eliminating labels, BASIC requires all statements to have one.

True structured programming is difficult in BASIC. It requires unmitigated attention and discipline to maintain a structured programming style. And the clarity which one normally acquires is lost because module boundaries are not distinct.

4. SUMMARY

BASIC does not provide those features which appear desireable in an interactive programming language to be used for real-world problem solving. The flaws are not superficial; they are buried deeply in the structure of the language. In particular, BASIC is not a vehicle for the best techniques known for the construction of programs: modular programming and structured programming.

Making programs is not an easy task. A problem of even moderate complexity often cannot be comprehended in the whole. We must abstract and localize the processing to make it tractable. Our contention is that BASIC does not help this process and, because of its structure, often hinders it. The time is ripe to find a better language, one more closely related to our needs.

5. REFERENCES

BASIC

Anon. (1970) BASIC, Fifth Edition, Dartmouth College, Hanover, New Hampshire.

Lee, J.A.N., The Formal Definition of the BASIC Language, The Computer Journal, Volume 15 Number 1 (February 1972), pages 37-41.

Ogdin, J., The Case Against BASIC, Datamation (September 1, 1971), pages 34-41.

Sammet, Jean E., Programming Languages, Prentice--Hall (1969).

Modular and Structured Programming

Baker, F. T., System Quality Through Structured Programming, Proc. FJCC (1972), pages 339-343.

Buxton, J. N. and B. Randell (eds.), Software Engineering Techniques, Report on a Conference Sponsored by NATO Science Committee, Rome Italy (1969).

Dijkstra, E. W., Notes on Structured Programming, Report 241, Technische Hogeschool Eindhoven (1969).

Levenworth, B. M., Programming With(out) the GOTO, Proc. ACM Nat. Conf. (1972), pages 782-786.

Liskov, B. H., A Design Methodology for Reliable Software Systems, Proc. FJCC (1972), pages 191-199.

Mills, H. D., Mathematical Foundations for Structured Programming, IBM FSD Report FSC72-6012, Gaithersburg Maryland (February 1972).

Naur, P. and B. Randell (eds.), Software Engineering, Report on a Conference Sponsored by the NATO Science Committee, Garmisch Germany (1968).

Wirth, N., Program Development by Stepwise Refinement, CACM 14 (April 1971).

Wulf, W. A., A Case Against the GOTO, Proc. ACM Nat. Conf. (1972), pages 791-797.

COMMERCIAL GOODIES OF INTEREST

PC BOARDS

For those inclined to design their own microcomputer system, Schweber Electronics (Westbury NY 11590, 516-334-7474), is marketing several PC boards that appear interesting. Their Microcomputer Panel No. 9045-3BD-60 purports to accommodate nearly all currently available microprocessor kits. Their Memory Panel No. 9042-3BD-60 accepts any 16-, 18-, 22-, or 28-pin LSI RAM, ROM, or PROM in 4K increments. These 6"x10" boards mate to standard 44-pin edge connectors for ease of system expansion. By standardizing pinouts for address and data buses, control and power lines, it becomes simple to interchange processor boards, memory boards, I/O boards, etc., all within the same card cage, regardless of whose LSI devices are on any board.

Schweber has 18 outlets in the U.S. If none of them are handy for you, contact the manufacturer, Excel Products, 401 Joyce Kilmer Ave, New Brunswick NJ 08903; 201-249-6600.

FAIRCHILD F-8 KITS

If you don't want to do hardware diddling, Fairchild is peddling a F-8 Microprocessor Kit for $185 (plus tax where applicable). The "kit" contains a fully assembled and tested unit including an F8 CPU, a pre-programmed PSU (Program Storage Unit), an F8 Memory Interface Circuit, and 1K bytes of static RAM. It includes a wired edge connector, one end for the board, another for your TTY, and three wires for power. The board includes 32 TTL-compatible I/O bits, two interrupt levels, two programmable timers, and all the necessary control circuits. Internal signals have been brought out to the edge connector for possible system expansion. Just add power; there is no additional soldering or wiring to do. The price includes a F8 Programming Manual, F8 Databook, and the "Fairbug" program in the PSU. Fairbug includes such capabilities as a loader, memory dumper, debugger, and TTY and paper tape I/O drivers. They say its immediately available from Fairchild Distributers or from Fairchild Microsystems Division, 1725 Technology Dr., San Jose CA 95110.

MUSIC OF A SORT

Steve Dompier, 2136 Essex St, Berkeley CA 94705;
415-841-1868
[Reprinted from May 1975 *PCC*, Vol. 3, No. 5.]

IT WORKS!

I received my ALTAIR 8800 in the mail at 10 a.m., and 30 hours later it was up and running with only one bug in the memory! That turned out to be a scratch in a printed circuit that took 6 more hours to find. After that was fixed, everything worked!!

Now, what do you do with a machine that so far has no I/O boards or peripherals? Well, there's always the front panel switches and machine language, so I was soon busy making up programs to test all of the 8080's functions; and getting a good set of calluses on my ten input devices. There's a lot of 8080 instructions!

ZZZIIIPPP

I had just finished setting in a 'sort' program, and at the same time I was listening to a weather broadcast on a little transistor, low-frequency radio, which was sitting next to the Altair. I hit the 'run' switch on the computer and it took off sorting the same list of numbers over and over again.

At the same time my radio also took off!

The computer was sorting numbers and the radio was going ZZZIIIPP! ZZZIIIPP! ZZZIIIPP!

Well, what do ya know! My first peripheral device!!!

The radio was picking up the switching noise of the 8800. I tried some other programs to see what they sounded like, and after about 8 hours of messing around I had myself a program that could produce musical tones and actually make music; of a sort. (Or any other program you have!)

MUSIC

The closest sheet of music that I could find was *The Fool on the Hill* by the Beatles, so I translated it into OCTAL code, picked up the Altair, and headed down to Menlo Park for the 3rd meeting of the Bay Area Amateur Computer Users Group-Home Brew Computer Club. I thought everyone there should see just what a computer can do!

RECITAL

This being the Altair's first recital, it was a bit shy at first, and refused to power up--even though Fred's tape recorder was plugged into the same wall outlet and working just fine. One forty-foot extension cord and half an hour later we were ready. (Fred's tape machine turned out to be running on its own battery power, and all of the wall plugs were dead!)

The recital then proceeded with nary a glitch, much to everyone's delight. (Although during the demanded encore, the machine did break into its own rendition of *Daisy*, apparently genetically inherited.)

Data For THE FOOL ON THE HILL
Beatles

Address	Data	Address	Data
040	105	120	055
041	105	121	053
042	125	122	071
043	100	123	066
044	071	124	100
045	063	125	071
046	063	126	071
047	063	127	100
050	071	130	071
051	063	131	066
052	055	132	066
053	053	133	071
054	053	134	100
055	055	135	100
056	071	136	100
057	063	137	071
060	046	140	066
061	046	141	060
062	046	142	060
063	071	143	066
064	063	144	071
065	046	145	066
066	046	146	066
067	053	147	060
070	042	150	053
071	046	151	046
072	046	152	046
073	063	153	046
074	071	154	046
075	063	155	044
076	053	156	046
077	053	157	053
100	063	160	053
101	053	161	053
102	071	162	053
103	063	163	053
104	063	164	002
105	071	165	002
106	063	166	002
107	046	167	377
110	046		
111	046		
112	053		
113	042		
114	053		
115	046		
116	046		
117	053		

Data for "DAISY"

Address	Data	Address	Data
170	034	250	040
171	034	251	042
172	034	252	046
173	042	253	034
174	042	254	034
175	042	255	042
176	053	256	046
177	053	257	053
200	053	260	053
201	071	261	053
202	071	262	053
203	071	263	046
204	063	264	042
205	055	265	042
206	053	266	053
207	063	267	063
210	063	270	063
211	053	271	053
212	071	272	063
213	071	273	071
214	071	274	071
215	071	275	071
216	071	276	071
217	071	277	071
220	046	300	053
221	046	301	053
222	046	302	042
223	034	303	046
224	034	304	046
225	034	305	071
226	042	306	053
227	042	307	053
230	042	310	042
231	053	311	046
232	053	312	042
233	053	313	040
234	063	314	034
235	055	315	042
236	053	316	053
237	046	317	046
240	046	320	046
241	042	321	071
242	046	322	053
243	046	323	053
244	046	324	053
245	046	325	053
246	046	326	002
247	042	327	377

OCTAL CODES FOR NOTES

C	252	LOW OCTAVE
C#	240	
D	230	
D#	220	
E	211	
F	200	
F#	172	
G	162	
G#	154	
A	146	
A#	140	
B	132	
C	125	MIDDLE OCTAVE
C#	120	
D	114	
D#	110	
E	105	
F	100	
F#	075	
G	071	
G#	066	
A	063	
A#	060	
B	055	
C	053	HIGH OCTAVE
C#	050	
D	046	
D#	044	
E	042	
F	040	
F#	036	
G	034	
G#	033	
A	031	
A#	030	
B	026	
C	025	
Q	002	

Note; This is the quietest of the data notes. It can be used for spaces and rests. You may also like to put a number of these quiet 'notes' at the end of the music data, to give a space between playings.

With a little experimentation, you can make all kinds of interesting sounds. ie; sirens, ray-guns, etc.

PROGRAM TO MAKE AN ALTAIR 8800 PLAY MUSIC

000	LXI H	041	
001	b2	xxx	– ADDRESS OF FIRST DATA ENTRY
002	b3	xxx	
003	MOV A,M	176	
004	CPI	376	
005	b2	377	– START OVER DATA
006	JZ	312	
007	b2	000	
010	b3	000	
011	MVI D	026	
012	b2	xxx	– TEMPO DATA
013	DCR B	005	
014	JNZ	302	
015	b2	020	
016	b3	000	
017	MOV B,M	106	
020	DCR C	015	
021	JNZ	302	
022	b2	013	
023	b3	000	
024	DCR D	025	
025	JNZ	302	
026	b2	013	
027	b3	000	
030	INR L	054	
031	JMP	303	
032	b2	003	
033	b3	000	

TO RUN THE PROGRAM:

To run the program, push the 'RESET' switch, then push the 'RUN' switch.

To stop the program, push the 'STOP' switch.

TO MAKE YOUR OWN MUSIC:

Begin loading your music data anyplace after address 034. Be sure to load the starting address into H&L at address 002, 003.

Each data entry will be one beat of music.

****

NOTES

† Tempo-- The tempo is controlled by the value placed in address 012. Start out by trying 040.

† To Play Backwards-- Put 377 in front of all music data (to cause looping). Change address 001 to read the END of the music data. Change address 030 to DCR L (055).

† To Play all of the Memory-- Change address 001 data to a NOP (000). Change address 004, 005, 006 to NOP (000). This will cause program to read all of the memory, including the program instructions themselves.

† Radio Information-- A low-frequency radio around 330 KC works best, but any AM radio will pick up the music at quiet places on the dial.

Set the radio on or very close to the computer, start the program, and turn the dial on the radio until you get good sound. Some places will be much better than others, and some will pick up different sounds from the computer. Also, try moving the radio to different positions on or around the computer. Just rotating the radio 90 degrees can make a lot of difference in the sound you will get.

SCELBAL--A HIGHER LEVEL LANGUAGE FOR 8008/8080 SYSTEMS

Mark Arnold & Nat Wadsworth
Scelbi Computer Consulting, Inc.,
1322 Rear, Boston Post Rd, Milford CT 06460

[The publication described in the following article will be sold for around $50, and will contain over 300 pages of information. --JCW, Jr]

The goal of about ninety percent of small systems owners appears to be to get their systems up and running with some kind of I/O and then procure enough memory to support a higher level language.

Unfortunately in the past when a system owner reached the stage of having enough memory a major problem arose. Unless the individual had purchased an entire system from one or two select suppliers, the cost of a copy of a higher level language was likely to be out of reach!

Even if one was financially able to purchase a higher level language from an equipment manufacturer one was likely to find that such programs were designed to operate with specific I/O devices which the prospective language user might not have access to or desire to obtain. If one did not have those specific devices for which the program was designed, one was usually in a tough spot. Despite advertisements that such programs came "fully documented," the "full documentation" was not likely to include a source listing of the program. Hence, attempting to modify such a complex program was a risky, frustrating, and often downright impossible task. And, without doing so, one was hard put to make the language work with unique types of I/O devices. Furthermore, such programs could not practically be modified to serve the particular wishes of individual users. If you were not satisfied with the program and what the program author's had decided to emphasize or leave out, that was simply too bad!

Few "canned" programs can be tailored to have all the features desired by all the possible potential users. To attempt to do so would result in programs requiring more memory than users could afford. The answer to this problem is, of course, to supply the programs in such a manner that they can be readily modified and altered by the users. This means, simply, that the detailed source listing for the program must be made available to the purchaser. Assisting the program owner by also providing detailed comments with the listing, a general overview of the program's organization and operation, and general flow charts can further enhance the value of the program to the owner. With this information available, the program user can safely proceed to tailor the capabilities of the program to serve the user's particular interests and requirements.

This is the approach SCELBI COMPUTER CONSULTING, INC., has taken in presenting its new higher level language for 8008/8080 machines. The language has been given the name SCELBAL for SCientific ELementary BAsic Language. As the reader can easily surmise from the title it is similar in capabilities to the highly popular language referred to as BASIC. This language was specifically developed to be able to run on 8008 based microcomputers. It is believed to be the first such higher level language to be made generally available that is capable of running in a system equipped with the ubiquitous 8008 CPU. The program can of course also be run on systems using the more powerful 8080 CPU though it is not as memory efficient as it could have been if the program had forsaken 8008 capability.

The language was developed to operate in an INTERPRETIVE mode. This means that the entire program resides in memory at one time along with the program written in the higher level language that is to be executed. When the INTERPRETER is given the RUN command it immediately proceeds to INTERPRET each line of the higher level language program and perform the necessary calculations and functions. This differs from a COMPILER which would first convert the higher level language source listing to machine code, then later execute the machine code.

A COMPILER oriented system generally is cumbersome to run on a small system that lacks reliable, high speed bulk memory storage facilities. For instance, if the program had been designed as a complier, the following steps would have been necessary in order to execute a higher level language program.

First one would have to load an Editor program into the computer and create the desired higher level language version of a program as a source listing. A copy of the source listing would then have to be saved on an external memory medium. Next, a portion of the compiler program - the actual compiler, would have to be loaded into memory. When it was resident, one would produce the desired machine code version of the higher level language statements by having the compiler process the source listing several times. (Much as an Assembler program would process the mnemonic listing when programming in machine language.) The machine code produced would have to be stored on an external memory device at this stage. Finally, the RUN TIME portion of the compiler would have to be loaded into the computer along with the machine code produced by the COMPILE portion of the program. The higher level language program would then finally be ready to run. Too bad if you made an error in the original source coding for the program that was not detected until run time. You would have to go all the way back to the Editor program to correct the higher level language source listing and start the process over again!

Developing the program as an INTERPRETER eliminates the requirement for the constant use of an external bulk memory device in order to get a program from the concept to execution stage. An INTERPRETER is definitely a much more convenient program for the small systems user. The entire INTERPRETER program resides in memory at one time. An area is set aside in memory to hold the higher level program. An executive portion of the program allows the user to enter the higher level language listing directly into the area where it will be operated on when the program is executed. The executive in SCELBAL will provide for the user entering a program from a manual input device such as a keyboard. Or, if the user desires to run a program that has been developed previously, a LOAD command will direct the program to read in a program from an external bulk memory device such as a magnetic tape peripheral.

SCELBAL has been designed so that it can operate in a "calculator" mode or operate in a stored program mode. In the calculator mode, each statement is executed immediately after it is entered by the input device. In this mode, the program is ideal for solving simple formulas when the user only needs to obtain a few values.

When operating in the stored program mode, the INTERPRETER will follow an entire series of instructions as directed by the higher level program. To enter a program that will be operated on as a stored program, the operator simply assigns a line number at the beginning of each statement.

The executive portion of the package allows the user to "edit" a program at any time. Lines may be deleted and new lines entered anywhere in the program. If the operator makes a clerical error while entering a line, a special erase code may be used to effectively backspace within a line and then re-enter the correct characters. Furthermore, the executive checks for various types of syntax errors as statements are entered, and will display a two character error code to the programmer when such errors are detected.

The executive portion of SCELBAL has five major commands available to the operator which are defined and explained below.

SCR for SCRatch effectively clears out any previous program stored in the program buffer along with any variable values.

LIST causes the present contents of the program buffer to be displayed for review or to make a copy for record keeping if a printing device is in use.

RUN causes the higher level language program stored in the program buffer to be executed by the INTERPRETER.

SAVE. This command directs the program to save a copy of the program stored in the program buffer on the user's external bulk storage device. A program saved in this manner can later be restored for execution by using the following command.

LOAD. This command causes the program to read in a copy of a program from an external device that was previously written using the above SAVE command.

A higher level language program is made up of STATEMENTS that direct the machine to perform selected types of operations. The SCELBAL language can execute 12 different types of STATEMENTS which are explained below plus the END statement which is used to signify the end of a program.

The REM for REMarks statement indicates a comments line which is ignored as far as program execution is concerned. Information on a REMarks line is intended only for the use of programmers and is used to document a program.

The LET statement is used to set a variable equal to a numerical value, another variable, or an expression. For instance the statement:

```
LET X = (Y*Y + 2*Y - 5)*(Z + 3)
```

would mean that the variable X was to be given the value of the expression on the right hand side of the equal sign.

The IF combined with the THEN statement allows the programmer to have the program make decisions. SCELBAL will allow more than one condition to be expressed in the statement. Thus:

```
IF X <= Y THEN LL
```

states that IF X is less than OR equal to Y that the program is to go directly to line number LL. Otherwise, the program is to continue on to the next statement in the program.

GOTO directs the program to jump immediately to a specified line number. The GOTO statement is used to skip over a block of instructions in a multiple segment or subroutined program.

The FOR, NEXT and STEP statements allow the programmer to form program loops. For example, the series of statements:

```
FOR X = 1 TO 10
LET Z = X*X + 2*X + 5
NEXT X
```

would result in Z being calculated for all the integer values of X from 1 to 10. While SCELBAL does not require the insertion of a STEP statement in a FOR - NEXT loop, a STEP value may be defined. The implied STEP value is always 1. However, it may be altered to be an integer value other than 1 by following the FOR range statement by the STEP statement and a parenthesis containing the STEP size. Thus:

```
FOR X = 1 TO 10 STEP (2)
```

would result in X assuming values of 1, 3, 5, 7 and 9 as the FOR - NEXT loop was traversed.

GOSUB is used to direct the program to execute a statement or group of statements as a subroutine. The statement is used by designating the line number in the program where subroutine execution is to begin.

The RETURN statement is used to indicate the end of a subroutine. When a RETURN statement is encountered the program will return to the next statement immediately following the GOSUB statement which directed the program to the subroutine.

SCELBAL permits multiple nesting of subroutines in a program.

DIM for DIMension is used to specify the formation of a one dimensional array in a program. Up to four such arrays having a total of up to 64 entries are permitted in a program when running SCELBAL. The statement:

DIM K(20)

sets up space for an array containing 20 entries. (Array size must be designated by a numerical value, not a variable.)

The DIM is an optional statement that may be left out of the program to provide additional program storage space in systems having limited memory.

INPUT is used to cause the program to wait for an operator to INPUT information to the program. After the information has been received, operation of the program automatically continues.

PRINT is used to output information from the program. Using the PRINT statement the user may direct the program to display the value of variables, expressions, or any information such as messages. The PRINT statement allows for multiple mixed output on a single line, and the option of providing a carriage-return and line-feed after outputting information or suppressing that function. For instance, the statement:

PRINT 'X IS EQUAL TO: ';X

would result in the program first printing the message "X IS EQUAL TO:" and then the value of the variable X on the same line. After the value of the variable had been displayed, a carriage-return and line-feed combination would be issued. To suppress the issueing of the CR & LF the programmer would merely include another semicolon at the end of the statement! A comma sign in a PRINT statement will direct the output to start at the next TAB point in a line. A special function may also be called upon to direct the output to begin at a specified position in a line to allow for neat formatting.

The power of the language is further enhanced by the inclusion of seven functions that may be used in statements. The seven functions available in SCELBAL are discussed below.

INT returns the INTeger value of the expression, variable, or number requested as the argument. This is the greatest integer number less than or equal to the argument.

SGN returns the SiGN of the variable, number, or expression. If the value is greater than zero, the value +1.0 is returned. If the value is less than zero, the value -1.0 is returned. The value 0 is returned when the expression or variable is zero.

ABS returns the ABSolute value (magnitude without regard to sign) of the variable or expression identified as the argument of the function.

SQR returns the SQuare Root of the expression, variable, or number.

RND produces a semi-psuedo-RaNDom number in the range of 0 to 0.99. This function is particularly useful to have available for games programs.

CHR is the CHaRacter function. It may be used in a PRINT statement and will cause the ASCII character corresponding to the decimal value of the argument to be displayed. (A reverse function is available for the INPUT statement which will return the decimal value of a character when it is inputted.)

TAB may also be used in a PRINT statement to direct the display device to space over to the column number specified in the argument. This function allows the programmer to format the output into neat columns.

GENERAL INFORMATION

User defined variables are limited to one or two characters. A variable must begin with a letter of the alphabet. Limiting variables to a maximum of two characters helps conserve memory space. Up to twenty different variables may be def'ned in a single program.

SCELBAL allows the use of fixed and floating point notation. A minimum of twenty-three binary bits are used in the mantissa portion of all calculations allowing for six to seven significant decimal digits to be entered or outputted. The exponent range is from plus to minus the 38th power. Numbers may be inputted in either fixed or floating point notation. Output from the program is automatically selected to be either fixed or floating point, depending on the size of the number that is to be displayed.

The package, without the optional DIM statement, is designed to run in an 8K 8008 or 8080 system leaving approximately 1250 bytes for program storage. With this amount of storage available, surprisingly complex programs can be executed. The program authors have successfully loaded and run such games as Lunar Landing in this configuration by reducing the number of messages issued to the player.

The DIM statement requires approximately three pages of memory. It is recommended that users desiring to include the DIM capability have more than the minimum 8K of memory available in their system. A particularly attractive feature of SCELBAL is that users with more than 8K of memory can use the additional space for program storage. Thus, for example, a 12K system will enable a user to execute SCELBAL programs having as many as 150 to 200 statements!

A major concern of the developers of SCELBAL was that the 8008 CPU might make the language so slow that it was impractical for the user. Our tests indicate that the time to perform typical calculations, while they are slow compared with more powerful machines, are certainly tolerable. For instance, the typical response time between the displaying of a new set of parameters when running the Lunar Landing game is in the order of six to seven seconds. A program that calculates the mortgage payments on a house on a monthly basis, and displays such values as the payment number and balance after each payment, requires a few seconds between the displaying of each new line of information. A dice playing game responds with new throws of the dice in the order of a second or so when using a formula that includes the use of the random number generator. These times are by no means fast, but they are certainly adequate for the intended uses of this language on an 8008 system. The developers were pleasantly surprised with the overall speed performance of the package. Of course, these response times can be cut almost in half by using an 8008-1 CPU. Naturally, if the program is installed in an 8080 system, the response time is improved an order of magnitude.

Since the program will be supplied in the form of a publication that includes a complete highly commented source listing (as well as assembled object code for both the 8008 and 8080), the user who desires to modify or expand the capabilities of the basic package will be in a position to do so. It is felt that the availibility of such a powerful program in this form will greatly enhance the general usefulness of small systems and open new vistas to users. The program in this form should also be of considerable value to educationalists who desire a good reference framework from which to introduce students to the development of similar packages.

The publication will be made available in June, 1976, by the developer, Scelbi Computer Consulting, Inc., 1322 Rear - Boston Post Road, Milford, CT 06460.

TINY BASIC, EXTENDED (Part Two)

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

In the preceeding article on TINY BASIC, EXTENDED (TBX), notes concerning the loading and use of TBX were presented, along with an octal listing of the entire interpreter [*Dr Dobb's Journal of Tiny BASIC Calisthenics & Orthodontia* Vol. 1, No. 1]. This article presents source code matching that octal code, documentation of the implementation, some modifications and error corrections, notes on the addition of the DTA statement, and an announcement of two relocated versions of TBX requested by some of our readers.

TBX is not meant to be the last word in Tiny BASIC interpreters. Almost certainly, its users will find ways to improve it. Please keep its creators, and our readers, informed of those improvements.

NOTES ON NOTES

Before continuing, a few remarks are necessary concerning assumptions and working used in the source listing and the text that follows it.

1. All addresses will be given in *split octal* with no separation character, i.e., 012504 (true octal); 025104 (split octal).

2. Registers will be referred to by letter: A, B, etc. Register pairs will be referred to as BC, DE, and HL. If a register pair holds an address, the most significant bits will fall in the first letter.

3. *The source listing is NOT the result of computer assembly*. It was hand-typed in a format similar to assembly language. One caution--labels are unique *only* within a given routine. Therefore, in the whole of TBX, labels may be duplicated.

4. The terms "label" and "line number" will be used interchangeably when referring to BASIC lines.

5. On the source listing, double lines are used to separate major routines.

A NEW FEATURE: The DTA Statement

The DTA statement allows the programmer to initialize several variables at one time and is thus more convenient to use than LET statements. DTA is more like DATA statements of FORTRAN than the READ-DATA statements of BASIC. DTA statements may be used anywhere in a program and as many times as required.

```
EX 1. 12 DTA A(1)=1,2,3,4;B(4)=5,6;X=10
      RESULTING VALUES:  A(1)=1
                         A(2)=2
                         A(3)=3
                         A(4)=4
                         B(4)=5
                         B(5)=6
                         X=10
EX 2. 20 DTA A=10,20,30
      RESULTING VALUES:  A=10
                         B=20
                         C=30
```

Changes required in TBX to add DTA statements (octal dump form):

```
031200 315 147 024 043 315 044 023 247
031210 311
031230 000 000 000 232 150 104 124 301
031240 133 310 232 330 275 132 343 231
031250 256 254 331 200 031 245 324 147
031260 231 265 273 031 240 322 304 322
031270 375
033326 231 233
```

RELOCATED VERSIONS OF TBX

At the requests of readers, we have made two relocations of TBX. As you are no doubt aware, the original TBX began at 020000 split octal. The two new versions begin at 000000 and 011000. The octal listing of the 000000 version will appear in a later issue of this Journal. The 011000 version seems to especially interest people with Suding operating systems (Suding CRT, etc.). At the present, either version--or both--can be obtained from us on Suding cassette. If you have already ordered one and received the original version on cassette, we will provide the relocated version free of charge if you will return the cassette with your order. The charge for new orders will be the same as indicated in the preceeding issue: $5, for the Suding cassette. Be sure to request the version desired, namely 020000, or 011000, or 000000. Send orders to:

TBX Tape
c/o John Arnold
Rt 4, Box 52A
Tyler TX 75701

HOW IT DOES WHAT IT DOES

IL Executor, ILXQT and IL Program (021254-022037, & 031300-033376):

The fundamental IL instruction consists of two bytes. The two most significant bits of the first byte are used to encode the type of IL instruction while the remaining bits in the first and second byte represent an address. The four IL instruction types are specified in octal as follows:

IL JUMP	Oxx yyy	Transfers IL Program to IL Instruction at Oxxyyy.
IL CALL	1xx yyy	Calls IL Subroutine at Oxxyyy.
TST	2xx yyy	Compares Character Strings. Test Failure Transfers IL Program to Oxxyyy.
ML CALL	3xx yyy	Calls Machine Language Program at Oxxyyy.

The IL Executor program, ILXQT, merely sorts out the IL instructions according to the above list and carries out the appropriate action. DE is used as the cursor in scanning the BASIC text. HL serves as the IL program counter. When a program other than ILXQT has system control, care must be exercised not to use DE for a purpose other than scanning unless its value is saved and returned before returning to ILXQT. BC, HL, and A may be used by system routines as required. (Note: HL is pushed and popped by ILXQT to maintain the status of the IL program counter.)

SPECIAL REMARKS

1. IL JUMP: Note that an IL JUMP and a machine language jump (JMP) are not the same. After an IL JUMP is executed, ILXQT expects to find another IL insstruction not a machine language instruction. Therefore, an IL JUMP cannot be used to transfer to a machine language program.

2. IL CALL: The same applies to IL CALLs. They cannot be used to call machine language subroutines. That requires the ML CALL type IL instruction (see **4.**, following). The IL return address for the IL CALL is placed on the 8080 stack for later use. ML routines must not leave trash on the stack or proper return within the IL program will not be made.

3. TST: The TST IL instruction is actually more than two bytes in length. Following the instruction byte pair are the ASCII string characters to be compared against BASIC text. There is one byte for each character in the string with the parity bit (No. 7) set only on the last character. The parity bit is used by ILXQT to detect the end of the string. As an example, consider the test for "LET" (see DIR, 032022):

```
232 }
041 }   TST
114     "L"
105     "E"
324     "T" + 200
```

If the comparison to BASIC text fails, the IL program transfers to the fail address 032041 for the next IL instruction. In this case the cursor (DE) is set back to rescan the BASIC text. If a match is found, the IL program continues at the IL instruction just after the " 'T' + 200."

4. ML CALL: The greater number of routines used in TBX are machine language calls made by ILXQT. Return to ILXQT from such a routine occurs when a machine langauge RET is executed. A return option is available to the programmer. If the carry is set upon return to ILXQT, the next IL instruction is skipped and the second one is executed. If the carry is reset, the next IL instruction is executed. This feature allows various tests to be handled as ML CALLs. TSTL--Test Label--is an example.

5. THE IL PROGRAM can be studied to get an idea of the manner in which BASIC lines are interpreted. Often used, run time commands are tested first to achieve greater speed in execution. System commands such as RUN, LST, etc. are placed last in the test sequence. You will notice that the IL Program is somewhat disordered. This "house that Jack built" appearance is the result of adding features to the original TB.

EVOLUTIONARY NOTE

A modification was made to TBX after the octal listing was published in the previous issue, but before this source listing was produced. Therefore the source contains the modifications but the octal listing does not. The change involves the INNUM and NLINE routines. The net effect of these changes will be that IN statements will be terminated by a CR--not a SPACE as in the original TBX. To make the modifications, follow the steps below:

1. Re-enter the INNUM routine using the source listing.
2. Re-enter the NLINE routine using the source listing.
3. Add a test in your INPUT routine that will inhibit echo of a CR.
4. Make changes in the IL program at locations given below. Use the source listing to obtain the corrected values.

031325	031326
031334	031335
032002	032003
032006	032007
032202	032203
032271	032272

ERRORS & CORRECTIONS

These errors were noted after the listing that follows was produced. Thus, the corrections given here should be made in the octal code given in the preceeding issue *and* in the listing given in this issue.

ERROR	FIX ADDRS	CHANGE FROM	TO
1. "FOR" statement syntax error not functioning properly.*	032127	226	232
	032130	363	121
2. "IN" statement does not issue a crlf. After fix, a semicolon ";" at the end of an "IN" line inhibits crlf.*	032245	322	032
	032246	304	143
	032143		232
	032144		202
	032145		273
	032146		032
	032147		216
3. Array syntax error not functioning properly.*	033211	226	233
	033212	355	077
	033223	226	233
	033224	355	077
	033241	226	233
	033242	355	077
	033254	226	233
	033255	355	077
	033266	226	233
	033267	355	077
	033275	226	233
	033276	355	077
4. System destroys itself after first line entry following turn-on. Issuing a "new" command first will initialize the system properly. The fix given is satisfactory as well.*	033354		001
	033355		034
	034000		377
	034001		377
5. "SZE" command giving erroneous values.	031007	376	115
	031010	033	026
6. "TB" function incorrectly named. It should be called "SP" for space.	031312	124	123
	031313	302	332

* These problems were reported by Linchen Wang with suggested corrections. Dick Whipple modified the modifications, and submitted these fixes.

BUFIN: A software buffer is used to hold line data from the input device. BUFIN is an ML routine used to load and edit the buffer. Character deletion and whole line erasure are provided for in this routine. B is used to count characters. If B exceeds 72, an error is reported. If entry to BUFIN is made at GETLINE (020070), a colon is output at the beginning of the line. HL is used as cursor in the buffer.

```
TAG      ADDRESS I1  I2  I3   MNEMONIC        COMMENTS

BUFIN    020000  041 111 020  LXI H BUFSTRT   SUBROUTINE TO LOAD BUFFER
         020003  006 110      MVI B 72D       SET LINE LENGTH
OVER     020005  337          RST INPUT
         020006  376 015      CPI 'CR'
         020010  312 036 020  JZ END
         020013  376 177      CPI 'DEL'
         020015  312 040 020  JZ RUBOUT
         020020  376 014      CPI 'CNTRL L'
         020022  312 067 020  JZ NEWLINE
         020025  167          MOV M,A
         020026  043          INX H
         020027  005          DCR B
         020030  312 306 026  JZ ERR1         LINE TOO LONG
         020033  303 005 020  JMP OVER
END      020036  167          MOV M,A
         020037  311          RET
RUBOUT   020040  053          DCX H
         020041  004          INR B
         020042  076 077      MVI A '?'
         020044  357          RST OUTPUT
         020045  303 005 020  JMP OVER
FIXINSR  020050  332 000 021  JPC CONT        MODIFICATION OF INSERT
                                              SUBROUTINE REQUIRED FOR
                                              ASCII VERSION OF TBX
         020053  076 057      MVI A '/'
         020055  276          CMP M
         020056  322 000 021  JNC CONT
         020061  303 371 020  JMP LOOP1+2
         020064  000 000 000  NOP'S
NEWLINE  020067  327          RST CRLF
GETLINE  020070  076 072      MVI A ':'       OUTPUT PROMPT
         020072  357          RST OUTPUT
         020073  303 000 020  JMP BUFIN
         020076  000 000
#
# LOC 020100-020110 IS UNUSED. THE BUFFER RESIDES BETWEEN 020111-020220.
#
```

ASCIN: This routine moves a byte from the BASIC text to A using the cursor DE. If the byte represents an ASCII number (060-071), the upper four bits are masked off.

```
ASCIN    020221  032          LDAX D          ASCII INPUT SUBROUTINE
                                              FROM CURSOR LOCATION.
                                              NUMBER DATA MASKED.
         020222  376 060      CPI 'Ø'
         020224  330          RC
         020225  376 072      CPI ':'
         020227  320          RNC
         020230  346 017      ANI 00001111B
         020232  311          RET
#
# LOC 020233-020264 IS UNUSED.
#
```

TSTL: This ML routine is used to determine whether the line in the buffer has a label or not. If so, the label is converted to binary by BIN and stored in CURLBL (Current Label). The carry is set and return is made to ILXQT. Otherwise, a zero is placed in CURLBL and return is made with the carry reset.

```
TSTL     020265  021 111 020  LXI D BUFSTRT   TEST FOR LABEL SUBROUTINE
         020270  325          PUSH D
         020271  032          LDAX D
         020272  376 040      CPI 'SP'
         020274  023          INX D
         020275  312 271 020  JZ SKIP
         020300  033          DCX D
         020301  041 000 000  LXI H ØD
         020304  376 100      CPI 'A'-1
         020306  332 320 020  JPC LBL
CMND     020311  042 350 033  SHL CURLBL
         020314  000          NOP
         020315  321          POP D
         020316  311          RET
         020317  000          NOP
LBL      020320  315 331 020  CALL BIN
         020323  042 350 033  SHL CURLBL
         020326  067          STC
         020327  321          POP D
         020330  311          RET
```

BIN: As the text cursor DE scans an ASCII number, BIN converts it to binary in HL. The first non-number encountered signals the end of conversion and return is made to the calling program.

```
BIN      020331  315 221 020  CALL ASCIN      ASCII-BINARY SUBROUTINE
         020334  376 012      CPI 1ØD
         020336  320          RNC
         020337  023          IND D
         020340  104          MOV B,H
         020341  115          MOV C,L
         020342  051          DAD H
         020343  051          DAD H
         020344  011          DAD B
         020345  051          DAD H
         020346  332 311 026  JPC ERR2        NUMBER TOO LARGE
         020351  117          MOV C,A
         020352  006 000      MVI B ØD
```

```
TAG     ADDRESS I1  I2  I3   MNEMONIC       COMMENTS

        020354  011          DAD B
        020355  303 331 020  JMP BIN
INSRT   020360  325          PUSH D         LINE INSERTION SUBROUTINE
                                            ALSO DELETES AND OVERWRITES
                                            AS REQUIRED
        020361  052 350 033  LHL CURLBL
        020364  104          MOV B,H
        020365  115          MOV C,L
        020366  041 111 020  LXI H BUFSTRT
        020371  076 071      MVI A '9'
LOOP1   020373  043          INX H
        020374  276          CMP M
        020375  303 050 020  JMP FIXINSR    REFER TO PAGE 1
        021000  345          PUSH H
        021001  026 001      MVI D 1D
        021003  076 015      MVI A 'CR'
LOOP2   021005  276          CMP M
        021006  312 016 021  JZ CONT1
        021011  024          INR D
        021012  043          INX H
        021013  303 005 021  JMP LOOP2
CONT1   021016  172          MOV A,D
        021017  062 356 033  STA COUNT      SAVE LENGTH OF TEXT
        021022  321          POP D
        021023  052 352 033  LHL PRGSTRT
LOOP4   021026  176          MOV A,M
        021027  270          CMP B
        021030  312 052 021  JZ CONT2
        021033  322 064 021  JNC HERE       INSERT LINE HERE
        021036  043          INX H
LOOP3   021037  043          INX H
        021040  175          MOV A,L
        021041  206          ADDR M
        021042  157          MOV L,A
        021043  322 026 021  JNC LOOP4
        021046  044          INR H
        021047  303 026 021  JMP LOOP4
CONT2   021052  043          INX H
        021053  176          MOV A,M
        021054  271          CMP C
        021055  312 170 021  JZ OVRDEL
        021060  332 037 021  JPC LOOP3
        021063  053          DCX H
HERE    021064  053          DCX H
        021065  325          PUSH D
        021066  353          XCHG
        021067  052 354 033  LHL PRGEND
        021072  345          PUSH H
        021073  072 356 033  LDA COUNT
        021076  306 003      ADI 3D
        021100  205          ADDR L
        021101  322 105 021  JNC CONT3
        021104  044          INR H
CONT3   021105  157          MOV L,A
        021106  315 340 030  CALL MEMTEST   CHECK FOR MEMORY DEPLETION
        021111  104          MOV B,H
        021112  115          MOV C,L
        021113  341          POP H
LOOP5   021114  176          MOV A,M
        021115  002          STAX B
        021116  053          DCX H
        021117  013          DCX B
        021120  174          MOV A,H
        021121  272          CMP D
        021122  302 114 021  JNZ LOOP5
        021125  175          MOV A,L
        021126  273          CMP E
        021127  302 114 021  JNZ LOOP5
        021132  023          INX D
        021133  052 350 033  LHL CURLBL
        021136  353          XCHG
        021137  162          MOV M.D
        021140  043          INX H
        021141  163          MOV M,E
        021142  043          INX H
        021143  072 356 033  LDA COUNT
        021146  074          INR A
        021147  167          MOV M,A
        021150  043          INX H
        021151  321          POP D
LOOP6   021152  032          LDAX D
        021153  167          MOV M,A
        021154  376 015      CPI 'CR'
        021156  312 166 021  JZ END
        021161  043          INX H
```

INSRT: This ML routine is perhaps the most powerful and intricate program in TBX. It handles virtually all line editing. Lines are inserted by label (line number), deleted, and over-written as required. Ignoring the label, INSRT gets the length of the line in the buffer, adds 1, and places the result in COUNT for later use. Beginning at PRGSTRT (Starting location for BASIC programs), INSRT compares the line numbers of lines already stored to the line number in CURLBL. If a direct match is found, then either a deletion or overwrite is needed. In either case a branch is made to point OVRDEL. If a match is not found, comparison continues until the stored line number exceeds CURLBL. At this point, the program branches to HERE (021064).

The HERE routine inserts the new line at the point designated in B above. Before insertion begins, a check is made to be sure that enough memory space is available for the new line. If not, an error is called. If all is well, insertion continues. Beginning at the first stored line number above CURLBL, all BASIC lines are moved up in memory an amount equal to COUNT plus 3 decimal. Space is thus made available for the new line. The line number, the length of text (as stored in COUNT), and the text of the new line are then moved into this space. At this point normal return is made to ILXQT.

```
TAG      ADDRESS I1  I2  I3   MNEMONIC         COMMENTS

         021162  023          INX D
         021163  303 152 021  JMP LOOP6
END      021166  321          POP D
         021167  311          RET
OVRDEL   021170  053          DCX H            OVERWRITES OR DELETES A LINE
         021171  345          PUSH H
         021172  043          INX H
         021173  043          INX H
         021174  043          INX H
LOOP7    021175  176          MOV A,M
         021176  376 015      CPI 'CR'
         021200  312 207 021  JZ CONT6
         021203  043          INX H
         021204  303 175 021  JMP LOOP7
CONT6    021207  043          INX H
         021210  353          XCHG
         021211  052 354 033  LHL PRGEND
         021214  043          INX H
         021215  104          MOV B,H
         021216  115          MOV C,L
         021217  341          POP H
LOOP8    021220  032          LDAX D
         021221  167          MOV M,A
         021222  043          INX H
         021223  023          INX D
         021224  172          MOV A,D
         021225  270          CMP B
         021226  302 220 021  JNZ LOOP8
         021231  173          MOV A,E
         021232  271          CMP C
         021233  302 220 021  JNZ LOOP8
         021236  053          DCX H
         021237  042 354 033  SHL PRGEND
         021242  072 356 033  LDA COUNT
         021245  376 001      CPI 1D
         021247  302 361 020  JNZ INSRT+1
         021252  321          POP D
======== 021253  311          RET
ILXQT    021254  041 002 032  LXI H ILSTRT     INTERPRETIVE LANGUAGE EXE-
NXTIL    021257  176          MOV A,M          CUTION ROUTINE
         021260  376 200      CPI 200
         021262  322 314 021  JNC ML
IL       021265  376 100      CPI 100
         021267  322 300 021  JNC ILCALL
ILJMP    021272  043          INX H
         021273  156          MOV L,M
         021274  147          MOV H,A
         021275  303 257 021  JMP NXTIL
ILCALL   021300  346 077      ANI 00111111B
         021302  107          MOV B,A
         021303  043          INX H
         021304  116          MOV C,M
         021305  043          INX H
         021306  345          PUSH H
         021307  140          MOV H,B
         021310  151          MOV L,C
         021311  303 257 021  JMP NXTIL
ML       021314  376 300      CPI 300
         021316  322 000 022  JNC MLCALL
TST      021321  346 077      ANI 00111111B    STRING COMPARASION ROUTINE
         021323  107          MOV B,A
         021324  043          INX H
         021325  116          MOV C,M
         021326  043          INX H
LOOP1    021327  032          LDAX D
         021330  023          INX D
         021331  376 040      CPI 'SP'
         021333  312 327 021  JZ LOOP1
         021336  033          DCX D
         021337  325          PUSH D
         021340  353          XCHG
LOOP2    021341  032          LDAX D
         021342  376 200      CPI 200
         021344  322 363 021  JNC END1
         021347  276          CMP M
         021350  043          INX H
         021351  023          INX D
         021352  312 341 021  JZ LOOP2
END2     021355  321          POP D
         021356  140          MOV H,B
         021357  151          MOV L,C
         021360  303 257 021  JMP NXTIL
END1     021363  346 177      ANI 01111111B
         021365  276          CMP M
         021366  302 355 021  JNZ END2
         021371  353          XCHG
         021372  301          POP B
         021373  023          INX D
         021374  043          INX H
```

OVRDEL first deletes the line with the same number as CURLBL. The length of this line is determined by scanning and then all lines above it in memroy are moved down by this amount. PRGEND (Endling location of BASIC program) is adjusted to always reflect the end of BASIC line storage. At this point COUNT is checked. If it is one, a deletion is all that is required and return to ILXQT is made. Otherwise, the program branches to near the beginning of INSRT so that the buffered line can be inserted. This time no match will be found (the deletion step took care of that).

When the BASIC program storage area is initialized after typing NEW, the highest line number (377377) is stored at the beginning of the area. This is required to establish a base for INSRT to begin its function.

```
TAG     ADDRESS I1  I2  I3   MNEMONIC           COMMENTS

        021375  303 257 021  JMP NXTIL
MLCALL  022000  346 077      ANI 00111111B      MACHINE LANGUAGE CALLING
                                                PROGRAM
        022002  043          INX H
        022003  116          MOV C,M
        022004  043          INX H
        022005  345          PUSH H
        022006  041 015 022  LXI H RETRN
        022011  345          PUSH H
        022012  147          MOV H,A
        022013  151          MOV L,C
        022014  351          PCHL
RETRN   022015  341          POP H
        022016  322 257 021  JNC NXTIL          NORMAL RETURN
        022021  043          INX H
        022022  043          INX H
        022023  303 257 021  JMP NXTIL          ALTERNATE RETURN
ASCOUT  022026  041 357 033  LXI H ZONE-1       ASCII OUTPUT ROUTINE
        022031  357          RST OUTPUT         ZONE DECREMENTED AND RESET
                                                AS REQUIRED
        022032  043          INX H
        022033  065          DCR M
        022034  300          RNZ
        022035  066 017      MVI M 15D
        022037  311          RET
```

ASCOUT: This routine outputs an ASCII character via the external OUTPUT routine and decrements the location called ZONE. ZONE is used to keep track of print positioning on the output device. If ZONE should reach zero on a given call of ASCOUT, it is reset to 15 decimal.

```
#
# LOC 022040-022100 IS UNUSED
#
IOUT    022101  345          PUSH H             INTEGER OUTPUT ROUTINE
        022102  325          PUSH D             H&L IS OUTPUTTED IN DECIMAL
        022103  305          PUSH B             PROVISION MADE FOR ZERO
        022104  353          XCHG               SUPPRESSION.
        022105  016 000      MVI C 0
        022107  041 020 047  LXI H 10,000D
        022112  315 147 022  CALL CNVRT
        022115  041 350 003  LXI H 1,000D
        022120  315 147 022  CALL CNVRT
        022123  041 144 000  LXI H 100D
        022126  315 147 022  CALL CNVRT
        022131  041 012 000  LXI H 10D
        022134  315 147 022  CALL CNVRT
        022137  173          MOV A,C
        022140  315 201 022  CALL ASCOUT
        022143  301          POP B
        022144  321          POP D
        022145  341          POP H
        022146  311          RET
CNVRT   022147  006 377      MVI B 377
LOOP    022151  004          INR B
        022152  173          MOV A,E
        022153  225          SUB L
        022154  137          MOV E,A
        022155  172          MOV A,D
        022156  234          SBB H
        022157  127          MOV D,A
        022160  322 151 022  JNC LOOP
        022163  173          MOV A,E
        022164  205          ADDR L
        022165  137          MOV E,A
        022166  172          MOV A,D
        022167  214          ADC H
        022170  127          MOV D,A
        022171  170          MOV A,B
        022172  271          CMP C
        022173  310          RZ
        022174  015          DCR C
        022175  315 201 022  CALL BCDOUT
        022200  311          RET
BCDOUT  022201  000 000 000  NOP'S              BCD TO ASCII CONVERSION
        022204  000          NOP
        022205  306 060      ADI 060
        022207  315 026 022  CALL ASCOUT
        022212  311          RET
```

IOUT: IOUT is used to convert the binary number in HL to ASCII for outputting. The routine works by subtracting binary equivalents of decreasing powers of 10 from the binary value in HL. The number of times each power of 10 can be subtracted without producing a negative result represents the digit for the respective decimal place. If C is zero, leading zeroes are not outputted. CNVRT is a subroutine of IOUT that actually does the conversion and outputting of each digit. BCDOUT is a subroutine that adds 060 to the BCD value of the digit to produce the ASCII value. ASCOUT is called so that ZONE will be decremented.

```
LST     022213  325          PUSH D             SUBROUTINE TO LIST BASIC
                                                PROGRAMS
        022214  052 306 033  LHL LSTEND
        022217  053          DCX H
        022220  104          MOV B,H
        022221  115          MOV C,L
        022222  052 304 033  LHL LSTSTRT
        022225  353          XCHG
        022226  033          DCX H
NLINE   022227  023          INX D
        022230  327          RST CRLF
        022231  170          MOV A,B
        022232  272          CMP D
        022233  302 243 022  JNZ CONT
        022236  171          MOV A,C
        022237  273          CMP E
        022240  312 275 022  JZ END
```

LST: An ML routine used to list BASCI program lines beginning at the location stored in LSTSTRT and ending at the location stored in LSTEND. LBLOUT is used to output the line number for each line. The text length byte is skipped and the text is outputted until a CR is detected. The CR produces a new line and so long as the address in LSTEND is not exceeded, listing continues.

```
TAG     ADDRESS I1  I2  I3   MNEMONIC        COMMENTS

        022243  032          LDAX D
        022244  147          MOV H,A
        022245  023          INX D
        022246  032          LDAX D
        022247  157          MOV L,A
        022250  315 205 026  CALL LBLOUT
        022253  023          INX D
LOOP    022254  023          INX D
        022255  032          LDAX D
        022256  376 015      CPI 'CR'
        022260  312 227 022  JZ NLINE
        022263  305          PUSH B
        022264  345          PUSH H
        022265  315 026 022  CALL ASCOUT
        022270  341          POP H
        022271  301          POP B
        022272  303 254 022  JMP LOOP
END     022275  321          POP D
        022276  311          RET
        022277  000          NOP
RTN     022300  341          POP H           ML SUBROUTINE USED TO RETURN
        022301  301          POP B           AN IL CALL
        022302  345          PUSH H
        022303  311          RET
DONE    022304  032          LDAX D          ML SUBROUTINE USED TO FOR
                                             TERMINATION OF A BASIC LINE
        022305  023          INX D
        022306  376 040      CPI 'SP'
        022310  312 304 022  JZ DONE
        022313  033          DCX D
        022314  376 015      CPI 'CR'
        022316  310          RZ
        022317  303 022 030  JMP FIXDONE     MODIFICATION TO PERMIT
                                             MULTI-STATEMENT LINES
PRS     022322  032          LDAX D          ML SUBROUTINE USED TO
        022323  023          INX D           PRINT LITERAL
        022324  376 042      CPI '"'
        022326  310          RZ
        022327  376 015      CPI 'CR'
        022331  312 317 026  JZ ERR4         CR ENCOUNTERED IN LITERAL
        022334  315 026 022  CALL ASCOUT
        022337  303 322 022  JMP PRS
SPC     022342  041 360 033  LXI H ZONE      ML SUBROUTINE USED TO SPACE
                                             TO NEXT ZONE
LOOP    022345  076 040      MVI A 'SP'
        022347  357          RST OUTPUT
        022350  065          DCR M
        022351  302 345 022  JNZ LOOP
        022354  066 017      MVI M 15D
        022356  247          ANA A
        022357  311          RET
NLINE   022360  327          RST CRLF        ML SUBROUTINE USED TO ISSUE
                                             CR AND LF. ALSO RESETS ZONE
        022361  041 360 033  LXI H ZONE
        022364  066 017      MVI M 15D
        022366  227          SUB A
        022367  311          RET
        022370  000 000 000  NOP'S
        022373  000 000      NOP'S
NXT     022375  052 350 033  LHL CURLBL      ML SUBROUTINE USED TO TRANS-
                                             FER EXECUTION TO NEXT BASIC
                                             LINE. ALSO CHECKS FOR DIRECT
                                             EXECUTION(NO LABEL).
        023000  227          SUB A
        023001  274          CMP H
        023002  302 021 023  JNZ CONT
        023005  275          CMP L
        023006  302 021 023  JNZ CONT
FIN     023011  041 004 032  LXI H ILBGN+1   ML SUBROUTINE USED TO RE-
                                             TURN TO LINE COLLECT ROUTINE
        023014  301          POP B
        023015  343          XTHL
        023016  305          PUSH B
        023017  247          ANA A
        023020  311          RET
CONT    023021  023          INX D
        023022  032          LDAX D
        023023  147          MOV H,A
        023024  023          INX D
        023025  032          LDAX D
        023026  157          MOV L,A
        023027  042 350 033  SHL CURLBL
        023032  023          INX D
        023033  023          INX D
NXTX    023034  301          POP B
        023035  041 022 032  LXI H STMT
        023040  343          XTHL
        023041  305          PUSH B
        023042  247          ANA A
```

RTN: This ML routine terminates an IL CALL. Just prior to calling RTN, the 8080 stack looks like this:

Top – Return address to ILXQT
– Present IL program address
– Return IL program address
. . . .

RTN simply deletes the "present IL program address" from the stack producing:

Top – Return address to ILXQT
– Return IL program address
. . . .

ILXQT will then start execution at the IL instruction following the last IL CALL.

DONE: DONE is an ML routine that checks for proper line termination. After scanning over spaces (ASCII 040), DONE checks for the presence of a CR or dollar sign. If neither is encountered an error is signalled. If a CR is detected, a normal return to ILXQT is made. If a dollar sign is found, another BASIC command exists on the same line. In this case a branch is made to the NXT routine (see below) where interpretation is permitted to continue.

PRS: PRS is an ML routine used to output literals in PRint statements. A quotation mark is used by PRS to signal the end of a character string. A CR encountered before an ending quotation mark signals an error.

SPC: An ML routine used to space the output device to the next zone. The memroy location ZONE is decremented and a space is output until ZONE reaches zero. ZONE is then reset to 15 decimal and control returned to ILXQT.

NLINE: This ML routine issues a CR and LF to produce a new line on the output device. ZONE is also reset to 15 decimal.

NXT: The NXT routine handles the transition between one BASIC line and the next during the execution phase. As interpretation of a line finishes, the NXT routine checks to see if the line number stored in CURLBL is a zero. If so, a direct interpretation of a line is indicated. In this case, NXT sets up GETLINE as the next IL instruction. If not, interpretation is set to begin at the next BASIC line in the program area. CURLBL is updated to the new line number. In this way the error routine can report at which line an error occurred.

FIN: Actually, this routine is part of NXT. When an END statement is encountered, FIN is called and IL execution is directed to GET LINE. This essentially terminates BASIC execution.

```
TAG     ADDRESS I1  I2  I3  MNEMONIC      COMMENTS

        023043  311         RET
PSHAE   023044  305         PUSH B        SUBROUTINE USED TO PUSH
                                          H&L ONTO AE STACK.
        023045  104         MOV B,H
        023046  115         MOV C,L
        023047  052 361 033 LHL AELVL
        023052  160         MOV M,B
        023053  043         INX H
        023054  161         MOV M,C
        023055  043         INX H
        023056  042 361 033 SHL AELVL
        023061  301         POP B
        023062  175         MOV A,L
        023063  376 177     CPI 177
        023065  330         RC
        023066  303 322 026 JMP ERR5      AE TOO COMPLEX
```

PSHAE: A subroutine that pushes a binary value in HL onto the arithmetic stack (separate from 8080 stack). The AE stack pointer is stored at AELVL. Each time PSHAE is called, the pointer is incremented twice. The space reserved for the AE stack will allow 32 pushes without pops. Exceeding 32 causes an error condition.

```
POPAE   023071  305         PUSH B        SUBROUTINE USED TO POP TOP
        023072  052 361 033 LHL AELVL     OF AE STACK INTO H&L
        023075  053         DCX H
        023076  106         MOV B,M
        023077  053         DCX H
        023100  042 361 033 SHL AELVL
        023103  146         MOV H,M
        023104  175         MOV A,L
        023105  376 100     CPI 100
        023107  150         MOV L,B
        023110  301         POP B
        023111  320         RNC
        023112  303 325 026 JMP ERR6
```

POPAE: This subroutine pops a binary value off the AE stack into HL. AELVL is decremented twice. If AELVL is decremented below the space reserved for the AE stack, an error is indicated.

```
TWOCMP  023115  174         MOV A,H       SUBROUTINE USED TO TAKE
                                          2'S COMPLEMENT OF H&L
        023116  057         CMA
        023117  147         MOV H,A
        023120  175         MOV A,L
        023121  057         CMA
        023122  157         MOV L,A
        023123  043         INX H
        023124  311         RET
```

TWOCMP: The value in HL is two's complemented and placed back in HL.

```
PRN     023125  315 071 023 CALL POPAE    SUBROUTINE USED TO OUTPUT
                                          THE TOP OF THE AE STACK
        023130  174         MOV A,H
        023131  267         ORA A
        023132  362 147 023 JP CONT
        023135  315 115 023 CALL TWOCMP
        023140  076 055     MVI A '-'
        023142  345         PUSH H
        023143  315 026 022 CALL ASCOUT
        023146  341         POP H
CONT    023147  315 101 022 CALL IOUT
        023152  247         ANA A
        023153  311         RET
```

PRN: An ML routine that outputs the numeric value on the top of the AE stack. If a negative number is detected (most significant bit of H equal to 1), a minus sign is printed and TWOCMP called before IOUT prints HL. If the number is positive, IOUT is called directly.

```
FNDLBL  023154  345         PUSH H        SUBROUTINE USED TO GET
                                          ADDRESS OF LABEL IN H&L
        023155  052 352 033 LHL PRGSTRT
        023160  104         MOV B,H
        023161  115         MOV C,L
        023162  341         POP H
OVER    023163  012         LDAX B
        023164  274         CMP H
        023165  312 174 023 JZ NXT1
        023170  320         RNC
        023171  303 204 023 JMP NEW1
NXT1    023174  003         INX B
        023175  012         LDAX B
        023176  275         CMP L
        023177  312 220 023 JZ END
        023202  320         RNC
        023203  013         DCX B
NEW1    023204  003         INX B
        023205  003         INX B
        023206  012         LDAX B
        023207  201         ADDR C
        023210  117         MOV C,A
        023211  322 163 023 JNC OVER
        023214  004         INR B
        023215  303 163 023 JMP OVER
END1    023220  013         DCX B
        023221  140         MOV H,B
        023222  151         MOV L,C
        023223  311         RET
```

FNDLBL: The FNDLBL routine is used to search the BASIC program area for the label stored in HL. A linear search is begun at PRGSTRT. In order to speed the search, the stored line length is used to skip over line text. If the label is not found, return to the calling program is with the zero status bit reset. If the label is found, the location is placed in HL and the zero bit is set before return.

```
XFER    023224  315 071 023 CALL POPAE    SUBROUTINE USED TO TRANSFER
                                          EXECUTION TO LABEL ON TOP
                                          OF AE STACK.
        023227  315 154 023 CALL FNDLBL
        023232  353         XCHG
```

XFER: This ML routine transfers execution to the label stored on the top of the AE stack. The line number is popped off the AE stack into HL and FNDLBL is called. If upon return the zero bit is set, HL contains the location of the next line to be executed. A branch to the NXT routine completes the transfer process.

```
TAG     ADDRESS I1  I2  I3   MNEMONIC        COMMENTS

        023233  312 022 023  JZ CONT+1       "CONT" IN NXT SUBROUTINE
        023236  303 330 026  JMP ERR7        LABEL NOT FOUND
INNUM   023241  325          PUSH D          ML SUBROUTINE USED TO INPUT
                                             A NUMBER FROM TTY AND PLACE
                                             ON AE STACK.
        023242  076 077      MVI A '?'
        023244  315 026 022  CALL ASCOUT
        023247  076 040      MVI A 'SP'
        023251  315 026 022  CALL ASCOUT
        023254  000          NOP
        023255  315 000 020  CALL BUFIN
        023260  021 111 020  LXI D BUFSTRT
        023263  032          LDAX D
        023264  376 055      CPI '-'
        023266  041 000 000  LXI H Ø
        023271  312 312 023  JZ NEG1
        023274  315 331 020  CALL BIN
END1    023277  315 044 023  CALL PSHAE
        023302  076 040      MVI A 'SP'
        023304  357          RST OUTPUT
        023305  321          POP D
        023306  247          ANA A
        023307  311          RET
        023310  247 311      NOP'S
NEG1    023312  023          INX D
        023313  315 331 020  CALL BIN
        023316  315 115 023  CALL TWOCMP
        023321  303 277 023  JMP END1
```

INNUM: INNUM is an ML routine used to input a number from the input device, convert it to binary, and place it on the AE stack. The routine first outputs a question mark and a space. BUFIN is then called permitting the number to be inputted to the buffer. When a CR is detected, INNUM examines the buffer to see if a minus sign is present. If so, the program branches to NEG1. Otherwise, BIN is called to convert the number to binary and HL is pushed onto the AE stack. For a negative number, NEG1 makes the binary conversion, calls TWOCMP, and then pushes HL onto the AE stack.

```
TSTV    023324  032          LDAX D          ML SUBROUTINE USED TO TEST
                                             FOR VARIABLE AND PLACE
                                             ADDRESS ON AE STACK.
        023325  376 040      CPI 'SP'
        023327  023          INX D
        023330  312 324 023  JZ TSTV
        023333  033          DCX D
        023334  306 300      ADI 300
        023336  320          RNC
        023337  007          RLC
        023340  157          MOV L,A
        023341  046 024      MVI H 024
        023343  315 044 023  CALL PSHAE
        023346  067          STC
        023347  023          INX D
        023350  311          RET
```

TSTV: TSTV is an ML routine that determines whether the cursor points to a variable. First, TSTV scans over spaces. 300 is then added to the first non-space value. A no-carry condition will result if a shifted character or number is present. Return will be made to ILXQT and the next IL instruction executed. An ASCII letter (A-Z) will produce a carry and at the same time zero the two most significant bits. In this case, the address of the variable is computed by doubling A to form the lower half. The upper half is a constant, 024. With A moved to L and 024 in H, PSHAE is called to place the variable address on the AE stack. The carry is set before return to ILXQT causing the next IL to be skipped.

```
TSTN    023351  032          LDAX D          ML SUBROUTINE USED TO TEST
                                             FOR A NUMBER AND PLACE IT
                                             ON TOP OF AE STACK.
        023352  376 040      CPI 'SP'
        023354  023          INX D
        023355  312 351 023  JZ TSTN
        023360  033          DCX D
        023361  376 100      CPI 'A'-1
        023363  322 310 023  JNC END1
        023366  376 050      CPI '('
        023370  310          RZ
        023371  041 000 000  LXI H Ø
        023374  303 124 024  JMP CONT
#
# LOC 023377 IS A NOP. LOC 024000-024077 RESERVED FOR VARIABLES.
```

TSTN: This routine tests for the presence of a number in the BASIC text. Spaces are scanned over and the first non-space is checked to find if it is a letter variable. If so, return to ILXQT is made with the carry reset. If not, a check is made to see if the byte is an open parenthesis. If it is, return is made with no carry. If the character is not a letter or open parenthesis, it is assumed to be the first digit of a number. In this case, BIN is called, HL pushed on the AE stack, and return is made with the carry set.

```
DONEX   024100  032          LDAX D          ML SUBROUTINE SIMILAR TO
                                             DONE BUT NO PROVISION FOR
                                             TRANSFER.
        024101  023          INX D
        024102  376 040      CPI 'SP'
        024104  312 100 024  JZ DONEX
        024107  033          DCX D
        024110  376 015      CPI 'CR'
        024112  310          RZ
        024113  376 044      CPI '$'
        024115  310          RZ
        024116  303 314 026  JMP ERR3        DONE FAIL
        024121  000 000 000  NOP'S
        024124  315 331 020  CALL BIN
        024127  315 044 023  CALL PSHAE
        024132  311          RET
```

DONEX: DONEX is identical to DONE except that detection of a dollar sign does not lead to transfer in the NXT routine. Instead, it produces a simple return to ILXQT exactly as a CR would do.

```
IND     024133  315 071 023  CALL POPAE      ML SUBROUTINE USED TO
                                             REPLACE TOP OF AE STACK
                                             BY VARIABLE IT INDEXES.
        024136  106          MOV B,M
        024137  043          INX H
        024140  146          MOV H,M
        024141  150          MOV L,B
        024142  315 044 023  CALL PSHAE
        024145  247          ANA A
        024146  311          RET
```

IND: This ML routine pops the AE stack to bring the address of a variable into HL. The value of the variable is then obtained and pushed onto the AE stack.

```
STORE   024147  315 071 023  CALL POPAE      ML SUBROUTINE USED TO
                                             PLACE TOP OF STACK INTO
                                             VARIABLE INDEXED.
```

STORE: An ML routine that first pops a variable address off the AE stack and places it in BC. A numeric value is then popped off the AE stack and placed at the variable address in BC.

```
TAG      ADDRESS I1  I2  I3   MNEMONIC        COMMENTS

         024152  114          MOV C,H
         024153  105          MOV B,L
         024154  315 071 023  CALL POPAE
         024157  160          MOV M,B
         024160  043          INX H
         024161  161          MOV M,C
         024162  247          ANA A
         024163  311          RET
#
# LOC 024164-024177 RESERVED FOR BASIC SUBROUTINE STACK
#
ADD      024200  315 071 023  CALL POPAE      ML SUBROUTINE USED TO ADD
                                              TWO TOPMOST ELEMENTS ON
                                              STACK.
         024203  104          MOV B,H
         024204  115          MOV C,L
         024205  315 071 023  CALL POPAE
         024210  011          DAD B
         024211  315 044 023  CALL PSHAE
         024214  247          ANA A
         024215  311          RET
```

ADD: An ML routine used to perform signed addition on the two top elements of the AE stack, this sum being placed back on the AE stack.

```
SUB      024216  315 071 023  CALL POPAE      ML SUBROUTINE USED TO FIND
                                              THE DIFFERENCE OF THE TWO
                                              TOPMOST ELEMENTS OF THE AE
                                              STACK.
         024221  315 115 023  CALL TWOCMP
         024224  104          MOV B,H
         024225  115          MOV C,L
         024226  315 071 023  CALL POPAE
         024231  011          DAD B
         024232  315 044 023  CALL PSHAE
         024235  247          ANA A
         024236  311          RET
         024237  000          NOP
```

SUBTRACT: An ML routine used to perform signed subtraction on the two top elements of the stack. After the first value is popped off the stack, the two's complement is taken to produce the subtrahend. From that point on, the routine is similar to the ADD routine.

```
MUL      024240  325          PUSH D          ML SUBROUTINE USED TO MULTI-
                                              PLY THE TWO TOPMOST ELEMENTS
                                              OF AE STACK
         024241  006 000      MVI B Ø
         024243  315 071 023  CALL POPAE
         024246  174          MOV A,H
         024247  267          ORA A
         024250  374 301 024  CM NINOX
         024253  353          XCHG
         024254  315 071 023  CALL POPAE
         024257  174          MOV A,M
         024260  267          ORA A
         024261  374 301 024  CM NINOX
         024264  315 306 024  CALL MULT
         024267  005          DCR B
         024270  314 115 023  CZ TWOCMP
         024273  315 044 023  CALL PSHAE
         024276  321          POP D
         024277  247          ANA A
         024300  311          RET
```

MUL: This routine performs signed multiplication of the two top elements of the AE stack. MUL essentially takes care of sign determination while another routine, MULT, actually performs the multiplication. Register B is used to logically determine if either or both of the factors are negative. B is originally set to zero and then incremented in NINOX once for each negative factor. In addition, each negative factor is two's complemented to produce the corresponding positive value. At the end of MUL if B = 1, then the product should be negative. In this case two's complement routine is called. All other values of B indicate a positive product.

```
NINOX    024301  004          INR B
         024302  315 115 023  CALL TWOCMP
         024305  311          RET
```

NINOX: A routine used with MUL and DIV is sign determination (see MUL).

```
MULT     024306  305          PUSH B          SUBROUTINE MULTIPLIES H&L
                                              BY D&E ANSWER IN H&L
         024307  104          MOV B,H
         024310  115          MOV C,L
         024311  041 000 000  LXI H Ø
         024314  076 021      MVI A 17D
         024316  062 363 033  STA INDX
LOOP     024321  170          MOV A,B
         024322  037          RAR
         024323  107          MOV B,A
         024324  171          MOV A,C
         024325  037          RAR
         024326  117          MOV C,A
         024327  322 333 024  JNC NXT1
         024332  031          DAD D
NXT1     024333  174          MOV A,H
         024334  037          RAR
         024335  147          MOV H,A
         024336  175          MOV A,L
         024337  037          RAR
         024340  157          MOV L,A
         024341  072 363 033  LDA INDX
         024344  075          DCR A
         024345  312 356 024  JZ END1
         024350  062 363 033  STA INDX
         024353  303 321 024  JMP LOOP
END1     024356  140          MOV H,B
         024357  151          MOV L,C
         024360  301          POP B
```

MULT: This routine multiplies the two 16-bit numbers in HL and DE. The product is shifted into BC as multiplication takes place. This routine is a little unconventional and may require some close study to understand.

```
TAG     ADDRESS I1  I2  I3   MNEMONIC         COMMENTS

        024361  311          RET
DIV     024362  325          PUSH D           ML SUBROUTINE USED TO DI-
                                              VIDE TWO TOPMOST ELEMENTS
                                              OF AE STACK.
        024363  006 000      MVI B Ø
        024365  315 071 023  CALL POPAE
        024370  174          MOV A,H
        024371  267          ORA A
        024372  374 301 024  CM NINOX
        024375  353          XCHG
        024376  315 071 023  CALL POPAE
        025001  174          MOV A,H
        025002  267          ORA A
        025003  374 301 024  CM NINOX
        025006  353          XCHG
        025007  227          SUB A
        025010  274          CMP H
        025011  302 020 025  JNZ CONT
        025014  275          CMP L
        025015  312 333 026  JZ ERR8          DIVIDE BY ZERO
CONT    025020  315 026 025  CALL DIVD
        025023  303 267 024  JMP NINOX-10
DIVD    025026  305          PUSH B
        025027  006 001      MVI B 1
LOOP    025031  174          MOV A,H
        025032  346 100      ANI 01000000B
        025034  302 044 025  JNZ OUT
        025037  051          DAD H
        025040  004          INR B
        025041  303 031 025  JMP LOOP
OUT     025044  170          MOV A,B
        025045  062 363 033  STA INDX
        025050  104          MOV B,H
        025051  115          MOV C,L
        025052  041 000 000  LXI H Ø
OVER    025055  173          MOV A,E
        025056  221          SUB C
        025057  137          MOV E,A
        025060  172          MOV A,D
        025061  230          SBB B
        025062  127          MOV D,A
        025063  322 117 025  JNC DIVØ
        025066  173          MOV A,E
        025067  201          ADDR C
        025070  137          MOV E,A
        025071  172          MOV A,D
        025072  210          ADC B
        025073  127          MOV D,A
        025074  051          DAD H
        025075  072 363 033  LDA INDX
        025100  075          DCR A
        025101  312 115 025  JZ END
CONT    025104  062 363 033  STA INDX
        025107  353          XCHG
        025110  051          DAD H
        025111  353          XCHG
        025112  303 055 025  JMP OVER
END     025115  301          POP B
        025116  311          RET
DIVØ    025117  051          DAD H
        025120  043          INX H
        025121  072 363 033  LDA INDX
        025124  075          DCR A
        025125  312 115 025  JZ END
        025130  303 104 025  JMP CONT
NEG     025133  315 071 023  CALL POPAE       ML SUBROUTINE USED TO NE-
                                              GATE TOP OF AE STACK.
        025136  315 115 023  CALL TWOCMP
        025141  315 044 023  CALL PSHAE
        025144  247          ANA A
        025145  311          RET
        025146  000 000 000  NOP'S
CMPR    025151  325          PUSH D           ML SUBROUTINE USED TO COM-
                                              PARE TWO TOPMOST ELEMENTS
                                              OF AE STACK.
        025152  315 071 023  CALL POPAE
        025155  353          XCHG
        025156  315 071 023  CALL POPAE
        025161  345          PUSH H
        025162  315 071 023  CALL POPAE
        025165  174          MOV A,H
        025166  346 200      ANI 10000000B
        025170  302 262 025  JNZ CONT1
        025173  172          MOV A,D
        025174  346 200      ANI 10000000B
        025176  302 227 025  JNZ B:4
CONT2   025201  174          MOV A,H
        025202  272          CMP D
        025203  312 214 025  JZ OVR
```

DIV: This ML routine is basically similar to the MUL routine in that it handles sign determination for division. The quotient of the two top elements of the stack is taken with the first popped off the stack treated as the divisor. Integer division actually takes place in a called routine, DIVD.

DIVD: The contents of DE is divided by the contents of HL in this routine. The divisor (HL) is left justified before division actually begins. The number of left shifts required for this determines the number of shifted subtractions used in the binary division process. A check is made for division by zero and an error is reported if this is the case. The quotient is developed in HL.

NEG: An ML routine used to negate the top element of the AE stack. The two's complement routine is used and the result is placed back on the stack.

CMPR: At the time CMPR is called, the AE stack has at least three elements consisting of the first expression value, the logical operator address code (labeled 0:, 1:, 2:, etc. on listing), and a second expression value. Testing is performed on the two expression values and A is made a 0, 1, or 4 depending on the numerical comparison:

Expression values equal	*A = 0*
First expression greater than second	*A = 1*
Second expression greater than first	*A = 4*

The logical operator address code then sends the program to one of 6 testing subroutines (labelled 0:, 1:, 2:, etc.) where a check is made on A to see if the condition is true or false. If the zero bit is set upon return then a true condition exists; otherwise, the condition is false. For a true condition, execution is set to continue at the next statement following the IF. A false cause execution of the next numbered line.

```
TAG     ADDRESS I1  I2  I3   MNEMONIC        COMMENTS

        025206  322 227 025  JNC B:4
        025211  303 224 025  JMP B:1
OVR     025214  175          MOV A,L
        025215  273          CMP E
        025216  312 232 025  JZ B:Ø
        025221  322 227 025  JNC B:4
B:1     025224  076 001      MVI A 1D
        025226  041          SPECIAL
B:4     025227  076 004      MVI A 4D
        025231  041          SPECIAL
B:Ø     025232  076 000      MVI A ØD
        025234  341          POP H
        025235  021 242 025  LXI D ALPHA
        025240  325          PUSH D
        025241  351          PCHL
ALPHA   025242  312 260 025  JZ TRUE
FALSE   025245  321          POP D
LOOP    025246  032          LDAX D
        025247  376 015      CPI 'CR'
        025251  312 375 022  JZ NXT          "NXT" SUBROUTINE
        025254  023          INX D
        025255  303 246 025  JMP LOOP
TRUE    025260  321          POP D
        025261  311          RET
CONT1   025262  172          MOV A,D
        025263  346 200      ANI 10000000B
        025265  302 201 025  JNZ CONT2
        025270  303 224 025  JMP B:1
Ø:      025273  376 000      CPI ØD
        025275  311          RET
1:      025276  376 001      CPI 1D
        025300  311          RET
2:      025301  376 000      CPI ØD
        025303  310          RZ
        025304  376 001      CPI 1D
        025306  311          RET
3:      025307  376 001      CPI 1D
        025311  310          RZ
        025312  376 004      CPI 4D
        025314  311          RET
4:      025315  376 004      CPI 4D
        025317  311          RET
5:      025320  376 000      CPI ØD
        025322  310          RZ
        025323  376 004      CPI 4D
        025325  311          RET
LITØ    025326  056 273      MVI L 273
        025330  001          SPECIAL
LIT1    025331  056 276      MVI L 276
        025333  001          SPECIAL
LIT2    025334  056 301      MVI L 301
        025336  001          SPECIAL
LIT3    025337  056 307      MVI L 307
        025341  001          SPECIAL
LIT4    025342  056 315      MVI L 315
        025344  001          SPECIAL
LIT5    025345  056 320      MVI L 320
        025347  046 025      MVI H 024
        025351  315 044 023  CALL PSHAE
        025354  247          ANA A
        025355  311          RET
PSHSBR  025356  305          PUSH B          SUBROUTINE USED TO SAVE
                                             BASIC SUBROUTINE RETURN
                                             ADDRESS.
        025357  104          MOV B,H
        025360  115          MOV C,L
        025361  052 364 033  LHL SBRLVL
        025364  160          MOV M,B
        025365  043          INX H
        025366  161          MOV M,C
        025367  043          INX H
        025370  042 364 033  SHL SBRLVL
        025373  301          POP B
        025374  175          MOV A,L
        025375  376 177      CPI 177
        025377  330          RC
        026000  303 336 026  JMP ERR9        NESTING TOO DEEP
POPSBR  026003  305          PUSH B          SUBROUTINE USED TOO RETRIEVE
                                             BASIC SUBROUTINE ADDRESS.
        026004  052 364 033  LHL SBRLVL
        026007  053          DCX H
        026010  106          MOV B,M
        026011  053          DCX H
        026012  042 364 033  SHL SBRLVL
        026015  146          MOV H,M
        026016  175          MOV A,L
        026017  376 164      CPI 164
        026021  150          MOV L,B
        026022  301          POP B
```

LIT0 through LIT5: These ML routines are used to put the logical operator address on the AE stack during execution of an IF statement. See CMPR.

PSHSBR: A routine used to place the return address of GOSUB on the subroutine stack. The subroutine stack is separate from the AE stack. SBRLVL is a pair of locations used to keep track of the level of subroutining.

POPSBR: A routine used to pop the GOSUB return address off the subroutine stack.

```
TAG      ADDRESS I1  I2  I3   MNEMONIC         COMMENTS

         026023  320          RNC
         026024  303 341 026  JMP ERR10        TOO MANY RETURN STATEMENTS
SAVE     026027  142          MOV H,D          ML SUBROUTINE USED TO PLACE
                                               RETURN ADDRESS ON SBR STACK.
         026030  153          MOV L,E
         026031  315 356 025  CALL PSHSBR
         026034  247          ANA A
         026035  311          RET
```

SAVE: This ML routine places the GOSUB return address on the subroutine stack using PSHSBR.

```
RSTR     026036  315 003 026  CALL POPSBR      ML SUBROUTINE USED TO RE-
                                               TRIEVE RETURN ADDRESS FROM
                                               SBR STACK.
         026041  353          XCHG
         026042  247          ANA A
         026043  311          RET
```

RSTR: Upon execution of a RET statement, this ML routine uses POPSBR to fetch the return address off the subroutine stack.

```
SPCONE   026044  076 040      MVI A 'SP'       ML SUBROUTINE USED TO OUT-
                                               PUT ONE SPACE TO TTY.
         026046  315 026 022  CALL ASCOUT
         026051  247          ANA A
         026052  311          RET
```

SPCONE: ML routine that issues one space on the output device. This is the execution routine for a semicolon in the PR statement.

```
INIT     026053  000 000 000  NOP'S            ML SUBROUTINE USED TO
                                               INITIALIZE BASIC SYSTEM.
         026056  041 077 026  LXI H STRT
         026061  001 350 033  LXI B CURLBL
LOOP     026064  176          MOV A,M
         026065  002          STAX B
         026066  175          MOV A,L
         026067  376 130      CPI 130
         026071  310          RZ
         026072  003          INX B
         026073  043          INX H
         026074  303 064 026  JMP LOOP
         026077  000          DATA
         026100  000 000 034  DATA
         026103  001 034 000  DATA
         026106  040 017 100  DATA
         026111  030 000 164  DATA
         026114  024 377 057  DATA
         026117  000 000 056  DATA
         026122  241 051 321  DATA
         026125  377 057 377  DATA
         026130  377          DATA
```

INIT: This ML routine initializes the TBX system when a NEW statement is executed. The program area is preset to that a new program can be entered.

```
XINIT    026131  041 100 030  LXI H            ML SUBROUTINE USED TO
                                               PREPARE SYSTEM FOR EXECUTION
         026134  042 361 033  SHL AELVL
         026137  041 164 024  LXI H
         026142  042 364 033  SHL SBRLVL
         026145  315 020 027  CALL INIARY
         026150  052 352 033  LHL PRGSTRT
         026153  126          MOV D,M
         026154  043          INX H
         026155  136          MOV E,M
         026156  353          XCHG
         026157  000          NOP
         026160  042 350 033  SHL CURLBL
         026163  023          INX D
         026164  023          INX D
         026165  247          ANA A
         026166  311          RET
```

XINIT: When RUN is typed, certain locations in TBX must be initialized. The XINIT routine performs the following tasks:

1. *AE and subroutine stacks are emptied;*
2. *The array storage area is preset at zero length; and*
3. *The label number of the first statement to be executed is placed in CURLBL.*

```
#
# LOC 026167-026204 IS UNUSED.
#
LBLOUT   026205  345          PUSH H           SUBROUTINE USED TO OUTPUT
                                               LABEL(NO ZERO SUPPRESSION)
         026206  325          PUSH D
         026207  305          PUSH B
         026210  353          XCHG
         026211  016 377      MVI C 377
         026213  303 107 022  JMP IOUT+6
```

LBLOUT: This routine is called by LIST to output a label number of a TBX statement. IOUT is used with C preset to 377 octal preventing zero suppression.

```
ERRMAIN  026216  000          NOP
         026217  000          NOP
         026220  327          RST CRLF
         026221  000 000 000  NOP'S
         026224  076 105      MVI A 'E'
         026226  357          RST OUTPUT
         026227  076 122      MVI A 'R'
         026231  357          RST OUTPUT
         026232  357          RST OUTPUT
         026233  076 040      MVI A 'SP'
         026235  357          RST OUTPUT
         026236  046 000      MVI H Ø
         026240  000 000 000  NOP'S
         026243  315 101 022  CALL IOUT
         026246  052 350 033  LHL CURLBL
         026251  076 040      MVI A 'SP'
         026253  357          RST OUTPUT
         026254  315 205 026  CALL LBLOUT
         026257  016 010      MVI C 8D
```

ERRMAIN: This routine is used to process an error condition. "ERR" is outputted followed by the error number and the CURLBL. Entry to ERRMAIN is made through ERR1, ERR2, etc., where L is set to the error number desired.

```
TAG       ADDRESS I1  I2  I3   MNEMONIC         COMMENTS

          026261  041 357 033  LXI H            PARTIAL REINITIALIZATION
          026264  021 106 026  LXI D            SEQUENCE
LOOP      026267  032          LDAX D
          026270  167          MOV M,A
          026271  015          DCR C
          026272  302 267 026  JNZ LOOP
          026275  041 002 032  LXI H
          026300  061 077 002  LXI SP
          026303  303 257 021  JMP ILXQT
ERRLIST,  026306  056 001      MVI L 1D
 ERR1
          026310  001          SPECIAL
ERR2      026311  056 002      MVI L 2D
          026313  001          SPECIAL
ERR3      026314  056 003      MVI L 3D
          026316  001          SPECIAL
ERR4      026317  056 004      MVI L 4D
          026321  001          SPECIAL
ERR5      026322  056 005      MVI L 5D
          026324  001          SPECIAL
ERR6      026325  056 006      MVI L 6D
          026327  001          SPECIAL
ERR7      026330  056 007      MVI L 7D
          026332  001          SPECIAL
ERR8      026333  056 010      MVI L 8D
          026335  001          SPECIAL
ERR9      026336  056 011      MVI L 9D
          026340  001          SPECIAL
ERR10     026341  056 012      MVI L 10D
          026343  001          SPECIAL
ERR11     026344  056 013      MVI L 11D
          026346  001          SPECIAL
ERR12     026347  056 014      MVI L 12D
          026351  001          SPECIAL
ERR13     026352  056 015      MVI L 13D
          026354  001          SPECIAL
ERR14     026355  056 016      MVI L 14D
          026357  001          SPECIAL
ERR15     026360  056 017      MVI L 15D
          026362  001          SPECIAL
ERR16     026363  056 020      MVI L 16D
          026365  303 216 026  JMP ERRMAIN
#
# LOC 026370-027017 IS UNUSED.
#
INIARY    027020  076 012      MVI A 'LF'       ARRAY INITIALIZATION SUB-
                                                ROUTINE
          027022  357          RST OUTPUT
          027023  052 115 026  LHL
          027026  042 366 033  SHL ARYSTRT
          027031  311          RET
DIM2      027032  325          PUSH D           ML SUBROUTINE USED TO
                                                SET UP TWO DIMENSIONAL
                                                ARRAYS.
          027033  315 071 023  CALL POPAE
          027036  353          XCHG
          027037  315 071 023  CALL POPAE
          027042  104          MOV B,H
          027043  115          MOV C,L
          027044  315 044 023  CALL PSHAE
          027047  353          XCHG
          027050  315 044 023  CALL PSHAE
          027053  321          POP D
          027054  305          PUSH B
          027055  315 240 024  CALL MULT
          027060  315 071 023  CALL POPAE
          027063  303 072 027  JMP CONT
DIM1      027066  315 071 023  CALL POPAE       ML SUBROUTINE USED TO SET UP
                                                ONE DIMENSIONAL ARRAYS.
          027071  345          PUSH H
CONT      027072  051          DAD H
          027073  104          MOV B,H
          027074  115          MOV C,L
          027075  052 366 033  LHL ARYSTRT
          027100  175          MOV A,L
          027101  221          SUB C
          027102  117          MOV C,A
          027103  174          MOV A,H
          027104  230          SBB B
          027105  107          MOV B,A
          027106  013          DCX B
          027107  052 354 033  LHL PRGEND
          027112  274          CMP H
          027113  302 120 027  JNZ CONT1
          027116  171          MOV A,C
          027117  275          CMP L
CONT1     027120  332 360 026  JC ERR15
          027123  140          MOV H,B
          027124  151          MOV L,C
          027125  301          POP B
          027126  160          MOV M,B
```

INIARY: A subroutine called by XINIT to preset the array area of memory.

DIM1 and DIM2: These ML routines are used to set up array storage as a result of execution of a DIMension statement. DIM2 handles two dimensional arrays while DIM1 handles the one-dimensional arrays. At the time these routines are called, the array dimensions are on top of the AE stack. MULT and double register addition are used to calculate memory needed for a given array dimension. The variable associated with the array name is used to hold the location of the beginning of that array.

```
TAG      ADDRESS I1  I2  I3  MNEMONIC        COMMENTS

         027127  053         DCX H
         027130  161         MOV M,C
         027131  104         MOV B,H
         027132  115         MOV C,L
         027133  042 366 033 SHL ARYSTRT
         027136  315 071 023 CALL POPAE
         027141  161         MOV M,C
         027142  043         INX H
         027143  160         MOV M,B
         027144  247         ANA A
         027145  311         RET
ARRAY1   027146  315 071 023 CALL POPAE      ML SUBROUTINE USED TO GET
                                             THE ADDRESS OY A ONE DIMEN-
                                             SIONAL ARRAY VARIABLE.
         027151  053         DCX H
         027152  051         DAD H
         027153  104         MOV B,H
         027154  115         MOV C,L
         027155  315 071 023 CALL POPAE
         027160  011         DAD B
         027161  315 044 023 CALL PSHAE
         027164  247         ANA A
         027165  311         RET
ARRAY2   027166  315 071 023 CALL POPAE      ML SUBROUTINE USED TO GET
                                             THE ADDRESS OF A TWO DIMEN-
                                             SIONAL ARRAY VARIABLE.
         027171  053         DCX H
         027172  315 044 023 CALL PSHAE
         027175  052 370 033 LHL ATEMP
         027200  315 044 023 CALL PSHAE
         027203  315 240 024 CALL MULT
         027206  315 200 024 CALL ADD
         027211  303 146 027 JMP ARRAY1
```

ARRAY1 and ARRAY2: These ML routines are used to calculate the address of an array variable. The array position values are on the AE stack at the time these routines are called. MULT and double register addition are used in the calculation.

```
TSTA     027214  032         LDAX D          ML SUBROUTINE USED TO TEST
                                             FOR AN ARRAY.
         027215  023         INX D
         027216  376 040     CPI 'SP'
         027220  312 214 027 JZ TSTA
         027223  033         DCX D
         027224  306 300     ADI 300
         027226  320         RNC
         027227  007         RLC
         027230  117         MOV C,A
         027231  023         INX D
         027232  032         LDAX D
         027233  376 050     CPI '('
         027235  312 243 027 JZ CONT
         027240  033         DCX D
         027241  247         ANA A
         027242  311         RET
CONT     027243  151         MOV L,C
         027244  046 024     MVI H 024
         027246  116         MOV C,M
         027247  043         INX H
         027250  146         MOV H,M
         027251  151         MOV L,C
         027252  116         MOV C,M
         027253  043         INX H
         027254  106         MOV B,M
         027255  043         INX H
         027256  315 044 023 CALL PSHAE
         027261  140         MOV H,B
         027262  151         MOV L,C
         027263  042 370 033 SHL ATEMP
         027266  067         STC
         027267  311         RET
         027270  000 000 000 NOP'S
#
# LOC 027273-027304 IS UNUSED.
#
```

TSTA: An ML routine used to test TBX text for the presence of an array variable. If a letter is immediately followed by an open parenthesis, then an array is indicated. Otherwise, an ordinary variable is present.

```
FOR      027305  325         PUSH D          ML SUBROUTINE USED TO SET UP
LOOP     027306  023         INX D           FOR LOOP.
         027307  032         LDAX D
         027310  376 015     CPI 'CR'
         027312  302 064 030 JNZ FIXFOR
CONT     027315  353         XCHG
         027316  315 044 023 CALL PSHAE
         027321  321         POP D
         027322  247         ANA A
         027323  311         RET
```

FOR: When a FOR statement is executed, this ML routine places the address of the next statement following the FOR on the AE stack.

```
NEXT     027324  325         PUSH D          ML SUBROUTINE USED TO CHECK
         027325  315 071 023 CALL POPAE      END OF FOR LOOP.
         027330  345         PUSH H
         027331  116         MOV C,M
         027332  043         INX H
         027333  106         MOV B,M
         027334  315 071 023 CALL POPAE
         027337  353         XCHG
         027340  315 071 023 CALL POPAE
```

NEXT: This ML routine is used to process a NXT instruction. During its execution the following tasks are performed:

1. *The index variable is incremented;*
2. *A check is made to see if the variable limit has been exceeded;*
3. *If so, the next TBX instruction is executed; and*
4. *If not, execution is returned to the statement following the appropriate FOR statement.*

```
TAG      ADDRESS I1  I2  I3   MNEMONIC       COMMENTS

         027343  003          INX B
         027344  172          MOV A,D
         027345  270          CMP B
         027346  302 361 027  JNZ CONT
         027351  173          MOV A,E
         027352  271          CMP C
         027353  322 361 027  JNZ CONT
         027356  303 006 030  JMP CONT1
CONT     027361  345          PUSH H
         027362  315 044 023  CALL PSHAE
         027365  341          POP H
         027366  353          XCHG
         027367  315 044 023  CALL PSHAE
         027372  341          POP H
         027373  315 044 023  CALL PSHAE
         027376  140          MOV H,B
         027377  151          MOV L,C
         030000  315 044 023  CALL PSHAE
         030003  341          POP H
         030004  247          ANA A
         030005  311          RET
CONT1    030006  341          POP H
         030007  315 044 023  CALL PSHAE
         030012  140          MOV H,B
         030013  151          MOV L,C
         030014  315 044 023  CALL PSHAE
         030017  321          POP D
         030020  247          ANA A
         030021  311          RET
FIXDONE  030022  376 044      CPI '$'
         030024  302 314 026  JNZ
         030027  303 033 023  JMP
TSTF     030032  032          LDAX D         ML SUBROUTINE USED TO TEST
                                             FOR FUNCTION.
```

TSTF: A test is performed by this ML routine to check for the presence of a function in the TBX text. The function is recognized by the occurrence of two letters in sequence, i.e., RN for the random number function.

```
         030033  376 040      CPI 'SP'
         030035  023          INX D
         030036  312 032 030  JZ TSTF
         030041  033          DCX D
         030042  306 300      ADI 300
         030044  320          RNC
         030045  325          PUSH D
         030046  023          INX D
         030047  032          LDAX D
         030050  306 300      ADI 300
         030052  321          POP D
         030053  320          RNC
         030054  376 015      CPI 'CR'
         030056  310          RZ
         030057  376 040      CPI 'SP'
         030061  310          RZ
         030062  067          STC
         030063  311          RET
FIXFOR   030064  376 044      CPI '$'
         030066  312 315 027  JZ CONT        IN 'FOR' ROUTINE
         030071  303 306 027  JMP LOOP       IN 'FOR' ROUTINE
#
# LOC 030074-030077 IS UNUSED. LOC 030100-030177 RESERVED FOR AE STACK.
# LOC 030200-030203 IS UNUSED.
#
RNDM     030204  041 375 033  LXI H SEED4    ML SUBROUTINE USED TO GEN-
                                             ERATE RANDOM NUMBERS
```

RNDM: A random number generator based on a technique by Jim Parker appearing in "The Computer Hobbyist" (Vol. 1, No. 5). The routine returns only when a value between 0 and 10,000 decimal is sensed.

```
         030207  006 010      MVI B 8D
LOOP     030211  176          MOV A,M
         030212  007          RLC
         030213  007          RLC
         030214  007          RLC
         030215  256          XOR M
         030216  027          RAL
         030217  027          RAL
         030220  055          DCR L
         030221  055          DCR L
         030222  055          DCR L
         030223  176          MOV A,M
         030224  027          RAL
         030225  167          MOV M,A
         030226  054          INR L
         030227  176          MOV A,M
         030230  027          RAL
         030231  167          MOV A,M
         030232  054          INR L
         030233  176          MOV A,M
         030234  027          RAL
         030235  167          MOV M,A
         030236  054          INR L
         030237  176          MOV A,M
         030240  027          RAL
         030241  167          MOV M,A
         030242  005          DCR B
         030243  302 211 030  JNZ LOOP
         030246  052 374 033  LHL SEED3
```

```
TAG       ADDRESS I1  I2  I3   MNEMONIC         COMMENTS

          030251  174          MOV A,H
          030252  346 077      ANI 00111111B
          030254  147          MOV H,A
          030255  376 047      CPI 047
          030257  312 272 030  JZ CONT1
CONT2     030262  322 204 030  JNC RNDM
          030265  315 044 023  CALL PSHAE
          030270  077          CMC
          030271  311          RET
CONT1     030272  175          MOV A,L
          030273  376 020      CPI 020
          030275  303 262 030  JMP CONT2
TAB       030300  315 071 023  CALL POPAE       ML SUBROUTINE USED TO PRO-
                                                DUCE TAB FUNCTION.
```

TAB: This ML routine spaces over an amount equal to the value stored on the top of the AE stack.

```
          030303  105          MOV B,A
LOOP      030304  076 040      MVI A 'SP'
          030306  315 026 022  CALL ASCOUT
          030311  005          DCR B
          030312  302 304 030  JNZ LOOP
          030315  063 063 063  3XINX SP
          030320  063 063 063  3XINX SP
          030323  301          POP B
          030324  341          POP H
          030325  043          INX H
          030326  043          INX H
          030327  345          PUSH H
          030330  305          PUSH B
          030331  073 073 073  3XDCX SP
          030334  073 073 073  3XDCX SP
          030337  311          RET
MEMTEST   030340  072 367 033  LDA ASTRT(H)     SUBROUTINE USED TO TEST FOR
                                                MEMORY DEPLETION.
```

MEMTEST: A routine used to test for memory depletion. If array storage area overlaps the program area, an error is reported.

```
          030343  274          CMP H
          030344  312 360 030  JZ CONT
          030347  332 360 026  JPC ERR15
END       030352  042 354 033  SHL PRGEND
          030355  311          RET
          030356  000 000      NOP'S
CONT      030360  072 366 033  LDA ASTRT(L)
          030363  326 000      SUI Ø
          030365  275          CMP L
          030366  322 352 030  JNC END
          030371  303 360 026  JMP ERR15
#
# LOC 030374-030377 IS NOT USED.
#
SIZE      031000  052 354 033  LHL PRGEND       ML SUBROUTINE USED TO DETER-
          031003  053          DCX H            MINE SIZE OF PROGRAM AND
          031004  104          MOV B,H          AMOUNT OF MEMORY REMAINING.
```

SIZE: This ML routine computes the amount of memory being used and the amount left.

```
          031005  115          MOV C,H
          031006  052 376 033  LHL MEMEND
          031011  011          DAD B
          031012  345          PUSH H
          031013  052 366 033  LHL ASTRT
          031016  104          MOV B,H
          031017  115          MOV C,L
          031020  052 352 033  LHL PRGSTRT
          031023  011          DAD B
          031024  301          POP B
          031025  315 060 031  CALL DIFF
          031030  315 101 022  CALL IOUT
          031033  076 040      MVI A 'SP'
          031035  357          RST OUTPUT
          031036  052 366 033  LHL ASTRT
          031041  104          MOV B,H
          031042  115          MOV C,L
          031043  052 354 033  LHL PRGEND
          031046  053          DCX H
          031047  315 060 031  CALL DIFF
          031052  315 101 022  CALL IOUT
          031055  327          RST CRLF
          031056  247          ANA A
          031057  311          RET
DIFF      031060  171          MOV A,C
          031061  225          SUB L
          031062  157          MOV L,A
          031063  170          MOV A,B
          031064  234          SBB H
          031065  147          MOV H,A
          031066  311          RET
LSTØ      031067  052 352 033  LHL PRGSTRT      ML SUBROUTINE USED TO LIST
                                                ENTIRE BASIC PROGRAM.
```

LST0, LST1, and LST2: These three routines set up LSTSTRT and LSTEND so that when LIST is called, only the required lines will be listed. LST0 is called if the entire program is to be listed. LST1 is called to list only one line. LST2 sets up a listing between two given lines.

```
          031072  042 304 033  SHL LSTSTRT
          031075  052 354 033  LHL PRGEND
          031100  042 306 033  SHL LSTEND
          031103  247          ANA A
          031104  311          RET
LST1      031105  315 165 031  CALL FIND        ML SUBROUTINE USED TO LIST
                                                ONE LINE.
          031110  042 304 033  SHL LSTSTRT
```

```
TAG     ADDRESS I1  I2  I3   MNEMONIC          COMMENTS

        031113  043          INX H
        031114  043          INX H
        031115  076 015      MVI A 'CR'
LOOP    031117  043          INX H
        031120  276          CMP M
        031121  302 117 031  JNZ LOOP
        031124  043          INX H
        031125  043          INX H
        031126  042 306 033  SHL LSTEND
        031131  247          ANA A
        031132  311          RET
        031133  000          NOP
LST2    031134  315 165 031  CALL FIND        ML SUBROUTINE USED TO LIST
                                              BETWEEN TWO LINE NUMBERS.
        031137  043          INX H
        031140  043          INX H
        031141  076 015      MVI A 'CR'
LOOP    031143  043          INX H
        031144  276          CMP M
        031145  302 143 031  JNZ LOOP
        031150  043          INX H
        031151  043          INX H
        031152  042 306 033  SHL LSTEND
        031155  315 165 031  CALL FIND
        031160  042 304 033  SHL LSTEND
        031163  247          ANA A
        031164  311          RET
FIND    031165  315 071 023  CALL POPAE
        031170  315 154 023  CALL FINDLBL
        031173  310          RZ
        031174  303 330 026  JMP ERR7
#
# LOC 031177-031277 IS UNUSED.
#
FN      031300  231 310      TST FNO 'RN'
        031302  122 316
        031304  330 204      RANDOM
        031306  322 300      RTN
FNO     031310  232 330      TST S17 'TB'
        031312  124 302
        031314  132 343      CALL EXPR
        031316  330 300      TAB
        031320  322 300      RTN
S8C     031322  231 331      TST S8B 'CR'
        031324  215
        031325  322 360      NLINE
        031327  322 375      NXT
S8B     031331  232 210      TST S8 '$'
        031333  244
        031334  322 360      NLINE
        031336  323 034      NXTX
S14     031340  231 351      TST S14A 'CR'
        031342  215
        031343  331 067      LISTO
        031345  322 213      LST
        031347  322 375      NXT
S14A    031351  132 343      CALL EXPR
        031353  231 366      TST S14B ','
        031355  254
        031356  132 343      CALL EXPR
        031360  331 134      LIST2
        031362  322 213      LST
        031364  032 216      JMP S8A
S14B    031366  331 105      LIST1
        031370  322 213      LST
        031372  032 216      JMP S8A
```

IL PROGRAM

```
TAG     ADDRESS I1  I2  I3   MNEMONIC

        032000  326 053      INIT
        032002  322 360      NLINE
LNECLT  032004  320 070      GETLINE
        032006  322 360      NLINE
```

```
TAG     ADDRESS I1  I2  I3   MNEMONIC

        032010  320 265      TSTL
        032012  032 022      JMP DIR
        032014  320 360      INSRT
        032016  032 004      JMP LNECLT
XEQ     032020  326 131      XINIT
DIR     032022  232 041      TST S1 'LET'
        032024  114 105 324
        032027  133 310      CALL AVTEST
        032031  132 340      CALL EXPR1
        032033  324 147      STORE
        032035  322 304      DONE
        032037  322 375      NXT
S1      032041  232 074      TST S3 'GO'
        032043  107 317
        032045  232 057      TST S2 'TO'
        032047  124 317
        032051  132 343      CALL EXPR
        032053  322 304      DONE
        032055  323 224      XFER
S2      032057  232 275      TST S14 'SUB'
        032061  123 125 302
        032064  132 343      CALL EXPR
        032066  324 100      DONEX
        032070  326 027      SAVE
        032072  323 224      XFER
S3      032074  232 112      TST S3A 'IF'
        032076  111 306
        032100  132 343      CALL EXPR
        032102  133 114      CALL RELOP
        032104  132 343      CALL EXPR
        032106  325 151      CMPR
        032110  032 022      JMP DIR
S3A     032112  233 326      TST S4A 'FOR'
        032114  106 117 322
        032117  323 324      TSTV
        032121  326 363      ERR16
        032123  132 340      CALL EXPR
        032125  324 147      STORE
        032127  226 363      TST ERR
        032131  124 317
        032133  327 305      FOR
        032135  132 343      CALL EXPR
        032137  322 304      DONE
        032141  322 375      NXT
#
# LOC 032143-032147 IS UNUSED.
#
S4      032150  232 226      TST S9 'PR'
        032152  120 322
S5      032154  231 322      TST S8C '"'
        032156  242
        032157  322 322      PRS
S6      032161  232 173      TST S7A ','
        032163  254
        032164  322 342      SPCZONE
S7B     032166  232 332      TST S5A 'CR'
        032170  215
        032171  322 375      NXT
S7A     032173  232 202      TST S7 ';'
        032175  273
        032176  326 044      SPCONE
        032200  032 166      JMP S7B
S7      032202  322 360      NLINE
        032204  322 304      DONE
        032206  322 375      NXT
S8      032210  132 343      CALL EXPR
        032212  323 125      PRN
        032214  032 161      JMP S6
S8A     032216  322 304      DONE
        032220  322 375      NXT
#
# LOC 032222-032225 IS UNUSED.
#
S9      032226  232 251      TST S12 'IN'
        032230  111 316
S10     032232  133 310      CALL AVTEST
        032234  323 241      INNUM
        032236  324 147      STORE
        032240  232 245      TST S11 ','
        032242  254
        032243  032 232      JMP S10
S11     032245  322 304      DONE
        032247  322 375      NXT
S12     032251  232 264      TST S13 'RET'
        032253  122 105 324
        032256  326 036      DONE
        032260  322 304      RSTR
        032262  322 375      NXT
S13     032264  233 200      TST S14 'END'
        032266  105 116 304
```

```
TAG      ADDRESS I1  I2  I3   MNEMONIC

         032271  322 360      NLINE
         032273  323 011      FIN
S18      032275  232 306      TST S15 'LST'
         032277  114 123 324
         032302  031 340      LST
         032304  322 375      JMP S8A
S15      032306  232 317      TST S16 'RUN'
         032310  122 125 316
         032313  322 304      DONE
         032315  032 020      JMP XEQ
S16      032317  233 101      TST S17A 'NEW'
         032321  116 105 327
         032324  322 304      DONE
         032326  032 000      JMP STRT
S17      032330  326 347      ERR12
S5A      032332  232 154      TST S5A '$'
         032334  244
         032335  323 034      NXTX
         032337  000          NOP
EXPR1    032340  232 343      TST EXPR '+'
         032342  275
EXPR     032343  232 354      TST EØ '-'
         032345  255
         032346  133 003      CALL TERM
         032350  325 133      NEG
         032352  032 361      JMP E1
EØ       032354  232 357      TST E3 ' '
         032356  253
         032357  133 003      CALL TERM
E1       032361  232 372      TST E2 '+'
         032363  253
         032364  133 003      CALL TERM
         032366  324 200      ADD
         032370  032 361      JMP E1
E2       032372  233 055      TST E4 '-'
         032374  255
         032375  133 003      CALL TERM
         032377  324 216      SUB
         033001  032 361      JMP E1
TERM     033003  133 027      CALL FACT
TØ       033005  233 016      TST T1 '*'
         033007  252
         033010  133 027      CALL FACT
         033012  324 240      MULT
         033014  033 005      JMP TØ
T1       033016  233 055      TST T2 '/'
         033020  257
         033021  133 027      CALL FACT
         033023  324 362      DIV
         033025  033 005      JMP TØ
FACT     033027  330 032      TSTF
         033031  033 035      JMP F4
         033033  031 300      JMP FN
F4       033035  327 214      TSTA
         033037  033 047      JMP FØ
         033041  133 254      CALL ARRAY
         033043  324 133      IND
         033045  322 300      RTN
FØ       033047  323 324      TSTV
         033051  033 057      JMP F1
         033053  324 133      IND
T2:E4    033055  322 300      RTN
F1       033057  323 351      TSTN
         033061  033 065      JMP F2
         033063  322 300      RTN
F2       033065  233 077      TST F3 '('
         033067  250
         033070  132 343      CALL EXPR
         033072  233 077      TST F3 ')'
         033074  251
         033075  322 300      RTN
F3       033077  326 352      ERR13
S17A     033101  232 330      TST S17 'SZE'
         033103  123 132 305
         033106  331 000      SIZE
         033110  032 216      JMP S8A
         033112  000 000      NOP'S
RELOP    033114  233 123      TST RØ '='
         033116  275
         033117  325 326      LITO
         033121  322 300      RTN
RØ       033123  233 150      TST R4 '<'
         033125  274
         033126  233 135      TST R1 '='
         033130  275
         033131  325 334      LIT2
         033133  322 300      RTN
R1       033135  233 144      TST R3 '>'
         033137  276

TAG      ADDRESS I1  I2  I3   MNEMONIC

         033140  325 337      LIT3
         033142  322 300      RTN
R3       033144  325 331      LIT1
         033146  322 300      RTN
R4       033150  232 330      TST S17 '>'
         033152  276
         033153  233 162      TST R5 '='
         033155  275
         033156  325 345      LIT5
         033160  322 300      RTN
R5       033162  233 171      TST R6 '<'
         033164  274
         033165  325 337      LIT3
         033167  322 300      RTN
R6       033171  325 342      LIT4
         033173  322 300      RTN
         033175  000 000 000  NOP'S
S14      033200  232 275      TST S18 'DIM'
         033202  104 111 315
ZØ       033205  323 324      TSTV
         033207  326 352      ERR13
         033211  233 077      TST F3    '('
         033213  250
         033214  132 343      CALL EXPR
         033216  233 241      TST Z1 ','
         033220  254
         033221  132 343      CALL EXPR
         033223  233 077      TST F3    ')'
         033225  251
         033226  327 032      DIM2
Z3       033230  233 235      TST Z2 ','
         033232  254
         033233  033 205      JMP ZØ
Z2       033235  322 304      DONE
         033237  322 375      NXT
Z1       033241  233 077      TST F3    ')'
         033243  251
         033244  327 066      DIM1
         033246  033 230      JMP Z3
         033250  000 000 000  NOP'S
         033253  000          NOP
ARRAY    033254  233 077      TST F3    '('
         033256  250
         033257  132 343      CALL EXPR
         033261  233 275      TST XØ ','
         033263  254
         033264  132 343      CALL EXPR
         033266  233 077      TST F3    ')'
         033270  251
         033271  327 166      ARRAY2
         033273  322 300      RTN
XØ       033275  233 077      TST F3    ')'
         033277  251
         033300  327 146      ARRAY1
         033302  322 300      RTN
LSTSTRT  033304  000 034
LSTEND   033306  036 037
AVTEST   033310  327 214      TSTA
         033312  033 320      JMP VØ
         033314  133 254      CALL ARRAY
         033316  322 300      RTN
VØ       033320  323 324      TSTV
         033322  326 344      ERR11
         033324  322 300      RTN
S4A      033326  232 150      TST S4 'NXT'
         033330  116 130 324
         033333  323 324      TSTV
         033335  326 352      ERR14
         033337  327 324      NEXT
         033341  324 147      STORE
         033343  322 304      DONE
         033345  322 375      NXT
         033347  000          NOP
CURLBL   033350  000 000
PRGSTRT  033352  000 034
PRGEND   033354  036 037
COUNT    033356  001
CASE     033357  040
ZONE     033360  004
AELVL    033361  100 030
INDX     033363  001
SBRLVL   033364  164 024
ASTRT    033366  377 057
ATEMP    033370  000 000
SEED1    033372  150
SEED2    033373  205
SEED3    033374  341
SEED4    033375  336
MEND     033376  377 057
```

[LAST INSTANT POSTSCRIPT TO WHAT FOLLOWS] JUST GOT THE WORD THAT THEY WILL SELL VOICE SYNTHESIZER KITS FOR $1000 IF THEY FEEL THEY CAN MARKET AT LEAST 50! DETAILS NEXT ISSUE. LET 'EM KNOW WHAT THEIR POTENTIAL MARKET IS!!!

JCW, JR.

jim day's DAZE

[reprinted from **PCC** *Vol. 4, No. 5]*

COMPUTERS THAT TALK

Wouldn't it be nice if your computer could speak to you in English, French, German, or Esperanto like the computer on the starship Enterprise? Then it could say things like, "Wake up, sir" or "Get with it, turkey" (depending on what kind of mood it was in) or maybe, "The time is six o'clock, the temperature is 46 degrees, and tomorrow is your wife's birthday." Most people have probably assumed that some day, perhaps by the year 2000, talking computers will be a reality instead of simply science fiction. Well, hang onto your prognostications, people, because that day is today!

In recent years many people have been working on voice output devices for computers. Some of these devices have been electro-mechanical analogs of the human vocal tract, similar in principle to the Voder exhibited at the New York World's Fair in 1939. Others have used electronic waveform generators to synthesize human speech sounds. Of these, the Votrax synthesizer can truly be said to represent a significant breakthrough with respect to voice quality, ease of programming, and cost.

Smaller than a breadbox and priced at about $3500 for the basic unit, Votrax is produced by the Vocal Interface Division of the Federal Screw Works (500 Stephenson Highway, Troy MI 48084; (313) 588-2050). Any computer capable of outputting a string of ASCII code to a terminal can be used to control Votrax. As an output device, Votrax can be used alone or in conjunction with an ordinary TTY, using embedded ASCII control codes and simple logic to switch voice strings to Votrax, and print strings to the TTY, TVT, or other conventional terminal.

Programming Votrax is a snap. Using BASIC, FORTRAN, APL, PL/1, or just about any other programming language, it's easy to convert ordinary English (or other natural language) into voice strings for Votrax. The best quality of vocal output is obtained by using a dictionary lookup technique to substitute a string of phoneme codes for each English word. Votrax responds to ASCII codes for 63 different phonemes (basic speech sounds) and each phoneme can have one of four levels of inflection.

If perfect voice quality is not essential and random-access file space is not available for a large dictionary, an algorithm can be used to convert English words to phoneme codes. Such an algorithm, developed by Bell Telephone Laboratories, is said to work almost as well as dictionary lookup. An unpronounceable string such as "PDP-8" can be spelled out phonetically as though written "pee dee pee dash ate," and the number 10.6 can be rendered as "ten point six" by means of a simple subroutine. Pauses can be inserted automatically in response to punctuation and paragraphing.

Maybe you are wondering whether anyone has actually used Votrax and, if so, how did they like it? The answer to both questions is yes. People are using Votrax and they like it a lot. For example, the Coast Community College District in Costa Mesa, California, is using Votrax for computer-aided instruction and also in an on-line student information system. Votrax was chosen in preference to other audio response units not only because it is much less expensive but also because it is ideal for a wide range of applications, the size of its vocabulary is unlimited, and it functions well in a real-time environment. In the student information system application, Touch-Tone telephones are used as "terminals." Although this limits the user to numeric input, it would be hard to find a cheaper or more readily available I/O device. Several extensions to the district's present use of Votrax are being developed, such as a voice-output interface for their on-line budget system, allowing administrators to inquire about specific accounts and receive immediate vocal replies. David Clements, senior programmer/analyst for the district's student information system, reports that he is amazed at the results achieved with Votrax and believes that synthesized voice output will become a widely used medium in the near future.

Another application of Votrax is as an aid to blind programmers. In the Homer system, written in FORTRAN for a CDC 6500 at Michigan State University, Votrax is used to echo each line input from a conventional terminal. It is also used to deliver FORTRAN diagnostics and as a tool in the editing of source program files.

Operating in conjunction with an optical page reader, Votrax can be used to convert printed matter, such as books, magazines, and newspapers, into audible form. If desired, the output from Votrax can be tape recorded for distribution to the blind.

These are but a few of the uses to which voice output can be put, and it appears likely that voice output will soon become a familiar feature of many computer systems. Maybe yours will be one of them.

(Also see "Talking Calculator" in November 1975 PCC [Vol. 4, No. 3, p. 9].)

COMPUTERS THAT TALK — UPDATE

Jim Day had an article in the most recent issue of PCC discussing the use of a Votrax machine to allow a computer to synthesize speech [article is reprinted, herein]. In the article, he indicated that the machine, essentially a solid-state phoneme generator, was priced at about $3500 for a basic system ... a bit high for most hobbyists' budget. [Phonemes are the basic components that make up spoken words.]

Well, we just finished talking to the west coast rep for Votrax for about an hour and a half, and have some exciting possibilities to report!

Votrax is currently selling relatively few of their systems. It would be easy for the computer hobbyist community to *significantly* increase their sales (and, presumably, thereby drive the price per unit significantly downward). And, the rep didn't even know the hobbyist market existed; *he does now.*

First of all, the price that Jim quoted was for a turnkey system; one that includes two 25-pin interconnect boards, an 80-byte buffer for the incoming phoneme codes, an amplifier, and a power supply Such a configuration is usually expected and demanded by the commerical and industrial users. However, it's a different matter with computer hobbyists. Hobbyists are accustomed to using breadboarding, can supply their own buffering via their system's memory, invariably have the ability to input to a hi fi amp, and usually can find super-cheap power supplies.

Assuming this, all that one *really* needs to purchase are the four phoneme generator boards, and have access to the interface engineering specifications and schematics. These are available for under $2K in small quantities; $1800 @ in groups of ten, and $1600 @ in groups of fifty.

Would you rather have a $1600 hardcopy device or *the ability to generate English speech, including inflection?* Since the Votrax equipment is based on phoneme generation, the vocabulary is essentially unlimited. Further, since the generators are entirely electronic, the equipment has much greater reliability than electro-mechanical equipment. Also, the Votrax equipment and circuitry has been in the field for about half a decade, now, and is thoroughly debugged.

If you would like for Votrax equipment to become available to the hobbyist community:

(1) Write to John McDaniel, Votrax, 4340 Campus Dr., No. 212 Newport Beach, Ca. 92660; tell him that you would like for your computer to be able to *talk* to you, and indicate how much you would be willing to pay for that facility. Give him correspondence to support him when he approaches Votrax management. Make him and them aware of their untapped potential market for stripped-down systems in the hobbyist community.

(2) Tell the owners of your local computer store about Votrax and encourage them to contact Mr. McDaniel.

A BIT OF BLUE SKYING

Bob, February 19, 1976

By all means keep up the Calisthenics & Orthodontia. But I suspect that as *Tiny BASIC* matures it will acquire a full set of canines, bicuspids, and molars. As the price of main memory continues to drop, the need for a minimal BASIC will assume less importance and the emphasis will shift to better performance and convenience. Still, IL is a good tool for those who may want to experiment with variants of BASIC or some other language. As unlikely as it may seem, I think that by 1980 most hobbyists will be using a subset of PL/1. I also preduct that the 1980 hobbyist will own a computer system the size of a breadbox and comprising a 16-bit CPU, 256K bytes of main memory, 8M byte floppy disc, dual tape cassettes, full ASCII keyboard, CRT display, modem, and non-impact printer (all in one box). The whole thing will sell (assembled) for $695 at Sears and will have the computing power of an IBM 370. Last, but not least, the CPU chip will be designed expressly for the hobbyist, not for some pedestrian application such as traffic signal control.

Jim Day 17042 Gunther St
Granada Hills CA 91344

Dear Bob, February 4, 1976

Thank you for your note and interest. Our system is growing by small leaps and bounds. We have an Altair 8800 with the Processor Tech. mother board. We also have the following items:

Qty	Description	
1	VLCT (octal loader)	Altair
1	PIO	Altair
1	256 byte static RAM board	Altair
2	4K RAM boards	Godbout
3	4K RAM boards	Proc. Tech.
1	3 P + S	Proc. Tech.
1	wire wrap prototype board	TCH
1	cassette interface	TCH
1	VDM	Proc. Tech.
1	Real time clock and VI	IMS
1	ASR-33 (10 cps)	Teletype
1	Silent 700 (30 cps)	TI
1	2K ROM board	Proc. Tech.

We are building a version of the TCH graphics interactive display with direct Altair plus in boards (double-sided).

We are also ordering the Processor Tech. dual cassette drive, controller and PTCOS.

We have several interactive editors, assemblers, monitors, and cross assemblers. We are currently experimenting with minimal editors and assemblers and have a strong desire to put together a micro-BASIC (Tiny BASIC). The editor package looks like it will be around 510-512 bytes and the same for a "mini-assembler." We are also looking for 4K, 8K, and 12K BASICs which are public.

We are hoping to eventually acquire a TV Dazzler and a floppy disc to extend our system. Future desires also include the IMS shared processor/memory and an additional CPU board in addition to 12K-16K more low power status RAM memory. Who knows what else the future has in store?

We are strongly interested in developing software (for the Altair and other micro-processors) which can be used for instruction and instructional support in the school media center.

Our research interests vary considerably here so we also will be running some basic human learning experiments under processor control. We have been involved in research in CAI and computer-managed instruction for about 9 years here. We have PLANIT, COURSEWRITER, PICLS, PLATO (TUTOR), and BASIC available and a wide range of instructional programs for these languages.

Franz Frederick 112 Education Blg
Associate Professor Purdue University
W. Lafayette IN 47907

Franz, We would be *very* interested in publishing the source code and documentation (user *and* implementation details) for the "tiny" editors and assemblers you are implementing. Any chance of your forwarding copies, once they are up and running? --JCW, Jr

TBX MODS FOR A SWTP TVT-2

Dear Dennis and all TB people,

First of all, thanks to Dick Whipple and John Arnold for a great job they have done on TB, and for making their program available. Many hobbyists, including myself, don't have the skill or time to write anything as complex as an interpreter.

TBX is working and programming is now FUN. It took about six hours to put TBX on a cassette. Loading TB from TP (tiny print) is a severe strain on the eyes.

A listing of the I/O routines for my Altair/TVT-2 system is enclosed. An instruction is encluded in the Entry Routine to turn on the TVT cursor and initiate a Home Up/Erase Frame. In the Input routine the code for ESC should be 033; otherwise a rubout (backspace in TBX) will give a system restart. The basic Altair executes a RST 7 if the keyboard is tied directly to the interrupt bus. I had to change the instruction at 000070 to 311. No harmful effects so far.

```
000  000 076 004     MVI A     Turn on cursor on
     002 323 002     OUT        TVT & initiate
     004 061 377 000 LXI SP     Home up/Erase fr.
     007 303 254 021 JMP       TBX entry point

     020 076 012     MVI A
     022 357         RST       Output LF
     023 076 015     MVI A
     025 357         RST       Output CR
     026 311         RET

     030 373         EI
     031 166         HLT       Wait for KBD entry
     032 333 001     IN        Input KBD charctr
     034 346 177     ANI       Mask parity bit
     036 376 033     CPI       "ESC"
     040 312 000 000 JZ        System entry
     043 357         RST       Echo character
     044 311         RET

     050 365         PUSH PSW  Save registers & flags
     051 323 001     OUT       Output character to TVT
     053 333 002     IN        Wait for "data ac-
     055 037         RAR        cepted" signal
     056 322 053 000 JNC        from TVT
     061 361         POP PSW   Restore register &
     062 311         RET        flags

     070 311         RET       Keyboard interrupt
```

PORT ASSIGNMENTS:

```
IN  001  ASCII keyboard input
OUT 001  Character output to TVT
IN  002  "Data accepted" from TVT
OUT 002  Cursor control to TVT
```

TINY BASIC AVAILABLE FOR THE 6800

A version of Tiny BASIC has been developed for the Motorola and AMI 6800. A tape and instruction manual for it are available for $5 from:

Tim Pittman
Box 23189
San Jose CA 95153
(408) 578-4944

We understand that the source code will not be made available, however, we expect that Tom will back his "product" . . . and the price is right.

We would be interested in hearing of the joys and/or woes incurred by those who purchase Tom's Tiny BASIC.

BYTE SWAP

We are experimenting with offering a "Want Ad" section. We will continue to do it as long as we can afford it (in terms of staff time and printing costs). Note: the charge for running an ad will undoubtedly increase as our circulation (and printing costs) increases.

Please follow these instructions in submitting ads. Ads received in other than this form cannot be accepted, and will be returned to the sender.

1. ***Type* the ad, with a blank space between each line, in lines no more than 50 character positions in length.**
2. **Include at least your name and address as part of the ad. "Blind" ads will not be accepted.**
3. **Compute the charge on the basis of $1 per line or partial line, per issue.**
4. **Forward the typed copy and a check or money order payable to "PCC," to: DDJ Byte Swap, PCC, Box 310, Menlo Park CA 94025. Do *not* send cash. Your cancelled check is your receipt. Payment *must* accompany the ad.**

SAVE MY MARRIAGE! *Buy my new assembled IMSAI 8080, loaded 22 slot mother board, 8k Ram, regular price, $1835.00. Will sell to highest bidder above $1700.00. Also, IMSAI 8080 kit, still in box, large mother board, regular price $578.00. Will sell to highest bidder above $547.00. Send bids to: Eric Stewart, 664 Via Alamo, San Lorenzo CA 94580.*

I am looking forward to an annotated source code listing for TBS; like to do some tinkering. Floating point and math functions would also be nice to have. Dr Suding's scientific calculator interface looks good. However, it's only available through MiniMicroMart and doing business with them has been a frustrating experience.

When deciding on the future of the newsletter keep in mind that hardware is available and getting cheaper. Software has been a big problem and probably will be for some time to come (unless you can afford to pay for it). The newsletter is a step in the right direction to solve this problem. Please don't stop after three issues.

Adolph Stumpf
5639-A Ute
Glendale AZ 85307

DR. DOBB'S JOURNAL OF COMPUTER CALISTHENTICS AND ORTHODONTIA is published ten time per year, monthly except in July and December.

U.S. Subscriptions:

☐ $1.50 for a single copy
☐ $3.00 for the first three issues
☐ $10.00 per year (10 issues/year)

For foreign subscriptions:

☐ *add* $4.00 per year for surface mail, or
☐ *add* $12.00 per year for air mail

Payment must accompany the subscription. We do not invoice for subscriptions or single orders.

Necessary Information:

Name (last name first) ______________________________

Mailing Address ______________________________

____

City ______________ State ______________ Zip Code ______________

☐ yes ☐ no: This information may be published in directories and lists of individuals interested in computers in non-commercial environments.

Optional Information:

Equipment that you have or are planning on purchasing, immediately:

Make & model ______________ Manufacturer ______________

CPU model ______________ CPU Manufacturer ______________

I/O Devices ______________________________

Mass storage peripherals ______________________________

Primary areas of interest concerning non-commercial and home computers:

____

____

Questions: What would you like to see published in **DR. DOBB'S JOURNAL**? It will help guide us if you will rate these, 1 to 10 (1 – minimally desire; 10 – super-eager to see) or 0 (would prefer we not waste space publishing it).

_____ Schematics and acticles from all of the computer club newsletters
_____ Short news articles directly related to home computers
_____ Short news articles concerning computers in general, particularly their social implications
_____ Indices to all articles in all other computer hobby publications
_____ Indices to selected articles from other computer, electronic, and trade publications
_____ Letters having technical, critical, or entertaining content
_____ Classified ads (as opposed to display advertising)
_____ Suggestions and "blue skying" about what can be done with home computers in the foreseeable future.

Directories of:

_____ Users of home computers and their equipment
_____ Computer stores and distributers
_____ Manufacturers of computer kits
_____ Computer clubs
_____ Sources of used equipment
_____ Microprocessor and minicomputer manufacturers

Source code listings and documentation: For which microprocessors? ______________ ______________

_____ Nearly full-sized (much less can be published)
_____ Reduced as in recent issues (more difficult to read, but more info included in each issue)

What kind of software would you like to see developed and placed in the public domain?

Importance Rating	Software Description
______________	______________________________
______________	______________________________
______________	______________________________

What else would you like to see us publish? *Please use another page or ten, if you need them.*

DR DOBB'S JOURNAL OF
COMPUTER CALISTHENICS & ORTHODONTIA
PCC
Box 310
Menlo Park CA 94025

Place
13-cent
stamp
here

DR DOBB'S JOURNAL OF
COMPUTER CALISTHENICS & ORTHODONTIA
PCC
BOX 310
MENLO PARK CA 94025

To use this as a "self-mailer":

1. Fold it so *this* third covers the *top* third.
2. Place the proper postage, above.
3. If you are subscribing, insert your check so that it crosses a fold.
4. Staple this closed *with a single staple*, making sure that the staple pierces the check.

(Better still, stick all of this in your own envelope and mail it to us.)